**Phase 2: Design Document: Scalable Web Application using Open Liberty on IBM Cloud**

**1. Design Objectives**

The main goal of this project is to design a scalable, containerized, and cloud-native Java web application using Open Liberty on IBM Cloud Kubernetes Service. The design aims to ensure:

* Horizontal scalability using Kubernetes
* High availability via load balancing
* Observability using MicroProfile Metrics & Health
* Secure deployment and configuration handling
* CI/CD automation for rapid iteration

**2. System Architecture**

The application is deployed using Docker containers and orchestrated with Kubernetes. The core architectural layers include:

+-----------------------+

| Web Client |

+-----------------------+

↓

+-----------------------+

| IBM Cloud Ingress |

| (HTTPS Endpoint) |

+-----------------------+

↓

+----------------------------+

| Kubernetes Load Balancer |

+----------------------------+

↓

+-----------------------------------------+

| Liberty Application Pods |

| - REST APIs via JAX-RS |

| - Config via MicroProfile |

| - Metrics & Health Checks |

+-----------------------------------------+

↓

+---------------------------+

| Cloud Database (PostgreSQL) |

+---------------------------+

**3. Component Design**

**3.1 Open Liberty Application**

* Framework: Open Liberty with Jakarta EE and MicroProfile.
* Features Used:
  + JAX-RS: For RESTful services
  + MicroProfile Config: Externalized configuration
  + MicroProfile Health: Liveness and readiness endpoints
  + MicroProfile Metrics: For Prometheus scraping

Sample REST Endpoint:

@GET

@Path("/status")

public Response getStatus() {

return Response.ok("Service is healthy").build();

}

**3.2 Containerization**

The app is packaged as a Docker image using a minimal Liberty base image.

**Dockerfile:**

FROM icr.io/appcafe/open-liberty

COPY target/\*.war /config/dropins/

COPY config/server.xml /config/

Benefits:

* Portability
* Fast deployments
* Environment isolation

**3.3 Kubernetes Deployment**

The Kubernetes deployment includes:

* Deployment YAML: Defines pod specs, replicas, and labels.
* Service YAML: Exposes the Liberty pod via ClusterIP or LoadBalancer.
* Ingress Controller: Manages public access via HTTPS.

**Text-Based Flow:**

Ingress Controller

↓

Kubernetes Service (LoadBalancer)

↓

Liberty Pod(s)

↓

Database

**4. Configuration & Secrets**

* Config handled via Kubernetes ConfigMaps and Secrets.
* Externalized settings like DB credentials and app configs.
* Loaded into the container at runtime via environment variables.

MicroProfile Config Example:

db.username=${DB\_USER}

db.password=${DB\_PASS}

**5. Observability & Monitoring**

Implemented using MicroProfile Metrics and Health Checks.

Endpoints:

* /health/live – Liveness check
* /health/ready – Readiness check
* /metrics – Application and system metrics

Prometheus scrapes the metrics for dashboarding and alerting.

**6. Auto Scaling & Load Balancing**

* Horizontal Pod Autoscaler (HPA) adjusts pod count based on CPU usage.
* Kubernetes Service distributes traffic evenly among pods.
* Rolling updates allow zero-downtime deployments.

**7. Security Design**

* Ingress Layer: TLS termination via HTTPS
* Secrets Management: Secure handling of sensitive data using Kubernetes Secrets
* Optional integration with IBM Cloud IAM and identity providers

**8. Deployment Workflow (Text Diagram)**

Developer Commit

↓

CI/CD Pipeline

↓

Build Docker Image

↓

Push to IBM Cloud Container Registry

↓

Deploy to Kubernetes (Helm/Manifest)

↓

Monitor with Prometheus/Grafana

**9. Summary**

The design ensures:

* Scalable deployment using Kubernetes
* Microservice readiness with Open Liberty
* Observability via standard MicroProfile tools
* Secure and configurable runtime using ConfigMaps and Secrets
* Efficient delivery through CI/CD automation