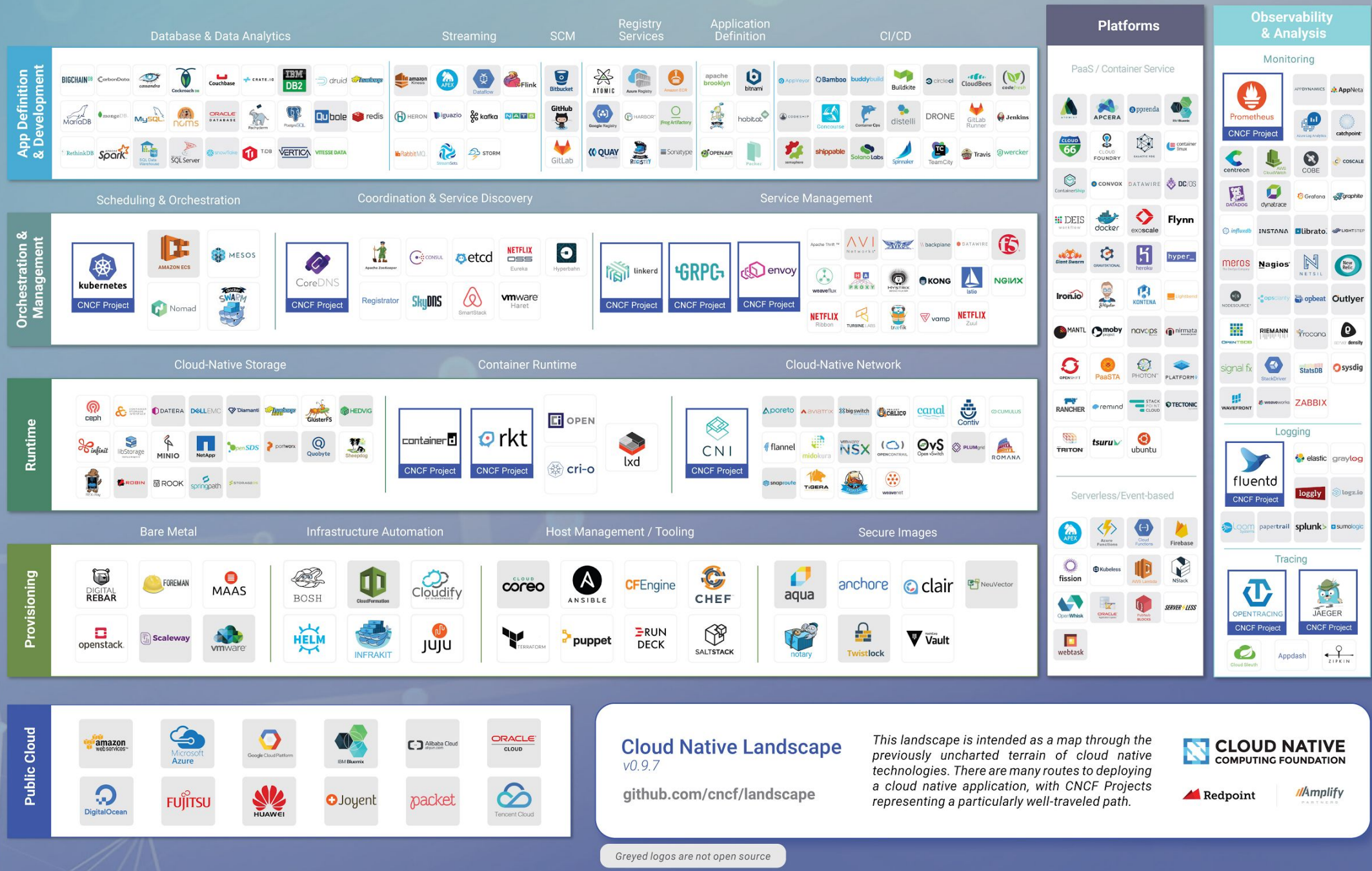
Slides credit: Lucas Käldström - luxas labs, Tim Hockin - Google

Kubernetes et les micro-services

Fabrice Jammes, IN2P3, fabrice.jammes@clermont.in2p3.fr

CLOUD NATIVE COMPUTING FOUNDATION

= Open Source Cloud Computing For Applications


App Definition & Development
Database & Data Analytics
Streaming
SCM
Registry Services
Application Definition
CI/CD
Orchestration & Management
Scheduling & Orchestration
Coordination & Service Discovery
Service Management
Runtime
Cloud-Native Storage
Container Runtime
Cloud-Native Network
Provisioning
Bare Metal
Infrastructure Automation
Host Management / Tooling
Secure Images
Public Cloud
Platforms
PaaS / Container Service
Serverless/Event-based
Observability & Analysis
Monitoring
Logging
Tracing
CNCF Project
Cloud Native Landscape
v0.9.7
github.com/cncf/landscape
This landscape is intended as a map through the previously uncharted terrain of cloud native technologies. There are many routes to deploying a cloud native application, with CNCF Projects representing a particularly well-traveled path.
CLOUD NATIVE COMPUTING FOUNDATION
Redpoint
Amplify
Greyed logos are not open source

La CNCF?

Une fondation à but non lucratif pour promouvoir le Cloud Natif.

Elle réunit des projets open source, des entreprises, des passionnés.

CNCF a été fondée en Décembre 2015 et fait partie de **la Fondation Linux.**

La CNCF:

- **promeut une boîte à outils de projets** pour le déploiement “moderne” des applications.
- aide les **projets** hébergés à **réussir**. Par exemple, elle **organise des événements** où la communauté peut se rencontrer.

Les projets hébergés par la CNCF

Orchestration

Monitoring

Distributed Tracing API

Logging

Service Mesh

Remote Procedure Call

Service Discovery

Container Runtime

Container Runtime

Networking API

Service Mesh

Distributed Tracing

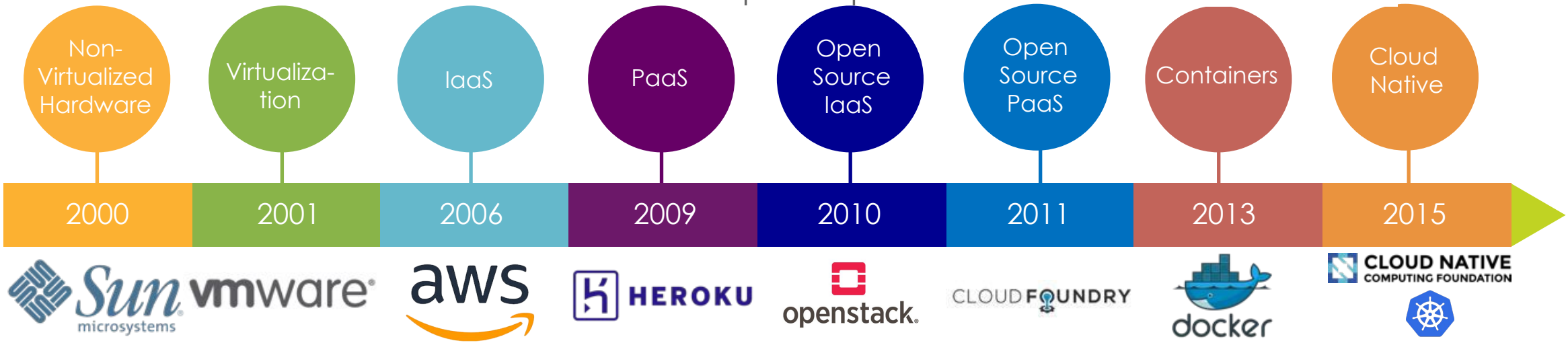
Les membres Platinum de la CNCF

De la Virtualisation au Cloud Natif

CLOUD NATIVE COMPUTING FOUNDATION

kubernetes

- Le cloud natif utilise une pile logicielle open source pour:
  - segmenter les applications en microservices,
  - emballer chaque partie dans son propre conteneur
  - et orchestrer dynamiquement ces conteneurs pour optimiser l'utilisation des ressources



Quel est l'état d'esprit «Cloud Native»?

Le Cloud Natif utilise une pile logicielle open source qui est:
- Conteneurisée
- Dynamiquement orchestrée
- Orientée microservices

Il y a trois mots-clés principaux:
- Vitesse
- Liberté
- Confiance

Alexis Richardson, PDG de Weaveworks, a effectué une excellente présentation sur ce sujet au KubeCon 2017 à Berlin.

EVERYONE'S EXCITED ABOUT

KUBERNETES

Que signifie "Kubernetes"?

Kubernetes
= Grec pour "pilote" ou "timonier d'un navire"

Qu’est-ce que Kubernetes?

= Un système mature et robuste d’orchestration de conteneur

Créé par Google, basé sur **Borg** et **Omega**, les systèmes qui fonctionnent aujourd’hui chez Google et dont la robustesse est éprouvée depuis plus de 10 ans. Google exécute ***2 milliards de conteneurs par semaine*** par semaine avec ces systèmes.

Créé par trois employés de Google initialement pendant l'**été 2014**;
il a grandi de façon exponentielle et est devenu le premier projet à être donné à la CNCF.

La première version de production: **v1.0.1 en Juillet 2015.**
Une nouvelle version mineure tous les trois mois depuis la v1.2.0 en Mars 2016.

La v1.9.1 vient de sortir le **4 Janvier 2018**.

Ce que fait *réellement* Kubernetes?

Abstraction de l'infrastructure matérielle grâce au concept de "**Node**"

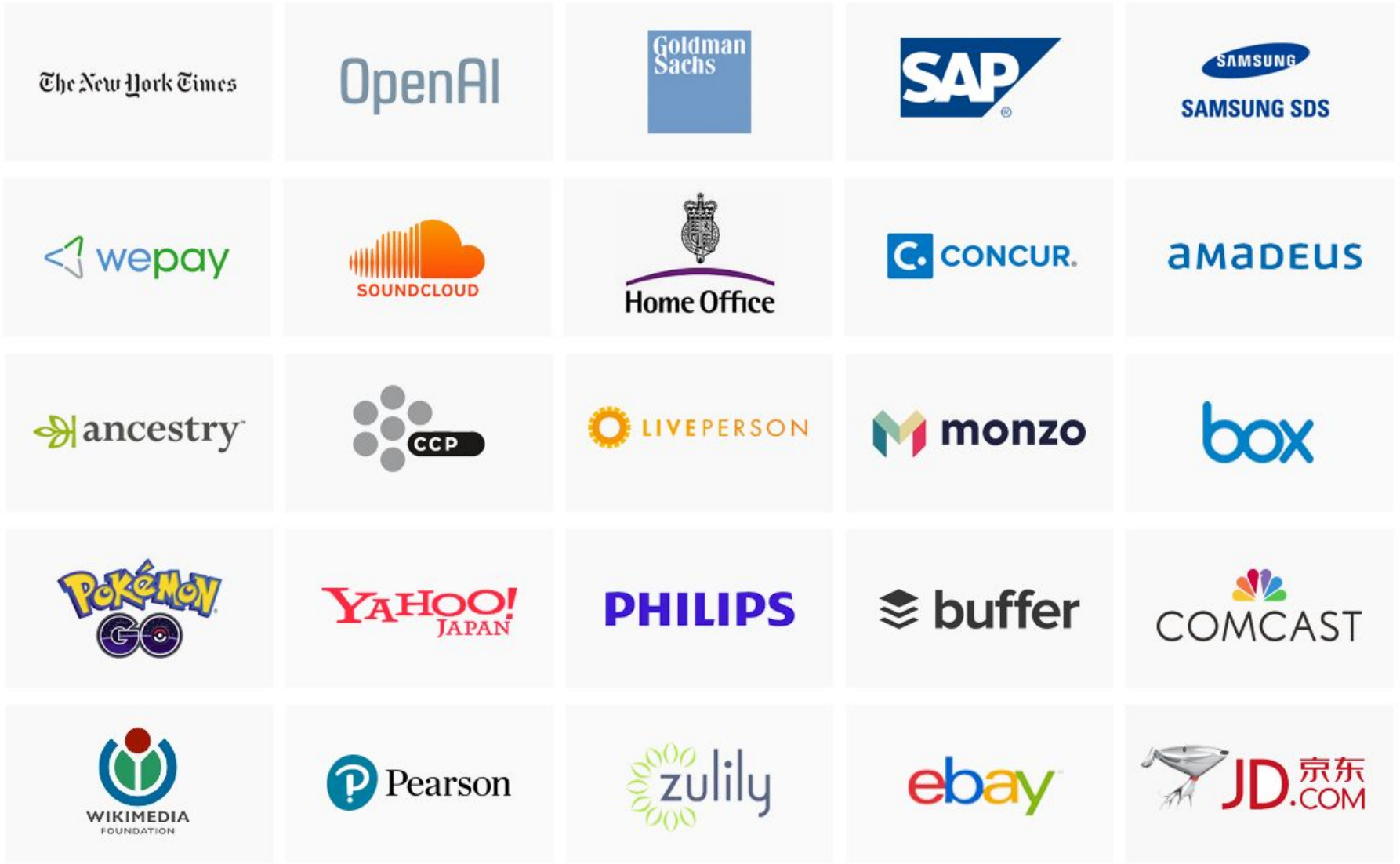
Principe: Gérez vos applications comme un troupeau (généricité, opérations de masse) plutôt que comme des animaux de compagnie (chaque opération est personnalisée avec soin et amour pour l'individu).

Kubernetes est le **Linux pour les systèmes distribués**:

- Linux (OS) résout les différences matérielles (avec différents types de CPU, etc.)-
- Kubernetes abstrait les milliers de nœuds d'un cluster et fournit des méthodes industrielles pour gérer les applications.

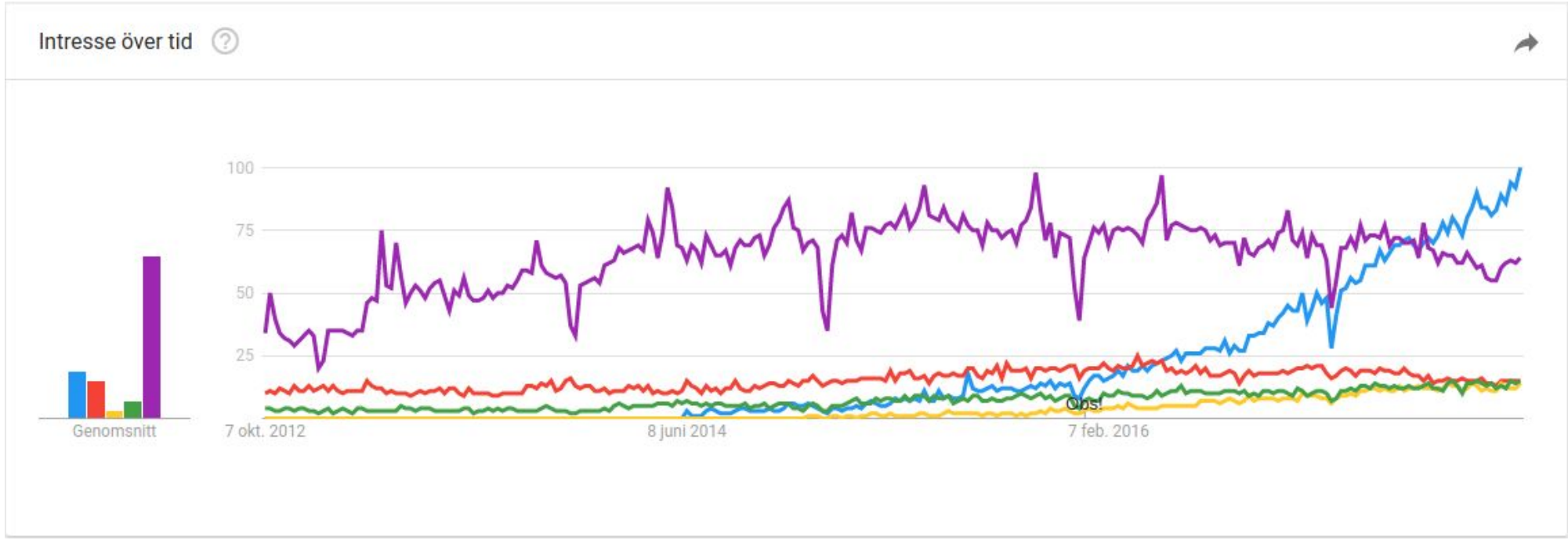
L'administrateur décrit et déclare l'"**état souhaité**", et Kubernetes convertit l'"**état courant**" en l'"**état souhaité**".

Quelques utilisateurs Kubernetes



La popularité de Kubernetes

Google Search interest over time in the 1.10.2012-1.10.2017 timespan



[Kubernetes est un des projet open source les plus actifs de l'histoire]

Statistiques concernant le projet Kubernetes

22 000+
utilisateurs sur Slack

5 700+
emploi pour Kubernetes

60 000+
commits dans l'année

32 000+
Pull Requests ouvertes dans l'année

25 000+
professionnels Kubernetes

11 000+
cours edX suivis dans les 3 derniers mois

18 000+
tickets ouverts dans l'année

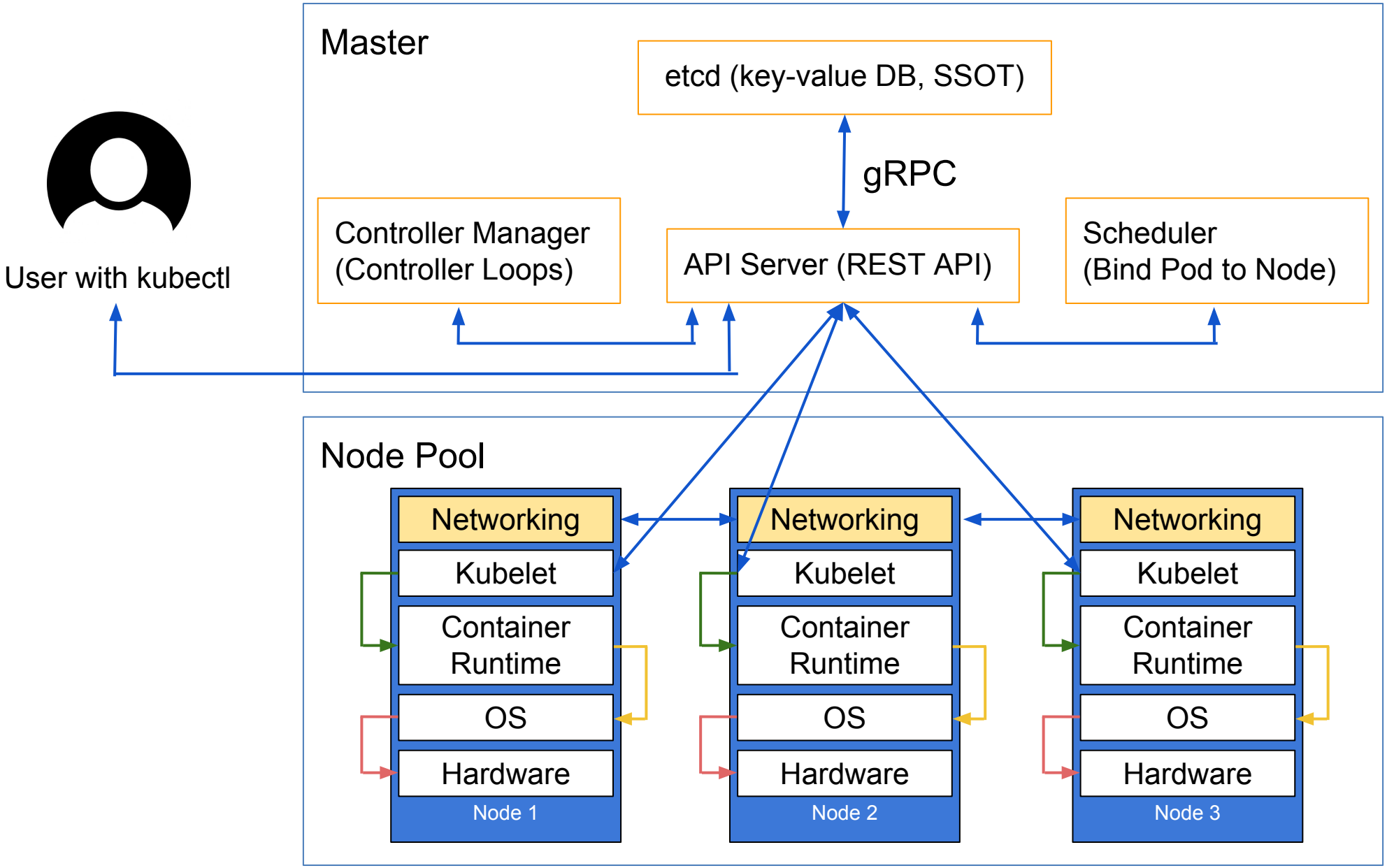
2 300+
auteurs

~23 PRs
merges/jour dans le repo central

Last updated: 1.10.2017

Source 1 Source 2 Source 3 Source 4

L’architecture de Kubernetes



Les concepts de base de Kubernetes (1)

Pod: Le bloc de base de Kubernetes, atomiquement planifiable, représente *une instance unique d'une application* dans Kubernetes. Chaque Pod a sa propre IP interne et unique. Les Pods sont mortels.

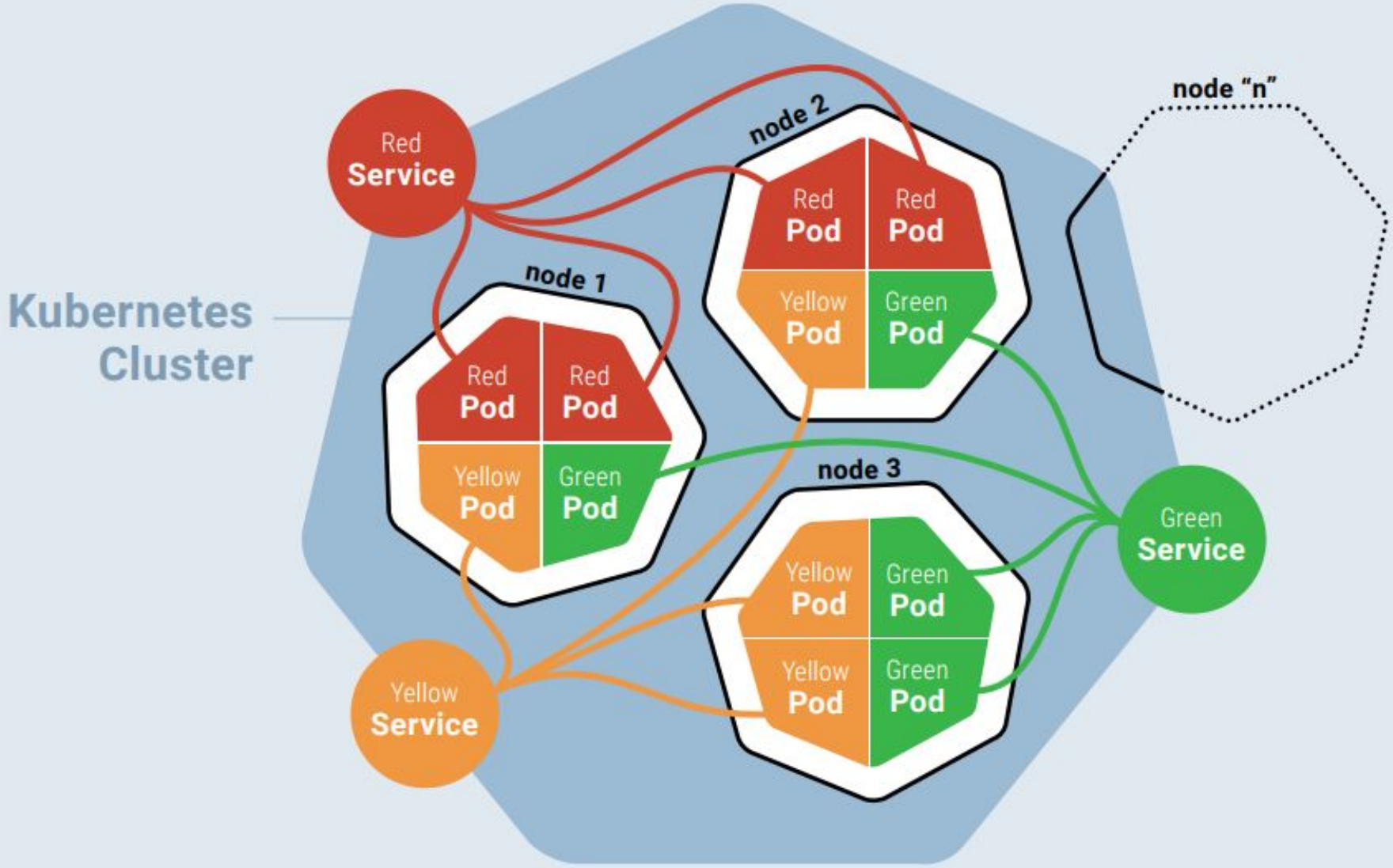
Deployment: Inclut un modèle de Pod et un ensemble de réplicas. Kubernetes s'assurera que l'état courant (nombre de réplicas, modèle de Pod) correspond toujours à l'état souhaité. La mise à jour d'un déploiement déclenche une "rolling update".

Les concepts de base de Kubernetes (2)

Service: Sélectionne les Pods à l'aide d'étiquettes (labels) et fournit un moyen stable et immortel de communiquer avec votre application en utilisant une IP interne ou un nom DNS.

Namespace: Méthode d'isolation logique, la plupart des objets kubernetes ont une portée relative à un seul Namespace. Vous pouvez y regrouper des applications et leur appliquer vos différentes stratégies de gestion.

Correspondance entre les Services et les Pods



Source: https://kubernetes.io/docs/concepts/services-networking/service/

THENEWSTACK

Exemple de “manifest” Kubernetes

```yaml
apiVersion: apps/v1beta2
kind: Deployment
metadata:
  labels:
    app: webapp
    role: frontend
  name: web-frontend
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: webapp
        role: frontend
    spec:
      containers:
      - image: nginx:1.13.1
        name: nginx
        ports:
        - containerPort: 443
          name: https
```

```yaml
apiVersion: v1
kind: Service
metadata:
  name: web-frontend
spec:
  selector:
    app: webapp
    role: frontend
  ports:
  - port: 443
    targetPort: 443
```

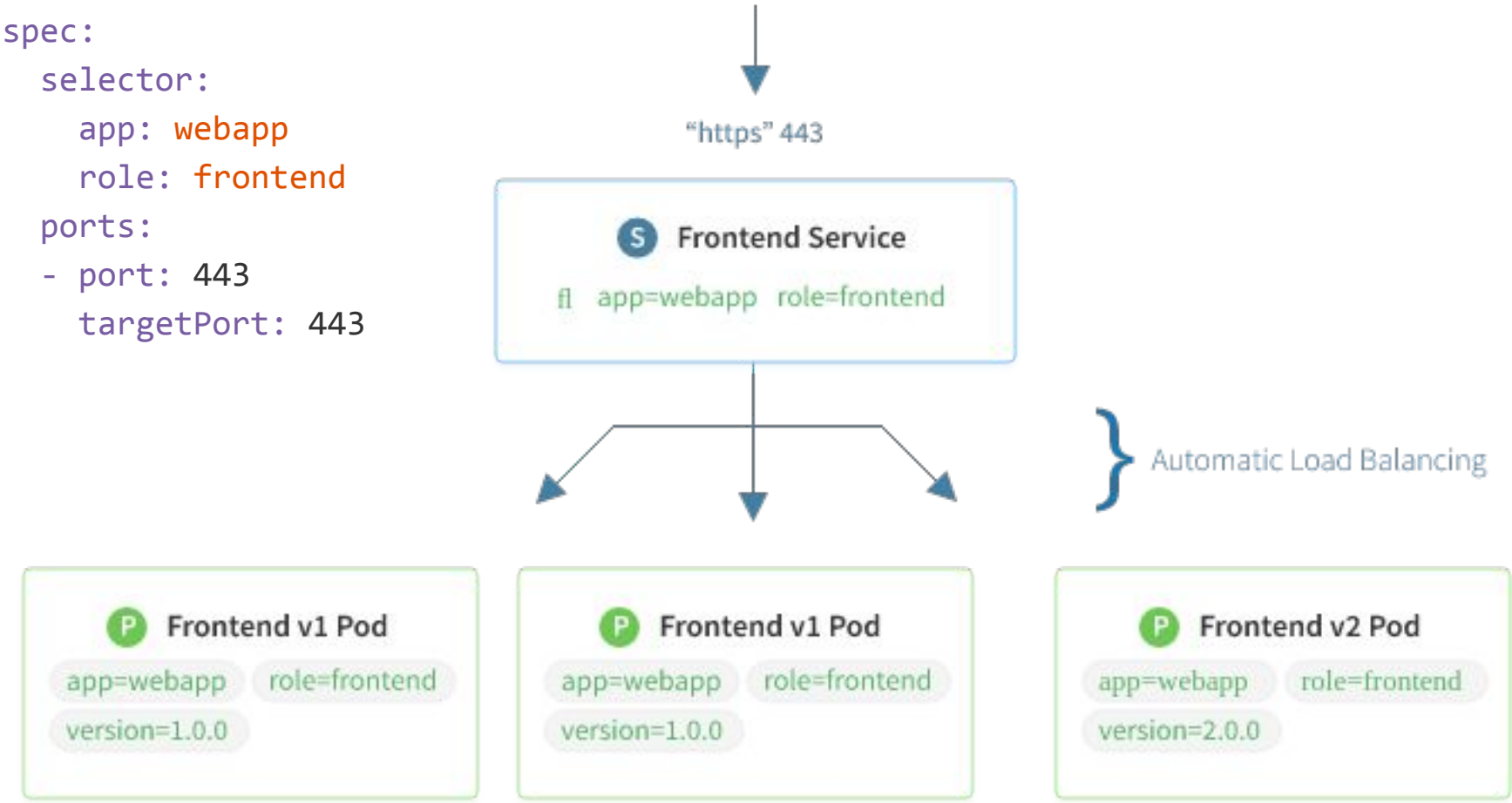
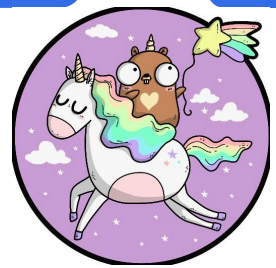


Image source

Comment démarrer avec Kubernetes



Play with Kubernetes: fonctionne instantanément dans votre navigateur

Créez un cluster sur votre portable ou votre poste de travail avec minikube

Créez un cluster en 2 lignes de commande avec kubeadm

Créer un cluster de production sur AWS avec kops

Créer un cluster de production sur GCE avec GKE (Google Container Engine)

kubicorn, Juju: d'autres méthodes visant à simplifier l'installation

Créer un cluster avec kubeadm

1. Créer une machine Linux avec Ubuntu, Debian, RHEL, CentOS or Fedora
2. Installer kubeadm:

```
curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | apt-key add -
cat <<EOF >/etc/apt/sources.list.d/kubernetes.list
deb http://apt.kubernetes.io/ kubernetes-xenial main
EOF
apt-get update && apt-get install -y kubeadm docker.io
```

3. Créer un noeud maître via kubeadm

```
kubeadm init
```

4. Installer un Pod réseau provenant d'un tiers:

```
kubectl apply -f https://git.io/weave-kube-1.6
```

5. Répéter les étapes 1 & 2 sur une autre machine et rejoindre le cluster:

```
kubeadm join --token <token> <master-ip>:6443
```

Quelques fonctionnalités de Kubernetes… (1)

- **Self-healing** : redémarre les conteneurs qui échouent, remplace et re-planifie les conteneurs lorsque les nœuds meurent, tue les conteneurs qui ne répondent pas au contrôle d'intégrité défini par l'utilisateur et les publie auprès des clients seulement lorsqu'il sont prêts

- **Automatic binpacking**: place automatiquement les conteneurs en fonction de leurs besoins en ressources et d'autres contraintes, sans sacrifier la disponibilité. Co-localisez les charges de travail critiques et "best-effort" afin de maximiser l'utilisation des ressources.

- **Horizontal scaling and autoscaling**: Faites passez votre application à l'échelle à l'aide d'une simple commande, d'une interface graphique, ou automatiquement en fonction de l'utilisation du processeur ou de métriques personnalisées.

- **Automated rollouts and rollbacks**: Kubernetes déploie progressivement les modifications apportées à votre application ou à sa configuration, tout en surveillant l'intégrité de l'application afin de s'assurer qu'elle ne tue pas toutes vos instances en même temps. En cas de problème, Kubernetes annulera les changements pour vous.

Quelques fonctionnalités de Kubernetes… (2)

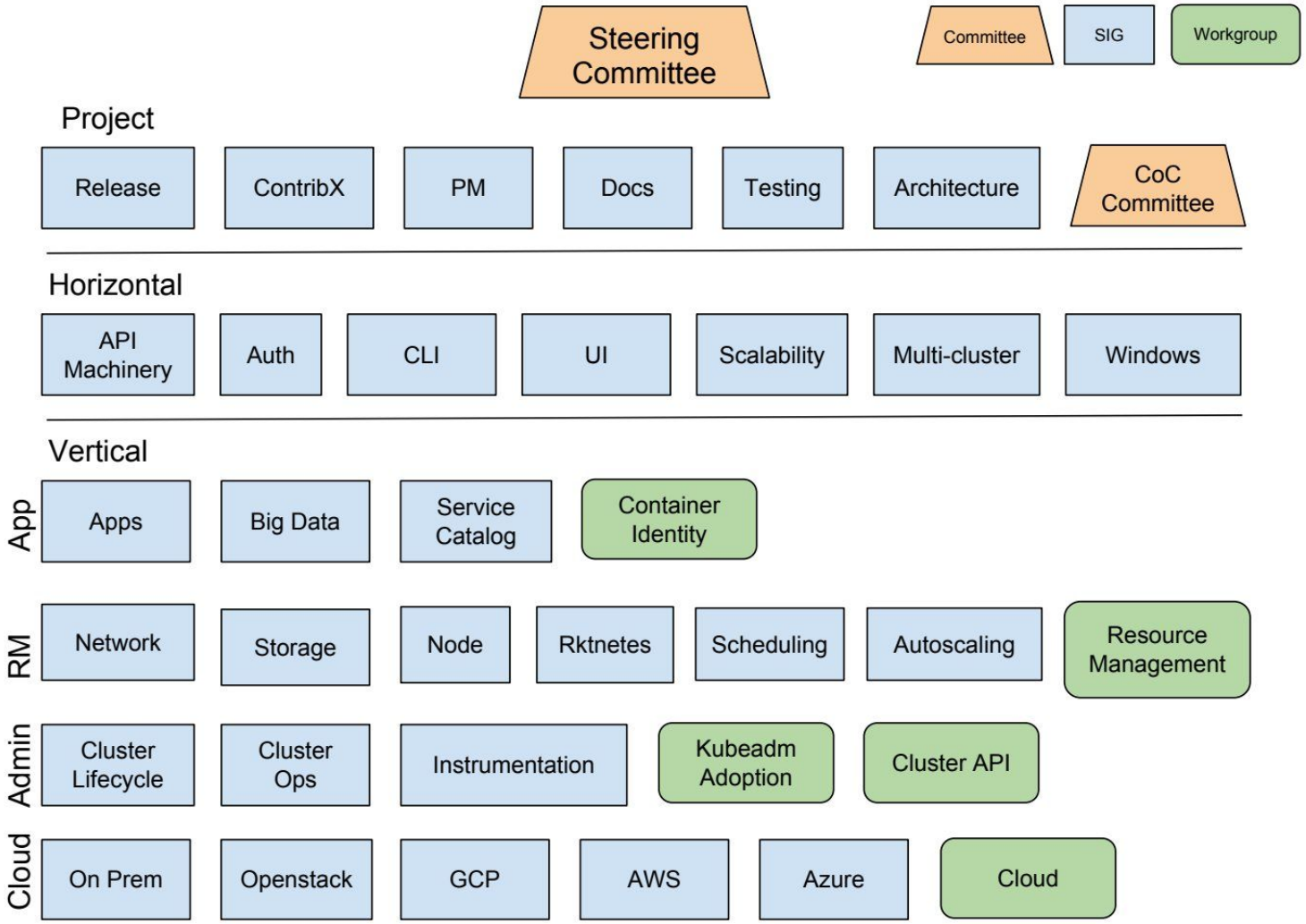
- **Service Discovery and Load Balancing**: inutile de modifier votre application pour utiliser un mécanisme de découverte de service tiers. Kubernetes donne aux conteneurs leurs propres adresses IP et un seul nom DNS pour un ensemble de conteneurs, et peut équilibrer la charge entre eux.

- **Secret and configuration management**: Déployez et mettez à jour les secrets et la configuration de l'application sans reconstruire votre image et sans exposer les secrets les fichiers de configuration de votre déploiement.

- **Storage Orchestration**: monte automatiquement le système de stockage de votre choix, qu'il s'agisse du stockage local, d'un fournisseur de cloud public tel que GCP ou AWS, ou d'un système de stockage réseau tel que NFS, iSCSI, Gluster, Ceph, Cinder ou Flocker

- **Batch Execution**: En plus des services, Kubernetes peut gérer vos batch et votre CI, en remplaçant les conteneurs qui échouent, si vous le souhaitez

Tout est conçu via les SIG (Special Interest Groups)



Les Special Interest Group gèrent les différents composants et fonctionnalités de Kubernetes.

Tout le code de l'organisation Kubernetes dans Github doit appartenir à un ou plusieurs SIG

Les SIG organisent des réunions vidéo régulières (souvent hebdomadaires) au cours desquelles les participants discutent des décisions de conception, des nouvelles fonctionnalités, des bogues, des tests, de l'intégration ou de tout ce qui est pertinent pour le groupe. **Assister à ces réunions est le meilleur moyen de connaître le projet**

Image source

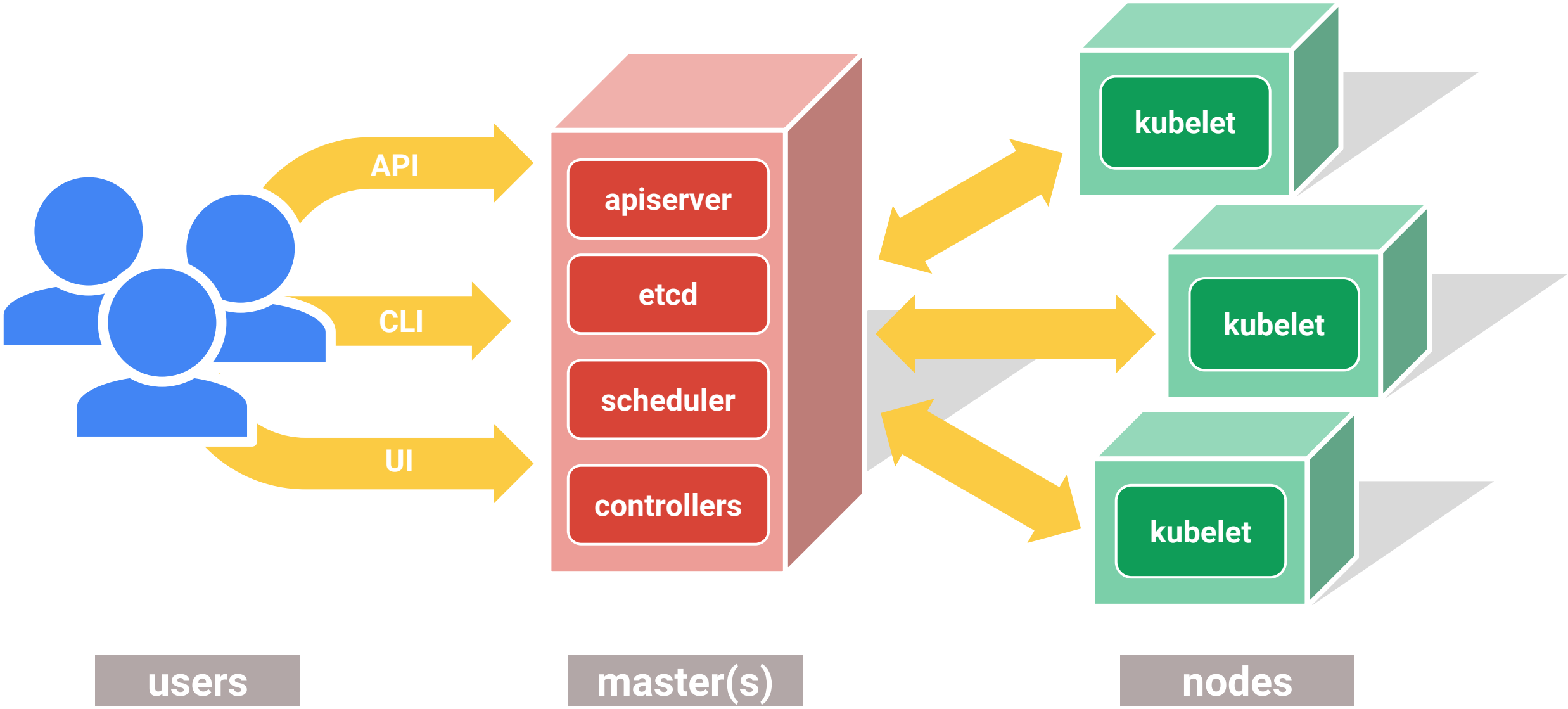
Kubernetes under the hood

Greek for *“Helmsman”*; also the root of the words *“governor”* and *“cybernetic”*

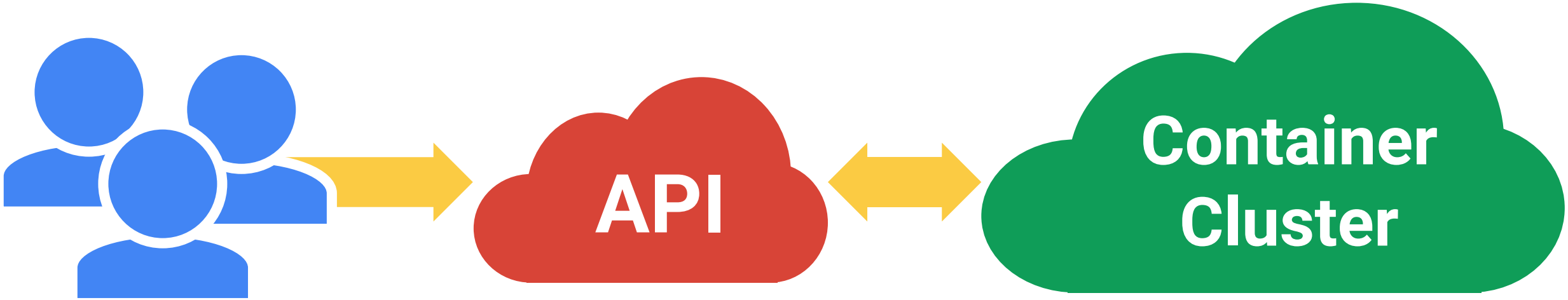
- Manages container clusters
- Inspired and informed by Google’s experiences and internal systems
- Supports multiple cloud and bare-metal environments
- Supports multiple container runtimes
- **100% Open source**, written in Go

Manage applications, not machines

The 10000 foot view



All you really care about



Container clusters: A story in two parts

Container clusters: A story in two parts

1. Setting up the cluster

- Choose a cloud: GCE, AWS, Azure, Rackspace, on-premises, ...
- Choose a node OS: CoreOS, Atomic, RHEL, Debian, CentOS, Ubuntu, ...
- Provision machines: Boot VMs, install and run kube components, ...
- Configure networking: IP ranges for Pods, Services, SDN, ...
- Start cluster services: DNS, logging, monitoring, ...
- Manage nodes: kernel upgrades, OS updates, hardware failures...

Not the easy or fun part, but unavoidable

Container clusters: A story in two parts

2. Using the cluster

- Run Pods & Containers
- Replication controllers
- Services
- Volumes

This is the fun part!

A distinct set of problems from cluster setup and management

Don’t make developers deal with cluster administration!

Accelerate development by focusing on the applications, not the cluster

Workload Portability

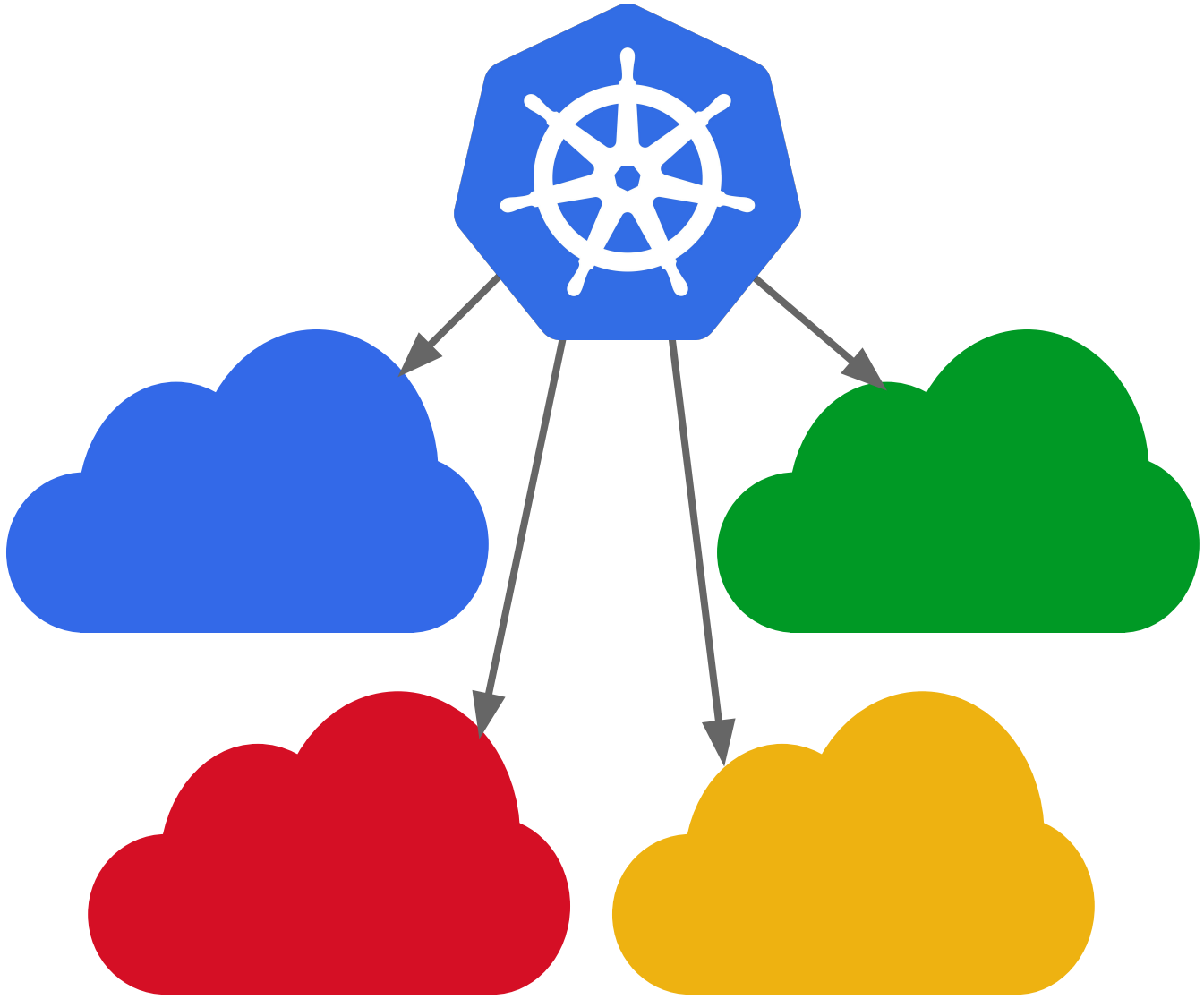
Workload portability

Goal: Avoid vendor lock-in

Runs in many environments, including “the cloud”, “bare metal”, and “your laptop”

The API and the implementation are 100% open

The whole system is modular and replaceable



Workload portability

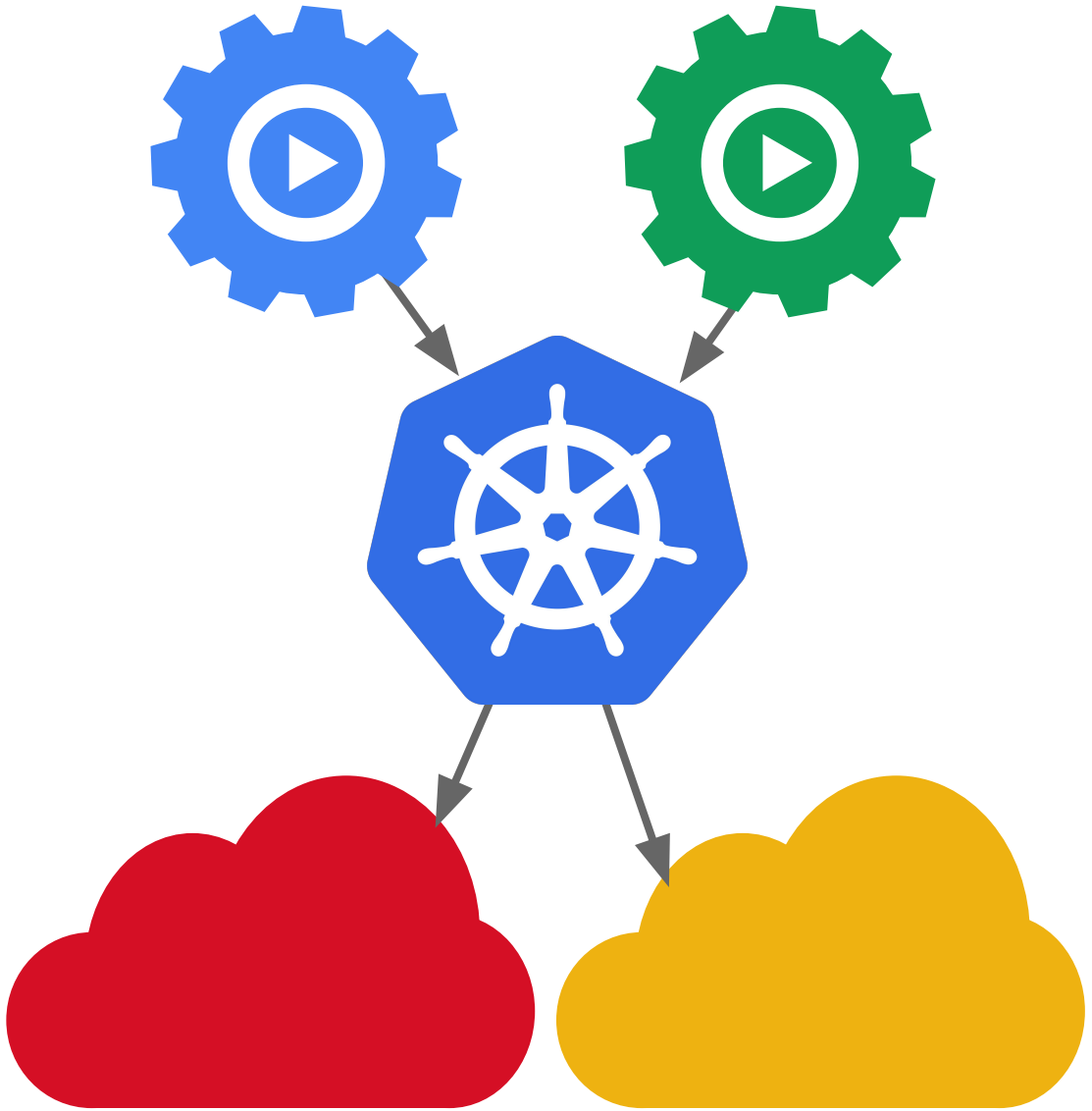
Goal: Write once, run anywhere*

Don’t force apps to know about concepts that are cloud-provider-specific

Examples of this:

- Network model
- Ingress
- Service load-balancers
- PersistentVolumes

** approximately*



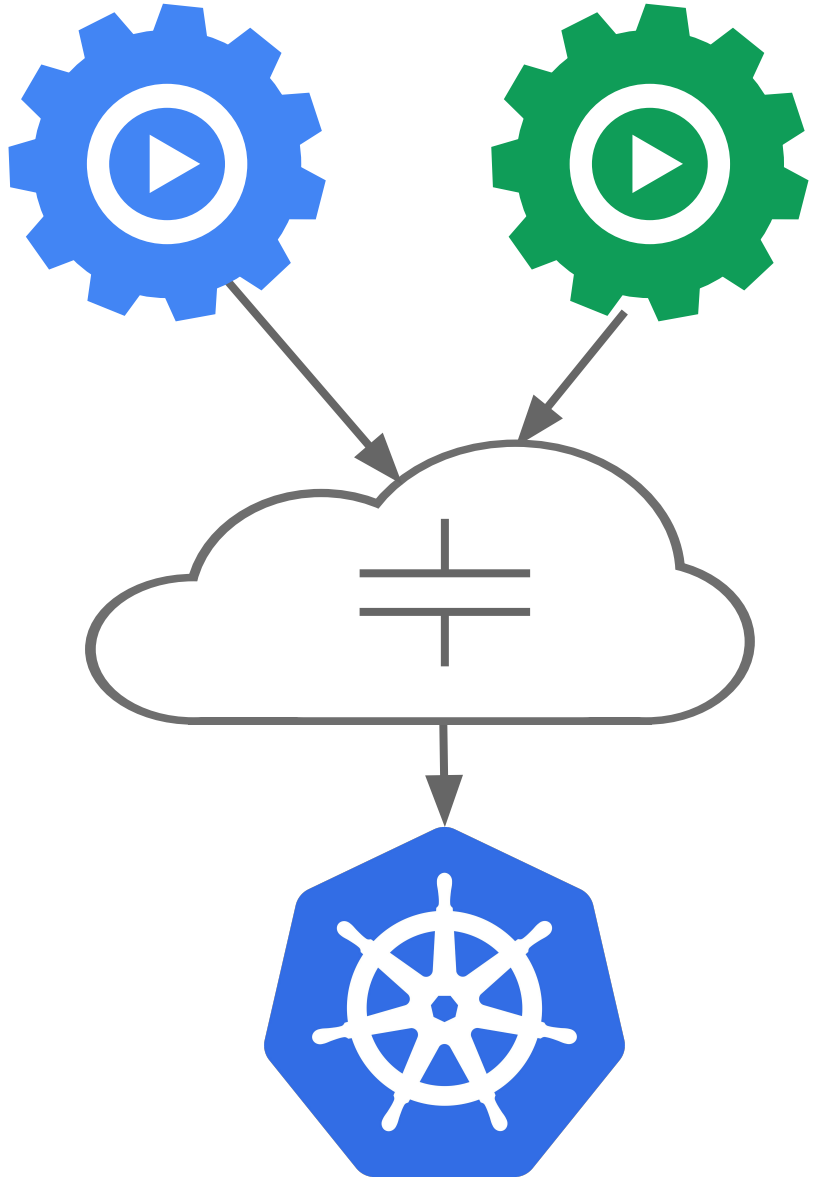
Workload portability

Goal: Avoid coupling

Don’t force apps to know about concepts that are Kubernetes-specific

Examples of this:

- Namespaces
- Services / DNS
- Downward API
- Secrets / ConfigMaps



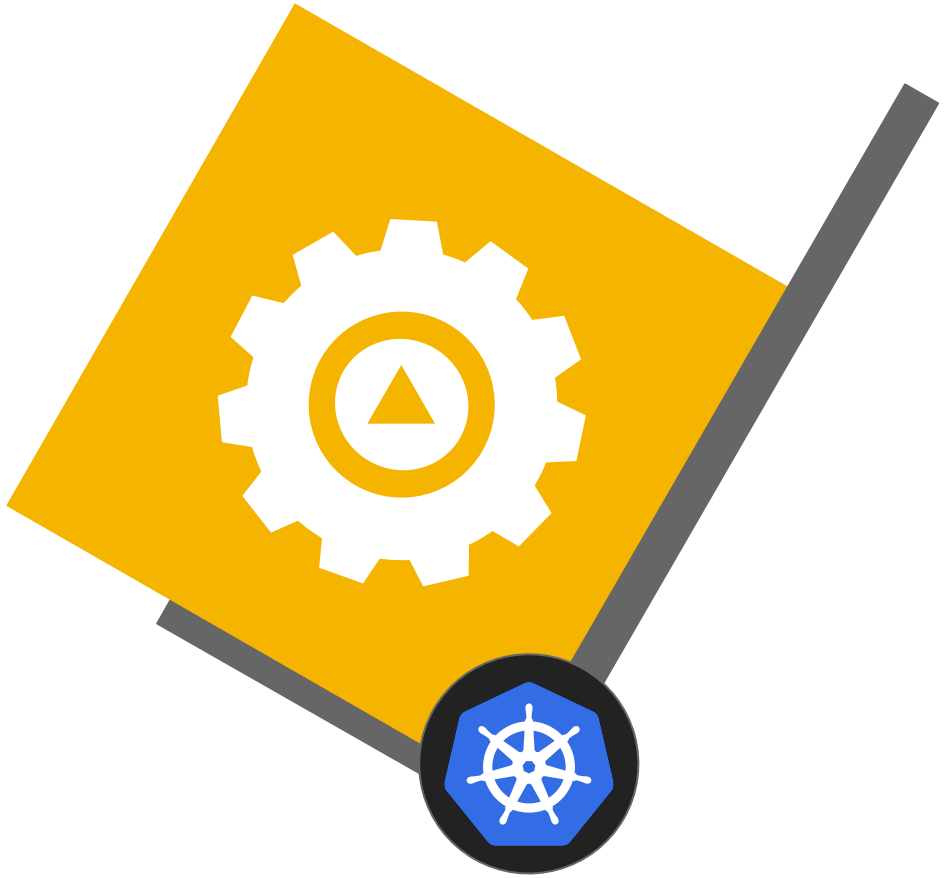
Workload portability

Result: Portability

Build your apps on-prem, lift-and-shift into cloud when you are ready

Don’t get stuck with a platform that doesn’t work for you

Put your app on wheels and move it whenever and wherever you need



Networking

Kubernetes networking

IPs are **cluster-scoped**

Pods can reach each other directly

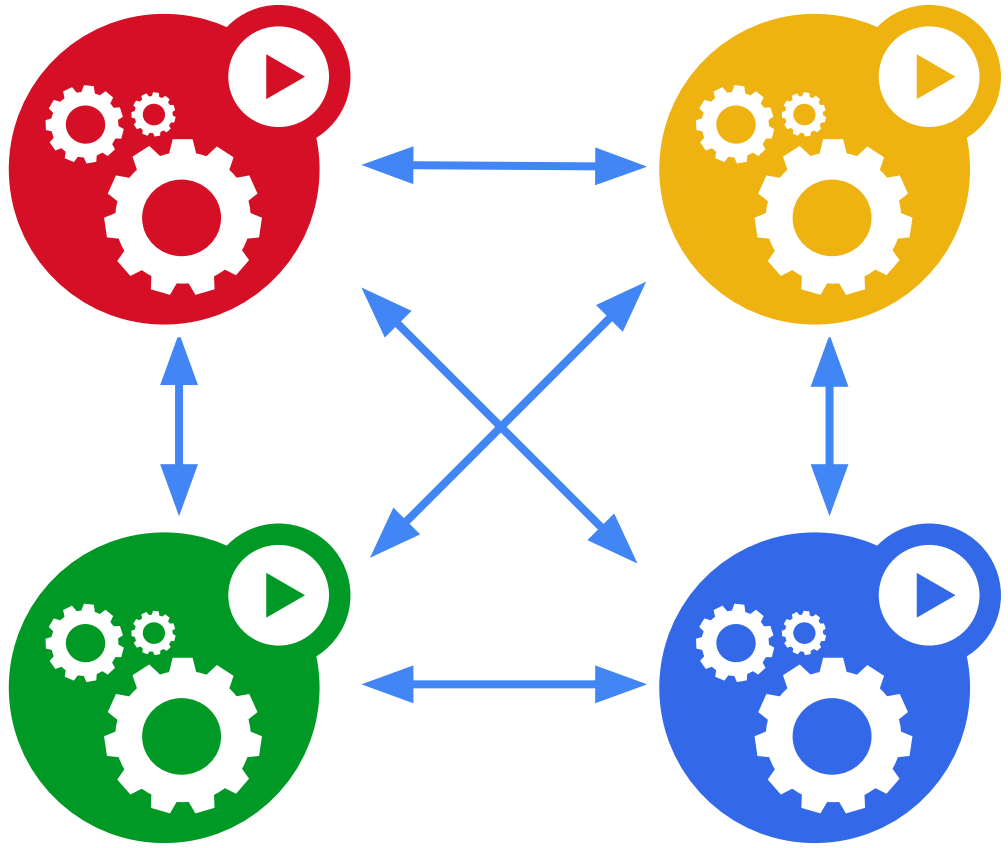
- even across nodes

No brokering of port numbers

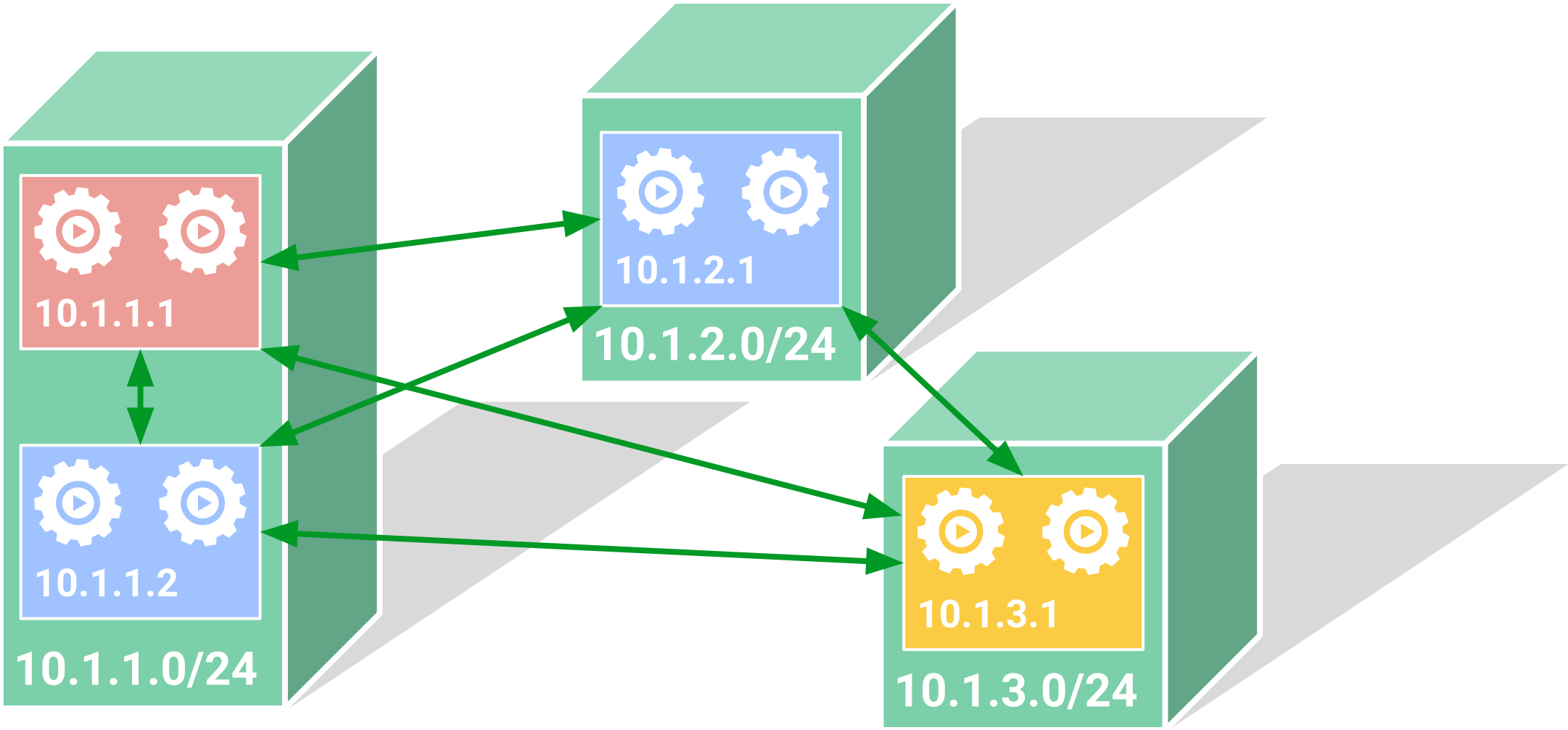
- too complex, why bother?

This is a fundamental requirement

- can be L3 routed
- can be underlayed (cloud)
- can be overlayed (SDN, software defined network)



Kubernetes networking



Pods

Pods

Small group of containers & volumes

Tightly coupled

The **atom** of scheduling & placement

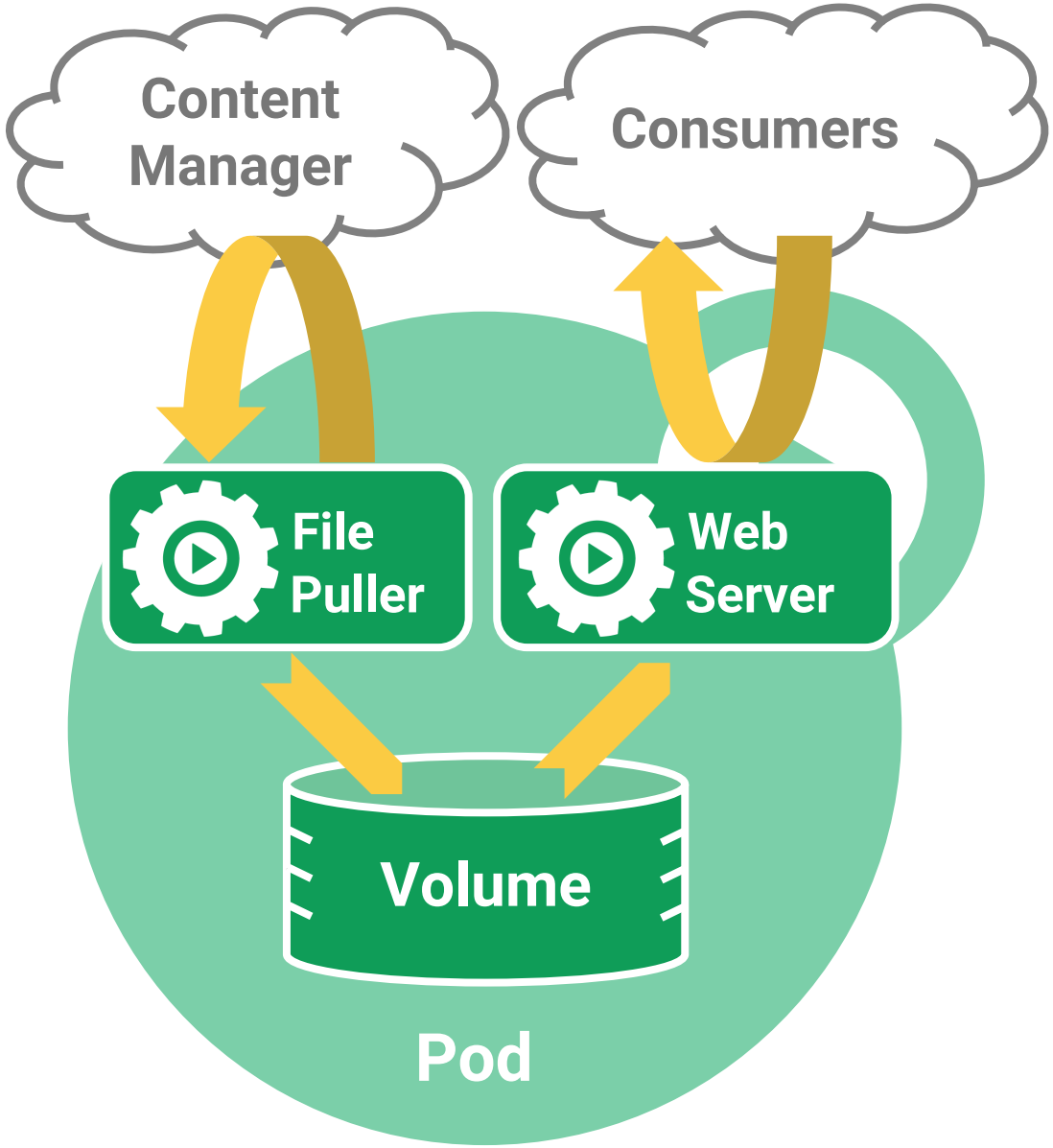
Shared namespace

- share IP address & localhost
- share IPC, etc.

Managed lifecycle

- bound to a node, restart in place
- can die, cannot be reborn with same ID

Example: data puller & web server



Volumes

Pod-scoped storage

Support many types of volume plugins

- Empty dir (and tmpfs)
- Host path
- Git repository
- GCE Persistent Disk
- AWS Elastic Block Store
- Azure File Storage
- iSCSI
- NFS
- Photon
- Portworx
- Quobyte
- vSphere
- GlusterFS
- Ceph File and RBD
- Cinder
- FibreChannel
- ScaleIO
- StorageOS
- Secret, ConfigMap, DownwardAPI
- Flex (exec a binary)
- ...

Labels & Selectors

Labels

Arbitrary metadata

Attached to **any API object**

Generally represent **identity**

Queryable by **selectors**

- think SQL '*select ... where ...*'

The **only** grouping mechanism

- pods in a Deployment
- pods in a Service
- capabilities of a node (constraints)

Selectors

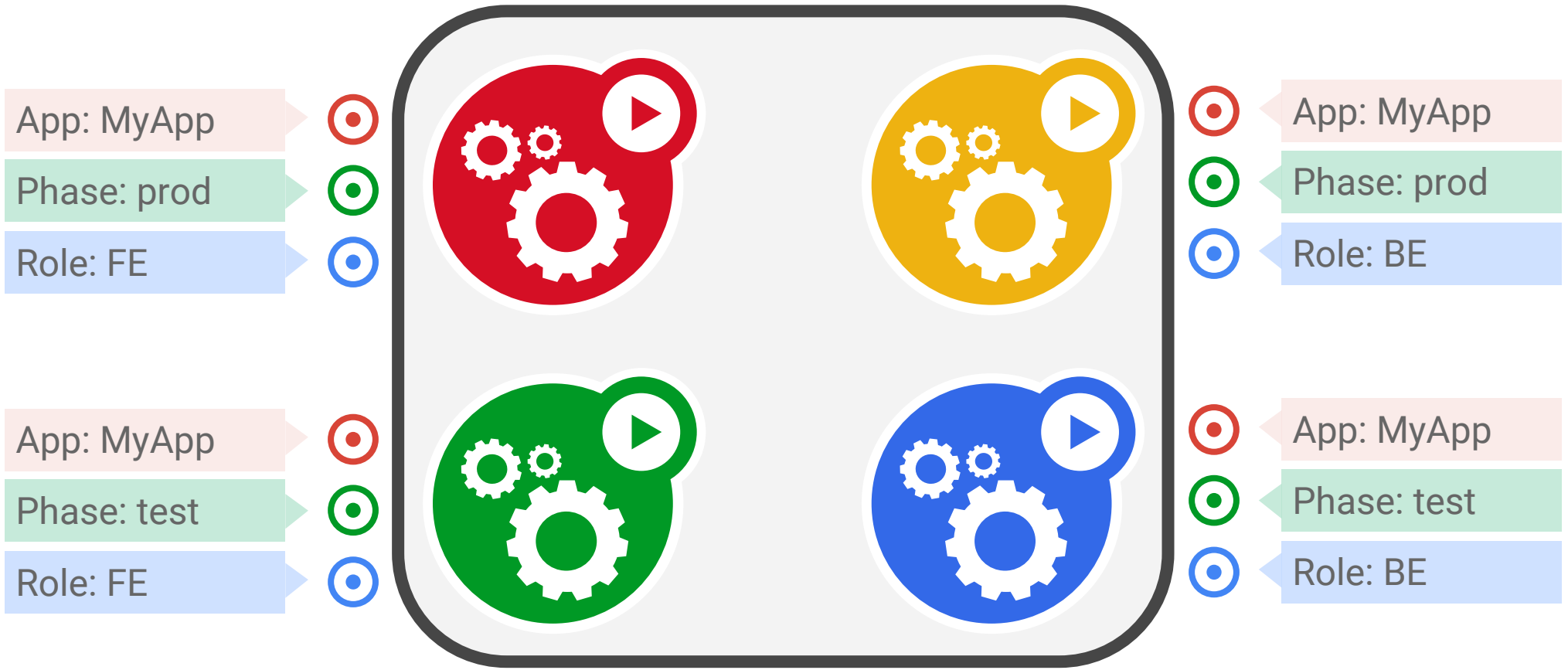
App: MyApp
Phase: prod
Role: FE

App: MyApp
Phase: prod
Role: BE

App: MyApp
Phase: test
Role: FE

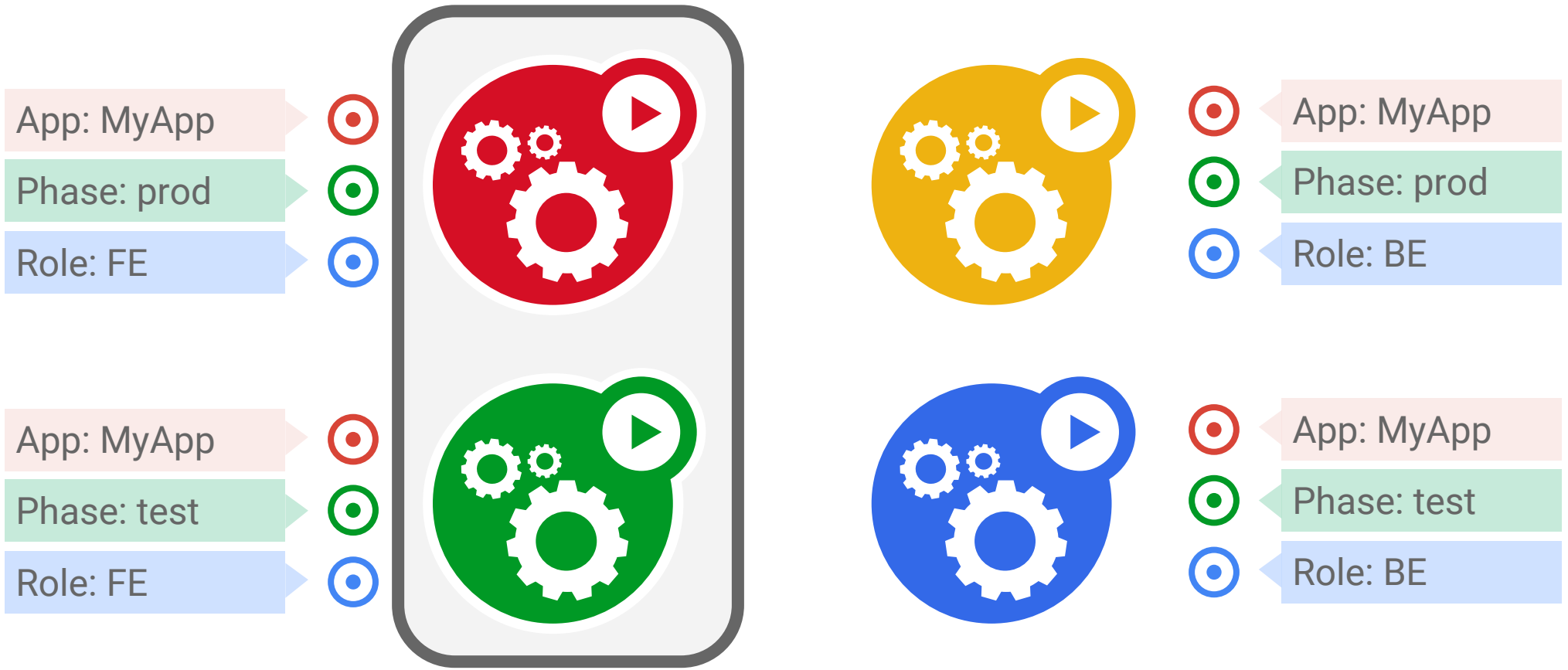
App: MyApp
Phase: test
Role: BE

Selectors



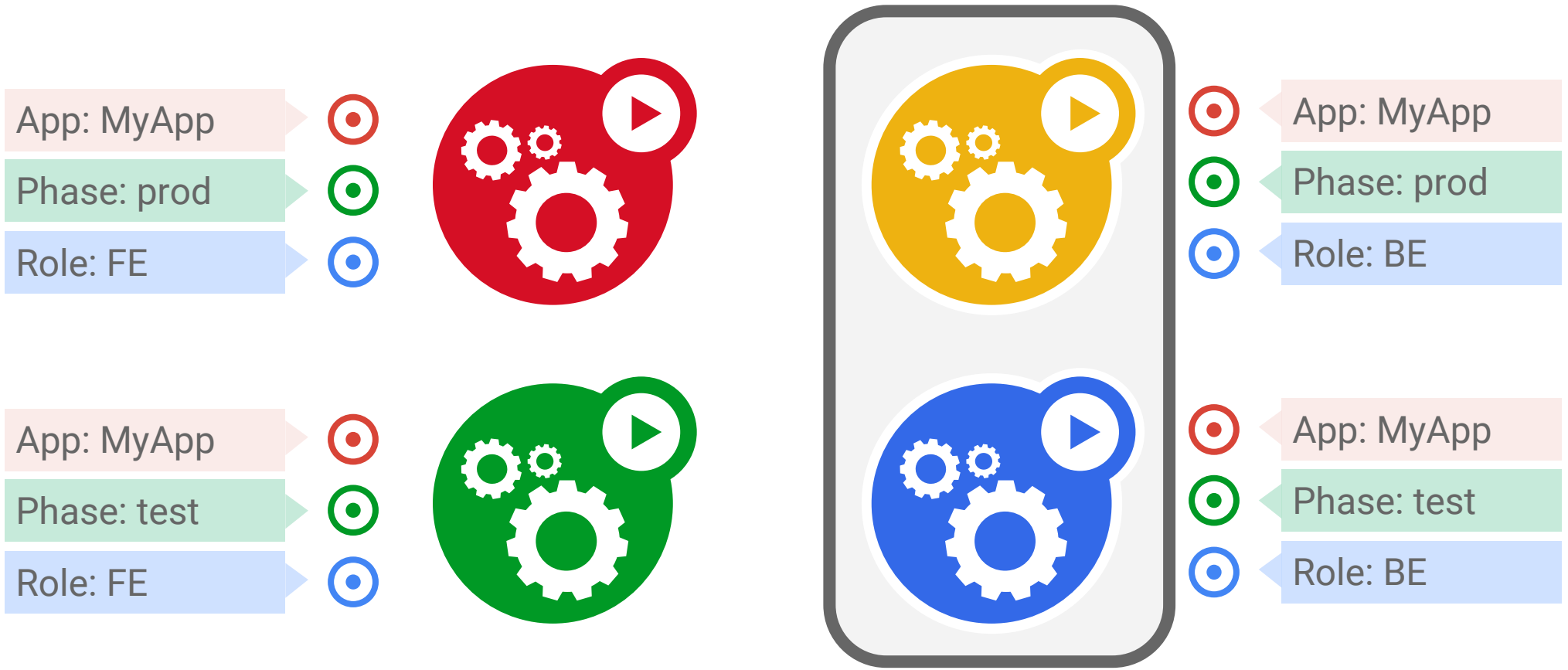
App = MyApp

Selectors



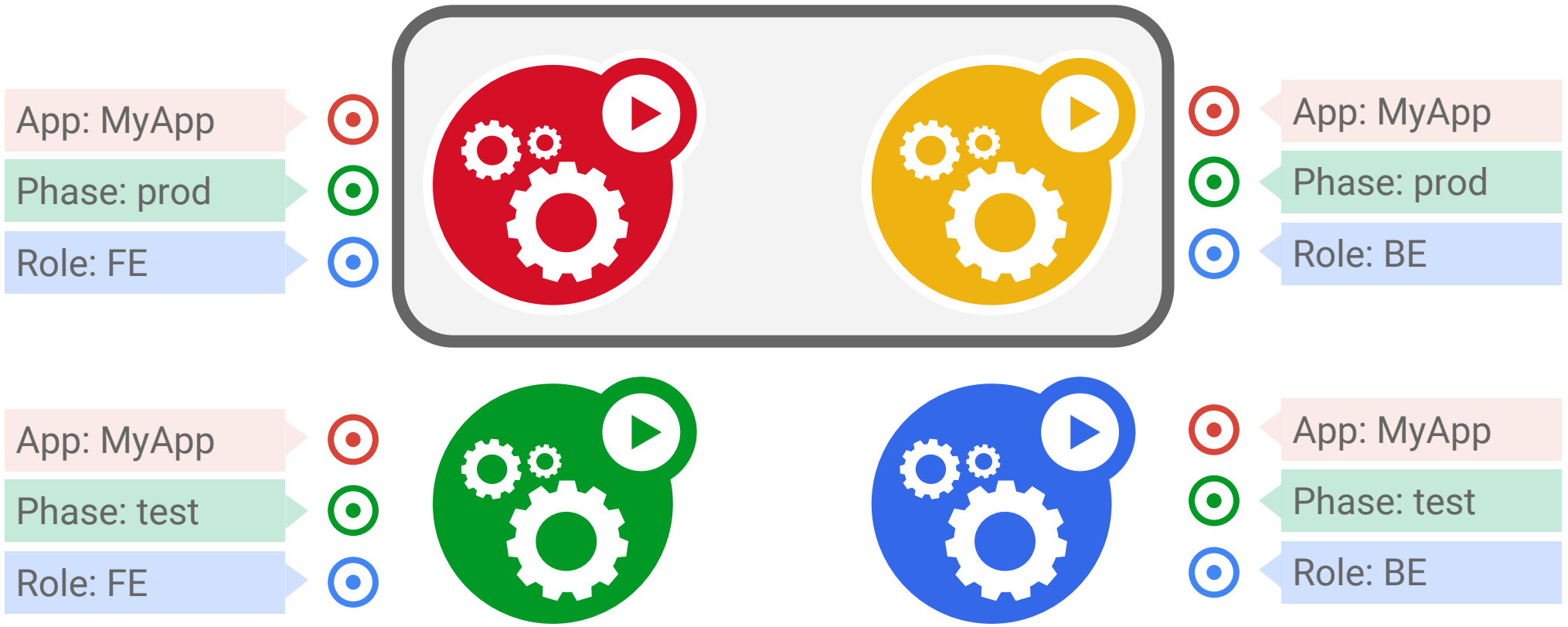
App = MyApp, Role = FE

Selectors



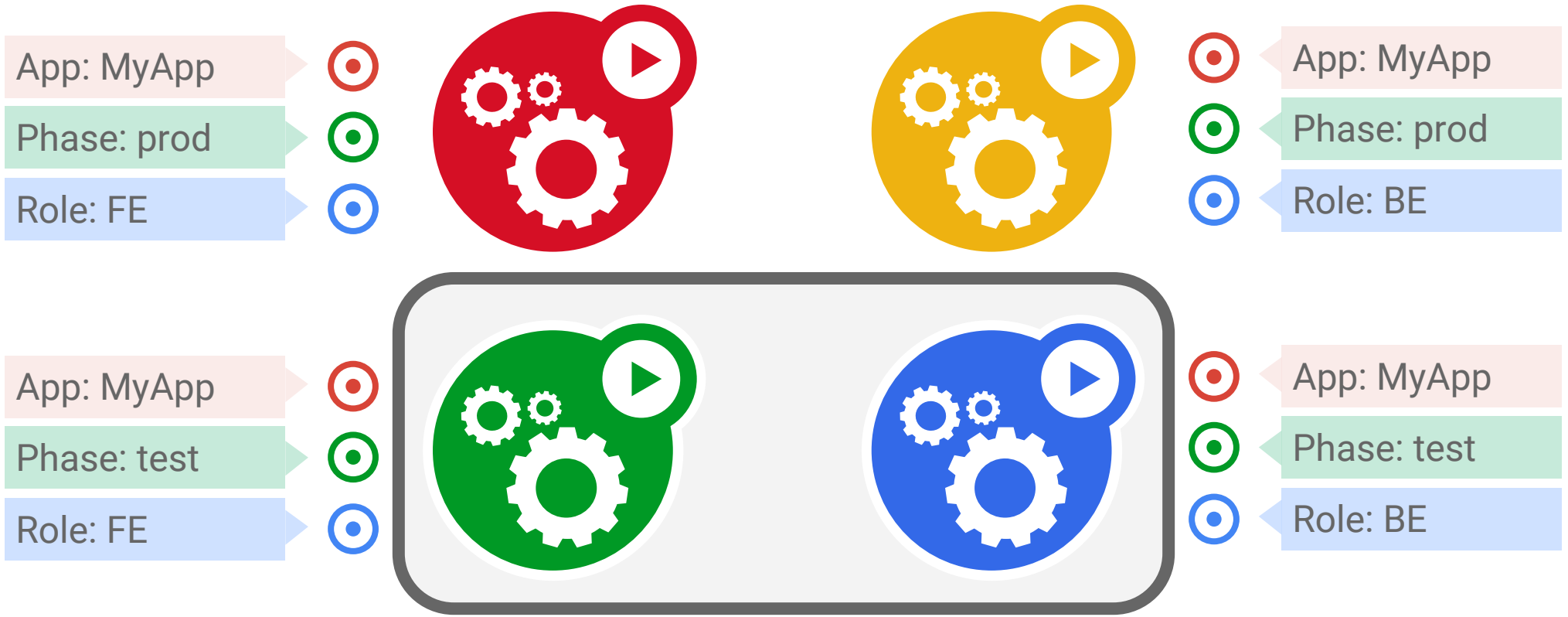
App = MyApp, Role = BE

Selectors



App = MyApp, Phase = prod

Selectors



App = MyApp, Phase = test

Running Pods

ReplicaSets

A simple controller loop

Runs out-of-process wrt API server

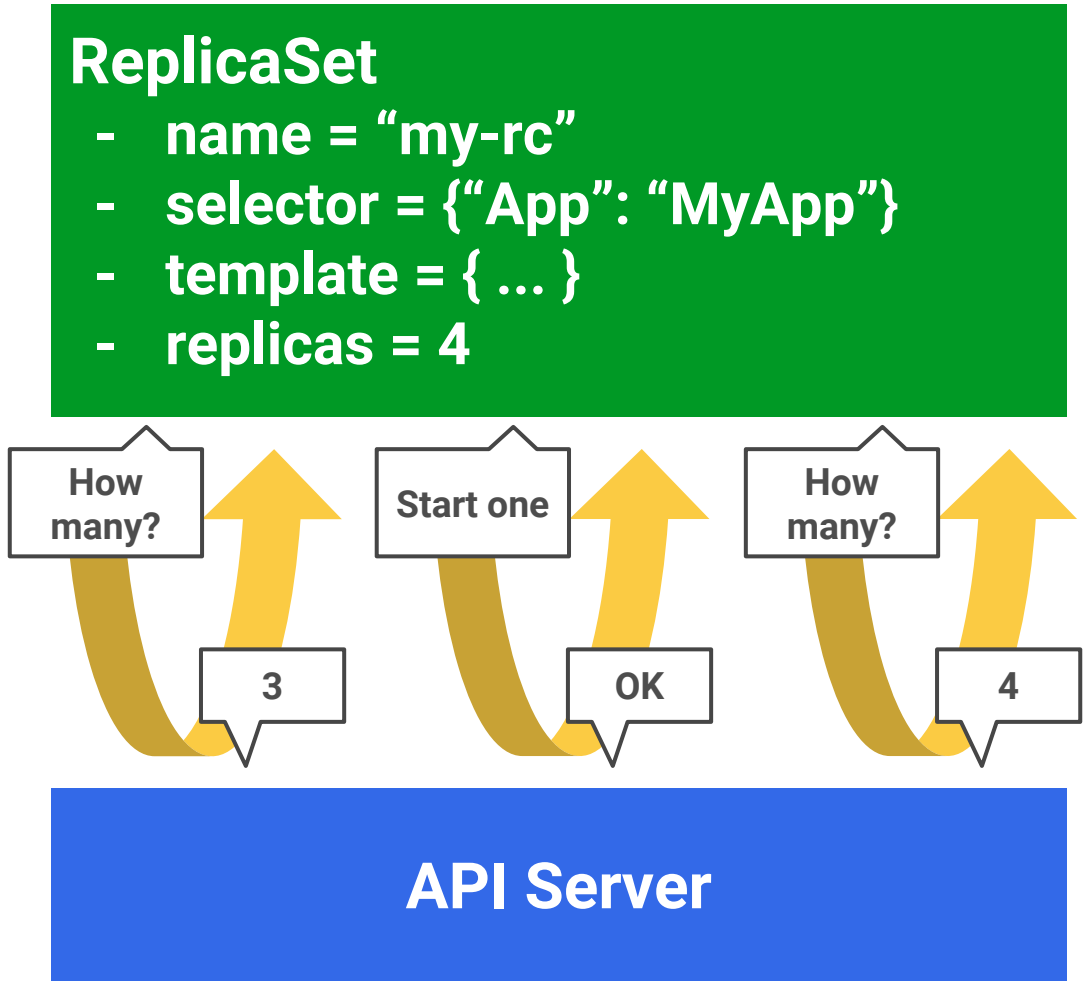
One job: ensure N copies of a pod

- grouped by a selector
- too few? start some
- too many? kill some

Layered on top of the public Pod API

Replicated pods are **fungible**

- Have no order or identity

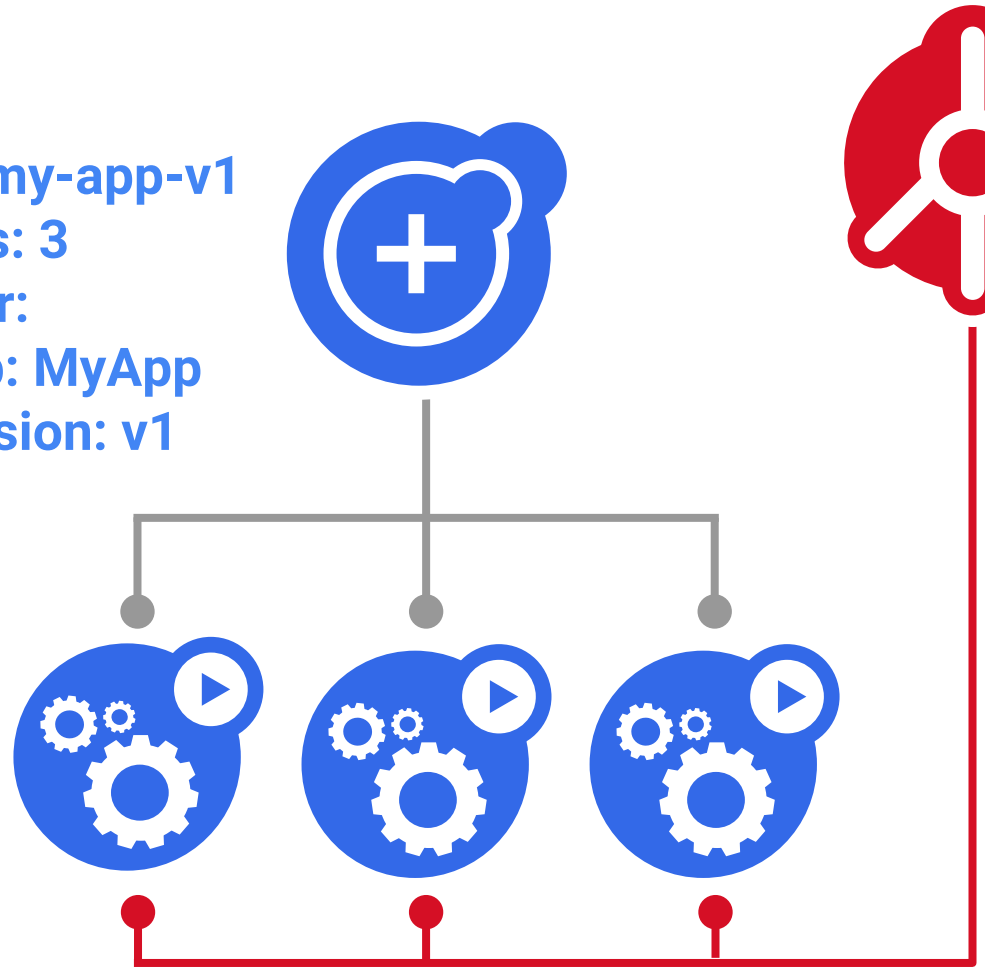


Rolling Updates

Rolling Update

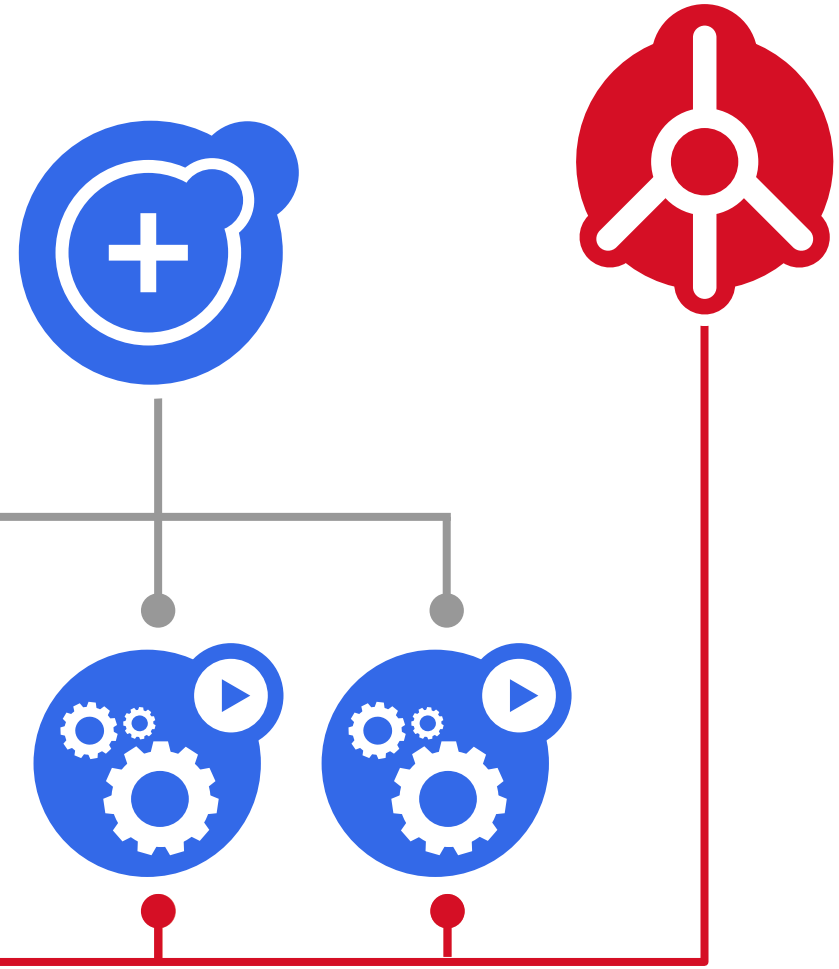
Service
- app: MyApp

ReplicaSet
- name: my-app-v1
- replicas: 3
- selector:
  - app: MyApp
  - version: v1



Rolling Update

Service
- app: MyApp

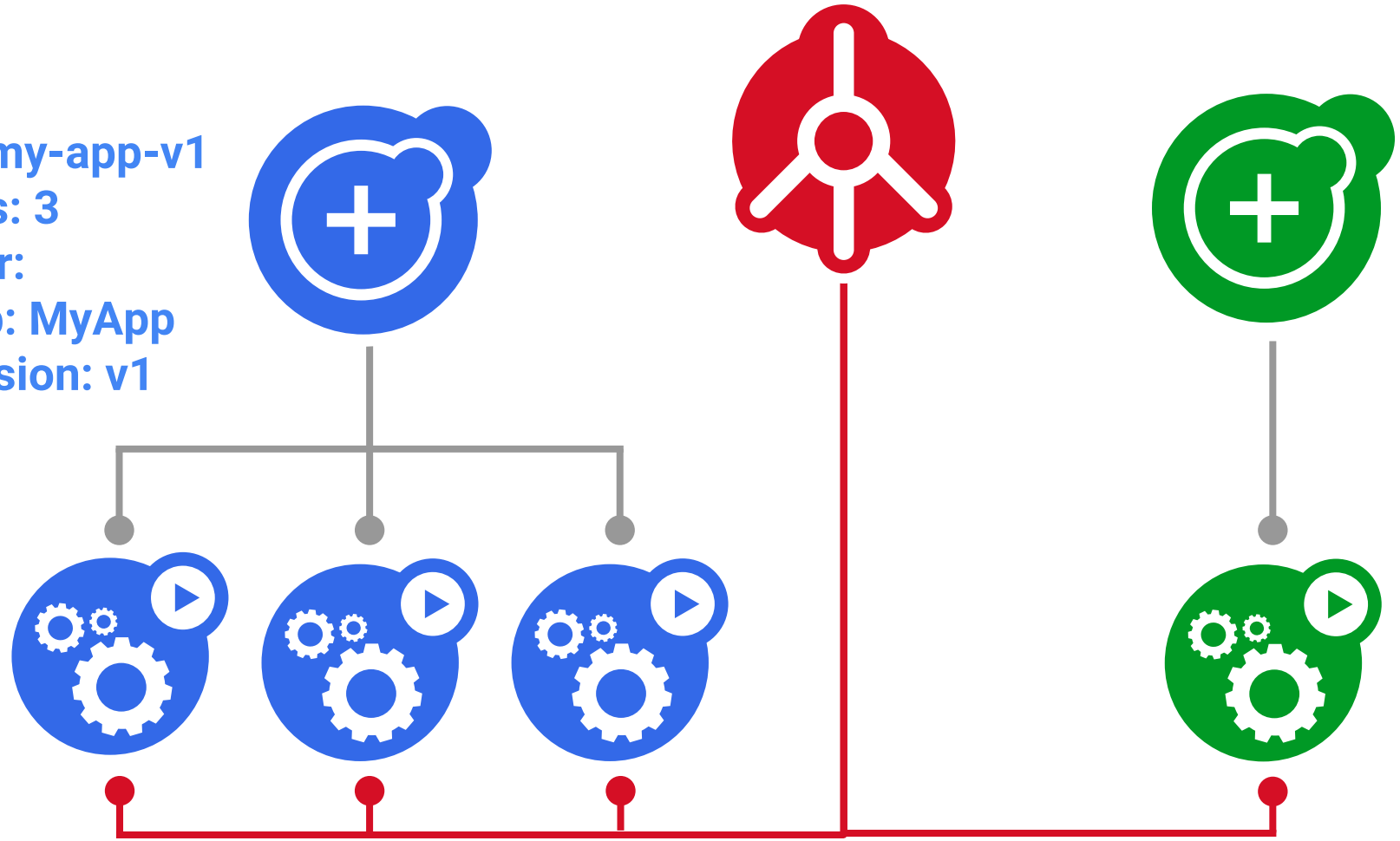


ReplicaSet
- name: my-app-v1
- replicas: 3
- selector:
  - app: MyApp
  - version: v1

ReplicaSet
- name: my-app-v2
- replicas: 0
- selector:
  - app: MyApp
  - version: v2

Rolling Update

Service
- app: MyApp

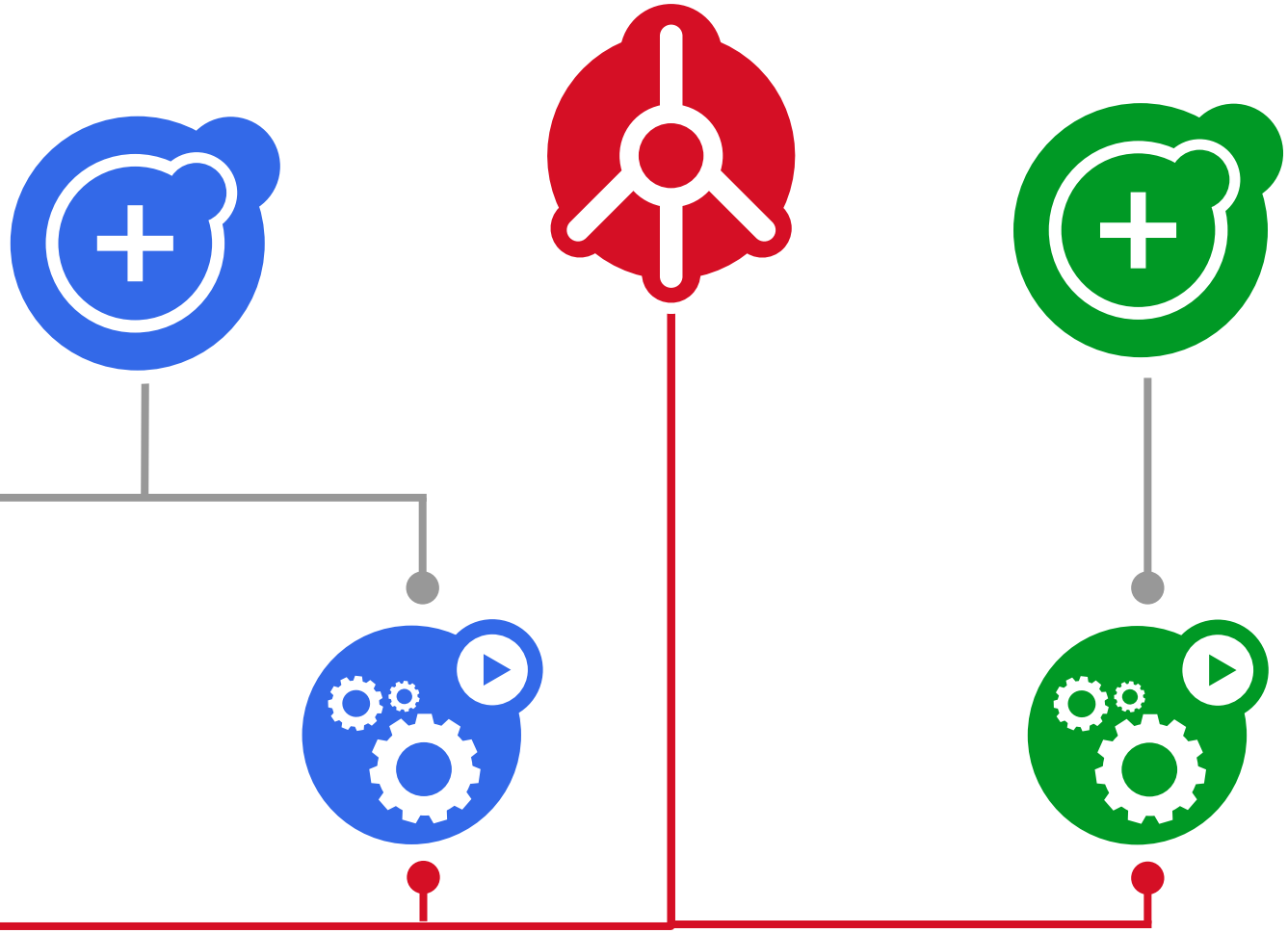


ReplicaSet
- name: my-app-v1
- replicas: 3
- selector:
  - app: MyApp
  - version: v1

ReplicaSet
- name: my-app-v2
- replicas: 1
- selector:
  - app: MyApp
  - version: v2

Rolling Update

Service
- app: MyApp



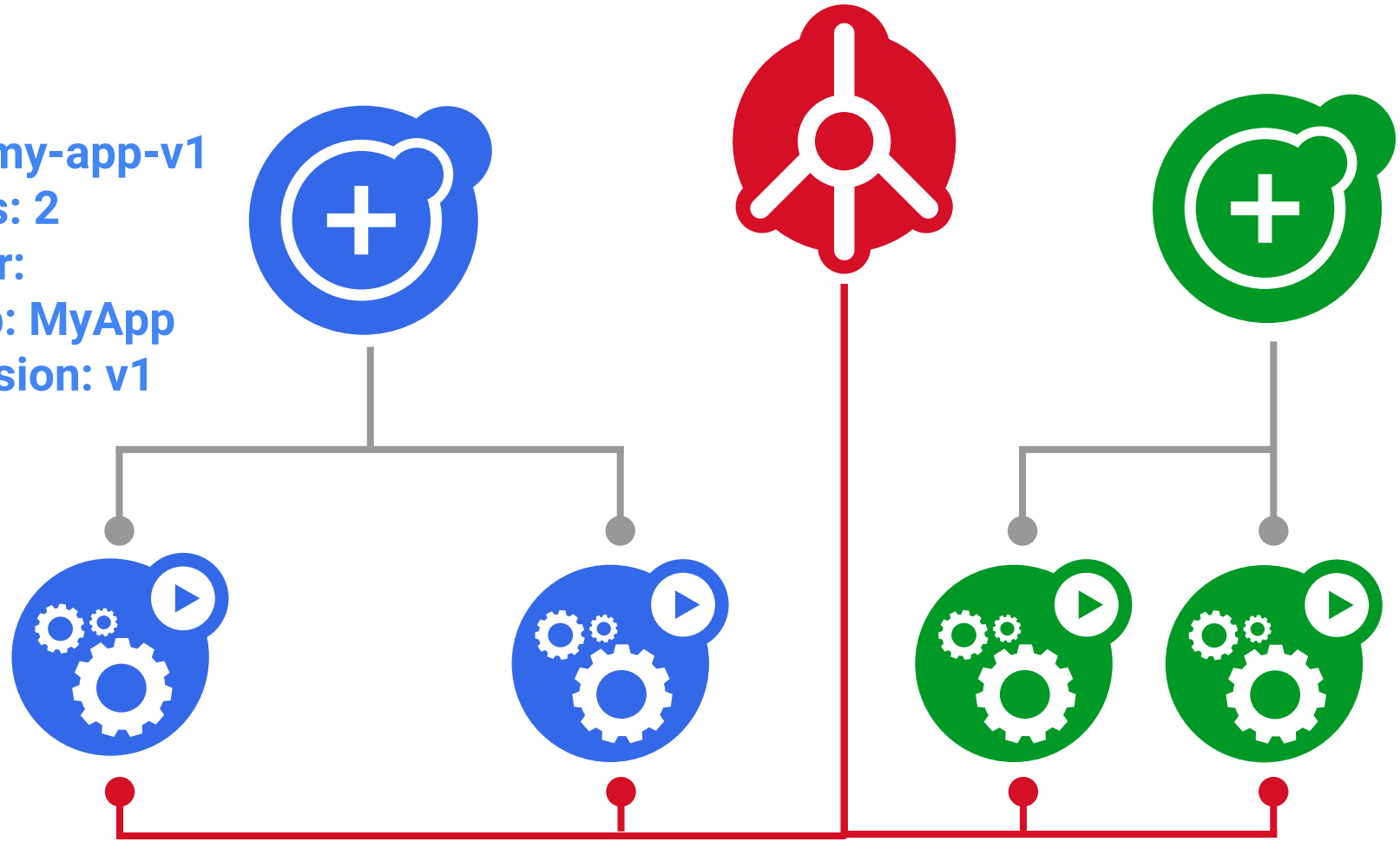
ReplicaSet
- name: my-app-v1
- replicas: 2
- selector:
  - app: MyApp
  - version: v1

ReplicaSet
- name: my-app-v2
- replicas: 1
- selector:
  - app: MyApp
  - version: v2

Rolling Update

Service
- app: MyApp

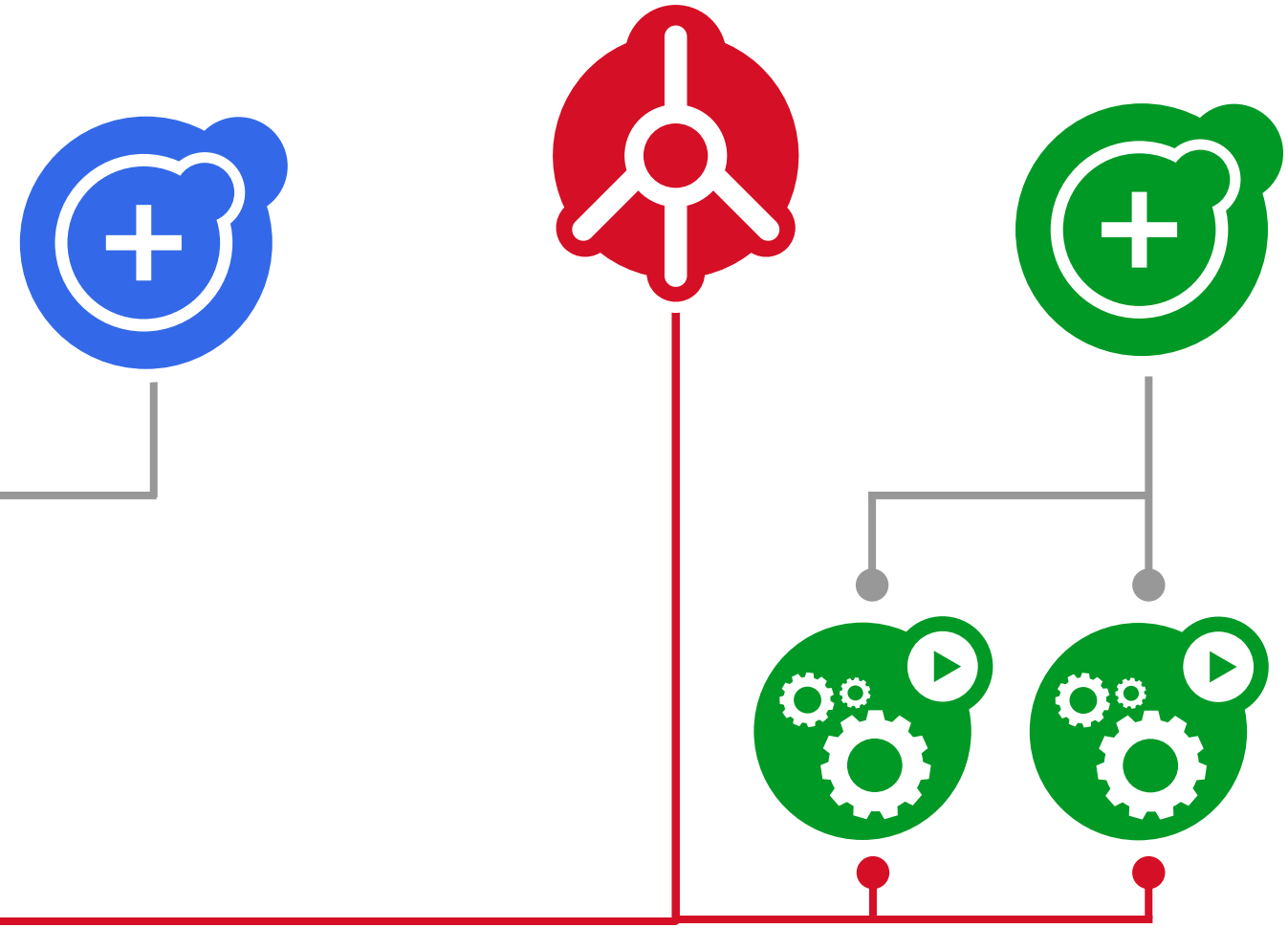
ReplicaSet
- name: my-app-v1
- replicas: 2
- selector:
  - app: MyApp
  - version: v1



ReplicaSet
- name: my-app-v2
- replicas: 2
- selector:
  - app: MyApp
  - version: v2

Rolling Update

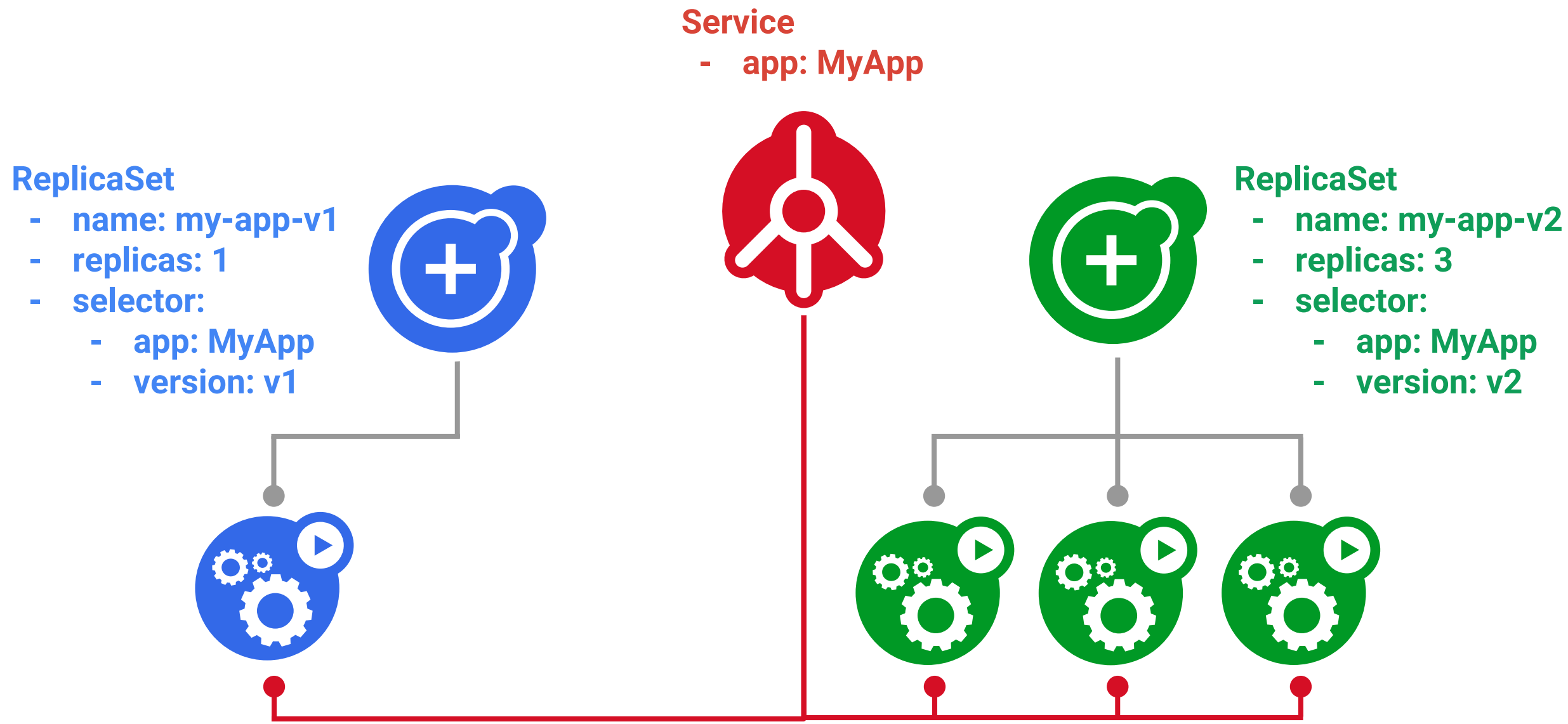
Service
- app: MyApp



ReplicaSet
- name: my-app-v1
- replicas: 1
- selector:
  - app: MyApp
  - version: v1

ReplicaSet
- name: my-app-v2
- replicas: 2
- selector:
  - app: MyApp
  - version: v2

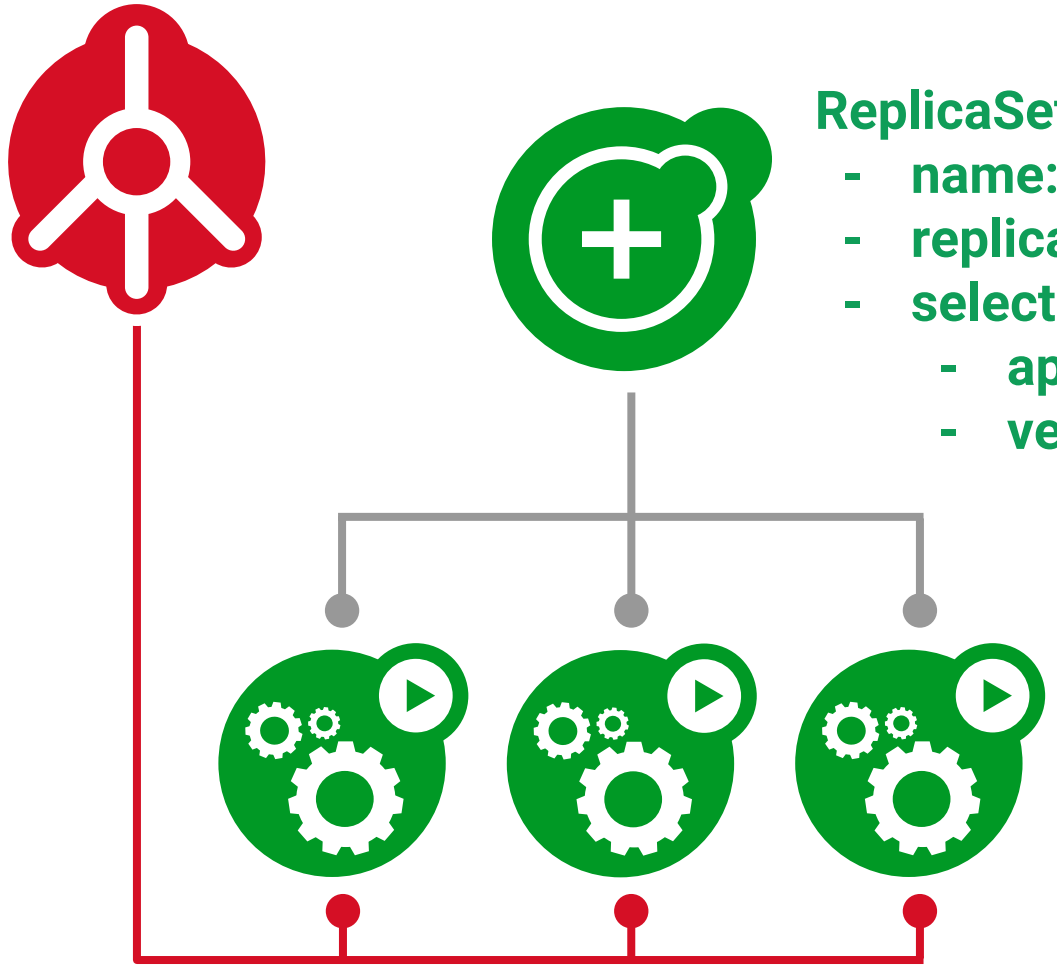
Rolling Update



Rolling Update

Service
- app: MyApp

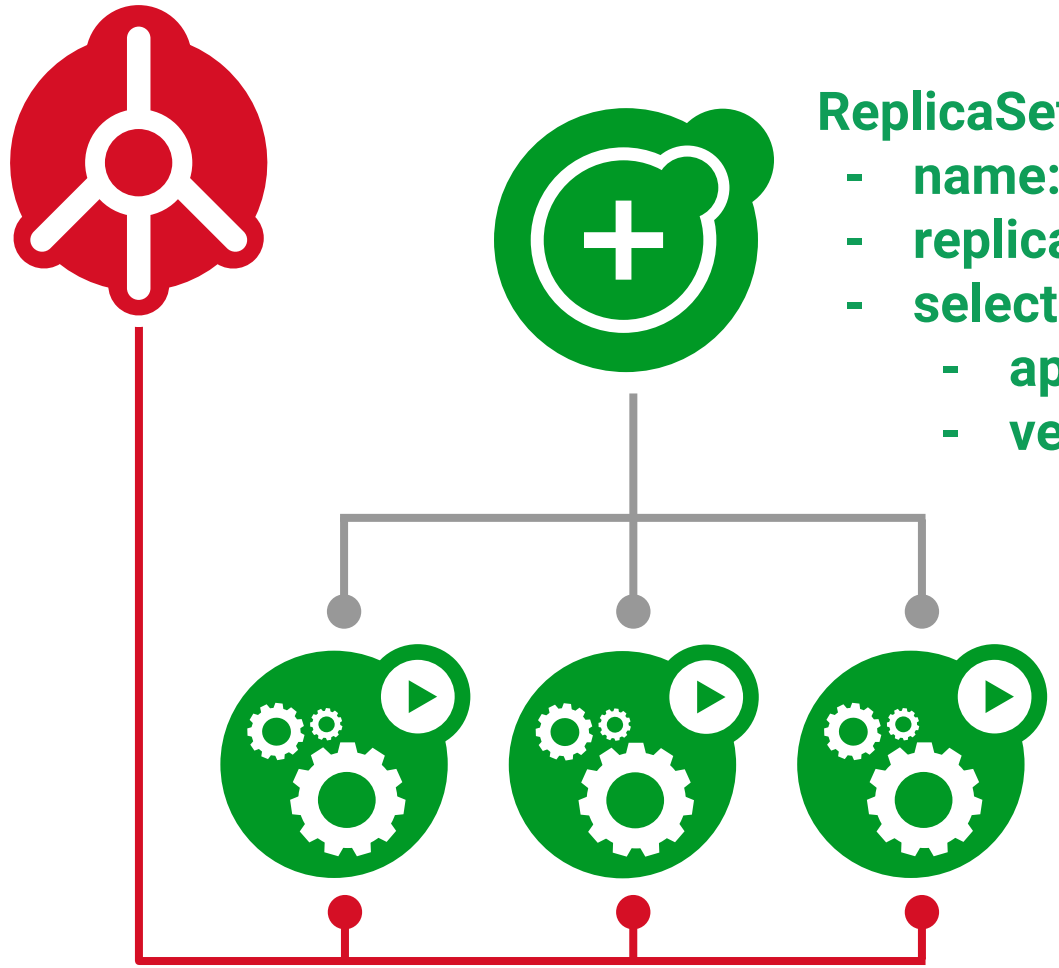
ReplicaSet
- name: my-app-v1
- replicas: 0
- selector:
  - app: MyApp
  - version: v1



ReplicaSet
- name: my-app-v2
- replicas: 3
- selector:
  - app: MyApp
  - version: v2

Rolling Update

Service
- **app: MyApp**



ReplicaSet
- **name: my-app-v2**
- **replicas: 3**
- **selector:**
  - **app: MyApp**
  - **version: v2**

La suite?

Suivez le Kubernetes blog,le YouTube channel & Twitter feed

Comme plus de 11 000+ personnes, suivez le free edX "Introduction to Kubernetes" course

Rejoignez 22 500+ personnes dans le Kubernetes Slack: http://slack.k8s.io

Preparez la certification Certified Kubernetes Administrator

Rejoignez les Special Interest Group et assistez aux réunions hebdomadaires

Installez Kubernetes sur vos machines minikube or kubeadm

Assistez au Kubernetes Community Meeting sur Zoom

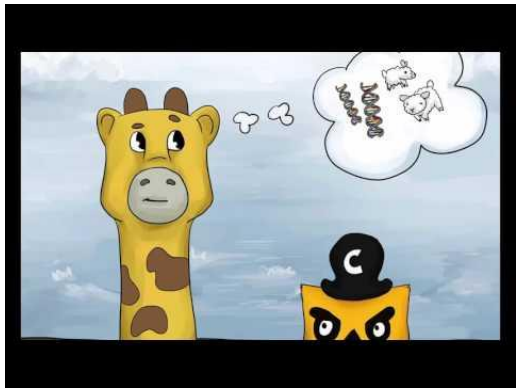
Lisez le document technique Kubernetes ecosystem ebook de The New Stack

Lisez l'excellent, clair et simple ouvrage: "**Kubernetes, Up and Running**"

Merci!

@fjammes on Github

Autres liens



[Kubernetes Community Meeting - Dan Kohn, 18th of May 2017]

[Kubernetes: State of the Project - Tim Hockin, 2nd June 2017]

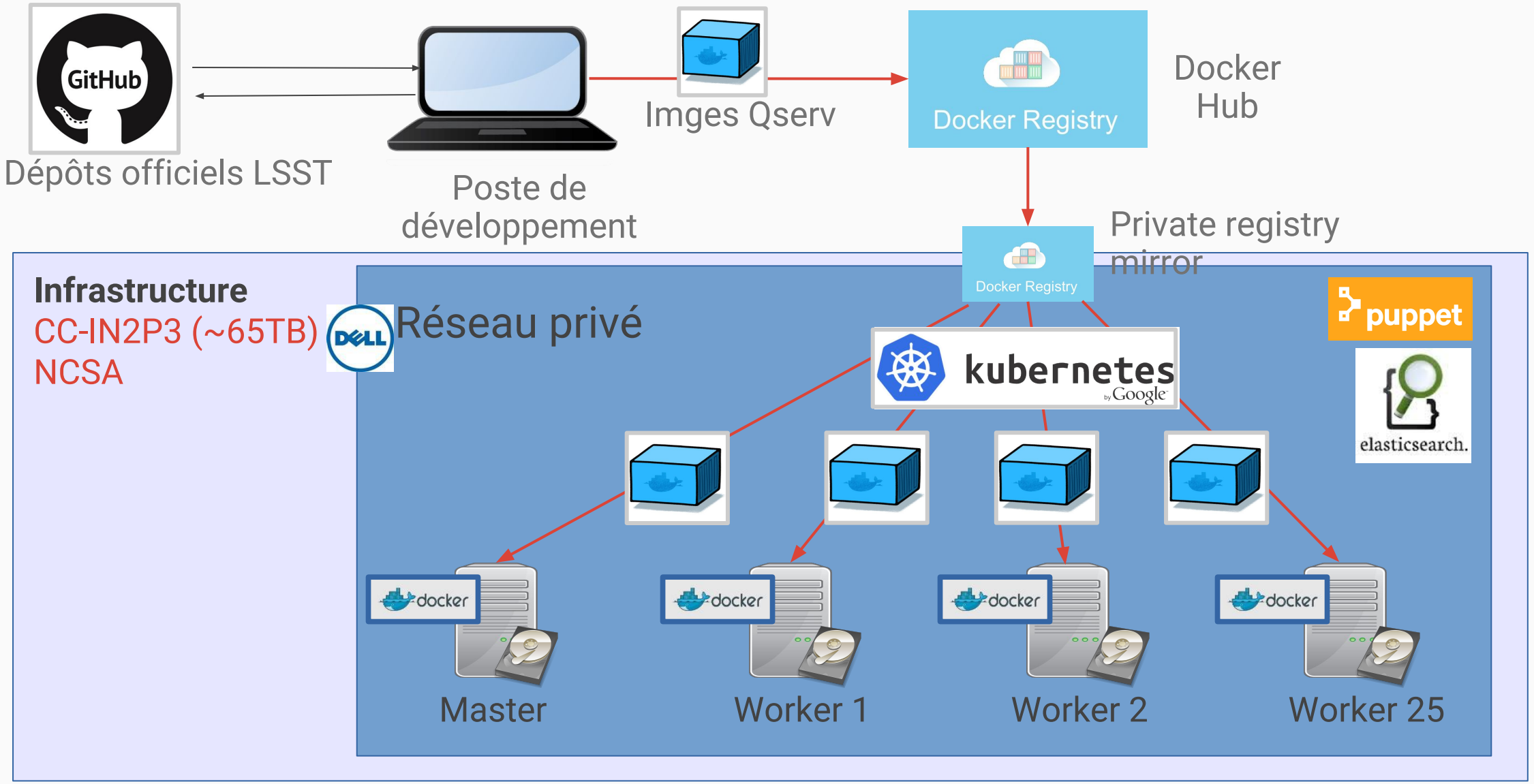
[Migrating Legacy Monoliths to Cloud Native Microservices Architectures on Kubernetes - Dan Kohn]

[CNCF Overview - Dan Kohn]

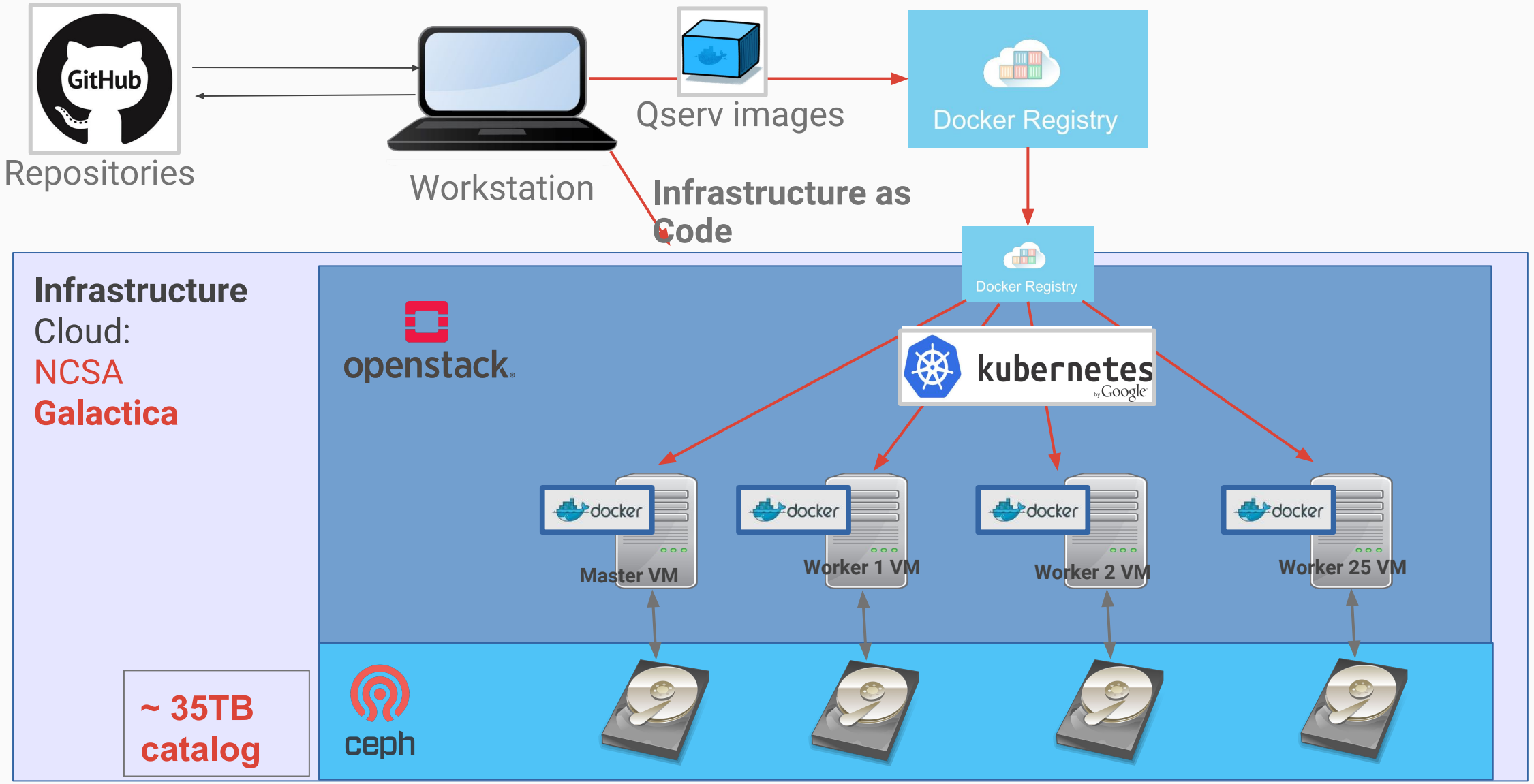
[Kubernetes Growth and Community - Chris Gaun, Sept 15 2016]

Et l'excellente: [introduction to Kubernetes video]

Déploiement automatisé: bare-metal

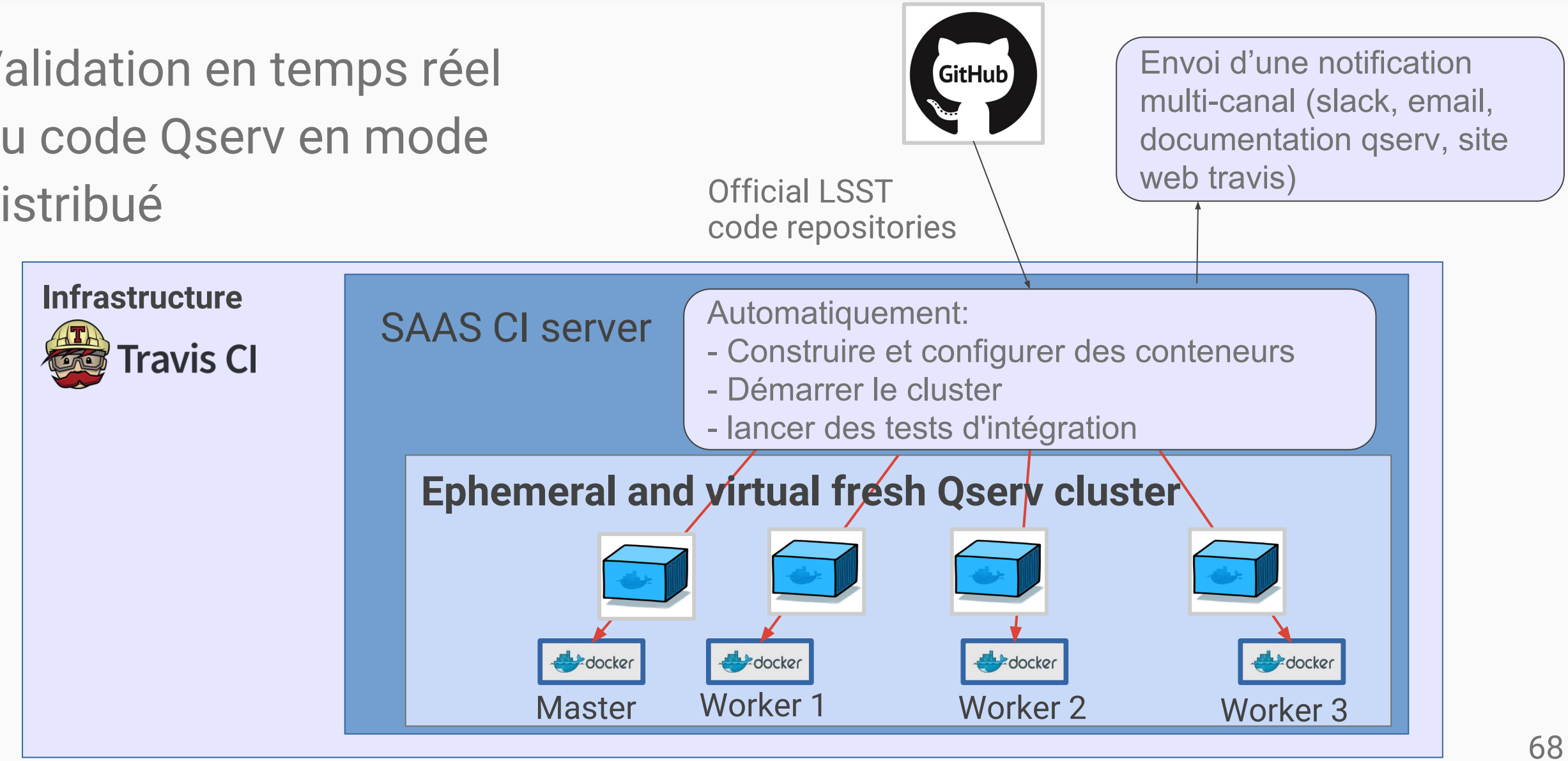


Déploiement automatisé: Openstack+CEPH



Déploiement automatisé: intégration continue

Validation en temps réel du code Qserv en mode distribué



Deployments

Deployments

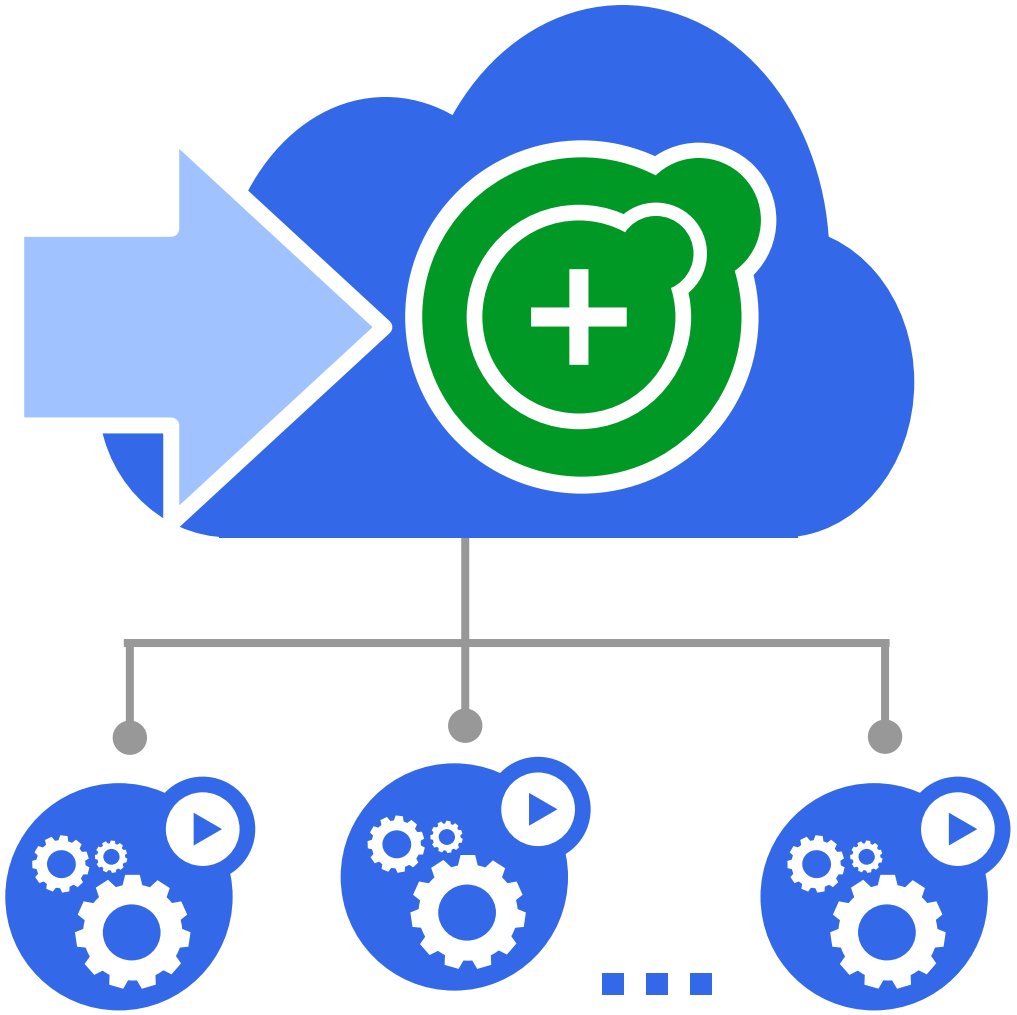
Updates-as-a-service

- Rolling update is imperative, client-side

Deployment manages rollouts for you

- stable object name in API
- updates are configurable, server-side
- `kubectl edit`, `kubectl apply`, etc.

Can have multiple updates in flight



Services

Services

A group of **pods that work together**

- grouped by a selector

Defines access policy

- “load balanced” or “headless”, external IP, etc.

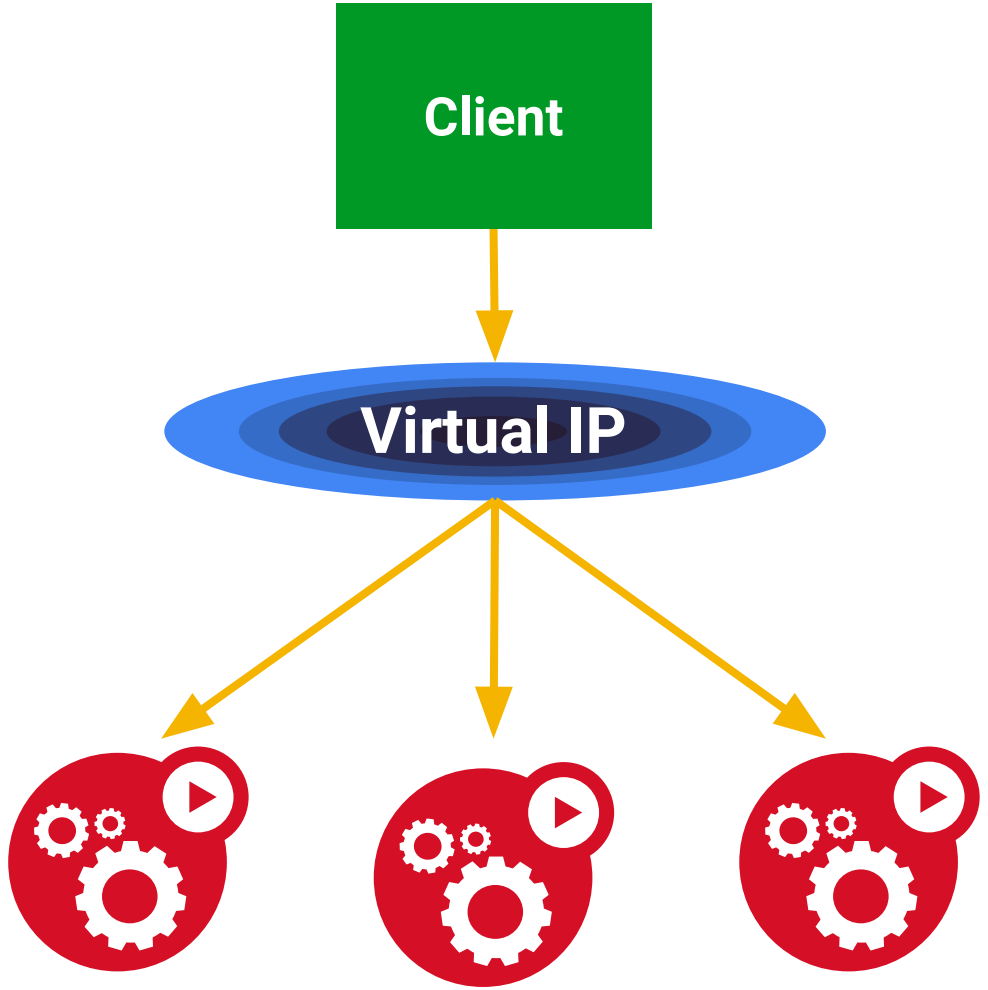
Can have a stable **virtual IP** and port(s)

- also a DNS name

VIP is managed by *kube-proxy*

- watches all services
- updates iptables when backends change
- default implementation - can be replaced!

Hides complexity



External services

Services VIPs are only available **inside** the cluster

Need to receive traffic from “the outside world”

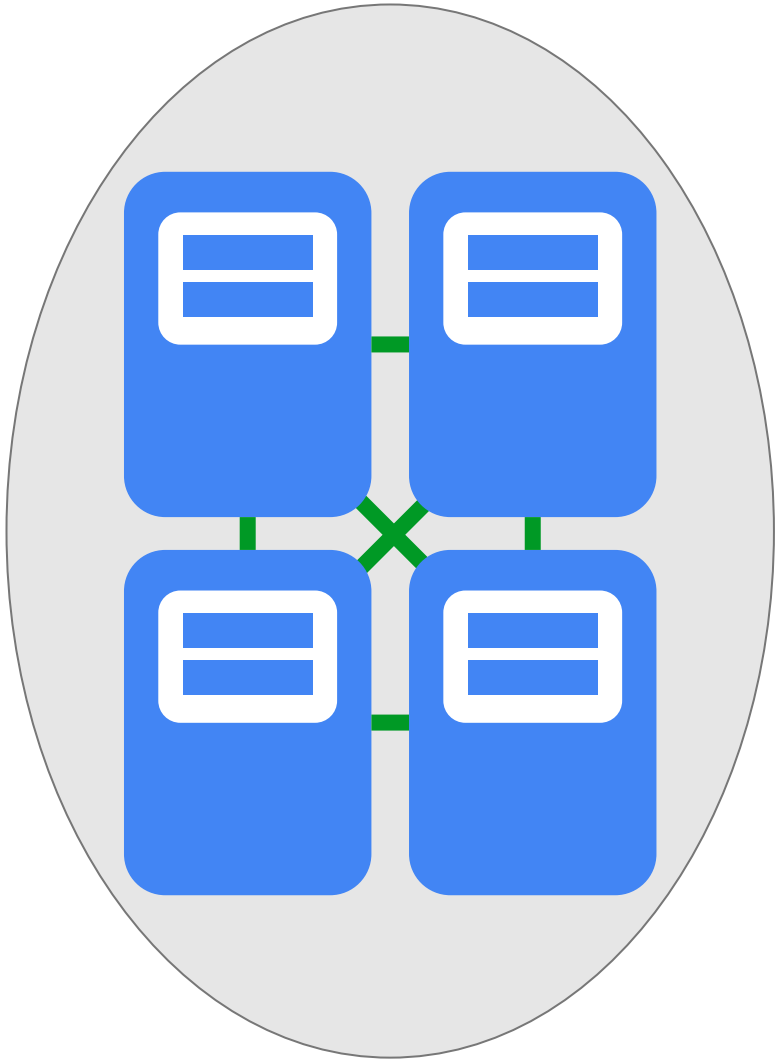
Service “type”

- NodePort: expose on a port on every node
- LoadBalancer: provision a cloud load-balancer

DiY load-balancer solutions

- socat (for NodePort remapping)
- haproxy
- nginx

Ingress (L7 LB)



Ingress (L7 LB)

Many apps are HTTP/HTTPS

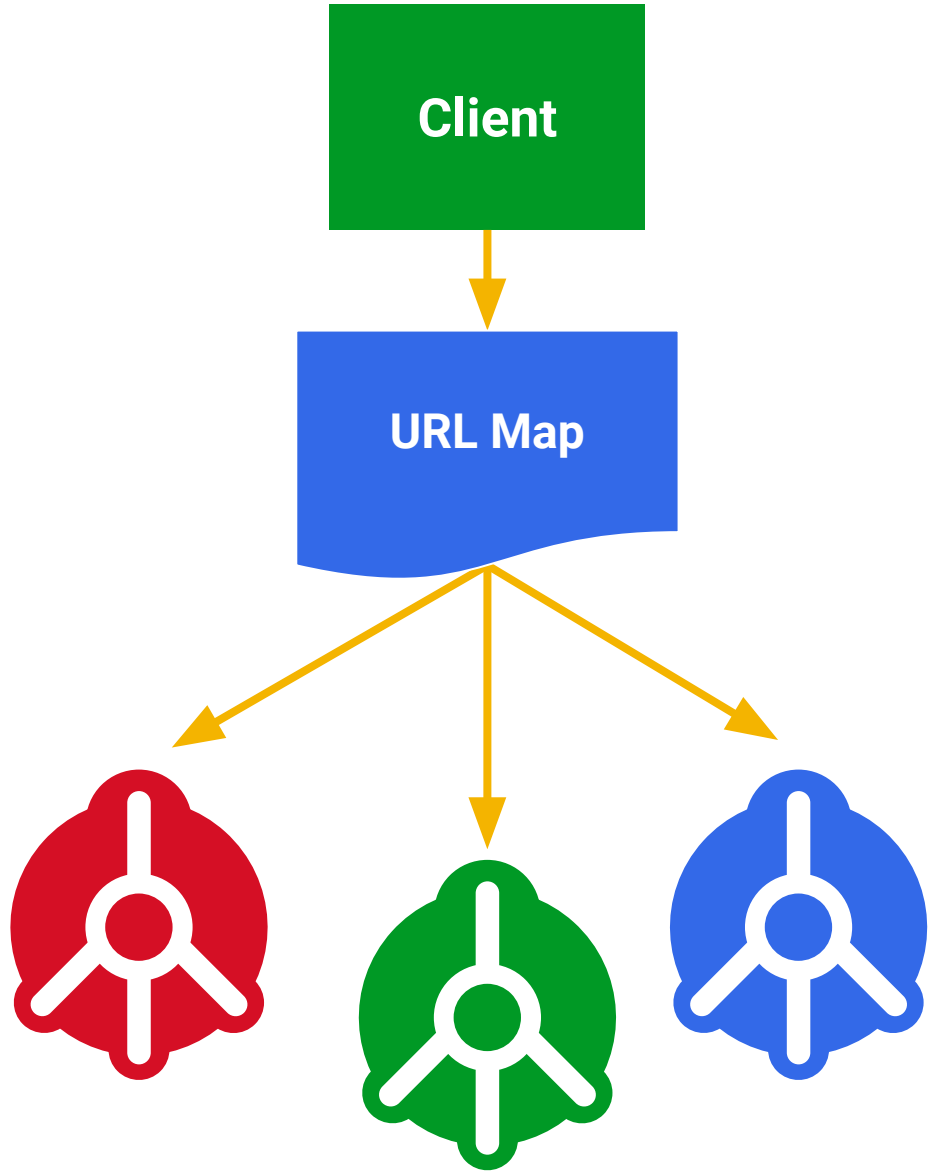
Services are L4 (IP + port)

Ingress maps incoming traffic to backend services

- by HTTP host headers
- by HTTP URL paths

HAProxy, NGINX, AWS and GCE implementations in progress

Now with SSL!



Running Daemons

DaemonSets

Problem: how to run a Pod on every node?

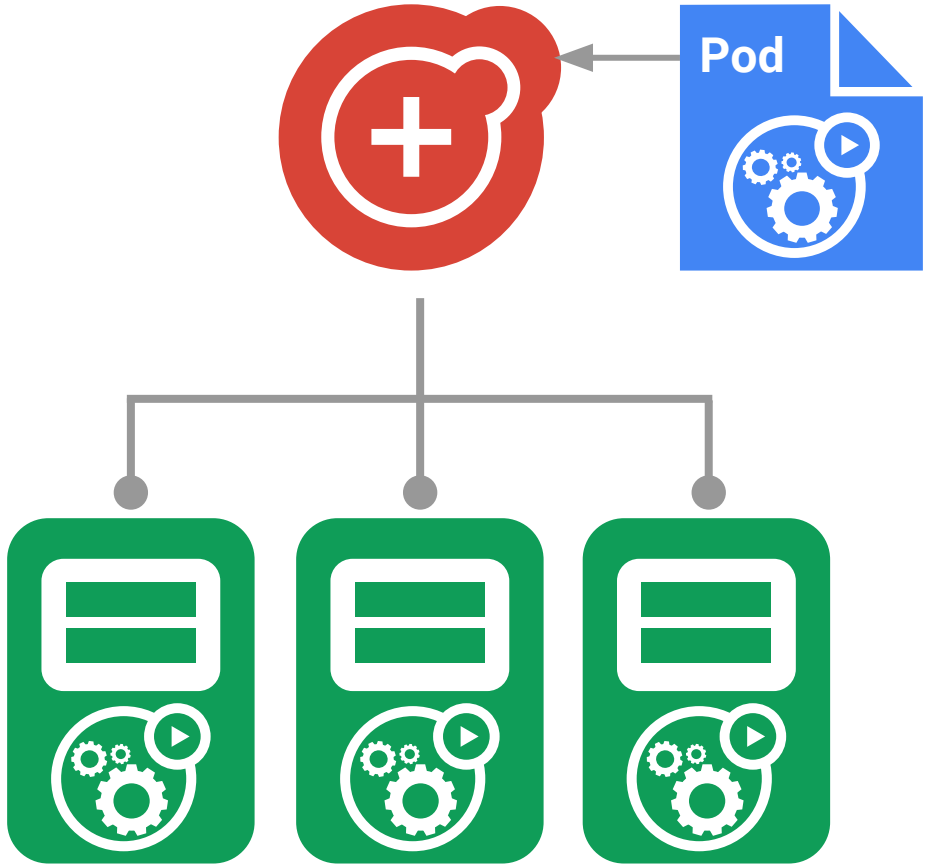
- or a subset of nodes

Similar to ReplicaSet

- principle: do one thing, don't overload

"Which nodes?" is a selector

Uses consistent tools and patterns



Running Jobs

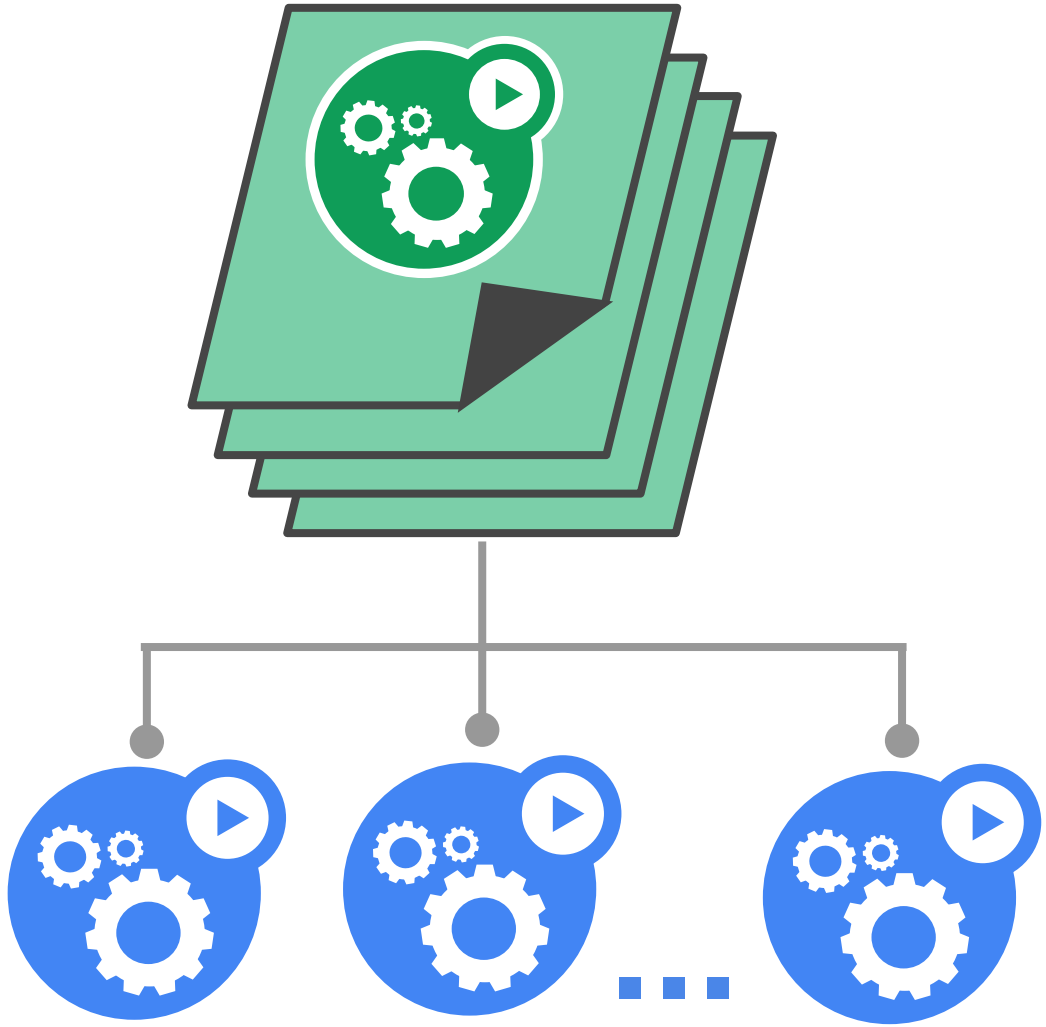
Jobs

Run-to-completion, as opposed to run-forever

- Express parallelism & required completions
- Workflow: restart on failure
- Build/test: don't restart on failure

Aggregates success/failure counts

Built for batch and big-data work



Running With State

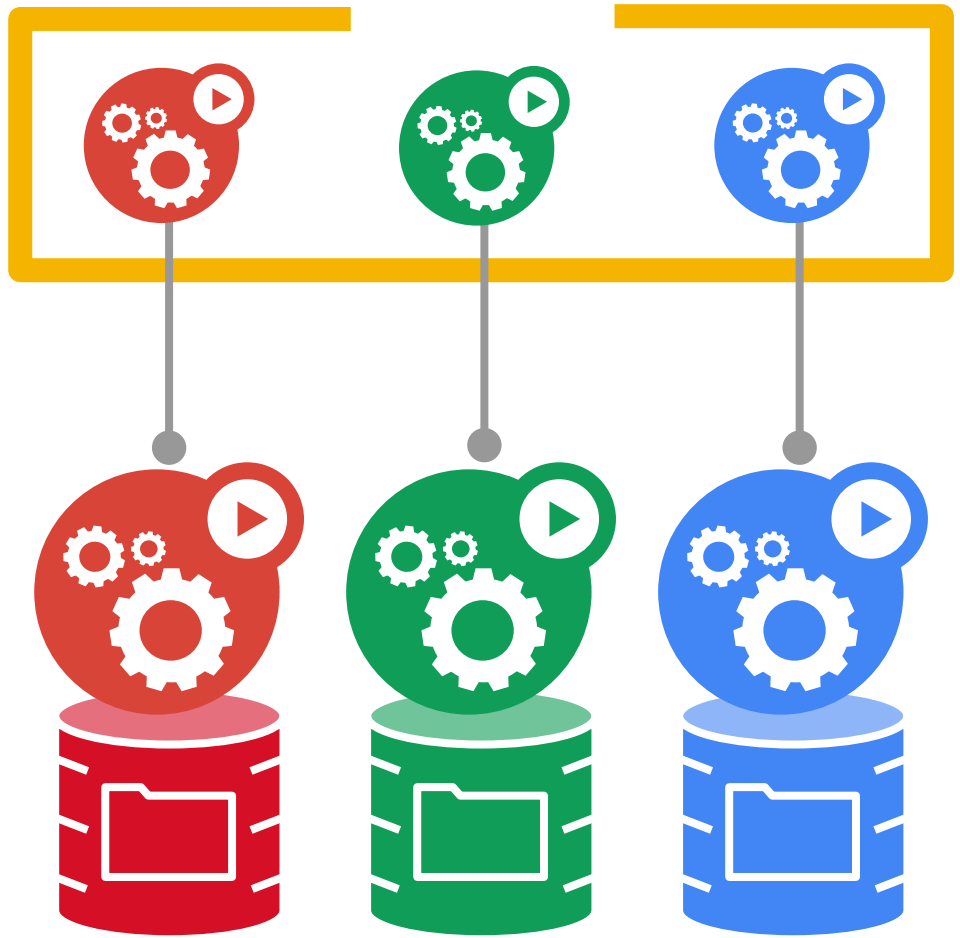
StatefulSets

Goal: enable clustered software on Kubernetes

- mysql, redis, zookeeper, ...

Clustered apps need “identity” and have sequencing constraints

- stable hostname, available in DNS
- an ordinal index
- stable storage: linked to the ordinal & hostname
- discovery of peers for quorum
- startup/teardown ordering



ConfigMaps

ConfigMaps

Goal: manage app configuration

- ...without making overly-brittle container images

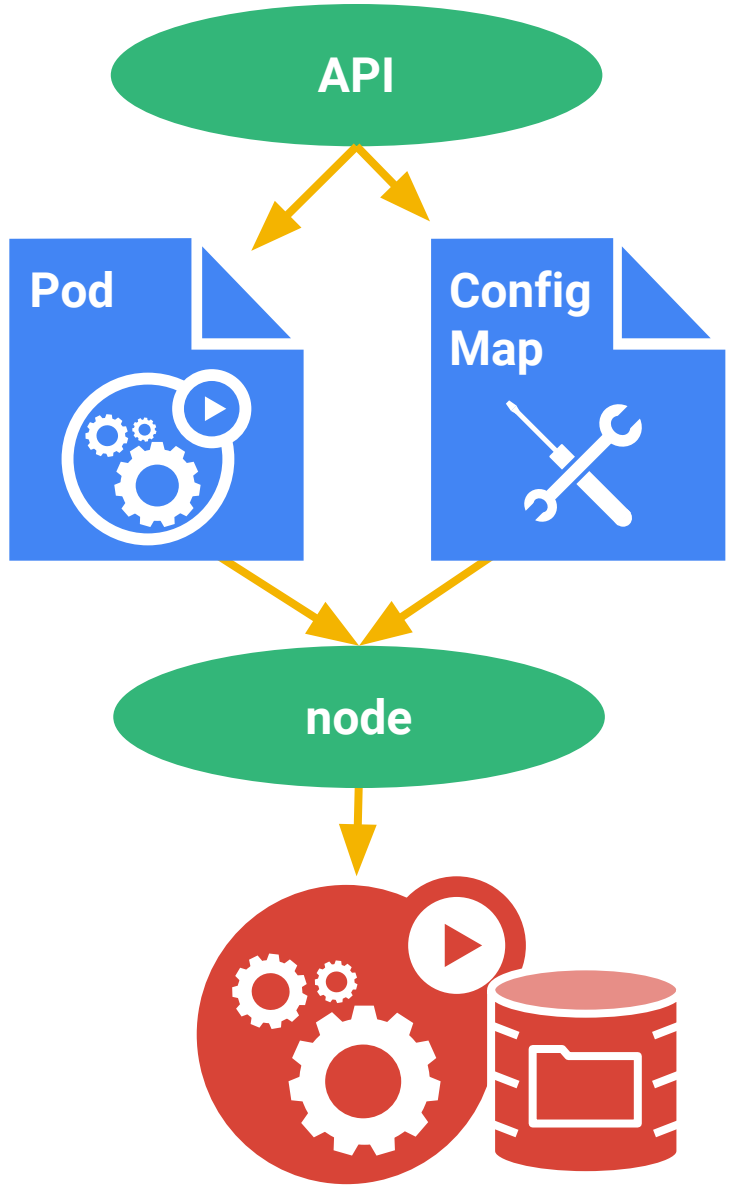
12-factor says config comes from the environment

- Kubernetes is the environment

Manage config via the Kubernetes API

Inject config as a virtual volume into your Pods

- late-binding, live-updated (atomic)
- also available as env vars



Secrets

Secrets

Goal: grant a pod access to a secured *something*

- don't put secrets in the container image!

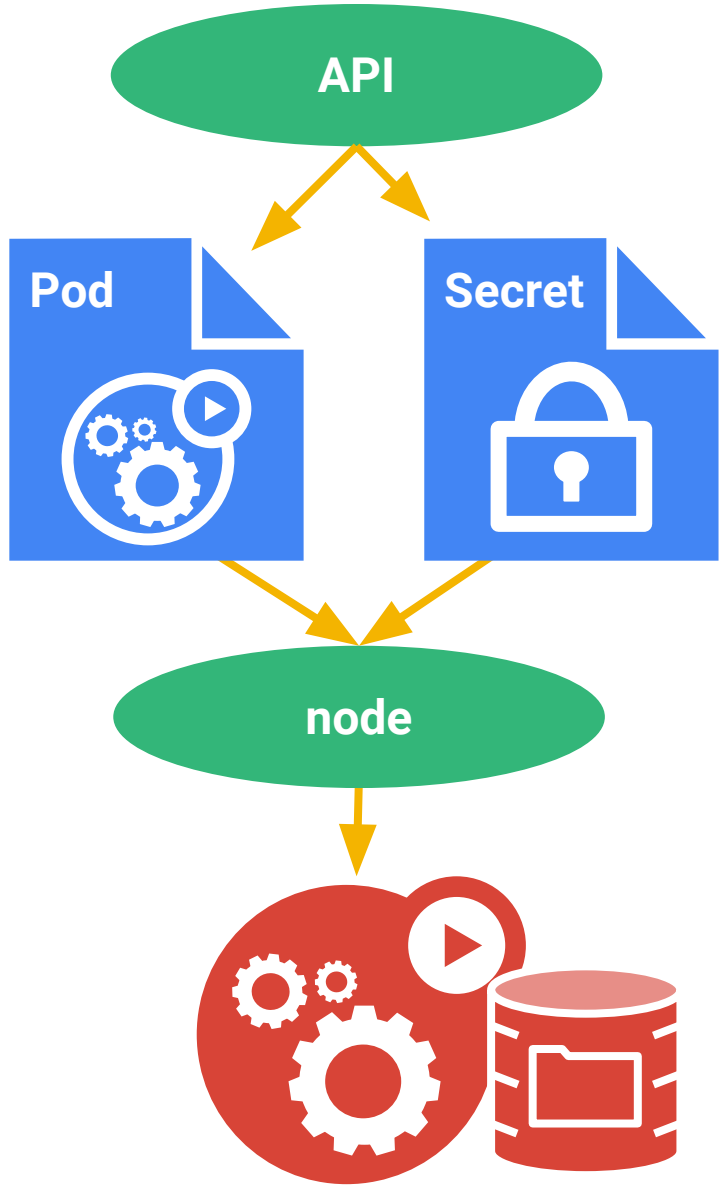
12-factor says config comes from the environment

- Kubernetes is the environment

Manage secrets via the Kubernetes API

Inject secrets as virtual volumes into your Pods

- late-binding, tmpfs - never touches disk
- also available as env vars



Graceful Termination

Graceful Termination

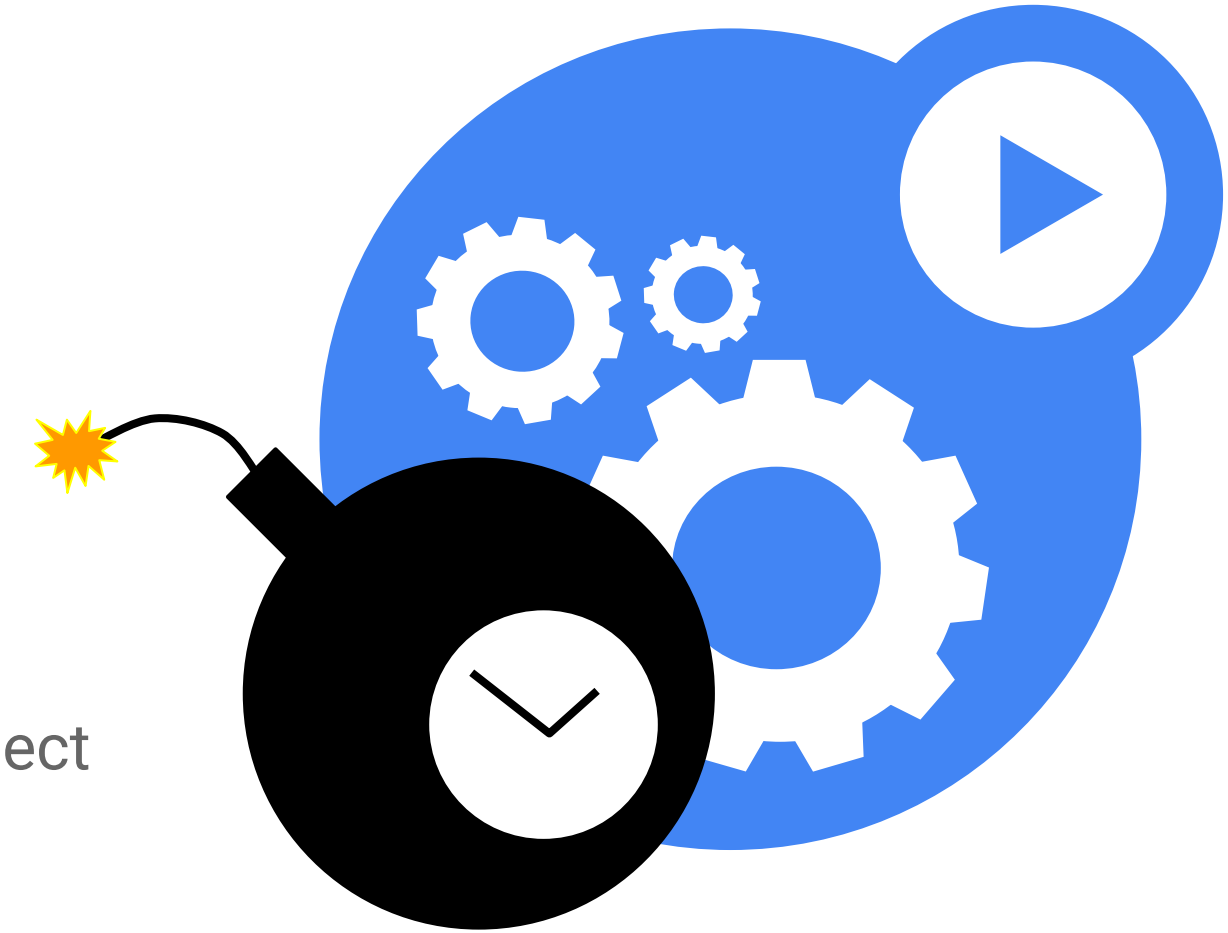
Goal: Give pods time to clean up

- finish in-flight operations
- log state
- flush to disk
- 30 seconds by default

Catch **`SIGTERM`**, cleanup, exit ASAP

Pod status “Terminating”

Declarative: ‘`DELETE`’ appears as an object field in the API



Auto-Scaling

HorizontalPodAutoScalers

Goal: Automatically scale pods as needed

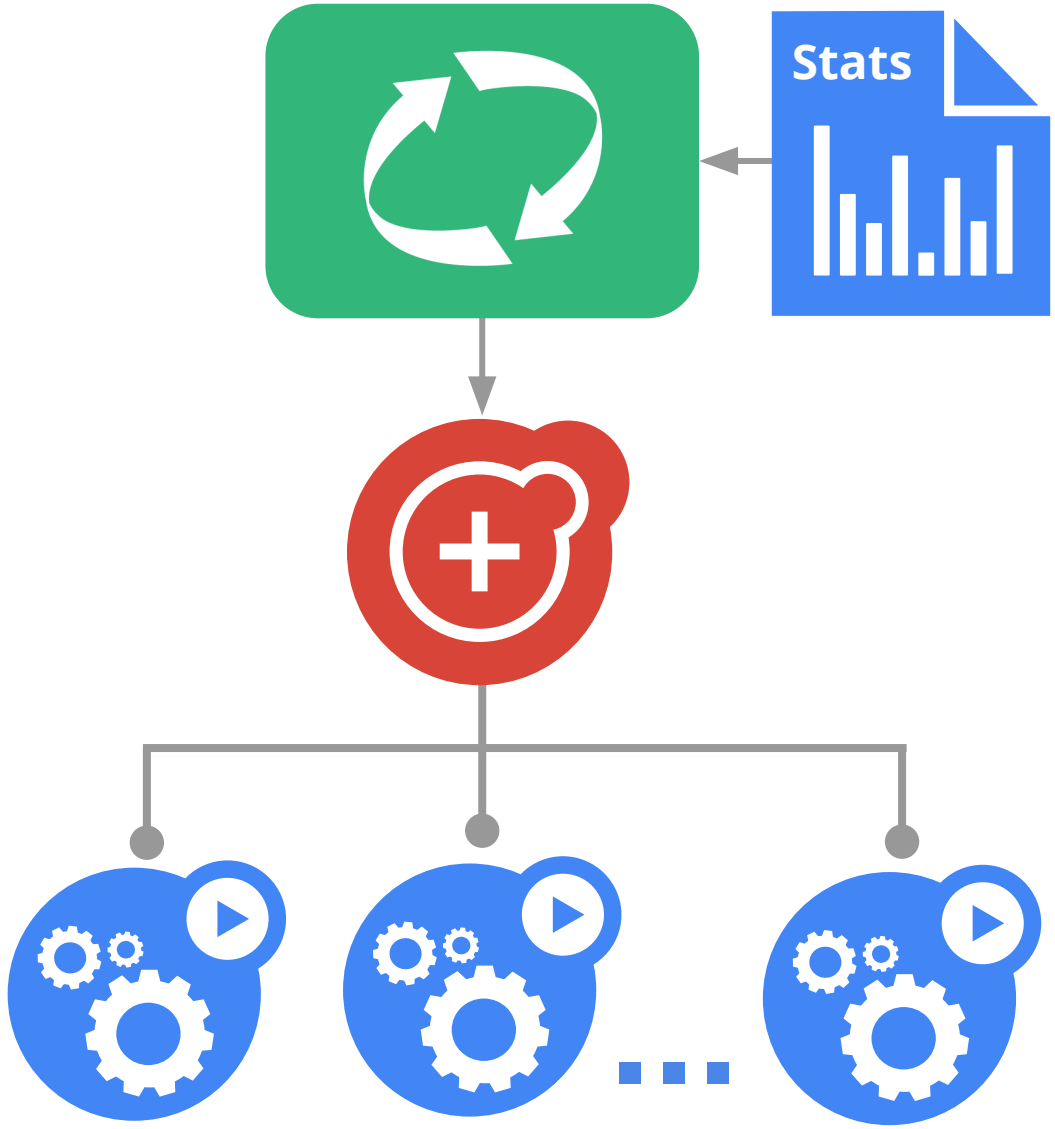
- based on CPU utilization (for now)
- custom metrics in Alpha

Efficiency now, capacity when you need it

Operates within user-defined min/max bounds

Set it and forget it

Status: Stable in kubernetes 1.9, see Heapster



PersistentVolumes

PersistentVolumes

A higher-level storage abstraction

- insulation from any one cloud environment

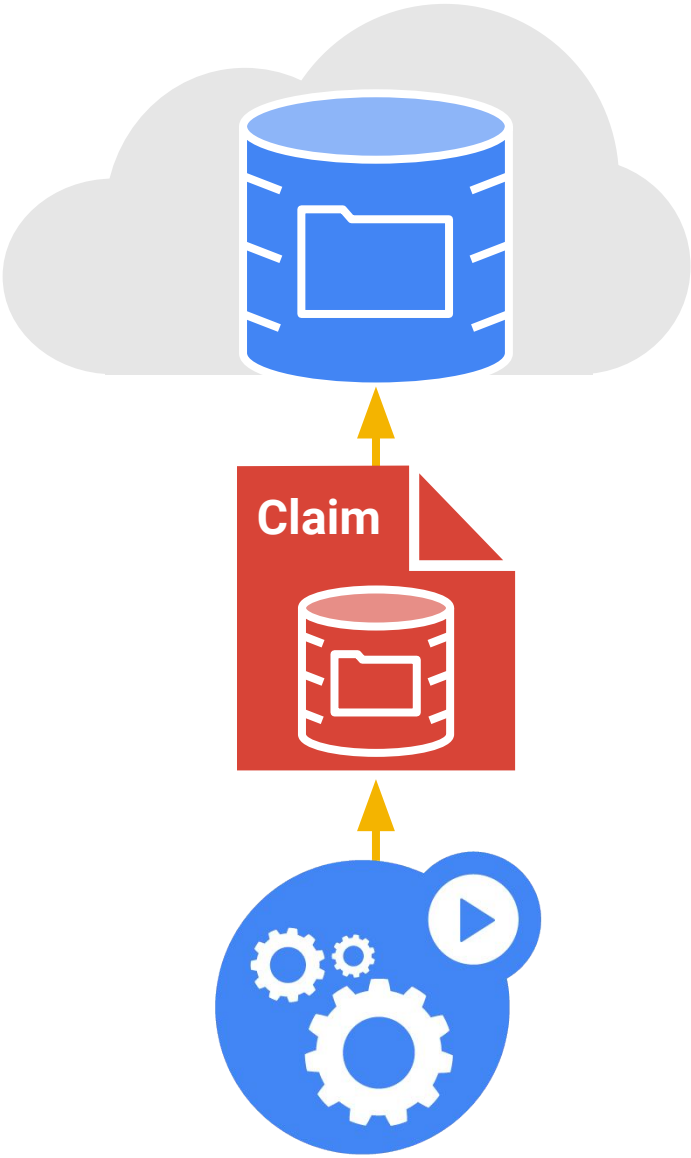
Admin provisions them, users claim them

- **NEW: auto-provisioning**

Independent lifetime from consumers

- lives until user is done with it
- can be handed-off between pods

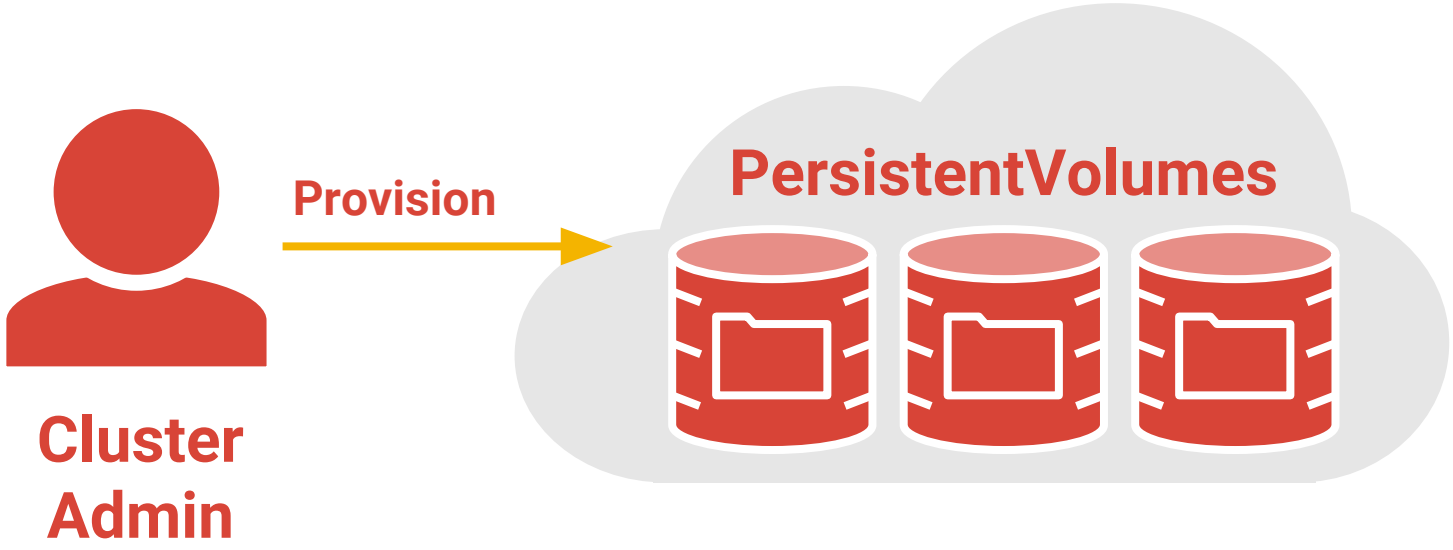
Dynamically “scheduled” and managed, like nodes and pods



PersistentVolumes

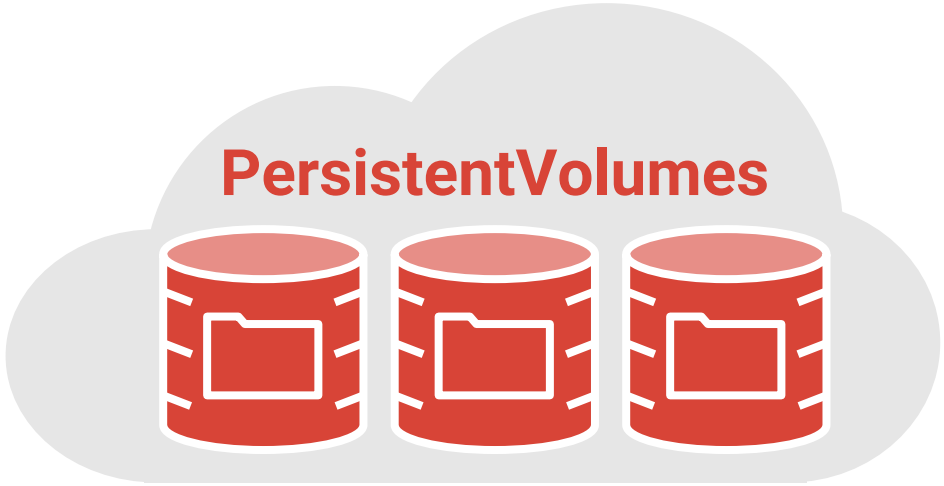
Cluster Admin

PersistentVolumes



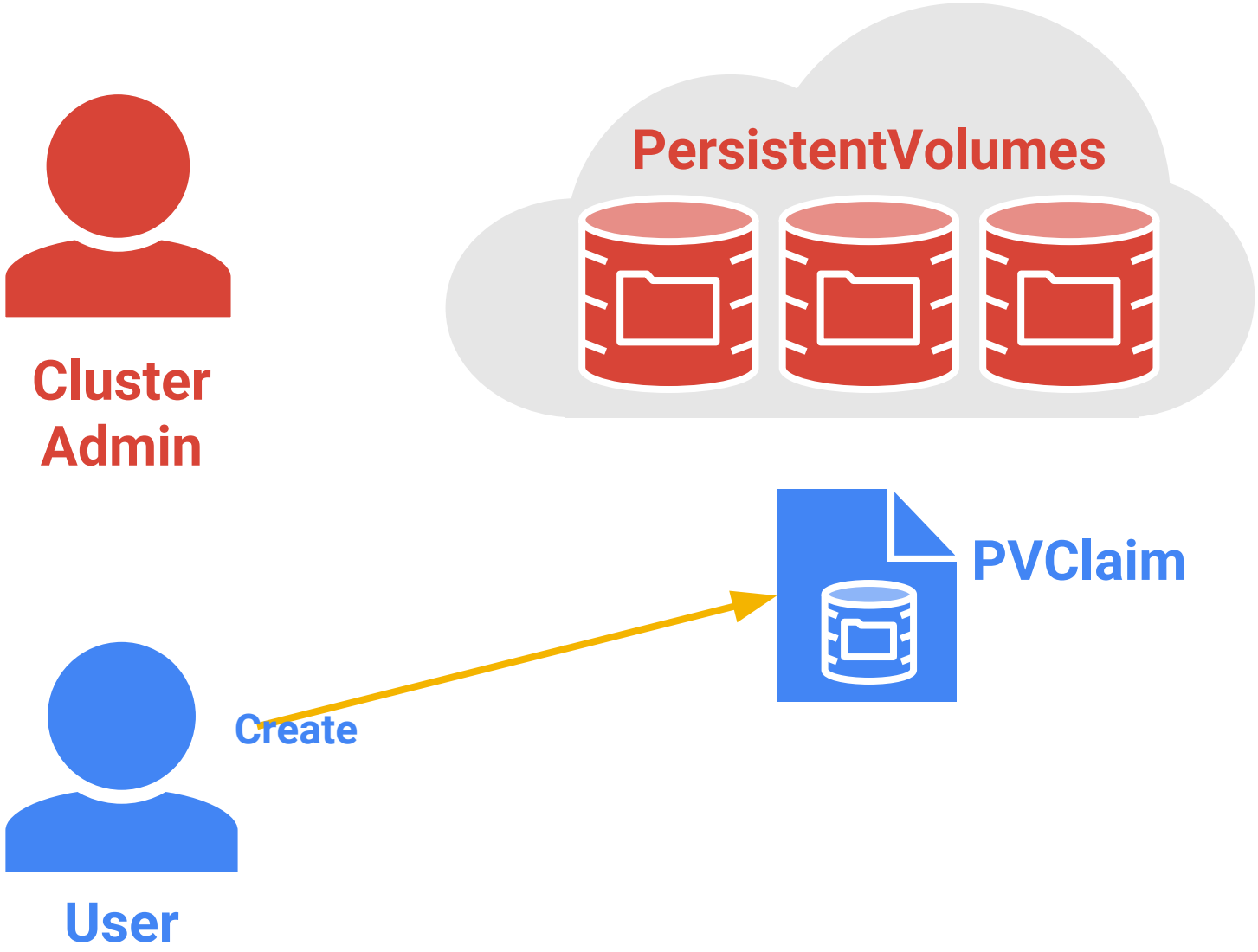
PersistentVolumes

Cluster Admin

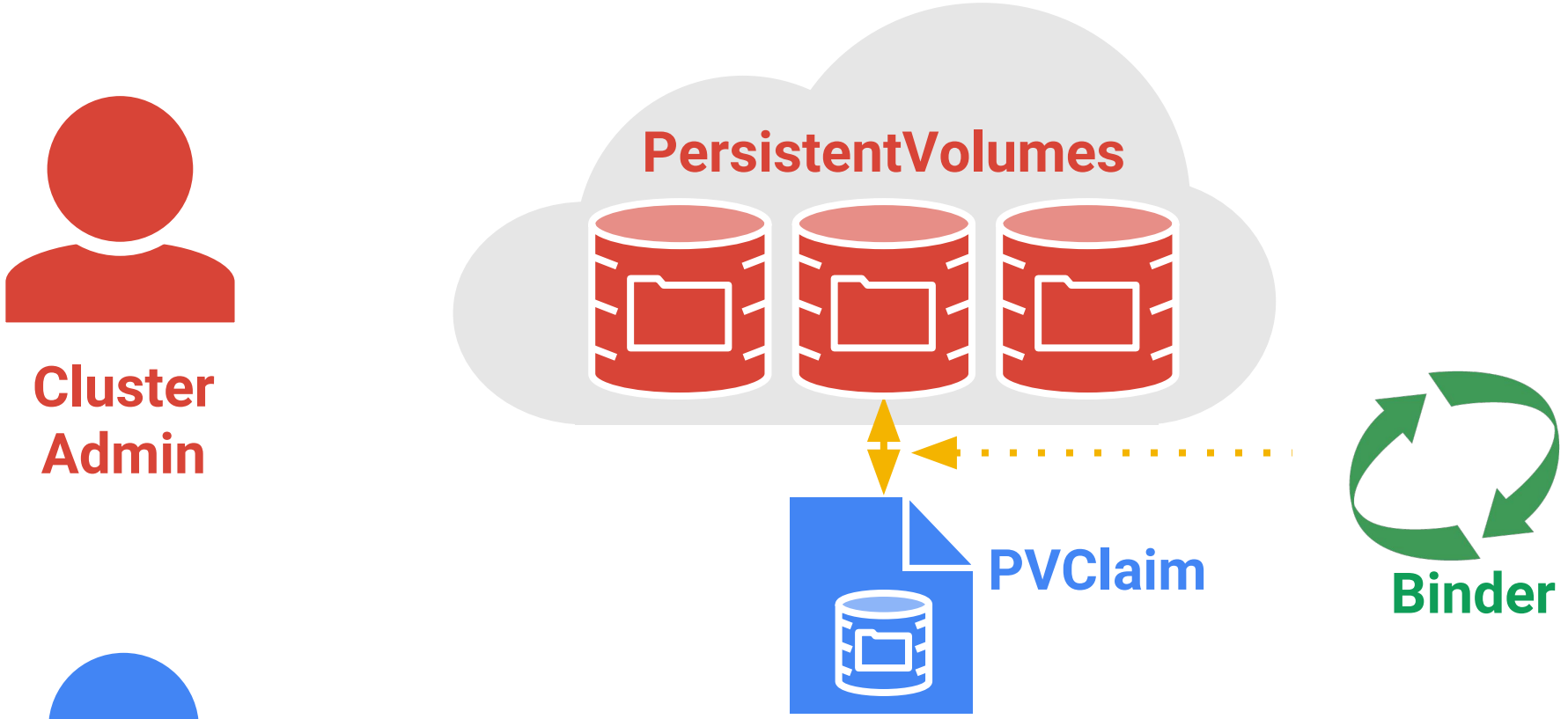


User

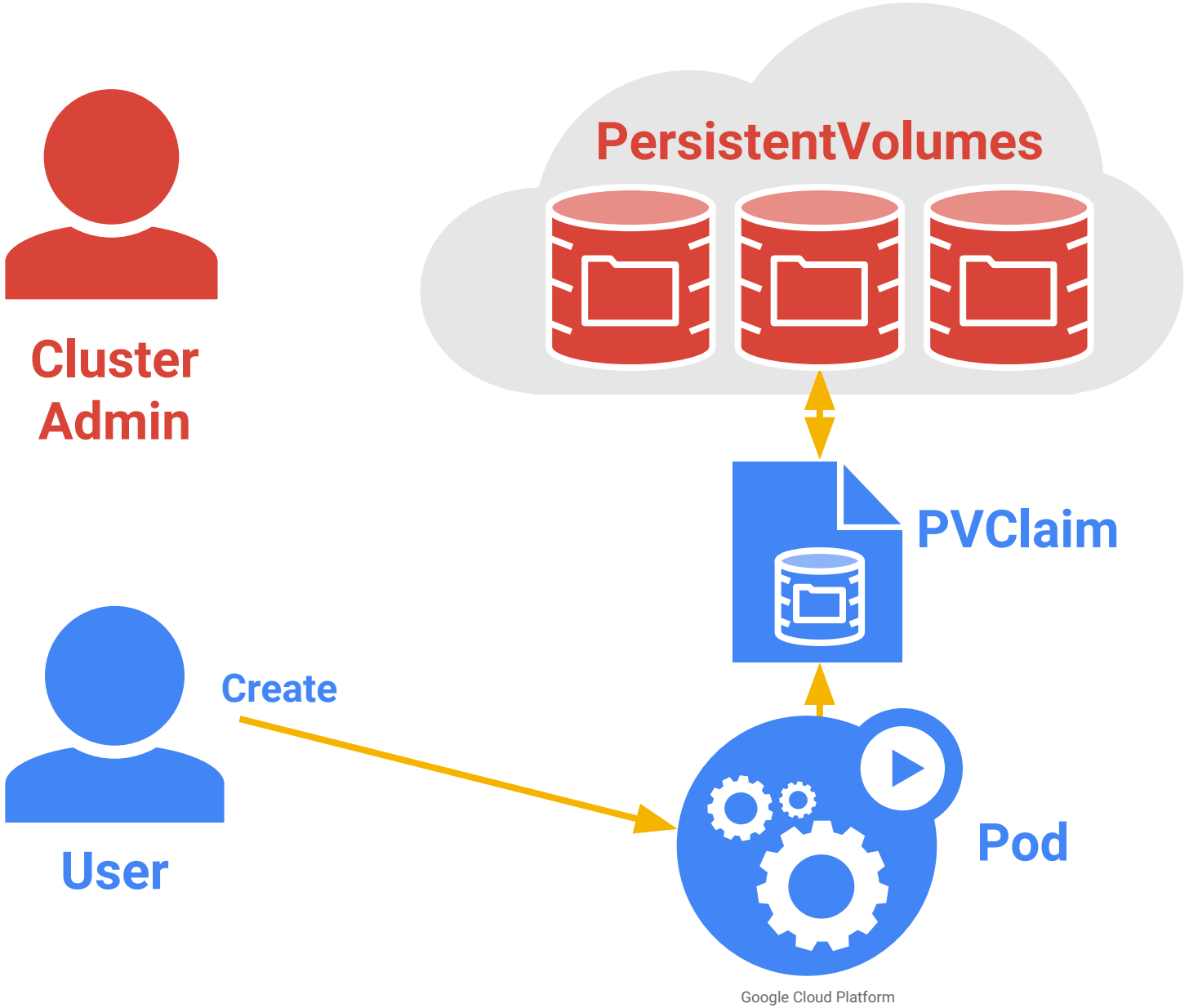
PersistentVolumes



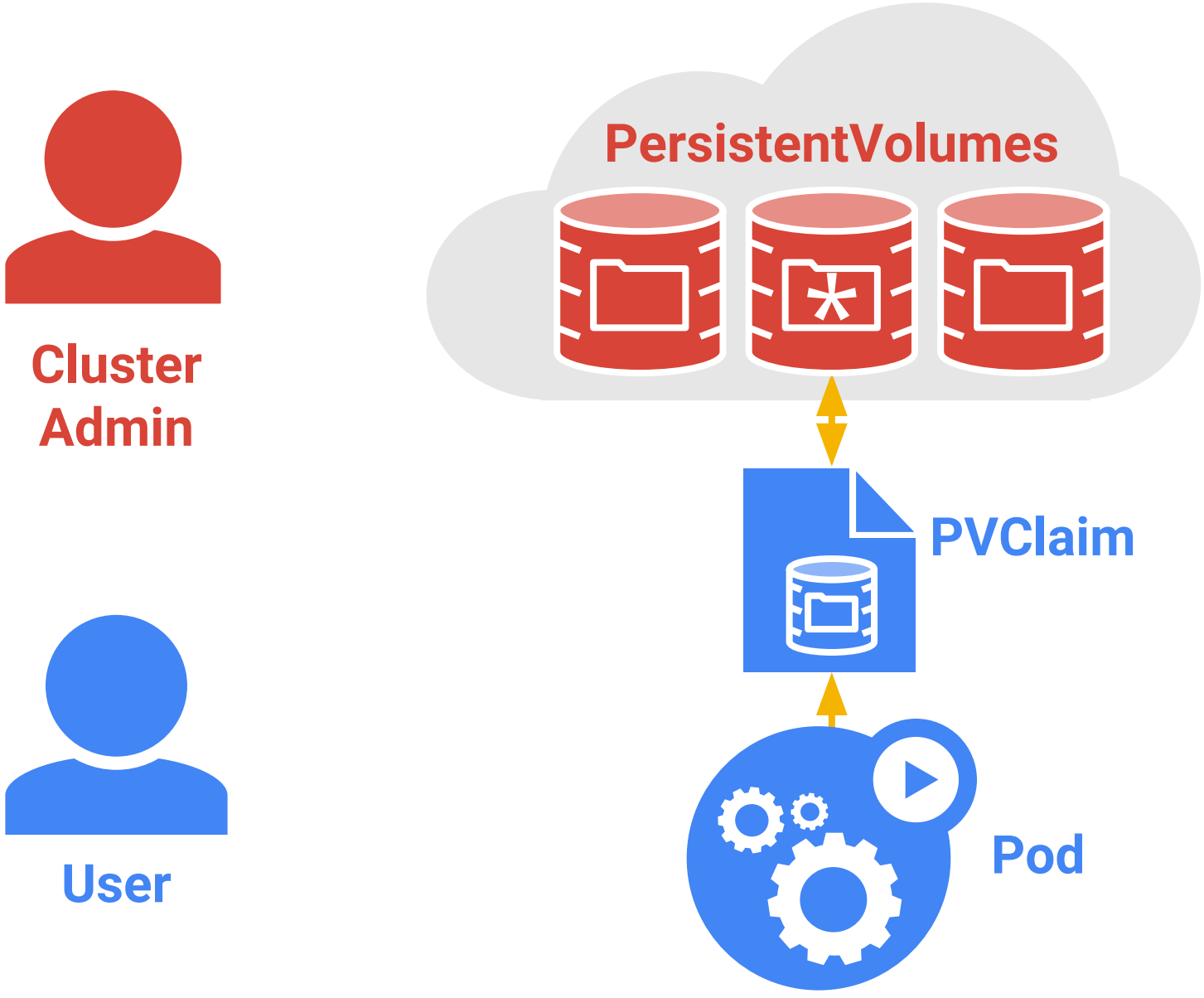
PersistentVolumes



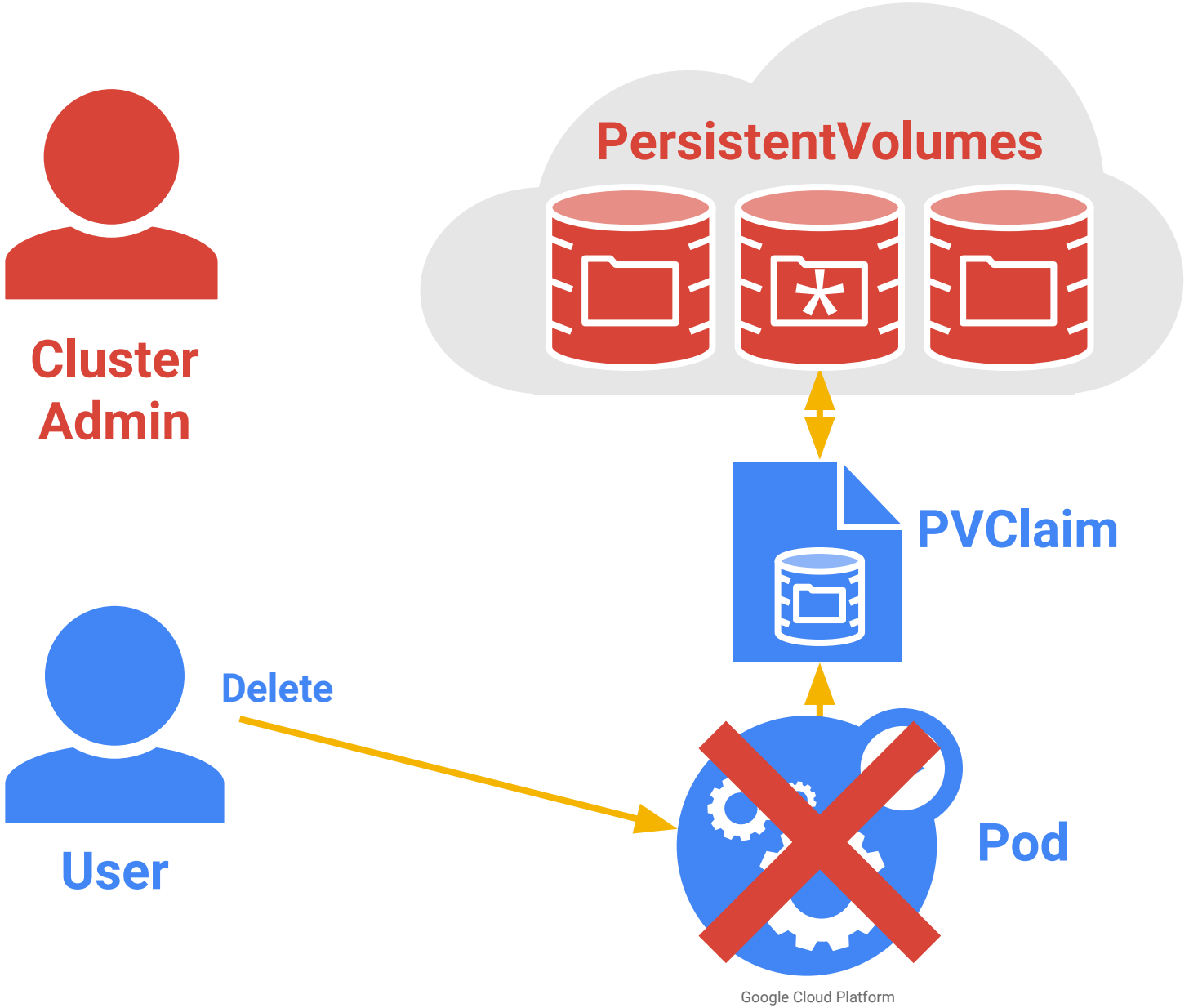
PersistentVolumes



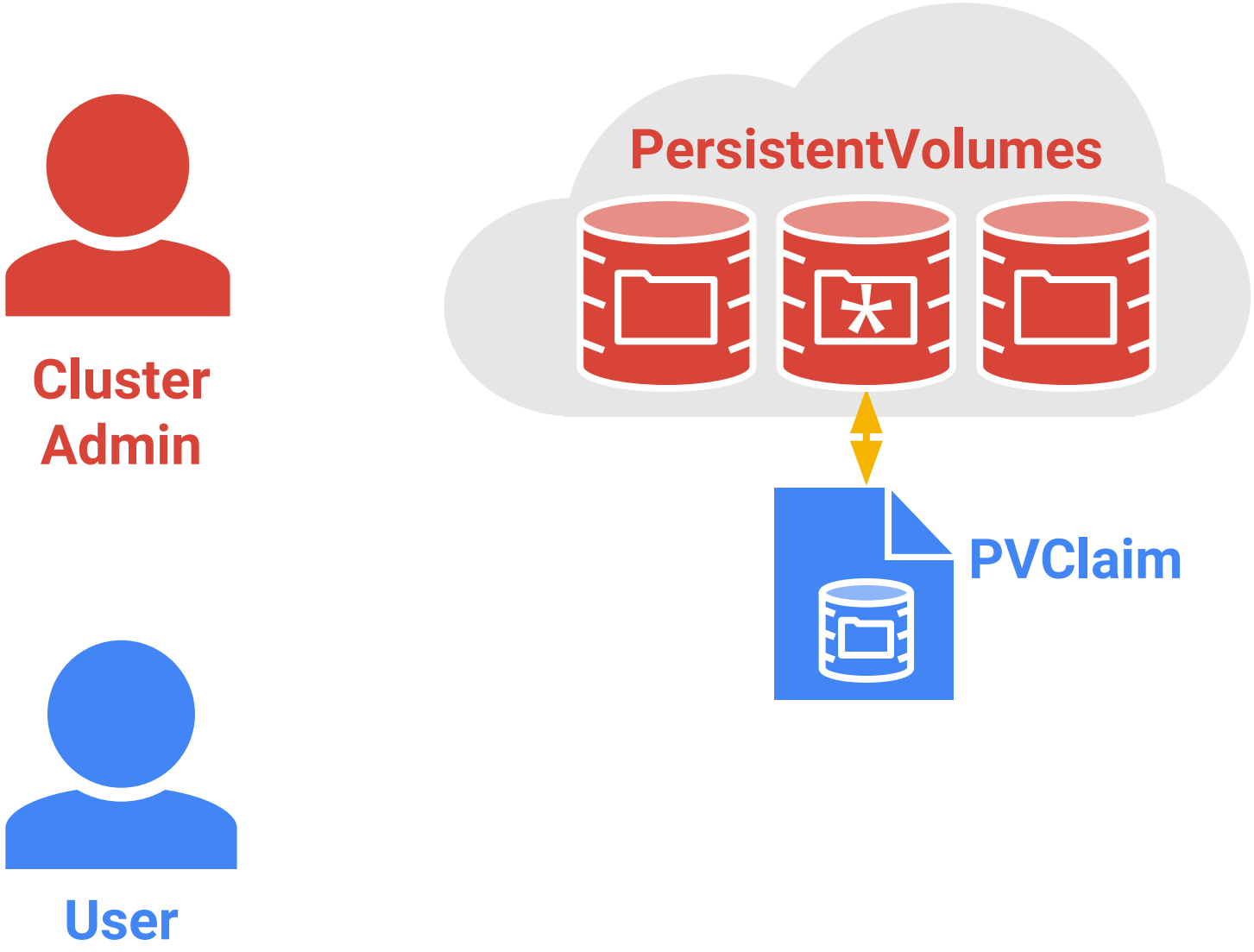
PersistentVolumes



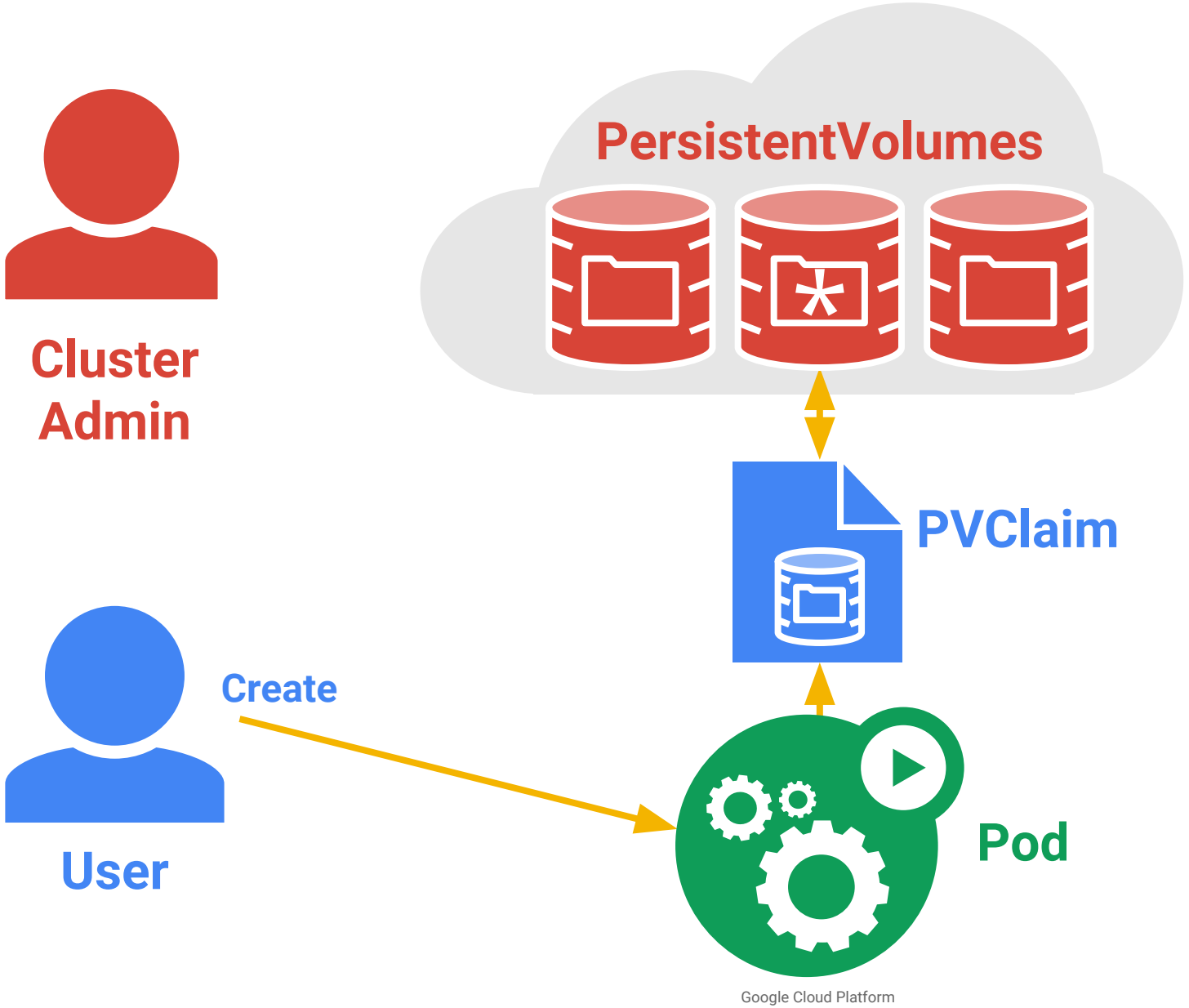
PersistentVolumes



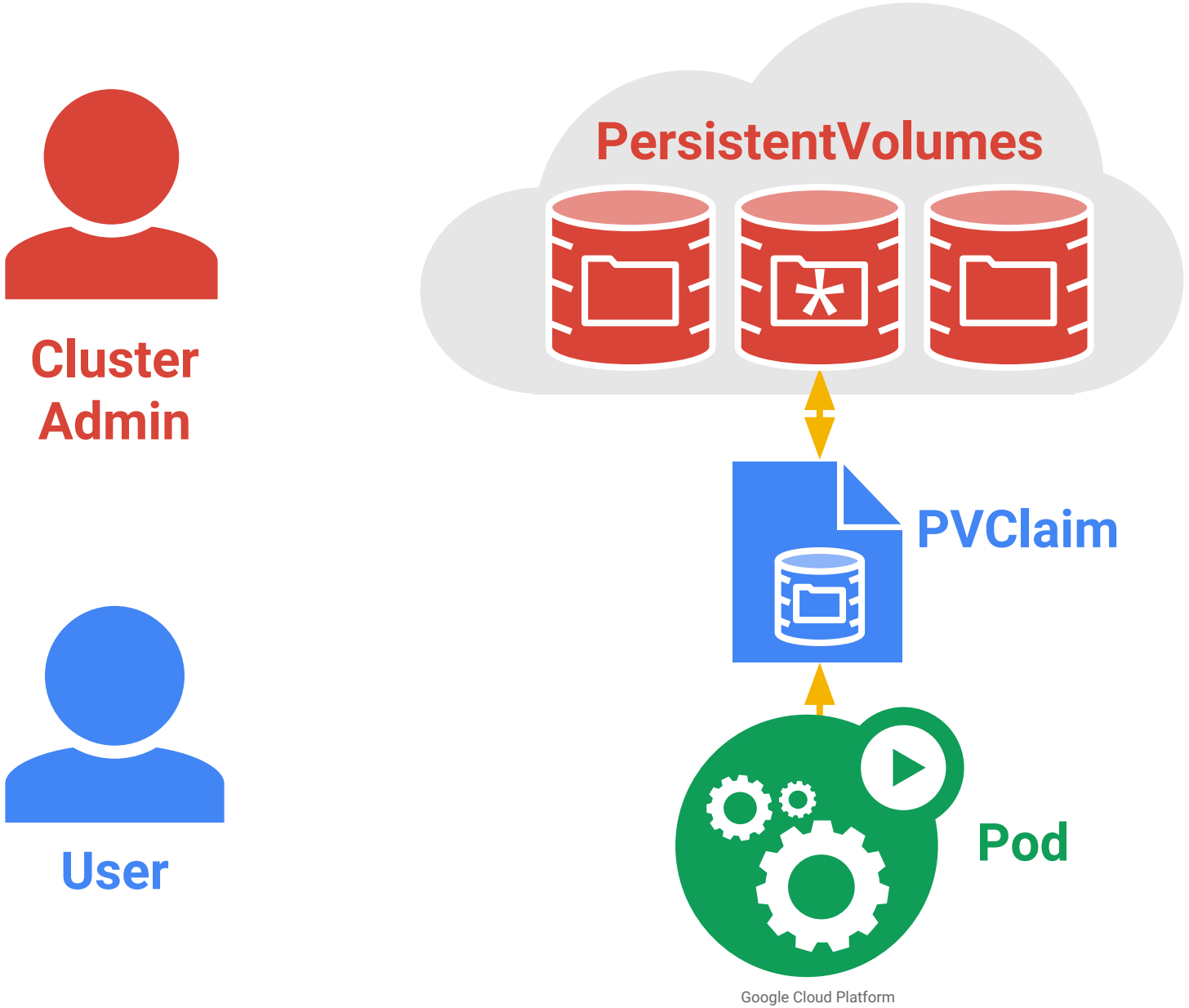
PersistentVolumes



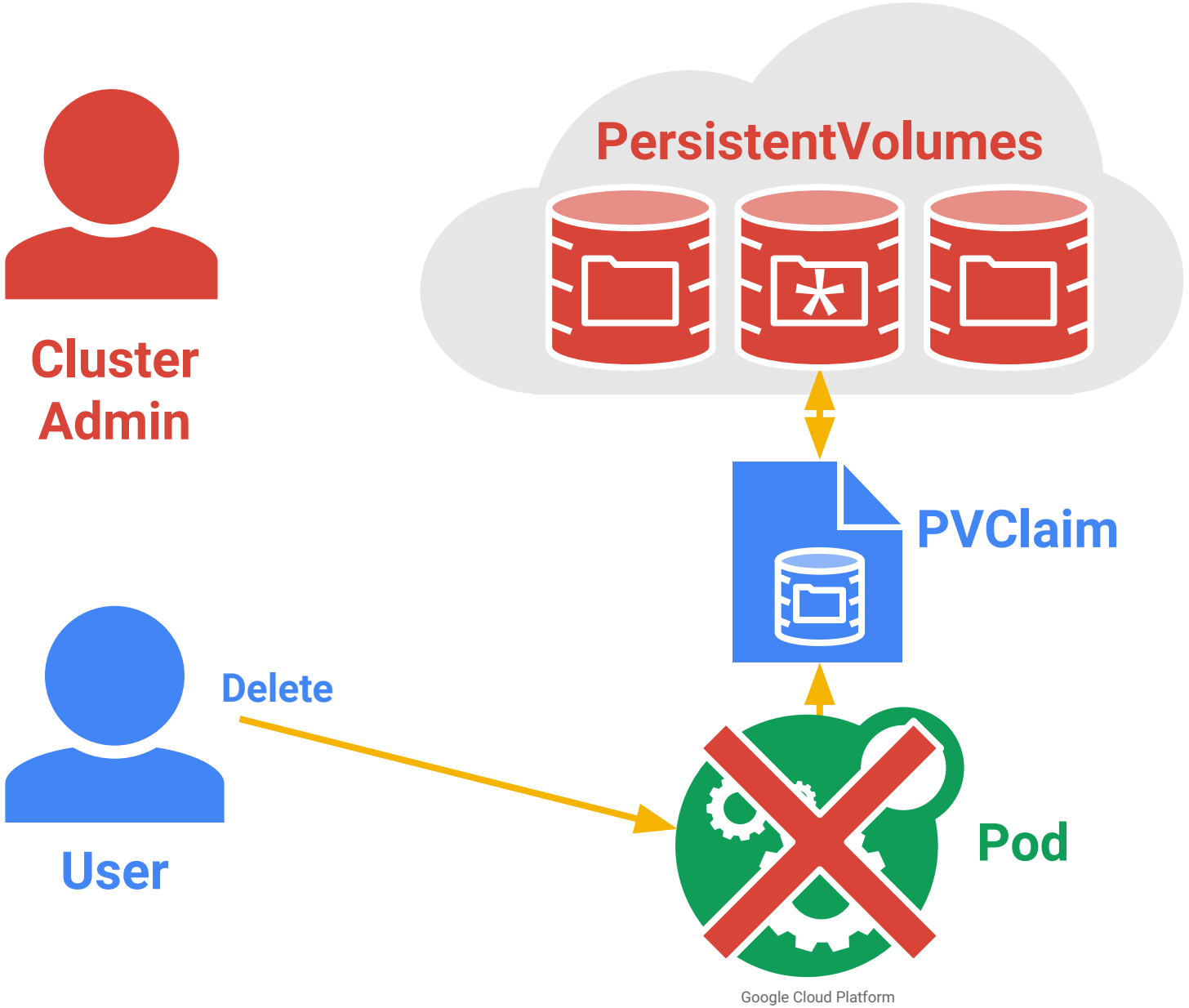
PersistentVolumes



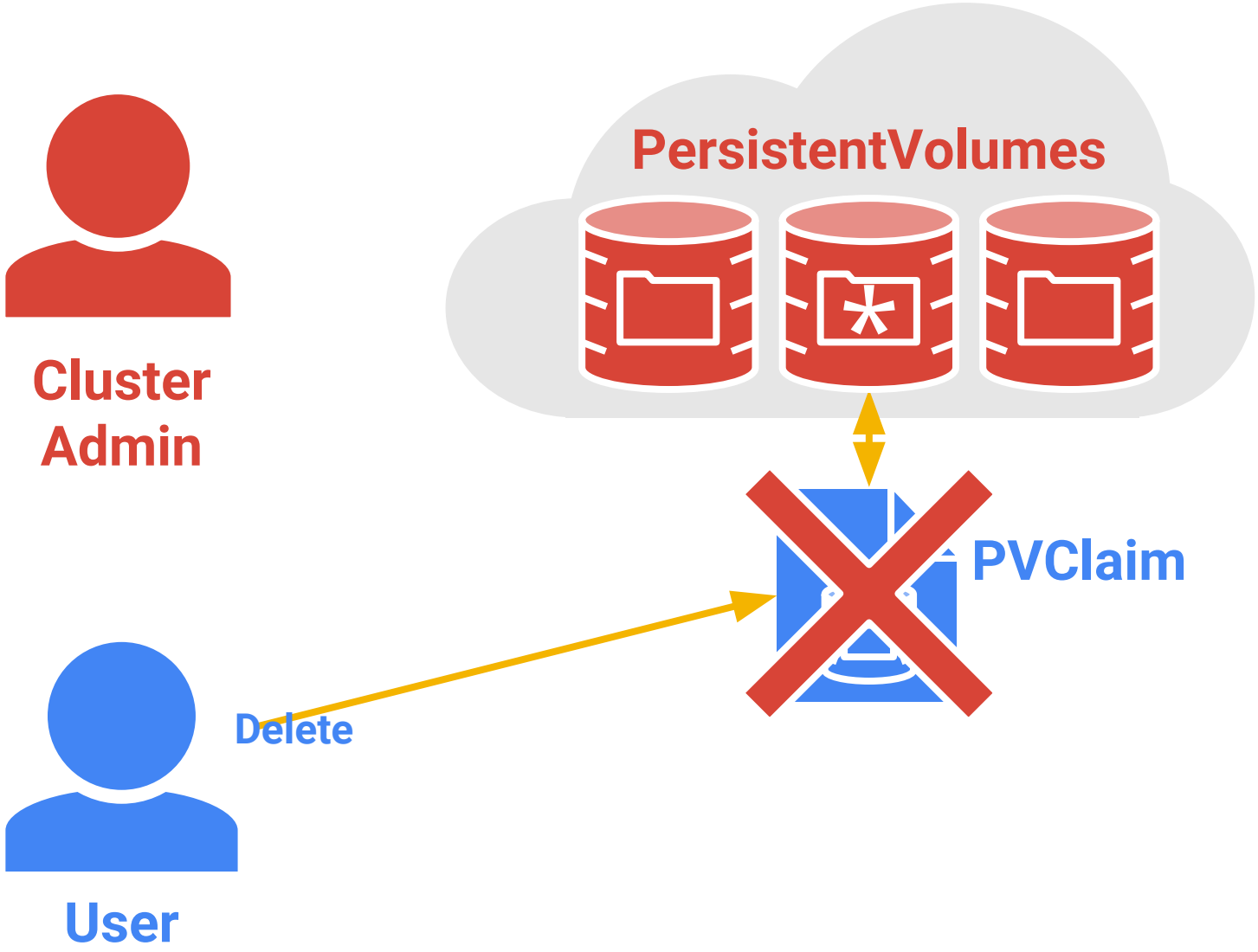
PersistentVolumes



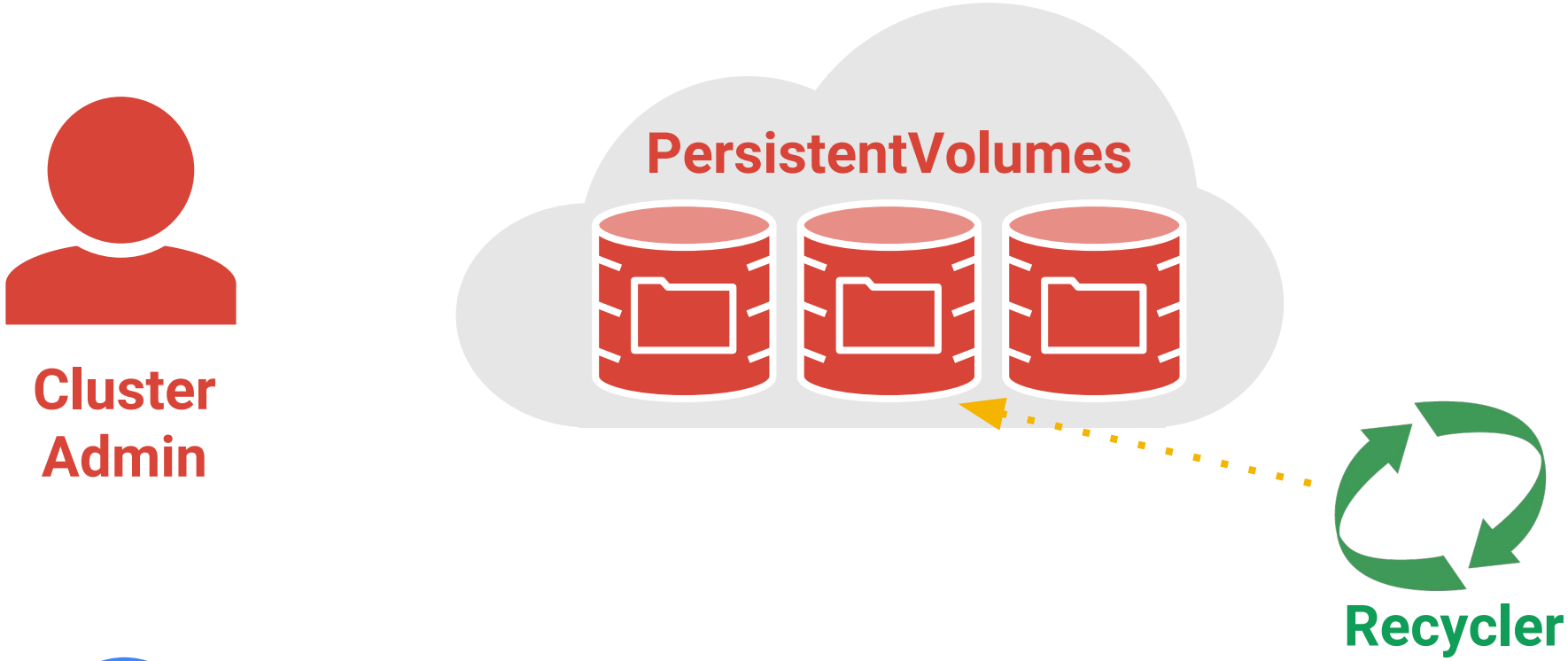
PersistentVolumes



PersistentVolumes



PersistentVolumes



Namespaces

Namespaces

Problem: I have too much stuff!

- name collisions in the API
- poor isolation between users
- don’t want to expose things like Secrets

Solution: Slice up the cluster

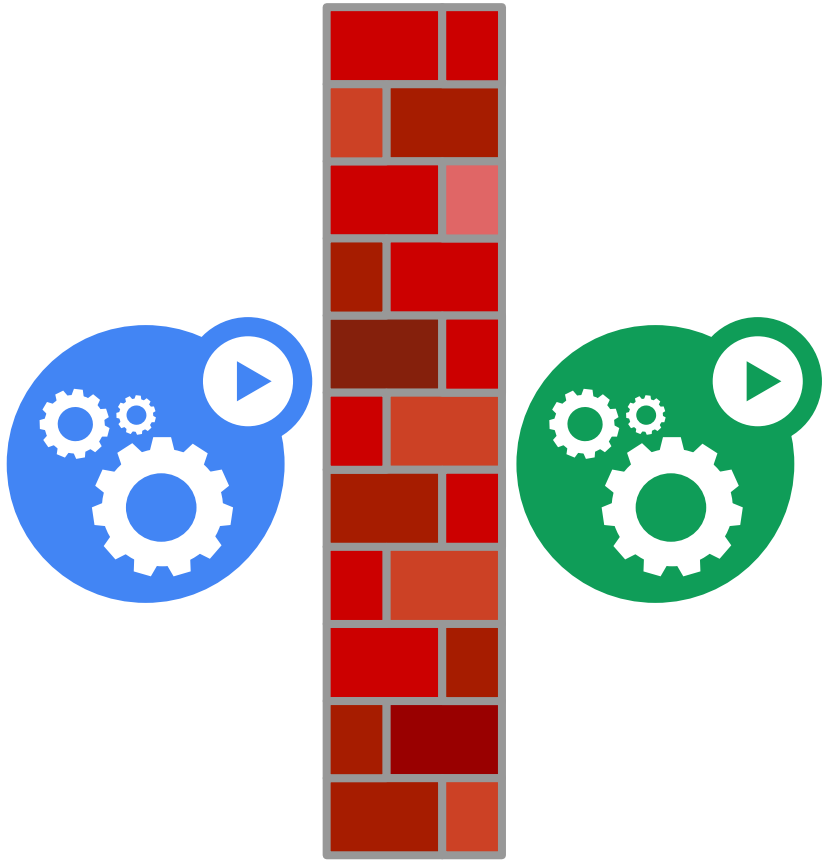
- create new Namespaces as needed
  - per-user, per-app, per-department, etc.
- part of the API - NOT private machines
- most API objects are namespaced
  - part of the REST URL path
- Namespaces are just another API object
- One-step cleanup - delete the Namespace
- Obvious hook for policy enforcement (e.g. quota)

Resource Isolation

Resource Isolation

Principles:

- Apps must not be able to affect each other's performance
  - if so it is an **isolation failure**
- Repeated runs of the same app should see ~equal behavior
- QoS levels drives resource decisions in (soft) real-time
- Correct in all cases, optimal in some
  - reduce unreliable components
- SLOs are the lingua franca



Strong isolation

Pros:

- Sharing - users don't worry about interference (aka the noisy neighbor problem)
- Predictable - allows us to offer strong SLAs to apps

Cons:

- Stranding - arbitrary slices mean some resources get lost
- Confusing - how do I know how much I need?
  - analog: what size VM should I use?
  - smart auto-scaling is needed!
- Expensive - you pay for certainty

In reality this is a multi-dimensional bin-packing problem: CPU, memory, disk space, IO bandwidth, network bandwidth, ...

Requests and Limits

Request:

- how much of a resource you are asking to use, with a strong guarantee of availability
  - CPU (seconds/second)
  - RAM (bytes)
- scheduler will not over-commit requests

Limit:

- max amount of a resource you can access

Repercussions:

- Usage > Request: resources **might** be available
- Usage > Limit: throttled or killed

Quality of Service

Defined in terms of Request and Limit

Guaranteed: highest protection

- `request > 0 && limit == request`

Burstable: medium protection

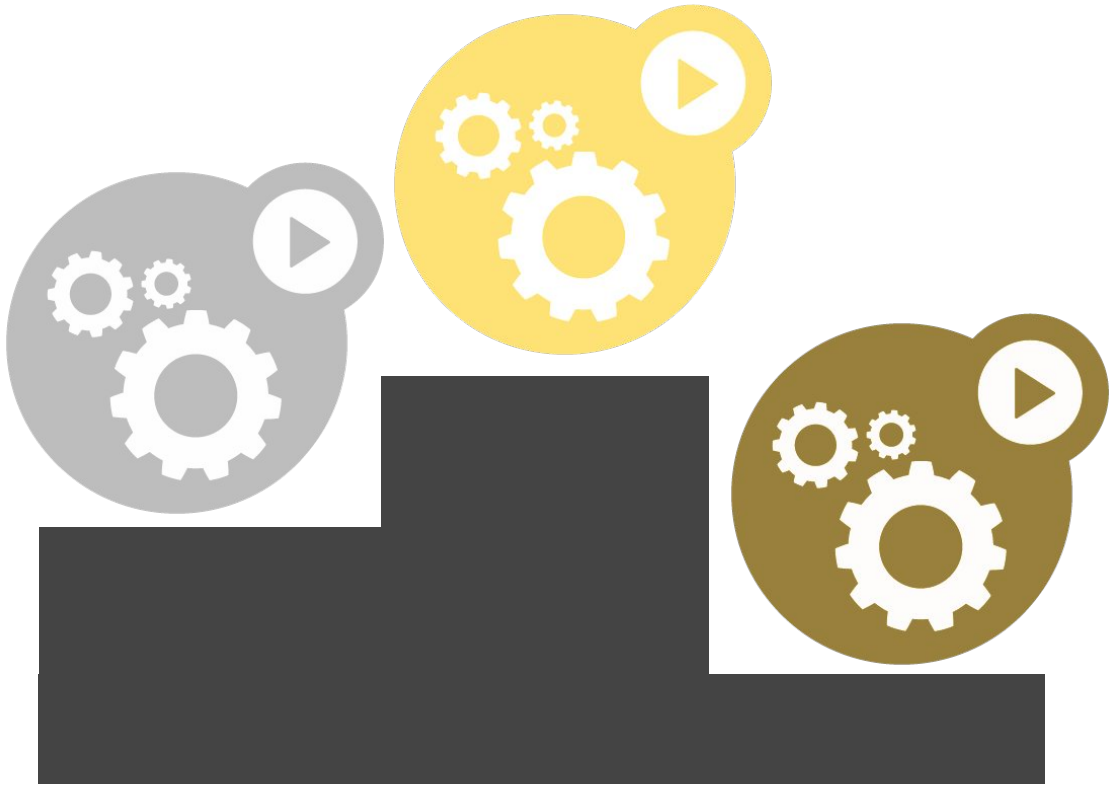
- `request > 0 && limit > request`

Best Effort: lowest protection

- `request == 0`

What does “protection” mean?

- OOM score
- CPU scheduling



Quota and Limits

ResourceQuota

Admission control: apply limits in **aggregate**

Per-namespace: ensure no user/app/department abuses the cluster

Reminiscent of disk quota by design

Applies to each type of resource

- CPU and memory for now

Disallows pods without resources

LimitRange

Admission control: limit the limits

- min and max
- ratio of limit/request

Default values for unspecified limits

Per-namespace

Together with ResourceQuota gives cluster admins powerful tools

Network Plugins

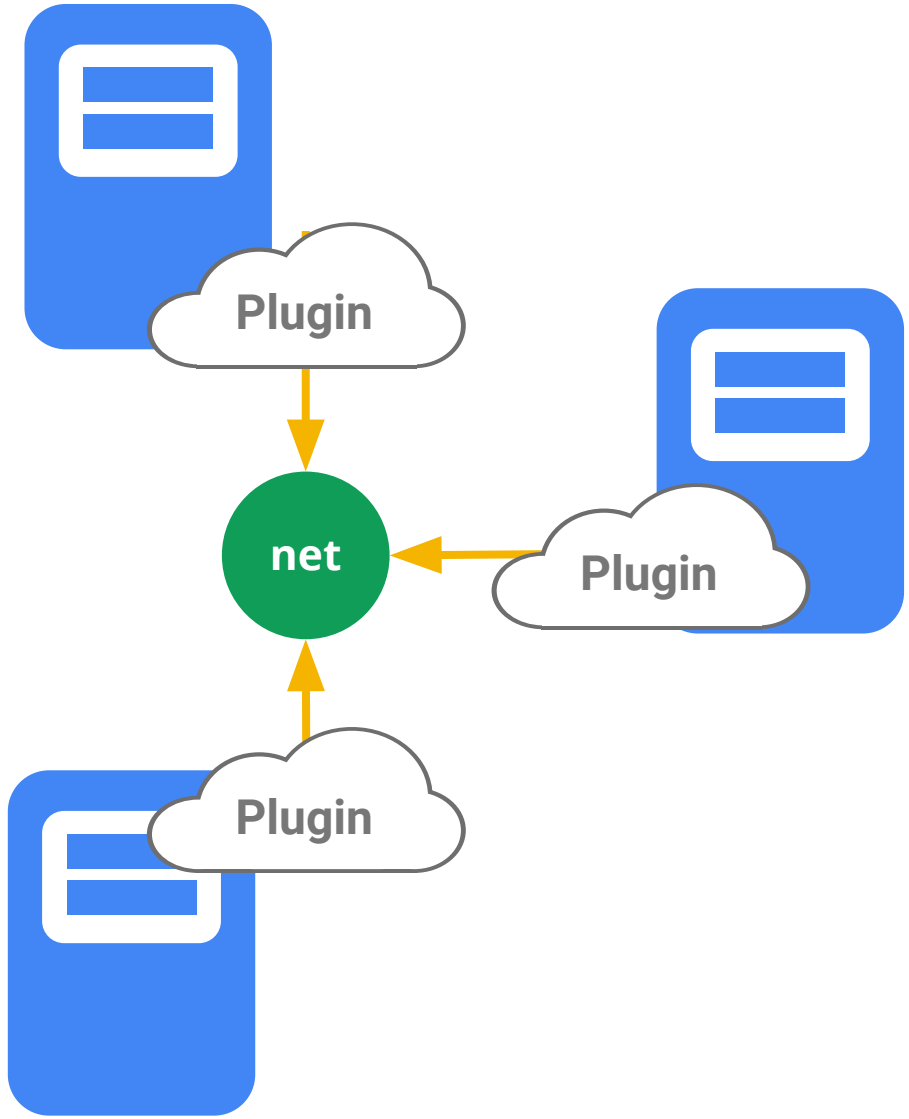
Network Plugins

Default Network plugin included in Kubernetes v1.9

Uses **CNI**:

- Simple exec interface
- Not using Docker libnetwork

Cluster admins can customize their installs



Network Isolation

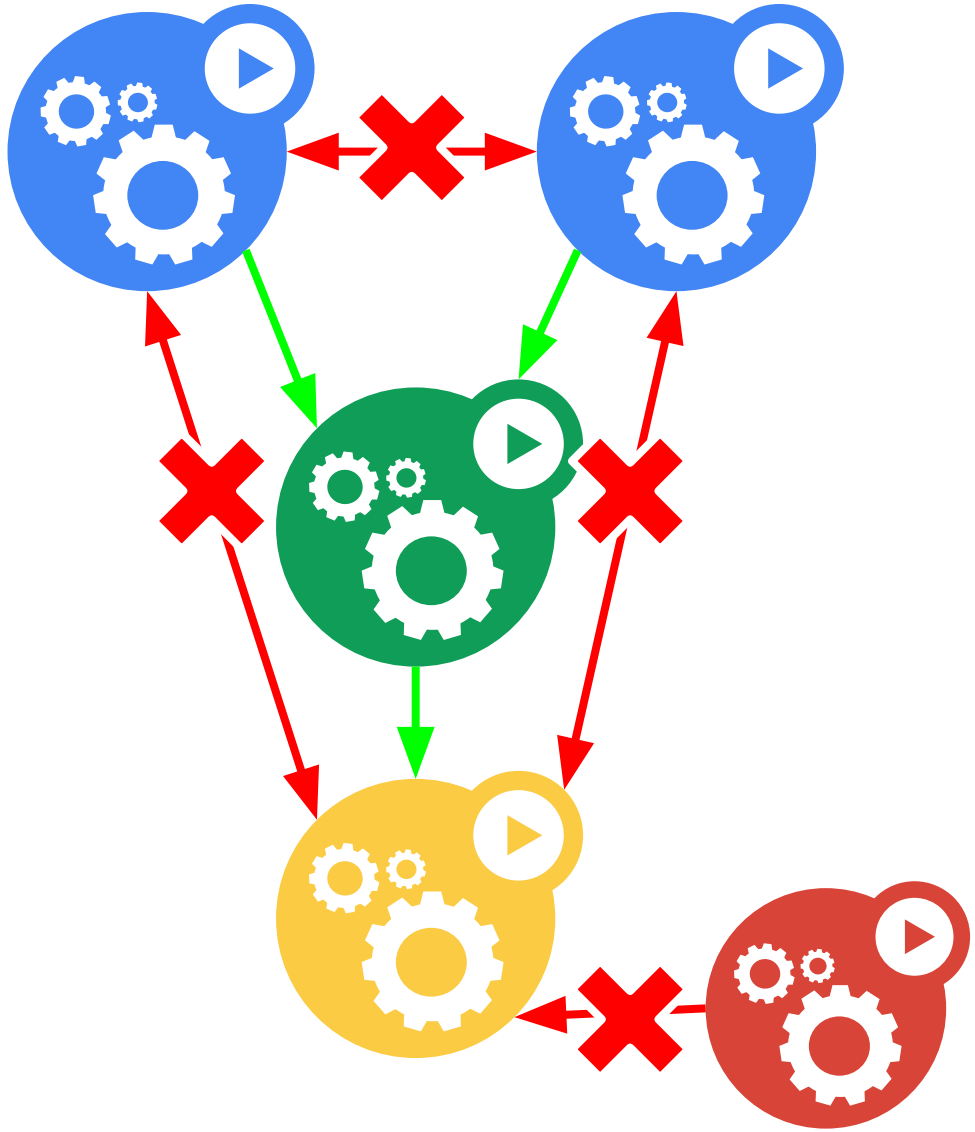
Network Isolation

Describe the DAG of your app, enforce it in the network

Restrict Pod-to-Pod traffic or across Namespaces

Designed by the network SIG

- implementations for Calico, OpenShift, Romana, OpenContrail (so far)



Node Drain

Node Drain

Goal: Evacuate a node for maintenance

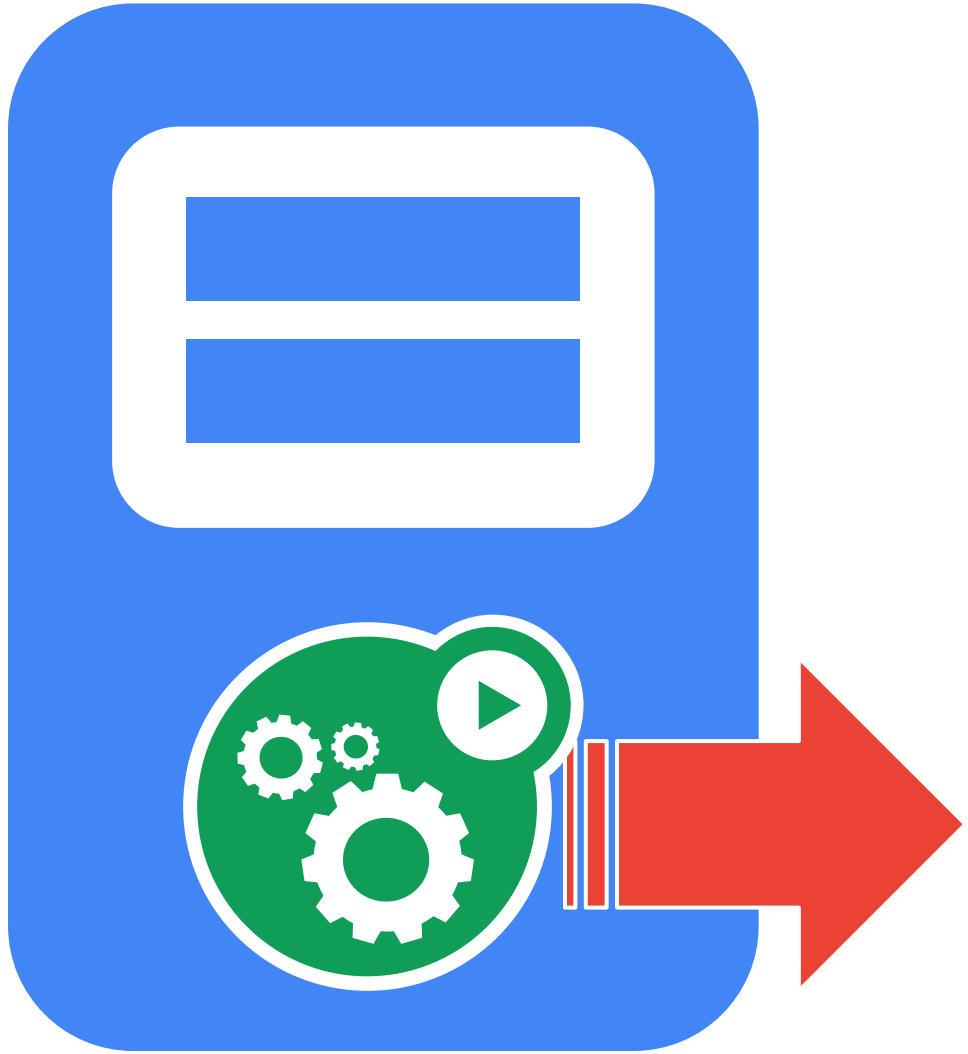
- e.g. kernel upgrades

CLI: `kubectl drain`

- disallow scheduling
- allow grace period for pods to terminate
- kill pods

When done: `kubectl uncordon`

- the node rejoins the cluster



Cluster Auto-Scaling

Cluster Autoscaler

Add nodes when needed

- there are pending pods
- some pending pods would fit if we add a node

Remove nodes when not needed

- after removal, all pods must fit remaining nodes

Status: Works on GCE, GKE and AWS

Primary concepts

Container: A sealed application package (Docker)

Pod: A small group of tightly coupled Containers

example: content syncer & web server

Controller: A loop that drives current state towards desired state

example: replication controller

Service: A set of running pods that work together

example: load-balanced backends

Labels: Identifying metadata attached to other objects

example: phase=canary vs. phase=prod

Selector: A query against labels, producing a set result

example: all pods where label phase == prod

Design principles

Declarative > imperative: State your desired results, let the system actuate

Control loops: Observe, rectify, repeat

Simple > Complex: Try to do as little as possible

Modularity: Components, interfaces, & plugins

Legacy compatible: Requiring apps to change is a non-starter

Network-centric: IP addresses are cheap

No grouping: Labels are the only groups

Cattle > Pets: Manage your workload in bulk

Open > Closed: Open Source, standards, REST, JSON, etc.

Control loops

Drive **current state -> desired state**

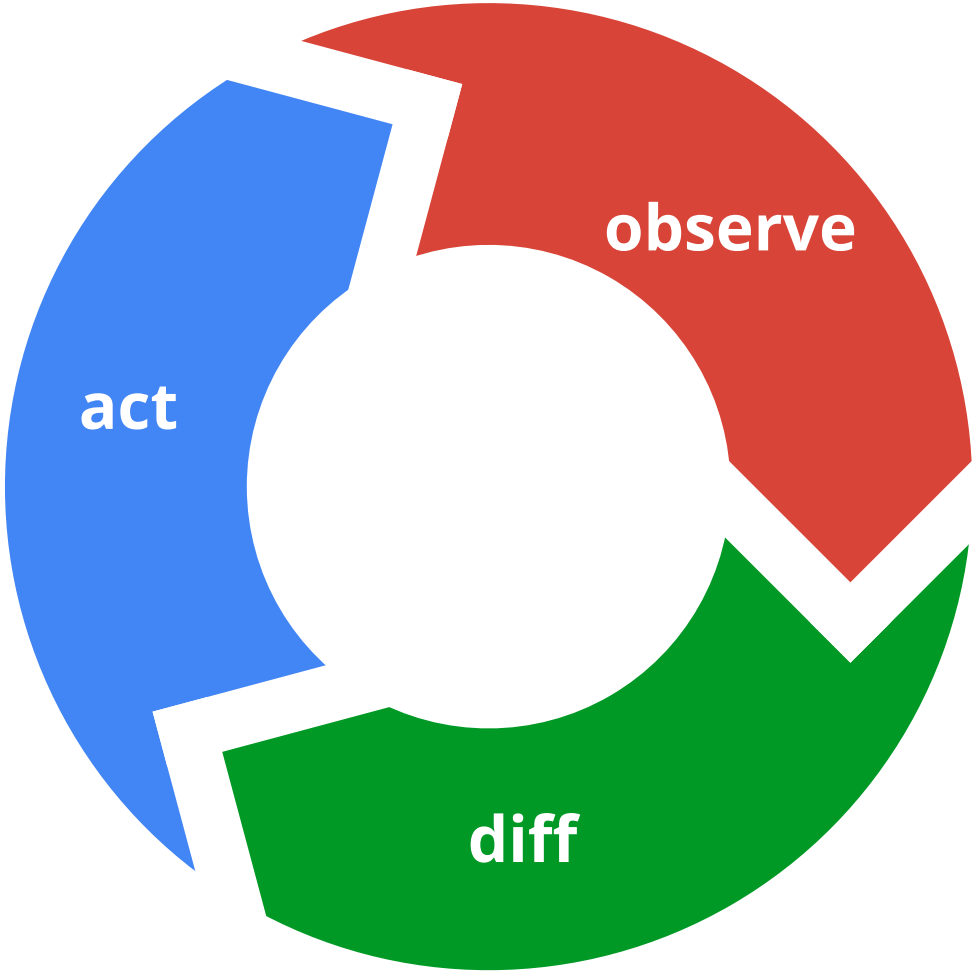
Act independently

APIs - **no shortcuts** or back doors

Observed state is truth

Recurring pattern in the system

Example: ReplicationController



Modularity

Loose coupling is a goal **everywhere**

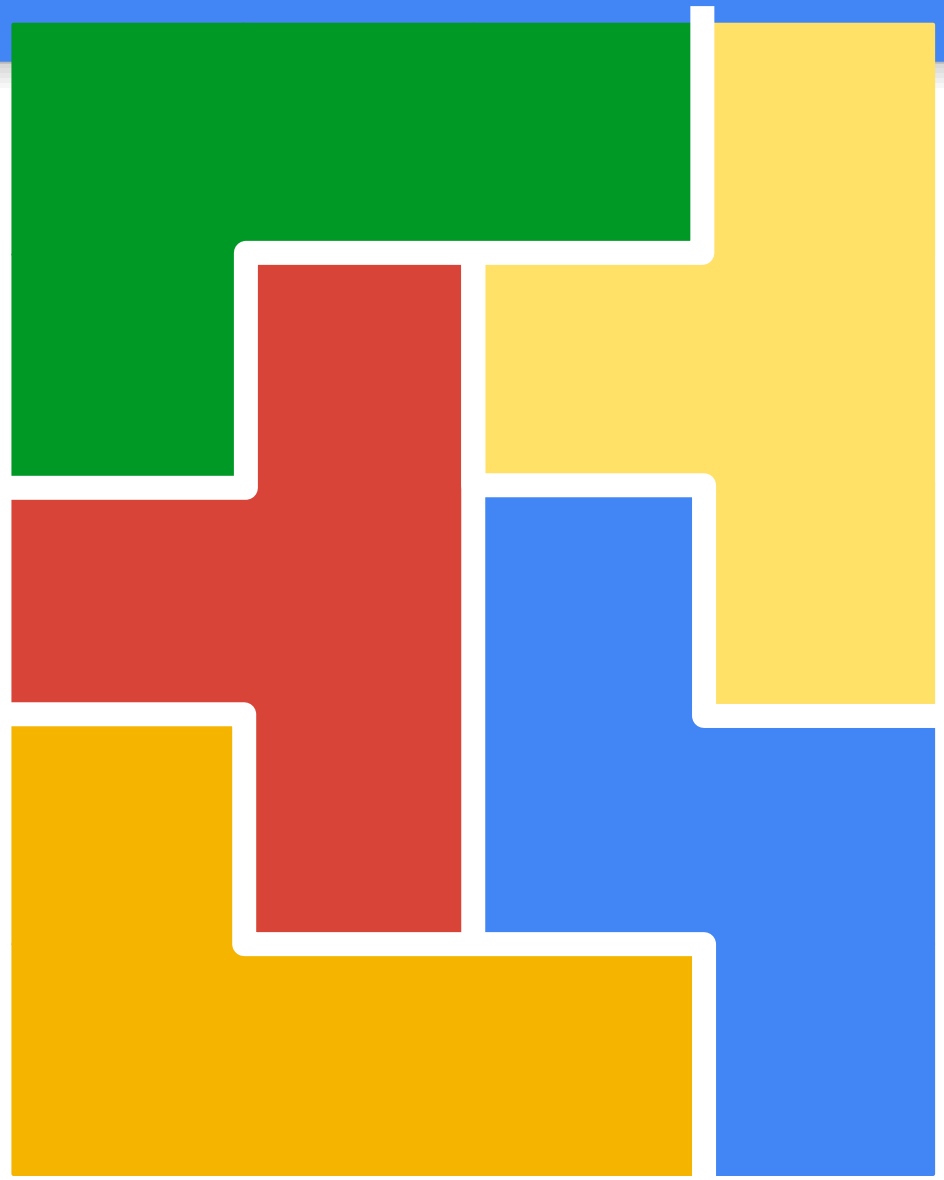
- simpler
- composable
- extensible

Code-level plugins where possible

Multi-process where possible

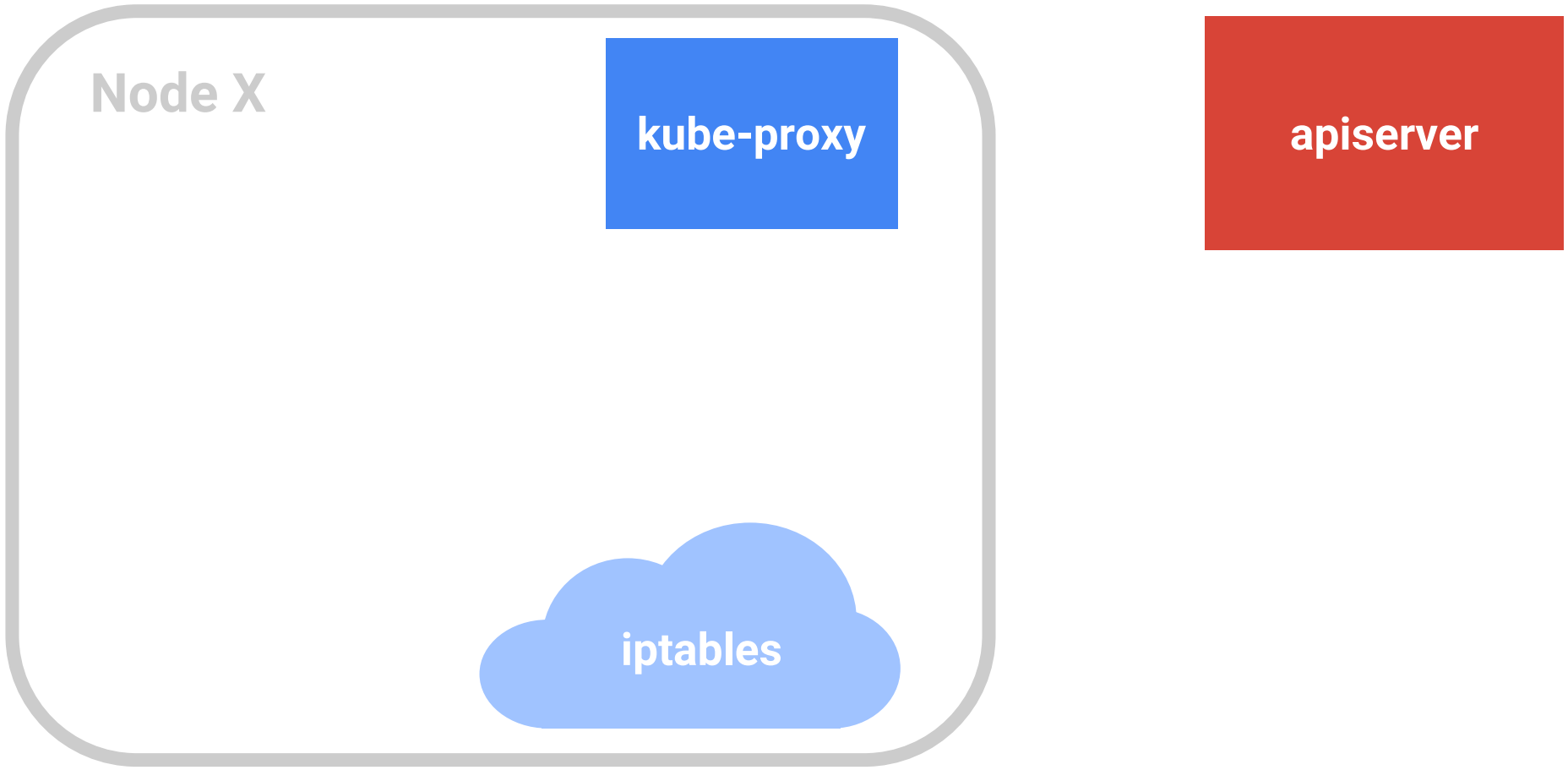
Isolate risk by interchangeable parts

Example: ReplicationController
Example: Scheduler



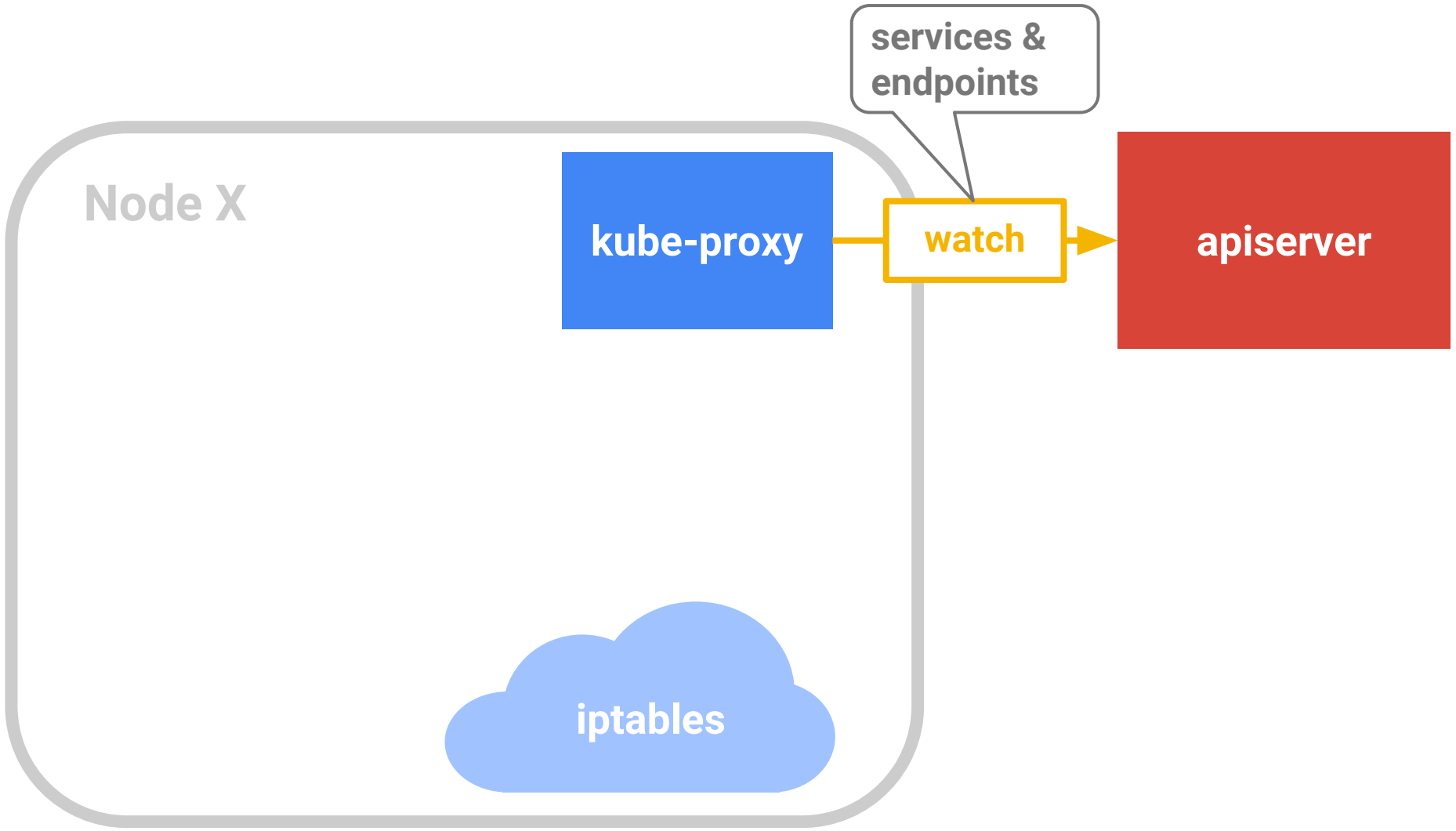
Kube Proxy iptables

iptables kube-proxy

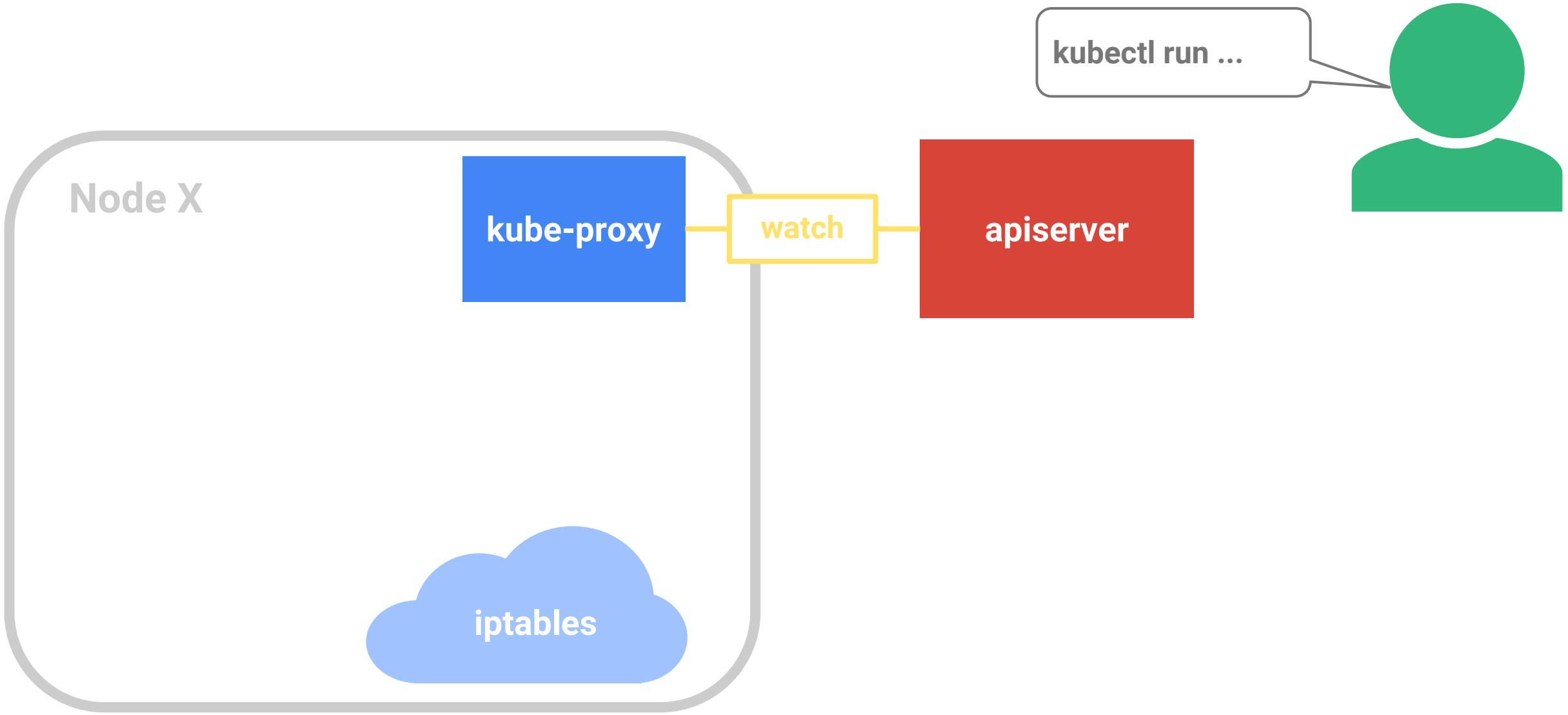


apiserver

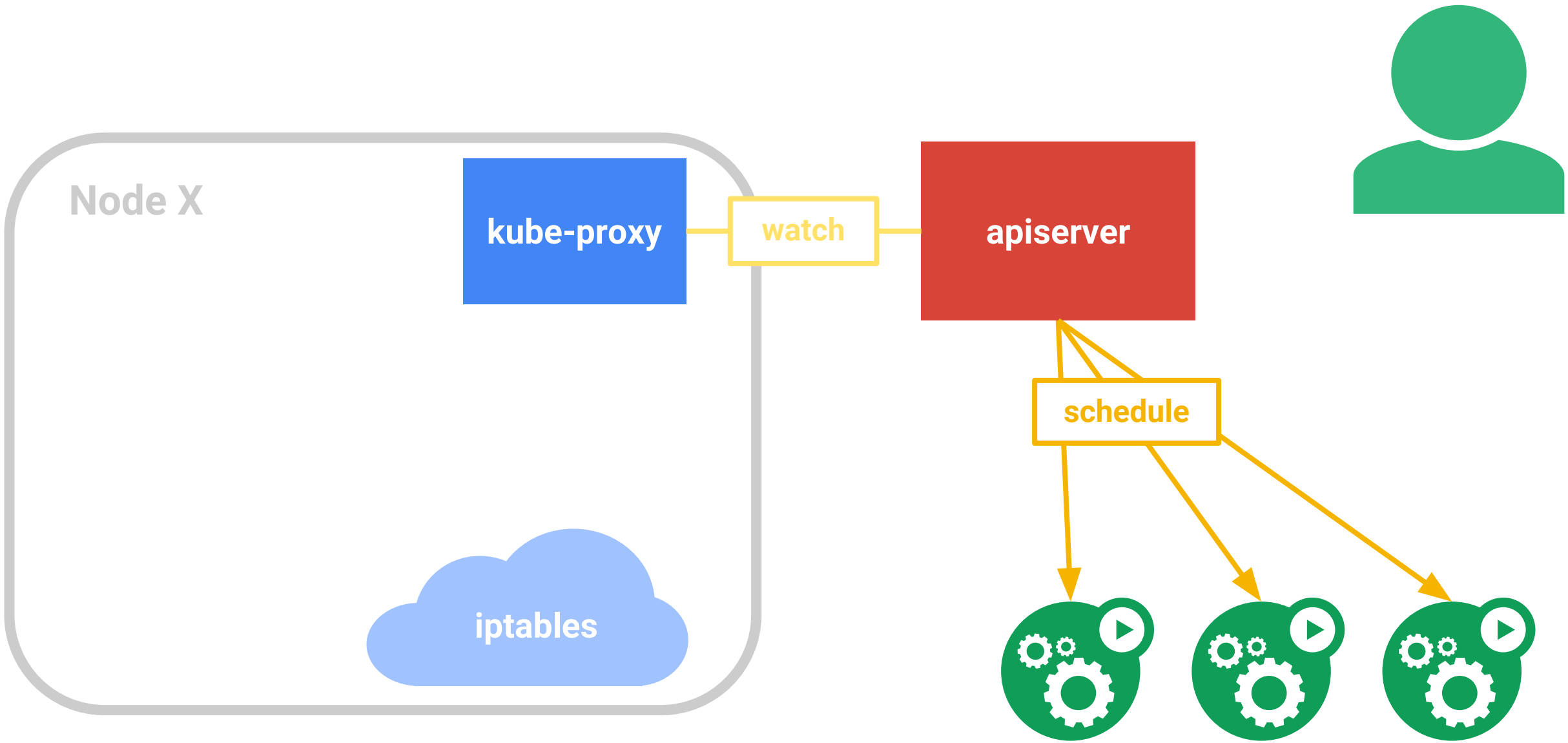
iptables kube-proxy



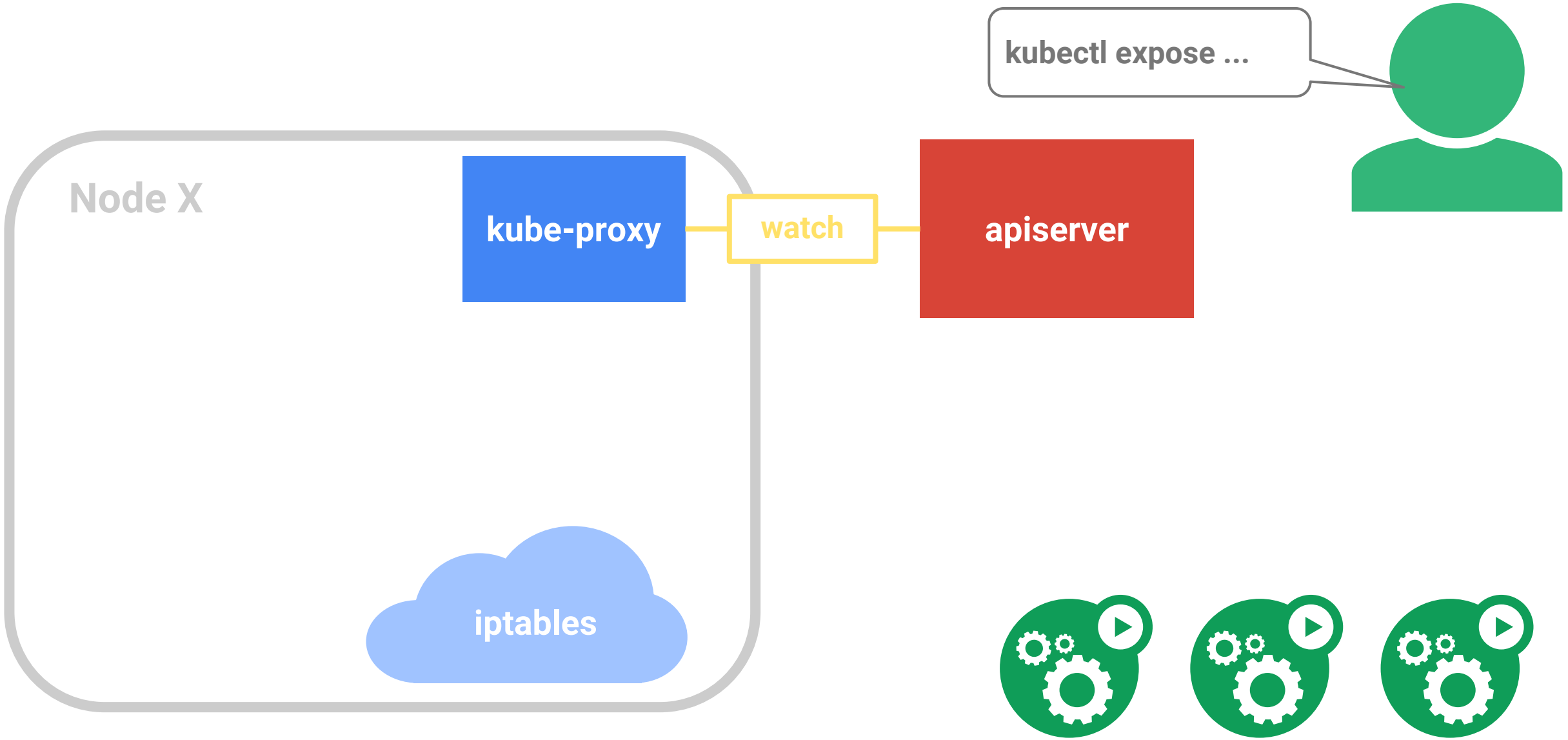
iptables kube-proxy



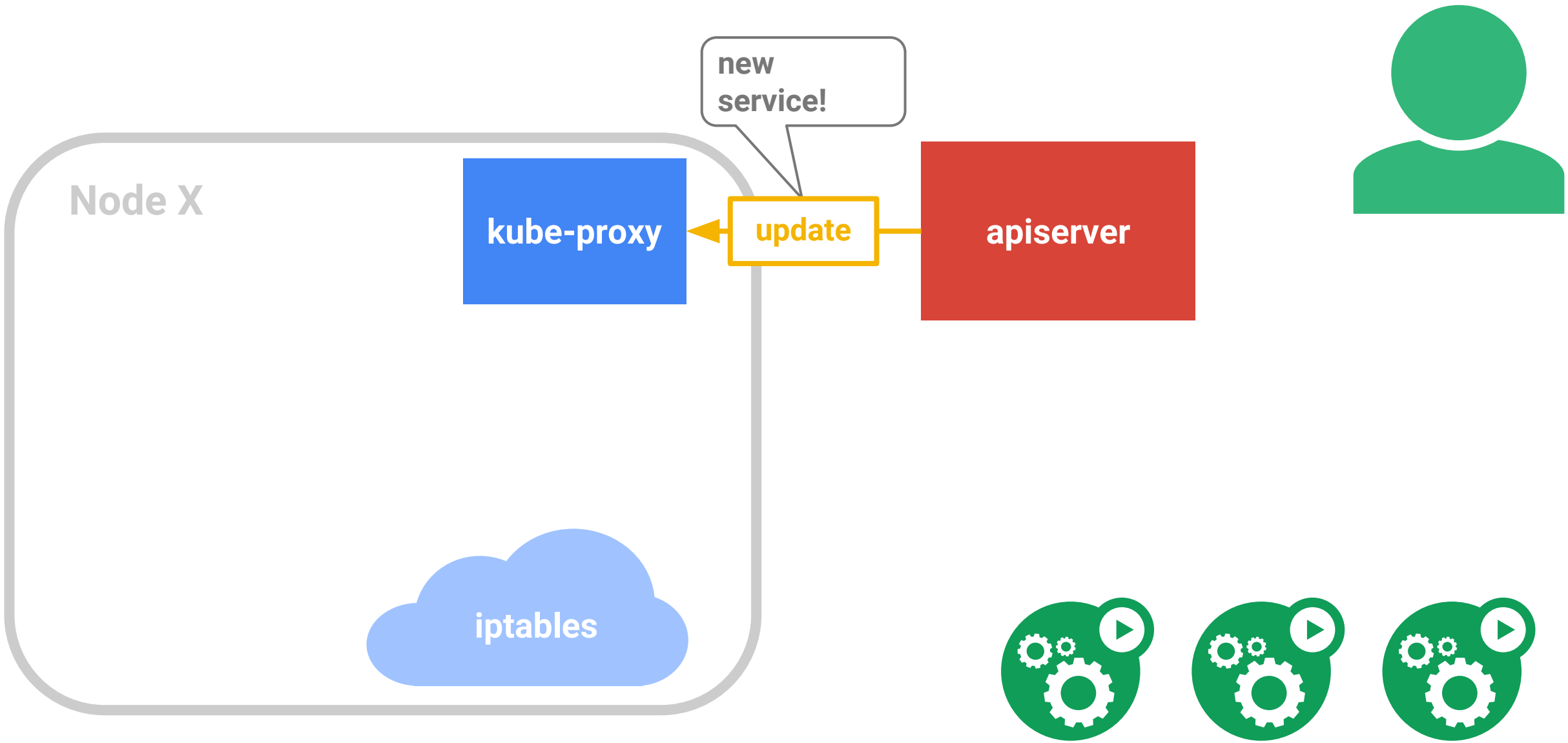
iptables kube-proxy



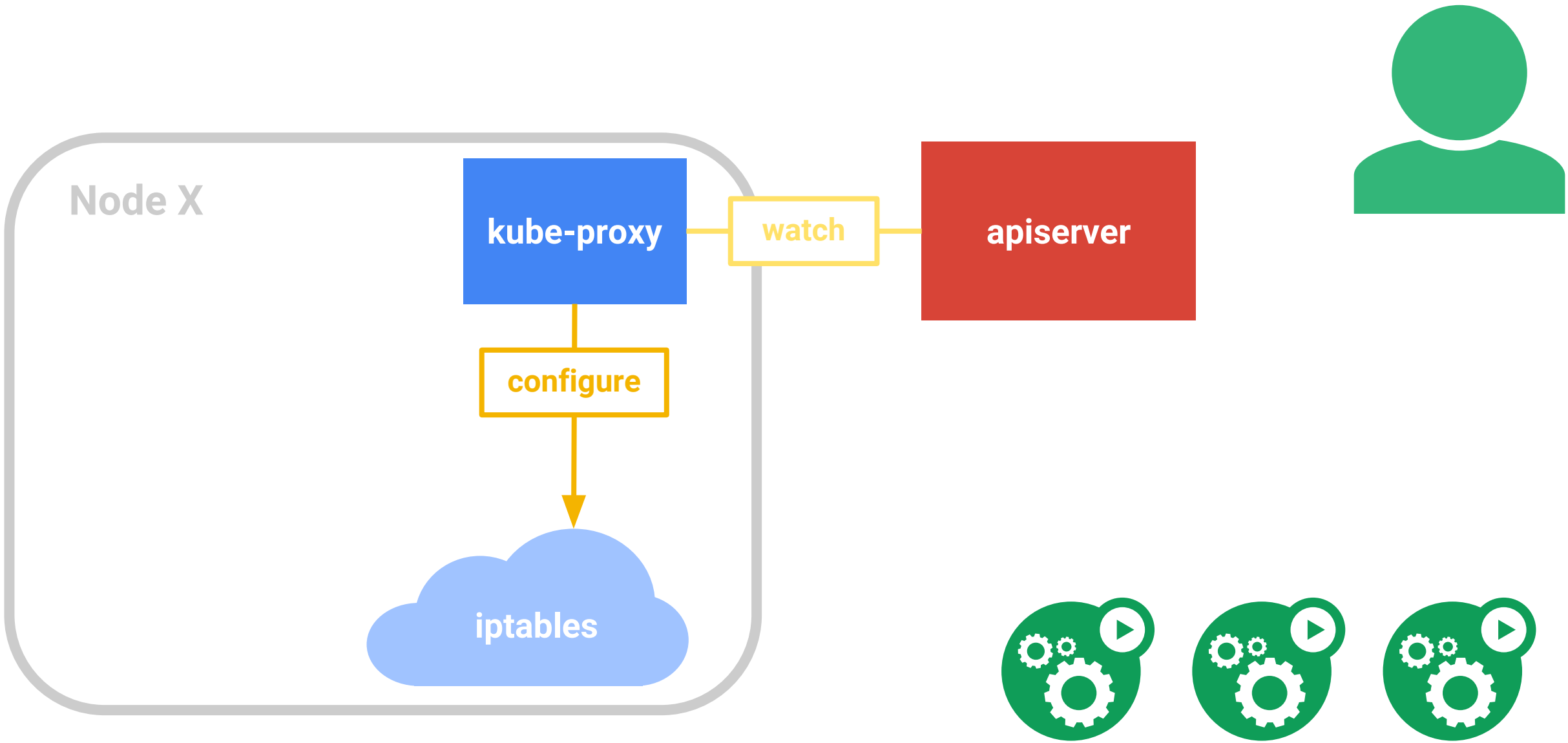
iptables kube-proxy



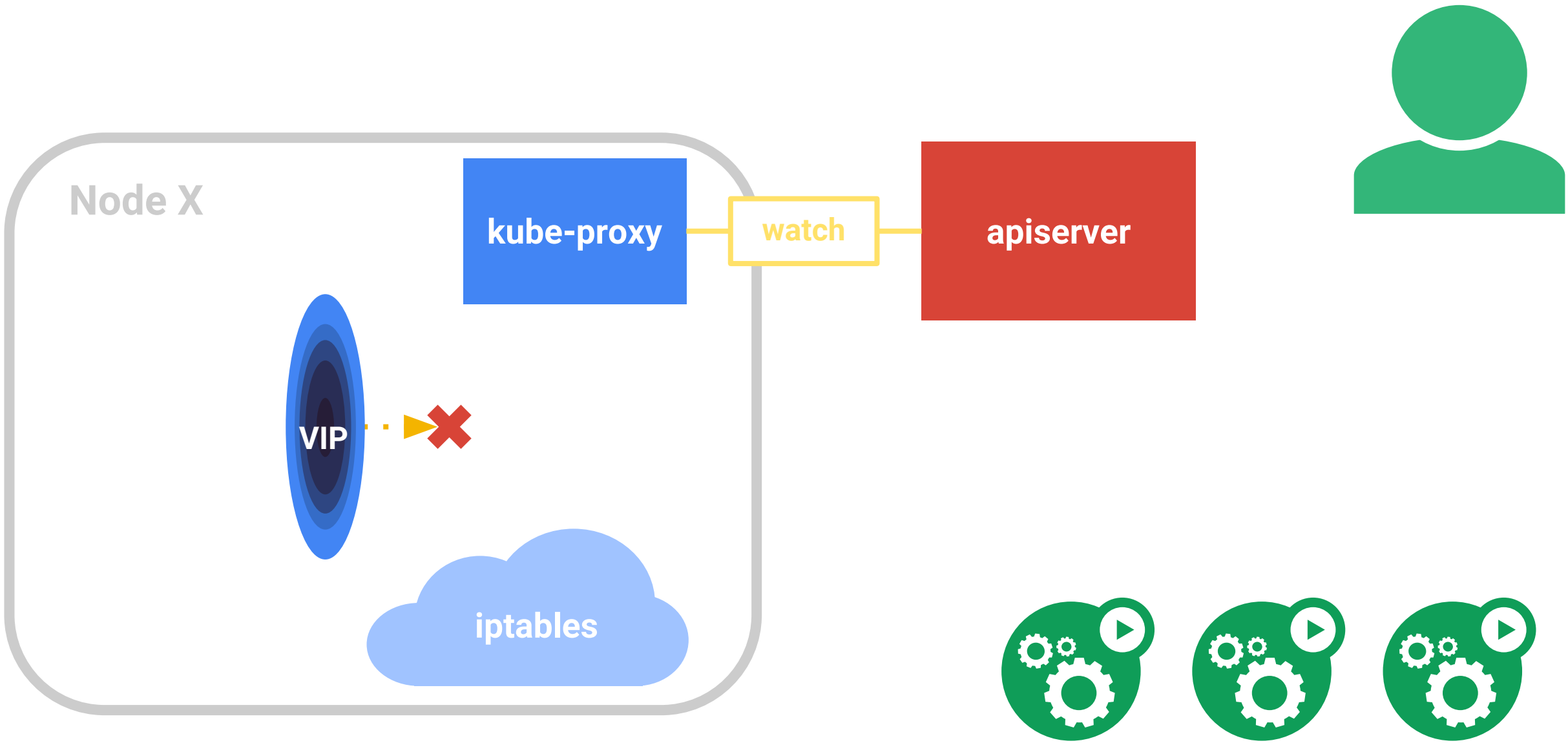
iptables kube-proxy



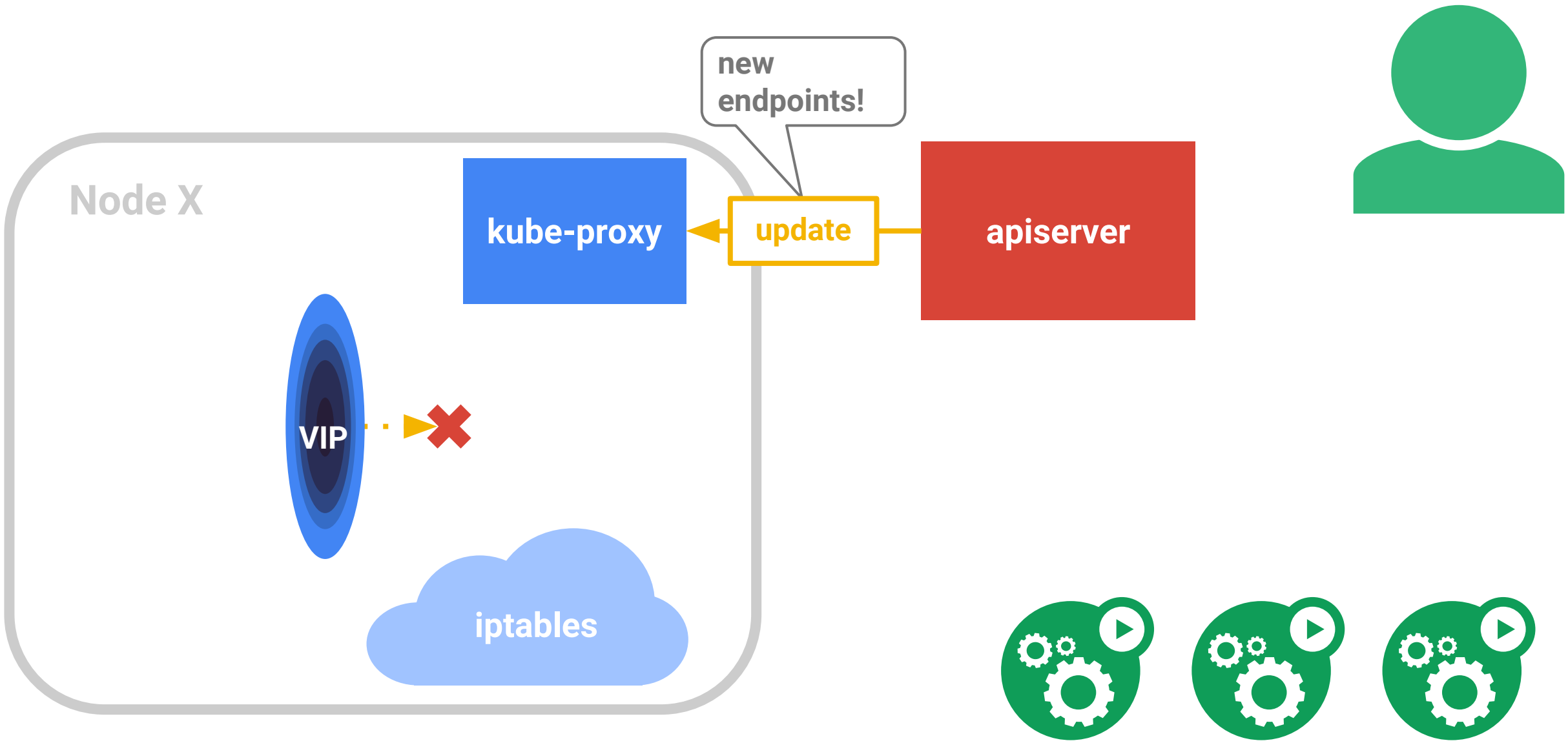
iptables kube-proxy



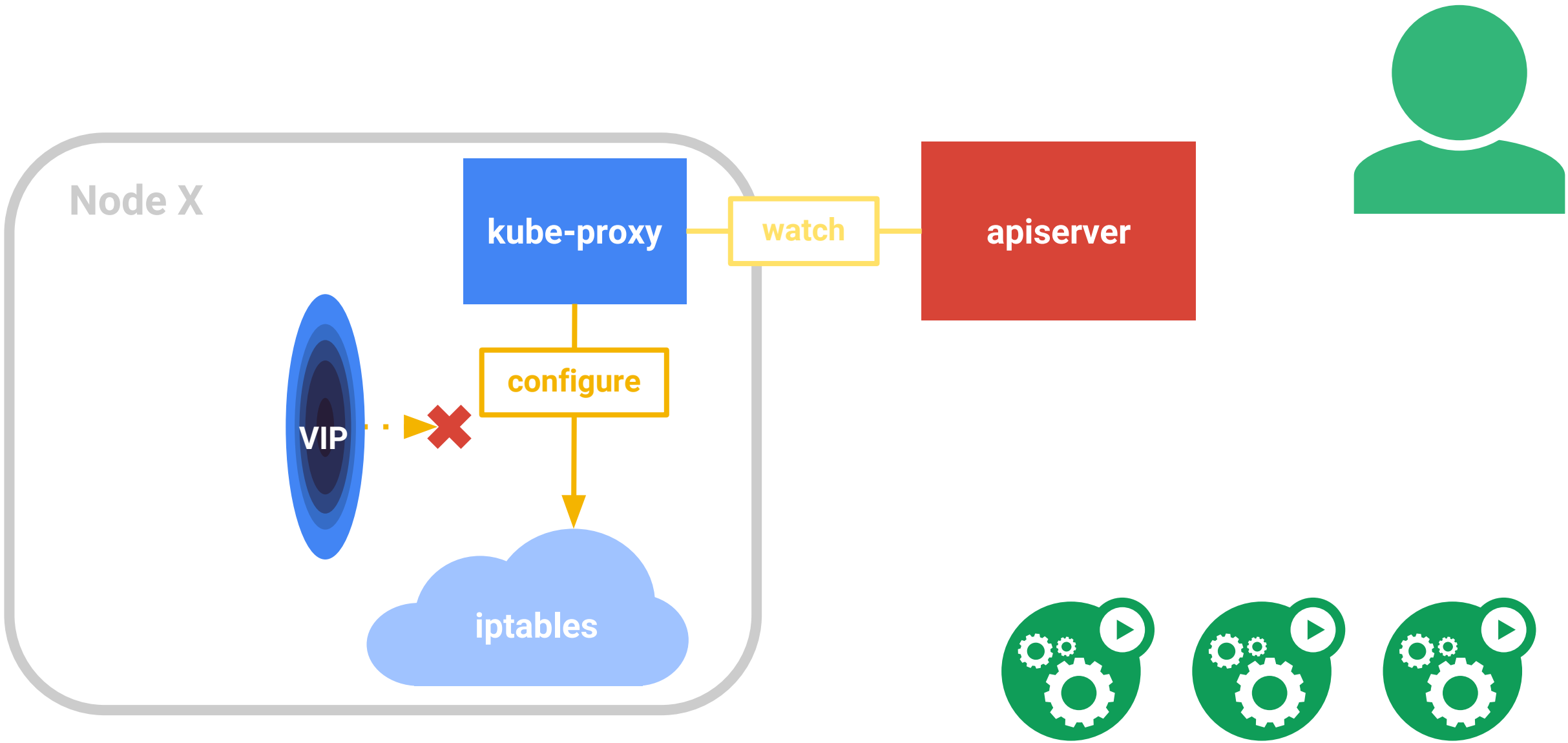
iptables kube-proxy



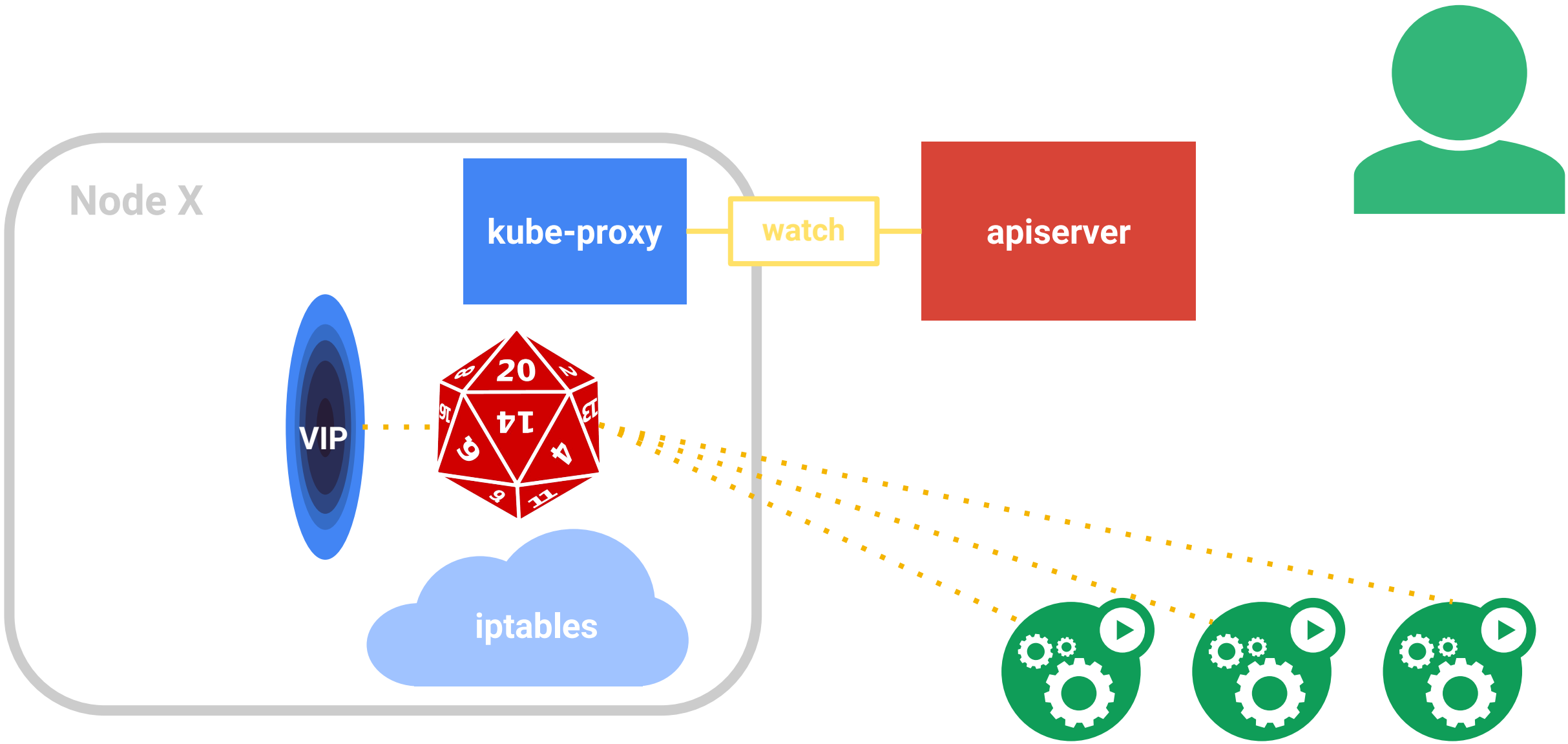
iptables kube-proxy



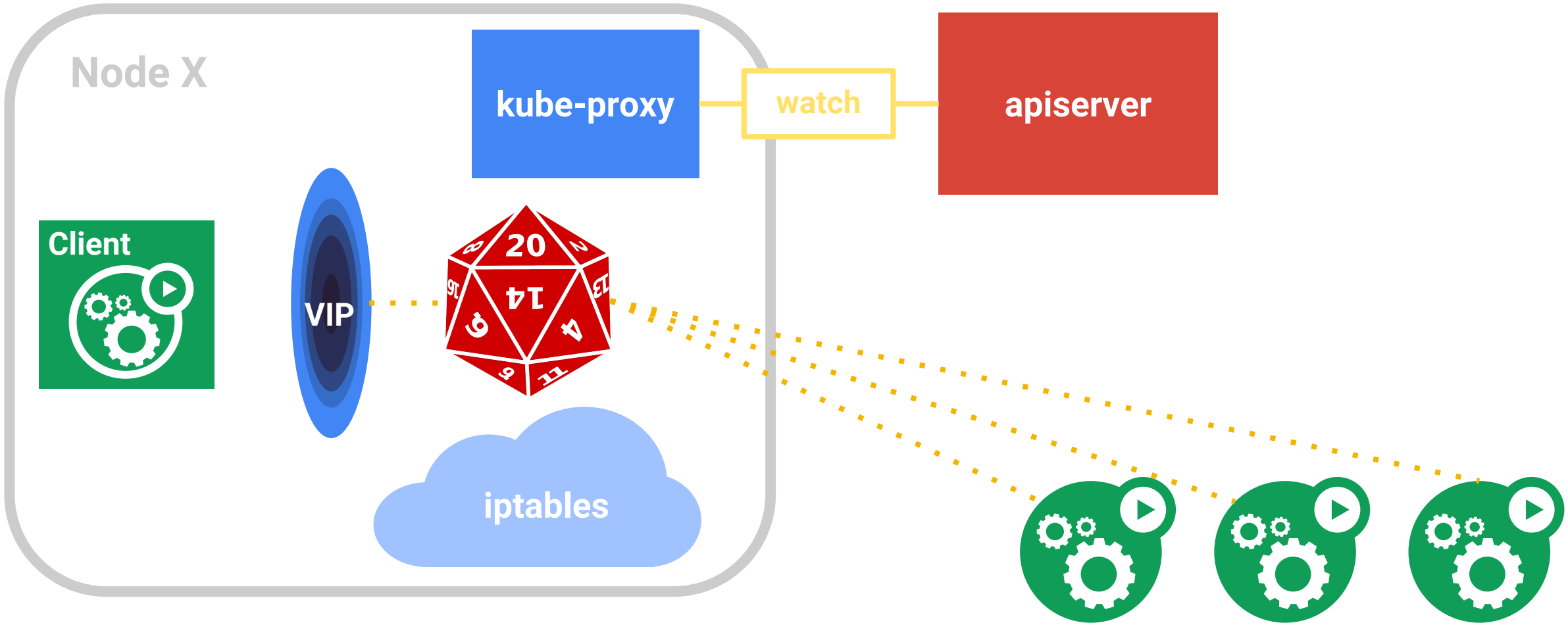
iptables kube-proxy



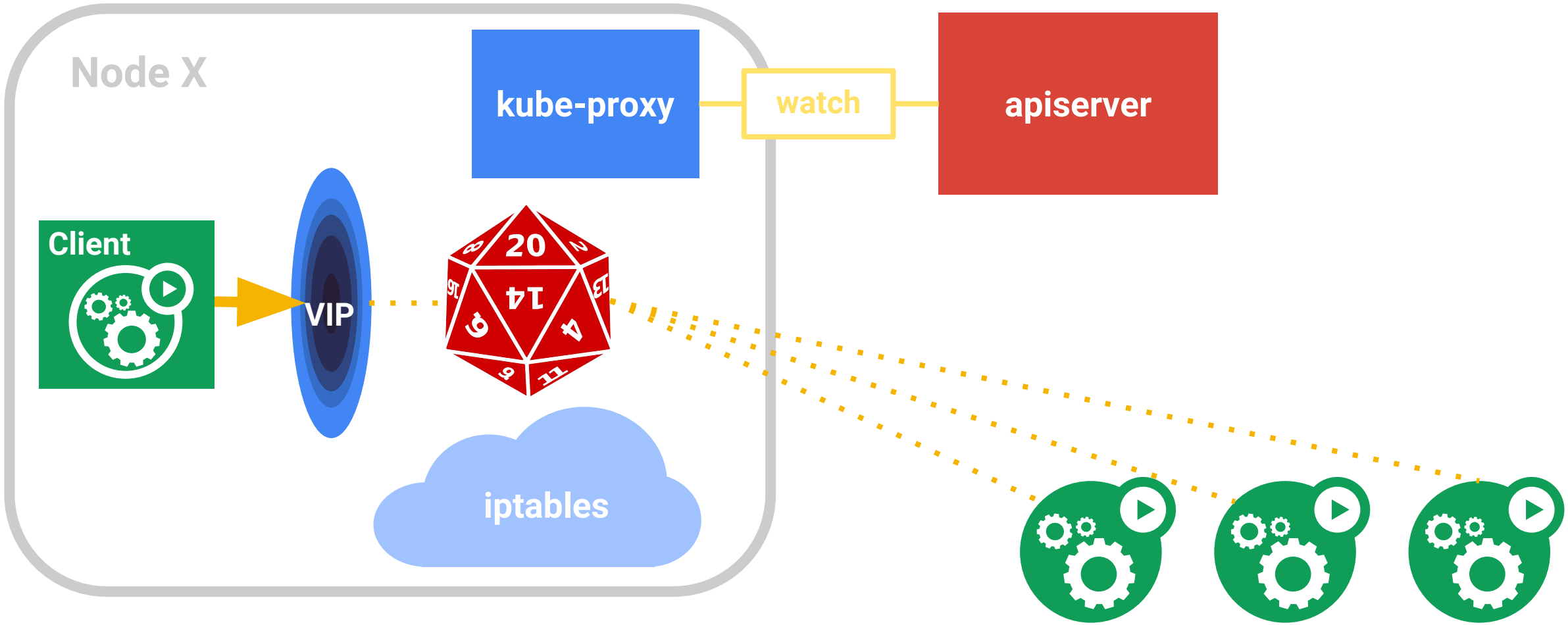
iptables kube-proxy



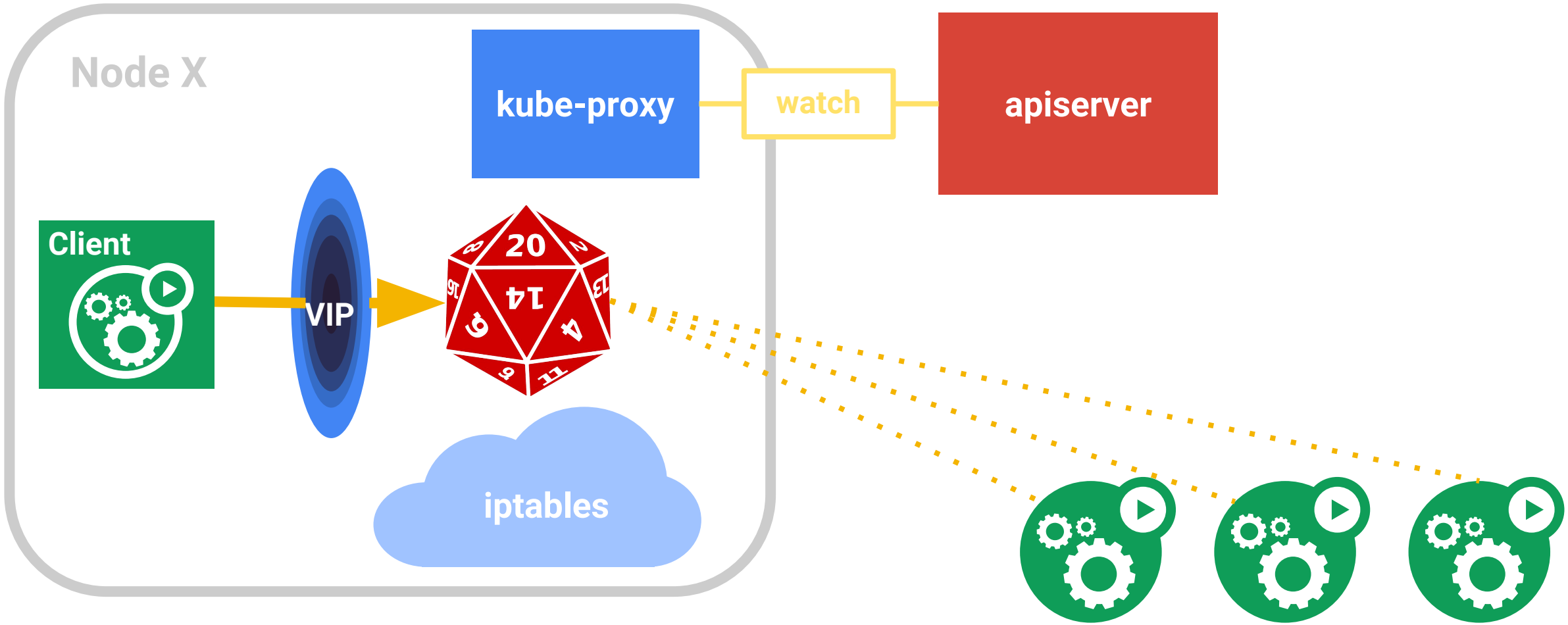
iptables kube-proxy



iptables kube-proxy



iptables kube-proxy



iptables kube-proxy