Chapter 11

BigData/Blockchain based solutions using Python

Introduction

Reader will learn the concepts of big data and distributed ledgers. They will be presented with examples on how to build big data solutions and blockchain based solutions. Big Data frameworks have got popular and opensource solutions are being used by developers. Block chain is getting adopted in finance world by the python developers.

Structure

In this chapter, we will discuss the following topics:

▪ BigQuery

▪ PySpark

▪ Pydoop & Kafka

▪ Blockchain using Python

Let us first look at the popular python frameworks in Big data opensource projects.

**Figure 11.1**: Big Data Frameworks

Big Data is the term related to very large data in a datacentre or a data cluster. Python language can handle large data and can process it better than other languages. The key challenges of big data are:

* Volume
* Variety
* Velocity
* Variability

Python frameworks based big data solutions can secure the data and persist the data into data sources. Big Data is getting adopted very quickly and there is a prediction around 100 billion $ by 2027. Current solutions have issues like handling data variations, data processing performance, compatibility with other tech stacks, Accuracy of the culled out insights, and scarcity of Big Data skilled developers. Python developers are overcoming these issues by using the opensource frameworks by handling memory issues, splitting of big data sets, handling smaller chunks in memory, using lazy evaluation, and reading/writing from data sources. Python language is easy to learn and opensource frameworks make it easy for the developers to learn Big Data solutions. The basics required for developers to excel in the big data area are:

* Mathematics
* Statistics
* Scientific Functions
* Calculus

Python Django, Flask, and other web application frameworks integrate well with these opensource frameworks. Opensource frameworks are popular as the developers have capability to see the code and change if necessary. These solutions based on opensource frameworks are portable on different operating systems and can be deployed on multi cloud environments. These frameworks can be used to build big data apps like Data Visualization, Analysis, Statistical Analysis, Numerical Analysis, Computational Intensive Tasks and Machine Learning applications.

There are other libraries which can help in building the big data solutions. They are:

* Pandas
* Numpy
* SciPy
* Scikit-learn
* Dask
* Dmelt.

Big data solutions based on opensource frameworks can handle different data formats like:

* XML
* JSON
* CSV
* XLS
* HTML
* SQL

Complex solutions can be built first as proof of concepts or technology and then later developed to large scale bigdata solutions. These solutions based on bigdata opensource frameworks can handle multi-user and multi-tenant-based solutions. These frameworks provide APIs, NLP services, Data Indexing, Search, Data processing, and conversion features. Data mining helps in identifying the patterns, trends, and correlations in the enterprise data. The data visualization packages provide plotting, charting, and analyzing the large data using python. These packages are:

* Matplotlib
* Plotly
* NetworkX
* Pyga
* Ggplot
* Seaborn
* Altair

Python based data solutions are becoming popular as more and more web/mobile/desktop/multitouch/smart UI applications are being built in python. Python language has features which are useful for computationally intensive tasks. These features are:

* Linked Lists
* Tuples
* Sets
* Dictionaries

Python libraries have the capabilities to handle the below:

* Data Frames
* Matrix based Calculations
* Scientific Computing tasks

Different communities like CodeMentor GitHub, Stack Overflow, Git Lab, and Real Python. Opensource packages in python have crossed 130k number and popularity continues to grow at rapid pace.

Now let us look at Big Query big data solution.

BigQuery

Big Query was created by Google team. It is based out of the Google Cloud. Big table is one of the components of Big Query platform. This platform can handle large data and process the requests for data slicing/dicing and other large data operations. This Big Query framework provides REST based API to analyse the large data. The Big query framework has features to extract, transform, and load data from/to the data sources. Big Query can handle data in size of petabytes and you can create data lake.

To use Big Query, you need to have a python client talking to the Big query API. You can also use Tylertreat (https://github.com/tylertreat/BigQuery-Python) opensource package to process data on Big Query. As always, you need to have the python environment setup done.

To setup the python environment, you can use the command below:

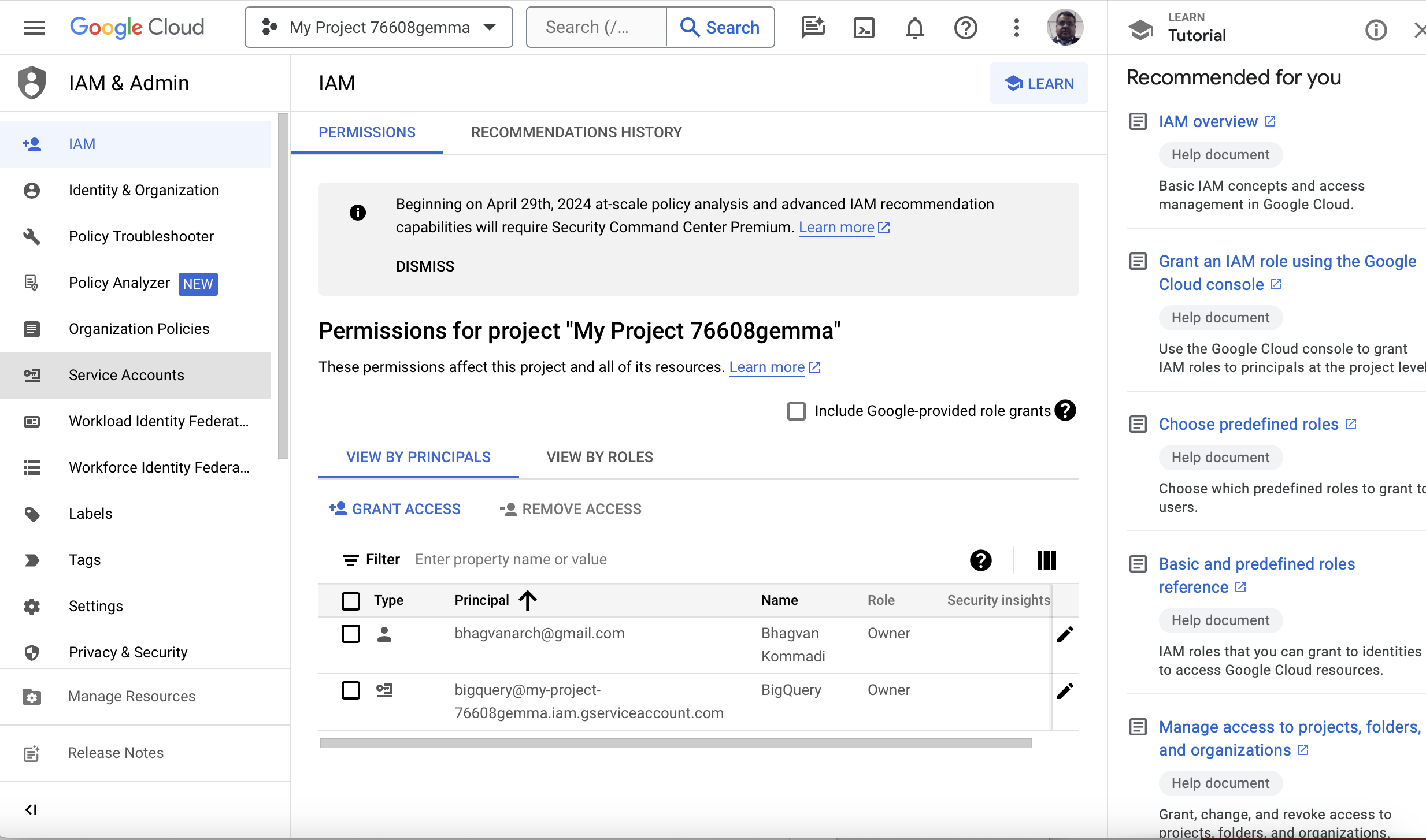
python3 -m venv bigdata\_env

After setting up the python development environment. You can use the following commands:

pip3 install --upgrade google-cloud-bigquery

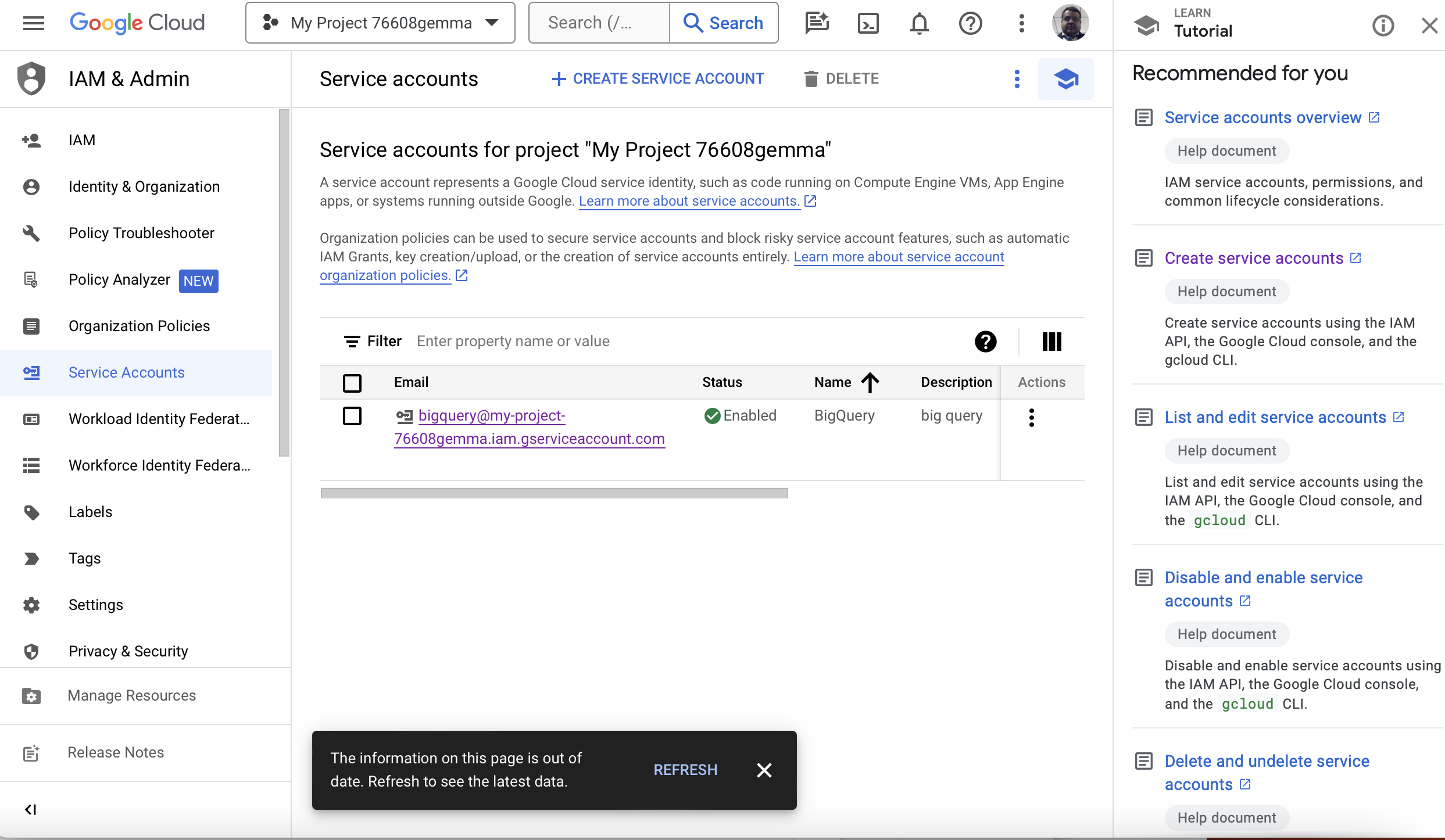
You need to create a service account on google cloud with Big Query service enabled.

You can access the Service Accounts from the dashboard on the left menu.



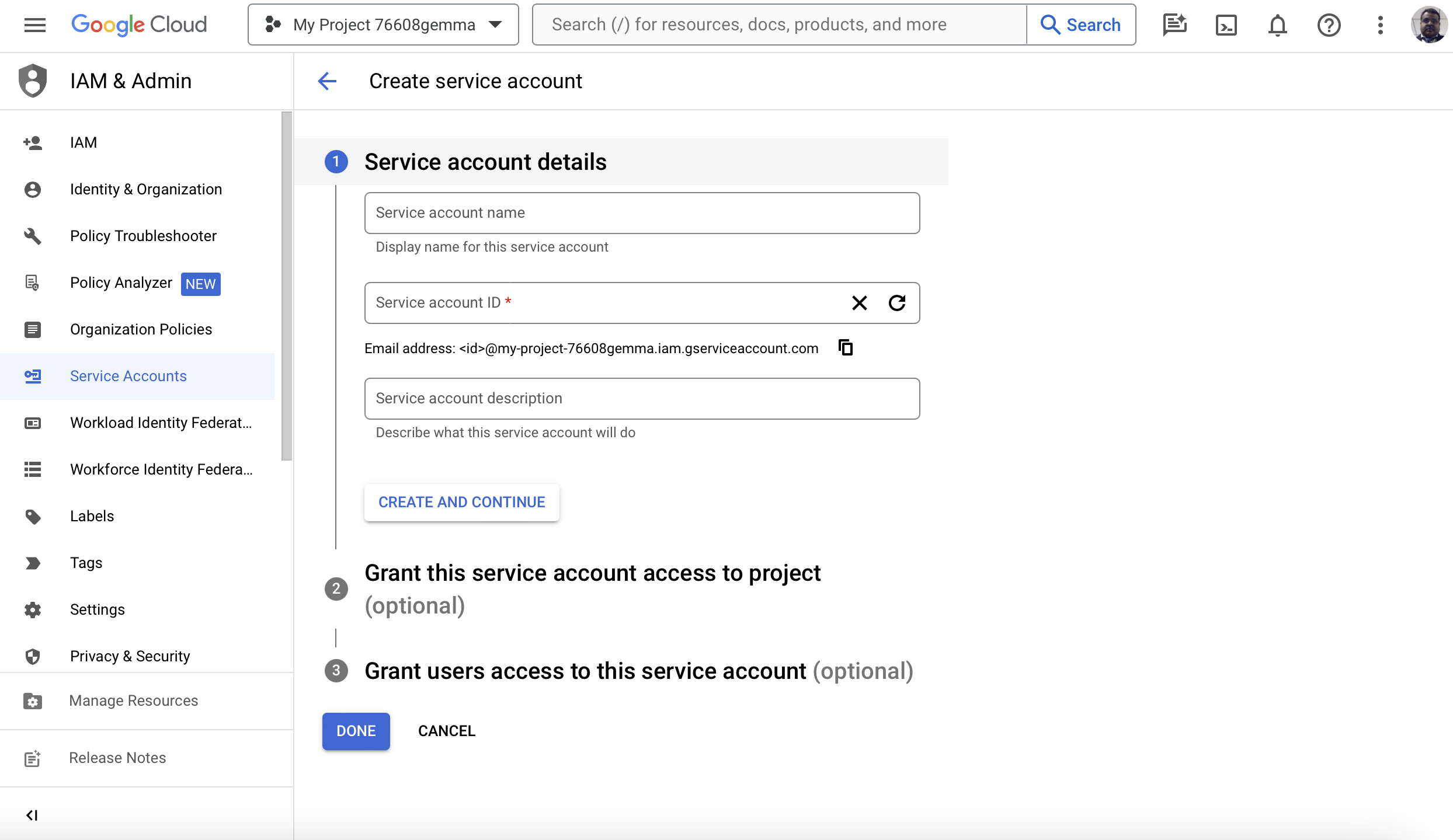
**Figure 11.2**: Google Cloud Service Accounts

If you do not have a project, create a project and enable Big Query API with billing account. You will see the service accounts for that project as shown below:



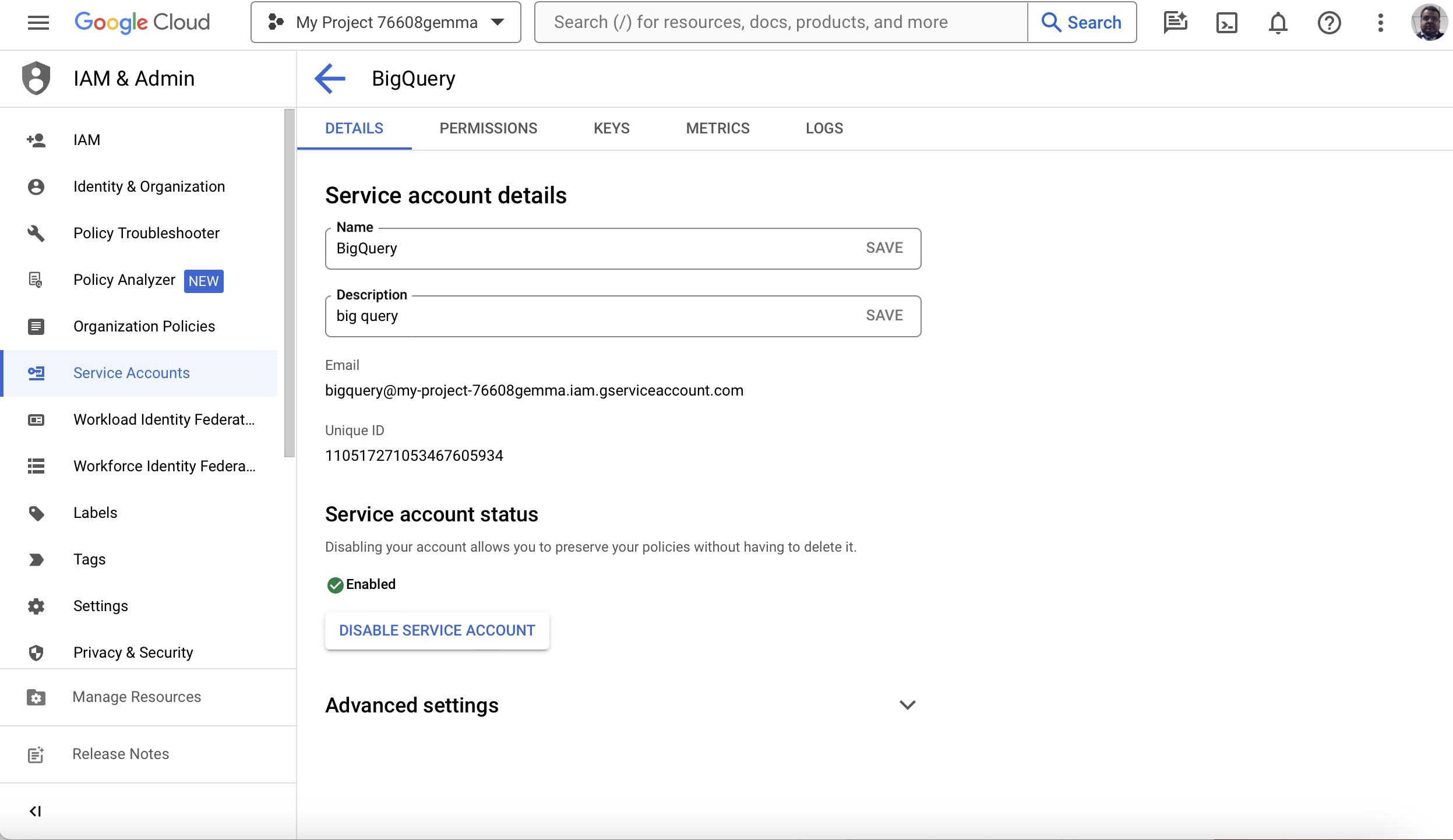
**Figure 11.3**: Google Project Service Account

If you do not have a service account, create one service account and grant the required privileges (owner is the default).



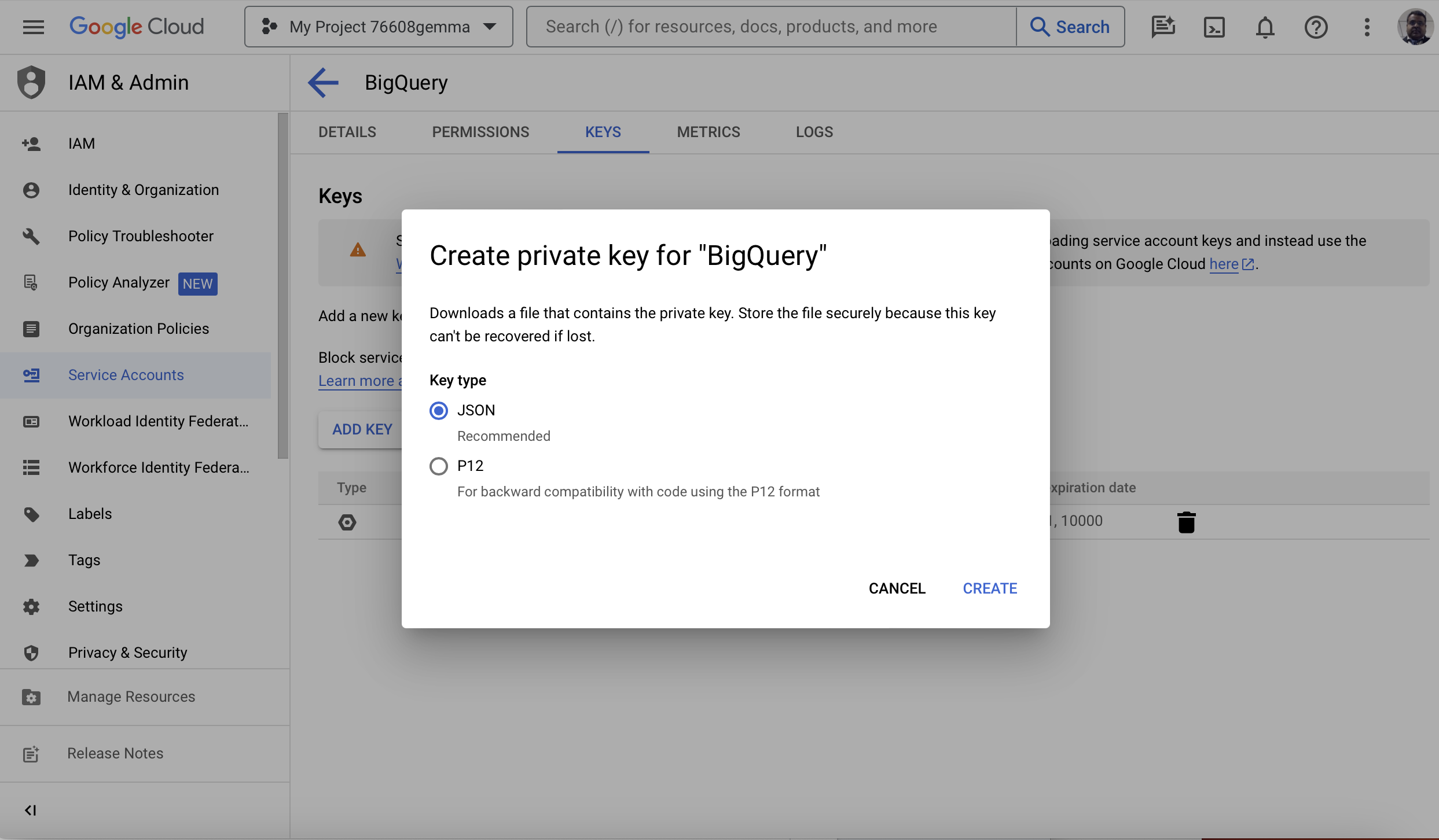
**Figure 11.4**: Create a service account

After creation, you can see the service account details as shown below:



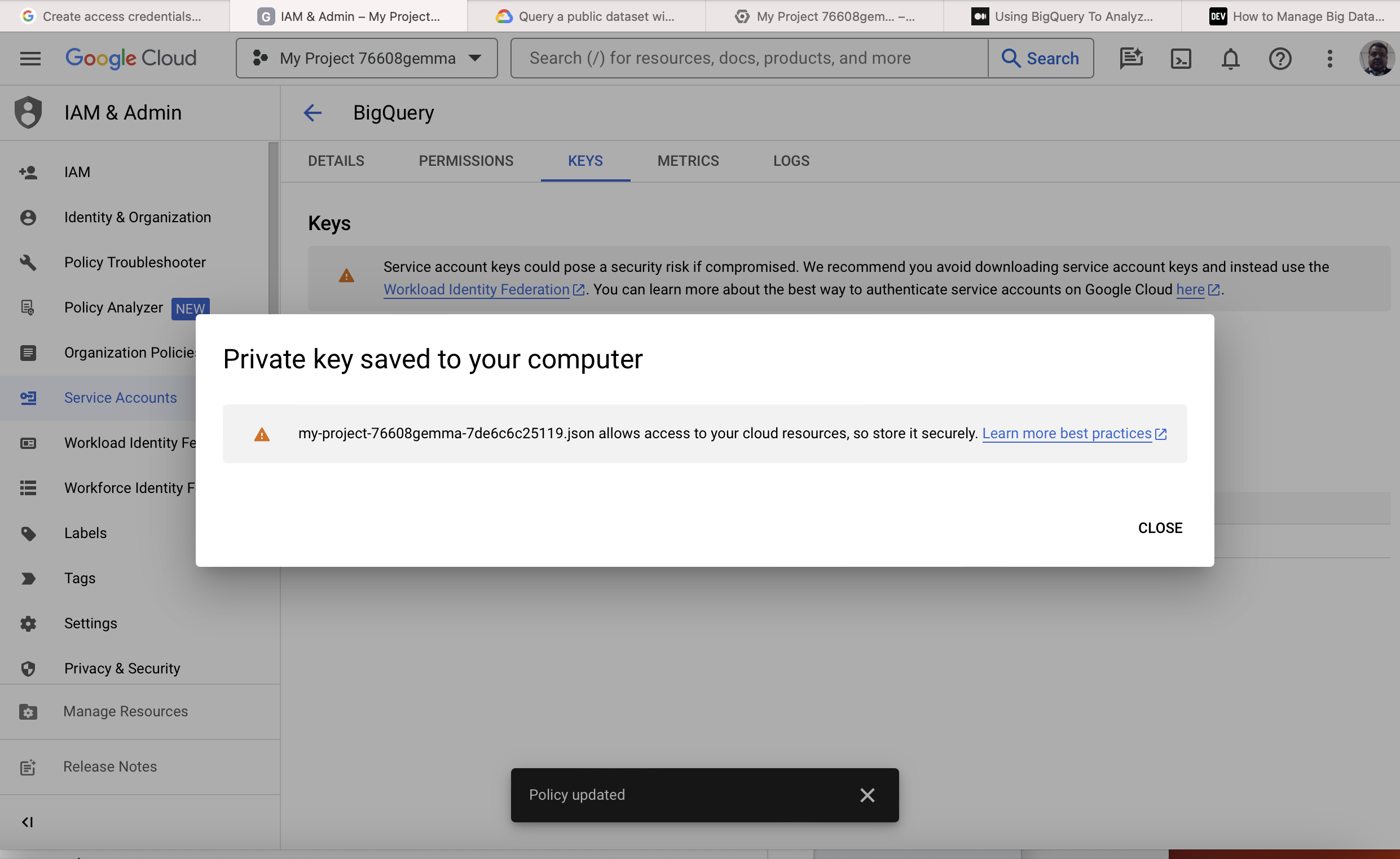
**Figure 11.5**: Service Account Details

Now you need to create a private key of JSON Type as shown below:



**Figure 11.6:** JSON Private Key

Now you can save the private key of JSON type as JSON file on your computer as shown below.



**Figure 11.7**: JSON File Saved on the disk

You can download the service account JSON and set the key path using the command below:

export key\_path = “public\_project\_g.json”

**querying\_publicdata.py**

from google.cloud import bigquery

from google.oauth2 import service\_account

account\_credentials = service\_account.Credentials.from\_service\_account\_file(

filename="public\_data\_g.json"

)

big\_query\_client = bigquery.Client(

credentials=account\_credentials,

project=account\_credentials.project\_id,

)

public\_data = big\_query\_client.dataset('cms\_medicare', project='bigquery-public-data')

print([data\_value.table\_id for data\_value in big\_query\_client.list\_tables(public\_data)])

The code above can be executed using the command below:

source big\_query\_env/bin/activate

python3 querying\_publicdata.py

The output when the command above is executed, is shown below:

(big\_query\_env) bhagvanarch@Bhagvans-MacBook-Air bigquery % python3 querying\_public\_data.py

['home\_health\_agencies\_2013', 'home\_health\_agencies\_2014', 'hospice\_providers\_2014', 'hospital\_general\_info', 'inpatient\_charges\_2011', 'inpatient\_charges\_2012', 'inpatient\_charges\_2013', 'inpatient\_charges\_2014', 'inpatient\_charges\_2015', 'nursing\_facilities\_2013', 'nursing\_facilities\_2014', 'outpatient\_charges\_2011', 'outpatient\_charges\_2012', 'outpatient\_charges\_2013', 'outpatient\_charges\_2014', 'outpatient\_charges\_2015', 'part\_d\_prescriber\_2014', 'physicians\_and\_other\_supplier\_2012', 'physicians\_and\_other\_supplier\_2013', 'physicians\_and\_other\_supplier\_2014', 'physicians\_and\_other\_supplier\_2015', 'referring\_durable\_medical\_equip\_2013', 'referring\_durable\_medical\_equip\_2014']

You can see in the data above as tables in the public data source focusing on healthcare in U.S.A.

Now let us look at making queries in another public data set using SQL. Below is the code sample:

**big\_query\_example.py**

from google.cloud import bigquery

from google.oauth2 import service\_account

account\_credentials = service\_account.Credentials.from\_service\_account\_file(

filename="public\_data\_g.json"

)

big\_query\_client = bigquery.Client(

credentials=account\_credentials,

project=account\_credentials.project\_id,

)

sql\_query = (

'SELECT name FROM `bigquery-public-data.usa\_names.usa\_1910\_2013` '

'WHERE state = "CA" '

'LIMIT 10')

big\_query\_job = big\_query\_client.query(sql\_query)

data\_rows = big\_query\_job.result()

for data\_row in data\_rows:

print(data\_row.name)

The code above can be executed using the command below:

source big\_query\_env/bin/activate

python3 big\_query\_example.py

The output when the command above is executed, is shown below:

(big\_query\_env) bhagvanarch@Bhagvans-MacBook-Air bigquery % python3 big\_query\_example.py

Beatrice

Marion

Thelma

Muriel

Anita

Gladys

Mildred

Victoria

Vivian

Audrey

(big\_query\_env) bhagvanarch@Bhagvans-MacBook-Air bigquery %

You can see in the names in public data of USA (limited by 10 count) in California.

Now let us look at saving the data in csv. Below is the code sample:

**big\_query\_example.py**

from google.cloud import bigquery

from google.oauth2 import service\_account

account\_credentials = service\_account.Credentials.from\_service\_account\_file(

filename="public\_data\_g.json"

)

big\_query\_client = bigquery.Client(

credentials=account\_credentials,

project=account\_credentials.project\_id,

)

sql\_query = (

'SELECT name FROM `bigquery-public-data.usa\_names.usa\_1910\_2013` '

'WHERE state = "CA" '

'LIMIT 10')

big\_query\_job = big\_query\_client.query(sql\_query)

data\_rows = big\_query\_job.result()

for data\_row in data\_rows:

print(data\_row.name)

data\_frame = big\_query\_job.to\_dataframe()

data\_frame.to\_csv('public\_usa\_names\_data.csv', index=False,header=True)

print('public data is written in csv')

The code above can be executed using the command below:

source big\_query\_env/bin/activate

python3 big\_query\_example.py

The output when the command above is executed, is shown below:

(big\_query\_env) bhagvanarch@Bhagvans-MacBook-Air bigquery % python3 big\_query\_example.py

Beatrice

Marion

Thelma

Muriel

Anita

Gladys

Mildred

Victoria

Vivian

Audrey

public data is written in csv

(big\_query\_env) bhagvanarch@Bhagvans-MacBook-Air bigquery % ls

big\_query\_env public\_data\_g.json querying\_public\_data.py

big\_query\_example.py public\_usa\_names\_data.csv requirements.txt

(big\_query\_env) bhagvanarch@Bhagvans-MacBook-Air bigquery % vi public\_usa\_names\_data.csv

You can see the csv created and the csv has the names in public data of USA (limited by 10 count) in California.

Now let us look at Pyspark big data opensource framework.

PySpark

PySpark is based on Apache Spark which is an opensource Big data framework. Apache Spark is used by the developers for large data processing and developing machine learning applications. PySpark is based on RDD (Resilient Distributed Data sets). RDD helps in managing the resources in an Apache Spark Cluster. There are other products like HBase, Pig, Hive, etc., which are part of the Apache Spark Big data platform. Apache Spark can be used for building analytics application on streaming data.

PySpark helps in supporting python-based solutions over Apache spark. It has Py4j package. The key features in Pyspark are:

* Realtime Computing and Processing data
* Supporting Multiple Tech stacks
* Caching and I/O Consistency
* High performance data processing
* Handling RDD
* Reading and Writing Flat files (CSV etc.,)
* Data Analysis
* Sorting and Grouping Data
* Arithmetic Operations
* Data Aggregation
* Data PreProcessing
* Data Type conversion

The use cases which Pyspark can handle are:

* Data Mining – Finding patterns/trends/unknown data/prefrences
* Customer Segmentation
* Customer Identification From Transactions
* Data Clustering – Groups/Sets

First step is to download Apache Spark from <https://spark.apache.org/downloads.html>. You need to Java Runtime setup on your machine.

You can set the environment variables as shown below:

export SPARK\_HOME= /Users/bhagvanarch/kazdesk/spark-3.5.1-bin-hadoop3

export PATH= $PATH:$SPARK\_HOME/bin

export

PYTHONPATH=$SPARK\_HOME/python:$SPARK\_HOME/python/lib/py4j-0.10.4-src.zip:$PYTHONPATH

**spark\_example.py**

from pyspark import SparkContext

spark\_context = SparkContext("local", "Word Counting Application")

list\_words = spark\_context.parallelize (

["Richard Henderson",

"Jack Smith",

"Harry Smith",

"Gregory Smith",

"Anderson Smith",

"Vincent Patternson",

"Keith Baron",

"Derek Thomas Smith"]

)

word\_counts = list\_words.count()

print("Number of words in Resilient Distributed Data -> %i" % (word\_counts))

The code above can be executed using the command below:

python3 -m venv pyspark\_env

source pyspark\_env /bin/activate

pip3 install -r requirements.txt

python3 spark\_example.py

The output when the command above is executed, is shown below:

(pyspark\_env) bhagvanarch@Bhagvans-MacBook-Air pyspark % python3 spark\_example.py

24/03/03 22:51:23 WARN Utils: Your hostname, Bhagvans-MacBook-Air.local resolves to a loopback address: 127.0.0.1; using 192.168.1.4 instead (on interface en0)

24/03/03 22:51:23 WARN Utils: Set SPARK\_LOCAL\_IP if you need to bind to another address

Setting default log level to "WARN".

To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).

Number of words in Resilient Distributed Data -> 8

(pyspark\_env) bhagvanarch@Bhagvans-MacBook-Air pyspark %

In the above example, we have seen an example where number of words are counted in an array.

Let us see an example where we can see collection on RDD.

**spark\_example.py**

from pyspark import SparkContext

spark\_context = SparkContext("local", "Word Counting Application")

list\_words = spark\_context.parallelize (

["Richard Henderson",

"Jack Smith",

"Harry Smith",

"Gregory Smith",

"Anderson Smith",

"Vincent Patternson",

"Keith Baron",

"Derek Thomas Smith"]

)

word\_counts = list\_words.count()

print("Number of words in Resilient Distributed Data -> %i" % (word\_counts))

collection\_words = list\_words.collect()

print("Elements in RDD -> %s" % (collection\_words))

The code above can be executed using the command below:

python3 -m venv pyspark\_env

source pyspark\_env /bin/activate

pip3 install -r requirements.txt

python3 spark\_example.py

The output when the command above is executed, is shown below:

(pyspark\_env) bhagvanarch@Bhagvans-MacBook-Air pyspark % python3 spark\_example.py

24/03/03 23:09:55 WARN Utils: Your hostname, Bhagvans-MacBook-Air.local resolves to a loopback address: 127.0.0.1; using 192.168.1.4 instead (on interface en0)

24/03/03 23:09:55 WARN Utils: Set SPARK\_LOCAL\_IP if you need to bind to another address

Setting default log level to "WARN".

To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).

24/03/03 23:09:55 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable

Number of words in Resilient Distributed Data -> 8

Elements in RDD -> ['Richard Henderson', 'Jack Smith', 'Harry Smith', 'Gregory Smith', 'Anderson Smith', 'Vincent Patternson', 'Keith Baron', 'Derek Thomas Smith']

(pyspark\_env) bhagvanarch@Bhagvans-MacBook-Air pyspark %

In the above example, you can see the collection of the words in a RDD (Resilient Distributed Data).

Let us see an example where we filter data in a RDD.

**spark\_example.py**

from pyspark import SparkContext

spark\_context = SparkContext("local", "Word Counting Application")

list\_words = spark\_context.parallelize (

["Richard Henderson",

"Jack Smith",

"Harry Smith",

"Gregory Smith",

"Anderson Smith",

"Vincent Patternson",

"Keith Baron",

"Derek Thomas Smith"]

)

word\_counts = list\_words.count()

print("Number of words in Resilient Distributed Data -> %i" % (word\_counts))

collection\_words = list\_words.collect()

print("Elements in RDD -> %s" % (collection\_words))

filtered\_words = list\_words.filter(lambda y: 'Smith' in y)

filtered\_collection = filtered\_words.collect()

print("Fitered RDD -> %s" % (filtered\_collection))

The code above can be executed using the command below:

python3 -m venv pyspark\_env

source pyspark\_env /bin/activate

pip3 install -r requirements.txt

python3 spark\_example.py

The output when the command above is executed, is shown below:

(pyspark\_env) bhagvanarch@Bhagvans-MacBook-Air pyspark % python3 spark\_example.py

24/03/03 23:09:55 WARN Utils: Your hostname, Bhagvans-MacBook-Air.local resolves to a loopback address: 127.0.0.1; using 192.168.1.4 instead (on interface en0)

24/03/03 23:09:55 WARN Utils: Set SPARK\_LOCAL\_IP if you need to bind to another address

Setting default log level to "WARN".

To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).

24/03/03 23:09:55 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable

Number of words in Resilient Distributed Data -> 8

Elements in RDD -> ['Richard Henderson', 'Jack Smith', 'Harry Smith', 'Gregory Smith', 'Anderson Smith', 'Vincent Patternson', 'Keith Baron', 'Derek Thomas Smith']

Fitered RDD -> ['Jack Smith', 'Harry Smith', 'Gregory Smith', 'Anderson Smith', 'Derek Thomas Smith']

(pyspark\_env) bhagvanarch@Bhagvans-MacBook-Air pyspark %

In the above example, you can see the filtered collection of the words (smith filter) in a RDD (Resilient Distributed Data).

Let us see an example where we map data in a RDD.

**spark\_example.py**

from pyspark import SparkContext

spark\_context = SparkContext("local", "Word Counting Application")

list\_words = spark\_context.parallelize (

["Richard Henderson",

"Jack Smith",

"Harry Smith",

"Gregory Smith",

"Anderson Smith",

"Vincent Patternson",

"Keith Baron",

"Derek Thomas Smith"]

)

word\_counts = list\_words.count()

print("Number of words in Resilient Distributed Data -> %i" % (word\_counts))

collection\_words = list\_words.collect()

print("Elements in RDD -> %s" % (collection\_words))

filtered\_words = list\_words.filter(lambda y: 'Smith' in y)

filtered\_collection = filtered\_words.collect()

print("Fitered RDD -> %s" % (filtered\_collection))

index = 1

list\_words\_map = list\_words.map(lambda y: (y,index))

list\_word\_mapping = list\_words\_map.collect()

print("Key value pair -> %s" % (list\_word\_mapping))

The code above can be executed using the command below:

python3 -m venv pyspark\_env

source pyspark\_env /bin/activate

pip3 install -r requirements.txt

python3 spark\_example.py

The output when the command above is executed, is shown below:

(pyspark\_env) bhagvanarch@Bhagvans-MacBook-Air pyspark % python3 spark\_example.py

24/03/03 23:21:42 WARN Utils: Your hostname, Bhagvans-MacBook-Air.local resolves to a loopback address: 127.0.0.1; using 192.168.1.4 instead (on interface en0)

24/03/03 23:21:42 WARN Utils: Set SPARK\_LOCAL\_IP if you need to bind to another address

Setting default log level to "WARN".

To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).

24/03/03 23:21:42 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable

Number of words in Resilient Distributed Data -> 8

Elements in RDD -> ['Richard Henderson', 'Jack Smith', 'Harry Smith', 'Gregory Smith', 'Anderson Smith', 'Vincent Patternson', 'Keith Baron', 'Derek Thomas Smith']

Fitered RDD -> ['Jack Smith', 'Harry Smith', 'Gregory Smith', 'Anderson Smith', 'Derek Thomas Smith']

Key value pair -> [('Richard Henderson', 1), ('Jack Smith', 1), ('Harry Smith', 1), ('Gregory Smith', 1), ('Anderson Smith', 1), ('Vincent Patternson', 1), ('Keith Baron', 1), ('Derek Thomas Smith', 1)]

(pyspark\_env) bhagvanarch@Bhagvans-MacBook-Air pyspark %

In the above example, we have created a map from the data in a RDD.

Let us see an example where we SQL context to create a dataset with rows and columns.

**pyspark\_sql\_context.py**

from pyspark import SparkContext

from pyspark.sql import Row

from pyspark.sql import SQLContext

spark\_context = SparkContext("local", "SQL Context")

sqlContext = SQLContext(spark\_context)

data=[('Handbag',1),('Gold Chain',2),('Choco Bar',3),('Thai Curry Powder',4)]

rdd = spark\_context.parallelize(data)

product\_map=rdd.map(lambda y: Row(product=y[0], product\_id=int(y[1])))

data\_frame = sqlContext.createDataFrame(product\_map).collect()

print(data\_frame)

The code above can be executed using the command below:

python3 -m venv pyspark\_env

source pyspark\_env /bin/activate

pip3 install -r requirements.txt

python3 pyspark\_sql\_context.py

The output when the command above is executed, is shown below:

(pyspark\_env) bhagvanarch@Bhagvans-MacBook-Air pyspark % python3 pyspark\_sql\_context.py

24/03/03 23:01:14 WARN Utils: Your hostname, Bhagvans-MacBook-Air.local resolves to a loopback address: 127.0.0.1; using 192.168.1.4 instead (on interface en0)

24/03/03 23:01:14 WARN Utils: Set SPARK\_LOCAL\_IP if you need to bind to another address

Setting default log level to "WARN".

To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).

[Row(product='Handbag', product\_id=1), Row(product='Gold Chain', product\_id=2), Row(product='Choco Bar', product\_id=3), Row(product='Thai Curry Powder', product\_id=4)]

(pyspark\_env) bhagvanarch@Bhagvans-MacBook-Air pyspark %

In the above example, you can see the Sql context creating a data frame with rows and colums.

Let us see an example where we see an example of Accumulator

**accumulator.py**

from pyspark import SparkContext

spark\_context = SparkContext("local", "Pyspark Accumulator application")

snumdata = spark\_context.accumulator(10)

def g(y):

global snumdata

snumdata+=y

spark\_rdd = spark\_context.parallelize([70,80,90,10])

spark\_rdd.foreach(g)

final\_accum\_value = snumdata.value

print("Spark Accumulated value of Function g -> %i" % (final\_accum\_value))

The code above can be executed using the command below:

python3 -m venv pyspark\_env

source pyspark\_env /bin/activate

pip3 install -r requirements.txt

python3 accumulator\_example.py

The output when the command above is executed, is shown below:

(pyspark\_env) bhagvanarch@Bhagvans-MacBook-Air pyspark % python3 accumlator\_example.py

24/03/03 23:27:18 WARN Utils: Your hostname, Bhagvans-MacBook-Air.local resolves to a loopback address: 127.0.0.1; using 192.168.1.4 instead (on interface en0)

24/03/03 23:27:18 WARN Utils: Set SPARK\_LOCAL\_IP if you need to bind to another address

Setting default log level to "WARN".

To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).

Spark Accumulated value of Function g -> 260

(pyspark\_env) bhagvanarch@Bhagvans-MacBook-Air pyspark %

In the above example, you can see the accumulated final value of the g function after initialization.

Now let us look at Pydoop opensource big data framework.

Pydoop

Pydoop is based on Apache Hadoop. Hadoop has the following components in its platform.

* HDFS
* HIVE
* PIG
* YARN
* Spark
* Oozie
* Zookeeper
* Mahout

Pydoop has API to access Apache Hadoop’s Map Reduce and HDFS components. Pydoop helps in supporting python solutions using Apache Hadoop. You can download Hadoop from https://dlcdn.apache.org/hadoop/common/hadoop-3.3.6/hadoop-3.3.6.tar.gz

You need to set the PATH and JAVA\_HOME environment variables.

export PATH="/Users/bhagvanarch/kazdesk/hadoop-3.3.6/bin:${PATH}"

export JAVA\_HOME="/Library/Java/JavaVirtualMachines/jdk-1.8.jdk/Contents/Home"

pip3 install pydoop

**pydoop\_example.py**

import pydoop.mapreduce.api as api

import pydoop.mapreduce.pipes as pipes

class WCMapper(api.Mapper):

def \_\_init\_\_(self, context):

super(WordCountMapper, self).\_\_init\_\_(context)

def map(self, wc\_context):

list\_words = wc\_context.getInputValue().split()

for word in lisgt\_words:

wc\_context.emit(word, "1")

class WCReducer(api.Reducer):

def \_\_init\_\_(self, context):

super(WordCountReducer, self).\_\_init\_\_(context)

def reduce(self, wc\_context):

wc\_sum = 0

while wc\_context.nextValue():

wc\_sum += int(wc\_context.getInputValue())

print("wc\_sum",wc\_sum)

result = wc\_context.emit(wc\_context.getInputKey(), str(wc\_sum))

print("result",result)

def main():

wc\_factory = pipes.Factory(WCMapper, reducer\_class=WCReducer)

output = pipes.run\_task(wc\_factory,private\_encoding=False)

print("after running task", output)

if \_\_name\_\_ == '\_\_main\_\_':

main()

The code above can be executed using the command below:

pydoop submit --upload-file-to-cache pydoop\_example.py wc input output

The output when the command above is executed,The output folder will have the word count which is shown below :

17

Now let us look at Kafka – Opensource Big data framework.

Kafka

Apache Kafka is based on the topic and publish & subscribe messaging pattern. Messages are stored in the message queue or message topic. The kafka users can have them in replicated mode or partitioned mode. The messages received from the kafka client are persisted in the partitions. There are brokers like zookeeper which integrate well with Apache Kafka.

Let us an example where we have a KafkaProducer and Kafka Consumer. Producer creates the messages and sends it to the topic. Consumer processes the received message from topic.

First set the JAVA\_HOME and KAFKA\_HOME the environment variables.

export JAVA\_HOME="/Library/Java/JavaVirtualMachines/jdk-1.8.jdk/Contents/Home"

export KAFKA\_HOME=” /Users/bhagvanarch/kazdesk/kafka\_2.12-3.7.0”

pip3 install kafka-python

First let us see the code for Producer.

**kafka\_producer.py**

from time import sleep

from json import dumps

from kafka import KafkaProducer

def main():

kafka\_producer = KafkaProducer(bootstrap\_servers=['localhost:9092'],

value\_serializer=lambda x:

dumps(x).encode('utf-8'))

print("sending the message")

for number in range(300):

message\_data = {'Message id' : 'Message code '+str(number)}

kafka\_producer.send('kafka\_topic', value=message\_data)

sleep(1)

if \_\_name\_\_ == '\_\_main\_\_':

main()

Now let us look at the consumer code.

**kafka\_consumer\_example.py**

from time import sleep

from json import dumps

from kafka import KafkaConsumer

def main():

kafka\_consumer = KafkaConsumer(

'kafka\_topic',

bootstrap\_servers=['localhost:9092'],

auto\_offset\_reset='earliest',

enable\_auto\_commit=True,

group\_id='kafka-group')

for rec\_message in kafka\_consumer:

val\_message = rec\_message.value

print(' message',val\_message)

if \_\_name\_\_ == '\_\_main\_\_':

main()

The topic is created as kafka\_topic in kafka-group. Messages with id number starting from 1 to 300 are sent in the producer and consumer receives the messages and prints the message value.

The code above can be executed after starting zookeeper and kafka. First start the zookeeper using the command below.

$KAFKA\_HOME/bin/zookeeper-server-start.sh config/zookeeper.properties

The output for the zookeeper command is shown below:

bhagvanarch@Bhagvans-MacBook-Air kafka\_2.12-3.7.0 % ./bin/zookeeper-server-start.sh config/zookeeper.properties

[2024-03-04 22:49:49,922] INFO Reading configuration from: config/zookeeper.properties (org.apache.zookeeper.server.quorum.QuorumPeerConfig)

[2024-03-04 22:49:49,923] WARN config/zookeeper.properties is relative. Prepend ./ to indicate that you're sure! (org.apache.zookeeper.server.quorum.QuorumPeerConfig)

[2024-03-04 22:49:49,931] INFO clientPortAddress is 0.0.0.0:2181 (org.apache.zookeeper.server.quorum.QuorumPeerConfig)

[2024-03-04 22:49:49,931] INFO secureClientPort is not set (org.apache.zookeeper.server.quorum.QuorumPeerConfig)

[2024-03-04 22:49:49,931] INFO observerMasterPort is not set (org.apache.zookeeper.server.quorum.QuorumPeerConfig)

[2024-03-04 22:49:49,931] INFO metricsProvider.className is org.apache.zookeeper.metrics.impl.DefaultMetricsProvider (org.apache.zookeeper.server.quorum.QuorumPeerConfig)

[2024-03-04 22:49:49,933] INFO autopurge.snapRetainCount set to 3 (org.apache.zookeeper.server.DatadirCleanupManager)

[2024-03-04 22:49:49,933] INFO autopurge.purgeInterval set to 0 (org.apache.zookeeper.server.DatadirCleanupManager)

[2024-03-04 22:49:49,933] INFO Purge task is not scheduled. (org.apache.zookeeper.server.DatadirCleanupManager)

[2024-03-04 22:49:49,933] WARN Either no config or no quorum defined in config, running in standalone mode (org.apache.zookeeper.server.quorum.QuorumPeerMain)

[2024-03-04 22:49:49,933] INFO Log4j 1.2 jmx support not found; jmx disabled. (org.apache.zookeeper.jmx.ManagedUtil)

[2024-03-04 22:49:49,934] INFO Reading configuration from: config/zookeeper.properties (org.apache.zookeeper.server.quorum.QuorumPeerConfig)

[2024-03-04 22:49:49,934] WARN config/zookeeper.properties is relative. Prepend ./ to indicate that you're sure! (org.apache.zookeeper.server.quorum.QuorumPeerConfig)

[2024-03-04 22:49:49,934] INFO clientPortAddress is 0.0.0.0:2181 (org.apache.zookeeper.server.quorum.QuorumPeerConfig)

[2024-03-04 22:49:49,934] INFO secureClientPort is not set (org.apache.zookeeper.server.quorum.QuorumPeerConfig)

[2024-03-04 22:49:49,934] INFO observerMasterPort is not set (org.apache.zookeeper.server.quorum.QuorumPeerConfig)

Then start the kafka by using the command below:

$KAFKA\_HOME/bin/kafka-server-start.sh config/server.properties

The output for the kafka start command is shown below:

bhagvanarch@Bhagvans-MacBook-Air kafka\_2.12-3.7.0 % ./bin/kafka-server-start.sh config/server.properties

[2024-03-04 22:50:55,780] INFO Registered kafka:type=kafka.Log4jController MBean (kafka.utils.Log4jControllerRegistration$)

[2024-03-04 22:50:56,078] INFO Setting -D jdk.tls.rejectClientInitiatedRenegotiation=true to disable client-initiated TLS renegotiation (org.apache.zookeeper.common.X509Util)

[2024-03-04 22:50:56,129] INFO Registered signal handlers for TERM, INT, HUP (org.apache.kafka.common.utils.LoggingSignalHandler)

[2024-03-04 22:50:56,130] INFO starting (kafka.server.KafkaServer)

[2024-03-04 22:50:56,131] INFO Connecting to zookeeper on localhost:2181 (kafka.server.KafkaServer)

[2024-03-04 22:50:56,142] INFO [ZooKeeperClient Kafka server] Initializing a new session to localhost:2181. (kafka.zookeeper.ZooKeeperClient)

[2024-03-04 22:50:56,146] INFO Client environment:zookeeper.version=3.8.3-6ad6d364c7c0bcf0de452d54ebefa3058098ab56, built on 2023-10-05 10:34 UTC (org.apache.zookeeper.ZooKeeper)

[2024-03-04 22:50:56,146] INFO Client environment:host.name=localhost (org.apache.zookeeper.ZooKeeper)

[2024-03-04 22:50:56,146] INFO Client environment:java.version=1.8.0\_401 (org.apache.zookeeper.ZooKeeper)

[2024-03-04 22:50:56,146] INFO Client environment:java.vendor=Oracle Corporation (org.apache.zookeeper.ZooKeeper)

[2024-03-04 22:50:56,146] INFO Client environment:java.home=/Library/Internet Plug-Ins/JavaAppletPlugin.plugin/Contents/Home (org.apache.zookeeper.ZooKeeper)

Now the code can be executed by starting Producer and consumer in two different terminals.

The output for Producer is shown below:

(kafka\_env) bhagvanarch@Bhagvans-MacBook-Air kafka % python3 kafka\_producer.py

sending the message

The output for Consumer is shown below:

(kafka\_env) bhagvanarch@Bhagvans-MacBook-Air kafka % python3 kafka\_consumer\_example.py

message b'{"Message id": "Message code 0"}'

message b'{"Message id": "Message code 1"}'

message b'{"Message id": "Message code 2"}'

message b'{"Message id": "Message code 3"}'

………..

message b'{"Message id": "Message code 298"}'

message b'{"Message id": "Message code 299"}'

message b'{"Message id": "Message code 300"}'

As you can see in the example above, we can see the producer sending the messages and consumed by the consumer for processing the message.

Now let us look at Blockchain in python.

Blockchain using Python

BlockChain was first created by Satoshi Nakamoto in 2008. Blockchain is a chain of blocks. Blocks are linked to each other like in a linked list. The blocks are data records. Finance and other domains like Retail and Insurance use Blockchain to store transactions and other entities. The data stored in the blockchain is immutable, securable, persistent, and distributed across different nodes. Data stored in blocks will have the time stamp and an identifier. The hash of the data block will be used to link each other. Any tampering of data can be detected because of the secure hash.

Blockchain has distributed ledger, blocks, cryptographic hashing algorithm, consensus mechanism, mining algorithm, and the miner workers.

**blockchain.py**

import hashlib as hasher

import datetime as date

class FinBlockChain:

def \_\_init\_\_(self,Name):

self.name = Name

self.chain = []

def append(self,block):

self.chain.append(block)

def next\_block(self,Name,last\_block):

this\_index = last\_block.index + 1

this\_timestamp = date.datetime.now()

this\_data = Name + str(this\_index)

this\_hash = last\_block.hash

return FinBlock(Name,this\_index, this\_timestamp, this\_data, this\_hash)

class FinBlock:

def \_\_init\_\_(self,Name,index, timestamp, data, previous\_hash):

self.name = Name

self.index = index

self.timestamp = timestamp

self.data = data

self.previous\_hash = previous\_hash

self.hash = self.hash\_block()

def hash\_block(self):

sha = hasher.sha256()

sha.update(str(self.index).encode('utf-8') + str(self.timestamp).encode('utf-8') + str(self.data).encode('utf-8') + str(self.previous\_hash).encode('utf-8'))

return sha.hexdigest()

fin\_blockchain = FinBlockChain("Fintech BlockChain")

fin\_block = FinBlock("Block1",0, date.datetime.now(), "Block1".encode('utf-8'), "0".encode('utf-8'));

print("Blockchain name is ",fin\_blockchain.name);

fin\_previous\_block = fin\_block

fin\_num\_of\_blocks = 30

for i in range(0, fin\_num\_of\_blocks):

fin\_block\_to\_add = fin\_blockchain.next\_block("num"+str(i),fin\_previous\_block)

fin\_blockchain.append(fin\_block\_to\_add)

fin\_previous\_block = fin\_block\_to\_add

print("The Block #{} is added to the blockchain!".format(fin\_block\_to\_add.index))

print("The Hash: {}\n".format(fin\_block\_to\_add.hash))

The code above can be executed using the command below:

python3 blockchain.py

The output when the command above is executed, is shown below:

bhagvanarch@Bhagvans-MacBook-Air blockchain % python3 blockchain.py

Blockchain name is Fintech BlockChain

The Block #1 is added to the blockchain!

The Hash: 66b3263c59489846c648bf40ce2ff66afb2c43b8f3929c67d073c8b22cf71c86

The Block #2 is added to the blockchain!

The Hash: b399d2f97a6742f96d4e4012ace53a3d1d9219a74050fb789df24f51c91b5ce9

The Block #3 is added to the blockchain!

The Hash: 72eebb165bd42f94a9cdbb5fc401fb763cc646a61a89748104592d861edc7457

……….

The Block #28 is added to the blockchain!

The Hash: 196a64b9a3ce20ccc7301d21b24e58bc0b9a6131b3224d4a05bd53a3eeeb1843

The Block #29 is added to the blockchain!

The Hash: 4fcaccac90d4d61689d731934c3f2ca5cf0d0a1cf3beafb9e8bed643e85b3a89

The Block #30 is added to the blockchain!

The Hash: 78467bc1b2cdd34290a1a033e71e4b072fc7254f97c22c9f39fad3c1c8d2677b

bhagvanarch@Bhagvans-MacBook-Air blockchain %

In the above example, we have seen the block chain created and how the blocks are added to blockchain and the hash of each block.

Conclusion

We have seen different big data opensource frameworks like BigQuery, Pyspark, Pydoop, and Kafka . We have seen examples from these frameworks. We saw Blockchain and how the block chain can be written in python. In the Next chapter, Next generation fintech python apps will be presented and the latest techniques & algorithms will be discussed. Readers will be presented with examples from the real world how web 3.0, and AI techniques can be applied.

Points to remember

* We looked at the different big data python opensource packages.
* We looked at Big Query framework and how it is used for querying data on Google cloud.
* We have presented how Pyspark can be used to build big data solutions in python.
* We also looked at Pydoop and Kafka and how the big data solutions can be built.
* You have seen how to build a simple blockchain in python.

Multiple choice questions

1. Which Big data opensource package can use publish-subscribe mechanism?
2. Big Query
3. Pyspark
4. Pydoop
5. Kafka
6. Which cloud access and service account is required for Bigquery ?
7. AWS
8. Google Cloud
9. IBM Bluemix
10. Oracle
11. Which data structure is similar to a Block chain?
12. Linked List
13. Set
14. BinaryTree
15. HashMap

Answers

1. d
2. b
3. a