Kubernetes 批处理调度与定时调度

前言：

课程名称：Kubernetes 批处理调度和定时调度

实验环境：

本章节 Kubernetes 集群环境如下：

角色	IP	主机名	组件	硬件
控制节点	192.168.128.11	k8s-master01	apiserver controller-manager scheduler etcd containerd	CPU：4vCPU 硬盘：100G 内存：4GB 开启虚拟化
工作节点	192.168.128.21	k8s-node01	kubelet kube-proxy containerd calico coredns	CPU：6vCPU 硬盘：100G 内存：6GB 开启虚拟化

张岩峰老师微信，加我微信，邀请你加入 VIP 交流答疑群：

微信号：ZhangYanFeng0429

二维码：

1、Job：批处理调度

官方文档：
https://kubernetes.io/zh/docs/concepts/workloads/controllers/job/

1.1、什么是 Job？

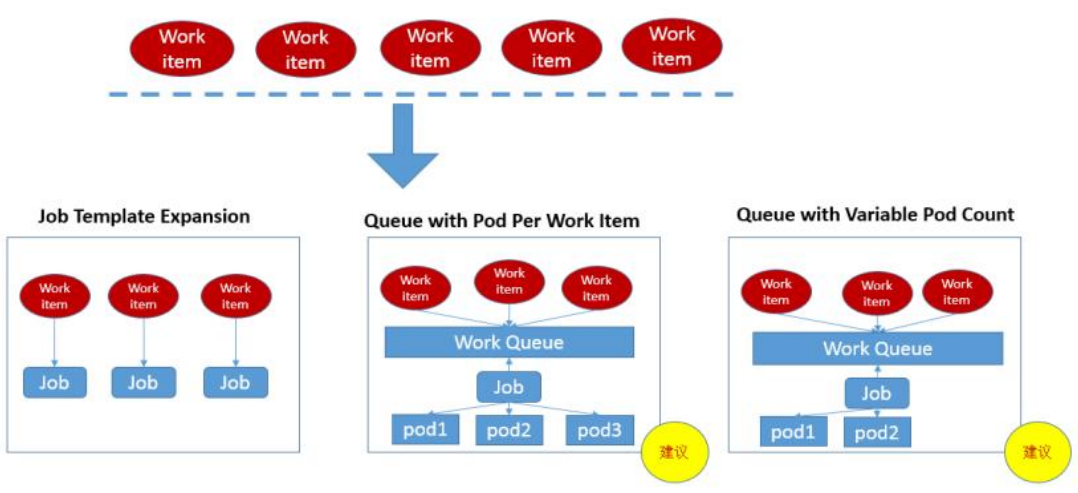
Kubernetes jobs 主要是针对短时和批量的工作负载。它是为了结束而运行的，而不是像 deployment、replicasets、DaemonSets 等其他对象那样持续运行。

我们可以通过 Kubernetes Job 资源对象来定义并启动一个批处理任务。批处理任务通常并行（或者串行）启动多个计算进程去处理一批工作项（work item），处理完成后，整个批处理任务结束。

按照批处理任务实现方式的不同，批处理任务可以分为如下的几种模式。

- Job Template Expansion 模式：一个 Job 对象对应一个待处理的 Work item，有几个 Work item 就产生几个独立的 Job，通常适合 Work item 数量少、每个 Work item 要处理的数据量比较大的场景。
- Queue with Pod Per Work Item 模式：采用一个任务队列存放 Work item，一个 Job 对象作为消费者去完成这些 Work item，在这种模式下，Job 会启动 N 个 Pod，每个 Pod 都对应一个 Work item。
- Queue with Variable Pod Count 模式：也是采用一个任务队列存放 Work item，一个 Job 对象作为消费者去完成这些 Work item，但与上面的模式不同，Job 启动的 Pod 数量是可变的。

批处理任务的几种模式，如下图所示：



还有一种被称为 Single Job with Static Work Assignment 的模式，也是一个 Job 产生多个 Pod，但它采用程序静态方式分配任务项，而不是采用队列模式进行动态分配。

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

批处理任务的模式对比：

模式名称	是否是一个Job	Pod的数量少于Work Item	用户程序是否要做响应的修改	k8s是否支持
Job Template Expansion	/	/	是	是
Qucue with Pod Per Work Item	是	/	有时候需要	是
Qucue with Variable Pod Count	是	/	/	是
Single Job with Work Assignment	是	/	是	/

考虑到批处理的并行问题，Kubernetes 将 Job 分以下三种类型。

1、非并行 Job

通常一个 Job 只启动一个 Pod，除非 Pod 异常，才会重启该 Pod，一旦此 Pod 正常结束，Job 将结束。

2、具有确定完成计数的并行 job

并行 Job 会启动多个 Pod，此时需要设定 Job 的.spec.completions 参数为一个正数，当正常结束的 Pod 数量达至此参数设定的值后，Job 结束。此外，Job 的.spec.parallelism 参数用来控制并行度，即同时启动几个 Job 来处理 Work Item。

3、带工作队列的并行 job

任务队列方式的并行 Job 需要一个独立的 Queue，Work item 都在一个 Queue 中存放，不能设置 Job 的.spec.completions 参数，此时 Job 有以下特性。

- 多个 Pod 之间必须相互协调，或者借助外部服务确定每个 Pod 要处理哪个工作条目。例如，任一 Pod 都可以从工作队列中取走最多 N 个工作条目。
- 每个 Pod 都可以独立确定是否其它 Pod 都已完成，进而确定 Job 是否完成。
- 如果某个 Pod 正常结束，则 Job 不会再启动新的 Pod。
- 如果一个 Pod 成功结束，并且所有 Pod 都已终止，则整个 Job 成功结束。
- 一旦任何 Pod 成功退出，任何其它 Pod 都不应再对此任务执行任何操作或生成任何输出。所有 Pod 都应启动退出过程。

说明：Job 资源清单定义和 Pod 类似，这里就不做太多介绍了。以案例带过。

1.2、Job Template Expansion 模式

Job Template Expansion 模式，由于在这种模式下每个 Work item 对应一个 Job 实例，所以这种模式首先定义一个 Job 模板，模板里的主要参数是 Work item 的标识，因为每个 Job 都处理不同的 Work item。

1、创建 job 文件

（1）创建资源清单存放位置

```
[root@k8s-master01 ~]# mkdir job
[root@k8s-master01 ~]# cd job/
```

（2）创建资源清单模板文件

```
[root@k8s-master01 ~]# mkdir /etc/kubernetes/yaml/job/
[root@k8s-master01 ~]# cd /etc/kubernetes/yaml/job/
[root@k8s-master01 job]# vi job.txt
apiVersion: batch/v1
kind: Job
metadata:
  name: process-item-$ITEM
  labels:
    jobgroup: jobexample
spec:
  template:
    metadata:
      name: jobexample
      labels:
        jobgroup: jobexample
    spec:
      containers:
      - name: c
        image: busybox
        command: ["sh", "-c", "echo Processing item $ITEM && sleep 60"]
      restartPolicy: Never
```

2、生成 3 个对应的 Job 定义文件并创建 Job

（1）批量替换，生成 yaml 资源清单文件

```
[root@k8s-master01 job]# for i in apple banana cherry
do
  cat job.txt | sed "s/\$ITEM/$i/" > /root/job/job-$i.yaml
done

[root@k8s-master01 job]# ll
total 16
-rw-r--r-- 1 root root 367 Aug 29 05:02 job-apple.yaml
-rw-r--r-- 1 root root 369 Aug 29 05:02 job-banana.yaml
-rw-r--r-- 1 root root 369 Aug 29 05:02 job-cherry.yaml
-rw-r--r-- 1 root root 367 Aug 29 04:57 job.txt
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

（2）更新资源清单文件

```
[root@k8s-master01 job]# kubectl apply -f .
job.batch/process-item-apple created
job.batch/process-item-banana created
job.batch/process-item-cherry created
```

（3）查看 pod 和 job

```
# 查看生成的 pods 和 Jobs，此时任务还没执行结束，所以 pod 是 running 状态，Job 的 COMPLETIONS 是 0/1。
[root@k8s-master01 job]# kubectl get pods
NAME                         READY   STATUS    RESTARTS   AGE
process-item-apple-jtxgh     1/1     Running   0          60s
process-item-banana-6t2vl    1/1     Running   0          60s
process-item-cherry-fm4fr    1/1     Running   0          60s

[root@k8s-master01 job]# kubectl get jobs
NAME                  COMPLETIONS   DURATION   AGE
process-item-apple    0/1           62s        62s
process-item-banana   0/1           62s        62s
process-item-cherry   0/1           62s        62s
```

（4）一分钟后，再查看 pod 和 job：

```
# 生成的 pods 完成了相应的工作，所以一旦此 Pod 正常结束，Job 将结束
[root@k8s-master01 job]# kubectl get pods
NAME                         READY   STATUS      RESTARTS   AGE
process-item-apple-jtxgh     0/1     Completed   0          2m35s
process-item-banana-6t2vl    0/1     Completed   0          2m35s
process-item-cherry-fm4fr    0/1     Completed   0          2m35s

# 创建的 jobs 都已经完成了相应的工作，此时状态为 1/1
[root@k8s-master01 job]# kubectl get jobs
NAME                  COMPLETIONS   DURATION   AGE
process-item-apple    1/1           91s        2m36s
process-item-banana   1/1           107s       2m36s
process-item-cherry   1/1           2m2s       2m36s
```

3、删除 job

```
[root@k8s-master01 job]# kubectl delete -f .
```

1.3、并行运行多 Job pods

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

并行 Job 会启动多个 Pod，此时需要设定 Job 的.spec.completions 参数为一个正数，当正常结束的 Pod 数量达至此参数设定的值后，Job 结束。此外，Job 的.spec.parallelism 参数用来控制并行度，即同时启动几个 Job 来处理 Work Item。

1、创建 Job

（1）创建资源清单文件

```
[root@k8s-master01 ~]# cd /root/job/
[root@k8s-master01 job]# vi job-number.yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: job-number
  labels:
    jobgroup: jobexample
spec:
  completions: 6
  parallelism: 2
  template:
    metadata:
      name: job-number
      labels:
        jobgroup: jobexample
    spec:
      containers:
      - name: c
        image: devopscube/kubernetes-job-demo:latest
        args: ["100"]
      restartPolicy: OnFailure
```

（2）更新资源清单文件

```
[root@k8s-master01 job]# kubectl apply -f job-number.yaml
job.batch/job-number created
```

2、观察 Pod

（1）可以看到生成了两个 pod 容器

```
[root@k8s-master01 job]# kubectl get pods
NAME                      READY   STATUS    RESTARTS   AGE
job-number-86snc          1/1     Running   0          48s
job-number-v5tzl          1/1     Running   0          48s
```

（2）这两个 pod 容器干完活，然后 job 变成了 2/6，干完活的 pod 就变成了 Completed 状

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

```
态，然后 job 再启动两个容器继续干活，直到整个任务完成。
    [root@k8s-master01 ~]# kubectl get jobs
    NAME         COMPLETIONS   DURATION   AGE
    job-number   2/6           6m2s       6m2s

    [root@k8s-master01 job]# kubectl get pods
    NAME                   READY   STATUS      RESTARTS   AGE
    job-number-86snc       0/1     Completed   0          6m20s
    job-number-f9lrj       1/1     Running     0          2m15s
    job-number-q5blb       1/1     Running     0          2m32s
    job-number-v5tzl       0/1     Completed   0          6m20s

（3）稍等 100 秒，查看：
    [root@k8s-master01 ~]# kubectl get jobs
    NAME         COMPLETIONS   DURATION   AGE
    job-number   4/6           8m10s      8m10s

    [root@k8s-master01 job]# kubectl get pods
    NAME                   READY   STATUS      RESTARTS   AGE
    job-number-86snc       0/1     Completed   0          8m21s
    job-number-dqr7m       1/1     Running     0          55s
    job-number-f9lrj       0/1     Completed   0          4m16s
    job-number-q5blb       0/1     Completed   0          4m33s
    job-number-v5tzl       0/1     Completed   0          8m21s
    job-number-zxhcp       1/1     Running     0          38s

（4）稍等 100 秒，查看：
    [root@k8s-master01 ~]# kubectl get jobs
    NAME         COMPLETIONS   DURATION   AGE
    job-number   6/6           11m        13m

    [root@k8s-master01 job]# kubectl get pods
    NAME                   READY   STATUS      RESTARTS   AGE
    job-number-86snc       0/1     Completed   0          12m
    job-number-dqr7m       0/1     Completed   0          4m35s
    job-number-f9lrj       0/1     Completed   0          7m56s
    job-number-q5blb       0/1     Completed   0          8m13s
    job-number-v5tzl       0/1     Completed   0          12m
    job-number-zxhcp       0/1     Completed   0          4m18s
```

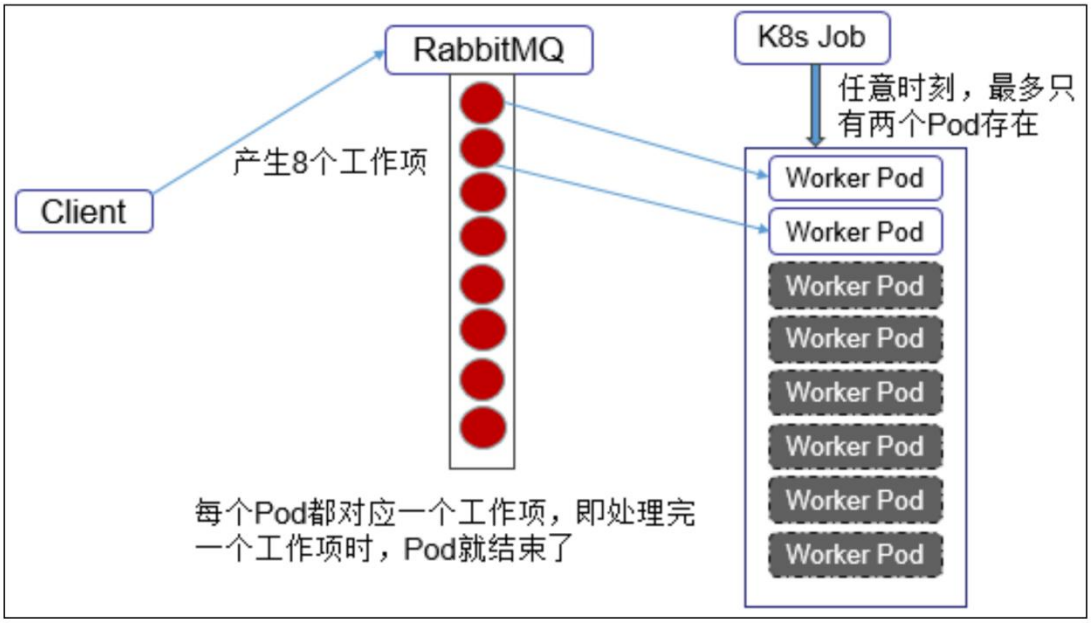
3、删除 job

```
    [root@k8s-master01 job]# kubectl delete -f .
```

1.4、Queue with Pod Per Work Item 模式

Queue with Pod Per Work Item 模式，在这种模式下需要一个任务队列存放 Work item，比如 RabbitMQ，客户端程序先把要处理的任务变成 Work item 放入任务队列，然后编写 Worker 程序、打包镜像并定义成为 Job 中的 Work Pod。Worker 程序的实现逻辑是从任务队列中拉取一个 Work item 并处理，在处理完成后即结束进程。

Queue with Pod Per Work Item 模式示例图：



1、使用 deployment 资源创建 rabbitmq

```
（1）创建 deployment rabbitmq 资源清单文件
   [root@k8s-master01 ~]# cd /root/job/
   [root@k8s-master01 job]# vi rabbitmq-deploy.yaml
   apiVersion: apps/v1
   kind: Deployment
   metadata:
     name: rabbitmq
     labels:
       component: rabbitmq
   spec:
     replicas: 1
     selector:
       matchLabels:
         app: taskQueue
```

```
        component: rabbitmq
    template:
      metadata:
        labels:
          app: taskQueue
          component: rabbitmq
      spec:
        containers:
        - image: rabbitmq:3.6.15
          name: rabbitmq
          ports:
          - containerPort: 5672

（2）更新资源清单文件
  [root@k8s-master01 job]# kubectl apply -f rabbitmq-deploy.yaml
  deployment.apps/rabbitmq-controller created

（3）查看 pod
  [root@k8s-master01 job]# kubectl get pods
  NAME                         READY   STATUS    RESTARTS   AGE
  rabbitmq-857495f894-c8t68    1/1     Running   0          2s

（4）创建 rabbitmq service
  [root@k8s-master01 job]# vi rabbitmq-service.yaml
  apiVersion: v1
  kind: Service
  metadata:
    labels:
      component: rabbitmq
    name: rabbitmq-service
  spec:
    ports:
    - port: 5672
    selector:
      app: taskQueue
      component: rabbitmq

（5）更新资源清单文件
  [root@k8s-master01 job]# kubectl apply -f rabbitmq-service.yaml
  service/rabbitmq-service created

（6）查看 svc
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

```
[root@k8s-master01 job]# kubectl get svc rabbitmq-service
NAME               TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)    AGE
rabbitmq-service   ClusterIP   10.10.74.240   <none>        5672/TCP   40s
```

2、测试消息队列

创建一个临时的可交互的 Pod，在它上面安装一些工具，然后测试消息的发布和消费。

```
（1）创建临时容器，安装 amqp-tools
   # 必须安装 amqp-tools 这个工具，要不然无法使用消息队列
   [root@k8s-master01 job]# kubectl run -it --rm temp --image ubuntu:14.04
   root@temp:/# apt-get update
   root@temp:/# apt-get install -y curl ca-certificates amqp-tools python dnsutils

（2）验证 RabbitMQ 服务
   root@temp:/# nslookup rabbitmq-service
   Server:      10.10.0.10
   Address:     10.10.0.10#53

   Name:    rabbitmq-service.default.svc.cluster.local
   Address: 10.10.74.240

（3）创建队列，发布消息和消费消息
   # 设置一个变量
   root@temp:/# export BROKER_URL=amqp://guest:guest@rabbitmq-service:5672

   # 创建一个名叫 foo 的队列
   root@temp:/#   /usr/bin/amqp-declare-queue   --url=$BROKER_URL   --queue=foo
--durable=foo
   foo

   # 向它推送一条消息
   root@temp:/# /usr/bin/amqp-publish --url=$BROKER_URL -r foo -p -b Hello

   # 然后取回它
   root@temp:/# /usr/bin/amqp-consume --url=$BROKER_URL -q foo -c 1 cat && echo
   Hello
```

最后一个命令中，amqp-consume 工具从队列中取走了一个消息，并把该消息传递给了随机命令的标准输出。在这种情况下，cat 会打印它从标准输入中读取的字符，echo 会添加回车符以便示例可读。

3、为队列增加任务

创建一个 job1 消息队列，并往这个队列中填充几个任务。

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

```
（1）创建消息队列
   root@temp:/#  /usr/bin/amqp-declare-queue  --url=$BROKER_URL  --queue=job1
--durable=job1

（2）填充任务
   root@temp:/# for f in apple banana cherry date fig grape lemon melon
   do
     /usr/bin/amqp-publish --url=$BROKER_URL -r job1 -p -b $f
   done
   # 这样，我们给队列中填充了 8 个消息。

（3）退出 pod，自动销毁
   root@temp:/# exit
```

4、创建镜像（在 k8s-node01 节点操作，因为我们只有一个 node 节点，所以 pod 肯定要调度到这里。）

现在我们创建一个镜像。用 amqp-consume 来从队列中读取消息并实际运行的程序。

```
（1）创建 python 程序和 dockerfile 文件
   [root@k8s-node01 ~]# mkdir examples
   [root@k8s-node01 ~]# cd examples/
   [root@k8s-node01 examples]# vi Dockerfile
   FROM ubuntu:14.04

   RUN apt-get update && \
       apt-get install -y curl ca-certificates amqp-tools python \
          --no-install-recommends \
       && rm -rf /var/lib/apt/lists/*
   COPY ./worker.py /worker.py

   CMD /usr/bin/amqp-consume --url=$BROKER_URL -q $QUEUE -c 1 cat && echo

（2）编译构建镜像
   [root@k8s-node01 examples]# docker build -t job-rabbitmq:v1 .

（3）打成 tar 包
   [root@k8s-node01 examples]# docker save -o job.tar job-rabbitmq:v1

（4）导入到 containerd 中
   [root@k8s-node01 examples]# ctr -n k8s.io i import job.tar
```

5、创建 job

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

（1）创建资源清单文件

```
[root@k8s-master01 ~]# cd /root/job/
[root@k8s-master01 job]# vi job-rabbitmq.yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: job-rabbitmq
spec:
  completions: 8
  parallelism: 2
  template:
    metadata:
      name: job-rabbitmq
    spec:
      containers:
      - name: job-rabbitmq
        image: job-rabbitmq:v1
        imagePullPolicy: IfNotPresent
        env:
        - name: BROKER_URL
          value: amqp://guest:guest@rabbitmq-service:5672
        - name: QUEUE
          value: job1
      restartPolicy: OnFailure
```

每个 Pod 使用队列中的一个消息然后退出。这样，Job 的完成计数就代表了完成的工作项的数量。上面中我们设置.spec.completions: 8，因为我们放了 8 项内容在队列中。

（2）更新资源清单文件

```
[root@k8s-master01 job]# kubectl apply -f job-rabbitmq.yaml
job.batch/job-rabbitmq created
```

（3）查看创建的 job

```
[root@k8s-master01 job]# kubectl describe job.batch/job-rabbitmq
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

```
[root@k8s-master01 job]# kubectl describe job.batch/job-rabbitmq
Name:             job-rabbitmq
Namespace:        default
Selector:         controller-uid=927dd8ac-8066-4c00-ad4b-bb52af701e48
Labels:           controller-uid=927dd8ac-8066-4c00-ad4b-bb52af701e48
                  job-name=job-rabbitmq
Annotations:      batch.kubernetes.io/job-tracking:
Parallelism:      2
Completions:      8
Completion Mode:  NonIndexed
Start Time:       Tue, 29 Aug 2023 06:29:43 -0400
Completed At:     Tue, 29 Aug 2023 06:29:58 -0400
Duration:         15s
Pods Statuses:    0 Active (0 Ready) / 8 Succeeded / 0 Failed
Pod Template:
  Labels:  controller-uid=927dd8ac-8066-4c00-ad4b-bb52af701e48
           job-name=job-rabbitmq
  Containers:
   job-rabbitmq:
    Image:      job-rabbitmq:v1
    Port:       <none>
    Host Port:  <none>
    Environment:
      BROKER_URL:  amqp://guest:guest@rabbitmq-service:5672
      QUEUE:       job1
    Mounts:        <none>
  Volumes:         <none>
Events:
  Type    Reason            Age   From            Message
  ----    ------            ----  ----            -------
  Normal  SuccessfulCreate  33s   job-controller  Created pod: job-rabbitmq-54tb9
  Normal  SuccessfulCreate  33s   job-controller  Created pod: job-rabbitmq-zw9th
  Normal  SuccessfulCreate  28s   job-controller  Created pod: job-rabbitmq-fc6pk
  Normal  SuccessfulCreate  28s   job-controller  Created pod: job-rabbitmq-dr7qj
  Normal  SuccessfulCreate  25s   job-controller  Created pod: job-rabbitmq-l7mz8
  Normal  SuccessfulCreate  25s   job-controller  Created pod: job-rabbitmq-pjxxv
  Normal  SuccessfulCreate  21s   job-controller  Created pod: job-rabbitmq-nrb64
  Normal  SuccessfulCreate  21s   job-controller  Created pod: job-rabbitmq-zjf2n
  Normal  Completed         18s   job-controller  Job completed
[root@k8s-master01 job]#
```

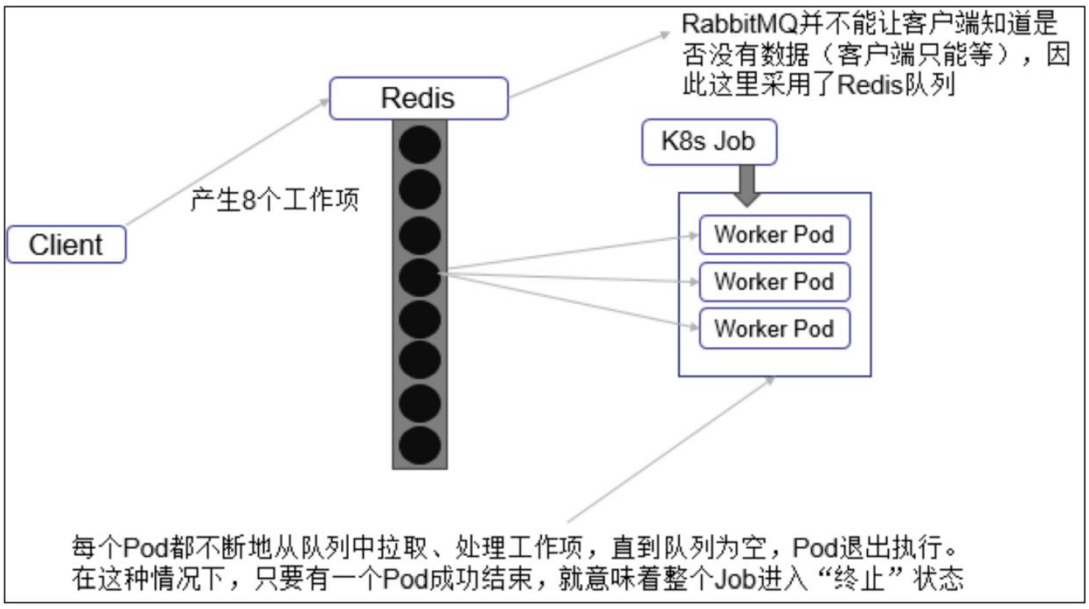
可以看到已经启动了 8 个 pod 容器，job 工作也已经完成。

1.5、Queue with Variable Pod Count 模式

Queue with Variable Pod Count 模式。由于这种模式下，Worker 程序需要知道队列中是否还有等待处理的 Work item，如果有就取出来处理，否则就认为所有工作完成并结束进程，所以任务队列通常要采用 Redis 或者数据库来实现。

Queue with Variable Pod Count 模式示例图：

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。



1、创建 redis pod

（1）创建资源清单文件

```
[root@k8s-master01 ~]# cd /root/job/
[root@k8s-master01 job]# vi redis-pod-svc.yaml
---
apiVersion: v1
kind: Pod
metadata:
  name: redis
  labels:
    app: redis
spec:
  containers:
    - name: redis
      image: redis
      env:
        - name: MASTER
          value: "true"
      ports:
        - containerPort: 6379
---
apiVersion: v1
kind: Service
metadata:
  name: redis
  labels:
```

```
      app: redis
  spec:
    ports:
      - port: 6379
        targetPort: 6379
    selector:
      app: redis
```

（2）更新资源清单文件

```
[root@k8s-master01 job]# kubectl apply -f redis-pod-svc.yaml
pod/redis created
service/redis created
```

（3）查看创建的资源

```
[root@k8s-master01 job]# kubectl get pods -l app=redis
NAME    READY   STATUS    RESTARTS   AGE
redis   1/1     Running   0          25s

[root@k8s-master01 job]# kubectl get svc -l app=redis
NAME    TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)    AGE
redis   ClusterIP   10.10.217.32   <none>        6379/TCP   26s
```

2、进入到 redis 容器中，创建键为 job2 的工作队列

（1）进入 redis pod 内，连接 redis

```
[root@k8s-master01 job]# kubectl exec -it redis -- bash
root@redis:/data# redis-cli -h redis
```

（2）rpush 表示在列表中添加一个或多个值

```
redis:6379> rpush job2 "apple" "banana" "cherry" "date" "fig" "grape" "lemon" "melon" "orange"
(integer) 9
```

（3）查看列表中的值

```
redis:6379> lrange job2 0 -1
1) "apple"
2) "banana"
3) "cherry"
4) "date"
5) "fig"
6) "grape"
7) "lemon"
8) "melon"
```

```
    9) "orange"
```

3、构建镜像（在 k8s-node01 节点操作，因为我们只有一个 node 节点，所以 pod 肯定要调度到这里。）

（1）使用一个带有 redis 客户端的 python 工作程序从消息队列中读出消息。

```
[root@k8s-node01 ~]# mkdir redis-examples
[root@k8s-node01 ~]# cd redis-examples/
[root@k8s-node01 redis-examples]# vi Dockerfile
FROM python
RUN pip install redis
COPY ./worker.py /worker.py
COPY ./rediswq.py /rediswq.py
CMD python worker.py
```

（2）创建 rediswq.py 文件（这里大家不用知道这两个脚本是怎么写的）

```
[root@k8s-node01 redis-examples]# vi rediswq.py
#!/usr/bin/env python
import redis
import uuid
import hashlib

class RedisWQ(object):
    def __init__(self, name, **redis_kwargs):
        self._db = redis.StrictRedis(**redis_kwargs)
        self._session = str(uuid.uuid4())
        self._main_q_key = name
        self._processing_q_key = name + ":processing"
        self._lease_key_prefix = name + ":leased_by_session:"

    def sessionID(self):
        return self._session

    def _main_qsize(self):
        return self._db.llen(self._main_q_key)

    def _processing_qsize(self):
        return self._db.llen(self._processing_q_key)

    def empty(self):
        return self._main_qsize() == 0 and self._processing_qsize() == 0

    def _itemkey(self, item):
```

```
            return hashlib.sha224(item).hexdigest()

        def _lease_exists(self, item):
            return self._db.exists(self._lease_key_prefix + self._itemkey(item))

        def lease(self, lease_secs=60, block=True, timeout=None):
            if block:
                item          =          self._db.brpoplpush(self._main_q_key,
self._processing_q_key, timeout=timeout)
            else:
                item          =          self._db.rpoplpush(self._main_q_key,
self._processing_q_key)
            if item:
                itemkey = self._itemkey(item)
                self._db.setex(self._lease_key_prefix  +  itemkey,  lease_secs,
self._session)
            return item

        def complete(self, value):
            self._db.lrem(self._processing_q_key, 0, value)
            itemkey = self._itemkey(value)
            self._db.delete(self._lease_key_prefix + itemkey)
```

（3）创建 worker.py 文件

```
[root@k8s-node01 redis-examples]# vi worker.py
#!/usr/bin/env python

import time
import rediswq   #引入上面创建的 python 程序

host="redis"   #传入主机地址，就是 svc 名称
q = rediswq.RedisWQ(name="job2", host=host)  #redis 工作队列名称
print("Worker with sessionID: " +  q.sessionID())
print("Initial queue state: empty=" + str(q.empty()))
while not q.empty():
  item = q.lease(lease_secs=10, block=True, timeout=2)
  if item is not None:
    itemstr = item.decode("utf-8")
    print("Working on " + itemstr)
    time.sleep(10) # Put your actual work here instead of sleep.
    q.complete(item)
  else:
```

```
        print("Waiting for work")
    print("Queue empty, exiting")

（4）构建镜像
    [root@k8s-node01 redis-examples]# docker build -t redis-job:v1 .

（5）导出镜像，导入到containerd
    [root@k8s-node01 redis-examples]# docker save -o redis-job.tar redis-job:v1
    [root@k8s-node01 redis-examples]# ctr -n k8s.io i import redis-job.tar
```

4、创建 Job

```
（1）创建资源清单文件
    [root@k8s-master01 ~]# cd /root/job/
    [root@k8s-master01 job]# vi job-redis.yaml
    apiVersion: batch/v1
    kind: Job
    metadata:
      name: job-redis
    spec:
      parallelism: 2
      template:
        metadata:
          name: job-redis
        spec:
          containers:
          - name: job-redis
            image: redis-job:v1
          restartPolicy: OnFailure

（2）更新资源清单文件
    [root@k8s-master01 job]# kubectl apply -f job-redis.yaml
    job.batch/job-redis created
```

说明：

每个 Pod 处理队列中的多个项目，直到队列中没有项目时便退出。

5、查看 Job、Pod 工作详情

```
    [root@k8s-master01 job]# kubectl describe job.batch/job-redis
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

```
[root@k8s-master01 ~]# kubectl describe job.batch/job-redis
Name:             job-redis
Namespace:        default
Selector:         controller-uid=26fe5b8e-46f4-4a81-9fcc-7686e4c57ea3
Labels:           controller-uid=26fe5b8e-46f4-4a81-9fcc-7686e4c57ea3
                  job-name=job-redis
Annotations:      batch.kubernetes.io/job-tracking:
Parallelism:      2
Completions:      <unset>
Completion Mode:  NonIndexed
Start Time:       Tue, 29 Aug 2023 21:25:42 -0400
Completed At:     Tue, 29 Aug 2023 21:26:36 -0400
Duration:         54s
Pods Statuses:    0 Active (0 Ready) / 2 Succeeded / 0 Failed
Pod Template:
  Labels:  controller-uid=26fe5b8e-46f4-4a81-9fcc-7686e4c57ea3
           job-name=job-redis
  Containers:
   job-redis:
    Image:        redis-job:v1
    Port:         <none>
    Host Port:    <none>
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Events:
  Type    Reason            Age   From            Message
  ----    ------            ----  ----            -------
  Normal  SuccessfulCreate  60s   job-controller  Created pod: job-redis-q2cqv
  Normal  SuccessfulCreate  60s   job-controller  Created pod: job-redis-lvr7m
  Normal  Completed         6s    job-controller  Job completed
[root@k8s-master01 ~]#
```

```
[root@k8s-master01 ~]# kubectl logs job-redis-q2cqv
Worker with sessionID: 44eaad90-2a9c-4c19-8406-14f90ff33ed7
Initial queue state: empty=False
Working on orange
Working on lemon
Working on fig
Working on cherry
Waiting for work
Waiting for work
Waiting for work
Waiting for work
Waiting for work
Queue empty, exiting

[root@k8s-master01 ~]# kubectl logs job-redis-lvr7m
Worker with sessionID: f71c2af8-92f0-4b17-a90e-835bfaabdf0e
Initial queue state: empty=False
Working on melon
Working on grape
Working on date
Working on banana
Working on apple
Queue empty, exiting
```

可以看到，一个 pod 处理了若干个工作单元。

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

2、CronJob：定时任务

2.1、CronJob 介绍

Cronjob 类似 Linux 系统的定时任务 Cron。

我们来看一下如何定义和使用 Cronjob：

需要掌握 CronJob 的定时表达式，它基本上照搬了 Linux Cron 的表达式，区别是第 1 位是分钟而不是秒，格式如下：

```
Minutes Hours DayofMonth Month DayofWeek Year
```

其中每个域都可出现的字符如下：

- Minutes（分）：可能出现“,”“-”“*”“/”这 4 个字符，有效范围为 0～59 的整数。
- Hours（时）：可能出现“,”“-”“*”“/”这 4 个字符，有效范围为 0～23 的整数。
- DayofMonth（日）：可能出现“,”“-”“*”“/”“?”“L”“W”“C”这 8 个字符，有效范围为 0～31 的整数。
- Month（月）：可能出现“,”“-”“*”“/”这 4 个字符，有效范围为 1～12 的整数。
- DayofWeek（周）：可能出现“,”“-”“*”“/”“?”“L”“W”“C”这 8 个字符，有效范围为 1～7 的整数。1 表示星期天、2 表示星期一，以此类推。

表达式中的特殊符号“*”与“/”的含义如下：

- *：表示匹配该域的任意值，假如在 Minutes 域使用“*”，则表示每分钟都会触发事件。
- /：表示从起始时间开始触发，然后每隔固定时间触发一次，例如在 Minutes 域设置为 5/20，则意味着第 1 次触发在第 5min 时，接下来每 20min 触发一次，将在第 25min、第 45min 等时刻分别触发。

例如，我们要每隔 1min 执行一次任务，则 Cron 表达式如下：

```
*/1 * * * *
```

2.2、CronJob 应用

定义了一个名为 hello 的 CronJob，任务每隔 1min 执行一次，运行的镜像是 busybox，执行的命令是 Shell 脚本，脚本执行时会在控制台输出当前时间和字符串“Hello from the Kubernetes cluster”。

1、创建 CronJob

```
（1）创建资源清单文件
[root@k8s-master01 ~]# cd /root/job/
```

```
[root@k8s-master01 job]# vi cronjob.yaml
apiVersion: batch/v1
kind: CronJob
metadata:
  name: hello
spec:
  schedule: "*/1 * * * *"
  jobTemplate:
    spec:
      template:
        spec:
          containers:
          - name: hello
            image: busybox
            imagePullPolicy: IfNotPresent
            command:
            - /bin/sh
            - -c
            - date; echo Hello from the Kubernetes cluster
          restartPolicy: OnFailure
```

（2）更新资源清单文件

```
[root@k8s-master01 job]# kubectl apply -f cronjob.yaml
cronjob.batch/hello created
```

（3）查看创建的 cronjob

```
[root@k8s-master01 job]# kubectl get cronjob
NAME    SCHEDULE      SUSPEND   ACTIVE   LAST SCHEDULE   AGE
hello   */1 * * * *   False     0        30s             60s
```

2、观察 job 和 pod

（1）创建 cronjob，定期执行会创建 job，一个 job 对应一个 pod 容器

```
[root@k8s-master01 job]# kubectl get job
NAME             COMPLETIONS   DURATION   AGE
hello-28222691   1/1           3s         2m3s
hello-28222692   1/1           3s         63s
hello-28222693   0/1           3s         3s
```

（2）查看生成的 pod 容器

```
[root@k8s-master01 job]# kubectl get pods
NAME                     READY   STATUS      RESTARTS      AGE
hello-28222691-9nzzl     0/1     Completed   0             2m17s
```

```
hello-28222692-d4k5x        0/1     Completed   0             77s
hello-28222693-rjvnq        0/1     Completed   0             17s

（3）查看相邻两个 pod 的日志
[root@k8s-master01 job]# kubectl logs -f hello-28222692-d4k5x
Wed Aug 30 02:12:00 UTC 2023
Hello from the Kubernetes cluster

[root@k8s-master01 job]# kubectl logs -f hello-28222693-rjvnq
Wed Aug 30 02:13:00 UTC 2023
Hello from the Kubernetes cluster
```

3、最后，清理 CronJob

```
[root@k8s-master01 job]# kubectl delete -f cronjob.yaml
cronjob.batch "hello" deleted
```

2.3、并发策略

spec.concurrencyPolicy 参数也是可选的。它声明了 CronJob 创建的任务执行时发生重叠如何处理。spec 仅能声明下列规则中的一种：

- Allow（默认）：CronJob 允许并发任务执行。
- Forbid：CronJob 不允许并发任务执行；如果新任务的执行时间到了而老任务没有执行完，CronJob 会忽略新任务的执行。
- Replace：如果新任务的执行时间到了而老任务没有执行完，CronJob 会用新任务替换当前正在运行的任务。

注意，并发性规则仅适用于相同 CronJob 创建的任务。如果有多个 CronJob，它们相应的任务总是允许并发执行的。

- 应用案例

示例 1：测试 Forbid，如果新任务的执行时间到了而老任务没有执行完，CronJob 会忽略新任务的执行。

```
（1）创建资源清单文件
[root@k8s-master01 ~]# cd /root/job/
[root@k8s-master01 job]# vi cronjob-forbid.yaml
apiVersion: batch/v1
kind: CronJob
metadata:
  name: hello
spec:
  concurrencyPolicy: Forbid
  schedule: "*/1 * * * *"
```

```
    jobTemplate:
      spec:
        template:
          spec:
            containers:
            - name: hello
              image: busybox
              imagePullPolicy: IfNotPresent
              command:
              - /bin/sh
              - -c
              - date; echo Hello from the Kubernetes cluster; sleep 101
            restartPolicy: OnFailure
```

对上面参数解释：

concurrencyPolicy: Forbid #如果新任务的执行时间到了而老任务没有执行完，CronJob 会忽略新任务的执行。

（2）创建资源清单文件

```
[root@k8s-master01 job]# kubectl apply -f cronjob-forbid.yaml
cronjob.batch/hello created
```

（3）查看创建的 cronjob

```
[root@k8s-master01 job]# kubectl get cronjob
NAME    SCHEDULE      SUSPEND   ACTIVE   LAST SCHEDULE   AGE
hello   */1 * * * *   False     0        <none>          10s
```

2、查看生成的 job 和 pod

（1）查看 job

```
[root@k8s-master01 job]# kubectl get job
NAME             COMPLETIONS   DURATION   AGE
hello-28222758   1/1           104s       2m34s
hello-28222759   0/1           50s        50s
```

（2）查看生成的 pod

```
[root@k8s-master01 job]# kubectl get pods
NAME                   READY   STATUS      RESTARTS   AGE
hello-28222758-m4zc5   0/1     Completed   0          2m46s
hello-28222759-tkd44   1/1     Running     0          62s
```

（3）查看生成的 pod 日志

```
[root@k8s-master01 job]# kubectl logs hello-28222758-m4zc5
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

```
    Wed Aug 30 03:18:00 UTC 2023
    Hello from the Kubernetes cluster

    [root@k8s-master01 job]# kubectl logs hello-28222759-tkd44
    Wed Aug 30 03:19:45 UTC 2023
    Hello from the Kubernetes cluster
```

第一个 pod，执行输出的时间是 03:18:00，定时任务写的是每隔一分钟执行一次，正常应该是在 03:19:00 执行，因为我们设置了 concurrencyPolicy: Forbid 这个参数，yaml 文件中写了 sleep 101 睡 101 秒，第二个 pod 执行输出的实际为 03:19:45，那么就说明新任务的执行时间到了而老任务没有执行完，CronJob 会忽略新任务的执行。

3、清除创建的 cronjob

```
    [root@k8s-master01 job]# kubectl delete -f cronjob-forbid.yaml
    cronjob.batch "hello" deleted
```

示例 2：测试 Replace，如果新任务的执行时间到了而老任务没有执行完，CronJob 会用新任务替换当前正在运行的任务。

（1）创建资源清单文件

```
    [root@k8s-master01 ~]# cd /root/job/
    [root@k8s-master01 job]# vi cronjob-replace.yaml
    apiVersion: batch/v1
    kind: CronJob
    metadata:
      name: hello
    spec:
      concurrencyPolicy: Replace
      schedule: "*/1 * * * *"
      jobTemplate:
        spec:
          template:
            spec:
              containers:
              - name: hello
                image: busybox
                imagePullPolicy: IfNotPresent
                command:
                - /bin/sh
                - -c
                - date; echo Hello from the Kubernetes cluster; sleep 100; echo
success
              restartPolicy: OnFailure
```

```
（2）创建资源清单文件
   [root@k8s-master01 job]# kubectl apply -f cronjob-replace.yaml
   cronjob.batch/hello created

（3）查看创建的 cronjob
   [root@k8s-master01 job]# kubectl get cronjob
   NAME    SCHEDULE      SUSPEND   ACTIVE   LAST SCHEDULE   AGE
   hello   */1 * * * *   False     0        <none>          3s
```

2、查看生成的 job 和 pod

```
（1）查看 job
   [root@k8s-master01 job]# kubectl get job
   NAME             COMPLETIONS   DURATION   AGE
   hello-28222779   0/1           7s         7s

（2）查看生成的 pod
   [root@k8s-master01 job]# kubectl get pods
   NAME                   READY   STATUS    RESTARTS   AGE
   hello-28222779-kdwqb   1/1     Running   0          35s

（3）查看生成的 pod 日志
   [root@k8s-master01 job]# kubectl get job -w
   NAME             COMPLETIONS   DURATION   AGE
   hello-28222779   0/1           0s         0s
   hello-28222779   0/1           60s        60s
   hello-28222780   0/1                      0s
   hello-28222780   0/1           2s         2s
```

从上面可以看出，sleep 100 时间还没到，新任务的执行时间到了而老任务没有执行完，CronJob 会用新任务替换当前正在运行的任务。

3、清除创建的 cronjob

```
   [root@k8s-master01 job]# kubectl delete -f cronjob-replace.yaml
   cronjob.batch "hello" deleted
```

2.4、Job 历史限制

spec.successfulJobsHistoryLimit 和 spec.failedJobsHistoryLimit 这两个字段是可选的。它们指定了可以保留完成和失败 Job 数量的限制。

默认没有限制，所有成功和失败的 Job 都会被保留。然而，当运行一个 Cron Job 时，很快就会堆积很多 Job，推荐设置这两个字段的值。设置限制值为 0，相关类型的 Job 完成后将不会被保留。

应用实例：

1、创建 cronjob

（1）创建资源清单文件

```
[root@k8s-master01 ~]# cd /root/job/
[root@k8s-master01 job]# vi cronjob-his.yaml
apiVersion: batch/v1
kind: CronJob
metadata:
  name: hello
spec:
  successfulJobsHistoryLimit: 1
  failedJobsHistoryLimit: 1
  schedule: "*/1 * * * *"
  jobTemplate:
    spec:
      template:
        spec:
          containers:
          - name: hello
            image: busybox
            imagePullPolicy: IfNotPresent
            command:
            - /bin/sh
            - -c
            - date; echo Hello from the Kubernetes cluster;
          restartPolicy: OnFailure
```

（2）更新资源清单文件

```
[root@k8s-master01 job]# kubectl apply -f cronjob-his.yaml
cronjob.batch/hello created
```

2、查看创建的第一个 job

```
[root@k8s-master01 job]# kubectl get job
NAME             COMPLETIONS   DURATION   AGE
hello-28222790   1/1           3s         19s

[root@k8s-master01 job]# kubectl get pods
NAME                   READY   STATUS      RESTARTS   AGE
hello-28222790-dcth8   0/1     Completed   0          29s

[root@k8s-master01 job]# kubectl logs hello-28222790-dcth8
```

```
Wed Aug 30 03:50:00 UTC 2023
Hello from the Kubernetes cluster
```

3、查看创建的第二个 job

```
[root@k8s-master01 job]# kubectl get job
NAME             COMPLETIONS   DURATION   AGE
hello-28222791   1/1           3s         4s

[root@k8s-master01 job]# kubectl get pods
NAME                   READY   STATUS      RESTARTS   AGE
hello-28222791-hdjpj   0/1     Completed   0          18s

[root@k8s-master01 job]# kubectl logs hello-28222791-hdjpj
Wed Aug 30 03:51:00 UTC 2023
Hello from the Kubernetes cluster
```

可以看到之前创建的 job 和 pod 已经被删除了。

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。