KUBERNETES DAEMONSETS, JOBS & CRONJOBS

DAEMONSETS:

1. Daemonsets ensures that exactly one pod is running on each node of the cluster.
2. If New node is added to the cluster, Daemonset will create a new pod on that node automatically.
3. If an existing node is removed from the cluster, Daemonset will delete its pod from that node.
4. If we try to delete a deamonset pod manually, Daemonset controller creates a new pod instantly on that node.

Use Cases for DaemonSets:

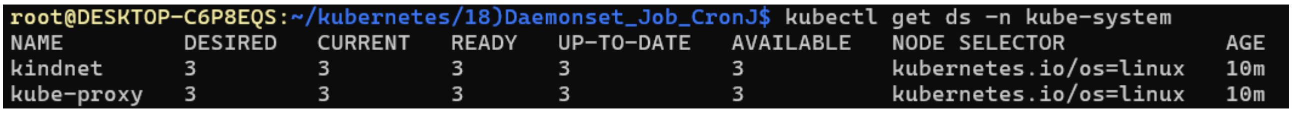
DaemonSets are typically used to deploy **cluster-wide system services** such as:

- Monitoring agents (e.g., Prometheus Node Exporter, Datadog Agent)
- Logging agents (e.g., Fluentd, Fluent Bit, Filebeat)
- Networking plugins (e.g., CNI plugins like AWS VPC CNI, Azure CNI, Calico, Flannel, Cilium)
- Storage plugins (e.g., CSI Node Drivers like AWS EBS CSI, AWS EFS CSI, GCP PD CSI etc)
- Security agents (e.g., Falco runtime security monitors)
- Telemetry collectors (e.g., OpenTelemetry Collector DaemonSet)

Example of Daemonset:

kube-proxy, a critical system component responsible for **Service-to-Pod networking** inside the cluster.

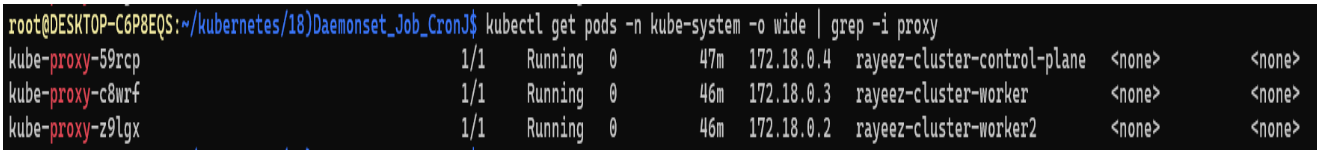
```
kubectl get ds -n kube-system
```



```
root@DESKTOP-C6P8EQS:~/kubernetes/18)Daemonset_Job_CronJ$ kubectl get ds -n kube-system
NAME         DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR            AGE
kindnet      3         3         3       3            3           kubernetes.io/os=linux   10m
kube-proxy   3         3         3       3            3           kubernetes.io/os=linux   10m
```

Kube-proxy is running as daemonset in the K8s cluster.

```
kubectl get pods -n kube-system -o wide | grep -i proxy
```

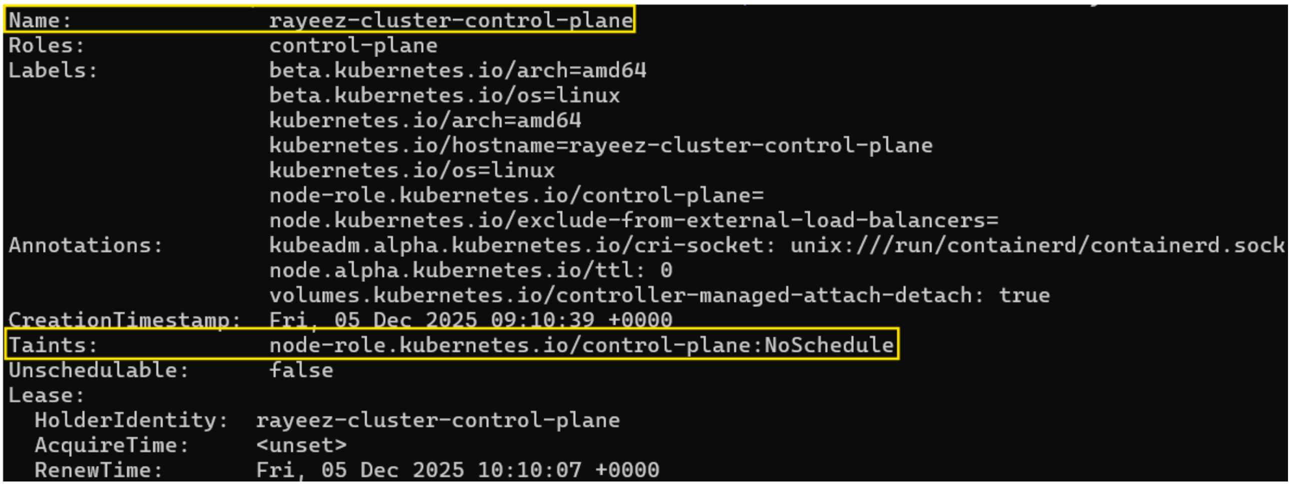


```
root@DESKTOP-C6P8EQS:~/kubernetes/18)Daemonset_Job_CronJ$ kubectl get pods -n kube-system -o wide | grep -i proxy
kube-proxy-59rcp                    1/1     Running   0          47m   172.18.0.4   rayeez-cluster-control-plane   <none>           <none>
kube-proxy-c8wrf                    1/1     Running   0          46m   172.18.0.3   rayeez-cluster-worker          <none>           <none>
kube-proxy-z9lgx                    1/1     Running   0          46m   172.18.0.2   rayeez-cluster-worker2         <none>           <none>
```

But how it was possible to deploy daemonset pod of kube-proxy on the control plane node?

Control plain node is tainted by default as follows in **KIND** cluster:

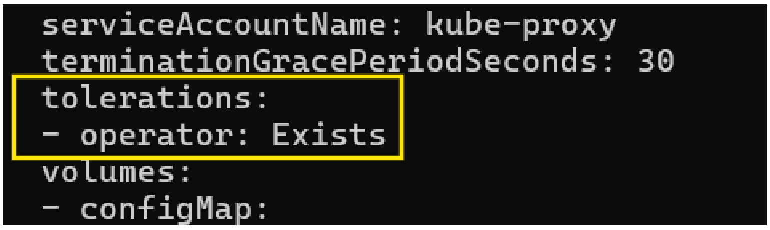
```
kubectl describe nodes rayeez-cluster-control-plane
```



```
Name:                   rayeez-cluster-control-plane
Roles:                  control-plane
Labels:                 beta.kubernetes.io/arch=amd64
                        beta.kubernetes.io/os=linux
                        kubernetes.io/arch=amd64
                        kubernetes.io/hostname=rayeez-cluster-control-plane
                        kubernetes.io/os=linux
                        node-role.kubernetes.io/control-plane=
                        node.kubernetes.io/exclude-from-external-load-balancers=
Annotations:            kubeadm.alpha.kubernetes.io/cri-socket: unix:///run/containerd/containerd.sock
                        node.alpha.kubernetes.io/ttl: 0
                        volumes.kubernetes.io/controller-managed-attach-detach: true
CreationTimestamp:      Fri, 05 Dec 2025 09:10:39 +0000
Taints:                 node-role.kubernetes.io/control-plane:NoSchedule
Unschedulable:          false
Lease:
  HolderIdentity:  rayeez-cluster-control-plane
  AcquireTime:     <unset>
  RenewTime:       Fri, 05 Dec 2025 10:10:07 +0000
```

Control plain node is tainted but even though daemonset pod of kube-proxy is scheduled on it because it has following tolerations:

```
kubectl get ds -n kube-system kube-proxy -o yaml
```

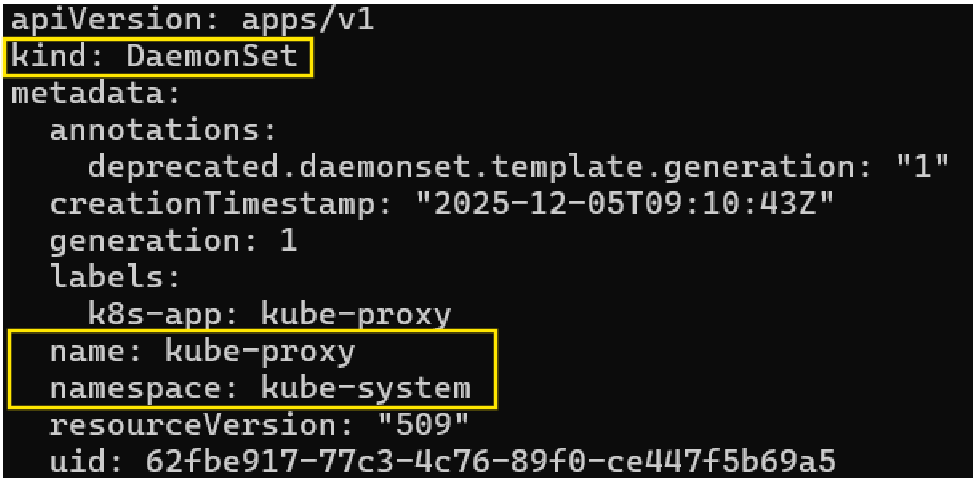


```
  serviceAccountName: kube-proxy
  terminationGracePeriodSeconds: 30
  tolerations:
  - operator: Exists
  volumes:
  - configMap:
```

This means kube-proxy daemonset pod can tolerate any taint, which make it possible to deployed on control plain node.

Note: **System-level DaemonSets are typically deployed into their own dedicated namespace** for better organization and access control.

Kube-proxy daemonsets pods are deployed in kube-system namespace:



```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  annotations:
    deprecated.daemonset.template.generation: "1"
  creationTimestamp: "2025-12-05T09:10:43Z"
  generation: 1
  labels:
    k8s-app: kube-proxy
  name: kube-proxy
  namespace: kube-system
  resourceVersion: "509"
  uid: 62fbe917-77c3-4c76-89f0-ce447f5b69a5
```

Lets Explore the Concept of Daemonset practically!

ds.yaml:

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: log-collector
  namespace: logging-ns
  labels:
    app: log-collector
spec:
  selector:
    matchLabels:
      app: log-collector
  template:
    metadata:
      labels:
        app: log-collector
    spec:
      tolerations:
        - key: "node-
role.kubernetes.io/control-plane"
          operator: "Exists"
          effect: "NoSchedule"
      containers:
        - name: log-collector
```

```
          image: busybox
          command:
            [
              "/bin/sh",
              "-c",
              "while true; do echo 'Collecting
logs...'; sleep 30; done",
            ]
          resources:
            requests:
              cpu: "50m"
              memory: "50Mi"
            limits:
              cpu: "100m"
              memory: "100Mi"
          volumeMounts:
            - name: varlog
              mountPath: /var/log
      volumes:
        - name: varlog
          hostPath:
            path: /var/log
            type: Directory
```

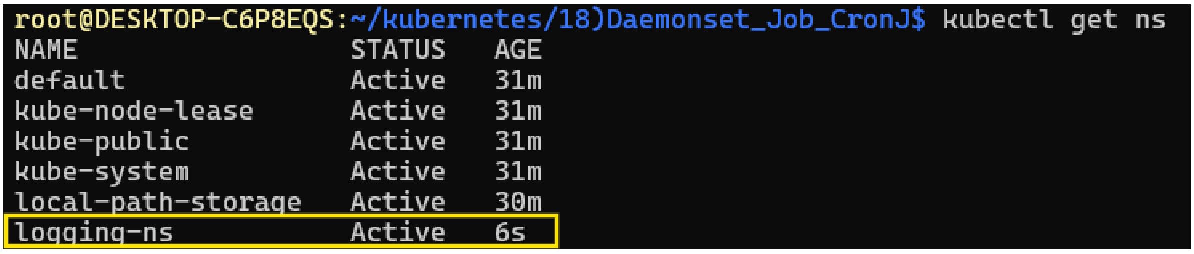
Description of above ds.yaml manifest:

- A Daemonset named log-collector will be deployed in logging-ns namespace.
- This will create and manage pods having label log-collector.
- Daemonset pod is having tolerations matching with Control plane node taint. Hence daemonset pod will also scheduled on control plain node.
- Container inside the pod will run a infinite while loop printing 'Collecting logs... ‘ every 30 seconds.

- CPU, Memory resources requests and limits are also defined for the container to provide proper resources to the daemonset pods and prevent it starving other pods.
- A HostPath volume is also defined and mounted on the container, mapping the node's /var/log directory to the container's /var/log directory to simulate a logging agent's behavior.

As system level Daemonset have their own dedicated namespace, First we have to create the namespace for it:

```
kubectl create ns logging-ns
```



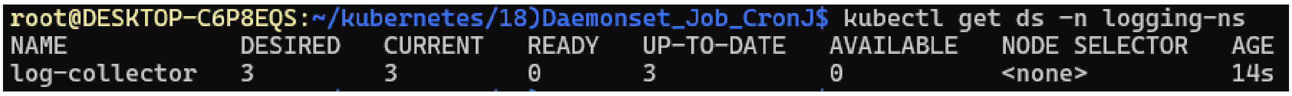
```
root@DESKTOP-C6P8EQS:~/kubernetes/18)Daemonset_Job_CronJ$ kubectl get ns
NAME                  STATUS   AGE
default               Active   31m
kube-node-lease       Active   31m
kube-public           Active   31m
kube-system           Active   31m
local-path-storage    Active   30m
logging-ns            Active   6s
```

Lets apply the ds.yaml file:

```
kubectl apply -f ds.yaml
```

Verify;

```
kubectl get ds -n logging-ns
```



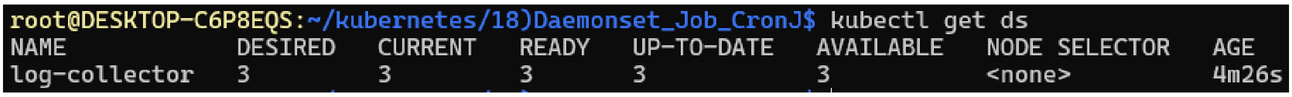
```
root@DESKTOP-C6P8EQS:~/kubernetes/18)Daemonset_Job_CronJ$ kubectl get ds -n logging-ns
NAME            DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR   AGE
log-collector   3         3         0       3            0           <none>          14s
```

Let make logging-ns namespace as default for the current context:

```
kubectl config set-context --current --
namespace=logging-ns
```

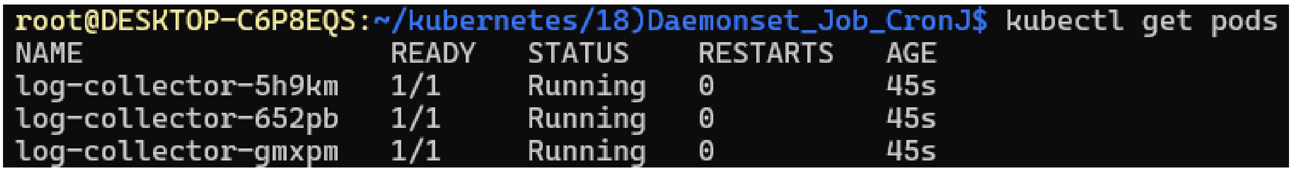
Now I can list out daemonsets for logging-ns namespace directly:

```
kubectl get ds
```



```
root@DESKTOP-C6P8EQS:~/kubernetes/18)Daemonset_Job_CronJ$ kubectl get ds
NAME            DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR   AGE
log-collector   3         3         3       3            3           <none>          4m26s
```

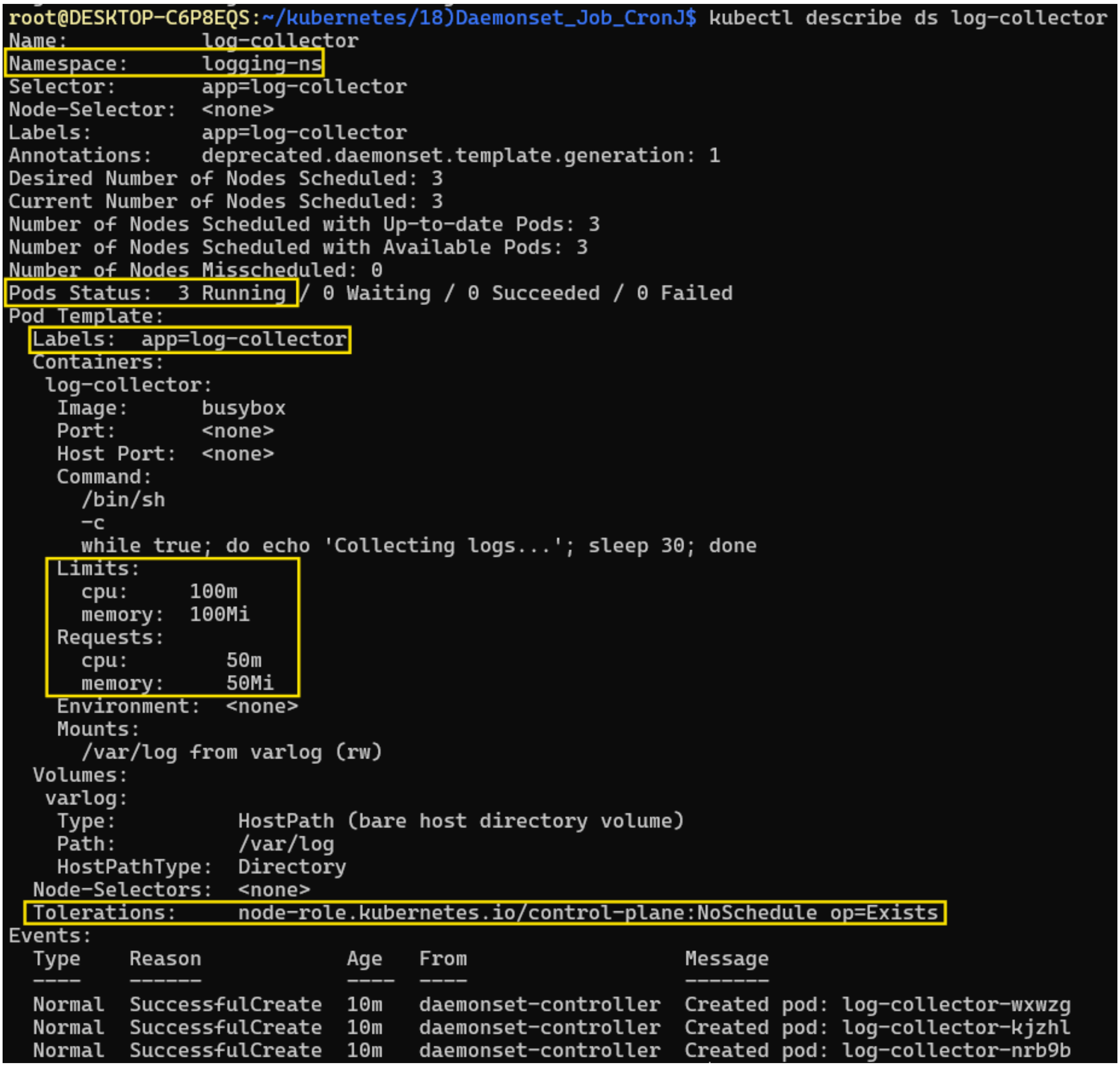
```
kubectl get pods
```



```
root@DESKTOP-C6P8EQS:~/kubernetes/18)Daemonset_Job_CronJ$ kubectl get pods
NAME                   READY   STATUS    RESTARTS   AGE
log-collector-5h9km    1/1     Running   0          45s
log-collector-652pb    1/1     Running   0          45s
log-collector-gmxpm    1/1     Running   0          45s
```

These are the Daemonset pods of log-collector in logging-ns namespace.

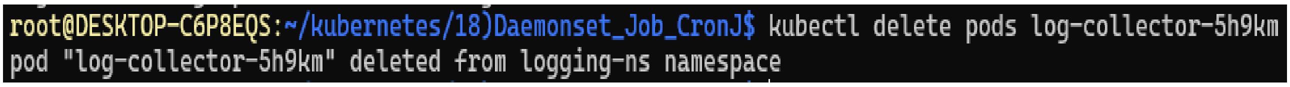
```
kubectl describe ds log-collector
```



```
root@DESKTOP-C6P8EQS:~/kubernetes/18)Daemonset_Job_CronJ$ kubectl describe ds log-collector
Name:           log-collector
Namespace:      logging-ns
Selector:       app=log-collector
Node-Selector:  <none>
Labels:         app=log-collector
Annotations:    deprecated.daemonset.template.generation: 1
Desired Number of Nodes Scheduled: 3
Current Number of Nodes Scheduled: 3
Number of Nodes Scheduled with Up-to-date Pods: 3
Number of Nodes Scheduled with Available Pods: 3
Number of Nodes Misscheduled: 0
Pods Status:  3 Running / 0 Waiting / 0 Succeeded / 0 Failed
Pod Template:
  Labels:  app=log-collector
  Containers:
   log-collector:
    Image:      busybox
    Port:       <none>
    Host Port:  <none>
    Command:
      /bin/sh
      -c
      while true; do echo 'Collecting logs...'; sleep 30; done
    Limits:
      cpu:     100m
      memory:  100Mi
    Requests:
      cpu:        50m
      memory:     50Mi
    Environment:  <none>
    Mounts:
      /var/log from varlog (rw)
  Volumes:
   varlog:
    Type:          HostPath (bare host directory volume)
    Path:          /var/log
    HostPathType:  Directory
  Node-Selectors:  <none>
  Tolerations:     node-role.kubernetes.io/control-plane:NoSchedule op=Exists
Events:
  Type    Reason            Age   From                  Message
  ----    ------            ----  ----                  -------
  Normal  SuccessfulCreate  10m   daemonset-controller  Created pod: log-collector-wxwzg
  Normal  SuccessfulCreate  10m   daemonset-controller  Created pod: log-collector-kjzhl
  Normal  SuccessfulCreate  10m   daemonset-controller  Created pod: log-collector-nrb9b
```

Lets delete one of the daemonset pod and watch it from another terminal:

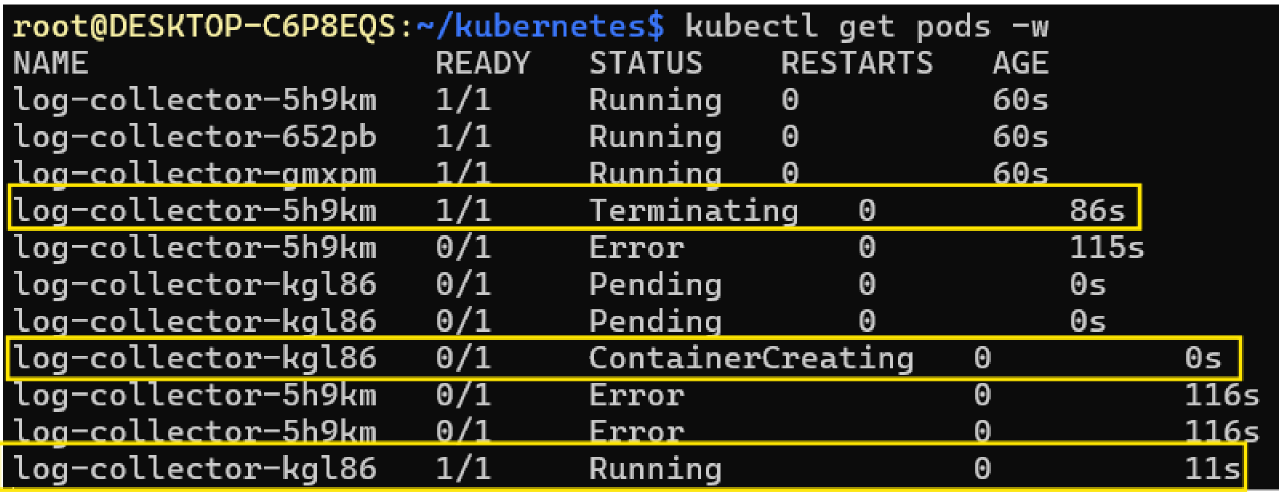
```
kubectl delete pods log-collector-5h9km
```



```
root@DESKTOP-C6P8EQS:~/kubernetes/18)Daemonset_Job_CronJ$ kubectl delete pods log-collector-5h9km
pod "log-collector-5h9km" deleted from logging-ns namespace
```

Watch;

```
kubectl get pods -w
```



```
root@DESKTOP-C6P8EQS:~/kubernetes$ kubectl get pods -w
NAME                  READY   STATUS    RESTARTS   AGE
log-collector-5h9km   1/1     Running   0          60s
log-collector-652pb   1/1     Running   0          60s
log-collector-gmxpm   1/1     Running   0          60s
log-collector-5h9km   1/1     Terminating   0          86s
log-collector-5h9km   0/1     Error         0          115s
log-collector-kgl86   0/1     Pending       0          0s
log-collector-kgl86   0/1     Pending       0          0s
log-collector-kgl86   0/1     ContainerCreating   0          0s
log-collector-5h9km   0/1     Error               0          116s
log-collector-5h9km   0/1     Error               0          116s
log-collector-kgl86   1/1     Running             0          11s
```

Termination of this existing daemonset pod and Creation of new daemonset pod takes place simultaneously. This is performed by Daemonset controller.

JOB:

1. Job is a kubernetes object that runs one-off tasks such as copying a file from location A to location B, ping an IP address such as 8.8.8.8 to check connectivity, Deleting temporary files etc.
2. Once the task finishes successfully (Pod status = Completed), the Job is considered finished.
3. It creates one or more Pods to run a specific task to completion.
4. Jobs are useful for:
    - Running batch jobs, Database migrations, clearing unused files or creating backups reliably and One-time report generation.

Lets Explore Job practically!

job.yaml:

```
apiVersion: batch/v1
kind: Job
metadata:
  name: hello-job
spec:
  ttlSecondsAfterFinished: 60
  completions: 2
  parallelism: 2
  backoffLimit: 4
  template:
    spec:
      containers:
        - name: hello
          image: busybox
          command: ["bin/sh", "-c", "echo Hello
from the Job! && sleep 10"]
      restartPolicy: Never
```

ttlSecondsAfterFinished: 60

This means delete the job 60 seconds after the task is finished(Either successfully or Failed)

completions: 2

This means Job runs 2 pods for the task to be succeed. If 1 Pod has suceessfully completed the and second pod failed the task, Then Job status is marked as failed.

parallelism: 2

This means run 2 pods concurrently, If parallelism is 1, it means run pods sequencially one after another.

backoffLimit: 4

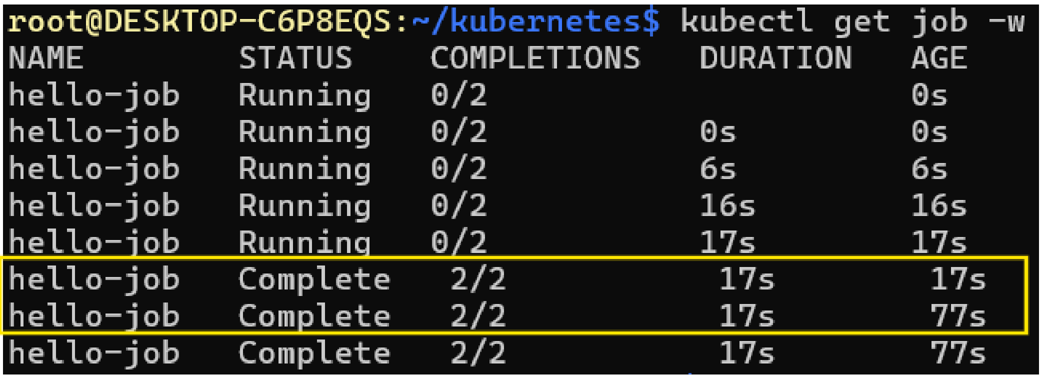
This means If task is failing, recreate the pod till 4 attempts, If it still fails, Job is considered as failed. This recreation of pods is performed by **Job Controller.**

kubelet will restart the container within the same Pod on the same node depending on **restartPolicy** specified.

Container inside each pods runs echo command and sleep for 10 seconds.

Apply the job.yaml manifest:

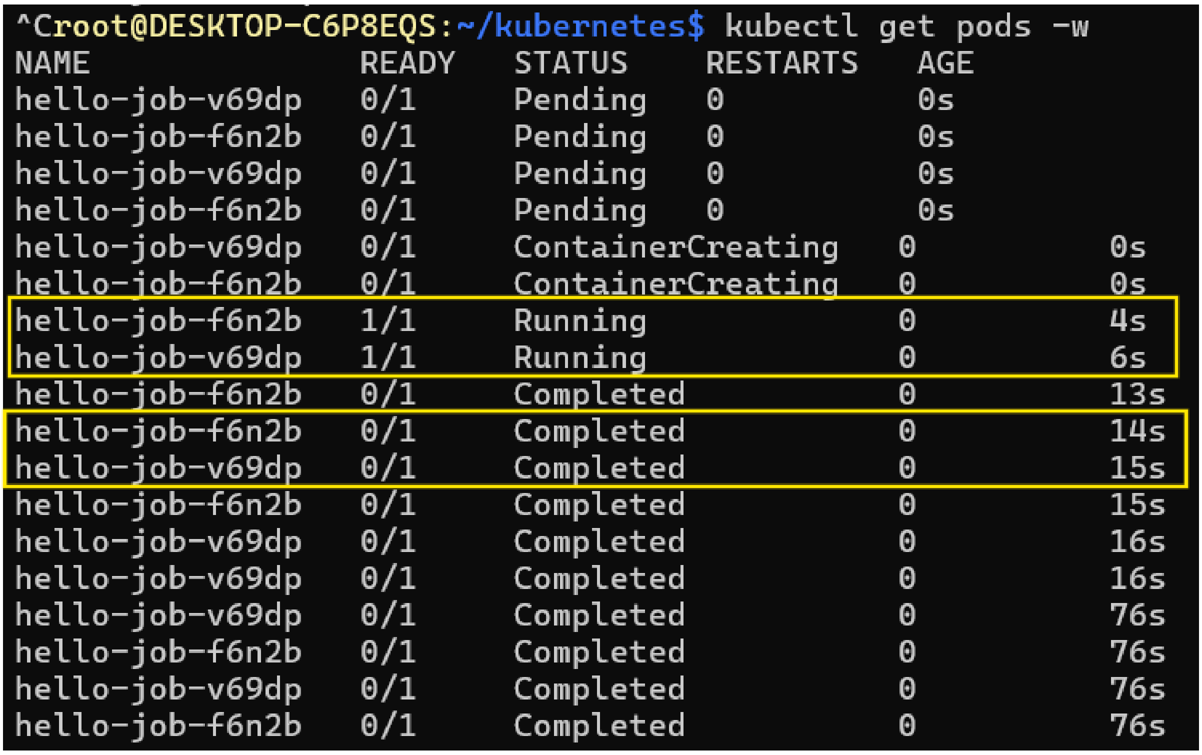
```
kubectl apply -f job.yaml
kubectl get job -w
```



```
root@DESKTOP-C6P8EQS:~/kubernetes$ kubectl get job -w
NAME        STATUS     COMPLETIONS   DURATION   AGE
hello-job   Running    0/2                      0s
hello-job   Running    0/2           0s         0s
hello-job   Running    0/2           6s         6s
hello-job   Running    0/2           16s        16s
hello-job   Running    0/2           17s        17s
hello-job   Complete    2/2           17s        17s
hello-job   Complete    2/2           17s        77s
hello-job   Complete    2/2           17s        77s
```

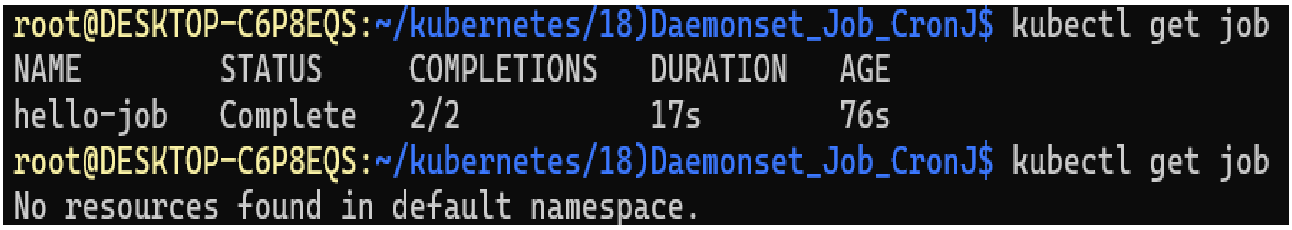
Once the Job is completed suceessfully, It removed after 60 seconds.

```
kubectl get pods -w
```



```
^Croot@DESKTOP-C6P8EQS:~/kubernetes$ kubectl get pods -w
NAME              READY   STATUS    RESTARTS   AGE
hello-job-v69dp   0/1     Pending   0          0s
hello-job-f6n2b   0/1     Pending   0          0s
hello-job-v69dp   0/1     Pending   0          0s
hello-job-f6n2b   0/1     Pending   0          0s
hello-job-v69dp   0/1     ContainerCreating   0          0s
hello-job-f6n2b   0/1     ContainerCreating   0          0s
hello-job-f6n2b   1/1     Running             0          4s
hello-job-v69dp   1/1     Running             0          6s
hello-job-f6n2b   0/1     Completed           0          13s
hello-job-f6n2b   0/1     Completed           0          14s
hello-job-v69dp   0/1     Completed           0          15s
hello-job-f6n2b   0/1     Completed           0          15s
hello-job-v69dp   0/1     Completed           0          16s
hello-job-v69dp   0/1     Completed           0          16s
hello-job-v69dp   0/1     Completed           0          76s
hello-job-f6n2b   0/1     Completed           0          76s
hello-job-v69dp   0/1     Completed           0          76s
hello-job-f6n2b   0/1     Completed           0          76s
```

Pods status converted to from running to completed in 10 seconds and then after 60 seconds Job is removed from the cluster.



```
root@DESKTOP-C6P8EQS:~/kubernetes/18)Daemonset_Job_CronJ$ kubectl get job
NAME        STATUS     COMPLETIONS   DURATION   AGE
hello-job   Complete   2/2           17s        76s
root@DESKTOP-C6P8EQS:~/kubernetes/18)Daemonset_Job_CronJ$ kubectl get job
No resources found in default namespace.
```

CRONJOB:

1. Jobs are scheduled to run on specified time using Cronjob.

```
┌───────────── minute (0 - 59)
│ ┌───────────── hour (0 - 23)
│ │ ┌───────────── day of month (1 - 31)
│ │ │ ┌───────────── month (1 - 12)
│ │ │ │ ┌───────────── day of week (0 - 6)
│ │ │ │ │                         (#0 – Sunday)
* * * * * <command to run>
```

2. At each scheduled time, the CronJob controller automatically creates a new Job object, and that Job is responsible for running one or more Pods to complete the task.
3. Once created, the Job behaves like any other Kubernetes Job — handling retries, backoffLimit, and managing success or failure.

Lets explore the concept of Cronjob practically!

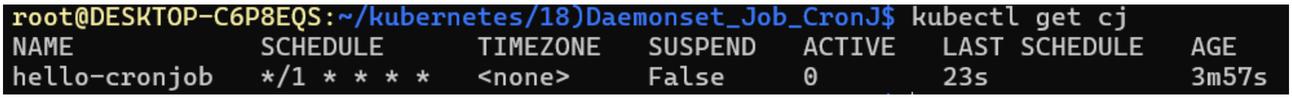
cronjob.yaml:

```
apiVersion: batch/v1
kind: CronJob
metadata:
  name: hello-cronjob
spec:
  schedule: "*/1 * * * *"   # Cron expression
specifying the job to run every 1 minute
  jobTemplate:
    spec:
      backoffLimit: 4
      ttlSecondsAfterFinished: 60
      completions: 2
      parallelism: 2
      template:
        spec:
          containers:
            - name: hello
              image: busybox
```

```
                command: ["bin/sh", "-c", "echo
Hello from CronJob! && sleep 10"]
            restartPolicy: Never
```

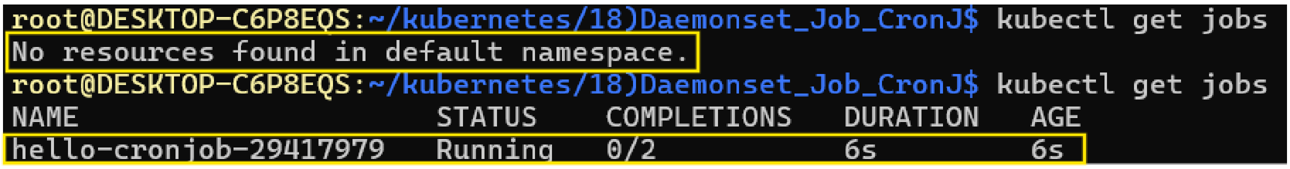
Lets Apply this yaml;

```
kubectl apply -f cronjob.yaml
```



```
root@DESKTOP-C6P8EQS:~/kubernetes/18)Daemonset_Job_CronJ$ kubectl get cj
NAME            SCHEDULE       TIMEZONE    SUSPEND    ACTIVE    LAST SCHEDULE    AGE
hello-cronjob   */1 * * * *    <none>      False      0         23s              3m57s
```

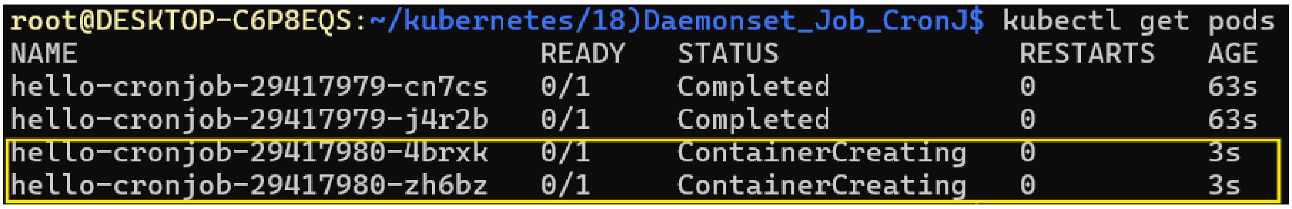
```
kubectl get jobs
```



```
root@DESKTOP-C6P8EQS:~/kubernetes/18)Daemonset_Job_CronJ$ kubectl get jobs
No resources found in default namespace.
root@DESKTOP-C6P8EQS:~/kubernetes/18)Daemonset_Job_CronJ$ kubectl get jobs
NAME                     STATUS     COMPLETIONS    DURATION    AGE
hello-cronjob-29417979   Running    0/2            6s          6s
```

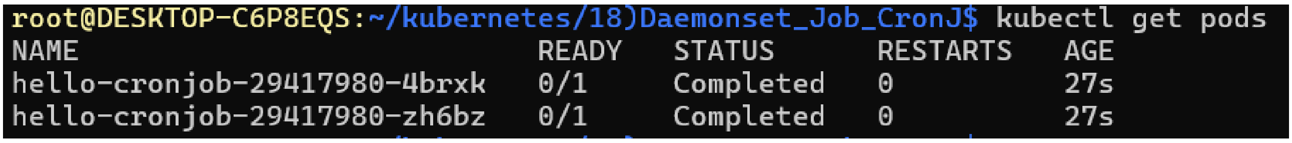
Jobs is scheduled to run every minute.

```
kubectl get pods
```



```
root@DESKTOP-C6P8EQS:~/kubernetes/18)Daemonset_Job_CronJ$ kubectl get pods
NAME                           READY   STATUS              RESTARTS   AGE
hello-cronjob-29417979-cn7cs   0/1     Completed           0          63s
hello-cronjob-29417979-j4r2b   0/1     Completed           0          63s
hello-cronjob-29417980-4brxk   0/1     ContainerCreating   0          3s
hello-cronjob-29417980-zh6bz   0/1     ContainerCreating   0          3s
```

New pods are created after a minute as per the schedule.

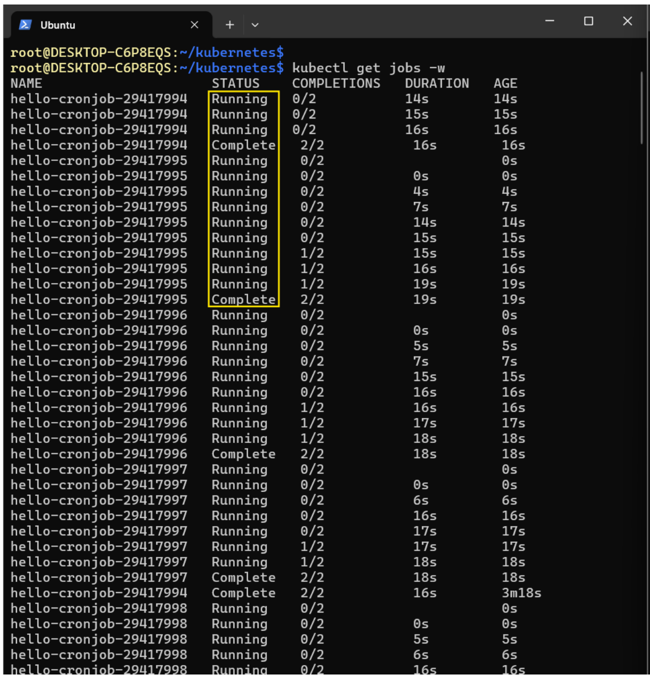


```
root@DESKTOP-C6P8EQS:~/kubernetes/18)Daemonset_Job_CronJ$ kubectl get pods
NAME                           READY   STATUS      RESTARTS   AGE
hello-cronjob-29417980-4brxk   0/1     Completed   0          27s
hello-cronjob-29417980-zh6bz   0/1     Completed   0          27s
```

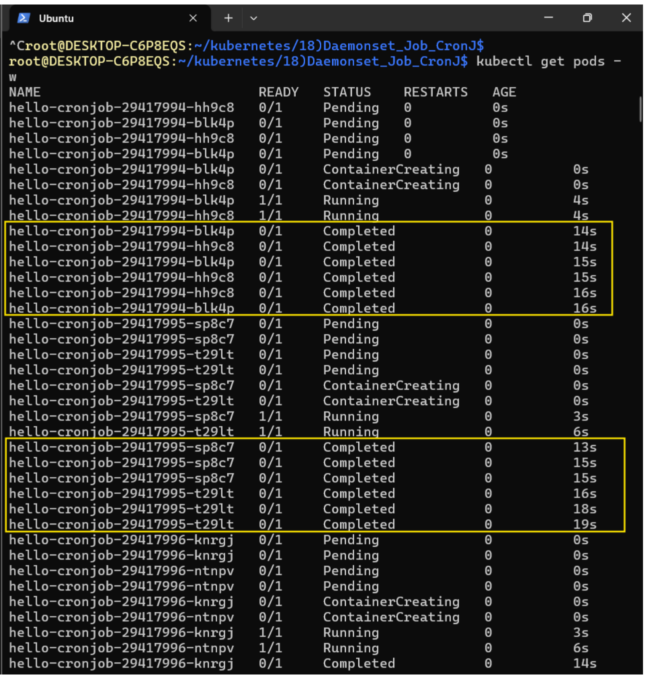
Previous Job is removed as per the TTL seconds timer and pods are garbage collected by K8s.

The CronJob creates a new Job every minute. Once each Job completes—whether it succeeds or fails—it's automatically cleaned up based on the TTL set in the manifest.

Without a TTL value, the CronJob keeps creating Jobs every minute, causing completed Pods to accumulate in the cluster and clutter the environment.



```
root@DESKTOP-C6P8EQS:~/kubernetes$
root@DESKTOP-C6P8EQS:~/kubernetes$ kubectl get jobs -w
NAME                     STATUS    COMPLETIONS   DURATION   AGE
hello-cronjob-29417994   Running   0/2           14s        14s
hello-cronjob-29417994   Running   0/2           15s        15s
hello-cronjob-29417994   Running   0/2           16s        16s
hello-cronjob-29417994   Complete   2/2           16s        16s
hello-cronjob-29417995   Running    0/2                      0s
hello-cronjob-29417995   Running    0/2           0s         0s
hello-cronjob-29417995   Running    0/2           4s         4s
hello-cronjob-29417995   Running    0/2           7s         7s
hello-cronjob-29417995   Running    0/2           14s        14s
hello-cronjob-29417995   Running    0/2           15s        15s
hello-cronjob-29417995   Running    1/2           15s        15s
hello-cronjob-29417995   Running    1/2           16s        16s
hello-cronjob-29417995   Running    1/2           19s        19s
hello-cronjob-29417995   Complete   2/2           19s        19s
hello-cronjob-29417996   Running    0/2                      0s
hello-cronjob-29417996   Running    0/2           0s         0s
hello-cronjob-29417996   Running    0/2           5s         5s
hello-cronjob-29417996   Running    0/2           7s         7s
hello-cronjob-29417996   Running    0/2           15s        15s
hello-cronjob-29417996   Running    0/2           16s        16s
hello-cronjob-29417996   Running    1/2           16s        16s
hello-cronjob-29417996   Running    1/2           17s        17s
hello-cronjob-29417996   Running    1/2           18s        18s
hello-cronjob-29417996   Complete   2/2           18s        18s
hello-cronjob-29417997   Running    0/2                      0s
hello-cronjob-29417997   Running    0/2           0s         0s
hello-cronjob-29417997   Running    0/2           6s         6s
hello-cronjob-29417997   Running    0/2           16s        16s
hello-cronjob-29417997   Running    0/2           17s        17s
hello-cronjob-29417997   Running    1/2           17s        17s
hello-cronjob-29417997   Running    1/2           18s        18s
hello-cronjob-29417997   Complete   2/2           18s        18s
hello-cronjob-29417994   Complete   2/2           16s        3m18s
hello-cronjob-29417998   Running    0/2                      0s
hello-cronjob-29417998   Running    0/2           0s         0s
hello-cronjob-29417998   Running    0/2           5s         5s
hello-cronjob-29417998   Running    0/2           6s         6s
hello-cronjob-29417998   Running    0/2           16s        16s
```



```
^Croot@DESKTOP-C6P8EQS:~/kubernetes/18)Daemonset_Job_CronJ$
root@DESKTOP-C6P8EQS:~/kubernetes/18)Daemonset_Job_CronJ$ kubectl get pods -
w
NAME                            READY   STATUS    RESTARTS   AGE
hello-cronjob-29417994-hh9c8    0/1     Pending   0          0s
hello-cronjob-29417994-blk4p    0/1     Pending   0          0s
hello-cronjob-29417994-hh9c8    0/1     Pending   0          0s
hello-cronjob-29417994-blk4p    0/1     Pending   0          0s
hello-cronjob-29417994-blk4p    0/1     ContainerCreating   0          0s
hello-cronjob-29417994-hh9c8    0/1     ContainerCreating   0          0s
hello-cronjob-29417994-blk4p    1/1     Running             0          4s
hello-cronjob-29417994-hh9c8    1/1     Running             0          4s
hello-cronjob-29417994-blk4p    0/1     Completed           0          14s
hello-cronjob-29417994-hh9c8    0/1     Completed           0          14s
hello-cronjob-29417994-blk4p    0/1     Completed           0          15s
hello-cronjob-29417994-hh9c8    0/1     Completed           0          15s
hello-cronjob-29417994-hh9c8    0/1     Completed           0          16s
hello-cronjob-29417994-blk4p    0/1     Completed           0          16s
hello-cronjob-29417995-sp8c7    0/1     Pending             0          0s
hello-cronjob-29417995-sp8c7    0/1     Pending             0          0s
hello-cronjob-29417995-t29lt    0/1     Pending             0          0s
hello-cronjob-29417995-t29lt    0/1     Pending             0          0s
hello-cronjob-29417995-sp8c7    0/1     ContainerCreating   0          0s
hello-cronjob-29417995-t29lt    0/1     ContainerCreating   0          0s
hello-cronjob-29417995-sp8c7    1/1     Running             0          3s
hello-cronjob-29417995-t29lt    1/1     Running             0          6s
hello-cronjob-29417995-sp8c7    0/1     Completed           0          13s
hello-cronjob-29417995-sp8c7    0/1     Completed           0          15s
hello-cronjob-29417995-sp8c7    0/1     Completed           0          15s
hello-cronjob-29417995-t29lt    0/1     Completed           0          16s
hello-cronjob-29417995-t29lt    0/1     Completed           0          18s
hello-cronjob-29417995-t29lt    0/1     Completed           0          19s
hello-cronjob-29417996-knrgj    0/1     Pending             0          0s
hello-cronjob-29417996-knrgj    0/1     Pending             0          0s
hello-cronjob-29417996-ntnpv    0/1     Pending             0          0s
hello-cronjob-29417996-ntnpv    0/1     Pending             0          0s
hello-cronjob-29417996-knrgj    0/1     ContainerCreating   0          0s
hello-cronjob-29417996-ntnpv    0/1     ContainerCreating   0          0s
hello-cronjob-29417996-knrgj    1/1     Running             0          3s
hello-cronjob-29417996-ntnpv    1/1     Running             0          6s
hello-cronjob-29417996-knrgj    0/1     Completed           0          14s
```