Leave Analysis Platform

TABLE OF CONTENT

| [Overview: 1](#_heading=h.gjdgxs)  [Objectives: 1](#_heading=h.30j0zll)  [Scope 1](#_heading=h.1fob9te)  [C4 Model 2](#_heading=h.3znysh7)  [Context Diagram 2](#_heading=h.2et92p0)  [Container Diagram 2](#_heading=h.tyjcwt)  [Component Diagram 3](#_heading=h.3dy6vkm)  [Code Diagram 4](#_heading=h.1t3h5sf)  [System Architecture 5](#_heading=h.4d34og8)  [Key Components and Their Interactions: 5](#_heading=h.2s8eyo1)  [Data Ingestion 5](#_heading=h.17dp8vu)  [Parallel Processing 6](#_heading=h.3rdcrjn)  [Database (MySQL) 6](#_heading=h.26in1rg)  [How do they Work Together? 7](#_heading=h.lnxbz9)  [Backup And Data Persistence 7](#_heading=h.35nkun2)  [Database Schema 8](#_heading=h.1ksv4uv)  [Tables: 8](#_heading=h.44sinio)  [Deployment on Amazon EC2 10](#_heading=h.2jxsxqh)  [Patterns 12](#_heading=h.z337ya)  [Design Patterns 12](#_heading=h.3j2qqm3)  [Processing Patterns 12](#_heading=h.1y810tw)  [Error Handling Patterns 12](#_heading=h.4i7ojhp)  [Future Test Cases 13](#_heading=h.2xcytpi)  [1 Unit Tests 13](#_heading=h.1ci93xb)  [2 Integration Tests 13](#_heading=h.3whwml4)  [3 Performance Tests 13](#_heading=h.2bn6wsx)  [4 Scalability Tests 13](#_heading=h.qsh70q)  [5 Security Tests 14](#_heading=h.3as4poj)  [6 Disaster Recovery Tests 14](#_heading=h.1pxezwc)  [Enhancements: 15](#_heading=h.49x2ik5)  [REFERENCES: 15](#_heading=h.2p2csry) |
| --- |

# Overview:

The **Leave Analysis Platform** is designed to pull leave and attendance data from external APIs, load it into a raw database table, process it into structured tables (employee, leaves, leave transactions,designation), and then provide insightful analytics. This platform is built to be scalable, robust, and secure, utilizing modern technologies like Apache Airflow for orchestration, MySQL for database management, and Kubernetes for scaling.

# Objectives:

* To automate the extraction, transformation, and loading (ETL) of leave and attendance data.
* To provide real-time analytics on leave transactions, employee attendance, and other HR-related metrics.
* To integrate seamlessly with existing HR systems and provide visibility through monitoring and reporting.
* To ensure scalability, fault tolerance, and high availability for the platform.
* To maintain a system with audit capabilities, tracking leave transactions over time.

# Scope

The scope of the **Leave Analysis Platform** is in the planning stages and includes:

1. **Data Sources**: APIs from existing HR and attendance systems.
2. **Data Processing**: ETL pipeline planned to be built using Apache Airflow.
3. **Data Storage**: MySQL database for raw and structured data storage.
4. **Data Analysis**: Ability to query and analyze leave transactions in the future.
5. **Deployment**: A deployment strategy is being planned using cloud services (AWS) with containerization (Docker, Kubernetes).
6. **Monitoring and Alerting**: Real-time monitoring with Prometheus and Grafana, to be implemented in the future.

# Functional Requirement

## 1. **Data Upload and API Consumption**

**Description**:

* Administrators can either upload bulk data manually (via CSV or similar formats) or consume data directly from the API provided by Vyaguta.
* The data includes employee profiles and leave records.

**Process**:

* The API fetches data using a GET request to the Vyaguta endpoint:

| https://dev.vyaguta.lftechnology.com.np/api/leave/leaves?fetchType=all&startDate=2021-07-17&endDate=2024-04-23&size=10000&roleType=issuer |
| --- |

* Authentication is handled via Bearer tokens, ensuring security and proper authorization. A valid Bearer token must be included in the request header.
* The data retrieved from the API is stored in a **raw table** (leave\_raw), temporarily holding the unprocessed leave records.
* The status of the data ingestion is updated in a **status table** to track the completion of the process.

## 2. **Employee Profile Management**

**Description:**

* The system extracts employee profiles from the raw data and inserts them into a user table.

**Process**:

* Once the raw data is ingested, the **ETL process** extracts employee-specific information from the raw table.
* This information includes details such as employee ID, name, role, department, and other relevant fields.
* The transformed data is loaded into the user table in MySQL.
* Profiles can be displayed through a user interface or API that retrieves the information from the user table.

## **3. Comprehensive Leave Data Visualization**

**Description**:

* + The system provides visual representations of leave data to help analyze leave trends, balances, and distributions.

**Process**:

* + After processing the data, the system stores leave-related details in the leave and leave\_txmn tables.
  + A **visualization API** consumes this processed data and provides four different types of data analysis:
    1. **Leave Trends**: Visualization of how leave requests have changed over time.
    2. **Leave Balances**: Display current leave balances for individual employees or teams.
    3. **Leave Distribution**: A breakdown of leave usage by department, role, or other categorizations.
    4. **Leave Types**: Analysis of different types of leave (sick leave, vacation, etc.) and their usage.
  + The visualization API can be integrated with front-end visualization tools like **Grafana** or other BI platforms.

## 4. **Real-time Data Updates**

**Description**:

* + The system is capable of processing and updating data in real-time, reflecting the latest leave records and employee information.

**Process**:

* + After every data ingestion or upload, the **status table** is updated to reflect the completion.
  + This triggers the **ETL process**, ensuring that the data from the leave\_raw table is transformed and loaded into the respective user, leave, designation and leave\_txn tables.
  + Airflow schedules the data ingestion and ETL processes at regular intervals or based on the completion of specific events.
  + Real-time updates ensure that leave balances, trends, and employee records reflect the most recent data.

## 5. **Customizable Visualization Options**

**Description**:

* + The system allows customizable options for visualizing leave data, enabling users to tailor the presentation based on their needs.

**Process**:

* + Users can customize the visualizations by selecting specific date ranges, departments, roles, or leave types.
  + The system supports filtering by various parameters (e.g., date range, leave type, department, or employee role).
  + The **visualization API** provides different endpoints for accessing these customized views.
  + Advanced users can select specific chart types (bar graphs, pie charts, etc.) and metrics for in-depth analysis.

## 6. **Reporting and Insights**

**Description**:

* + The system provides options for generating reports and gaining insights into leave trends, employee behavior, and overall leave distribution.

**Process**:

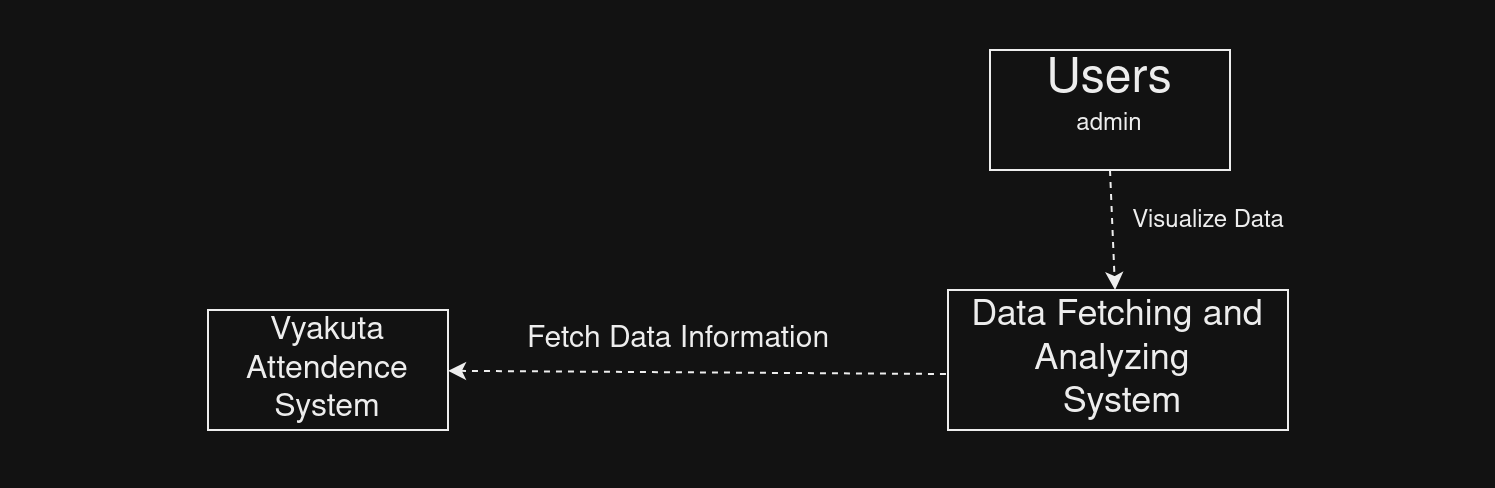
* + **Predefined Reports**:
    - The system provides standard reports like:
      * Leave balances by department or individual employees.
      * Historical leave usage trends.
  + **Custom Reports**:
    - Users can generate custom reports by selecting parameters like date ranges.
  + **Insights and Trends**:
    - The system performs analysis to identify patterns such as:
      * Periods with high leave requests.
      * Employees with consistently low or high leave balances.
  + **API for Reports**:
    - The **reporting API** allows external systems or administrators to generate reports programmatically, making it easier to integrate with other tools.

### Summary

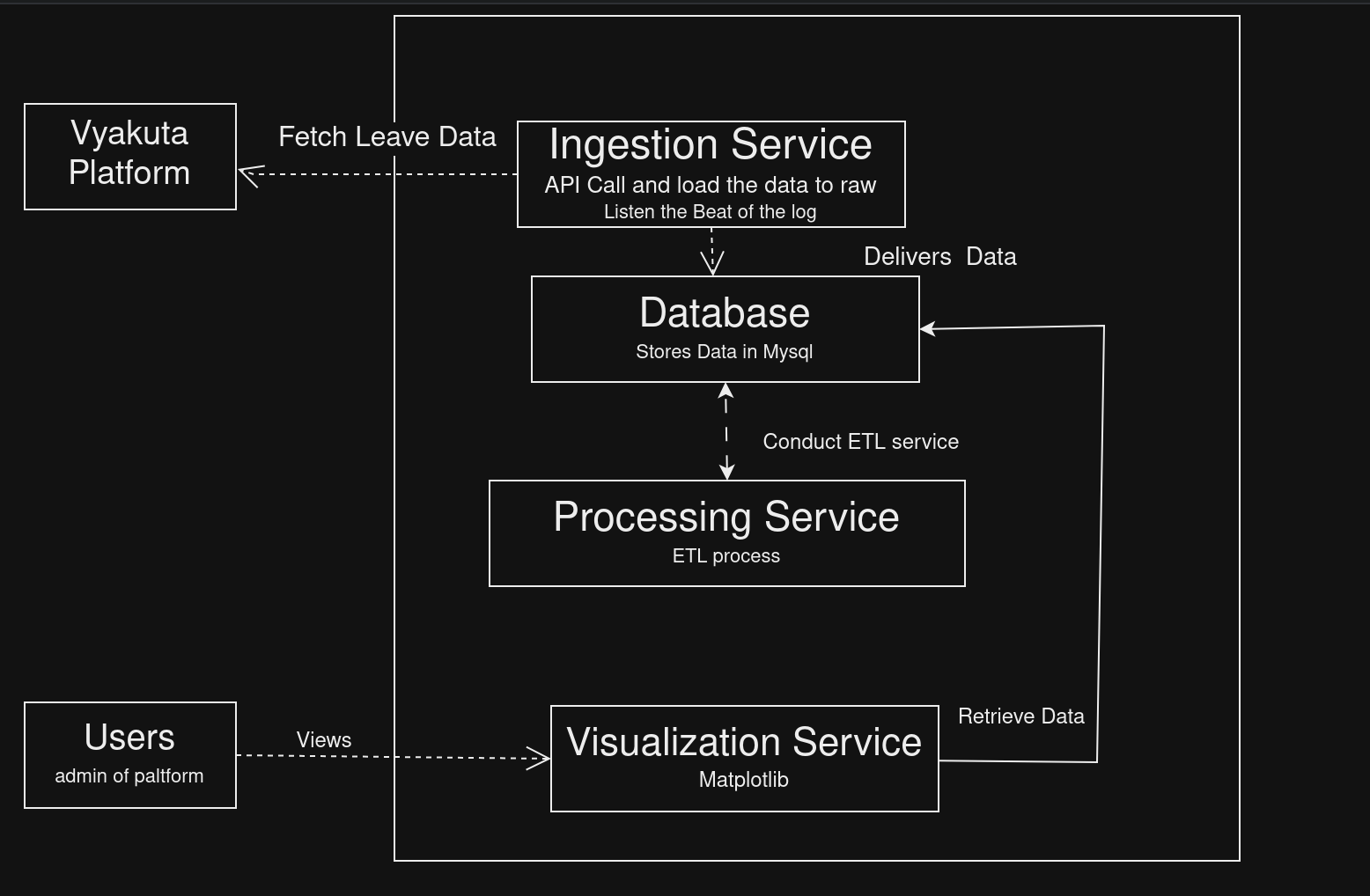
The system facilitates bulk data uploads and real-time data consumption via an API, stores the data in raw format, and processes it using an ETL pipeline. The data is transformed into different tables (user, leave,designation, leave\_txn) and made available for visualizations, custom reports, and insights. Airflow orchestrates the entire process, triggering actions like status updates and ETL jobs to ensure data is always current.

# C4 Model

## Context Diagram

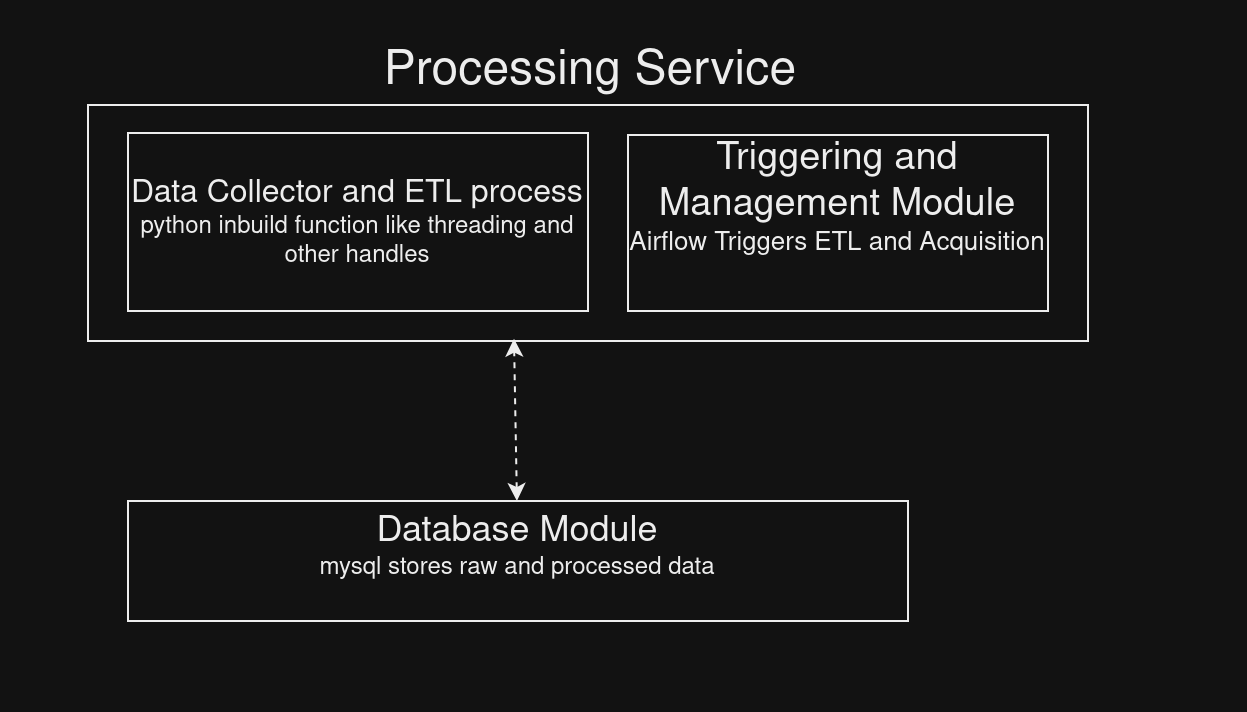
**Fig 1: C4 Context Diagram**

## **Container Diagram**



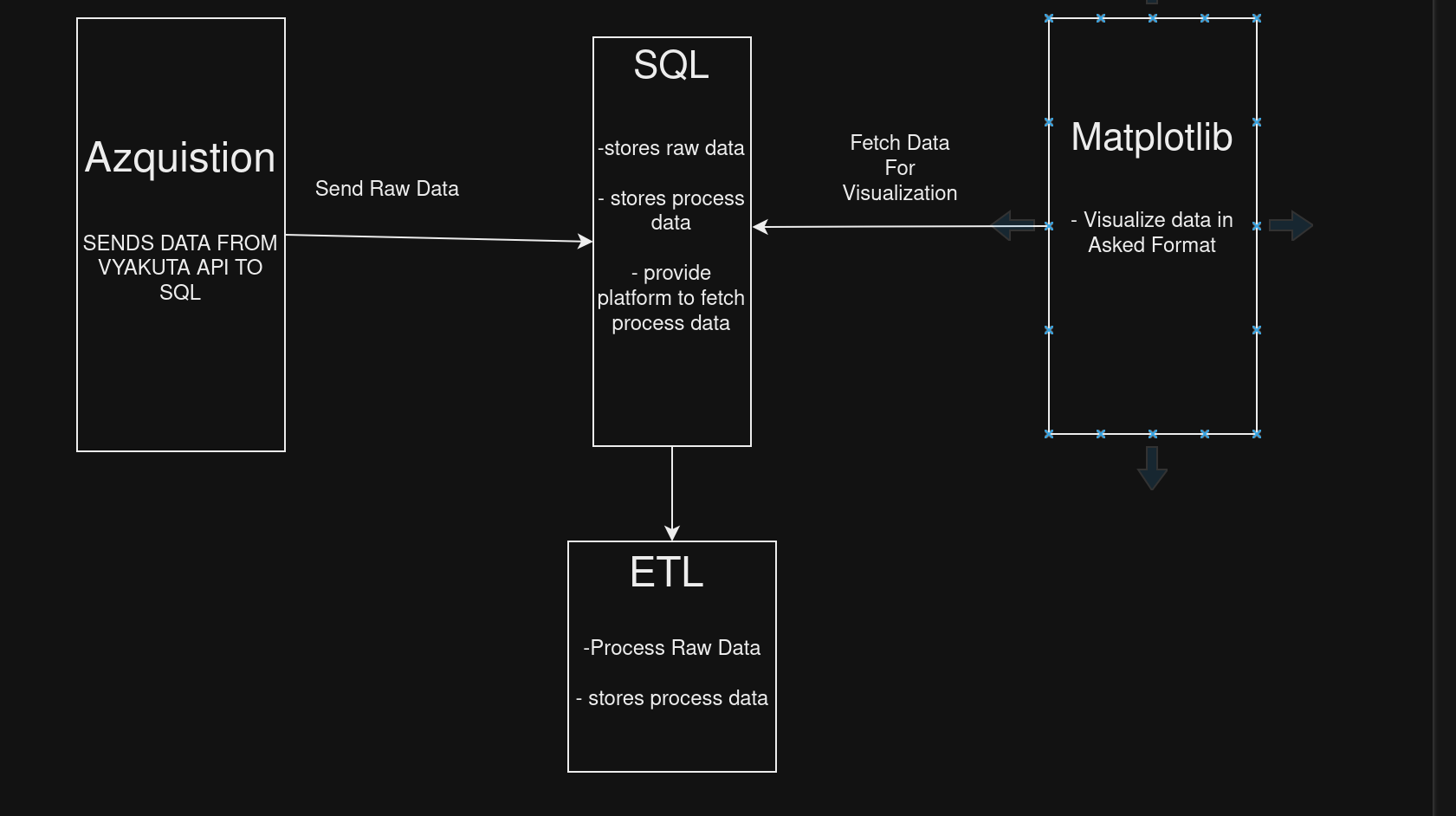
**Fig 2: C4 Container Diagram**

## **Component Diagram**

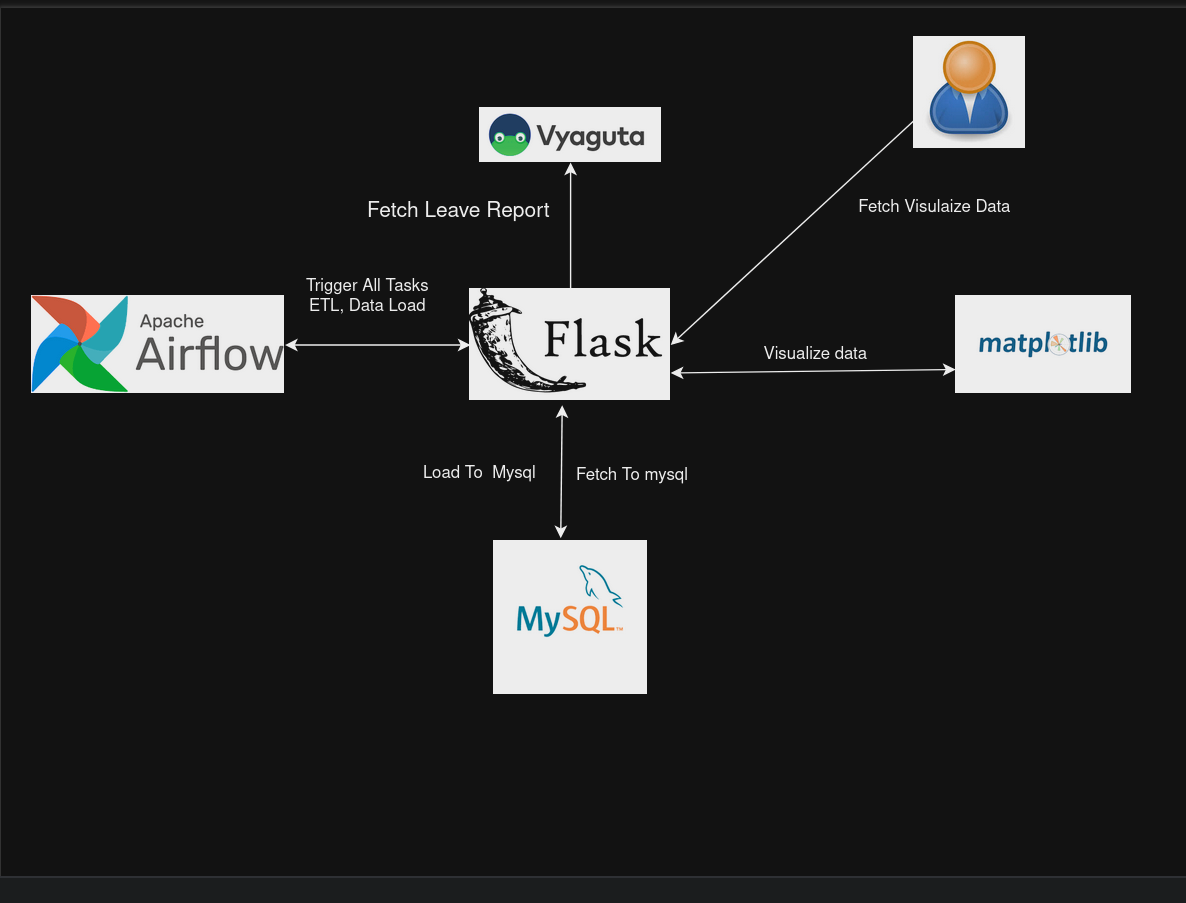
****

**Fig 3: C4 Component Diagram**

## **Code Diagram**

**Fig 4: C4 Code Diagram**

# System Architecture

** Fig 5: System Diagram**

# Key Components and Their Interactions:

### Data Ingestion(Handled By Flask)

* **API Data Source**: The platform begins by pulling data from an external API, which provides raw information about employee, leave requests, designation and leave transactions. The data is ingested in real-time to ensure the platform stays up to date.
* **Parallel Ingestion Using Thread Pools**: The system employs Python’s concurrent.futures.ThreadPoolExecutor for parallel processing during data ingestion. Multiple threads are spawned to concurrently pull data from the API for different entities (such as employee, leave transactions, etc.) and insert the data into the raw table. This significantly speeds up the ingestion process.

### Parallel Processing(Handled By Flask)

Once the raw data is ingested, the system uses threading for concurrent processing of the raw data. A second thread pool handles the transformation of the raw data into structured tables (employee, leaves, designation,leave\_transactions).

* Each thread operates on a portion of the raw data, applying the necessary business logic and transformation rules in parallel.
* The processed data is then inserted into the respective structured tables in the database for further analysis and reporting.

### Database (MySQL)

* **Raw Table**: The platform first stores the unprocessed data in the raw table, which serves as a staging area for data collected from the API. This table contains detailed information on employees, their leave details, and fiscal year data.
* **Processed Tables**: After ingestion and transformation, the data is moved to more structured tables (employee, leaves, designation,leave\_transactions). These tables are optimized for querying and further analysis.
* **Employee**: Contains employee data, including their designation, department, and supervisory roles.
* **Leaves**: Stores information on leave types, default days, and fiscal year details.
* **Leave Transactions**: Tracks employee leave transactions, including start/end dates, reasons, and leave statuses.
* **Designation**: Stores Designation details

### How do they Work Together?

**Data Ingestion with Thread Pools:**

* Multiple threads pull data from the API concurrently, fetching different entities (e.g., employee, leave transactions).
* This data is inserted into the raw table for temporary storage and transformation.

**Parallel Processing with Threading:**

* After ingestion, additional threads handle the transformation of raw data into structured formats.
* Each thread processes a portion of the raw data simultaneously and loads it into the appropriate tables (employee, leaves, designation,leave\_transactions).

**Data Persistence and Storage:**

* The structured tables (employee, leaves,designation, leave\_transactions) store processed and cleaned data, which can be queried and analyzed by the system's analytics layer.
* Backup and Data Persistence: In future stages, a backup strategy will be implemented using MySQL dumps or incremental backups to ensure data is not lost during system failures.

## Backup And Data Persistence

**MySQL Backup Strategy**: Currently, data is stored in the MySQL database, with plans to implement a robust backup strategy. This will include taking periodic snapshots or using incremental backups to ensure data integrity.

**Future Cloud Backup**: The platform may integrate cloud storage solutions (e.g., Amazon S3, Google Cloud Storage) for off-site backups to support disaster recovery and data redundancy.

# Database Schema

## Tables:

**Raw Data Table**

| id: integer  user\_id: integer  emp\_id: VARCHAR(255)  team\_manager\_id: integer  designation\_id: integer  designation\_name: VARCHAR(255)  first\_name: VARCHAR(255)  middle\_name: VARCHAR(255)  last\_name: VARCHAR(255)  email: VARCHAR(255)  is\_hr: VARCHAR(255)  is\_supervisor: VARCHAR(255)  leave\_issuer\_id: integer  issuer\_first\_name: VARCHAR(255)  issuer\_middle\_name: VARCHAR(255)  issuer\_last\_name: VARCHAR(255)  current\_leave\_issuer\_id: integer  current\_leave\_issuer\_email: VARCHAR(255)  department\_description: VARCHAR(255)  start\_date: DATE  end\_date: DATE  leave\_days: integer  reason: TEXT  leave\_status: VARCHAR(255)  status: VARCHAR(255)  response\_remarks: TEXT  leave\_type\_id: integer  leave\_type: VARCHAR(255)  default\_days: integer  transferable\_days: integer  is\_consecutive: VARCHAR(255)  fiscal\_id: integer  fiscal\_start\_date: DATE  fiscal\_end\_date: DATE  fiscal\_is\_current: VARCHAR(255)  created\_at: TIMESTAMP  updated\_at: TIMESTAMP  is\_automated: VARCHAR(255)  is\_converted: VARCHAR(255)  total\_count: integer  inserted\_at: DATE  allocations: TEXT |
| --- |

**Processed Data Tables**

| **employee**  schema:  id: integer  emp\_id: VARCHAR(50)  first\_name: VARCHAR(100)  middle\_name: VARCHAR(100)  last\_name: VARCHAR(100)  email: VARCHAR(100)  designation\_id: integer  designation\_name: VARCHAR(100)  department\_description: VARCHAR(255)  is\_hr: BOOLEAN  is\_supervisor: BOOLEAN  **leaves**  schema:  id: integer  leave\_type: VARCHAR(100)  default\_days: integer  transferable\_days: integer  fiscal\_id: integer  fiscal\_start\_date: DATE  fiscal\_end\_date: DATE  fiscal\_is\_current: BOOLEAN  **leave\_transactions**  schema:  id: integer  user\_id: integer  leave\_type\_id: integer  start\_date: DATE  end\_date: DATE  leave\_days: integer  reason: VARCHAR(255)  response\_remarks: VARCHAR(255)  leave\_status: VARCHAR(50)  is\_converted: BOOLEAN  created\_at: TIMESTAMP  updated\_at: TIMESTAMP  current\_leave\_issuer\_id: integer  issuer\_first\_name: VARCHAR(100)  issuer\_middle\_name: VARCHAR(100)  issuer\_last\_name: VARCHAR(100)  current\_leave\_issuer\_email: VARCHAR(100)  is\_consecutive: BOOLEAN  is\_automated: BOOLEAN  department\_description: VARCHAR(100)  designation\_name: VARCHAR(100)  designation\_id: integer  is\_supervisor: BOOLEAN  is\_hr: BOOLEAN  **Status**  schema:  id: integer  status\_type: VARCHAR(255)  start\_date: DATE  end\_date: DATE  started\_at: DATETIME  status: integer  updated\_at: TIMESTAMP  **Designation**  schema  id: integer  designation\_name: VARCHAR(255) |
| --- |

# **Deployment on Amazon EC2**

Amazon EC2 (Elastic Compute Cloud) provides scalable virtual servers in the cloud. Here’s how you can deploy the Leave Analysis Platform on EC2:

a. Set Up EC2 Instances

* Choose an AMI (Amazon Machine Image): Select an AMI that matches your operating system requirements (e.g., Ubuntu, Amazon Linux).
* Select Instance Type: Choose the instance type based on your computing, memory, and storage needs. For the Leave Analysis Platform, you might start with a general-purpose instance like t2.medium and scale up as needed.
* Configure Security Groups: Set up security groups to control inbound and outbound traffic. For example, allow HTTP/HTTPS traffic for the web application and restrict SSH access to specific IP addresses.
* Launch Instances: Launch the EC2 instances and assign an Elastic IP if necessary for static access.

b. Install and Configure Software

Install Required Software: Connect to your EC2 instances via SSH and install the necessary software, such as:

* Web Server: Apache or Nginx
* Database: MySQL Server
* Python: Ensure Python is installed for running your ETL scripts and Flask API.
* Other Dependencies: Install Apache Airflow, and any other necessary libraries or frameworks.

Deploy Application Code: Transfer your application code (Flask API, Django Admin Interface, React Employee Portal) to the EC2 instance. You can use tools like scp, rsync, or deploy via a CI/CD pipeline.

Configure Services:

* Web Application: Configure your web server to serve the Flask API, Django Admin Interface, and React Employee Portal.
* MySQL Database: Set up your MySQL database, including creating necessary databases and tables as outlined in your schema.
* Airflow: Configure Apache Airflow to run your ETL pipelines, set up connections, and schedule your tasks.
* Start Services: Start your web server, database, and Airflow services. Ensure they are running as expected and can communicate with each other.

c. Testing and Monitoring

* Test the Deployment: Verify that your application is accessible and functioning correctly. Test data ingestion, processing, and reporting features.
* Set Up Monitoring: Use monitoring tools like Amazon CloudWatch to monitor your EC2 instance performance, including CPU usage, memory usage, and network traffic.

d. Integration in Kubernetes

* **Containerize Your Application**: Create Dockerfiles for each service, build Docker images, and push them to a container registry (e.g., Amazon ECR).
* **Set Up Kubernetes Cluster**: Choose a Kubernetes setup (e.g., Amazon EKS or self-managed EC2) and configure cluster networking for pod communication and external access.
* **Deploy Applications**: Define Kubernetes resources (Deployments, Services, ConfigMaps) in YAML files and apply configurations using kubectl.
* **Manage and Scale**: Use Kubernetes to manage and scale services within the cluster.
* **Integrate with Other Services**: Run MySQL as a Kubernetes service or use a managed option, set up monitoring with Prometheus and Grafana, centralize logging with Fluentd or Logstash, and integrate CI/CD pipelines for automated deployments and testing.

# Workflow Patterns

**Pipeline Pattern (Data Transformation)**: The system uses a **pipeline pattern** during data transformation, where raw data flows from ingestion, through a series of transformation steps (filtering, business logic application), and finally into the structured tables.

**Ingestion and Processing Pattern (Thread Pool → Transformation)**: The platform follows an ingestion and processing pattern where Python’s ThreadPoolExecutor pulls data from external APIs in parallel (producer), and threads transform the data before loading it into MySQL (consumer).

## Processing Patterns

* **Batch Processing (Thread Pool for Transformation)**: The system processes data in batches, with each thread handling a chunk of the raw data. This increases efficiency when processing large volumes of leave data, ensuring that all records are processed in parallel.
* **Stream Processing (API Ingestion)**: Real-time data from the API is streamed into the system using multiple threads. Each thread ingests a specific entity (e.g., employee, leave transactions) for low-latency, real-time updates.

## Error Handling Patterns

* **Retry Pattern (Database Insertion)**: When threads attempt to insert transformed data into the MySQL database, failures such as connection timeouts trigger an automatic retry mechanism. This retry is implemented with exponential back off to ensure system resilience during transient failures.
* **Dead Letter Queue (Failed Data Processing)**: In the case of data that fails transformation or insertion, the system will use a **dead letter queue** pattern to store failed records for later analysis and processing, ensuring that no data is lost.

# Future Test Cases

## 1 Unit Tests

**Objective**: Verify each individual component works as expected.

**Tools**:

* **Pytest** for Airflow tasks and database testing.
* **Postman** for API.

**Test Cases:**

* Test API data retrieval.
* Test the transformation logic for structured tables.

## 2 Integration Tests

**Objective**: Ensure seamless interactions between components.

**Tools**: Custom Python scripts and database verification.

**Test Cases**:

* Verify data flows correctly from API to MySQL.
* Check if raw data is correctly ingested into the raw table and transformed into structured tables (employee, leaves, designation,leave\_transactions).

## 3 Performance Tests

**Objective**: Test how the system handles load during data ingestion and processing.

**Tools**: JMeter for API load testing.

**Test Cases**:

* Simulate high API load for leave transactions and test the system's data ingestion speed.
* Test MySQL performance under high read/write loads during data transformation and querying.

## 4 Scalability Tests

**Objective**: Test how well the system scales under increasing data volume and traffic.

**Test Cases**:

* **Thread Pool Scaling**: Ensure that the thread pool dynamically handles an increasing number of API requests without performance degradation.
* **Database Scalability**: Verify that MySQL efficiently handles large volumes of leave transactions and user data.

## 5 Security Tests

**Objective**: Ensure the platform is secure from external threats and data tampering.

**Test Cases**:

* **User Role-Based Access Control (RBAC)**: Validate that only authorized users can access leave data based on roles (e.g., admin, manager, employee).
* **Data Integrity Checks**: Ensure that there are no unauthorized modifications to user or leave data during processing.
* **API Authentication**: Verify that API calls require proper authentication before processing leave requests or user data.

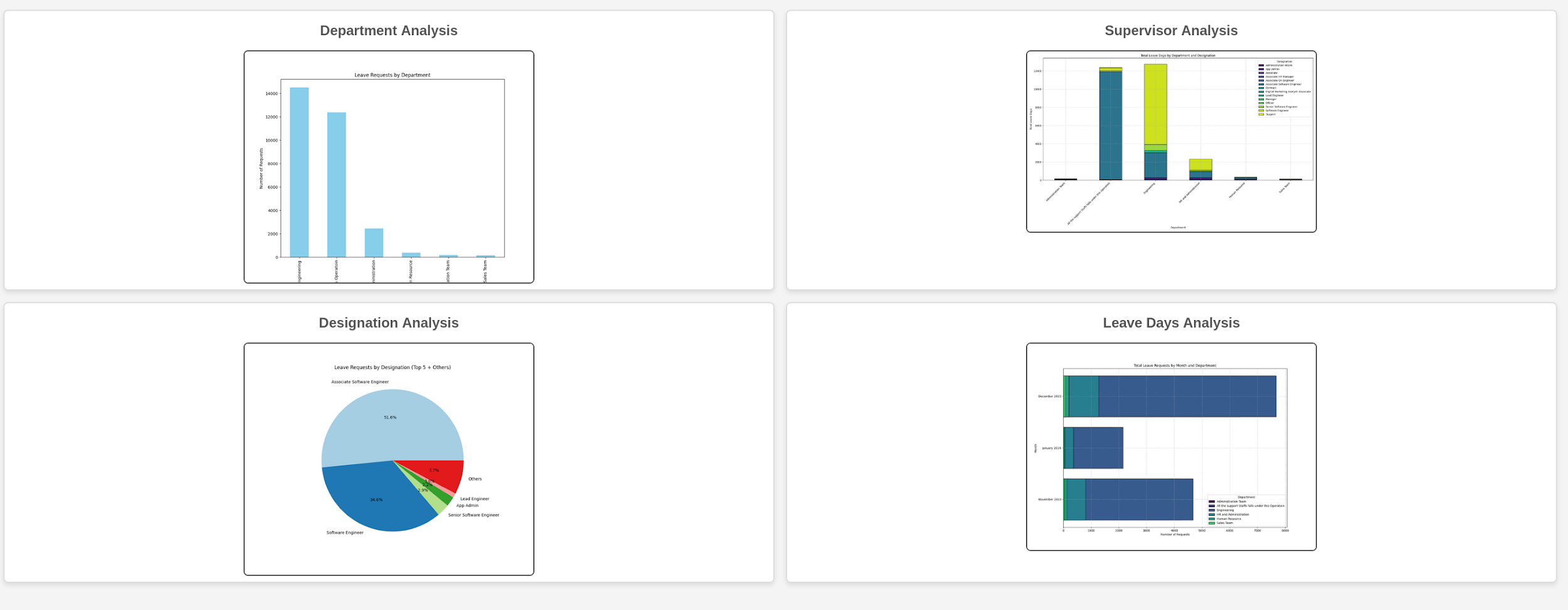
## 6 Disaster Recovery Tests

**Objective**: Validate the platform's ability to recover from failures.

**Test Cases**:

* **Thread Failover**: Simulate thread failures during data ingestion and ensure other threads continue processing data.
* **MySQL Backup and Recovery**: Test the platform's ability to restore data from backups in case of database failure.
* **API Recovery**: Simulate API downtime and test how the system handles failed API requests, ensuring retries and consistent data ingestion.

# Sample:



# Enhancements:

Several future enhancements are planned for the platform:

* **Support for Additional APIs**: Integrate additional HR and attendance systems.
* **Machine Learning**: Apply machine learning models for anomaly detection in leave patterns.
* **Data Export**: Allow exporting reports in CSV or other formats.

Link:https://github.com/lfbibek2024/leave-visualize

# REFERENCES:

* [Airflow Documentation](https://airflow.apache.org/)
* [MySQL Documentation](https://dev.mysql.com/doc/)