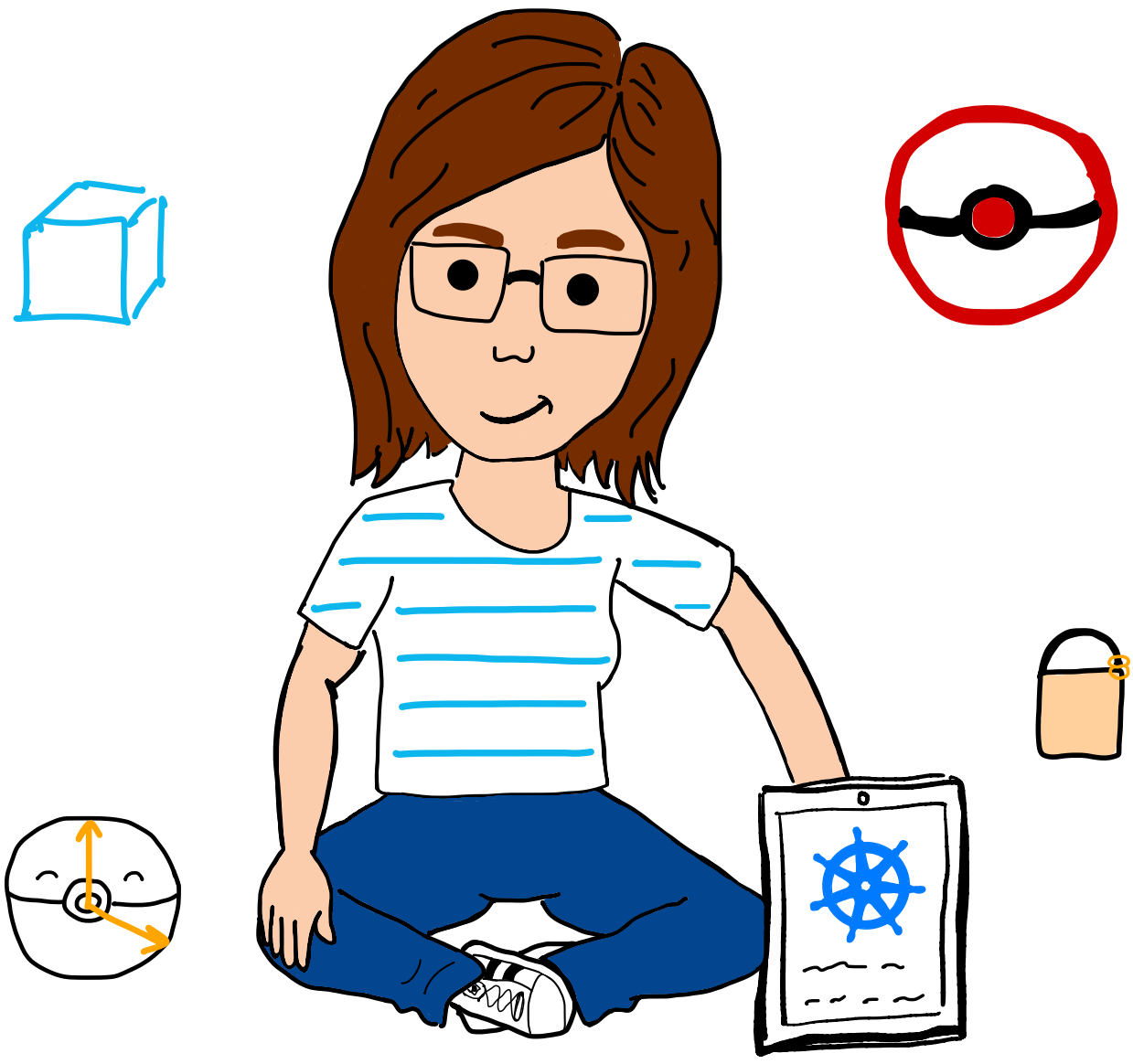
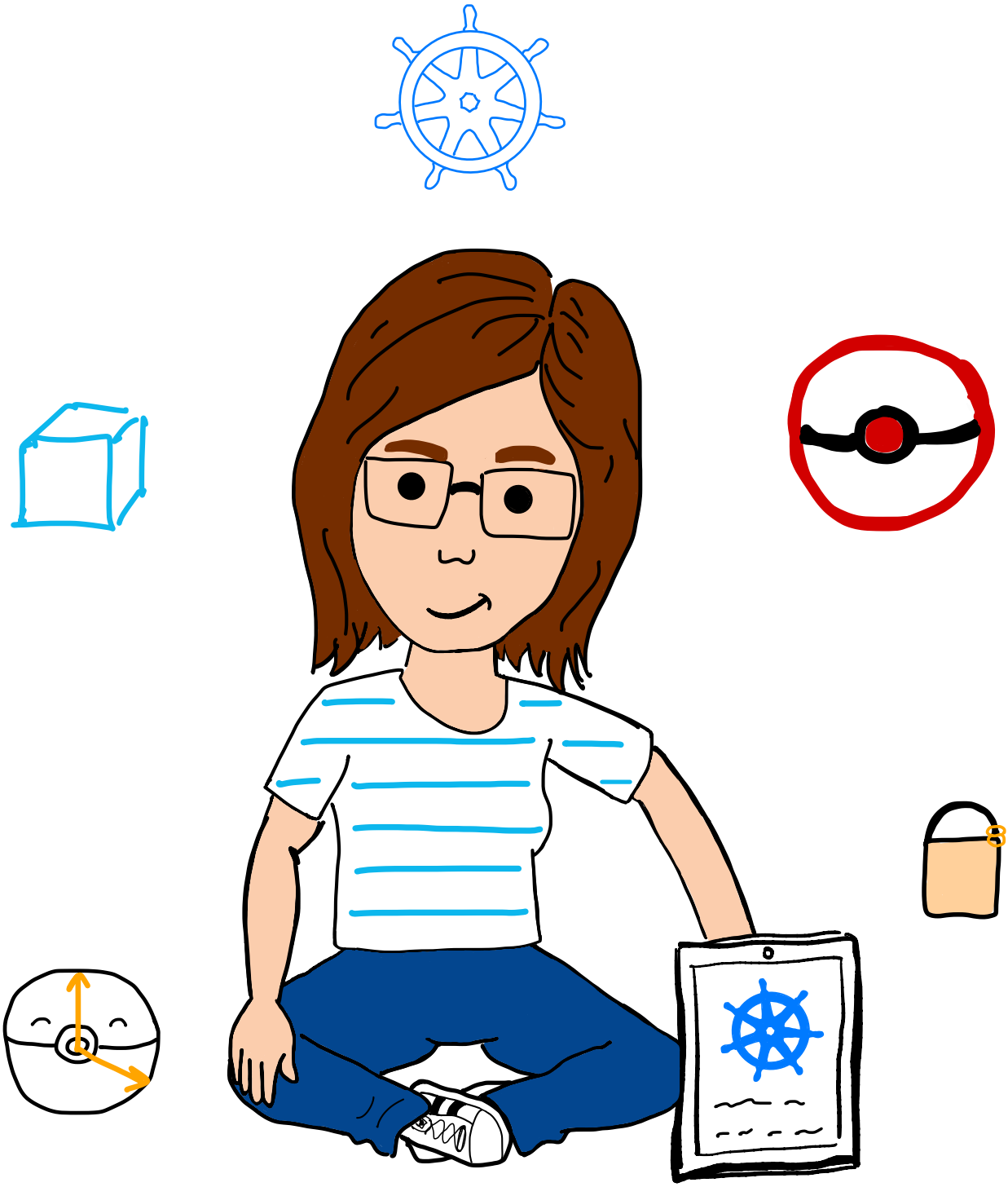


Understanding Kubernetes in a visual way

Learn & discover Kubernetes in sketchnotes
- with some **tips** included -

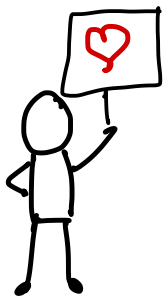
Aurélie Vache



Understanding Kubernetes in a visual way

Aurélie Vache

Special thanks to:



Laurent, my husband, Alexandre, my son, and all the caring people who believe in me everyday ☺

Thank you so much !!!

Reviewer:

Thanks to Gaelle Acas, Denis Germain & Stéphane Philippart who took the time to review this book.

Changelog:

Release Date	:	31/05/2020
Updated Date	:	19/12/2022
Release Version	:	2.0.1
Changelog	:	1.26 Release
Length	:	272 pages
Kubernetes	:	1.26

Licence:

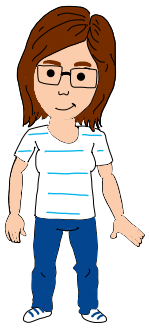
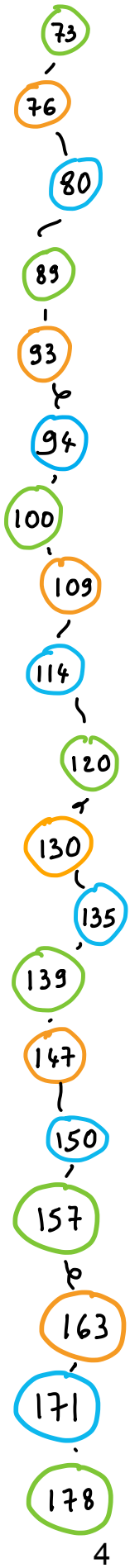


Table of contents

- Secret
- Deployment >
  - Rolling Update
  - Pull images configuration
- ReplicaSet
- DaemonSet
- Service
- Label & Selector
- Ingress
- PV, PVC & StorageClass
- Horizontal Pod Autoscaler (HPA)
- LimitRange
- Resource's requests & limits
- Pod Disruption Budget (PDB)
- Quality of Service (QoS)
- Network Policies
- RBAC
- Pod Security Policy (PSP)
- Pod (Anti) Affinity & Node (Anti) Affinity



Kyverno

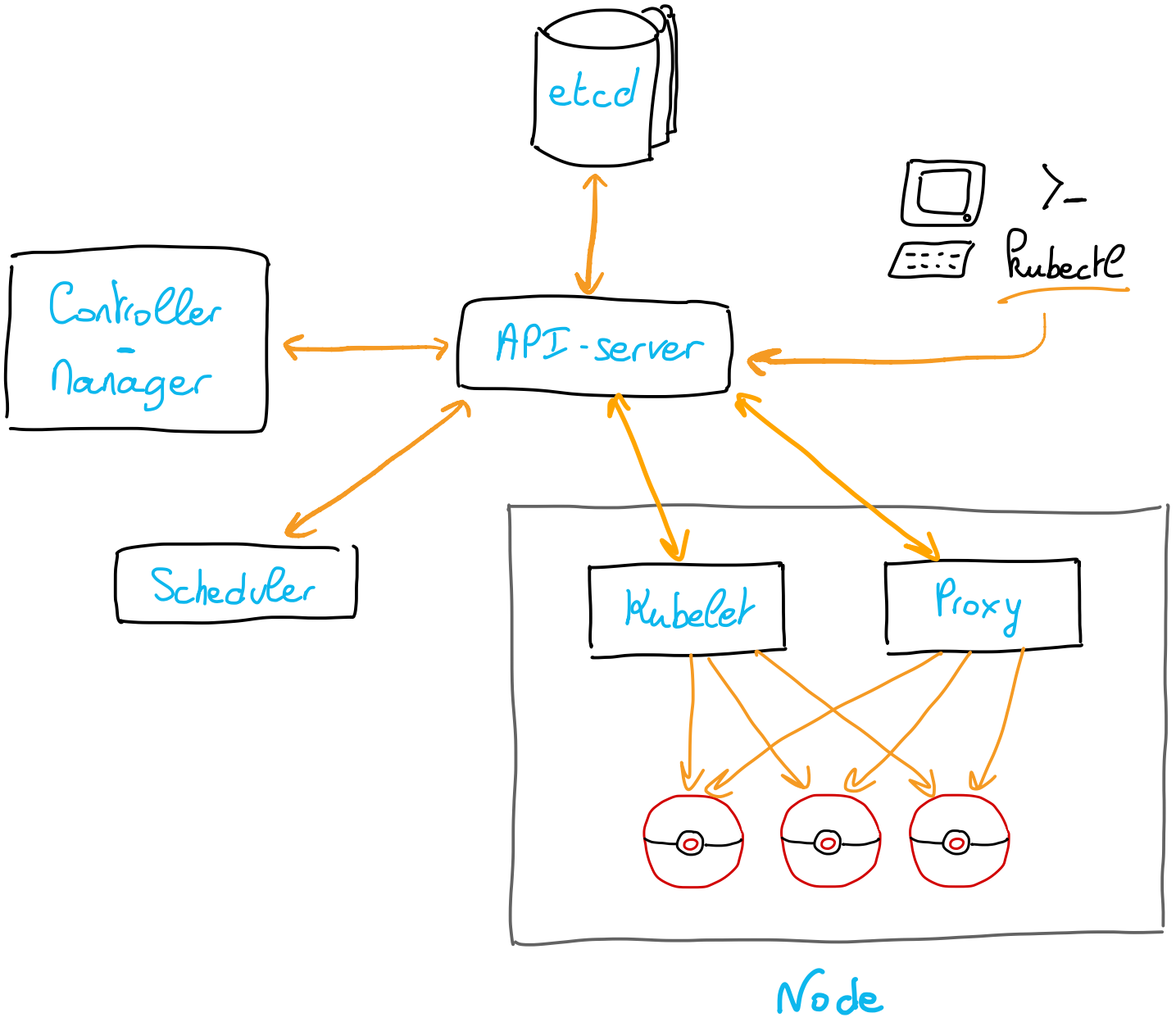
Changes >

- Kubernetes 1.18
- Kubernetes 1.19
- Kubernetes 1.20
- Kubernetes 1.21
- Kubernetes 1.22
- Kubernetes 1.23
- Kubernetes 1.24
- Kubernetes 1.25
- Kubernetes 1.26

⚠ Docker deprecation

Glossary

Kubernetes Components



etcd

- Distributed key-value database
- Stores & replicates cluster state
- Distributed consensus based on the quorum model
- ⚠ Master's etcd : Single Point of failure !

API-server

- Exposes Kubernetes API
- Kubernetes Control Plane Frontend

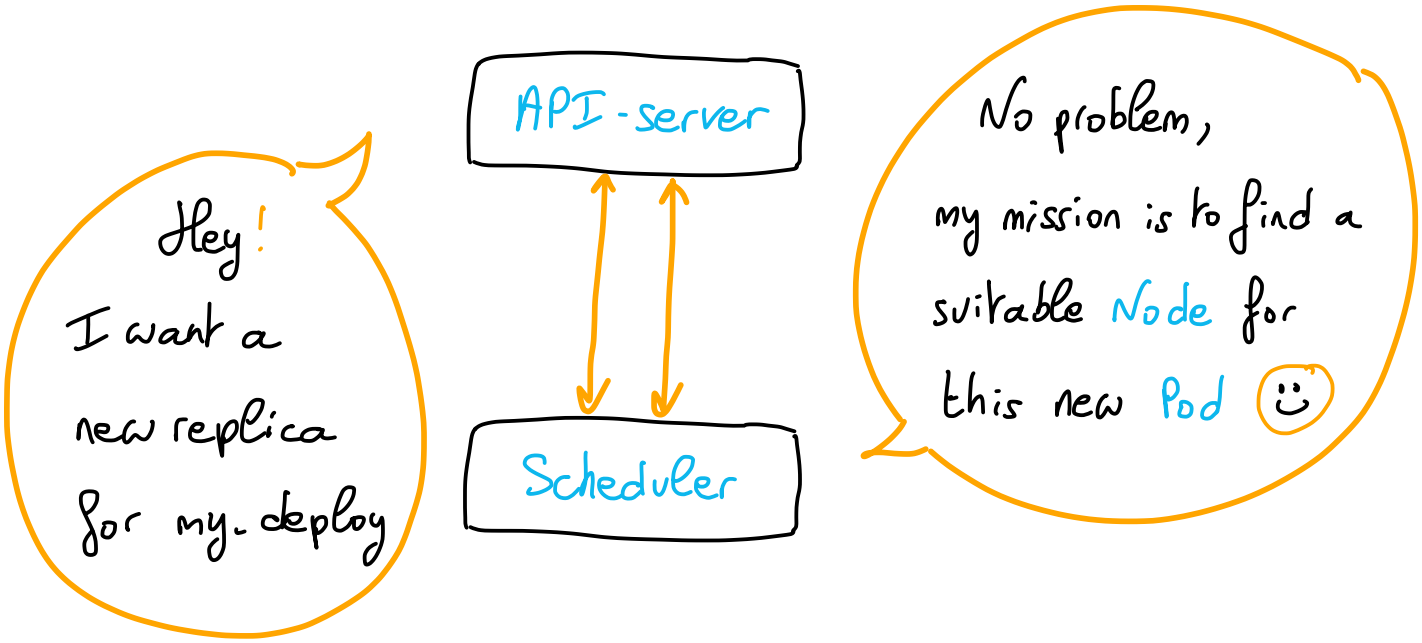
Everyone is talking to me, it's cool !
But if I have to much work,
I'm designed to scale horizontally,
so you can balance traffic between instances.

API-server

Scheduler

→ Responsible for finding the best Node for newly created Pods

→ Existing Nodes need to be filtered according to the specific scheduling requirements



Controller-Manager

→ Responsible to make changes in order to move the current state to the desired state

→ Runs several separate controller processes:

- Node Controller

  ↝ watches Nodes & do action when they are down

- Replication Controller

  ↝ ensures the desired number of Pods are always up and available

- Endpoint Controller

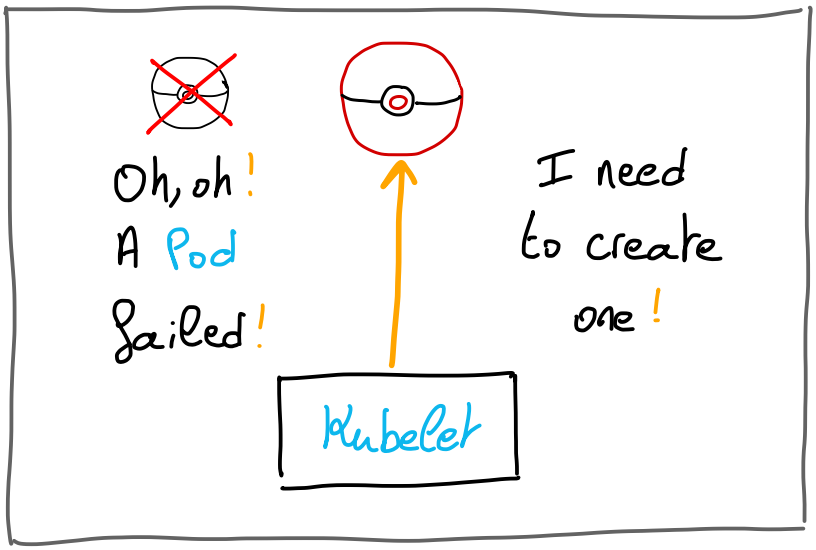
  ↝ populates the endpoint objects (responsible for Services & Pods connections)

- Service Account & TokenController

  ↝ creates default accounts and API access tokens for new namespaces

Kubelet

- Responsible for running state on each Node
- Checks that all containers on each Node are healthy
- Can create and delete Pods
- One instance of Kubelet per Node



Node

Proxy

- Runs on each Nodes
- Allows network communication to Pods

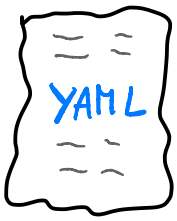
I want the list of Pods
in my-ns namespace

```
$ kubectl get pods -n my-ns
```

OK,
but let's find in
which cluster

→ Kubectl knows where (in which cluster)
to execute the command thanks to Kubeconfig file

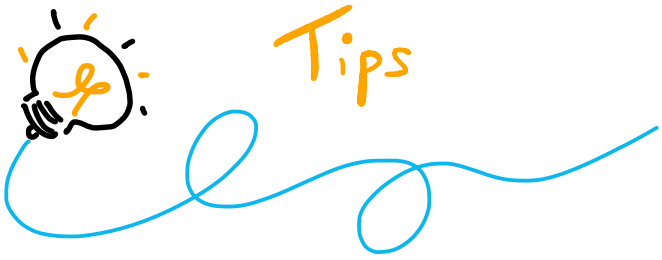
→ Kubeconfig files are structured in YANL files



How kubeconfig files are loaded?

1. `--kubeconfig` flag, if specified
2. `KUBECONFIG` environment variable, if set
3. `$HOME/.kube/config`, by default

You can't just append YAML kubeconfig files in one kubeconfig file.



It's possible to specify several files in `KUBECONFIG` environment variable:

```
$ export KUBECONFIG=file1:file2:file3
```

Kubectl Tips

Kubectl version

Which version of Kubectl should I install & use?

Answer:

Kubectl is supported one minor version of Kubernetes API-Server (older and newer)

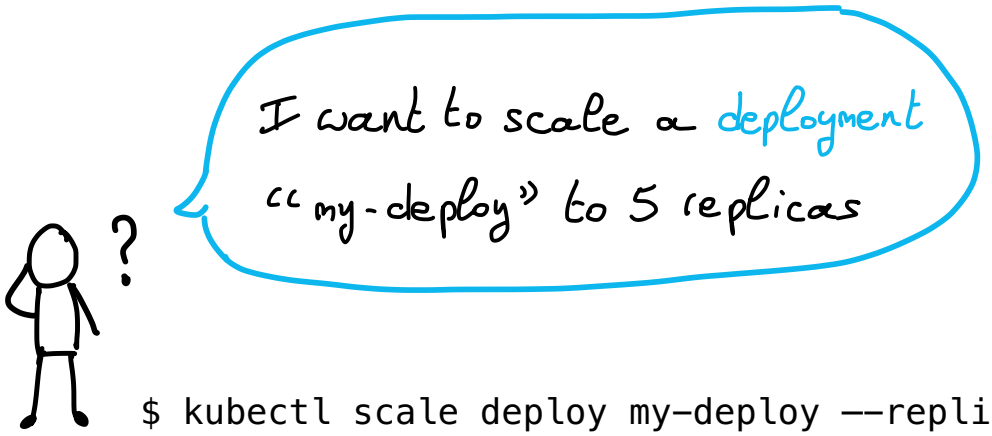
Example:

API-Server: 1.23

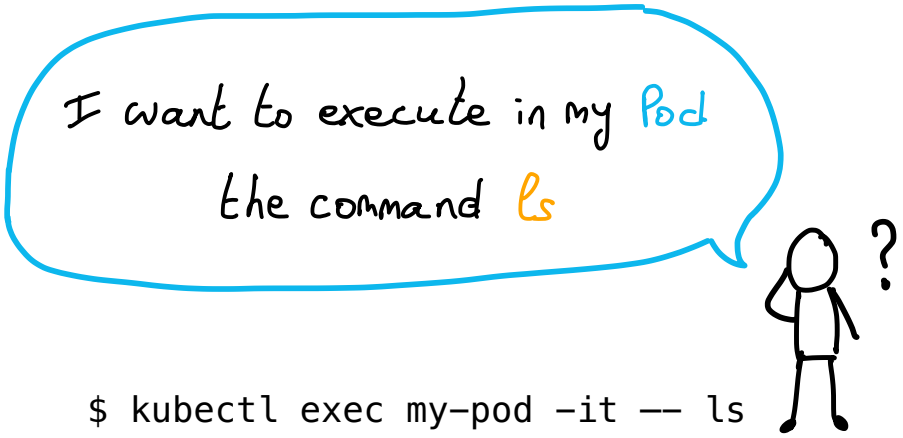
Kubectl: 1.22, 1.23 or 1.24

Kubectl CLI

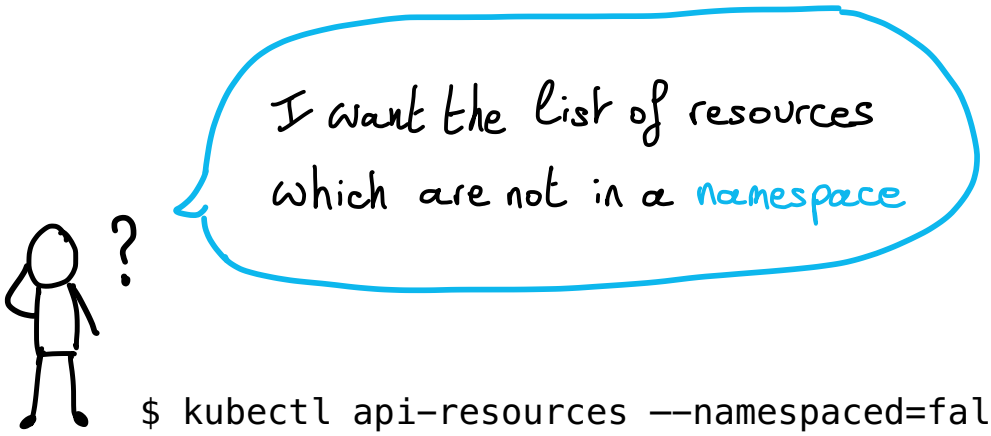
Kubectl = A very logical CLI!



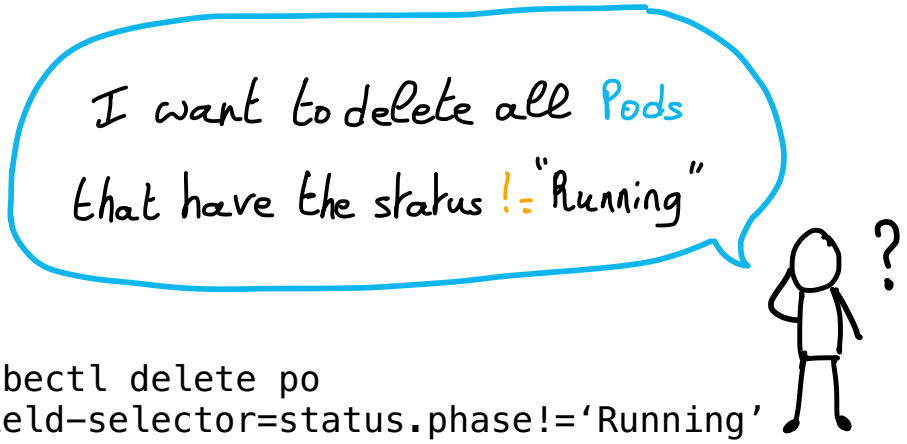

```
$ kubectl scale deploy my-deploy --replicas=5
```



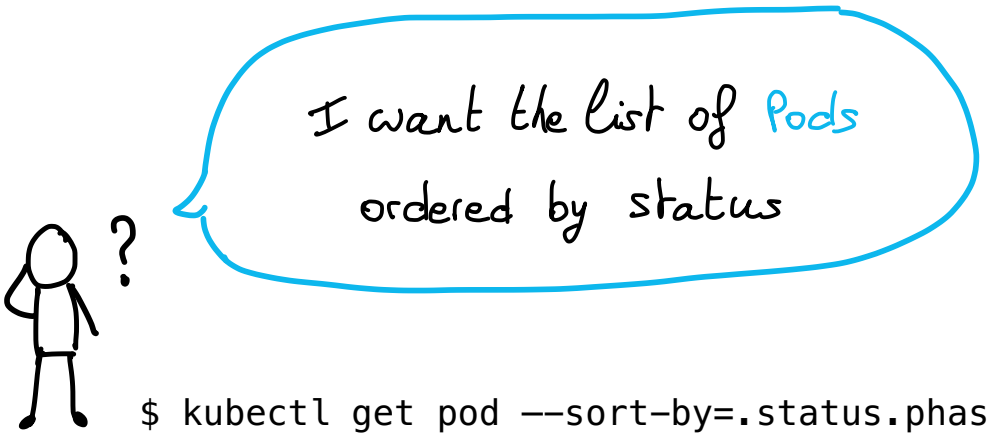

```
$ kubectl exec my-pod -it -- ls
```



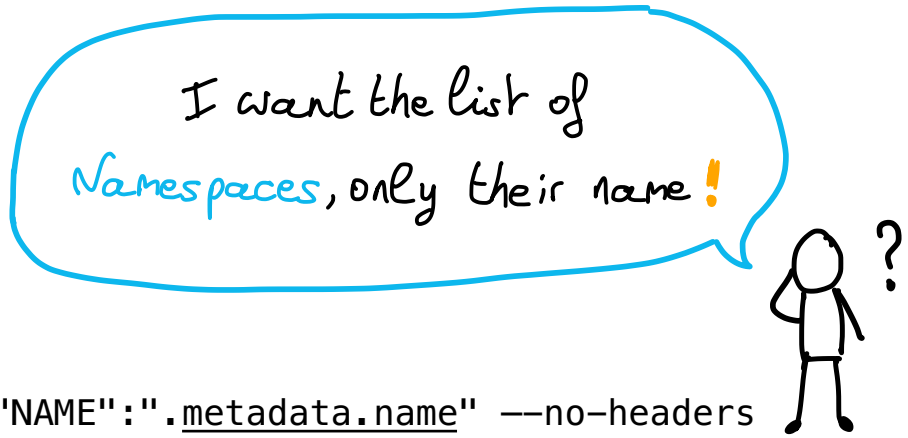

```
$ kubectl api-resources --namespaced=false
```



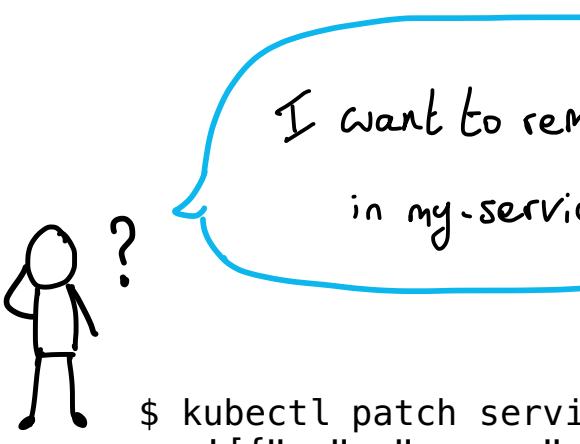

```
$ kubectl delete po
--field-selector=status.phase!='Running'
```




```
$ kubectl get pod --sort-by=.status.phase
```




```
$ kubectl get ns
-o custom-columns="NAME":".metadata.name" --no-headers
```




```
$ kubectl patch service my-service --type json
-p '[{"op": "remove", "path": "/spec/selector/version"}]'
```

--watch

This option aims to listen for changes to a particular object

Example:

```
$ kubectl get pod -w
```

-o wide

This kubectl option includes additional informations

Example:

```
$ kubectl get pod my-pod -o wide
```

-o name

This kubectl option displays only the type
& the name of the resource

Example:

```
$ kubectl get secret
-l sealedsecrets.bitnami.com/sealed-secrets-key -o name
```

cascade

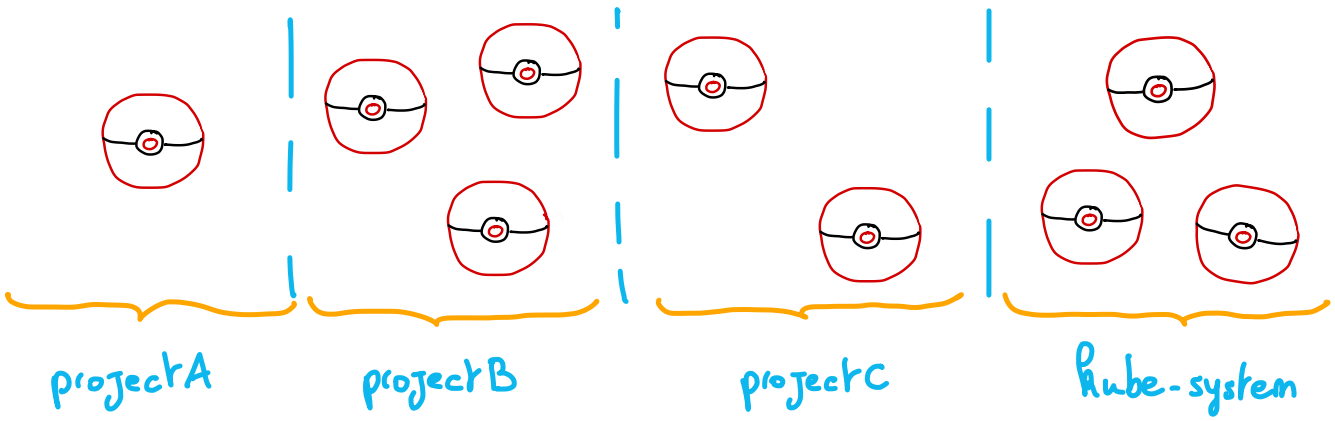
This option allows to delete only resource (not child objects)

Example:

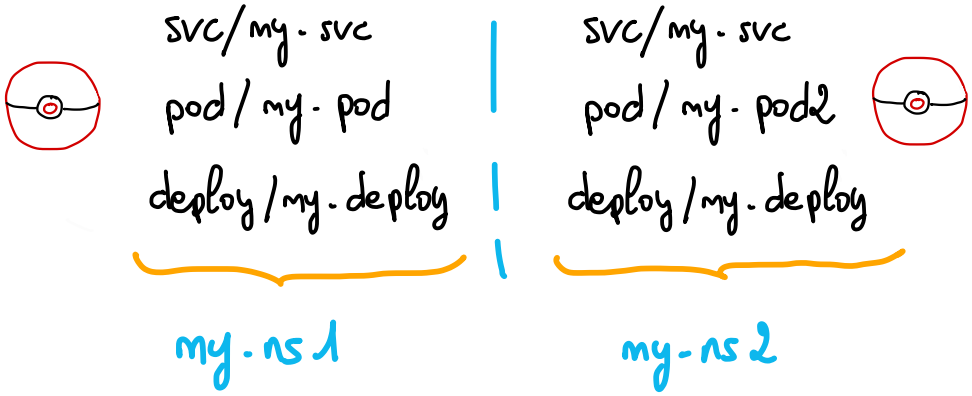
```
$ kubectl delete job my-job cascade=false -n my-namespace
```

Namespaces

→ Way of isolation
- per project
- per team
- per family of component

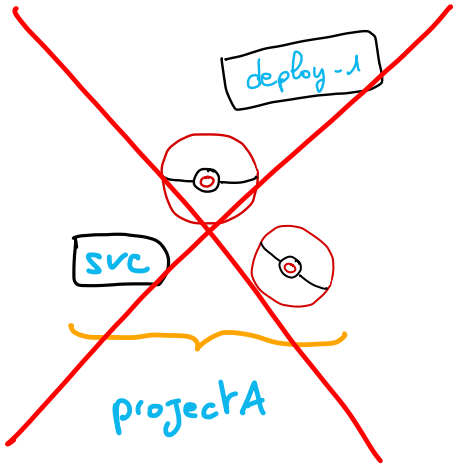


→ Resources names must be unique inside a namespace, but can be identical across namespaces:



→ Each resources can appear in only one namespace

⚠ If a namespace is deleted, all resources inside are deleted too

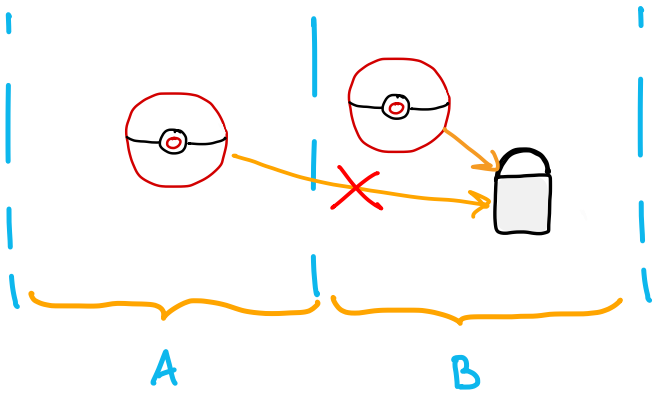


⚠

Not all resources are namespaced ?

Node, PV, PSP ...

⚠ A Pod in namespace A can't read a secret from namespace B



Special namespaces:

default	→	namespace by default
kube-system	→	for objects created by Kubernetes
kube-public	→	reserved mainly for cluster use & in case specific resources should be publicly available
kube-node-lease	→	Node's heartbeat lease object

Heartbeats, sent by Nodes, help determine the availability of a Node.

2 forms of heartbeats:

- updates of NodeStatus
- lease object

Switch to my-ns namespace

```
$ kubectl config set-context --current --namespace=my-ns
```

or install and use Kubens tool (p.211)

```
$ kubens my-ns
```

View current namespace

```
$ kubectl config view | grep namespace
```

List all namespaces

```
$ kubectl get namespaces
```

List all Pods in all namespaces

```
$ kubectl get pods --all-namespaces
```

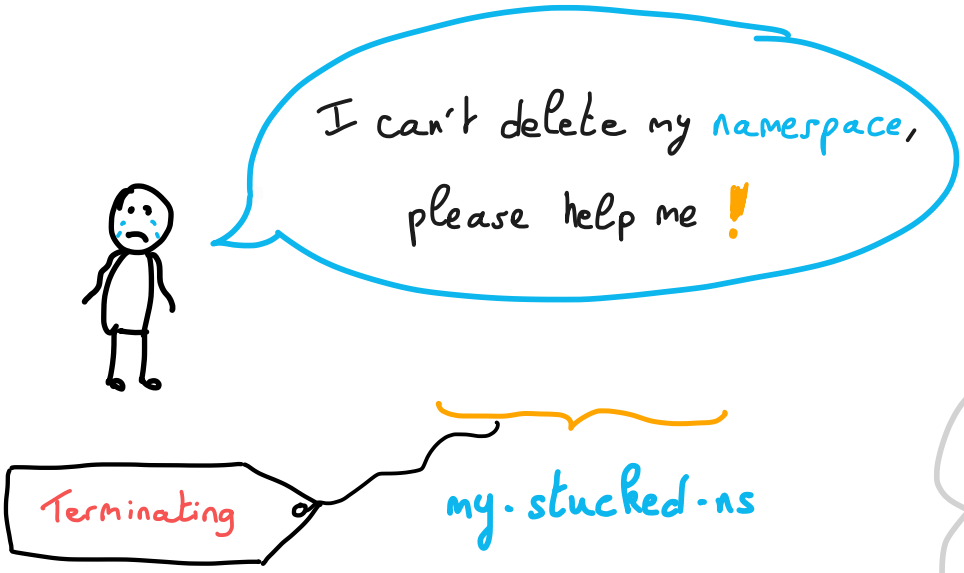
or

```
$ kubectl get pods -A
```

List all resources types that are "namespaced"

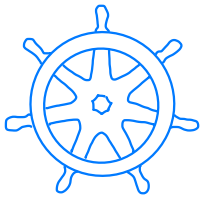
```
$ kubectl api-resources --namespaced=true
```

Delete a namespace stucked in "Terminating" state




```
$ kubectl edit ns my-stucked-ns
```

and remove entries in finalizers section



Resource Quotas

→ Limit usage and number of resources in a namespace

If a quota is enabled with CPU & Memory,
you need to specify requests & limits for Pods

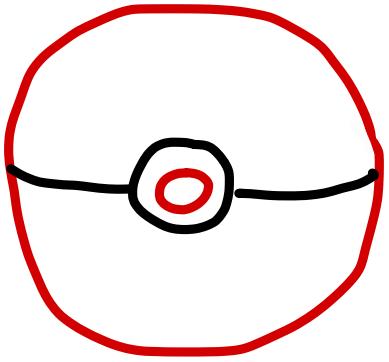
- Create a ResourceQuota

```
$ kubectl create quota my-quota --hard=cpu=1,pods=2
```

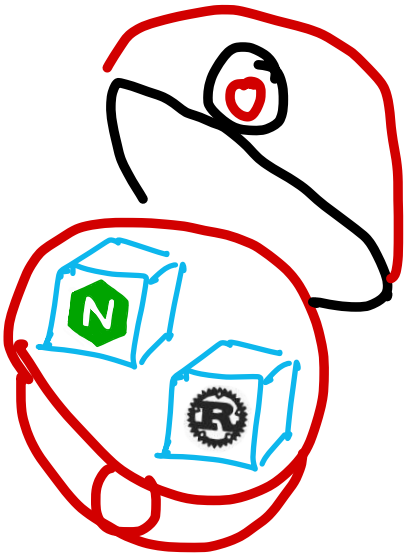
- Describe a ResourceQuota & display what is consumed and the limits

```
$ kubectl describe quota my-quota -n my-namespace
```

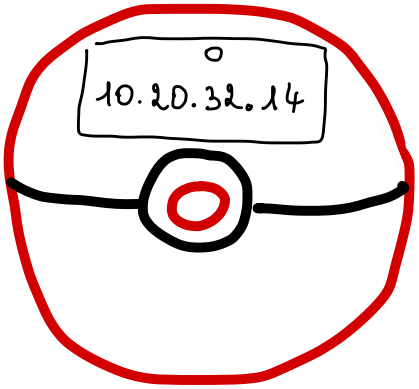
Pod



Smallest unit
deployable & runnable
in a cluster



Can contain
several containers



One IP address
per Pod

› Create a Pod with busybox image in my-namespace

```
$ kubectl run busybox --image=busybox --restart=Never
-n my-namespace
```

› Create a Pod with busybox image

& run a command 'env'

```
$ kubectl run busybox --image=busybox --restart=Never
-n my-namespace -it -- /bin/sh -c 'env'
```

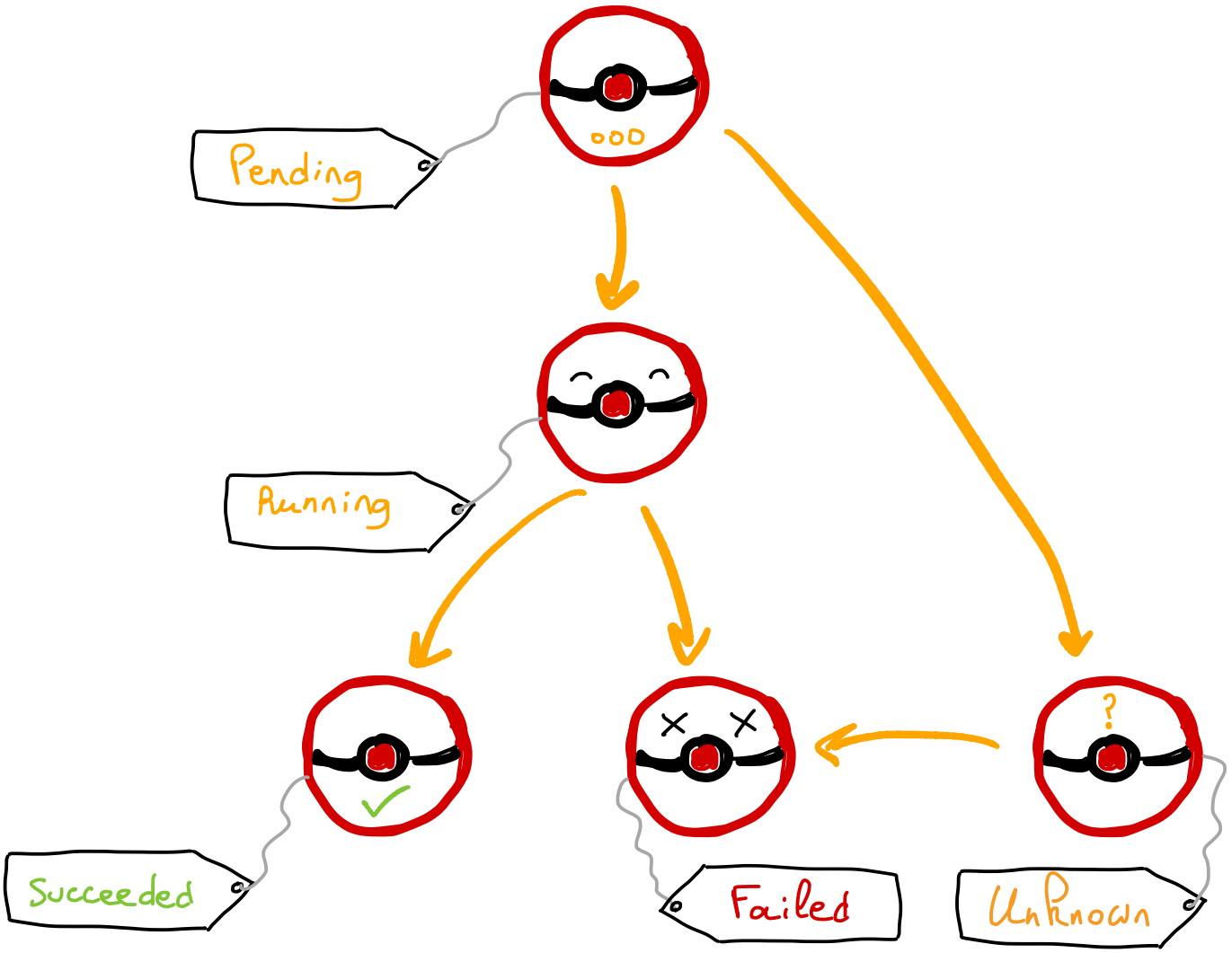
› Copy a file stored locally to a Pod

```
$ kubectl cp myfile.txt my-pod:/path/myfile.txt
```

› List all Pods & in which Nodes they are running

```
$ kubectl get pod -o wide --all-namespaces
```

Pod Lifecycle



Pending

Container images have not yet been created.

Running

Bound to a Node. All containers are created.

At least one is running.

Succeeded

All containers are terminated with success.

Will not be restarted.

Failed

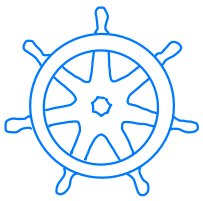
At least one container is in failure (exited with non-zero code or terminated by system).

Unknown

Status of Pod could not be obtained.

Why?

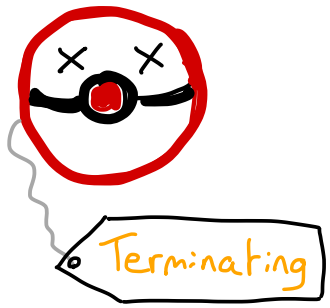
Temporary problem of communication with the host of the Pod.



Pod deletion

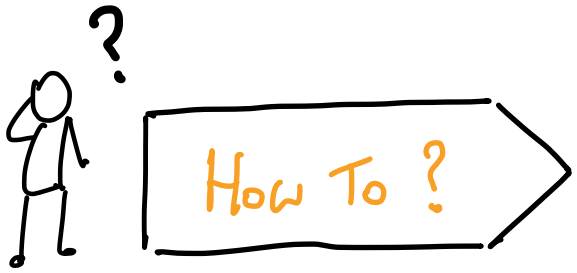
→ A Pod can be deleted in order to ask Kubernetes to start another one (with a new configuration or code for ex...)

→ Sometimes Pods are stucked in Terminating / UnKnown state



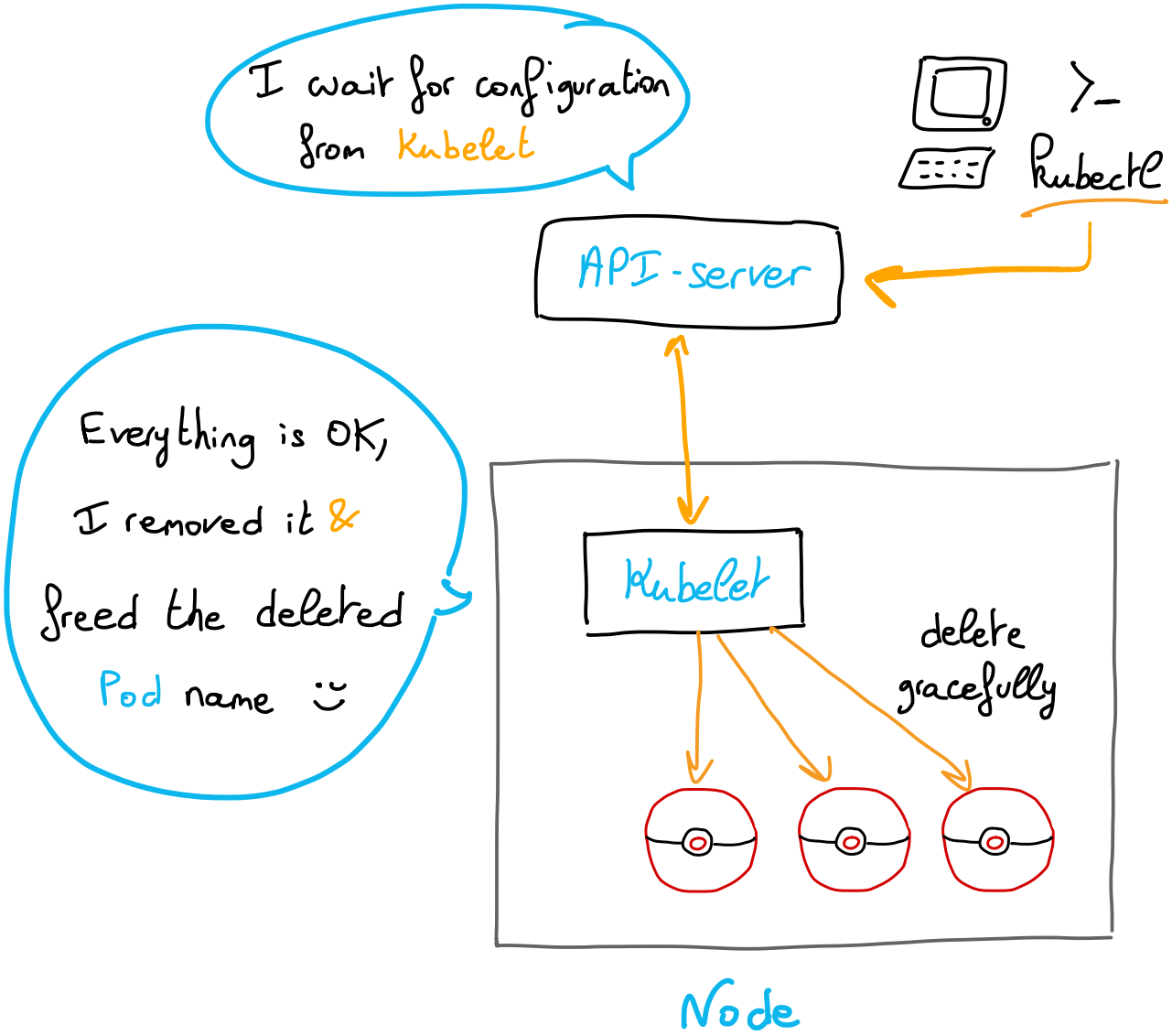
→ The deletion can be manually forced

Manual force deletion should be used carefully



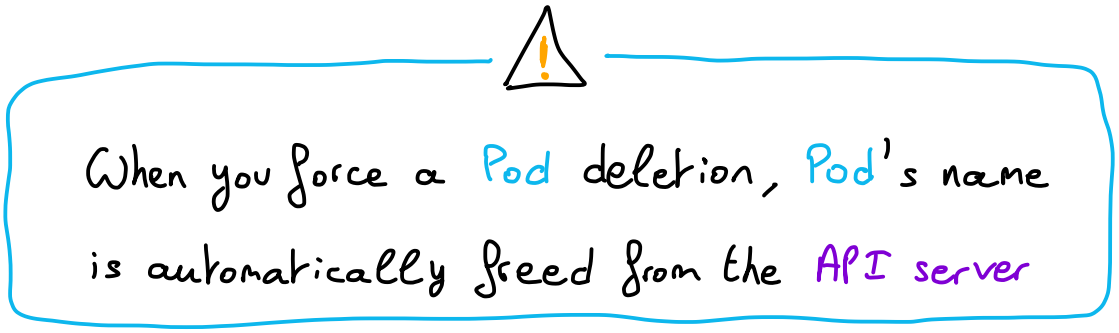
Delete a Pod gracefully

```
$ kubectl delete pod my-pod
```



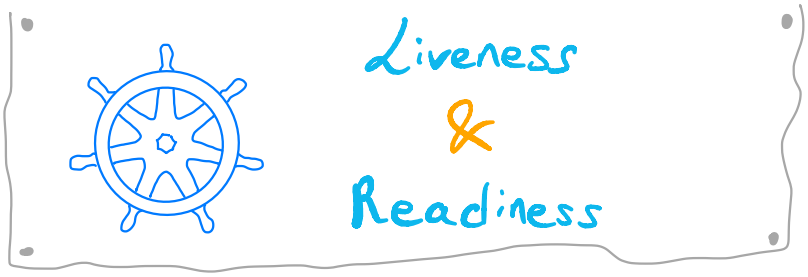
Delete a Pod instantly (manually forced)

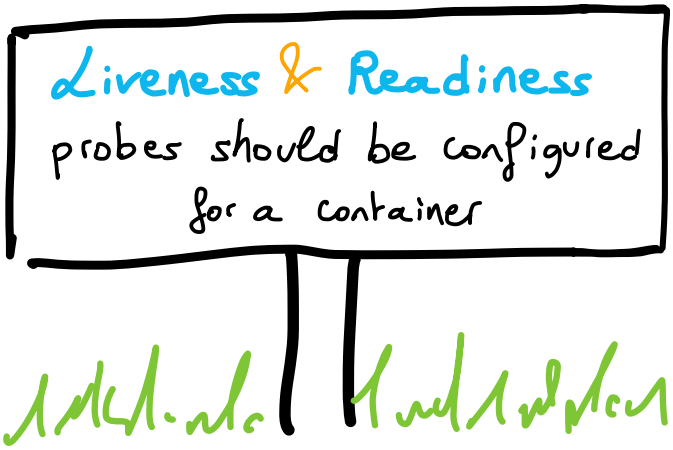
```
$ kubectl delete pod my-pod --grace-period=0 --force
```



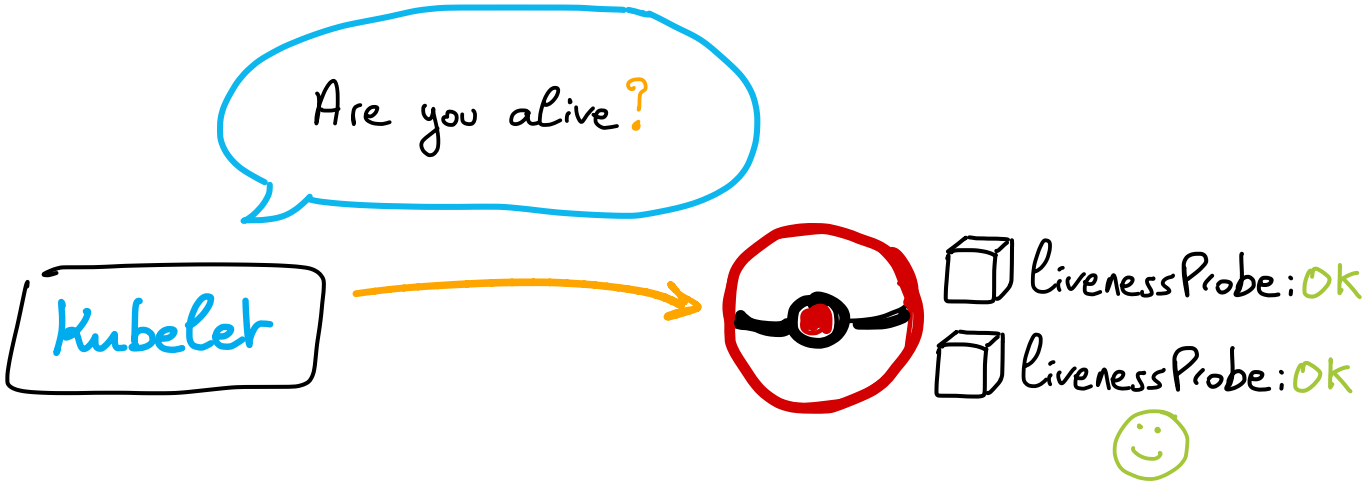
Delete a Pod stuck on Unknown state

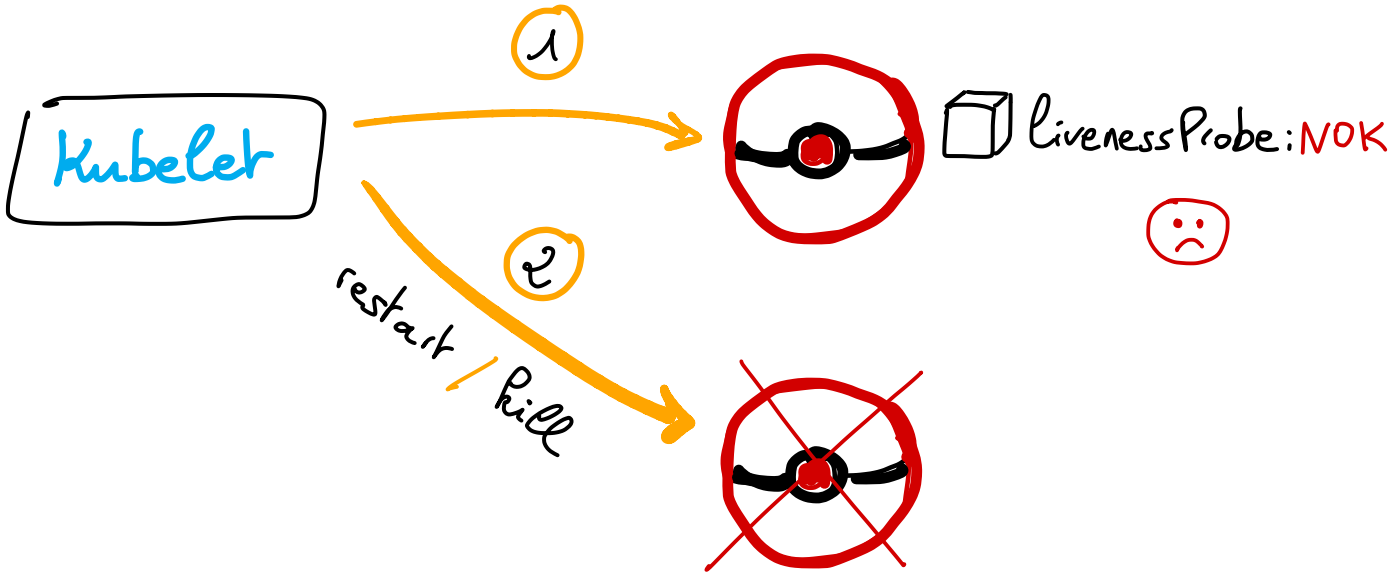
```
$ kubectl patch pod my-pod -p '{"metadata: {"finalizers": null}}'
```


Liveness
&
Readiness


Liveness & Readiness
probes should be configured
for a container


Liveness


Are you alive?
Kubelet
livenessProbe: Ok
livenessProbe: Ok



→ Kubelet uses Liveness probes to know when to restart a container

→ periodSeconds : Ask Kubelet to perform a Liveness probe every xx seconds

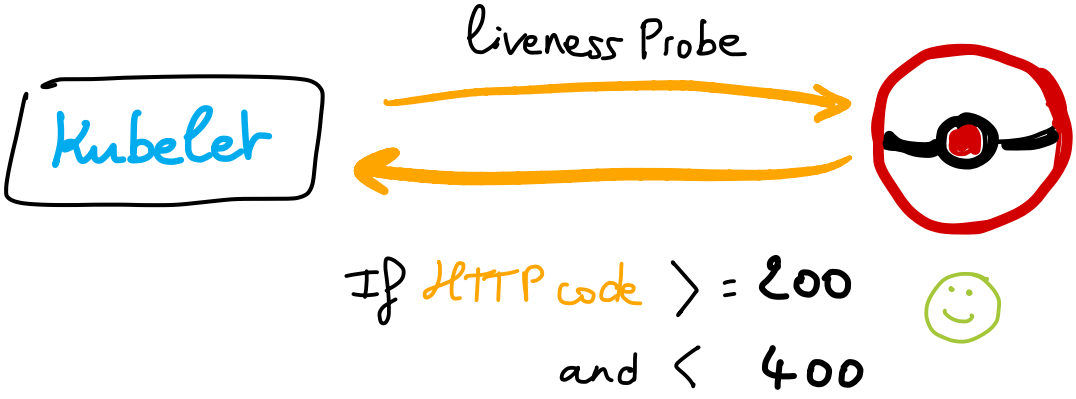
→ initialDelaySeconds : Ask Kubelet to wait xx seconds before to perform the first probe

→ Different types of Liveness probes exist:

- HTTP

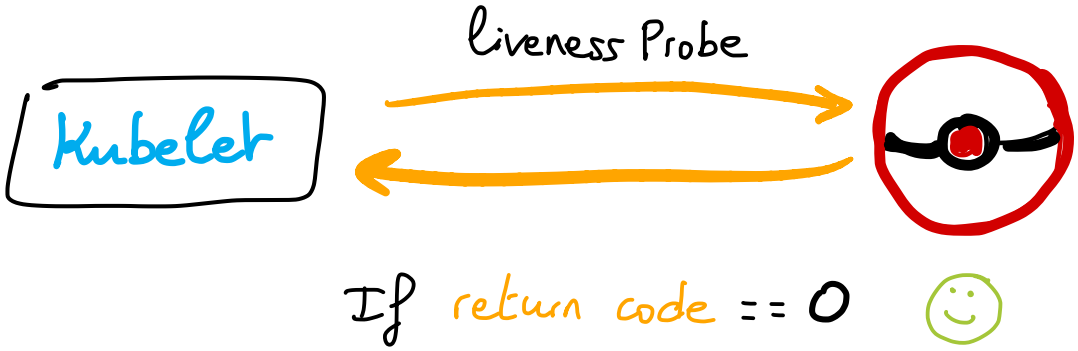
Define a Liveness probe that sends a HTTP request

- on container's server
- on port 8080
- on /healthz




```
apiVersion: v1
kind: Pod
…
spec:
  containers:
  - name: my-container
    image: my-image:1.0
    livenessProbe:
      httpGet:
        path: /healthz
        port: 8080
    initialDelaySeconds: 60
    periodSeconds: 20
```

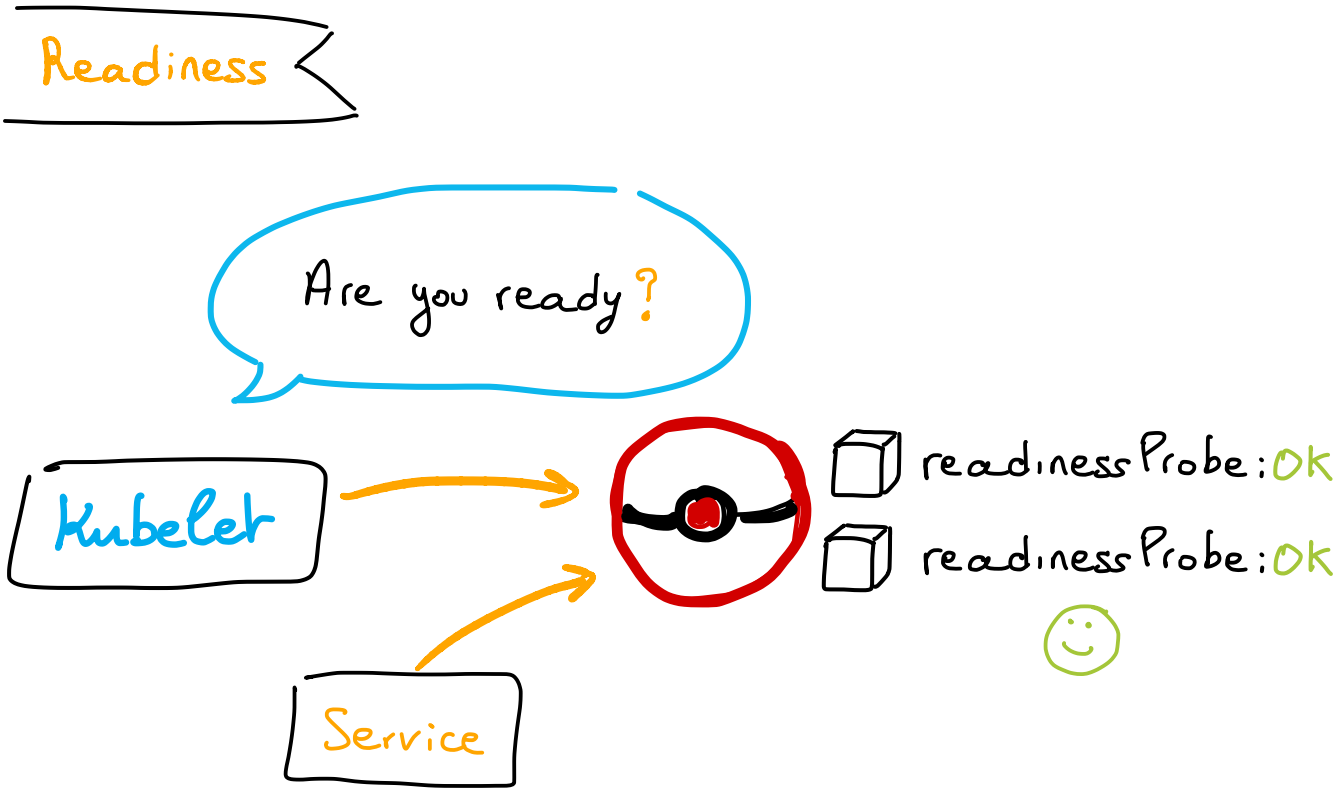
Define a Liveness probe that executes the command **cat /tmp/healthy** in the container

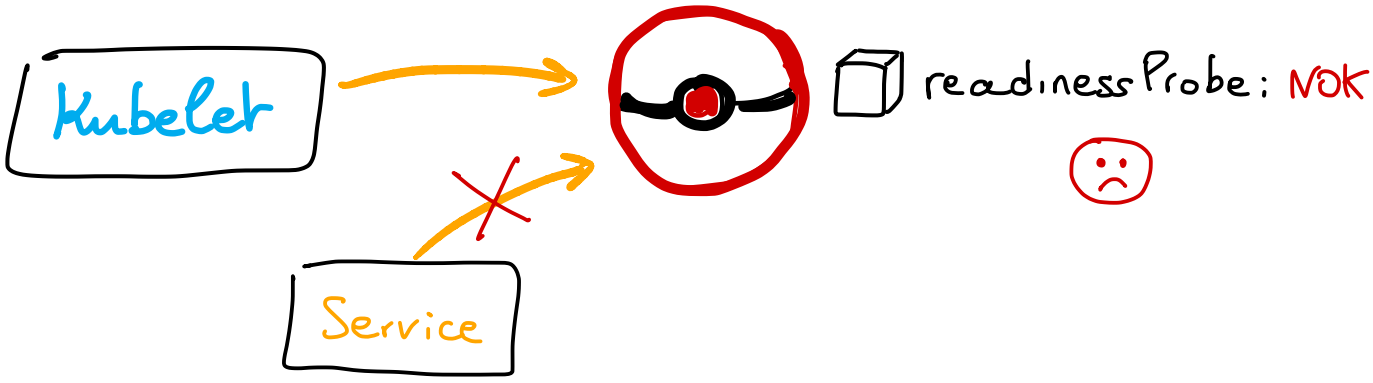



```
apiVersion: v1
kind: Pod
…
spec:
  containers:
  - name: my-container
    image: my-image:1.0
    livenessProbe:
      exec:
        command:
        - cat
        - /tmp/healthy
    initialDelaySeconds: 5
    periodSeconds: 5
```

Define a Liveness probe which connects to port 8080

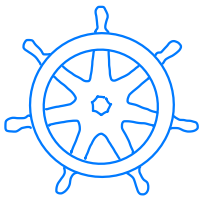
```
apiVersion: v1
kind: Pod
…
spec:
  containers:
  - name: my-container
    image: my-image:1.0
    livenessProbe:
      tcpSocket:
        port: 8080
    initialDelaySeconds: 60
    periodSeconds: 30
```





→ Kubelet uses Readiness probes to know when a container is ready to start accepting traffic. If not, Kubelet can remove the link between Service and Pod.

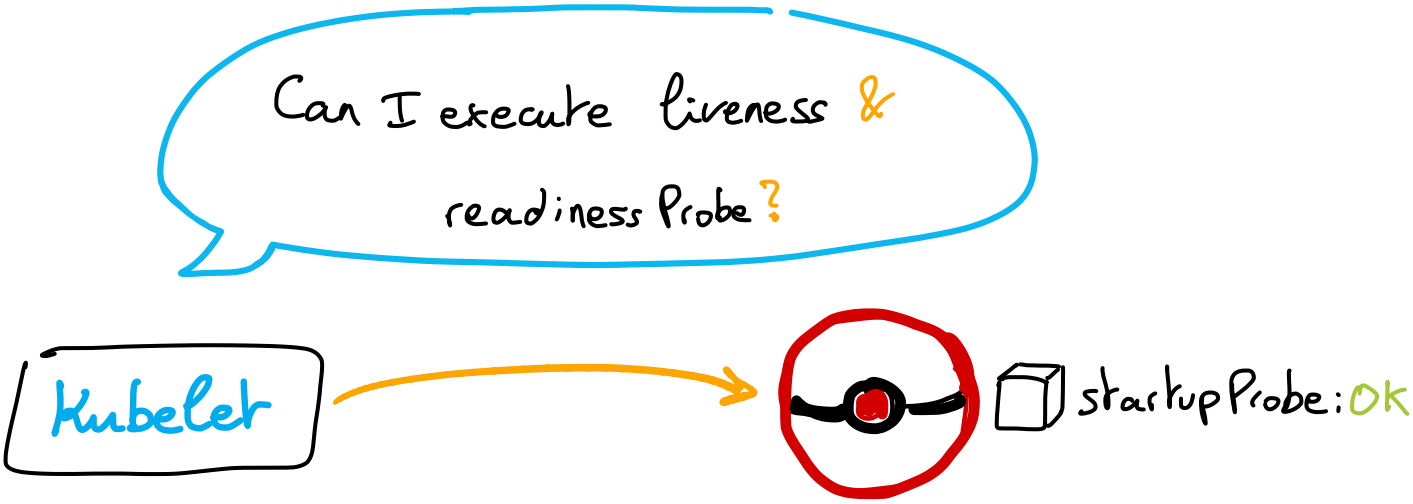
→ ReadinessProbe configuration is quite similar to LivenessProbe ☺



Startup probe

→ Hold off all the other probes until the Pod finishes its startup

→ Give time to a container to finish its startup

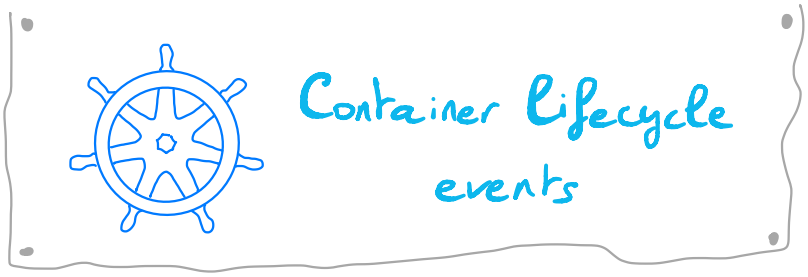


It's a solution for slow-starting Pods.

- Don't test for liveness until HTTP endpoint is available

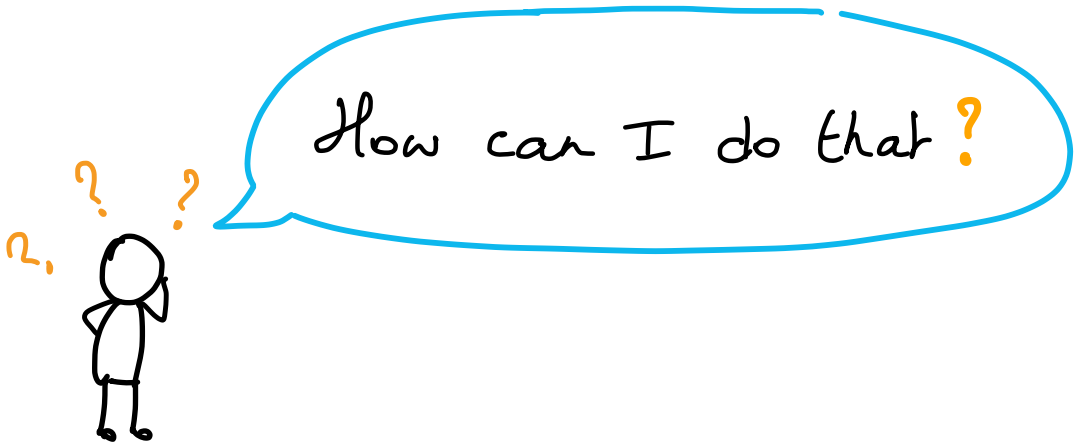
```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
  labels:
    app: my-app
spec:
  containers:
  - name: my-container
    image: my-image:1.0
    ports:
- name: liveness-port
  containerPort: 8080
  livenessProbe:
    httpGet:
      path: /healthz
      port: liveness-port
    failureThreshold: 1
    periodSeconds: 10
  startupProbe:
    httpGet:
      path: /healthz
      port: liveness-port
    failureThreshold: 30
    periodSeconds: 10
```

the app have 5 min (30 x 10s) to finish its startup

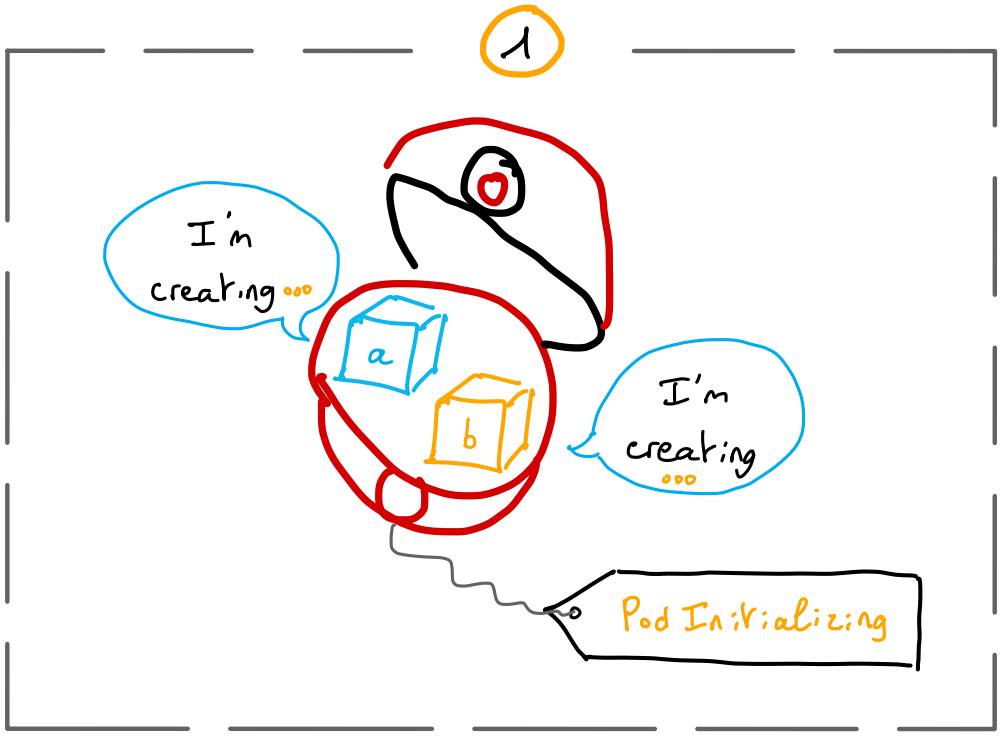


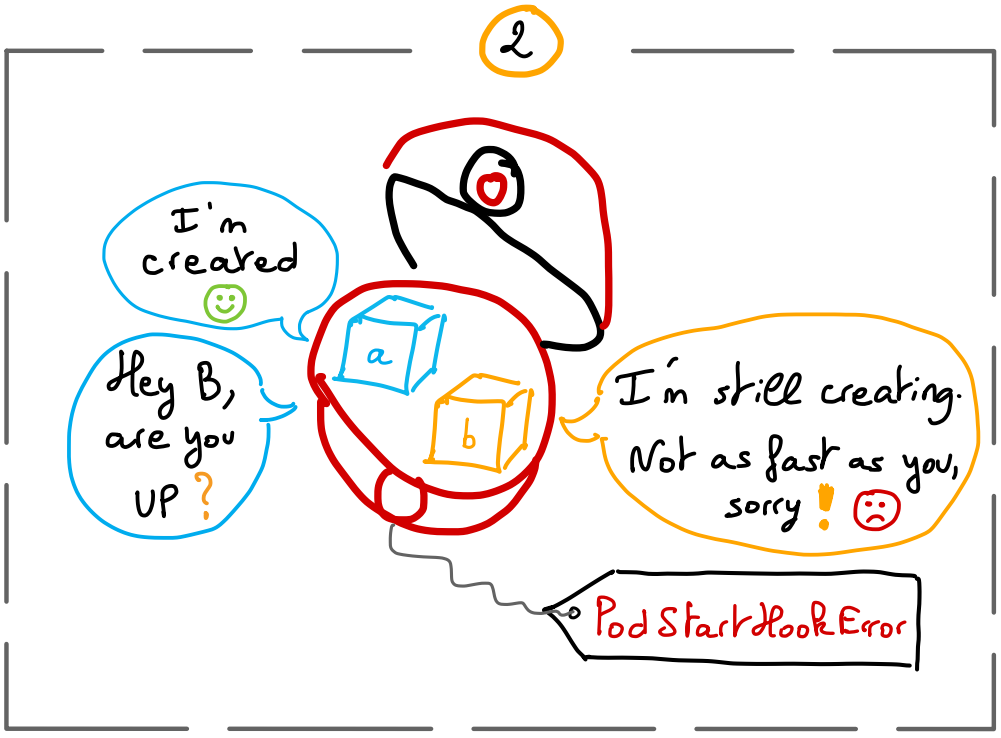
→ Sometimes you need to tell Kubernetes that your Pod should only start when a condition is satisfied

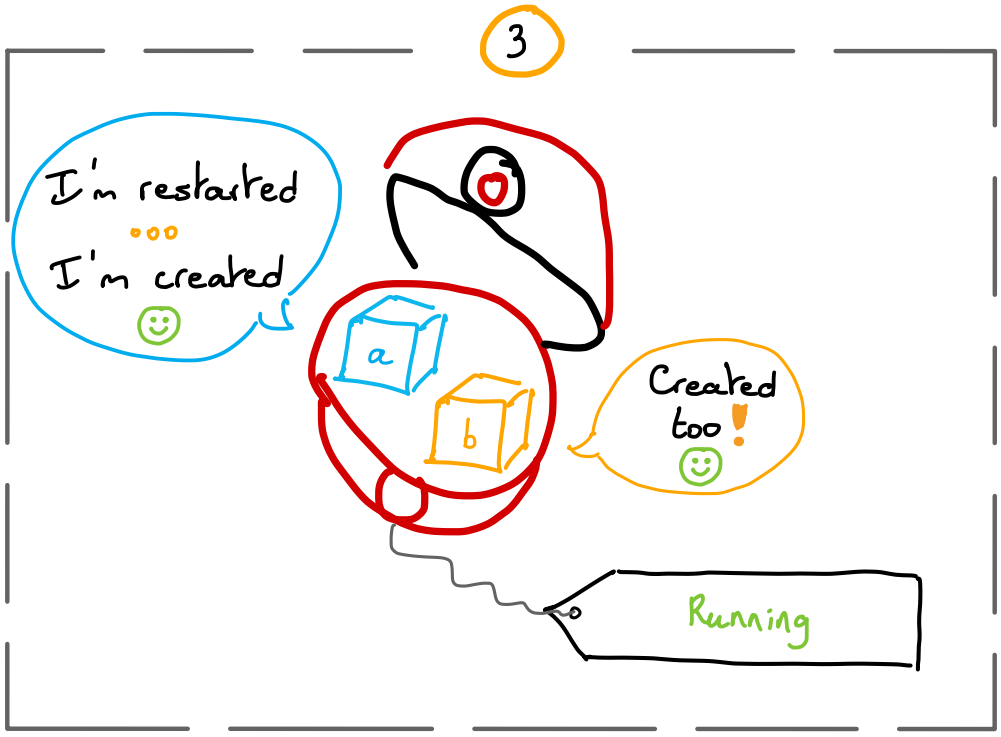
→ Sometimes you want to execute commands before Kubernetes terminates a Pod



Pod Start


1
I'm creating...
a
b
I'm creating...
Pod Initializing


2
I'm created
Hey B, are you UP?
a
b
I'm still creating. Not as fast as you, sorry!
Pod StartHookError


3
I'm restarted
...
I'm created
a
b
Created too!
Running

› Create a Deployment with 3 replicas.

Its Pod will start only when istio-proxy (envoy) is ready

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app
  labels:
    app: my-app
spec:
  replicas: 3
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
      - name: my-app
        image: my-image:latest
        lifecycle:
          postStart:
            httpGet:
              path: /healthz/ready
              port: 15020
```

App container will be restarted as long as the hook postStart fails.

› Create a Deployment with a container that executes a command at the start of the Pod

```
…
    spec:
      containers:
      - name: my-app
        image: my-image:latest
        lifecycle:
          postStart:
            exec:
              command: ["/bin/sh", "-c", "echo Hello
Kubernetes lovers"]
```

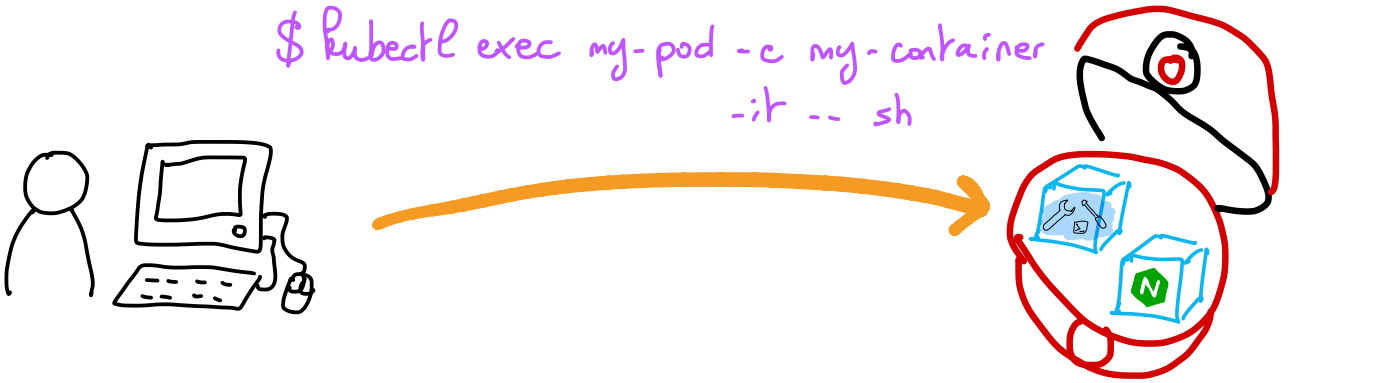
› Create a Deployment that kills gracefully my-app before the Pod terminates

```
…
    spec:
      containers:
      - name: lifecycle-demo-container
        image: nginx
        lifecycle:
          preStop:
            exec:
              command: ["/bin/sh", "-c", "my-app -kill;
do sleep 1; done"]
```

Execute commands in containers

→ kubectl exec command allows you to run commands inside a Pod in a container

→ Useful in order to debug in your container

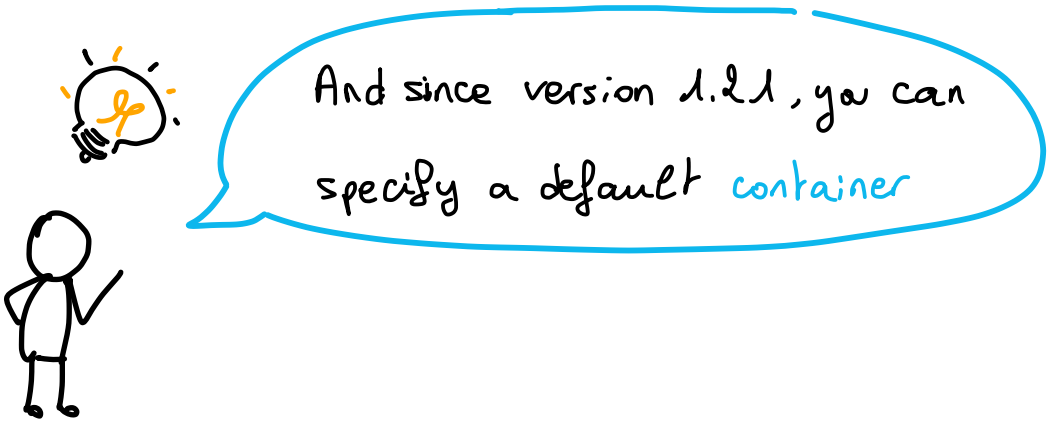


› Connect to a specific container & open an interactive shell

```
$ kubectl exec my-pod -c my-container -it -- sh
```

-it / --stdin --tty : interactive shell

```
> ls -l
> tail -f /var/log/debug.log
```




```yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
  annotations:
    kubectl.kubernetes.io/default-container: my-container-2
spec:
  containers:
  - name: my-container
    image: my-image
  - name: my-container-2
    image: my-image-2
    command: ["/bin/sh", "-c"]
    args:
      - while true; do
          date >> /html/index.html;
          sleep 1;
        done
```

> Connect to my-container-2 container

(no need to specify -c option thanks to the annotation)

```
$ kubectl exec my-pod -it -- sh
```

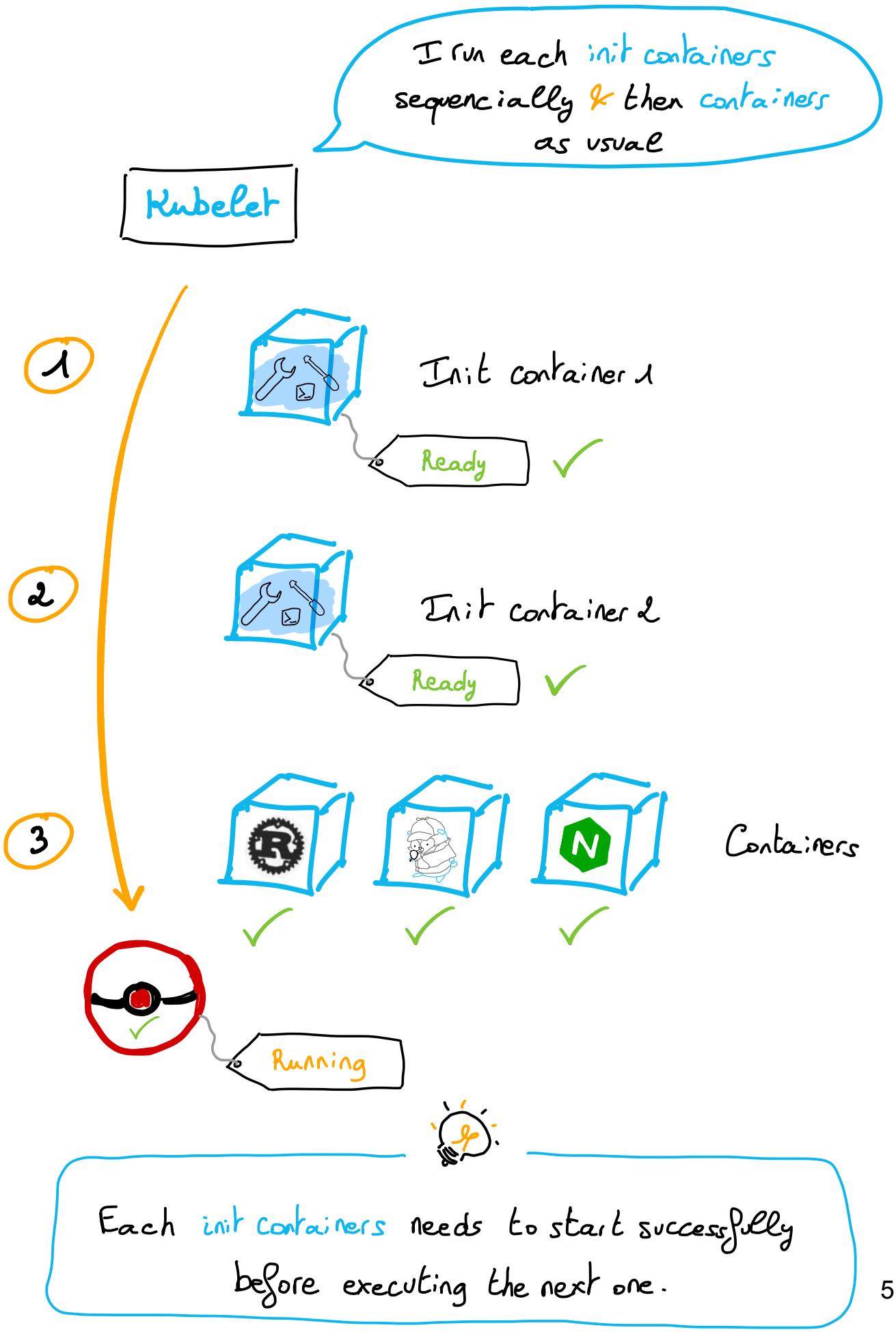
Init Containers

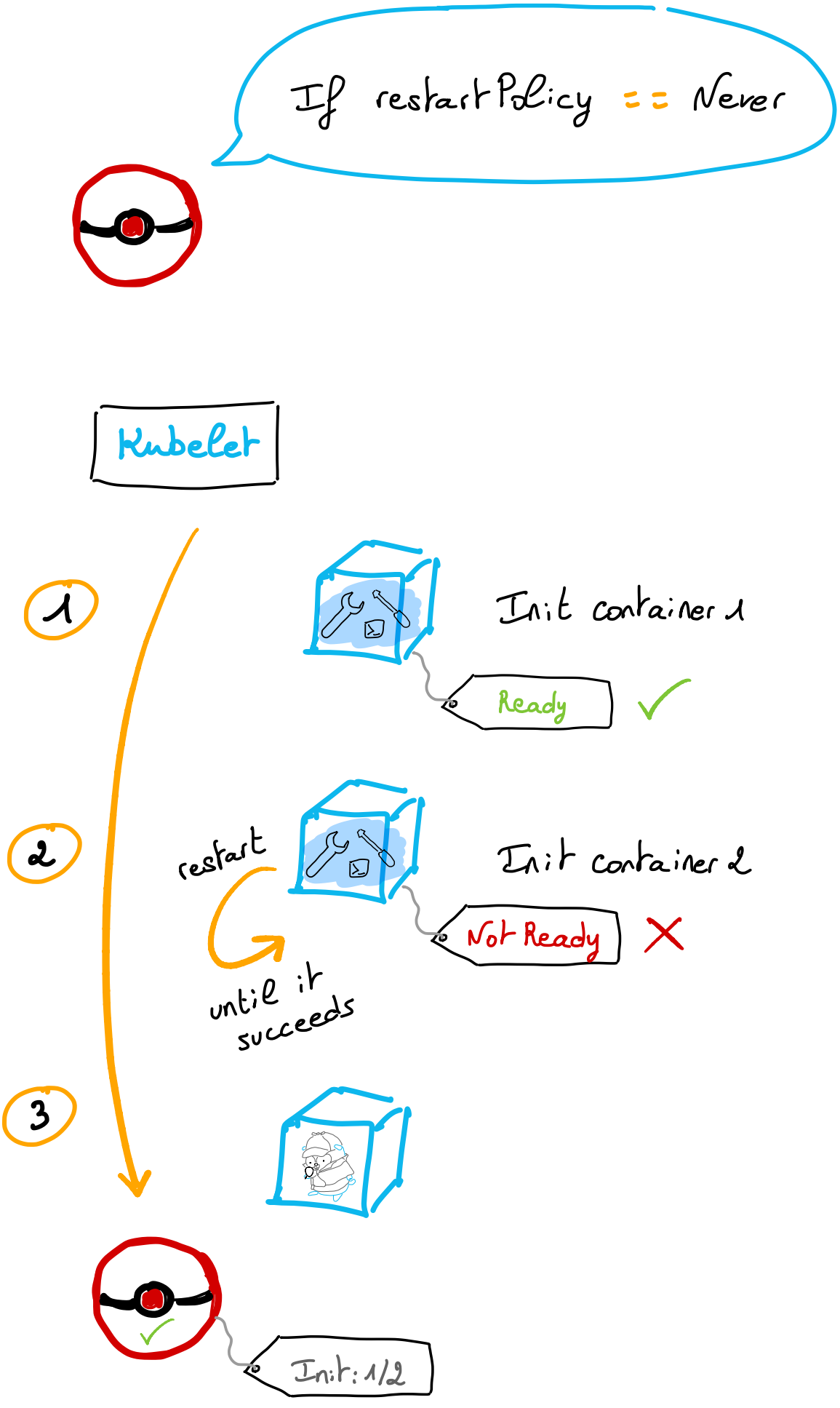
→ Init containers run before app containers in a Pod

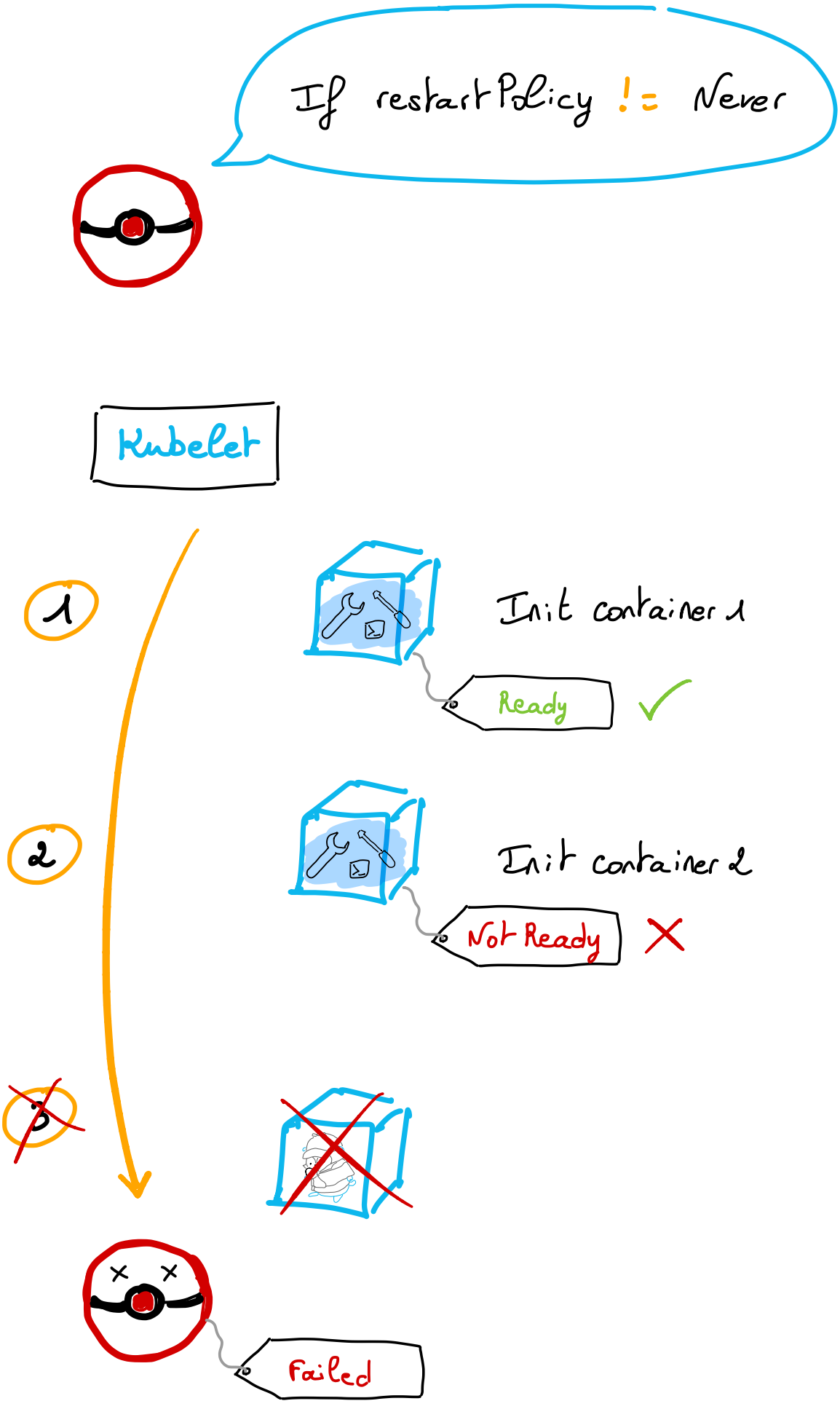
→ Allows you to prepare the main container(s) & separate your app from the initialization logic

→ Containers & Init containers share the same volume

A Pod can have one or more containers & one or more init containers


I run each init containers sequencially & then containers as usual
Kubelet
1
Init container 1
Ready
2
Init container 2
Ready
3
Containers
Running
Each init containers needs to start successfully before executing the next one.


If restartPolicy == Never
Kubelet
1
Init container 1
Ready
2
restart
Init container 2
Not Ready
until it succeeds
3
Init: 1/2


If restartPolicy != Never
Kubelet
1
Init container 1
Ready
2
Init container 2
Not Ready
Failed

→ Useful for :

- Init DB schemas
- Set up permissions
- Clone a Git repository to a shared volume
- Send data to the main app
- Check the accessibility of a Service ✓

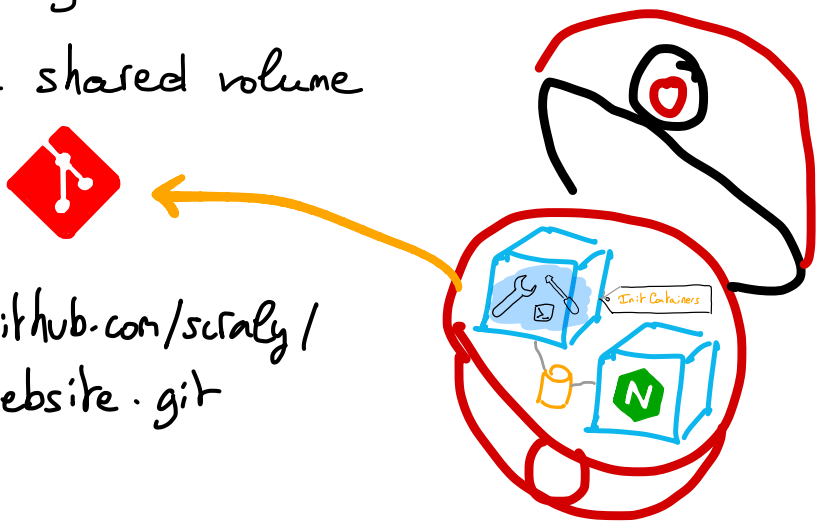
...

> Create a Pod with my-app container & an init container that check a Service is accessible

```
apiVersion: v1
kind: Pod
metadata:
  name: my-app
spec:
  containers:
  - name: my-container
    image: my-app:1.0
    command: ['sh', '-c', 'echo The app is running! && sleep 3600']
  initContainers:
  - name: check-service
    image: busybox
    command: ['sh', '-c', "until nslookup my-svc.$(cat /var/run/
secrets/kubernetes.io/serviceaccount/namespace).svc.cluster.local;
do echo waiting for my-svc; sleep 5; done"]
```

> Create a Pod with:

- an init container that execute a Git clone to /usr/share/nginx/html folder
- a nginx container
- a shared volume

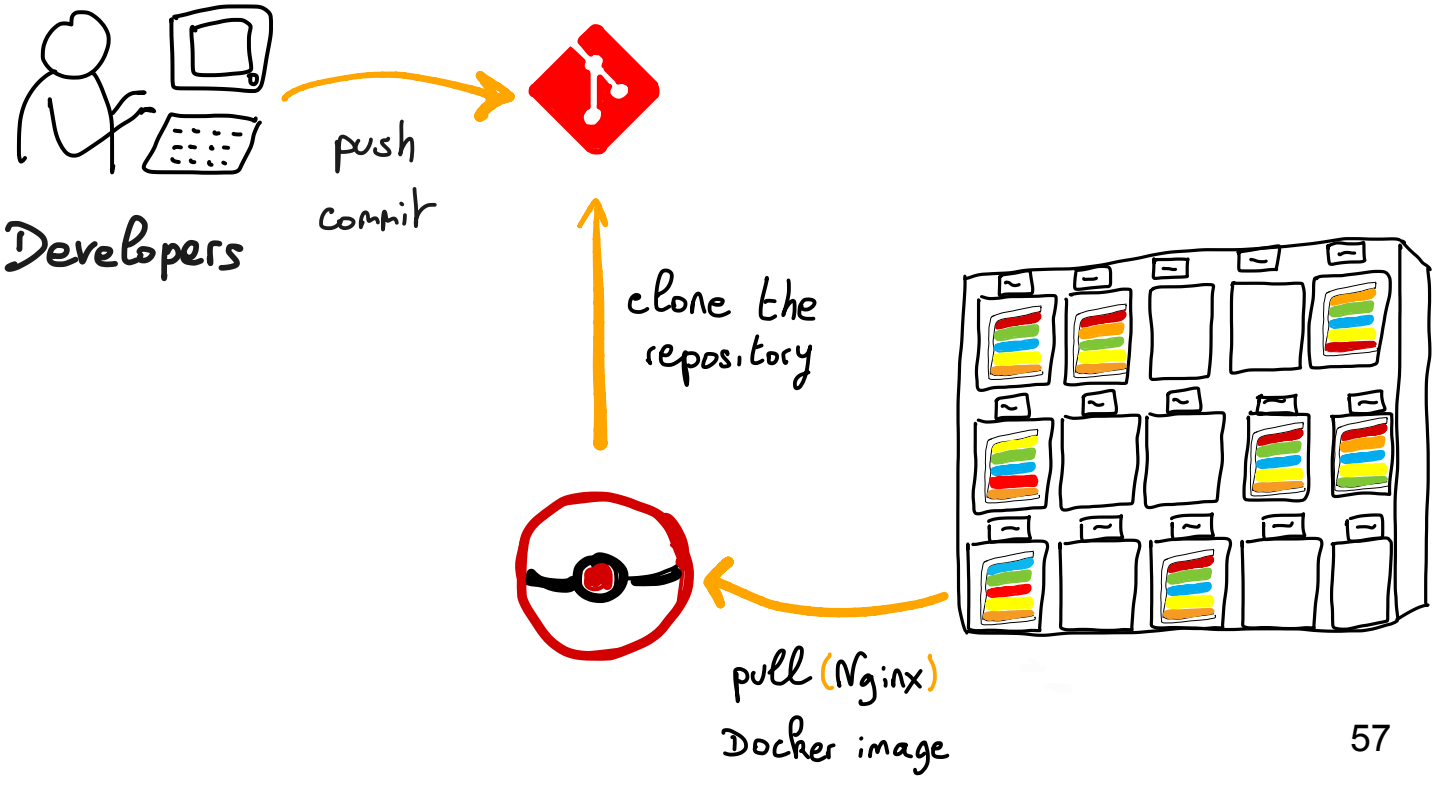


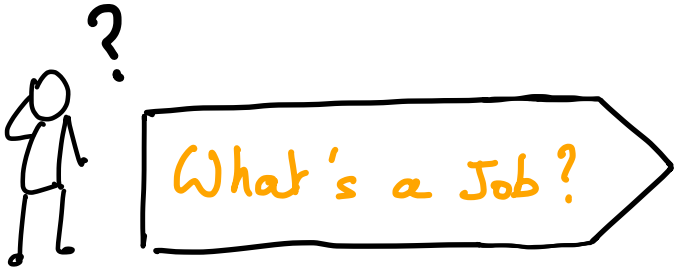
https://github.com/scraly/my-website.git

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-website
spec:
  initContainers:
    - name: clone-repo
      image: alpine/git
      command:
        - git
        - clone
        - --progress
        - https://github.com/scraly/my-website.git
        - /usr/share/nginx/html
      volumeMounts:
        - name: website-content
          mountPath: "/usr/share/nginx/html"
  containers:
    - name: nginx
      image: nginx
      ports:
        - name: http
          containerPort: 80
      volumeMounts:
        - name: website-content
          mountPath: "/usr/share/nginx/html"
  volumes:
    - name: website-content
      emptyDir: {}
```

Even if hosting/running a website/blog in Kubernetes can be overkill, if you want to, as you can see, a solution can be to split static content into webserver.
Thanks to that you no longer have to put your website content inside a Docker image.

A Pod is mortal, so everytime a new one is starting, it will have a fresh content of your Git repository retrieved by the init container.





> A process that runs a certain time to completion:

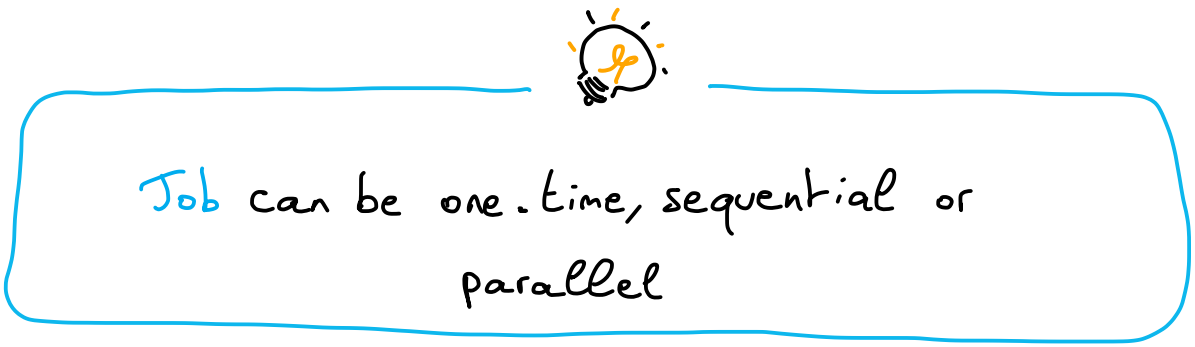
- ☑ batch process
- ☑ backup
- ☑ database migration / clean

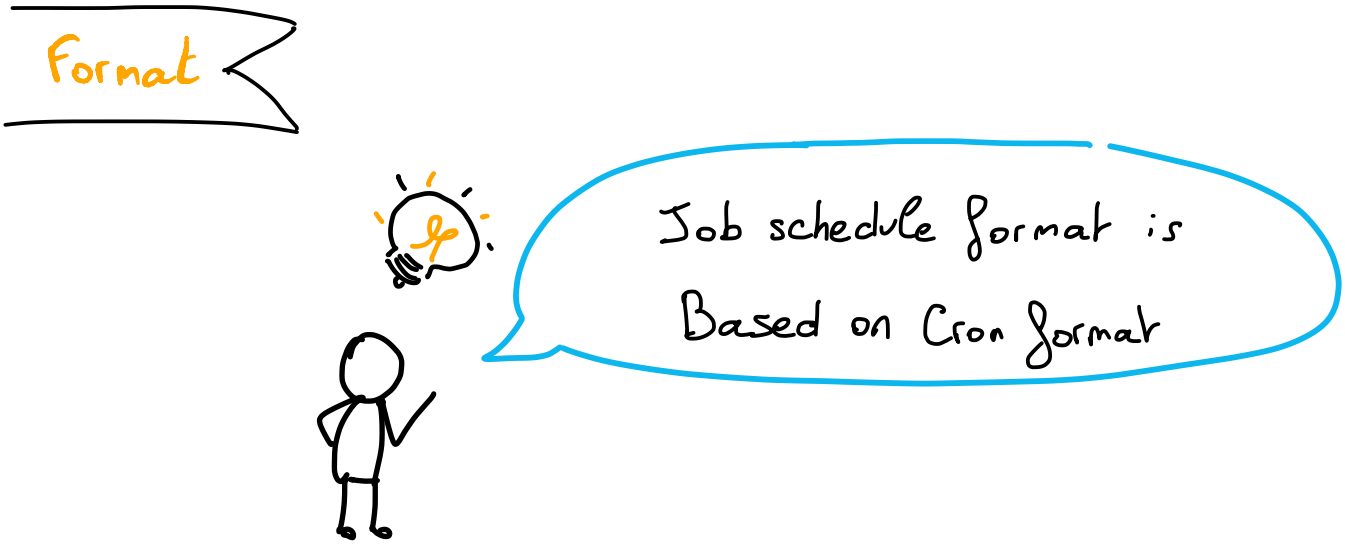
> A Job runs one or more Pods and ensures a specified number of them successfully terminate

› When the Pod launched by a Job is finished, its status will be Completed.

If the Pod failed, another Pod will run until number of completions is done.

› Job can be launched by a Cron Job






```
$ kubectl get cronjob
NAME        SCHEDULE       SUSPEND   ACTIVE   LAST SCHEDULE   AGE
my-cj       0 5 */1 * *    False     0        3h53m           10s
```

Create a Job from busybox image (Imperative way)

```
$ kubectl create job my-job --image=busybox
```

Create a Job from busybox image (Declarative way)

```
apiVersion: batch/v1
kind: Job
metadata:
  name: my-job
spec:
  backoffLimit: 5
  activeDeadlineSeconds: 120
  completions: 1
  parallelism: 1
    template:
      spec:
        containers:
        - name: busybox
          image: busybox
        restartPolicy: onFailure
```

Optional

Required

Create a Job from a CronJob

```
$ kubectl create job my-job --from=cronjob/my-cronjob
```

› Create a Job that will be deleted automatically after the time defined

1.23

```
apiVersion: batch/v1
kind: Job
metadata:
  name: my-job-with-ttl
spec:
  ttlSecondsAfterFinished: 100
  template:
    metadata:
      name: my-job-with-ttl
    spec:
      containers:
      - name: busybox
        image: busybox
```

) TTL = Time To Leave

› Delete a Job (and all its childs Pods)

```
$ kubectl delete job my-job
```

› Delete a Job only

```
$ kubectl delete job my-job cascade=false
```

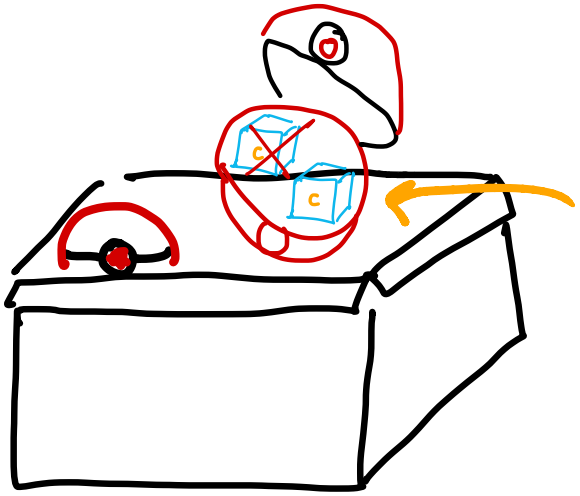
restart Policy

Never

If the Pod failed, Job controller starts a new Pod.

On Failure

If the Pod failed (for any reason), the Pod stays on the Node but containers will re-run.



run a new container

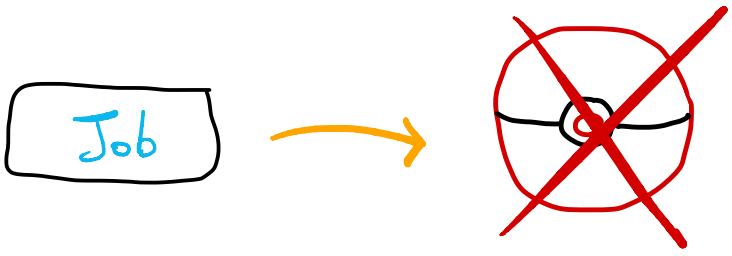
restartPolicy is applied to a Pod (not a Job)
and cannot be set to Always.
When a Pod terminates with success,
a Job will never re-run / restart a Pod.

backoffLimit

Number of retries before considering a Job as failed.
By default backoffLimit is equal to 6.

activeDeadlineSeconds

Once a Job reaches the number of activeDeadlineSeconds,
all of running Pods will be killed :



status : DeadlineExceeded

A Job will not deploy additional Pods when activeDeadlineSeconds is reached, even if the backoffLimit is not yet reached !!

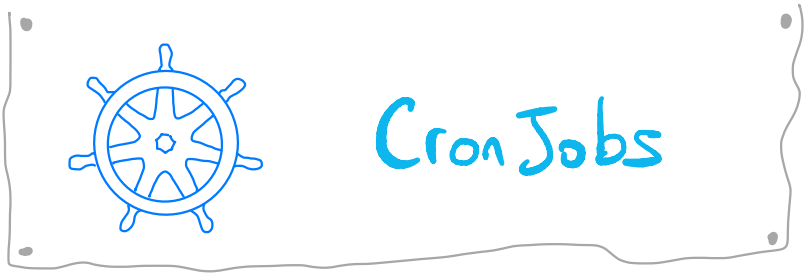
completions

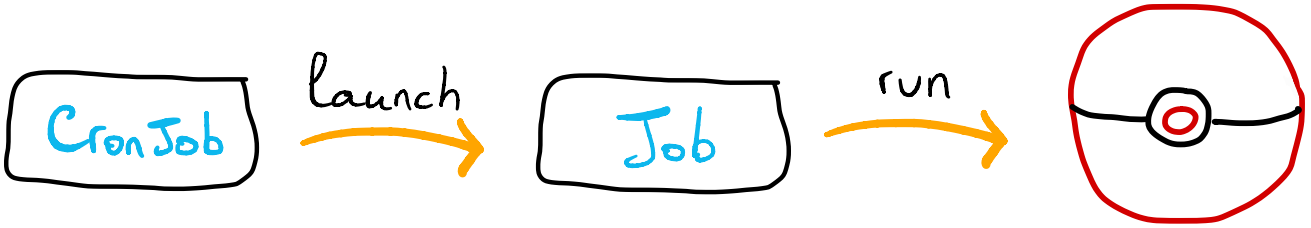
By default completions is equal to 1, which means the normal behavior for a Job is to run Pods until one Pod terminates successfully.
If we set completions equals to 5 for ex., Job controller will spawn Pods until the number of complete Pods will reach this value.

parallelism

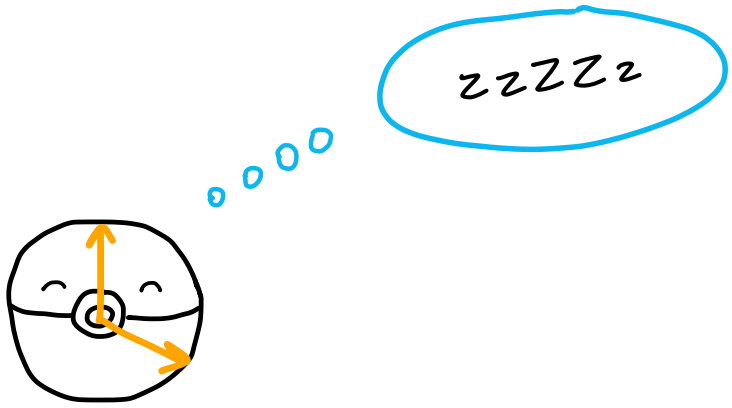
parallelism allows you to run multiple Pods at the same time.
By default parallelism is set to 1.

If completions is not defined, it is equal to the parallelism number.

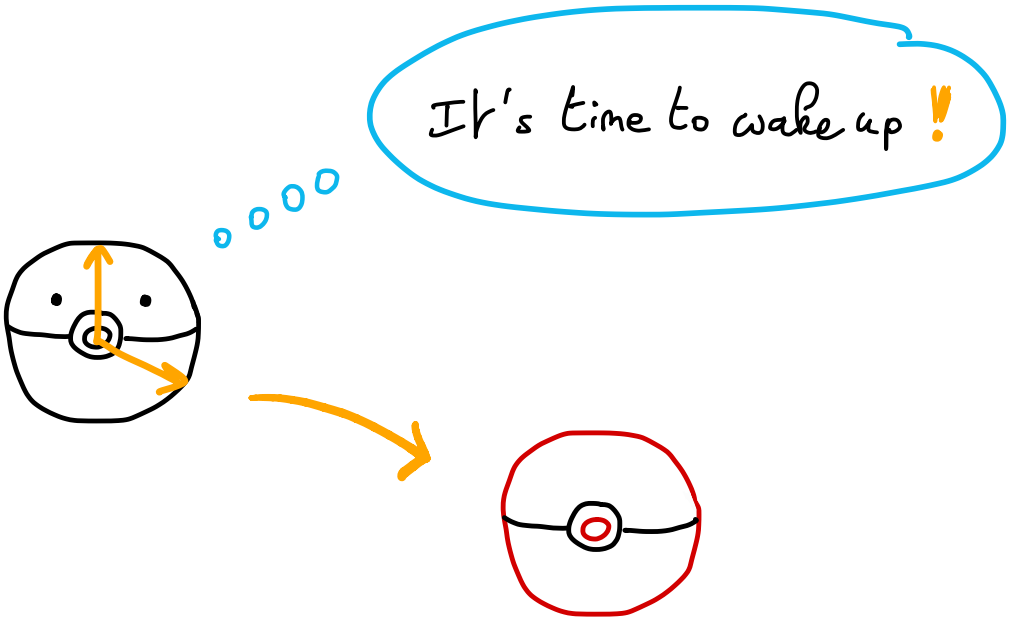




→ Creates Job on a time-based schedule

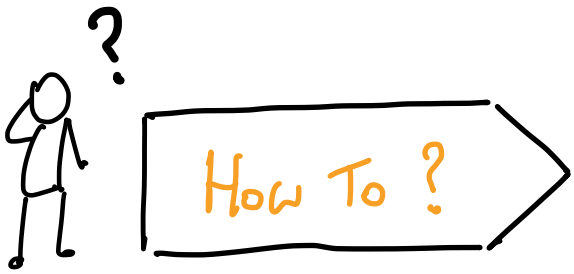


→ Or periodically



→ Based on Cron format

```
$ kubectl get cronjob
NAME        SCHEDULE       SUSPEND   ACTIVE   LAST SCHEDULE   AGE
my-cj       0 5 */1 * *      False       0         3h53m        10s
```



› Create a CronJob from busybox image (Declarative way)

```
apiVersion: batch/v1
kind: CronJob
metadata:
  name: hello
spec:
  schedule: "*/1 * * * *"
  jobTemplate:
    spec:
      template:
        spec:
          failedHistoryLimit: 3 } number of failed Jobs
          successfulJobHistoryLimit: 1 } number of successful Job
          containers:
          - name: hello
            image: busybox
            args:
            - /bin/sh
            - -c
            - date; echo Hello readers
          restartPolicy: onFailure
```

If startingDeadlineSeconds is < than 10 seconds, CronJob may not been scheduled !

Create a CronJob from busybox image & run every 5 minutes

```
$ kubectl create cronjob my-cj --image=busybox
--schedule="*/5 * * * *" -n my-namespace
```

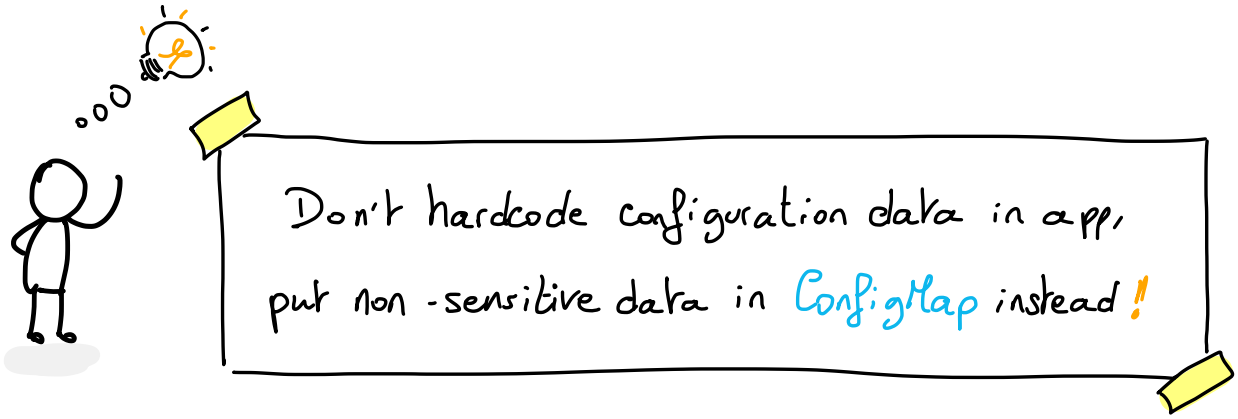
Change CronJob scheduling

```
$ kubectl patch cronjob my-cronjob
-p '{"spec":{"schedule": "0 */1 */1 * *"}}'
```

Launch a Job from an existing CronJob

```
$ kubectl create job --from=cronjob/my-cronjob my-job
-n my-namespace
```

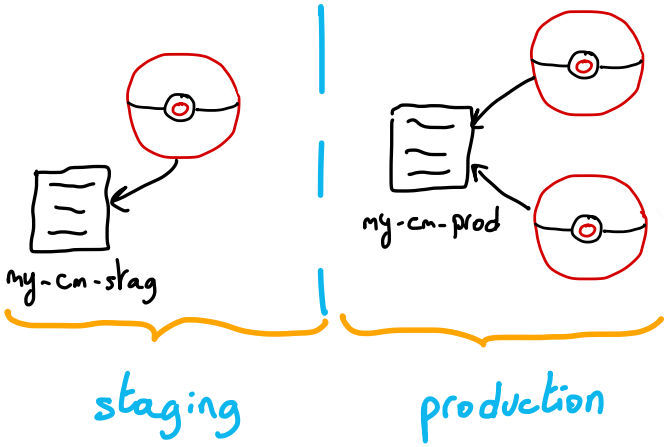
ConfigMaps



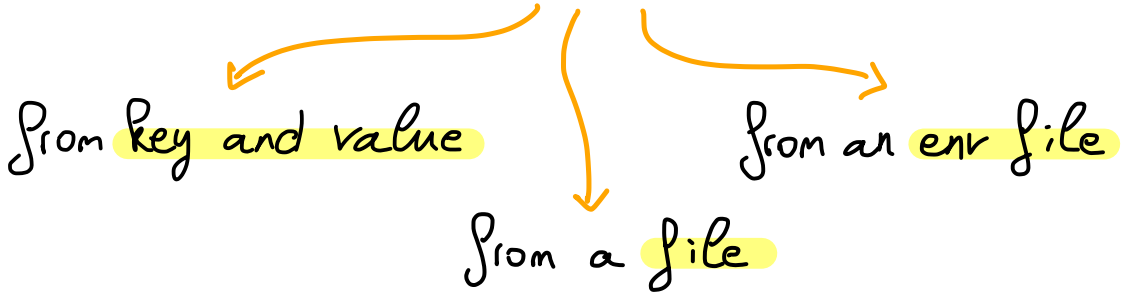
Twelve-Factor App compliant

→ Aim : separate configuration from Pods

→ Allows to deploy a same app in different environments



→ 3 ways to create a ConfigMap



Create a ConfigMap from key and value

```
$ kubectl create cm my-cm-1 --from-literal=my-key=my-value
-n my-namespace
```

Create a ConfigMap from a file

```
$ kubectl create cm my-cm-1 --from-file=my-file.txt
-n my-namespace
```

Create a ConfigMap from an env file

```
$ kubectl create cm my-cm-1 --from-env-file=my-envfile.txt
-n my-namespace
```

Create a Pod with ConfigMap mounted as a volume

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  containers:
  - name: my-container
    image: busybox
    volumeMounts:
    - name: my-cm-volume
      mountPath: /etc/myfile.txt
  volumes:
  - name: my-cm-volume
    configMap:
      name: config
```

› Create a Pod and load the value from variable my-key in an env variable MY-ENV-KEY

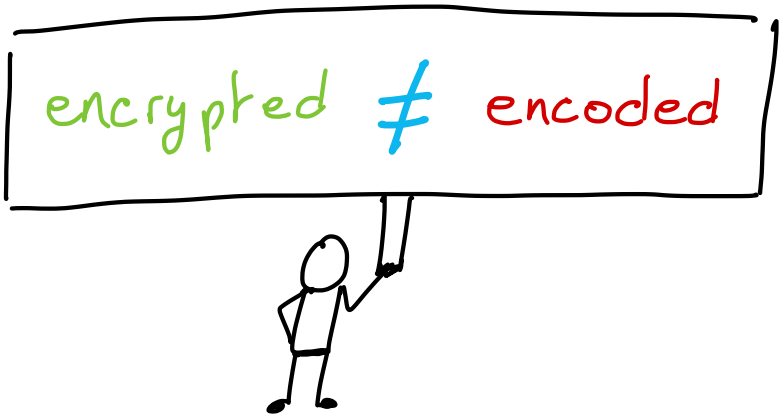
```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod-2
spec:
  containers:
    - name: my-container
      image: busybox
      env:
        - name: MY-ENV-KEY
          valueFrom:
            configMapKeyRef:
              name: my-cm
              key: my-key
```

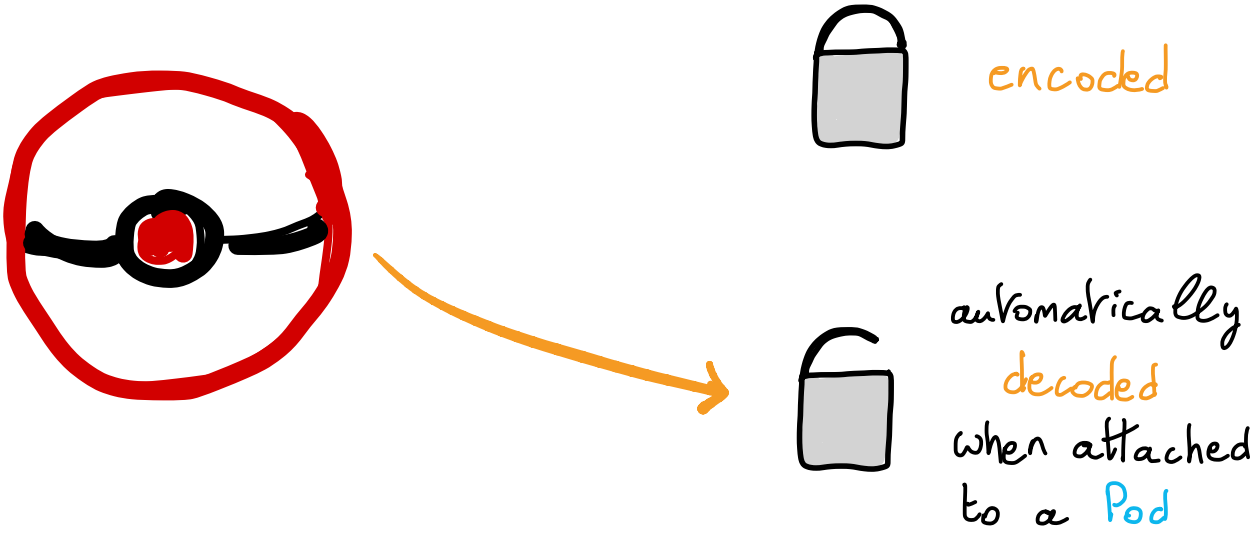
› Create a Pod & fill env vars from a ConfigMap

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod-3
spec:
  containers:
    - name: my-container
      image: busybox
      envFrom:
        - configMapRef:
          name: my-cm
```

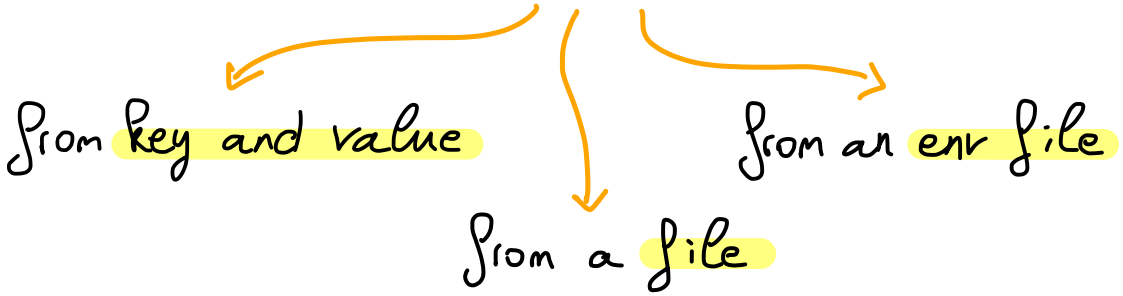
Secrets

- Save sensitive data in secrets
- Base 64 encoded "at rest"





→ 3 ways to create a Secret :



→ Several types of Secret

- generic
- docker-registry
- tls

- Create a Secret from key and value

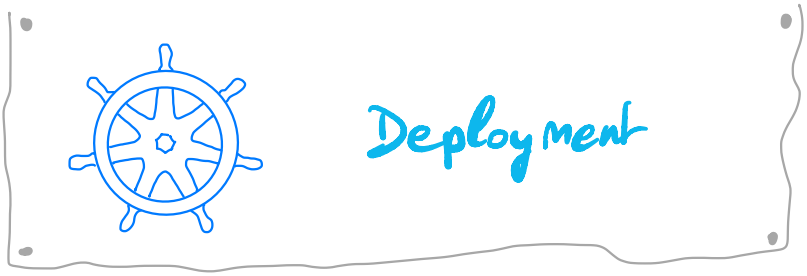
```
$ kubectl create secret generic my-secret
--from-literal=password='my-awesome-password'
```

- Create a Secret from a file in my-namespace

```
$ kubectl create secret generic my-secret
--from-file=password.txt -n my-namespace
```

- Retrieve a Secret, open it, extract desired data & decode it

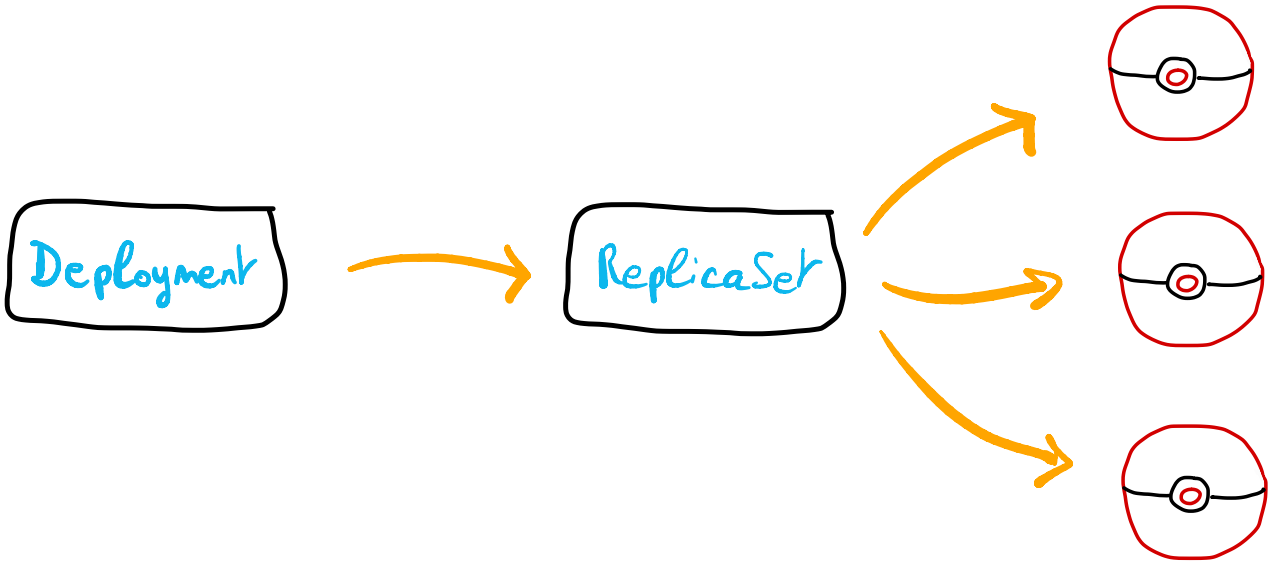
```
$ kubectl get secret my-ca-secret
-o=go-template='{{index .data "ca.crt" | base64decode}}'
```



→ Deployment handles Pod creation

→ Deployment are responsible for Pods
& must ensure they are running

→ Deployment creates a ReplicaSet that creates Pod
according to the number of specified replicas



→ Features:

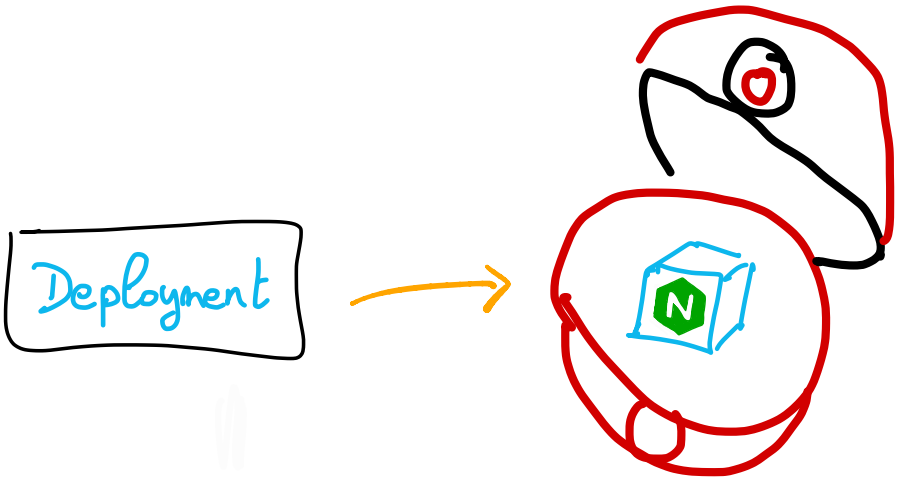
- ☑ Rolling update / Rollout
- ☑ Deployment history

› Create a Deployment that creates one Pod with busybox image (Imperative way)

```
$ kubectl create deploy my-deploy --image=nginx
```

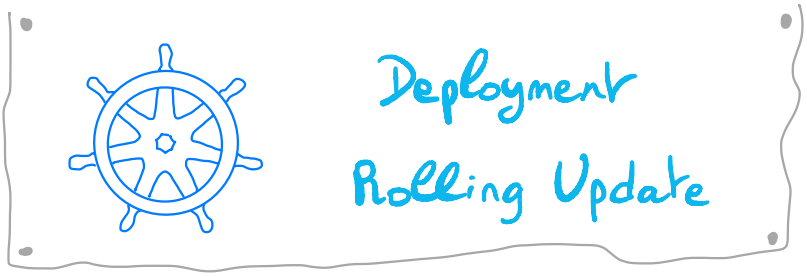
› Create a Deployment that creates 3 replicas of Pod image (Declarative way)

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-deploy
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: my-deploy
  template:
    metadata:
      labels:
        app: my-deploy
    spec:
      containers:
      - name: nginx
        image: nginx
```



Scale a Deployment to 5 replicas

```
$ kubectl scale deploy my-deploy --replicas=5
```



- Rolling Update allows zero-downtime during Deployment update
- Incrementally update instances by creating new ones
- Updates are versioned & any update can be reverted to a previous version

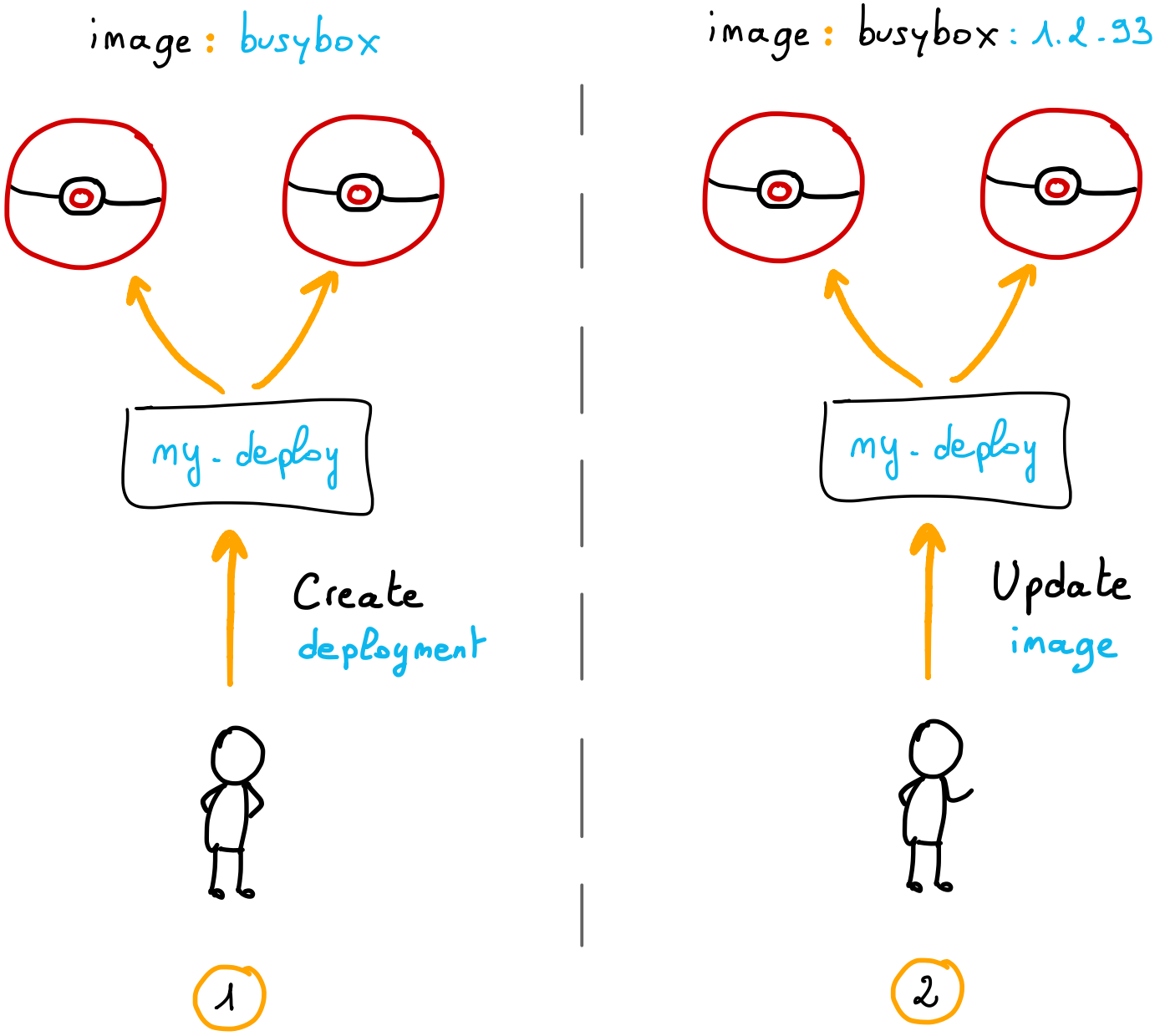


image : busybox
my-deploy
Create deployment
1
image : busybox : 1.2-93
my-deploy
Update image
2

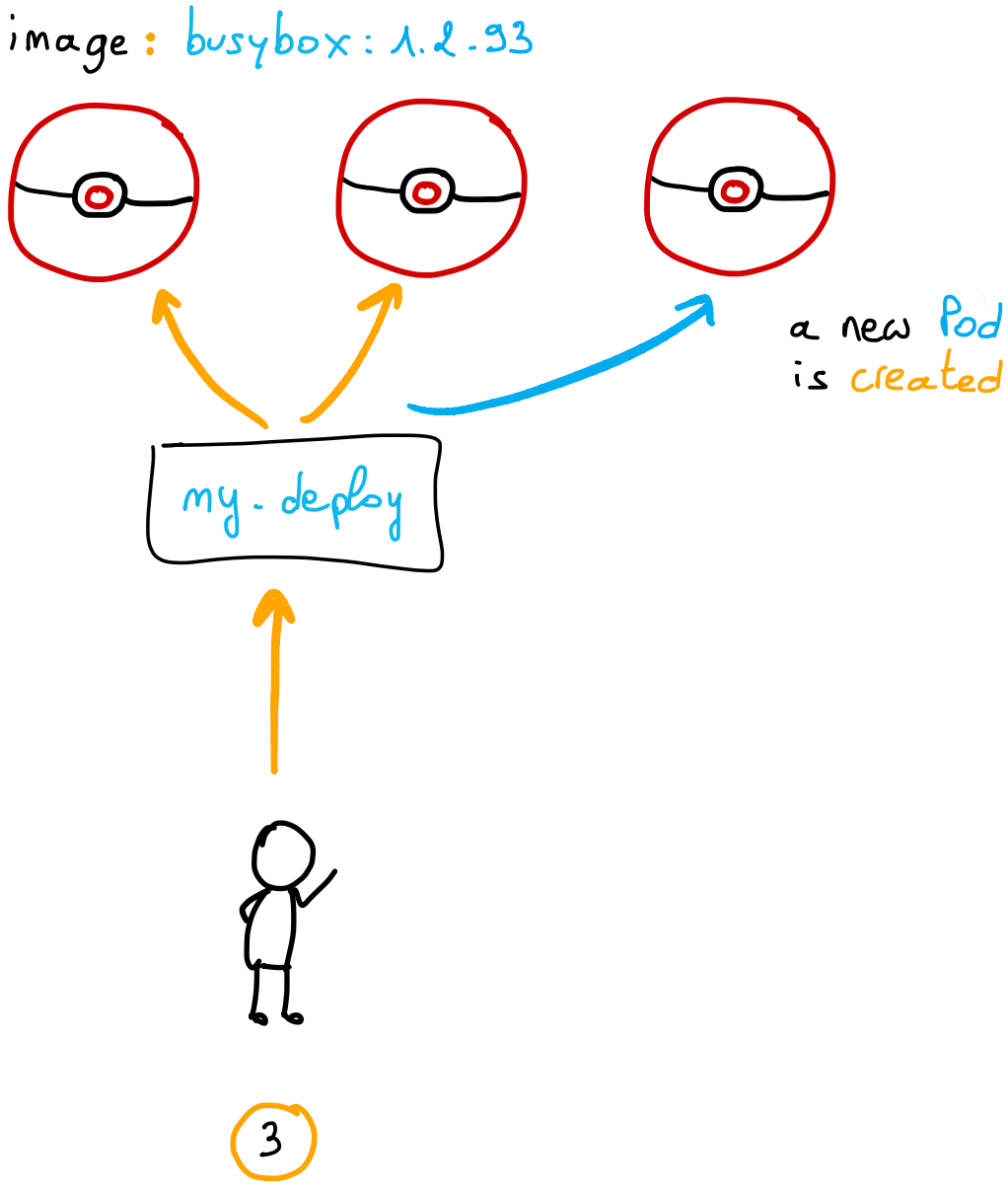


image : busybox : 1.2.93
a new Pod
is created
my-deploy
3

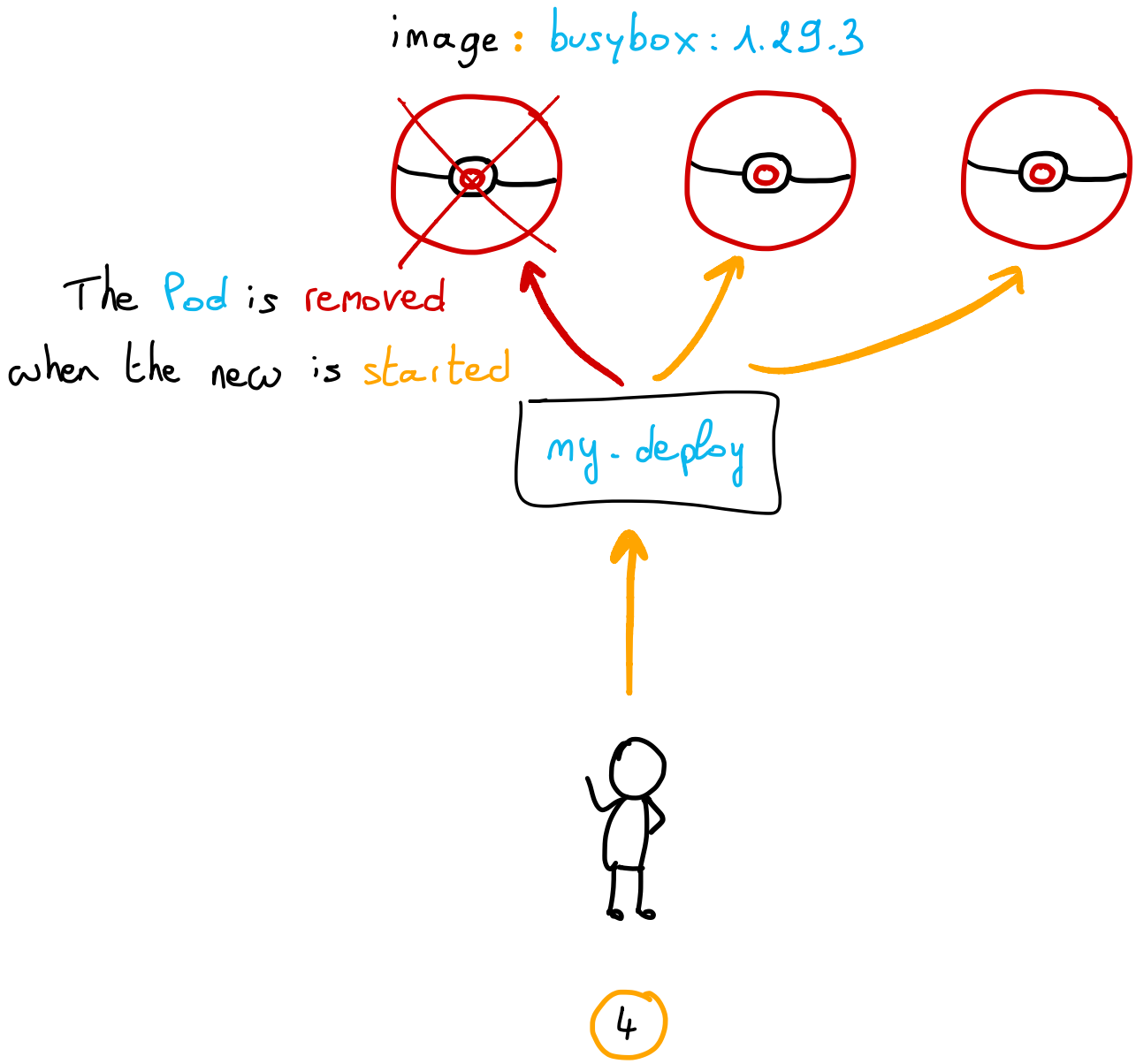


image : busybox : 1.29.3
The Pod is removed
when the new is started
my-deploy
4

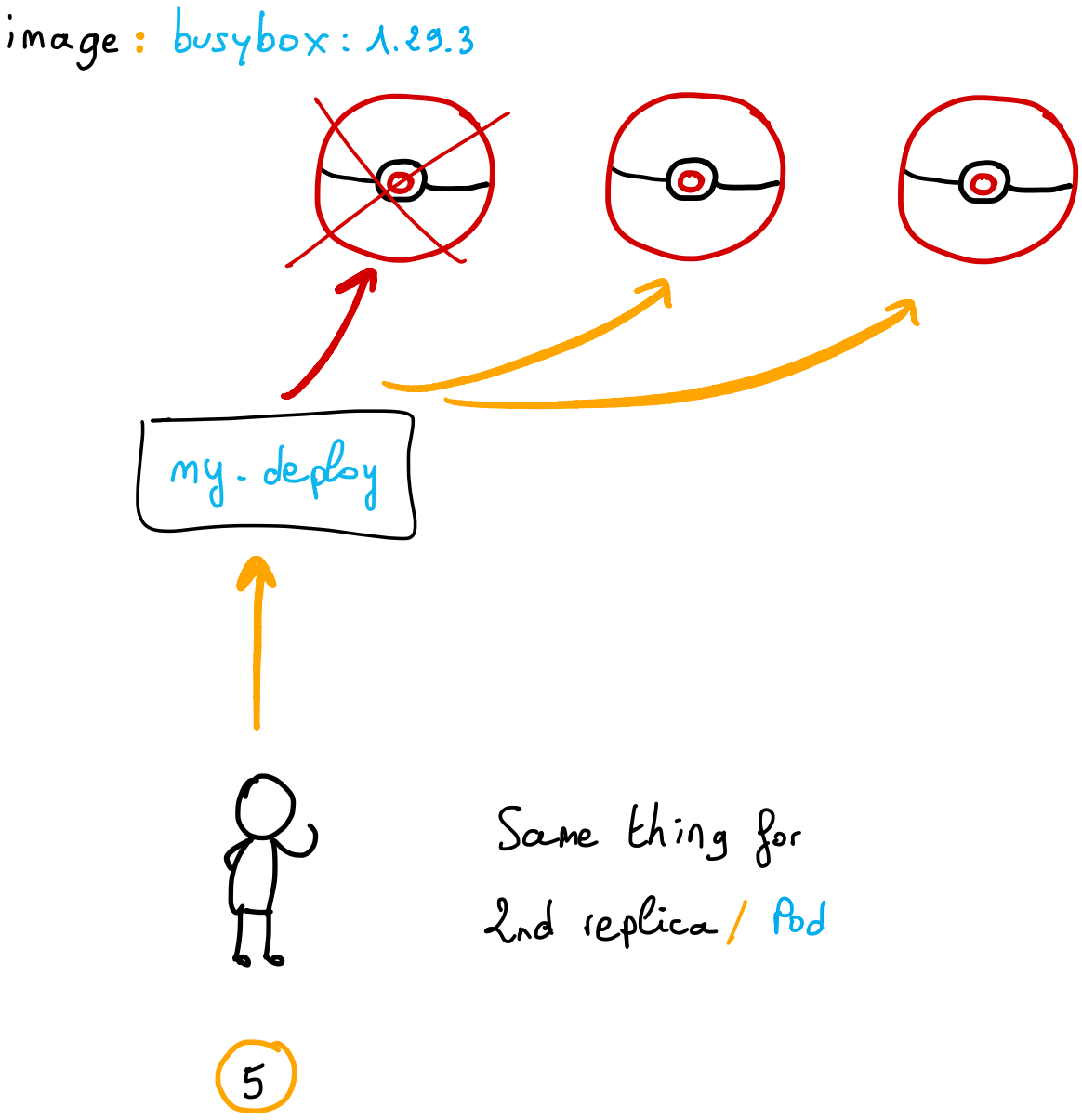
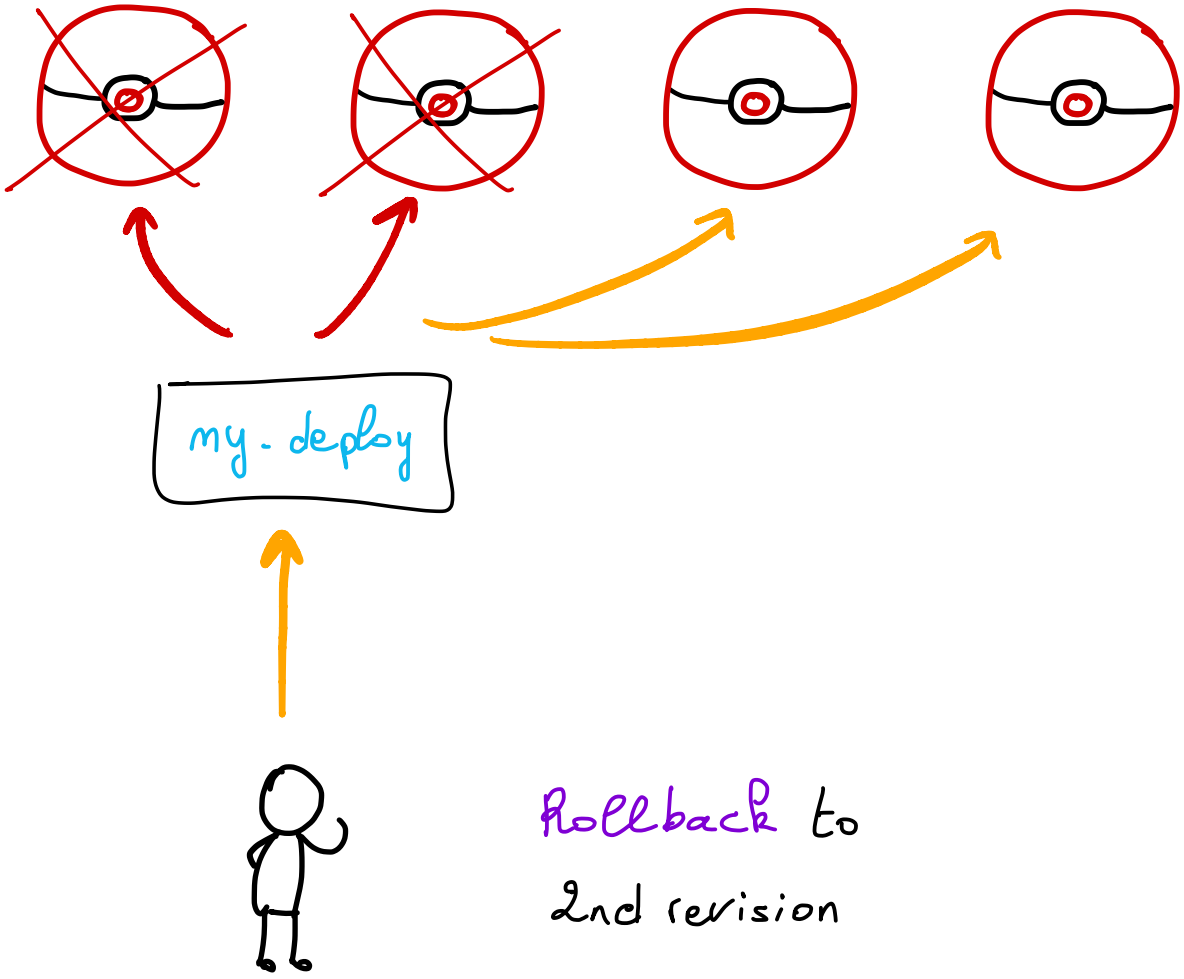


image : busybox : 1.29.3
my - deploy
Same thing for
2nd replica / Pod
5


my-deploy
Rollback to
2nd revision

1 Create a Deployment in a file named my-deploy.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-deploy
  labels:
    app: busybox
spec:
  replicas: 3
  selector:
    matchLabels:
      app: busybox
  template:
    metadata:
      labels:
        app: my-deploy
    spec:
      containers:
      - name: busybox
        image: busybox
  strategy:
    rollingUpdate:
      maxSurge: 100%
      maxUnavailable: 0%
    type: RollingUpdate
```

100% of Pods will be created & then old Pods will be deleted

2 Apply manifest YAML file

```
$ kubectl apply -f my-deploy.yaml
```

3 Update deployment container's image name / tag

```
$ kubectl set image deploy my-deploy
busybox=busybox:1.29.3 --record
```

4 Show deployment rolling update logs

```
$ kubectl rollout status deploy my-deploy
```

5 Show history of deployment creation / updates

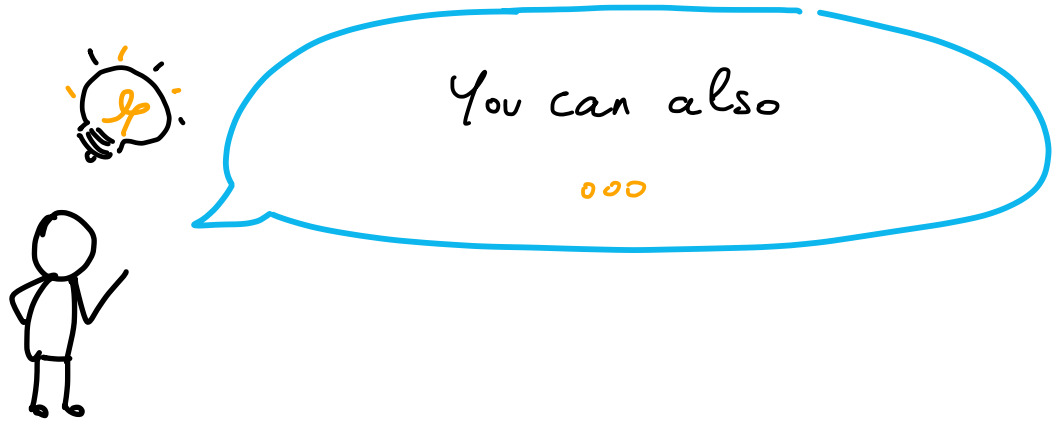
```
$ kubectl rollout history deploy my-deploy
```

6 Show history of deployment creation / updates of the 2nd revision

```
$ kubectl rollout history deploy my-deploy --revision=2
```

7 Rollback to previous deployment

```
$ kubectl rollout undo deploy my-deploy
```

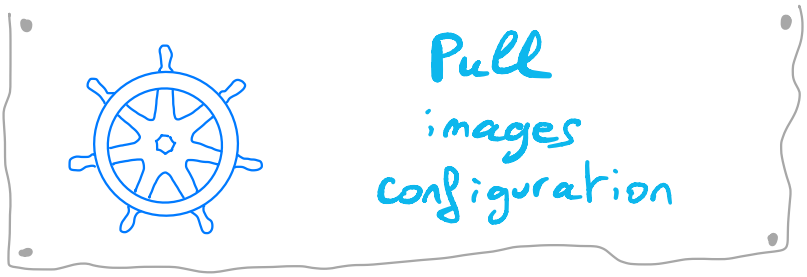


- Restart a deployment / kill existing Pods and start new Pods

```
$ kubectl rollout restart deploy my-deploy
```

- Pause and then resume the deployment of the resource

```
$ kubectl rollout pause deploy my-deploy
$ kubectl rollout resume deploy my-deploy
```



Pull images configuration

→ In order to pull container images, Kubernetes need several configuration

→ Theses configurations can be set on:

- ☐ Pod
- ☐ Deployment
- ☐ CronJob
- ☐ Job
- ☐ ReplicaSet
- ☐ DaemonSet
- ☐ StatefulSet

ooo Every resource type including container spec

ImagePullPolicy

Thanks to this configuration, I know when I need to pull container images

Kubelet

```
...
    containers:
    - name: my-container
      image: my-registry.com/my-app:tag
      imagePullPolicy: Always
```

☆ IfNotPresent (default value):

Kubelet doesn't pull image from registry if image already exists in the Node

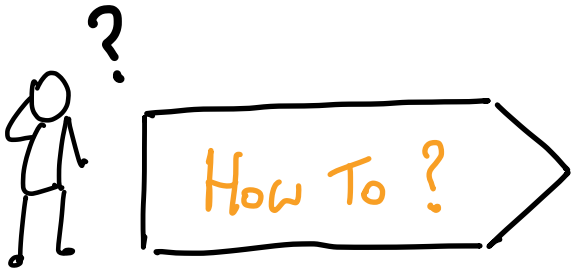
☆ Always:

Image is pulled everytime the Pod is started

☆ Never:

Image is assumed to exist locally.
No attempt is made to pull the image.


ImagePullSecrets
Thanks to this configuration,
I know I need to use a secret
to access to the
image registry
Kubelet



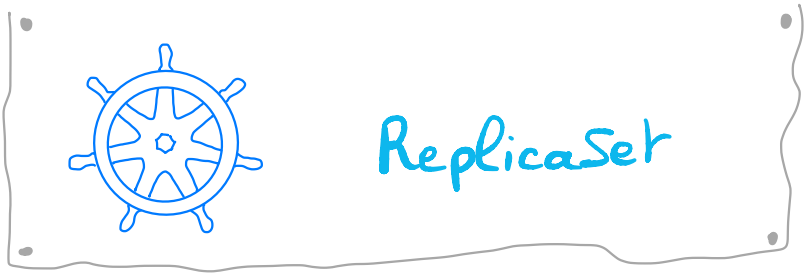

```
…
      containers:
      - name: my-container
        image: my-registry.com/my-app:tag
        imagePullPolicy: Always
      imagePullSecrets:
      - name: registry-secret
```

› Kubernetes uses docker-registry secret type for image registry authentication

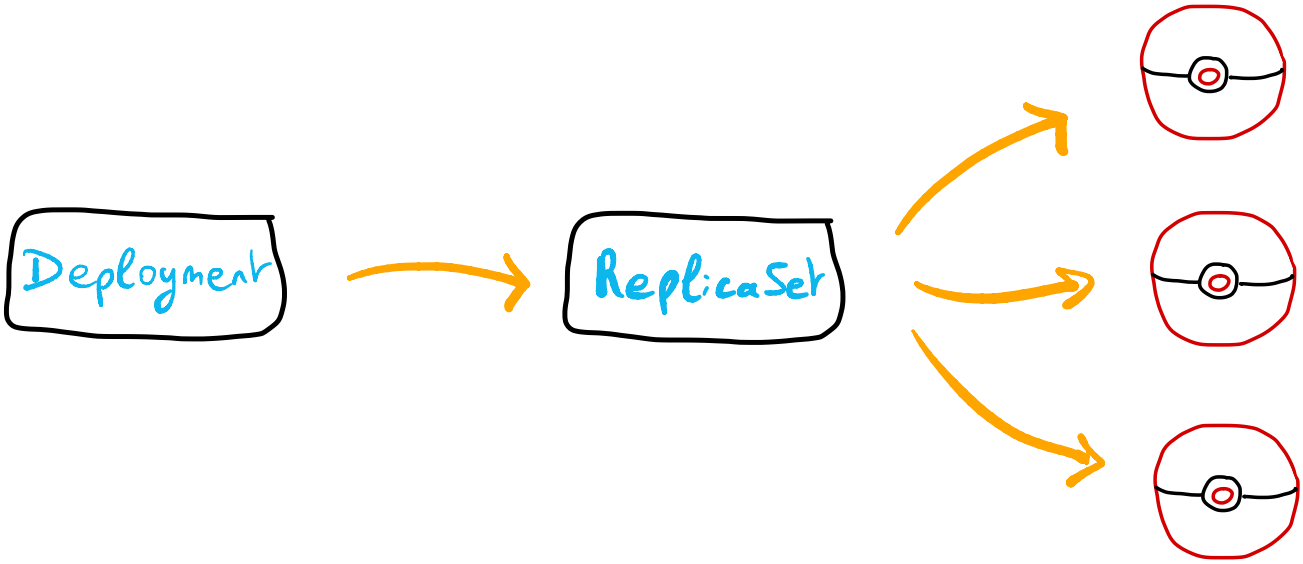
```
$ kubectl create secret docker-registry registry-secret
--docker-server=<host>:8500
--docker-username=<user_name>
--docker-password=<user_password>
--docker-email=<user_email>
```

› You can also create a secret from an existing .docker/config.json file:

```
$ kubectl create secret generic registry-secret
--from-file=.dockerconfigjson=<path/to/.docker/config.json>
--type=kubernetes.io/dockerconfigjson
```



→ ReplicaSet creates Pods according to the number of specified replicas

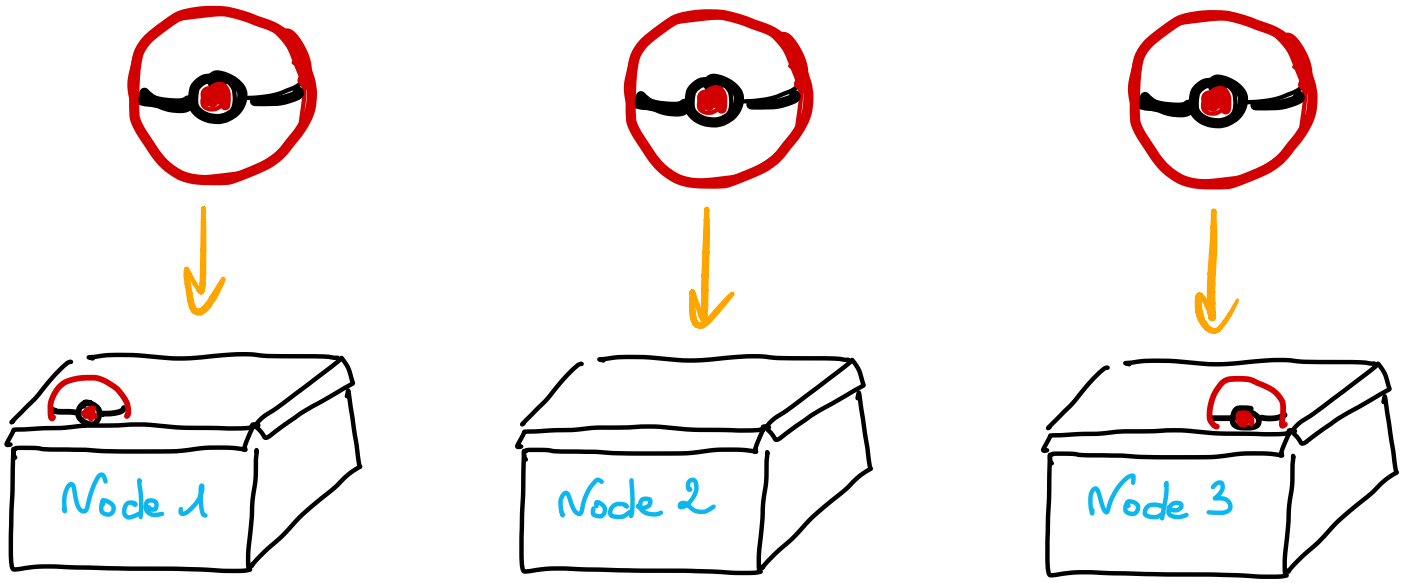


→ ReplicaSet can be created manually or automatically by a Deployment

It's useful to know the existence of ReplicaSet but not mandatory to manage them manually.

DaemonSet

→ Ensures that all eligibles Nodes run a copy of a Pod defined by a DaemonSet



⚠ Kubernetes scheduler doesn't manage theses Pods but DaemonSet controller will handle them instead.

→ If a new Node is added in the cluster, DaemonSet controller will create a Pod on it.

Update strategies

RollingUpdate

Update strategy by default.
Old Pods will be automatically killed and new ones will be created : one Pod per Node.

onDelete

Existing Pods need to be manually deleted in order for the new ones to be created.

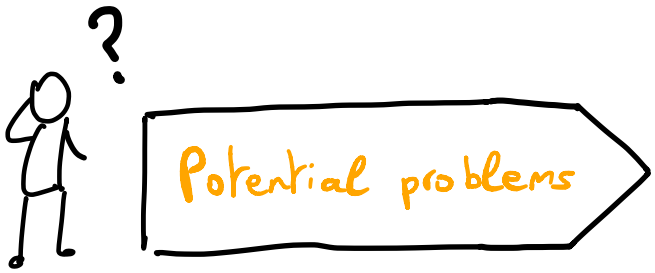
1. Create a DaemonSet that creates one busybox Pod per Node

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: my-app
spec:
  selector:
    matchLabels:
      name: my-app
  template:
    metadata:
      labels:
        name: my-app
    spec:
      tolerations:
      - key: node-role.kubernetes.io/master
        effect: NoSchedule
      containers:
      - name: busybox
        image: busybox
```

Specify DaemonSet will run on Master Nodes

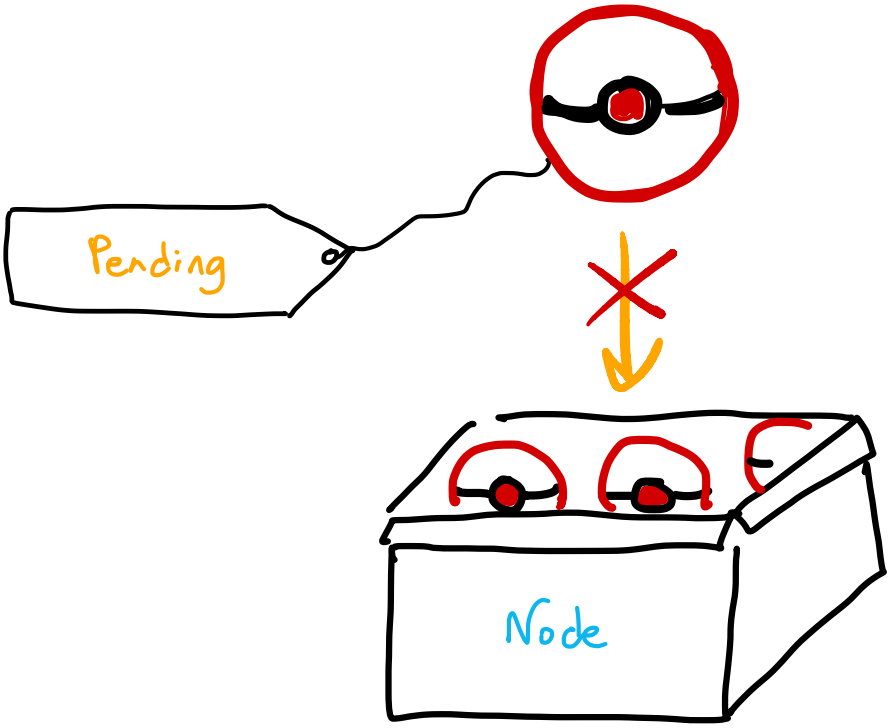
2 Create a DaemonSet that creates Pods only on desired Nodes

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: my-app
spec:
  selector:
    matchLabels:
      name: my-app
  template:
    metadata:
      labels:
        name: my-app
    spec:
      nodeSelector:
        my-key: my-value
      containers:
      - name: busybox
        image: busybox
```



DaemonSet wants to create Pods on each Node

but one of them doesn't have enough capacities



Pod is stuck in

"Pending" state

Quick solutions

- List all the Nodes in the cluster

```
$ kubectl get nodes
```

- Find the Nodes that have your Pods

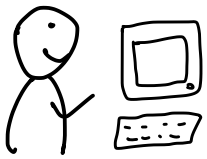
```
$ kubectl get pods -l name=my-app -o wide -n my-ns
```

- Compare the lists in order to find Node in trouble.

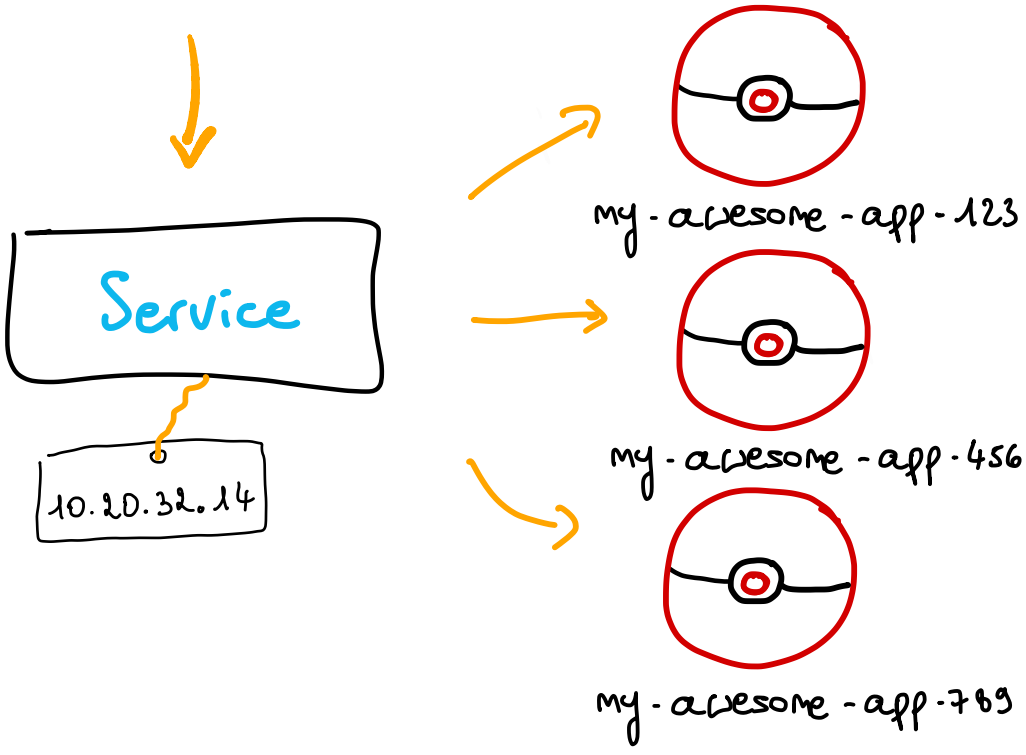
- Delete manually on this Node, all the Pods not controlled by DaemonSet

→ Allows to reach an application (Pods are mortal)
with a single IP address

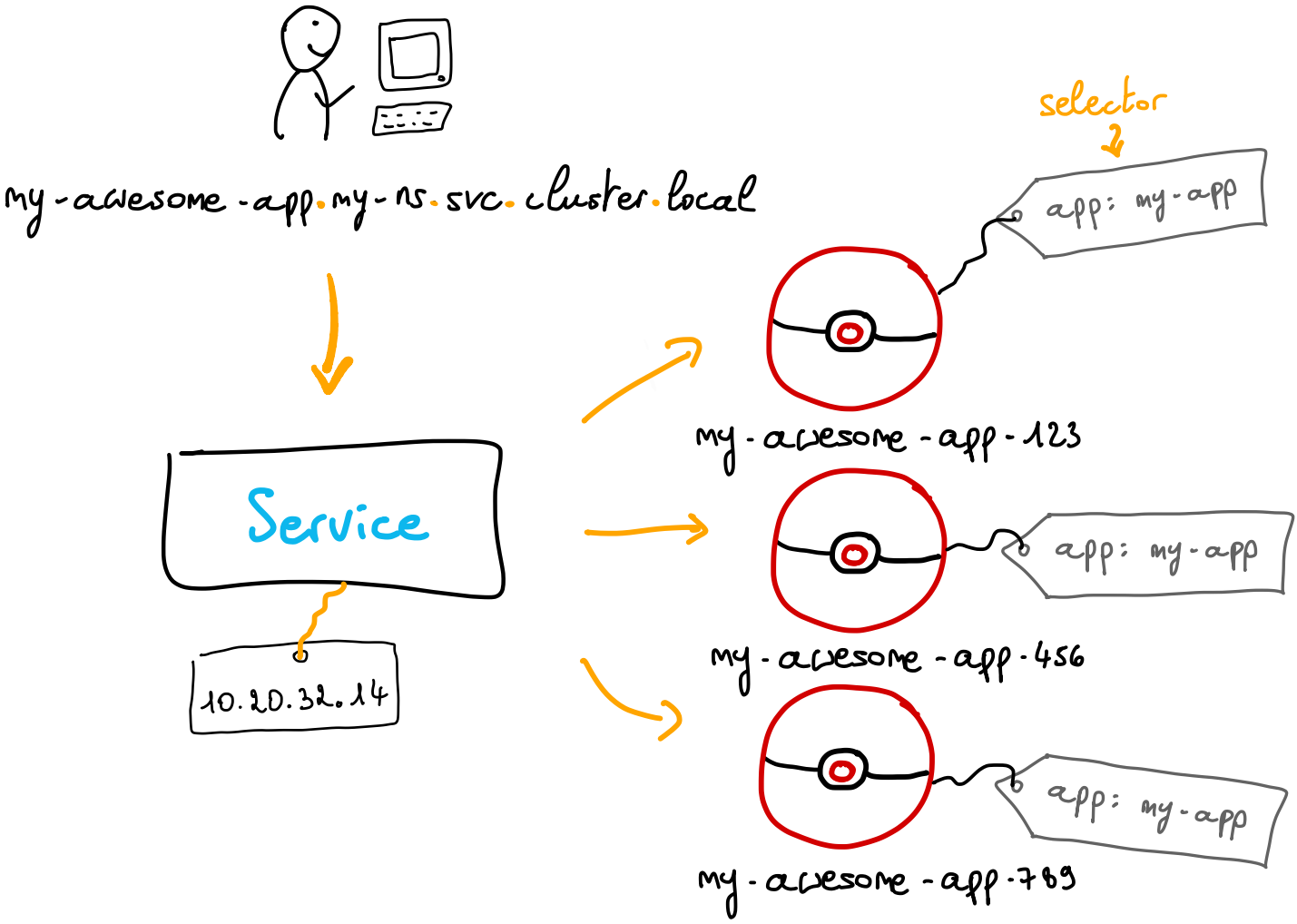
→ Assigns to a group of Pods a unique DNS name



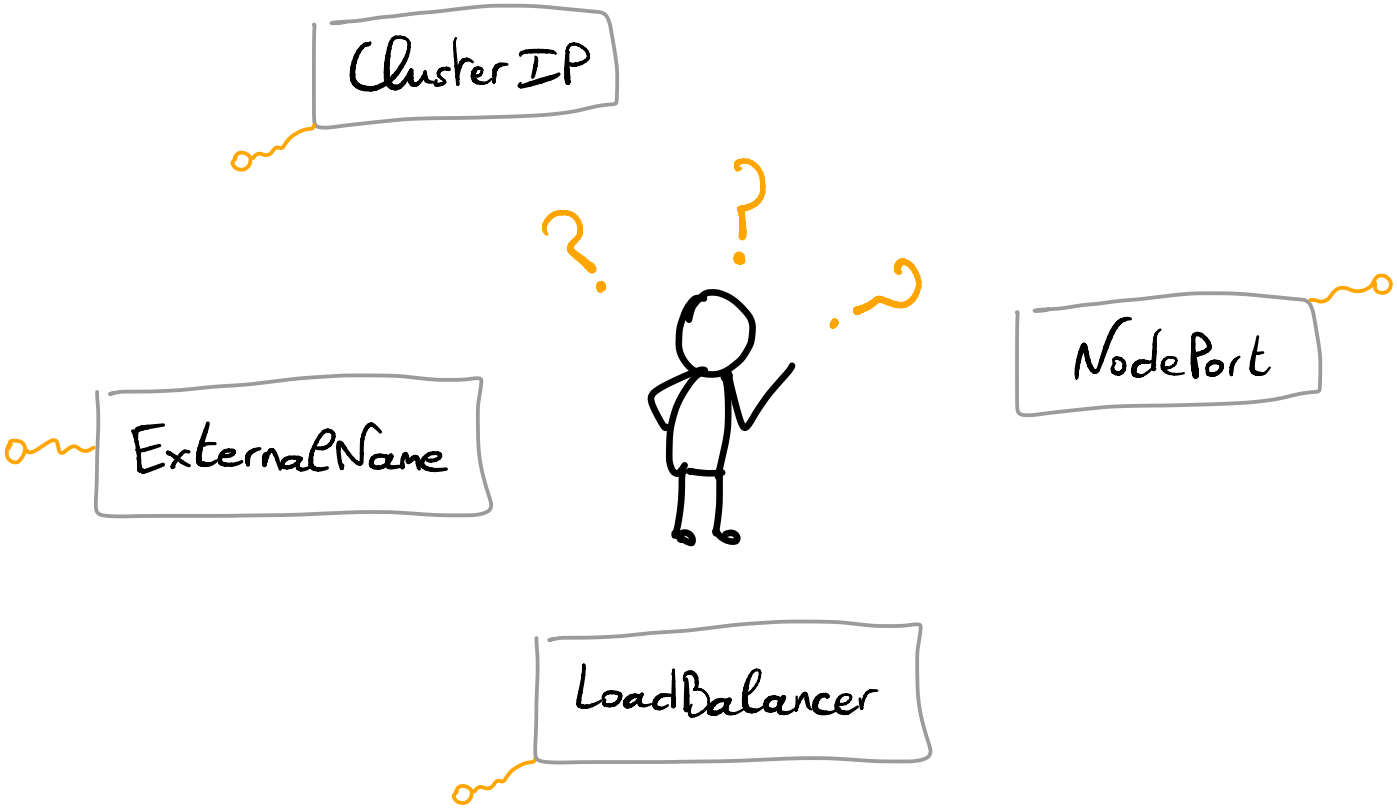
my-awesome-app.my-ns.svc.cluster.local



→ The set of Pods targeted by a Service is usually determined by a selector



→ Several kind of Services:

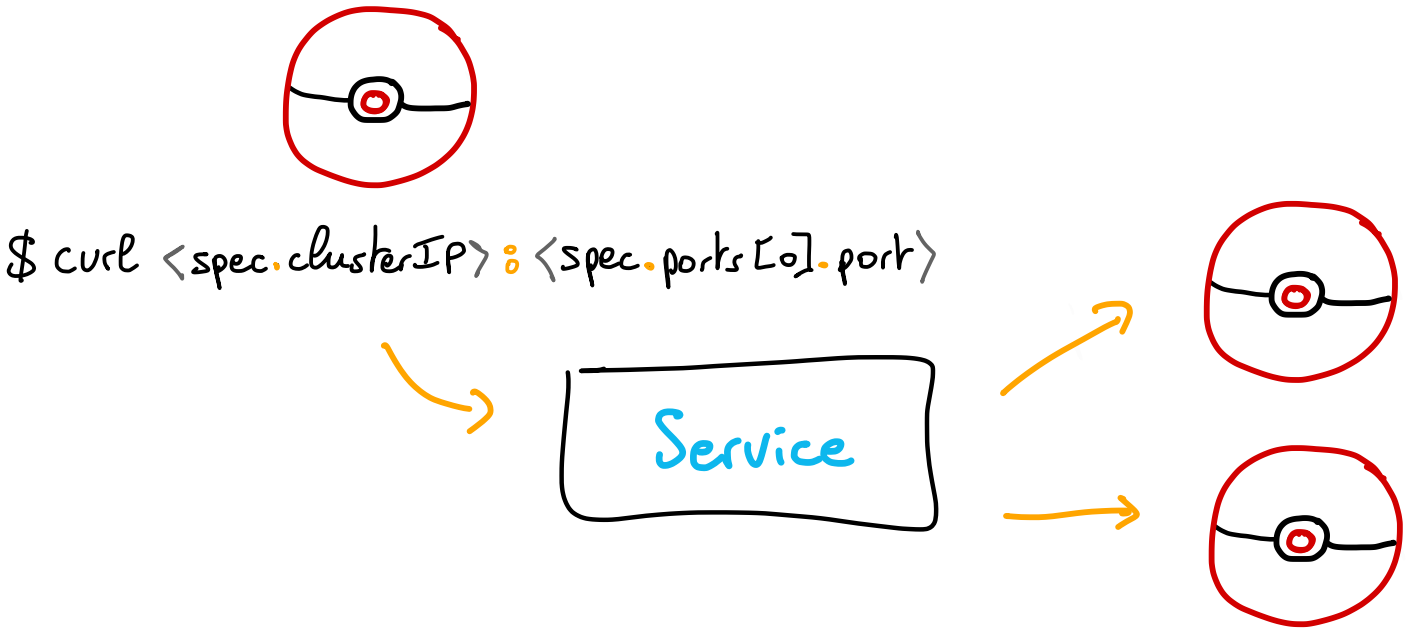


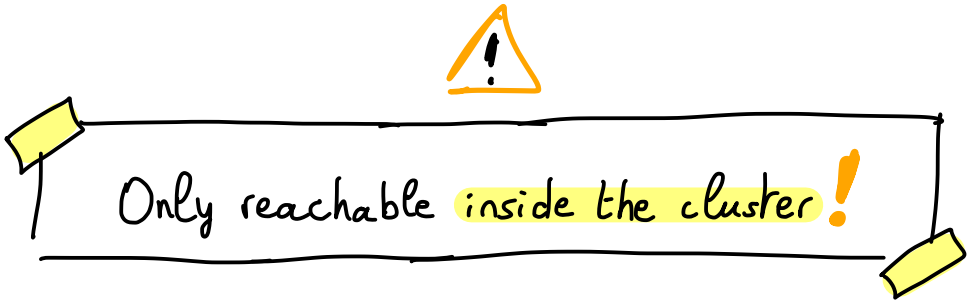
Types

ClusterIP

by Default

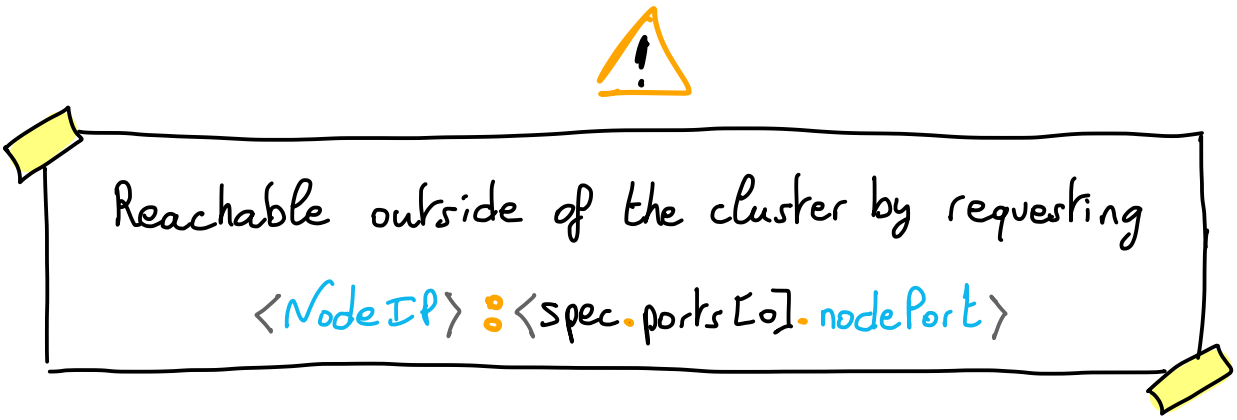
→ Exposes the service on a cluster internal IP

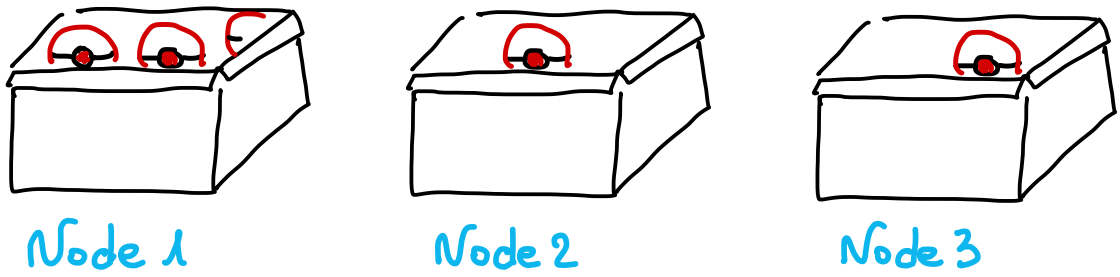




NodePort

→ Exposes the service on each Node's IP





LoadBalancer

→ Creates an external Load Balancer

→ Assigns a fixed external IP address

Reachable outside of the cluster

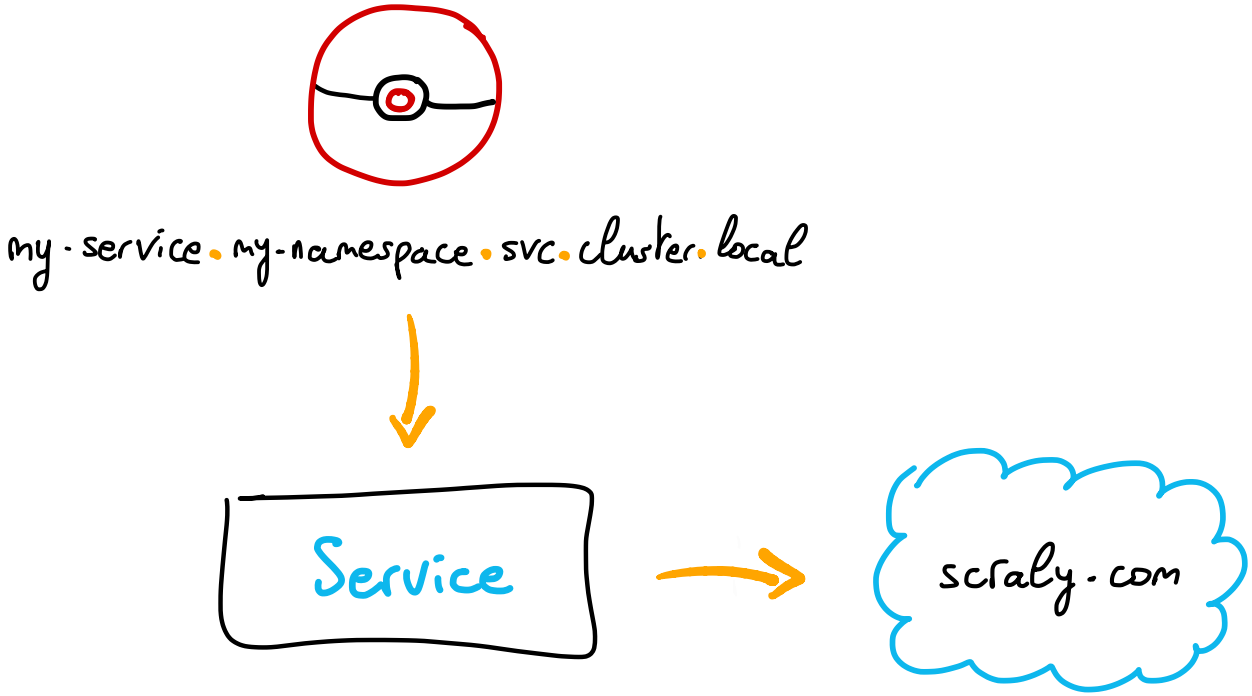
Only for managed clusters

&

if cloud provider supports

ExternalName

- Provides an internal alias for an external DNS name
- Requests will be redirected to the external name



Headless

Uh, it's not a real type

But it's useful in some cases

- Useful in order to interface with other service discovery mechanism
- A ClusterIP is not allocated
- Kube-proxy doesn't handle the service
- Exposes all replicas as DNS entry

spec.clusterIP

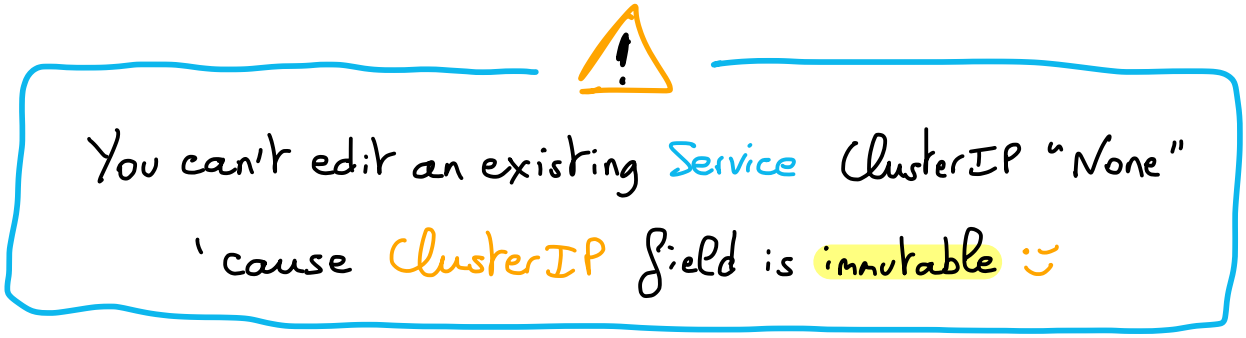
should be equal to None

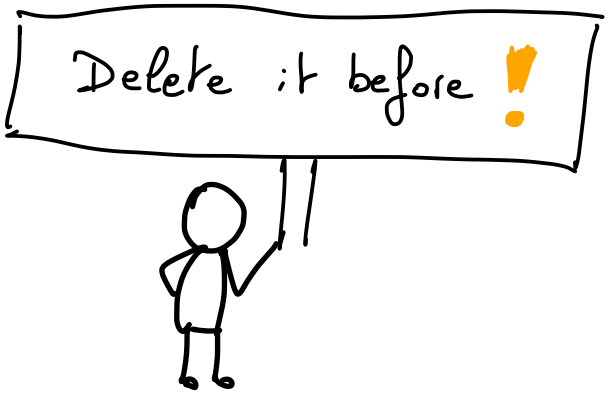
› Create a Service that exposes a Deployment on port 80

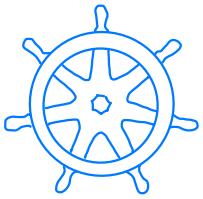
```
$ kubectl expose deploy my-deploy --type=LoadBalancer
--name=my-svc --target-port=8080 -n my-namespace
```

› Create a Pod and expose it through a Service

```
$ kubectl run my-pod --image=nginx --restart=Never
--port=80 --expose -n my-namespace
```



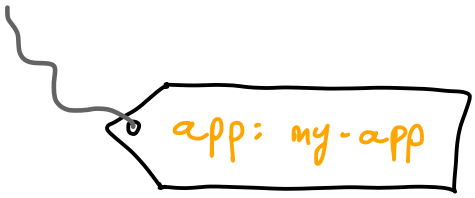




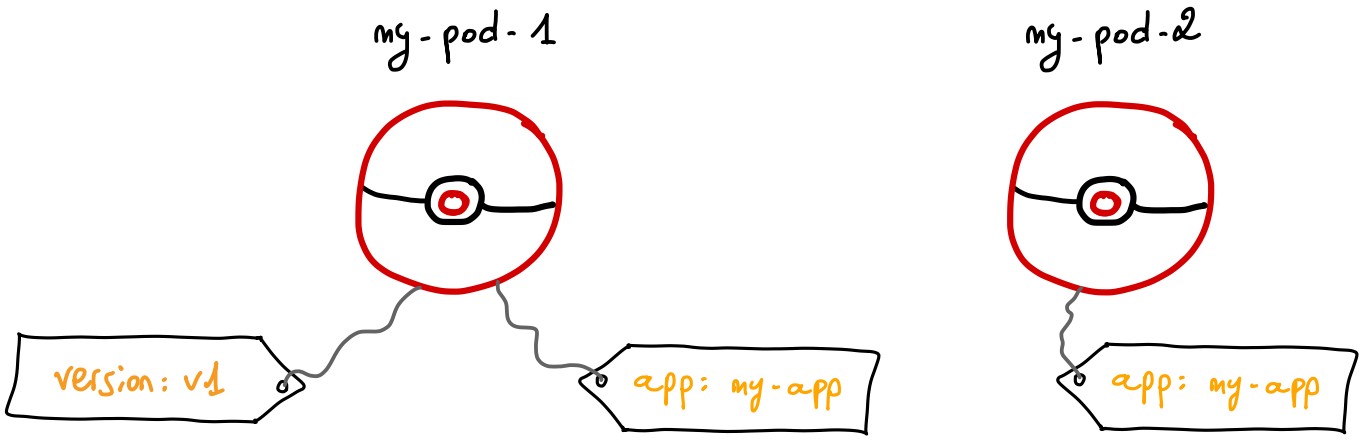
Labels & Selectors

Labels

→ Key / value pairs attached to an object



→ Several objects / resources can have the same label



kubernetes.io / k8s.io prefixes in label keys are reserved for Kubernetes core components

> Create a Pod with two labels

```
apiVersion: v1
kind: Pod
metadata:
  name: my-app
  labels:
    app: my-app
    version: v1
spec:
  containers:
  - name: nginx
    image: nginx
    ports:
    - containerPort: 80
```

Selectors

→ Selectors use labels to filter

or select objects / resources

→ Selectors can be :

matchLabels

exact match : = , == , !=

matchExpressions

match expressions : in , notin , exists

> Show labels for all of my Pods

```
$ kubectl get pod --show-labels -n my-namespace
```

> Create a Deployment that will manage Pods
> label app: my-app

```
apiVersion: v1
kind: Deployment
metadata:
  name: my-deploy
  labels:
    app: my-app
spec:
  selector:
    matchLabels:
      app: my-app
```

} equality based

› Create a Deployment that manage Pods that have label version value equals to v1 or v2 & app = my-app

```
apiVersion: v1
kind: Deployment
metadata:
  name: my-deploy
  labels:
    app: my-app
spec:
  selector:
    matchLabels:
      app: my-app
    matchExpressions:
      - {key: version, operator: In, values: ["v1","v2"]}
```

} set based

› List Pods that have labels app: my-app & version: v2

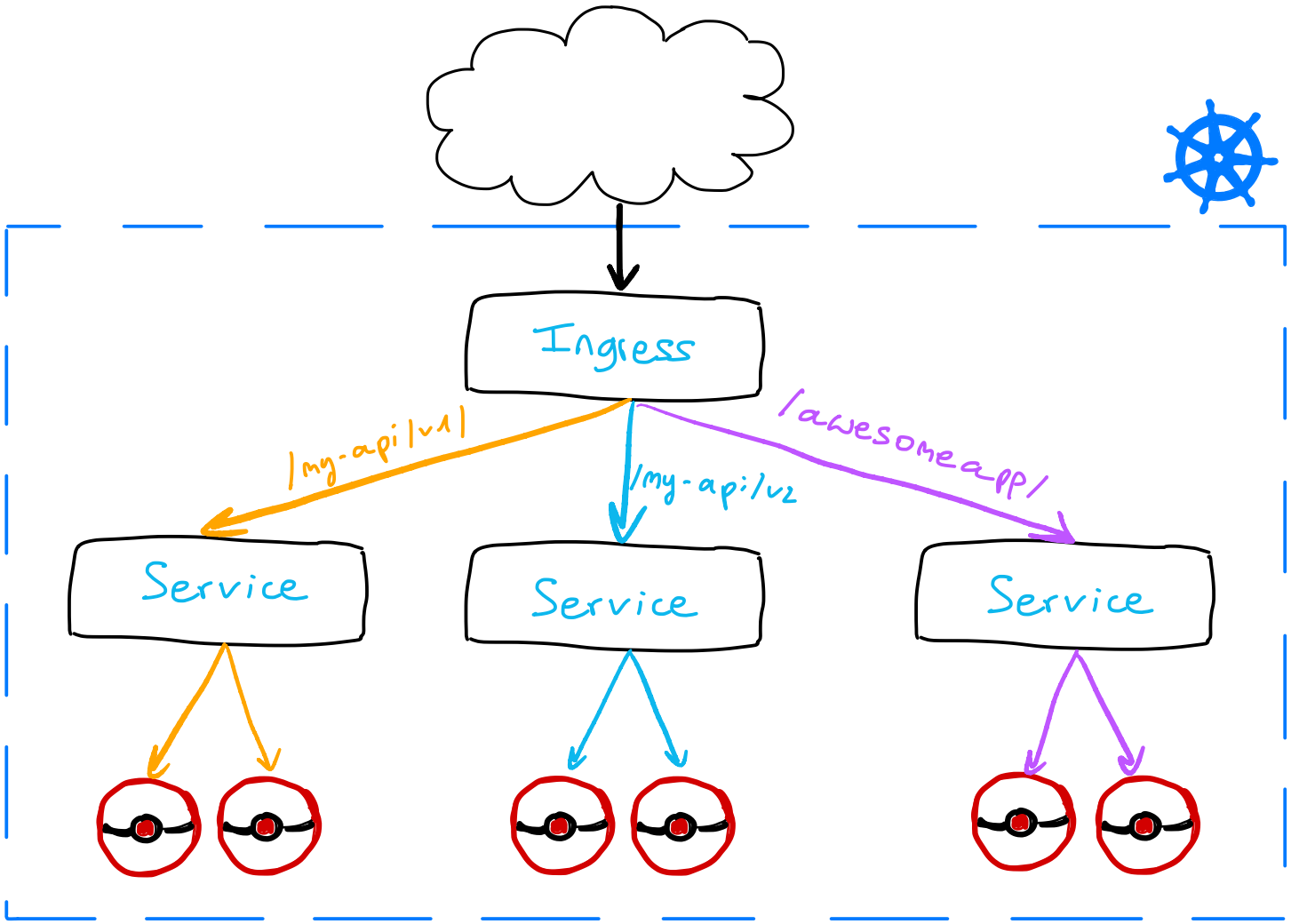
```
$ kubectl get pod -l app=my-app,version=v2
```

› List Pods that have labels app: my-app & version = v1, v2 or v3

```
$ kubectl get pod -l app=my-app,version in (v1,v2,v3)
```

Ingress

- Allows access to your Services from outside the cluster
- Avoids creating a Load Balancer per Service
- Single external endpoint (secured through SSL/TLS)

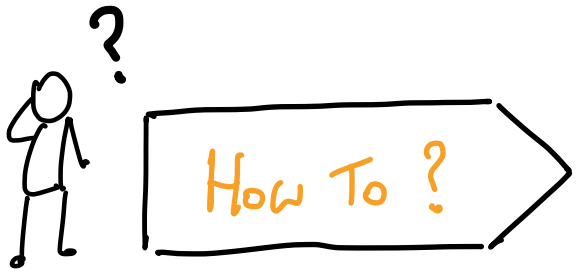


→ An Ingress is implemented by a 3rd party:

an Ingress Controller

↳ extends specs to support additional features

→ Consolidates several routes in a single resource



> Create a Nginx based Ingress which defines two routes:

- /my-api/v1/ to myapi-v1 Service
- /my-api/v2/ to myapi-v2 Service

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - host: scraly.com
    http:
      paths:
      - path: /my-api/v1
        pathType: Prefix
        backend:
          service:
            name: my-api-v1
            port:
              number: 80
      - path: /my-api/v2
        pathType: Prefix
        backend:
          service:
            name: my-api-v2
            port:
              number: 80
```

If no host is specified, rules are applied to all inbound HTTP traffic

IngressClass

In order to specify which Ingress should be handled by controller

```
…
metadata:
  name: my-gce-ingress
  annotations:
    kubernetes.io/ingress.class: gce
```

PathType → How HTTP path should be matched

Exact: Matches the URL with case sensitivity

Prefix: Matches the URL path prefix split by /

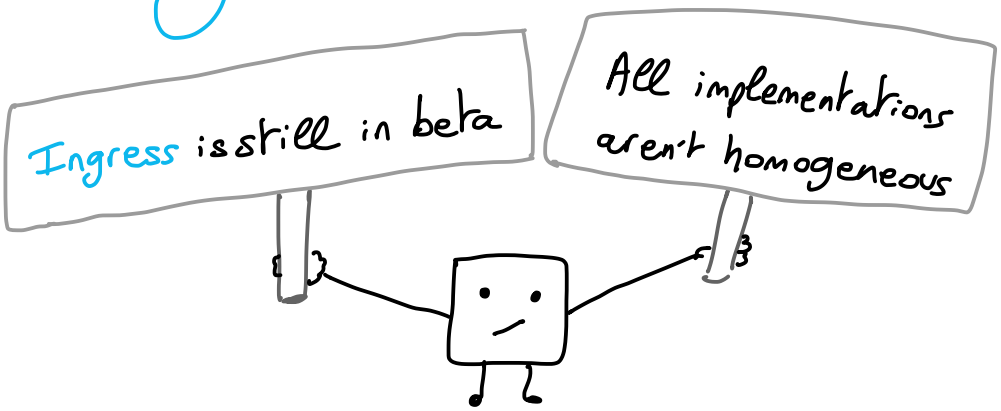
/my-api/v1/ matches /my-api/v1/

/my-api/v1/toto

Create an Ingress secured through TLS

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-ingress-with-tls
spec:
  tls:
  - hosts:
      - scraly.com
    secretName: secret-tls
  rules:
  - host: scraly.com
    …
```

But



Istio

My implementation of an Ingress is a Gateway

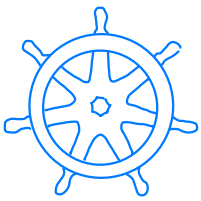
In my side, I introduces HTTP Proxy

Contour

træfik

I took the old name of Contour's name : IngressRoute

I'm sure you have understood it, depending on the choosen 3rd party, implementation is different ...

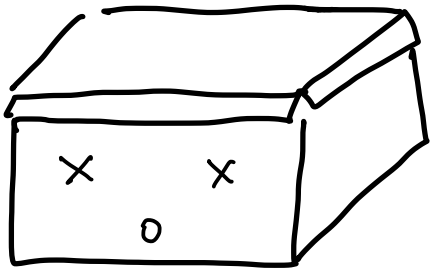


PV, PVC & StorageClass

In short :

→ Pods are mortal by default

& Nodes will not live forever too

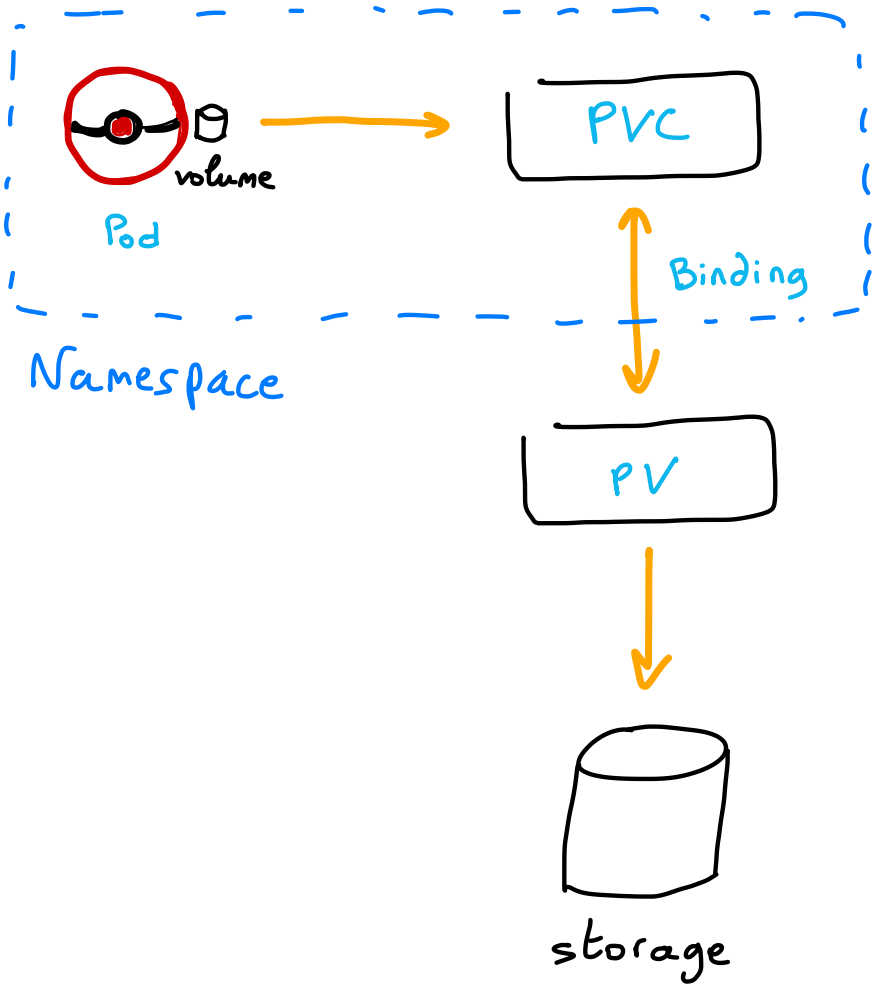


→ To store data permanently, use PersistentVolume

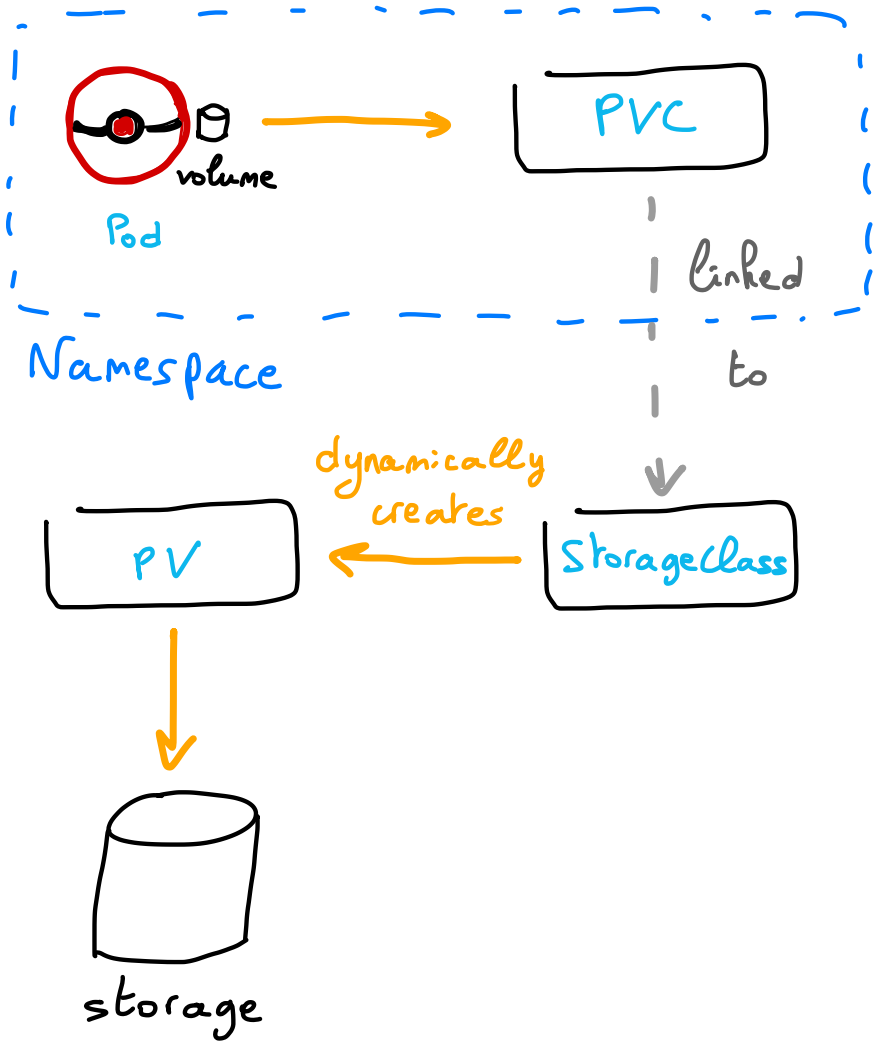
→ Create a PersistentVolumeClaim to request this physical storage

→ Pods will use this PVC as a volume

Static provisioning (without StorageClass):

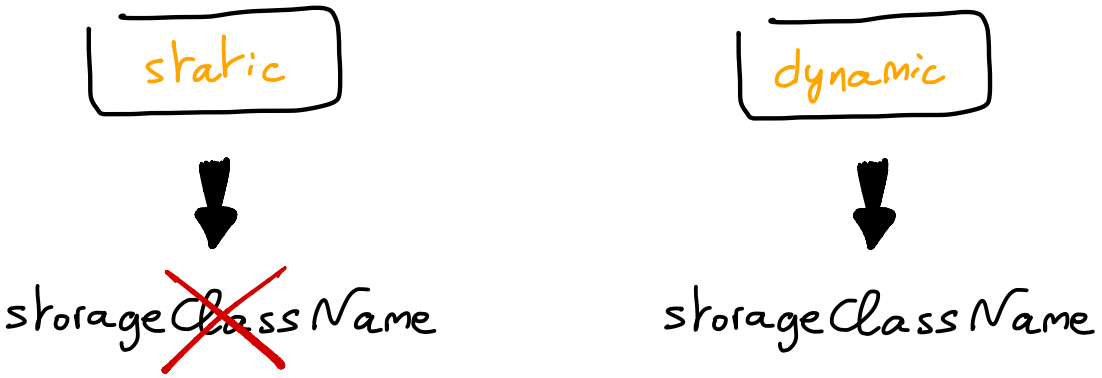


Dynamic provisioning (with StorageClass):



PersistentVolume

- Provides a storage location that has a lifetime independant of any Pod or Node
- PV is not sticked in a namespace
- Supports different persistent storage types : NFS, cloud provider specific storage system ...
- After creation, PV has a STATUS of Available. Means not yet bound to a PVC.
- 2 sorts of provisioning :



Access Modes

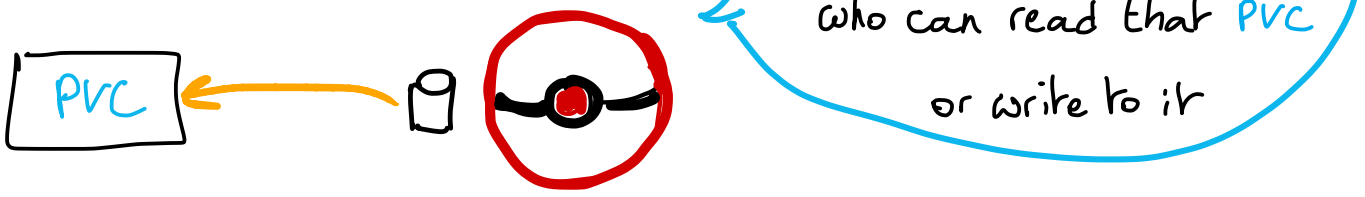
ReadWriteOnce : the volume can be mounted as read-write by a single Node
↳ RWO

ReadWriteMany : the volume can be mounted as read-write by many Nodes
↳ RWX

ReadOnlyMany : the volume can be mounted as read-only by many Nodes
↳ ROX

ReadWriteOncePod : the volume can be mounted as read-write by a single Pod.

since → 1.22



ReclaimPolicy

Retain : Appropriate for precious data.
PV is not deleted if an user delete the PVC.

Delete : **Default ReclaimPolicy** for dynamic provisioning.
PV is automatically deleted if an user delete the PVC.
Status of the PV will change to Released and all the data can be manually recovered.

> Create a PV, with NFS type, linked to a AWS EFS

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: my-pv
spec:
  capacity:
    storage: 100Gi
  accessModes:
    - ReadWriteMany
  nfs:
    path: /
    server: fs-xxx.efs.eu-central-1.amazon.com
    readOnly: false
```

> Change ReclaimPolicy for my-pv to Retain

```
$ kubectl patch pv my-pv
-p '{"spec":{"persistentVolumeReclaimPolicy":"Retain"}}'
```

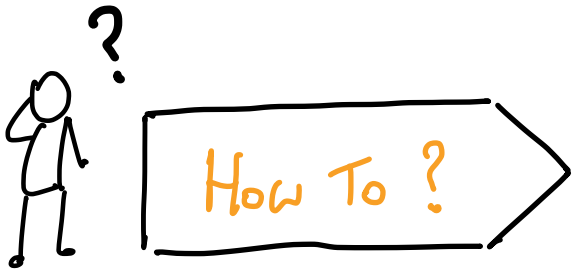
StorageClass

→ In order to set-up dynamic provisioning

→ Different provisioners exists :

GCEPersistentDisk AzureDisk AWS EBS ...

→ volumeBindingMode : Immediate means that volume binding & dynamic provision will occur when the PVC is created



> Create a StorageClass with SSD disks for Google Compute Engine (GCE)

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: faster
provisioner: kubernetes.io/gce-pd
  parameters:
    type: pd-ssd
  volumeBindingMode: Immediate
```

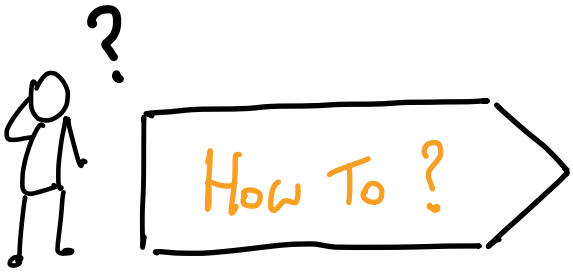
PersistentVolumeClaim

→ Claim / Request a suitable PersistentVolume

→ Pod use PVC to request physical storage

→ After PVC deployment, ControlPlane search a suitable PV with the same StorageClass

→ When PV is found & PVC is attached, PV status changes to Bound



> Create a PVC that requests a storage with 100 Gb of storage & read-write many access

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: my-pvc
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 100Gi
    storageClassName: ""
    volumeName: my-pv
```

Attach to a Pod

→ A volume is accessible for all containers in a Pod

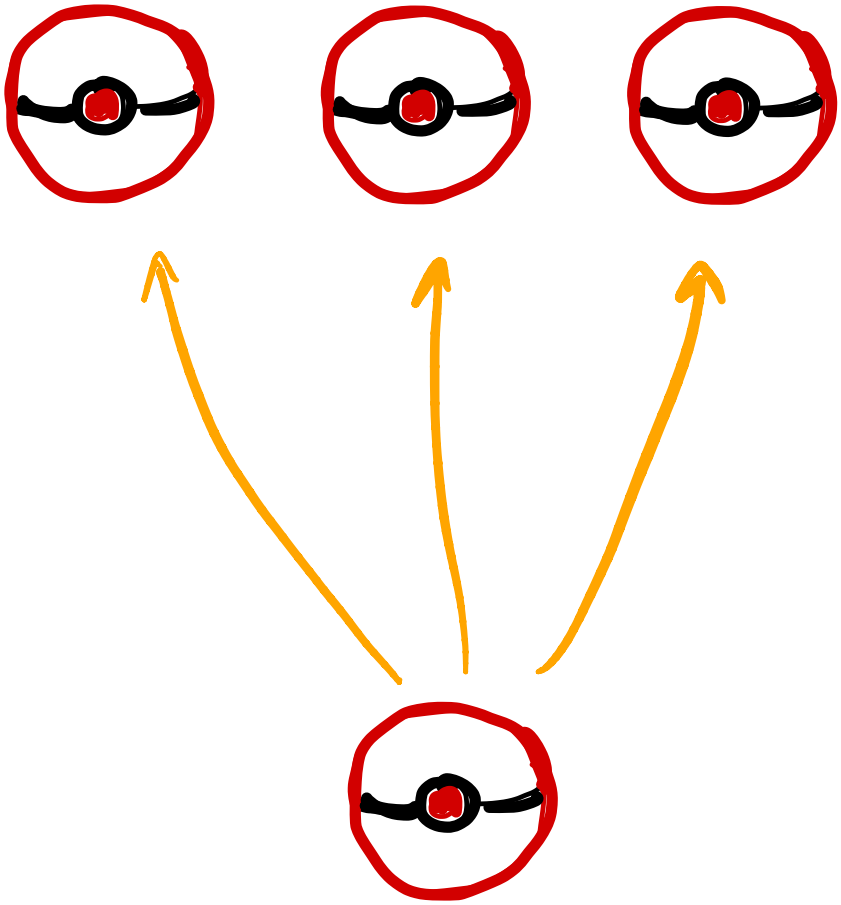
→ A container must mount the volume to access to it

> Mount PVC as a volume in a Pod with read-only access

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  containers:
    - name: my-container
      image: nginx
      volumeMounts:
      - mountPath: "/data"
        name: nfs-storage
        readOnly: true
  volumes:
    - name: nfs-storage
      persistentVolumeClaim:
        claimName: my-pvc
```

Horizontal Pod Autoscaler

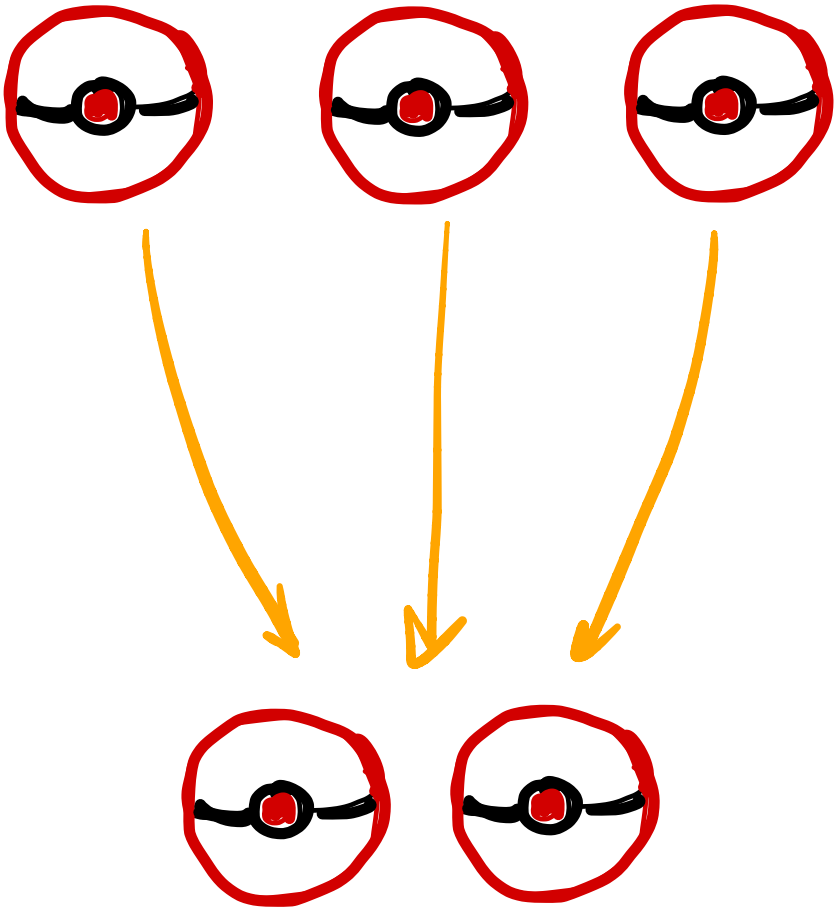
→ Scales the number of Pods automatically on observed CPU



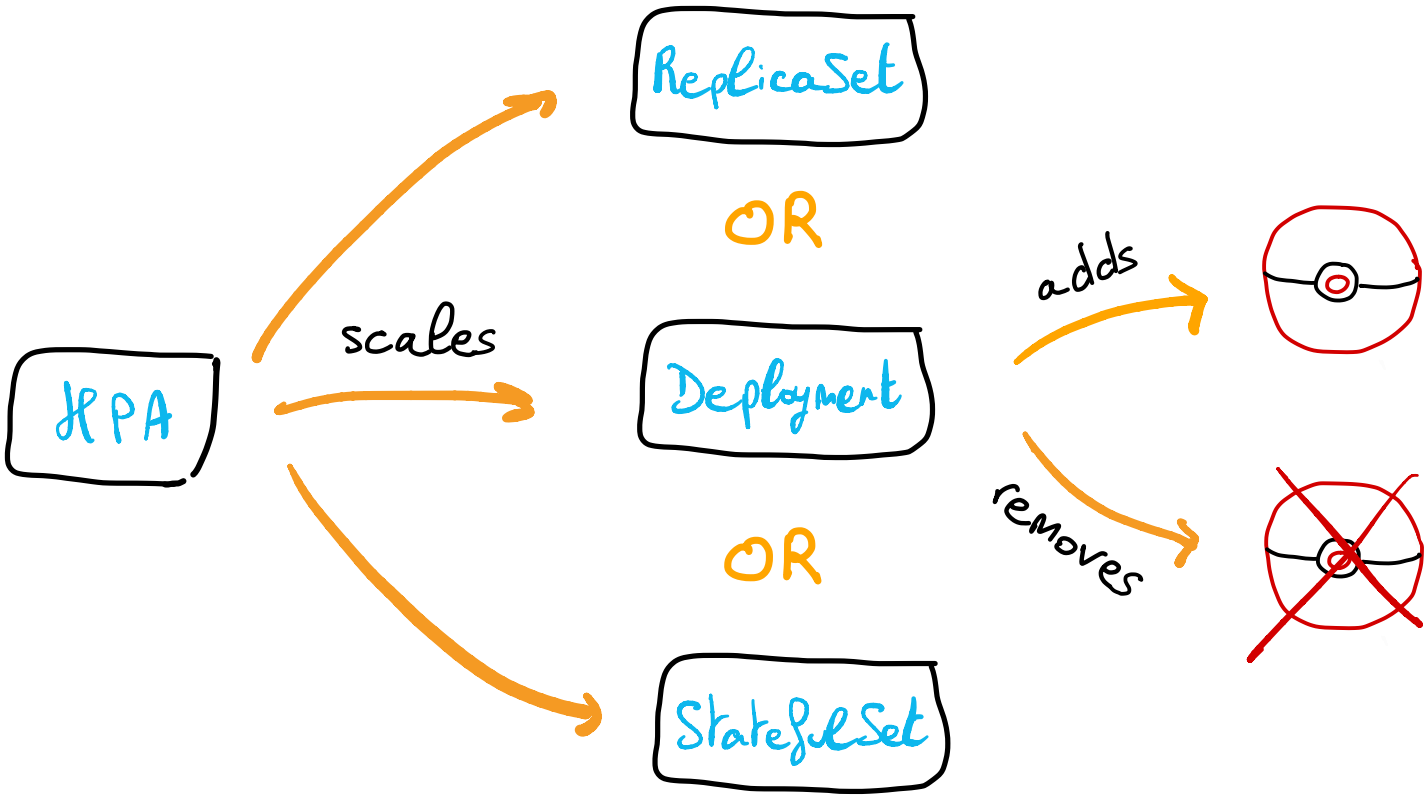
since

autoscaling /v2beta2

→ by custom, external & multiples metrics



→ Available for ReplicaSet, Deployment & StatefulSet



DaemonSet cannot be scaled!

> Autoscale / create a HPA for a deployment that maintains an average CPU usage accross all Pods of 80 %

```
$  kubectl autoscale deploy my-deployment
--min=3 --max=10 --cpu-percent=80
```

New Features

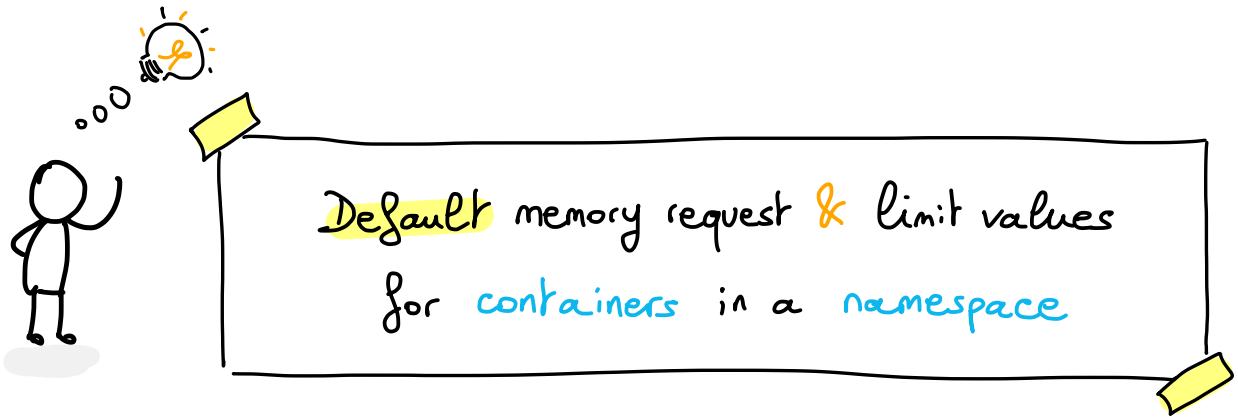
Create a HPA with behavior:

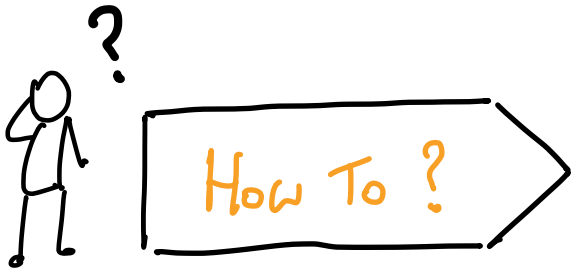
scale up 90% of Pods every 15 sec

scale down 1 Pod every 10 min

```yaml
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: my-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: my-deploy
  minReplicas: 3
  maxReplicas: 10
  targetCPUUtilizationPercentage: 80
  behavior:
    scaleUp:
      policies:
      - type: Percent
        value: 90
        periodSeconds: 15
    scaleDown:
      policies:
      # scale down 1 Pod every 10 min
      - type: Pods
        value: 1
        periodSeconds: 600
```

1.18

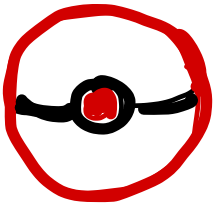




> Create a LimitRange

```
apiVersion: v1
kind: LimitRange
metadata:
  name: memory-limit-range
spec:
  limits:
  - default:
      memory: 512Mi
    defaultRequest:
      memory: 256Mi
    type: Container
```

1 Pod creation with one container without memory request & limit



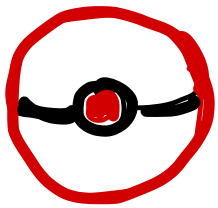
Hello, I need to be scheduled on a Node

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  containers:
  - name: my-container
    image: nginx
    resources:
      requests:
        memory: 256Mi
      limit:
        memory: 512Mi
```

A Limit Range exists in your namespace, I append your configuration with default values, now I can schedule you in a Node !

Limit Range admission controller

2 Pod creation with one container with memory limit only



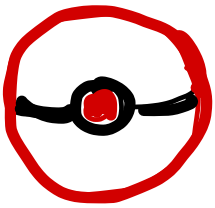
Hello, I need to be scheduled on a Node

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  containers:
  - name: my-container
    image: nginx
    resources:
      limit:
        memory: 512Mi
      requests:
        memory: 256Mi
```

OK, Limit Range exists in your namespace, I add memory request equals to defined request.

Limit Range admission controller

3 Pod creation with one container with memory request only

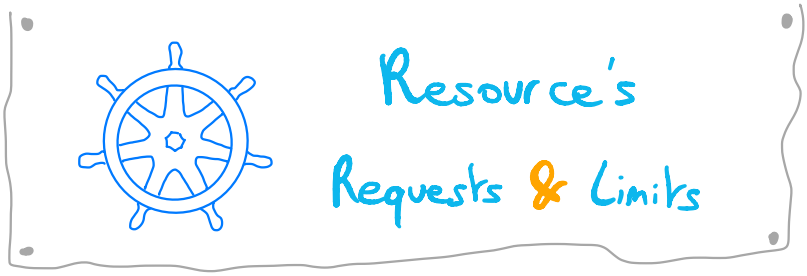


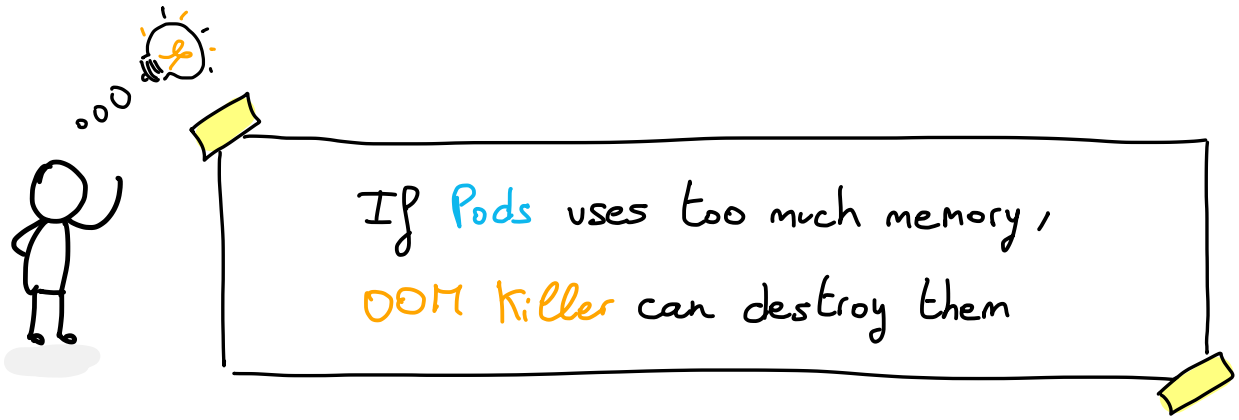
Hello, I need to be scheduled on a Node

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  containers:
  - name: my-container
    image: nginx
    resources:
      requests:
        memory: 100Mi
      limit:
        memory: 512Mi
```

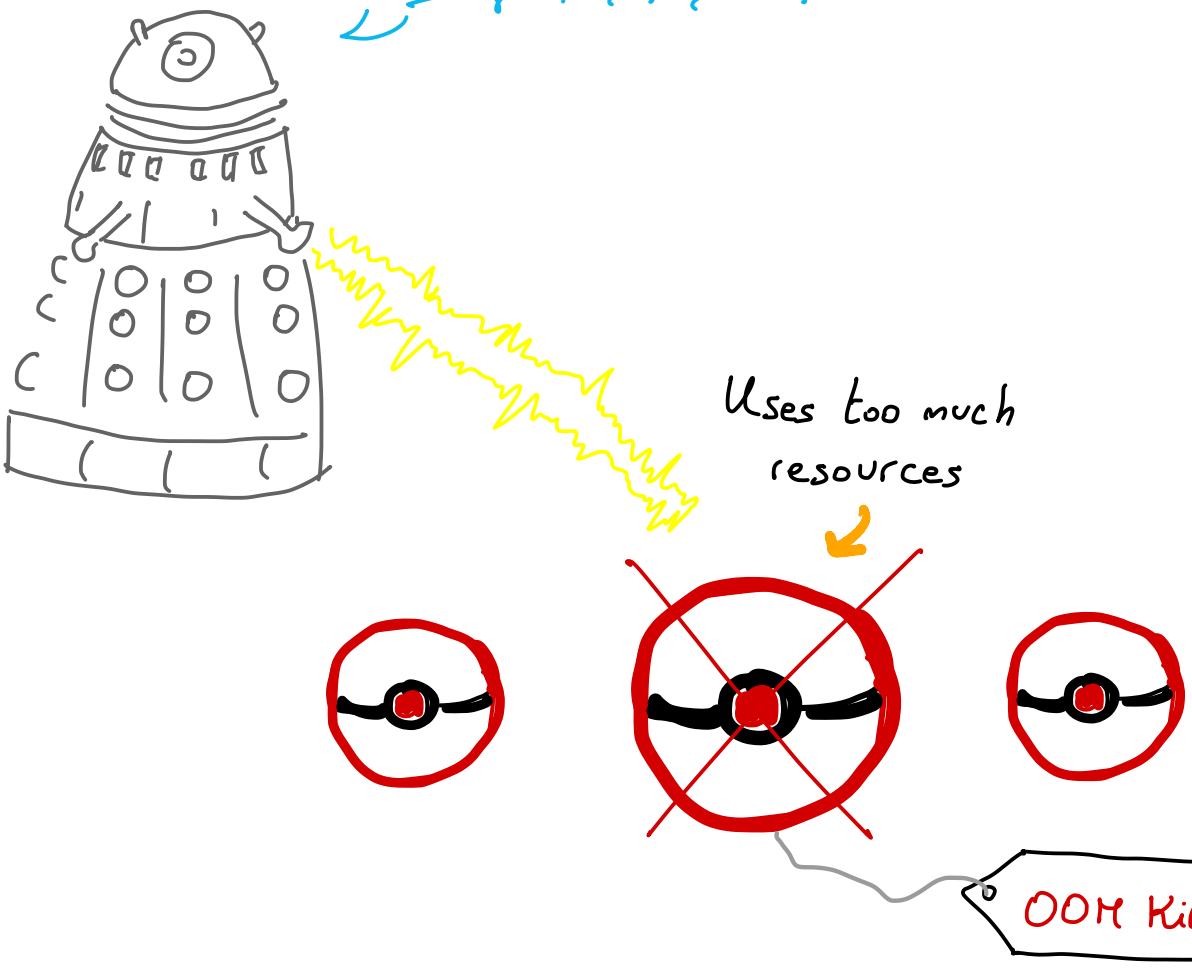
OK, Limit Range exists in your namespace, I add memory limit equals to the one defined in Limit Range.

Limit Range admission controller

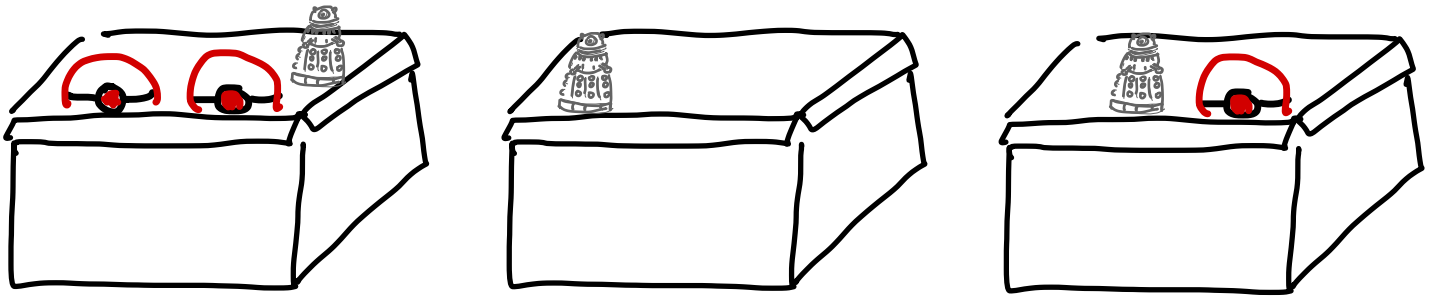

Resource's
Requests & Limits


If Pods uses too much memory,
OOM Killer can destroy them


EX-TER-MI-NATE


Uses too much resources
OOM Killed

→ A OOM (Out of Memory) Killer runs on each Node

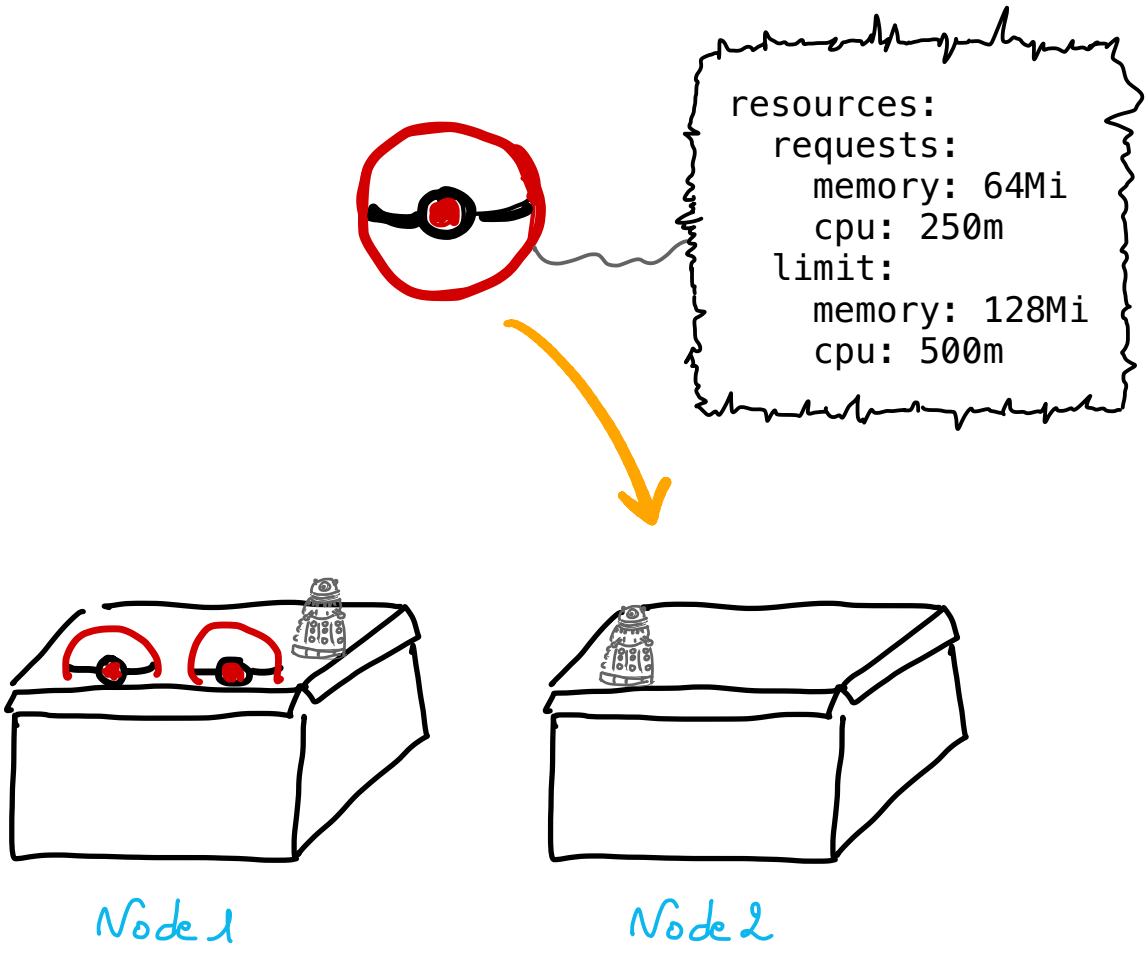


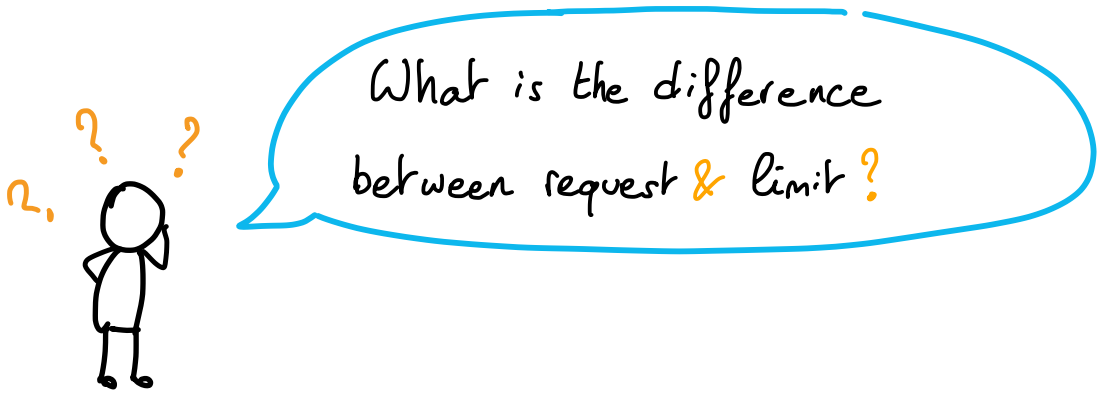
Node 1 Node 2 Node 3

→ In order to avoid this situation,
you should define requests and limits CPU & Memory

```
resources:
  requests:
    memory: 64Mi
    cpu: 250m
  limit:
    memory: 128Mi
    cpu: 500m
```

→ Scheduler uses these information to decide in which Node the Pod can be





REQUEST

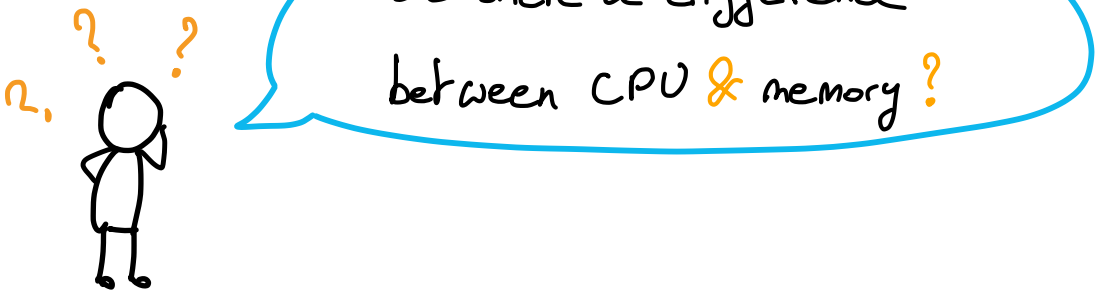
=

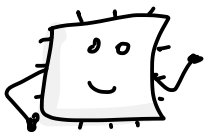
Minimum needed for Pod to be scheduled

LIMIT

=

Limit maximum usage

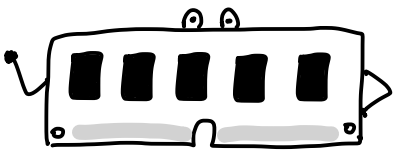




CPU is a **compressible** resource.

Once a container reaches the limit, it will keep running.

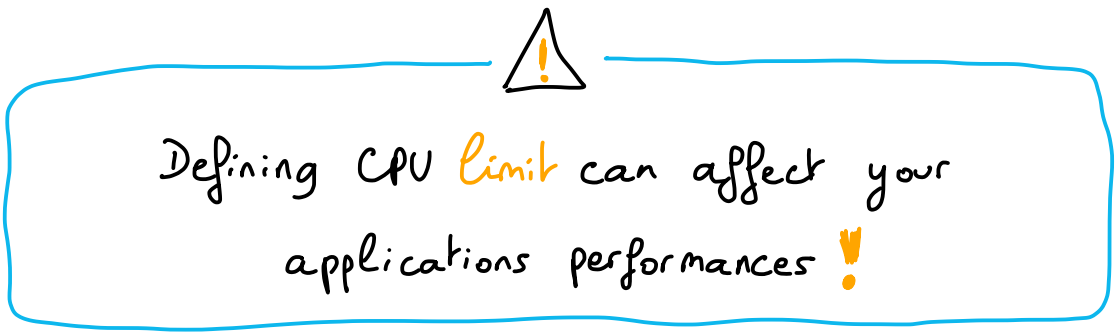
Defined in millicore: 1 core = 1000m



Memory is a **none compressible** resource.

Once a container reaches the memory limit, it will be terminated (OOM Killed).

Defined in bytes.


Defining CPU limit can affect your
applications performances !

› Create a Pod with defined requests & limits

```
apiVersion: v1
kind: Pod
metadata:
  name: my-app
spec:
  containers:
  - name: my-container
    image: my-image:1.0.0
    resources:
      requests:
        memory: 64Mi
        cpu: 250m
      limit:
        memory: 128Mi
        cpu: 500m
```

} can't be lower than requests !

› Display Pod's memory & CPU consumption

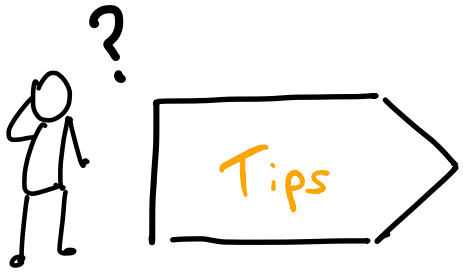
```
$ kubectl top pod
```

› Display Containers's memory & CPU consumption

```
$ kubectl top pod --containers
```

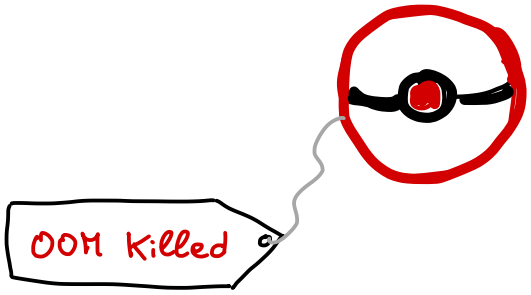
› Display Node's memory & CPU consumption

```
$ kubectl top node
```



→ Requests and limits should be defined for each individual Pods, in order to be scaled correctly by Horizontal Pod Autoscaler (HPA)

→ If Pods have been killed by OOM Killer, it will have the status OOM Killed.



I defined 3 replicas for my Deployment, is it sufficient for my application availability?

Unfortunately, no ...

→ PDB is a way to increase application availability

→ PDB provides protection against voluntary evictions :

↝ Node drain

↝ Rolling upgrade

↝ Delete a Deployment

↝ and ... delete a Pod :)

My configuration:

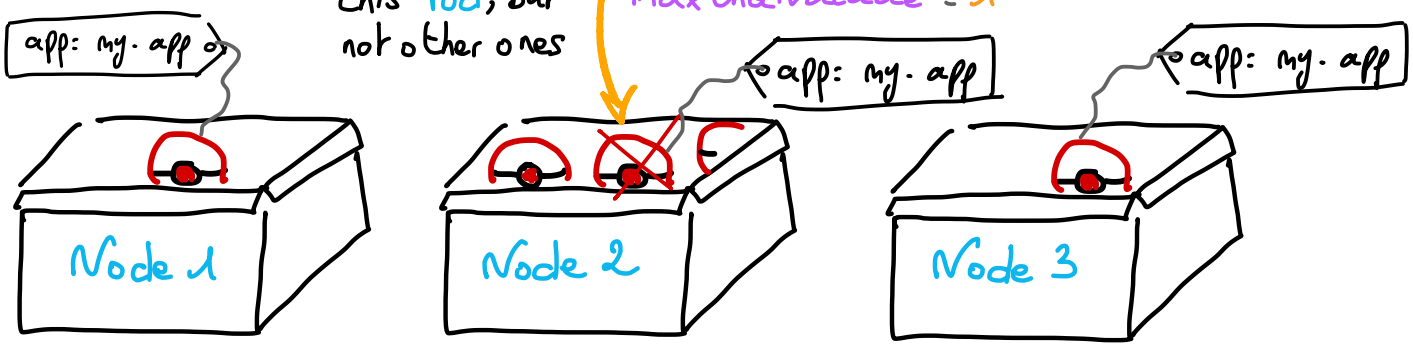
Deployment: .spec.replicas: 3

PDB: maxUnavailable: 1

Eviction API

I can delete this Pod, but not other ones

because maxUnavailable = 1



PDB cannot prevent involuntary disruptions

› Create a PodDisruptionBudget (PDB) that allow only one disruption

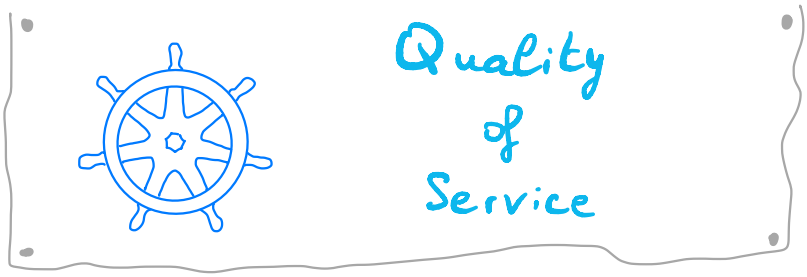
```
apiVersion: policy/v1beta1
kind: PodDisruptionBudget
metadata:
  name: ingress-nginx-controller-pdb
spec:
  minAvailable: 1
  maxUnavailable: 1
  selector:
    matchLabels:
      app: my-app
```

} how many Pods must always be available
} how many Pods can be evicted
} match only Pods with label app: my-app

It's also possible to set minAvailable as a percentage: `minAvailable: 50%`

› Show PDB in my namespace

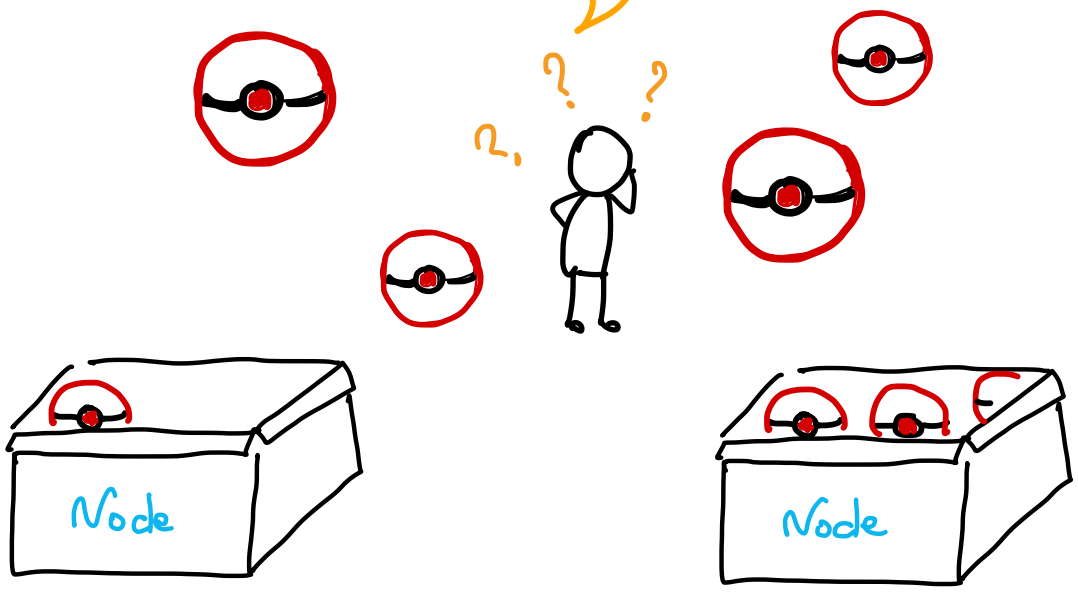
```
$ kubectl get pdb -n my-namespace
```



→ Scheduler uses requests to make decisions about scheduling Pods onto Nodes.

This Pod needs 300Mi of memory,
OK I can put it in this Node.

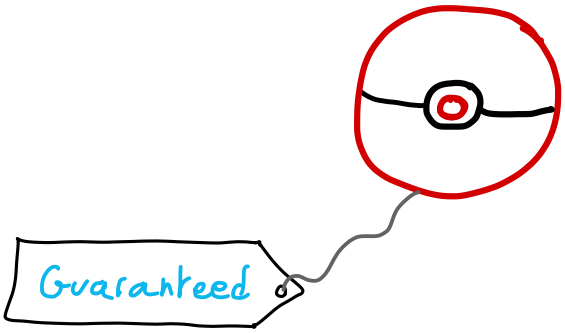
This Pod haven't any requests and limits, I don't know if it's a big app or not.



→ Depending on requests and limits defined, a Pod is classified in a certain QoS class.

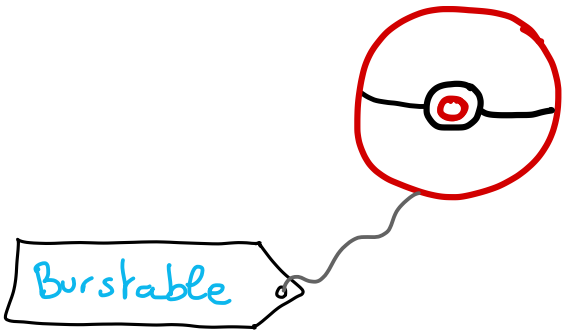
Classes

3 QoS classes :

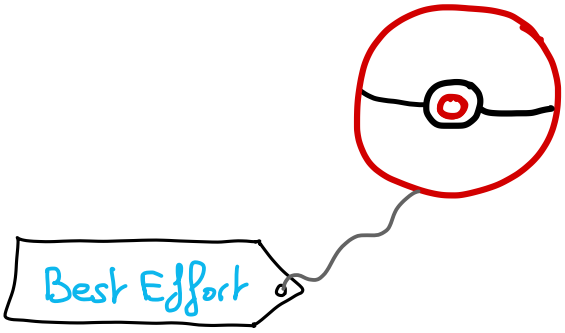


› Scheduler assigns Guaranteed Pods only to Nodes which have enough resources compliant with CPU & memory requests.

› Pods are guaranteed to not be killed until their exceed their limits.



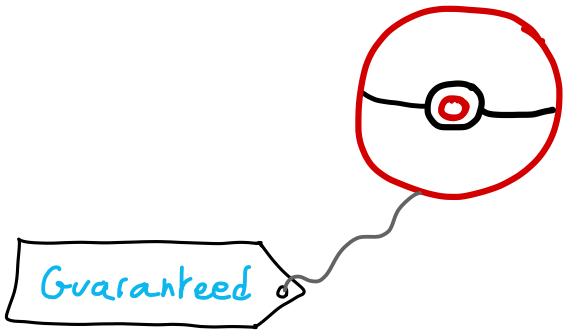
- When they reached their limit,
  these Pods are killed before Guaranteed ones,
  if there aren't any Best Effort Pods,



- Containers can use any amount of free memory
  and CPU in the Node.

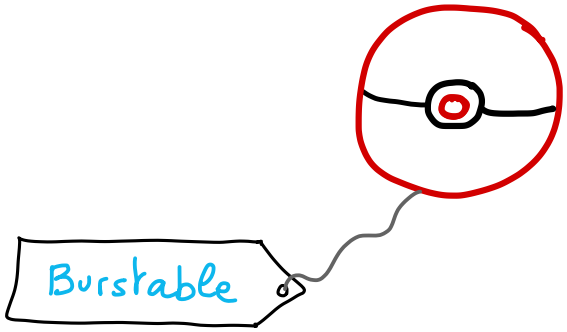
- If system don't have enough resources,
  Pods with Best Effort QoS will be the first killed.

Assign a specific QoS class to a Pod



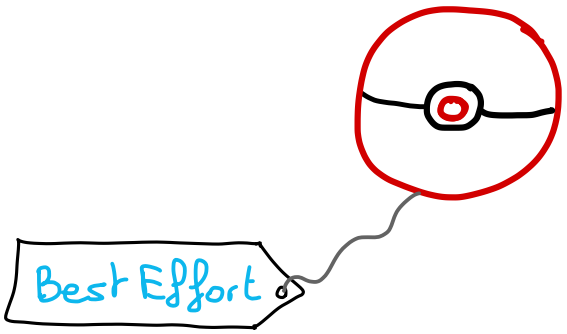
- Requests and limits must be equal in all containers (even Init containers)

```
apiVersion: v1
kind: Pod
metadata:
  name: my-app
spec:
  containers:
  - name: my-container
    image: nginx
    ports:
    - containerPort: 80
    resources:
      requests:
        memory: "64Mi"
        cpu: "20m"
      limit:
        memory: "64Mi"
        cpu: "20m"
```



- At least one container in the Pod must have a memory or CPU request defined

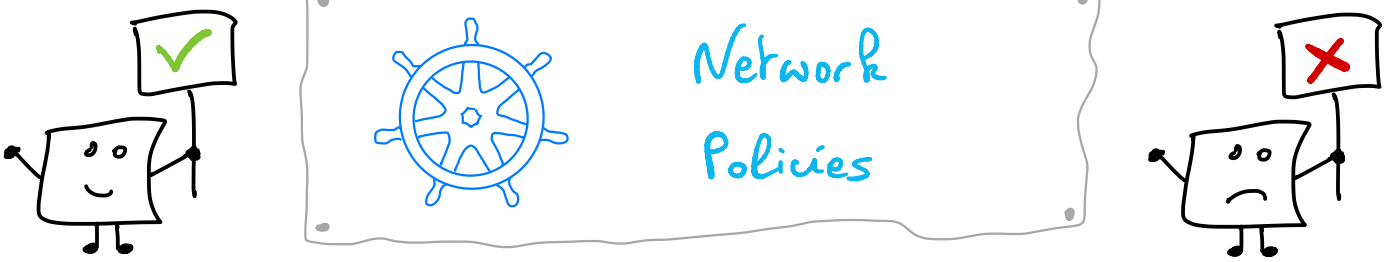
```
apiVersion: v1
kind: Pod
metadata:
  name: my-app
spec:
  containers:
  - name: my-container
    image: nginx
    ports:
    - containerPort: 80
    resources:
      requests:
        memory: "64Mi"
      limit:
        memory: "128Mi"
```



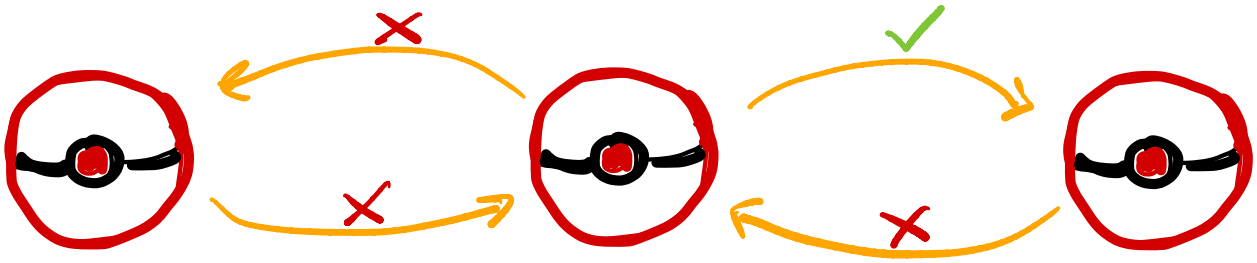
- If request and limit are not set, for all containers Pods will be treated as lowest priority and Pods are classified as Best Effort.

> Get QoS class for my-pod Pod

```
$ kubectl get pod my-pod -o 'jsonpath={...qosClass}'
```



Network Policies allow to control communication between Pods in a cluster



→ Scoped by Namespaces

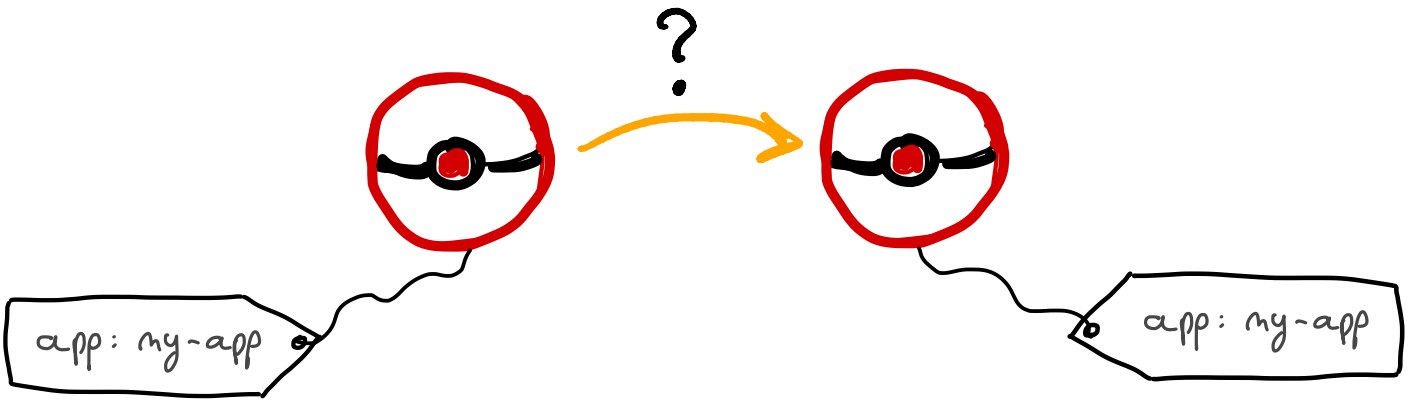
→ Stable since 1.7 (June 2017!)

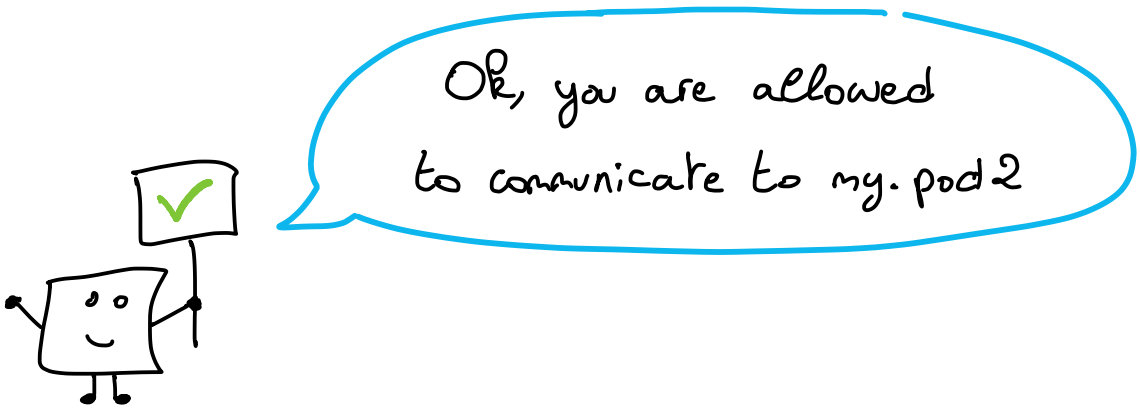
→ Limited to IP addresses (no domain name)

→ Pre-requisites:

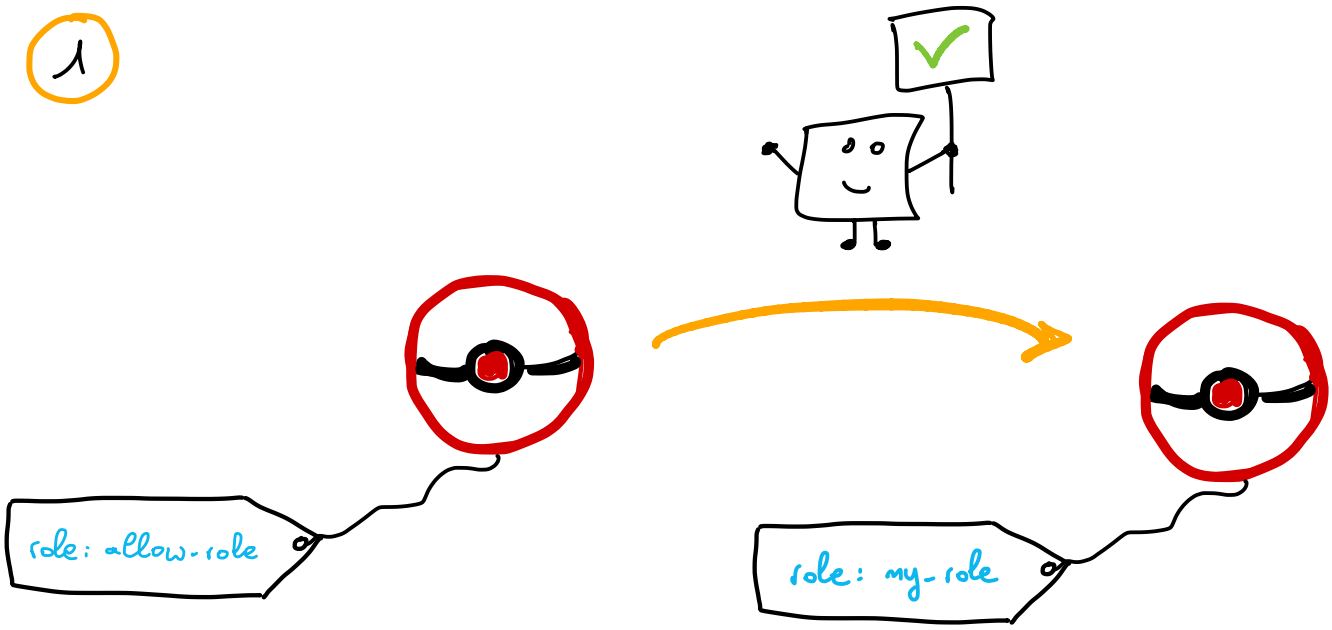
Use a CNI plugin which supports NP

→ Network Policies use labels to select Pods and define rules which specify what traffic is allowed





1



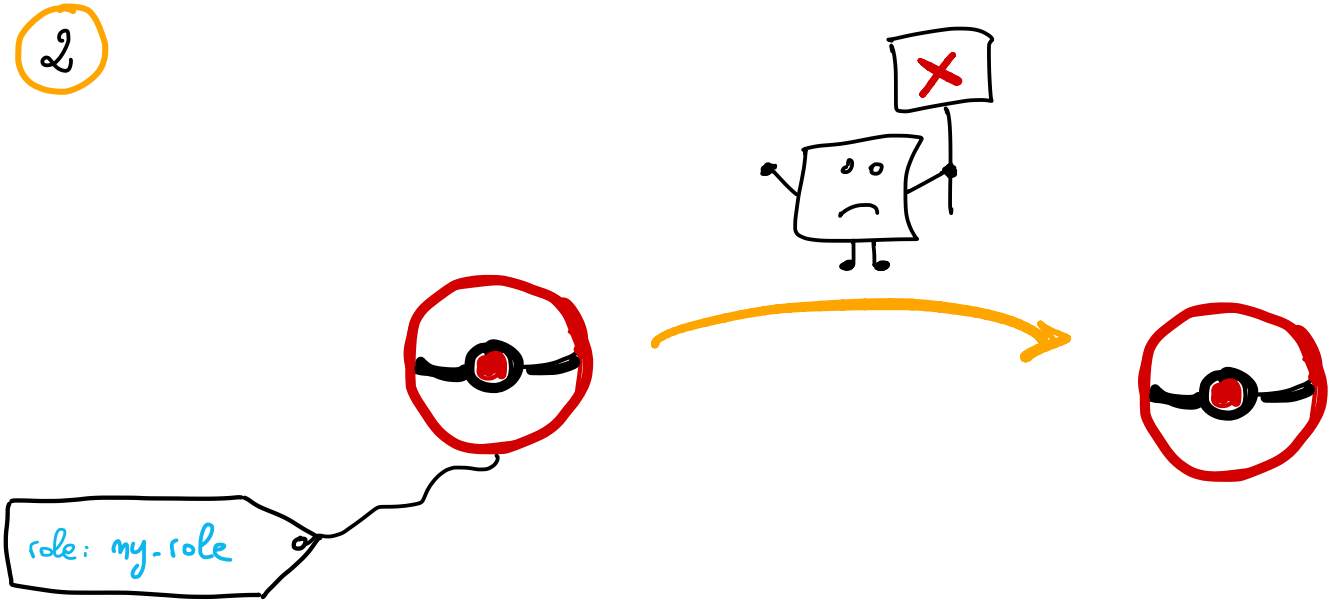

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: test-network-policy
  namespace: my-namespace
spec:
  podSelector:
    matchLabels:
      role: my-role
  policyTypes:
    - Ingress
    - Egress
  ingress:
    - from:
        - podSelector:
            matchLabels:
              role: allow-role
      ports:
        - protocol: TCP
          port: 6379
```

select Pods

⚠ If empty podSelector: all Pods in namespace

Allow connections to all Pods labels role: allow-role to communicate to our Pods

2




```
egress:
  - to:
      - podSelector:
          matchLabels:
            role: allow-to-role
    ports:
      - protocol: TCP
        port: 5978
```

Allow connections

for our Pods

to communicate to Pods

with labels role: allow-to-role

When a NetworkPolicy selects a group of Pods,

targeted Pods become isolated

& reject all traffic not defined in any NP

3 Deny all Ingress traffic to our Pods

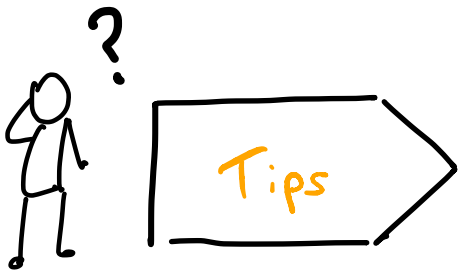
```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny-ingress
  namespace: my-namespace
spec:
  podSelector: {}
  policyTypes:
    - Ingress
```

} all Pods in my-namespace

4 Allow all Egress traffic from our Pods

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-all-egress
  namespace: my-namespace
spec:
  podSelector: {}
  policyTypes:
    - Egress
  egress:
  - {}
```

} allow all

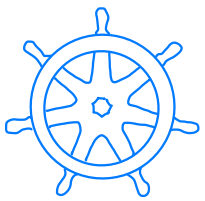


→ In Ingress & Egress, several selectors exists:

podSelector

namespaceSelector

ipBlock



RBAC

Role - Based Access Control

→ Method of regulating access to resources based on the roles of individual users

→ Useful when you want to control what your users can do for which kind of resources in your cluster

→ Introduces 4 Kubernetes objects

- Role
- ClusterRole
- RoleBinding
- ClusterRoleBinding

→ Operations (verbs) allowed on resources:

- create
- list
- get
- update
- delete
- watch

Role → List of verbs allowed on specific resources

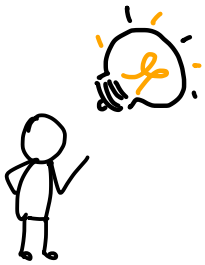
→ Sets permissions within a given namespace

ClusterRole → bigger power than Role

→ Grants permissions also but ... cluster-wide

&

- cluster-scoped resources (Node ...)
- non-resource endpoint (/healthz ...)
- namespaced resources (like Pods) accross all namespaces



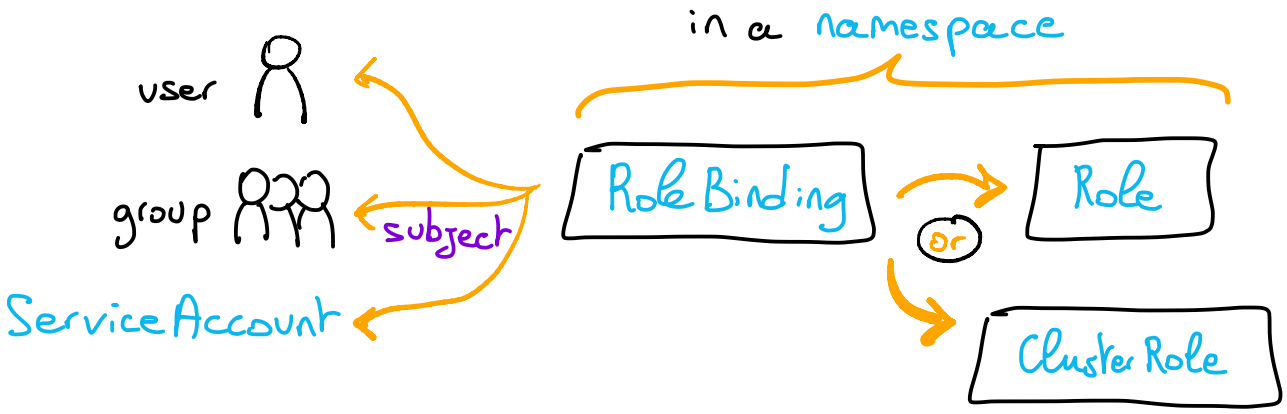
Yes, Role & ClusterRole objects do the same things, but their scopes are different

A Kubernetes object is either namespaced or not, but not both !

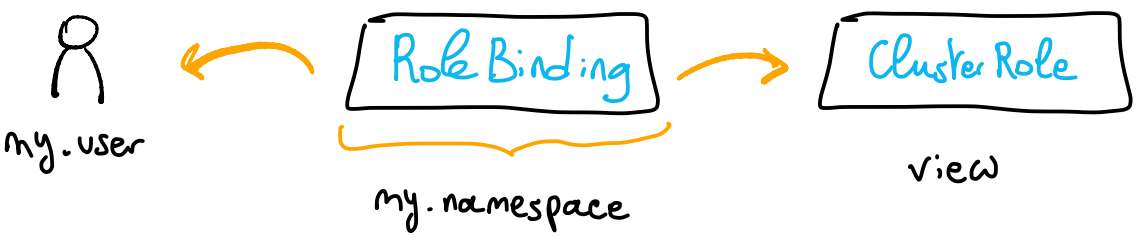
RoleBinding — Link between a Role and a subject

→ Grants the permissions defined in a Role to a user or a set of users within a specific namespace

→ A RoleBinding can reference any Role in the same namespace



→ A RoleBinding should be defined per namespace



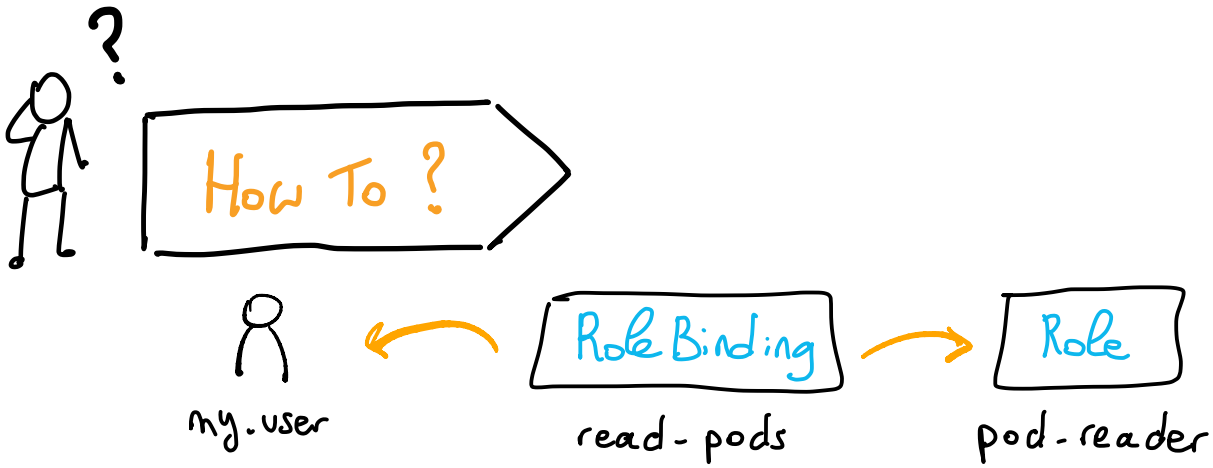
ClusterRoleBinding

→ Same as RoleBinding but for all namespaces (cluster-wide)

Allow any users in the group my-group to read secrets in any ns

ClusterRoleBinding → ClusterRole: read-secret

Several ClusterRole can be aggregated into one thanks to aggregationRule



1 Create a Role that grants read Pods access in my-namespace namespace

```yaml
apiVersion:
rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: my-namespace
  name: pod-reader
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "watch", "list"]
```

read access

2 Create a RoleBinding that grants pod-reader Role to a user

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: read-pods
  namespace: my-namespace
subjects:
- kind: User
  name: my-user
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: pod-reader
  apiGroup: rbac.authorization.k8s.io
```

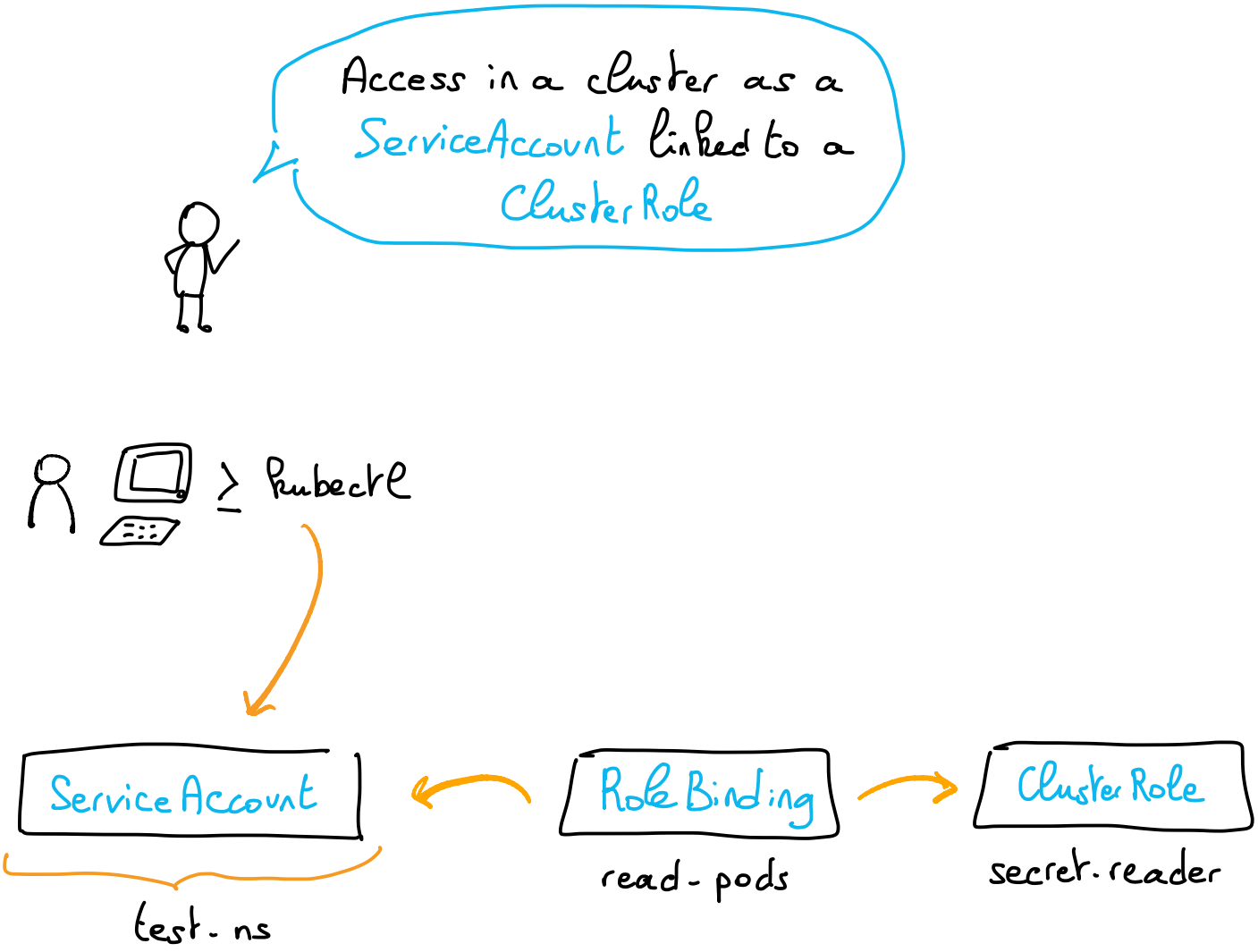
1. Create a ClusterRole that grants read secrets access in all namespaces in the cluster

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: secret-reader
rules:
- apiGroups: [""]
  resources: ["secrets"]
  verbs: ["get", "watch", "list"]
```

⚠ No namespace in metadata

2. Create a ClusterRoleBinding that grants secret-reader ClusterRole to a group

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: read-secrets-global
subjects:
- kind: Group
  name: my-group
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: secret-reader
  apiGroup: rbac.authorization.k8s.io
```



1. Create a namespace "test-ns"

```
$ kubectl create ns test-ns
```

2. Create a ServiceAccount in this namespace

```
$ kubectl create serviceaccount my-sa-test -n test-ns
```

3. Create a RoleBinding that grant secret-reader to my-sa.test

```
$ kubectl create rolebinding my-sa-test-rolebinding -n test-ns
--clusterrole=secret-reader --serviceaccount=test-ns.my-sa-test
```

4 Create a Kubeconfig file for the created ServiceAccount

```
$ export TEAM_SA=my-sa-test
$ export NAMESPACE_SA=test-ns

$ export SECRET_NAME_SA=`kubectl get sa ${TEAM_SA} -n $NAMESPACE_SA
-ojsonpath="{ .secrets[0].name }"`
$ export TOKEN_SA=`kubectl get secret $SECRET_NAME_SA
-n $NAMESPACE_SA -ojsonpath='{.data.token}' | base64 -d`

$ kubectl config view --raw --minify > kubeconfig.txt
$ kubectl config unset users --kubeconfig=kubeconfig.txt
$ kubectl config set-credentials ${SECRET_NAME_SA}
--kubeconfig=kubeconfig.txt --token=${TOKEN_SA}
$ kubectl config set-context --current --kubeconfig=kubeconfig.txt
--user=${SECRET_NAME_SA}
```

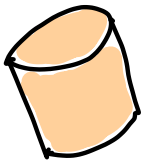
5 Execute kubectl commands in the cluster as the ServiceAccount

```
$ kubectl --kubeconfig=kubeconfig.txt get secrets -n test-ns
```

General

→ Set of conditions/rules a Pod must follow in order to be accepted on the cluster

→ A policy can contain rules on :



Types of volumes

Priviledged Pods

Read. only filesystem

volume mount GID

Kernel authorized capabilities (sysctl ...)

hostPort

Privileges elevation (= sudo)

UID / GID used by Pod

Seccomp / AppArmor profiles

→ By default, a Kubernetes cluster accepts all Pods

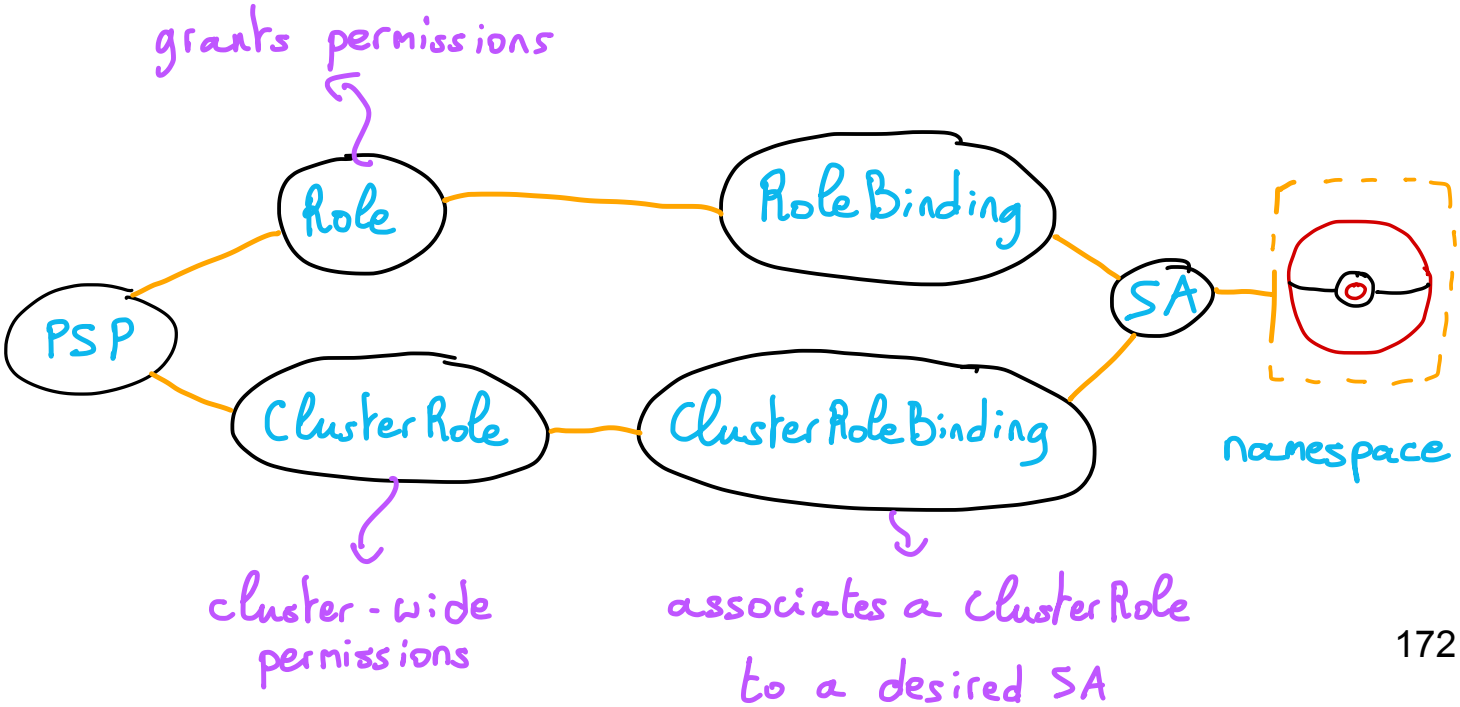
→ So, first you need to create policies

You'll need to enable PodSecurityPolicy admission controller on your cluster in order to use them ☺

→ ⚠ When PSP are activated, every Pod that wants to run on the cluster, needs to use at least one policy

In Practice

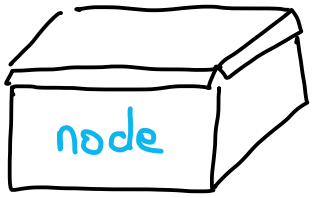
→ A Pod is linked to PSP thanks to his ServiceAccount

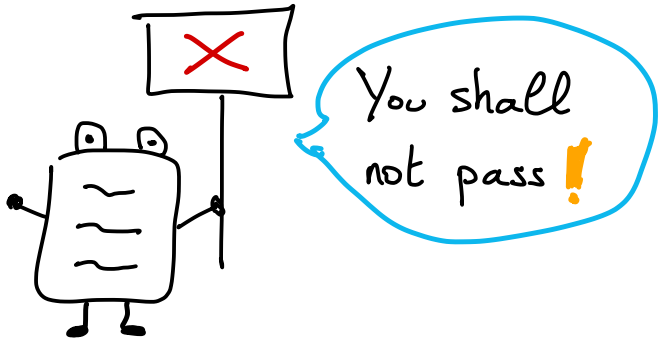


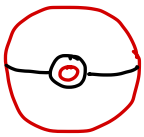
→ ⚠ Only one PSP for a Pod can be applied

→ PSP controller accept or refuse Pods in the cluster

Mutable & Non mutable policies

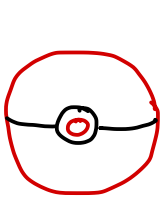


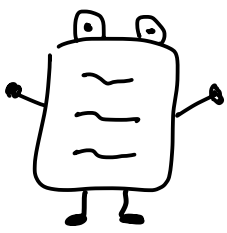




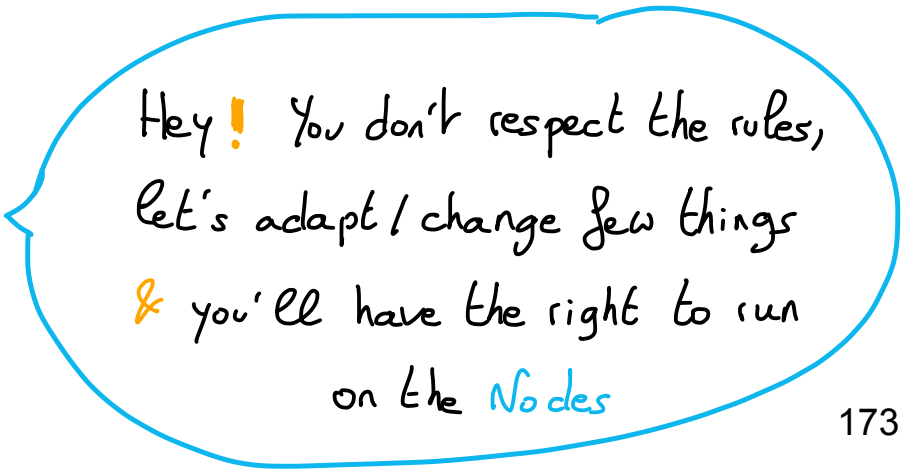
non mutable policy

→ don't change the Pod



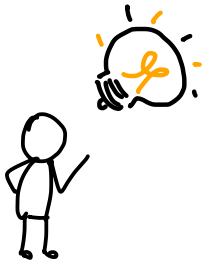


mutable policy



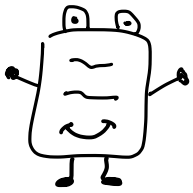
Policy order

But, if several PSP exists, how is it working?

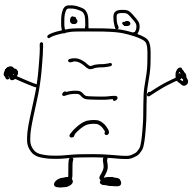


Pod security policy controller

1. choose non mutable policies first

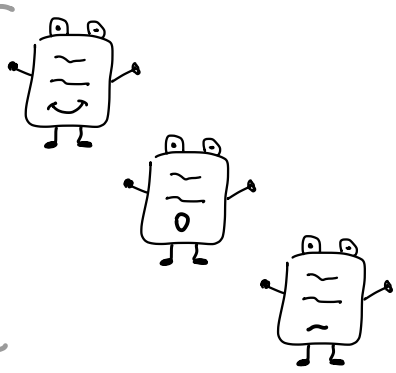


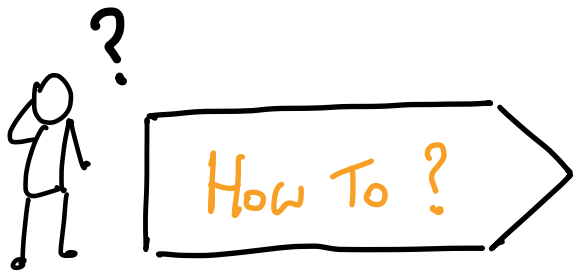
non mutable



mutable

2. if several mutable policies matches, choose one ordered by name (alphabetically)





1. Define a policy named "my-policy" that prevents the creation of privileged Pods

```
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
  name: my-psp
spec:
  privileged: false
  seLinux:
    rule: RunAsAny
  supplementalGroups:
    rule: RunAsAny
  runAsUser:
    rule: RunAsAny
  fsGroup:
    rule: RunAsAny
  volumes:
  - '*'
```

`privileged: false` → Don't allow the creation of privileged Pods

other control aspects

Allow access to all available volumes

2. Create a ClusterRole

```
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: my-cluster-role
rules:
- apiGroups:
  - policy
  resources:
  - podsecuritypolicies
  verbs:
  - use
  resourceNames:
  - my-psp
```

4. policy
3. which is a PSP
1. Grant access
2. to my-psp

3 Create a RoleBinding linked to a desired ServiceAccount (SA)

```yaml
# Bind the ClusterRole to the desired set of service accounts.
# Policies should typically be bound to service accounts in a
namespace.
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: my-rolebinding
  namespace: my-namespace
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: my-cluster-role
subjects:
- kind: ServiceAccount
  name: default
  namespace: my-namespace
```

bind my-cluster-role

to the SA default in my-namespace ns

→ which accounts the ClusterRole is bound

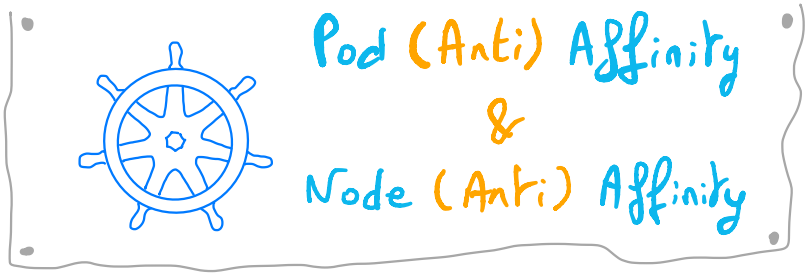
OR

Create a RoleBinding linked to all ServiceAccounts in my-namespace

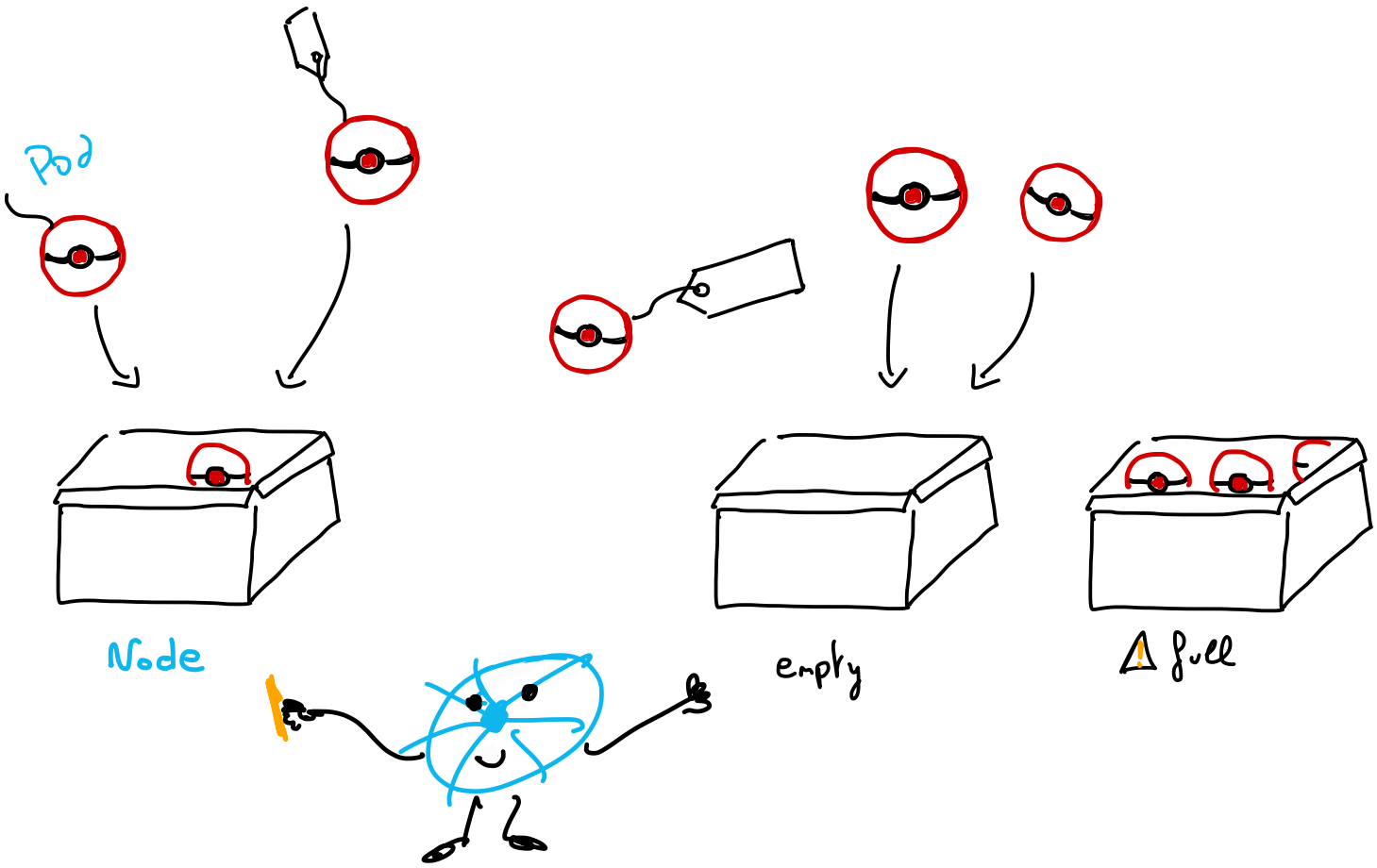
```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: my-rolebinding
  namespace: my-namespace
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: my-cluster-role
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: system:serviceaccounts
```

4 Enable PodSecurityPolicy admission controller in an existing GKE cluster

```
$ gcloud beta container clusters update my-cluster
--enable-pod-security-policy
```

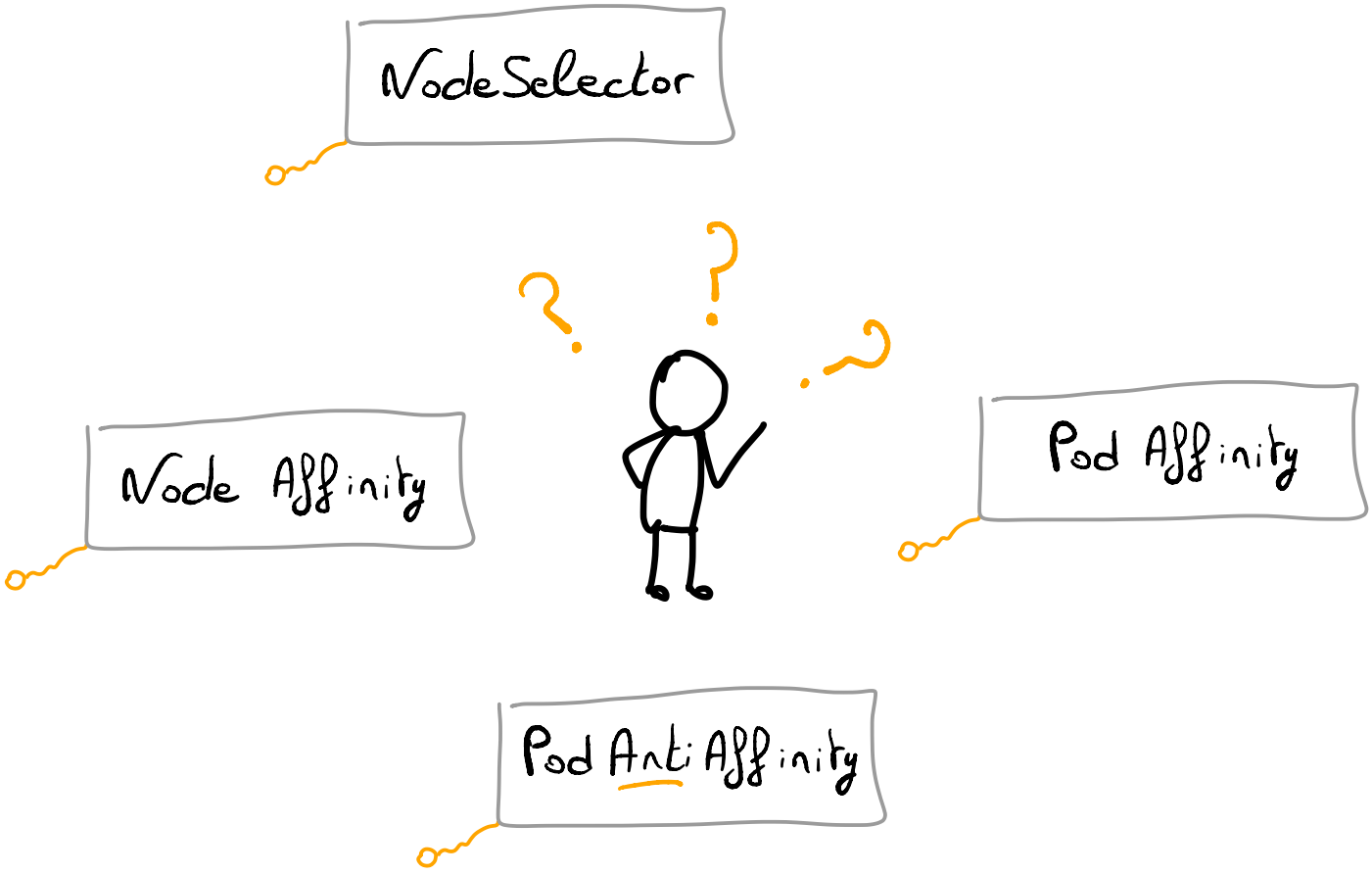


→ By default, when you deploy a Pod, Kubernetes chooses the right Node according to its need

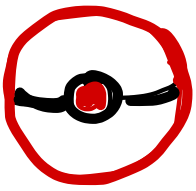


But do you know that you can control Pod and Node (anti) affinities?

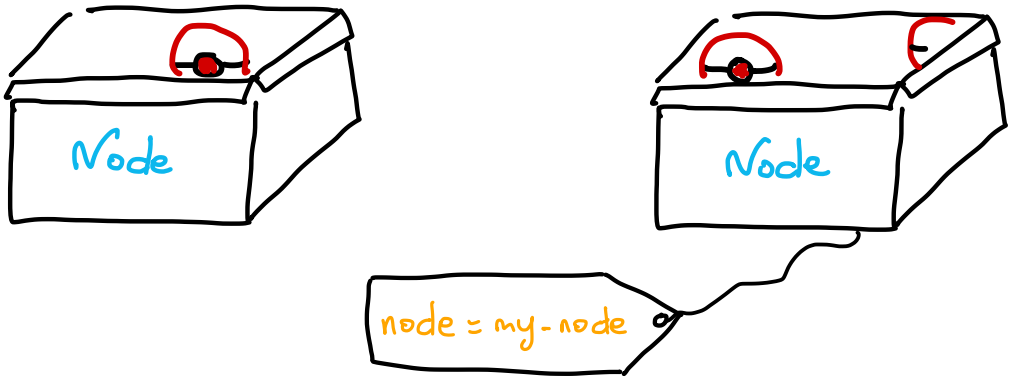
→ Several ways to handle node and pod affinities :



Node Selector



I want to run in a Node with label "node = my-node"



By default, Nodes have pre-defined labels but you can add your own labels to force a Pod to run on a Node.

Node Affinity

→ Similar to NodeSelector but more expressive syntax

Hard rule: Only run the Pod on Nodes that match criterias

requiredDuringSchedulingIgnoredDuringExecution

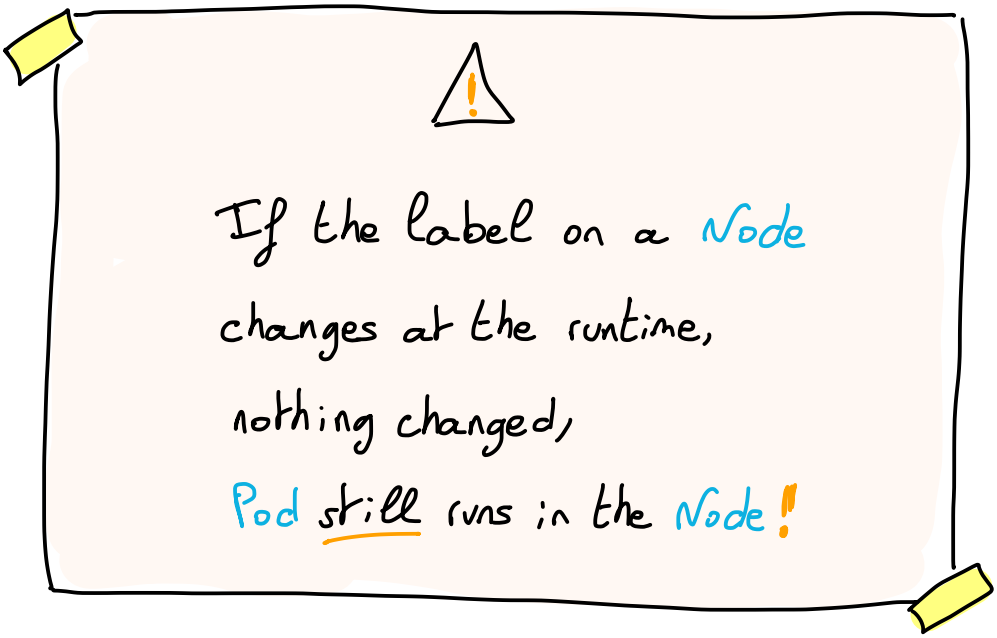
Soft rule: Try to find the wanted Node (s) If not possible, Pod will be on another Node

preferredDuringSchedulingIgnoredDuringExecution

Allowed operators:

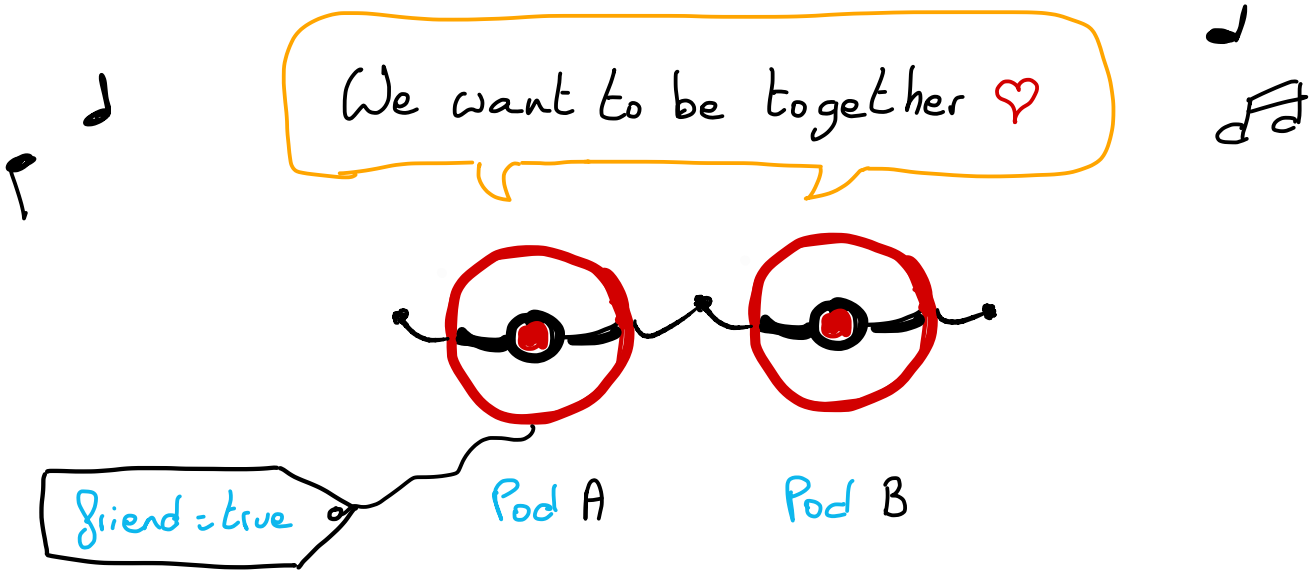
In, Exists, Gt, Lt, NotIn, DoesNotExists

"Node Anti-Affinity"



Pod Affinity

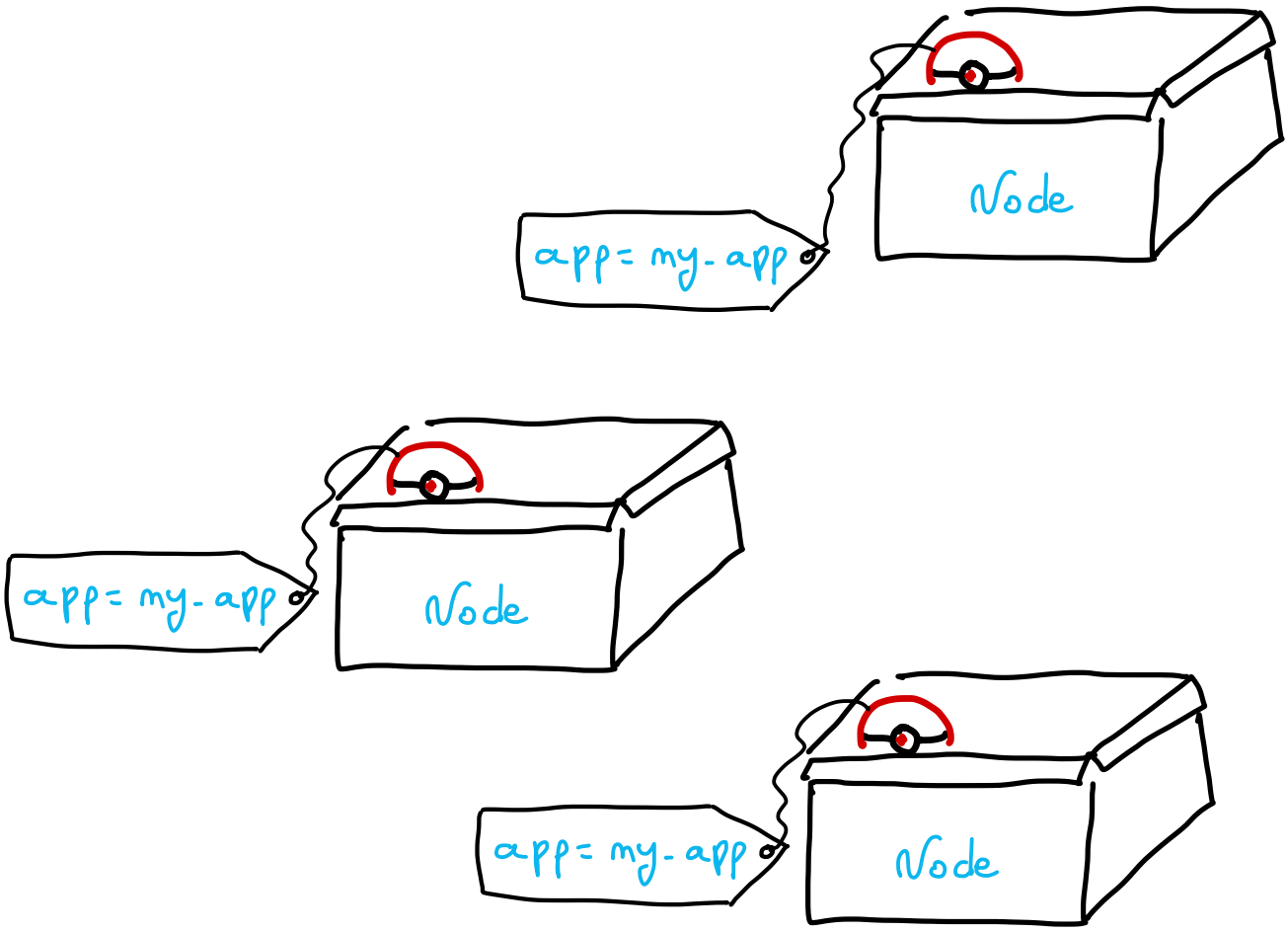
→ Allow to run a Pod on the same Node than another Pod



Useful for Pods that need to run on the same machine.

Pod Anti Affinity

→ Allow to run several replicas of a Deployment on a different Node



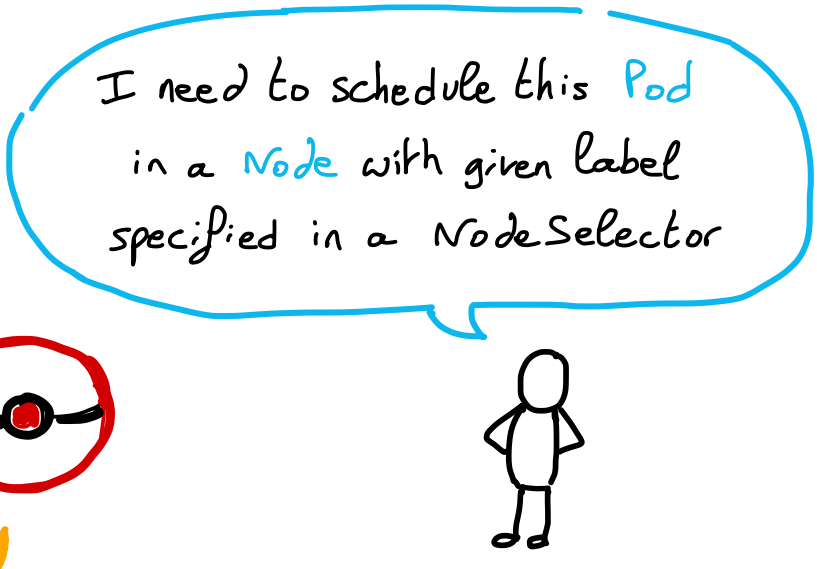
You will be able to distribute your Pods on different Nodes. If one Node died, the app will still be available ☺

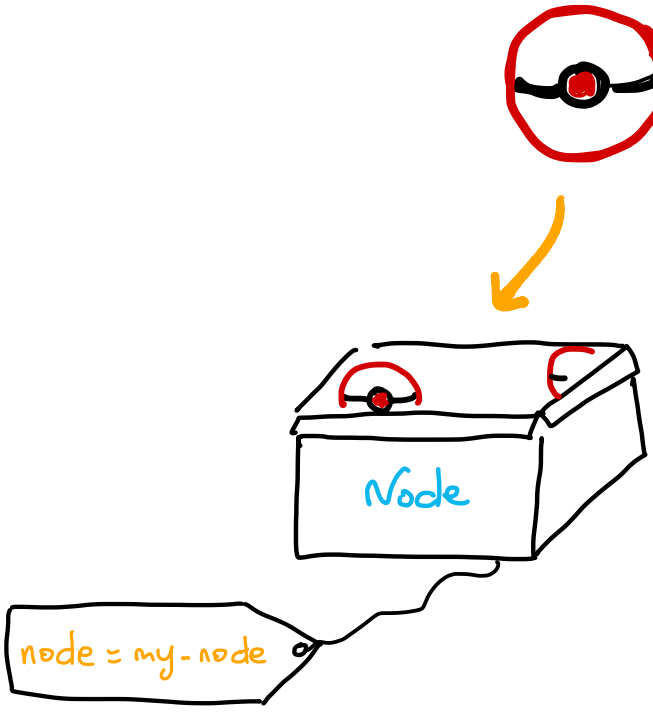
List all Nodes with their labels

```
$ kubectl get node --show-labels
```

Deploy a Pod in a Node that have a specific label

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  containers:
  - name: nginx
    image: nginx
  nodeSelector:
    node: my-node
```

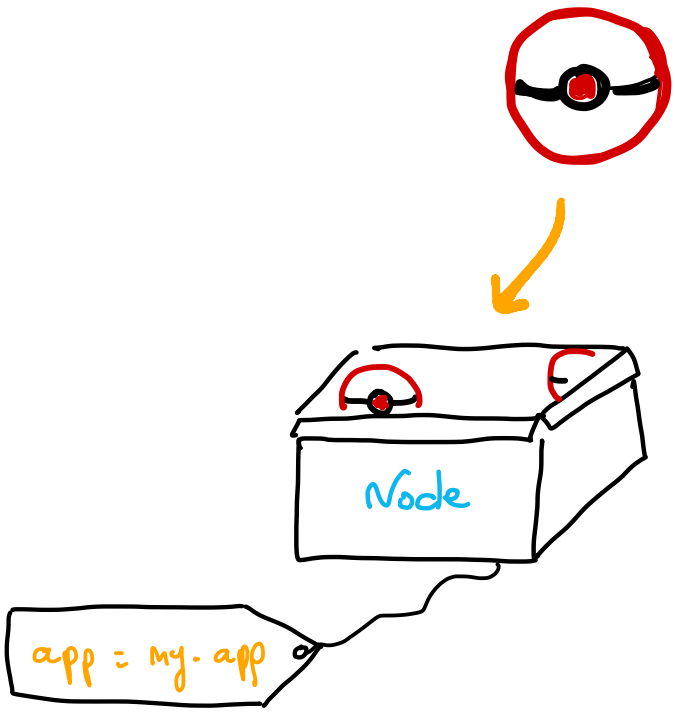




Create a Pod with Node Affinity, that will run in a Node that have a specific label

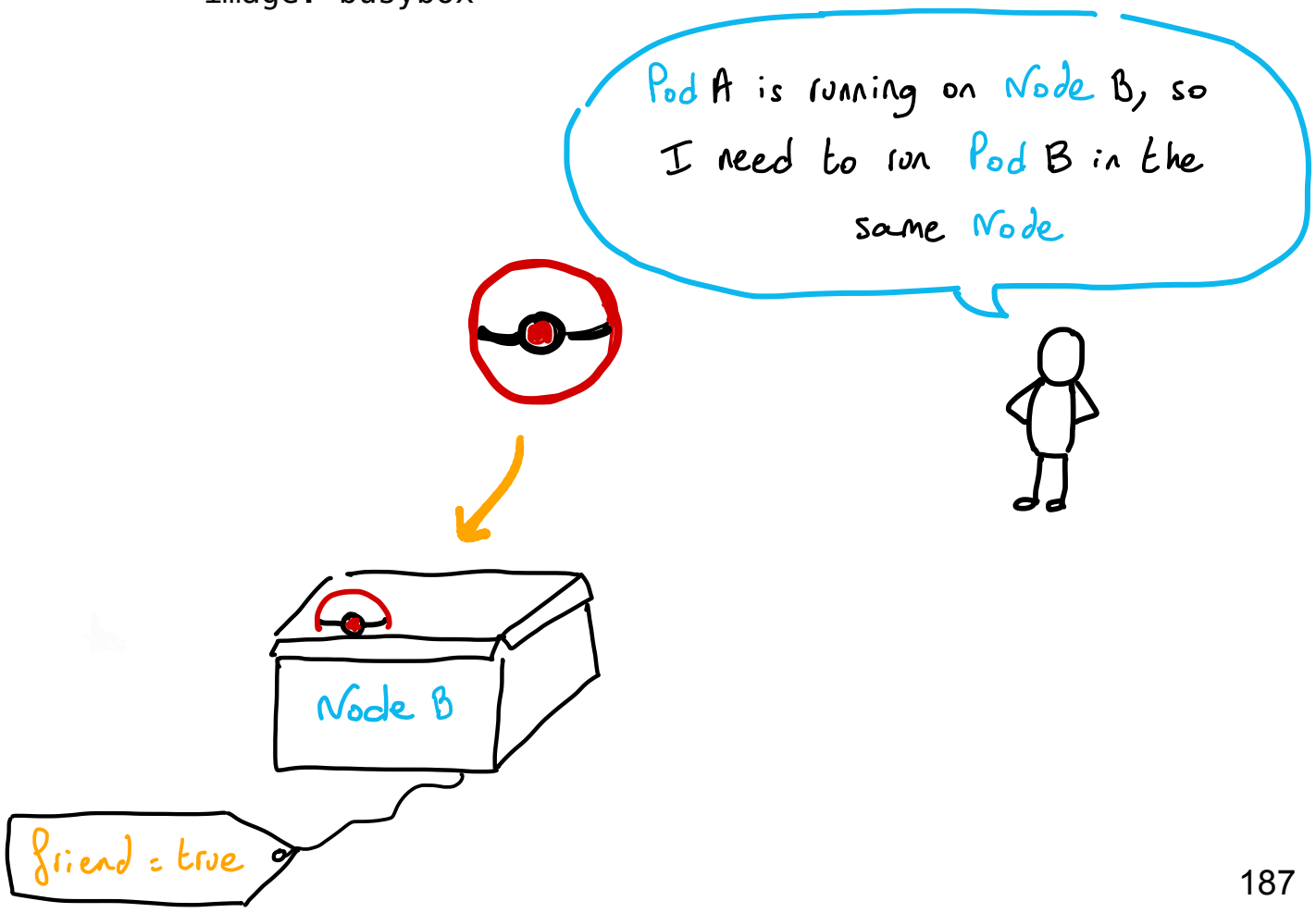
ooo if possible :)

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-pod-with-node-affinity
spec:
  affinity:
    nodeAffinity:
      preferredDuringSchedulingIgnoredDuringExecution:
      - weight: 1
        preference:
          matchExpressions:
          - key: app
            operator: In
            values:
            - my-app
  containers:
  - name: my-container
    image: busybox
```



Create a Pod that will run next to Pod with label "friend = true"

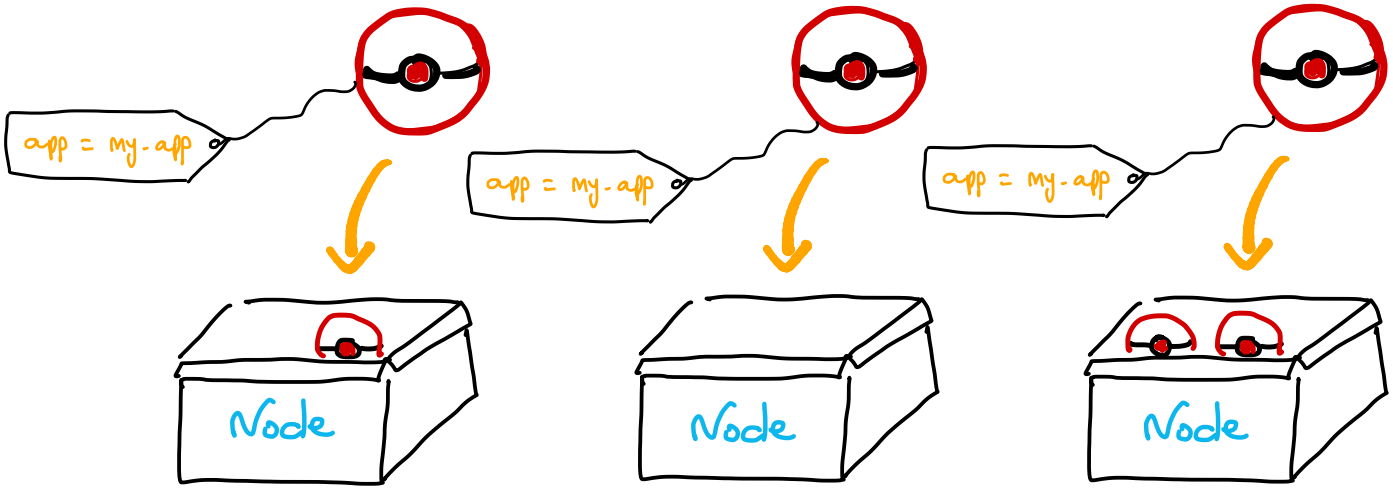
```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod-with-pod-affinity
spec:
  affinity:
    podAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
      - labelSelector:
          matchExpressions:
          - key: friend
            operator: In
            values:
            - true
        topologyKey: topology.kubernetes.io/zone
  containers:
  - name: my-container
    image: busybox
```

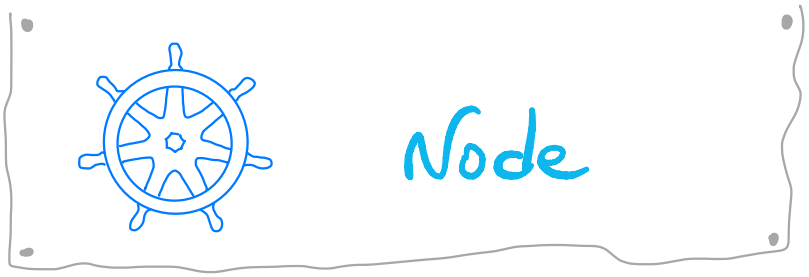


Create a Deployment with 3 replicas.

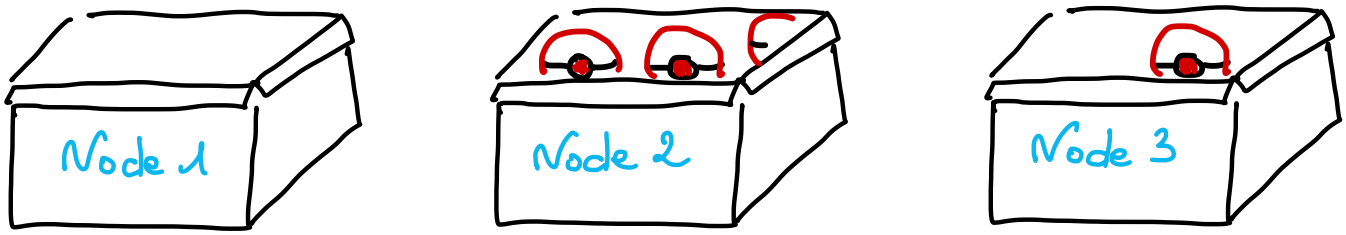
Each Pod must not run in the same Node.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-db
spec:
  replicas: 3
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      affinity:
        podAntiAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
          - labelSelector:
              matchExpressions:
              - key: app
                operator: In
                values:
                - my-app
            topologyKey: kubernetes.io/hostname
      containers:
      - image: my-image:1.0
        name: my-container
```



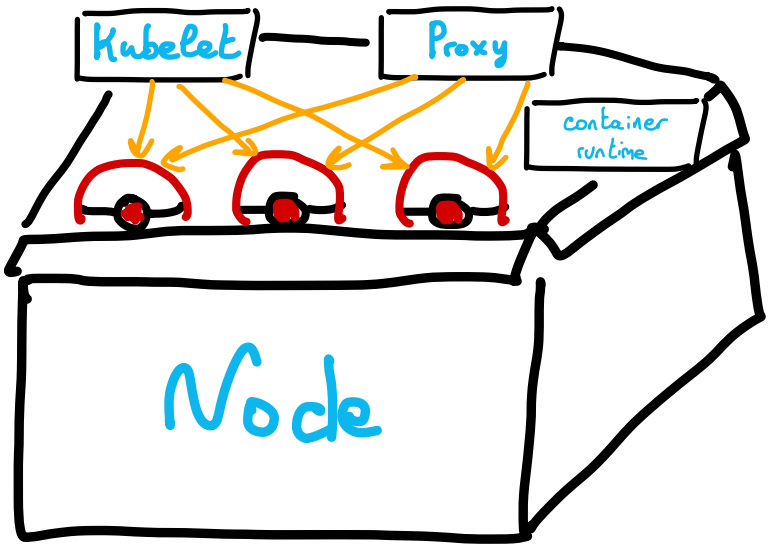


→ Pods run on a Node

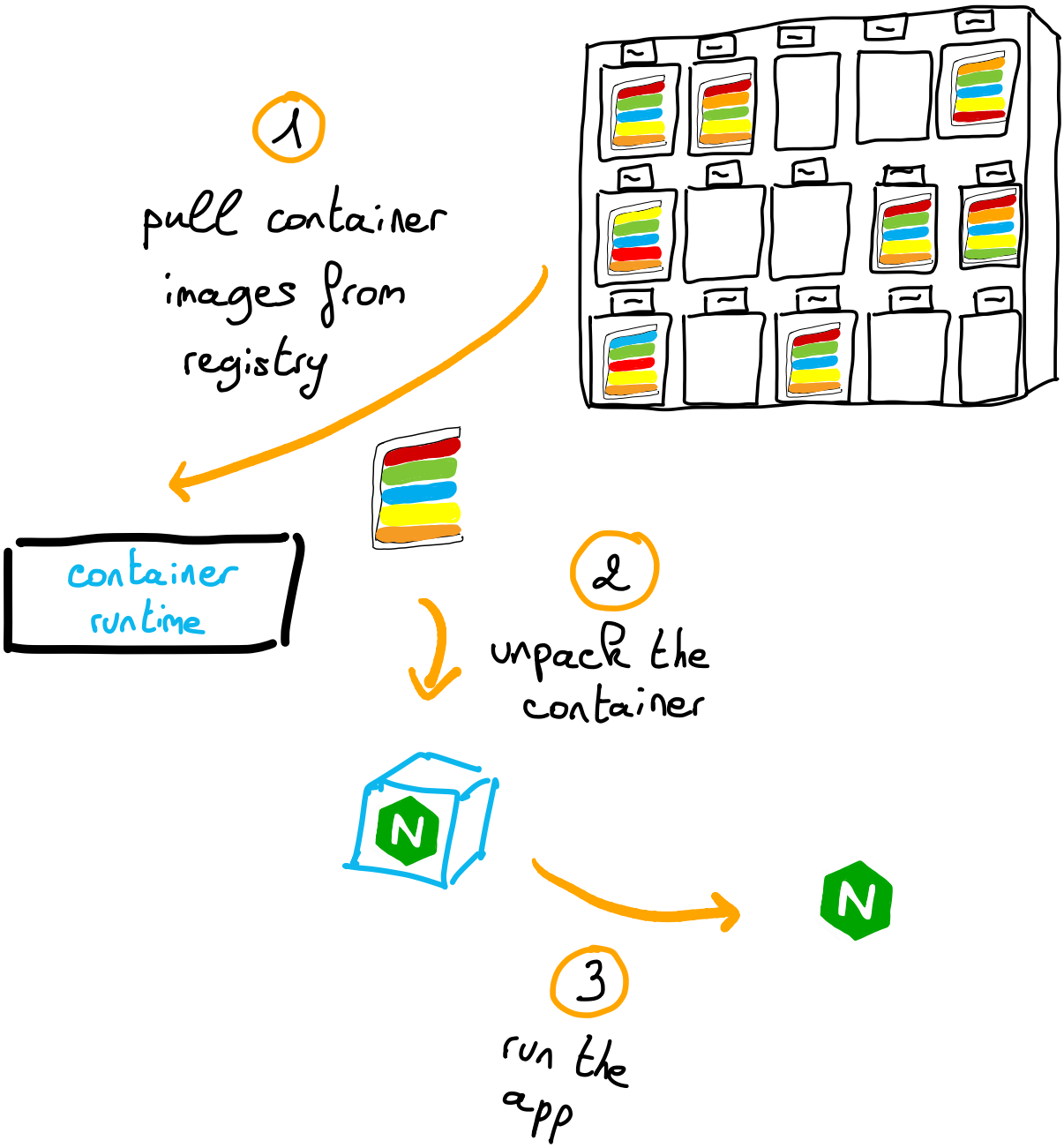


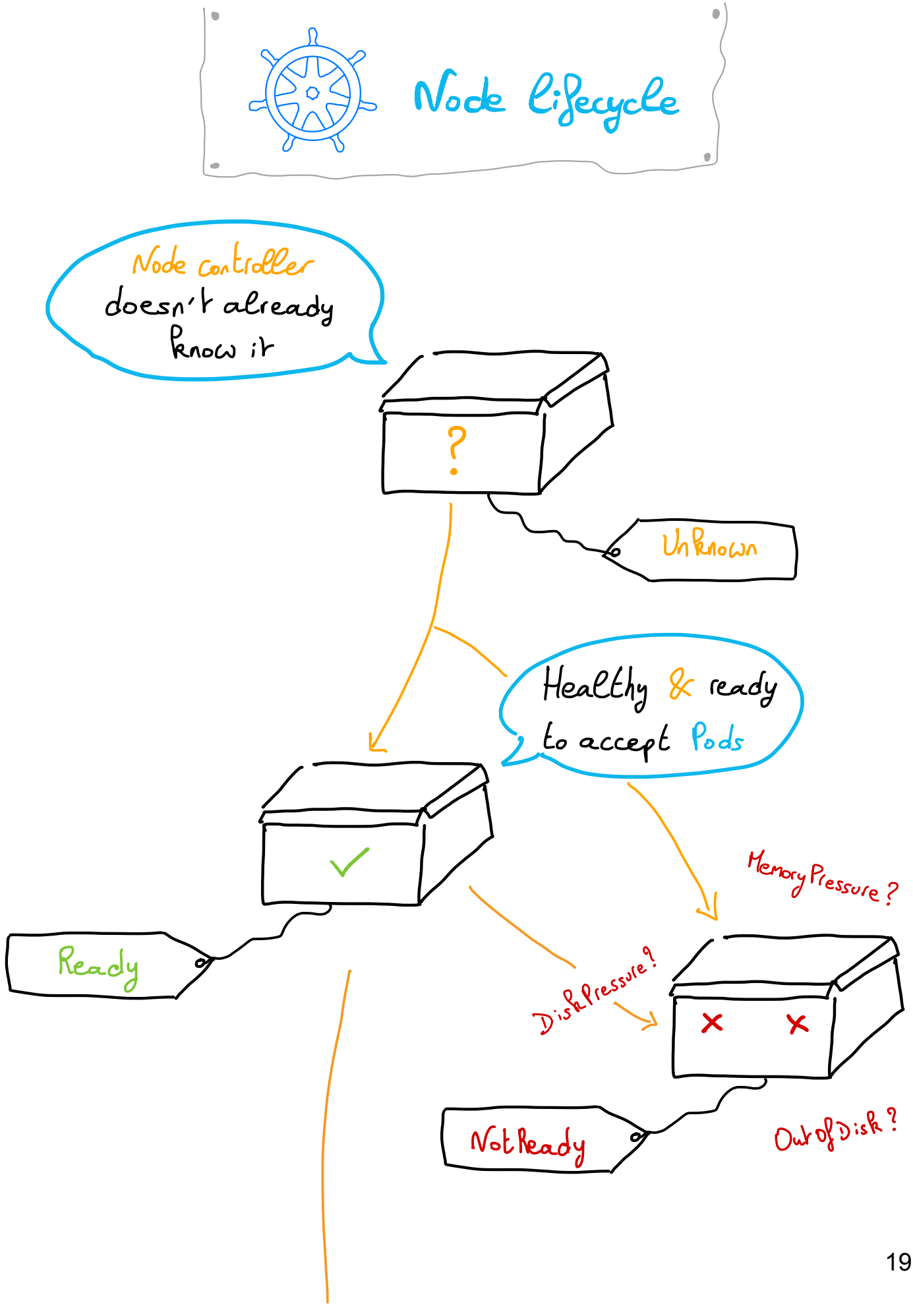
→ A Node is a physical or Virtual Machine (VM)

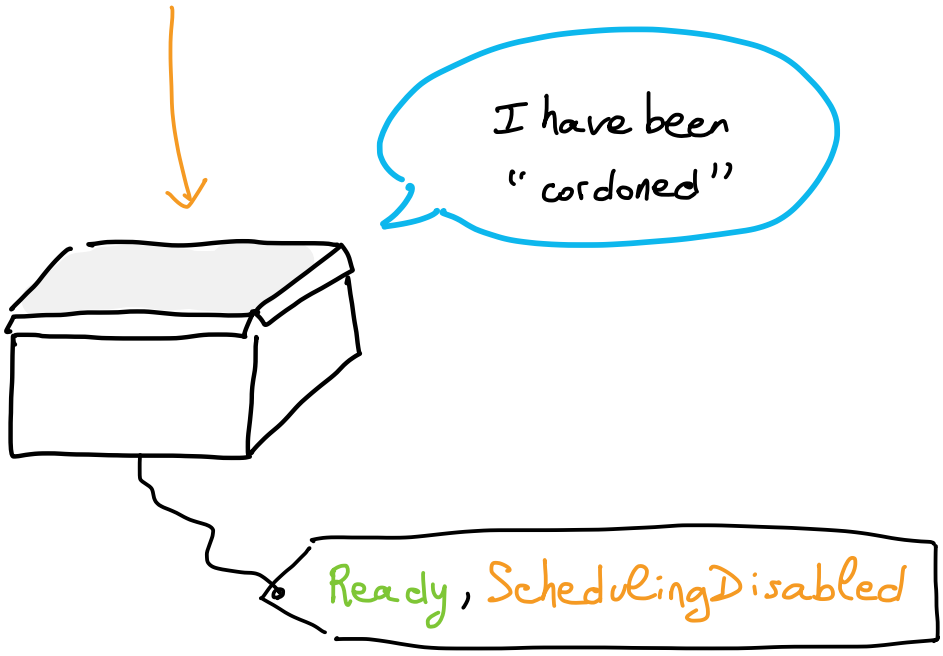
Overview



Container runtime:




Node lifecycle
Node controller doesn't already know it
?
UnKnown
Healthy & ready to accept Pods
Ready
Memory Pressure ?
Disk Pressure ?
Not Ready
Out of Disk ?


I have been
"cordoned"
Ready, SchedulingDisabled

> Show more informations about a Node

(conditions, capacity ...)

```
$ kubectl describe node my-node
```

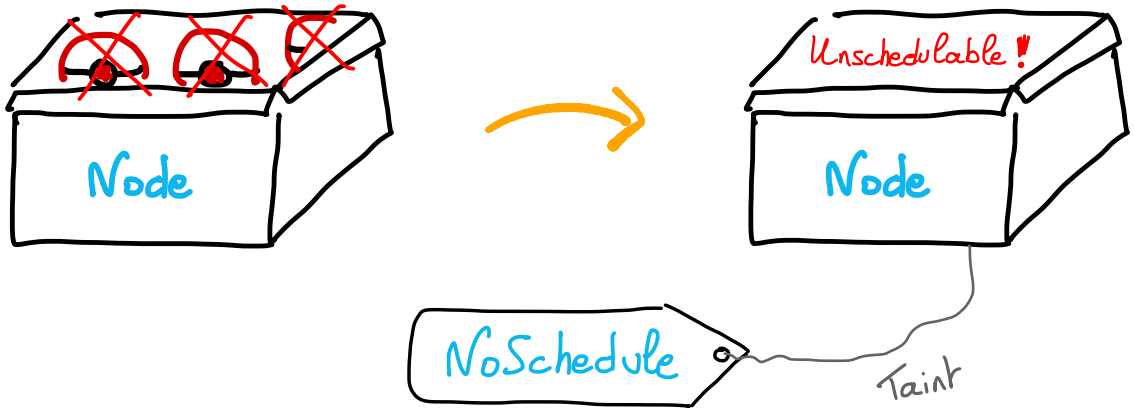
Node operations

Why?

- Debug issues on Node
- Upgrade a Node
- Scale down the cluster
- Reboot the Node

...

Drain



→ Evict all Pods from a Node & mark the Node as unschedulable

```
$ kubectl drain my-node
```

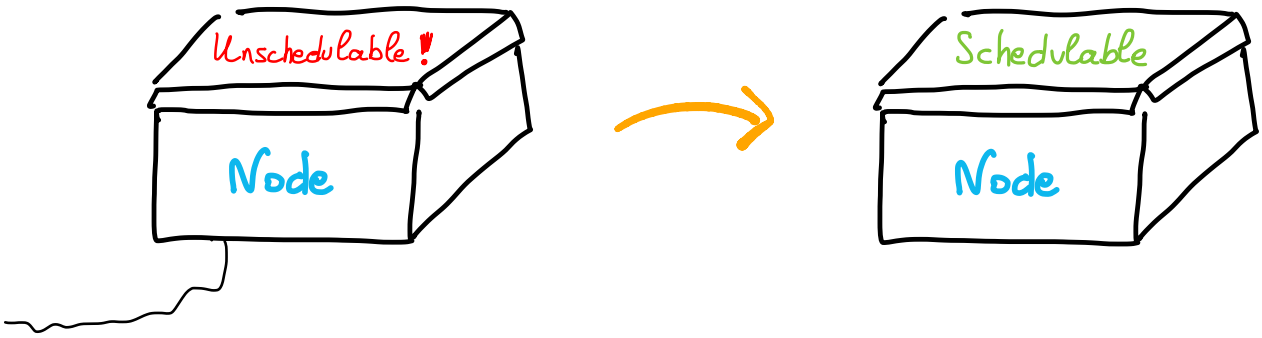
→ Evict all Pods, also DaemonSets

```
$ kubectl drain my-node --ignore-daemonsets
```

→ Evict all Pods also the ones not managed by a controller

```
$ kubectl drain my-node --force
```

Cordon



→ Make the Node unschedulable

```
$ kubectl cordon my-node
```

→ Make the Node schedulable again, undo the cordon / train

```
$ kubectl uncordon my-node
```

Taint

Node taints are key : value pairs associated with an effect

NoSchedule

Pods that don't tolerate the taint can't be schedule on the Node

PreferNoSchedule

Kubernetes avoid scheduded Pods that don't tolerate the taint

NoExecute

Pods are evicted from the Node if they are already running, else they can't be scheduling

> Display taint informations for all Nodes

```
$ kubectl get node
-o jsonpath='{range .items[*]}{"\n"}{.metadata.name}:{.spec.taints}'
```

> Pause a Node, don't accept new workloads on it

```
$ kubectl taint nodes my-node my-key=my-value:NoSchedule
```

> Unpause a Node

```
$ kubectl taint nodes my-node my-key:NoSchedule-
```

› Create a Pod that can be scheduled on a Node who have the taint specialkey = specialvalue : NoSchedule

```
...
spec:
  tolerations:
  - key: "specialkey"
    operator: "Equal"
    value: "specialvalue"
    effect: "NoExecute"
```

› Create a Pod that stay bound 6000 seconds before being evicted

```
...
spec:
  tolerations:
  - key: "my-key"
    operator: "Equal"
    value: "my-value"
    effect: "NoExecute"
    tolerationSeconds: 6000
```

} Delay Pod eviction

Node controller can automatically taint a Node without a manual action

When ?

- ☐ Node is not ready
- ☐ Node is unreachable
- ☐ Out of disk
- ☐ Network unavailable

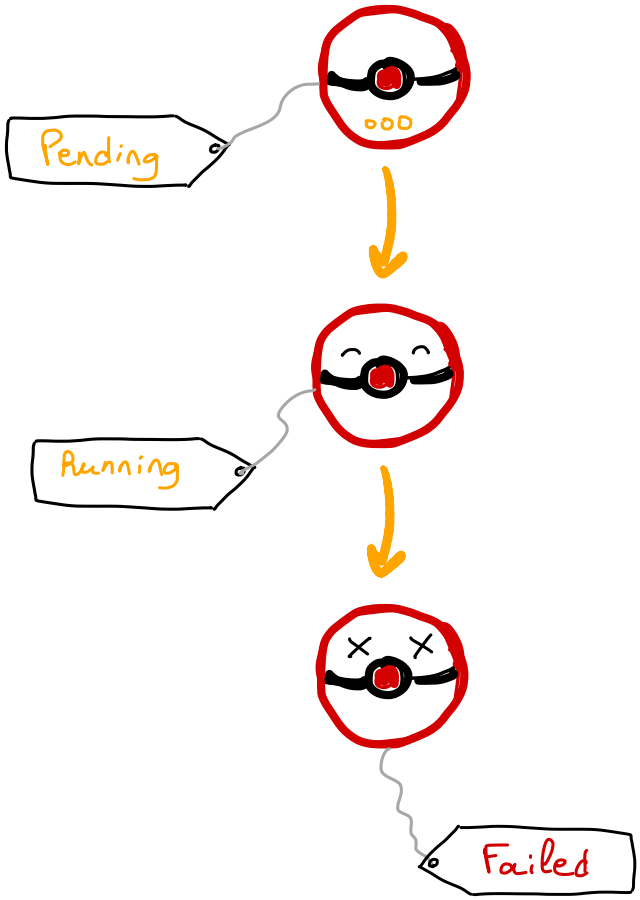
ooo

Not Ready

Debugging / Troubleshooting

Pending

When errors occur, debugging Kubernetes can be very tricky



1 Show more information about your Pods

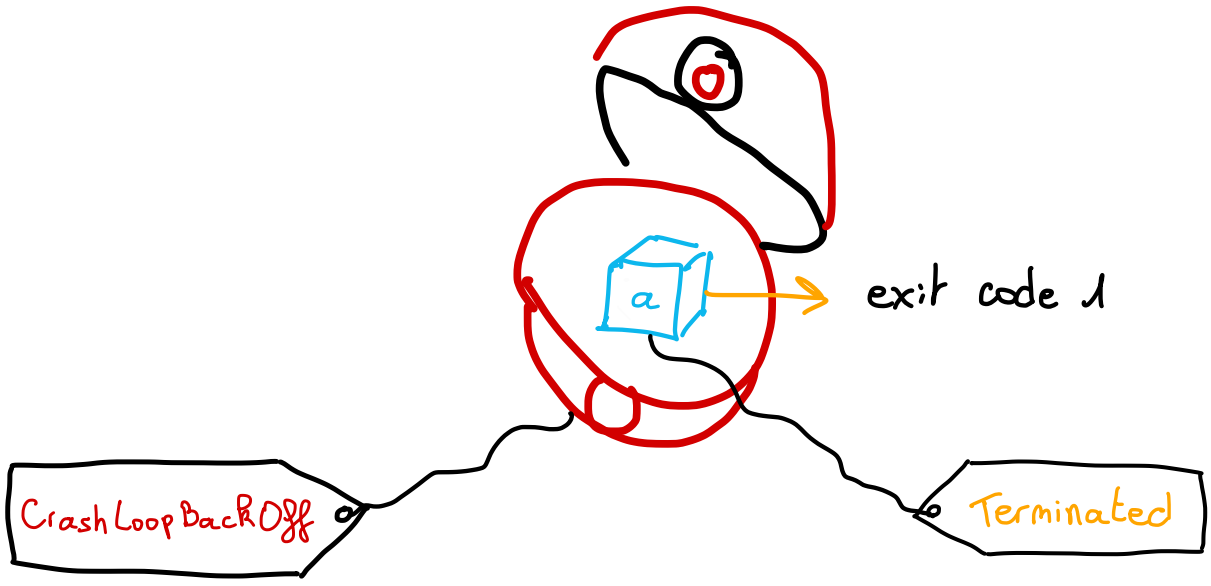
→ Display all Pods in a namespace "my-ns"

```
$ kubectl get pod -n my-ns
```

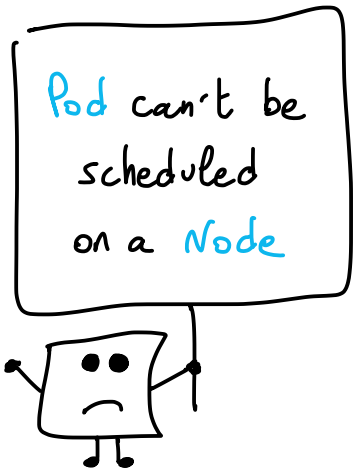
→ Display information about a Pod

```
$ kubectl describe pod my-pod -n my-ns
```

These two easy commands can be helpful in many situations, for example when a Pod is stuck in CrashLoopBackOff status:



2 My Pod struck in Pending status



Solutions :

- ☑ Resource limit are maybe > than cluster capacities
- ☑ Delete Pods / Clean unused resources in the cluster
- ☑ Add Nodes capacities
- ☑ Add more Nodes

...

3 Show cluster error messages

→ Display all events for my-ns namespace

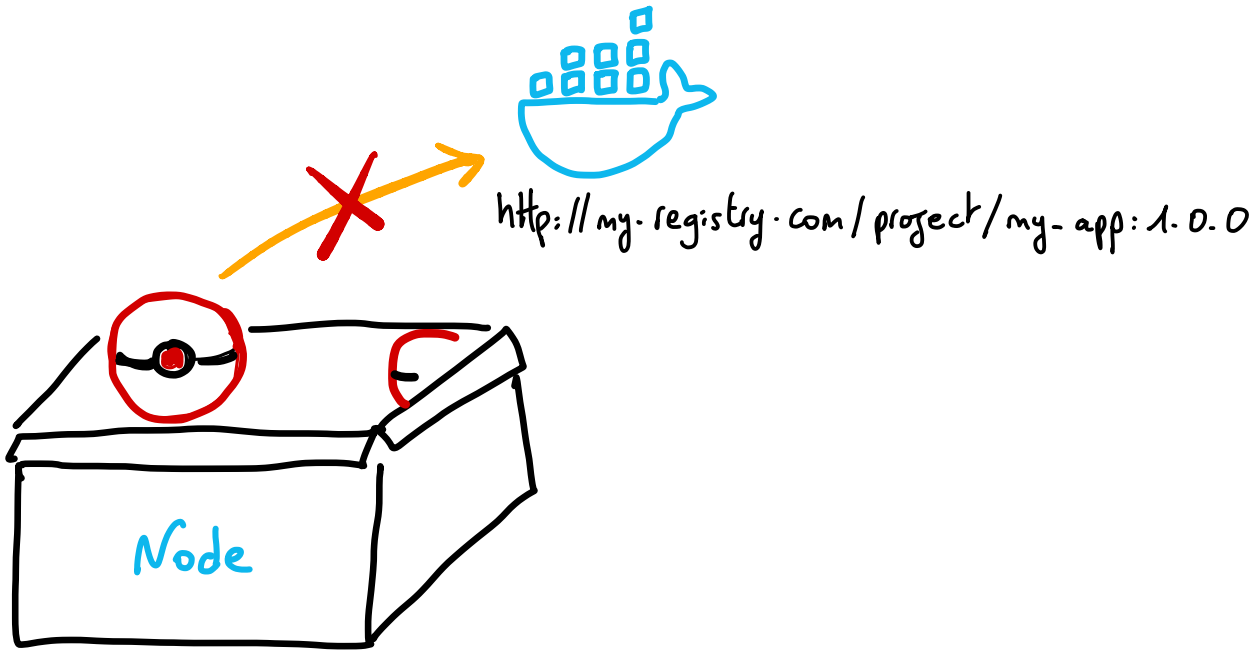
```
$ kubectl get events
--sort-by=.metadata.creationTimestamp -n my-ns
```

→ Display only Warning messages

```
$ kubectl get events --field-selector type=Warning
```

Events are namespaced.
When you execute kubectl describe command,
events are displayed at the end of
the output for a Pod.

4 My Pod stuck in Waiting / ImagePullBackOff status



Questions :

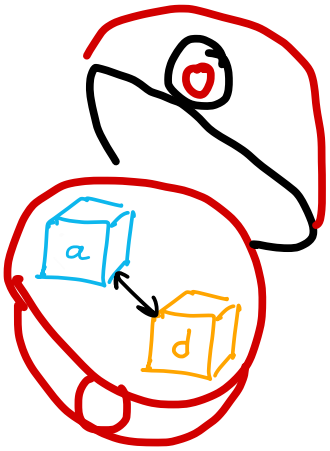
- ☑ Image name, tag & URL are good ?
- ☑ Image exists in the Docker registry ?
- ☑ Can you pull it ?
- ☑ Kubernetes have permissions to pull the image ?
- ☑ ImagePullSecret is configured for secret registry ?

5 Kubectl debug

1.18

Alpha

- Run an Ephemeral Container near the Pod we want to debug
- In this container you can execute curl, wget (...) commands inside Kubernetes environment
- Pre-requisites: active feature-gate
  EphemeralContainers = true



a application container

d debug container

```
$ kubectl alpha debug -it my-pod --image=busybox
--target=my-pod                  --container=my-debug-container
```

Share a process namespace with a container inside the Pod

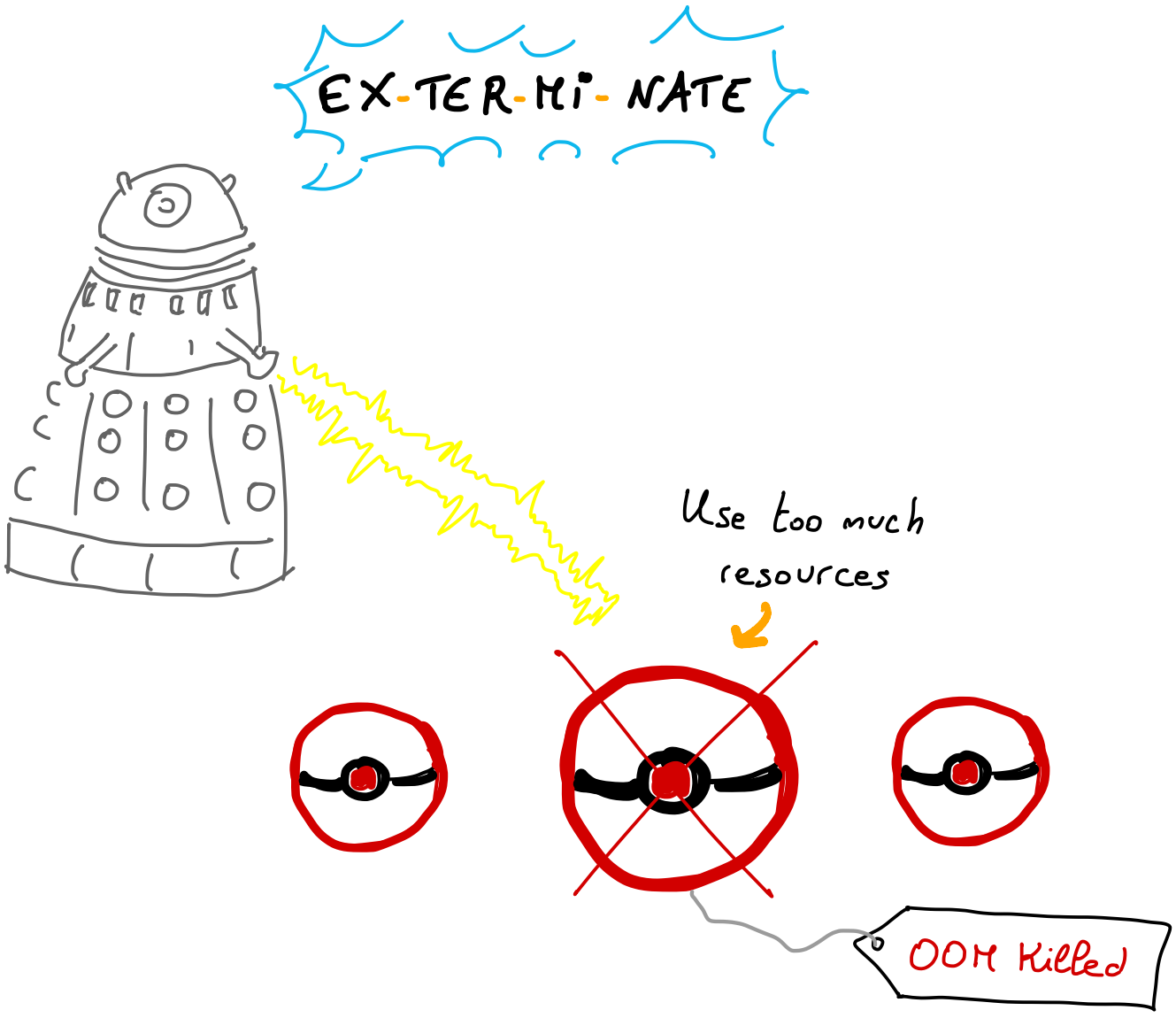
debug container name

debug container can see all processes created by my-pod

6 My Pod have been restarted multiple times

→ If Pods use too much memory

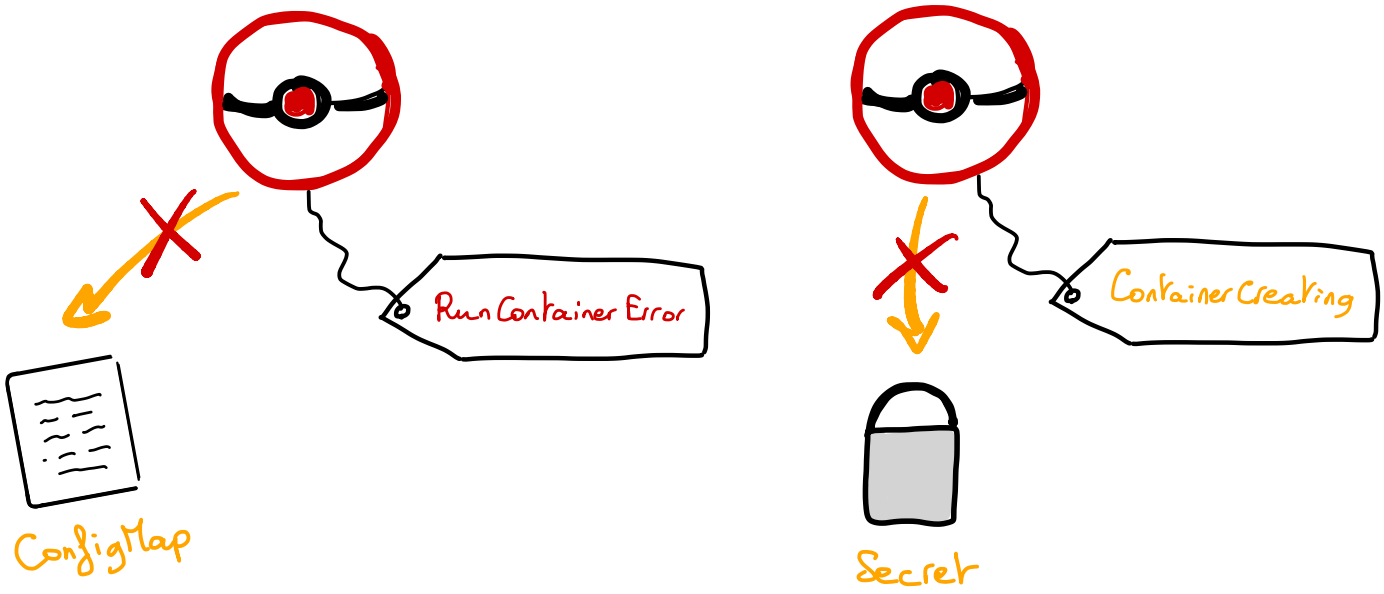
OOMKiller can destroy them



Solution :

☑ Define requests and limits memory

7 My Pod can't start ...



When you link a Pod to a ConfigMap and/or a Secret, pay attention to create the linked resources first to the name you used for linked resources

8 My Pod is Running but I don't know why it's not working?

You can simply watch Pod logs in order to try to understand:

```
$ kubectl logs my-pod
```

When a Pod is evicted from a Node or Terminated, logs are no longer available.

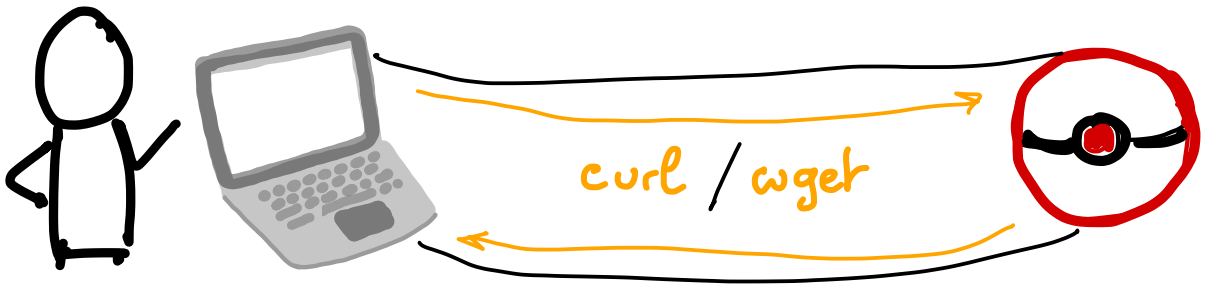
9 Container is restarting over and over...

→ LivenessProbe is used to tell container is healthy, but if it's misconfigured container will be consired as unhealthy.

Possible issues:

- ☑ Probe target is accessible?
- ☑ Application take a long time to start

10 I want to access my Pod without an external load-balancer



→ You can mount a tunnel between a Pod and your computer:

```
$ kubectl port-forward my-pod 8080
```

→ And mount a tunnel to a Service directly:

```
$ kubectl port-forward svc/my-svc 5050:5555
```

If remote port and local port are the same, specify just one

→ Then you can simply curl to localhost on the local port:

```
$ curl localhost:8080/my-api
```

Kubectl convert

Why

1.21

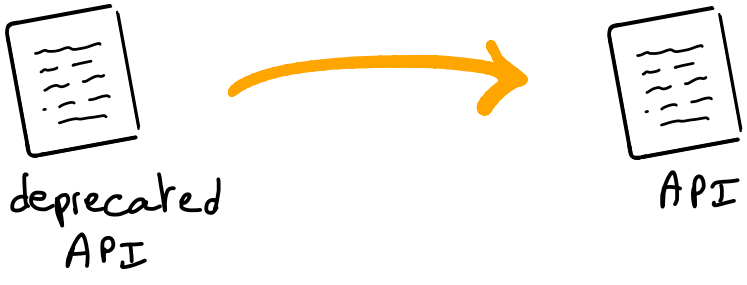
→ Kubernetes have 3 or 4 releases every year

→ Several APIs are deprecated & then removed

1.22

What

Kubectl convert allows you to update manifests to use a specific API version

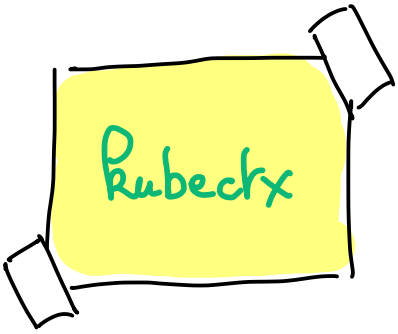


> Convert my-ingress with old API to v1

```
$ kubectl convert -f my-ingress.yaml
--output-version networking.k8s.io/v1 > my-ingress-updated.yaml
```

> Convert my-pod to latest version

```
$ kubectl convert -f my-pod.yaml
```



Tools

« Manage & switch between Kubectl contexts »

https://github.com/ahmetb/kubectx

List all the existing clusters in your KUBECONFIG

```
$ kubectx
```

Switch / connect to a cluster

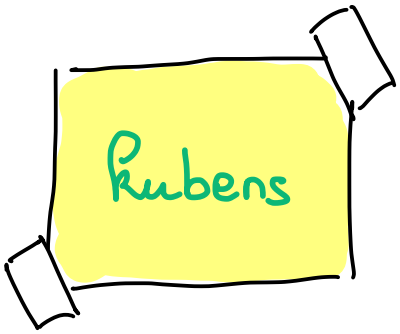
```
$ kubectx my-cluster
```

Switch to the previous cluster

```
$ kubectx -
```

Better than:

```
$ kubectl config use-context my-cluster
```



« Manage & switch between namespaces »

https://github.com/ahmetb/kubectx

List all the namespaces in the cluster

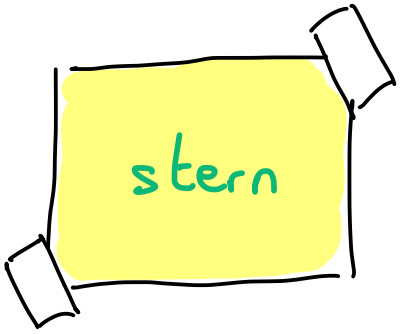
```
$ kubens
```

Switch to a namespace

```
$ kubens my-namespace
```

Better than :

```
$ kubectl config set-context --current
--namespace=my-namespace
```



"Kubectl logs under steroids ;)

https://github.com/wercker/stern

Display logs of all Pods starting by a "name"

```
$ stern my-pod-start
```

Show Pods logs since 10 seconds

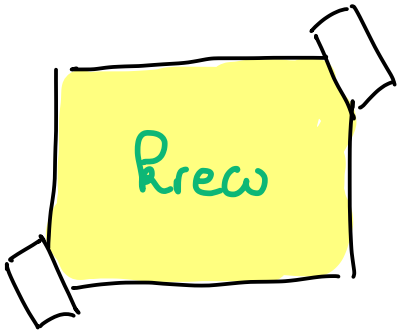
```
$ stern my-pod -c my-container -s 10s
```

Show Pods's container with timestamp

```
$ stern my-pod-start my-container -t
```

Better than :

```
$ kubectl logs -f pod-1
$ kubectl logs -f pod-2
```



“ Package manager
for kubectl plugins ”

https://github.com/kubernetes-sigs/krew

List available plugins (in the krew public index)

```
$ kubectl krew search
```

Install kubectx and kubens tools

```
$ kubectl krew install ctx
$ kubectl krew install ns
```

After installation, these tools are available as

$ kubectl ctx and $ kubectl ns

Display installed plugins

```
$ kubectl krew list
```

Add 'scraly' private index

```
$ kubectl krew index add scraly
https://github.com/scraly/krew-index
```

List all installed index

```
$ kubectl krew index list
```

List all available plugins in 'scraly' index

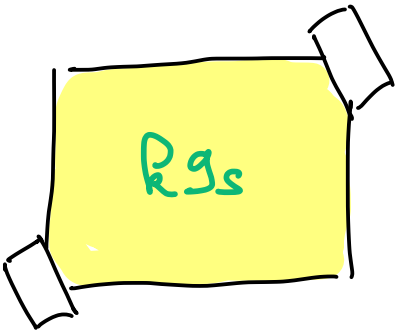
```
$ kubectl krew search scraly
```

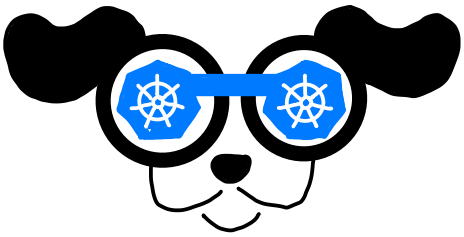
Install "season" plugin
(available in 'scraly' private index)

```
$ kubectl krew install scraly/season
```

Upgrade "season" plugin

```
$ kubectl krew upgrade season
```





https://github.com/derailed/k9s

→ UI in CLI for Kubernetes

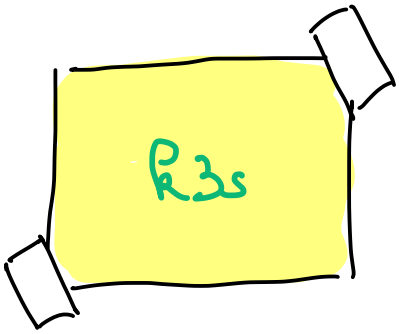
→ k9s continually watches for changes

Launch k9s

```
$ k9s
```

Run k9s in a namespace

```
$ k9s -n my-namespace
```

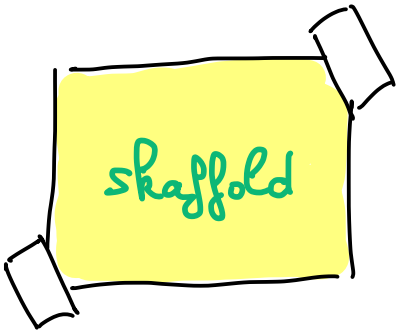


“Lightweight Kubernetes cluster”

https://k3s.io

Install & launch k3s Kubernetes cluster

```
$ curl -sfL https://get.k3s.io | sh -
```



"Build, push & deploy in one CLI"

https://skaffold.dev

Init your project (and create skaffold.yaml)

```
$ skaffold init
```

Build, tag, deploy & stream the logs everytime the code of your app changes

```
$ skaffold dev
```

Just build & tag an image

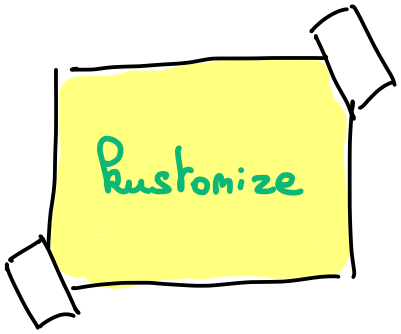
```
$ skaffold build
```

Deploy image

```
$ skaffold deploy
```

Build, tag image and output templated manifests

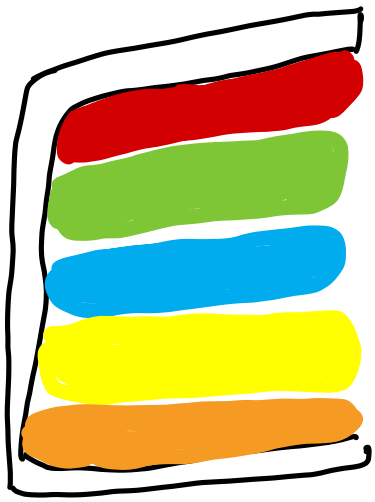
```
$ skaffold render
```



https://github.com/kubernetes-sigs/kustomize

→ Built-in Kubectl since v1.4

→ Declarative like Kubernetes (≠ Imperative like Helm)



mix-in toto

mix-in secrets

mix-in env

mix-in replica

Base

→ The aim is to add layers modifications on top of base in order to add functionalities we want

→ Works like Docker or Git : each layers represents "an intermediate system state"

→ Each YAML files are valid/usable outside of Kustomize

① Define a deployment.yaml file

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: kustomize-app
  labels:
    app: kustomize
spec:
  replicas: 1
  selector:
    matchLabels:
      app: kustomize
  template:
    metadata:
      labels:
        app: kustomize
    spec:
      containers:
      - name: app
        image: gcr.io/foo/kustomize:latest
        ports:
        - containerPort: 8080
          name: http
          protocol: TCP
```

2 Create a file called custom-env.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: kustomize-app
  labels:
    app: kustomize
spec:
  replicas: 1
  selector:
    matchLabels:
      app: kustomize
  template:
    metadata:
      labels:
        app: kustomize
    spec:
      containers:
      - name: app
        env:
        - name: MESSAGE_BODY
          value: by Kustomize ❤
        - name: MESSAGE_FROM
          value: overlay 'custom-env'
```

What we want to add to our base

3 Create a kustomization.yaml file

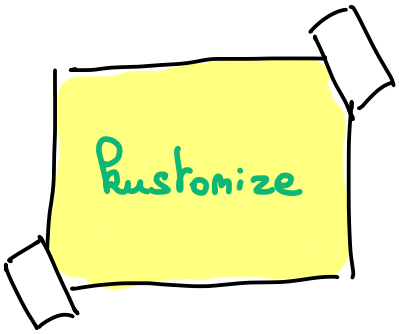
```yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization

resources:
  - ../../base

patchesStrategicMerge:
  - custom-env.yaml
```

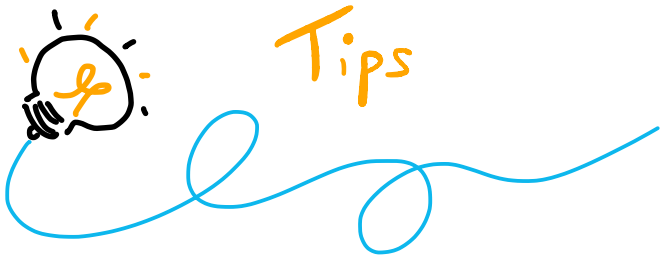
base = our deployment

patchs to apply

4 Apply

```
$ kubectl apply -k /src/main/k8s/overlay/prod
```



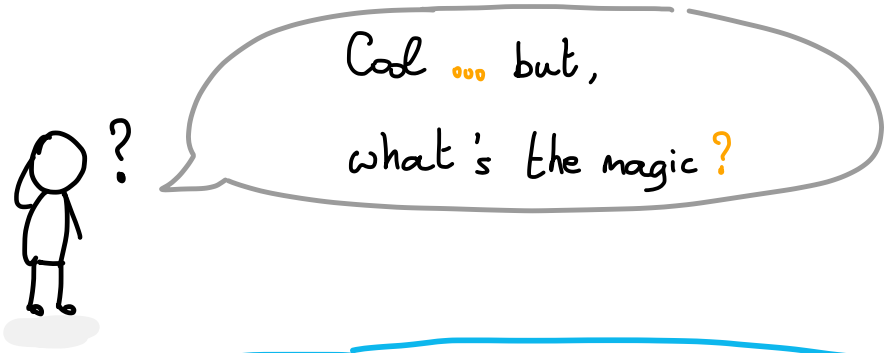


1 Set / Change image tag

```
$ kustomize edit set image
my-image=my-repo/project/my-image:tag
```

2 Create a secret

```
$ kustomize edit add secret my-secret
--type kubernetes.io/dockerconfigjson
--from-literal=password="toto"
```



In fact, this command:

- edit kustomization.yaml file
- add a SecretGenerator into it

generated kustomization.yaml :

```
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization

bases:
- ../../base

patchesStrategicMerge:
- custom-env.yaml
- replica.yaml

secretGenerator:
- literals:
  - password=toto
  name: my-secret
  type: kubernetes.io/dockerconfigjson
```

3 Add a prefix and a suffix in resource's name

```
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization

bases:
- ../../base

namePrefix: my-
nameSuffix: -v1
```

4 Create a kustomization.yaml which creates a ConfigMap

From a file:

```
configMapGenerator:
- name: my-configmap
  files:
  - application.properties
```

From literal:

```
configMapGenerator:
- name: my-configmap2
  literals:
  - key=value
```

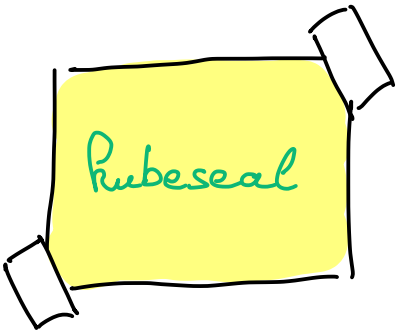
5 Merge several files in one ConfigMap

```
configMapGenerator:
- name: my-configmap
  behavior: merge
  files:
  - application.properties
  - secret.properties
```

6 Disable automatic hash suffix added in ConfigMap and Secret generated resources name

```
generatorOptions:
  disableNameSuffixHash: true
```

generatorOptions change behavior of all ConfigMap & Secret generators



« Encrypt your secret

into SealedSecret

& store them in Git »

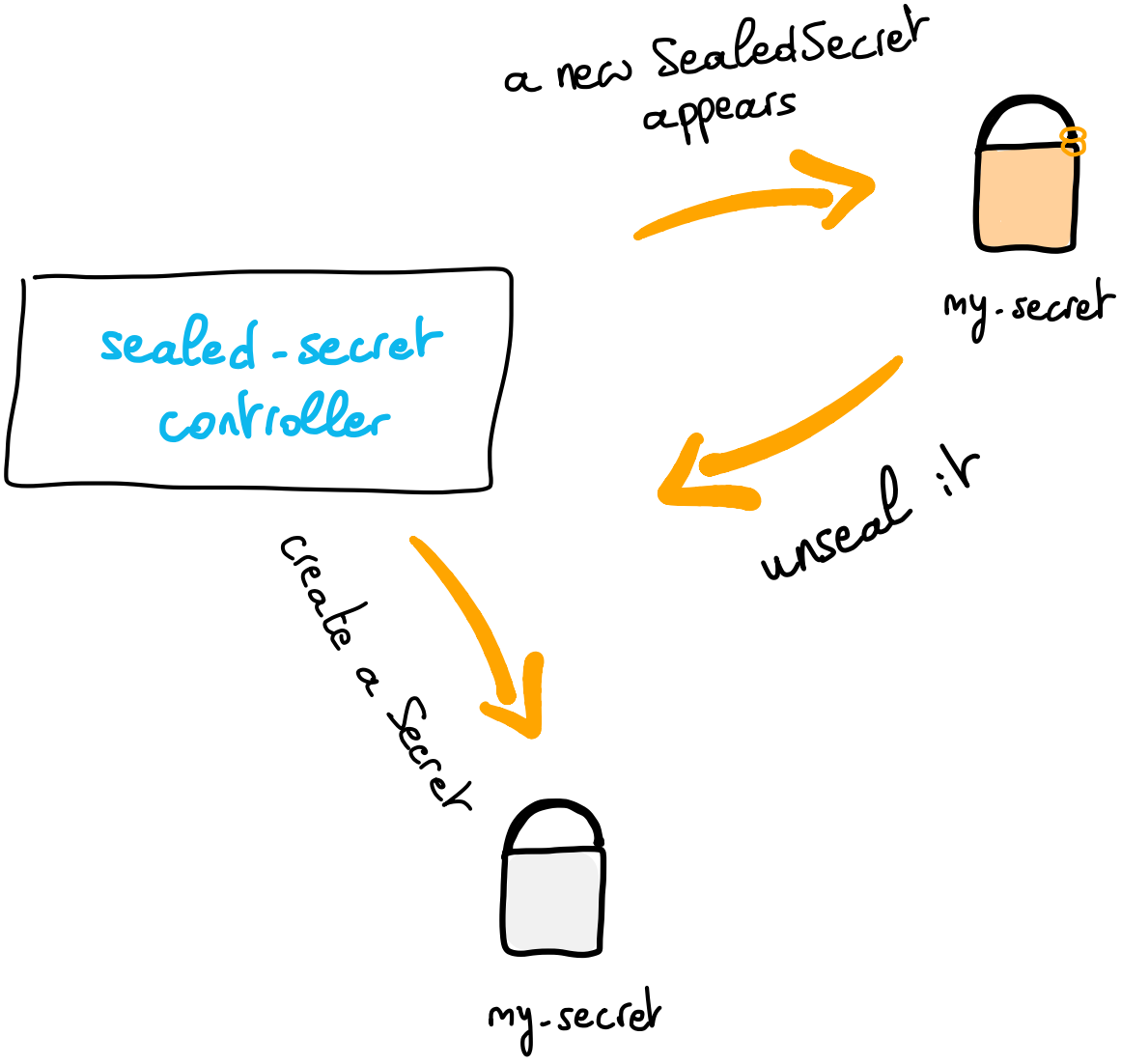
https://github.com/bitnami-labs/sealed-secrets

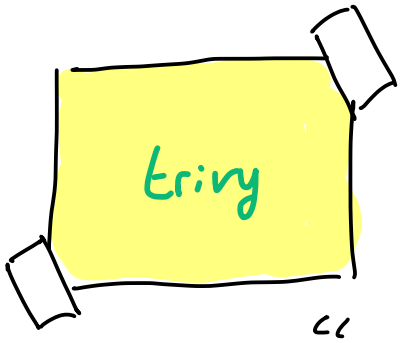
Create a Secret

```
$ kubectl create secret generic my-token
--from-literal=my_token='12345' --dry-run=client -o yaml
-n my-namespace > my-token.yaml
```

Seal the Secret

```
$ kubeseal --cert tls.crt --format=yaml < my-token.yaml >
mysealedsecret.yaml
```


a new SealedSecret appears
my-secret
sealed-secret controller
unseal it
create a Secret
my-secret



« Security scanner for  containers, clusters ... »

https://github.com/aquasecurity/trivy

- Vulnerabilities detection :
  - of OS packages
  - Application dependencies

    (npm, yarn, cargo, pipenv, Composer, bundler ...)
- Misconfiguration detection

  (Kubernetes, Docker, Terraform ...)
- Secret detection
- Simple and fast scanner
- Easy integration for CI
- A Kubernetes operator

Scan an image

```
$ trivy image python:3.4-alpine
```

Scan and save in a JSON report

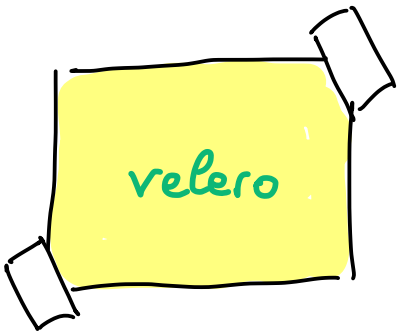
```
$ trivy image golang:1.13-alpine -f json -o results.json
```

Scan all Nodes in the default namespace of a Kubernetes cluster & display a summary report

```
$ trivy k8s -n default --report summary
```

Scan & display only URGENT vulnerabilities

```
$ trivy k8s -n default --report all
--severity MEDIUM,HIGH,CRITICAL
```



"Backup & Restore Kubernetes

resources and persistent volumes"

https://velero.io

Create a full backup

```
$ velero backup create my-backup
```

Create a backup only for resources matching with label

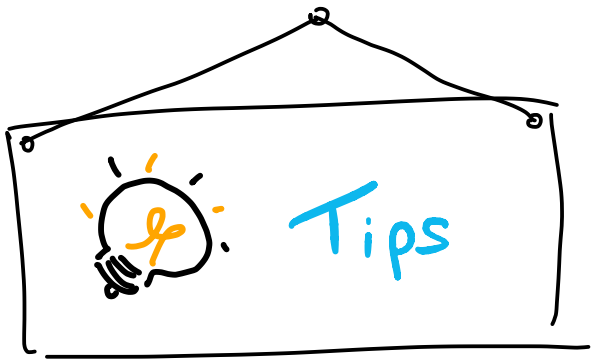
```
$ velero backup create my-light-backup
--selector app=my-app
```

Restore from a backup "Istio's Gateway & VirtualServices"

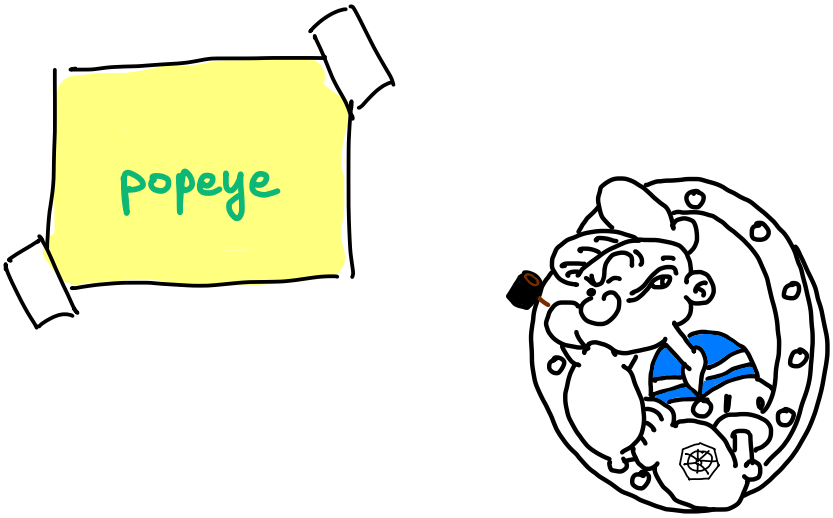
```
$ velero restore create --from-backup my-backup
--include-resources
gateways.networking.istio.io,virtualservices.networking.
istio.io
```

Schedule backup

```
$ velero schedule create daily-backup
--schedule="0 10 * * *"
```



- Create a backup before a cluster upgrade
- And it's a good practice to backup daily / weekly your resources in sensitive clusters
- Add revisions feature in the backup location bucket



https://github.com/derailed/popeye

→ Scan cluster and output a report (with a score)

→ Customizable scan & reports

→ Several ways to install it
(locally, Docker, in the cluster ...)

Scan a cluster only in one namespace

```
$ popeye --context my-cluster -n my-ns
```

Scan only a list of specified resources

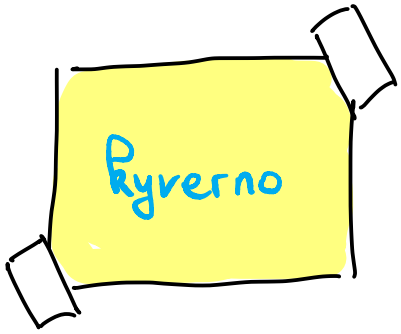
```
$ popeye --context my-cluster -n my-ns -s po,deploy,svc
```

Scan with a config file

```
$ popeye -f spinach.yaml
```

Scan & save the report (in HTML) in a S3 bucket

```
$ popeye -s3-bucket my-bucket/folder --out html
```



https://kyverno.io

→ Kubernetes native policy management

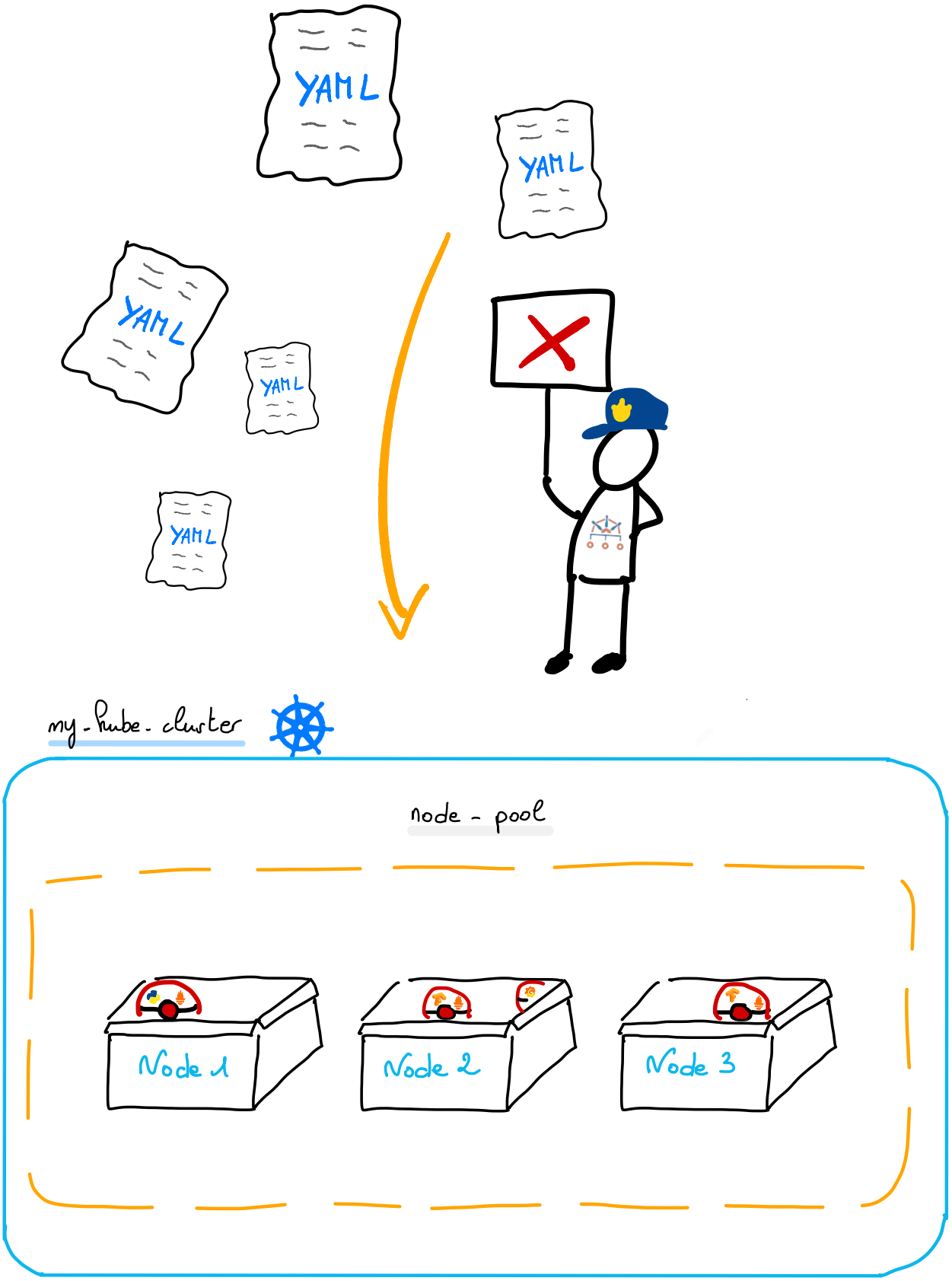
→ 3 kind of rules:

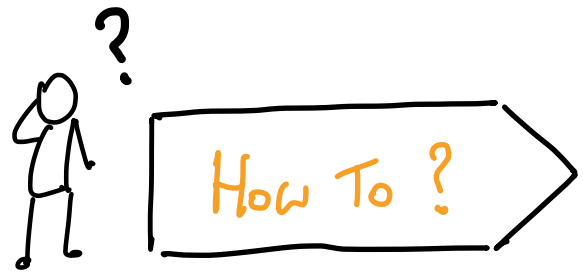
- validate
- generate
- mutate

→ Kyverno installs several webhooks:

```
NAME                                                                    WEBHOOKS
AGE
validatingwebhookconfiguration.admissionregistration.k8s.io/kyverno-policy-validating-webhook-cfg      1
52s
validatingwebhookconfiguration.admissionregistration.k8s.io/kyverno-resource-validating-webhook-cfg    2
52s
```

```
NAME                                                                    WEBHOOKS   AGE
mutatingwebhookconfiguration.admissionregistration.k8s.io/kyverno-policy-mutating-webhook-cfg      1          52s
mutatingwebhookconfiguration.admissionregistration.k8s.io/kyverno-resource-mutating-webhook-cfg    2          52s
mutatingwebhookconfiguration.admissionregistration.k8s.io/kyverno-verify-mutating-webhook-cfg      1          52s
```


YAML
YAML
YAML
YAML
YAML
my-kube-cluster
node-pool
Node 1
Node 2
Node 3



Create a policy that disallow deployments for Pods in the default namespace

```
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: disallow-default-namespace
spec:
  validationFailureAction: enforce
  rules:
  - name: validate-namespace
    match:
      resources:
        kinds:
        - Pod
    validate:
      message: "Using \"default\" namespace is not
allowed."
      pattern:
        metadata:
          namespace: "!default"
  - name: require-namespace
    match:
      resources:
        kinds:
        - Pod
    validate:
      message: "A namespace is required."
      pattern:
        metadata:
          namespace: "?*"
```

Set validationFailureAction to enforce to block resource creation or updates, audit to report violations.

> Create a policy that creates a ConfigMap in all namespaces excepted kube-system, kube-public & kyverno

```
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: zk-kafka-address
spec:
  rules:
  - name: zk-kafka-address
    match:
      resources:
        kinds:
        - Namespace
    exclude:
      resources:
        namespaces:
        - kube-system
        - kube-public
        - kyverno
    generate:
      synchronize: true
      kind: ConfigMap
      name: zk-kafka-address
      # generate the resource in the new namespace
      namespace: "{{request.object.metadata.name}}"
      data:
        kind: ConfigMap
        metadata:
          labels:
            somekey: somevalue
        data:
          ZK_ADDRESS: "192.168.10.10:2181,192.168.10.11:2181"
          KAFKA_ADDRESS:
"192.168.10.13:9092,192.168.10.14:9092"
```

Set synchronize to true to keep resources synchronized accross changes.

› Create a policy that adds label my.awesome.app to Pods, Services, ConfigMaps & Secrets in a given namespace

```
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: add-label
spec:
  rules:
  - name: add-label
    match:
      resources:
        kinds:
        - Pod
        - Service
        - ConfigMap
        - Secret
        namespaces:
        - team-a
    mutate:
      patchStrategicMerge:
        metadata:
          labels:
            app: my-awesome-app
```

› Deploy policy in the cluster

```
$ kubectl apply -f policy-add-label.yaml
```

Display existing policies for all namespaces

```
$ kubectl get cpol -A
NAME                           BACKGROUND   ACTION    READY
add-label                      true         audit
disallow-default-namespace     true         enforce   true
```

Display reports for all namespaces

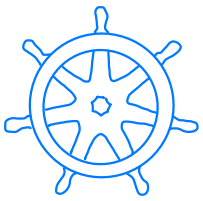
```
$ kubectl get policyreport -A
```

View all policy violations in the cluster

```
$ kubectl describe polr -A | grep -i "Result: \+fail" -B10
```

Check if policies are validates

```
$ kyverno validate *.yaml
```



Kubernetes 1.18

fit & finish

GENERAL

- First release for 2020
- own logo

INGRESS CLASS

annotation `ingress.class`

new resource `IngressClass`

↳ Define `IngressController` name & configuration parameters

INGRESS RULES

wildcard in `host` is now allowed ❗

```
host: *.scraly.com
```

KUBECTL DEBUG

Alpha

Aim to run Ephemeral Container near the Pod we want to debug

Pre-requisites: active feature-gate EphemeralContainers = true

How to:

```
$ kubectl alpha debug -it my-pod --image=busybox
--target=my-pod          --container=my-debug-container
```

Share a process namespace with a container inside the Pod

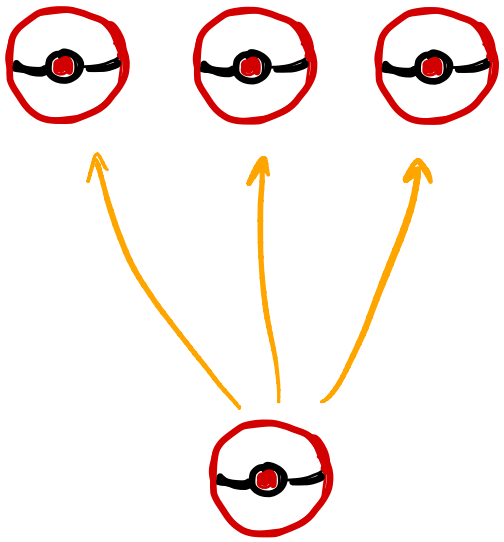
debug container name

debug container can see all processes created by my-pod

HORIZONTAL POD AUTOSCALER

Possibility to define scale up & down by HPA with behavior field

```
apiVersion: autoscaling/v2beta2
kind: HorizontalPodAutoscaler
metadata:
  name: my-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: my-deploy
  minReplicas: 3
  maxReplicas: 10
  targetCPUUtilizationPercentage: 80
  behavior:
    scaleUp:
      policies:
      - type: Percent
        value: 90
        periodSeconds: 15
    scaleDown:
      policies:
      # scale down 1 Pod every 10 min
      - type: Pods
        value: 1
        periodSeconds: 600
```



IMMUTABLE SECRET & CONFIGMAP

Pre.requisites : active feature.gate

ImmutableEphemeralVolumes = true

Allows to not edit sensitive data by mistake

How ?

immutable : true field

❤ — KUBECTL RUN

kubectl run command now create only a Pod

Aim : one command → one usage 

Kubernetes 1.19

maturity

GENERAL

- longest delivery cycle due to COVID
- 34 enhancements

IMMUTABLE SECRET & CONFIGMAP

Beta

Allows to not edit sensitive data by mistake.

Why?

Protect against accidental updates that can cause downtime & optimize performance because Control Plane don't have to check updates.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: my-cm
immutable: true
data:
  my-key: my-value
```

TLS 1.3

Now Kubernetes support TLS 1.3.

WARNING MECHANISM

Kubernetes returns warning messages when you use a deprecated API.

```
$ kubectl get ingress -n my-namespace
Warning: extensions/v1beta1 Ingress is deprecated in
v1.14+, unavailable in v1.22+; use
networking.k8s.io/v1 Ingress
No resource found in my-namespace namespace
```

SECCOMP

GA

SecComp (Secure Computing Mode) is a security feature of the Linux kernel for limiting system calls that app make.

- Provide sandboxing
- Reduce the actions that a process can perform

Keep your Pods secure

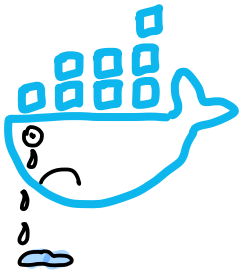
```
apiVersion: v1
kind: Pod
metadata:
  name: my-audit-pod
  labels:
    app: my-audit-pod
spec:
  securityContext:
    seccompProfile:
      type: Localhost
      localhostProfile: profiles/audit.json
  containers:
  - name: test-container
    image: hashicorp/http-echo:0.2.3
    args:
    - "-text=just made some syscalls!"
    securityContext:
      allowPrivilegeEscalation: false
```

Kubernetes 1.20

GENERAL

- normal release cycle cadence again
- 42 enhancements

DOCKER DEPRECIATION



Docker support in Kubelet is now deprecated & will be removed in a future release.

But ... don't panic your Docker-produced images will continue to work in your clusters.

EXEC PROBE TIMEOUT HANDLING

Fixed bug in Liveness / Readiness's exec probe timeouts.

Now the field timeoutSeconds is respected.

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  containers:
  - name: my-container
    image: my-image:1.0
    livenessProbe:
      exec:
        command:
        - cat
        - /tmp/healthy
    initialDelaySeconds: 5
    periodSeconds: 5
    timeoutSeconds: 5
```

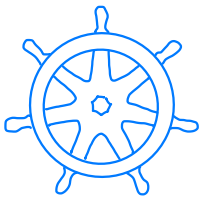
STARTUP PROBE

Hold off all the other probes until the Pod finishes its startup

A solution for slow-starting Pods.

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
  labels:
    app: my-app
spec:
  containers:
  - name: my-container
    image: my-image:1.0
    ports:
- name: liveness-port
  containerPort: 8080
  livenessProbe:
    httpGet:
      path: /healthz
      port: liveness-port
    failureThreshold: 1
    periodSeconds: 10
  startupProbe:
    httpGet:
      path: /healthz
      port: liveness-port
    failureThreshold: 30
    periodSeconds: 10
```

the app have 5 min (30 x 10s) to finish its startup



Kubernetes 1.21

GENERAL

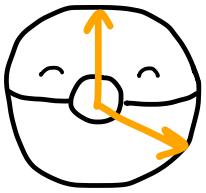
- the way of communication has changed
- 51 enhancements

POD SECURITY POLICY DEPRECATION

PSP is deprecated & will continue to be available and fully. functional until 1.25.

https://kubernetes.io/blog/2021/04/06/podsecuritypolicy-deprecation-past-present-and-future/

CRONJOBS ARE NOW STABLE



Introduced in 1.4, CronJobs are finally Generally Available (GA).

Welcome to the new CronJob controller v2 that has increased performance improvements.

KUBECTL

```
$ kubectl get <resource-type> <your-resource>
-o yaml
```

will no longer display managed fields which makes reading and parsing the output much easier!

DEFAULT CONTAINER

You can now specify a default container.

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
  annotations:
    kubectl.kubernetes.io/default-container:
my-container-2
spec:
  containers:
  - name: my-container
    image: my-image
  - name: my-container-2
    image: my-image-2
    command: ["/bin/sh", "-c"]
    args:
      - while true; do
          date >> /html/index.html;
          sleep 1;
        done
```

And no longer have to specify it with `-c`:

```
$ kubectl exec my-pod -it -- sh
```

Kubernetes 1.22

GENERAL

- The release cadence has changed.
- 53 enhancements
- ⚠ 3 features deprecated
- Several APIs have been removed

MIGRATION PLUGIN

kubectl convert helps you to migrate older deprecated/removed definition resources:

```
$ kubectl convert -f role.yaml --output-version
rbac.authorization.k8s.io/v1 > role-updated.yaml
```

WARNING MECHANISM

Stable

Kubernetes returns warning messages when you use a deprecated API.

```
$ kubectl get ingress -n my-namespace
Warning: extensions/v1beta1 Ingress is deprecated in
v1.14+, unavailable in v1.22+; use
networking.k8s.io/v1 Ingress
No resource found in my-namespace namespace
```

IMMUTABLE LABEL SELECTOR

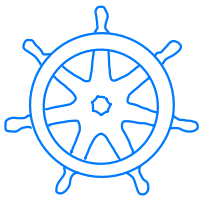
Stable

By default, namespaces are not guaranteed to have any identified labels.

Welcome to new immutable label kubernetes.io/metadata.name that has been added to all namespaces.

↳ namespace name

This label can be used with any ns selector (NetworkPolicy ...).



Kubernetes 1.23

GENERAL

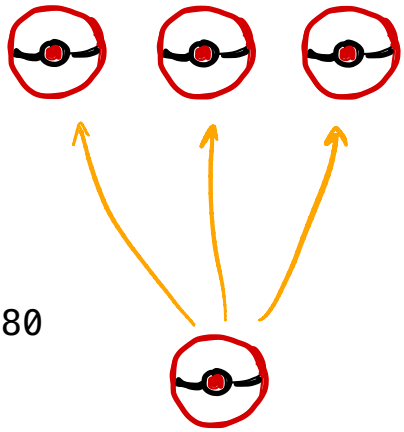
- Last release of 2021
- 47 enhancements

HORIZONTAL POD AUTOSCALER

autoscaling/v2 stable API moves to GA

and autoscaling/v2beta2 has been deprecated.

```yaml
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: my-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: my-deploy
  minReplicas: 3
  maxReplicas: 10
  targetCPUUtilizationPercentage: 80
  behavior:
    scaleUp:
      policies:
      - type: Percent
        value: 90
        periodSeconds: 15
```



KUBECTL DEBUG

Beta

Aim to run Ephemeral Container near the Pod we want to debug

```
$ kubectl alpha debug -it my-pod --image=busybox
--target=my-pod          --container=my-debug-container
```

Share a process namespace with a container inside the Pod

debug container name

debug container can see all processes created by my-pod

TTL for JOBS

Welcome to the new field ttlSecondsAfterFinished in Jobs spec that will delete the Job once it has been finished for a certain amount of time.

 No need to schedule cleanup of Jobs anymore!

```
apiVersion: batch/v1
kind: Job
metadata:
  name: my-job-with-ttl
spec:
  ttlSecondsAfterFinished: 100
  template:
    metadata:
      name: my-job-with-ttl
    spec:
      containers:
      - name: busybox
        image: busybox
```

POD SECURITY

Beta

PSP are deprecated since 1.21 & will be removed in 1.25.

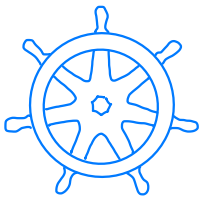
Now the new Pod Security Admission is enabled by default, and built-in, in order to prevent Pods from ever having dangerous capabilities.

IPV4/V6 DUAL STACK

GA

All clusters now support IPv4 and IPv6 at the same time.

To use this feature, Nodes must have IPv4 & IPv6 network interfaces. So you must use a dual-stack capable CNI plugin like Calico & configure Services to use both with ipFamilyPolicy field.

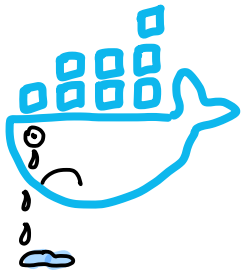


Kubernetes 1.24

GENERAL

- First release of 2022
- 46 enhancements

DOCKERSHIM REMOVED



Docker support in Kubelet is now removed

But ... don't panic your Docker-produced images will continue to work in your clusters.

GRPC PROBE

Beta

Aim to configure Startup | Liveness | Readiness probes for gRPC apps (without exposing HTTP endpoint).

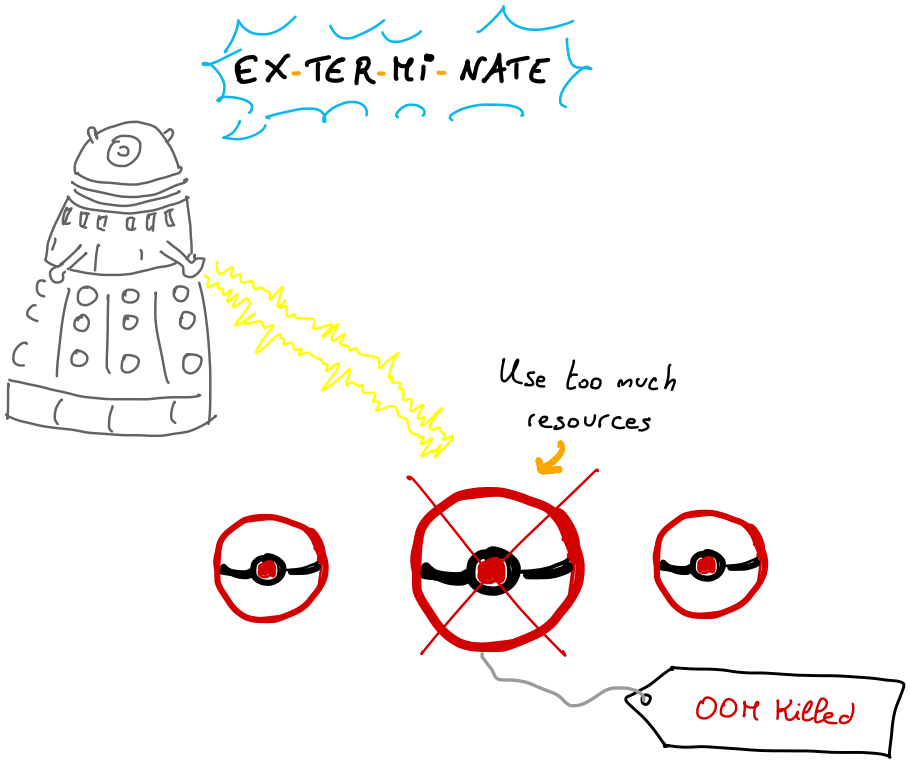
```
apiVersion: v1
kind: Pod
metadata:
  name: etcd-with-grpc
spec:
  containers:
  - name: etcd
    image: k8s.gcr.io/etcd:3.5.1-0
    command: [ "/usr/local/bin/etcd", "--data-dir",
"/var/lib/etcd", "--listen-client-urls", "http://
0.0.0.0:2379", "--advertise-client-urls", "http://
127.0.0.1:2379", "--log-level", "debug"]
    ports:
    - containerPort: 2379
    livenessProbe:
      grpc:
        port: 2379
      initialDelaySeconds: 10
```

CLASS OF LOAD BALANCER

New field LoadBalancerClass for Service that allows to specify the kind of load Balancer you want.

NEW METRIC : OOM EVENTS

Welcome to a new metric in Kubelet : container_oom_events_total that allow you to count OOM (Out of Memory) events that happen in each container.



FUTURE BETA API

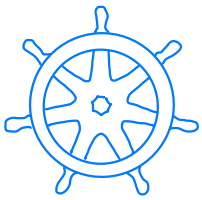
Alpha APIs are disabled by default and can be enabled or disabled with feature gates.

Now Beta APIs are disabled by default too !

CREATE SA TOKEN

You can now create additional API tokens for a ServiceAccount (SA).

```
apiVersion: v1
kind: Secret
metadata:
  name: my-sa-secret
  annotations:
    kubernetes.io/service-account.name: mysa
type: kubernetes.io/service-account-token
---

apiVersion: v1
kind: ServiceAccount
metadata:
  name: read-only
  namespace: default
secrets: my-sa-secret
```



Kubernetes 1.25

GENERAL

- Stability and novelties
- 40 enhancements

POD SECURITY POLICY REMOVED

PSP have been removed because of its complexity but don't panic, a replacement exists : Pod Security Admission.

Migration guide :

https://kubernetes.io/docs/tasks/configure-pod-container/migrate-from-psp/

USER NAMESPACES

Alpha

The fact is that there are a lot of vulnerabilities due to excessive privileges given to a Pod.

Welcome to a long awaited feature: user namespaces support in Kubernetes!

Pre-requisites: active feature-gate UserNamespacesSupport = true

How to:

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  hostUsers: false
  containers:
  - name: my-container
    image: my-image
```

For the moment, this feature is not recommanded for production.

POD SECURITY

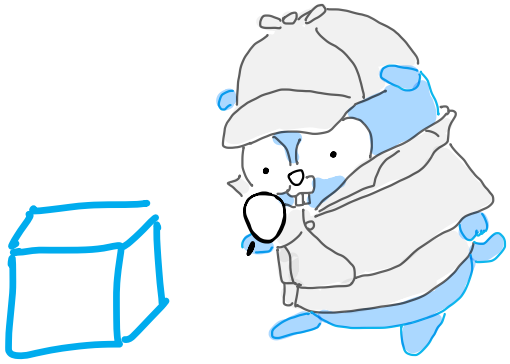
Stable

The new Pod Security Admission is now stable, and built-in, in order to prevent Pods from ever having dangerous capabilities.

CONTAINER CHECKPOINTING

Alpha

This feature allows to taking a snapshot of a running container, that can be transferred to another Node for analysis needs.



Pre-requisites: active feature-gate

ContainerCheckpointRestore = true

RETRIABLE JOBS

Alpha

Thanks to this feature you can determine if a Job should be retried or not in case of failure.

Pre-requisites: active feature-gate JobBackoffPolicy = true

KUBECTL DEBUG

Stable

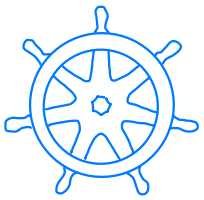
Aim to run Ephemeral Container near the Pod we want to debug

```
$ kubectl debug -it my-pod --image=busybox
--target=my-pod          --container=my-debug-container
```

Share a process namespace with a container inside the Pod

debug container name

debug container can see all processes created by my-pod



Kubernetes 1.26

GENERAL

- The power of the community
- 37 enhancements

POD DISRUPTION BUDGET & HEALTHY PODS

Alpha

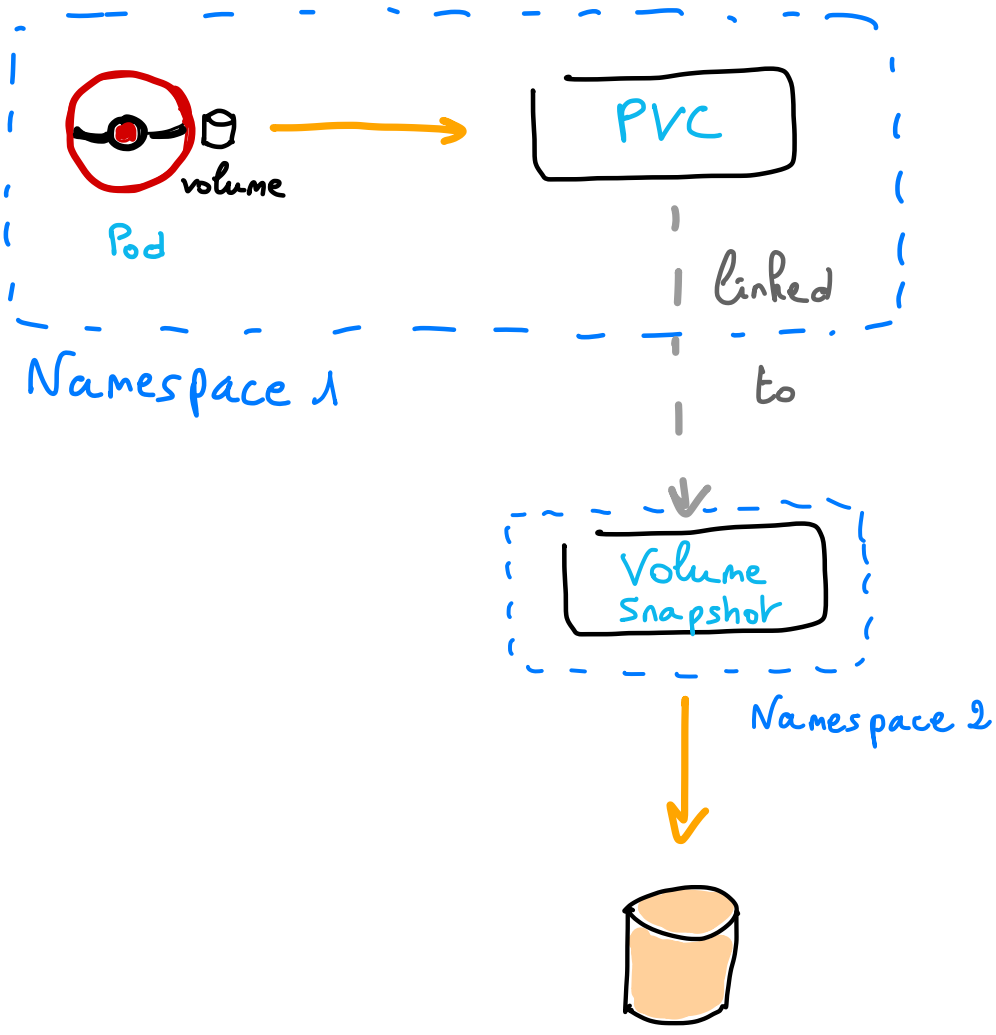
PDB only take in account Running Pods but a Pod can be Running and not Ready.
This new feature allow to prevent an eviction.

Pre-requisites: active feature-gate
PDBUnhealthyPodEvictionPolicy = true

PROVISION FROM VOLUME SNAPSHOT ACROSS NAMESPACES

Alpha

Aim to create a PersistentVolumeClaim from a VolumeSnapshot accross namespaces.



LOAD BALANCER SERVICE

GA

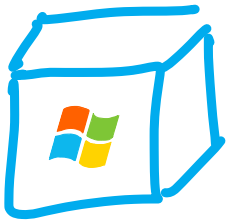
Aim to create a Service of type equals to Load Balancer that has different port definitions with different protocols.

```
apiVersion: v1
kind: Service
metadata:
  name: ingress-nginx
  namespace: ingress-nginx
  labels:
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/part-of: ingress-nginx
spec:
  type: LoadBalancer
  ports:
    - name: tcp
      port: 5555
      targetPort: 5555
      protocol: TCP
    - name: udp
      port: 5556
      targetPort: 5556
      protocol: UDP
  args:
  selector:
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/part-of: ingress-nginx
```

WINDOWS PRIVILEGED CONTAINERS

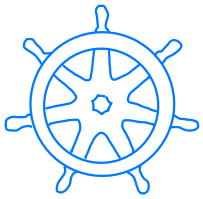
GA

Host process containers, the Windows equivalent to Linux privileged containers, are now supported.



CRI v1ALPHA2 REMOVAL

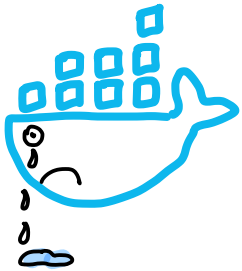
⚠ containerd, as container runtime for Kubelet, < 1.6 is no longer supported.



Docker deprecation

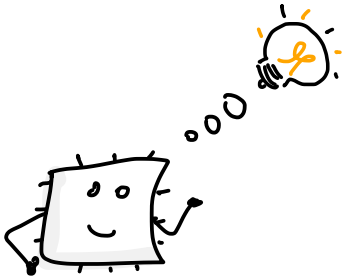
since 1.20

News



→ Docker support in Kubelet is now deprecated
& will be removed in a future release

First of all



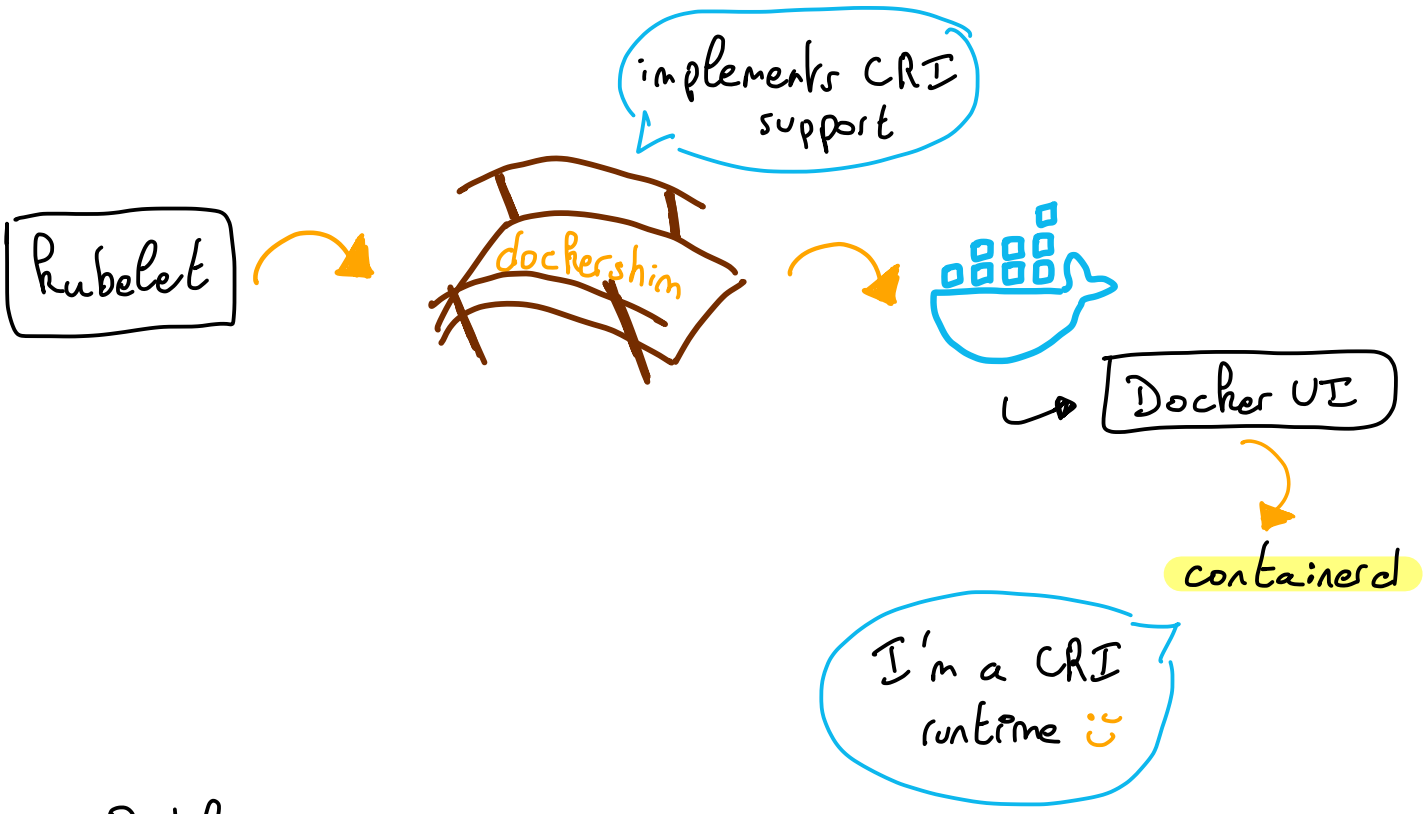
Build ≠ Runtime

→ Images still can be | built created | locally with Docker
& through CI/CD & pulled by Kubernetes

Docker produces OCI (Open Container Initiative) images.

→ Kubernetes will use containerd as CRI runtime

Previous architecture



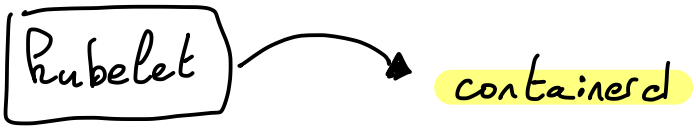
→ Problems :

› Docker includes so many components (networking, volumes, UX enhancements ...)

→ ⚠ security risks

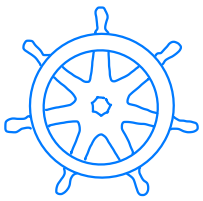
- Kubelet only understands CRI runtime
- Need to maintain dockershim bridge

New archi



→ containerd is an option for this container runtime but also CRI-O.

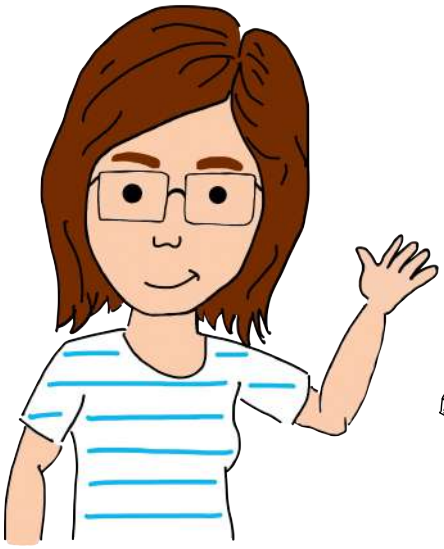
A container runtime is responsible for pulling & running container images.



Glossary

CJ	CronJob
CM	ConfigMap
CNI	Container Network Interface
CRI	Container Runtime Interface
HPA	Horizontal Pod Autoscaler
NP	Network Policy
NS	Namespace
OCI	Open Container Initiative
PDB	Pod Disruption Budget
PSP	Pod Security Policy
PV	Persistent Volume
PVC	Persistent Volume Claim
QoS	Quality of Service
RBAC	Role-Based Access Control
SA	Service Account

Who am I ?



Aurélie Vache

DevRel ❤ DevOps - + 16 years xp

Google Developer Expert in Cloud technologies

CNCF Ambassador - Docker Captain

Duchess France / Women in tech association

Technical writer - Speaker - Sketchnoter

Contact me !

@aurelievache

Abstract

Understanding Kubernetes can be difficult or time-consuming. I created this collection of sketchnotes about Kubernetes in order to try to explain the technology in a visual way.

Included :

- ☑ Kubernetes components
- ☑ Resources
- ☑ Concretes examples
- ☑ Tips & Tools