Wordcount + PageRank

Kadirbek Sharau

19995

kubernetes

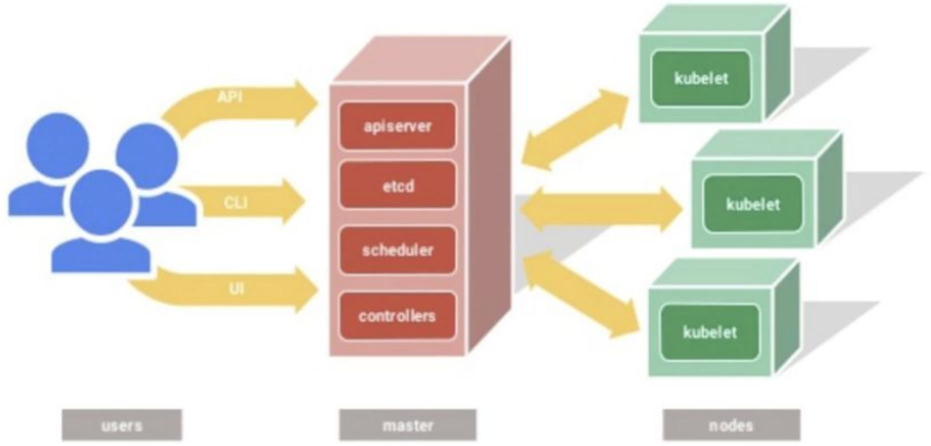
Apache Spark + Python = PySpark

Table of Content:

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.

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.

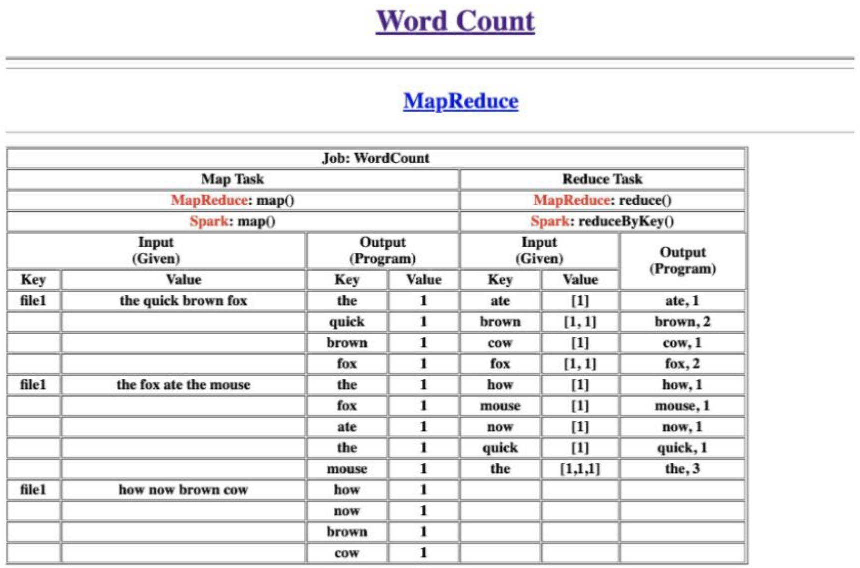


Design

Spark Concept

Spark:

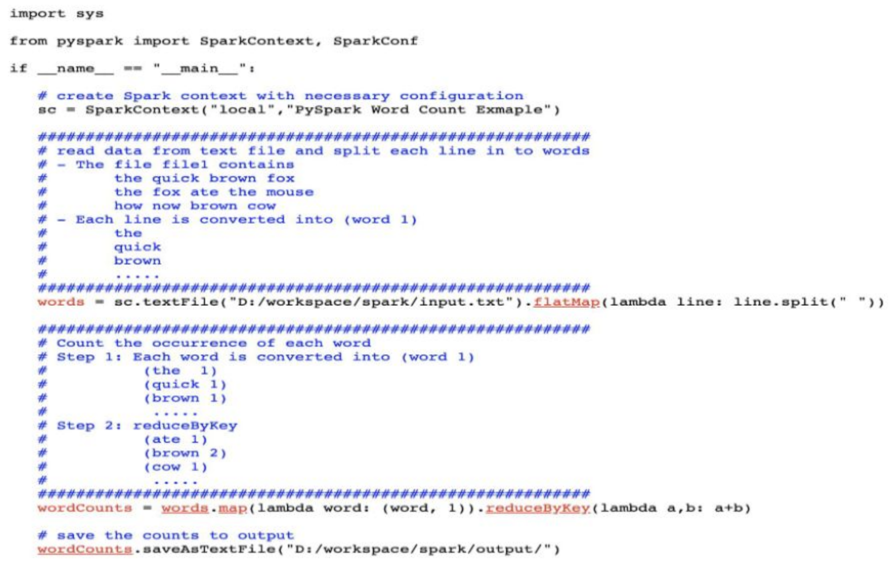
Apache Spark is a multi-purpose distributed computing framework known for its in-memory processing capabilities. It provides high-level APIs in Java, Scala, and Python, coupled with an optimized execution engine that can handle various execution graphs. Spark also offers specialized tools such as Spark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



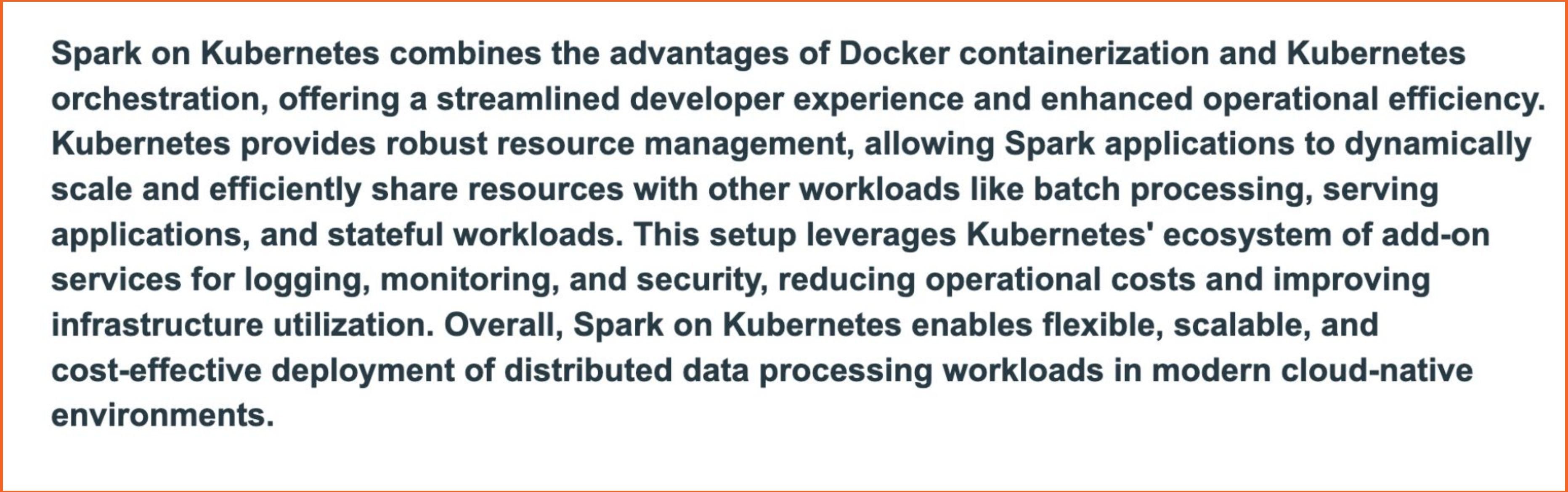
Word Count

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			

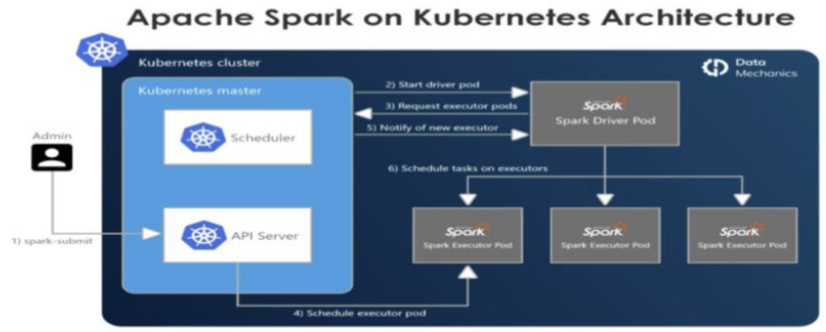


```
import sys

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/")
```


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.

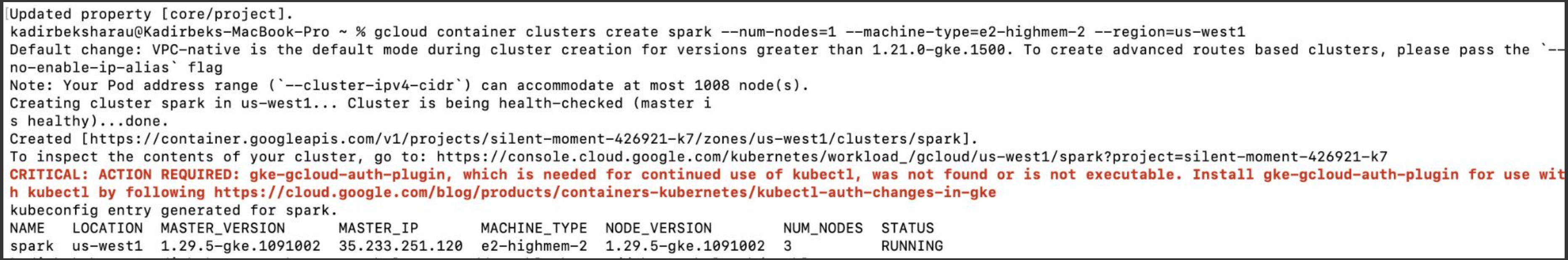
Spark-Submit simplifies Spark application submission to Kubernetes:

- 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.

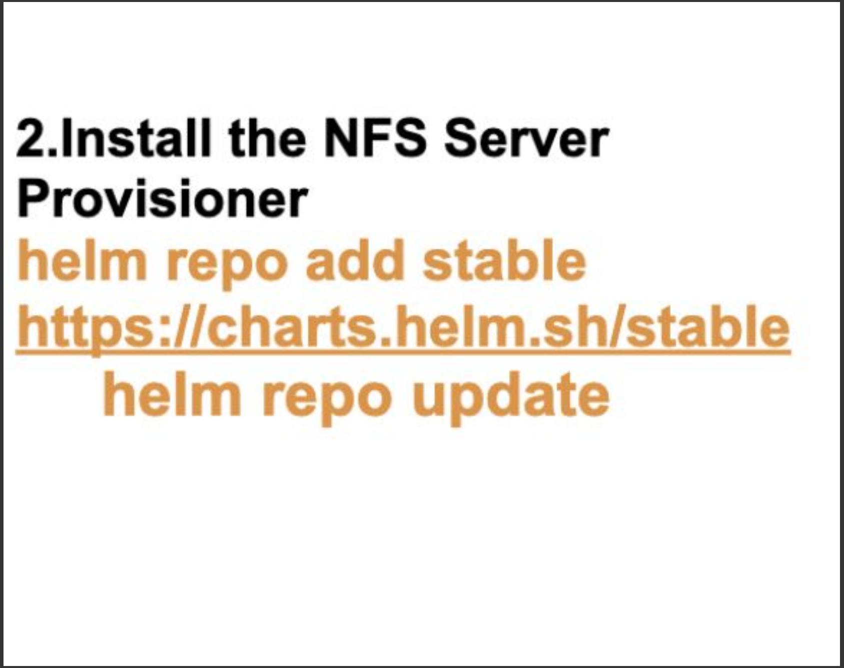


1. Create a cluster on GKE with

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

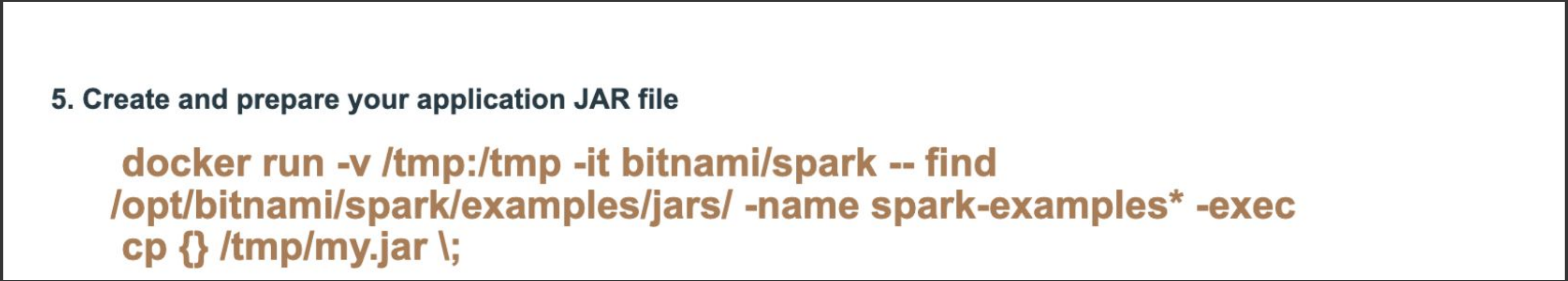


```
Updated property [core/project].
kadirbeksharau@Kadirbeks-MacBook-Pro ~ % gcloud container clusters create spark --num-nodes=1 --machine-type=e2-highmem-2 --region=us-west1
Default change: VPC-native is the default mode during cluster creation for versions greater than 1.21.0-gke.1500. To create advanced routes based clusters, please pass the `--
no-enable-ip-alias` flag
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 i
s healthy)...done.
Created [https://container.googleapis.com/v1/projects/silent-moment-426921-k7/zones/us-west1/clusters/spark].
To inspect the contents of your cluster, go to: https://console.cloud.google.com/kubernetes/workload_/gcloud/us-west1/spark?project=silent-moment-426921-k7
CRITICAL: ACTION REQUIRED: gke-gcloud-auth-plugin, which is needed for continued use of kubectl, was not found or is not executable. Install gke-gcloud-auth-plugin for use wit
h kubectl by following https://cloud.google.com/blog/products/containers-kubernetes/kubectl-auth-changes-in-gke
kubeconfig entry generated for spark.
NAME   LOCATION  MASTER_VERSION      MASTER_IP       MACHINE_TYPE  NODE_VERSION        NUM_NODES  STATUS
spark  us-west1  1.29.5-gke.1091002  35.233.251.120  e2-highmem-2  1.29.5-gke.1091002  3          RUNNING
```



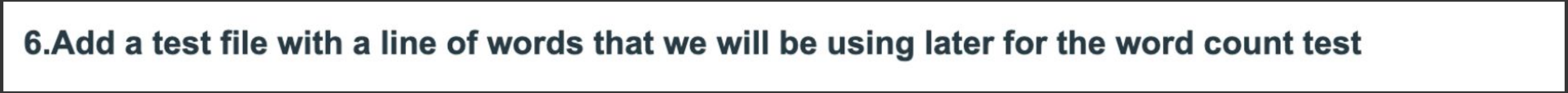
2.Install the NFS Server Provisioner

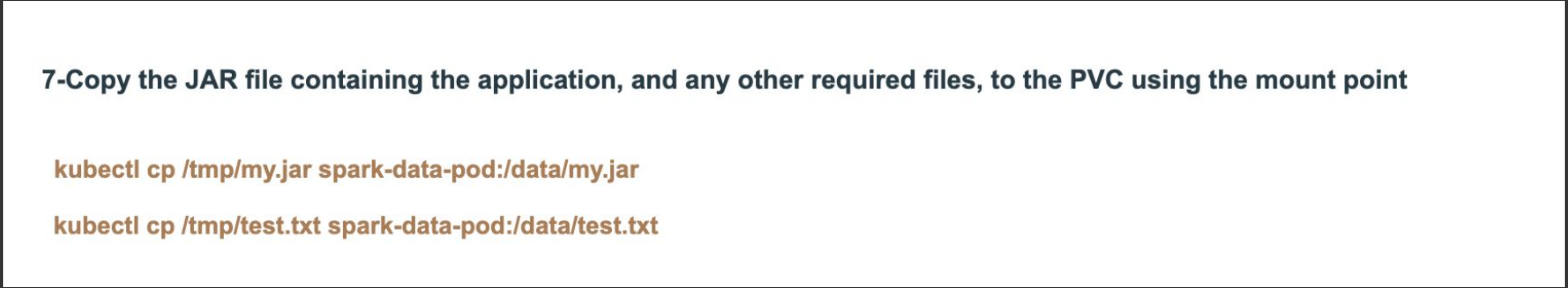
```
helm repo add stable https://charts.helm.sh/stable
    helm repo update
```



5. Create and prepare your application JAR file

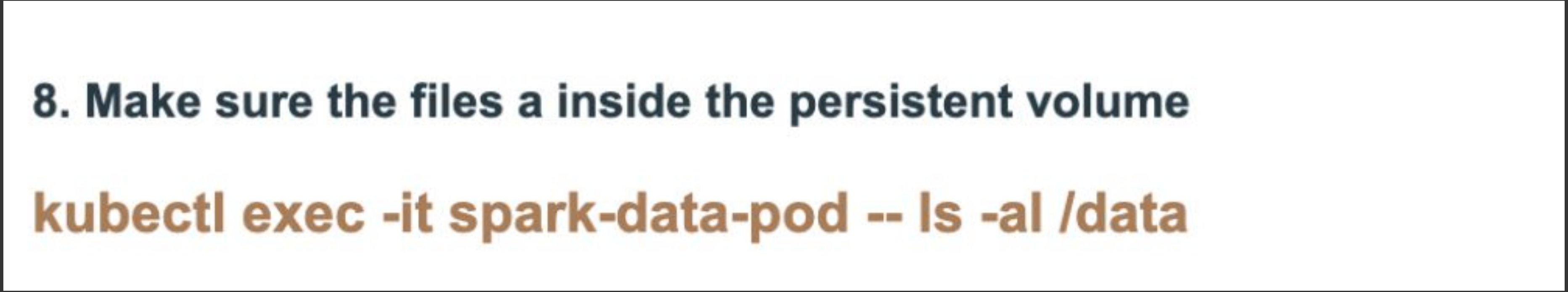
```
 docker run -v /tmp:/tmp -it bitnami/spark -- find
/opt/bitnami/spark/examples/jars/ -name spark-examples* -exec
 cp {} /tmp/my.jar \;
```


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



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
```

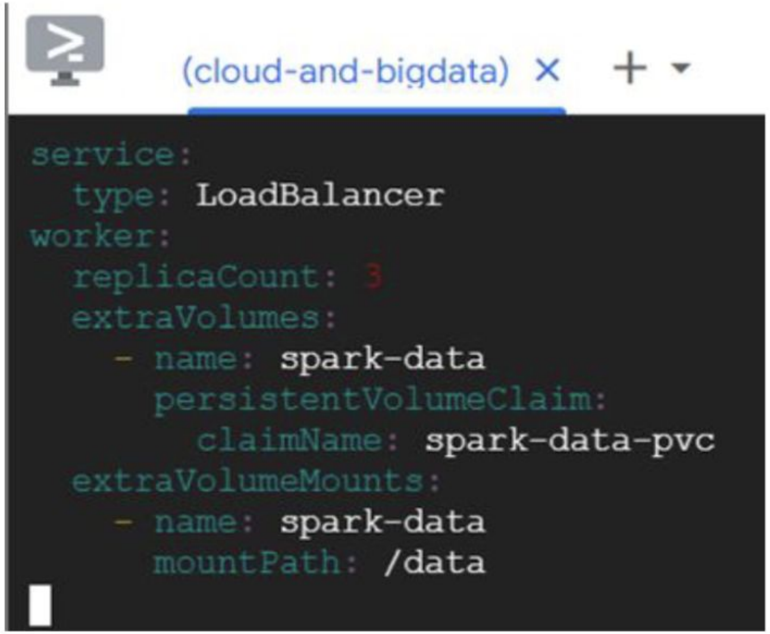


8. Make sure the files a inside the persistent volume

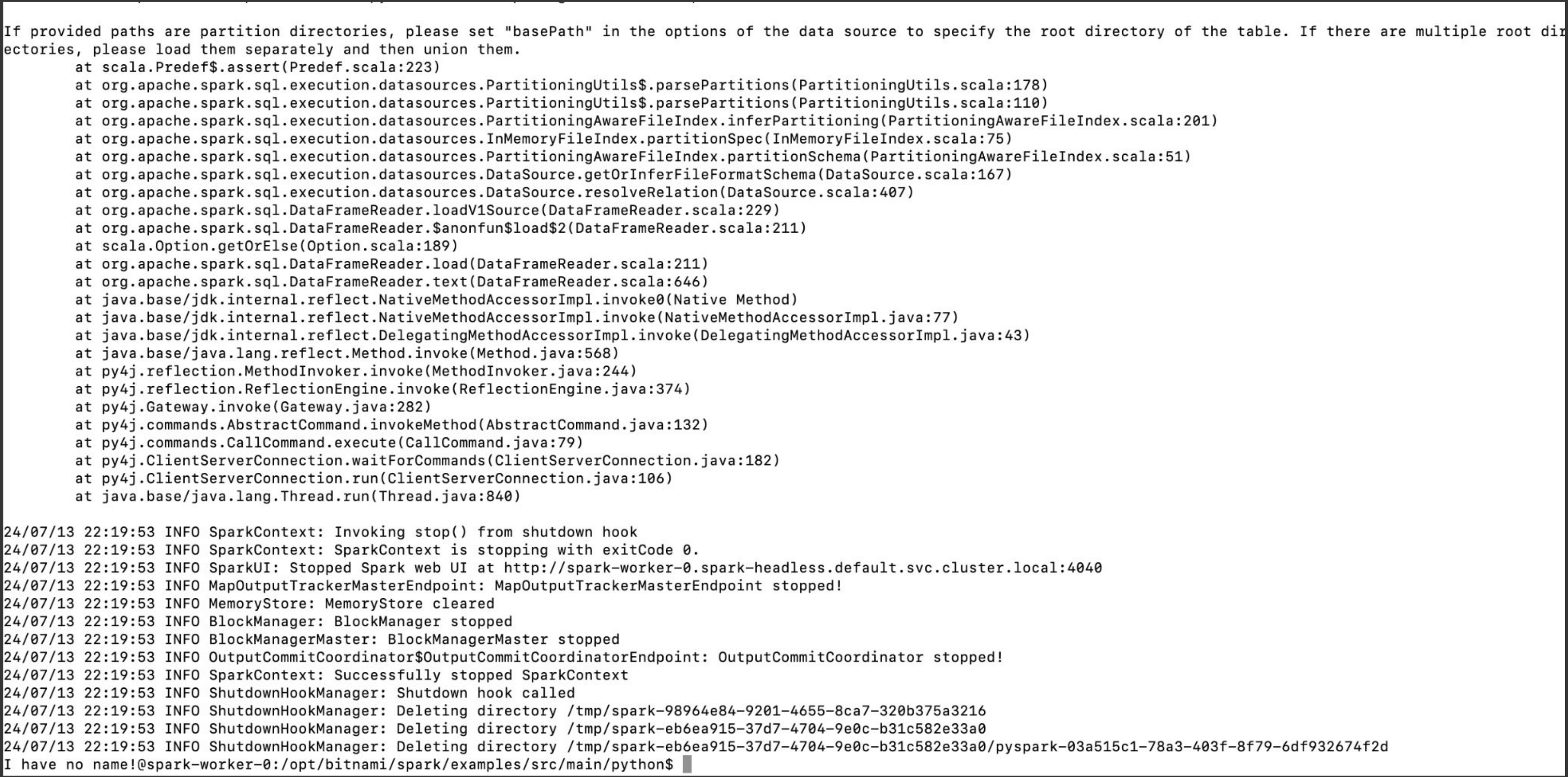
```
kubectl exec -it spark-data-pod -- ls -al /data
```

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
```



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



```
If provided paths are partition directories, please set "basePath" in the options of the data source to specify the root directory of the table. If there are multiple root dir
ectories, please load them separately and then union them.
        at scala.Predef$.assert(Predef.scala:223)
        at org.apache.spark.sql.execution.datasources.PartitioningUtils$.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)

24/07/13 22:19:53 INFO SparkContext: Invoking stop() from shutdown hook
24/07/13 22:19:53 INFO SparkContext: SparkContext is stopping with exitCode 0.
24/07/13 22:19:53 INFO SparkUI: Stopped Spark web UI at http://spark-worker-0.spark-headless.default.svc.cluster.local:4040
24/07/13 22:19:53 INFO MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
24/07/13 22:19:53 INFO MemoryStore: MemoryStore cleared
24/07/13 22:19:53 INFO BlockManager: BlockManager stopped
24/07/13 22:19:53 INFO BlockManagerMaster: BlockManagerMaster stopped
24/07/13 22:19:53 INFO OutputCommitCoordinator$OutputCommitCoordinatorEndpoint: OutputCommitCoordinator stopped!
24/07/13 22:19:53 INFO SparkContext: Successfully stopped SparkContext
24/07/13 22:19:53 INFO ShutdownHookManager: Shutdown hook called
24/07/13 22:19:53 INFO ShutdownHookManager: Deleting directory /tmp/spark-98964e84-9201-4655-8ca7-320b375a3216
24/07/13 22:19:53 INFO ShutdownHookManager: Deleting directory /tmp/spark-eb6ea915-37d7-4704-9e0c-b31c582e33a0
24/07/13 22:19:53 INFO ShutdownHookManager: Deleting directory /tmp/spark-eb6ea915-37d7-4704-9e0c-b31c582e33a0/pyspark-03a515c1-78a3-403f-8f79-6df932674f2d
I have no name!@spark-worker-0:/opt/bitnami/spark/examples/src/main/python$
```