K8s: Deployments vs StatefulSets vs DaemonSets

Understanding StatefulSets in Kubernetes

Kubernetes provides a basic resource called Pod. A pod is the smallest deployable unit in Kubernetes which is actually a wrapper around containers. A pod can have one or more containers and you can pass different configurations to the container(s) using the pod's configuration e. g. passing environment variables, mounting volumes, having health checks, etc. For more details about pods, check Pod.

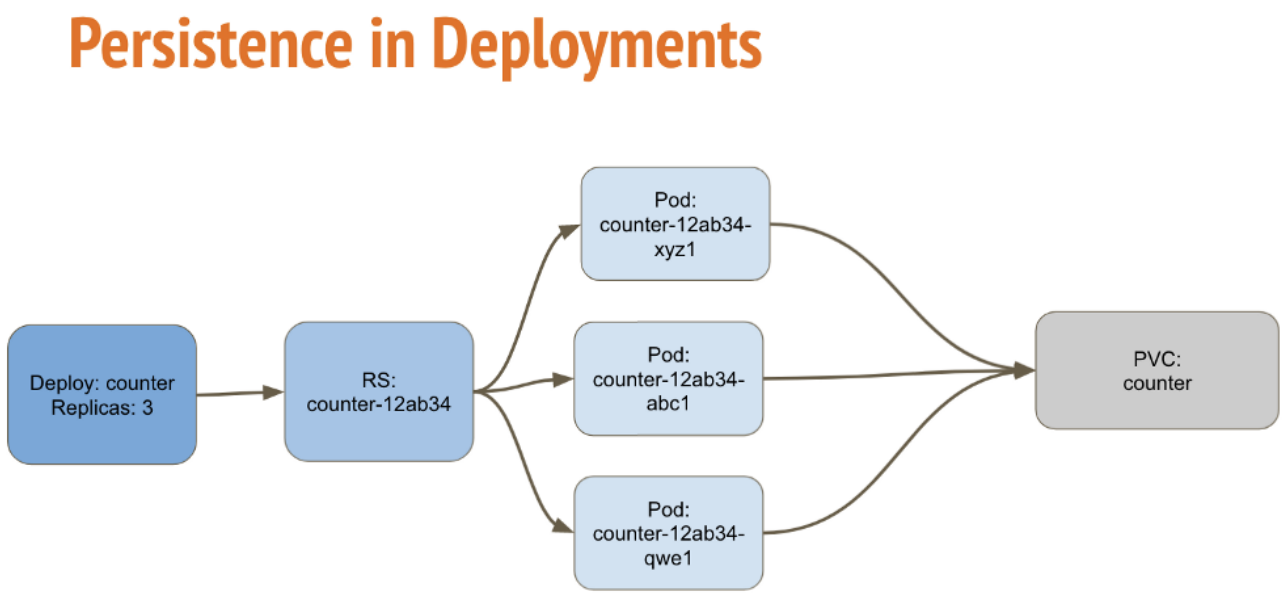
In this post, we will be discussing three different ways to deploy your application (pods) on Kubernetes using different Kubernetes resources. Below are 3 different resources that Kubernetes provides for deploying pods.

1. **Deployments**
2. **StatefulSets**
3. **DaemonSets**

Deployments

Deployment is the easiest and most used resource for deploying your application. It is a Kubernetes controller that matches the current state of your cluster to the desired state mentioned in the Deployment manifest. e.g. If you create a deployment with 1 replica, it will check that the desired state of ReplicaSet is 1 and the current state is 0, so it will create a ReplicaSet, which will further create the pod. If you create a deployment with name **counter**, it will create a ReplicaSet with name **counter-<replica-set-id>**, which will further create a Pod with name **counter-<replica-set->-<pod-id>.**

Deployments are usually used for stateless applications. However, you can save the state of deployment by attaching a Persistent Volume to it and making it stateful, but all the pods of deployment will be sharing the **same Volume, and data across all of them will be the same.**



Persistence for Deployments sharing single Volume can **cause Data Inconsistency. This is not good for databases deployment because we will encounter data corruption.**

Deployment example:

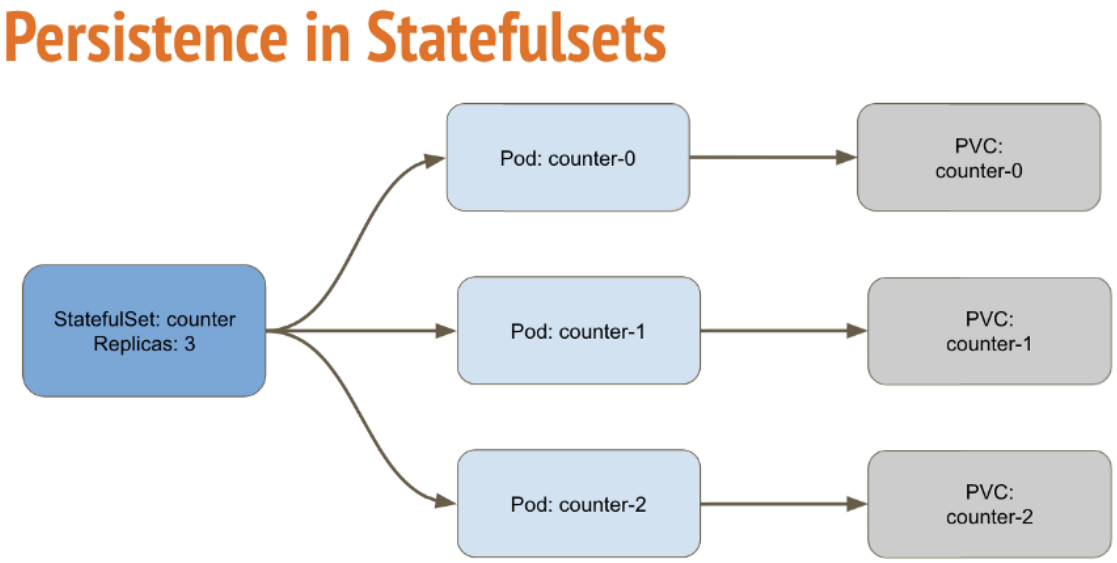
```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: counter
spec:
  replicas: 1
  selector:
    matchLabels:
      app: counter
  template:
    metadata:
      labels:
        app: counter
    spec:
      containers:
      - name: counter
        image: "kahootali/counter:1.1"
        volumeMounts:
        - name: counter
          mountPath: /app/
      volumes:
      - name: counter
        persistentVolumeClaim:
          claimName: counter
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: counter
spec:
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 50Mi
  storageClassName: efs
```

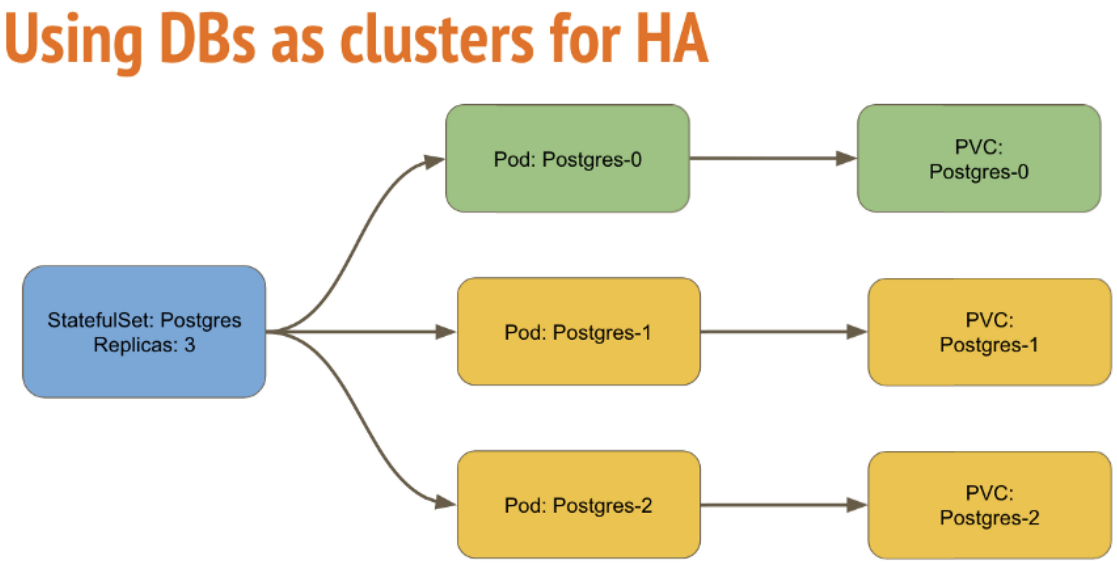
StatefulSets

StatefulSet is a Kubernetes resource used to manage **stateful applications**. It manages the deployment and scaling of a set of Pods and provides guarantees about the **ordering and uniqueness of these Pods.**

StatefulSet is also a Controller but unlike Deployments, it doesn't create ReplicaSet rather it creates the Pod with a unique n**aming convention**. e.g. If you create a StatefulSet with name **counter,** it will create a pod with name **counter-0,** and for multiple replicas of a statefulset, their names will increment like **counter-0**, **counter-1**, **counter-2, etc**

Every replica of a stateful set will have its own state, and each of the pods will be creating its own PVC **(Persistent Volume Claim)**. So a statefulset with 3 replicas will create 3 pods, each having its own Volume, so **total 3 PVCs.**





StatefulSets example:

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: counter
spec:
  serviceName: "counter-app"
  selector:
    matchLabels:
      app: counter
  replicas: 1
  template:
    metadata:
      labels:
        app: counter
    spec:
      containers:
      - name: counter
        image: "kahootali/counter:1.1"
        volumeMounts:
        - name: counter
          mountPath: /app/
  volumeClaimTemplates:
  - metadata:
      name: counter
    spec:
      accessModes: [ "ReadWriteMany" ]
      storageClassName: efs
      resources:
        requests:
          storage: 50Mi
```

DaemonSet

A DaemonSet is a controller that ensures that the pod runs on **all the nodes of the cluster**. If a node is added/removed from a cluster, **DaemonSet automatically adds/deletes the pod.**

Some typical use cases of a DaemonSet is to run cluster level applications like:

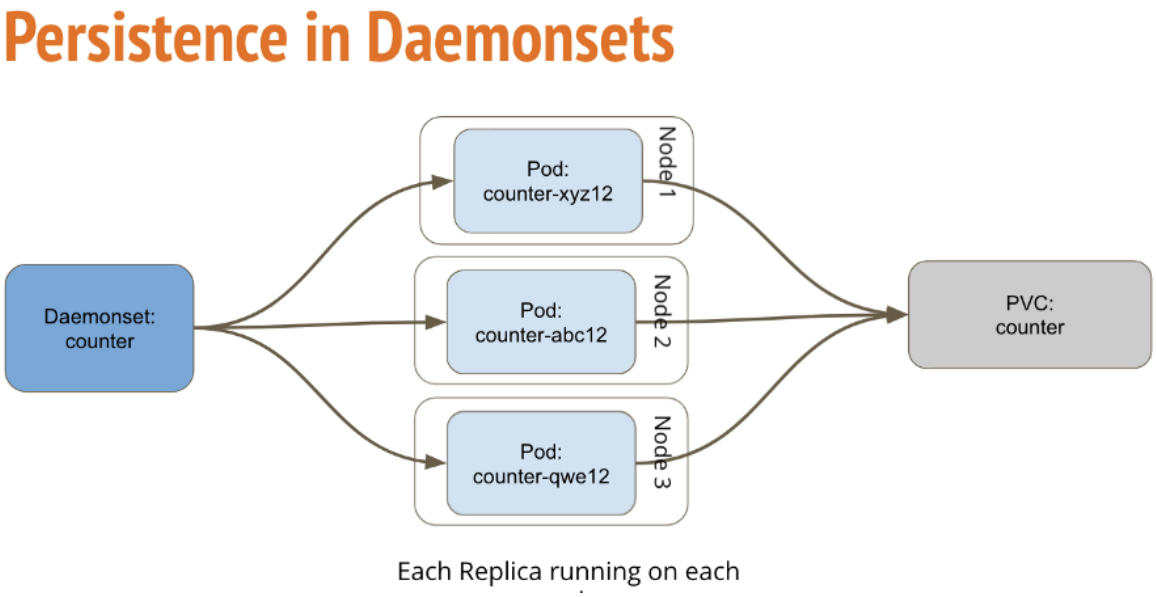
- **Monitoring Exporters**: You would want to monitor all the nodes of your cluster so you will need to run a monitor on all the nodes of the cluster like NodeExporter.
- **Logs Collection Daemon:** You would want to export logs from all nodes so you would need a DaemonSet of log collectors like Fluentd to export logs from all your nodes.

However, Daemonset automatically doesn't run on nodes that have a taint e.g. Master. You will have to specify the tolerations for it on the pod.

Taints are a way of telling the nodes to repel the pods i.e. no pods will be scheduled on this node unless the pod tolerates the node with the same toleration. The master node is already tainted by:

This means it will repel all pods that do not tolerate this taint, so for daemonset to run on all nodes, you would have to add the following tolerations on DaemonSet

which means that it should tolerate all nodes.



Each Replica running on each

Similar to ReplicaSet, but DaemonSets run one replica per node in the cluster

DaemonSet Example:

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: counter-app
spec:
  selector:
    matchLabels:
      app: counter
  template:
    metadata:
      name: counter-app
      labels:
        app: counter
    spec:
      tolerations:
      - effect: NoSchedule
        operator: Exists
      containers:
      - name: counter
        image: "kahootali/counter:1.1"
        volumeMounts:
        - name: counter
          mountPath: /app/
      volumes:
      - name: counter
        persistentVolumeClaim:
          claimName: counter
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: counter
spec:
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 50Mi
  storageClassName: efs
```