Kubernetes: Container Orchestration and Micro-Services

University of Washington 590s
2016-11-16

Alexander Mohr <mohr@google.com>
Technical Lead / Manager on Google Container Engine and Kubernetes
Github: @alex-mohr   Email: mohr@google.com

Contents

1. Systems Projects at Google Seattle and Kirkland (2-3 mins)
2. Brief Docker Container Primer (5-10 mins)
3. Kubernetes: Container Orchestration (many mins)

Prelude: Systems Projects at Google Seattle and Kirkland

Seattle:

- Chrome Cloud (incl. Flywheel)
  - (Matt Welch)
- Flume / Dataflow / Apache Beam
  - (Craig Chambers)
- Compute Engine VM Hypervisor
  - (Mike Dahlin)
- Kubernetes + Container Engine
  - (Alex Mohr)
- App Engine Flex
  - (Tomas Isdal)
- Cloud Storage
  - (?)
- $FOO
  - (Michael Piatek)

Kirkland:

- Cloud Machine Learning
  - (Mona Attariyan)
- Spanner
  - (?)
- Compute Engine's Control Plane
  - (Mike Dahlin)
- Compute Engine's Persistent Disk
  - (?)
- Thialfi notifications
  - (Atul Adya)

These are some of the (public) projects explicitly focused on systems. Other areas require systems knowledge too!

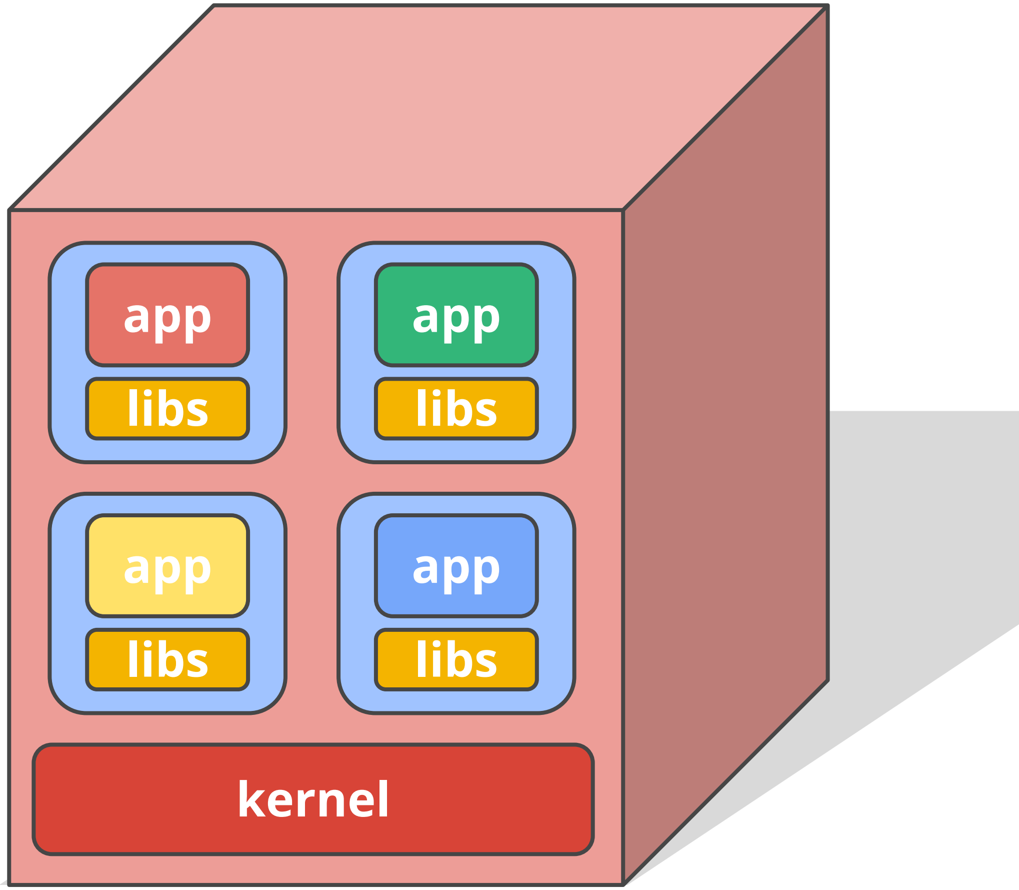
Contents

What are Containers? (Part 1: the Runtime)

Virtualize the kernel’s syscall interface
- no guest OS or hypervisor as with VMs

Isolation (from each other and from the host)
- chroots
- namespaces
- cgroups

Packaging
- hermetically sealed bundles
- no external dependencies
- no DLL hell
- portable from dev laptop to on-prem & clouds



What are Containers? (Part 2: Building an Image)

```
% cat - > Dockerfile
  FROM node:4.4
  EXPOSE 8080
  COPY server.js .
  CMD node server.js
```

What are Containers? (Part 2: Building an Image)

% cat Dockerfile
  FROM node:4.4
  EXPOSE 8080
  COPY server.js .
  CMD node server.js
% **docker build -t gcr.io/mohr-dev/hello-node:v1 .**
[log spam]

What are Containers? (Part 2: Building an Image)

```
% cat Dockerfile
  FROM node:4.4
  EXPOSE 8080
  COPY server.js .
  CMD node server.js
% docker build -t gcr.io/mohr-dev/hello-node:v1 .
[log spam]
% docker run -d -p 8080:8080 --name hello_tutorial gcr.io/mohr-dev/hello-node:v1
```

What are Containers? (Part 2: Building an Image)

```
% cat Dockerfile
  FROM node:4.4
  EXPOSE 8080
  COPY server.js .
  CMD node server.js
% docker build -t gcr.io/mohr-dev/hello-node:v1 .
[log spam]
% docker run -d -p 8080:8080 --name hello_tutorial gcr.io/mohr-dev/hello-node:v1
% curl http://localhost:8080/
Hello World!
```

What are Containers? (Part 3: Shipping an Image)

The magic:

% gcloud docker --authorize-only

% **docker push gcr.io/mohr-dev/hellonode:v1**

The push refers to a repository [gcr.io/mohr-dev/hellonode] (len: 1)

[...]

v1: digest: sha256:d2f8b1387c535de6d6752a7c02c107576e86f9435d275be861fa8c6df5a29c4d size: 12985

What are Containers? (Part 3: Shipping an Image)

The magic:

% gcloud docker --authorize-only

% **docker push gcr.io/mohr-dev/hellonode:v1**

The push refers to a repository [gcr.io/mohr-dev/hellonode] (len: 1)

[...]

v1: digest: sha256:d2f8b1387c535de6d6752a7c02c107576e86f9435d275be861fa8c6df5a29c4d size: 12985

Then, from any other machine:

% **docker pull gcr.io/mohr-dev/hellonode:v1**

v1: Pulling from mohr-dev/hellonode

Digest: sha256:d2f8b1387c535de6d6752a7c02c107576e86f9435d275be861fa8c6df5a29c4d

Status: Image is up to date for gcr.io/mohr-dev/hellonode:v1

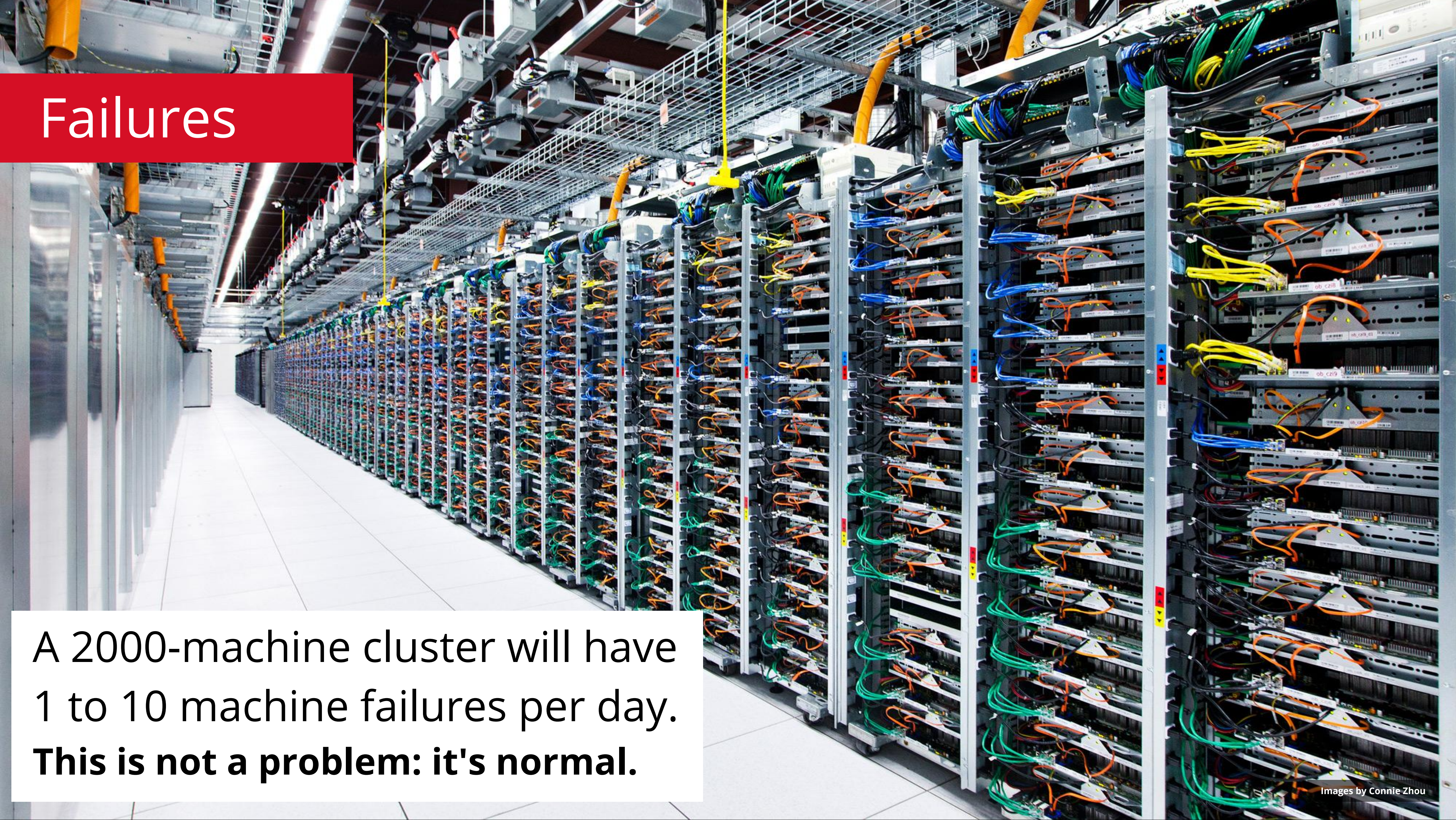
% **docker run $ARGS gcr.io/mohr-dev/hellonode:v1**

...

Contents

Image by Connie Zhou

Failures



A 2000-machine cluster will have 1 to 10 machine failures per day.

This is not a problem: it's normal.

Images by Connie Zhou

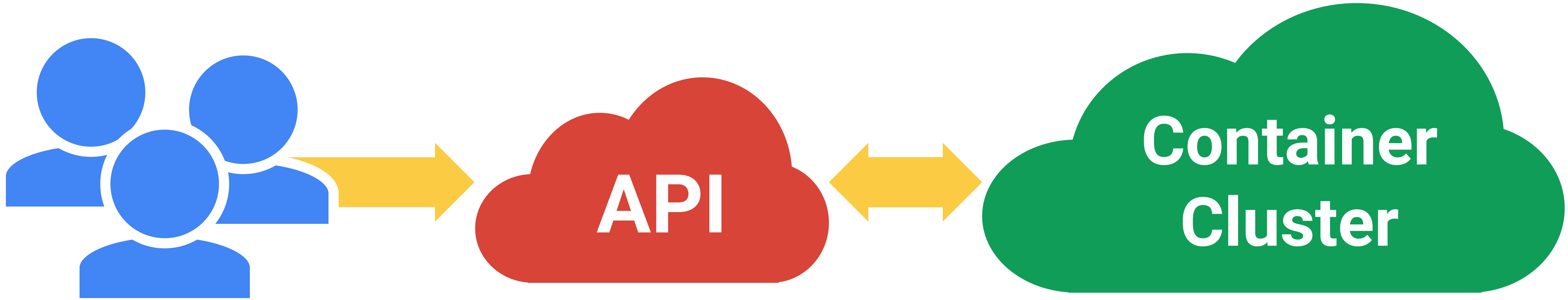
Kubernetes

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

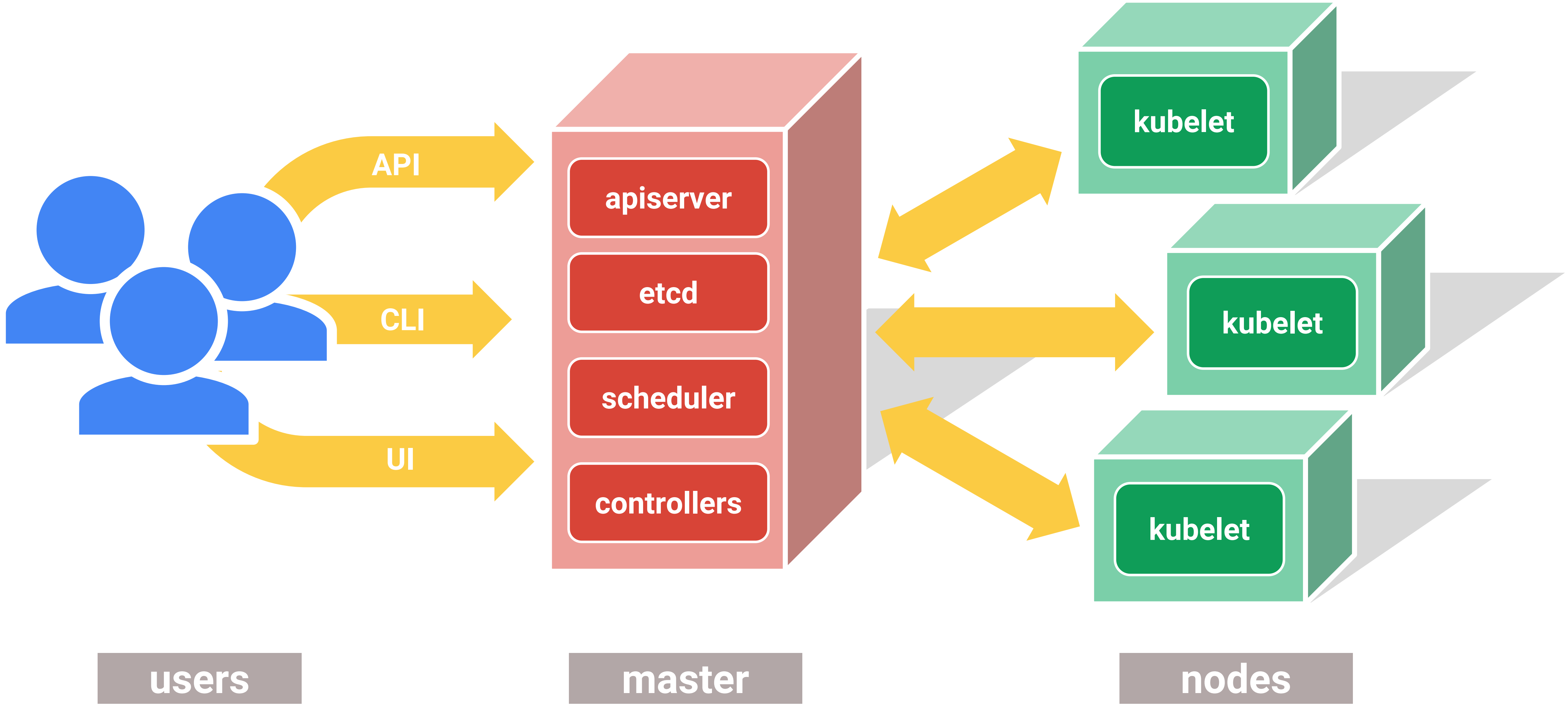
- Manages container clusters
- Inspired and informed by Google's experiences and internal systems
- Supports multiple cloud and bare-metal environments
- Supports multiple container runtimes
- **100% Open source**, written in Go

Manage <u>applications</u>, not machines

All you really care about



The 10000 foot view



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

This is where things like **Google Container Engine (GKE)** really help

Container clusters: A story in two parts

2. Using the cluster

- Run Pods & Containers
- ReplicaSets & Deployments & DaemonSets & StatefulSets
- Services & Volumes & Secrets & Autoscalers

This is the fun part!

A distinct set of problems from cluster setup and management

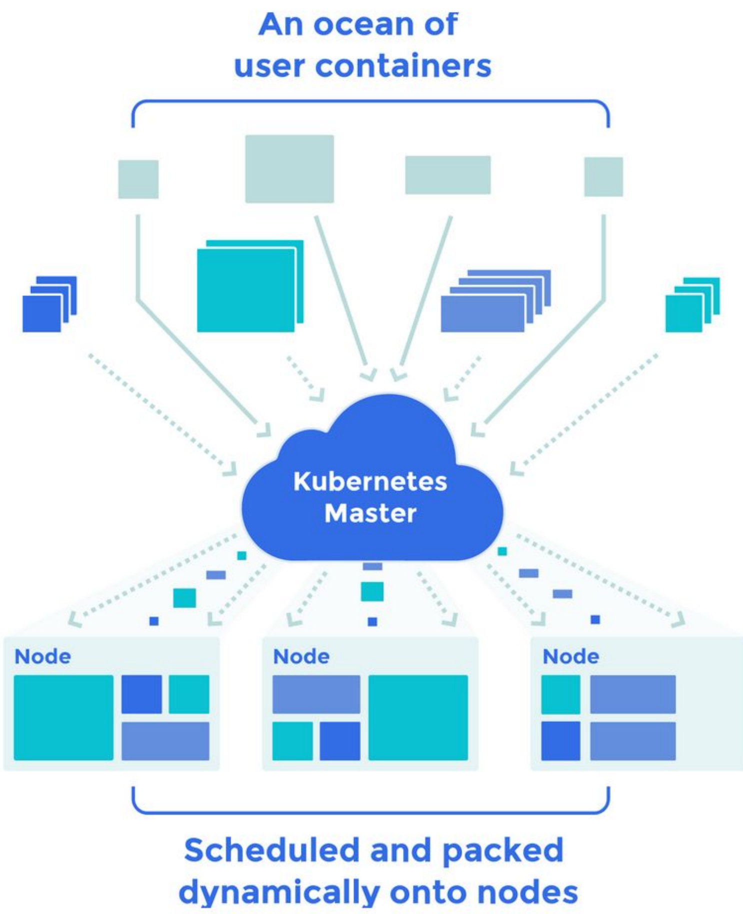
Don’t make developers deal with cluster administration!

Accelerate development by focusing on the applications, not the cluster

Kubernetes: a Cloud OS?

Perhaps grandiose, but attempts at "Cloud OS" primitives:

- **Scheduling**: Decide where my containers should run
- **Lifecycle and health**: Keep my containers running despite failures
- **Scaling**: Make sets of containers bigger or smaller
- **Naming and discovery**: Find where my containers are now
- **Load balancing:** Distribute traffic across a set of containers
- **Storage volumes**: Provide data to containers
- **Logging and monitoring**: Track what's happening with my containers
- **Debugging and introspection**: Enter or attach to containers
- **Identity and authorization**: Control who can do things to my containers



Workload Portability

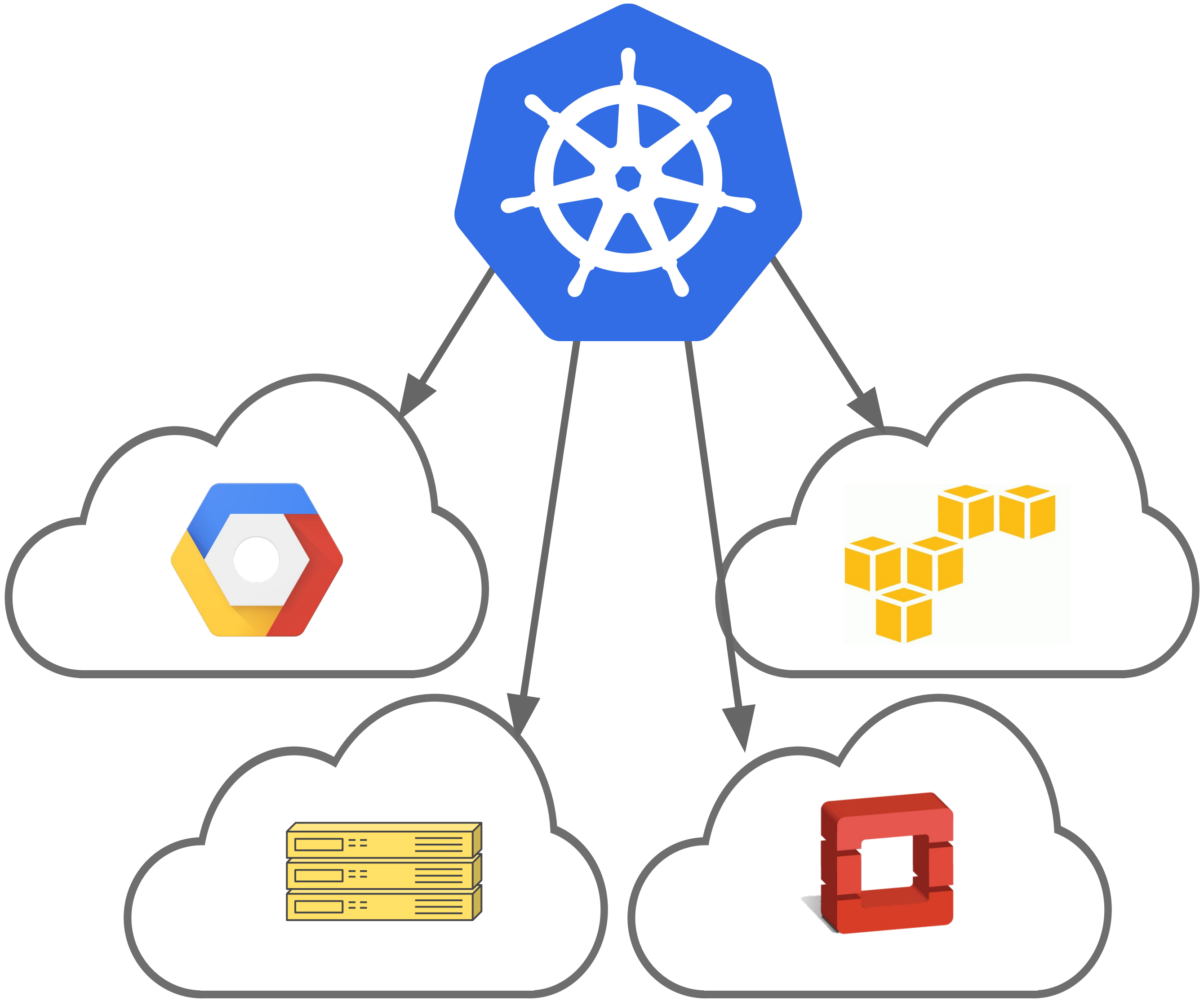
Workload portability

Goal: Avoid vendor lock-in

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

The API and the implementation are 100% open

The whole system is modular and replaceable



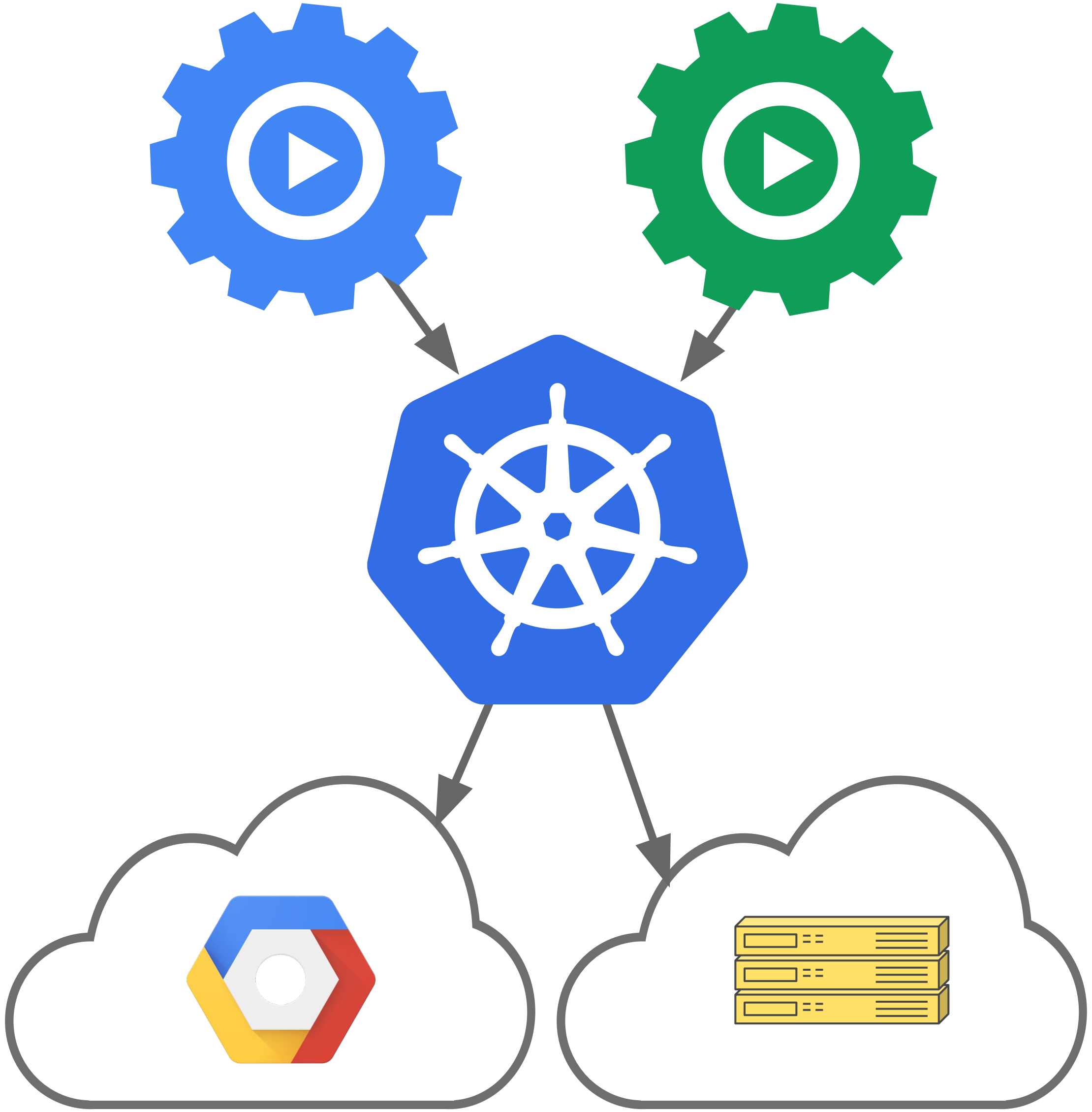
Workload portability

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*

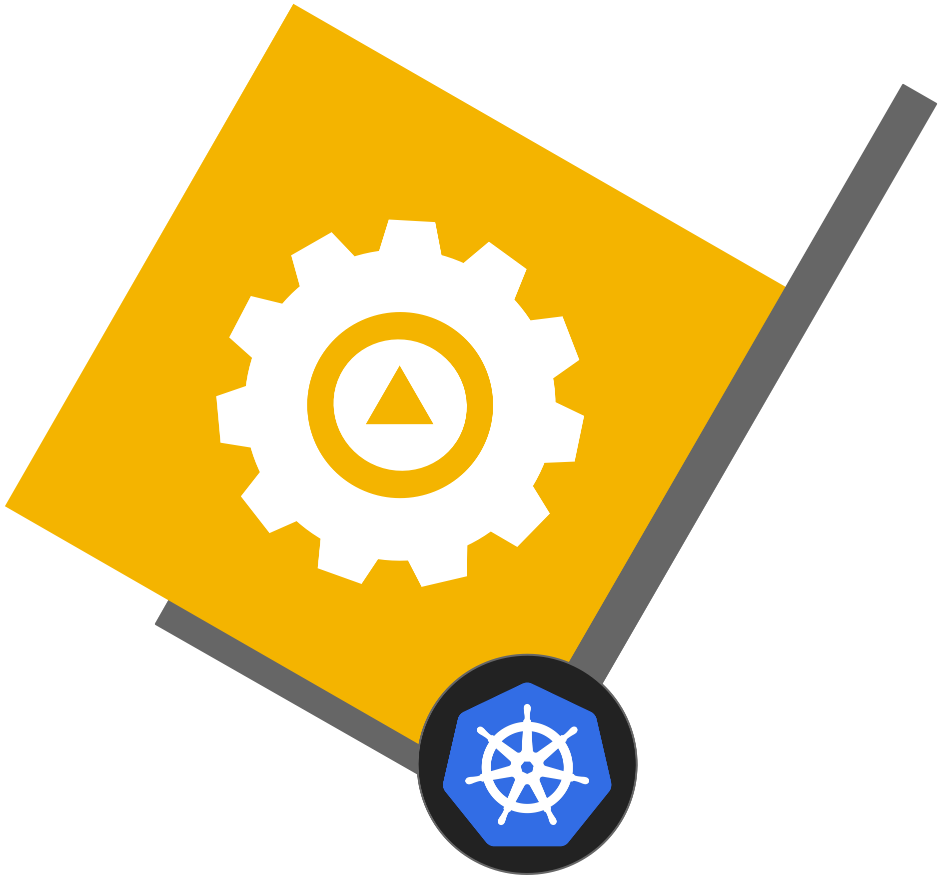
Workload portability

Result: Portability

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

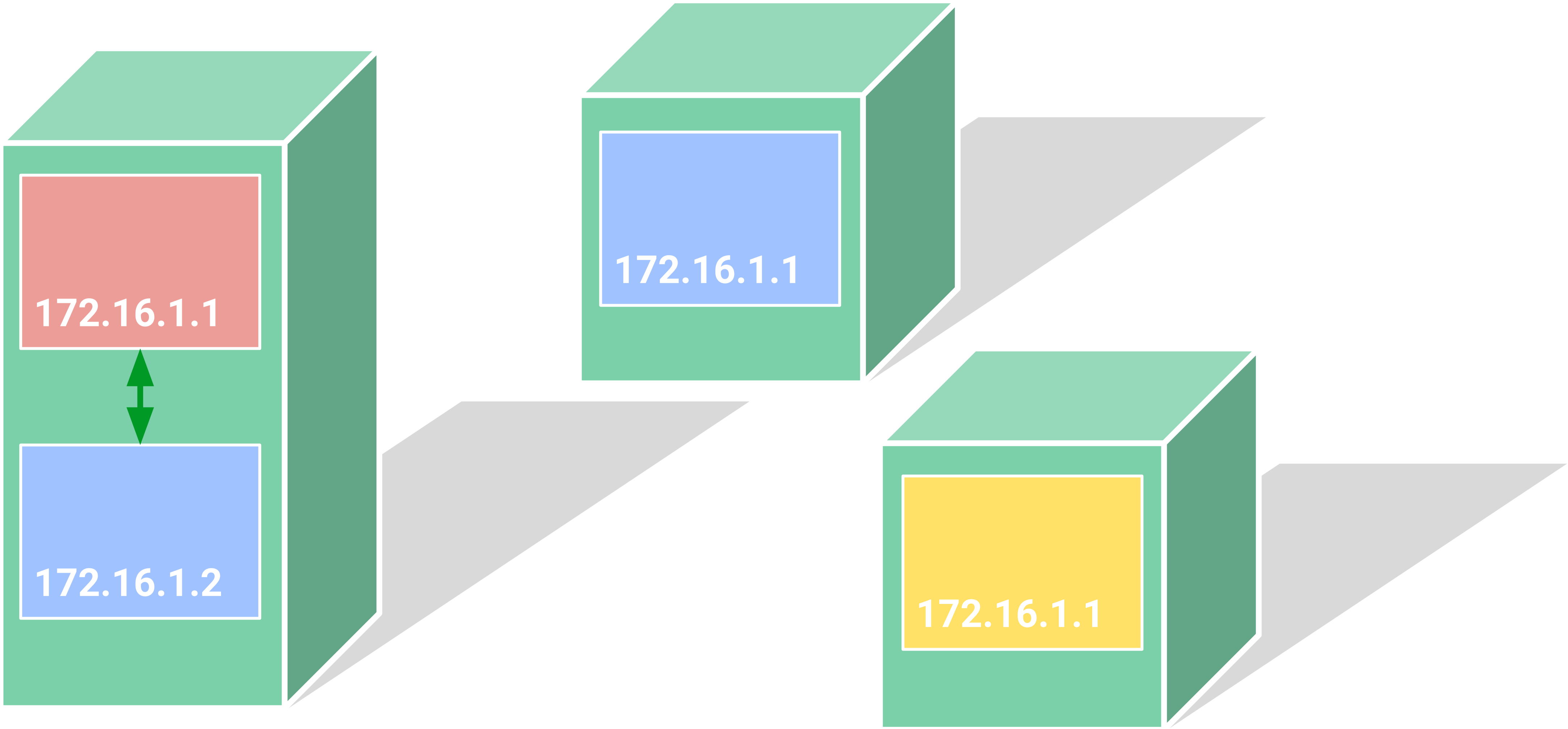
Don't get stuck with a platform that doesn't work for you

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

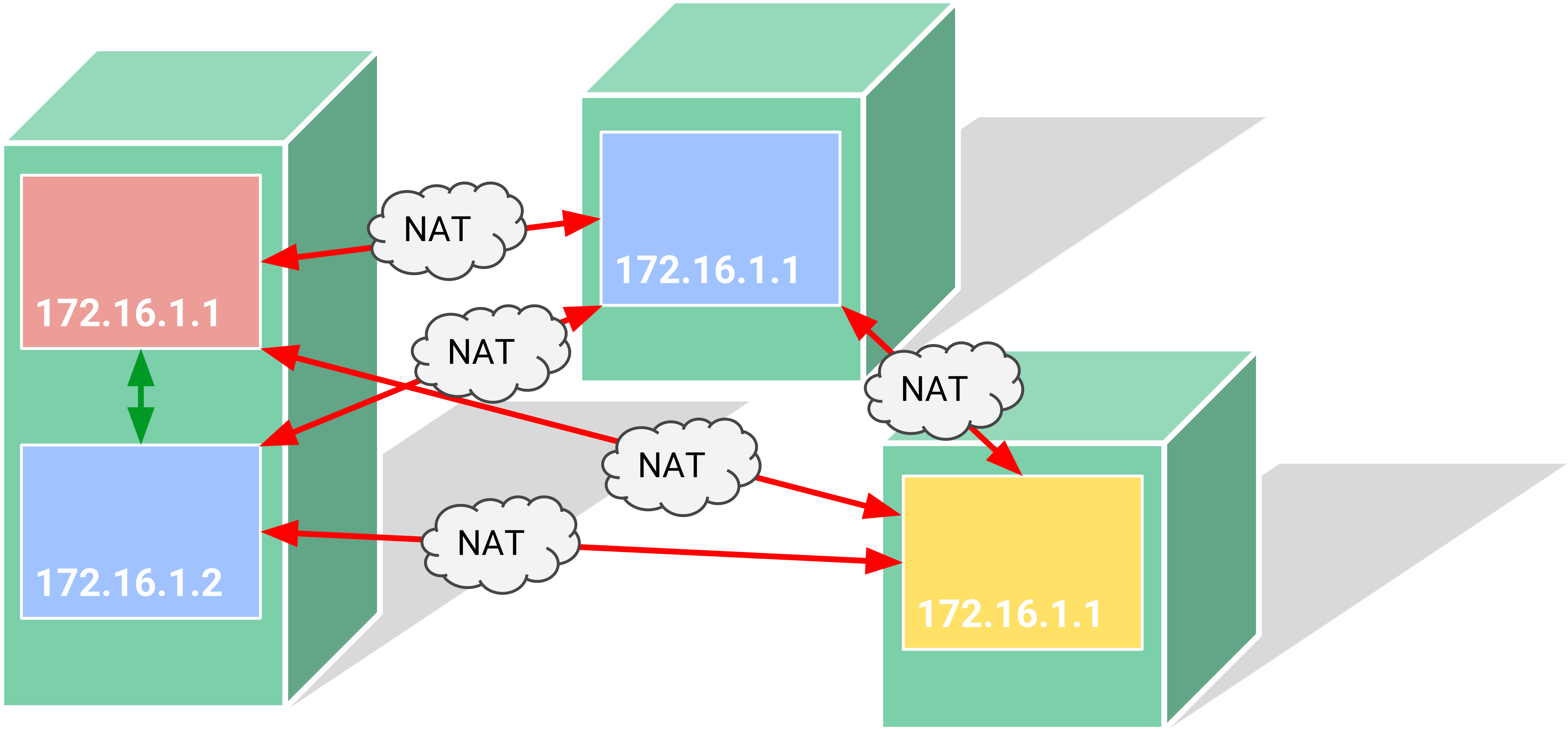


Networking

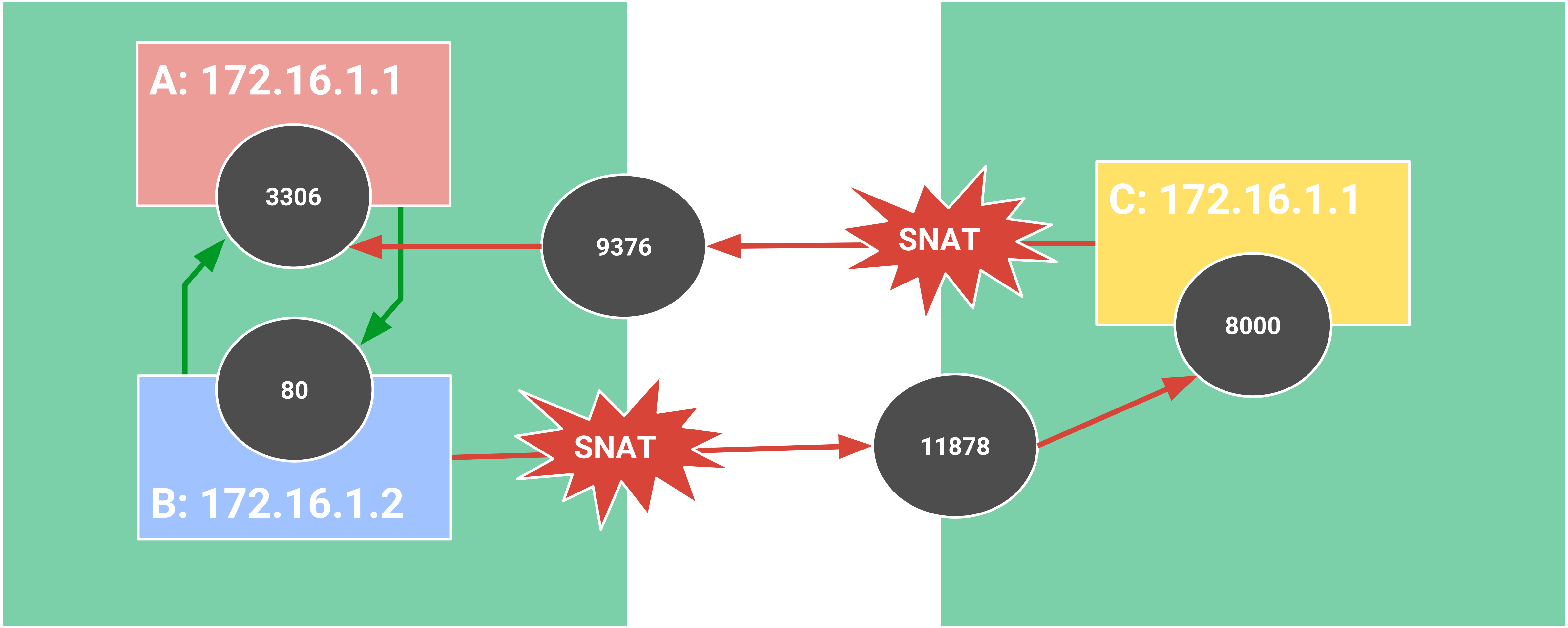
Docker networking



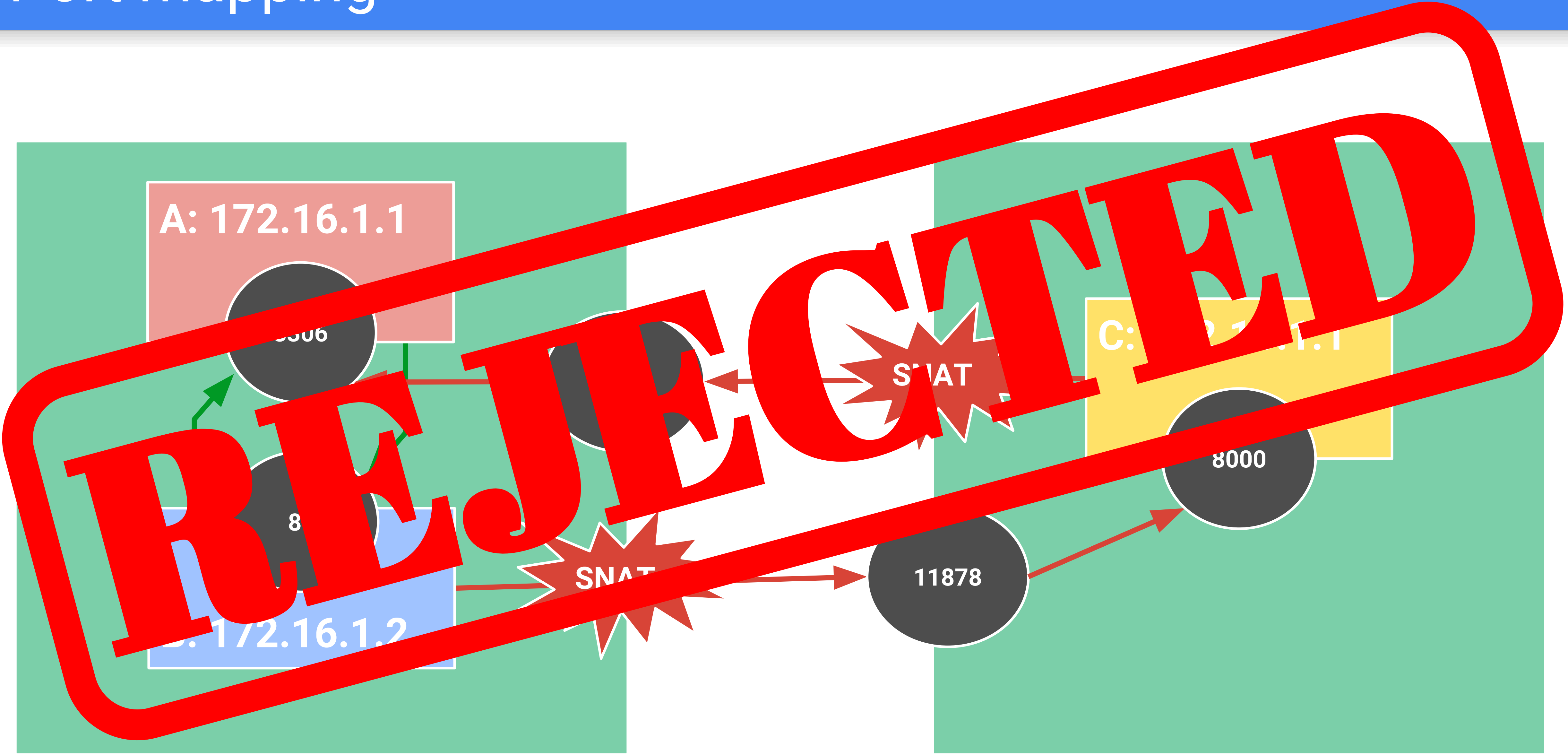
Docker networking



Port mapping



Port mapping



Kubernetes networking

IPs are **cluster-scoped**

- vs docker default private IP

Pods can reach each other directly

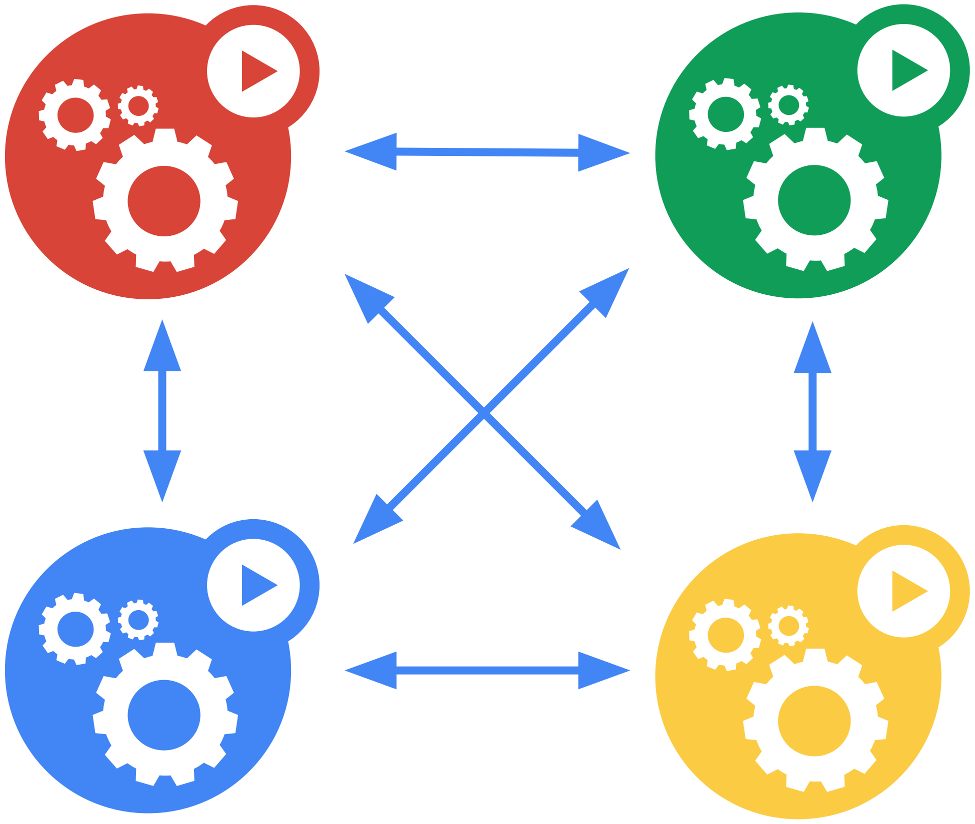
- even across nodes

No brokering of port numbers

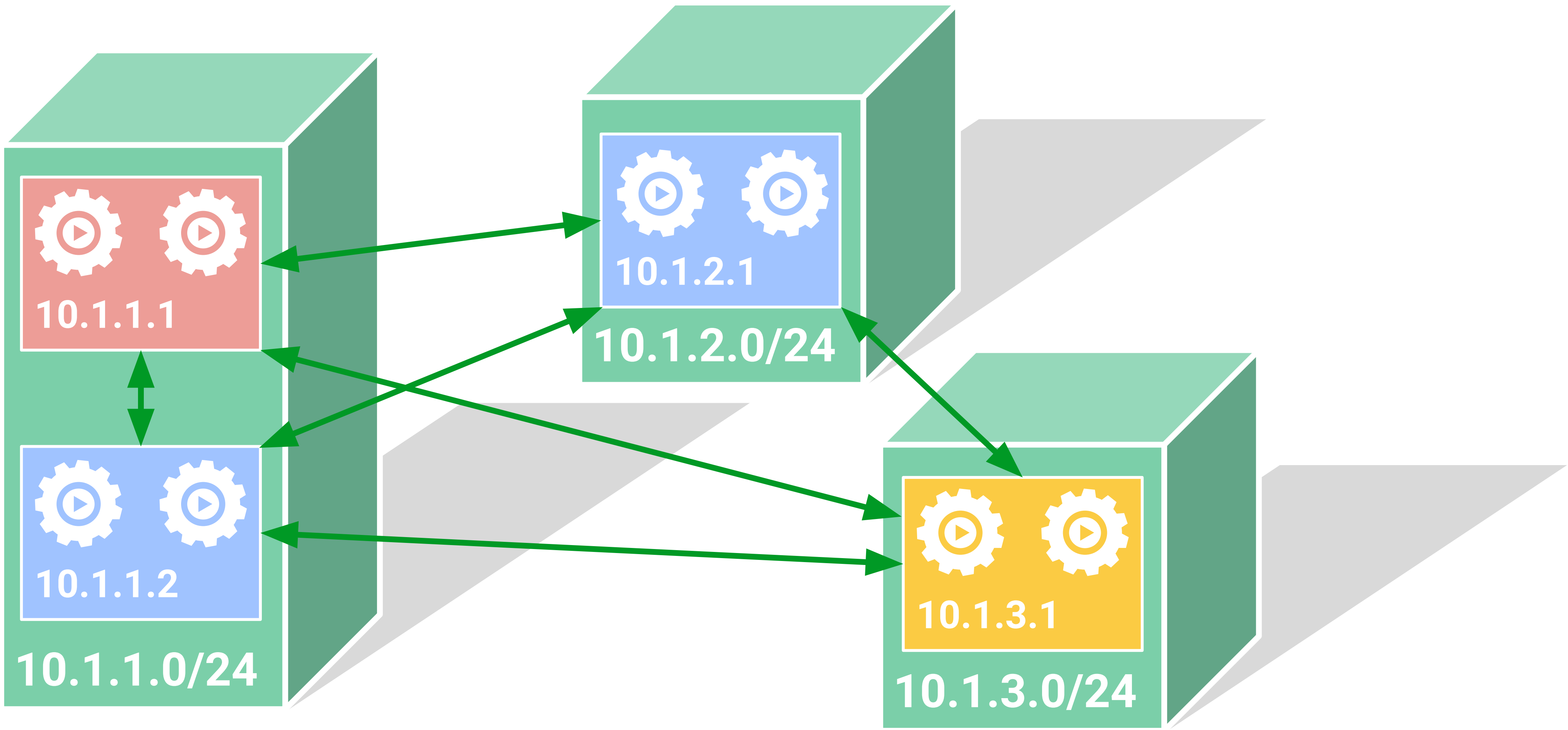
- too complex, why bother?

This is a fundamental requirement

- can be L3 routed
- can be underlayed (cloud)
- can be overlayed (SDN)



Kubernetes networking



Pods

Pods

Small group of containers & volumes

Tightly coupled

The atom of scheduling & placement

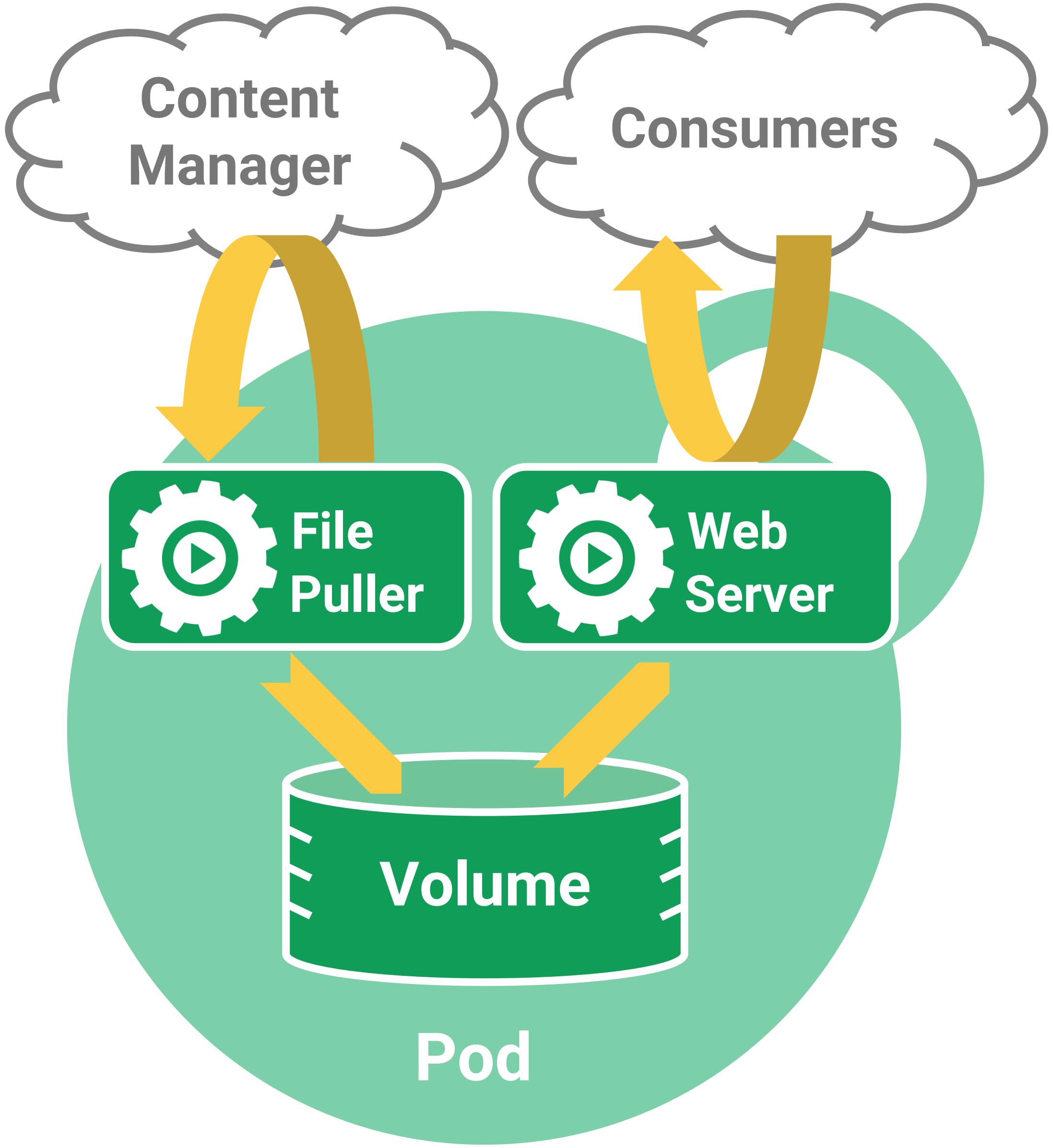
Shared namespace

- 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
- Flocker
- NFS
- vSphere
- GlusterFS
- Ceph File and RBD
- Cinder
- FibreChannel
- Secret, ConfigMap, DownwardAPI
- Flex (exec a binary)
- ...

Labels & Selectors

Labels

Arbitrary metadata

Attached to any API object

Generally represent **identity**

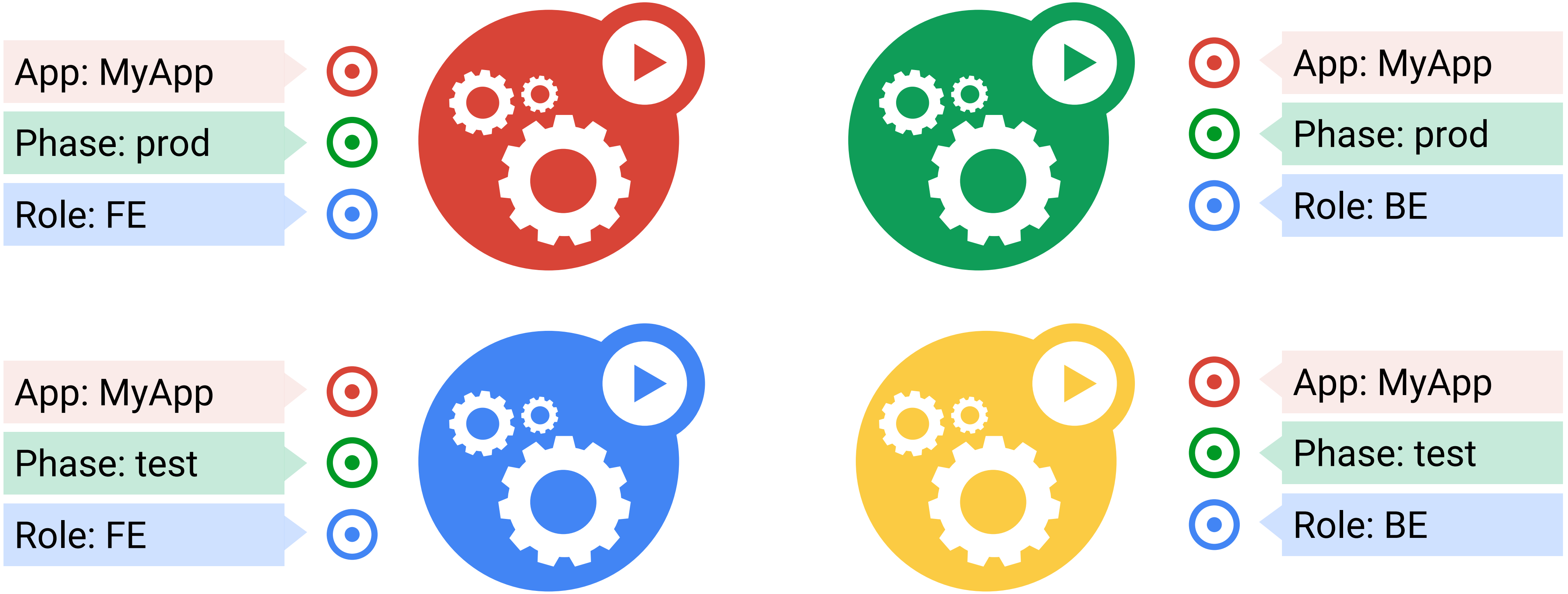
Queryable by **selectors**

- think SQL ‘*select ... where ...*’

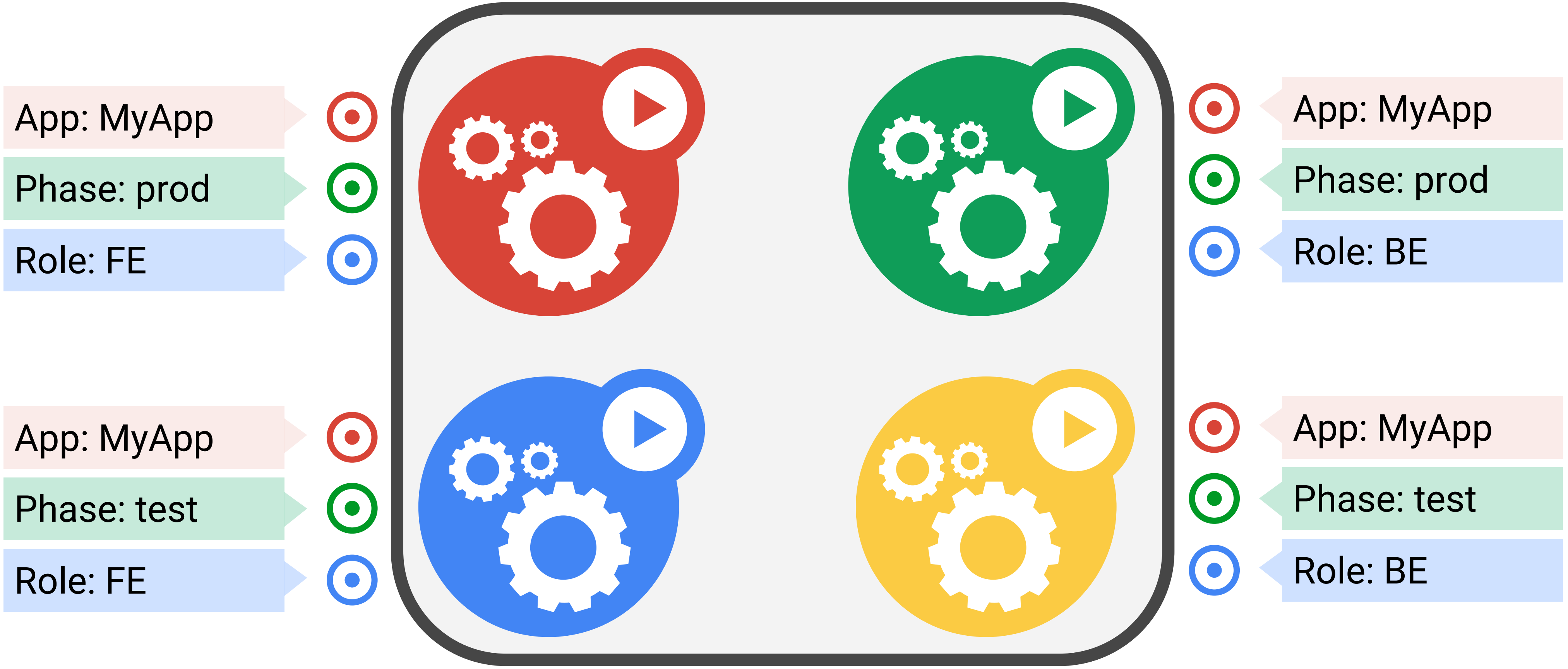
The **only** grouping mechanism

- pods under a ReplicaSet
- pods in a Service
- capabilities of a node (constraints)

Selectors

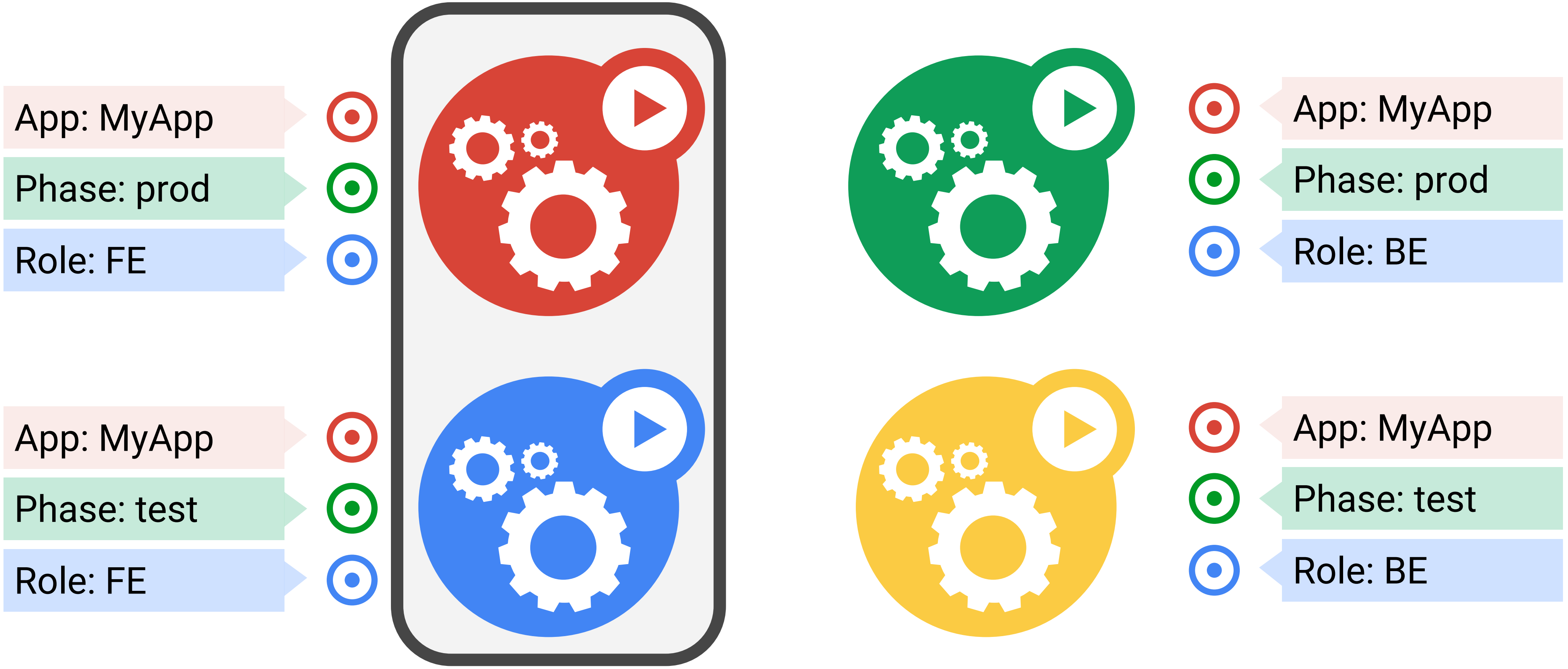

App: MyApp
Phase: prod
Role: FE
App: MyApp
Phase: prod
Role: BE
App: MyApp
Phase: test
Role: FE
App: MyApp
Phase: test
Role: BE

Selectors



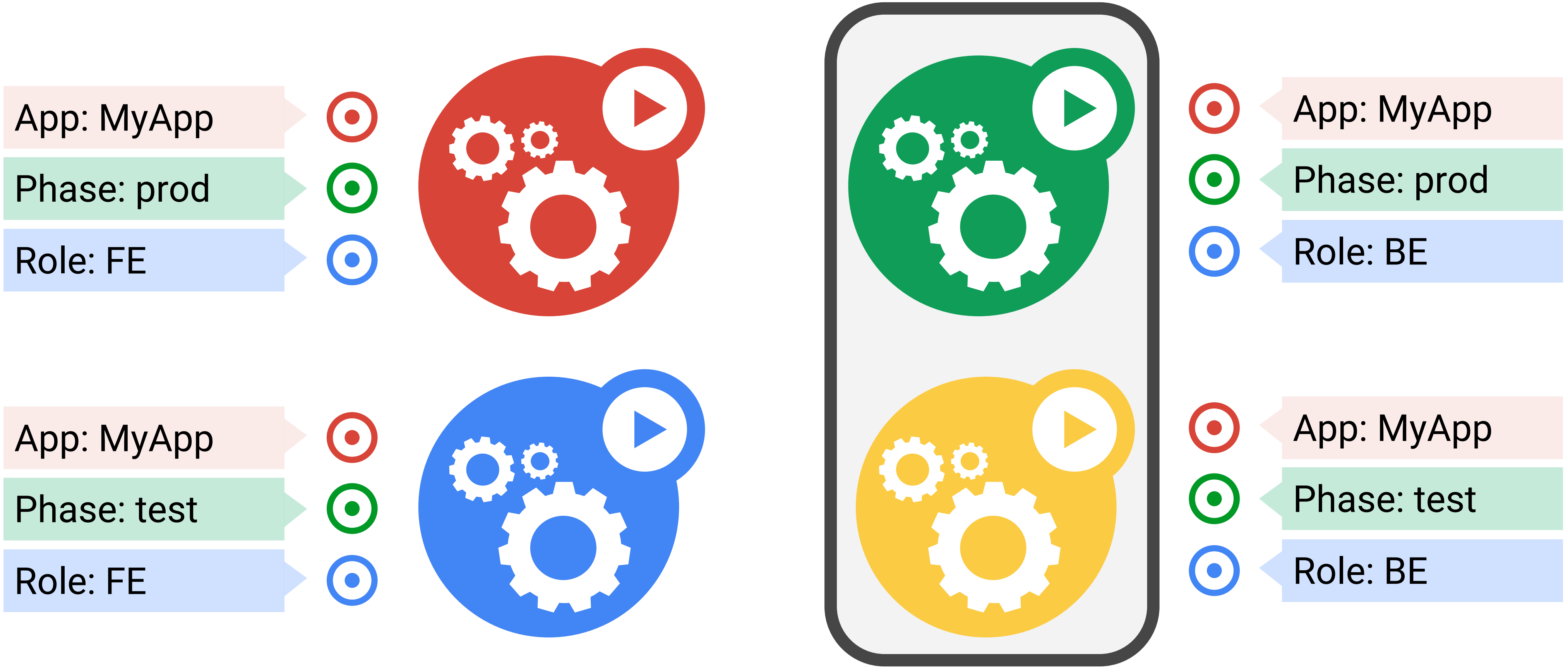
App = MyApp

Selectors



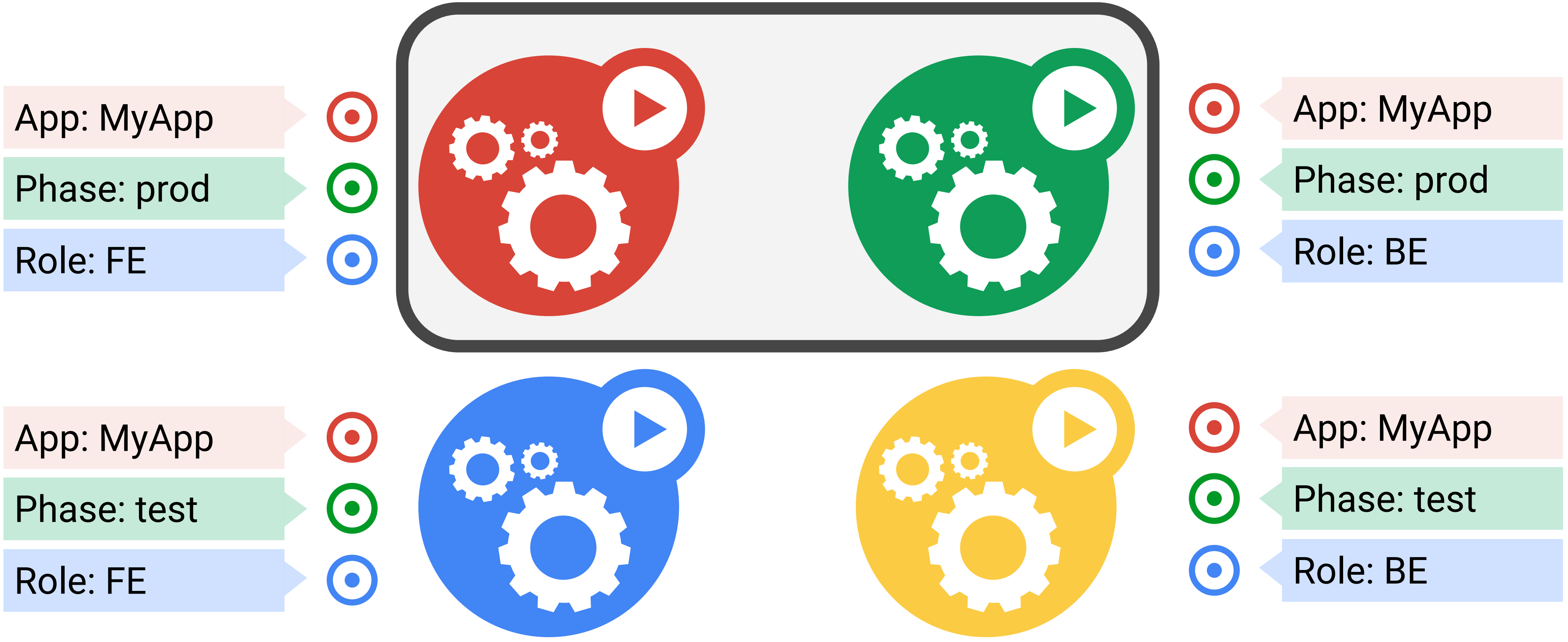
App = MyApp, Role = FE

Selectors



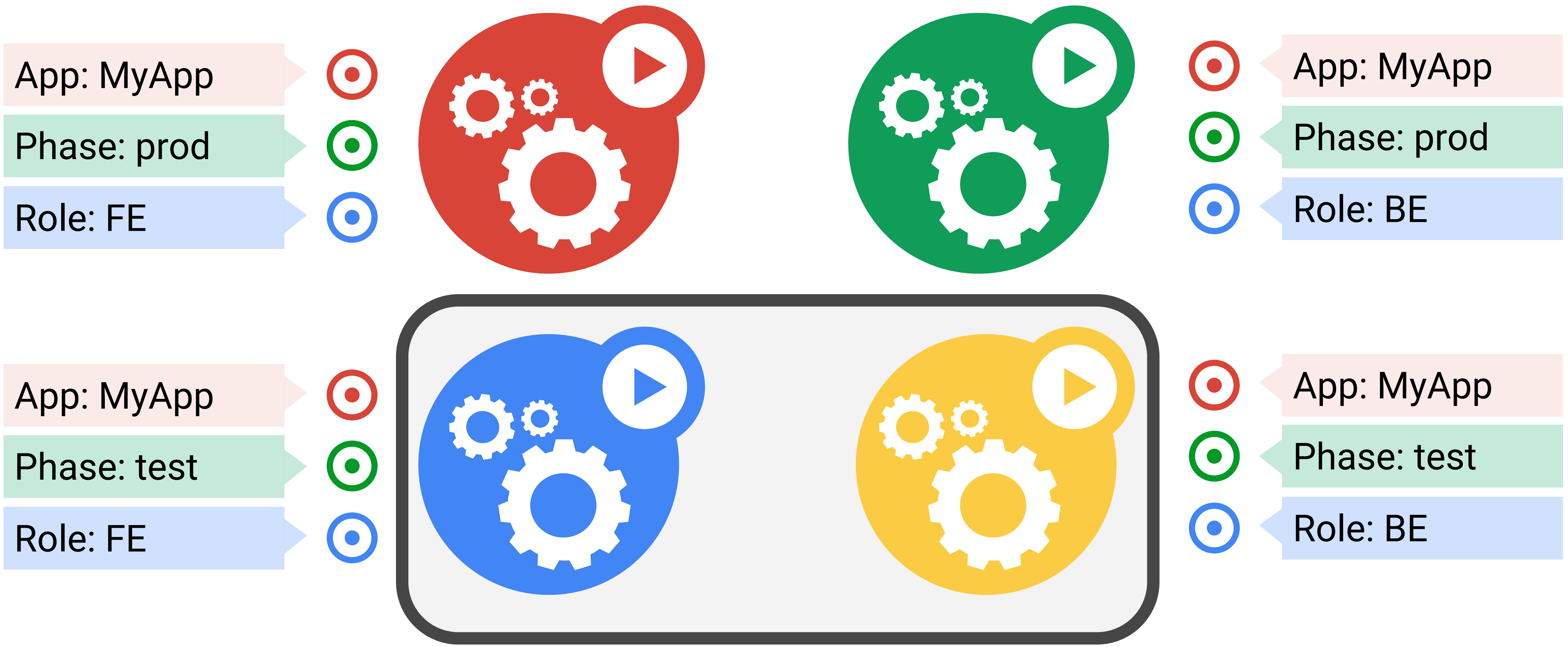
App = MyApp, Role = BE

Selectors



App = MyApp, Phase = prod

Selectors



App = MyApp, Phase = test

Replication

ReplicaSets

A simple control loop

Runs out-of-process wrt API server

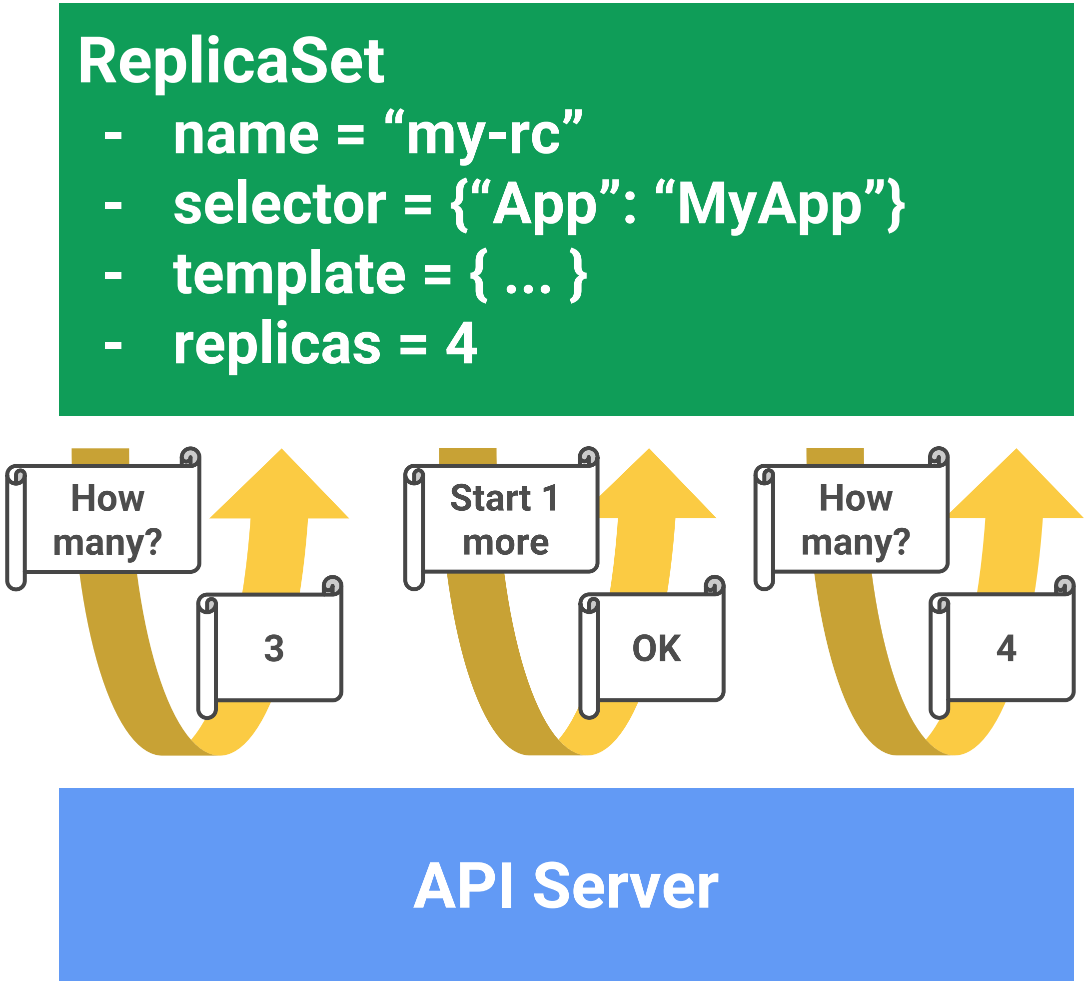
One job: ensure N copies of a pod

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

Layered on top of the public Pod API

Replicated pods are **fungible**

- No implied order or identity



Control loops: the Reconciler Pattern

Drive **current state -> desired state**

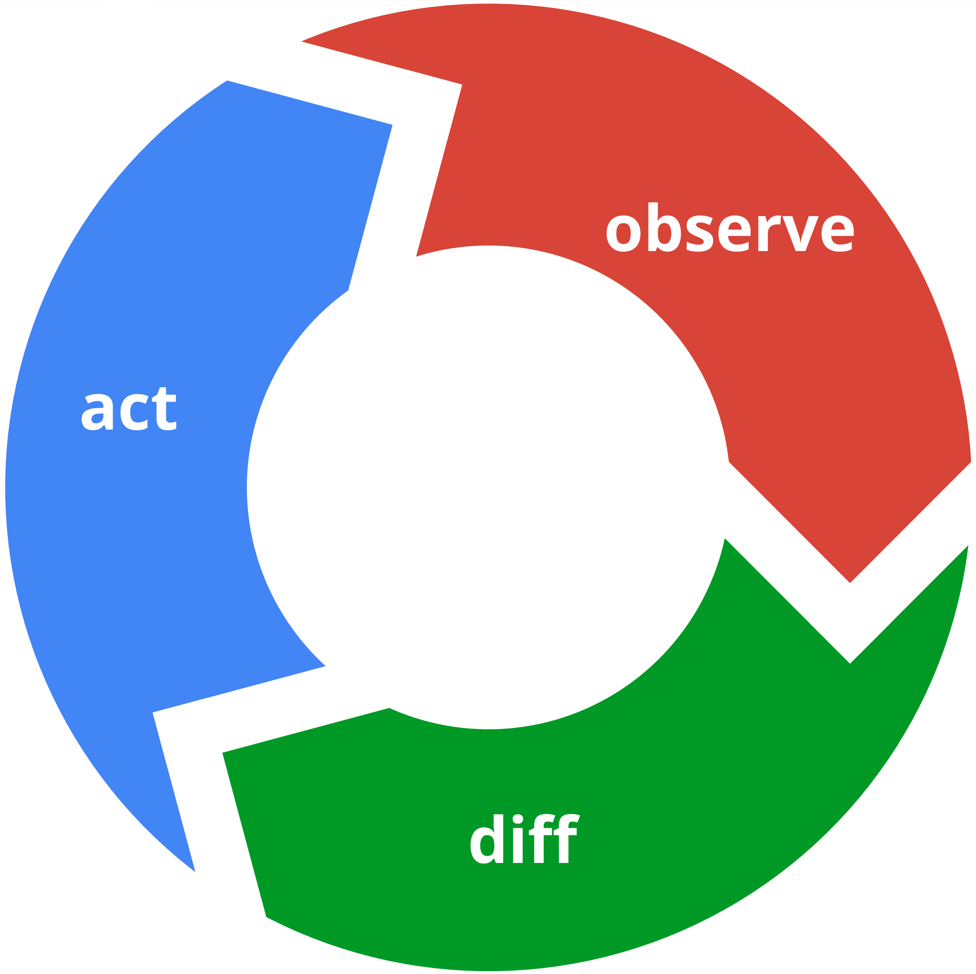
Act independently

APIs - **no shortcuts** or back doors

Observed state is truth*

Recurring pattern in the system

Example: ReplicaSet



* Observations are really stale caches of what once was your view of truth.

Services

Services

A group of **pods that work together**

- grouped by a selector

Defines access policy

- “load balanced” or “headless”

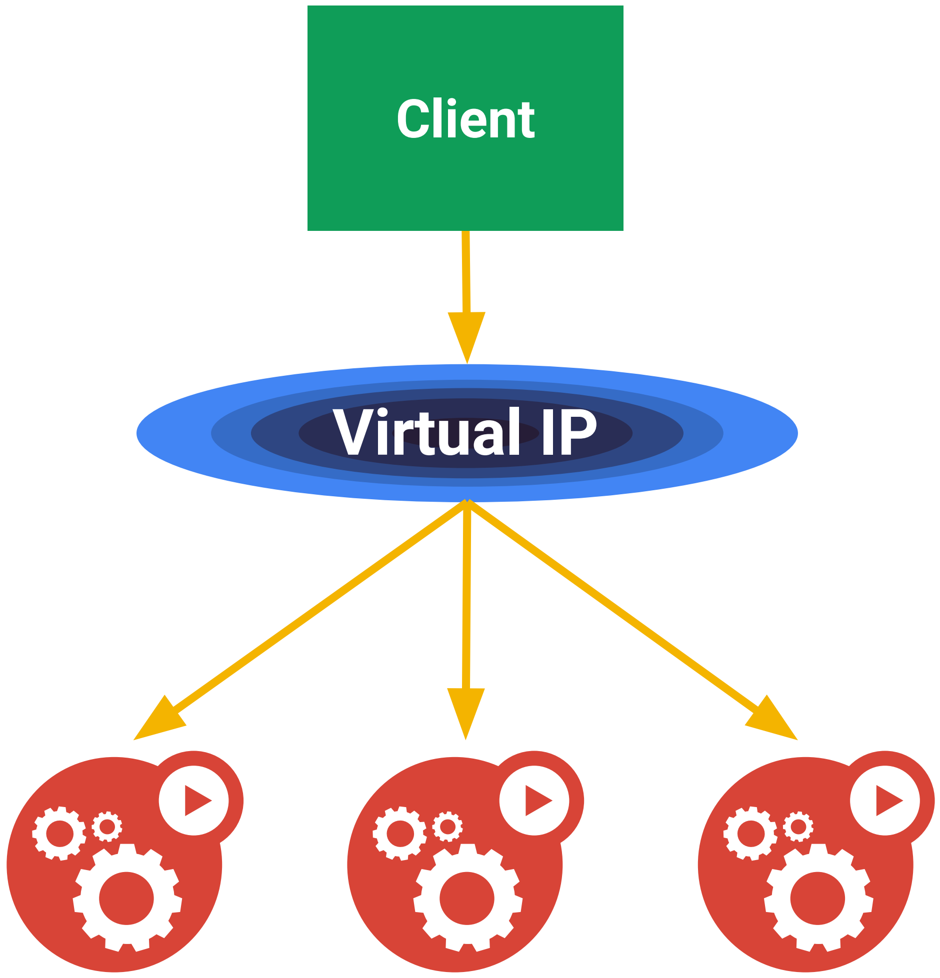
Can have a stable **virtual IP** and port

- also a DNS name

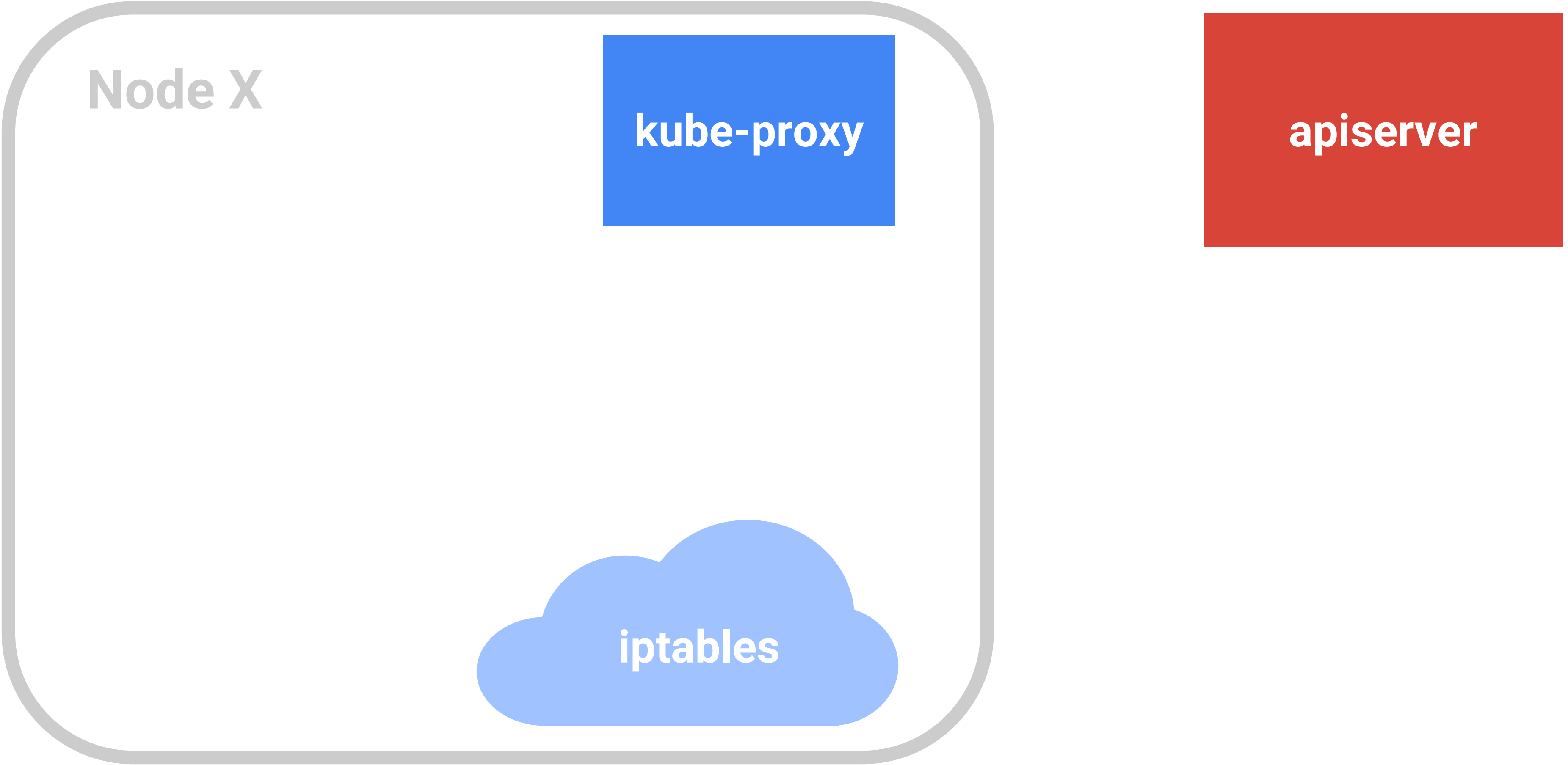
VIP is managed by *kube-proxy*

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

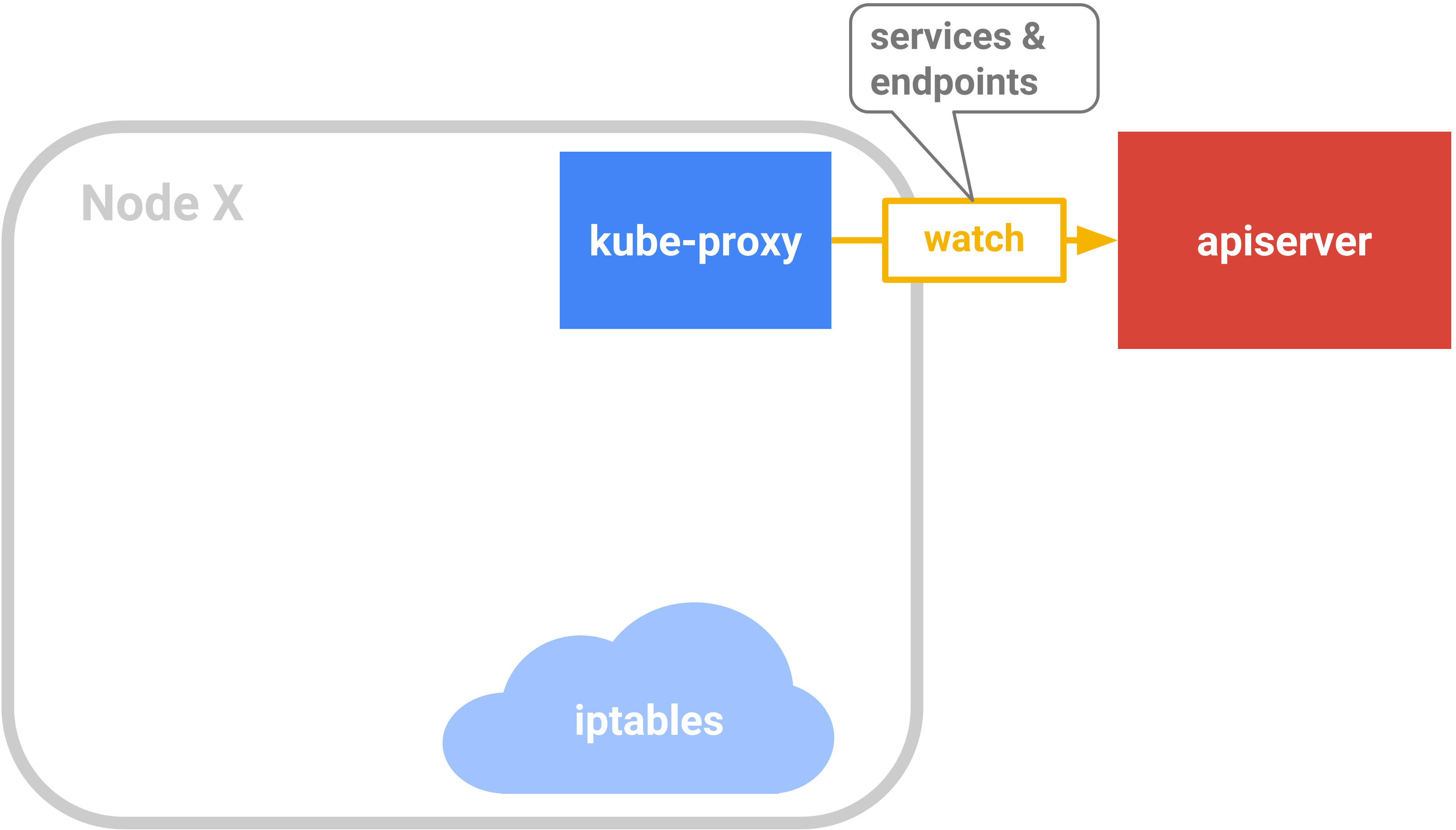
Hides complexity



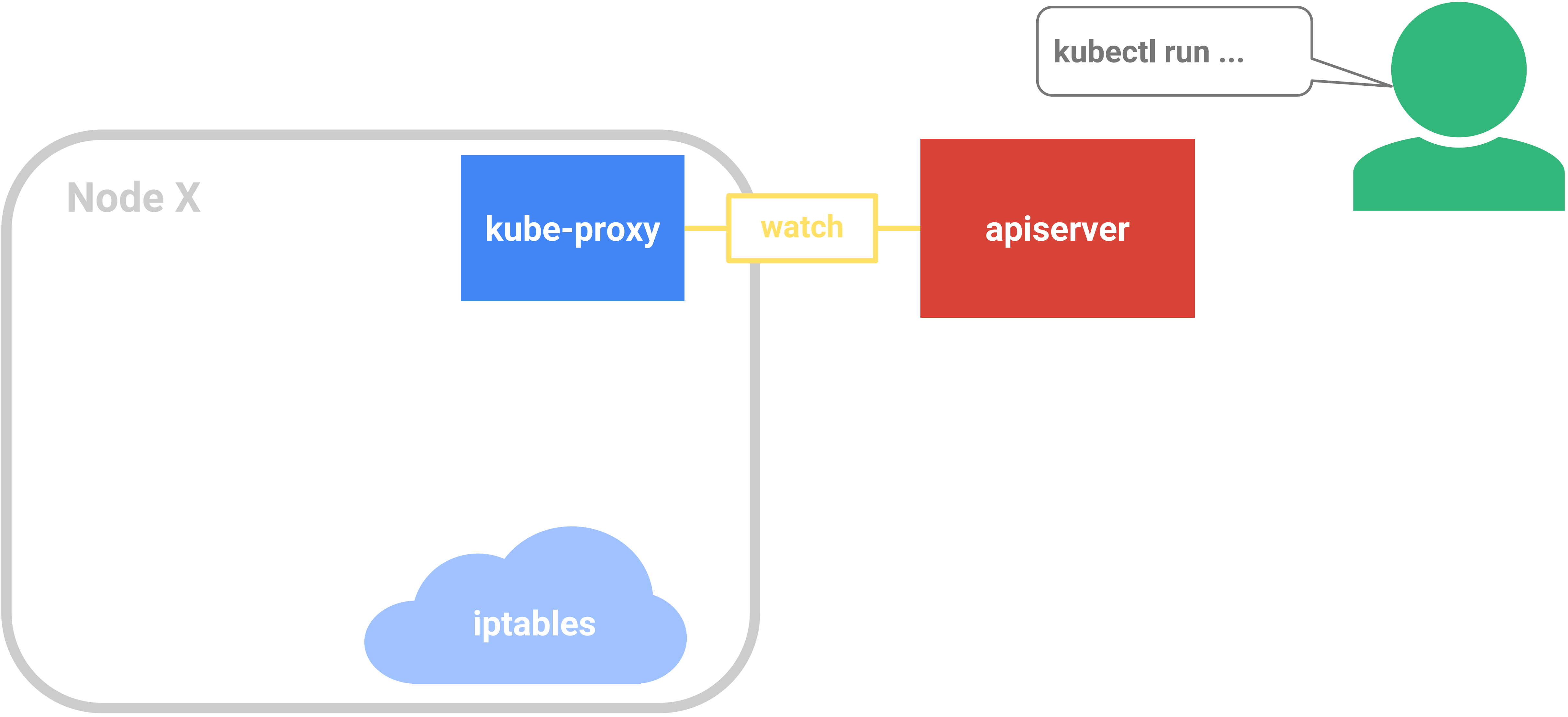
iptables kube-proxy



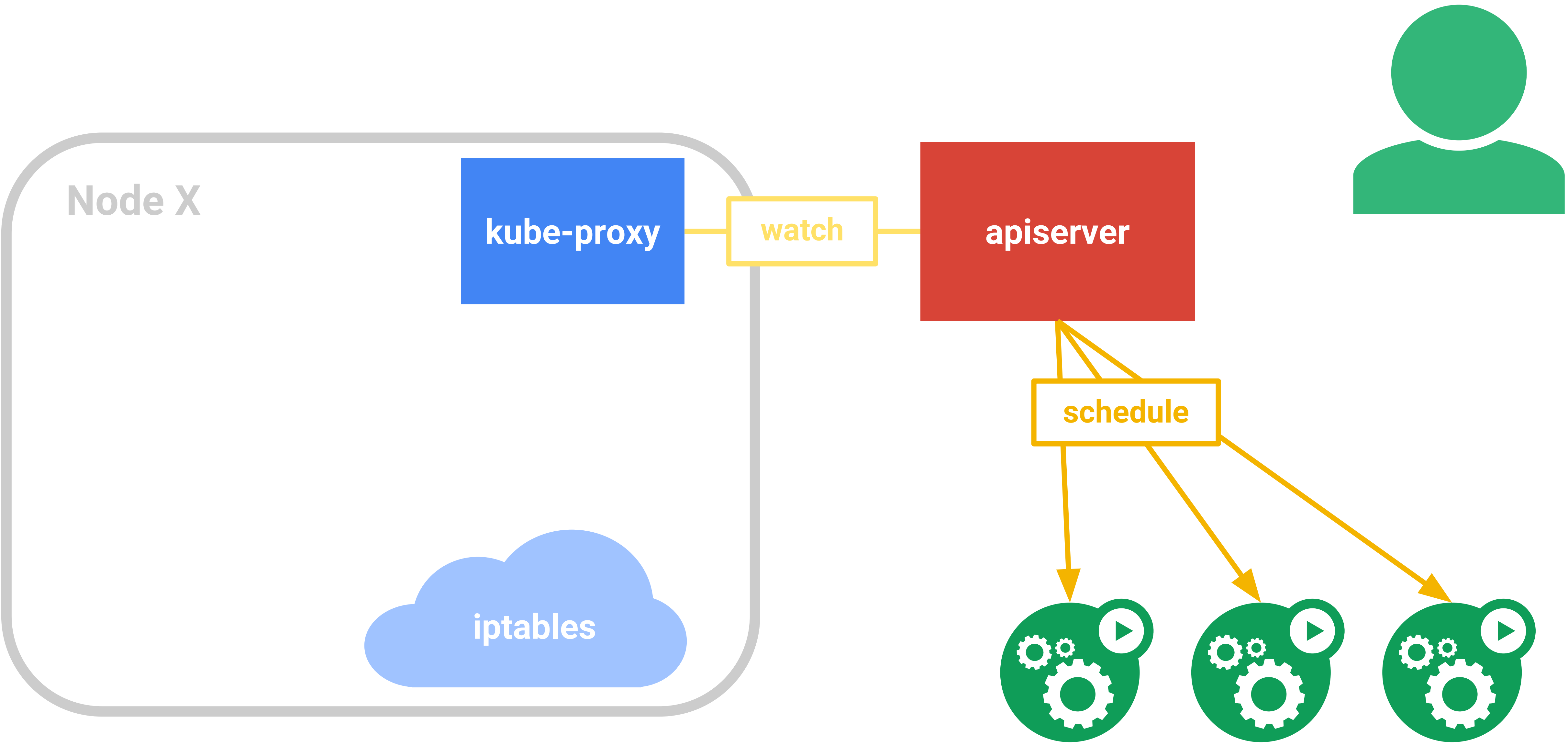
iptables kube-proxy



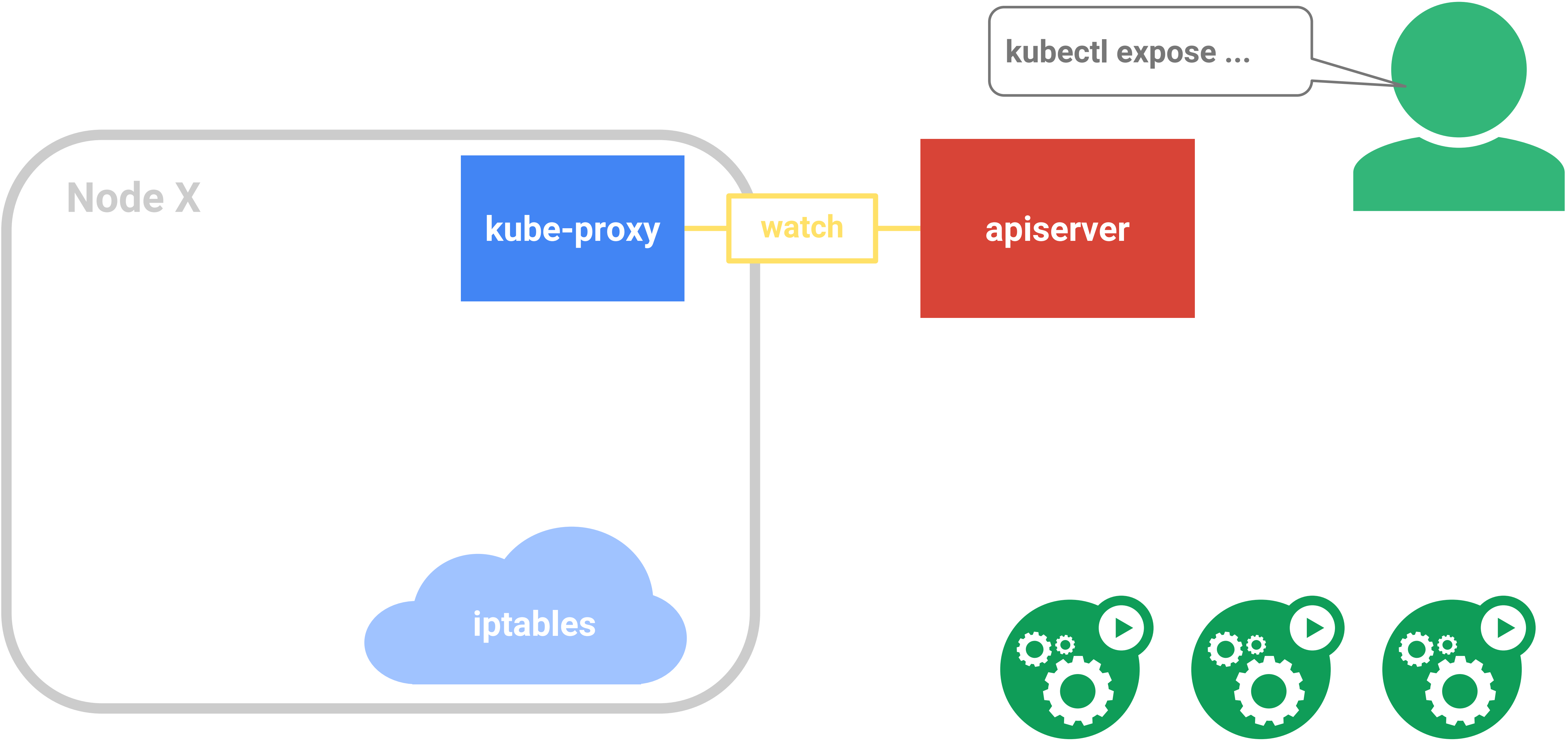
iptables kube-proxy



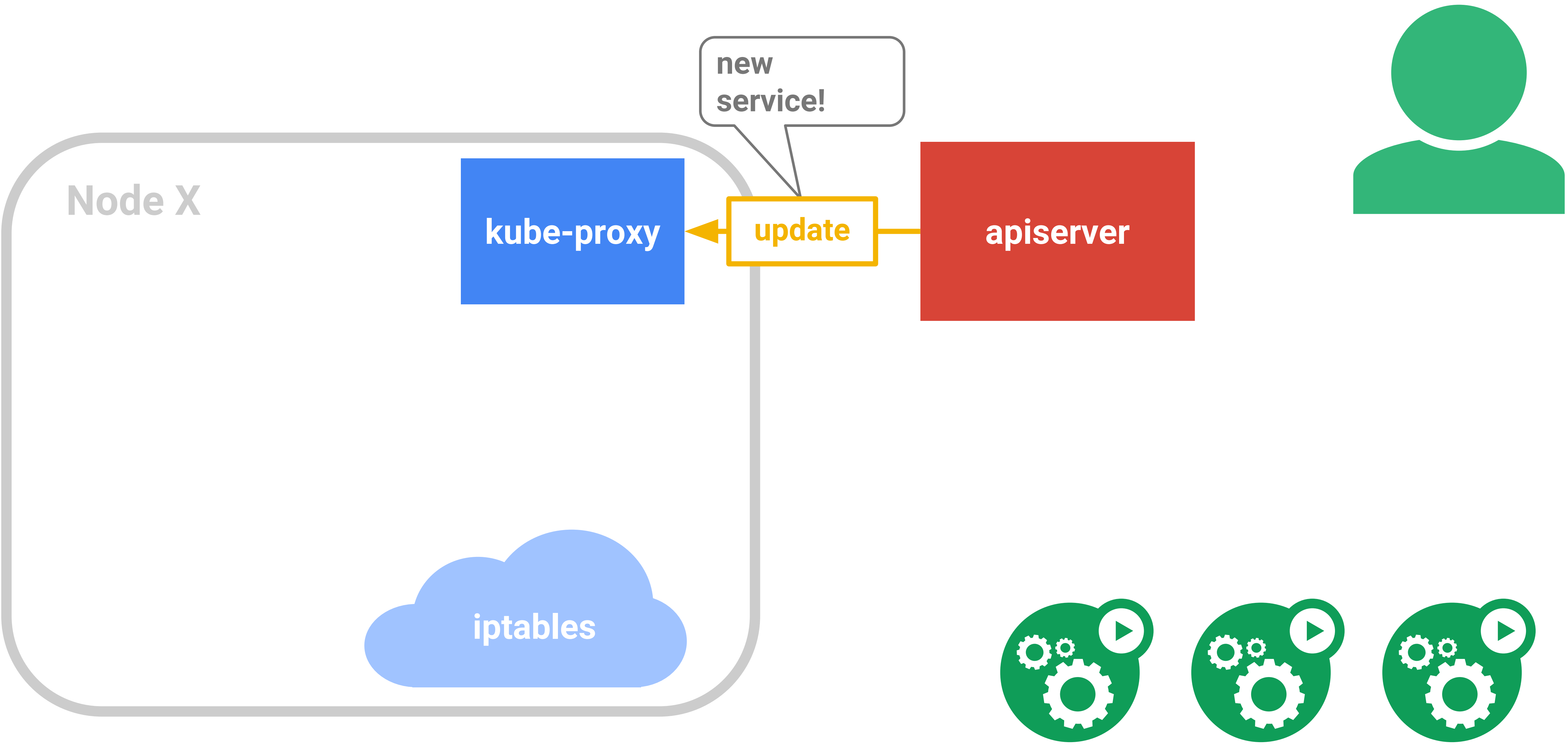
iptables kube-proxy



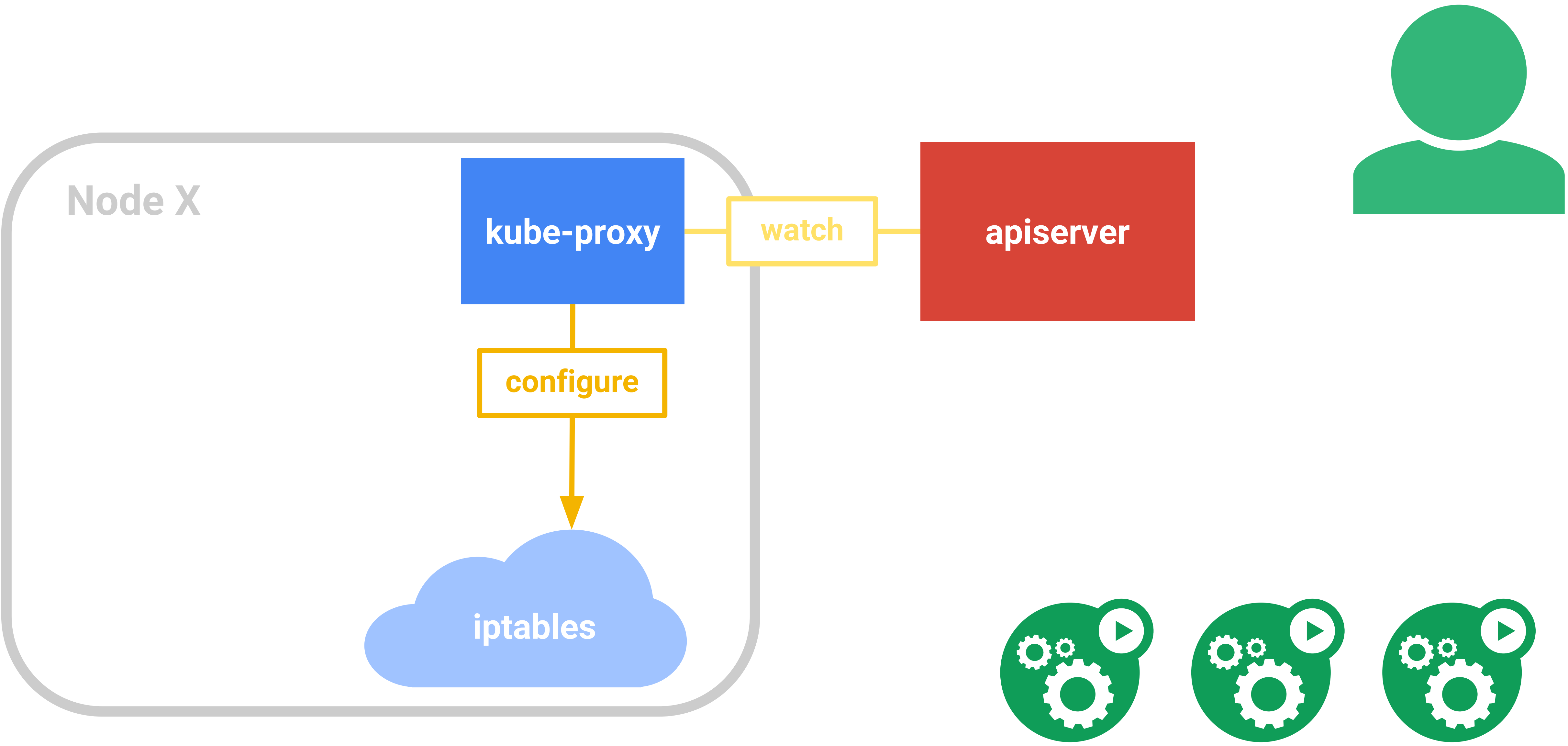
iptables kube-proxy



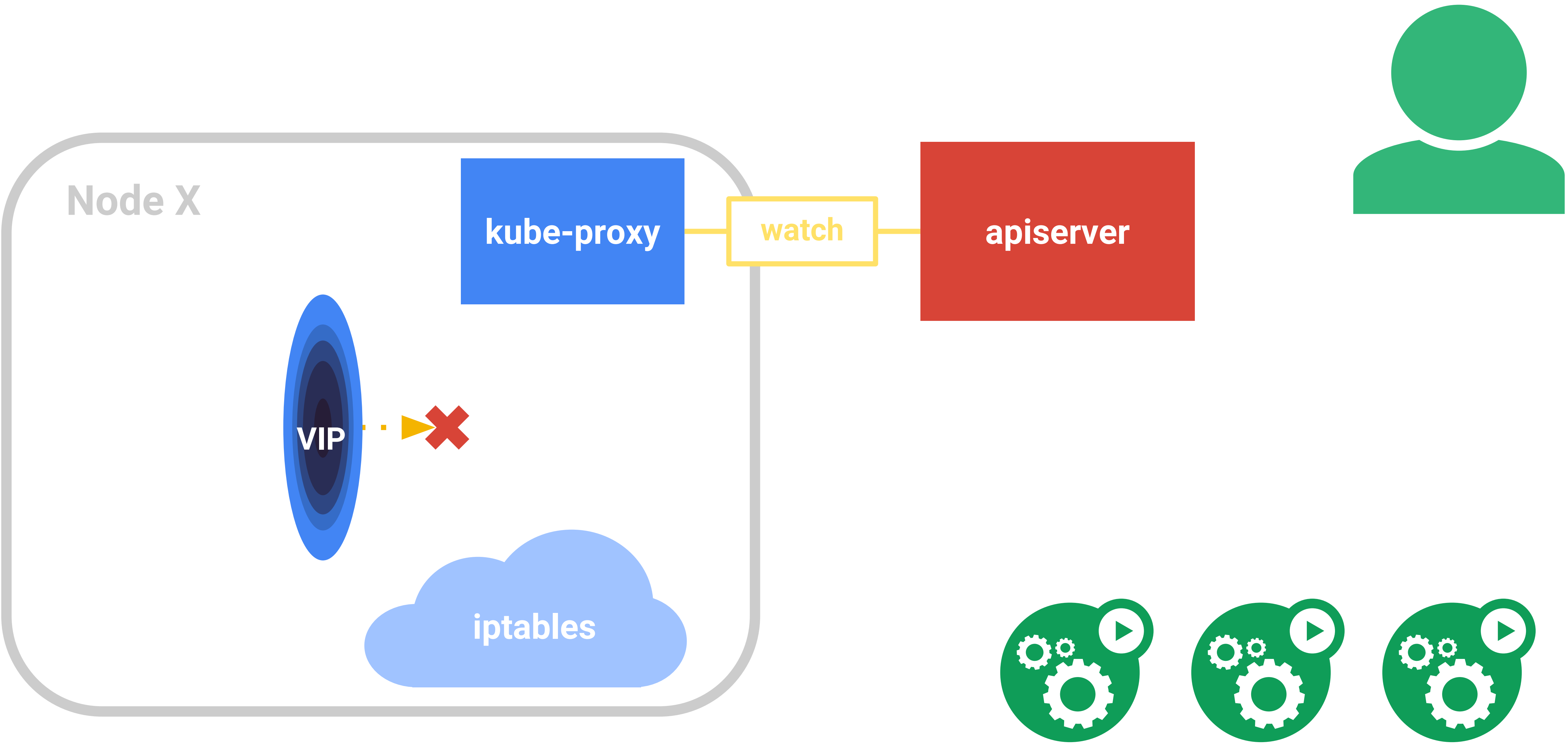
iptables kube-proxy



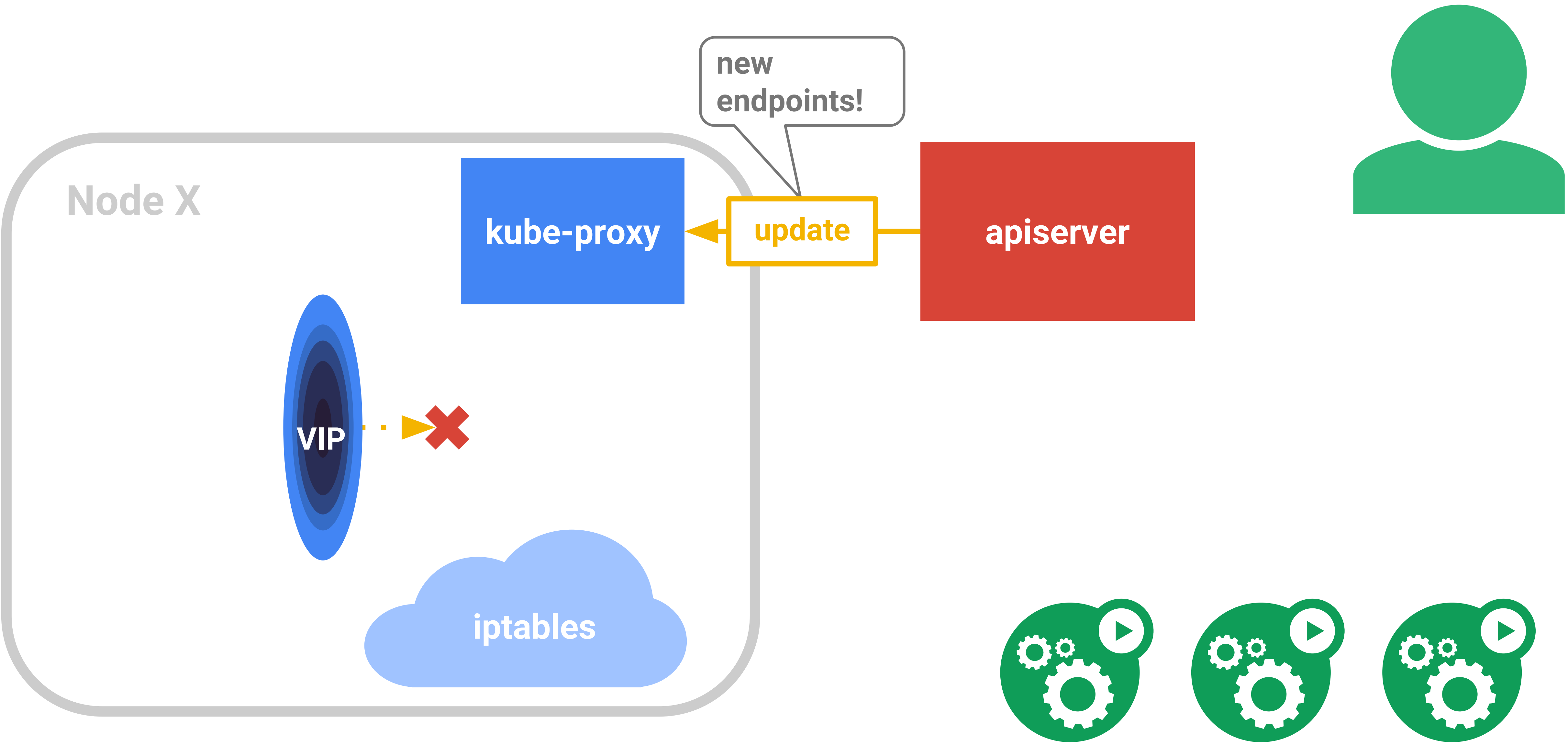
iptables kube-proxy



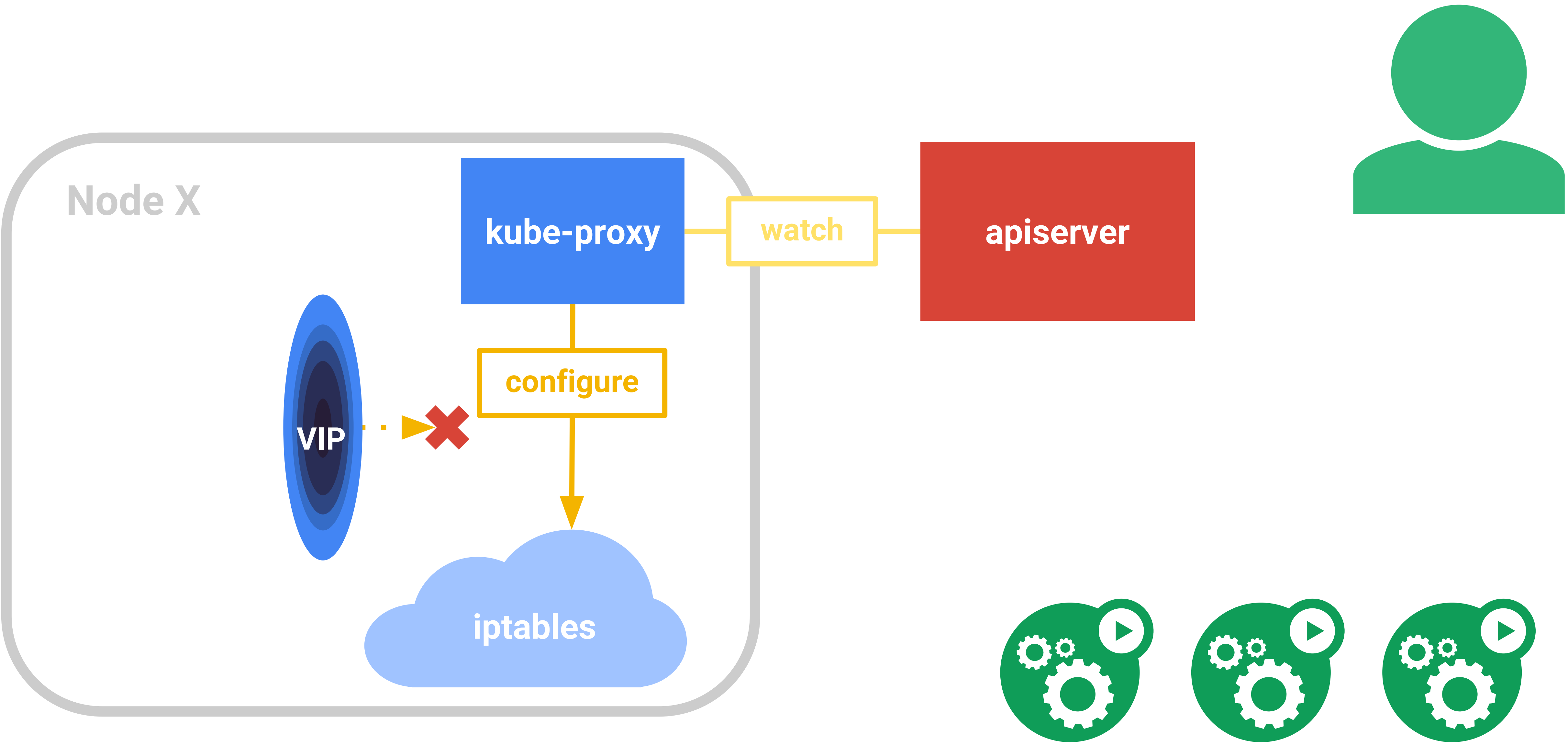
iptables kube-proxy



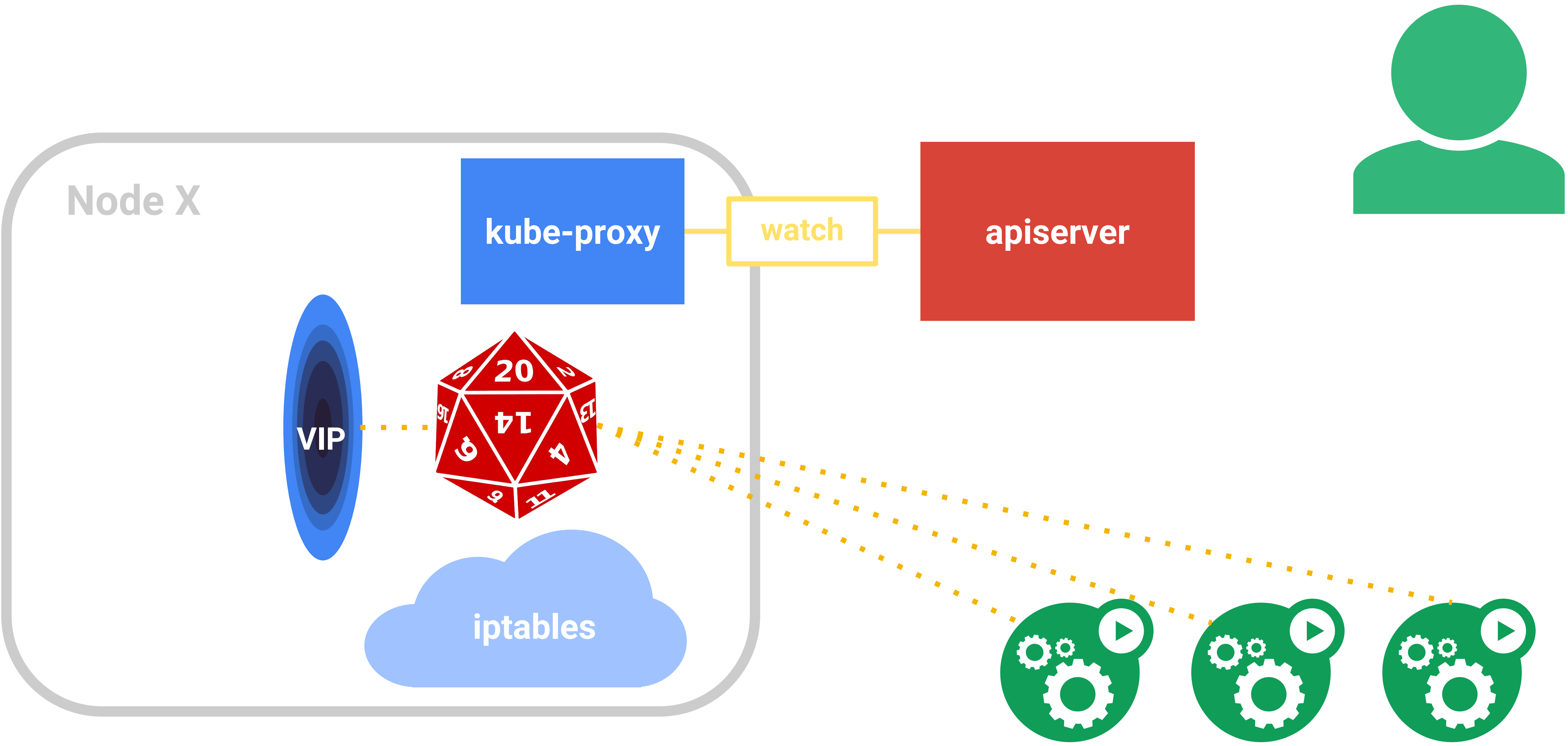
iptables kube-proxy



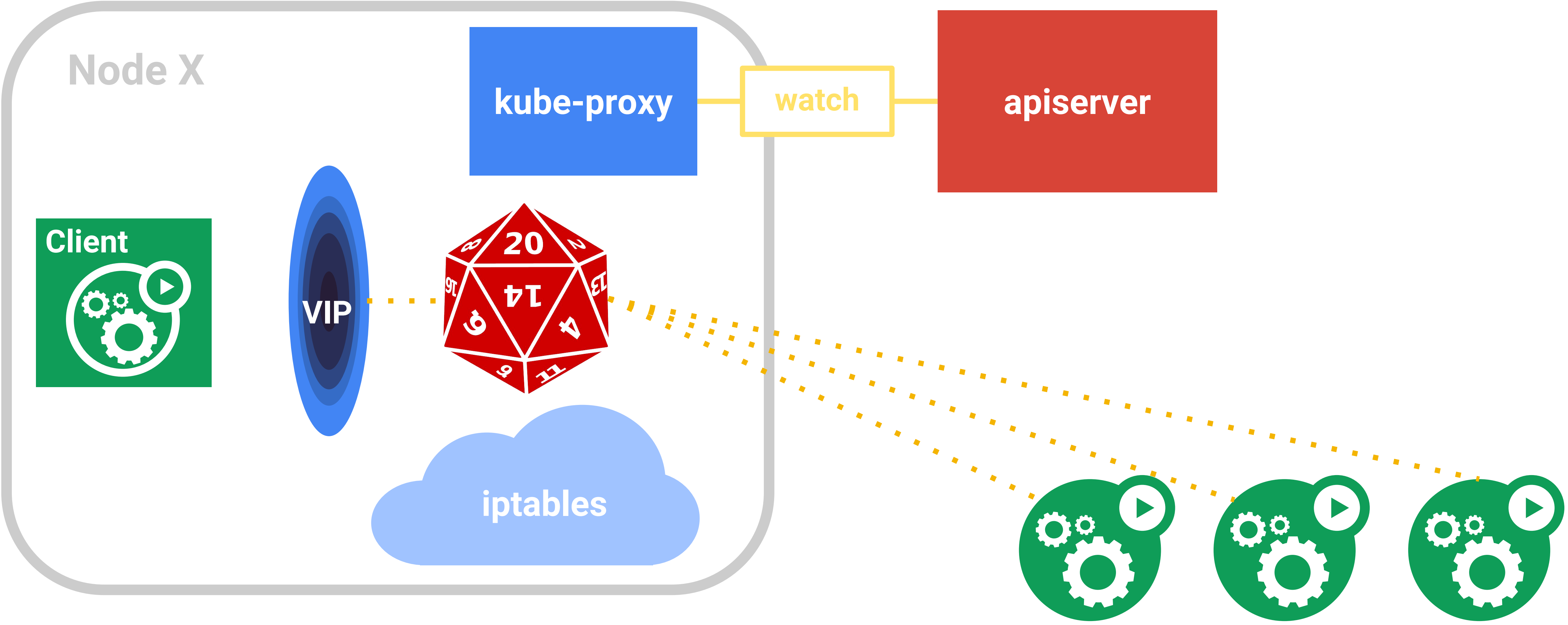
iptables kube-proxy



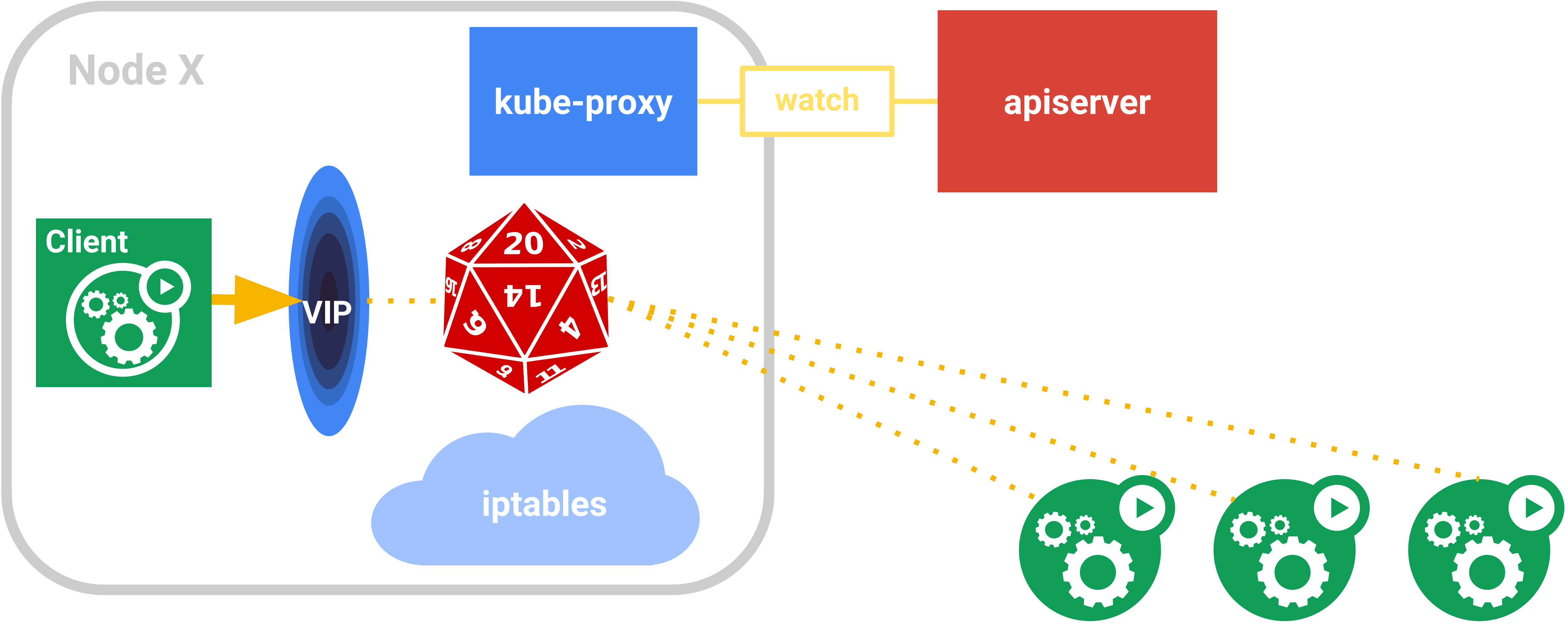
iptables kube-proxy



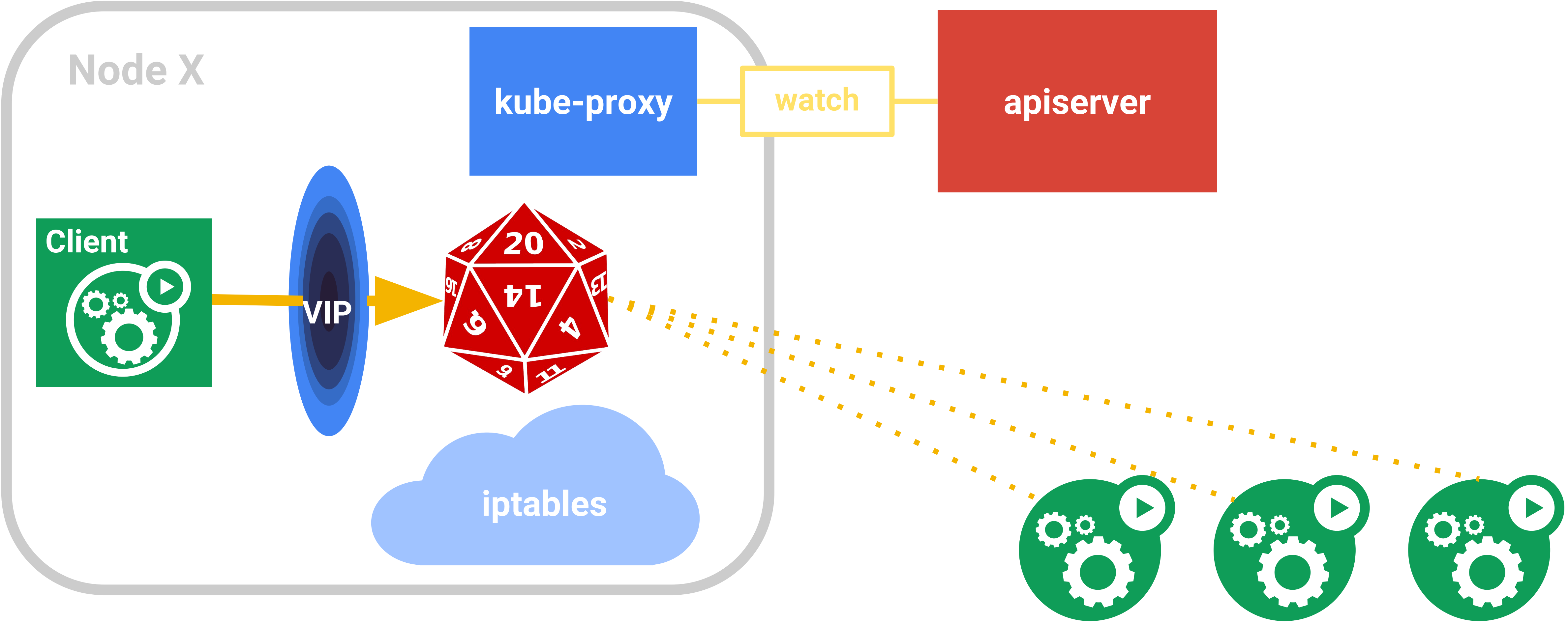
iptables kube-proxy



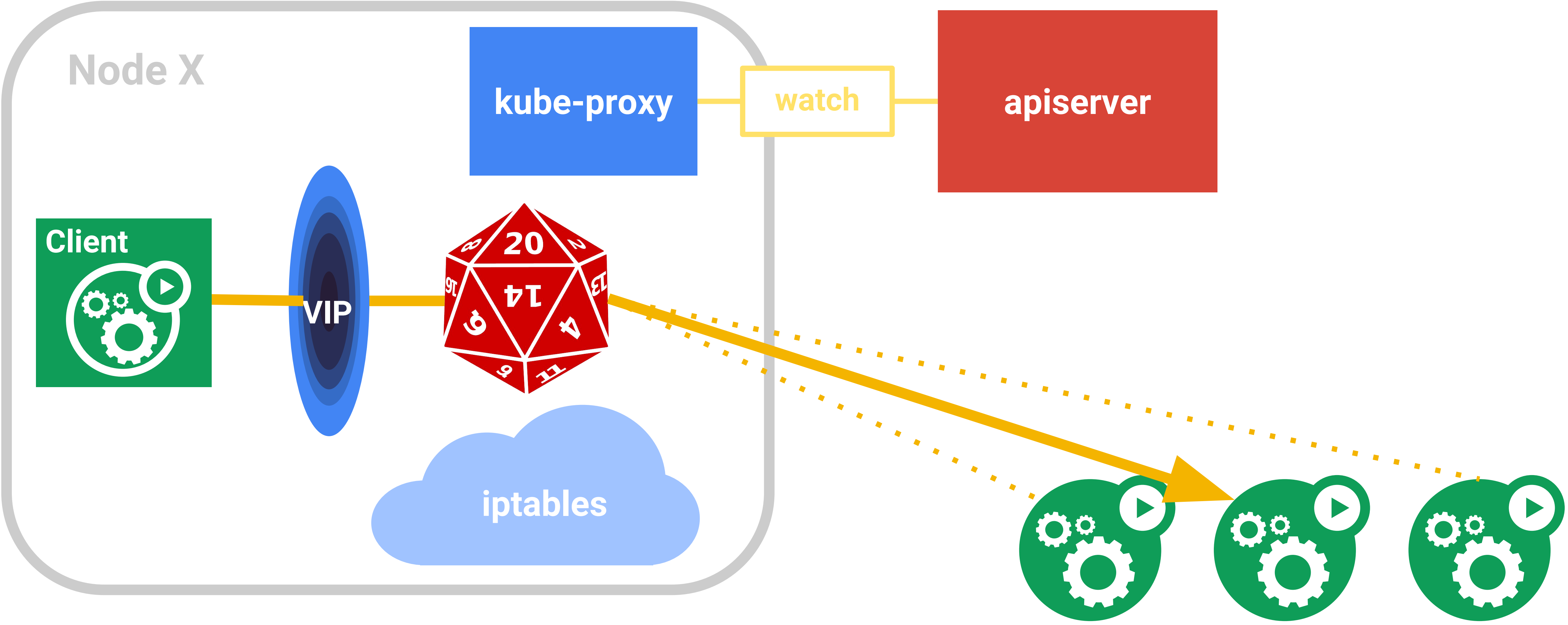
iptables kube-proxy



iptables kube-proxy



iptables kube-proxy



External services

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

Need to receive traffic from “the outside world”

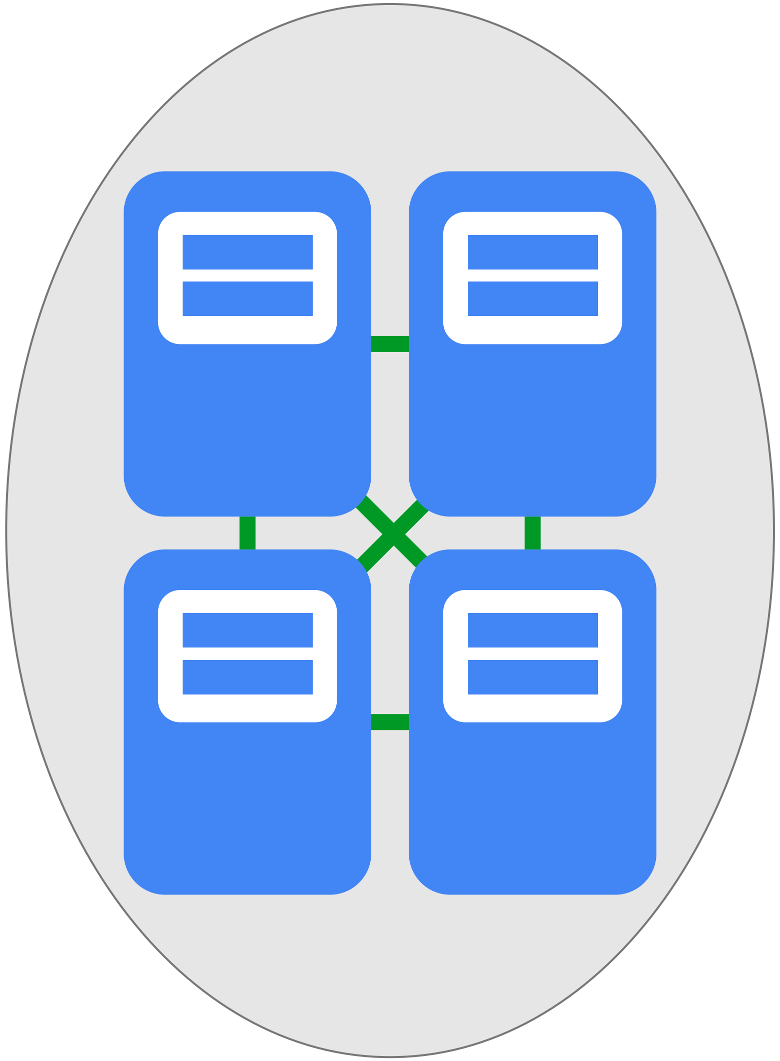
Service “type”

- 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)

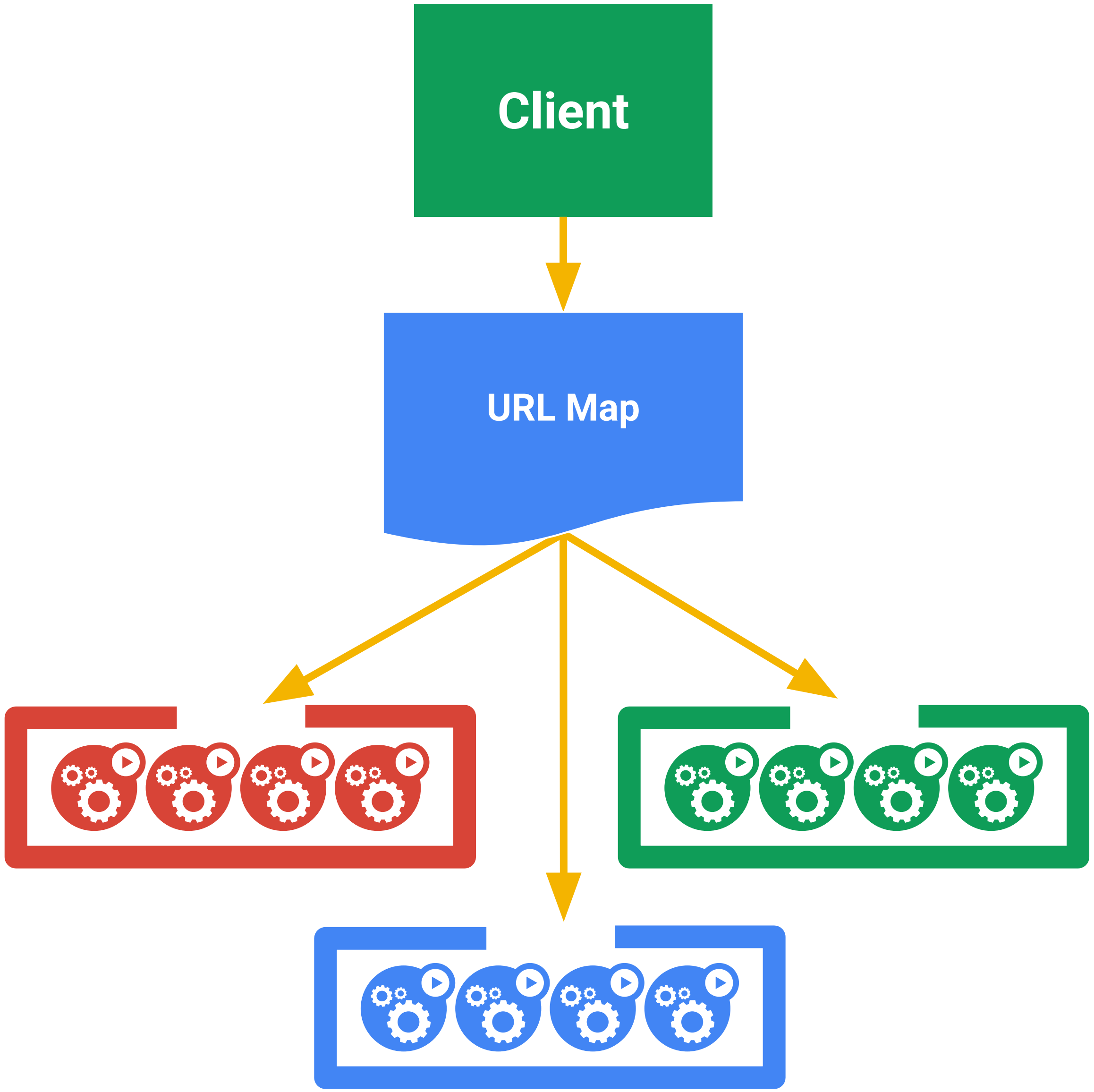
Ingress maps incoming traffic to backend services

- by HTTP host headers
- by HTTP URL paths

HAProxy, NGINX, AWS and GCE implementations in progress

Now with SSL!

Status: BETA in Kubernetes v1.2



Rolling Update

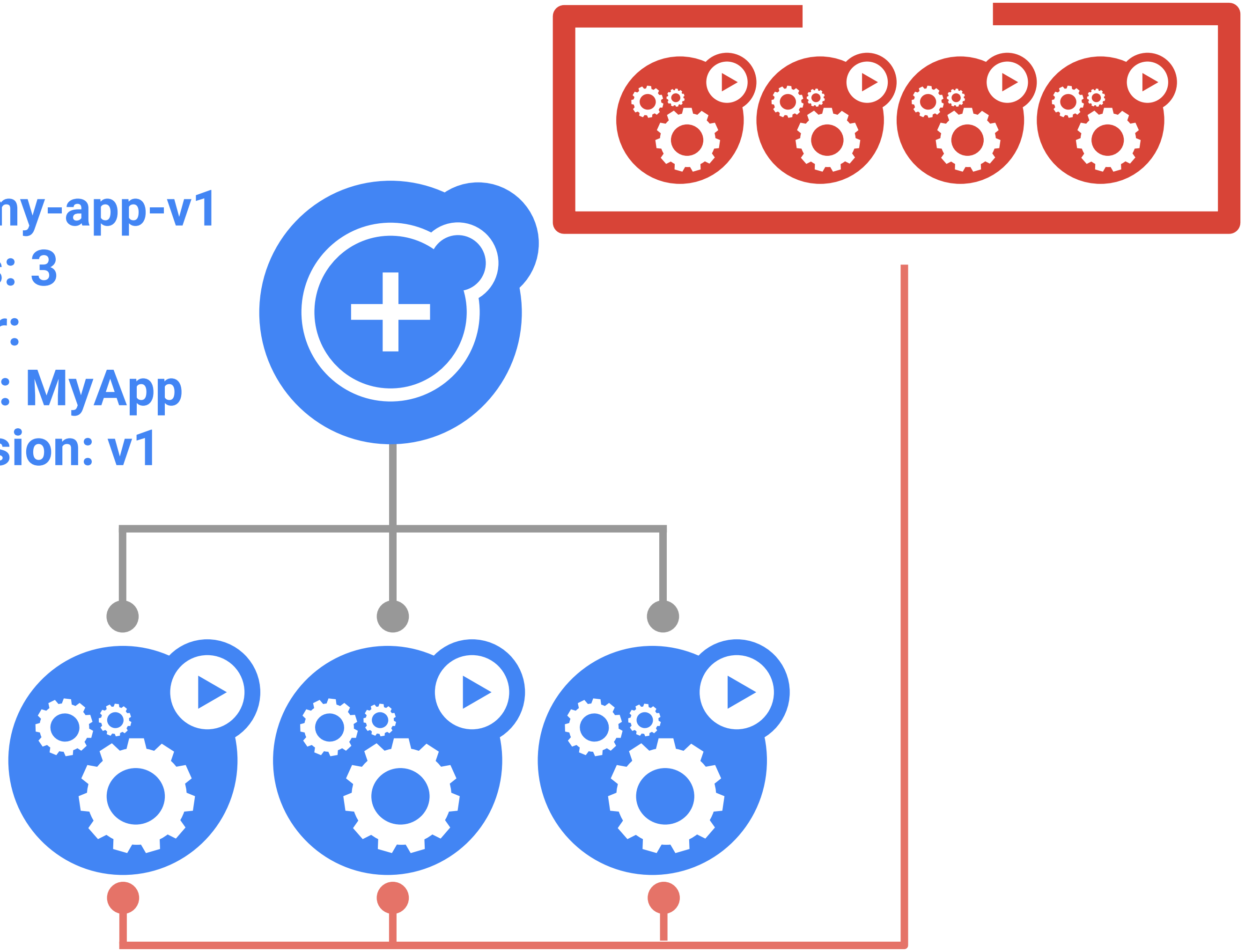
Rolling Update

Service

- app: MyApp

ReplicaSet

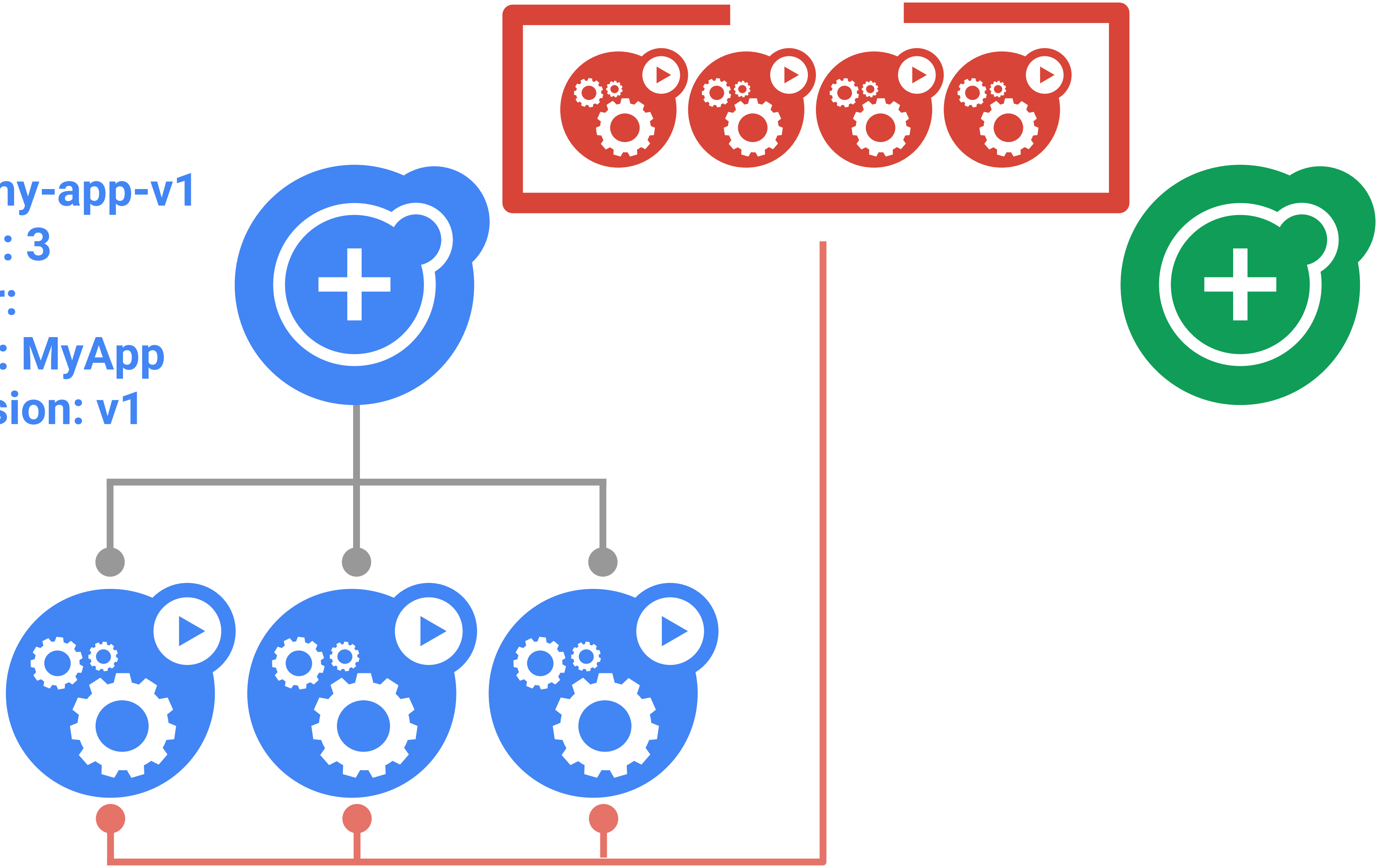
- name: my-app-v1
- replicas: 3
- selector:
    - app: MyApp
    - version: v1



Rolling Update

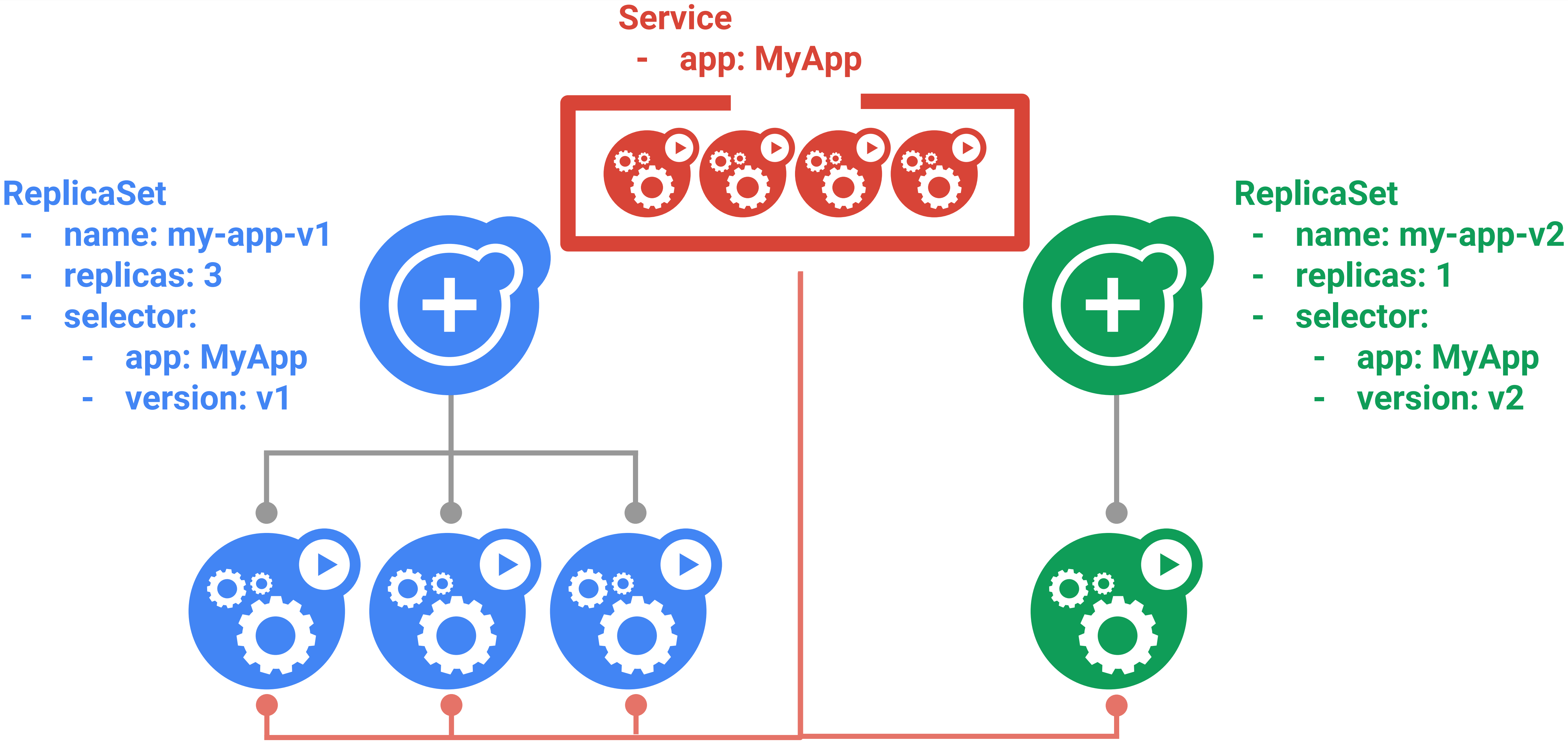
Service
- app: MyApp

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

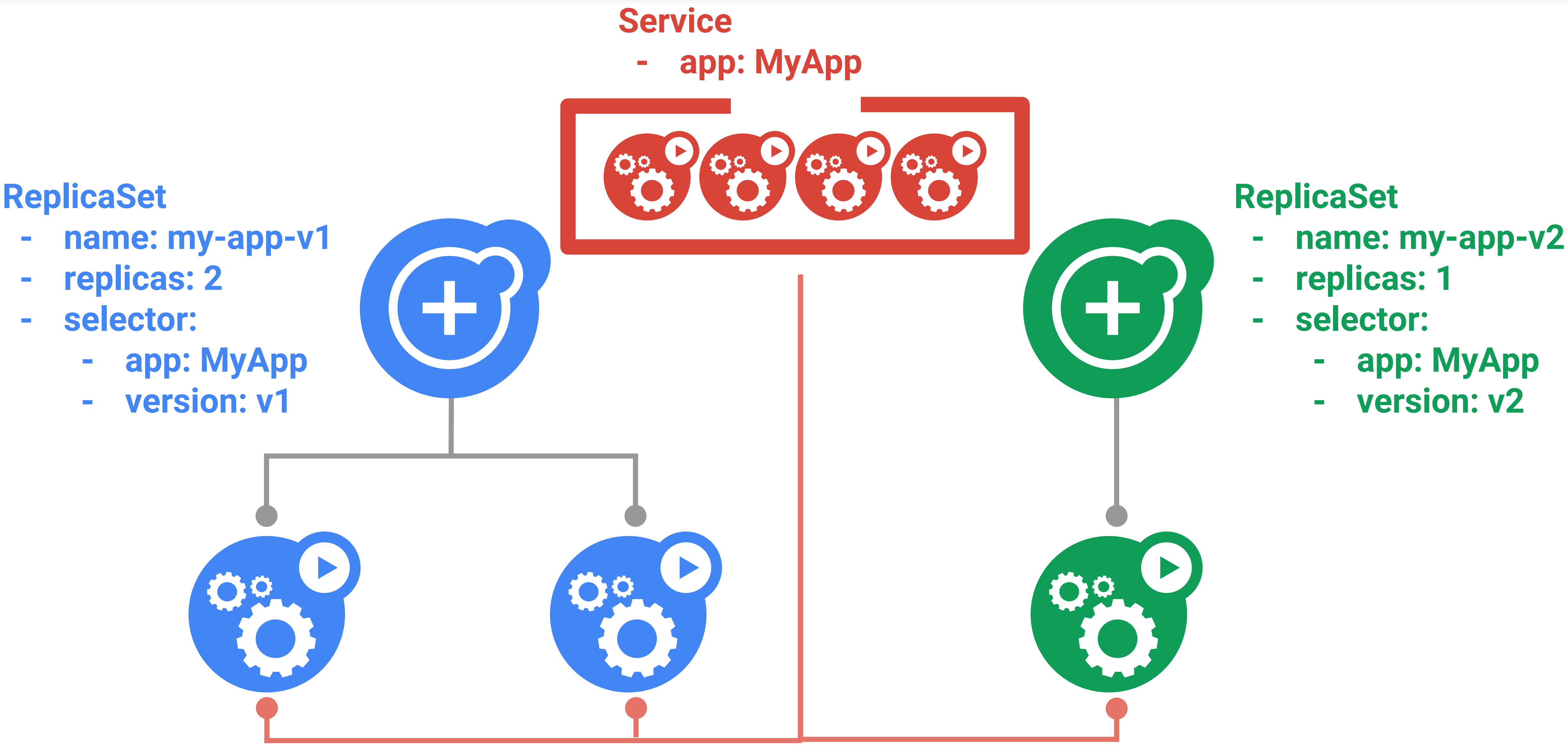


ReplicaSet
- name: my-app-v2
- replicas: 0
- selector:
    - app: MyApp
    - version: v2

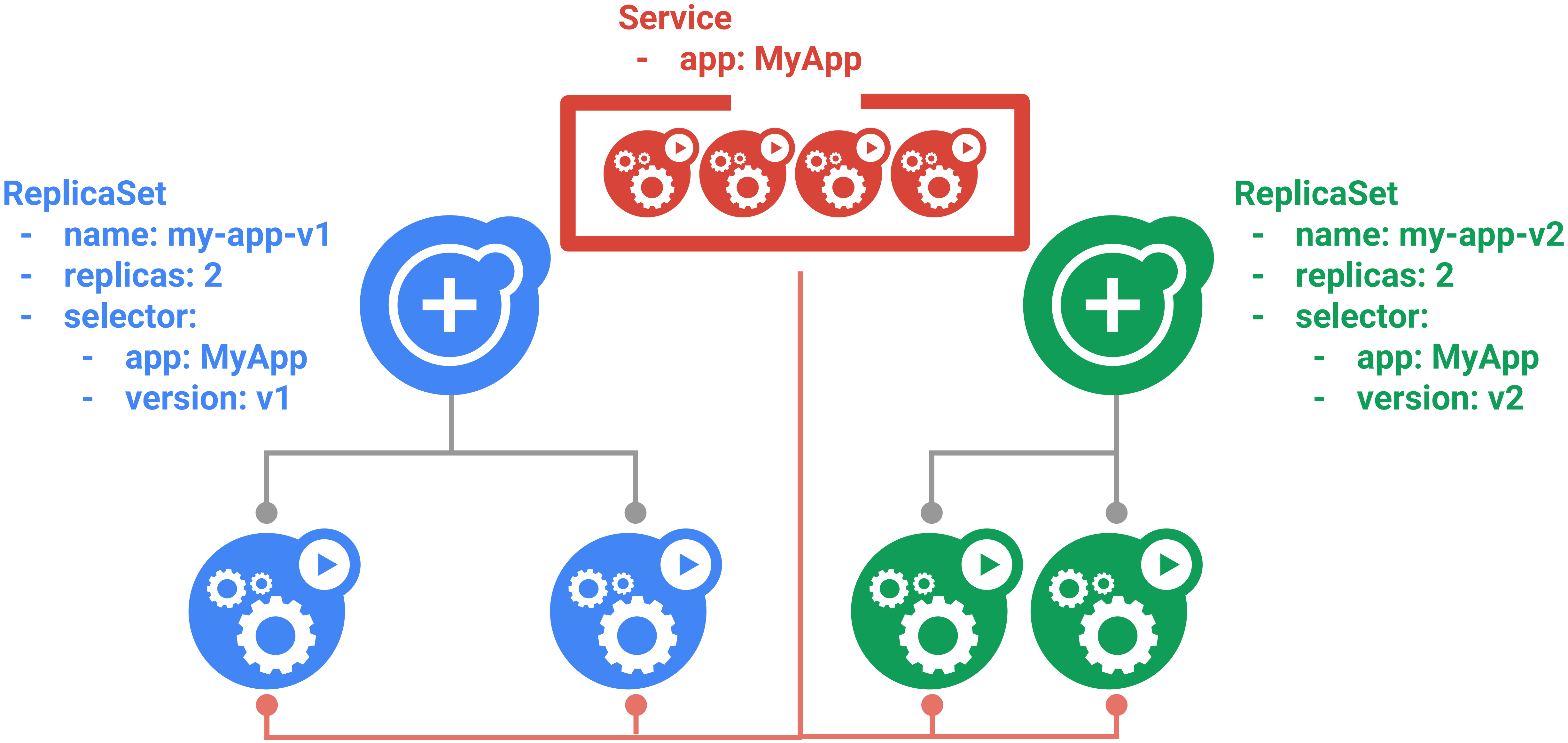
Rolling Update



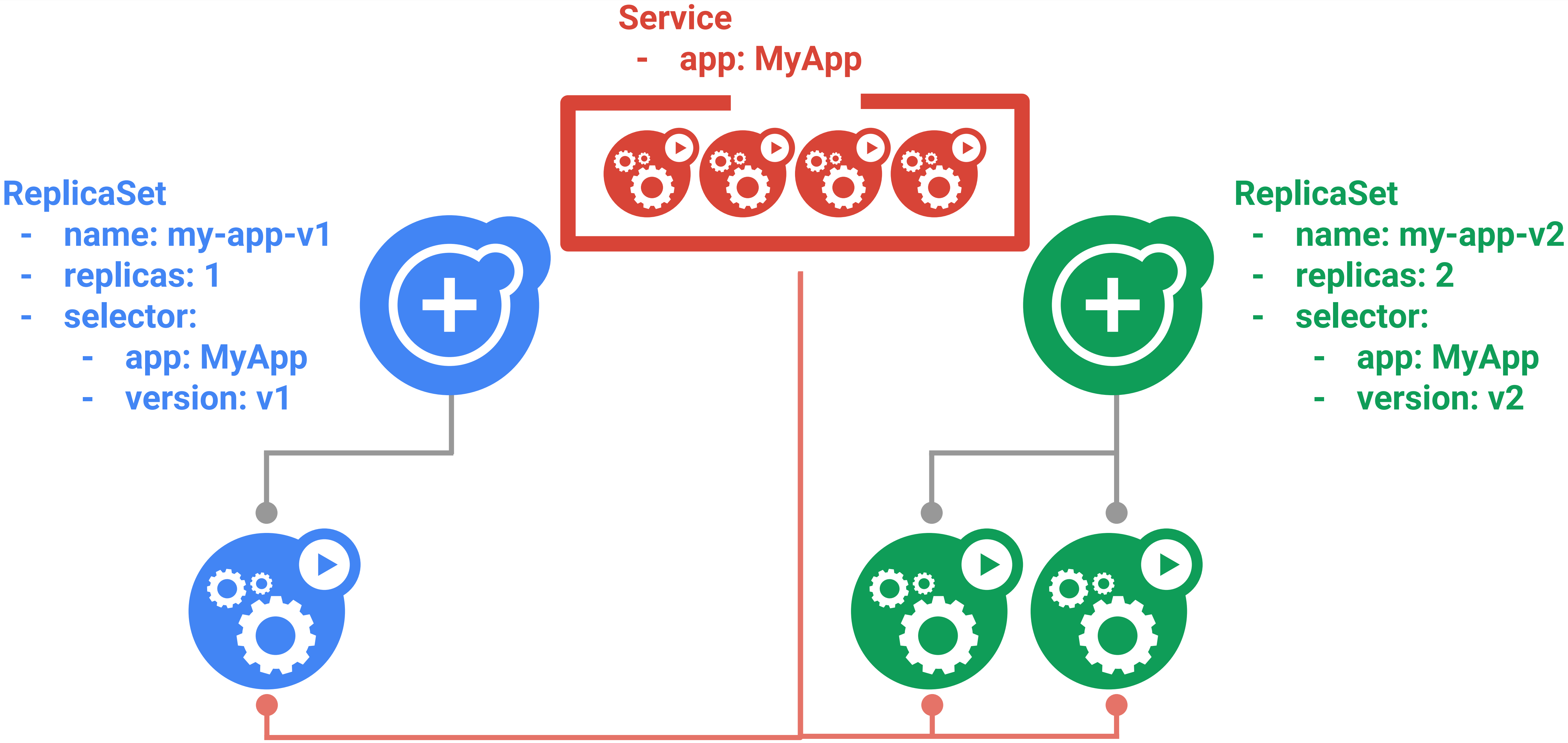
Rolling Update



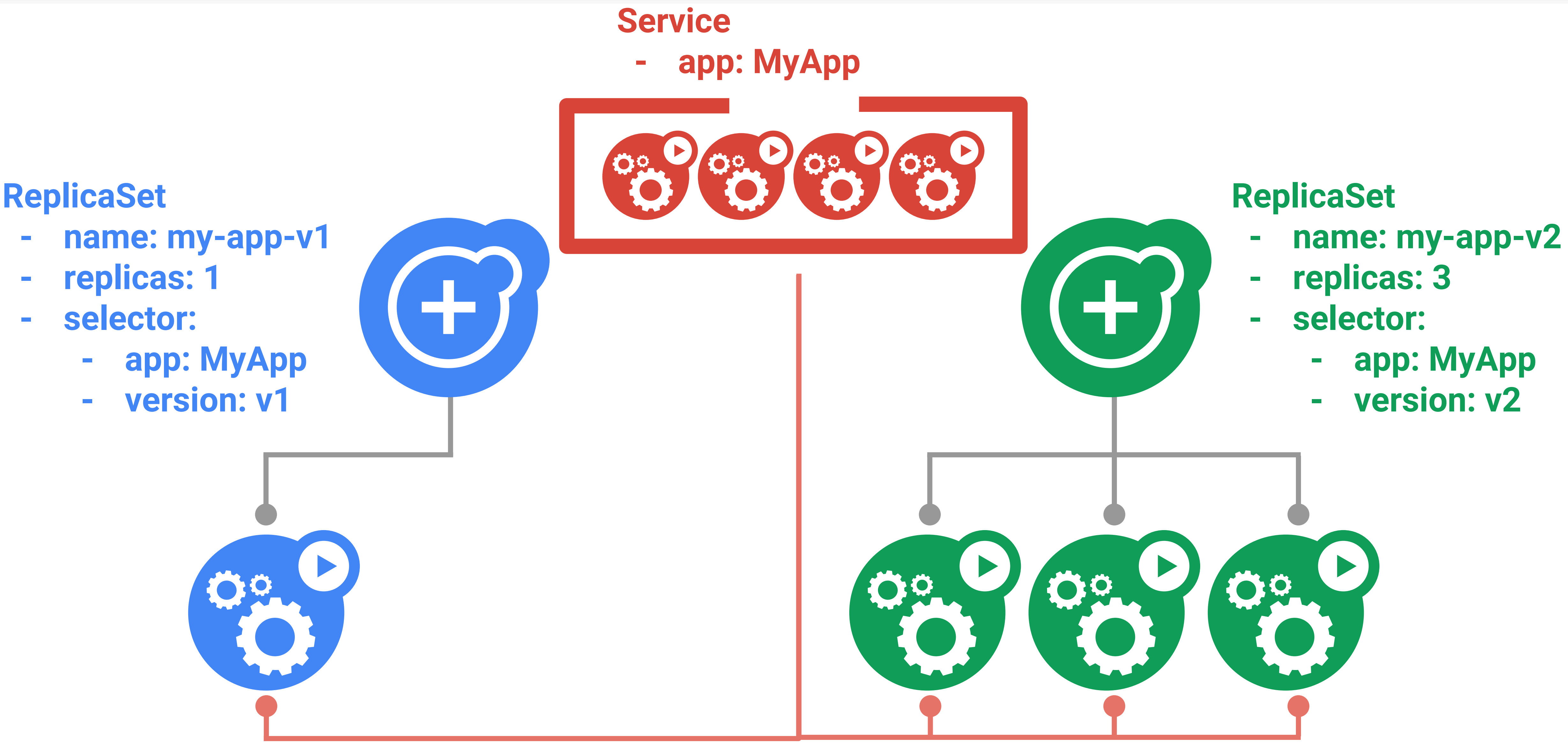
Rolling Update



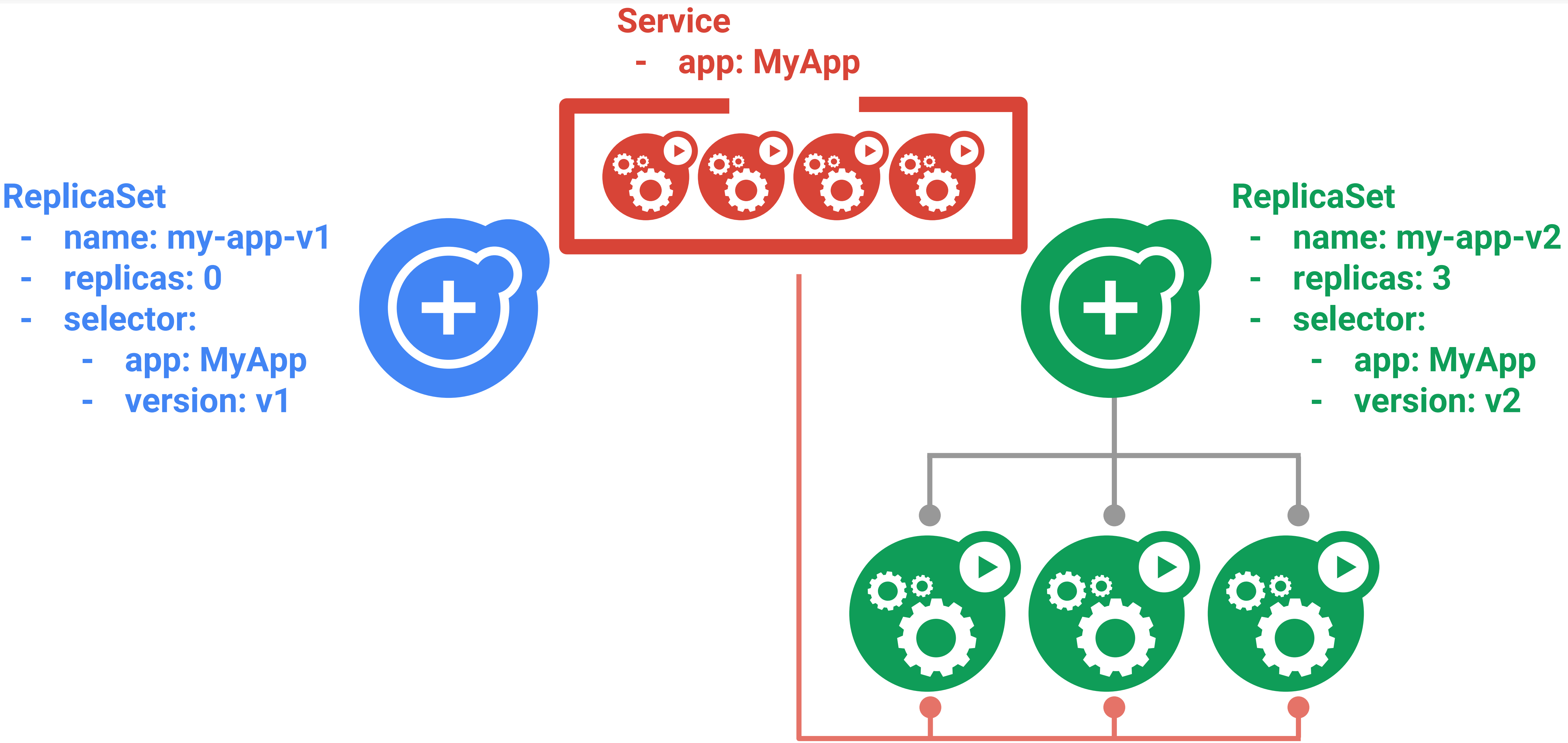
Rolling Update



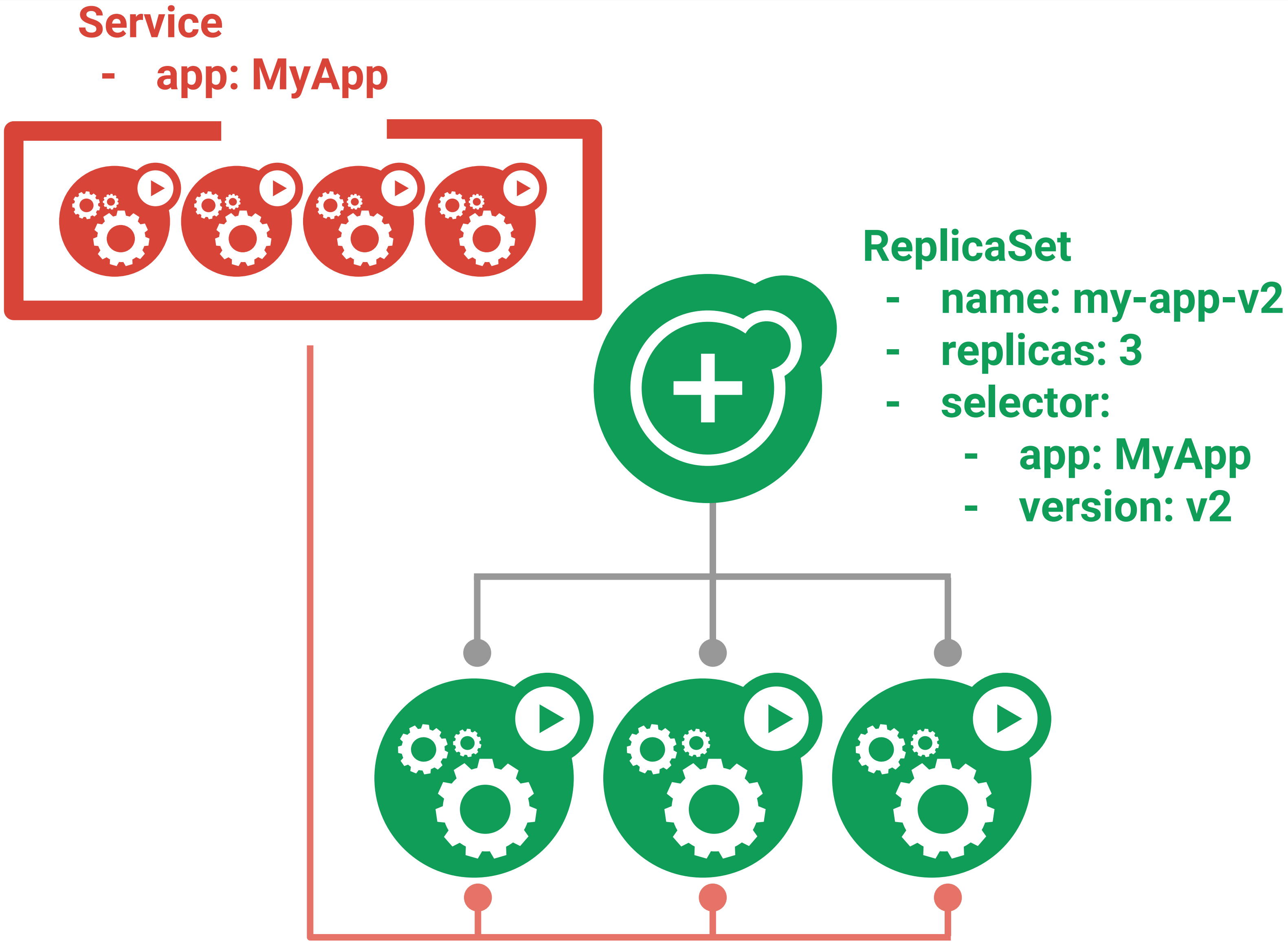
Rolling Update



Rolling Update



Rolling Update



Deployments

Deployments

Updates-as-a-service

- Rolling update is imperative, client-side

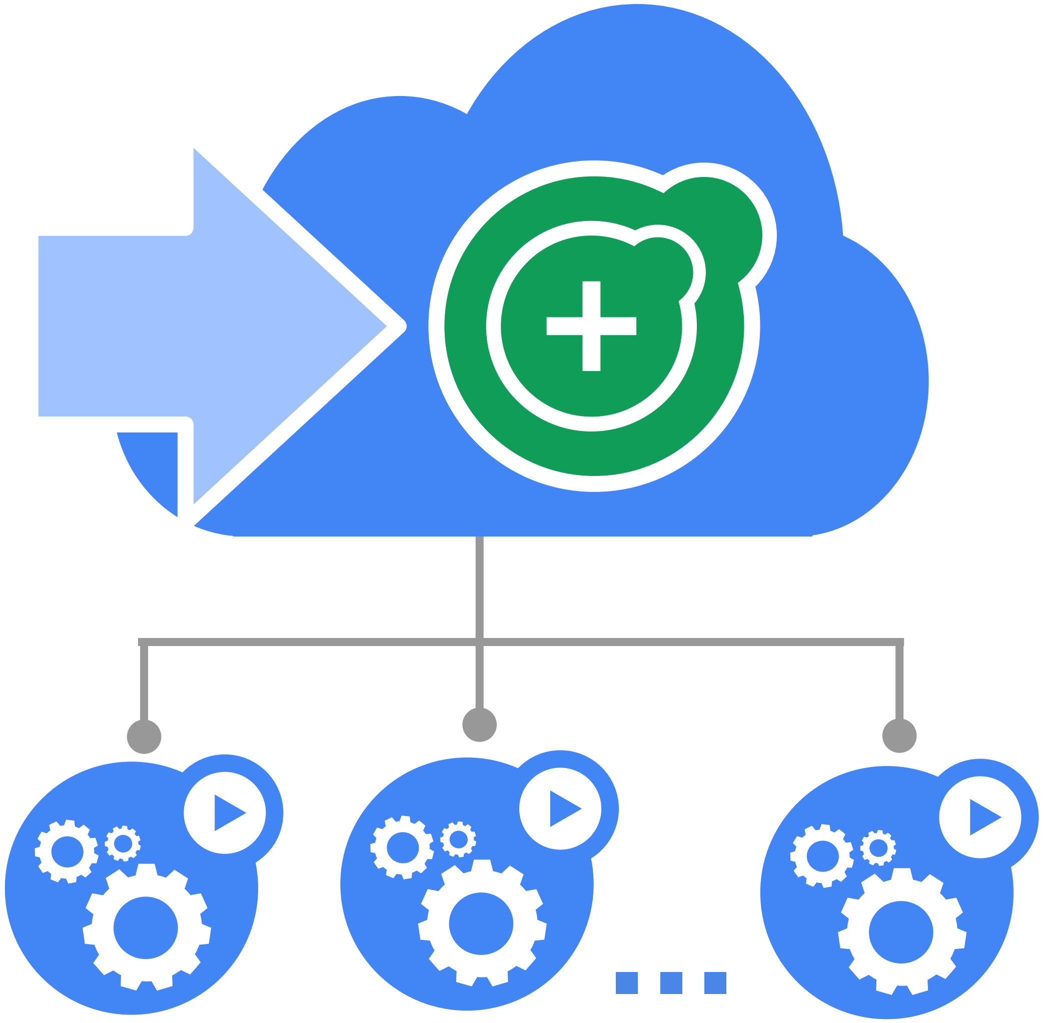
Deployment manages replica changes for you

- stable object name
- updates are configurable, done server-side
- `kubectl edit` or `kubectl apply`

Aggregates stats

Can have multiple updates in flight

Status: BETA in Kubernetes v1.2



DaemonSets

DaemonSets

Problem: how to run a Pod on every node?

- or a subset of nodes

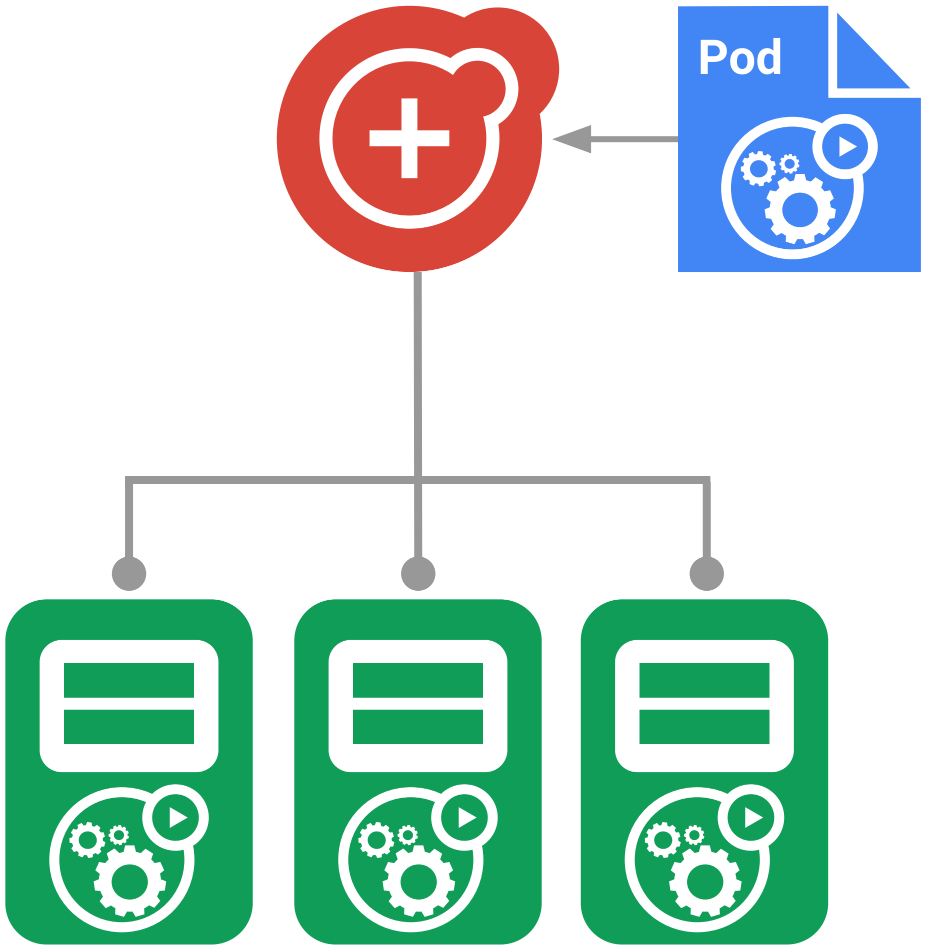
Similar to ReplicaSet

- principle: do one thing, don't overload

"Which nodes?" is a selector

Use familiar tools and patterns

Status: BETA in Kubernetes v1.2



Jobs

Jobs

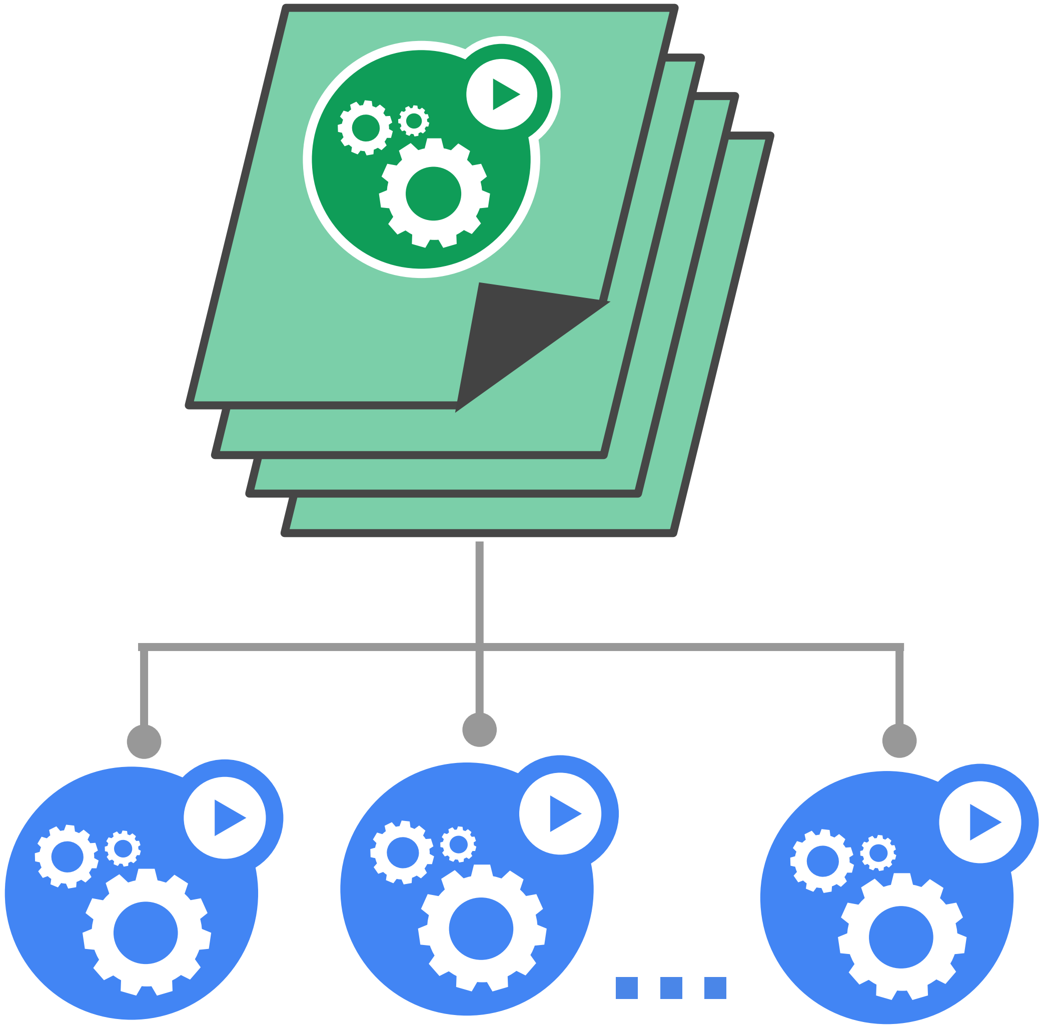
Run-to-completion, as opposed to run-forever

- Express parallelism vs. required completions
- Workflow: restart on failure
- Build/test: don’t restart on failure

Aggregates success/failure counts

Built for batch and big-data work

Status: GA in Kubernetes v1.2



PersistentVolumes

PersistentVolumes

A higher-level storage abstraction

- insulation from any one cloud environment

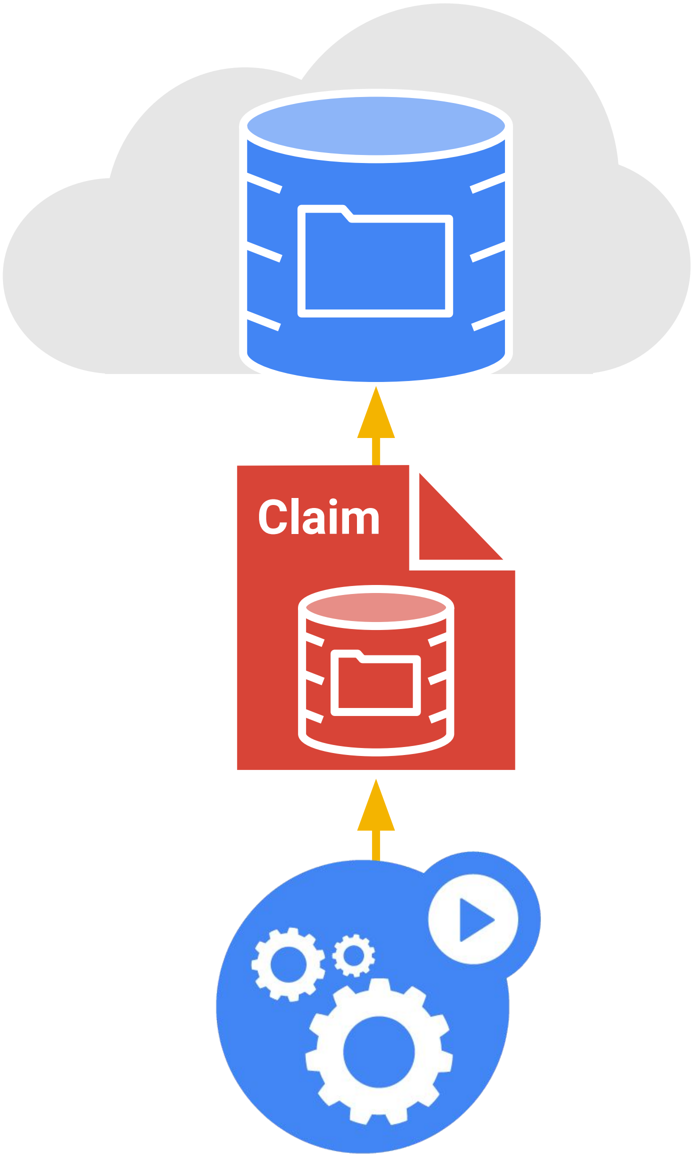
Admin provisions them, users claim them

- **NEW: auto-provisioning (alpha in v1.2)**

Independent lifetime from consumers

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

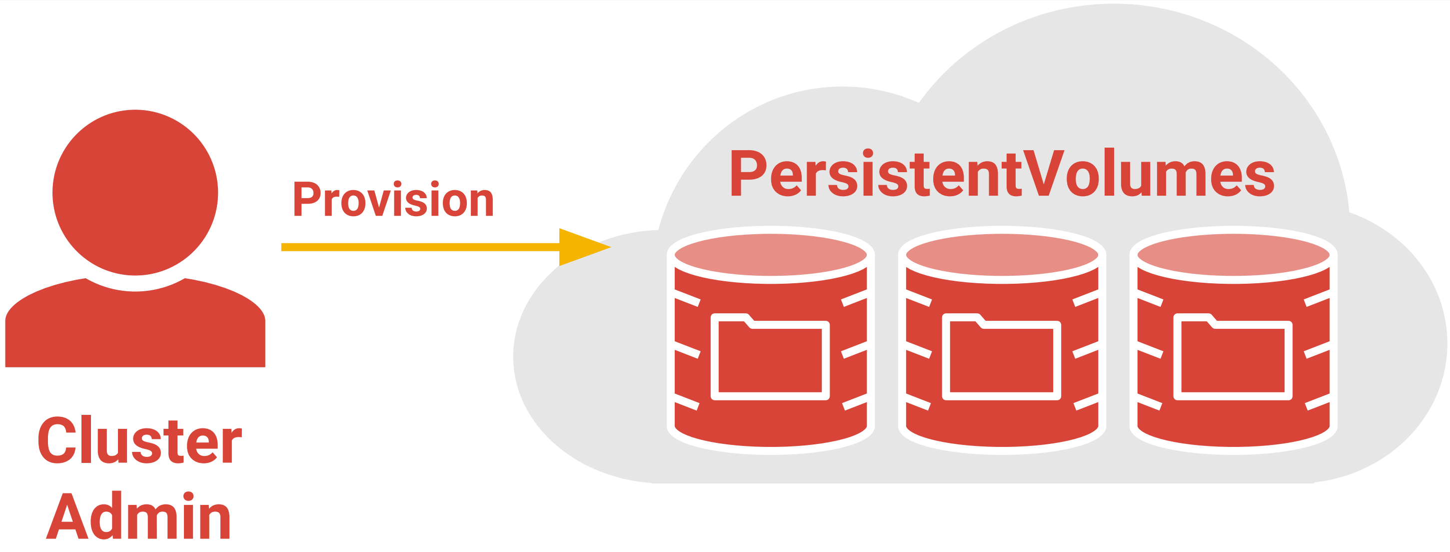
Dynamically “scheduled” and managed, like nodes and pods



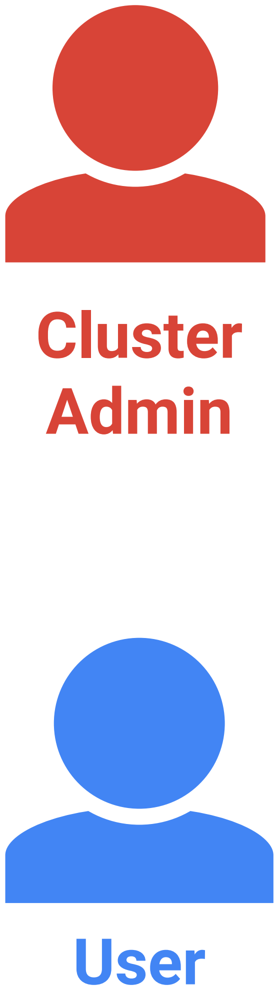
PersistentVolumes

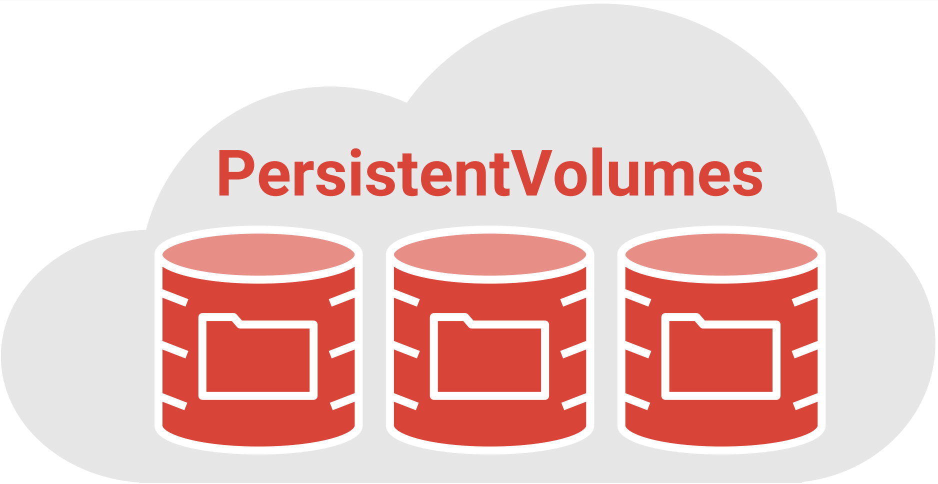
Cluster Admin

PersistentVolumes

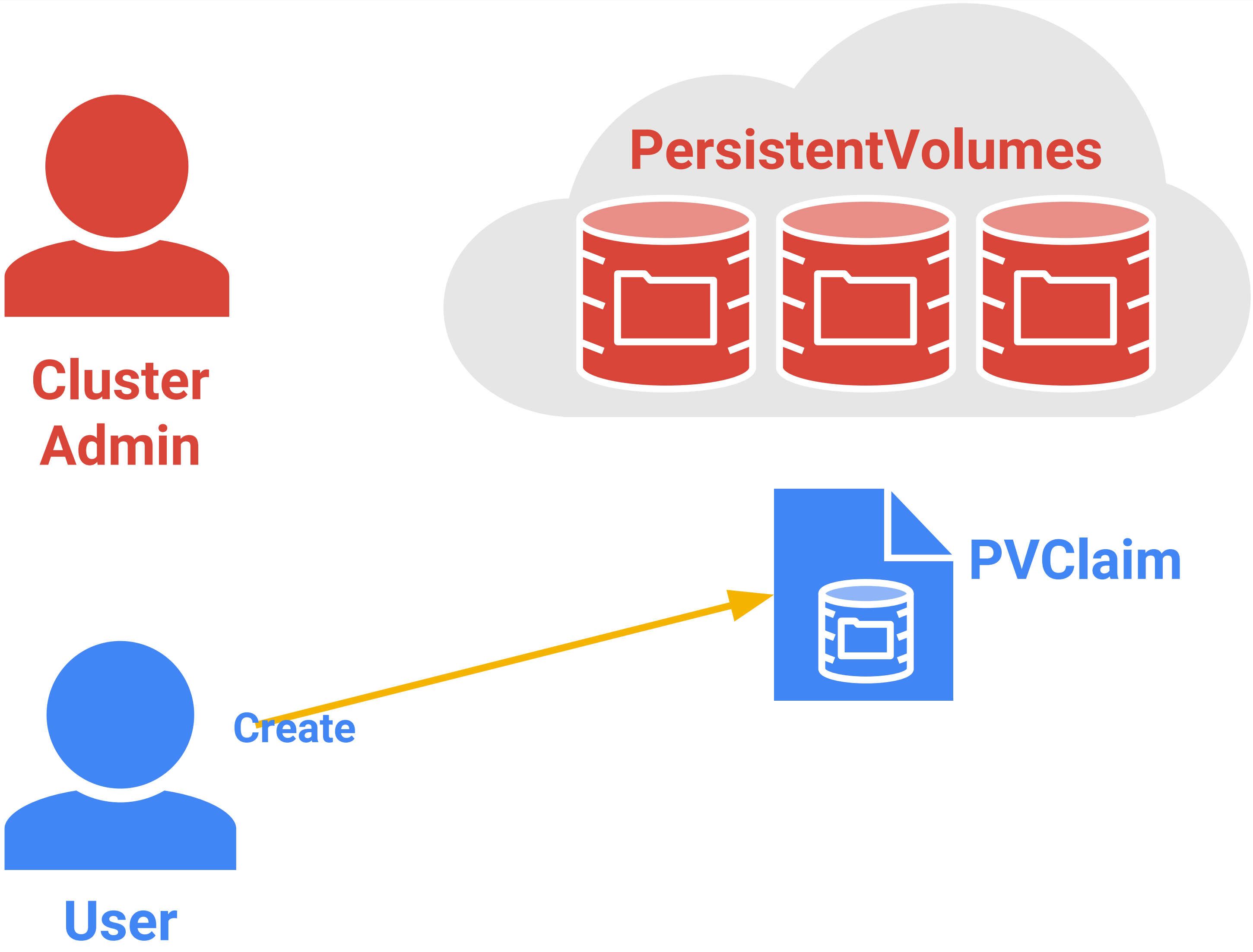


PersistentVolumes

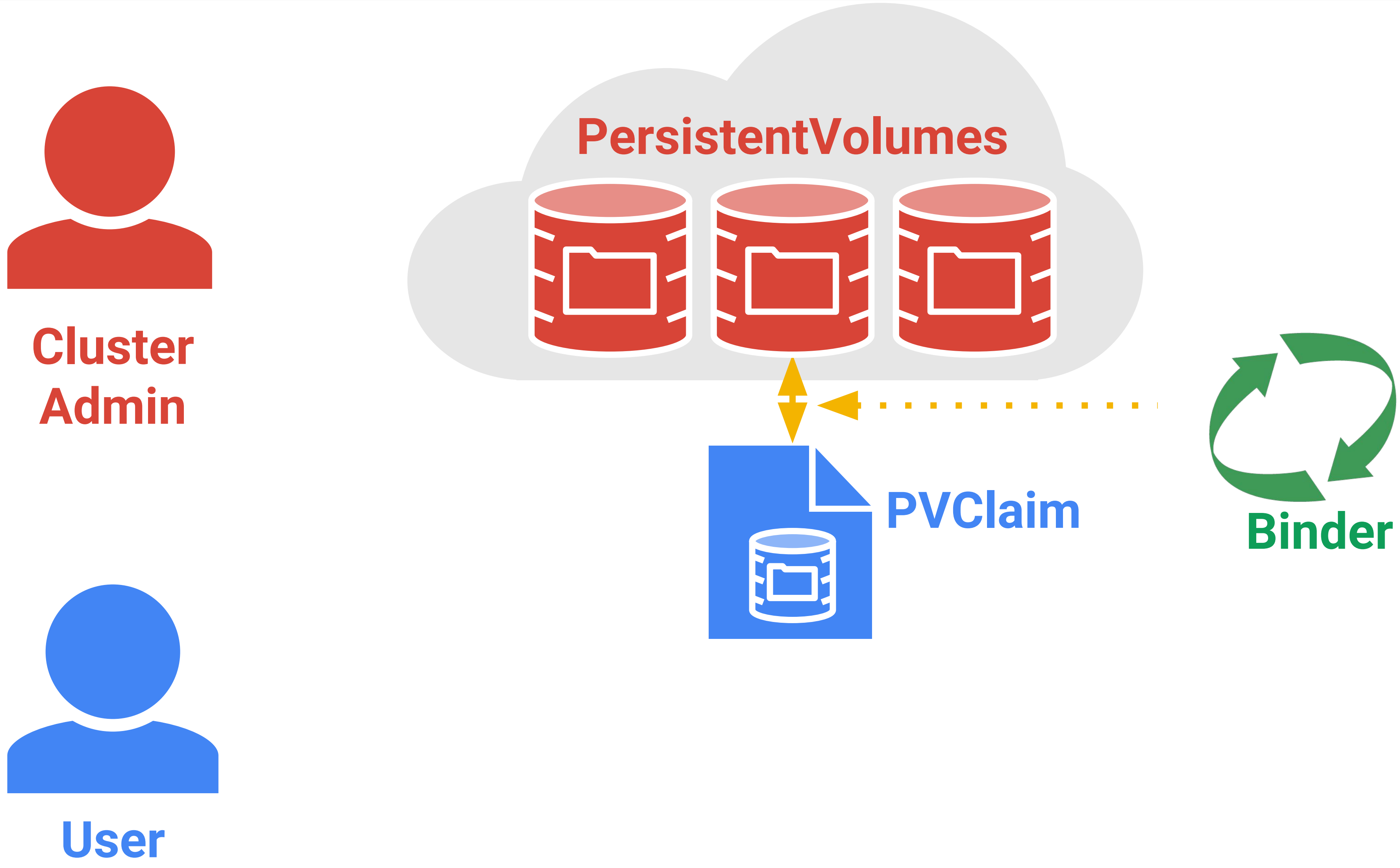




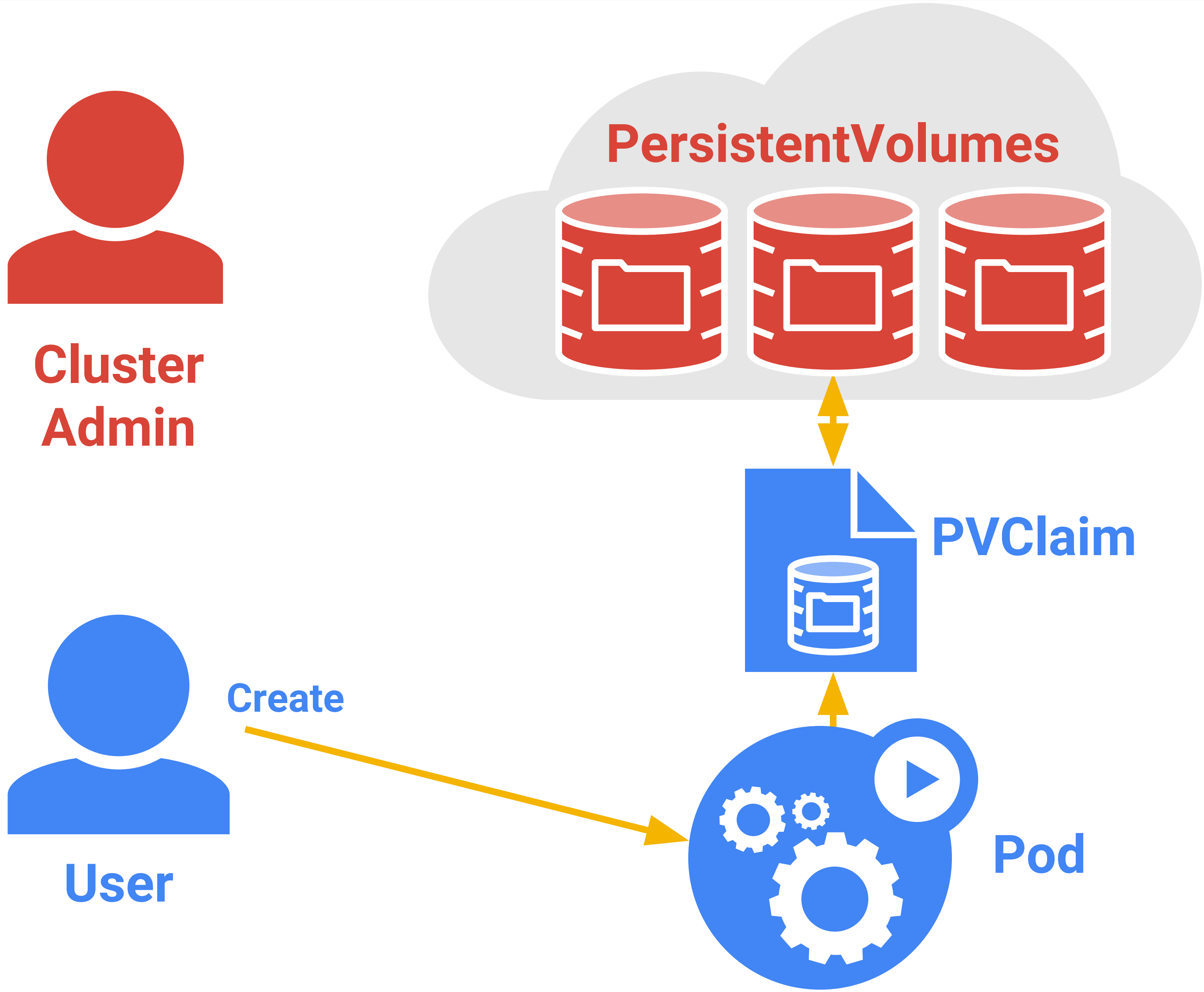
PersistentVolumes



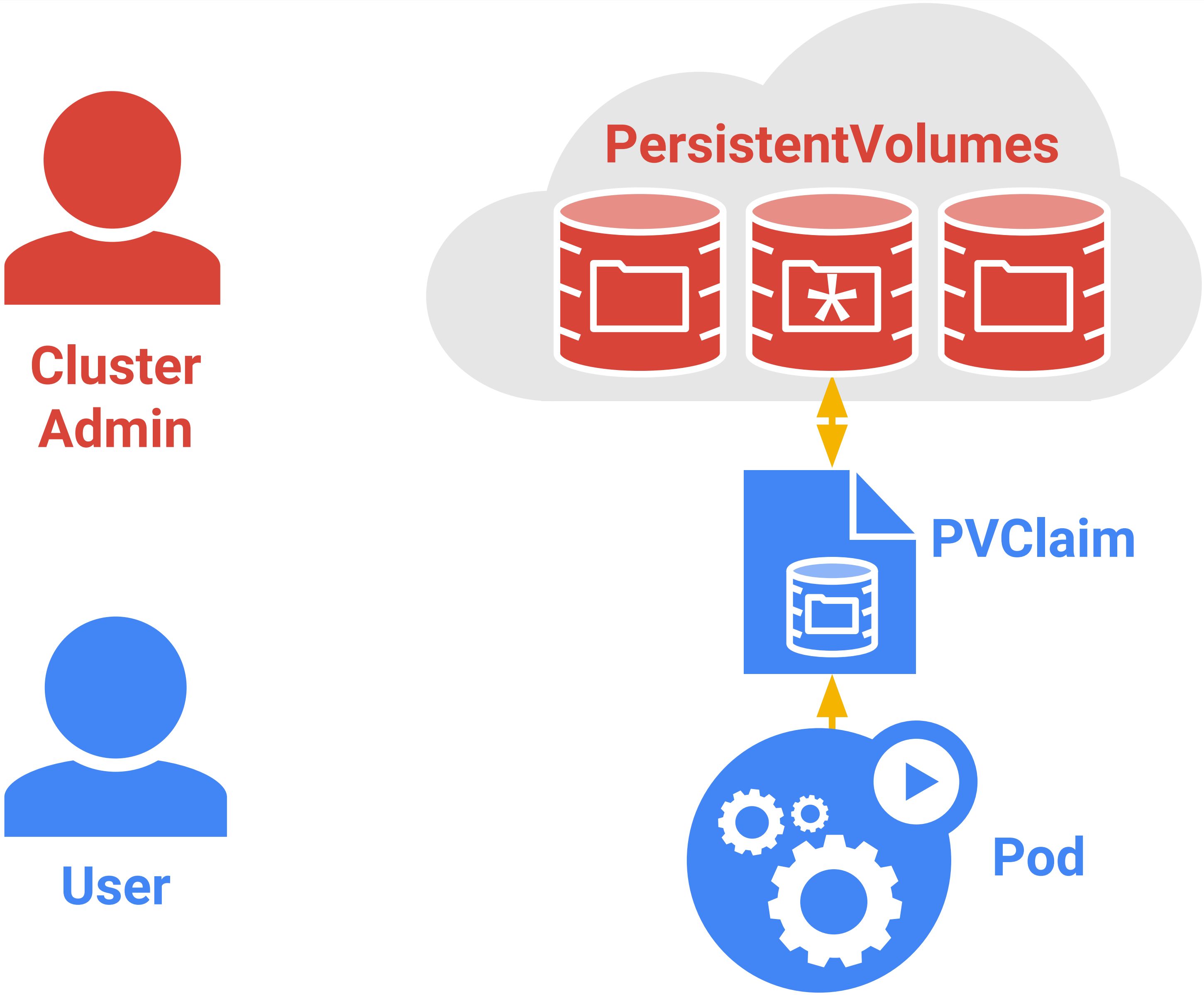
PersistentVolumes



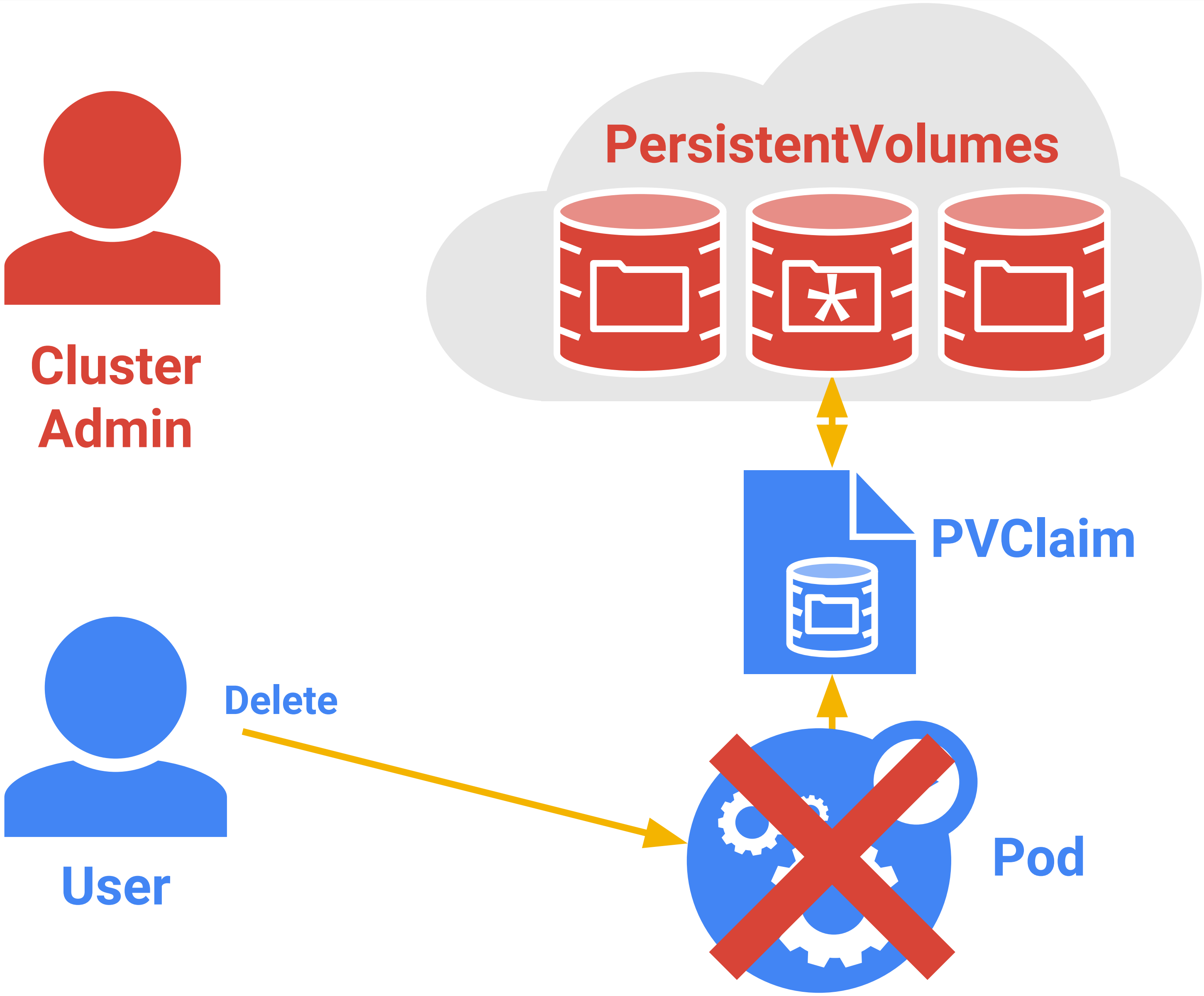
PersistentVolumes



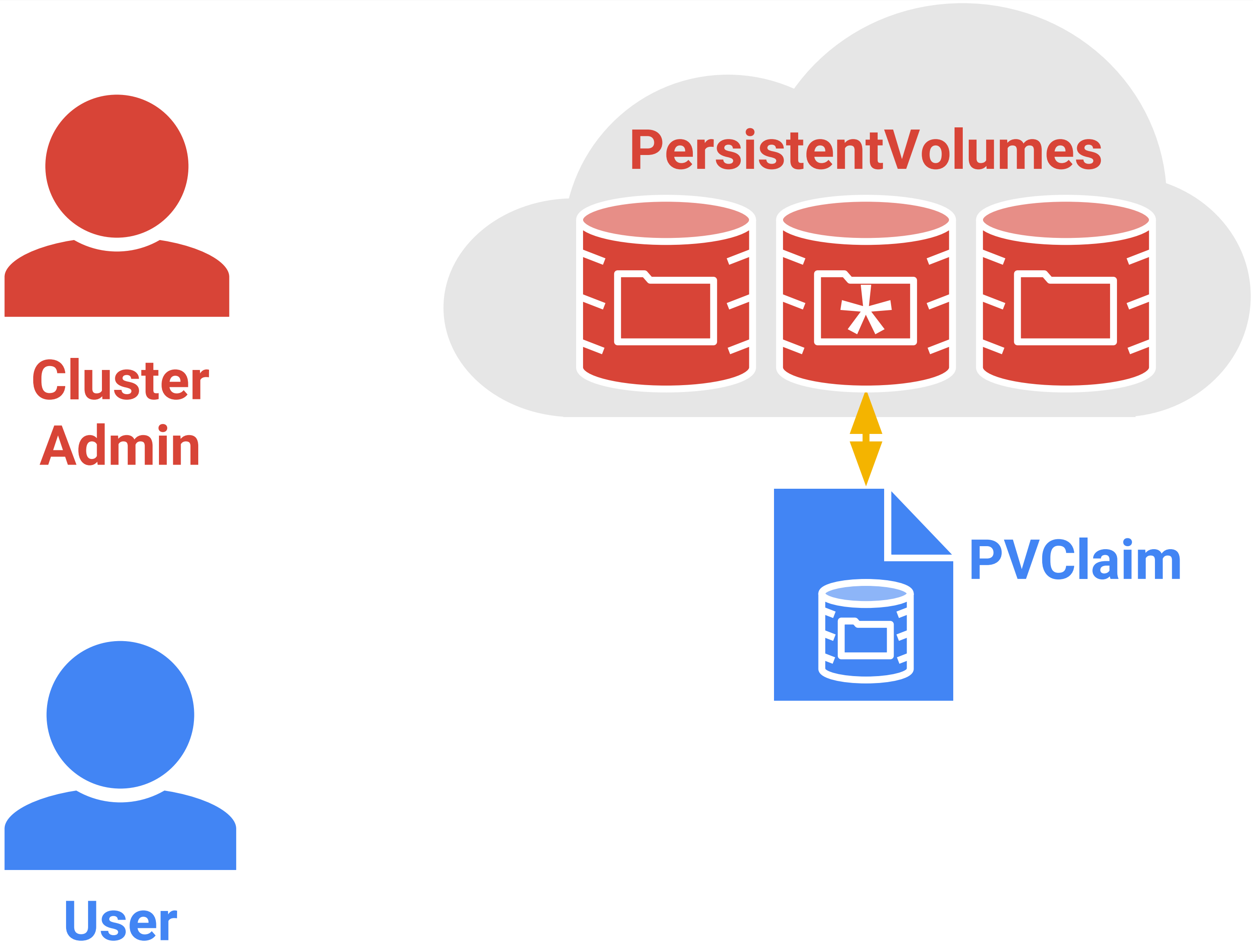
PersistentVolumes



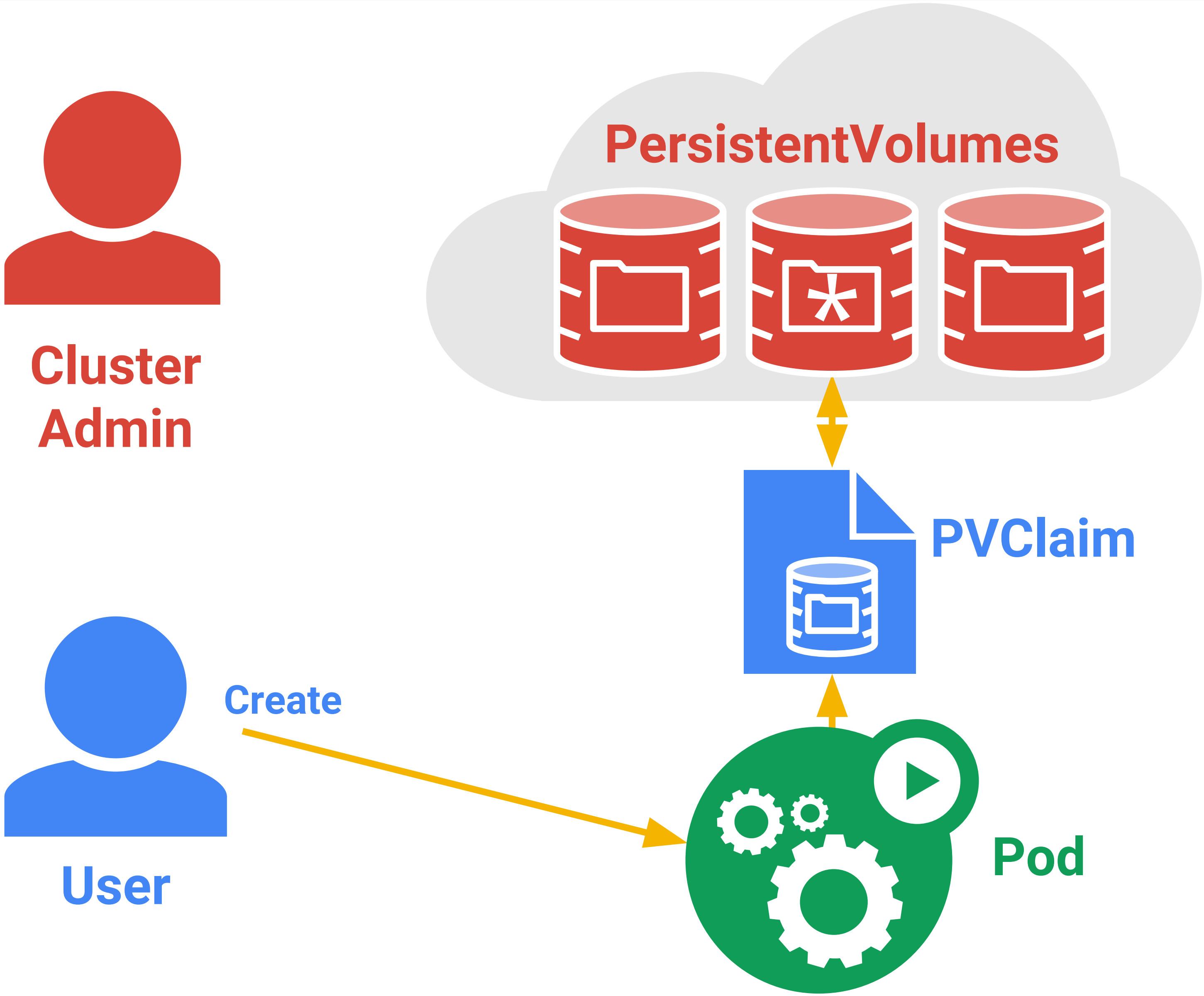
PersistentVolumes



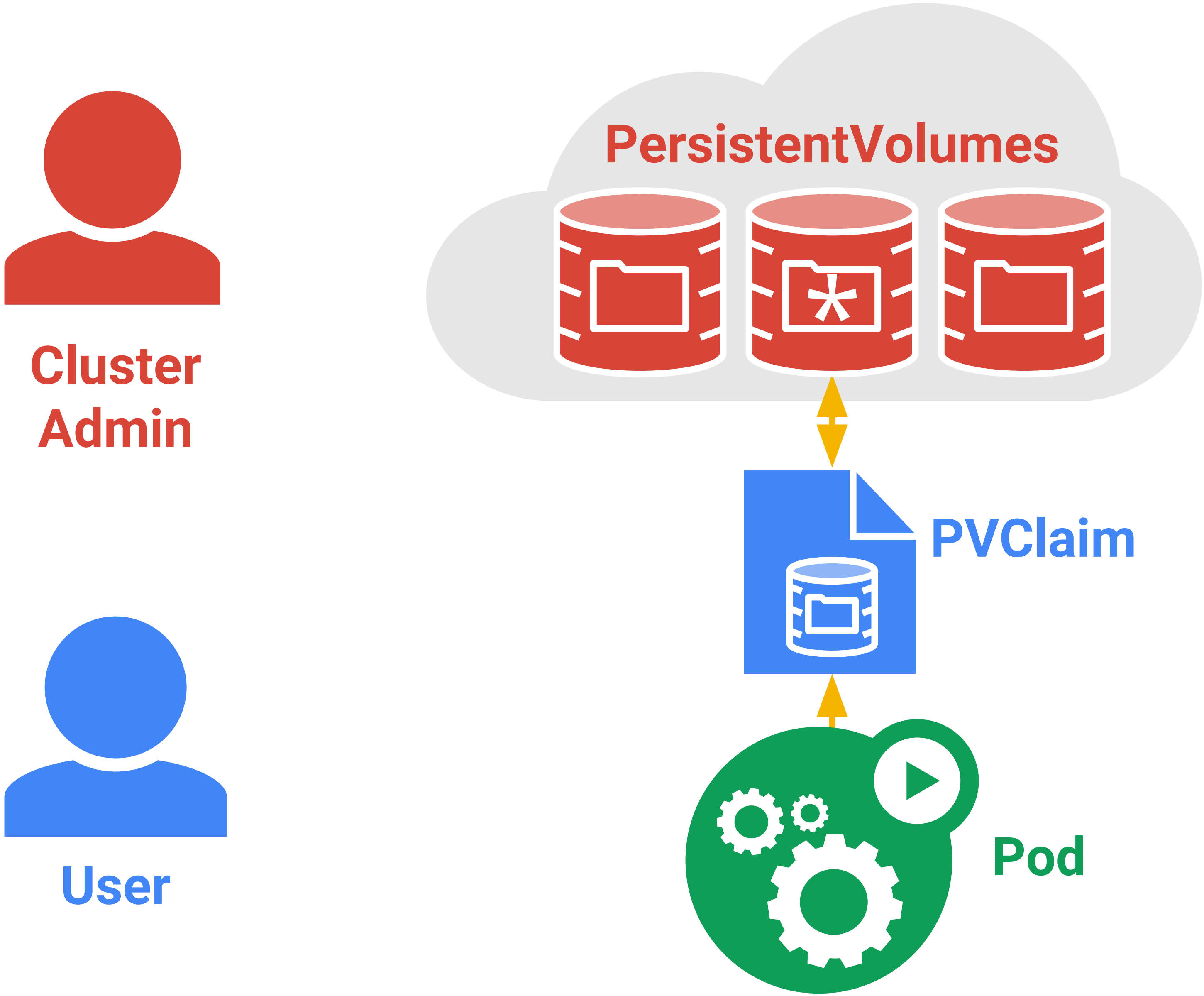
PersistentVolumes



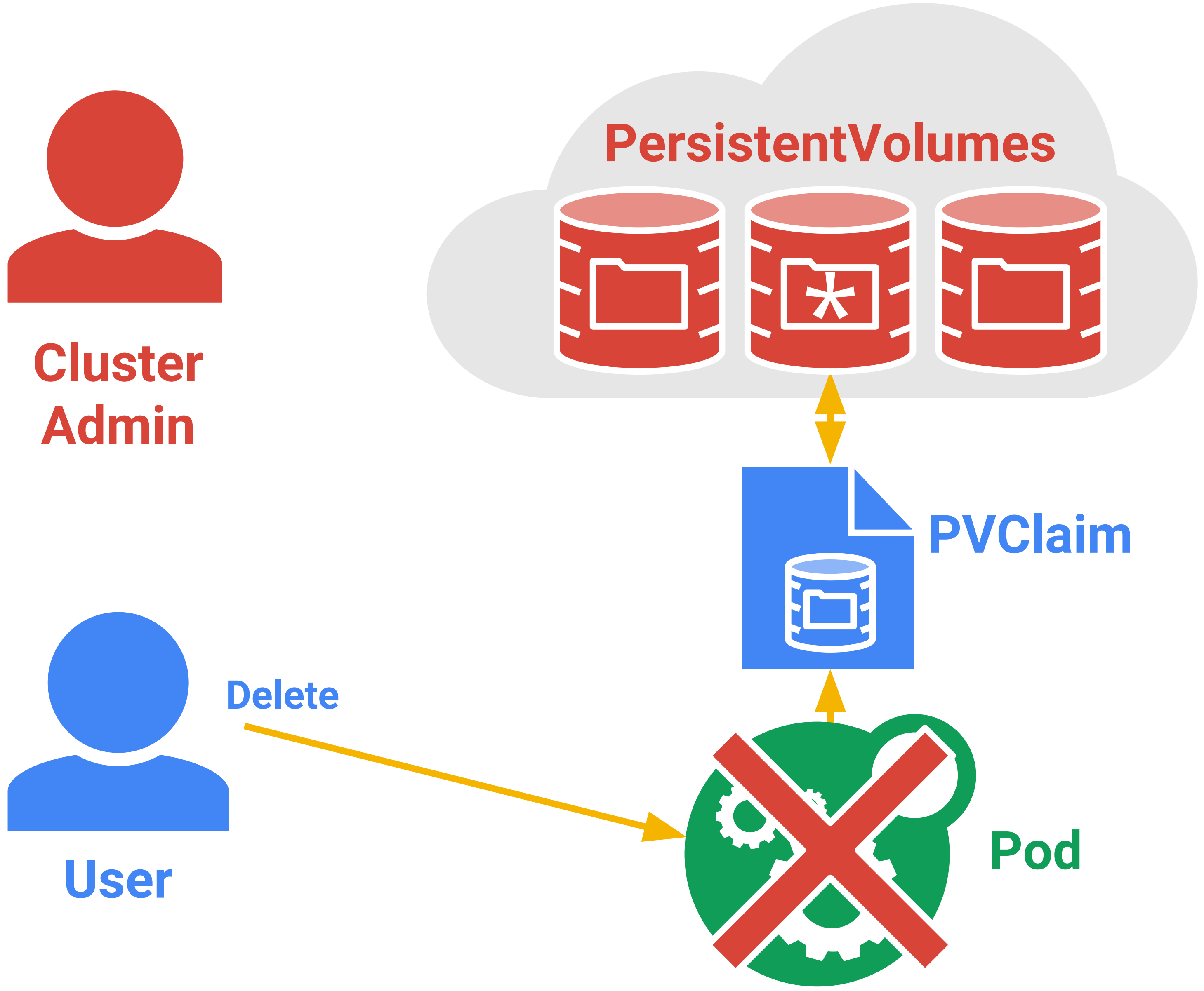
PersistentVolumes



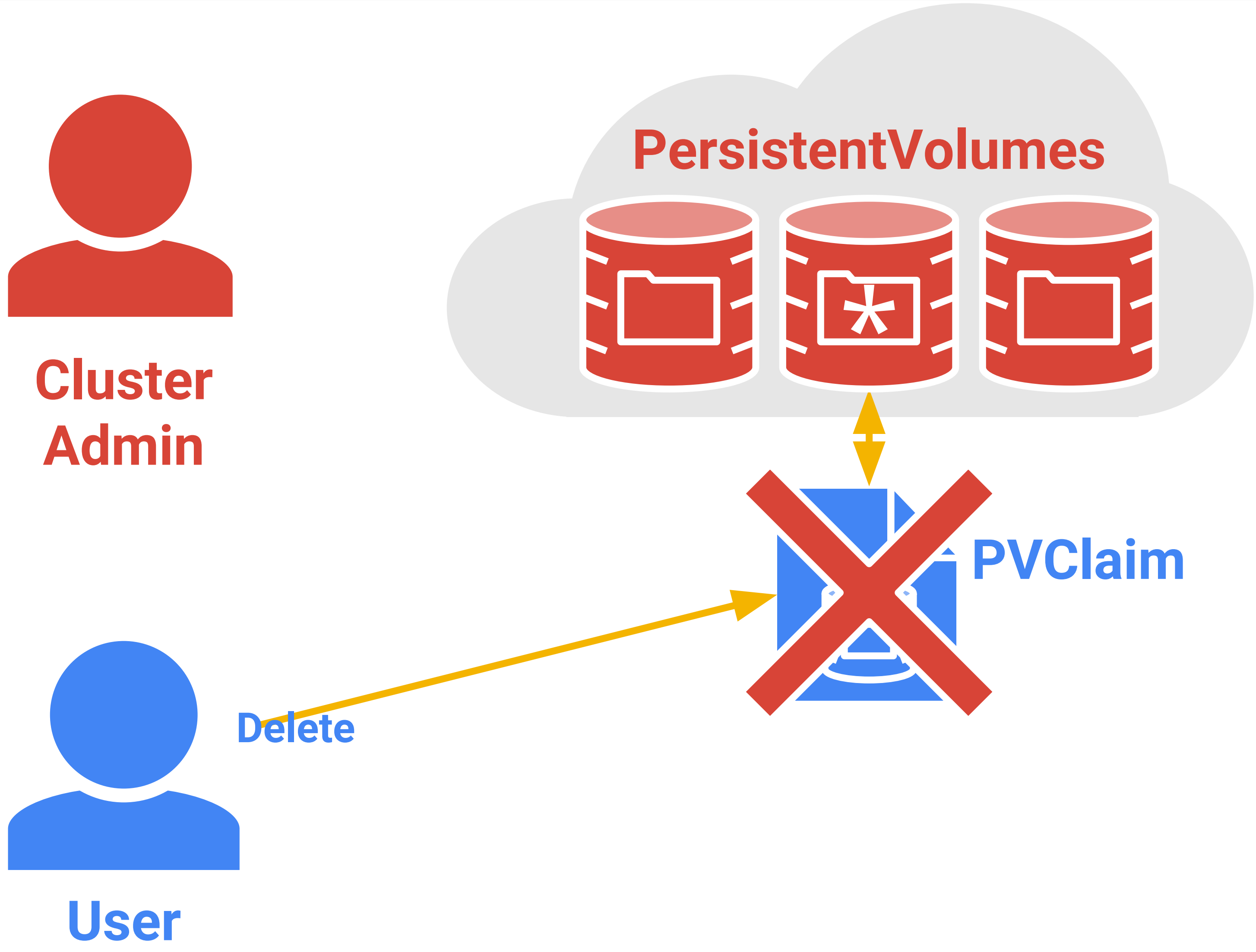
PersistentVolumes



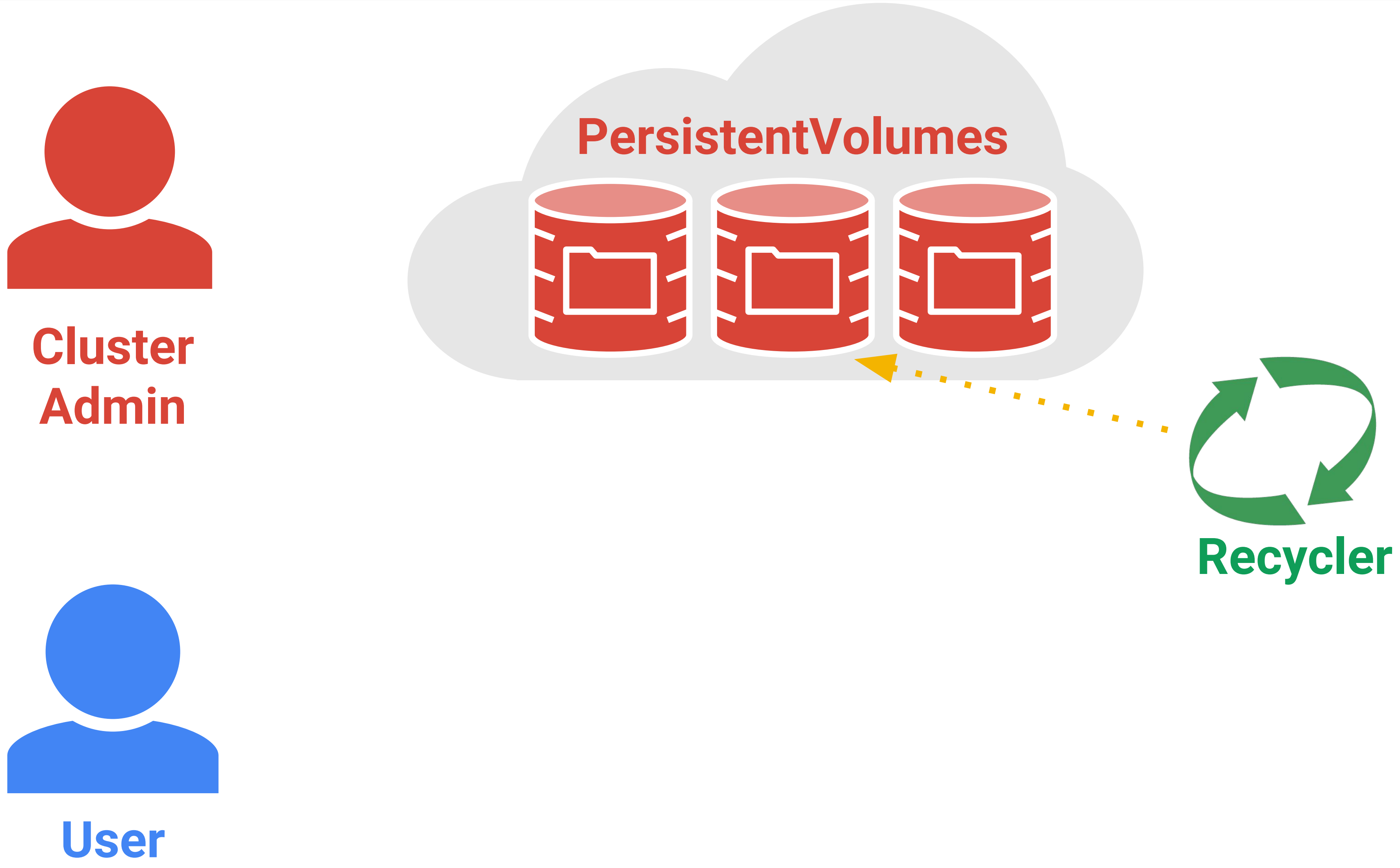
PersistentVolumes



PersistentVolumes



PersistentVolumes



StatefulSets

StatefulSets

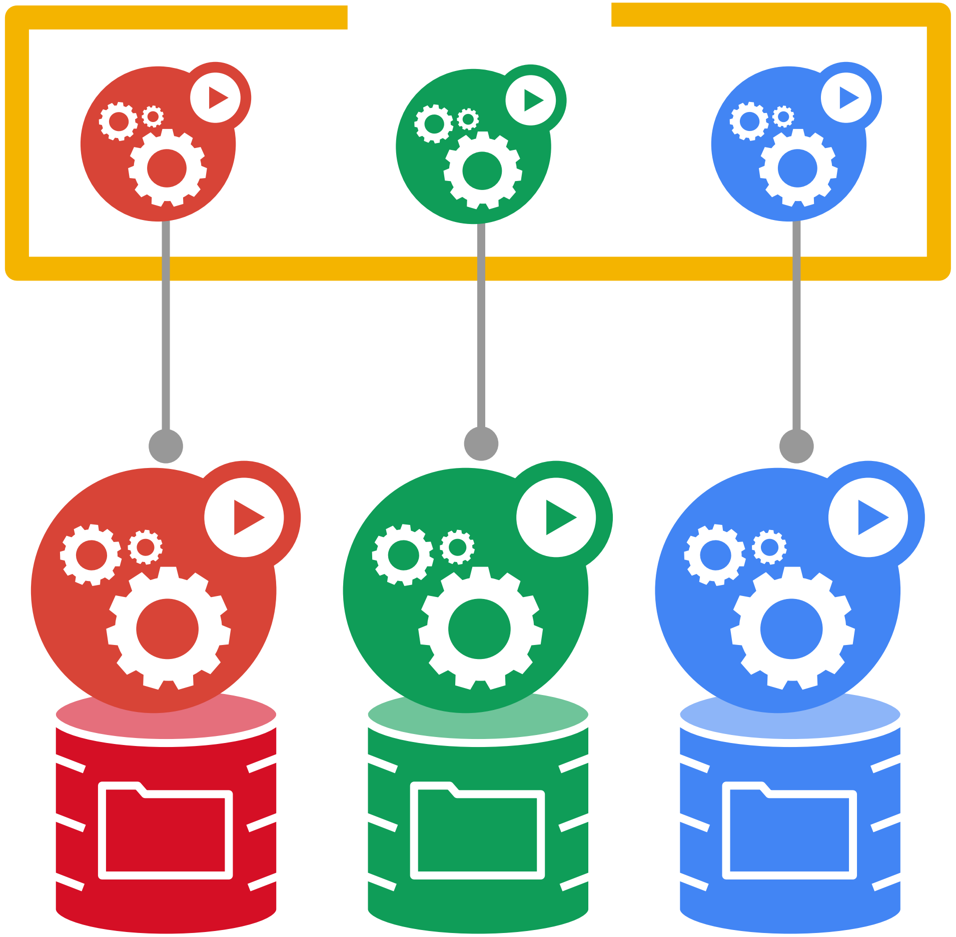
Goal: enable clustered software on Kubernetes

- mysql, redis, zookeeper, ...

Clustered apps need "identity" and sequencing guarantees

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

Status: ALPHA in Kubernetes v1.3



ConfigMaps

ConfigMaps

Goal: manage app configuration

- ...without making overly-brittle container images

12-factor says config comes from the environment

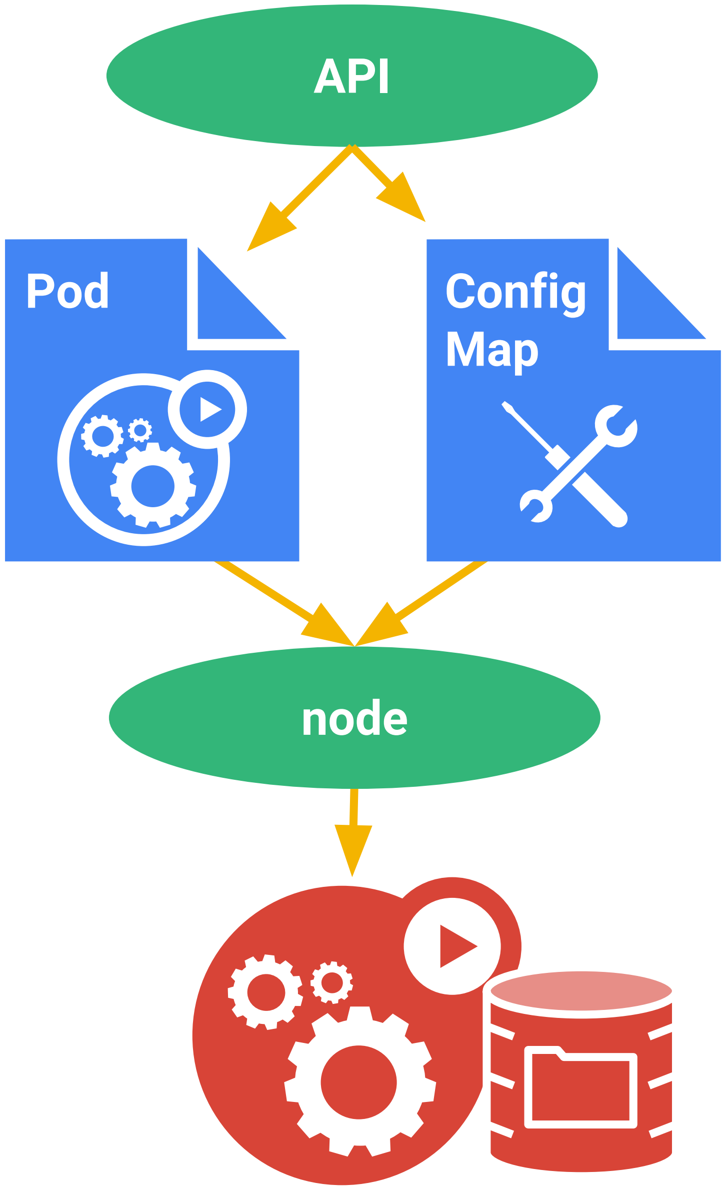
- Kubernetes is the environment

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

Status: GA in Kubernetes v1.2



Secrets

Secrets

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

- don't put secrets in the container image!

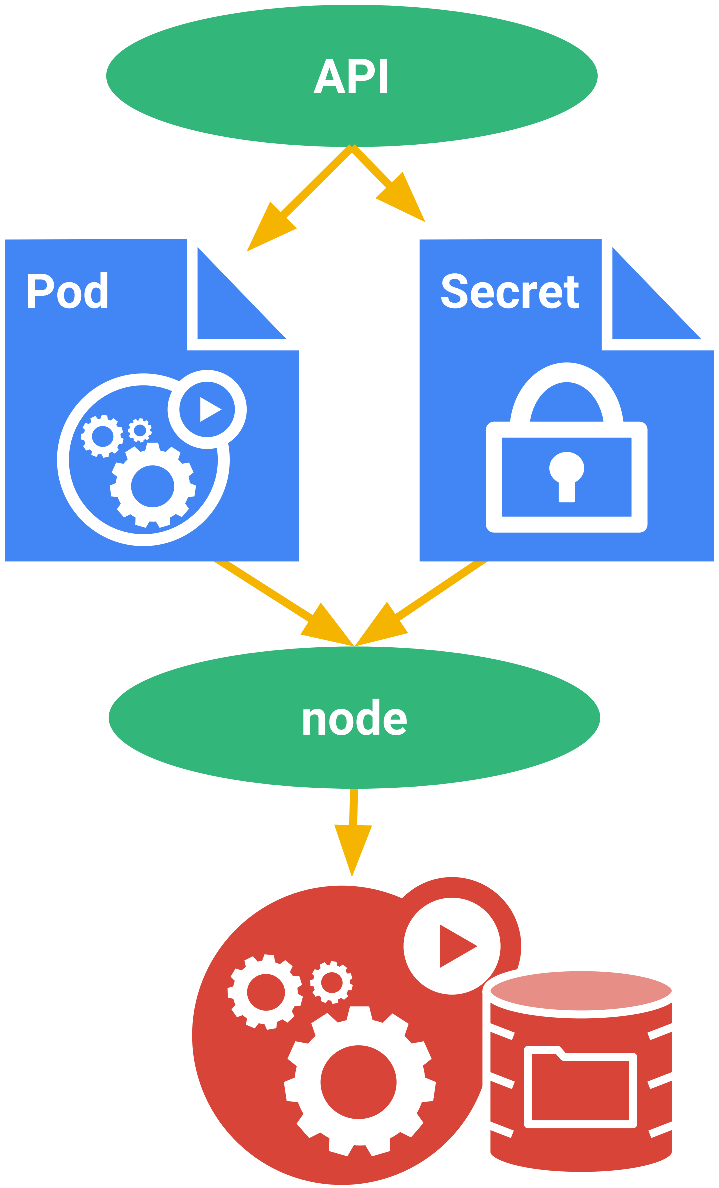
12-factor says config comes from the environment

- 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



HorizontalPodAutoscalers

HorizontalPodAutoScalers

Goal: Automatically scale pods as needed

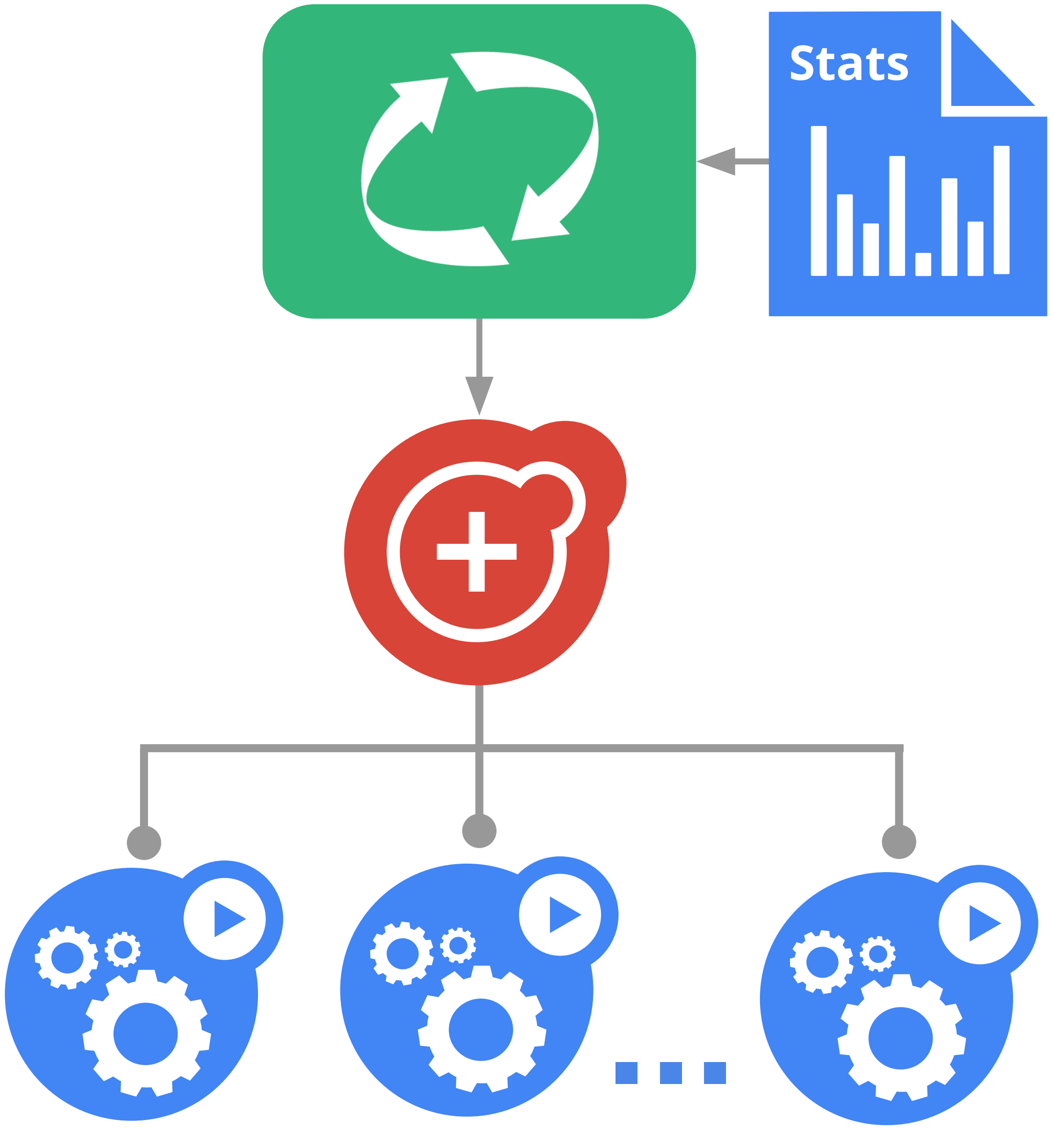
- based on CPU utilization (for now)
- custom metrics in Alpha

Efficiency now, capacity when you need it

Operates within user-defined min/max bounds

Set it and forget it

Status: GA in Kubernetes v1.2



Multi-Zone Clusters

Multi-Zone Clusters

Goal: zone-fault tolerance for applications

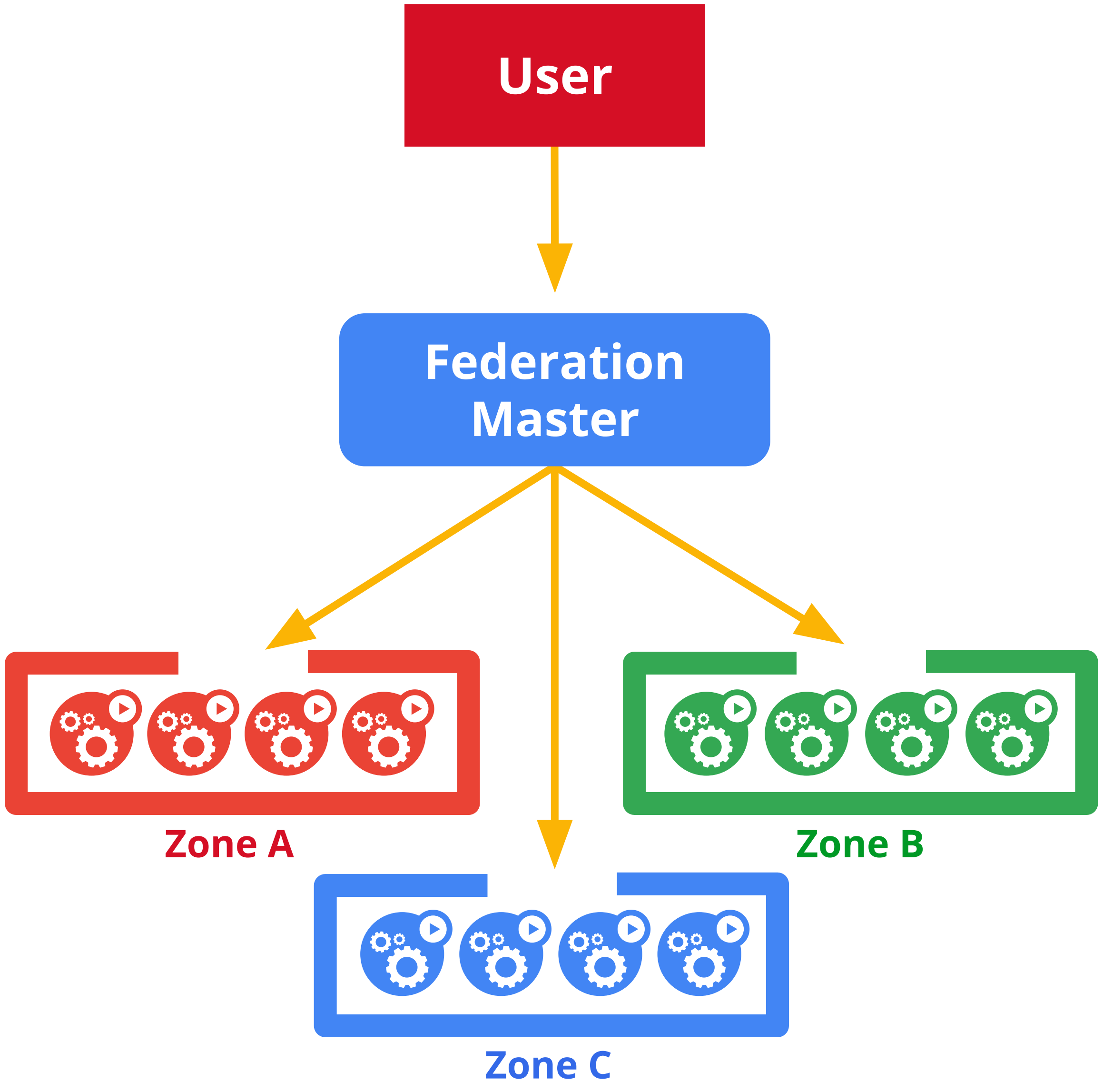
Zero API changes relative to kubernetes

- Create services, ReplicaSets, etc. exactly as usual

Nodes and PersistentVolumes are labelled with their availability zone

- Fully automatic for GKE, GCE, AWS
- Manual for on-premise and other cloud providers (for now)

Status: GA in Kubernetes v1.2



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

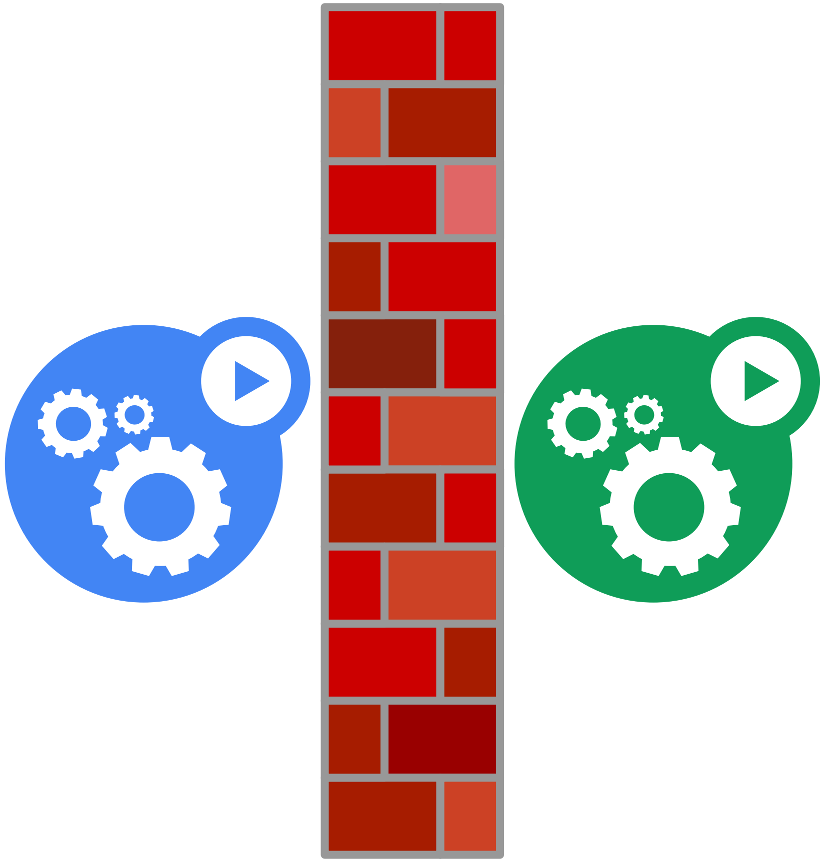
- create new Namespaces as needed
  - per-user, per-app, per-department, etc.
- part of the API - NOT private machines
- most API objects are namespaced
  - part of the REST URL path
- Namespaces are just another API object
- One-step cleanup - delete the Namespace
- Obvious hook for policy enforcement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

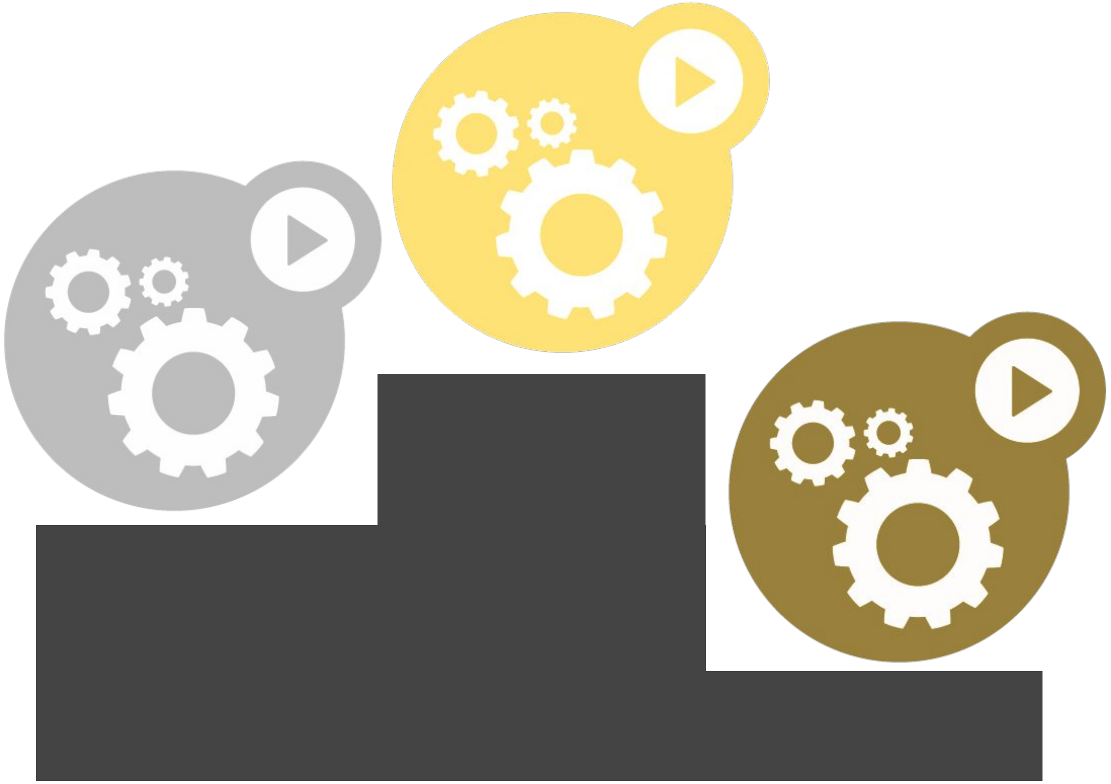
- `request > 0 && limit > request`

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

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

Scalability

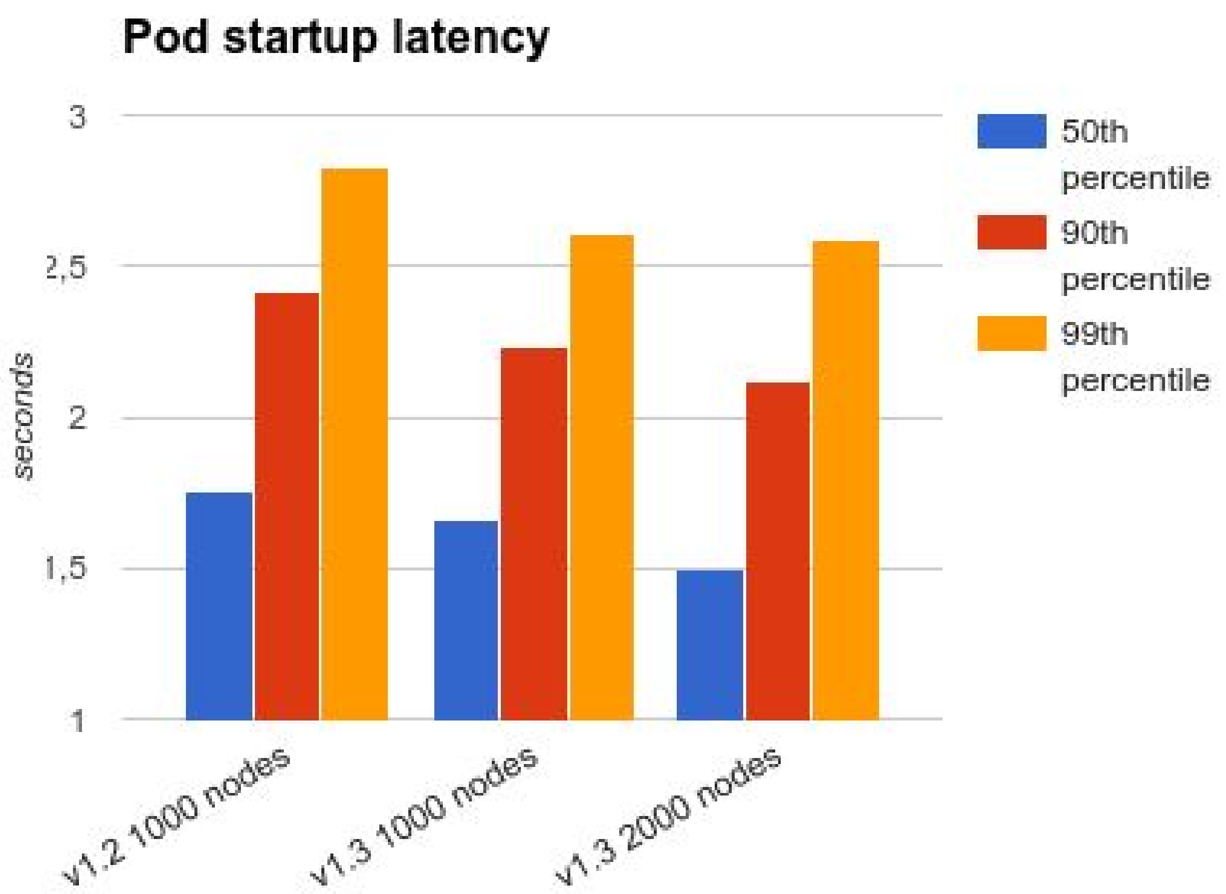
Scalability & Performance

SLO met at <2000 nodes, <60000 pods

- 99% of API calls return in < 1 second
- 99% of pods start in < 5 seconds

Coming soon

- protobufs in API storage (already enabled on the wire)
- 5000 nodes



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 <u>non-starter</u>

Network-centric: IP addresses are cheap

No grouping: Labels are the <u>only</u> groups

Sets > Pets: Manage your workload in bulk

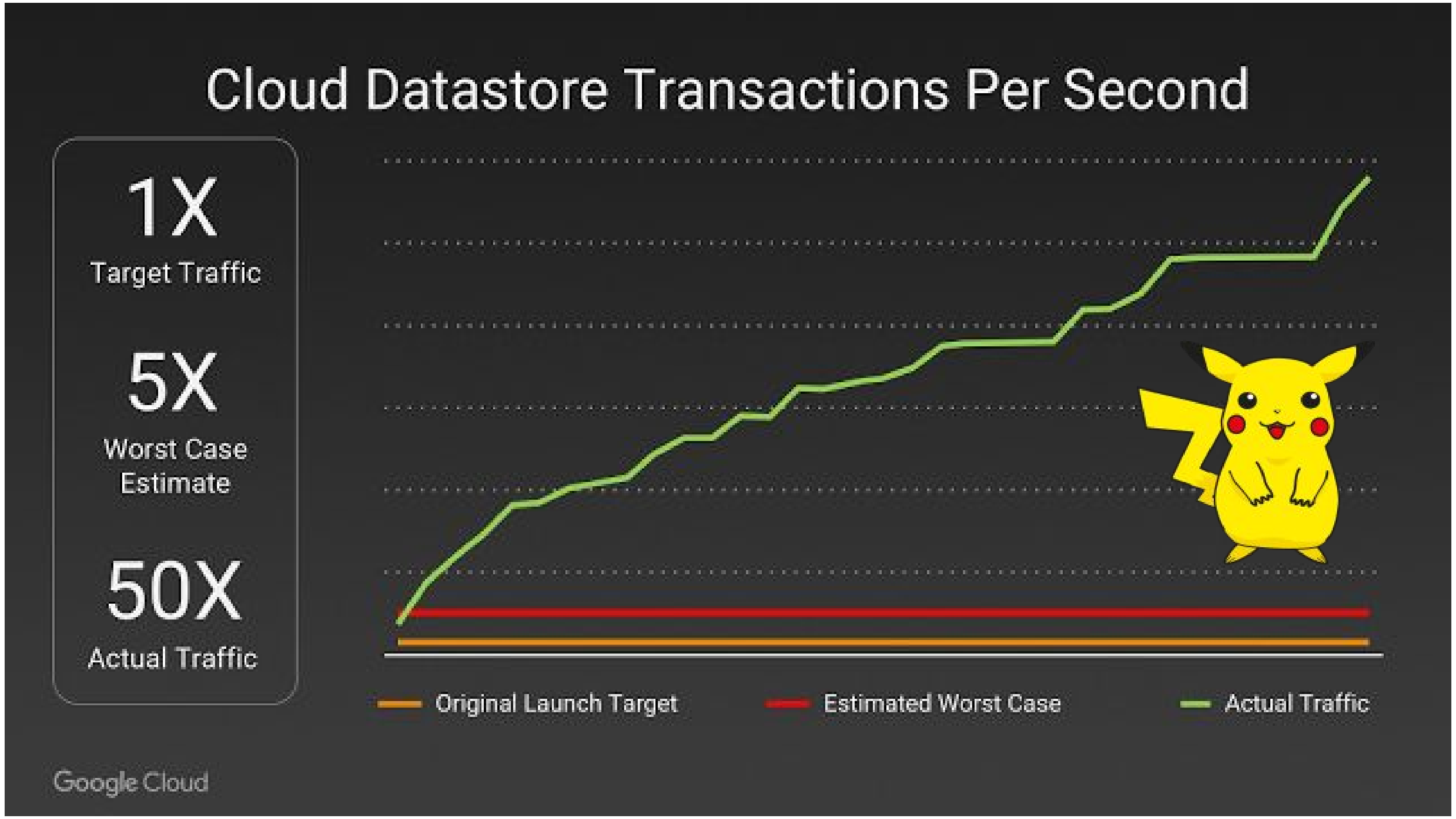
Open > Closed: Open Source, standards, REST, JSON, etc.

Kubernetes (K8s) Community

~5k Commits in 1.4 over 3 months	> 800 Unique Contributors	Top 0.01% of all Github Projects	2500+ External Projects Based on K8s

Companies Contributing

Companies Using



“Niantic chose GKE for its ability to **orchestrate their container cluster at planetary-scale**, freeing its team to focus on deploying live changes for their players.” - Niantic

Further Reading

If this talk was interesting, deeper academic reading on cluster management:

"Borg, Omega, and Kubernetes"

ACM Queue, March 2, 2016, Volume 14, issue 1

http://queue.acm.org/detail.cfm?id=2898444

Or a hands-on "Hello World" quickstart to build a Docker image and run it on a Kubernetes cluster:

http://kubernetes.io/docs/hellonode/

Another hard problem: how do you run *N* Kubernetes clusters as a service?

- create/delete, update, monitor, repair, escalate, upgrade, backup/restore, zonal isolation, incremental rollouts, support ticket escalation, provisioning, and more!

Questions?

Potential discussion:

- What about Docker Swarm?
- … Mesos?
- What's next for Kubernetes and Container Engine?
- Why Google not FB/Uber/MS/Ama/etc?
- How do I get an internship / job?
  - Let's discuss!

- Alex on Philosophy:
  - Imperative vs. declarative
  - Orchestration vs. choreography
  - Product vs. tech
  - User guide vs. design doc
  - Engineering code vs. organizations
  - Your team is a design parameter
  - Launch and iterate; MVP

More questions?
Happy to chat!

- Lunch
- 1:1's after that
- mohr@google.com
- 590s@alexmohr.com