CronJob

A CronJob creates Jobs on a time-based schedule.

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

> ***Note*** *All CronJob `schedule:` times are denoted in UTC.*

Cron Job Limitations

A cron job creates a job object about once per execution time of its schedule. We say “about” because there are certain circumstances where two jobs might be created, or no job might be created. We attempt to make these rare, but do not completely prevent them. Therefore, jobs should be idempotent.

If `startingDeadlineSeconds` is set to a large value or left unset (the default) and if `concurrencyPolicy` is set to `Allow`, the jobs will always run at least once.

For every CronJob, the CronJob controller checks how many schedules it missed in the duration from its last scheduled time until now. If there are more than 100 missed schedules, then it does not start the job and logs the error

```
Cannot determine if job needs to be started. Too many missed start time (> 100). Set
 or decrease .spec.startingDeadlineSeconds or check clock skew.
```

It is important to note that if the `startingDeadlineSeconds` field is set (not `nil`), the controller counts how many missed jobs occurred from the value of `startingDeadlineSeconds` until now rather than from the last scheduled time until now. For example, if `startingDeadlineSeconds` is `200`, the controller counts how many missed jobs occurred in the last 200 seconds.

A CronJob is counted as missed if it has failed to be created at its scheduled time. For example, If `concurrencyPolicy` is set to `Forbid` and a CronJob was attempted to be scheduled when there was a previous schedule still running, then it would count as missed.

For example, suppose a cron job is set to start at exactly `08:30:00` and its `startingDeadlineSeconds` is set to `10`, if the CronJob controller happens to be down from `08:29:00` to `08:32:00`, the job will not start. Set a longer `startingDeadlineSeconds` if starting later is better than not starting at all.

The Cronjob is only responsible for creating Jobs that match its schedule, and the Job in turn is responsible for the management of the Pods it represents.

Running Automated Tasks with a CronJob

You can use CronJobs to run jobs on a time-based schedule. These automated jobs run like Cron tasks on a Linux or UNIX system.

Each line of a crontab file represents a job, and looks like this:

```
# ┌──────────────── minute (0 - 59)
# │ ┌──────────────── hour (0 - 23)
# │ │ ┌──────────────── day of the month (1 - 31)
# │ │ │ ┌──────────────── month (1 - 12)
# │ │ │ │ ┌──────────────── day of the week (0 - 6) (Sunday to Saturday;
# │ │ │ │ │                                    7 is also Sunday on some systems)
# │ │ │ │ │
# │ │ │ │ │
# * * * * * command to execute
```

Cron jobs are useful for creating periodic and recurring tasks, like running backups or sending emails. Cron jobs can also schedule individual tasks for a specific time, such as if you want to schedule a job for a low activity period.

Before you begin

- You need a working Kubernetes cluster at version >= 1.8

```
kubectl version
```

```
Client Version: version.Info{Major:"1", Minor:"13", GitVersion:"v1.13.1", GitCommit:
"eec55b9ba98609a46fee712359c7b5b365bdd920", GitTreeState:"clean", BuildDate:"2018-12
-13T19:44:19Z", GoVersion:"go1.11.2", Compiler:"gc", Platform:"darwin/amd64"}
Server Version: version.Info{Major:"1", Minor:"10+", GitVersion:"v1.10.9-gke.5", Git
Commit:"d776b4deeb3655fa4b8f4e8e7e4651d00c5f4a98", GitTreeState:"clean", BuildDate:"
2018-11-08T20:33:00Z", GoVersion:"go1.9.3b4", Compiler:"gc", Platform:"linux/amd64"}
```

Creating a Cron Job

Cron jobs require a config file. This example cron job config `.spec` file prints the current time and a hello message every minute:

```
cat << EOF > cronjob.yaml
apiVersion: batch/v1beta1
kind: CronJob
```

```
metadata:
  name: hello
spec:
  schedule: "*/1 * * * *"
  jobTemplate:
    spec:
      template:
        spec:
          containers:
            - name: hello
              image: busybox
              args:
                - /bin/sh
                - -c
                - date; echo Hello from Admatic Kubernetes cluster
          restartPolicy: OnFailure
EOF
```

```
kubectl create -f cronjob.yaml
```

```
cronjob.batch/hello created
```

Alternatively, you can use `kubectl run` to create a cron job without writing a full config:

```
kubectl run hello2 --schedule="*/2 * * * *" --restart=OnFailure --image=busybox -- /
bin/sh -c "date; echo Hello 2 from Admatic Kubernetes cluster"
```

```
kubectl run --generator=cronjob/v1beta1 is DEPRECATED and will be removed in a futur
e version. Use kubectl run --generator=run-pod/v1 or kubectl create instead.
cronjob.batch/hello2 created
```

After creating the cron job, get its status using this command:

```
kubectl get cronjobs
```

```
NAME     SCHEDULE      SUSPEND   ACTIVE   LAST SCHEDULE   AGE
hello    */1 * * * *   False     0        21s             1m
hello2   */2 * * * *   False     0        21s             25s
```

You should see that the cron job “hello” successfully scheduled a job at the time specified in `LAST-SCHEDULE`. There are currently 0 active jobs, meaning that the job has completed or failed.

Watch for the job to be created in around one minute:

```
kubectl get jobs --watch
```

```
NAME                 COMPLETIONS   DURATION   AGE
hello-1546624980     1/1           3s         110s
hello-1546625040     1/1           3s         50s
hello2-1546625040    1/1           2s         50s
hello-1546625100    0/1           0s
hello-1546625100    0/1    0s     0s
hello-1546625100    1/1    2s     2s
hello-1546625160    0/1           0s
hello2-1546625160    0/1           0s
hello-1546625160    0/1    0s     0s
hello2-1546625160    0/1    0s     0s
hello2-1546625160    1/1    2s     2s
hello-1546625160    1/1    2s     2s
```

Now, find the pods that the last scheduled job created and view the standard output of one of the pods.

```
pod=$(kubectl get pods --selector=job-name=hello-1546625160 --output=jsonpath={.item
s..metadata.name})
echo $pod
```

```
hello-1546625160-xhbld
```

```
kubectl logs $pod
```

```
Fri Jan  4 18:06:02 UTC 2019
Hello from Admatic Kubernetes cluster
```

```
kubectl get po
```

```
NAME                     READY   STATUS      RESTARTS   AGE
hello-1546625160-xhbld   0/1     Completed   0          2m
hello-1546625220-wrdqx   0/1     Completed   0          1m
hello-1546625280-zd6hx   0/1     Completed   0          57s
hello2-1546625040-k7xbl  0/1     Completed   0          4m
hello2-1546625160-4p2tf  0/1     Completed   0          2m
hello2-1546625280-ssvcd  0/1     Completed   0          57s
```

Deleting a Cron Job

When you don't need a cron job any more, delete it with `kubectl delete cronjob`:

```
kubectl delete cronjob hello
```

```
cronjob.batch "hello" deleted
```

Deleting the cron job removes all the jobs and pods it created and stops it from creating additional jobs.

```
kubectl get po
```

```
NAME                      READY   STATUS      RESTARTS   AGE
hello2-1546625040-k7xbl   0/1     Completed   0          6m
hello2-1546625160-4p2tf   0/1     Completed   0          4m
hello2-1546625280-ssvcd   0/1     Completed   0          2m
hello2-1546625400-2x8zb   0/1     Completed   0          7s
```