Project Title: Load-Balanced URL Shortener using Docker & Kubernetes

Problem Statement:

Develop a containerized URL shortener service that allows users to submit long URLs and get a shortened version. The system should be scalable using Docker and Kubernetes, with a load balancer distributing requests across multiple instances. Data should be stored in an in-memory key-value store running in a separate container.

Week 1: Build the URL Shortener in Docker

Objective: Develop and containerize the basic URL shortener service.

- **Task 1:** Write a simple Python or Node.js application to:
    - Accept long URLs via an API.
    - Generate a short URL.
    - Redirect users from short URLs to the original long URLs.
- **Task 2:** Use an in-memory key-value store (e.g., Redis or a dictionary in Python) to store URL mappings.
- **Task 3:** Write a Dockerfile and build a Docker container for the application.
- **Task 4:** Test the containerized app locally using docker run.

Deliverable: A working Docker container running a URL shortener.

Week 2: Deploy the URL Shortener using Kubernetes

Objective: Deploy the application on Kubernetes and configure networking.

- **Task 1:** Write Kubernetes Deployment & Service manifests for:
    - URL Shortener Pod (runs multiple instances of the app).
    - Key-Value Store Pod (stores shortened URLs).
    - ClusterIP Service to expose the key-value store internally.
    - LoadBalancer/NodePort Service to expose the URL shortener.
- **Task 2:** Deploy the application on Kubernetes using kubectl apply.

- **Task 3:** Test the API with multiple instances running.
- **Task 4:** Implement Kubernetes ConfigMaps & Secrets for managing environment variables.

Deliverable: A Kubernetes cluster running the URL shortener with multiple replicas.

Week 3: Scaling, Load Balancing & Monitoring

Objective: Make the system scalable and fault-tolerant.

- **Task 1:** Set up Kubernetes Horizontal Pod Autoscaler (HPA) to scale based on CPU usage.
- **Task 2:** Implement Kubernetes Ingress or a LoadBalancer for better traffic routing.
- **Task 3:** Configure Kubernetes Logs and kubectl get pods / logs for monitoring.
- **Task 4:** Conduct stress testing to see how the system scales under load.

Deliverable: A scalable & load-balanced URL shortener running on Kubernetes.

Additional Implementation (Optional):

1) **Database Integration -**

- Replace the in-memory key-value store with a persistent database like PostgreSQL, MongoDB, or DynamoDB, so shortened URLs remain available even after a system restart.

2) **CI/CD & DevOps Enhancements -**

- Set up a GitHub Actions / Jenkins pipeline to automate testing and deployment.