***What is a service and its purpose? How does k8s implement a service?***

A service in Kubernetes is like a stable address for a group of pods. Its main purpose is to make it easy for other parts of your application to find and talk to these pods.

Imagine a restaurant with many chefs (pods) in the kitchen. The service is like the waiter who knows which chef is free and can take your order. You don't need to know which specific chef is cooking your meal.

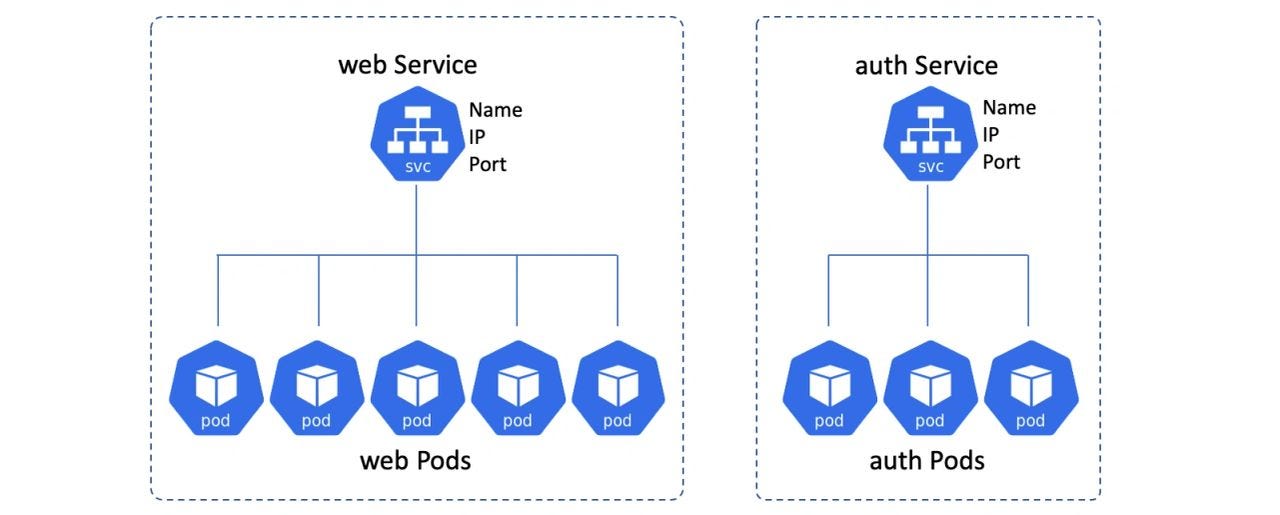
Kubernetes implements a service by:

Label selectors: Services use labels to identify which Pods to route traffic to.

Virtual IP (ClusterIP): Each Service gets a stable IP address within the cluster.

Service discovery: Kubernetes provides DNS names for Services, making it easy for other components to find and connect to them.

Load balancing: Traffic is automatically distributed among the Pods backing the Service.



***Explain PV and PVCs***

PV (Persistent Volume) is like a hard drive in the cloud that Kubernetes can use. It's a piece of storage in the cluster that has been set up by an admin or created automatically.

PVC (Persistent Volume Claim) is a request for storage by a user. It's like saying "I need a 10GB drive for my application."

When you create a PVC, Kubernetes looks for a PV that meets the requirements and connects it to your pod.

***Explain pod resource requests vs limits and how k8s uses them to schedule pods***

Resource requests are what a pod asks for as a minimum. Limits are the maximum a pod is allowed to use.

Requests: "I need at least 2 CPU cores and 4GB of RAM to run." Limits: "Don't let me use more than 4 CPU cores or 8GB of RAM."

Kubernetes uses requests to decide which node to put a pod on. It looks for a node with enough free resources to meet the pod's requests.

Pod Request: 2 CPU, 4GB RAM

Pod Limit: 4 CPU, 8GB RAM

[Node 1: 4 CPU, 16GB RAM] ← Kubernetes schedules pod here

[Node 2: 1 CPU, 8GB RAM] ← Not enough resources

***Explain how to achieve pod affinity/anti-affinity***

Pod affinity and anti-affinity help you control where pods are placed relative to each other.

Affinity: "I want to be near pods with this label." Anti-affinity: "I don't want to be near pods with this label."

You can use these to:

* Keep related pods together for better performance
* Spread pods across nodes for reliability

Example: Affinity: "Put me on a node with a 'GPU' label." Anti-affinity: "Don't put me on a node that already has a database pod."

Here's a simple YAML example to demonstrate pod affinity and anti-affinity:

apiVersion: apps/v1

kind: Deployment

metadata:

name: web-app

spec:

replicas: 3

selector:

matchLabels:

app: web

template:

metadata:

labels:

app: web

spec:

affinity:

podAffinity:

requiredDuringSchedulingIgnoredDuringExecution:

- labelSelector:

matchExpressions:

- key: app

operator: In

values:

- cache

topologyKey: "kubernetes.io/hostname"

podAntiAffinity:

preferredDuringSchedulingIgnoredDuringExecution:

- weight: 100

podAffinityTerm:

labelSelector:

matchExpressions:

- key: app

operator: In

values:

- web

topologyKey: "kubernetes.io/hostname"

containers:

- name: web-app

image: web-app:v1

Pod Affinity:

* The web-app pods want to be scheduled on nodes that also have pods with the label "app: cache"
* This might be to ensure the web app is close to its cache for better performance

Pod Anti-Affinity:

* The web-app pods prefer not to be scheduled on nodes that already have web-app pods.
* This helps spread the web-app pods across different nodes for better reliability.

Key points:

* "requiredDuringSchedulingIgnoredDuringExecution": Must be satisfied for pod scheduling
* "preferredDuringSchedulingIgnoredDuringExecution": Kubernetes will try to satisfy, but not required.