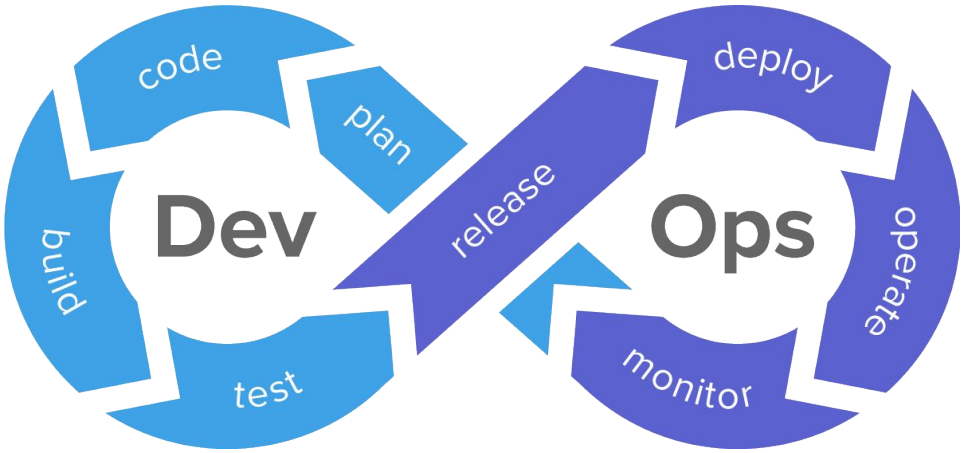


Kubernetes

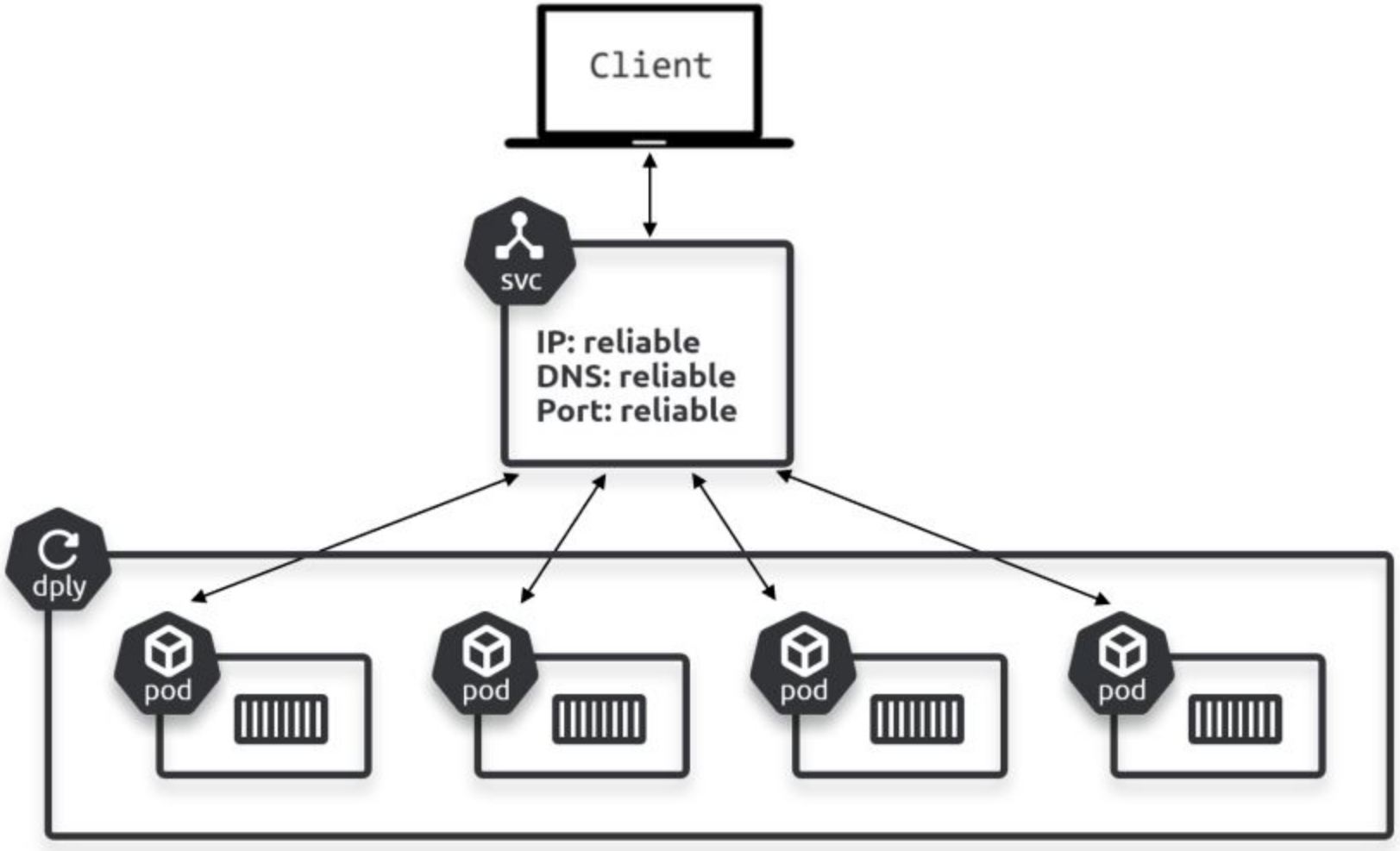
Assoc. Prof. Panche Ribarski, PhD | Assoc. Prof. Milos Jovanovik, PhD

Continuous Integration and Delivery 2023

Services

- Pods get updated, destroyed or just plain fail
- Pods are immutable
  - New Pods will be created in their place
  - That means new IP
- Pod DNS is tied to the unique Pod name
  - who would keep a list of running Pods? Not us
- Enter Services
  - stable IP
  - stable DNS
  - stable ports

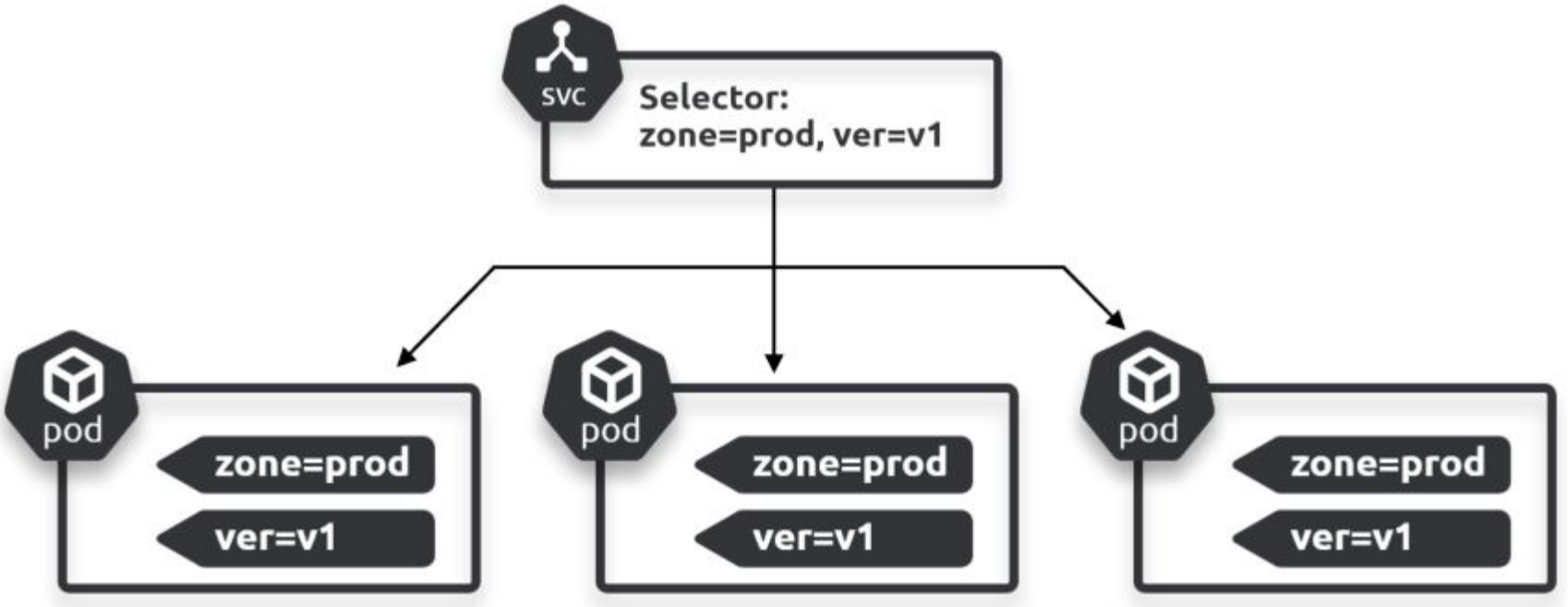
Service object in Kubernetes



Service benefits

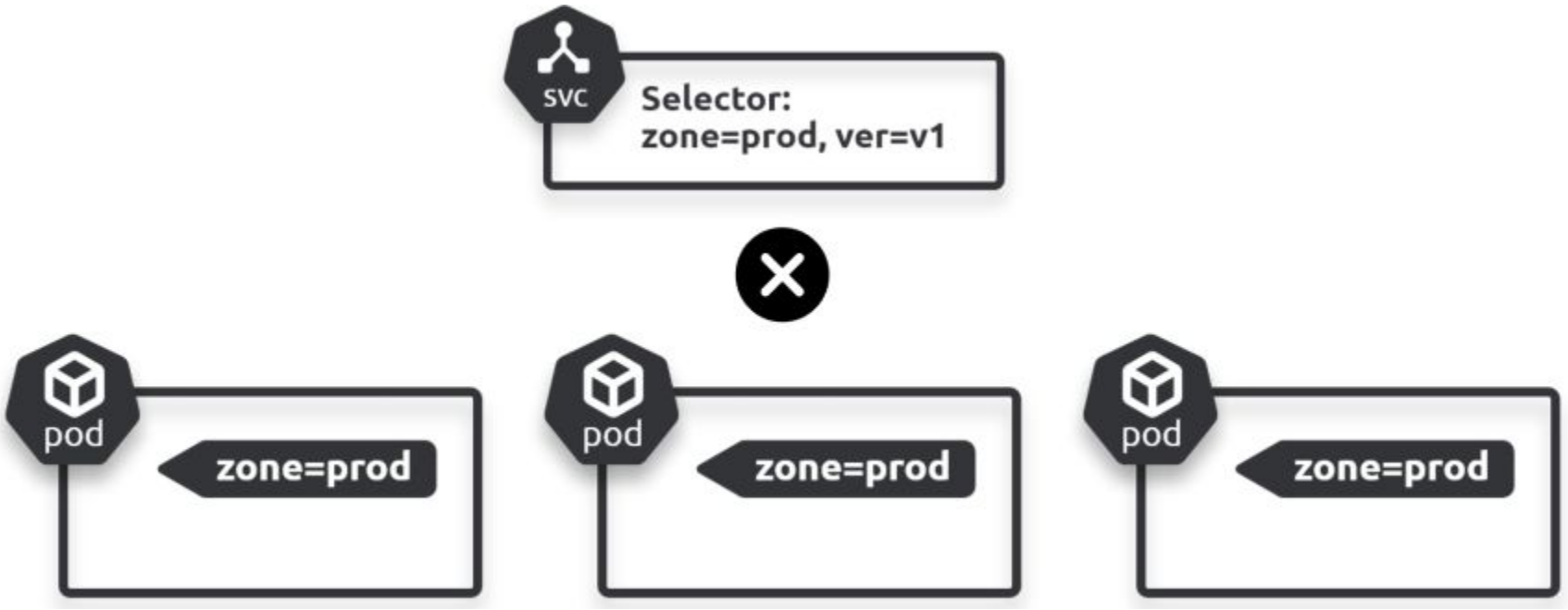
- Deployments can overlook Pods
  - scale
  - update (Rollout)
  - revert (Rollback)
- Everyone else need a stable handle that links to those Pods from the Deployment
- Create a Service object
  - will point to all the Pods (even LB them)
  - keep a list of (healthy) Pods
    - and their IP’s
  - index Pods via
    - labels
    - selectors

Service coupling to Pods



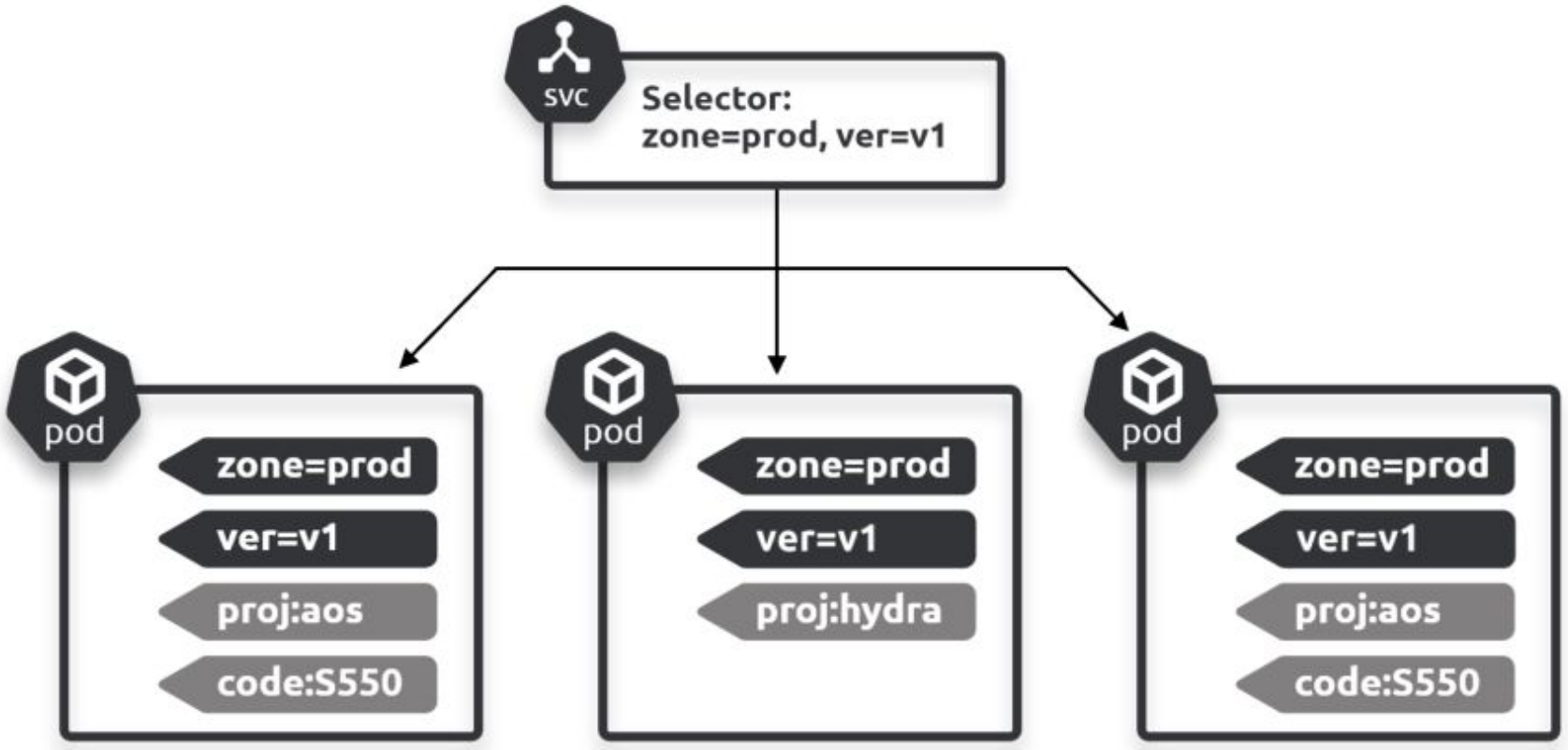
Service selectors

- Service will look for Pods with the listed selectors:
  - all listed tags in selectors must be labels in the Pods



Service selectors

- Your Pods can have extra labels



Our first Service

```
apiVersion: v1
kind: Service
metadata:
  name: hello-svc
spec:
  ports:
  - port: 8080
  selector:
    app: hello-world          <<==== Send to Pods with these labels
    env: tkb                  <<==== Send to Pods with these labels
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello-deploy
spec:
  replicas: 10
  <Snip>
  template:
    metadata:
      labels:
        app: hello-world      <<====  Pod labels
        env: tkb              <<====  Pod labels
    spec:
      containers:
```


EndpointSlice object

- created by the Service object
- Kubernetes curates EndpointSlice lists
  - constantly watch Services with their selectors and Pods with their labels
  - all healthy Pods matching selectors are added to the Service's EndpointSlice object
- When we access the Service IP, our request gets routed through the IP's from the EndpointSlice
- You can query KubeAPI and directly get EndpointSlice content
  - in a Microservice worlds, this is called Service Discovery

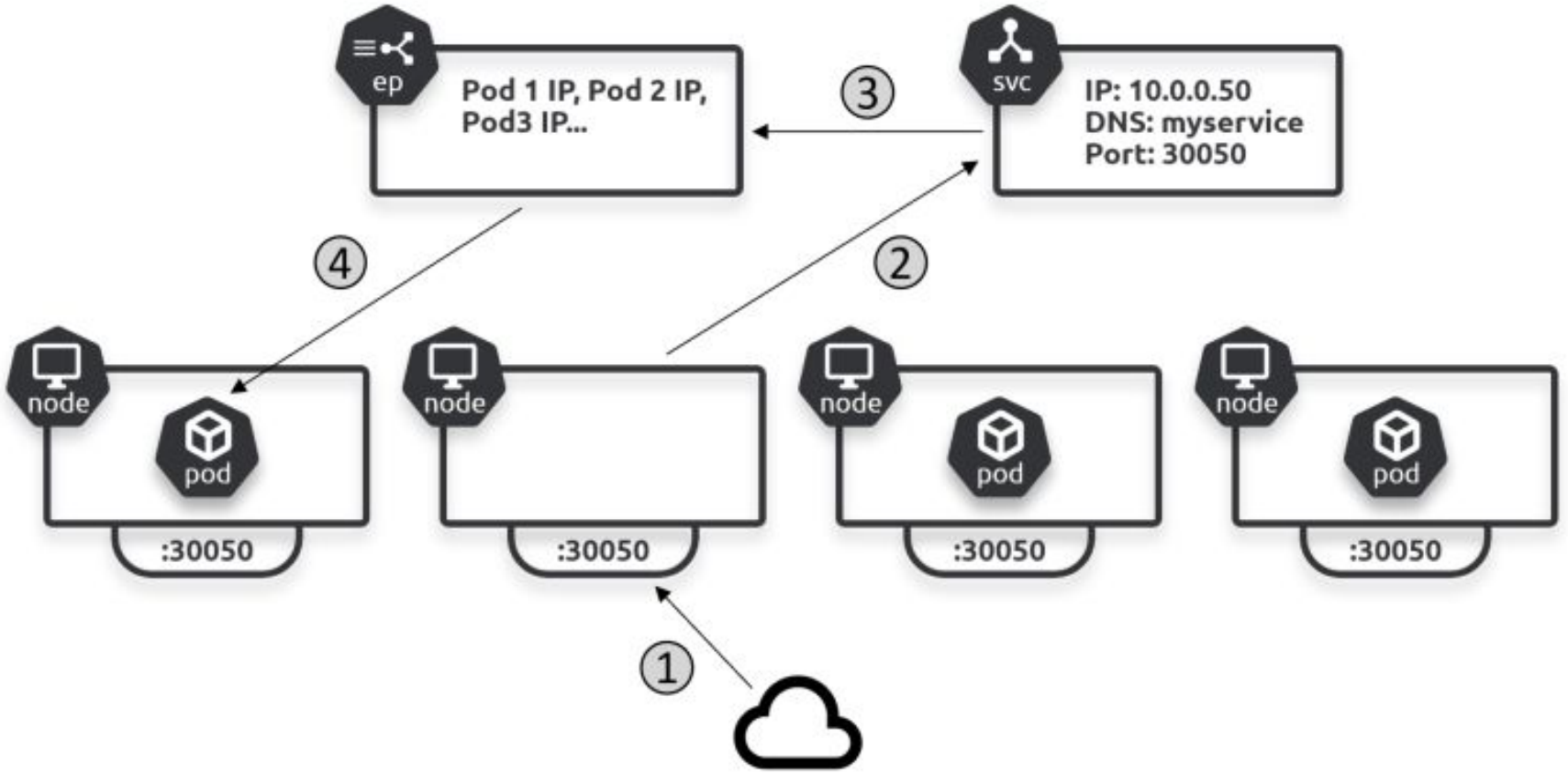
Access a Service from inside

- Service type: ClusterIP
  - our Service gets an IP from the internal K8S network overlay
  - this means our Service can be accessed only from inside
- DNS
  - Services get assigned A and AAAA records:
    - my-svc.my-namespace.svc.cluster-domain.example
  - https://kubernetes.io/docs/concepts/services-networking/dns-pod-service/#services
- Stable
  - ClusterIP type will generate
    - stable IP
    - stable DNS

Access a Service from the outside

- We can access Kubernetes objects from the outside in two ways:
  - NodePort
  - LoadBalancer
- NodePort builds on top of ClusterIP, opening the same port on **every** worker node and routing this port to the designated Service
  - You're guessing right, only one deployed Service can be bound to a given port
  - Not scalable
  - Not usable on production clusters
    - with some exceptions, if you know what you're doing

NodePort access



Before we create Services

- There are several ways to get access to the cluster from the inside
  - Ingress (later on)
  - NodePort (now)
- Let’s create K3D with NodePort exposed
  - destroy old one
    - k3d cluster delete kube3
  - create new one
    - k3d cluster create kube4 -p "30001:30001@agent:0" -s 1 -a 1
    - Creates port mapping to the worker and inside
      - https://k3d.io/v5.4.9/usage/exposing_services/?h=nodeport#2-via-nodeport

Create a Service the imperative way

- First of all - not the right way
- Go the service folder
- Deploy a Deployment
  - kubectl apply -f deploy.yml
- Create an imperative Service with NodePort access
  - kubectl expose deployment svc-test --type=NodePort
- Get Service info
  - kubectl describe service svc-test

Service description

```
$ kubectl describe svc svc-test
Name:                     svc-test
Namespace:                default
Labels:                   <none>
Annotations:              <none>
Selector:                 chapter=services
Type:                     NodePort
IP Family Policy:         SingleStack
IP Families:              IPv4
IP:                       10.43.56.24
IPs:                      10.43.56.24
Port:                     <unset>  8080/TCP
TargetPort:               8080/TCP
NodePort:                 <unset>  30013/TCP
Endpoints:                10.42.0.19:8080,10.42.0.20:8080,10.42.0.21:8080 + 7 more...
Session Affinity:         None
External Traffic Policy:  Cluster
Events:                   <none>
```


Thoughts

- We can’t reach this Service
  - port 30013 was not mapped
- Not the right way (imperative)
- Let’s destroy it
  -

Deploy a Service, the declarative way because This is the way

```yaml
apiVersion: v1
kind: Service
metadata:
  name: svc-test
spec:
  type: NodePort
  ports:
  - port: 8080
    nodePort: 30001
    targetPort: 8080
    protocol: TCP
  selector:
    chapter: services
```

Get Services, describe ours and access it

- Get Services
  - $ kubectl get service
  - NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE
  - kubernetes ClusterIP 10.43.0.1 <none> 443/TCP 15m
  - svc-test NodePort 10.43.188.110 <none> 8080:30001/TCP 38s
- Describe our Service
  - $ kubectl describe service svc-test
  - Name: svc-test
  - Namespace: default
  - Labels: chapter=services
  - Annotations: <none>
  - Selector: chapter=services
  - Type: NodePort
  - IP Family Policy: SingleStack
  - IP Families: IPv4
  - IP: 10.43.188.110
  - IPs: 10.43.188.110
  - Port: <unset> 8080/TCP
  - TargetPort: 8080/TCP
  - NodePort: <unset> 30001/TCP
  - Endpoints: 10.42.0.10:8080,10.42.0.6:8080,10.42.0.7:8080 + 7 more...
  - Session Affinity: None
  - External Traffic Policy: Cluster
  - Events: <none>

EndpointSlices

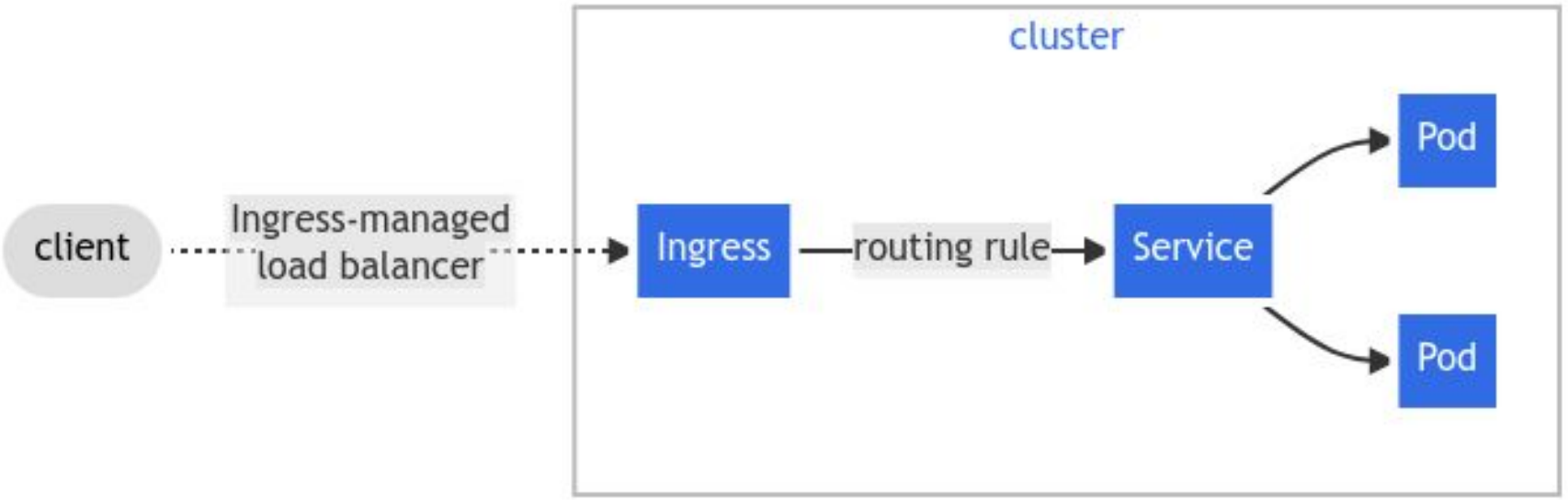
```
$ kubectl get endpointslices
NAME             ADDRESSTYPE   PORTS   ENDPOINTS                                   AGE
svc-test-n7jg4   IPv4          8080    10.42.1.16,10.42.1.17,10.42.0.19  + 7 more...   2m1s
svc-test-9s6sq   IPv6          8080    fd00:10:244:1::c,fd00:10:244:1::9 + 7 more...   2m1s

$ kubectl describe endpointslice svc-test-n7jg4
Name:         svc-test-n7jg4
Namespace:    default
Labels:       chapter=services
              endpointslice.kubernetes.io/managed-by=endpointslice-controller.k8s.io
              kubernetes.io/service-name=svc-test
Annotations:  endpoints.kubernetes.io/last-change-trigger-time: 2022-01-10T16:20:46Z
AddressType:  IPv4
Ports:
  Name     Port  Protocol
  ----     ----  --------
  <unset>  8080  TCP
Endpoints:
  - Addresses:  10.42.1.16
    Conditions:
      Ready:    true
    Hostname:   <unset>
    TargetRef:  Pod/svc-test-9d7b4cf9d-hnvbf
    NodeName:   k3d-tkb-agent-2
    Zone:       <unset>
  - Addresses:  10.42.1.17
<Snip>
Events:         <none>
```


LoadBalancer Service type

```yaml
apiVersion: v1
kind: Service
metadata:
  name: cloud-lb
spec:
  type: LoadBalancer
  ports:
  - port: 9000
    targetPort: 8080
  selector:
    chapter: services
```


Ingress

- Ingress exposes HTTP and HTTPS routes from outside the cluster to services within the cluster
- Traffic routing is controlled by rules defined on the Ingress resource
- https://kubernetes.io/docs/concepts/services-networking/ingress/



Ingress Controllers

- Mostly used:
  - nginx
  - traefik
- Many many more
  - https://kubernetes.io/docs/concepts/services-networking/ingress-controllers/
- You can have more than one Ingress Controller on your cluster
  - control which one is used by a particular ingress object by spec.ingressClassName in the ingress manifest

K3D Ingress Controller

- K3D implements K3S in docker
- K3S has traefik as Ingress Controller by default on a Load Balancer node (service of LoadBalancer type)
  - $ kubectl get deployment traefik --namespace kube-system -o yaml
  - $ kubectl get svc traefik --namespace kube-system -o yaml
- You need to expose port 80 on the LB worker
  - destroy current cluster
    - k3d cluster delete k3d
  - create new one with port forwarding to traefik
    - k3d cluster create kube4 -p "80:80@loadbalancer" -s 1 -a 1

Ingress Object

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: minimal-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  ingressClassName: nginx-example
  rules:
  - http:
      paths:
      - path: /testpath
        pathType: Prefix
        backend:
          service:
            name: test
            port:
              number: 80
```

Deploy example app

- Go to ingress folder
- kubectl apply -f app.yml

Simple traefik path example

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: mcu-all
  annotations:
    ingress.kubernetes.io/ssl-redirect: "false"
spec:
  rules:
  - http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: svc-shield
            port:
              number: 8080
```

Complex traefik example

- traefik does not support rewrite target
  - it needs special middleware for these rules
- see k3ing-complex.yml file from the course

Complex traefik - middleware

```
apiVersion: traefik.containo.us/v1alpha1
kind: Middleware
metadata:
  name: mcu-all
spec:
  stripPrefix:
    forceSlash: false
    prefixes:
      - /shield
      - /hydra
---
```

Complex traefik - path example

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: mcu-all
  annotations:
    ingress.kubernetes.io/ssl-redirect: "false"
    traefik.ingress.kubernetes.io/router.middlewares: default-mcu-all@kubernetescrd
spec:
  rules:
  - http:
      paths:
      - path: /shield
        pathType: Prefix
        backend:
          service:
            name: svc-shield
            port:
              number: 8080
      - path: /hydra
        pathType: Prefix
        backend:
          service:
            name: svc-hydra
            port:
              number: 8080
```


Complex traefik - host example

```
- host: shield.mcu.com
  http:
    paths:
    - path: /
      pathType: Prefix
      backend:
        service:
          name: svc-shield
          port:
            number: 8080
- host: hydra.mcu.com
  http:
    paths:
    - path: /
      pathType: Prefix
      backend:
        service:
          name: svc-hydra
          port:
            number: 8080
```

- don’t forget to edit your hosts file:
  - 127.0.0.1 shield.mcu.com hydra.mcu.com

Inspecting ingress objects

```
$ kubectl get ing
NAME       CLASS     HOSTS                       ADDRESS                    PORTS   AGE
mcu-all    traefik   shield.mcu.com,hydra.mcu.com   192.168.64.2,192.168.64.3   80   9m44s
```

Describe an ingress object

```
Name:             mcu-all
Labels:           <none>
Namespace:        default
Address:          192.168.64.2,192.168.64.3
Ingress Class:    traefik
Default backend:  <default>
Rules:
  Host            Path  Backends
  ----            ----  --------
  *
                  /shield   svc-shield:8080 (10.42.1.6:8080)
                  /hydra    svc-hydra:8080 (10.42.1.5:8080)
  shield.mcu.com
                  /   svc-shield:8080 (10.42.1.6:8080)
  hydra.mcu.com
                  /   svc-hydra:8080 (10.42.1.5:8080)
Annotations:      ingress.kubernetes.io/ssl-redirect: false
                  traefik.ingress.kubernetes.io/router.middlewares: default-mcu-all@kubernetescrd
Events:           <none>
```

Clean up your cluster

- k3d cluster delete kube4

Questions?

Reading materials

- The Kubernetes Book (2023 edition)
  - Nigel Poulton
  - Pushkar Joglekar
- Chapter 7: Kubernetes Services
- Chapter 8: Ingress

Class Assignment / Homework

- Create deployments and services for two Pods:
  - One for the app version 1.0 from the last homework
  - One for the app version 2.0 from the last homework
- Create ingress pointing to the two apps:
  - path based: localhost/ver1
  - path based: localhost/ver2
  - host based: ver1.<index>.com
  - host based: ver2.<index>.com
- Deploy manifests and ingress
- Access the four ingress rules in your local browser