Cloud Computing, Part 2

Distributed and Pervasive Systems, MSc

Christian Marius Lillelund
cl@ece.au.dk

Department of Electrical and Computer Engineering
Aarhus University

Revised on February 5, 2021

Outline

Outline

From last time

What did we learn?

- Cloud computing is delivery of online computing services
- Microservices are SoA-styled small, independent services that do one thing
- Containers are lightweight isolated software environments
- Docker is a container tool that can run and create containers
- Kubernetes is an open-source container orchestration engine

Outline

Pods in Kubernetes

What is a pod?

- A pod is a group of one or more tightly related containers that run together and share namespace
- Each pod is like a separate logical machine.
- All containers in a pod will appear to be running on the same logical machine.
- Can only run on one node

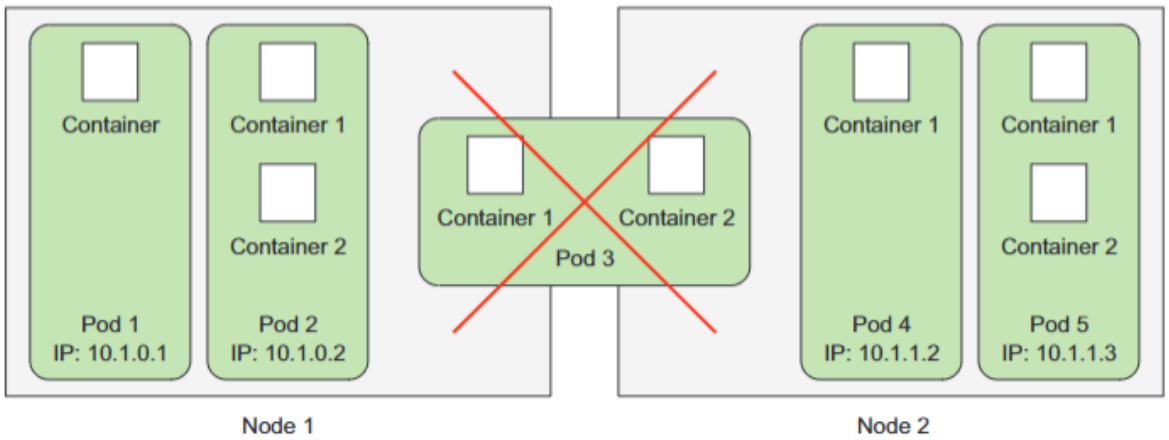


Figure: Fig. by courtesy of Marko Luksa[1]

Why pods?

- Containers run only a single process.
- Pods allow us to bind containers together as a single unit.
- Pods run closely related processes together in the same environment.
- Processes think they are running together. Closed world.

Network with pods

- All pods reside in a single flat, shared, network address space.
- Containers share the same IP
- Avoid port conflicts

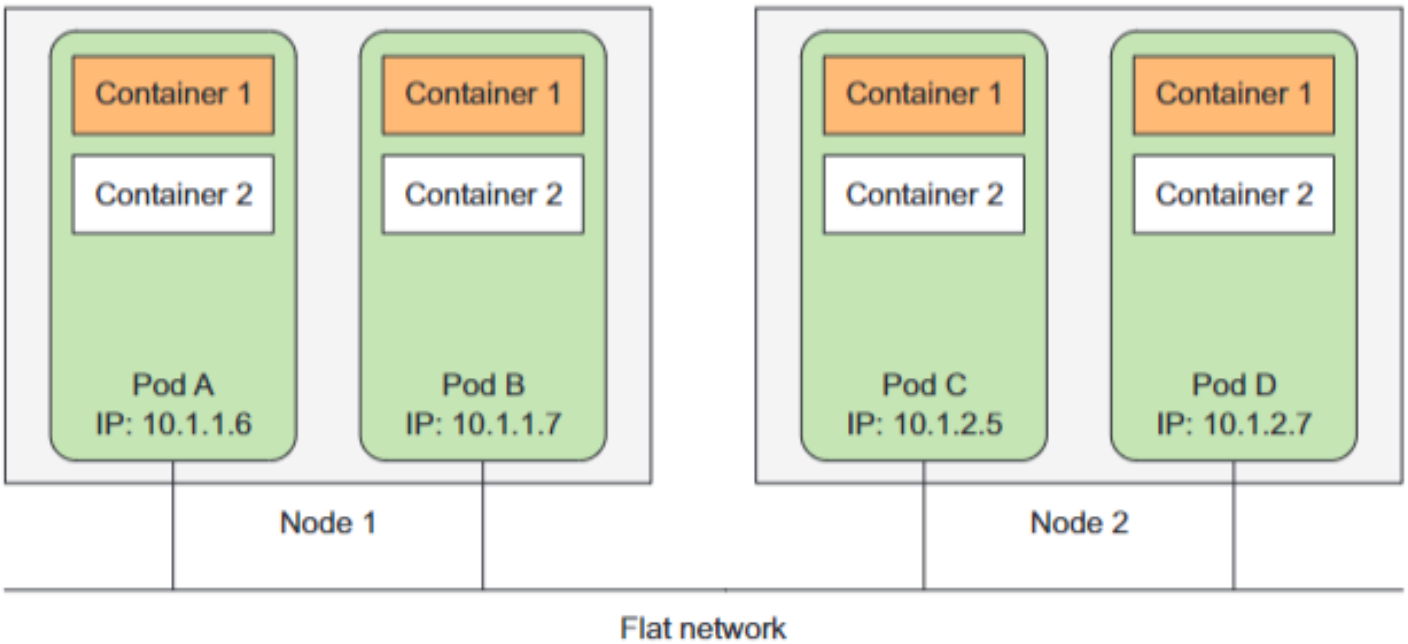


Figure: Fig. by courtesy of Marko Luksa[1]

The inside of a pod

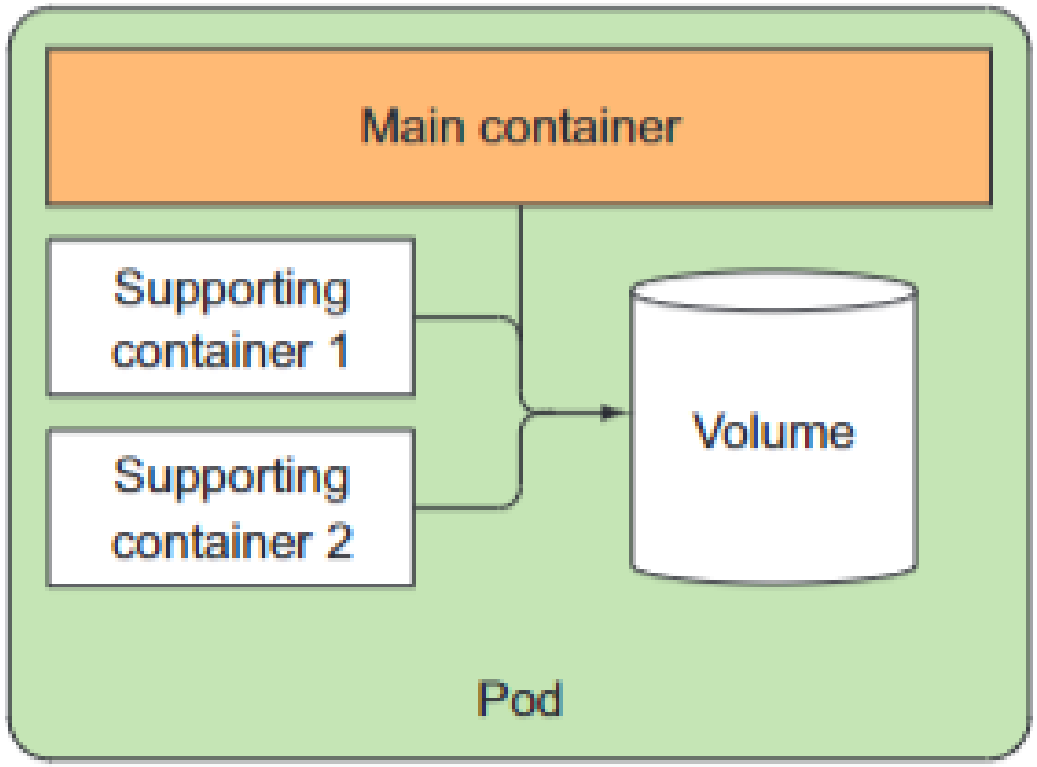


Figure: Fig. by courtesy of Marko Luksa[1]

Using multiple containers

When to use multiple containers?

- Do they need to be run together?
- Do they scale together?
- Are they single components or one whole?

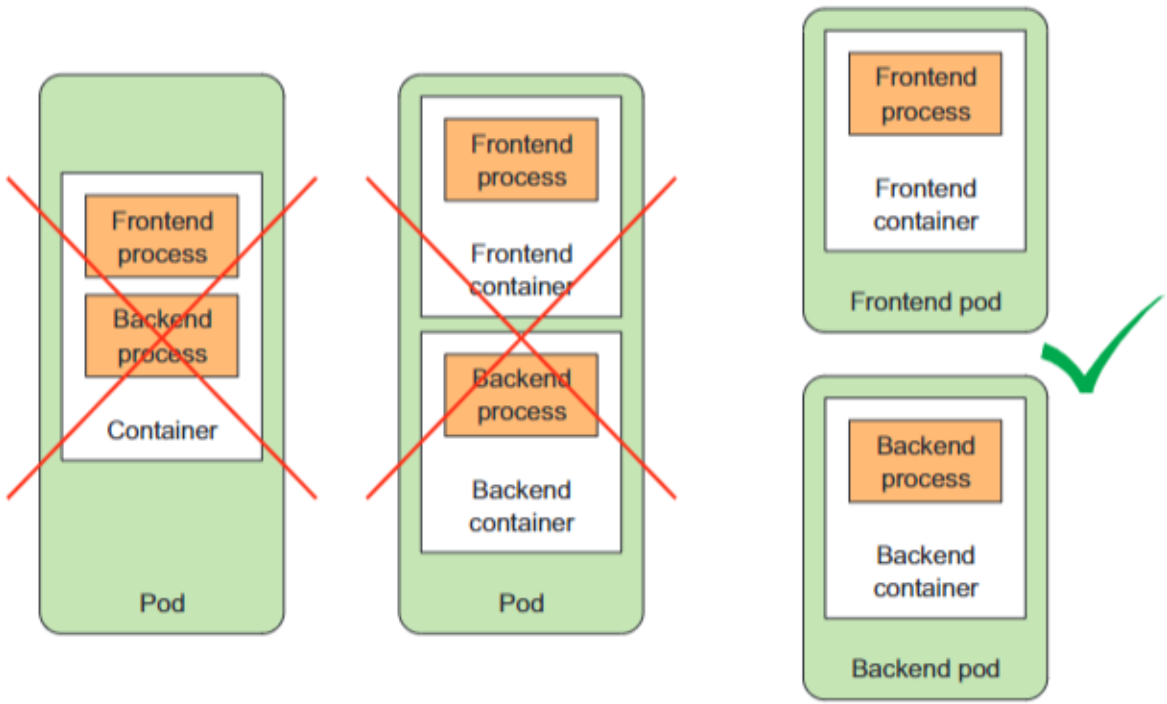


Figure: Fig. by courtesy of Marko Luksa[1]

Creating pods

- Created by posting a YAML or JSON to the Kubernetes API
- Instead of "kubectl run", you post a YAML file
- Enables more options and version control

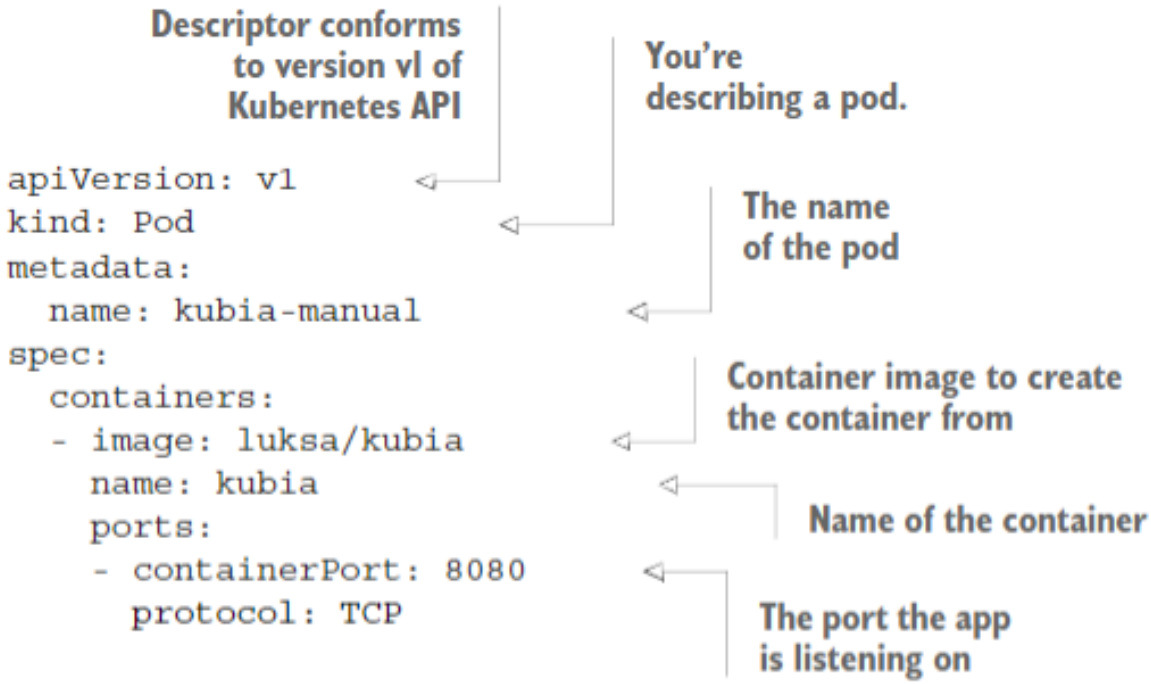


Figure: Listing by courtesy of Marko Luksa[1]

Creating pods commands

Useful commands for creating pods and getting the manifest

```
$ kubectl create -f kubia-manual.yamlpod
$ kubectl get po kubia-manual -o yaml
$ kubectl get po kubia-manual -o json
$ kubectl get pods
$ kubectl logs kubia-manual
```

Connecting to pods

Connect without a service

```
$ kubectl port-forward kubia-manual 8888:8080
... Forwarding from 127.0.0.1:8888 -> 8080
... Forwarding from [::1]:8888 -> 8080
$ curl localhost:8888
```

Note that minikube uses nodeIP:nodePort and not localhost:nodePort. Check with "minikube ip".

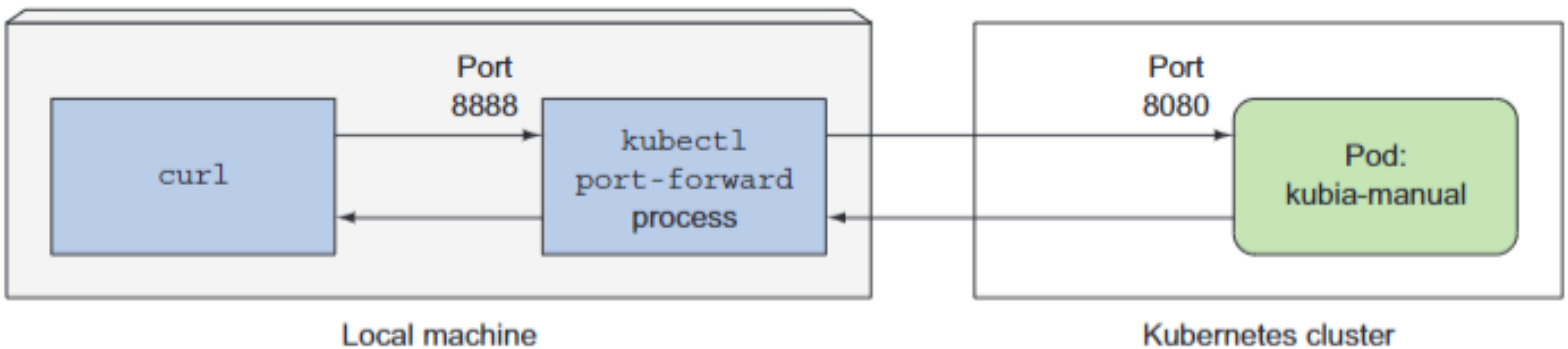


Figure: Fig. by courtesy of Marko Luksa[1]

Organizing pods with labels

- Use labels to organize all Kubernetes resources.
- One or more labels
- Vertical and horizontal.

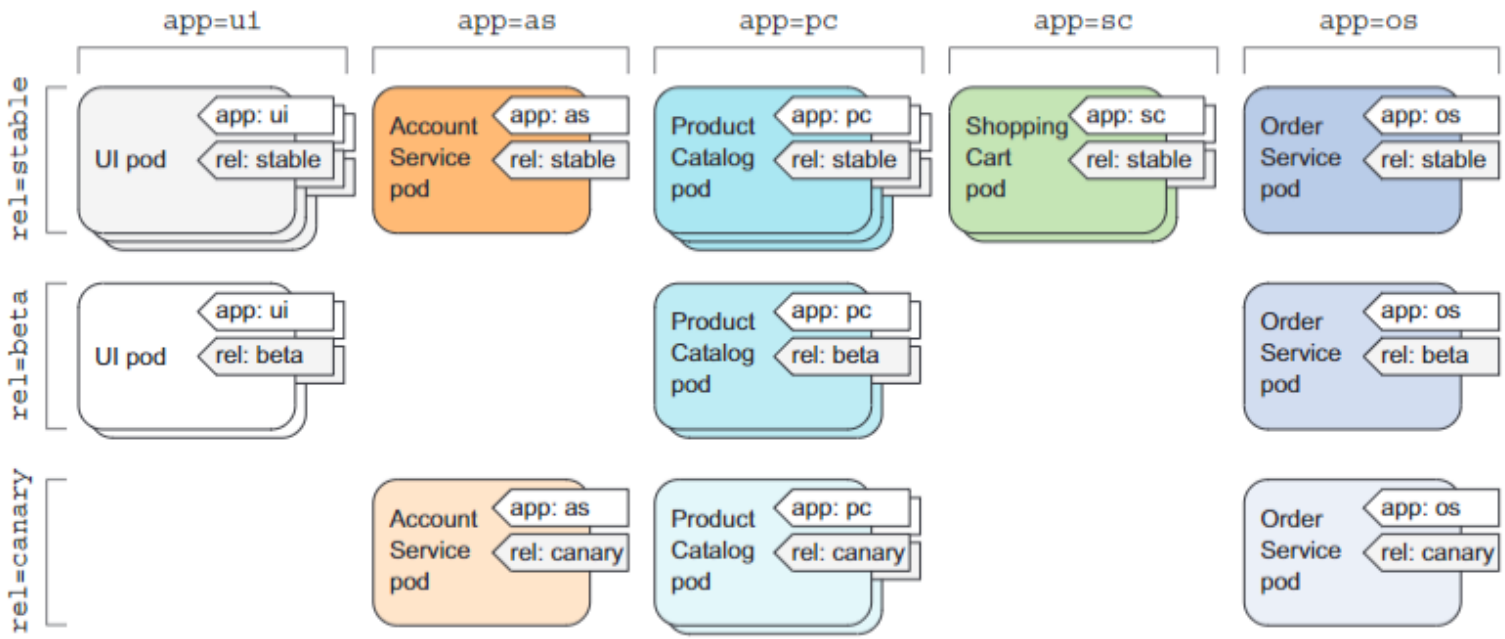


Figure: Fig. by courtesy of Marko Luksa[1]

Organizing pods with labels cont.

```
apiVersion: v1
kind: Pod
metadata:
  name: kubia-manual-v2
  labels:
    creation_method: manual
    env: prod
spec:
  containers:
  - image: luksa/kubia
    name: kubia
    ports:
    - containerPort: 8080
      protocol: TCP
```

Two labels are attached to the pod.

Figure: Listing by courtesy of Marko Luksa[1]

Organizing pods with labels cont.

Create and show pods with labels

```
$ kubectl create -f kubia-manual-with-labels.yaml
$ kubectl get po --show-labels
$ kubectl get po -L creation_method,env
$ kubectl get po -l creation_method=manual
```

- Dont worry about scheduling. Kubernetes handles that.
- Never say specifically what node a pod should run on.

Organizing pods with labels cont.

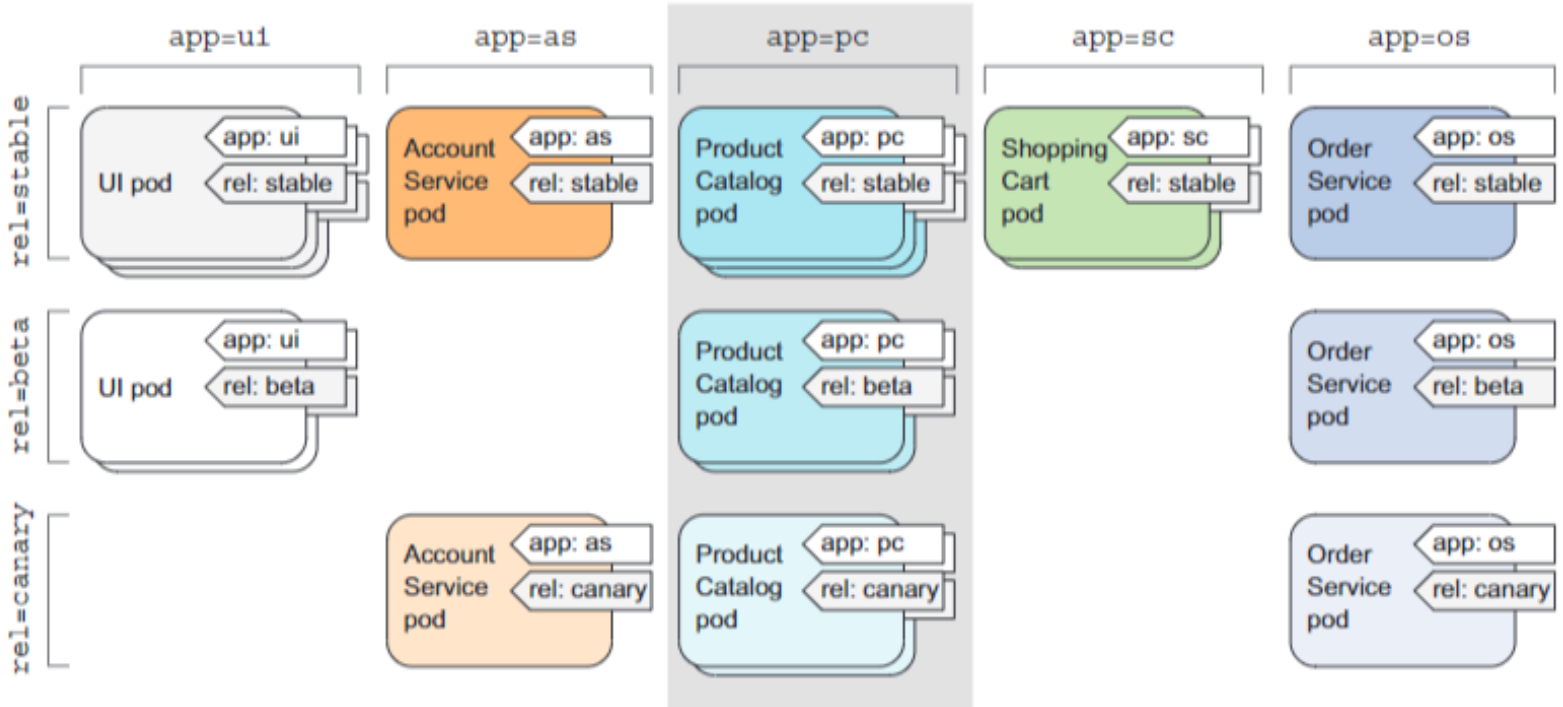


Figure: Fig. by courtesy of Marko Luksa[1]

Scheduling pods to specific nodes

Not best practice, but it is possible

```
$ kubectl label node gke-kubia-85f6-node-0rrx
gpu=true
$ kubectl get nodes -l gpu=true
```

```
apiVersion: v1
kind: Pod
metadata:
  name: kubia-gpu
spec:
  nodeSelector:
    gpu: "true"
  containers:
  - image: luksa/kubia
    name: kubia
```

nodeSelector tells Kubernetes to deploy this pod only to nodes containing the gpu=true label.

Figure: Listing by courtesy of Marko Luksa[1]

Stopping and removing pods

Kubernetes sends a SIGTERM, waits 30 seconds, then SIGKILL.

```
$ kubectl delete po kubia-gp
$ kubectl delete po -l creation_method=manual
$ kubectl delete po -l rel=canary
```

Stopping and removing pods cont.

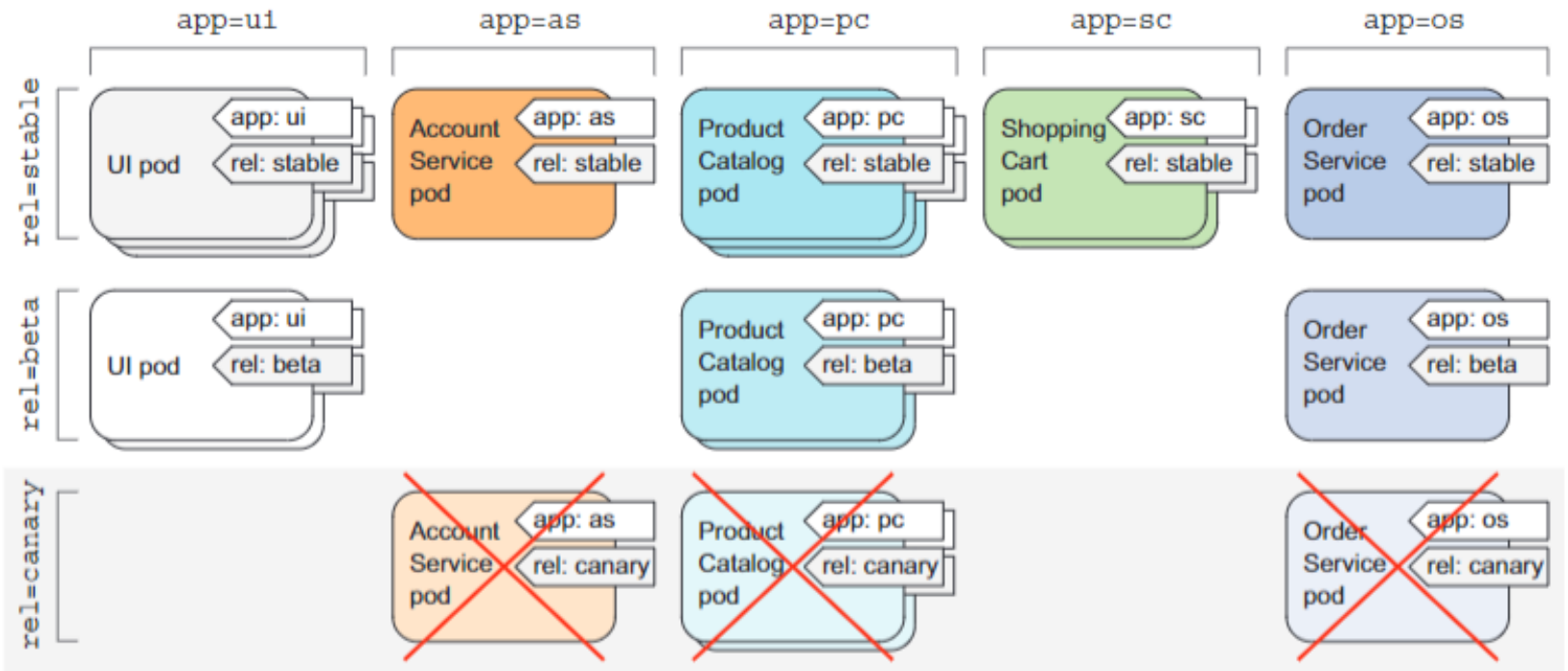


Figure: Fig. by courtesy of Marko Luksa[1]

Outline

Services in Kubernetes

Motivation

- We need a way to connect to pods from the outside.
- Pods are short-lived, they come and go.
- Clients should not know the IP's of pods.
- Scaling means multiple pods can provide the same service.

How do they work?

- A Service is a resource you create to make a single, constant point of entry to pods.
- Clients can now find frontend service, and frontend can find backend service.

Services in Kubernetes cont.

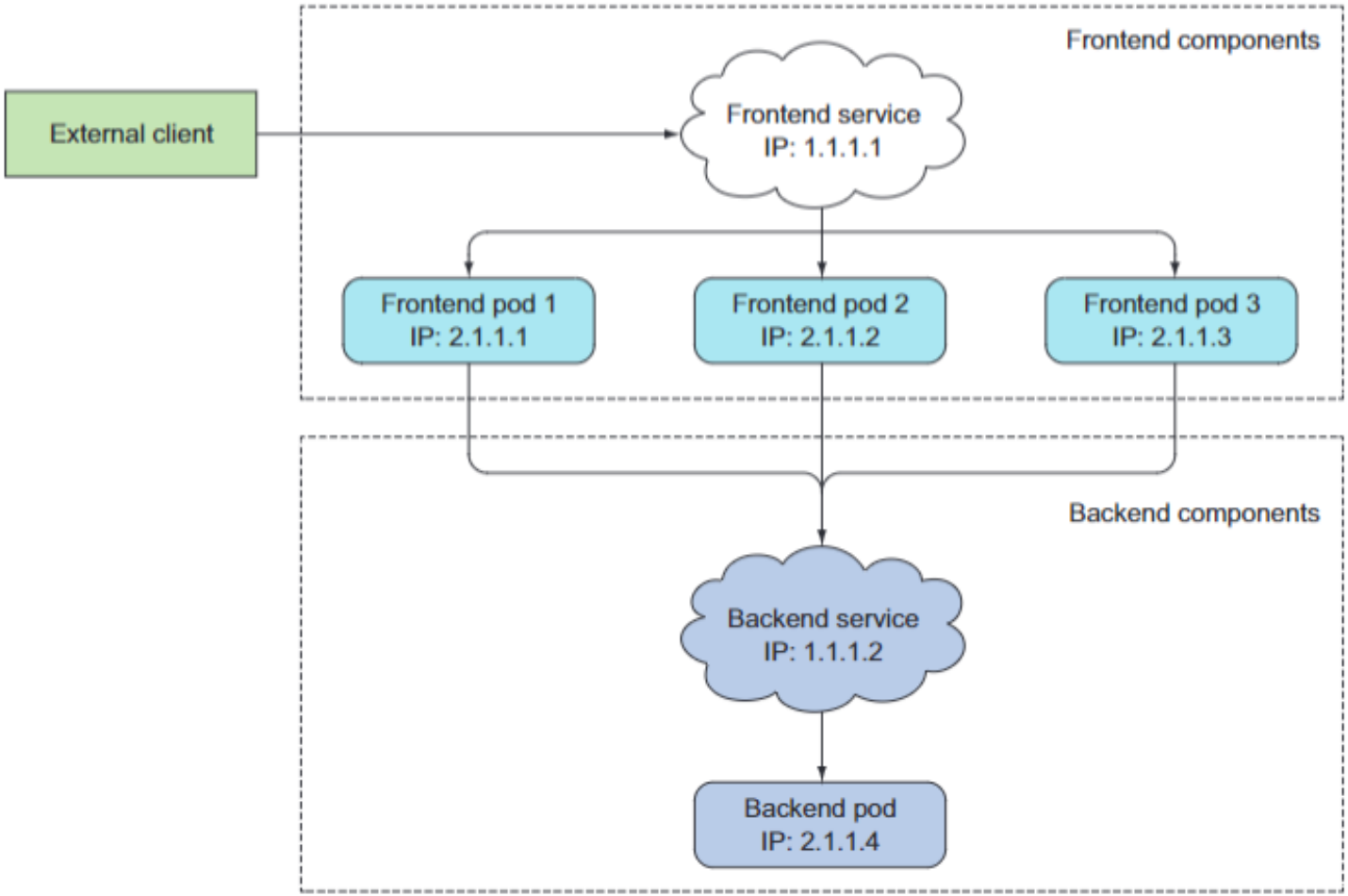


Figure: Fig. by courtesy of Marko Luksa[1]

Services in Kubernetes cont.

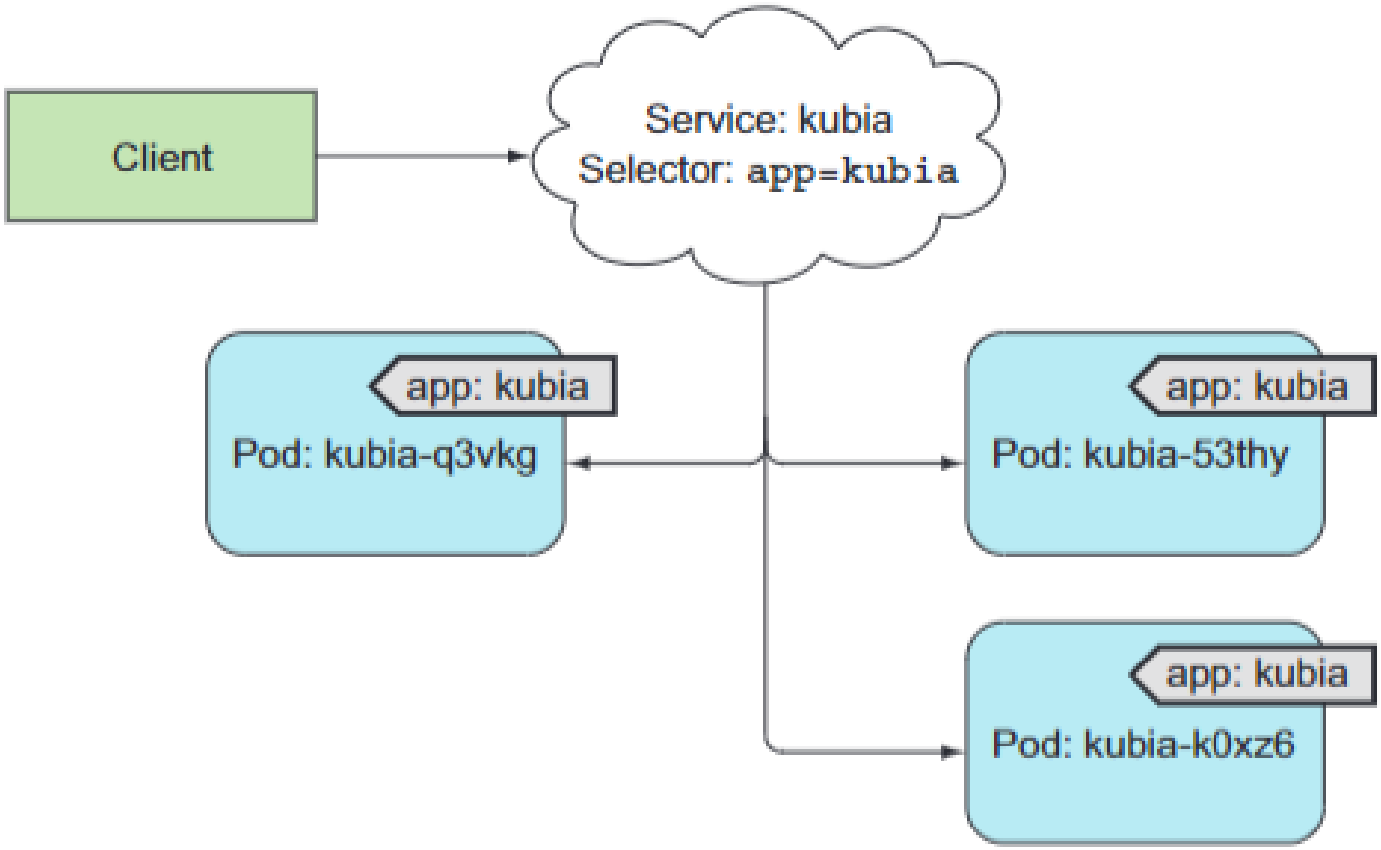


Figure: Fig. by courtesy of Marko Luksa[1]

Services in Kubernetes cont.

An example of a YAML file for a service. Our service accepts connections port 80 and route each connection to port 8080 to a pod matching *app=kubia*.

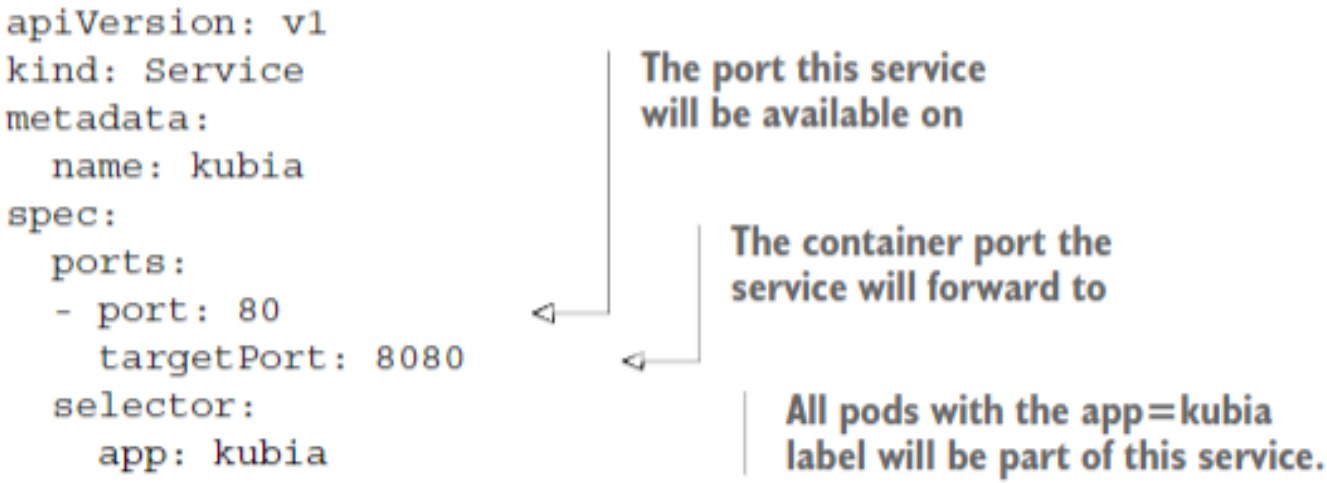


Figure: Listing by courtesy of Marko Luksa[1]

Services in Kubernetes cont.

When created, we check to see if the service is running

```
$ kubectl get svc
NAME: kubia
CLUSTER-IP: 10.111.249.153
EXTERNAL-IP: <none>
PORT(S): 80/TCP
AGE: 6m
```

Testing the service from the inside

With *kubectl exec* command we can access the service from inside
Replace with your target pod and cluster IP

```
$ kubectl exec kubia-7nog1 -- curl -s \\
http://10.111.249.153
You've hit kubia-gzwli
```

The *kubectl exec* behaves similarly to *ssh*.

Testing the service from the inside cont.

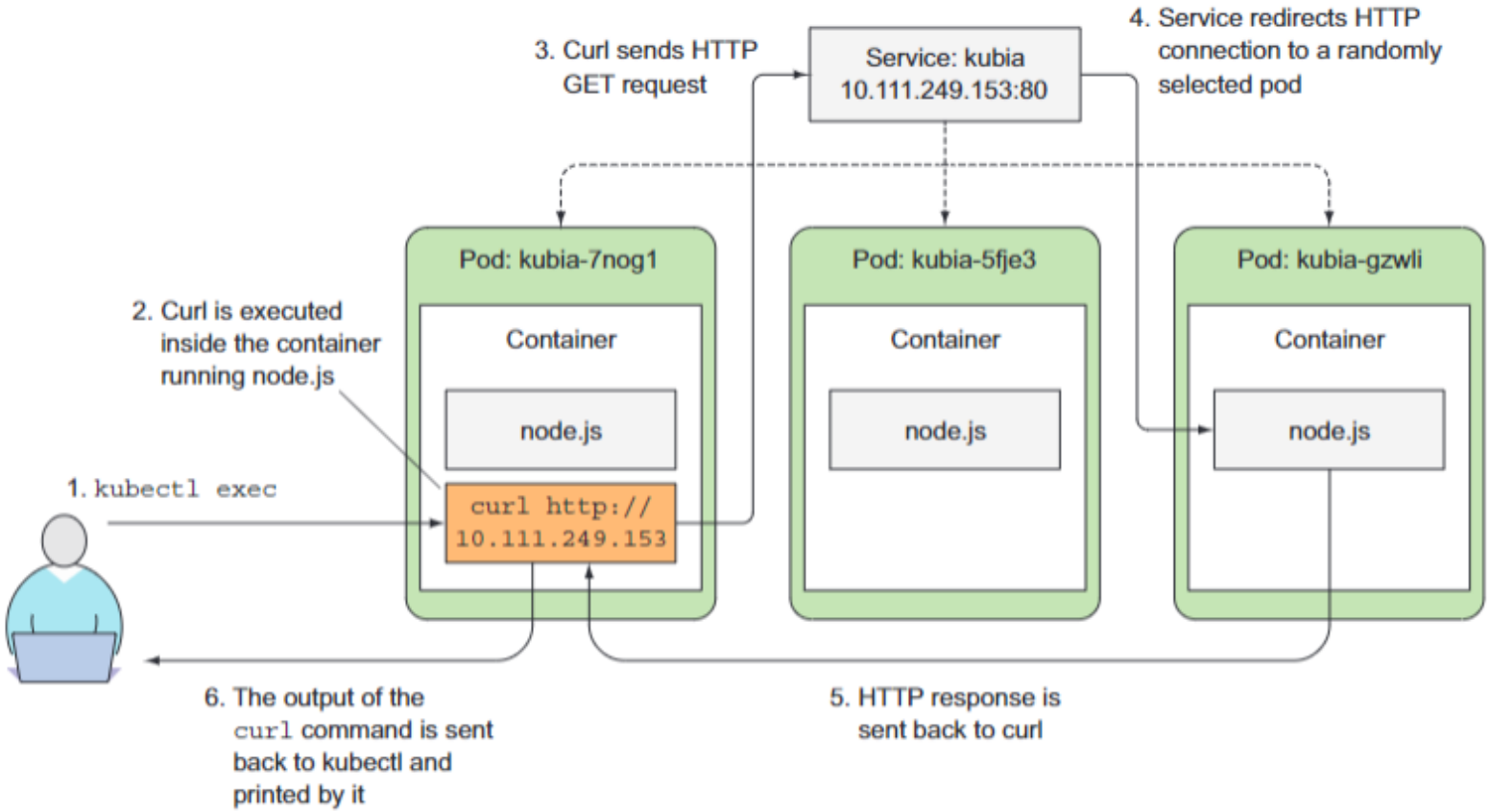


Figure: Listing by courtesy of Marko Luksa[1]

Discovering our service via DNS

- Kubernetes comes with a *kube-dns* for free
- Each service gets a DNS entry in the internal DNS server
- Client pods can access the service by its FQDN.

```
$ kubectl exec -it kubia-3inly
root@kubia-3inly:/#
$ curl http://kubia.default.svc.cluster.local
Youve hit kubia-5asi2
$ curl http://kubia.default
You've hit kubia-3inly
$ curl http://kubia
You've hit kubia-8awf3
```

Exposing services to external clients

- By setting the service type to NodePort (open up a port on the node itself)
- By setting the service type to LoadBalancer (deciated loud-balancer, AWS)
- By creating an Ingress resource (OSI level 7 resource)

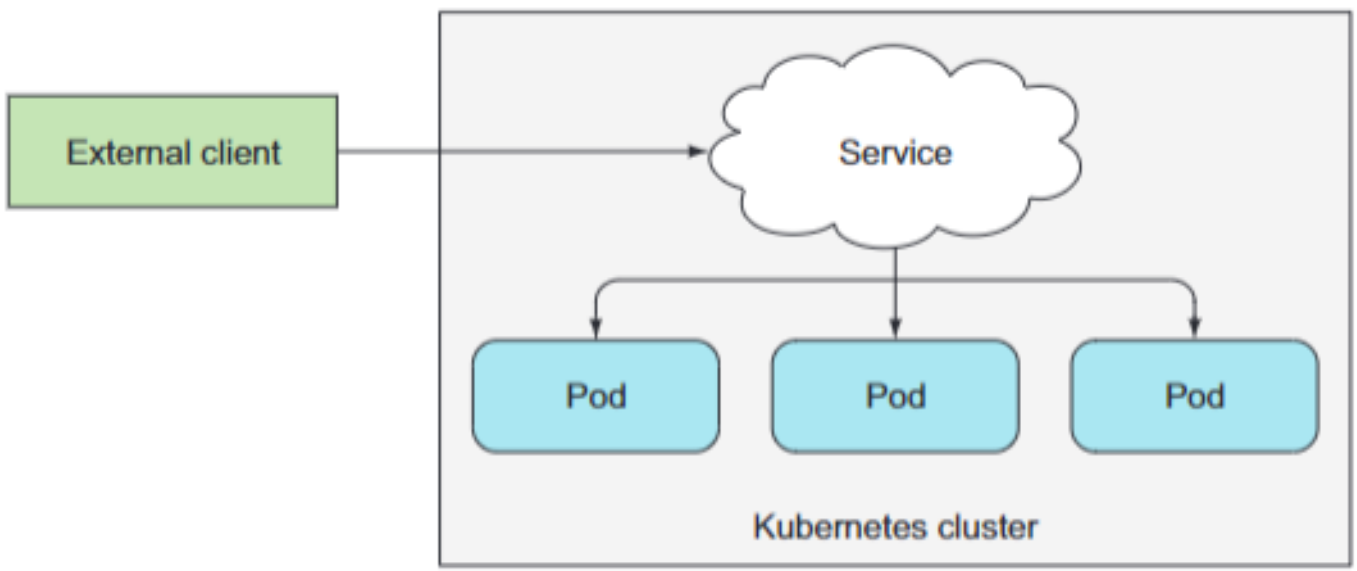


Figure: Fig. by courtesy of Marko Luksa[1]

Using a NodePort service

- For a NodePort service, each cluster node opens a port on the node itself and redirects traffic received on that port to the underlying service.
- This allows external traffic to our service.
- Can be configured with a YAML file.

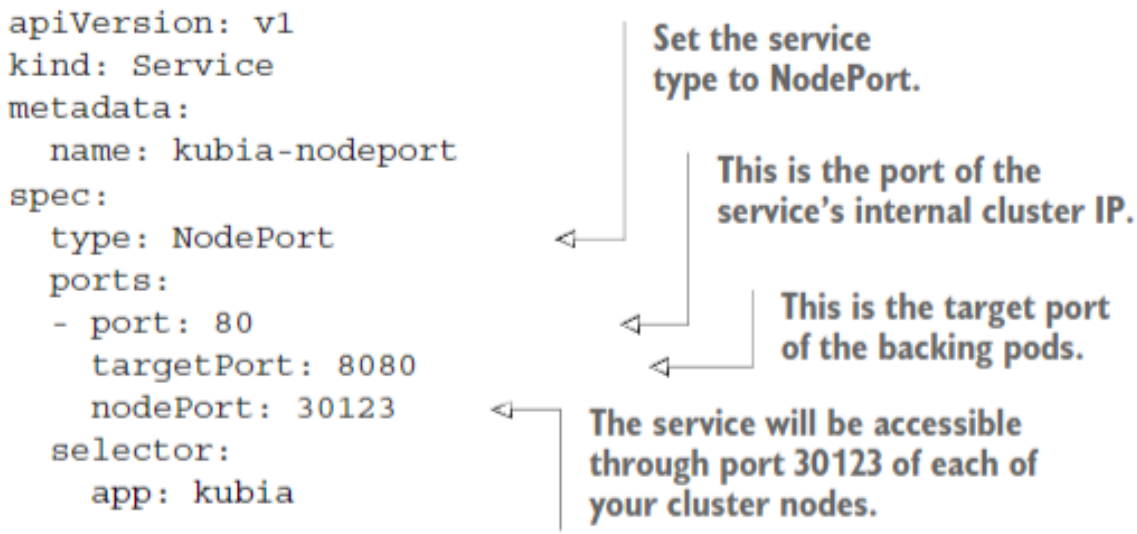


Figure: Listing by courtesy of Marko Luksa[1]

Using a NodePort service cont.

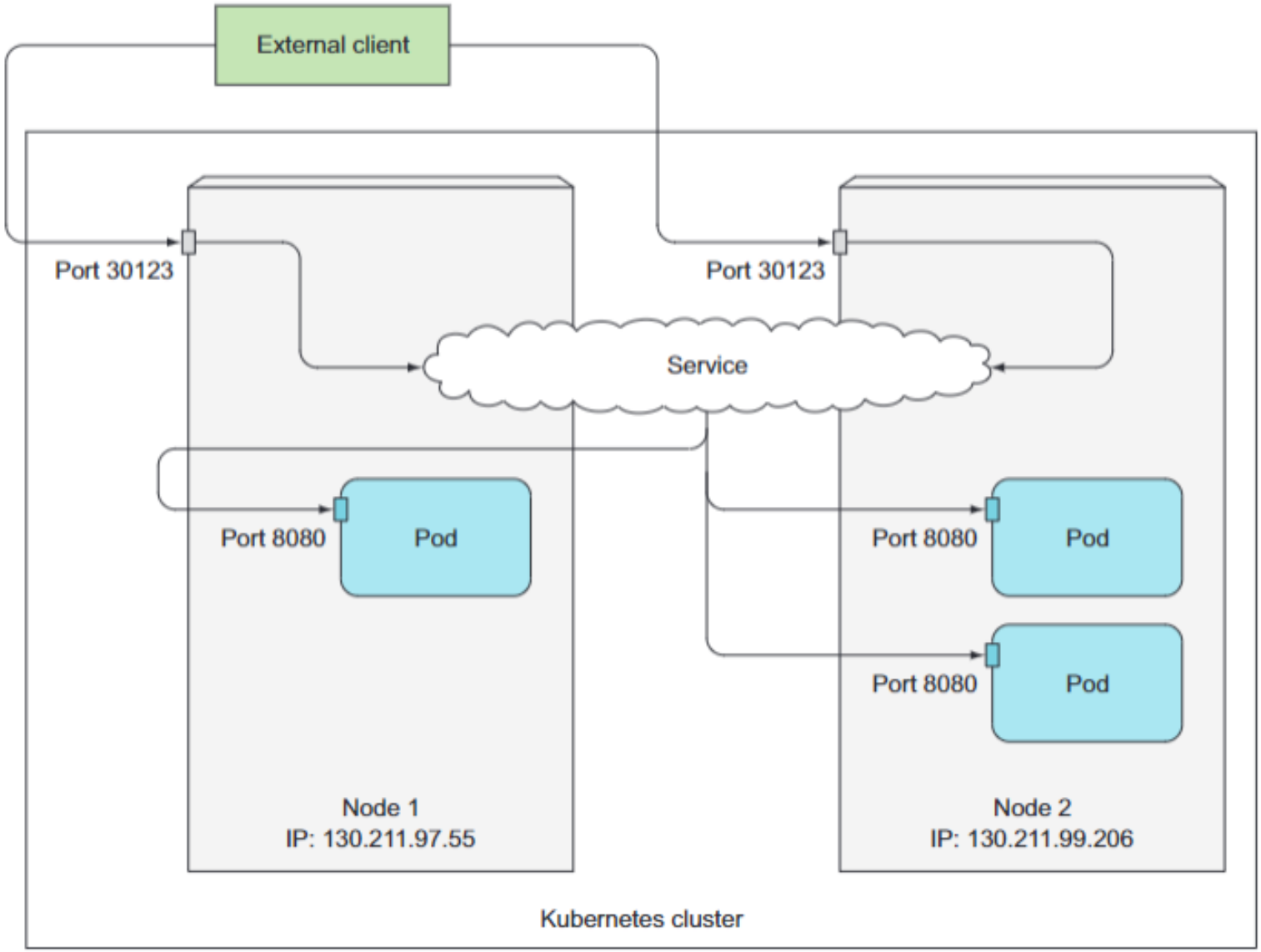
We can check our service with *kubectl get svc* command

```
$ kubectl get svc kubia-nodeport
NAME: kubia-nodeport
CLUSTER-IP: 10.111.254.223
EXTERNAL-IP: <nodes>
PORT(S): 80:30123/TCP
AGE: 2m
```

The service is now accessible at the following addresses:

- 10.11.254.223:80
- <1st node's IP>:30123
- <2nd node's IP>:30123, and so on.

Using a NodePort service cont.



Using a LoadBalancer service

- A LoadBalancer service is the default way to expose a service to the Internet
- Each service gets its own IP
- Was not supported by minikube until recently
- Available by its EXTERNAL-IP

```
apiVersion: v1
kind: Service
metadata:
  name: kubia-loadbalancer
spec:
  type: LoadBalancer
  ports:
  - port: 80
    targetPort: 8080
  selector:
    app: kubia
```

This type of service obtains a load balancer from the infrastructure hosting the Kubernetes cluster.

Figure: Listing by courtesy of Marko Luksa[1]

Using a LoadBalancer service cont.

We can check our service with *kubectl get svc* command

```
$ kubectl get svc kubia-loadbalancer
NAME: kubia-loadbalancer
CLUSTER-IP: 10.111.241.153
EXTERNAL-IP: 130.211.53.173
PORT(S): 80:32143/TCP
AGE: 1m
```

The service is now accessible by its external IP:
curl http://130.211.53.173

Using a LoadBalancer service cont.

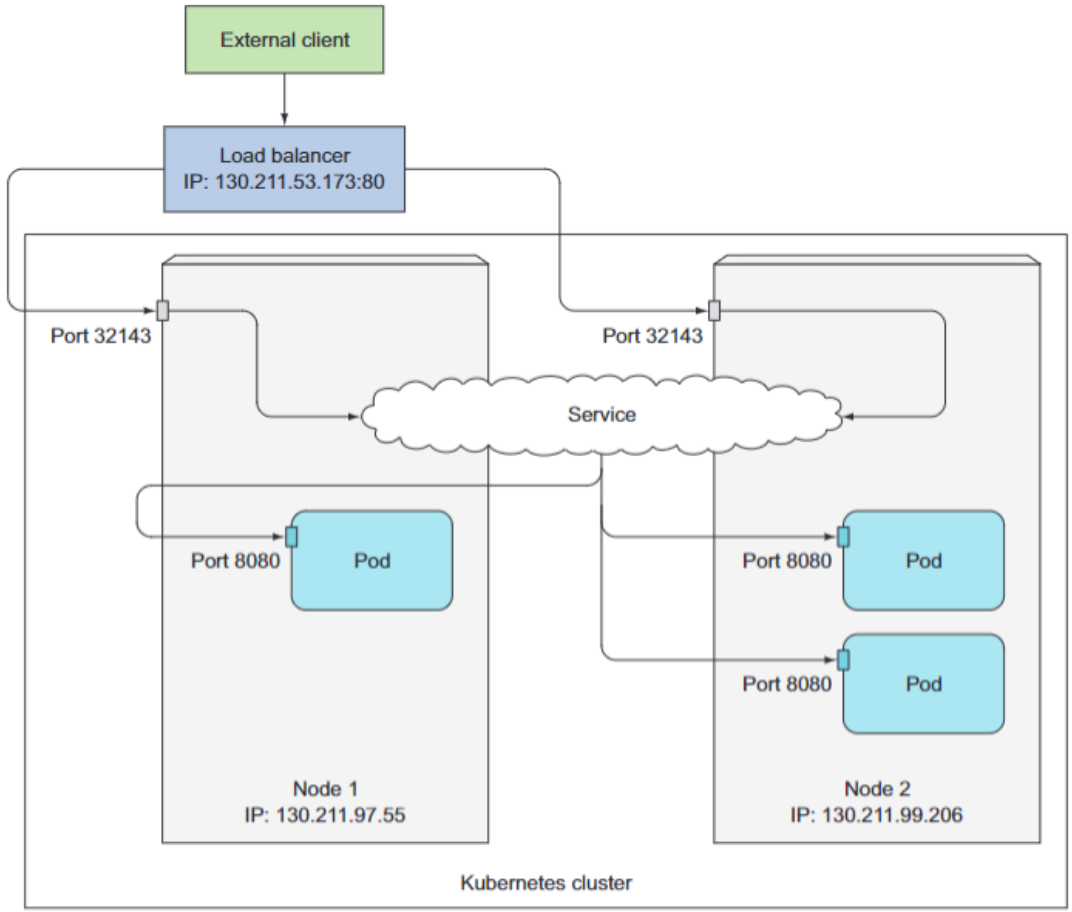


Figure: Fig. by courtesy of Marko Luksa[1]

Using a headless service

Needed when client needs to connect to all pods

- A normal service just connects you to one pod
- A headless service will return the IP's of all annotated pods
- Can be accessed via DNS lookup
- Client libraries exist to ease communication with API

```
apiVersion: v1
kind: Service
metadata:
  name: kubia-headless
spec:
  clusterIP: None
  ports:
  - port: 80
    targetPort: 8080
  selector:
    app: kubia
```

This makes the service headless.

Figure: Listing by courtesy of Marko Luksa[1]

Using a headless service cont.

We can use the dnsutils image to verify a headless service

```
$ kubectl run dnsutils --image=tutum/dnsutils \\
--generator=run-pod/v1 --command -- \\
sleep infinity
pod "dnsutils" created
$ kubectl exec dnsutils nslookup kubia-headless
...
Name:    kubia-headless.default ...
Address: 10.108.1.4
Name:    kubia-headless.default ...
Address: 10.108.2.5
```

Outline

Deployments in Kubernetes

What are deployments?

- A Deployment is a high-level resource
- It can deploy applications and update them declaratively
- Makes it easy to manage and make rolling-updates
- A Deployment is composed of a label selector, a replica count, and a pod template.

Figure: Fig. by courtesy of Marko Luksa[1]

Deployments in Kubernetes cont.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: kubia
spec:
  replicas: 3
  template:
    metadata:
      name: kubia
      labels:
        app: kubia
    spec:
      containers:
      - image: luksa/kubia:v1
        name: nodejs
```

You've changed the kind from ReplicationController to Deployment.

There's no need to include the version in the name of the Deployment.

Figure: Listing by courtesy of Marko Luksa[1]

```
$ kubectl create -f kubia-deployment-v1.yaml
--record
$ kubectl rollout status deployment kubia
$ kubectl get po
```

Rolling out updates

To roll-out an update, simply do

```
$ kubectl set image deployment kubia
nodejs=luksa/kubia:v2
```

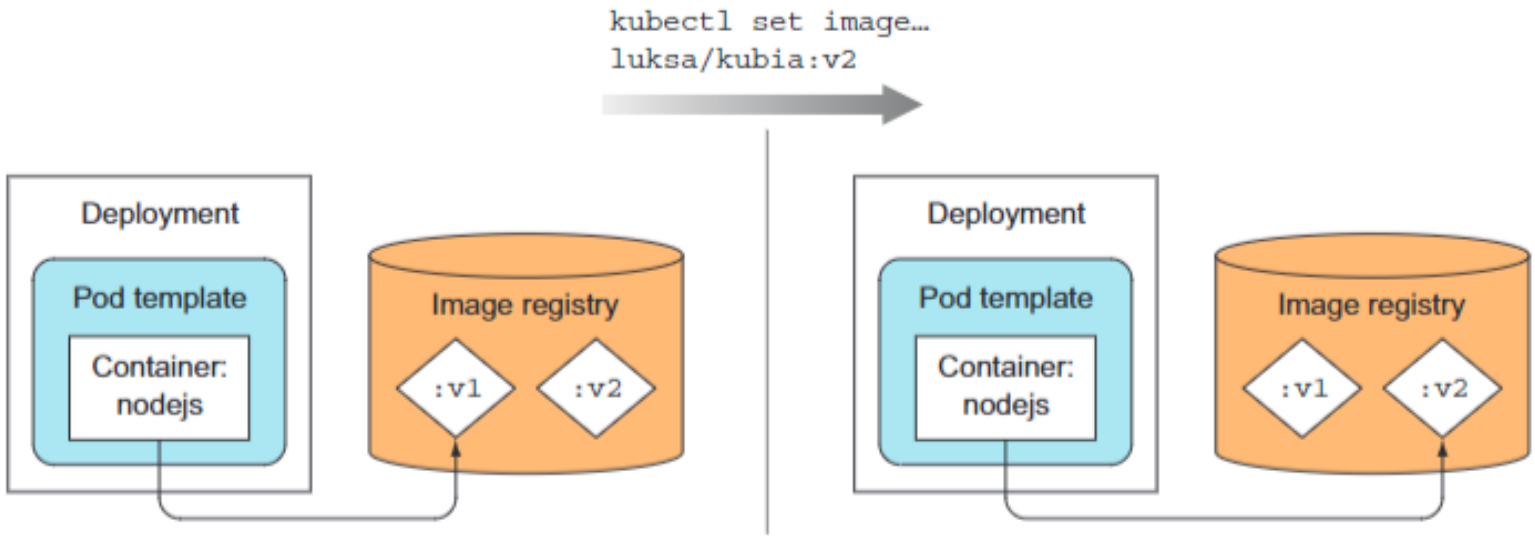


Figure: Fig. by courtesy of Marko Luksa[1]

Rolling out updates cont.

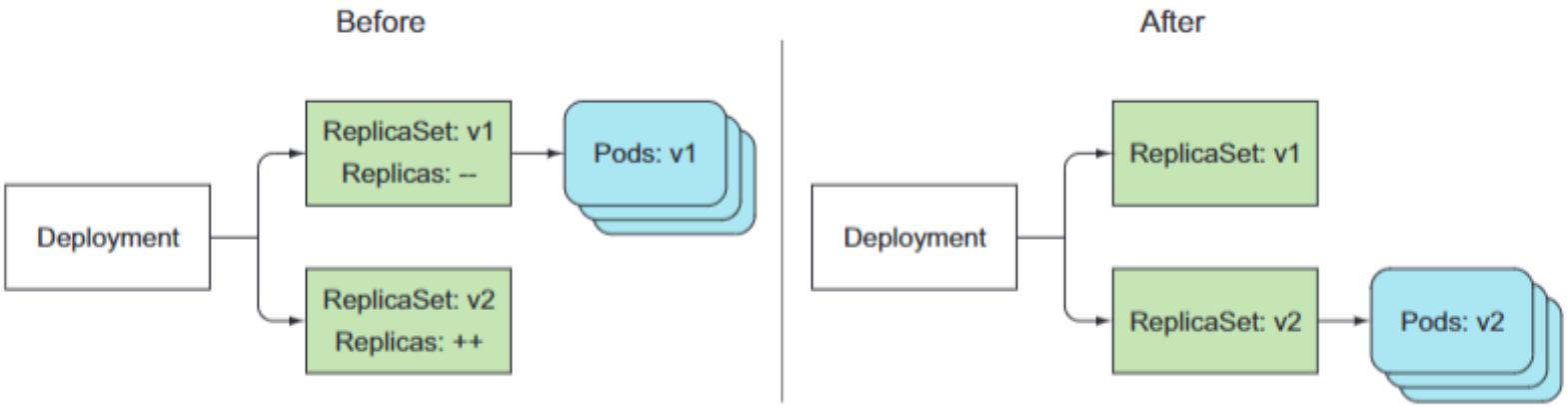


Figure: Fig. by courtesy of Marko Luksa[1]

Undoing a rollout

Deployments make it easy to roll back an update.

```
$ kubectl rollout undo deployment kubia
```

View and using history

We can check the history of our deployments

```
$ kubectl rollout history deployment kubia
$ kubectl rollout undo deployment kubia
--to-revision=1
```

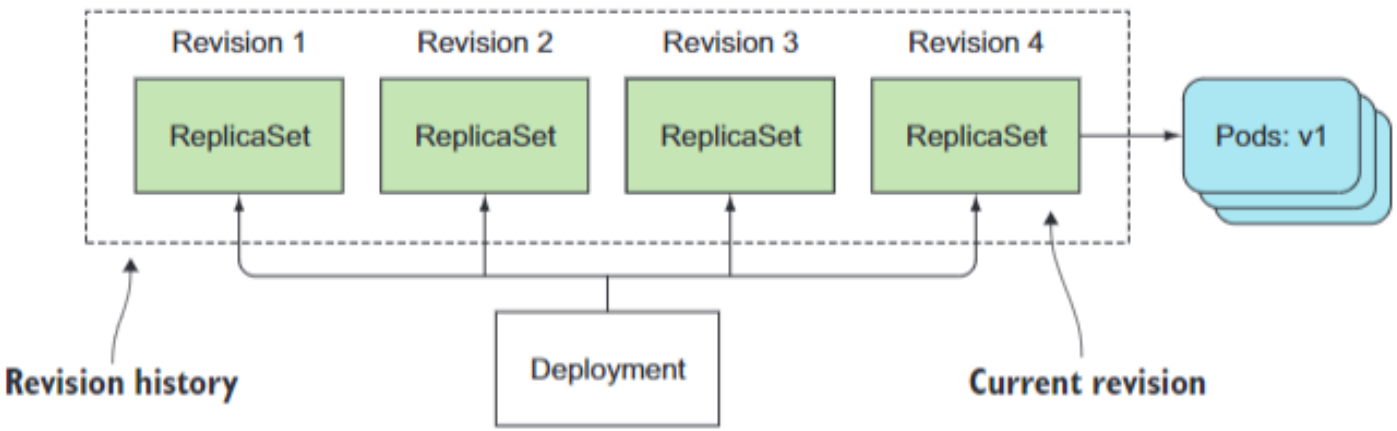


Figure: Fig. by courtesy of Marko Luksa[1]

Control the rate of the rollout

You can control the max surge and max unavailable pods in the Deployment manifest

```
spec:
  strategy:
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 0
    type: RollingUpdate
```

Figure: Listing by courtesy of Marko Luksa[1]

- maxSurge: Determines how many pods you allow to exist above the desired replica count. Default 25
- maxUnavailable: Determines how many pods can be unavailable relative to the desired replica count. Default 25

Control the rate of the rollout cont.

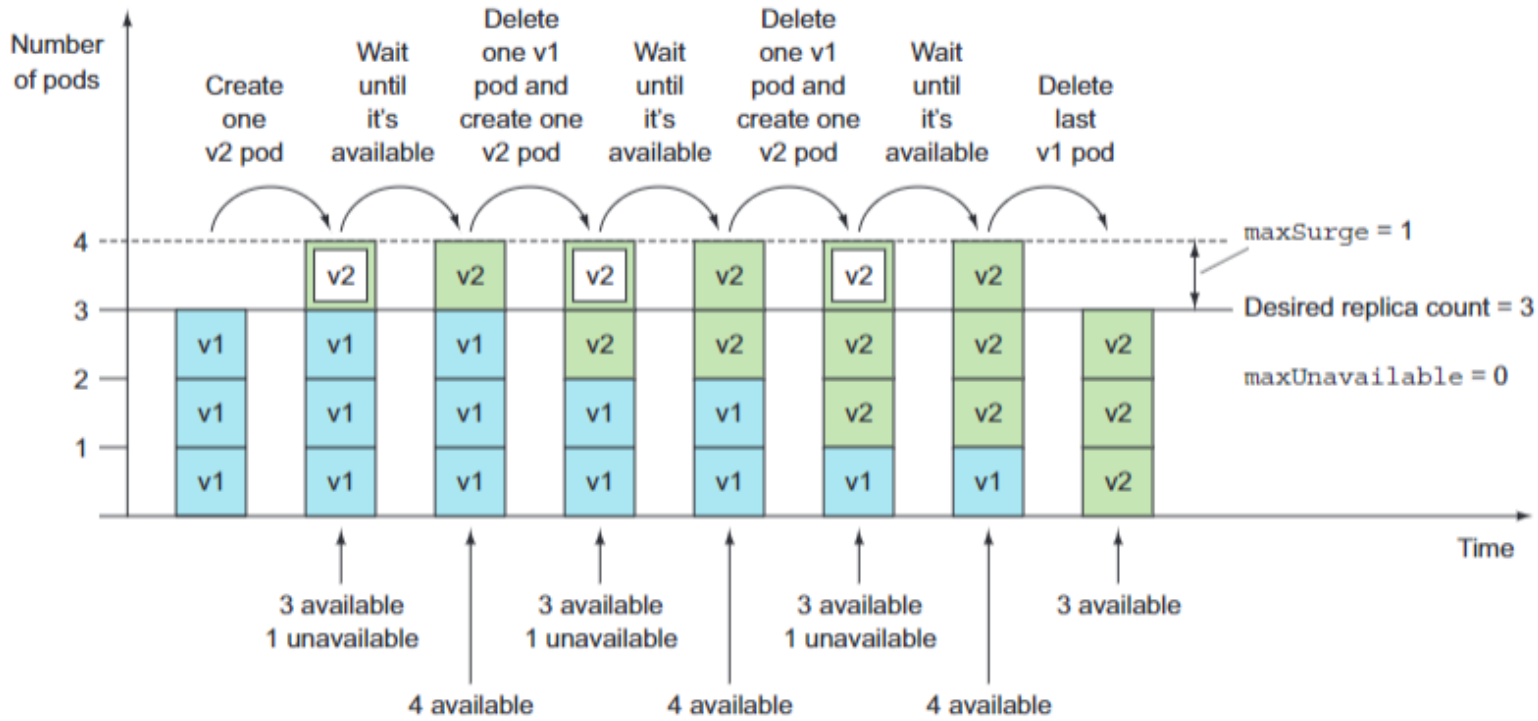


Figure: Fig. by courtesy of Marko Luksa[1]

Control the rate of the rollout cont.

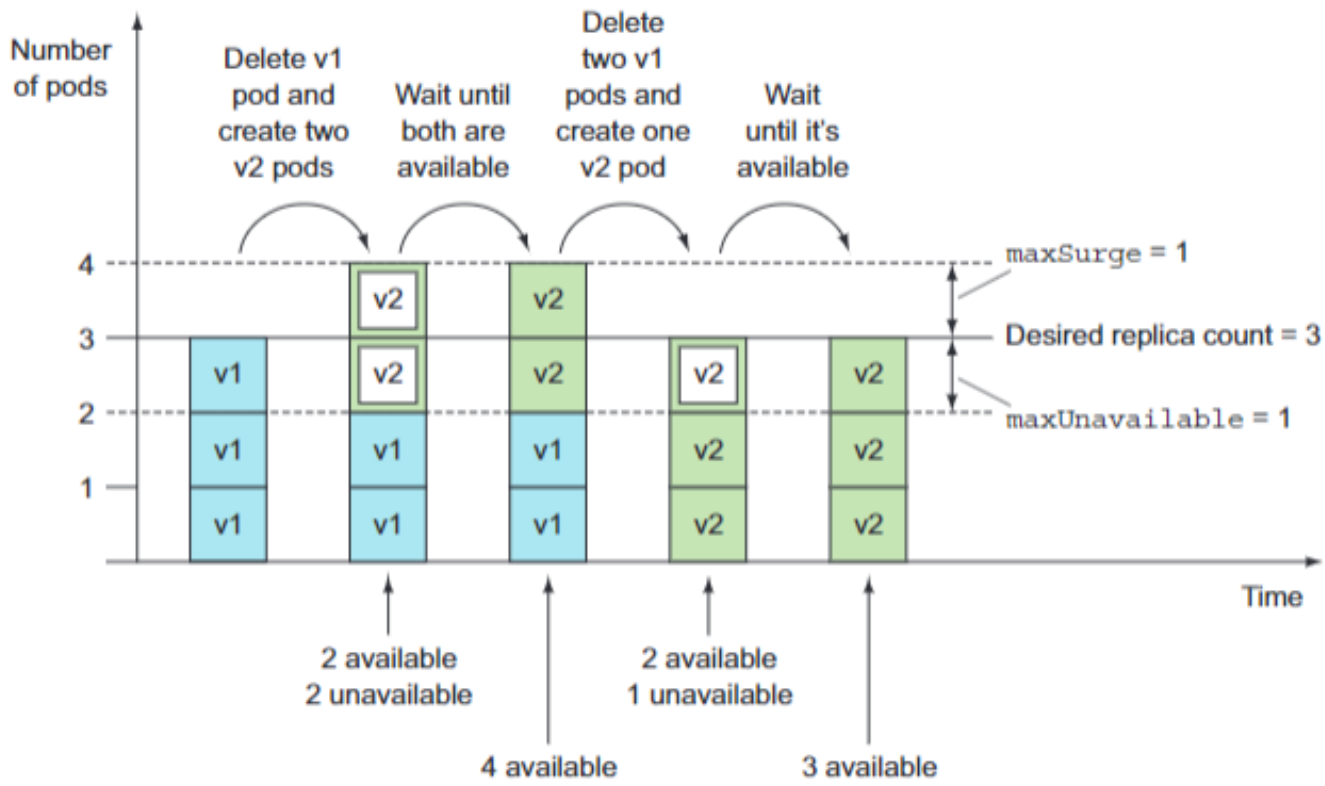


Figure: Fig. by courtesy of Marko Luksa[1]

Different ways to modify resources

There are several ways to modify a resource in Kubernetes

- kubectl edit - Opens a object's manifest in default editor
- kubectl patch - Changes individual properties of an object
- kubectl apply - Changes or creates an object from a YAML/JSON file
- kubectl replace - Replaces existing object from a YAML/JSON file
- kubectl set image - Changes a deployment's container image

References I

[1] Luksa, M. (2018). *Kubernetes in Action*. Manning Publications Co.