Services

a method for exposing a network application that is running as one or more Pods in your cluster.

NodePort	ClusterIP	Load Balancer
Expose a service on specific port of each node in cluster.	Expose a service to other service in cluster by cluster internal IP. (Not to outside world)	Expose service to outside world by a specific ip

```yaml
apiVersion: v1
kind: Service
metadata:
  name: myapp-service
spec:
  type: NodePort
  ports:
   - targetPort: 80
     port: 80
     nodePort: 30008
  selector:
    name: MyApp
```

name

Pod's Port

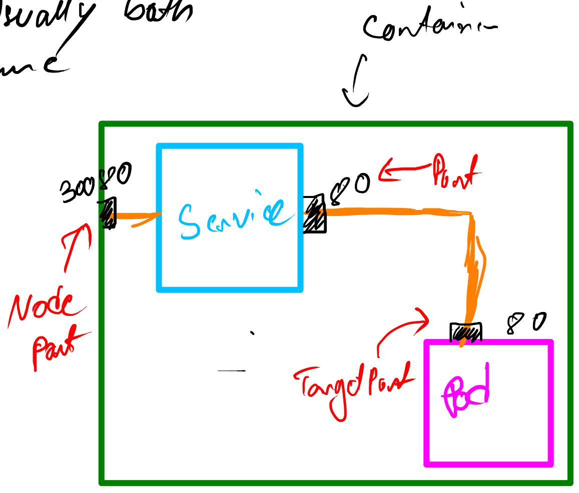
Pod connected to service to this port → Usually both same

Port that's exposed on Node

Selector for pods

→ Access service via

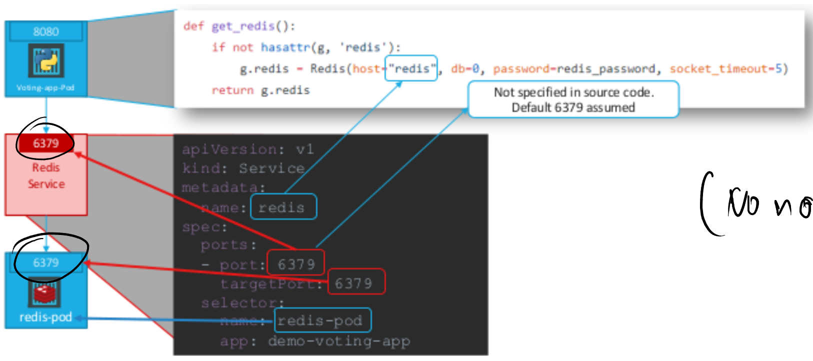
<node-ip> : <node-port>



ClusterIP → (Type : ClusterIP)

→ We can access the service by its name

Service →



```python
def get_redis():
    if not hasattr(g, 'redis'):
        g.redis = Redis(host="redis", db=0, password=redis_password, socket_timeout=5)
    return g.redis
```

```yaml
apiVersion: v1
kind: Service
metadata:
  name: redis
spec:
  ports:
  - port: 6379
    targetPort: 6379
  selector:
    name: redis-pod
    app: demo-voting-app
```

(No nodeport here!)

LoadBalancer

→ This mainly loadbalance between available resources

→ If there, is nodeport

- 10.0.0.1 : 80
- 10.0.0.2 : 80
- 10.0.0.3 : 80
- 10.0.0.4 : 80

→ Loadbalance (Access via single IP)

192.168.0.1 : 80

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    name: MyApp
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376
  clusterIP: 10.0.171.239
  type: LoadBalancer
status:
  loadBalancer:
    ingress:
    - ip: 192.0.2.127
```

→ NodePort not required, as it will create some random

→ Cluster IP

→ This handle exposing Cluster IP to world