|  |  |
| --- | --- |
| **Ex. No 1** | **Linux Commands** |
| **16th July** |

**Aim: To understand basic linux commands**

**A screenshot of a computer

Description automatically generated**

A screenshot of a computer program

Description automatically generated A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

**Result:** Linux Commands were executed

|  |  |
| --- | --- |
| **Ex. No 2** | **Install Hadoop** |
|  |

**Aim: To install Hadoop in the working system**

**Output:**

A computer screen with white text

Description automatically generated

**Result:** Hadoop was successfully installed on the PC

|  |  |
| --- | --- |
| **Ex. No: 3** | **Implementing MapReduce** |
| **30th July** |

**Aim: To Implement a simple map-reduce code for the wordcount problem using Java/Python.**

**Program:**

Mapper.py

#!/usr/bin/env python3

import sys

for line in sys.stdin:

    fields = line.strip().split()

    if len(fields) >= 8:

        date = fields[1]

        max\_temp = fields[7]

        min\_temp = fields[6]

        try:

            print(f"{date}\t{max\_temp}\t{min\_temp}")

        except ValueError:

            continue

Reducer.py

#!/usr/bin/env python3

import sys

current\_date = None

current\_max\_temp = float('-inf')

current\_min\_temp = float('inf')

for line in sys.stdin:

    line = line.strip()

    date, max\_temp, min\_temp = line.split("\t")

    try:

        max\_temp = float(max\_temp)

        min\_temp = float(min\_temp)

    except ValueError:

        continue

    if current\_date == date:

        current\_max\_temp = max(current\_max\_temp, max\_temp)

        current\_min\_temp = min(current\_min\_temp, min\_temp)

    else:

        if current\_date:

            print(f"{current\_date}\t{current\_max\_temp}\t{current\_min\_temp}")

        current\_date = date

        current\_max\_temp = max\_temp

        current\_min\_temp = min\_temp

if current\_date:

    print(f"{current\_date}\t{current\_max\_temp}\t{current\_min\_temp}")

**Output:**

**A computer screen with white text

Description automatically generated**

**Result: Thus map reduce program is implemented using python**

|  |  |
| --- | --- |
| **Ex. No: 4** | **Implementing MapReduce2** |
| **6th Aug** |

**Aim: To understand how map reduce program works using Apriori algorithm**

**Program:**

Mapper.py

import sys

from itertools import combinations

def generate\_combinations(items, k):

    return combinations(sorted(set(items)), k)

k = 3

for line in sys.stdin:

    items = line.strip().split()[1:]

    item\_combinations = generate\_combinations(items, k)

    for itemset in item\_combinations:

        print(f"{','.join(itemset)}\t1")

Reducer.py

import sys

current\_itemset = None

current\_count = 0

min\_support = 2

for line in sys.stdin:

    itemset, count = line.strip().split("\t")

    count = int(count)

    if current\_itemset == itemset:

        current\_count += count

    else:

        if current\_itemset and current\_count >= min\_support:

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

        current\_itemset = itemset

        current\_count = count

if current\_itemset and current\_count >= min\_support:

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

**Output:**

**A black screen with white text

Description automatically generated**

**Result: Thus Apriori implemented using the map reduce program**

|  |  |
| --- | --- |
| **Ex. No: 5** | **Spark and PySpark** |
| **16th August** |

**Aim: To Install and understand the usage of the Pyspark**

**Program:**

Word\_count.py

from pyspark import SparkContext

# Initialize SparkContext

sc = SparkContext("local", "Word Count")

# Read the file

text\_file = sc.textFile("u.txt")

# Perform word count

counts = text\_file.flatMap(lambda line: line.split("\t")) \

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

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

# Collect the results

output = counts.collect()

# Print the output

for (word, count) in output:

    print(f"{word}: {count}")

# Stop the SparkContext

sc.stop()

movie\_rating.py

from pyspark import SparkContext

# Initialize SparkContext

sc = SparkContext("local", "Movie Ratings Distribution")

# Read the file

text\_file = sc.textFile("u.txt")

# Parse the lines to extract movie ID and rating

movie\_ratings = text\_file.map(lambda line: line.split("\t")) \

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

# Count the number of each rating for each movie

ratings\_distribution = movie\_ratings.map(lambda x: ((x[0], x[1]), 1)) \

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

                                    .map(lambda x: (x[0][0], (x[0][1], x[1]))) \

                                    .groupByKey() \

                                    .mapValues(list)

# Collect and print the results

output = ratings\_distribution.collect()

for (movie\_id, ratings) in output:

    print(f"Movie ID: {movie\_id}, Ratings Distribution: {ratings}")

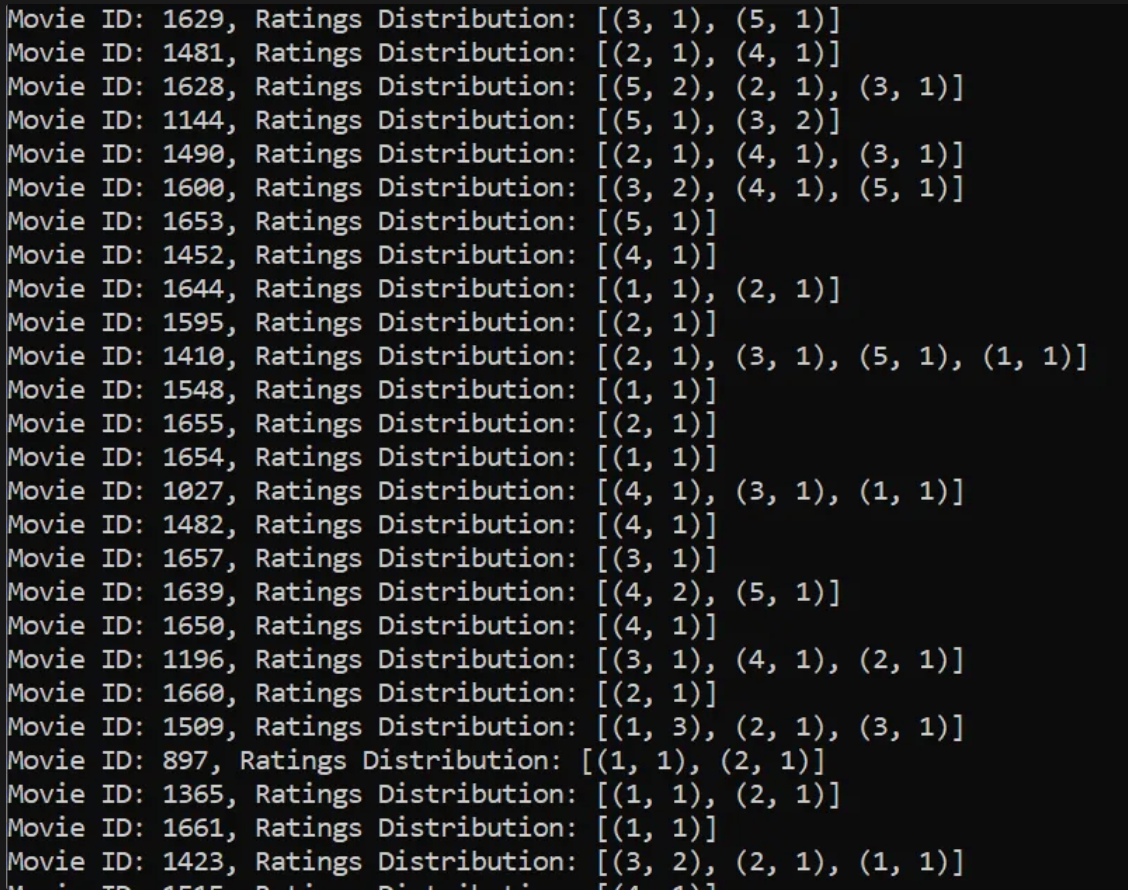
# Stop the SparkContext

sc.stop()

**Output:**

**A screen shot of a computer

Description automatically generated**

****

**Result: Required results achieved by performing following codes**

|  |  |
| --- | --- |
| **Ex. No: 6** | **PySpark** |
| **20th August** |

**Aim: To understand map values of RDD in spark**

**Program:**

Q1

from pyspark import SparkContext

sc = SparkContext("local", "Friends Average")

data = sc.textFile("friends\_test.csv").map(lambda line: line.split(","))

age\_friends\_rdd = data.map(lambda x: (int(x[2]), int(x[3])))

combined = age\_friends\_rdd.combineByKey(

    lambda friends: (friends, 1),

    lambda acc, friends: (acc[0] + friends, acc[1] + 1),

    lambda acc1, acc2: (acc1[0] + acc2[0], acc1[1] + acc2[1])

)

average\_friends\_rdd = combined.mapValues(lambda x: x[0] / x[1])

results = average\_friends\_rdd.collect()

for age, avg\_friends in results:

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

sc.stop()

Q2

from pyspark import SparkContext

sc = SparkContext("local", "Temperature Analysis")

data = sc.textFile("temp.csv").zipWithIndex().filter(lambda x: x[1] != 0).map(lambda x: x[0].split(","))

temp\_rdd = data.map(lambda x: (x[0], x[1], x[2], int(x[3])))

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

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

print(f"Overall Minimum Temperature: {overall\_min\_temp}")

min\_temp\_per\_itemID = tmin\_rdd.map(lambda x: (x[0], x[3])).reduceByKey(lambda a, b: min(a, b))

print("Minimum Temperature for each ItemID:")

for itemID, min\_temp in min\_temp\_per\_itemID.collect():

    print(f"ItemID: {itemID}, Minimum Temperature: {min\_temp}")

min\_temp\_per\_stationID = tmin\_rdd.map(lambda x: (x[1], x[3])).reduceByKey(lambda a, b: min(a, b))

print("Minimum Temperature for each StationID:")

for stationID, min\_temp in min\_temp\_per\_stationID.collect():

    print(f"StationID: {stationID}, Minimum Temperature: {min\_temp}")

sc.stop()

Q3

from pyspark import SparkContext

sc = SparkContext("local", "Temperature Analysis")

data = sc.textFile("temp.csv").zipWithIndex().filter(lambda x: x[1] != 0).map(lambda x: x[0].split(","))

temp\_rdd = data.map(lambda x: (x[0], x[1], x[2], int(x[3])))

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

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

print(f"Overall Maximum Temperature: {overall\_max\_temp}")

max\_temp\_per\_itemID = tmax\_rdd.map(lambda x: (x[0], x[3])).reduceByKey(lambda a, b: max(a, b))

print("Maximum Temperature for each ItemID:")

for itemID, max\_temp in max\_temp\_per\_itemID.collect():

    print(f"ItemID: {itemID}, Maximum Temperature: {max\_temp}")

max\_temp\_per\_stationID = tmax\_rdd.map(lambda x: (x[1], x[3])).reduceByKey(lambda a, b: max(a, b))

print("Maximum Temperature for each StationID:")

for stationID, max\_temp in max\_temp\_per\_stationID.collect():

    print(f"StationID: {stationID}, Maximum Temperature: {max\_temp}")

sc.stop()

**Output:**

A screenshot of a computer

Description automatically generatedA screen shot of a computer

Description automatically generated

**Result: The required output is achieved using pyspark**

|  |  |
| --- | --- |
| **Ex. No: 9** | **EC2** |
| **10th September** |

**Aim: To create and EC2 Instance on the AWS**

**Program:**

- Go to Instances in EC2

- Launch Instance

- Configure the Snapshots needed

- Connect to the instance

- Sudo apt update

- Sudo apt install apache2

- service apache2 status

- cd /var/www/html/

- sudo chmod 777 index.html

- Change the index file

- copy the public IP and paste on the browser (get results)

**Output:**

A screenshot of a computer

Description automatically generated

A computer screen shot of a black screen

Description automatically generated

A screenshot of a computer

Description automatically generated

**Result:**

Thus using AWS, we have successfully created an EC2 Instance

|  |  |
| --- | --- |
| **Ex. No: 10** | **Route53** |
| **1st October** |

**Aim: To perform DNS mapping using Route 53 service in AWS**

**Program:**

- Start the AWS Akshay S G Instance

- Go to Godaddy

- Create a NS record by rollnumber (21011101079) and the value with one from the AWS route 53 service (.com)

- Comeback to AWS create a record with www. and simple routing policy

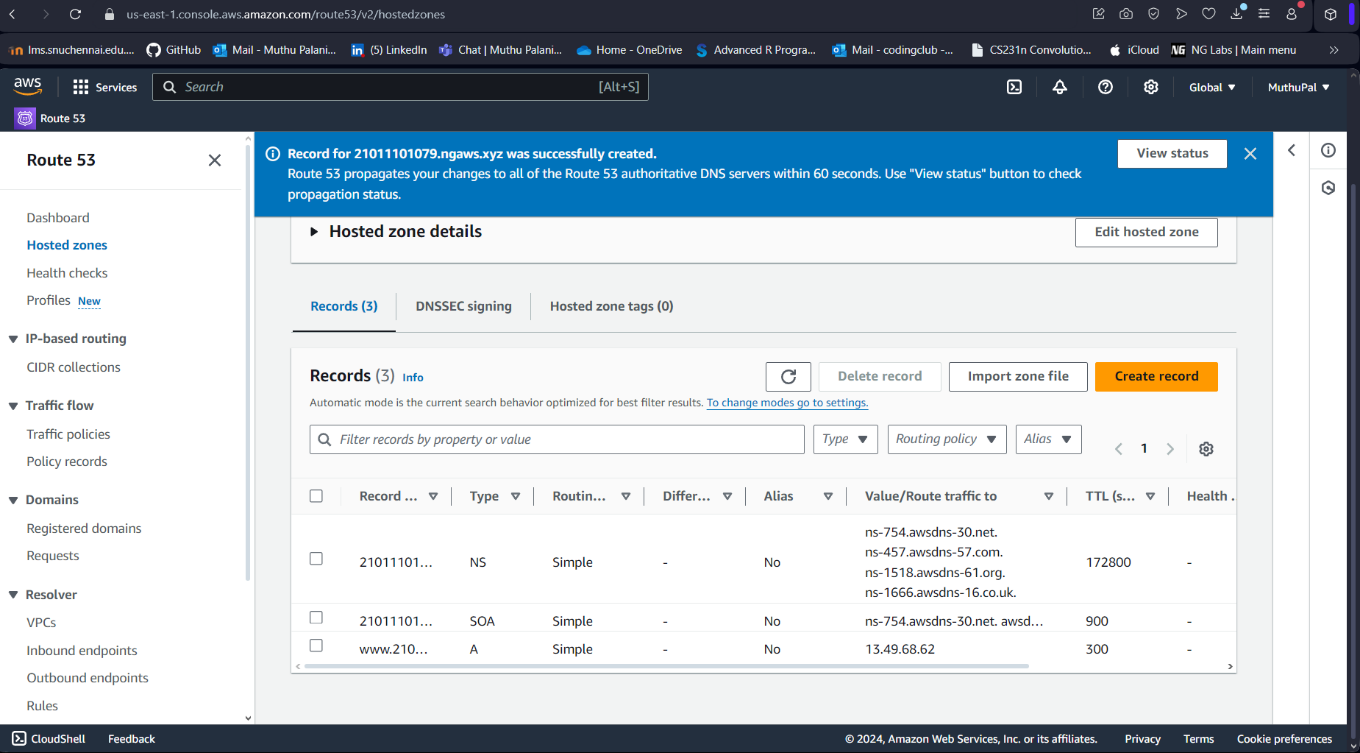
- nslookup 21011101079.ngaws.xyz

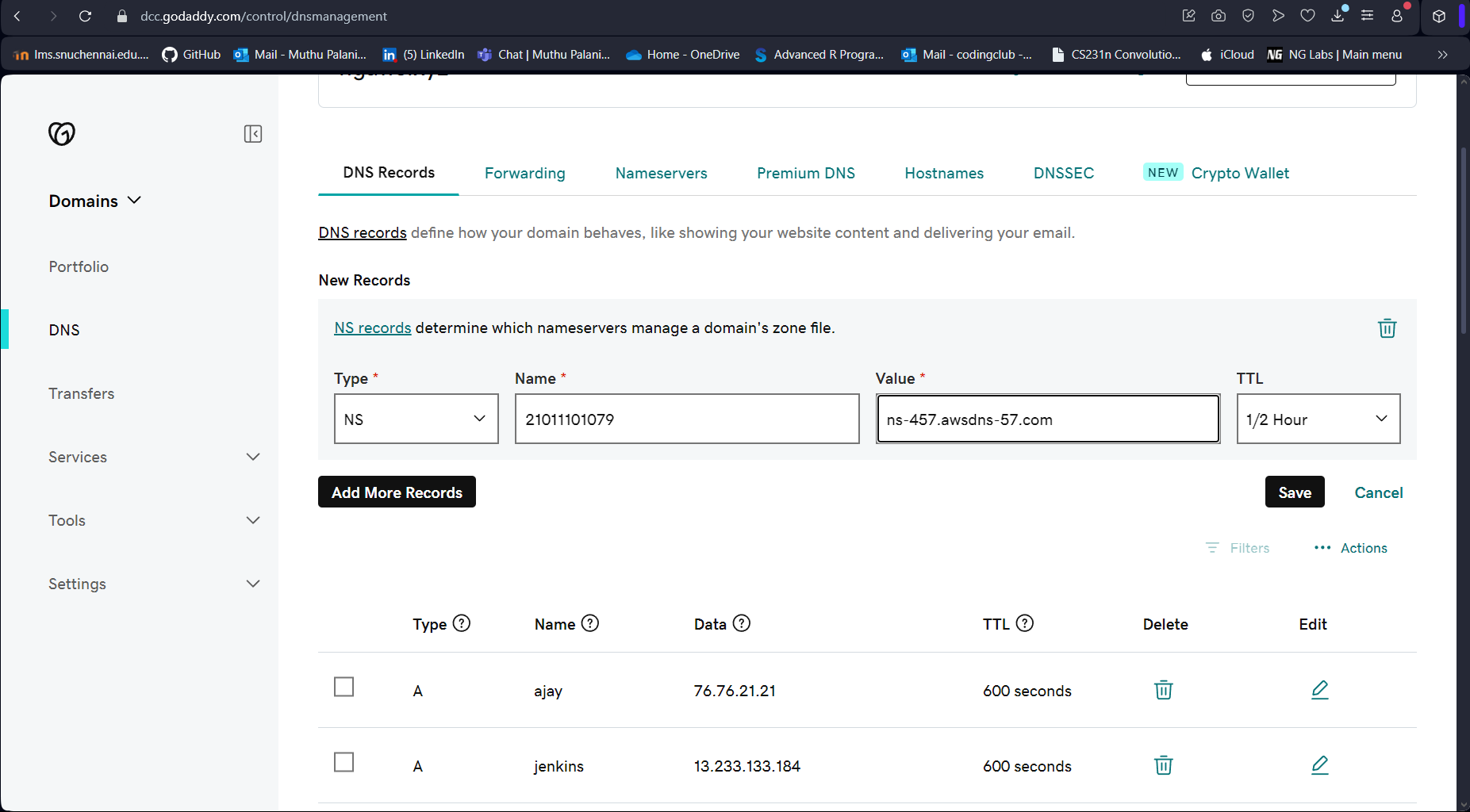
- links 21011101079.ngaws.xyz

**Output:**

A screenshot of a computer

Description automatically generated





A screenshot of a computer

Description automatically generated

**Result:** Thus, using Route 53 service in AWS, we have successfully mapped DNS

|  |  |
| --- | --- |
| **Ex. No: 11** | **IAM** |
| **8th October** |

**Aim: To understand the working of IAM service in AWS**

**Program:**

**Create IAM Groups**

* Create a group named server admin and attach a policy for full access to EC2.
* Create a group named dns admin and attach a policy for full access to Route 53.

**Add Users to Groups**

* Create IAM users alice and bob, and add them to the server admin group.
* Create IAM users cathy and david, and add them to the dns admin group.

**Create a Billing User**

* Create IAM user eve and attach a policy to provide billing access.

**Create an Admin User**

* Create IAM user Akshay S Gand attach a policy for full access to all services.

**Create Account Alias**:

* Create an alias for your account in the IAM console.
* Use the alias URL to log in instead of the account ID.

**Output:**

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Result:** Thus, using AWS, we understand how IAM works