Deploying Term Frequency Application in Kubernetes

Team Members: [redacted]

I. Project Proposal:

The intent of the project is to port the PySpark application for finding Term Frequency (TF) to Kubernetes through the concept of containerization using Docker. In the previous project proposal submission, we have also included that we would deploy the Inverse Document Frequency (IDF) in the Kubernetes but we had constrained it only to Term Frequency (TF) Deployment.

In order to develop this application, we have constructed a docker file that builds a container which consists of all the required dependencies of the application that is to be deployed and a "YAML" file which is used to deploy the created container to Kubernetes.

II. Dataset Description:

The dataset that we have used is "SimpleWikiData" which consists of the Wikipedia articles and has been extracted from the Wikimedia website. The "XML" files obtained are preprocessed into text files. The final data set consists of 30 text files generated which comprises a total memory of 700MB. But as the project only requires the data set range between 50MB to 100 MB, we have chosen the first 3 text files for our project and constructed the term frequency index and performed query processing on the created term frequency index.

The input text files consist of two user defined tag elements called as the "Title" and its associated "Text" tag. Each "Title" tag is unique representing a Wikipedia article and its page content is stored in a "Text" tag.

Dataset Source:

https://dumps.wikimedia.org/simplewiki/20200301/simplewiki-20200301-pages-articles.xml.bz2

III. Project Description:

a. Term Frequency (TF): The number of times a term occurs in a document.

TF = (Number of time the word occurs in the document) / (Total number of words in document)

Log-Weighted TF = 1 +log10 (TF)

b. Docker: It is a platform as a service product that uses OS level virtualization to deliver software packages called containers. The containers are isolated from one another and have their own bundle of libraries, software and configuration files which can communicate with each other through well-defined channels.

c. Dockerfile: It is a text document that contains all the commands a user could call the command line to assemble an image.

d. Container: It is a standard unit of software that packages up code and its all dependencies so the application runs quickly from one computing environment to another.

e. Kubernetes: It is an open-source container orchestration system for automating application deployment, scaling, and management.

f. YAML (Yet Another Markup Language): It is a human-readable data serialization standard that can be used in conjunction with all programming languages and is often used to write configuration files. Kubernetes resources, such as pods, services, and deployments are created by using the YAML files.

IV. Results:

Our Project folder consists of the following files. All files must be in the same level as that of docker file.

1. **Dockerfile**: Standard convention to name the file is Dockerfile.The docker file here pulls Ubuntu images from docker hub and sets up the environment by installing necessary software such as python,java and spark into it. It also downloads the wikipedia article files required for our data creation. Move the required execution files into the environment. Project is kick started with the CMD command which invokes the run.sh shell script which inturn invokes all other files to be executed.
2. **run.sh:** Shell script file which contains basic python commands to run the python script as well as linux commands to create folders and move files into it.
3. **Parse_wikipedia_data.py:** Python script required to parse the downloaded wikipedia articles to text files
4. **TF1.py:** Python spark program written to preprocess the text files and creates a log-weighted term frequency index for the terms in the document.
5. **TF1_query.py:** Python spark program written to query the documents on the index created to retrieve relevant files matching the query.
6. **Sample.txt**: Contains the user inputs to be queried on the created index

A. Steps to Build a Container:

(i) Docker Build: Build the docker file to create a image using the docker build command

Command: docker build . -t image_name

```
nandh@LAPTOP-00N7TG82 MINGW64 ~/OneDrive/Desktop/CloudComputing/Project
$ docker build . -t test-pyspark
Sending build context to Docker daemon  16.38kB
Step 1/29 : FROM ubuntu
 ---> 4e5021d210f6
Step 2/29 : MAINTAINER  Raj
 ---> Using cache
 ---> f82a9641c937
Step 3/29 : RUN apt-get update
 ---> Using cache
 ---> c4425b8ce820
Step 4/29 : RUN apt-get install -y default-jdk default-jre
 ---> Using cache
 ---> 0e4f6c754556
Step 5/29 : RUN apt-get install -y python3-pip python3-dev   && cd /usr/local/bin   && ln -s /usr/bin/python3 python   && pip3 install --upgrade pip
 ---> Using cache
 ---> 2f03cf3c1267
Step 6/29 : RUN apt-get -y install vim
 ---> Using cache
 ---> 172f8f725aaf
Step 7/29 : RUN apt-get install -y wget dos2unix
 ---> Using cache
 ---> f2c72aa6030f
```

```
 ---> 4ec8286d63d1
Step 22/29 : COPY ./parse_wikipedia_data.py /code
 ---> Using cache
 ---> dddda9384e98
Step 23/29 : COPY ./TF1.py /code
 ---> Using cache
 ---> 0d137dc30edd
Step 24/29 : COPY ./TF1_query.py /code
 ---> Using cache
 ---> 5cb8e58af55e
Step 25/29 : COPY ./Sample.txt /code
 ---> Using cache
 ---> 66f705a93cdb
Step 26/29 : COPY ./run.sh /code
 ---> Using cache
 ---> 6dbc464e57ed
Step 27/29 : RUN chmod +x /code/run.sh
 ---> Using cache
 ---> 9bfc8e5f53c3
Step 28/29 : RUN dos2unix /code/run.sh
 ---> Using cache
 ---> bf666a274fd8
Step 29/29 : CMD /code/run.sh
 ---> Using cache
 ---> 29daa79d83e3
Successfully built 29daa79d83e3
Successfully tagged test-pyspark:latest
SECURITY WARNING: You are building a Docker image from Windows against a non-Windows Docker host. All files and directories added to build context will have '-rwxr-xr-x
' permissions. It is recommended to double check and reset permissions for sensitive files and directories.
```

(ii) Docker Images: View the created image using docker images command in the terminal

```
nandh@LAPTOP-00N7TG82 MINGW64 ~/OneDrive/Desktop/CloudComputing/Project
$ docker images
REPOSITORY              TAG                 IMAGE ID            CREATED             SIZE
test-pyspark            latest              29daa79d83e3        5 days ago          1.88GB
nvelusw/tfindex-query   1.0.0               29daa79d83e3        5 days ago          1.88GB
ubuntu                  latest              4e5021d210f6        4 weeks ago         64.2MB
```

(iii) Docker Run: Use docker run command to create the container with the built docker image.**"-itd"** option in docker run is used to create an interactive shell for the created container.

Command: docker run -itd --name container_name image_name

```
nandh@LAPTOP-00N7TG82 MINGW64 ~/OneDrive/Desktop/CloudComputing/Project
$ docker run -itd --name test-pyspark-container1 test-pyspark
65e69384d779704090c1164f595e02524d6416c2ced2aa36529c961db9bf95e1
```

(iv) Docker ps: Used docker ps command to view active container. Running the command with -a option list all the containers in the docker

```
nandh@LAPTOP-00N7TG82 MINGW64 ~/OneDrive/Desktop/CloudComputing/Project
$ docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS          PORTS     NAMES
65e69384d779   test-pyspark   "/bin/sh -c /code/ru…"   4 minutes ago    Up 4 minutes              test-pyspark-container1

nandh@LAPTOP-00N7TG82 MINGW64 ~/OneDrive/Desktop/CloudComputing/Project
$ docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS          PORTS     NAMES
65e69384d779   test-pyspark   "/bin/sh -c /code/ru…"   6 minutes ago    Up 6 minutes              test-pyspark-container1
```

(v) Docker exec: Docker exec command is used for logging into the interactive shell created for the container. Exit command is used to exit from it.

```
nandh@LAPTOP-00N7TG82 MINGW64 ~/OneDrive/Desktop/CloudComputing/Project
$ docker exec -it  test-pyspark-container1 sh
# ls
'TFQID - 1'   TFQuery.out   bin    datacopy.out   home    media       opt            project_TFIndex   run     spark-3.0.0-preview2-bin-hadoop2.7.tgz   tmp
'TFQID - 2'   TFindex.out   boot   dev            lib     mnt         parsedata.out  projectdata       sbin    srv                                      usr
'TFQID - 3'   app_data      code   etc            lib64   nohup.out   proc           root              spark   sys                                      var
# exit
```

(vi) Docker rm: Docker rm command to remove the container created. Run with -f option to forcefully remove the running container and -v to remove the anonymous volume created by the container.

Command: docker rm -f -v container_name

```
nandh@LAPTOP-00N7TG82 MINGW64 ~/OneDrive/Desktop/CloudComputing/Project
$ docker rm -f -v test-pyspark-container1
test-pyspark-container1
```

(vii) Docker rmi: Docker rmi command is used to remove the image created.Run with -f option to forcefully remove an image.

Command: docker rmi -image_name

```
nandh@LAPTOP-00N7TG82 MINGW64 ~/OneDrive/Desktop/CloudComputing/Project
$ docker rmi ubuntu
Untagged: ubuntu:latest
Untagged: ubuntu@sha256:bec5a2727be7fff3d308193cfde3491f8fba1a2ba392b7546b43a051853a341d
```

B. Steps to Deploy a Container to kubernetes (K8S):

(i) YAML File: Contains information about the image to be pulled for deployment, and deployment identifier, no of replicas (i.e) the replication factor and container information such as container name, size etc . Below is the screenshot of the YAML file used for deployment

```
feifei@feifeideMacBook-Air DockerTest % cat test_Pyspark_deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: pyspark-deployment-new-v2
spec:
  selector:
    matchLabels:
      app: test-pyspark-new-v2
  replicas: 1
  template:
    metadata:
      labels:
        app: test-pyspark-new-v2
    spec:
      containers:
      - name: test-pyspark-dep-new-v2
        image: test-pyspark-new-v2
        imagePullPolicy: Never
        ports:
        - containerPort: 80
```

(ii) Creating a Kubernetes (K8S) Deployment: The below command is used to create a deployment for the associated YAML file.

```
feifei@feifeideMacBook-Air DockerTest % ls
Project                     test_Pyspark_deployment.yaml
feifei@feifeideMacBook-Air DockerTest % kubectl create -f test_Pyspark_deployment.yaml
deployment.apps/pyspark-deployment-new-v2 created
```

(iii) View the available deployments:

```
feifei@feifeideMacBook-Air DockerTest % kubectl get deployment
NAME                        READY   UP-TO-DATE   AVAILABLE   AGE
pyspark-deployment-new      1/1     1            1           15h
pyspark-deployment-new-v2   1/1     1            1           15m
feifei@feifeideMacBook-Air DockerTest %
feifei@feifeideMacBook-Air DockerTest %
```

(iv) View the created pods:

```
feifei@feifeideMacBook-Air DockerTest % kubectl get pod -o wide
NAME                                    READY   STATUS    RESTARTS   AGE   IP          NODE             NOMINATED NODE   READINESS GATES
pyspark-deployment-new-64969df59c-g9vg5   1/1     Running   0          15h   10.1.0.40   docker-desktop   <none>           <none>
pyspark-deployment-new-v2-86989586c4-wqk4k   1/1     Running   0          15m   10.1.0.41   docker-desktop   <none>           <none>
feifei@feifeideMacBook-Air DockerTest %
```

(v) Executing the Kubernetes Deployment File: The below command is used to execute the kubernetes deployment interactively.

```
feifei@feifeideMacBook-Air DockerTest % kubectl exec -ti pyspark-deployment-new-64969df59c-g9vg5   -- /bin/sh
# ls
'TFQID - 1'    TFindex.out   code           home    mnt            project_TFIndex   sbin                                  sys
'TFQID - 2'    app_data      datacopy.out   lib     opt            projectdata       spark                                 tmp
'TFQID - 3'    bin           dev            lib64   parsedata.out  root              spark-3.0.0-preview2-bin-hadoop2.7.tgz   usr
 TFQuery.out   boot          etc            media   proc           run               srv                                   var
```

(vi) Output of the Log Weighted Term Frequency Index: The below command displays the result of log weighted term frequency which is of the format,

Format:

"title1@word1#weight_termfreq1+word2#weight_termfreq2+.....wordn#weight_termfreqn
…..titlen@word1#weight_termfreq1+.....wordn#weight_termfreqn"

```
# ls
_SUCCESS  part-00000  part-00001  part-00002
# cat part-00000 | head -2
swiss@Wiliberg#1.0+Lausen#1.0+Holziken#1.0+Btterkinden#1.0+Anires#1.6020599913279623+Winznau#1.6020599913279623+Weinfelden#1.0+Aarau#1.6020599913279623+Gampel#1.0+Echic
hens#1.3010299956639813+November 18#1.0+Thurgau#1.3010299956639813+Ennenda#1.0+Anzonico#1.3010299956639813+Bleiken bei Oberdiessbach#1.0+Wikipedia:Simple talk/Archive 4
6#1.0+March 20#1.0+Burgistein#1.0+Corcelles-le-Jorat#1.0+Dagmersellen#1.0+Untergeri#1.0+Template:Swiss populations data CH-ZG/doc#1.3010299956639813+Capriasca#1.3010299
956639813+Bhl#1.0+Bulgaria at the Olympics#1.0+Habsburg, Switzerland#1.0+Fehraltorf#1.0+Plasselb#1.6020599913279623+Agiez#1.0+Saint-Martin, Fribourg#1.6020599913279623+
Dnikon#1.0+Forel (Lavaux)#1.0+Mervelier#1.0+Niederbuchsiten#1.0+Ependes, Fribourg#1.6020599913279623+Vernayaz#1.0+Widen#1.0+Chancy#1.6020599913279623+Wachseldorn#1.0+Co
rsier#1.6020599913279623+Lignires, Neuchtel#1.0+Saint-Sulpice, Neuchtel#1.0+La Chaux-de-Fonds#1.3010299956639813+List of Disney movies#1.0+Trimstein#1.0+Mastrils#1.0+Mo
nible#1.0+Ursenbach#1.0+Sant'Abbondio#1.3010299956639813+EulerMascheroni constant#1.0+Egerkingen#1.0+Teufen, Appenzell Ausserrhoden#1.0+Stein am Rhein#1.0+List of commo
n misconceptions#1.0+Raperswilen#1.0+Berg am Irchel#1.0+Reitnau#1.0+Denges#1.0+1955#1.0+Anwil#1.6020599913279623+Chtelat#1.0+Bressaucourt#1.0+Module:Citation/CS1/Config
```

(vii) Content of the query file: The below image shows the list of queries in "Sample.txt" that are given as input for query processing.

```
# cd code
# ls
Sample.txt  TF1.py  TF1_query.py  parse_wikipedia_data.py  run.sh
#
#
# cat Sample.txt
book major irish $@%^@^$@% HGSDFHKJAKL
book major influenced #$!#%!#
tolkien dunsany
```

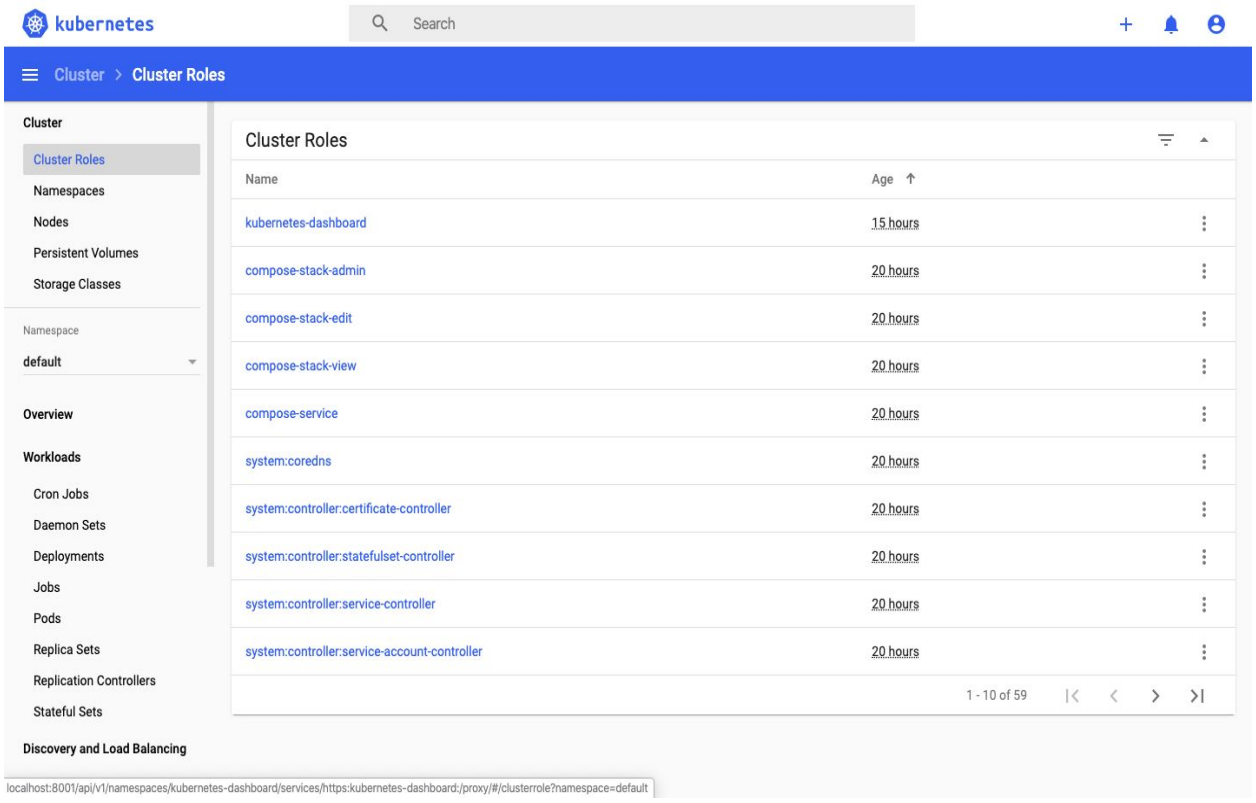
(viii) Output of the Term Frequency Querying: The below image displays the top 10 titles for the given query file i.e., for the query-1 in the "Sample.txt" file

```
# cd 'TFQID - 1'
# ls
_SUCCESS  part-00000
# cat part-00000
Jackie Robinson
List of characters in the Camp Half-Blood series
The Titan's Curse
Kamakura shogunate
Olympic Games
Cairo International Book Fair
Geophysics
Club de Gimnasia y Esgrima La Plata
History of the world
Ramayana
#
```

C. Kubernetes Dashboard:

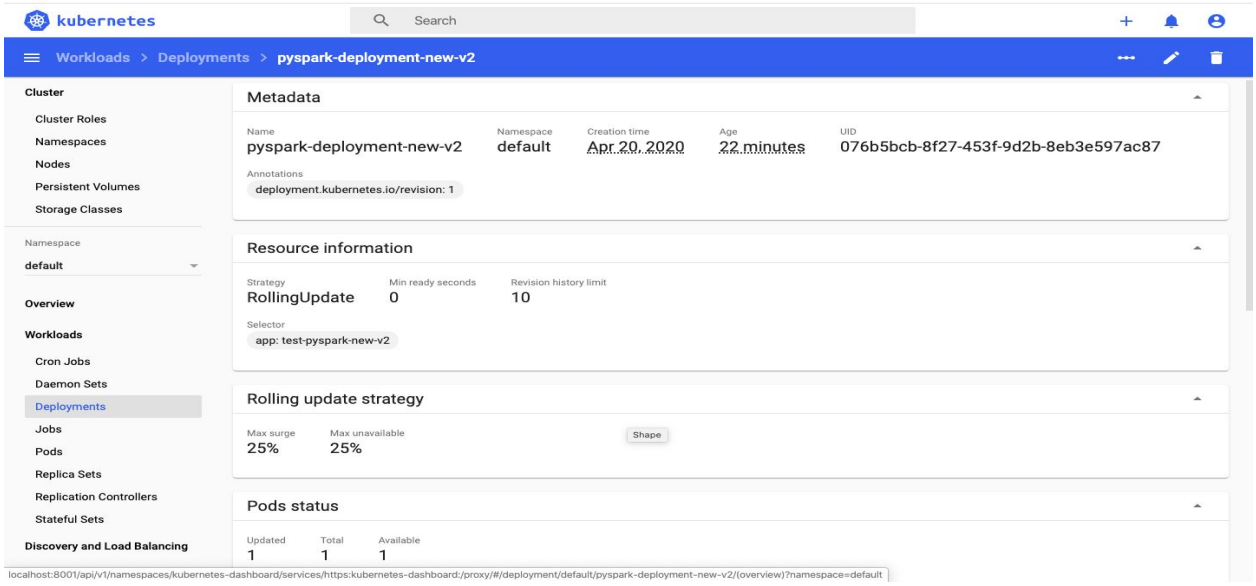
(i) Cluster Roles in Kubernetes:

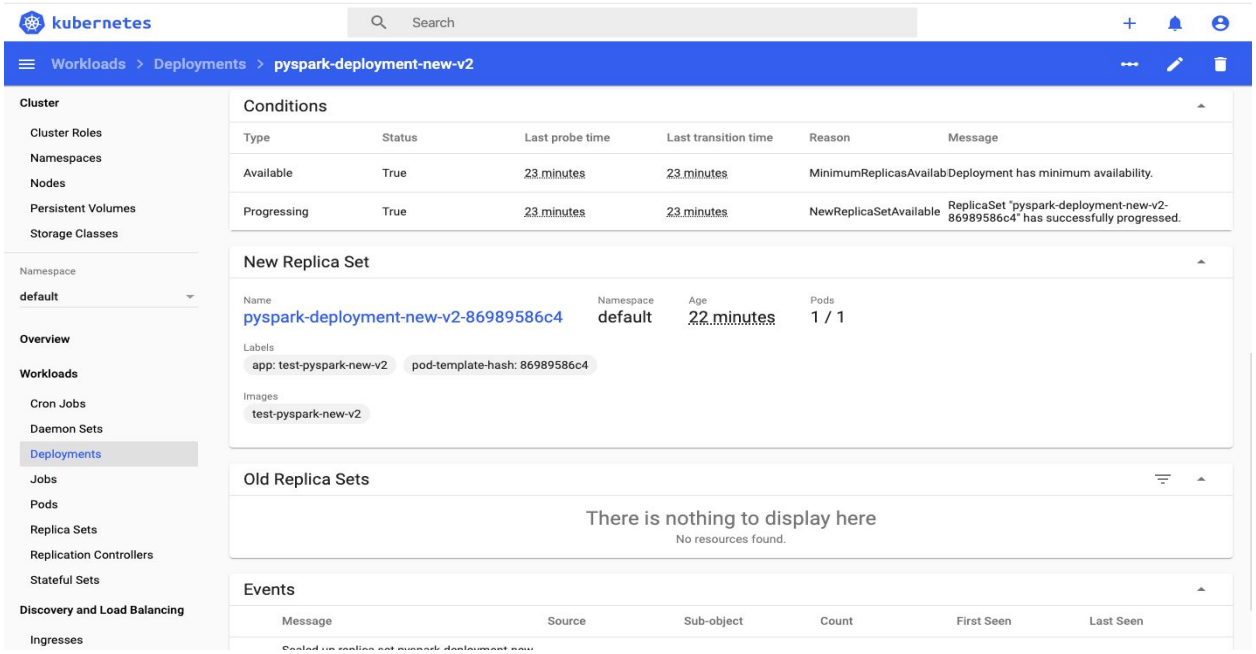
The below image is a kubernetes dashboard which displays the different cluster roles which allows superuser operations in all the cluster resources.Enables the user to have different permissions according to roles.



(ii) <u>Dashboard Contents:</u>

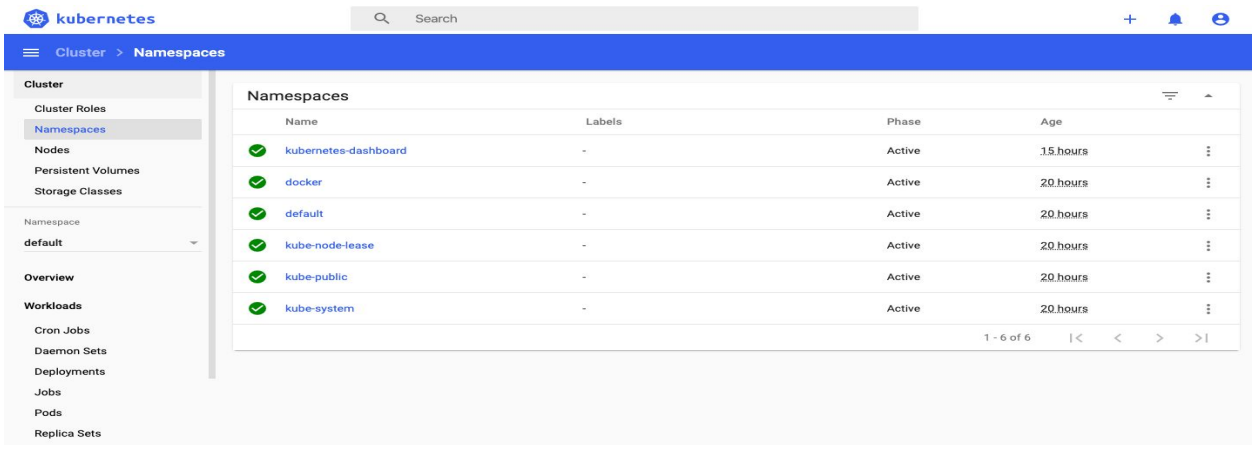
The below image explains about the contents of the newly created deployment named "pyspark-deployment-new-v2". It explains various sections such as resource information, rolling update strategy, pods status, the status/condition of the created deployment, replica sets and some associated features.





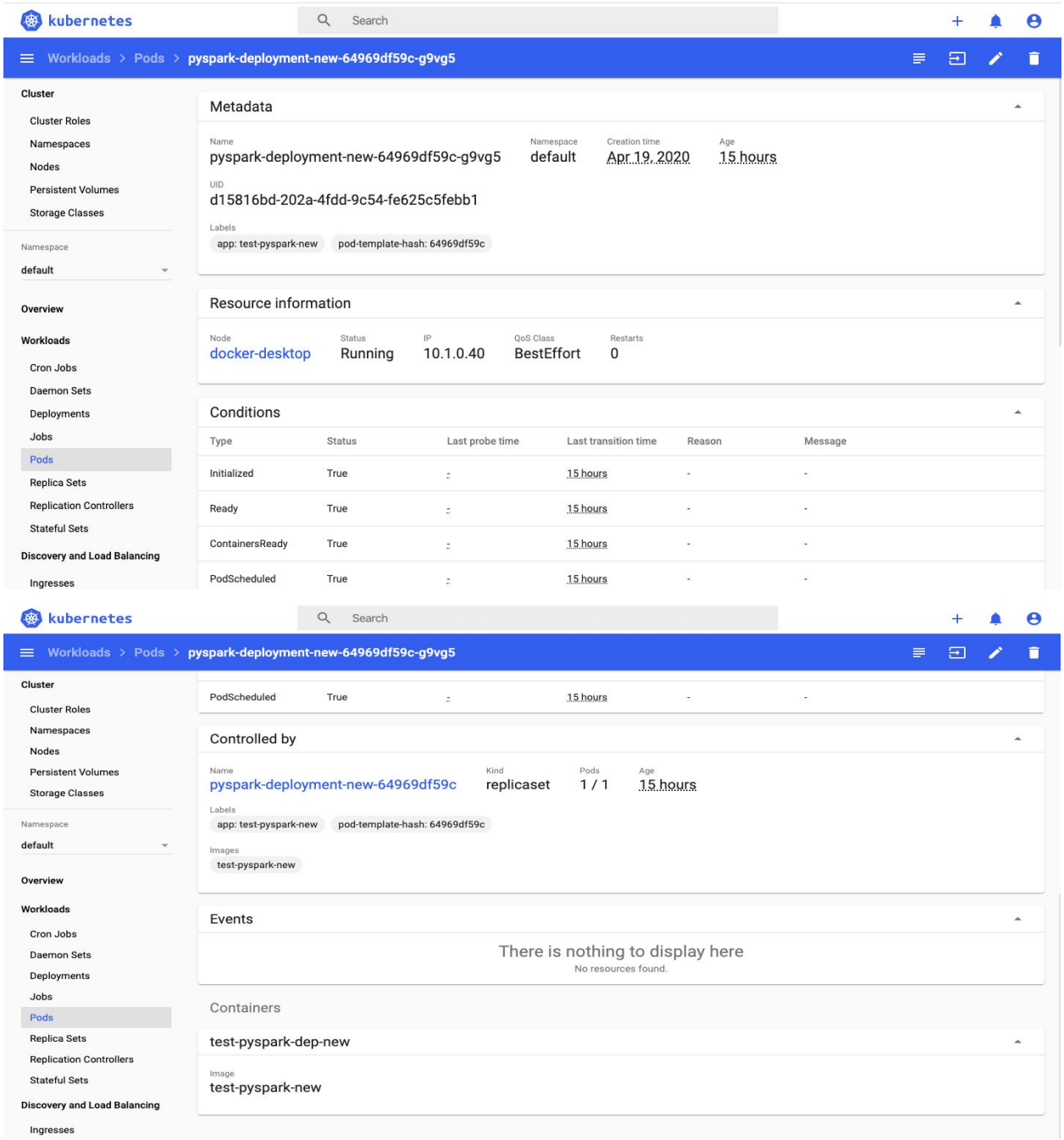
(iii) Namespaces:

The below image describes the available namespaces (virtual clusters) which are a way to divide cluster resources between multiple users. The namespace that we have used for the project is "default".



(iv) Pod Information:

The below image explains about the available data pods within the Kubernetes Dashboard. The Kubernetes Pod is a group of containers that are deployed together on the same host. A pod always runs on a node and a node can have multiple pods, and the Kubernetes master automatically handles scheduling the pods across the nodes in the cluster.



V. Conclusion and Future Scope:

Containerization of the application simplifies the project deployment and migration of the process. It eventually scales up the execution time for the process, it can control & automate all necessary deployments and updates, orchestrate containers on multiple hosts. Kubernetes simplifies the creation and management of services and machines within the cluster, thereby maximising the containerization effect. Deploying an application to kubernetes has the

advantages of efficient load balancing, scale up or down easily when required, and auto redeployment on failure.The execution time was quite efficient when compared to execution in a local machine.

The future work for this project would be to create a container to actively process query inputs through an User Interface, send the query to existing containers to retrieve the query results and these results obtained are sent back to the user. By this we could achieve communication between containers. We could also migrate this application to multiple cloud environments such as AWS (or) Azure for evaluating the performances.