WSO2Con 2015 USA San Francisco

Kubernetes: a platform for automating deployment, scaling, and operations

Brian Grant

Kubernetes: a platform for automating deployment, scaling, and operations

WSO2Con 2015

Brian Grant
@bgrant0607

What is Kubernetes?

Old way: install applications on host

Application and OS share filesystem

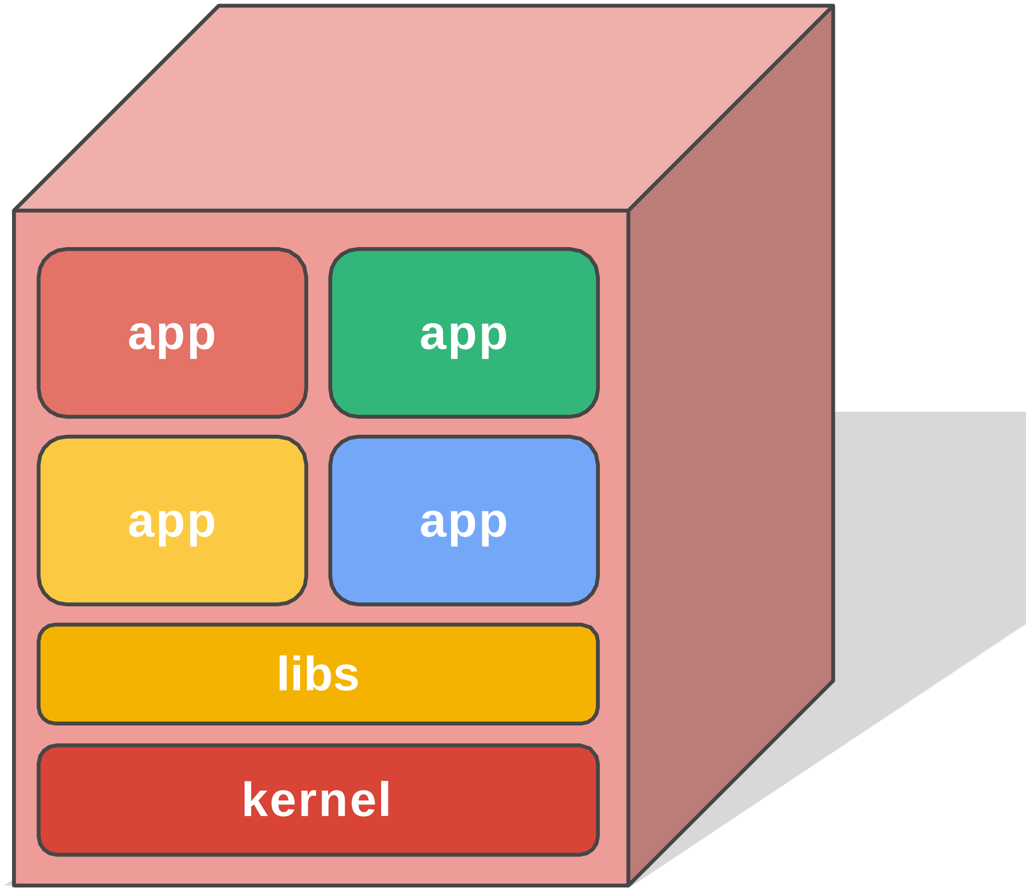
Use OS distribution package manager

Entangled with each other and with host

- Executables
- Configuration
- Shared libraries
- Process and lifecycle management

Immutable VM images provide predictable rollouts and rollbacks

- but are not portable and heavyweight



New way: deploy containers

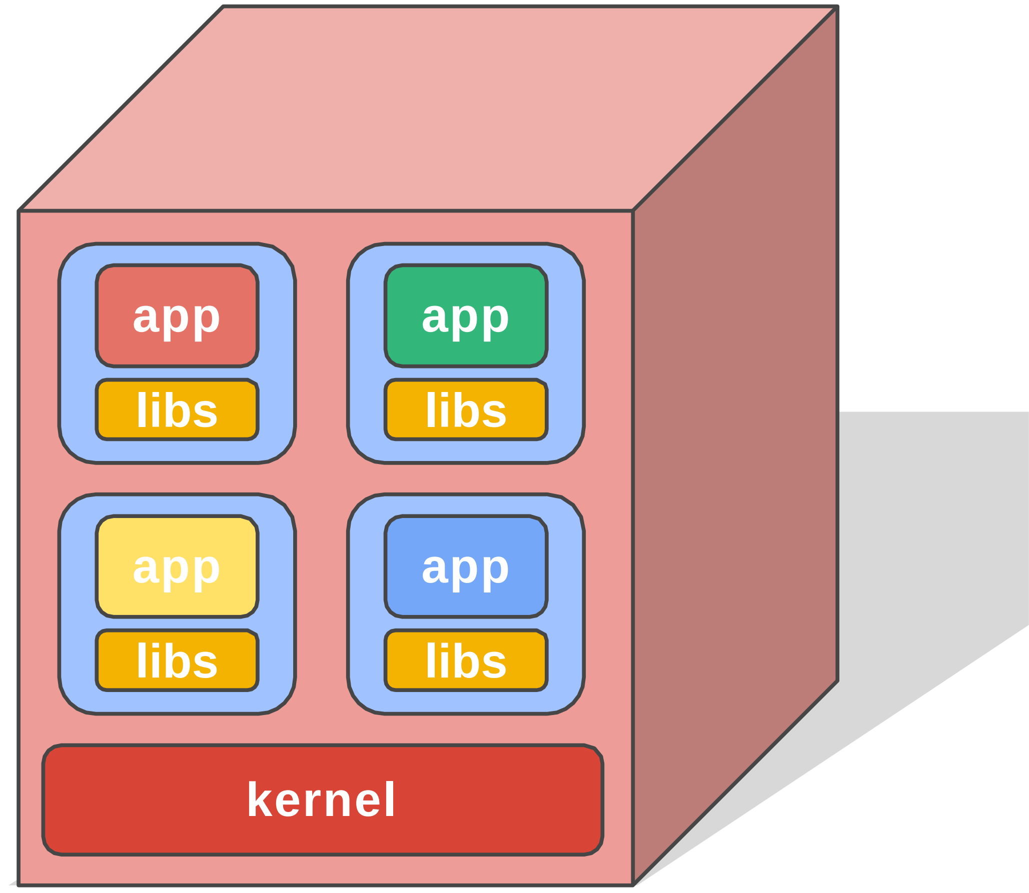
OS-level virtualization

Isolated, from each other and from the host

- filesystems
- processes
- resources

Small and fast ⇒ enables 1:1 app to image

- Unlocks benefits of microservices
- Decouple build (Dev) from deployment (Ops)
- Consistency from development to production
- Portable across OS distros and clouds
- Application-centric management



Need container-centric infrastructure

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

Want to automate orchestration for velocity & scale

Diverse workloads and use cases demand still more functionality

- Rolling updates and blue/green deployments
- Application secret and configuration distribution
- Continuous integration and deployment
- Workflows
- Batch processing
- Scheduled execution
- Application-specific orchestration

...

A composable, extensible **Platform** is needed

Kubernetes

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

- Infrastructure for containers
- Schedules, runs, and manages containers on virtual and physical machines
- Platform for automating deployment, scaling, and operations
- Inspired and informed by Google's experiences and internal systems
- **100% Open source**, written in Go

Deployment

```
$ kubectl run my-nginx--image=nginx
replicationcontroller "my-nginx" created

$ kubectl get po
NAME              READY     STATUS     RESTARTS     AGE
my-nginx-         1/1       Running    0            1m
wepbv
```

Scaling

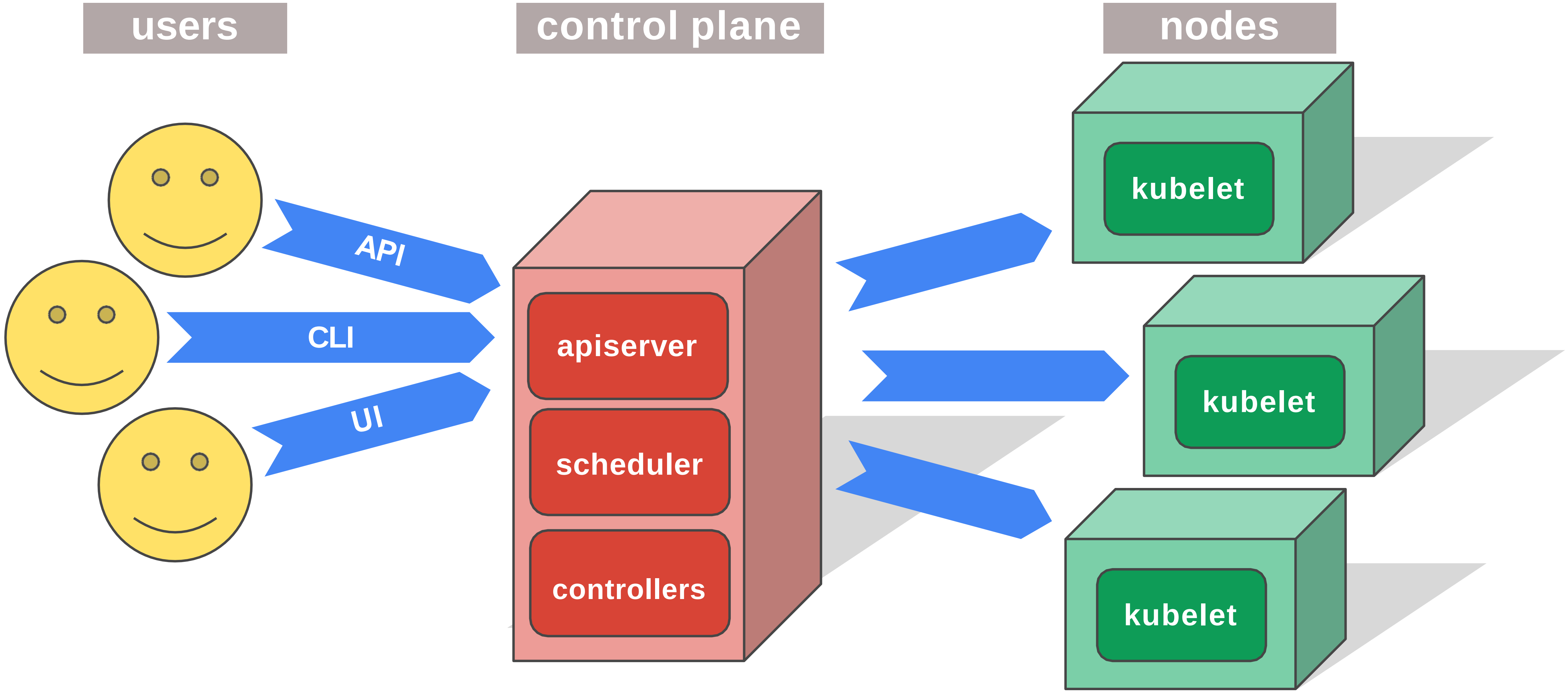
```
$ kubectl scale rc my-nginx --replicas=2
replicationcontroller "my-nginx" scaled

$ kubectl get po
NAME              READY     STATUS     RESTARTS    AGE
my-nginx-         1/1       Running    0           1m
wepbv
my-nginx-yrf3u    1/1       Running    0           20s
```

Shutdown

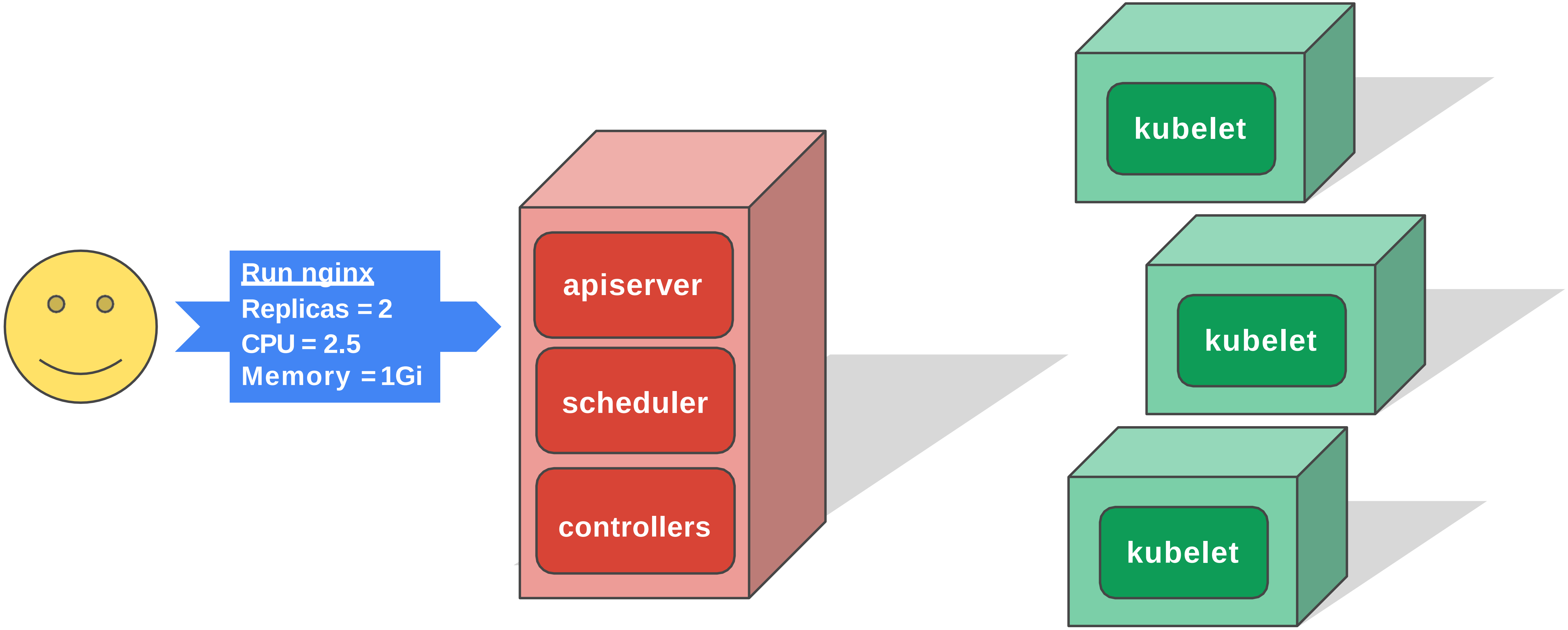
```
$ kubectl delete rc my-nginx
replicationcontroller "my-nginx" deleted

$ kubectl get po
NAME              READY   STATUS        RESTARTS   AGE
my-nginx-         0/1     Terminating   0          4m
wepbv
my-nginx-yrf3u    0/1     Terminating   0          3m

$ kubectl get po


$
```

Kubernetes architecture

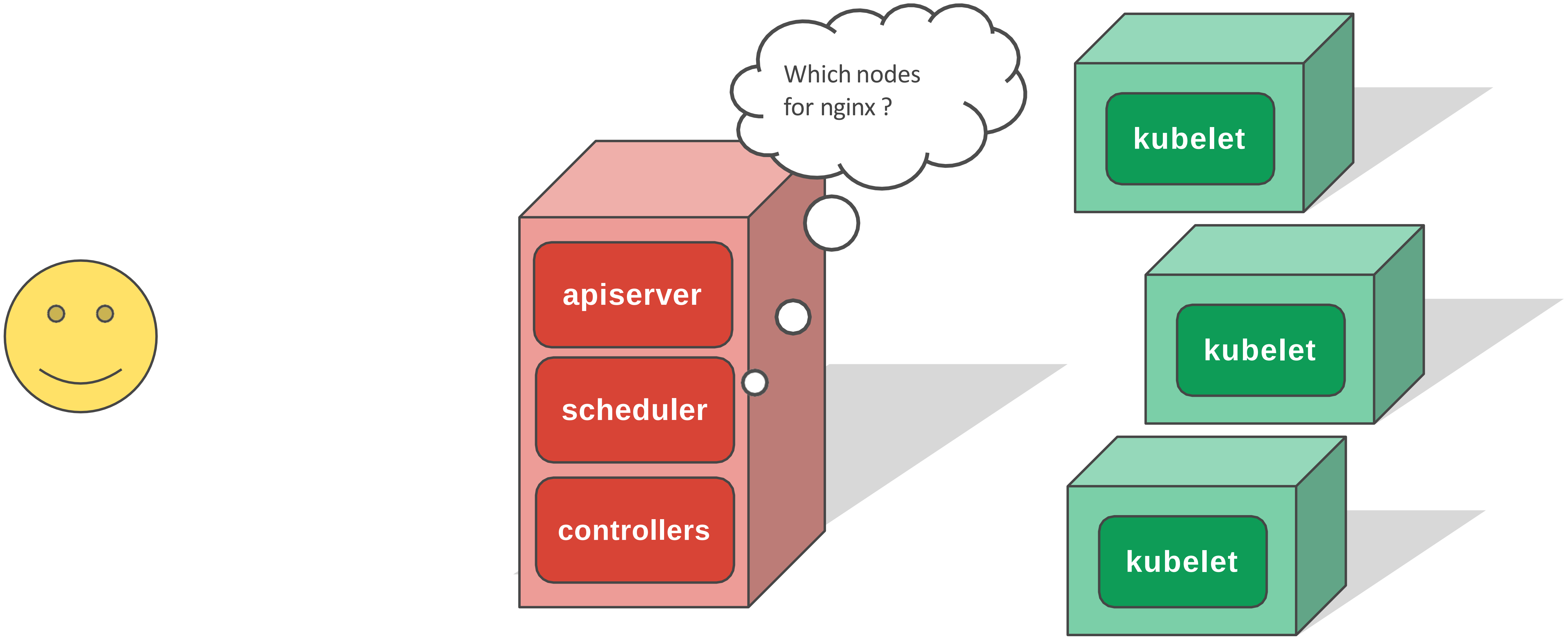
Kubernetes architecture



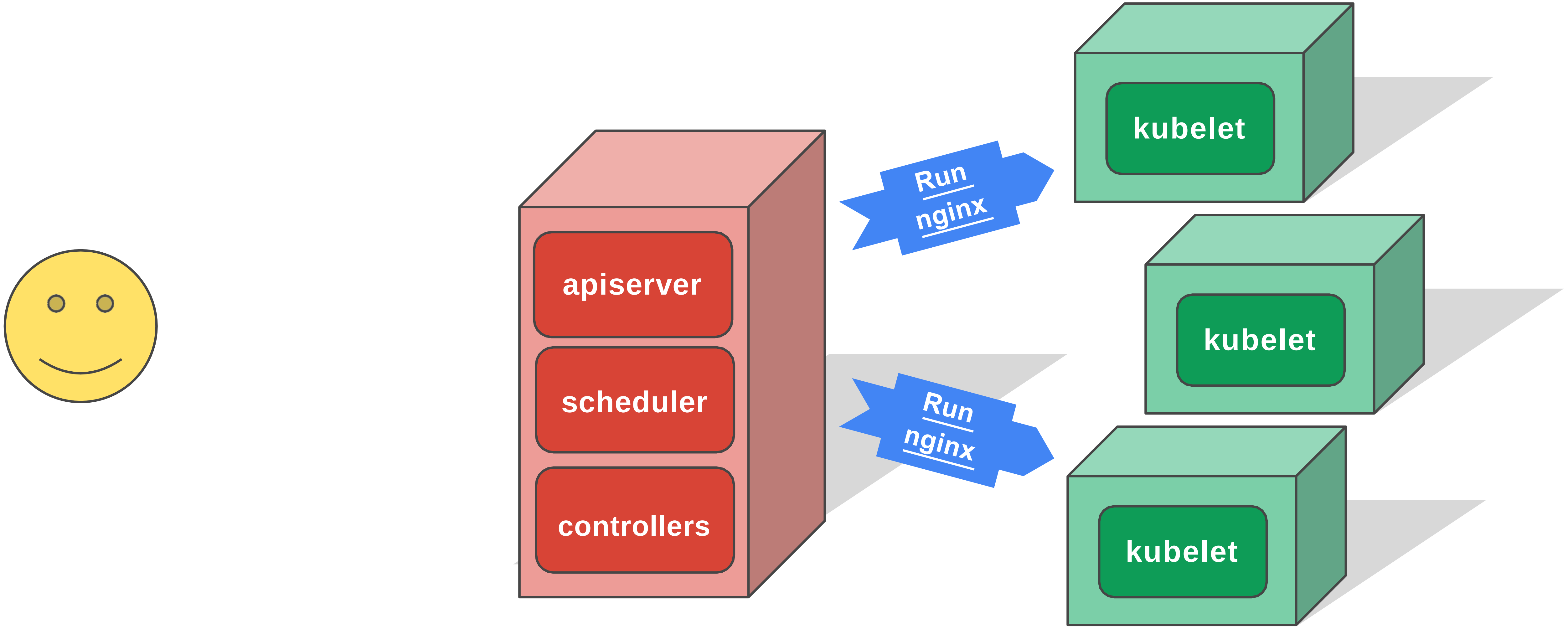
Post desired state (aka spec) via API



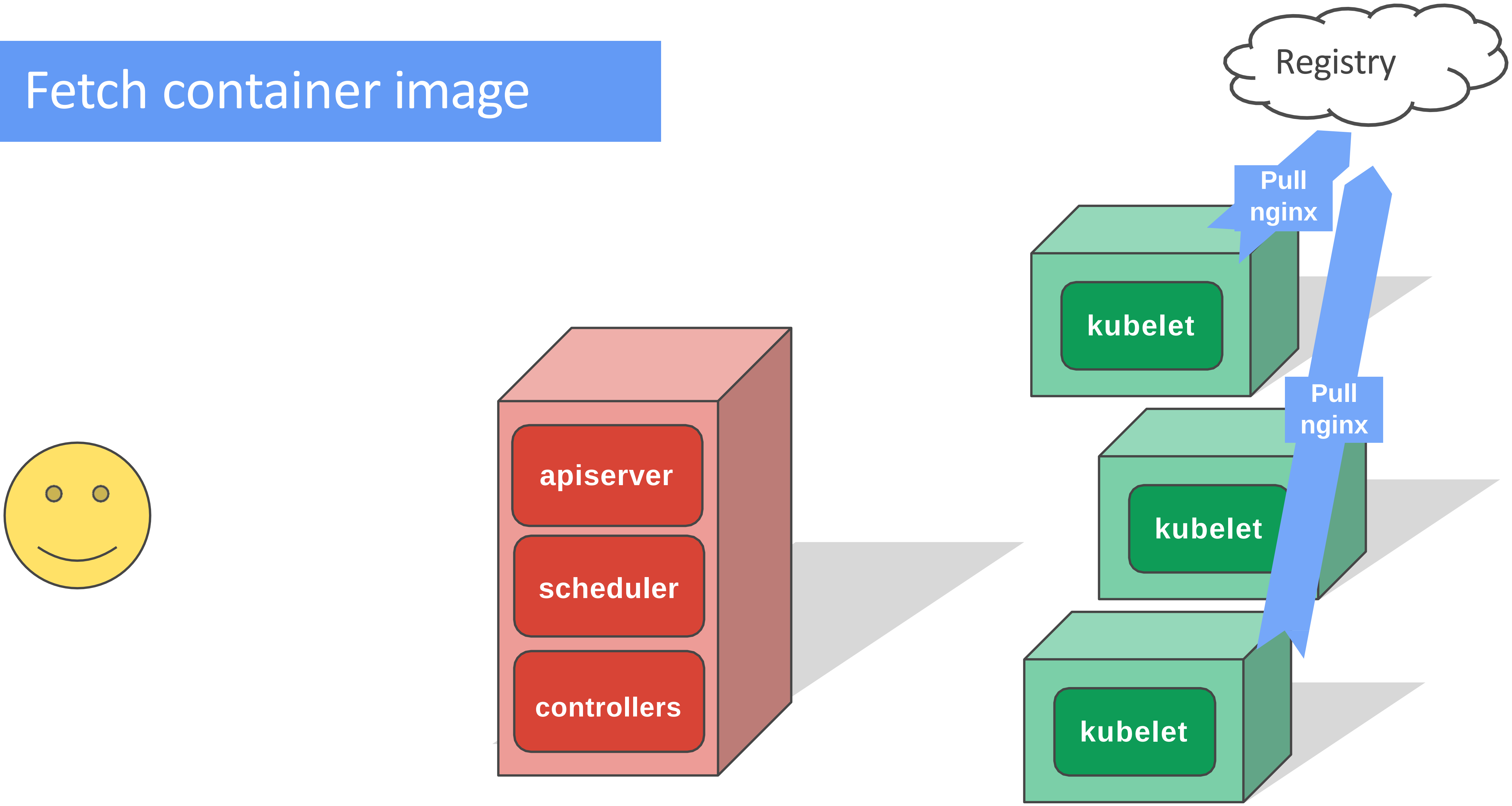
Placement (aka scheduling)



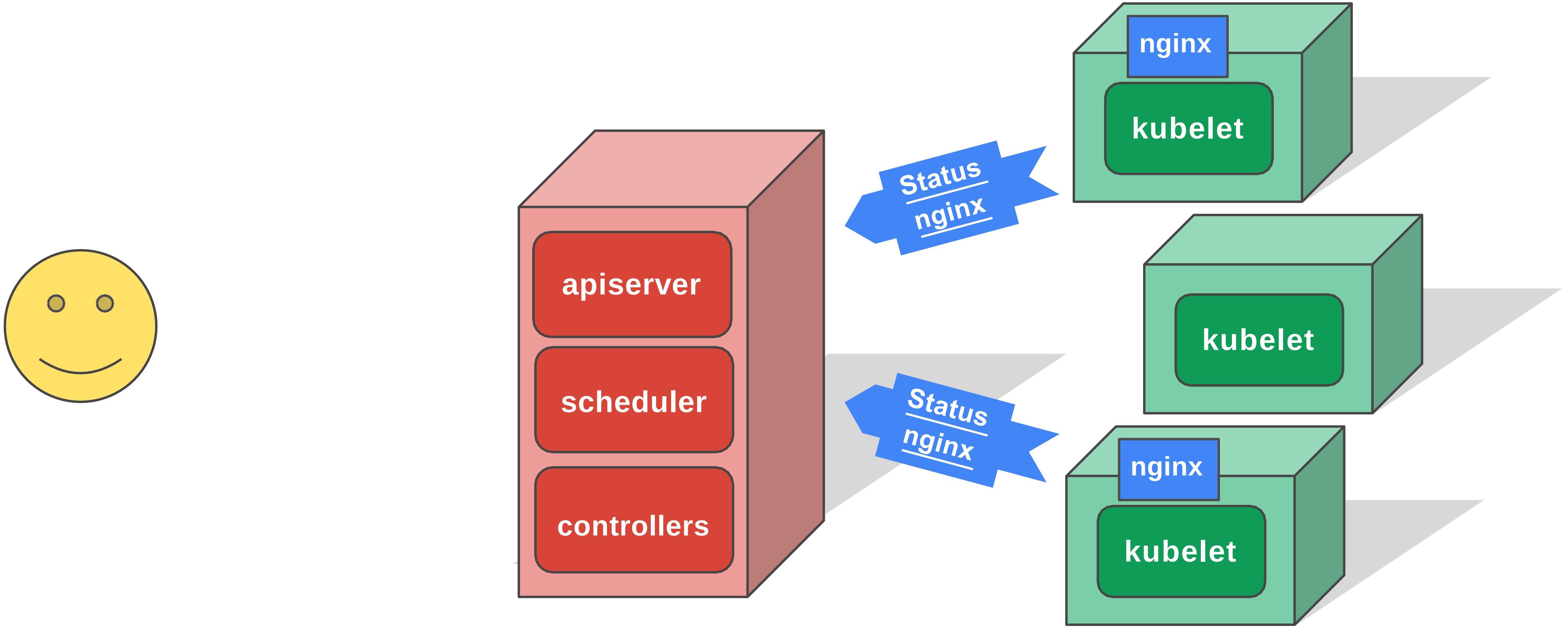
Assignment (aka binding)



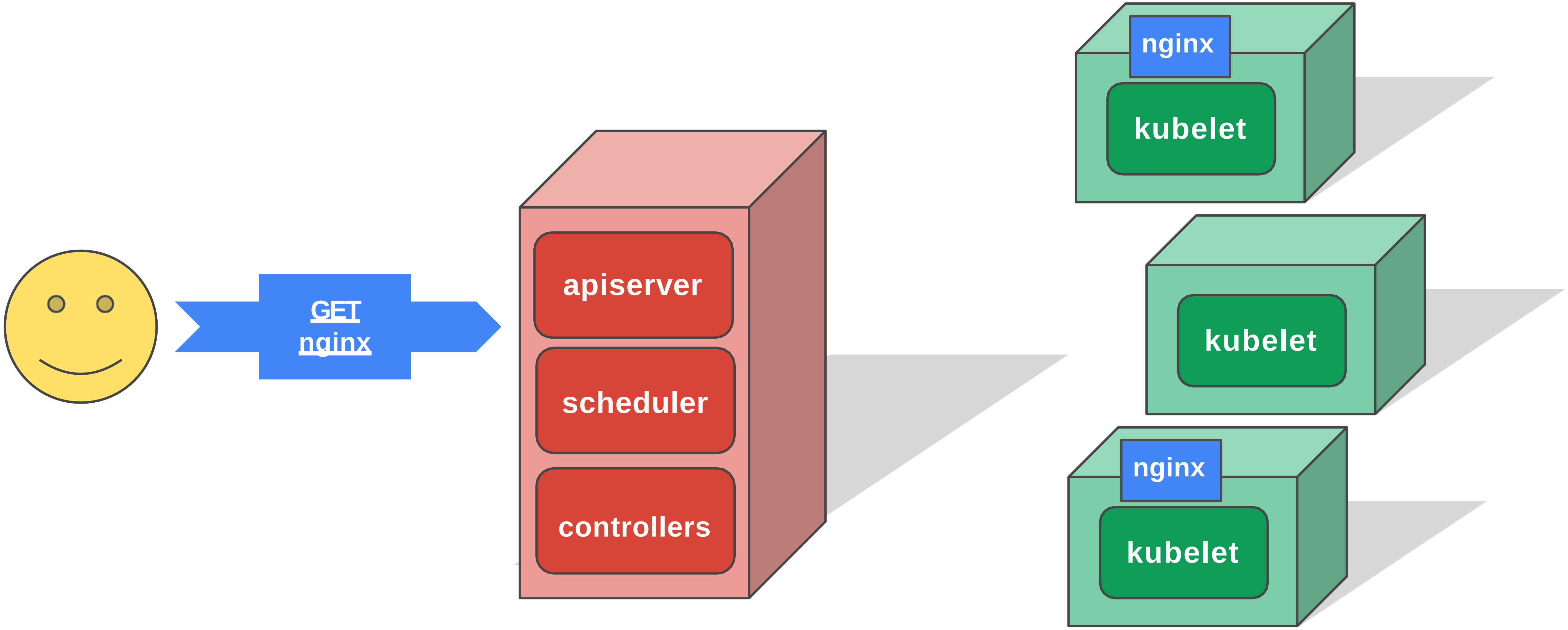
Fetch container image



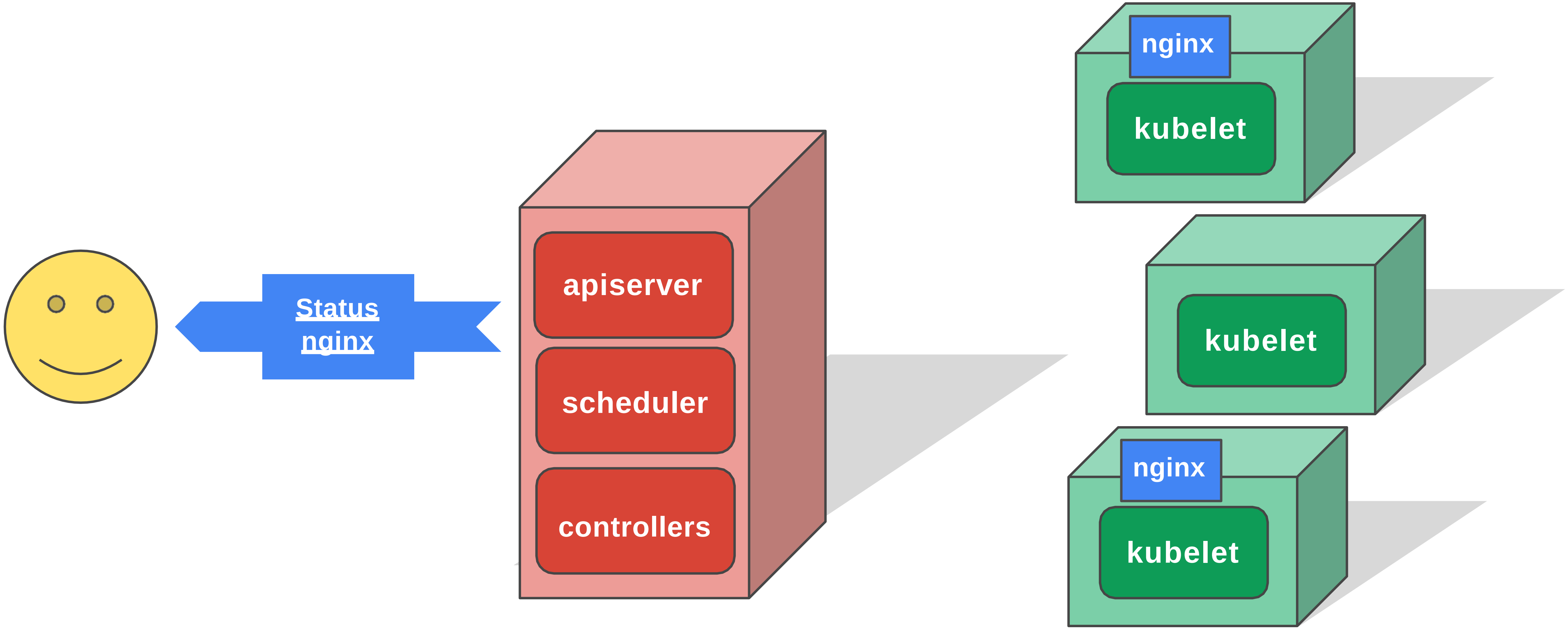
Execution and lifecycle management



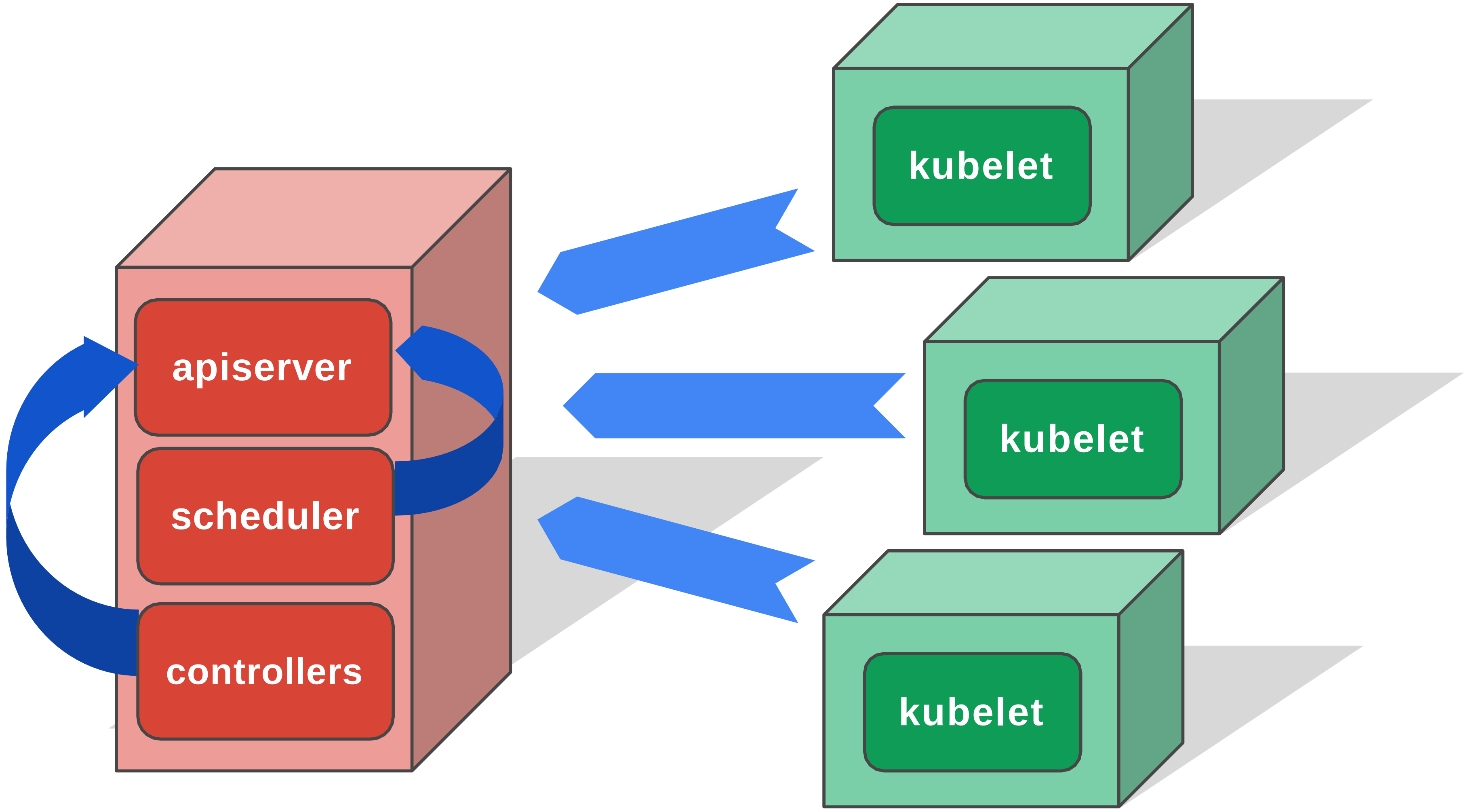
Get current status via API



Get current status via API



Kubernetes uses the same APIs as users



Modularity

Modularity facilitates

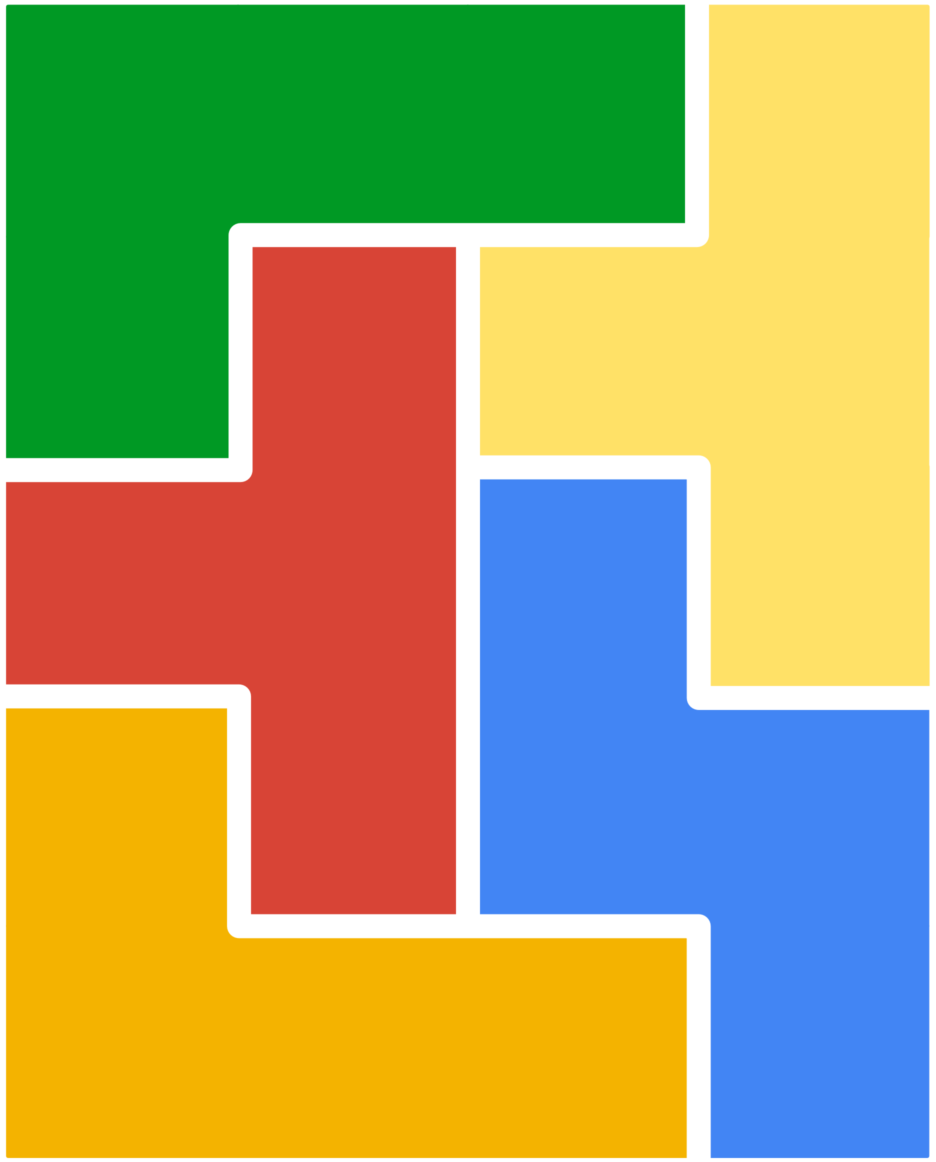
- composability
- extensibility

APIs - **no shortcuts** or back doors

- ensures extensions are on equal footing

Example: Scheduler

Example: Controllers



Control loops

Drive **current state → desired state**

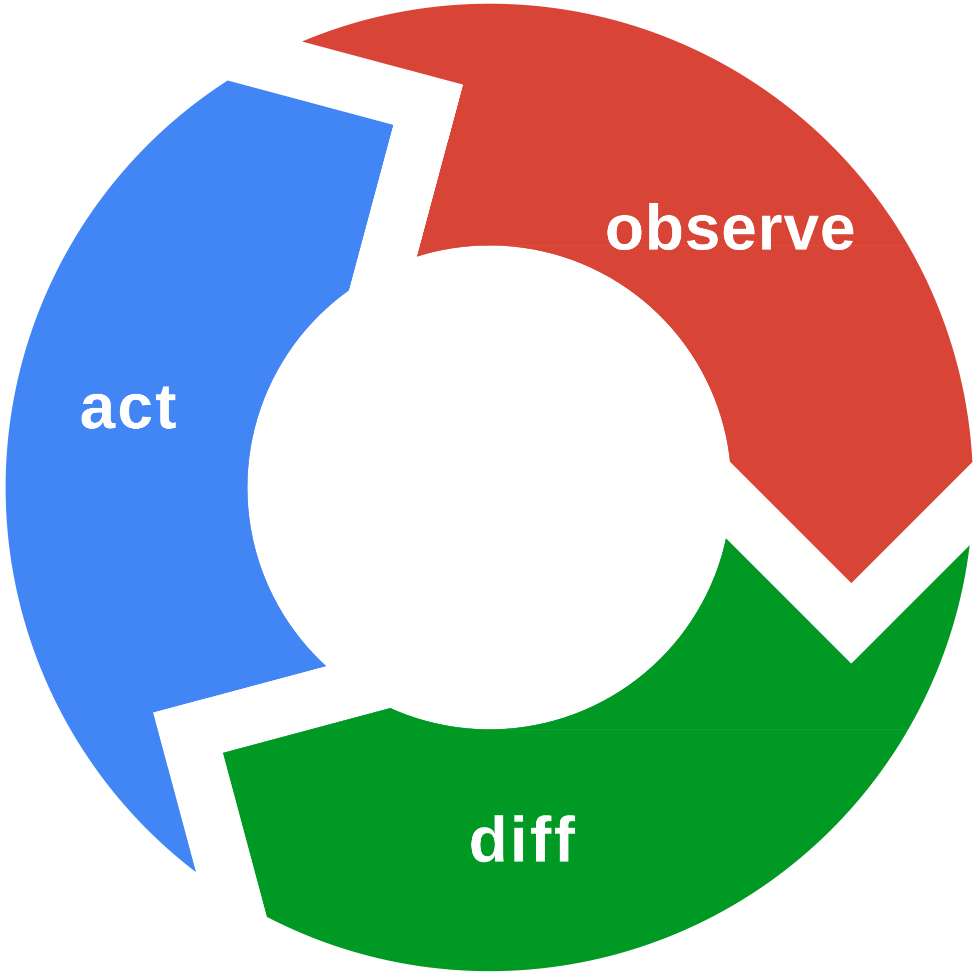
Observed state is truth

Act independently

- *choreography* rather than *orchestration*

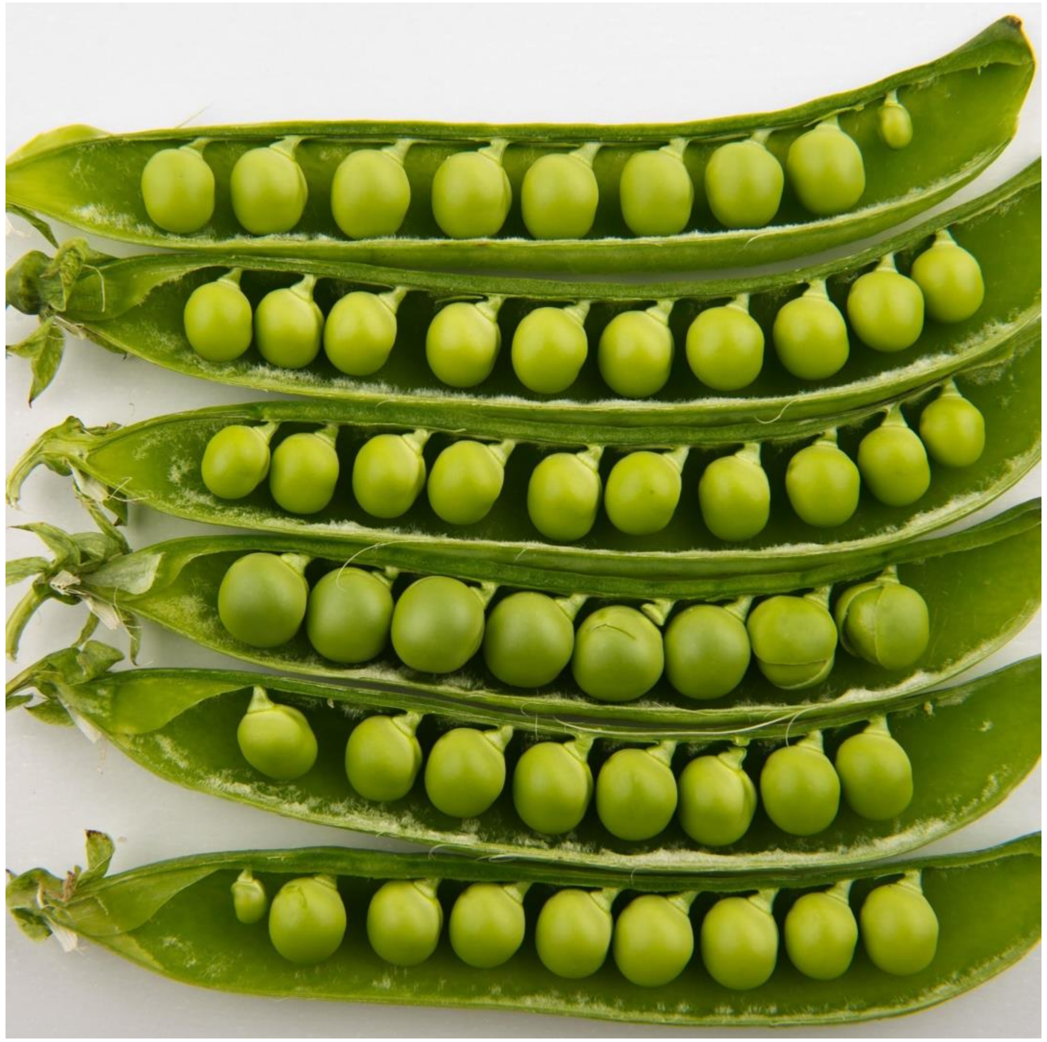
Recurring pattern in the system

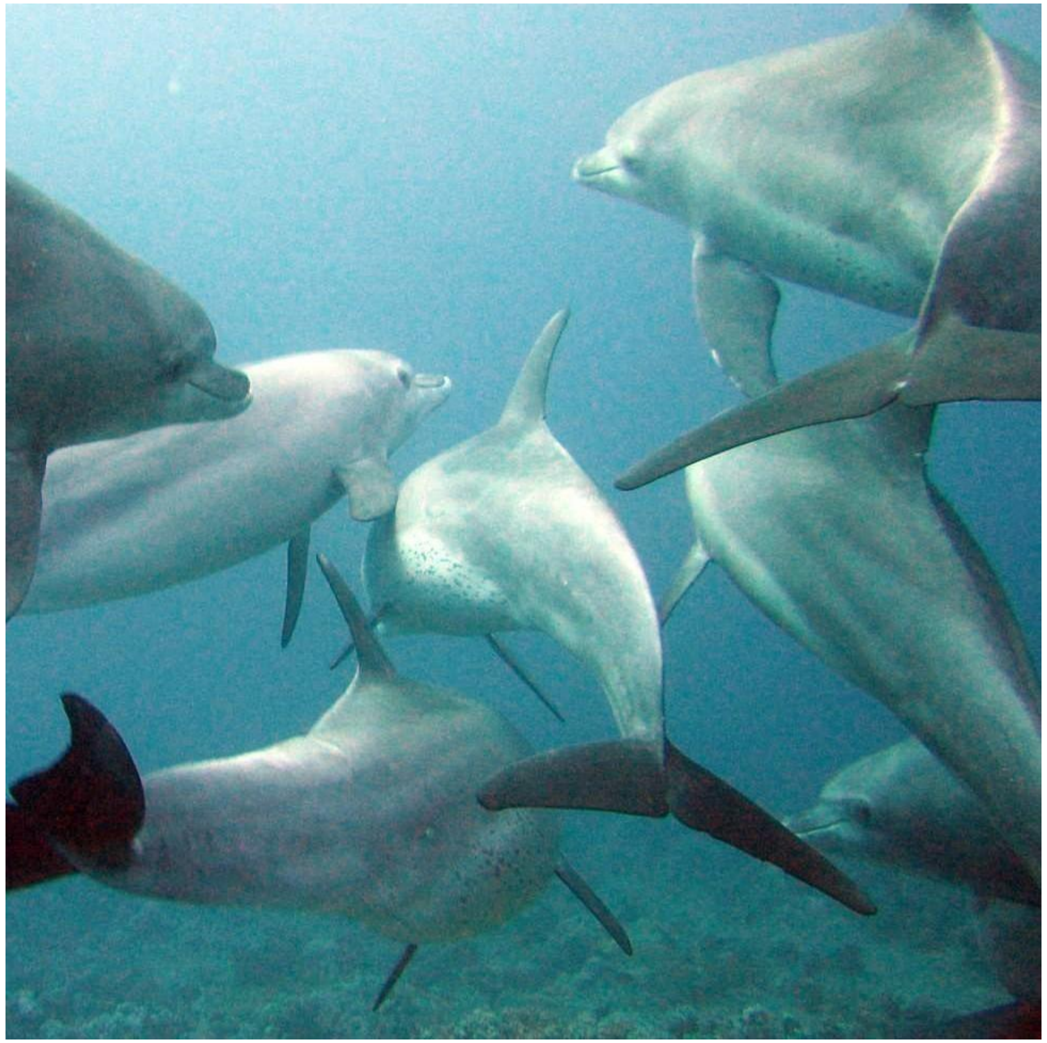
Example: Scheduler
Example: Controllers



Core primitives

Pods





Pods

Small group of containers & volumes

Tightly coupled

- the atom of replication & placement

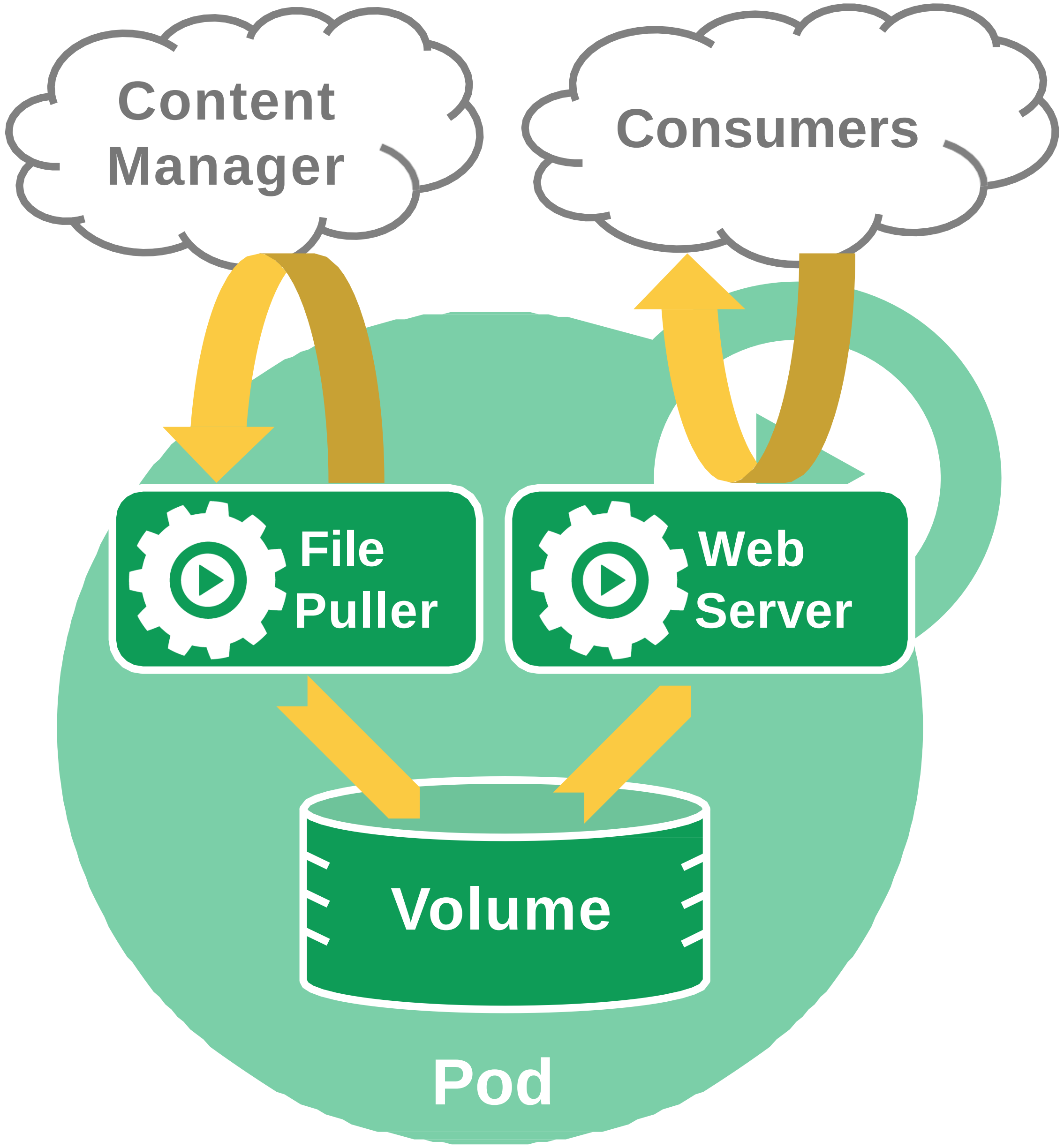
"Logical" host for containers

- each pod gets an IP address
- share data: localhost, volumes, IPC, etc.

Facilitates composite applications

- mix and match components, languages, etc.
- preserves 1:1 app to image

Example: data puller & web server



Volumes

Storage automatically attached to pod

- Local scratch directories created on demand
- Cloud block storage
  - GCE Persistent Disk
  - AWS Elastic Block Storage
- Cluster storage
  - File: NFS, Gluster, Ceph
  - Block: iSCSI, Cinder, Ceph
- Special volumes
  - Git repository
  - Secret

Critical building block for higher-level automation

Secrets

How to grant a pod access to a secured *something?*

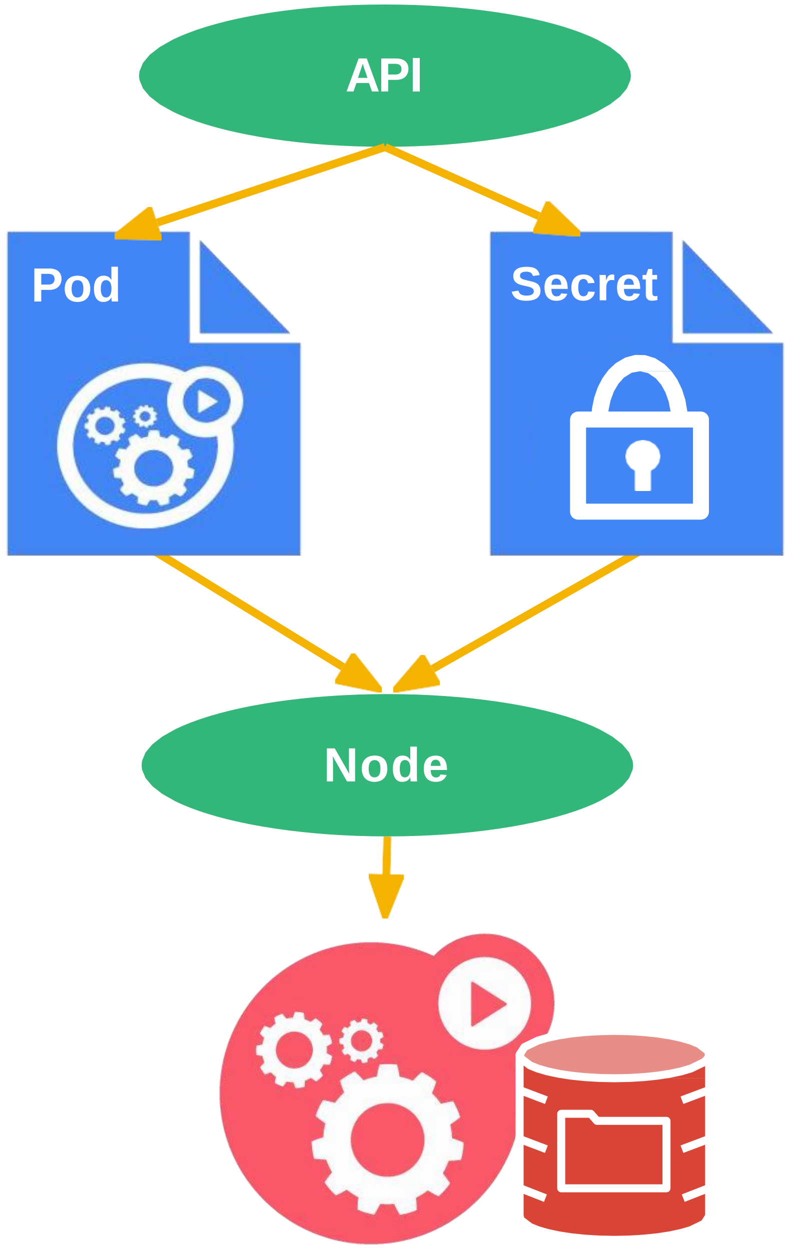
- **secrets**: credentials, tokens, passwords, ...
- don't put them in the container image!

12-factor says should come from the environment

Inject them as "virtual volumes" into Pods

- not baked into images nor pod configs
- kept in memory - never touches disk
- not coupled to non-portable metadata API

Manage secrets via the Kubernetes API



Labels

User-provided key-value attributes

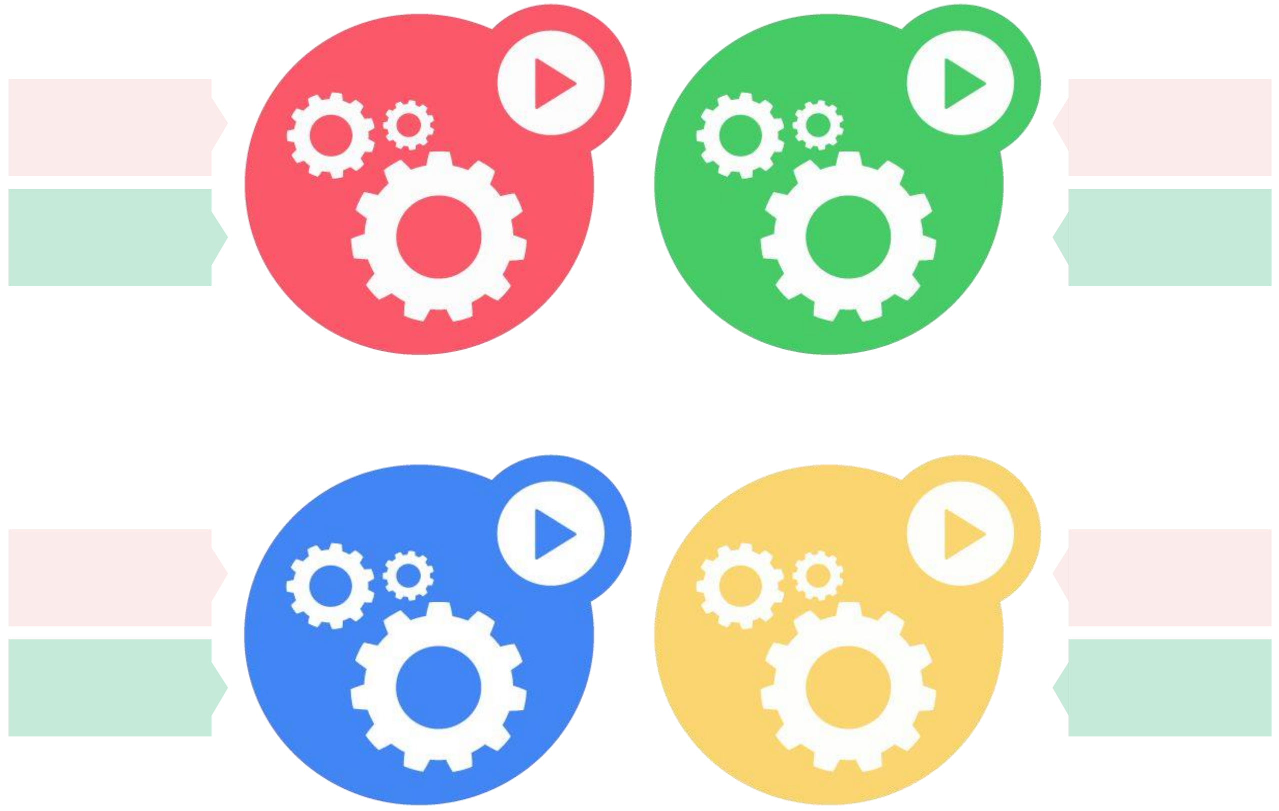
Attached to any API object

Generally represent **identity**

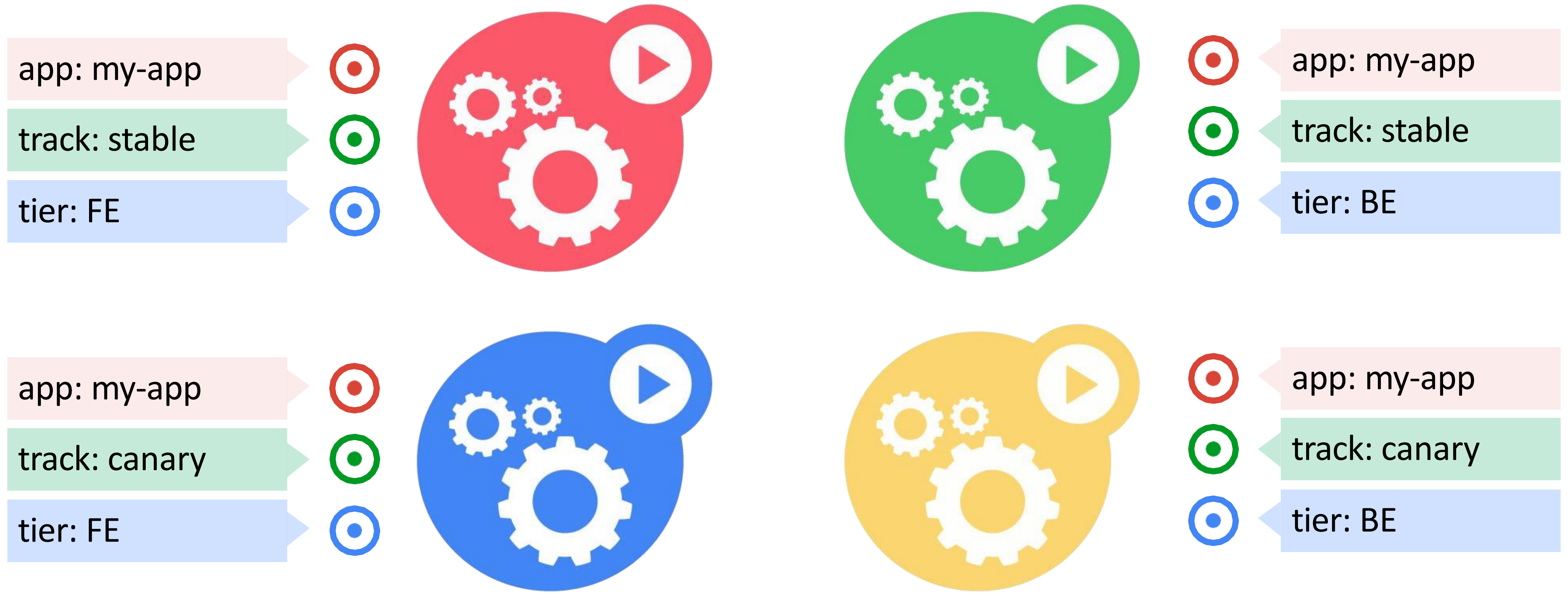
Queryable by **selectors**

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

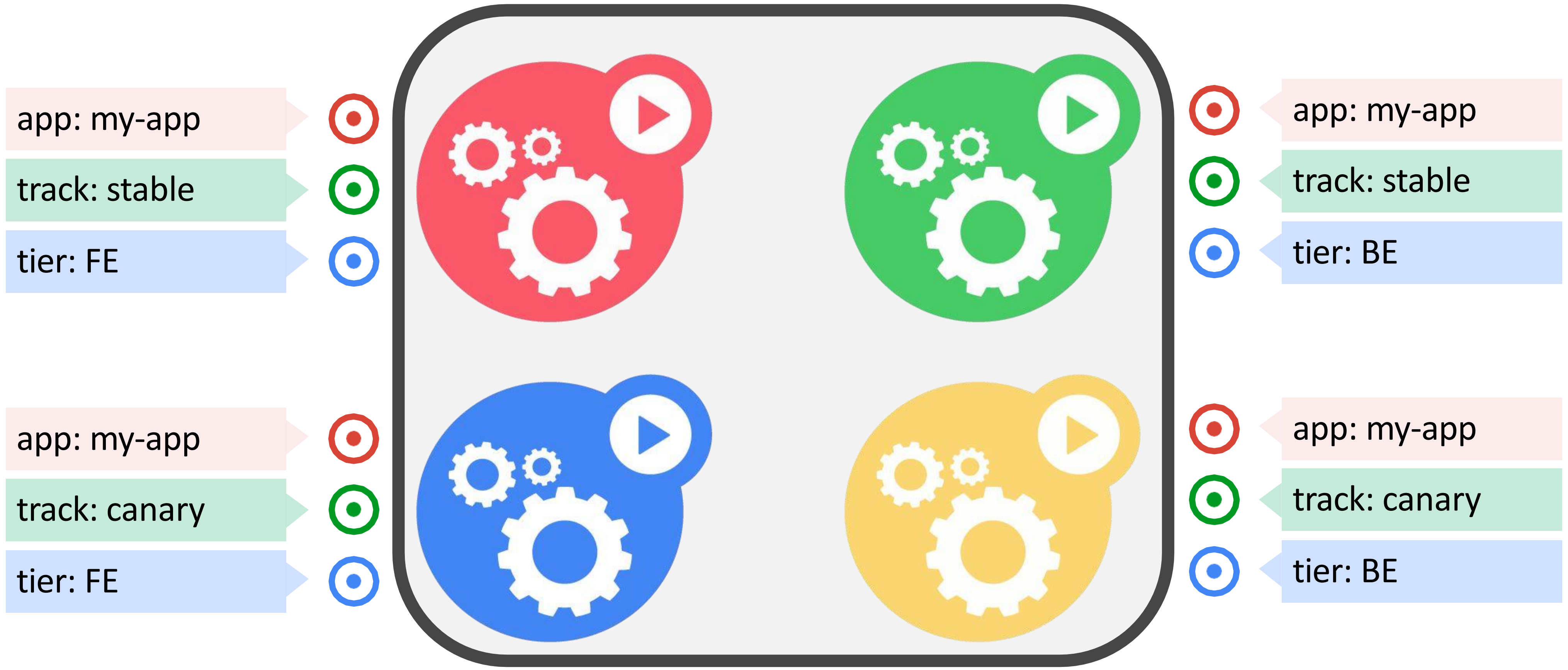
The **only** grouping mechanism



Selectors

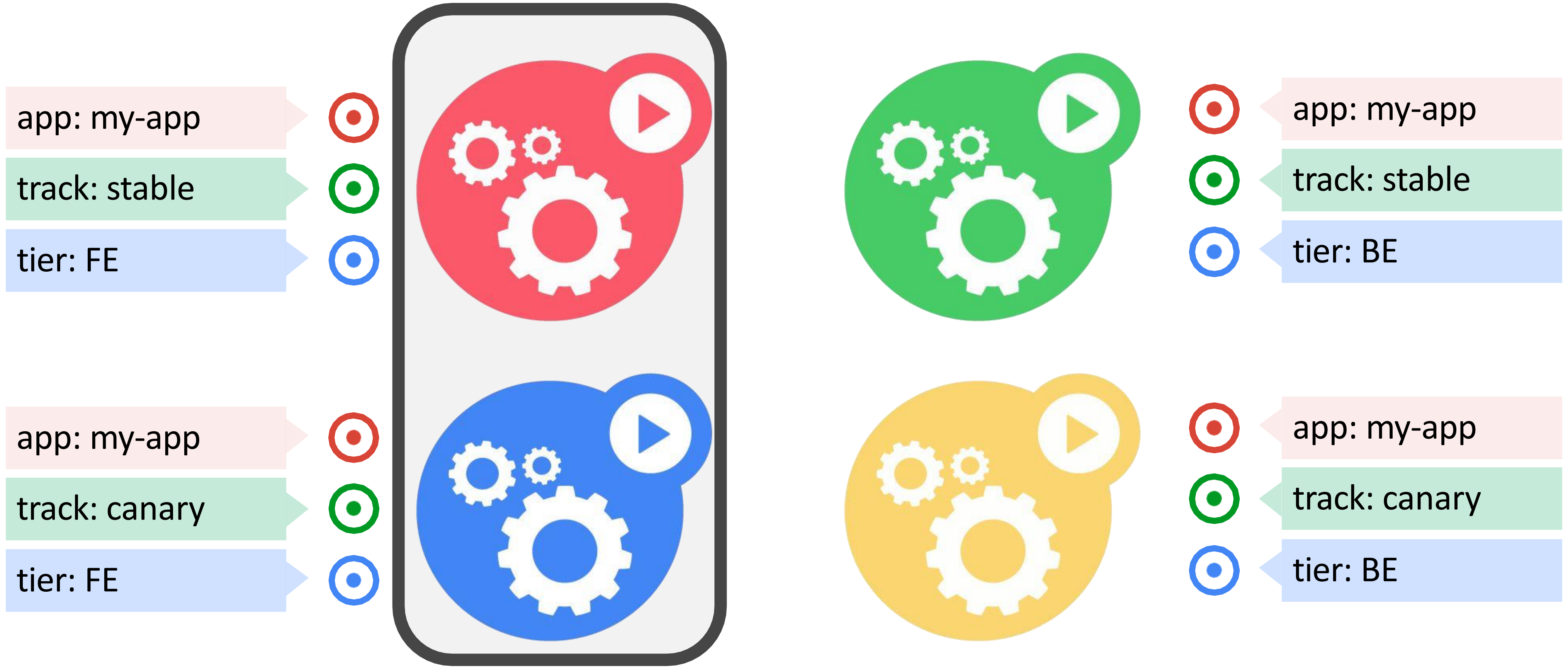

app: my-app
track: stable
tier: FE
app: my-app
track: stable
tier: BE
app: my-app
track: canary
tier: FE
app: my-app
track: canary
tier: BE

Selectors



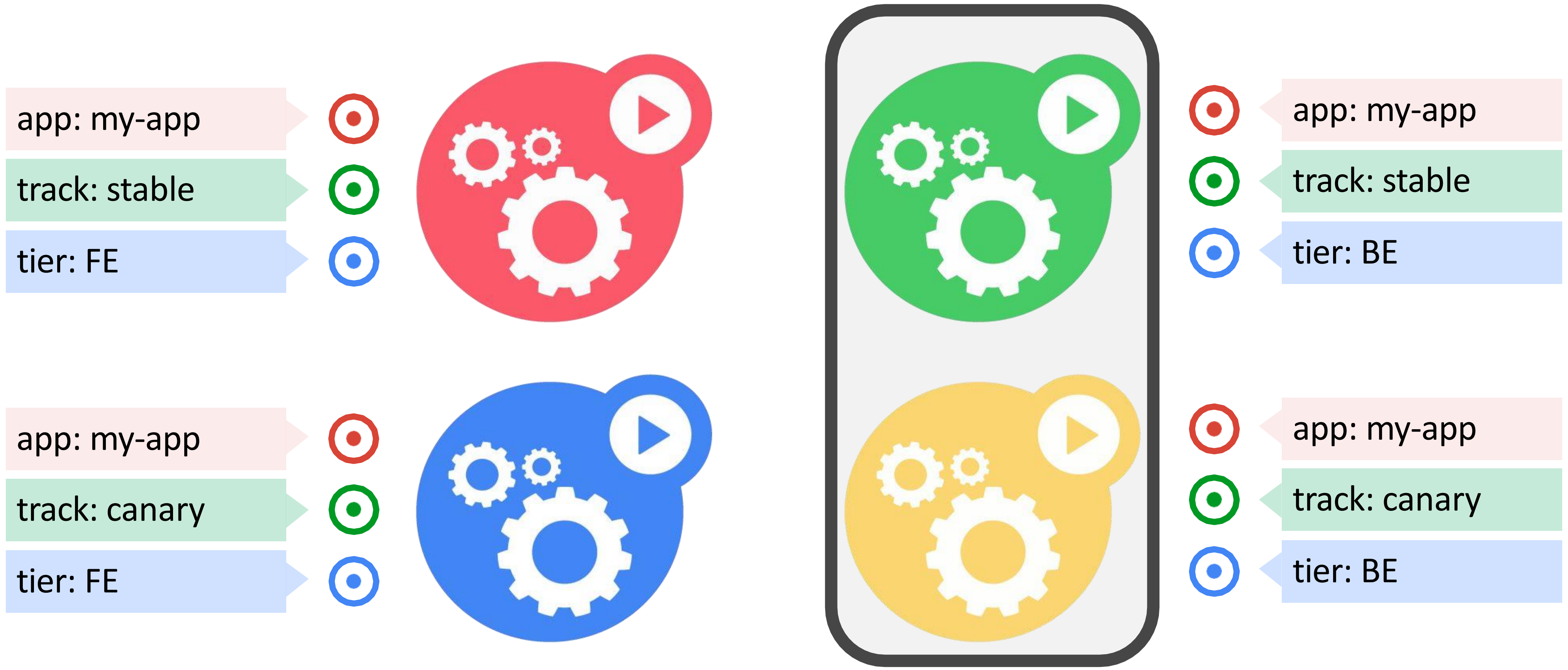
app = my-app

Selectors



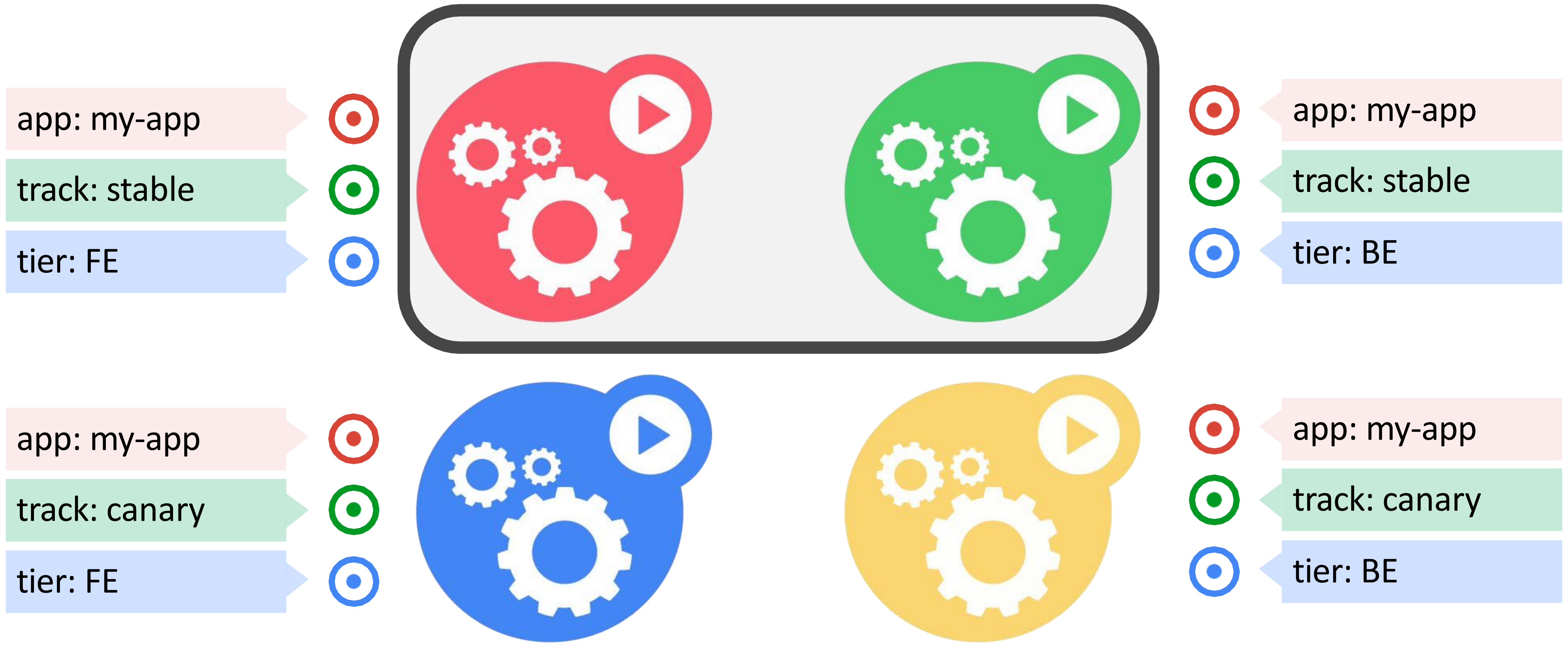
app = my-app, tier = FE

Selectors



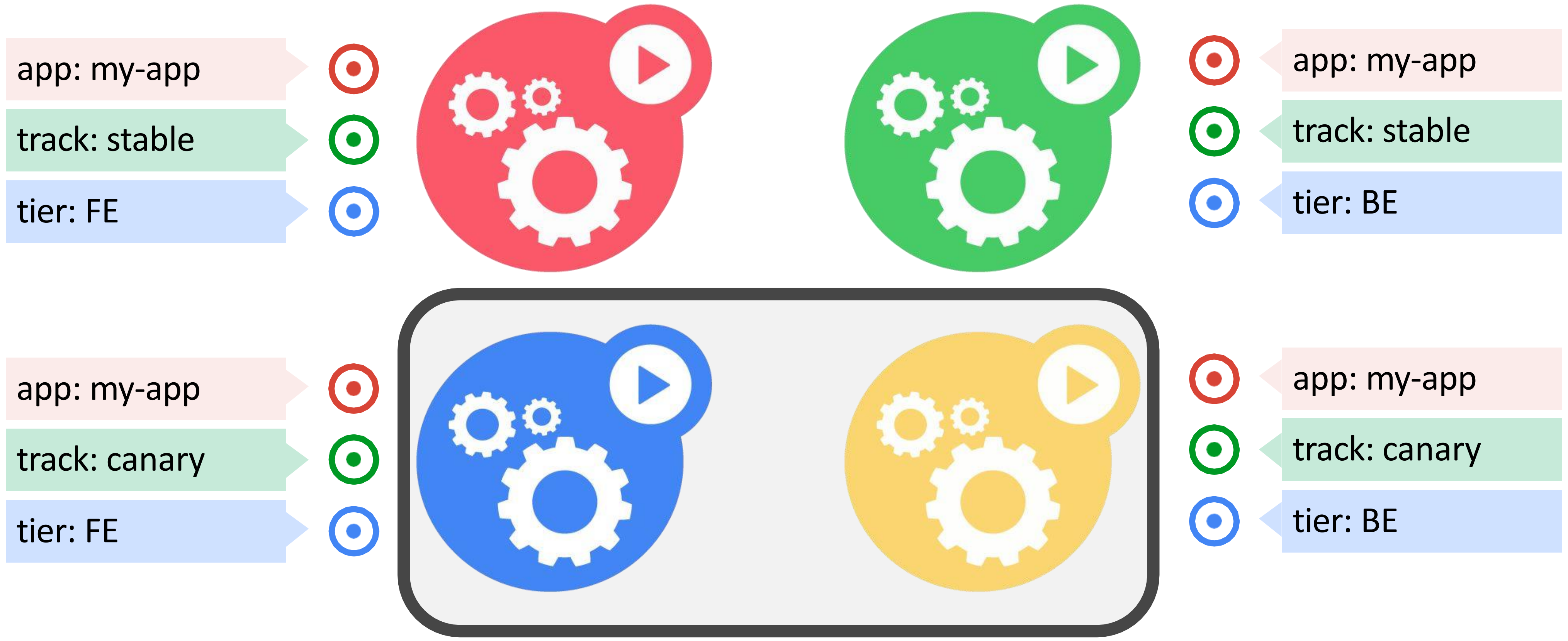
app = my-app, tier = BE

Selectors



app = my-app, track =stable

Selectors



app = my-app, track = canary

Running Microservices

ReplicationControllers

Ensures N copies of a Pod

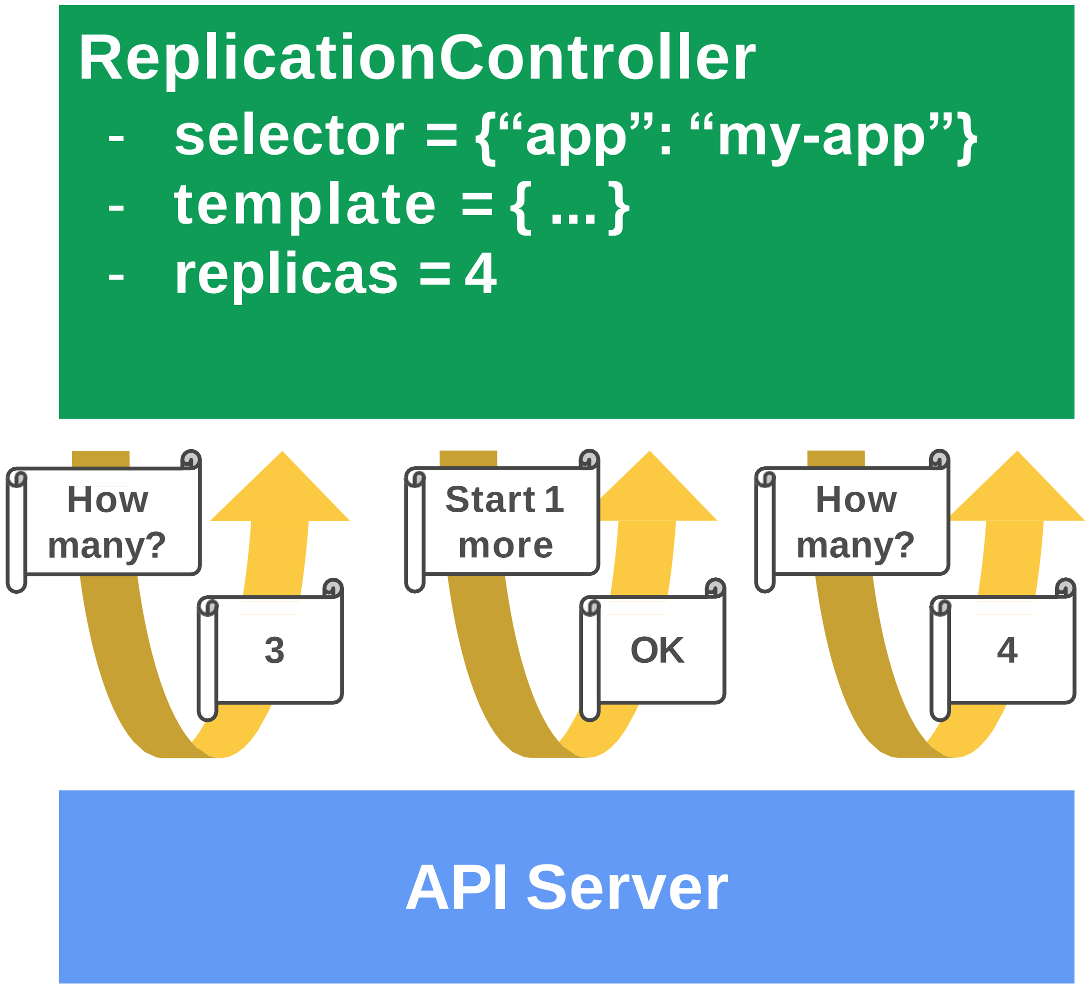
- if too few, start new ones
- if too many, kill some
- grouped by a label selector

Explicit specification of desired scale

- client doesn't just create N copies
- enables self-healing
- facilitates auto-scaling

An example of a controller

- calls public APIs



Services

A group of pods that **work together**

- grouped by a label selector

Publishes how to access the service

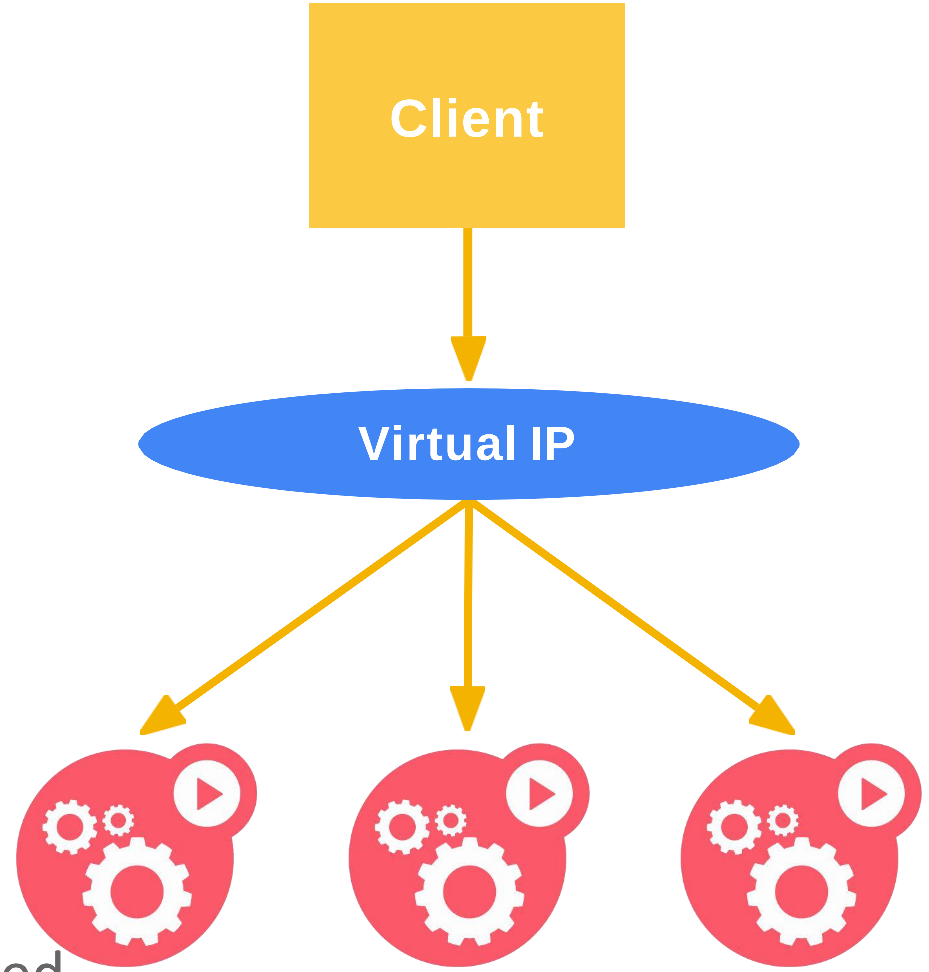
- DNS name
- DNS SRV records for ports (well known ports work, too)
- Kubernetes Endpoints API

Defines access policy

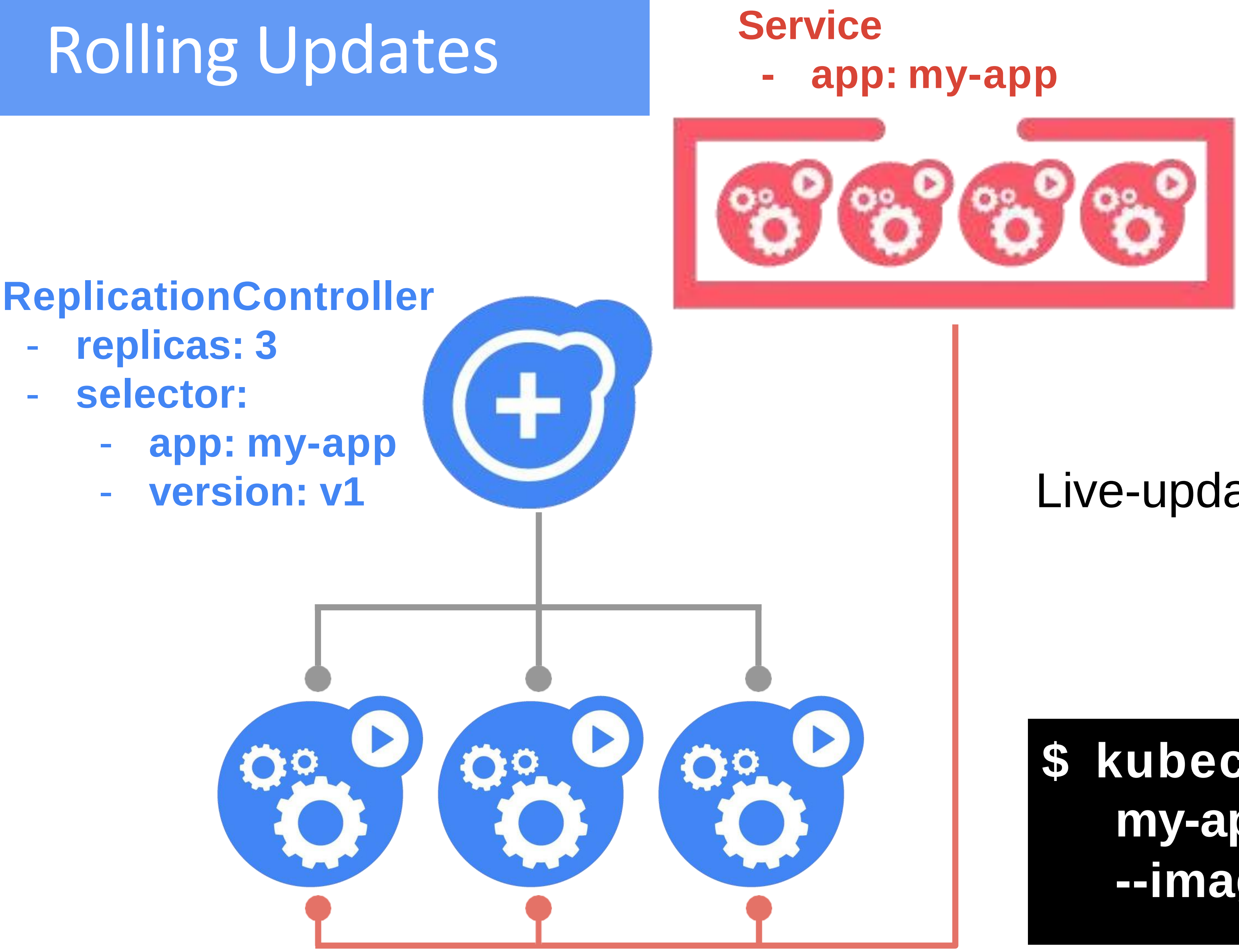
- Load-balanced: name maps to stable **virtual IP**
- “Headless”: name maps to set of pod IPs

Hides complexity - ideal for non-native apps Decoupled

from Pods and ReplicationControllers



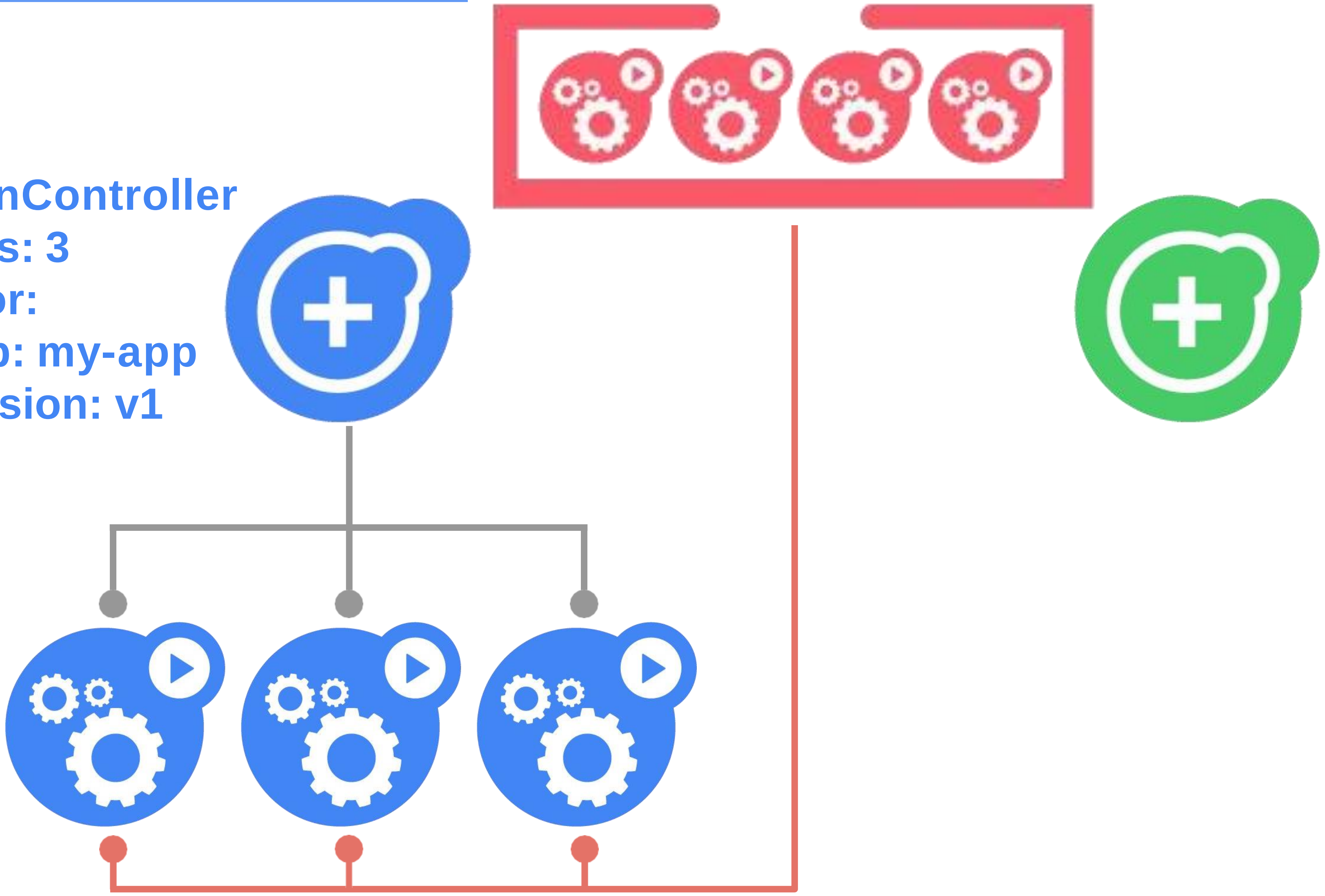
Rolling Updates



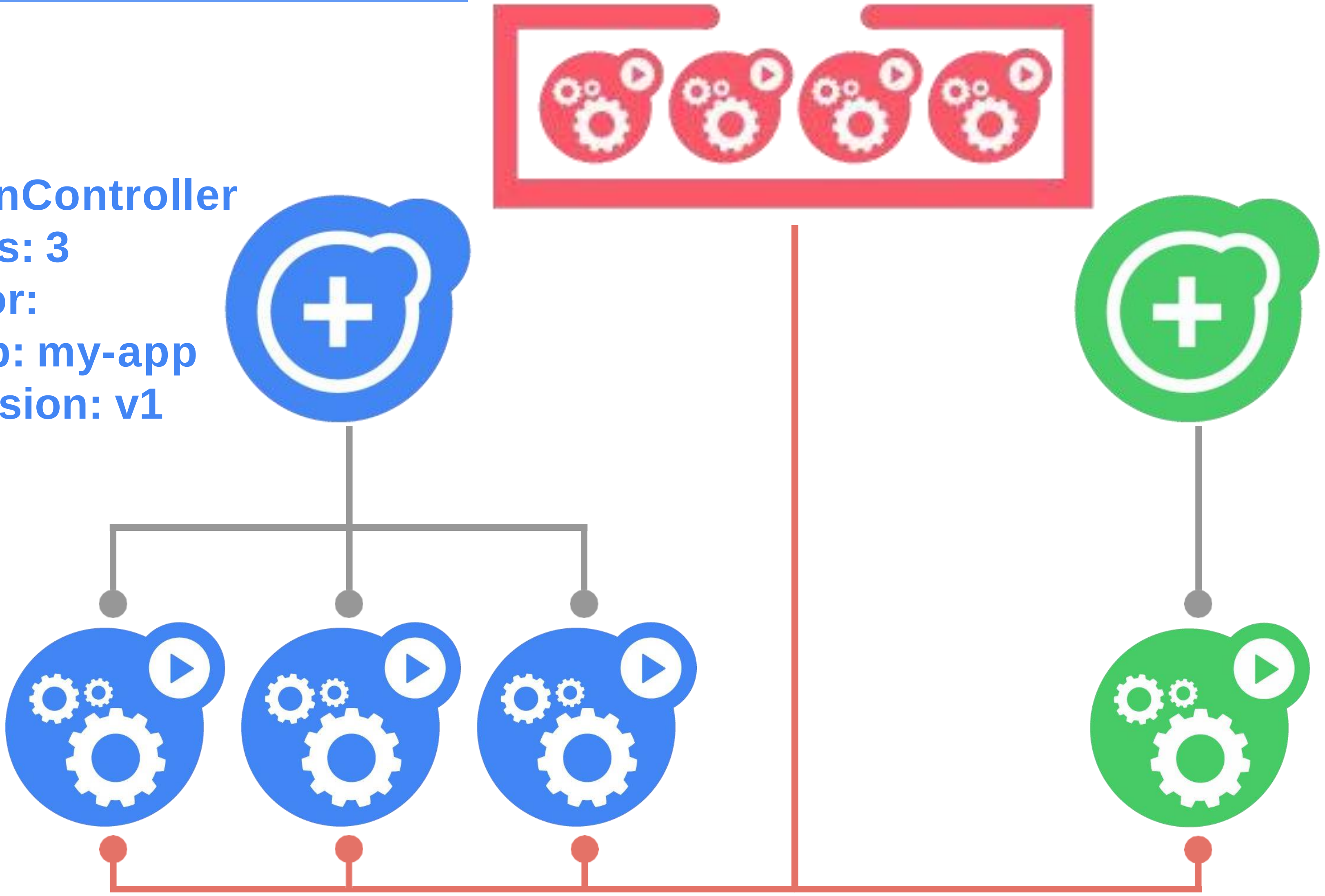
Live-update an application

```
$ kubectl rolling-update \
    my-app-v1 my-app-v2 \
    --image=image:v2
```

Rolling Updates



Rolling Updates



Service
- app: my-app

ReplicationController
- replicas: 3
- selector:
  - app: my-app
  - version: v1

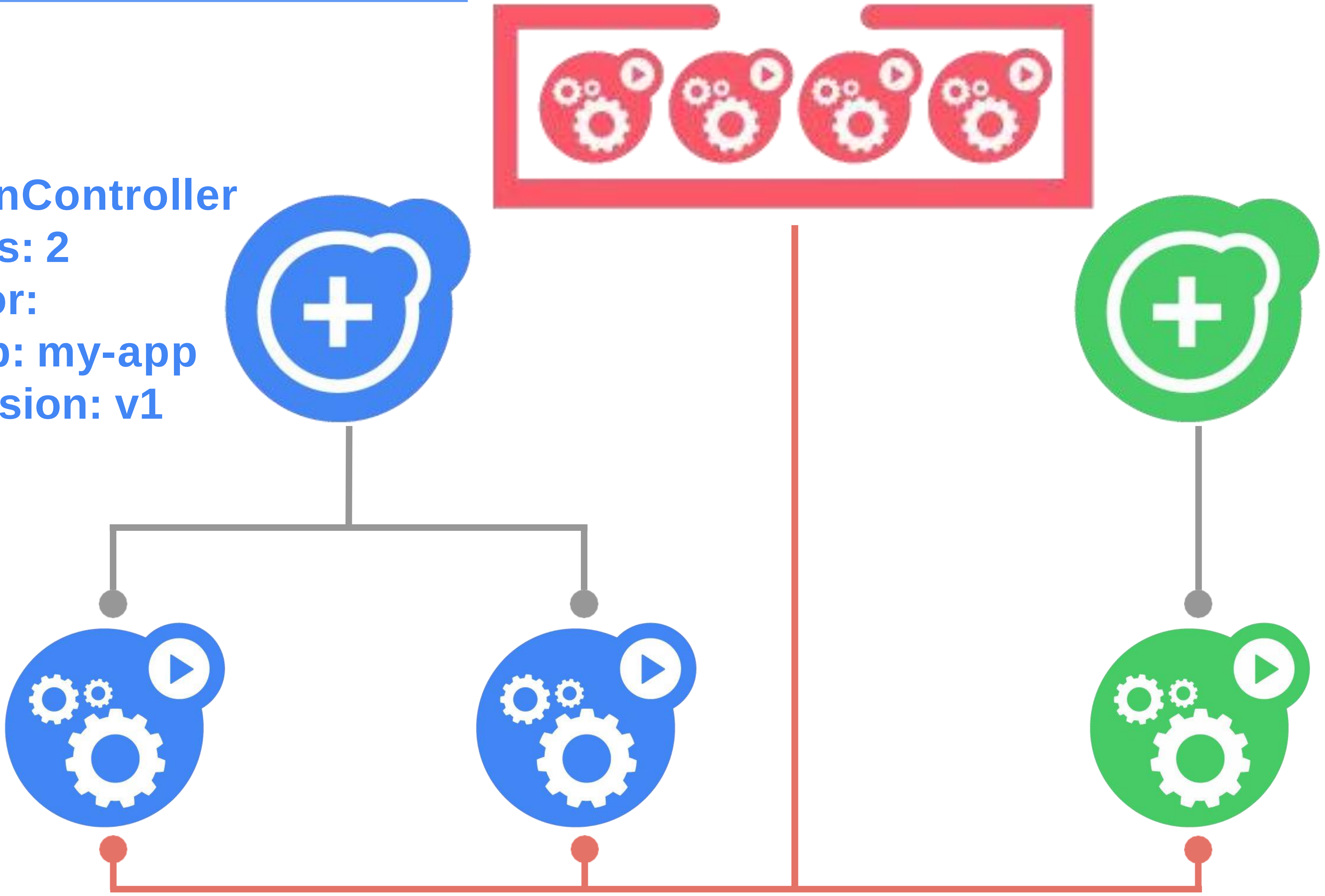
ReplicationController
- replicas: 1
- selector:
  - app: my-app
  - version: v2

Rolling Updates

Service
- app: my-app

ReplicationController
- replicas: 2
- selector:
  - app: my-app
  - version: v1

ReplicationController
- replicas: 1
- selector:
  - app: my-app
  - version: v2

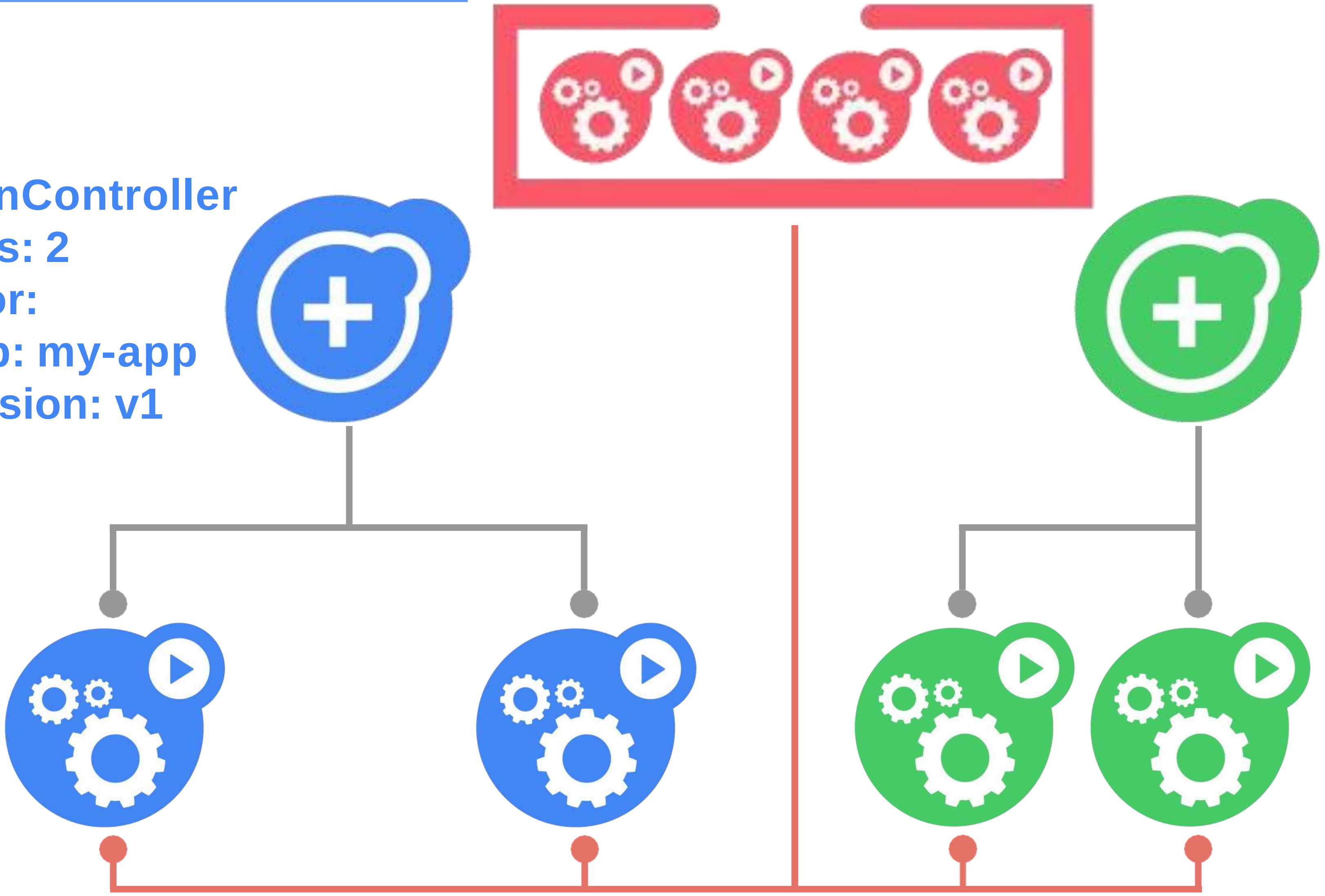


Rolling Updates

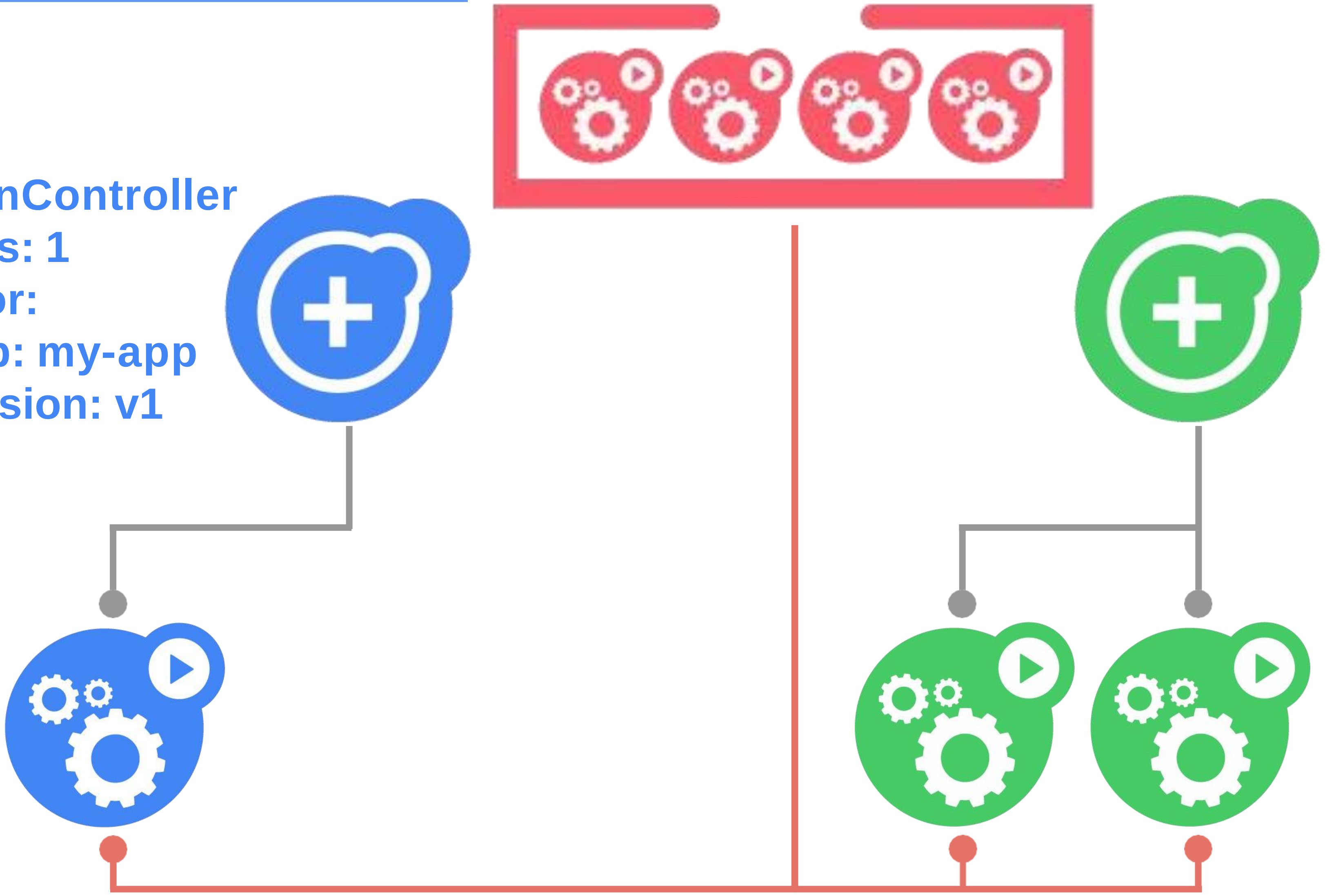
Service
- **app: my-app**

ReplicationController
- **replicas: 2**
- **selector:**
    - **app: my-app**
    - **version: v1**

ReplicationController
- **replicas: 2**
- **selector:**
    - **app: my-app**
    - **version: v2**



Rolling Updates

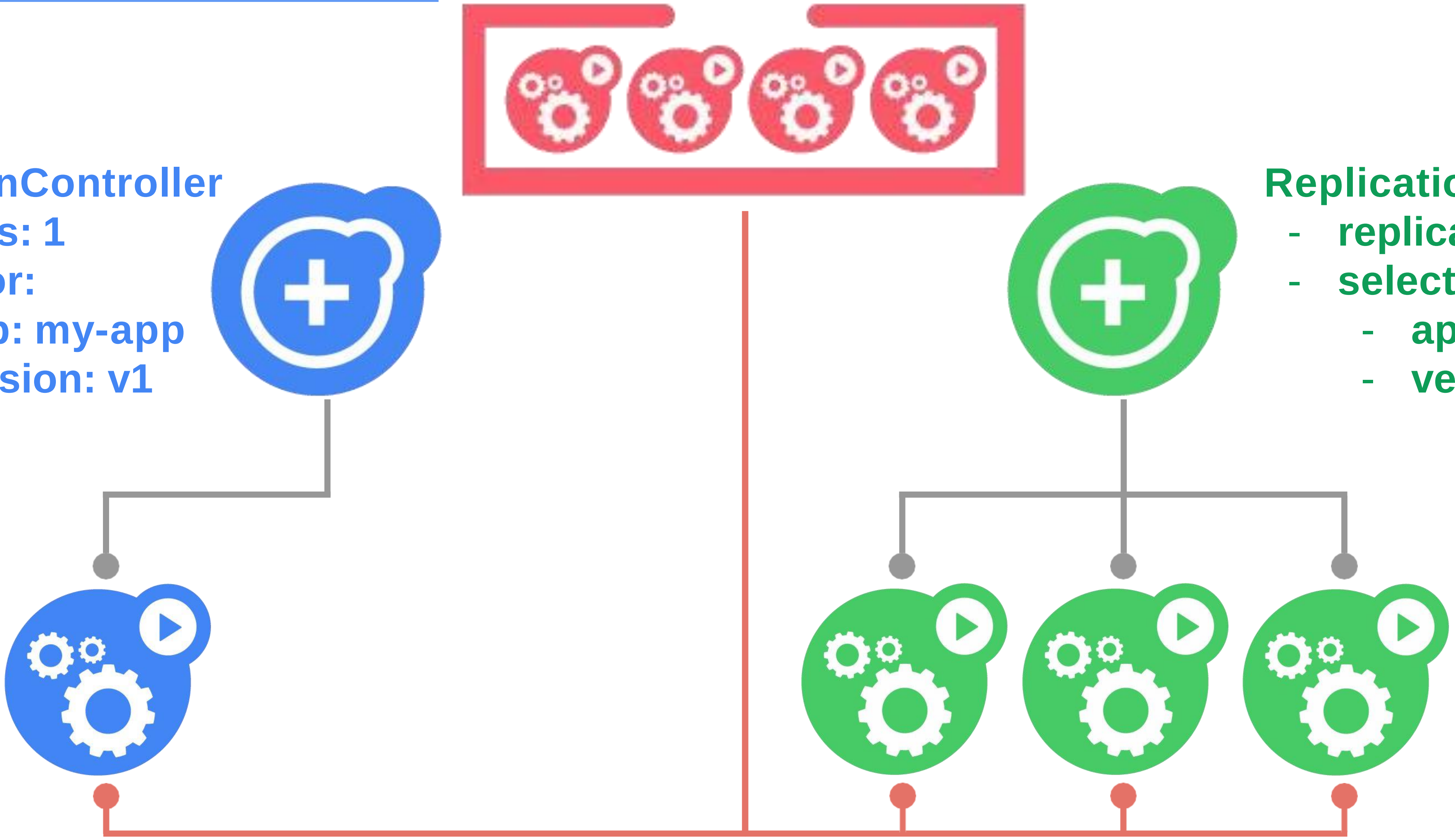


Service
- app: my-app

ReplicationController
- replicas: 1
- selector:
  - app: my-app
  - version: v1

ReplicationController
- replicas: 2
- selector:
  - app: my-app
  - version: v2

Rolling Updates



Service
- app: my-app

ReplicationController
- replicas: 1
- selector:
  - app: my-app
  - version: v1

ReplicationController
- replicas: 3
- selector:
  - app: my-app
  - version: v2

Rolling Updates

Service

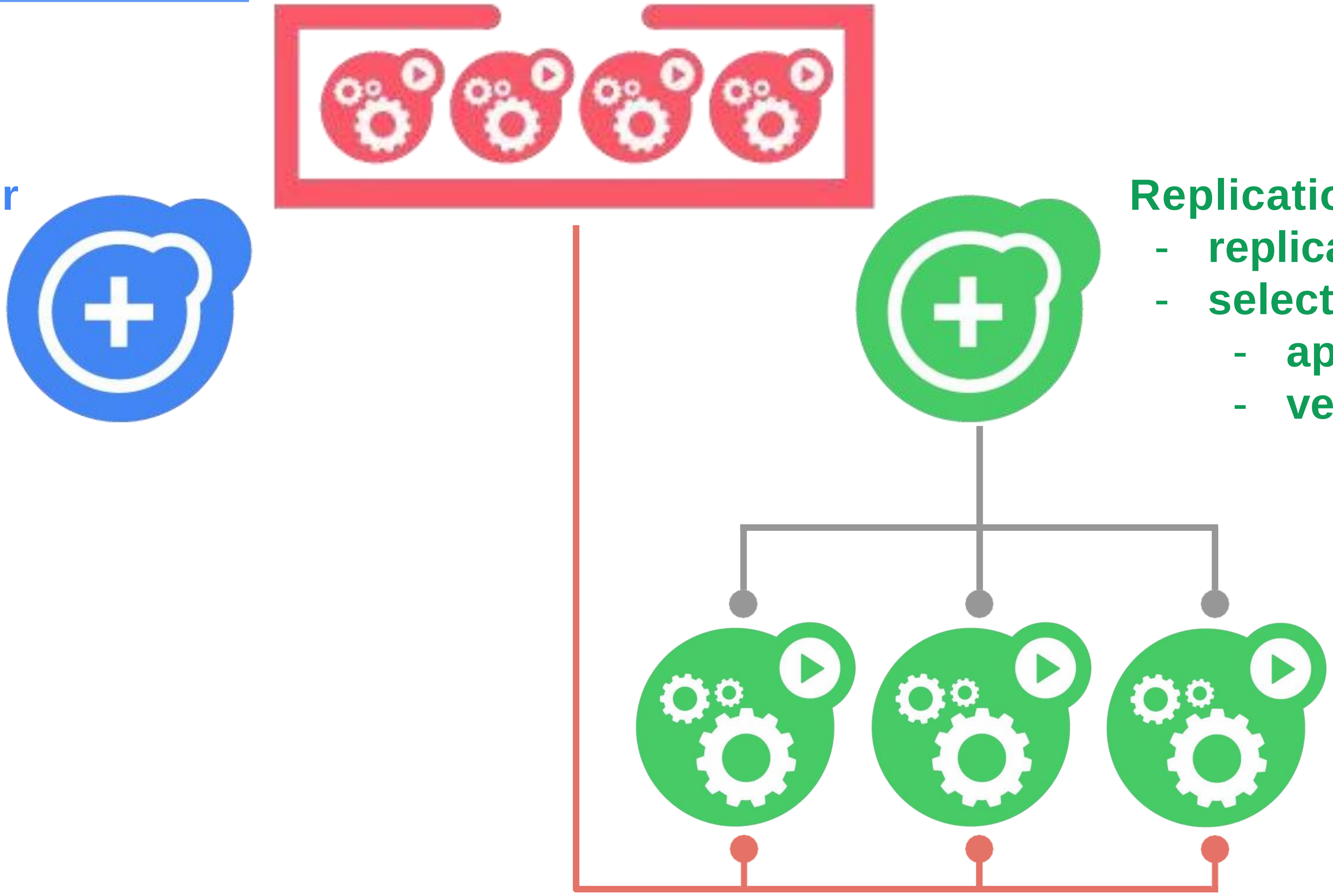
- **app: my-app**

ReplicationController

- **replicas: 0**
- **selector:**
    - **app: my-app**
    - **version: v1**

ReplicationController

- **replicas: 3**
- **selector:**
    - **app: my-app**
    - **version: v2**



New controllers in v1.1

Jobs

Manages pods that run to completion

- differentiates number running at any one time from the total number of completed runs

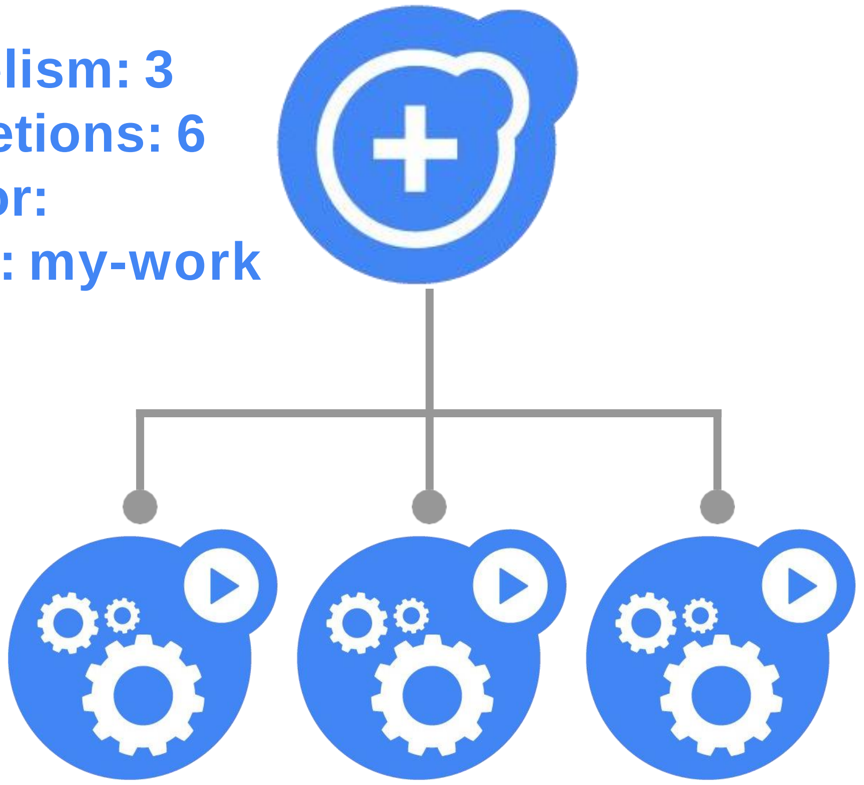
Similar to ReplicationController, but for pods that don't always restart

- workflow: restart on failure
- build/test: don't restart on app. failure

Principle: do one thing, don't overload

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

Job
- parallelism: 3
- completions: 6
- selector:
  - job: my-work



Jobs

Manages pods that run to completion

- differentiates number running at any one time from the total number of completed runs

Similar to ReplicationController, but for pods that don't always restart

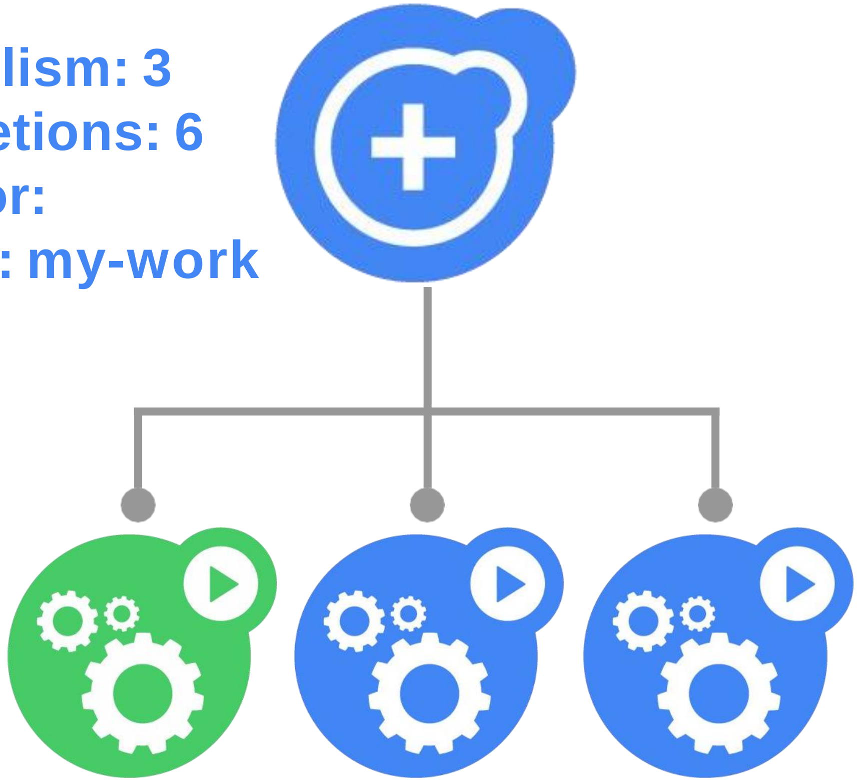
- workflow: restart on failure
- build/test: don't restart on app. failure

Principle: do one thing, don't overload

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

Job

- **parallelism: 3**
- **completions: 6**
- **selector:**
  - **job: my-work**



Jobs

Manages pods that run to completion

- differentiates number running at any one time from the total number of completed runs

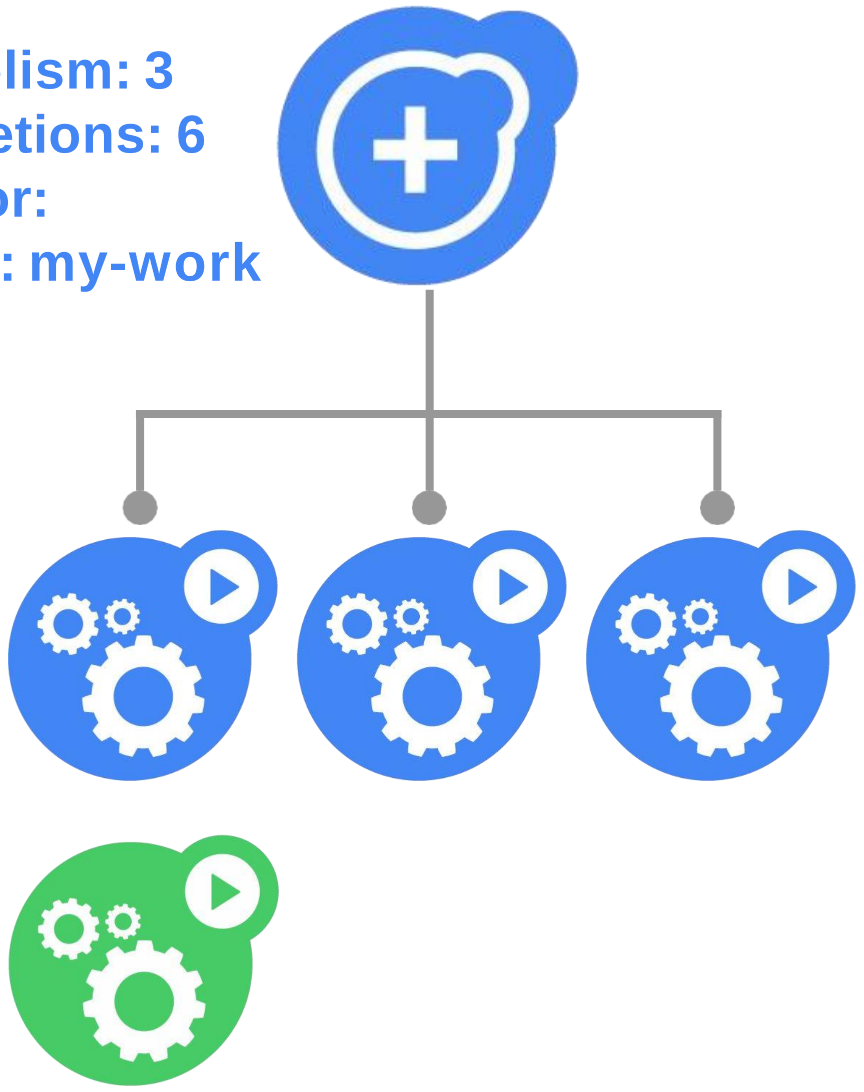
Similar to ReplicationController, but for pods that don't always restart

- workflow: restart on failure
- build/test: don't restart on app. failure

Principle: do one thing, don't overload

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

```
Job
 -  parallelism: 3
 -  completions: 6
 -  selector:
     -  job: my-work
```



Jobs

Manages pods that run to completion

- differentiates number running at any one time from the total number of completed runs

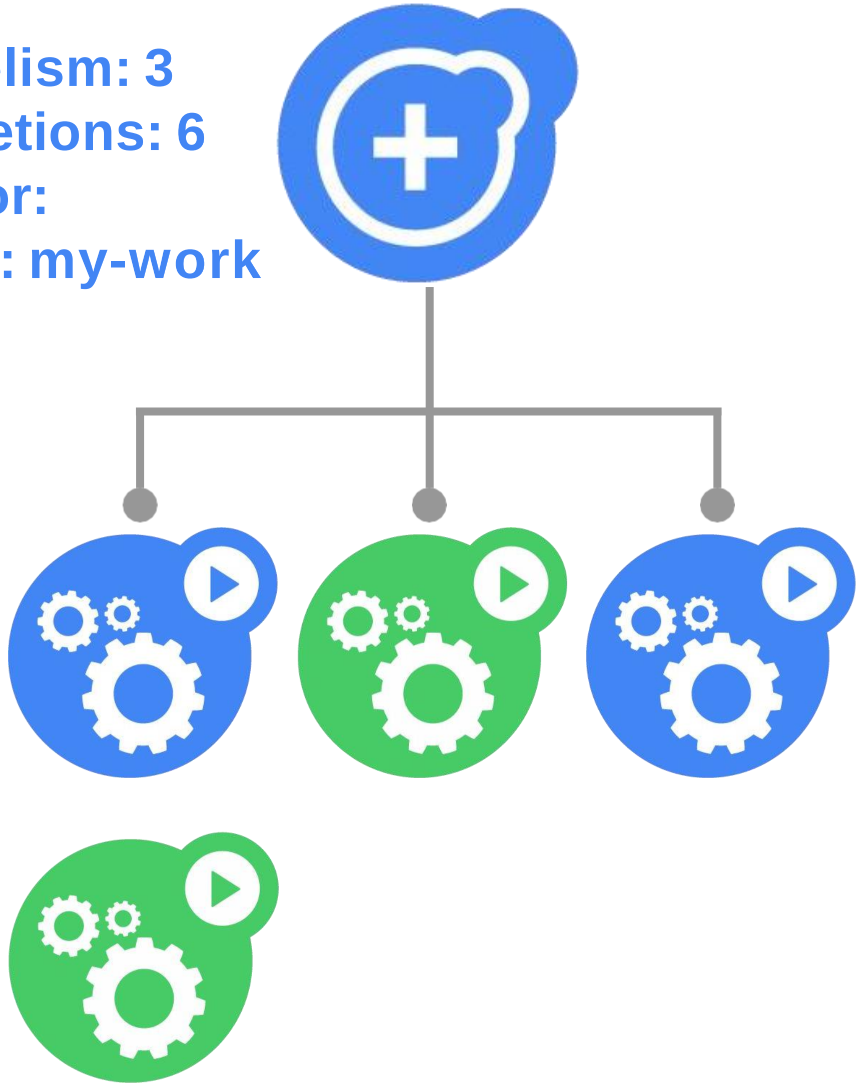
Similar to ReplicationController, but for pods that don't always restart

- workflow: restart on failure
- build/test: don't restart on app. failure

Principle: do one thing, don't overload

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

Job
- **parallelism: 3**
- **completions: 6**
- **selector:**
  - **job: my-work**



Jobs

Manages pods that run to completion

- differentiates number running at any one time from the total number of completed runs

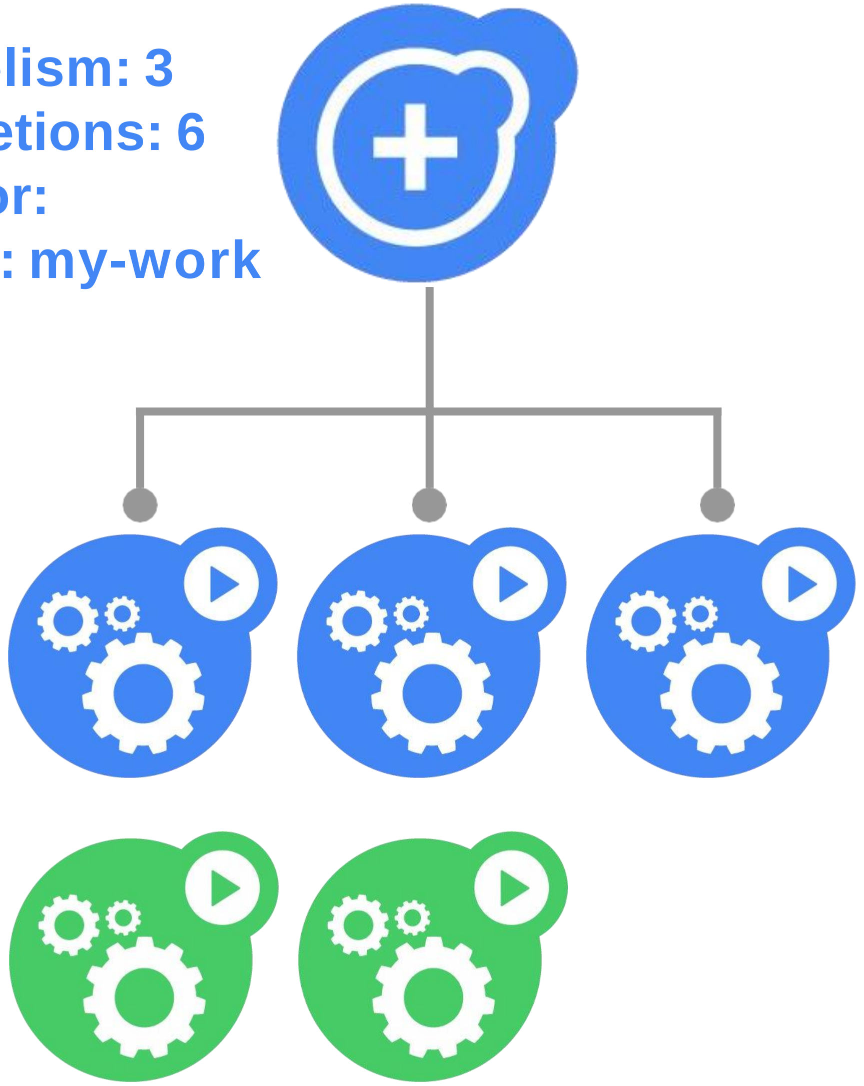
Similar to ReplicationController, but for pods that don't always restart

- workflow: restart on failure
- build/test: don't restart on app. failure

Principle: do one thing, don't overload

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

```
Job
 -  parallelism: 3
 -  completions: 6
 -  selector:
     -  job: my-work
```



Jobs

Manages pods that run to completion

- differentiates number running at any one time from the total number of completed runs

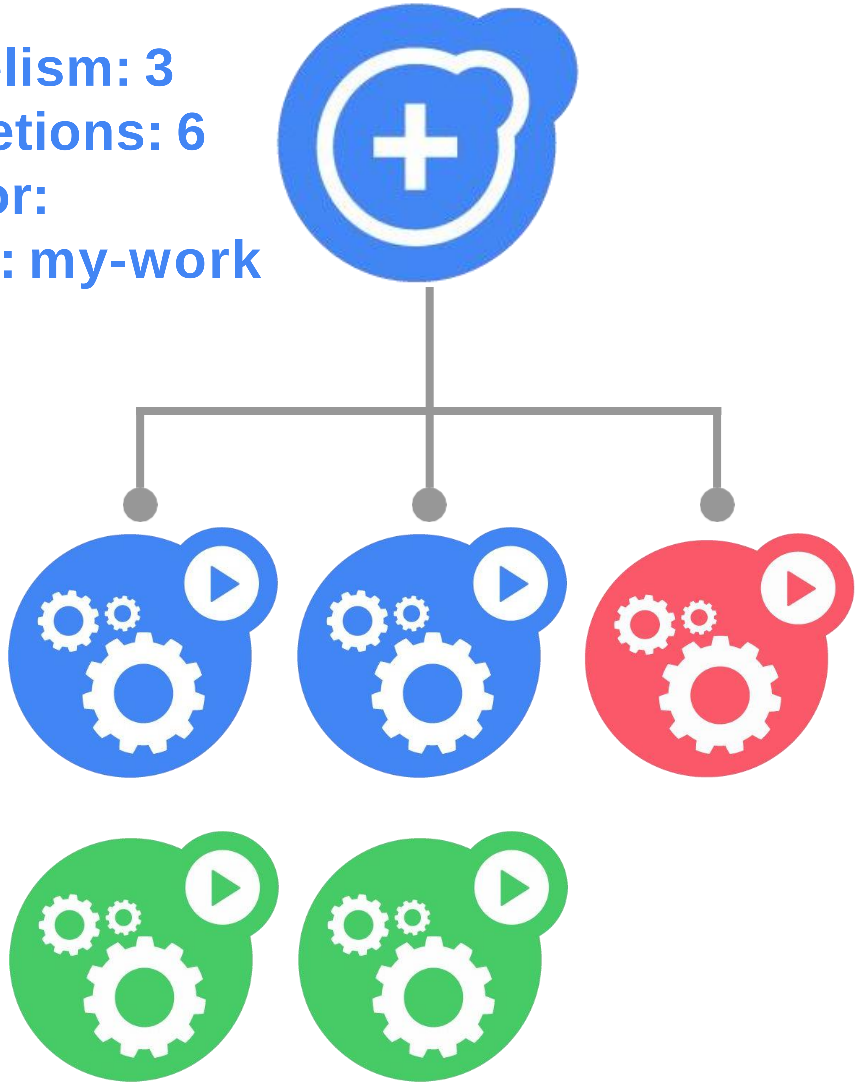
Similar to ReplicationController, but for pods that don't always restart

- workflow: restart on failure
- build/test: don't restart on app. failure

Principle: do one thing, don't overload

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

```
Job
 -  parallelism: 3
 -  completions: 6
 -  selector:
      -  job: my-work
```



Jobs

Manages pods that run to completion

- differentiates number running at any one time from the total number of completed runs

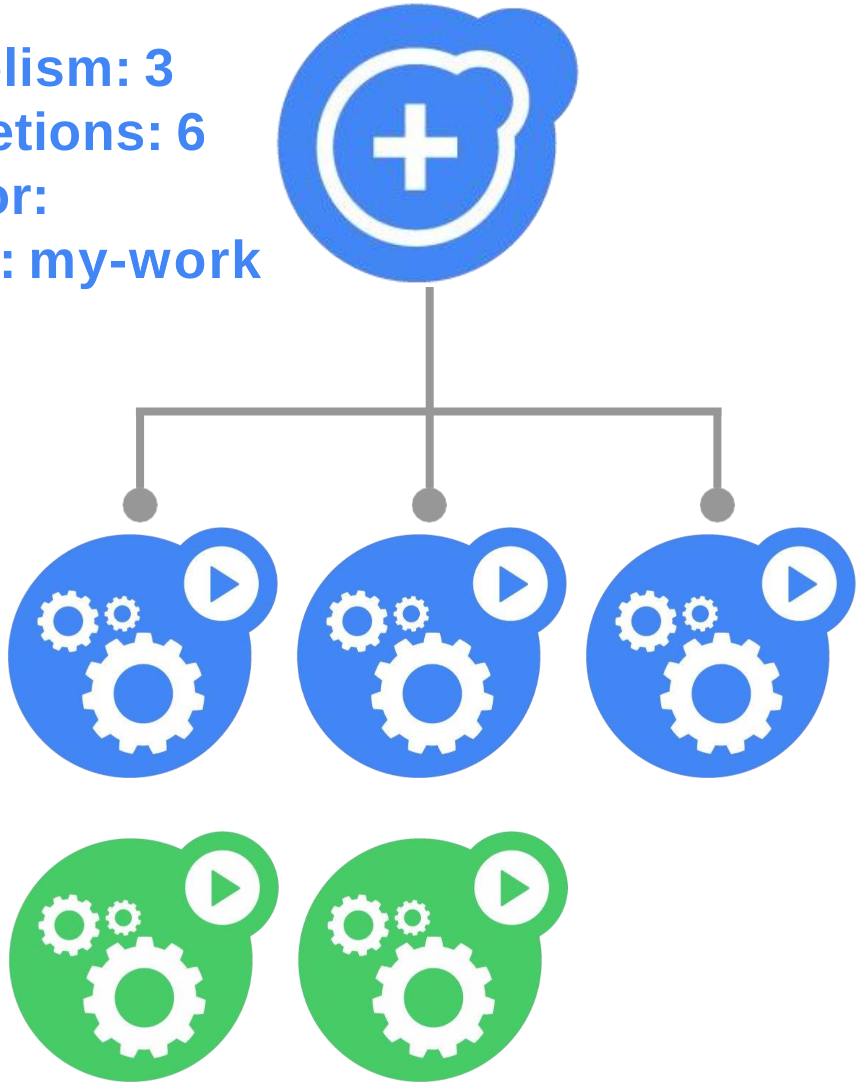
Similar to ReplicationController, but for pods that don't always restart

- workflow: restart on failure
- build/test: don't restart on app. failure

Principle: do one thing, don't overload

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

```
Job
 -  parallelism: 3
 -  completions: 6
 -  selector:
      -  job: my-work
```



Jobs

Manages pods that run to completion

- differentiates number running at any one time from the total number of completed runs

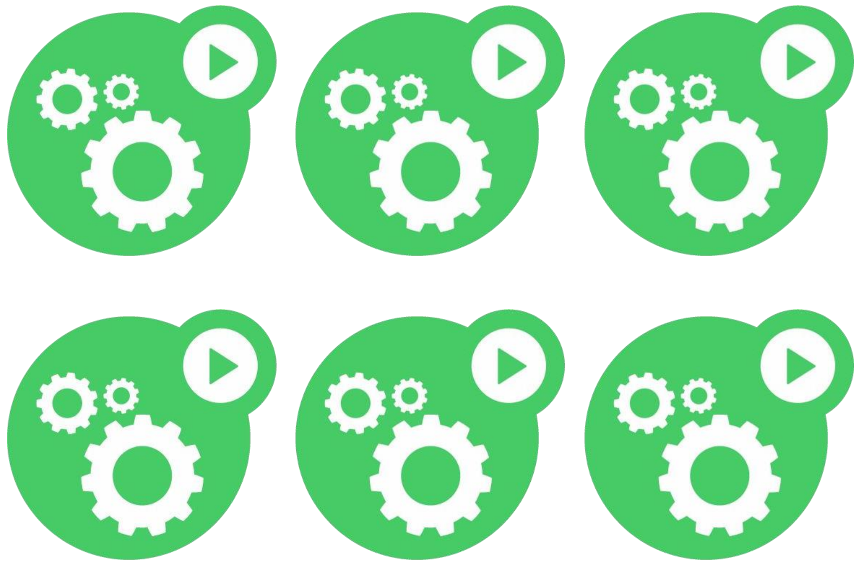
Similar to ReplicationController, but for pods that don't always restart

- workflow: restart on failure
- build/test: don't restart on app. failure

Principle: do one thing, don't overload

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

Job
- **parallelism: 3**
- **completions: 6**
- **selector:**
  - **job: my-work**



DaemonSets

Runs a Pod on every node

- or a selected subset of nodes

Not a fixed number of replicas

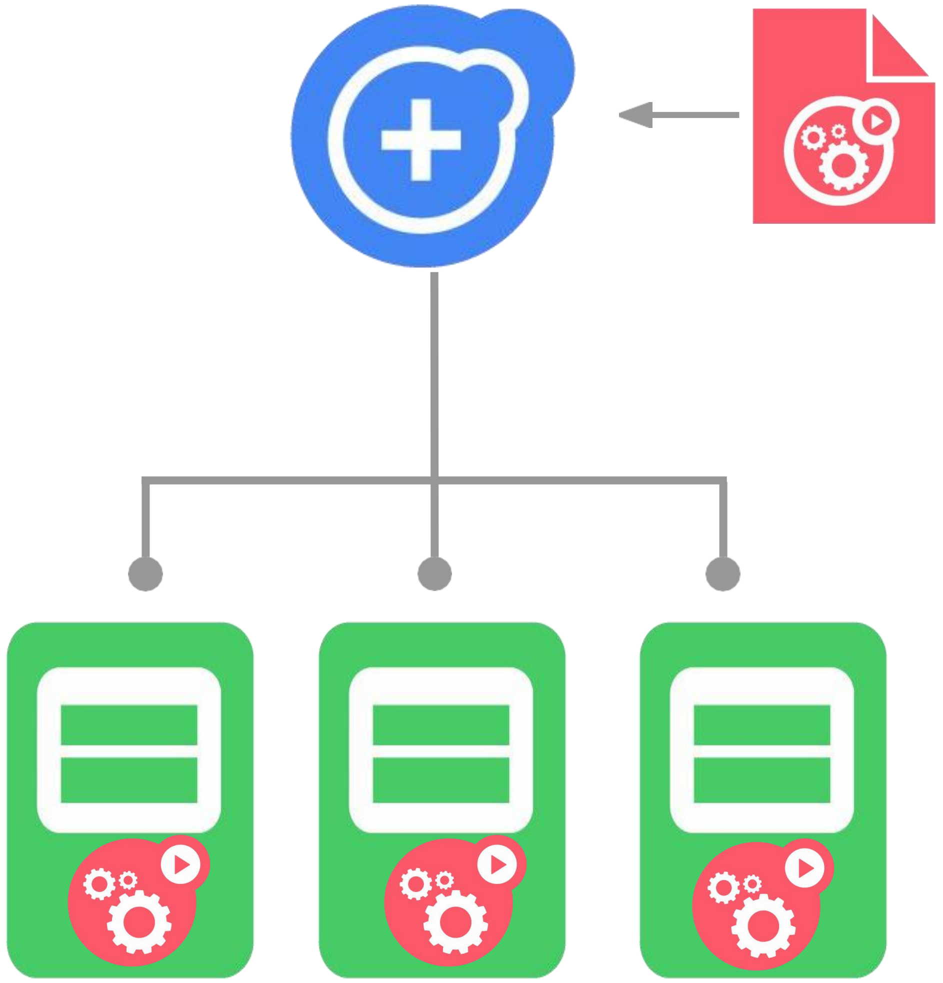
- created and deleted as nodes come and go

Useful for running cluster-wide services

- logging agents
- storage systems

DaemonSet manager is both a controller and scheduler

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



Deployment

Rollouts as a service

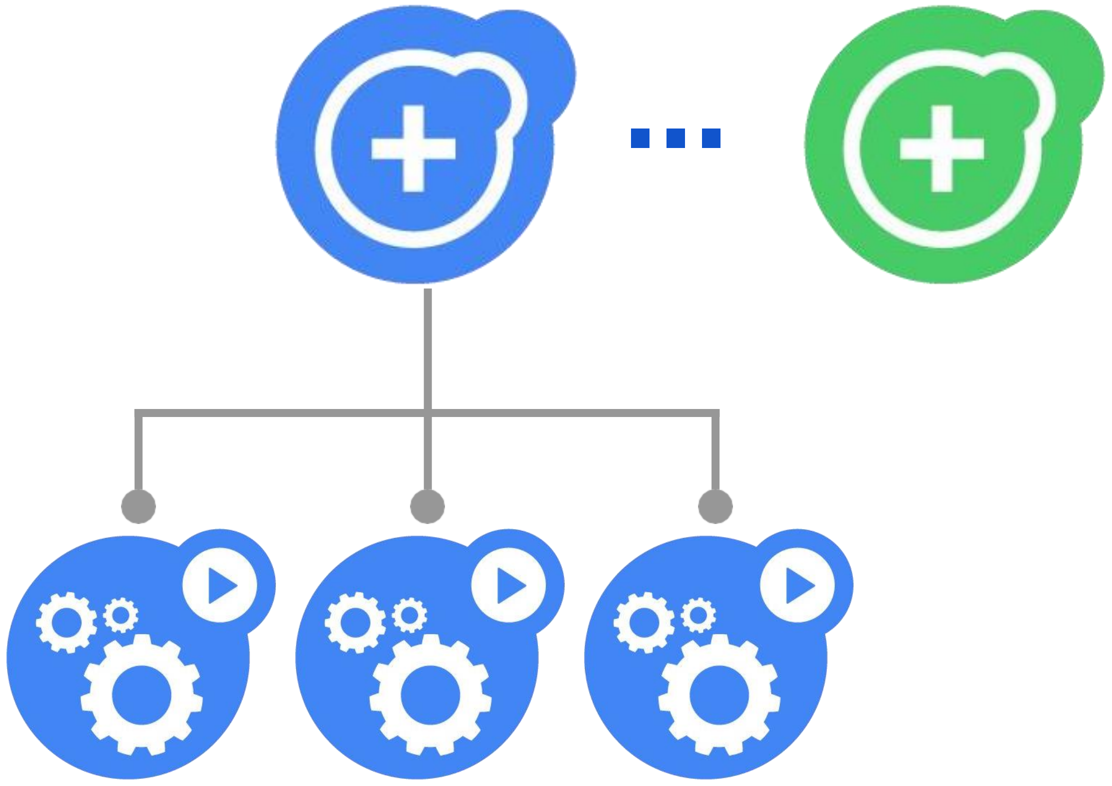
- updates to pod template will be rolled out by controller
- can choose between rolling update and recreate

Enables declarative updates

- manipulates replication controllers and pods so clients don't have to

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

Deployment
- **strategy: {type: RollingUpdate}**
- **replicas: 3**
- **selector:**
  - **app: my-app**



Conclusion

Take away

- Decoupling applications from infrastructure creates new opportunities
- Kubernetes
    - is container-centric infrastructure
        - which includes a lot more than just running containers
    - facilitates management of containers in production
    - provides a foundation for building a workload-management ecosystem
- This has enabled Platform as a Service systems to be built on Kubernetes
    - Apache Stratos
    - Openshift 3: co-designed and co-developed with Kubernetes
    - Deis: Heroku-inspired Docker-based PaaS
    - Gondor: Python-aaS

Kubernetes is **Open**

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

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

WSO2Con 2015 USA San Francisco

Thank You

Design principle summary

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 **Cattle >**

Pets: Manage your workload in bulk

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