**Jobs**

**A Job creates one or more Pods and will continue to retry execution of the Pods until a specified number of them successfully terminate**. As pods successfully complete, the Job tracks the successful completions. When a specified number of successful completions is reached, the task (ie, Job) is complete.

**Deleting a Job will clean up the Pods it created**.

Suspending a Job will delete its active Pods until the Job is resumed again.

A simple case is to create one Job object in order to reliably run one Pod to completion.

**The Job object will start a new Pod if the first Pod fails or is deleted (for example due to a node hardware failure or a node reboot).**

If you want to run a Job (either a single task, or several in parallel) on a schedule, see [CronJob](https://kubernetes.io/docs/concepts/workloads/controllers/cron-jobs/).

Running an example Job

Here is an example Job config. It computes π to 2000 places and prints it out. It takes around 10s to complete. [controllers/job.yaml](https://raw.githubusercontent.com/kubernetes/website/main/content/en/examples/controllers/job.yaml)

**apiVersion**: batch/v1

**kind**: Job

**metadata**:

**name**: pi

**spec**:

**template**:

**spec**:

**containers**:

- **name**: pi

**image**: perl:5.34.0

**command**: ["perl", "-Mbignum=bpi", "-wle", "print bpi(2000)"]

**restartPolicy**: Never

**backoffLimit**: 4

 [RestartPolicy](https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle/" \l "restart-policy) equal to Never or OnFailure is allowed

backoffLimit to specify the number of retries before considering a Job as failed

**When a Job completes, no more Pods are created, but the Pods are**[**usually**](https://kubernetes.io/docs/concepts/workloads/controllers/job/#pod-backoff-failure-policy)**not deleted either**. Keeping them around allows you to still view the logs of completed pods to check for errors, warnings, or other diagnostic output. **The job object also remains after it is completed** so that you can view its status. It is up to the user to delete old jobs after noting their status.

**Another way to clean up finished Jobs (either Complete or Failed) automatically is to use a TTL mechanism provided by a**[**TTL controller**](https://kubernetes.io/docs/concepts/workloads/controllers/ttlafterfinished/)**for finished resources, by specifying the .spec.ttlSecondsAfterFinished field of the Job.**

**spec**:

**ttlSecondsAfterFinished**: 100

**To suspend a Job, you can update the .spec.suspend field of the Job to true; later, when you want to resume it again, update it to false. Creating a Job with .spec.suspend set to true will create it in the suspended state.**

**When a Job is resumed from suspension, its .status.startTime field will be reset to the current time.**

**You can also use a Job to run multiple Pods in parallel depends on .Job patterns**

The Job object can be used to support reliable parallel execution of Pods. The Job object is not designed to support closely-communicating parallel processes, as commonly found in scientific computing. **It does support parallel processing of a set of independent but related *work items***.

In a complex system, there may be multiple different sets of work items. Here we are just considering **one set of work items that the user wants to manage together — a *batch job*.**

**CronJob**

**A *CronJob* creates**[**Jobs**](https://kubernetes.io/docs/concepts/workloads/controllers/job/)**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](https://en.wikipedia.org/wiki/Cron) format.

Example

This example CronJob manifest prints the current time and a hello message every minute: [application/job/cronjob.yaml](https://raw.githubusercontent.com/kubernetes/website/main/content/en/examples/application/job/cronjob.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

Schedule syntaxschedule field is required. The value of that field follows the [Cron](https://en.wikipedia.org/wiki/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, sat

# │ │ │ │ │

# \* \* \* \* \*