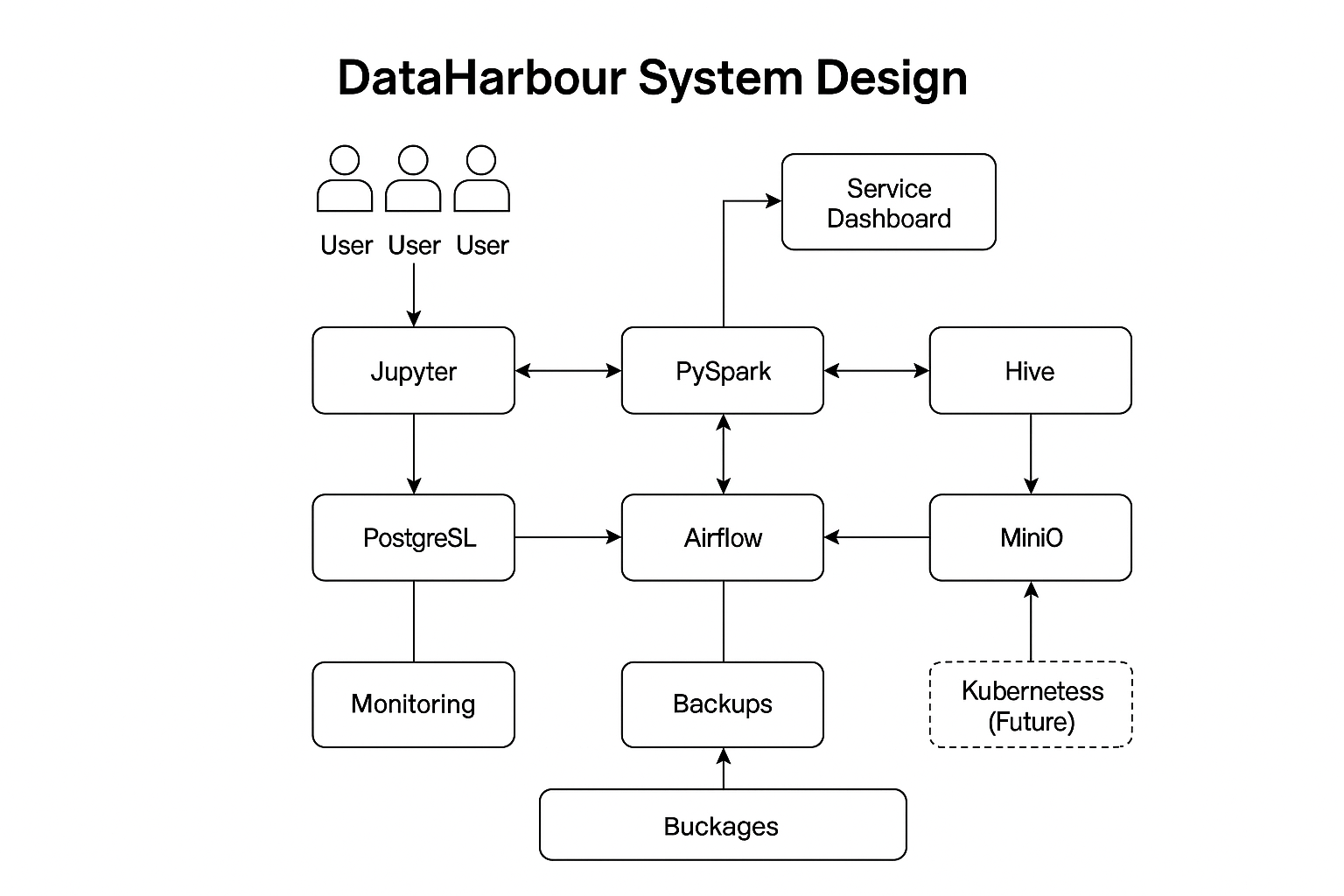
**Project Introduction – DataHarbour**

DataHarbour is a comprehensive, Docker-based data engineering platform designed for students, educators, and small businesses to process, analyze, and manage data efficiently. It integrates key tools like PySpark, Jupyter, Airflow, MinIO, PostgreSQL, and Hive into a unified system for data workflows. The platform supports real-time orchestration, interactive development, and S3-compatible storage, all accessible through a user-friendly dashboard. With built-in monitoring, backup, and scalability features, DataHarbour enables on-premises deployment for secure and flexible data infrastructure.



**🔹 Functional Requirements**

These describe what the system should do.

1. User Authentication & Authorization
   * Admins and users (students, engineers) should be able to log in securely.
   * Role-based access control (RBAC) for services like Airflow, Jupyter, MinIO.
2. Service Deployment via Docker
   * Launch and manage containers for Spark, Airflow, MinIO, PostgreSQL, Jupyter, and Dashboard.
3. Data Processing and Analysis
   * Enable users to run Spark jobs through Jupyter notebooks.
   * Use PySpark to process datasets stored in MinIO.
   * Store and retrieve Delta Lake tables.
4. Job Scheduling
   * Admins can define DAGs in Apache Airflow for ETL or ML workflows.
   * Schedule and monitor job execution and logs.
5. Storage Management
   * Upload/download data via MinIO (S3-compatible).
   * Persistent volume support for Jupyter notebooks, Spark outputs, and PostgreSQL metadata.
6. Service Dashboard
   * Visual monitoring for CPU/memory usage, service status, and logs.
   * Ability to start/stop services from the web interface.
7. Monitoring & Logging
   * Centralized logging with Loki + Filebeat.
   * Metrics collection via Prometheus + Grafana dashboards.
8. Backup and Restore
   * Periodic backup of PostgreSQL and MinIO using cron jobs.
   * Restore support for disaster recovery.

**🔸 Non-Functional Requirements**

These define how the system should behave.

1. Scalability
   * Support scaling services for larger workloads or concurrent users (Kubernetes-ready in future).
2. Performance
   * Efficient execution of Spark jobs and low-latency service access.
   * Dashboard and logs must refresh in near real-time.
3. Availability
   * 99.9% uptime for all critical services like Jupyter, Airflow, MinIO, PostgreSQL.
   * Self-healing via Docker restart policies.
4. Security
   * HTTPS and BasicAuth with NGINX reverse proxy.
   * Secure storage of credentials.
   * Limited container access (Principle of Least Privilege).
5. Portability
   * Entire environment must be reproducible via docker-compose up.
6. Usability
   * Easy onboarding for students and small teams with minimal technical setup.
   * UI/UX consistency across services.
7. Maintainability
   * Modular architecture to add/remove services easily.
   * Central config file for port bindings and credentials.
8. Data Integrity
   * ACID support via Delta Lake and PostgreSQL.
   * Consistent data across pipeline executions

***📊* Capacity Estimation – DataHarbour**

|  |  |  |  |
| --- | --- | --- | --- |
| Component | Estimation Basis | Expected Load | Notes |
| Users | Concurrent users (students, engineers, admins) | 10–50 active users | Can scale to 100+ with Kubernetes. |
| Notebooks | Avg. notebook size + data | ~50MB per user | Stored in mounted volumes; persistent across sessions. |
| MinIO Storage | Uploaded datasets, processed files, logs | ~500GB – 1TB for small teams | Easily extendable via MinIO's scaling. |
| Airflow DAGs | ETL/ML job orchestration | 20–50 DAGs | Avg. 1–2 DAGs per user for recurring pipelines. |
| Spark Cluster | PySpark job processing | 2–3 worker containers with 2–4 cores, 8GB RAM each | Suitable for medium workloads; can scale horizontally. |
| PostgreSQL | Metadata and Airflow backend | ~2–5GB | Requires backup & indexing for performance. |
| Jupyter | Notebook runtime per user | 512MB–1GB RAM per session | Controlled via Docker resource limits. |
| Logging | Logs via Filebeat → Loki | 1–2GB per week | Rotated weekly; old logs offloaded to MinIO. |
| Monitoring | Prometheus metrics, Grafana dashboards | ~1GB/month | Lightweight metric collection. |

📌 Key Considerations

* Storage: Use SSDs with minimum 1TB capacity for high I/O performance.
* Memory: At least 16–32GB RAM on host for smooth multi-service operations.
* CPU: 8+ vCPUs recommended for concurrent usage and Spark workloads.
* Network: Internal bandwidth of 1Gbps+ for fast inter-container communication.

**🧩 Low-Level Design (LLD) – DataHarbour**

1. Architecture Overview

All components run in isolated Docker containers and communicate over a Docker network. Services are orchestrated using docker-compose with shared volumes for persistence.

┌────────────┐

│ NGINX │

└───────────┘

│

┌─────────────────────────────────────┐

│ │ │

┌────▼────┐ ┌────▼─────┐ ┌────▼─────┐

│ Jupyter │ │ Airflow │ │ MinIO UI │

└────┬────┘ └────┬─────┘ └────┬─────┘

│ │ │

▼ ▼ ▼

┌────────┐ ┌──────────┐ ┌────────────┐

│ PySpark│ │PostgreSQL│ │ MinIO │

└────────┘ └──────────┘ └────────────┘

▲

│

┌──────┴──────┐

│ pgAdmin │

└─────────────┘

2. Service Breakdown

🔹 Jupyter Notebook

* Runs interactive Python/PySpark notebooks
* Mounts ./data/jupyter for persistence
* Connects to Spark via internal Docker hostname (spark://spark-master:7077)

🔹 PySpark Cluster

* Spark master and worker containers
* Reads/writes data from MinIO using Hadoop S3 connector
* Communicates with Jupyter and Airflow

🔹 Airflow

* Webserver + scheduler + worker + metadata DB (PostgreSQL)
* DAGs mounted from ./data/airflow/dags
* Uses PostgreSQL as backend and Spark for ETL

🔹 PostgreSQL

* Stores Airflow metadata
* Persistent volume ./data/postgres
* Accessed by Airflow, pgAdmin

🔹 MinIO

* S3-compatible object storage
* Used for storing datasets, logs, backup
* Mounts ./data/minio/data
* Interacts with Spark and Jupyter

🔹 NGINX

* Reverse proxy for Jupyter, Airflow, MinIO, and pgAdmin
* Configurable for HTTPS and basic auth

🔹 Monitoring Stack

* Prometheus scrapes metrics from services
* Grafana displays dashboards
* Loki + Filebeat aggregate logs

🔹 Dashboard (Optional)

* Web UI for container health, metrics, logs (accesses Docker socket)

3. Network Design

* All services share a bridge network (dataharbour\_net)
* Internal DNS allows services to communicate using container names
* External ports are mapped for user access:
  + Jupyter: 8888
  + Airflow: 8081
  + MinIO: 9000/9001
  + pgAdmin: 5050

4. Security Design

* NGINX with basic auth in front of services
* MinIO and PostgreSQL use access keys and credentials
* Docker volumes are permission-controlled
* Optionally use HTTPS with self-signed or Let's Encrypt certs

5. Data Flow

1. User uploads data via Jupyter or MinIO.
2. Data is stored in MinIO buckets.
3. Spark jobs read data from MinIO and process it.
4. Results are stored back in MinIO or PostgreSQL.
5. Airflow orchestrates ETL jobs using Spark operators.
6. Monitoring stack captures metrics and logs.