MonoRepo CICD Pipeline Implementation Guide

Executive Summary

This document outlines comprehensive approaches for transitioning from individual microservice repositories to MonoRepo-based CICD pipelines while maintaining the existing toolchain (CloudBees CI, SonarQube, Fortify, NexusIQ, uDeploy, PCF).

Current State Analysis

Existing Architecture

- **Repository Structure**: Individual repositories per microservice
- **Pipeline Per Service**: Each microservice has dedicated Jenkins pipeline
- **Build Detection**: Automatic build type detection (Maven/Gradle/Node.js)
- **Security Scanning**: Parallel execution of SonarQube, Fortify, NexusIQ
- **Deployment**: uDeploy to PCF platform
- **Infrastructure**: CloudBees CI with multiple controllers and dynamic worker nodes

Key Challenges in MonoRepo Transition

1. **Change Detection**: Identifying which services need rebuilding
2. **Parallel Execution**: Maintaining efficient build parallelization
3. **Security Scanning**: Managing individual Fortify App IDs per service
4. **Artifact Management**: Handling multiple artifacts from single repository
5. **Deployment Orchestration**: Coordinating deployments across multiple services

MonoRepo CICD Implementation Approaches

Approach 1: Path-Based Change Detection with Matrix Builds

Overview

Leverage Jenkins Matrix builds with intelligent path-based change detection to trigger builds only for modified services.

Implementation Strategy

Pipeline Structure:

groovy

```
// Jenkinsfile (Root Level)
@Library('shared-library') _

pipeline {
    agent none

    stages {
        stage('Change Detection') {
            agent { label 'lightweight' }
            steps {
                script {
                    def changedServices = detectChangedServices()
                    env.CHANGED_SERVICES = changedServices.join(',')
                }
            }
        }

        stage('Matrix Build') {
            when {
                expression { env.CHANGED_SERVICES != '' }
            }
            matrix {
                axes {
                    axis {
                        name 'SERVICE'
                        values script {
                            return env.CHANGED_SERVICES.split(',')
                        }
                    }
                }
                stages {
                    stage('Build & Scan') {
                        agent {
                            label 'dynamic-worker'
                        }
                        steps {
                            buildMicroservice(env.SERVICE)
                        }
                    }
                }
            }
        }
    }
}
```

Shared Library Enhancement:

```groovy
```

```
// vars/detectChangedServices.groovy
def call() {
    def changedFiles = sh(
        script: "git diff --name-only HEAD~1 HEAD",
        returnStdout: true
    ).trim().split('\n')

    def serviceDirectories = [:]
    def changedServices = [] as Set

    // Map service directories
    dir('.') {
        def services = sh(
            script: "find . -maxdepth 2 -name 'pom.xml' -o -name 'build.gradle' -o -name 'package.json' | xargs dirname | so
            returnStdout: true
        ).trim().split('\n')

        services.each { service ->
            serviceDirectories[service] = service.replaceAll('^\\.\\/','')
        }
    }

    // Detect changed services
    changedFiles.each { file ->
        serviceDirectories.each { path, serviceName ->
            if (file.startsWith(path)) {
                changedServices.add(serviceName)
            }
        }
    }

    return changedServices.toList()
}

// vars/buildMicroservice.groovy
def call(String serviceName) {
    dir(serviceName) {
        // Auto-detect build type
        def buildTool = detectBuildTool()

        // Build stage
        stage("Build ${serviceName}") {
            buildWithTool(buildTool, serviceName)
        }

        // Parallel scanning
        def scanStages = [:]

        scanStages["SonarQube ${serviceName}"] = {
            sonarScan(serviceName, buildTool)
        }

        scanStages["Fortify ${serviceName}"] = {
            fortifyScan(serviceName, getFortifyAppId(serviceName))
        }

        scanStages["NexusIQ ${serviceName}"] = {
            nexusIQScan(serviceName, buildTool)
        }

        parallel scanStages

        // Deployment
        stage("Deploy ${serviceName}") {
            deployToUDeploy(serviceName)
        }
    }
}
```

Advantages:

- Minimal changes to existing shared libraries
- Efficient resource utilization
- Maintains parallel scanning per service
- Scales well with CloudBees CI controllers

Disadvantages:

- Complex change detection logic
- Potential for false positives in change detection
- Matrix builds can be resource-intensive

Approach 2: Multibranch Pipeline with Service-Specific Triggers

Overview

Create a sophisticated multibranch pipeline that analyzes changes and creates dynamic pipeline stages for affected services.

Implementation Strategy

Repository Structure:

```
monorepo/
├── services/
│   ├── user-service/
│   │   ├── pom.xml
│   │   └── src/
│   ├── order-service/
│   │   ├── build.gradle
│   │   └── src/
│   └── notification-service/
│       ├── package.json
│       └── src/
├── shared/
│   └── common-libraries/
├── config/
│   ├── fortify-app-ids.yml
│   └── service-config.yml
└── Jenkinsfile
```

Configuration Management:

```
yaml
```

```
# config/service-config.yml
services:
  user-service:
    fortifyAppId: "12345"
    buildTool: "maven"
    deploymentProfile: "user-profile"
    sonarProjectKey: "user-service"

  order-service:
    fortifyAppId: "12346"
    buildTool: "gradle"
    deploymentProfile: "order-profile"
    sonarProjectKey: "order-service"

  notification-service:
    fortifyAppId: "12347"
    buildTool: "nodejs"
    deploymentProfile: "notification-profile"
    sonarProjectKey: "notification-service"
```

Enhanced Pipeline:

```
groovy
```

```
@Library('monorepo-shared-library') _

pipeline {
    agent none

    environment {
        CHANGED_SERVICES = ""
        BUILD_SERVICES = ""
    }

    stages {
        stage('Initialize') {
            agent { label 'lightweight' }
            steps {
                script {
                    def analysis = analyzeChanges()
                    env.CHANGED_SERVICES = analysis.changed.join(',')
                    env.BUILD_SERVICES = analysis.buildRequired.join(',')

                    // Update build description
                    currentBuild.description = "Building: ${env.BUILD_SERVICES}"
                }
            }
        }

        stage('Parallel Service Builds') {
            when {
                expression { env.BUILD_SERVICES != '' }
            }
            steps {
                script {
                    def buildStages = [:]
                    def servicesToBuild = env.BUILD_SERVICES.split(',')

                    servicesToBuild.each { service ->
                        buildStages["Build ${service}"] = {
                            buildServicePipeline(service.trim())
                        }
                    }

                    parallel buildStages
                }
            }
        }

        stage('Integration Tests') {
            when {
                expression { env.BUILD_SERVICES != '' }
            }
            agent { label 'integration-test' }
            steps {
                runIntegrationTests(env.BUILD_SERVICES.split(','))
            }
        }
    }

    post {
        always {
            publishTestResults testResultsPattern: '**/target/surefire-reports/*.xml'
            publishHTML([
                allowMissing: false,
                alwaysLinkToLastBuild: true,
                keepAll: true,
                reportDir: 'reports',
                reportFiles: 'index.html',
                reportName: 'MonoRepo Build Report'
            ])
        }
    }
}
```

Advanced Shared Library:

```
groovy
```

```groovy
// vars/analyzeChanges.groovy
def call() {
    def config = readYaml file: 'config/service-config.yml'
    def changedFiles = getChangedFiles()
    def changedServices = [] as Set
    def buildRequired = [] as Set

    // Analyze changed files
    changedFiles.each { file ->
        config.services.each { serviceName, serviceConfig ->
            if (file.startsWith("services/${serviceName}/")) {
                changedServices.add(serviceName)
                buildRequired.add(serviceName)
            }
        }
    }

    // Check for shared library changes
    def sharedChanged = changedFiles.any { it.startsWith('shared/') }
    if (sharedChanged) {
        // If shared code changed, rebuild all services
        buildRequired.addAll(config.services.keySet())
    }

    return [
        changed: changedServices.toList(),
        buildRequired: buildRequired.toList(),
        sharedChanged: sharedChanged
    ]
}

// vars/buildServicePipeline.groovy
def call(String serviceName) {
    def config = readYaml file: 'config/service-config.yml'
    def serviceConfig = config.services[serviceName]

    node('dynamic-worker') {
        try {
            checkout scm

            dir("services/${serviceName}") {
                // Build stage
                stage("Build ${serviceName}") {
```

```
                buildService(serviceConfig.buildTool, serviceName)
            }

            // Parallel security scans
            def scanTasks = [:]

            scanTasks["SonarQube"] = {
                node('sonar-scanner') {
                    checkout scm
                    dir("services/${serviceName}") {
                        sonarQubeAnalysis(serviceName, serviceConfig)
                    }
                }
            }

            scanTasks["Fortify"] = {
                node('fortify-scanner') {
                    checkout scm
                    dir("services/${serviceName}") {
                        fortifyAnalysis(serviceName, serviceConfig.fortifyAppId)
                    }
                }
            }

            scanTasks["NexusIQ"] = {
                node('nexus-scanner') {
                    checkout scm
                    dir("services/${serviceName}") {
                        nexusIQAnalysis(serviceName, serviceConfig.buildTool)
                    }
                }
            }

            stage("Security Scans ${serviceName}") {
                parallel scanTasks
            }

            // Deployment
            stage("Deploy ${serviceName}") {
                deployService(serviceName, serviceConfig)
            }
        }
    } catch (Exception e) {
        currentBuild.result = 'FAILURE'
```

```
            throw e
        }
    }
}
```

Advantages:

- Clean separation of concerns
- Configuration-driven approach
- Better resource management
- Supports complex dependency scenarios

Disadvantages:

- Requires significant refactoring of existing libraries
- More complex initial setup
- Learning curve for development teams

Approach 3: Hybrid Pipeline with Conditional Stages

Overview

Maintain existing pipeline structure while adding MonoRepo capabilities through conditional stage execution.

Implementation Strategy

Pipeline Framework:

```
groovy
```

```
@Library('hybrid-monorepo-library') _

pipeline {
    agent none

    parameters {
        choice(
            name: 'EXECUTION_MODE',
            choices: ['AUTO_DETECT', 'ALL_SERVICES', 'SPECIFIC_SERVICES'],
            description: 'Pipeline execution mode'
        )
        string(
            name: 'SPECIFIC_SERVICES',
            defaultValue: '',
            description: 'Comma-separated list of services (when SPECIFIC_SERVICES mode)'
        )
        booleanParam(
            name: 'FORCE_BUILD_ALL',
            defaultValue: false,
            description: 'Force build all services regardless of changes'
        )
    }

    stages {
        stage('Repository Analysis') {
            agent { label 'analysis-node' }
            steps {
                script {
                    def analyzer = new MonoRepoAnalyzer()
                    def analysisResult = analyzer.analyze(params)

                    env.TARGET_SERVICES = analysisResult.targetServices.join(',')
                    env.EXECUTION_PLAN = analysisResult.executionPlan

                    // Store analysis results
                    writeJSON file: 'analysis-result.json', json: analysisResult
                    stash includes: 'analysis-result.json', name: 'analysis'
                }
            }
        }

        stage('Service Discovery & Validation') {
            agent { label 'lightweight' }
            steps {
                script {
                    validateServiceConfiguration(env.TARGET_SERVICES.split(','))
                }
            }
        }

        stage('Parallel Service Processing') {
            steps {
                script {
                    executeServiceBuilds()
                }
            }
        }

        stage('Cross-Service Integration') {
            when {
                expression {
                    def services = env.TARGET_SERVICES.split(',')
                    return services.length > 1
                }
            }
            agent { label 'integration-node' }
            steps {
                runCrossServiceTests()
            }
        }
    }
}
```

MonoRepo Analyzer Class:

groovy

```groovy
// src/com/company/MonoRepoAnalyzer.groovy
package com.company

class MonoRepoAnalyzer {

    def analyze(params) {
        def result = [
            targetServices: [],
            executionPlan: [:],
            changeAnalysis: [:]
        ]

        switch(params.EXECUTION_MODE) {
            case 'AUTO_DETECT':
                result = autoDetectServices(params.FORCE_BUILD_ALL)
                break
            case 'ALL_SERVICES':
                result = getAllServices()
                break
            case 'SPECIFIC_SERVICES':
                result = getSpecificServices(params.SPECIFIC_SERVICES)
                break
        }

        return result
    }

    private def autoDetectServices(forceAll) {
        if (forceAll) {
            return getAllServices()
        }

        def changedFiles = sh(
            script: "git diff --name-only HEAD~1 HEAD || echo ''",
            returnStdout: true
        ).trim().split('\n').findAll { it.trim() }

        def services = discoverServices()
        def affectedServices = [] as Set

        changedFiles.each { file ->
            services.each { service, path ->
                if (file.startsWith(path)) {
                    affectedServices.add(service)
                }
            }
        }

        // Check for infrastructure changes
        def infraChanged = changedFiles.any {
            it.startsWith('shared/') ||
            it.startsWith('config/') ||
            it == 'Jenkinsfile'
        }

        if (infraChanged && !forceAll) {
            // Infrastructure changes affect all services
            affectedServices.addAll(services.keySet())
        }

        return [
            targetServices: affectedServices.toList(),
            executionPlan: createExecutionPlan(affectedServices.toList()),
            changeAnalysis: [
                changedFiles: changedFiles,
                infraChanged: infraChanged
            ]
        ]
    }

    private def createExecutionPlan(services) {
        def plan = [:]

        services.each { service ->
            plan[service] = [
                buildType: detectBuildType(service),
                fortifyAppId: getFortifyAppId(service),
                dependencies: getServiceDependencies(service),
                deploymentConfig: getDeploymentConfig(service)
            ]
        }

        return plan
    }
}
```

Service Build Orchestrator:

```groovy
```

```
// vars/executeServiceBuilds.groovy
def call() {
    def services = env.TARGET_SERVICES.split(',').findAll { it.trim() }

    if (services.isEmpty()) {
        echo "No services to build"
        return
    }

    def buildGroups = organizeBuildGroups(services)

    buildGroups.each { groupName, groupServices ->
        stage("Build Group: ${groupName}") {
            def parallelBuilds = [:]

            groupServices.each { service ->
                parallelBuilds["${service}"] = {
                    buildServiceWorkflow(service)
                }
            }

            parallel parallelBuilds
        }
    }
}

def buildServiceWorkflow(serviceName) {
    node('dynamic-worker') {
        def stagePrefix = "[${serviceName}]"

        try {
            stage("${stagePrefix} Checkout") {
                checkout scm
                unstash 'analysis'
            }

            stage("${stagePrefix} Build") {
                dir("services/${serviceName}") {
                    def analysisResult = readJSON file: '../analysis-result.json'
                    def serviceConfig = analysisResult.executionPlan[serviceName]

                    buildWithConfig(serviceName, serviceConfig)
                }
        }

        // Parallel security scanning
        def scanJobs = [:]

        scanJobs["${stagePrefix} SonarQube"] = {
            node('sonar-node') {
                checkout scm
                dir("services/${serviceName}") {
                    sonarQubeAnalysis(serviceName)
                }
            }
        }

        scanJobs["${stagePrefix} Fortify"] = {
            node('fortify-node') {
                checkout scm
                unstash 'analysis'
                dir("services/${serviceName}") {
                    def analysisResult = readJSON file: '../analysis-result.json'
                    def appId = analysisResult.executionPlan[serviceName].fortifyAppId
                    fortifyAnalysis(serviceName, appId)
                }
            }
        }

        scanJobs["${stagePrefix} NexusIQ"] = {
            node('nexus-node') {
                checkout scm
                dir("services/${serviceName}") {
                    nexusIQAnalysis(serviceName)
                }
            }
        }

        stage("${stagePrefix} Security Scans") {
            parallel scanJobs
        }

        stage("${stagePrefix} Deploy") {
            deployToEnvironment(serviceName)
        }

    } catch (Exception e) {
            currentBuild.result = 'FAILURE'
            error("Build failed for service: ${serviceName} - ${e.message}")
        }
    }
}
```

Advantages:

- Gradual migration path
- Backward compatibility with existing processes
- Flexible execution modes
- Maintains existing tool integrations

Disadvantages:

- Code complexity increases
- Maintenance overhead
- Potential performance impact

Approach 4: Event-Driven Pipeline with Webhook Integration

Overview

Implement an event-driven architecture using Bitbucket webhooks and CloudBees CI API to trigger selective builds based on changed paths.

Implementation Strategy

Webhook Handler:

```groovy
```

```
// Webhook Pipeline (webhook-handler/Jenkinsfile)
@Library('webhook-monorepo-library') _

pipeline {
  agent { label 'webhook-processor' }

  triggers {
    bitbucketPush()
  }

  stages {
    stage('Process Webhook') {
      steps {
        script {
          def webhookPayload = parseWebhookPayload()
          def affectedServices = analyzeChangedPaths(webhookPayload)

          if (affectedServices.isEmpty()) {
            echo "No services affected by this change"
            return
          }

          // Trigger individual service builds
          triggerServiceBuilds(affectedServices, webhookPayload)
        }
      }
    }
  }
}

// vars/triggerServiceBuilds.groovy
def call(affectedServices, webhookData) {
  def buildJobs = [:]

  affectedServices.each { service ->
    buildJobs["Trigger ${service}"] = {
      build job: "monorepo-service-builder",
         parameters: [
           string(name: 'SERVICE_NAME', value: service),
           string(name: 'COMMIT_SHA', value: webhookData.commitSha),
           string(name: 'BRANCH_NAME', value: webhookData.branchName),
           booleanParam(name: 'TRIGGERED_BY_WEBHOOK', value: true)
         ],
                  wait: false
        }
    }

    parallel buildJobs
}
```

Service Builder Pipeline:

```groovy
```

```
// Service Builder (monorepo-service-builder/Jenkinsfile)
@Library('monorepo-shared-library') _

pipeline {
    agent none

    parameters {
        string(name: 'SERVICE_NAME', description: 'Name of the service to build')
        string(name: 'COMMIT_SHA', description: 'Commit SHA to build')
        string(name: 'BRANCH_NAME', description: 'Branch name')
        booleanParam(name: 'TRIGGERED_BY_WEBHOOK', defaultValue: false)
    }

    environment {
        SERVICE_PATH = "services/${params.SERVICE_NAME}"
        BUILD_NUMBER_SUFFIX = "${params.COMMIT_SHA.take(8)}"
    }

    stages {
        stage('Validate Service') {
            agent { label 'lightweight' }
            steps {
                script {
                    validateServiceExists(params.SERVICE_NAME)
                }
            }
        }

        stage('Build Service') {
            agent {
                label 'dynamic-worker'
                customWorkspace "workspace/monorepo-${params.SERVICE_NAME}-${env.BUILD_NUMBER}"
            }
            steps {
                checkoutAtCommit(params.COMMIT_SHA, params.BRANCH_NAME)

                dir(env.SERVICE_PATH) {
                    buildService(params.SERVICE_NAME)
                }
            }
        }

        stage('Security Scans') {
```

```
        parallel {
          stage('SonarQube') {
            agent {
              label 'sonar-scanner'
              customWorkspace "workspace/sonar-${params.SERVICE_NAME}-${env.BUILD_NUMBER}"
            }
            steps {
              checkoutAtCommit(params.COMMIT_SHA, params.BRANCH_NAME)
              dir(env.SERVICE_PATH) {
                sonarAnalysis(params.SERVICE_NAME)
              }
            }
          }

          stage('Fortify') {
            agent {
              label 'fortify-scanner'
              customWorkspace "workspace/fortify-${params.SERVICE_NAME}-${env.BUILD_NUMBER}"
            }
            steps {
              checkoutAtCommit(params.COMMIT_SHA, params.BRANCH_NAME)
              dir(env.SERVICE_PATH) {
                fortifyAnalysis(params.SERVICE_NAME)
              }
            }
          }

          stage('NexusIQ') {
            agent {
              label 'nexus-scanner'
              customWorkspace "workspace/nexus-${params.SERVICE_NAME}-${env.BUILD_NUMBER}"
            }
            steps {
              checkoutAtCommit(params.COMMIT_SHA, params.BRANCH_NAME)
              dir(env.SERVICE_PATH) {
                nexusAnalysis(params.SERVICE_NAME)
              }
            }
          }
        }
      }

      stage('Deploy') {
        agent { label 'deployment-agent' }
```

```
            steps {
                checkoutAtCommit(params.COMMIT_SHA, params.BRANCH_NAME)
                dir(env.SERVICE_PATH) {
                    deployService(params.SERVICE_NAME)
                }
            }
        }
    }
}
```

Advantages:

- True event-driven architecture
- Immediate response to changes
- Optimal resource utilization
- Clear separation of concerns

Disadvantages:

- Complex setup and maintenance
- Requires webhook infrastructure
- Debugging can be challenging

Migration Strategy & Implementation Plan

Phase 1: Preparation (Weeks 1-2)

1. **Repository Consolidation**
    - Create monorepo structure
    - Migrate existing repositories
    - Set up shared configuration
2. **Shared Library Enhancement**
    - Extend existing libraries for monorepo support
    - Add change detection capabilities
    - Implement service discovery
3. **Testing Environment Setup**
    - Configure test CloudBees CI controller
    - Set up dynamic worker nodes

- Test basic monorepo functionality

Phase 2: Pilot Implementation (Weeks 3-4)

1. **Select Pilot Services**
   - Choose 2-3 non-critical services
   - Implement chosen approach
   - Run parallel builds (old vs new)
2. **Validation & Tuning**
   - Performance testing
   - Security scan validation
   - Deployment verification

Phase 3: Gradual Rollout (Weeks 5-8)

1. **Batch Migration**
   - Migrate services in groups of 5-10
   - Monitor performance and stability
   - Gather feedback from development teams
2. **Documentation & Training**
   - Create developer guidelines
   - Conduct training sessions
   - Update operational procedures

Phase 4: Full Migration (Weeks 9-12)

1. **Complete Migration**
   - Migrate remaining services
   - Decommission old pipelines
   - Performance optimization
2. **Monitoring & Support**
   - Set up monitoring dashboards
   - Establish support procedures
   - Create troubleshooting guides

Best Practices & Recommendations

Repository Organization

```
monorepo/
├── services/           # Individual microservices
│   ├── service-a/
│   ├── service-b/
│   └── service-c/
├── shared/             # Shared libraries and utilities
│   ├── common/
│   ├── testing/
│   └── infrastructure/
├── config/             # Configuration files
│   ├── service-registry.yml
│   ├── fortify-mappings.yml
│   └── deployment-configs/
├── scripts/            # Build and deployment scripts
├── .jenkins/           # Jenkins-specific configurations
└── docs/               # Documentation
```

Configuration Management

1. **Centralized Configuration**
   - Use YAML files for service metadata
   - Version control all configurations
   - Environment-specific overrides
2. **Security Configuration**
   - Secure storage of Fortify App IDs
   - Encrypted secrets management
   - Access control policies

Performance Optimization

1. **Build Caching**
   - Implement artifact caching
   - Use Docker layer caching
   - Leverage Nexus for dependency caching
2. **Resource Management**

  - Right-size worker nodes
  - Implement build queuing strategies
  - Monitor resource utilization

Monitoring & Observability

1. **Build Metrics**
   - Build duration tracking
   - Success/failure rates
   - Resource utilization
2. **Alerting**
   - Failed build notifications
   - Performance degradation alerts
   - Security scan failures

Tool-Specific Considerations

CloudBees CI

- **Controller Distribution**: Distribute monorepo builds across multiple controllers
- **Dynamic Workers**: Configure appropriate worker templates for different workloads
- **Pipeline Optimization**: Use pipeline caching and parallel execution

Bitbucket Integration

- **Webhook Configuration**: Set up path-based webhooks
- **Branch Policies**: Configure merge requirements
- **Permission Management**: Service-specific access controls

Security Tools Integration

- **SonarQube**: Project keys mapping for individual services
- **Fortify**: App ID management and result aggregation
- **NexusIQ**: Component analysis per service

Deployment (uDeploy)

- **Application Mapping**: Service to application mapping
- **Environment Management**: Coordinate multi-service deployments

- **Rollback Strategies**: Service-specific rollback capabilities

Risk Mitigation

Technical Risks

1. **Build Performance**: Implement incremental builds and caching
2. **Resource Contention**: Monitor and scale infrastructure
3. **Dependency Conflicts**: Use dependency management tools

Operational Risks

1. **Team Adoption**: Provide training and support
2. **Process Changes**: Gradual migration approach
3. **Rollback Plan**: Maintain parallel systems during transition

Security Risks

1. **Access Control**: Implement fine-grained permissions
2. **Audit Trail**: Maintain comprehensive logging
3. **Compliance**: Ensure regulatory requirements are met

Conclusion

The recommended approach is **Approach 2: Multibranch Pipeline with Service-Specific Triggers** for the following reasons:

1. **Scalability**: Handles complex scenarios with multiple services
2. **Maintainability**: Clean, configuration-driven approach
3. **Performance**: Optimal resource utilization
4. **Flexibility**: Supports various build and deployment patterns
5. **Integration**: Works well with existing CloudBees CI infrastructure

This approach provides the best balance of functionality, maintainability, and performance while minimizing risks during the migration process.