EEL 6761: Cloud Computing

Project

Spring 2022

Deploy an API service for a distributed data processing pipeline on CloudLab

Anmol Lingan Gouda Patil
UFID: 1967 3150

A. The 10 windows of Spark outputs as described in part g of task 1.
Ans:

```
[anmollp@node-1:~$ curl -X GET http://128.105.144.46:10001/window?id=1

-------------------------------------------------------------------------------------
Window #  |  Plant Records    |  Animal Records   |  Fungi Records |Unique Species Records
-------------------------------------------------------------------------------------
1         |  3561             |  108              |  307           |386
-------------------------------------------------------------------------------------
[anmollp@node-1:~$ curl -X GET http://128.105.144.46:10001/window?id=2

-------------------------------------------------------------------------------------
Window #  |  Plant Records    |  Animal Records   |  Fungi Records |Unique Species Records
-------------------------------------------------------------------------------------
2         |  0                |  0                |  0             |0
-------------------------------------------------------------------------------------
[anmollp@node-1:~$ curl -X GET http://128.105.144.46:10001/window?id=3

-------------------------------------------------------------------------------------
Window #  |  Plant Records    |  Animal Records   |  Fungi Records |Unique Species Records
-------------------------------------------------------------------------------------
3         |  12965            |  3624             |  0             |5272
-------------------------------------------------------------------------------------
[anmollp@node-1:~$ curl -X GET http://128.105.144.46:10001/window?id=4

-------------------------------------------------------------------------------------
Window #  |  Plant Records    |  Animal Records   |  Fungi Records |Unique Species Records
-------------------------------------------------------------------------------------
4         |  49628            |  361              |  0             |11608
-------------------------------------------------------------------------------------
[anmollp@node-1:~$ curl -X GET http://128.105.144.46:10001/window?id=5

-------------------------------------------------------------------------------------
Window #  |  Plant Records    |  Animal Records   |  Fungi Records |Unique Species Records
-------------------------------------------------------------------------------------
5         |  0                |  25825            |  1317          |12821
-------------------------------------------------------------------------------------
```

```
[anmollp@node-1:~$ curl -X GET http://128.105.144.46:10001/window?id=6

-------------------------------------------------------------------------------------
Window #  |  Plant Records    |  Animal Records   |  Fungi Records |Unique Species Records
-------------------------------------------------------------------------------------
6         |  0                |  0                |  0             |12673
-------------------------------------------------------------------------------------
[anmollp@node-1:~$ curl -X GET http://128.105.144.46:10001/window?id=7

-------------------------------------------------------------------------------------
Window #  |  Plant Records    |  Animal Records   |  Fungi Records |Unique Species Records
-------------------------------------------------------------------------------------
7         |  0                |  0                |  0             |10651
-------------------------------------------------------------------------------------
[anmollp@node-1:~$ curl -X GET http://128.105.144.46:10001/window?id=8

-------------------------------------------------------------------------------------
Window #  |  Plant Records    |  Animal Records   |  Fungi Records |Unique Species Records
-------------------------------------------------------------------------------------
8         |  0                |  0                |  0             |8732
-------------------------------------------------------------------------------------
[anmollp@node-1:~$ curl -X GET http://128.105.144.46:10001/window?id=9

-------------------------------------------------------------------------------------
Window #  |  Plant Records    |  Animal Records   |  Fungi Records |Unique Species Records
-------------------------------------------------------------------------------------
9         |  0                |  0                |  0             |12296
-------------------------------------------------------------------------------------
[anmollp@node-1:~$ curl -X GET http://128.105.144.46:10001/window?id=10

-------------------------------------------------------------------------------------
Window #  |  Plant Records    |  Animal Records   |  Fungi Records |Unique Species Records
-------------------------------------------------------------------------------------
10        |  0                |  0                |  0             |12169
-------------------------------------------------------------------------------------
```

Their response codes:

```
[anmollp@node-0:~$ python3 webserver.py 10001
http://0.0.0.0:10001/
128.105.144.45:46912 - - [19/Apr/2022 15:15:00] "HTTP/1.1 GET /window" - 200 OK
128.105.144.45:47304 - - [19/Apr/2022 15:15:12] "HTTP/1.1 GET /window" - 200 OK
128.105.144.45:47542 - - [19/Apr/2022 15:15:18] "HTTP/1.1 GET /window" - 200 OK
128.105.144.45:47770 - - [19/Apr/2022 15:15:26] "HTTP/1.1 GET /window" - 200 OK
128.105.144.45:48072 - - [19/Apr/2022 15:15:35] "HTTP/1.1 GET /window" - 200 OK
128.105.144.45:48320 - - [19/Apr/2022 15:15:42] "HTTP/1.1 GET /window" - 200 OK
128.105.144.45:48582 - - [19/Apr/2022 15:15:50] "HTTP/1.1 GET /window" - 200 OK
128.105.144.45:48814 - - [19/Apr/2022 15:15:57] "HTTP/1.1 GET /window" - 200 OK
128.105.144.45:49056 - - [19/Apr/2022 15:16:05] "HTTP/1.1 GET /window" - 200 OK
128.105.144.45:49314 - - [19/Apr/2022 15:16:13] "HTTP/1.1 GET /window" - 200 OK
128.105.144.45:49556 - - [19/Apr/2022 15:16:20] "HTTP/1.1 GET /window" - 200 OK
```

Description:

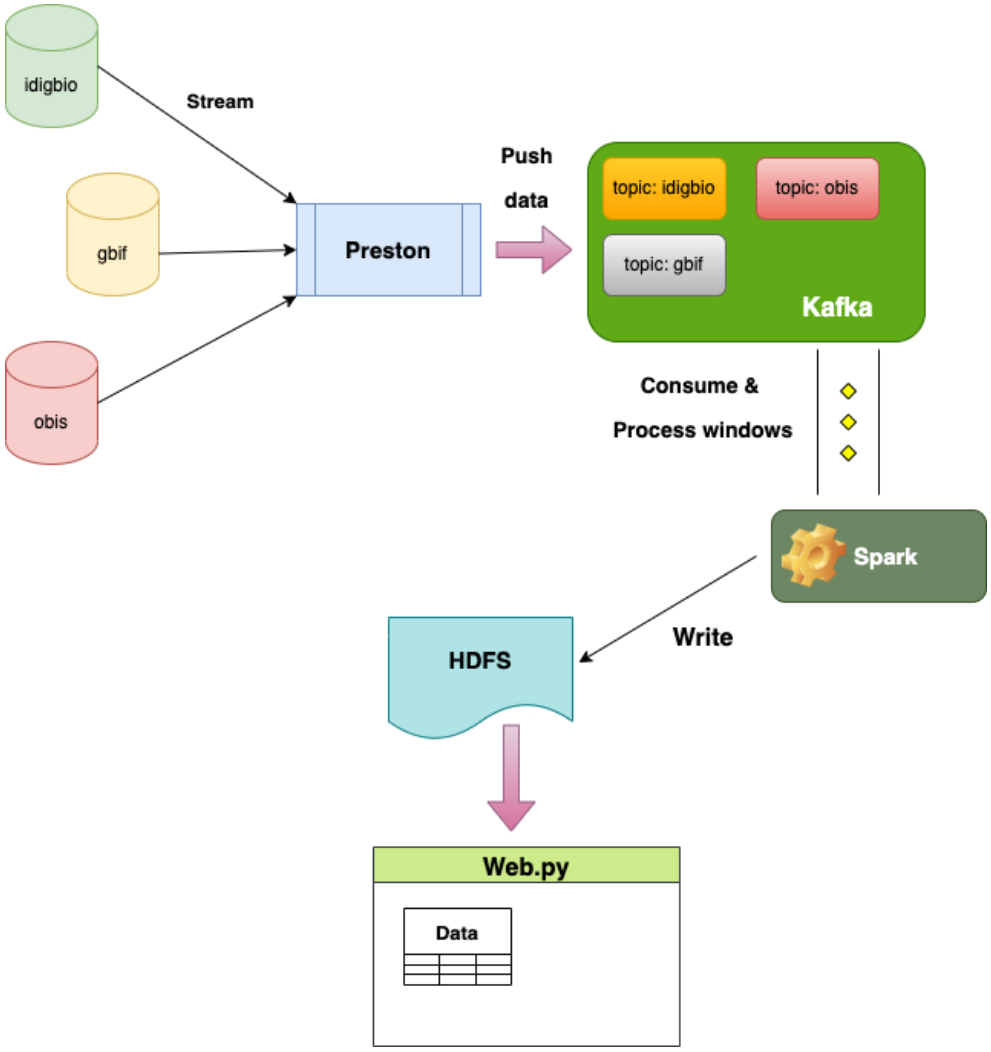
Open source technologies used and their versions:

Java	**11**
Kafka	**3.1.0**
Spark	**3.2.1**
Hadoop	**3.2.3**
Preston	**0.3.5**
Python	**3.9.0**

- Firstly a **cluster of 3 nodes** has to be set up with an Ubuntu operating system on cloud lab.
- Since the project is **JVM** heavy, **Java 11** needs to be installed.
- To exploit parallel processing we need to install **Kafka** and **Spark** appropriate versions.
- For the persistence layer we will be using **HDFS** and it would require us using **hadoop**.
- **Python** is the primary language used to develop the spark code.
- Libraries like **kafka-python**, **web.py** need to be installed.
- When configuring kafka, make sure the **broker id** is different for different nodes.
- Set **max.retention.minutes = 2** and set the **zookeeper address** correctly in server.properties file for kafka. Also need to carefully set the **replication factor**, **number of partitions** and **number of in sync replicas** for topics.
- Setup **systemd services** for zookeeper and kafka brokers to startup automatically on system boot.
- For spark, we need to create our own **spark-env.sh** file that includes spark master details and details for pyspark.

- Also spark needs to know the **master** and **slave ip addresses** inorder to run distributedly.
- Our spark cluster runs as a **standalone** server.
- Spark submit is used to start the spark processing of streaming records from kafka in client mode with **spark structured streaming jars** passed as arguments.
- While configuring hdfs, we need to configure the following files correctly namely **core-site.xml**, **hdfs-site.xml**, **yarn-site.xml**, **hadoop-env.sh**, **masters** and **slaves**.
- For all of the distributed processes to run properly a **passwordless ssh** has to be set up amongst these nodes.
- Therefore the master node's **public key** has to be present in all the slave server node's **authorised_keys** file under the **.ssh/** folder.
- Finally when all these steps are taken care of we can spark submit our consumer job and subsequently we can start the producer processes that also takes care of starting **preston streamer**.
- As I am using a **trigger process interval** of **2 minutes** , windows of data pertaining to kingdom, species and source related records are processed by spark and stored on hdfs in a distributed fashion.
- To control the pipeline and to visualize the data processed I use a **web server** developed in **web.py** that exposes APIs to achieve the same.

High Level System design:



B. List of scripts and their description:

- **consumer.py:** contains spark code that consumes data from all three topics using spark structured streaming creating windows of required data every 2 minutes.
- **gbif_producer.py:** python code that starts gbif stream and streams it to a producer that pushes data to the kafka broker assigned to it to a topic called gbif, takes broker ip as input.
- **obis_producer.py:** python code that starts an obis stream and streams it to a producer that pushes data to the kafka broker assigned to it to a topic called gbif, takes broker ip as input.

- **idigbio_producer.py**: python code that starts an idigbio stream and streams it to a producer that pushes data to the kafka broker assigned to it to a topic called idigbio, takes broker ip as input.
- **webserver.py:** this code deploys a web server that exposes APIs to read processed data residing on HDFS.
- **start-script.sh:** a shell script thats starts zookeeper, kafka, hdfs and spark master and slaves.
- **stop-script.sh:** shell script that stops zookeeper, kafka, hdfs and spark.
- **start-slave-script.sh:** shell script that starts kafka broker on a slave.
- **stop-slave-script.sh:** shell script that stops kafka broker running on a slave.
- **stop-stream.sh:** stops preston streams streaming data to kafka.
- **zookeeper.service:** as soon as the master node is powered and network is available, start the zookeeper using this systemd service profile.
- **kafka.service:** as soon as a node is powered and network is available, start the kafka broker using this systemd service profile.
  Supported APIs:
  - **curl -X GET http://128.105.144.46:10001/addSource?url=<url>**
  - **curl -X GET http://128.105.144.46:10001/addSource?url=<url>**
  - **curl -X GET http://128.105.144.46:10001/addSource?url=<url>**
  - **curl -X GET http://128.105.144.46:10001/listSources**
  - **curl -X GET http://128.105.144.46:10001/count?by=totalSpecies**
  - **curl -X GET http://128.105.144.46:10001/count?by=kingdom**
  - **curl -X GET http://128.105.144.46:10001/count?by=source**
  - **curl -X GET http://128.105.144.46:10001/window?id=<int>**

C. Source code of the above files in order:

1. **consumer.py**

```
from pyspark import SparkContext, SparkConf, TaskContext
from pyspark.sql import SparkSession, SQLContext
from pyspark.sql.types import StructType, StructField, StringType, IntegerType
from pyspark.sql.functions import col, from_json, udf, window, approx_count_distinct

spark =
SparkSession.builder.master('spark://c240g5-110107.wisc.cloudlab.us:7078').appName('
dataPipeline').getOrCreate()

bootstrap_servers = "128.105.144.46:9092,128.105.144.51:9092,128.105.144.45:9092"

df = spark.readStream \
    .format("kafka") \
    .option("kafka.bootstrap.servers", bootstrap_servers) \
    .option("subscribe", "idigbio, gbif, obis") \
    .option("startingOffsets", "earliest") \
```

```
    .option("failOnDataLoss", "false") \
    .load()


def save_kingdom_batch(df, epoch_id):
    df.persist()
    df.write.format("csv").save("hdfs://128.105.144.46:9000/kingdom/{}".format(epoch_id
+ 1))
    df.unpersist()


def save_species_batch(df, epoch_id):
    df.persist()
    df.write.format("csv").save("hdfs://128.105.144.46:9000/species/{}".format(epoch_id +
1))
    df.unpersist()


def get_batch_num():
    return 1 + int(TaskContext.get().getLocalProperty("streaming.sql.batchId"))


batch_id_udf = udf(get_batch_num)

spark.udf.register("get_batch_num", get_batch_num, IntegerType())

cast_df = df.selectExpr("CAST(key AS STRING) as key", "CAST(value AS STRING) as
value")
schema = StructType([
    StructField("http://rs.tdwg.org/dwc/terms/kingdom", StringType(), True),
    StructField("http://rs.tdwg.org/dwc/terms/scientificName", StringType(), True)
])
required_df = cast_df.select("key", "value").withColumn("value", from_json(col("value"),
schema))
required_df.createOrReplaceTempView("table")
query_df = spark.sql(
    "SELECT key,"
    " value.`http://rs.tdwg.org/dwc/terms/kingdom` AS kingdom,"
    " value.`http://rs.tdwg.org/dwc/terms/scientificName` AS species,"
    " now() as event_timestamp from table")

query_df.createOrReplaceTempView("table1")
```

```
kingdom_df = query_df.withWatermark("event_timestamp", "2
minutes").groupBy(window("event_timestamp", "2 minutes"),
                                                    "key",
"kingdom").count().select("key","kingdom","count")
species_df = query_df.withWatermark("event_timestamp", "2
minutes").groupBy("key",window("event_timestamp", "2
minutes")).agg(approx_count_distinct("species").alias("unique_species_count")).select("
key", "unique_species_count")
query1 =
kingdom_df.writeStream.outputMode("update").foreachBatch(save_kingdom_batch).trigg
er(
   processingTime="2 minutes").start()
query2 =
species_df.writeStream.outputMode("update").foreachBatch(save_species_batch).trigge
r(
   processingTime="2 minutes").start()

query1.awaitTermination()
query2.awaitTermination()
```

2. **gbif_producer.py**

```
import argparse
from kafka import KafkaProducer
import json
import subprocess
import shlex

parser = argparse.ArgumentParser(description='Start a kafka producer')
parser.add_argument("--bootstrap-servers", nargs='+', metavar='0.0.0.0', type=str,
                help="Space separated broker ip addresses")
args = parser.parse_args()
args_dict = vars(args)
brokers = list(map(lambda x: x + ":9092", args_dict.get('bootstrap_servers', 'localhost')))

producer = KafkaProducer(key_serializer=str.encode,
                    bootstrap_servers=",".join(brokers))

command1 = "preston track --seed https://gbif.org"
command2 = "./clean-cache"
command3 = "preston json-stream"

# invoke process
process1 = subprocess.Popen(shlex.split(command1), shell=False,
stdout=subprocess.PIPE, stderr=subprocess.DEVNULL)
```

```
process2 = subprocess.Popen(shlex.split(command2), shell=False,
stdin=process1.stdout, stdout=subprocess.PIPE)
process3 = subprocess.Popen(shlex.split(command3), shell=False,
stdin=process2.stdout, stdout=subprocess.PIPE, stderr=subprocess.DEVNULL)

#process1.stdout.close()
#process2.stdout.close()

while True:
   output = process3.stdout.readline()

   if process3.poll() is not None:
      break
   if output:
      producer.send('gbif', key='https://gbif.org', value=output.strip())
rc = process3.poll()
print(rc)
```

3. **obis_producer.py**

```
import argparse
from kafka import KafkaProducer
import json
import subprocess
import shlex

parser = argparse.ArgumentParser(description='Start a kafka producer')
parser.add_argument("--bootstrap-servers", nargs='+', metavar='0.0.0.0', type=str,
                help="Space separated broker ip addresses")
args = parser.parse_args()
args_dict = vars(args)
brokers = list(map(lambda x: x + ":9092", args_dict.get('bootstrap_servers', 'localhost')))

producer = KafkaProducer(key_serializer=str.encode,
                    bootstrap_servers=",".join(brokers))

command1 = "preston track --seed https://obis.org"
command2 = "./clean-cache"
command3 = "preston json-stream"

# invoke process
process1 = subprocess.Popen(shlex.split(command1), shell=False,
stdout=subprocess.PIPE, stderr=subprocess.DEVNULL)
process2 = subprocess.Popen(shlex.split(command2), shell=False,
stdin=process1.stdout, stdout=subprocess.PIPE)
```

```
process3 = subprocess.Popen(shlex.split(command3), shell=False,
stdin=process2.stdout, stdout=subprocess.PIPE, stderr=subprocess.DEVNULL)

#process1.stdout.close()
#process2.stdout.close()

while True:
   output = process3.stdout.readline()
   if process3.poll() is not None:
      break
   if output:
      producer.send('obis', key='https://obis.org', value=output.strip())
rc = process3.poll()
print(rc)
```

4. **idigbio_producer.py**

```
import argparse
from kafka import KafkaProducer
import json
import subprocess
import shlex

parser = argparse.ArgumentParser(description='Start a kafka producer')
parser.add_argument("--bootstrap-servers", nargs='+', metavar='0.0.0.0', type=str,
                help="Space separated broker ip addresses")
args = parser.parse_args()
args_dict = vars(args)
brokers = list(map(lambda x: x + ":9092", args_dict.get('bootstrap_servers', 'localhost')))

producer = KafkaProducer(key_serializer=str.encode,
                    bootstrap_servers=",".join(brokers))

command1 = "preston track --seed https://idigbio.org"
command2 = "./clean-cache"
command3 = "preston json-stream"

# invoke process
process1 = subprocess.Popen(shlex.split(command1), shell=False,
stdout=subprocess.PIPE, stderr=subprocess.DEVNULL)
process2 = subprocess.Popen(shlex.split(command2), shell=False,
stdin=process1.stdout, stdout=subprocess.PIPE)
process3 = subprocess.Popen(shlex.split(command3), shell=False,
stdin=process2.stdout, stdout=subprocess.PIPE, stderr=subprocess.DEVNULL)
```

```
#process1.stdout.close()
#process2.stdout.close()

while True:
    output = process3.stdout.readline()

    if process3.poll() is not None:
        break
    if output:
        producer.send('idigbio', key='https://idigbio.org', value=output.strip())
rc = process3.poll()
print(rc)
```

5. **webserver.py**

```
import web
import subprocess
from subprocess import Popen, PIPE
import shlex

urls = (
    '/count?(.+)', 'Count',
    '/addSource?(.+)', 'AddSource',
    '/listSources', 'ListSources',
    '/window?(.+)', 'Window'
)

def notfound():
    return web.notfound("Sorry, the page you were looking for was not found.")

active_sources = []
required_list = ["Fungi", "Animalia", "Plantae"]

class Count(object):
    def GET(self, by):
        result = []
        data = web.input()
        by = data.by
        data_dictionary = {}
        text = ""
        if by == "kingdom":
            cat = Popen(["hadoop", "fs", "-cat", "/kingdom/*/*"], stdout=PIPE)
            for line in cat.stdout:
                _, kingdom, records = line.decode().strip("\n").split(",")
                if kingdom in required_list:
```

```
                data_dictionary[kingdom] = int(records) + data_dictionary.get(kingdom, 0)
        elif by == "totalSpecies":
            cat = Popen(["hadoop", "fs", "-cat", "/species/*/*"], stdout=PIPE)
            key = 'Species'
            for line in cat.stdout:
                _, records = line.decode().strip("\n").split(",")
                data_dictionary[key] = int(records) + data_dictionary.get(key, 0)
        elif by == "source":
            cat = Popen(["hadoop", "fs", "-cat", "/kingdom/*/*"], stdout=PIPE)
            for line in cat.stdout:
                x = line.decode().strip("\n").split(",")
                if len(x) == 2:
                    source, records = x[0], x[1]
                elif len(x) == 3:
                    source, records = x[0], x[2]
                data_dictionary[source] = int(records) + data_dictionary.get(source, 0)
        else:
            return web.badrequest(message="Unrecognized parameter '{}'.".format(by))
        result.append(data_dictionary)
        for key, value in data_dictionary.items():
            text += "{:<20} {:<15}\n".format(key, value)
        return text
#       return result

class AddSource:
    def __init__(self):
        self.url_list = ['https://idigbio.org', 'https://gbif.org', 'https://obis.org']
        self.command_list = ['python3 idigbio_producer.py', 'python3 gbif_producer.py',
'python3 obis_producer.py']
        self.bootstrap_servers = ["128.105.144.46:9092", "128.105.144.45:9092",
"128.105.144.51:9092"]

    def GET(self, url):
        data = web.input()
        url = data.url
        if url in self.url_list:
            command = None
            if url == self.url_list[0]:
                command = self.command_list[0] + " --bootstrap-servers " +
self.bootstrap_servers[0]
                process = Popen(shlex.split(command), shell=False,
stderr=subprocess.DEVNULL)
                active_sources.append(self.url_list[0])
                return web.ok("Source {} added".format(url))
```

```
            elif url == self.url_list[1]:
                command = self.command_list[1] + " --bootstrap-servers " +
self.bootstrap_servers[1]
                process = Popen(shlex.split(command), shell=False,
stderr=subprocess.DEVNULL)
                active_sources.append(self.url_list[1])
                return web.ok("Source {} added".format(url))
            elif url == self.url_list[2]:
                command = self.command_list[2] + " --bootstrap-servers " +
self.bootstrap_servers[2]
                process = Popen(shlex.split(command), shell=False,
stderr=subprocess.DEVNULL)
                active_sources.append(self.url_list[2])
                return web.ok("Source {} added".format(url))
        else:
            return web.badrequest("Unrecognized url")

class ListSources:
    def GET(self):
        text = ""
        for source in active_sources:
            text += "{:<15}\n".format(source)
        return text

class Window:

    def GET(self, id):
        data = web.input()
        window_num = data.id
        data_dictionary = {"Fungi": 0, "Animalia": 0, "Plantae": 0, "Unique Species": 0}
        cat = Popen(["hadoop", "fs", "-cat", "/kingdom/{}/*".format(window_num)],
stdout=PIPE)
        for line in cat.stdout:
            _, kingdom, records = line.decode().strip("\n").split(",")
            if kingdom in required_list:
                data_dictionary[kingdom] = int(records) + data_dictionary.get(kingdom, 0)
        cat = Popen(["hadoop", "fs", "-cat", "/species/{}/*".format(window_num)],
stdout=PIPE)
        for line in cat.stdout:
            _, records = line.decode().strip("\n").split(",")
            data_dictionary["Unique Species"] = int(records) + data_dictionary.get("Unique
Species", 0)
        table = """
--------------------------------------------------------------------------------
```

```
{:<10}|  {:<17}|  {:<17}|  {:<14}|{:<23}
-------------------------------------------------------------------------------------------
{:<10}|  {:<17}|  {:<17}|  {:<14}|{:<23}
-------------------------------------------------------------------------------------------
""".format("Window #", "Plant Records", "Animal Records", "Fungi Records", "Unique
Species Records", window_num, data_dictionary["Plantae"], data_dictionary["Animalia"],
          data_dictionary["Fungi"], data_dictionary["Unique Species"])
       return table

if __name__ == "__main__":
   app = web.application(urls, globals())
   app.notfound = notfound
   app.run()
```

6. **start-script.sh**

```
#!/bin/bash
sudo systemctl start zookeeper.service
sudo systemctl start kafka.service
start-all.sh
$SPARK_HOME/sbin/start-all.sh
```

7. **stop-script.sh**

```
#!/bin/bash
sudo systemctl stop zookeeper.service
sudo systemctl stop kafka.service
stop-all.sh
$SPARK_HOME/sbin/stop-all.sh
```

8. **start-script-slave.sh**

```
#!/bin/bash
sudo systemctl start kafka.service
```

9. **stop-script-slave.sh**

```
#!/bin/bash
sudo systemctl stop kafka.service
```

10. **stop-stream.sh**

```
ps -ef | grep 'preston' | grep -v grep | awk '{print $2}' | xargs -r kill -9
```

11. **clean-cache**

```
#!/bin/sh
mkdir -p data
cd data

while IFS='$\n' read -r line; do
# Ignore unnecessary information
```

```
        if [ $(echo $line | cut -d " " -f2) = "<http://purl.org/pav/hasVersion>" ]
        then
        # Keep the newest 5 files, delete everything else
                rm -rf $(ls -1t */*/* | tail -n +11)

        # Echo stdin to stdout
                printf "%s\n" "$line"
        fi
done
```

12. zookeeper.service

```
[Unit]
Description=Zookeeper Daemon
Documentation=http://zookeeper.apache.org
Requires=network.target
After=network.target

[Service]
WorkingDirectory=/opt/kafka
User=root
Group=root
ExecStart=/opt/kafka/bin/zookeeper-server-start.sh
/opt/kafka/config/zookeeper.properties
ExecStop=/opt/kafka/bin/zookeeper-server-stop.sh
/opt/kafka/config/zookeeper.properties
TimeoutSec=30

[Install]
WantedBy=default.target
```

13. kafka.service

```
[Unit]
Description=Apache Kafka Server (broker)
Documentation=http://kafka.apache.org
Requires=network.target remote-fs.target
After=network.target remote-fs.target zookeeper.service

[Service]
WorkingDirectory=/opt/kafka
User=root
Group=root
ExecStart=/opt/kafka/bin/kafka-server-start.sh /opt/kafka/config/server.properties
ExecStop=/opt/kafka/bin/kafka-server-stop.sh
TimeoutSec=30
```

```
[Install]
WantedBy=multi-user.target
```