**Functional Specification Document**

**Control Center for Development Fabric**

**1. Introduction**

**1.1 Purpose**

The Control Center is a centralized platform within the Development Fabric that orchestrates the software development lifecycle, integrating **Git** and **Jenkins** for seamless build, test, and deployment pipelines. The system provides role-based access control (RBAC), manual build triggers, environment management, and rollback functionality.

**1.2 Scope**

This document defines the functionality and technical requirements for the Control Center, which will:

* Allow developers to commit code directly within the Fabric.
* Push code to Git repositories.
* Manually trigger Jenkins pipelines for build, test, and deployment.
* Use **Camunda** for workflow orchestration, task management, and approvals.
* Provide an approval-based rollback mechanism.
* Support multi-environment configurations (Dev, QAT, UAT, Prod).
* Implement Role-Based Access Control (RBAC).

**2. Functional Requirements**

**2.1 User Roles & Access Control**

|  |  |
| --- | --- |
| **Roles** | **Permission** |
| Release Manager | Can send build to QAT, UAT and Live and can approve UAT and Production deployments |
| Developer | Develop interfaces and can push code to Git, trigger builds |
| QA Tester | Test the build (Automated and Manual) |
| UA Tester | Test the build |
| Admin | Full Access |

**2.2 Key Features**

**2.2.1 Code Management**

* Developers write and update code directly in the **Fabric IDE**.
* Control Center automatically commits code changes to **Git repositories**.
* Push-to-Git functionality is **manual**, triggered via a button in the Control Center.

**2.2.2 Build & Deployment Pipeline**

* Users manually trigger builds via a **“Trigger Build”** button.
* Control Center sends an **API request** to Jenkins to initiate a build.
* Jenkins builds the code and provides **status updates** to the Control Center.
* A **“Deploy to QAT”** button is available for Release Manager to push a build to the QAT environment.

**2.2.3 Rollback Mechanism**

* Stores metadata of **latest and previous successful deployments**.
* “Rollback” button allows reverting to the last stable build.
* **Approval required** from Managers before executing a rollback.
* Logs all rollback actions for auditing.

**2.2.4 Multi-Environment Management**

* Stores Jenkins **URLs, credentials, and deployment servers** for Dev, QAT, UAT, and Prod.
* Environments configurable via **database**.
* UI dropdown to **select target environment** before triggering deployment.

**2.2.5 Status & Notifications**

* Displays real-time **build and deployment status**.
* Sends email/SMS notifications for **successful or failed deployments**.

**3. Technical Design**

**3.1 System Architecture**

* **Frontend:** Angular (UI for triggering builds, deployments, rollbacks, and monitoring status)
* **Backend:** Node.js (API layer to communicate with Jenkins & Git)
* **Database:** MongoDB (Store deployment metadata, logs, and RBAC data)
* **Integration:** Jenkins API & Git API

**3.2 API Endpoints**

| **Endpoint** | **Method** | **Description** |
| --- | --- | --- |
| /api/push-to-git | POST | Pushes latest code changes to Git |
| /api/trigger-build | POST | Starts Jenkins build process |
| /api/deploy/<env> | POST | Deploys build to specified environment |
| /api/rollback/<env> | POST | Rolls back to previous stable build |
| /api/get-status | GET | Retrieves build and deployment status |

**4. User Interface Mockups**

**4.1 Dashboard Overview**

* **Push to Git** button
* **Trigger Build** button
* **Deploy to QAT** button
* **Deploy to UAT** button
* **Deploy to Live** button
* **Rollback Button (Admins Only)**
* **Build Status Table** (Latest build history and logs)

**4.2 Role-Based UI**

|  |  |
| --- | --- |
| **Role** | **Actions to do** |
| **Developer** | Development and push code to Git |
| **Release Manager** | Code Push to QAT environment, Code Push to QAT environment, Build push to UAT Environment, Build push to Live Environment |
| **QA Tester** | QA Test the Build and make pass or fail |
| **UA Tester** | UA Test the build and make pass or fail |
| **Admin** | Rollback if required |

**5. Security & Compliance**

* **RBAC implementation** to restrict actions based on roles.
* **Audit logs** for tracking builds, deployments, and rollbacks.
* **Secure API authentication** using JWT or OAuth.
* **Approval workflow** for rollback actions in production.

**6. Deployment & Monitoring**

* **Jenkins integration** for automation and build history tracking.
* **Logging & Alerts** via ELK Stack / Prometheus + Grafana.
* **Automated Tests** triggered in QAT before moving to UAT.
* **Monitoring & Health Checks** on deployed builds.

**7. Process Flow**

1. **Developer** pushes code to **Git** → Camunda Triggers task for **Release Manager**
2. **Release Manager** pushes to **QAT** → Camunda Triggers task for **QA Tester**
3. **QA Tester** tests and approves/rejects → If approved, Camunda task moves to **Release Manager**
4. **Release Manager** pushes to **UAT** → Camunda Triggers task for **UAT Tester**
5. **UAT Tester** tests and approves/rejects → If approved, Camunda move task to **Release Manager**
6. **Release Manager** pushes to **Live**
7. **Admin** can trigger **rollback** if needed

