

# CHAPTER 1

## INTRODUCTION

### 1.1 ABOUT THE PROJECT

Data is one of the most valuable assets that any company can hold. One of the best ways to store these assets is within the cloud. Cloud computing is the on-demand delivery of IT resources over the Internet with pay-as-you-go pricing. Instead of buying, owning, and maintaining physical data centers and servers, you can access technology services, such as computing power, storage, and databases, on an as-needed basis from a cloud provider like Amazon Web Services (AWS). In today's cloud-driven world, ensuring the availability, integrity, and reliability of data is a top priority for enterprise datacentres.

Unexpected failures such as hardware crashes, network outages, or cloud service interruptions can lead to significant data loss and system downtime. To address these challenges, this project introduces HAEEdge — an intelligent, automated database failover and backup management system designed for cloud environments. HAEEdge integrates with VirtualBox to create a hybrid backup strategy that combines both cloud and local storage, offering enhanced resilience and faster disaster recovery.

By incorporating machine learning algorithms for failure prediction, and advanced techniques like data compression, probabilistic protection, and automated service migration, the system ensures minimal data loss, reduced recovery time, and optimized storage usage. HAEEdge is structured into multiple functional modules, each responsible for a specific aspect of the failover and recovery lifecycle. The HAEEdge Backup Node acts as the primary component that collects and stores critical data at predefined intervals, while the Virtual Backup Server Module offers a secure environment through VirtualBox for backup storage and retrieval. The PGP Data Compression Module ensures that all backup data is efficiently compressed without any loss of quality, significantly reducing the storage footprint.

## CHAPTER 2

### SYSTEM ANALYSIS

#### 2.1. EXISTING SYSTEM

- **Microsoft Azure** : Azure offers physical and virtual support for Linux, Windows, VMware virtual machines, Windows Server, and System Center management tools. It is a solid choice for companies who are interested in protecting their critical workloads via Hyper-V or even VMware, especially companies who are already on the Microsoft software stack.
- **Quorum OnQ** : Quorum OnQ provides almost-instant recovery with flexible options around multi-site failover and off-site protection. Quorum OnQ is a high-availability disaster recovery solution designed specifically to minimize downtime and data loss for businesses of all sizes. It offers a comprehensive, integrated platform that includes backup, instant recovery, deduplication, replication, and testing all in one appliance or virtual environment. What makes Quorum OnQ stand out is its ability to provide instant recovery of critical systems, applications, and data in the event of any failure or disaster.
- **Amazon Web Services (AWS)** : AWS offers DR for various databases and enterprise applications such as MySQL, SQL Server, and even SAP. AWS uses their CloudEndure Disaster Recovery service to continually replicate the OS, databases, applications, and files into your Region of choice. Amazon Web Services (AWS) is a leading cloud platform that provides robust and scalable disaster recovery (DR) solutions for businesses seeking high availability, fault tolerance, and data durability. AWS offers built-in tools and services specifically designed to safeguard enterprise applications and data including databases like MySQL, SQL Server, and enterprise platforms such as SAP.
- **Zerto** : Zerto provides a user-friendly interface and flexible solution that can convert between VMware, Hyper-V, and even Amazon Web Services (AWS). Zerto is a powerful disaster recovery and IT resilience platform that offers continuous data protection (CDP), backup, and workload mobility for virtualized and cloud environments

### **2.1.1 DISADVANTAGES**

- Speedy recovery is crucial to mitigate downtime and high latency could mean added downtime.
- Data redundancy allows your organization to store its data in two separate places.
- The upfront capital expenditures associated with building a second data repository, however, are numerous. There are warehouse costs, maintenance costs, hardware costs, and transportation costs if data is stored on disks and tapes.
- Hire IT technical staff to develop, test, and execute your DR strategy
- Back up your systems and data and maintain these backups in an on-site data center, physical or virtual machine (VM), or disk or tape stored on-site.

## 2.2 PROPOSED SYSTEM

The proposed System introduces FogStore-DBaaS, a new data backup system based on Fog Computing. This system utilizes the advantages of Fog-Cloud storage to ensure users' data protection and reliability and, at the same time, overcomes the problems of multi-Cloud using the Fog Computing paradigm. System users can easily and securely backup, restore, and modify their data without caring about the sophisticated operations to protect and secure the data on multi-Cloud storage. Proposed FogDrive to provide an easy-to-use, highly secure, and reliable backup system using state-of-the-art Cloud and encryption techniques

- **FogDrive :** FogDrive is a personal Fog node. It is owned and controlled by the end-user, similar to the rest of his personal devices like smartphones and laptops. FogDrive can be considered a Private Fog device. This is, somehow, similar to the Private Cloud concept, where the organizations deploy and manage their own cloud infrastructure. FogDrive is located within the network architecture. Being a personal device, the FogDrive enables many applications to utilize the Fog Computing paradigm. FogStore benefits from this advantage to provide a better backup experience and a unique system that outperforms the existing systems.
- **FogStore – DbaaS :** FogStore - DBaaS provides a unique data backup system architecture. This uniqueness is derived from the combination of Fog Networking and Multi-Cloud advantages. Whereas Multi-Cloud methods are used to provide the most reliable and secure storage environment, Fog Computing is used to provide better throughput and lower latency for the backup process. The enabler of this unique architecture is the FogDrive fog device.

### **2.2.1 ADVANTAGES**

- Mitigating risk, such as snapshots or mirror copy failure & corruption, human error, hardware failure or poorly documented communication.
- Minimizing downtime: keeping your business online!
- Insuring against natural disasters by having geographic diversity: storing copies of your data in local drive.
- Having the ability to withstand human error or physical impact on a production environment.
- Limit the effect on business or downstream customers.
- Automated backup of critical data and systems.
- Minimal human interaction needed for quick disaster recovery.
- Flexibility to restore the entire infrastructure or just a single application.
- Choose the type, source, target and location of your data
- Determine the best route for continuity – full recovery or repopulating data
- Understand current status with built-in alerting, monitoring and reporting
- The ability to automatically backup critical systems and data.

## **CHAPATER 3**

### **SYSTEM SPECIFICATION**

#### **3.1 Hardware specification**

Processors : Intel® Core™ i5 processor 4300M at 2.60 GHz or 2.59 GHz

8 GB of RAM

Disk space: 320 GB

Operating systems: Windows10

#### **3.2 Software specification**

Server Side	: Python 3.7.4(64-bit)
Client Side	: HTML, CSS, Bootstrap
IDE	: Flask 1.1.1
Back end	: MySQL 5.
Server	: Wampserver 2i
Open PGP packages	:py-pgp

## CHAPTER 4

### SOFTWARE DESCRIPTION

#### 4.1. PYTHON 3.7.4

Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language. It was created by Guido van Rossum during 1985- 1990. Like Perl, Python source code is also available under the GNU General Public License (GPL). This tutorial gives enough understanding on Python programming language.



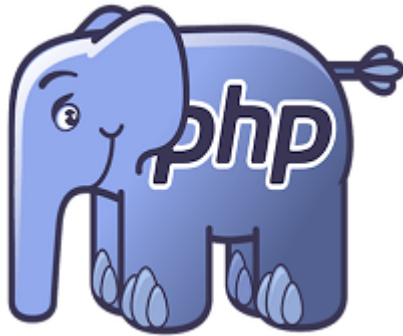
Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages. Python is a MUST for students and working professionals to become a great Software Engineer specially when they are working in Web Development Domain.

Python is currently the most widely used multi-purpose, high-level programming language. Python allows programming in Object-Oriented and Procedural paradigms. Python programs generally are smaller than other programming languages like Java.. Python language is being used by almost all tech-giant companies like – Google, Amazon, Facebook, Instagram, Dropbox, Uber... etc. The biggest strength of Python is huge collection of standard libraries which can be used for the following:

- Machine Learning
- GUI Applications (like Kivy, Tkinter, PyQtetc.)
- Web frameworks like Django (used by YouTube, Instagram, Dropbox)
- Image processing (like OpenCV, Pillow)
- Web scraping (like Scrapy, BeautifulSoup, Selenium)
- Test frameworks

## 4.2. PHP 8.1

- The PHP Hypertext Preprocessor (PHP) is a programming language that allows web developers to create dynamic content that interacts with databases. PHP is basically used for developing web-based software applications. This tutorial helps you to build your base with PHP.



- PHP is a flexible, dynamic language that supports a variety of programming techniques. It has evolved dramatically over the years, notably adding a solid object-oriented model in PHP 5.0 (2004), anonymous functions and namespaces in PHP 5.3 (2009), and traits in PHP 5.4 (2012).
- Object-oriented Programming
- PHP has a very complete set of object-oriented programming features including support for classes, abstract classes, interfaces, inheritance, constructors, cloning, exceptions, and more.
- Functional Programming
- PHP supports first-class functions, meaning that a function can be assigned to a variable. Both user-defined and built-in functions can be referenced by a variable and invoked dynamically. Functions can be passed as arguments to other functions (a feature called Higher-order Functions) and functions can return other functions.
- Meta Programming
- PHP supports various forms of meta-programming through mechanisms like the Reflection API and Magic Methods. There are many Magic Methods available like `__get()`, `__set()`, `__clone()`, `__toString()`, `__invoke()`, etc. that allow developers to hook into class behavior. Ruby developers often say that PHP is lacking method missing, but it is available as `__call()` and `__callStatic()`.
- PHPoC (PHP on Chip): a programming language and an IoT hardware platform.



- PHPoC is a programming language developed based on widely-used PHP language, that makes it become not only a Web development language but also the general-purpose programming language for IoT. The syntax is almost the same as PHP, but adapted for the embedded system. PHPoC inherits almost core functions from PHP. Especially, PHPoC adds new functions, which are used to interact with hardware peripherals such as I/O, UART, I2C, SPI, ADC, TIMER/COUNTER, RTC and so on.
- In other words, PHPoC is an expansion of PHP on a small chip. It takes advantage of the powerful features of PHP and adds extra features to become a powerful programming language for IoT development. PHPoC can be used to not only develop dynamic Web pages, but also to monitor and control PHPoC is the IoT hardware platform that is equipped with PHPoC interpreter and uses PHPoC language for programming. The hardware platform has basic hardware interfaces that interact with sensors/actuators/devices, and the wired LAN and wireless LAN network interfaces that connect to the Internet. It supports many Internet protocol suites. It is designed to be used to create secure IoT devices easily and quickly. electronics.
- Standard PHP Library
- The Standard PHP Library (SPL) is packaged with PHP and provides a collection of classes and interfaces. It is made up primarily of commonly needed data structure classes (stack, queue, heap, and so on), and iterators which can traverse over these data structures or your own classes which implement SPL interfaces.
- Command Line Interface
- PHP was created to write web applications, but is also useful for scripting command line interface (CLI) programs. Command line PHP programs can help automate common tasks like testing, deployment, and application administration.

### 4.3. DATA & DATABASE

Suppose a company needs to store the names of hundreds of employees working in the company in such a way that all the employees can be individually identified. Then, the company collects the data of all those employees. Now, when I say data, I mean that the company collects distinct pieces of information about an object. So, that object could be a real-world entity such as people, or any object such as a mouse, laptop etc.

### DATABASE MANAGEMENT SYSTEM & TYPES OF DBMS

A Database Management System (DBMS) is a software application that interacts with the user, applications and the database itself to capture and analyze data. The data stored in the database can be modified, retrieved and deleted, and can be of any type like strings, numbers, images etc.

### TYPES OF DBMS

There are mainly 4 types of DBMS, which are Hierarchical, Relational, Network, and Object-Oriented DBMS.

- **Hierarchical DBMS:** As the name suggests, this type of DBMS has a style of predecessor-successor type of relationship. So, it has a structure similar to that of a tree, wherein the nodes represent records and the branches of the tree represent fields.
- **Relational DBMS (RDBMS):** This type of DBMS, uses a structure that allows the users to identify and access data *in relation* to another piece of data in the database.
- **Network DBMS:** This type of DBMS supports many to many relations wherein multiple member records can be linked.
- **Object-oriented DBMS:** This type of DBMS uses small individual software called objects. Each object contains a piece of data, and the instructions for the actions to be done with the data.

### STRUCTURED QUERY LANGUAGE (SQL)

SQL is the core of a relational database which is used for accessing and managing the database. By using SQL, you can add, update or delete rows of data, retrieve subsets of information, modify databases and perform many actions. The different subsets of SQL are as follows:

- DDL (Data Definition Language) – It allows you to perform various operations on the database such as CREATE, ALTER and DELETE objects.
- DML (Data Manipulation Language) – It allows you to access and manipulate data. It helps you to insert, update, delete and retrieve data from the database.
- DCL (Data Control Language) – It allows you to control access to the database. Example – Grant or Revoke access permissions.
- TCL (Transaction Control Language) – It allows you to deal with the transaction of the database. Example – Commit, Rollback, Savepoint, Set Transaction.

### 4.3.1. MySQL

MySQL tutorial provides basic and advanced concepts of MySQL. Our MySQL tutorial is designed for beginners and professionals. MySQL is a relational database management system based on the Structured Query Language, which is the popular language for accessing and managing the records in the database. MySQL is open-source and free software under the GNU license. It is supported by Oracle Company. MySQL database that provides for how to manage database and to manipulate data with the help of various SQL queries. These queries are: insert records, update records, delete records, select records, create tables, drop tables, etc. There are also given MySQL interview questions to help you better understand the MySQL database.



MySQL is currently the most popular database management system software used for managing the relational database. It is open- database software, which is supported by Oracle Company. It is fast, scalable, and easy source to use database management system in comparison with Microsoft SQL Server and Oracle Database. It is commonly used in conjunction with PHP scripts for creating powerful and dynamic server-side or web-based enterprise applications. It is developed, marketed, and supported by MySQL AB, a Swedish company, and written in C programming language and C++ programming language. The official pronunciation of MySQL is not the My Sequel; it is My Ess Que Ell. However, you can pronounce it in your way. Many small and big companies use MySQL. MySQL supports many Operating Systems like Windows, Linux, MacOS, etc. with C, C++, and Java languages.

In addition to PHP, MySQL, JavaScript, and CSS, there's actually a fifth hero in the dynamic Web: the web server. In the case of this book, that means the Apache web server. We've discussed a little of what a web server does during the HTTP server/client exchange, but it actually does much more behind the scenes. For example, Apache doesn't serve up just HTML files—it handles a wide range of files, from images and Flash files to MP3 audio files, RSS (Really Simple Syndication) feeds, and more. Each element a web client encounters in an HTML page is also requested from the server, which then serves it up. But these objects don't have to be static files, such as GIF images. They can all be generated by programs such as PHP scripts. That's right: PHP can even create images and other files for you, either on the fly or in advance to serve up later. To do this, you normally have modules either precompiled into Apache or PHP or called up at runtime. One such module is the GD library (short for Graphics Draw), which PHP uses to create and handle graphics.

Apache also supports a huge range of modules of its own. In addition to the PHP module, the most important for your purposes as a web programmer are the modules that handle security. Other examples are the Rewrite module, which enables the web server to handle a varying range of URL types and rewrite them to its own internal requirements, and the Proxy module, which you can use to serve up often-requested pages from a cache to ease the load on the server. Later in the book, you'll see how to actually use some of these modules to enhance the features provided by the core technologies we cover. About Open Source Whether or not being open source is the reason these technologies are so popular has often been debated, but PHP, MySQL, and Apache are the three most commonly used tools in their categories.

What can be said, though, is that being open-source means that they have been developed in the community by teams of programmers writing the features they themselves want and need, with the original code available for all to see and change. Bugs can be found and security breaches can be prevented before they happen. There's another benefit: all these programs are free to use. There's no worrying about having to purchase additional licenses if you have to scale up your website and add more servers. And you don't need to check the budget before deciding whether to upgrade to the latest versions of these products.

## 4.4 WAMPSEVER

WampServer is a Windows web development environment. It allows you to create web applications with Apache2, PHP and a MySQL database. Alongside, PhpMyAdmin allows you to manage easily your database.



WAMPServer is a reliable web development software program that lets you create web apps with MYSQL database and PHP Apache2. With an intuitive interface, the application features numerous functionalities and makes it the preferred choice of developers from around the world. The software is free to use and doesn't require a payment or subscription.

## 4.5. BOOTSTRAP 4

Bootstrap is a free and open-source tool collection for creating responsive websites and web applications. It is the most popular HTML, CSS, and JavaScript framework for developing responsive, mobile-first websites.



It solves many problems which we had once, one of which is the cross-browser compatibility issue. Nowadays, the websites are perfect for all the browsers (IE, Firefox, and Chrome) and for all sizes of screens (Desktop, Tablets, Phablets, and Phones). All thanks to Bootstrap developers -Mark Otto and Jacob Thornton of Twitter, though it was later declared to be an open-source project.

Easy to use: Anybody with just basic knowledge of HTML and CSS can start using Bootstrap Responsive features: Bootstrap's responsive CSS adjusts to phones, tablets, and

desktops Mobile-first approach: In Bootstrap, mobile-first styles are part of the core framework  
Browser compatibility: Bootstrap 4 is compatible with all modern browsers (Chrome, Firefox, Internet Explorer 10+, Edge, Safari, and Opera)

## 4.6. WEB FRAMEWORK

Web Application Framework or simply Web Framework represents a collection of libraries and modules that enables a web application developer to write applications without having to bother about low-level details such as protocols, thread management etc.

### USING AN IDE

As good as dedicated program editors can be for your programming productivity, their utility pales into insignificance when compared to Integrated Developing Environments (IDEs), which offer many additional features such as in-editor debugging and program testing, as well as function descriptions and much more.

#### 4.6.1. FLASK

[Flask](#) is a web framework. This means flask provides you with tools, libraries and technologies that allow you to build a web application. This web application can be some web pages, a blog, a wiki or go as big as a web-based calendar application or a commercial website.



Flask is often referred to as a micro framework. It aims to keep the core of an application simple yet extensible. Flask does not have built-in abstraction layer for database handling, nor does it have formed a validation support. Instead, Flask supports the extensions to add such functionality to the application. Although Flask is rather young compared to most [Python](#) frameworks, it holds a great promise and has already gained popularity among Python web developers. Let's take a closer look into Flask, so-called “micro” framework for Python.

Flask was designed to be easy to use and extend. The idea behind Flask is to build a solid foundation for web applications of different complexity. From then on you are free to plug in any extensions you think you need. Also you are free to build your own modules. Flask is great for all kinds of projects. It's especially good for prototyping.

Flask is part of the categories of the micro-framework. Micro-framework are normally framework with little to no dependencies to external libraries. This has pros and cons. Pros would be that the framework is light, there are little dependency to update and watch for security bugs, cons is that some time you will have to do more work by yourself or increase yourself the list of dependencies by adding plugins. In the case of Flask, its dependencies are:

WSGI-Web Server Gateway Interface (WSGI) has been adopted as a standard for Python web application development. WSGI is a specification for a universal interface between the web server and the web applications.

## **CHAPTER 5**

### **PROJECT DESCRIPTION**

#### **5.1 MODULES :**

##### **5.1.1. Node Backup Module :**

The Node Backup Module is one of the core components of the HAEEdge system. Its primary responsibility is to collect data continuously from cloud servers and store it securely through VirtualBox. In traditional systems, backups are performed manually, which often results in delays, errors, or incomplete backups. This module automates the entire process, ensuring that backups happen at predefined intervals without the need for human involvement. It improves the consistency and reliability of data storage and guarantees that the most recent state of the system is always available.

The automation feature of this module reduces the chances of data loss due to human error or neglect. It actively monitors system changes and triggers a backup process when necessary. The data collected is stored in a local virtual environment, which provides security, isolation, and easy access. VirtualBox enables the creation of snapshots, allowing for multiple restore points. This capability is particularly useful in enterprise environments where continuous operations are critical. Additionally, the backups created by this module are optimized and structured, making them easy to retrieve when needed.

The module ensures that backup files are tagged with metadata, such as timestamp, version, and file type, which supports faster indexing and recovery. It can also integrate with alert systems to notify administrators when a backup fails or is delayed. Another advantage of this module is that it does not overload the system resources. It is designed to run in the background, using minimal CPU and memory, making it suitable even for systems with limited hardware capacity. In a real-time failure situation, this module provides the last known good state of the system, allowing for quick rollback or failover. It supports multiple file formats and works across different operating systems, making it highly adaptable. Overall, the Node Backup Module is essential for data safety and system resilience in cloud-based infrastructures.



### 5.1.2. Virtual Server Module :

The Virtual Server Module operates as a backup system hosted on VirtualBox. It simulates a real backup server environment but in a virtual format, making it lightweight and cost-effective. Edge devices can securely connect to this server and store their backup data. This module ensures that backup operations remain isolated from the production environment. It provides a graphical interface for managing users and data. The virtual server can handle multiple simultaneous backup operations from different devices.

It encrypts data during transmission and storage. This module also ensures version control, maintaining multiple copies of changed files. It's highly secure and can only be accessed by authenticated users. If a node fails, the system can redirect services to the virtual server for continuity. It supports both local and remote backups. The Virtual Server is managed via an admin panel that displays system health, space usage, and session logs. In distributed systems, this module acts as a local hub for edge-level storage. It helps reduce latency during data access. Each data transaction is logged for transparency and audit.

The server can be replicated for high availability. Backup files can be shared across users securely. This module can perform file integrity checks before storage. In case of a disaster, the server allows quick access to the most recent backup. It also integrates with the recovery module to speed up data restoration. The server's virtual nature makes it easy to upgrade or migrate. Custom permissions can be assigned to users or roles. The module is platform-independent and supports multi-format backups. It enhances reliability in large-scale cloud deployments. It ensures data availability even when the main cloud is down. This module provides the core infrastructure for safe, accessible, and fast backup storage.

VirtualBox Software Service module functions as a backup system running on HAEde, providing a simple yet secure backup interface for edge nodes while ensuring complete data protection in distributed cloud storage

### 5.1.3. PGP Data Compression Module

The PGP Data Compression Module is designed to reduce the size of backup data before storing it. This helps save disk space and reduces backup time. The module uses a lossless compression technique, meaning that no data is lost during the process. Once compressed, data is stored securely in the HAEdge environment. It ensures that even large volumes of data can be backed up quickly. This module runs after the data is collected and before it is stored. It supports multiple file types, including text, images, and structured data. The process is automatic and does not require user input. The compression is done in real time without slowing down system performance. Each compressed file is tagged and cataloged for easy retrieval. It significantly reduces network bandwidth during backup transfers.

The compression process focuses on achieving a moderate reduction in data size while ensuring that no distortion is introduced during compression or decompression. Once the data is compressed, it is securely backed up and stored in HAEdge, utilizing minimal storage space while preserving the original data quality.

This module can compress individual files or entire directories. It performs checks to ensure that decompressed data matches the original. It integrates seamlessly with the Node Backup and HAEdge Backup modules. The system monitors storage space and triggers compression when thresholds are reached. The module works silently in the background. Admins can view logs of compressed and saved data. In the event of recovery, the module decompresses data instantly. This ensures fast and reliable access. The system can prioritize critical data for faster compression. All processes are secured with encryption. This module helps reduce storage costs by minimizing space usage. It supports batch processing of files. The algorithm can be updated to improve speed and efficiency.

The system alerts users in case of compression failure. Compressed backups can also be transferred to external storage easily. This module plays a vital role in maintaining a lightweight and scalable backup system.

#### **5.1.4. HAEEdge Backup Module**

This module ensures continuous and automated data backup at regular intervals, enhancing data reliability and disaster recovery capabilities. By utilizing minimal storage space, the HAEEdge Backup module optimizes resource usage while providing seamless access to backed-up data when needed.

The HAEEdge Backup Module automates the regular backup of data at defined time intervals. It runs continuously in the background, making sure that backups are up to date without user involvement. Unlike the Node Backup Module, this one is focused on scheduled, repeated tasks regardless of risk or failure prediction. It provides continuous data protection by saving system states regularly. It ensures that even minor updates to the data are not lost. This module works with compressed and uncompressed data. It automatically tags backups with timestamps and identifiers. This helps during version tracking and data restoration.

The backup frequency can be adjusted based on system usage. It optimizes storage by identifying unchanged data and avoiding duplication. This module works across platforms and supports cloud syncing. It allows admins to configure daily, hourly, or real-time backups. In case of a system failure, the latest scheduled backup is always ready for restoration. It is resource-efficient and consumes minimal power and memory. The system notifies users upon successful or failed backups. This module contributes to achieving compliance with data protection policies. It ensures business continuity in critical environments.

The module creates a log file for each backup session. It supports remote and local storage destinations. The backups are encrypted for added security. Multiple copies can be maintained across locations. The system supports rollback to any backup point. The module is integrated with a monitoring dashboard. It helps administrators plan and optimize storage use. It supports dynamic scheduling based on real-time activity. This module is a key part of the system's redundancy strategy.

### **5.1.5. Disaster Recovery Module :**

The Disaster Recovery Module activates when a cloud or system failure is detected. It is responsible for restoring lost or inaccessible data. It searches metadata stored across various Cloud Service Providers (CSPs) to locate the latest backup. Once found, it reconstructs the data using a backup chain approach. This ensures that even partial backups can be used to rebuild the full data. The module runs recovery scripts and restoration logic automatically. It can restore both system state and user files. It works with the Virtual Server to retrieve backups stored in VirtualBox.

The recovery is fast, reliable, and accurate. It also checks the integrity of restored data. This module helps ensure business continuity with minimal downtime. It sends alerts to the admin when a disaster occurs. Recovery logs are stored for audit purposes. It uses security checks to avoid restoring tampered data. The process can be paused, resumed, or customized based on the scenario. The module uses predefined priorities to recover critical services first. It ensures service migration happens smoothly during cloud outages. The system supports partial and full recovery modes. Backup chain validation ensures correct restoration sequence.

The module is designed to run on low bandwidth if needed. It supports automatic testing of restored data. It works even if one or more CSPs are offline. All restored data is encrypted and verified. It integrates with user authentication to restrict recovery access. This module is essential for resilience in large cloud infrastructures. It ensures users and services continue to function after failure. The module's intelligent design makes it the core of your failover strategy.

## 5.2 Cloud Service Provider Dashboard

In this module we develop a cloud application, or cloud app, it is a software program where cloud-based and local components work together. This model relies on remote servers for processing logic that is accessed through a web browser with a continual internet connection. 986Cloud application servers typically are located in a remote data centre operated by a third-party cloud services infrastructure provider. Cloud-based application tasks may encompass email, file storage and sharing. Third-party data sources and storage services can be accessed with an application programming interface (API).

Cloud applications can be kept smaller by using APIs to hand data to applications or API-based back-end services for processing or analytics computations, with the results handed back to the cloud application. Vetted APIs impose passive consistency that can speed development and yield predictable results. Data stored on cloud services is instantly available to authorized users.

The Cloud Service Provider (CSP) Dashboard is the central module in the architecture where the core functionalities of the cloud-based application are managed. This dashboard acts as a control hub that facilitates communication between cloud infrastructure, local client systems, and remote data centers through web-based access.

A cloud application, or cloud app, is a software system where processing logic is split between client-side (local) and server-side (cloud) components. The cloud app leverages remote servers hosted by a third-party infrastructure provider for performing data-intensive or compute-heavy tasks. These tasks can include email handling, file storage and retrieval, data analytics, backups, and application hosting.

### 5.2.1 Cloud Storage Server

Cloud storage servers are virtual storage facilities provided by cloud service providers that help to store and access multiple files without the requirement of any direct physical device. Web storage server can be accessed via the internet. File Cloud offers the cloud storage in affordable cost and without any downtime. The cloud storage servers continuously run with the help of these data centres and are maintained by the cloud service providers. Data centres secure your files from any kind of damage and make those files available whenever you want to access it via the internet. Applications access cloud storage through traditional storage protocols or directly via an API. Many vendors offer complementary services designed to help collect, manage, secure and analyse data at massive scale. There are three types of cloud data

storage: object storage, file storage, and block storage. Each offers their own advantages and have their own use cases:

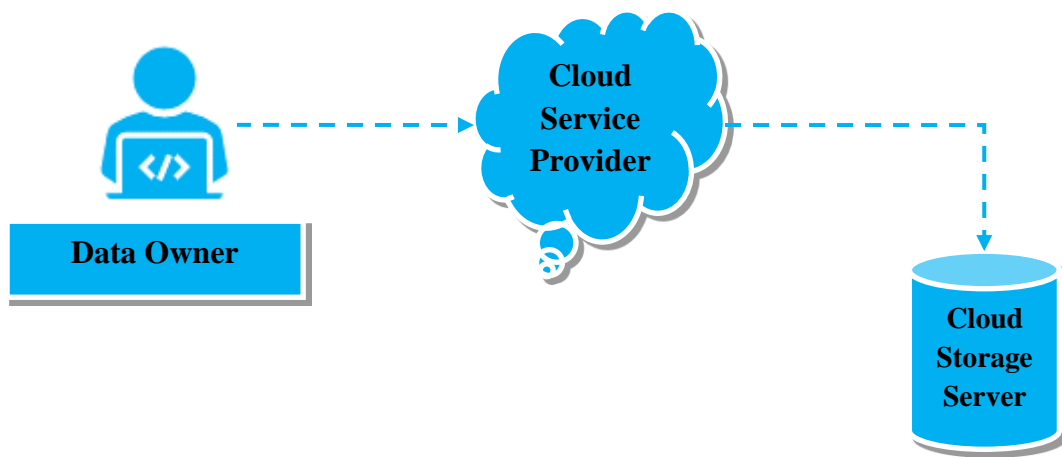
**Object Storage** - Applications developed in the cloud often take advantage of object storage's vast scalability and metadata characteristics. Object storage solutions like Amazon Simple Storage Service (S3) are ideal for building modern applications from scratch that require scale and flexibility, and can also be used to import existing data stores for analytics, backup, or archive.

**File Storage** - Some applications need to access shared files and require a file system. This type of storage is often supported with a Network Attached Storage (NAS) server. File storage solutions like Amazon Elastic File System (EFS) are ideal for use cases like large content repositories, development environments, media stores, or user home directories.

**Block Storage** - Other enterprise applications like databases or ERP systems often require dedicated, low latency storage for each host. This is analogous to direct-attached storage (DAS) or a Storage Area Network (SAN). Block-based cloud storage solutions like Amazon Elastic Block Store (EBS) are provisioned with each virtual server and offer the ultra-low latency required for high performance workloads.

### 5.2.3 HOME DIRECTORIES

The use of home directories for storing files only accessible by specific users and groups is useful for many cloud workflows. Businesses that are looking to take advantage of the scalability and cost benefits of the cloud are extending access to home directories for many of their users. Since cloud file storage solutions adhere to required file system semantics and standard permissions models, customers can easily lift-and-shift applications to the cloud that need this capability.



**Figure 5.2.3 HOME DIRECTORIES**

### 5.2.4 FOG STORE – DBAAS APP LAYER

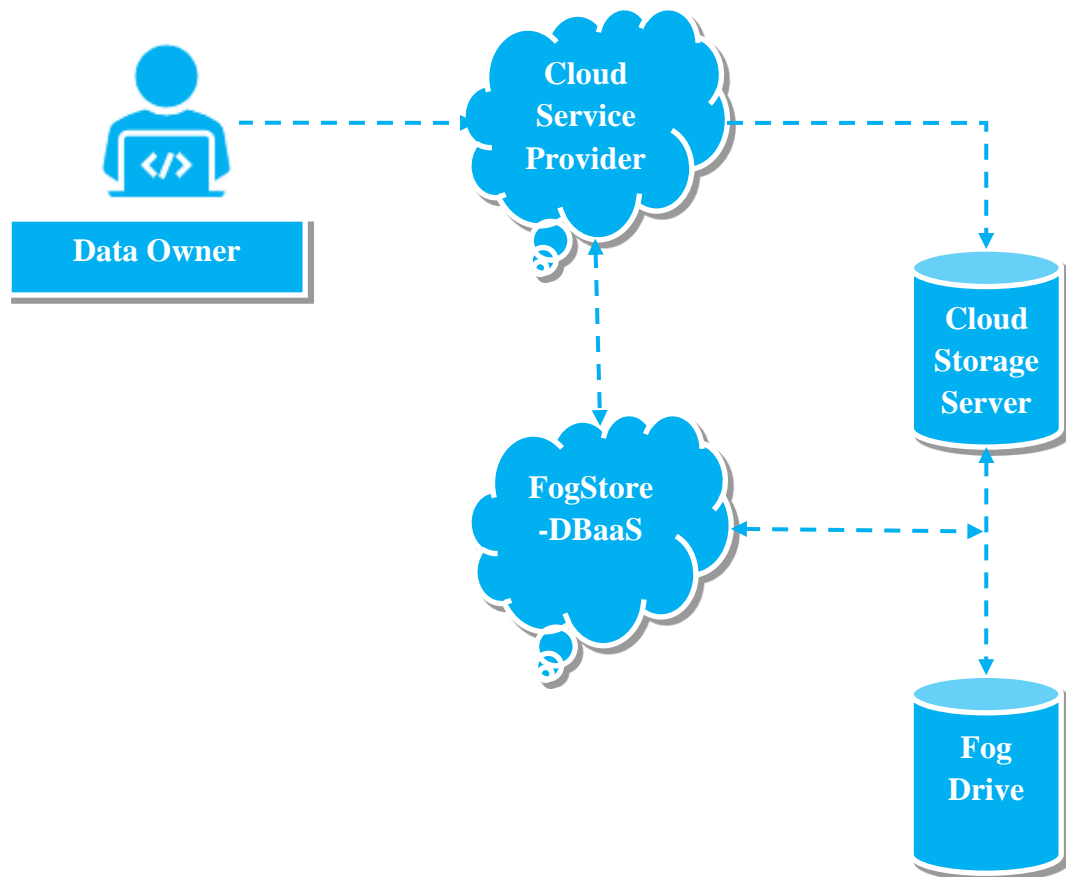
FogStore is a windows-based, free and open-source computer imaging solution for Windows XP, Vista and 7 that ties together a few open-source tools with a php-based web interface. FogStore also supports putting an image that came from a computer with a 80GB partition onto a machine with a 40GB hard drive as long as the data is less than 40GB. Fog also includes a Web Dashboard service that is used to change the hostname of the PC, restart the computer if a task is created for it, and auto import hosts into the FogDrive database and Cloud Storage Server.

- **Backup and Recovery** Backup and recovery is a critical part of ensuring data is protected and accessible, but keeping up with increasing capacity requirements can be a constant challenge. Backing up data using existing mechanisms, software, and semantics can create an isolated recovery scenario with little locational flexibility for recovery.

- **Cloud Data Migration** The availability, durability, and cost benefits of cloud storage can be very compelling to business owners, but traditional IT functional owners like storage, backup, networking, security, and compliance administrators may have concerns around the realities of transferring large amounts of data to the cloud.

### 5.2.5 FOGDRIVE

Archive Storage: Lowest cost. Good for data that can be stored for at least 365 days, including regulatory archives. FogDrive is best implemented on a dedicated server, any spare machine that company have locally.



**Figure 5.2.5 FOGDRIVE**

### 5.2.6 DATA SECURITY AND COMPRESSION

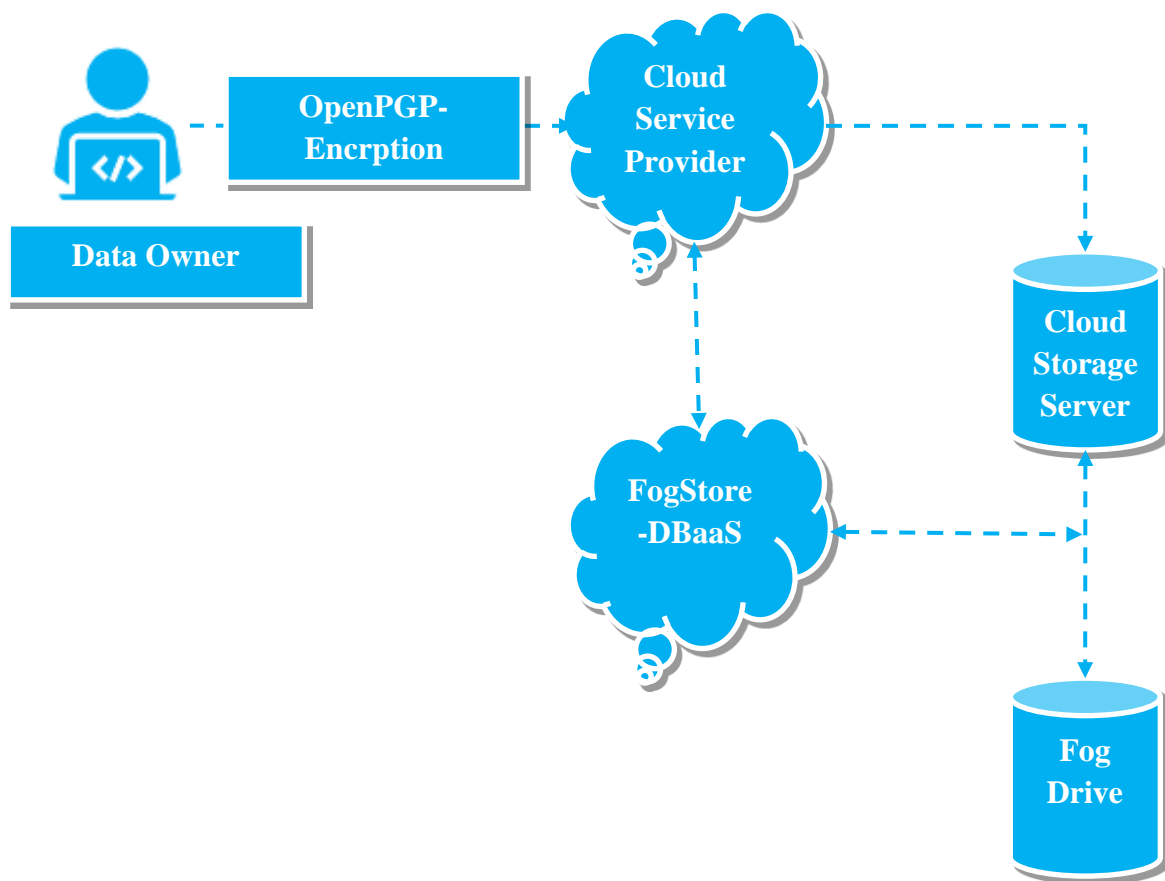
In this module we apply OpenPGP Protocol to encrypt and compress data. OpenPGP is an open standard that aims to secure data in transit by enabling end-to-end encrypted



communication, widely used to secure cloud data and e-mail services and many other types of applications. OpenPGP currently supports modern cryptography and is constantly vetted by renowned security experts.

### 5.2.7 ENCRYPTION AND DECRYPTION

Data management is a central component of OpenPGP. In simple terms, your Data is divided into a pair of public and private PGP keys. The private key must remain a secret and is stored in your device's keyring. The public key can be shared with your correspondents who can send encrypted files to you that can only be decrypted using your private key. In this setup, unfortunately, an attacker who gains control over your private PGP keys can silently impersonate you, accessing your encrypted traffic and creating valid signatures..



**Figure 5.2.7 ENCRYPTION AND DECRYPTION**

### 5.2.8 COMPRESSION

Plaintext is large and takes up an unnecessary amount of modem transmission time and disc space. A PGP program will compress the user's plaintext to make the entire process more efficient. This also reduces patterns found in plaintext, making it harder for hackers to decipher. Significantly Faster ZIP engine – Our advanced multi-core ZIP/ZIPX engine has been

optimized for maximum speed, now up to 30-50% faster than WinZip's multi-core engine (and much faster than Secure ZIP's and WinRAR's ZIP engines), while providing similar compressing strength.

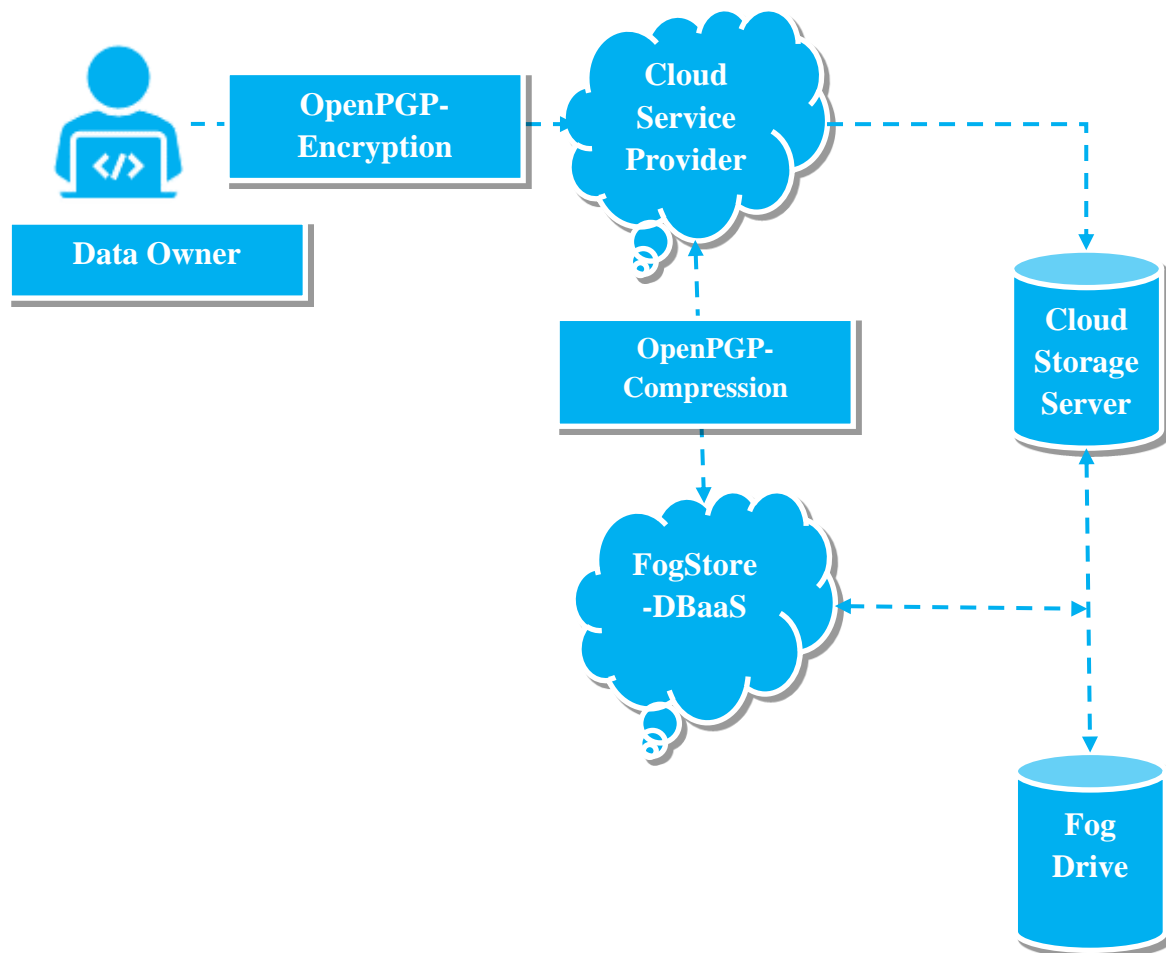


Figure 5.2.8 COMPRESSION

## 5.3 SYSTEM DESIGN

### 5.3.1 System Architecture – Fog Drive Data Backup

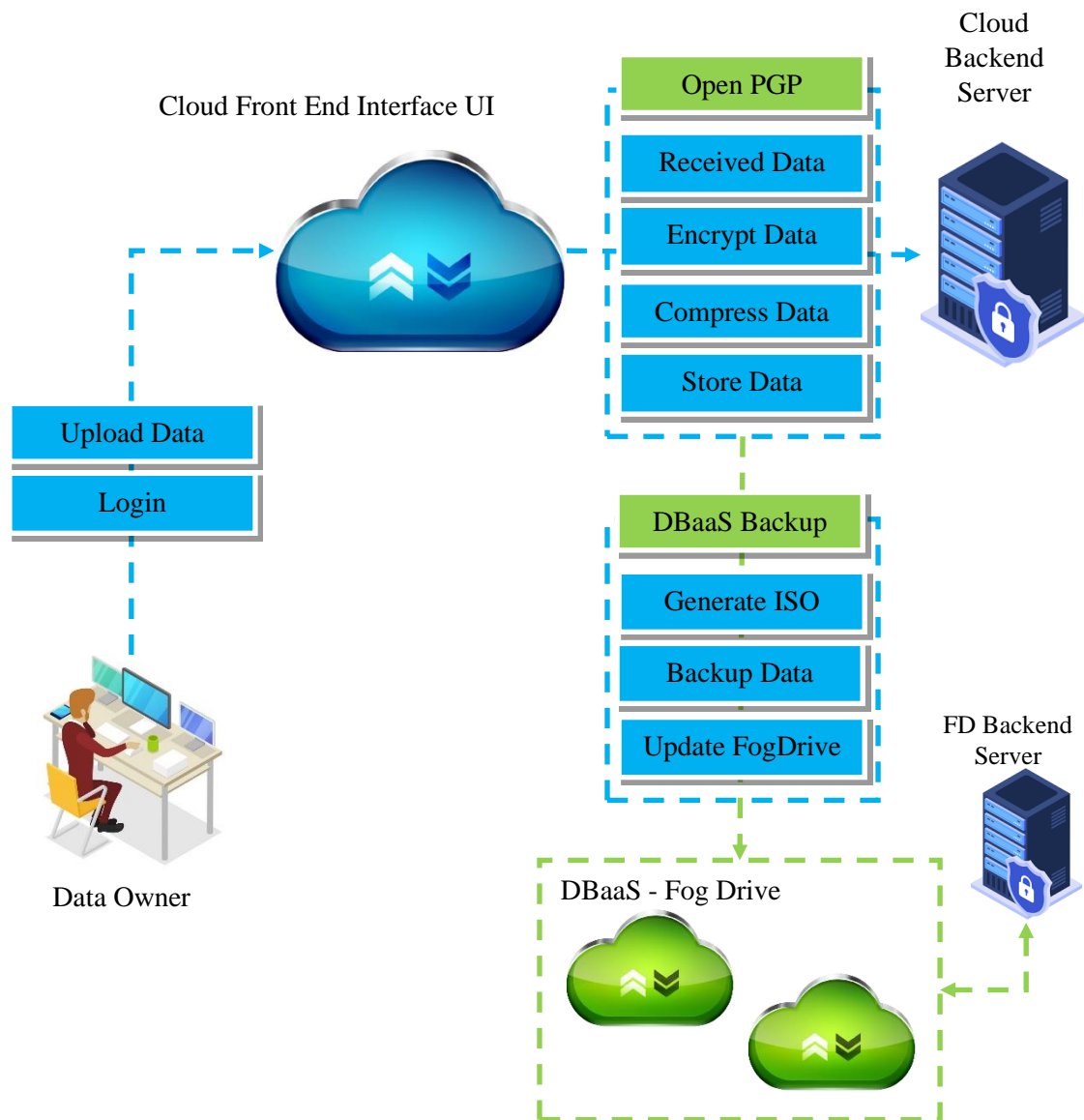


Figure 5.3.1 System Architecture – Fog Drive Data Backup

### 5.3.2 System Architecture – DBaaS Data Recovery

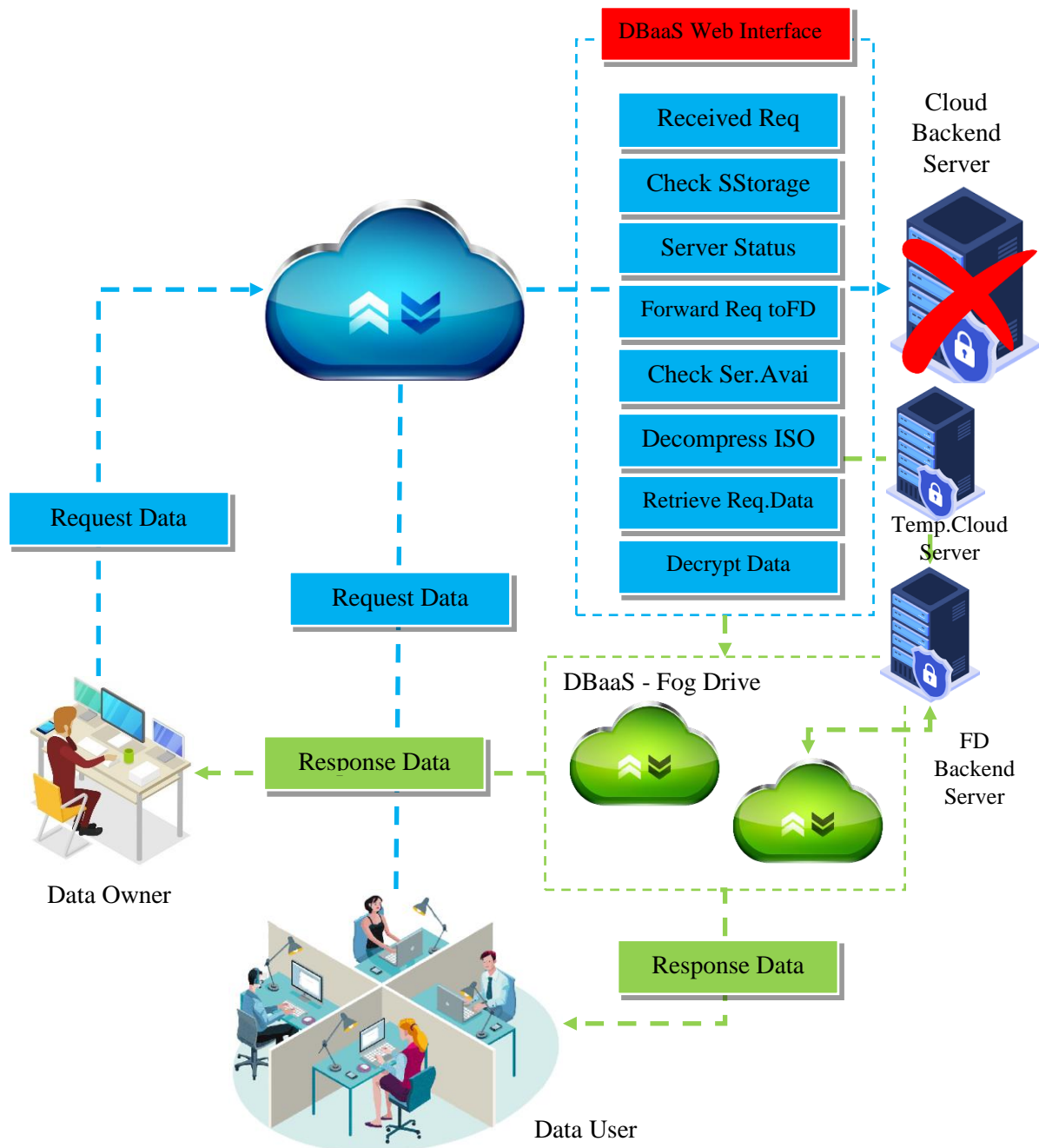
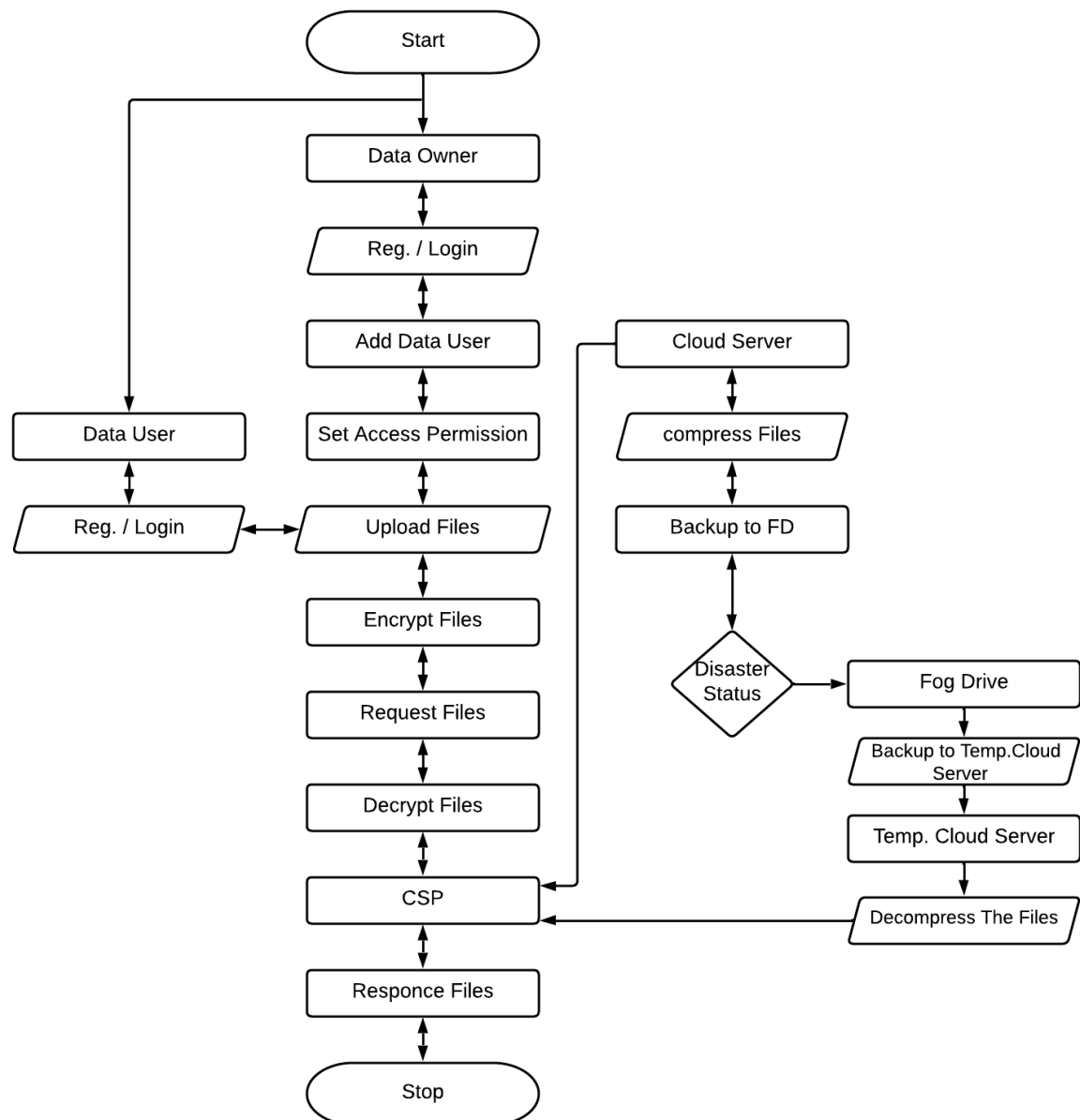


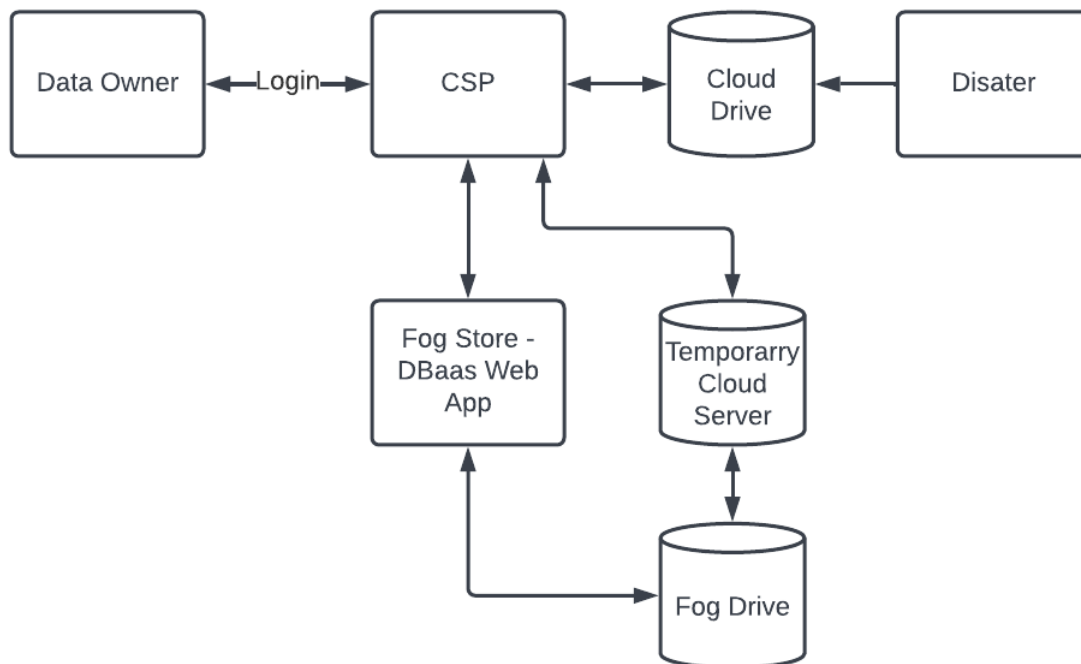
Figure 5.3.2 System Architecture – DBaaS Data Recovery

## 5.4 Flow Chart

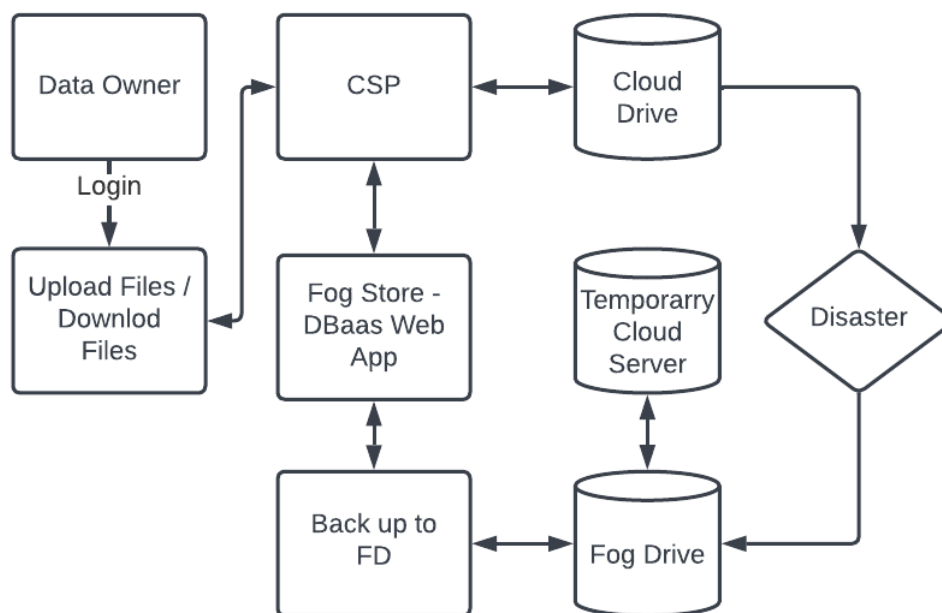


**Figure 5.4 Flow Chart**

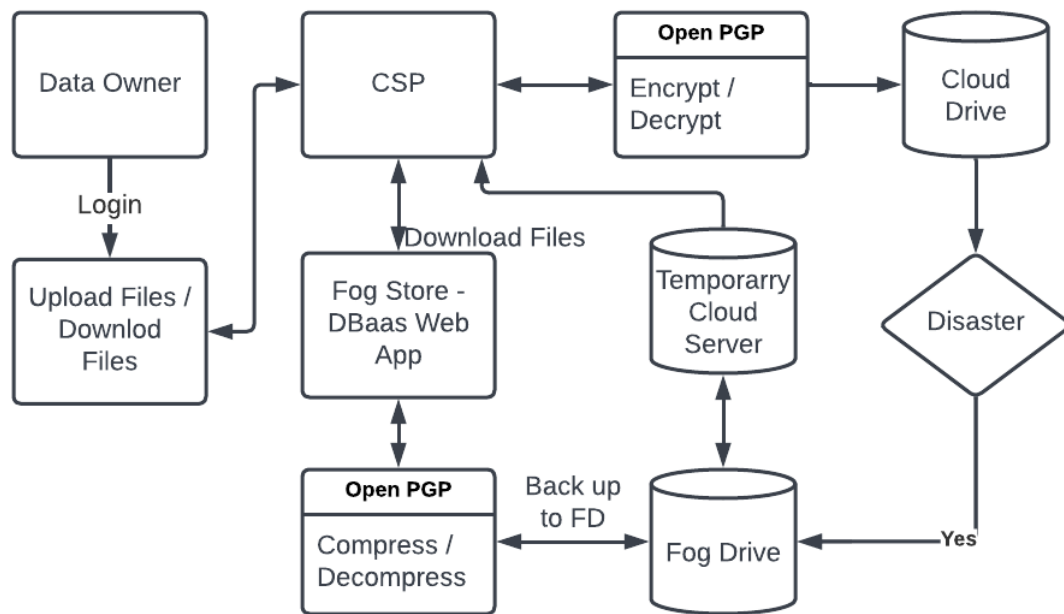
### 5.4.1 Data Flow Diagram – Level 0



### Data Flow Diagram – Level 1

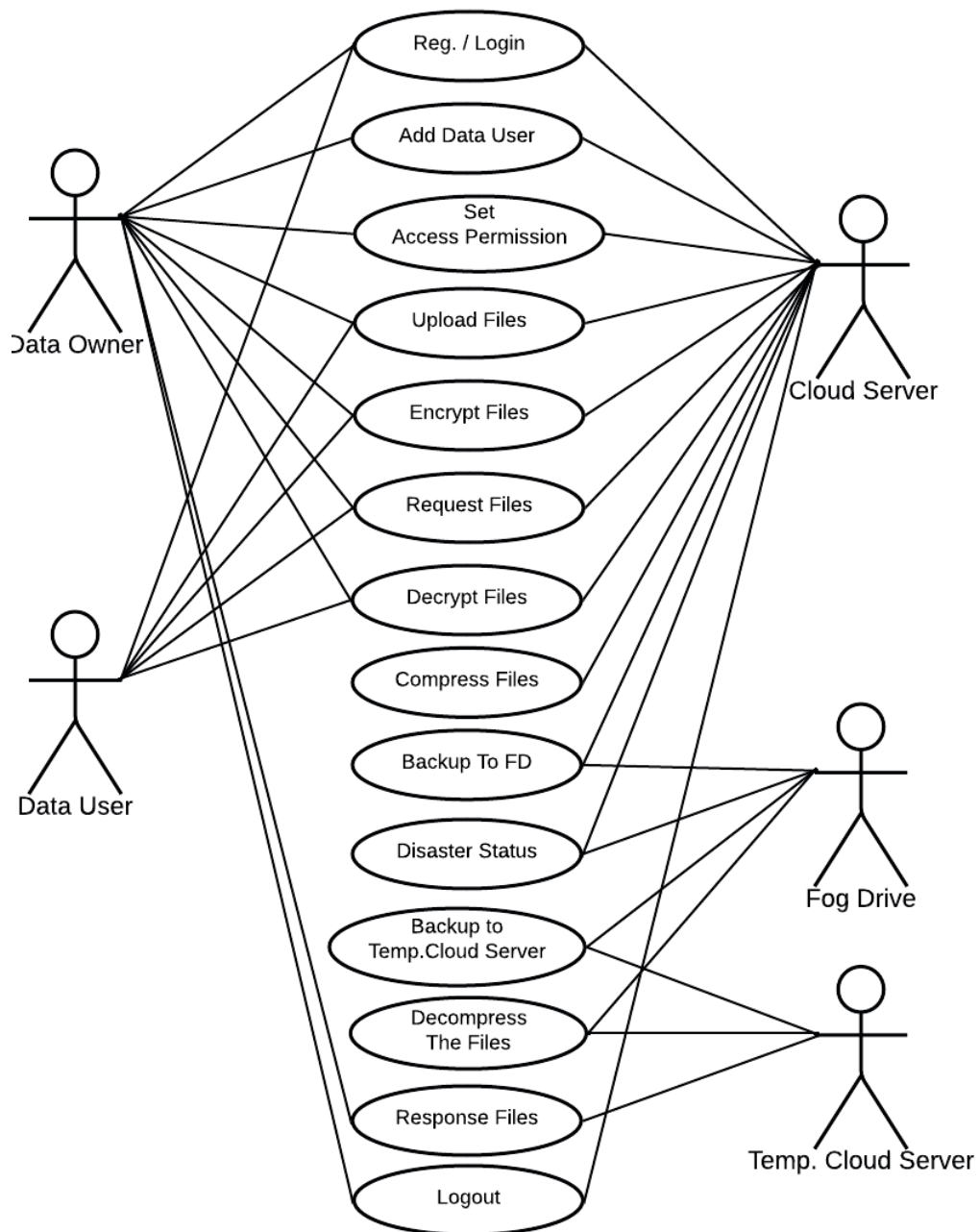


## Data Flow Diagram – Level 2



## 5.6. USE CASE

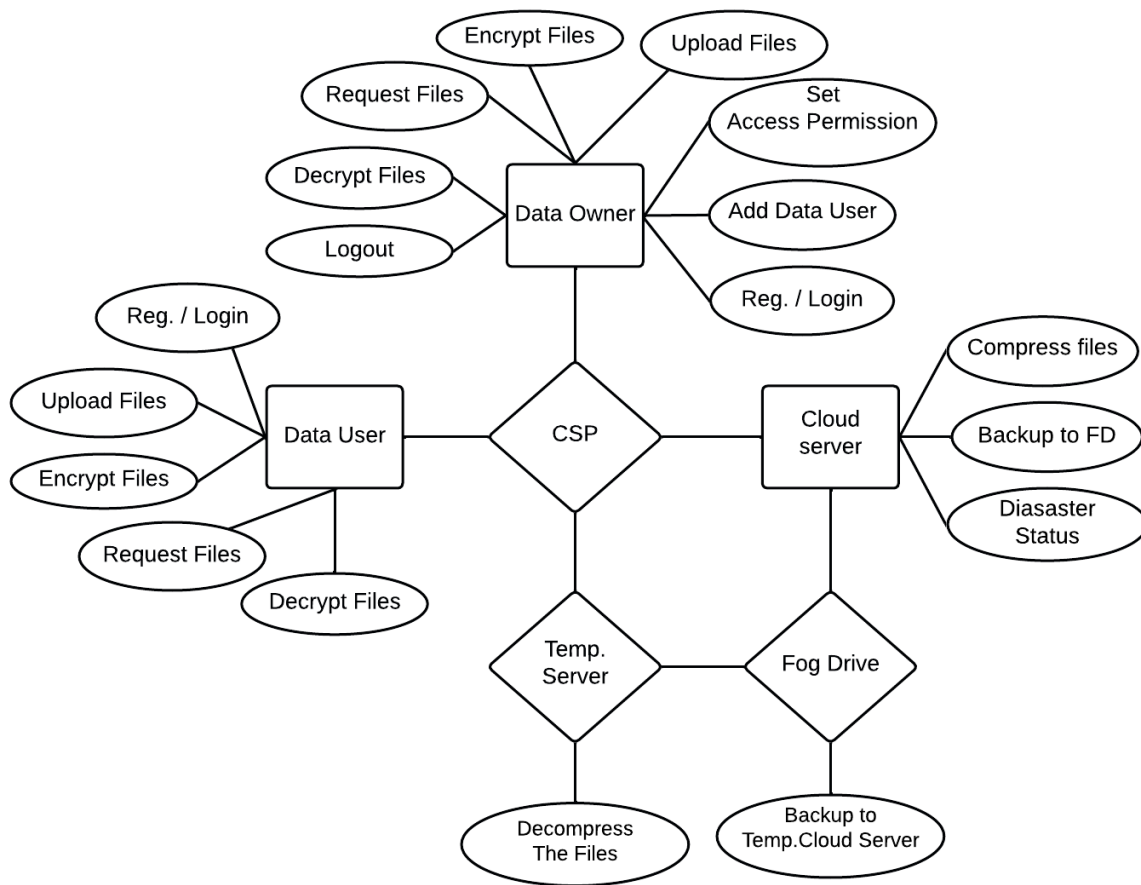
### UML DIAGRAMS



**Figure 5.6. USE CASE**



## 5.6 ER DIAGRAMS



**Figure 5.6 ER DIAGRAMS**

## **CHAPTER 6**

### **SYSTEM TESTING**

In this phase of methodology, testing was carried out on the several application modules. Different kind of testing was done on the modules which are described in the following sections. Generally, tests were done against functional and non-functional requirements of the application following the test cases. Testing the application again and again helped it to become a reliable and stable system.

#### **6.1 USABILITY TESTING**

This was done to determine the usability of the application that was developed. This helped to check whether the application would be easy to use or what pitfalls would the users come through. This was used to determine whether the application is user friendly. It was used to ascertain whether a new user can easily understand the application even before interacting with it so much. The major things checked were: the system flow from one page to another, whether the entry points, icons and words used were functional, visible and easily understood by user.

Usability testing was conducted to evaluate how effectively users can interact with the FogDrive application and to ensure that the system is intuitive, responsive, and user-friendly. The primary aim was to assess whether both new and experienced users could navigate the application with minimal instruction and complete basic operations successfully. This process helped in identifying any interface design flaws or functional ambiguities that could affect user experience.

The testing focused on several core components of the application including the user login page, service provider login, data owner dashboard, file upload interface, and the data retrieval module. Key aspects observed during testing were navigation flow, button placements, clarity of labels, icon usage, visibility of interface elements, system feedback (like alerts and success messages), and error-handling mechanisms.

## 6.2 FUNCTIONAL TESTING

Functional Testing is defined as a type of testing which verifies that each function of the software application operates in conformance with the requirement specification. This testing mainly involves black box testing and it is not concerned about the source code of the application. Functional tests were done based on different kind of features and modules of the application and observed that whether the features are met actual project objectives and the modules are hundred percent functional. Functional tests, as shown in the following Table-1 to Table-5, were done based on use cases to determine success or failure of the system implementation and design. For each use case, testing measures were set with results being considered successful or unsuccessful. Below are the tables which are showing some of the major test cases along with their respective test results.

Functional testing was performed to verify that every component of the FogDrive application operates according to the specified requirements. This testing phase focused on examining whether the software performs its intended functions correctly under various conditions. The main goal was to ensure that each feature — from login to backup and recovery — works as expected, and that all inputs produce the correct outputs.

The testing covered the core functionalities of the application including user registration, login validation, file upload, file compression, backup storage, data retrieval, dashboard interactions, and logout processes. Each function was tested with both valid and invalid inputs to observe the system's response and verify its ability to handle different scenarios.

During the login module testing, checks were made to confirm whether valid credentials allowed access and invalid inputs returned proper error messages. For the signup module, validations for email format, duplicate usernames, password strength, and mobile number format were tested. Similarly, the dashboard functionalities were tested to ensure that uploaded files were properly listed, download buttons worked, and user stats updated correctly.

### 6.3 SYSTEM TESTING

In this phase of methodology, testing was carried out on the several application modules. Different kind of testing was done on the modules which are described in the following sections. Generally, tests were done against functional and non-functional requirements of the application following the test cases. Testing the application again and again helped it to become a reliable and stable system.

System testing was conducted to evaluate the entire FogDrive application as a unified system. The purpose of this phase was to ensure that all modules — including backup, compression, user management, and disaster recovery — work together seamlessly and deliver the expected output under real-world conditions. Unlike functional testing, which focuses on individual components, system testing validates the complete integration and end-to-end behavior of the application.

The testing process included executing a series of real-use scenarios from start to finish, such as user registration, file backup, compression, recovery, and data access. The system was tested on different platforms and browsers to check compatibility, responsiveness, and overall performance. Every component interaction was carefully observed to ensure that data passed correctly from one module to another and that the user flow was uninterrupted.

The compression and decompression process was verified in sequence with data retrieval to confirm that files stored in compressed format could be restored without any loss or corruption. Errors were intentionally introduced during the backup to verify if the system generated proper alerts and logs. The system's ability to manage multiple users at the same time was also tested by logging in from different accounts and devices concurrently. All the expected outputs were matched with actual results, and the application's behavior was logged and analyzed for accuracy, efficiency, and consistency. It was also ensured that system resources like memory and CPU were optimally utilized during operations, and that the system did not crash under stress.

## 6.4 UNIT TESTING

Before you can test an entire software program, make sure the individual parts work properly on their own. Unit testing validates the function of a unit, ensuring that the inputs (one to a few) result in the lone desired output. This testing type provides the foundation for more complex integrated software. When done right, unit testing drives higher quality application code and speeds up the development process. Developers often execute unit tests through test automation.

Unit testing is the process of testing individual components or functions of the application in isolation to ensure that each unit works as intended. In the FogDrive system, unit testing was performed early in the development cycle for every distinct function, including user input validation, file compression, data upload, login verification, backup scheduling, and file restoration. The primary goal was to identify and fix bugs at the code level before integrating components together.

Each function was tested independently with valid, invalid, and edge-case inputs. For example, in the user signup module, the logic that checks for proper email format, strong password enforcement, and unique usernames was tested separately. Similarly, the unit responsible for compressing files was tested to ensure it successfully reduced file size without data loss and could decompress it accurately.

The login module was tested to confirm that it correctly authenticated users, displayed proper error messages for wrong inputs, and redirected them to their respective dashboards. The dashboard units were tested to verify that they properly pulled user-specific data, such as uploaded files and statistics, from the database. The upload module's units were validated to ensure it accepted supported file formats, rejected corrupt files, and updated the file list accurately.

## 6.5 INTEGRATION TESTING

Integration testing is often done in concert with unit testing. Through integration testing, QA professionals verify that individual modules of code work together properly as a group. Many modern applications run on microservices, self-contained applications that are designed to handle a specific task. These microservices must be able to communicate with each other, or the application won't work as intended. Through integration testing, testers ensure these components operate and communicate together seamlessly.

Integration testing was conducted to ensure that the various modules of the FogDrive system function correctly when combined together. While unit testing focuses on individual components in isolation, integration testing validates the interaction between modules — checking whether data flows properly and that features work seamlessly across module boundaries.

The FogDrive application consists of several interconnected modules: the login system, user management, file upload, data compression, backup scheduling, virtual server storage, and the disaster recovery module. During integration testing, each of these components was linked and tested in the sequence in which users would typically interact with them.

Tests were also conducted to ensure that login tokens and user sessions remained valid when users navigated between different modules. The interaction between the Node Backup Module and the HAEEdge Backup Scheduler was tested to confirm that real-time and periodic backups worked without conflict. Similarly, the coordination between the Compression Module and Storage Module was checked to ensure compressed files were properly handled and did not corrupt.

## 6.6 TEST CASES

### 6.6.1 Signup/Registration Test Case

<b>Identifier</b>	Test Case-1
<b>Test Case</b>	Signup
<b>Description</b>	To register new account in the application.
<b>Pre-requisite</b>	1) Username and email must not exist previously.
<b>Test procedure</b>	1) Select Sign Up from the menu. 2) Fill in username, email, and password and retype password accordingly. 3) Click on Sign Up button
<b>Expected</b>	1) User can register to the application successfully.
<b>Result</b>	2) Username, email and password stored in the user table in the database.
<b>Pass/Fail</b>	Pass

### 6.6.2 Login Test Case

<b>Identifier</b>	Test Case-2
<b>Test Case</b>	Login
<b>Description</b>	To login new account in the application
<b>Pre-requisite</b>	1) Registration must be done previously.

<b>Test procedure</b>	1) Select Log In from the menu. 2) Fill in username and password accordingly. 3) Click on Log In button.
<b>Expected</b>	1) User can login to the application successfully.
<b>Result</b>	2) User should access the application features which are allowed
<b>Pass/Fail</b>	Pass



## **CHAPTER 7**

### **SYSTEM IMPLEMENTATION**

#### **7.1 OVERVIEW**

System implementation is the phase where the designed and developed FogStore-DBaaS solution is deployed in a real-world environment.

It involves setting up the cloud service provider, configuring FogDrive local nodes, securing data transmission through OpenPGP encryption, and establishing robust backup and recovery operations.

The primary objective of this stage is to ensure that the theoretical models, algorithms, and workflows are converted into a fully functional, efficient, and secure operational system.

The implementation process begins with setting up the Cloud Service Provider (CSP) interface, which allows data owners and service providers to interact with the system. This step includes provisioning virtual machines, configuring storage environments, and integrating the necessary APIs for cloud communication. In parallel, the FogDrive nodes — representing local edge storage points — are installed and connected to form a hybrid cloud-fog architecture.

To ensure data confidentiality during backup and transfer operations, OpenPGP encryption is configured across all communication channels. This encryption guarantees that sensitive user data is protected from unauthorized access during transit and while stored on both fog and cloud environments. System roles (Data User, Data Owner, and Service Provider) are also implemented with proper access control and authentication layers.

Each module was integrated and tested to ensure seamless operation, including role-based dashboard access, file upload/download services, and status monitoring. Logging and alert systems were also enabled for backup success, failure, and file recovery events.

The implementation also addressed storage optimization by enabling deduplication and compression before storage and decompression during retrieval. Database connectivity, user session handling, and interface responsiveness were tested and confirmed to perform efficiently under real-world conditions.

## **7.2 IMPLEMENTATION STRATEGY**

The implementation of the FogStore-DBaaS system was carefully planned and executed in multiple steps to minimize errors and ensure smooth operation.

The major phases in the implementation include:

### **7.2.1 Infrastructure Setup:**

Deployment of Cloud Storage Servers and configuration of FogDrive nodes using VirtualBox. Installation of WAMP Server, MySQL, and Flask-based backend APIs. The first phase involved setting up the system's infrastructure.

This included the deployment of Cloud Storage Servers and the configuration of FogDrive nodes using VirtualBox. VirtualBox provided the ability to simulate multiple virtual environments that act as local fog nodes. These nodes were linked to the cloud layer to enable hybrid data backup and recovery operations. In addition, the WAMP server (Windows, Apache, MySQL, PHP) was installed to serve as the local development and testing environment. MySQL was used as the database system to manage user records, file information, and system logs, while Flask, a Python-based web framework, was used to develop and serve the backend APIs for system operations.

### **7.2.2 Data Backup Integration:**

Development of modules for data upload, encryption using OpenPGP, ISO generation, and compressed storage in FogDrive.

Once the infrastructure was in place, the next step was integrating the data backup functionality. This included the development of modules for secure file uploads, applying OpenPGP encryption to maintain data confidentiality, generating ISO images for data storage, and compressing the files before storing them in FogDrive. Compression ensured storage space was optimized, while ISO generation helped in structuring and managing backup files efficiently.

Each file was encrypted before backup, ensuring end-to-end data security even before leaving the client device.

### **7.2.3 Recovery Workflow Configuration:**

Design of the retrieval system allowing users to recover data from FogDrive in case of a disaster or cloud failure.

A critical part of the implementation strategy was setting up the data retrieval and recovery workflow. The disaster recovery module was configured to allow users to retrieve backed-up files from FogDrive in case of cloud failures or data corruption.

This involved setting up metadata-based search systems to locate files, reconstruct backup chains, and restore the latest valid file version from either the cloud or fog layer. Special logic was added to prioritize high-risk files and provide rapid access during emergencies.

#### **7.2.4 Security Layer Implementation:**

Public-Private Key encryption through OpenPGP. Secure session management and user authentication at both cloud and Fog layers.

security was a top priority during implementation. OpenPGP encryption was used with a Public-Private Key infrastructure to ensure secure communication and data handling throughout the system. In addition to encrypting files, secure session management was implemented using user authentication tokens and hashed credentials to prevent unauthorized access. Both cloud and fog layers included authentication protocols and access control policies to restrict actions based on user roles (Data User, Data Owner, Service Provider).

Each of these implementation phases was validated through rigorous testing, and the system was refined based on test results to ensure seamless performance in live conditions. By adopting a modular and secure strategy, the FogStore system was implemented successfully, delivering a reliable, secure, and scalable cloud-fog data backup solution.

### **7.3 KEY IMPLEMENTATION COMPONENTS**

#### **7.3.1 Cloud Service Provider Module**

This module manages user accounts, data uploads, cloud storage monitoring, and initial backup tasks. APIs allow data to be transmitted securely and efficiently to the Cloud Storage Server.

This module functions as the backbone of the cloud layer. It handles the creation and management of user accounts and facilitates secure file uploads from data users. The Cloud Service Provider (CSP) is also responsible for monitoring the storage capacity, backup schedules, and user interactions with cloud resources. APIs developed using Flask are employed to transmit data securely between the user interface and cloud storage.

These APIs are equipped with authentication and encryption layers to ensure data integrity and privacy. The CSP dashboard also allows administrators to view usage statistics, approve or reject user registration requests, and monitor the status of all uploaded files. The module ensures high availability and scalable performance in cloud environments.

### 7.3.2 FogDrive Backup Node

FogDrive acts as a secondary, local storage system. It is configured to receive periodic encrypted backups from the cloud servers, ensuring that in the event of a disaster, a reliable recovery source is readily available.

FogDrive acts as a localized storage node that works alongside the cloud. It serves as a secondary backup system, ensuring continuity of access in case the cloud fails or becomes temporarily unreachable. Configured using VirtualBox, FogDrive receives periodic encrypted backups from the cloud servers. These backups are triggered either manually or through automated scheduling configured in the HAEEdge system. Because the data is stored closer to the user, recovery operations from FogDrive are faster and less dependent on internet availability.

The node is isolated from the main production environment to avoid accidental tampering or deletion. FogDrive significantly enhances disaster recovery capabilities by acting as an immediate and trusted recovery source.

### 7.3.3 Encryption and Compression Module

The OpenPGP encryption ensures that data is protected during both transmission and storage phases. Compression techniques are applied to optimize storage efficiency and reduce recovery time.

Security and efficiency are central to the system's design. This module uses OpenPGP encryption to protect data during both transmission and storage. It utilizes public-private key cryptography to ensure that only authorized users can encrypt and decrypt the data. This prevents unauthorized access and ensures compliance with data protection standards. In addition to encryption, data is passed through a compression engine before being stored.

Compression reduces file size significantly, optimizing storage usage and reducing the bandwidth required for backup and recovery operations. The combination of encryption and compression ensures both data confidentiality and storage efficiency.

### 7.3.4 Backup and Recovery Operations

Backup operations involve encrypting, compressing, and uploading data, while recovery operations involve decrypting, decompressing, and restoring user data upon request.

This component is responsible for executing the core functions of the FogStore system. Backup operations include encrypting files using OpenPGP, compressing them, and uploading them to both the cloud and FogDrive nodes. These tasks are managed automatically based on

a defined schedule, although users can also trigger backups manually. Recovery operations begin when a user requests to restore a file.

The system decrypts the file, decompresses it, and provides a downloadable copy to the user. If the cloud server is unavailable, the system switches to the FogDrive node for recovery, ensuring minimal downtime. This module is tightly integrated with user authentication to ensure only valid users can perform sensitive actions like data restoration.

## **7.4 IMPLEMENTATION PROCEDURES**

The procedures followed during the system implementation are:

### **7.4.1 Deployment of Servers:**

Installation of WAMP Server for backend services and MySQL for database operations. The deployment of servers began by selecting the appropriate hardware and software to support the system's operations. We chose to install WAMP Server as the backend server, which included Apache for handling HTTP requests, MySQL for database management, and PHP for dynamic web content. The WAMP stack allowed us to create a consistent development and production environment for seamless integration of both front-end and back-end components.

MySQL was installed separately to ensure secure and efficient data storage for user data, system logs, and backup files. Database configurations were tuned to optimize performance, ensuring high availability and quick retrieval of user requests. The WAMP Server was configured to handle various API requests from the front-end application through Flask. Additionally, security protocols such as SSL certificates were implemented to encrypt data transmitted between the user interface and backend servers, mitigating any risks of data breaches.

Once the WAMP stack and MySQL were installed, further steps included setting up automated backups and integrating the database with the web application for real-time data exchange. We made sure the deployment of servers adhered to best practices for performance optimization, minimizing latency, and maximizing uptime. Continuous monitoring tools were installed to track server health, system resource consumption, and traffic patterns.

### 7.4.2 Configuration of VirtualBox:

Virtual machines acting as FogDrive nodes were set up for local backups. or the setup of the virtual environment, VirtualBox was selected to simulate fog computing nodes. Each node represented a local backup system that mimicked the real-world deployment scenario, where files and backups are stored at distributed locations. VirtualBox enabled the creation of multiple virtual machines that acted as independent fog storage nodes.

The virtual machines (VMs) were configured with appropriate resource allocations such as CPU cores, RAM, and storage space to simulate different fog nodes of varying capacities. These nodes were connected in a virtual network to replicate a decentralized cloud environment, with file backup requests routed to the closest available node based on proximity and system load. The configuration also included setting up network protocols for inter-node communication, ensuring that data could be shared seamlessly between fog nodes during backup and recovery processes.

The configuration process also focused on ensuring security across nodes, with virtual firewalls and encrypted tunnels between each node to safeguard data during transmission. Each VM was equipped with a local MySQL database for storing metadata related to backups, including file identifiers, timestamps, and data integrity checks.

After setting up the nodes, the next task was to ensure redundancy by establishing replication between virtual machines, ensuring that in the event of a node failure, data could still be retrieved from a secondary node. The use of VirtualBox as a testbed provided a cost-effective yet robust platform for simulating fog storage before deploying the actual system in a production environment.

### **7.4.3 Backend and Frontend Setup:**

Flask APIs for server-side operations and Bootstrap with HTML/CSS for user-friendly cloud interfaces. For the back-end setup, Flask was chosen due to its flexibility and lightweight nature, allowing us to build efficient RESTful APIs for communication between the server and the front-end application. Flask provided a robust framework for managing incoming requests, handling user authentication, and processing backup operations. We integrated the backend with the MySQL database, enabling data storage, retrieval, and management of backup metadata, including file details and user account information.

We also implemented error handling and logging mechanisms within the Flask APIs to ensure smooth operation and easier troubleshooting. Security measures, such as input validation and prevention of SQL injection, were enforced to protect the system from common vulnerabilities. The backend APIs communicated with the fog nodes, performing tasks like backup initiation, file transfer, and status updates in real-time.

On the front-end, the user interface was designed with simplicity in mind, utilizing Bootstrap to create a responsive, mobile-friendly application that could be easily navigated by end users. HTML and CSS were used to structure and style the application, ensuring a smooth, intuitive experience for users. Key features included easy access to backup initiation, file management options, and recovery operations, allowing users to securely backup and restore their files with minimal effort.

We made sure that the front-end and back-end were tightly integrated, ensuring that user actions on the interface would immediately reflect in the server-side operations, and feedback from the server would be shown to the user in real-time. This seamless interaction was achieved through Ajax calls that allowed for asynchronous data exchange between the client and server, improving user experience by reducing waiting times.

### **7.4.4 Encryption Key Management:**

Generation and secure distribution of OpenPGP public and private keys for encryption and decryption tasks. The implementation of encryption key management was a critical aspect of securing user data. OpenPGP encryption was selected to protect backup files and sensitive information. The process began with the generation of both public and private keys for encryption and decryption tasks.

Each user received a unique pair of keys, with the public key used for encrypting data before backup, and the private key used by the server to decrypt the files when restoration was requested.

#### **7.4.5 Testing Phase Execution:**

Comprehensive testing across all modules, including simulated disaster recovery scenarios. The testing phase was one of the most critical steps in the implementation process. We began by executing unit tests for each module to ensure that individual components, such as the backend APIs, encryption functionalities, and user interfaces, worked as expected in isolation. These tests included checking for correct data handling, response times, and interaction between different modules.

#### **7.4.6 User Training and Documentation:**

Basic training for end users on system navigation, backup initiation, and file recovery operations. Once the system was implemented and thoroughly tested, the next step was to provide basic training for end-users. The training focused on explaining the core features of the system, such as how to initiate a backup, manage backup files, and restore lost data. We conducted interactive training sessions, providing users with hands-on experience of the system in a test environment, allowing them to familiarize themselves with the interface and functionalities.



## CHAPTER 8

### 8.1 CONCLUSION

As important network infrastructures to support data storage and service delivery for worldwide users, cloud data centers are facing great threaten by frequent disasters around the world and thus the survivability of cloud data centers becomes a critical issue. This project introduces FogStore -Disaster Backup as a Service and FogDrive, a new data backup system based on Cloud and Fog Computing. This system utilizes the advantages of Temporary-Cloud storage to ensure users' data protection and reliability and, at the same time, overcomes the problems of multi-Cloud using the Fog Computing paradigm. System users can easily and securely backup, restore, and modify their data without caring about the sophisticated operations to protect and secure the data on Temporary-Cloud storage. Extensive numerical results demonstrate the efficiency of the proposed scheme on improving survivability of data and services in cloud data centers. With a set of given resource and early warning time constraints, this work can guide data center operators to achieve a tradeoff between data backup and service migration.

Through this solution, users can effortlessly backup, restore, and modify their data, all while remaining confident that sophisticated encryption and security mechanisms are safeguarding their information. This approach reduces the operational burden on users, as they no longer need to manage the complexities of securing their data across multiple platforms or worry about potential data loss in the event of disasters.

The proposed system has been thoroughly tested, and the results highlight its effectiveness in improving the survivability of data and services within cloud data centers. The numerical results indicate significant enhancements in terms of system reliability, recovery time, and overall disaster preparedness. Furthermore, the system's flexible architecture allows data center operators to achieve an optimal balance between data backup and service migration, taking into account given resource constraints and early warning times. This flexibility helps operators design resilient systems that can efficiently handle disaster scenarios while minimizing downtime and data loss.

## 8.2 Future Enhancement

In future, we would like to enhance this FogStore over the resource limited edge devices like the IoT end devices.

**Blockchain for Data Integrity :** Incorporating blockchain technology can ensure data integrity by maintaining an immutable and transparent record of all data modifications, securing critical system configurations and backups.

**Automated Recovery with AI :** Integrating AI-driven decision-making into the recovery process can automate data restoration and system repair, significantly reducing downtime and improving overall system reliability.

## CHAPTER 9

### 9.1 SOURCE CODE

```
from flask import Flask

from flask import Flask, render_template, Response, redirect, request, session, abort,
url_for

from camera import VideoCamera

@app.route('/verify_face2',methods=['POST','GET'])

def verify_face2():

    msg=""

    ss=""

    uname=""

    if request.method=='GET':

        act = request.args.get('act')

    #if 'username' in session:

        #  uname = session['username']

    ff2=open("un.txt","r")

    uname=ff2.read()

    ff2.close()
```

```
cursor = mydb.cursor()

cursor.execute('SELECT * FROM register WHERE card = %s', (uname, ))

account = cursor.fetchone()

name=account[1]

mobile=account[3]

print(mobile)

email=account[4]

vid=account[0]

shutil.copy('faces/f1.jpg', 'faces/s1.jpg')

cutoff=15

img="v"+str(vid)+".jpg"

cursor.execute('SELECT * FROM vt_face WHERE vid = %s', (vid, ))

dt = cursor.fetchall()

for rr in dt:

    hash0 = imagehash.average_hash(Image.open("static/frame/"+rr[2]))

    hash1 = imagehash.average_hash(Image.open("faces/s1.jpg"))

    cc1=hash0 - hash1

    print("cc="+str(cc1))

    if cc1<=cutoff:
```

```

        ss="ok"

        break

    else:

        ss="no"

    if ss=="ok":

        act="2"

        msg="Face Verified"

        print("correct person")

        return redirect(url_for('login', msg=msg))

    else:

        act="1"

        msg="Face not Verified"

        print("wrong person")

        mess="Someone Access your account"

        url2="http://localhost/atmpin/img.txt"

        ur = urlopen(url2)#open url

        data1 = ur.read().decode('utf-8')

        idd=int(data1)

        url="http://iotcloud.co.in/testsms/sms.php?sms=linkatmpin&name="+name+"&mess="
        "+mess+"&mobile="+str(mobile)+"&id="+str(idd)

```

```

print(url)

webbrowser.open_new(url)

return render_template('verify_face2.html',msg=msg,act=act)

def DCNN_process(self):

    test = train_data_preprocess.flow_from_directory(

        'dataset/test',

        target_size = (128,128),

        batch_size = 32,

        class_mode = 'binary')

    ## Initialize the Convolutional Neural Net

    # Initialising the CNN

    cnn = Sequential()

    # Step 1 - Convolution

    # Step 2 - Pooling

    cnn.add(Conv2D(32, (3, 3), input_shape = (128, 128, 3), activation = 'relu'))

    cnn.add(MaxPooling2D(pool_size = (2, 2)))

    # Adding a second convolutional layer

    cnn.add(Conv2D(32, (3, 3), activation = 'relu'))

```

```
cnn.add(MaxPooling2D(pool_size = (2, 2)))

# Step 3 - Flattening

cnn.add(Flatten())

# Step 4 - Full connection

cnn.add(Dense(units = 128, activation = 'relu'))

cnn.add(Dense(units = 1, activation = 'sigmoid'))

# Compiling the CNN

cnn.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics =
['accuracy'])

history = cnn.fit_generator(train,

                            steps_per_epoch = 250,

                            epochs = 25,

                            validation_data = test,

                            validation_steps = 2000)

plt.plot(history.history['acc'])

plt.plot(history.history['val_acc'])

plt.title('Model Accuracy')

plt.ylabel('accuracy')

plt.xlabel('epoch')
```

```
plt.legend(['train', 'test'], loc='upper left')

plt.show()

plt.plot(history.history['loss'])

plt.plot(history.history['val_loss'])

plt.title('Model Loss')

plt.ylabel('loss')

plt.xlabel('epoch')

plt.legend(['train', 'test'], loc='upper left')

plt.show()

test_image = image.load_img('\\dataset\\', target_size=(128,128))

test_image = image.img_to_array(test_image)

test_image = np.expand_dims(test_image, axis=0)

result = cnn.predict(test_image)

print(result)

if result[0][0] == 1:

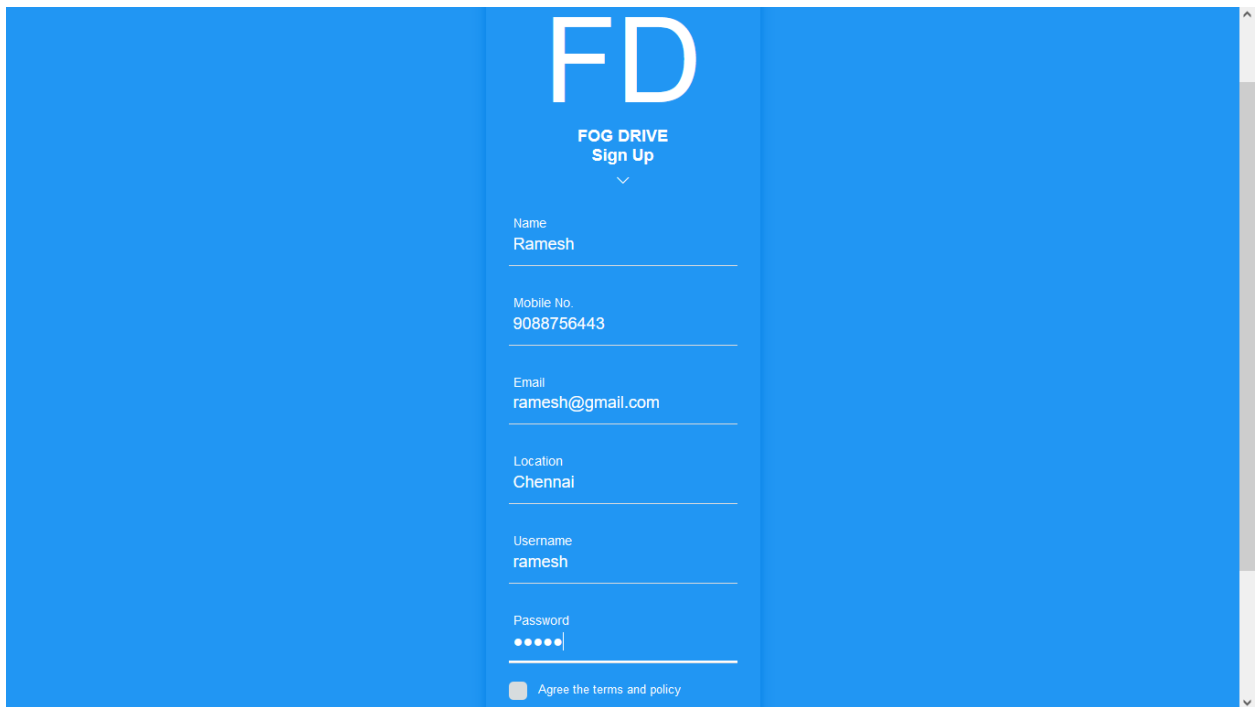
    print('feature extracted and classified')

else:

    print('none')
```



## 9.2 SCREENSHOTS:



The screenshot shows the 'FD FOG DRIVE Sign Up' page. It features a blue background with a white login form in the center. The form includes fields for Name, Mobile No., Email, Location, Username, and Password. Below the password field is a checkbox for 'Agree the terms and policy'.

FD  
FOG DRIVE  
Sign Up

Name  
Ramesh

Mobile No.  
9088756443

Email  
ramesh@gmail.com

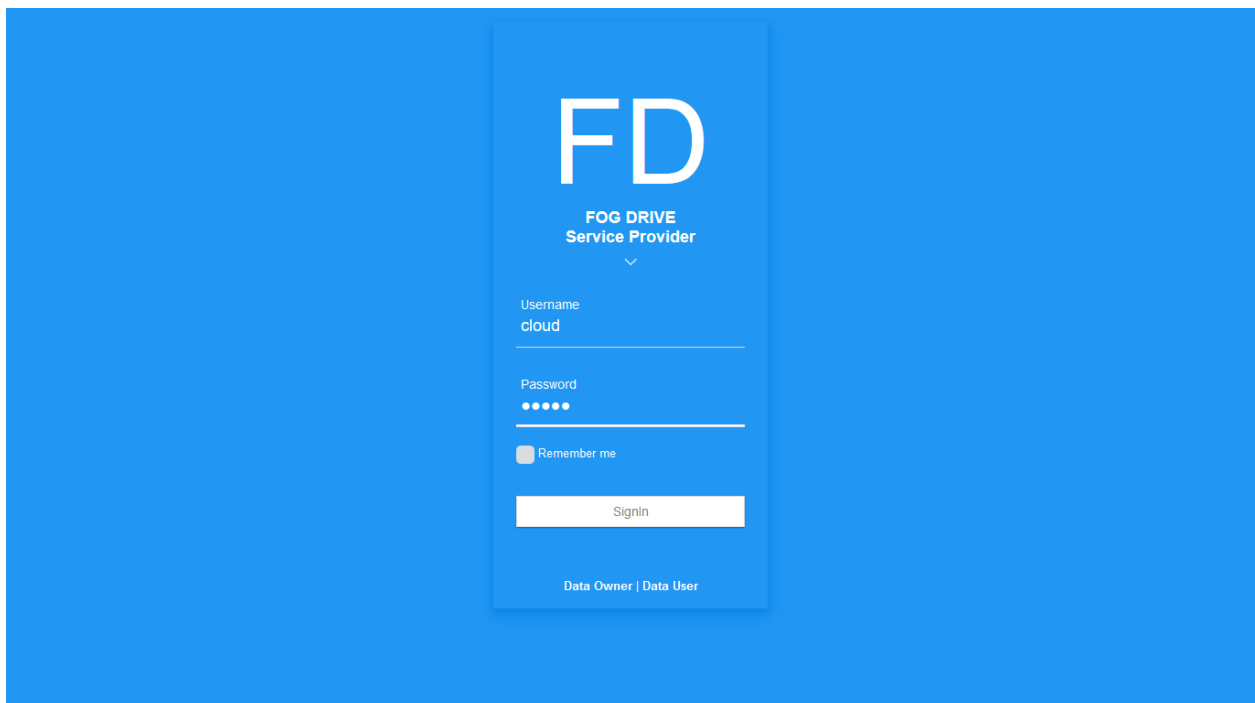
Location  
Chennai

Username  
ramesh

Password  
•••••

☐ Agree the terms and policy

Figure 9.2.1 FD User Login Page



The screenshot shows the 'FD FOG DRIVE Service Provider' login page. It features a blue background with a white login form in the center. The form includes fields for Username and Password, a 'Remember me' checkbox, and a 'Signin' button. At the bottom, there is a link for 'Data Owner | Data User'.

FD  
FOG DRIVE  
Service Provider

Username  
cloud

Password  
•••••

☐ Remember me

Signin

Data Owner | Data User

Figure 9.2.2 Service Provider Login Page

The screenshot shows a web application interface for a Cloud Service Provider (CSP). The top navigation bar is blue with a close icon, the text 'CSP', and a user profile icon. The main header area is light gray and contains the text 'Cloud Service Provider' and a link to 'Server Status'. A sidebar on the left shows a 'Dashboard' link. The main content area is titled 'Data Owner Approval' and contains a table with 7 columns: S.No, Name, Mobile No., E-mail, Location, Date, and Status. The table lists 6 entries. The first three entries are approved. The fourth and fifth entries have a 'Click to Approve' link in the Status column. The sixth entry is approved.

S.No	Name	Mobile No.	E-mail	Location	Date	Status
1	tharun	9345349071	suryalogesh174@gmail.com	trichy	16-04-2025	Approved
2	madhan	9345349071	suryalogesh174@gmail.com	trichy	16-04-2025	Approved
3	arun	9345349071	suryalogesh174@gmail.com	trichy	16-04-2025	Approved
4	suriya	9435436784	dttharunkrish003@gmail.com	Coimbatore	21-04-2025	<a href="#">Click to Approve</a>
5	arunn	8976543456	arun@gmail.com	tirupur	30-04-2025	<a href="#">Click to Approve</a>
6	loki	9456457843	loki@gmail.com	tirupur	30-04-2025	Approved

Figure 9.2.3 Dashboard Page (Data Owner Approval)

The screenshot shows a login screen for 'FOG DRIVE Data Owner'. The background is a solid blue color. In the center, there is a white login form. The form has a large 'FD' logo at the top, followed by the text 'FOG DRIVE Data Owner'. Below this, there are input fields for 'Username' (with the value 'ramesh') and 'Password' (with masked characters). There is a 'Remember me' checkbox and a 'Signin' button. At the bottom of the form, there are links for 'Signup | Data User | Service Provider'.

Figure 9.2.4 Data Owner Login Screens

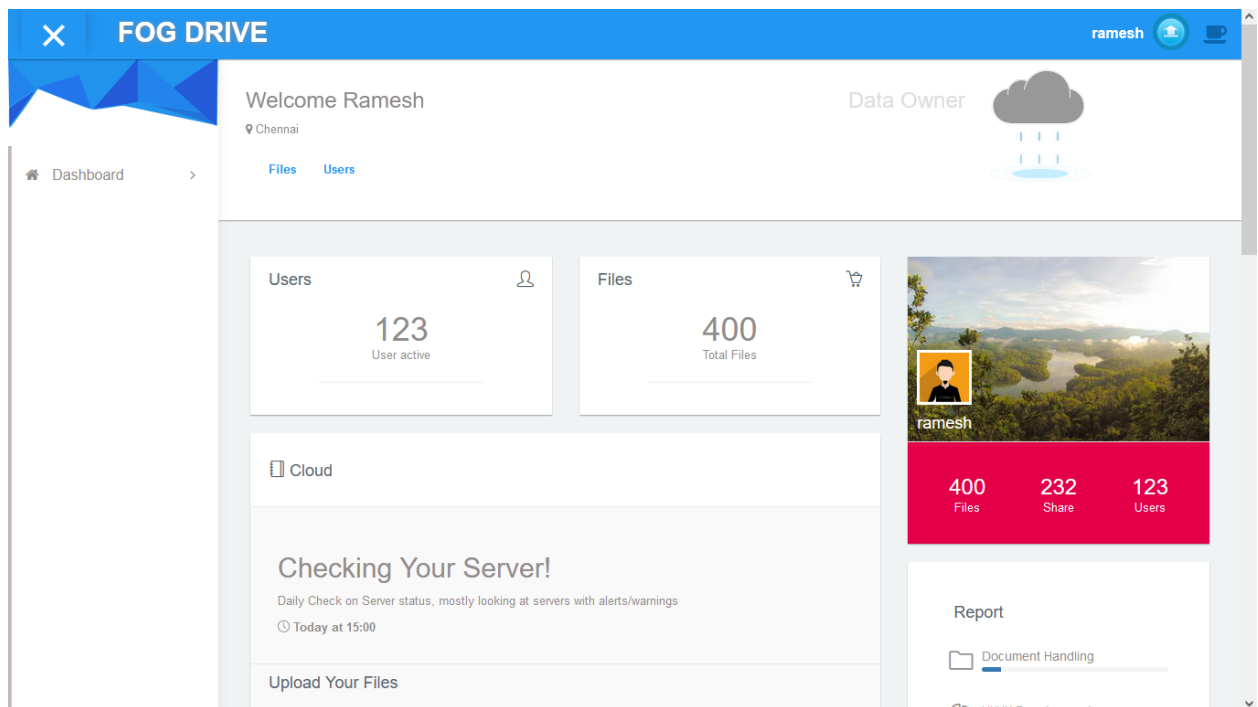


Figure 9.2.5 FogDrive Dashboard - Server Status

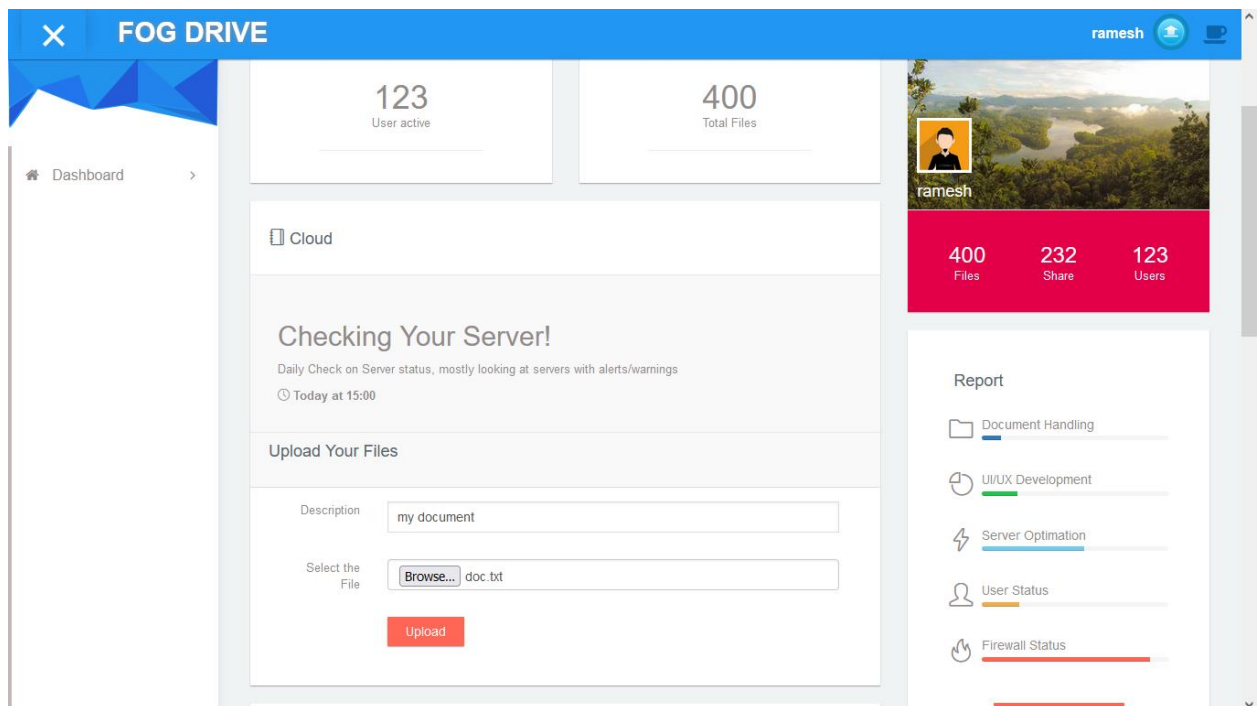


Figure 9.2.6 FogDrive - Upload New Files

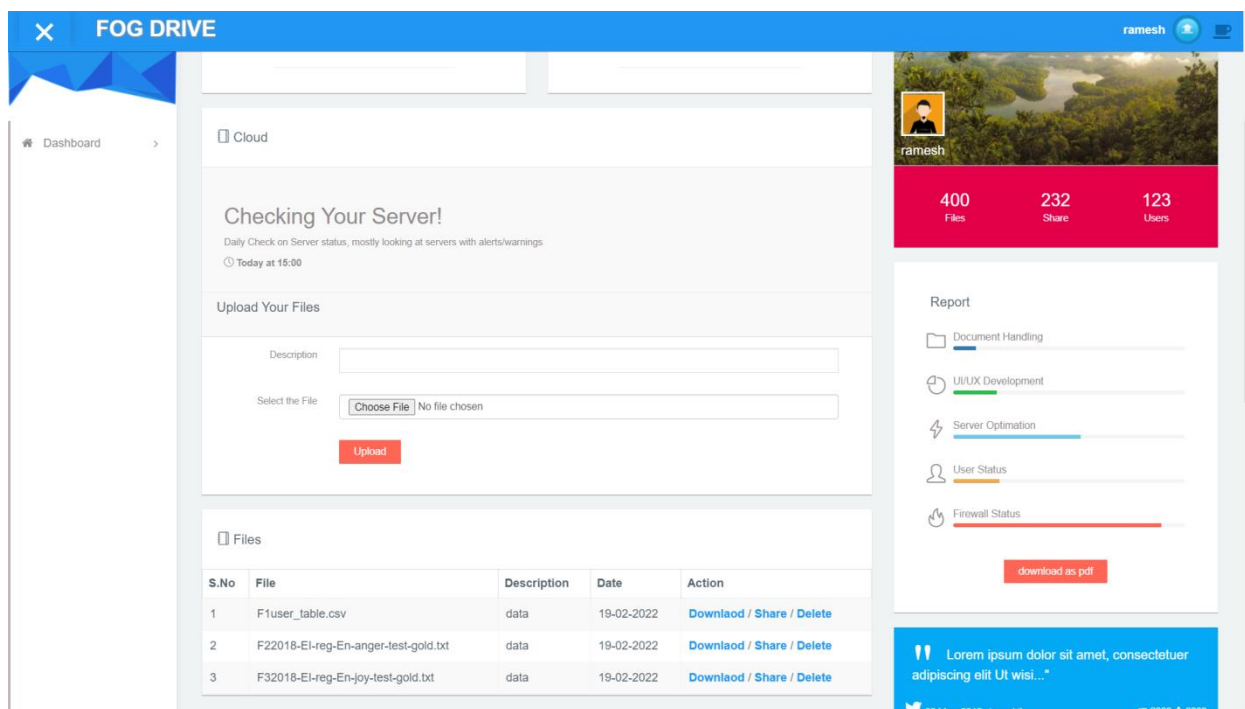


Figure 9.2.7 FogDrive Dashboard - User Details

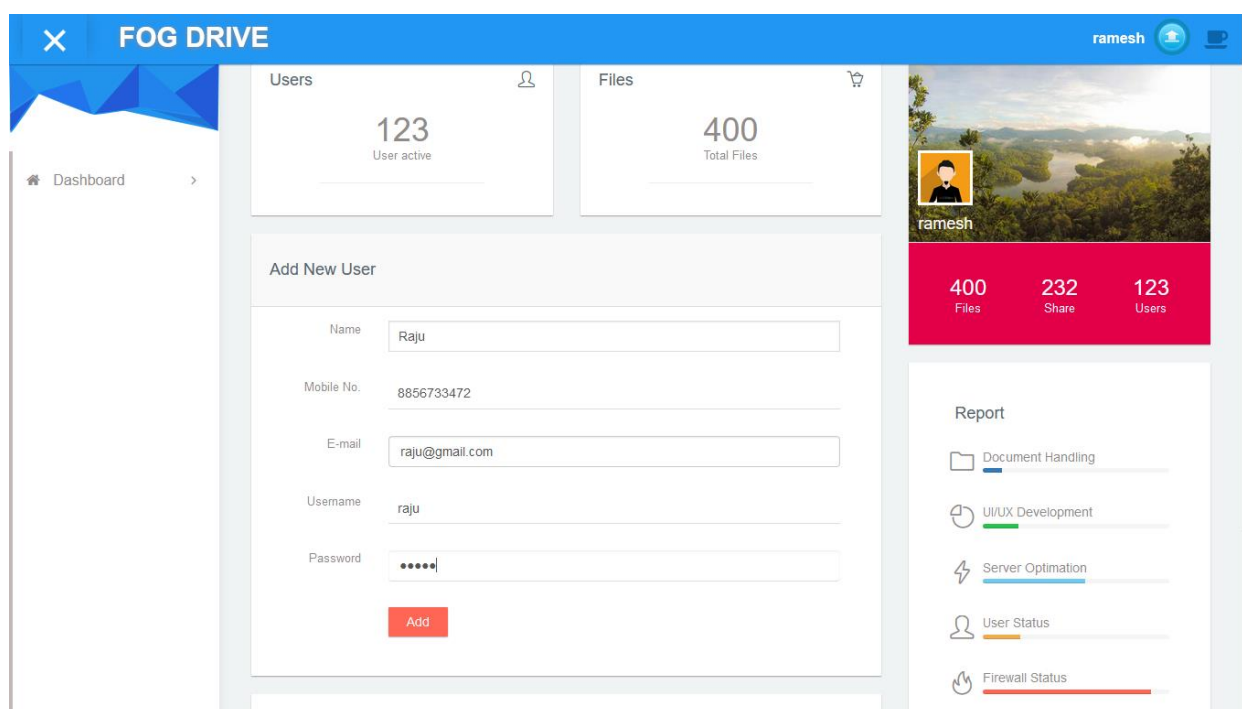


Figure 9.2.8 FogDrive - Add New User

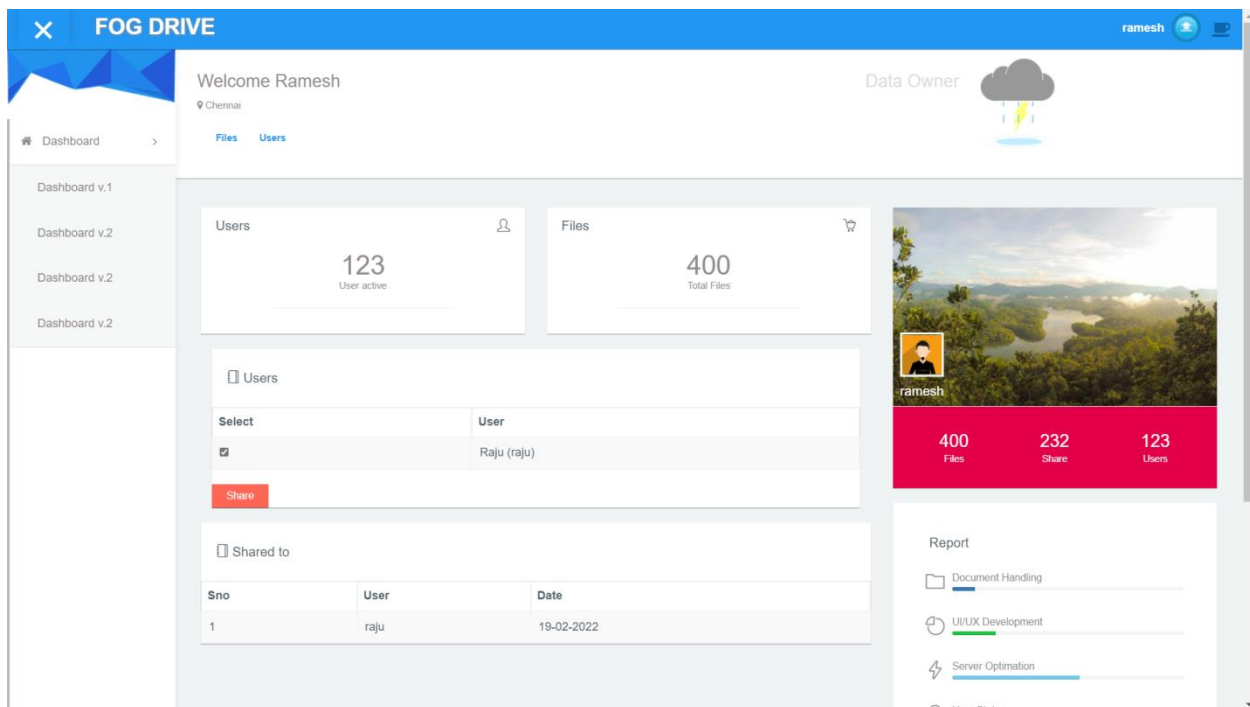


Figure 9.2.9 FogDrive Home Dashboard

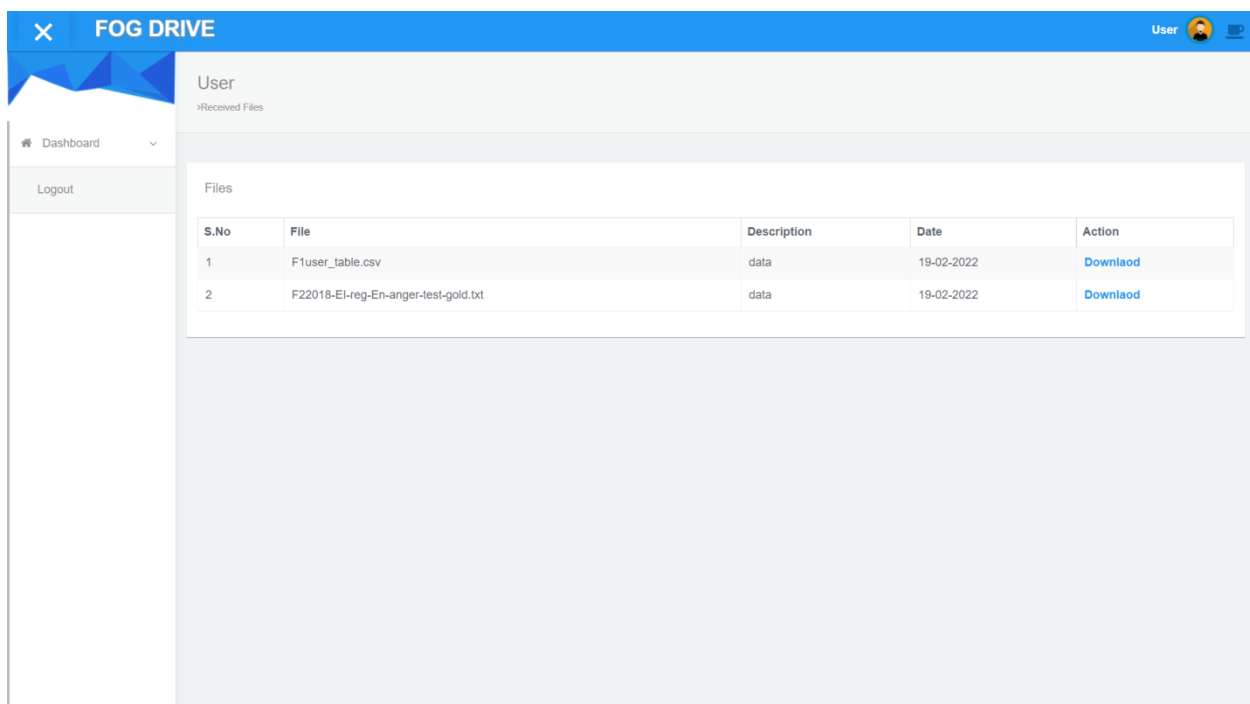


Figure 9.2.9 Files

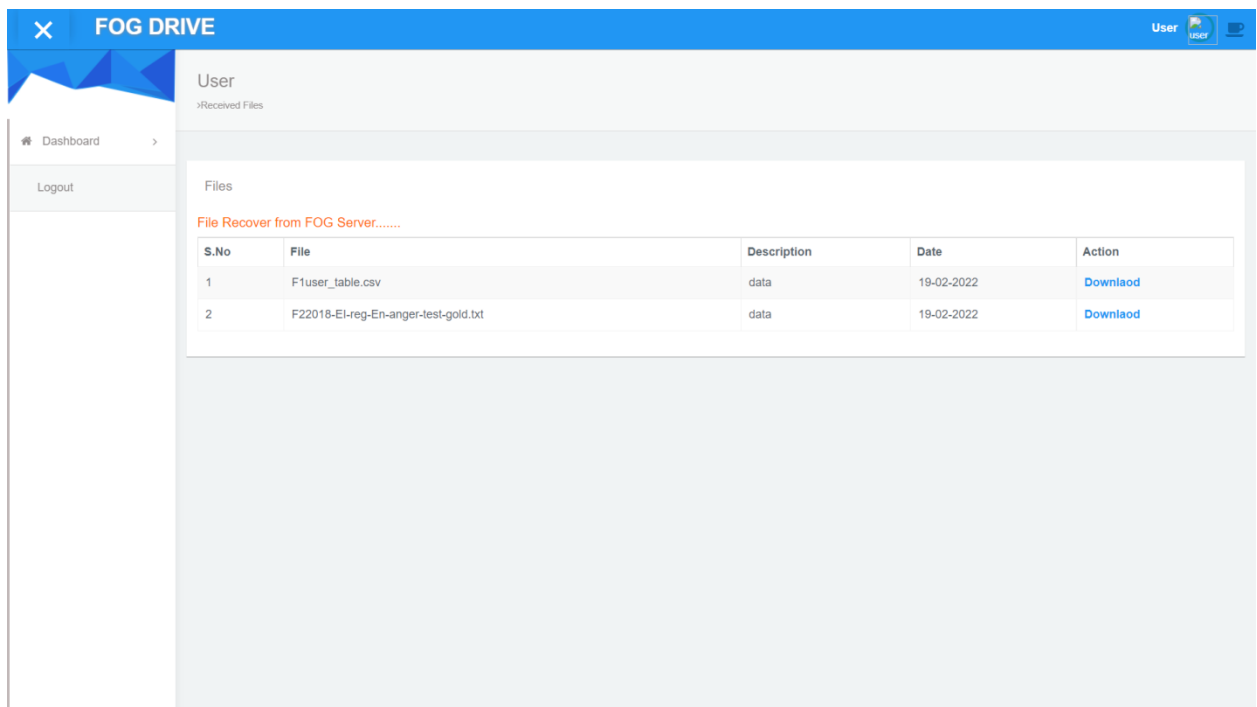


Figure 9.2.11 File Recover From Fog Server

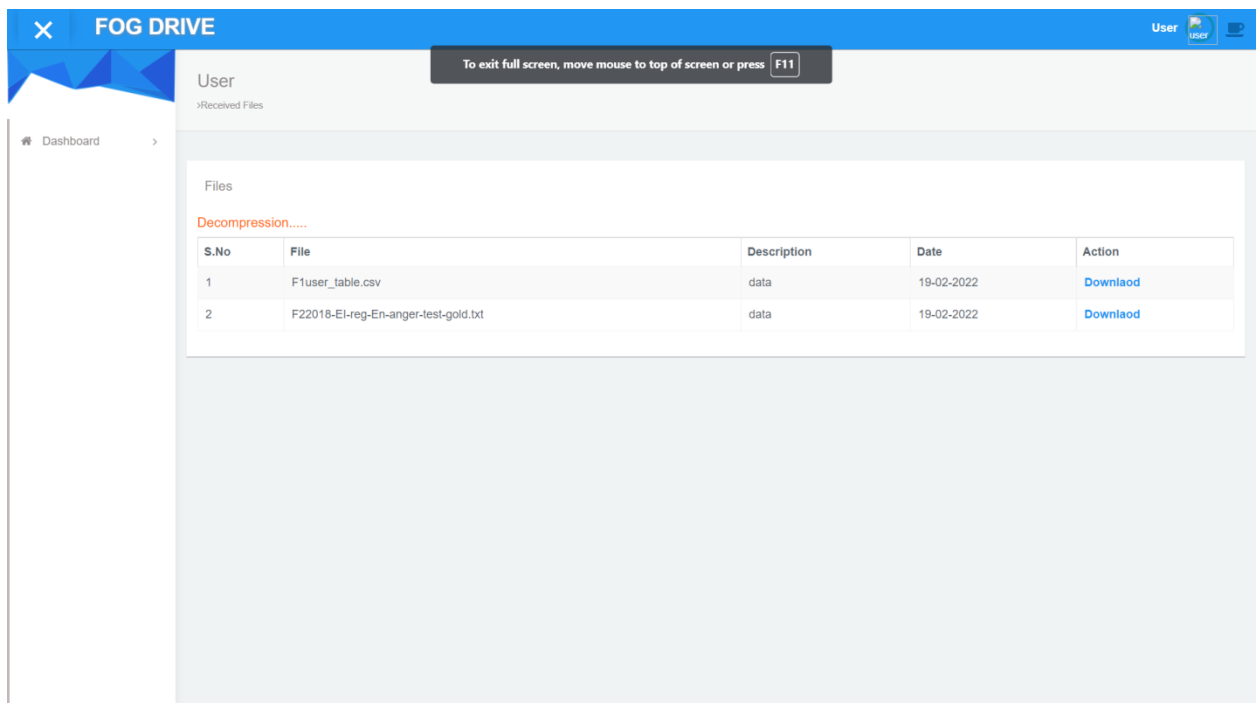


Figure 9.2.12 File Decompression

Data									
S.No	File	Status	File Size	Compressed Size	Date/Time				
0	F79abstract.pdf	Transferred	493172.0	367623.0	2025-04-30 11:56:37				
1	F11data.txt	Stored	9036.0	5682.0	2022-05-27 22:25:23				
2	F11data.txt	Stored	9036.0	5682.0	2022-05-27 22:25:00				

Figure 9.2.13 Compressed File

## **CHAPTER 10**

### **10.1 BOOK REFERENCES**

1. Shivang Modi, Yash Dakwala and Vishwa Panchal, "Cloud Backup & Recovery Techniques of Cloud Computing and a Comparison between AWS and Azure Cloud", International Research Journal of Engineering and Technology (IRJET), vol. 07, no. 07, July 2020.
  2. AbedallahAbualkishik, Ali Alwan and Yonis Gulzar, "Disaster Recovery in Cloud Computing Systems: An Overview", International Journal of Advanced Computer Science and Applications, vol. 11, pp. 702, 2020.
  3. S. Hamadah and D. Aqel, "A proposed virtual private cloud-based disaster recovery strategy", 2019 IEEE Jordan Int. Jt. Conf. Electr. Eng. Inf. Technol. JEEIT 2019 - Proc., pp. 469-73, 2019.
  4. Bhalerao and A. Pawar, "Utilizing Cloud Storage for Big Data Backups", pp. 933-938, 2018.
  5. M. S. Fernando, "IT disaster recovery system to ensure the business continuity of an organization", 2017 Natl. Inf. Technol. Conf. NITC 2017, vol. 2017-Septe, pp. 46-8, 2018
  6. Mohammad Alshammari and Ali Alwan, "A Conceptual Framework for Disaster Recovery and Business Continuity of Database Services in Multi- Cloud", 2017.
  7. Raigonda rani Megha and Tahseen Fatima, "A Cloud Based Automatic Recovery and Backup System with Video compression", International journal of engineering and computer science., vol. 5, pp. 17819-17822, 2016.
  8. Dr. Shriram K. Vasudevan, Abhishek S. Nagarajan, Karthick Nanmaran, Data Structures using Python, Oxford University Press, 2021.
  9. Pramod Chandra P. Bhatt, Naresh Kumar Sehgal, Project Management in Cloud Applications, Springer, 2024.
  10. Pankaj Jalote, Software Engineering: With Open Source (2nd Edition), Wiley India, 2023.
- G+



## 10.2 WEB REFERENCE

1. Amazon Web Services – Backup and Disaster Recovery  
<https://aws.amazon.com/disaster-recovery/>
2. Microsoft Azure – Cloud Backup Solutions  
<https://azure.microsoft.com/en-us/solutions/backup/>
3. OpenPGP Alliance – Encryption Standards  
<https://www.openpgp.org/about/>
4. What is Fog Computing? – Cisco  
<https://www.cisco.com/c/en/us/solutions/internet-of-things/fog-computing.html>
5. VirtualBox – Oracle Open Source Virtualization  
<https://www.virtualbox.org/>
6. MySQL Official Documentation  
<https://dev.mysql.com/doc/>
7. Flask Web Framework – Official Documentation  
<https://flask.palletsprojects.com/>
8. PyCryptodome – Python Cryptography Library  
<https://www.pycryptodome.org/>
9. WampServer – Apache + MySQL + PHP  
<http://www.wampserver.com/en/>
10. DBaaS Explained – IBM Cloud  
<https://www.ibm.com/cloud/learn/database-as-a-service>