HOSTING, DEPLOYMENT, MAINTENANCE, AND SUPPORT DOCUMENTATION

Global Alliance for Improved Nutrition (GAIN) Project

Abstract

The GAIN Project enhances food fortification monitoring and quality assessment through FortifyMIS and Digital FQT+. This document details the hosting, deployment, maintenance, and support strategies, leveraging AWS cloud infrastructure with Kubernetes-based deployment and CI/CD automation.

Key focus areas include scalability, security, continuous monitoring, backup & disaster recovery, and a structured support framework with SLA-based issue resolution. This guide ensures the project's reliability, efficiency, and long-term sustainability.

CIRT AND INFRA TEAM, BUSINESS AUTOMATION LTD.

Version: 1.1

Project Overview

The **Global Alliance for Improved Nutrition (GAIN) Project** aims to enhance food quality and nutritional standards through digital solutions. The project consists of two sub-projects:

- **FortifyMIS**: A management information system (MIS) for monitoring fortification programs.
- **DFQT**: A digital solution for food quality testing and analysis.

Document Purpose

This document provides a detailed overview of:

- Hosting infrastructure
- Deployment procedures
- Maintenance strategies
- Support and issue resolution processes

Project 1: FortifyMIS Hosting Deployment Procedure

1. Hosting Infrastructure

1.1 Hosting Provider and Environment

- **Cloud Provider**: AWS (Amazon Web Services).
- **Hosting Setup**: Kubernetes-based infrastructure using lightweight K3s clusters.
- **Storage**: AWS EBS (50GB).
- **Database**: AWS RDS (MySQL-based).
- **Security Controls**: IAM roles, encryption, firewall configurations.

1.2 Resource Allocation

Component	Instance Type	VCPUs	RAM	Storages	Network
K3S Instance & GitLab CI	c5.xlarge	4	8 GB	50 GB EBS	Up to 10 Gbps

1.3 Estimated Hosting Costs

- **Monthly Cost**: ~$70
- **Annual Cost**: ~$840

2. Deployment Procedure

2.1 Deployment Strategy

- **CI/CD Pipeline**: Automated using Argo CI/CD
- **GitOps Practices**: Ensures traceability and version control
- **Security Measures**: Role-based access, authentication, and encrypted data transfers

2.2 Deployment Steps

1. **Code Commit & Version Control**
   - Code changes are committed to GitLab and reviewed through merge requests.
2. **Automated Build & Testing**
   - CI/CD pipeline initiates automated testing and builds the application.
3. **Containerization & Image Deployment**
   - Docker images are built and pushed to the container registry.
4. **Kubernetes-Based Deployment**
   - Kubernetes manages deployment across EC2 instances with auto-scaling.
5. **Post-Deployment Validation**
   - System verification, smoke testing, and rollback mechanisms ensure stability.

3. Maintenance & Monitoring

3.1 System Monitoring & Logging

- **Tools Used**: AWS CloudWatch, Prometheus, Grafana monitored by Business Automation Ltd.
- **Monitoring Scope**: System uptime, API latency, and database performance by Business Automation Ltd.
- **Incident Handling**: Automated alerts and log-based debugging managed by Business Automation Ltd.

3.2 Regular Maintenance Activities

Activity	Frequency	Description
System Updates	Monthly	Security patches and performance updates
Database Optimization	Weekly	Indexing, performance tuning, backups
Log Analysis	Continuous	Issue detection, troubleshooting
Backup Validation	Weekly	Ensuring recoverability of stored data

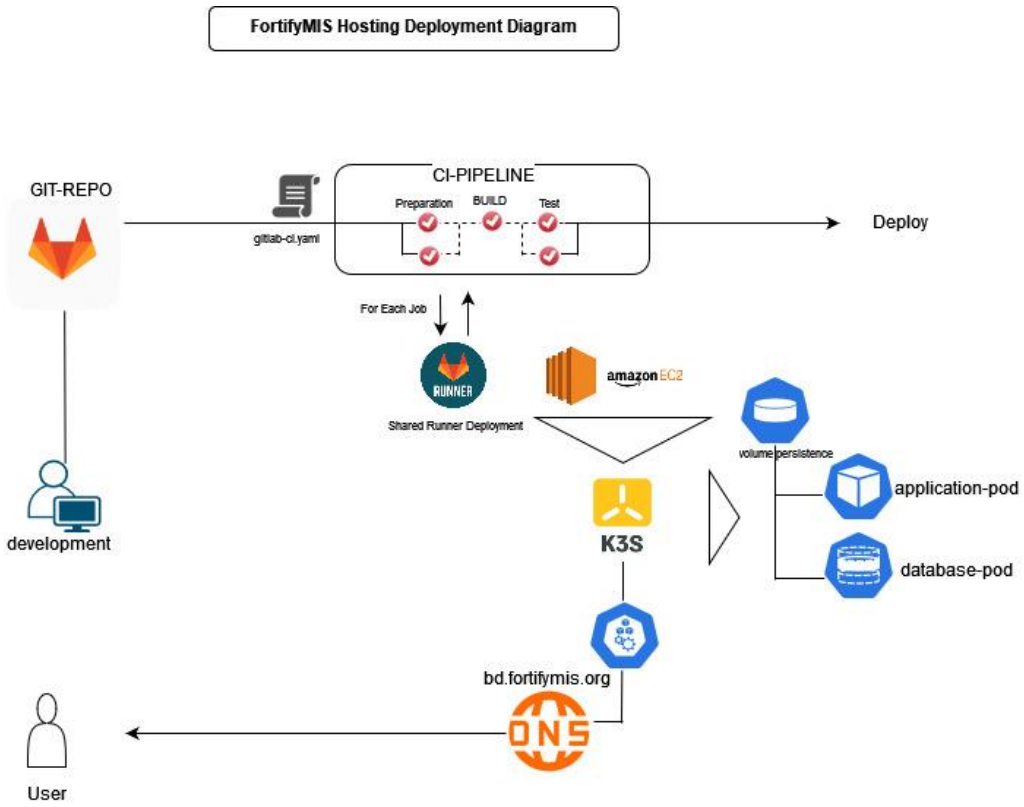

Figure 1: FortyMIS Hosting Deployment Diagram

Project 2: Kubernetes Deployment Documentation for DFQT Project

1. Overview

This document provides the procedure for deploying the **DFQT** application to a **Kubernetes cluster**. The deployment process involves retrieving the application code, following deployment instructions, and ensuring successful synchronization with the Kubernetes cluster. It also includes steps for monitoring the deployment, checking the UI, and notifying stakeholders (GAIN) about the deployment status.

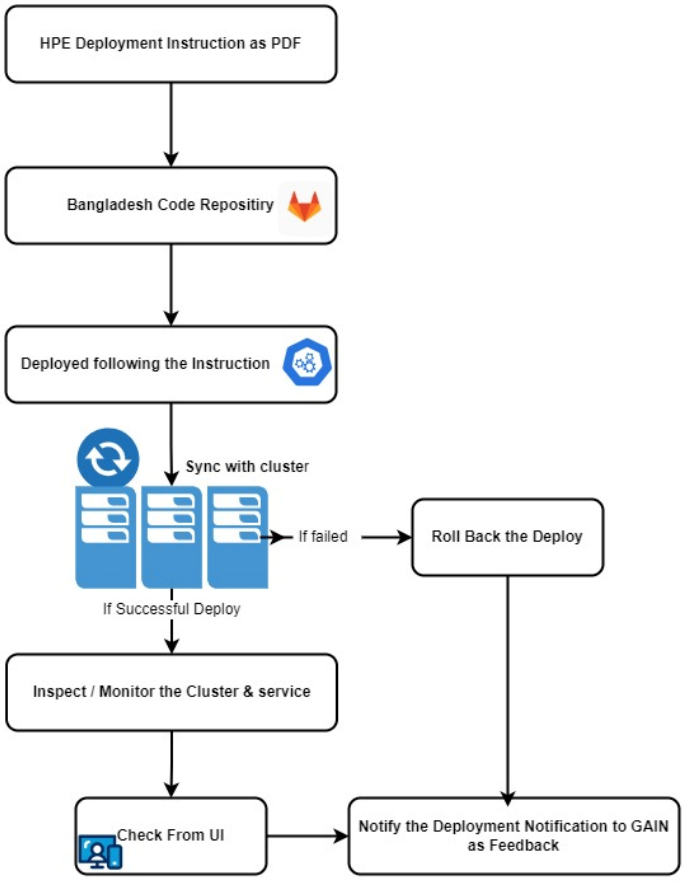

Fig 2: DFQT Deployment procedure of Business Automation Ltd.

2. Deployment Process Flow

2.1 Step 1: HPE Deployment Instruction as PDF

The deployment process begins with receiving the **HPE Deployment Instructions** in **PDF format**. These instructions contain the necessary steps and configurations to deploy the application properly in the Kubernetes environment.

- **Purpose**: To ensure the deployment is done according to the guidelines specified for the DFQT project.
- **Format**: PDF document containing detailed steps.

2.2 Step 2: Retrieve Code from Bangladesh Code Repository

The next step involves retrieving the **DFQT source code** from the **Bangladesh Code Repository** hosted on **GitLab**.

- **Repository**: The code repository should contain the latest version of the application's source code.
- **Action**: Clone or pull the latest version of the code using the GitLab repository URL.

2.3 Step 3: Deploy Code Following the Instructions

The code is deployed based on the instructions provided in the **HPE Deployment PDF**. This step involves setting up the necessary configurations, building the application, and pushing it to the Kubernetes cluster.

- **Action**: Deploy the code using the instructions to ensure the application is configured correctly.
- **Key Technologies**: GitLab, Helm (for Kubernetes), Docker images (for containerization).

2.4 Step 4: Sync with Cluster

Once the deployment is initiated, synchronize the deployed code with the Kubernetes cluster.

- **Action**: The application deployment is synchronized to ensure consistency and correct configurations are applied to the Kubernetes environment.
- **Verification**: Syncing ensures that the deployment in the cluster matches the intended configuration.

2.5 Step 5: Deployment Success Check

After the synchronization, a check is performed to verify if the deployment was successful. If the deployment fails, the process triggers a rollback.

- **Success Criteria**:
  - Application is successfully deployed and is operational in the cluster.
  - No errors or failures during synchronization.

- **Failure Criteria**:
  - Deployment is unsuccessful due to misconfigurations or failed deployment scripts.
  - In case of failure, a **rollback** to the previous stable version of the application is triggered.

2.6 Step 6: Monitor Cluster and Service

Once the deployment is successful, the cluster and application services are continuously monitored for performance, health, and availability.

- **Monitoring Tools**: AWS CloudWatch, Prometheus, Grafana.
- **Action**: Inspect logs and metrics to ensure proper functionality and detect any issues.

2.7 Step 7: Check from UI

The **User Interface (UI)** is checked to verify that the application functions as expected from the user's perspective.

- **Action**: Perform manual testing or automated UI tests to ensure the deployed code is accessible and operational.

2.8 Step 8: Notify Deployment Feedback to GAIN

After a successful deployment and verification, the final step is to notify the relevant stakeholders (GAIN) about the deployment status.

- **Notification Format**: Email or automated deployment notification system.
- **Content**: Deployment success, version details, and any relevant information.

3. Kubernetes Cluster Setup in BCC Environment

The **Kubernetes cluster** is hosted in the **BCC environment**, providing a stable and secure platform for deploying and managing the DFQT project. All necessary configurations for the project have been prepared and are regularly maintained to ensure smooth deployment and operation.

- **Cluster Provider**: BCC environment
- **Configuration Management**: All configurations for the project are pre-configured to ensure seamless deployment and operation received from GAIN.
- **Maintenance**: Business Automation Ltd. performs regular maintenance, which includes monitoring the cluster, checking resources, and upgrading the system as needed.
- **Auditing**: Business Automation Ltd. performs audits regularly to ensure that the system complies with security standards and regulatory requirements and to notify the GAIN.

4. Monitoring and Support

The deployment is monitored in real-time using integrated monitoring tools, and the system is continuously evaluated to ensure it operates efficiently and securely.

- **Monitoring Tools**: AWS CloudWatch, Prometheus, Grafana.
- **Continuous Monitoring**: Logs, metrics, and system health are monitored continuously to ensure operational efficiency.
- **Incident Management**: Automated alerts and logging systems help to quickly address issues, with a structured support model for issue resolution.

5. Proposed Optimization: Resource Reduction (DFQT Project)

To enhance cost efficiency while maintaining operational effectiveness, a **resource optimization strategy** has been approved for the **DFQT Project**. The proposal involves reducing total resource allocation by **50%**, adjusting CPU to **10 cores** and memory to **20 GB**, leading to projected annual savings of **$1,160 USD**. This reduction aligns with the **current workload capacity** and ensures sufficient computing power under standard operating conditions. However, in case of **increased user demand or deployment expansions**, resources may need to be scaled back to higher levels to maintain system performance and reliability. This optimization is a proactive step toward **cost reduction and efficient resource utilization** without compromising system integrity.