**About Restart Policy**

A container and thus a Pod may fail for a number of reasons:

1. The container process is exited with a non-zero exit code.
2. The container was killed for exceeding a memory limit (But not CPU Limit).
3. Pod is kicked off the node (node is upgraded, rebooted, deleted, etc.)
4. Node failure or maintenance.

**Container Restart Policy:**

* A container in a Pod can restart independent of the Pod.
* Applies to containers inside a Pod and defined inside the Pod’s Spec.
* Not rescheduled to another Node, but restarted by the Kubelet on that Node.
* Kubelet restarts containers with an exponential backoff, 10s, 20s, 40s capped at 5mins and reset to 0 after 10mins of successful runtime.

apiVersion: v1

kind: Pod

metadata:

  name: demo

  labels:

    purpose: demo

spec:

  containers:

  - name: nginx-container

    image: nginx

  restartPolicy: OnFailure

* **restartPolicy = "Always"** (default) - will restart all containers inside a Pod. This is only valid value when Deployment controller is used.
* **restartPolicy = "OnFailure":** If thecontainer in a Pod fails (non graceful termination), the Pod stays on the node, but the **container is re-run**. Therefore, your program needs to handle the case when it is restarted locally.
* **restartPolicy = "Never":** When a Pod fails, then the Job controller starts a **new Pod**. This means that your application needs to handle the case when it is restarted in a new pod. In particular, it needs to handle temporary files, locks, incomplete output and the like caused by previous runs.

**Jobs**

* Sometimes you want a Pod to execute some work and then stop. You could deploy a Pod spec, but that has limited retry support if the work fails, but you can't use a Deployment because it will replace the Pod if it exits successfully.
* Jobs is a Pod controller which creates a Pod and ensures it **runs to completion**. If the Pod **fails** the Job will start a **replacement**, but when the Pod **succeeds** the Job is **done**.
* Job 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.
* Provide options for running maintenance tasks in your k8s Cluster. Tasks like backing up databases, data migration, data loads, etc.

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

apiVersion: batch/v1

kind: Job

metadata:

  name: hello-job

spec:

  template:

    spec:

      containers:

      - name: hello-con

        image: ubuntu

        command: ["echo", "welcome"]   # Replaces EntryPoint property of Image

        args: ["one", "two"]   # Replaced CMD property of Image.

        imagePullPolicy: IfNotPresent

      restartPolicy: Never

Note: **restartPolicy** - the default Pod restart policy is Always and is not allowed for Jobs; you must specify either **Never** or **OnFailure**.

**Execute the following commands:**

* kubectl apply -f job.yaml
* kubect get pods
* kubectl logs <podname>
* kubectl describe pod <podname>
* kubectl delete -f job.yaml

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

Jobs aren't just for a single task, in some scenarios you want the same Pod to run for a fixed number of times.

apiVersion: batch/v1

kind: Job

metadata:

  name: hello-job

spec:

  completions: 10

  parallelism: 2

  template:

    spec:

      containers:

      - name: hello-con

        image: ubuntu

        command: ["echo", "welcome"]

        imagePullPolicy: IfNotPresent

      restartPolicy: Never

**Pod backoff failure policy**

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) or the container application returns an non-zero exit code.

There are situations where you want to fail a Job after some amount of retries due to a logical error in configuration etc. To do so, set .**spec.backoffLimit** to specify the number of retries before considering a Job as failed. The **back-off limit** is set by default to 6.

**Job.yaml**

apiVersion: batch/v1

kind: Job

metadata:

  name: hello-job

spec:

  backoffLimit: 2

  template:

    spec:

      containers:

      - name: hello-con

        image: ubuntu

        command: ["ls", "/nodir"]

        imagePullPolicy: IfNotPresent

      restartPolicy: Never

The Pod is created but the container will immediately exit, causing the Pod to restart:

Note: You'll see **RunContainerError** statuses & multiple restarts until the Pod goes into **CrashLoopBackoff.**

**Scheduling using CRON Jobs**

* A CronJob creates [Jobs](https://kubernetes.io/docs/concepts/workloads/controllers/job/) on a repeating 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](https://en.wikipedia.org/wiki/Cron) format.
* Jobs are not automatically cleared up so you can work with the Pods and see the logs.
* <https://crontab.cronhub.io/>

Diagram

Description automatically generated

apiVersion: batch/v1

kind: CronJob

metadata:

  name: hello-job

**spec**:

  schedule: "\* \* \* \* \*"

  jobTemplate:

**spec**:

backoffLimit: 2

      template:

**spec**:

          containers:

          - name: hello-con

            image: ubuntu

            command: ["echo", "welcome"]

          restartPolicy: Never