Workloads

Workloads

Different types of pods:

- DaemonSets
- Jobs
- CronJobs

DaemonSets

Ensures a copy of a Pod is running across a set of nodes in a Kubernetes cluster

Typically used for log collectors, monitoring tools etc.

Like ReplicaSets. DaemonSets run exactly 1 copy per node.

Add additional capabilities and features to the Kubernetes cluster itself instead of running normal services.

When Nodes are added to the cluster, pods will be started on those nodes

DaemonSet definition

```
apiVersion: extensions/v1beta1
kind: DaemonSet
metadata:
  name: fluentd
  namespace: kube-system
  labels:
    app: fluentd
spec:
  template:
    metadata:
      labels:
        app: fluentd
    spec:
      containers:
      - name: fluentd
        image: fluent/fluentd:v0.14.10
```

Node selectors with DaemonSets

- If you specify a nodeSelector, the DaemonSet controller will create Pods on nodes which match that node selector.
- Alternatively if you specify a affinity, the Daemonset controller will create Pods on nodes which match the node affinity.
- When neither is specified, DaemonSet controller creates Pods on all nodes

```
spec:
      nodeSelector:
        ssd: "true"
```

Jobs

So far we've looked at long running processes

Jobs are short-lived, typically one-off tasks

A Job creates Pods that run until successful termination

Pods restart even when successful terminated

Job Types

Type	Use case	Behavior	completions	parallelism
One shot	Database migrations	A single pod running once until successful termination	1	1
Parallel fixed completions	Multiple pods processing a set of work in parallel	One or more Pods running one or more times until reaching a fixed completion count	1+	1+
Work queue: parallel Jobs	Multiple pods processing from a centralized work queue	One or more Pods running once until successful termination	1	2+

```
kubectl run -i oneshot \
  --image=<some-image:latest> \
  --restart=OnFailure
```

```
kubectl delete jobs oneshot
```

```
kubectl get pod -a -l job-name=oneshot
```

Describe, edit etc are available

```
apiVersion: batch/v1
kind: Job
metadata:
  name: oneshot
  labels:
    chapter: jobs
spec:
  template:
    metadata:
      labels:
        chapter: jobs
    spec:
      containers:
      - name: my-job-container
        image: <some-image>
        imagePullPolicy: Always
        args:
restartPolicy: OnFailure
```

Parallel jobs

Configure the following variables.

completions: 10

- Number of jobs to run

parallelism: 5

- Max. parallel executions (same time)

```
apiVersion: batch/v1
kind: Job
metadata:
  name: parallel
  labels:
    chapter: jobs
spec:
  parallelism: 5
  completions: 10
  template:
    metadata:
      labels:
        chapter: jobs
    spec:
      containers:
      - name: kuard
        image: <some-image>
        imagePullPolicy: Always
        args:
restartPolicy: OnFailure
```

Cronjobs

A CronJob creates Jobs on a repeating schedule.

CronJob is meant for performing regular scheduled actions such as backups, report generation, and so on.

One CronJob object is like one line of a crontab (cron table) file on a Unix system. It runs a Job periodically on a given schedule, written in Cron format.

Example

```
apiVersion: batch/v1
kind: CronJob
metadata:
  name: hello
spec:
  schedule: "* * * * *"
  jobTemplate:
    spec:
      template:
        spec:
          containers:
            - name: hello
              image: busybox:1.28
              imagePullPolicy: IfNotPresent
              command:
              - /bin/sh
              - -c
              - date; echo Hello from the Kubernetes cluster
              restartPolicy: OnFailure
```

spec.schedule

```
# ┌──────────────── minute (0 - 59)
# │ ┌──────────────── hour (0 - 23)
# │ │ ┌──────────────── day of the month (1 - 31)
# │ │ │ ┌──────────────── month (1 - 12)
# │ │ │ │ ┌──────────────── day of the week (0 - 6) (Sunday to Saturday)
# │ │ │ │ │                                    OR sun, mon, tue, wed, thu, fri, sat
# │ │ │ │ │
# │ │ │ │ │
# * * * * *
```

Extra settings

`.spec.startingDeadlineSeconds`
Set how many seconds a job is allowed to start after the scheduled point in time. If not configured Kubernetes will skip the job and mark it as failed

`.spec.concurrencyPolicy`
specifies how to treat concurrent executions.
Can be set to following:

- Allow (default): allow concurrent executions

- Forbid: skips new job if a previous instance is still running

- Replace: the new job is started and replaced the old one