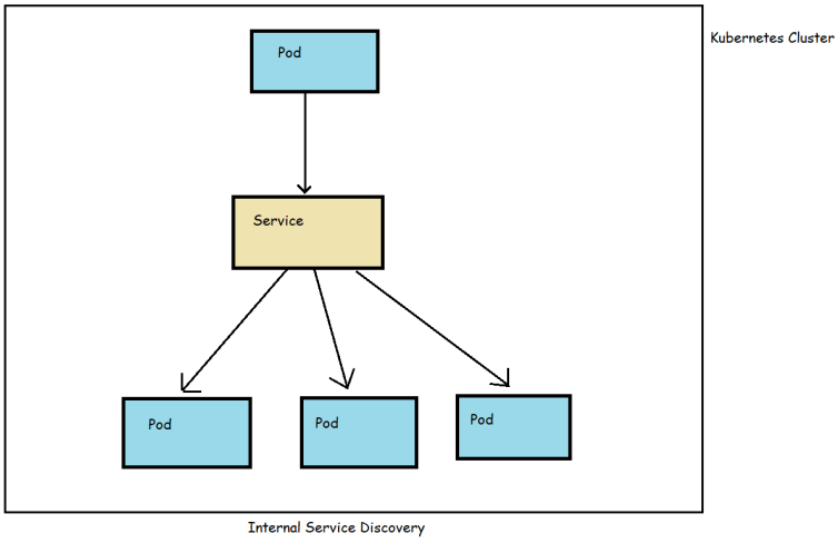
Service Discovery

- Overview: The Service Discovery Pattern provides a stable endpoint at which clients of service can access the instances providing the service. For this purpose, k8s provides multiple mechanisms, depending on whether the service consumers and producers are located on or off the cluster

Solution: Internal Service Discovery:

- Let's assume we have a web application in k8s so we create a Deployment with few replicas, the scheduler places the Pods on the suitable nodes and each Pod gets an Cluster-internal IP address assigned.
- If another client service within a different Pod wishes to consume the web application endpoint, there is no easy way to find pod ip address & To provide constant & stable entry point to collection of Pods offering same functionality we create a Kuberenetes Service.
- K8s service creates a virtual IP address referred as clusterIP and it pulls both Pod selectors and port numbers to create the Service Definition

```
---
apiVersion: v1
kind: Service
metadata:
  name: simple-svc
spec:
  selector:
    app: nginx
  ports:
    - port: 80
      targetPort: 80
      protocol: TCP
```



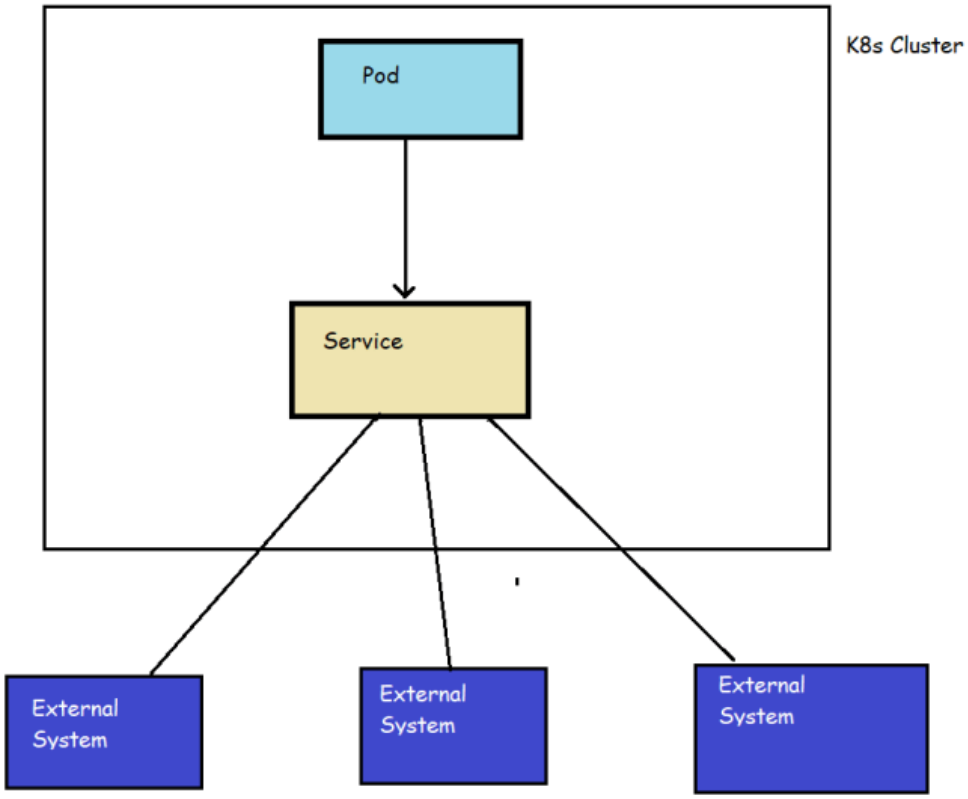
Internal Service Discovery

Discovery through environmental variables: ENV variables will be injected into the Pod while creating with all the Services that exist up to that moment. Problem is the Services created after Pod Creation will not be injected.

- SIMPLE_SVC_SERVICE_HOST=10.79.72.25
- SIMPLE_SVC_SERVICE_PORT=80

Discovery through DNS Lookup: If client is aware of name of the service it wants to access, it can reach by fqdn such as simple-svc.default.svc.cluster.local

Pod wants to access External Systems: for endpoint and cluster ip for external service



In addition to this to enable external access to the pods running in k8s cluster we have

- NodePort
- LoadBalancers

Structural Patterns

- The patterns in this category are focused on structuring and organizing containers in a Pod to satisfy different usecase.
    - Init Container introduces a separate lifecycle or initialization-related tasks and the main application containers
    - Sidecar: Describes how to extend and enhance the functionality of pre-existing container without changing it
    - Adapter: takes an heterogenous system and makes it confirm to consistent unified interface that can be consumed by outside world
    - Ambassador: Describes a proxy that decouples access to external services