CronJob

A CronJob starts one-time Jobs on a repeating schedule.

FEATURE STATE: `Kubernetes v1.21 [stable]`

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.

CronJobs have limitations and idiosyncrasies. For example, in certain circumstances, a single CronJob can create multiple concurrent Jobs. See the limitations below.

When the control plane creates new Jobs and (indirectly) Pods for a CronJob, the `.metadata.name` of the CronJob is part of the basis for naming those Pods. The name of a CronJob must be a valid DNS subdomain value, but this can produce unexpected results for the Pod hostnames. For best compatibility, the name should follow the more restrictive rules for a DNS label. Even when the name is a DNS subdomain, the name must be no longer than 52 characters. This is because the CronJob controller will automatically append 11 characters to the name you provide and there is a constraint that the length of a Job name is no more than 63 characters.

Example

This example CronJob manifest prints the current time and a hello message every minute:

application/job/cronjob.yaml

```yaml
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
```

(Running Automated Tasks with a CronJob takes you through this example in more detail).

Writing a CronJob spec

Schedule syntax

The `.spec.schedule` field is required. The value of that field follows the Cron syntax:

```
# ┌──────────────── 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)
# │ │ │ │ │                                  OR sun, mon, tue, wed, thu, fri, s
# │ │ │ │ │
# * * * * *
```

For example, `0 0 13 * 5` states that the task must be started every Friday at midnight, as well as on the 13th of each month at midnight.

The format also includes extended "Vixie cron" step values. As explained in the FreeBSD manual:

> Step values can be used in conjunction with ranges. Following a range with `/<number>` specifies skips of the number's value through the range. For example, `0-23/2` can be used in the hours field to specify command execution every other hour (the alternative in the V7 standard is `0,2,4,6,8,10,12,14,16,18,20,22`). Steps are also permitted after an asterisk, so if you want to say "every two hours", just use `*/2`.

> **Note:** A question mark (?) in the schedule has the same meaning as an asterisk *, that is, it stands for any of available value for a given field.

Other than the standard syntax, some macros like `@monthly` can also be used:

Entry	Description	Equivalent to
@yearly (or @annually)	Run once a year at midnight of 1 January	0 0 1 1 *
@monthly	Run once a month at midnight of the first day of the month	0 0 1 * *
@weekly	Run once a week at midnight on Sunday morning	0 0 * * 0
@daily (or @midnight)	Run once a day at midnight	0 0 * * *
@hourly	Run once an hour at the beginning of the hour	0 * * * *

To generate CronJob schedule expressions, you can also use web tools like crontab.guru.

Job template

The `.spec.jobTemplate` defines a template for the Jobs that the CronJob creates, and it is required. It has exactly the same schema as a Job, except that it is nested and does not have an `apiVersion` or `kind`. You can specify common metadata for the templated Jobs, such as labels or annotations. For information about writing a Job `.spec`, see Writing a Job Spec.

Deadline for delayed job start

The `.spec.startingDeadlineSeconds` field is optional. This field defines a deadline (in whole seconds) for starting the Job, if that Job misses its scheduled time for any reason.

After missing the deadline, the CronJob skips that instance of the Job (future occurrences are still scheduled). For example, if you have a backup job that runs twice a day, you might allow it to start up to 8 hours late, but no later, because a backup taken any later wouldn't be useful: you would instead prefer to wait for the next scheduled run.

For Jobs that miss their configured deadline, Kubernetes treats them as failed Jobs. If you don't specify `startingDeadlineSeconds` for a CronJob, the Job occurrences have no deadline.

If the `.spec.startingDeadlineSeconds` field is set (not null), the CronJob controller measures the time between when a job is expected to be created and now. If the difference is higher than that limit, it will skip this execution.

For example, if it is set to `200`, it allows a job to be created for up to 200 seconds after the actual schedule.

Concurrency policy

The `.spec.concurrencyPolicy` field is also optional. It specifies how to treat concurrent executions of a job that is created by this CronJob. The spec may specify only one of the following concurrency policies:

- `Allow` (default): The CronJob allows concurrently running jobs
- `Forbid`: The CronJob does not allow concurrent runs; if it is time for a new job run and the previous job run hasn't finished yet, the CronJob skips the new job run
- `Replace`: If it is time for a new job run and the previous job run hasn't finished yet, the CronJob replaces the currently running job run with a new job run

Note that concurrency policy only applies to the jobs created by the same cron job. If there are multiple CronJobs, their respective jobs are always allowed to run concurrently.

Schedule suspension

You can suspend execution of Jobs for a CronJob, by setting the optional `.spec.suspend` field to true. The field defaults to false.

This setting does *not* affect Jobs that the CronJob has already started.

If you do set that field to true, all subsequent executions are suspended (they remain scheduled, but the CronJob controller does not start the Jobs to run the tasks) until you unsuspend the CronJob.

> **Caution:** Executions that are suspended during their scheduled time count as missed jobs. When `.spec.suspend` changes from `true` to `false` on an existing CronJob without a starting deadline, the missed jobs are scheduled immediately.

Jobs history limits

The `.spec.successfulJobsHistoryLimit` and `.spec.failedJobsHistoryLimit` fields are optional. These fields specify how many completed and failed jobs should be kept. By default, they are set to 3 and 1 respectively. Setting a limit to `0` corresponds to keeping none of the corresponding kind of jobs after they finish.

For another way to clean up jobs automatically, see Clean up finished jobs automatically.

Time zones

FEATURE STATE: `Kubernetes v1.27 [stable]`

For CronJobs with no time zone specified, the kube-controller-manager interprets schedules relative to its local time zone.

You can specify a time zone for a CronJob by setting `.spec.timeZone` to the name of a valid time zone. For example, setting `.spec.timeZone: "Etc/UTC"` instructs Kubernetes to interpret the schedule relative to Coordinated Universal Time.

A time zone database from the Go standard library is included in the binaries and used as a fallback in case an external database is not available on the system.

CronJob limitations

Unsupported TimeZone specification

The implementation of the CronJob API in Kubernetes 1.27 lets you set the `.spec.schedule` field to include a timezone; for example: `CRON_TZ=UTC * * * * *` or `TZ=UTC * * * * *`.

Specifying a timezone that way is **not officially supported** (and never has been).

If you try to set a schedule that includes `TZ` or `CRON_TZ` timezone specification, Kubernetes reports a warning to the client. Future versions of Kubernetes will prevent setting the unofficial timezone mechanism entirely.

Modifying a CronJob

By design, a CronJob contains a template for *new* Jobs. If you modify an existing CronJob, the changes you make will apply to new Jobs that start to run after your modification is complete. Jobs (and their Pods) that have already started continue to run without changes. That is, the CronJob does *not* update existing Jobs, even if those remain running.

Job creation

A CronJob creates a Job object approximately once per execution time of its schedule. The scheduling is approximate because there are certain circumstances where two Jobs might be created, or no Job might be created. Kubernetes tries to avoid those situations, but does not completely prevent them. Therefore, the Jobs that you define 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.

> **Caution:** If `startingDeadlineSeconds` is set to a value less than 10 seconds, the CronJob may not be scheduled. This is because the CronJob controller checks things every 10 seconds.

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).
```

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 CronJob is set to schedule a new Job every one minute beginning at `08:30:00`, and its `startingDeadlineSeconds` field is not set. If the CronJob controller happens to be down from `08:29:00` to `10:21:00`, the job will not start as the number of missed jobs which missed their schedule is greater than 100.

To illustrate this concept further, suppose a CronJob is set to schedule a new Job every one minute beginning at `08:30:00`, and its `startingDeadlineSeconds` is set to 200 seconds. If the CronJob controller happens to be down for the same period as the previous example (`08:29:00` to `10:21:00`,) the Job will still start at 10:22:00. This happens as the controller now checks how many missed schedules happened in the last 200 seconds (i.e., 3 missed schedules), rather than from the last scheduled time until now.

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.

What's next

- Learn about Pods and Jobs, two concepts that CronJobs rely upon.
- Read about the detailed format of CronJob `.spec.schedule` fields.
- For instructions on creating and working with CronJobs, and for an example of a CronJob manifest, see Running automated tasks with CronJobs.
- `CronJob` is part of the Kubernetes REST API. Read the CronJob API reference for more details.

Feedback

Was this page helpful?

Yes No

Last modified October 29, 2022 at 6:33 PM PST: Improve overview for workload APIs (50635afc37)