Git
https://github.com/git-for-windows/git/releases/download/v2.41.0.windows.2/Git-2.41.0.2-64-bit.exe

AWS CLI
https://awscli.amazonaws.com/AWSCLIV2.msi

MAC AWS CLI
https://awscli.amazonaws.com/AWSCLIV2.pkg

Winutils Setup

==

Goto any drive C or D drive
Create a folder D:/hadoop/bin
Paste the downloaded winutils file in **bin** folder

=================

AWS CLI MAC CURL COMMANDS
Works with this as well.

```
$ curl "https://s3.amazonaws.com/aws-cli/awscli-bundle.zip" -o "awscli-bundle.zip"
unzip awscli-bundle.zip
sudo ./awscli-bundle/install -i /usr/local/aws -b /usr/local/bin/aws
```

================

AWS Configure

```
aws configure
AWS Access Key ID = AKIAS3H27Y6U3W4RLXNI
AWS Secret Access Key = 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
Default region name =  ap-south-1
Default output format = json
aws s3 ls
```

=================

Windows cmd Folks

```
notepad.exe zeyofile
dir
aws s3 cp zeyofile s3://36buck/URNAMEdir/
aws s3 ls s3://36buck/URNAMEdir/
```

================

Linux/Mac

```
touch zeyofile
ls
aws s3 cp zeyofile s3://36buck/URNAMEdir/
aws s3 ls s3://36buck/URNAMEdir/
```

Find the second highest salary Reference

https://stackoverflow.com/questions/58490229/second-highest-value-by-department-using-apache-spark-dataframe

====================

Windows cmd

```
mkdir localdir
cd localdir
notepad.exe file1
cd ..
aws s3 sync localdir/  s3://buck36/36dir/URNAMEdir/
```

Windows Users
Step 1 -- install aws cli and Git bash windows users
Step 2 -- Open windows cmd / git bash

```
aws configure
```

AWS Access Key ID = AKIAS3H27Y6U3W4RLXNI
AWS Secret Access Key = 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
Default region name = ap-south-1
Default output format = json

```
aws s3 ls
```

step 3 --
Git bash

```
mkdir localdir
echo zeyo1> localdir/file1
echo zeyo2> localdir/file2
aws s3 sync localdir/  s3://buck36/36dir/URNAMEdir/
aws s3 ls s3://buck36/36dir/URNAMEdir/
```

======================================

Project Passage

This is __ My total years of exp and Relevant exp ..

I got chance to work on different Big Data Stack.Like(Hadoop,hdfs,hive,spark,sqoop,AWS). Recently i started migrating project AWS.

Before I used to in the data ingestion team where we used RDBMS as a source and we sqoop the data to HDFS and we processed it using hive and write to HDFS as avro. We use avro because of schema evolution. We have multiple RDBMS table in which we run multiple Sqoop Jobs and do the processing.

Later for example (1 year ago). I started working with Data application team where I have spark rigorously. In the data application. We have so many WEB apis coming with complex json json with different data models we almost run 7 spark jobs for different use cases like Customer data cleansing, Prediction Model spark jobs with currency conversions and few of the spark jobs do joins with AVRO data which generated during data ingestion and we write data to different HDFS directories as per the requirement also with complex nested data generation.

My business uses impala to do analytics on processed data.

In the recent times we started migrating the jobs to AWS. with services s3 , EMR for spark jobs,Athena for Business analytics and ec2 for scheduling.

We have a done POC on EMR step executions run those spark jobs using EMR command Runner.

Interview -- AWS

We have data sources from Webapi powered with SSL and AWS S3 along with snowflake. We run our jobs in AWS EMR. We consume the data from all the sources and do the necessary processing and finally write the data to 2 different destination -- s3,snowflake

We perform all the necessary DSL operations in spark. We almost run 10-11 steps in AWS EMR UNDER STEP Execution.

Deployment

We create our own cluster in the daily basis and do the development/implementation.and terminate the cluster by end of the day. And copy the necessary copy intermediate to s3 for next day use.

Once the implementation - we commit the code to GIT .

For the production deployment - We run the Jenkins Pipeline created by Devops Team which would enable the Jar in the production s3 bucket.

However we will create a step execution command runner Automation emr script and test in the dev environment and take it to the production.

We schedule the Job using Nifi Running in EC2 machine. Processor Name (Execute processor)

Putty download Link
https://the.earth.li/~sgtatham/putty/latest/w64/putty.exe

====================
Cloudera Folks

Task 1 ---
open pyspark shell --- (type pyspark and enter)

```
value='zeyobron'
sc.parallelize([value]).foreach(print)
```

Lab Folks --- 1pm

```
value='zeyobron'
sc.parallelize([value]).collect()
```

Task 2 --- (Optional)
Cloudera Folks
open terminal

```
cd
echo 1,sai>data
hadoop fs -put data /user/cloudera/
```

pyspark and go inside

```
spark.read.format("csv").load("/user/cloudera/data").show()
```

================
Lab Folks

================

open terminal

```
cd
echo 1,sai>data
hadoop fs -put data /user/<LABUSER>/
```

pyspark and go inside

```
spark.read.format("csv").load("/user/<LABUSER>/data").show()
```

WINSCP LINK TO DOWNLOAD
https://winscp.net/download/WinSCP-6.1.1-Setup.exe

Filezilla for Mac

https://dl3.cdn.filezilla-project.org/client/FileZilla_3.65.0_macosx-x86.app.tar.bz2?h=ibsShuwvqaG7liI4mG0InQ&x=1690095964

=========================

Kafka handson -- Windows Folks

1) Download zookeeper and Kafka
2) Place it in drive (NOT INSIDE ANY SUB FOLDERS)
3) Extract Both of them
4) In the same Drive (E or D or C) -- remove the tmp if you have
5) Go inside zookeeper folder and Go inside bin folder and open cmd
6) then trigger below command and do not close that window just minimize it .\zkserver
7) then come back go inside kafka folder and open cmd
8) Trigger below command and do not close that window just minimize it .\bin\windows\kafka-server-start.bat .\config\server.properties
9) then come back go inside kafka folder---> Bin folder ----> windows folder open cmd
10) Execute below command to create kafka topic kafka-topics.bat --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic manipur1
11) Come to the same windows folder and again open cmd and execute below producer command kafka-console-producer.bat --broker-list localhost:9092 --topic manipur1
12) Come to the same windows folder and again open cmd and execute below consumer command kafka-console-consumer.bat --zookeeper localhost:2181 --topic manipur1
13) Start pushing the data atleast 10 messages --- check the consumer console to validate the data

========
https://randomuser.me/api/0.8/?results=500

1 ---- Start Nifi
2-------Start zookeeper after removing tmp folder
In E or D or C -- remove the tmp if you have
Go inside zookeeper folder and Go inside bin folder and open cmd then trigger below command and do not close that window just minimize it.\zkserver
3 ---- start kafka service and create topic then come back go inside kafka folder and open cmd
Trigger below command and do not close that window just minimize it
.\bin\windows\kafka-server-start.bat .\config\server.properties
then come back go inside kafka folder---> Bin folder ----> windows folder open cmd
Execute below command to create kafka topic
kafka-topics.bat --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic newtk
4 --- Configure invokehttp with below URL

remote url - https://randomuser.me/api/0.8/?results=10

5 --- Configure putkafka

known brokers - localhost:9092
topic name - newtk
clientname - zeyo
Ensure under settings -- check mark -- success and Failure
6 -- Open consumer console for newtk
Come to the same windows folder and again open cmd and execute below consumer command

```
kafka-console-consumer.bat --zookeeper localhost:2181 --topic newtk
```

8 -- Start Nifi check the consumer console for data

Task 1 ------

Remove Tmp folder
Start zookeeper (zookeeper commands)
Start kafka (kafka commands)
Create a topic sparktk (or use existing topic if you created any)
Open eclipse/Intellij and add spark jars
Add kafk spark streaming jars to the project
Use below template and start the stream in eclipse
Start pushing to kafka using producer console
Change the package name and object if its different

```
Cloudera Folks

================================

Task 1 ---

open pyspark shell ---  (type pyspark and enter)
value='zeyobron'
sc.parallelize([value]).foreach(print)

Lab Folks --- 1pm
value='zeyobron'
sc.parallelize([value]).collect()
=========================================================
Task 2 --- (Optional)

Cloudera Folks
----------------------------
open terminal
cd
echo 1,sai>data
hadoop fs -put data /user/cloudera/
pyspark  and go inside
spark.read.format("csv").load("/user/cloudera/data").show()

Lab Folks
----------------------------
open terminal
cd
echo 1,sai>data
hadoop fs -put data /user/<LABUSER>/
pyspark  and go inside
spark.read.format("csv").load("/user/<LABUSER>/data").show()

===============================
Scala Spark

val file1 = sc.textFile("file:///home/cloudera/revdata/file1.txt")
file1.foreach(println)
--------------------
```

```
Py Spark
----------------------------
file1 = sc.textFile("file:///home/cloudera/revdata/file1.txt")
file1.foreach(print)

===============================
Scala Spark

val gymdata = file1.filter( x => x.contains("Gymnastics"))
gymdata.foreach(println)

Py Spark

gymdata = file1.filter( lambda x :  'Gymnastics' in x )
gymdata.foreach(print)

===============================
Scala Spark

case class
schema(txnno:String,txndate:String,custno:String,amount:String,category:String,product:
String,city:String,state:String,spendby:String)
val mapsplit=gymdata.map(x => x.split(","))
val schemardd = mapsplit.map( x => schema(x(0),x(1),x(2),x(3),x(4),x(5),x(6),x(7),x(8)))
val schemafilter = schemardd.filter( x => x.product.contains("Gymnastics"))
schema.foreach(println)
--------------------------------------
Py Spark

from collections import namedtuple
schema=
namedtuple("schema",["txnno","txndate","custno","amount","category","product","city","st
ate","spendby"])

mapsplit = gymdata.map( lambda x : x.split(","))
schemardd = mapsplit.map(lambda x : schema(x[0],x[1],x[2],x[3],x[4],x[5],x[6],x[7],x[8]))
schemafilter = schemardd.filter( lambda x : 'Gymnastics' in x.product)
schemafilter.foreach(print)

===============================
```

Scala Spark

```
val schemadf = schemafilter.toDF()
schemadf.show()
```

Py Spark

```
schemadf = schemafilter.toDF()
schemadf.show()
```

==============================

Scala Spark

```
val file2= sc.textFile("file:///home/cloudera/revdata/file2.txt")
val rowmapsplit = file2.map( x => x.split(","))
import org.apache.spark.sql.Row
val rowrdd = rowmapsplit.map( x => Row(x(0),x(1),x(2),x(3),x(4),x(5),x(6),x(7),x(8)))
rowrdd.foreach(println)
```

Py Spark

```
file2= sc.textFile("file:///home/cloudera/revdata/file2.txt")
rowmapsplit = file2.map(lambda x : x.split(","))
from pyspark.sql import Row
rowrdd = rowmapsplit.map(lambda x : Row(x[0],x[1],x[2],x[3],x[4],x[5],x[6],x[7],x[8]))
rowrdd.foreach(print)
```

Scala Spark

```
import org.apache.spark.sql.types._
val structschema = StructType(Array(
    StructField("txnno",StringType,true),
    StructField("txndate",StringType,true),
    StructField("custno",StringType,true),
    StructField("amount", StringType, true),
    StructField("category", StringType, true),
    StructField("product", StringType, true),
    StructField("city", StringType, true),
    StructField("state", StringType, true),
    StructField("spendby", StringType, true)
  ))
```

```
val rowdf = spark.createDataFrame(rowrdd,structschema)
rowdf.show()
```

Py Spark

```
from pyspark.sql.types import *
structschema = StructType([ \
    StructField("txnno",StringType(),True), \
    StructField("txndate",StringType(),True), \
    StructField("custno",StringType(),True), \
    StructField("amount", StringType(), True), \
    StructField("category", StringType(), True), \
    StructField("product", StringType(), True), \
    StructField("city", StringType(), True), \
    StructField("state", StringType(), True), \
    StructField("spendby", StringType(), True) \
  ])

rowdf = spark.createDataFrame(rowrdd,structschema)
rowdf.show()
==============================
```

Scala Spark

Py Spark

```
csvdf = spark.read.format("csv").option("header","true").load("file:///home/cloudera/revdata/file3.txt")
jsondf = spark.read.format("json").load("file:///home/cloudera/revdata/file4.json")
parquetdf = spark.read.load("file:///home/cloudera/revdata/file5.parquet")
xmldf = spark.read.format("xml").option("rowtag","txndata").load("file:///home/cloudera/revdata/file6")

collist = ["txnno","txndate","custno","amount","category","product","city","state","spendby"]

schemadf1 = schemadf.select(*collist)
rowdf1 = rowdf.select(*collist)
```

```
csvdf1 = csvdf.select(*collist)
jsondf1= jsondf.select(*collist)
parquetdf1 = parquetdf.select(*collist)
xmldf1 = xmldf.select(*collist)

uniondf =
schemadf1.union(rowdf1).union(csvdf1).union(jsondf1).union(parquetdf1).union(xmldf1)
==============================
```

Scala Spark

```
import org.apache.spark.sql.functions._

val resdf =uniondf.withColumn("txndate",expr("split(txndate,'-
')[2]")).withColumnRenamed("txndate","year").withColumn("status",expr("case when
spendby='cash' then 1 else 0 end")).filter(col("txnno")>50000)
```

Py Spark

```
from pyspark.sql.functions import *
resdf =uniondf.withColumn("txndate",expr("split(txndate,'-
')[2]")).withColumnRenamed("txndate","year").withColumn("status",expr("case when
spendby='cash' then 1 else 0 end")).filter(col("txnno")>50000)

==============================
```

Scala Spark

```
val aggdf =
resdf.groupBy("category").agg(sum("amount").cast(IntegerType()).alias("total"))
```

Py Spark

```
aggdf = resdf.groupBy("category").agg(sum("amount").cast(IntegerType()).alias("total"))
==============================
```

Scala Spark

```
uniondf.write.format("parquet").partitionBy("category").mode("overwrite").save("/user/clo
udera/revdirectory")
```

Py Spark

```
uniondf.write.format("parquet").partitionBy("category").mode("overwrite").save("/user/cloudera/revdirectory")
```

==============================

Scala Spark

```
val cust =
spark.read.format("csv").option("header","true").load("file:///home/cloudera/revdata/cust.csv")

val prod =
spark.read.format("csv").option("header","true").load("file:///home/cloudera/revdata/prod.csv")

val inner = cust.join(prod,Seq("id"),"inner")
val left = cust.join(prod,Seq("id"),"left")
val right = cust.join(prod,Seq("id"),"right")
val full = cust.join(prod,Seq("id"),"full")
val anti = cust.join(prod,Seq("id"),"left_anti")
```

Py Spark

```
cust =
spark.read.format("csv").option("header","true").load("file:///home/cloudera/revdata/cust.csv")
prod =
spark.read.format("csv").option("header","true").load("file:///home/cloudera/revdata/prod.csv")

inner = cust.join(prod,["id"],"inner")
left = cust.join(prod,["id"],"left")
right = cust.join(prod,["id"],"right")
full = cust.join(prod,["id"],"full")
anti = cust.join(prod,["id"],"left_anti")
```

==============================

Scala Spark

Py Spark

```
actor = spark.read.format("json").option("multiline","true").load("file:///home/cloudera/revdata/actorsj.json")
actor.printSchema()
flattendf = actor.withColumn("Actors",explode(col("Actors")))
flattendf.printSchema()
finalflatten= flattendf.select("Actors.Birthdate","Actors.BornAt","Actors.age","Actors.hasChildren","Actors.hasGreyHair","Actors.name","Actors.photo","Actors.picture.*","Actors.weight","Actors.wife","country","version")
finalflatten.printSchema()
finalflatten.show()
================================
```

Py Spark

```
==============
Python 3
==============

import urllib.request
import ssl
context = ssl.create_default_context()
context.check_hostname = False
context.verify_mode = ssl.CERT_NONE
url = "https://randomuser.me/api/0.8/?results=500"
response = urllib.request.urlopen(url, context=context).read().decode('utf-8')
urlstring = response
print(urlstring)

==============
Python 2.7
==============

import urllib2
import ssl

# Disable SSL certificate verification by creating a custom SSL context
context = ssl.create_default_context()
context.check_hostname = False
context.verify_mode = ssl.CERT_NONE

# Make the HTTP request
url = "https://randomuser.me/api/0.8/?results=500"
response = urllib2.urlopen(url, context=context)

# Check if the request was successful
if response.getcode() == 200:
    content = response.read()
    urlstring = content
else:
    print("Failed to fetch data. Status code:", response.getcode())
    urlstring = None

# Do whatever you need with the 'urlstring' variable here
===============
```

Pyspark Project Code

```
==============================
```

https://randomuser.me/api/0.8/?results=500

```
from pyspark import SparkConf
from pyspark import SparkContext
from pyspark.sql import SparkSession
from pyspark.sql.functions import *

import urllib.request
import ssl
context = ssl.create_default_context()
context.check_hostname = False
context.verify_mode = ssl.CERT_NONE
url = "https://randomuser.me/api/0.8/?results=500"
response = urllib.request.urlopen(url, context=context).read().decode('utf-8')
urlstring = response
print(urlstring)

conf = SparkConf().setAppName("first").set("spark.driver.allowMultipleContexts","true")
sc = SparkContext(conf)

spark=SparkSession.builder.getOrCreate()
```

```
rdd = sc.parallelize([urlstring])
df = spark.read.json(rdd)
df.show()

arrayflatten= df.withColumn("results",expr("explode(results)"))

finalflatten = arrayflatten.select(
"nationality",
"results.user.cell",
"results.user.username",
"results.user.dob",
"results.user.email",
"results.user.gender",
"results.user.location.city",
"results.user.location.state",
"results.user.location.street",
"results.user.location.zip",
"results.user.md5",
"results.user.name.first",
"results.user.name.last",
"results.user.name.title",
"results.user.password",
"results.user.phone",
"results.user.picture.large",
"results.user.picture.medium",
"results.user.picture.thumbnail",
"results.user.registered",
"results.user.salt",
"results.user.sha1",
"results.user.sha256",
"seed",
"version"
)

finalflatten.show()

avrodf = spark.read.format("parquet")
              .load("file:///home/cloudera/revdata/projectsample.parquet")
avrodf.show
avrodf.printSchema()
```

```
numdf = finalflatten.withColumn("username",regexp_replace(col("username"),  "([0-9])", ""))
numdf.show()

joindf = avrodf.join(numdf,["username"],"left")
joindf.show()

availablecustomerinapi=joindf.filter("nationality is not null")
availablecustomerinapi.show()

notavailablecustomerinapi=joindf.filter("nationality is  null")
notavailablecustomerinapi.show()


availablecustomerinapi.write.format("parquet").mode("append").save("/user/cloudera/available")

notavailablecustomerinapi.write.format("parquet").mode("append").save("/user/cloudera/notavailable")
```

================================

WINSCP LINK TO DOWNLOAD

https://winscp.net/download/WinSCP-6.1.1-Setup.exe

Filezilla for Mac

===============================

https://dl3.cdn.filezilla-project.org/client/FileZilla_3.65.0_macosx-x86.app.tar.bz2?h=ibsShuwvqaG7liI4mG0InQ&x=1690095964

==================================

Spark Streaming + Kafka Integration Guide

=======================================

https://spark.apache.org/docs/2.2.0/streaming-kafka-0-10-integration.html

Task 1 ------

Remove Tmp folder
Start zookeeper (zookeeper commands)
Start kafka (kafka commands)

Create a topic sparktk (or use existing topic if you created any)

Open eclipse/Intellij and add spark jars

Add kafk spark streaming jars to the project

Use below template and start the stream in eclipse

Start pushing to kafka using producer console

Change the package name and object if its different

Code
==========

```
package pack
import org.apache.spark.SparkConf
import org.apache.spark.SparkContext
import org.apache.spark.streaming.StreamingContext
import org.apache.spark.streaming._
import org.apache.spark.sql._
import org.apache.spark.sql.functions
import org.apache.kafka.clients.consumer.ConsumerRecord
import org.apache.kafka.common.serialization.StringDeserializer
import org.apache.spark.streaming.kafka010._
import org.apache.spark.streaming.kafka010.LocationStrategies.PreferConsistent
import org.apache.spark.streaming.kafka010.ConsumerStrategies.Subscribe
import org.apache.spark.sql.functions._
object obj {
        def main(args:Array[String]):Unit={
                println("Streaming started")
                val conf = new
SparkConf().setAppName("ES").setMaster("local[*]").set("spark.driver.allowMultipleConte
xts","true")
                val sc = new SparkContext(conf)
                sc.setLogLevel("Error")
                val spark = SparkSession
                .builder()
                .getOrCreate()
                import spark.implicits._
                val ssc = new StreamingContext(conf,Seconds(2))
                val topics = Array("sparktk")
                val kafkaParams = Map[String, Object](
                        "bootstrap.servers" -> "localhost:9092",
                        "key.deserializer" -> classOf[StringDeserializer],
                        "value.deserializer" -> classOf[StringDeserializer],
                        "group.id" -> "zeyogroupid",
                        "auto.offset.reset" -> "earliest"
                        )
        val stream = KafkaUtils.createDirectStream[String, String](
                    ssc,
                    PreferConsistent,
                    Subscribe[String, String](topics, kafkaParams)
                    )
                val stream1=stream.map( x =>  x.value)
                stream1.print
                /*stream1.foreachRDD(x=>

                if(!x.isEmpty())
                {
                val df = x.toDF("value").withColumn("timstamp", current_timestamp)
                    df.show(false)
                }
                    )
*/
                ssc.start()
                ssc.awaitTermination()
        }
}
```

https://spark.apache.org/docs/2.2.0/streaming-kafka-0-10-integration.html