



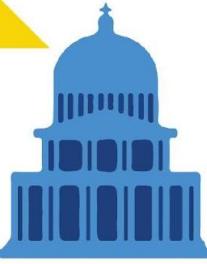
**KubeCon**

**CloudNativeCon**

---

**Europe 2025**

---





KubeCon



CloudNativeCon

Europe 2025

# Extending Kubernetes Resource Model (KRM) Beyond Kubernetes workloads

Mangirdas Judeikis & Nabarun Pal



# Introduction



Mangirdas (MJ) Judeikis  
 [@mangirdas.bsky.social](https://@mangirdas.bsky.social)  
 [@mangirdas](https://@mangirdas)



Nabarun Pal  
 [@nabarun.dev](https://@nabarun.dev)  
 [@theonlynabarun](https://@theonlynabarun)

# Agenda

**Part 1:** What is Kubernetes Resource Model (KRM) and why it got such success.

# Agenda

**Part 1:** What is Kubernetes Resource Model (KRM) and why it got such success.

**Part 2:** We will explore how we, kcp maintainers, push for the next KRM evolution level.

# Agenda

**Part 1:** What is Kubernetes Resource Model (KRM) and why it got such success.

**Part 2:** We will explore how we, kcp maintainers, push for the next KRM evolution level.

**Part 3:** For the audience, we will show how to create custom APIs without using CRDs but still using Kubernetes API machinery or kcp.

# Why are APIs hard?

Imagine managing a complex hybrid cloud environment with dozens of different APIs

# Why are APIs hard?

Imagine managing a complex hybrid cloud environment with dozens of different APIs

- Consistency

# Why are APIs hard?

Imagine managing a complex hybrid cloud environment with dozens of different APIs

- Consistency
- Interoperability

# Why are APIs hard?

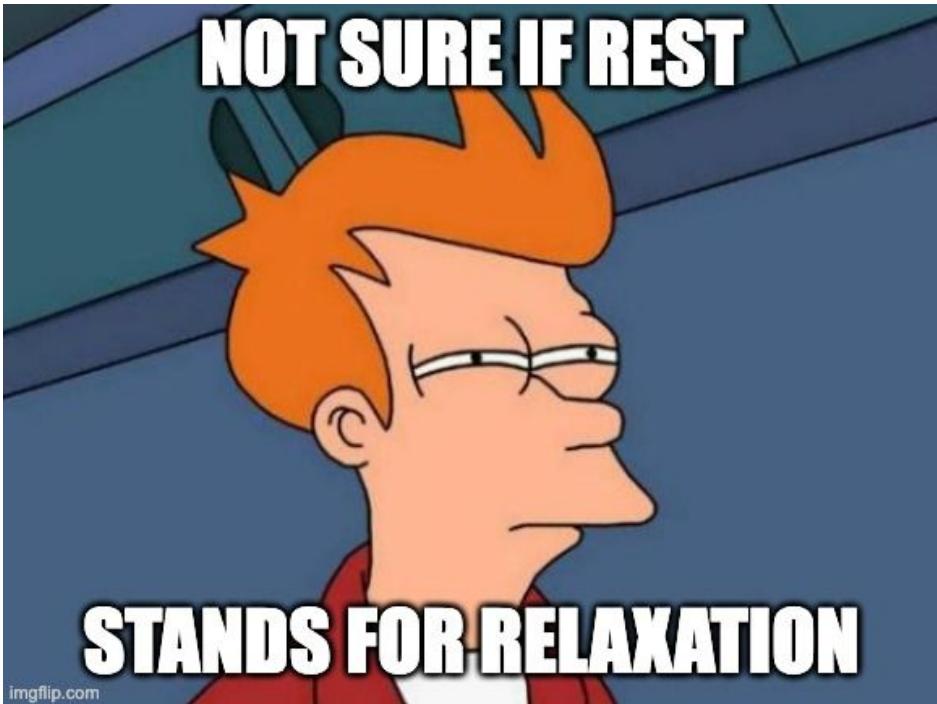
Imagine managing a complex hybrid cloud environment with dozens of different APIs

- Consistency
- Interoperability
- Ecosystem

# Why are APIs hard?

Imagine managing a complex hybrid cloud environment with dozens of different APIs

- Consistency
- Interoperability
- Ecosystem



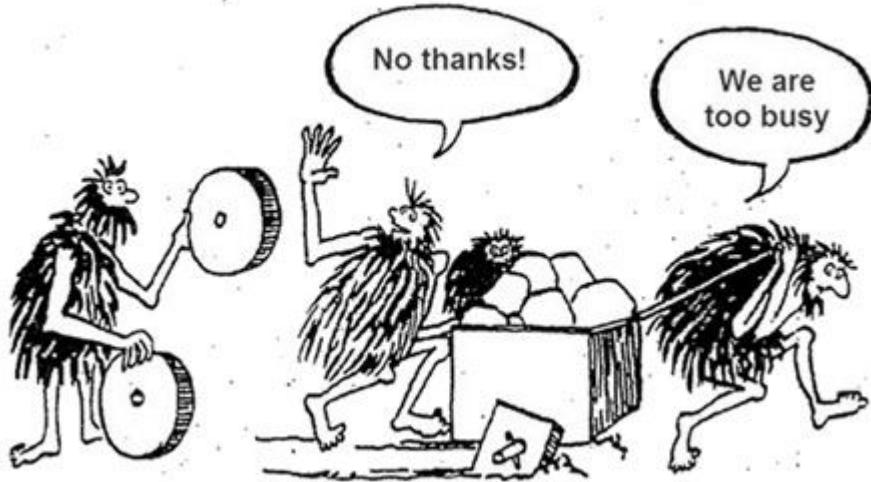
# Why are APIs hard?

# Why are APIs hard?

NIH - “Not Invented Here” syndrome. We like ***re-inventing!***

# Why are APIs hard?

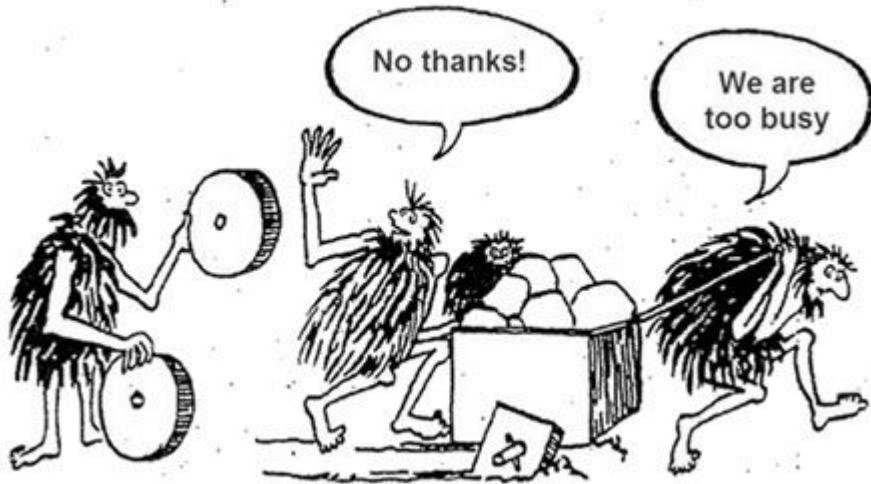
NIH - “Not Invented Here” syndrome. We like ***re-inventing!***



# Why are APIs hard?

NIH - “Not Invented Here” syndrome. We like ***re-inventing!***

- Know your area of business
- Know where is your value proposition
- Stop inventing things outside those areas
- Adoption & mental model





KubeCon



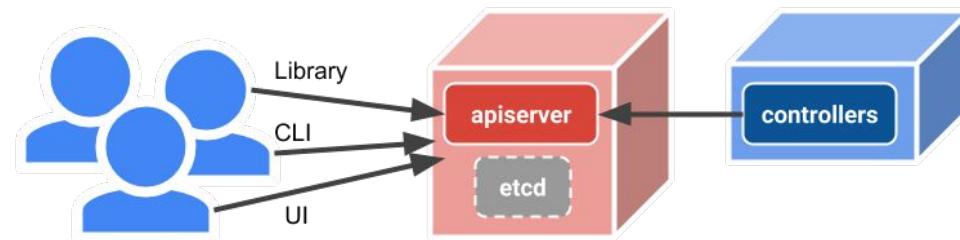
CloudNativeCon

Europe 2025

---

But the question is **should we re-invent?**

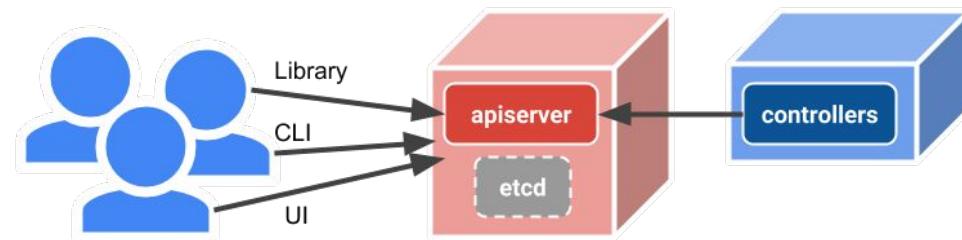
# Kubernetes Resource Model (KRM)



# Kubernetes Resource Model (KRM)



Basically kubernetes API.



# Declarativeness

Declarative models rely on preconfigured capabilities in the language  
**without explicit case-by-case instructions on what steps to take to accomplish a particular task**



# Structural Consistency



```
apiVersion: shop.k8s.io/v1
kind: Pizza
metadata:
  name: hawaiian
  namespace: default
spec:
  ingredients:
    - pineapples
    - mushrooms
status:
  phase: baking
  conditions:
    - lastTransitionTime: "2025-01-16T15:41:43Z"
      status: "True"
    type: OrderAccepted
```



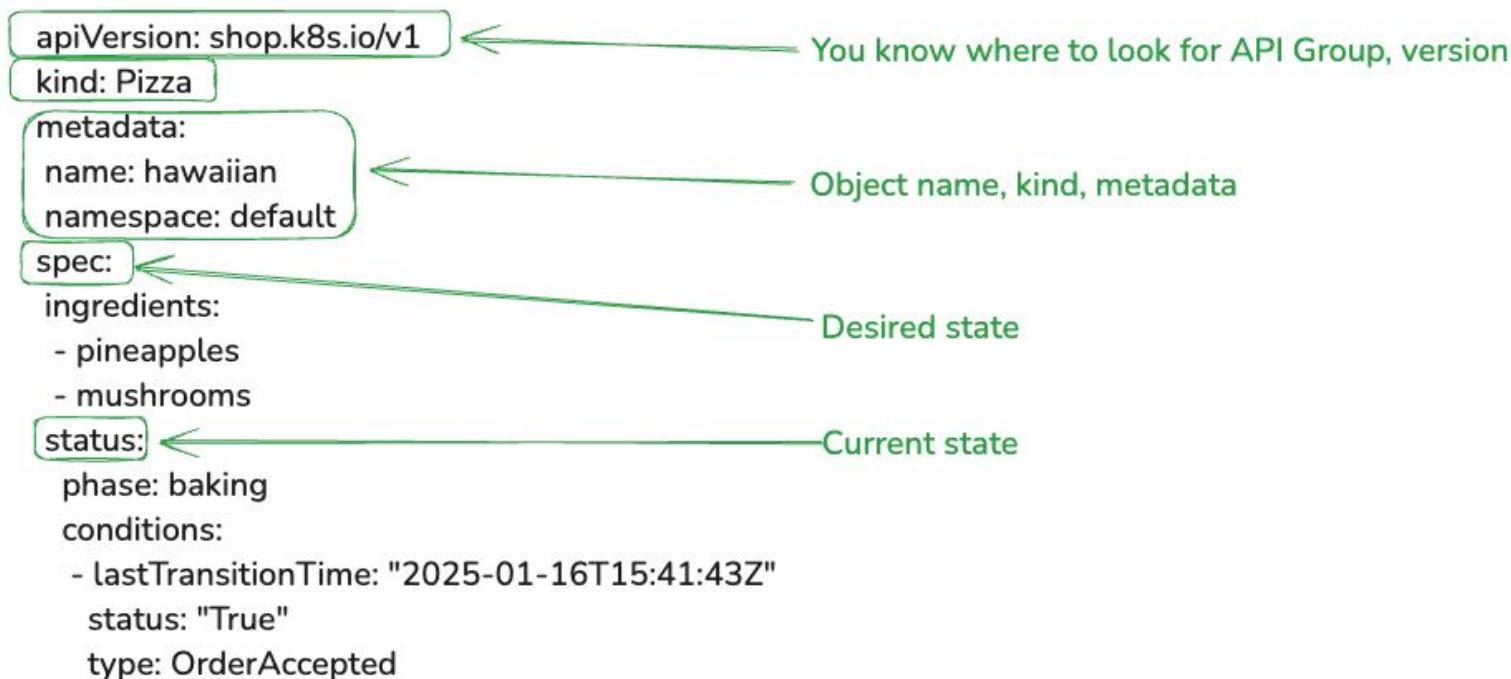
KubeCon



CloudNativeCon

Europe 2025

# Structural Consistency





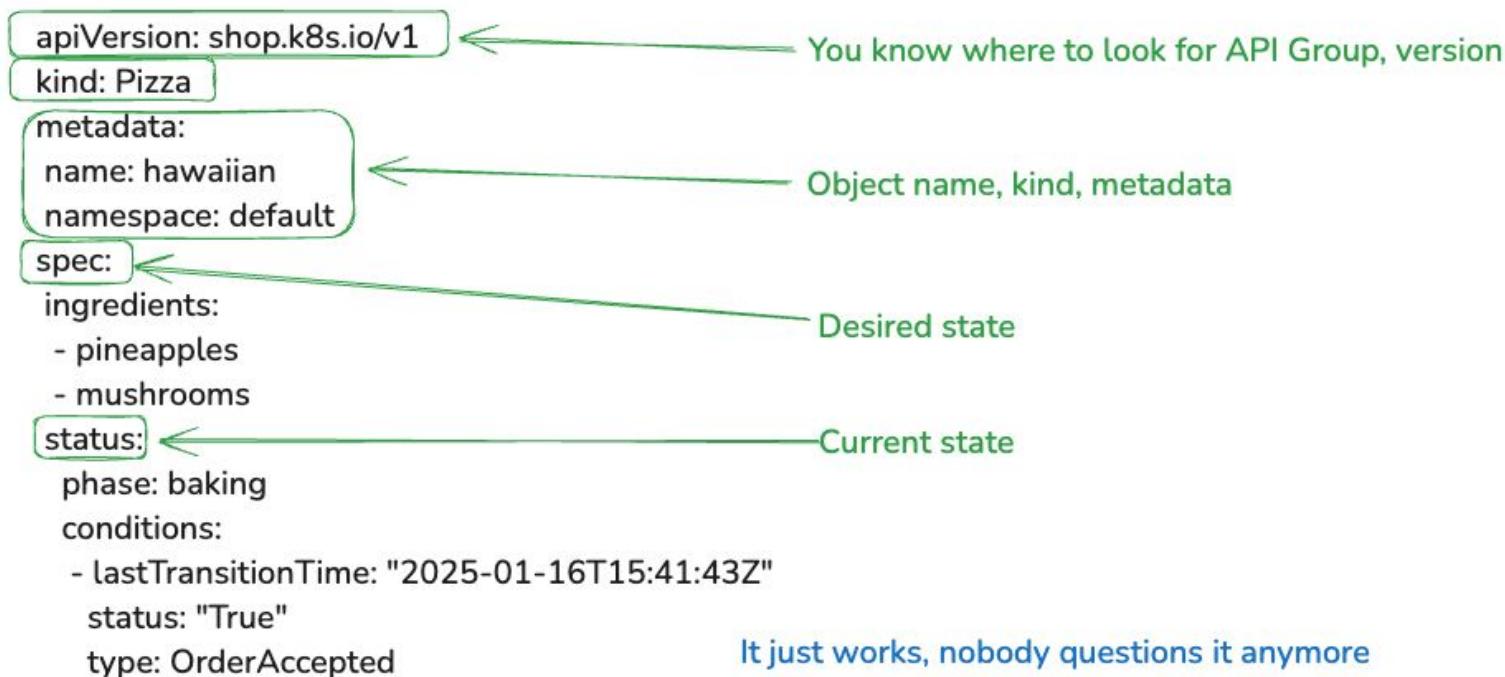
KubeCon



CloudNativeCon

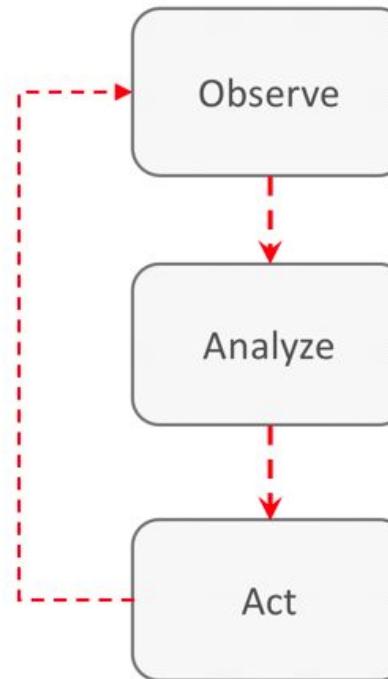
Europe 2025

# Structural Consistency



# Reconciliation Loop

```
apiVersion: shop.k8s.io/v1
kind: Pizza
metadata:
  name: hawaiian
  namespace: default
spec:
  ingredients:
    - pineapples
    - mushrooms
status:
  phase: baking
  conditions:
    - lastTransitionTime: "2025-01-16T15:41:43Z"
      status: "True"
      type: OrderAccepted
```



Easy Right?

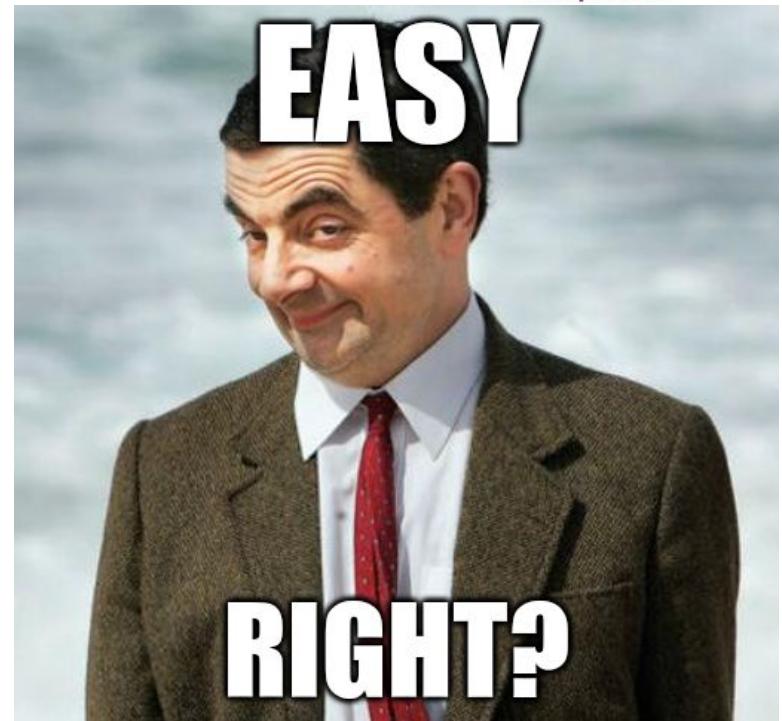


KubeCon



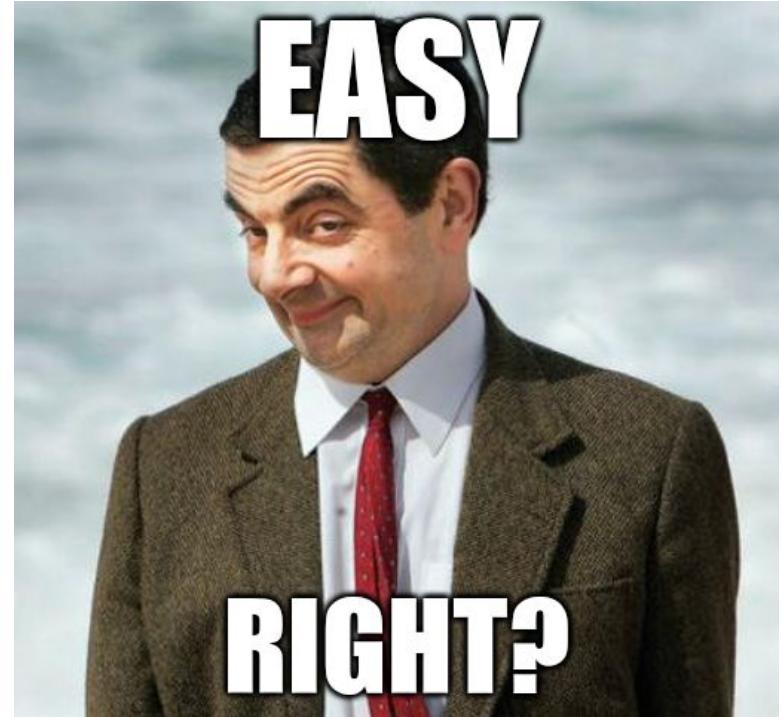
CloudNativeCon

Europe 2025



## Easy Right?

Having strict standard does not mean you  
can't do stupid things!



KubeCon



CloudNativeCon

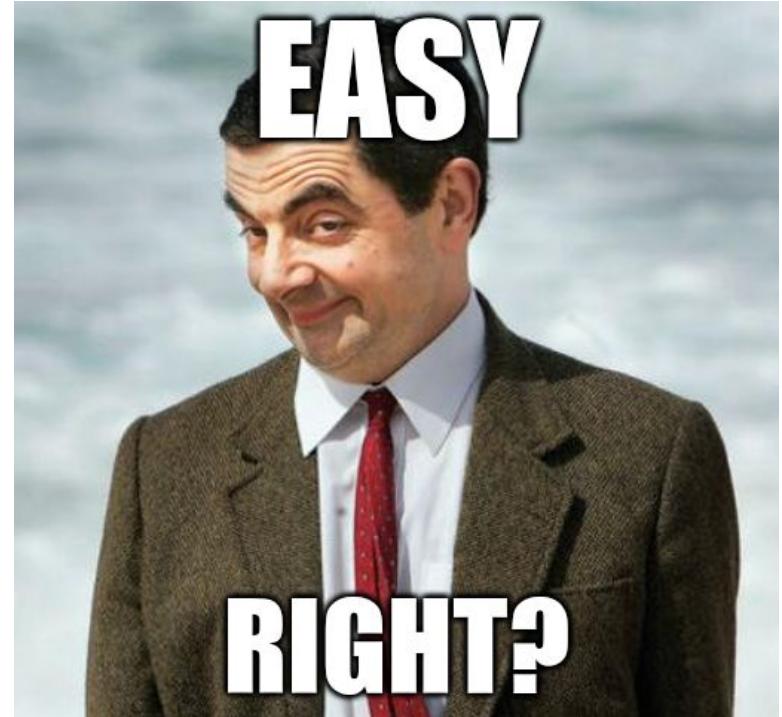
Europe 2025

# Easy Right?

Having strict standard does not mean you can't do stupid things!

“Two things are infinite: the universe and human stupidity; and I’m not sure about the universe.”

— Albert Einstein



KubeCon



CloudNativeCon

Europe 2025



KubeCon



CloudNativeCon

Europe 2025

# Few Things to Avoid



# Mixing Business Logic into the CRD's Design



```
apiVersion: v1
kind: Service
metadata:
  creationTimestamp: "2025-01-16T15:41:22Z"
  labels:
    component: apiserver
    provider: kubernetes
    name: kubernetes
    namespace: default
    resourceVersion: "198"
    uid: 76d1f21d-4ba1-4913-8343-9394f7666656
spec:
  clusterIP: 10.96.0.1
  clusterIPs:
  - 10.96.0.1
  internalTrafficPolicy: Cluster
  ipFamilies:
  - IPv4
  ipFamilyPolicy: SingleStack
  ports:
  - name: https
    port: 443
    protocol: TCP
    targetPort: 6443
  sessionAffinity: None
  type: ClusterIP
status:
  loadBalancer: {}
```

## Spec vs. Status Misuse

# Mixing Business Logic into the CRD's Design

```
apiVersion: v1
kind: Service
metadata:
  creationTimestamp: "2025-01-16T15:41:22Z"
  labels:
    component: apiserver
    provider: kubernetes
  name: kubernetes
  namespace: default
  resourceVersion: "198"
  uid: 76d1f21d-4ba1-4913-8343-9394f7666656
spec:
  clusterIP: 10.96.0.1
  clusterIPs:
  - 10.96.0.1
  internalTrafficPolicy: Cluster
  ipFamilies:
  - IPv4
  ipFamilyPolicy: SingleStack
  ports:
  - name: https
    port: 443
    protocol: TCP
    targetPort: 6443
  sessionAffinity: None
  type: ClusterIP
status:
  loadBalancer: {}
```



## Spec vs. Status Misuse

Embedding **too much logic or assumptions directly** into the CRD's spec instead of relegating dynamic state information to the status field.

# Mixing Business Logic into the CRD's Design

```
apiVersion: v1
kind: Service
metadata:
  creationTimestamp: "2025-01-16T15:41:22Z"
  labels:
    component: apiserver
    provider: kubernetes
    name: kubernetes
    namespace: default
    resourceVersion: "198"
    uid: 76d1f21d-4ba1-4913-8343-9394f7666656
spec:
  clusterIP: 10.96.0.1
  clusterIPs:
  - 10.96.0.1
  internalTrafficPolicy: Cluster
  ipFamilies:
  - IPv4
  ipFamilyPolicy: SingleStack
  ports:
  - name: https
    port: 443
    protocol: TCP
    targetPort: 6443
    sessionAffinity: None
    type: ClusterIP
status:
  loadBalancer: {}
```



## Spec vs. Status Misuse

**Not clearly separating** the desired state (spec) from the observed state (status).

# Ignoring Established API Conventions and Best Practices



```
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: dbconnections.mycompany.io
spec:
  group: mycompany.io
  versions:
    - name: v1
      served: true
      storage: true
  scope: Namespaced
  names:
    plural: dbconnection # ✗ Bad: singular instead of plural
    singular: dbconnection
    kind: Dbconnection # ✗ Bad: inconsistent capitalization and lack of clarity
    shortNames:
      - dbc
```

# Ignoring Established API Conventions and Best Practices



```
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: sheep.mycompany.io
spec:
  group: mycompany.io
  versions:
    - name: v1
      served: true
      storage: true
  scope: Namespaced
  names:
    plural: sheep # ❌ Both singular and plural are "sheep" (flocks is wrong too)
    singular: sheep # ❌ Ambiguity: no clear distinction between one item and many
  kind: Sheep
  shortNames:
    - shp
```

# Insufficient Schema Validation and Lack of Structural Schema



- Not defining a comprehensive OpenAPI v3 schema for your CRD
- Omitting validations for required fields, value ranges, or types

```
// +kubebuilder:validation:XValidation:rule="has(self.__namespace__) || has(self.name)",message="at least one field must be set"
type ResourceSelector struct {
    // name of an object within a claimed group/resource.
    // It matches the metadata.name field of the underlying object.
    // If namespace is unset, all objects matching that name will be claimed.
    //
    // +optional
    // +kubebuilder:validation:Pattern="^([a-z0-9][a-z0-9_.]*?[a-z0-9]$"
    // +kubebuilder:validation:MaxLength=253
    // +kubebuilder:validation:MinLength=1
    Name string `json:"name,omitempty"`

    // namespace containing the named object. Matches metadata.namespace field.
    // If "name" is unset, all objects from the namespace are being claimed.
    //
    // +optional
    // +kubebuilder:validation:MinLength=1
    Namespace string `json:"namespace,omitempty"`}
```

You have a hammer in your tool-box, stop using a fork to nail these things!

# Insufficient Schema Validation and Lack of Structural Schema

- Not defining a comprehensive OpenAPI v3 schema for your CRD
- Omitting validations for required fields, value ranges, or types

```
// +kubebuilder:validation:XValidation:rule="has(self.__namespace__) || has(self.name)",message="at least one field must be set"
type ResourceSelector struct {
    // name of an object within a claimed group/resource.
    // It matches the metadata.name field of the underlying object.
    // If namespace is unset, all objects matching that name will be claimed.
    //
    // +optional
    // +kubebuilder:validation:Pattern="^([a-z0-9][a-z0-9_.]*?[a-z0-9]$"
    // +kubebuilder:validation:MaxLength=253
    // +kubebuilder:validation:MinLength=1
    Name string `json:"name,omitempty"`

    // namespace containing the named object. Matches metadata.namespace field.
    // If "name" is unset, all objects from the namespace are being claimed.
    //
    // +optional
    // +kubebuilder:validation:MinLength=1
    Namespace string `json:"namespace,omitempty"`
}
```

You have a hammer in your tool-box, stop using a fork to nail these things!

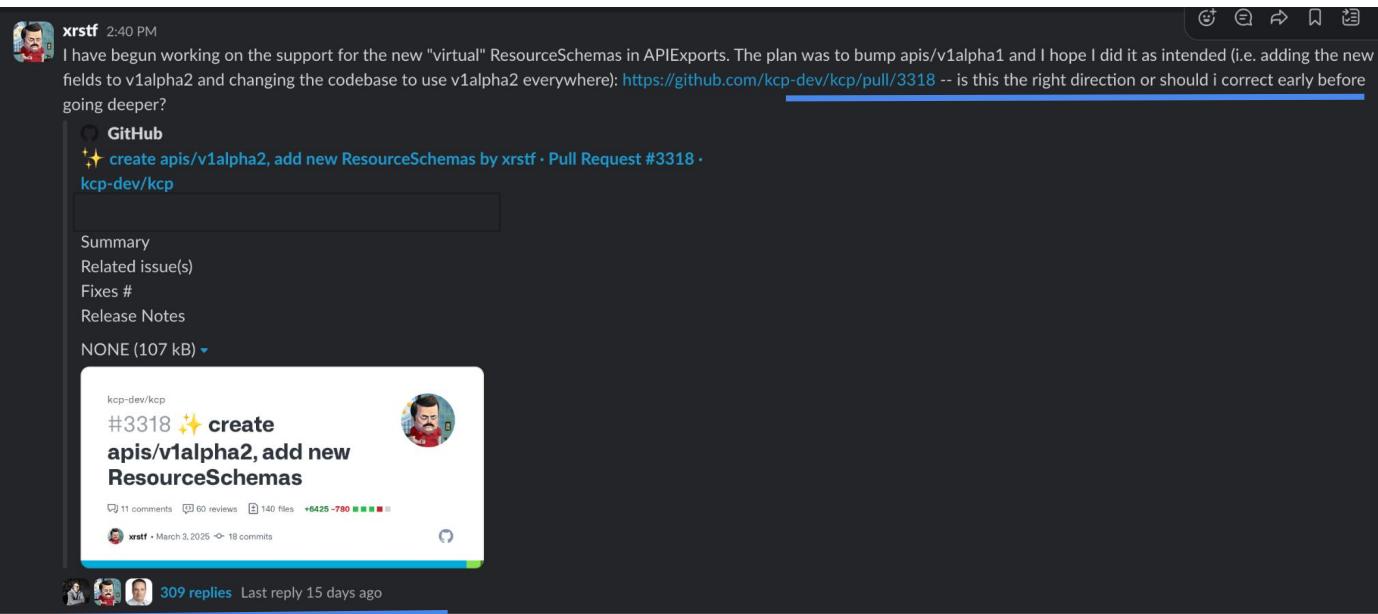


# Improper Versioning and Lack of Conversion Strategy



# Improper Versioning and Lack of Conversion Strategy

CRD versioning is hard! Spend some time creating best strategy for your use case, maybe even before you need it!



xrstf 2:40 PM  
I have begun working on the support for the new "virtual" ResourceSchemas in APIExports. The plan was to bump apis/v1alpha1 and I hope I did it as intended (i.e. adding the new fields to v1alpha2 and changing the codebase to use v1alpha2 everywhere): <https://github.com/kcp-dev/kcp/pull/3318> -- is this the right direction or should i correct early before going deeper?

GitHub  
create apis/v1alpha2, add new ResourceSchemas by xrstf · Pull Request #3318 · kcp-dev/kcp

Summary  
Related issue(s)  
Fixes #  
Release Notes  
NONE (107 kB) ▾

kcp-dev/kcp  
**#3318 ✨ create apis/v1alpha2, add new ResourceSchemas**

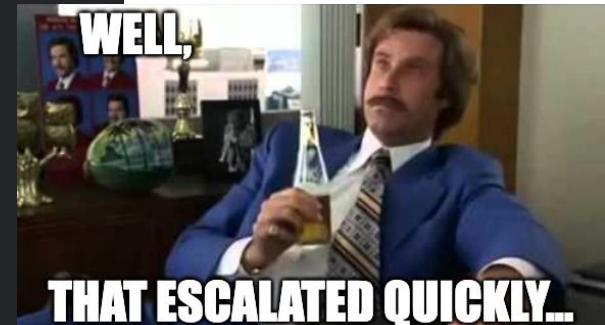
11 comments 60 reviews 140 files +6425 -780 309 replies Last reply 15 days ago

# Improper Versioning and Lack of Conversion Strategy

CRD versioning is hard! Spend some time creating best strategy for your use case, maybe even before you need it!

A screenshot of a GitHub pull request page for issue #3318. The title is "create apis/v1alpha2, add new ResourceSchemas". The description reads: "I have begun working on the support for the new "virtual" ResourceSchemas in APIExports. The plan was to bump apis/v1alpha1 and I hope I did it as intended (i.e. adding the new fields to v1alpha2 and changing the codebase to use v1alpha2 everywhere): <https://github.com/kcp-dev/kcp/pull/3318> -- is this the right direction or should i correct early before going deeper?" The pull request has 11 comments, 60 reviews, 140 files, and 6425 changes. It was last updated by xrstf on March 3, 2025, with 18 commits. There are 309 replies and the last reply was 15 days ago.

Somes (most of the times) we  
don't know how to do  
it too :)



# Neglecting Controller Design and Operational Considerations

```
reconcilers := []reconciler{
    &metaDataReconciler{},
    &finazilerReconciler{},
    &deletionReconciler{
        getLogicalCluster: func(ctx context.Context, cluster logicalcluster.Path) {}
        deleteLogicalCluster: func(ctx context.Context, cluster logicalcluster.Path) {}
        ...
    },
    &schedulingReconciler{
        ...
    },
    &phaseReconciler{
        ...
    },
}
```

Each of these implements reconcile()

- Easy to test
- No mocking needed
- Easy to read

# Neglecting Controller Design and Operational Considerations



```
var errs []error
    requeue := false
    for _, r := range reconcilers {
        var err error
        var status reconcileStatus
        status, err = r.reconcile(ctx, ws)
        if err != nil {
            errs = append(errs, err)
        }
        if status == reconcileStatusStopAndRequeue {
            requeue = true
            break
        }
    }
    return requeue, utilerrors.NewAggregate(errs)
```

One for next slide :)

- Easy to test
- No mocking needed
- Easy to read

# Neglecting Controller Design and Operational Considerations



```
const (
    reconcileStatusStopAndRequeue reconcileStatus = iota
    reconcileStatusStop
    reconcileStatusContinue
)
```

- Small steps leads small decisions
  - Should I **StopAndRequeue**? Example: Add reconciler if not exists
  - Should I **Stop**? Example: Object is in terminal state - not much we can do.
  - **Continue?** Example: Reconciler computed desired replicas, next reconciler step will apply it (bad, example I know...)

# Neglecting Controller Design and Operational Considerations



```
// If the object being reconciled changed as a result, update it.  
oldResource := &workspaceResource{ObjectMeta: old.ObjectMeta, Spec: &old.Spec, Status: &old.Status}  
newResource := &workspaceResource{ObjectMeta: workspace.ObjectMeta, Spec: &workspace.Spec, Status: &workspace.Status}  
if err := c.commit(ctx, oldResource, newResource); err != nil {  
    errs = append(errs, err)
```

## Use existing patterns

- Committer
- Controller-runtime
- Kubebuilder
- client-go
- utilerrors "k8s.io/apimachinery/pkg/util/errors"
- utilruntime "k8s.io/apimachinery/pkg/util/runtime"
- ...

# How this is better than what we have now?



KubeCon



CloudNativeCon

Europe 2025

---

# How this is better than what we have now?

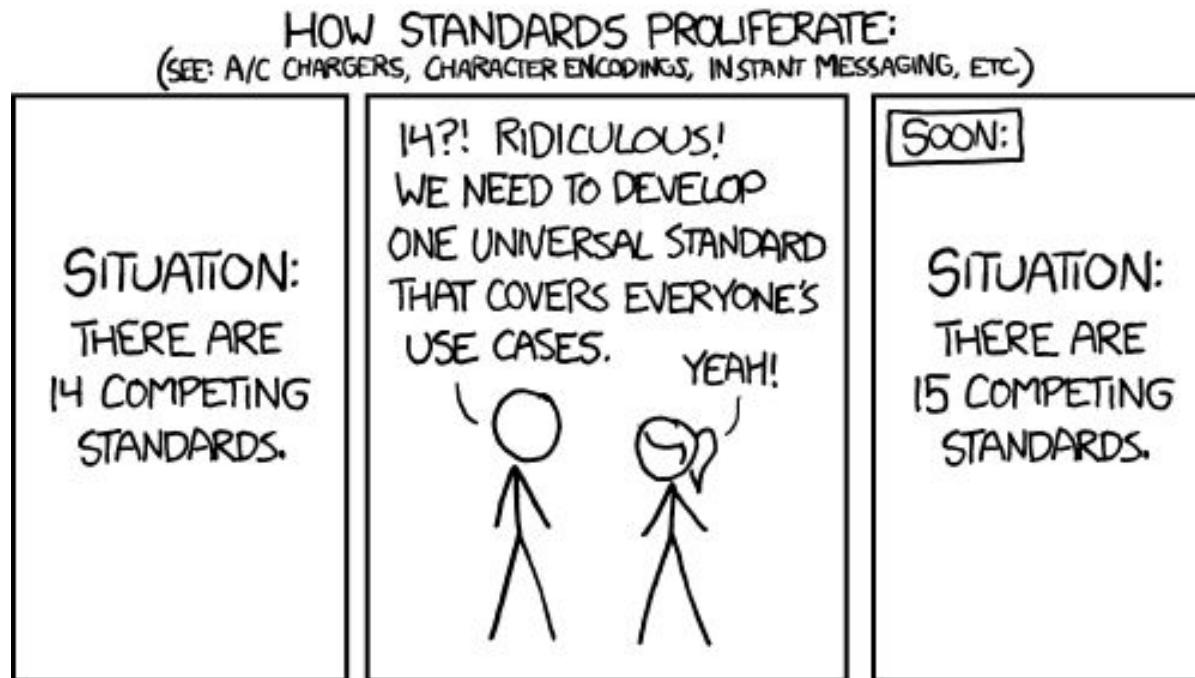


KubeCon



CloudNativeCon

Europe 2025



# How this is better than what we have now?

It's a standard you gonna use either way, you like it or not :)

**Scheduling & Orchestration**  

**API Gateway**  

**Remote Procedure Call**  

**Orchestration & Management**

 CNCF GRADUATED	 CNCF GRADUATED	 CNCF INCUBATING	 CNCF INCUBATING	 CNCF INCUBATING	 CNCF INCUBATING	 CNCF INCUBATING	 CNCF INCUBATING	 HESOS
 ARMada	 Akun Mesh Fabric	 capsule	 EGEE CNAF	 中移混布	 ClusterNet	 Glimpse	 Trekster	 ERASER
 kubers	 KubeAdmiral	 KubeWharf	 KubeSlice	 KUBERNETE	 Kured	 Slurm	 Nomad	 Open Cluster Management
 OPEN FUNCTION	 PREFECT	 SERVERLESS DEVs	 Sidekiq	 upbound	 Katalyst	 kestra	 koordinator	 kube-green

 CNCF INCUBATING	 gSCALE	 akana	 APIDAK	 APISIX
 EMISSIONARY INGRESS	 Emissary Ingress	 Emissary Ingress	 Emissary Ingress	 Emissary Ingress
 GLOO GATEWAY	 gRPC-GATEWAY	 Hongo	 Migress	 Kong
 KrakenD	 Kuadrant	 KubeGateway	 Kuksa	 MuleSoft
 ngrok	 trafik	 Tyk	 WSO2	 TARS

 Service Proxy	
 envoy	 CONTOUR
 AVI	 BFE
 Caddy	 citrix
 NETFLIX OSS	 HAProxy
 inlets	 LoxiLB
 MetalLB	 mosn
 NOVA	 Tengine

 Service Mesh	
 Istio	 LINKERD
 gRPC-MESH	 Envoy Mesh
 Consul	 Emissary Mesh
 gRPC-CONSUL	 Envoy Mesh
 gRPC-KUBE	 Envoy Mesh
 gRPC-NOMAD	 Envoy Mesh
 gRPC-NGINX	 Envoy Mesh

 Coordination & Service Discovery	
 CoreDNS	 etcd
 KubeBrain	 Nacos
 K8S-DNS	 NETFLIX OSS
 Xline	

# How this is better than what we have now?

It's a standard you gonna use either way, you like it or not :)

## So why not stick with it?

Orchestration & Management

Scheduling & Orchestration											
KEDA	kubernetes	Crossplane	KARMADA	Knative	Kubeflow	VOLCANO	wasmcloud	Amazon ETS	EMISSARY INGRESS	iSCAPE	akana
Armos	Azurkino Habitat	capsule	BDBE CNP	Cilium	ClusterNet	Clusterpedia	Dokku	ERASER	Fluid	HAKU	kestra
kubes	KubeAdmiral	KubeWerf	KubeSlice	Kured	slurm	Nomad	Open Cluster Management	OPEN FUNCTION	Open Nebula	PREFECT	SERVERLESS DEVX
KEDa	Kubernetes	Crossplane	Karmada	Knative	Kubeflow	Volcano	wasmCloud	Amazon ETS	Emissary Ingress	iSCAPE	akana
Armos	Azurkino Habitat	capsule	BDBE CNP	Cilium	ClusterNet	Clusterpedia	Dokku	ERASER	Fluid	HAKU	kestra
kubes	KubeAdmiral	KubeWerf	KubeSlice	Kured	slurm	Nomad	Open Cluster Management	OPEN FUNCTION	Open Nebula	PREFECT	SERVERLESS DEVX

API Gateway

API Gateway											
gRPC	Apache Thrift	Apache APACHE	CloudWeGo	DUBBO	GO	Haproxy	Kong	KrakenD	NGINX	Trafik	WSO2
Easyservice	shard.io	Envoy	Envoy	Emissary	Emissary	GLOO GATEWAY	Higress	Kuadrant	KubeDuties	Kuks	Juniper
gRPC	Apache Thrift	Apache APACHE	CloudWeGo	DUBBO	GO	Haproxy	Kong	KrakenD	NGINX	Trafik	WSO2
Easyservice	shard.io	Envoy	Envoy	Emissary	Emissary	GLOO GATEWAY	Higress	Kuadrant	KubeDuties	Kuks	Juniper
gRPC	Apache Thrift	Apache APACHE	CloudWeGo	DUBBO	GO	Haproxy	Kong	KrakenD	NGINX	Trafik	WSO2
Easyservice	shard.io	Envoy	Envoy	Emissary	Emissary	GLOO GATEWAY	Higress	Kuadrant	KubeDuties	Kuks	Juniper

Service Proxy

Service Proxy											
envoy	CONTOUR	AVI	BFE	Caddy	citrix	NETFLIX OSS ZULU	HAProxy	inlets	Istio	LINKERD	Aeraki Mesh
envoy	CONTOUR	AVI	BFE	Caddy	citrix	NETFLIX OSS ZULU	HAProxy	inlets	Istio	LINKERD	Aeraki Mesh
envoy	CONTOUR	AVI	BFE	Caddy	citrix	NETFLIX OSS ZULU	HAProxy	inlets	Istio	LINKERD	Aeraki Mesh
envoy	CONTOUR	AVI	BFE	Caddy	citrix	NETFLIX OSS ZULU	HAProxy	inlets	Istio	LINKERD	Aeraki Mesh
envoy	CONTOUR	AVI	BFE	Caddy	citrix	NETFLIX OSS ZULU	HAProxy	inlets	Istio	LINKERD	Aeraki Mesh

Service Mesh

Service Mesh											
gRPC	Apache Thrift	Apache APACHE	CloudWeGo	DUBBO	GO	Haproxy	Kong	KrakenD	NGINX	Trafik	WSO2
gRPC	Apache Thrift	Apache APACHE	CloudWeGo	DUBBO	GO	Haproxy	Kong	KrakenD	NGINX	Trafik	WSO2
gRPC	Apache Thrift	Apache APACHE	CloudWeGo	DUBBO	GO	Haproxy	Kong	KrakenD	NGINX	Trafik	WSO2
gRPC	Apache Thrift	Apache APACHE	CloudWeGo	DUBBO	GO	Haproxy	Kong	KrakenD	NGINX	Trafik	WSO2
gRPC	Apache Thrift	Apache APACHE	CloudWeGo	DUBBO	GO	Haproxy	Kong	KrakenD	NGINX	Trafik	WSO2

Coordination & Service Discovery

Coordination & Service Discovery											
CoreDNS	etcd	Consul	Emissary	Aeraki Mesh	KubeDuties	Juniper	Trafik	Kuks	Nasp	OpenServiceMesh	gRPC
CoreDNS	etcd	Consul	Emissary	Aeraki Mesh	KubeDuties	Juniper	Trafik	Kuks	Nasp	OpenServiceMesh	gRPC
CoreDNS	etcd	Consul	Emissary	Aeraki Mesh	KubeDuties	Juniper	Trafik	Kuks	Nasp	OpenServiceMesh	gRPC
CoreDNS	etcd	Consul	Emissary	Aeraki Mesh	KubeDuties	Juniper	Trafik	Kuks	Nasp	OpenServiceMesh	gRPC
CoreDNS	etcd	Consul	Emissary	Aeraki Mesh	KubeDuties	Juniper	Trafik	Kuks	Nasp	OpenServiceMesh	gRPC



KubeCon



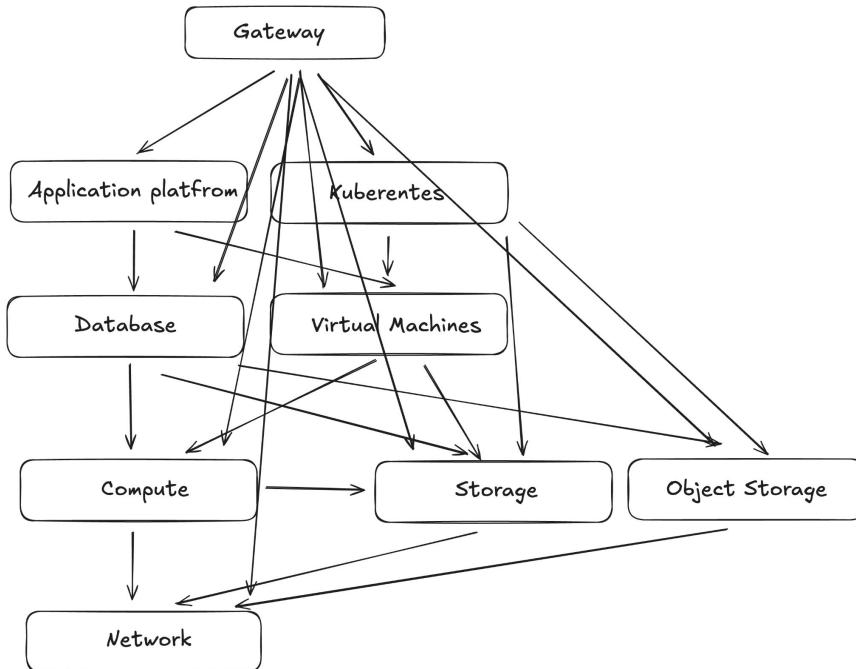
CloudNativeCon

Europe 2025

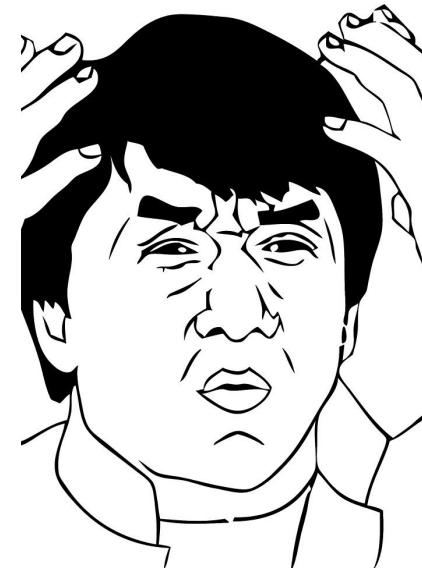
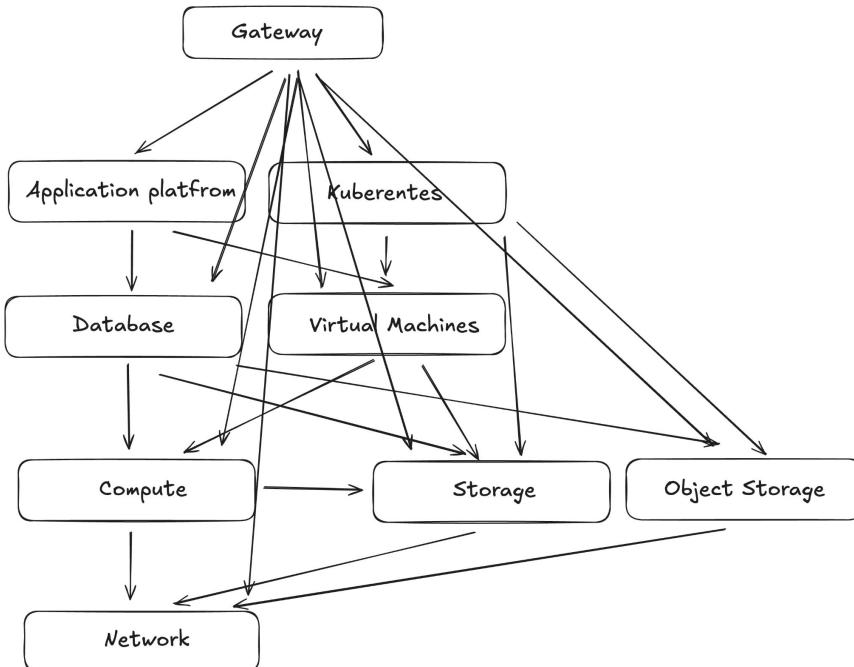
## Part 2: How kcp makes it easier



# Take a look as “multi-team microservice architecture”



# Take a look as “multi-team microservice architecture”





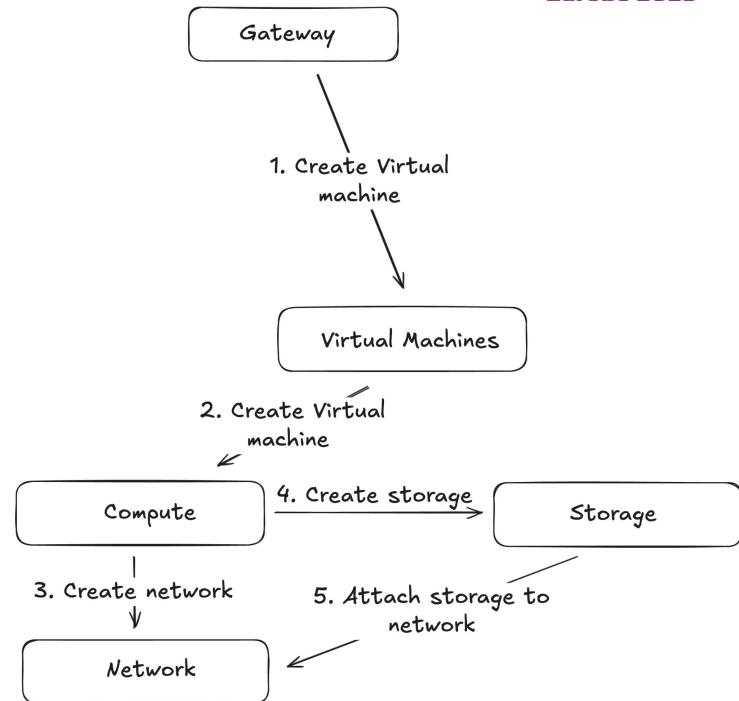
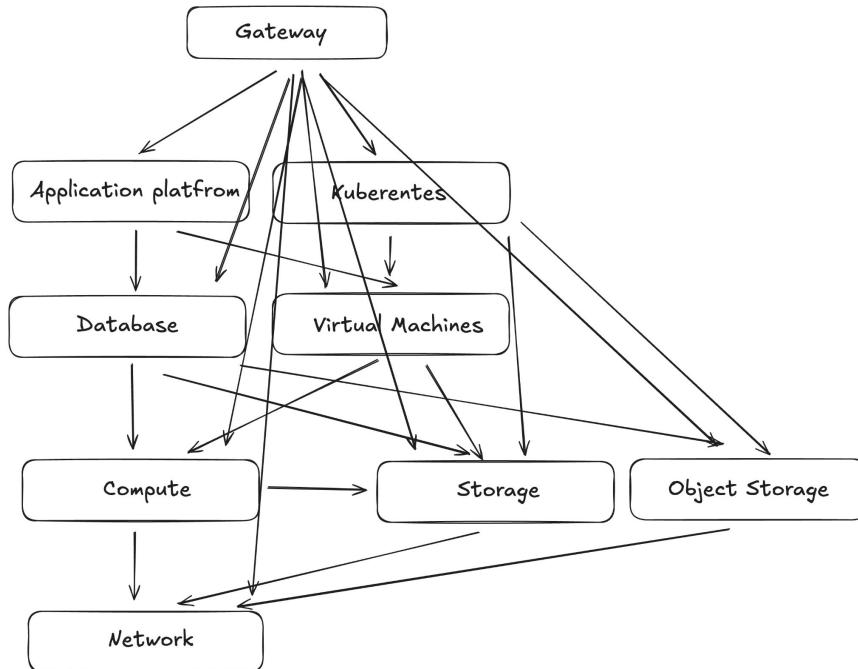
KubeCon



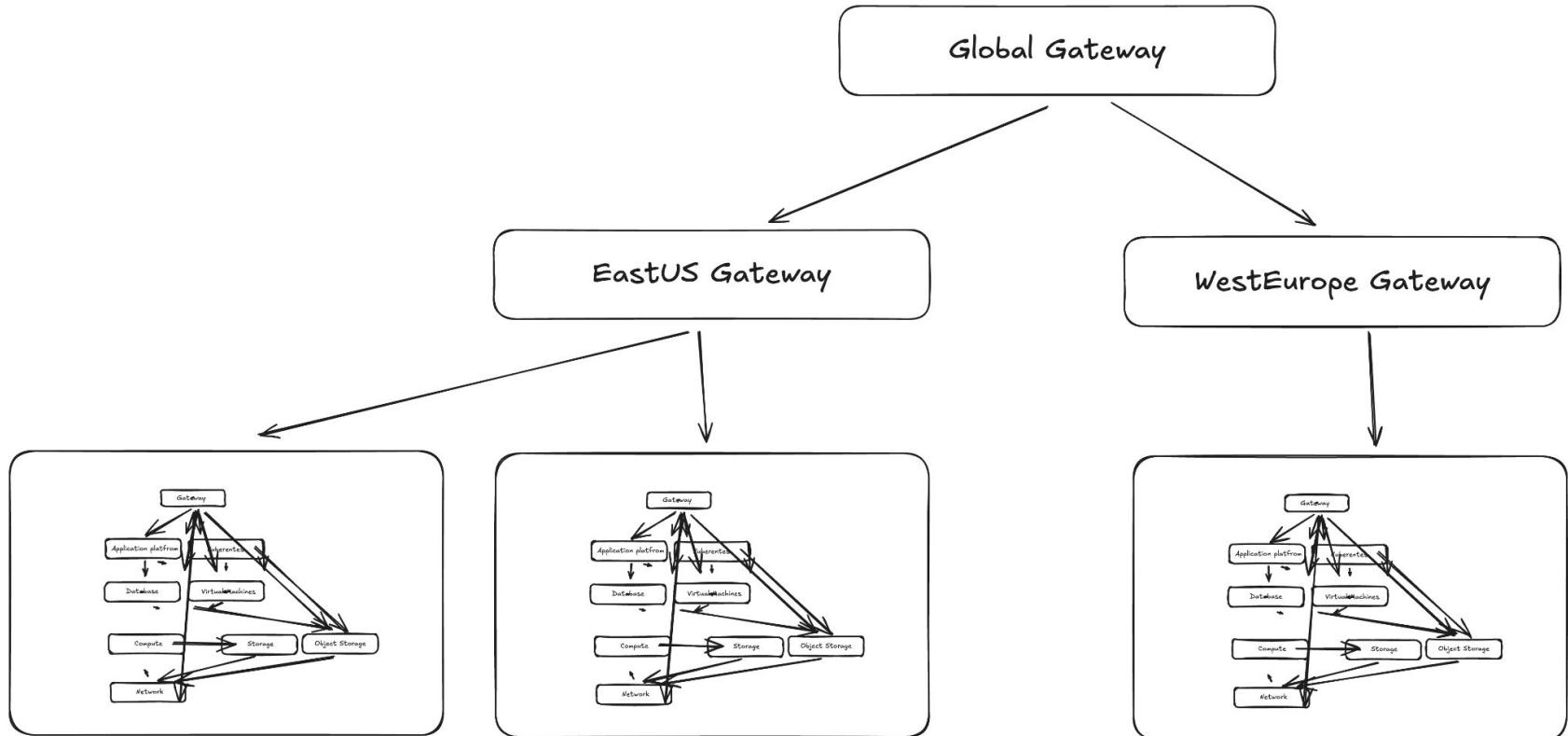
CloudNativeCon

Europe 2025

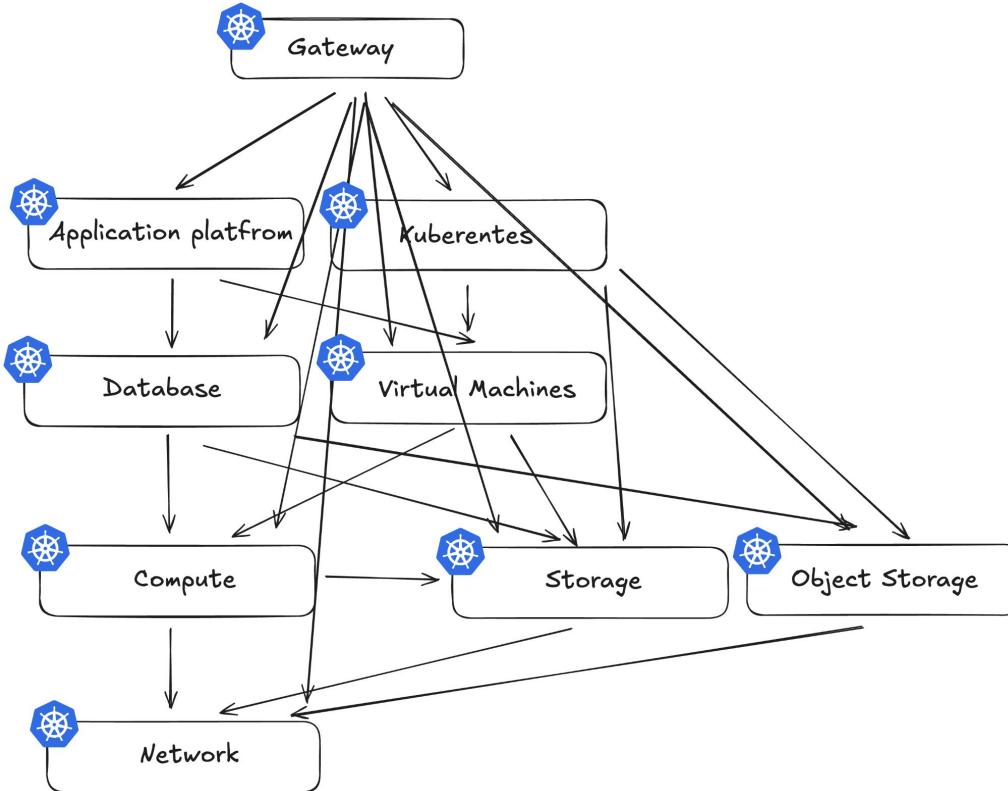
# Take a look as “multi-team microservice architecture”



# Take a look as “multi-team microservice architecture”

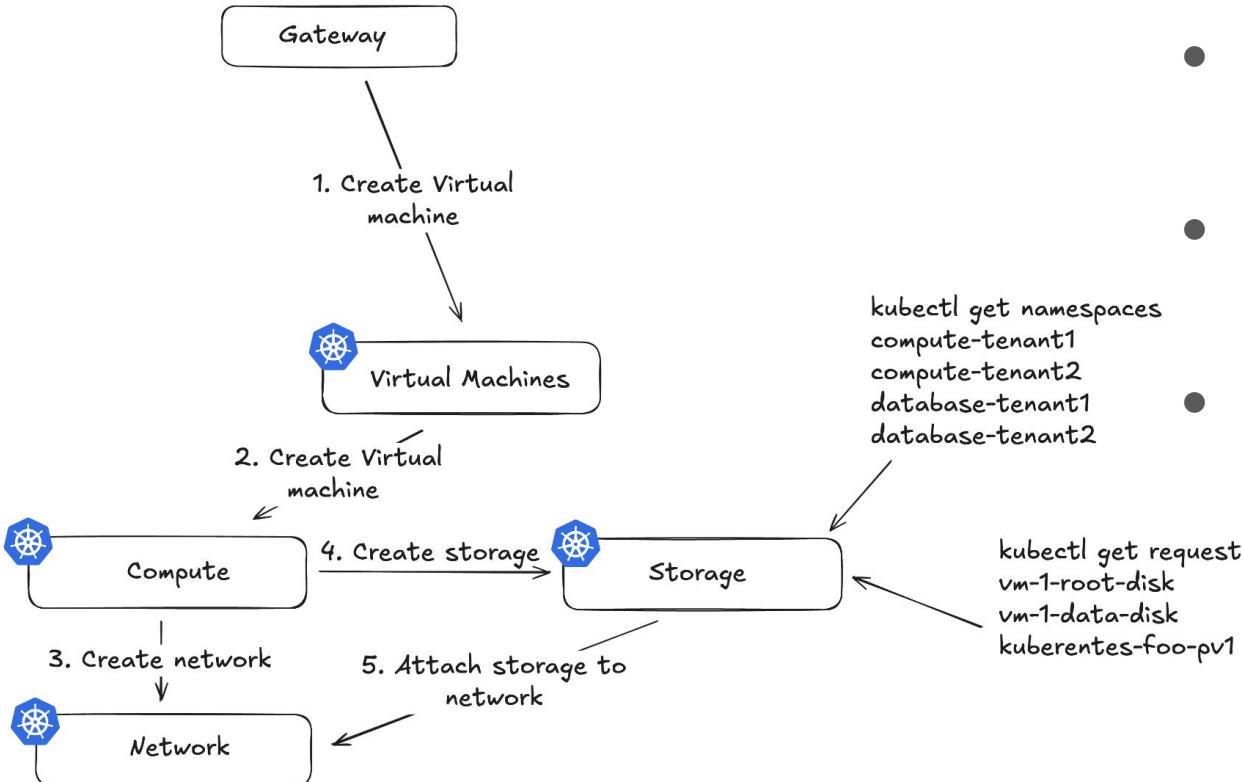


# Try doing this in Kubernetes?



- Every team will potentially have their own Kubernetes cluster
- Team will create CR's inside other teams clusters
- Cross cluster reconcilers everywhere!
- **Why this is bad?**

# Try doing this in Kubernetes?



- Every team will potentially have their own Kubernetes cluster
- Team will create CR's inside other teams clusters
- Cross cluster reconcilers everywhere!

# Try doing this in Kubernetes?

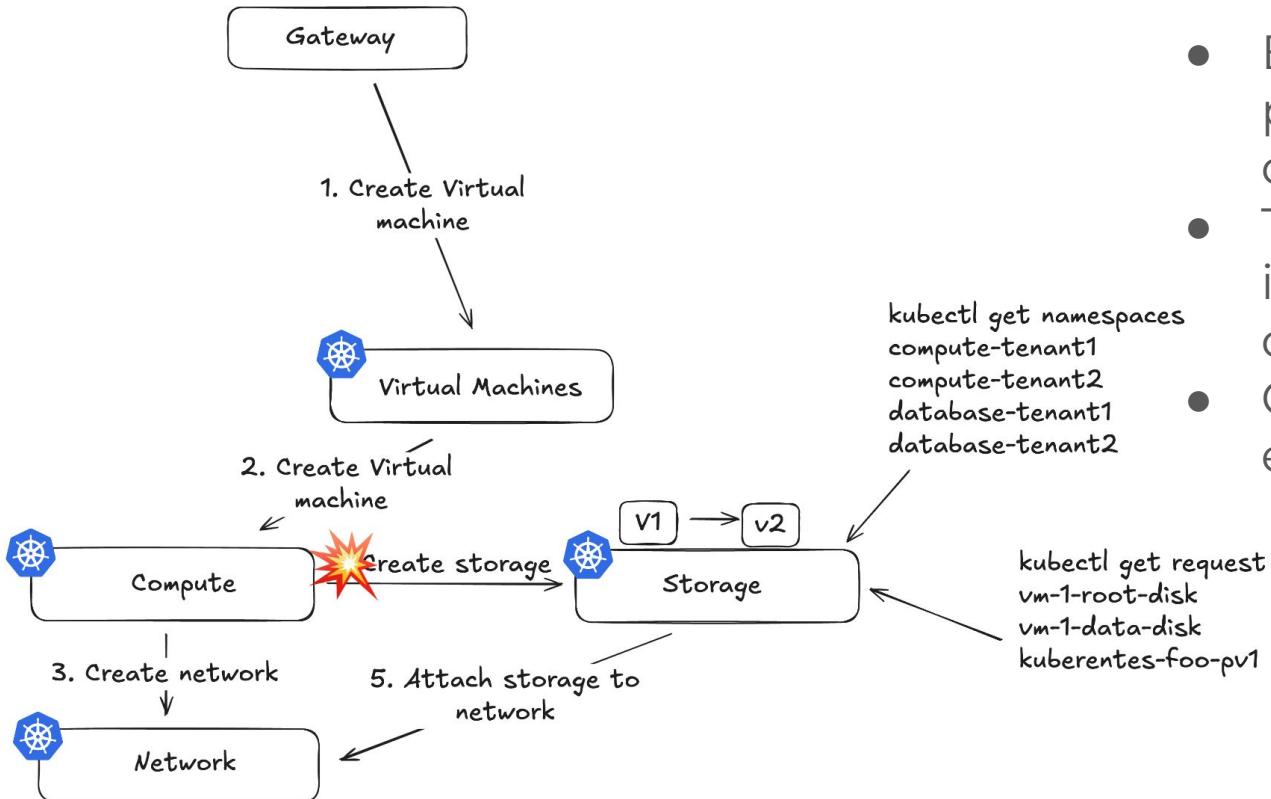


KubeCon



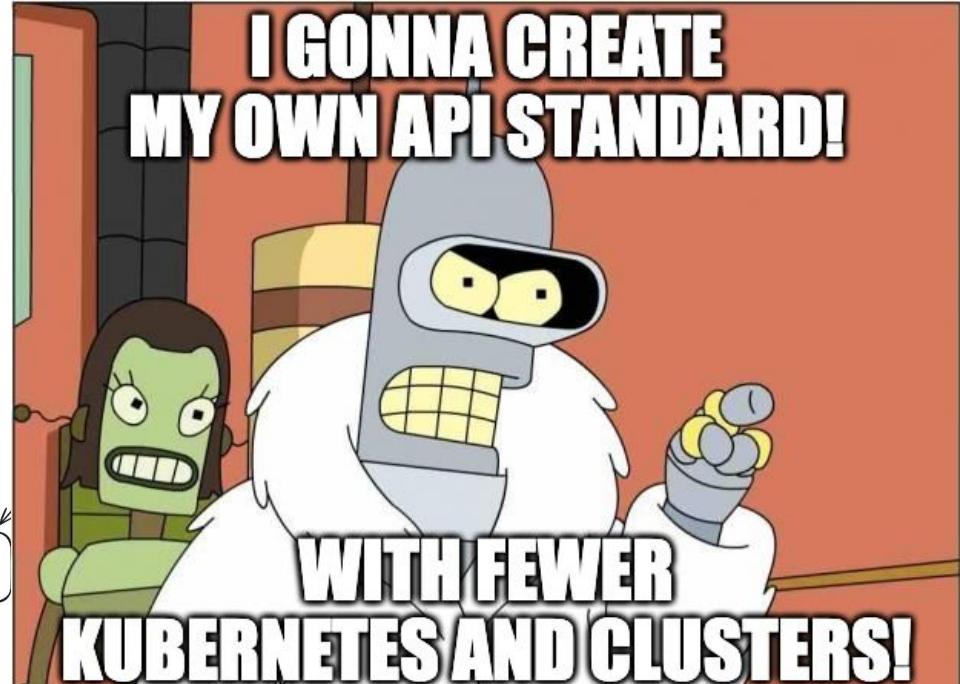
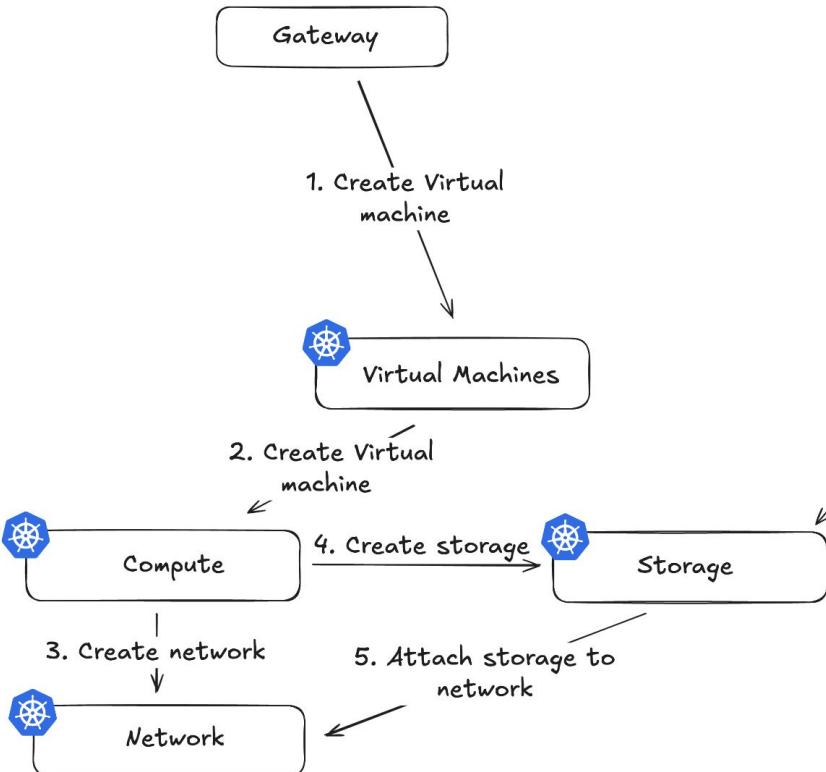
CloudNativeCon

Europe 2025



- Every team will potentially have their own Kubernetes cluster
- Team will create CR's inside other teams clusters
- Cross cluster reconcilers everywhere!

Try doing this in Kubernetes?





## Many clusters problem

While API standard (KRM) is amazing, its vessel (🚢) is not (or is it?)!

- First: We are forced to do fragmentation to keep tenancy & isolation!
- Second: API management frameworks are not well adopted



## Alternatives?

vClusters:

- Required pod(s) per tenant. You save on “cluster count”, but all other challenges still there.
- API management is not addressed.
- (Many clusters problem)<sup>^2</sup>.



## Alternatives?

Generic control plane:

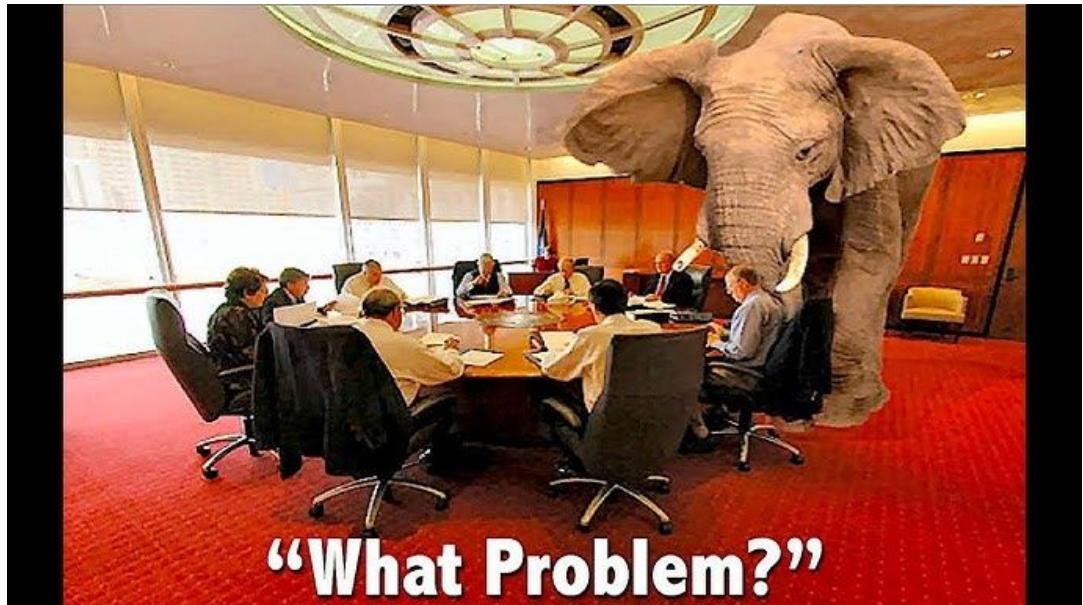
- Required pod per tenant. You save on “cluster count”, but all other challenges still there.
- API management is not addressed.
- (Many clusters problem)<sup>^3</sup>.
- You save on resources (a bit).

<https://github.com/kcp-dev/generic-controlplane>



## Alternatives?

- Strong multi-tenancy
- API management!





# kcp - Kubernetes-like control plane



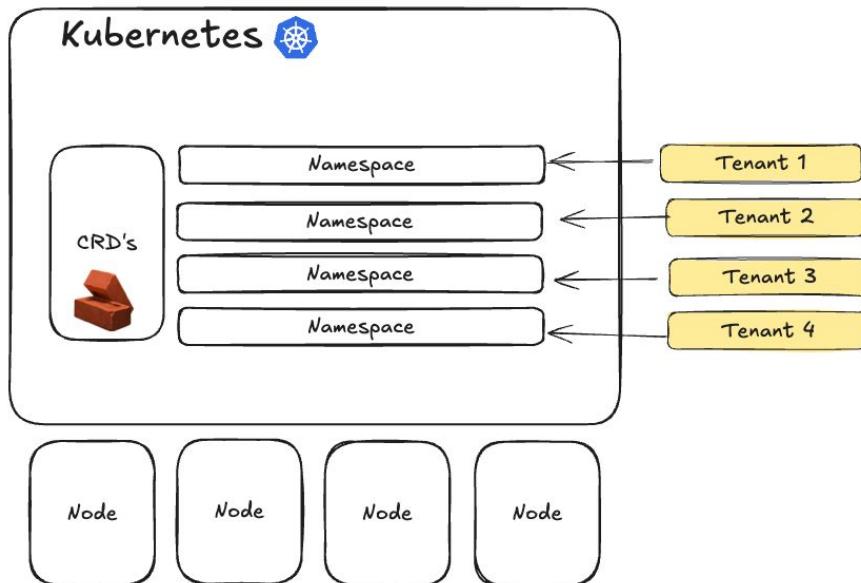
KubeCon



CloudNativeCon

Europe 2025

Old world





# kcp - Kubernetes-like control plane



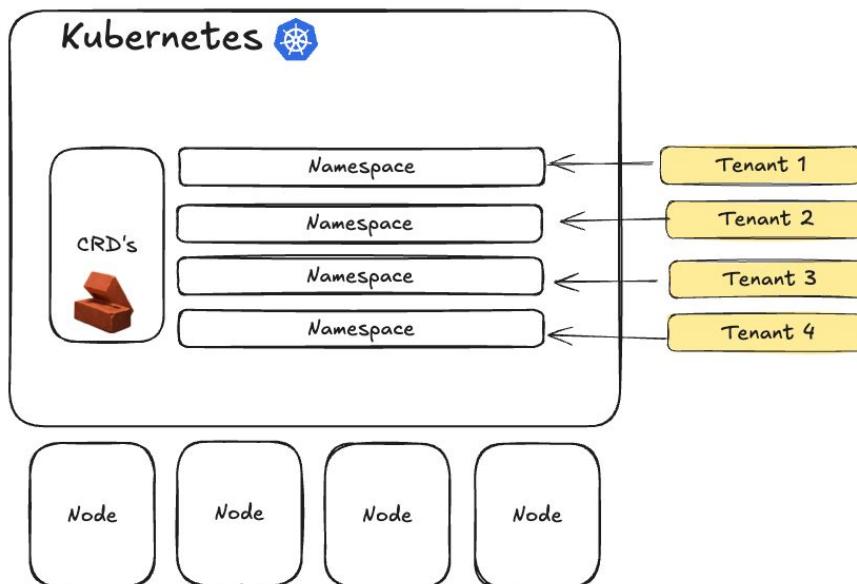
KubeCon



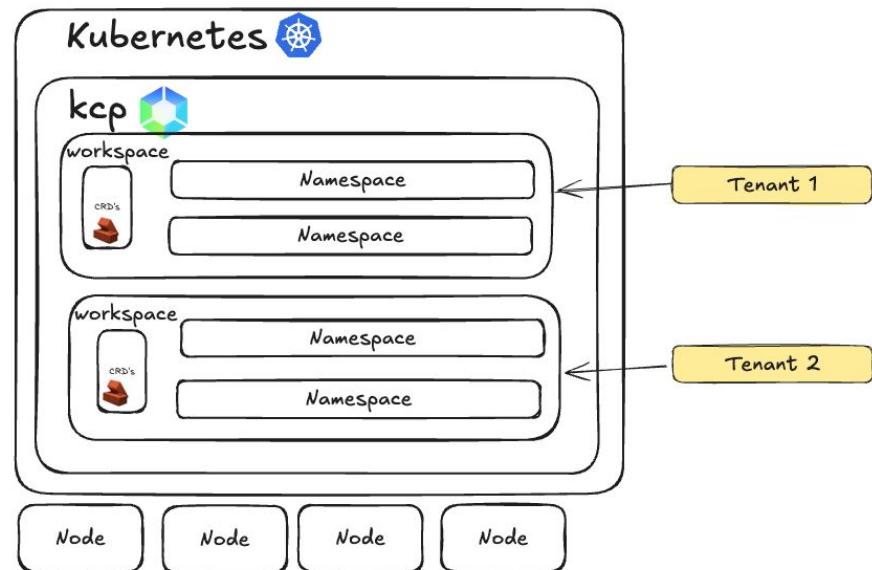
CloudNativeCon

Europe 2025

Old world



New world





# Control Plane

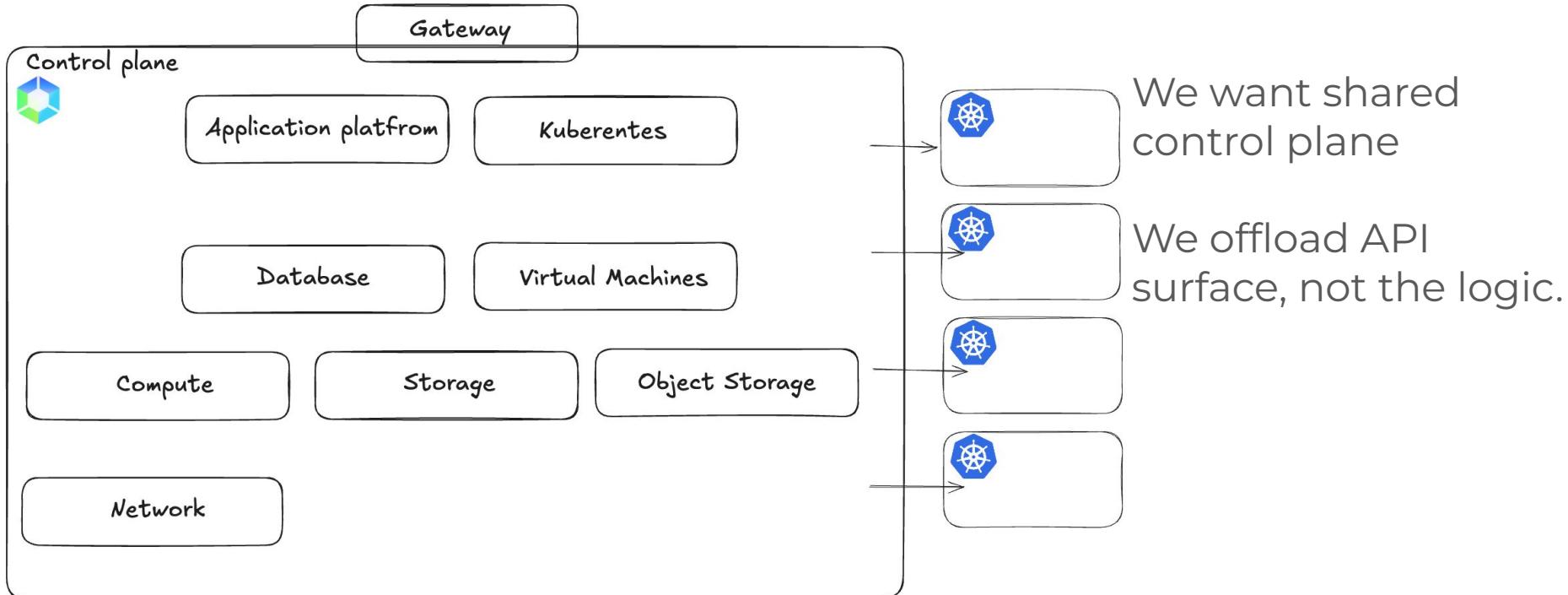


KubeCon



CloudNativeCon

Europe 2025





# Control Plane

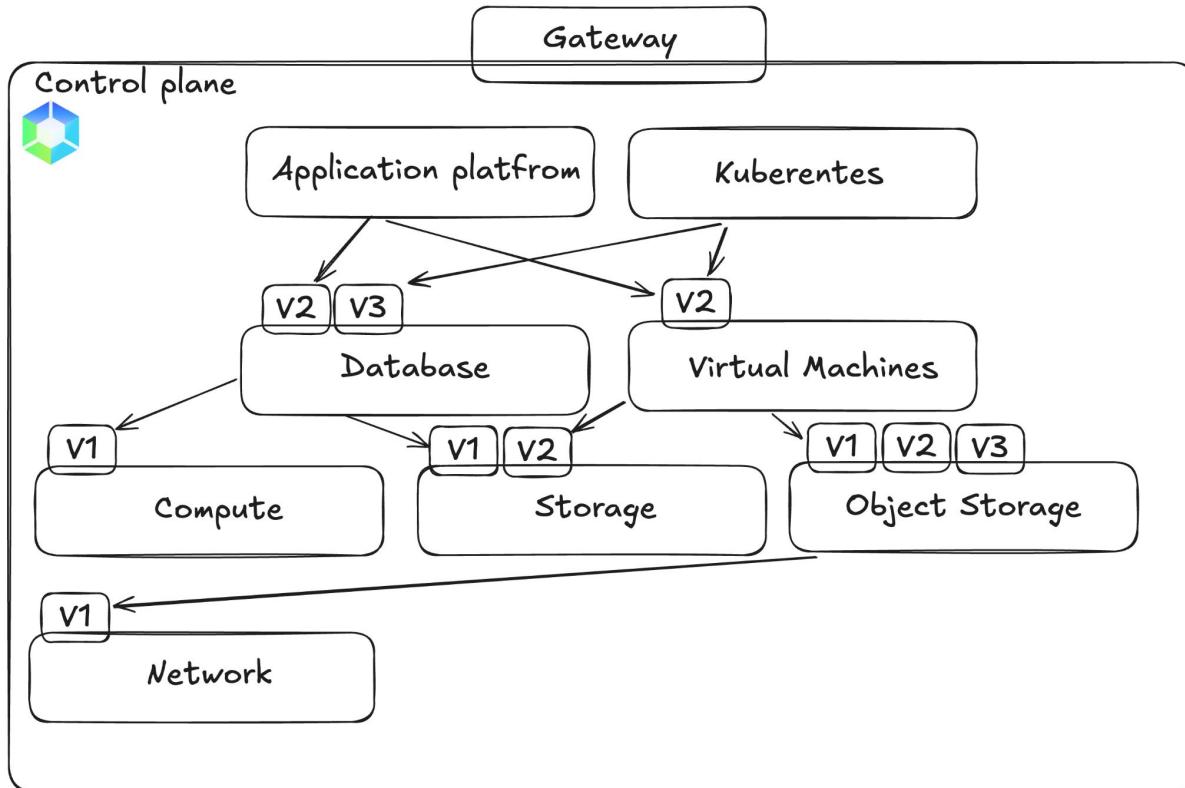


KubeCon



CloudNativeCon

Europe 2025



We want shared  
control plane

Strong  
multi-tenancy

Less fragmentation



# Control Plane

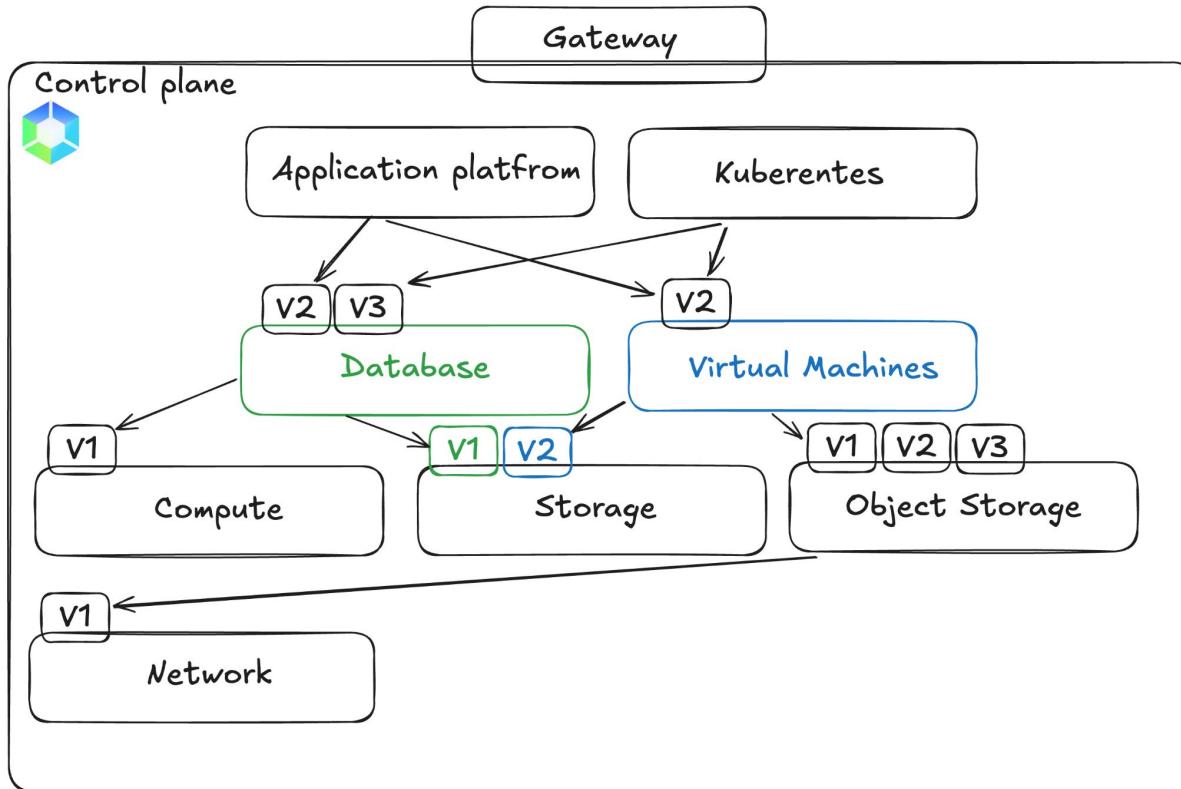


KubeCon



CloudNativeCon

Europe 2025



We want shared control plane

Strong multi-tenancy

Less fragmentation

Ability to support multiple version and clients



# How? We make KRM management easier

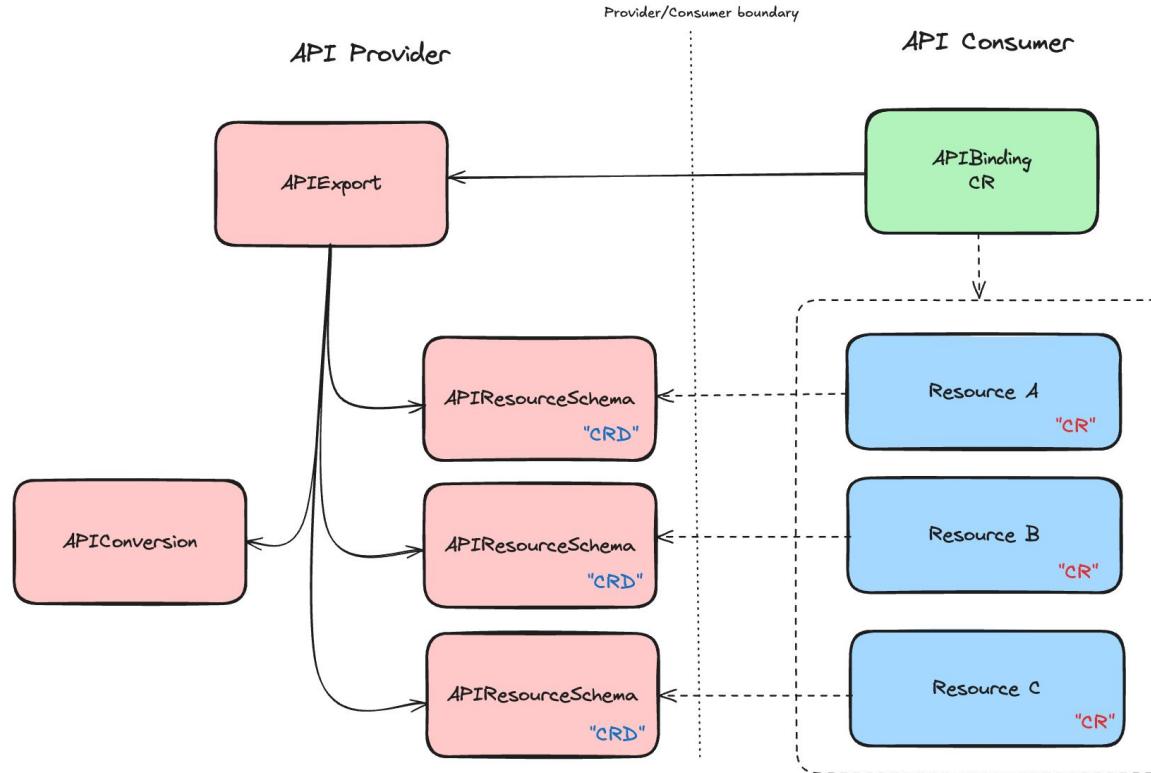


KubeCon



CloudNativeCon

Europe 2025





# How? We make KRM management easier

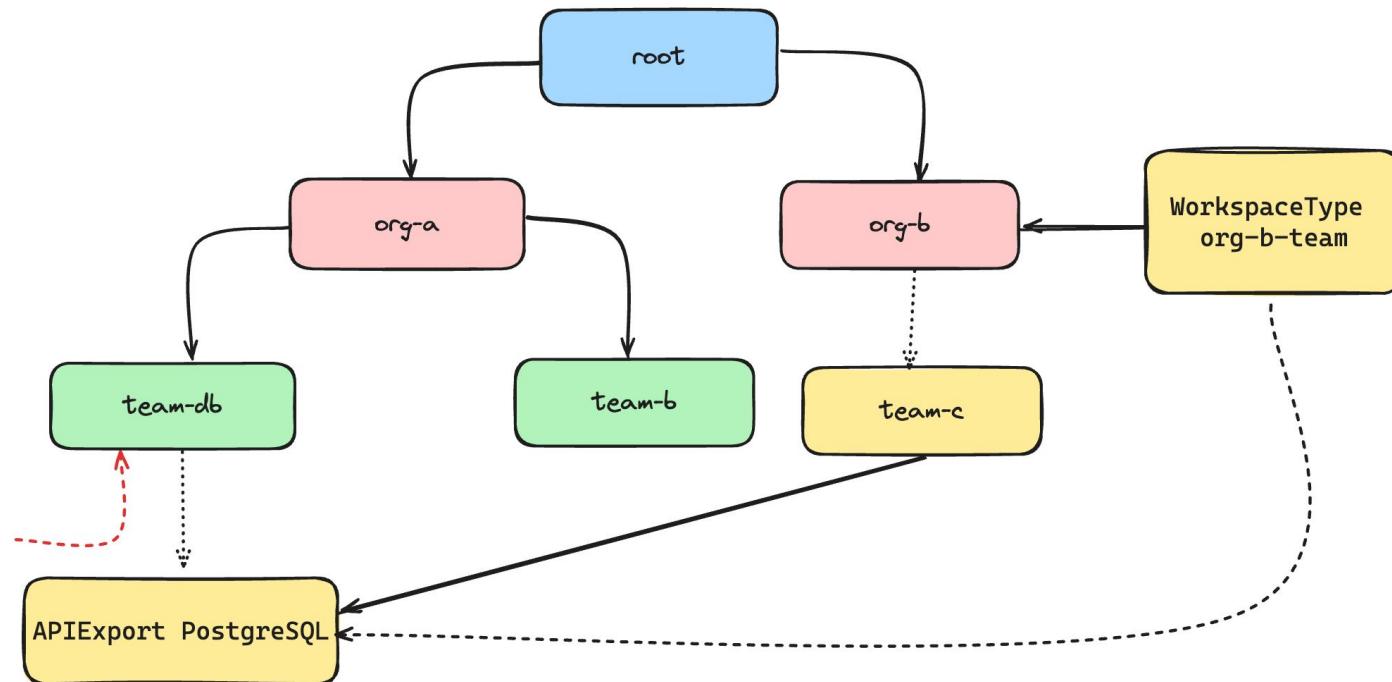


KubeCon



CloudNativeCon

Europe 2025





# How? We make KRM management easier



› k ws tree

```
.  
└── root  
    ├── applications  
    │   ├── application-platform-x  
    │   └── managed-kubernetes  
    ├── compute  
    │   └── virtual-machines  
    ├── networks  
    └── core  
    └── storage  
        ├── databases  
        └── object-storage
```



# How? We make KRM management easier



KubeCon

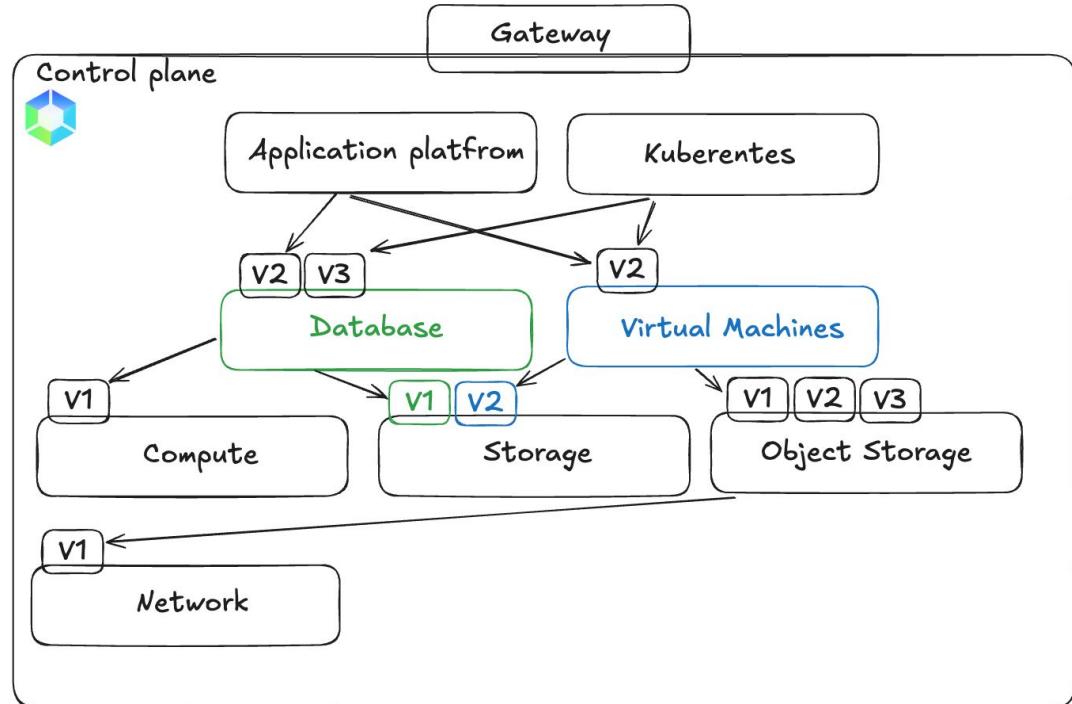


CloudNativeCon

Europe 2025

› k ws tree

```
.  
  └── root  
      ├── applications  
      │   ├── application-platfrom-x  
      │   └── managed-kubernetes  
      ├── compute  
      │   └── virtual-machines  
      ├── networks  
      └── core  
          ├── storage  
          │   ├── databases  
          │   └── object-storage
```





# How? We make KRM management easier

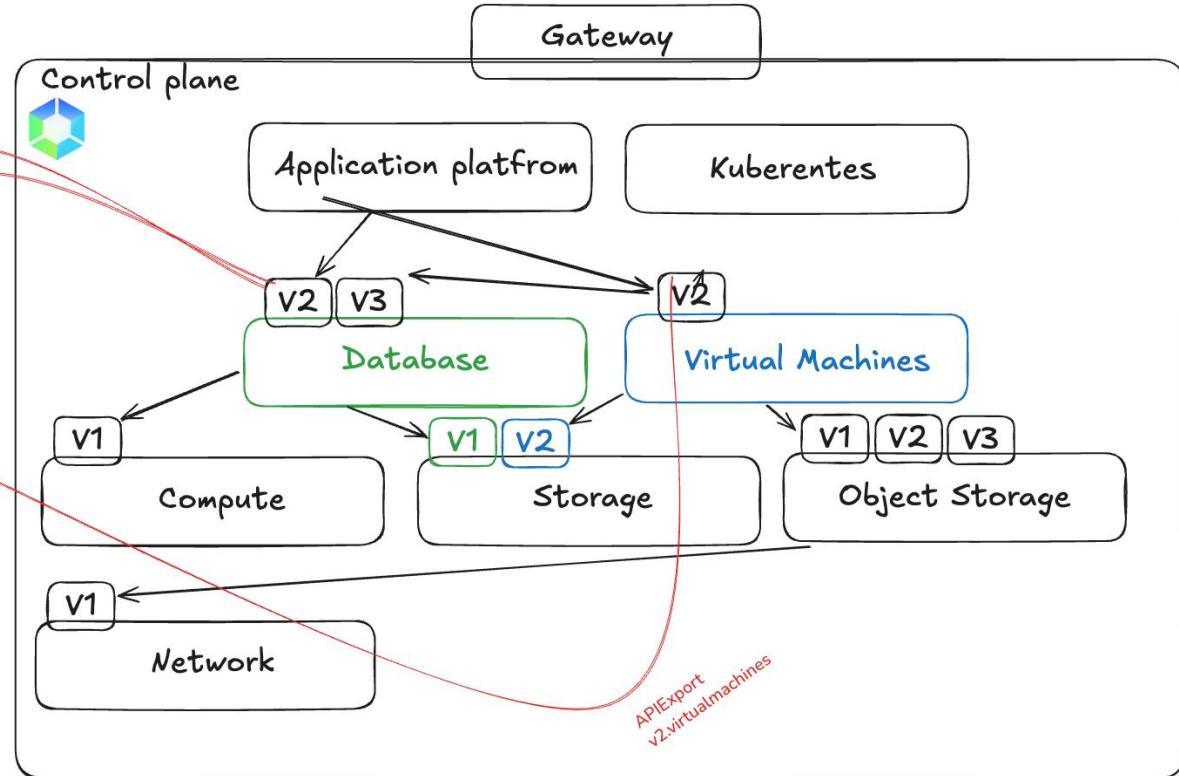
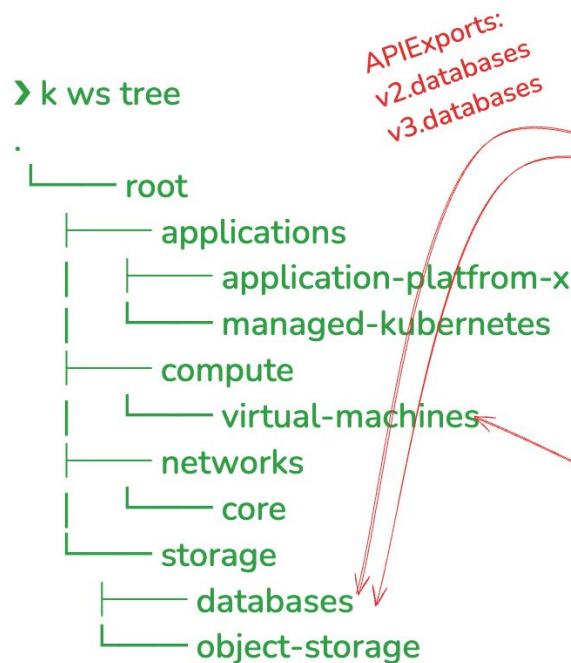


KubeCon



CloudNativeCon

Europe 2025





# How? We make KRM management easier



KubeCon



CloudNativeCon

Europe 2025

create

POST /v1beta1/{parent=projects/\*/locations/\*}/clusters

Creates a cluster, consisting of the specified number and type of Google Compute Engine instances.

root

HTTP request

POST https://sqladmin.googleapis.com/v1/projects/{project}/instances/{instance}/databases

compute

HTTP request

POST https://compute.googleapis.com/compute/v1/projects/{project}/zones/{zone}/instances

object-storage

Network

APIExport  
v2virtualmachines



**How? We make KRM management easier**



KubeCon



CloudNativeCon

Europe 2025

**Single platform to manage multiple Kubernetes resources, APIs and tenants  
by shifting API surface out of Kubernetes clusters**





## Part 3: Demo



Database as a service (DBaaS) - lift and shift

<https://github.com/kcp-dev/contrib/tree/main/20250401-kubecon-london/workshop>

# More sessions on KCP

Wednesday, April 2

11:15 BST

- Tutorial: Exploring Multi-Tenant Kubernetes APIs and Controllers With **Kcp** - Robert Vasek, Clyso GmbH; Nabarun Pal, Independent; Varsha Narsing, Red Hat; Marko Mudrinic, Kubermatic GmbH; Mangirdas Judeikis, Cast AI

Thursday, April 3

11:00 BST

- Extending Kubernetes Resource Model (KRM) Beyond Kubernetes Workloads - Mangirdas Judeikis, Cast AI & Nabarun Pal, Independent (Description: kcp)

15:00 BST

- Dynamic Multi-Cluster Controllers With Controller-runtime - Marvin Beckers, Kubermatic & Stefan Schimanski, Upbound (Description: kcp)

You are here!

# Questions?



Talk to us - look for logo!



Find us at #kcp-dev kubernetes slack  
Personas: @mjudeikis, @palnabarun



[github.com/kcp-dev/kcp](https://github.com/kcp-dev/kcp)



@kcp @mangirdas @theonlynabarun

Feedback 

