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 January 21, 2021

Outline

Outline

Pods in Kubernetes

A short brush-up from last time

- 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

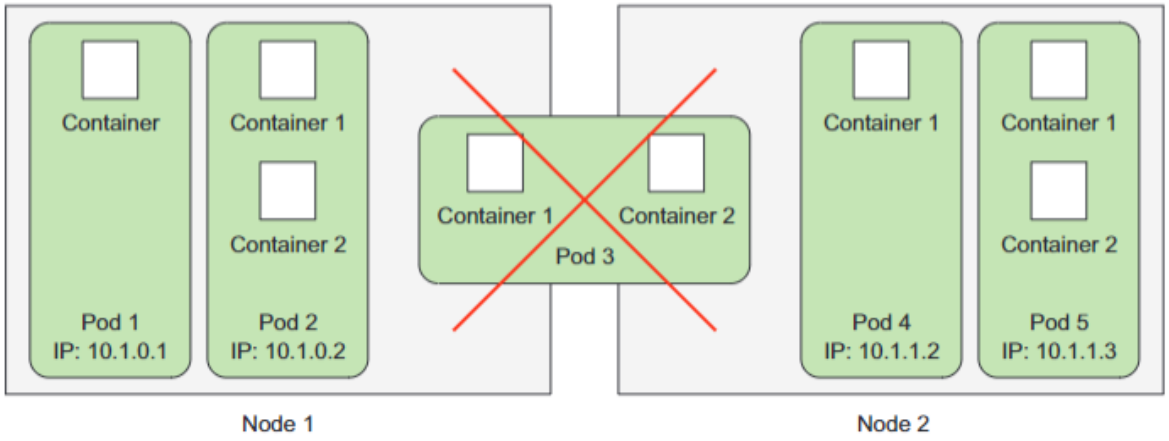


Figure: Fig. by courtesy of Marko Luksa[1]

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

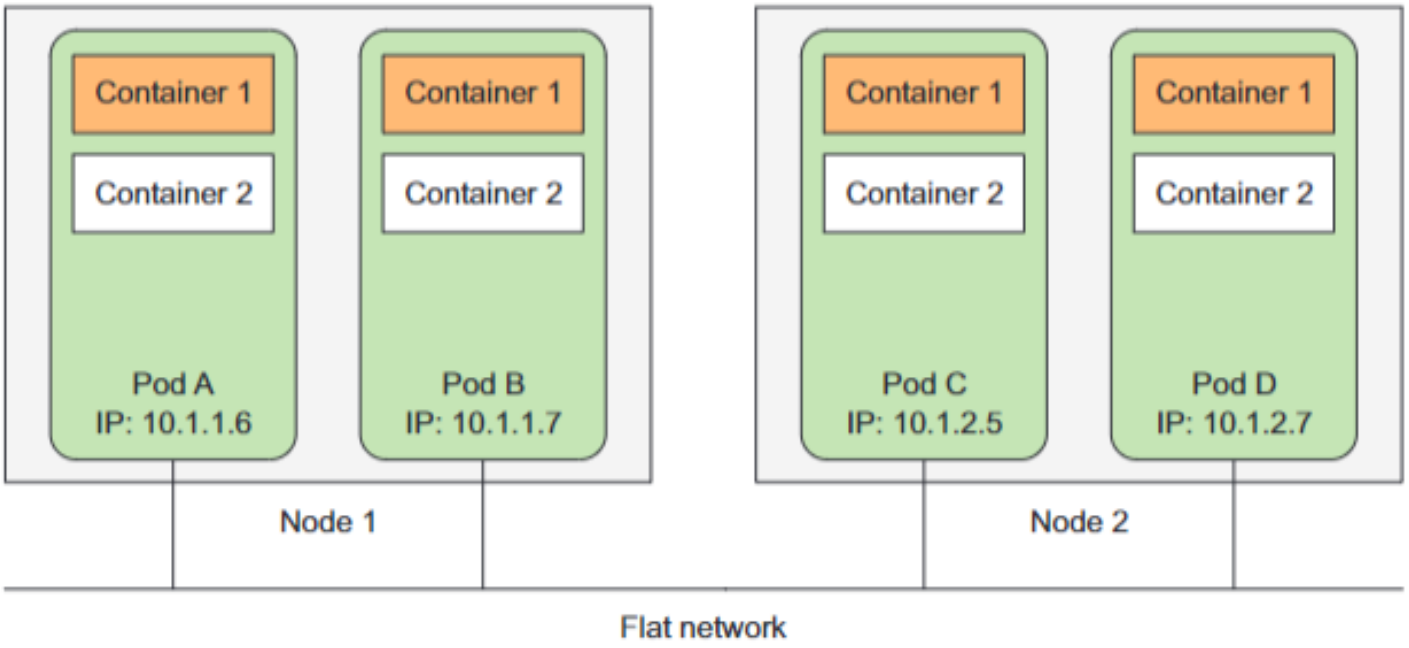


Figure: Fig. by courtesy of Marko Luksa[1]

The inside of a pod

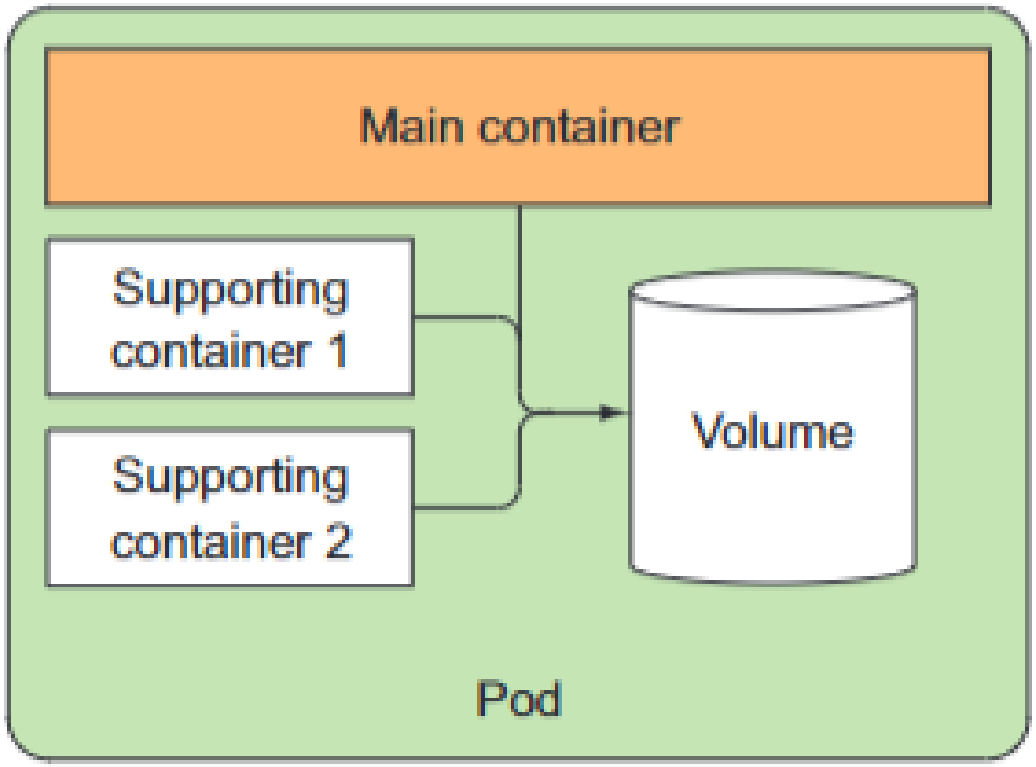


Figure: Fig. by courtesy of Marko Luksa[1]

Using multiple containers

When to use multiple containers?

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

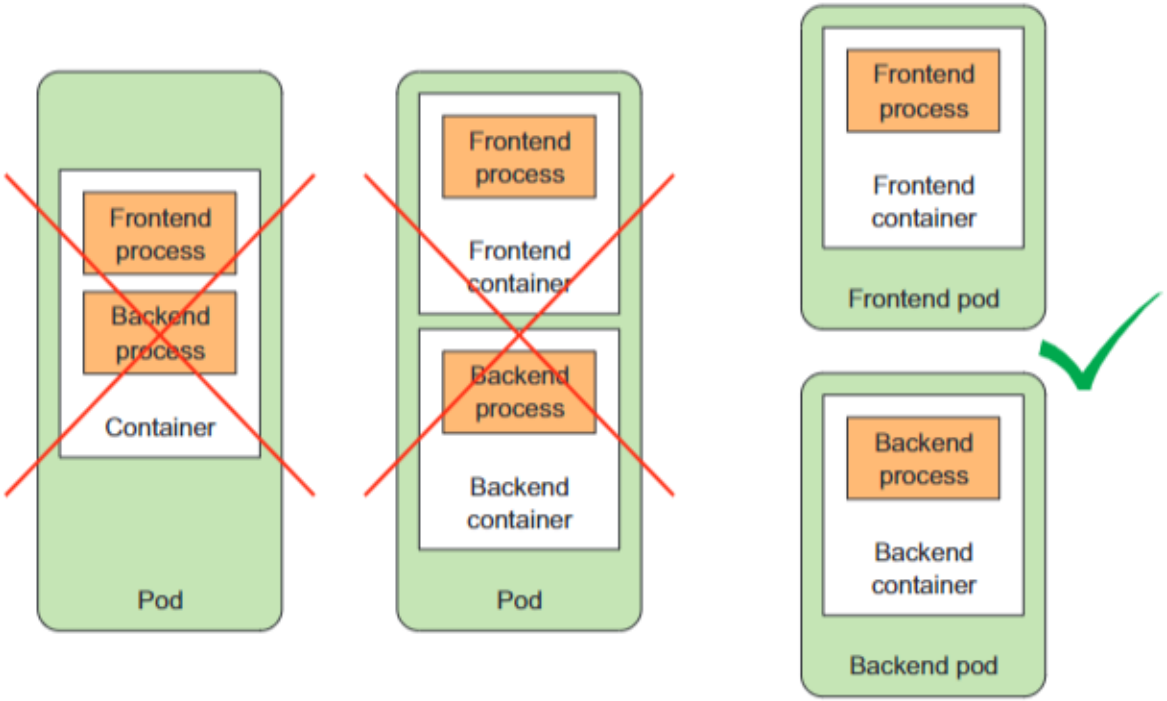


Figure: Fig. by courtesy of Marko Luksa[1]

Creating pods

- Created by posting a YAML or JSON to the Kubernetes API
- Instead of "kubectl run", you post a YAML file

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

Descriptor conforms to version v1 of Kubernetes API

You're describing a pod.

The name of the pod

Container image to create the container from

Name of the container

The port the app is listening on

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
$ curl localhost:8888
```

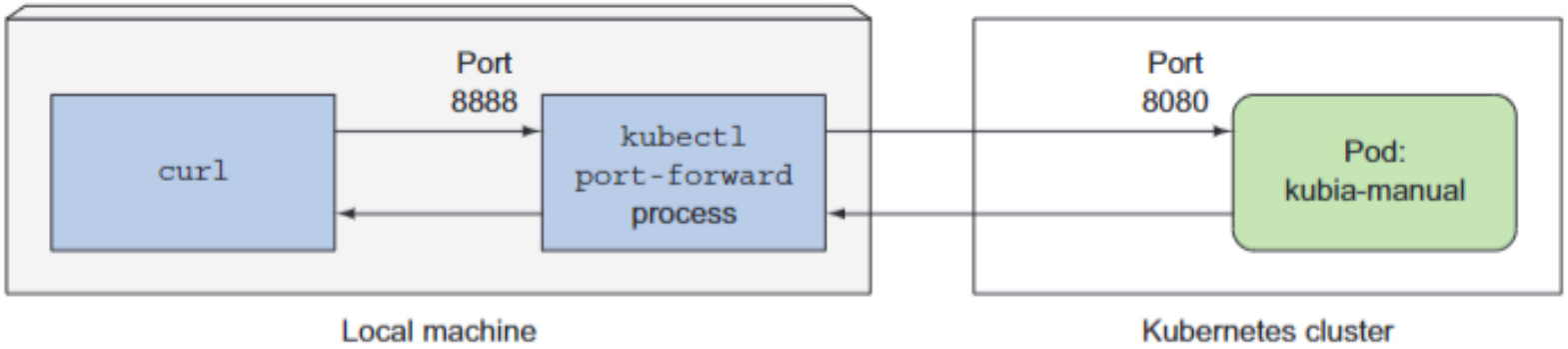


Figure: Fig. by courtesy of Marko Luksa[1]

Organizing pods with labels

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

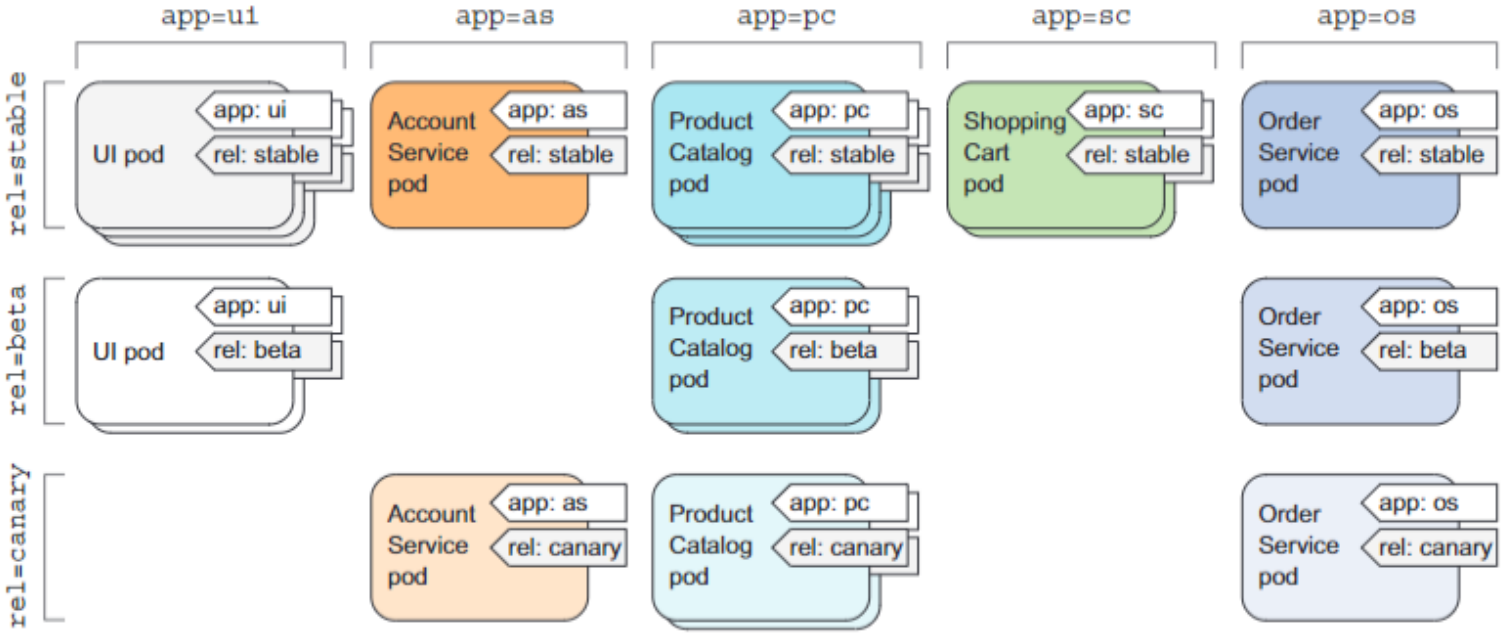


Figure: Fig. by courtesy of Marko Luksa[1]

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.

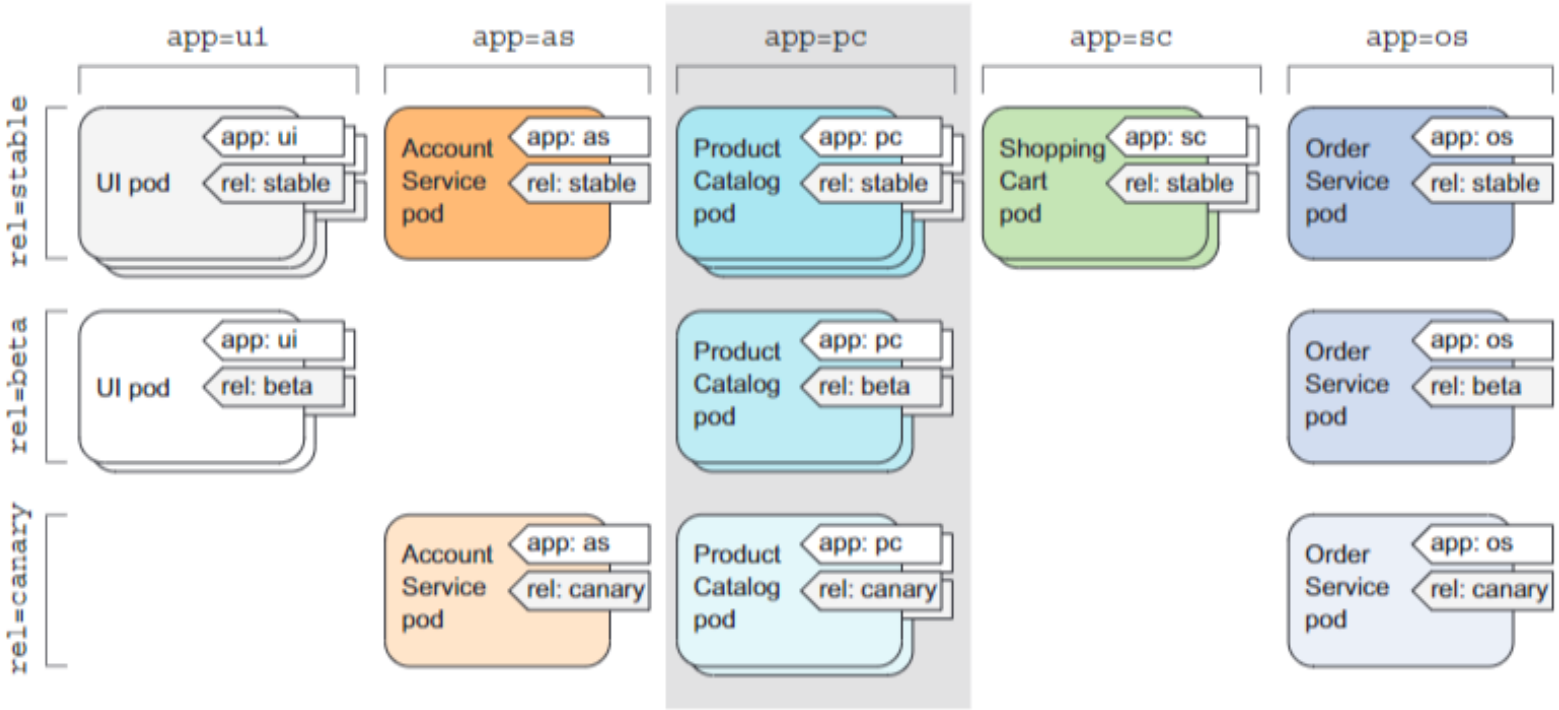


Figure: Fig. by courtesy of Marko Luksa[1]

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.

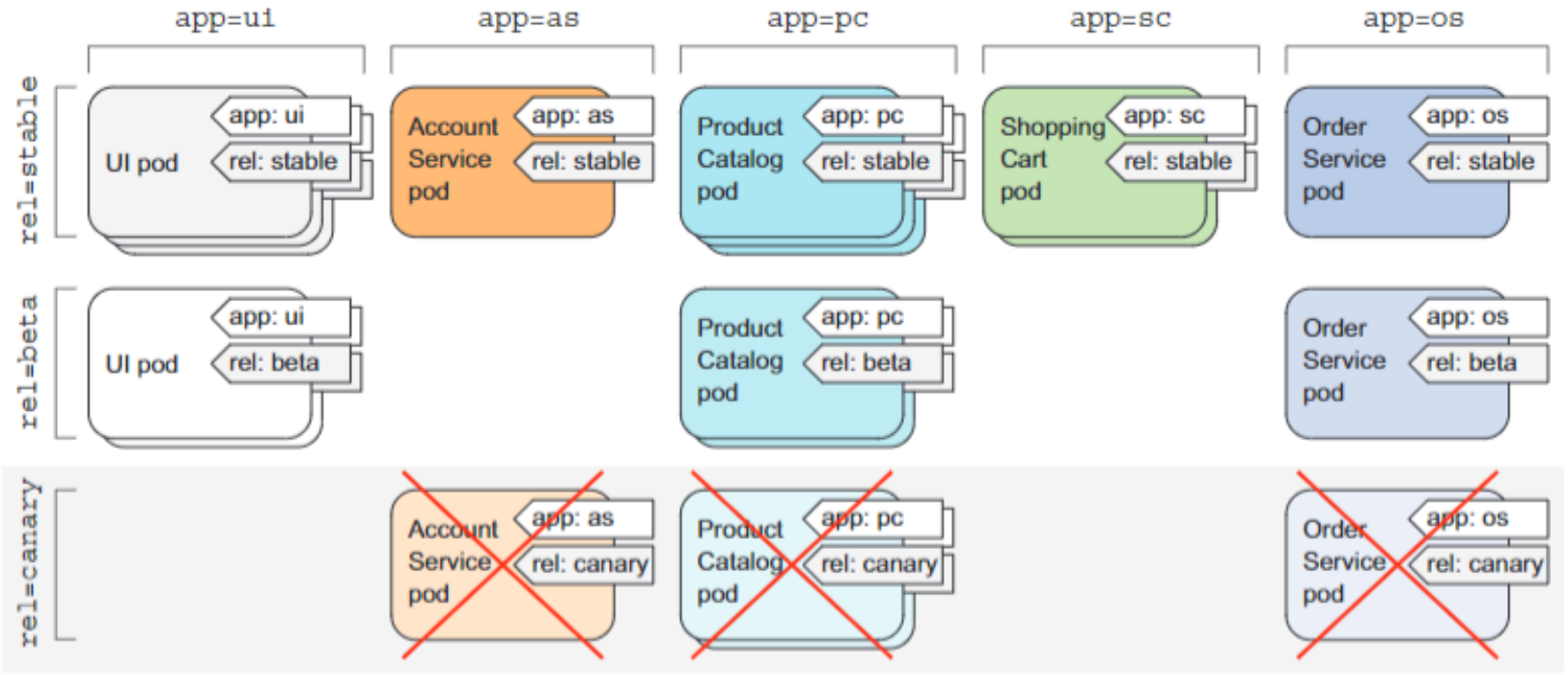


Figure: Fig. by courtesy of Marko Luksa[1]

Outline

Services in Kubernetes

Motivation

- We need a way to connect to pods from the outside.
- Pods are ephemeral, they come and go.
- Clients don't 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.

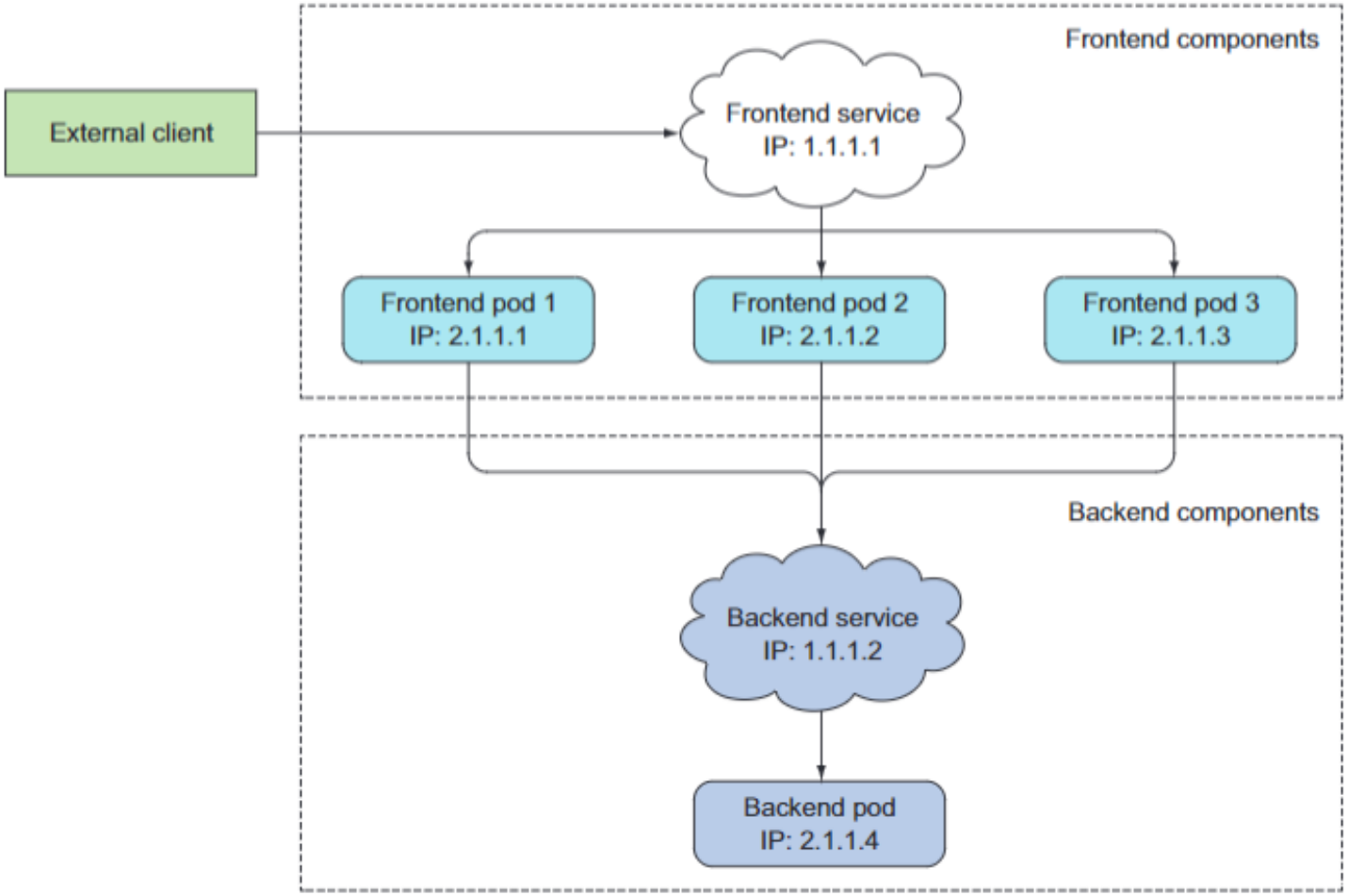


Figure: Fig. by courtesy of Marko Luksa[1]

Services in Kubernetes cont.

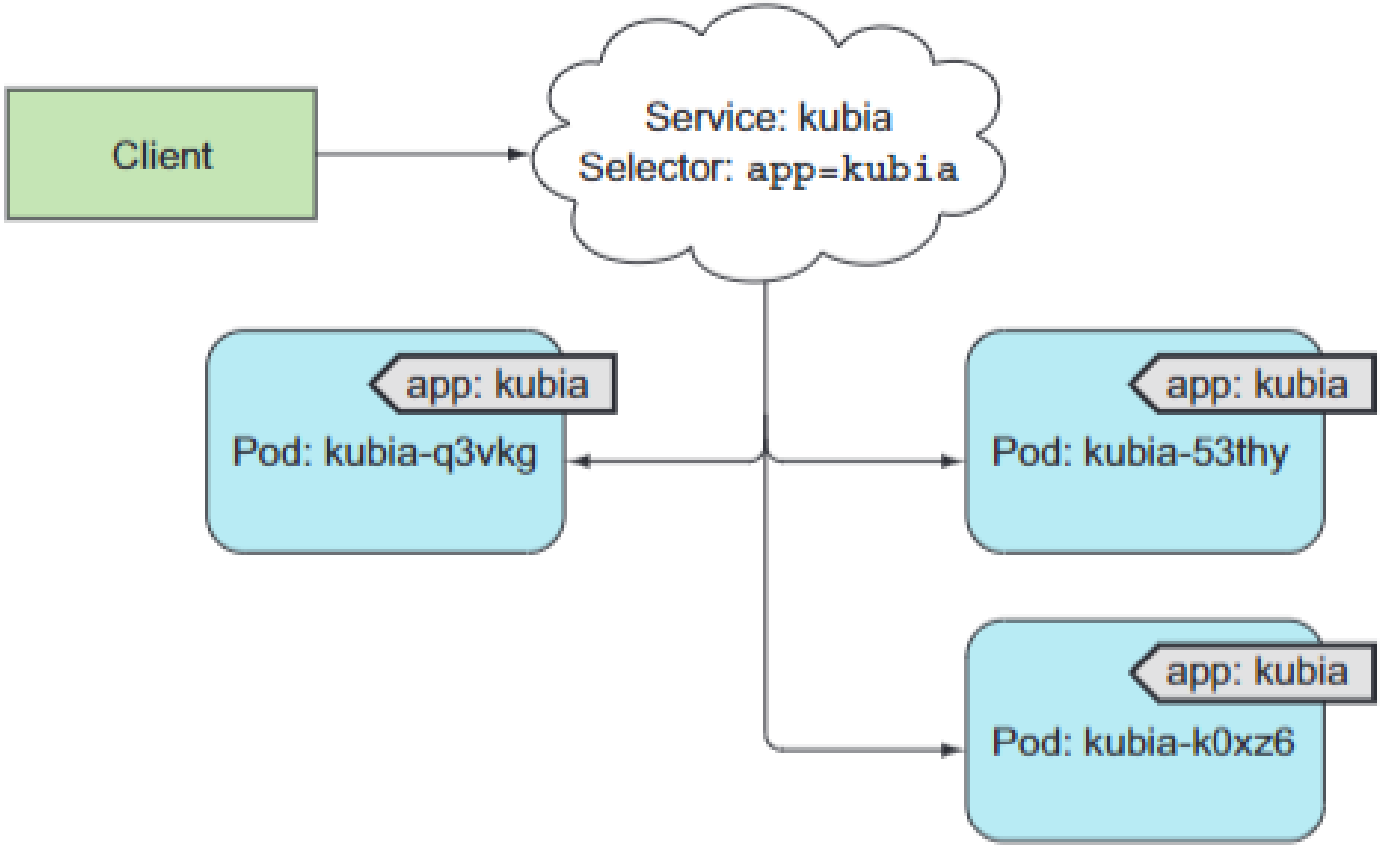


Figure: Fig. by courtesy of Marko Luksa[1]

Services in Kubernetes cont.

An example of a YAML file for a service

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

The port this service will be available on

The container port the service will forward to

All pods with the app=kubia label will be part of this service.

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

Exposing services to external clients

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

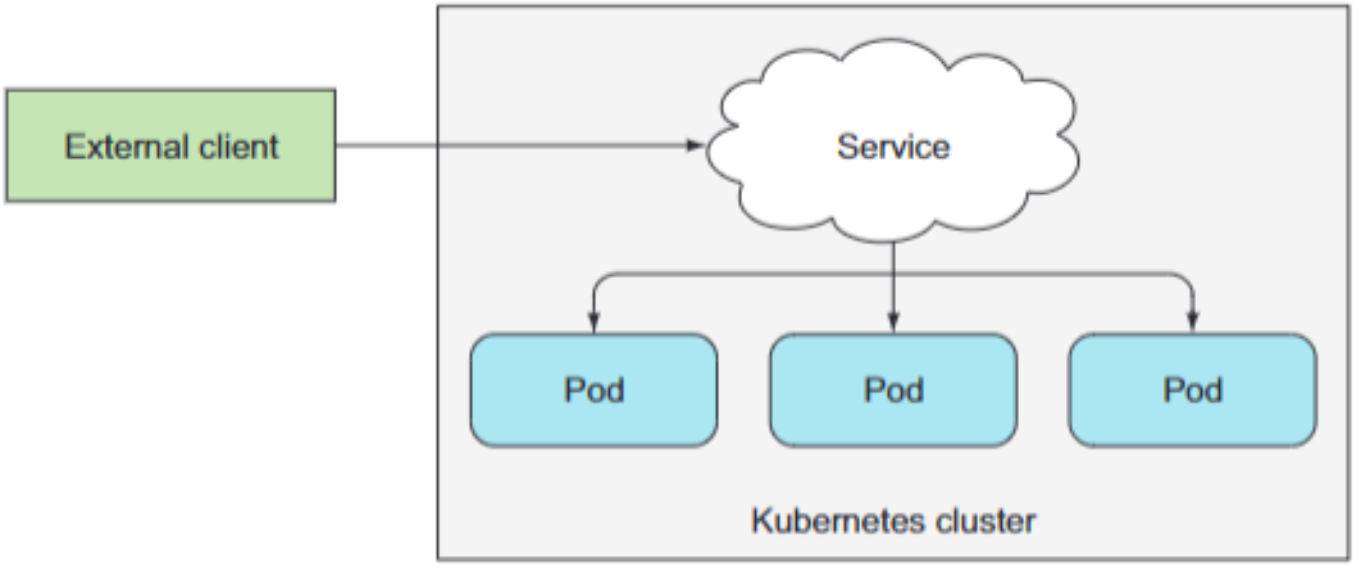


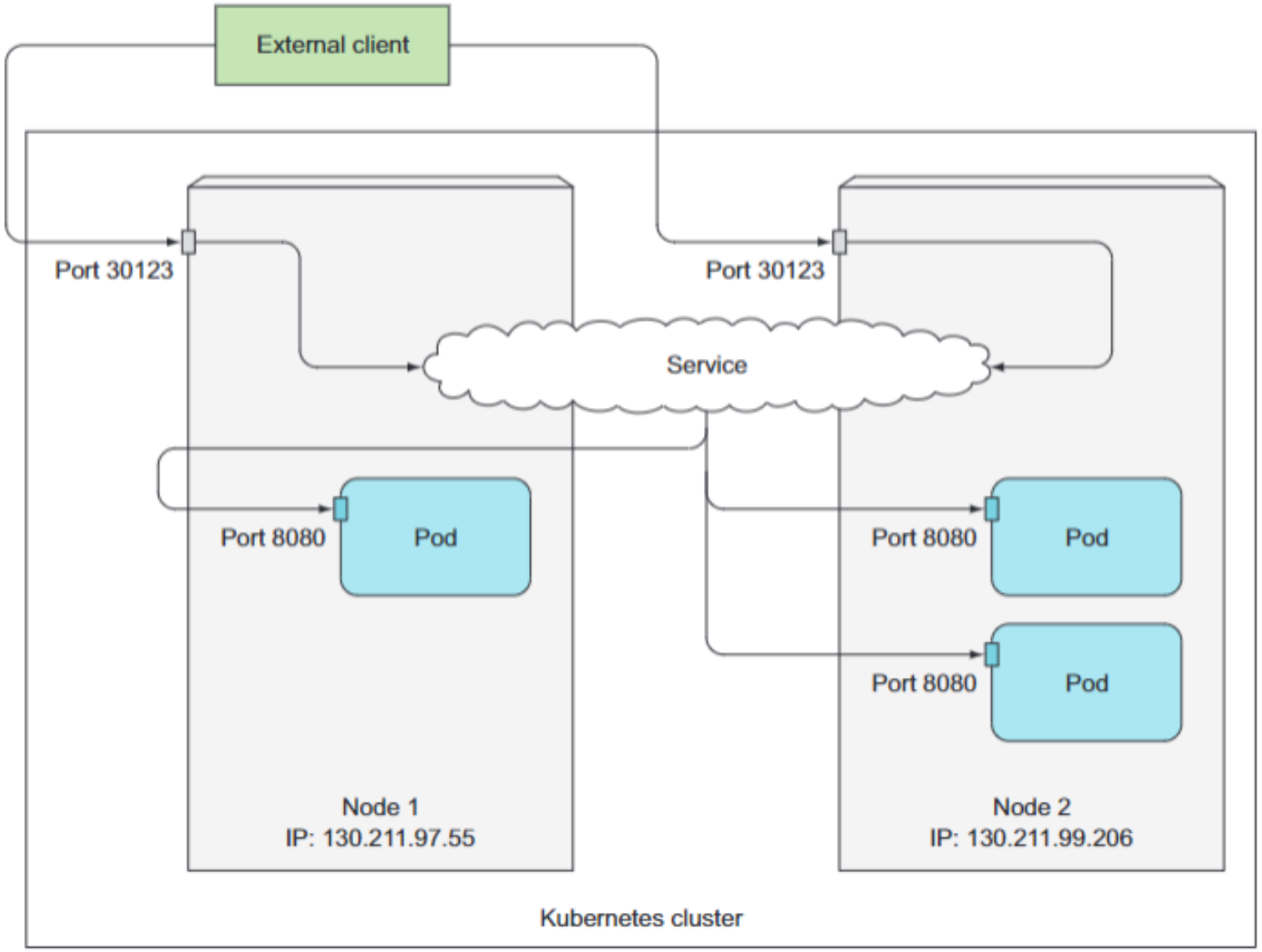
Figure: Fig. by courtesy of Marko Luksa[1]

Using a NodePort service

Allows external traffic to our service

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

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.

Allows external traffic to our service (Internet)

```
$ 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
```

Using a LoadBalancer service cont.

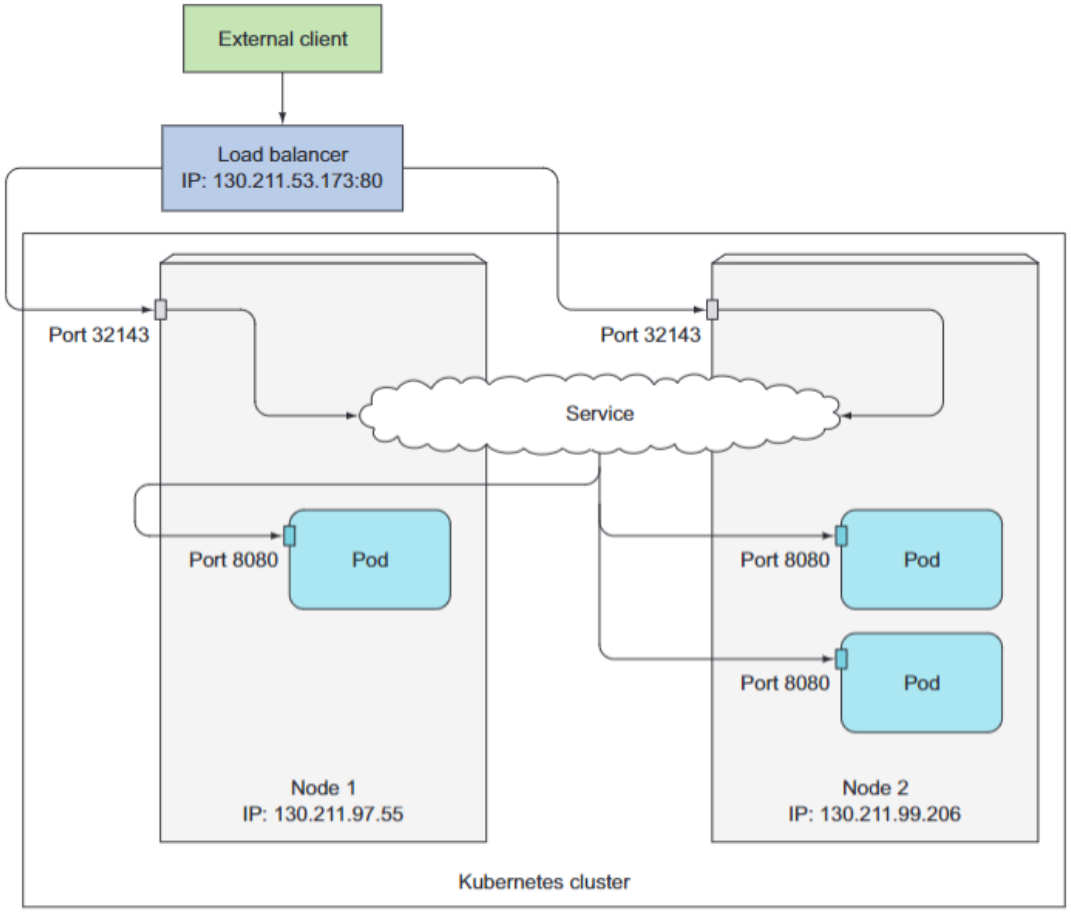


Figure: Fig. by courtesy of Marko Luksa[1]

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/v1beta1
kind: Deployment
metadata:
  name: kubia
spec:
  replicas: 3
  template:
    metadata:
      name: kubia
      labels:
        app: kubia
    spec:
      containers:
      - image: luksa/kubia:v1
        name: nodejs
```

Deployments are in the apps API group, version v1beta1.

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
```

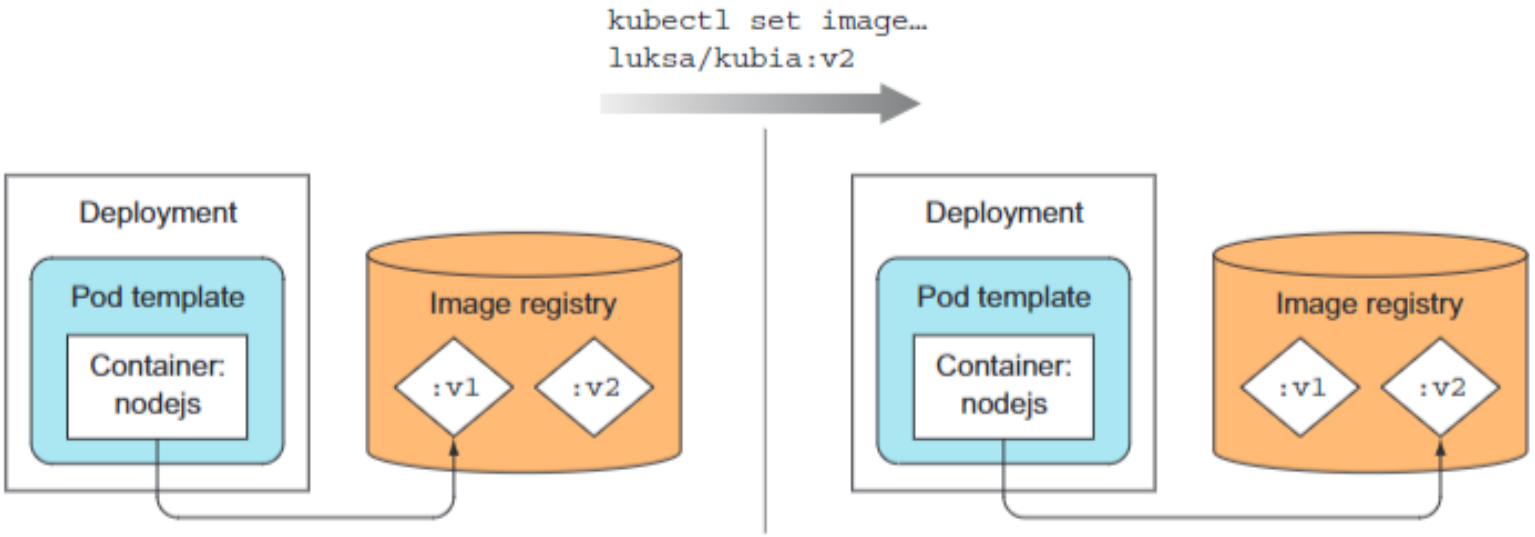


Figure: Fig. by courtesy of Marko Luksa[1]

Rolling out updates cont.

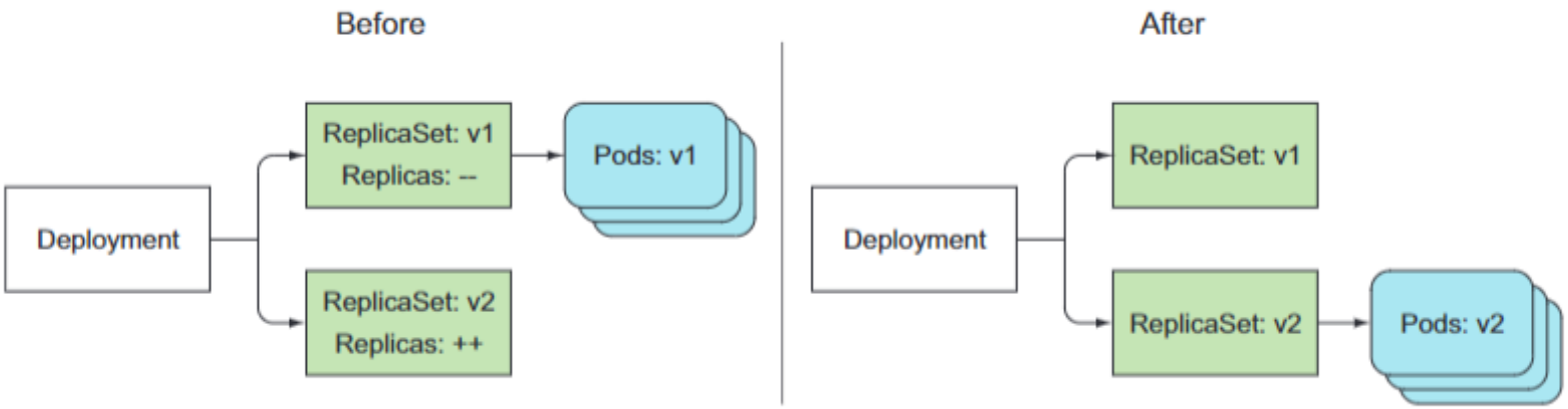


Figure: Fig. by courtesy of Marko Luksa[1]

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
```

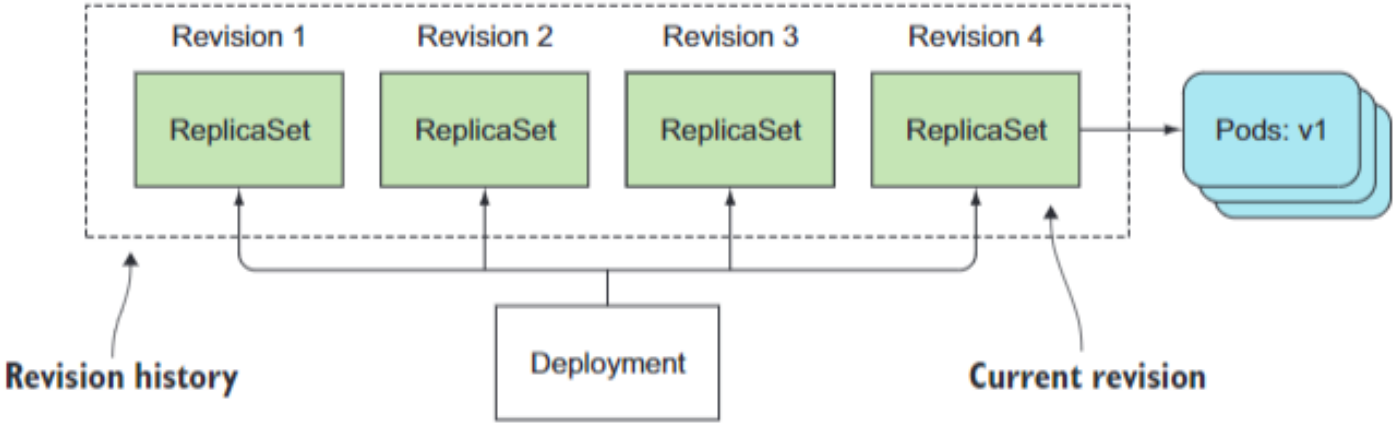


Figure: Fig. by courtesy of Marko Luksa[1]

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.

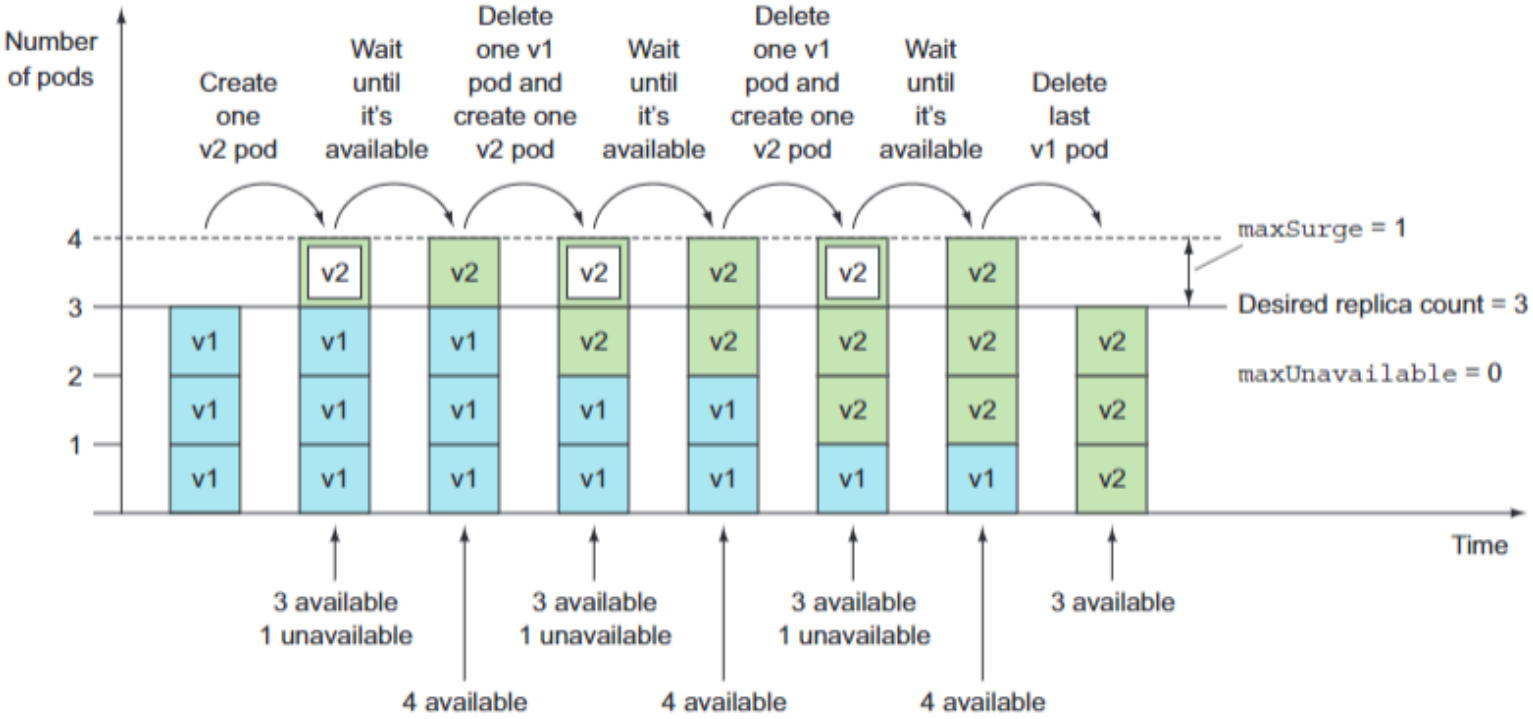


Figure: Fig. by courtesy of Marko Luksa[1]

Control the rate of the rollout cont.

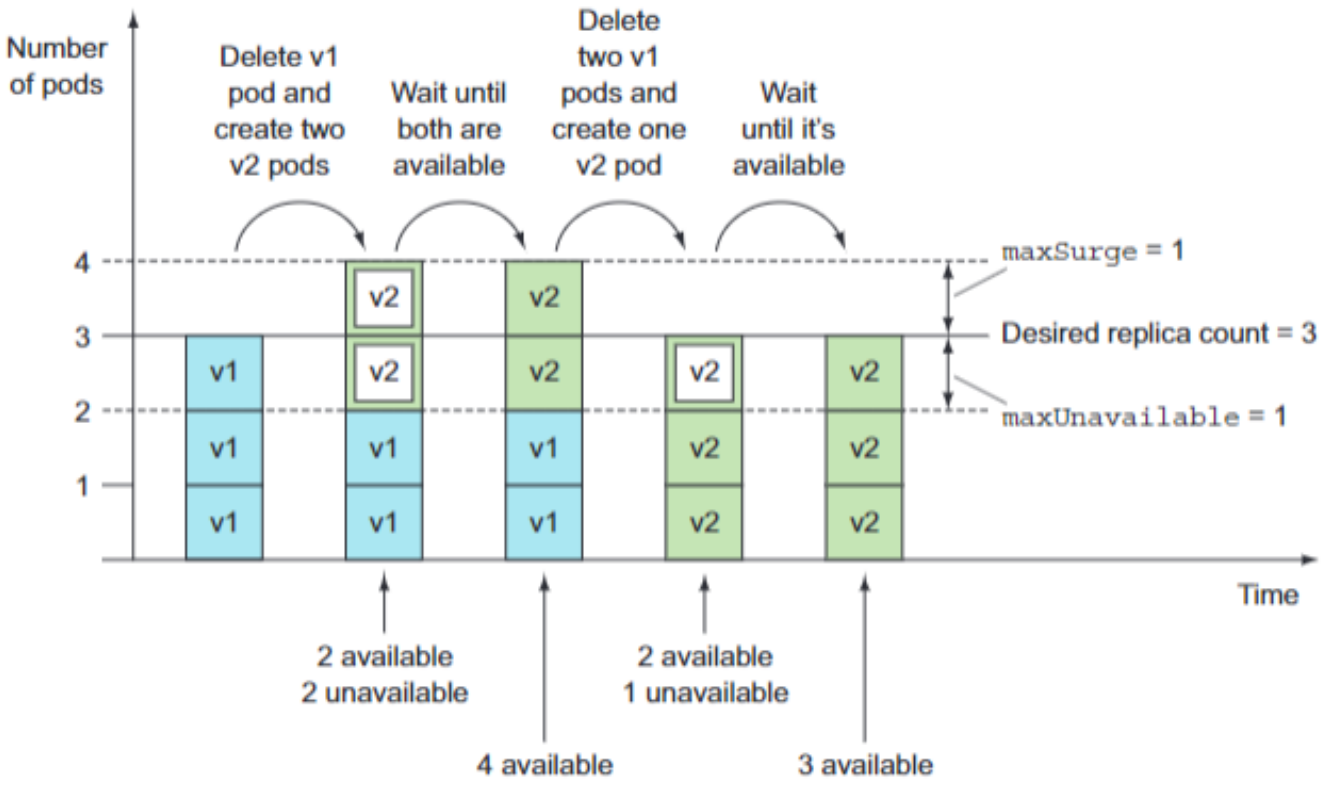


Figure: Fig. by courtesy of Marko Luksa[1]

References I

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