**Building an Apache Spark Cluster using Kubernetes**

a. Section 1.2 Configuring Minikube

Building an Apache Spark cluster using Kubernetes

1)Created a working directory, mySparkCluster.

As default minikube configuration is not enough for running Spark applications, Spark recommends 3 CPUs and 4g of memory to be able to start a simple Spark application with a single executor. See prerequisites (<http://spark.apache.org/docs/latest/running-onkubernetes.html>)

We will use • CPU: 6 CPUs (recommended 3 CPU) • Memory: 14 GiB (recommended at least 4 GiB)

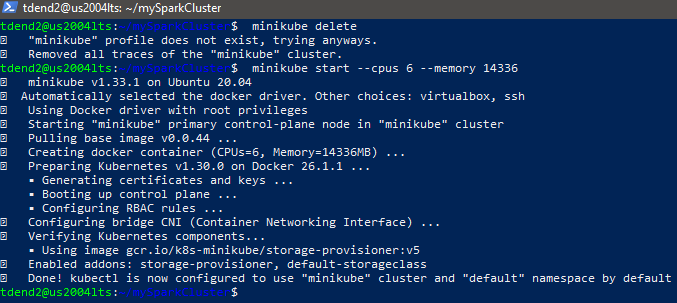
2)Passed the memory and CPU options to the minikube start command as follows:

*minikube start --cpus 6 --memory 14336*

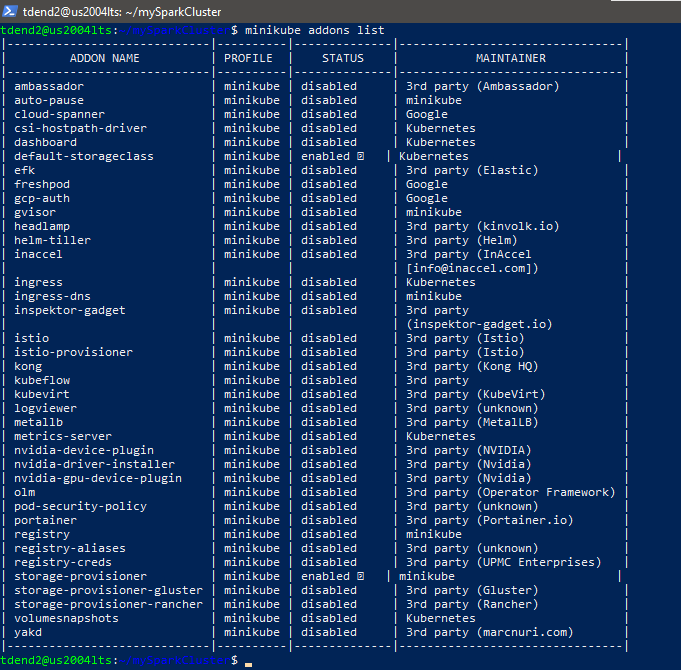
Or we may use the minikube config command like below:

*minikube config set memory 14336*

*minikube config set cpus 6*



Minikube addon list is shown below:*minikube addons list*



default-storageclass | minikube | enabled and storage-provisioner | minikube | enabled

—--------------------------------------------------------------------------------------------------------------------------

**b. Section 1.3 Creating Spark Docker Images**

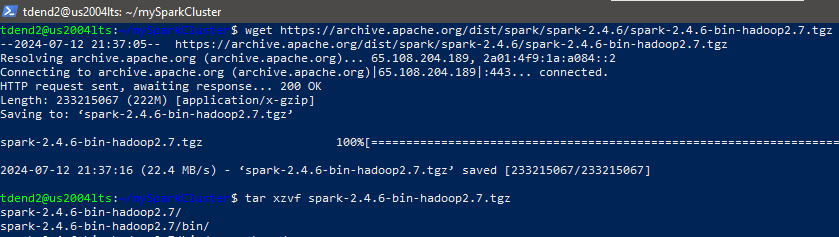
1.3.1 Downloading Apache Spark Official download site https://spark.apache.org/downloads.html

Note: Not recommended Spark 3.x.x for this exercise. Used version 2.4.6, 2.4.x, or minor updates. We will use the archive link to download it. https://archive.apache.org/dist/spark/ Copy the link address below (or find it using the archive link above), then download it using the ‘wget’ command. <https://archive.apache.org/dist/spark/spark-2.4.6/spark-2.4.6-bin-hadoop2.7.tgz>

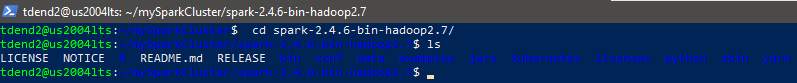
as below:

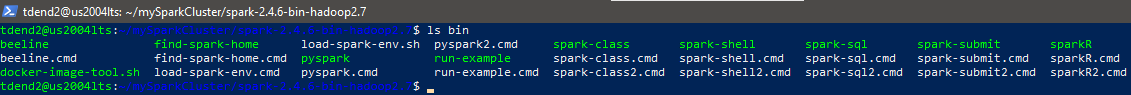
*wget https://archive.apache.org/dist/spark/spark-2.4.6/spark-2.4.6-bin-hadoop2.7.tgz*

And unzipped the spark distribution: *tar xzvf spark-2.4.6-bin-hadoop2.7.tgz*



Inside the extracted spark folder, checked the shell script for building spark-docker image: docker-image-tool.sh





1.3.2 Compatibility issue and modifying the Docker file

Currently, Apache Spark Kubernetes has a compatibility issue. Bugs in okhttp of fabric8io need to be fixed. See https://github.com/fabric8io/kubernetes-client/issues/2145, https://github.com/fabric8io/kubernetes-client/issues/2168, https://github.com/fabric8io/kubernetes-client/issues/2175, etc. Until the bugs in okhttp are fixed, we need to be careful on choosing the Java version. So, we will modify the Docker file like below. Opened the Dockerfile and updated it to use an openjdk:8u242-slim image which is based on OpenJDK 1.8.0\_242. The original image is based on OpenJDK 1.8.0\_252. Note: Not added the highlighted comments in the Dockerfile.

*vi kubernetes/dockerfiles/spark/Dockerfile*

*#FROM openjdk:8-jdk-slim <==Java 1.8.0\_252 Docker image / Comment this line*

*FROM openjdk:8u242-slim <==Java 1.8.0\_242 Docker image / Add this line!*

*ARG spark\_jars=jars*

*ARG img\_path=kubernetes/dockerfiles*

*ARG k8s\_tests=kubernetes/tests*

*RUN set -ex && \*

*apt-get update && \*

*ln -s /lib /lib64 && \*

*apt install -y bash tini libc6 libpam-modules libnss3 && \*

*mkdir -p /opt/spark && \*

*mkdir -p /opt/spark/work-dir && \*

*touch /opt/spark/RELEASE && \*

*rm /bin/sh && \*

*ln -sv /bin/bash /bin/sh && \*

*echo "auth required pam\_wheel.so use\_uid" >> /etc/pam.d/su && \*

*chgrp root /etc/passwd && chmod ug+rw /etc/passwd && \*

*rm -rf /var/cache/apt/\**

*COPY ${spark\_jars} /opt/spark/jars*

*COPY bin /opt/spark/bin*

*COPY sbin /opt/spark/sbin*

*COPY ${img\_path}/spark/entrypoint.sh /opt/*

*COPY examples /opt/spark/examples*

*COPY ${k8s\_tests} /opt/spark/tests*

*COPY data /opt/spark/data*

*ENV SPARK\_HOME /opt/spark*

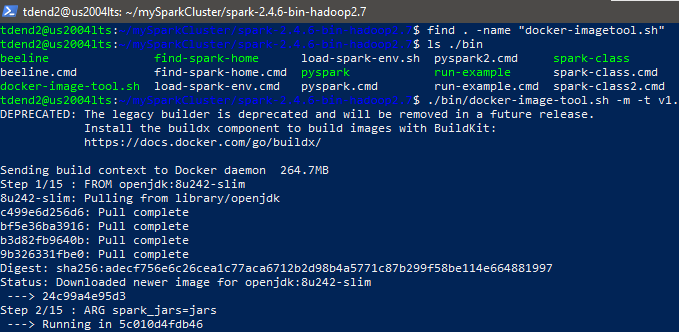
*WORKDIR /opt/spark/work-dir*

*ENTRYPOINT [ "/opt/entrypoint.sh" ]*

*1.3.3 Creating a Spark Docker image Kubernetes requires users to supply images that can be deployed into containers within pods. The images are built to be run in a container runtime environment that Kubernetes supports. Spark (starting with version 2.3) ships with a Dockerfile that can be used for this purpose, or customized to match an individual application’s needs. It can be found in the kubernetes/dockerfiles/ directory. Spark also ships with a bin/docker-image-tool.sh script that can be used to build and publish the Docker images to use with the Kubernetes backend.*

*To build a spark-docker image, run the command below. It takes time.*

*./bin/docker-image-tool.sh -m -t v1.0.uis build*



Last part of execution of above:



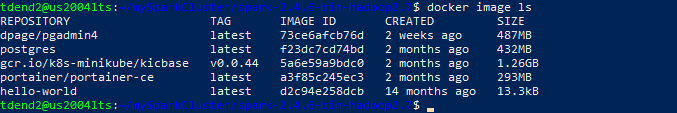
Note: The -m parameter here indicates a minikube build. Using minikube when building images

will do so directly into minikube's Docker daemon. There is no need to push the images into

minikube in that case; they'll be automatically available when running applications inside the

minikube cluster. Reference:<https://github.com/apache/spark/blob/master/bin/docker-image-tool.sh>

Checked the created images using ‘docker image ls’ command:



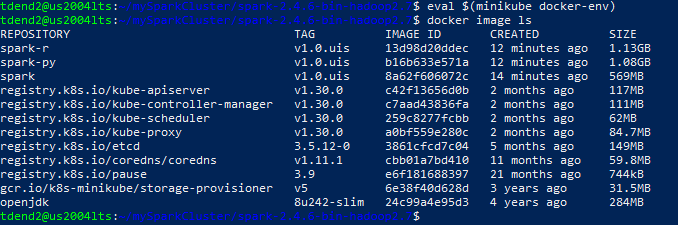
Note2: Did not find openjdk:8u242-slim in the output in addition to three different spark images:

• For Scala: spark:v1.0.uis

• For Python: spark-py:v1.0.uis

• For R: spark-r:v1.0.uis

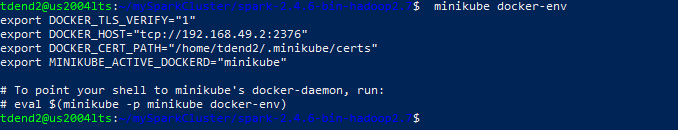
So we need to connect to minikube’s docker daemon to find your spark images.



NOTE: You need to rerun the *$ eval $(minikube docker-env )* command if you log off and on.

Now, we see three new images you created: spark-r, spark-py, and spark.

Later, when you no longer wish to use the Minikube host, you can undo this change by running. The command ‘minikube docker-env’ gives some environment variables (export). Then, adding those variables in shell environment. [https://www.tutorialspoint.com/unix\_commands/export.htm, https://www.tutorialspoint.com/unix\_commands/eval.htm, https://stackoverflow.com/questions/52310599/what-does-minikube-docker-env-mean](https://www.tutorialspoint.com/unix_commands/export.htm)



c. Section 1.4 Running Spark app (analytical queries) using spark-submit

There are two ways of submitting jobs: client or cluster mode.

There is a subtle difference between the two. If we use client mode, we can tell the driver to run on dedicated infrastructure (separate from executors), whereas if we choose cluster mode, both drivers and executors run in the same cluster. We can use Spark configurations as well as Kubernetes-specific options within your command. Reference, <https://aws.amazon.com/blogs/containers/optimizing-spark-performanceon-kubernetes/>

In this exercise, we used cluster mode.

• Cluster Mode: <https://spark.apache.org/docs/2.4.6/running-on-kubernetes.html#clustermode>

• Client Mode: <https://spark.apache.org/docs/2.4.6/running-on-kubernetes.html#clientmode>

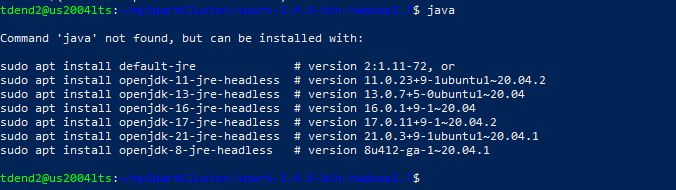
Reference:

[https://www.lightbend.com/blog/how-to-manage-monitor-spark-onkubernetes-introduction-spark-submit-kubernetesoperator#:~:text=How%20Does%20Spark%2DSubmit%20Work,pods%20in%20your%20Kuber netes%20cluster](https://www.lightbend.com/blog/how-to-manage-monitor-spark-onkubernetes-introduction-spark-submit-kubernetesoperator#:~:text=How%20Does%20Spark%2DSubmit%20Work,pods%20in%20your%20Kuber).

The spark-submit script in Spark’s bin directory is used to launch applications on a cluster. It can use all of Spark’s supported cluster managers through a uniform interface, so you don’t have to configure your application, especially for each one. We will run an example Spark app that will calculate Pi. Reference to calculate Pi, <https://www.mathscareers.org.uk/article/calculating-pi/>

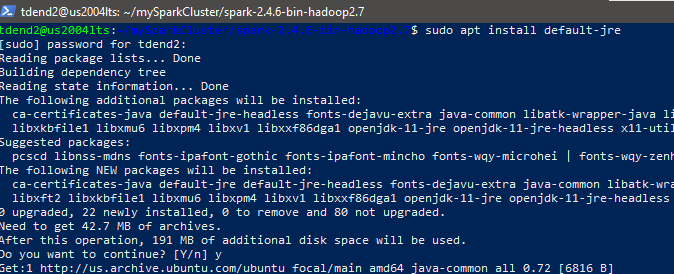
1.4.1 Installing Java JRE and compatibility

To run ‘spark-submit’, we need Java. Due to the Java compatibility issue mentioned earlier, we will use default-jre based on OpenJDK-11 image

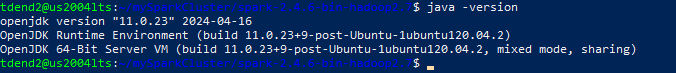


Java not found so installing as shown here:

*sudo apt install default-jre*

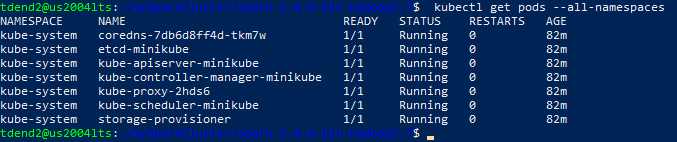
**

The default-jre and openjdk-14 are tested and work well. However, openjdk-8-jre doesn’t work. Checked your Java’s version:



Included Kube-system pods:

*kubectl get pods --all-namespaces*



1.4.2 Role-based access control (RBAC)

Role-based access control (RBAC) is a method of regulating access to a computer or network

resources based on the roles of individual users within your organization.

Reference:

<https://kubernetes.io/docs/reference/access-authn-authz/rbac/>

In Kubernetes clusters with RBAC enabled, users, can configure Kubernetes RBAC roles and

service accounts used by the various Spark on Kubernetes components to access the Kubernetes API server.

Reference: <https://spark.apache.org/docs/2.4.6/running-on-kubernetes.html#rbac>

In this exercise, we created one service account for running the spark job with the

required access as below if we do not have one to use. Allowed service account default:spark

access namespace default



kubectl create clusterrolebinding spark-role --clusterrole=edit --serviceaccount=default:spark --namespace=default

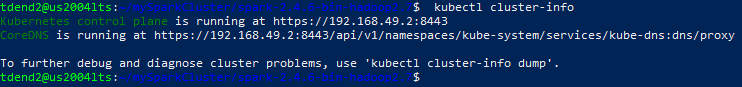


We need to add a parameter in the spark-submit script( refer 3.4.4 Submitting Spark job using spark-submit)

1.4.3 Preparing for running a spark job

First, checked Kubernetes master’s IP address in Minikube: e.g., 172.17.0.3

*kubectl cluster-info*



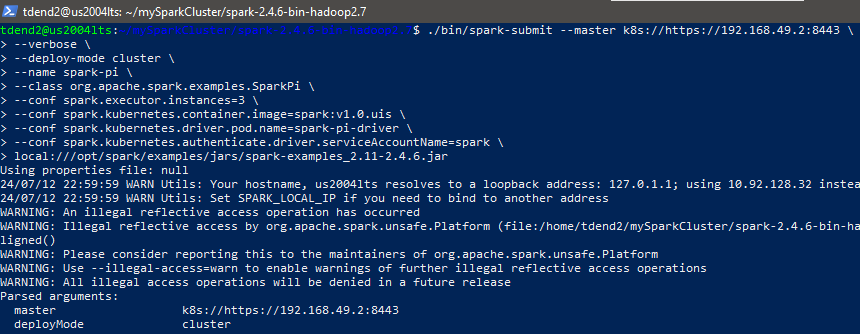
Or we may use a command ‘minikube ip’

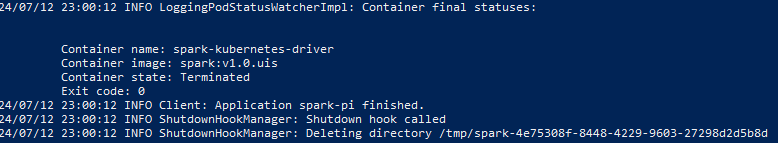


1.4.4 Submitting Spark job using spark-submit

./bin/spark-submit --master k8s://https://192.168.49.2:8443 \

--verbose \ --deploy-mode cluster \ --name spark-pi \ --class org.apache.spark.examples.SparkPi \ --conf spark.executor.instances=3 \ --conf spark.kubernetes.container.image=spark:v1.0.uis \ --conf spark.kubernetes.driver.pod.name=spark-pi-driver \ --conf spark.kubernetes.authenticate.driver.serviceAccountName=spark \ local:///opt/spark/examples/jars/spark-examples\_2.11-2.4.6.jar





We will use cluster mode (deploy-mode cluster) and name it spark-pi. We will use SparkPi class to calculate Pi with 3 instances/executors/nodes. We will use the spark image we created before, spark:v1.0.uis, and named the driver pod as spark-pi-driver. We need to give serviceAccountName=spark for RBAC, see Section 3.4.2. The spark-exemples jar will be used. The location of the jar file is for Spark Docker image/container/Pod we created, not for downloaded Apache Spark in Section 3.3.1.

Everything went fine, we are able to check logs and see the result as below:



Then we can cat or grep a log from that pod to see the results.



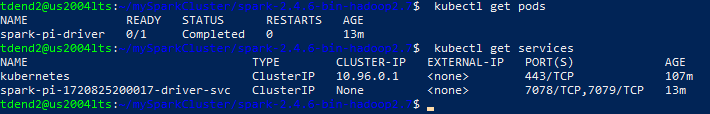
No errors hence result obtained.

If any error occurs we can n debug it using kubectl logs.

*kubectl logs spark-pi-driver*

In the log if we see any error, we need to check RBAC.

If we delete the pod, related services will also be deleted



**• Run the minikube Dashboard again.**

**a. explain information of the below:**

**i. Cluster > Nodes > minikube**

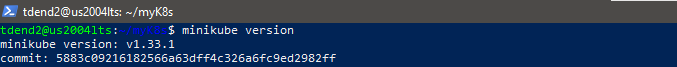
**ii. Workloads > Pods > spark-pi-driver**

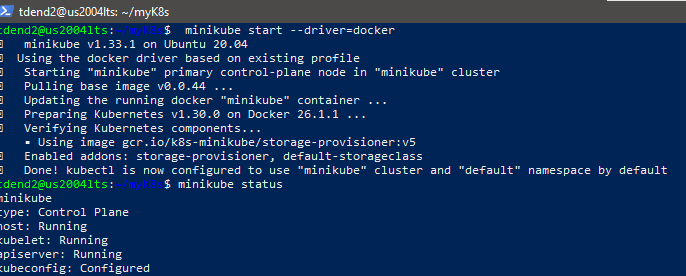
**iii. Service > Services > spark-pi-xxxxx-driver-svc**

Steps:

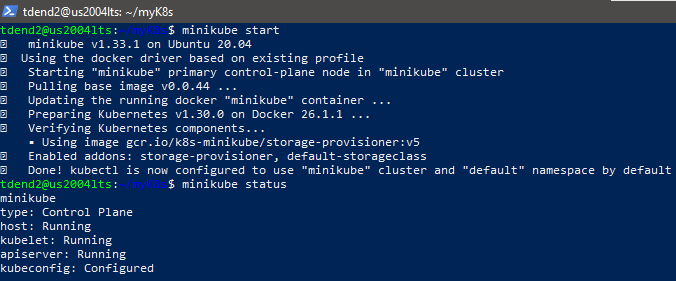
1)Created and navigated to myK8s directory

2) Checked the minikube version, started driver named docker and checked the status of minikube

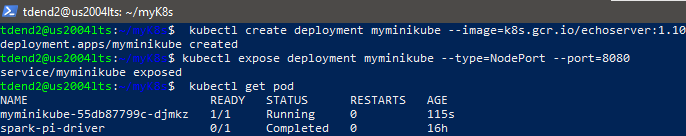




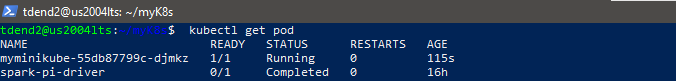
3)Started minikube and checked the status of cluster



4) Created and exposed deployment on port 8080 so that it can be accessed.The option --type=NodePort specifies the type of the Service.



5)Verified if the pod is up and running



6) Got the URL of the exposed Service to view the Service details:[*http://192.168.49.2:30727*](http://192.168.49.2:30727) *and port forwarded as follows:*

**

*SSH port forwarding option: -L*

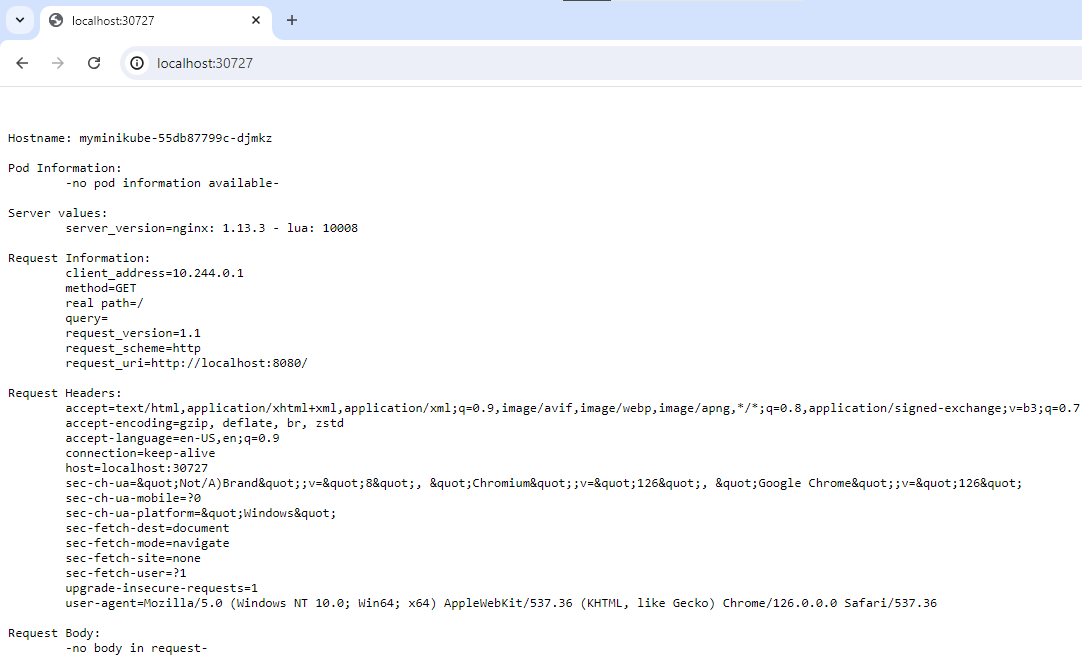
*• Your laptop’s local port: 30727*

*• Your cluster’s IP and its port inside your host/docker VM: 192.168.49.2:30727*

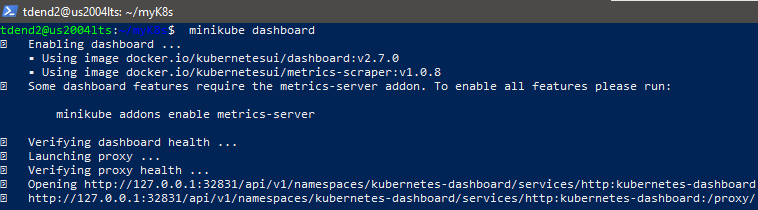
*• Your host/docker VM’s login and IP: tdend2@10.92.128.32*

*ssh -L 30727:192.168.49.2:30727 tdend2@10.92.128.32*

Accessed the webpage in your browser with localhost and the port number.

**

7)Now we are ready to use the ***minikube dashboard*** to get an overview of applications running on your cluster, as well as for creating or modifying individual Kubernetes resources (such as Deployments, Jobs, DaemonSets, etc.) and soto access it port forwarded it to access on browser of my laptop instead of browser of virtual machine as follows;



SSH port forwarding option: -L

• Your laptop’s local port: 32831

• Your dashboard’s proxy IP and its port inside your host/docker VM: 127.0.0.1:32831 or

localhost:32831

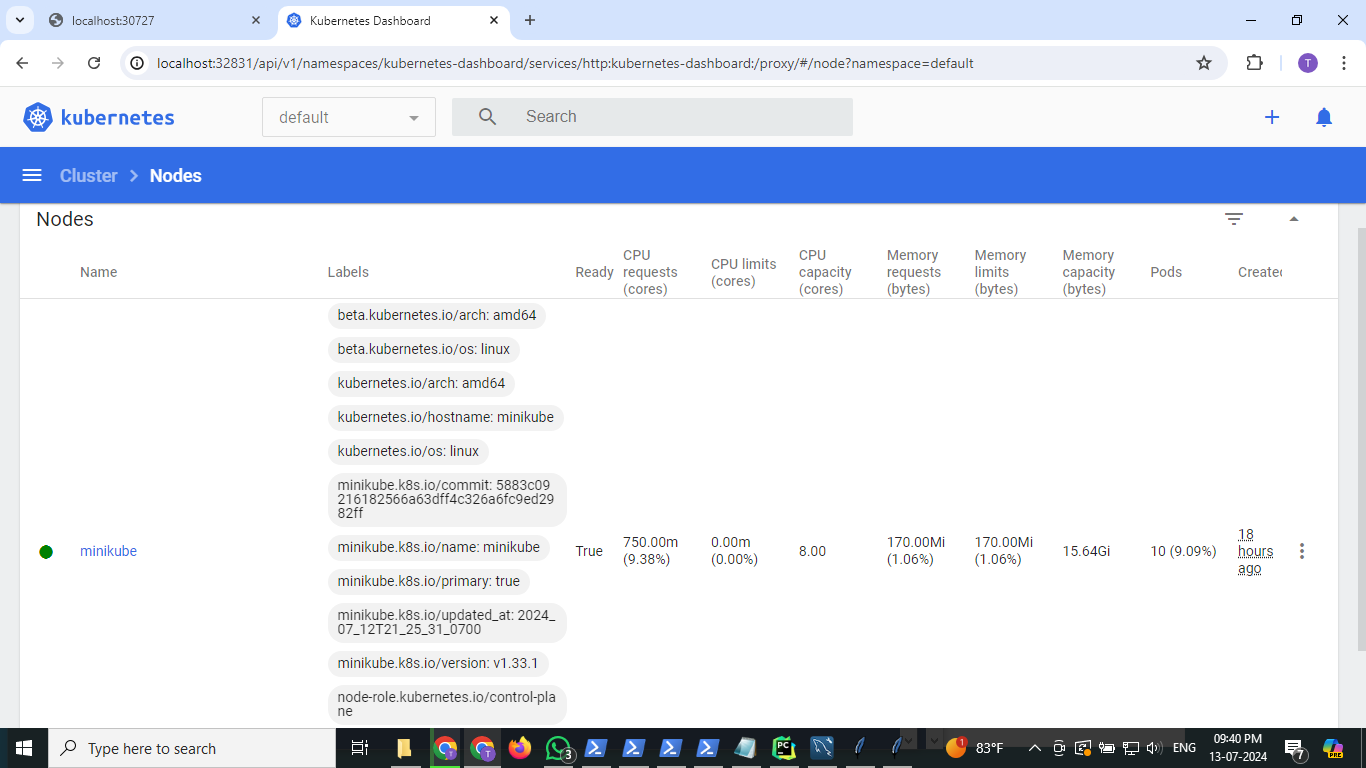
• Your host/docker VM’s login and IP: tdend2@10.92.128.32

ssh -L 32831:localhost:32831 tdend2@10.92.128.32

8) Accessed the dashboard on my browser using my port number 32831 as follows:

<http://localhost:32831/api/v1/namespaces/kubernetes-dashboard/services/http:kubernetes-dashboard:/proxy/#/overview?namespace=default>

**i. Cluster > Nodes > minikube**



We already knew that *Minikube node* is up and running.

**Node Name and Labels**:

* Node Name: minikube
* Labels: Various labels that identify the node properties and roles.
* To inspect node details: *kubectl describe node minikube*

**Node Status**:

* The node is Ready which implies that it can accept workloads.
* From command line utility node status can checked as

*kubectl get nodes*

**CPU Metrics**:kubectl top nodes- *Monitor Resource Usage:*

* CPU Requests: 750.00m (which is 9.38% of the total CPU capacity).
* CPU Limits: 0.00m (0.00%), meaning there are no specific CPU limits set.
* CPU Capacity: 8.00 cores.

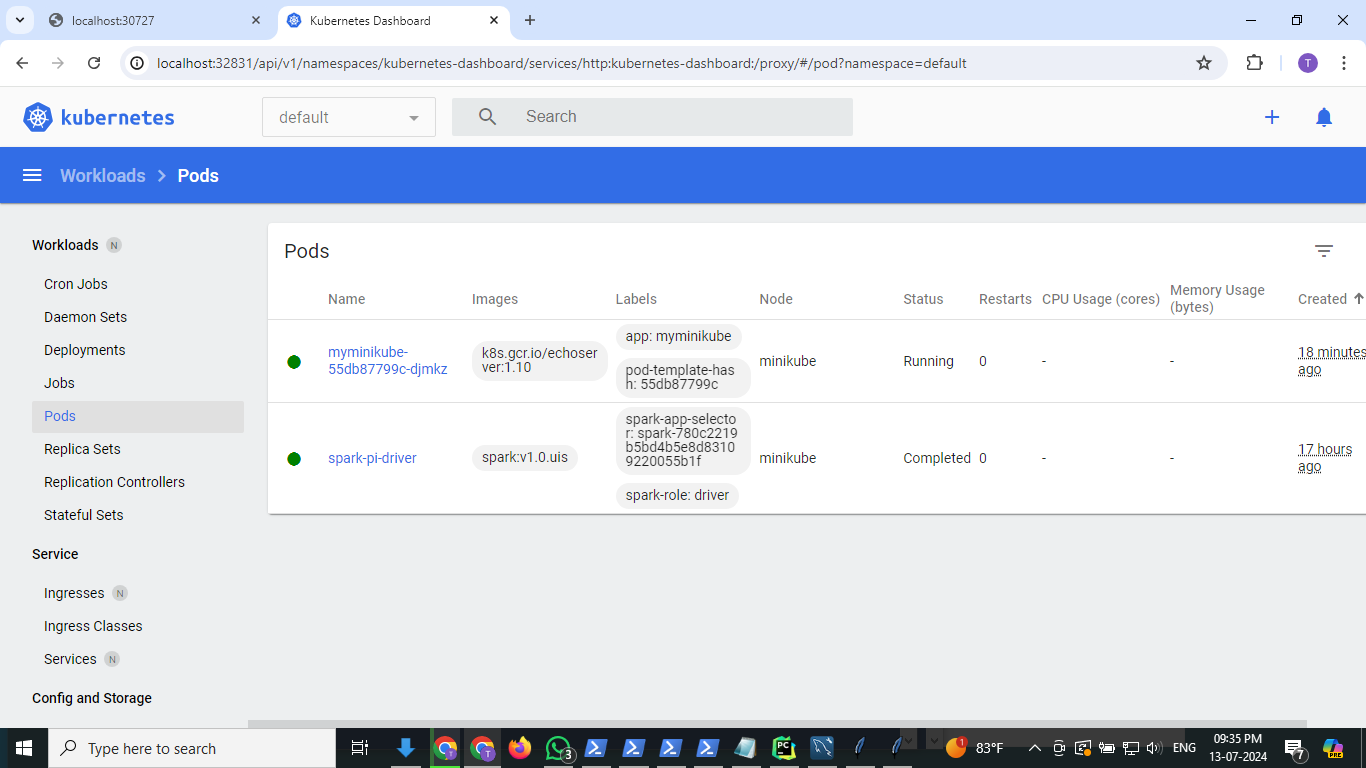
**Memory Metrics**:*kubectl top nodes (Monitor Resource Usage*)

* Memory Requests: 170.00Mi (1.06% of the total memory capacity).
* Memory Limits: 170.00Mi (1.06%), meaning the set limits are equal to the requests.
* Memory Capacity: 15.64Gi.

**Pod Information**: kubectl get pods -A (*List Running Pods)*

* Pods: 10 (9.09% of the total possible pods).

**ii. Workloads > Pods > spark-pi-driver**



Pod is an atomic unit of scheduling and a shared execution environment for one or more containers act as a vehicle for executing an application.

It has a set of resources that are shared by every container that is part of the Pod. These resources - IP addresses, ports, hostname, sockets, memory, volumes, and more…

Each Pod creates its own network namespace

• E.g., Single IP address, a single range of TCP and UDP ports, and a single routing table. Pods get scheduled on nodes

Two pods running in your Minikube cluster.The details are as follows:

### **1. myminikube-55db87799c-djmkz**

* **Image**: k8s.gcr.io/echoserver:1.10
* **Labels**:
  + app: myminikube
  + pod-template-hash: 55db87799c
* **Node**: minikube
* **Status**: Running
* **Restarts**: 0

This pod is running an echo server. It has been up and running for 56 minutes without any restarts, indicating that it is stable.

### **2. spark-pi-driver**

* **Image**: spark:v1.0.uis
* **Labels**:
  + spark-app-selector: spark-780c2219b5bd4b5e8d83109220055b1f
  + spark-role: driver
* **Node**: minikube
* **Status**: Completed
* **Restarts**: 0

This pod was running a Spark driver. Its status is Completed, which indicates that the job it was running has finished successfully. It is not restarted as it ran without errors.

From the command line utility the above details can be obtained as follows:

The pods can be gotten using *kubectl get pods*.

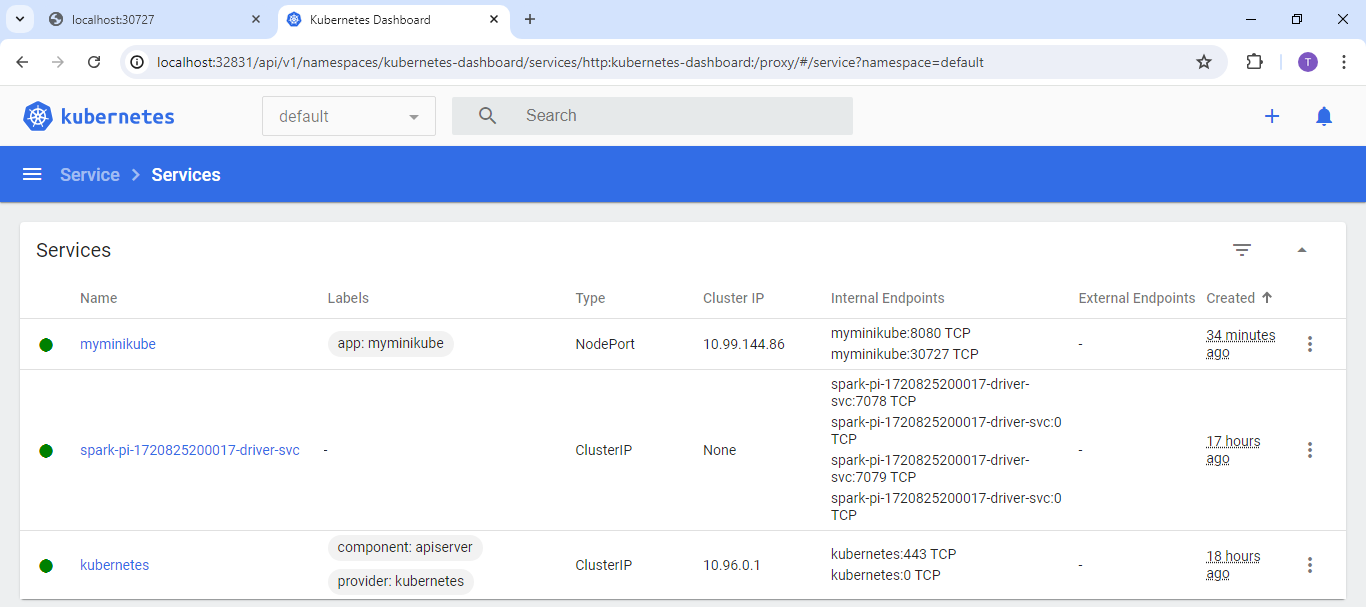
Description of specific pod,for example as *kubectl describe pod spark-pi-driver*

Log of a pod can be obtained as *kubectl logs spark-pi-driver*

Resource usage can be checked if metric server is installed.

For example:*kubectl top pod spark-pi-driver*

**iii. Service > Services > spark-pi-xxxxx-driver-svc**



•Services are all about providing stable networking for Pods. Also provides load-balancing and ways to be accessed from outside of the cluster.

*• Front-end of a Service* provides a stable IP, DNS name and port that is guaranteed not to change for the entire life of the Service

*• Back-end of a Service* uses labels to load-balance traffic across a

potentially dynamic set of application Pods.

Three services running in your Minikube cluster.They are as follows:

### **1. myminikube**

* **Labels**: app: myminikube
* **Type**: NodePort
* **Cluster IP**: 10.99.144.86
* **Internal Endpoints**: myminikube:8080 TCP
* **External Endpoints**: myminikube:30727 TCP

This service is accessible within the cluster at 10.99.144.86:8080 and externally at NodeIP:30727 which means it can be accessed from inside and outside the cluster..

NodePort Service

• Builds on top of ClusterIP and enables access from outside of the cluster

• By adding another port that can be used to reach the Service from outside the cluster and additional port is called the NodePort

*ClusterIP Service (default)*

• Has a stable IP address and port that is only accessible from inside the cluster

• It’s programmed into the network fabric and guaranteed to be stable for the

life of the Service

• Network just knows about it and you don’t need to bother with the details

• ClusterIP gets registered against the name of the Service on the

cluster’s internal DNS service

• All Pods in the cluster are pre-programmed to know about the cluster’s DNS

service

• All Pods can resolve Service names.

### **2. spark-pi-1720825200017-driver-svc**

* **Labels**: None specified
* **Type**: ClusterIP
* **Cluster IP**: None
* **Internal Endpoints**:
  + spark-pi-1720825200017-driver-svc:7078 TCP
  + spark-pi-1720825200017-driver-svc:0 TCP
  + spark-pi-1720825200017-driver-svc:7079 TCP
  + spark-pi-1720825200017-driver-svc:0 TCP

This service is intended for internal communication only, and it doesn't have a Cluster IP, which means it may be using a headless service. It exposes several ports internally within the cluster.

A Kubernetes-native application can query the Endpoints API directly bypassing the DNS lookup and use of the Service’s IP.

### **3. kubernetes**

* **Labels**:
  + component: apiserver
  + provider: kubernetes
* **Type**: ClusterIP
* **Cluster IP**: 10.96.0.1
* **Internal Endpoints**:
  + kubernetes:443 TCP
  + kubernetes:0 TCP

This is the default Kubernetes API server service, accessible internally at 10.96.0.1:443.

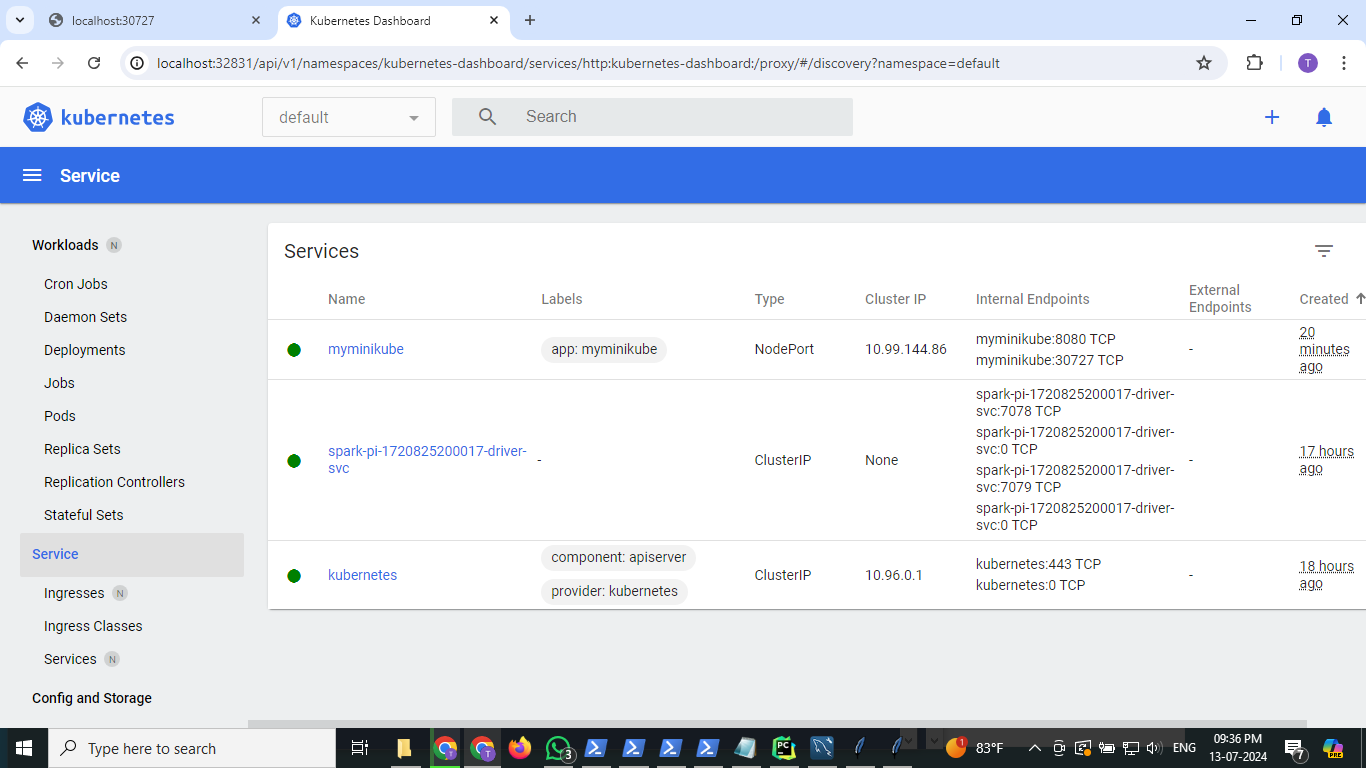
The following commands can be used for the same above information about the configuration and status of each service, including endpoints and any potential issues from the command line utility apart from the UI interface of dashboard to get a detailed overview:

To get the services: *kubectl get services*

For the description of a specific service:

*kubectl describe service spark-pi-1720825200017-driver-svc*

To check endpoints:*kubectl get endpoints kubernetes*



==============================THE END===================================