# "P2P – Cloud-Based Tracker Assisted Peer-to-Peer Content Exchange Allocated Network"

***A***

***Project Report***

*submitted in partial fulfillment of the*

*requirements for the award of the degree of*

**BACHELOR OF TECHNOLOGY**

**in**

**COMPUTER SCIENCE**

**Specialization in**

**Cloud Computing and Virtualization Technology**

**By :**


| | | |
|---|---|---|
| | **Name:** | Kanwaljit Singh |
| | **SAP ID:** | 500044606 |
| | **Roll No:** | R110215062 |


***Under the guidance of***

**Mr. Vishal Kaushik**

**Assistant Professor (SG),**

**Department of Virtualization**



**Department of Virtualization**

**School of Computer Science,**

**UNIVERSITY OF PETROLEUM AND ENERGY STUDIES**

**Bidholi, Via Prem Nagar, Dehradun, Uttarakhand**


**2018-2019**

UPES

UNIVERSITY WITH A PURPOSE

# CANDIDATE'S DECLARATION

I/We hereby certify that the project work entitled **"P2P – Cloud-Based Tracker Assisted Peer-to-Peer Content Exchange Allocated Network"** in partial fulfilment of the requirements for the award of the Degree of BACHELOR OF TECHNOLOGY in COMPUTER SCIENCE AND ENGINEERING with specialization in Cloud Computing and Virtualization Technology) and submitted to the Department of Computer Science & Engineering at School of Computer Science Engineering, University of Petroleum & Energy Studies, Dehradun, is an authentic record of my/ our work carried out during a period from **January**, **2019** to **May**, **2019** under the supervision of **Mr. Vishal Kaushik, Assistant Professor (SG), SoCS**.

The matter presented in this project has not been submitted by me/ us for the award of any other degree of this or any other University.

**Kanwaljit Singh (62)**

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

Date: 10th May, 2019

**Mr. Vishal Kaushik**
(Project Guide)

**Prof. (Dr.) Deepshikha Bhargava**
Head
Department of Virtualization
School of Computer Science,
University of Petroleum & Energy Studies
Dehradun – 248 001 (Uttarakhand)

# ACKNOWLEDGEMENT

I wish to express my deep gratitude to my guide **Mr. Vishal Kaushik**, for all advice, encouragement and constant support he has given me throughout my project work. This work would not have been possible without his support and valuable suggestions.

I sincerely thank to our respected Program Head of the Department, **Prof.(Dr.) Deepshikha Bhargava**, for his great support in doing my project in **P2P** at **SoCS**.

I am also grateful to **Dr. Manish Prateek, Professor and Dean, SoCS, UPES** for giving me the necessary facilities to carry out my project work successfully.

I would like to thank all our **friends** for their help and constructive criticism during my project work. Finally, I have no words to express my sincere gratitude to my **parents** who have shown me this world and for every support they have given me.

| | |
|---|---|
| **Name** | **Kanwaljit Singh** |
| **Roll No.** | **R-110215062** |

# ABSTRACT

Today, it is seen that generally most part of the content gets served from one single server or cluster of servers which impacts the server performance and network impact of transferring large files. The services depend on the client-server architecture takes a lot of time without redundancy and charges a lot in infrastructure. The modern age in the field of technology has seen the rapid growth of internet and network dependent services such as evolved peer-to-peer-based technology in which the content distribution is not centralized one whereas it provides more high-availability through wide-scale replication of content divided into pieces at large numbers of connected computers. The main purpose of this project is to expel the issues like high content downloading time, and the content can be transferred with low packet loss with low infrastructure cost by developing a P2P cloud-based tracker assisted distributed network that searches for other connected computers, i.e. called "peers." Most content web servers are punished as all the transferring is done from one centralized server, a rich content website needs more assets like CPU and transfer speed to have the capacity to adapt.

With the utilization of P2P, the clients consequently reflect the content they require, facilitating the distributer's load. The project facilitates with DevOps approach in computerisation of the cloud servers and their system designs under which the cloud practices done on AWS environment with audit reviews utilising Jenkins builds and infrastructure setup utilising terraform as orchestration tool for deployment. Moreover, the peer application is accepting the peer inputs dynamically through suitable executable hosted on cloud according to the prerequisites required by the end-client system and catching them into remotely-enabled database and shows on web interface.

**Key words:** p2p – peer-to-peer, cloud-based tracker, swarm, low packet-loss, high-availability, low-bandwidth, distributed-approach, socket-technology, redundant network.

# TABLE OF CONTENTS

# LIST OF FIGURES

**S.No. FIGURES Page No.**

# 1. Introduction:

The major is Python-based shared (P2P) content exchange allocated network utilizing cloud-based tracker for maintaining the record of files and connected peers in transferring the large data without investing in infrastructure. Peer-to-Peer (P2P) technology permits the sharing of PC assets, for example, documents which further get divided into pieces for a end-to-end exchange between end-user systems. P2P system implies content is not put away on a unified server. Rather, client-side software, (for example, the famous BitTorrent) functions as a server for shared records on a person's PC while accessing their file and other peer log record on public trackers [1]. P2P's usage gains from its reliability, low configuration and low bandwidth utilization that has changed the content framework of content sharers to make content accessible to one another around the world. P2P system has the following attributes:

- End-to-End communication between connected systems.
- Each hub, for example essentially associated PC is considered as an equivalent to all others.
- There is no centralized purpose of control inside the system so low chances of failure.
- Connected Computer and file records are available at public trackers.

Peer-to-peer networking is the exchanging of PC assets and administrations by end-to-end communication between associated PCs. In a shared p2p framework, the content being disseminated is partitioned into sections called pieces. As each companion gets another bit of the document, it turns into a source (of that piece) for different friends, making the first source not to send that piece to each PC or client wishing a duplicate once more.

### 1.1. Types of Peer-To-Peer System:

**1.1.1. Unstructured peer-to-peer network**: It doesn't force any structure on the overlay systems. Companions in these systems interface in a specially appointed manner. In a perfect world, unstructured P2P frameworks would have definitely no incorporated framework [5], yet by and by there are a few sorts of unstructured frameworks with different level of centralization. There are two primary kinds of unstructured P2P systems:

- **Pure peer-to-peer systems**: The whole system comprises exclusively of equipotent peers. There is just a single steering layer, as there are no favored hubs.
- **Hybrid peer-to-peer systems**: These frameworks enable such foundation hubs to exist, regularly called super-nodes. An unadulterated P2P organize does not have the idea of

customers or servers but rather just equivalent companion hubs that at the same time work as both "customers" and "servers" to other hubs on the system.

**1.1.2. Centralized peer-to-peer network**: In this, centralized server is utilized for maintaining file and peer logs of the whole network. In spite of the fact that this has similitudes with an organized design, the associations between peers are not controlled by an algorithm.

### 1.2. What is a Socket?

Sockets allow communication between two different processes on the same or different machines. To be more precise, it's a way to talk to other computers using standard Unix file descriptors. In Unix, every I/O action is done by writing or reading a file descriptor. A file descriptor is just an integer associated with an open file and it can be a network connection, a text file, a terminal, or something else.

#### 1.2.1. Where is socket used?

Socket is used in a client-server application framework. A server is a process that performs some functions on request from a client. Most of the application-level protocols like FTP, SMTP, and POP3 make use of sockets to establish connection between the client and server and then for exchanging data. The socket was used to establish the connection with the server.

## 2. Literature Review:

P2P systems can extensively be delegated unstructured and organized P2P systems. In the previous, there is an irregular position of information on companions. Such systems develop organically as companions participate and there is no settled or chosen overlay which is clung to. In the last mentioned, there is a settled structure and overlay organize in which information put meant happens as indicated by focused hashing systems. Since P2P frameworks result in the conglomeration of assets from a few detached machines, this aggregation makes ready for an arrangement of an imposing archive of assets which also be used for specific purposes. For instance, the inert CPU intensity of each friend, in a medium-sized P2P organize, whenever used well, can result in the advancement of a framework with refined computational abilities. Since the companions in the system don't know about the entire topology of the system that they are a piece of, finding peers with the ideal substance effectively ends up troublesome. The strategy utilized customarily is that of message flooding or the guide from a brought together query server [3], [4].

## 2.1. Unstructured P2P Networks

Based on the instrument used to find the companion from which wanted information can be downloaded proficiently and the strategy used to download the equivalent then, we can partition unstructured P2P systems with a level engineering into the accompanying general classes: -

- **Napste**r: Napster is a P2P arrange which was worked so as to share music pieces among various friends in the system. There is an incorporated ordering server which tracks the substance transferred and the area of the companions which have downloaded the substance.
- **Gnutella**: Gnutella [4] varies from Napster in the way that there is no brought together ordering a server in the previous. The companions communicate question messages to their neighboring friends to discover in the event that they have a duplicate of the ideal substance. So as to avoid flooding of the system with question messages, a perused communicate of these messages is finished.
- **Freenet[3[**: In this P2P arrange there is no unified control and the companions find information dependent on substance steering. All the substance put away in the Freenet [14] is related to an exceptional key field, which is the hash of the substance.

## 2.2. Structured P2P Networks

In organized P2P systems, the topological properties of the overlay arrange and the tending to instrument help to fabricate the framework in which trade of substance can occur by means of various conventions. A few such systems have been proposed previously, for example, Chord [2], Pastry [5] and so on. The accentuation of the exploration is on making the query administrations productive. Every one of these conventions makes utilization of the Distributed Hash Tables (DHTs) for the without semantic, content-driven directing. Each friend is related with a peerID (as a rule the hash of the IP address) and every datum content is related with a KeyID (ordinarily the hash of the substance).

# 3. Problem Statement:

In existing systems, i.e. centralized systems, most of the part of files content is served from single server or bunch of servers which impacts the server performance in peak hours when a lot of user requests from the server at a same time and makes the service unavailable. In this server also needs heavy processing power and more storage in case of multimedia media which increases the infrastructure cost too.

Thus, we are developing a peer-to-peer (P2P) based system under which the content of the files is served from various connected computers, i.e. called peers by dividing the file into pieces, and each peer will receive a new and different piece of file, it will become a source for other peers in the network by relieving the original source not to send the file again. This provides the entire system a high availability with high file redundancy with wide-scale replication, and its resources cost will also be minimized when all the peers joined in the network and maintain it logs on the cloud-based tracker for peer coverage.

# 4. Objectives Achieved:

In this project, we've developed a P2P based content exchange distributed application with high-availability and redundancy utilizing an online cloud facility for indexing and log records for files which comprises of various objectives such as: -

- A Cloud-based public tracker which maintains the peer coverage and log records for indexing the content in a p2p network.
- A P2P distributed application which increases robustness by replicating data over multiple peers in-case of failures and provides resources including bandwidth, storage space, and computing power.
- An indexed search engine which provides available file statistics and peer availability statistics in real-time connectivity of computers.
- Auditing of Tracker Servers in One-Click Using Jenkins Build.
- Graphical User Interface for registering the clients (peer) for backup identity management via LAMP Stack.
- Client-side executable present in both .exe and binary that sign up peer info. without any platform dependencies.

# 5. Limitations:

- Peer authentication system is not established at present. Any user in same environment can send request to indexing tracker server.
- File transfer on public network makes tracker server unstable, currently works in localhost environment.

# 6. Design Methodology:

The model we took for our software project depends on our whole analysis, i.e. Evolutionary Prototype Software Development Process.

### 6.1. Tracker: Index Server

- Initialize: The server is located on the cloud's public infrastructure which will invokes its functions when some peer gets connected.
- Log index: It will maintain the log index of all the files available with their real-time statistics.
- Peer Coverage: It will update the connected computers in the peer coverage while displaying them on screen.
- Terminate: On termination, server will gets shut off.

### 6.2. Peer: Client Server

- Interface Module: Whenever client attempts to interface with the server it must give a legitimate I.P deliver to get to the server with port number. Based on server address, either the client is associated with the server or denied the entrance.
- Online-peers: Once the peer and server get effectively associated the following thing that is required for the framework is that server must demonstrate the rundown of the peers that are dynamic that time. It is essential since a portion of the peers might be down to the end-client must not continue getting to the disconnected peers.
- Find Module: Once the online peers are shown the interface must give the option to look through the content. Hunt record ought to have the usefulness that it must demonstrate the paths of all the framework that have that content as indicated by the client.

- Select Module: After posting the client should choose one peer that he/she finds the best and empowers downloading.
- Download Module: While the downloading begins, advance bar continues demonstrating that the content has been downloaded effectively or not.

## Evolutionary Prototype Software Development Process: -

- Analysis: Different ways of classical downloading will be studied. The p2p bandwidth coverage and algorithms will be studied out.
- Design: We will design and use various downloading algorithms that considers various scenarios in which data can be transferred with high availability and redundancy.
- Coding: Use of Python language in combination with sockets for data communication.
- Testing: Speed Tests will be done on the data packets to check the packet loss. The developed prototype of P2P is verified and validated upon the software testing methods (Unit and integration testing).
- Documentation: After completion of the full project, documentation will be done by the entire team.
- Return to Step 1: if need to implement a future extension to this project, chat functionality between users can possibly be added for further advancements in sectors like health, military, etc.

# 7. Implementation:

**Online Tracker Webhost: www.cloudtracker.tk**

**Auto-Scaling**: Auto-scaling is a way to automatically scale up or down the number on compute resources that are being allocated to your application based on its needs at any given time.

We have configured autoscaling over the AWS and configure minimum 2 and Maximum 4 instance. If our application needs more computing power, you now have the ability to launch additional compute resources on-demand and use them for as long as you want, and then terminate them when they are no longer needed. Adding and Removing instances action are based upon the below CloudWatch

**(Alarm 1)**

Condition: CPU Utilization >= 80 for 3 datapoints within 5 minutes
Description: This alarm continuously check the server to see If any server or endpoint consume greater than and equal 80% CPU utilization the resultant will launch or add one new instance.

**(Alarm 2)**

Condition: CPU Utilization < 40 for 3 datapoints within 5 minutes
Description: This alarm continuously check the server to see If any server or endpoint consume less than 40% CPU utilization the resultant will remove the one instance.

**There are two main components in the peer to peer architecture:**

1. Cloud Indexing Server (centralizd_cloud-server.py)

2. Peer (peer with its sub-modules, client-side)

**Socket Connection Establish and Binding with IPV4** :

- Define the IP address and port number of host-server.
- Initialize the socket method structure.
- Create a TCP server socket using socket() method – Bind the server socket to server IP and Port number. (this is the port to which we will connect)
- Create a new connection from client using connect( ) method system call.

- Send/ receive data with client using the client socket.
- Close the connection on close and accept methods

**Cloud Indexing Server:**

- Manages peer registration.
- Manages file index.
- Manages Client requests for searching index.
- Manages Peer list in the network that are connected to the server.

**Peer – Client Side:**

Peer has three sub components:

1) **Peer (peer.py)**

- Serves as client for users using the peer.
- Gives option for Listing and Searching files from the Central Indexing Server.
- Initiates connection to Central server and registers to the network.
- Starts the peer server which will serve other peers (This is Daemonized).
- Starts the file system handler, which updates Central Server about the files it has.
- Initiates file transfer upon client request.

2) **File System Event Handler (FilesystemEventHandler.py)**

- This is a daemon thread that is spawned by the peer thread.
- This constantly monitors the allocated directory for file updates (Addition and Deletion).
- Upon any such event, it automatically updates the changes to the Central Indexing server.

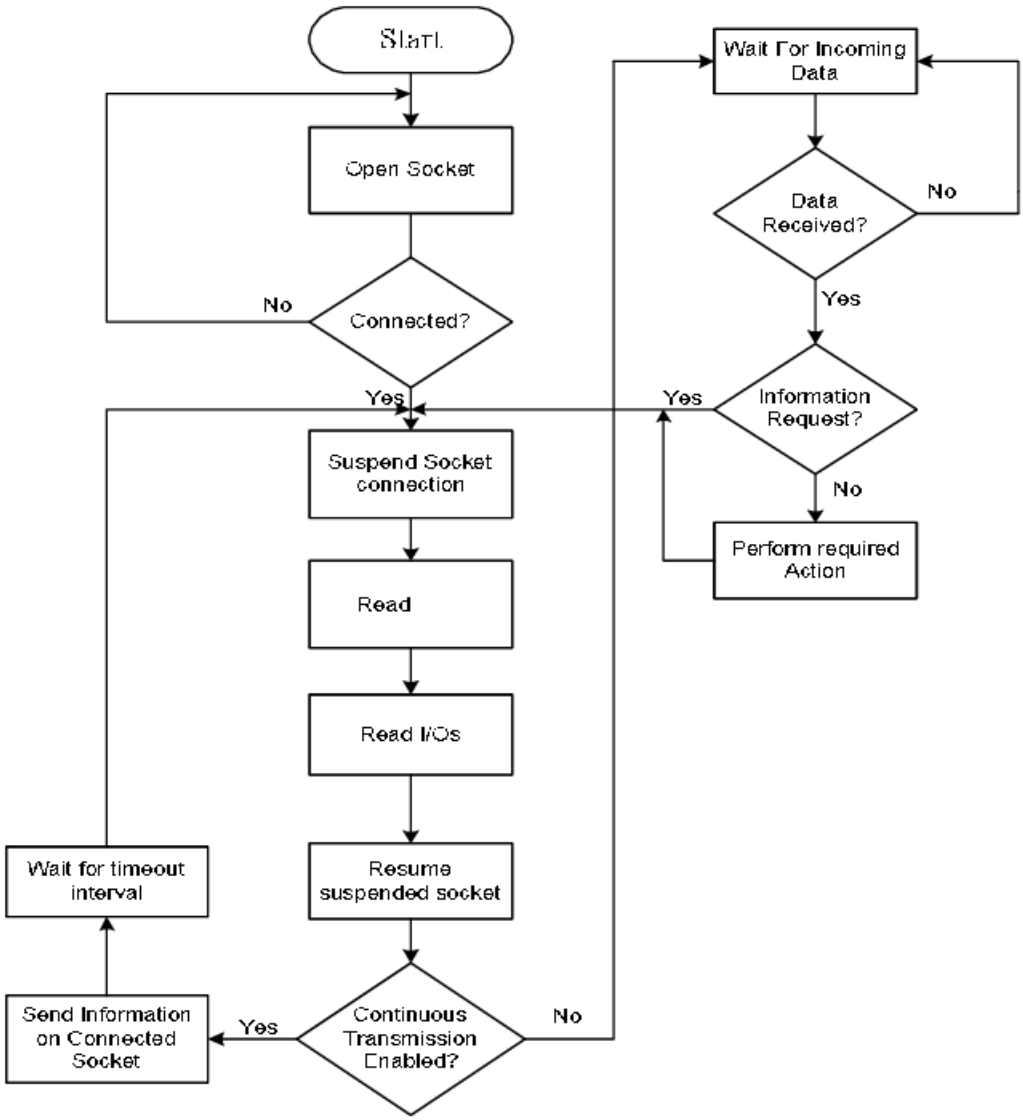
3) **Peer Server (server.py)**

- This is also a daemon server; this runs on the port which the Central Server allocates.
- This will listen to any peer requests and initiates file transfer.

**DevOps Approach:**

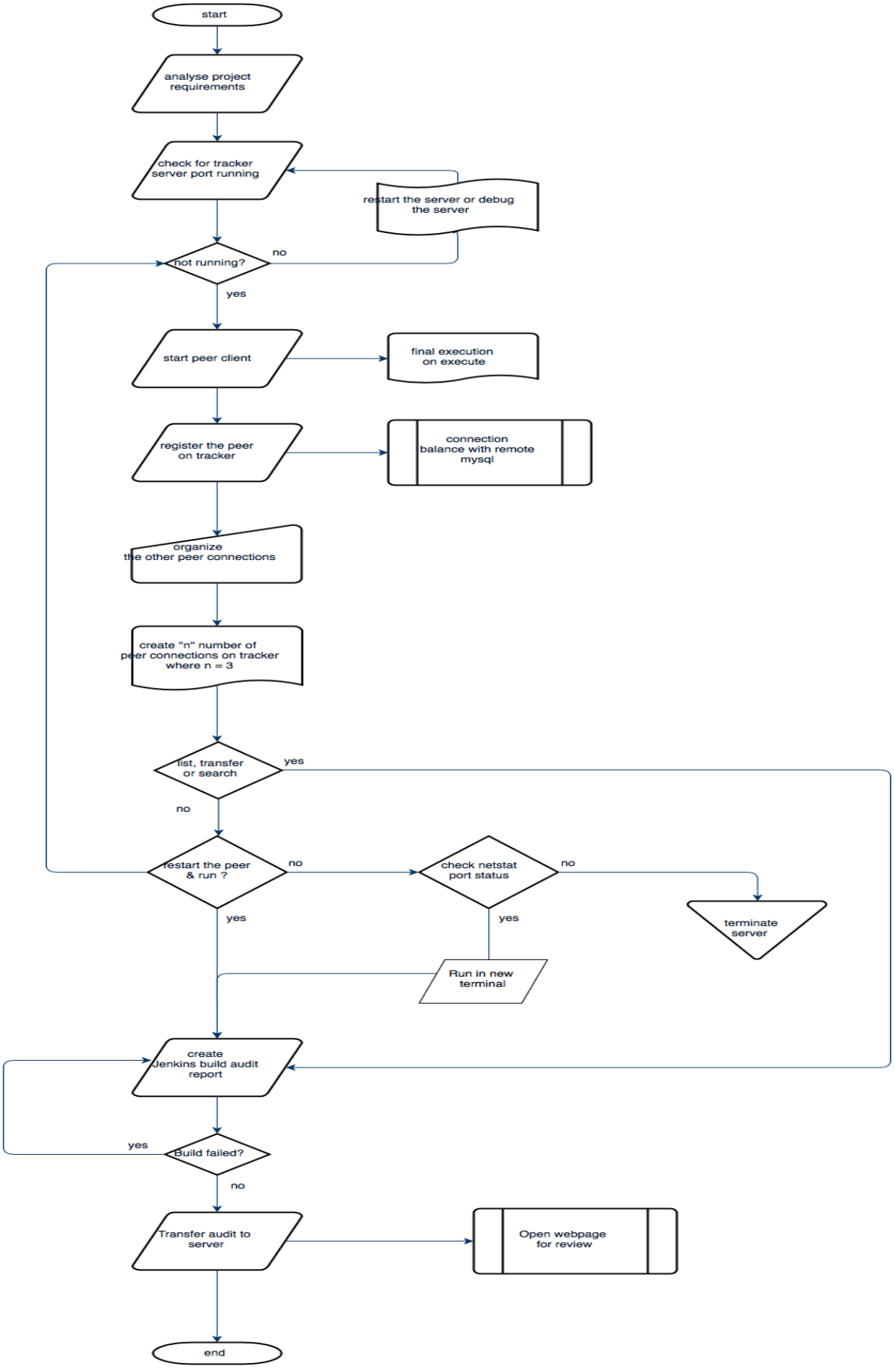
- Clones the code from develop branch from source's github repository.
- Making the master branches protective .
- Make test and Rel branch for release and feature updates.
- Test the codes and deploys it on test server using Jenkins build on successful development at developer server.
- Then cuts the Rel branch from code so that test code remains preserve just like develop code.
- On main release, merge the Rel branch into master to make it on live .

## 7.1. Flow Chart:



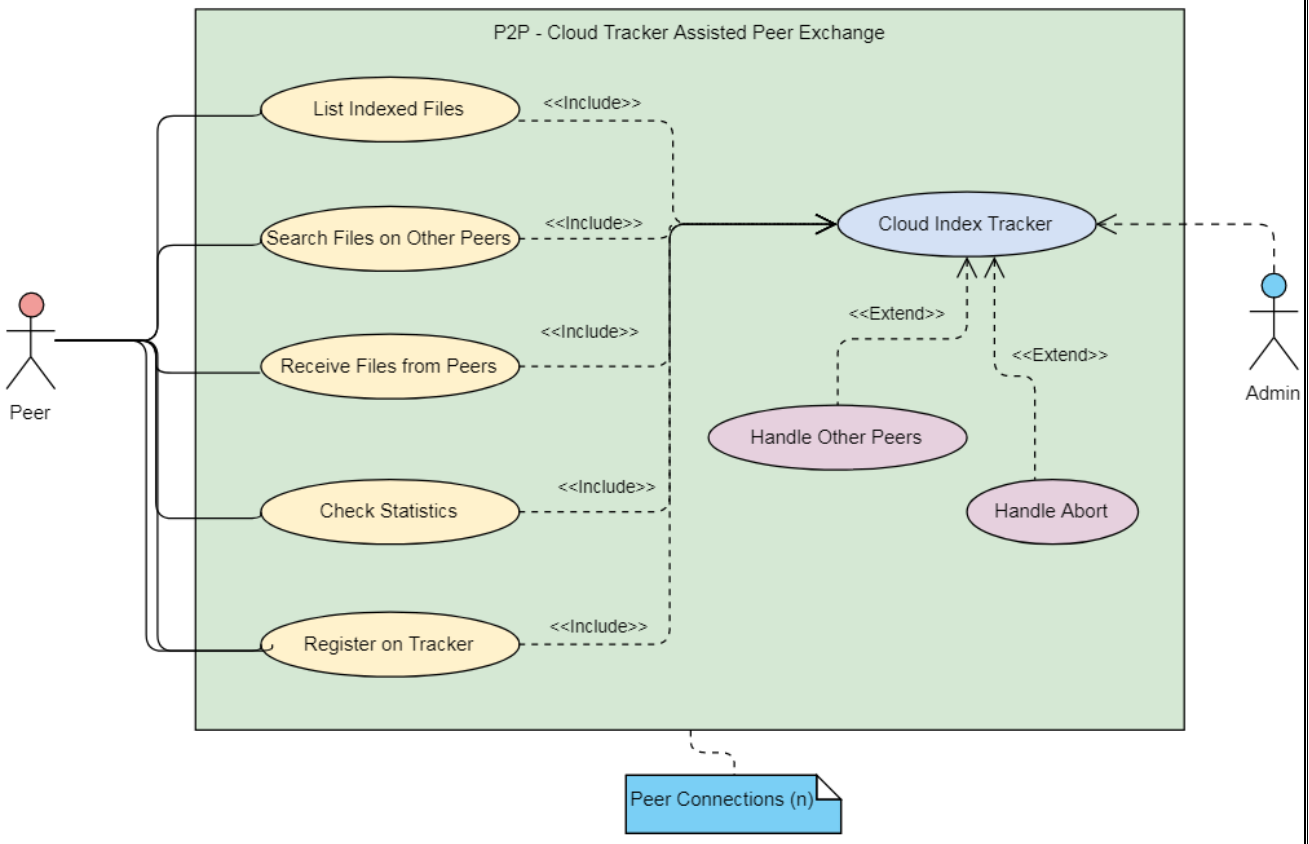

P2P - Socket Flow Chart - Cloud Index Tracker

**(Figure 7.1.1 – Flow Chart for socket connections b/w tracker server and peer client)**



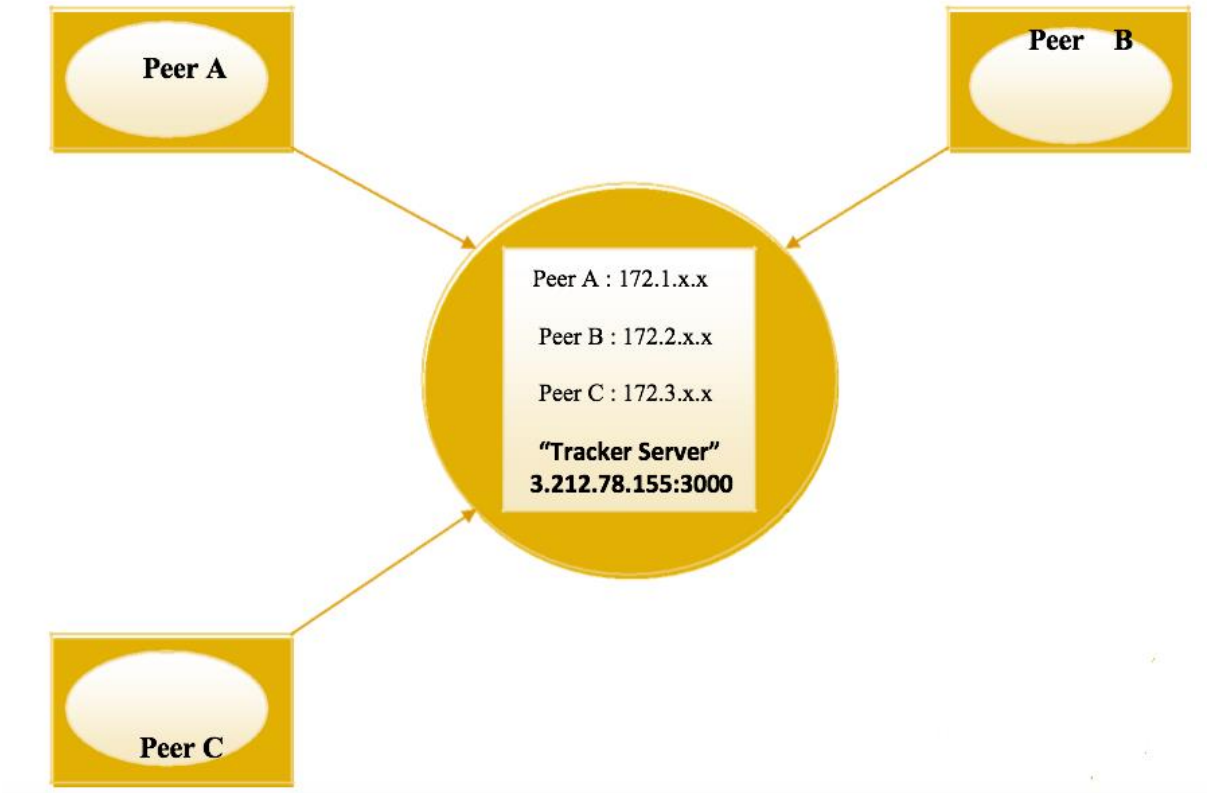

**(Figure 7.1.2 – Flow Chart for workflow of entire project with audit build generation)**

## 7.2. Use Case Diagram:

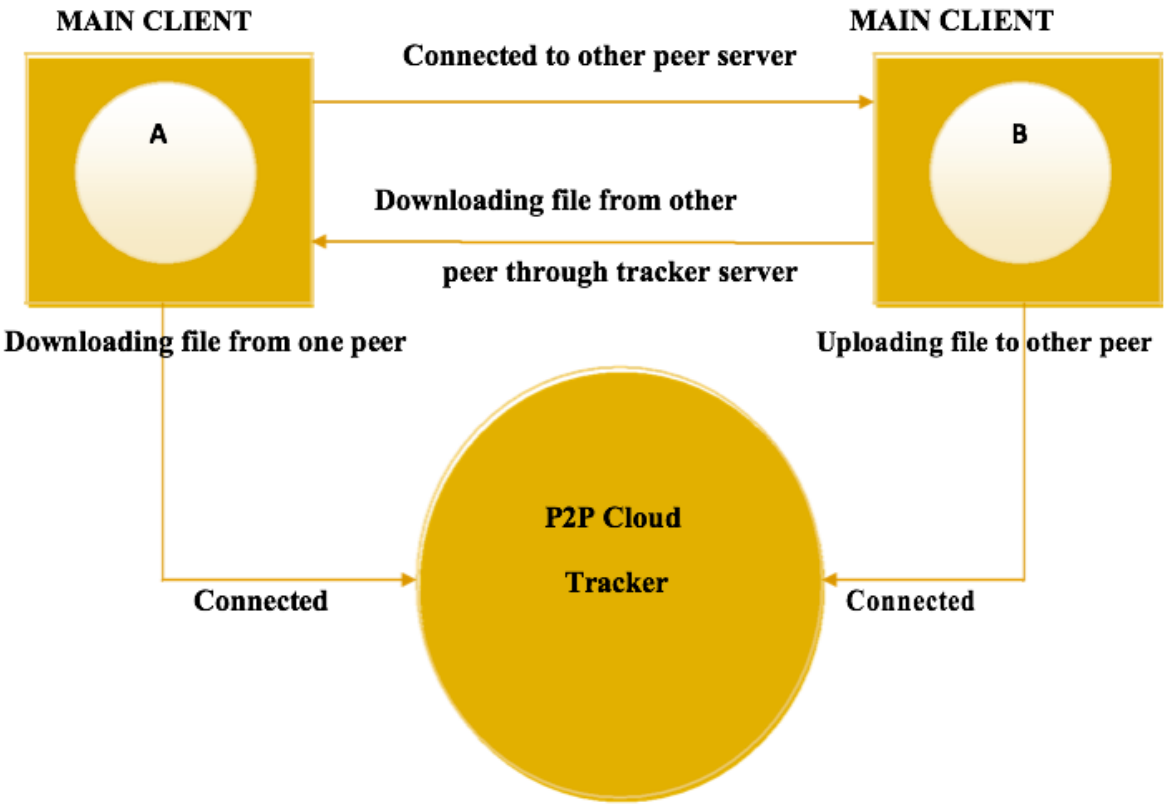



**(Figure 7.2 – Use Case Diagram for P2P Based Exchange Network via Tracker)**

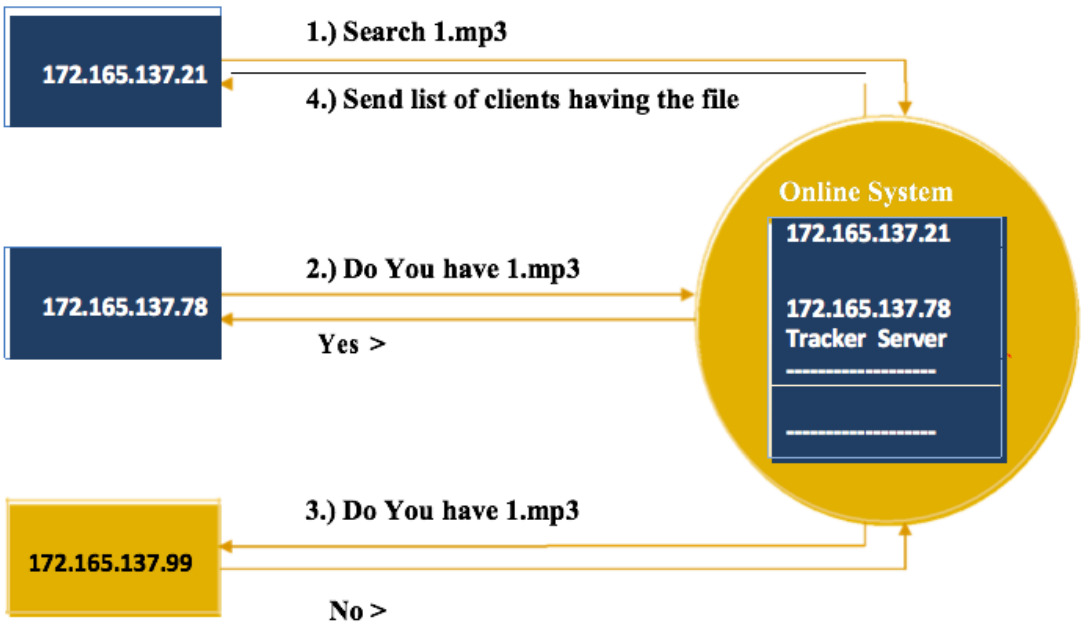
## 7.3. Result Visualization:




**(Figure 7.3.1 – Tracker Server making connections with n number of peers, here n = 3)**

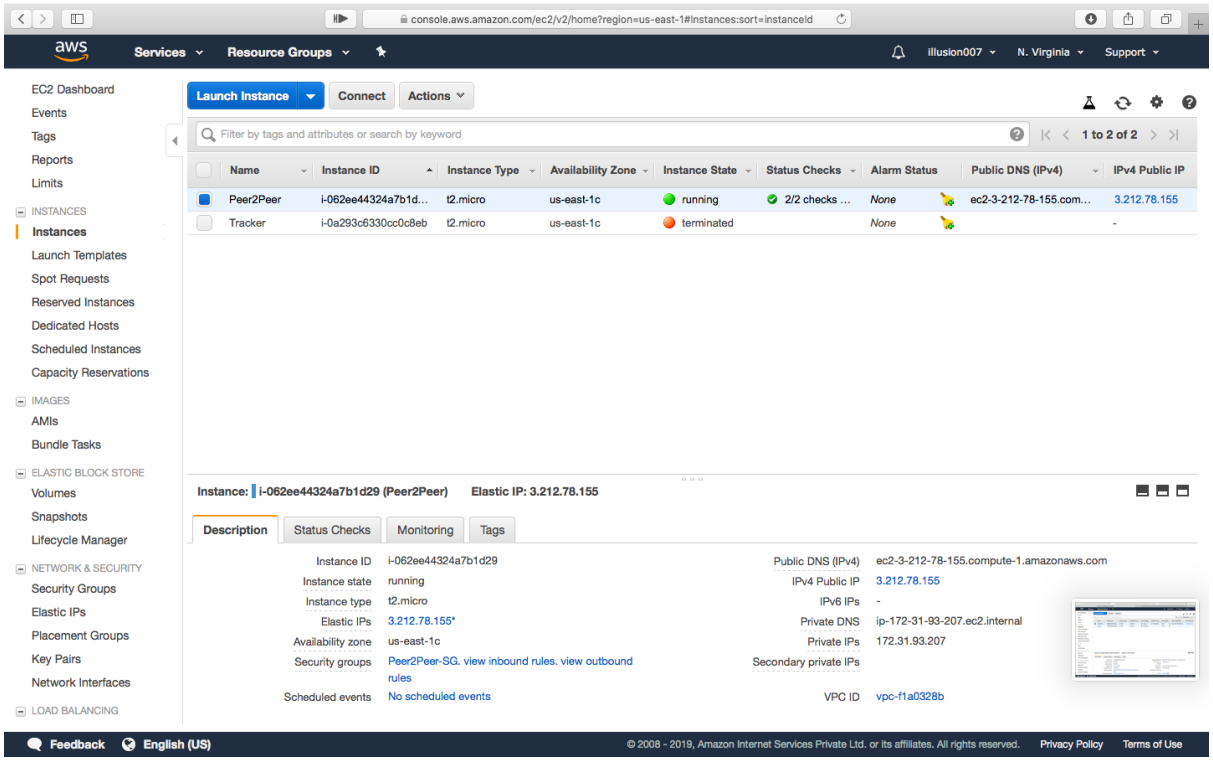



**(Figure 7.3.2 – P2P Tracker works as intermediate & clients transferring files b/w each)**




**(Figure 7.3.3 – Tracker server search about file from connected peers for initial query)**

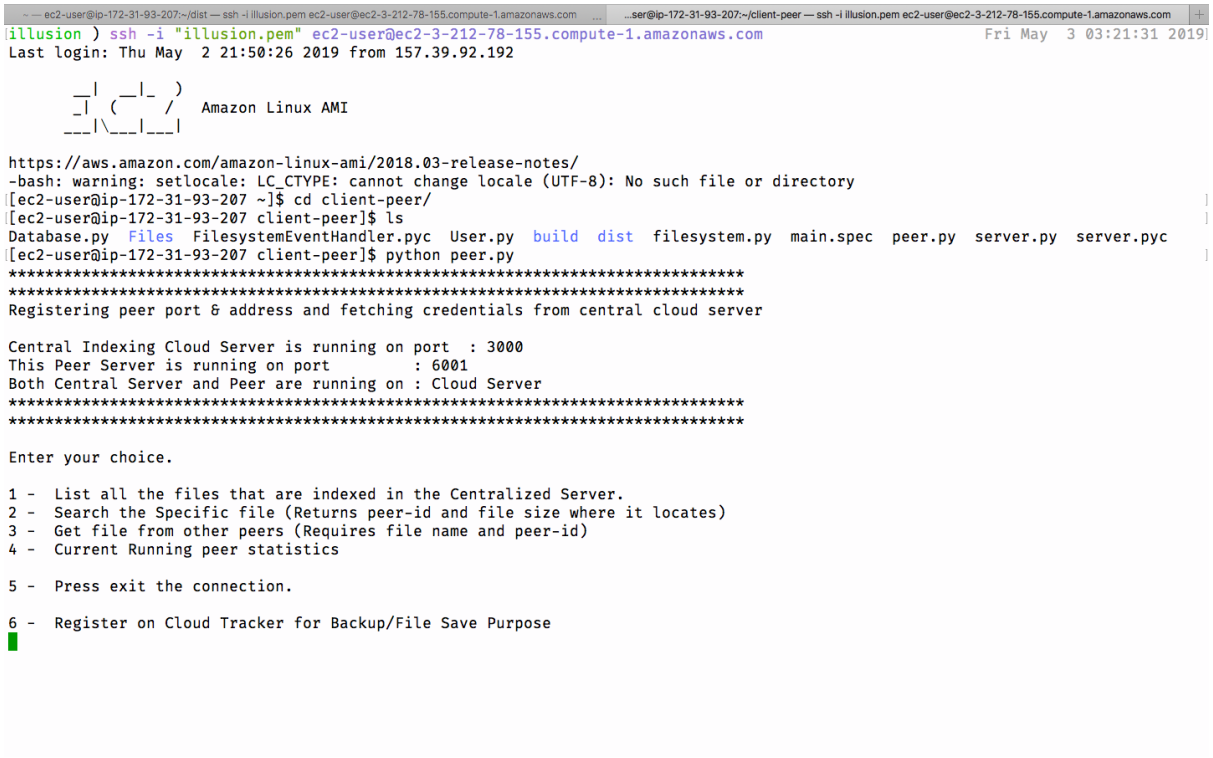
# 8. Output Screen



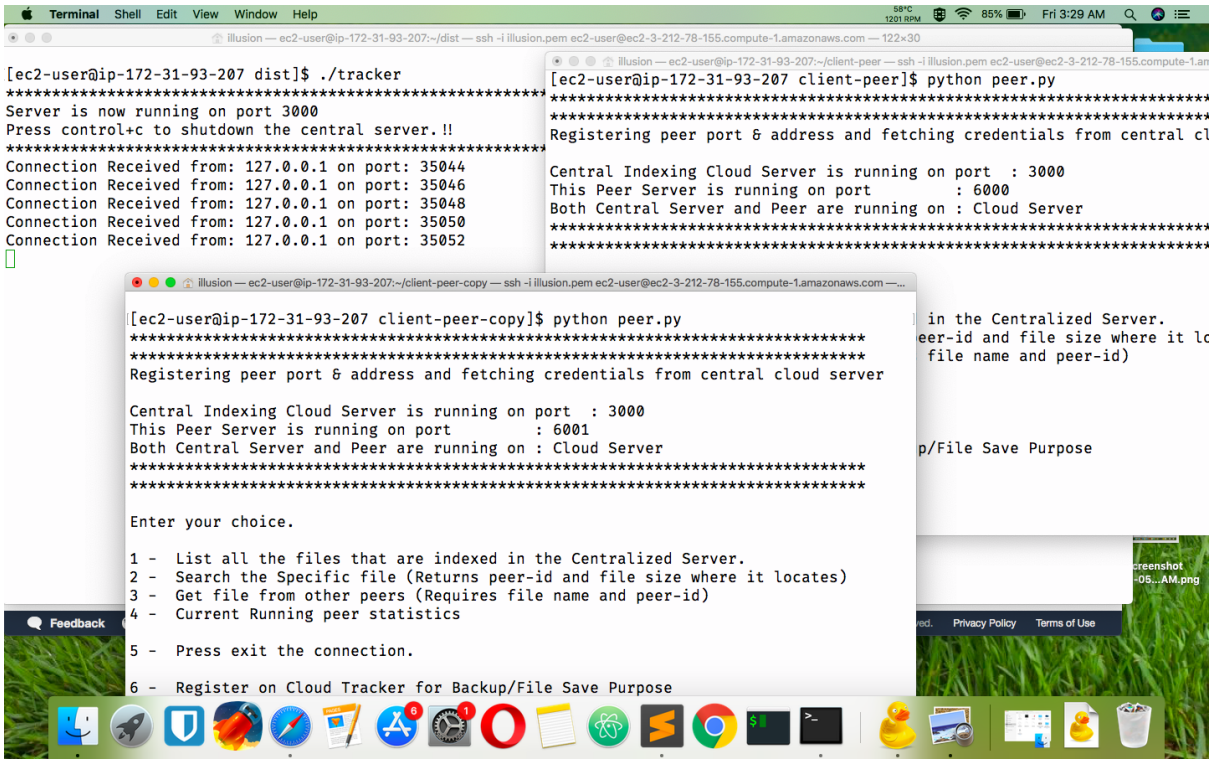

**(Figure 8.1 – AWS Cloud Instance Console for Managing Indexing tracker Server)**

```
[ec2-user@ip-172-31-93-207 ~]$ ls
build  central-cloud_tracker.py  client-peer  dist  tracker.spec
[ec2-user@ip-172-31-93-207 ~]$ cd dist
[ec2-user@ip-172-31-93-207 dist]$ ls
tracker
[ec2-user@ip-172-31-93-207 dist]$ ./tracker
********************************************************************************
Server is now running on port 3000
Press control+c to shutdown the central server.!!
********************************************************************************
```

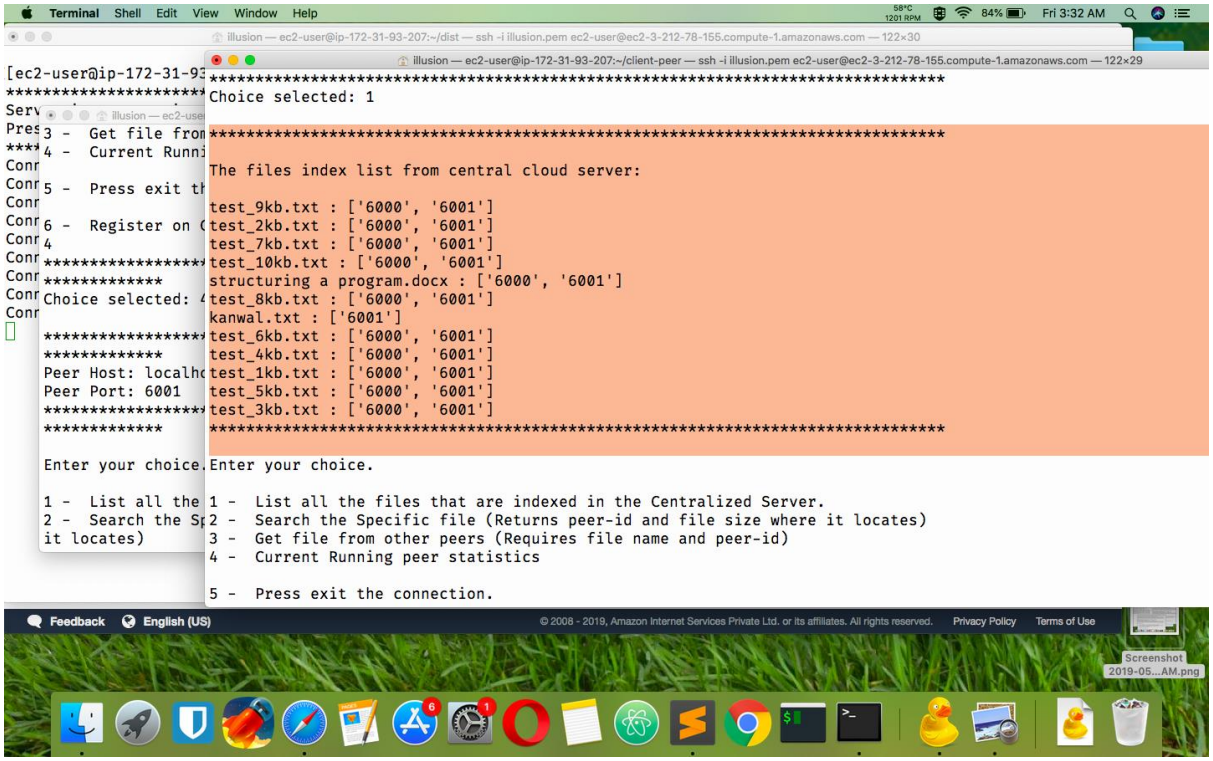
**(Figure 8.2 – Connection to Cloud Based Indexing tracker Server on Port 3000)**



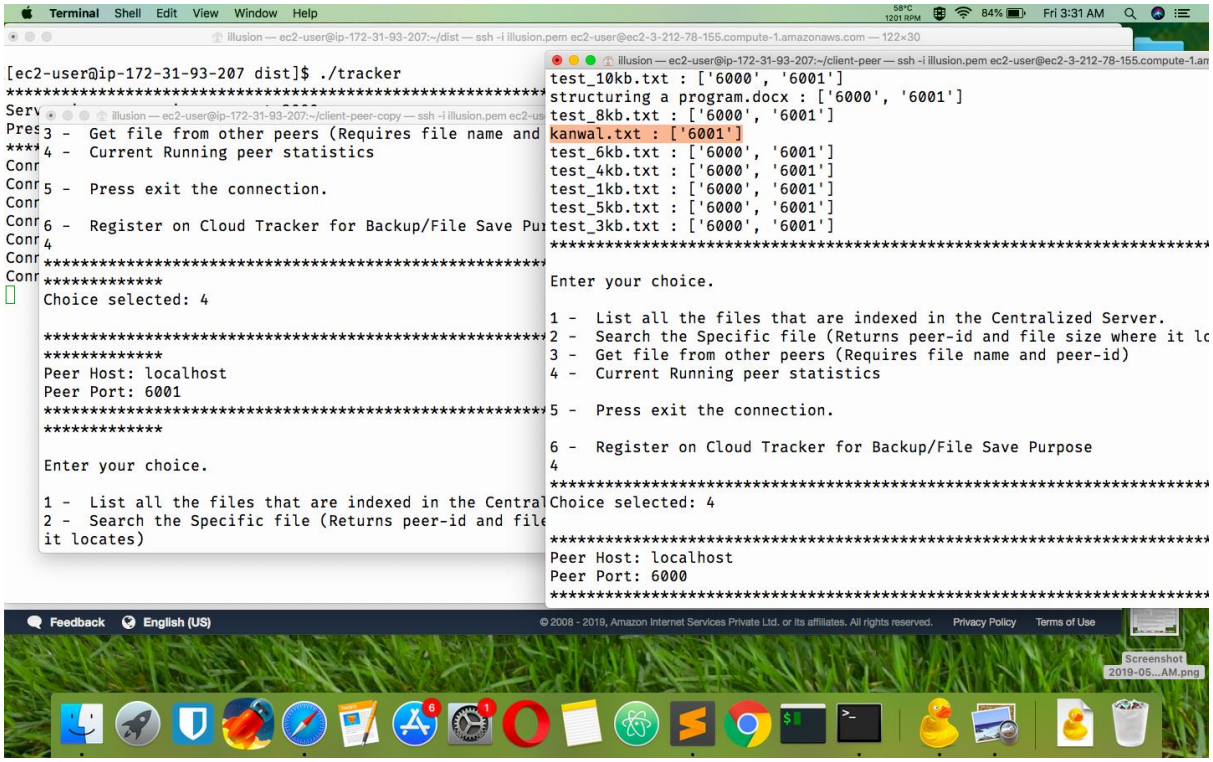

**(Figure 8.3 – Connection with Peer (Client) on Accept Status by Tracker Server)**



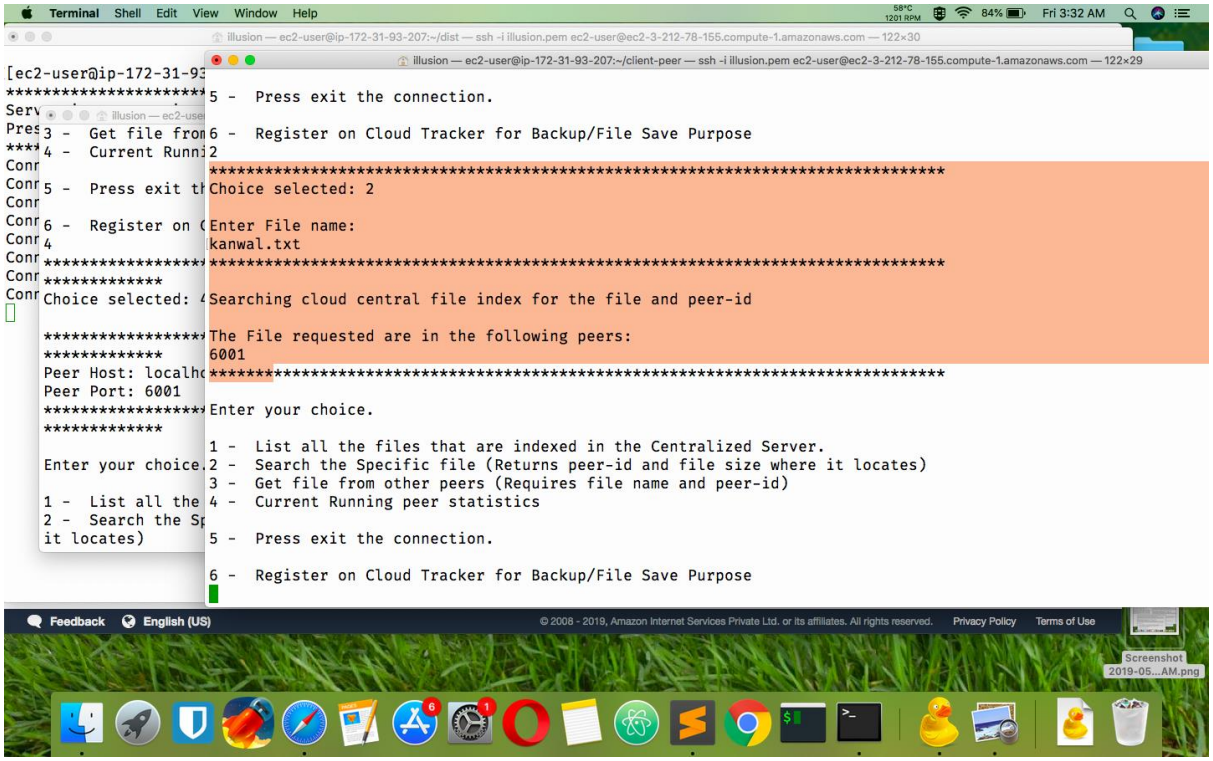

**(Figure 8.4 – Connection of Cloud Server with "n" number of Peer, here n = 2)**



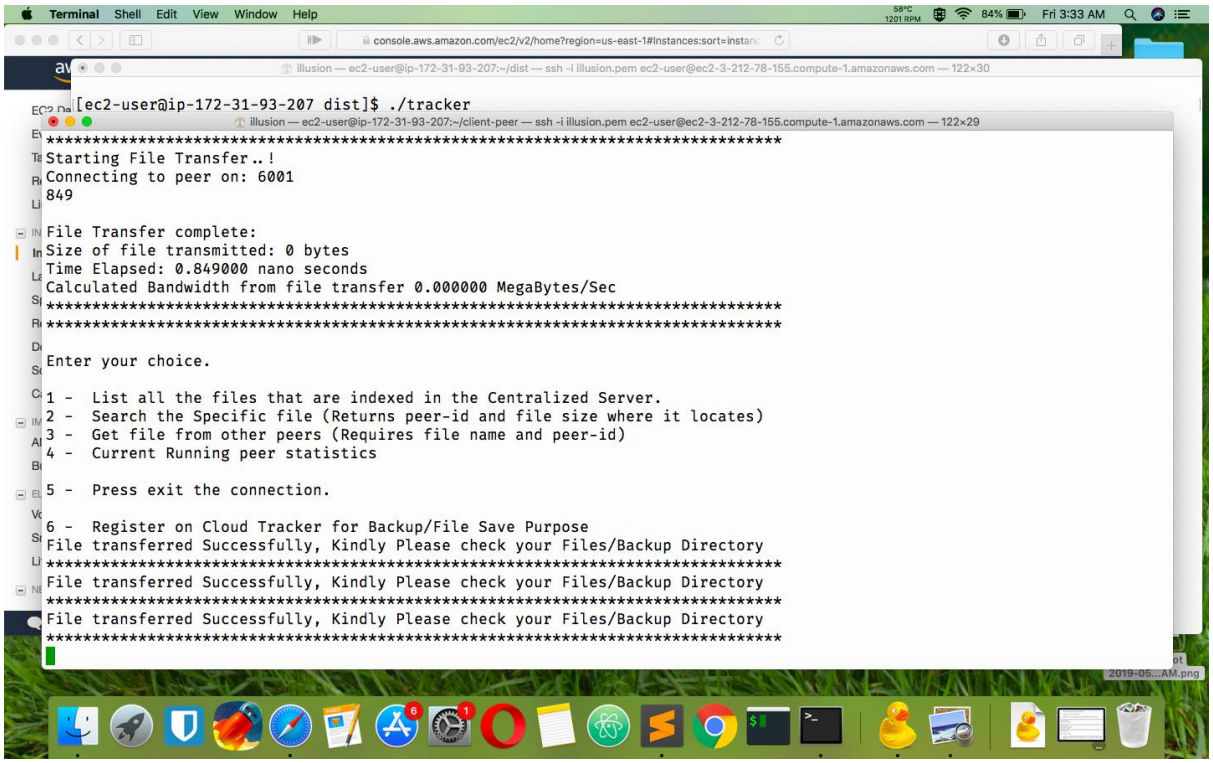

**(Figure 8.5 – List all files command indexed on Various Peers fetched via Tracker)**



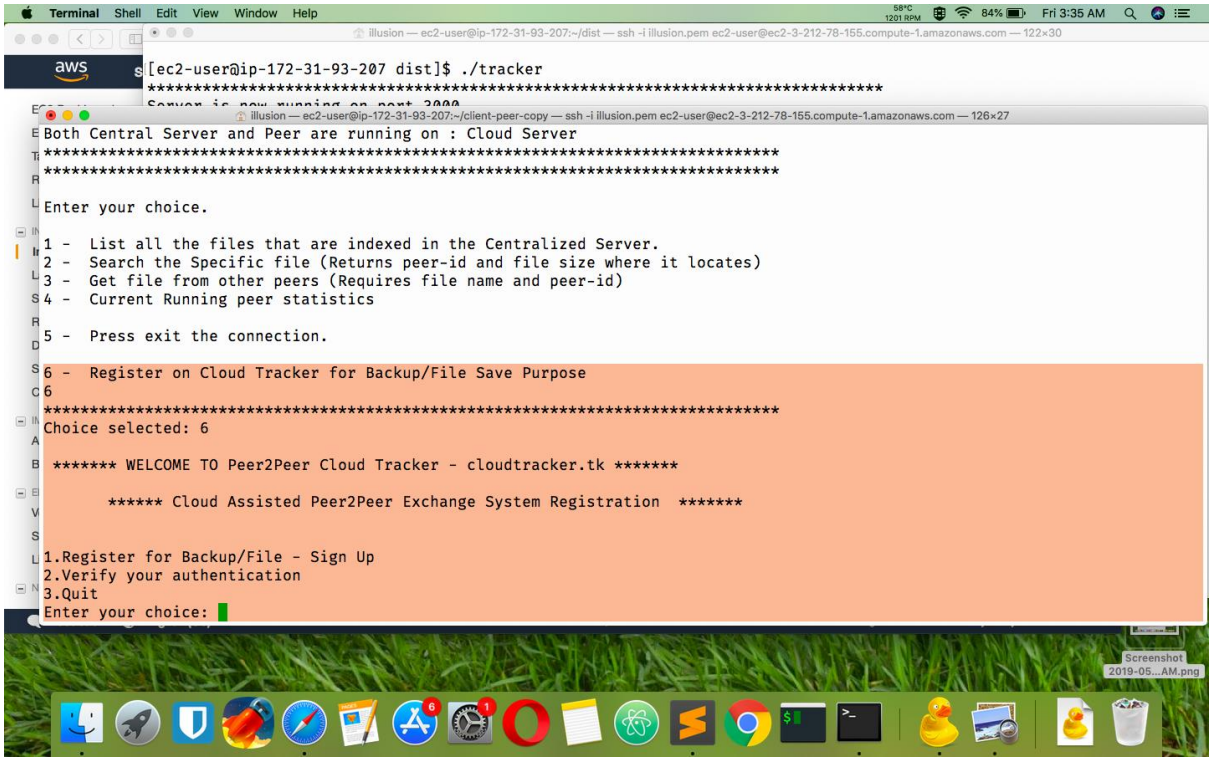

**(Figure 8.6 – Created custom file on peer id 6002 for transfer purpose via tracker)**




**(Figure 8.7 – Search file command invoked for checking particular peer which owns)**



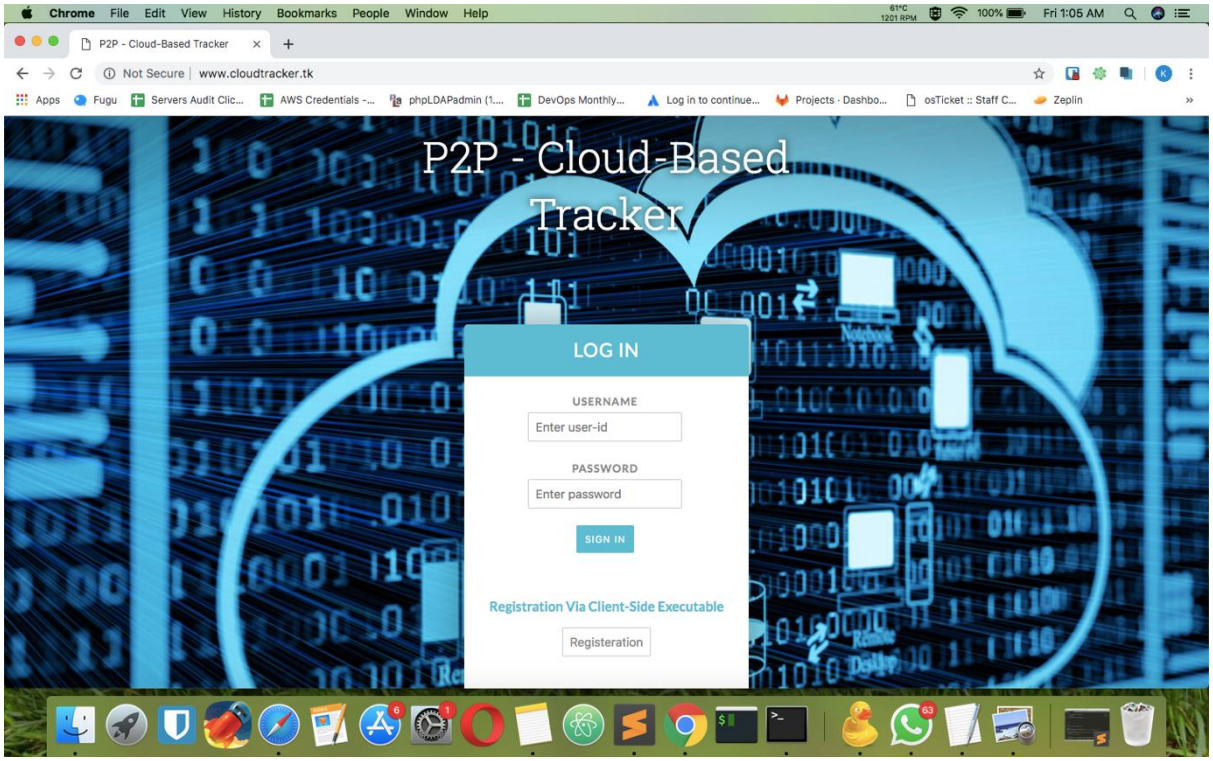

**(Figure 8.8 – Transfer invoked, sending file to the requested peer in network)**



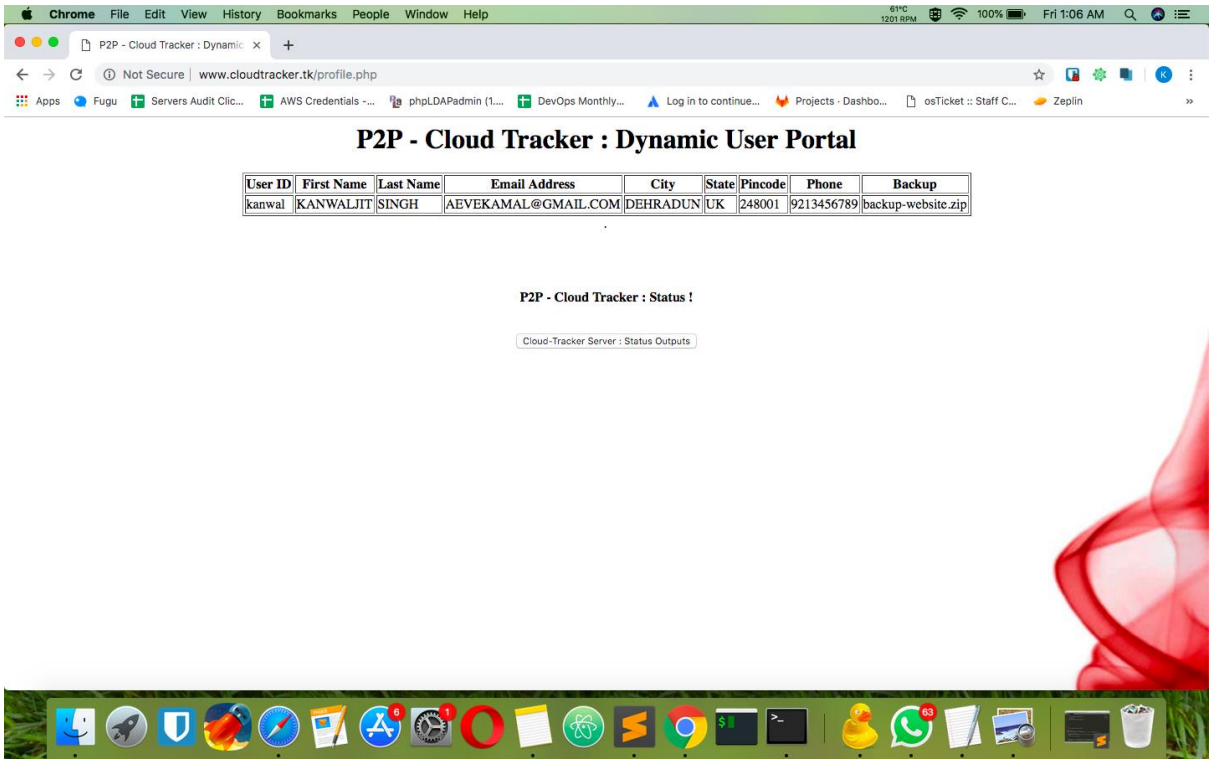

**(Figure 8.9 – Register command invoked for registering the user details on remote-Db)**

```
Choice selected: 6

******* WELCOME TO Peer2Peer Cloud Tracker - cloudtracker.tk *******

      ****** Cloud Assisted Peer2Peer Exchange System Registration  *******


1.Register for Backup/File - Sign Up
2.Verify your authentication
3.Quit
Enter your choice: 1
First Name: mayank
Last Name: bodwani
User id (Case-Sensitive): mayank
Password (Min. 6-12 Digit): 123456
Enter Email Address: mayaank.001@gmail.com
City: doon
State: uk
Pincode: 248001
Phone No.: 9876543215
Backup File Name: mayank-documents

Thanks for signing up!

mayank
Login cloudtracker.tk in browser for details


1.Register for Backup/File - Sign Up
2.Verify your authentication
3.Quit
Enter your choice: 2

Attempt 1:
User id (Case-Sensitive): kanwal
Password: 123456

kanwal
Successfully authenticated!
Login cloudtracker.tk in browser for details
```

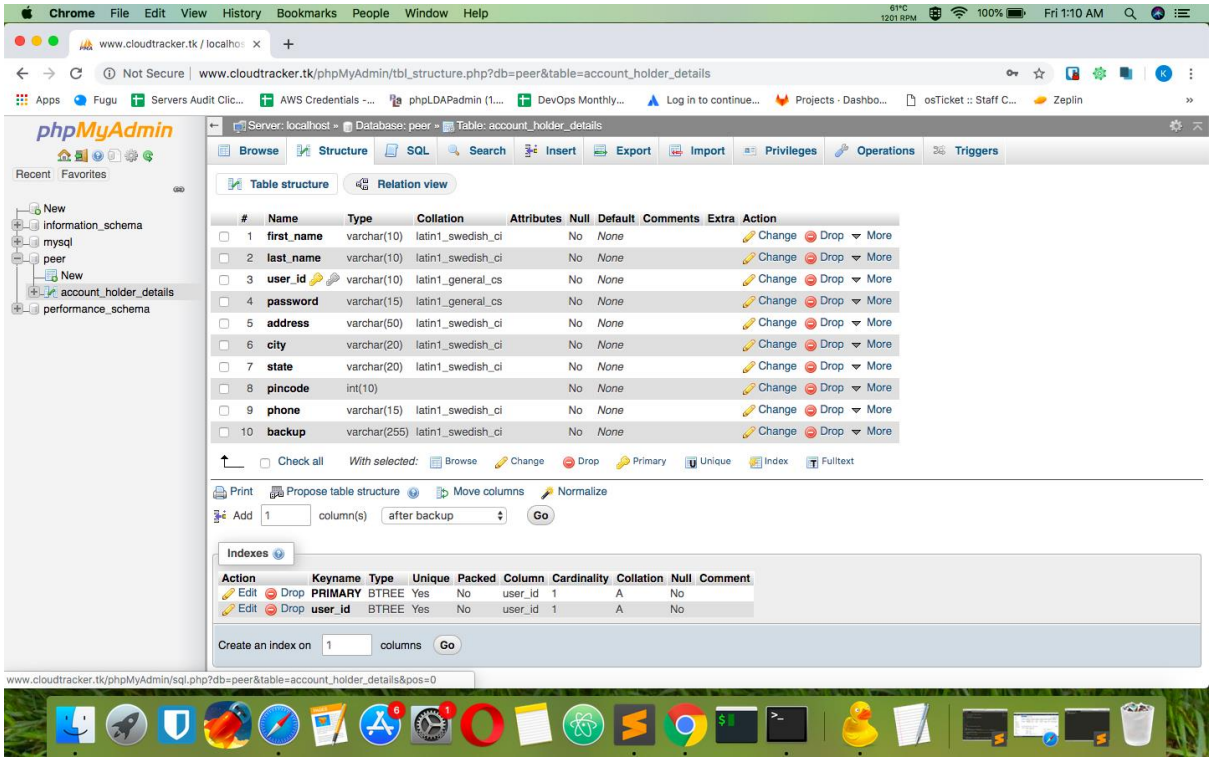
**(Figure 8.10 – Register Command Parameters for Saving End-Peer Information)**



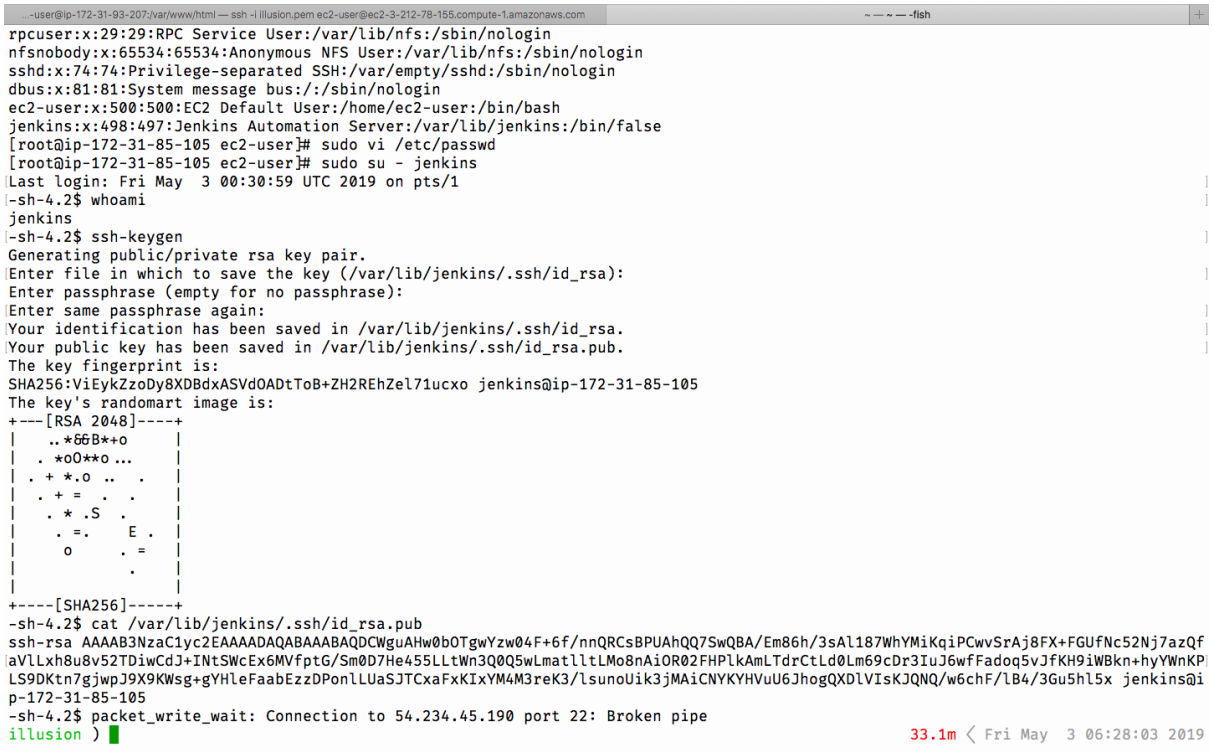

**(Figure 8.11 – Graphical Interface for Client (Peer) Interface serves via Apache2)**



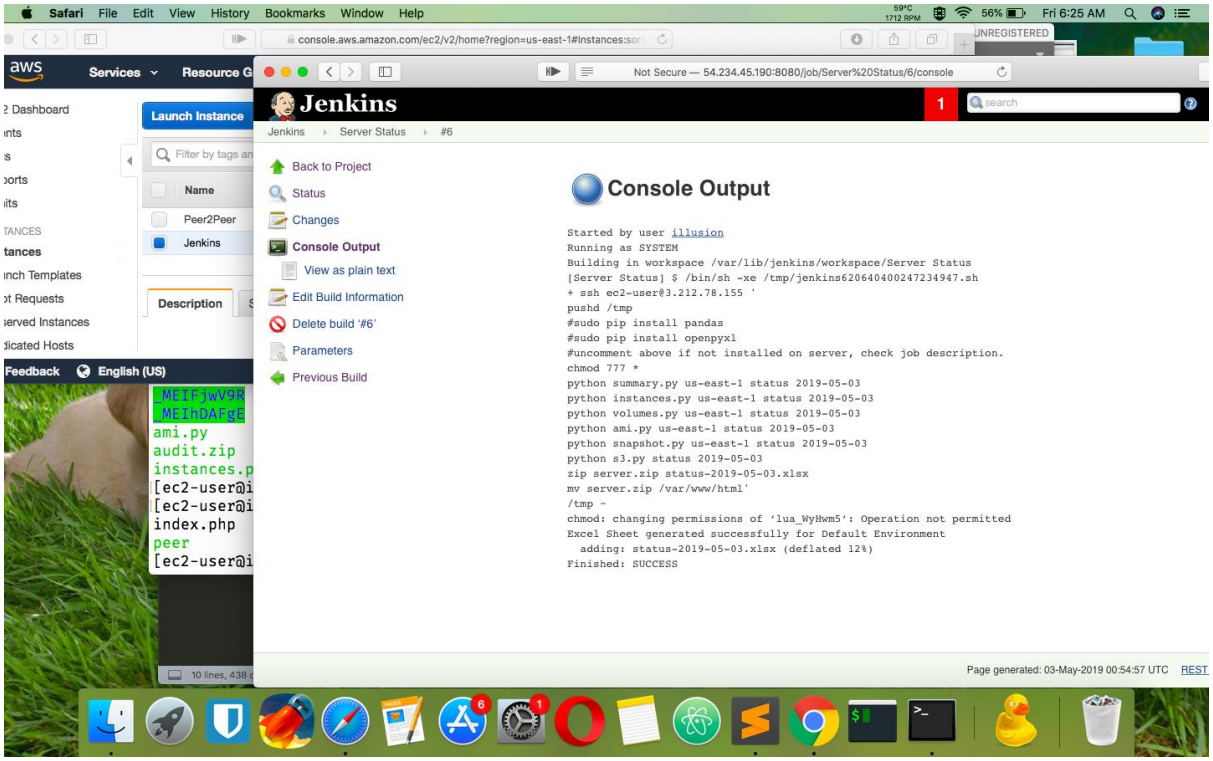

**(Figure 8.12 – Dynamic Data shown on GUI on session requests served via MySQL)**




**(Figure 8.13 – PhpMyAdmin Showing MySQL Database Structure and Schema)**




**(Figure 8.14 – SSH Key enabled for Jenkins User to SSH in Cloud Server for Audit)**




**(Figure 8.15 – Build made on Jenkins GUI for AWS Cloud report for tracker Audit)**

```
+ aws_instance.dns-server
    id:                                 <computed>
    ami:                                "ami-0c8b8e32659017cc5"
    arn:                                <computed>
    associate_public_ip_address:        <computed>
    availability_zone:                  <computed>
    cpu_core_count:                     <computed>
    cpu_threads_per_core:               <computed>
    ebs_block_device.#:                 <computed>
    ephemeral_block_device.#:           <computed>
    get_password_data:                  "false"
    host_id:                            <computed>
    instance_state:                     <computed>
    instance_type:                      "t2.micro"
    ipv6_address_count:                 <computed>
    ipv6_addresses.#:                   <computed>
    key_name:                           "dns-server"
    network_interface.#:                <computed>
    network_interface_id:               <computed>
    password_data:                      <computed>
    placement_group:                    <computed>
    primary_network_interface_id:       <computed>
    private_dns:                        <computed>
    private_ip:                         <computed>
    public_dns:                         <computed>
    public_ip:                          <computed>
    root_block_device.#:                <computed>
    security_groups.#:                  <computed>
    source_dest_check:                  "true"
    subnet_id:                          <computed>
    tags.%:                             "1"
    tags.Name:                          "dns-server"
    tenancy:                            <computed>
    volume_tags.%:                      <computed>
    vpc_security_group_ids.#:           <computed>

+ aws_key_pair.deployer
    id:                                 <computed>
    fingerprint:                        <computed>
```

**(Figure 8.16 – Terraform – Automatic Cloud Virtual Instance Deployment via API )**

# 9. System Requirements:

### 9.1. Hardware:

- **Memory:** Minimum 1 GB RAM (as per OS requirement)
- **Operating System:** Win XP, Linux Distro (Minimum)
- **Storage:** 512 MB (Minimum)
- **CPU:** Pentium 4 processor (3.0 GHz, or better)
- **Internet:** If Cloud Instance or Remote SQL Connectivity

### 9.2. Software:

- **Software:** Any Python IDE or terminal executing executable files.
- **Build:** Jenkins CI/CD Automation
- **Deploy:** Terraform Orchestration Tool
- **Environment:** Amazon Web Services (AWS) - Cloud
- **Web Stack:** Apache 2.4, PHP 7.0, MySQL 5.2.0

# 10. Conclusion and Future Scope:

The project results uncovers that there is possibility of backup service on decentralised framework conceivable which will diminishes the expenses of suppliers and buyers yet at the same time it needs legitimate security like encryption of documents and appropriate consistence represents information insurance and in addition as a future planned resume broken transmissions, virus filtering and chat support with versatile empowered mobile p2p application.

## 11. Schedule: (PERT CHART)

| Start | 80 days – 11 Weeks (Kanwaljit Singh) | | | |
|---|---|---|---|---|
| Analysis & Design 26/01/19 – 09/02/19 | | | | |
| 14 Days Kanwaljit Singh (Requirement Analysis and Designing the solution) | Planning Prototype 09/02/19 – 23/02/19<br>14 Days Kanwaljit (Planning a Prototype with defining all constraints for adapt future design) | Coding 23/02/19 – 07/04/19<br>42 Days Kanwaljit (Algorithm Implementation) | Testing 07/04/19 – 14/04/19<br>7 Days Kanwaljit (Debugging) | Documentation 14/04/19 – 17/04/19<br>3 Days Kanwaljit (Project Completion) |

# 12. References:

[1] Jovanovic, M., “Scalability Issues in Large Peer-to-Peer Networks - A Case Study of Gnutella," University of Cincinnati Technical Report 2001: http://www.ececs.uc.edu/ mjovanov/Research/paper.html

[2] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," Proceedings of SIGCOMM Conference on Applications, technologies, architectures, and protocols for computer communications, pp. 149-160, Aug. 2001.

[3] Freenet Protocol 1.0 Specification: http://freenetproject.org/index.php?page=protocol.

[4] The Gnutella Protocol Specification v0.4: http://dss.clip2.com/GnutellaProtocol04.pdf.

[5] H.T. Kung and W. Chun-Hsin, "Hierarchical Peer-to-Peer Network," Institute of Information Science Taiwan: Technical Report, vol. 2, no. 15, pp. 21-25, Apr. 2001.

[6] Jianguo Ding, Ilangko Balasingham, Pascal Bouvry. "Management of Overlay Networks: A Survey", 2009 Third International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies, 2009.

# A. APPENDIX I PROJECT CODE:

## ➢ Tracker Index Server Module (index_server.py)

```
#!/usr/bin/python
import socket
import sys
import threading
import time
import os
import json

HOSTNAME = '0.0.0.0'    # All Ipv4 listening
PORT = 3000             # Any port, taking it as 3000
BUFFER = 65536          # Max buffer

class central_server_class():
    def __init__(self):
        self.server = None
        self.threads_ = []
        self.file_index = {}
        self.peer_offset_port = 6000
        self.peer_list = []

    # Registering peer to the central tracker

    def register_peer(self):
        if len(self.peer_list) != 0:
            port_no = max(self.peer_list) + 1
            self.peer_list.append(port_no)
            return str(port_no)
        else:
            self.peer_list.append(self.peer_offset_port)
            return str(self.peer_offset_port)


    # Indexing the files, They are now in {file_name: [peer1,peer2..]} format.

    def index(self,request):
        for i,v in request.items():
            if i == 'command':
                pass
            else:
                if type(v) == list:
                    for sub_f in v:
                        sub_f = sub_f.lower()
                        if sub_f in self.file_index.keys():
                            self.file_index[sub_f].append(i)
                        else:
                            self.file_index[sub_f] = []
                            self.file_index[sub_f].append(i)

                elif type(v) == tuple:
                    files_added = v[0]
                    files_deleted = v[1]
                    if len(files_added) != 0:
                        for subs in files_added:
                            subs = subs.lower()
                            if subs in self.file_index.keys():
                                self.file_index[subs].append(i)
                            else:
                                self.file_index[subs] = []
                                self.file_index[subs].append(i)
                    if len(files_deleted) != 0:
                        for subs_ in files_deleted:
                            subs_ = subs_.lower()
```

```
63.                             self.file_index[subs_].pop(self.file_index.index(i))
64.
65.     def search(self,request):
66.         file_name = request['filename']
67.         if file_name in self.file_index.keys():
68.             return json.dumps({file_name:self.file_index[file_name]})
69.         else:
70.             return 'File not found in the index.'
71.
72.     def list_all_files(self):
73.         return json.dumps(self.file_index)
74.
75.     def destroy_peer(self,peer):
76.         for i,v in self.file_index.items():
77.             if unicode(peer) in v:
78.                 v.pop(v.index(unicode(peer)))
79.
80.     def process_request(self):
81.         client_connection = None
82.         print '*'*80
83.         print 'Server is now running on port %d' % PORT
84.         print 'Press control+c to shutdown the central server.!!'
85.         print '*'*80
86.         infinite = 1
87.
88.         while infinite:
89.             try:
90.                 self.server = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
91.                 self.server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
92.                 self.server.bind((HOSTNAME,PORT))
93.                 self.server.listen(10)  # Listen upto 10 connections before droping
    them (queue).
94.                 client_connection,client_addr = self.server.accept()
95.                 if client_connection:
96.                     print 'Connection Received from: %s on port: %d' % (client_addr
   [0],client_addr[1])
97.                     request = client_connection.recv(BUFFER)
98.                     req = json.loads(request)
99.                     command = req['command']
100.
101.                             if command == 'index':
102.                                 self.index(req)
103.                             elif command == 'list_all_files':
104.                                 all_files = self.list_all_files()
105.                                 client_connection.sendall(all_files)
106.                             elif command == 'search':
107.                                 search_results = self.search(req)
108.                                 client_connection.sendall(search_results)
109.                             elif command == 'register':
110.                                 peer_id = self.register_peer()
111.                                 client_connection.sendall(peer_id)
112.                             elif command == 'destroy':
113.                                 peer_id = req['peer']
114.                                 self.destroy_peer(peer_id)
115.                             else:
116.                                 pass
117.
118.                     except KeyboardInterrupt:
119.                         infinite = 0
120.                         print '*'*78
121.                         print '\nKeyboard Interrupt Caught.!'
122.                         print 'Shutting Down Peer Server..!!!'
123.                         print '*'*80
124.                         sys.exit(1)
125.
126.                     except Exception as e:
127.                         print '*'*80
128.                         print 'Processing Error..!!'
129.                         print e.message
130.                         print ''
```

```
                        #print '\nShutting down..!!'
                        sys.exit(1)
                        raise


                    finally:
                        self.server.close()

            def close(self):
                self.server.close()

            def run_(self):
                self.process_request()

        if __name__ == '__main__':
            try:
                cs = central_server_class()
                cs.run_()

            except KeyboardInterrupt:
                print '*'*78
                print '\nKeyboard Interrupt Caught.!'
                print 'Shutting Down Peer Server..!!!'
                print '*'*80
                cs.close()
                sys.exit(1)
```

## ➢ Peer Module (peer.py)

```
1.  from server import server_class
2.  from filesystem import FilesystemEventHandler
3.  from filesystem import destroy_peer
4.  import socket
5.  import sys
6.  import json
7.  import time
8.  import datetime
9.  import os
10. from User import User
11. from Database import Database as db
12. BUFFER = 65536
13. OUTPUT_DIR = './Files/'
14.
15. def mainMenu():
16.     logo = ''''' ******* WELCOME TO Peer2Peer Cloud Tracker - cloudtracker.tk *****
    **
17.
18.        ****** Cloud Assisted Peer2Peer Exchange System Registration  ******* '''
19.
20.     print(logo)
21.     choice  = -1
22.     current_user = -1
23.
24.     while choice != '4':
25.         print '\n'
26.         print '1.Register for Backup/File - Sign Up'
27.         print '2.Verify your authentication'
28.         print '3.Quit'
29.
30.         choice = raw_input('Enter your choice: ')
31.
32.         if choice == '1':
33.             first_name = raw_input('First Name: ').upper()
34.             last_name = raw_input('Last Name: ').upper()
35.             user_id = raw_input('User id (Case-Sensitive): ')
36.             while True:
37.                 password = raw_input('Password (Min. 6-12 Digit): ')
38.                 if 6 <= len(password) < 12:
39.                     break
40.                 print("")
```

```
                print ('The password must be between 6 and 12 characters.\n')

            address = raw_input('Enter Address: ').upper()
            city = raw_input('City: ').upper()
            state = raw_input('State: ').upper()
            pincode = int(raw_input('Pincode: '))
            phone = int(raw_input('Phone No.: '))
            backup = raw_input('Backup File Name: ')
            new_user = User(first_name, last_name, user_id, password, address, city
    , state, pincode, phone, backup)

            if new_user.save():
                print ''
                print 'Thanks for signing up!'
                current_user = user_id
                print ''
                print(current_user)
                print 'Login cloudtracker.tk in browser for details'
            else:
                print 'Cannot create an account for you'

        elif choice == '2':
            attempts = 1
            while attempts <=3:
                print ''
                print 'Attempt %d: ' % (attempts)
                user_id = raw_input('User id (Case-Sensitive): ')
                password = raw_input('Password: ')

                if (User.authenticate(user_id,password)):
                    current_user = user_id
                    print ''
                    print(current_user)
                    print 'Successfully authenticated!'
                    print 'Login cloudtracker.tk in browser for details'
                    break
                else:
                    print 'Invalid credentials!'
                attempts += 1
            else:
                print 'Max sign in attempts reached'
        elif choice == '3':
            exit()
        else:
            print 'Invalid option!'

class query_indexer():
    def __init__(self):
        self.ci_server_host = 'localhost'
        self.ci_server_port = 3000
        self.ci_server_addr = (self.ci_server_host,self.ci_server_port)
        self.index_socket = None
        self.credentials = None
        self.LIST_FILES = json.dumps({'command':'list_all_files'})
        self.GET_CREDENTIALS = json.dumps({'command':'register'})
        self.SEARCH_FOR_FILE = {'command':'search'}

    def send_command_to_cs(self,cmd):
        try:
            self.index_socket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
            self.index_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

                self.index_socket.connect(self.ci_server_addr)
                self.index_socket.sendall(cmd)
                response = self.index_socket.recv(BUFFER)
                return response

            except Exception as e:
                print 'Cannot connect to the Centralized Cloud Server'
                print 'Please make sure that the Server is running'
                print '*'*80
```

```
109.                     print e.message
110.                     sys.exit(1)
111.                     return 'error'
112.
113.                 finally:
114.                     self.index_socket.close()
115.
116.             def get_credentials(self):
117.                 print '*'*80
118.                 print 'Registering peer port & address and fetching credentials from
    central cloud server\n'
119.                 try:
120.                     self.credentials = self.send_command_to_cs(self.GET_CREDENTIALS)
121.                     if self.credentials == 'error':
122.                         raise
123.                     self.credentials = int(self.credentials)
124.                     return self.credentials
125.                 except Exception as e:
126.                     print 'Retrive credentials failed'
127.                     print '*'*80
128.
129.             def list_all_files(self):
130.                 try:
131.                     all_files = self.send_command_to_cs(self.LIST_FILES)
132.                     all_files_ = json.loads(all_files)
133.                     print '*'*80
134.                     print '\nThe files index list from central cloud server:\n'
135.
136.                     for i,v in all_files_.items():
137.                         print '%s : %s' % (i,map(unicode.encode,v))
138.                     print '*'*80
139.                 except Exception as e:
140.                     print 'Retrive files list failed'
141.                     print e.message
142.                     print '*'*80
143.
144.             def search_for_file(self,file_name):
145.                 print '*'*80
146.                 print '\nSearching cloud central file index for the file and peer-
    id'
147.                 try:
148.                     self.SEARCH_FOR_FILE['filename'] = file_name
149.                     search_command = json.dumps(self.SEARCH_FOR_FILE)
150.                     search_file = self.send_command_to_cs(search_command)
151.                     search_results = json.loads(search_file)
152.                     try:
153.                         print '\nThe File requested are in the following peers:'
154.                         for files_ in search_results[file_name]:
155.                             print files_,
156.                         print ''
157.                         print '*'*80
158.                     except:
159.                         print search_results
160.                         print '*'*80
161.                 except Exception as e:
162.                     print 'Retrive search file list failed (FILE NOT FOUND)'
163.                     print e.message
164.                     print '*'*80
165.
166.             def obtain(self,peer_id,file_name):
167.                 print '*'*80
168.                 print 'Starting File Transfer..!'
169.                 print 'Connecting to peer on:', peer_id
170.
171.                 st1 = datetime.datetime.now()
172.                 try:
173.                     server_addr = ('localhost',int(peer_id))
174.                     connection = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
175.                     connection.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
```

```
                    connection.connect(server_addr)
                    connection.sendall(file_name)
                    response = connection.recv(BUFFER)
                    if response == 'Nope':
                        print '\nOOPS, File Not Found'
                        print '*'*80
                        return
                except Exception as e:
                    print 'File Transfer failed'
                    print e.message
                    print '*'*80
                    return
                et1 = datetime.datetime.now()

                try:
                    file_path = OUTPUT_DIR + file_name
                    fh = open(file_path,'wb')
                    fh.write(response)
                    fh.close()
                    et2 = datetime.datetime.now()
                    time_elapsed = (et2 - et1) + (et1 - st1)
                    ns = time_elapsed.microseconds * pow(10,-3)
                    file_size = os.path.getsize(file_path)
                    print time_elapsed.microseconds
                    bandwidth = ((file_size * pow(10,3)) / time_elapsed.microseconds
    )
                    print '\nFile Transfer complete:'
                    print 'Size of file transmitted: %d bytes' % file_size
                    print 'Time Elapsed: %f nano seconds' % ns
                    print 'Calculated Bandwidth from file transfer %f MegaBytes/Sec'
     % bandwidth
                    print '*'*80

                except Exception as e:
                    print 'File Transfer failed, check you connection parameters.'
                    print e.message
                    print '*'*80
                    return

                finally:
                    connection.close()

            def peer_stats(self):
                print '*'*80
                print 'Peer Host: localhost'
                print 'Peer Port: %d' % self.credentials
                print '*'*80

        if __name__ == '__main__':
            os.system(‘mkdir Files’)
            print '*'*80

            try:
                qi = query_indexer()
            except Exception as e:
                print 'Failed to import query package.!'
                print '*'*80
                sys.exit(1)

            credentials = qi.get_credentials()

            if credentials == 'error':
                print 'Central Indexing cloud Server is not running.!!, Please start
     that first'
                print '*'*80
                sys.exit(1)

            try:
                server = server_class(credentials)
                server.setDaemon(True)
                server.start()
```

```
244.            except Exception as e:
245.                print 'Peer Server Could not be started.'
246.                print e.message
247.                print '*'*80
248.                sys.exit(1)
249.
250.            try:
251.                fs_handler = FilesystemEventHandler(OUTPUT_DIR,credentials)
252.                fs_handler.setDaemon(True)
253.                fs_handler.start()
254.            except Exception as e:
255.                print 'File system monitor could not be started.'
256.                print e.message
257.                print '*'*80
258.                sys.exit(1)
259.
260.            print 'Central Indexing Cloud Server is running on port  : 3000'
261.            print 'This Peer Server is running on port         : %d' % credentials
262.            print 'Both Central Server and Peer are running on : Cloud Server'
263.
264.            try:
265.                possibilities = [1,2,3,4,5,6]
266.                print '*'*80
267.                while 1:
268.                    time.sleep(1)
269.                    print '\nEnter your choice.\n'
270.
271.                    print '1 -  List all the files that are indexed in the Centraliz
    ed Server.'
272.                    print '2 -  Search the Specific file (Returns peer-
    id and file size where it locates)'
273.                    print '3 -  Get file from other peers (Requires file name and pe
    er-id)'
274.                    print '4 -  Current Running peer statistics\n'
275.                    print '5 -  Press exit the connection.\n'
276.                    print '6 -  Register on Cloud Tracker for Backup/File Save Purpo
    se'
277.                    command = raw_input()
278.                    if int(command) not in possibilities:
279.                        print 'Invalid choice entered, please select again\n'
280.                        continue
281.                    print '*'*80
282.                    print 'Choice selected: %d \n' % int(command)
283.
284.                    if int(command) == 3:
285.                        print 'Enter Peer Id:\n'
286.                        peer_transfer_id = raw_input()
287.                        print 'Enter File name:'
288.                        file_name = raw_input()
289.                        file_name = file_name.lower()
290.                        qi.obtain(peer_transfer_id,file_name)
291.
292.                    elif int(command) == 1:
293.                        qi.list_all_files()
294.
295.                    elif int(command) == 6:
296.                         mainMenu();
297.
298.                    elif int(command) == 2:
299.                        print 'Enter File name:'
300.                        file_name = raw_input()
301.                        file_name = file_name.lower()
302.                        qi.search_for_file(file_name)
303.
304.                    elif int(command) == 4:
305.                        qi.peer_stats()
306.
307.                    elif int(command) == 5:
308.                         exit()
309.
310.
```

```
311.            except Exception as e:
312.                print e.message
313.            except KeyboardInterrupt:
314.                destroy_peer(int(credentials))
315.                print '*'*78
316.                print '\nKeyboard Interrupt Caught.!'
317.                print 'Shutting Down Peer Server..!!!'
318.                print '*'*80
319.                time.sleep(1)
320.                sys.exit(1)
321.
322.            finally:
323.                server.close()
```

## ➢ Client Server (server.py)

```
1.  #!/usr/bin/python
2.  import socket
3.  import sys
4.  import threading
5.  import time
6.  import os
7.
8.  HOSTNAME = 'localhost'
9.  BUFFER = 65536
10.
11. class handlers(threading.Thread):
12.     def __init__(self,client):
13.         super(handlers,self).__init__()
14.         self.client = client
15.
16.     def request_handler(self):
17.         try:
18.             file_to_fetch = self.client.recv(BUFFER)
19.             path_to_file = './Files/'+file_to_fetch
20.             if os.path.isfile(path_to_file):
21.                 fh = open(path_to_file,'rb')
22.                 binary_data = fh.read()
23.                 self.client.sendall(binary_data)
24.                 return 'File Sent.!!'
25.                 fh.close()
26.             else:
27.                 self.client.sendall('Nope')
28.                 return 'File not found..!!'
29.
30.         except Exception as e:
31.             self.client.close()
32.             return  'File dosent exist.!!'
33.
34.     def response_handler(self,data):
35.         try:
36.             self.client.sendall(data)
37.         except Exception as e:
38.             self.client.send('Unable to send the data, Check the connection.!')
39.             self.client.close()
40.             return
41.
42.     def run(self):
43.         print '*'*80
44.         print 'Responding to client requests..!!'
45.
46.         try:
47.             client_data = self.request_handler()
48.
49.         finally:
50.             self.client.close()
51.
52. class server_class(threading.Thread):
53.     def __init__(self,port):
54.         super(server_class,self).__init__()
```

```
        self.PORT = port
        self.server = None
        self.threads_ = []

    def process_data(self):
        client_connection = None

        while True:
            try:
                self.server = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
                self.server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
                self.server.bind((HOSTNAME,self.PORT))
                self.server.listen(10)  # Listen upto 10 connections before droping them (queue).
                client_connection,client_addr = self.server.accept()
                if client_connection:
                    print 'Connection Received from: %s on port: %d' % (client_addr[0],client_addr[1])
                    handle = handlers(client_connection)
                    #multiple_cli.setDaemon(True)
                    thread_ = handle.start()
                    self.threads_.append(thread_)

            except Exception as e:
                print '*'*80
                print 'Processing Error..!!'
                print e.message
                print '\nShutting down..!!'
                sys.exit(1)
                raise

            finally:
                self.server.close()

    def close(self):
        self.server.close()

    def run(self):
        self.process_data()

    # Establish Connection between this peer and the centralized indexing server
    # Get the port number and other credentials from the centralized indexeing server
```

## ➢ File System Handler (filesystem.py)

```
import sys
import threading
import os
import time
import socket
import json

class FilesystemEventHandler(threading.Thread):
    def __init__(self,monitor_dir,peer_id):
        try:
            super(FilesystemEventHandler,self).__init__()
            self.cs_indexing_server_addr = ('localhost',3000)
            self.monitor_dir = monitor_dir
            self.files = []
            self.current_directory = './Files'
            self.peer_id = peer_id
            self.connection = None
        except socket.error as e:
            print 'Indexing cloud server is down.!!'
            sys.exit(1)

    def monitor(self):
        while 1:
```

```
            if len(self.files) != 0:
                self.files.sort()
                cur_files = os.listdir(self.current_directory)
                cur_files.sort()
                if cur_files == self.files:
                    pass
                else:
                    # Make a note of files added and deleted.
                    changes_added = list(set(cur_files) - set(self.files))
                    changes_removed = list(set(self.files) - set(cur_files))
                    changes = (changes_added,changes_removed)
                    self.registry(changes,self.peer_id)
            else:
                self.files = os.listdir(self.current_directory)
                self.registry(self.files,self.peer_id)

            time.sleep(1)

    def registry(self,changes,peer_id):
        to_send_ = {peer_id:changes,'command':'index'}
        to_send = json.dumps(to_send_)
        try:
            self.connection = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
            self.connection.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
            self.connection.connect(self.cs_indexing_server_addr)
            self.connection.sendall(to_send)
        except Exception as e:
            print 'File transferred Successfully, Kindly Please check your Files/Ba
ckup Directory'
            print '*'*80

    def run(self):
        self.monitor()

class destroy_peer():
    def __init__(self,peer_id):
        self.cs_indexing_server_addr = ('localhost',3000)
        self.destroy_ = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
        self.destroy_.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        self.destroy_.connect(self.cs_indexing_server_addr)
        self.destroy_cmd = {'command':'destroy','peer':peer_id}
        self.kill_peer(self.destroy_cmd)

    def kill_peer(self,destroy_cmd):
        destroy_cmd = json.dumps(destroy_cmd)
        self.destroy_.sendall(destroy_cmd)
```

## ➢ Online Code - Portal (index.php)

```
<?php
session_start();
if(isset($_POST['username'])) {
   $_SESSION['username'] = $_POST['username'];
}
$message="";
if(count($_POST)>0) {
    $conn = mysqli_connect(null,"root","peer2peer","peer");
    $result = mysqli_query($conn,"SELECT * FROM account_holder_details WHERE user_i
d='" . $_POST["username"] . "' and password = '". $_POST["password"]."'");
    $count  = mysqli_num_rows($result);
    if($count==0) {
        $message = "Invalid Username or Password!";
    } else {
        header('Location: http://www.cloudtracker.tk/profile.php');
    }
}
?>
<html>
<head>
    <title>P2P - Cloud-Based Tracker</title>
```

```
21.
22.     <!-- Google Fonts -->
23.     <link href='https://fonts.googleapis.com/css?family=Roboto+Slab:400,100,300,700
    |Lato:400,100,300,700,900' rel='stylesheet' type='text/css'>
24.
25.
26.
27.     <link rel="stylesheet" href="style.css">
28. </head>
29. <body>
30.     <div class="container">
31.         <div class="top">
32.             <h1 id="title" class="hidden"><span id="logo">P2P - Cloud-
    Based Tracker</span></span></h1>
33.         </div>
34.         <div class="login-box animated fadeInUp">
35.             <div class="box-header">
36.                 <h2>Log In</h2>
37.             </div><form name="quiz" method="post" action="" >
38.             <label for="username">Username</label>
39.             <br/>
40.             <input type="text" name="username" placeholder="Enter user-id
41. " id="username">
42.             <br/>
43.             <label for="password">Password</label>
44.             <br/>
45.             <input type="password" name="password" placeholder="Enter password" id=
    "password">
46.             <br/>
47.             <button type="submit">Sign In</button>
48.             <br/></br>
49.             <div class="message"><?php if($message!="") { echo $message; } ?></br>

50.                         <b><p>Client-Side Peer Executable</p></b>
51.                         <input type="button" onclick="location.href='http://www.clo
    udtracker.tk/peer_x64_executable.exe';" value="Peer Client via Windows" /></div>
52.                         <input type="button" onclick="location.href='http://www.clo
    udtracker.tk/peer';" value="Peer Client via linux/MacOS" /></div>
53.
54.
55.         </div>
56.     </div>
57. </body>
58. </html>
```

## ➢ Profile (profile.php)

```
1.  <html>
2.  <head>
3.
4.      <title>P2P - Cloud Tracker : Dynamic User Portal</title>
5.
6.    <style type="text/css">
7.      body{
8.          text-align:center;
9.          background-image: url('peer.jpg');
10.
11.     }
12.
13.
14.
15.
16.     </style>
17.
18.
19. </head>
20.
21. <body>
22. <center>
23.     <h1>P2P - Cloud Tracker : Dynamic User Portal</h1>
```

```
24.
25.
26. <?php
27. session_start();
28. $conn = mysqli_connect(null,"root","peer2peer","peer");
29. $result = mysqli_query($conn,"SELECT * FROM account_holder_details WHERE user_id='"
    .$_SESSION['username']."'");
30.
31.
32. echo '<table class="text" border=1px>';
33. echo '<th>User ID</th><th>First Name</th><th>Last Name</th><th>Email Address</th><t
    h>City</th><th>State</th><th>Pincode</th><th>Phone</th><th>Backup</th>';
34.
35.  while($data = mysqli_fetch_array($result))
36. {
37.
38.
39. echo'<tr>';
40. echo '<td>'.$data['user_id'].'</td><td>'.$data['first_name'].'</td><td>'.$data['las
    t_name'].'</td><td>'.$data['address'].'</td><td>'.$data['city'].'</td><td>'.$data['
    state'].'</td><td>'.$data['pincode'].'</td><td>'.$data['phone'].'</td><td>'.$data['
    backup'].'</td>';
41. echo'</tr>';
42.
43. }
44.
45. echo '</table>. </br><br/><br/>';
46.
47.
48.
49.
50.
51. ?>
52. </br>
53. <b><p>P2P - Cloud Tracker : Server Status !</p></b></br>
54. <input type="button" onclick="location.href='https://cloudtracker.tk/server.zip';"
    value="Cloud-Tracker Server : Status Outputs" />
55.
56.   </center>
57.
58.
59.
60.
61.
62.
63. </body>
64.
65. </html>
```

## ➢ Users Admin (users.php)

```
1.  <html>
2.  <head>
3.
4.      <title>Tracker - All Users Database</title>
5.
6.    <style type="text/css">
7.      body{
8.          text-align:center;
9.          background-image: url('peer.jpg');
10.
11.     }
12.
13.
14.
15.
16.     </style>
17.
18.
19. </head>
```

```
20.
21. <body>
22. <center>
23.     <h1>Tracker : Dynamic All Users Database Portal</h1>
24.
25.
26. <?php
27.
28. $conn = mysqli_connect(null,"root","peer2peer","peer");
29. $result = mysqli_query($conn,"SELECT * FROM account_holder_details");
30.
31.
32. echo '<table class="text" border=1px>';
33. echo '<th>User ID</th><th>First Name</th><th>Last Name</th><th>Address</th><th>City
    </th><th>State</th><th>Pincode</th><th>Phone</th>';
34.
35.  while($data = mysqli_fetch_array($result))
36. {
37.
38.
39. echo'<tr>';
40. echo '<td>'.$data['user_id'].'</td><td>'.$data['first_name'].'</td><td>'.$data['las
    t_name'].'</td><td>'.$data['address'].'</td><td>'.$data['city'].'</td><td>'.$data['
    state'].'</td><td>'.$data['pincode'].'</td><td>'.$data['phone'].'</td><td>'.$data['
    backup'].'</td>';
41. echo'</tr>';
42.
43. }
44.
45. echo '</table>. </br><br/><br/>';
46.
47.
48.
49.
50.
51.
52. ?>
53. </br>
54. <b><p>P2P - Cloud Tracker : Server Status !</p></b></br>
55. <input type="button" onclick="location.href='https://cloudtracker.tk/server.zip';"
    value="Cloud-Tracker Server : Status Outputs" />
56.
57.   </center>
58.
59.
60.
61.
62.
63.
64. </body>
65.
66. </html>
```

## ➢ Remote MySQL Dump (peer.sql)

```
1.  -- phpMyAdmin SQL Dump
2.  -- version 4.8.5
3.  -- https://www.phpmyadmin.net/
4.  --
5.  -- Host: localhost
6.  -- Generation Time: May 02, 2019 at 10:20 PM
7.  -- Server version: 5.6.43
8.  -- PHP Version: 7.0.33
9.
10. SET SQL_MODE = "NO_AUTO_VALUE_ON_ZERO";
11. SET AUTOCOMMIT = 0;
12. START TRANSACTION;
13. SET time_zone = "+00:00";
14.
15.
```

```sql
/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8mb4 */;

--
-- Database: `peer`
--

-- --------------------------------------------------------

--
-- Table structure for table `account_holder_details`
--

CREATE TABLE `account_holder_details` (
  `first_name` varchar(10) NOT NULL,
  `last_name` varchar(10) NOT NULL,
  `user_id` varchar(10) CHARACTER SET latin1 COLLATE latin1_general_cs NOT NULL,
  `password` varchar(15) CHARACTER SET latin1 COLLATE latin1_general_cs NOT NULL,
  `address` varchar(50) NOT NULL,
  `city` varchar(20) NOT NULL,
  `state` varchar(20) NOT NULL,
  `pincode` int(10) NOT NULL,
  `phone` varchar(15) NOT NULL,
  `backup` varchar(255) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--
-- Dumping data for table `account_holder_details`
--

INSERT INTO `account_holder_details` (`first_name`, `last_name`, `user_id`, `passwo
rd`, `address`, `city`, `state`, `pincode`, `phone`, `backup`) VALUES
('KANWALJIT', 'SINGH', 'kanwal', '123456', 'AEVEKAMAL@GMAIL.COM', 'DEHRADUN', 'UK',
 248001, '9213456789', 'backup-website.zip'),
('MAYANK', 'BODWANI', 'mayank', '123456', 'MAYAANK.001@GMAIL.COM', 'DOON', 'UK', 24
8001, '9876543215', 'mayank-documents'),
('VIKAS', 'GOSWAMI', 'vikas', '123456', 'VIKAS.001@GMAIL.COM', 'LUCKNOW', 'UK', 160
018, '9876543261', 'vikas.txt');

--
-- Indexes for dumped tables
--

--
-- Indexes for table `account_holder_details`
--
ALTER TABLE `account_holder_details`
  ADD PRIMARY KEY (`user_id`),
  ADD UNIQUE KEY `user_id` (`user_id`);
COMMIT;

/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
```