Implementing SaaS on Kubernetes

Multi-Tenancy and Tenant Isolation on Kubernetes

Michael Knapp
Senior Software Engineer
October 11, 2018
Certified Kubernetes Administrator

Andrew Gao
Software Engineer
October 11, 2018

Goals

- Understand how “Software as a Service” products can be architected on Kubernetes.

Pre-Requisites

- Have a basic understanding of restful web APIs.
- **Preferred**: basic knowledge of Kubernetes:
  - Namespaces
  - Pods
  - Deployments
  - Services
  - Volumes
  - Config Maps
  - Ingress

Agenda

- Kubernetes Review
- Kubernetes Tools for Isolation
- Tools for distributed applications in Kubernetes
- Architecture of SaaS in Kubernetes

Problem

- Assumption: Your team is running a Kubernetes cluster
- Problem: External teams or people must collaborate with your team to run their software on your platform.
- Examples:
  - Add a Flink application to a Flink cluster
  - Provision apache NiFi instances on demand
  - Create a new Flink cluster
  - Create a custom database
  - Score events with a machine learning model

SaaS?

- Software as a Service
- At a user's request, we deploy a software application and make it available to them.
- Examples:
  - RDS
  - DynamoDB
  - Elasticache
  - SQS
  - SNS

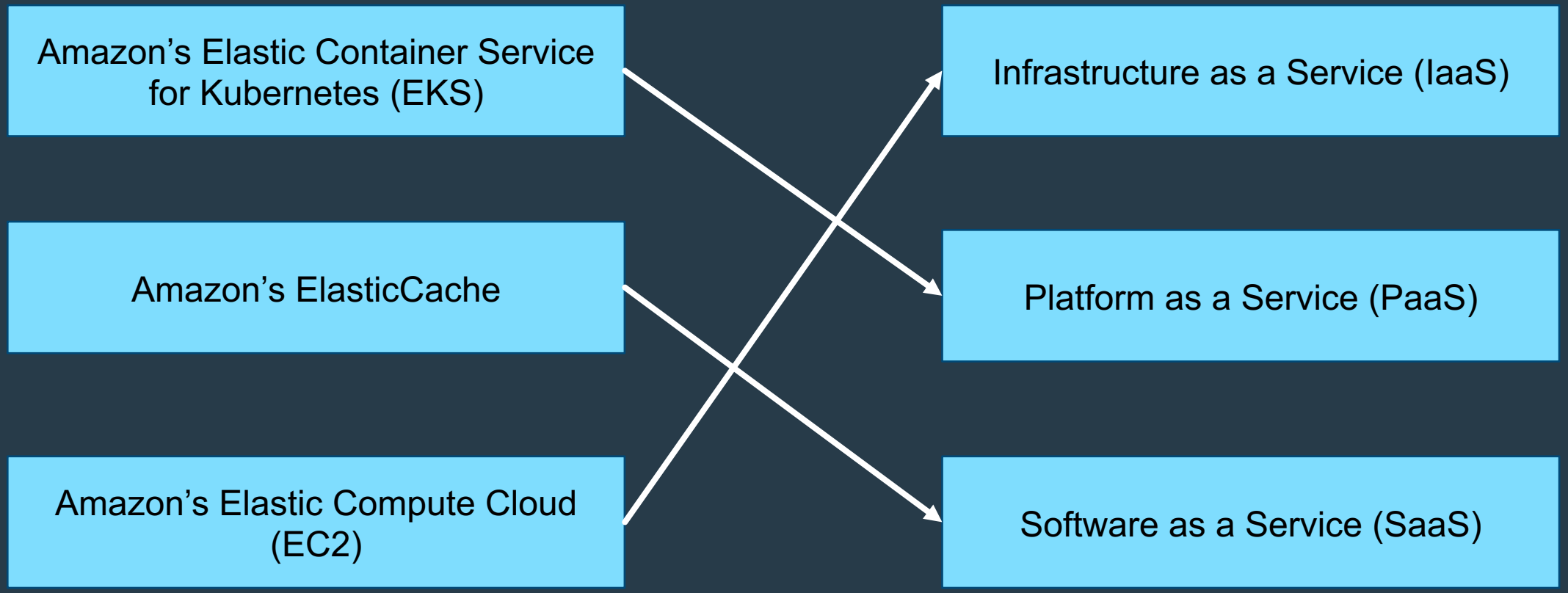
Challenge

Match these up:

Amazon’s Elastic Container Service for Kubernetes (EKS)	Infrastructure as a Service (IaaS)
Amazon’s ElasticCache	Platform as a Service (PaaS)
Amazon’s Elastic Compute Cloud (EC2)	Software as a Service (SaaS)

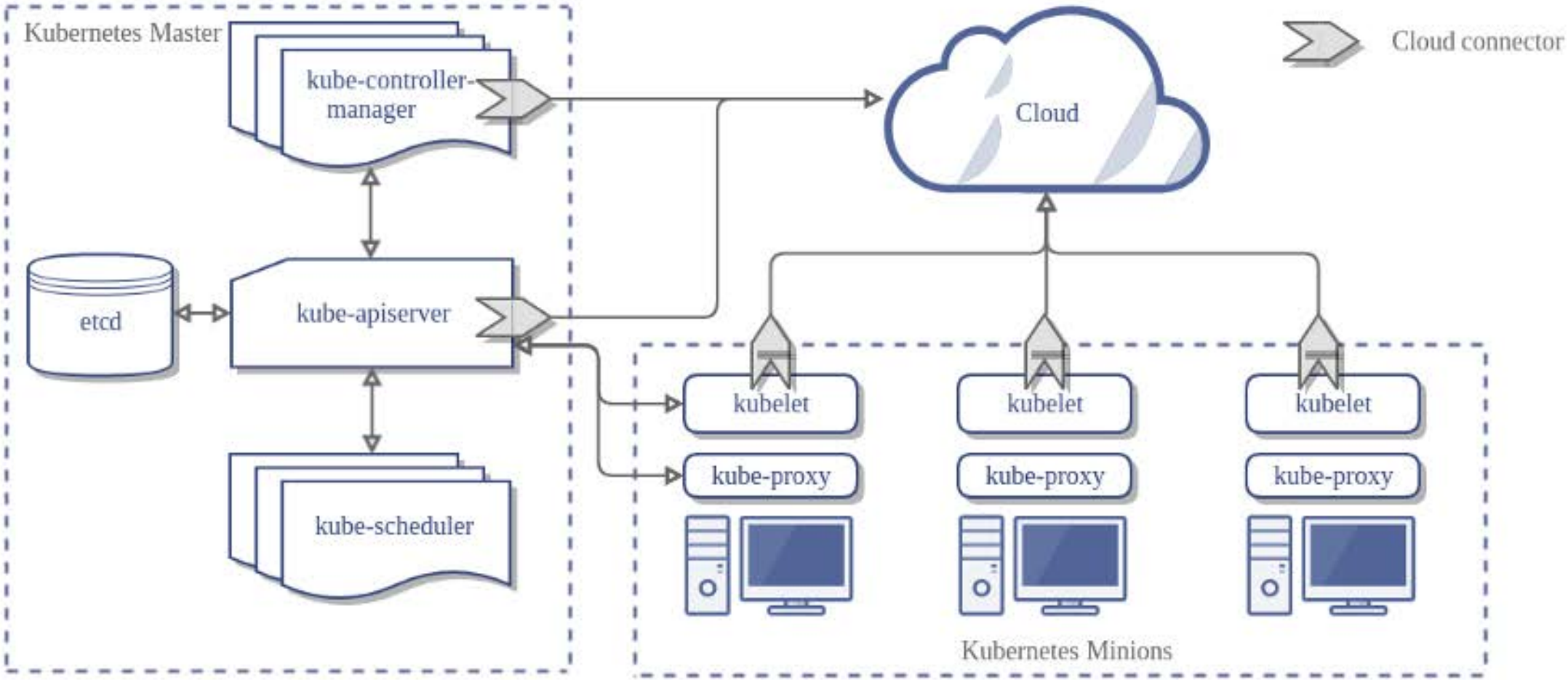
Challenge

Match these up:

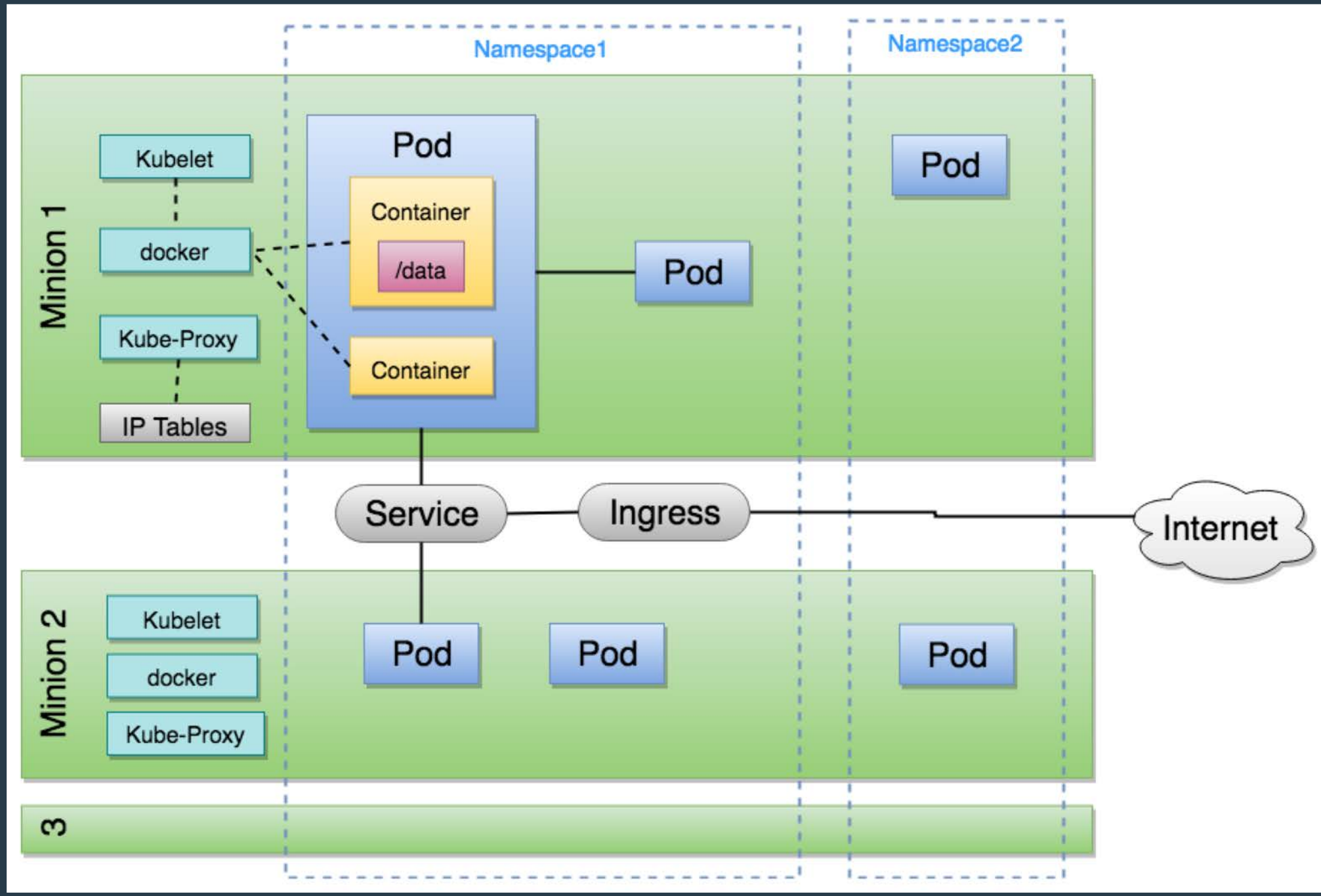


Brief Kubernetes Review

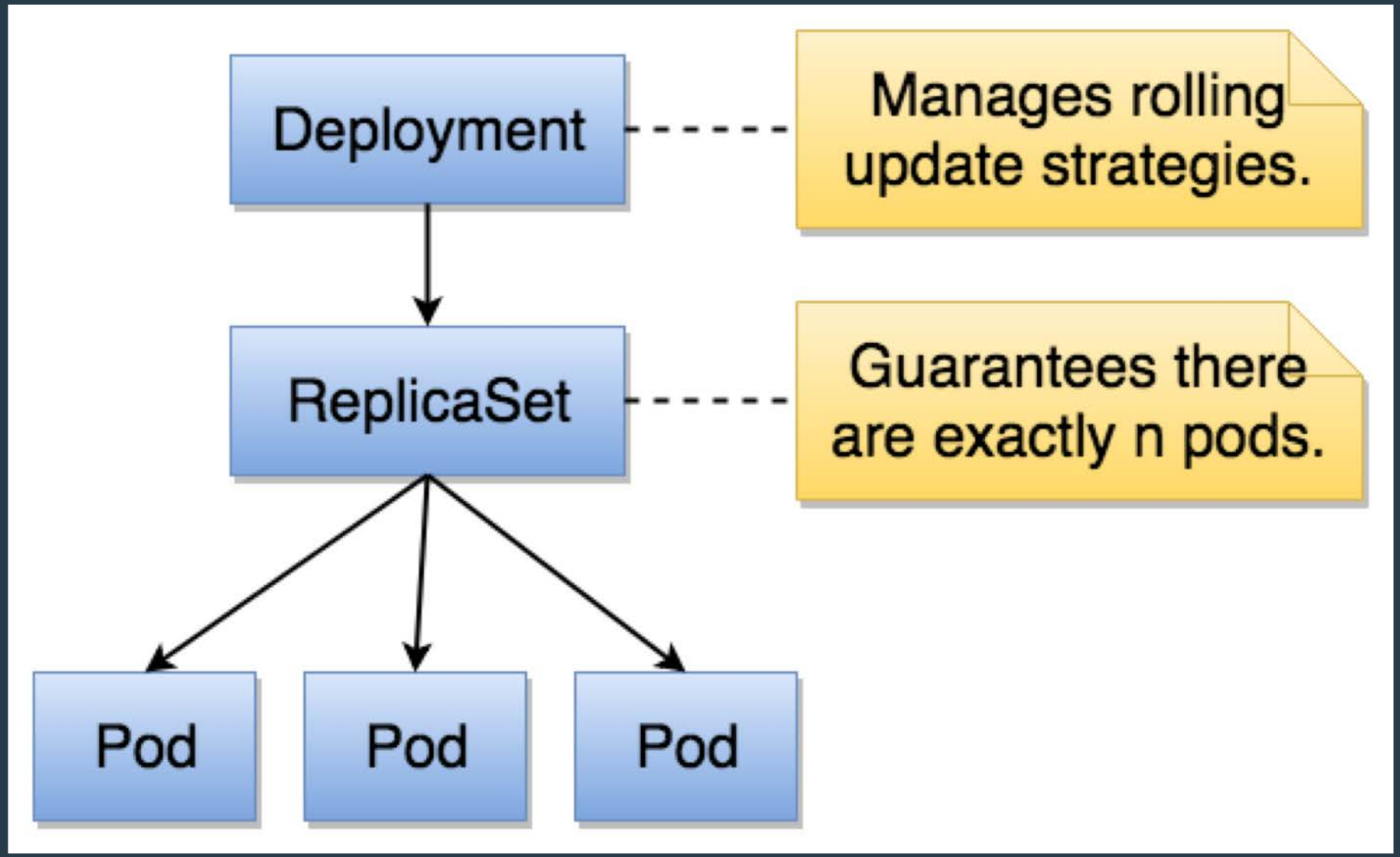
Kubernetes Architecture



https://kubernetes.io/docs/concepts/architecture/cloud-controller/

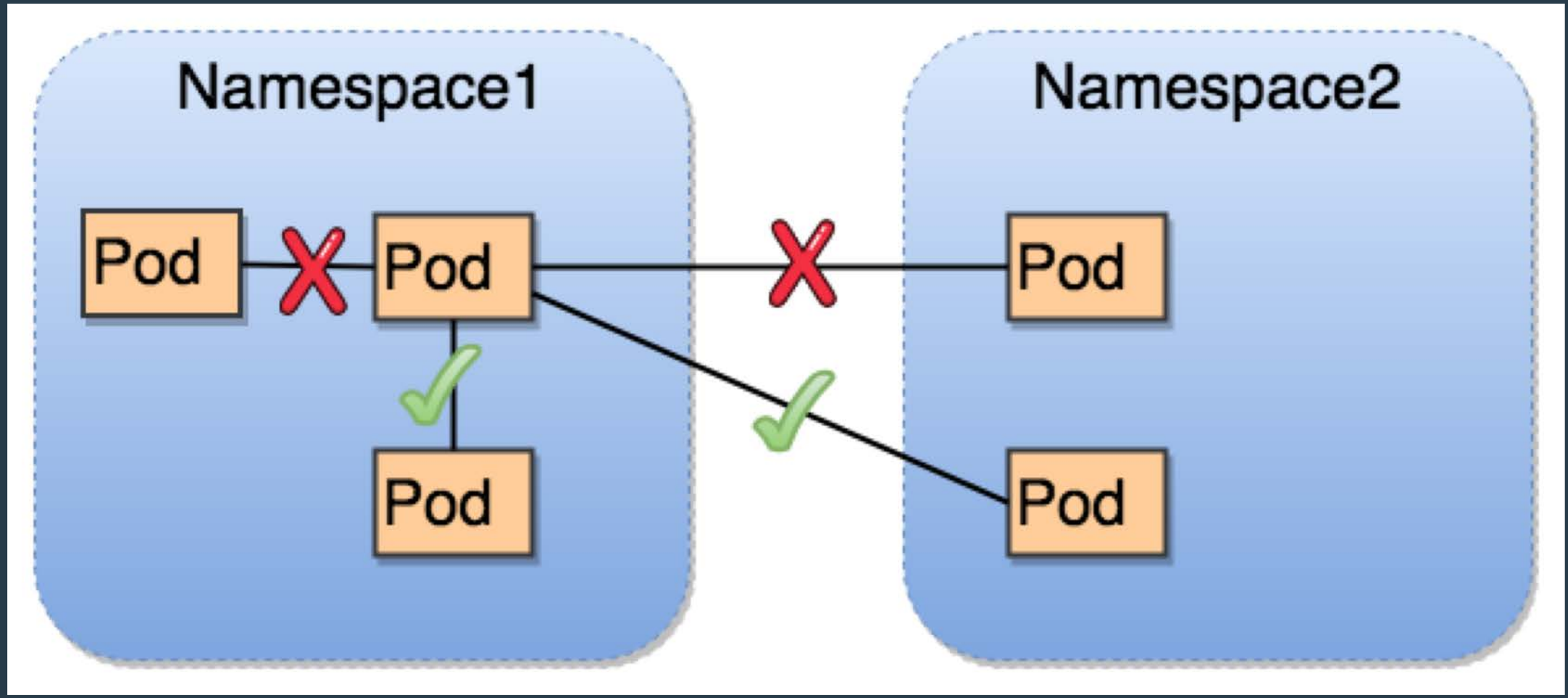


Deployment

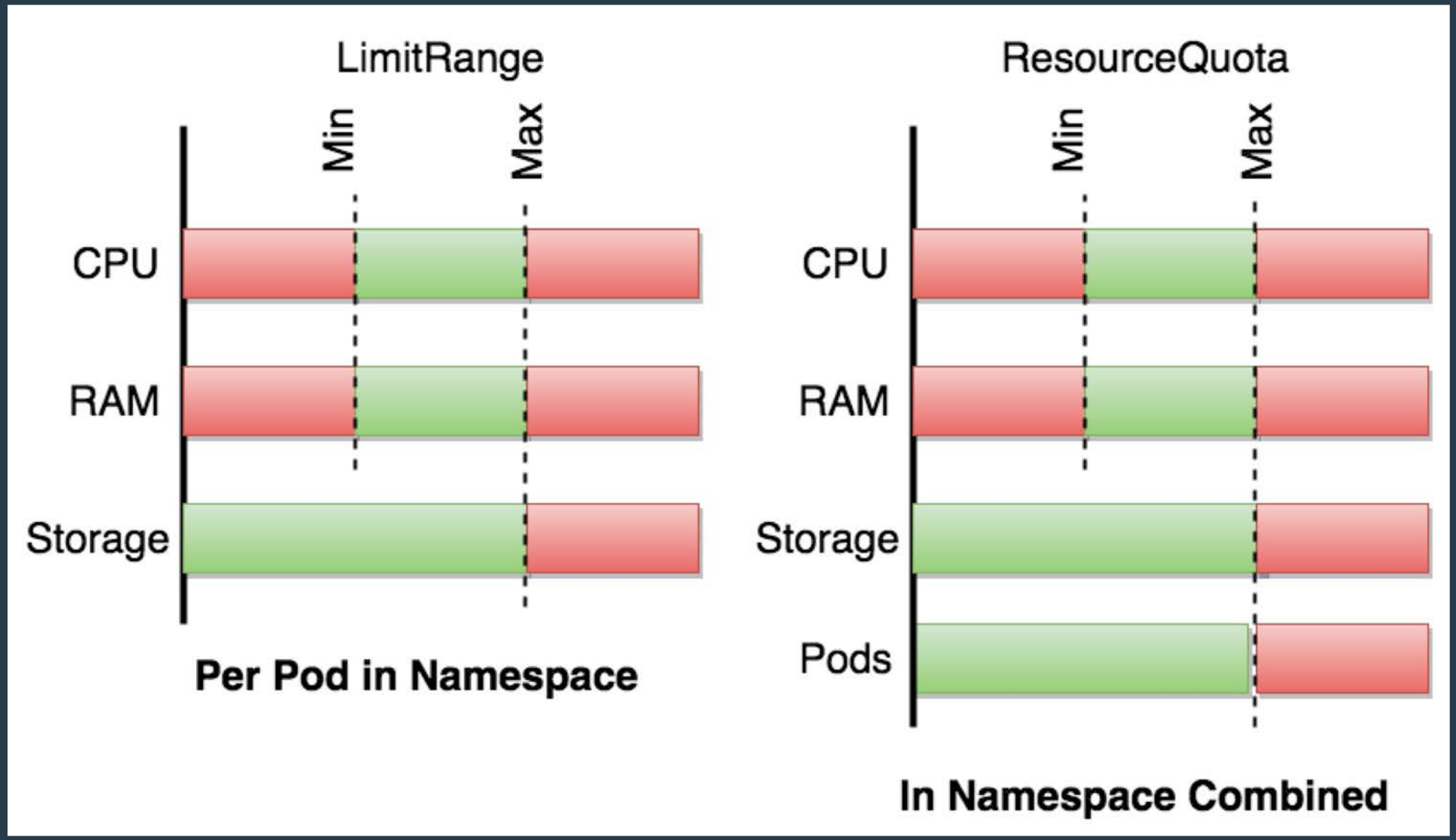


Kubernetes Tools for Isolation

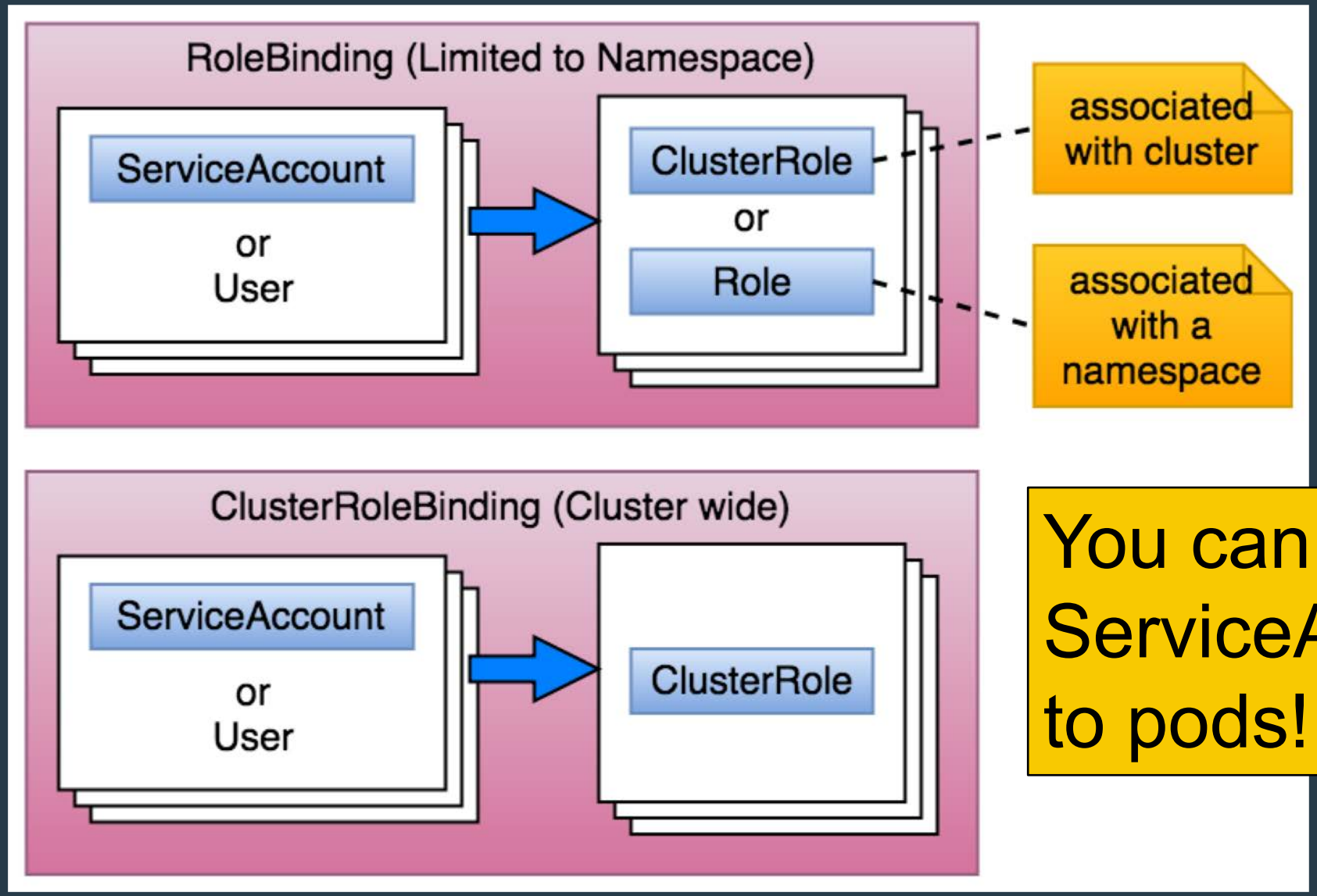
NetworkPolicy



Constraining Resources

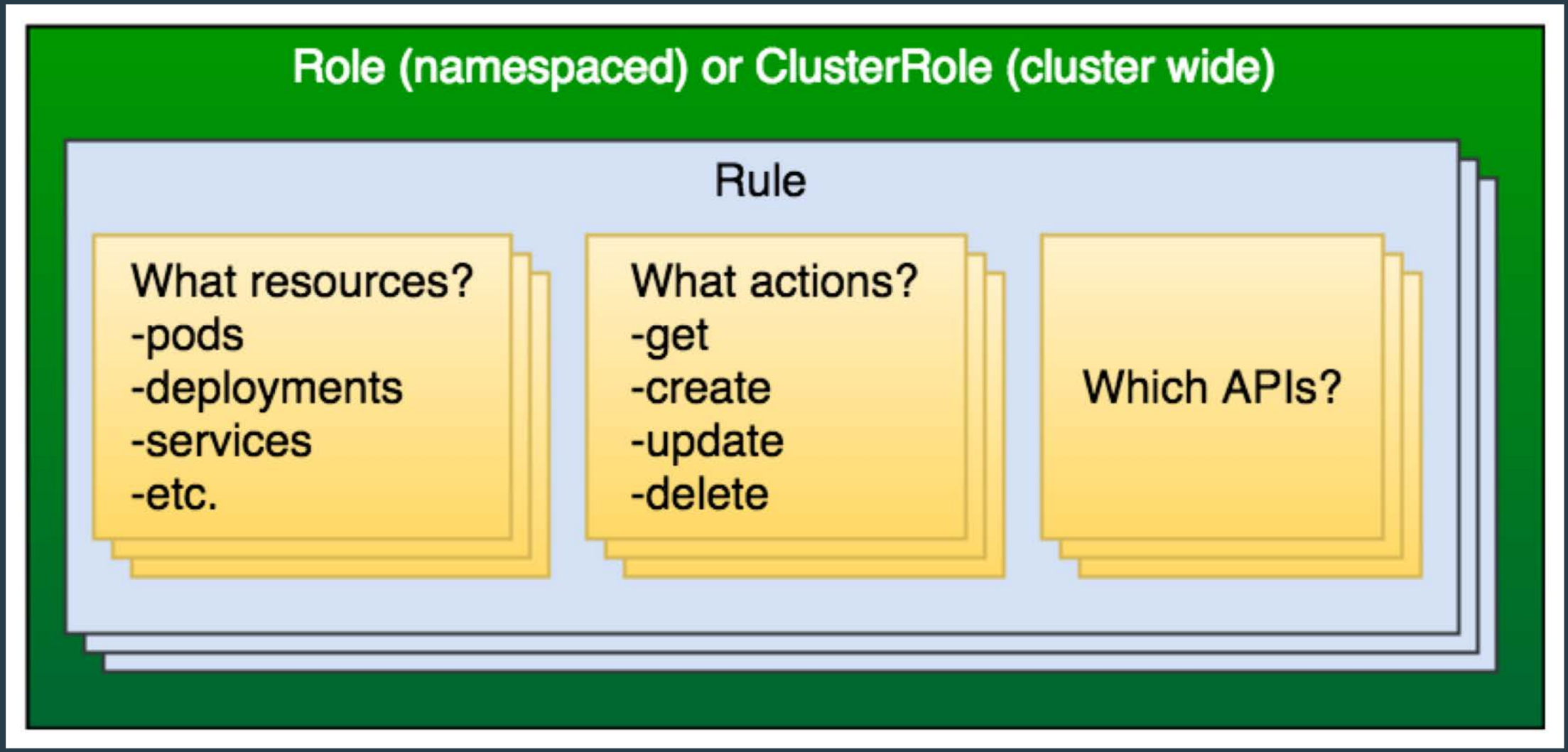


Role Based Access Control



You can assign ServiceAccounts to pods!

Role Based Access Control



Pop Quiz

What can we leverage to prevent tenants from hogging all the RAM in our cluster?

a) Roles, RoleBinding, RBAC
b) NetworkPolicy
c) ResourceQuota
d) LimitRange

Pop Quiz

What can we leverage to prevent tenants from hogging all the RAM in our cluster?

~~a) Roles, RoleBinding, RBAC~~

~~b) NetworkPolicy~~

c) ResourceQuota

~~d) LimitRange~~

A LimitRange may constrain the RAM usage of a single pod, but it cannot limit the total number of pods. A ResourceQuota can.

Kubernetes Tools for Distributed Applications

ETCD for your clustered deployment

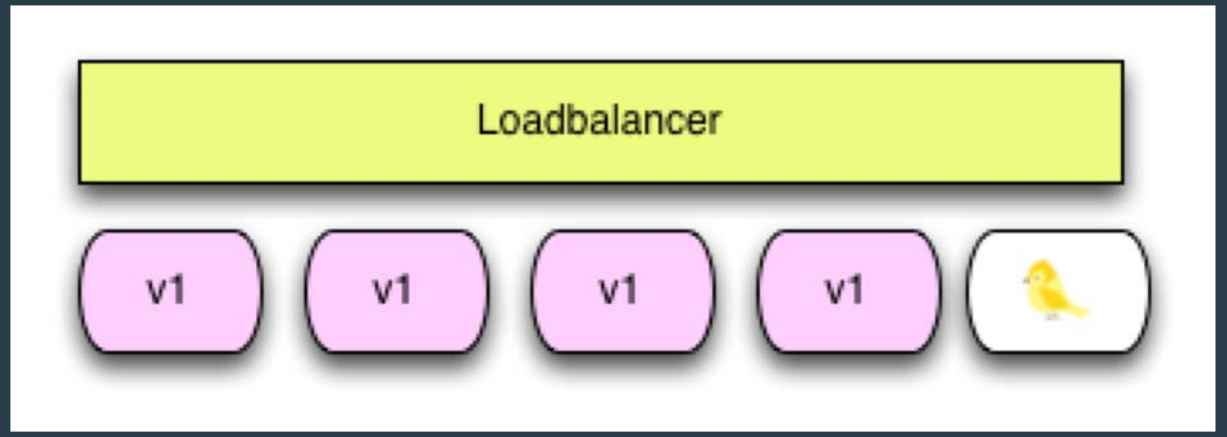
- Overview
  - Open-source key value store
  - Built for clusters
  - Backbone of K8s

- Advantages
  - Automated restore from backup upon cluster node failure
  - Use etcd revision watchers for ordered/reliable/atomic event streams
  - Out-of-the-box leader election

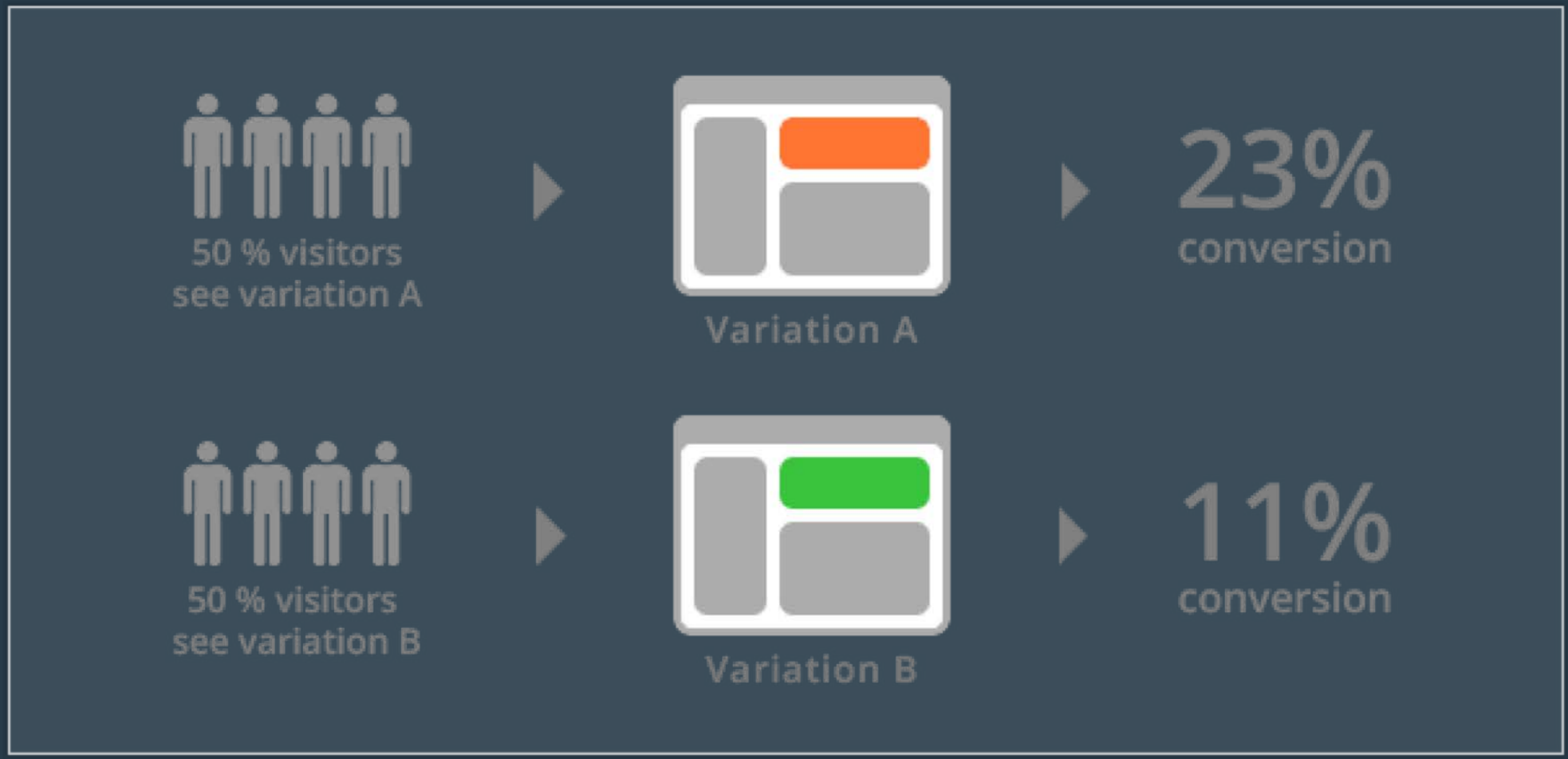
Making a Mesh with Istio

- Overview
  - Service mesh
  - Load Balancing
  - Metrics

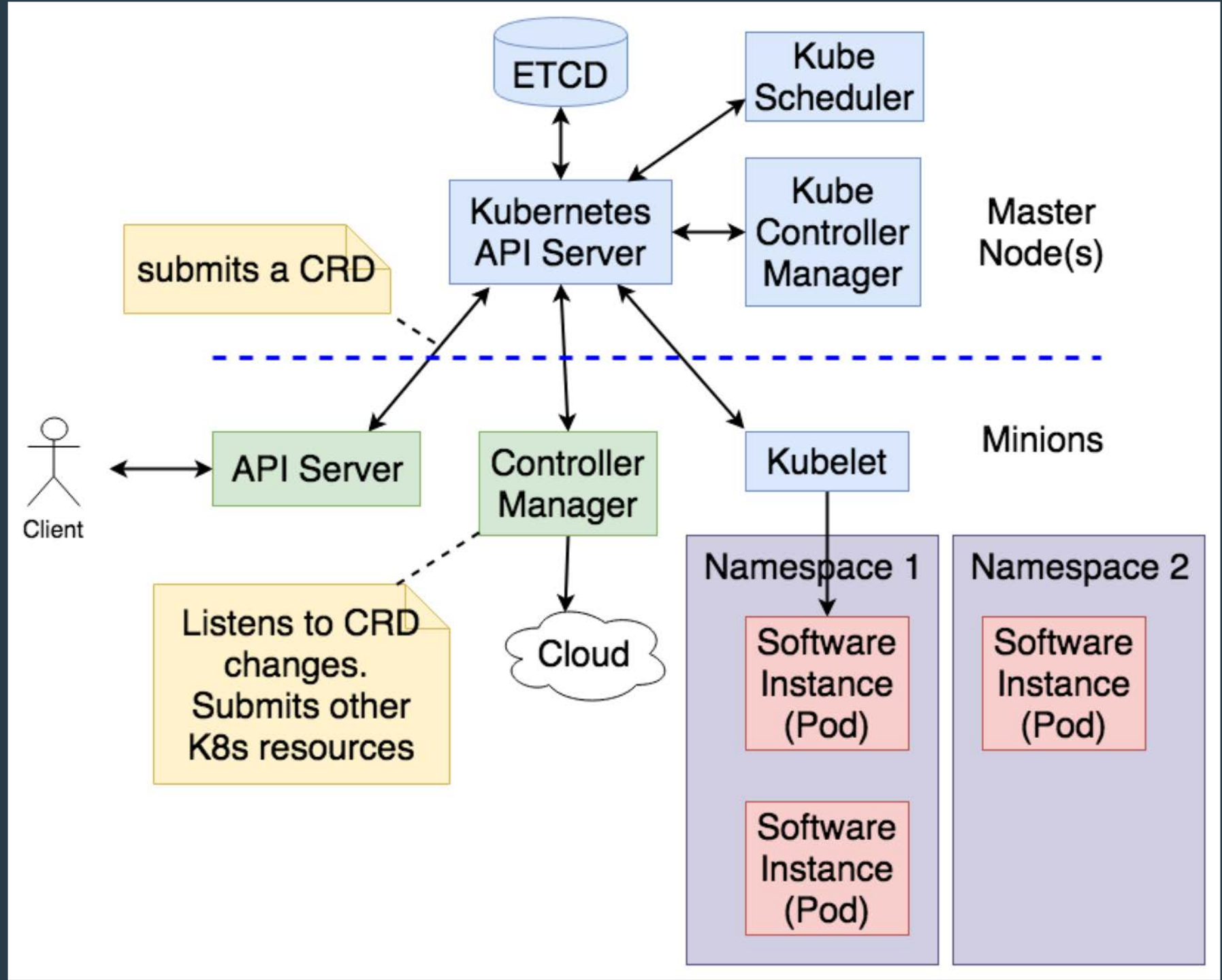
- Advantages
  - Discovery
  - Rate Limiting
  - Canary Releases
  - A/B testing



A/B Testing


50 % visitors
see variation A
Variation A
23%
conversion
50 % visitors
see variation B
Variation B
11%
conversion

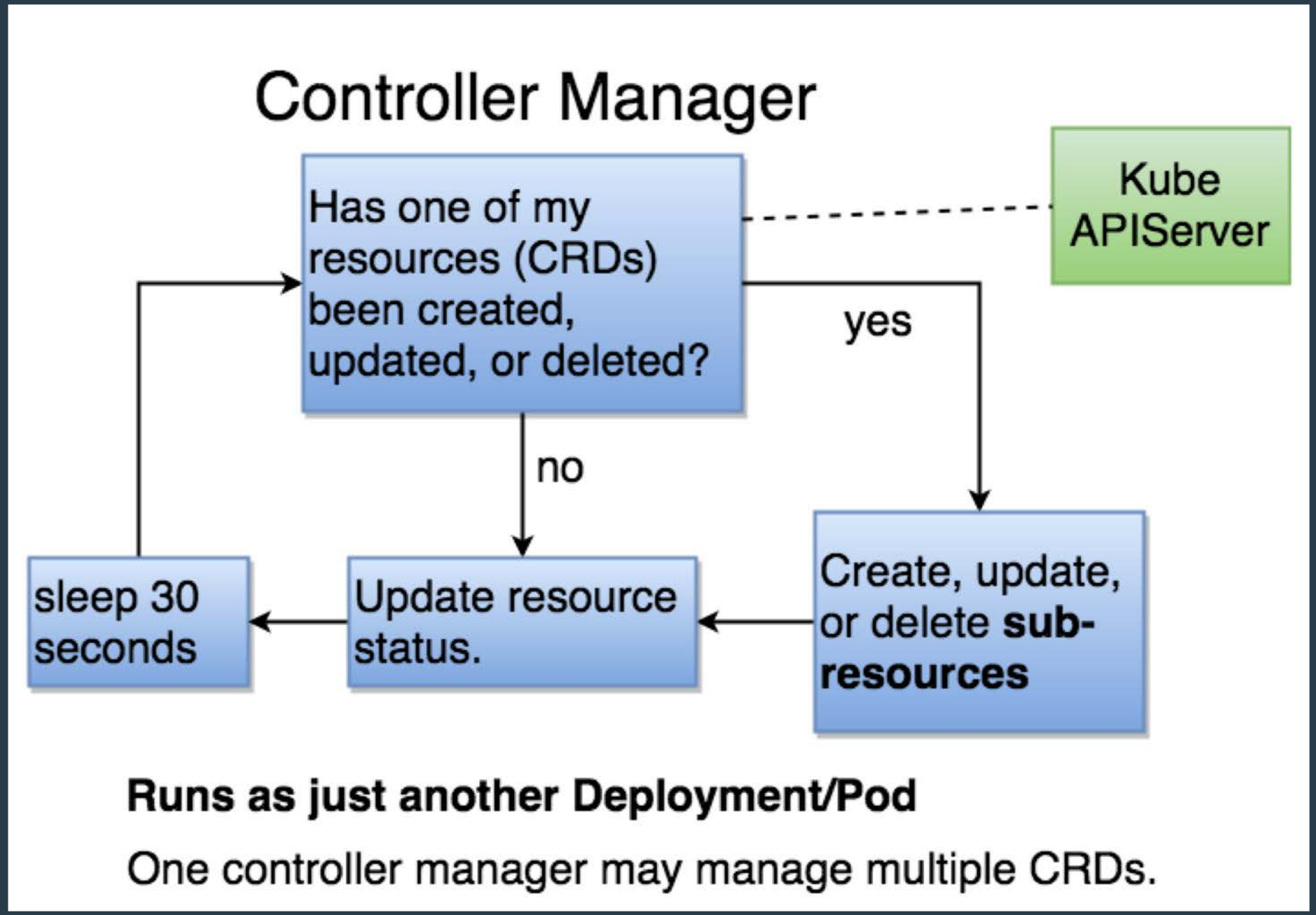
Kubernetes Software as a Service



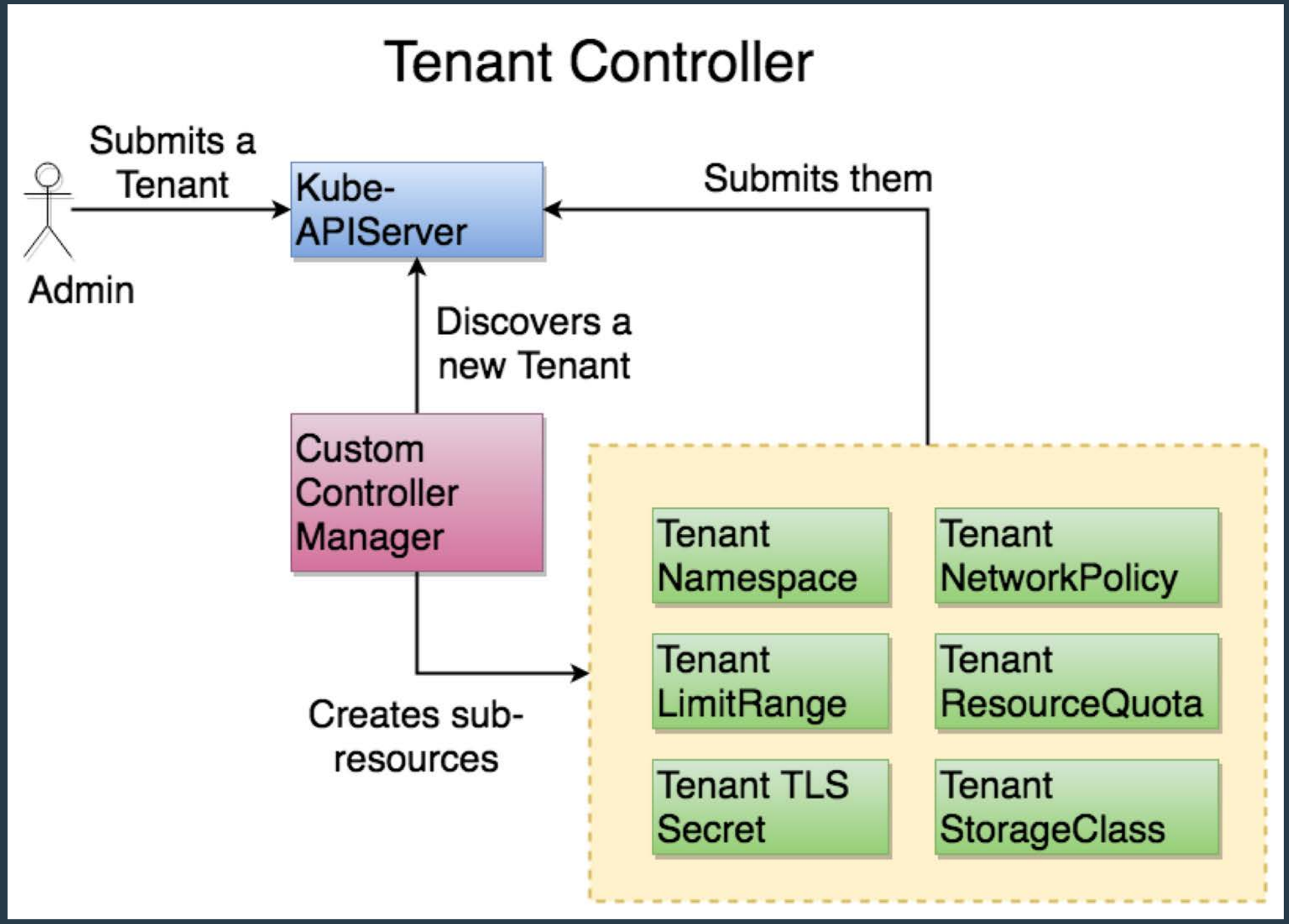
Custom Resource Definition (CRD)

- Defines a nomenclature for an object.
- Does NOT define fields that it has!
- The controller-manager dictates what fields it has.

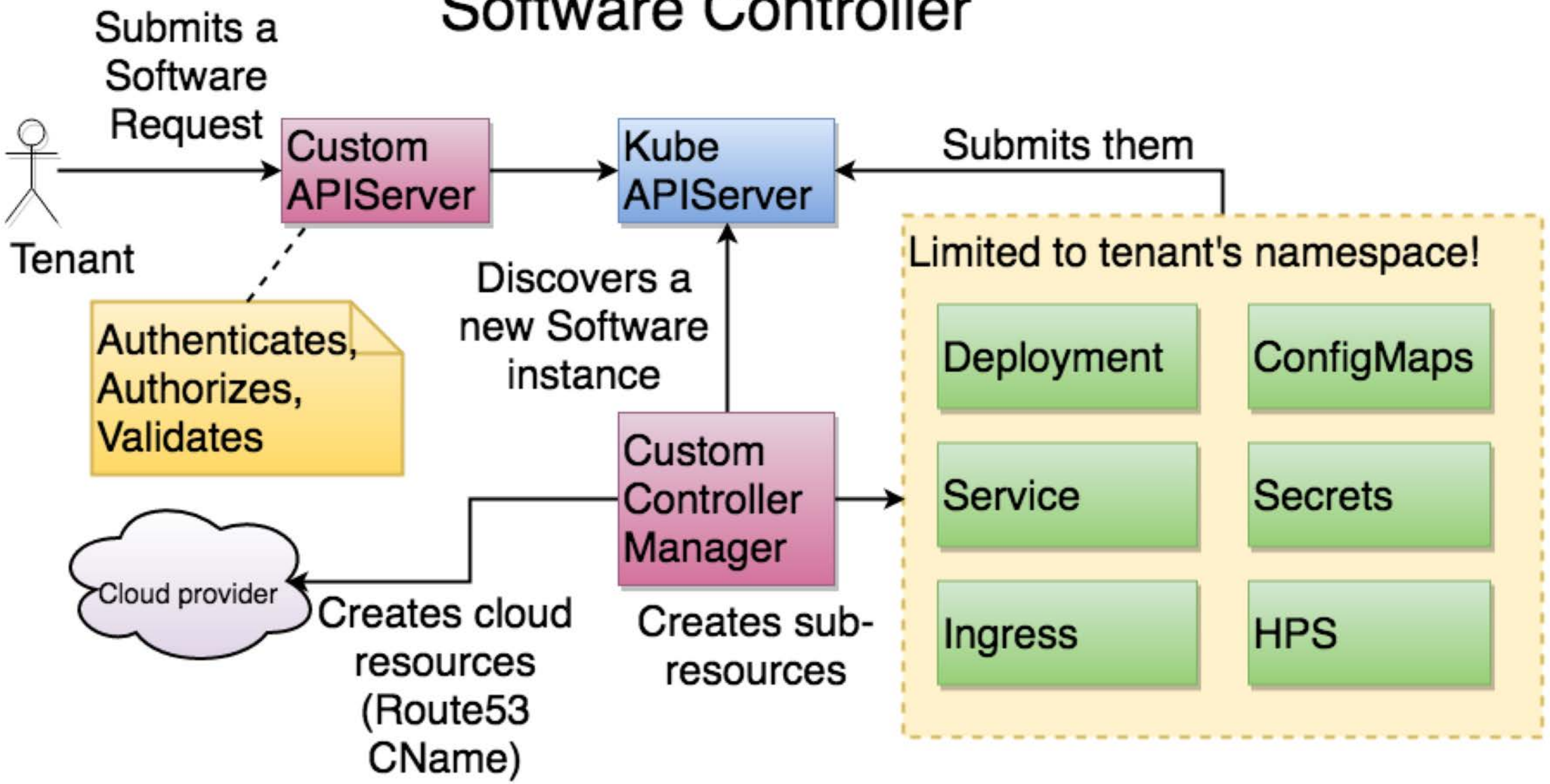
```
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
  name: tenants.example.com
spec:
  group: example.com
  version: v1
  scope: Cluster
  names:
    plural: tenants
    singular: tenant
    kind: Tenant
    shortNames:
    - tnt
```

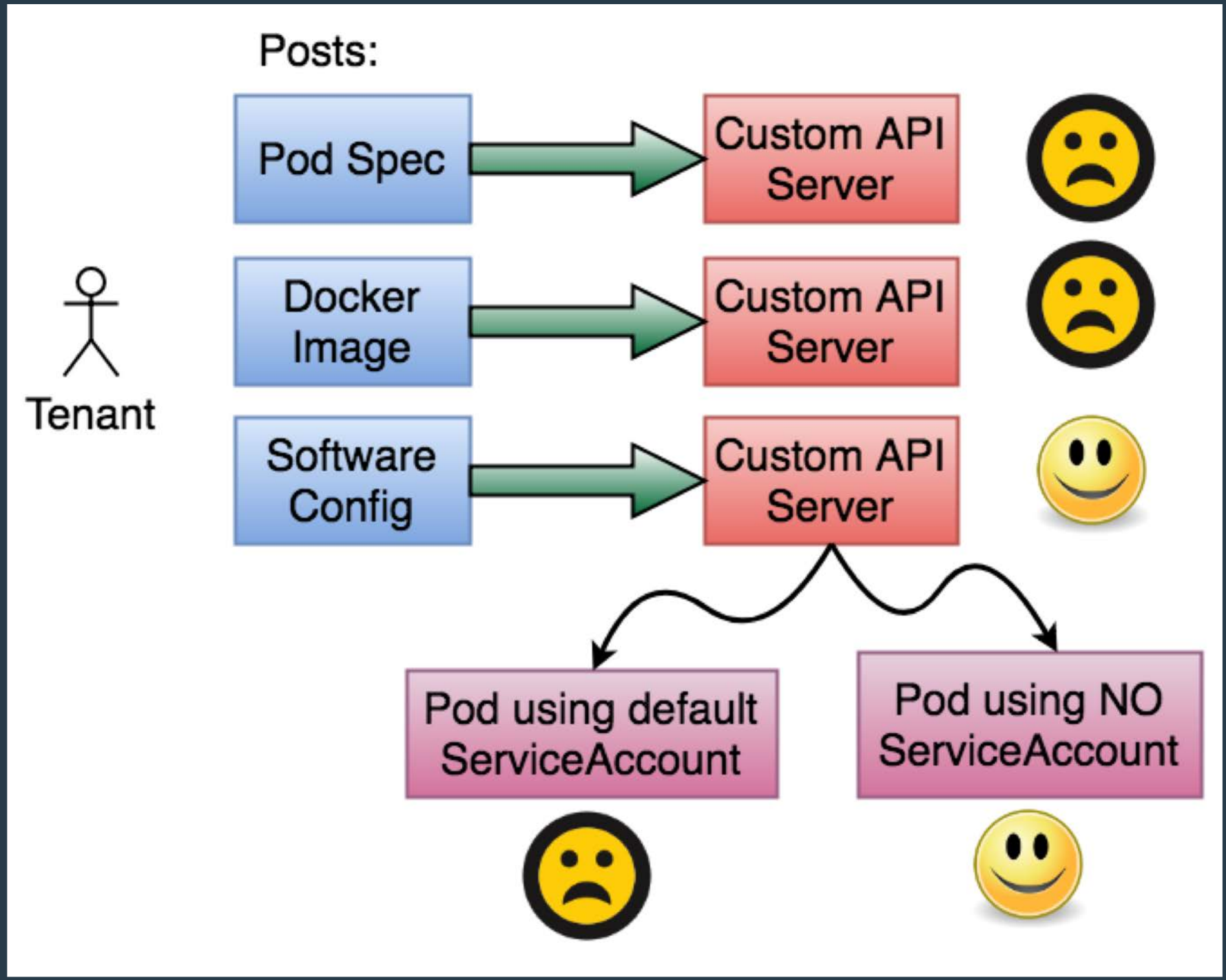


Follow this example: https://github.com/kubernetes/sample-controller



Software Controller

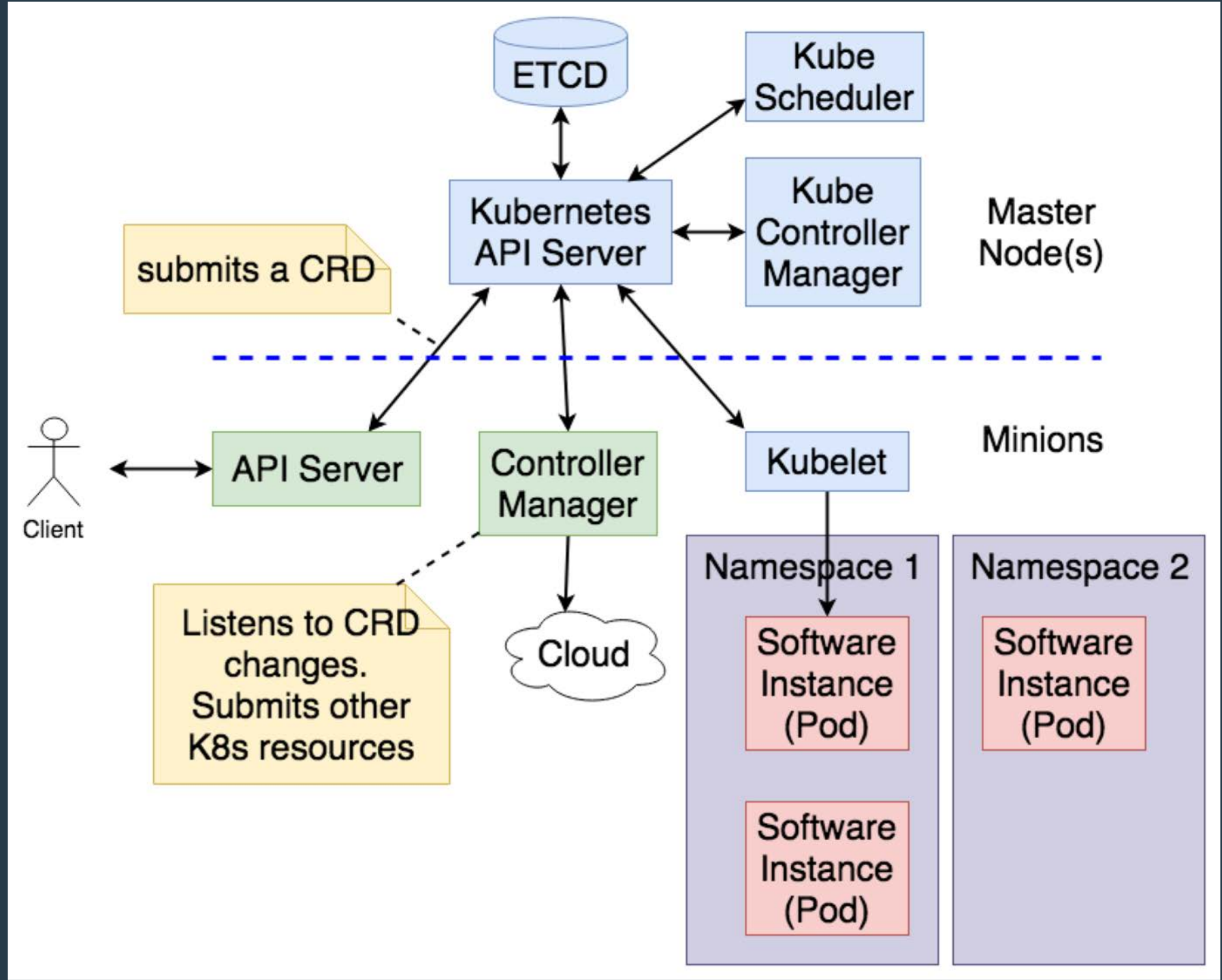




Questions

1. Why are the custom API server and the custom controller manager separate?

2. Why have a separate custom API server? Why not just use the Kube-APIServer?



Answer

1. Why are the custom API server and the custom controller manager separate?

- Update them separately
- Can post CRDs directly to the Kube-APIServer.

Answer

2. Why have a separate custom API server? Why not just use the Kube-APIServer?

- Tenants don’t need to learn Kubernetes.
- Don’t want tenants to even know Kubernetes is hosting their software.
- Limit tenants to only deploying our approved software.

Any Questions?

Supplemental Material

Tip: Use Kubernetes Labels

- Label tenant resources:
  - Tenant Name (i.e. red-team)
  - Creator (i.e. bob)
  - Software Application Name (i.e. redis)
  - Software Instance Name (i.e. bobs-redis)
- Makes it much easier to discover who is causing problems, and to manage their resources.
- For instance, you can bounce all their pods, or delete their software instance all together with one command.

Common API Endpoints

- CRUD – create, read, update, delete.
- List/Query all instances of CRD for tenant. Usually has some method of filtering.
- Describe – provides thorough information about the resource and its status.
- For your CRD and also tenant instances.

Middleware aka Web Filters

- Nonce – prevents repeat attacks
- Login/API key check
  - Authentication
- Signature check – hashes the nonce and other request parameters to confirm the user made the request.
- Authorization
  - To use this API
  - To act on behalf of this tenant
    - *Admin? Or tenant member?*
  - Read-only vs Write access
  - To view/alter the specific resource

More Information

- https://kubernetes.io/docs/tasks/access-kubernetes-api/custom-resources/custom-resource-definitions/
- https://github.com/kubernetes/sample-controller