Kubernetes and Container Management

Container Camp 2015

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

Google has been developing and using containers to manage our applications for **over 10 years.**

Images by Connie Zhou

Everything at Google runs in containers:

- Gmail, Web Search, Maps, ...
- MapReduce, batch, ...
- GFS, Colossus, ...
- Even **GCE itself**: 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 **GCE itself**: VMs run in containers

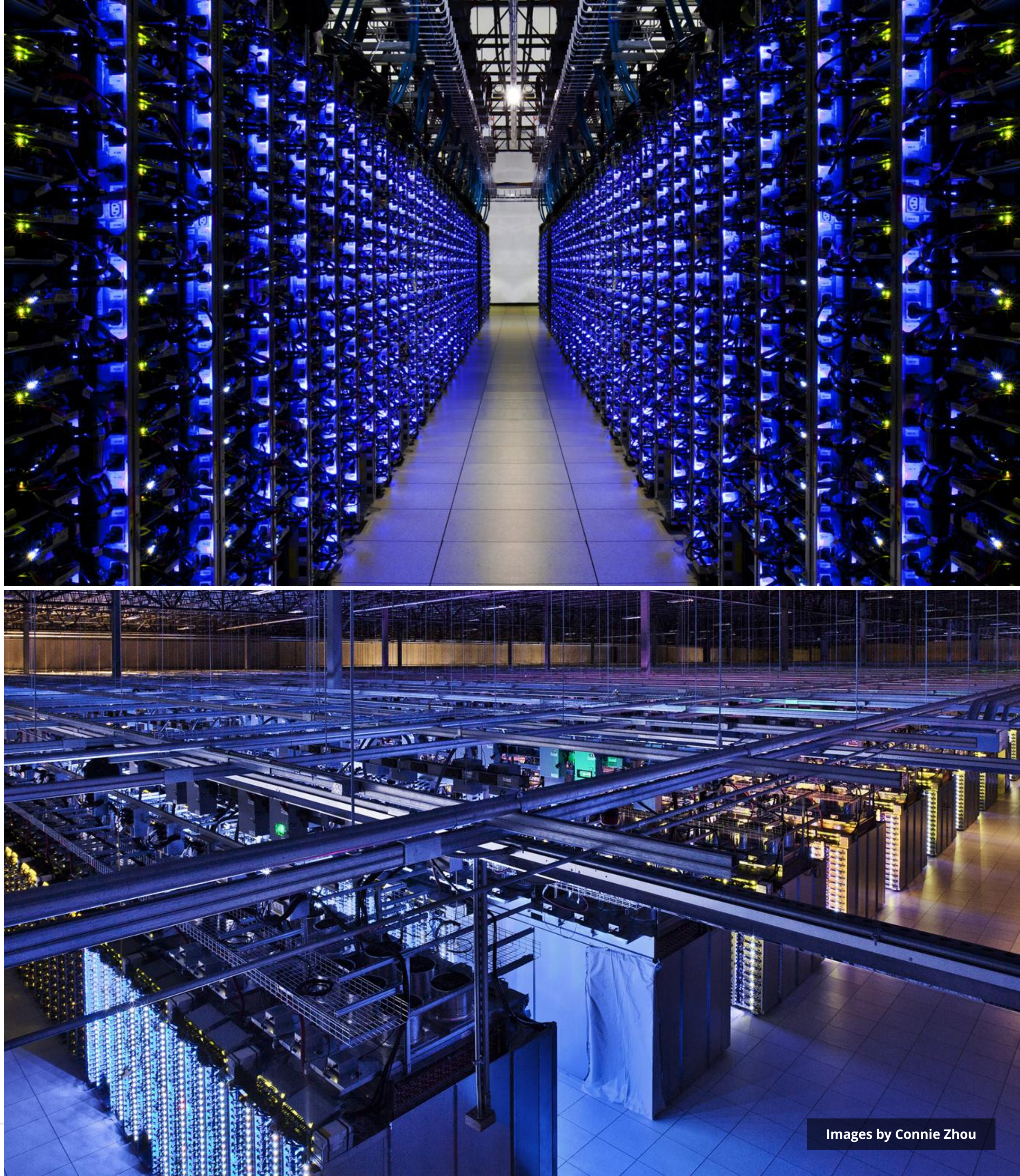
We launch over **2 billion** containers **per week**.

Shipping Containers At Clyde, by Steve Gibson

But it’s 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

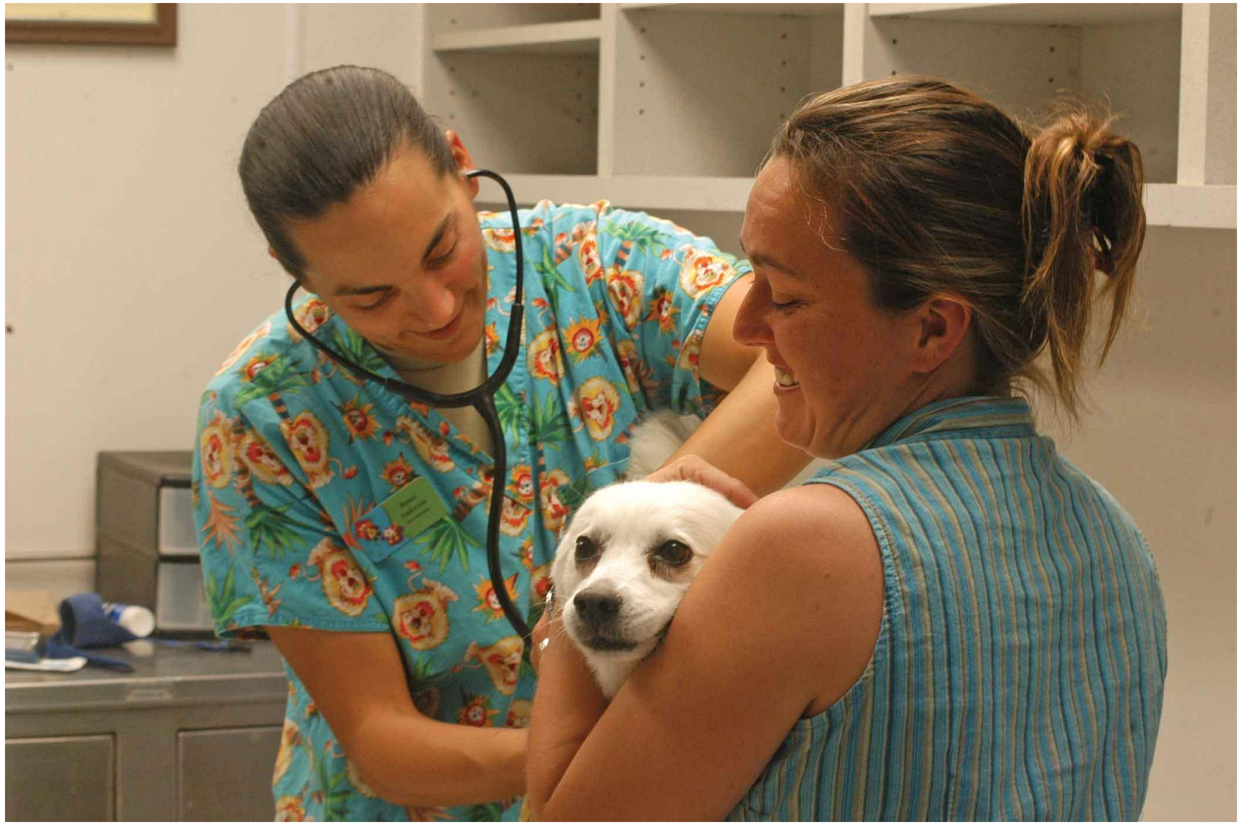
Enter Kubernetes

Greek for *“Helmsman”*; also the root of the word *“Governor”* and *“cybernetic”*

- Container orchestrator
- Runs and manages containers
- Supports multiple cloud and bare-metal environments
- Inspired and informed by Google’s experiences and internal systems
- **100% Open source**, written in **Go**

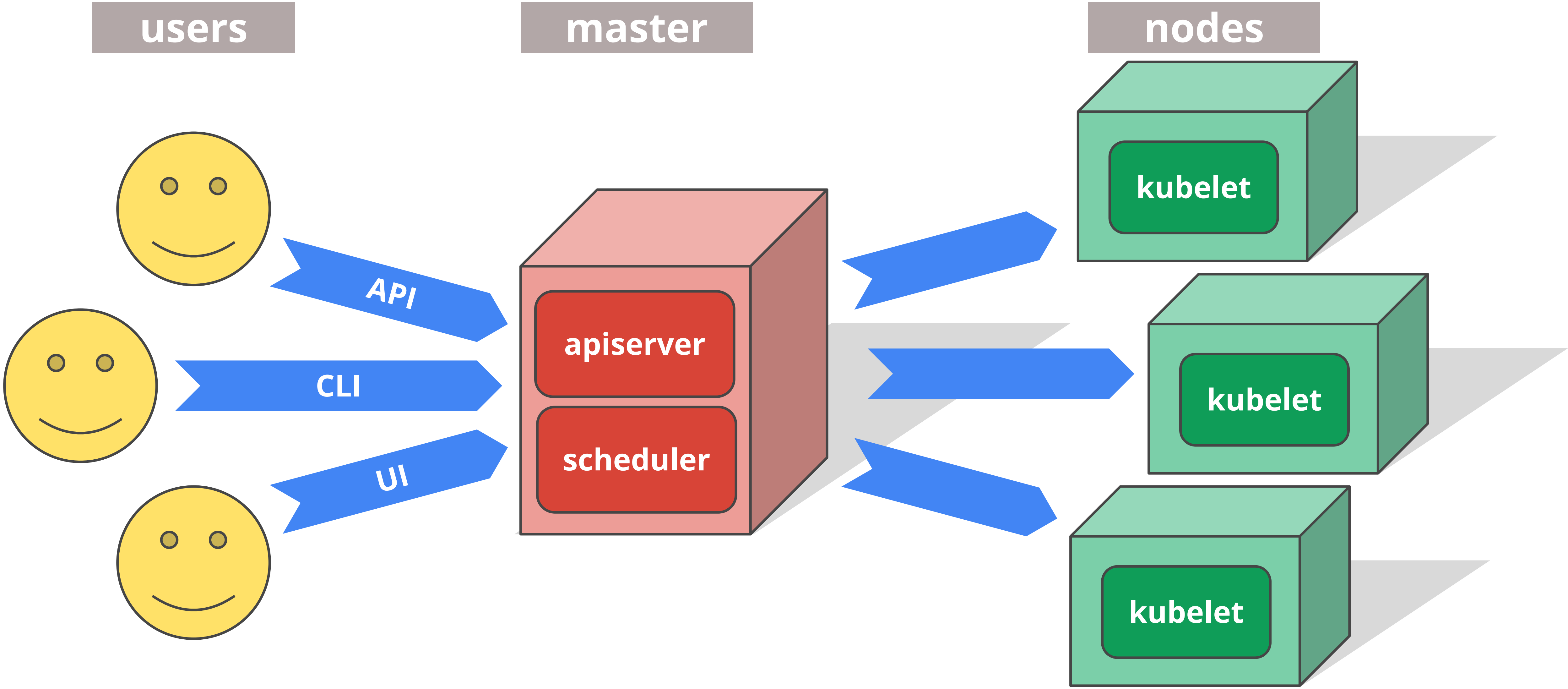
Manage applications, not machines

Pets vs. Cattle

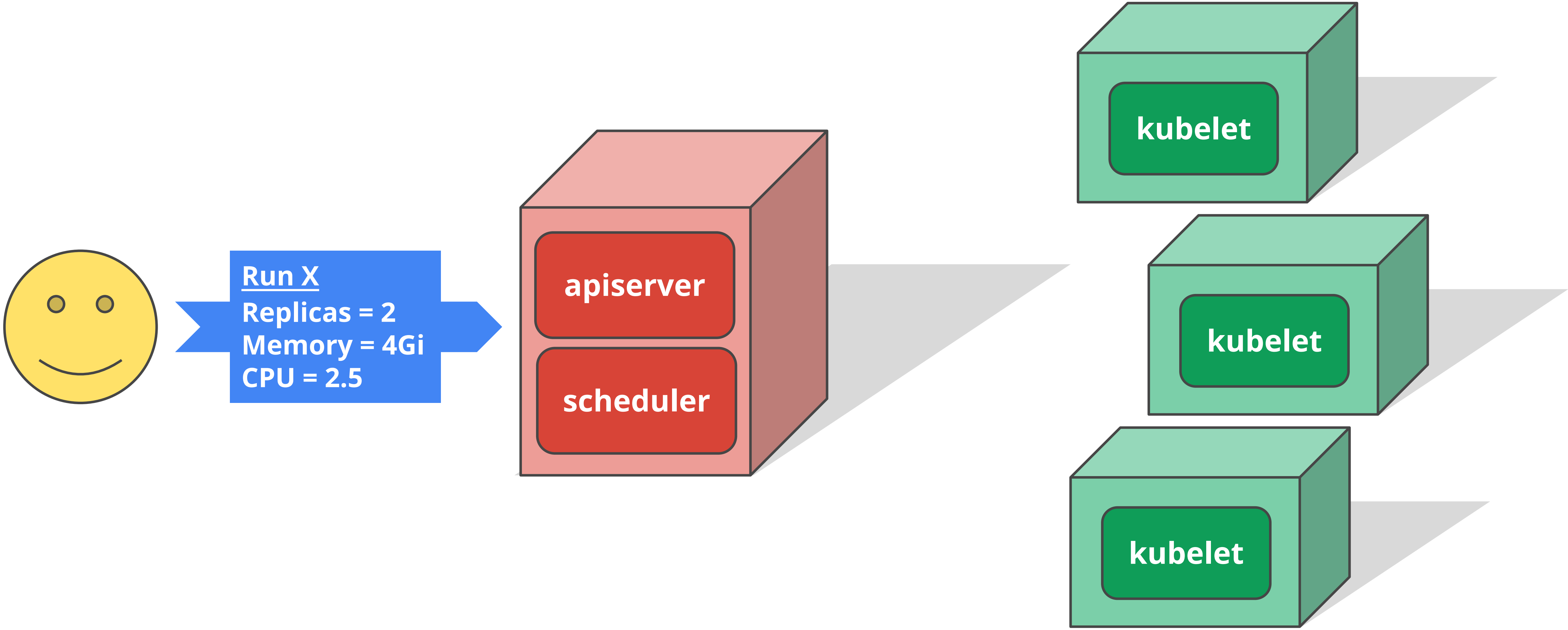


A 50000 foot view

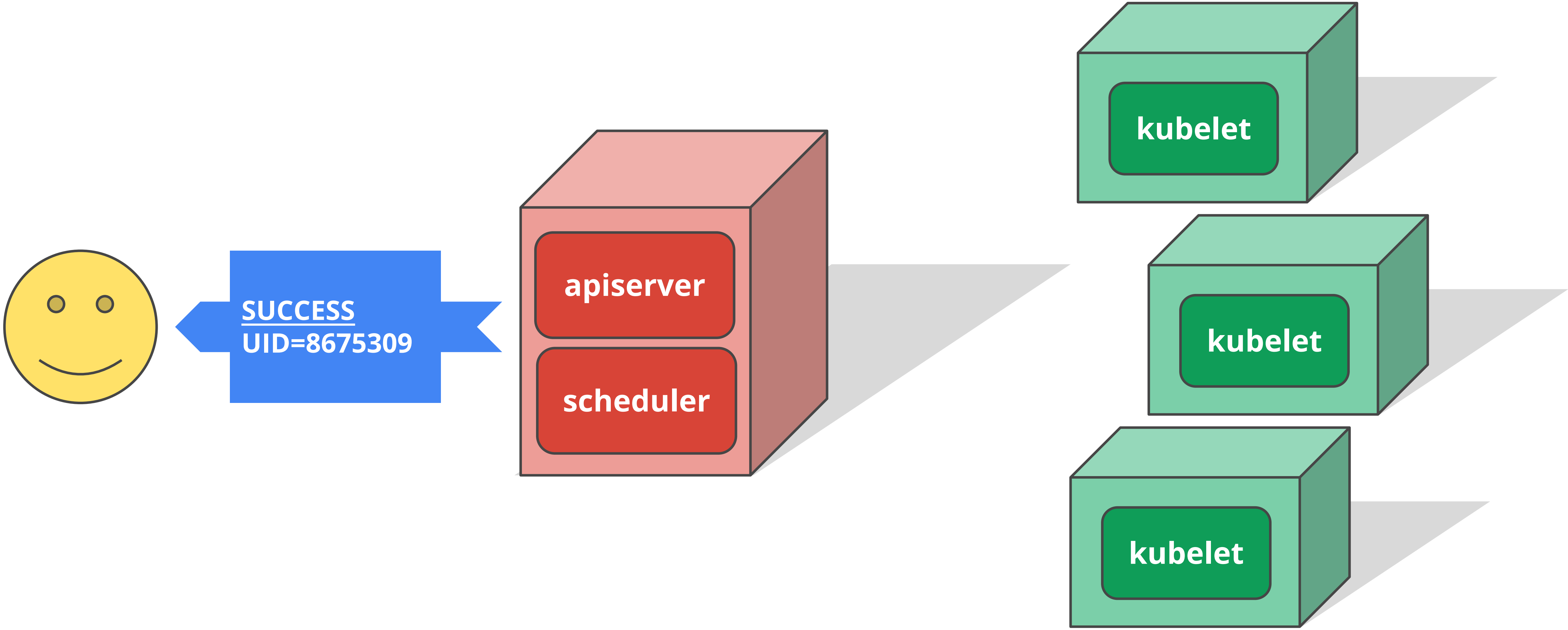
A 50000 foot view



A 50000 foot view

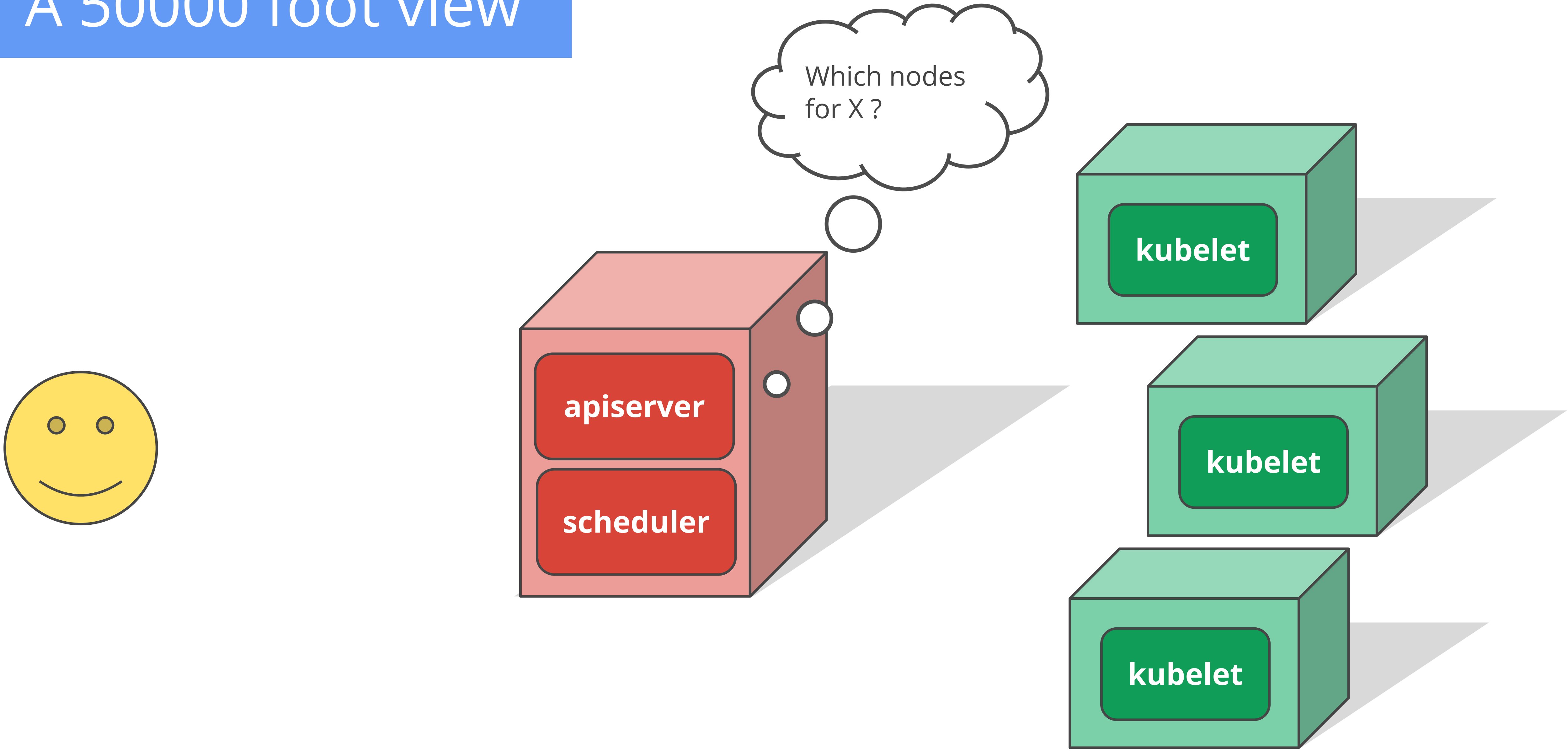


A 50000 foot view

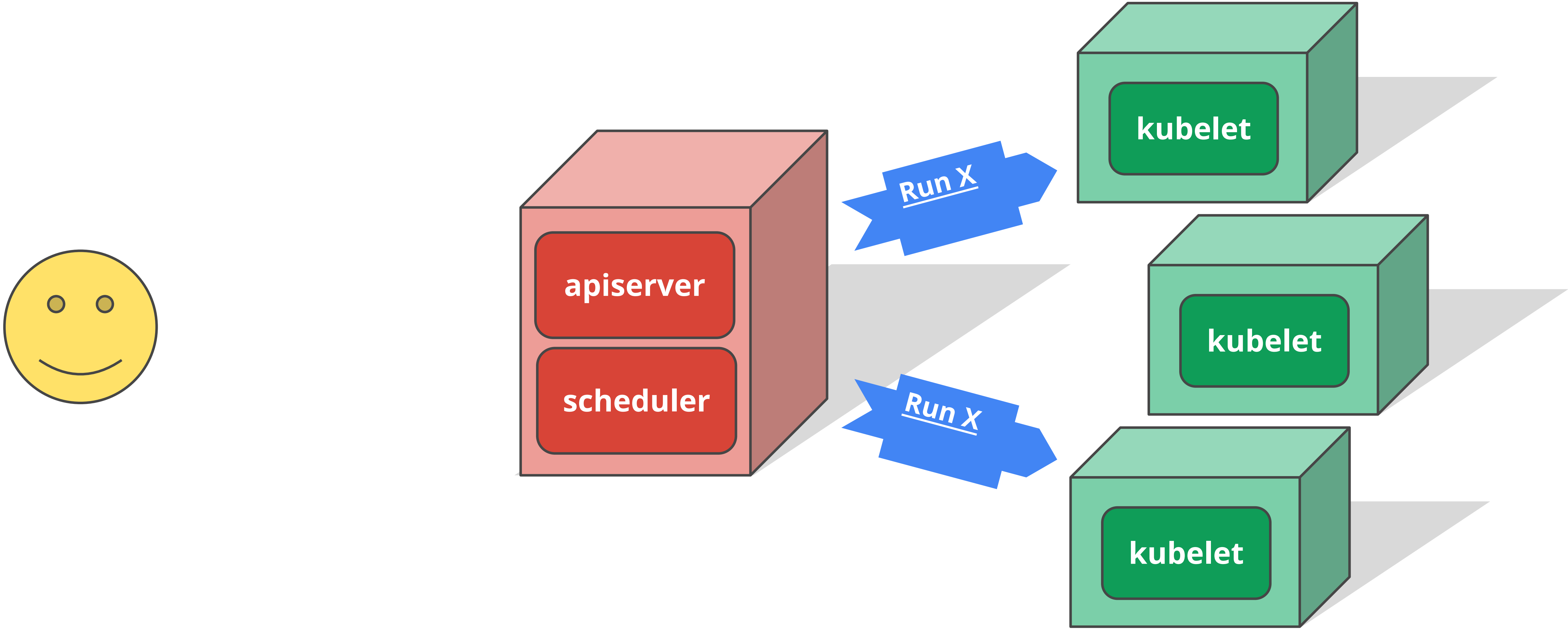


SUCCESS
UID=8675309

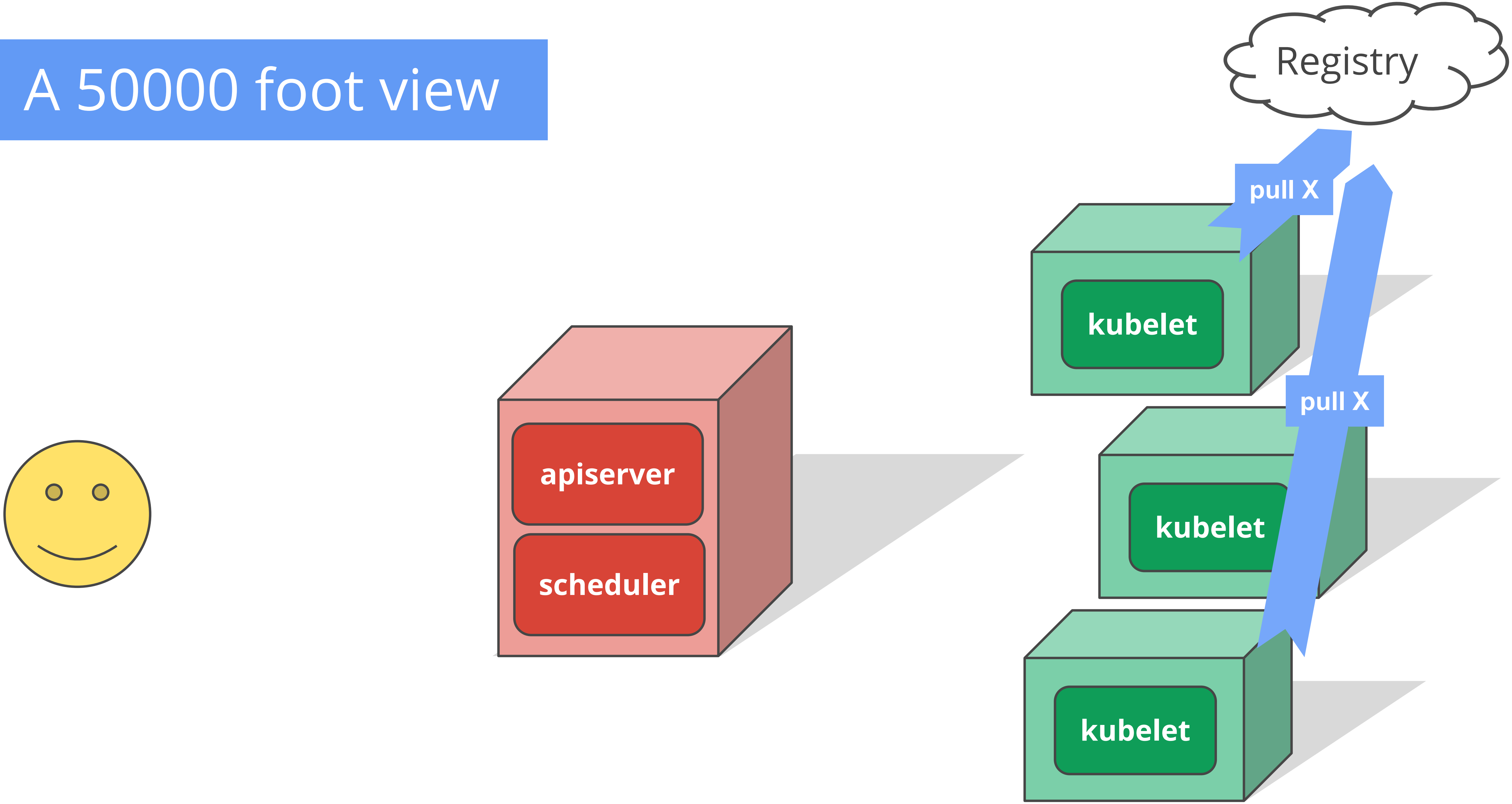
A 50000 foot view



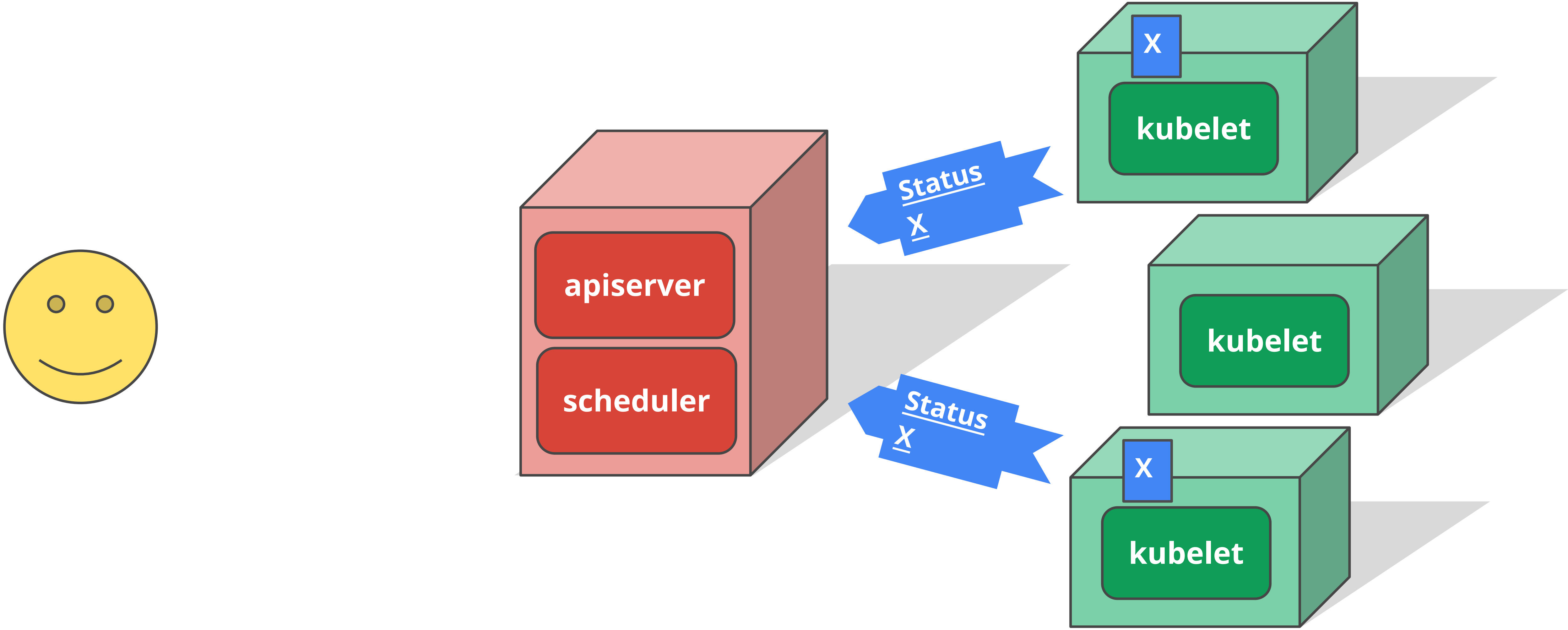
A 50000 foot view



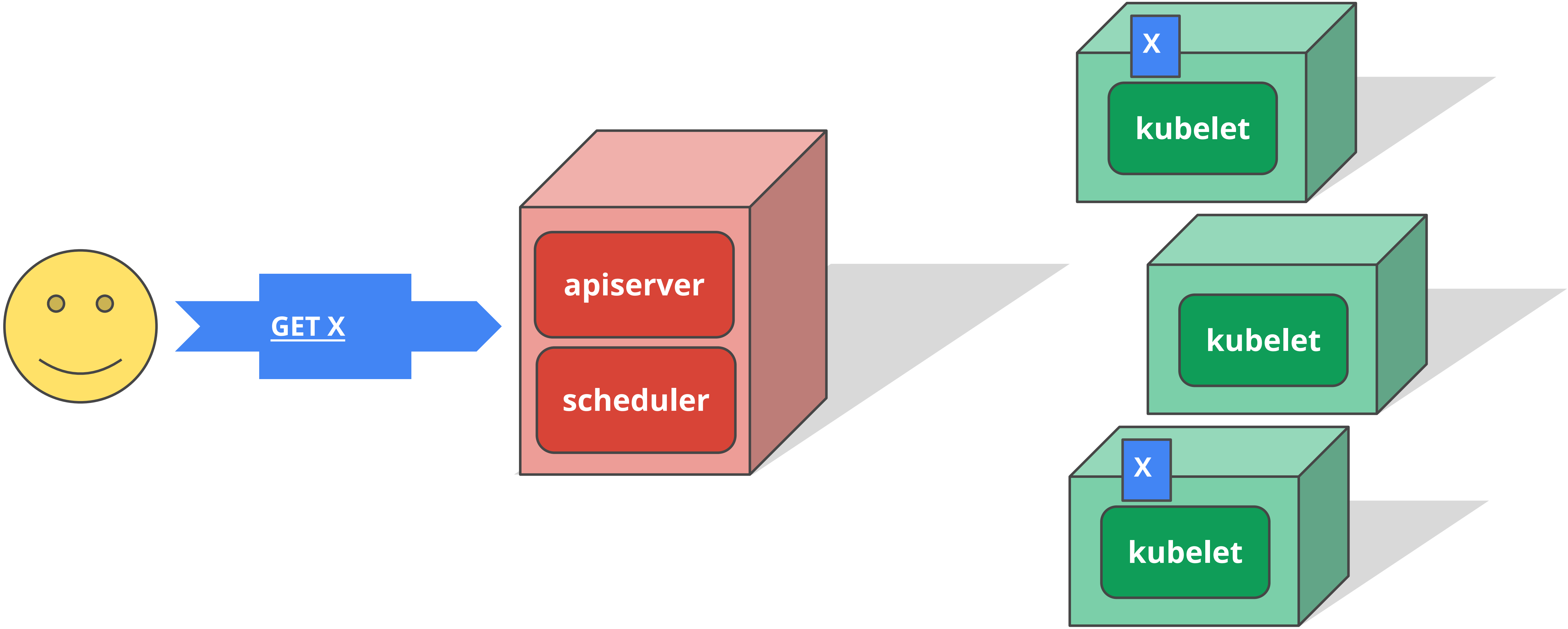
A 50000 foot view



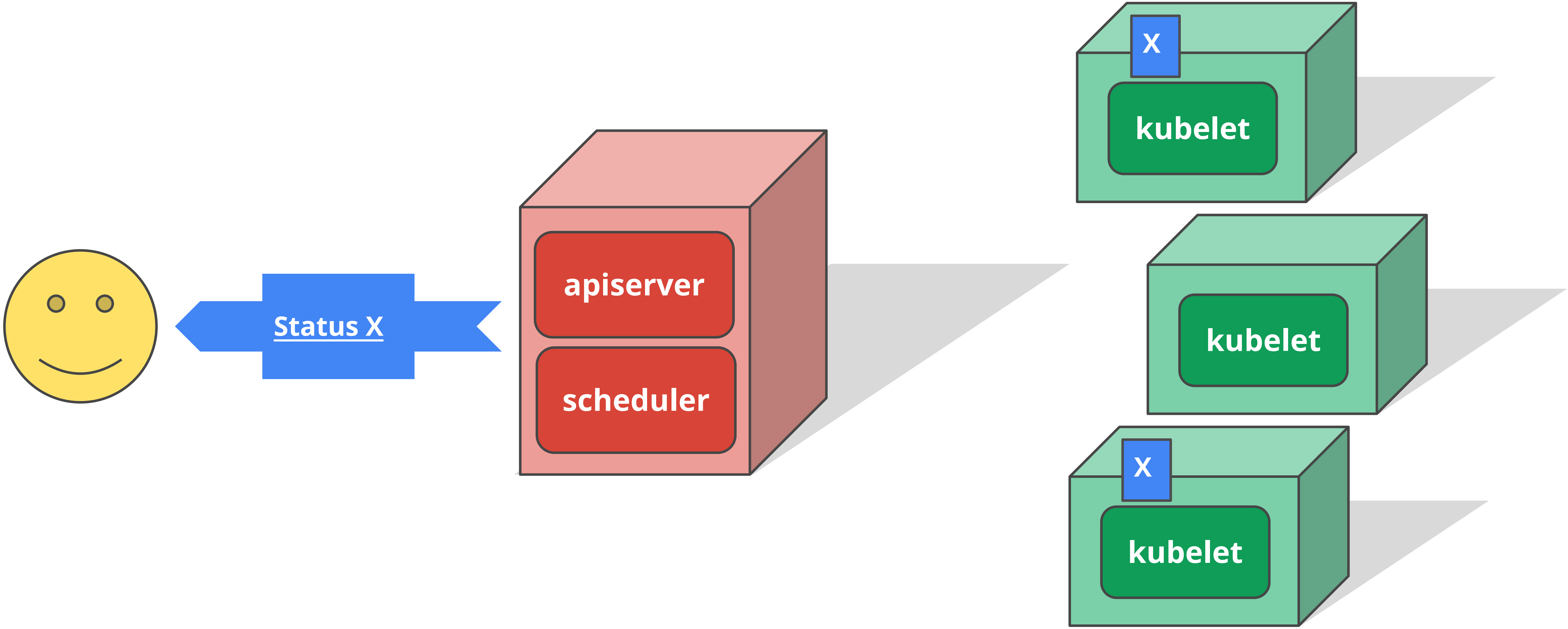
A 50000 foot view



A 50000 foot view

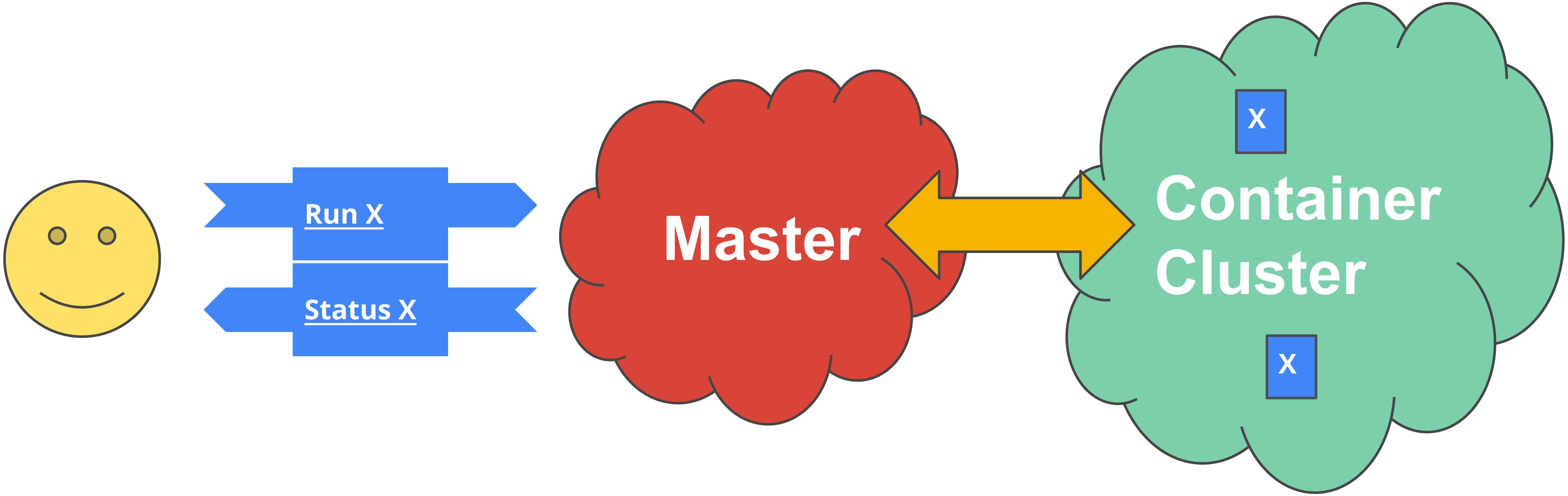


A 50000 foot view



Container clusters: A story in two parts

All you really care about



Container clusters: A story in two parts

1. **Setting up a 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 a cluster**
   - Run Pods & Containers
   - Replication controllers
   - Services
   - Volumes
   - Secrets

This is the fun part!

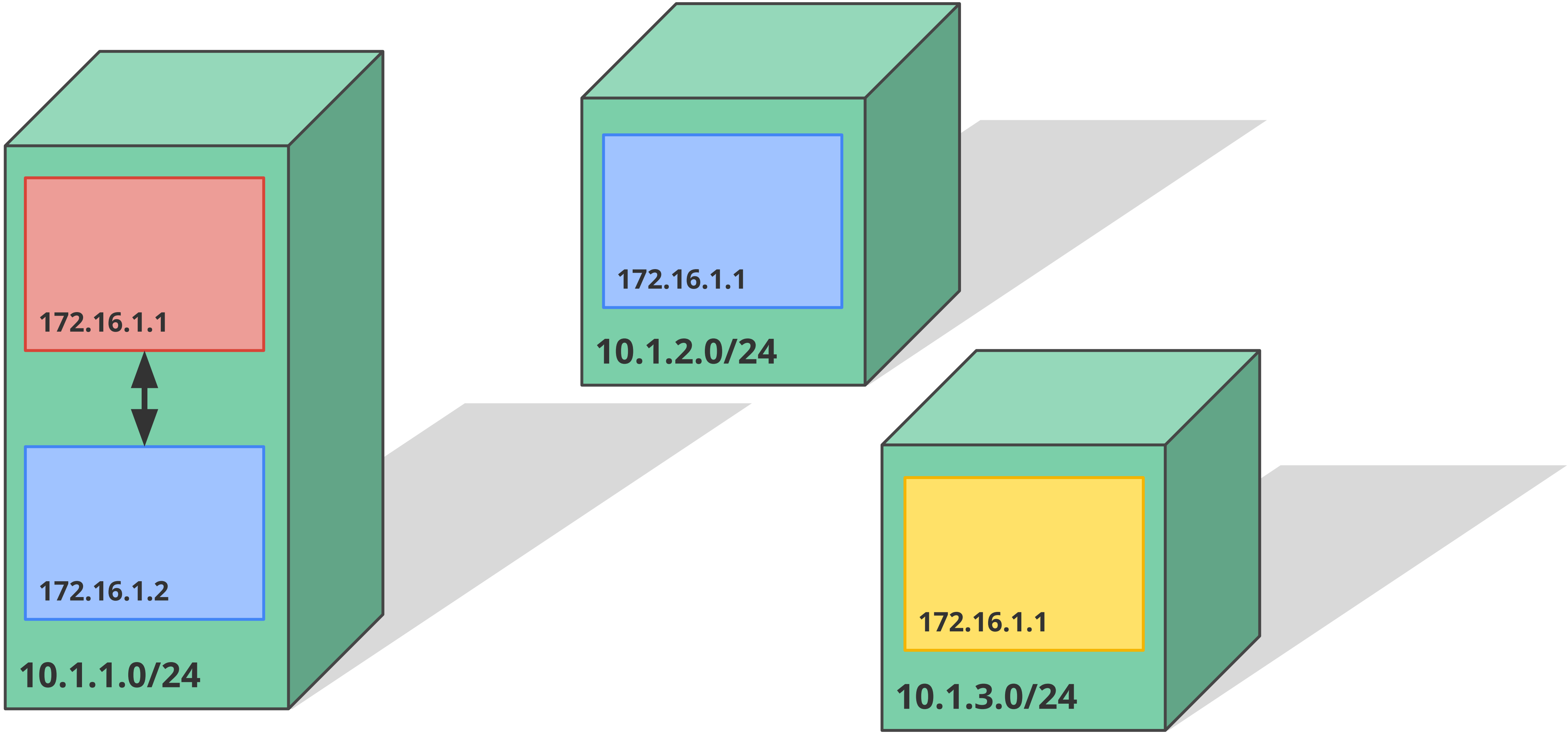
A distinct set of problems from cluster setup and management

Don't make developers deal with cluster administration!

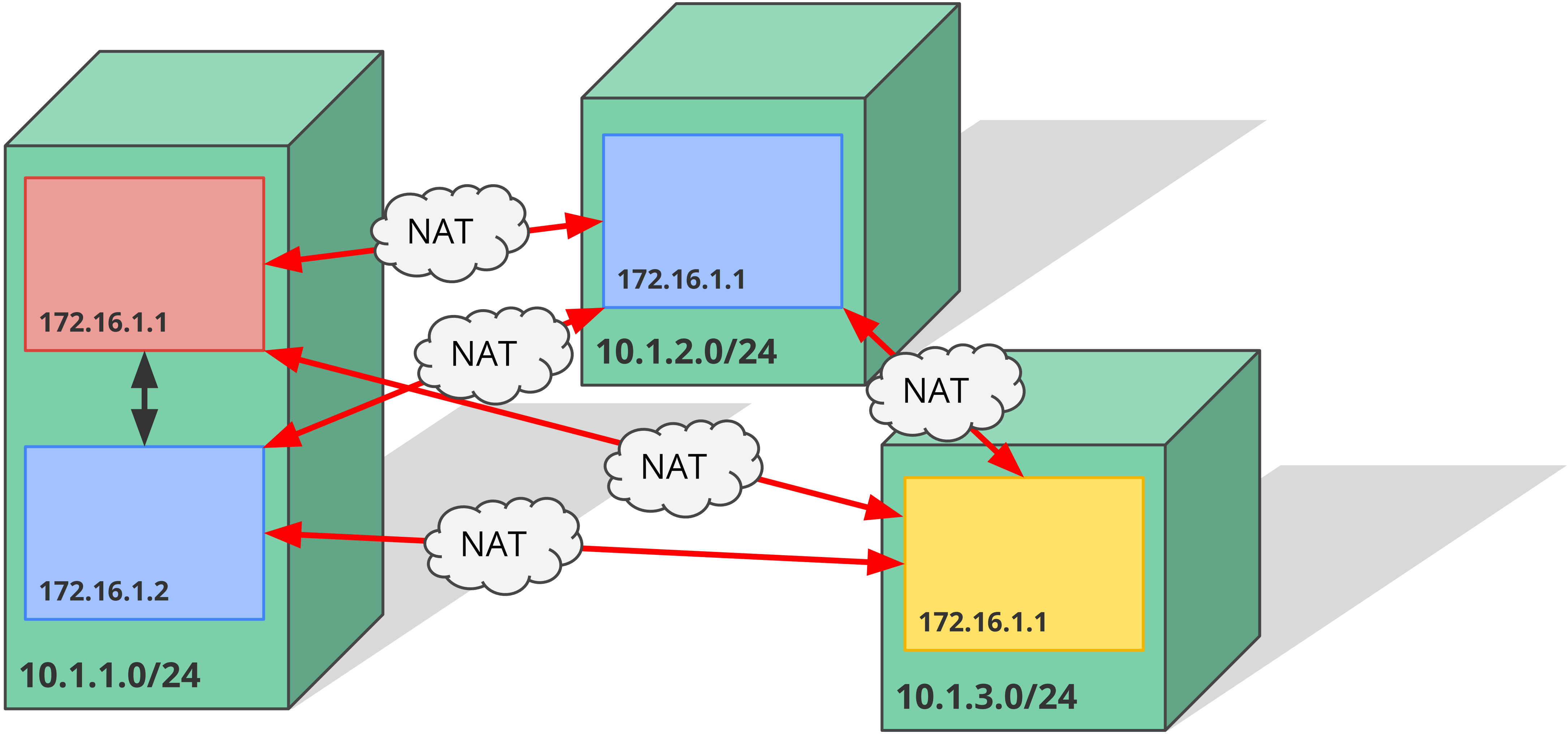
Accelerate development by focusing on the applications, not the cluster

Networking

Docker networking



Docker networking



Kubernetes networking

Pod IPs are **routable**

- docker default is private IP

Pods can reach each other without NAT

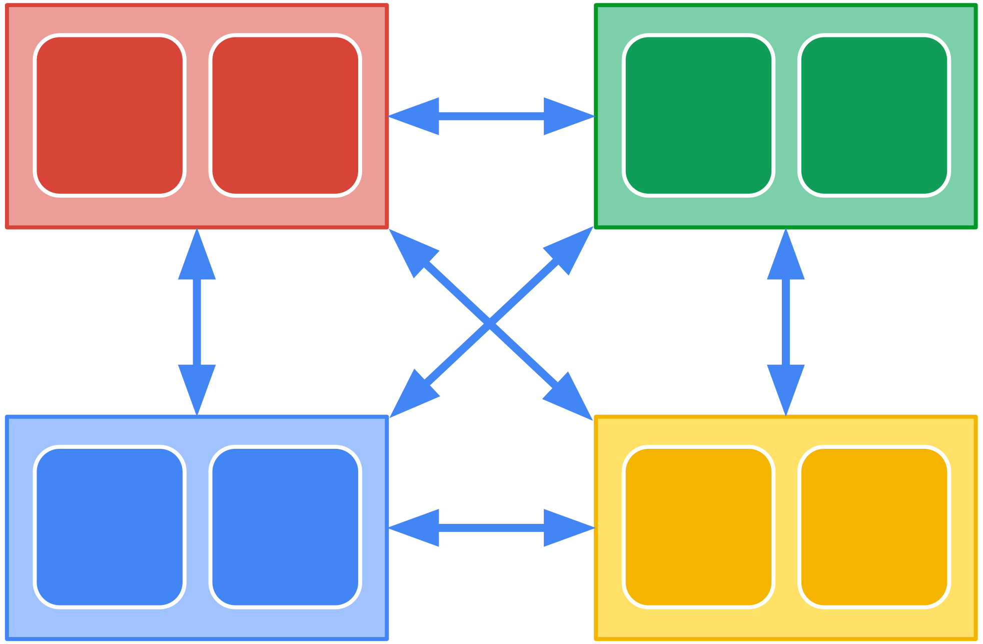
- even across nodes

No brokering of port numbers

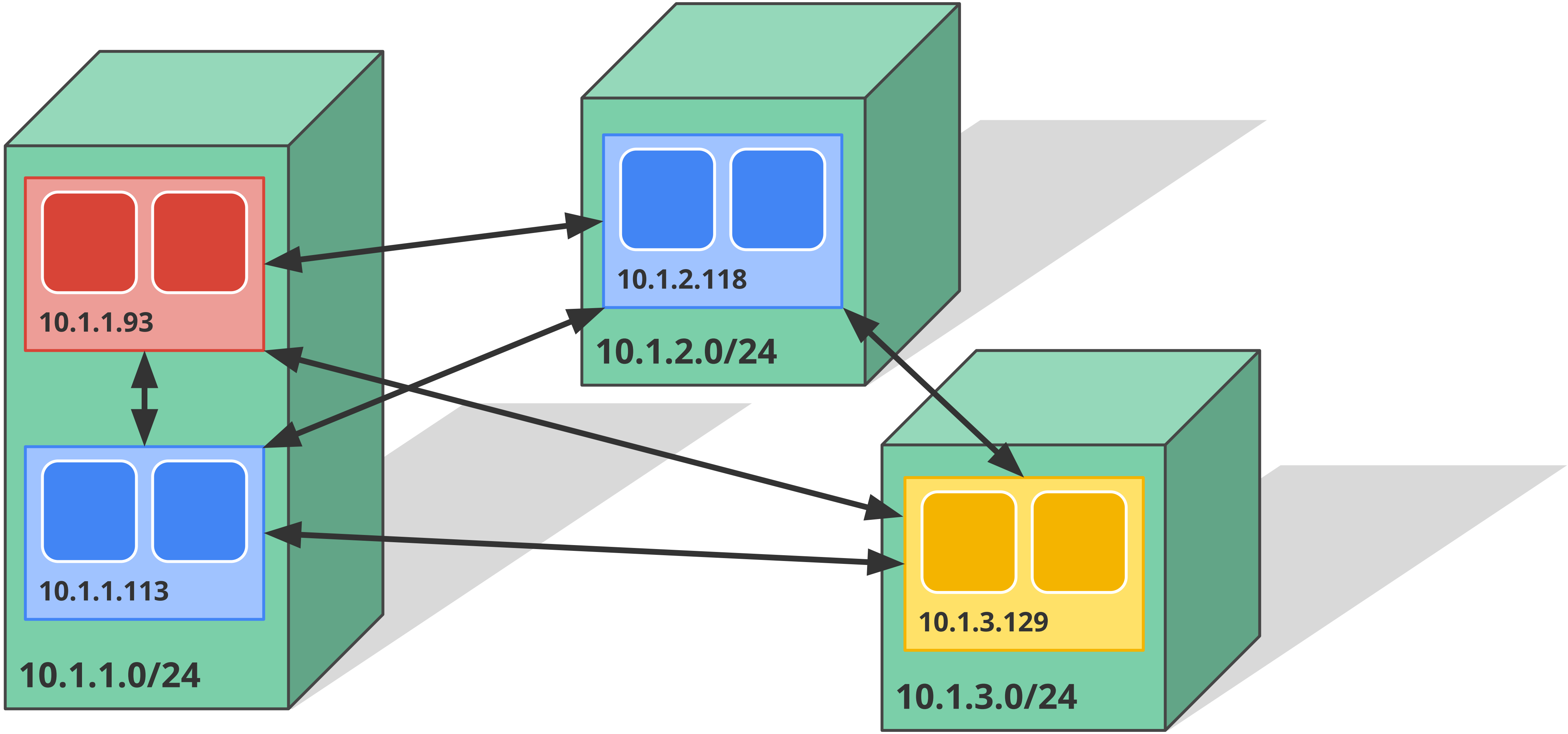
- too complex, why bother?

This is a fundamental requirement

- several SDN solutions exist



Kubernetes networking



Concept: Pods

Small group of containers & volumes

Tightly coupled

The atom of scheduling & placement in Kubernetes

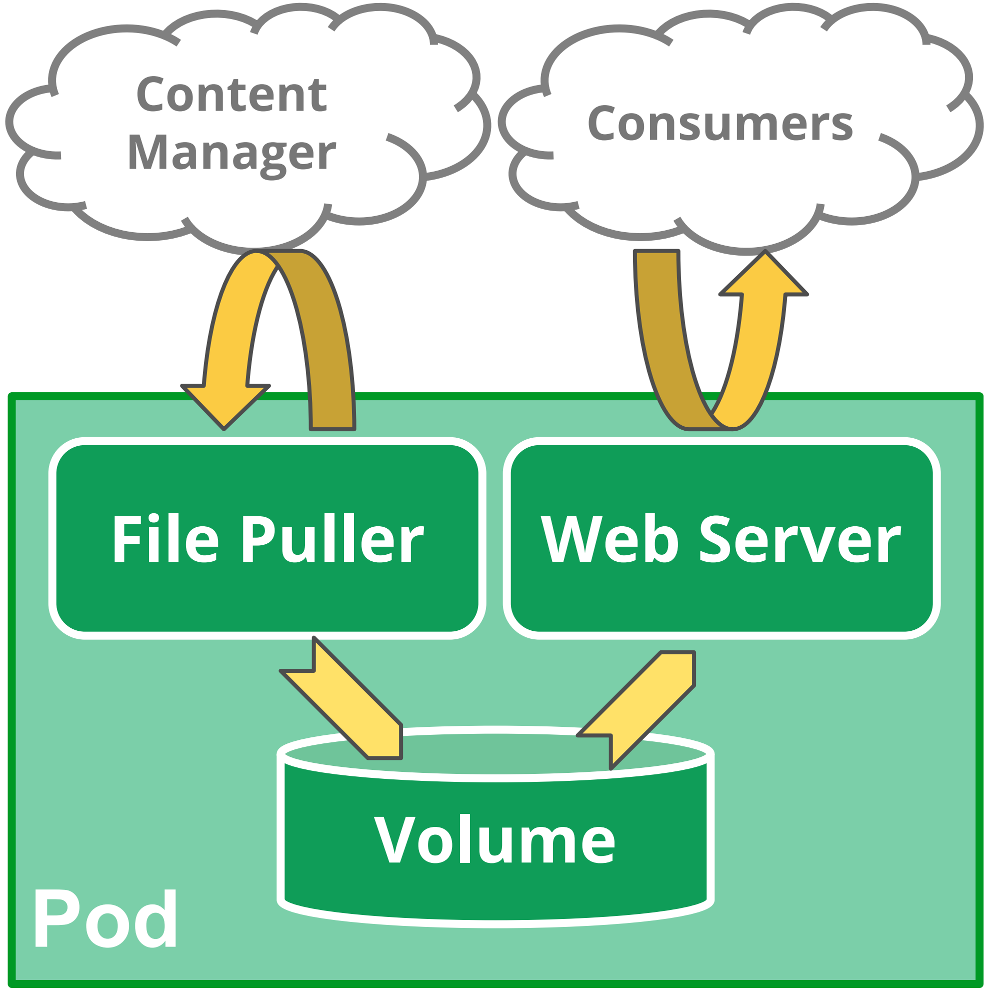
Shared namespace

- share IP address & localhost
- share IPC

Mortal

- can die, cannot be reborn

Example: data puller & web server



Concept: Services

A group of pods that **work together**

- grouped by a selector

Defines access policy

- “load balanced” or “headless” for now

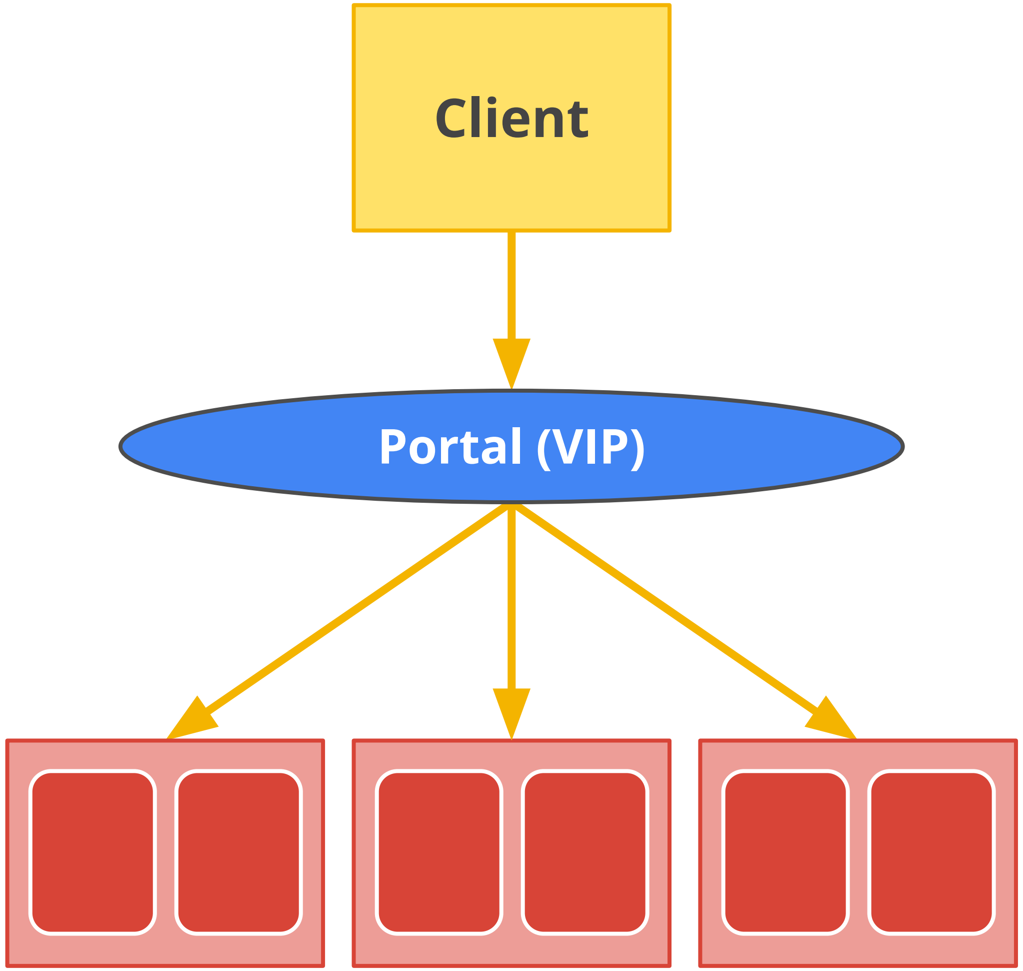
Gets a stable **virtual IP** and port

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

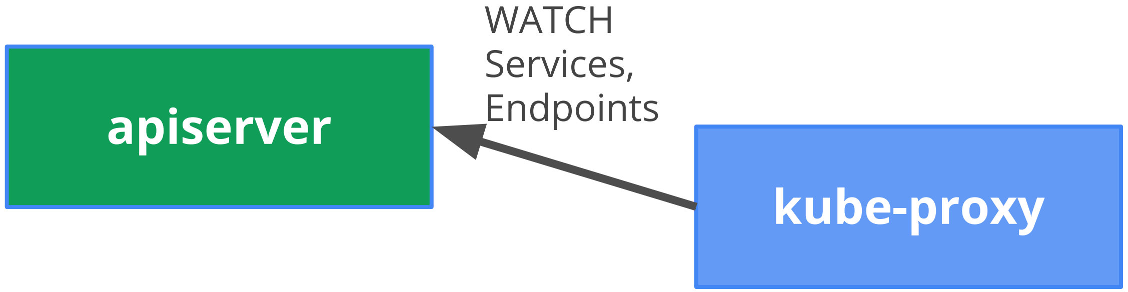
VIP is captured by *kube-proxy*

- watches the service constituency
- updates when backends change

Hides complexity - ideal for non-native apps



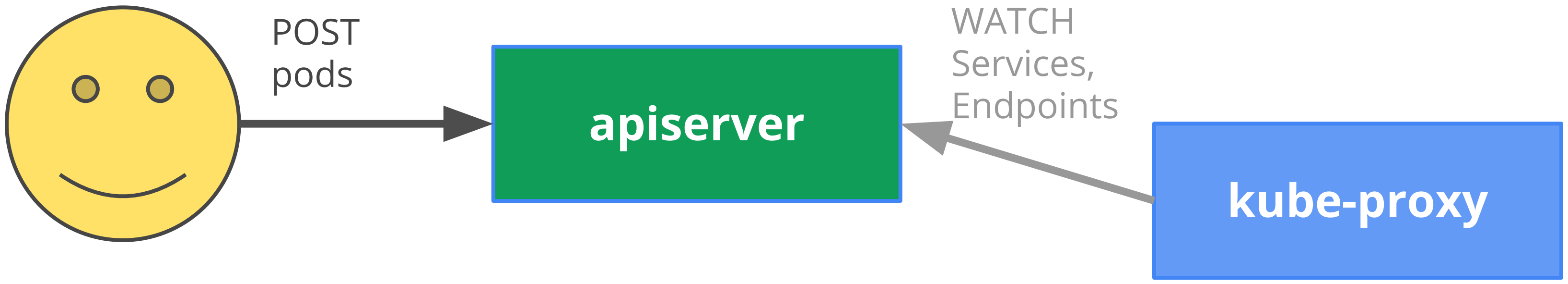
Services



Services

Pod

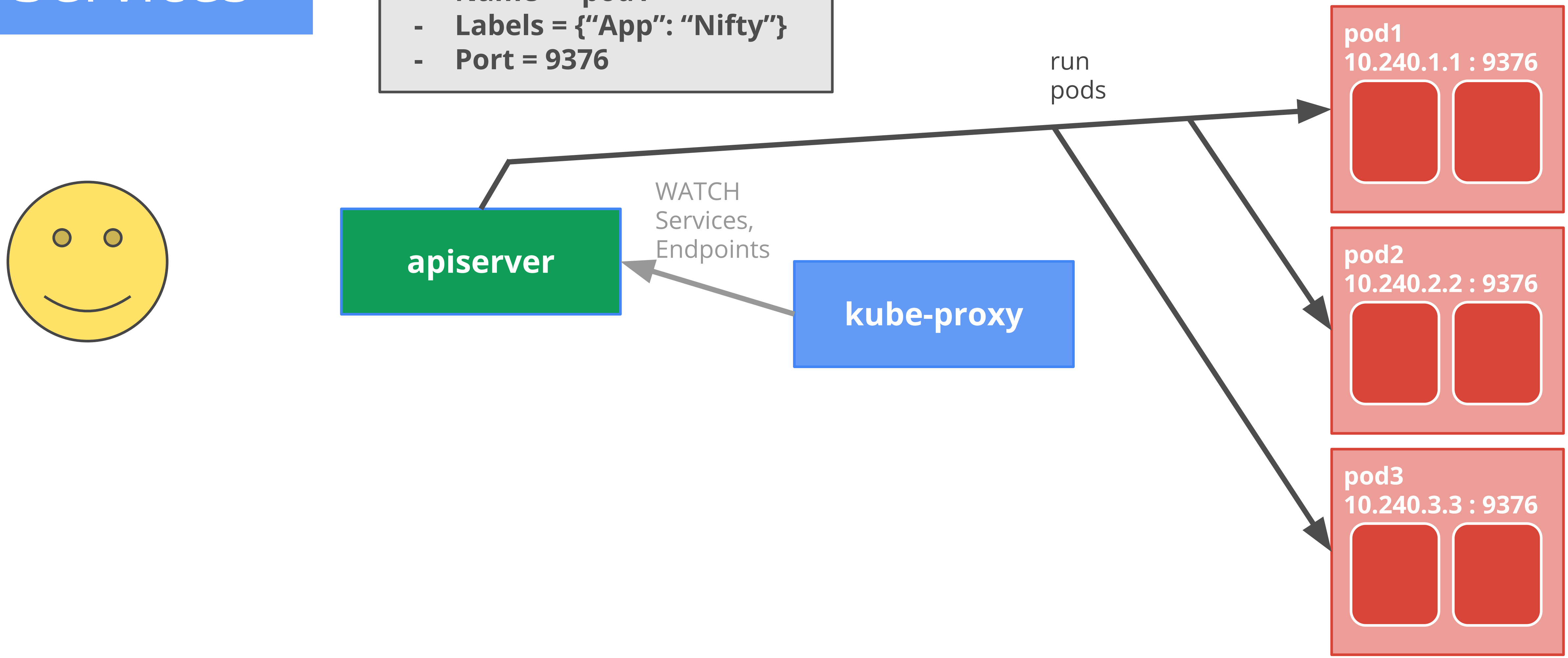
- **Name = “pod1”**
- **Labels = {“App”: “Nifty”}**
- **Port = 9376**



Services

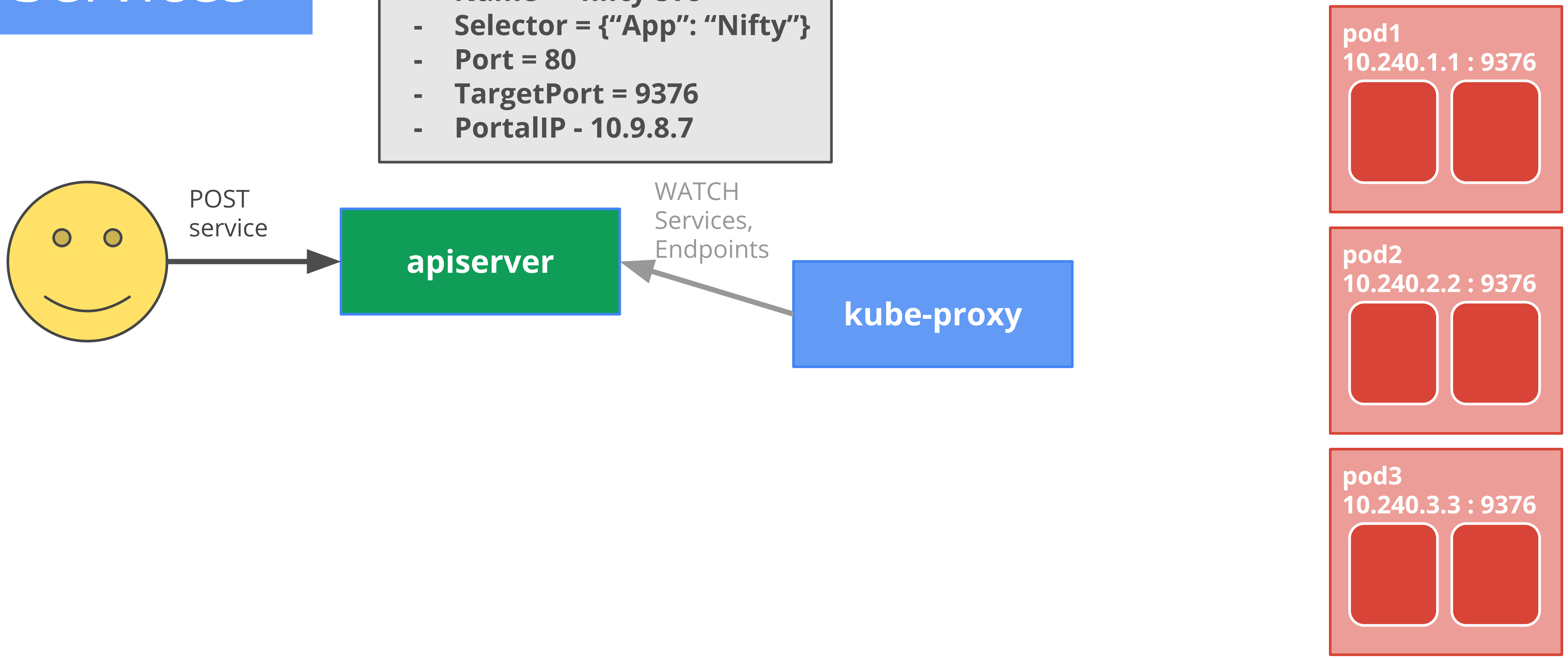
Pod

- **Name = “pod1”**
- **Labels = {“App”: “Nifty”}**
- **Port = 9376**

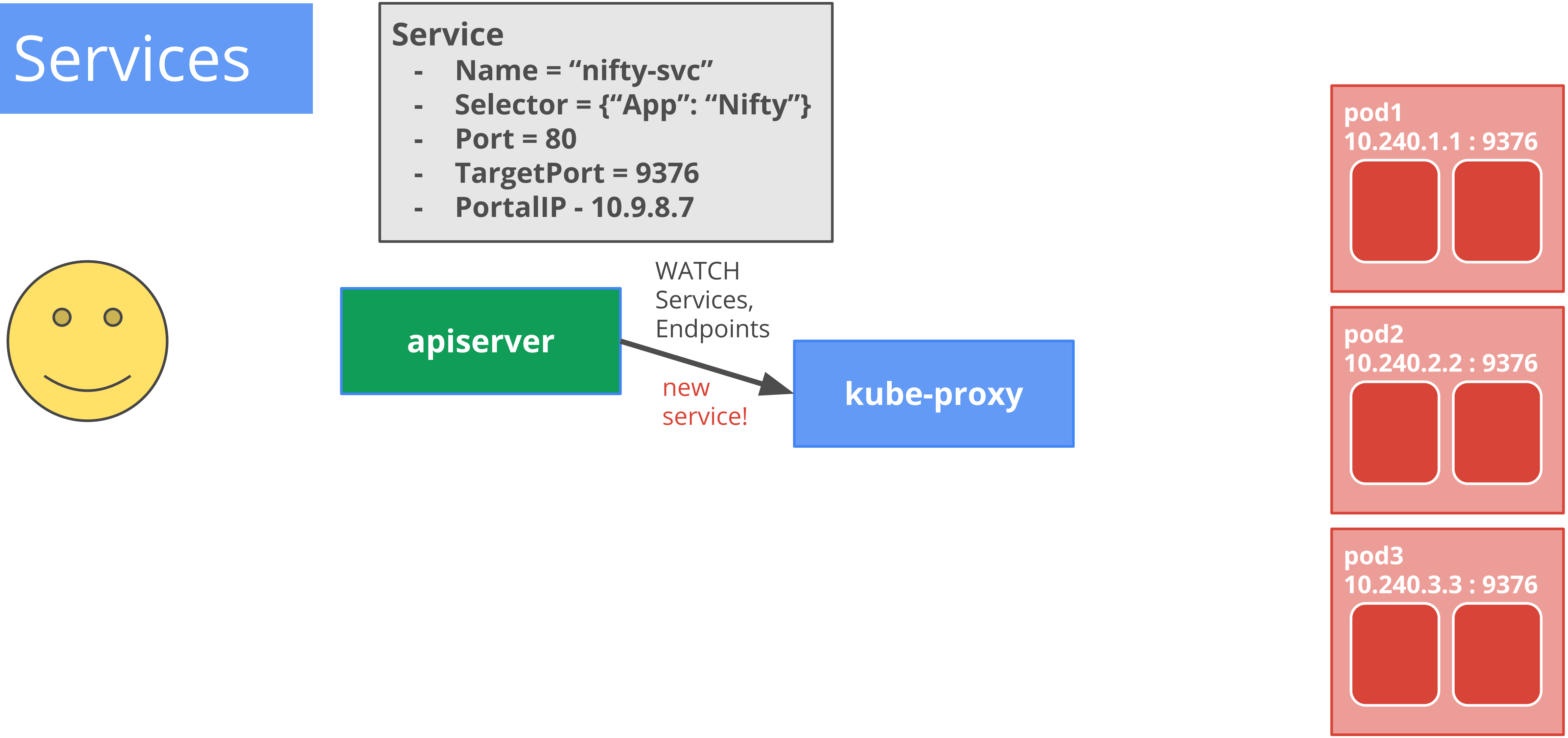


Services

Service
- **Name = “nifty-svc”**
- **Selector = {“App”: “Nifty”}**
- **Port = 80**
- **TargetPort = 9376**
- **PortalIP - 10.9.8.7**



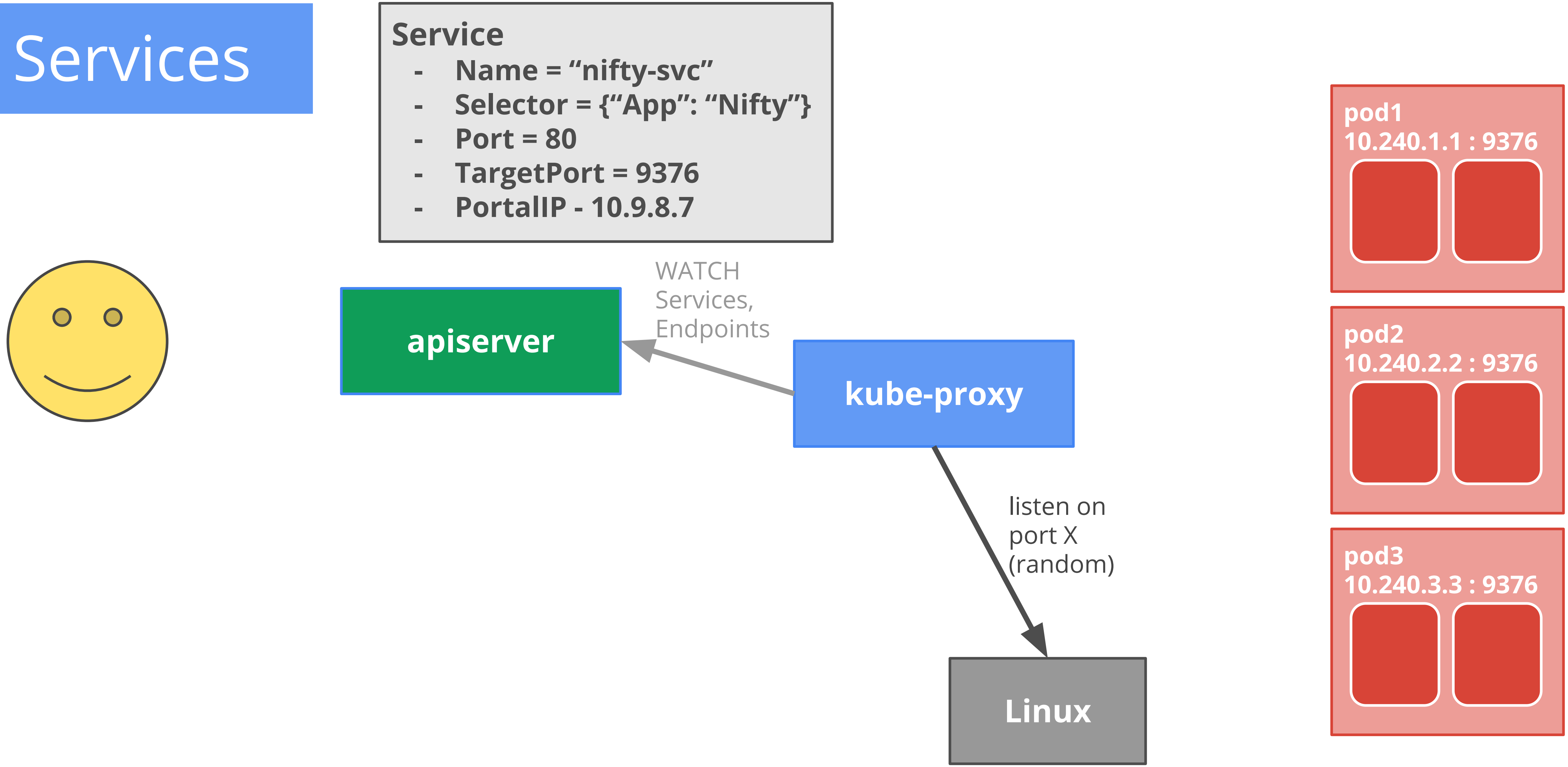
Services



Service

- **Name = “nifty-svc”**
- **Selector = {“App”: “Nifty”}**
- **Port = 80**
- **TargetPort = 9376**
- **PortalIP - 10.9.8.7**

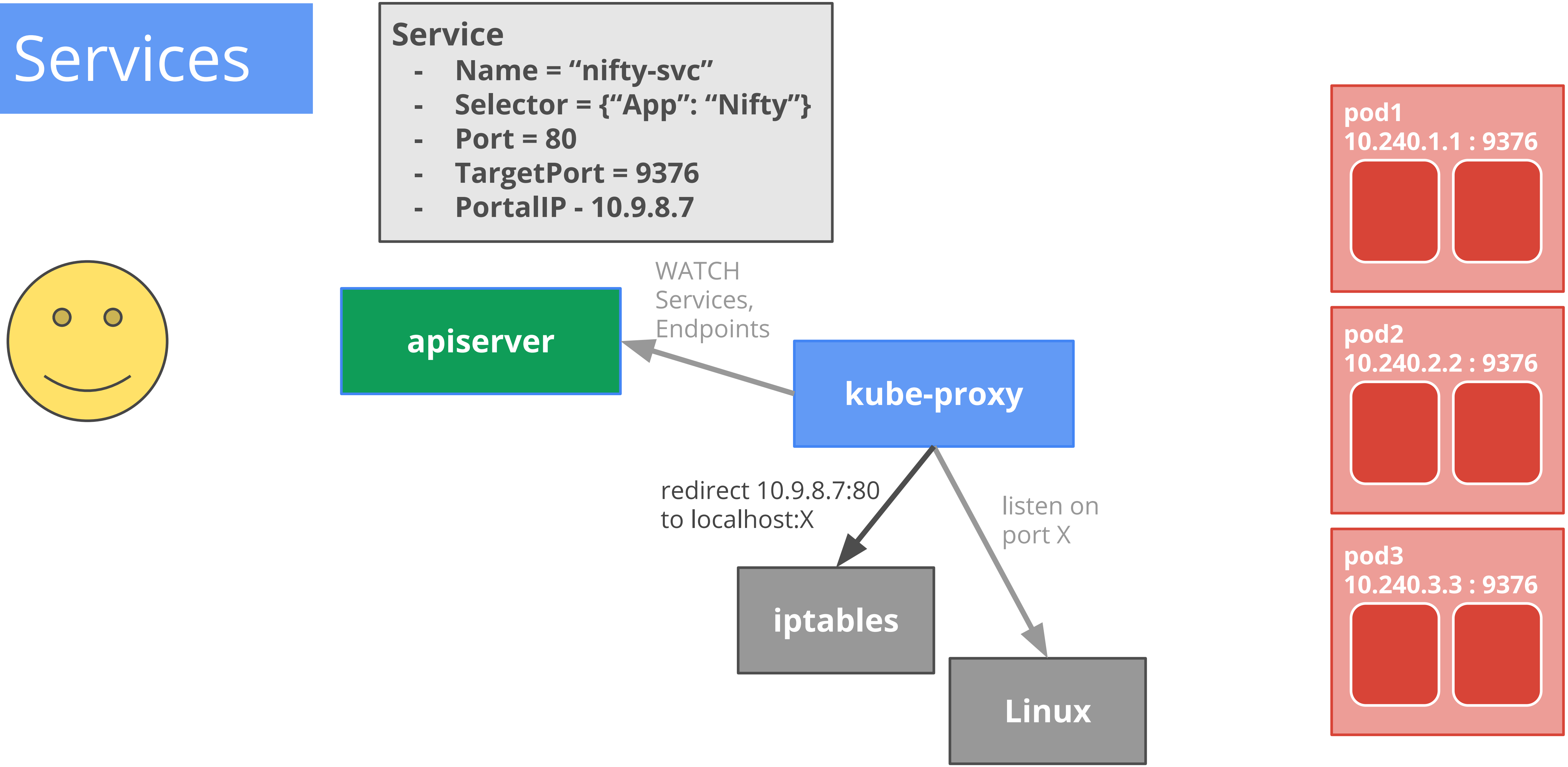
Services



Service

- Name = “nifty-svc”
- Selector = {“App”: “Nifty”}
- Port = 80
- TargetPort = 9376
- PortalIP - 10.9.8.7

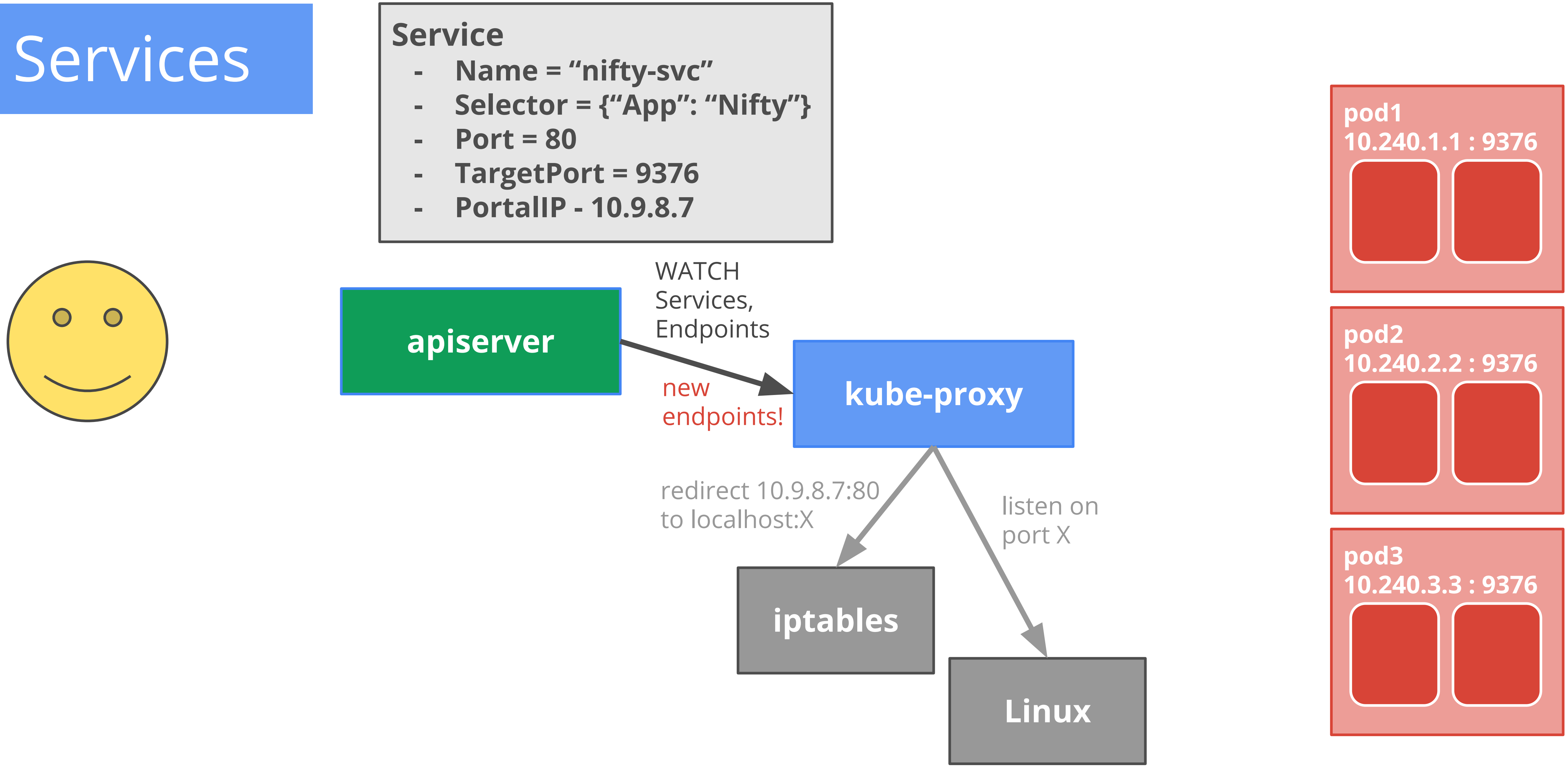
Services



Service

- Name = “nifty-svc”
- Selector = {“App”: “Nifty”}
- Port = 80
- TargetPort = 9376
- PortalIP - 10.9.8.7

Services

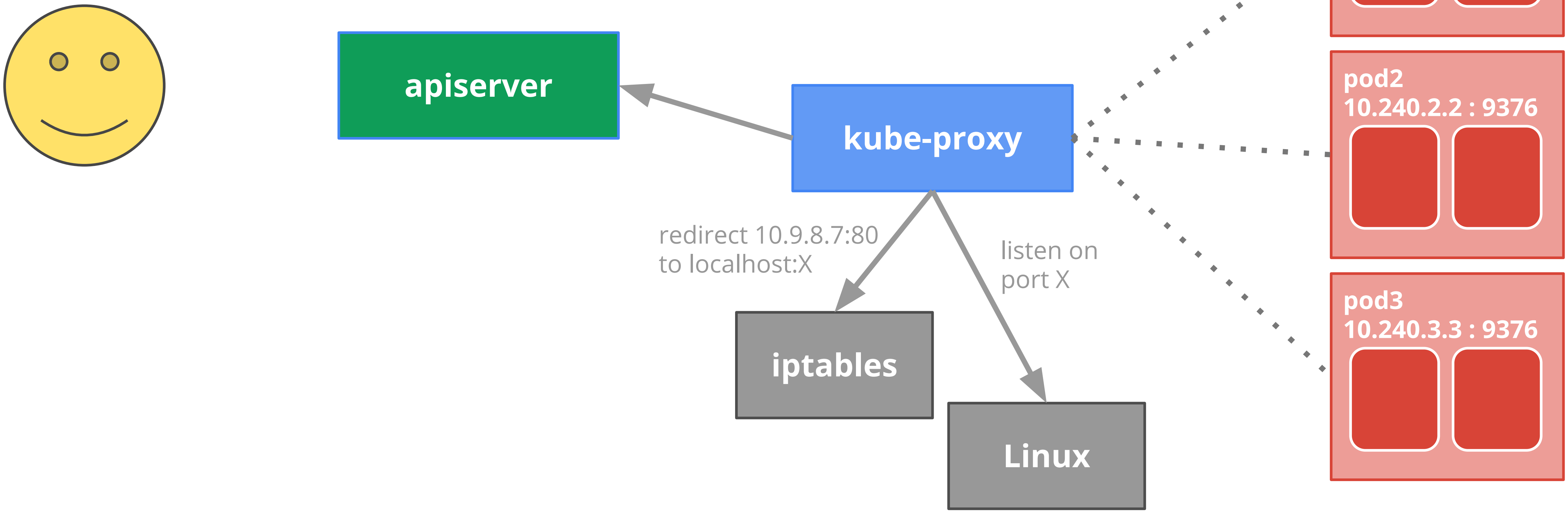


Service
- **Name = “nifty-svc”**
- **Selector = {“App”: “Nifty”}**
- **Port = 80**
- **TargetPort = 9376**
- **PortalIP - 10.9.8.7**

Services

Service

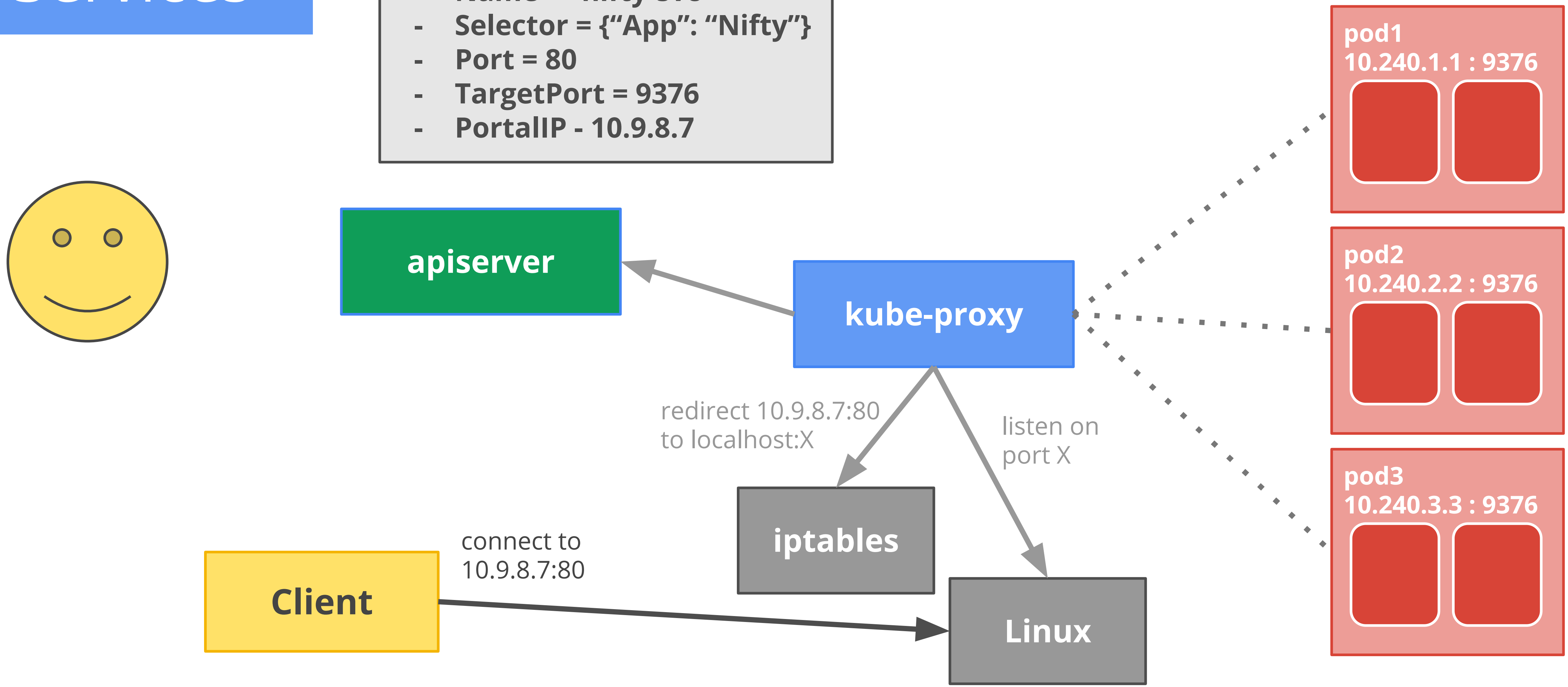
- **Name = “nifty-svc”**
- **Selector = {“App”: “Nifty”}**
- **Port = 80**
- **TargetPort = 9376**
- **PortalIP - 10.9.8.7**



Services

Service

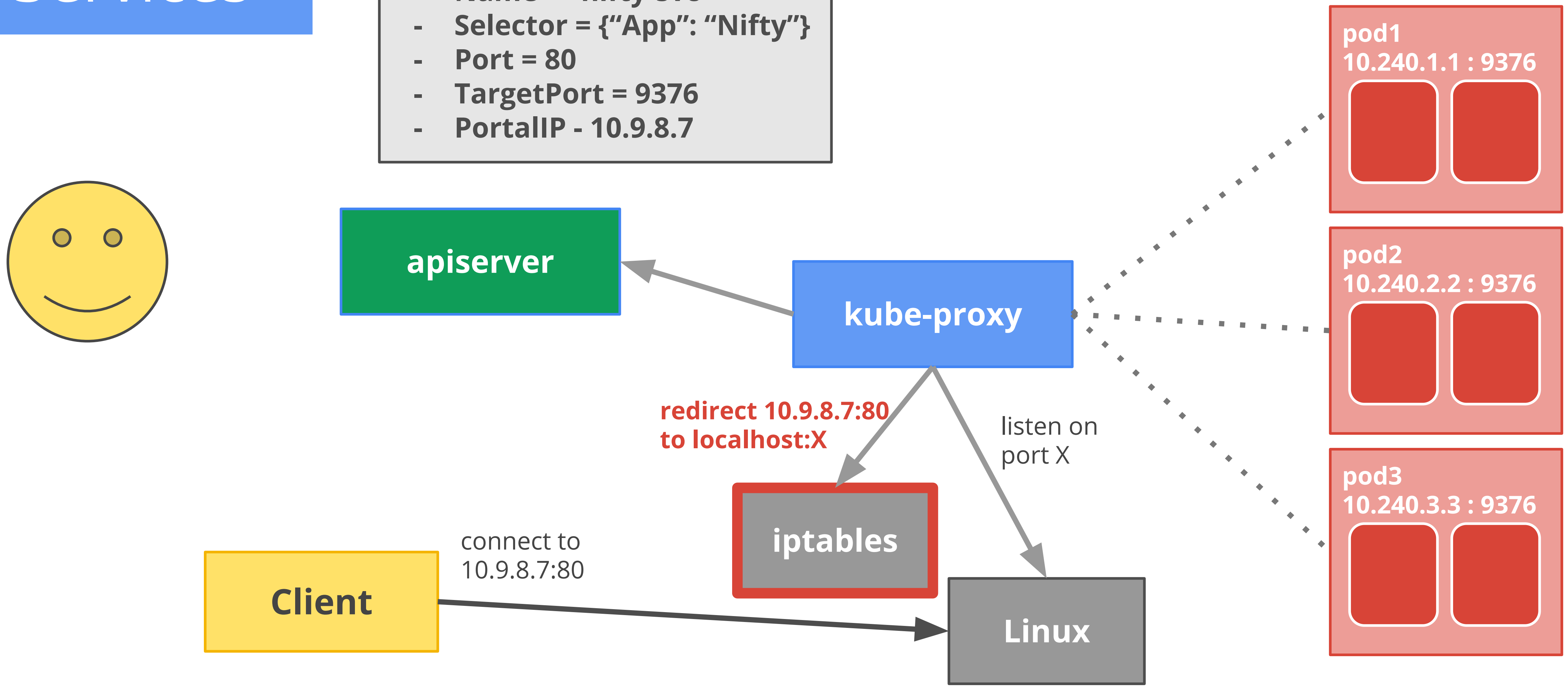
- Name = “nifty-svc”
- Selector = {“App”: “Nifty”}
- Port = 80
- TargetPort = 9376
- PortalIP - 10.9.8.7



Services

Service

- **Name = “nifty-svc”**
- **Selector = {“App”: “Nifty”}**
- **Port = 80**
- **TargetPort = 9376**
- **PortalIP - 10.9.8.7**



apiserver

kube-proxy

redirect 10.9.8.7:80
to localhost:X

listen on
port X

iptables

Client

connect to
10.9.8.7:80

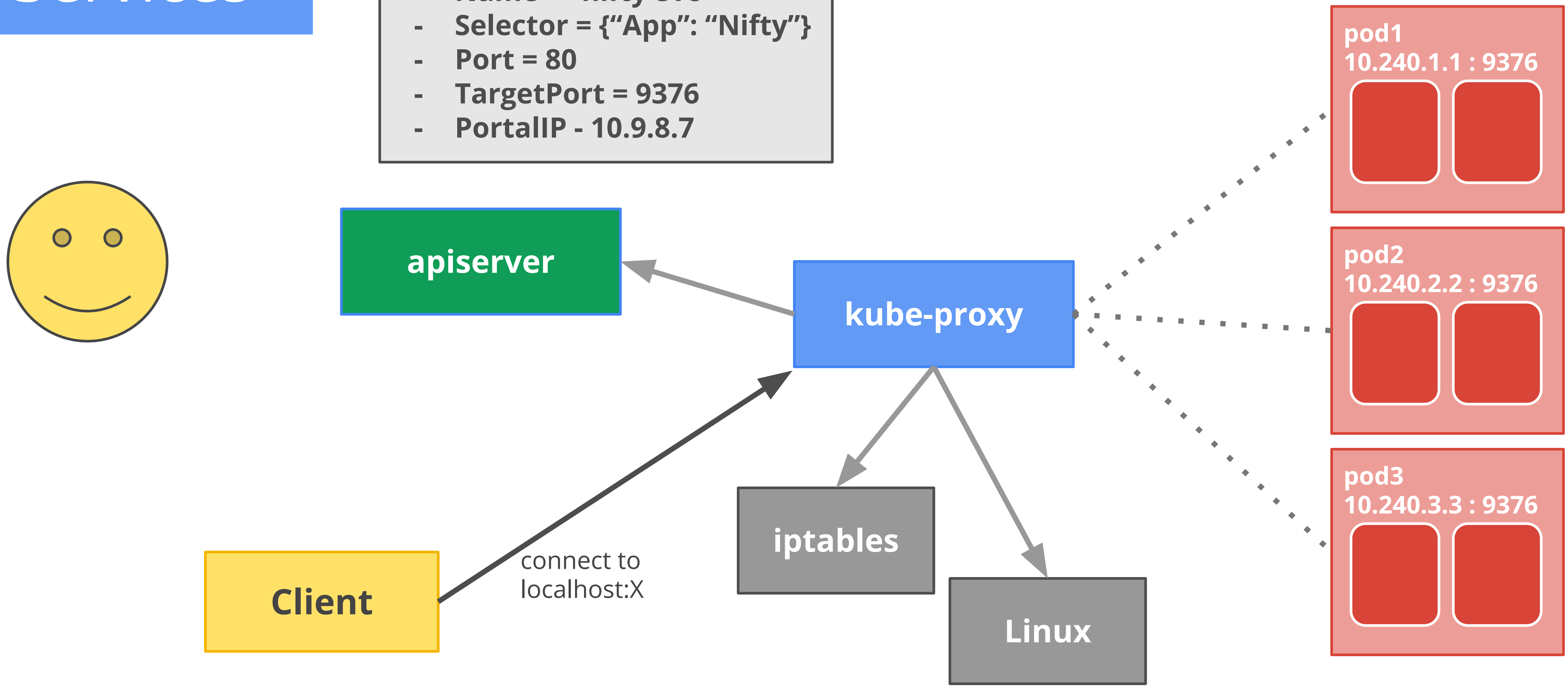
Linux

pod1
10.240.1.1 : 9376

pod2
10.240.2.2 : 9376

pod3
10.240.3.3 : 9376

Services



Service

- Name = “nifty-svc”
- Selector = {“App”: “Nifty”}
- Port = 80
- TargetPort = 9376
- PortalIP - 10.9.8.7

apiserver

kube-proxy

iptables

Linux

Client

connect to
localhost:X

pod1
10.240.1.1 : 9376

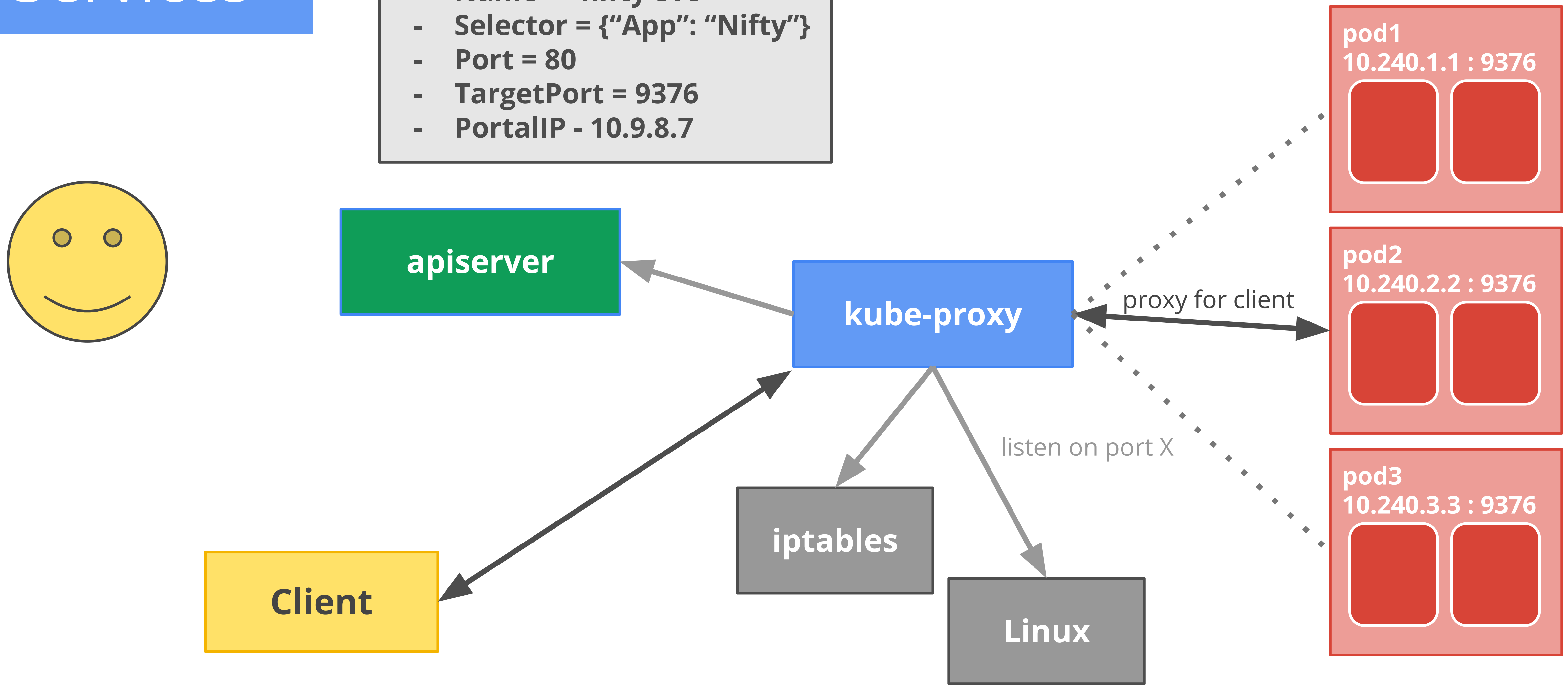
pod2
10.240.2.2 : 9376

pod3
10.240.3.3 : 9376

Services

Service

- **Name = “nifty-svc”**
- **Selector = {“App”: “Nifty”}**
- **Port = 80**
- **TargetPort = 9376**
- **PortalIP - 10.9.8.7**



Concept: Volumes

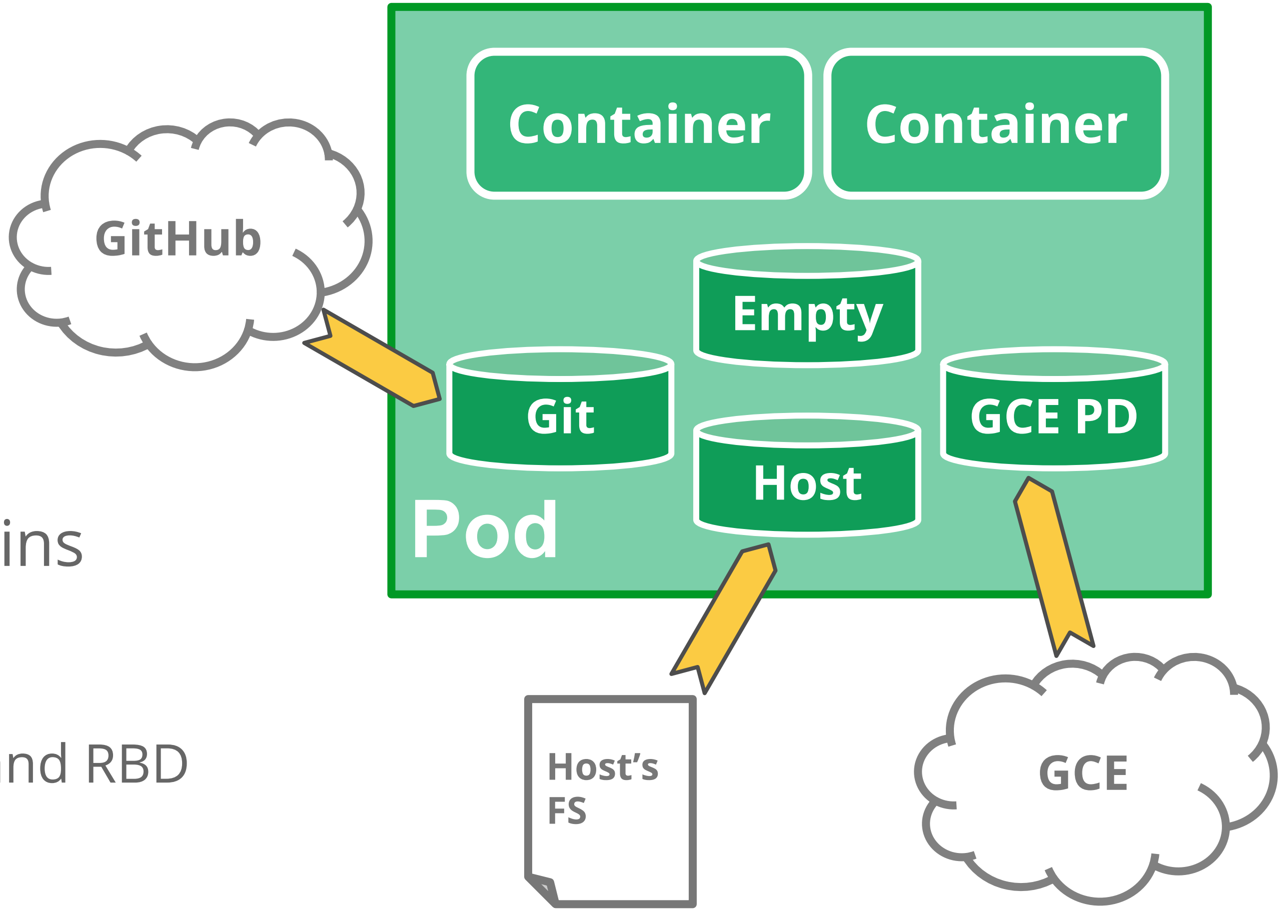
Very similar to Docker's concept

Pod scoped storage

Share the pod's lifetime & fate

Support many types of volume plugins

- Empty directory
- Host path
- Git repository
- GCE Persistent Disk
- AWS Elastic Block Store
- iSCSI
- NFS
- GlusterFS
- Ceph File and RBD
- Cinder
- …



New: Persistent Volumes

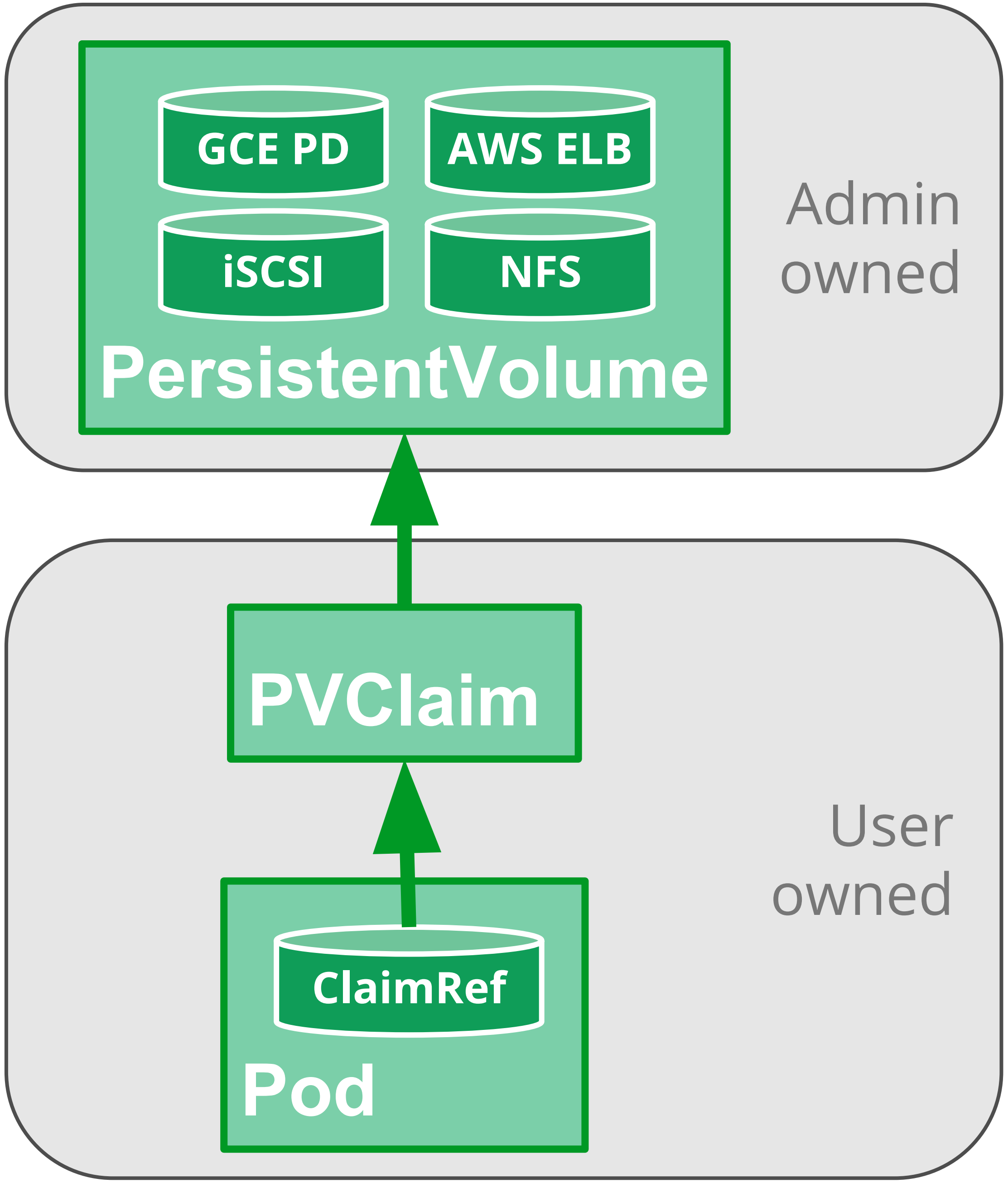
A higher-level abstraction - insulation from any one cloud environment

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



Docker, Rocket, LXC, Oh my!

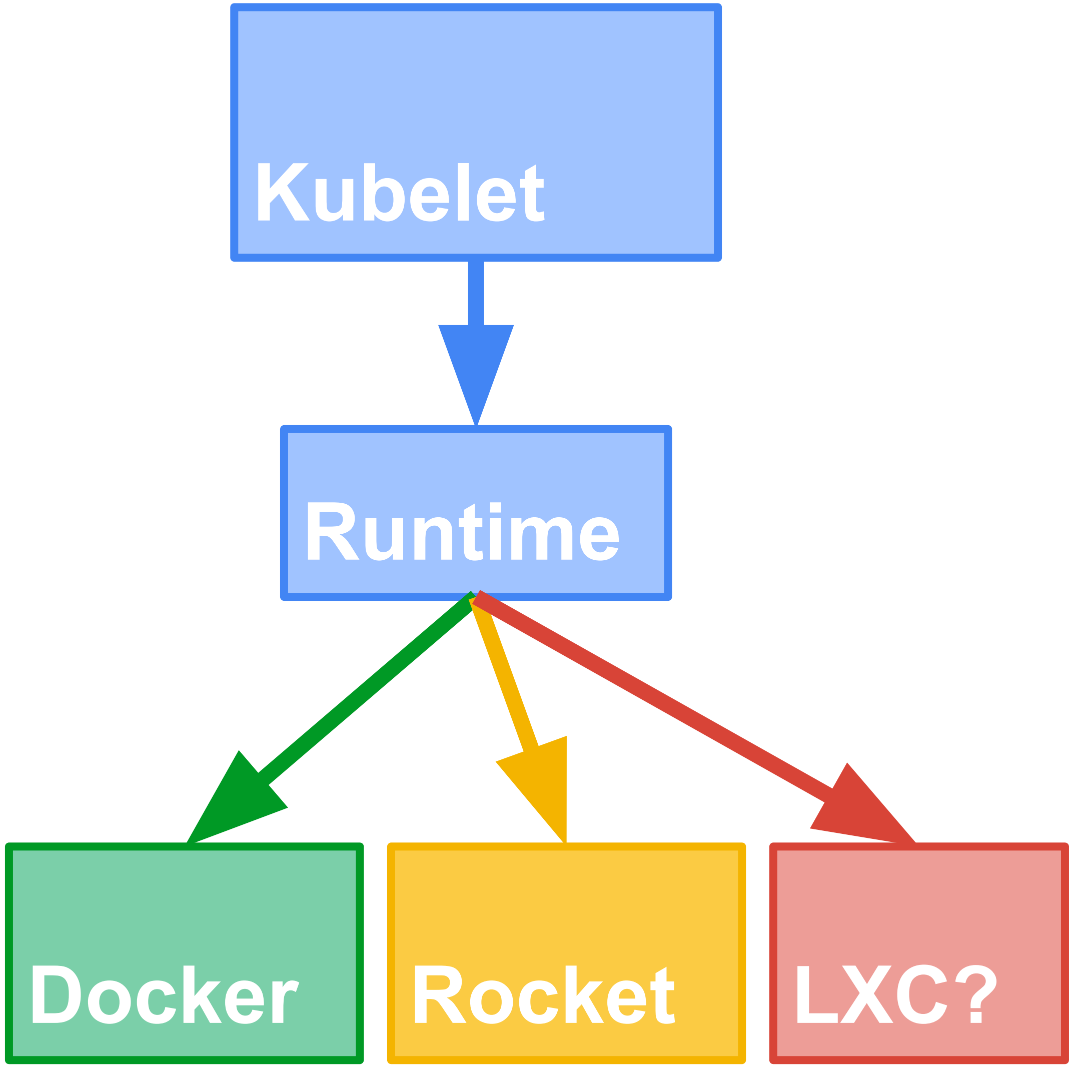
Currently built on Docker

Work is in progress to abstract that (a bit) into a Runtime abstraction

Interest in Rocket and LXC support

Rocket support is in flight (we like plugins)

Dynamically “scheduled” and managed, like nodes and pods



What else is in new?

- Network plugins
- Secrets
- Graceful termination
- Quota
- More volumes
- Downward API
- More platforms

- Performance
- Scalability
- High availability masters
- Scheduling
- Cluster federation
- Multi-cloud
- Easier setup

Kubernetes status & plans

Open sourced in June, 2014

- won the 2014 BlackDuck “rookie of the year” award

Google Container Engine (GKE)

- hosted Kubernetes - don’t think about cluster setup

Red Hat: OpenShift 3

- open PaaS on Kubernetes

CoreOS: Tectonic

- ready-to-run Kubernetes - don’t think about cluster setup

Mirantis: Murano

- Kubernetes and OpenStack

Driving towards a 1.0 release in O(months)

Roadmap:

- https://github.com/GoogleCloudPlatform/kubernetes/blob/master/docs/roadmap.md

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 to ideas
- open source

http://kubernetes.io

https://github.com/GoogleCloudPlatform/kubernetes

irc.freenode.net *#google-containers*

@kubernetesio