KubeCon | CloudNativeCon

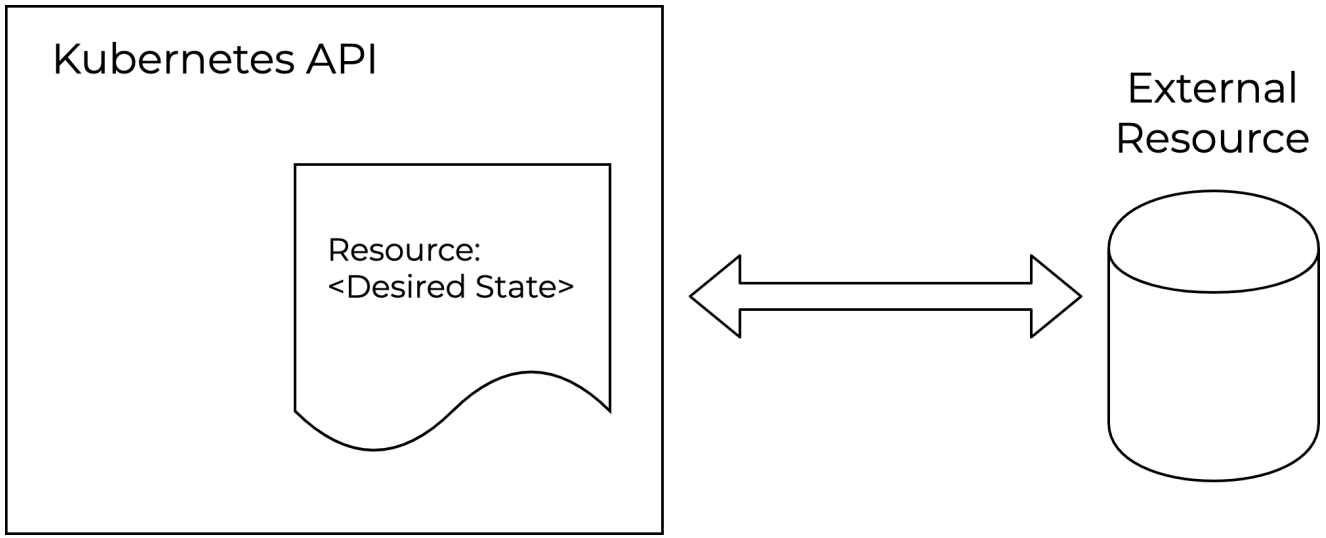
Europe 2020

Virtual

# Kubernetes as a General Purpose Control Plane: Scaling on Kubernetes

*Hasan Türken, Upbound*

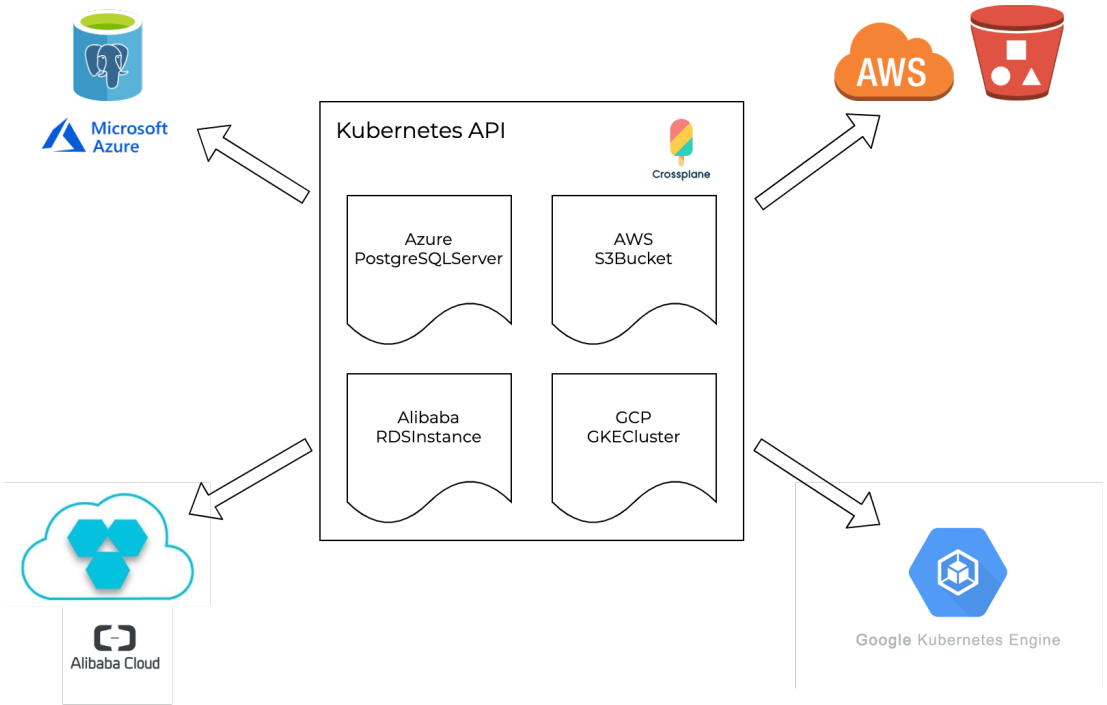
# Kubernetes
# as a
# General Purpose Control Plane



# Example: Crossplane

## Uses Kubernetes  as a General Purpose Control Plane

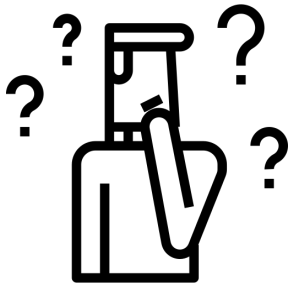
- Open Source
- Submitted as CNCF Sandbox project
- Extensible by Providers
- Manage infrastructure from Kubernetes



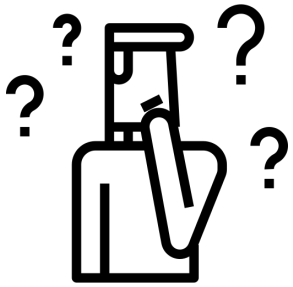
# Why Kubernetes API?

- **One API for All**
- Extensible
- Declarative
  - Define desired state
  - Active reconciliation
  - Self healing
  - GitOps Style
- Existing machinery and tools
  - Namespaces, RBAC
  - Garbage collection, finalizers
  - API CRUD semantics
  - Clients (e.g. kubectl, client-go)
- Existing tooling for controllers
  - Kubebuilder
  - Operator SDK
  - Controller runtime

# Problem

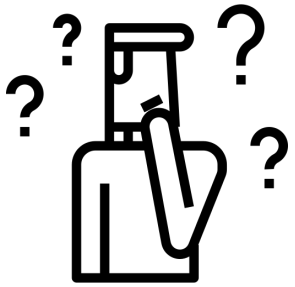


Running
isolated Instances of
Kubernetes Control Planes?



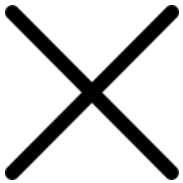
Running
isolated Instances of
Kubernetes Control Planes?

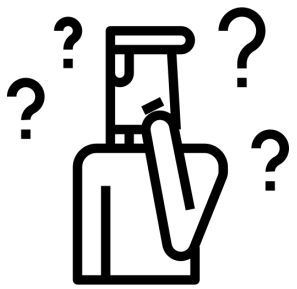
Run
a dedicated Kubernetes Cluster
per instance



Running
isolated Instances of
Kubernetes Control Planes
**at scale ?**

Run
a dedicated Kubernetes Cluster
per instance

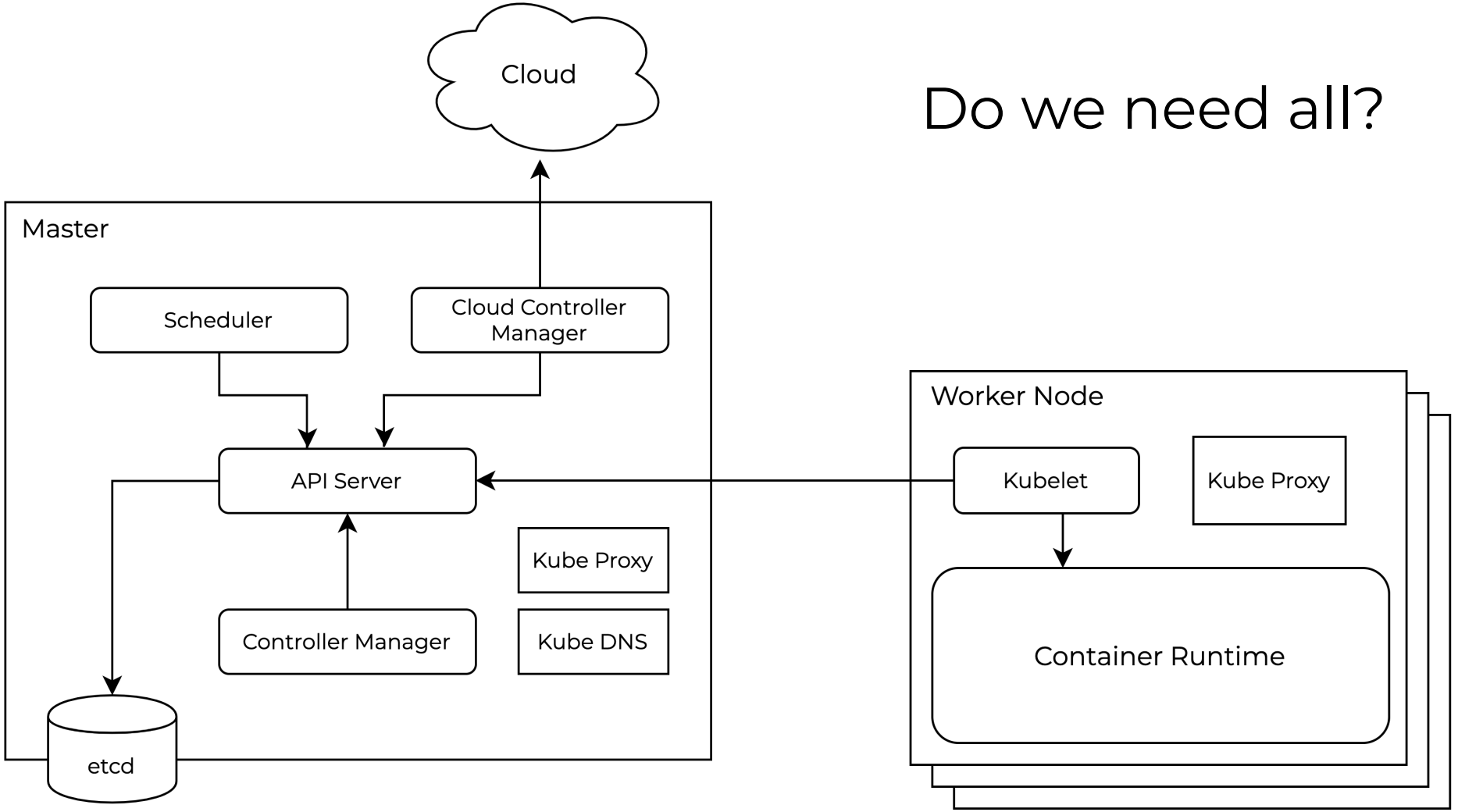




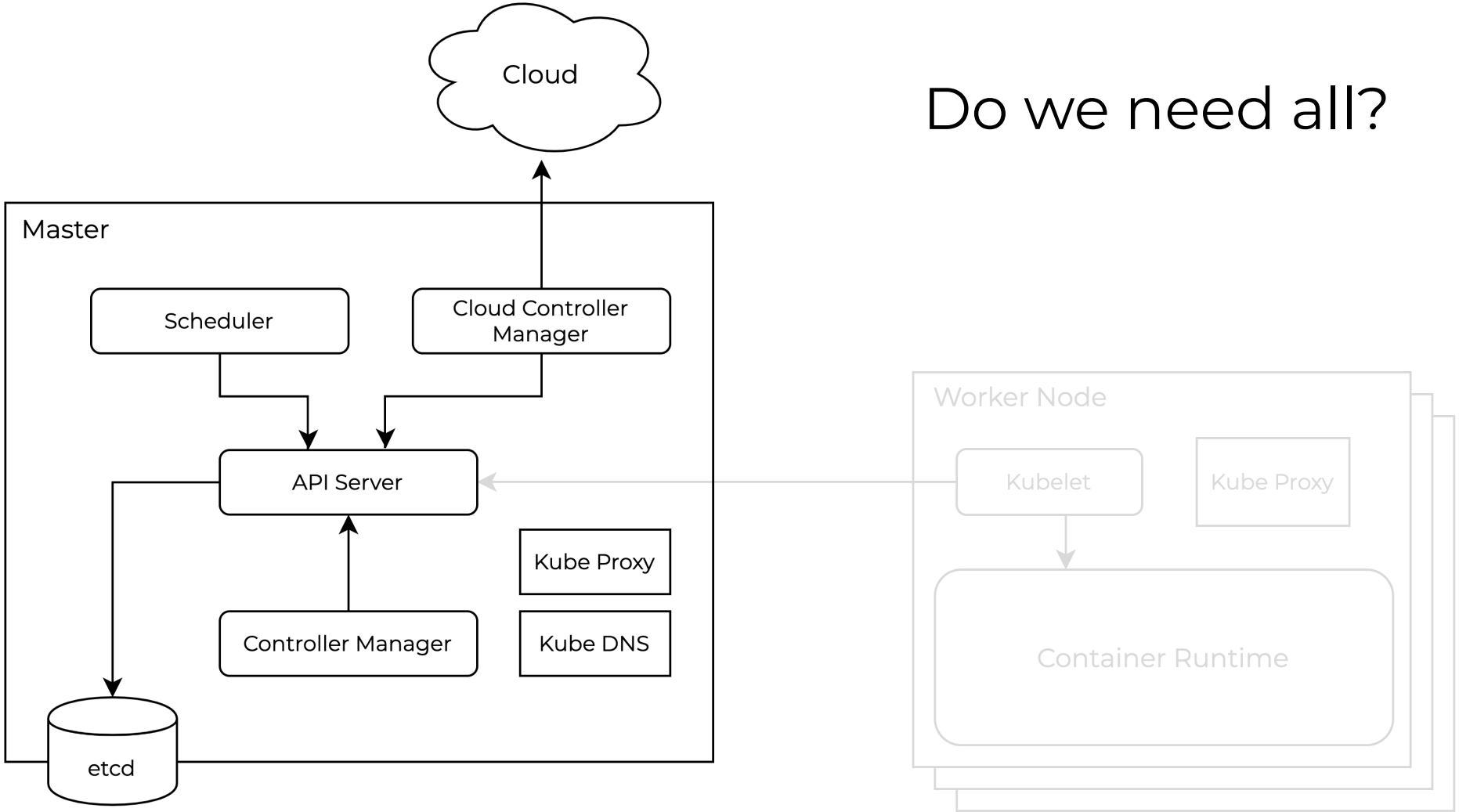
Running
isolated Instances of
Kubernetes Control Planes
**at scale**
where
**Kubernetes** is being used **as a**
**General Purpose Control Plane ?**

# Solution

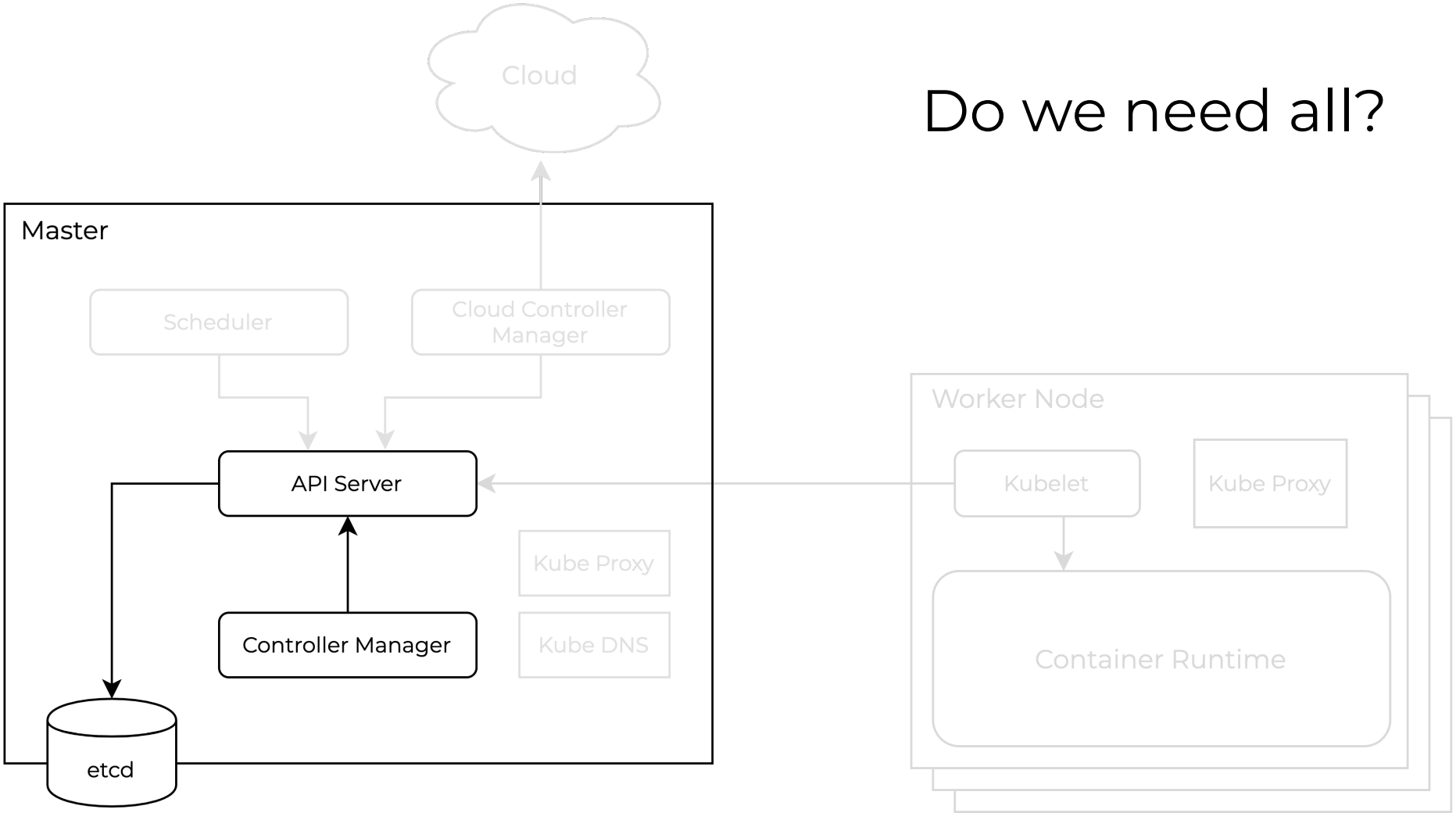
# Do we need all?



# Do we need all?



# Do we need all?



A closer look at
**Controller Manager**

*attachdetach* *bootstrapsigner* *cloud-node-lifecycle* *clusterrole-aggregation*

*cronjob* *csrcleaner* *csrapproving* *csrsigning* *daemonset* *deployment* *disruption*

*endpoint* *garbagecollector* *horizontalpodautoscaling* *job* *namespace* *nodeipam*

*nodelifecycle* *endpointslice* *persistentvolume-expander* *podgc* *pv-protection*

*pvc-protection* *replicaset* *persistentvolume-binder* *resourcequota* *root-ca-cert-publisher*

*route* *service* *replicationcontroller* *statefulset* *ttl* *ttl-after-finished*

*serviceaccount* *serviceaccount-token*

A closer look at
**Controller Manager**

*attachdetach* *bootstrapsigner* *cloud-node-lifecycle* *clusterrole-aggregation*

*cronjob* *csrcleaner* *csrapproving* *csrsigning* *daemonset* *deployment* *disruption*

*endpoint* *garbagecollector* *horizontalpodautoscaling* *job* *namespace* *nodeipam*

*nodelifecycle* *endpointslice* *persistentvolume-expander* *podgc* *pv-protection*

*pvc-protection* *replicaset* *persistentvolume-binder* *resourcequota* *root-ca-cert-publisher*

*route* *service* *replicationcontroller* *statefulset* *ttl* *ttl-after-finished*

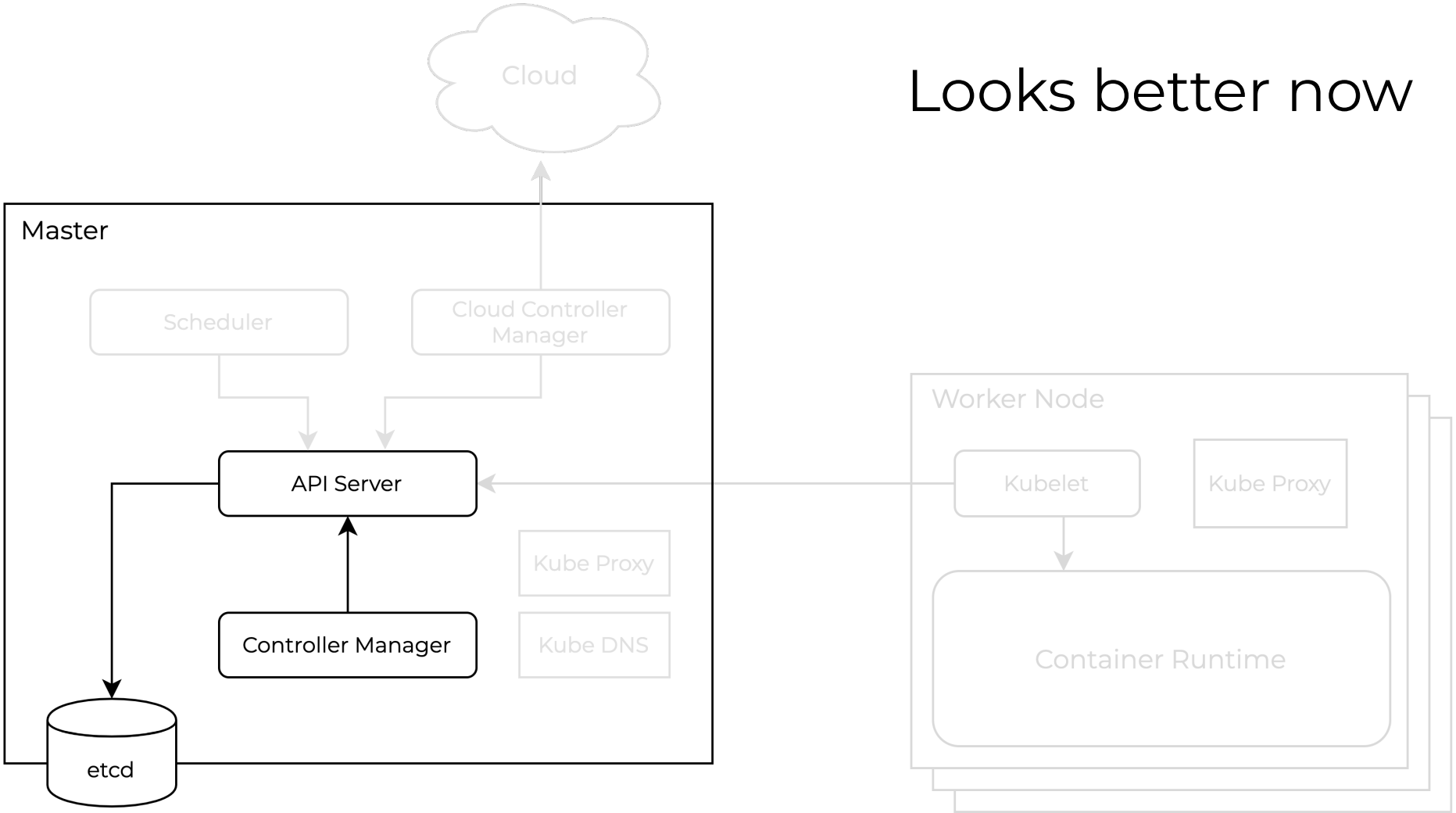
*serviceaccount* *serviceaccount-token*

Enabled Controllers in
**Controller Manager**

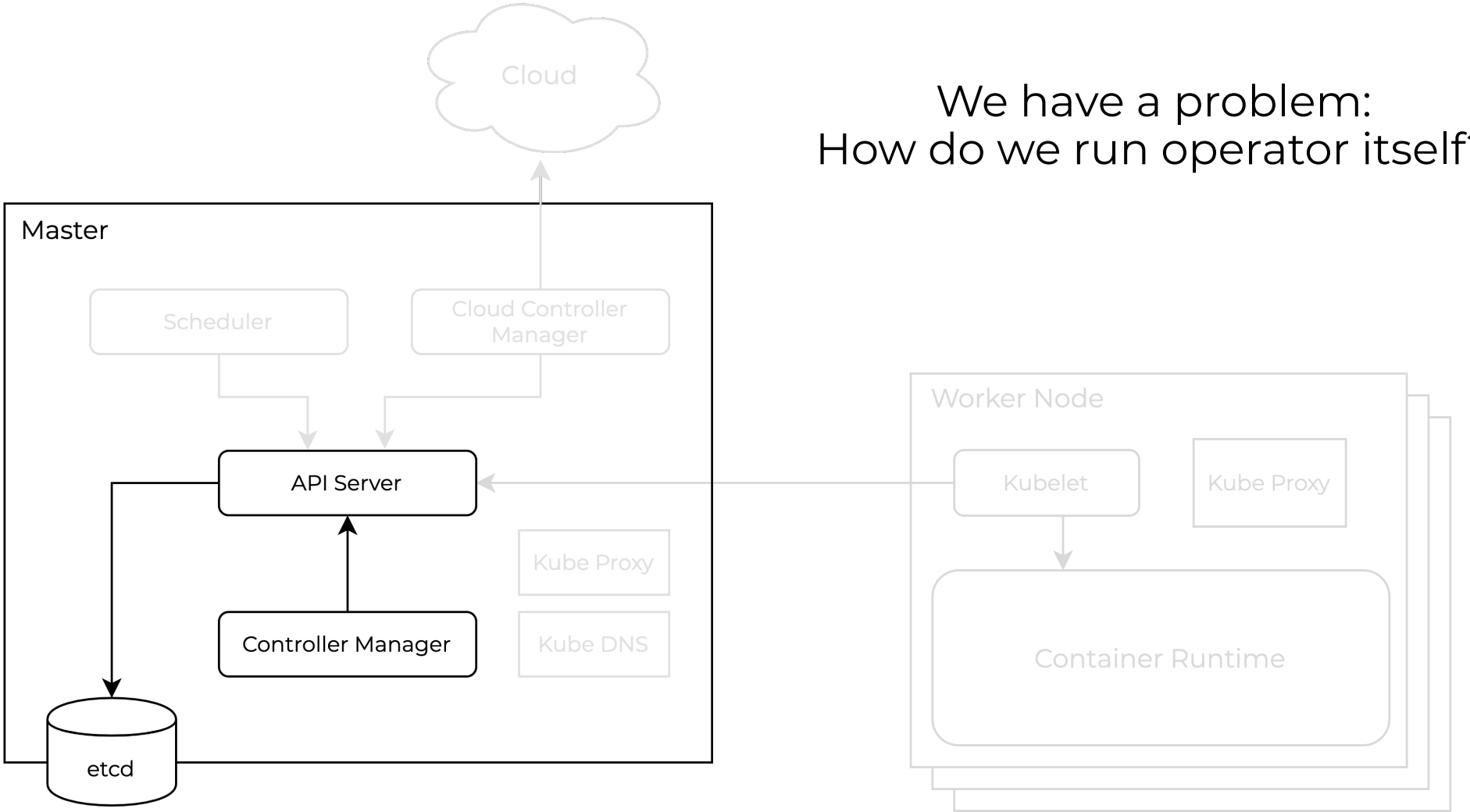
*clusterrole-aggregation* *garbagecollector* *namespace*

*serviceaccount* *serviceaccount-token*

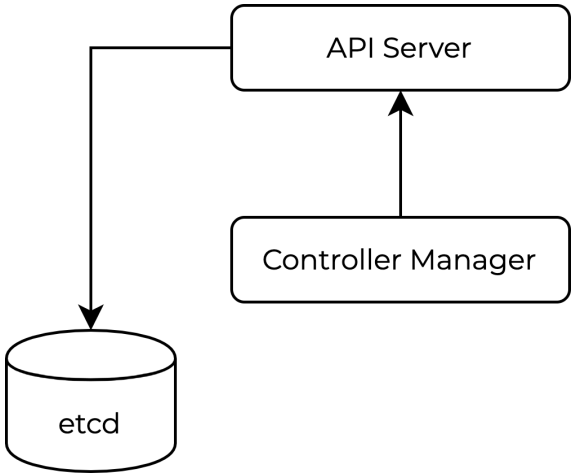
# Looks better now



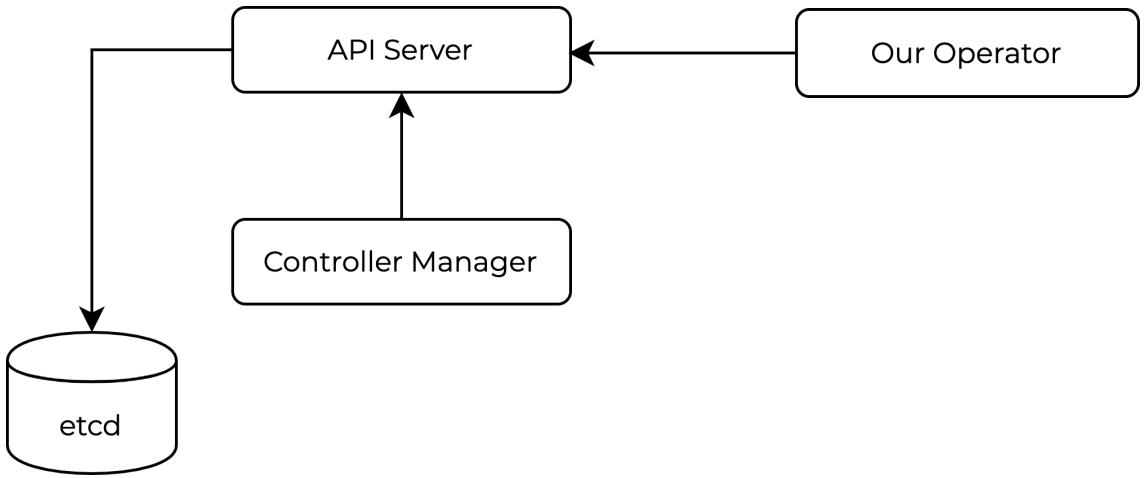
We have a problem:
How do we run operator itself?



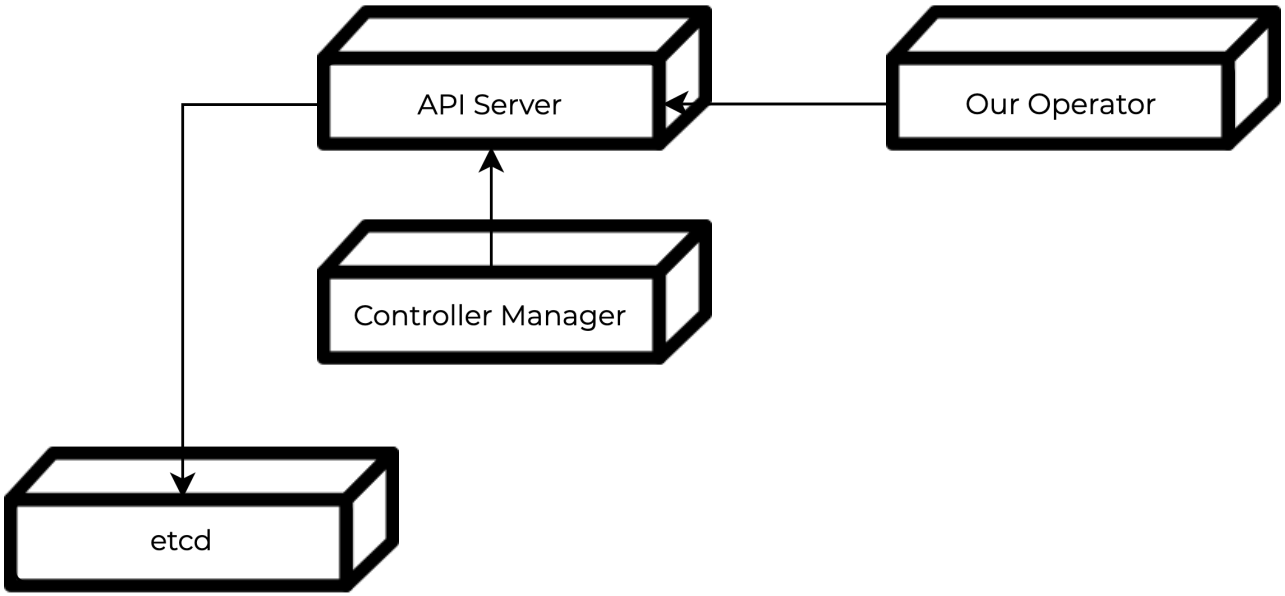
# All We Need



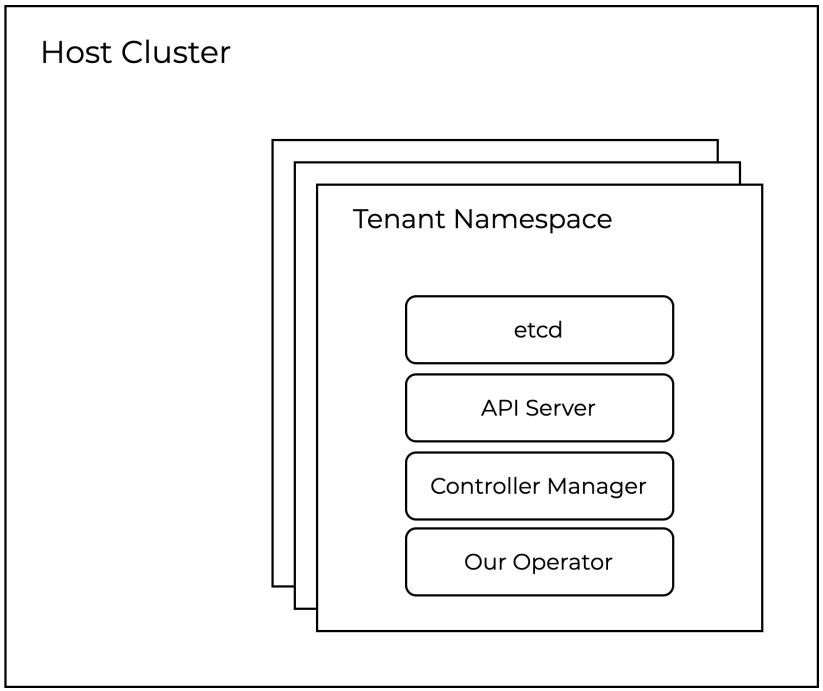
# All We Need

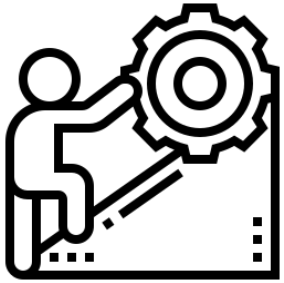


# Let’s put them into containers …



# .... and run inside a Kubernetes cluster

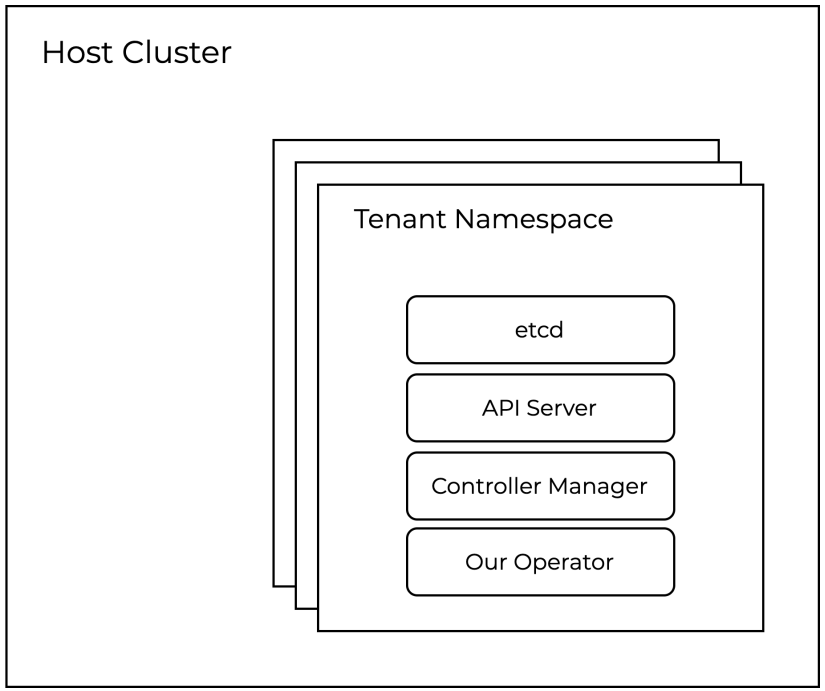




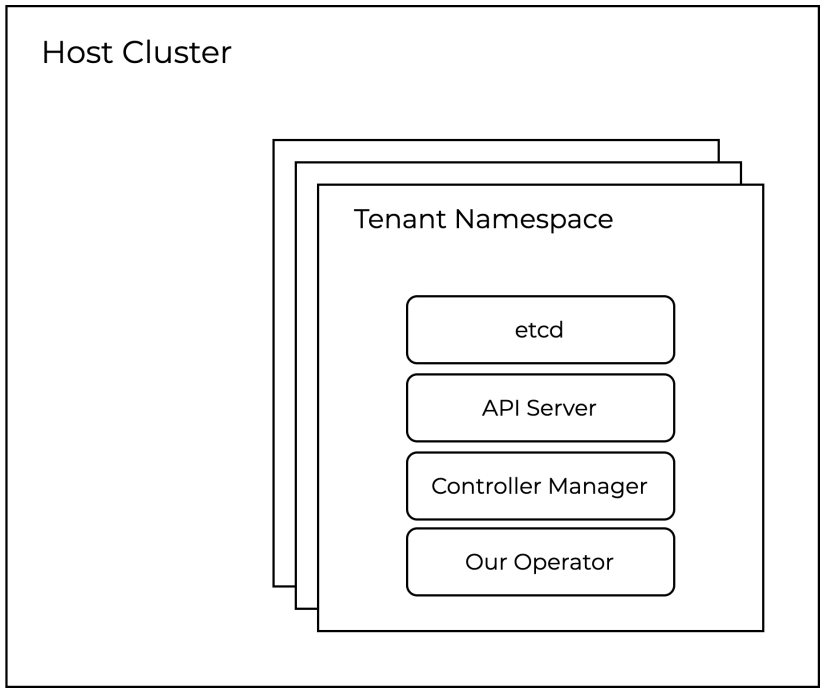
# Challenges

Operators should
run on host k8s
but
watch tenant k8s

# Operator should run on host k8s but watch tenant k8s



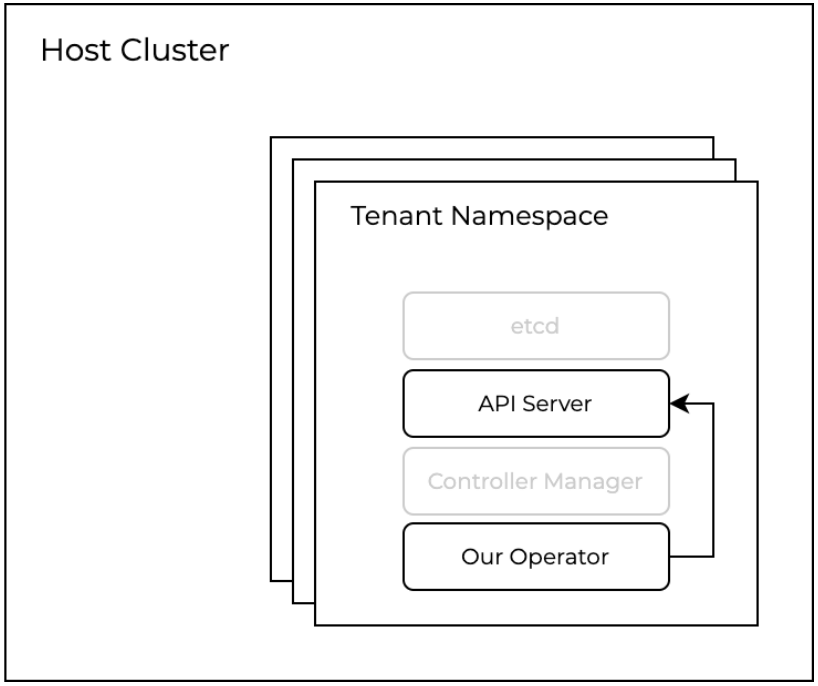
Operator should
run on host k8s
but
watch tenant k8s




**Problems**

- Connectivity
- Authentication / Authorization
- Packaging

Operator should
run on host k8s
but
watch tenant k8s




**Problems**

- Connectivity
- Authentication / Authorization
- Packaging

In cluster Config

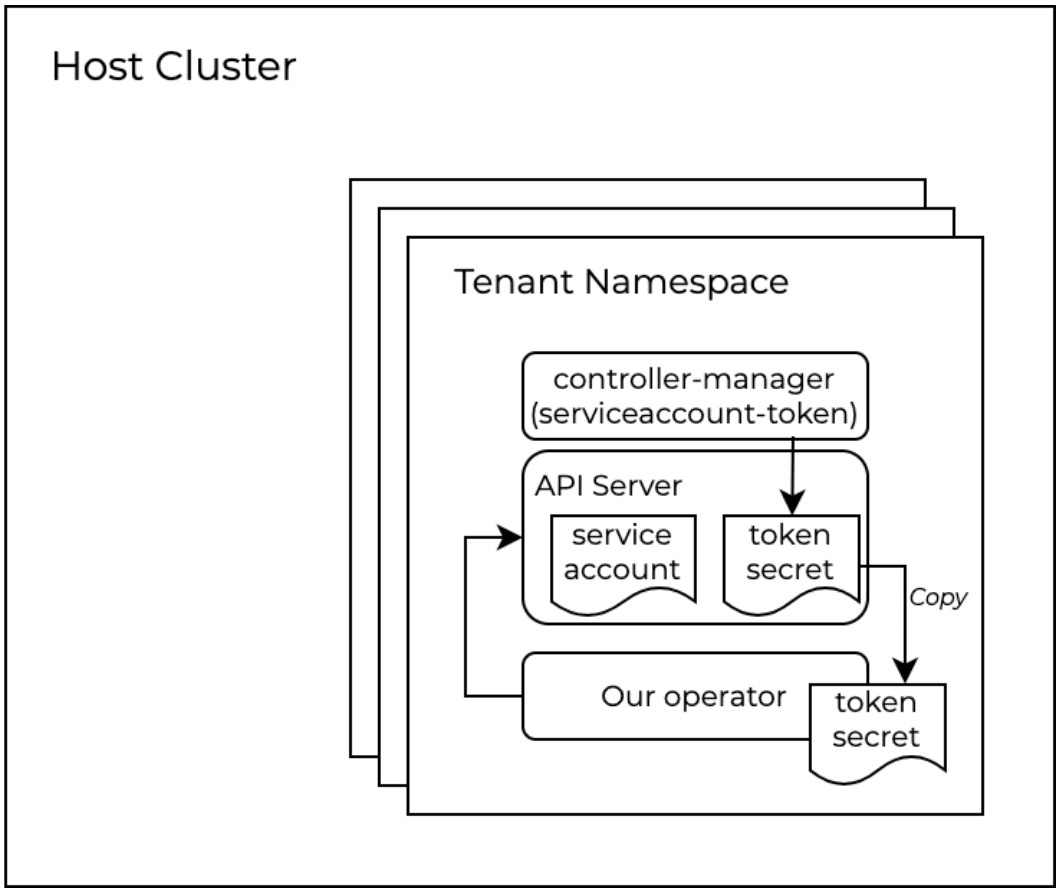
# In cluster Config

- To find kubernetes API endpoint
  - KUBERNETES_SERVICE_HOST
  - KUBERNETES_SERVICE_PORT
- To authenticate/authorize
  - serviceaccount token

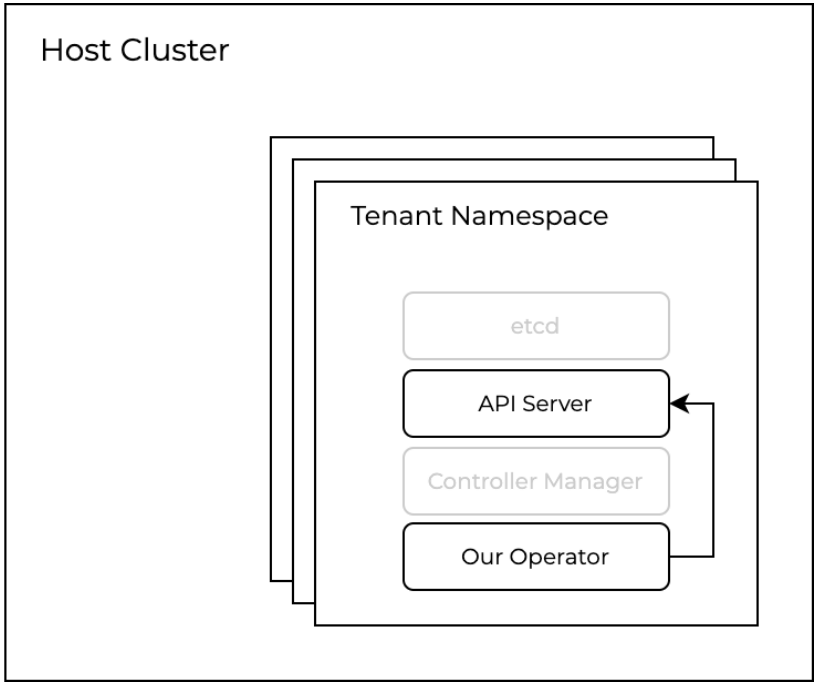
```go
// InClusterConfig returns a config object which uses the service account
// kubernetes gives to pods. It's intended for clients that expect to be
// running inside a pod running on kubernetes. It will return ErrNotInCluster
// if called from a process not running in a kubernetes environment.
func InClusterConfig() (*Config, error) {
        const (
                tokenFile  = "/var/run/secrets/kubernetes.io/serviceaccount/token"
                rootCAFile = "/var/run/secrets/kubernetes.io/serviceaccount/ca.crt"
        )
        host, port := os.Getenv("KUBERNETES_SERVICE_HOST"), os.Getenv("KUBERNETES_SERVICE_PORT")
        if len(host) == 0 || len(port) == 0 {
                return nil, ErrNotInCluster
        }

        token, err := ioutil.ReadFile(tokenFile)
        if err != nil {
                return nil, err
        }

        tlsClientConfig := TLSClientConfig{}

        if _, err := certutil.NewPool(rootCAFile); err != nil {
                klog.Errorf("Expected to load root CA config from %s, but got err: %v", rootCAFile, err)
        } else {
                tlsClientConfig.CAFile = rootCAFile
        }

        return &Config{
                // TODO: switch to using cluster DNS.
                Host:            "https://" + net.JoinHostPort(host, port),
                TLSClientConfig: tlsClientConfig,
                BearerToken:     string(token),
                BearerTokenFile: tokenFile,
        }, nil
}
```

# Service account token for Tenant API Server

1. Deploy *serviceaccount* into tenant API server
2. Tenant controller manager creates a token secret inside tenant API server
3. Copy token secret from tenant API server to host cluster
4. Mount token secret to the operator pod



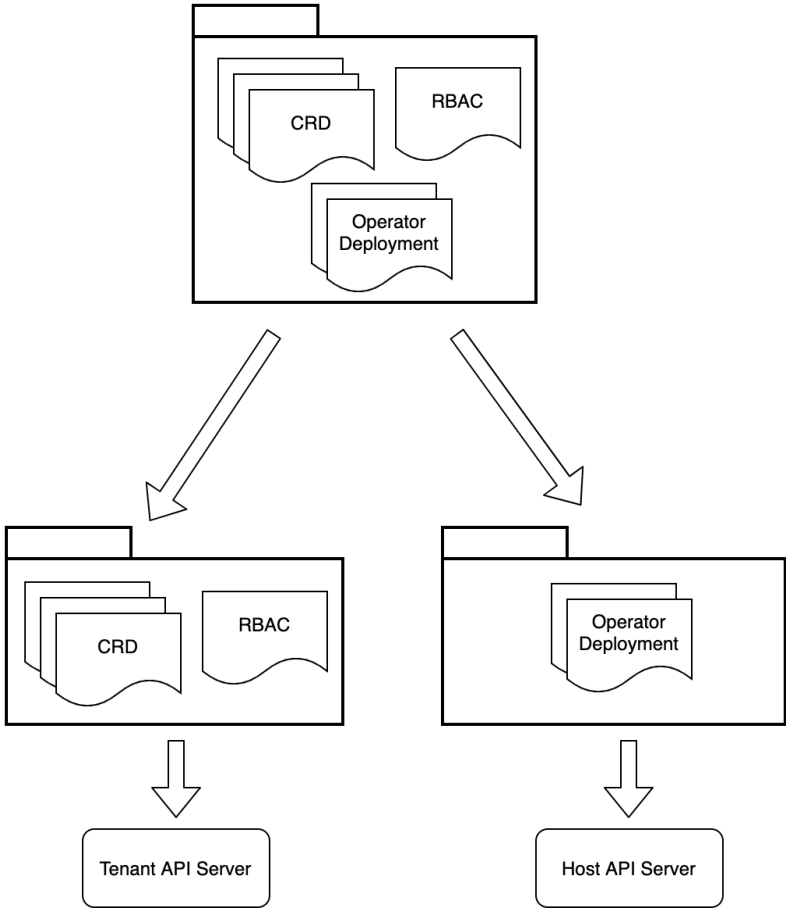
Operator should
run on host k8s
but
watch tenant k8s




## Problems

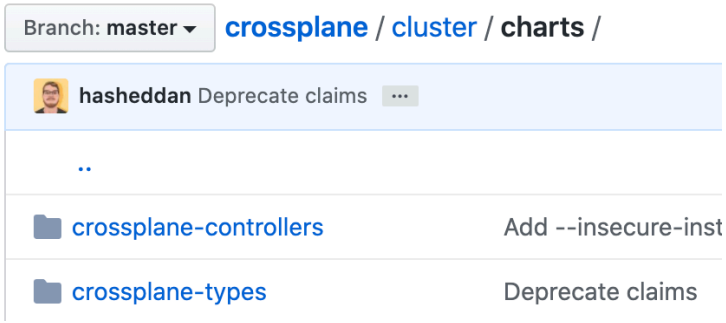
- Connectivity
- Authentication / Authorization
- Packaging

```
{{- if .Values.hostedConfig.enabled }}
  env:
    - name: KUBERNETES_SERVICE_HOST
      value: {{ .Values.hostedConfig.tenantKubernetesServiceHost | quote }}
    - name: KUBERNETES_SERVICE_PORT
      value: {{ .Values.hostedConfig.tenantKubernetesServicePort | quote }}
  volumeMounts:
    - mountPath: /var/run/secrets/kubernetes.io/serviceaccount
      name: sa-token
      readOnly: true
automountServiceAccountToken: false
serviceAccount: ""
serviceAccountName: ""
volumes:
  - name: sa-token
    secret:
      defaultMode: 420
      secretName: {{ .Values.hostedConfig.crossplaneSATokenSecret | quote }}
```




# Problems

- Connectivity
- Authentication / Authorization
- Packaging

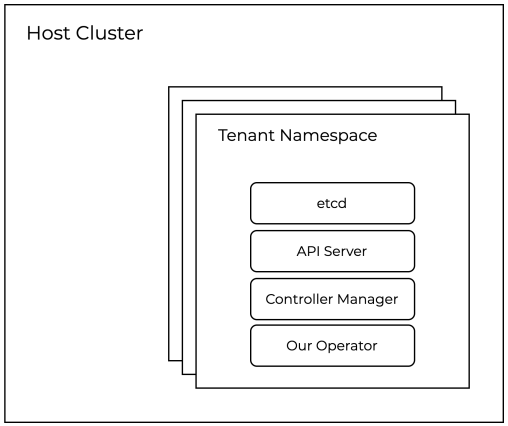



Package controllers and types separately!

# Running etcd for tenants

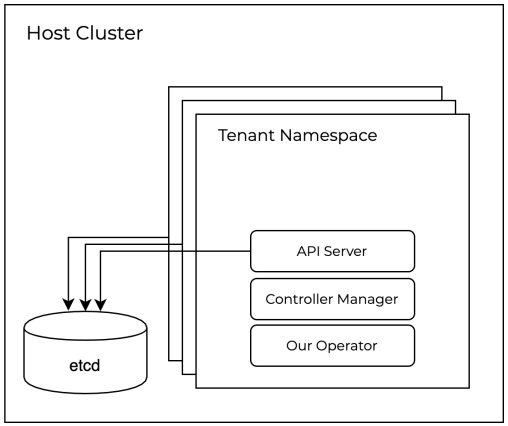
# Running etcd for tenants

## dedicated etcd



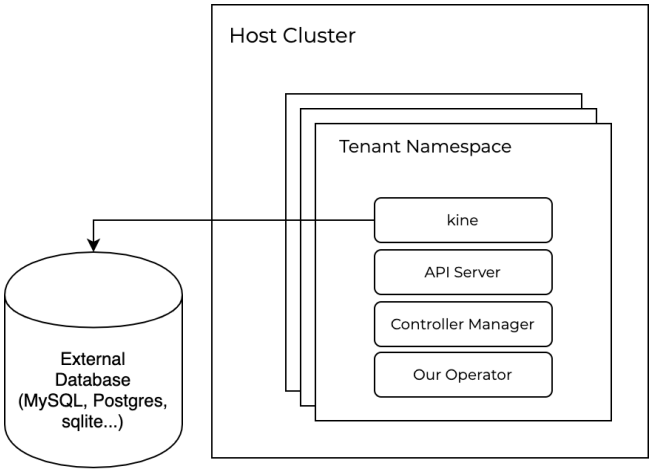

- Higher cost
- Maintain etcd per tenant

## shared etcd




- Noisy Neighbor
- Horizontal scaling problem

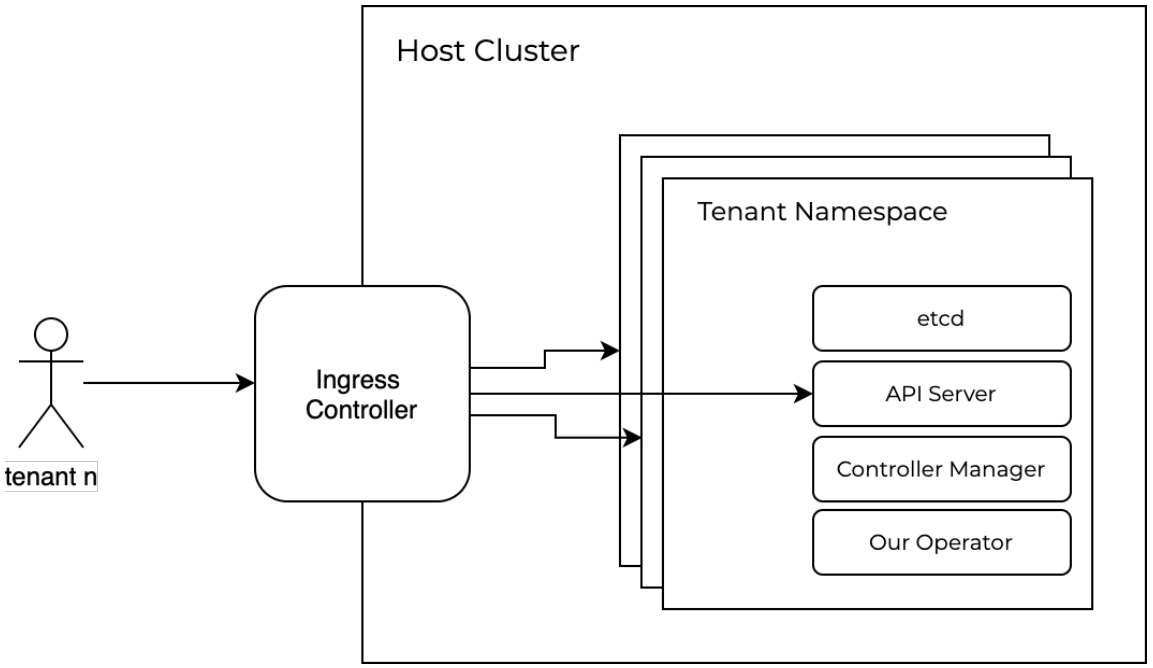
## kine + external db




- Requires external db

# Accessing tenant api-servers

# Accessing tenant api-servers



# Security and Isolation

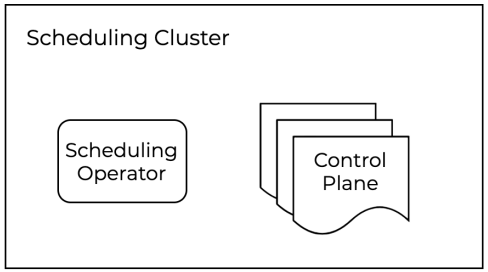
# Security and Isolation

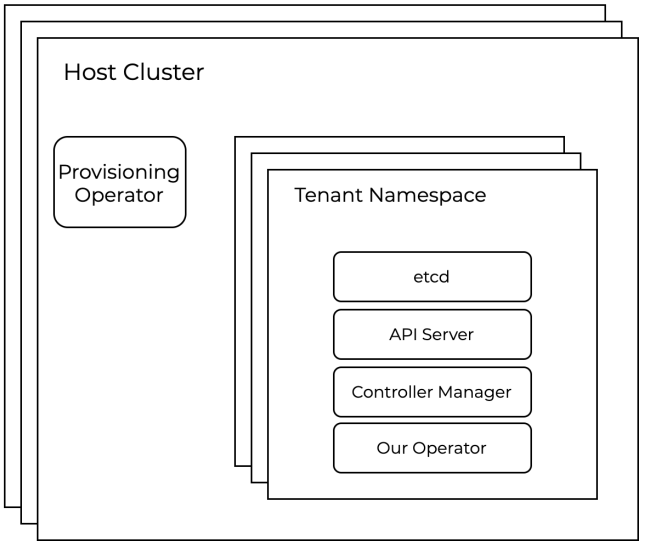
- Thanks to Kubernetes
  - Namespaces
  - Network Policies
  - Pod Security Policies
  - LimitRanges
  - ResourceQuota
- Sandboxed with gVisor
- mTLS between components

# Scheduling and Provisioning

# Scheduling and Provisioning

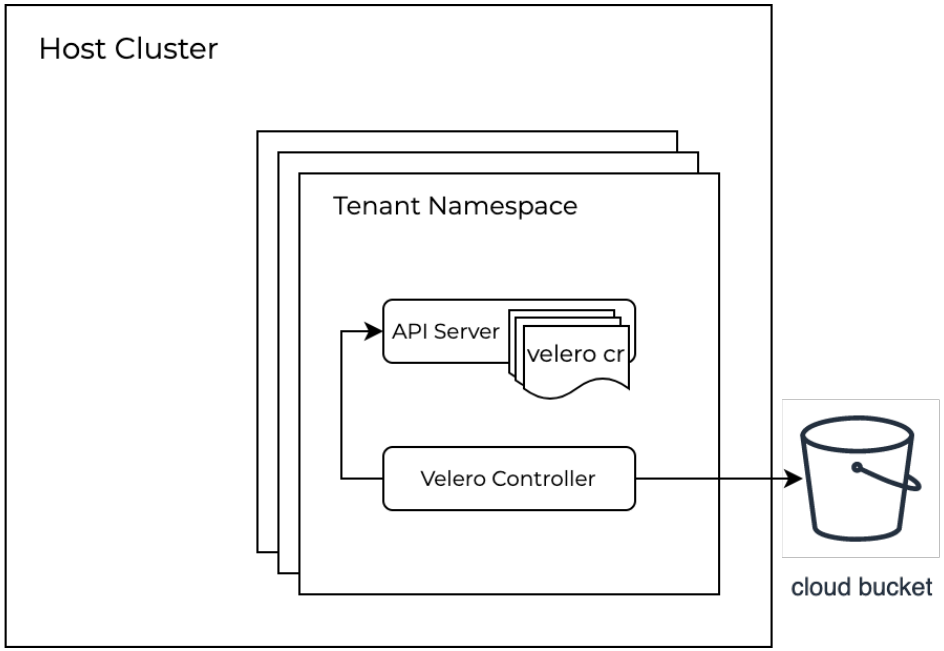
- Scalable to millions of control planes
- Need multiple host clusters
- Using Crossplane to provision infrastructure
- Automated with Kubernetes operators!



# Backups and Migrations

# Backing up Tenants

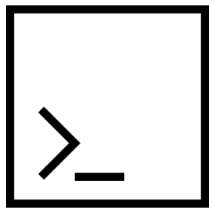



## Backup using velero

- Configure against tenant k8s API
- Enables migration between host clusters

# Future Work

- Evaluate Kine in Production
- Single control plane binary
  - Crossplane + K8S API Server + Controller Manager
- Crossplane Composition for Provisioning Infra + Application



# Demo