Project Architecture

Written By: Mahyar Esteki

Document Version: 1.0

Release Date: March 24, 2025

Contents

1 Abstract

Proper knowledge of the infrastructure and architecture of any system will lead to a better understanding of its performance. This information helps developers to reach a common view on the use and operation of the system. The purpose of presenting the architectural document of this project will be to create a common understanding of the structure and operation of the system. In this document you will get acquainted with system architecture, how services work, network architecture and more. This document contains general information and brief descriptions of the system architecture. Therefore, additional technical information will be provided in the relevant documents.

2 Project Schema

This diagram represents the architectural schema, focusing on a modular and scalable system designed to support research operations, productivity evaluation, and human resource management.

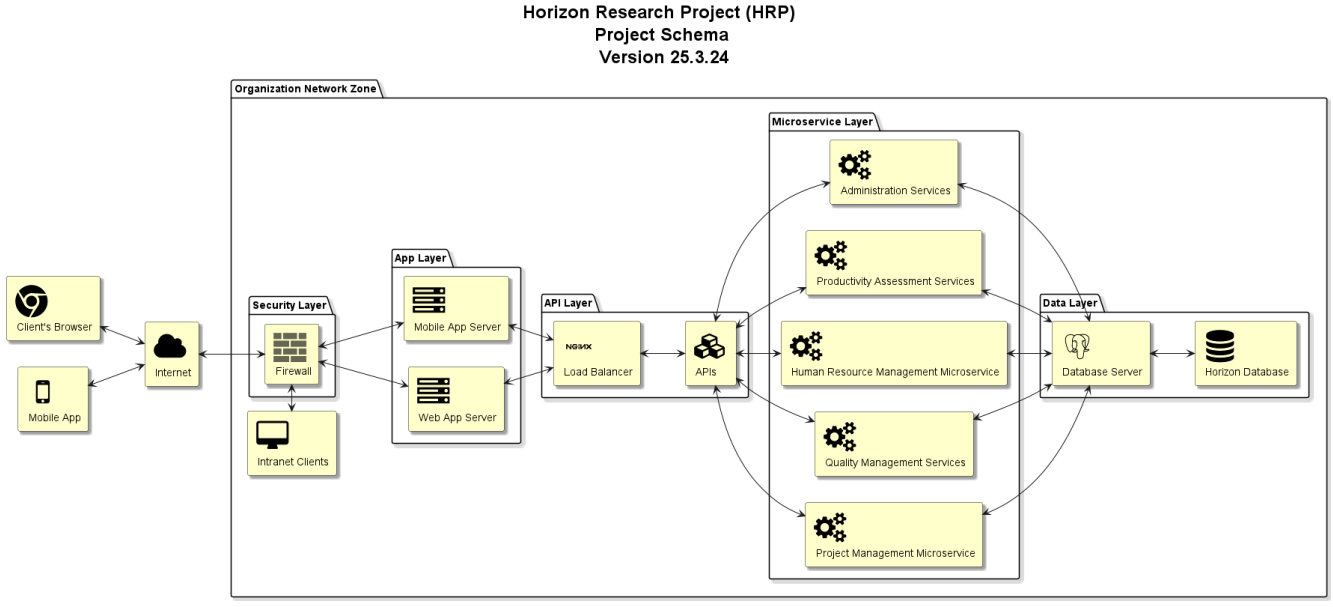

Figure 1: Project Schema Diagram

The system is organized into multiple layers within the "Organization Network Zone":

1. **Microservice Layer**:

   This layer hosts several independent business services, each responsible for a core functionality:

   - *Administration Services* (adminService)
   - *Productivity Assessment Services* (paService)

- *Human Resource Management* (hrService)
- *Quality Management Services* (qcService)
- *Project Management* (projectService)

2. **API Layer**:
    - Contains an *NGINX Load Balancer* (webServer1) for traffic distribution.
    - Exposes internal *APIs* (apis) that route requests to the microservices.
3. **Application Layer**:
    - Hosts both *Mobile App Server* and *Web App Server* to serve client applications.
4. **Security Layer**:
    - Implements a *Firewall* for controlling and securing network traffic.
5. **Data Layer**:
    - Includes a *PostgreSQL Database Server* and the *Horizon Database* where all persistent data is stored.
6. **Client Access**:
    - Internal users interact via *Intranet Clients* (userStation).
    - External access is provided to both the *Mobile App* and the *Client's Browser* through the *Internet*.

Data Flow Overview:

Clients (web and mobile) connect through the internet to the system, passing through the firewall and reaching the appropriate application servers. Requests are processed via the API layer, which communicates with the relevant microservices. All microservices interact directly with the database server for data operations.

3 Components of the System

The **Component Diagram** illustrates the modular structure of the system, showcasing how the key software components interact to deliver the platform's core functionalities.

At the user access level, clients interact with the system via modern web browsers such as **Chrome or Firefox**. These browsers connect securely over **HTTPS** to the **Nginx Web Server**, which serves the **Horizon App** as the frontend interface.

The system's backend is handled by a second **Nginx Web Server** responsible for managing **Horizon APIs**. These APIs encapsulate various core services, including:

- **Administration Services**
- **Human Resource Management Services**
- **Project Management Services**
- **Finance Management Services**
- **Quality Management Services**
- **Productivity Assessment Services**

Each of these services relies on a structured backend environment built using:

- **Python 3.x (SDK)**
- **Flask (Web Framework)**
- **SQLAlchemy (ORM Framework)**
- **Psycopg2 (Data Provider)**

The services utilize the ORM layer (**SQLAlchemy**) to abstract database interactions and leverage **Psycopg2** to connect to the **PostgreSQL** relational database. Data transactions are managed over **TCP** protocol.

The **PostgreSQL** database is designed as the central data repository (**Main Database**), accessible via port **5432**, ensuring reliable data storage and retrieval for all microservices.

This diagram effectively captures the modularity and separation of concerns in HRP's architecture, supporting scalability, maintainability, and secure operations across its various components.

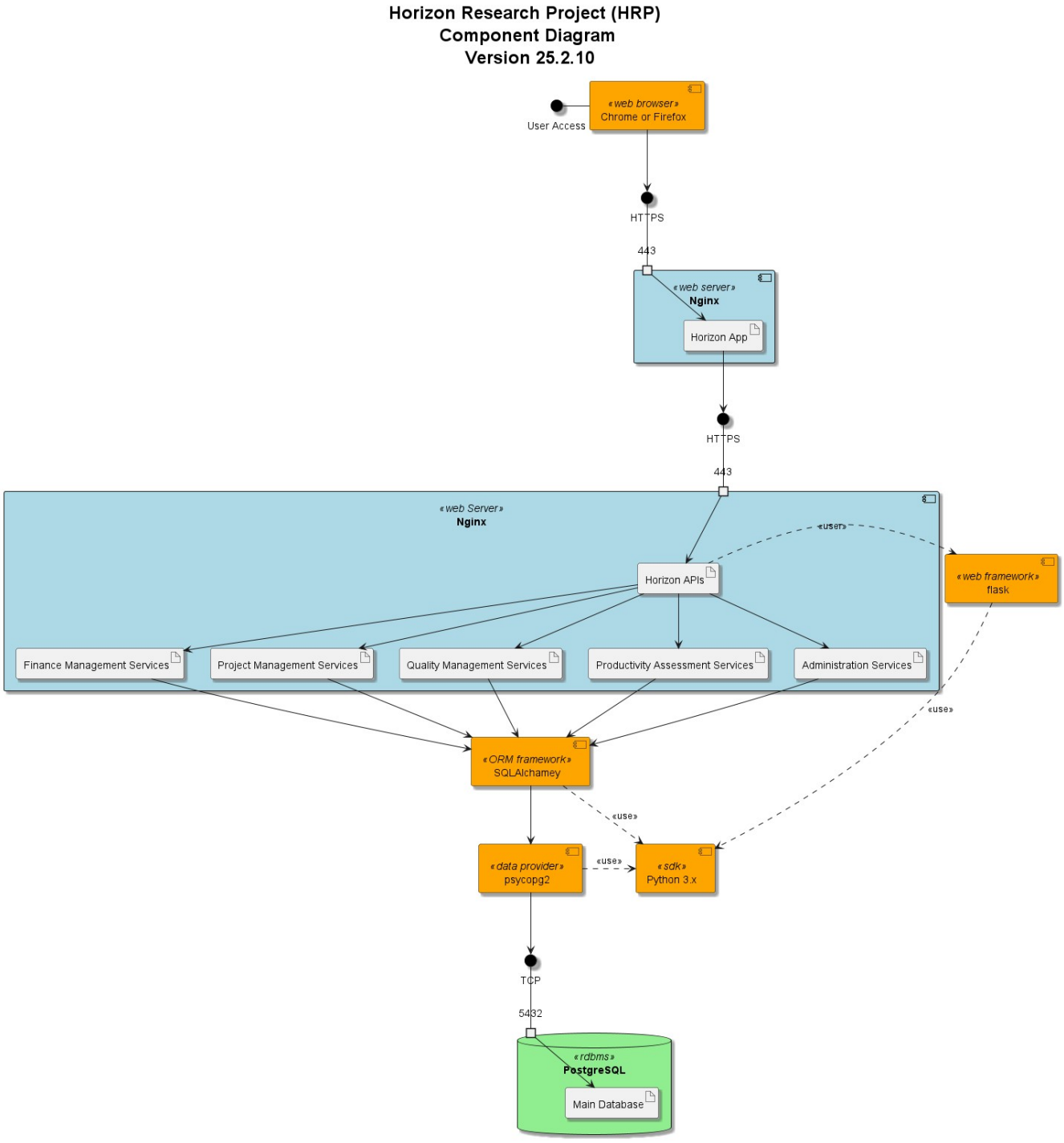

Figure 3: Component Diagram

4 Deployment Model

This deployment diagram illustrates the **network and system architecture** of the *Horizon Research Project (HRP)*, designed for high availability, modularity, and secure data processing.

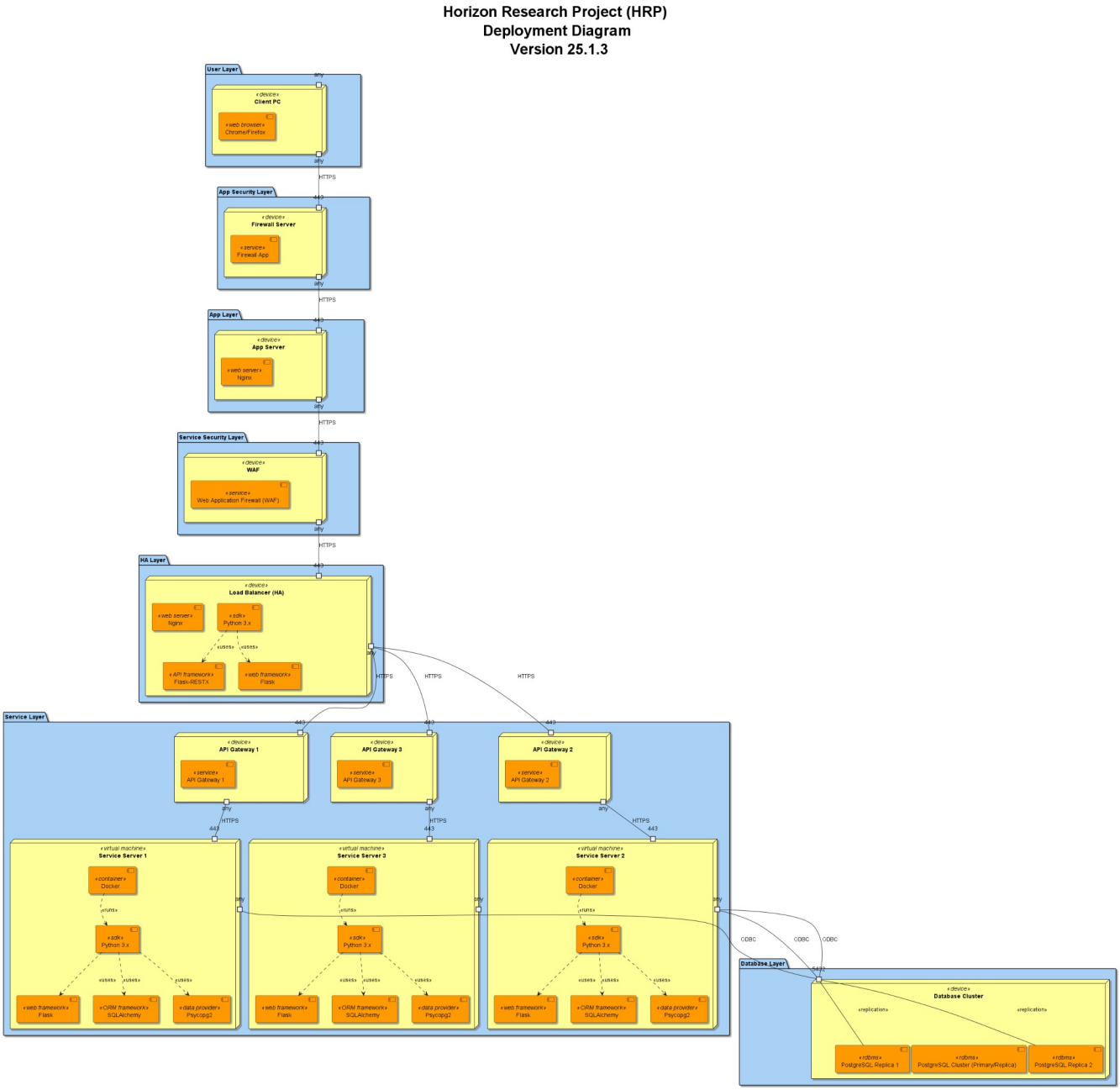

Figure 3: Deoployment Diagram

4.1 Architecture Layers Overview:

1. **User Layer:**
    - **Client PCs** access the system through web browsers (Chrome/Firefox) over HTTPS.
2. **Application Security Layer:**
    - A **Firewall Server** filters incoming traffic before forwarding requests to the application layer.
3. **Application Layer:**
    - The **App Server** runs an **Nginx web server**, serving as the entry point to the web application and forwarding traffic securely to the next layers.
4. **Service Security Layer:**
    - A **Web Application Firewall (WAF)** protects the backend services by inspecting and filtering traffic.
5. **High Availability (HA) Layer:**
    - A **Load Balancer (HA)** running **Nginx** and **Flask** components distributes traffic to multiple API gateways.
    - **Flask-RESTX** is used for building RESTful APIs.
6. **Service Layer:**
    - Contains three independent **API Gateways** and **Service Servers**.
    - Each Service Server runs inside a **Docker container** with:
        - **Python 3.x**
        - **Flask** (web framework)
        - **SQLAlchemy** (ORM)
        - **Psycopg2** (PostgreSQL adapter)
    - Services communicate with the database via **ODBC** over HTTPS.
7. **Database Layer:**
    - A **PostgreSQL Cluster** with a **Primary database** and two **replicas** ensures data persistence, high availability, and replication.
    - Service servers interact directly with the primary database.

4.2 Key Connections and Data Flows:

- Clients send HTTPS requests through the **Firewall ➔ App Server ➔ WAF ➔ HA Layer**.
- The **Load Balancer** distributes traffic to **API Gateways**.
- **API Gateways** route the requests to **Service Servers (Dockerized Microservices)**.
- Each **Service Server** processes the request and communicates with the **PostgreSQL Database Cluster**.
- The database replicates data to two read replicas for failover and load distribution.

4.3 Technology Stack Highlights:

- **Web & App Layer:** Nginx, Flask, Flask-RESTX
- **Service Layer:** Python 3.x, Docker, SQLAlchemy, Psycopg2
- **Database Layer:** PostgreSQL (Primary + Replicas)
- **Security:** Firewall, Web Application Firewall (WAF)
- **Communication Protocols:** HTTPS for services and ODBC for database operations

This architecture ensures **scalability**, **security**, **high availability**, and **modular service deployment**, supporting the research-oriented operations of the Horizon Project.

5 Access Management

The **Access Control Model** is designed to provide a flexible and robust framework for managing user permissions across the system. Access to system resources is granted through two main mechanisms:

1. **Direct Permissions**:

   Users can be assigned specific permissions individually, allowing fine-grained control over what actions each user can perform within the system.

2. **Role and Group-Based Permissions**:

   - **Roles** represent predefined sets of permissions associated with specific responsibilities or job functions. Users assigned to a role automatically inherit all permissions granted to that role.
   - **Groups** allow the system to organize users into logical collections. Permissions can be assigned at the group level, providing an efficient way to manage access rights for multiple users simultaneously.

This dual-layered model ensures that the system can handle both simple and complex organizational structures. It supports scalability, simplifies permission management, and enhances security by ensuring that users only have access to the resources necessary for their role or group membership. The model also allows for dynamic adjustments, enabling administrators to modify roles, groups, or individual permissions as organizational needs evolve.

6 Authentication

The **Authentication Model** is based on **JSON Web Tokens (JWTs)** for secure web-based user verification. When a user logs in, the system validates the user's credentials and generates a JWT. This token contains key **user profile data**, such as the user's name, email, and other identifying information. The token serves as proof of the user's identity and is used for further interactions within the system. The token is sent with each request to authenticate the user, confirming their identity without the need for maintaining server-side session data, making the system more scalable and stateless.

7 Authorization

The **Authorization Model** utilizes the data embedded in the **JWT** to manage user access to system resources and functionality. Once the user is authenticated, the system checks the JWT for **access rights** associated with the user. This includes the permissions and roles that define which parts of the system the user is allowed to interact with. Based on this information, the system determines whether the user has sufficient privileges to perform the requested actions or access specific resources. By utilizing JWTs for both authentication and authorization, HRP ensures that only authorized users can access restricted areas, while maintaining a secure and efficient system where access control is handled directly through the token’s embedded data.

8 Platform

Programming Language	Python 3.x
API connection protocol	HTTP
API type	RESTful
API Technology	Flask (will be upgraded to FastAPI)
ORM	SQLAlchamy
Data Provider	psycopg2
Database	PostgresSQL 16
In-memory Database	Redis

9 Appendix I: Acknowledgement of Assistance

The content of this document has been generated with the support of **ChatGPT**, an AI language model, based on the detailed diagrams, explanations, and specifications provided by **Mahyar Esteki**. The project details, technical components, and system architecture described within this document are derived directly from the information shared by Mahyar Esteki and have been synthesized in collaboration with ChatGPT to ensure clarity and coherence.

This document is a reflection of the original insights provided by Mahyar Esteki, ensuring that the work is both transparent and properly attributed. No part of the content is derived from external or unauthenticated sources, and every effort has been made to maintain the integrity and accuracy of the information presented.