JOBS (Cron, Parallel, Coarse Parallel with Message Queue)

[Edition 1]

[Last Update 200909]

For any issues/help contact : ***support@k21academy.com***

Contents

1 Introduction....3
2 Documentation....4
2.1 Kubernetes Documentation....4
2. Running Automated Tasks with a CronJob....4
https://kubernetes.io/docs/tasks/job/automated-tasks-with-cron-jobs/....4
2.2 Linux Commands and VIM Commands....4
3 Working with Cronjob....5
3.1 Creating cronjob to execute on a scheduled time....5
3.2 Verifying and analysing the created cronjob....6
3.3 Clean-up the resources....7
4 Non-Parallel jobs....8
4.1 Creating kind job that supervises a pod counting from 9 to 1....8
4.2 Clean-up the resources....10
5 Handling Batch processing with Prallel Job using Template....11
5.1 Creating Job based on a Template....11
5.2 Clean-up the resources....13
6 Handling Coarse Parallel processing using Message Queue....14
6.1 Exercise Description....14
6.2 Starting a Message Queue service....14
6.3 Testing The Message Queue service....15
6.4 Filling the Queue with tasks....17
6.5 Create an image for the Job and Push to Docker Hub....17
6.6 Create Job and Pod using the image pushed to Docker Hub....20
6.7 Clean-up the resources....23
7 Summary....24

1 INTRODUCTION

A deployment is a controller that ensures an application's pods run according to a desired state. Deployments create and control replicaSets, which create and remove pods according to the deployment's *desired state*. Kubelets report the *current state* to the Kubernetes API server. The API server compares the *current state* to the *desired state* (stored in etcd). If the *current* and *desired* states differ, the Kubernetes API server tells the kubelet(s) to make deployment changes to match the *desired state*.

Parallel jobs without a fixed completion - jobs run multiple pods in parallel and when one pod is successful then the job is complete and all other pods terminate; this is also called a work queue.

1. Working with Cronjob
    - Creating cronjob to execute on a scheduled time
    - Verifying and analysing the created cronjob
    - Clean-up the resources

2. Supervising Pods with Jobs
    - Creating kind job that supervises a pod counting from 9 to 1

3. Handling Batch processing with Prallel Job using Template
    - Creating Job based on a Template

4. Handling Coarse Parallel processing using Message Queue
    - Exercise Description
    - Starting a Message Queue service
    - Testing The Message Queue service
    - Filling the Queue with tasks
    - Create an image for the Job and Push to Docker Hub

2 DOCUMENTATION

2.1 Kubernetes Documentation

1. Application Developer (CKAD)
   https://www.cncf.io/certification/ckad/
2. Running Automated Tasks with a CronJob

   https://kubernetes.io/docs/tasks/job/automated-tasks-with-cron-jobs/

3. Run Jobs
   https://kubernetes.io/docs/tasks/job/
4. Coarse Parallel Jobs
   https://kubernetes.io/docs/tasks/job/coarse-parallel-processing-work-queue/

2.2 Linux Commands and VIM Commands

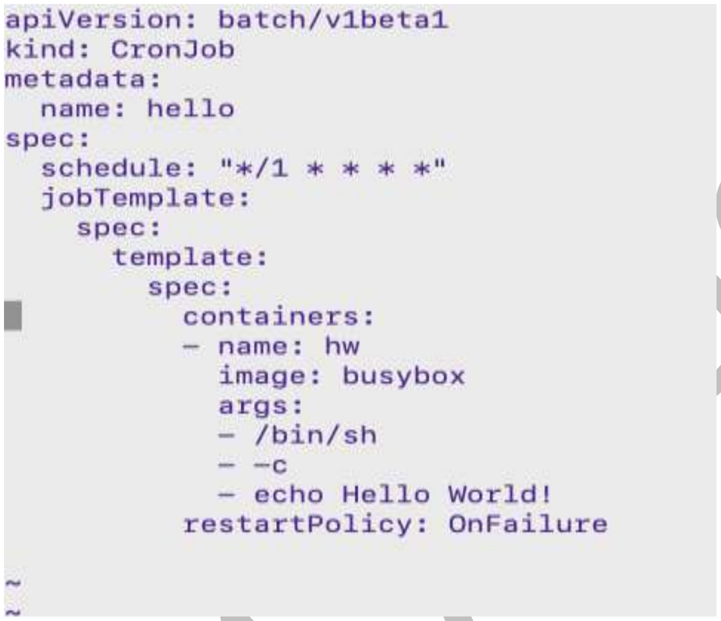
1. Basic Linux Commands
   https://maker.pro/linux/tutorial/basic-linux-commands-for-beginners
   https://www.hostinger.in/tutorials/linux-commands
2. Basic VIM Commands
   https://coderwall.com/p/adv71w/basic-vim-commands-for-getting-started
3. Popular VIM Commands
   https://www.keycdn.com/blog/vim-commands

3 WORKING WITH CRONJOB

3.1 Creating cronjob to execute on a scheduled time

1. View the file cron.yaml

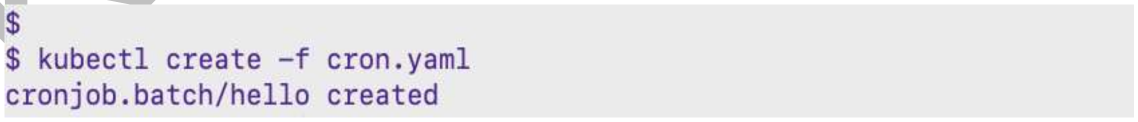
```
$ vi cron.yaml
```



```
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: hello
spec:
  schedule: "*/1 * * * *"
  jobTemplate:
    spec:
      template:
        spec:
          containers:
          - name: hw
            image: busybox
            args:
            - /bin/sh
            - -c
            - echo Hello World!
          restartPolicy: OnFailure

~
~
```

2. Create the cronjob using the yaml from previous step

```
$ kubectl create -f cron.yaml
```

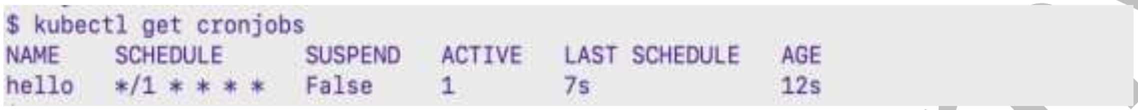


```
$
$ kubectl create -f cron.yaml
cronjob.batch/hello created
```

3.2 Verifying and analysing the created cronjob

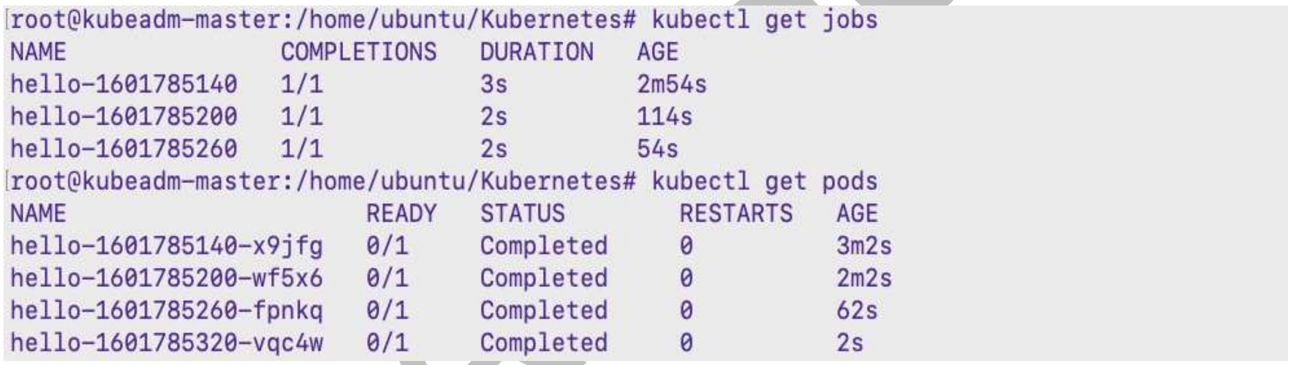
1. View the status of the cronjob

```
$ kubectl get cronjobs
```



```
$ kubectl get cronjobs
NAME    SCHEDULE      SUSPEND   ACTIVE   LAST SCHEDULE   AGE
hello   */1 * * * *   False     1        7s              12s
```

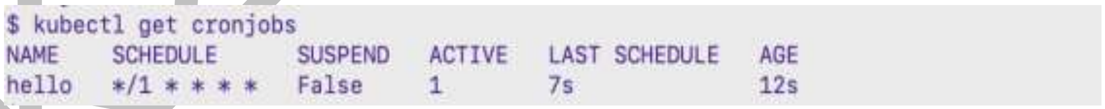
```
$ kubectl get jobs
$ kubectl get pods
```



```
root@kubeadm-master:/home/ubuntu/Kubernetes# kubectl get jobs
NAME               COMPLETIONS   DURATION   AGE
hello-1601785140   1/1           3s         2m54s
hello-1601785200   1/1           2s         114s
hello-1601785260   1/1           2s         54s
root@kubeadm-master:/home/ubuntu/Kubernetes# kubectl get pods
NAME                     READY   STATUS      RESTARTS   AGE
hello-1601785140-x9jfg   0/1     Completed   0          3m2s
hello-1601785200-wf5x6   0/1     Completed   0          2m2s
hello-1601785260-fpnkq   0/1     Completed   0          62s
hello-1601785320-vqc4w   0/1     Completed   0          2s
```

2. Wait for about a minute and you will notice that cronjob executes on the specified configured time and creates pod to execute the assigned job

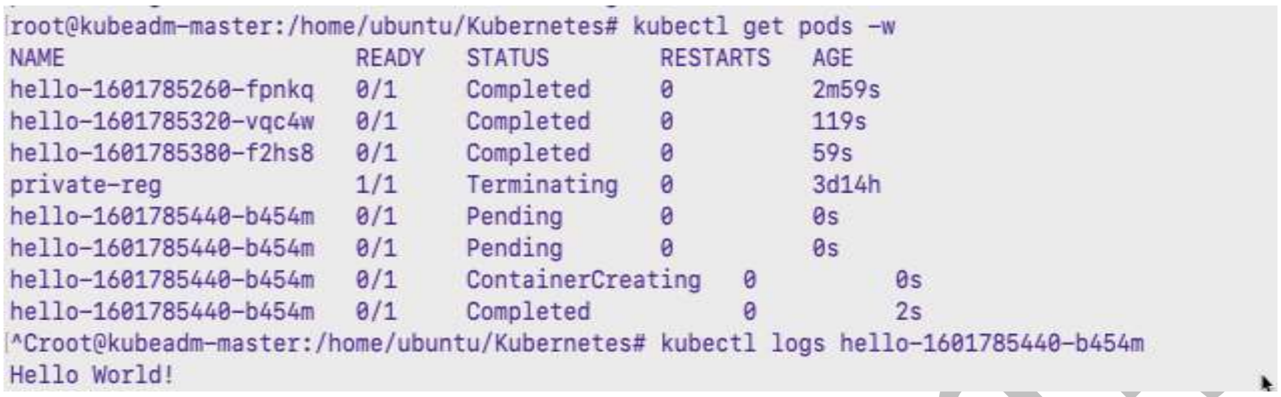
```
$ kubectl get cronjobs
```



```
$ kubectl get cronjobs
NAME    SCHEDULE      SUSPEND   ACTIVE   LAST SCHEDULE   AGE
hello   */1 * * * *   False     1        7s              12s
```

3. View the pods created by the job. Use the pod name to view its logs and verify that the jobs has run successfully

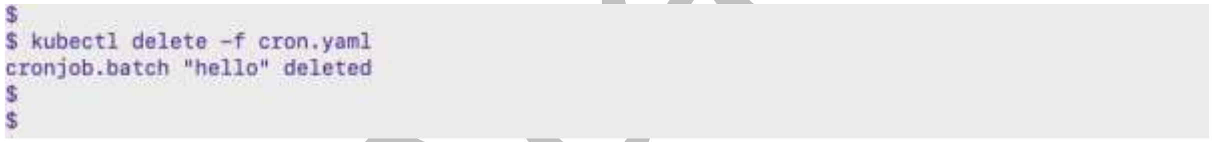
```
$ kubectl get pods -w
$ kubectl logs <pod_name>
```



```
root@kubeadm-master:/home/ubuntu/Kubernetes# kubectl get pods -w
NAME                        READY   STATUS        RESTARTS   AGE
hello-1601785260-fpnkq      0/1     Completed     0          2m59s
hello-1601785320-vqc4w      0/1     Completed     0          119s
hello-1601785380-f2hs8      0/1     Completed     0          59s
private-reg                 1/1     Terminating   0          3d14h
hello-1601785440-b454m      0/1     Pending       0          0s
hello-1601785440-b454m      0/1     Pending       0          0s
hello-1601785440-b454m      0/1     ContainerCreating   0          0s
hello-1601785440-b454m      0/1     Completed           0          2s
^Croot@kubeadm-master:/home/ubuntu/Kubernetes# kubectl logs hello-1601785440-b454m
Hello World!
```

3.3 Clean-up the resources

```
$ kubectl delete -f cron.yaml
```



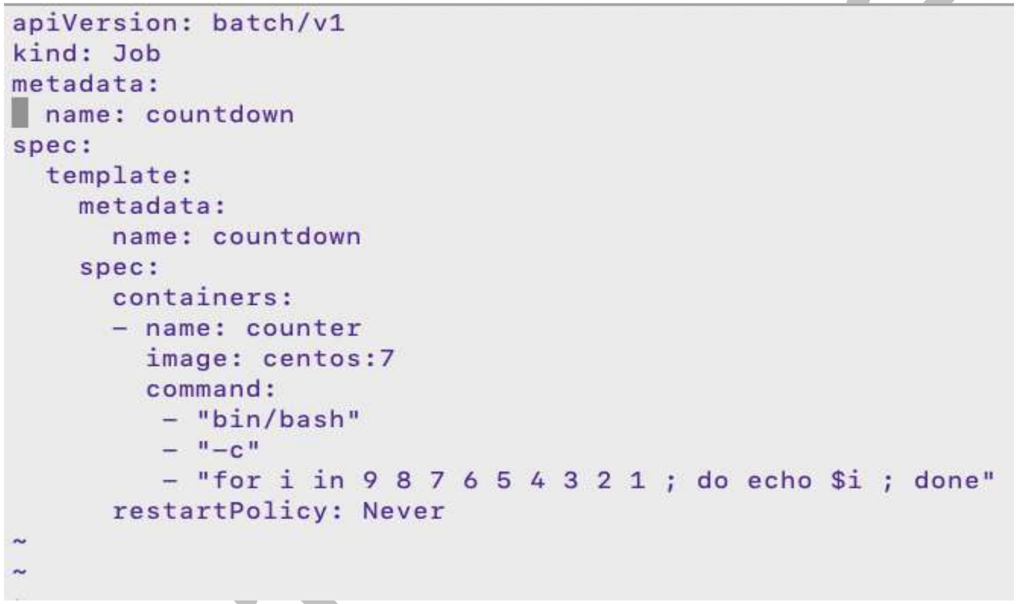
```
$
$ kubectl delete -f cron.yaml
cronjob.batch "hello" deleted
$
$
```

4 NON-PARALLEL JOBS

4.1 Creating kind job that supervises a pod counting from 9 to 1

1. Create a kind Job using the below file, this helps us create Pod. Pod will execute the task but is supervised by the Job

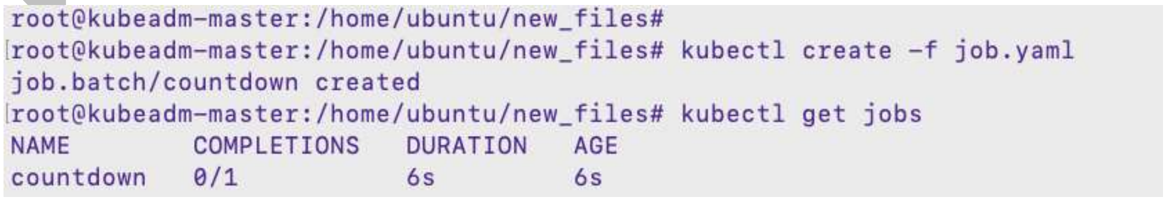
```
$ vim job.yaml
```



```
apiVersion: batch/v1
kind: Job
metadata:
  name: countdown
spec:
  template:
    metadata:
      name: countdown
    spec:
      containers:
      - name: counter
        image: centos:7
        command:
         - "bin/bash"
         - "-c"
         - "for i in 9 8 7 6 5 4 3 2 1 ; do echo $i ; done"
      restartPolicy: Never
~
~
```

2. Creating resources using the job.yaml file and listing the job type resources in the cluster

```
$ kubectl create -f job.yaml

$ kubectl get jobs
```

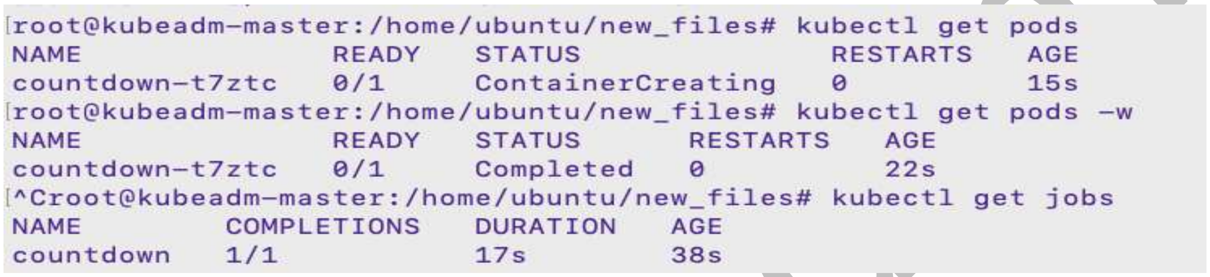


```
root@kubeadm-master:/home/ubuntu/new_files#
root@kubeadm-master:/home/ubuntu/new_files# kubectl create -f job.yaml
job.batch/countdown created
root@kubeadm-master:/home/ubuntu/new_files# kubectl get jobs
NAME        COMPLETIONS   DURATION   AGE
countdown   0/1           6s         6s
```

3. List the pod created and supervised by the countdown job and see to it that Pod executes the task and then marks as completed state.

4. Job also marks as completed once Pod has executed its task and completed with Success state.

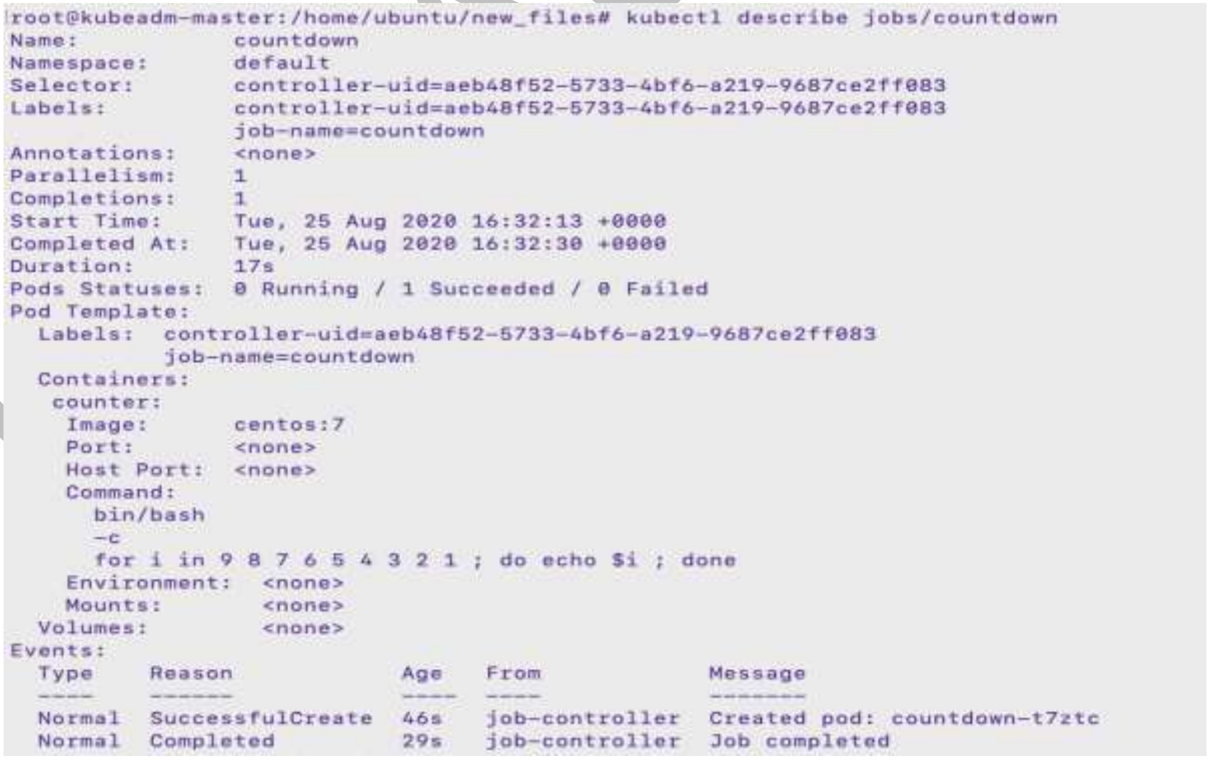
```
$ kubectl get pods
$ kubectl get pods -w
$ kubectl get jobs
```



```
root@kubeadm-master:/home/ubuntu/new_files# kubectl get pods
NAME              READY   STATUS              RESTARTS   AGE
countdown-t7ztc   0/1     ContainerCreating   0          15s
root@kubeadm-master:/home/ubuntu/new_files# kubectl get pods -w
NAME              READY   STATUS      RESTARTS   AGE
countdown-t7ztc   0/1     Completed   0          22s
^Croot@kubeadm-master:/home/ubuntu/new_files# kubectl get jobs
NAME        COMPLETIONS   DURATION   AGE
countdown   1/1           17s        38s
```

5. Describe the job and get more details and important parameters like Parallelism, Completions, Pod Statuses and Events section.

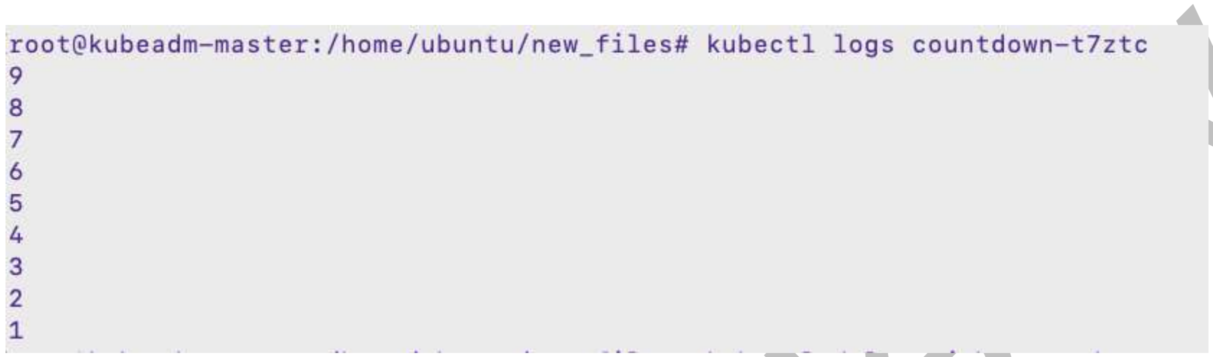
```
$ kubectl describe jobs/countdown
```



```
root@kubeadm-master:/home/ubuntu/new_files# kubectl describe jobs/countdown
Name:           countdown
Namespace:      default
Selector:       controller-uid=aeb48f52-5733-4bf6-a219-9687ce2ff083
Labels:         controller-uid=aeb48f52-5733-4bf6-a219-9687ce2ff083
                job-name=countdown
Annotations:    <none>
Parallelism:    1
Completions:    1
Start Time:     Tue, 25 Aug 2020 16:32:13 +0000
Completed At:   Tue, 25 Aug 2020 16:32:30 +0000
Duration:       17s
Pods Statuses:  0 Running / 1 Succeeded / 0 Failed
Pod Template:
  Labels:  controller-uid=aeb48f52-5733-4bf6-a219-9687ce2ff083
           job-name=countdown
  Containers:
   counter:
    Image:      centos:7
    Port:       <none>
    Host Port:  <none>
    Command:
      bin/bash
      -c
      for i in 9 8 7 6 5 4 3 2 1 ; do echo $i ; done
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Events:
  Type    Reason            Age   From            Message
  ----    ------            ----  ----            -------
  Normal  SuccessfulCreate  46s   job-controller  Created pod: countdown-t7ztc
  Normal  Completed         29s   job-controller  Job completed
```

6. We can also look into pod logs and see that it has printed from 9 to 1 as that was the task assigned to be the pod. Make sure to mention the Pod name as in your cluster.

```
$ kubectl logs <pod_name>
```

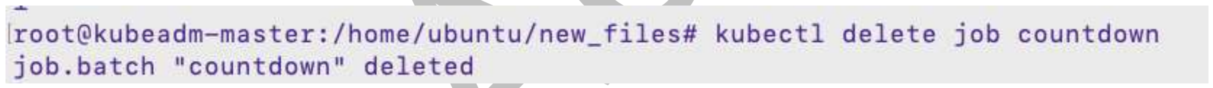


```
root@kubeadm-master:/home/ubuntu/new_files# kubectl logs countdown-t7ztc
9
8
7
6
5
4
3
2
1
```

4.2 Clean-up the resources

1. Clean-up: Delete the Job created in this task either by specifying the name of the resorce or else we can do the same as we do in all exercises using the yaml file

```
$ kubectl delete job countdown
```



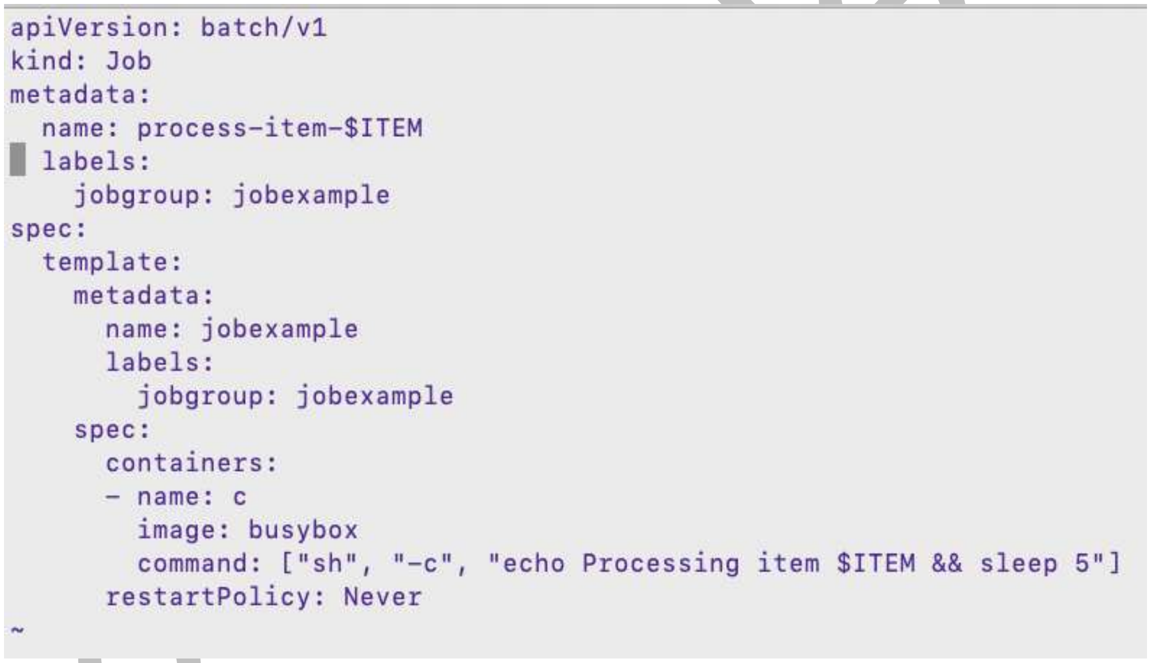
```
root@kubeadm-master:/home/ubuntu/new_files# kubectl delete job countdown
job.batch "countdown" deleted
```

5 HANDLING BATCH PROCESSING WITH PRALLEL JOB USING TEMPLATE

5.1 Creating Job based on a Template

1. Create a multiple jobs based on a standard template and $ITEM in the template has to be replaced before use

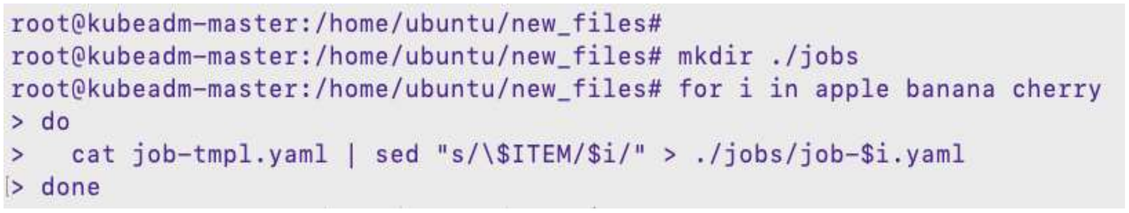
```
$ vim job-tmpl.yaml
```



```
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 5"]
      restartPolicy: Never
~
```

2. Creating a template using which we can create multiple job manifest file.
3. Create Job manifests from the template

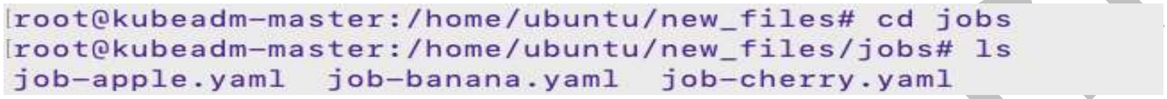
```
$ mkdir ./jobs
$ for i in apple banana cherry
do
  cat job-tmpl.yaml | sed "s/\$ITEM/$i/" > ./jobs/job-$i.yaml
done
```



```
root@kubeadm-master:/home/ubuntu/new_files#
root@kubeadm-master:/home/ubuntu/new_files# mkdir ./jobs
root@kubeadm-master:/home/ubuntu/new_files# for i in apple banana cherry
> do
>   cat job-tmpl.yaml | sed "s/\$ITEM/$i/" > ./jobs/job-$i.yaml
> done
```

4. Check that a folder named jobs is created and it has 3 Job manifest files

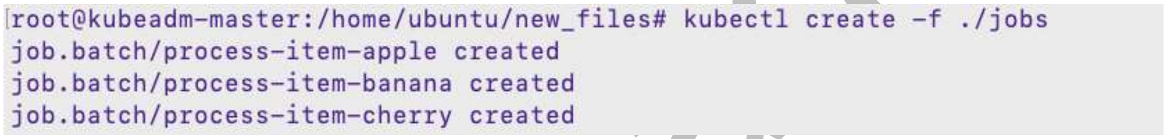
```
$ cd jobs
$ ls
```



```
root@kubeadm-master:/home/ubuntu/new_files# cd jobs
root@kubeadm-master:/home/ubuntu/new_files/jobs# ls
job-apple.yaml  job-banana.yaml  job-cherry.yaml
```

5. Create Jobs from the manifests. All jobs can be created in parallel using one kubectl command

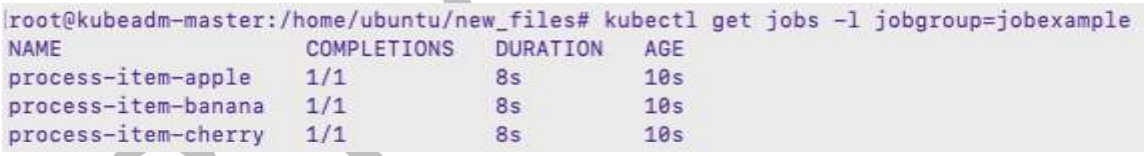
```
$ kubectl create -f ./jobs
```



```
root@kubeadm-master:/home/ubuntu/new_files# kubectl create -f ./jobs
job.batch/process-item-apple created
job.batch/process-item-banana created
job.batch/process-item-cherry created
```

6. All the jobs created have the same label. So use the label and get the list of the jobs created

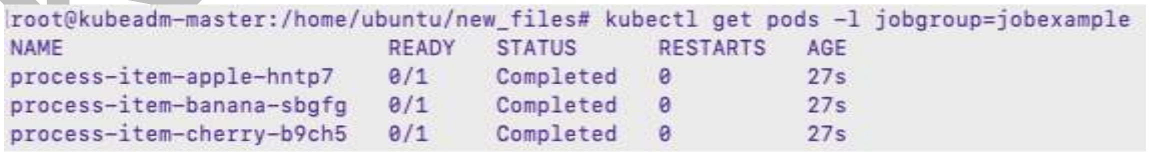
```
$ kubectl get jobs -l jobgroup=jobexample
```



```
root@kubeadm-master:/home/ubuntu/new_files# kubectl get jobs -l jobgroup=jobexample
NAME                  COMPLETIONS   DURATION   AGE
process-item-apple    1/1           8s         10s
process-item-banana   1/1           8s         10s
process-item-cherry   1/1           8s         10s
```

7. Use the label and get the list of the pods created and completed

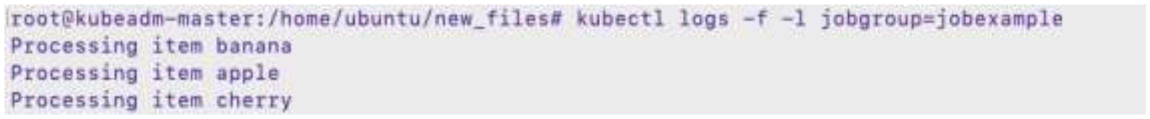
```
$ kubectl get pods -l jobgroup=jobexample
```



```
root@kubeadm-master:/home/ubuntu/new_files# kubectl get pods -l jobgroup=jobexample
NAME                        READY   STATUS      RESTARTS   AGE
process-item-apple-hntp7    0/1     Completed   0          27s
process-item-banana-sbgfg   0/1     Completed   0          27s
process-item-cherry-b9ch5   0/1     Completed   0          27s
```

8. Check the logs of all of the pods using single kubectl command

```
$ kubectl logs -f -l jobgroup=jobexample
```



```
root@kubeadm-master:/home/ubuntu/new_files# kubectl logs -f -l jobgroup=jobexample
Processing item banana
Processing item apple
Processing item cherry
```

5.2 Clean-up the resources

1. Clean-up: Delete the all Jobs created in this task with single kubectl command

```
$ kubectl delete job -l jobgroup=jobexample
```

6 HANDLING COARSE PARALLEL PROCESSING USING MESSAGE QUEUE

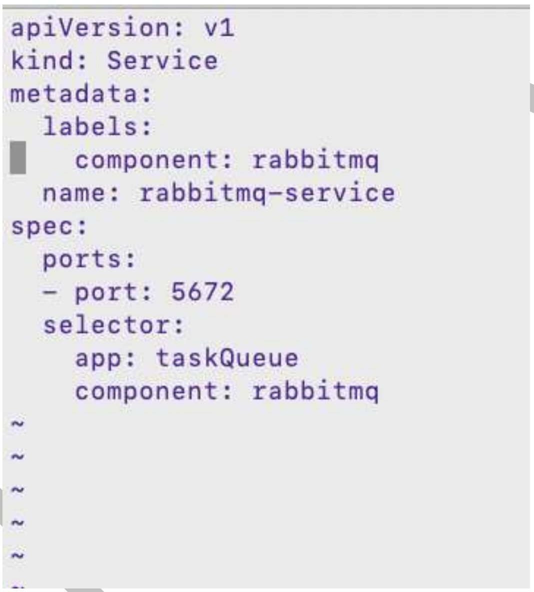
6.1 Exercise Description

1. **Start a message queue service.** In this exercise, we will use RabbitMQ, we would set up a message queue service once and reuse it for many jobs.
2. **Create a queue, and fill it with messages.** Each message represents one task to be done.
3. **Start a Job that works on tasks from the queue**. The Job starts several pods. Each pod takes one task from the message queue, processes it, and repeats until the end of the queue is reached.

6.2 Starting a Message Queue service

1. Start RabbitMQ service using the below yaml file. Message Queue service is listening on port 5672

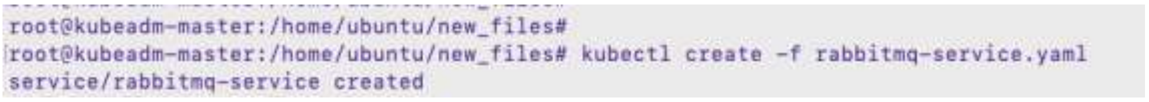
```
$ vim rabbitmq-service.yaml
```



```
apiVersion: v1
kind: Service
metadata:
  labels:
    component: rabbitmq
  name: rabbitmq-service
spec:
  ports:
  - port: 5672
  selector:
    app: taskQueue
    component: rabbitmq
~
~
~
~
~
```

2. Creating a service for RabbitMQ application using the above yaml file.

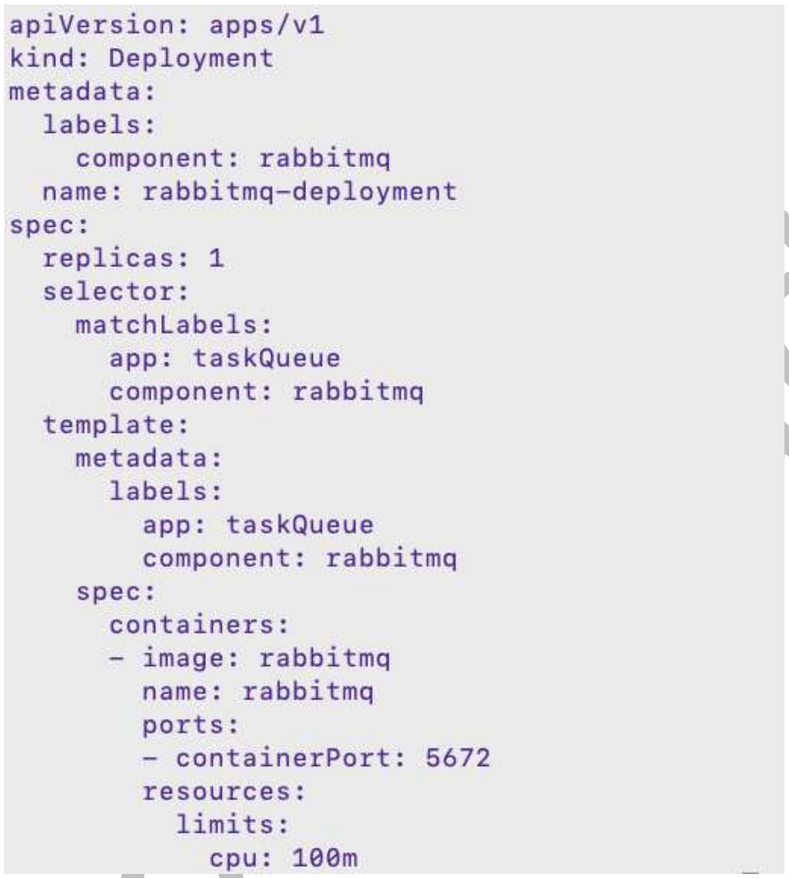
```
$ kubectl create -f rabbitmq-service.yaml
```



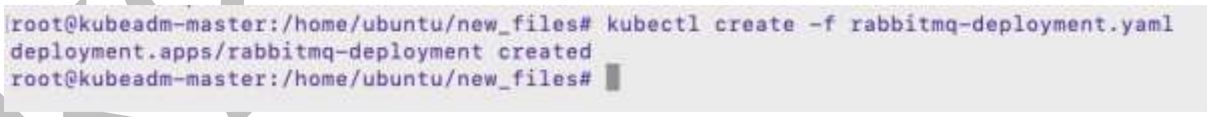
```
root@kubeadm-master:/home/ubuntu/new_files#
root@kubeadm-master:/home/ubuntu/new_files# kubectl create -f rabbitmq-service.yaml
service/rabbitmq-service created
```

3. Create RabbitMQ application Pod to able to host the message queue functionality. Create it as deployment.

```
$ vim rabbitmq-deployment.yaml
$ kubectl create -f rabbitmq-deployment.yaml
```



```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    component: rabbitmq
  name: rabbitmq-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: taskQueue
      component: rabbitmq
  template:
    metadata:
      labels:
        app: taskQueue
        component: rabbitmq
    spec:
      containers:
      - image: rabbitmq
        name: rabbitmq
        ports:
        - containerPort: 5672
        resources:
          limits:
            cpu: 100m
```

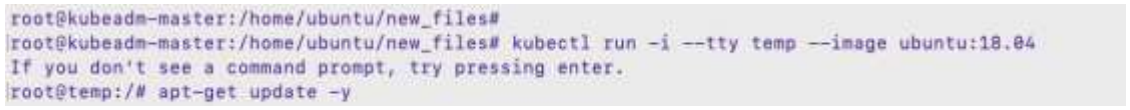


```
root@kubeadm-master:/home/ubuntu/new_files# kubectl create -f rabbitmq-deployment.yaml
deployment.apps/rabbitmq-deployment created
root@kubeadm-master:/home/ubuntu/new_files#
```

6.3 Testing The Message Queue service

1. Create a temporary interactive pod, install some tools on it, and experiment with queues

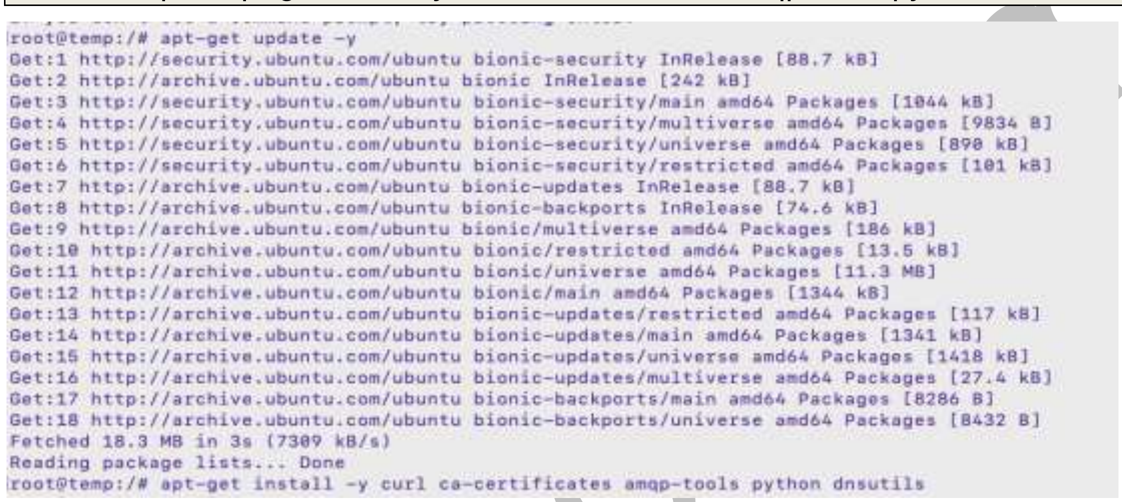
```
$ kubectl run -it temp --image ubuntu:18.04
root@temp:/#
```



```
root@kubeadm-master:/home/ubuntu/new_files#
root@kubeadm-master:/home/ubuntu/new_files# kubectl run -i --tty temp --image ubuntu:18.04
If you don't see a command prompt, try pressing enter.
root@temp:/# apt-get update -y
```

2. We would update the software repos and install the amqp-tools to work with message queues

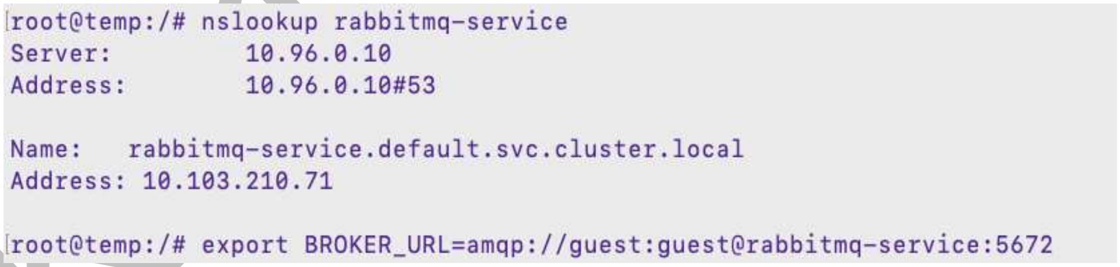
```
root@temp:/# apt-get update -y
root@temp:/# apt-get install -y curl ca-certificates amqp-tools python dnsutils
```



```
root@temp:/# apt-get update -y
Get:1 http://security.ubuntu.com/ubuntu bionic-security InRelease [88.7 kB]
Get:2 http://archive.ubuntu.com/ubuntu bionic InRelease [242 kB]
Get:3 http://security.ubuntu.com/ubuntu bionic-security/main amd64 Packages [1044 kB]
Get:4 http://security.ubuntu.com/ubuntu bionic-security/multiverse amd64 Packages [9834 B]
Get:5 http://security.ubuntu.com/ubuntu bionic-security/universe amd64 Packages [898 kB]
Get:6 http://security.ubuntu.com/ubuntu bionic-security/restricted amd64 Packages [101 kB]
Get:7 http://archive.ubuntu.com/ubuntu bionic-updates InRelease [88.7 kB]
Get:8 http://archive.ubuntu.com/ubuntu bionic-backports InRelease [74.6 kB]
Get:9 http://archive.ubuntu.com/ubuntu bionic/multiverse amd64 Packages [186 kB]
Get:10 http://archive.ubuntu.com/ubuntu bionic/restricted amd64 Packages [13.5 kB]
Get:11 http://archive.ubuntu.com/ubuntu bionic/universe amd64 Packages [11.3 MB]
Get:12 http://archive.ubuntu.com/ubuntu bionic/main amd64 Packages [1344 kB]
Get:13 http://archive.ubuntu.com/ubuntu bionic-updates/restricted amd64 Packages [117 kB]
Get:14 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 Packages [1341 kB]
Get:15 http://archive.ubuntu.com/ubuntu bionic-updates/universe amd64 Packages [1418 kB]
Get:16 http://archive.ubuntu.com/ubuntu bionic-updates/multiverse amd64 Packages [27.4 kB]
Get:17 http://archive.ubuntu.com/ubuntu bionic-backports/main amd64 Packages [8286 B]
Get:18 http://archive.ubuntu.com/ubuntu bionic-backports/universe amd64 Packages [8432 B]
Fetched 18.3 MB in 3s (7309 kB/s)
Reading package lists... Done
root@temp:/# apt-get install -y curl ca-certificates amqp-tools python dnsutils
```

3. Verify if we can discover the rabbitmq service

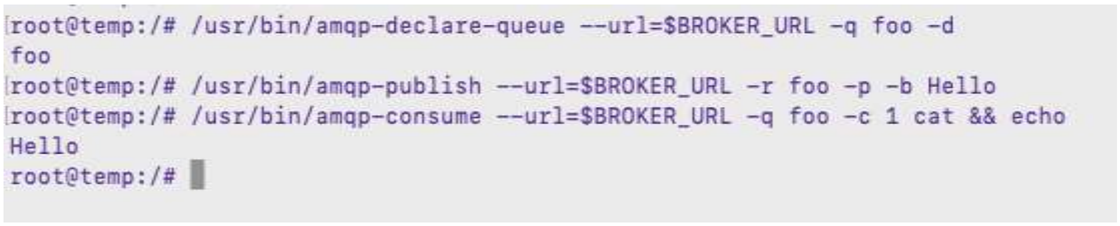
```
root@temp:/# nslookup rabbitmq-service
root@temp:/# export BROKER_URL=amqp://guest:guest@rabbitmq-service:5672
```



```
root@temp:/# nslookup rabbitmq-service
Server:         10.96.0.10
Address:        10.96.0.10#53

Name:   rabbitmq-service.default.svc.cluster.local
Address: 10.103.210.71

root@temp:/# export BROKER_URL=amqp://guest:guest@rabbitmq-service:5672
```

4. Create a dummy queue called “foo” and try publishing and consume dummy message “Hello”

```
root@temp:/# /usr/bin/amqp-declare-queue --url=$BROKER_URL -q foo -d
root@temp:/# /usr/bin/amqp-publish --url=$BROKER_URL -r foo -p -b Hello
root@temp:/# /usr/bin/amqp-consume --url=$BROKER_URL -q foo -c 1 cat && echo
```

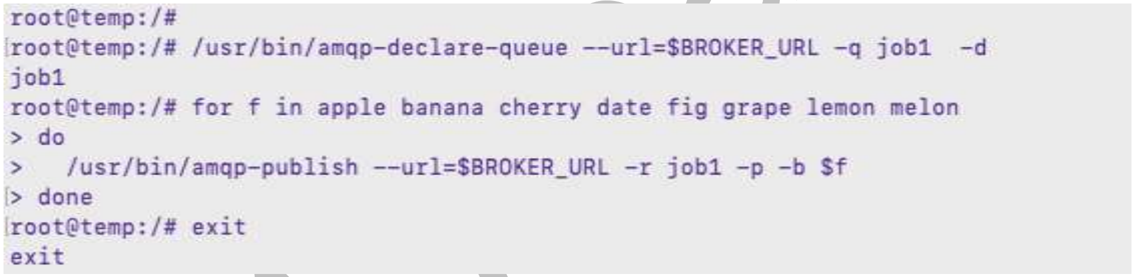


```
root@temp:/# /usr/bin/amqp-declare-queue --url=$BROKER_URL -q foo -d
foo
root@temp:/# /usr/bin/amqp-publish --url=$BROKER_URL -r foo -p -b Hello
root@temp:/# /usr/bin/amqp-consume --url=$BROKER_URL -q foo -c 1 cat && echo
Hello
root@temp:/#
```

6.4 Filling the Queue with tasks

1. Create queue named “job1” and publish 8 messages to the message queue so that we can consume them in our application pod

```
root@temp:/# /usr/bin/amqp-declare-queue --url=$BROKER_URL -q job1 -d
root@temp:/# for f in apple banana cherry date fig grape lemon melon
 do
/usr/bin/amqp-publish --url=$BROKER_URL -r job1 -p -b $f
 done
root@temp:/# exit
```

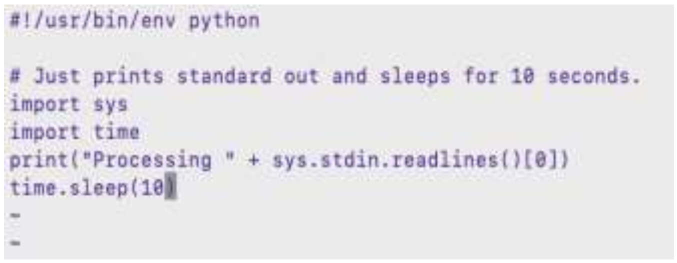


```
root@temp:/#
root@temp:/# /usr/bin/amqp-declare-queue --url=$BROKER_URL -q job1  -d
job1
root@temp:/# for f in apple banana cherry date fig grape lemon melon
> do
>   /usr/bin/amqp-publish --url=$BROKER_URL -r job1 -p -b $f
> done
root@temp:/# exit
exit
```

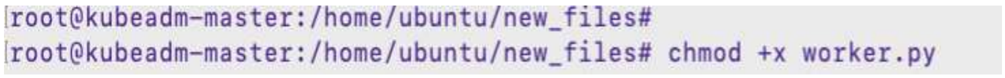
6.5 Create an image for the Job and Push to Docker Hub

1. Write python code to use the amqp-consume utility to read the message from the queue and run our actual program. Give executable permission to the python script. This is simple example program:

```
$ vim worker.py
$ chmod +x worker.py
```



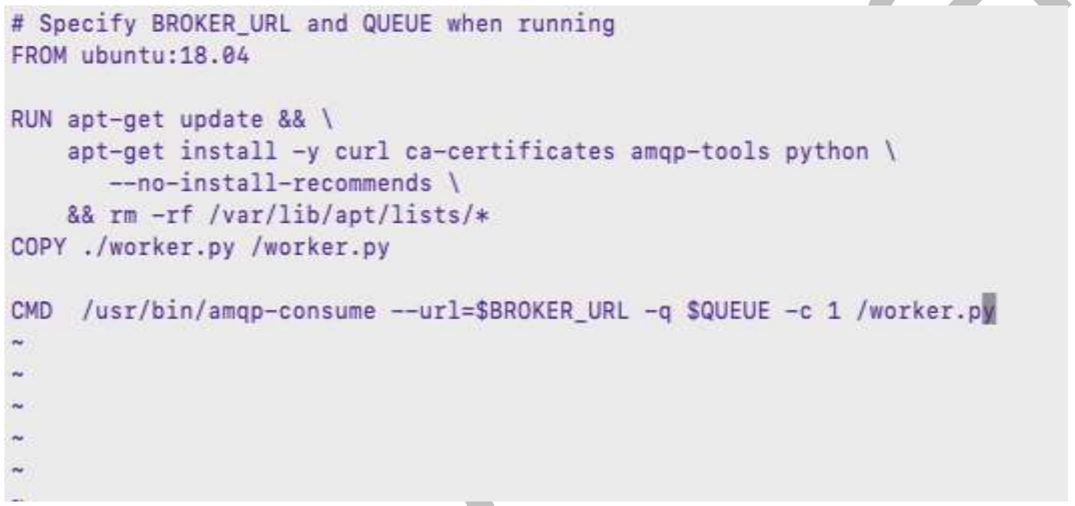
```
#!/usr/bin/env python

# Just prints standard out and sleeps for 10 seconds.
import sys
import time
print("Processing " + sys.stdin.readlines()[0])
time.sleep(10)
~
~
```



```
root@kubeadm-master:/home/ubuntu/new_files#
root@kubeadm-master:/home/ubuntu/new_files# chmod +x worker.py
```

2. Create a docker file to build image for the python application

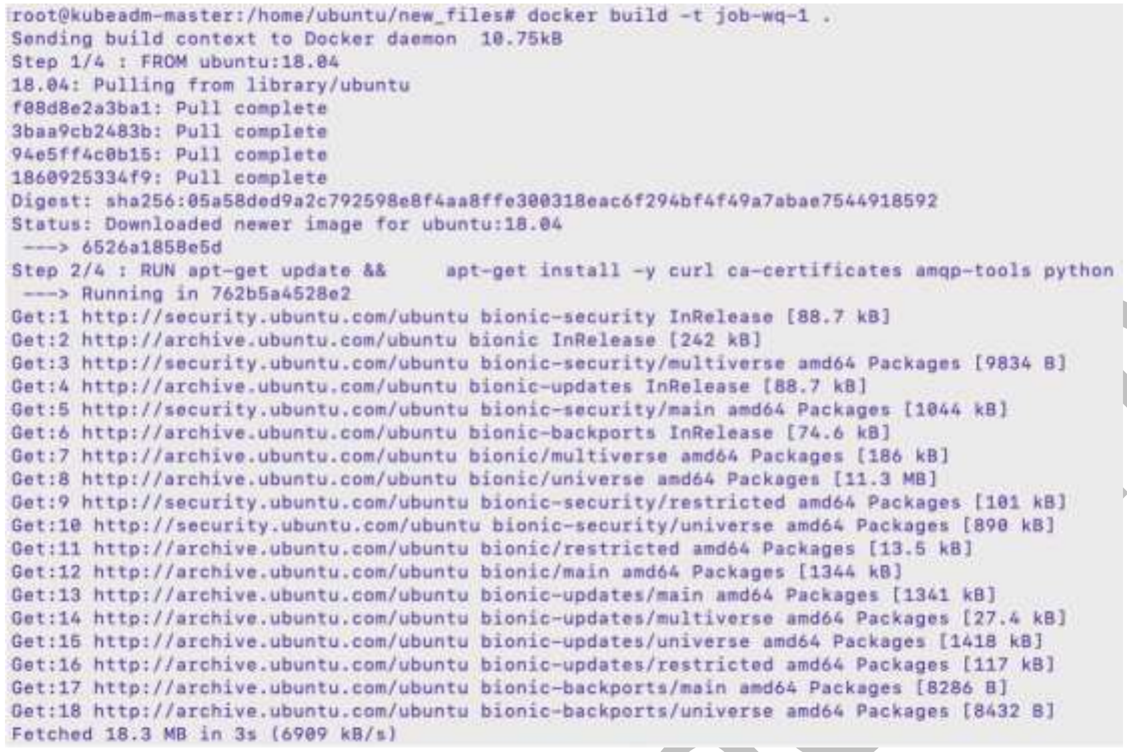
```
$ vim dockerfile-mq
```



```
# Specify BROKER_URL and QUEUE when running
FROM ubuntu:18.04

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 /worker.py
~
~
~
~
~
```

3. Create image using the above shown dockerfile

```
$ docker build -t job-wq-1 -f dockerfile-mq .
```

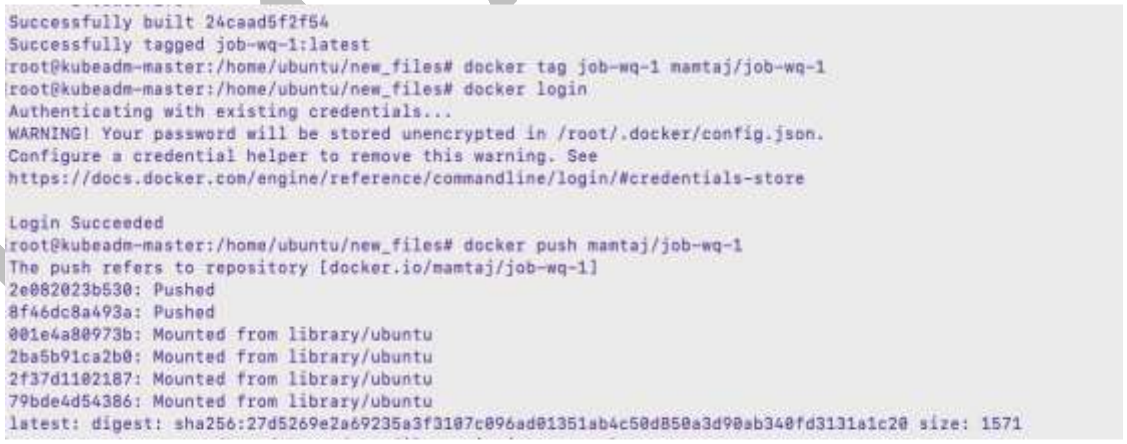


```
root@kubeadm-master:/home/ubuntu/new_files# docker build -t job-wq-1 .
Sending build context to Docker daemon  10.75kB
Step 1/4 : FROM ubuntu:18.04
18.04: Pulling from library/ubuntu
f08d8e2a3ba1: Pull complete
3baa9cb2483b: Pull complete
94e5ff4c0b15: Pull complete
1860925334f9: Pull complete
Digest: sha256:05a58ded9a2c792598e8f4aa8ffe300318eac6f294bf4f49a7abae7544918592
Status: Downloaded newer image for ubuntu:18.04
 ---> 6526a1858e5d
Step 2/4 : RUN apt-get update &&     apt-get install -y curl ca-certificates amqp-tools python
 ---> Running in 762b5a4528e2
Get:1 http://security.ubuntu.com/ubuntu bionic-security InRelease [88.7 kB]
Get:2 http://archive.ubuntu.com/ubuntu bionic InRelease [242 kB]
Get:3 http://security.ubuntu.com/ubuntu bionic-security/multiverse amd64 Packages [9834 B]
Get:4 http://archive.ubuntu.com/ubuntu bionic-updates InRelease [88.7 kB]
Get:5 http://security.ubuntu.com/ubuntu bionic-security/main amd64 Packages [1044 kB]
Get:6 http://archive.ubuntu.com/ubuntu bionic-backports InRelease [74.6 kB]
Get:7 http://archive.ubuntu.com/ubuntu bionic/multiverse amd64 Packages [186 kB]
Get:8 http://archive.ubuntu.com/ubuntu bionic/universe amd64 Packages [11.3 MB]
Get:9 http://security.ubuntu.com/ubuntu bionic-security/restricted amd64 Packages [101 kB]
Get:10 http://security.ubuntu.com/ubuntu bionic-security/universe amd64 Packages [890 kB]
Get:11 http://archive.ubuntu.com/ubuntu bionic/restricted amd64 Packages [13.5 kB]
Get:12 http://archive.ubuntu.com/ubuntu bionic/main amd64 Packages [1344 kB]
Get:13 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 Packages [1341 kB]
Get:14 http://archive.ubuntu.com/ubuntu bionic-updates/multiverse amd64 Packages [27.4 kB]
Get:15 http://archive.ubuntu.com/ubuntu bionic-updates/universe amd64 Packages [1418 kB]
Get:16 http://archive.ubuntu.com/ubuntu bionic-updates/restricted amd64 Packages [117 kB]
Get:17 http://archive.ubuntu.com/ubuntu bionic-backports/main amd64 Packages [8286 B]
Get:18 http://archive.ubuntu.com/ubuntu bionic-backports/universe amd64 Packages [8432 B]
Fetched 18.3 MB in 3s (6909 kB/s)
```

4. For the Docker Hub, tag your app image with your username and push to the Hub with the below commands. Replace <username> with your Hub username.

5. Login to Docker Hub and Push you image.

```
$ docker tag job-wq-1 mamtaj/job-wq-1
$ docker login
$ docker push mamtaj/job-wq-1
```

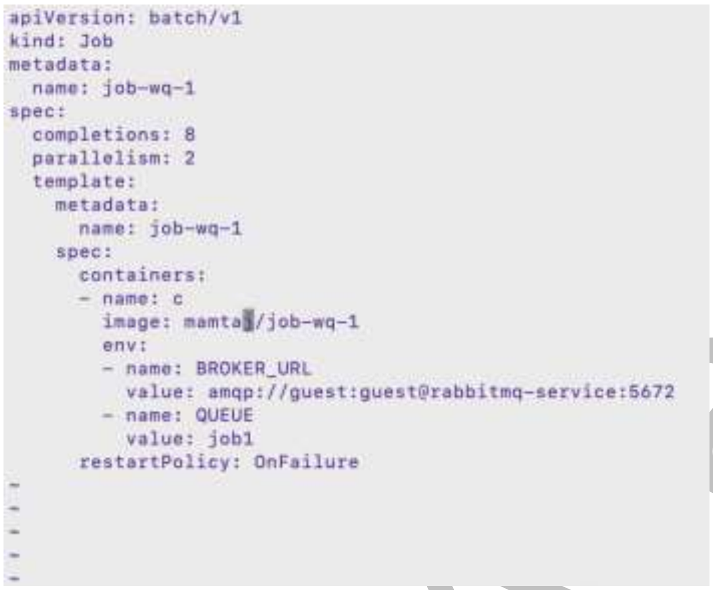


```
Successfully built 24caad5f2f54
Successfully tagged job-wq-1:latest
root@kubeadm-master:/home/ubuntu/new_files# docker tag job-wq-1 mamtaj/job-wq-1
root@kubeadm-master:/home/ubuntu/new_files# docker login
Authenticating with existing credentials...
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
root@kubeadm-master:/home/ubuntu/new_files# docker push mamtaj/job-wq-1
The push refers to repository [docker.io/mamtaj/job-wq-1]
2e082023b530: Pushed
8f46dc8a493a: Pushed
001e4a80973b: Mounted from library/ubuntu
2ba5b91ca2b0: Mounted from library/ubuntu
2f37d1102187: Mounted from library/ubuntu
79bde4d54386: Mounted from library/ubuntu
latest: digest: sha256:27d5269e2a69235a3f3107c096ad01351ab4c50d850a3d90ab340fd3131a1c20 size: 1571
```

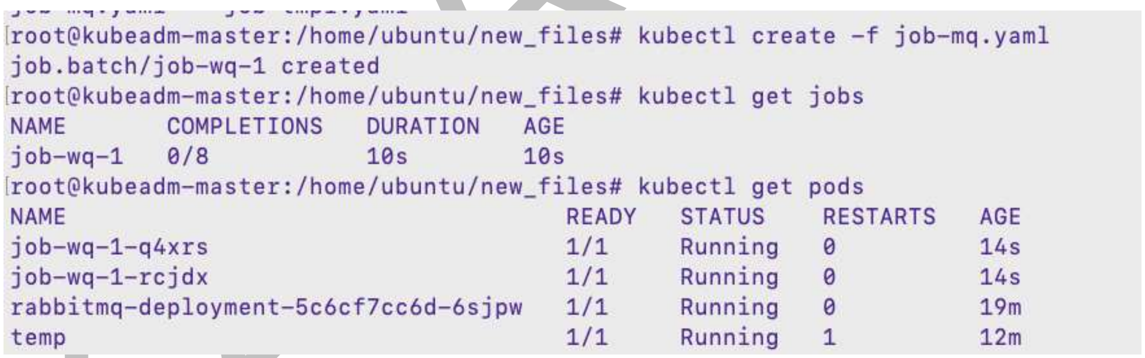
6.6 Create Job and Pod using the image pushed to Docker Hub

1. Create job to work on the created task queue. Open the job-mq.yaml file and edit the image to match the name you have created.

```
$ vim job-mq.yaml

$ kubectl create -f job-mq.yaml
```



```
apiVersion: batch/v1
kind: Job
metadata:
  name: job-wq-1
spec:
  completions: 8
  parallelism: 2
  template:
    metadata:
      name: job-wq-1
    spec:
      containers:
      - name: c
        image: mamta█/job-wq-1
        env:
        - name: BROKER_URL
          value: amqp://guest:guest@rabbitmq-service:5672
        - name: QUEUE
          value: job1
      restartPolicy: OnFailure
~
~
~
~
~
```



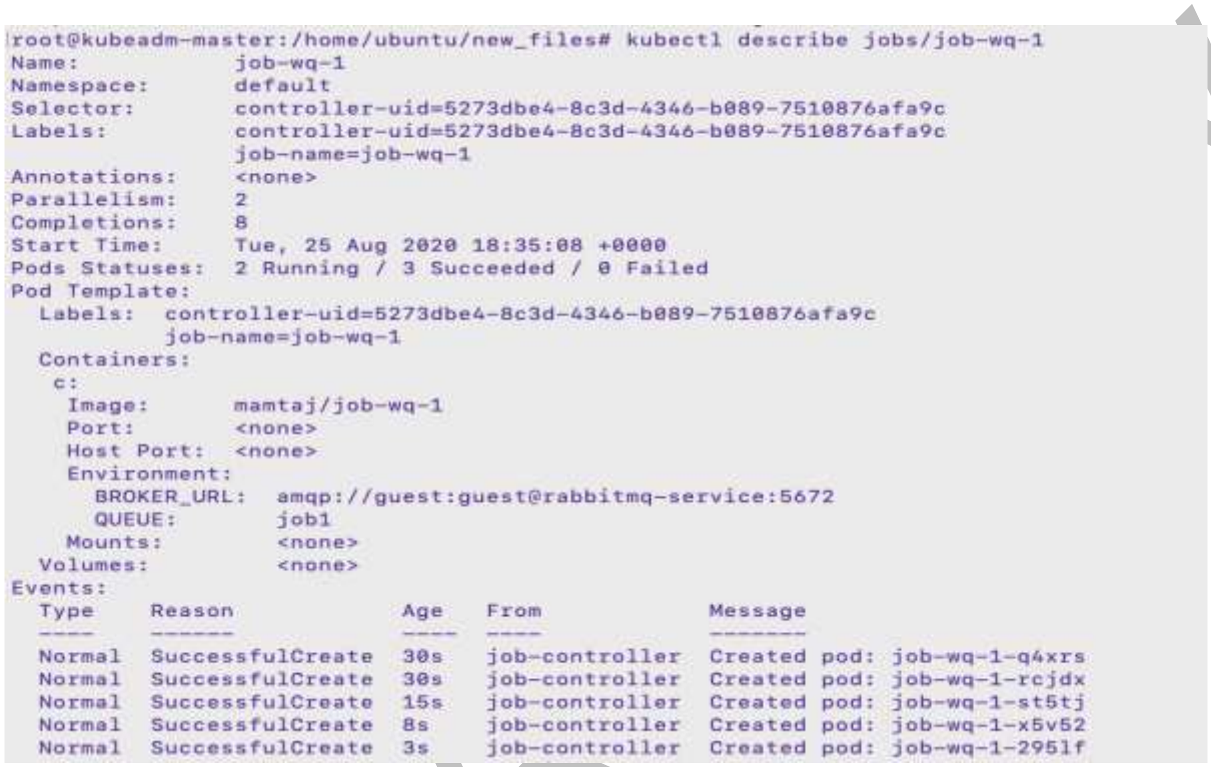
```
root@kubeadm-master:/home/ubuntu/new_files# kubectl create -f job-mq.yaml
job.batch/job-wq-1 created
root@kubeadm-master:/home/ubuntu/new_files# kubectl get jobs
NAME       COMPLETIONS   DURATION   AGE
job-wq-1   0/8           10s        10s
root@kubeadm-master:/home/ubuntu/new_files# kubectl get pods
NAME                                   READY   STATUS    RESTARTS   AGE
job-wq-1-q4xrs                         1/1     Running   0          14s
job-wq-1-rcjdx                         1/1     Running   0          14s
rabbitmq-deployment-5c6cf7cc6d-6sjpw   1/1     Running   0          19m
temp                                   1/1     Running   1          12m
```

6. Describe the job and see all the details. Completions is 8, Parallelism is marked as 2. At one point in time 2 jobs can run in parallel, process the message from queue and mark as complete.

7. Verify that the status of the job shows as running as all messages of the queue is yet not processed.

8. Few pods show as succeeded and few are in running state. Few of them have yet not triggered.

```
$ kubectl describe jobs/job-wq-1
```

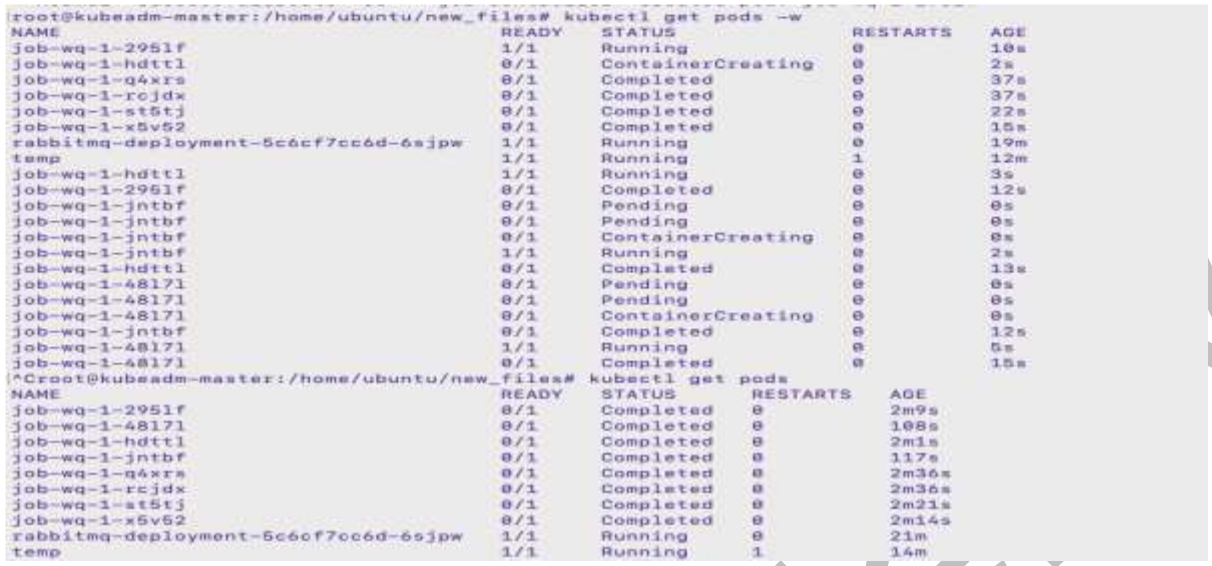


```
root@kubeadm-master:/home/ubuntu/new_files# kubectl describe jobs/job-wq-1
Name:            job-wq-1
Namespace:       default
Selector:        controller-uid=5273dbe4-8c3d-4346-b089-7510876afa9c
Labels:          controller-uid=5273dbe4-8c3d-4346-b089-7510876afa9c
                 job-name=job-wq-1
Annotations:     <none>
Parallelism:     2
Completions:     8
Start Time:      Tue, 25 Aug 2020 18:35:08 +0000
Pods Statuses:   2 Running / 3 Succeeded / 0 Failed
Pod Template:
  Labels:  controller-uid=5273dbe4-8c3d-4346-b089-7510876afa9c
           job-name=job-wq-1
  Containers:
   c:
    Image:       mamtaj/job-wq-1
    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  30s   job-controller  Created pod: job-wq-1-q4xrs
  Normal  SuccessfulCreate  30s   job-controller  Created pod: job-wq-1-rcjdx
  Normal  SuccessfulCreate  15s   job-controller  Created pod: job-wq-1-st5tj
  Normal  SuccessfulCreate  8s    job-controller  Created pod: job-wq-1-x5v52
  Normal  SuccessfulCreate  3s    job-controller  Created pod: job-wq-1-2951f
```

9. Execute watch command on pod and see that new pods get created and on processing the message are marked as completed.

10. After all pods have marked completed, list and see that 8 pods were created as there were 8 tasks in the message queue.

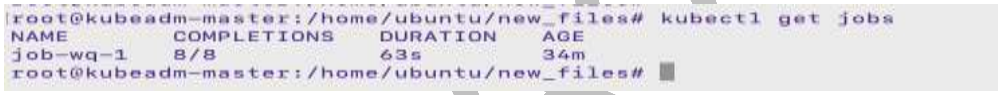
```
$ kubectl get pods -w
$ kubectl get pods
```



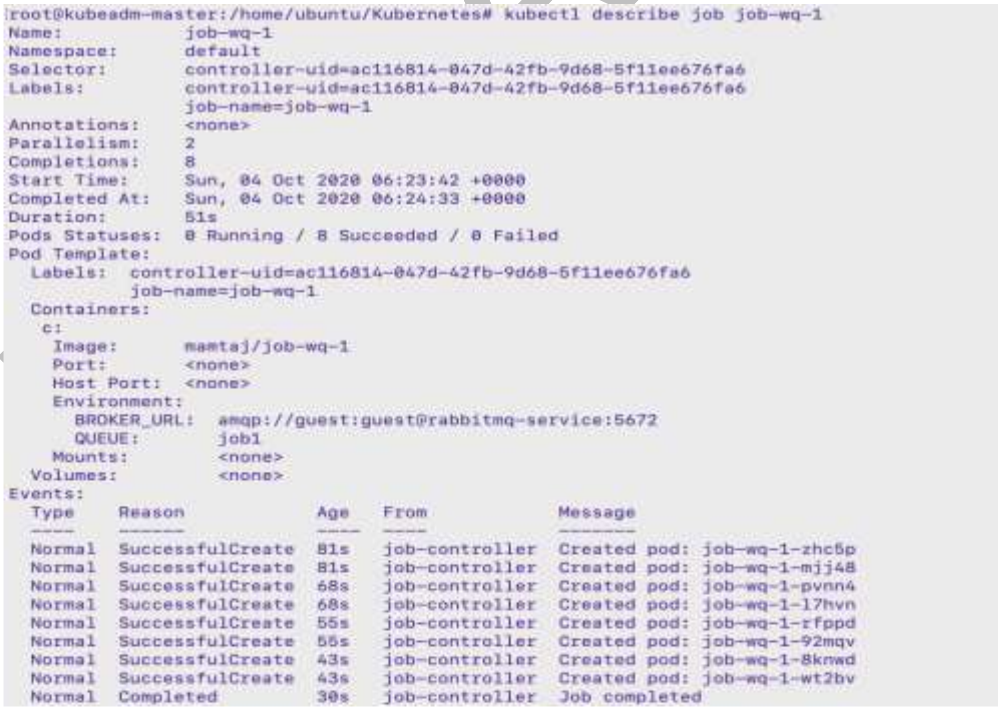
```
root@kubeadm-master:/home/ubuntu/new_files# kubectl get pods -w
NAME                                  READY   STATUS              RESTARTS   AGE
job-wq-1-2951f                        1/1     Running             0          10s
job-wq-1-hdttl                        0/1     ContainerCreating   0          2s
job-wq-1-q4xrs                        0/1     Completed           0          37s
job-wq-1-rcjdx                        0/1     Completed           0          37s
job-wq-1-st5tj                        0/1     Completed           0          22s
job-wq-1-x5v52                        0/1     Completed           0          15s
rabbitmq-deployment-5c6cf7cc6d-6sjpw  1/1     Running             0          19m
temp                                  1/1     Running             1          12m
job-wq-1-hdttl                        1/1     Running             0          3s
job-wq-1-2951f                        0/1     Completed           0          12s
job-wq-1-jntbf                        0/1     Pending             0          0s
job-wq-1-jntbf                        0/1     Pending             0          0s
job-wq-1-jntbf                        0/1     ContainerCreating   0          0s
job-wq-1-jntbf                        1/1     Running             0          2s
job-wq-1-hdttl                        0/1     Completed           0          13s
job-wq-1-48171                        0/1     Pending             0          0s
job-wq-1-48171                        0/1     Pending             0          0s
job-wq-1-48171                        0/1     ContainerCreating   0          0s
job-wq-1-jntbf                        0/1     Completed           0          12s
job-wq-1-48171                        1/1     Running             0          5s
job-wq-1-48171                        0/1     Completed           0          15s
^Croot@kubeadm-master:/home/ubuntu/new_files# kubectl get pods
NAME                                  READY   STATUS      RESTARTS   AGE
job-wq-1-2951f                        0/1     Completed   0          2m9s
job-wq-1-48171                        0/1     Completed   0          108s
job-wq-1-hdttl                        0/1     Completed   0          2m1s
job-wq-1-jntbf                        0/1     Completed   0          117s
job-wq-1-q4xrs                        0/1     Completed   0          2m36s
job-wq-1-rcjdx                        0/1     Completed   0          2m36s
job-wq-1-st5tj                        0/1     Completed   0          2m21s
job-wq-1-x5v62                        0/1     Completed   0          2m14s
rabbitmq-deployment-5c6cf7cc6d-6sjpw  1/1     Running     0          21m
temp                                  1/1     Running     1          14m
```

11. After all pods are marked completed, list the job and verify that its marked as 8/8 completed. Describe the job, verify that 8 pods show succeeded status and is marked completed in the event section.

```
$ kubectl get jobs

$ kubectl describe jobs/job-wq-1
```



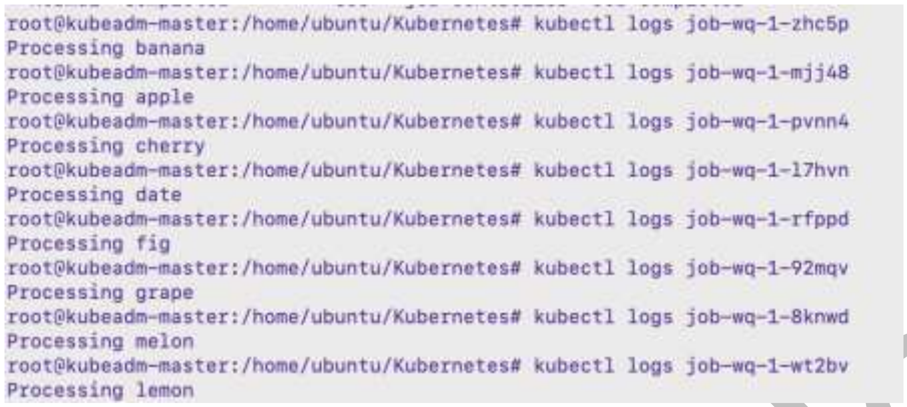
```
root@kubeadm-master:/home/ubuntu/new_files# kubectl get jobs
NAME       COMPLETIONS   DURATION   AGE
job-wq-1   8/8           63s        34m
root@kubeadm-master:/home/ubuntu/new_files#
```



```
root@kubeadm-master:/home/ubuntu/Kubernetes# kubectl describe job job-wq-1
Name:           job-wq-1
Namespace:      default
Selector:       controller-uid=ac116814-047d-42fb-9d68-5f11ee676fa6
Labels:         controller-uid=ac116814-047d-42fb-9d68-5f11ee676fa6
                job-name=job-wq-1
Annotations:    <none>
Parallelism:    2
Completions:    8
Start Time:     Sun, 04 Oct 2020 06:23:42 +0000
Completed At:   Sun, 04 Oct 2020 06:24:33 +0000
Duration:       51s
Pods Statuses:  0 Running / 8 Succeeded / 0 Failed
Pod Template:
  Labels:  controller-uid=ac116814-047d-42fb-9d68-5f11ee676fa6
           job-name=job-wq-1
  Containers:
   c:
    Image:      mamtaj/job-wq-1
    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  81s   job-controller  Created pod: job-wq-1-zhc5p
  Normal  SuccessfulCreate  81s   job-controller  Created pod: job-wq-1-mjj48
  Normal  SuccessfulCreate  68s   job-controller  Created pod: job-wq-1-pvnn4
  Normal  SuccessfulCreate  68s   job-controller  Created pod: job-wq-1-17hvn
  Normal  SuccessfulCreate  55s   job-controller  Created pod: job-wq-1-rfppd
  Normal  SuccessfulCreate  55s   job-controller  Created pod: job-wq-1-92mqv
  Normal  SuccessfulCreate  43s   job-controller  Created pod: job-wq-1-8knwd
  Normal  SuccessfulCreate  43s   job-controller  Created pod: job-wq-1-wt2bv
  Normal  Completed         30s   job-controller  Job completed
```

12. We can even check which task was picked up by which Pod by looking into logs of all the pods

```
$ kubectl logs <Pod Name>
```



```
root@kubeadm-master:/home/ubuntu/Kubernetes# kubectl logs job-wq-1-zhc5p
Processing banana
root@kubeadm-master:/home/ubuntu/Kubernetes# kubectl logs job-wq-1-mjj48
Processing apple
root@kubeadm-master:/home/ubuntu/Kubernetes# kubectl logs job-wq-1-pvnn4
Processing cherry
root@kubeadm-master:/home/ubuntu/Kubernetes# kubectl logs job-wq-1-l7hvn
Processing date
root@kubeadm-master:/home/ubuntu/Kubernetes# kubectl logs job-wq-1-rfppd
Processing fig
root@kubeadm-master:/home/ubuntu/Kubernetes# kubectl logs job-wq-1-92mqv
Processing grape
root@kubeadm-master:/home/ubuntu/Kubernetes# kubectl logs job-wq-1-8knwd
Processing melon
root@kubeadm-master:/home/ubuntu/Kubernetes# kubectl logs job-wq-1-wt2bv
Processing lemon
```

6.7 Clean-up the resources

1. Clean-up: Delete the all Jobs created in this task with single kubectl command

```
$ kubectl delete job job-wq-1
$ kubectl delete pod temp
$ kubectl delete -f rabbitmq-service.yaml
$ kubectl delete -f rabbitmq-deployment.yaml
```

7 SUMMARY

In this guide we Covered:

5. Working with Cronjob
   - Creating cronjob to execute on a scheduled time
   - Verifying and analysing the created cronjob
   - Clean-up the resources

6. Supervising Pods with Jobs
   - Creating kind job that supervises a pod counting from 9 to 1

7. Handling Batch processing with Prallel Job using Template
   - Creating Job based on a Template

8. Handling Coarse Parallel processing using Message Queue
   - Exercise Description
   - Starting a Message Queue service
   - Testing The Message Queue service
   - Filling the Queue with tasks
   - Create an image for the Job and Push to Docker Hub
   - Create Job and Pod to process task from Message Queue