



What is a DaemonSet?

Kubernetes - Beginners | Intermediate | Advanced

[View on GitHub](#) [Join Slack](#)

What is a DaemonSet?

Say, you want to run a process on all the nodes of the cluster. One of the easy solution could be running cron job that runs on machine boot or reboot. Also, alternatively one can use the `/etc/init.local` file to ensure that a specific process or command gets executed as soon as the server gets started. Though it looks to be viable solution, using the node itself to control the daemons that run on it (especially within a Kubernetes cluster) suffers some drawbacks:

- We need the process to remain running on the node as long as it is part of the cluster. It should be terminated when the node is evicted.
- The process may need a particular runtime environment that may or may not be available on the node (for example, a specific JDK version, a required kernel library, a specific Linux distro...etc.). So, the process should run inside a container. Kubernetes uses Pods to run containers. This daemon should be aware that it is running within Kubernetes. Hence, it has access to other pods in the cluster and is part of the network.

Enter DaemonSets

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.

A Daemonset is another controller that manages pods like Deployments, ReplicaSets, and StatefulSets. It was created for one particular purpose: ensuring that the pods it manages to run on all the cluster nodes. As soon as a node joins the cluster, the DaemonSet ensures that it has the necessary pods running on it. When the node leaves the cluster, those pods are garbage collected.

DaemonSets are used in Kubernetes when you need to run one or more pods on all (or a subset of) the nodes in a cluster. The typical use case for a DaemonSet is logging and monitoring for the hosts. For example, a node needs a service (daemon) that collects health or log data and pushes them to a central system or database (like ELK stack). DaemonSets can be deployed to specific nodes either by the nodes' user-defined labels or using values provided by Kubernetes like the node hostname.

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 every node, 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.

Creating your first DaemonSet Deployment

```
git clone https://github.com/collabnix/kubelabs
cd kubelabs/DaemonSet101
kubectl apply -f daemonset.yml
```

The other way to do this:

```
$ kubectl create -f daemonset.yml --record
```

The `--record` flag will track changes made through each revision.

Getting the basic details about daemonsets:

```
$ kubectl get daemonsets/prometheus-daemonset
```

Further Details

```
kubectl describe daemonset/prometheus-daemonset
```

```
[node1 DaemonSet101]$ kubectl describe daemonset/prometheus-daemonset
Name:                prometheus-daemonset
Selector:             name=prometheus-exporter,tier=monitoring
Node-Selector:        <none>
Labels:               name=prometheus-exporter
                     tier=monitoring
Annotations:          deprecated.daemonset.template.generation: 1
                     kubectl.kubernetes.io/last-applied-configuration:
                       {"apiVersion":"extensions/v1beta1","kind":"DaemonSet","met
Desired Number of Nodes Scheduled: 1Current Number of Nodes Scheduled: 1
Number of Nodes Scheduled with Up-to-date Pods: 1
Number of Nodes Scheduled with Available Pods: 1
Number of Nodes Misscheduled: 0
```

```

Pods Status:  1 Running / 0 Waiting / 0 Succeeded / 0 Failed
Pod Template:
  Labels:  name=prometheus-exporter
          tier=monitoring
  Containers:
    prometheus:
      Image:          prom/node-exporter
      Port:           80/TCP
      Host Port:      0/TCP
      Environment:    <none>
      Mounts:         <none>
  Volumes:          <none>
Events:
  Type      Reason              Age   From                      Message
  ----      -
Normal     SuccessfulCreate    3m21s daemonset-controller      Created pod: prometheus-exporter-1

```

Getting pods in daemonset:

```
$ kubectl get pods -lname=prometheus-exporter
```

```

[node1 DaemonSet101]$ kubectl get pods -lname=prometheus-exporterNAME
READY   STATUS    RESTARTS   AGE
prometheus-daemonset-nsjwx  1/1      Running    0          4m12s
[node1 DaemonSet101]$

```

Delete a daemonset:

```
$ kubectl delete -f daemonset.yml
```

Restrict DaemonSets To Run On Specific Nodes

By default, a DaemonSet schedules its pods on all the cluster nodes. But sometimes you may need to run specific processes on specific nodes. For example, nodes that host

database pods need different monitoring or logging rules. DaemonSets allow you to select which nodes you want to run the pods on. You can do this by using nodeSelector. With nodeSelector, you can select nodes by their labels the same way you do with pods. However, Kubernetes also allows you to select nodes based on some already-defined node properties. For example, `kubernetes.io/hostname` matches the node name. So, our example cluster has two nodes. We can modify the DaemonSet definition to run only on the first node. Let's first get the node names:

```
$kubectl get nodes
NAME      STATUS    ROLES    AGE   VERSION
node1     Ready     master   17m   v1.14.9
node2     Ready     <none>   17m   v1.14.9
```

You need to add the below entry in the above YAML file:

```
nodeSelector:
  kubernetes.io/hostname: node1
```

How To Reach a DaemonSet Pod

- There are several design patterns DaemonSet-pods communication in the cluster:
- The Push pattern: pods do not receive traffic. Instead, they push data to other services like ElasticSearch, for example.
- NodeIP and known port pattern: in this design, pods use the hostPort to acquire the node's IP address. Clients can use the node IP and the known port (for example, port 80 if the DaemonSet has a web server) to connect to the pod.
- DNS pattern: create a Headless Service that selects the DaemonSet pods. Use Endpoints to discover DaemonSet pods.
- Service pattern: create a traditional service that selects the DaemonSet pods. Use NodePort to expose the pods using a random port. The drawback of this approach is that there is no way to choose a specific pod.

Contributors


Sangam Biradar

Join KubeDaily

7 Members Online

Support


MEMBERS ONLINE

 h3ll_boy


 MEE6


 Nitinkashyap

Apex Legends

 ojaswa

 Parmeshwar

 prasad

 vikas027

Free voice chat from Discord

Connect

[Tweets by collabnix](#)

kubelabs is maintained by collabnix.

This page was generated by [GitHub Pages](#).