kubernetes

Lev Epshtein
lev@opsguru.io

NAYA College

Kubernetes

Basic part II

- Init container
- DaemonSet
- Jobs
- ConfigMaps/Secrets
- Volumes
- Stateless Set vs Stateful Set
- Kustomize

- K8S Init Container -

K8S Init container

A Pod can have multiple containers running apps within it, but it can also have one or more init containers, which are run before the app containers are started.

Init containers are exactly like regular containers, except:

- Init containers always run to completion.
- Each init container must complete successfully before the next one starts.

  https://kubernetes.io/docs/concepts/workloads/pods/init-containers/

Demo/Labs

K8S Init container:

In our repository

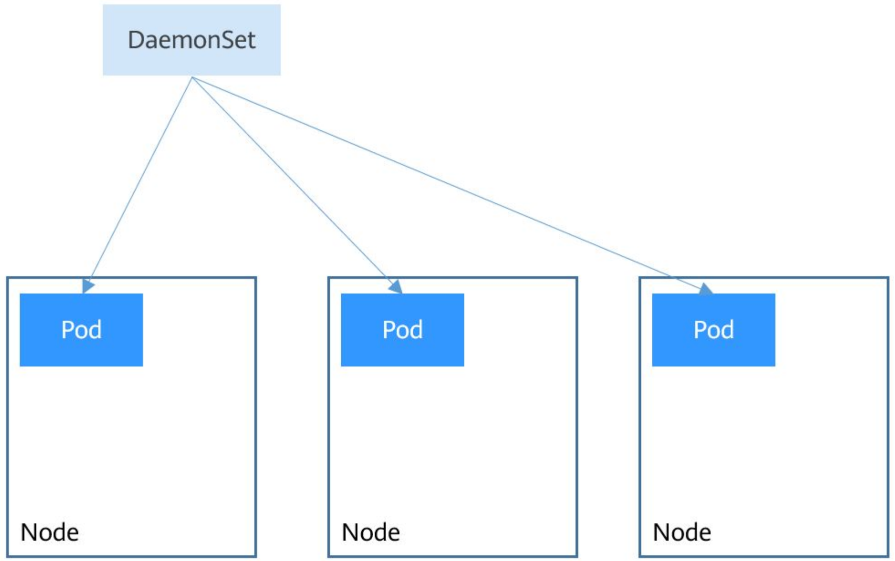
```
$ cd cognyte-devops-32/k8s-2/init-container/
```

Follow README.MD

- K8S DaemonSet -

K8S DaemonSet

- A DaemonSet ensures a copy of a Pod is running across a set of nodes in a
- Kubernetes cluster. DaemonSets are used to deploy system daemons such as **log collectors** and **monitoring** agents, which typically must run on every node. DaemonSets share similar functionality with ReplicaSets; both create Pods that are expected to be long-running services and ensure that the desired state and the observed state of the cluster match.



Demo/Labs

K8S DaemonSet:

In our repository

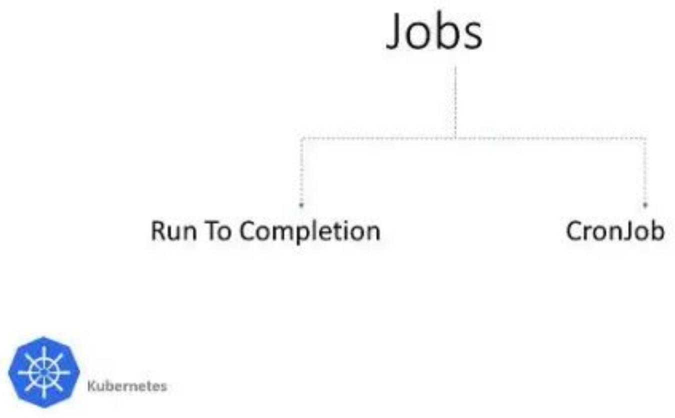
$ cd cognyte-devops-32/k8s-2/DaemonSet/

Follow README.MD

- K8S Jobs -

K8S Jobs

- A Job creates one or more Pods and ensures that a specified number of them successfully terminate.
- As pods successfully complete, the Job tracks the successful completions. When a specified number of successful completions is reached, the task (ie, Job) is complete.
- Deleting a Job will clean up the Pods it created.



K8S Jobs patterns

- The Job object can be used to support reliable **parallel execution** of Pods.
- It does support parallel processing of a set of independent but related work items. These might be emails to be sent, frames to be rendered, files to be transcoded, ranges of keys in a NoSQL database to scan, and so on.

For more information please check documentation:
https://kubernetes.io/docs/concepts/workloads/controllers/job/#job-patterns:~:text=namespace%20can%20consume.-,Job%20patterns,-The%20Job%20object

Job Example - Fine Parallel Processing Using a Work Queue

- Start a storage service to hold the work queue
- Create a queue, and fill it with messages
- Start a Job that works on tasks from the queue.

Demo/Labs

K8S Job:

In our repository

```
$ cd cognyte-devops-32/k8s-2/Job/
```

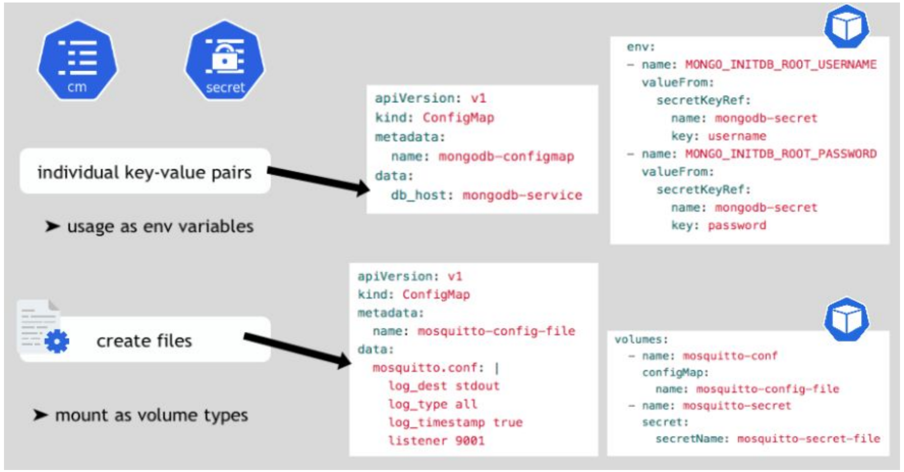
Follow README.MD

K8S ConfigMaps/Secrets

K8S - ConfigMap/Secrets

ConfigMaps allow you to decouple configuration artifacts from image content to keep containerized applications portable.

Kubernetes secret objects let you store and manage sensitive information, such as passwords, OAuth tokens, and ssh keys. Putting this information in a secret is safer and more flexible than putting it verbatim in a Pod definition or in a container image.



```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: mongodb-configmap
data:
  db_host: mongodb-service
```

```yaml
env:
- name: MONGO_INITDB_ROOT_USERNAME
  valueFrom:
    secretKeyRef:
      name: mongodb-secret
      key: username
- name: MONGO_INITDB_ROOT_PASSWORD
  valueFrom:
    secretKeyRef:
      name: mongodb-secret
      key: password
```

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: mosquitto-config-file
data:
  mosquitto.conf: |
    log_dest stdout
    log_type all
    log_timestamp true
    listener 9001
```

```yaml
volumes:
  - name: mosquitto-conf
    configMap:
      name: mosquitto-config-file
  - name: mosquitto-secret
    secret:
      secretName: mosquitto-secret-file
```

Demo/Labs

K8S ConfigMap:

In our repository

```
$ cd cognyte-devops-32/k8s-2/configMap/
```

Follow README.MD

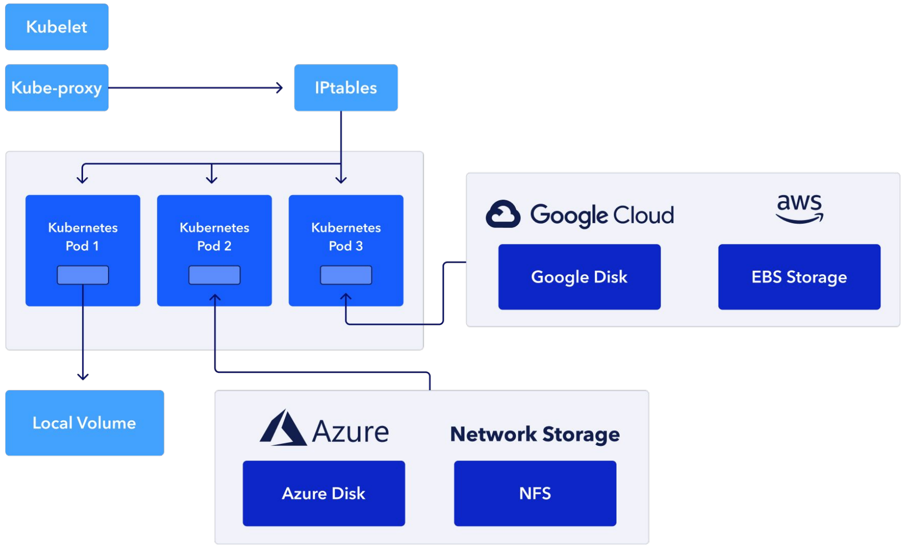
K8S Volumes

K8S Volumes

Kubernetes supports several types of volumes.

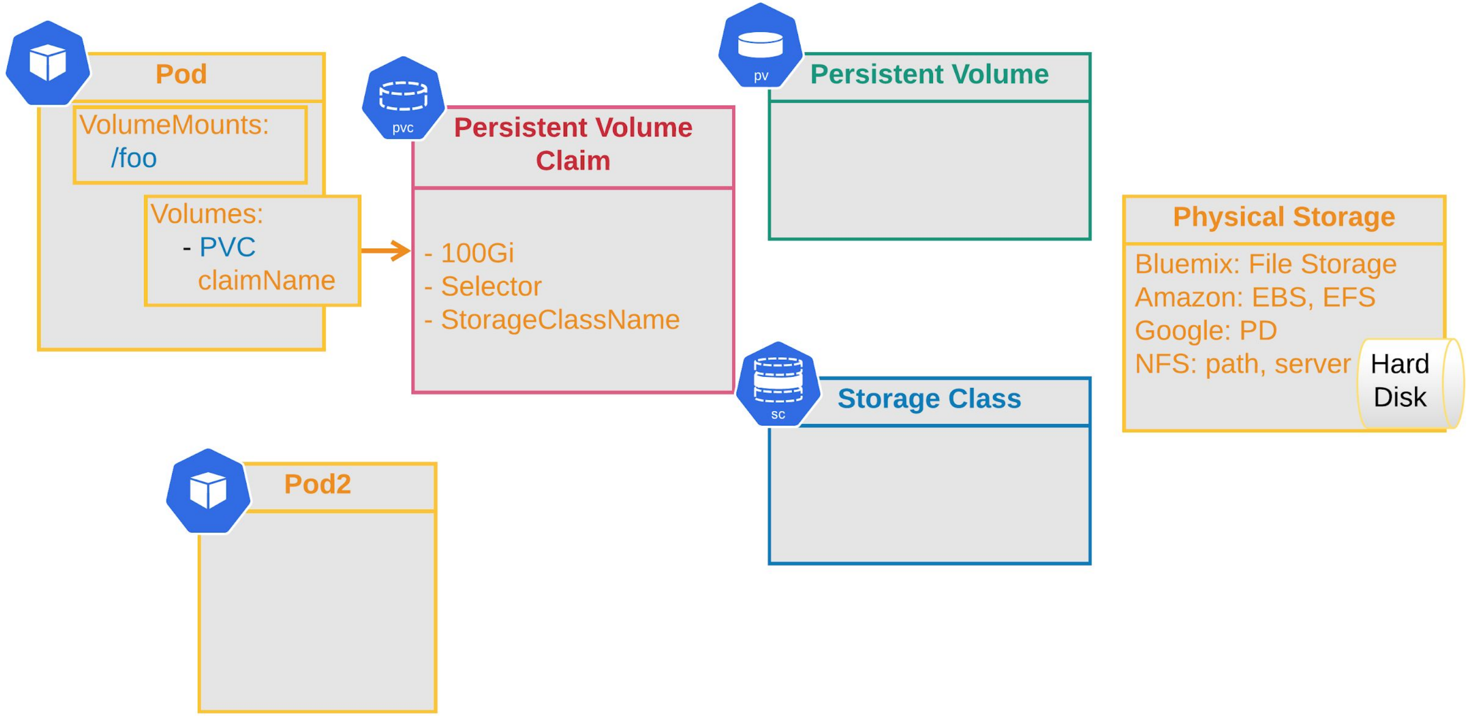
- emptydir
- hostPath
- Cloud Volumes
- NFS
- Persistent Volume Claim - PVC
- For full list visit:

https://kubernetes.io/docs/concepts/storage/volumes/



NAYA College

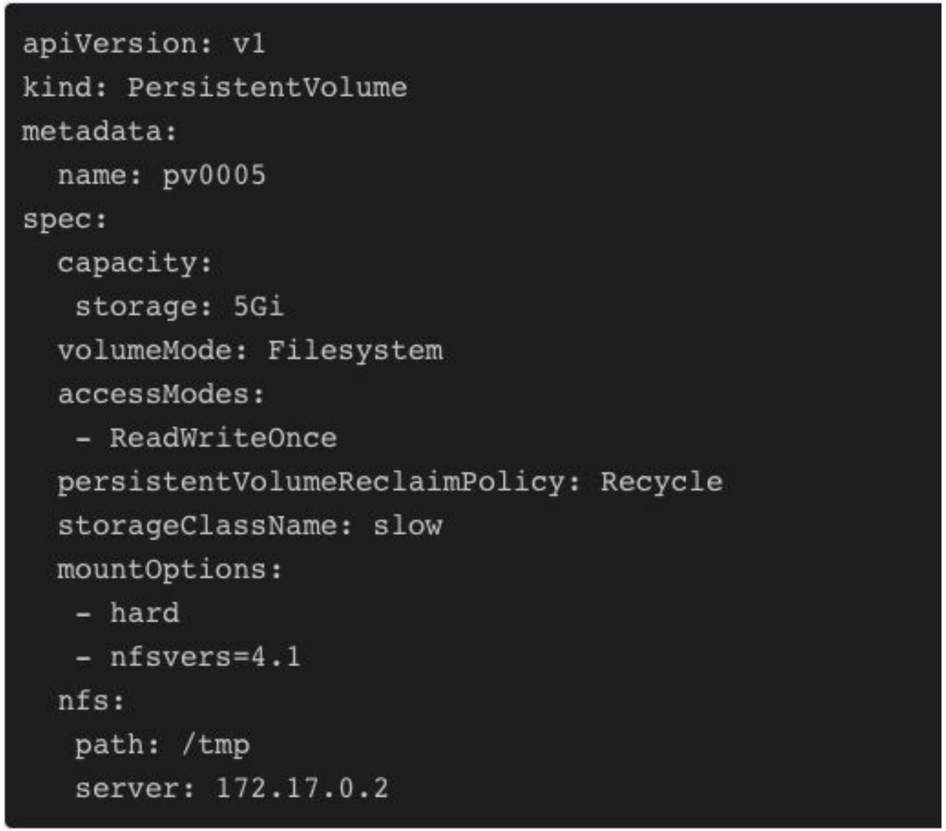
K8S Volumes


Pod
VolumeMounts:
/foo
Volumes:
- PVC
claimName
pvc
Persistent Volume Claim
- 100Gi
- Selector
- StorageClassName
pv
Persistent Volume
sc
Storage Class
Physical Storage
Bluemix: File Storage
Amazon: EBS, EFS
Google: PD
NFS: path, server
Hard Disk
Pod2

What are Persistent Volumes (PVs)?

PV is the way to define the storage data, such as storage classes or storage implementations. Unlike ordinary volumes, PV is a resource object in a Kubernetes cluster; creating a PV is equivalent to creating a storage resource object. To use this resource, it must be requested through persistent volume claims (PVC). A PVC volume is a request for storage, which is used to mount a PV into a Pod. The cluster administrator can map different classes to different service levels and different backend policies.

What are Persistent Volumes (PVs)?



```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv0005
spec:
  capacity:
   storage: 5Gi
  volumeMode: Filesystem
  accessModes:
   - ReadWriteOnce
  persistentVolumeReclaimPolicy: Recycle
  storageClassName: slow
  mountOptions:
   - hard
   - nfsvers=4.1
  nfs:
   path: /tmp
   server: 172.17.0.2
```

Features of Persistent Volumes

Capacity - Generally, a PV will specify storage capacity. This is set by using the capacity property of the PV.

Volume Modes - Kubernetes supports two volume modes of persistent volumes. A valid value for volume mode can be either Filesystem or Block. Filesystem is the default mode if the volume mode is not defined.

Access Modes

- ReadOnlyMany(ROX) allows being mounted by multiple nodes in read-only mode.
- ReadWriteOnce(RWO) allows being mounted by a single node in read-write mode.
- ReadWriteMany(RWX) allows multiple nodes to be mounted in read-write mode.

Features of Persistent Volumes

Capacity - Generally, a PV will specify storage capacity. This is set by using the capacity property of the PV.

Volume Modes - Kubernetes supports two volume modes of persistent volumes. A valid value for volume mode can be either Filesystem or Block. Filesystem is the default mode if the volume mode is not defined.

Access Modes

- ReadOnlyMany(ROX) allows being mounted by multiple nodes in read-only mode.
- ReadWriteOnce(RWO) allows being mounted by a single node in read-write mode.
- ReadWriteMany(RWX) allows multiple nodes to be mounted in read-write mode.

Features of Persistent Volumes

Class - A PV can specify a StorageClass to dynamically bind the PV and PVC, where the specific StorageClass is specified via the storageClassName property. If no PV is specified with this property, it can only bind to a PVC that does not require a specific class.

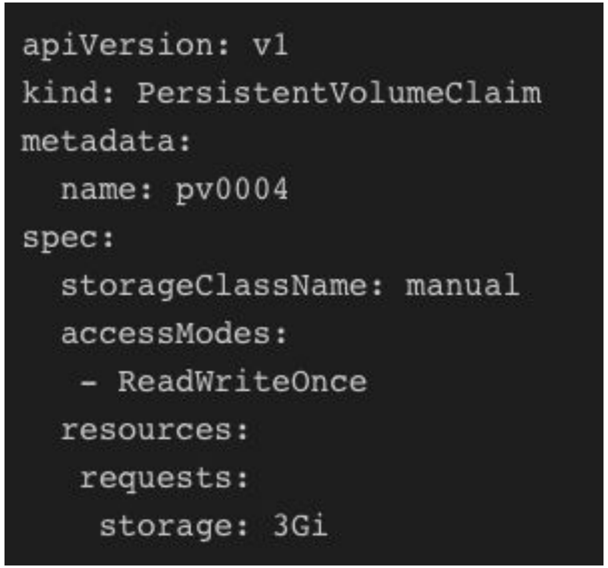
Reclaim Policy - When the node no longer needs persistent storage, the reclaiming strategies that can be used include:

- Retain - meaning the PV, until deleted, is kept alive.
- Recycle - meaning the data can be restored later after getting scrubbed.
- Delete - associated storage assets (such as AWS EBS, GCE PD, Azure Disk, and OpenStack Cinder volumes) are deleted.

What are Persistent Volume Claims (PVCs)?

PVC is a declaration defining the request for storage data usage, which is mounted into a Pod for use. PVC is configured for use by developers, who do not necessarily care about the specific implementation of the underlying data storage, but more so about the business-related data storage size, access methods, etc.

configuration file for the PersistentVolumeClaim



```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pv0004
spec:
  storageClassName: manual
  accessModes:
   - ReadWriteOnce
  resources:
   requests:
    storage: 3Gi
```

Lifecycle of PV and PVC

In a Kubernetes cluster, a PV exists as a storage resource in the cluster. PVCs are requests for those resources and also act as claim checks to the resource. The interaction between PVs and PVCs follows this lifecycle:

- Provisioning - the creation of the PV, either directly (static) or dynamically using StorageClass.
- Binding - assigning the PV to the PVC.
- Using - Pods use the volume through the PVC.
- Reclaiming - the PV is reclaimed, either by keeping it for the next use or by deleting it directly from the cloud storage.

A volume will be in one of the following states

- Available - this state shows that the PV is ready to be used by the PVC.
- Bound - this state shows that the PV has been assigned to a PVC.
- Released - the claim has been deleted, but the cluster has not yet reclaimed the resource.
- Failed - this state shows that an error has occurred in the PV.

Demo/Labs

K8S Volumes:

In our repository

```
$ cd cognyte-devops-32/k8s-2/Volumes/
```

Follow README.MD

Stateless Set vs Stateful Set

Stateless Set vs Stateful Set

- **Stateless** Protocols do not need the server to save the state of a process.
- They act independently by taking the previous or next request into consideration.
- They are easy to implement.
- It is relatively easier to scale architecture.

- **Stateful** Protocols require the server to save the state of a process.
- They react only by the current state of a transaction or request.
- They are logically heavy to implement.
- Scaling architecture is difficult and complex.

K8S StatefulSet

StatefulSets are valuable for applications that require one or more of the following.

- Stable, unique network identifiers.
- Stable, persistent storage.
- Ordered, graceful deployment and scaling.
- Ordered, automated rolling updates.

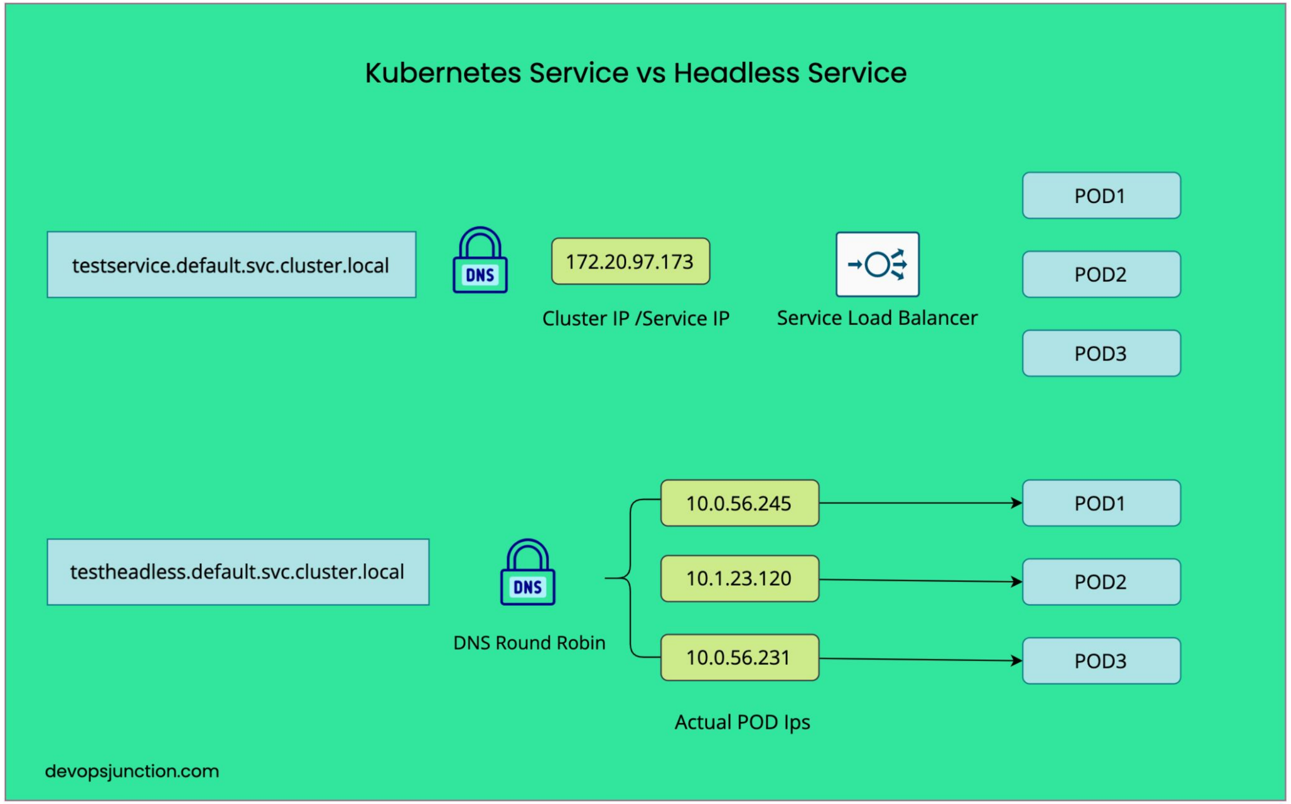
Kubernetes Headless Service

Kubernetes headless service is a Kubernetes service that does not assign an IP address to itself. Instead, it returns the IP addresses of the pods associated with it directly to the DNS system, allowing clients to connect to individual pods directly.

The main difference between Kubernetes service and Kubernetes headless service is that a regular Kubernetes service allocates a single IP address to the service, which is used as a proxy to balance traffic between multiple pods,

whereas a Kubernetes headless service does not allocate an IP address to the service and returns the IP addresses of the individual pods directly to the DNS system.

Kubernetes Headless Service



Demo/Labs

K8S StatefulSet:

In our repository

```
$ cd cognyte-devops-32/k8s-2/StatefulSet/
```

Follow README.MD

Q&A