WHAT IS KUBERNETES DAEMONSET? K8S DAEMONSETS EXPLAINED

In this blog post, we will discuss Kubernetes DaemonSet, including what it's used for, how to create one, and how to work with it, using a simple example. To understand this topic, you'll need a basic understanding of K8S, Kubectl, and Pods. To best follow along with the demo, you'll want to have a k8s cluster with multiple nodes.

What is a DaemonSet?

DaemonSets are used to ensure that some or all of your K8S nodes run a copy of a pod, which allows you to run a daemon on every node.

When you add a new node to the cluster, a pod gets added to match the nodes. Similarly, when you remove a node from your cluster, the pod is put into the trash. Deleting a DaemonSet cleans up the pods that it previously created.

Why use DaemonSets?

Now that we understand DaemonSets, here are some examples of why and how to use it:

- To run a daemon for cluster storage on each node, such as:
  - **glusterd**
  - **ceph**
- To run a daemon for logs collection on each node, such as:
  - **fluentd**

    - **logstash**
- To run a daemon for node monitoring on ever note, such as:
    - Prometheus Node Exporter
    - **collectd**
    - Datadog agent

As your use case gets more complex, you can deploy multiple DaemonSets for one kind of daemon, using a variety of flags or memory and CPU requests for various hardware types.

How are DaemonSets scheduled?

DaemonSets are scheduled either with **DaemonSet controller** or **default scheduler**. Let's compare:

- **DaemonSet controller.** When you specify **.spec.nodeName** during pod creation, these pods will have the machine already selected. In this type of scheduler, the unschedulable node field is not respected. The scheduler can also create pods without starting the scheduler—this helps cluster bootstrap.
    - By default, this controller is disabled in K8S v1.12+.
- **Default scheduler.** Using ScheduleDaemonSetPods allows for scheduling DaemonSets using defaults, not DaemonSet controller. To do this, add **NodeAffinity** to the DaemonSet pods (instead of .spec.nodeName).
    - By default, the scheduler will replace your DaemonSet pod node affinity if it already exists.

Working with DaemonSets

Like every manifest in K8S, the following fields are required:

- **apiVersion**
- **kind**
- **metadata**

There are certain things to keep in mind when using DaemonSets:

- When you create a DaemonSet, the **.spec.selector** cannot be changed. Changing it will break things.
- You must specify a pod selector to match the **.spec.template** labels.
- Typically, you should not create pods with labels that match this selector—either directly via another DaemonSet, or indirectly, via another controller (like ReplicaSet). If you do, the DaemonSet controller thinks it created those pods.

Note that you can deploy a DaemonSet to run only on some nodes, not all nodes. To do so, specify **.spec.template.spec.nodeSelector**. It will deploy to any node that matches the selector.

Deleting a DaemonSet

Deleting a DaemonSet is simple. Run **kubectl delete fluentd-es-demo**. This will delete the DaemonSet and its associated pods.

To delete DaemonSet without deleting the pods, add the flag **–cascade=false with kubectl**.

A DaemonSet example

To show additional fields in the manifest, we'll deploy this example of **fluentd-elasticsearch** image that will run on every node. This idea is that we want to have a daemon of this on every node collecting logs for us and sending it to ES.

demo.yaml

```
apiVersion: apps/v1 #required fields
kind: DaemonSet #required fields
metadata: #required fields
name: fluentd-es-demo
labels:
k8s-app: fluentd-logging
spec:
selector:
matchLabels:
name: fluentd-es #this must match the label below
template: #required fields
metadata:
labels:
name: fluentd-es #this must match the selector above
spec:
tolerations:
- key: node-role.kubernetes.io/master
effect: NoSchedule
containers:
- name: fluentd-es-example
image: k8s.gcr.io/fluentd-elasticsearch:1.20
resources:
limits:
memory: 200Mi
requests:
cpu: 100m
memory: 200Mi
volumeMounts:
- name: varlog
mountPath: /var/log
- name: varlibdockercontainers
mountPath: /var/lib/docker/containers
readOnly: true
terminationGracePeriodSeconds: 30
volumes:
- name: varlog
hostPath:
path: /var/log
- name: varlibdockercontainers
hostPath:
```

```
path: /var/lib/docker/containers
```

Now, we'll run **kubectl create -f demo.yaml** to deploy the example.

```
$ kubectl create -f demo.yaml
daemonset.apps "fluentd-es-demo" created
```

Make sure it's running:

```
$ kubectl get daemonset
NAME              DESIRED   CURRENT   READY     UP-TO-DATE   AVAILABLE   NODE
SELECTOR   AGE
fluentd-es-demo   3         3         3         3            3
<none>          59s
```

Now, let's see how many nodes we have. Run **kubectl get nodes** to see the identity of our nodes.

```
$ kubectl get node
NAME                 STATUS    ROLES     AGE       VERSION
node2                Ready     <none>    92d       v1.10.3
node1                Ready     <none>    92d       v1.10.3
node3                Ready     <none>    92d       v1.10.3
```

Finally, let's confirm that we have all pods running and to make sure they are running on every node.

```
$ kubectl get pod -o wide
NAME                            READY     STATUS    RESTARTS   AGE       IP
NODE
fluentd-es-demo-bfpf9           1/1       Running   0          1m
10.0.0.3            node3
fluentd-es-demo-h4w85           1/1       Running   0          1m
10.0.0.1            node1
fluentd-es-demo-xm2rl           1/1       Running   0          1m
10.0.0.2           node2
```

We can see that not only is our **fluentd-es-demo** pods running, but there is a copy of each on every node.

Additional resources

For more on Kubernetes, explore these resources:

- Kubernetes Guide, with 20+ articles and tutorials
- BMC DevOps Blog
- The State of Kubernetes in 2020
- CKA Labs (10) - Kubernetes DaemonSets
- An Introduction to Kubernetes DaemonSets
- GitHub