DevOps Project Proposal

1. Project Description

This project focuses on building, containerizing, and monitoring a functional URL shortener webservice. The entire stack, including the application and its monitoring tools (Prometheus, Grafana), will run locally using Docker. The webservice will store, map, and handle redirects for shortened URLs. Custom metrics will be instrumented using Prometheus, and a comprehensive dashboard for visualizing service health and usage patterns will be created with Grafana.

2. Group Members & Roles

Name	Role	Email	ID
Mahmoud El-Shahat Saeid Mohamed	Team Leader	mahmoudelshahat595@gmail.com	21033925
Youssef Elsayed Elsayed Elshamekh	Team Member	elshamekhyoussef@gmail.com	21052550
Abdulrahman Muhammad Sayed Saeed	Team Member	am6958@fayoum.edu.eg	21094445
Abdulrahman mohammed elshazly	Team Member	Abdulrahamedleshazly028@gmail.com	201031774

3. Objectives

The main objectives of this project are:

- **Develop and Containerize a URL Shortener Webservice:** Create a simple URL shortener application with primary API endpoints (POST for shortening, GET for redirection) and containerize it using Docker.
- **Implement Data Storage:** Utilize SQLite for storing shortened URLs and their corresponding long URLs.
- **Instrument with Custom Metrics:** Modify the webservice code to track key operational metrics such as the number of URLs shortened, successful redirects, failed lookups (404 errors), and request latency.
- **Configure Prometheus for Monitoring:** Set up Prometheus to scrape custom metrics from the webservice.

- **Visualize Data with Grafana:** Create a Grafana dashboard to visualize the collected metrics, including total URL creations, redirect rates, error rates, and request latency percentiles.
- **Configure Alerting:** Set up meaningful alerts in Grafana for critical events, such as high error rates or increased request latency.
- **Ensure Data Persistence:** Configure Docker volumes for SQLite database and Prometheus/Grafana data to prevent data loss.
- **Document the Project:** Provide comprehensive project documentation, including API specifications and a README.md file.

4. Tools & Technologies

The following tools and technologies will be utilized in this project:

- **Containerization:** Docker, Docker Compose
- **Monitoring:** Prometheus, Grafana
- **Web Framework:** Python's Flask or Node.js's Express (as specified in the project idea, choice will be made during implementation)
- **Database:** SQLite
- **Version Control:** Git
- **CI/CD Workflows:** Jenkins, Docker, and Ansible (as suggested in the proposal guidelines for Infrastructure & Automation, though the project description focuses on local Docker deployment for monitoring)

5. Milestones & Deadlines

The project will be divided into weekly milestones.

Week	Focus Area	Deliverables
Week 1	Build & Containerize the URL Shortener	- Functional URL shortener webservice with source code. - Dockerfile for building a runnable image. - docker-compose.yml for starting the webservice. - Locally running and successfully shortening/redirecting URLs.
Week 2	Instrumenting with Custom Prometheus Metrics	- Updated webservice exposing custom metrics. - prometheus.yml configured to scrape webservice metrics. - docker-compose.yml updated with Prometheus service. - Custom metrics visible in Prometheus UI.
Week 3	Advanced Visualization with Grafana	- Full docker-compose.yml with webservice, Prometheus, and Grafana. - Custom Grafana dashboard with actionable insights. - Running stack where metrics change on dashboard in real-time.

Week	Focus Area	Deliverables
Week 4	Alerting, Persistence, and Documentation	- docker-compose.yml with persistent volumes. - Alerting rules configured in Grafana. - Fully tested, stable, and persistent local webservice and monitoring stack. - Comprehensive project documentation including API specifications.

6. KPIs (Key Performance Indicators)

To measure the success and performance of the project, the following KPIs will be customized and tracked:

- **Infrastructure & Automation:**
  - Successful setup of Jenkins, Docker, and Ansible for automated CI/CD workflows (if implemented beyond local Docker deployment).
- **Pipeline Efficiency & Performance:**
  - Optimized build, test, and deployment stages with minimal downtime.
- **Code Integration & Testing:**
  - Automated unit and integration testing with seamless Git repository integration.
- **Deployment & Cloud Management:**
  - Successful cloud deployment using Ansible, Docker Hub, and Kubernetes (if applicable).
- **Monitoring & Reliability:**
  - Implementation of notifications, logging, and system uptime tracking.
  - Number of URLs shortened per minute.
  - Percentage of successful redirects.
  - Rate of 404 errors for invalid shortened URLs.
  - 95th percentile request latency for URL shortening and redirection.
  - Uptime of the URL shortener webservice and monitoring stack.
  - Alerts triggered for critical events (e.g., high error rates, increased latency).