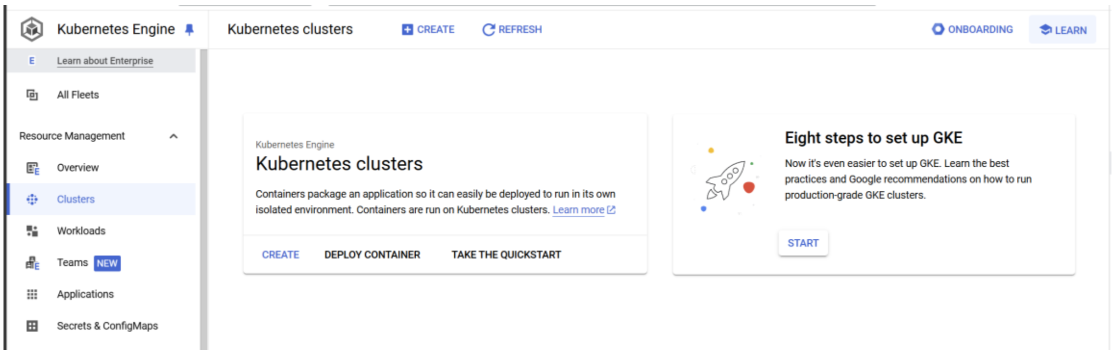
Work Count

Using PySpark to implement Word Count on Apache Spark running on Kubernetes.

Enable Required APIs:

- Google Kubernetes Engine API
- Google Cloud Storage API



Kubernetes Engine

Kubernetes clusters

Containers package an application so it can easily be deployed to run in its own isolated environment. Containers are run on Kubernetes clusters. Learn more

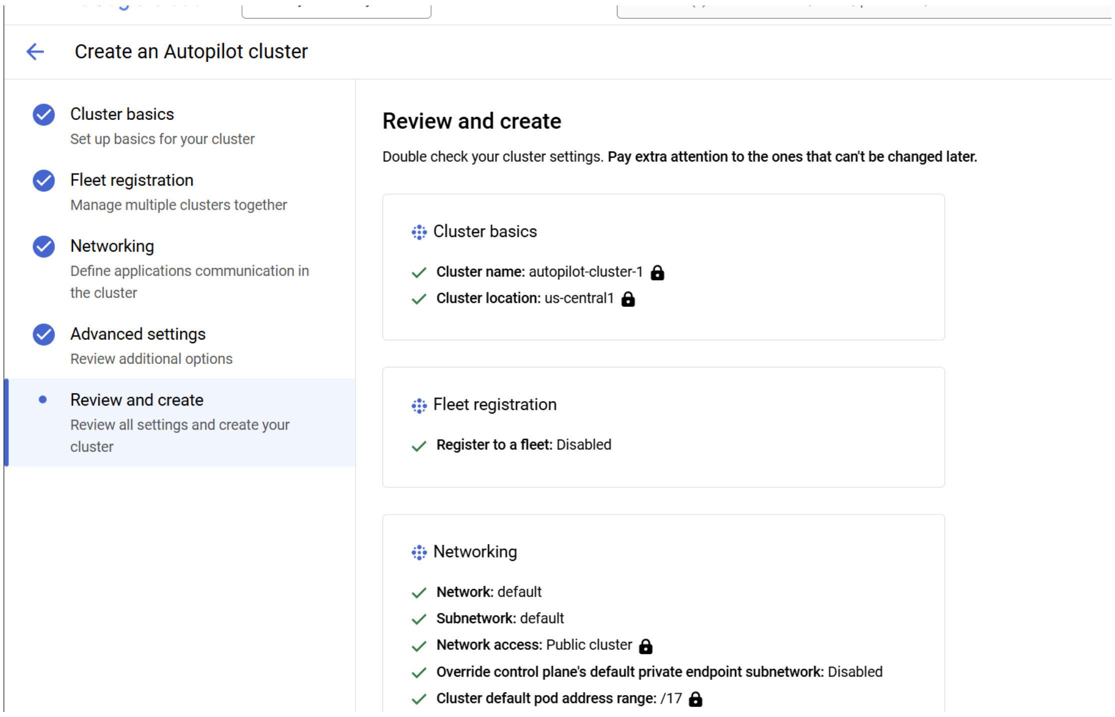
CREATE DEPLOY CONTAINER TAKE THE QUICKSTART

Eight steps to set up GKE

Now it's even easier to set up GKE. Learn the best practices and Google recommendations on how to run production-grade GKE clusters.

START

Creating a Kubernetes Cluster in GKE



Create an Autopilot cluster

- Cluster basics — Set up basics for your cluster
- Fleet registration — Manage multiple clusters together
- Networking — Define applications communication in the cluster
- Advanced settings — Review additional options
- Review and create — Review all settings and create your cluster

Review and create

Double check your cluster settings. **Pay extra attention to the ones that can't be changed later.**

Cluster basics

- **Cluster name:** autopilot-cluster-1
- **Cluster location:** us-central1

Fleet registration

- **Register to a fleet:** Disabled

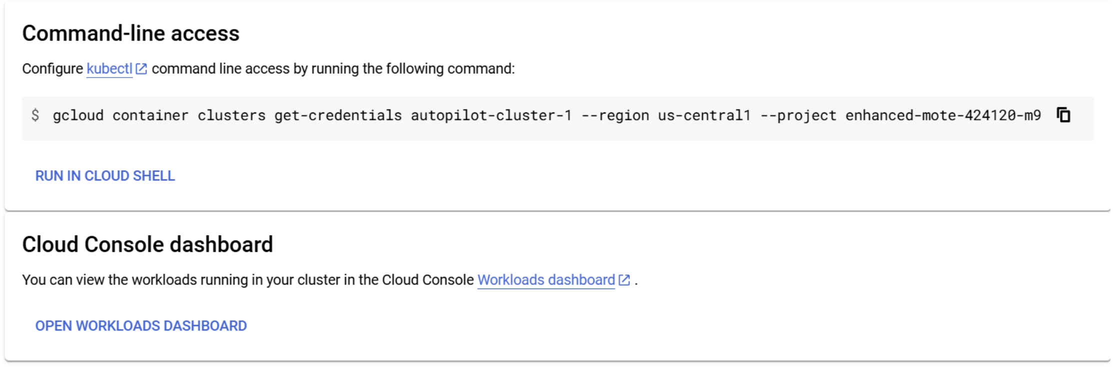
Networking

- **Network:** default
- **Subnetwork:** default
- **Network access:** Public cluster
- **Override control plane's default private endpoint subnetwork:** Disabled
- **Cluster default pod address range:** /17

Get Cluster Credentials:

- Run the following command to get credentials for your cluster

gcloud container clusters get-credentials wordcount-cluster --zone=us-central1-a



Command-line access

Configure kubectl command line access by running the following command:

```
$ gcloud container clusters get-credentials autopilot-cluster-1 --region us-central1 --project enhanced-mote-424120-m9
```

RUN IN CLOUD SHELL

Cloud Console dashboard

You can view the workloads running in your cluster in the Cloud Console Workloads dashboard .

OPEN WORKLOADS DASHBOARD

Need to download pyspark first
Download Spark:

wget https://downloads.apache.org/spark/spark-3.1.3/spark-3.1.3-bin-hadoop2.7.tgz

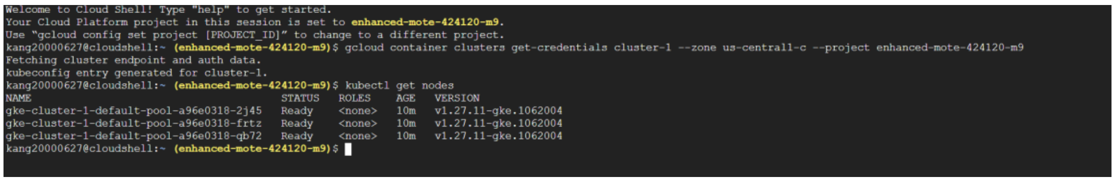
Extract the Spark Archive
tar xvf spark-3.1.3-bin-hadoop2.7.tgz

Set Up Spark Environment
Set the PATH to include Spark binaries:
export PATH=$PATH:~/spark-3.1.3-bin-hadoop2.7/bin

Verify Your Cluster:

- Run the following command to ensure your cluster is running:

kubectl get nodes



```
Welcome to Cloud Shell! Type "help" to get started.
Your Cloud Platform project in this session is set to enhanced-mote-424120-m9.
Use "gcloud config set project [PROJECT_ID]" to change to a different project.
kang20000627@cloudshell:~ (enhanced-mote-424120-m9)$ gcloud container clusters get-credentials cluster-1 --zone us-central1-c --project enhanced-mote-424120-m9
Fetching cluster endpoint and auth data.
kubeconfig entry generated for cluster-1.
kang20000627@cloudshell:~ (enhanced-mote-424120-m9)$ kubectl get nodes
NAME                                       STATUS   ROLES    AGE   VERSION
gke-cluster-1-default-pool-a96e0318-2j45   Ready    <none>   10m   v1.27.11-gke.1062004
gke-cluster-1-default-pool-a96e0318-frtz   Ready    <none>   10m   v1.27.11-gke.1062004
gke-cluster-1-default-pool-a96e0318-qb72   Ready    <none>   10m   v1.27.11-gke.1062004
kang20000627@cloudshell:~ (enhanced-mote-424120-m9)$
```

Create the input.txt File

1. **Create the text file**:

echo -e "Hello world\nHello Kubernetes\nHello PySpark" > input.txt

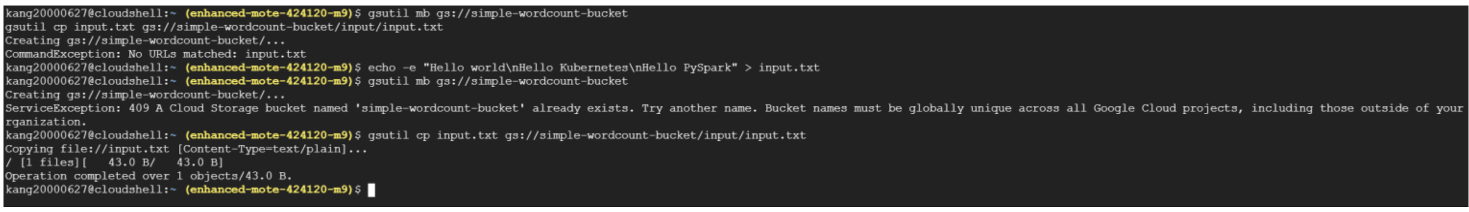
Step 2: Upload the input.txt File to Google Cloud Storage

1. **Create the bucket (if not already created)**:

gsutil mb gs://simple-wordcount-bucket

2. **Upload the file to the bucket**:

gsutil cp input.txt gs://simple-wordcount-bucket/input/input.txt



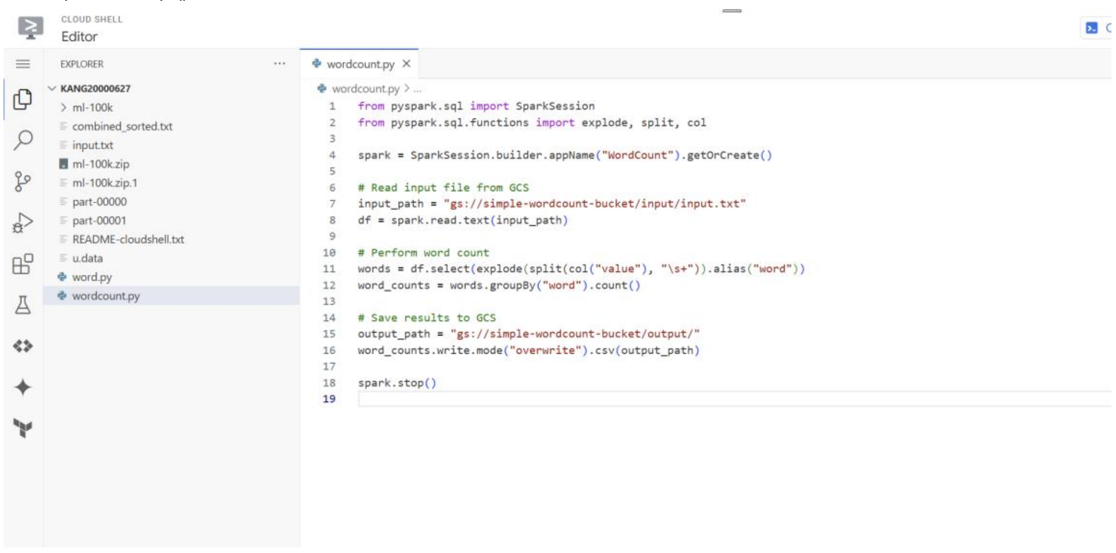
```
kang20000627@cloudshell:~ (enhanced-mote-424120-m9)$ gsutil mb gs://simple-wordcount-bucket
gsutil cp input.txt gs://simple-wordcount-bucket/input/input.txt
Creating gs://simple-wordcount-bucket/...
CommandException: No URLs matched: input.txt
kang20000627@cloudshell:~ (enhanced-mote-424120-m9)$ echo -e "Hello world\nHello Kubernetes\nHello PySpark" > input.txt
kang20000627@cloudshell:~ (enhanced-mote-424120-m9)$ gsutil mb gs://simple-wordcount-bucket
Creating gs://simple-wordcount-bucket/...
ServiceException: 409 A Cloud Storage bucket named 'simple-wordcount-bucket' already exists. Try another name. Bucket names must be globally unique across all Google Cloud projects, including those outside of your
rganization.
kang20000627@cloudshell:~ (enhanced-mote-424120-m9)$ gsutil cp input.txt gs://simple-wordcount-bucket/input/input.txt
Copying file://input.txt [Content-Type=text/plain]...
/ [1 files][   43.0 B/   43.0 B]
Operation completed over 1 objects/43.0 B.
kang20000627@cloudshell:~ (enhanced-mote-424120-m9)$
```

Create a New File:

- In the Cloud Shell Editor, create a new file named wordcount.py

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import explode, split, col

spark = SparkSession.builder.appName("WordCount").getOrCreate()

# Read input file from GCS
input_path = "gs://simple-wordcount-bucket/input/input.txt"
```

```
df = spark.read.text(input_path)

# Perform word count
words = df.select(explode(split(col("value"), "\s+")).alias("word"))
word_counts = words.groupBy("word").count()

# Save results to GCS
output_path = "gs://simple-wordcount-bucket/output/"
word_counts.write.mode("overwrite").csv(output_path)

spark.stop()
```



```
from pyspark.sql import SparkSession
from pyspark.sql.functions import explode, split, col

spark = SparkSession.builder.appName("WordCount").getOrCreate()

# Read input file from GCS
input_path = "gs://simple-wordcount-bucket/input/input.txt"
df = spark.read.text(input_path)

# Perform word count
words = df.select(explode(split(col("value"), "\s+")).alias("word"))
word_counts = words.groupBy("word").count()

# Save results to GCS
output_path = "gs://simple-wordcount-bucket/output/"
word_counts.write.mode("overwrite").csv(output_path)

spark.stop()
```

Upload the Script to Google Cloud Storage:

```
gsutil cp wordcount.py gs://simple-wordcount-bucket/wordcount.py
```

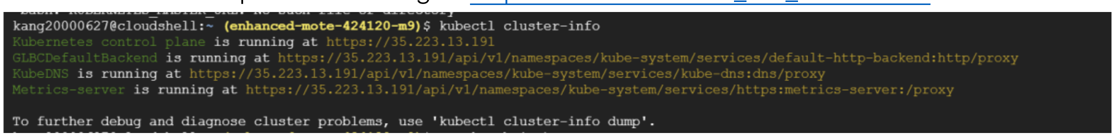
Get the Kubernetes API Endpoint:

- Run the following command to get the Kubernetes API server endpoint:

```
kubectl cluster-info
```

- You should see an output like this:

Kubernetes control plane is running at https://<KUBERNETES_API_ENDPOINT>

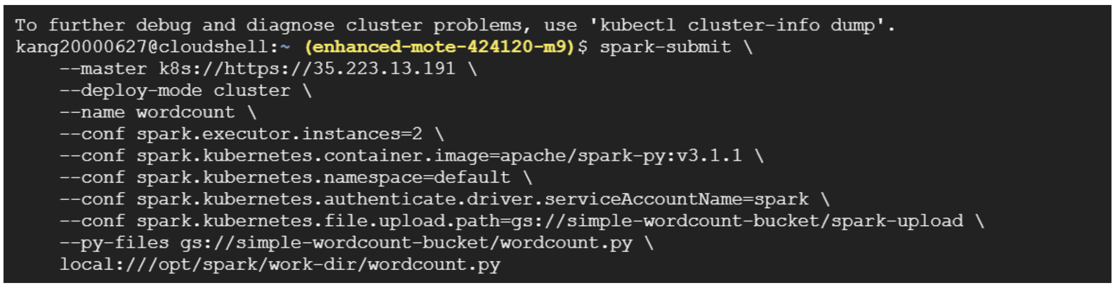


```
kang20000627@cloudshell:~ (enhanced-mote-424120-m9)$ kubectl cluster-info
Kubernetes control plane is running at https://35.223.13.191
GLBCDefaultBackend is running at https://35.223.13.191/api/v1/namespaces/kube-system/services/default-http-backend:http/proxy
KubeDNS is running at https://35.223.13.191/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy
Metrics-server is running at https://35.223.13.191/api/v1/namespaces/kube-system/services/https:metrics-server:/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
```

Submit the Spark Job:

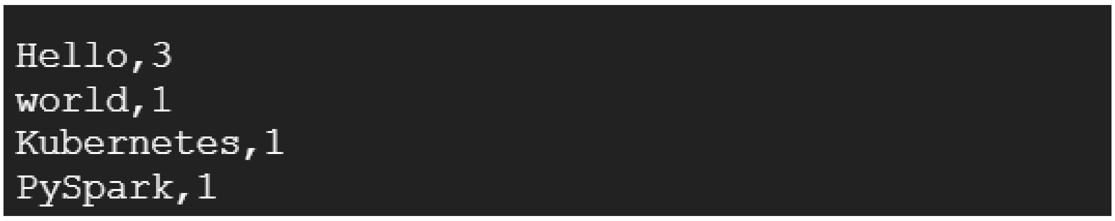
```
spark-submit \
   --master k8s://https://34.68.12.34 \
   --deploy-mode cluster \
   --name wordcount \
   --conf spark.executor.instances=2 \
   --conf spark.kubernetes.container.image=apache/spark-py:v3.1.1 \
   --conf spark.kubernetes.namespace=default \
```

--conf spark.kubernetes.authenticate.driver.serviceAccountName=spark \
--conf spark.kubernetes.file.upload.path=gs://simple-wordcount-bucket/spark-upload \
--py-files gs://simple-wordcount-bucket/wordcount.py \
local:///opt/spark/work-dir/wordcount.py



```
To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
kang20000627@cloudshell:~ (enhanced-mote-424120-m9)$ spark-submit \
    --master k8s://https://35.223.13.191 \
    --deploy-mode cluster \
    --name wordcount \
    --conf spark.executor.instances=2 \
    --conf spark.kubernetes.container.image=apache/spark-py:v3.1.1 \
    --conf spark.kubernetes.namespace=default \
    --conf spark.kubernetes.authenticate.driver.serviceAccountName=spark \
    --conf spark.kubernetes.file.upload.path=gs://simple-wordcount-bucket/spark-upload \
    --py-files gs://simple-wordcount-bucket/wordcount.py \
    local:///opt/spark/work-dir/wordcount.py
```

Then you can see the result



```
Hello,3
world,1
Kubernetes,1
PySpark,1
```

PageRank

The implementation logic is the same as above so I will abbreviate it

Input File for PageRank

1. **Create a File named pagerank_input.txt**:

A B
A C
B C
C A
D C
E F
F C

```
A B
A C
B C
C A
D C
E F
F C
```

This file represents a directed graph where each line shows an edge from one node to another.

Step-by-Step Guide to Implement PageRank with PySpark on Kubernetes

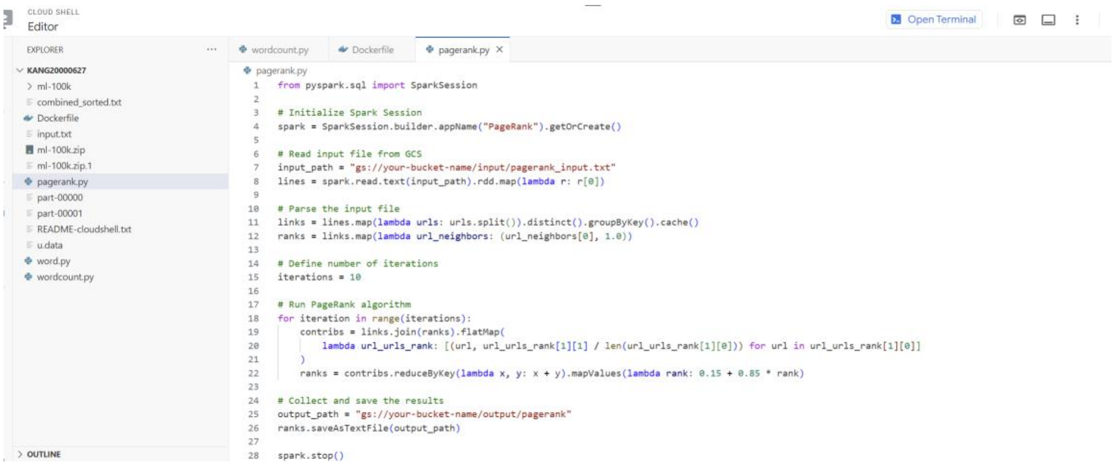
Step 1: Create the pagerank.py Script

1. **Create the PySpark Script**:

In your Cloud Shell, create a file named pagerank.py with the following content:

```
from pyspark.sql import SparkSession

# Initialize Spark Session
spark = SparkSession.builder.appName("PageRank").getOrCreate()

# Read input file from GCS
input_path = "gs://your-bucket-name/input/pagerank_input.txt"
lines = spark.read.text(input_path).rdd.map(lambda r: r[0])

# Parse the input file
links = lines.map(lambda urls: urls.split()).distinct().groupByKey().cache()
ranks = links.map(lambda url_neighbors: (url_neighbors[0], 1.0))

# Define number of iterations
iterations = 10
# Run PageRank algorithm
for iteration in range(iterations):
    contribs = links.join(ranks).flatMap(
        lambda url_urls_rank: [(url, url_urls_rank[1][1] / len(url_urls_rank[1][0])) for url in
url_urls_rank[1][0]]
    )
    ranks = contribs.reduceByKey(lambda x, y: x + y).mapValues(lambda rank: 0.15 + 0.85 *
rank)
# Collect and save the results
output_path = "gs://your-bucket-name/output/pagerank"
ranks.saveAsTextFile(output_path)
```

spark.stop()



```python
from pyspark.sql import SparkSession

# Initialize Spark Session
spark = SparkSession.builder.appName("PageRank").getOrCreate()

# Read input file from GCS
input_path = "gs://your-bucket-name/input/pagerank_input.txt"
lines = spark.read.text(input_path).rdd.map(lambda r: r[0])

# Parse the input file
links = lines.map(lambda urls: urls.split()).distinct().groupByKey().cache()
ranks = links.map(lambda url_neighbors: (url_neighbors[0], 1.0))

# Define number of iterations
iterations = 10

# Run PageRank algorithm
for iteration in range(iterations):
    contribs = links.join(ranks).flatMap(
        lambda url_urls_rank: [(url, url_urls_rank[1][1] / len(url_urls_rank[1][0])) for url in url_urls_rank[1][0]]
    )
    ranks = contribs.reduceByKey(lambda x, y: x + y).mapValues(lambda rank: 0.15 + 0.85 * rank)

# Collect and save the results
output_path = "gs://your-bucket-name/output/pagerank"
ranks.saveAsTextFile(output_path)

spark.stop()
```

2. Replace your-bucket-name with your actual Google Cloud Storage bucket name.

Step 2: Upload the Files to Google Cloud Storage

1. **Upload the pagerank_input.txt file**:

gsutil cp pagerank_input.txt gs://your-bucket-name/input/pagerank_input.txt

2. **Upload the pagerank.py script**:

gsutil cp pagerank.py gs://your-bucket-name/pagerank.py

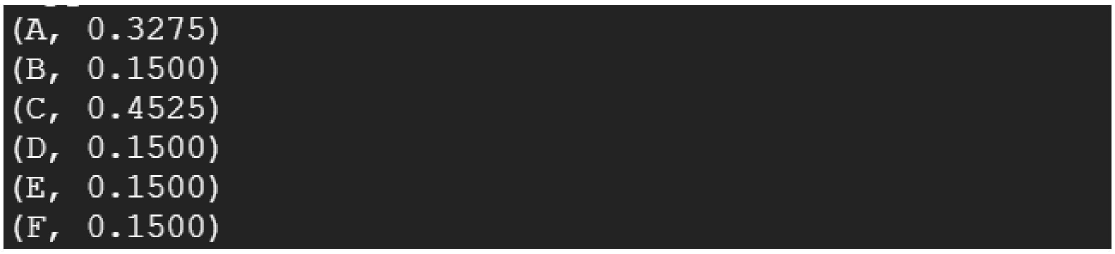
Step 3: Submit the PySpark Job on Kubernetes

1. **Ensure Spark is Installed and Configured**:

Follow the previous steps to download and configure Spark if not already done.

2. **Submit the Spark Job**:

```
spark-submit \
  --master k8s://https://35.223.13.191 \
  --deploy-mode cluster \
  --name pagerank \
  --conf spark.executor.instances=2 \
  --conf spark.kubernetes.container.image=apache/spark-py:v3.3.2 \
  --conf spark.kubernetes.namespace=default \
  --conf spark.kubernetes.authenticate.driver.serviceAccountName=spark \
  --conf spark.kubernetes.file.upload.path=gs://your-bucket-name/spark-upload \
  --py-files gs://your-bucket-name/pagerank.py \
  gs://your-bucket-name/pagerank.py
```



```
(A, 0.3275)
(B, 0.1500)
(C, 0.4525)
(D, 0.1500)
(E, 0.1500)
(F, 0.1500)
```

Summary

Step-by-Step Process to Run Word Count and PageRank with PySpark on Kubernetes

Step 1: Prepare the Input Data

- **Create an input file** that represents the graph structure. Each line in the file represents a directed edge between two nodes.
- **Upload the input file** to a Google Cloud Storage (GCS) bucket.

Step 2: Create the PySpark Script

- **Write a PySpark script** to implement the PageRank algorithm. The script reads the input data, computes PageRank values, and writes the output to GCS.
- **Upload the PySpark script** to the GCS bucket.

Step 3: Set Up Kubernetes and Spark

- **Set up a Kubernetes cluster** if you don't have one already.
- **Install Spark** on your local machine or Cloud Shell environment to use spark-submit.

Step 4: Submit the PySpark Job to Kubernetes

- **Use the spark-submit command** to submit the PySpark job to the Kubernetes cluster. This command specifies the master URL, deployment mode, Docker image, and paths to the PySpark script and input data on GCS.

Step 5: Monitor and Verify the Job

- **Monitor the status** of the Spark job using kubectl commands to ensure that the job is running successfully.
- **Check the output files** in the specified GCS bucket to verify the results of the PageRank computation.

Overview of the Procedure

1. **Prepare Input Data**:
   - Create and upload the graph representation file to GCS.
2. **Create PySpark Script**:
   - Write and upload the PySpark script to GCS.
3. **Set Up Environment**:
   - Ensure Kubernetes cluster is set up.
   - Install and configure Spark.
4. **Submit the Job**:
   - Use spark-submit to run the job on Kubernetes.
5. **Verify Output**:
   - Monitor the job status and verify the output in GCS.