Kubernetes: What's New

LISA'15

Tim Hockin <thockin@google.com>
Senior Staff Software Engineer
@thockin

Google Cloud Platform

This is “Kubernetes 201”

If you’re lost, I’m happy to answer questions later or at the BoF tonight

Obligatory Background

Everything at Google runs in containers:

- Gmail, Web Search, Maps, ...
- MapReduce, batch, ...
- GFS, Colossus, ...
- Even **Google's Cloud Platform:** VMs run in containers!

Shipping Containers At Clyde, by Steve Gibson

Everything at Google runs in containers:

- Gmail, Web Search, Maps, ...
- MapReduce, batch, ...
- GFS, Colossus, ...
- Even **Google's Cloud Platform:** VMs run in containers!

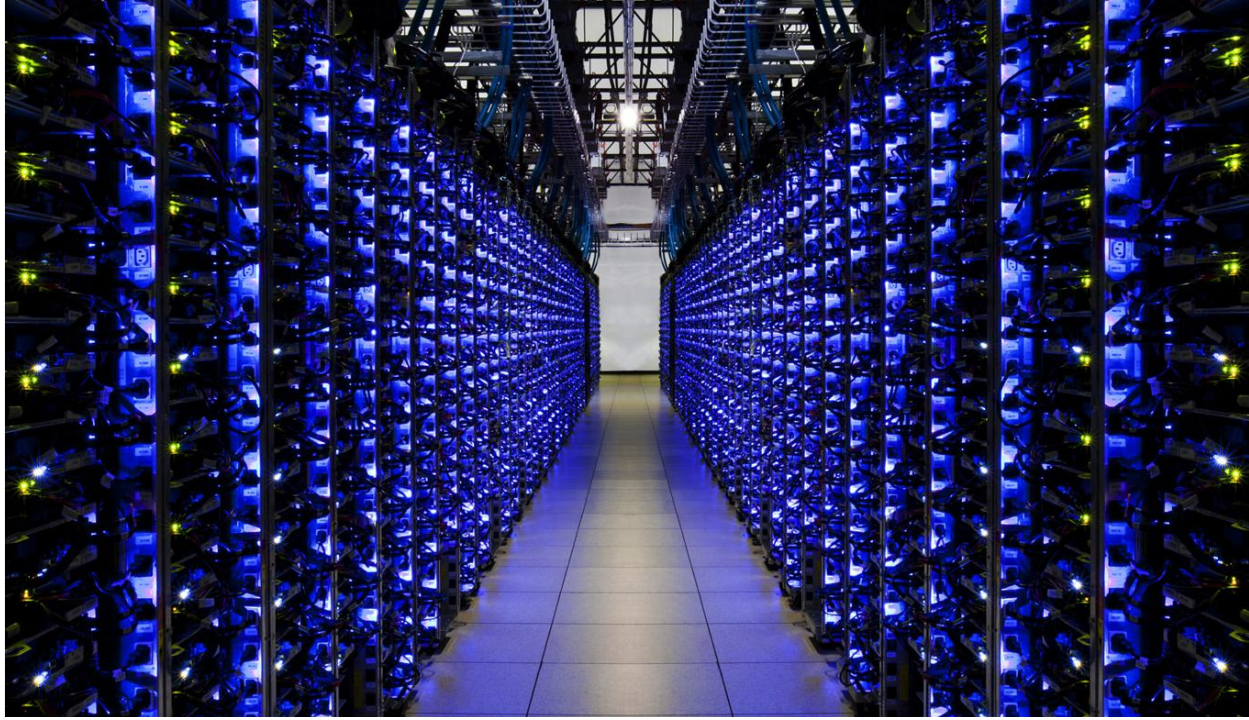
We launch **billions** of containers **every week**

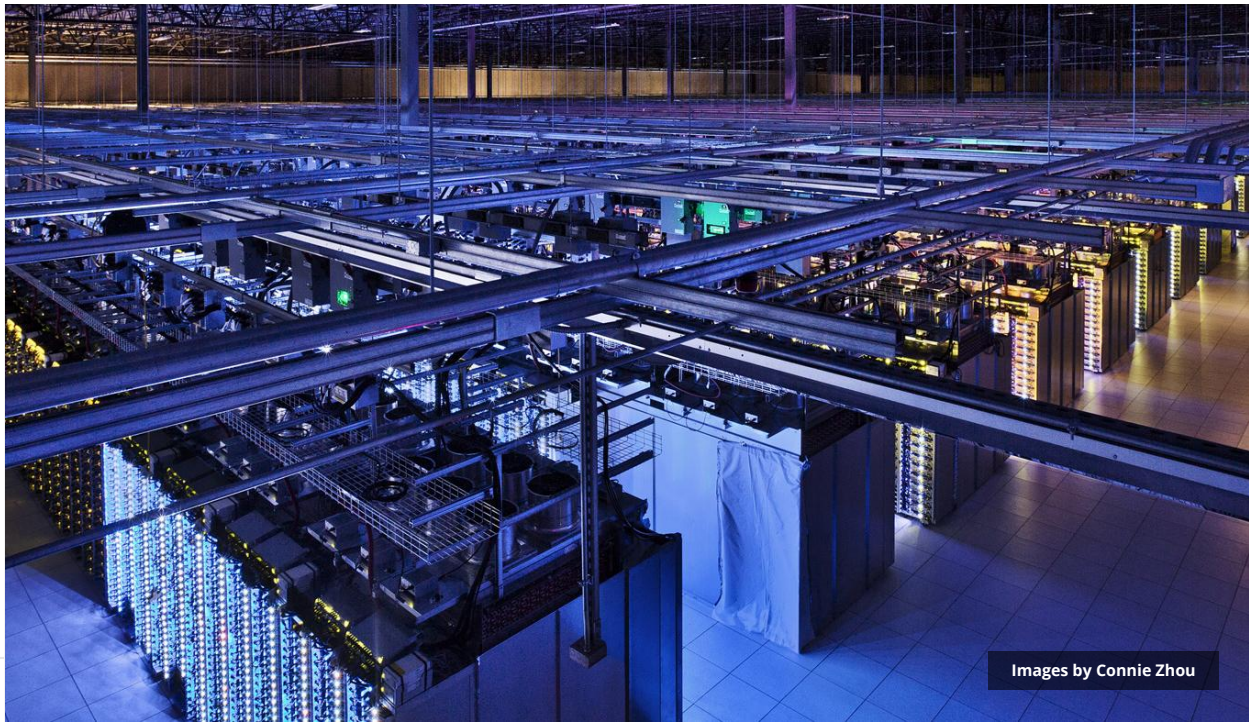
Shipping Containers At Clyde, by Steve Gibson

But it’s all so different!

- Deployment
- Management, monitoring
- Isolation (very complicated!)
- Updates
- Discovery
- Scaling, replication, sets

A **fundamentally different** way of managing applications requires different tooling and abstractions





Images by Connie Zhou

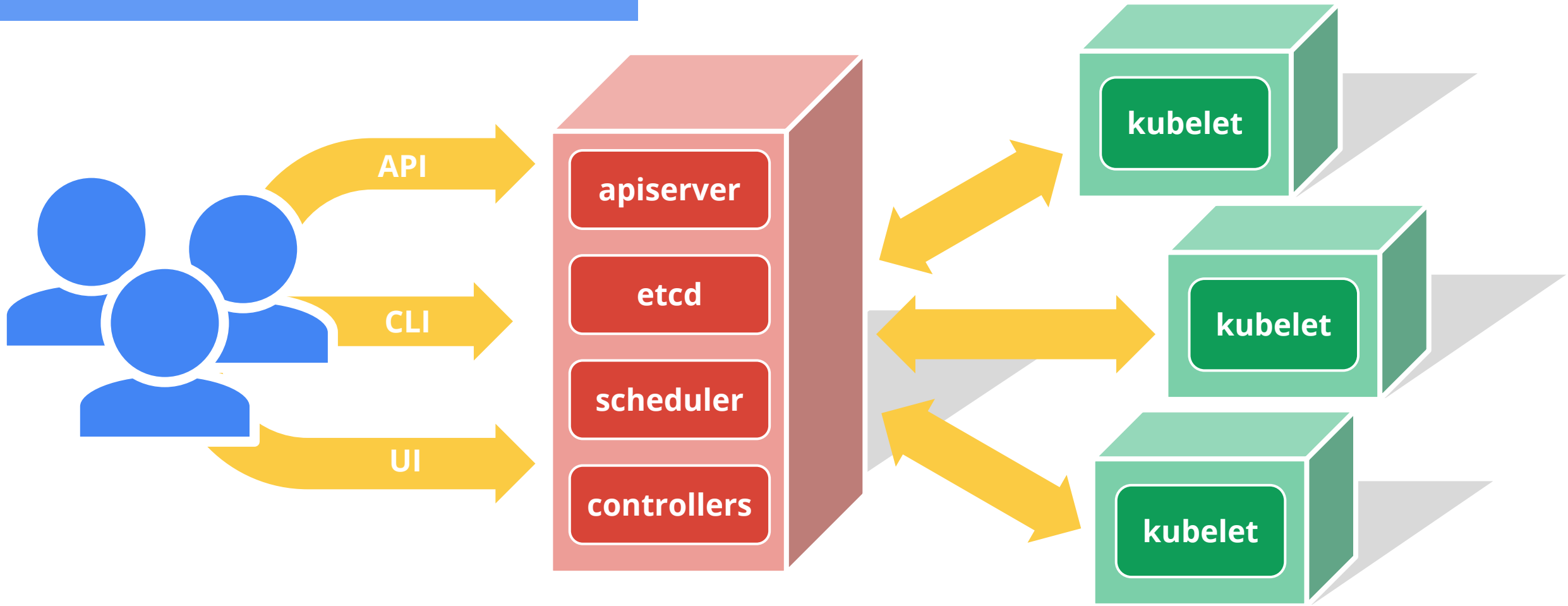
Kubernetes

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

- Runs and manages containers
- 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

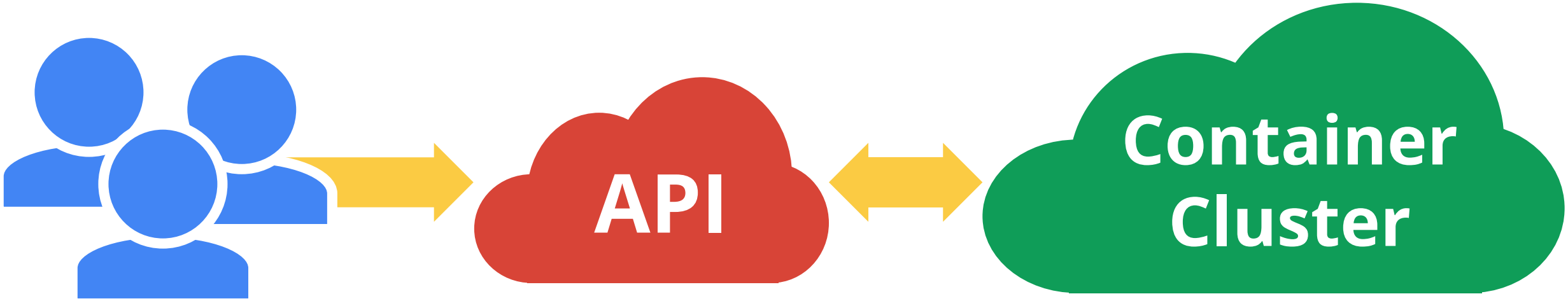


users

master

nodes

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

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
- Replication controllers
- Services
- Volumes

<u>This is the fun part!</u>

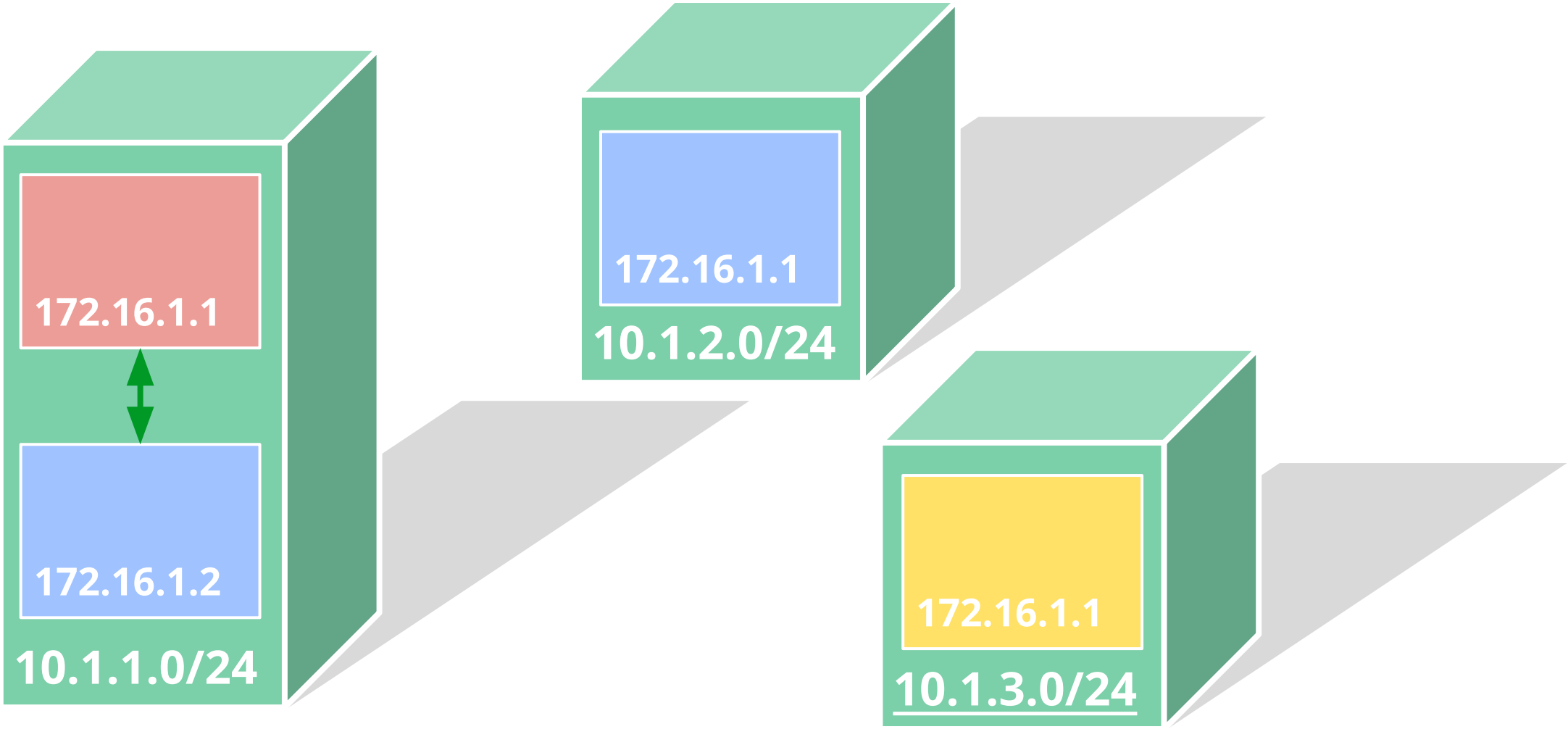
A distinct set of problems from cluster setup and management

Don’t make developers deal with cluster administration!

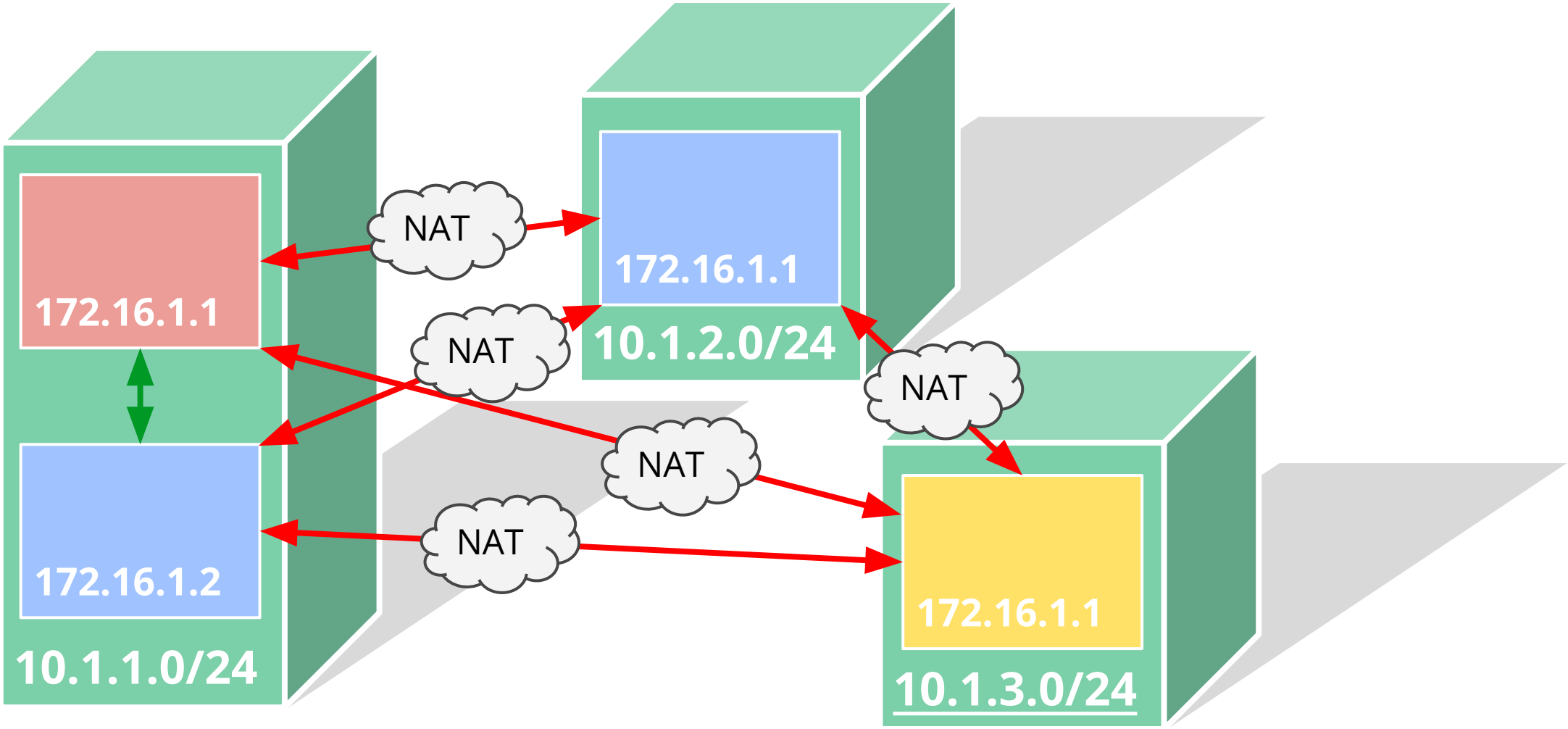
Accelerate development by focusing on the applications, not the cluster

Networking

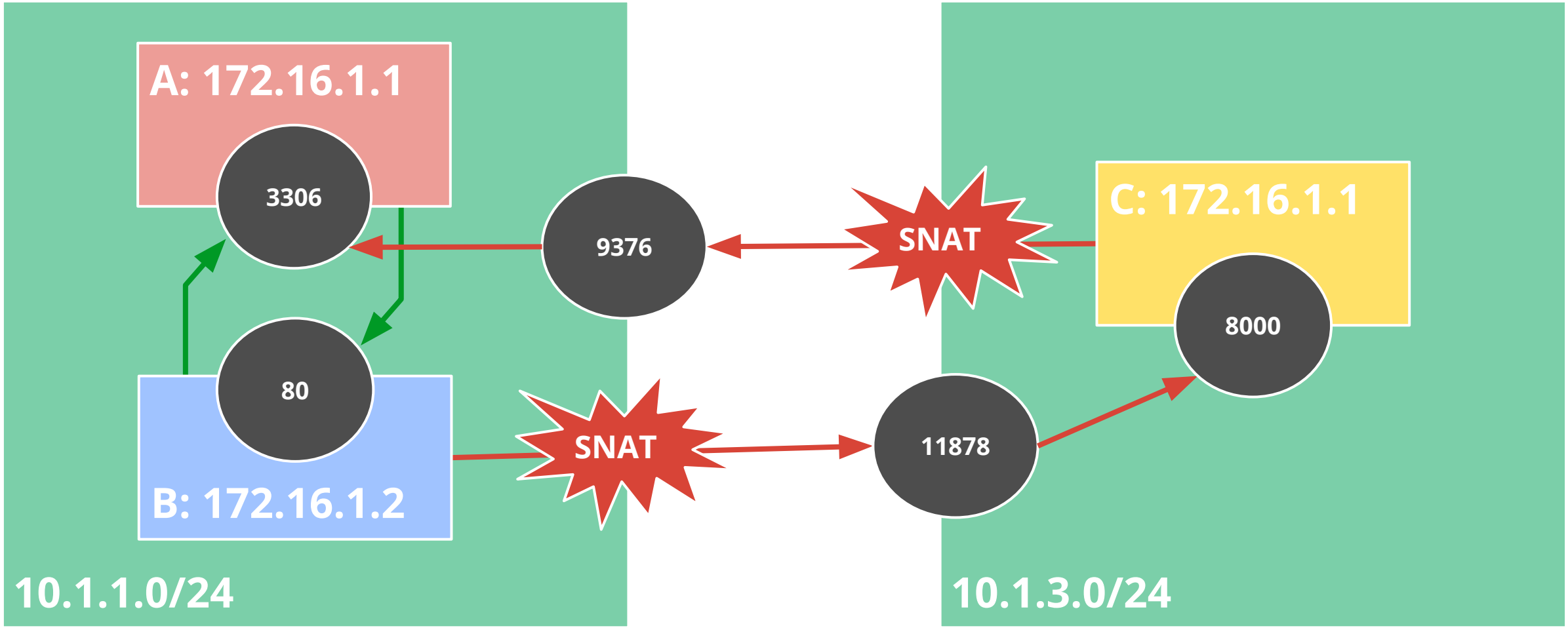
Docker networking



Docker networking



Host ports


A: 172.16.1.1
3306
9376
SNAT
C: 172.16.1.1
8000
80
SNAT
11878
B: 172.16.1.2
10.1.1.0/24
10.1.3.0/24

Host ports

Kubernetes networking

IPs are **routable**

- vs docker default private IP

Pods can reach each other without NAT

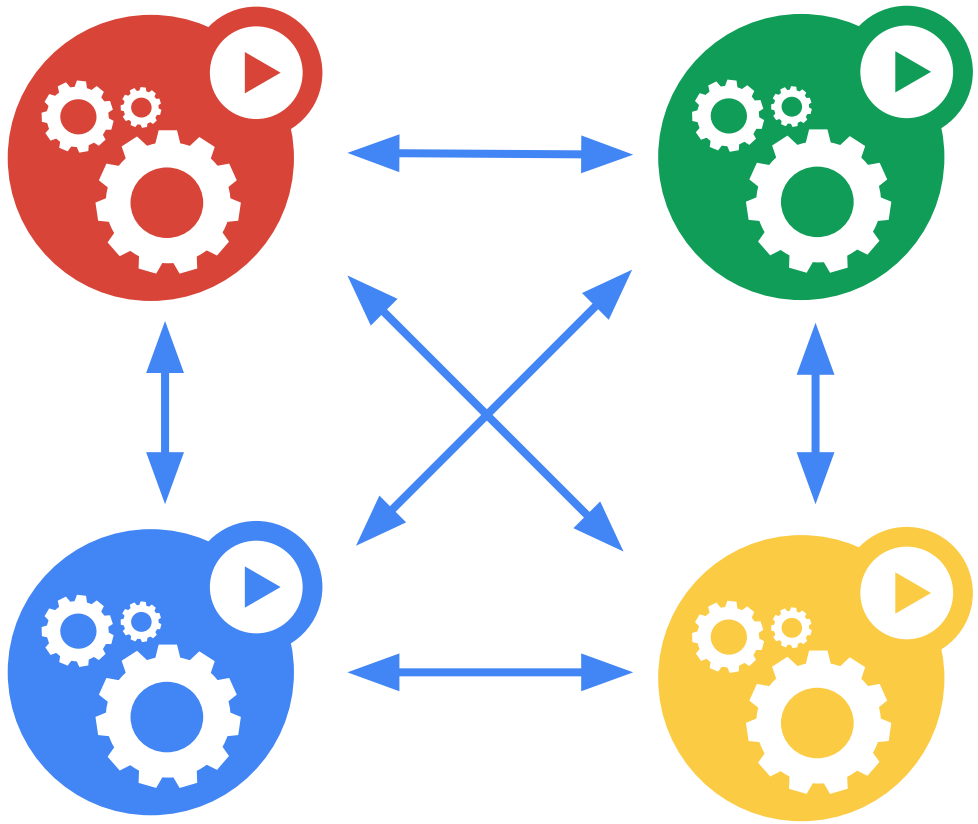
- even across nodes

No brokering of port numbers

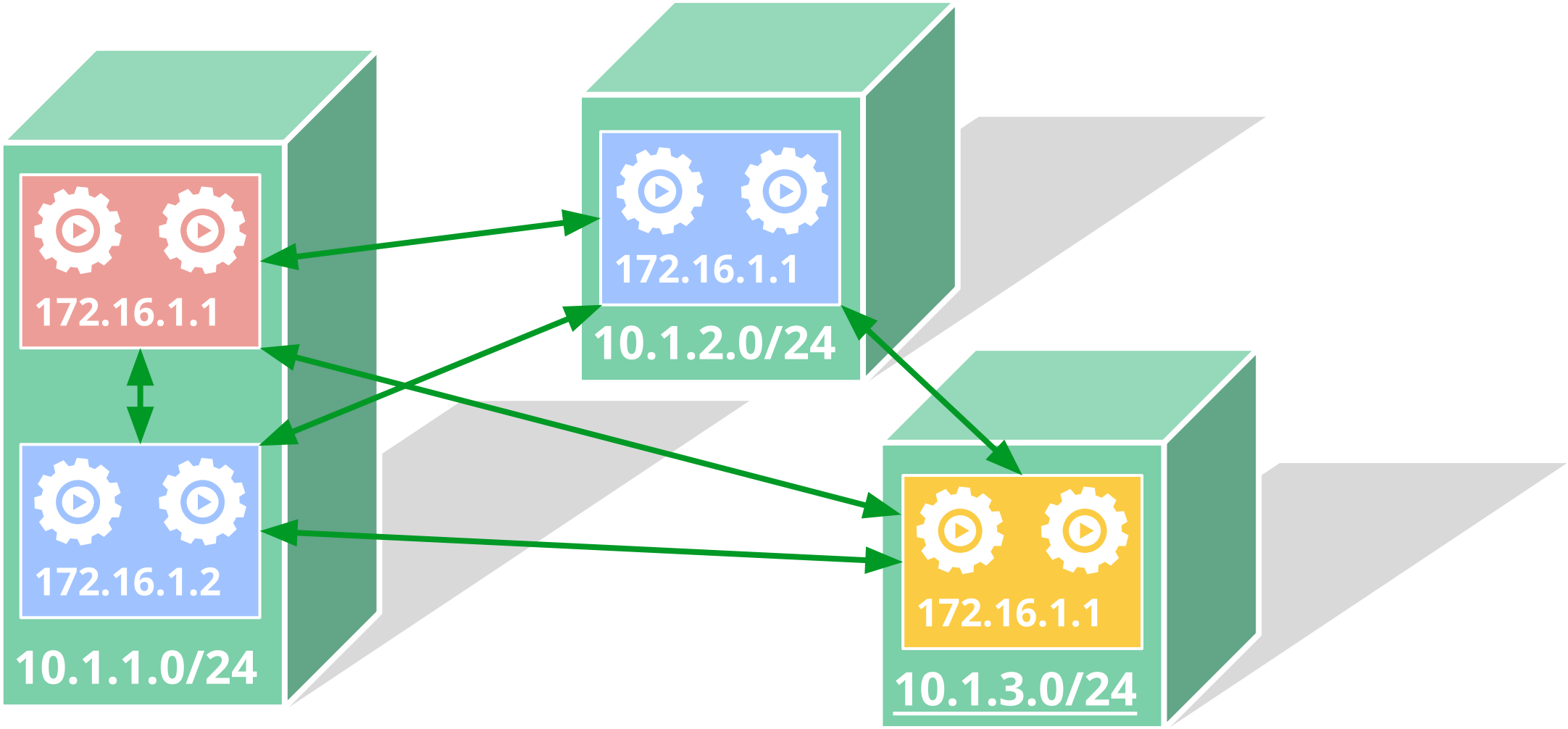
- too complex, why bother?

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

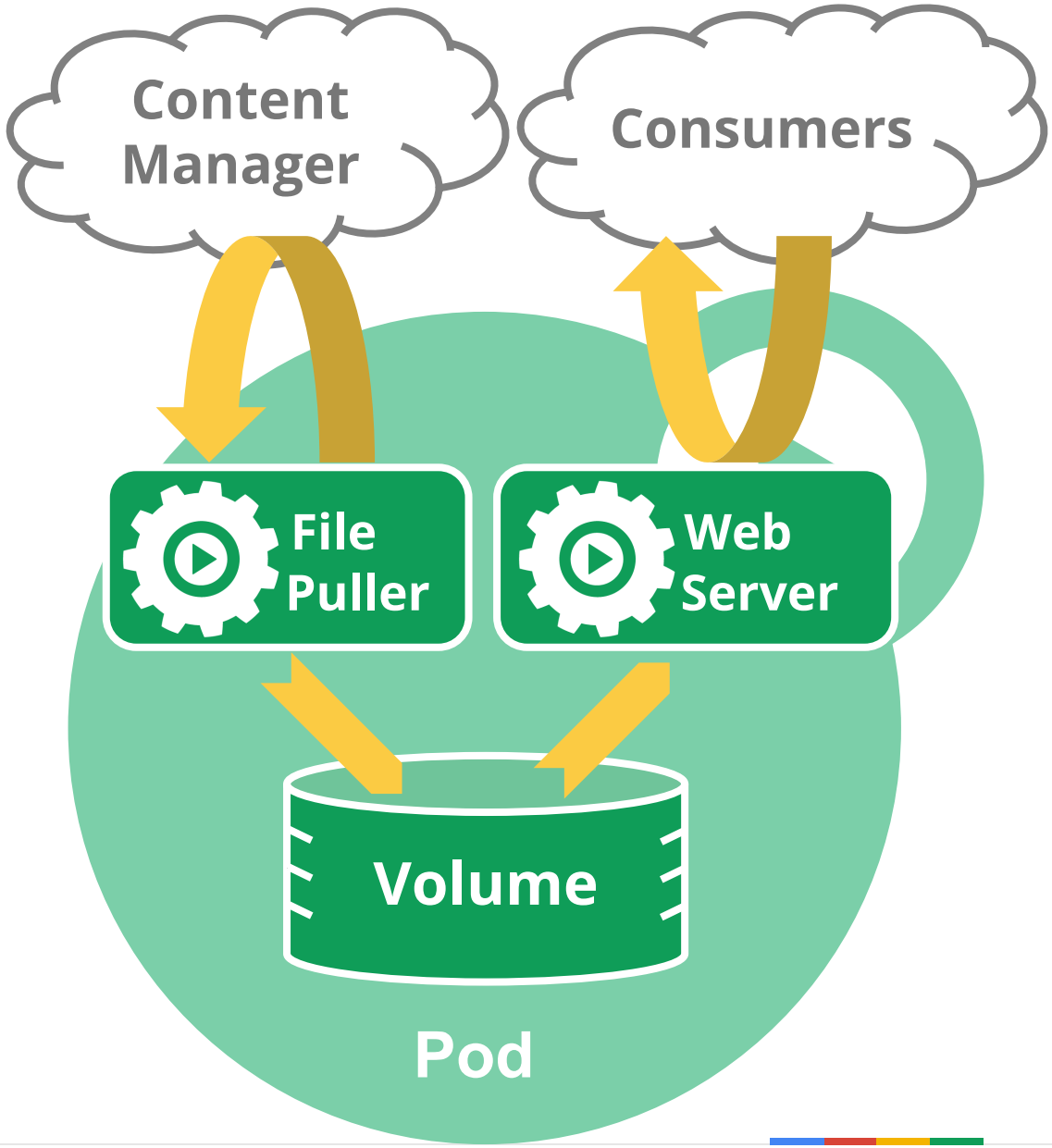
Shared namespace

- 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



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

ReplicationControllers

ReplicationControllers

A simple control loop

Runs out-of-process wrt API server

Has 1 job: ensure N copies of a pod

- if too few, start some
- if too many, kill some
- grouped by a selector

Cleanly layered on top of the core

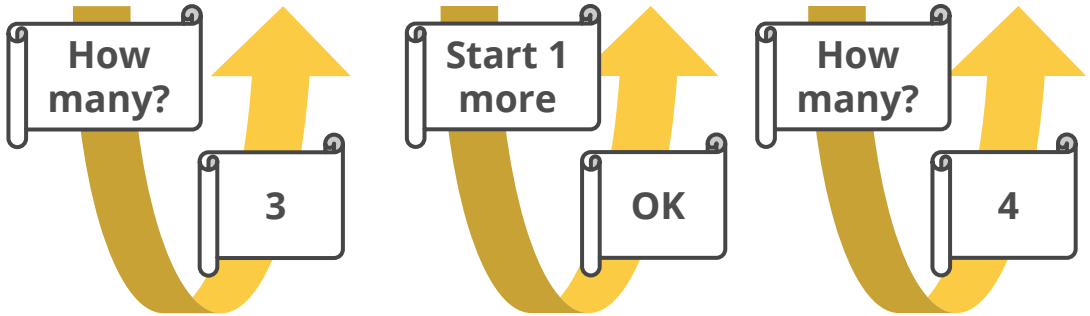
- all access is by public APIs

Replicated pods are fungible

- No implied order or identity

ReplicationController

- **name = “my-rc”**
- **selector = {“App”: “MyApp”}**
- **podTemplate = { … }**
- **replicas = 4**



API Server

Services

Services

A group of pods that **work together**

- grouped by a selector

Defines access policy

- “load balanced” or “headless”

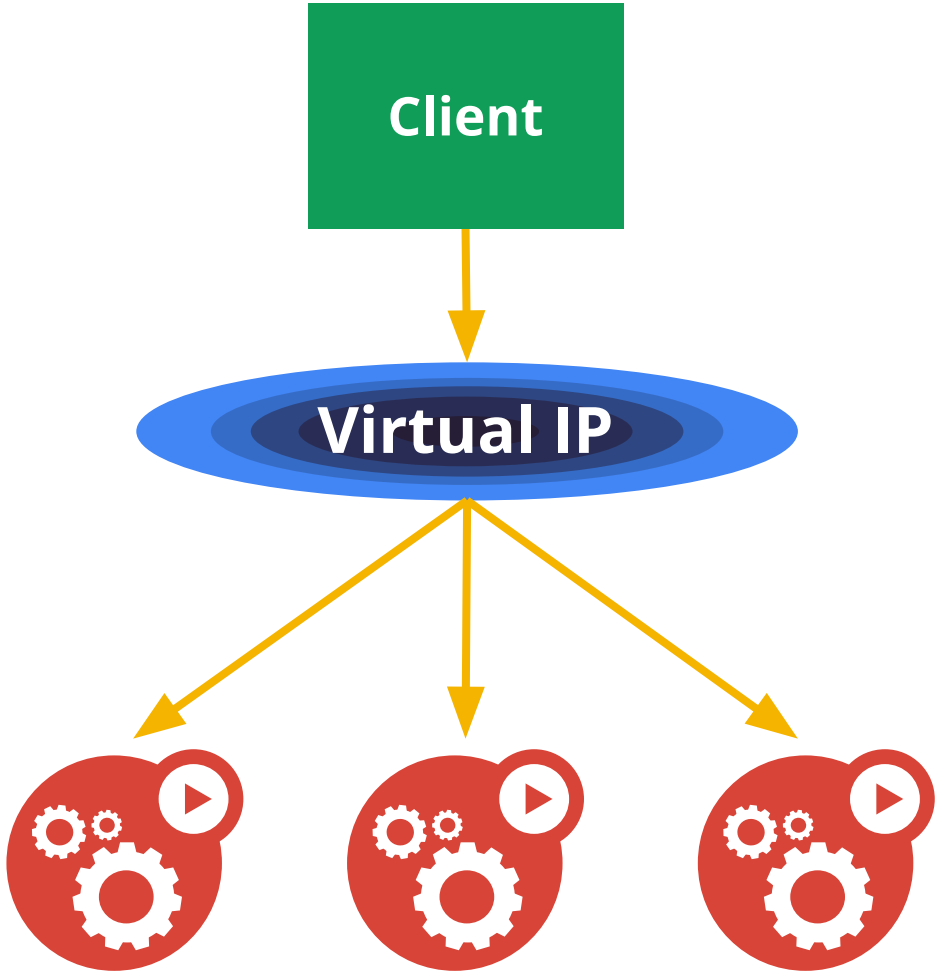
Gets a stable **virtual IP** and port

- sometimes called the service *portal*
- also a DNS name

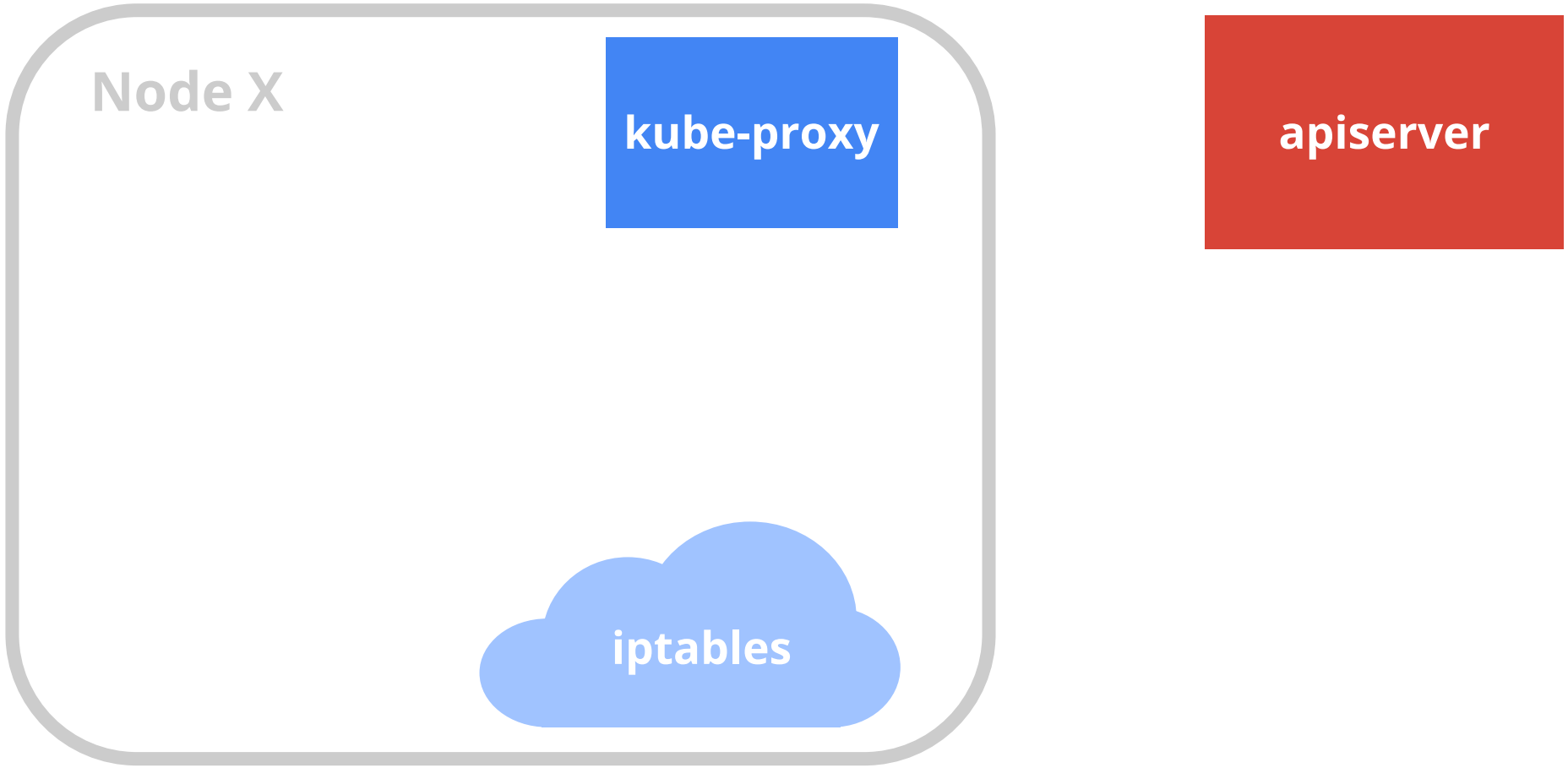
VIP is managed by *kube-proxy*

- watches all services
- updates iptables when backends change

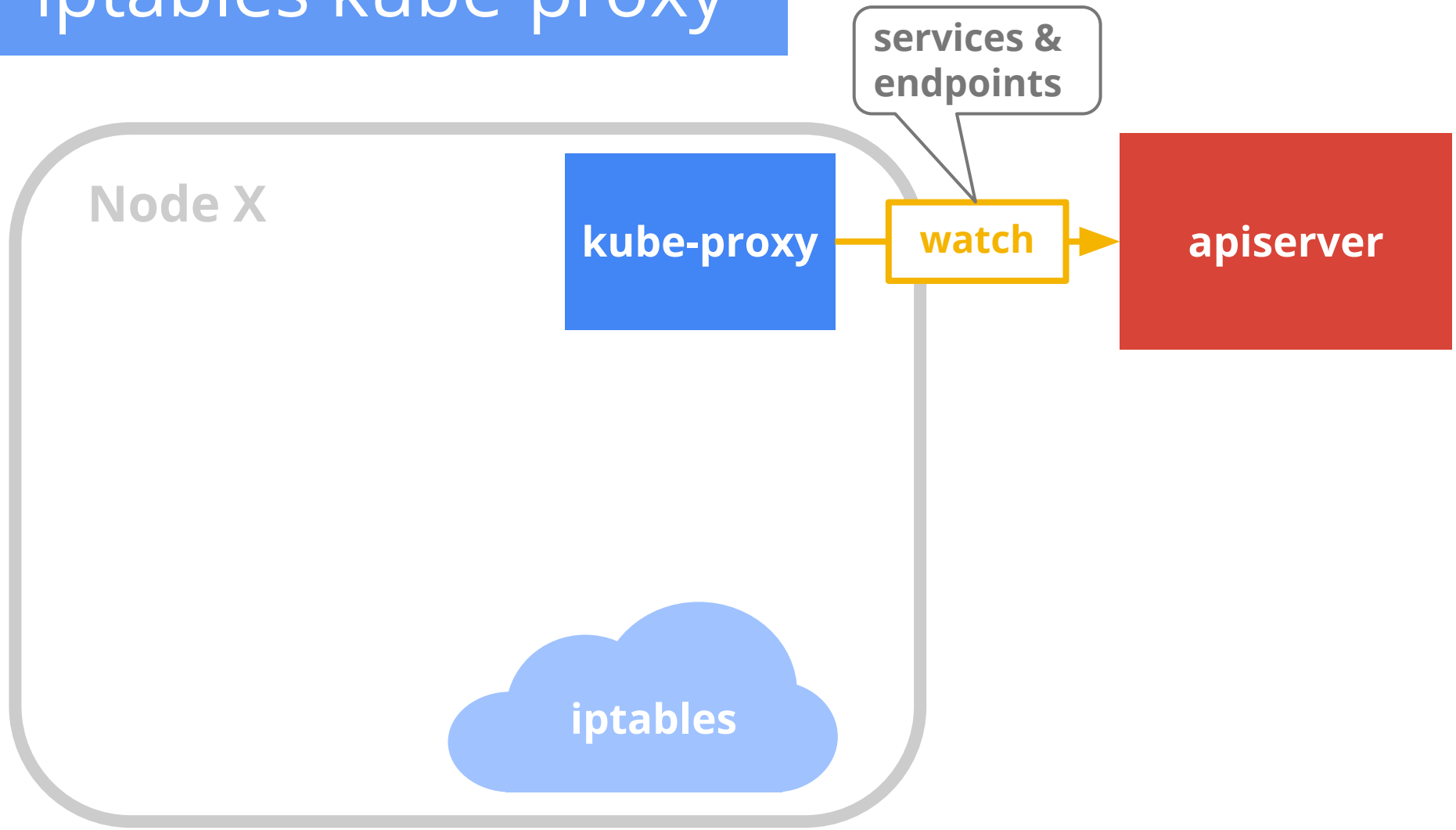
Hides complexity - ideal for non-native apps



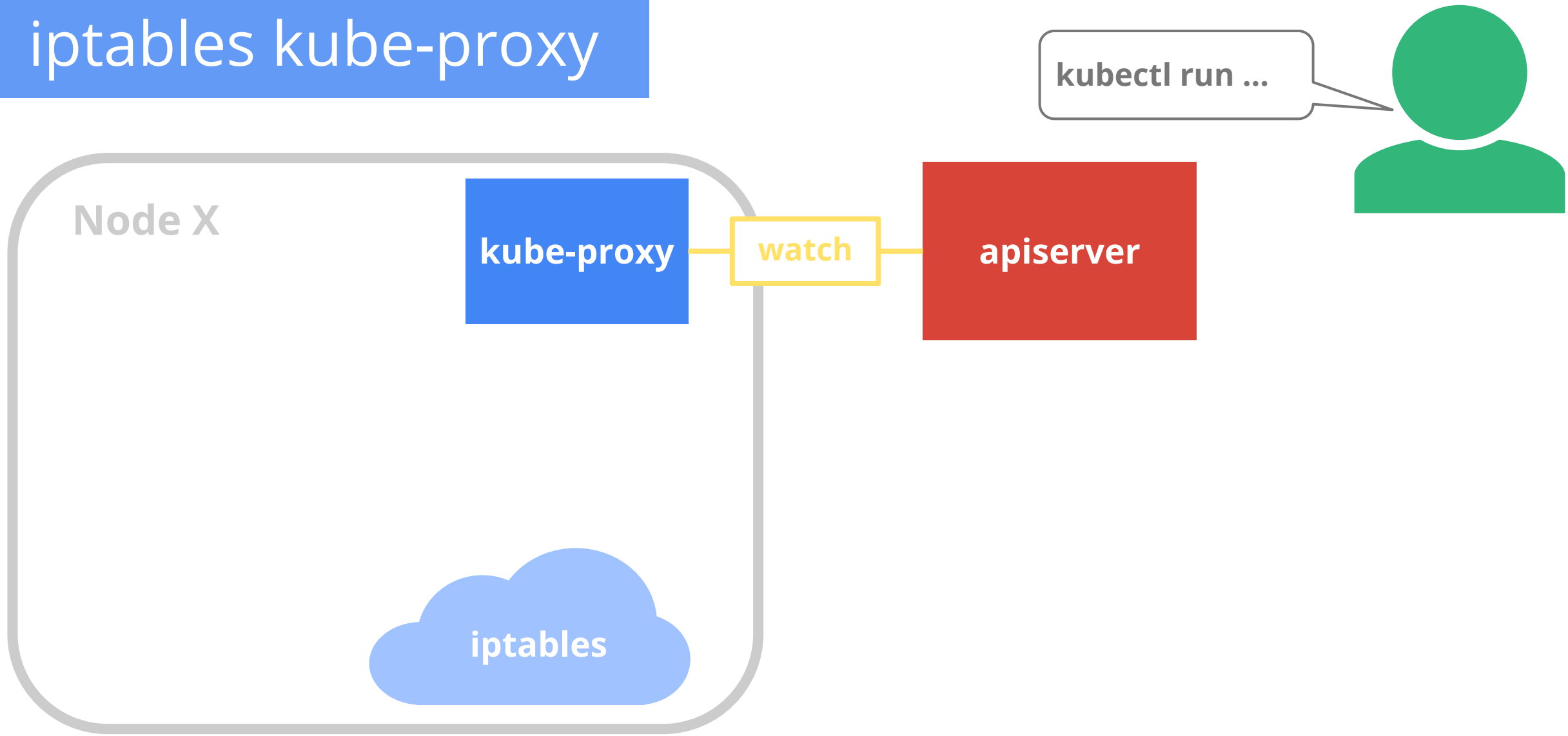
iptables kube-proxy



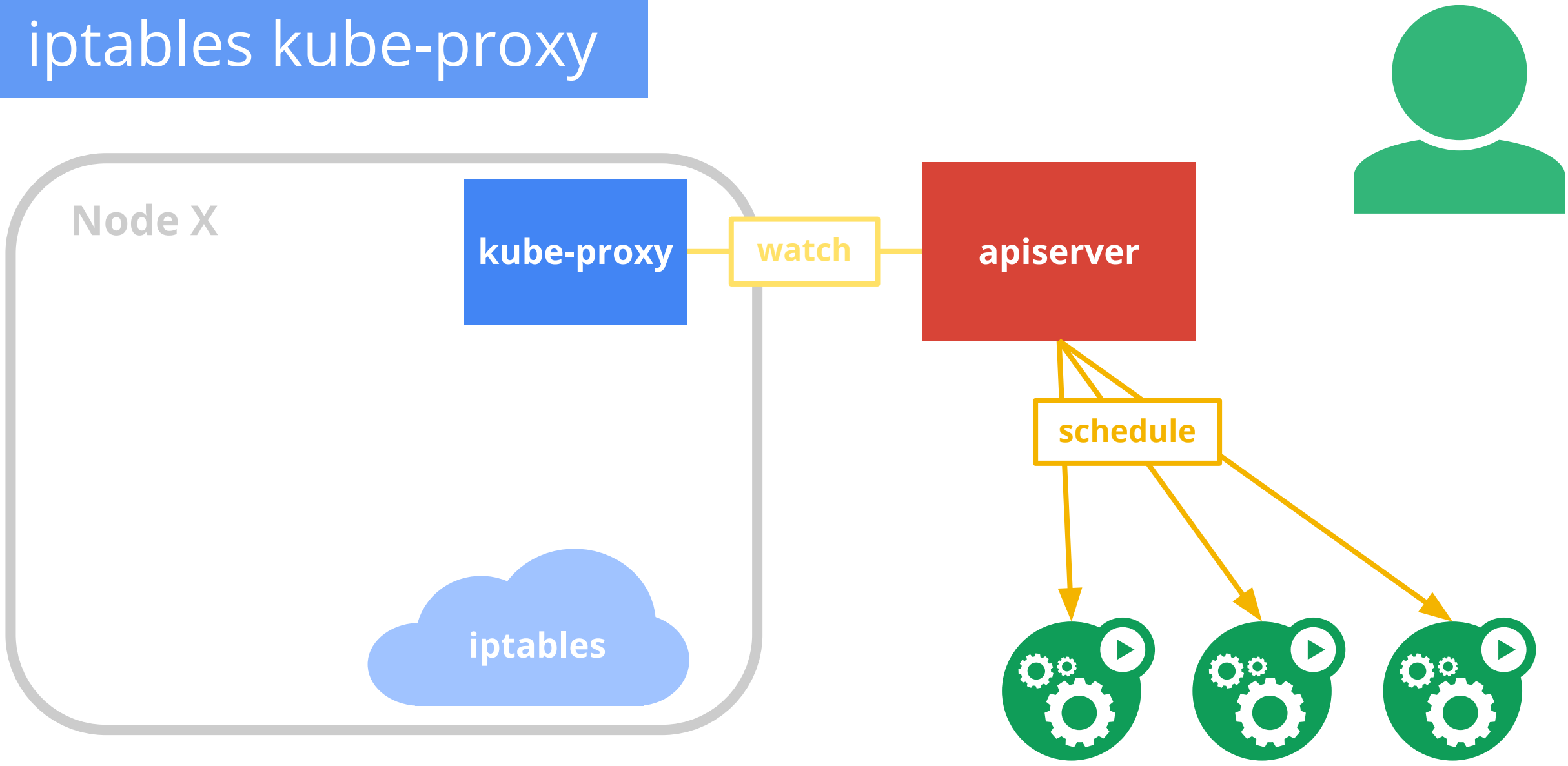
iptables kube-proxy



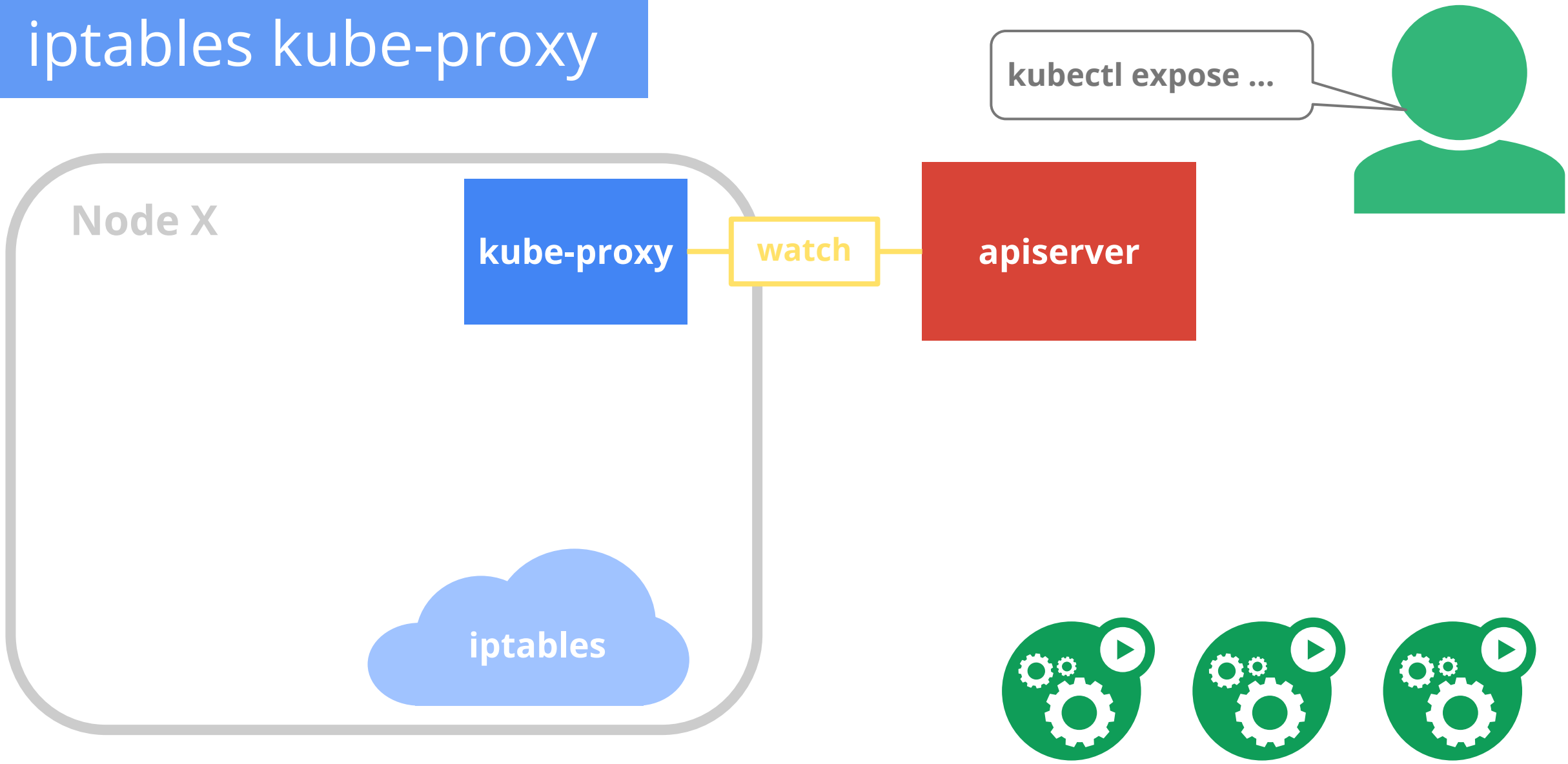
iptables kube-proxy



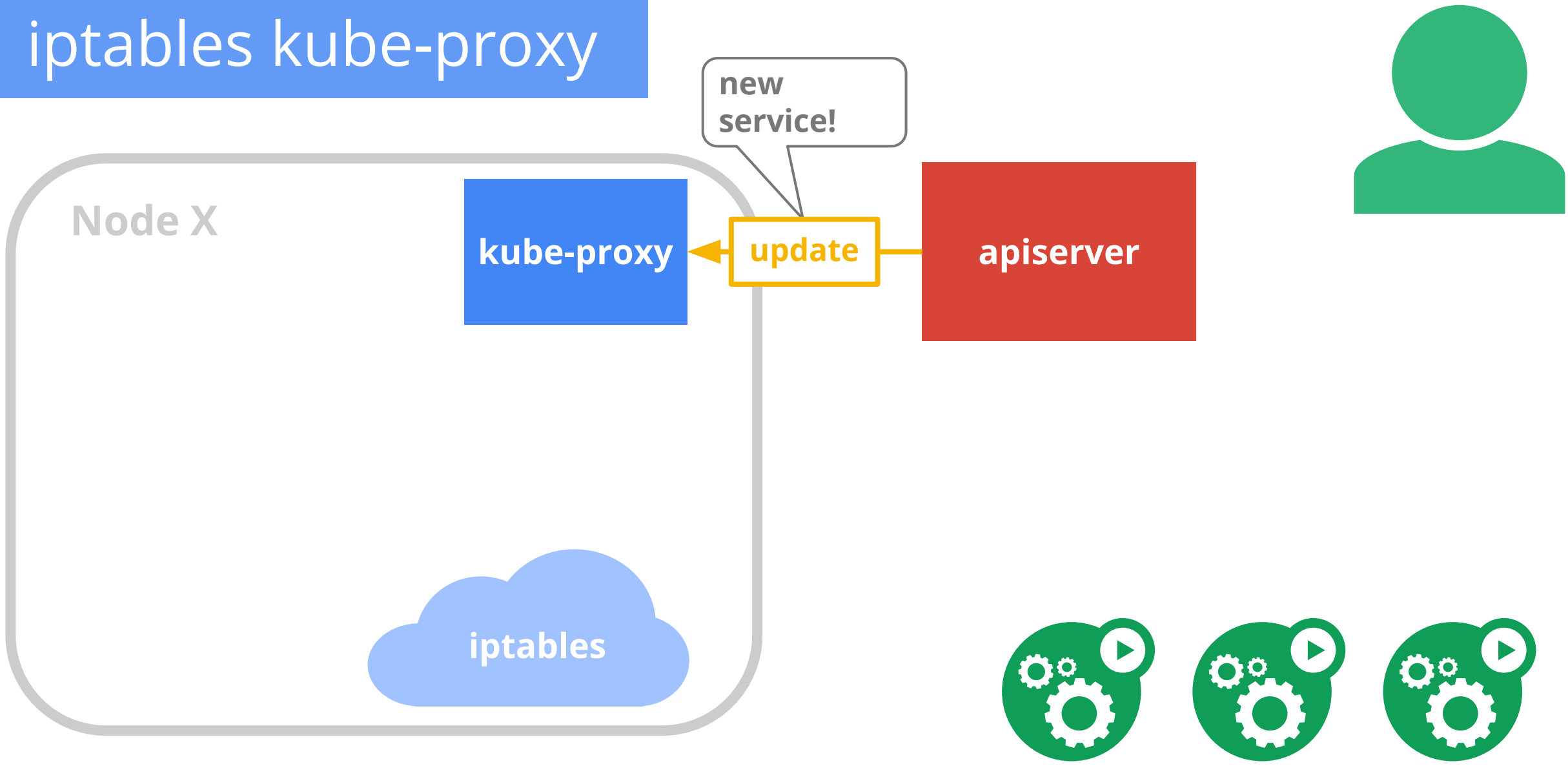
iptables kube-proxy



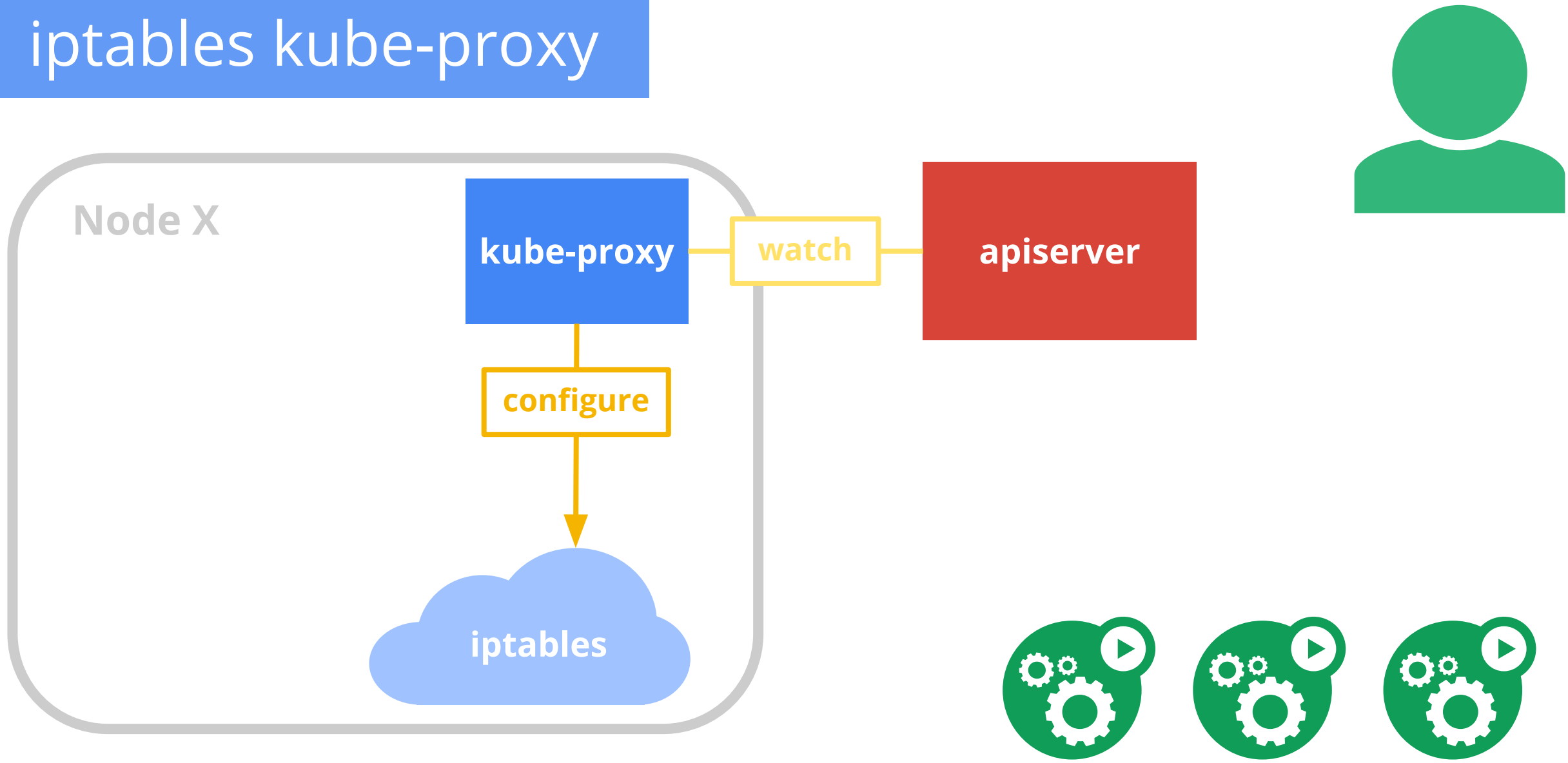
iptables kube-proxy



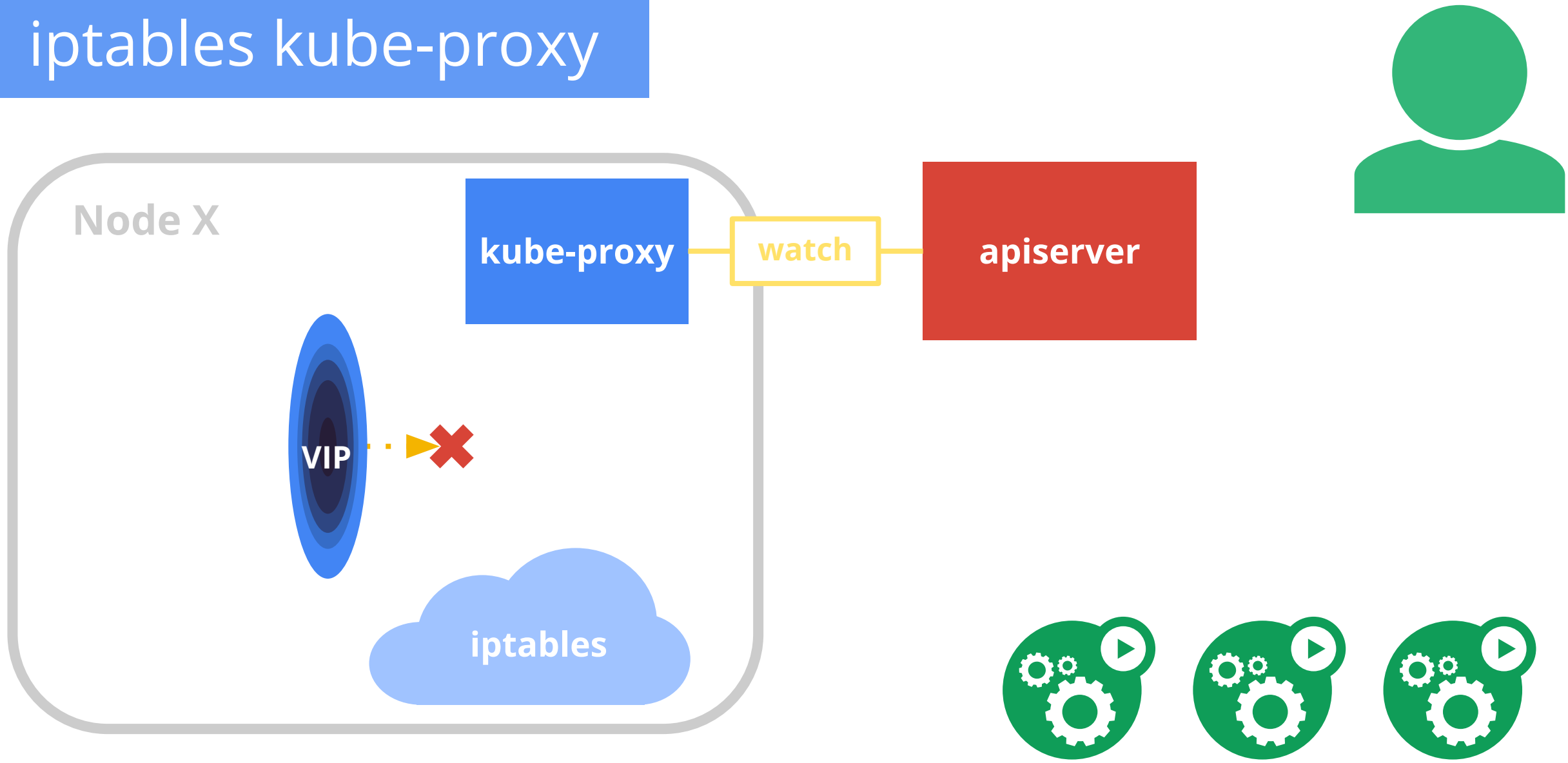
iptables kube-proxy



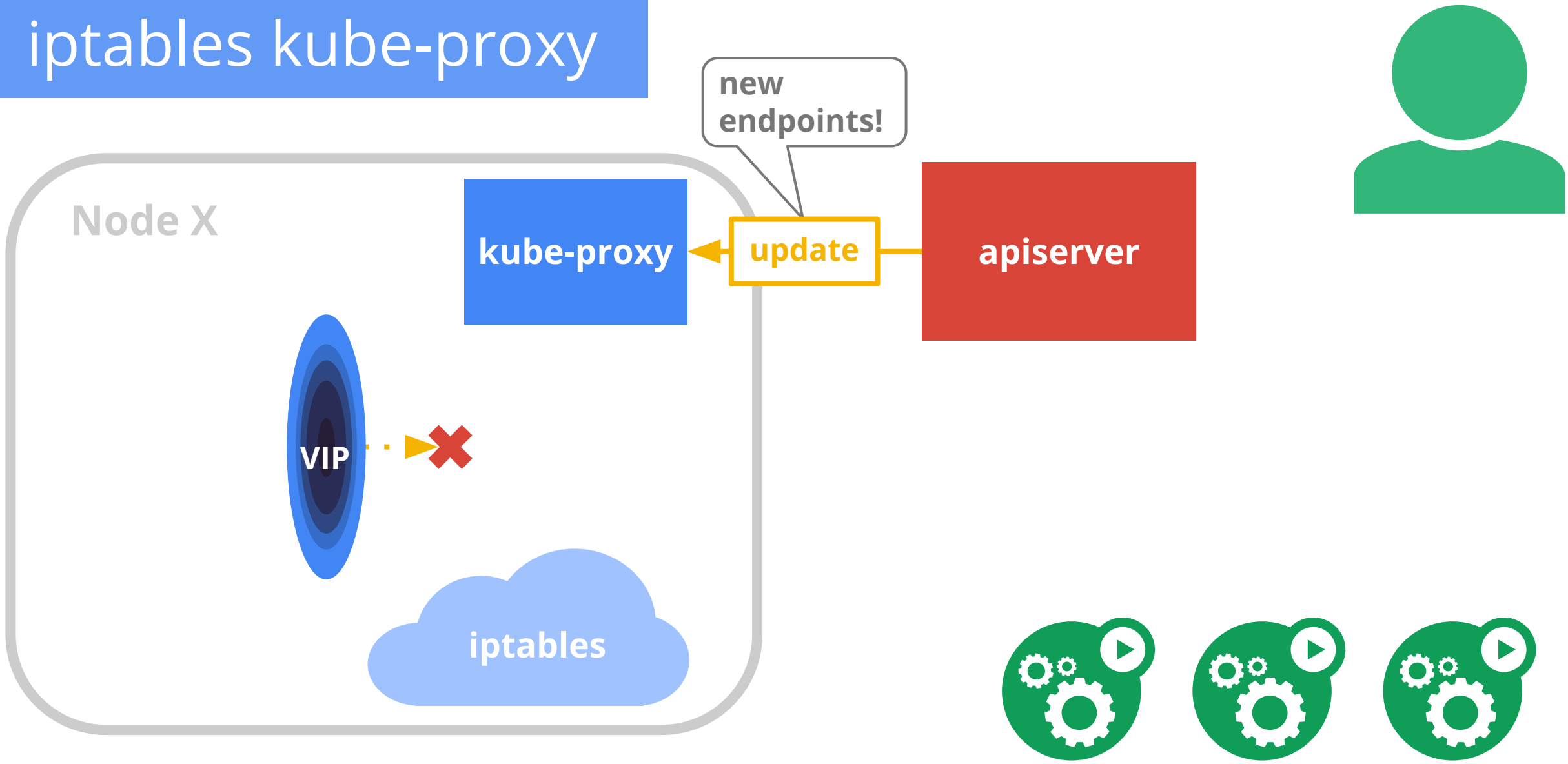
iptables kube-proxy



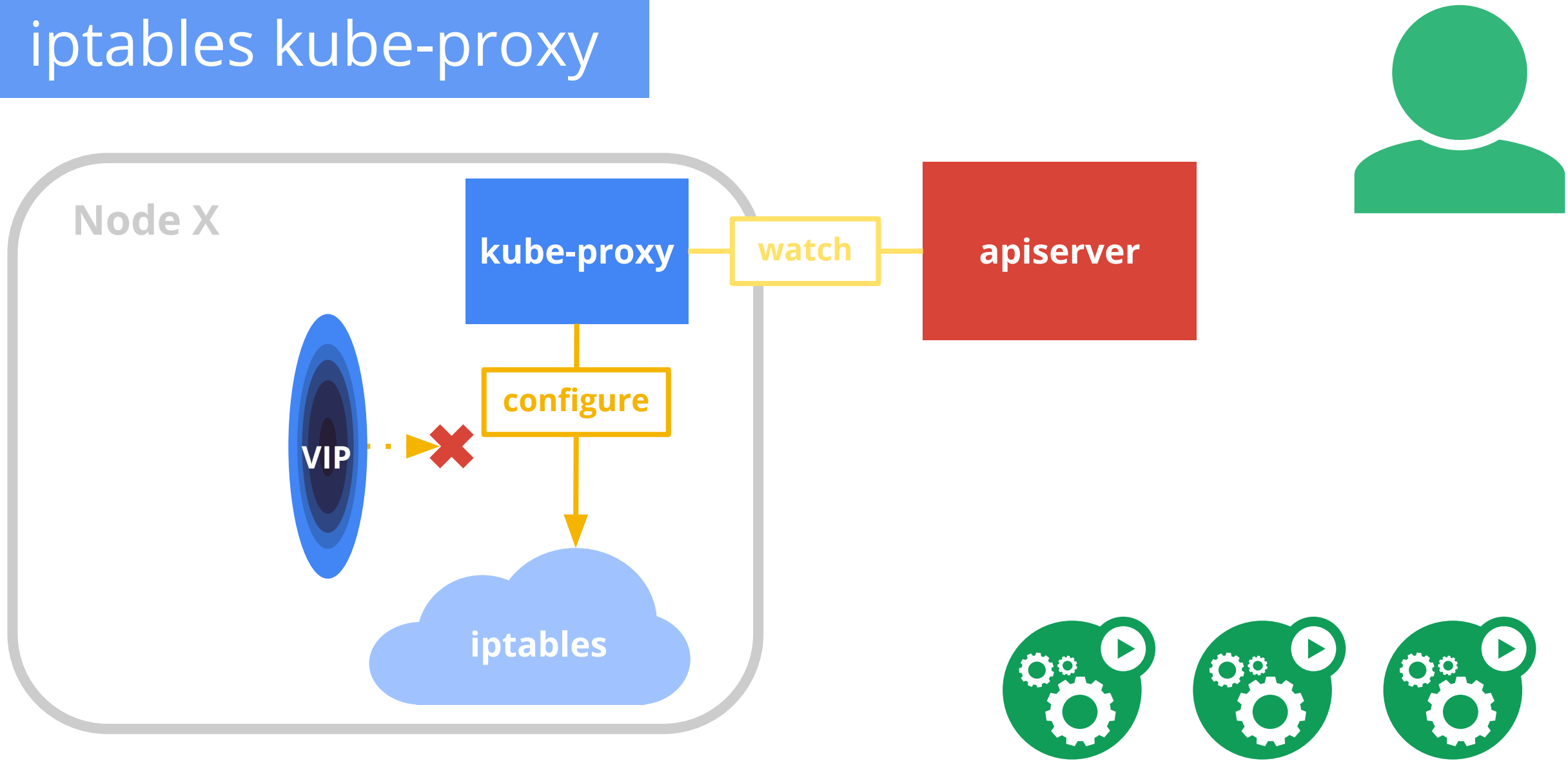
iptables kube-proxy



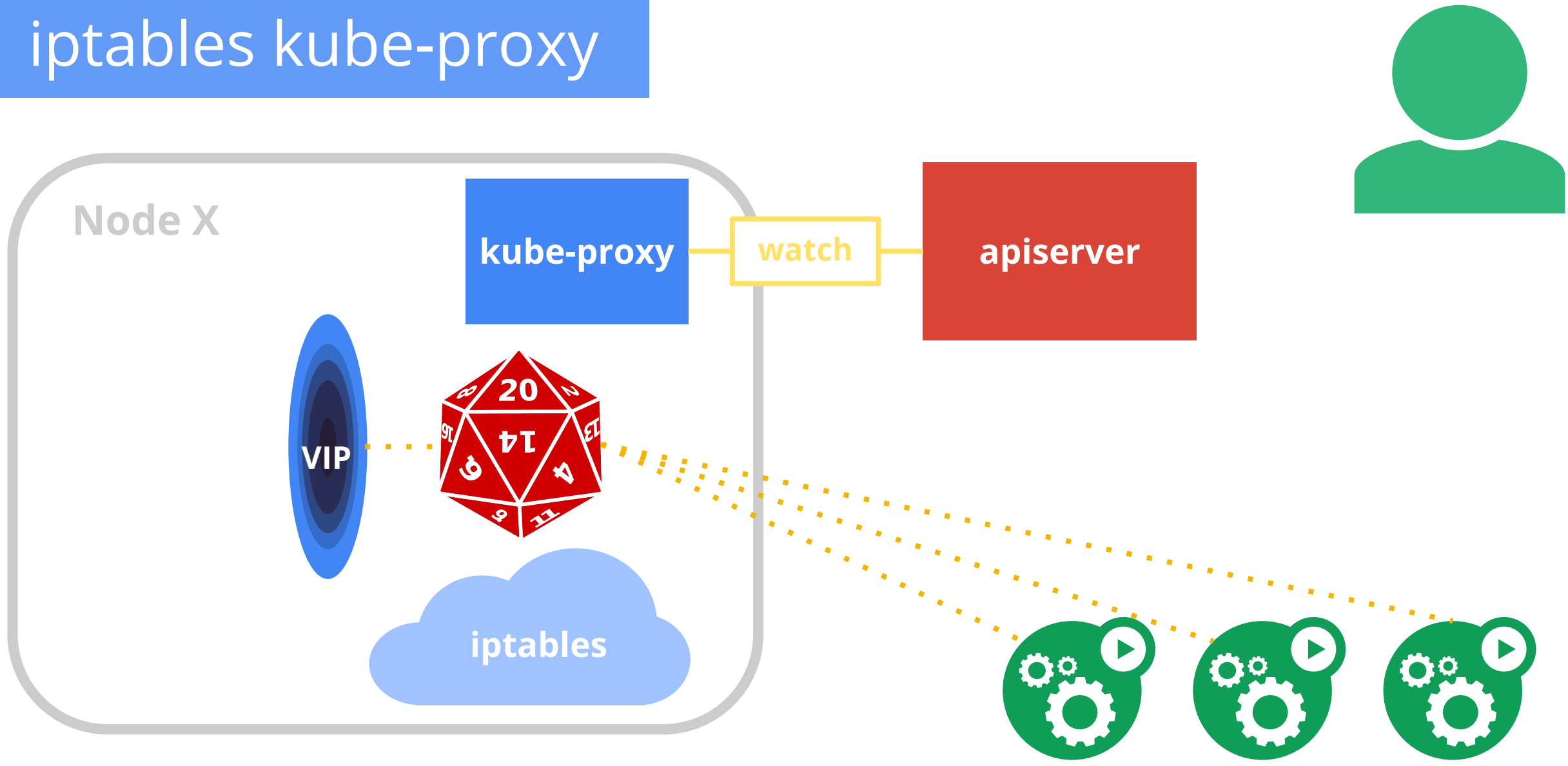
iptables kube-proxy



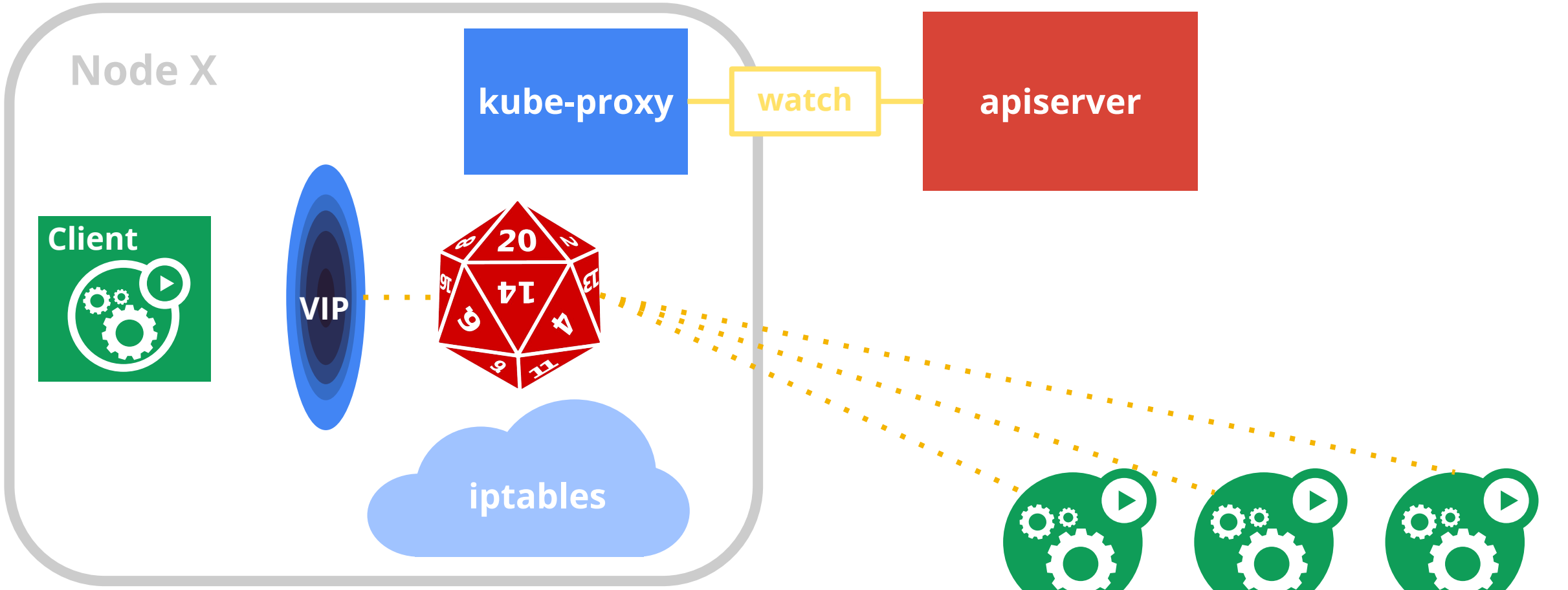
iptables kube-proxy



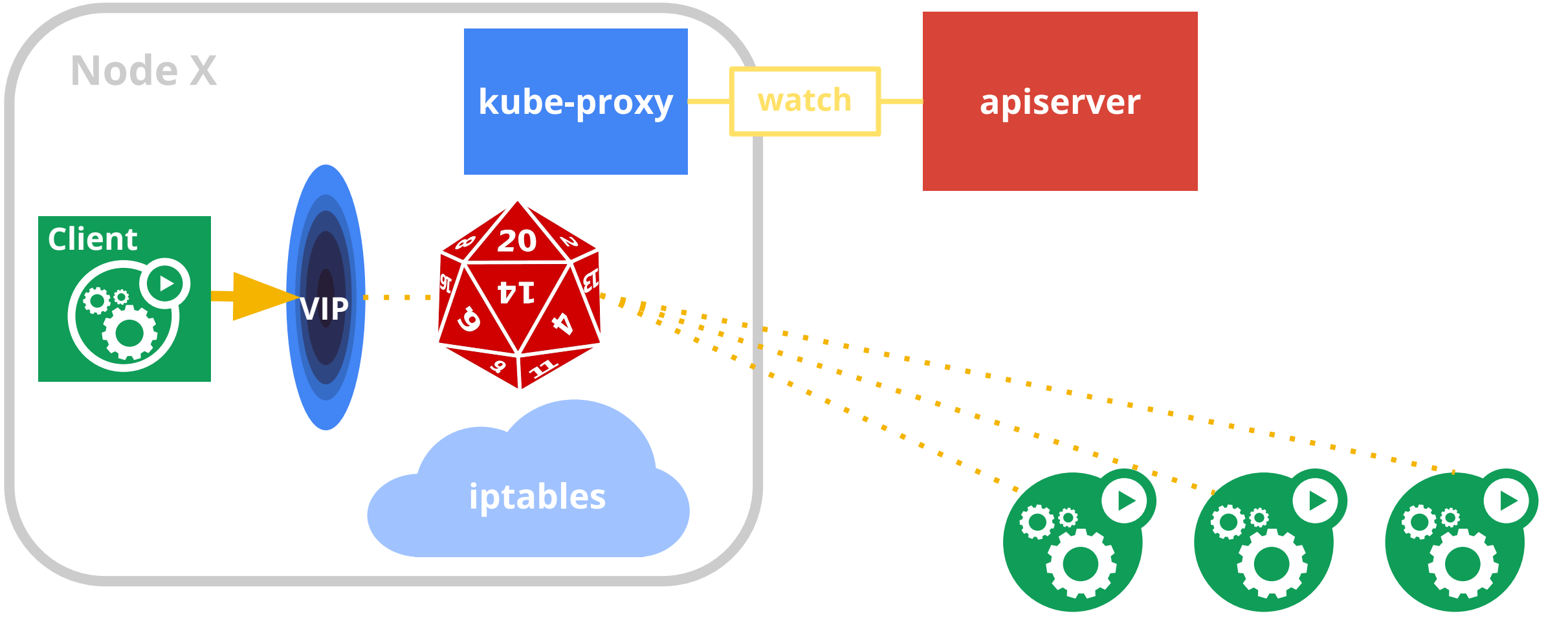
iptables kube-proxy



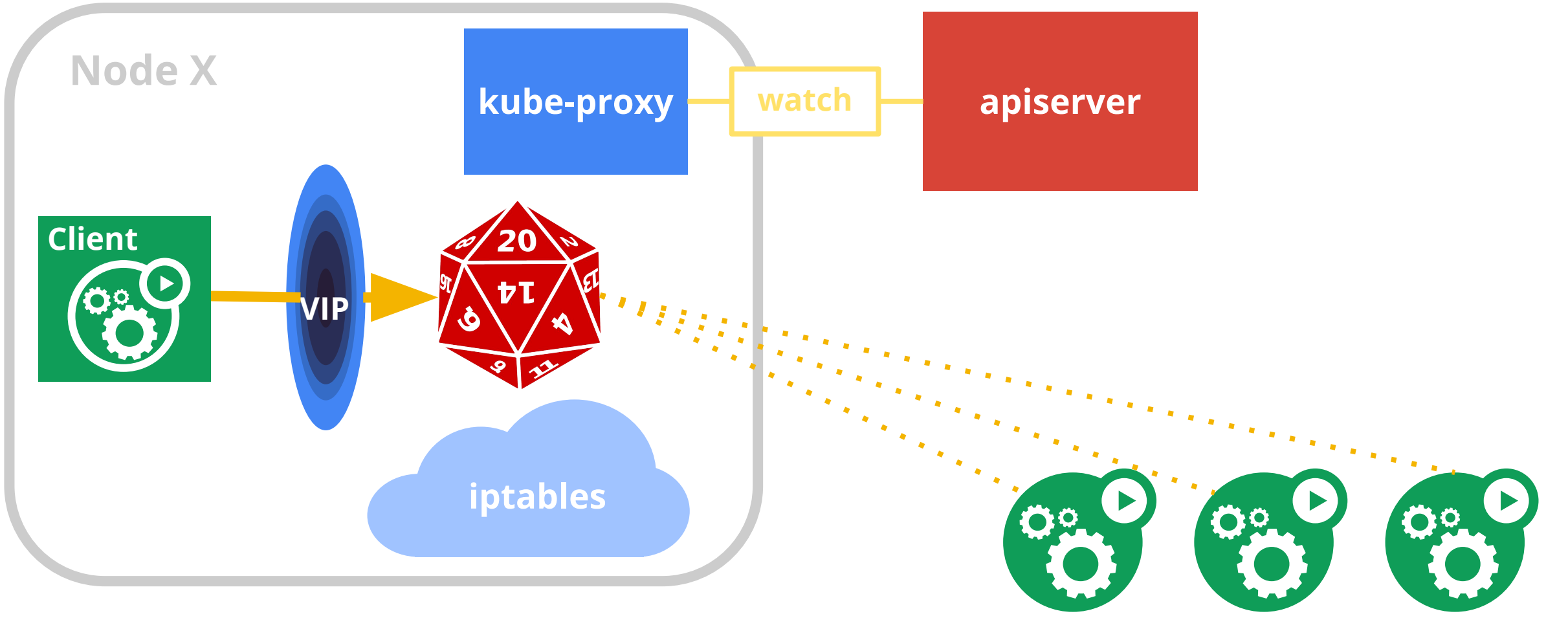
iptables kube-proxy



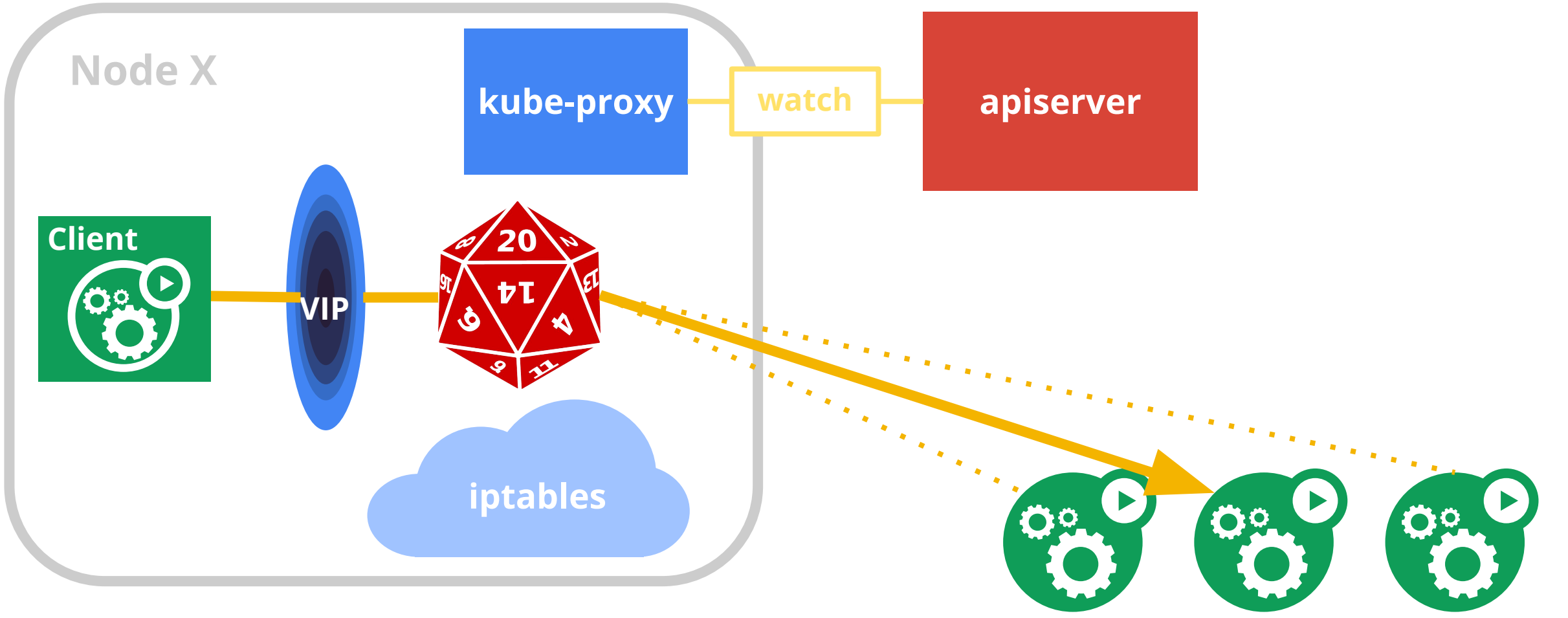
iptables kube-proxy



iptables kube-proxy



iptables kube-proxy



Ingress (L7)

Services are assumed L3/L4

Lots of apps want HTTP/HTTPS

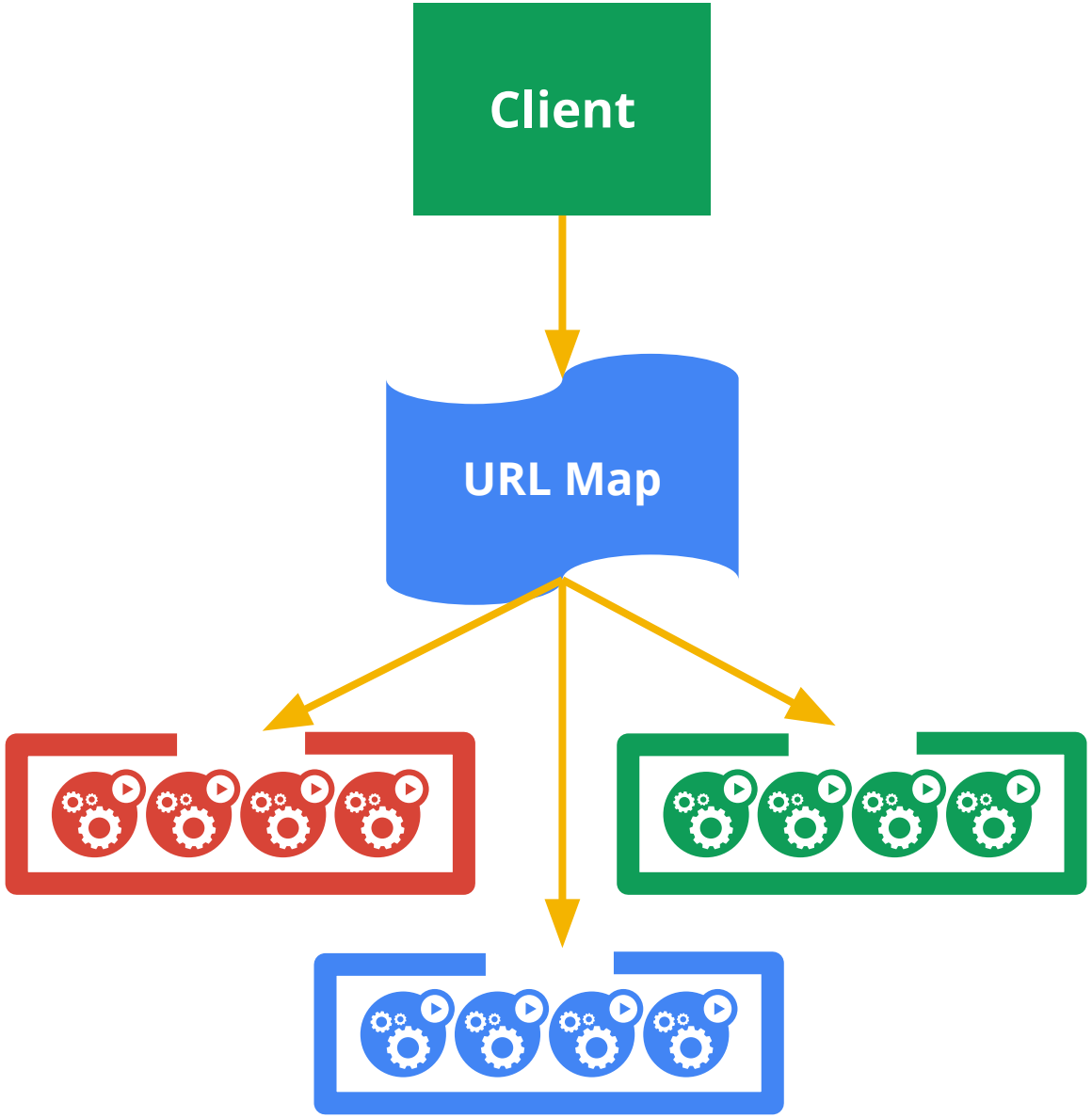
Ingress maps incoming traffic to backend services

- by HTTP host headers
- by HTTP URL paths

HAProxy and GCE implementations

No SSL yet

Status: **BETA** in Kubernetes v1.1



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

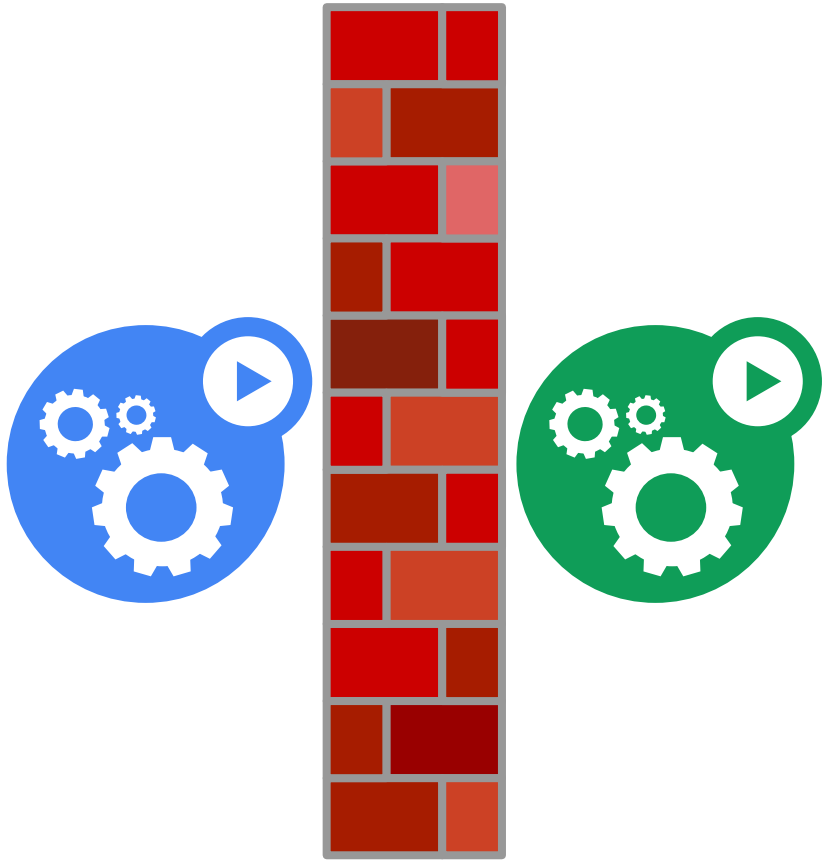
- create new Namespaces as needed
  - per-user, per-app, per-department, etc.
- part of the API - NOT private machines
- most API objects are namespaced
  - part of the REST URL path
- Namespaces are just another API object
- One-step cleanup - delete the Namespace
- Obvious hook for policy enforcement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

Resource Isolation

Resource Isolation

Principles:

- Apps must not be able to affect each other's perf
  - 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

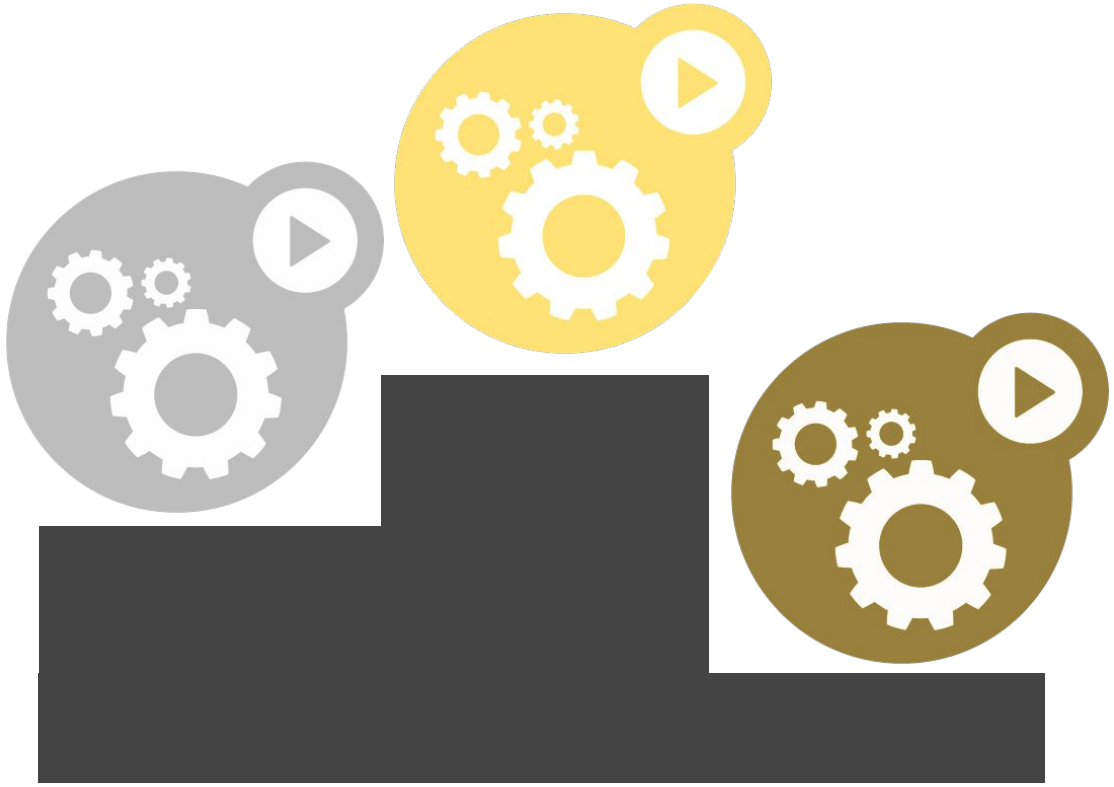
- `request > 0 && limit > request`

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

Introduced in Kubernetes v1.0

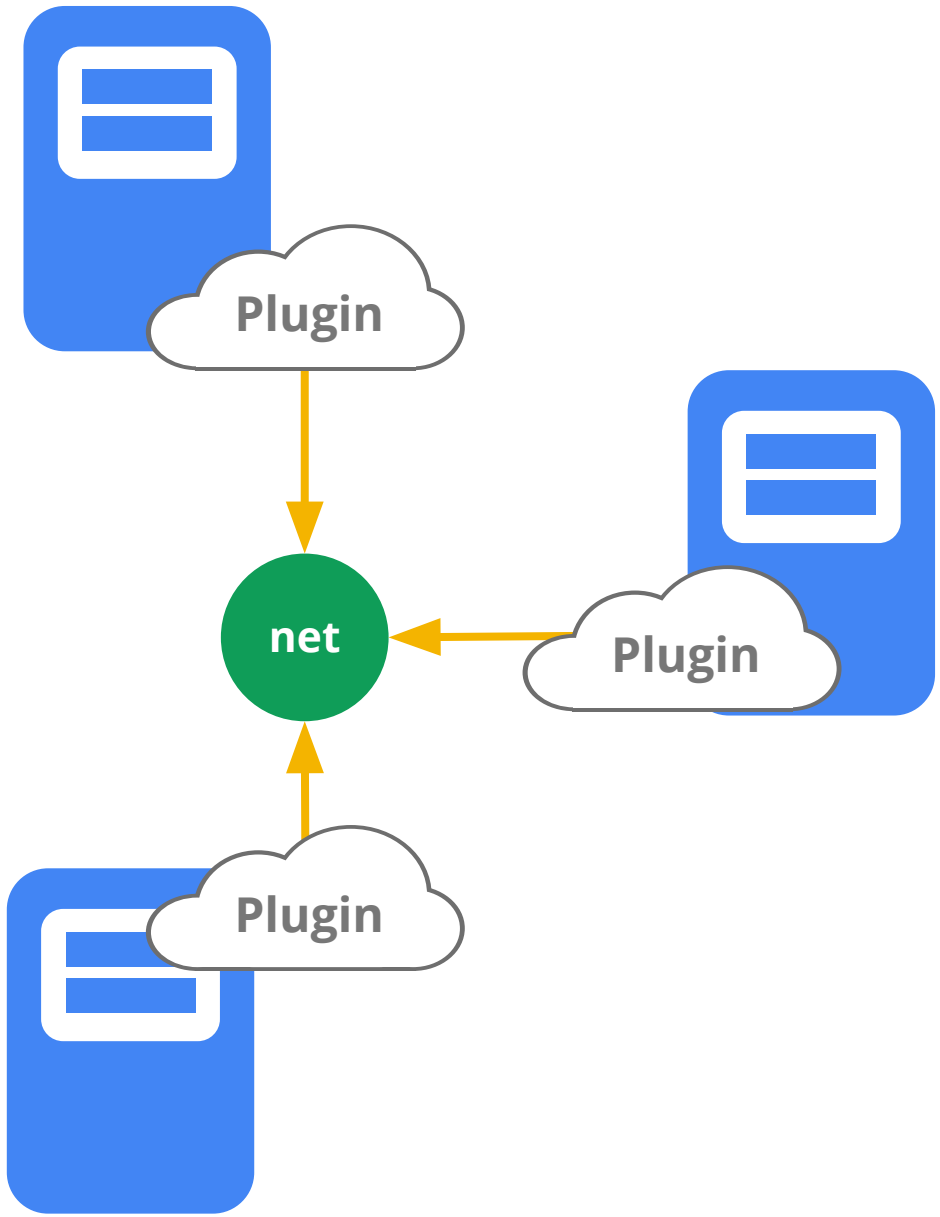
- VERY experimental

Uses **CNI** (CoreOS) in v1.1

- Simple exec interface
- Not using Docker libnetwork
  - but can defer to Docker for networking

Cluster admins can customize their installs

- DHCP, MACVLAN, Flannel, custom



Cluster Auto-Scaling

Cluster Scaling

Add nodes when needed

- e.g. CPU usage too high
- nodes self-register with API server

Remove nodes when not needed

- e.g. CPU usage too low

Status: Works on GCE, need other implementations

DaemonSets

DaemonSets

Problem: how to run a Pod on every node

- or a subset of nodes

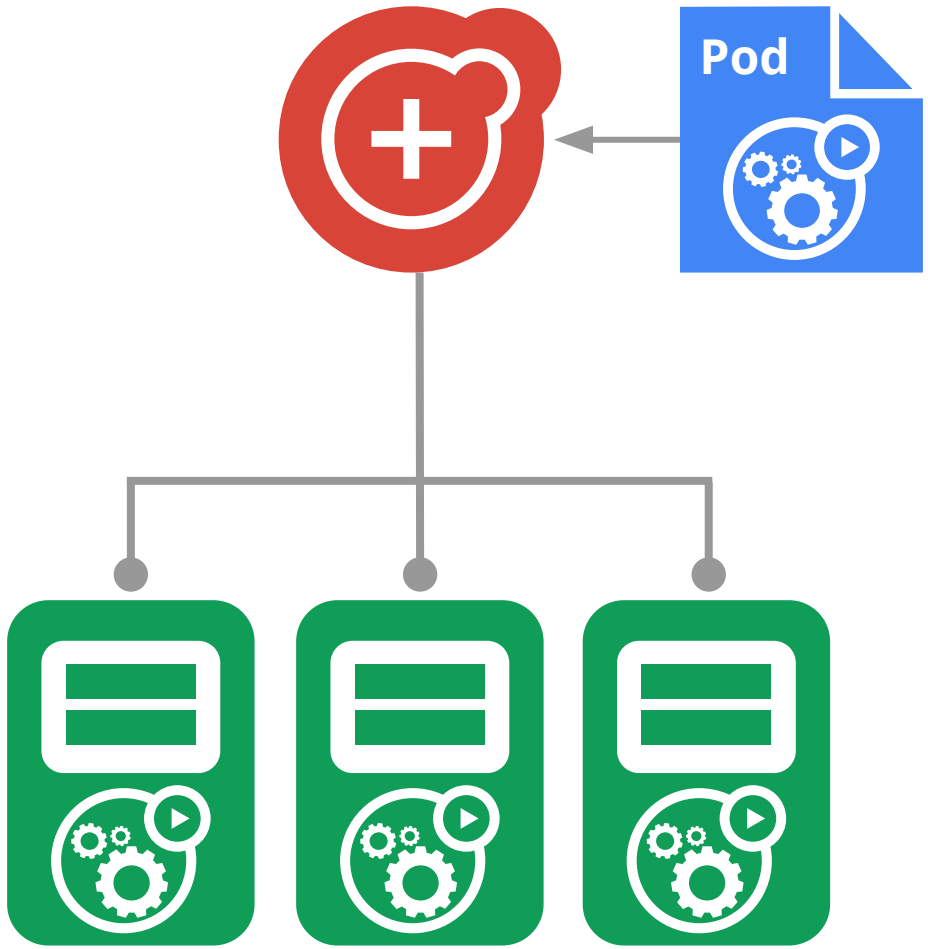
Similar to ReplicationController

- principle: do one thing, don't overload

"Which nodes?" is a selector

Use familiar tools and patterns

Status: **ALPHA** in Kubernetes v1.1



PersistentVolumes

PersistentVolumes

A higher-level abstraction

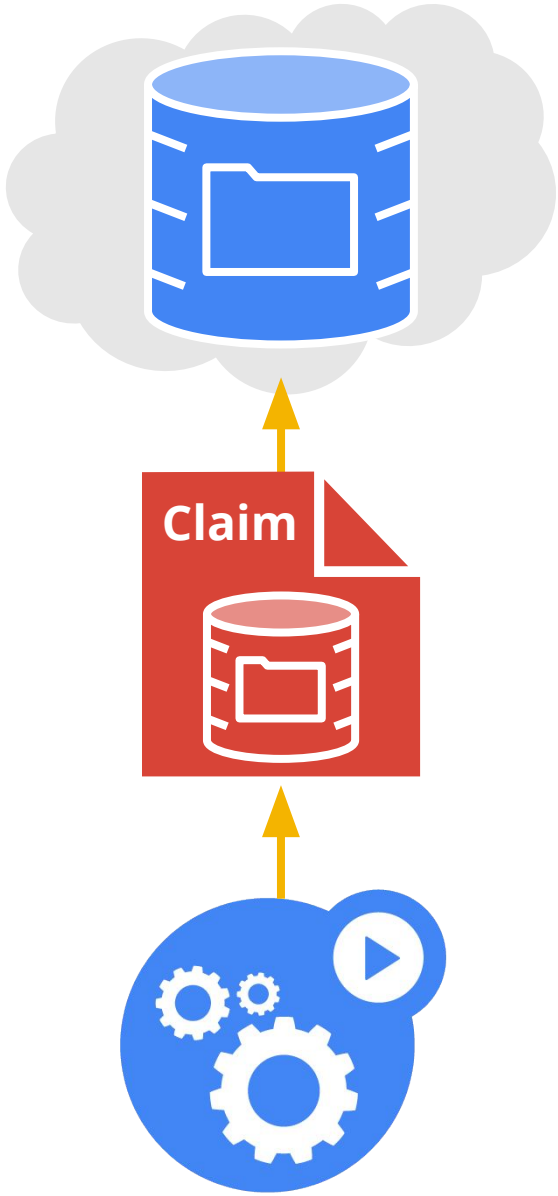
- insulation from any one cloud environment

Admin provisions them, users **claim** them

Independent lifetime and fate

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

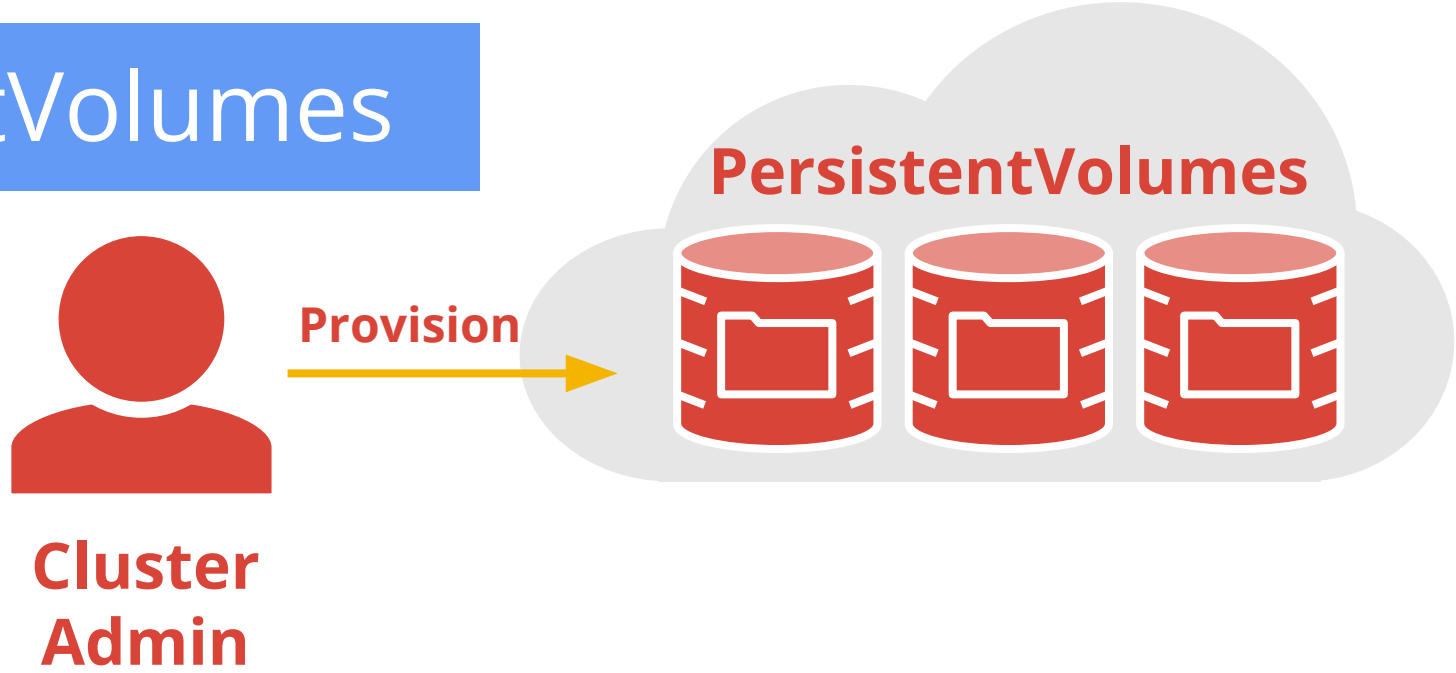
Dynamically “scheduled” and managed, like nodes and pods



PersistentVolumes

Cluster Admin

PersistentVolumes

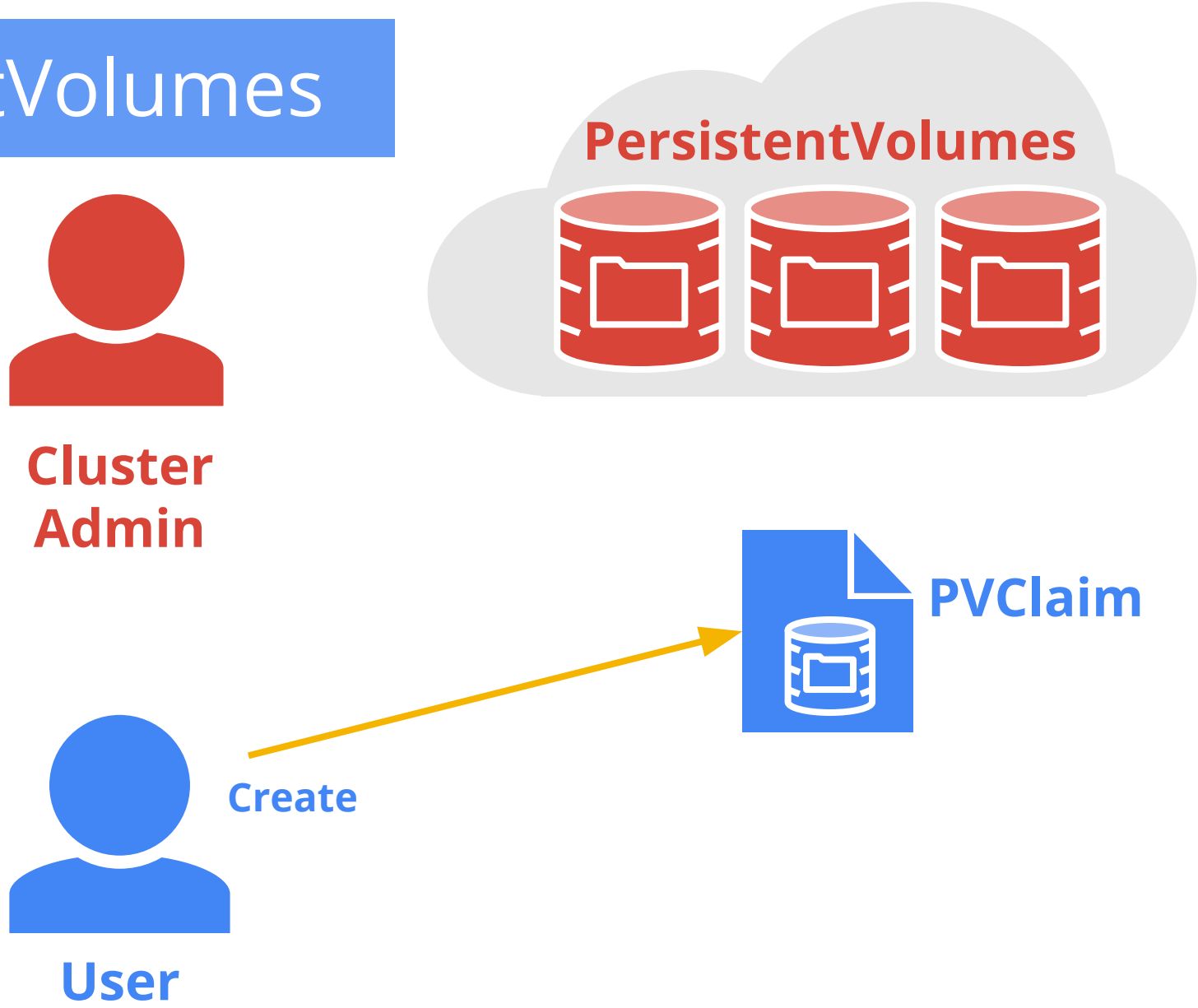


PersistentVolumes

Cluster Admin

User

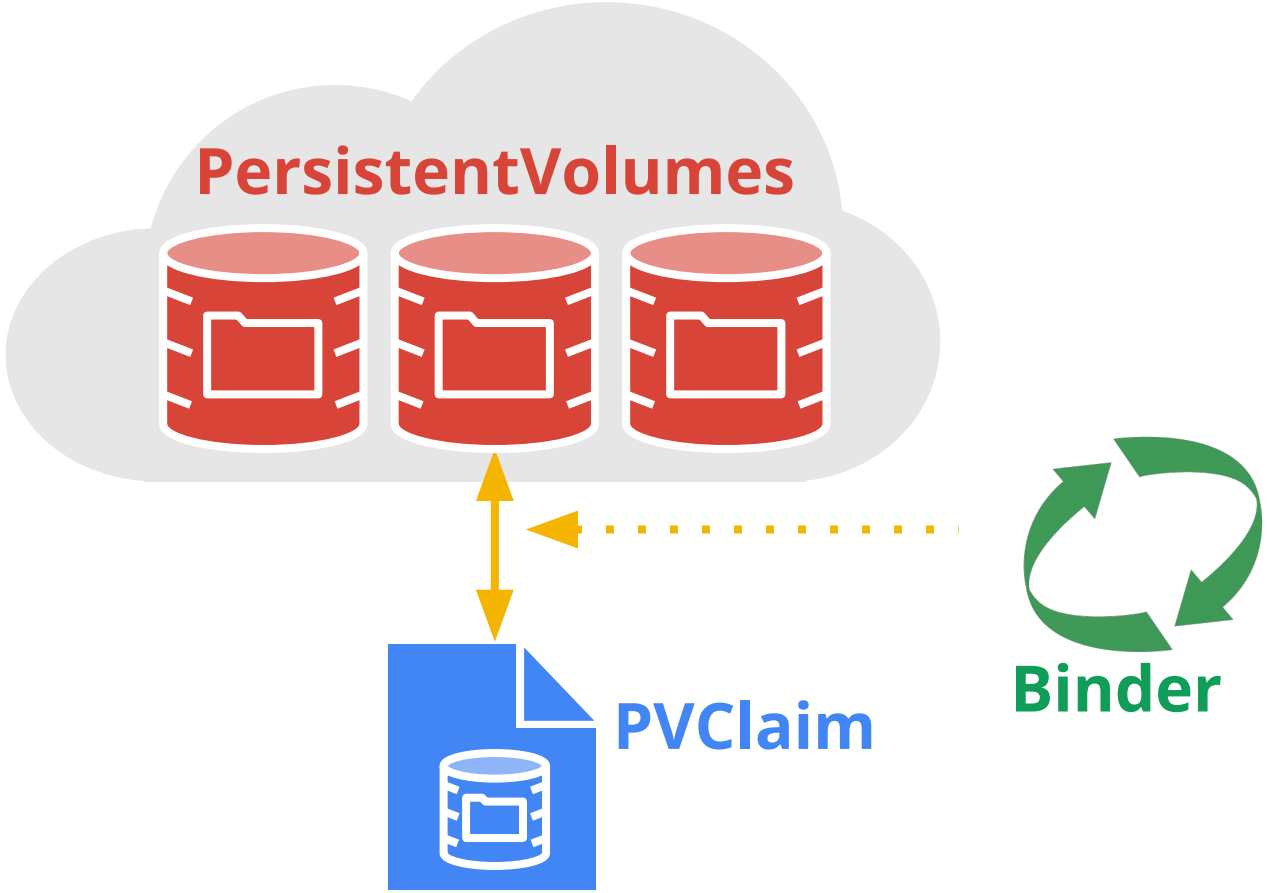
PersistentVolumes



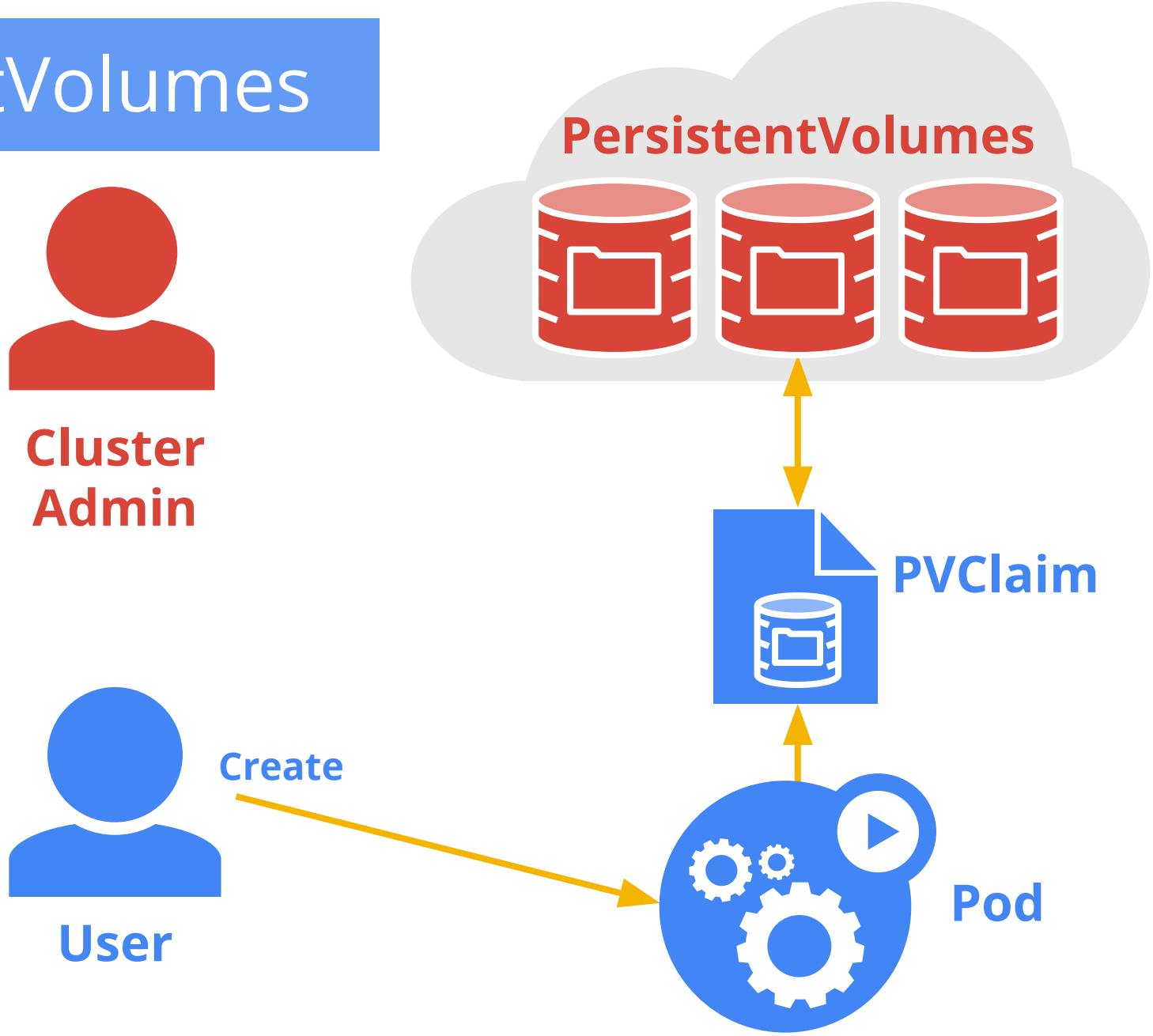
PersistentVolumes

Cluster Admin

User



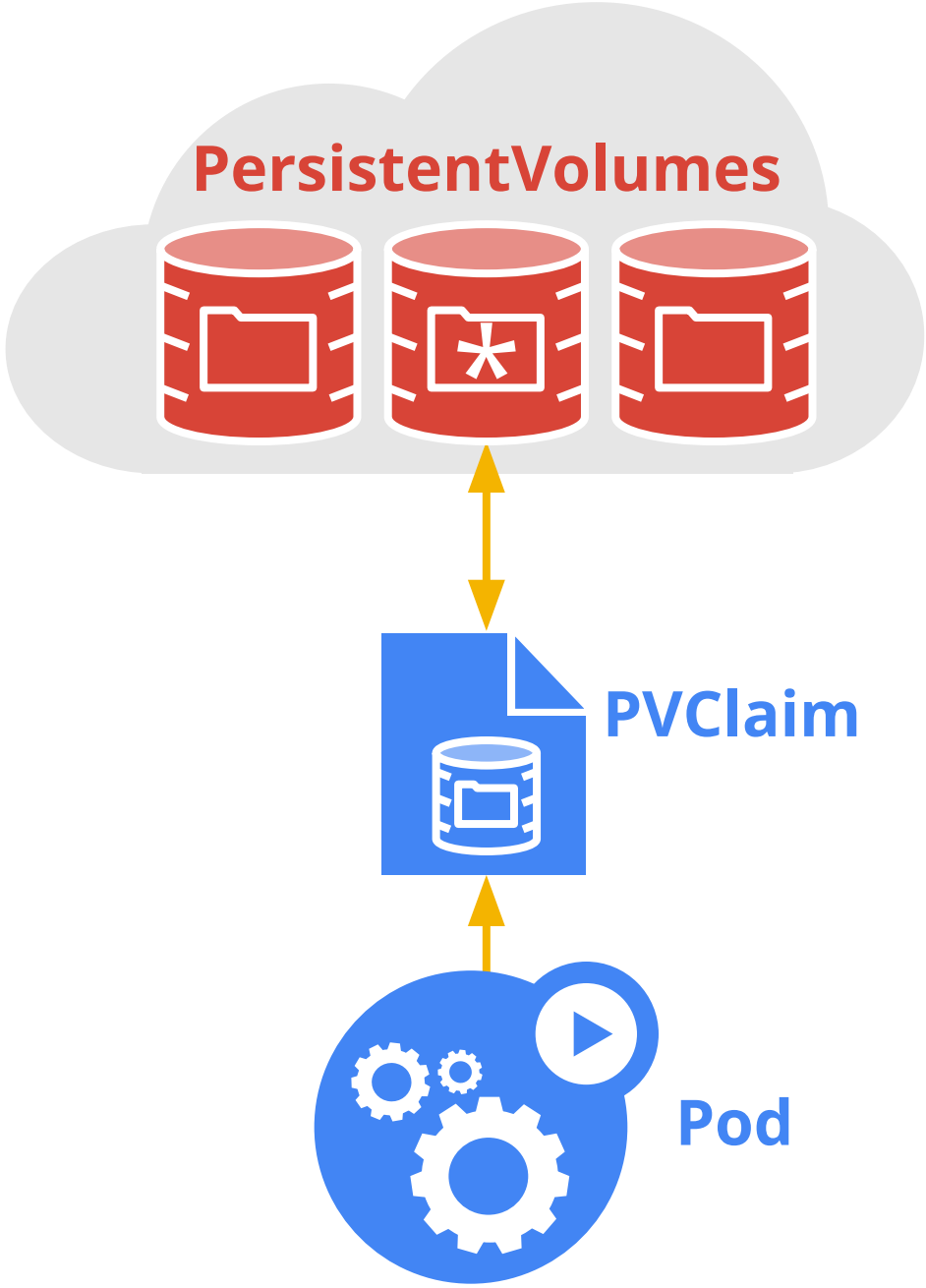
PersistentVolumes



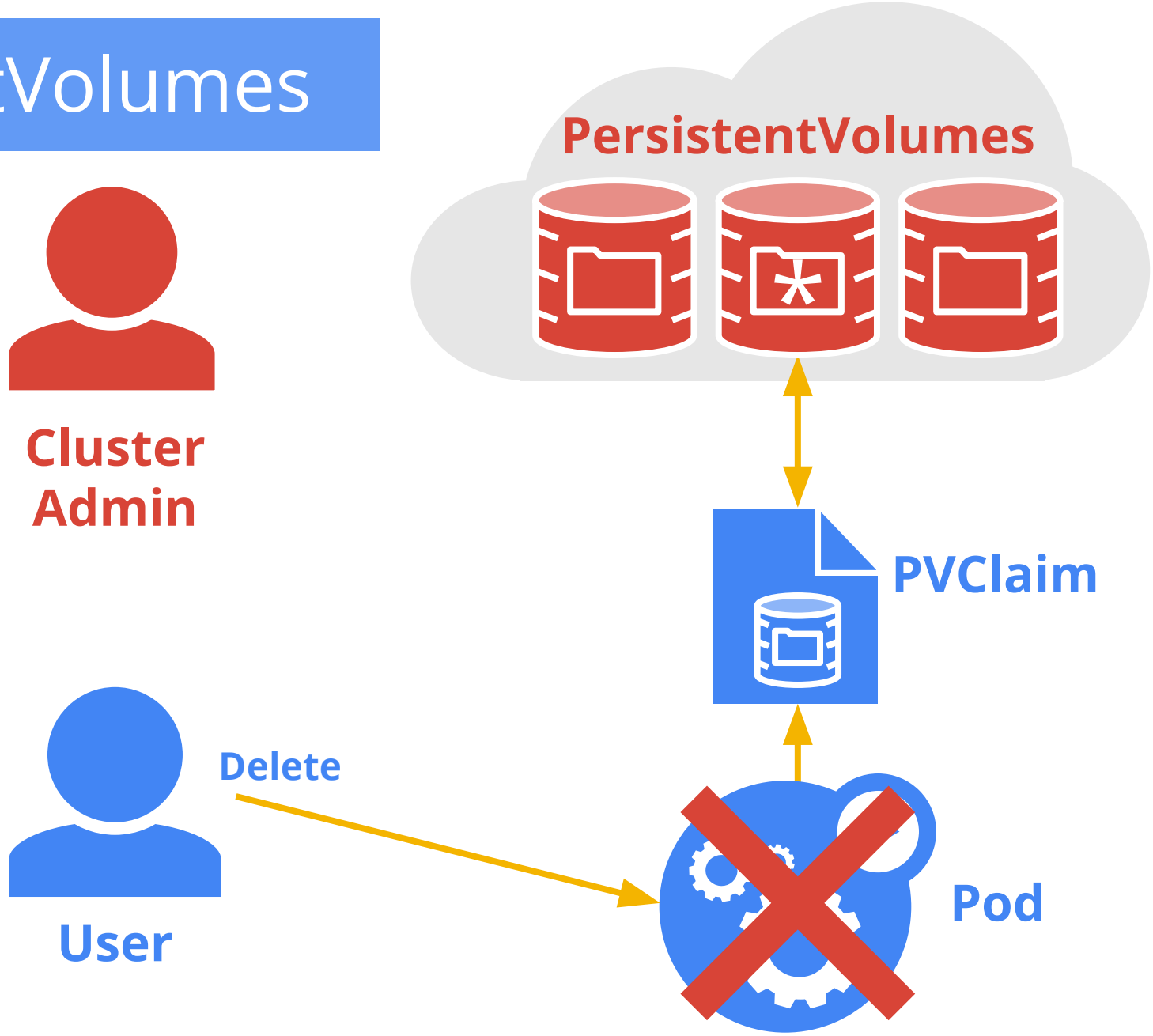
PersistentVolumes

Cluster Admin

User



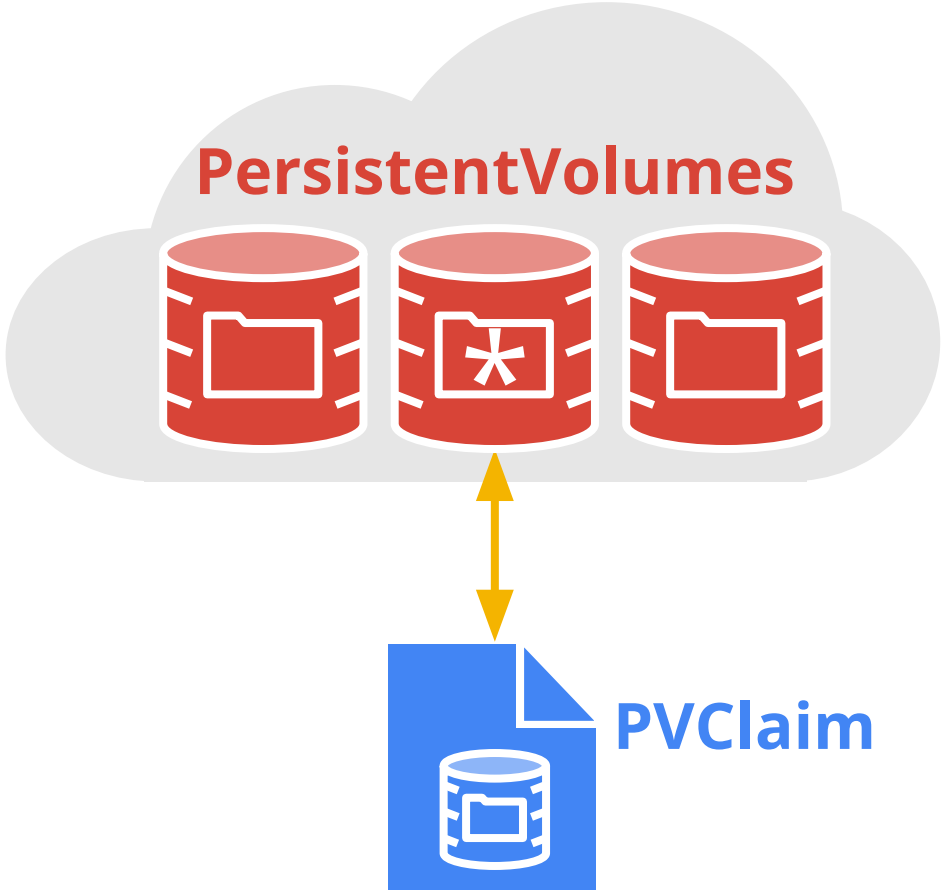
PersistentVolumes



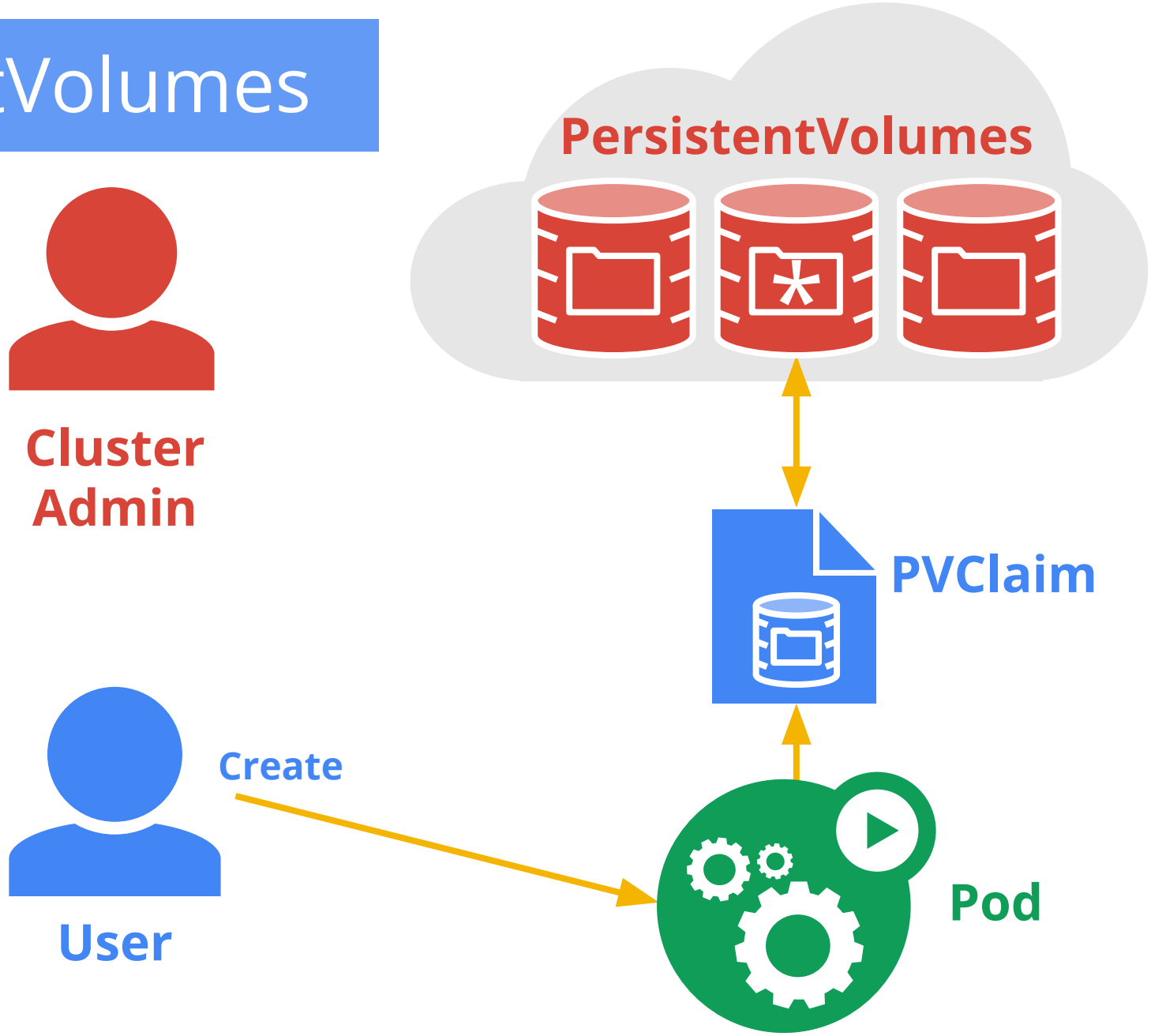
PersistentVolumes

Cluster Admin

User



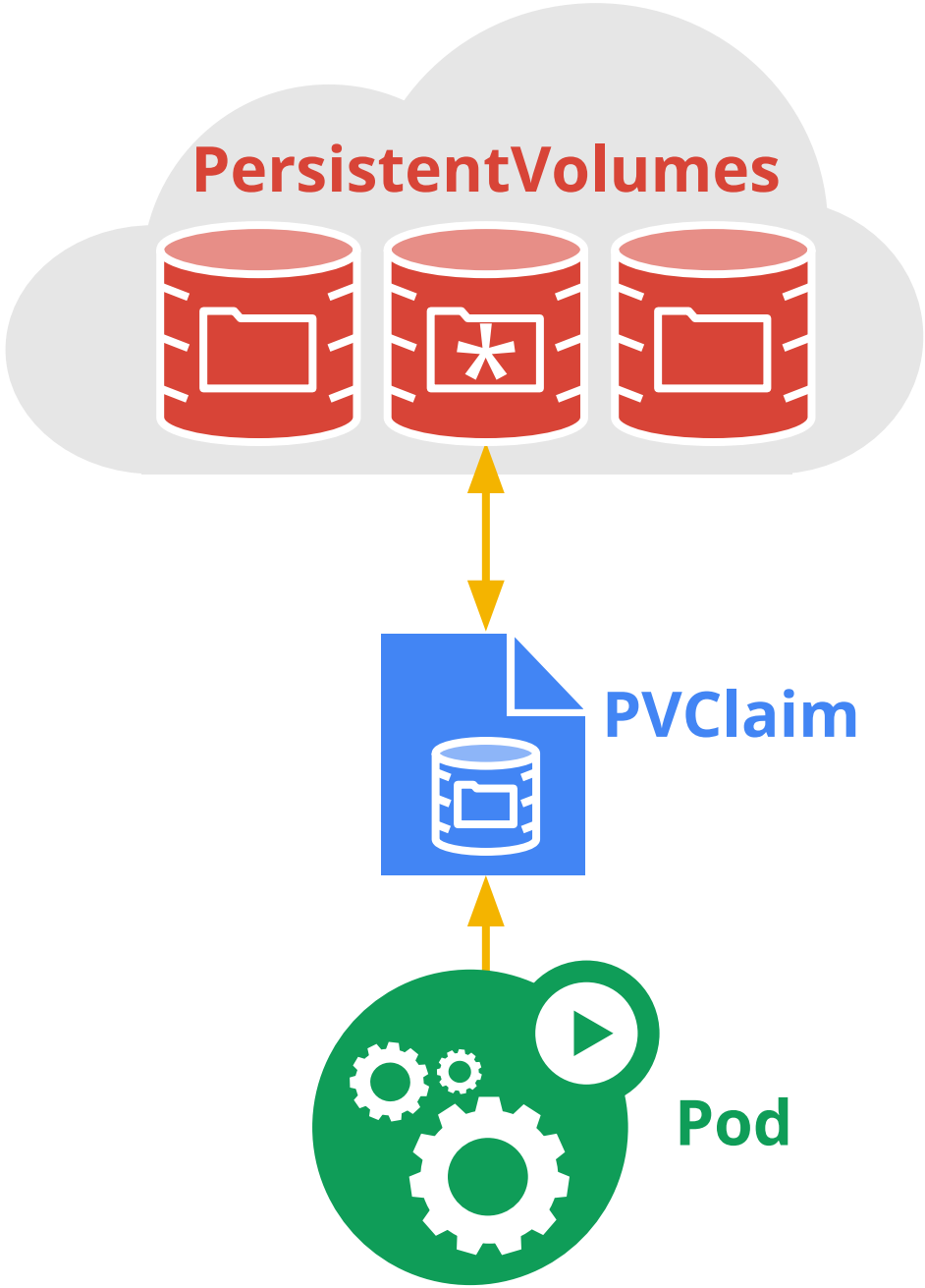
PersistentVolumes



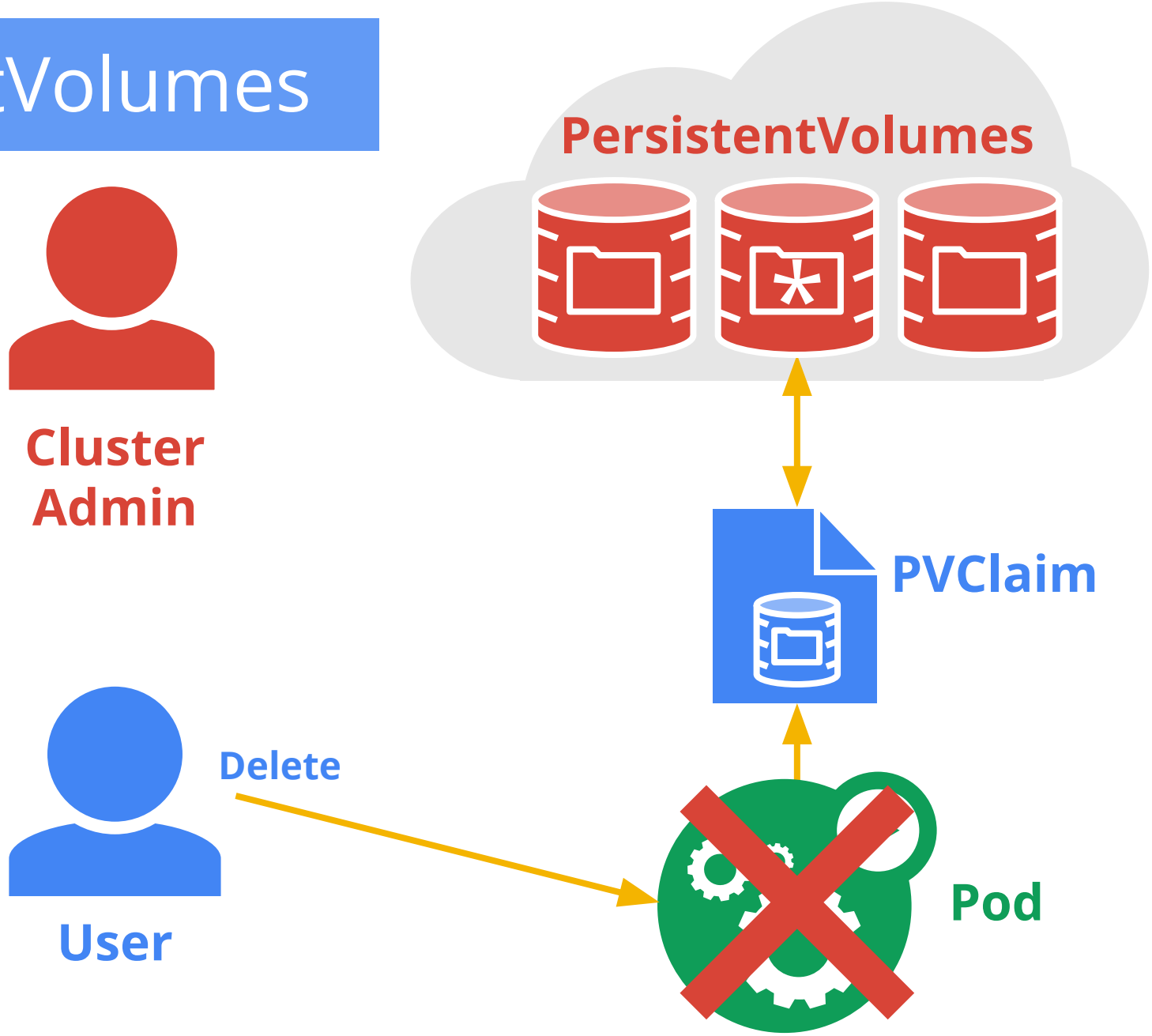
PersistentVolumes

Cluster Admin

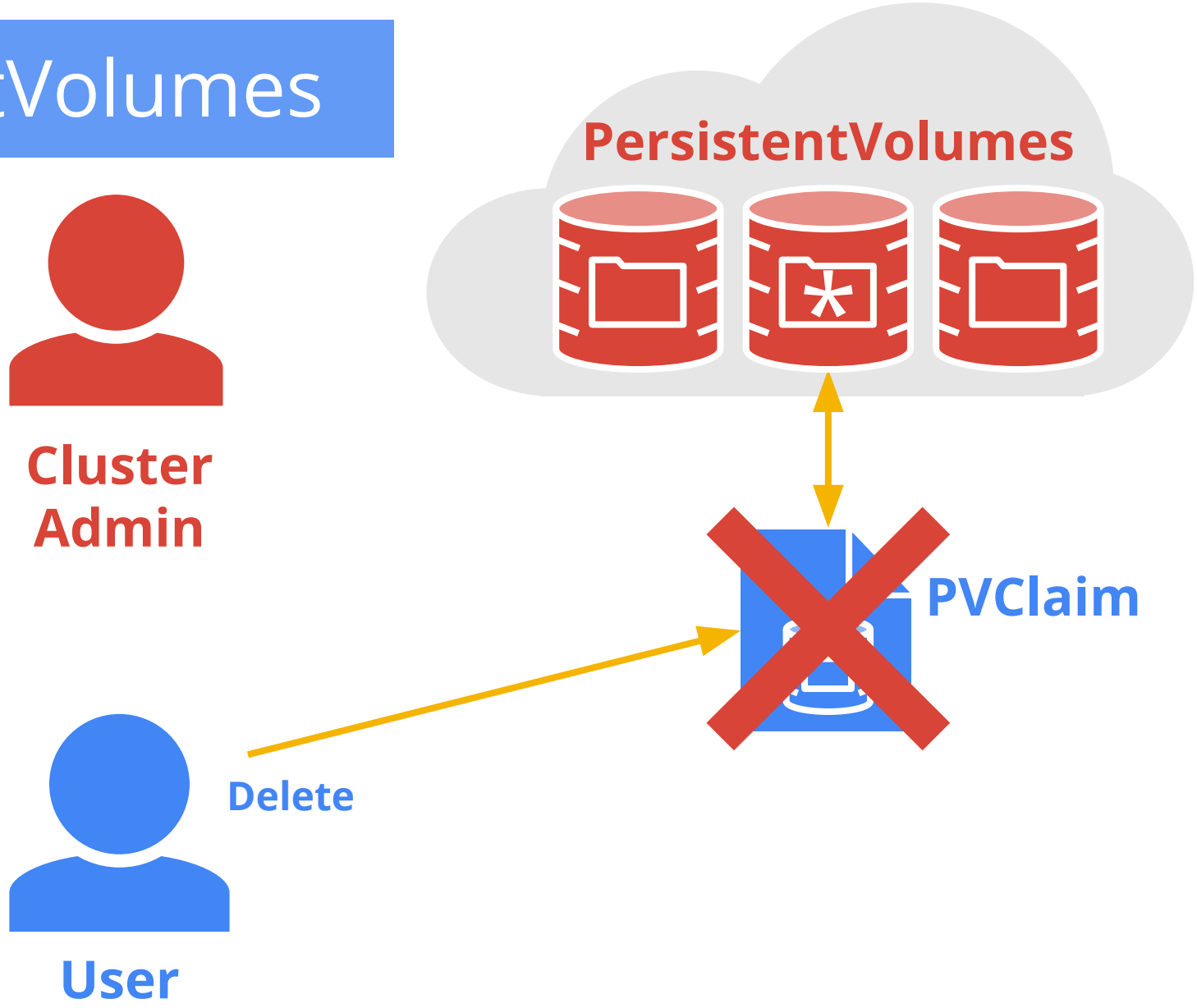
User



PersistentVolumes



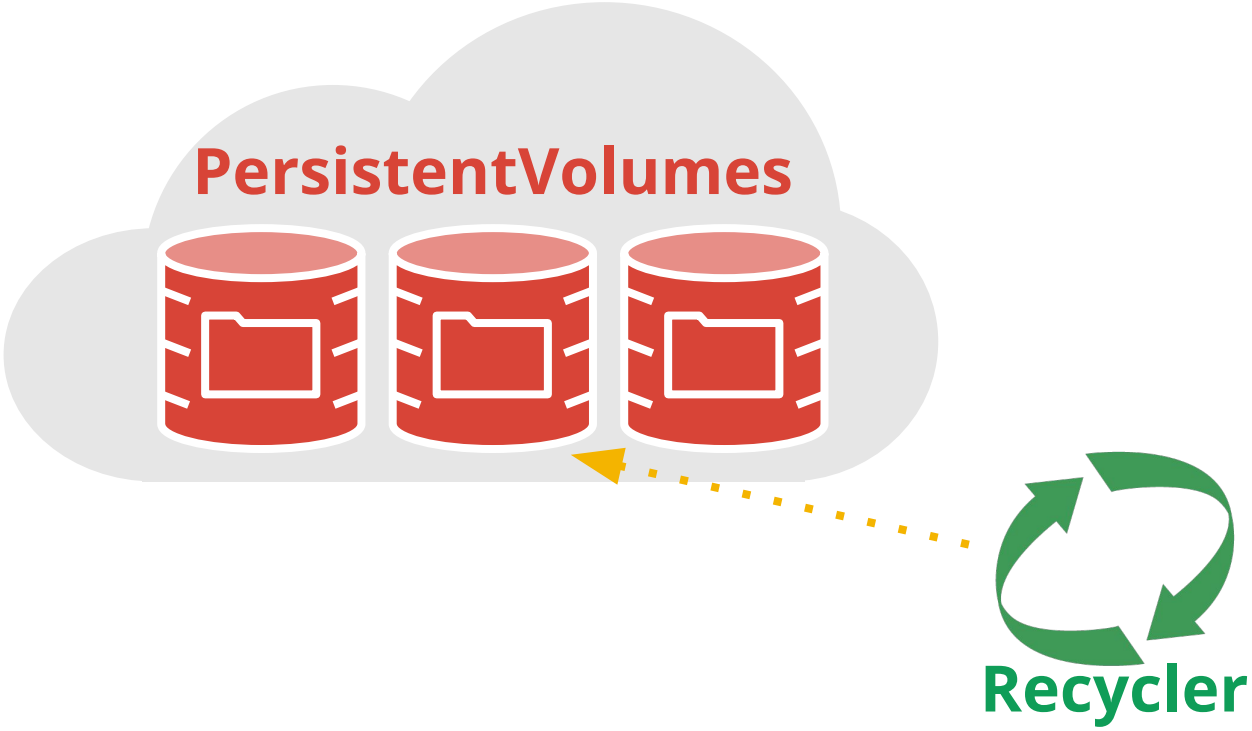
PersistentVolumes



PersistentVolumes

Cluster Admin

User



Secrets

Secrets

Problem: how to grant a pod access to a secured *something?*

- don't put secrets in the container image!

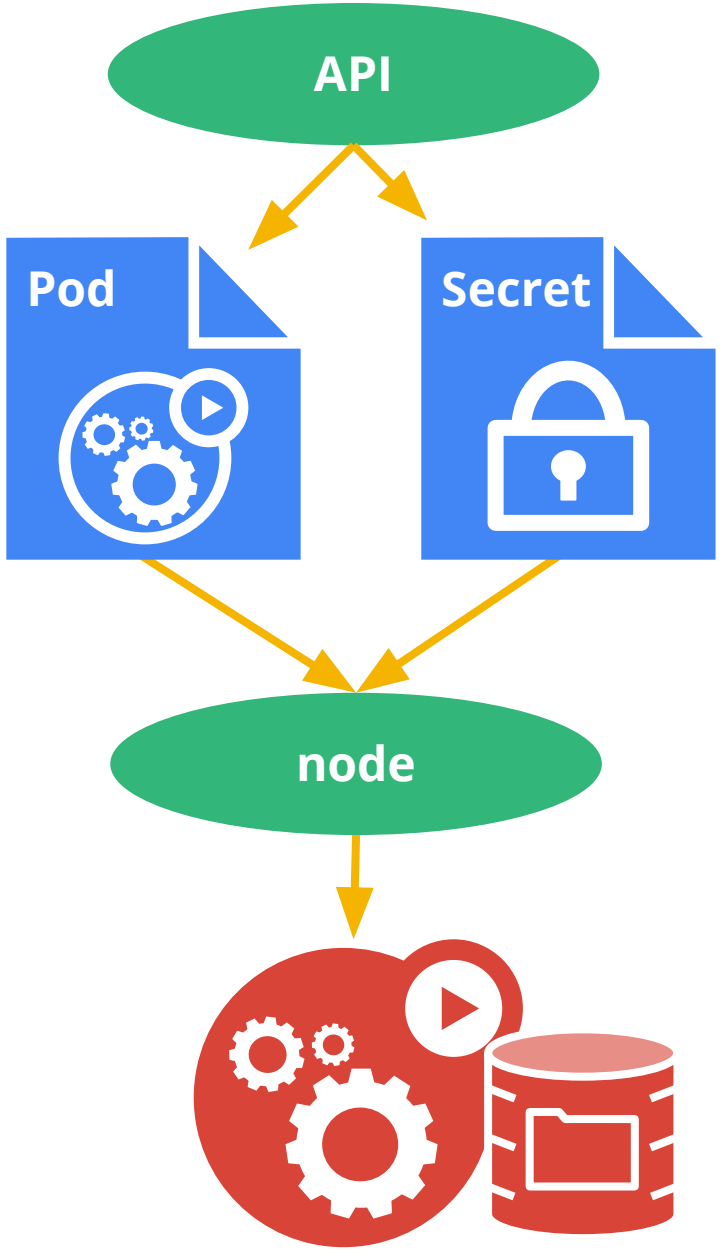
12-factor says: config comes from the environment

- Kubernetes is the environment

Manage secrets via the Kubernetes API

Inject them as virtual volumes into Pods

- late-binding
- tmpfs - never touches disk



Cluster Add-Ons

Monitoring

Run cAdvisor on each node (in kubelet)

- gather stats from all containers
- export via REST

Run Heapster as a pod in the cluster

- just another pod, no special access
- aggregate stats

Run Influx and Grafana in the cluster

- more pods
- alternately: store in Google Cloud Monitoring

Or plug in your own!

- e.g. Google Cloud Monitoring

Logging

Run fluentd as a pod on each node

- gather logs from all containers
- export to elasticsearch

Run Elasticsearch as a pod in the cluster

- just another pod, no special access
- aggregate logs

Run Kibana in the cluster

- yet another pod
- alternately: store in Google Cloud Logging

Or plug in your own!

- e.g. Google Cloud Logging

DNS

Run SkyDNS as a pod in the cluster

- kube2sky bridges Kubernetes API -> SkyDNS
- Tell kubelets about it (static service IP)

Strictly optional, but practically required

- LOTS of things depend on it
- Probably will become more integrated

Or plug in your own!

New and coming soon

- Jobs (run-to-completion)
- Cron (scheduled jobs)
- Privileged containers
- Downward API
- Interactive containers
- Bandwidth shaping
- Simpler deployments
- Third-party API objects
- HA master
- Easier to run as non-root
- Scalability++
- Performance++
- Easier installation & setup
- Config injection
- Cluster federation
- More volume types
- Private Docker registry
- External DNS integration
- Volume auto-provisioning
- Node fencing
- DiY Cloud Provider plugins
- Better auth{n,z}
- Network policy
- Big data integrations
- Better affinity policies
- Device scheduling (e.g. GPUs)

Kubernetes status & plans

Open sourced in June, 2014

- v1.0 in July, 2015
- v1.1 in November, 2015

Google Container Engine (GKE)

- hosted Kubernetes - don't think about cluster setup
- GA in August, 2015

PaaSes:

- RedHat OpenShift, Deis, Stratos

Distros:

- CoreOS Tectonic, Mirantis Murano (OpenStack), RedHat Atomic, Mesos

Driving towards a v1.2 release in O(months)

The Goal: Shake things up

Containers are a **new way of working**

Requires new concepts and new tools

Google has a **lot** of experience...

...but we are **listening to the users**

Workload portability is important!

Kubernetes is **Open**

- open community
- open design
- open source
- open to ideas

http://kubernetes.io
https://github.com/kubernetes/kubernetes
slack: kubernetes
twitter: @kubernetesio