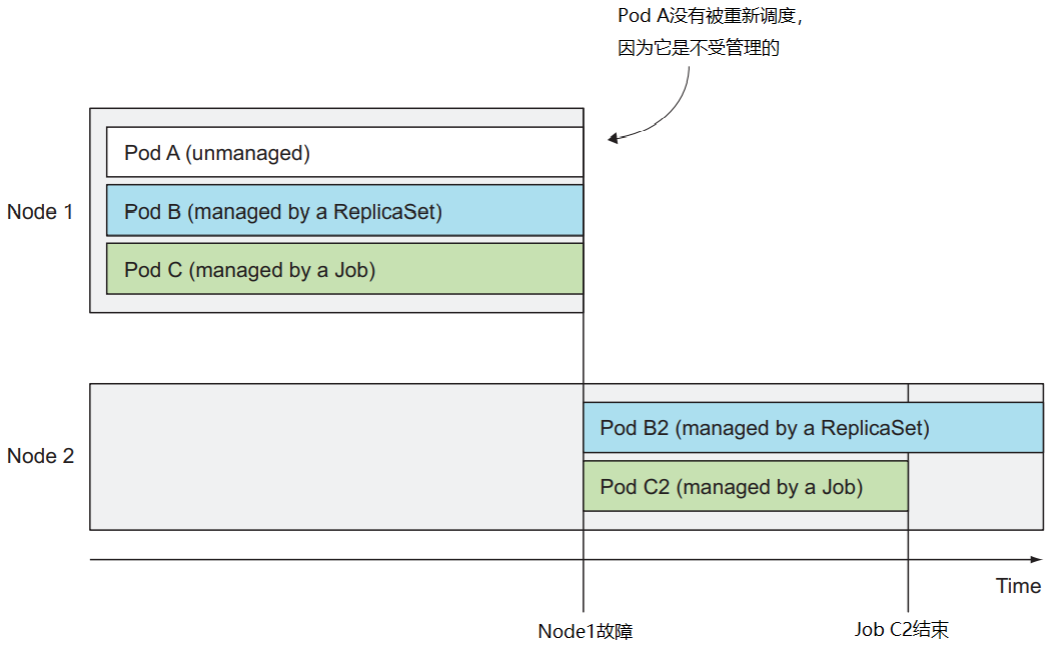
23.Job和CronJob

在之前的章节中我们提到的pod都是需要持续运行的pod。但有时候我们也会希望运行一个在处理完成后就结束的任务，不像ReplicationController、ReplicaSet以及DaemonSet运行的是持续的、不会结束的任务，而且它们管理的pod中的进程退出时就会重启。但是对于一个可完成的任务来说，其进程结束后不应该再重启。

什么是Job

Kubernetes中的Job资源使我们能够运行一个pod，并且该pod中的容器不会随着其中的进程运行完成而重启，一旦进程结束，pod就会被认为是完成状态。
当某个节点发生故障时，位于该节点上由Job管理的Pod就会被重新调度到另一个节点，就像ReplicaSet管理的pod一样。当进程本身出问题时（当进程返回一个错误退出码时），可以配置Job是否重启容器。
下图展示了一个由Job创建的pod在节点发生故障时是如何被重新调度到一个新节点上的。该图还显示了一个不受管控的pod，当节点故障时它不会被重新调度，以及一个受ReplicaSet管控的pod，节点故障时，它会被重新调度：



Job对于某些特殊任务来说很有用，这些任务能否正常结束至关重要。可以在一个不受管控的pod中运行这个任务并等待它结束，但是当节点出故障或者pod在执行任务的时候从节点上被逐出时，就需要手动创建这个任务。手动创建并不合理，特别是在任务需要几个小时才能完成的情况下。

创建Job

在创建Job之前，我们先构建一个新的镜像。

下面我们基于busbox镜像创建一个名为batch-job的容器镜像并将其发布到阿里云镜像仓库，该镜像了每隔两分钟调用一次sleep命令，Dockerfile内容如下：

FROM busybox

ENTRYPOINT echo "$(date) Batch job starting"; sleep 120; echo "$(date) Finished succesfully"

```
FROM busybox
ENTRYPOINT echo "$(date) Batch job starting"; sleep 120; echo "$(date) Finished succesfully"
~
```

镜像的具体构建和发布过程此处不再赘述。

制作并发布的镜像名为：registry.cn-shanghai.aliyuncs.com/david-ns01/batch-job:1.0

创建Job的YAML定义文件batch-job.yaml：

apiVersion: batch/v1

kind: Job

metadata:

  name: batch-job

spec:

  template:

```
  metadata:
    labels:
      app: batch-job
  spec:
    restartPolicy: OnFailure
    containers:
    - name: main
      image: registry.cn-shanghai.aliyuncs.com/david-ns01/batch-job:1.0
```

```
apiVersion: batch/v1
kind: Job
metadata:
  name: batch-job
spec:
  template:
    metadata:
      labels:
        app: batch-job
    spec:
      restartPolicy: OnFailure
      containers:
      - name: main
        image: registry.cn-shanghai.aliyuncs.com/david-ns01/batch-job:1.0
~
```

该文件定义了一个类型为Job的资源，它会运行指定的batch-job镜像，该镜像调用一个运行120秒的进程，然后退出。

在pod定义信息中的spec区域中，我们可以指定当运行在这个容器中的进程结束后Kubernetes应该采取的操作，通过pod的spec区域的restartPolicy属性（默认为Always）可以达到这个目的。Job所管理的pod不能使用默认策略，因为设计Job的目的不是为了无限期地运行。因此我们需要明确地设置restartPolicy为OnFailure或者Never，这样可以防止容器在完成任务时重新启动。

执行如下命令创建Job：

kubectl create –f batch-job.

```
[david@dhr-demo ~]$ kubectl create -f batch-job.yaml
job.batch/batch-job created
[david@dhr-demo ~]$ kubectl get jobs
NAME        COMPLETIONS   DURATION   AGE
batch-job   0/1           26s        27s
[david@dhr-demo ~]$ kubectl get po
NAME              READY   STATUS    RESTARTS   AGE
batch-job-fjjv5   1/1     Running   0          35s
[david@dhr-demo ~]$
```

从上图可以看到Job为我们自动创建并启动了一个pod。

2分钟过后再次执行：

kubectl get po

```
[david@dhr-demo ~]$ kubectl get po
NAME              READY   STATUS      RESTARTS   AGE
batch-job-fjjv5   0/1     Completed   0          2m23s
```

可以看到pod的状态变为Completed。

执行kubectl get jobs查看Job的状态:

```
[david@dhr-demo ~]$ kubectl get jobs
NAME        COMPLETIONS   DURATION   AGE
batch-job   1/1           2m10s      13m
[david@dhr-demo ~]$
```

Job的COMPLETETION列由0/1变为1/1。

Pod在任务完成后没有被自动删除掉，这样我们就可以查看pod的日志：

kubectl logs batch-job-fjjv5

```
[david@dhr-demo ~]$ kubectl logs batch-job-fjjv5
Thu Dec 10 08:29:39 UTC 2020 Batch job starting
Thu Dec 10 08:31:39 UTC 2020 Finished succesfully
[david@dhr-demo ~]$
```

如果删除这个Job，由它创建的pod也会被删除掉：

kubectl delete job batch-job

```
[david@dhr-demo ~]$ kubectl delete job batch-job
job.batch "batch-job" deleted
[david@dhr-demo ~]$ kubectl get jobs
No resources found in default namespace.
[david@dhr-demo ~]$ kubectl get po
No resources found in default namespace.
[david@dhr-demo ~]$
```

在一个Job中运行多个pod实例

我们可以配置Job创建多个pod实例，然后以并行或者串行的方式执行它们。可以在Job的Spec区段中设置completions和parallelism属性达到这个目的。

以串行方式运行Job pod

如果需要Job运行多次，可以设置completions为期望pod运行的次数，例如定义如下一个YAML文件：

```
apiVersion: batch/v1
kind: Job
metadata:
  name: multi-completion-batch-job
spec:
  completions: 5
  template:
    metadata:
      labels:
        app: batch-job
    spec:
      restartPolicy: OnFailure
      containers:
      - name: main
        image: registry.cn-shanghai.aliyuncs.com/david-ns01/batch-job:1.0
```

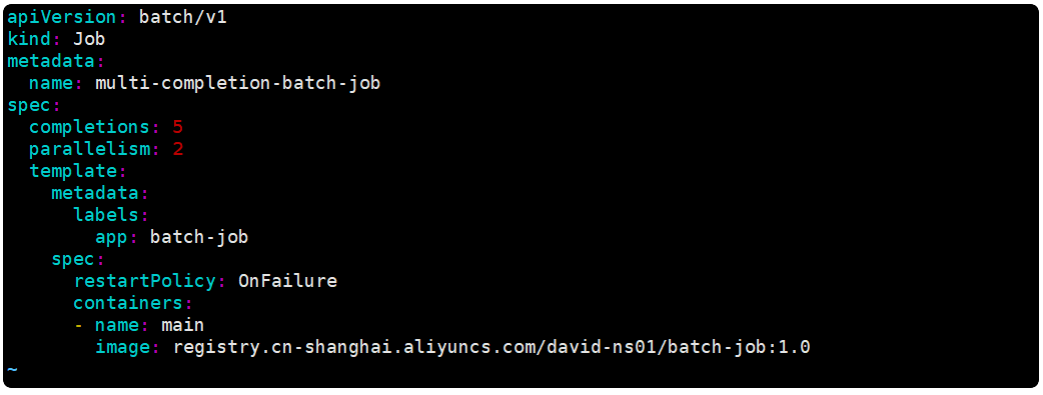
基于该文件创建的Job会以串行的方式前后运行5个pod。它最初创建一个pod，当pod的容器运行完成时，又会创建第二个pod，以此类推，直到五个pod都成功运行完成。如果某一个pod发生故障了，Job就会创建一个新的pod，因此Job创建的pod总数可能不止5个。

以并方式运行Job pod

与一个接一个的运行单个pod不同，我们还可以让Job以并行的方式运行多个pod。通过Job Spec区段中的parallelism属性，可以指定允许以并行的方式运行的pod数。如下面的YAML文件所示：

vim multi-completion-parallel-batch-job.yaml

```
apiVersion: batch/v1
kind: Job
metadata:
  name: multi-completion-batch-job
spec:
  completions: 5
  parallelism: 2
  template:
    metadata:
      labels:
        app: batch-job
    spec:
      restartPolicy: OnFailure
      containers:
      - name: main
        image: registry.cn-shanghai.aliyuncs.com/david-ns01/batch-job:1.0
```



通过设置parallelism为2，Job会创建两个pod然后以并行的方式运行它们。

执行：

kubectl create -f multi-completion-parallel-batch-job.yaml

```
[david@dhr-demo ~]$ kubectl create -f multi-completion-parallel-batch-job.yaml
job.batch/multi-completion-batch-job created
[david@dhr-demo ~]$
```

```
[david@dhr-demo ~]$ kubectl get jobs
NAME                        COMPLETIONS   DURATION   AGE
multi-completion-batch-job   0/5           46s        46s
[david@dhr-demo ~]$ kubectl get po
NAME                              READY   STATUS    RESTARTS   AGE
multi-completion-batch-job-b585t   1/1     Running   0          55s
multi-completion-batch-job-z29vb   1/1     Running   0          55s
[david@dhr-demo ~]$
```

一旦其中一个运行完成，Job就会创建并运行下一个Pod，直到5个pod成功运行完成

```
[david@dhr-demo ~]$ kubectl get jobs
NAME                        COMPLETIONS   DURATION   AGE
multi-completion-batch-job   5/5           6m21s      38m
[david@dhr-demo ~]$
```

```
[david@dhr-demo ~]$ kubectl get po
NAME                              READY   STATUS      RESTARTS   AGE
multi-completion-batch-job-4xbvv   0/1     Completed   0          34m
multi-completion-batch-job-b585t   0/1     Completed   0          37m
multi-completion-batch-job-gmgt2   0/1     Completed   0          34m
multi-completion-batch-job-xmpvd   0/1     Completed   0          32m
multi-completion-batch-job-z29vb   0/1     Completed   0          37m
```

缩放Job

在旧版本中我们甚至可以在Job运行的过程中更改Job的parallelism属性。这点与ReplicaSet和ReplicationController的缩放操作类似，可以通过kubectl scale命令完成：

kubectl scale job multi-completion-batch-job --replicas 3

但是在最新版本中kubectl scale job已经被废弃掉了。

限制Job pod完成任务的时间

请思考如下问题：

1.Job需要等待一个pod多长时间完成任务?

2.如果pod卡住了并且根本无法完成任务或者不能足够快地完成任务怎么办?

可以在pod spec区段通过设置activeDeadlineSeconds属性来限制pod的时间。如果pod的运行时间超过了这个值，系统就会尝试终止pod，并将Job标记为失败。

在Job的定义文件中，我们可以通过设置spec.backoffLimit属性来配置Job在被标记为失败之前可以重试的次数。该属性值默认为6。

调度Job

当创建Job资源后，Job就会立即运行对应的Pod。但是很多批处理任务需要在特定的时间运行或者以特定的时间间隔重复地运行。在Linux和类Unix操作系统中，这些任务通常被称为cron任务。Kubernetes也支持这种任务。

在Kubernetes中可以通过创建CronJob资源来配置cron任务。运行任务的调度信息以cron格式指定，因此如果熟悉常规的cron任务的话，就能很快理解Kubernetes的CronJob了。

Kubernetes会根据配置在CronJob定义信息中的Job模板来创建Job资源。当创建好Job时，就会根据Job的pod模板创建并启动一个或者多个pod副本。

创建CronJob

假设我们希望每隔15分钟运行一次上面提到的批处理Job，可以创建一个CronJob资源：

```
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: batch-job-every-five-minutes
spec:
  schedule: "0,15,30,45 * * * *"
  jobTemplate:
    spec:
      template:
        metadata:
          labels:
            app: periodic-batch-job
        spec:
          restartPolicy: OnFailure
          containers:
          - name: main
            image: registry.cn-shanghai.aliyuncs.com/david-ns01/batch-job:1.0
```

执行如下命令创建CronJob：

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

查看创建的CronJob：

```
kubectl get cronjob
```

```
[david@dhr-demo ~]$ kubectl create -f cronjob.yaml
cronjob.batch/batch-job-every-five-minutes created
[david@dhr-demo ~]$ kubectl get cronjob
NAME                          SCHEDULE            SUSPEND   ACTIVE   LAST SCHEDULE   AGE
batch-job-every-five-minutes   0,15,30,45 * * * *   False     0        <none>          11s
[david@dhr-demo ~]$
```

```
kubectl get jobs
```

```
[david@dhr-demo root]$ kubectl get pods
No resources found in default namespace.
[david@dhr-demo root]$
```

```
kubectl get po
```

```
[david@dhr-demo root]$ kubectl get po
No resources found in default namespace.
[david@dhr-demo root]$
```

可以看到还未创建Job和Pod。

当当前时间处于某一个小时内的0分、15分、30分或者45分时，可以看到创建了CronJob、Job以及pod：

```
[david@dhr-demo ~]$ kubectl get jobs
No resources found in default namespace.
[david@dhr-demo ~]$ kubectl get jobs
NAME                                     COMPLETIONS   DURATION   AGE
batch-job-every-five-minutes-1607845500   0/1           0s         0s
[david@dhr-demo ~]$ kubectl get jobs
```

```
[david@dhr-demo ~]$ kubectl get pod
NAME                                           READY   STATUS    RESTARTS   AGE
batch-job-every-five-minutes-1607845500-n5v4c   1/1     Running   0          29s
[david@dhr-demo ~]$
```

```
[david@dhr-demo ~]$ kubectl get cronjob
NAME                          SCHEDULE            SUSPEND   ACTIVE   LAST SCHEDULE   AGE
batch-job-every-five-minutes   0,15,30,45 * * * *   False     0        2m54s           13m
[david@dhr-demo ~]$
```

上面的Job在CronJob被创建后立即被创建。但是实际情况可能比较复杂，导致在CronJob被创建后，Job或者pod过一段时间之后才被创建。如果对Job的启动时间不能太晚于调度时间有硬性的要求，可以通过在CronJob的spec中设定startingDeadlineSeconds来指定一个期限：

```
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: batch-job-every-fifteen-minutes
spec:
  schedule: "0,15,30,45 * * * *"
  startingDeadlineSeconds: 15
  ......
```

该文件指定的Job按理应该在每个小时的0分、15分、30分、45分运行，例如应该在10:30:00执行，但是如果到了10:30:15由于某些原因还未执行，该Job就不会运行并被标记为Failed。

Cron表达式

Cron表达式由五部分组成：

- Minute (某分钟或者每分钟)
- Hour (某小时或者每小时)
- Day of month (每月的第几天或者每一天)
- Month (某月或者每月)

- Day of week (每周的每一天或者某一天)