16 de julio de 2025

AYRTON YOSHUA CALAHORRANO VILLAMARIN

central university of ecuador

Ing. Juan Pablo Guevara

PROJECT TITLE: dIstributed plataform for personalized clothing sales

course: Distributed programing

**Content**

[**Introduction** 2](#_Toc199449401)

[**Software Architecture** 2](#_Toc199449402)

[**General Overview** 2](#_Toc199449403)

[**High level Software Architecture Diagram** 2](#_Toc199449404)

[**Backend Languages and Design Principles** 2](#_Toc199449405)

[**Communication Styles** 2](#_Toc199449406)

[**Infrastructure architecture** 3](#_Toc199449407)

[**High-Level Infrastructure Diagram** 3](#_Toc199449408)

[**Infrastructure Components** 4](#_Toc199449409)

[**DevOps and Operability** 4](#_Toc199449410)

[**Functional Modeling** 4](#_Toc199449411)

[**Use Case Diagrams** 4](#_Toc199449412)

[**Process Flow Diagram** 5](#_Toc199449413)

[**State Diagram** 5](#_Toc199449414)

[**Security Model** 6](#_Toc199449415)

[**Disaster Recovery Plan** 6](#_Toc199449416)

[**Annual Maintenance Cost** 6](#_Toc199449417)

[**Conclusions** 7](#_Toc199449418)

[**References** 7](#_Toc199449419)

# **Introduction**

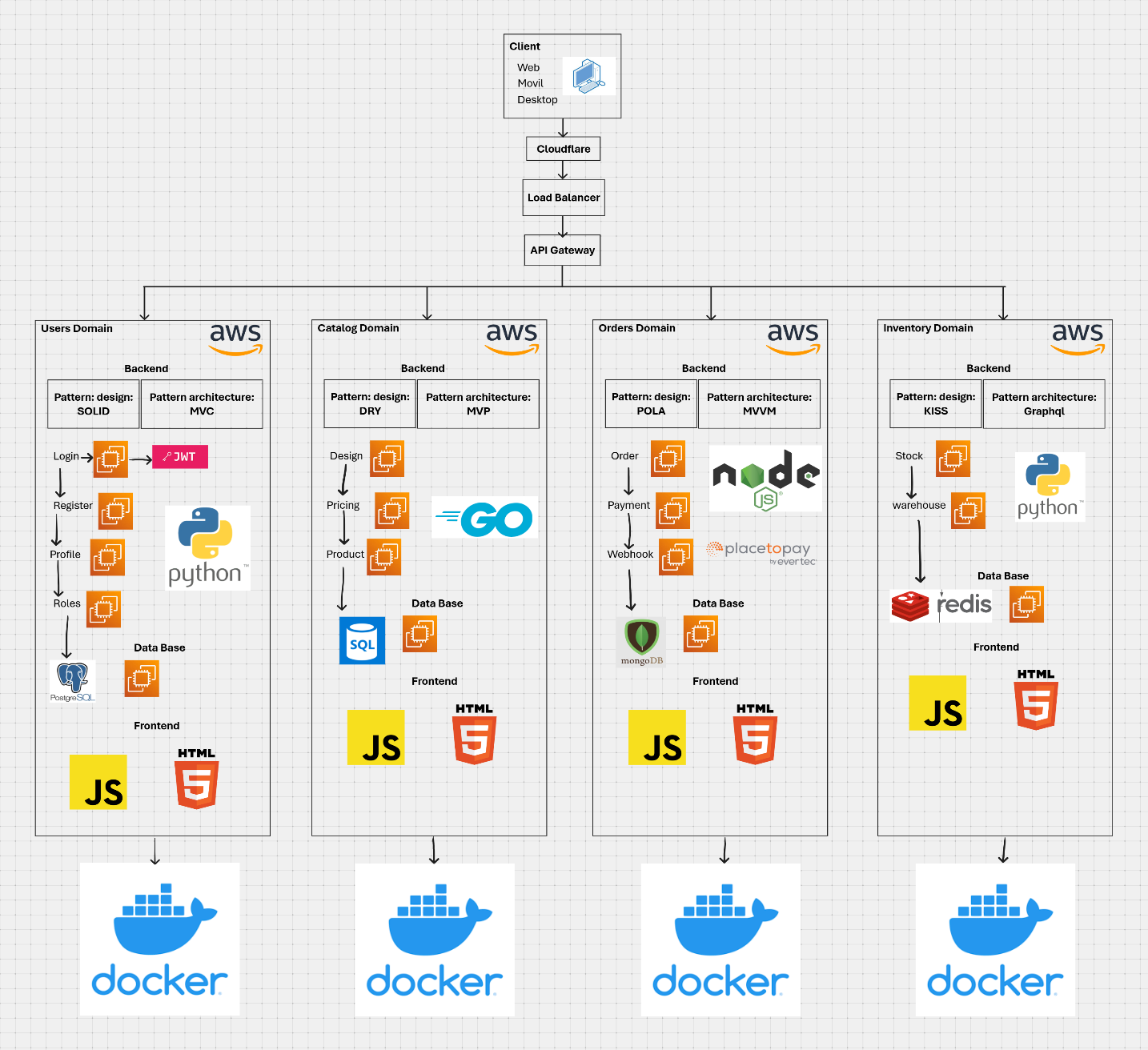
This Project presents a distributed system designed for sale of personalized clothing. The platform is structured using microservices and incorporates multiple backend technologies, architectural styles, and observability tools. The goal is to ensure scalability, maintainability, and fault tolerance in a real-world environment.

# **Software Architecture**

## **General Overview**

The system is divided into five business domains: Users, Catalog, Orders, Inventory, and Analytics. Each domain contains several microservices communicating with REST, gRPC, SOAP, or Webhooks.

## **High level Software Architecture Diagram**



## **Backend Languages and Design Principles**

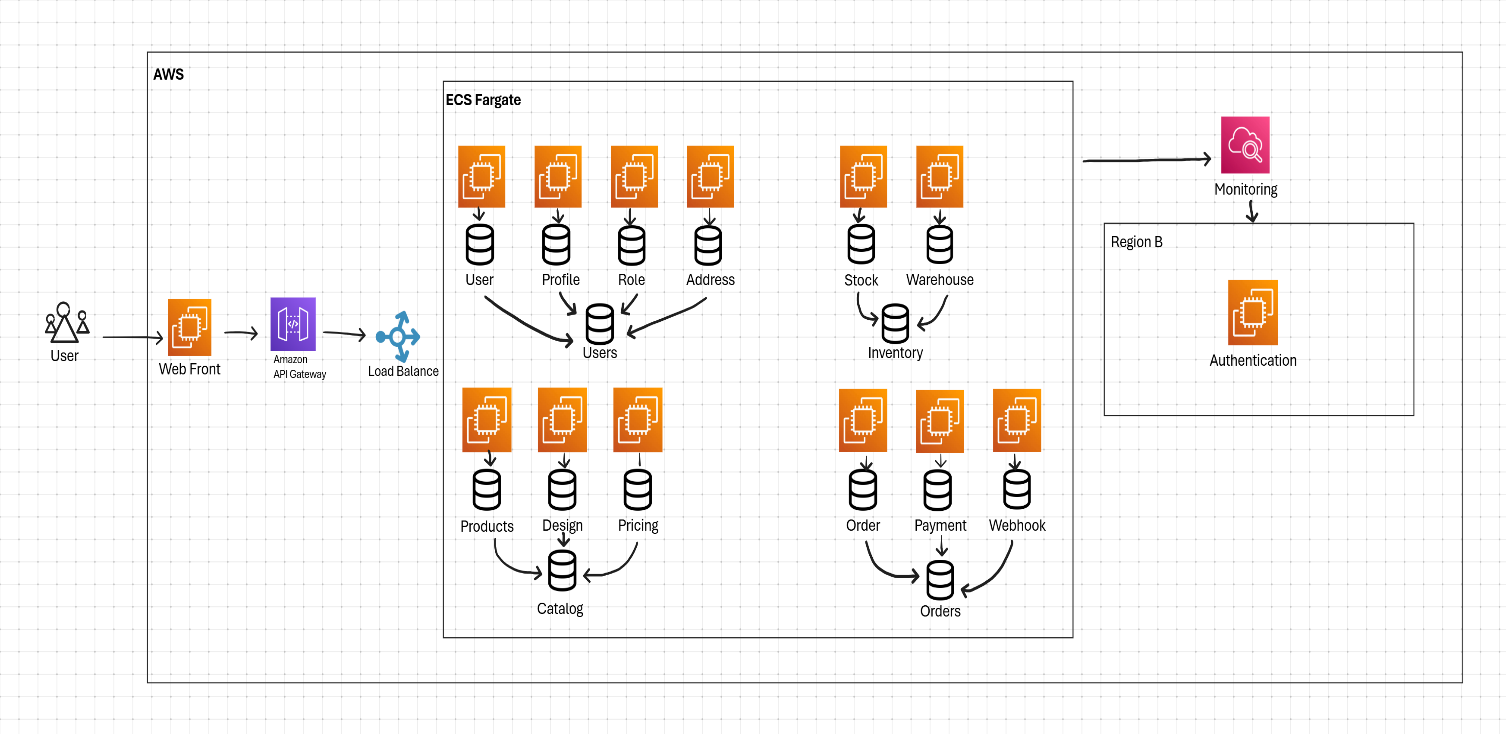
* Languages used: Python, HTML, JavaScript, Go, Node.js
* Design principles: SOLID, DRY, KISS and POLA
* Documentation follows clean code practices, Swagger for API docs, and conventional commits.

## **Communication Styles**

* REST: CRUD operations
* gRPC: Inter-service communication
* Webhook: Notifications to third-party services
* SOAP: External billing system integration (PlacetoPay API for payment precessing)

# **Infrastructure architecture**

## **High-Level Infrastructure Diagram**



## **Infrastructure Components**

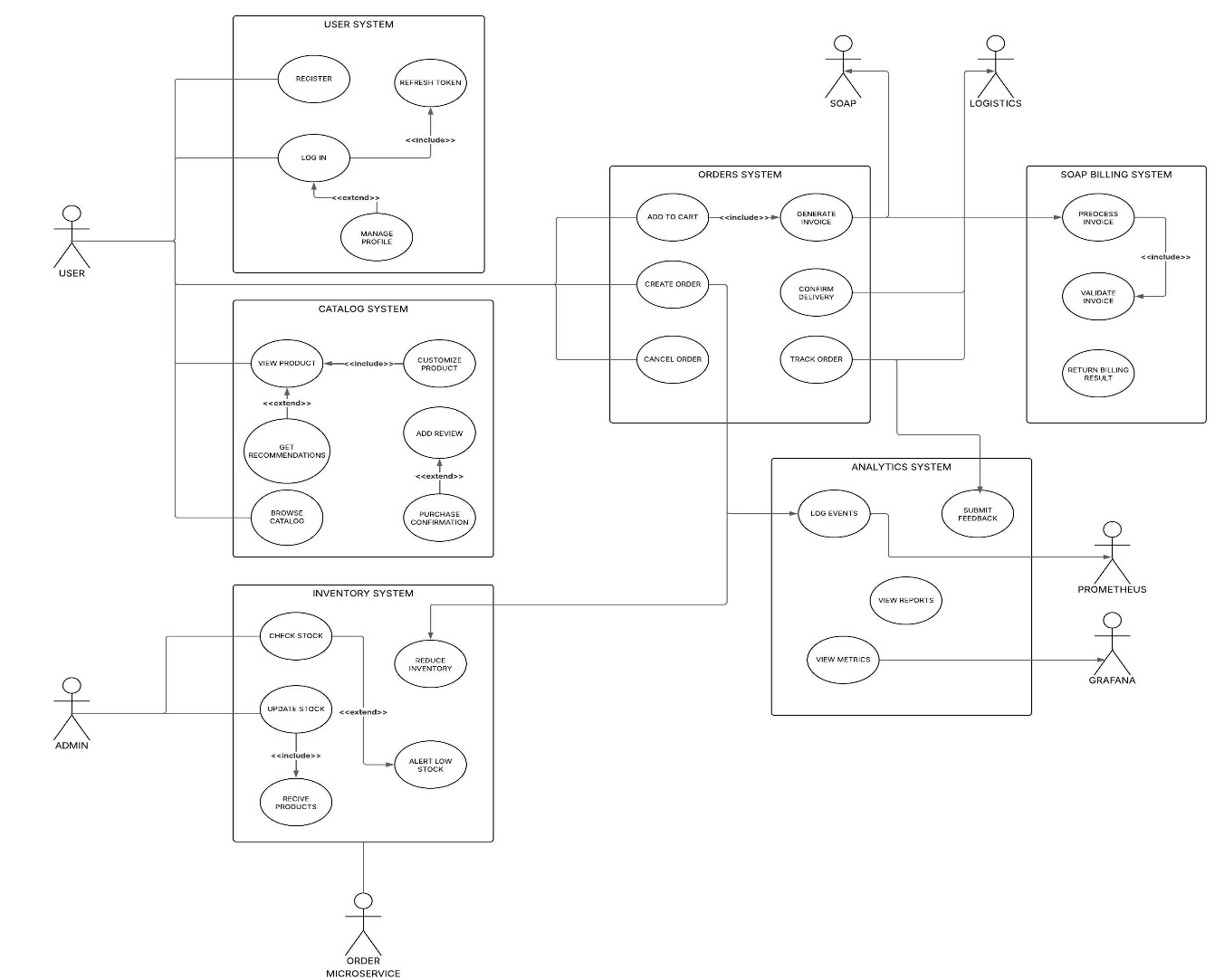
* AWS ECS: with containers for microservices
* PostgreSQL, MongoDB, Redis for persistence
* Kafka for messaging
* Terraform for infrastructure as Code (IaC)
* Load Balancer, API Gateway, NAT, and VPC subnets for network control
* Cloudflare for web application firewall and DNS
* Pay pal Soap API for payment processing

# **DevOps and Operability**

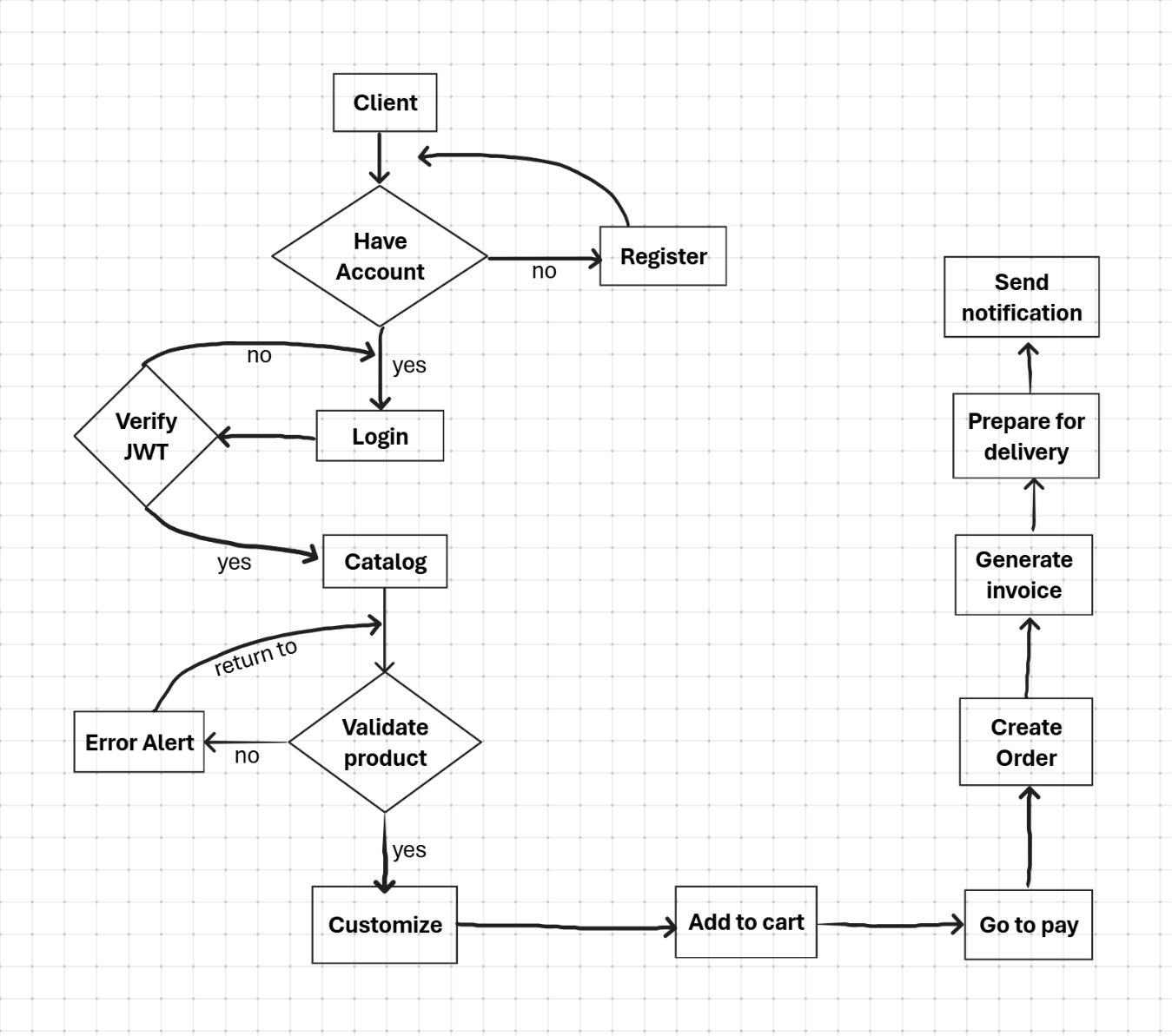
* CI/CD pipelines implemented via GitHub Actions
* DockerHub used for image storage
* Pull request required for production approval
* JWT and CORS implemented across all environments
* Monitoring with Prometheus and Grafana
* Centralized logging via CloudWatch

# **Functional Modeling**

## **Use Case Diagrams**



## **Process Flow Diagram**



## **State Diagram**

Diagrama

El contenido generado por IA puede ser incorrecto.

# **Security Model**

* JWT for authentication in all services
* Role-based access control
* Bastion Host for QA/Production secure access
* Cloudflare for rate-limiting and security filtering

# **Disaster Recovery Plan**

To ensure the continuity and resilience of the platform in the face of unexpected failures, the system incorporates a comprehensive disaster recovery strategy based on cloud-native principles, redundancy, and automation.

## **Multi-AZ Deployment**

All critical components (ECS services, RDS, Kafka, and Redis) are deployed across multiple AWS Availability Zones to guarantee high availability. This architecture ensures that if one zone fails, traffic and services are automatically rerouted to healthy zones without manual intervention.

## **Automated Backups**

Databases (PostgreSQL, MongoDB) and configuration data are backed up daily using AWS backup solutions. Snapshots are stored in S3 with lifecycle policies that allow quick restoration and cost-efficient retention.

## **Infrastructure-as-Code for Rapid Recovery**

The entire infrastructure is defined using Terraform. In case of a complete environment failure, the platform can be redeployed in minutes in a new region or AZ using the latest version of the Terraform state.

## **Observability and Alerting**

The system is continuously monitored via Prometheus, Grafana, and CloudWatch. Custom alerts are configured for CPU usage, memory pressure, container crashes, network anomalies, and queue lags in Kafka. Alerting is integrated with email and Slack for immediate response.

## **JumpBox and Secure Access Recovery**

A Bastion EC2 instance (JumpBox) is maintained for emergency SSH access to private subnets. This JumpBox is only accessible via IP whitelisting and MFA.

## **Periodic Testing and Simulation**

Chaos engineering principles are applied in controlled environments using tools like Gremlin or AWS Fault Injection Simulator. These test the system’s behavior under partial outages, high latency, and resource failures to verify recovery paths.

# **Annual Maintenance Cost**

Below is an estimate annual maintenance cost for the cloud infrastructure and related services required by the distributed platform:

|  |  |  |
| --- | --- | --- |
| Category | Service | Estimated annual Cost (USD) |
| Compute | AWS ECS (Fargate) | $1800 |
| Networking | Load Balancer, NAT, VPC | $600 |
| Storage and DB | PostgreSQL (RDS) | $1200 |
|  | MongoDB (EC2 or Atlas) | $800 |
|  | Redis (ElastiCache) | $700 |
| Messaging | Kafka (MSK or EC2 setup) | $1000 |
| Monitoring and Logging | Prometheus, Grafana, CW | $600 |
| Security and edge | Cloudflare (pro plan) | $240 |
| DevOps Tooling | GitHub (Teams), DockerHub | $480 |
| Infrastructure as Code | Terraform Clod (Team) | $240 |
| Domain and Miscellaneous | DNS, backups, bandwidth | $340 |
| Total |  | $8000/year |

# 

# **Conclusions**

This project demonstrates how distributed system principles can be applied to build a robust, scalable, and maintainable platform. Using multiple programming languages, communication protocols, and DevOps practices, the system achieves real-world reliability.

# **References**

* AWS Documentation
* Apache Kafka Official Docs
* Prometheus and Grafana
* Swagger/OpenAPI
* Clean Code by Robert C. Martin
* Microservices.io
* PayPal Developer Documentation (SOAP API)