Configuration Manual

MSc Research Project
Cloud Computing

Sathish Kumar Krisnamoorthy
Student ID: X20208057

School of Computing
National College of Ireland

Supervisor: Shivani Jaswal

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Sathish Kumar Krisnamoorthy
Student ID:	X20208057
Programme:	Cloud Computing
Year:	2022
Module:	MSc Research Project
Supervisor:	Shivani Jaswal
Submission Due Date:	15/08/2022
Project Title:	Configuration Manual
Word Count:	1407
Page Count:	22

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	15th August 2022

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Sathish Kumar Krisnamoorthy
X20208057

1 Prescript

The prescript is used for generating random coordinates with the specified country boundaries and list of peers within 1000 km of proximity. This script is deployed to the AWS EC2 instance T2.X2Large. We used seven such instances running in parallel to generate 50K random coordinates for India and 5K coordinates for each of China and the USA.

The EC2 template Figure 1 pulls the prescript code from git and instals all the dependencies with the help of the shell script written in user-data Figure 2. With the help of EC2 Template Figure 3 the process of instance deployment is automated and starts running in Figure 4. After SSH to the running instance, it can be seen that the dependencies are installed and the code is pulled from GitHub Figure 5. Once the application is started in the EC2 instance, it looks like this: Figure 6. This is the memory usage of the application, Figure 7.

The code contains various methods to do the following tasks:

1. Join the two datasets with respect to the country's geolocation. (One dataset is from Ookla, which contains info about mobile devices and their attributes, and the other dataset contains the geographical boundary information of all the countries in the world.)

2. Generate random coordinates and peer device information for the specified country, including its border.

3. Collect them in a CSV file and upload the dataset to the S3 Bucket. The dataset folders are shown in Figure 8 and the total S3 memory usage metrics are shown in Figure 9.

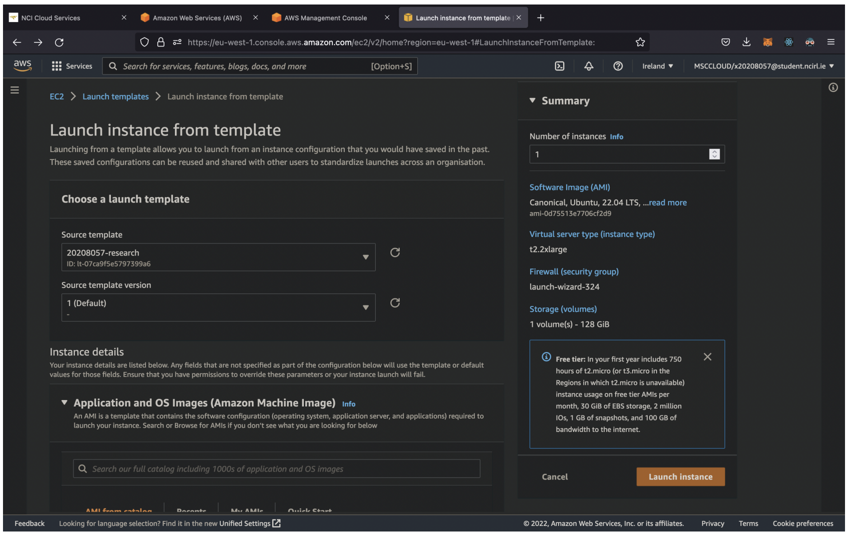

Figure 1: Prescript Template AWS

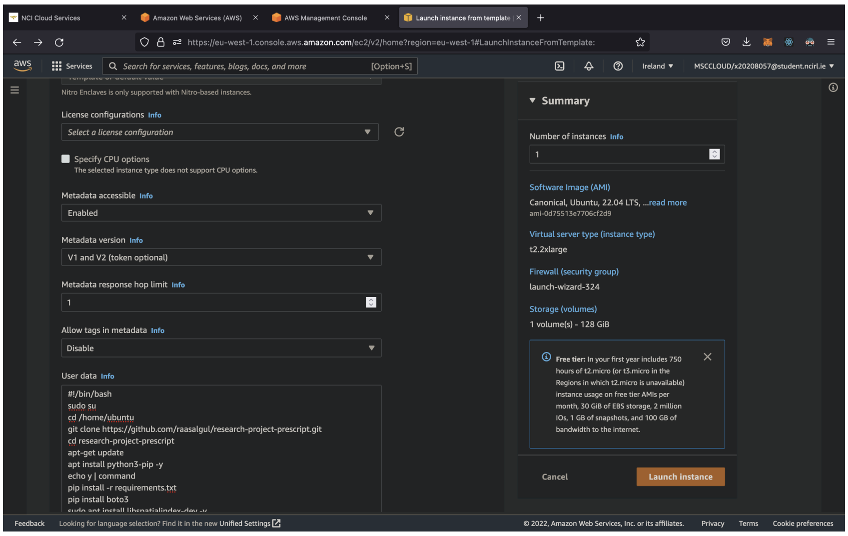

Figure 2: Prescript Template User Data

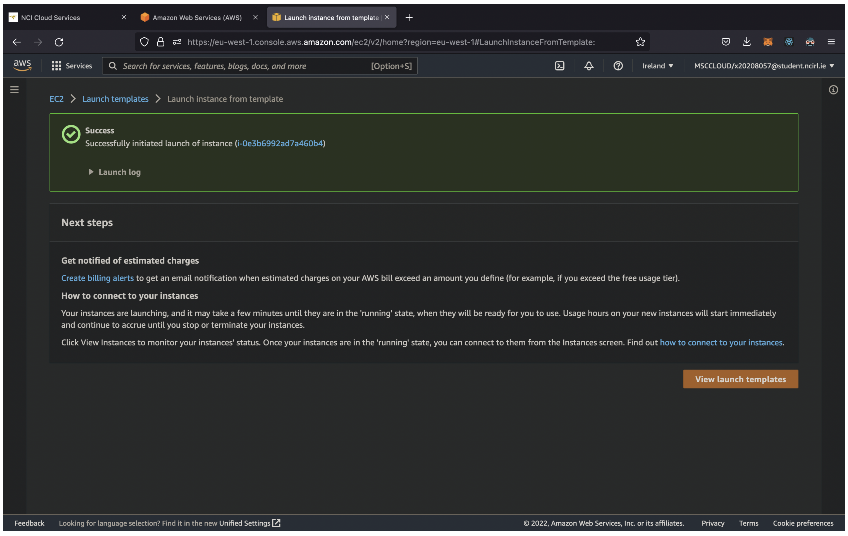

Figure 3: EC2 Instance Deployed

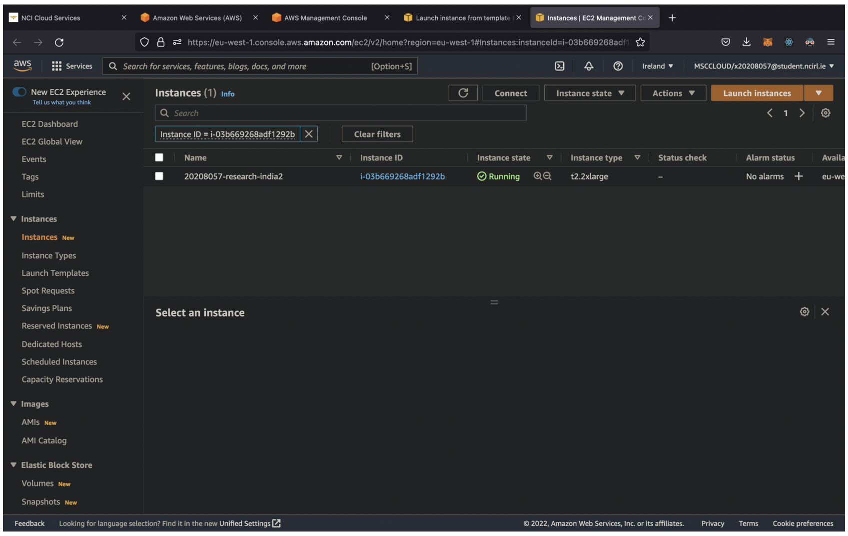

Figure 4: EC2 Instance Running

Figure 5: Prescript SSH Initial State

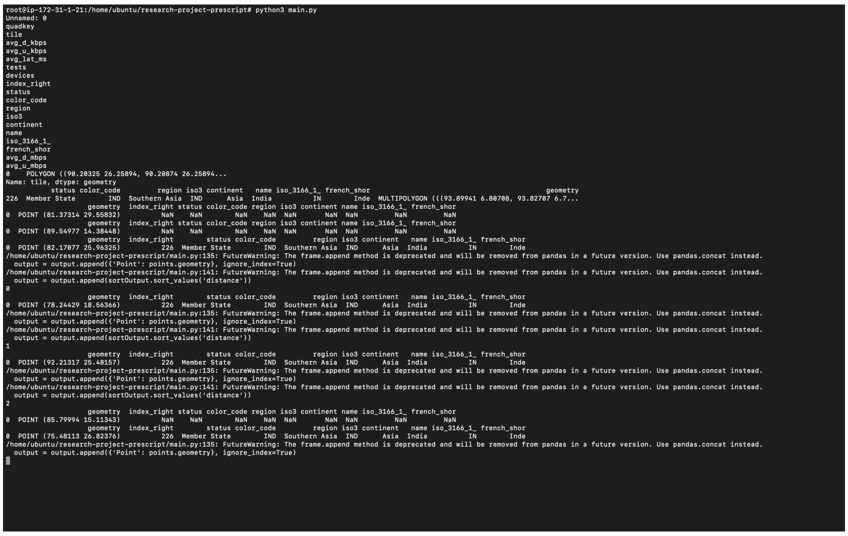

Figure 6: Prescript Logs after starting the app

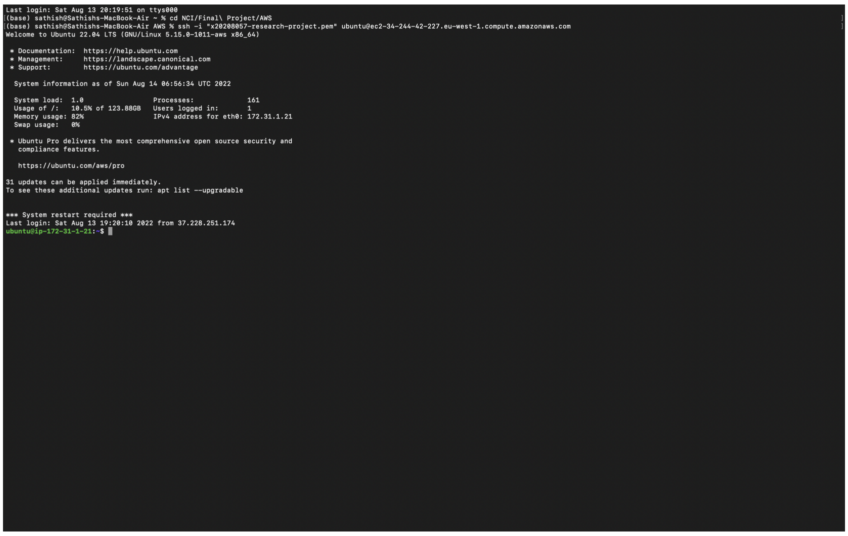

Figure 7: Prescript Memory Usage

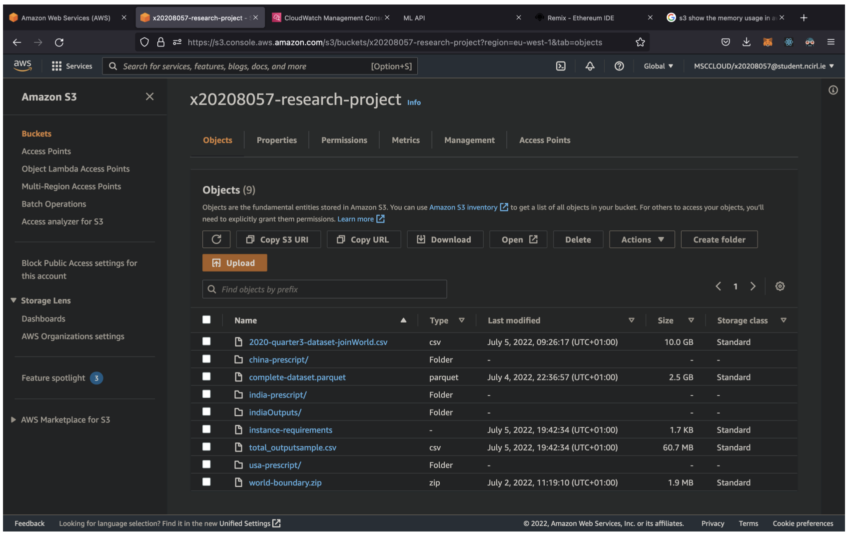

Figure 8: Prescript S3 Folders

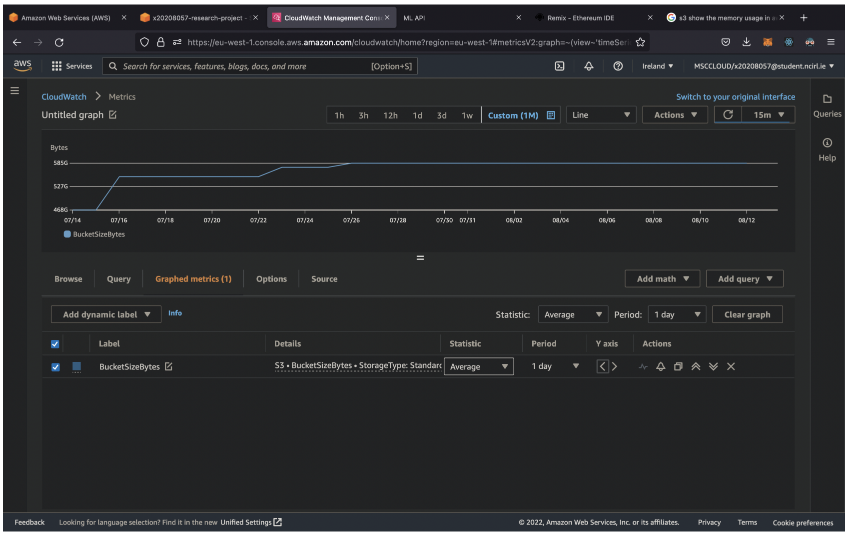

Figure 9: Prescript S3 Resources

1.1 Setup

The code can be setup in two ways:

1. Just run the AWS-Instance-Template Figure to pull this code from git and install all the dependencies.

2. Install Python3 from `https://www.python.org/downloads/`
pip by running curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py
python get-pip.py
pip install -r requirements.txt

To run the application type
python3 main.py

The country name can be changed to any and simulate the performance of them. Uncomment the "'upload_dataset_to_s3 method"' and run the main.py file again to upload the result to the S3 Bucket.

2 Simulator

This code will get the Prescript output as input and crunch those 1 TB datasets to produce a result in the form of a graph and table. The simulation is performed for data transfers of 1 GB, 10 GB, 100 GB, and 200 GB in the proposed system and traditional datacenters located 250, 500, and 1000 kilometres apart. The simulation is done for the top 3 countries based on their device count. From Figure 19, it is clear that India, USA, and China are the top 3 (the *test* attributed refers to how many speed tests are conducted through Ookla). The Simulator code will crunch the 500 GB pre-processed dataset and produce the desired result. Figure 12 shows how all the preprocessor output CSV files are loaded as 1 million records into the memory and passed to **generateFinalData** method. The preprocessor output CSV file looks like this Figure 20 and the generateFinalData method are shown in Figure 10, Figure 11 and Figure 12.

A deep analysis of India's 50K random points has been conducted. From the box plot of devices in Figure 14, it is clear that at any given point, eliminating the outliers, the user can have **450,000 devices**. **As per the system, each device contributes 256 MB. For any given point, the user could use 115 TB of cache at once.** The **Probability Distribution** also infers the same Figure 15, it also shows that the probability of having 100,000 devices is very high at any given point. The propagation delay of the system with respect to those 50K points and a traditional datacenter located at 1000 km is shown in Figure 16. The Ookla dataset joining with respect to Indian boundaries is shown in Figure 17 and in Figure 18, the visualisation of the random coordinate generator in the preprocessing script is shown.

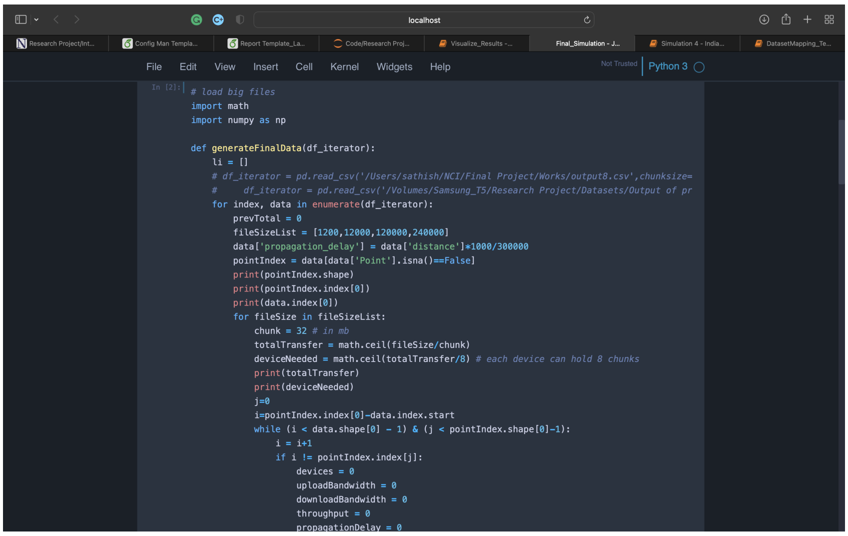

Figure 10: Simulator Code-1

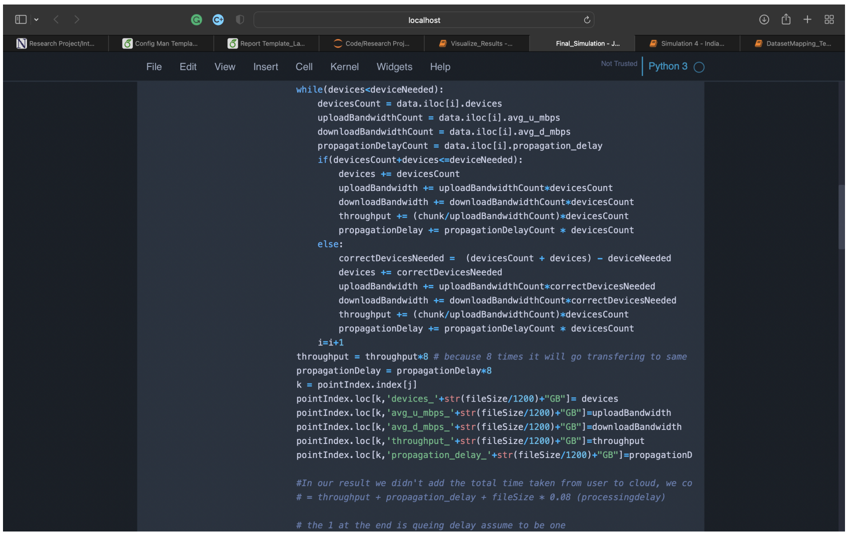

Figure 11: Simulator Code-2

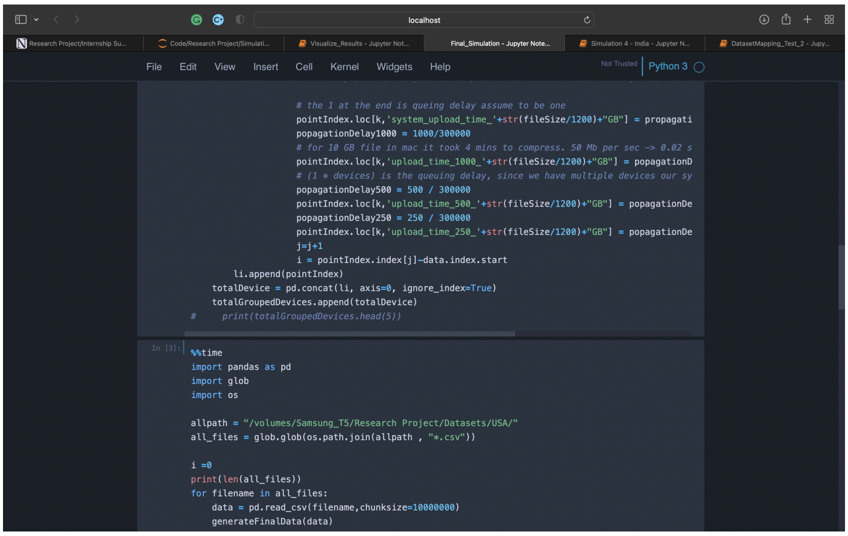

Figure 12: Simulator Code-3

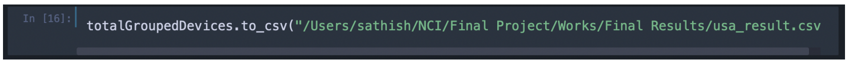

Figure 13: Export to CSV

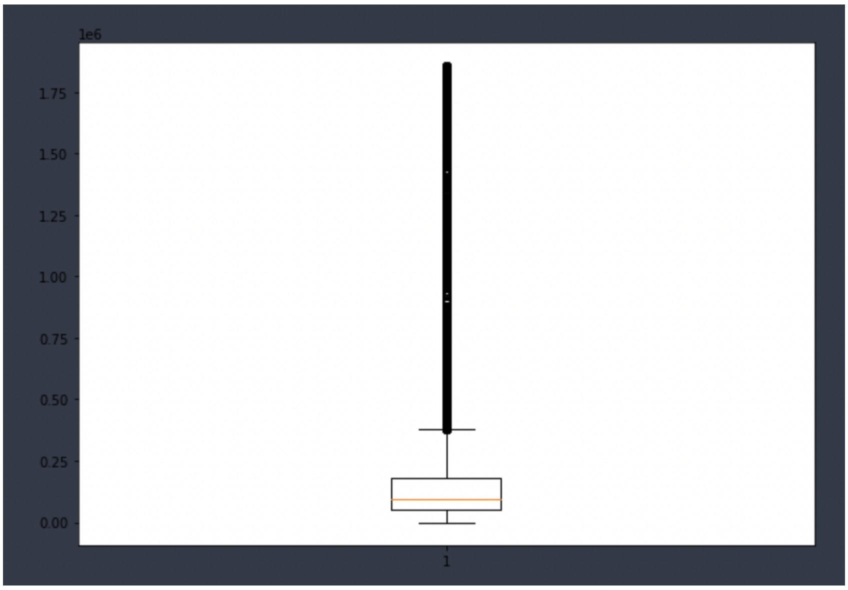

Figure 14: India Boxplot

```python
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline

# np.random.seed(42)
# x = np.random.normal(size=1000)

# plt.hist(x, density=True, bins=30)  # density=False would make counts
plt.hist(sortedTotalGroupedDevicesFinal['devices'], density=True, bins=70)
plt.ylabel('Probability')


plt.xlabel('Data');
```

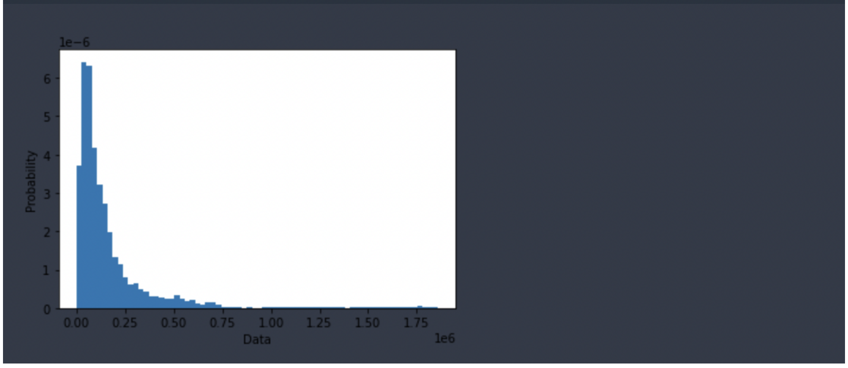

Figure 15: India Probability Distribution

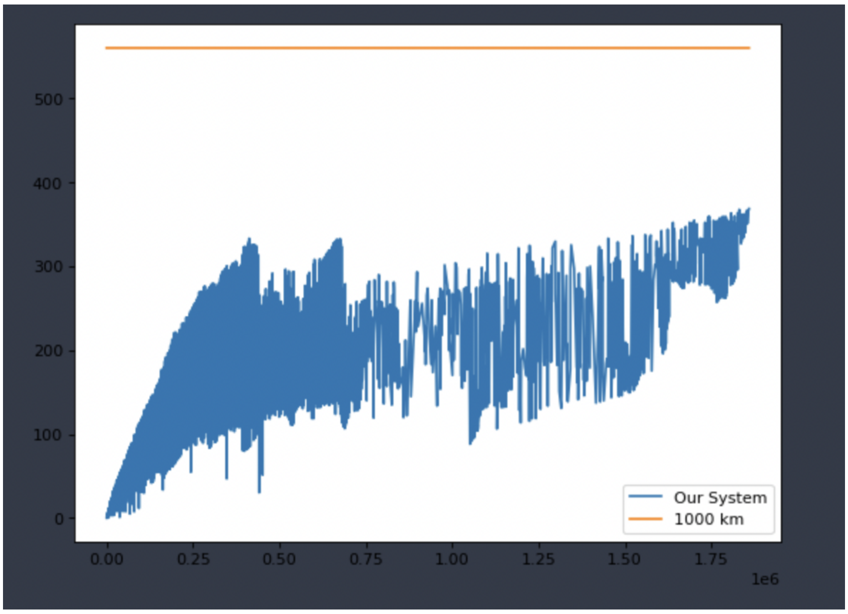

Figure 16: Propagation Delay Plot India

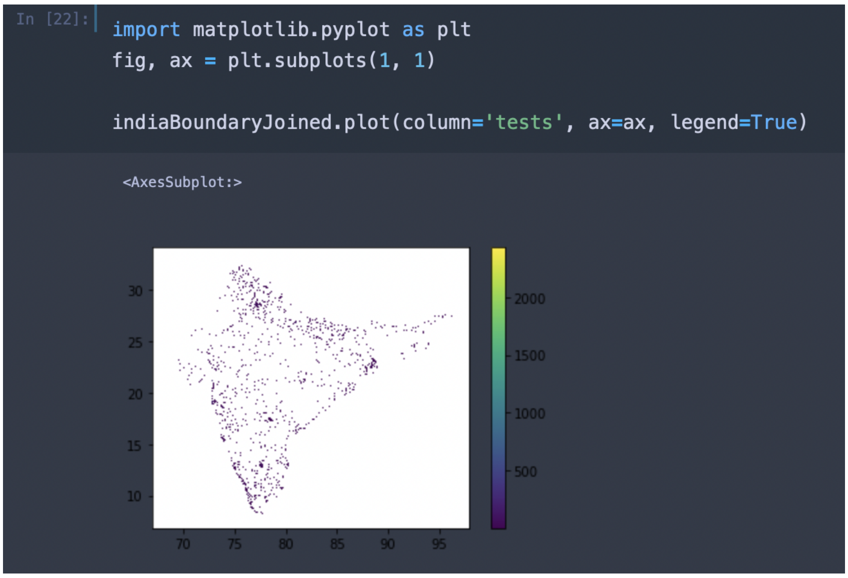

Figure 17: Mobile attributes dataset mapped to India

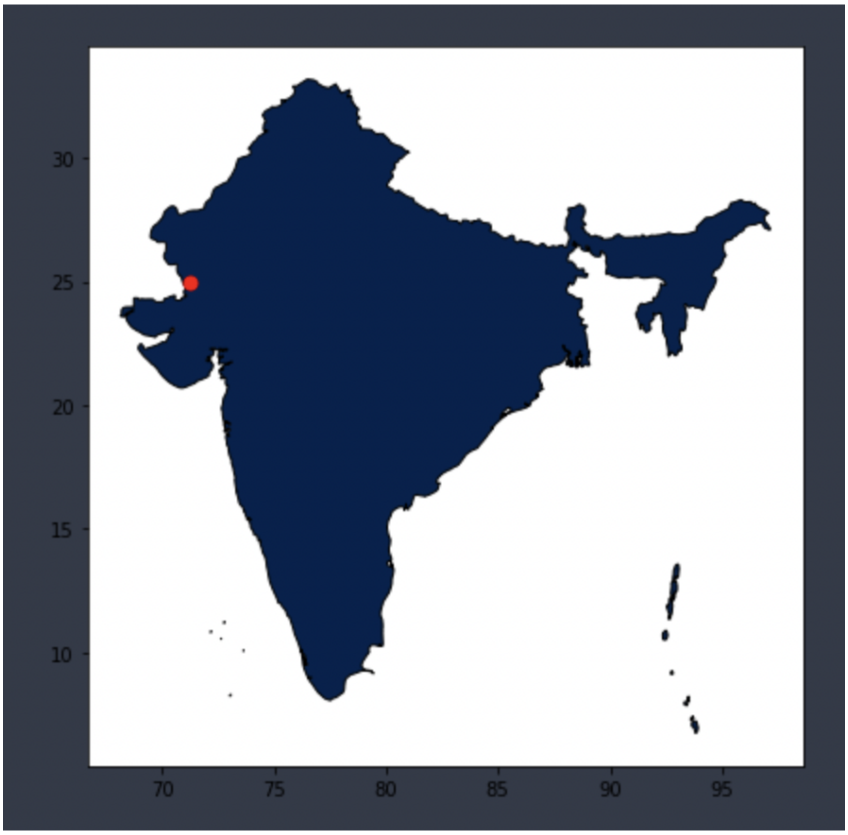

Figure 18: Visualisation of random point for India

```
In [14]: county_stats.sort_values(by=['tests'],ascending=False).head(5)
```

	name	avg_d_mbps_wt	avg_u_mbps_wt	tests
97	India	12.967881	5.563988	5492128
220	United States of America	75.898782	15.394886	3140723
43	China	206.796738	37.962229	1921009
98	Indonesia	16.369689	11.834148	1864908
174	Russian Federation	22.640738	10.614243	1657962

Figure 19: Top 5 countries from the dataset

instance2_output12

Point	index	Unnamed: 0	tile	avg_lat_ms	tests	devices	name	avg_d_mbps	avg_u_mbps	distance
0 POINT (-99.59215 33.92553) Name: geometry, dtype: geometry										
	32655875.0	32628822.0	POLYGON ((-99.5526123046875 34.0162418896	37.0	1.0	1.0	United States of America	105.534	4.5	0.09479367615383240
	6770140.0	6764692.0	POLYGON ((-99.5306396484375 34.0071350643	96.0	1.0	1.0	United States of America	0.49	0.082	0.09858798656259170
	21006339.0	20989060.0	POLYGON ((-99.51416015625 34.002581285433	56.0	1.0	1.0	United States of America	0.035	0.214	0.10647685625936000
	3937722.0	3934478.0	POLYGON ((-99.7174072265625 33.993472995	96.0	1.0	1.0	United States of America	5.663	2.19	0.135507329869915‌0
	18669048.0	18653742.0	POLYGON ((-99.656982421875 34.0526594213	61.0	1.0	1.0	United States of America	20.656	2.002	0.13618550940051700
	7820917.0	7814483.0	POLYGON ((-99.7283935546875 33.975253485	30.0	1.0	1.0	United States of America	46.939	6.894	0.13833541278584300
	9902032.0	9894039.0	POLYGON ((-99.722900390625 33.98891848376	94.0	16.0	2.0	United States of America	6.332	4.966	0.13838912780893200
	3528508.0	3525618.0	POLYGON ((-99.722900390625 33.98891848376	66.0	2.0	1.0	United States of America	1.266	1.52	0.13838912780893200
	25209786.0	25189065.0	POLYGON ((-99.722900390625 33.98891848376	142.0	6.0	1.0	United States of America	12.058	5.004	0.13838912780893200
	5344152.0	5339857.0	POLYGON ((-99.722900390625 33.9934729951	113.0	14.0	1.0	United States of America	4.96	0.489	0.14038598021285800
	15822106.0	15809112.0	POLYGON ((-99.7393798828125 33.947916898	29.0	1.0	1.0	United States of America	32.161	9.065	0.14285756978443800
	21076872.0	21059537.0	POLYGON ((-99.7283935546875 33.988918483	78.0	1.0	1.0	United States of America	0.21	0.72	0.14338021966227700
	3711865.0	3708820.0	POLYGON ((-99.7283935546875 33.988918483	29.0	1.0	1.0	United States of America	17.472	0.405	0.14338021966227700
	7936808.0	7930269.0	POLYGON ((-99.7283935546875 33.988918483	26.0	1.0	1.0	United States of America	5.016	0.263	0.14338021966227700
	25193844.0	25173140.0	POLYGON ((-99.7283935546875 33.988918483	125.0	1.0	1.0	United States of America	3.27	0.471	0.14338021966227700
	8225311.0	8218517.0	POLYGON ((-99.7393798828125 33.952473606	33.0	1.0	1.0	United States of America	32.948	3.09	0.14349749793627400
	3985949.0	3982666.0	POLYGON ((-99.7283935546875 33.993472995	36.0	1.0	1.0	United States of America	37.821	0.811	0.14530849987186200
	25179866.0	25159183.0	POLYGON ((-99.7283935546875 33.993472995	72.0	2.0	1.0	United States of America	8.203	0.96	0.14530849987186200
	19304737.0	19288902.0	POLYGON ((-99.73388671875 33.98436372829	110.0	1.0	1.0	United States of America	0.137	0.033	0.14666071778271600
	3894290.0	3891083.0	POLYGON ((-99.744873046875 33.9342453111	33.0	1.0	1.0	United States of America	8.198	1.852	0.14729259515513300
	8411348.0	8404397.0	POLYGON ((-99.744873046875 33.9433599465	31.0	1.0	1.0	United States of America	78.249	10.127	0.14783081384720400
	25161780.0	25141115.0	POLYGON ((-99.73388671875 33.988918483762	358.0	2.0	1.0	United States of America	0.564	0.216	0.14840678484464200
	7950166.0	7943622.0	POLYGON ((-99.7283935546875 34.0071350643	430.0	1.0	1.0	United States of America	1.567	0.531	0.15176726333962800
	16158644.0	16145349.0	POLYGON ((-99.7503662109375 33.929687627	26.0	1.0	1.0	United States of America	57.994	2.455	0.15272712942422100
	8165598.0	8158850.0	POLYGON ((-99.7503662109375 33.929687627	27.0	1.0	1.0	United States of America	57.68	1.158	0.15272712942422100
	15779029.0	15766069.0	POLYGON ((-99.7503662109375 33.934245311	33.0	1.0	1.0	United States of America	48.927	21.244	0.15278365126400200
	7879187.0	7872701.0	POLYGON ((-99.755859375 33.9205715286751,	28.0	1.0	1.0	United States of America	117.166	10.175	0.15829803849125300

Figure 20: Input to Simulator

2.1 Setup

Install anaconda from the following url `https://www.anaconda.com`.
Open Jupyter notebook and select the simulator code.
Click on "kernel" and then "restart & run all".

3 WebApp

The Webapp is used to calculate the performance of the proposed system at any given coordinate and file size. This web also creates a mock data about the peers to the DB which is running in an EC2 instance Figure 23. The mock data contains peers' names, location, device info, device network usage, device memory usage, etc. The rows in the transactions table are hashed and stored in a column named hash. This hash column is later used for calculating the merkle root hash by the DAPP.

The website screenshots are given in Figure 21, Figure 22. Once the user gives the lat, long, and filesize, the flask app will compute the number of devices. Consider that if it needs 'X' devices to transfer 'Y' MB of files then at MySQL 'X' number of mock users and 'X' number of mock devices will be created. The mock data of the userInfo table is shown in Figure 24 and mock data of deviceInfo is shown in Figure 25. The transaction table will also have mock data of how many device resources have been used by the system Figure 26. The DAPP will run every 1 hour and look for value "0" in the "isPicked" column. It will pick all the unpicked transactions and calculate the merkle root hash with the help of the "hash" column in the transaction table. Once the Merkle root is calculated, it is updated in the merkleRoot table Figure 29 and updates the isPicked column to 1 Figure 27.

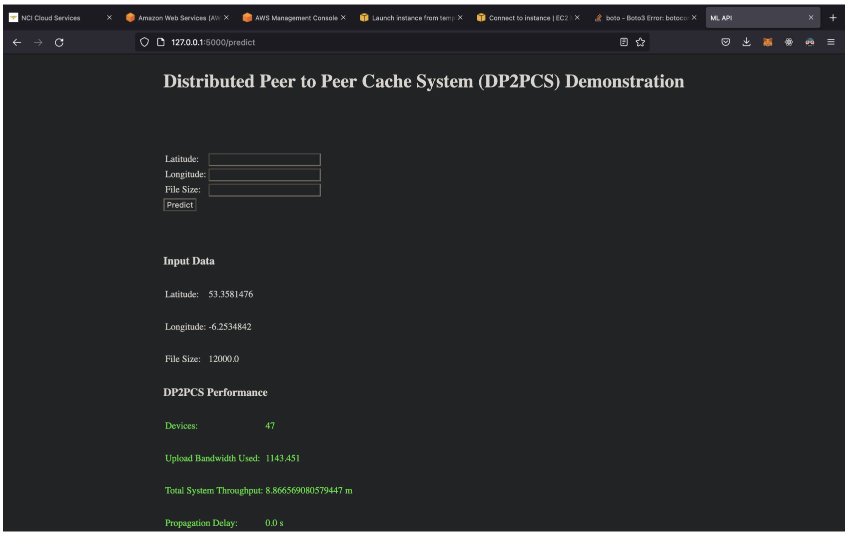

Figure 21: WebApp Screenshot-1

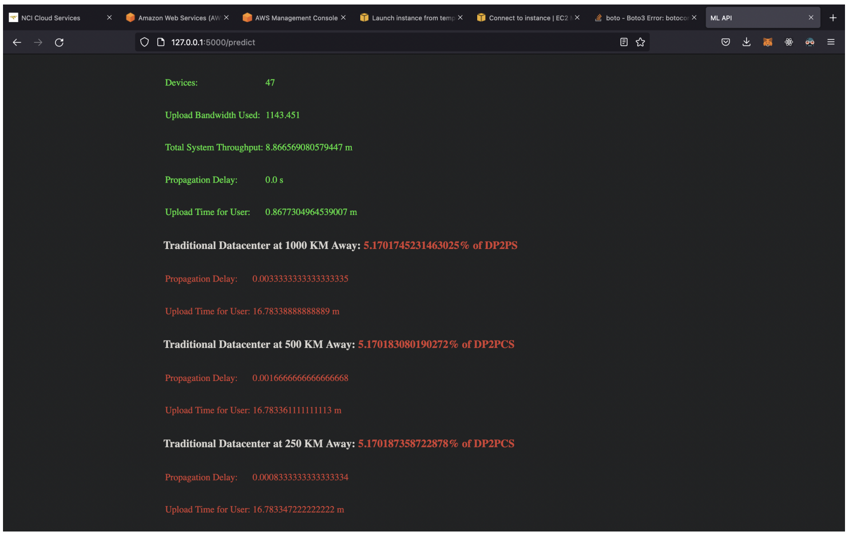

Figure 22: WebApp Screenshot-2

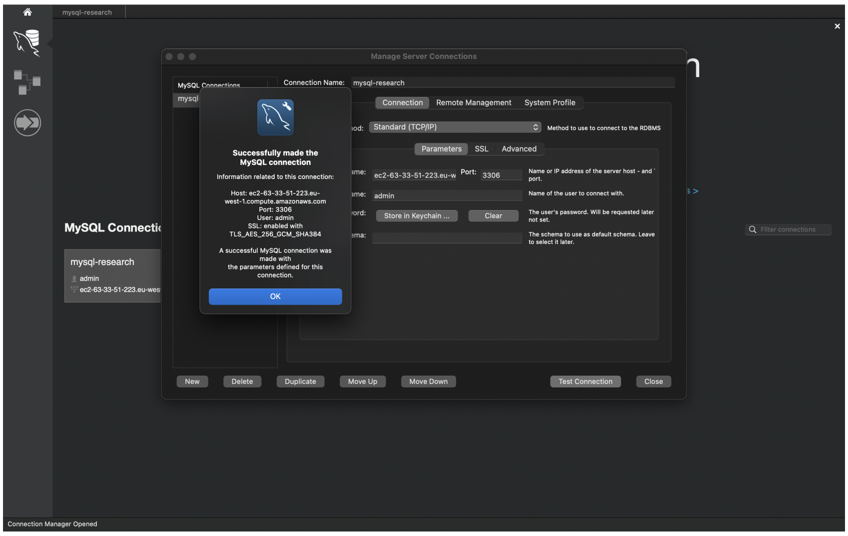

Figure 23: MySql Connection

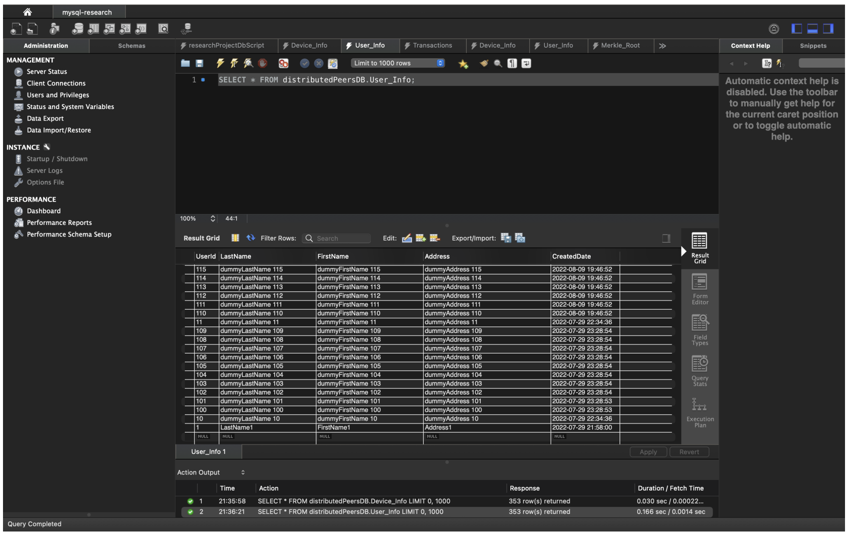

Figure 24: UserInfo Table

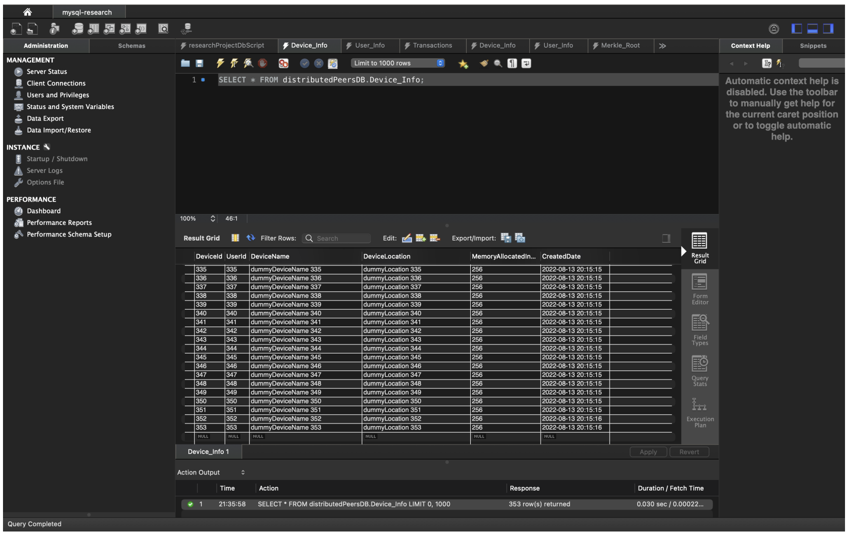

Figure 25: DeviceInfo Table

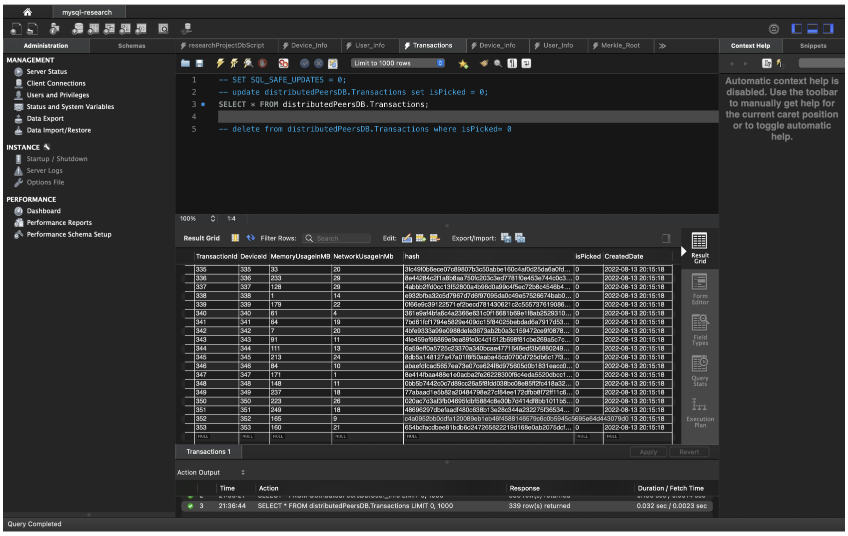

Figure 26: Transaction Table before DAPP update

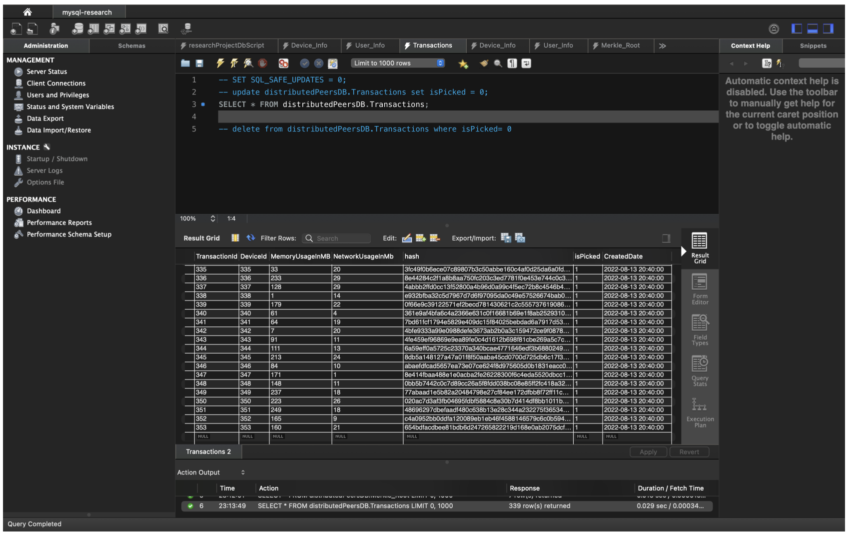

Figure 27: Transaction Table after DAPP update

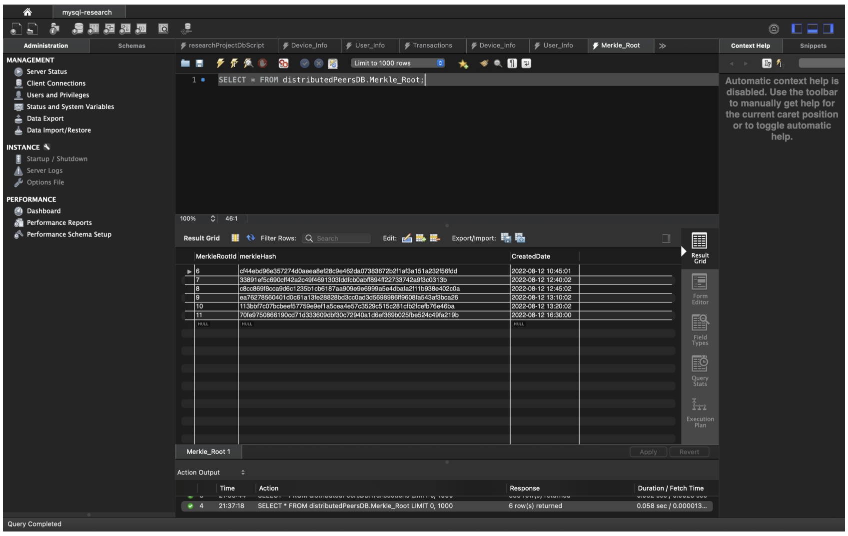

Figure 28: MerkleRoot table before DAPP update

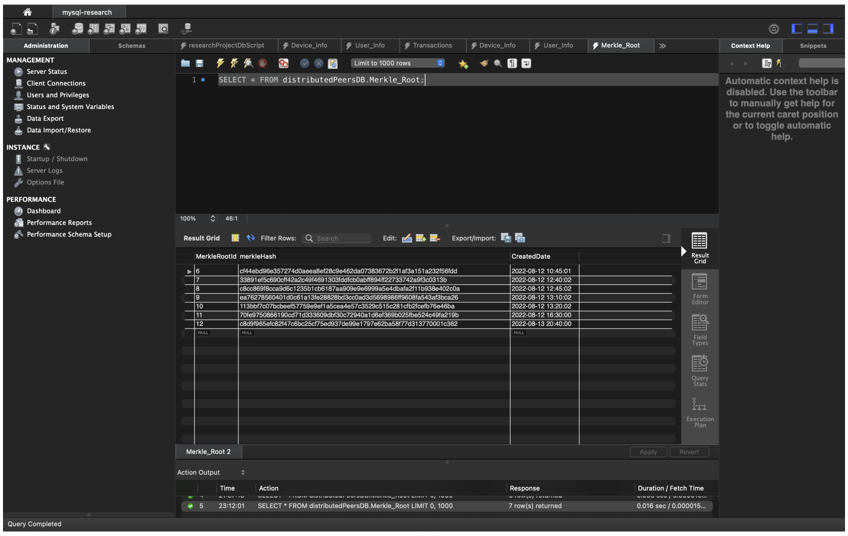

Figure 29: MerkleRoot table after DAPP update

3.1 Setup

Install Python3 from `https://www.python.org/downloads/`
Next install pip by running

curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py

python get-pip.py

Install requirements

pip install -r requirements.txt

To run the application type

export FLASK_APP=application flask run

4 Smart Contract

This is the smart contract code developed in Solidity and deployed using Remix. The smart contract will store the updated merkle root hash on the Ethereum public blockchain network. It also retrieves the top (recent) merkle root whenever the **challenge** method is called. With this top merkle root value, the SLA transparency is verified by the peers.

The smart contract code is given in Figure 30, and to deploy the smart contract on the Ethereum ropsten testnet, it must be deployed as an inject-provider metamask Figure 31 and it will cost some test ether from the metamask wallet as shown in Figure 32. Once the block is added, it will give the transaction id and contract id in metamask Figure 33. The transactions could also be seen in the Etherscan Figure 34.

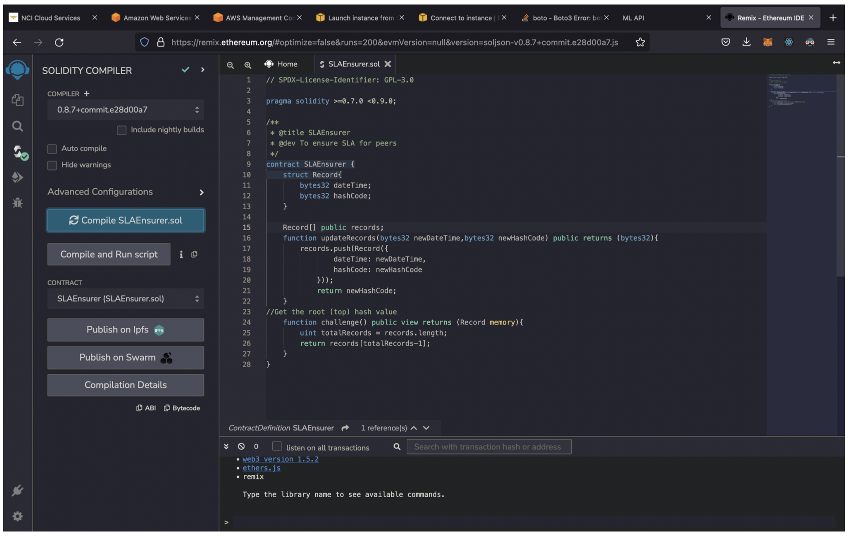

Figure 30: Smart Contract Compile

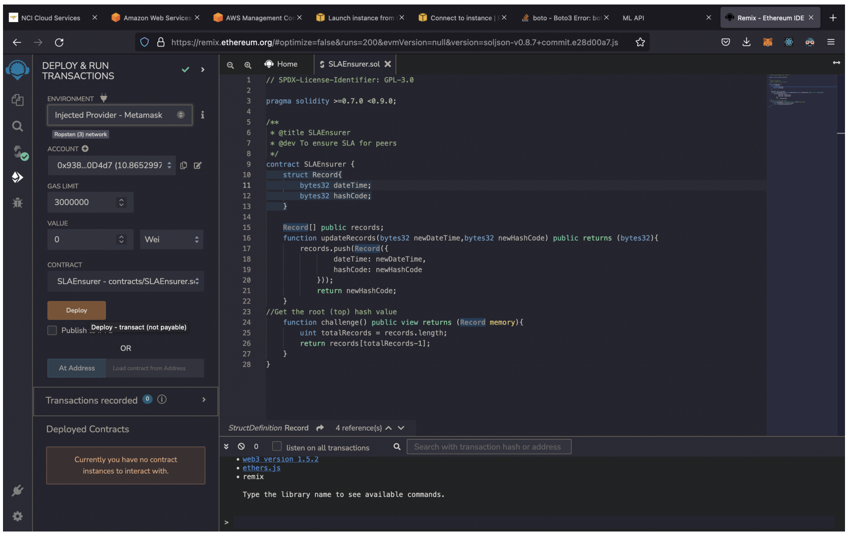

Figure 31: Smart Contract Deploy

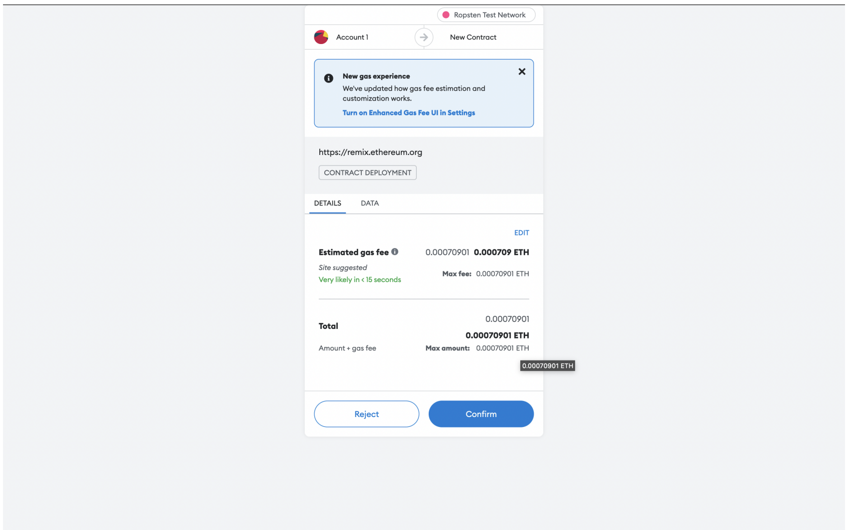

Figure 32: Metamask Gas Price

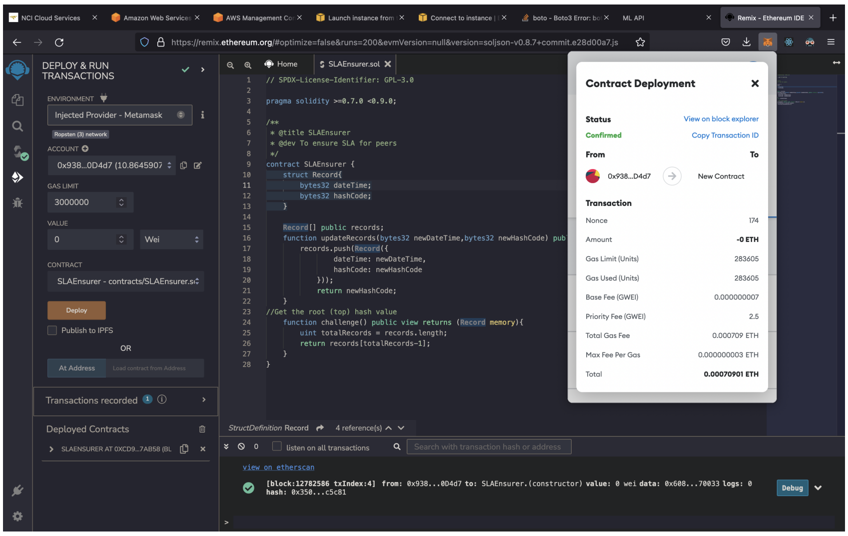

Figure 33: Metamask Contract Id

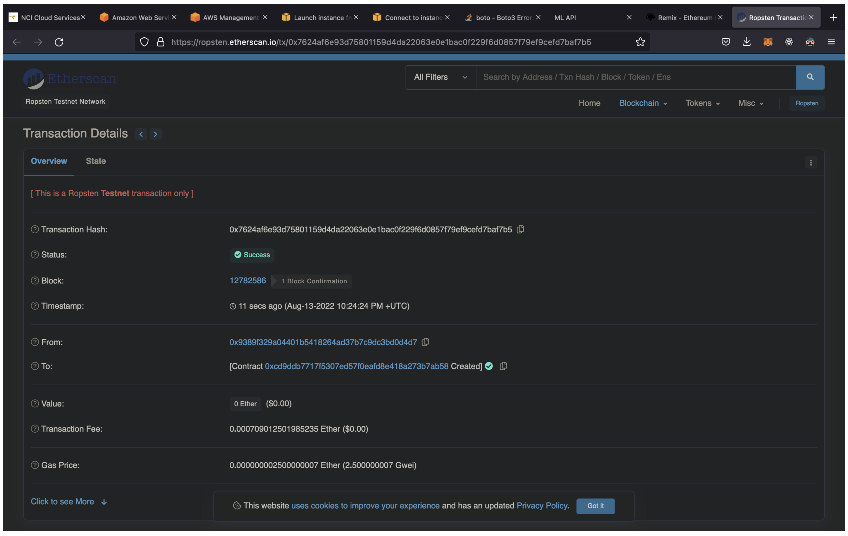

Figure 34: Etherscan transaction

4.1 Setup

Open Remix from `https://remix.ethereum.org`
Install Metamask from `https://metamask.io`
Get some free test ethers from `https://faucet.dimensions.network`
Upload the submitted solidity code to remix editor.
Compile the uploaded code.
Deploy it in **Inject Web3** and select **Ropsten testnet**.
This will deploy the code to Ethereum test network.

5 DAPP

The DAPP Figure 35 has a scheduler which runs every 1 hour and checks the MySQL DB for any new transactions with the help of **isPicked** column. If the isPicked value is 0, the rows are picked and the merkle root hash is computed by hashing all the hash values present in **hash** column of the transaction table. This merkle root hash is then updated in Merkle_Root table and Ethereum smart contract running in Ropsten test network Figure 36, Figure 37.

This DAPP also has two rest endpoint for accessing the root merkle value from both the Blockchain Figure38 and the DB Figure39. This REST end point will be handy for the peers to verify their SLA.

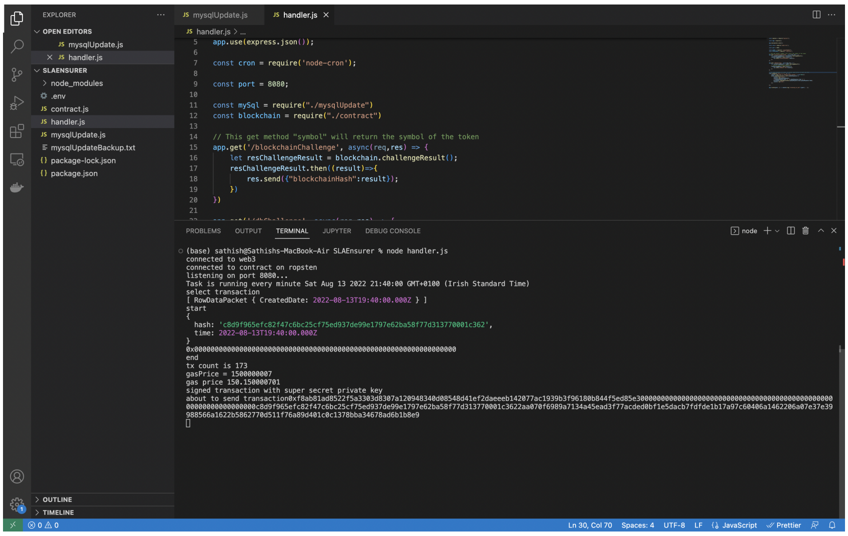

Figure 35: DAPP blockchain upload

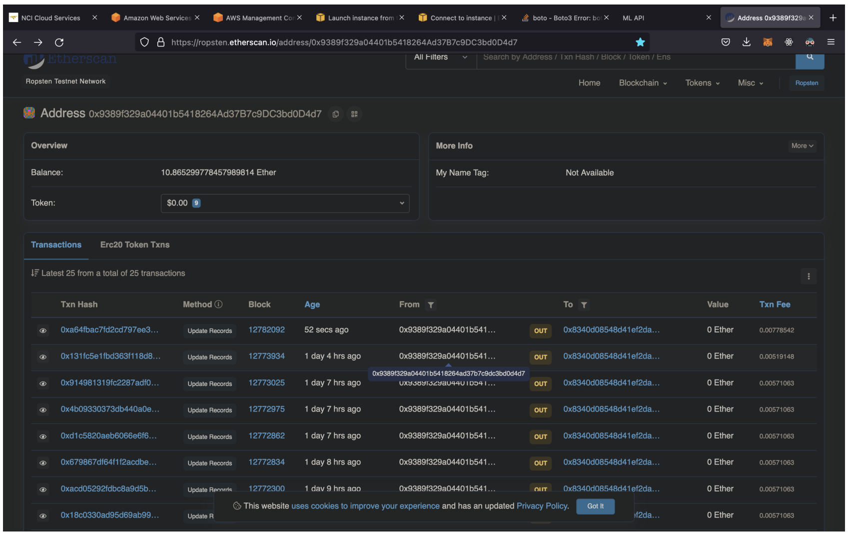

Figure 36: Etherscan update

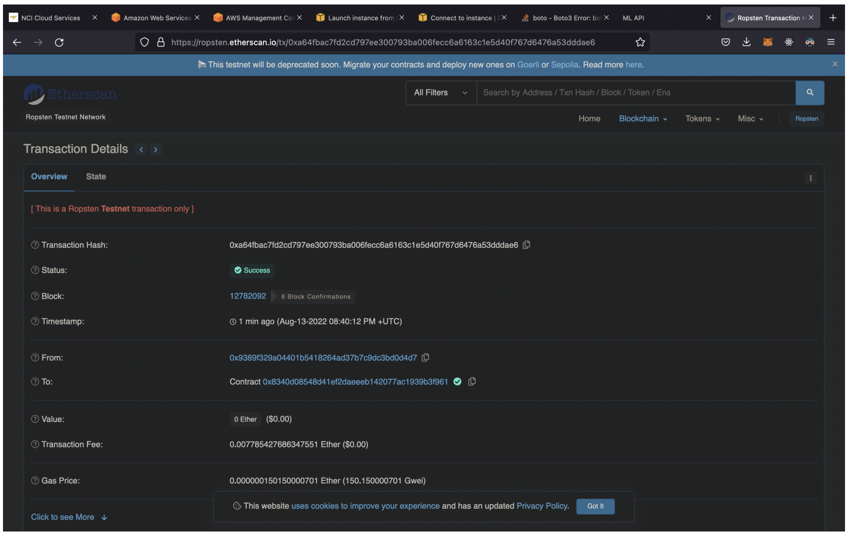

Figure 37: Etherscan transaction

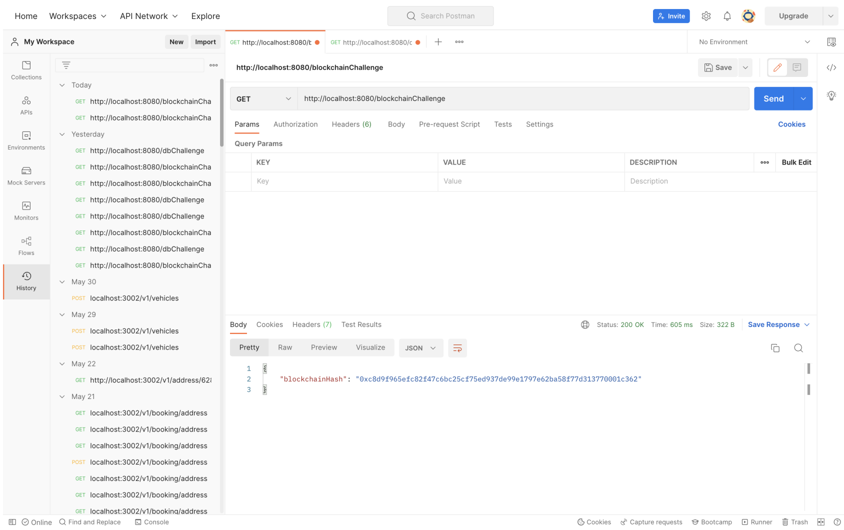

Figure 38: Rest call for Blockchain

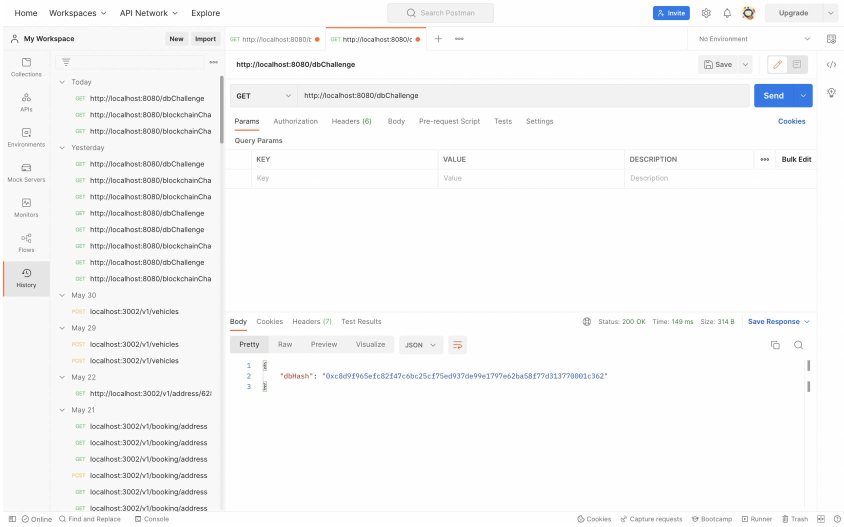

Figure 39: Rest call for DB

5.1 Setup

Install NodeJs from following link `https://nodejs.org/en/download/`
Open Terminal and cd into the project location, must be inside **SLAEnsurer**
Run the following command to start the application

npm install

node handler.js

Now to test the DAPP, install postman from the following link: `https://www.postman.com`
Make a "GET" request with the following URLS to see the root hash of both blockchain and DB.

`http://localhost:8080/blockchainChallenge`

`http://localhost:8080/dbChallenge`