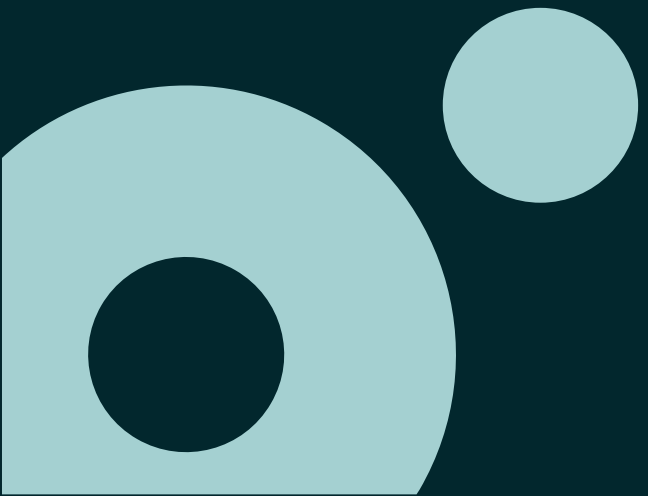
Containers & Kubernetes

Session #07



Services

ClusterIP

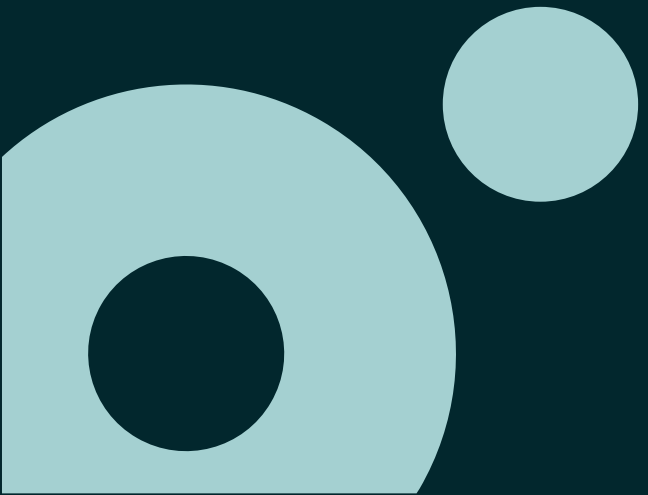
NodePort

LoadBalancer

Ingress & IngressController

Lab

Services



Motivation

Services

- Kubernetes Pods are created and destroyed to match the state of your cluster making them ephemeral
- Each Pod gets its own IP address, however in a Deployment, the set of Pods running in one moment in time could be different from the set of Pods running that application a moment later
- If some set of Pods provides functionality to other Pods inside your cluster, how do they find out and keep track of which IP address to connect to?

What is a Service?

Services

- An abstraction that defines a logical set of loosely-coupled pods and a policy by which to access them as a network service.
  - Use **selectors** to define which pods to include.
  - Load balances traffic to Pods (Layer 4)
  - Exposes Pods to other Pods within the cluster
  - Maps external ports to target (container) ports
- Kubernetes **automatically updates** which Pods are available to which service by creating Endpoints objects.
- 3 types
  - ClusterIP
  - NodePort
  - LoadBalancer

Service Manifest

Service

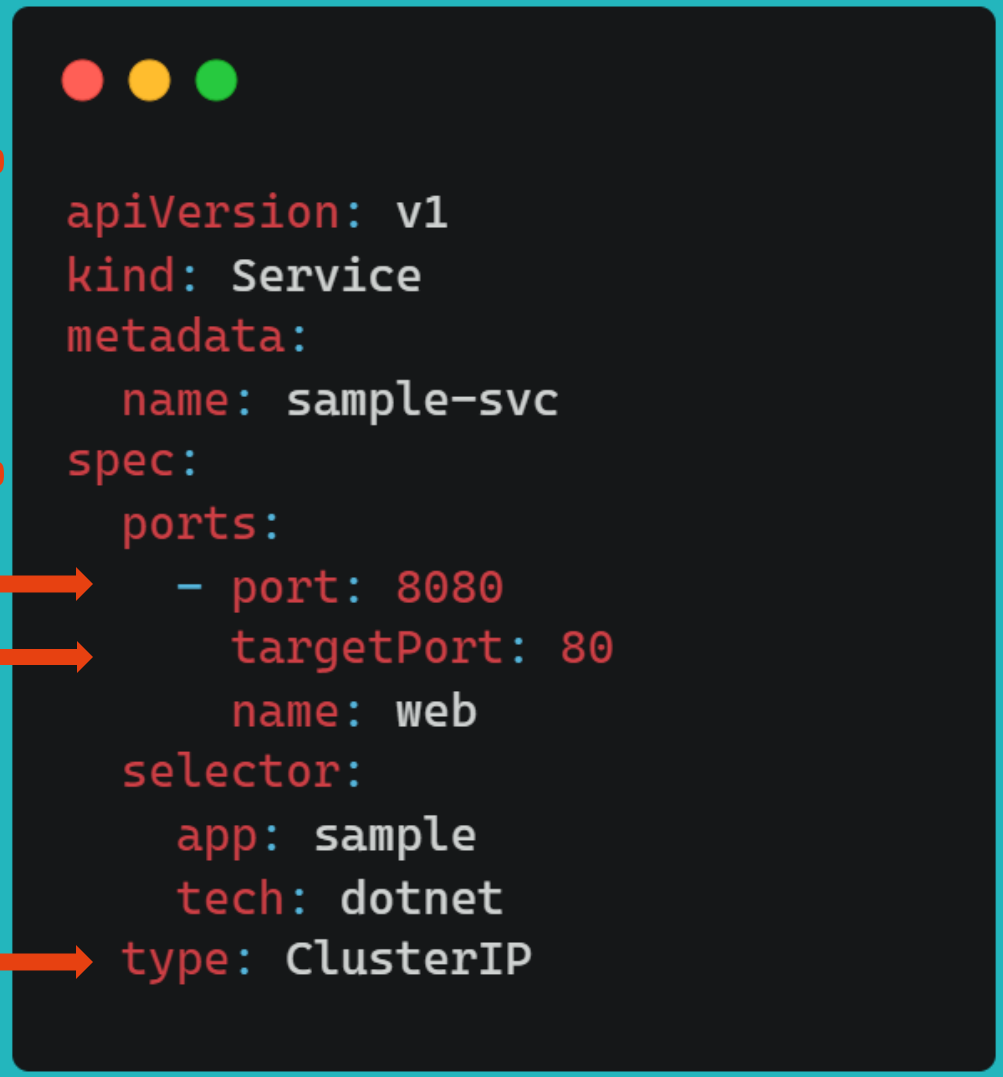
Service properties

port sets service port

targetPort refers pods port

selector defines pods selector

type defines service type



Endpoint Object

Service

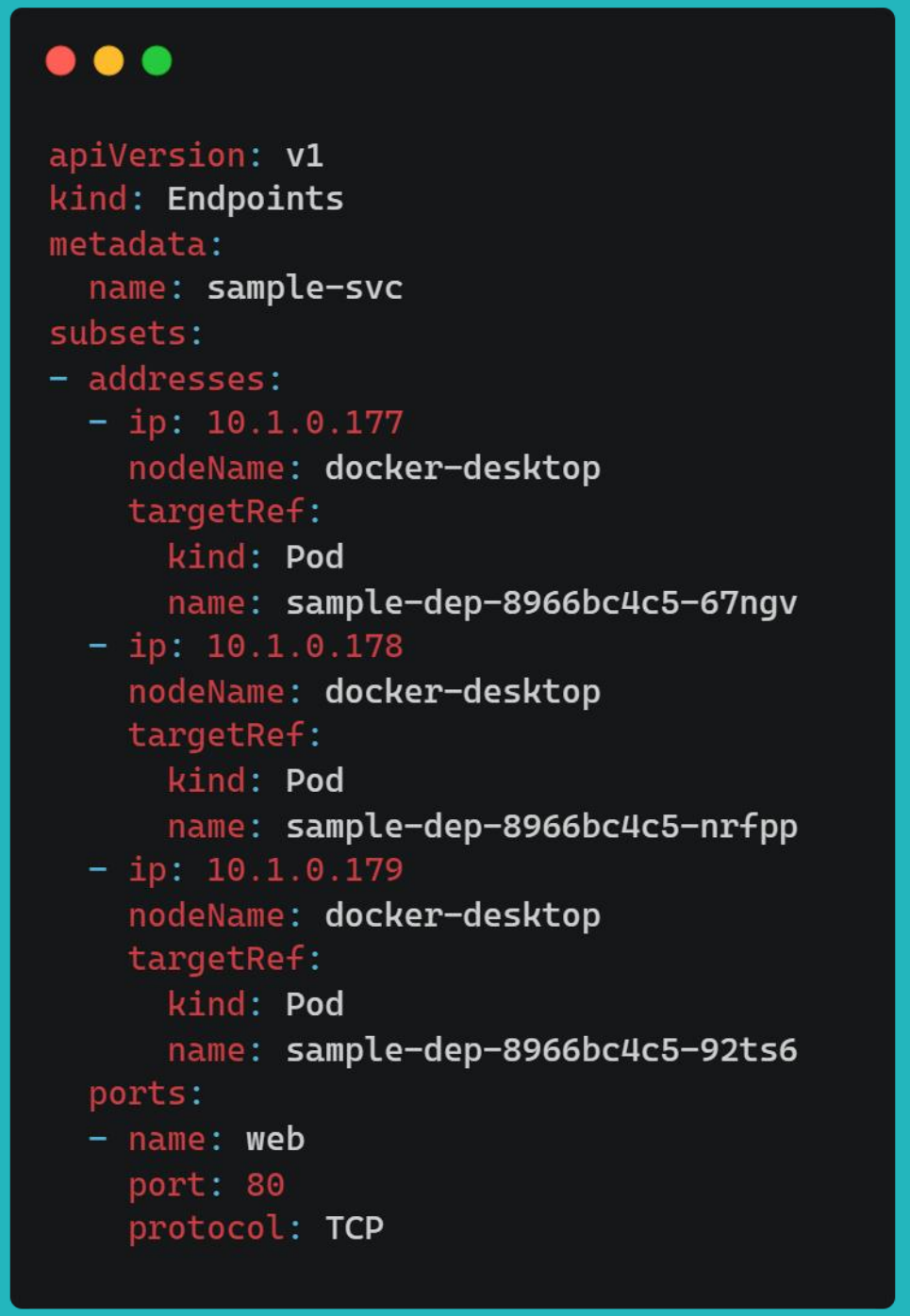
Endpoint properties

Each block defines pod settings

Each block defines pod settings

Each block defines pod settings

ports defines how to connect to pods



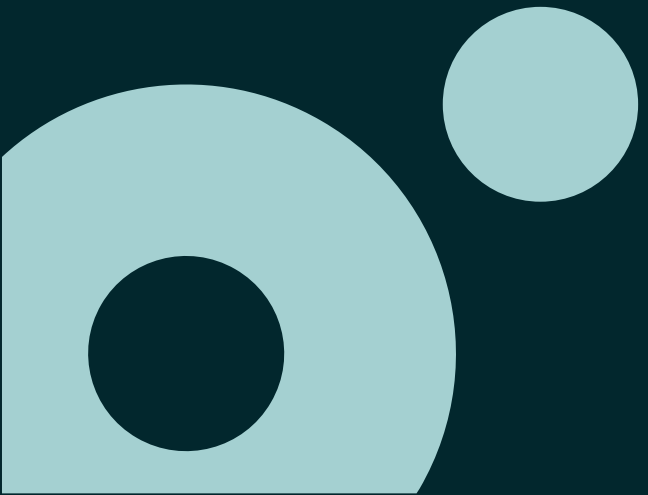
```yaml
apiVersion: v1
kind: Endpoints
metadata:
  name: sample-svc
subsets:
- addresses:
  - ip: 10.1.0.177
    nodeName: docker-desktop
    targetRef:
      kind: Pod
      name: sample-dep-8966bc4c5-67ngv
  - ip: 10.1.0.178
    nodeName: docker-desktop
    targetRef:
      kind: Pod
      name: sample-dep-8966bc4c5-nrfpp
  - ip: 10.1.0.179
    nodeName: docker-desktop
    targetRef:
      kind: Pod
      name: sample-dep-8966bc4c5-92ts6
  ports:
  - name: web
    port: 80
    protocol: TCP
```

Load Balancing

Services

- Depends on kube-proxy configuration mode
- User space proxy mode
  - Uses round-robin algorithm to select pods
- Iptables proxy mode (default)
  - Uses random selection of pods
- IPVS proxy mode
  - rr: round-robin
  - lc: least connection (smallest number of open connections)
  - dh: destination hashing
  - sh: source hashing
  - sed: shortest expected delay
  - nq: never queue

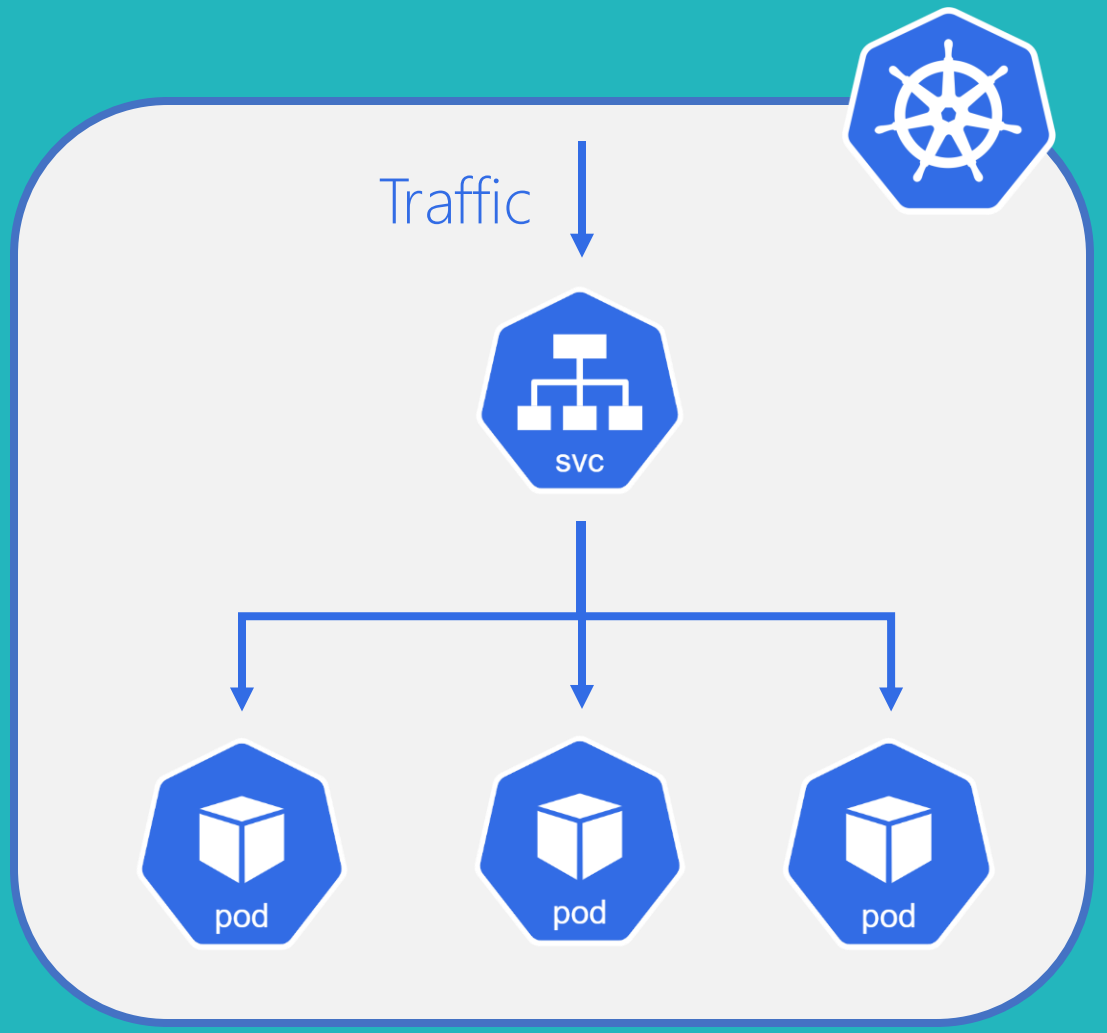
ClusterIP



ClusterIP

Services

- Exposes the Service on a cluster-internal IP
- Choosing this value makes the Service only reachable from within the cluster
- Default service type
- Uses service name as a DNS
  - http://svc-name
  - http://svc-name.ns.svc.cluster.local



Deployment

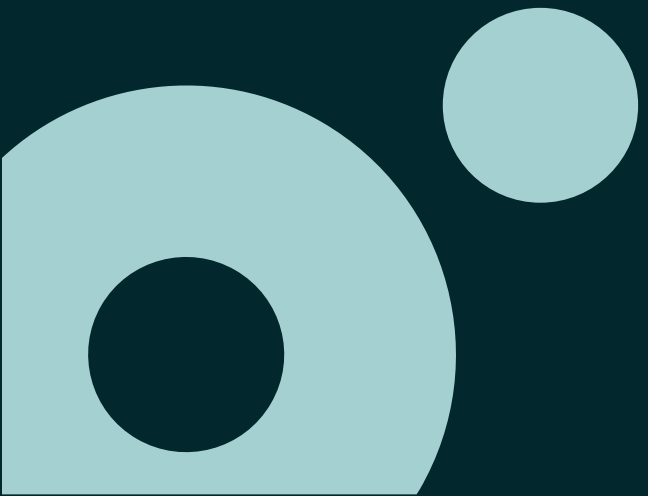
```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: sample-dep
spec:
  replicas: 3
  selector:
    matchLabels:
      app: sample
      tech: dotnet
  template:
    metadata:
      labels:
        app: sample
        tech: dotnet
    spec:
      containers:
      - name: sample
        image: mcr.microsoft.com/dotnet/samples:aspnetapp
        ports:
        - containerPort: 80
```

ClusterIP

```yaml
apiVersion: v1
kind: Service
metadata:
  name: sample-svc
spec:
  ports:
    - port: 8080
      targetPort: 80
      name: web
  selector:
    app: sample
    tech: dotnet
  type: ClusterIP
```

Demo: ClusterIP

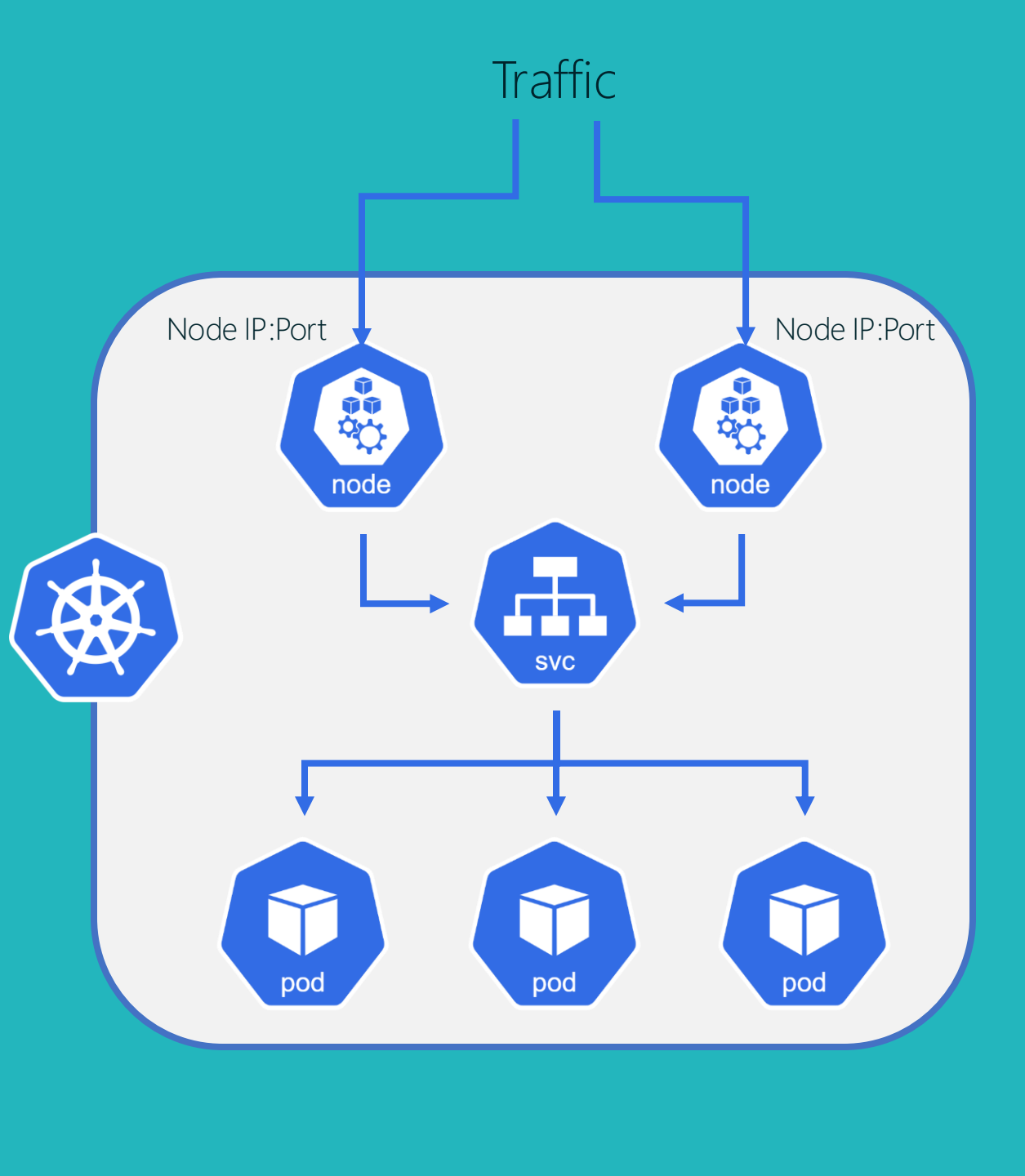
Node Port



NodePort

Services

- Exposes the Service on each Node's IP at a static port (the NodePort)
- A ClusterIP Service, to which the NodePort Service routes, is automatically created.
- You'll be able to contact the NodePort Service, from outside the cluster, by requesting **<NodeIP>:<NodePort>**
- If **nodePort** property is not specified, is automatically set an port from the range 30000-32767



NodePort

```yaml
apiVersion: v1
kind: Service
metadata:
  name: sample-svc
spec:
  ports:
    - port: 8080
      targetPort: 80
      nodePort: 30000
      name: web
  selector:
    app: sample
    tech: dotnet
  type: NodePort
```

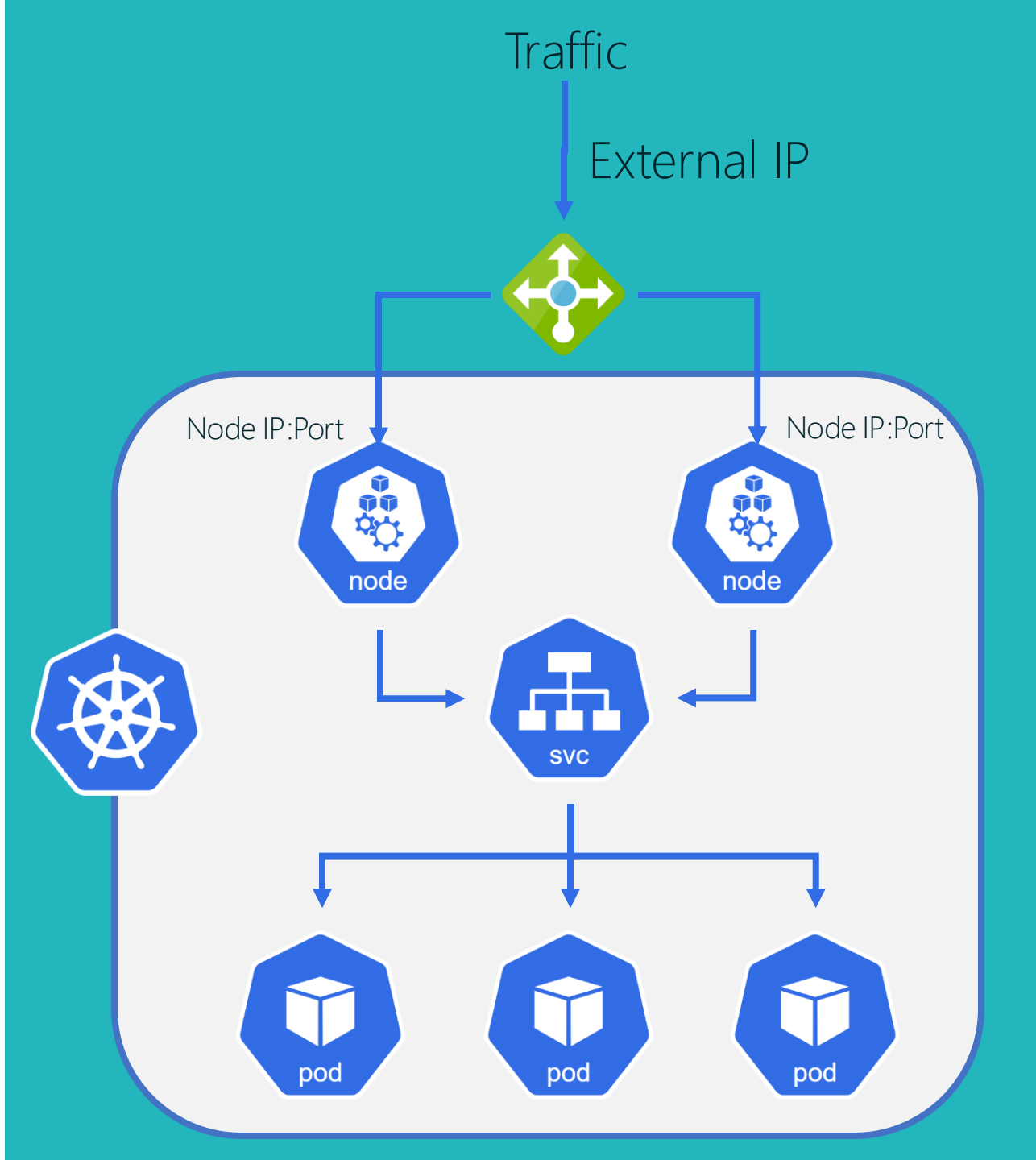
Demo: Node Port

Load Balancer

LoadBalancer

Services

- Exposes the Service externally using a cloud provider's load balancer
- NodePort and ClusterIP Services, to which the external load balancer routes, are automatically created
- On-prem needs manual configuration
- Can be internal (no public IP)



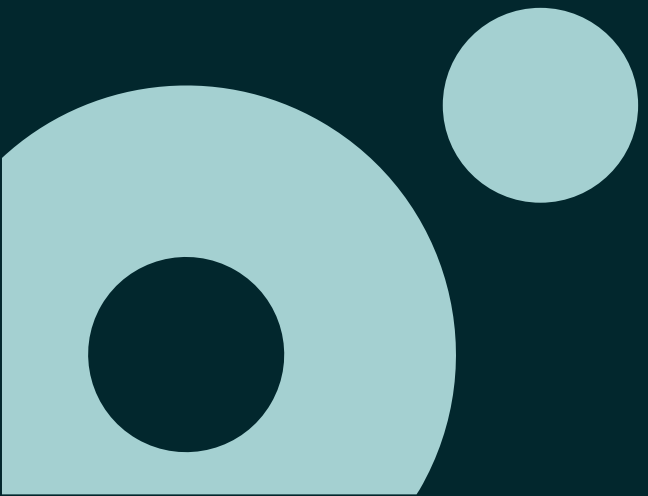
LoadBalancer

```yaml
apiVersion: v1
kind: Service
metadata:
  name: sample-svc
spec:
  ports:
    - port: 8080
      targetPort: 80
      nodePort: 30000
      name: web
  selector:
    app: sample
    tech: dotnet
  type: LoadBalancer
```

Demo: Load Balancer

Mao Ongy 2021

Ingress & Ingress Controller



Motivation

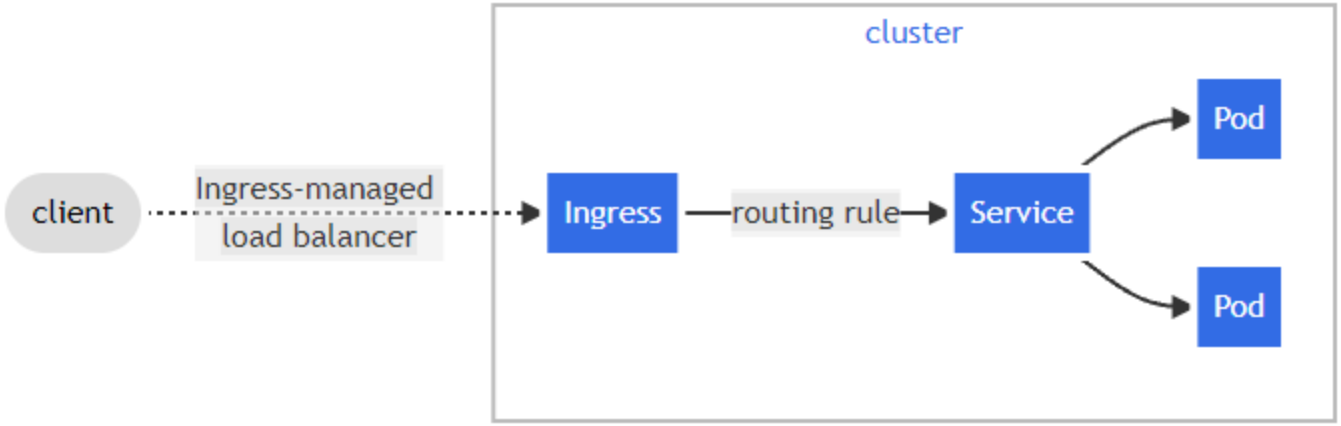
Ingress & Ingress Controller

- Creating several Load Balancer services will need several Public Ips to allow access to workloads
- Several issues on this approach
  - Several IPs to handle
  - Security with direct access to the cluster
  - SSL termination
- Needs to have a single ingress (inbound) connection from outside the cluster to inside services

Ingress

Ingress & Ingress Controller

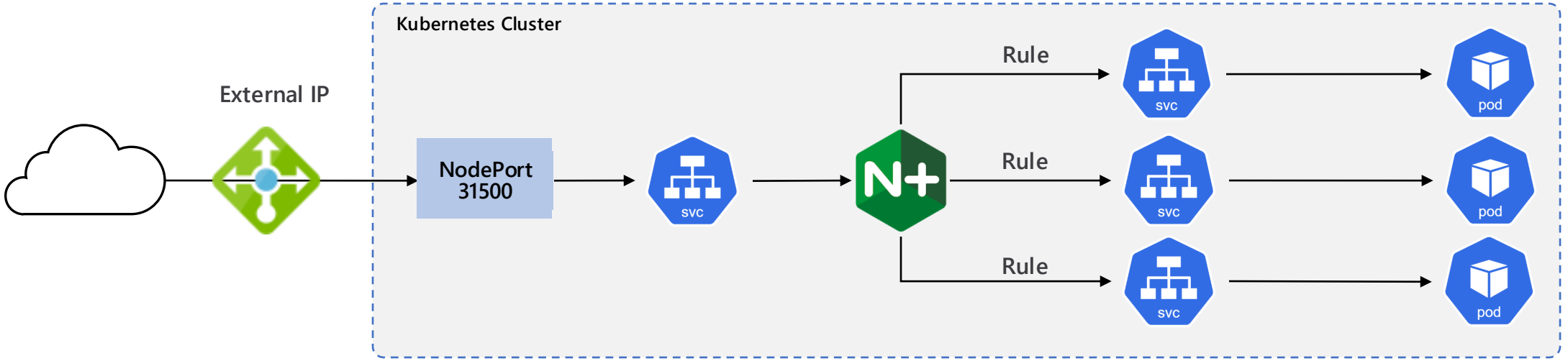
- Exposes HTTP and HTTPS routes from outside the cluster to services within the cluster
- Traffic routing is controlled by rules defined on the Ingress resource



Ingress Controller

Ingress & Ingress Controller

- Kubernetes don't have concrete implementation for this routing
- Ingress are used by Ingress Controllers to implement the routing
- Most used
  - The NGINX Ingress Controller for Kubernetes works with the NGINX webserver (as a proxy).
  - The Traefik Kubernetes Ingress provider is an ingress controller for the Traefik proxy.



Ingress

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: simple-fanout-example
spec:
  rules:
  - host: foo.bar.com
    http:
      paths:
      - path: /foo
        pathType: Prefix
        backend:
          service:
            name: service1
            port:
              number: 4200
      - path: /bar
        pathType: Prefix
        backend:
          service:
            name: service2
            port:
              number: 8080
```

Demo: Ingress