# 

# Ex 3

**Implement a simple map-reduce code for the wordcount problem using Java/Python. (Create the jar files and run the code using HDFS.)**

su - hadoop

mkdir ~/WordCountProject

cd ~/WordCountProject

nano WordCountMapper.java

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

public class WordCountMapper extends Mapper<LongWritable, Text, Text, IntWritable> {

private final static IntWritable one = new IntWritable(1);

private Text word = new Text();

@Override

public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {

String[] words = value.toString().split("\\s+");

for (String str : words) {

word.set(str);

context.write(word, one);

}

}

}

nano WordCountReducer.java

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Reducer;

public class WordCountReducer extends Reducer<Text, IntWritable, Text, IntWritable> {

@Override

public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {

int sum = 0;

for (IntWritable val : values) {

sum += val.get();

}

context.write(key, new IntWritable(sum));

}

}

nano WordCount.java

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount {

public static void main(String[] args) throws Exception {

Configuration conf = new Configuration();

Job job = Job.getInstance(conf, "word count");

job.setJarByClass(WordCount.class);

job.setMapperClass(WordCountMapper.class);

job.setCombinerClass(WordCountReducer.class);

job.setReducerClass(WordCountReducer.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

FileInputFormat.addInputPath(job, new Path(args[0]));

FileOutputFormat.setOutputPath(job, new Path(args[1]));

System.exit(job.waitForCompletion(true) ? 0 : 1);

}

}

mkdir wordcount\_classes

javac -classpath $(hadoop classpath) -d wordcount\_classes WordCountMapper.java WordCountReducer.java WordCount.java

jar -cvf WordCount.jar -C wordcount\_classes/ .

start-dfs.sh

start-yarn.sh

echo "Hello Hadoop Hello MapReduce" > input.txt

whoami (find username)

(hadoop - username)

hdfs dfs -mkdir /user/hadoop/input

hdfs dfs -put input.txt /user/hadoop/input

hadoop jar WordCount.jar WordCount /user/hadoop/input /user/hadoop/output

hdfs dfs -cat /user/hadoop/output/part-r-00000

Delete existing dic -error

hdfs dfs -rm -r /user/hadoop/output

hadoop jar WordCount.jar WordCount /user/hadoop/input /user/hadoop/output

hdfs dfs -cat /user/hadoop/output/part-r-00000

## Python version

mkdir ~/WordCountPythonProject

nano mapper.py

# mapper.py

import sys

# Input comes from standard input (line by line)

for line in sys.stdin:

line = line.strip() # Remove leading and trailing whitespace

words = line.split() # Split line into words

# Output each word with a count of 1

for word in words:

print(f"{word}\t1")

nano reducer.py

# reducer.py

import sys

current\_word = None

current\_count = 0

word = None

# Input comes from standard input

for line in sys.stdin:

line = line.strip()

word, count = line.split('\t', 1)

try:

count = int(count)

except ValueError:

continue

# Sum counts for each word

if current\_word == word:

current\_count += count

else:

if current\_word:

print(f"{current\_word}\t{current\_count}")

current\_word = word

current\_count = count

# Output the last word

if current\_word == word:

print(f"{current\_word}\t{current\_count}")

start-dfs.sh

start-yarn.sh

(Delete file output)

hdfs dfs -rm /user/hadoop/input/input.txt

echo "Hello Hadoop Hello MapReduce" > input1.txt

hdfs dfs -put input1.txt /user/hadoop/input

hadoop jar $HADOOP\_HOME/share/hadoop/tools/lib/hadoop-streaming-\*.jar \

-input /user/hadoop/input \

-output /user/hadoop/output \

-mapper "python3 mapper.py" \

-reducer "python3 reducer.py"

hdfs dfs -cat /user/hadoop/output/part-00000

# Ex 4

## 1. Implement map reduce for NCDC weather dataset using Hadoop -fine the max and min temperature.

start-dfs.sh

start-yarn.sh

## 2. Implement Apriori algorithm using map reduce paradigm.

start-dfs.sh

start-yarn.sh

# Create a directory for your input data in HDFS

hadoop fs -mkdir -p /user/hadoop/input

# Create the transactions file

echo -e "milk,bread,butter\nbread,butter,juice\nmilk,juice\nbread,milk,juice" > transactions.txt

# Upload the input file to HDFS

hadoop fs -put transactions.txt /user/hadoop/input

mkdir AprioriMR

cd AprioriMR

mkdir src

cd src

AprioriMR.java (create inside src)

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.Reducer;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import java.io.IOException;

import java.util.StringTokenizer;

public class AprioriMR {

public static class ItemsetMapper extends Mapper<Object, Text, Text, IntWritable> {

private final static IntWritable one = new IntWritable(1);

private Text item = new Text();

public void map(Object key, Text value, Context context) throws IOException, InterruptedException {

StringTokenizer itr = new StringTokenizer(value.toString(), ",");

while (itr.hasMoreTokens()) {

item.set(itr.nextToken().trim());

context.write(item, one);

}

}

}

public static class ItemsetReducer extends Reducer<Text, IntWritable, Text, IntWritable> {

private int minSupport;

@Override

protected void setup(Context context) {

Configuration conf = context.getConfiguration();

minSupport = conf.getInt("minSupport", 2); // Example threshold

}

public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {

int sum = 0;

for (IntWritable val : values) {

sum += val.get();

}

if (sum >= minSupport) {

context.write(key, new IntWritable(sum));

}

}

}

public static void main(String[] args) throws Exception {

Configuration conf = new Configuration();

conf.setInt("minSupport", 2); // Set minimum support here

Job job = Job.getInstance(conf, "apriori");

job.setJarByClass(AprioriMR.class);

job.setMapperClass(ItemsetMapper.class);

job.setCombinerClass(ItemsetReducer.class);

job.setReducerClass(ItemsetReducer.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

FileInputFormat.addInputPath(job, new Path(args[0]));

FileOutputFormat.setOutputPath(job, new Path(args[1]));

System.exit(job.waitForCompletion(true) ? 0 : 1);

}

}

# Navigate to the source directory

cd src

# Compile the code

javac -classpath `hadoop classpath` -d ../ AprioriMR.java

# Go back to the project root

cd ..

# Package the compiled code into a JAR file

jar -cvf AprioriMR.jar -C . .

# Run the Hadoop job

hadoop jar AprioriMR.jar AprioriMR /user/hadoop/input /user/hadoop/output

# List the output files in HDFS

hadoop fs -ls /user/hadoop/output

# View the results

hadoop fs -cat /user/hadoop/output/part-r-00000

(Troubleshooting)

# Remove input and output directories from HDFS

hadoop fs -rm -r /user/hadoop/input

hadoop fs -rm -r /user/hadoop/output

# List files in the src directory to check if AprioriMR.java is there

ls src

# Navigate to the src directory

cd src

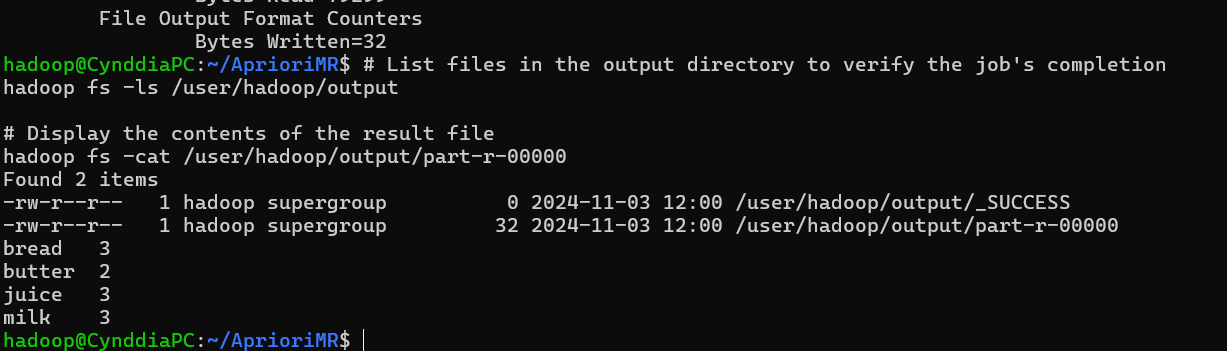
# Compile the code using Hadoop's classpath

javac -classpath `hadoop classpath` -d ../ AprioriMR.java

(if already output file is there)

hadoop fs -rm -r /user/hadoop/output

hadoop jar AprioriMR.jar AprioriMR /user/hadoop/input /user/hadoop/output



# Ex 5

## Installing pyspark + jupyter

[How to Run PySpark on Jupyter Notebook | phoenixNAP KB](https://phoenixnap.com/kb/jupyter-pyspark)

Pyspark with jupyter

<https://chatgpt.com/share/6727240e-8e94-8013-a25c-b04e1d94de0d>

## Run the wordcount program that you did using hadoop usingpyspark.

import os

import findspark

from pyspark.sql import SparkSession

# Set environment variables

os.environ['SPARK\_HOME'] = "/home/hadoop/.local/lib/python3.10/site-packages/pyspark" # Adjust this if necessary

os.environ['HADOOP\_HOME'] = "/path/to/hadoop" # If you have Hadoop installed

os.environ['PYSPARK\_PYTHON'] = "python3" # Or "python" depending on your setup

# Initialize findspark to find the Spark installation

findspark.init()

# Create a Spark session

spark = SparkSession.builder \

.appName("WordCount") \

.getOrCreate()

# Print Spark version to confirm it’s working

print(spark.version)

# Example: Word Count

text\_data = ["Hello world", "Hello Spark", "Hello Jupyter"]

rdd = spark.sparkContext.parallelize(text\_data)

word\_counts = rdd.flatMap(lambda line: line.split(" ")) \

.map(lambda word: (word, 1)) \

.reduceByKey(lambda a, b: a + b)

# Collect and print results

for word, count in word\_counts.collect():

print(f"{word}: {count}")

## Movielens dataset - find out for each movie, how are the ratings distributed

(create txt movielens dataset in jupyter directory - not local file )

from pyspark.sql import SparkSession

from pyspark.sql.functions import col, count

# Create a Spark session

spark = SparkSession.builder \

.appName("MovieLens Ratings Distribution") \

.getOrCreate()

data\_path = "Movielens.txt"

movies\_df = spark.read.csv(data\_path, sep='\t', inferSchema=True) \

.toDF("user\_id", "movie\_id", "rating", "timestamp")

# Calculate the distribution of ratings for each movie

rating\_distribution = movies\_df.groupBy("movie\_id", "rating") \

.agg(count("rating").alias("rating\_count")) \

.orderBy("movie\_id", "rating")

rating\_distribution.show()

# Ex 6

## 1. Use the "friends\_test" dataset. Col1 is ID, Col2 is name, Col 3 is Age, Col 4 is num of friends. Understand mapvalues function of RDD in spark and find the average number of friends for each unique age present in the dataset.

import pyspark

from pyspark.sql import SparkSession

spark = SparkSession.builder.master('local').appName('friends\_dataset').getOrCreate()

sc=spark.sparkContext

# Load dataset

data\_path='friends\_test.csv'

rdd=sc.textFile(data\_path)

age\_friends\_rdd = rdd.map(lambda line: line.split(',')) \

.map(lambda cols: (int(cols[2]), (int(cols[3]), 1)))

# Sum up num\_friends and count for each age

age\_friends\_totals = age\_friends\_rdd.reduceByKey(lambda a, b: (a[0] + b[0], a[1] + b[1]))

# Calculate the average number of friends for each age

average\_friends\_by\_age = age\_friends\_totals.mapValues(lambda total: total[0] / total[1])

# Collect and display the results

results = average\_friends\_by\_age.collect()

for age, avg\_friends in results:

print(f"Age: {age}, Average Friends: {avg\_friends:.2f}")

## 2. Use the "temp.csv" dataset. Column headers are present in the dataset. Understand filter operations and filter out only the "TMIN" values from the "desc" column. With the resultant data (RDD) find the following: a. Minimum temperature (overall) b. Minimum temperature for every ItemID c. Minimum temperature for every StationID.

## Use the same dataset, filter only "TMAX" column and find the maximum temperatures just like the ones mentioned above.

from pyspark.sql import SparkSession

# Initialize Spark session

spark = SparkSession.builder \

.appName("Temp\_dataset") \

.getOrCreate()

# Load dataset

data\_path = "temp.csv" # Replace with actual path

rdd = spark.sparkContext.textFile(data\_path)

# Extract the header

header = rdd.first()

data\_rdd = rdd.filter(lambda row: row != header) # Remove the header

# Split each row by comma and convert to (StationID, ItemID, desc, temp) format

data\_rdd = data\_rdd.map(lambda line: line.split(",")) \

.map(lambda cols: (cols[0], cols[1], cols[2], float(cols[3]))) # assuming temp is in column 4

# 1. Filter for "TMIN" and find minimum temperatures

tmin\_rdd = data\_rdd.filter(lambda x: x[2] == "TMIN")

# a. Overall minimum temperature

overall\_min\_tmin = tmin\_rdd.map(lambda x: x[3]).min()

# b. Minimum temperature for each ItemID

min\_temp\_by\_item = tmin\_rdd.map(lambda x: (x[1], x[3])) \

.reduceByKey(lambda a, b: min(a, b))

# c. Minimum temperature for each StationID

min\_temp\_by\_station = tmin\_rdd.map(lambda x: (x[0], x[3])) \

.reduceByKey(lambda a, b: min(a, b))

# Display results for TMIN

print(f"Overall minimum temperature (TMIN): {overall\_min\_tmin}")

print("Minimum temperature for each ItemID (TMIN):")

for item, min\_temp in min\_temp\_by\_item.collect():

print(f"ItemID: {item}, Min Temp: {min\_temp}")

print("Minimum temperature for each StationID (TMIN):")

for station, min\_temp in min\_temp\_by\_station.collect():

print(f"StationID: {station}, Min Temp: {min\_temp}")

# 2. Filter for "TMAX" and find maximum temperatures

tmax\_rdd = data\_rdd.filter(lambda x: x[2] == "TMAX")

# a. Overall maximum temperature

overall\_max\_tmax = tmax\_rdd.map(lambda x: x[3]).max()

# b. Maximum temperature for each ItemID

max\_temp\_by\_item = tmax\_rdd.map(lambda x: (x[1], x[3])) \

.reduceByKey(lambda a, b: max(a, b))

# c. Maximum temperature for each StationID

max\_temp\_by\_station = tmax\_rdd.map(lambda x: (x[0], x[3])) \

.reduceByKey(lambda a, b: max(a, b))

# Display results for TMAX

print(f"Overall maximum temperature (TMAX): {overall\_max\_tmax}")

print("Maximum temperature for each ItemID (TMAX):")

for item, max\_temp in max\_temp\_by\_item.collect():

print(f"ItemID: {item}, Max Temp: {max\_temp}")

print("Maximum temperature for each StationID (TMAX):")

for station, max\_temp in max\_temp\_by\_station.collect():

print(f"StationID: {station}, Max Temp: {max\_temp}")

# Ex 7

## Set up a simple Hadoop environment using Docker containers, including at least one NameNode and one DataNode. Ensure the containers are properly configured to interact with each other.After the setup, verify that the Hadoop cluster is operational by running a simple HDFS file operation (e.g., uploading a fileto HDFS).

Check and install docker

docker --version

sudo apt install docker.io

sudo docker run hello-world

docker network create hadoop-net

docker pull bde2020/hadoop-namenode:2.0.0-hadoop3.2.1-java8

docker pull bde2020/hadoop-datanode:2.0.0-hadoop3.2.1-java8

docker run -d \

--name namenode \

--network hadoop-net \

-e CLUSTER\_NAME="my-hadoop-cluster" \

-e CORE\_CONF\_fs\_defaultFS=hdfs://namenode:9000 \

bde2020/hadoop-namenode:2.0.0-hadoop3.2.1-java8

docker run -d \

--name datanode \

--network hadoop-net \

-e CORE\_CONF\_fs\_defaultFS=hdfs://namenode:9000 \

bde2020/hadoop-datanode:2.0.0-hadoop3.2.1-java8

docker ps

docker logs namenode

docker logs datanode

(Run simple hdfs file system)

echo "Hello Hadoop!" > hello.txt

docker cp hello.txt namenode:/hello.txt

docker exec -it namenode hdfs dfs -mkdir -p /user/root

docker exec -it namenode hdfs dfs -put /hello.txt /user/root/

docker exec -it namenode hdfs dfs -ls /user/root/

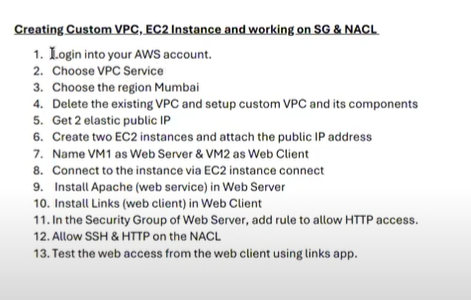
docker exec -it namenode hdfs dfs -cat /user/root/hello.txt

(clean up)

docker rm -f namenode datanode

docker network rm hadoop-net

# AWS



## VPC and EC2

VPC : snu-dc-vpc

Ipv4 : 192.168.0.0/16

=> Connect subnet to vpc created

Subnet name: public-subnet

Availability zone: Mumbai

IPv4 subnet block (2nd one ) : 192.168.1.0/24

=> Internet gateway

Name tag: snuc-dc-igw

=> Connect ig to vpc created

=> Route tables

Click on route table id -> edit routes -> add routes

0.0.0.0/0 - Internet gateway -> igw (auto-completion)

(chk point: completed setting up of vpc and it’s components)

=> Elastic IP

Allocate ip address (create web server and web client (rename after creation)

=> EC2

Go to instances -> launch instances

Number of instances: 2

Name: VMs

Quick start -> change to ubuntu

Key pair -> proceed without key pair (not)

Now launch instance

-> go to instance

Rename to web server and web client

-> go to elastic ip

Click on web server ip add -> associate elastic ip add with it

Instance: choose web server

Do same for web client

(chk point: create two ec2 instances and attach public ip address)

-> go to instances

Click web server and connect -> connect instance

(if error ipv4 not public check again and refresh then connect)

=>New tab for terminal opens up(ec2-instance-connect)

ping 8.8.8.8

sudo apt update

sudo apt install apache2

service apache2 status (ctrl c to escape)

=> connect web client to instance

(tab should pop up)

ping 8.8.8.8

sudo apt update

sudo apt install links

clear

(chck point: Install apache and links )

Click on web server ip add -> go to security

Click security group id

Click on edit inbound rules

-> add rule -> http -> source -> anywhere ipv4

(chck point: HTTP access )

Go to instances -. Click link for web server go to security

=> nacl is in vpc

Vpc ->network security -> nacl

Click on nacl id

Click on edit inbound rules

Remove the existing rule

Add new rule

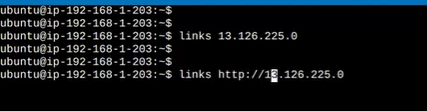
Type : SSH (20)

New rule -> 101 -> HTTP(80) -> save changes

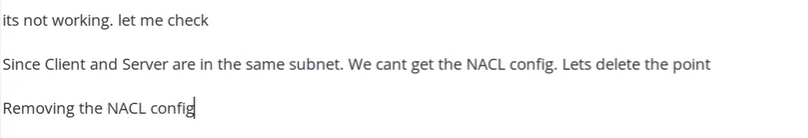
(chck point: test the web access)

Go to ec2 interface for web client (the cleared one)

links (the public ip of web server -> in interface terminal)



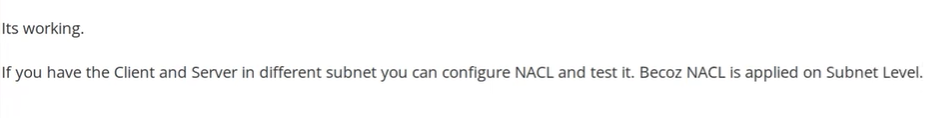
Now removing the nacl from



Remove the created inbound rules (100,101)

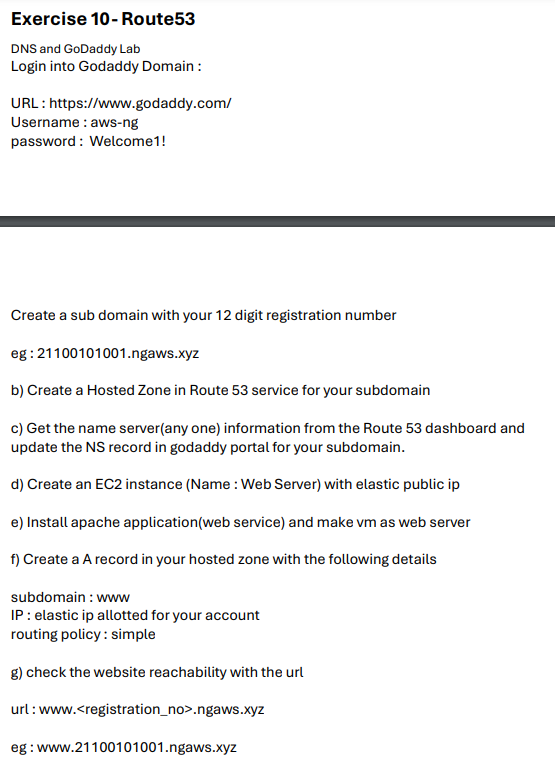
Add new rule -> 100-> all traffic -> save changes

Now it will work



(1. VPC & EC2 Lab : https://youtu.be/AsSQb--MNXA (no audio))

## Route 53 Labs



**Prerequisite :**

Create an instance(using default vpc ) and elastic ips (like for ec2) and do connection.(for web server and client.

(recommended- do vpc ec2 ex and use that web server and client - as default vpc is not working much )

=> Route 53 hosted zone

Search route 53

Go to hosted zones -> create hosted zone

Domain name: 21011101037.ngaws.xyz

Create it

=> Go daddy domain



<https://dcc.godaddy.com/control/portfolio/ngaws.xyz/settings>

(chck point:login into go daddy )

Copy the link with .com for the hosted zone created for type ns

(click the hosted zone name to go into the two diff hosted zones - ns and soa)

-> click new record in go daddy -> type : ns -> name: reg no. -> value: copied .com ->create it

(chck point: get name server from route 53 and update ns record in godaddy)

-> go back to route 53 and create record (from the pg where ns and soa is there )

Record name : www

Value : ip address of web server (go to instance and take public ipv4 of web server)

Routing policy : simple

Click create records

(chck point: create record in aws hosted zone)

=> check website reachability

[www.21011101037.ngaws.xyz](http://www.21011101037.ngaws.xyz)

->Go to ec2 connect instance (terminal) ( of web client)(click connect button for web client)

links website

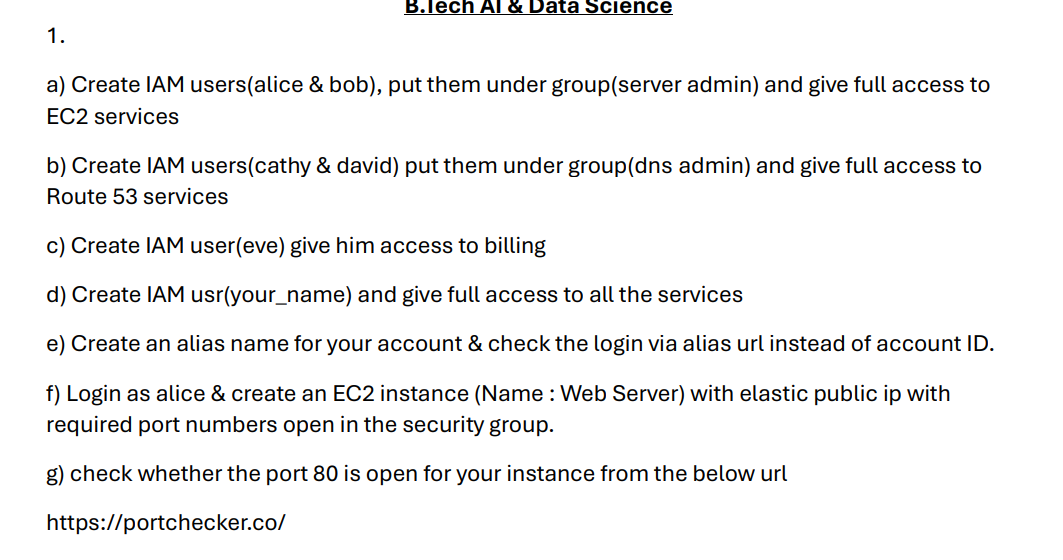
Eg. links [www.21011101037.ngaws.xyz](http://www.21011101037.ngaws.xyz)

Commands:

links [www.21011101037.ngaws.xyz](http://www.21011101037.ngaws.xyz) (connects the client to it)

nslookup [www.21011101037.ngaws.xyz](http://www.21011101037.ngaws.xyz)

## IAM :



=> go for IAM

=> users -> create user

=> Specify user details

Click Provide user access to the AWS Management Console - *optional*

-> click are you providing console access to person - yes -> next

Set permissions -> click next

Now create the user

Then copy the password of alice in retrieve password and the console sign-in url

Return to users list

=> open new tab -> sign in as alice (iam user) (do in incognito as it signs you out)

(use the link copied)

-> asks you to set a new password when login set something and note (password - password1)

(chck point: logged in as Alice) -> everything denied

Change alice to mumbai

-> go to alice -> ec2 (denied)

-> go to y/n -> go to iam -> users -> click alice (user name)

-> permission policies -> add permission

Attach policies directly -> permission policies (ec2) -> full acess -> next -> add permissions



Now alice will have ec2 access

-> go to y/n -> create user group name (server-team) -> click alice in user-name -> attach permission (search ec2f) ->click policy name ec2 full access -> create user group.

-> go to users -> click alice user name -> permissions -> ec2 full access -> remove this permission for user. (comes remove user from group server-team)

While removing alice must have Directly, Group [server-team](https://us-east-1.console.aws.amazon.com/iam/home?region=ap-south-1#/groups/details/server-team) for ec2 full access permission.

=> user groups ->server-team -> add users

-> click alice and put add

=> users -> create user -> bob -> i want to create iam user -> untick the users must create new password -> next -> add users group server-team ->create user

-> copy bob’s console password and sign-in url

=> sign out of alice -> sign in as bob

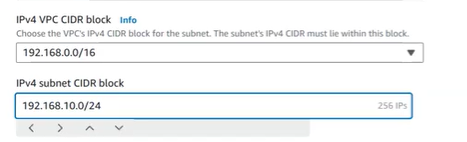
Bob will have the ec2 stuff accessible

=> go to vpc in bob

-> subnets -> create subnet -> (pre-requisite -> create vpc in y/n)

Subnet name: test

Then click -> ip 2nd address



=> go to y/n -> user groups -> server-team ->look

-> go to bob -> route 53 -> create hosted zone (use the get started for it)

Domain name: testzone

Create it -> will get error

-> go to y/n -> user groups ->permissions -> permission policy -> attach policy

Admin acess ->attach policies



-> go to bob -> route 53 -> create hostzone -> test-site.com -> create it

Now go to hosted zones -> click on host zone and delete it

=> Go to y/n -> dashboard -> aws account -> create alias -> snuc -> copy link

-> sign out of bob -> sign in with your alias -> iam user name: alice -> password : alice password

=> In alice@snuc

-> go to user groups -> click amazon fullaccess -> remove it. (all users in group notification)

-> last accessed -> only admin access will be there

—----------------------------------------------

# Pyspark

To get started with PySpark, let’s cover some basic code snippets, including initializing a Spark session, loading data, performing simple transformations, and displaying results.

### **1. Starting PySpark and Initializing a Spark Session**

python

Copy code

from pyspark.sql import SparkSession

# Initialize a Spark session

spark = SparkSession.builder \

.appName("BasicPySpark") \

.getOrCreate()

### **2. Loading a Dataset**

Assuming you have a CSV file to load, this is how you can load it as a DataFrame:

python

Copy code

# Load CSV file into DataFrame

df = spark.read.csv("path\_to\_file.csv", header=True, inferSchema=True)

# Show the schema (column names and types)

df.printSchema()

# Display the first few rows

df.show(5)

* **header=True**: Uses the first row as the header.
* **inferSchema=True**: Automatically detects the column data types.

### **3. Basic DataFrame Operations**

Now that we have the data, let’s perform some basic operations on it:

#### **a. Selecting Columns**

python

Copy code

# Select specific columns

df.select("column1", "column2").show()

#### **b. Filtering Data**

python

Copy code

# Filter rows based on conditions

filtered\_df = df.filter(df["column1"] > 100)

filtered\_df.show()

#### **c. Adding a New Column**

python

Copy code

# Create a new column

df = df.withColumn("new\_column", df["existing\_column"] \* 2)

df.show()

#### **d. Grouping and Aggregation**

python

Copy code

# Perform group by and aggregate

df.groupBy("column1").agg({"column2": "mean"}).show()

### **4. Saving a DataFrame**

You can save the DataFrame in various formats like CSV, JSON, or Parquet:

python

Copy code

# Save DataFrame as a CSV file

df.write.csv("path\_to\_output\_directory", header=True)

### **5. Stopping the Spark Session**

When you’re done, stop the Spark session to release resources:

python

Copy code

# Stop the Spark session

spark.stop()

These snippets will get you started with basic data loading, transformations, and saving in PySpark! Let me know if you need specific transformations or deeper insights on any part.

# 

# 

# 

# 

# 

# 