Kubernetes Headless Service

Kubernetes Services are a fundamental abstraction that defines how pods are accessed within a cluster. While most people are familiar with **ClusterIP**, **NodePort**, and **LoadBalancer** services, **Headless Services** serve a very different and powerful purpose. This article covers what a Kubernetes Headless Service is, why it exists, how it works, and when to use it.

What Is a Headless Service?

A **Headless Service** is a Kubernetes Service without **a Cluster IP**. Instead of a single virtual IP and load balancing, it provides direct pod IP access via DNS.

You create a headless service by setting:

```
spec:
  clusterIP: None
```

When this is done:

- Kubernetes does **not** allocate a virtual IP.
- Kube-proxy does **not** perform load balancing.
- DNS returns **pod IPs directly**, not a single service IP.

Why Headless Services exist

Traditional services abstract away individual pods and provide load balancing. However, some applications **need to know exactly which pod they are talking to**. There are some scenarios where more fine grained controlled is needed on which pod we want to connect for a specific need.

Common reasons can include:

- Stateful applications (databases, message brokers, caching systems)
- Leader–follower architectures
- Custom client-side load balancing
- Peer discovery systems

Headless services enable **service discovery without traffic routing**.

With a Normal Service

A DNS query in normal Service

```
my-service.default.svc.cluster.local
```

returns a service IP:

```
10.96.0.15
```

Traffic is load-balanced across pods.

With a Headless Service

DNS query:

```
my-headless-service.default.svc.cluster.local
```

Returns:

```
10.244.1.5
10.244.2.8
10.244.3.12
```

Each IP corresponds to a **pod**, not the service.

The client has the control to decide:

- Which pod to connect to
- How to load-balance or route traffic

Creating a Headless Service

To create a headless service, clusterIP needs to be set to **None** in the definition of a clusterIP server. Here is an example :

```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-headless-service
spec:
  clusterIP: None
```

```
  selector:
    app: my-app
  ports:
    - port: 80
      targetPort: 8080
```

Key points:

- **clusterIP: None** makes it headless
- Pods matching the selector are published via DNS.
- No virtual IP is created.

Headless Service with StatefulSets

Headless services are most commonly used with **StatefulSets**.

Why do we need it?

StatefulSets give each pod:

- A stable hostname
- A stable network identity
- Persistent storage

Example DNS names:

```
pod-0.my-headless-service.default.svc.cluster.local
pod-1.my-headless-service.default.svc.cluster.local
```

This is ideal for:

- Databases like Cassandra, MongoDB, MySQL
- Kafka and Zookeeper
- etcd clusters

DNS Records in Headless Services

Depending on the configuration, Kubernetes creates:

- **A records** → One per pod IP
- **SRV records** → For named ports

Example SRV lookup:

```
my-port.tcp.my-headless-service.default.svc.cluster.local
```

This allows advanced clients to discover:

- Pod IP
- Port
- Protocol

When to use a Headless Service

Use a headless service when:

- You need **direct pod-to-pod communication.**
- Clients must **discover all pods.**
- You want **custom or client-side load balancing.**
- You're running **stateful or clustered applications.**

Do **not** use it when:

- You just want simple load balancing.
- You want a single stable service IP.
- The app does not need pod awareness.

Advantages and Limitations

Advantages

- Full control over traffic routing.
- Essential for stateful workloads.
- Works seamlessly with StatefulSets.
- Enables advanced service discovery.

Limitations

- No built-in load balancing.
- Client logic becomes more complex.
- Not suitable for simple stateless apps.

Author: https://medium.com/@narinderkaurmakkar1