Kubernetes Standard API

Fabrice Jammes, http://k8s-school.fr

Kubernetes

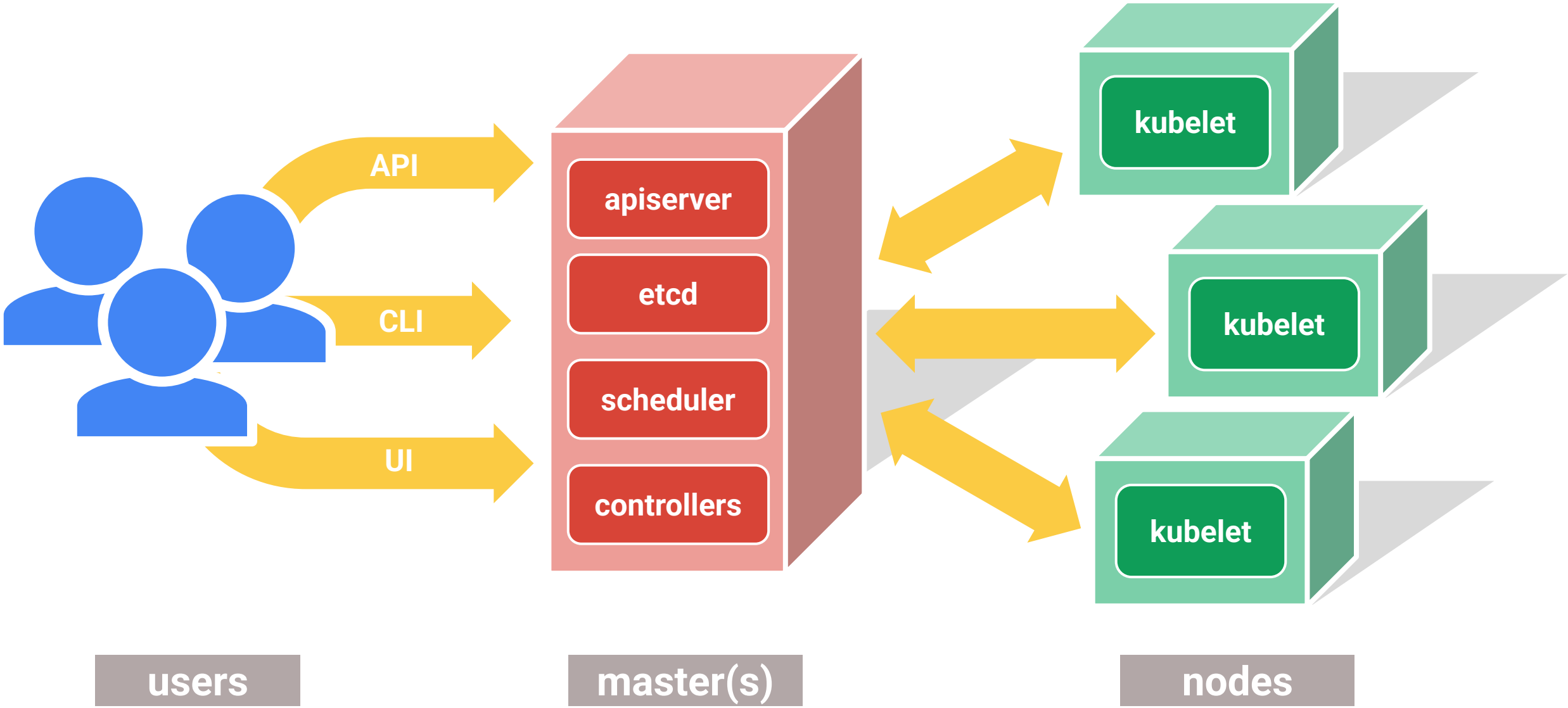
Greek for *“Helmsman”*; also the root of the words *“governor”* and *“cybernetic”*

- 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 applications, not machines

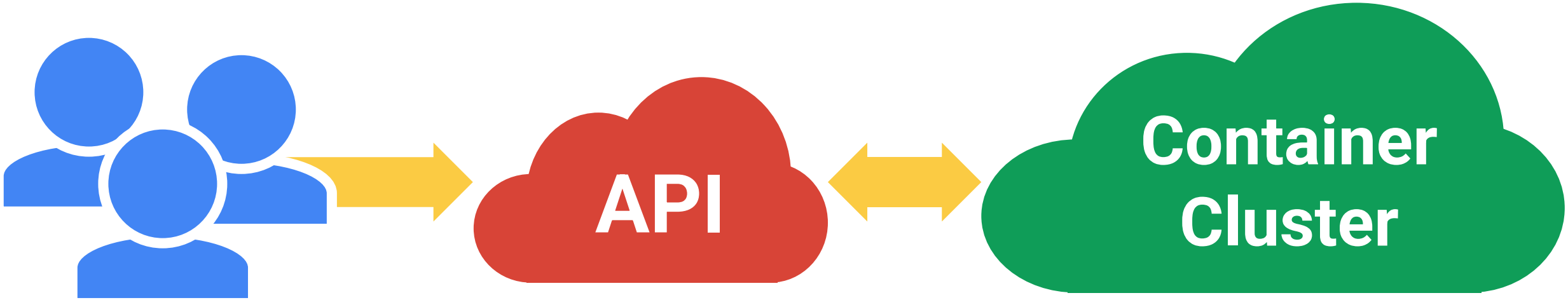
http://k8s-school.fr

The 10000 foot view



http://k8s-school.fr

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
- ReplicaSets
- 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

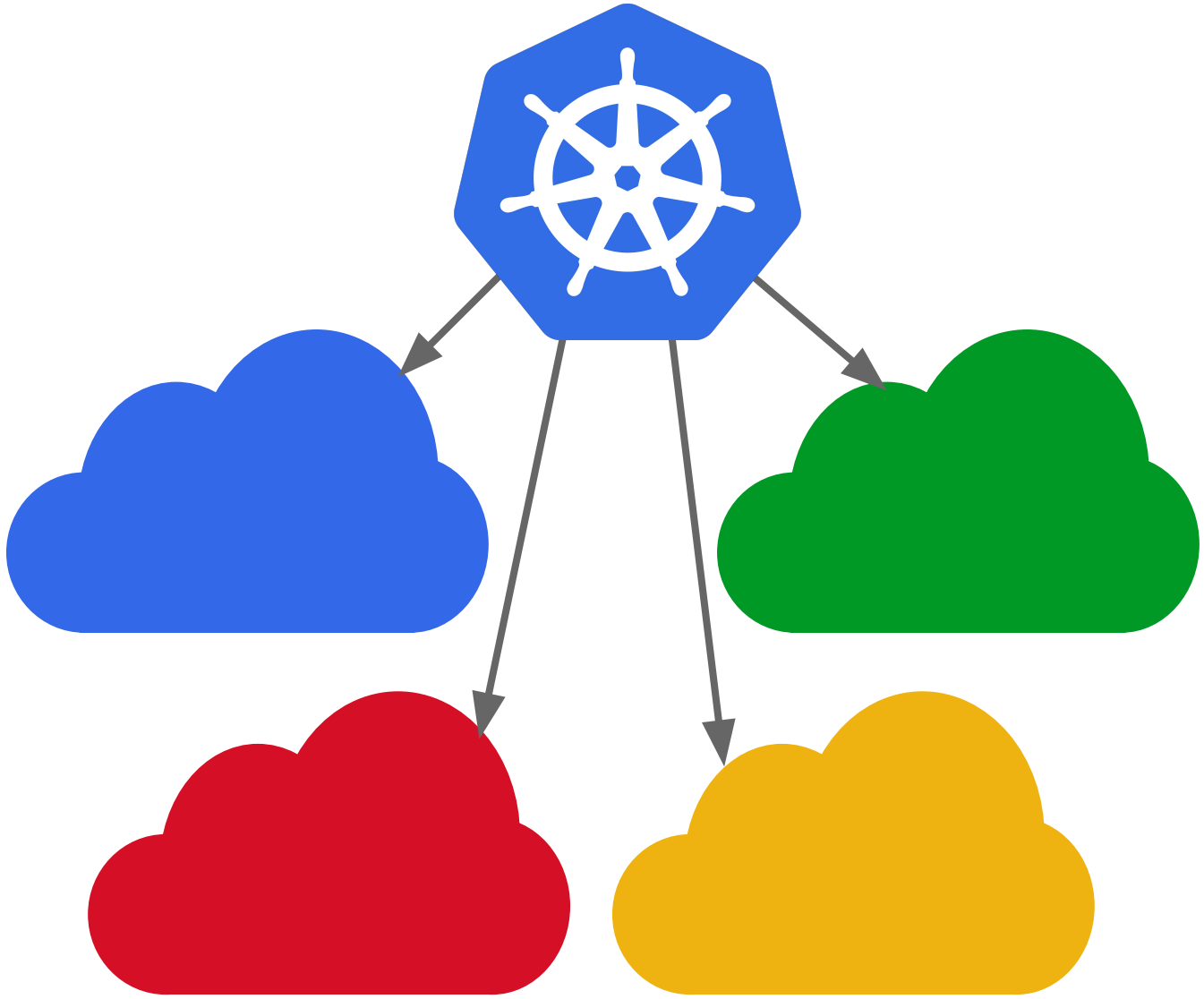
Workload portability

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



http://k8s-school.fr

Workload portability

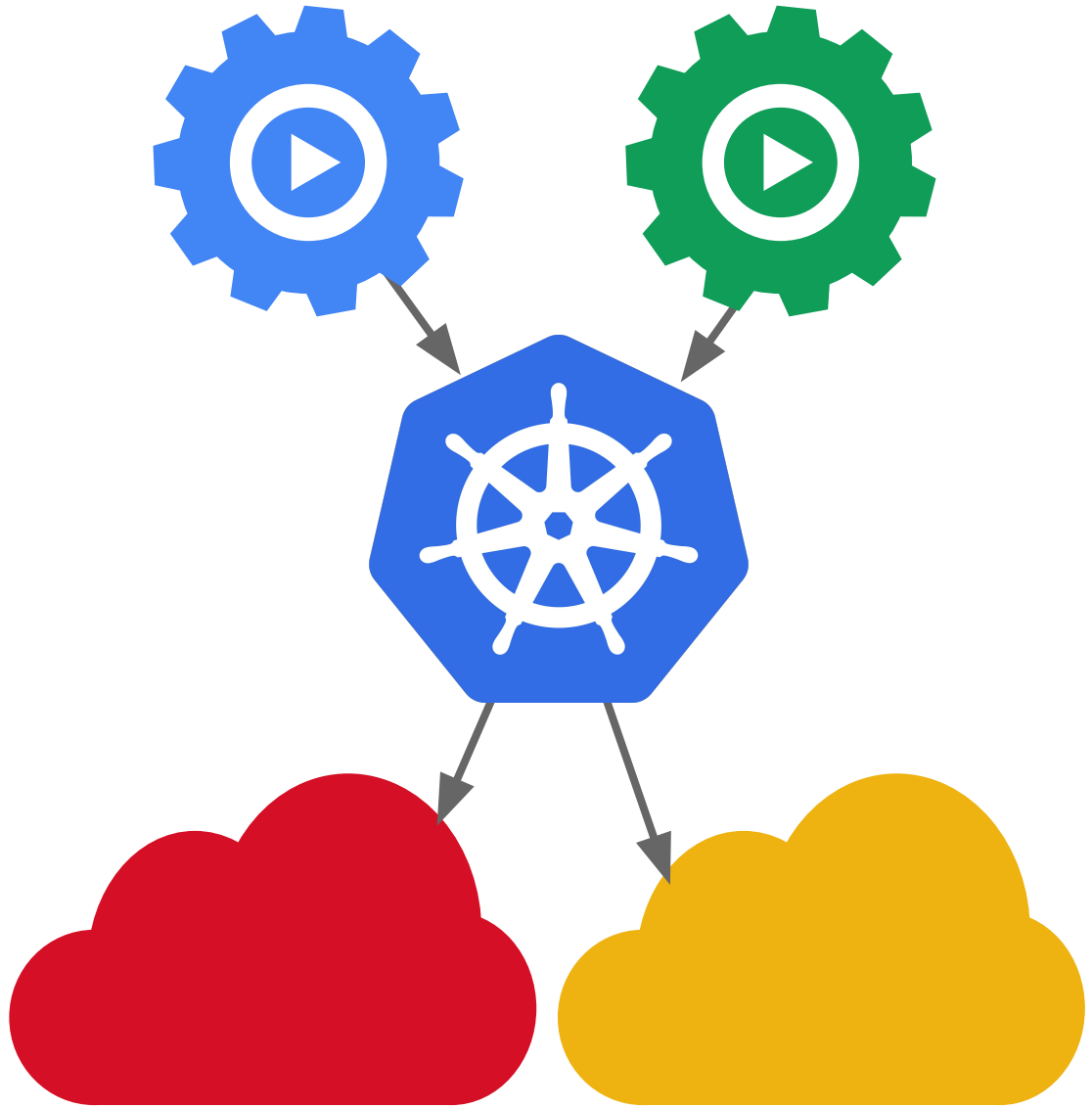
Goal: Write once, run anywhere*

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

Examples of this:

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

** approximately*



http://k8s-school.fr

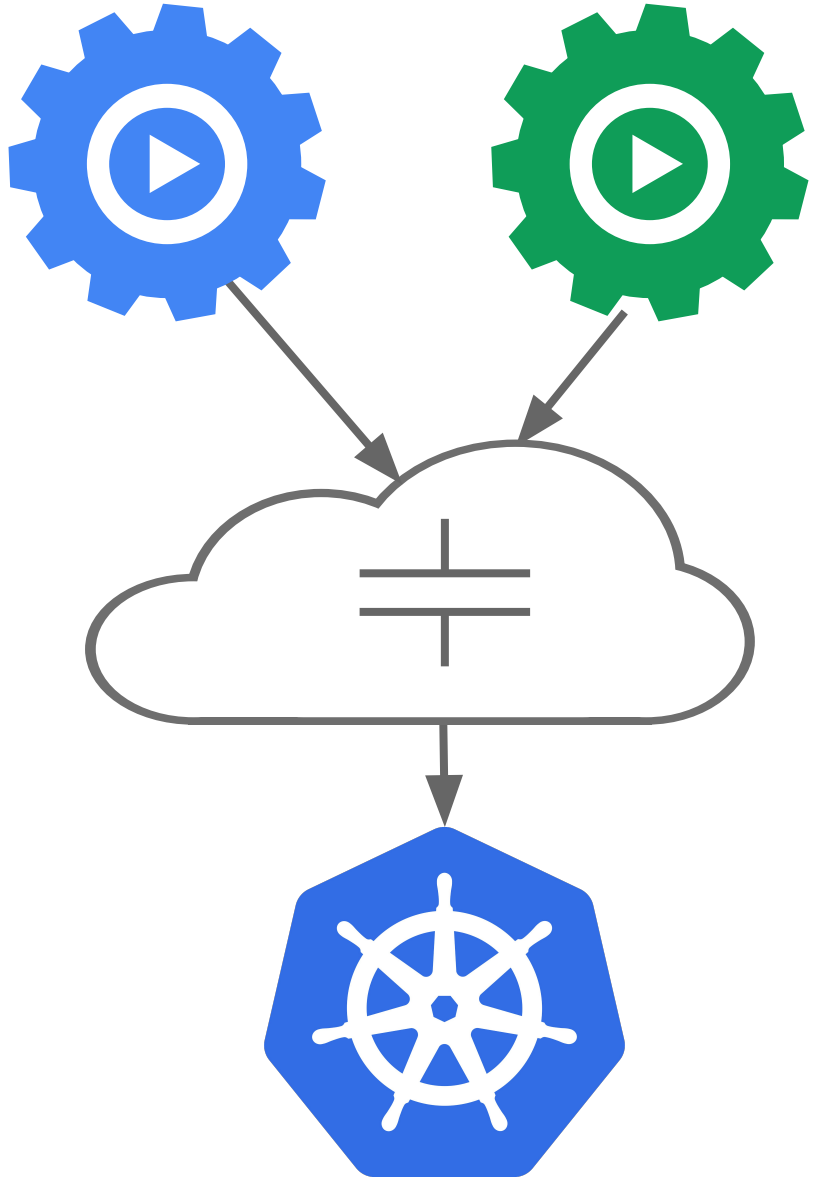
Workload portability

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



http://k8s-school.fr

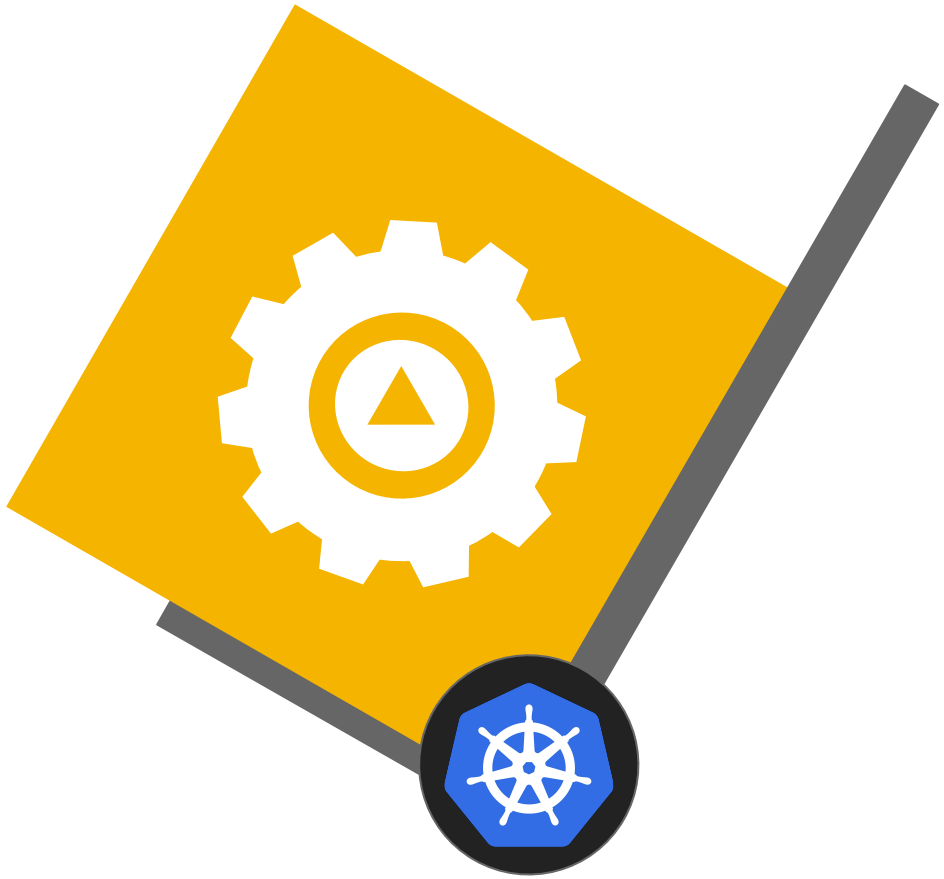
Workload portability

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



http://k8s-school.fr

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

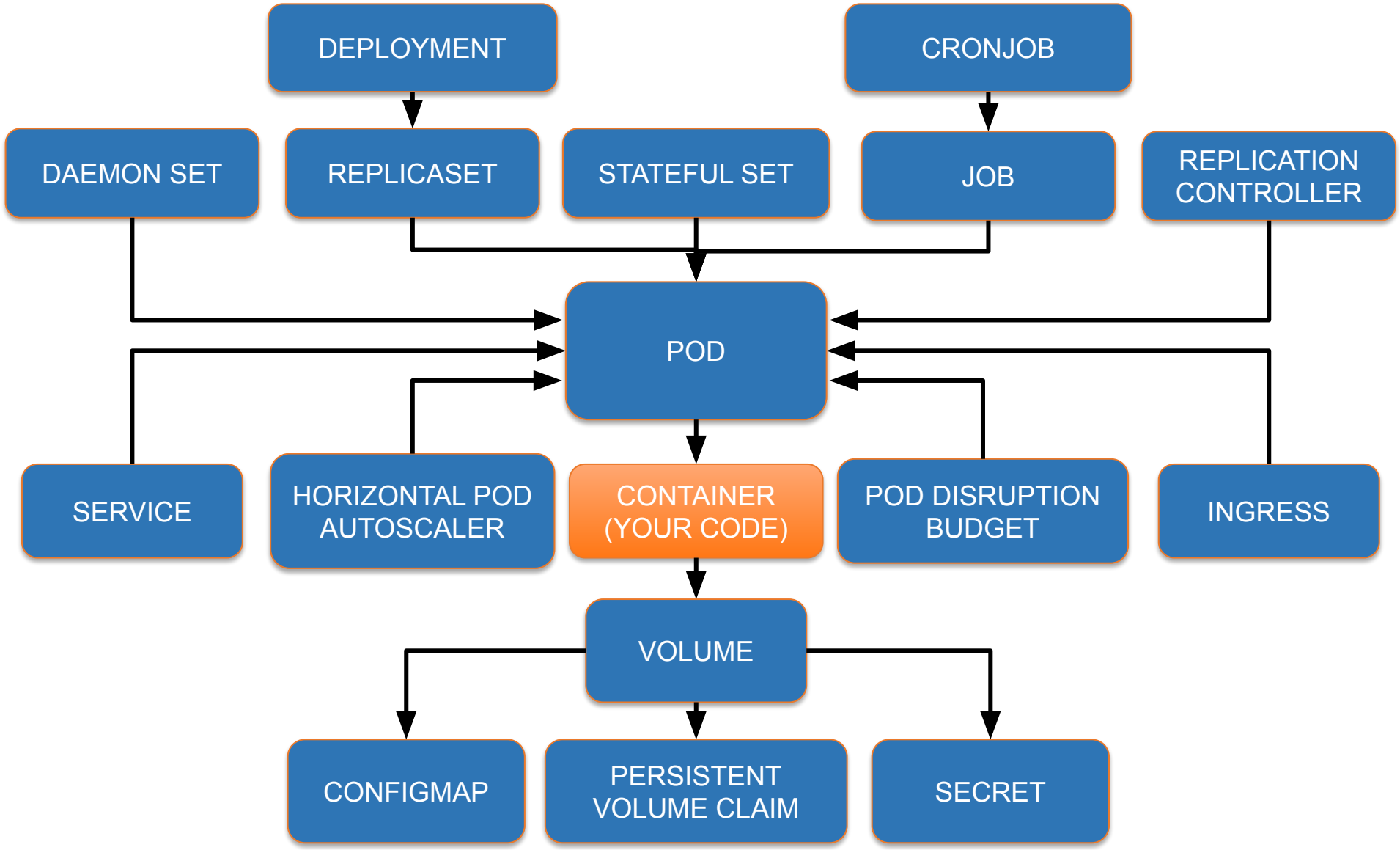
example: load-balanced backends

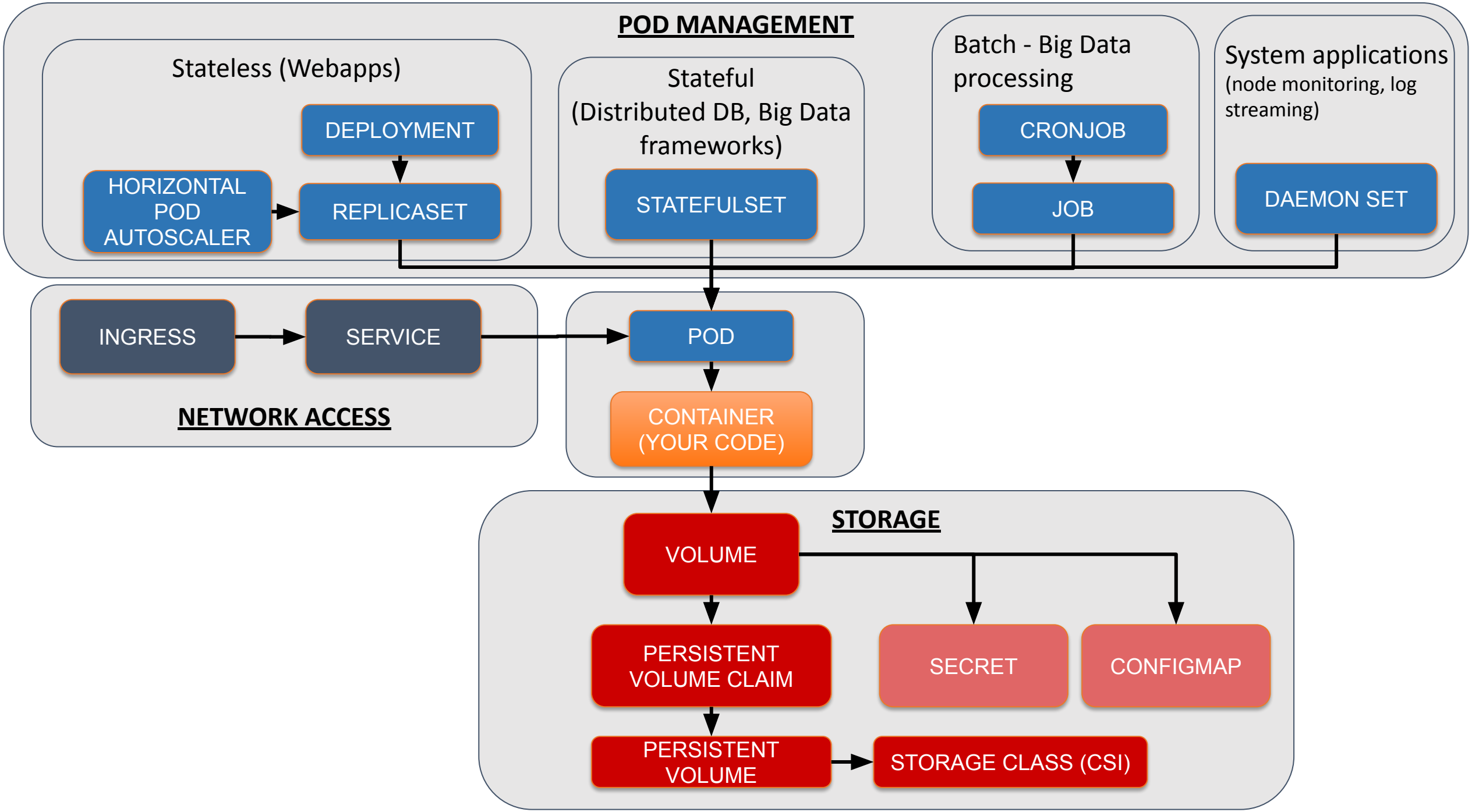
Labels: Identifying metadata attached to other objects

example: phase=canary vs. phase=prod

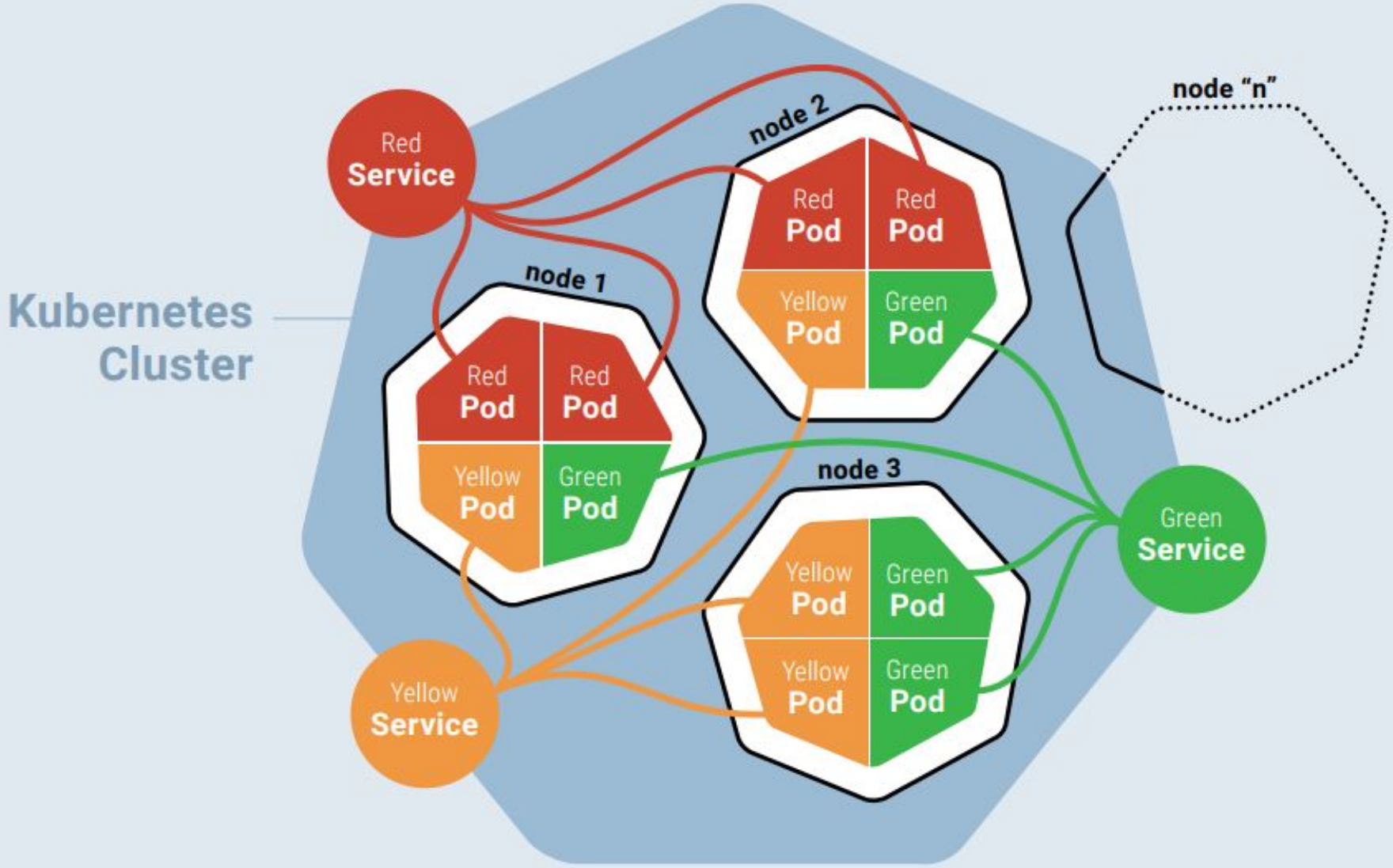
Selector: A query against labels, producing a set result

example: all pods where label phase == prod


DEPLOYMENT
CRONJOB
DAEMON SET
REPLICASET
STATEFUL SET
JOB
REPLICATION CONTROLLER
POD
SERVICE
HORIZONTAL POD AUTOSCALER
CONTAINER (YOUR CODE)
POD DISRUPTION BUDGET
INGRESS
VOLUME
CONFIGMAP
PERSISTENT VOLUME CLAIM
SECRET


POD MANAGEMENT
Stateless (Webapps)
DEPLOYMENT
HORIZONTAL POD AUTOSCALER
REPLICASET
Stateful (Distributed DB, Big Data frameworks)
STATEFULSET
Batch - Big Data processing
CRONJOB
JOB
System applications (node monitoring, log streaming)
DAEMON SET
INGRESS
SERVICE
NETWORK ACCESS
POD
CONTAINER (YOUR CODE)
STORAGE
VOLUME
PERSISTENT VOLUME CLAIM
SECRET
CONFIGMAP
PERSISTENT VOLUME
STORAGE CLASS (CSI)

Services, Pods and Nodes



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

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

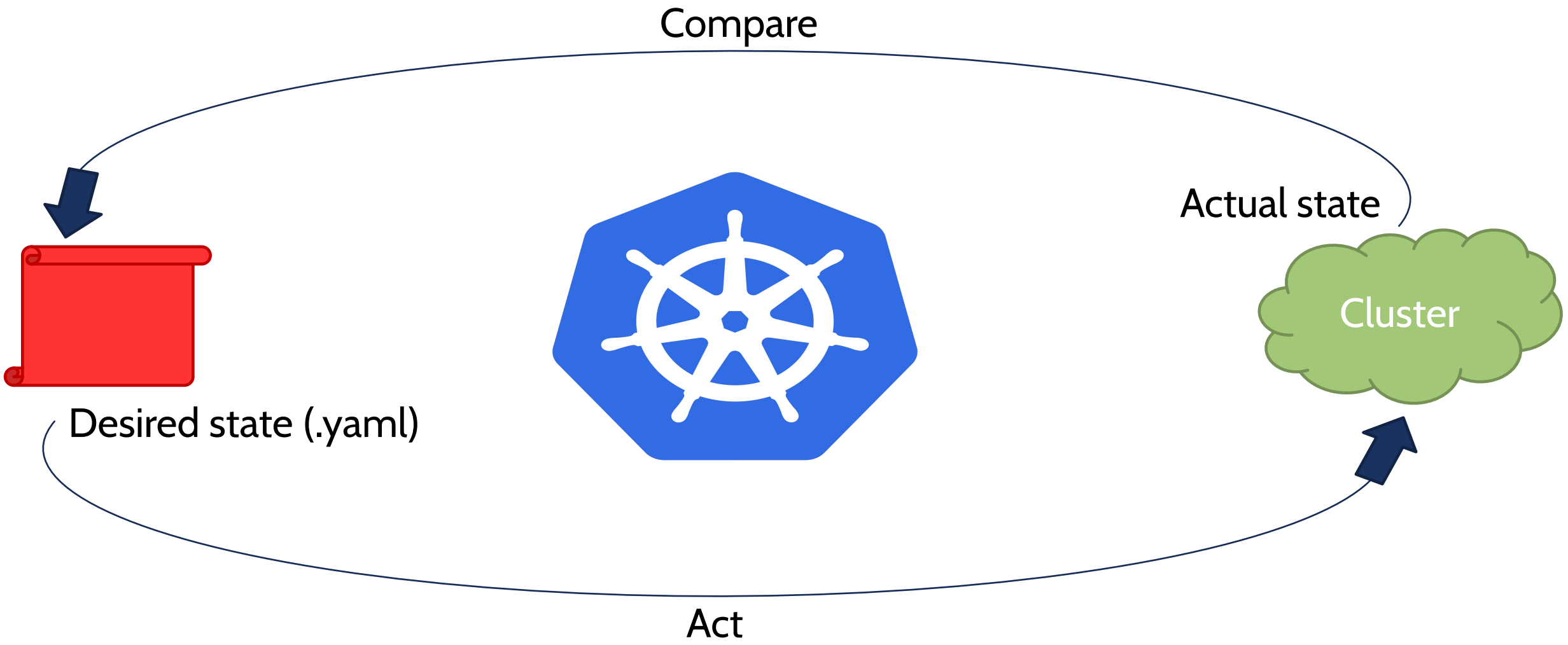
Network-centric: IP addresses are cheap

No grouping: Labels are the only groups

Cattle > Pets: Manage your workload in bulk

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

The reconciliation loop



Control loops

Drive **current state -> desired state**

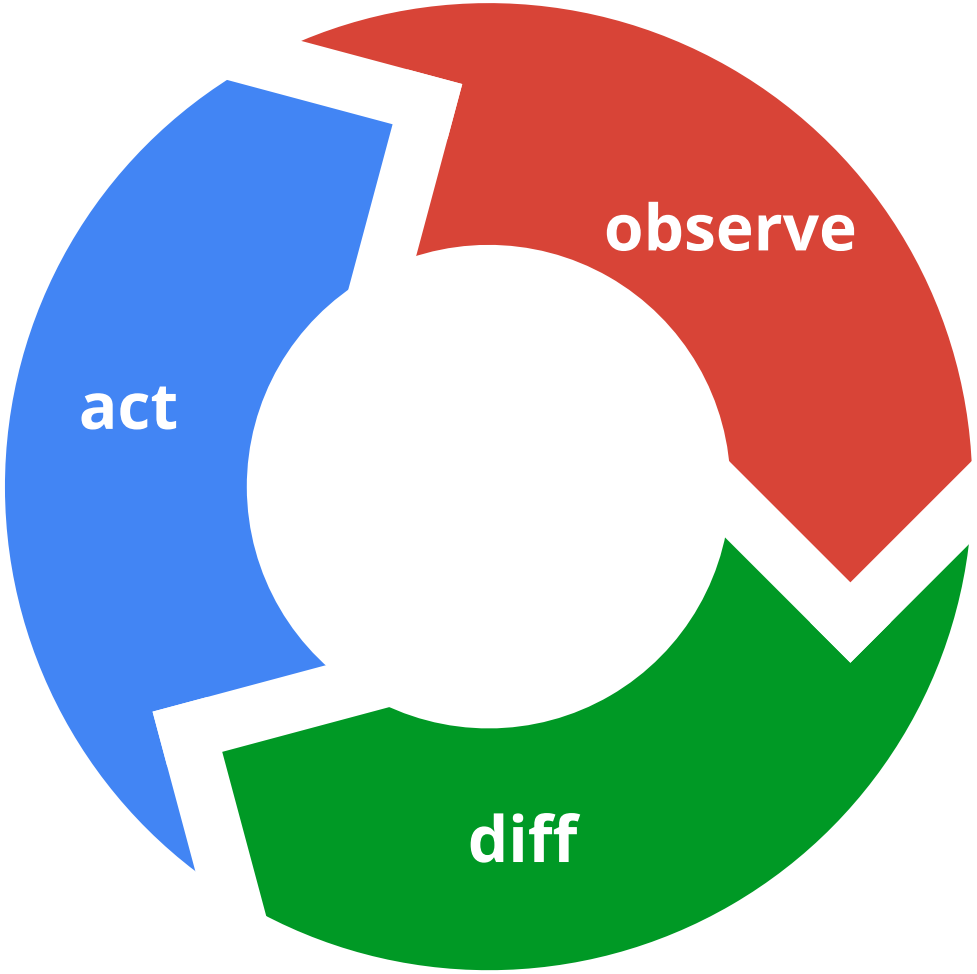
Act independently

APIs - **no shortcuts** or back doors

Observed state is truth

Recurring pattern in the system

Example: ReplicationController



Modularity

Loose coupling is a goal **everywhere**

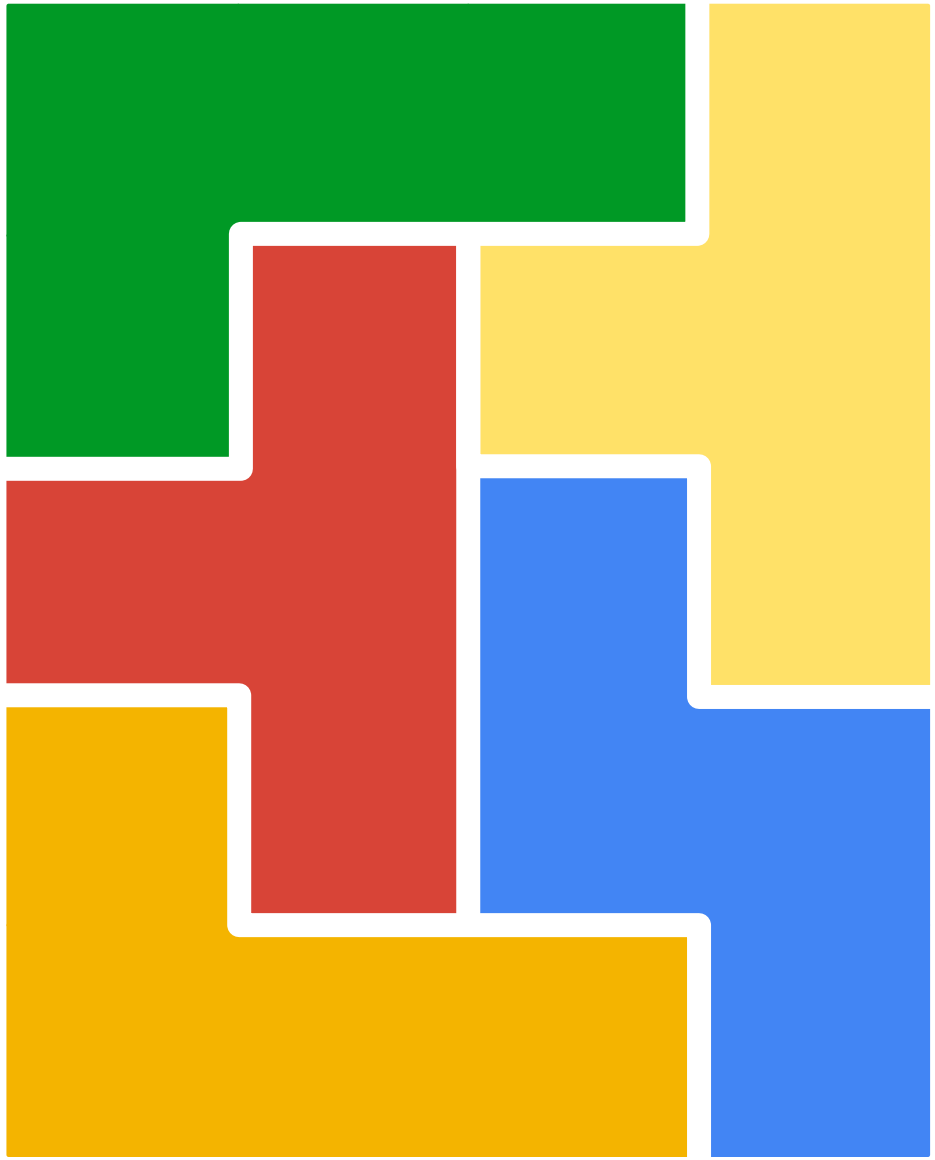
- simpler
- composable
- extensible

Code-level plugins where possible

Multi-process where possible

Isolate risk by interchangeable parts

Example: ReplicationController
Example: Scheduler



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)

Common kubectl commands

Common kubectl commands (1 / 2)

Get a given resource

```
kubectl get <resource-name> <obj-name> [-o yaml/json]
```

Describe a given resource

```
kubectl describe <resource-name> <obj-name>
```

Create/update resource(s) from file

```
kubectl apply -f obj.yaml
```

Edit resource in k8s database (i.e. etcd) (check https://12factor.net/codebase)

```
kubectl edit <resource-name> <obj-name>
```

Delete resource(s) from file

```
kubectl delete -f obj.yaml
```

Common kubectl commands (2 / 2)

Display inline documentation (and provide useful examples)

```
kubectl help create job
```

Describe yaml specification

```
kubectl explain pods.spec [--recursive]
```

See list of resources in use (require kubernetes-sigs/metrics-server)

```
kubectl top nodes
kubectl top pods
```

Debugging commands

Display logs for a container (i.e. stdout/stderr), see https://12factor.net/logs

```
kubectl logs <pod-name> [ -c <container-name> ]
```

Open an interactive shell inside a container

```
kubectl exec -it <pod-name> -- bash
```

Copy file to and from a container

```
kubectl cp <pod-name:/path/to/remote/file> </path/to/local/file>
```

Provide network access to a pod

```
# Listen on port 8080 locally, forwarding data to/from port 80
in the pod
kubectl port-forward pod/mypod 8080:80
```

Exercices: Common kubectl commands

1. **Create a pod** (use Kubernetes Documentation)

   ```
   kubectl apply -f obj.yaml
   ```

2. **Retrieve IP address for a pod in namespace kube-system**

```
kubectl get pods <my-pod> -o jsonpath
--template={.status.podIP}
```

3. **Play with K9s in k8s-toolbox**

TIPS

create a file without UI:

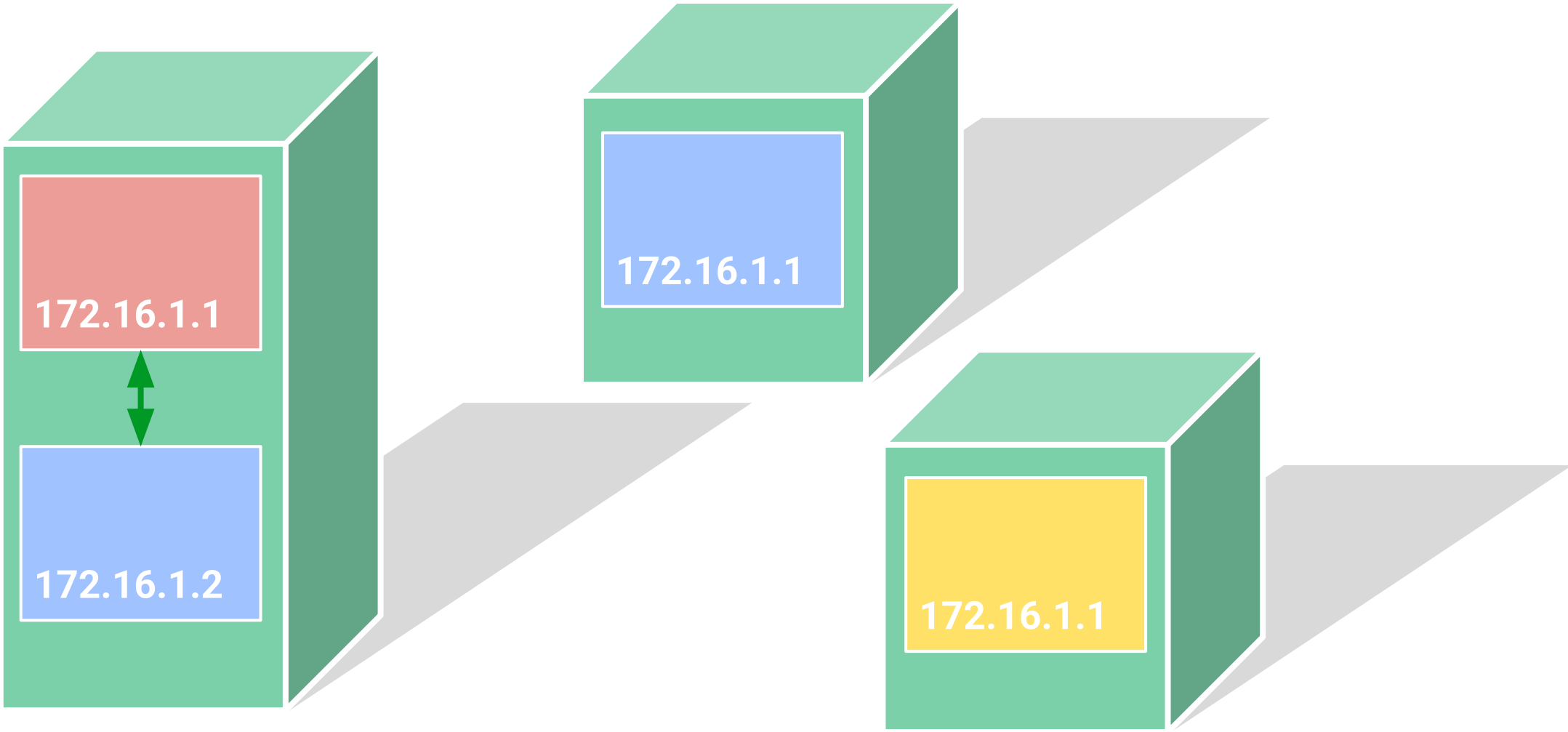
`cat > file.yaml` + Paste + Ctrl^D

enable autocompletion:

```
echo "source <(kubectl completion bash)" >> ${HOME}/.bashrc
```

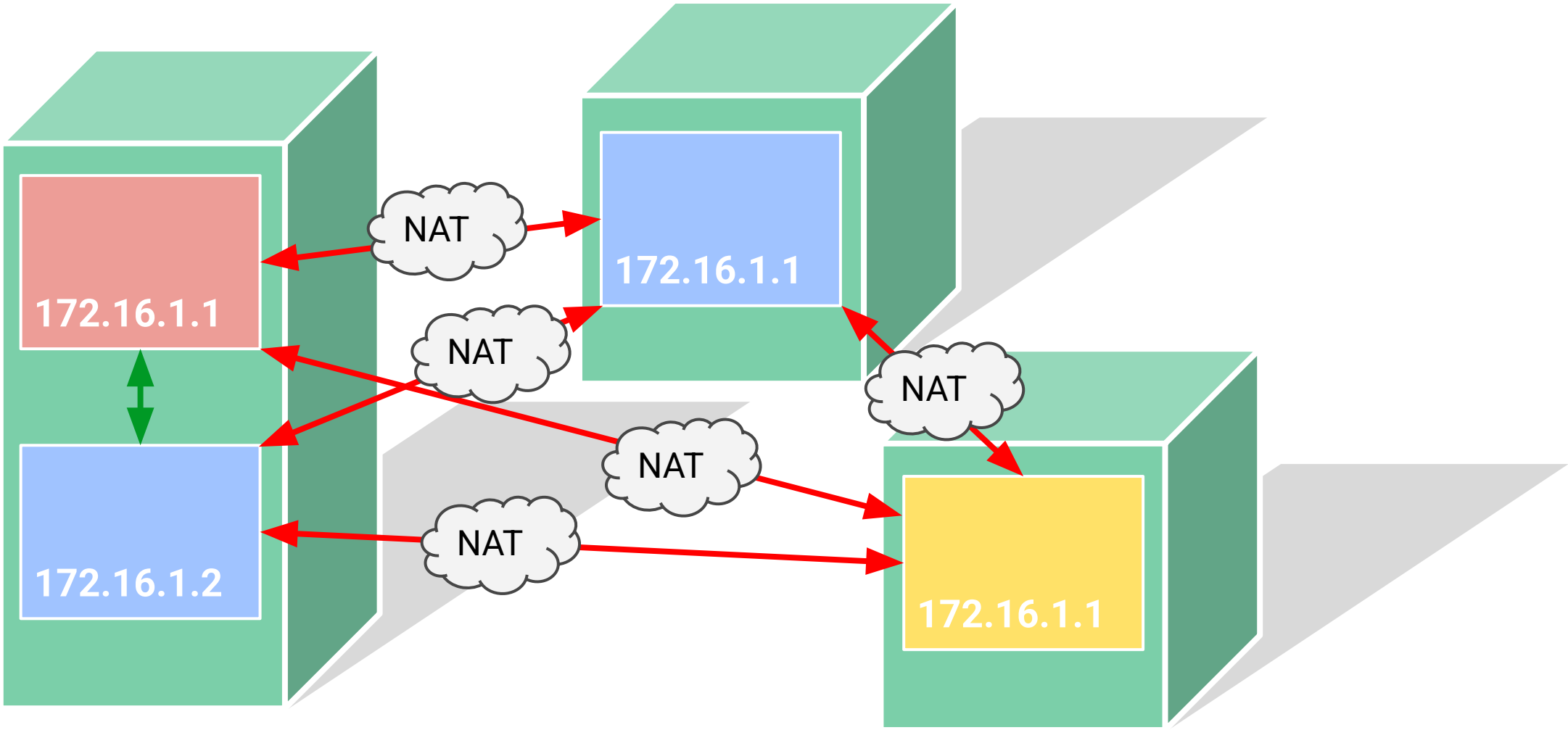
Networking

Docker networking (default)



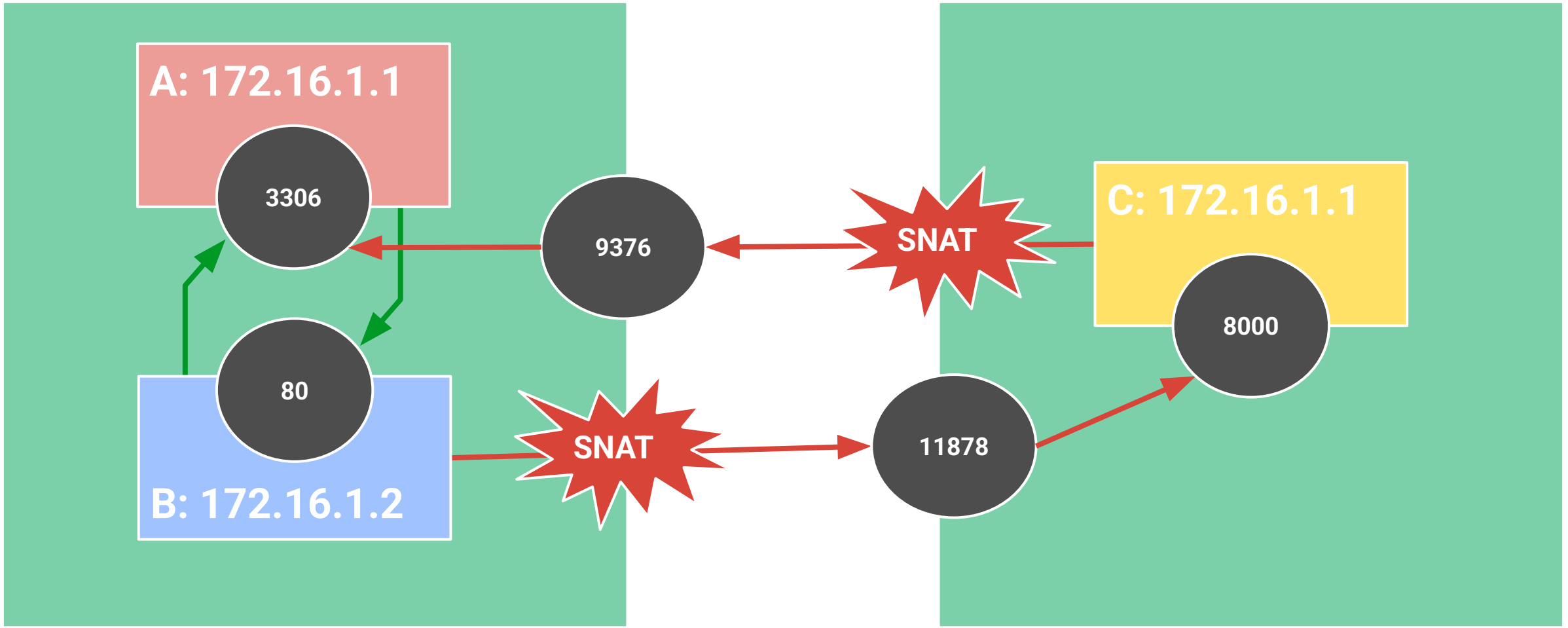
http://k8s-school.fr

Docker networking (default)



http://k8s-school.fr

Port mapping



http://k8s-school.fr

Kubernetes networking

IPs are **cluster-scoped**

Pods can reach each other directly

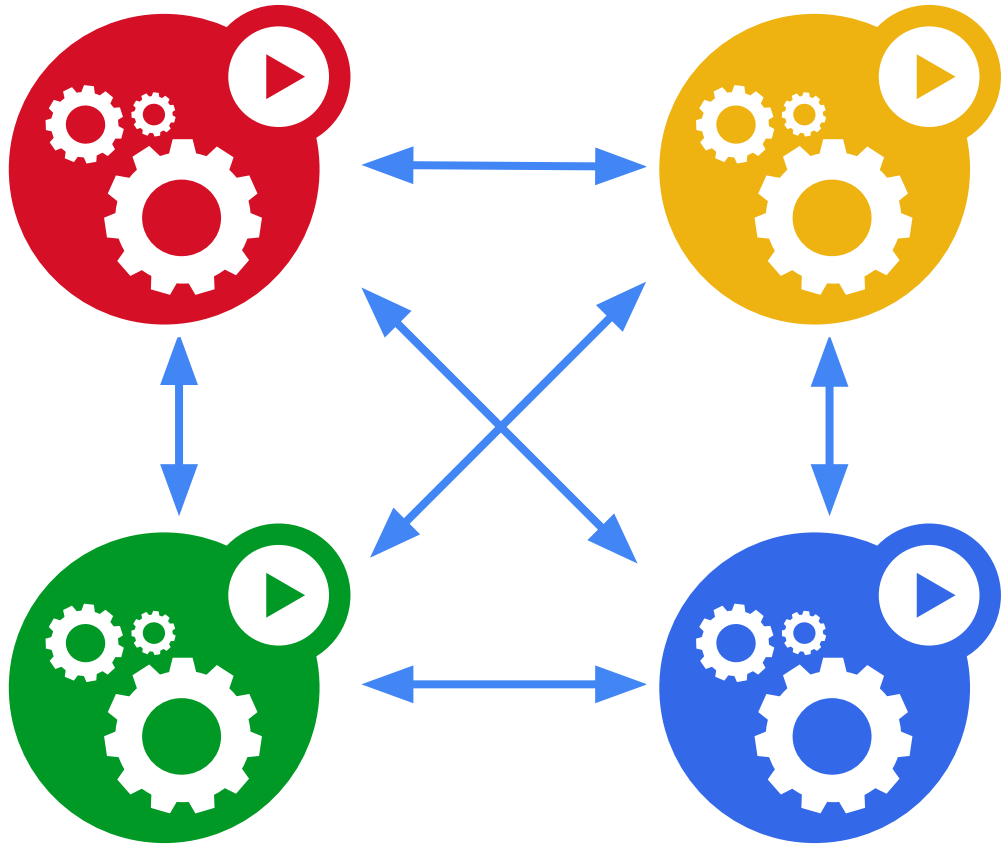
- even across nodes

No brokering of port numbers

- too complex, why bother?

This is a fundamental requirement

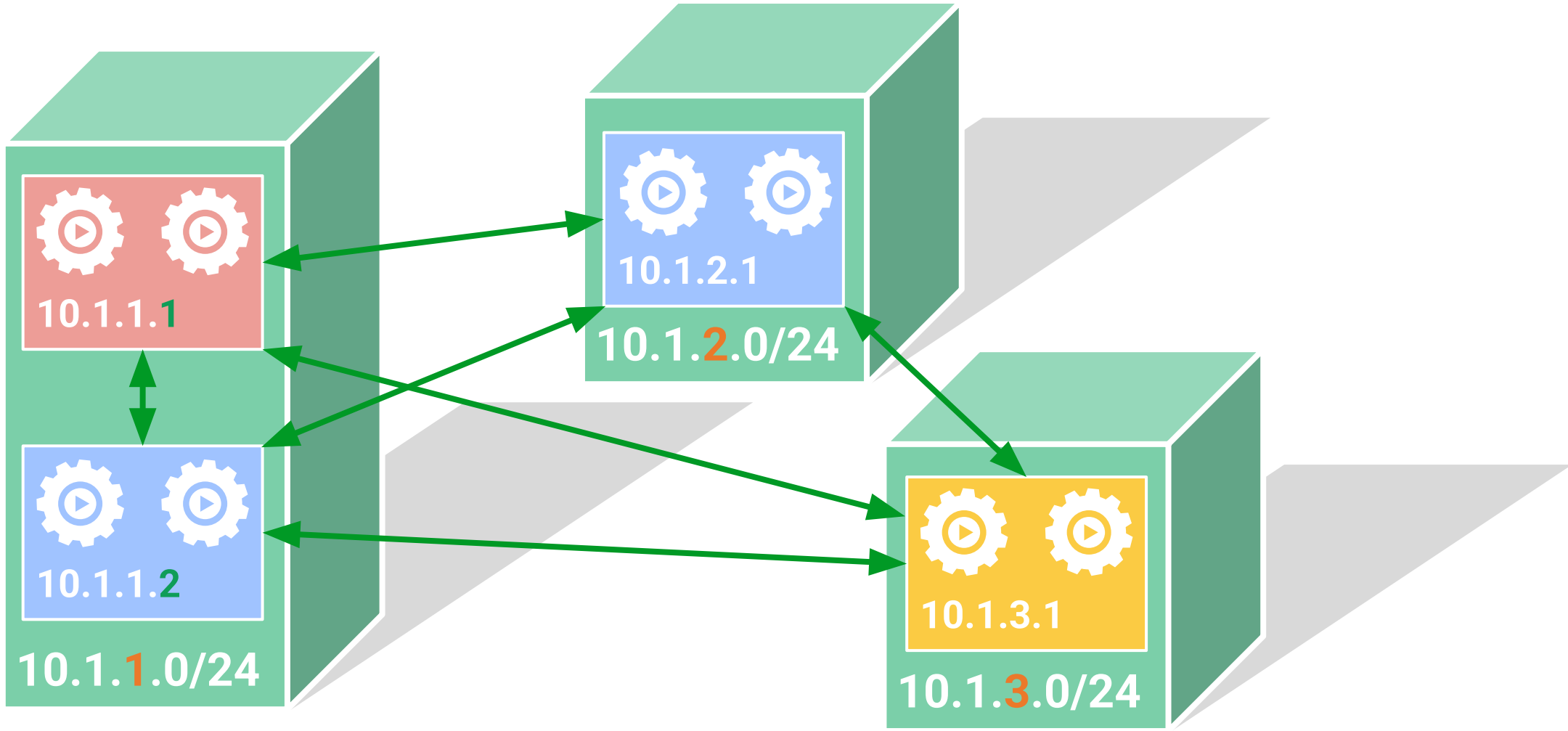
- can be L3 routed
- can be underlayed (cloud)
- can be overlayed (SDN)



http://k8s-school.fr

Kubernetes networking: CNI

Class B network `10.1.0.0/16`



Network Plugins

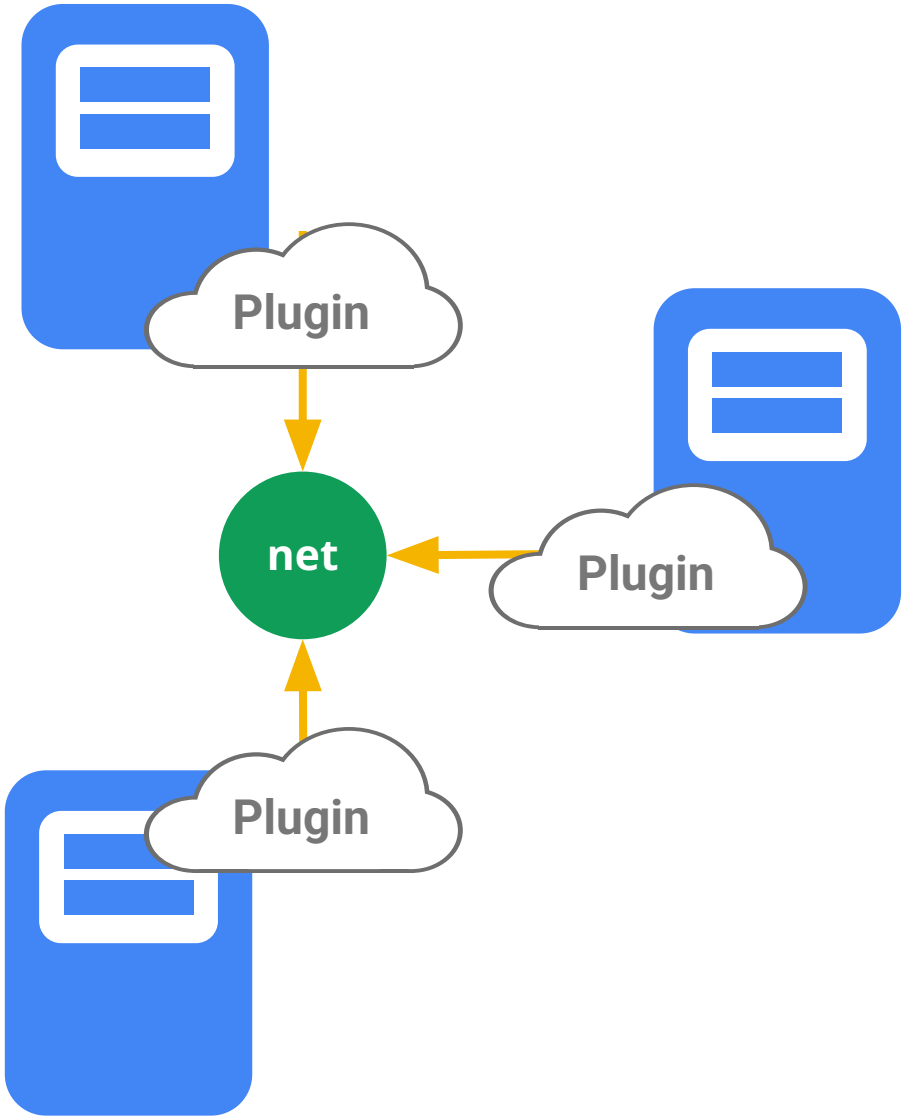
Network Plugins

Network plugin not included

Uses **CNI**:

- Simple exec interface
- Not using Docker libnetwork

Cluster admins can customize their installs



[Benchmark results of Kubernetes network plugins (CNI) over 40Gbit/s network [2024] | by Alexis Ducastel | ITNEXT]

Network Isolation

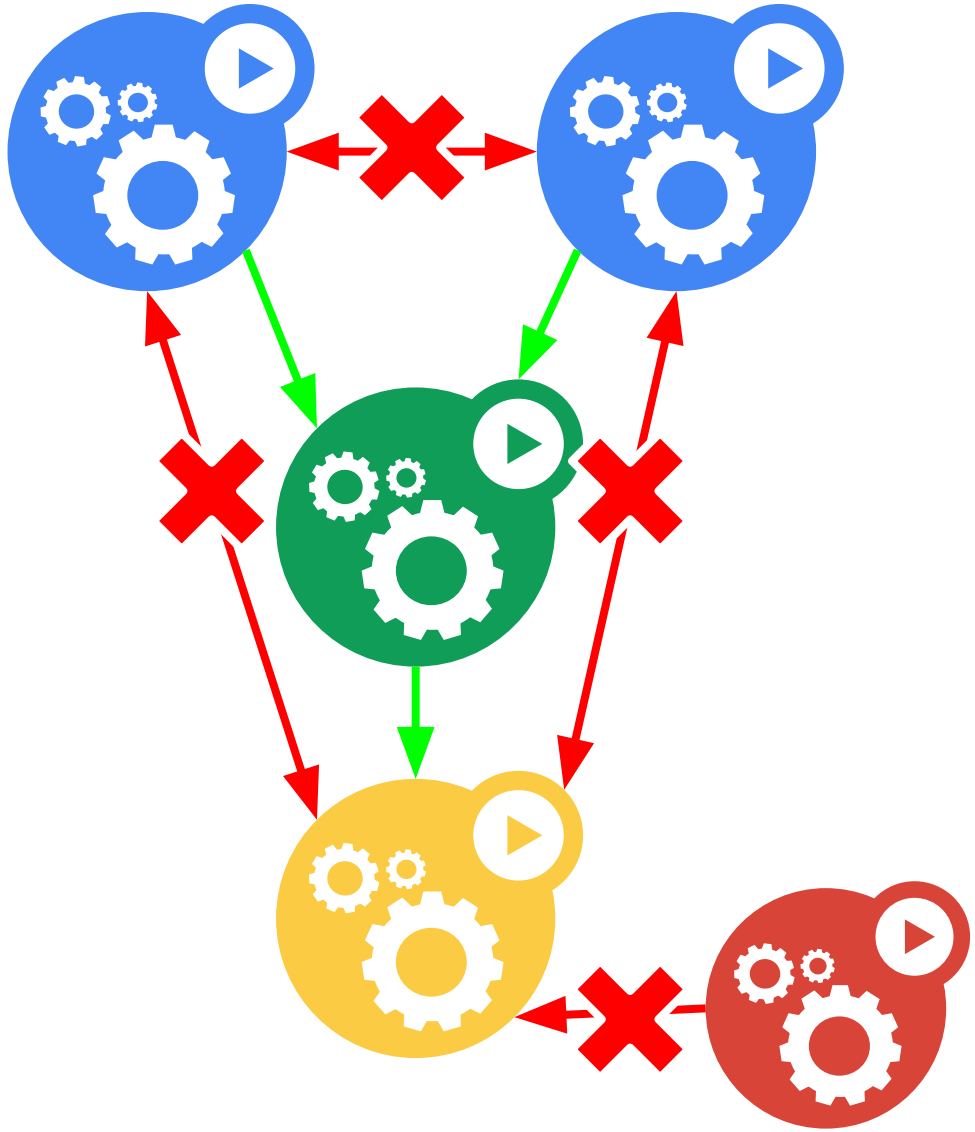
Network Isolation

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)



Pods

Pods

Small group of containers & volumes

Tightly coupled

The **atom** of scheduling & placement

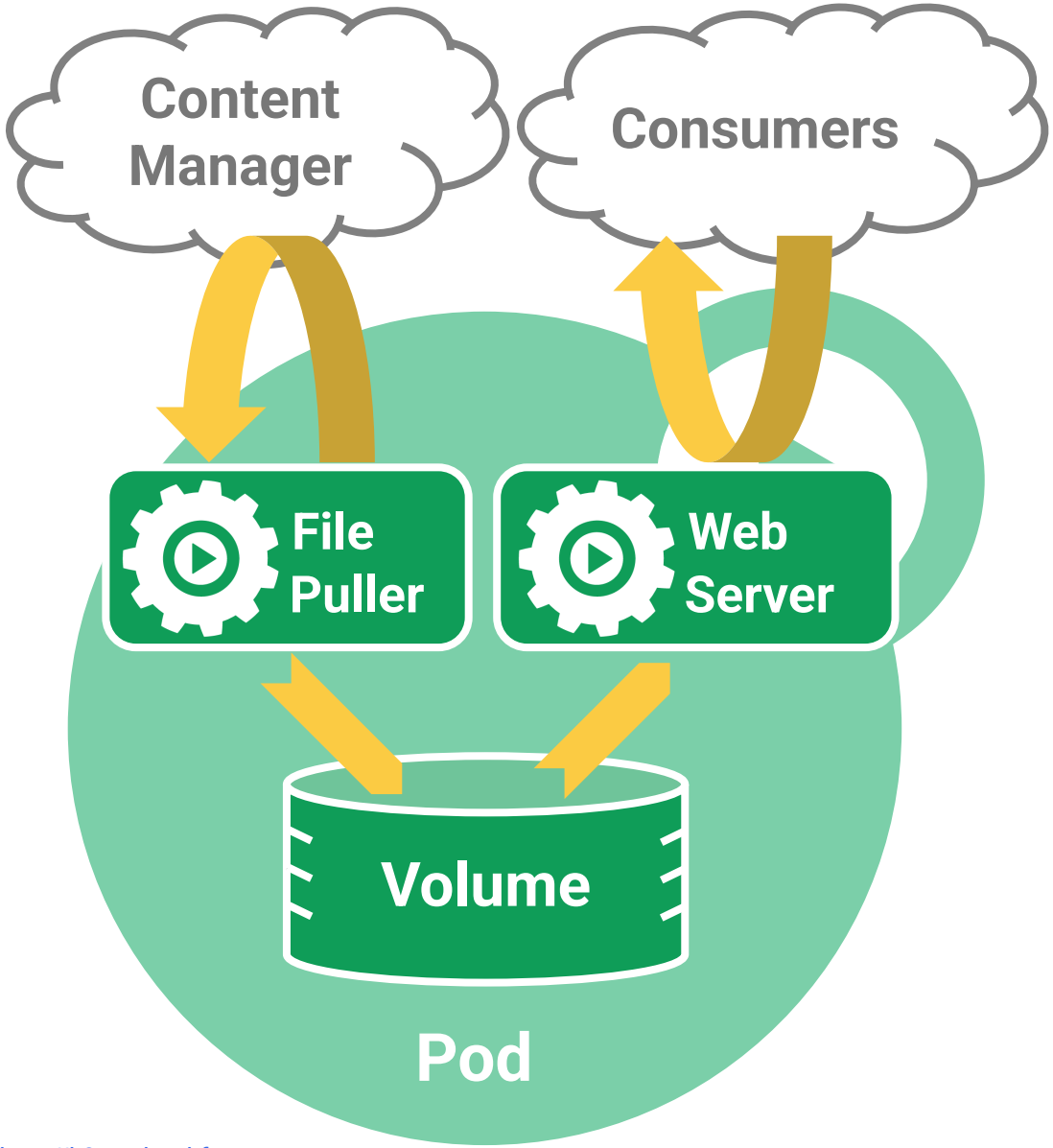
Shared namespace

- 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



http://k8s-school.fr

Exercice: Pods

Create a pod which use hostPath

`kubectl apply -f 5-5-kuard-pod-vol.yaml`

Create a file inside the hostPath volume and retrieve

`/var/lib/kuard/my-file` **and** `/data/my-file`

`(use docker exec -it kind-worker3 bash, instead of ssh)`

Port-forward to kuard pod

`kubectl port-forward --address 0.0.0.0 <pod-name> 808<ID>:8080 &`

Test it

`curl localhost:808<ID>`

Access kuard through web browser

`Use ssh tunnel to access kuard if needed, on workstation:`

`ssh k8s<ID>@51.15.223.252 -L 808<ID>:localhost:808<ID> -N`

Volumes

Pod-scoped storage

Support many types of volume plugins

- Empty dir (and tmpfs)
- Host path
- 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)
- ...

Liveness/Readiness probe

```
apiVersion: v1
kind: Pod
metadata:
 name: kuard
spec:
…
 containers:
   - image: gcr.io/kuar-demo/kuard-amd64:1
     name: kuard
     …
     livenessProbe:
       httpGet:
         path: /healthy
         port: 8080
       initialDelaySeconds: 5
       timeoutSeconds: 1
       periodSeconds: 10
       failureThreshold: 3
     readinessProbe:
       httpGet:
         path: /ready
         port: 8080
       initialDelaySeconds: 30
       timeoutSeconds: 1
       periodSeconds: 10
       failureThreshold: 3
```

If livenessProbe fails, kubelet will restart the container

WARNING: it will not restart the pod

One livenessProbe per container
3 methods
httpGet, tcpSocket, and exec

Liveness/Readiness probe

`kubectl explain pod.spec.containers.livenessProbe`
or Configure Liveness, Readiness and Startup Probes | Kubernetes

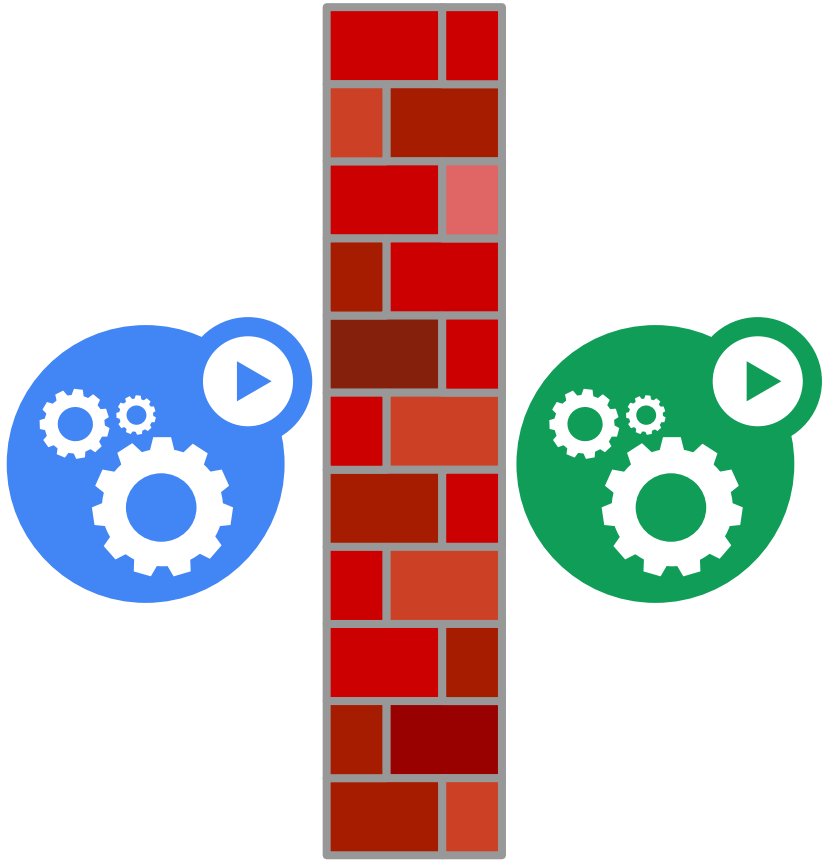
Exercice:
`kubectl apply -f 5-2-kuard-pod-health.yaml`
And check the kuard web UI

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

http://k8s-school.fr

Requests and Limits

```
apiVersion: v1
kind: Pod
metadata:
 name: kuard
spec:
…
 containers:
   - image: gcr.io/kuar-demo/kuard-amd64:1
     name: kuard
     …
     resources:
       requests:
         cpu: "500m"
         memory: "128Mi"
       limits:
         cpu: "1000m"
         memory: "256Mi"
```

Requests are guaranteed
Limits are upper bounds

Requests and Limits

```
kubectl explain pod.spec.containers.resources
```

Exercice:

```
kubectl apply -f 5-4-kuard-pod-reslim.yaml
```

And then allocate memory inside the kuard web UI

Quality of Service

Defined in terms of Request and Limit

Guaranteed: highest protection

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

Burstable: medium protection

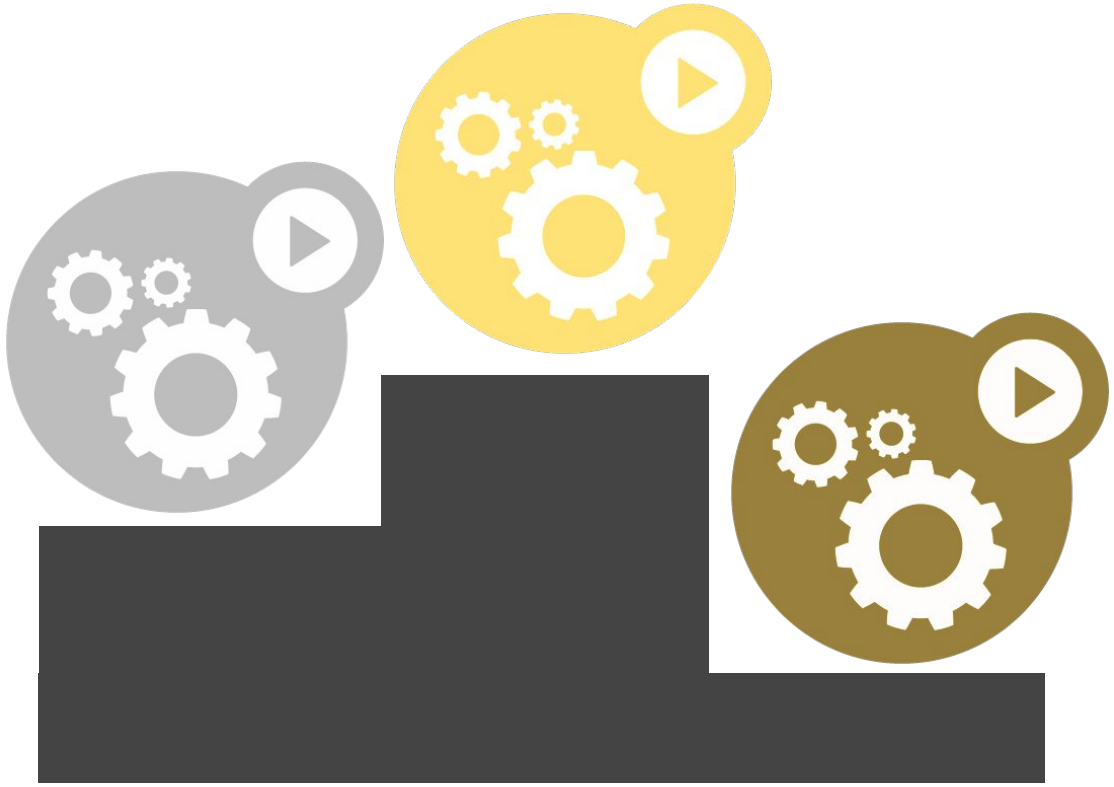
- `request > 0 && limit > request`

Best Effort: lowest protection

- `request == 0`

What does “protection” mean?

- OOM score
- CPU scheduling



Labels & Selectors

Labels

Arbitrary metadata

Attached to **any API object**

Generally represent **identity**

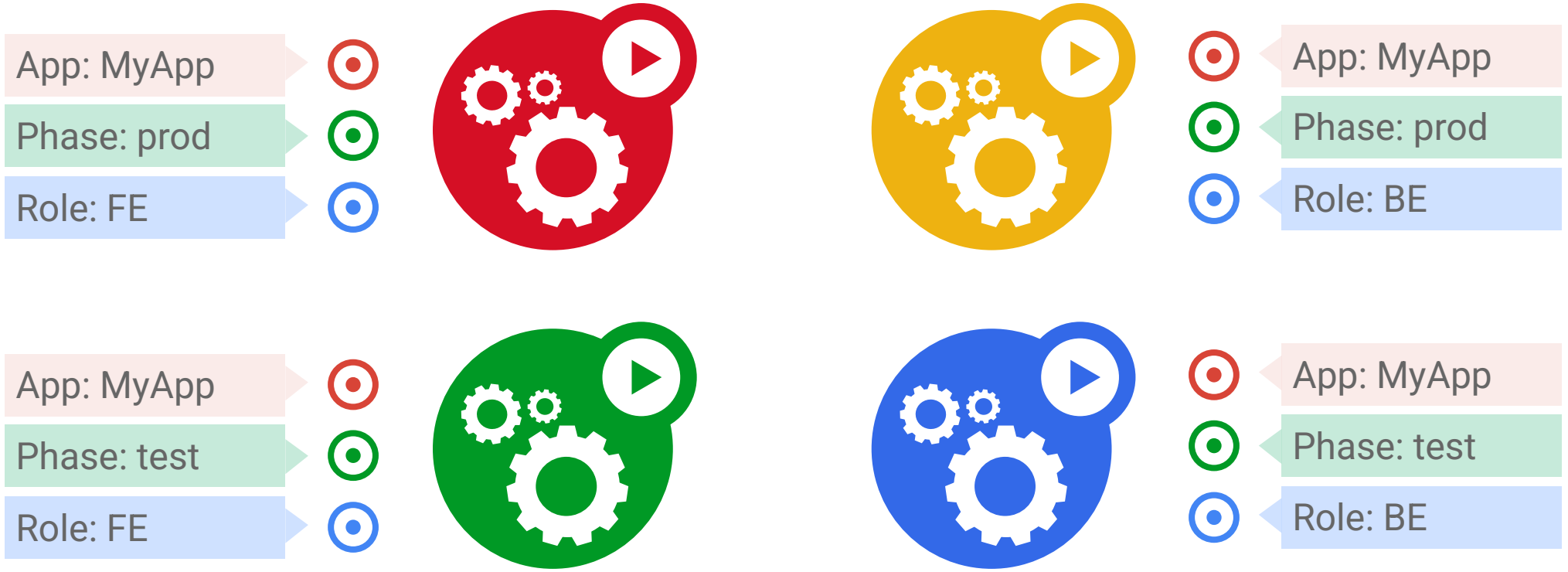
Queryable by **selectors**

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

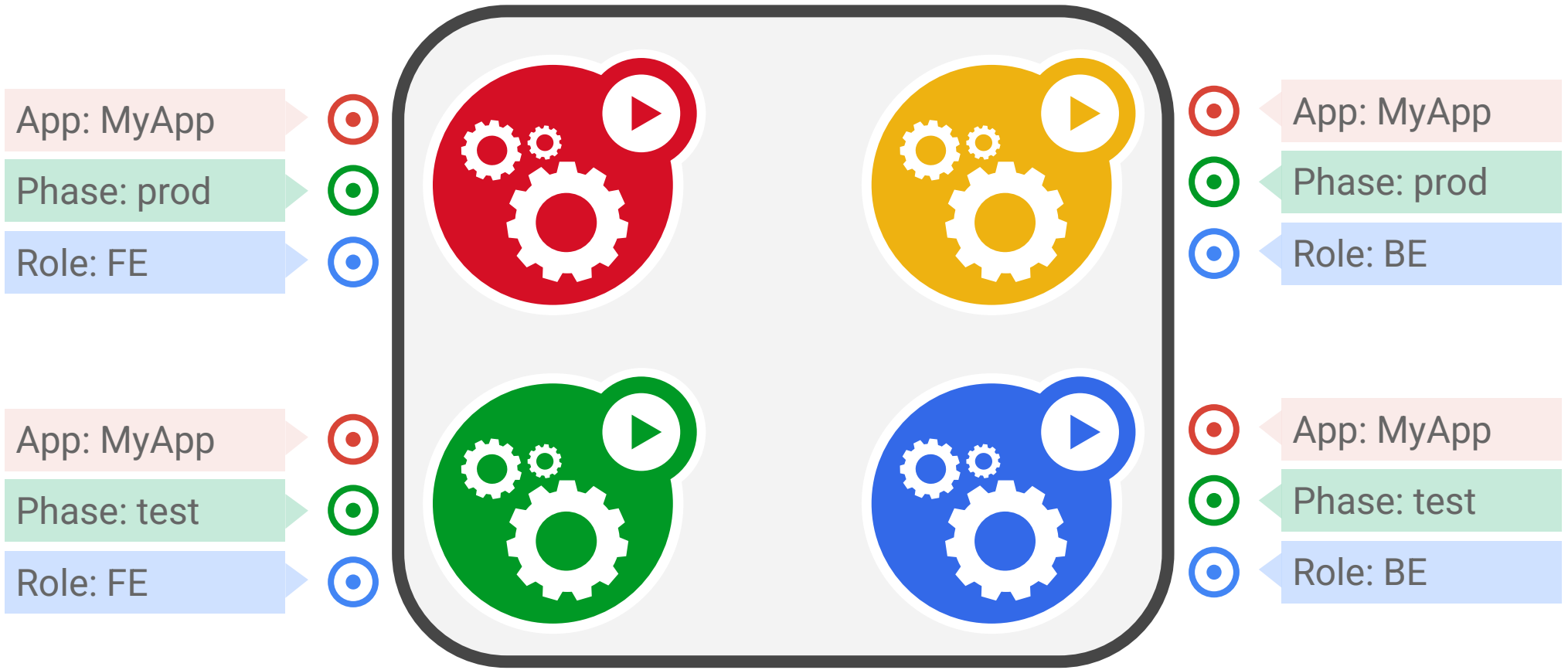
The **only** grouping mechanism

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

Selectors

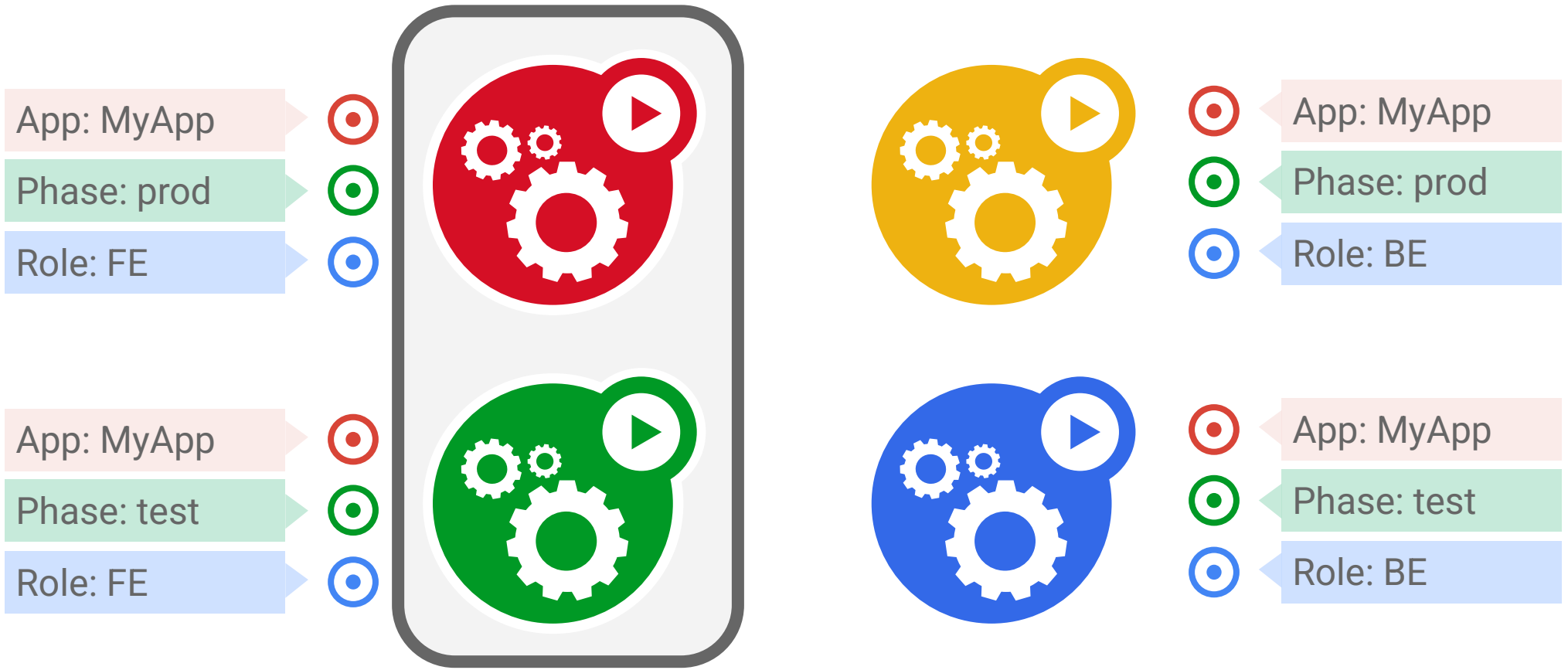

App: MyApp
Phase: prod
Role: FE
App: MyApp
Phase: prod
Role: BE
App: MyApp
Phase: test
Role: FE
App: MyApp
Phase: test
Role: BE

Selectors



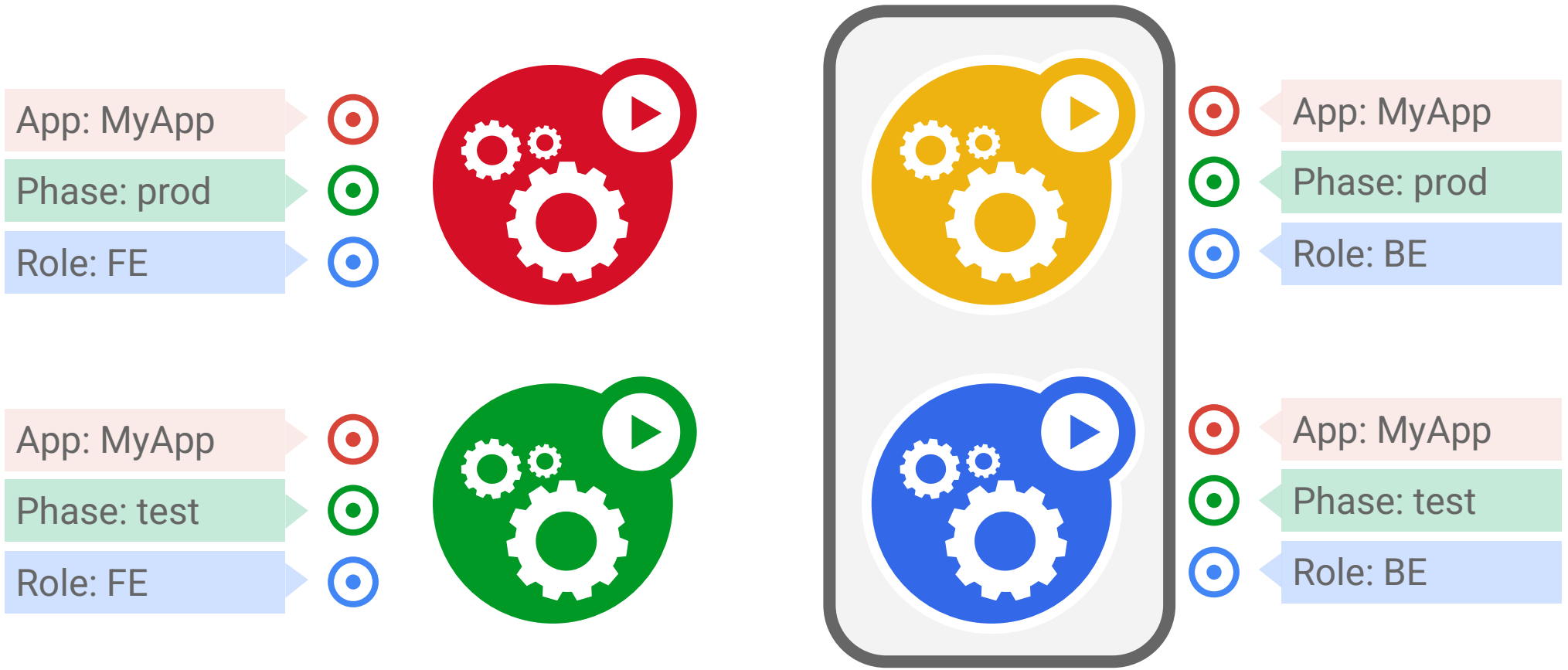
App = MyApp

Selectors



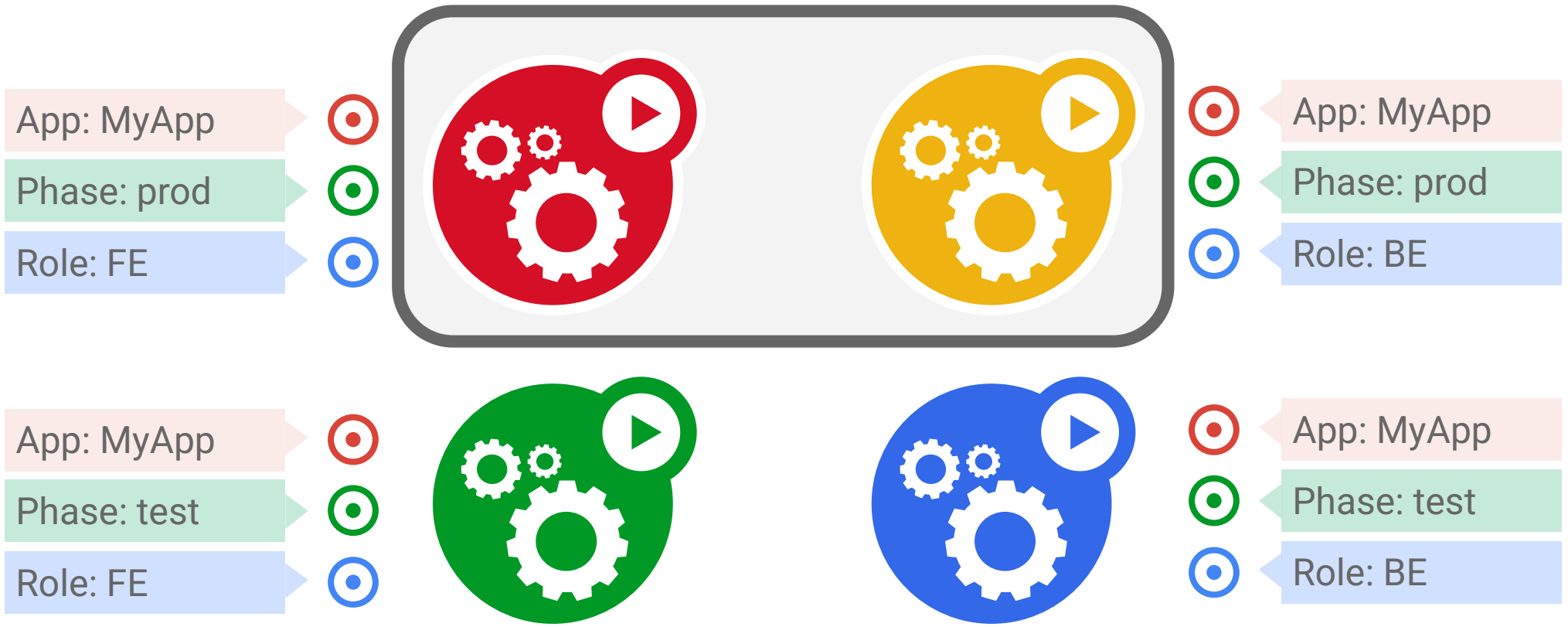
App = MyApp, Role = FE

Selectors



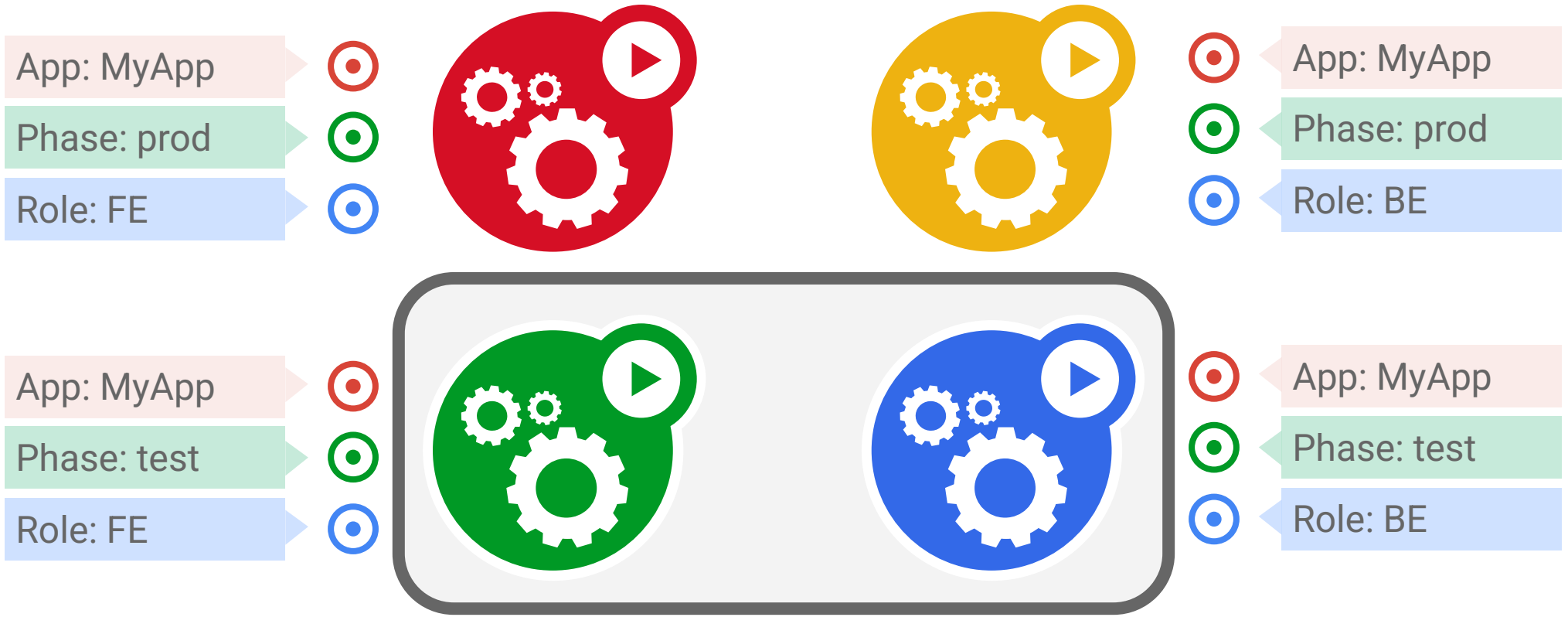
App = MyApp, Role = BE

Selectors



App = MyApp, Phase = prod

Selectors



App = MyApp, Phase = test

http://k8s-school.fr

Labels and selectors: exercice

1. Create 4 Deployments: `alpaca-prod, alpaca-test, bandicoot-prod, bandicoot-staging`

```
kubectl apply -f 6-0-alpaca-prod.yaml
```

2. Using `kubectl`:
   - Display pods and their labels (`--show-labels` option)
   - Add a label to a pod, then remove it (`kubectl label`)
   - Display a given label in a column (`-L` option)
   - Display pods which have labels (`-l` option):
     - `"app=bandicoot,ver=2"`
     - `“app in (bandicoot,alpaca)”`

Services

Services

A group of **pods that work together**

- grouped by a selector

Defines access policy

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

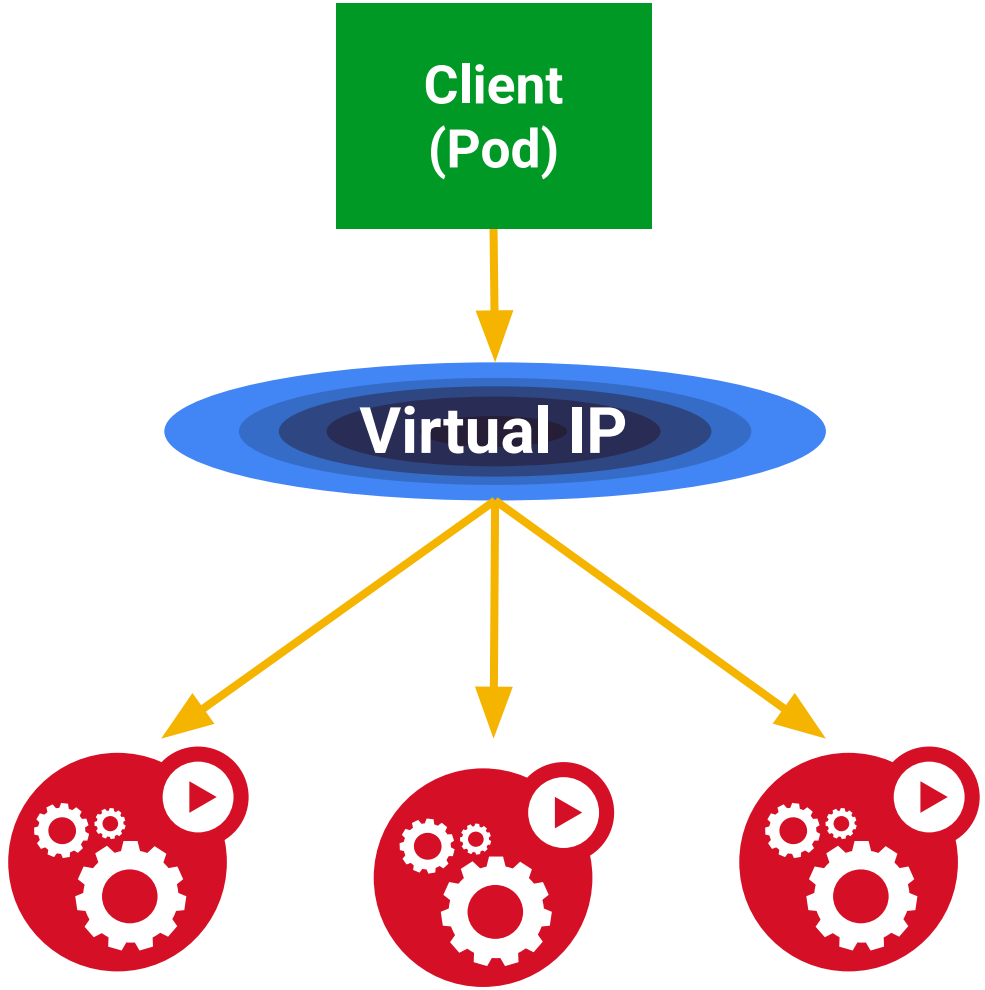
Can have a stable **virtual IP** and port(s)

- also a **DNS name**

VIP is managed by *kube-proxy*

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

Hides complexity



http://k8s-school.fr

Services: exercice

1. Create a Deployment alpaca-prod

`kubectl apply -f 7-1-alpaca-prod-readiness.yaml`

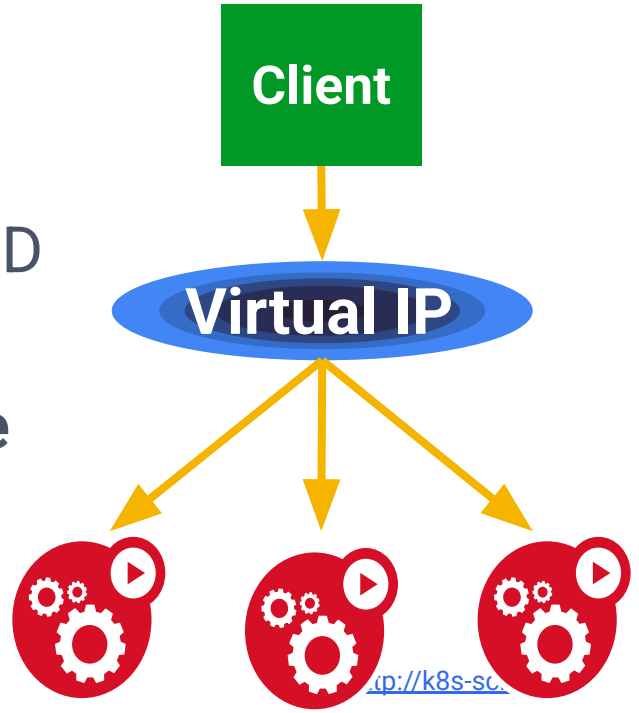
2. Create a Service for the alpaca-prod pods *(kubectl create service --help)*
3. Create a Deployment bandicoot-prod *(optional)*
4. Create a Service for the bandicoot-prod pods *(optional)*

TIP create a file without UI:

`cat > file.yaml` + Paste + Ctrl^D

TIP check pods exposed by service

`kubectl get endpoints`



Example for service file

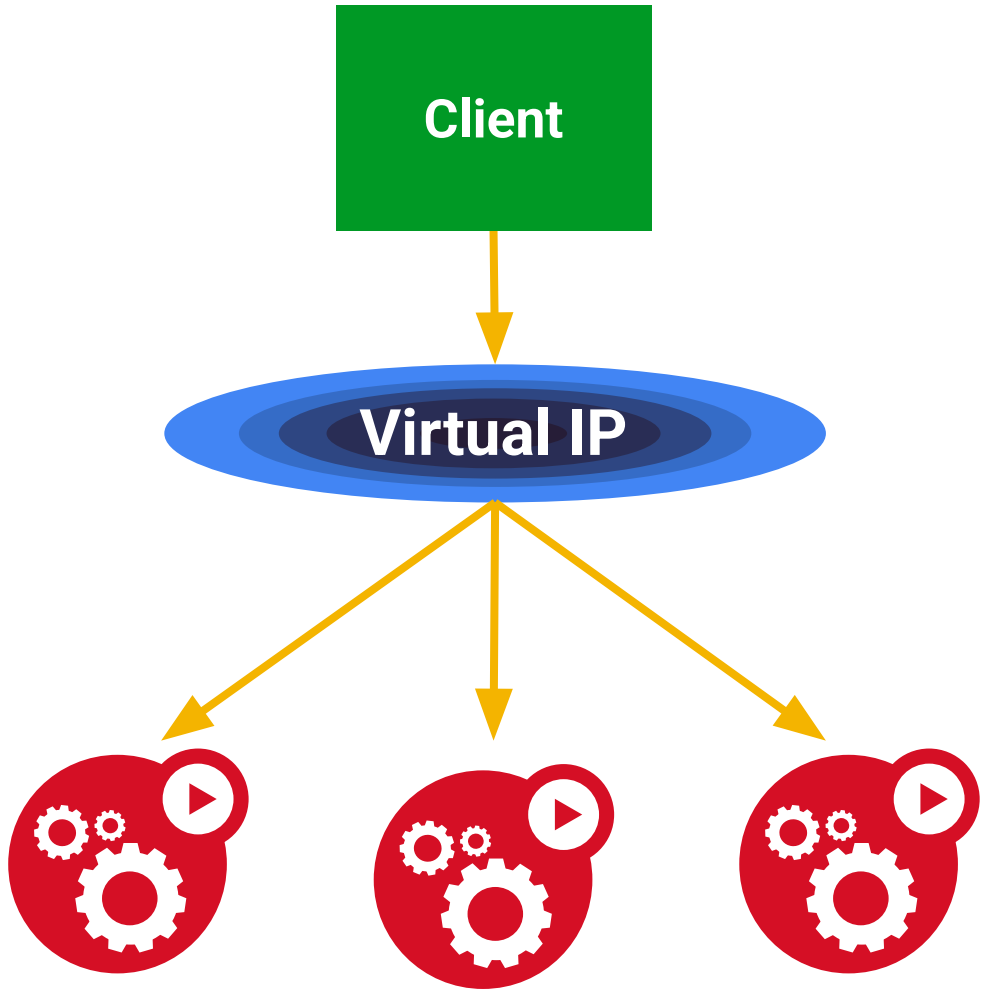
```
apiVersion: v1
kind: Service
metadata:
  name: bandicoot-prod
spec:
  selector:
    ver: "2"
    app: "bandicoot"
    env: "prod"
  ports:
  - protocol: TCP
    port: 8080
```

Services: exercice 2

1. Run a pod and open a shell inside it (for example use "ubuntu" container image) *kshell* may help
2. Install DNS client (dig or nslookup)
3. Launch a DNS query inside it to retrieve IP address of "alpaca-prod" service

```
$ nslookup  alpaca-prod
$ cat /etc/resolv.conf
```

4. Connect to an alpaca-prod pod using "`curl http://XXXXX:NN`"



Services: readiness probe

```
apiVersion: v1
kind: Deployment
metadata:
 name: alpaca-prod
spec:
 replicas: 3
 …
 template:
  …
    spec:
     containers:
       - image: gcr.io/kuar-demo/kuard-amd64:1
         name: kuard
         …
         readinessProbe:
           httpGet:
             path: /ready
             port: 8080
           initialDelaySeconds: 30
           timeoutSeconds: 1
           periodSeconds: 10
           failureThreshold: 3
```

1. Watch endpoints
   (use `kubectl --watch`)
2. Delete a pod or set it not ready using kuard UI
3. Check what happens to endpoints

One readinessProbe per container
3 methods
httpGet, tcpSocket, and exec

External services

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

Need to receive traffic from “the outside world”

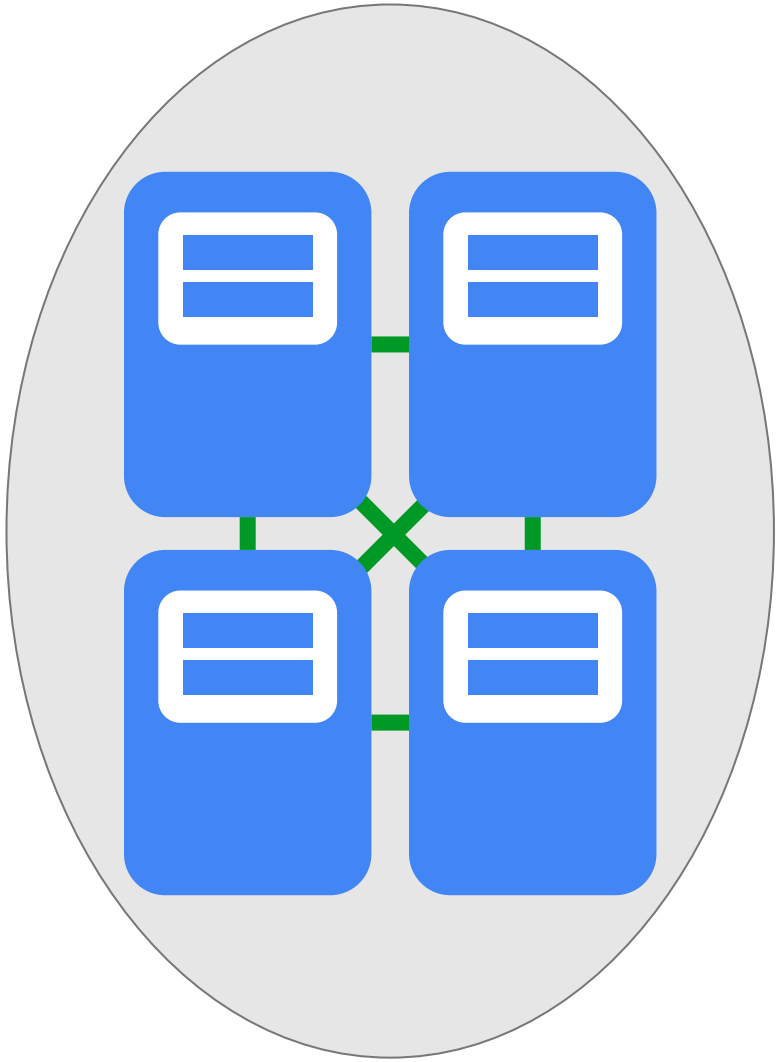
Service “type”

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

DiY load-balancer solutions

- socat (for NodePort remapping)
- haproxy
- nginx

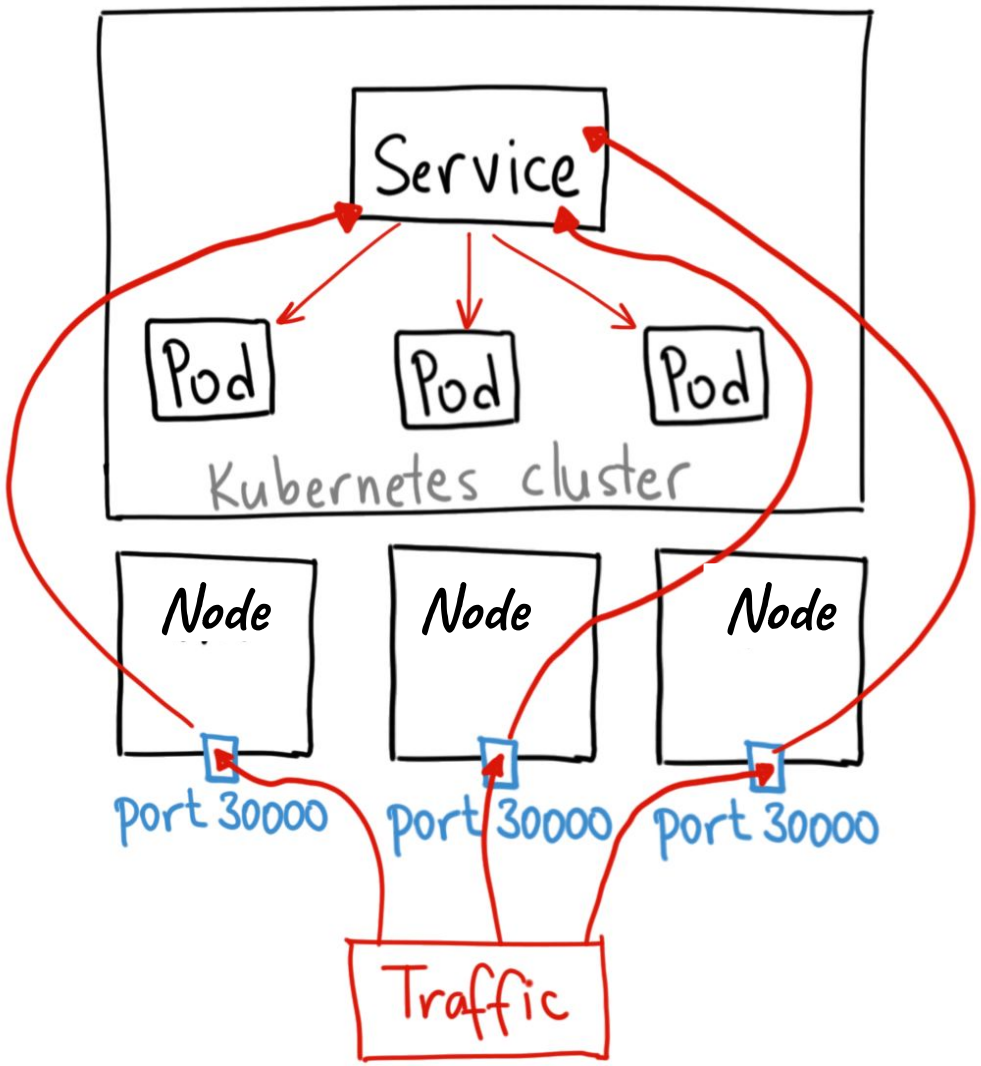
Ingress (L7 LB)



http://k8s-school.fr

The NodePort Service

Simplest way to access a Pod from outside the cluster



NodePort: exercice

Access to NodePort service from outside k8s

1. Create and scale the deployment

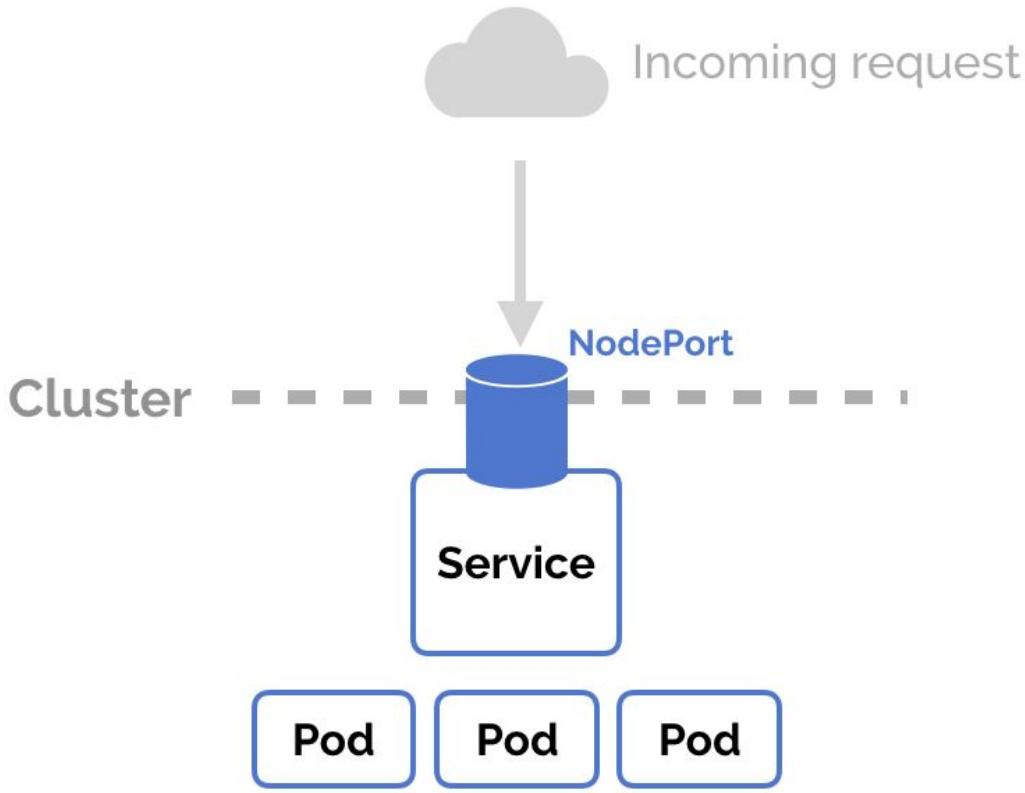
```
kubectl create deployment microbot
--image=dontrebootme/microbot:v1
kubectl scale deployment microbot
--replicas=3
```

2. Create the NodePort service

```
kubectl expose deployment microbot
--type=NodePort --port=80
--name=microbot-service
```

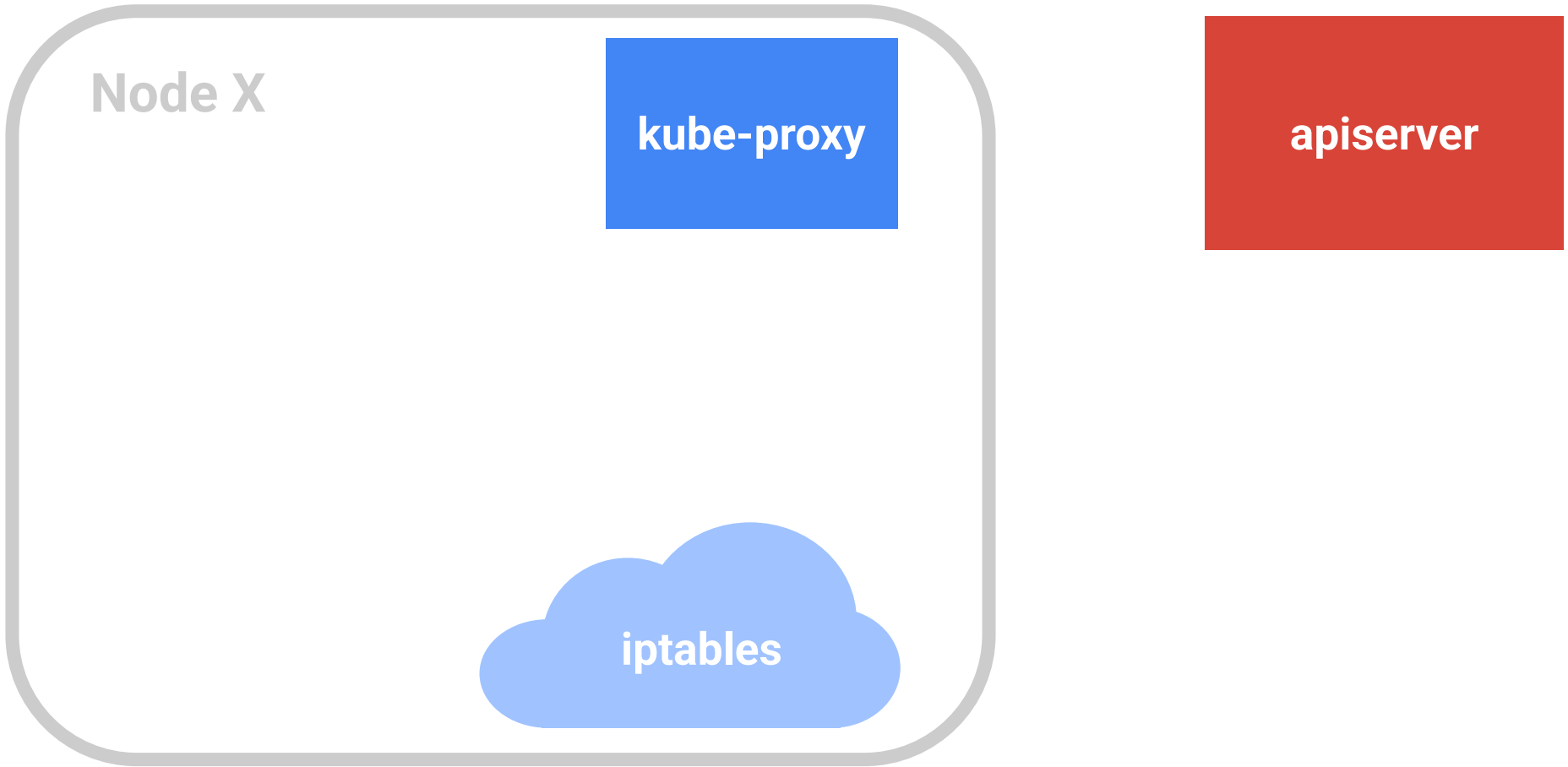
3. From outside the cluster

```
curl http://XXXXX
```



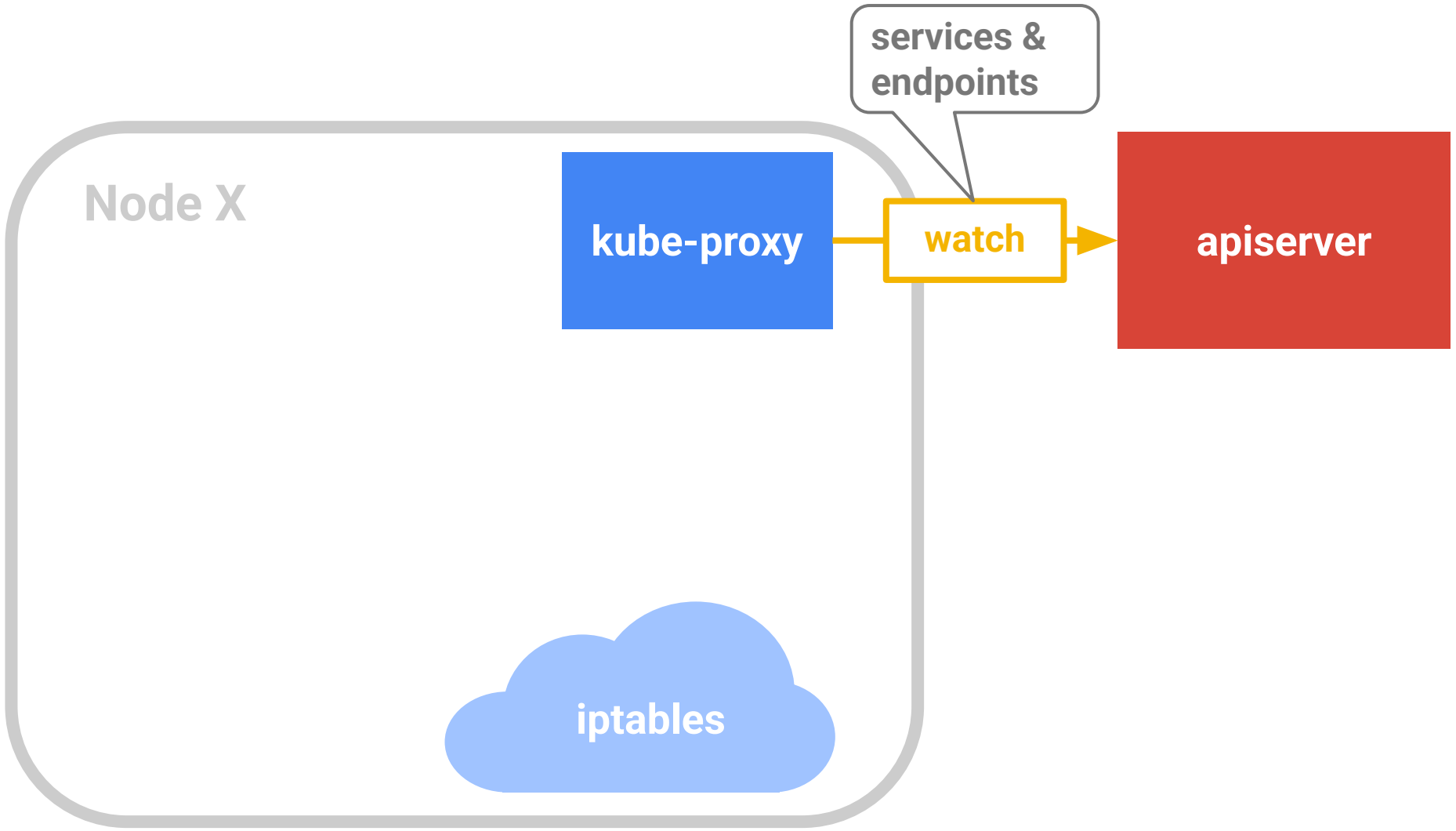
Kube Proxy iptables

iptables kube-proxy

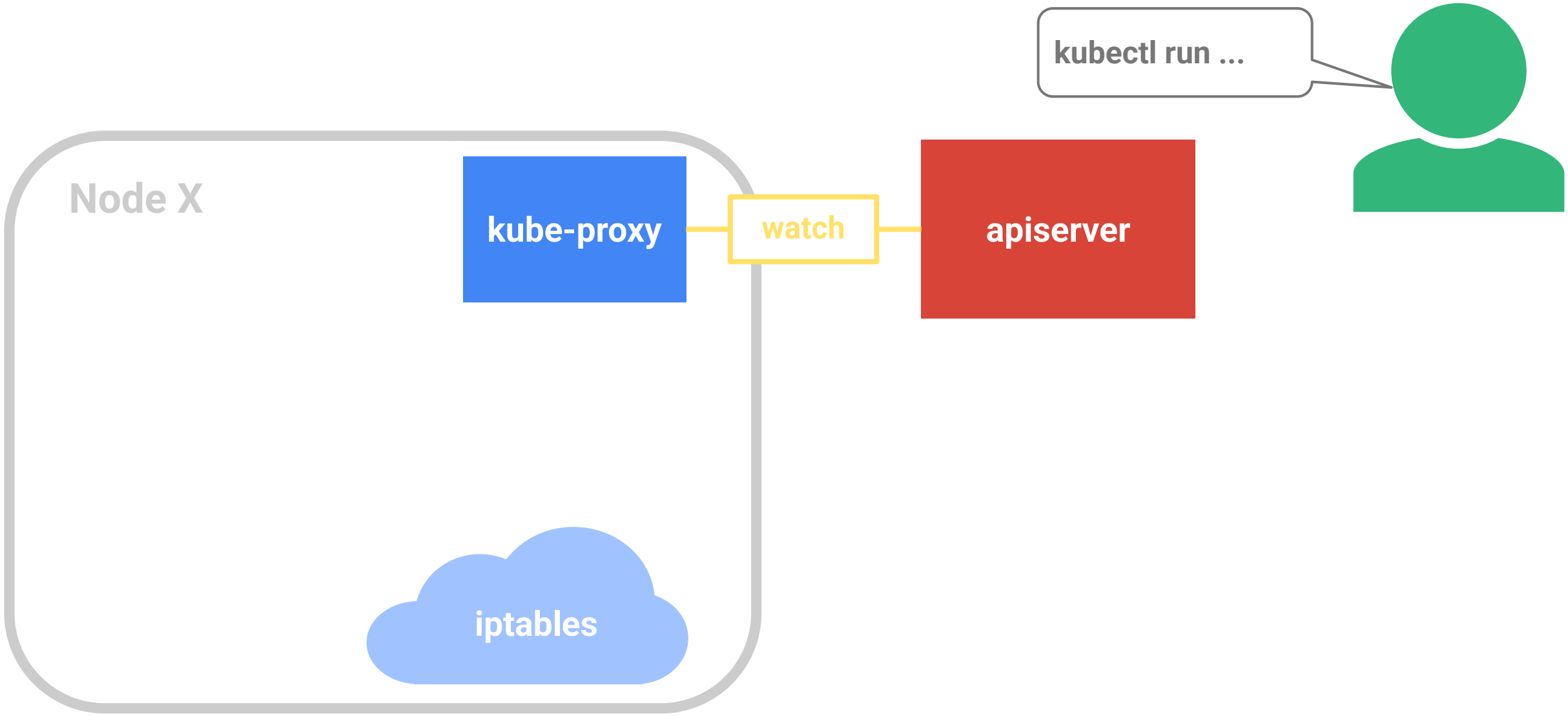


apiserver

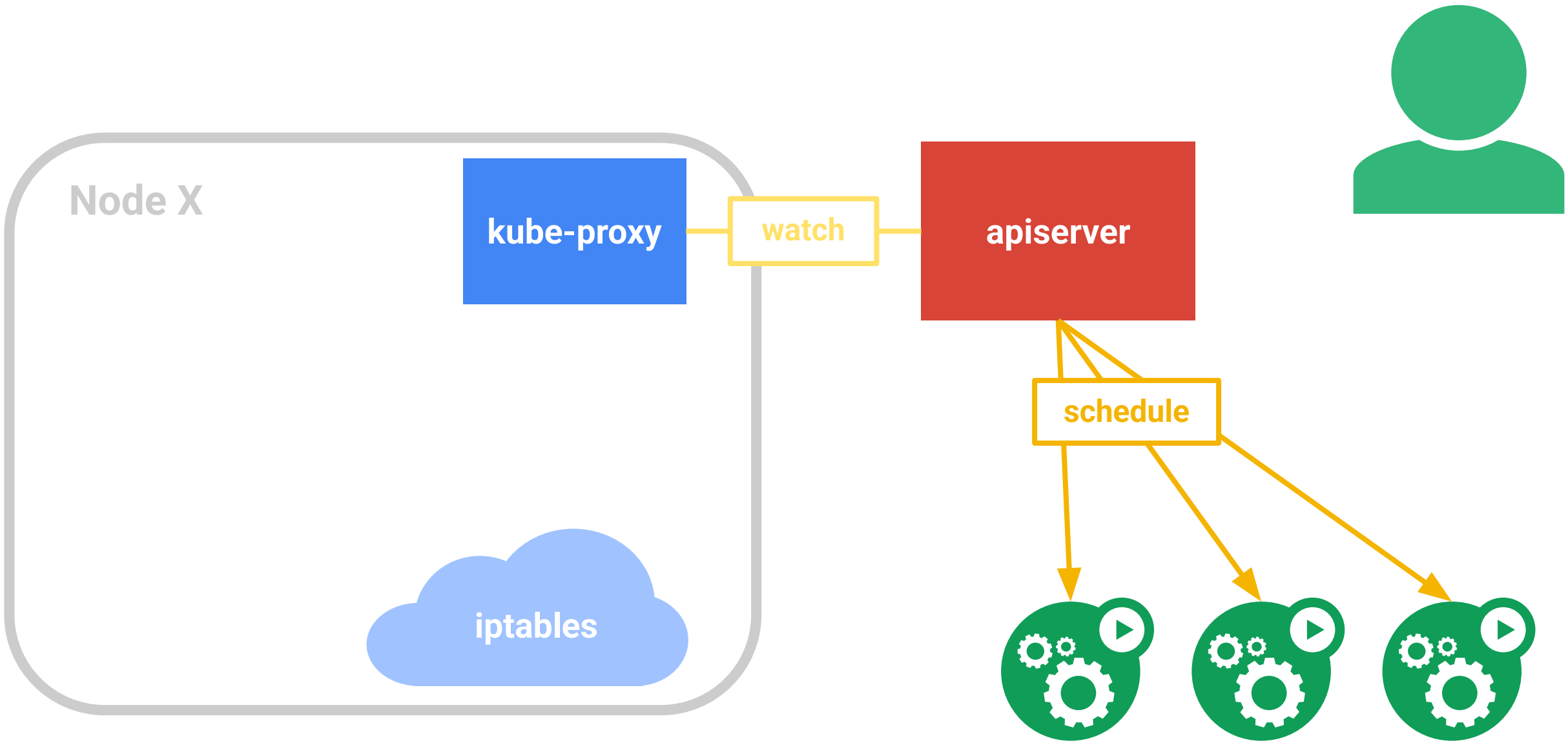
iptables kube-proxy



iptables kube-proxy

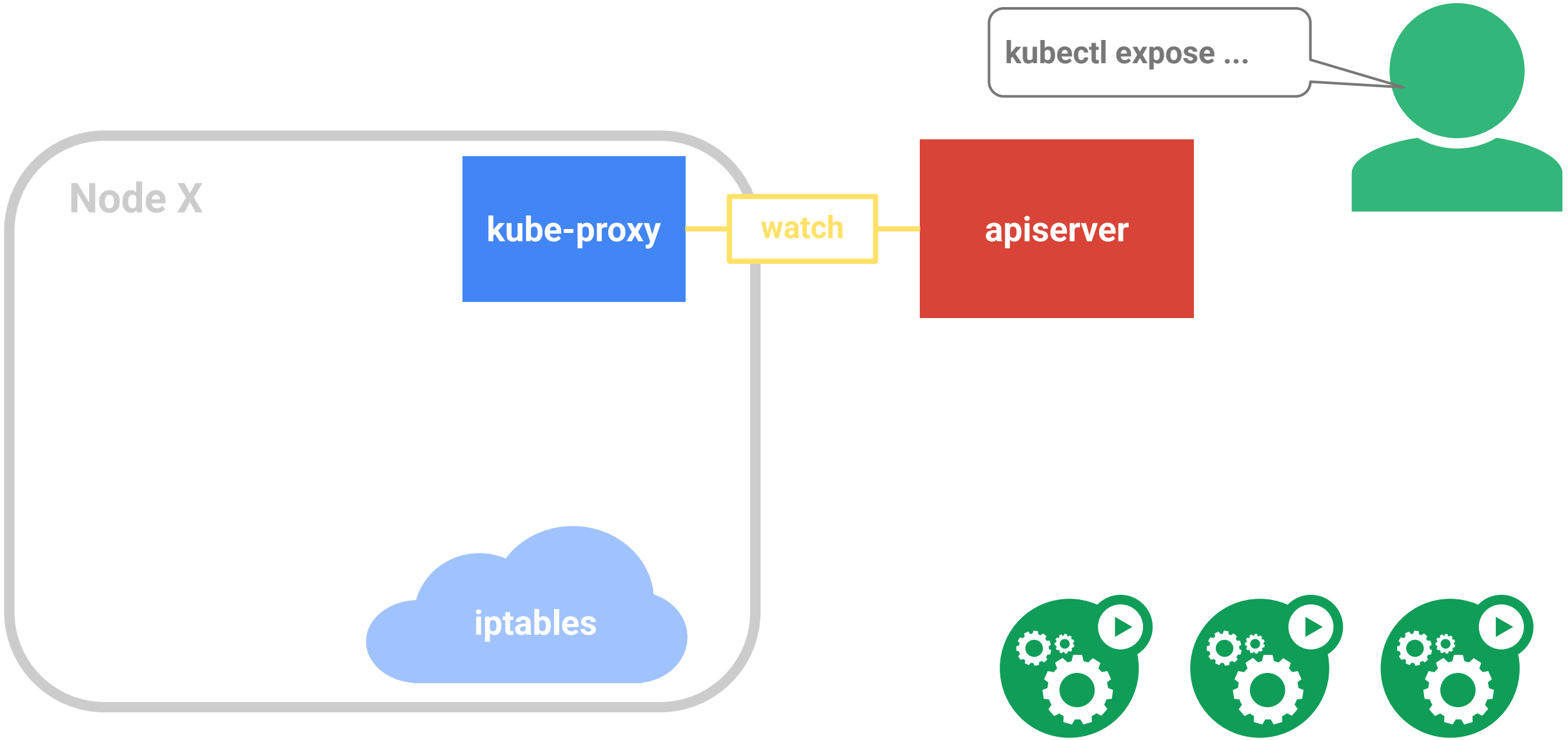


iptables kube-proxy

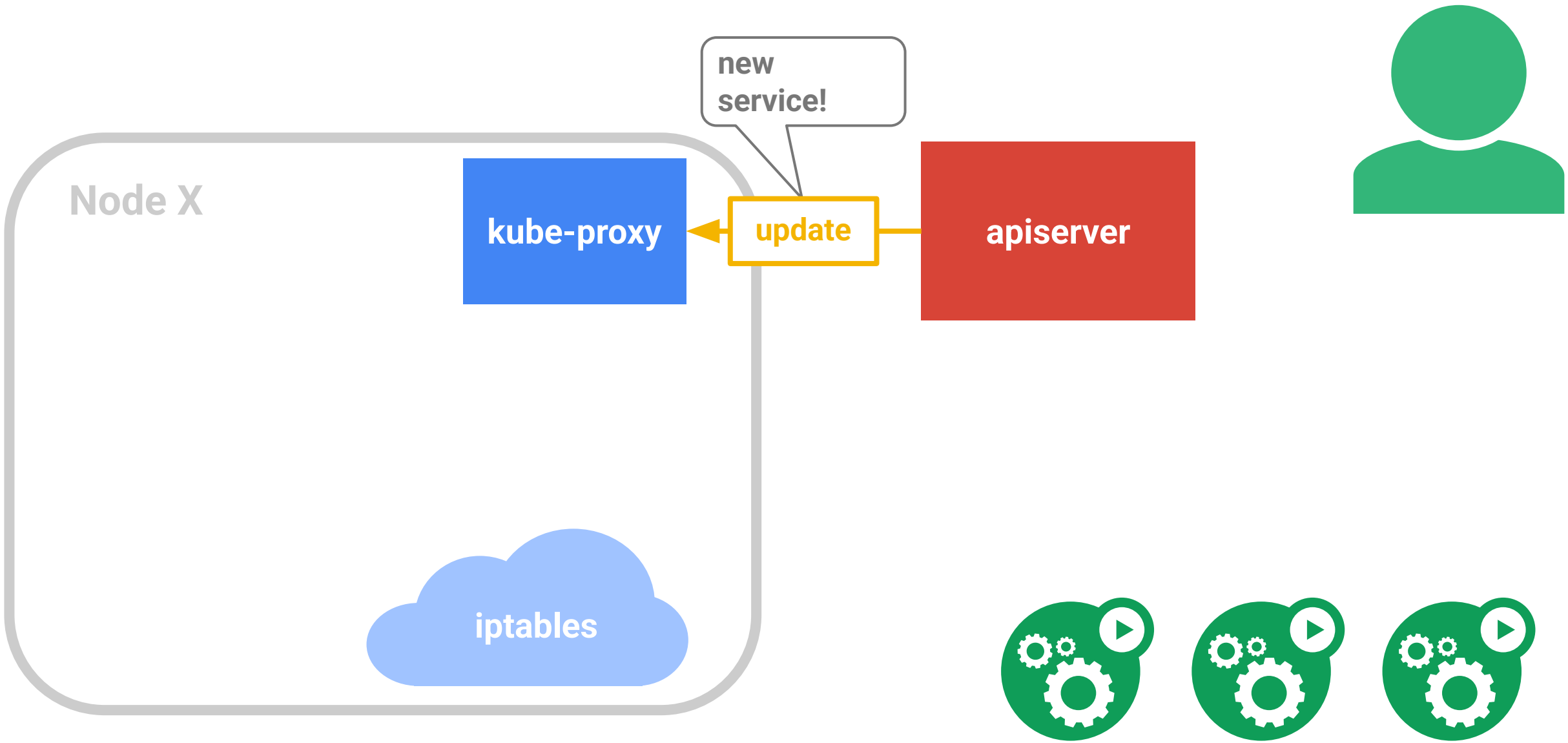


http://k8s-school.fr

iptables kube-proxy

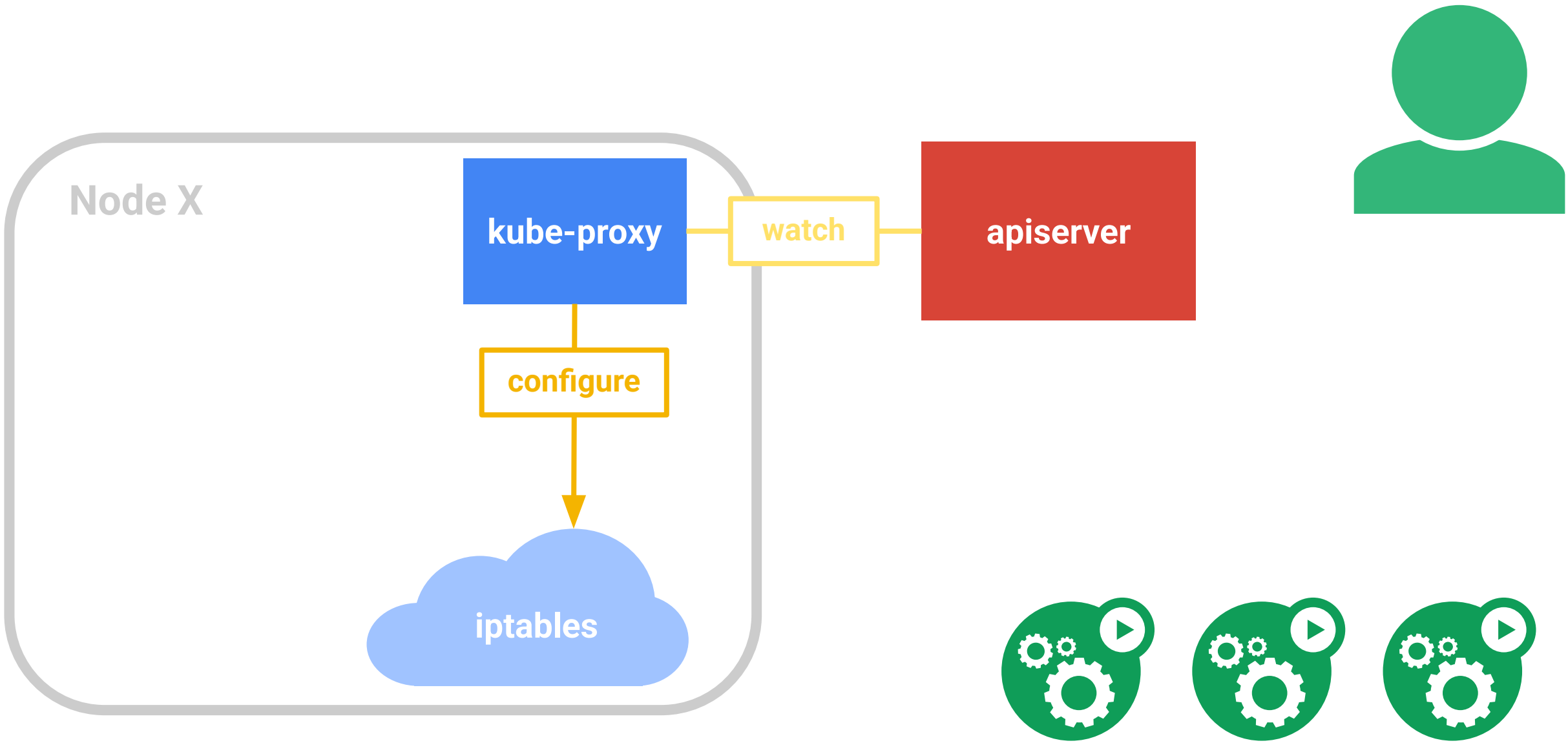


iptables kube-proxy



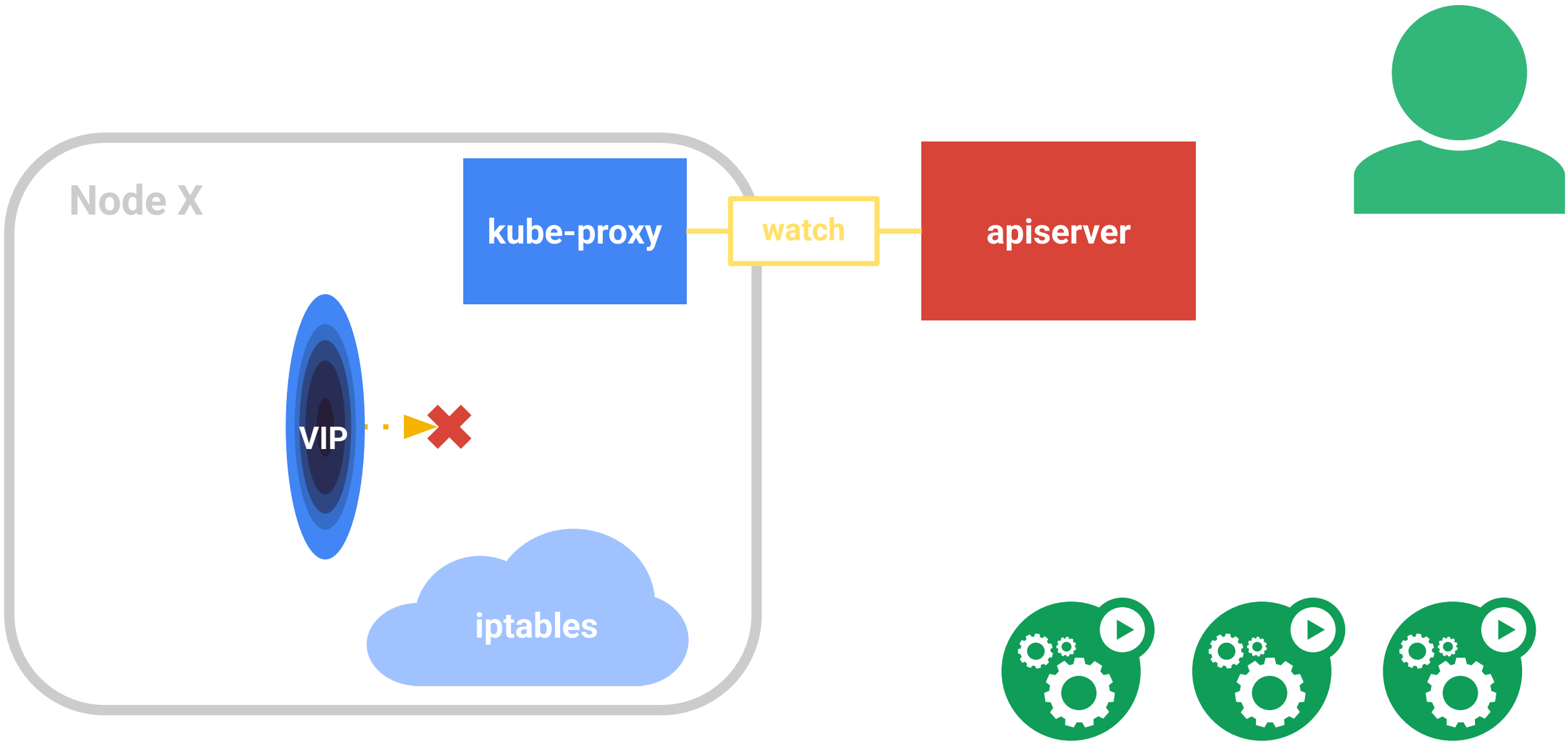
http://k8s-school.fr

iptables kube-proxy



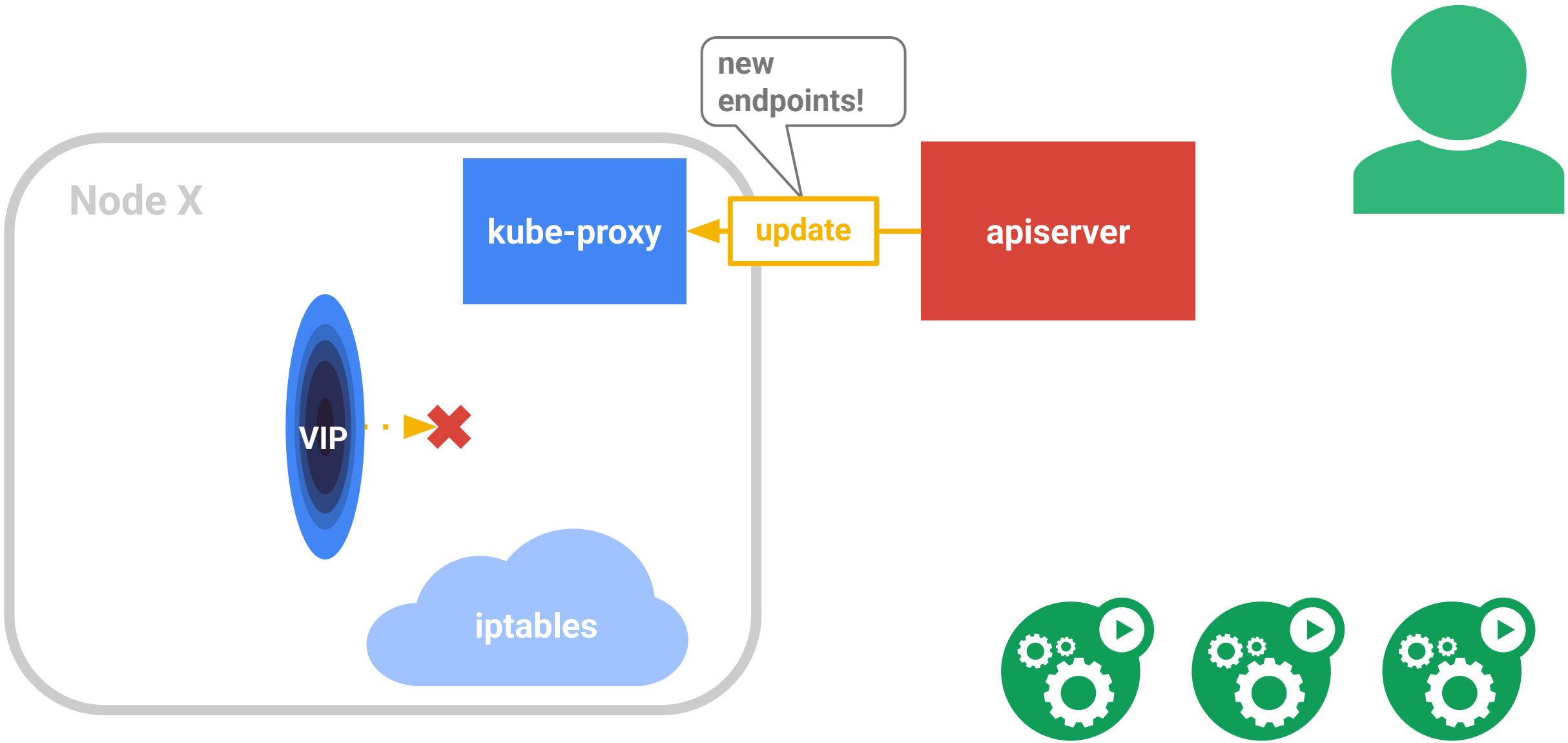
http://k8s-school.fr

iptables kube-proxy

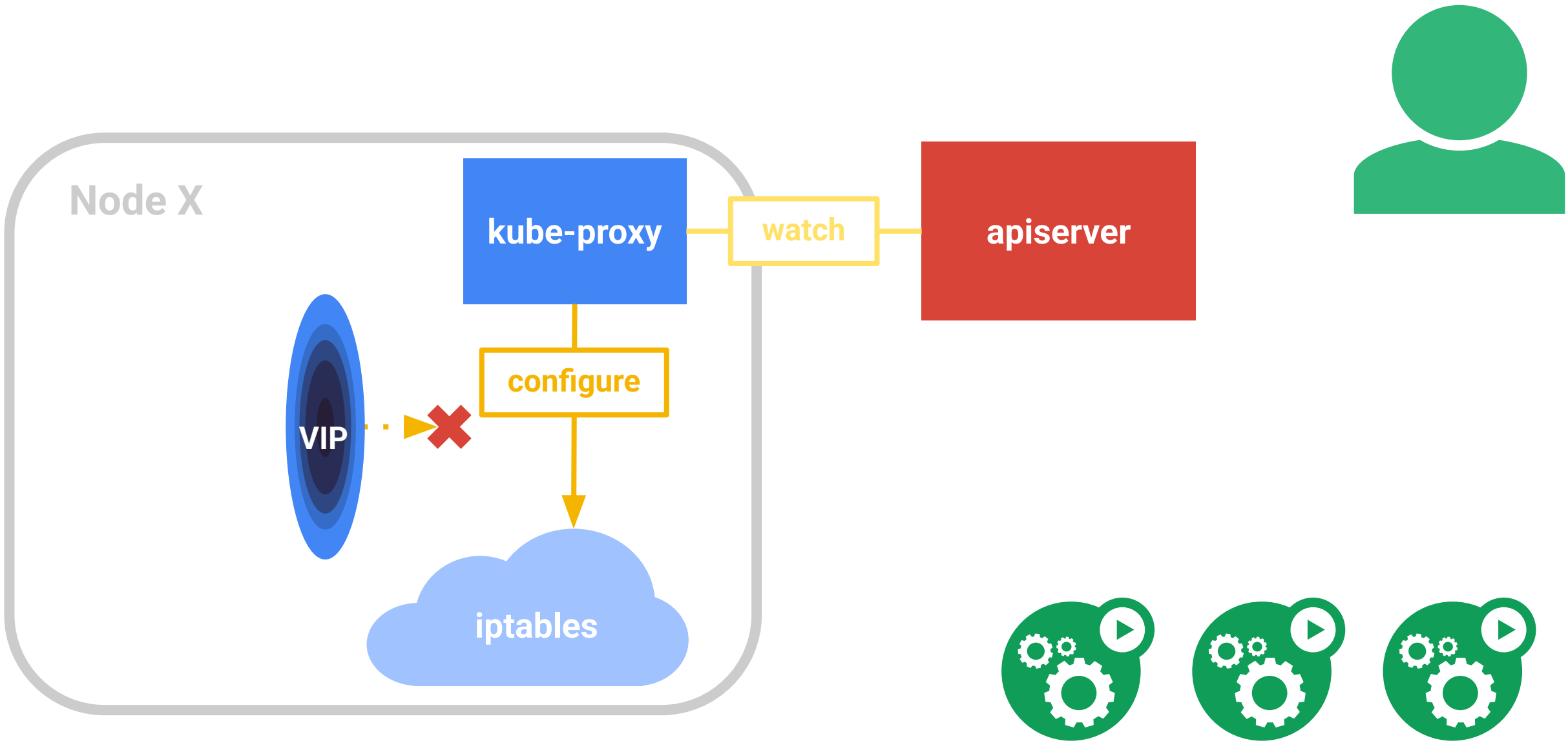


http://k8s-school.fr

iptables kube-proxy

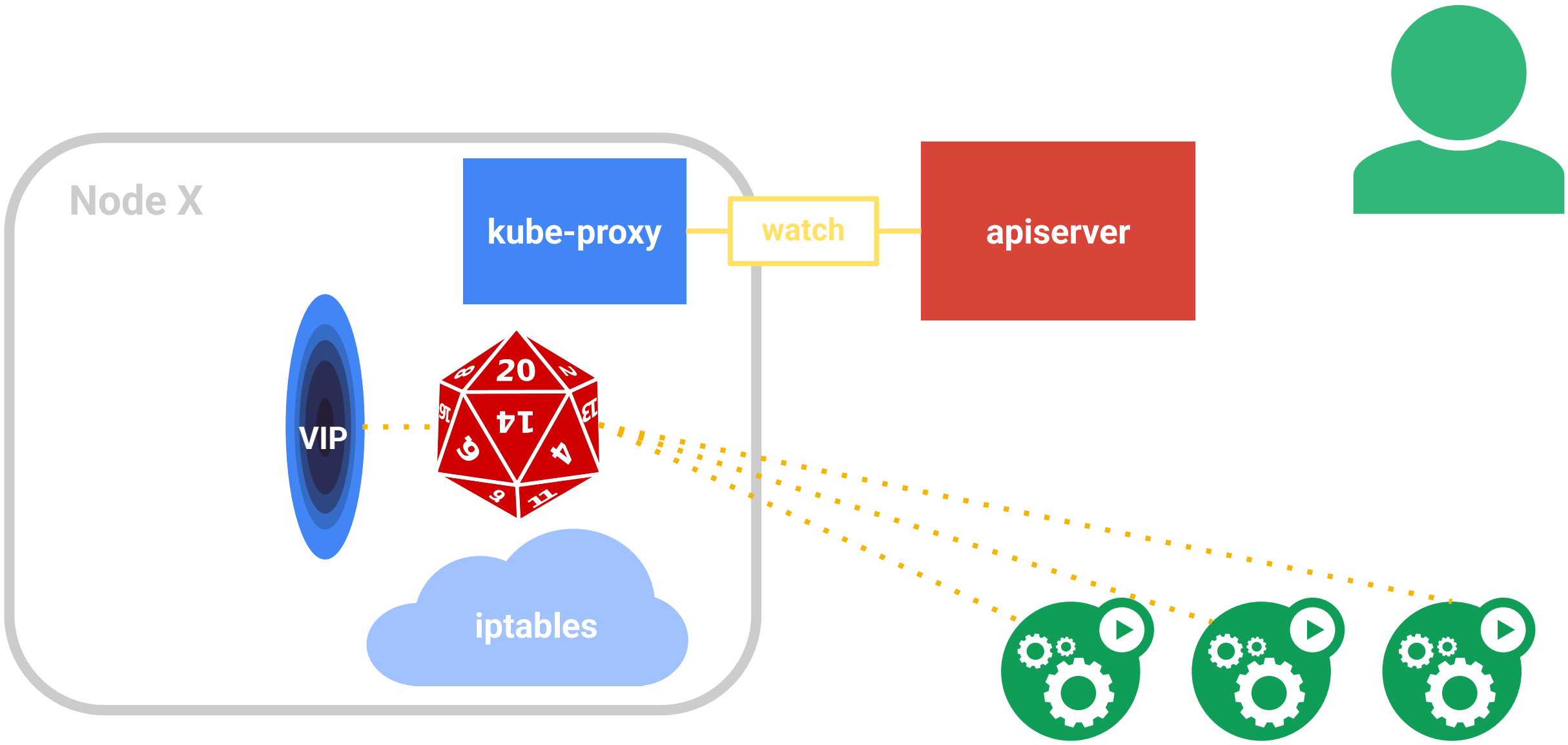


iptables kube-proxy



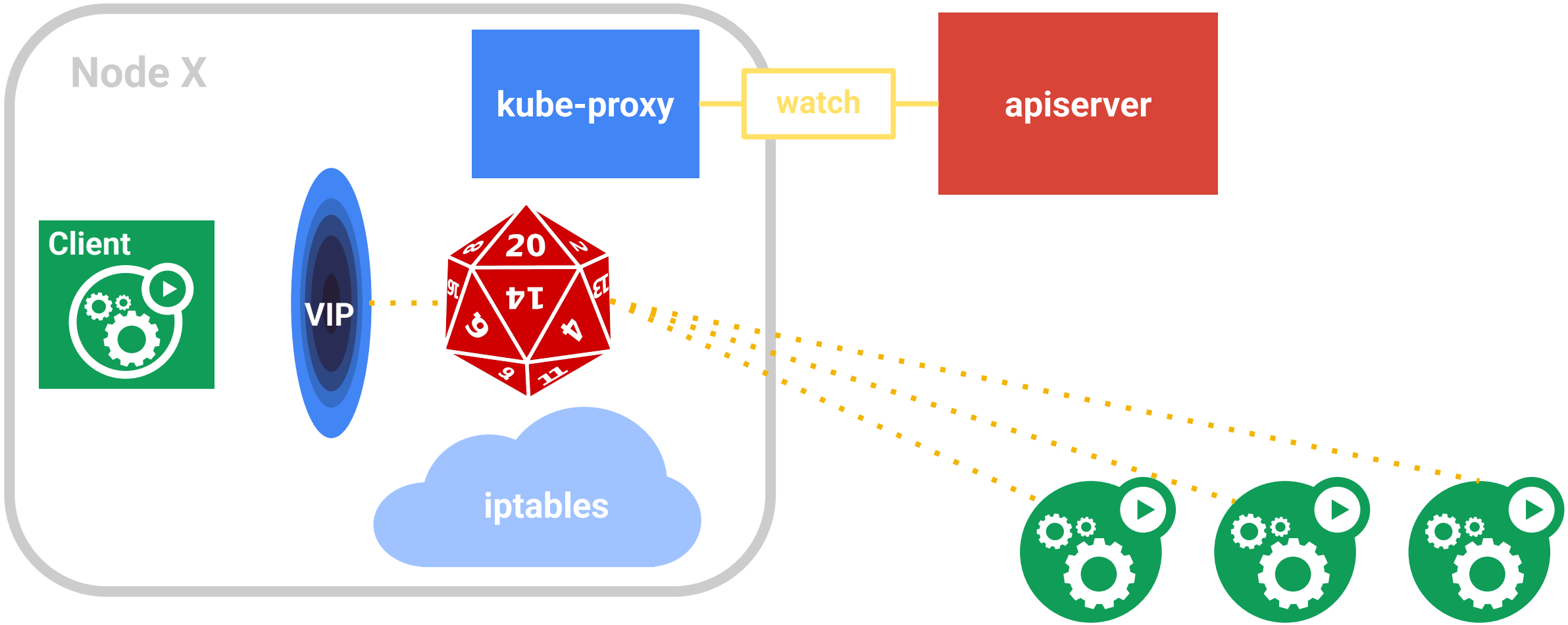
http://k8s-school.fr

iptables kube-proxy



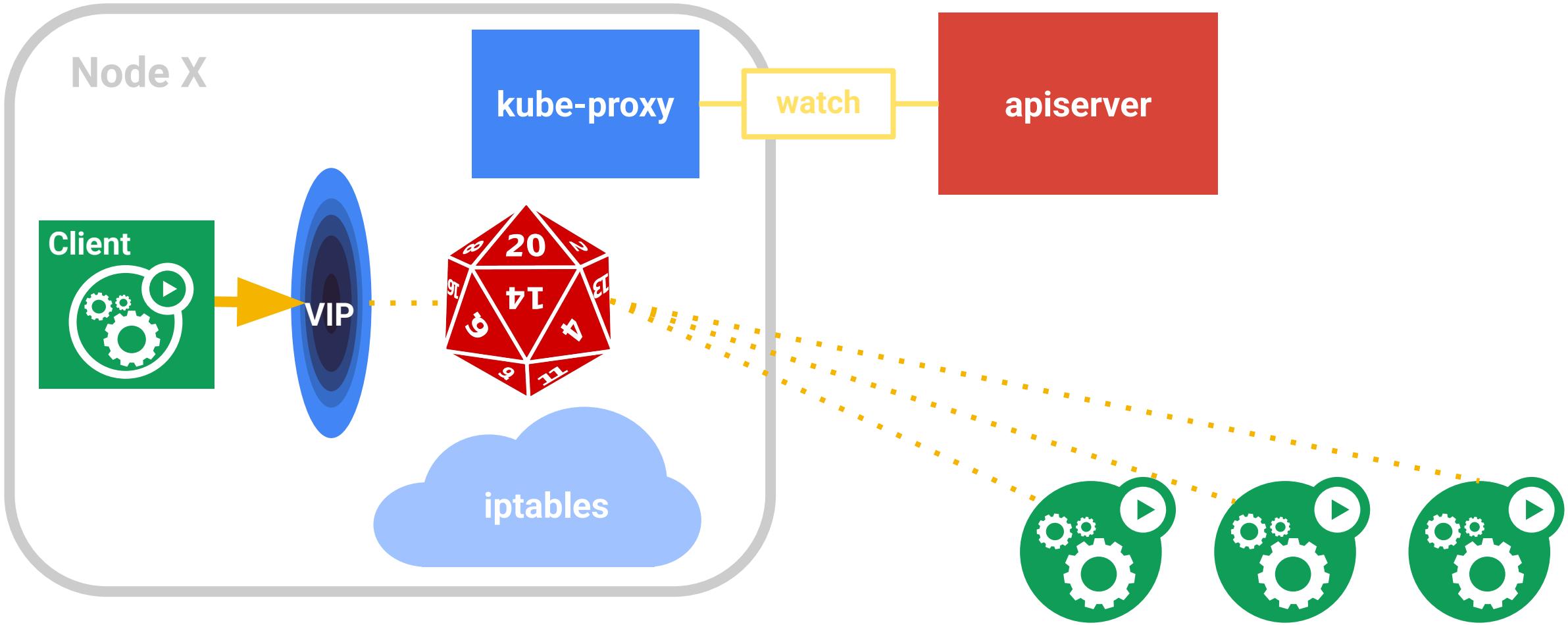
http://k8s-school.fr

iptables kube-proxy



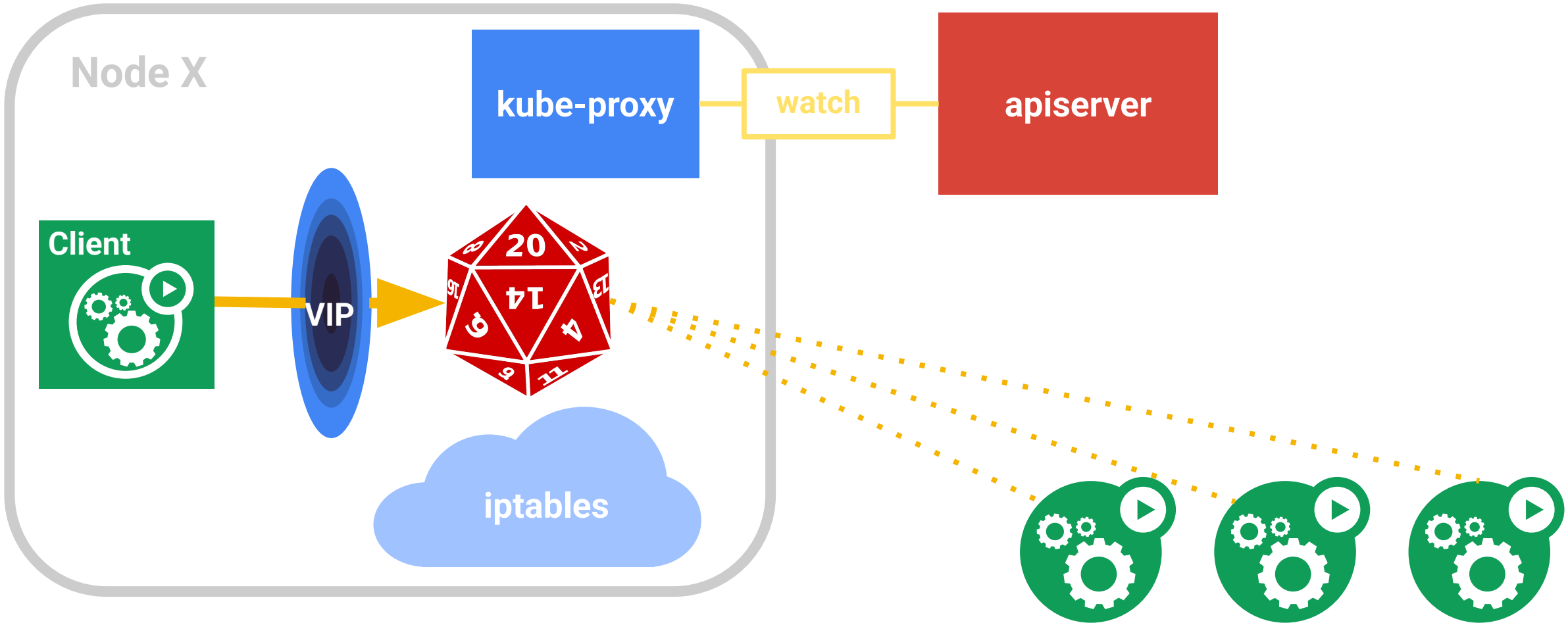
http://k8s-school.fr

iptables kube-proxy



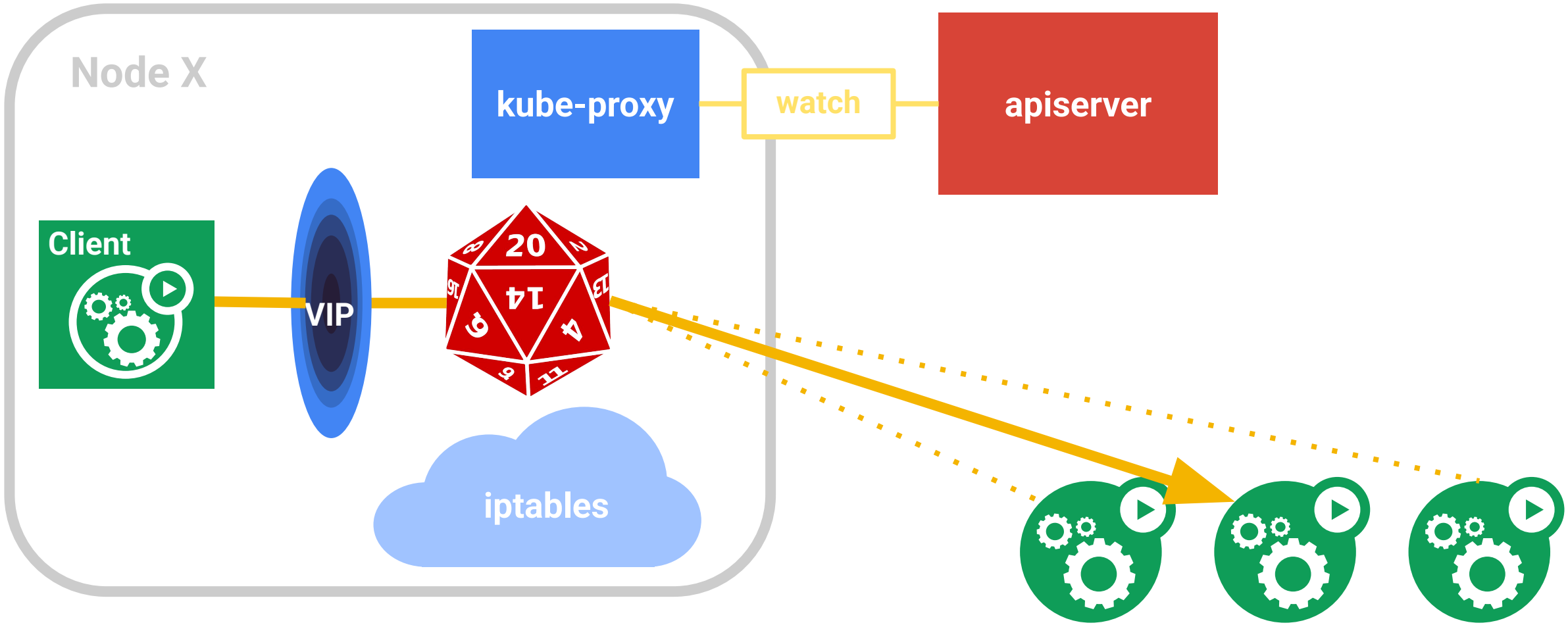
http://k8s-school.fr

iptables kube-proxy



http://k8s-school.fr

iptables kube-proxy



http://k8s-school.fr

Running Pods

ReplicaSets

A simple controller loop

Runs out-of-process wrt API server

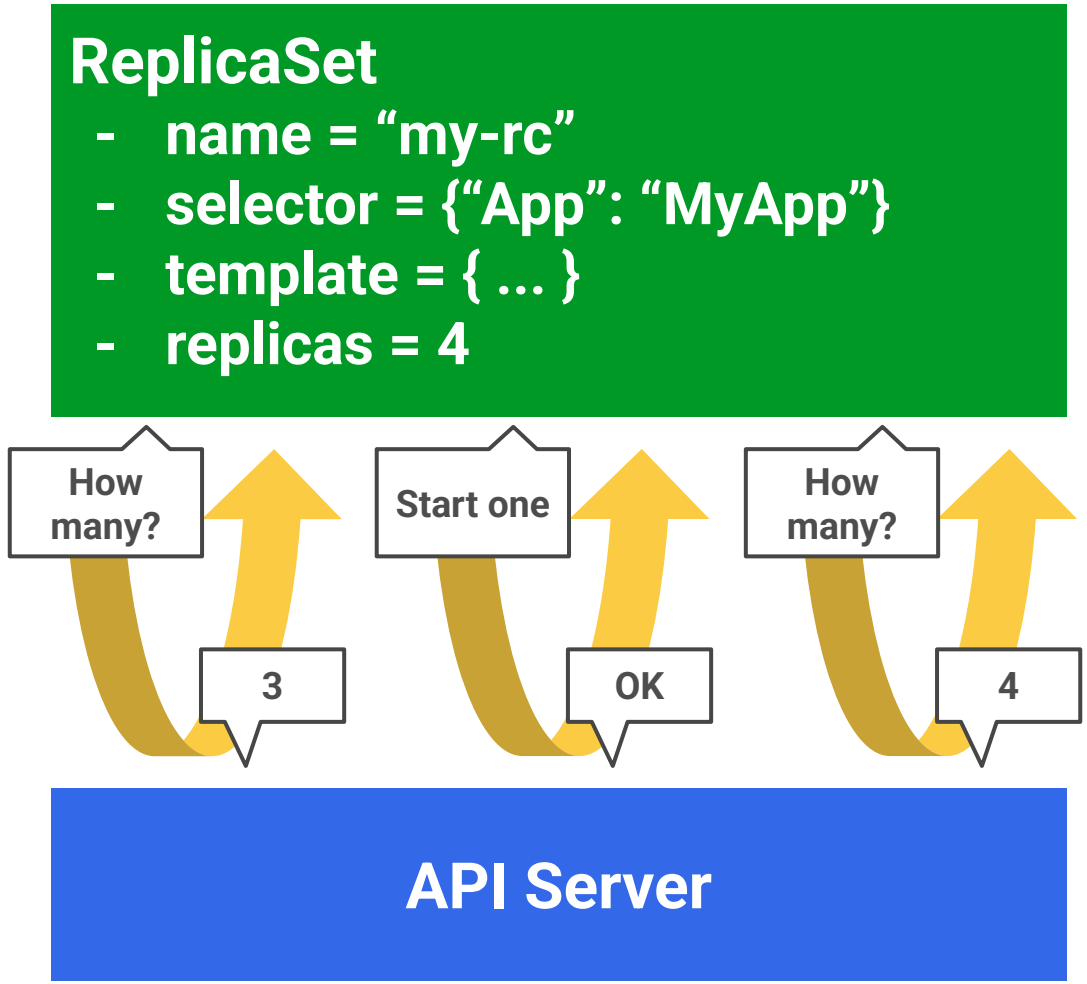
One job: ensure N copies of a pod

- 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



ReplicaSets

Redundancy
Multiple running instances mean failure can be tolerated.

Scale
Multiple running instances mean that more requests can be handled.

Sharding
Different replicas can handle different parts of a computation in parallel.

Control loops

Drive **current state -> desired state**

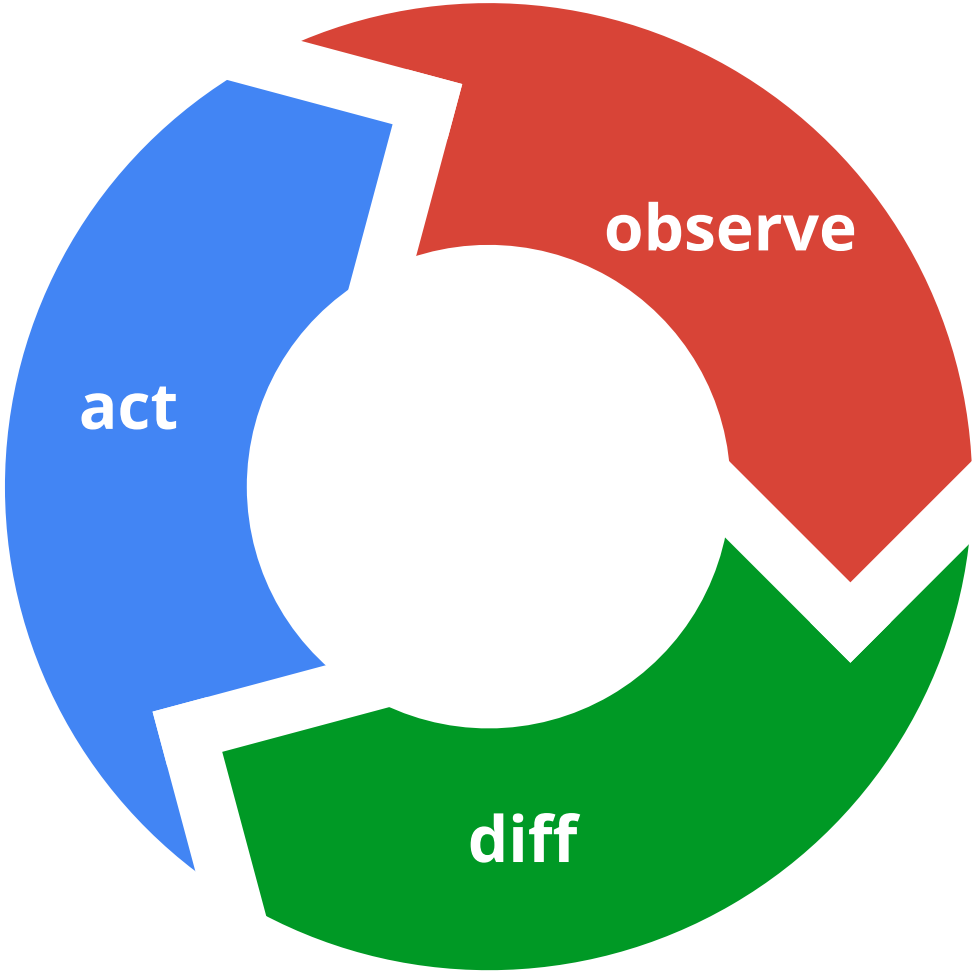
Act independently

APIs - **no shortcuts** or back doors

Observed state is truth

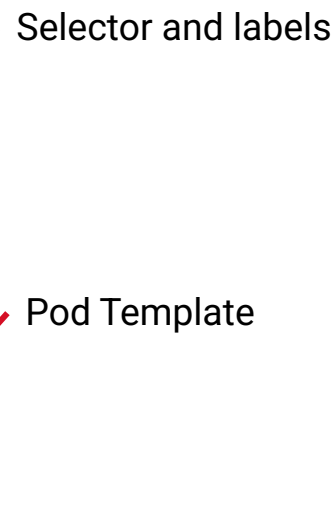
Recurring pattern in the system

Example: ReplicaSetController



ReplicaSets

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
 labels:
   app: kuard
 name: kuard
spec:
 replicas: 10
 selector:
   matchLabels:
     app: kuard
     version: "2"
 template:
   metadata:
     labels:
       app: kuard
       version: "2"
   spec:
     containers:
       - name: kuard
         image: "gcr.io/kuar-demo/kuard-amd64:2"
```



ReplicaSets: exercice

Create a ReplicaSet

```
kubectl apply -f 9-1-kuard-rs.yaml
```

List a ReplicaSet

```
kubectl get replicaset <rs_name>
```

Retrieve the pods for the ReplicaSet using kubectl selector `(kubectl -l)`

Scale the replicaset to 5/6 pods:

- using `kubectl edit`
- using `kubectl scale`

Auto-Scaling

HorizontalPodAutoScalers

Goal: Automatically scale pods as needed

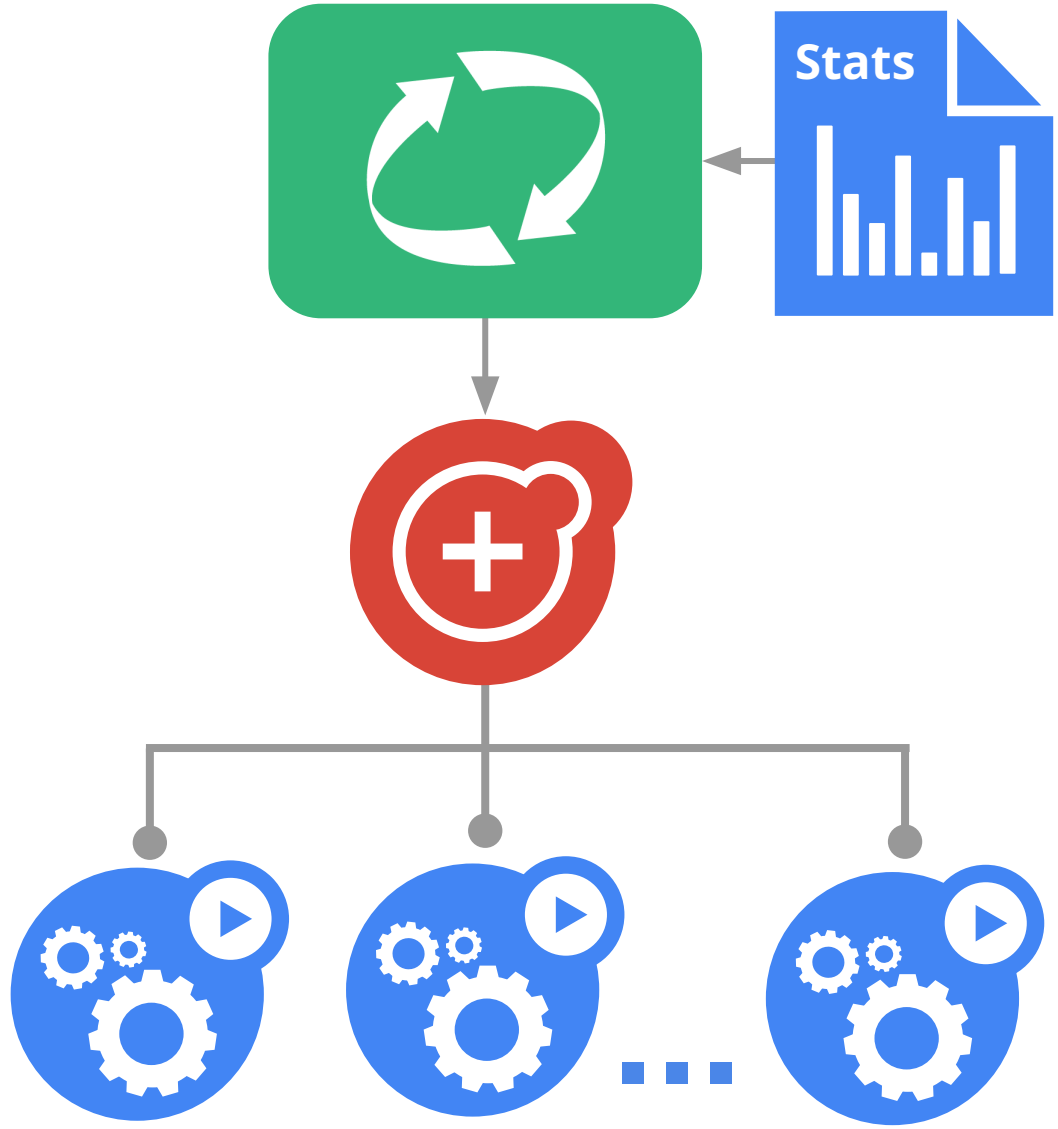
- based on CPU utilization (for now)
- custom metrics in Alpha

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 metrics-server



http://k8s-school.fr

ReplicaSets: HPA

NOTE: HPA requires kubernetes-sigs/metrics-server

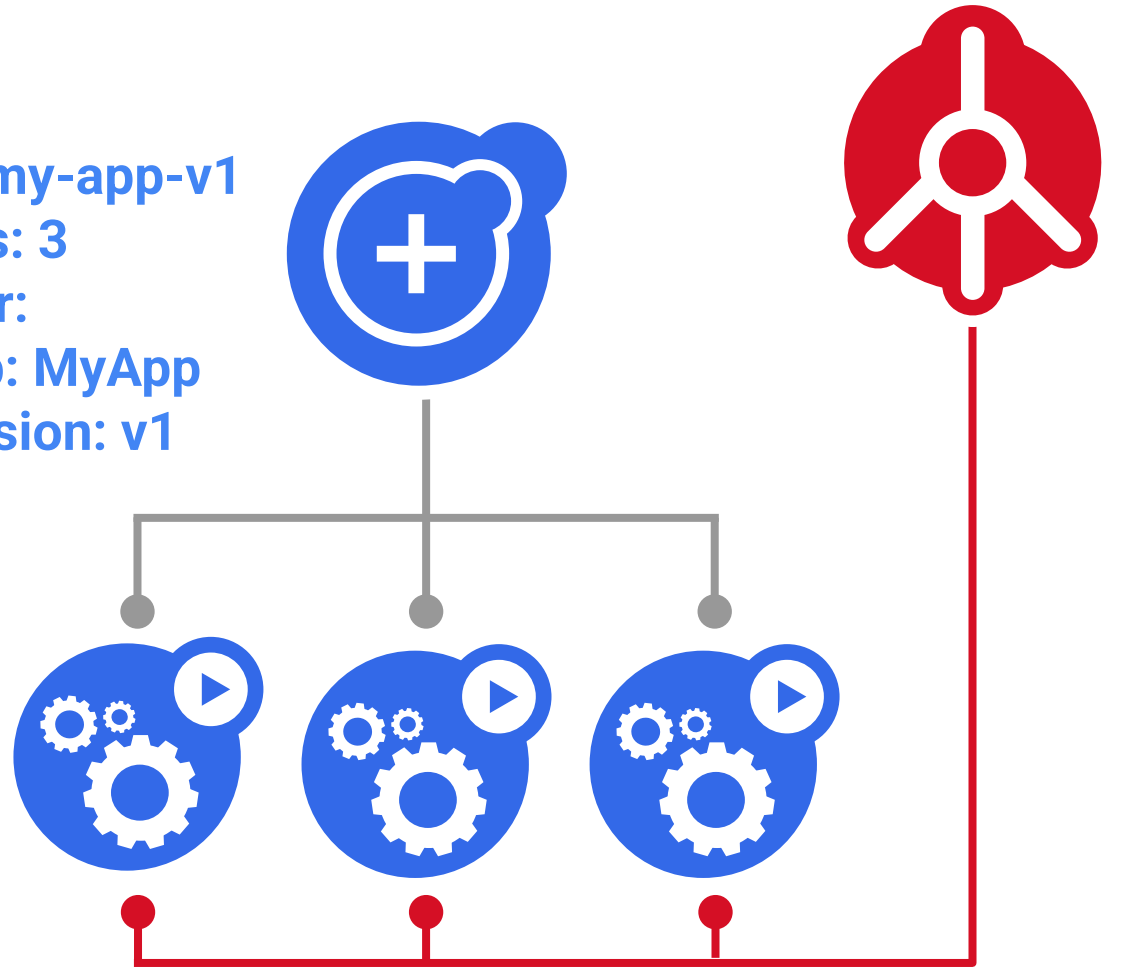
```
kubectl autoscale rs kuard --min=2 --max=5 --cpu-percent=80

kubectl get hpa
```

Rolling Updates

Rolling Update

Service
- app: MyApp

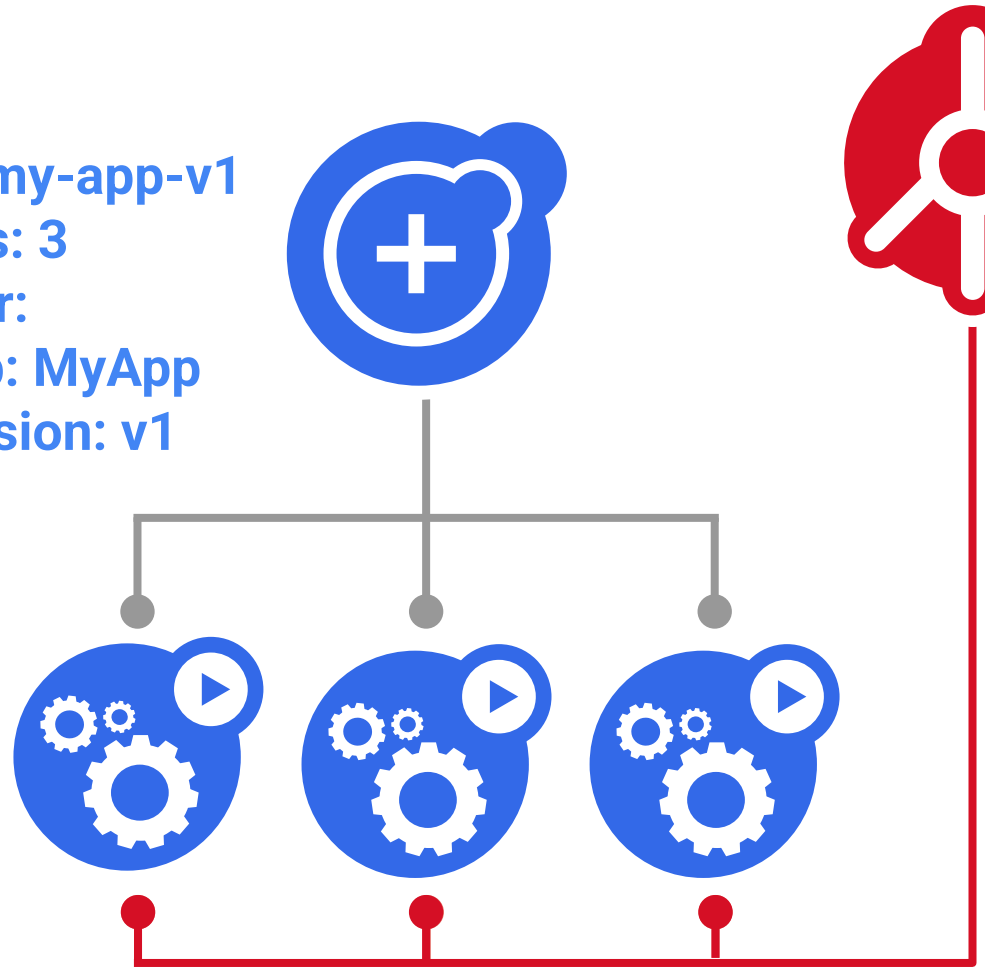
ReplicaSet
- name: my-app-v1
- replicas: 3
- selector:
  - app: MyApp
  - version: v1



http://k8s-school.fr

Rolling Update

Service
- app: MyApp



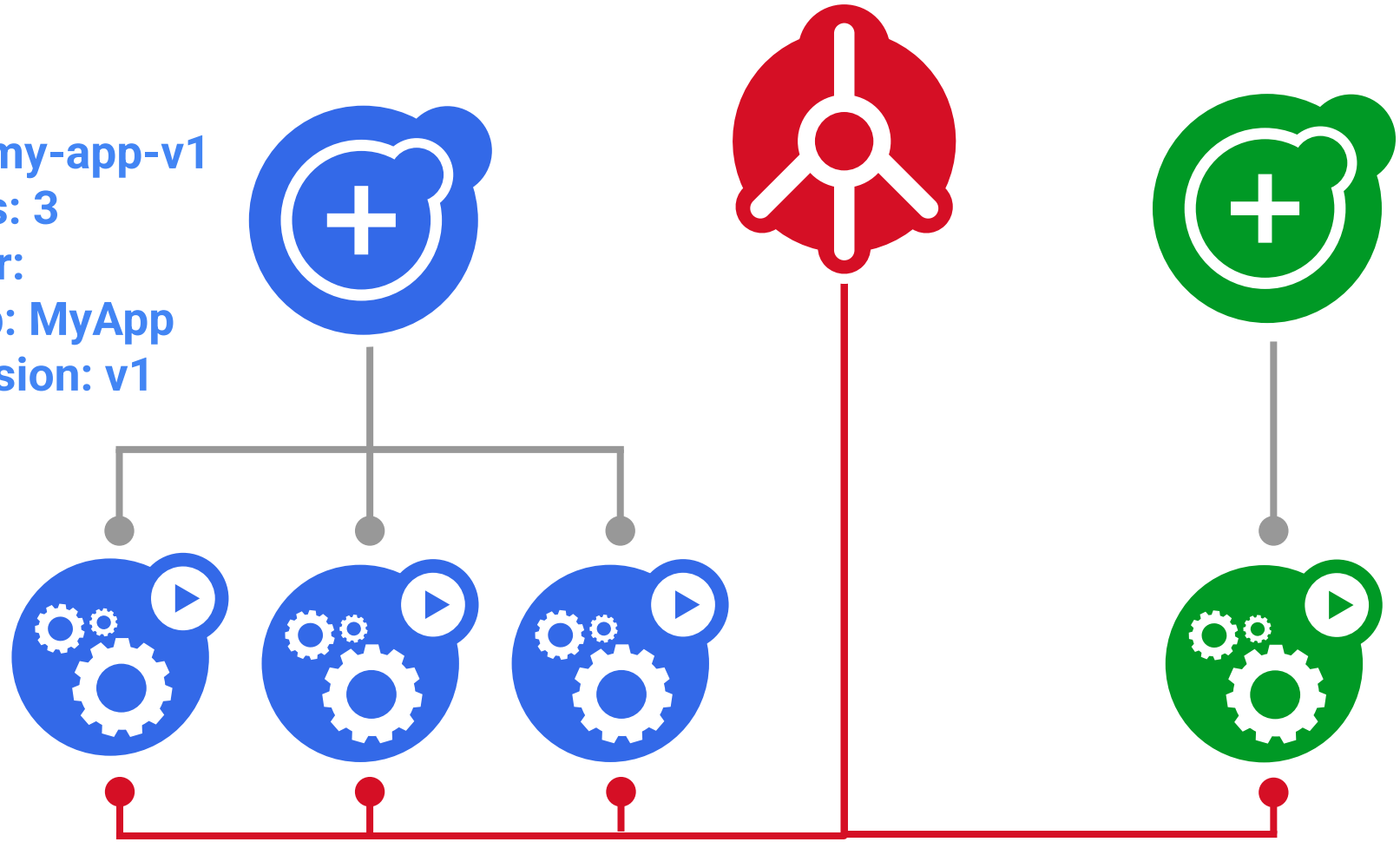
ReplicaSet
- name: my-app-v1
- replicas: 3
- selector:
  - app: MyApp
  - version: v1

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

Rolling Update

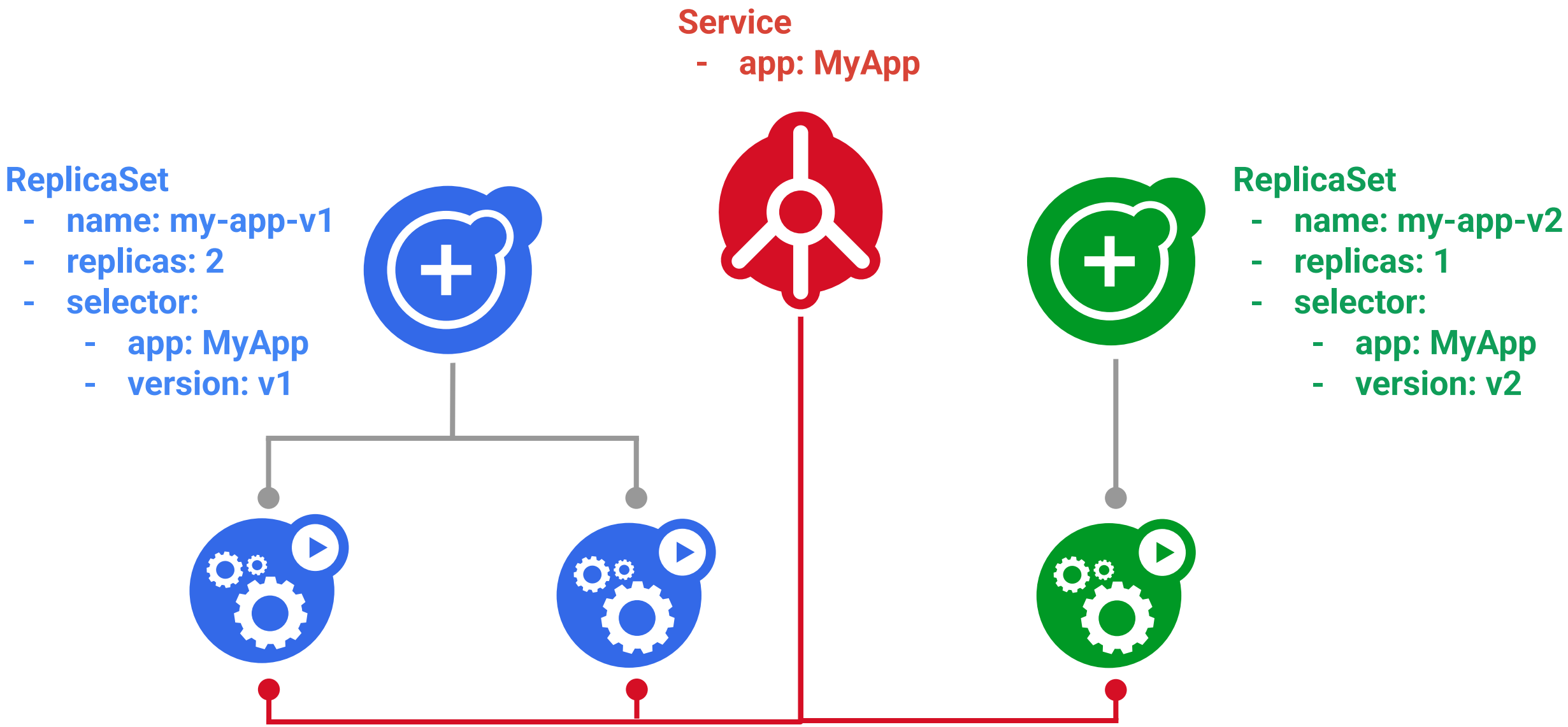
Service
- app: MyApp

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



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

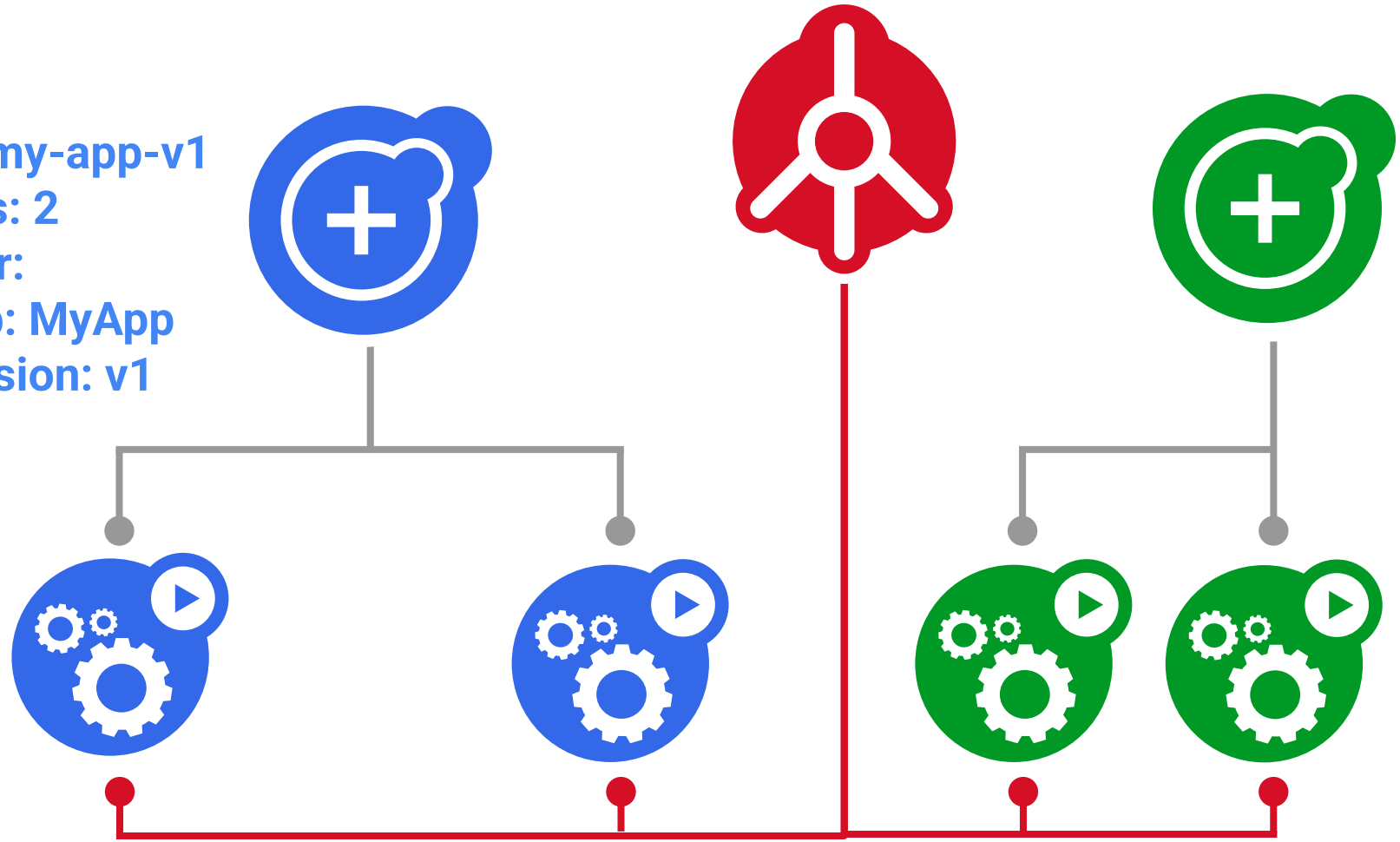
Rolling Update



Rolling Update

Service
- app: MyApp

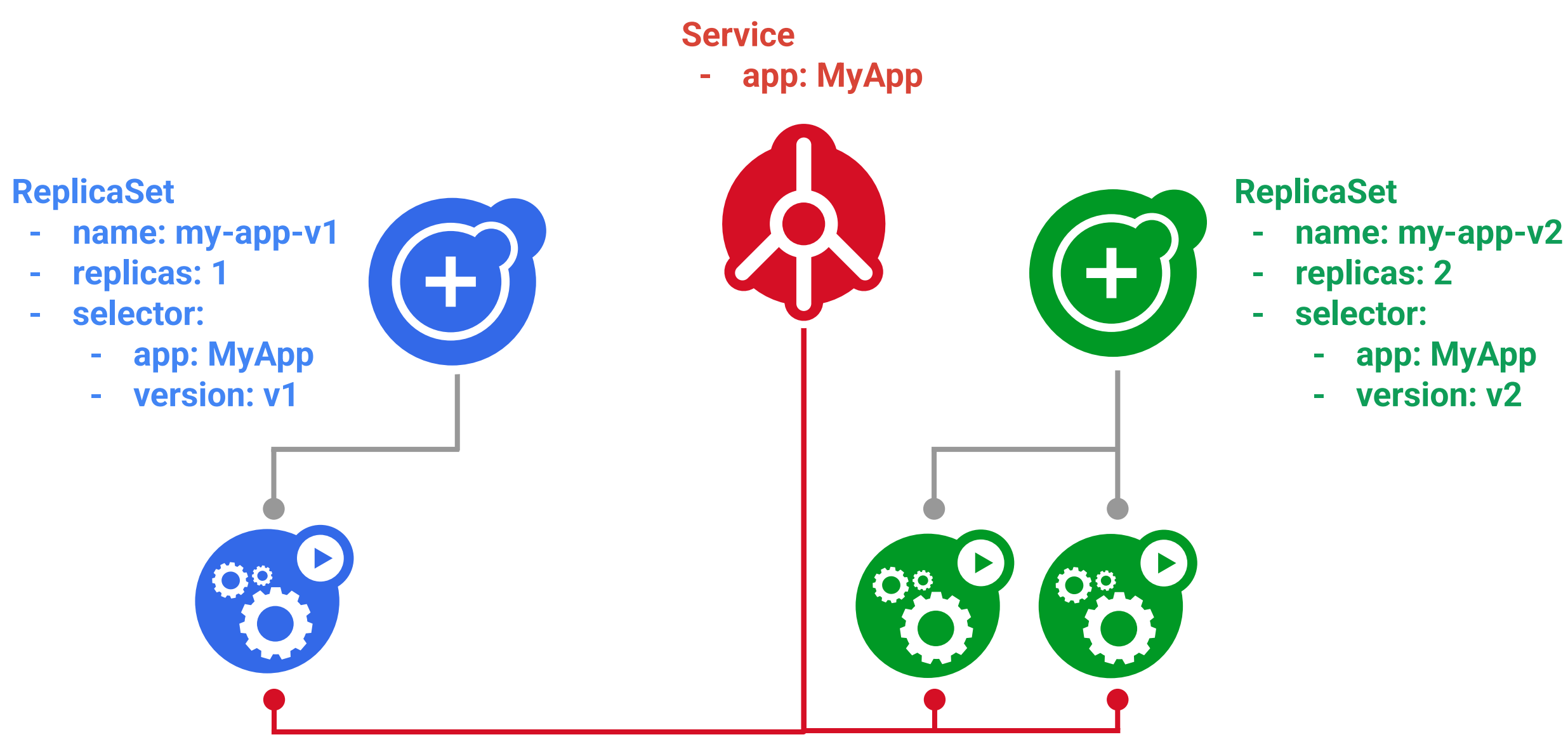
ReplicaSet
- name: my-app-v1
- replicas: 2
- selector:
  - app: MyApp
  - version: v1



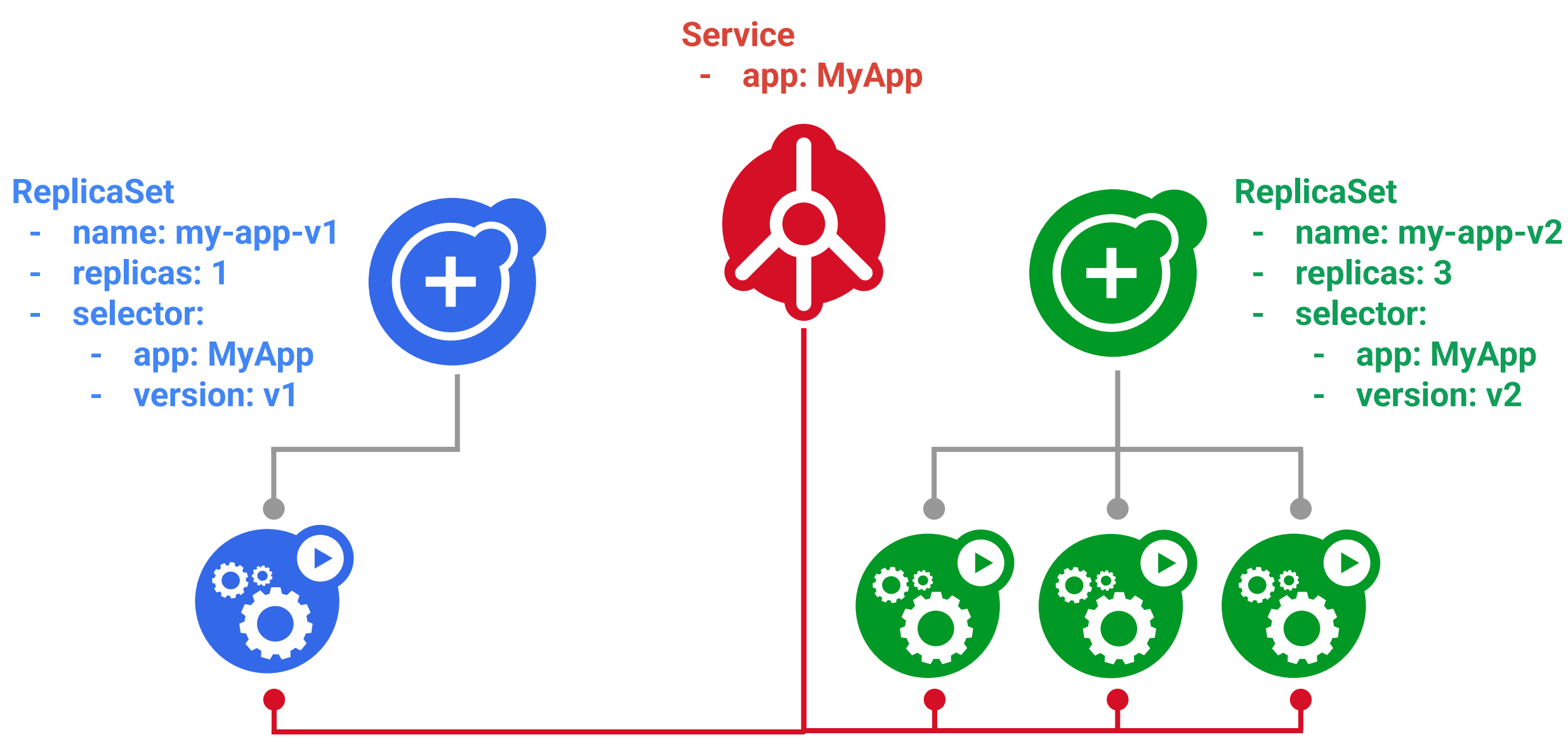
ReplicaSet
- name: my-app-v2
- replicas: 2
- selector:
  - app: MyApp
  - version: v2

http://k8s-school.fr

Rolling Update



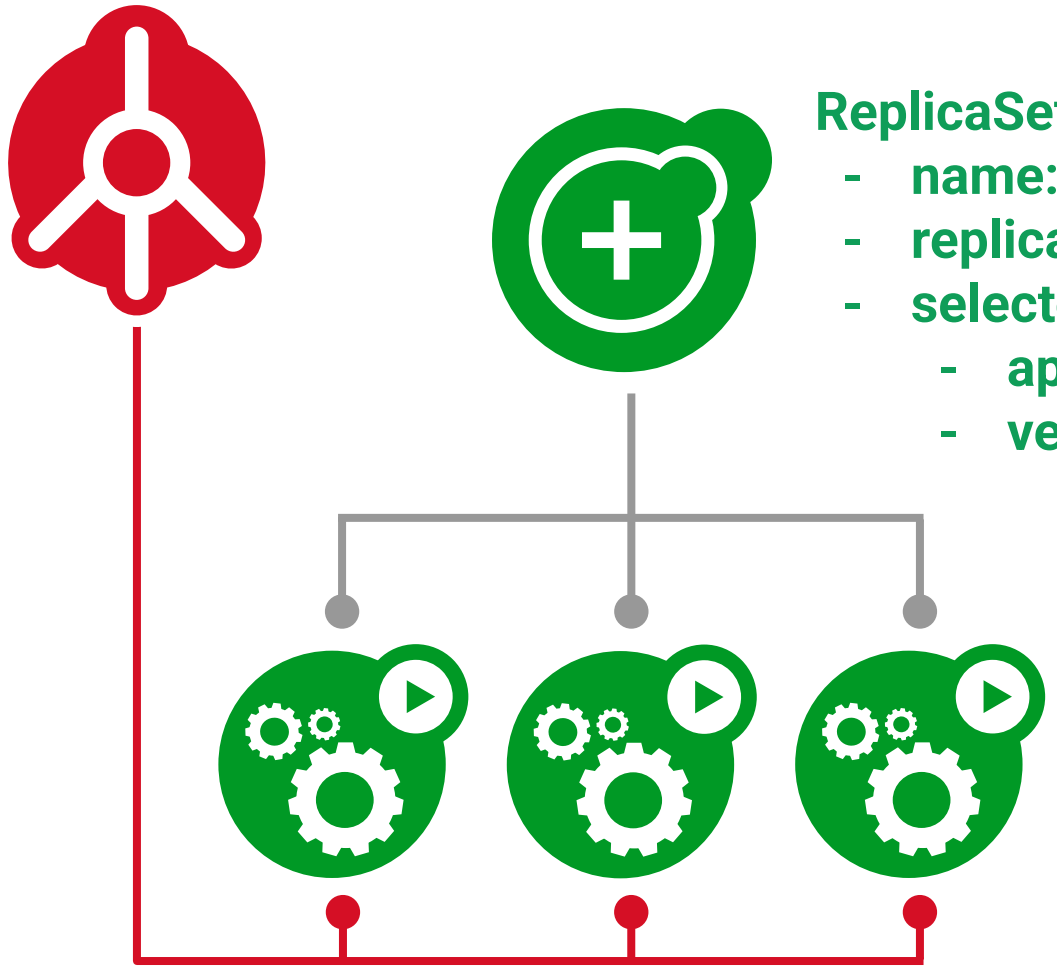
Rolling Update



Rolling Update

Service
- app: MyApp

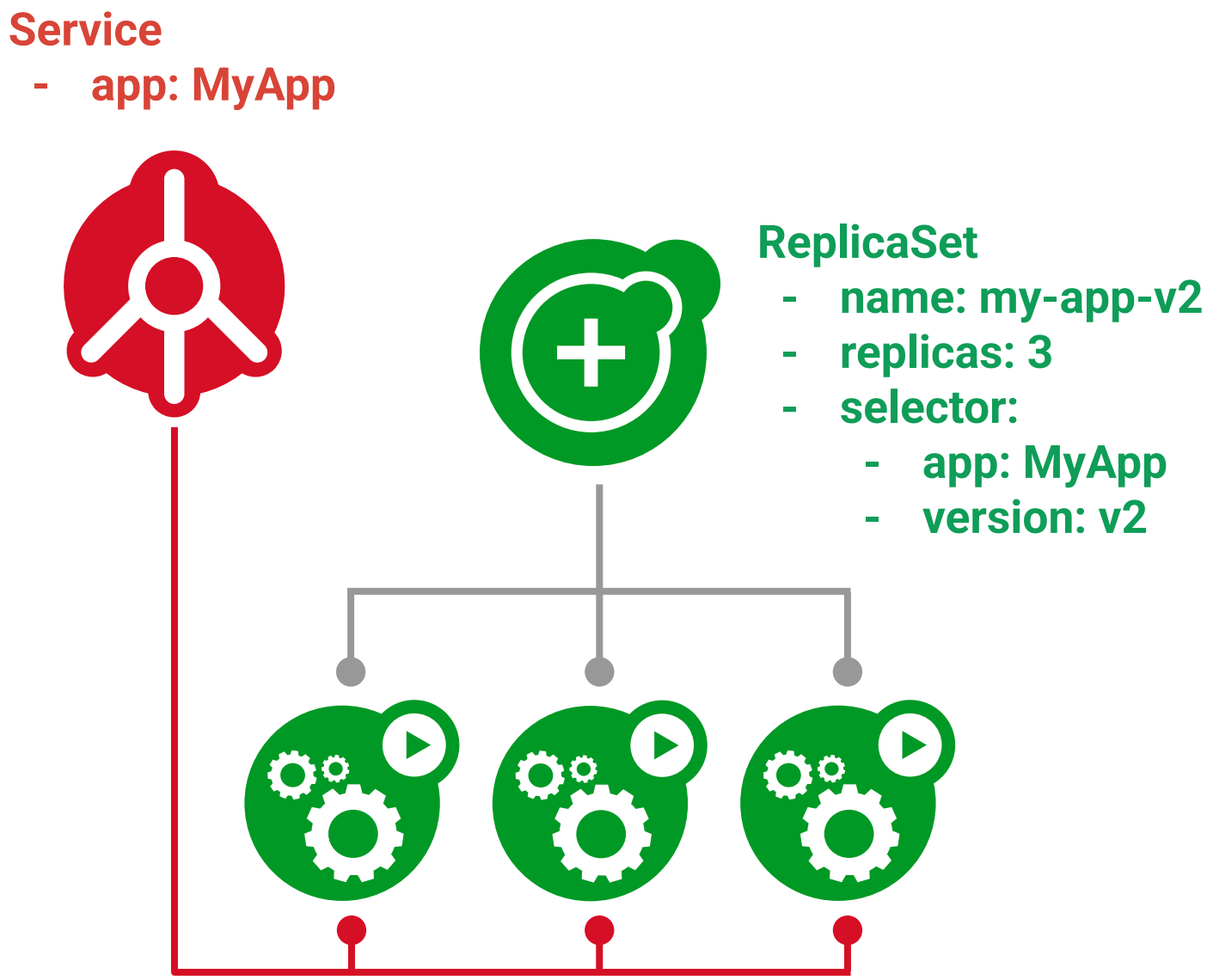
ReplicaSet
- name: my-app-v1
- replicas: 0
- selector:
  - app: MyApp
  - version: v1



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

http://k8s-school.fr

Rolling Update

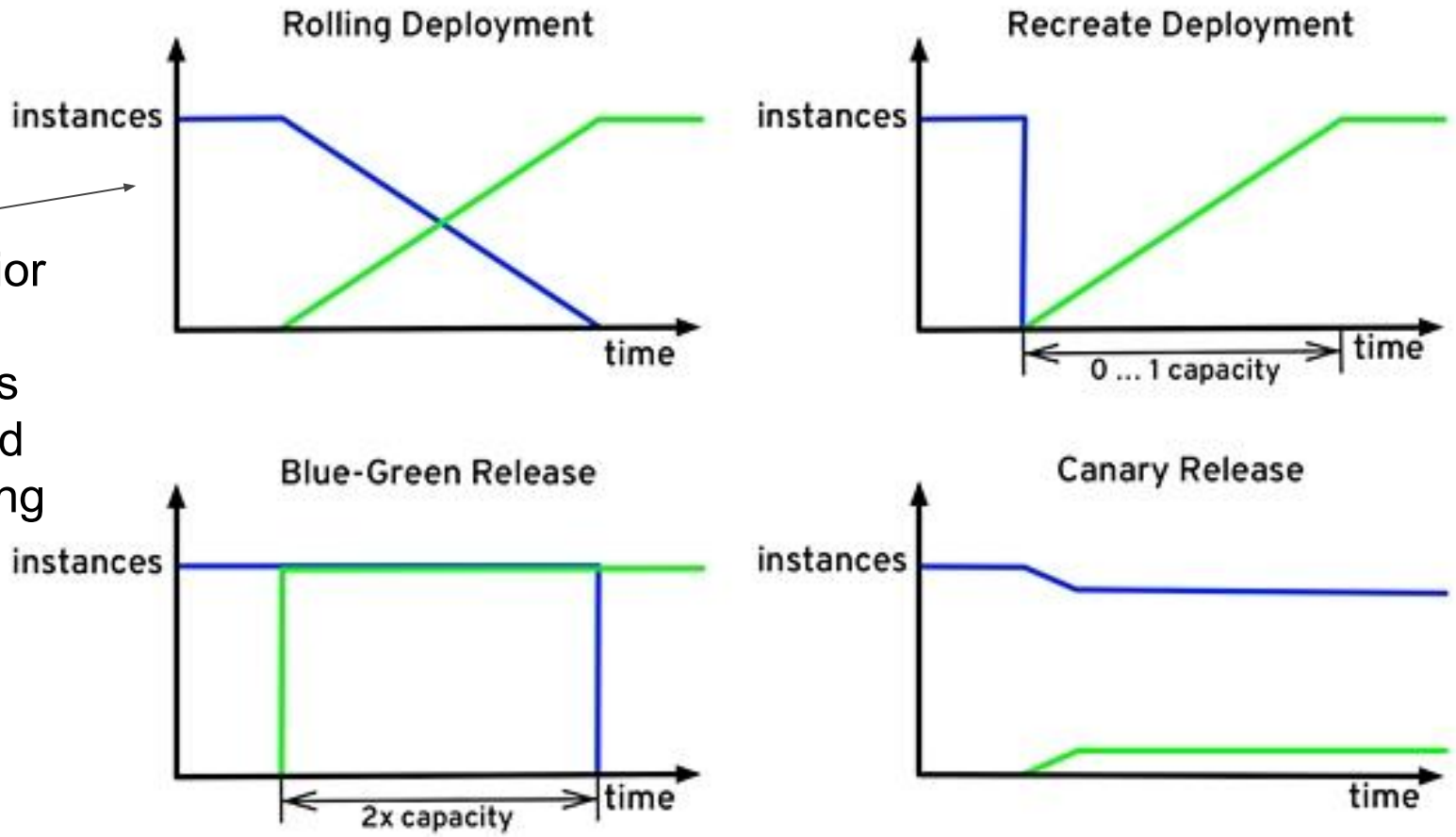


http://k8s-school.fr

Various possible Deployment upgrade strategies

The built-in Deployment behavior

The other strategies can be implemented fairly easily by talking to the API.



Picture source: [Kubernetes effect by Bilgin Ibryam]

Deployments

Deployments

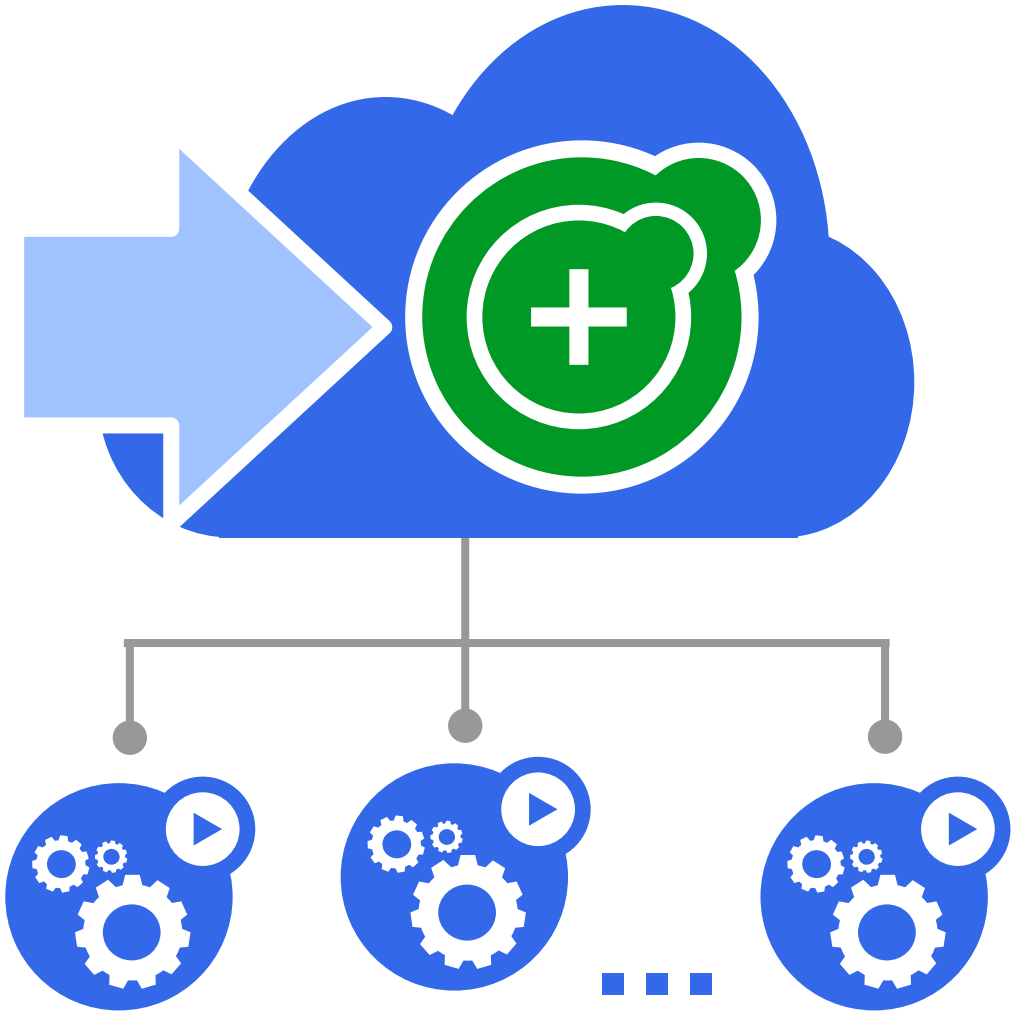
Updates-as-a-service

- 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



http://k8s-school.fr

Deployments: exercice

Perform a rolling update

1. Create the deployment

```
kubectl apply -f 10-0-deployment.yaml
```

2. Change image name and annotation (blue to green)
3. Apply the new file
4. Monitor the update

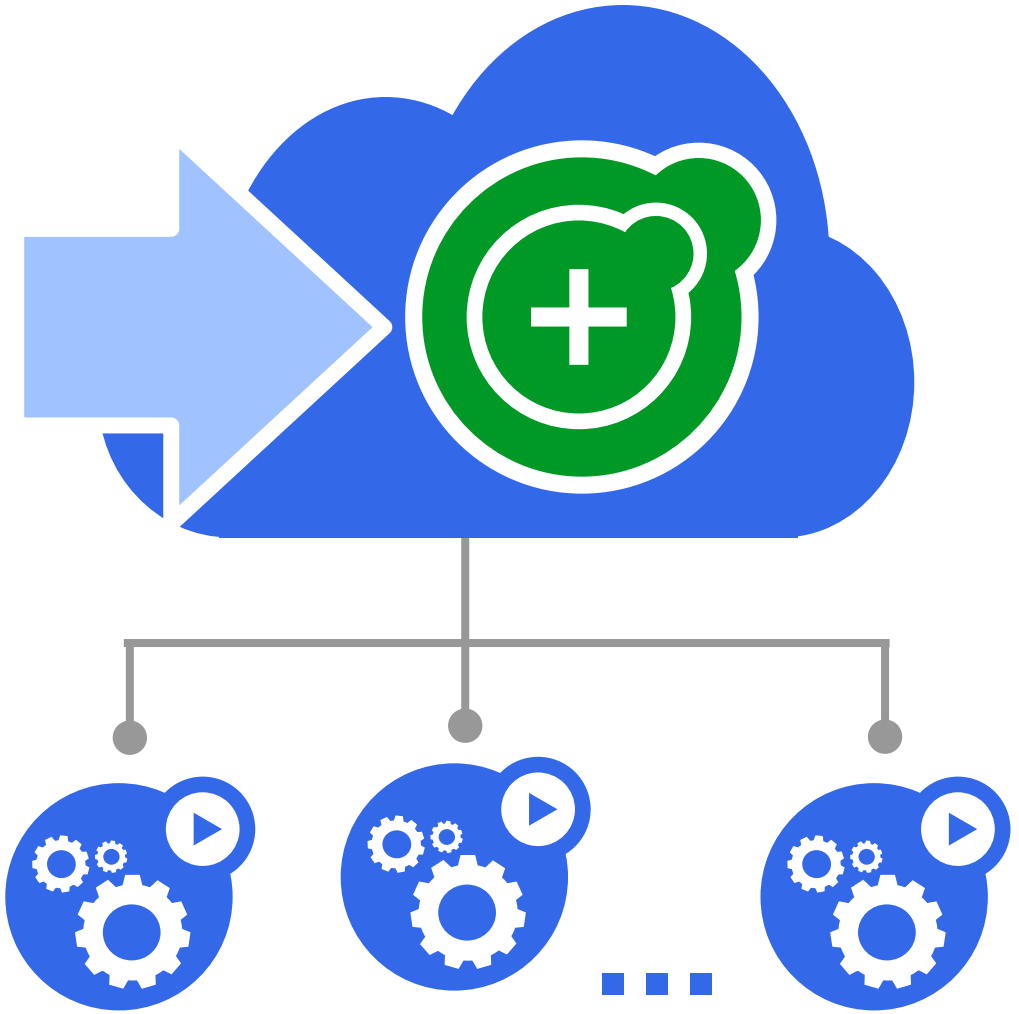
```
kubectl rollout status deployments kuard
```

5. Update to kuard 1.9.10
6. Check history

```
kubectl rollout history deployment kuard
--revision=2
```

7. Rollback to a working version

```
kubectl rollout undo deployments kuard
--to-revision=??
```



cf. Kubernetes Up and Running, 3rd Edition.pdf , p. 149

Updating a Container Image

```
# Pause and restart an upgrade
kubectl rollout pause deployments kuard
kubectl rollout resume deployments kuard

# View history
kubectl rollout history deployment kuard
kubectl rollout history deployment kuard --revision=2

# Undo an upgrade
kubectl rollout undo deployments kuard --to-revision=3
```

Running Daemons

DaemonSets

Problem: how to run a Pod on every node?

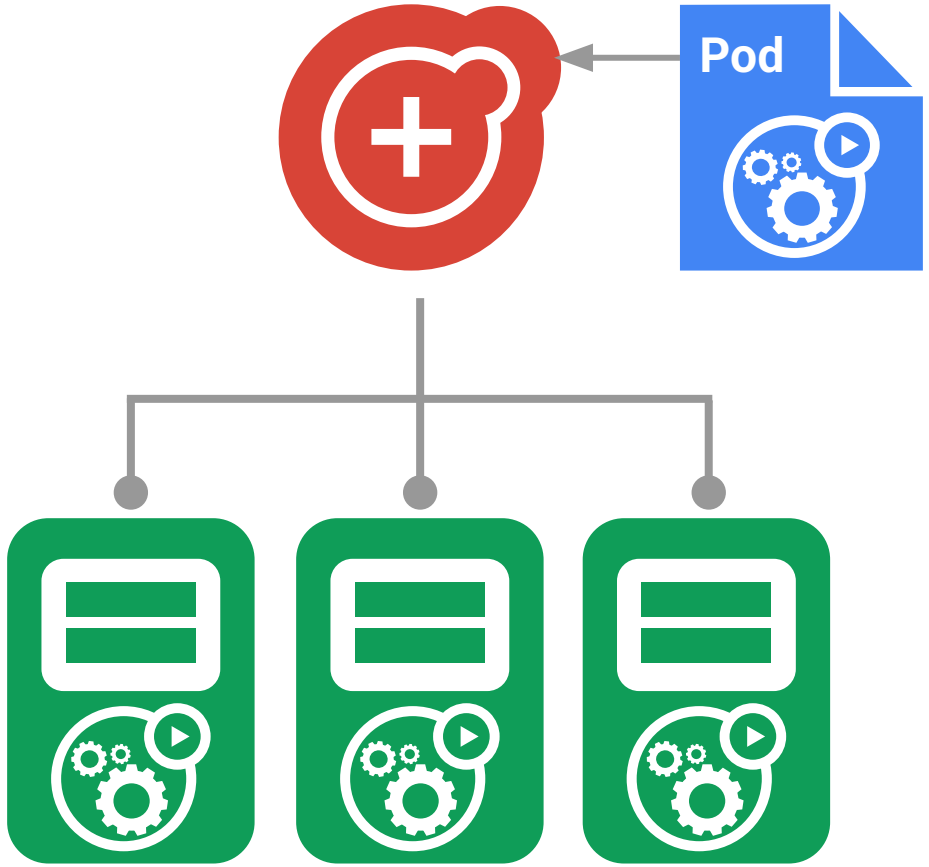
- or a subset of nodes

Similar to ReplicaSet

- principle: do one thing, don't overload

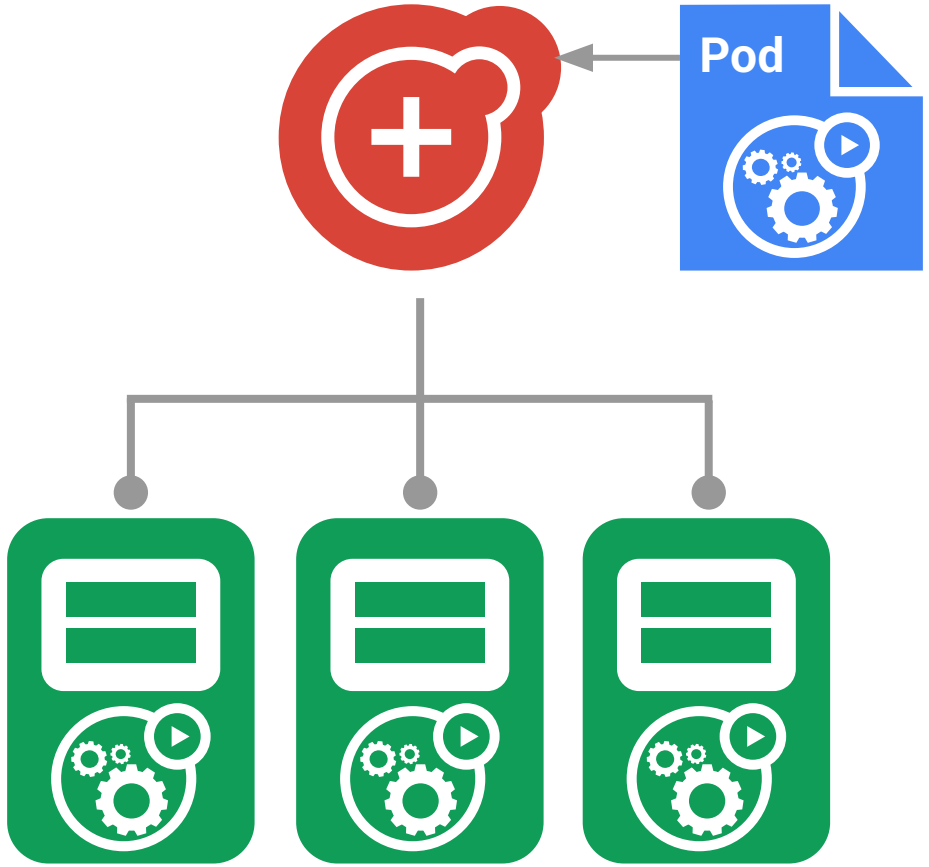
"Which nodes?" is a selector

Uses consistent tools and patterns



DaemonSets: exercice

1. Create fluentd daemonset `11-1-fluentd.yaml` (change Daemonset name using your ID, + label/selector)
2. Find out the kubectl command to list fluentd pods and their related node
3. Apply following label to a set of node `“user-<ID>/log=true”`
4. Edit `11-1-fluentd.yaml` with a `nodeSelector` and apply it.



Running Jobs

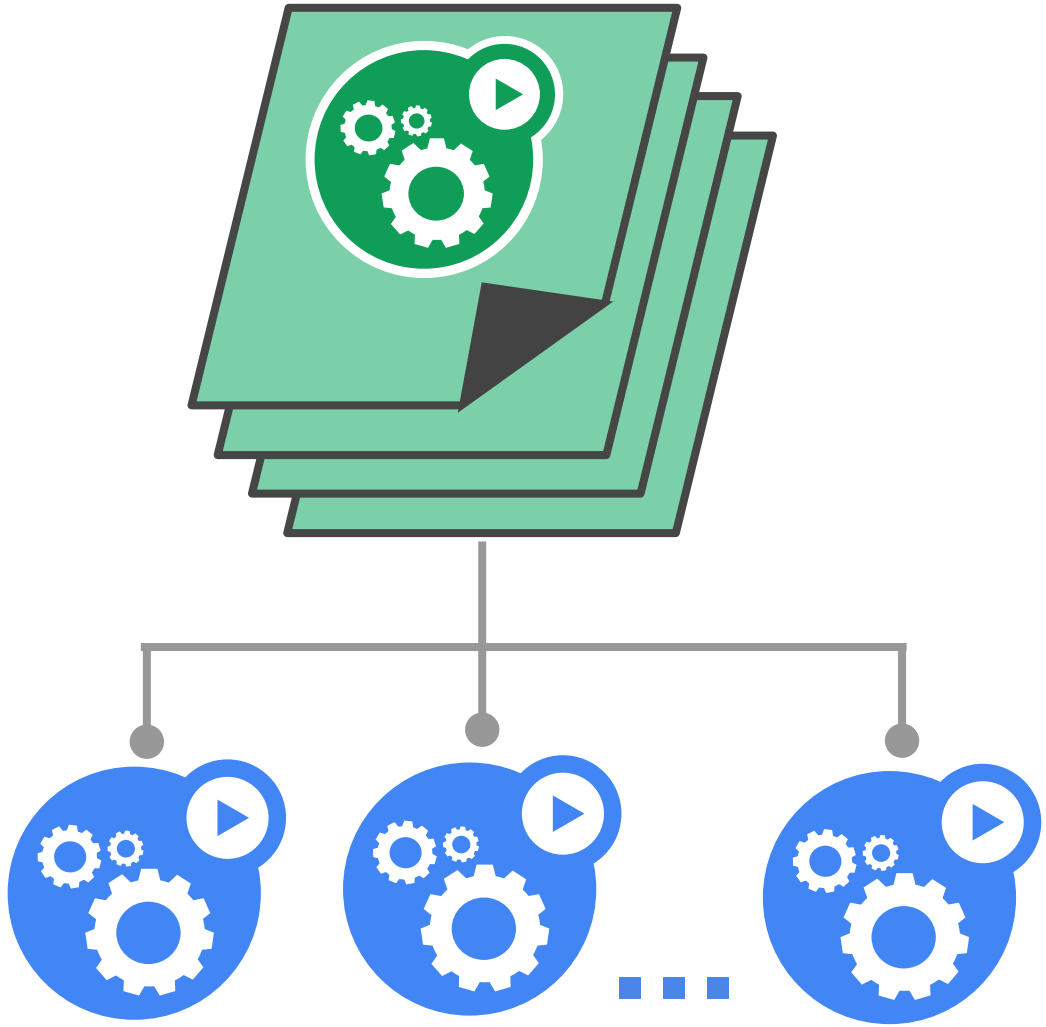
Jobs

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



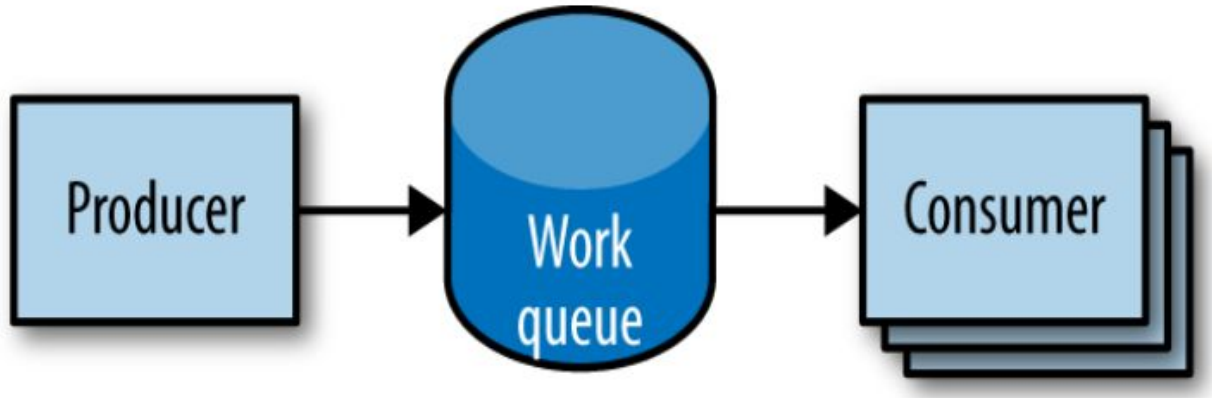
http://k8s-school.fr

Jobs: exercice

1. **Create the queue:**
   12-4-rs-queue.yaml
2. **Port-forward to the queue pod**
3. **Expose the queue pod :**
   12-5-service-queue.yaml
4. Load the queue with commands on the left (+ port-forward)
5. Create a job which consume the queue:
   12-7-job-consumers.yaml
6. Monitor it

cf. “Kubernetes Up and Running”, p146

```
# Producer
# --------
PORT=808<ID>
kubectl port-forward xxxx &

# Create a work queue called 'keygen'
curl -X PUT localhost:$PORT/memq/server/queues/keygen

# Create 100 work items and load up the queue.
for i in work-item-{0..99}; do
   curl -X POST localhost:$PORT/memq/server/queues/keygen/enqueue \
        -d "$i"
done

curl localhost:$PORT/memq/server/stats
```



ConfigMaps

ConfigMaps

Goal: manage app configuration

- ...without making overly-brittle container images

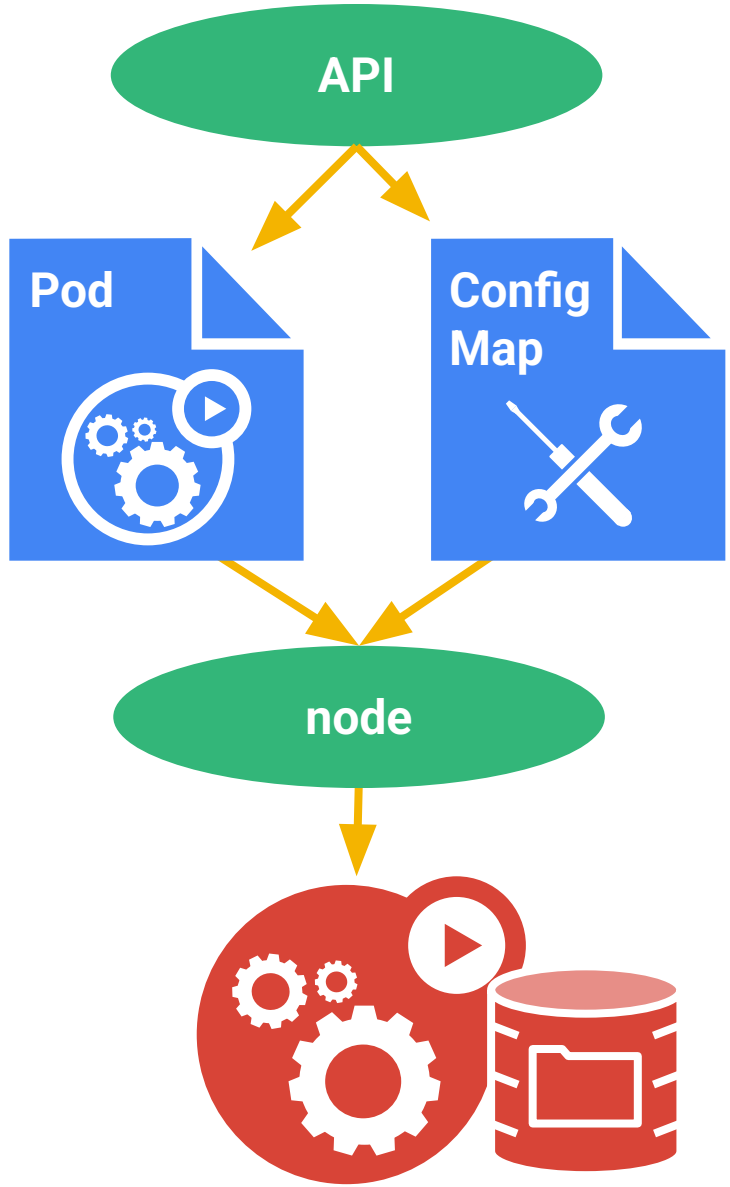
12-factor says config comes from the environment

- 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



http://k8s-school.fr

ConfigMaps: exercice

Create a ConfigMap

```
kubectl create configmap my-config \
  --from-file my-config.txt \
  --from-literal extra-param=extra-value \
  --from-literal another-param=another-value
```

Create a Pod which uses ConfigMap “my-config”

```
kubectl apply -f 13-2-kuard-config.yaml
```

Access file `my-config.txt` **and** `EXTRA-PARAM, ANOTHER-PARAM` **using** `“kubectl exec -it <podname> -- sh”`

Edit the ConfigMap and check what happens inside the Pod

Secrets

Secrets

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

- don't put secrets in the container image!

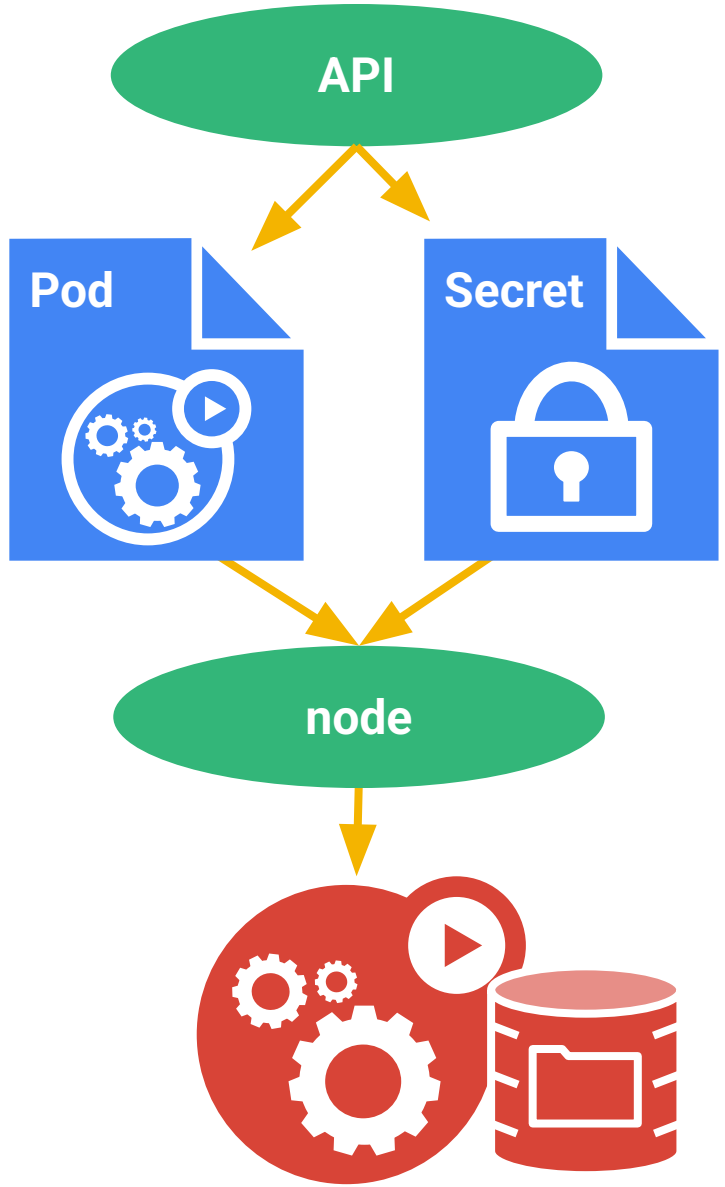
12-factor says config comes from the environment

- 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



Secret : imagePullSecret

Create a `docker-registry` Secret

```
kubectl create secret docker-registry my-image-pull-secret
--docker-server=<your-registry-server> \
--docker-username=<your-name> \
--docker-password=<your-pword> \
--docker-email=<your-email>
```

Create a Pod which uses ConfigMap “my-config”

```
kubectl apply -f 13-4-kuard-secret-ips.yaml
```

PersistentVolumes

PersistentVolumes

A higher-level storage abstraction

- insulation from any one cloud environment

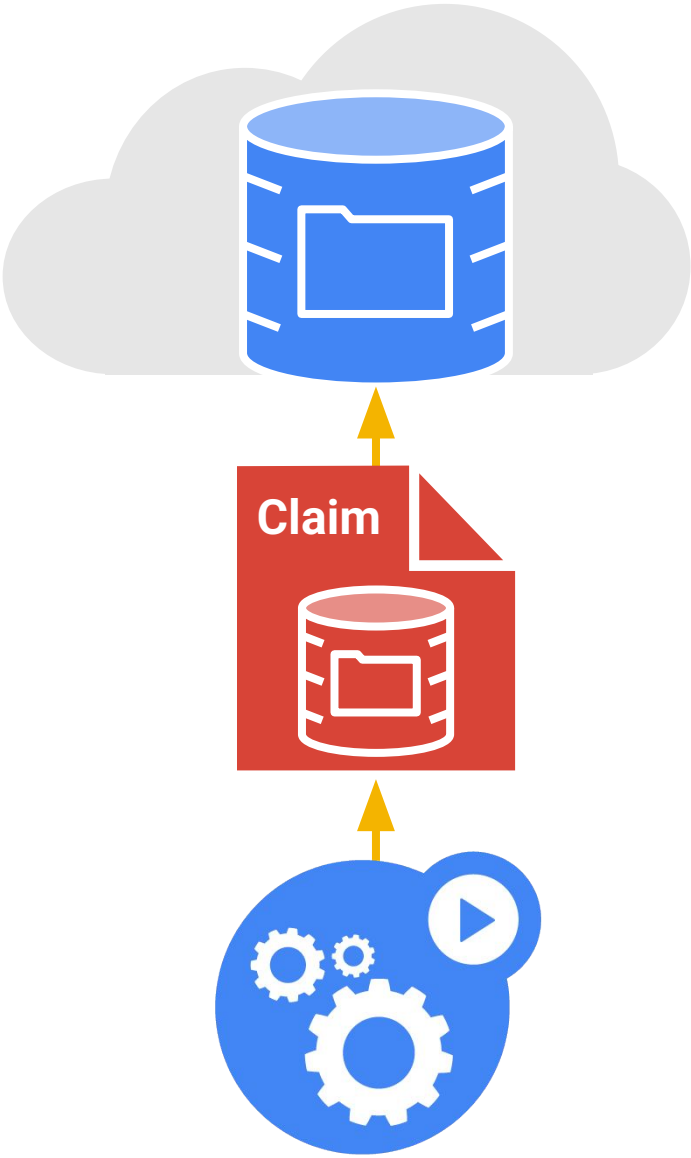
Admin provisions them, users claim them

- **auto-provisioning**

Independent lifetime from consumers

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

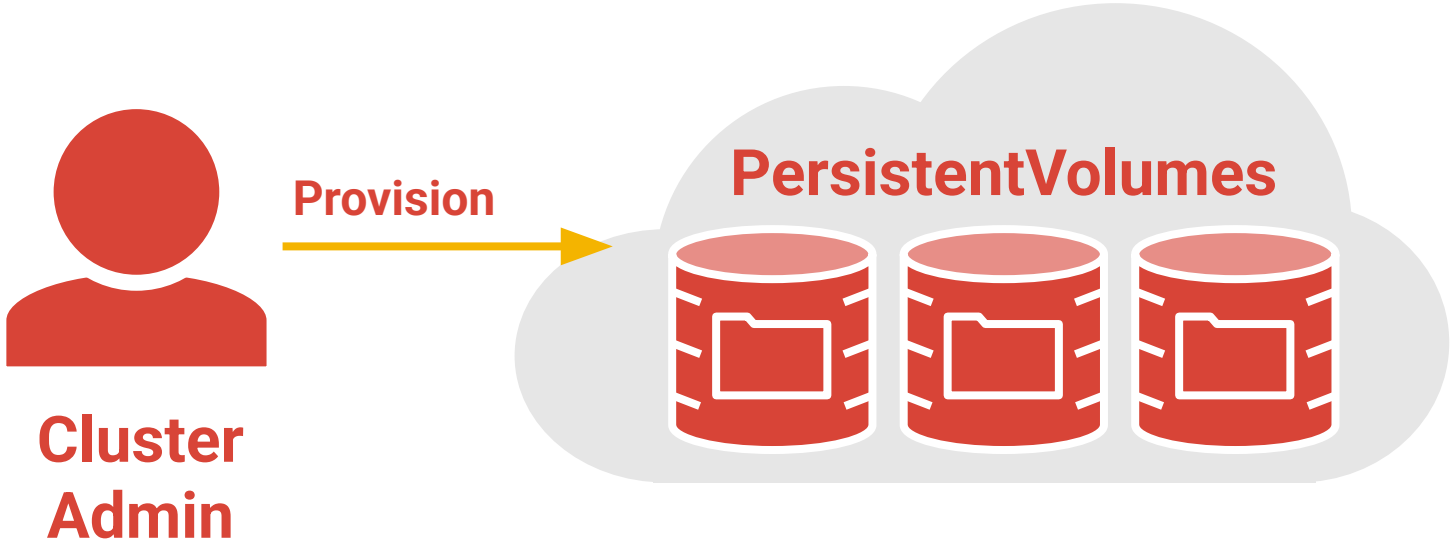
Dynamically “scheduled” and managed, like nodes and pods



PersistentVolumes

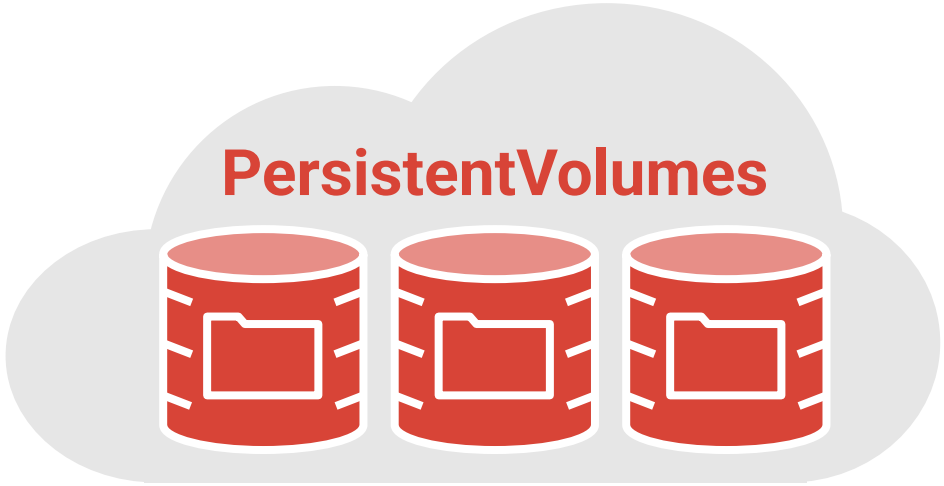
Cluster Admin

PersistentVolumes



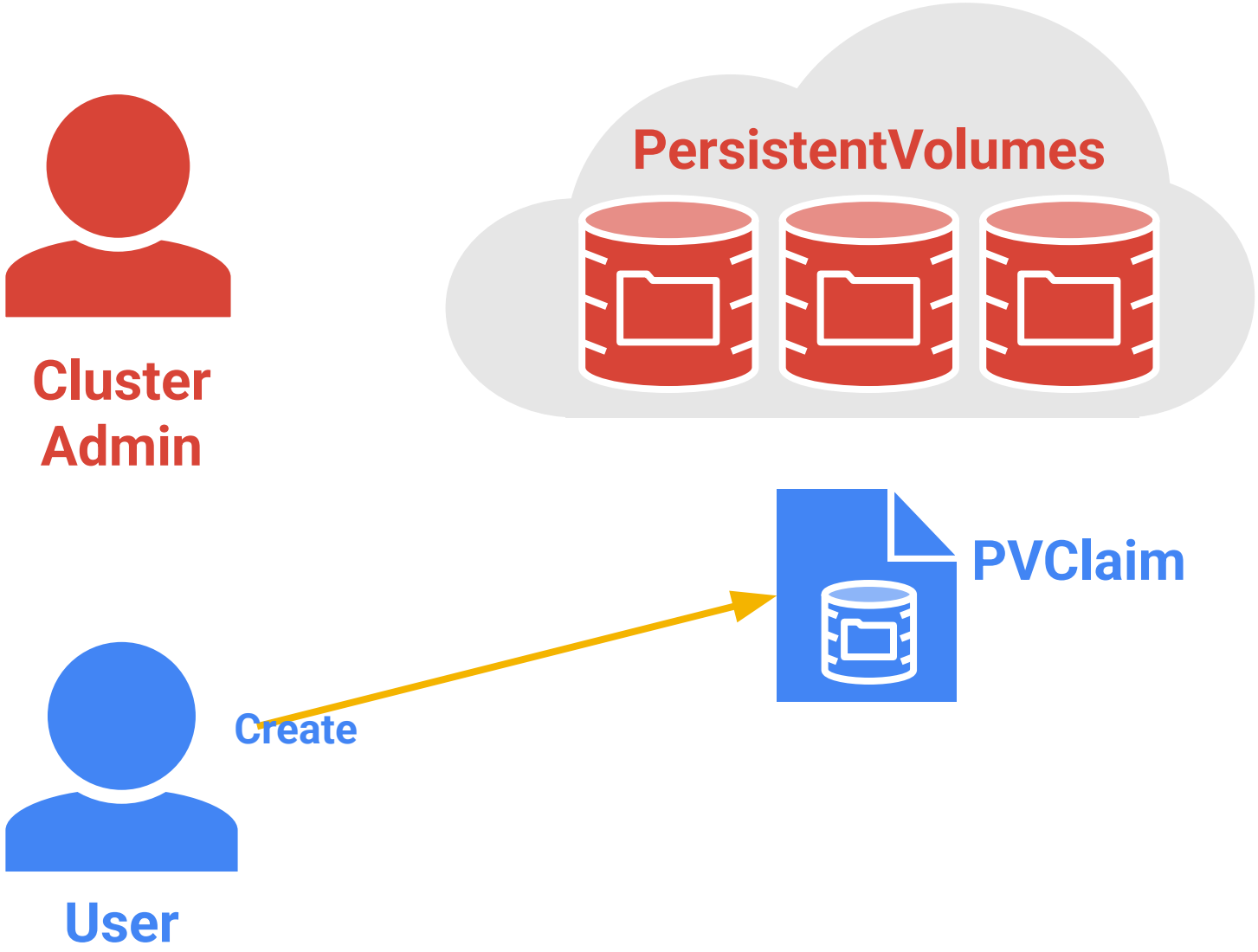
PersistentVolumes

Cluster Admin



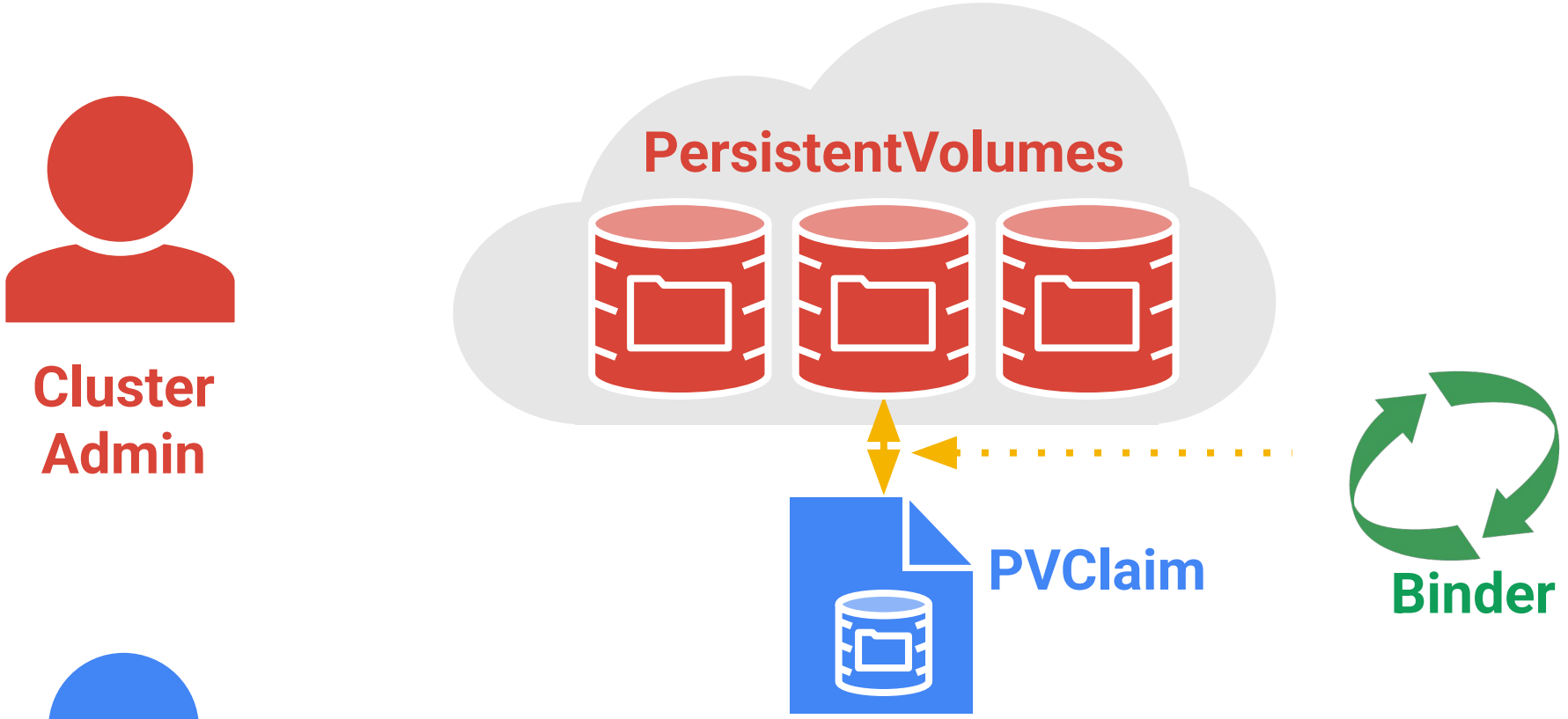
User

PersistentVolumes

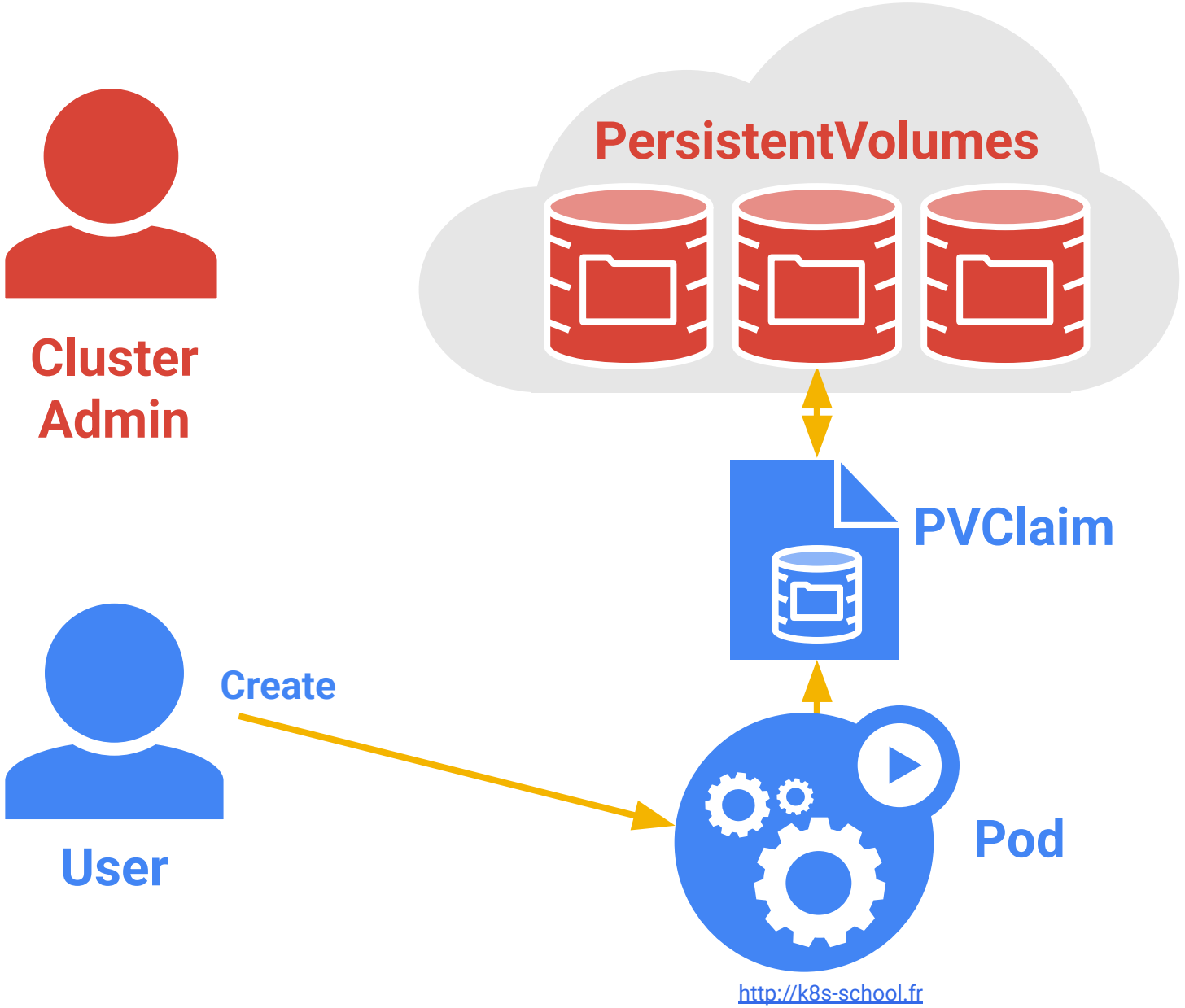


http://k8s-school.fr

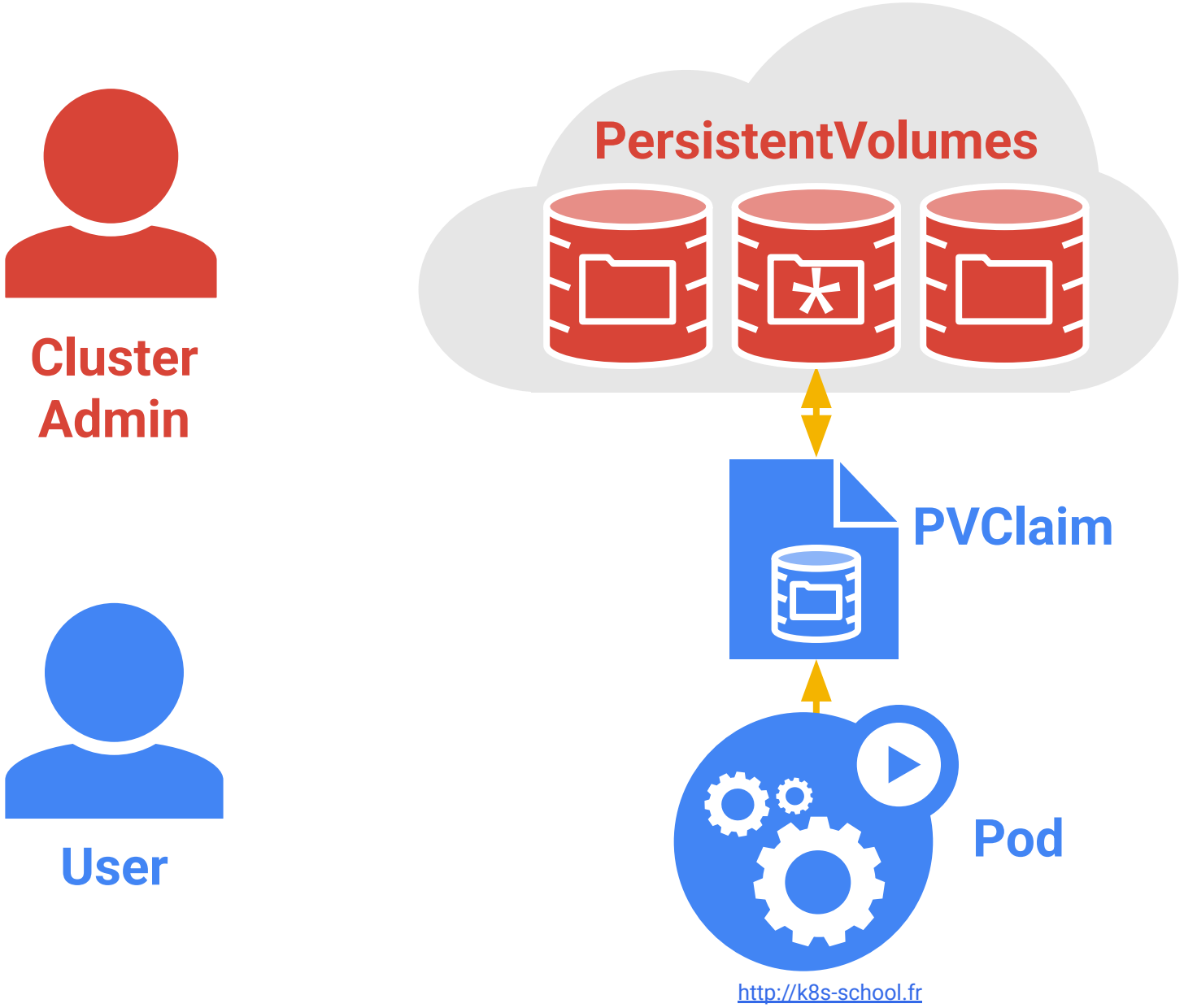
PersistentVolumes



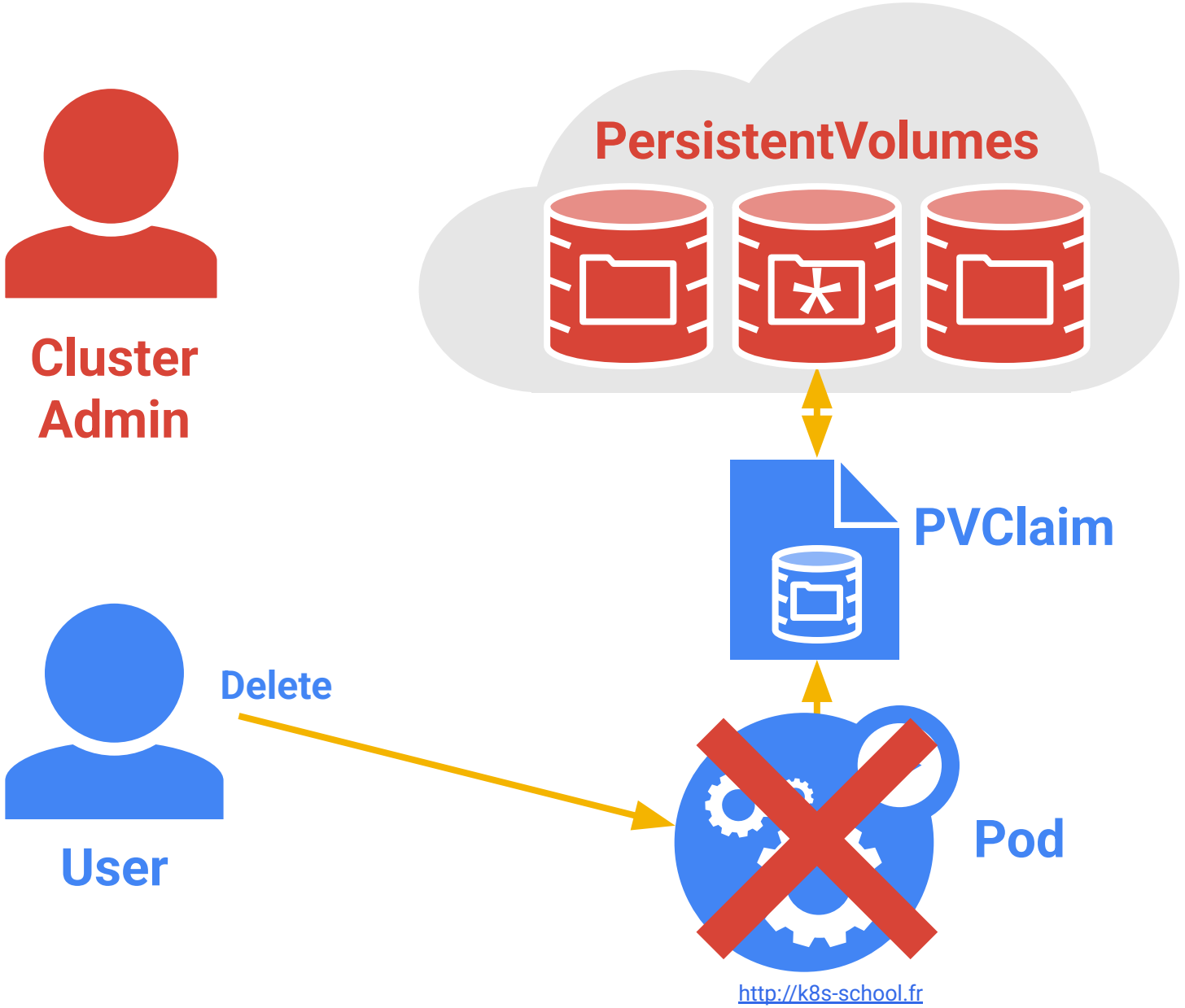
PersistentVolumes



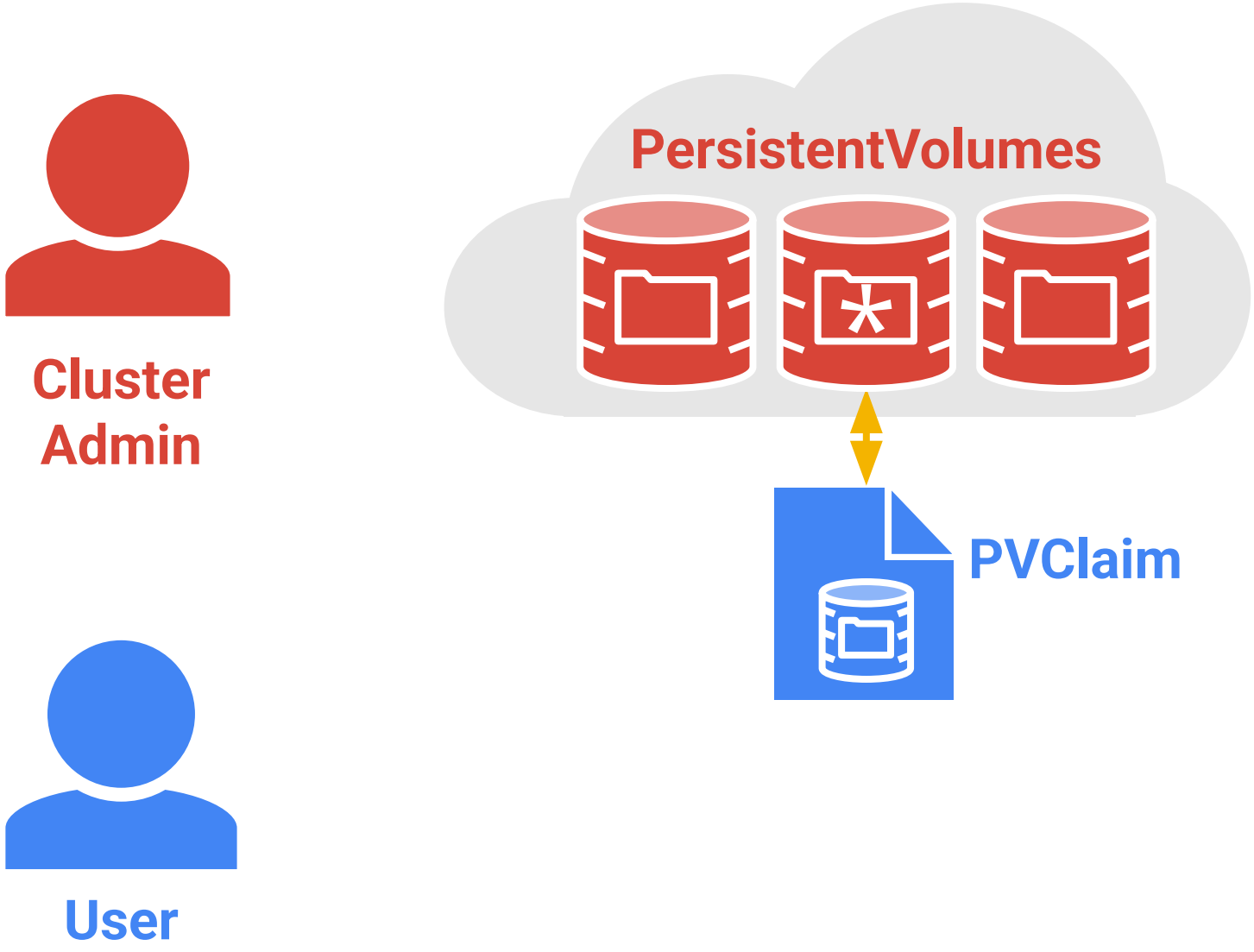
PersistentVolumes



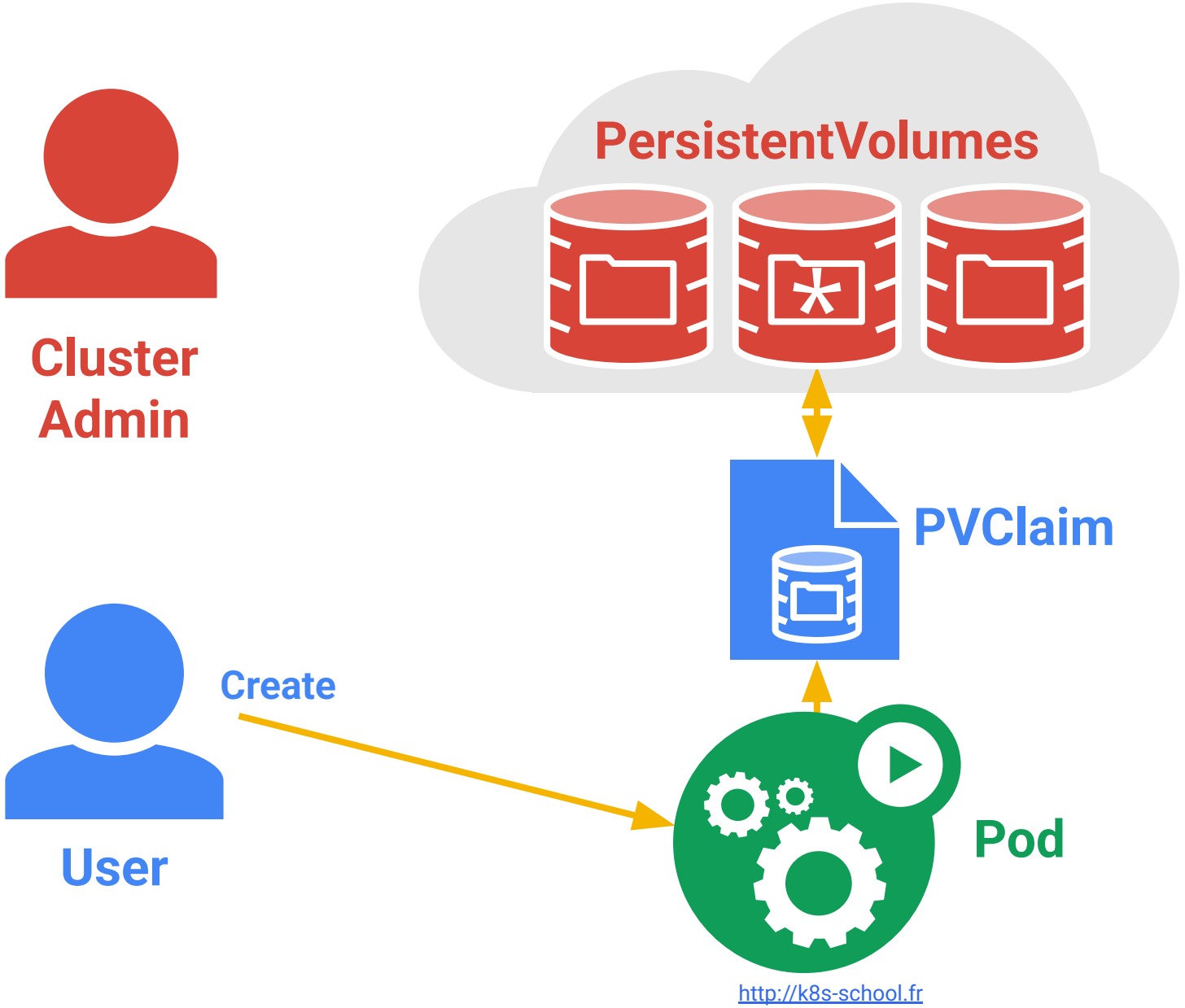
PersistentVolumes



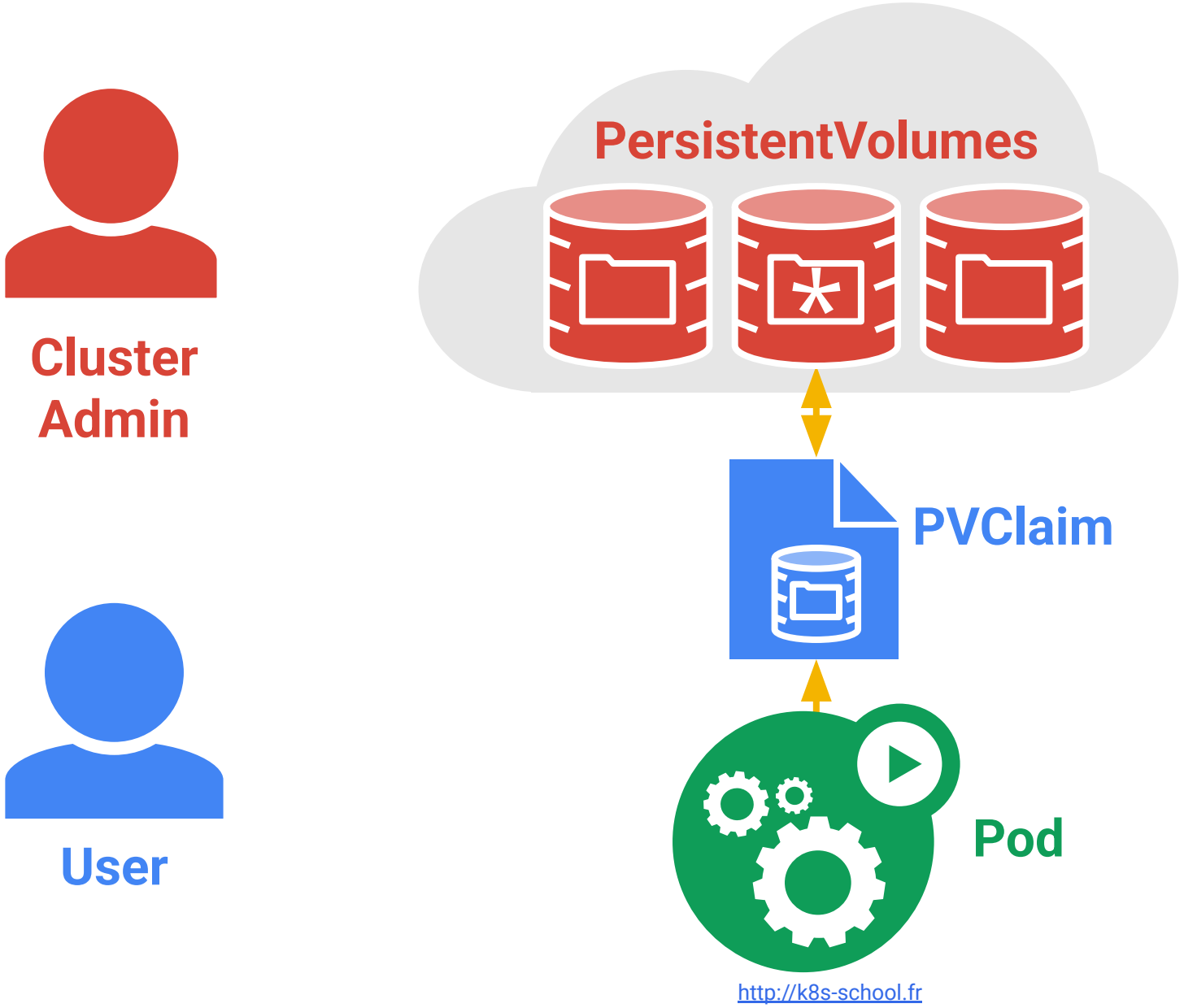
PersistentVolumes



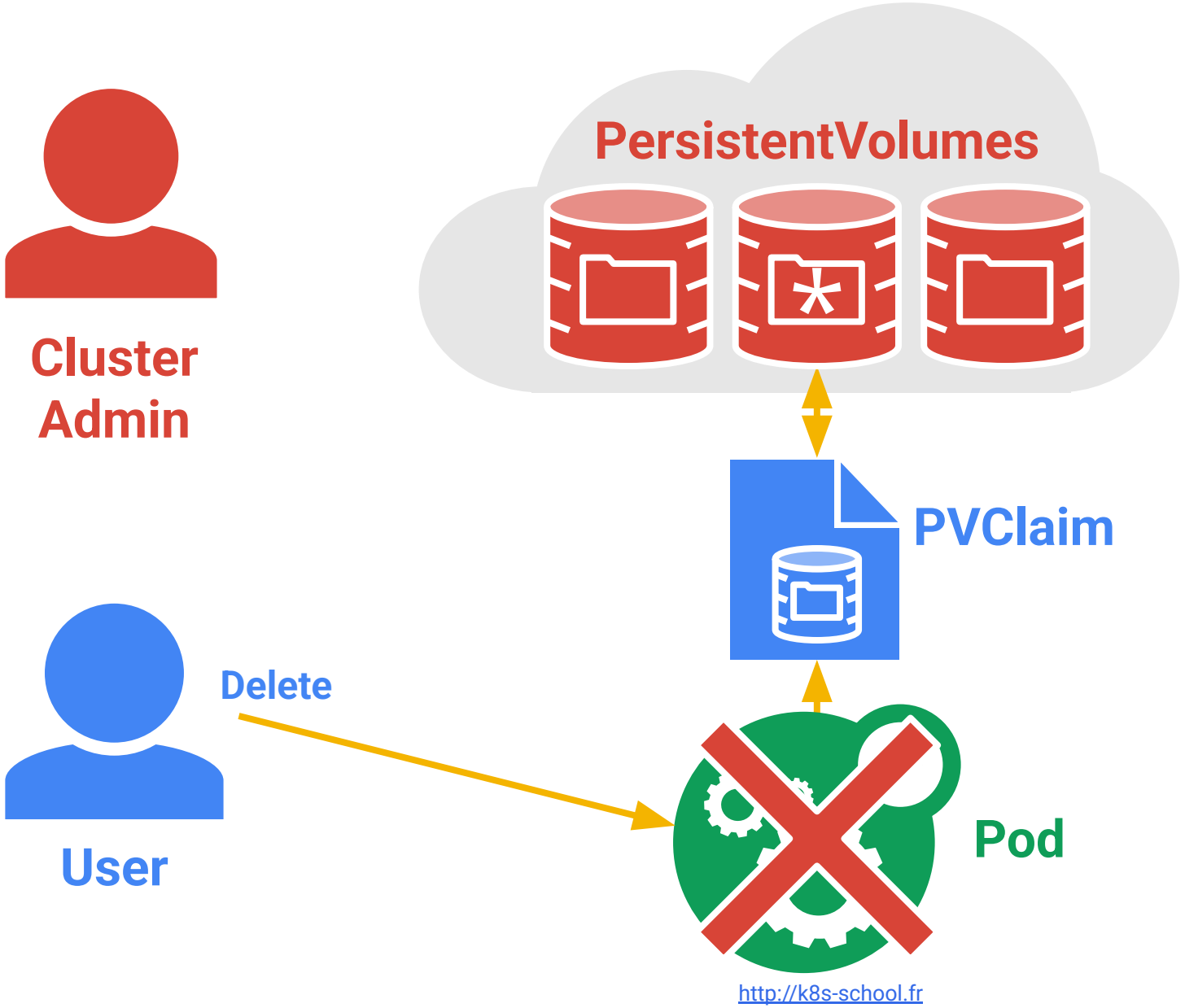
PersistentVolumes



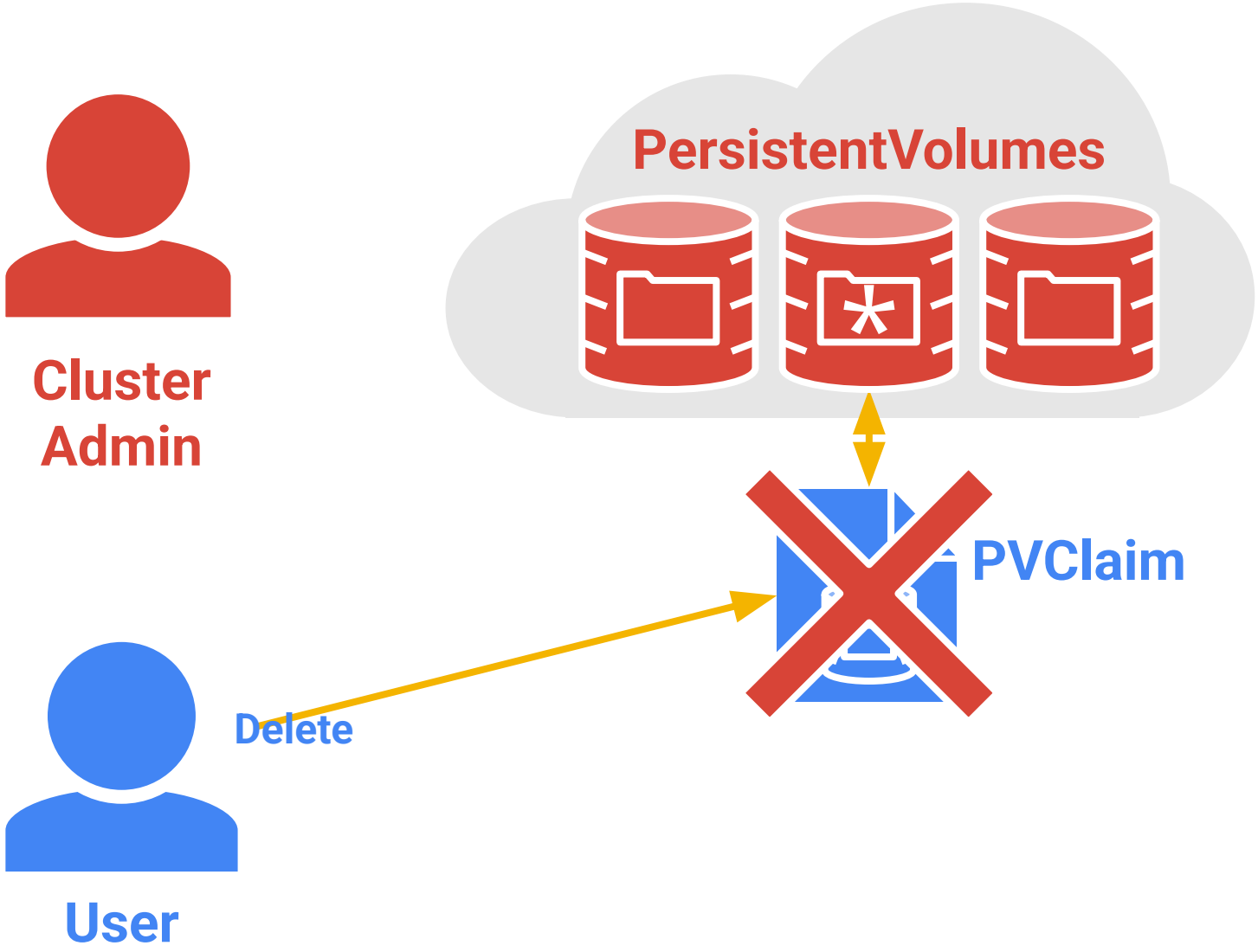
PersistentVolumes



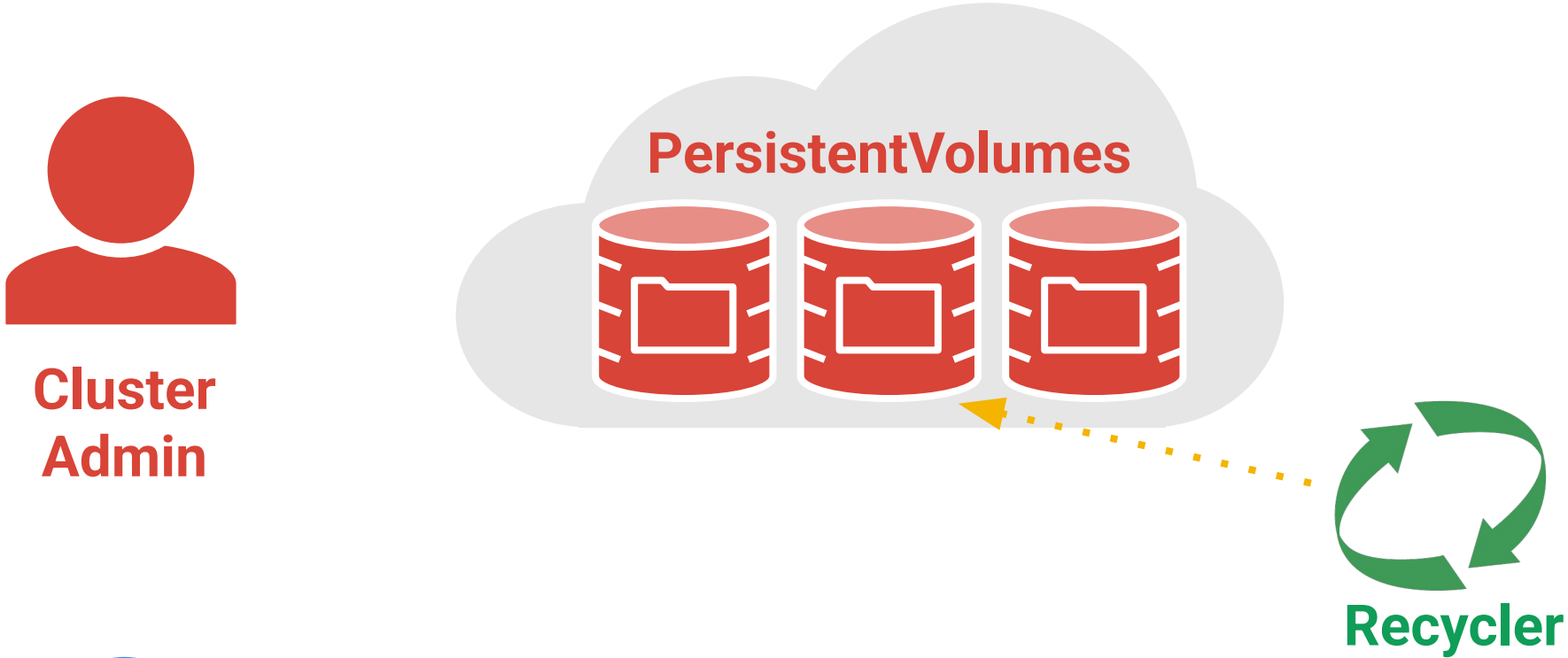
PersistentVolumes



PersistentVolumes



PersistentVolumes



Running With State

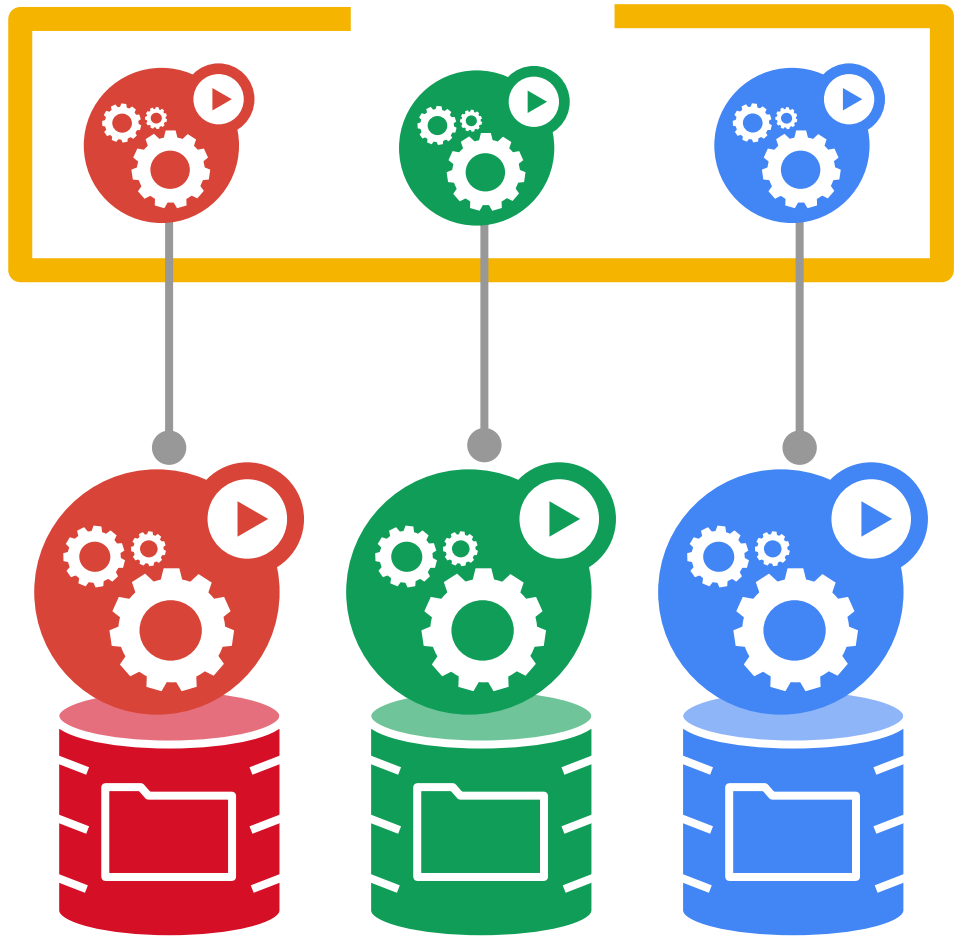
StatefulSets

Goal: enable clustered software on Kubernetes

- mongodb, redis, etcd, ...

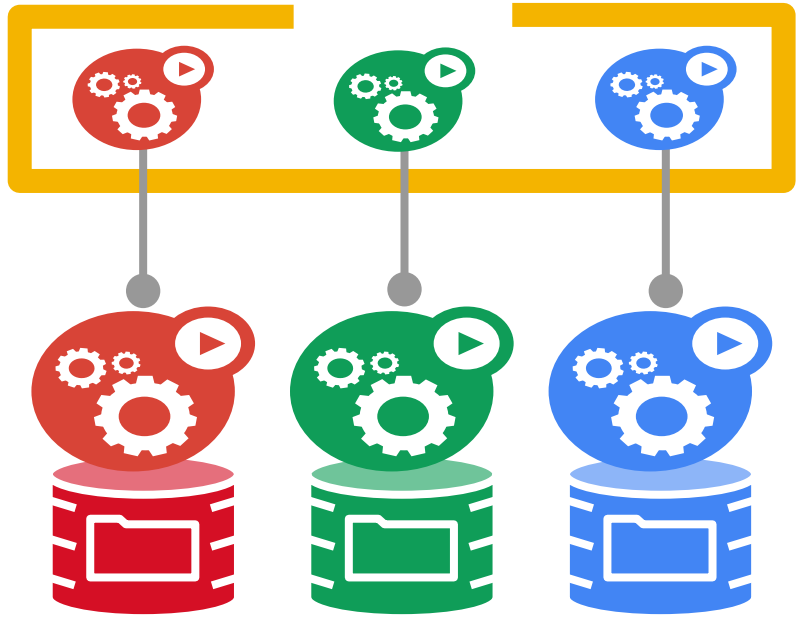
Clustered apps need “identity” and have sequencing constraints

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



http://k8s-school.fr

StatefulSets: exercice



cf. https://k8s-school.fr/labs/fr/1_labs/mongo/index.html

Goal: Create a MongoDB cluster using persistent storage, on kind

1. Use `15-12-mongo-configmap.yaml`
`15-11-mongo-service.yaml 15-13-mongo.yaml`

2. Edit 15-13-mongo.yaml and
   add **volume** and **volumeClaimTemplate** section
   (remove the annotation in the latter)
   See Kubernetes Up and Running, 3rd Edition.pdf p. 212

3. Test and check pv (PersistentVolume) and pvc (PersistentVolumeClaim)
4. Eventually add a liveness probe (optional)

5. Load some data and lauch a query:

```
curl -O https://raw.githubusercontent.com/mongodb/docs-assets/primer-dataset/primer-dataset.json
cat primer-dataset.json
cat primer-dataset.json | kubectl exec -i mongo-0 -c mongodb -- mongoimport --db test --collection restaurants --drop
kubectl exec -it mongo-0 -- mongo test --eval "db.restaurants.find()"
```

Ingress (L7 LB)

Many apps are HTTP/HTTPS

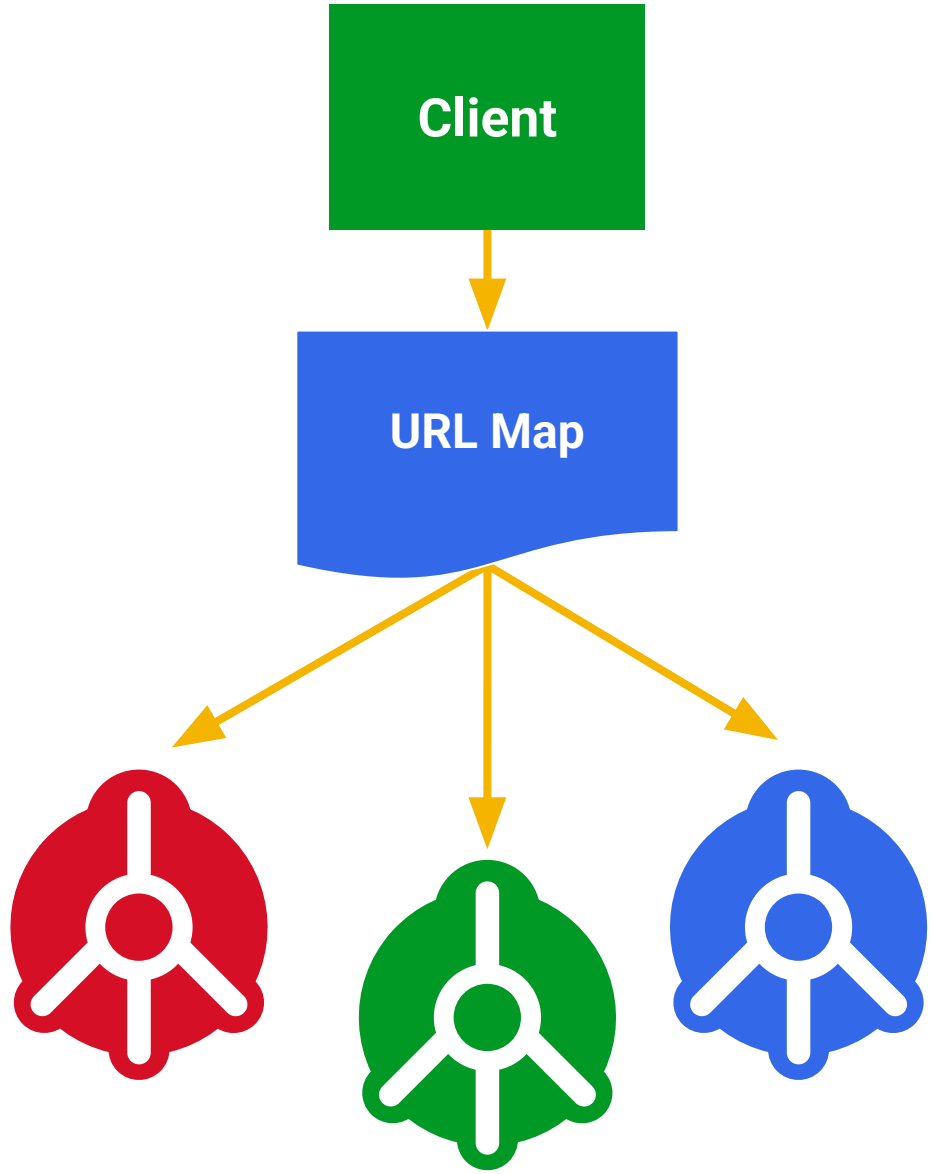
Services are L4 (IP + port)

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!



http://k8s-school.fr

Graceful Termination

Graceful Termination

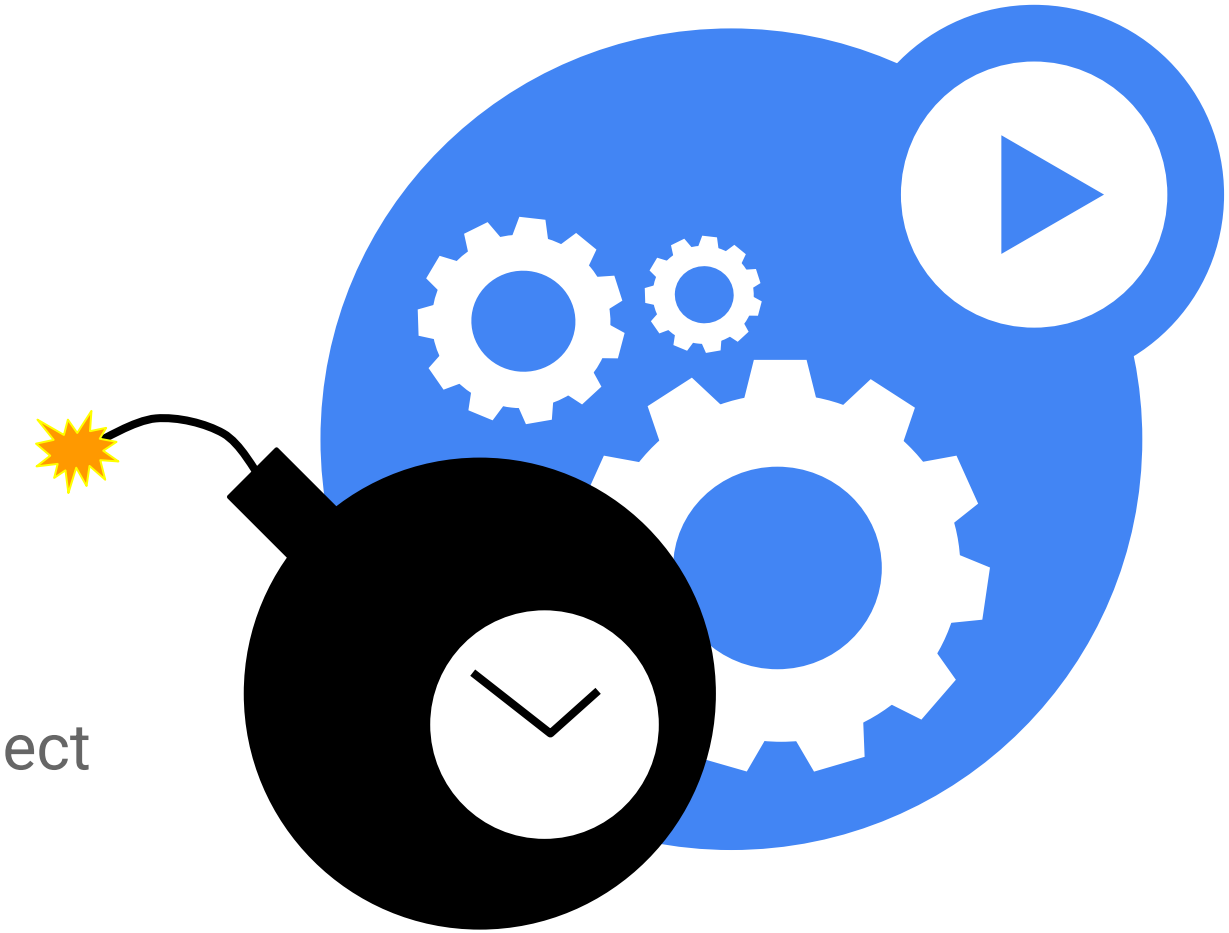
Goal: Give pods time to clean up

- 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



http://k8s-school.fr

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

Node Drain

Node Drain

Goal: Evacuate a node for maintenance

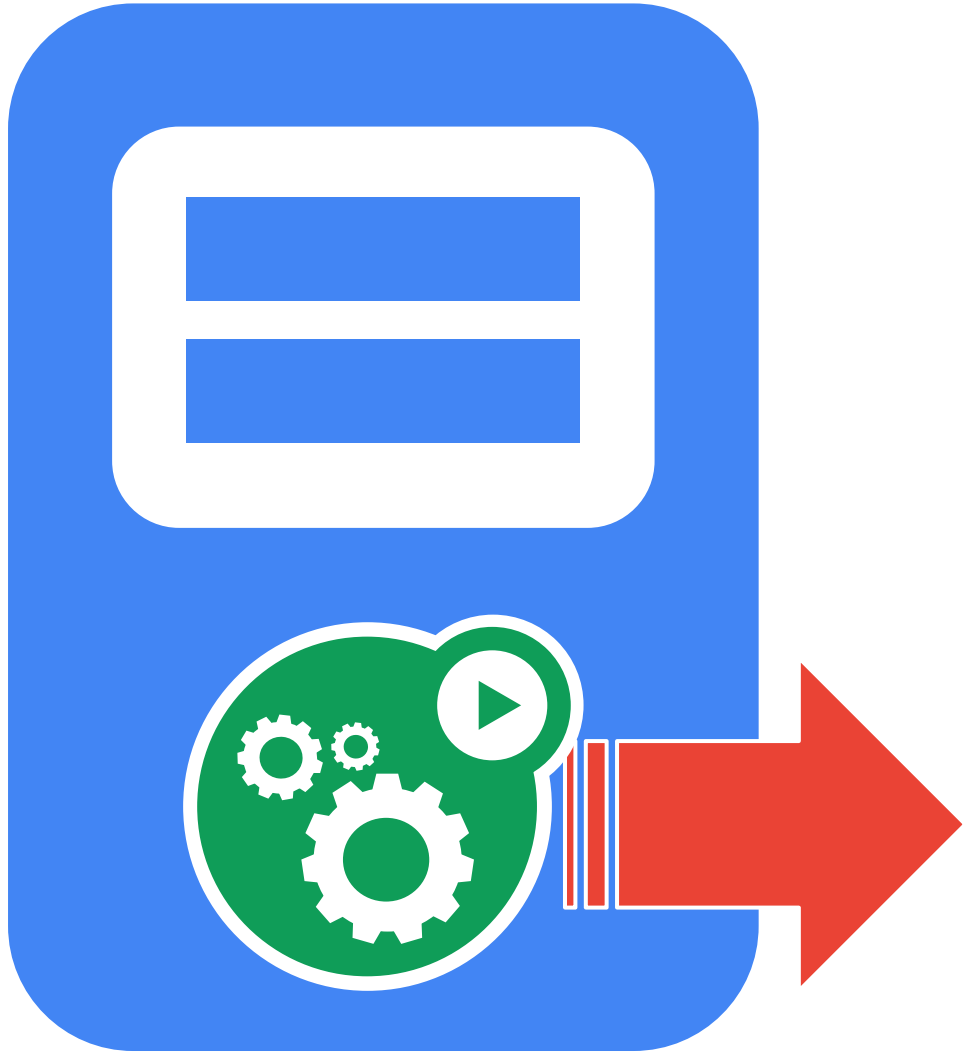
- e.g. kernel upgrades

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

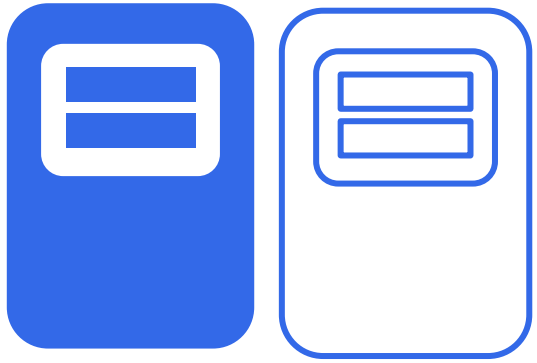
Icons

volume

container

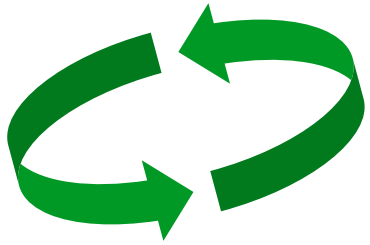
pod

labels

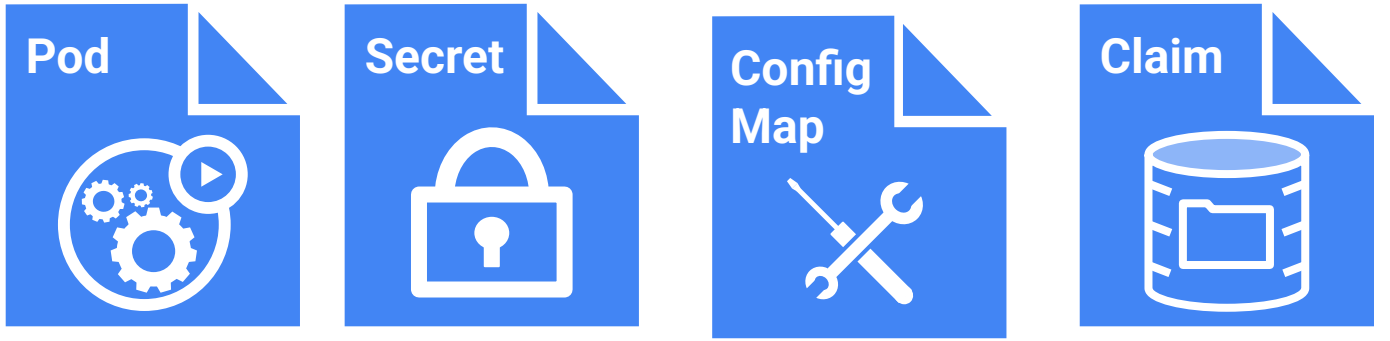


node

replication controller or
replica set

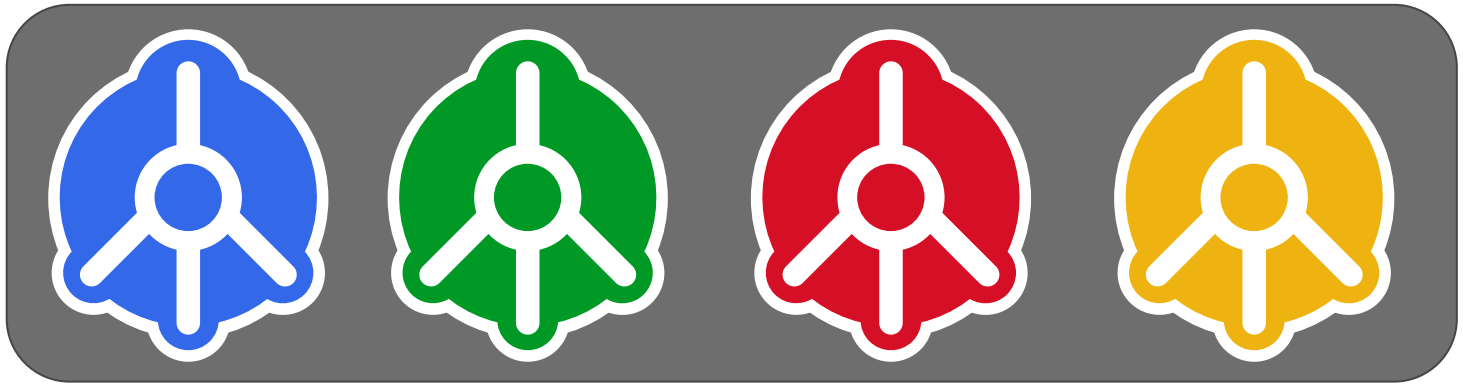


controller

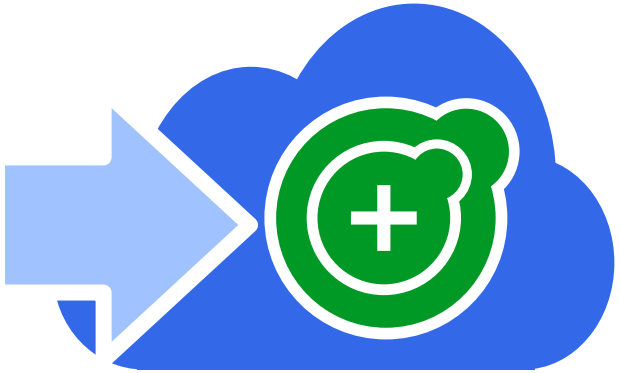


YAMLs

http://k8s-school.fr



service (new)



deployment

http://k8s-school.fr