# **DaemonSet**

A DaemonSet ensures that all Nodes run a copy of a Pod. As nodes are added to the cluster, Pods are added to them. As nodes are removed from the cluster, those Pods are garbage collected. Deleting a DaemonSet will clean up the Pods it created.

They're for workloads where you want **high-availabilty across multiple nodes**, but you don't need high levels of scale. Or where the app needs to work with each node, e.g. collecting logs.

Some typical uses of a DaemonSet are:

* running a cluster storage daemon on every node, such as: - glusterd / ceph
* running a logs collection daemon on every node, such as: - fluentd / logstash
* running a node monitoring daemon on every node, such as: - Prometheus Node Exporter / collectd / Datadog agent

**This is how the master pods on the worker nodes run, such as kube-proxy and kubelet.**

apiVersion: apps/v1

kind: DaemonSet

metadata:

  name: nginx-ds

spec:

 #replicas: 10

  selector:

    matchLabels:

      app: nginx-app

  template:

    metadata:

      name: nginx-pod

      labels:

        app: nginx-app

    spec:

      containers:

      - name: mynginx-con

        image: nginx:1.19.0

        ports:

        - containerPort: 80

**Updating DaemonSet**

DaemonSets run exactly one Pod on each node, so the update behaviour is to remove Pods before starting replacements.

This is different from Deployments, which default to starting new Pods and checking they're healthy before removing old ones. DaemonSet updates can break your app.

**Delete the DaemonSet but retain the Pod**

Kubernetes maintains the relationship between Pods and controllers, but it lets you break that relationship with non-cascading deletes.

* kubectl delete ds <dsname> --cascade=orphan
* kubectl get ds
* kubectl get po -l app=nginx

updateStrategy = OnDelete (manually delete old pods) or RollingUpdate