Requirement Analysis Document (RAD)

Project: CI/CD Pipeline Health Dashboard

Prepared for: Engineering / DevOps Team

Version: 1.0

Date: [Insert Date]

# 1. Introduction

## 1.1 Purpose

The CI/CD Pipeline Health Dashboard provides visibility into the health and performance of software delivery pipelines (initially GitHub Actions). It centralizes key metrics, alerts, and logs into a single, user-friendly interface, enabling engineering teams to detect failures quickly, analyze trends, and maintain high release reliability.

## 1.2 Scope

- Data ingestion: Poll pipeline execution metadata from GitHub Actions.  
- Metrics computation: Success/failure rate, average build duration, last build status.  
- Alerting: Slack notifications for failures with log snippets.  
- Visualization: Interactive dashboard (KPIs, charts, recent runs, run details with jobs/logs).  
- Deployment: Local Dockerized environment with backend, frontend, and database services.

# 2. Objectives

- Provide real-time visibility into CI/CD health.  
- Reduce MTTR (mean time to resolution) by surfacing failures with context.  
- Support developer productivity through simple, attractive, and intuitive UI.  
- Enable trend analysis of builds over time (success/failure distribution, average durations).  
- Ensure portability: solution runs locally in Docker, without dependency on cloud services.

# 3. Functional Requirements

## 3.1 Data Collection

- Source: GitHub Actions (all accessible repos + branches).  
- Polling interval: Default 30s (configurable).  
- Runs captured: Latest N runs per repo (default 50).  
- Logs: Downloaded only for failed jobs; gzipped storage with 7-day retention.

## 3.2 Metrics

- Success rate (%)  
- Failure rate (%)  
- Average build time  
- Last build status  
- Time series distribution

## 3.3 Alerts

- Channel: Slack Incoming Webhook.  
- Trigger: On every failed workflow run.  
- Content: Repo/branch, workflow name, conclusion, duration, run URL, failed jobs, and log snippet.  
- Mentions: @channel supported.

## 3.4 User Interface

- KPIs: Success %, failure %, avg duration, last build outcome.  
- Charts: Bar chart (build outcomes by day), Line chart (average duration by day).  
- Recent Runs Table: Run ID, repo, branch, status, conclusion, duration, links.  
- Run Details Modal: Jobs table + log snippet from failed job.  
- Features: Dark mode, auto-refresh, manual refresh, error states.

## 3.5 System Administration

- Configuration: .env file (GitHub PAT, Slack webhook, polling interval, DB credentials).  
- Persistence: Postgres + run logs volumes.  
- Security: Credentials managed via .env, excluded from git.

# 4. Non-Functional Requirements

## Performance

Must ingest builds for all repos within rate limits; optimize using sharding and ETags.

## Scalability

Handle dozens of repos and hundreds of builds efficiently; schema extensible for multiple CI providers.

## Availability

Containers auto-restart on failure; dashboard remains accessible as long as Docker is running.

## Security

PAT scoped to repo, read:org, workflow; Slack webhook secured; configs excluded via .gitignore.

## Usability

Responsive UI, mobile friendly, clear colors for states, dark mode support.

# 5. System Architecture

## 5.1 Components

1. Backend (FastAPI): API, scheduler, Slack integration, log storage.  
2. Database (Postgres): stores repos, runs, jobs, logs metadata.  
3. Frontend (React + Tailwind): renders dashboard UI.  
4. Nginx: serves static frontend, proxies /api.  
5. Slack: alert channel.

## 5.2 Deployment Topology

All services containerized via Docker Compose.  
Exposed ports: API 8080, Frontend 3001.  
Data persisted in Docker volumes.

# 6. Assumptions & Constraints

- Only GitHub Actions supported initially.  
- Requires GitHub PAT.  
- Alerts limited to Slack in MVP.  
- Local deployment only.  
- User manages token lifecycle.

# 7. Risks

- API rate limits → mitigated with sharding & ETags.  
- Data growth → mitigated with compressed logs and retention.  
- Slack noise → cooldowns may be needed.  
- Token misuse → strict handling of .env required.

# 8. Future Enhancements

- Support Jenkins, GitLab CI, CircleCI.  
- Role-based access control.  
- Advanced alert policies.  
- Export to Grafana/Prometheus.  
- Multi-channel notifications (Email, Teams).  
- PDF/CSV run history export.

# 9. Acceptance Criteria

- Dashboard shows KPIs and charts within 30s.  
- User can filter by repo/branch/time window.  
- Runs table & details modal fetch live data.  
- Slack alerts fire within 1 minute of failure.  
- Logs retained for 7 days.  
- UI responsive, supports dark mode.  
- Sensitive configs stored in .env only.