Apache Spark on Kubernetes

WordCount+PageRank+GKE

Aron Sbhatu Dagniew

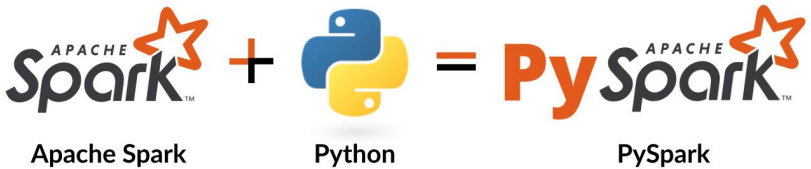
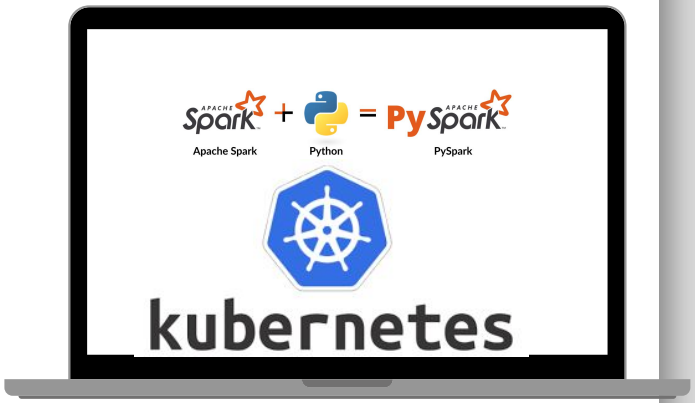
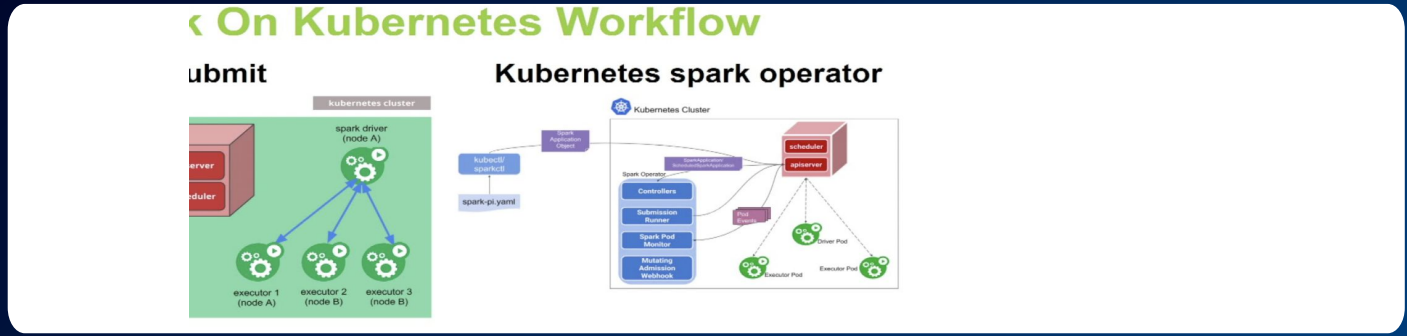


Table of Content:



Introduction

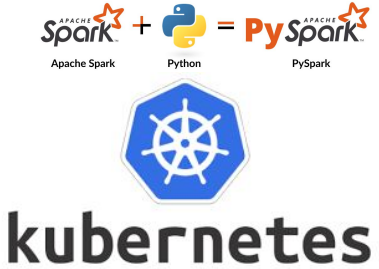


In this project, we aim to utilize PySpark, an open-source cluster-computing framework, to implement Word Count and PageRank on Apache Spark running on Kubernetes. When you submit a Spark application, you interact directly with the Kubernetes API server, which schedules the driver pod. The Spark driver container and the Kubernetes cluster then communicate to request and launch Spark executors, each in its own pod. With dynamic allocation enabled, the number of Spark executors adjusts based on the load; otherwise, it remains static.

Design

Spark Concept

Kubernetes is a rapidly expanding open-source platform designed to automate the deployment, scaling, and management of applications.



Design

Spark Concept

Containers offer:

- Repeatable builds and workflows
- Application portability
- High degree of control over software
- Faster development cycles
- Reduced DevOps workload
- Improved infrastructure utilization

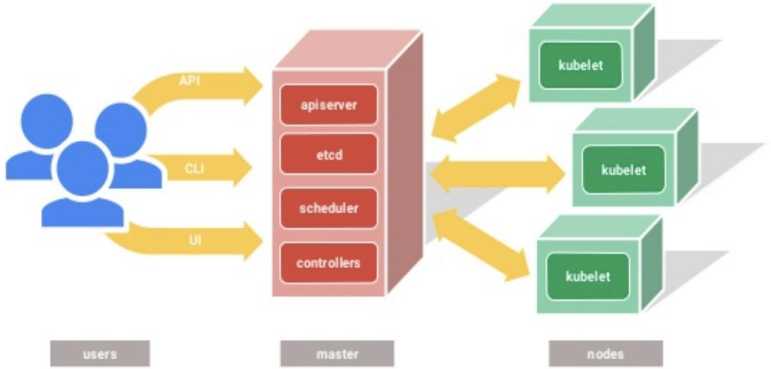
kubernetes

Design

Spark Concept

Nodes and Pods:

- - A pod is a group of co-located containers.
- - Pods are created by a declarative specification provided to the master.
- - Each pod has a unique IP address.
- - Volumes can be either local or network-attached.



Apache Spark + Python = PySpark

kubernetes

Design

Spark Concept

Spark:

Apache Spark is a multi-purpose distributed computing framework known for its in-memory processing capabilities. It provides high-level APIs in Java, Scala, and Python, coupled with an optimized execution engine that can handle various execution graphs. Spark also offers specialized tools such as Spark SQL for processing structured data and MLlib for machine learning tasks.

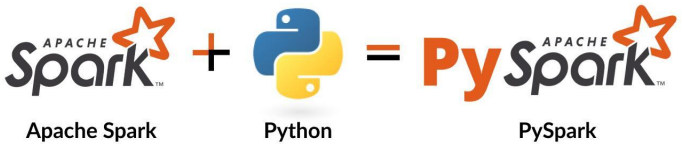
Apache Spark

Design

Spark Concept

PySpark

- PySpark merges the capabilities of Apache Spark with Python, a versatile and widely used high-level programming language. Apache Spark, renowned for its speed, ease of use, and support for streaming analytics, forms the backbone of this collaboration. Python's extensive library ecosystem, including popular tools like numpy, pandas, scikit-learn, seaborn, and matplotlib, enhances its utility for tasks such as machine learning and real-time streaming analytics.

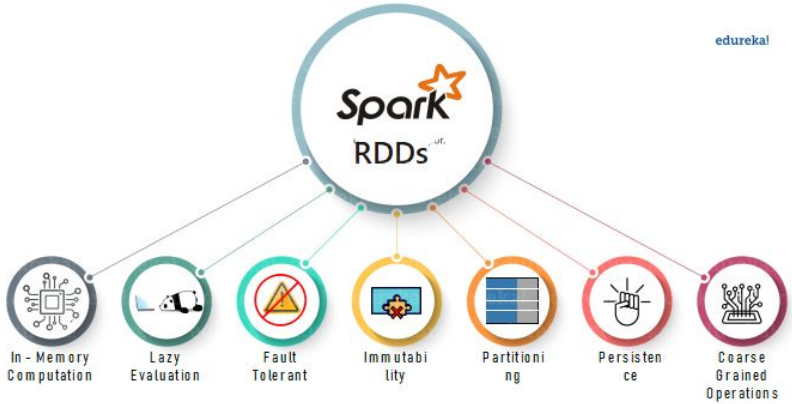


Design

Spark Concept
Fundamental of PySpark

RDDs (Resilient Distributed Datasets):

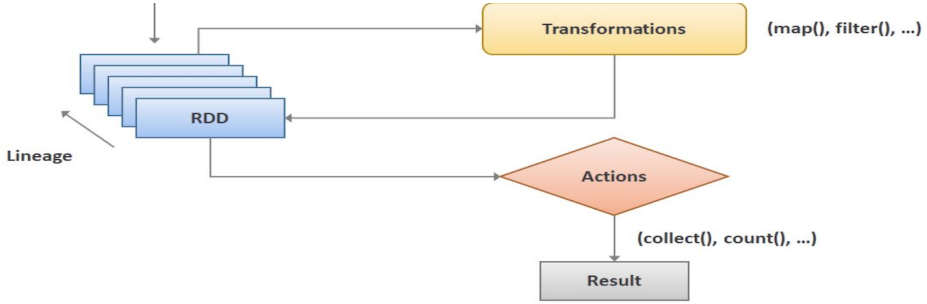
RDDs are distributed collections of objects that can be created from Python, Java, Scala, or user-defined classes. They are typically generated by loading an external dataset or by distributing a collection of objects within the driver program.



Design
Spark concept
Fundamental of PySpark

Two types of operations are supported by Spark RDDs:

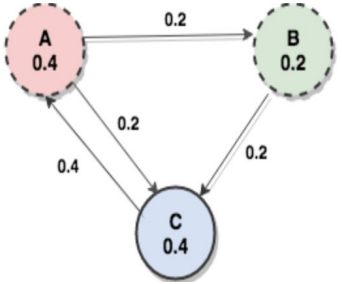
- **Transformations:** These operations create a new RDD by transforming data from an existing RDD.
- **Actions:** These operations compute a result or write a value to the driver program.



Design
Spark concept

kubernetes

1. **Initialization:**
   - Start with each page having an initial PageRank of 1/N, where N is the total number of pages.
2. **Iteration:**
   - Loop until PageRanks converge:
     - Distribute each page's current PageRank evenly to its outgoing links.
     - Accumulate PageRank contributions from pages linking to it.
3. **Formula:**
   - $$PR(u) = \sum_{v \in B_u} \frac{PR(v)}{L(v)},$$
4. **Convergence:**
   - Stop when PageRanks stabilize (minimal change between iterations).
5. **Application:**
   - Used by search engines to rank web pages based on importance.



This algorithm determines the order of search results by analyzing page interconnections and relevance.

Design
Spark concept

kubernetes

Word Count

MapReduce

Job: WordCount						
Map Task				Reduce Task		
MapReduce: map()				MapReduce: reduce()		
Spark: map()				Spark: reduceByKey()		
Input (Given)		Output (Program)		Input (Given)		Output (Program)
Key	Value	Key	Value	Key	Value	
file1	the quick brown fox	the	1	ate	[1]	ate, 1
		quick	1	brown	[1, 1]	brown, 2
		brown	1	cow	[1]	cow, 1
		fox	1	fox	[1, 1]	fox, 2
file1	the fox ate the mouse	the	1	how	[1]	how, 1
		fox	1	mouse	[1]	mouse, 1
		ate	1	now	[1]	now, 1
		the	1	quick	[1]	quick, 1
		mouse	1	the	[1,1,1]	the, 3
file1	how now brown cow	how	1			
		now	1			
		brown	1			
		cow	1			

```
import sys

from pyspark import SparkContext, SparkConf

if __name__ == "__main__":

    # create Spark context with necessary configuration
    sc = SparkContext("local","PySpark Word Count Exmaple")

    ##########################################################
    # read data from text file and split each line in to words
    # - The file file1 contains
    #       the quick brown fox
    #       the fox ate the mouse
    #       how now brown cow
    # - Each line is converted into (word 1)
    #       the
    #       quick
    #       brown
    #       .....
    ##########################################################
    words = sc.textFile("D:/workspace/spark/input.txt").flatMap(lambda line: line.split(" "))

    ##########################################################
    # Count the occurrence of each word
    # Step 1: Each word is converted into (word 1)
    #           (the  1)
    #           (quick 1)
    #           (brown 1)
    #            .....
    # Step 2: reduceByKey
    #           (ate 1)
    #           (brown 2)
    #           (cow 1)
    #            .....
    ##########################################################
    wordCounts = words.map(lambda word: (word, 1)).reduceByKey(lambda a,b: a+b)

    # save the counts to output
    wordCounts.saveAsTextFile("D:/workspace/spark/output/")
```

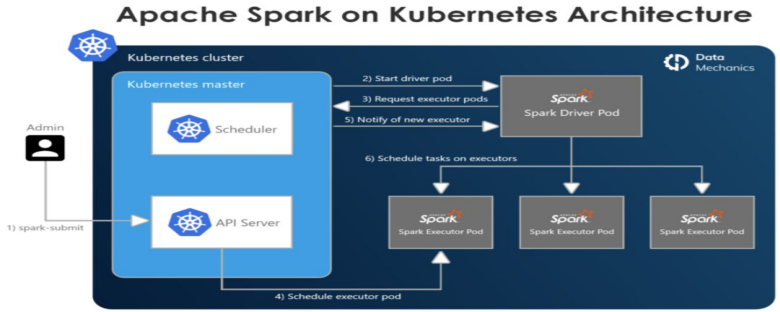
Design
Spark concept

Spark on Kubernetes combines the advantages of Docker containerization and Kubernetes orchestration, offering a streamlined developer experience and enhanced operational efficiency. Kubernetes provides robust resource management, allowing Spark applications to dynamically scale and efficiently share resources with other workloads like batch processing, serving applications, and stateful workloads. This setup leverages Kubernetes' ecosystem of add-on services for logging, monitoring, and security, reducing operational costs and improving infrastructure utilization. Overall, Spark on Kubernetes enables flexible, scalable, and cost-effective deployment of distributed data processing workloads in modern cloud-native environments.

Design
Spark concept
Apache Spark on Kubernetes Architecture

Spark-Submit simplifies Spark application submission to Kubernetes:

- Spark driver runs in a Kubernetes pod.
- Executors are also in Kubernetes pods, connecting to the driver and executing the application code.
- After execution, executor pods terminate and are cleaned up, while the driver pod persists in a "completed" state until garbage collected or manually cleaned up.
- Kubernetes manages pod scheduling, and fabric8 facilitates communication with the Kubernetes API.
- Node selectors allow scheduling of driver and executor pods on specific nodes using configuration properties.



Implementation:

kubernetes

1. Create a cluster on GKE with

gcloud container clusters create spark --num-nodes=1 --machinetype=e2-highmem-2 --region=us-west1

```
adagniew407@cloudshell:~ (cloud-and-bigdata)$ gcloud container clusters create spark --num-nodes=1 --machi
ne-type=e2-highmem-2 --region=us-west1
Default change: VPC-native is the default mode during cluster creation for versions greater than 1.21.0-gk
e.1500. To create advanced routes based clusters, please pass the `--no-enable-ip-alias` flag
Note: The Kubelet readonly port (10255) is now deprecated. Please update your workloads to use the recomme
nded alternatives. See https://cloud.google.com/kubernetes-engine/docs/how-to/disable-kubelet-readonly-por
t for ways to check usage and for migration instructions.
Note: Your Pod address range (`--cluster-ipv4-cidr`) can accommodate at most 1008 node(s).
Creating cluster spark in us-west1... Cluster is being health-checked (master is healthy)...done.
Created [https://container.googleapis.com/v1/projects/cloud-and-bigdata/zones/us-west1/clusters/spark].
To inspect the contents of your cluster, go to: https://console.cloud.google.com/kubernetes/workload_/gclo
ud/us-west1/spark?project=cloud-and-bigdata
kubeconfig entry generated for spark.
NAME: spark
LOCATION: us-west1
MASTER_VERSION: 1.29.4-gke.1043002
MASTER_IP: 34.83.221.222
MACHINE_TYPE: e2-highmem-2
NODE_VERSION: 1.29.4-gke.1043002
NUM_NODES: 3
STATUS: RUNNING
adagniew407@cloudshell:~ (cloud-and-bigdata)$
```

Implementation:

kubernetes

2.Install the NFS Server Provisioner

helm repo add stable https://charts.helm.sh/stable

helm repo update

```
adagniew407@cloudshell:~ (cloud-and-bigdata)$ helm repo add stable https://charts.helm.sh/stable
helm install nfs stable/nfs-server-provisioner --set persistence.enabled=true,persistence.size=5Gi
"stable" has been added to your repositories
WARNING: This chart is deprecated
NAME: nfs
LAST DEPLOYED: Fri Jun 28 10:54:38 2024
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
The NFS Provisioner service has now been installed.

A storage class named 'nfs' has now been created
and is available to provision dynamic volumes.

You can use this storageclass by creating a `PersistentVolumeClaim` with the
correct storageClassName attribute. For example:

    ---
    kind: PersistentVolumeClaim
    apiVersion: v1
    metadata:
      name: test-dynamic-volume-claim
    spec:
      storageClassName: "nfs"
      accessModes:
        - ReadWriteOnce
      resources:
        requests:
          storage: 100Mi
```

Implementation:

2.Install the NFS Server Provisioner

**helm install nfs stable/nfs-server-provisioner **

set persistence.enabled=true,persistence.size=5Gi

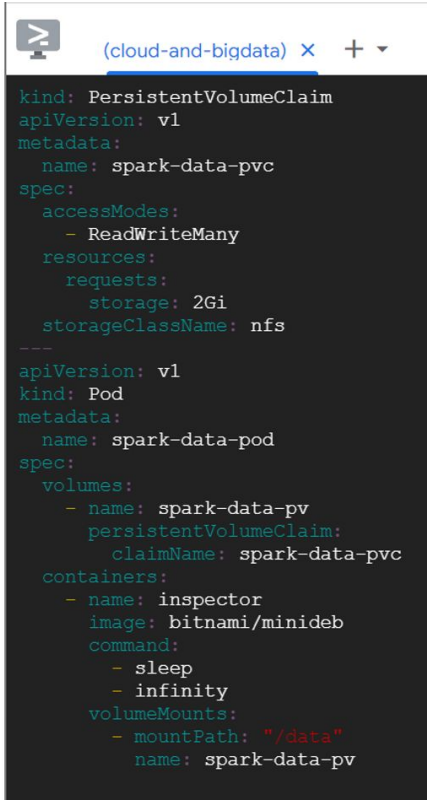
kubernetes

Apache Spark + Python = PySpark

```
adagniew407@cloudshell:~ (cloud-and-bigdata)$ helm repo add stable https://charts.helm.sh/stable
helm install nfs stable/nfs-server-provisioner --set persistence.enabled=true,persistence.size=5Gi
"stable" has been added to your repositories
WARNING: This chart is deprecated
NAME: nfs
LAST DEPLOYED: Fri Jun 28 10:54:38 2024
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
The NFS Provisioner service has now been installed.

A storage class named 'nfs' has now been created
and is available to provision dynamic volumes.

You can use this storageclass by creating a `PersistentVolumeClaim` with the
correct storageClassName attribute. For example:

    ---
    kind: PersistentVolumeClaim
    apiVersion: v1
    metadata:
      name: test-dynamic-volume-claim
    spec:
      storageClassName: "nfs"
      accessModes:
        - ReadWriteOnce
      resources:
        requests:
          storage: 100Mi
```

Implementation:

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: spark-data-pvc
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 2Gi
  storageClassName: nfs
---
apiVersion: v1
kind: Pod
metadata:
  name: spark-data-pod
spec:
  volumes:
    - name: spark-data-pv
      persistentVolumeClaim:
        claimName: spark-data-pvc
  containers:
    - name: inspector
      image: bitnami/minideb
      command:
        - sleep
        - infinity
      volumeMounts:
        - mountPath: "/data"
          name: spark-data-pv
```



Implementation:

kubernetes

4. Apply the above yaml descriptor

```
adagniew407@cloudshell:~ (cloud-and-bigdata)$ kubectl apply -f spark-pvc.yaml
persistentvolumeclaim/spark-data-pvc created
pod/spark-data-pod created
```

Implementation:

kubernetes

5. Create and prepare your application JAR file

```
docker run -v /tmp:/tmp -it bitnami/spark -- find
/opt/bitnami/spark/examples/jars/ -name spark-examples* -exec
cp {} /tmp/my.jar \;
```

```
adagniew407@cloudshell:~ (cloud-and-bigdata)$ docker run -v /tmp:/tmp -it bitnami/spark -- find /opt/bitnami/spark/examples/jars/
 -name spark-examples* -exec cp {} /tmp/my.jar \;
Unable to find image 'bitnami/spark:latest' locally
latest: Pulling from bitnami/spark
6d10d4f6c38d: Pull complete
Digest: sha256:9e997d4f9fb5ed0ac3942e7438478739f0243921792b0ade4479d11fbfcd6f8a
Status: Downloaded newer image for bitnami/spark:latest
spark 11:18:06.84 INFO  ==>
spark 11:18:06.84 INFO  ==> Welcome to the Bitnami spark container
spark 11:18:06.84 INFO  ==> Subscribe to project updates by watching https://github.com/bitnami/containers
spark 11:18:06.85 INFO  ==> Submit issues and feature requests at https://github.com/bitnami/containers/issues
spark 11:18:06.85 INFO  ==> Upgrade to Tanzu Application Catalog for production environments to access custom-configured and pre-
packaged software components. Gain enhanced features, including Software Bill of Materials (SBOM), CVE scan result reports, and V
EX documents. To learn more, visit https://bitnami.com/enterprise
spark 11:18:06.85 INFO  ==>
```

Implementation:

6.Add a test file with a line of words that we will be using later for the word count test

```
adagniew407@cloudshell:~ (cloud-and-bigdata)$ echo "how much wood could a woodpecker chuck if a woodpecker could chuck wood" > /t
mp/test.txt
adagniew407@cloudshell:~ (cloud-and-bigdata)$ cat /tmp/test.txt
how much wood could a woodpecker chuck if a woodpecker could chuck wood
```

Implementation:

7-Copy the JAR file containing the application, and any other required files, to the PVC using the mount point

kubectl cp /tmp/my.jar spark-data-pod:/data/my.jar

kubectl cp /tmp/test.txt spark-data-pod:/data/test.txt

```
adagniew407@cloudshell:~ (cloud-and-bigdata)$ kubectl cp /tmp/my.jar spark-data-pod:/data/my.jar
kubectl cp /tmp/test.txt spark-data-pod:/data/test.txt
```

Implementation:

8. Make sure the files a inside the persistent volume

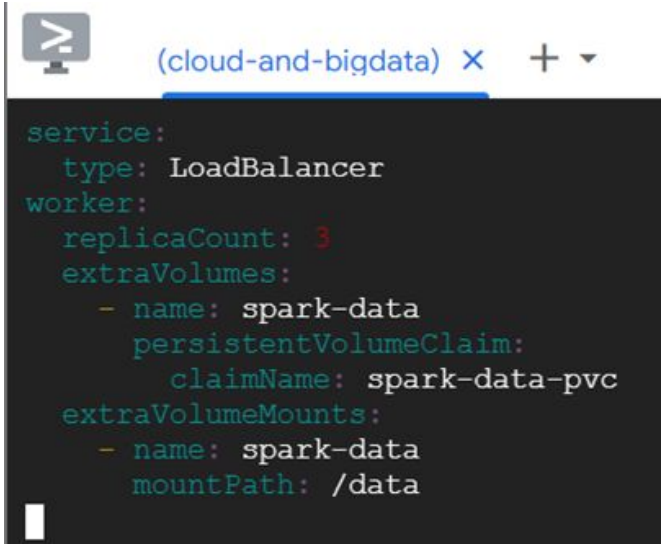
kubectl exec -it spark-data-pod -- ls -al /data

```
adagniew407@cloudshell:~ (cloud-and-bigdata)$ kubectl exec -it spark-data-pod -- ls -al /data
total 1540
drwxrwsrwx 2 root root    4096 Jun 28 11:20 .
drwxr-xr-x 1 root root    4096 Jun 28 11:16 ..
-rw-r--r-- 1 1001 root 1564260 Jun 28 11:20 my.jar
-rw-rw-r-- 1 1000 1000      72 Jun 28 11:20 test.txt
```

Implementation:

9. **Deploy Apache Spark on Kubernetes Using the Shared Volume:** Create a YAML file `spark-chart.yaml` with the following content:

```
service:
  type: LoadBalancer
worker:
  replicaCount: 3
  extraVolumes:
    - name: spark-data
      persistentVolumeClaim:
        claimName: spark-data-pvc
  extraVolumeMounts:
    - name: spark-data
      mountPath: /data
```



Implementation:

10. Deploy Apache Spark Using the Bitnami Helm Chart:

helm repo add bitnami https://charts.bitnami.com/bitnami
helm install spark bitnami/spark -f spark-chart.yaml

```
adegnie407@cloudshell:~ (cloud-and-bigdata)$ helm repo add bitnami https://charts.bitnami.com/bitnami
helm install spark bitnami/spark -f spark-chart.yaml
"bitnami" has been added to your repositories
NAME: spark
LAST DEPLOYED: Fri Jun 28 11:24:00 2024
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
CHART NAME: spark
CHART VERSION: 9.2.4
APP VERSION: 3.5.1

** Please be patient while the chart is being deployed **

1. Get the Spark master WebUI URL by running these commands:

    NOTE: It may take a few minutes for the LoadBalancer IP to be available.
    You can watch the status of by running 'kubectl get --namespace default svc -w spark-master-svc'

  export SERVICE_IP=$(kubectl get --namespace default svc spark-master-svc -o jsonpath="{.status.loadBalancer.ingress[0]['ip', 'hostname'] }")
  echo http://$SERVICE_IP:80

2. Submit an application to the cluster:

  To submit an application to the cluster the spark-submit script must be used. That script can be
  obtained at https://github.com/apache/spark/tree/master/bin. Also you can use kubectl run.

  Run the commands below to obtain the master IP and submit your application.

  export EXAMPLE_JAR=$(kubectl exec -ti --namespace default spark-worker-0 -- find examples/jars/ -name 'spark-example*\.jar' | tr -d '\r')
  export SUBMIT_IP=$(kubectl get --namespace default svc spark-master-svc -o jsonpath="{.status.loadBalancer.ingress[0]['ip', 'hostname'] }")

  kubectl run --namespace default spark-client --rm --tty -i --restart='Never' \
    --image docker.io/bitnami/spark:3.5.1-debian-12-r7 \
    -- spark-submit --master spark://$SUBMIT_IP:7077 \
    --deploy-mode cluster \
    --class org.apache.spark.examples.SparkPi \
    $EXAMPLE_JAR 1000

** IMPORTANT: When submit an application the --master parameter should be set to the service IP, if not, the application will not resolve the master. **



WARNING: There are "resources" sections in the chart not set. Using "resourcesPreset" is not recommended for production. For production installations, please set the following values according to
your workload needs:
  - master.resources
  - worker.resources
+info https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/
```

Implementation:

11. Deploy Apache Spark on the Kubernetes cluster using the Bitnami Apache Spark Helm chart and supply it with the configuration file above

helm repo add bitnami https://charts.bitnami.com/bitnami

```
taherzadeh19529@cloudshell:~/wordcount (phonic-axle-307118)$ helm repo add bitnami https://charts.bitnami.com/bitnami
"bitnami" has been added to your repositories
```

Implementation:

12. Get the external IP of the running pod

kubectl get svc -l "app.kubernetes.io/instance=spark,app.kubernetes.io/name=spark"

```
adagniew407@cloudshell:~ (cloud-and-bigdata)$ kubectl get svc -l "app.kubernetes.io/instance=spark,app.kubernetes.io/name
=spark"
NAME               TYPE           CLUSTER-IP       EXTERNAL-IP    PORT(S)                    AGE
spark-headless     ClusterIP      None             <none>         <none>                     95s
spark-master-svc   LoadBalancer   34.118.237.176   34.82.198.16   7077:32224/TCP,80:31299/TCP   95s
```

Implementation:

12. Get the external IP of the running pod

kubectl get svc -l "app.kubernetes.io/instance=spark,app.kubernetes.io/name=spark"

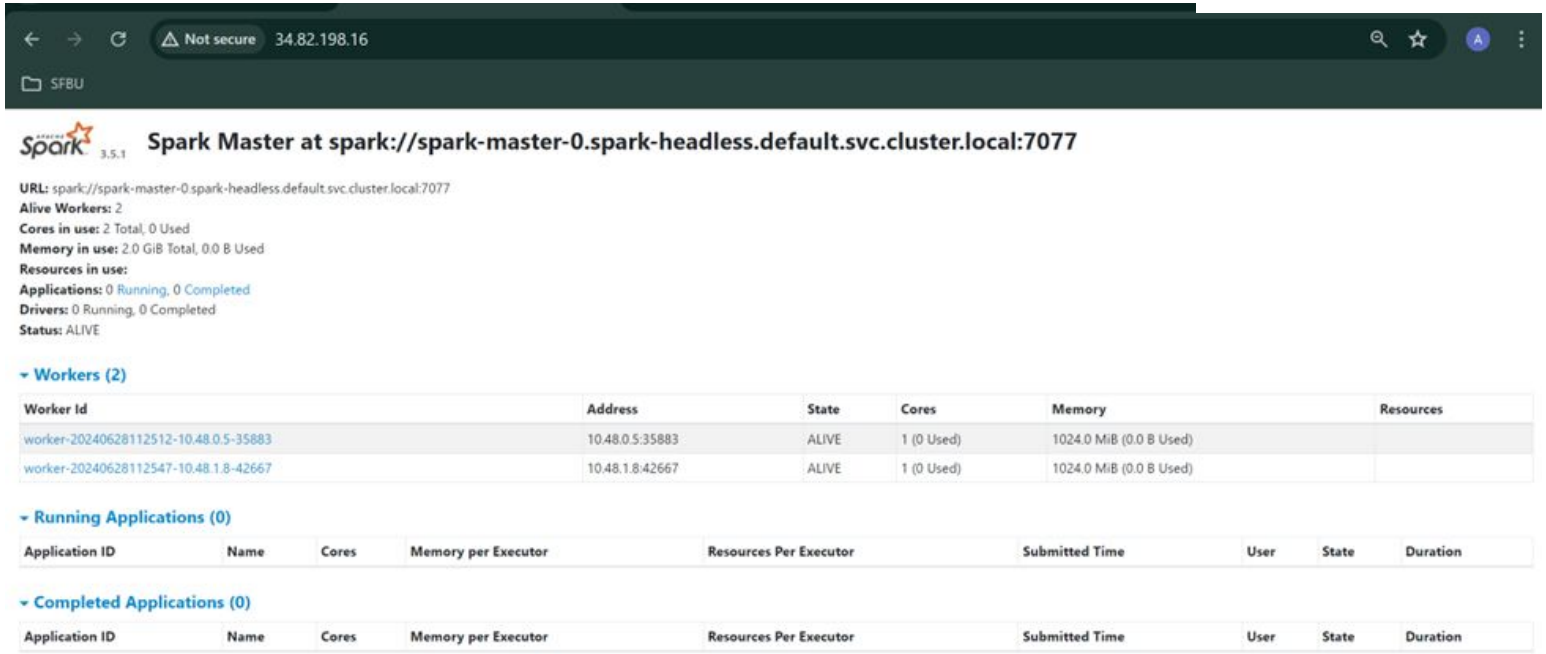
```
adagniew407@cloudshell:~ (cloud-and-bigdata)$ kubectl get svc -l "app.kubernetes.io/instance=spark,app.kubernetes.io/name
=spark"
NAME               TYPE           CLUSTER-IP       EXTERNAL-IP    PORT(S)                      AGE
spark-headless     ClusterIP      None             <none>         <none>                       95s
spark-master-svc   LoadBalancer   34.118.237.176   34.82.198.16   7077:32224/TCP,80:31299/TCP   95s
```

Test Result:

kubernetes

Apache Spark + Python = PySpark

1. Open the external ip on your browser



Test Result:

Word Count on Spark
Submit a word count task :

```
kubectl run --namespace default spark-client --rm --tty -i --restart='Never' \

--image docker.io/bitnami/spark:3.0.1-debian-10-r115 \

-- spark-submit --master spark://LOAD-BALANCER-External-ipADDRESS:7077 \
--deploy-mode cluster \

--class org.apache.spark.examples.JavaWordCount \
/data/my.jar /data/test.txt
```

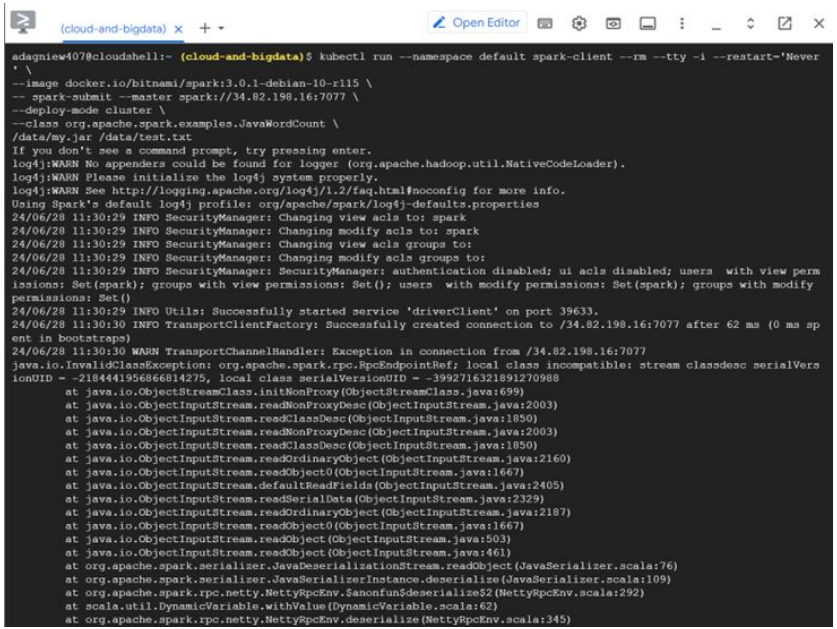
Test Result:

Error: Task Failed

Solution:

After encountering issues with submitting Spark jobs using `gcloud`, I switched to using `kubectl exec` from the Spark master node to successfully run the job.

```
kubectl exec -it spark-master-0 -- spark-submit
--master spark://34.82.198.16:7077
--deploy-mode cluster --class
org.apache.spark.examples.JavaWordCount
/data/my.jar /data/test.txt
```



Test Result:

kubernetes

Success

Solution:

After encountering issues with submitting Spark jobs using `gcloud`, I switched to using `kubectl exec` from the Spark master node to successfully run the job.

```
kubectl exec -it spark-master-0 -- spark-submit
--master spark://34.82.198.16:7077
--deploy-mode cluster --class
org.apache.spark.examples.JavaWordCount
/data/my.jar /data/test.txt
```

```
adagniew407@cloudshell:~ (cloud-and-bigdata)$ kubectl exec -it spark-master-0 -- bash
spark-submit --master spark://34.82.198.16:7077 --deploy-mode cluster --class org.apache.spark.exam
ples.JavaWordCount /data/my.jar /data/test.txt
I have no name!@spark-master-0:/opt/bitnami/spark$ spark-submit --master spark://34.82.198.16:7077
--deploy-mode cluster --class org.apache.spark.examples.JavaWordCount /data/my.jar /data/test.txt
24/06/28 11:44:38 INFO SecurityManager: Changing view acls to: spark
24/06/28 11:44:38 INFO SecurityManager: Changing modify acls to: spark
24/06/28 11:44:38 INFO SecurityManager: Changing view acls groups to:
24/06/28 11:44:38 INFO SecurityManager: Changing modify acls groups to:
24/06/28 11:44:38 INFO SecurityManager: SecurityManager: authentication disabled; ui acls disabled;
 users with view permissions: spark; groups with view permissions: EMPTY; users with modify permiss
ions: spark; groups with modify permissions: EMPTY
24/06/28 11:44:38 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform...
using builtin-java classes where applicable
24/06/28 11:44:40 INFO Utils: Successfully started service 'driverClient' on port 38629.
24/06/28 11:44:40 INFO TransportClientFactory: Successfully created connection to /34.82.198.16:707
7 after 111 ms (0 ms spent in bootstraps)
24/06/28 11:44:40 INFO ClientEndpoint: ... waiting before polling master for driver state
24/06/28 11:44:41 INFO ClientEndpoint: Driver successfully submitted as driver-20240628114440-0003
24/06/28 11:44:46 INFO ClientEndpoint: State of driver-20240628114440-0003 is RUNNING
24/06/28 11:44:46 INFO ClientEndpoint: Driver running on 10.48.0.5:35883 (worker-20240628112512-10.
48.0.5-35883)
24/06/28 11:44:46 INFO ClientEndpoint: spark-submit not configured to wait for completion, exiting
spark-submit JVM.
24/06/28 11:44:46 INFO ShutdownHookManager: Shutdown hook called
24/06/28 11:44:46 INFO ShutdownHookManager: Deleting directory /tmp/spark-944a4ef6-f307-49e1-9894-1
27faade3bdd
I have no name!@spark-master-0:/opt/bitnami/spark$
```

TestResult

And on your browser, you should see this task finished

SFBU

Alive Workers: 3
Cores in use: 3 Total, 0 Used
Memory in use: 3.0 GiB Total, 0.0 B Used
Resources in use:
Applications: 0 Running, 1 Completed
Drivers: 0 Running, 4 Completed
Status: ALIVE

Workers (3)

Worker Id	Address	State	Cores	Memory	Resources
worker-20240628112512-10.48.0.5-35883	10.48.0.5:35883	ALIVE	1 (0 Used)	1024.0 MiB (0.0 B Used)	
worker-20240628112547-10.48.1.8-42667	10.48.1.8:42667	ALIVE	1 (0 Used)	1024.0 MiB (0.0 B Used)	
worker-20240628112704-10.48.2.11-46029	10.48.2.11:46029	ALIVE	1 (0 Used)	1024.0 MiB (0.0 B Used)	

Running Applications (0)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration

Running Drivers (0)

Submission ID	Submitted Time	Worker	State	Cores	Memory	Resources	Main Class	Duration

Completed Applications (1)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
app-20240628114458-0000	JavaWordCount	2	1024.0 MiB		2024/06/28 11:44:58	spark	FINISHED	36 s

Completed Drivers (4)

Submission ID	Submitted Time	Worker	State	Cores	Memory	Resources	Main Class
driver-20240628114440-0003	2024/06/28 11:44:40	worker-20240628112512-10.48.0.5-35883	FINISHED	1	1024.0 MiB		org.apache.spark.examples.JavaWordCount
driver-20240628113558-0002	2024/06/28 11:35:58	worker-20240628112704-10.48.2.11-46029	FAILED	1	1024.0 MiB		org.apache.spark.examples.JavaWordCount
driver-20240628113505-0001	2024/06/28 11:35:05	worker-20240628112512-10.48.0.5-35883	FAILED	1	1024.0 MiB		org.apache.spark.examples.JavaWordCount
driver-20240628113030-0000	2024/06/28 11:30:30	worker-20240628112547-10.48.1.8-42667	FAILED	1	1024.0 MiB		org.apache.spark.examples.JavaWordCount

TestResult

kubernetes

Apache Spark + Python = PySpark

2. Get the name of the worker node

`kubectl get pods -o wide`

This command retrieves the IP address of the worker node that processed the word count task which is 10.48.0.5 in my case which we can see it in the website as well. The name is spark-worker-0

```
adagniew407@cloudshell:~ (cloud-and-bigdata)$ kubectl get pods -o wide
NAME                                READY   STATUS    RESTARTS   AGE   IP           NODE                                          NOMINATED NODE   READINESS GATES
nfs-nfs-server-provisioner-0        1/1     Running   0          66m   10.48.1.5    gke-spark-default-pool-0920d4e8-qvgq          <none>           <none>
spark-data-pod                      1/1     Running   0          44m   10.48.1.6    gke-spark-default-pool-0920d4e8-qvgq          <none>           <none>
spark-master-0                      1/1     Running   0          36m   10.48.1.7    gke-spark-default-pool-0920d4e8-qvgq          <none>           <none>
spark-worker-0                      1/1     Running   0          36m   10.48.0.5    gke-spark-default-pool-9fae7073-0s7s          <none>           <none>
spark-worker-1                      1/1     Running   0          35m   10.48.1.8    gke-spark-default-pool-0920d4e8-qvgq          <none>           <none>
spark-worker-2                      1/1     Running   0          34m   10.48.2.11   gke-spark-default-pool-2825ef6f-jg04          <none>           <none>
```

Submission ID	Submitted Time	Worker
driver-20240628114440-0003	2024/06/28 11:44:40	worker-20240628112512-10.48.0.5-35883

Test Result

kubernetes

3. Execute this pod and see the result of the finished tasks

kubectl exec -it spark-worker-0 -- bash

cd /opt/bitnami/spark/work

ls -l

```
adagniew407@cloudshell:~ (cloud-and-bigdata)$ kubectl exec -it spark-worker-0 -- bash
I have no name!@spark-worker-0:/opt/bitnami/spark$ cd /opt/bitnami/spark/work
I have no name!@spark-worker-0:/opt/bitnami/spark/work$ ls -l
total 8
drwxr-sr-x 2 1001 1001 4096 Jun 28 11:35 driver-20240628113505-0001
drwxr-sr-x 2 1001 1001 4096 Jun 28 11:44 driver-20240628114440-0003
```

Test Result

cd driver-20240628114440-0003

cat stdout

These commands allow you to access the worker node pod and view the result of the word count task by reading the `stdout` file.

```
adagniew407@cloudshell:~ (cloud-and-bigdata)$ kubectl exec -it spark-worker-0 -- bash
I have no name!@spark-worker-0:/opt/bitnami/spark$ cd /opt/bitnami/spark/work
I have no name!@spark-worker-0:/opt/bitnami/spark/work$ ls -l
total 8
drwxr-sr-x 2 1001 1001 4096 Jun 28 11:35 driver-20240628113505-0001
drwxr-sr-x 2 1001 1001 4096 Jun 28 11:44 driver-20240628114440-0003
I have no name!@spark-worker-0:/opt/bitnami/spark/work$ cd driver-20240628114440-0003
cat stdout
if: 1
a: 2
how: 1
could: 2
wood: 2
woodpecker: 2
much: 1
chuck: 2
```

Test Result

Running python PageRank onPySpark on the pods

Execute the spark master pods

kubectl exec -it spark-master-0 – bash

2. Stark pyspark

Pyspark

Error

```
I have no name!@spark-master-0:/opt/bitnami/spark$ pyspark
Error: pyspark does not support any application options.

Usage: ./bin/pyspark [options]

Options:
  --master MASTER_URL         spark://host:port, mesos://host:port, yarn,
                              k8s://https://host:port, or local (Default: local[*]).
  --deploy-mode DEPLOY_MODE   Whether to launch the driver program locally ("client") or
                              on one of the worker machines inside the cluster ("cluster")
```

Test Result

kubernetes

Apache Spark + Python = PySpark

If you face the above issue solution is available in the below github link:

https://github.com/bitnami/containers/issues/38139#issuecomment-1600923429

It seems to be the --name argument that is causing the issue in script: /opt/bitnami/spark/bin/pyspark - line 68:

exec "${SPARK_HOME}"/bin/spark-submit pyspark-shell-main --name "PySparkShell" "$@"

When I run the steps of that script manually without the --name arg, I can get an interactive PySpark shell:

export PYTHONPATH=/opt/bitnami/spark/python/lib/py4j-0.10.9.7-src.zip:/opt/bitnami/spark/python/:/opt/bitnami/spark/python/:

export PYTHONSTARTUP=/opt/bitnami/spark/python/pyspark/shell.py

exec "${SPARK_HOME}"/bin/spark-submit pyspark-shell-main

```
   /__ / .__/\_,_/_/ /_/\_\   version 3.5.1
      /_/

Using Python version 3.11.9 (main, May 13 2024 22:31:31)
Spark context Web UI available at http://spark-master-0.spark-headless.default.svc.cluster.local:4040
Spark context available as 'sc' (master = local[*], app id = local-1719579016707).
SparkSession available as 'spark'.
>>> exit()
```

Test Result


```
        at org.apache.spark.sql.execution.datasources.PartitioningUtils$.parsePartitions(PartitioningUtils.scala:178)
        at org.apache.spark.sql.execution.datasources.PartitioningUtils$.parsePartitions(PartitioningUtils.scala:110)
        at org.apache.spark.sql.execution.datasources.PartitioningAwareFileIndex.inferPartitioning(PartitioningAwareFileIndex.scala:201)
        at org.apache.spark.sql.execution.datasources.InMemoryFileIndex.partitionSpec(InMemoryFileIndex.scala:75)
        at org.apache.spark.sql.execution.datasources.PartitioningAwareFileIndex.partitionSchema(PartitioningAwareFileIndex.scala:51)
        at org.apache.spark.sql.execution.datasources.DataSource.getOrInferFileFormatSchema(DataSource.scala:167)
        at org.apache.spark.sql.execution.datasources.DataSource.resolveRelation(DataSource.scala:407)
        at org.apache.spark.sql.DataFrameReader.loadV1Source(DataFrameReader.scala:229)
        at org.apache.spark.sql.DataFrameReader.$anonfun$load$2(DataFrameReader.scala:211)
        at scala.Option.getOrElse(Option.scala:189)
        at org.apache.spark.sql.DataFrameReader.load(DataFrameReader.scala:211)
        at org.apache.spark.sql.DataFrameReader.text(DataFrameReader.scala:646)
        at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:77)
        at java.base/jdk.internal.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
        at java.base/java.lang.reflect.Method.invoke(Method.java:568)
        at py4j.reflection.MethodInvoker.invoke(MethodInvoker.java:244)
        at py4j.reflection.ReflectionEngine.invoke(ReflectionEngine.java:374)
        at py4j.Gateway.invoke(Gateway.java:282)
        at py4j.commands.AbstractCommand.invokeMethod(AbstractCommand.java:132)
        at py4j.commands.CallCommand.execute(CallCommand.java:79)
        at py4j.ClientServerConnection.waitForCommands(ClientServerConnection.java:182)
        at py4j.ClientServerConnection.run(ClientServerConnection.java:106)
        at java.base/java.lang.Thread.run(Thread.java:840)

24/06/28 13:08:26 INFO SparkContext: Invoking stop() from shutdown hook
24/06/28 13:08:26 INFO SparkContext: SparkContext is stopping with exitCode 0.
24/06/28 13:08:26 INFO SparkUI: Stopped Spark web UI at http://spark-master-0.spark-headless.default.svc.cluster.local:4040
24/06/28 13:08:26 INFO MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
24/06/28 13:08:26 INFO MemoryStore: MemoryStore cleared
24/06/28 13:08:26 INFO BlockManager: BlockManager stopped
24/06/28 13:08:26 INFO BlockManagerMaster: BlockManagerMaster stopped
24/06/28 13:08:26 INFO OutputCommitCoordinator$OutputCommitCoordinatorEndpoint: OutputCommitCoordinator stopped!
24/06/28 13:08:26 INFO SparkContext: Successfully stopped SparkContext
24/06/28 13:08:26 INFO ShutdownHookManager: Shutdown hook called
24/06/28 13:08:26 INFO ShutdownHookManager: Deleting directory /tmp/spark-305c4045-b8c7-4932-9923-c6c348c828dd
24/06/28 13:08:26 INFO ShutdownHookManager: Deleting directory /tmp/spark-54322e4f-c816-4ab4-a8a1-fcad19b08127/pyspark-0a742ffb-961d-40a4-8f23-b6cb5043a2a0
24/06/28 13:08:26 INFO ShutdownHookManager: Deleting directory /tmp/spark-54322e4f-c816-4ab4-a8a1-fcad19b08127
```

Enhancement Ideas

1. **Performance Optimization:**
   - Optimize Spark jobs with partitioning, caching, and broadcast variables.
   - Use Kubernetes autoscaling for dynamic Spark worker node management.
2. **Fault Tolerance and Reliability:**
   - Implement Spark checkpoints and Kubernetes anti-affinity rules.
   - Ensure job recovery and resilience against node failures.
3. **Security and Access Control:**
   - Integrate Kubernetes RBAC for secure cluster access.
   - Implement network policies to enhance Spark cluster security.
4. **Monitoring and Logging:**
   - Use Prometheus and Grafana for Spark and cluster monitoring.
   - Centralize logging with ELK stack for analysis and troubleshooting.
5. **Integration with Data Pipelines:**
   - Integrate Spark with Kafka for real-time data processing.
   - Use Kubernetes Operators for automated Spark application management.
6. **Cost Optimization:**
   - Utilize Kubernetes spot instances for cost-effective Spark deployments.
   - Set resource quotas and limits to optimize spending.
7. **Scalability and Elasticity:**
   - Implement Spark dynamic resource allocation on Kubernetes.
   - Use StatefulSets for managing stateful Spark applications.

Conclusion

This project utilized PySpark for implementing Word Count and PageRank on Apache Spark deployed on Kubernetes. Spark on Kubernetes leverages containerization for portability, efficient resource sharing for cost savings, and integration within a diverse ecosystem, promoting cloud-agnosticism and reducing vendor lock-in.

References

- https://www.datamechanics.co/blog-post/pros-and-cons-of-running-apache-spark-on-kubernetes
- https://towardsdatascience.com/how-to-guide-set-up-manage-monitor-spark-on-kubernetes-with-code-examples-c5364ad3aba2
- https://www.datamechanics.co/apache-spark-on-kubernetes
- https://spark.apache.org/docs/latest/running-on-kubernetes.html
- https://npu85.npu.edu/~henry/npu/classes/master_apache_spark/kubernetes/slide/exercise_kubernetes.html
- https://npu85.npu.edu/~henry/npu/classes/master_apache_spark/kubernetes/slide/index_slide.html

Appendix

https://github.com/ASD-Are/Big_Data/tree/main/Work%20Count%20%2B%20PageRank

Google Slide