Jenkins CI/CD Pipeline Setup Guide

Leap Cloud Deployment 2025 - Day 3

Overview

This guide provides two deployment approaches:

1. **Option A:** Local Jenkins installation on RVC
2. **Option B:** Jenkins on Kubernetes (Recommended for teams)

Both options support:

- ✅ Multi-tenant JFrog Artifactory integration (5 endpoints)
- ✅ CI Pipeline: Build and push Docker images
- ✅ CD Pipeline: Deploy to Docker Compose and Kubernetes (EKS)
- ✅ AWS account authentication per student
- ✅ Generic reusable pipelines

Table of Contents

Prerequisites

Required Information

Each student needs:

- **Corp ID:** `a######`
- **JFrog Endpoint:** One of 5 available endpoints
  - `leapfse1.jfrog.io`
  - `leapfse2.jfrog.io`
  - `leapfse3.jfrog.io`
  - `leapfse4.jfrog.io`
  - `leapfse5.jfrog.io`
- **JFrog Credentials:** `leapfse#` / `fse#Deploy@Cloud`
- **AWS Account:** Individual account alias `fse1team1`

- **AWS Credentials:** Access Key ID and Secret Access Key
- **RDS Endpoint:** From Day 1
- **Docker Images:** Built and pushed to JFrog (Day 2)

Software Requirements

bash

```bash
# Verify installations
docker --version
docker-compose --version
kubectl version --client
aws --version
git --version
```

Option A: Local Jenkins Setup

Step 1: Install Jenkins on RVC

bash

```bash
# Install Java 17
sudo yum install -y java-17-openjdk java-17-openjdk-devel

# Verify Java installation
java -version

# Add Jenkins repository
sudo wget -O /etc/yum.repos.d/jenkins.repo \
    https://pkg.jenkins.io/redhat-stable/jenkins.repo

sudo rpm --import https://pkg.jenkins.io/redhat-stable/jenkins.io-2023.key

# Install Jenkins
sudo yum install -y jenkins

# Start Jenkins
sudo systemctl start jenkins
sudo systemctl enable jenkins

# Check Jenkins status
sudo systemctl status jenkins
```

Step 2: Initial Jenkins Configuration

bash

```bash
# Get initial admin password
sudo cat /var/lib/jenkins/secrets/initialAdminPassword

# Jenkins will be available at:
# http://localhost:8080
```

Browser Setup:

1. Open `http://localhost:8080`
2. Enter initial admin password
3. Install suggested plugins
4. Create admin user
5. Configure Jenkins URL

Step 3: Install Required Plugins

Navigate to: **Manage Jenkins → Plugins → Available Plugins**

Install:

- ✅ Docker Pipeline
- ✅ Kubernetes CLI
- ✅ AWS Steps
- ✅ Credentials Binding
- ✅ Git Plugin
- ✅ Pipeline
- ✅ SSH Agent
- ✅ Environment Injector

Step 4: Configure Docker Access

bash

```bash
# Add jenkins user to docker group
sudo usermod -aG docker jenkins

# Restart Jenkins
sudo systemctl restart jenkins

# Verify docker access
sudo -u jenkins docker ps
```

Option B: Kubernetes Jenkins Setup

Step 1: Deploy Jenkins to Kubernetes

bash

```bash
# Create jenkins namespace and deploy
kubectl apply -f jenkins-complete-deployment.yaml

# Wait for Jenkins to be ready
kubectl wait --for=condition=ready pod -l app=jenkins -n jenkins --timeout=300s

# Get Jenkins pod name
kubectl get pods -n jenkins
```

Step 2: Access Jenkins

Method 1: Port Forward (Local Access)

bash

```bash
kubectl port-forward -n jenkins svc/jenkins 9090:9090
```

Access at: `http://localhost:9090`

Method 2: Ingress (AWS ALB)

bash

```bash
# Get ALB DNS name
kubectl get ingress jenkins-ingress -n jenkins

# Access Jenkins via ALB URL
# http://<alb-dns-name>
```

Step 3: Get Initial Admin Password

bash

```bash
# For Kubernetes deployment
kubectl exec -n jenkins $(kubectl get pods -n jenkins -l app=jenkins -o jsonpath='{.items[0].metadata.name}') -- cat /var/je
```

Step 4: Configure Jenkins on Kubernetes

1. Access Jenkins UI
2. Enter initial admin password
3. Install suggested plugins
4. Create admin user

JFrog Configuration

Step 1: Create JFrog Credentials in Jenkins

Navigate to: **Manage Jenkins → Credentials → System → Global credentials → Add Credentials**

Create **5 credential sets** (one for each JFrog endpoint):

Credential ID	Type	Username	Password	Description
jfrog-1	Username with password	leapfse1	fse1Deploy@Cloud	JFrog Endpoint 1
jfrog-2	Username with password	leapfse2	fse2Deploy@Cloud	JFrog Endpoint 2
jfrog-3	Username with password	leapfse3	fse3Deploy@Cloud	JFrog Endpoint 3
jfrog-4	Username with password	leapfse4	fse4Deploy@Cloud	JFrog Endpoint 4
jfrog-5	Username with password	leapfse5	fse5Deploy@Cloud	JFrog Endpoint 5

Step 2: Create JFrog Endpoint Mapping ConfigMap

yaml

```yaml
# jfrog-config.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: jfrog-endpoints
  namespace: jenkins
data:
  endpoints: |
    {
      "jfrog-1": "leapfse1.jfrog.io",
      "jfrog-2": "leapfse2.jfrog.io",
      "jfrog-3": "leapfse3.jfrog.io",
      "jfrog-4": "leapfse4.jfrog.io",
      "jfrog-5": "leapfse5.jfrog.io"
    }
```

bash

```bash
kubectl apply -f jfrog-config.yaml
```

AWS Credentials Setup

Step 1: Create AWS Credentials

Navigate to: **Manage Jenkins → Credentials → System → Global credentials → Add Credentials**

For Each Student (or create generic):

```
      Field                                  Value
Kind              AWS Credentials
ID                aws-creds-a###### (student-specific) or aws-creds-generic
Access Key ID     [Student's AWS Access Key]
Secret Access Key [Student's AWS Secret Key]
Description       AWS Credentials for a######
```

Step 2: Configure AWS CLI (For Local Jenkins)

bash

```bash
# Configure AWS CLI as jenkins user
sudo su - jenkins
aws configure

# Verify AWS access
aws sts get-caller-identity
aws eks list-clusters --region ap-south-1
```

CI Pipeline: Docker Build

Generic CI Pipeline Template

Create new Pipeline: **New Item → Pipeline → OK**

Pipeline Configuration:

groovy

```
pipeline {
    agent any

    parameters {
        // Student-specific parameters
        string(name: 'CORP_ID', defaultValue: 'a000000', description: 'Your Corp ID (e.g., a643580)')
        choice(name: 'JFROG_ENDPOINT', choices: ['jfrog-1', 'jfrog-2', 'jfrog-3', 'jfrog-4', 'jfrog-5'], description: 'Select your
        string(name: 'JFROG_REPO', defaultValue: 'fse1team1', description: 'Your JFrog repository name')
        string(name: 'GIT_REPO', defaultValue: 'https://github.com/user/repo.git', description: 'Git repository URL')
        string(name: 'GIT_BRANCH', defaultValue: 'feature/docker', description: 'Git branch to build')
        string(name: 'IMAGE_VERSION', defaultValue: '1.0', description: 'Docker image version')
        booleanParam(name: 'BUILD_FRONTEND', defaultValue: true, description: 'Build Frontend?')
        booleanParam(name: 'BUILD_MIDTIER', defaultValue: true, description: 'Build Midtier?')
        booleanParam(name: 'BUILD_BACKEND', defaultValue: true, description: 'Build Backend?')
        booleanParam(name: 'BUILD_FMTS', defaultValue: true, description: 'Build FMTS?')
    }

    environment {
        // JFrog endpoint mapping
        JFROG_URL_1 = 'leapfse1.jfrog.io'
        JFROG_URL_2 = 'leapfse2.jfrog.io'
        JFROG_URL_3 = 'leapfse3.jfrog.io'
        JFROG_URL_4 = 'leapfse4.jfrog.io'
        JFROG_URL_5 = 'leapfse5.jfrog.io'

        // Dynamically set JFROG_URL based on selection
        JFROG_URL = "${params.JFROG_ENDPOINT == 'jfrog-1' ? env.JFROG_URL_1 :
                      params.JFROG_ENDPOINT == 'jfrog-2' ? env.JFROG_URL_2 :
                      params.JFROG_ENDPOINT == 'jfrog-3' ? env.JFROG_URL_3 :
                      params.JFROG_ENDPOINT == 'jfrog-4' ? env.JFROG_URL_4 :
                      env.JFROG_URL_5}"
    }

    stages {
        stage('Checkout') {
            steps {
                echo "🔄 Checking out code from ${params.GIT_REPO}"
                git branch: "${params.GIT_BRANCH}", url: "${params.GIT_REPO}"
            }
        }
```

```
stage('Docker Login') {
    steps {
        script {
            echo "🔐 Logging into JFrog: ${env.JFROG_URL}"
            withCredentials([usernamePassword(
                credentialsId: "${params.JFROG_ENDPOINT}",
                usernameVariable: 'JFROG_USER',
                passwordVariable: 'JFROG_PASS'
            )]) {
                sh """
                    kubectl create secret docker-registry jfrog-secret \\
                        --docker-server=${env.JFROG_URL} \\
                        --docker-username=${JFROG_USER} \\
                        --docker-password=${JFROG_PASS} \\
                        --namespace=${params.NAMESPACE} \\
                        --dry-run=client -o yaml | kubectl apply -f -
                """
            }
        }
    }
}

stage('Create Database Secret') {
    when {
        expression { params.ACTION == 'deploy' }
    }
    steps {
        script {
            sh """
                kubectl create secret generic db-credentials \\
                    --from-literal=username=${params.DB_USERNAME} \\
                    --from-literal=password=${params.DB_PASSWORD} \\
                    --from-literal=endpoint=${params.RDS_ENDPOINT} \\
                    --namespace=${params.NAMESPACE} \\
                    --dry-run=client -o yaml | kubectl apply -f -
            """
        }
    }
}

stage('Generate Kubernetes Manifests') {
```

```
        when {
            expression { params.ACTION == 'deploy' }
        }
        steps {
            script {
                echo "📝 Generating Kubernetes manifests..."

                // Backend Deployment
                writeFile file: 'backend-deployment.yaml', text: """
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ${params.CORP_ID}-backend
  namespace: ${params.NAMESPACE}
  labels:
    app: backend
    corp-id: ${params.CORP_ID}
spec:
  replicas: 2
  selector:
    matchLabels:
      app: backend
      corp-id: ${params.CORP_ID}
  template:
    metadata:
      labels:
        app: backend
        corp-id: ${params.CORP_ID}
    spec:
      imagePullSecrets:
      - name: jfrog-secret
      containers:
      - name: backend
        image: ${env.JFROG_URL}/${params.JFROG_REPO}/${params.CORP_ID}-backend:${params.IMAGE_VERSIO
        ports:
        - containerPort: 8080
        env:
        - name: SPRING_DATASOURCE_URL
          value: "jdbc:oracle:thin:@\$(DB_ENDPOINT):1521/ORCL"
        - name: SPRING_DATASOURCE_USERNAME
          valueFrom:
```

```
          secretKeyRef:
            name: db-credentials
            key: username
        - name: SPRING_DATASOURCE_PASSWORD
          valueFrom:
            secretKeyRef:
              name: db-credentials
              key: password
        - name: DB_ENDPOINT
          valueFrom:
            secretKeyRef:
              name: db-credentials
              key: endpoint
        - name: SPRING_DATASOURCE_DRIVER_CLASS_NAME
          value: "oracle.jdbc.OracleDriver"
        - name: SPRING_JPA_DATABASE_PLATFORM
          value: "org.hibernate.dialect.Oracle12cDialect"
        resources:
          requests:
            memory: "512Mi"
            cpu: "250m"
          limits:
            memory: "1Gi"
            cpu: "500m"
        livenessProbe:
          httpGet:
            path: /actuator/health
            port: 8080
          initialDelaySeconds: 60
          periodSeconds: 10
        readinessProbe:
          httpGet:
            path: /actuator/health
            port: 8080
          initialDelaySeconds: 30
          periodSeconds: 5
---
apiVersion: v1
kind: Service
metadata:
  name: ${params.CORP_ID}-backend
```

```
  namespace: ${params.NAMESPACE}
spec:
  type: ClusterIP
  ports:
  - port: 8080
    targetPort: 8080
  selector:
    app: backend
    corp-id: ${params.CORP_ID}
"""

                // Midtier Deployment
                writeFile file: 'midtier-deployment.yaml', text: """
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ${params.CORP_ID}-midtier
  namespace: ${params.NAMESPACE}
  labels:
    app: midtier
    corp-id: ${params.CORP_ID}
spec:
  replicas: 2
  selector:
    matchLabels:
      app: midtier
      corp-id: ${params.CORP_ID}
  template:
    metadata:
      labels:
        app: midtier
        corp-id: ${params.CORP_ID}
    spec:
      imagePullSecrets:
      - name: jfrog-secret
      containers:
      - name: midtier
        image: ${env.JFROG_URL}/${params.JFROG_REPO}/${params.CORP_ID}-midtier:${params.IMAGE_VERSION
        ports:
        - containerPort: 3000
        env:
```

```
        - name: NODE_ENV
          value: "production"
        - name: BACKEND_URL
          value: "http://${params.CORP_ID}-backend:8080"
        - name: FMTS_URL
          value: "http://${params.CORP_ID}-fmts:5000"
        resources:
          requests:
            memory: "256Mi"
            cpu: "200m"
          limits:
            memory: "512Mi"
            cpu: "400m"
        livenessProbe:
          httpGet:
            path: /health
            port: 3000
          initialDelaySeconds: 30
          periodSeconds: 10
        readinessProbe:
          httpGet:
            path: /health
            port: 3000
          initialDelaySeconds: 15
          periodSeconds: 5
---
apiVersion: v1
kind: Service
metadata:
  name: ${params.CORP_ID}-midtier
  namespace: ${params.NAMESPACE}
spec:
  type: ClusterIP
  ports:
  - port: 3000
    targetPort: 3000
  selector:
    app: midtier
    corp-id: ${params.CORP_ID}
"""
```

```
                // Frontend Deployment
                writeFile file: 'frontend-deployment.yaml', text: """
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ${params.CORP_ID}-frontend
  namespace: ${params.NAMESPACE}
  labels:
    app: frontend
    corp-id: ${params.CORP_ID}
spec:
  replicas: 2
  selector:
    matchLabels:
      app: frontend
      corp-id: ${params.CORP_ID}
  template:
    metadata:
      labels:
        app: frontend
        corp-id: ${params.CORP_ID}
    spec:
      imagePullSecrets:
      - name: jfrog-secret
      containers:
      - name: frontend
        image: ${env.JFROG_URL}/${params.JFROG_REPO}/${params.CORP_ID}-frontend:${params.IMAGE_VERSIC
        ports:
        - containerPort: 4200
        resources:
          requests:
            memory: "128Mi"
            cpu: "100m"
          limits:
            memory: "256Mi"
            cpu: "200m"
        livenessProbe:
          httpGet:
            path: /
            port: 4200
          initialDelaySeconds: 30
```

```
          periodSeconds: 10
        readinessProbe:
          httpGet:
            path: /
            port: 4200
          initialDelaySeconds: 15
          periodSeconds: 5
---
apiVersion: v1
kind: Service
metadata:
  name: ${params.CORP_ID}-frontend
  namespace: ${params.NAMESPACE}
spec:
  type: LoadBalancer
  ports:
  - port: 80
    targetPort: 4200
  selector:
    app: frontend
    corp-id: ${params.CORP_ID}
"""

                // FMTS Deployment
                writeFile file: 'fmts-deployment.yaml', text: """
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ${params.CORP_ID}-fmts
  namespace: ${params.NAMESPACE}
  labels:
    app: fmts
    corp-id: ${params.CORP_ID}
spec:
  replicas: 1
  selector:
    matchLabels:
      app: fmts
      corp-id: ${params.CORP_ID}
  template:
    metadata:
```

```
      labels:
        app: fmts
        corp-id: ${params.CORP_ID}
    spec:
      imagePullSecrets:
      - name: jfrog-secret
      containers:
      - name: fmts
        image: ${env.JFROG_URL}/${params.JFROG_REPO}/${params.CORP_ID}-fmts:${params.IMAGE_VERSION}
        ports:
        - containerPort: 5000
        env:
        - name: NODE_ENV
          value: "production"
        volumeMounts:
        - name: uploads
          mountPath: /app/uploads
        resources:
          requests:
            memory: "256Mi"
            cpu: "200m"
          limits:
            memory: "512Mi"
            cpu: "400m"
      volumes:
      - name: uploads
        persistentVolumeClaim:
          claimName: ${params.CORP_ID}-fmts-pvc
---
apiVersion: v1
kind: Service
metadata:
  name: ${params.CORP_ID}-fmts
  namespace: ${params.NAMESPACE}
spec:
  type: ClusterIP
  ports:
  - port: 5000
    targetPort: 5000
  selector:
    app: fmts
```

```
    corp-id: ${params.CORP_ID}
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: ${params.CORP_ID}-fmts-pvc
  namespace: ${params.NAMESPACE}
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 5Gi
  storageClassName: gp2
"""

                // Ingress
                writeFile file: 'ingress.yaml', text: """
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ${params.CORP_ID}-ingress
  namespace: ${params.NAMESPACE}
  annotations:
    alb.ingress.kubernetes.io/scheme: internet-facing
    alb.ingress.kubernetes.io/target-type: ip
    alb.ingress.kubernetes.io/healthcheck-path: /
spec:
  ingressClassName: alb
  rules:
  - http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: ${params.CORP_ID}-frontend
            port:
              number: 80
      - path: /api
        pathType: Prefix
```

```
    backend:
      service:
        name: ${params.CORP_ID}-midtier
        port:
          number: 3000
"""
            }
        }
    }

    stage('Deploy to Kubernetes') {
        when {
            expression { params.ACTION == 'deploy' }
        }
        steps {
            script {
                withCredentials([[$class: 'AmazonWebServicesCredentialsBinding',
                                  credentialsId: "${params.AWS_CREDS}"]]) {
                    sh """
                        echo "🚀 Deploying to Kubernetes..."

                        # Apply manifests
                        kubectl apply -f backend-deployment.yaml
                        kubectl apply -f midtier-deployment.yaml
                        kubectl apply -f fmts-deployment.yaml
                        kubectl apply -f frontend-deployment.yaml
                        kubectl apply -f ingress.yaml

                        echo "⏳ Waiting for deployments to be ready..."
                        kubectl wait --for=condition=available --timeout=300s \\
                            deployment/${params.CORP_ID}-backend -n ${params.NAMESPACE}
                        kubectl wait --for=condition=available --timeout=300s \\
                            deployment/${params.CORP_ID}-midtier -n ${params.NAMESPACE}
                        kubectl wait --for=condition=available --timeout=300s \\
                            deployment/${params.CORP_ID}-fmts -n ${params.NAMESPACE}
                        kubectl wait --for=condition=available --timeout=300s \\
                            deployment/${params.CORP_ID}-frontend -n ${params.NAMESPACE}

                        echo "✅ Deployment complete!"
                    """
                }
```

```
        }
    }
}

stage('Get Status') {
    when {
        expression { params.ACTION == 'status' || params.ACTION == 'deploy' }
    }
    steps {
        script {
            withCredentials([[$class: 'AmazonWebServicesCredentialsBinding',
                              credentialsId: "${params.AWS_CREDS}"]]) {
                sh """
                    echo "📊 Deployment Status:"
                    echo "====================="

                    kubectl get deployments -n ${params.NAMESPACE} -l corp-id=${params.CORP_ID}
                    echo ""

                    kubectl get pods -n ${params.NAMESPACE} -l corp-id=${params.CORP_ID}
                    echo ""

                    kubectl get services -n ${params.NAMESPACE} -l corp-id=${params.CORP_ID}
                    echo ""

                    echo "🌐 Ingress Information:"
                    kubectl get ingress ${params.CORP_ID}-ingress -n ${params.NAMESPACE}
                    echo ""

                    echo "🔗 Application URL:"
                    kubectl get ingress ${params.CORP_ID}-ingress -n ${params.NAMESPACE} \\
                        -o jsonpath='{.status.loadBalancer.ingress[0].hostname}'
                    echo ""
                """
            }
        }
    }
}

stage('Delete Deployment') {
    when {
```

```
        expression { params.ACTION == 'delete' }
    }
    steps {
        script {
            withCredentials([[$class: 'AmazonWebServicesCredentialsBinding',
                              credentialsId: "${params.AWS_CREDS}"]]) {
                sh """
                    echo "🗑️ Deleting deployment..."

                    kubectl delete deployment -n ${params.NAMESPACE} -l corp-id=${params.CORP_ID}
                    kubectl delete service -n ${params.NAMESPACE} -l corp-id=${params.CORP_ID}
                    kubectl delete ingress ${params.CORP_ID}-ingress -n ${params.NAMESPACE} || true
                    kubectl delete pvc ${params.CORP_ID}-fmts-pvc -n ${params.NAMESPACE} || true

                    echo "✅ Resources deleted!"
                """
            }
        }
    }
}

stage('Rollback') {
    when {
        expression { params.ACTION == 'rollback' }
    }
    steps {
        script {
            withCredentials([[$class: 'AmazonWebServicesCredentialsBinding',
                              credentialsId: "${params.AWS_CREDS}"]]) {
                sh """
                    echo "⏪ Rolling back deployments..."

                    kubectl rollout undo deployment/${params.CORP_ID}-backend -n ${params.NAMESPACE}
                    kubectl rollout undo deployment/${params.CORP_ID}-midtier -n ${params.NAMESPACE}
                    kubectl rollout undo deployment/${params.CORP_ID}-fmts -n ${params.NAMESPACE}
                    kubectl rollout undo deployment/${params.CORP_ID}-frontend -n ${params.NAMESPACE}

                    echo "✅ Rollback complete!"
                """
            }
        }
```

```
        }
      }
    }

    post {
      success {
        script {
          if (params.ACTION == 'deploy') {
            echo """
            ✅ ========================================
            ✅ KUBERNETES DEPLOYMENT SUCCESSFUL!
            ✅ ========================================
            📦 Namespace: ${params.NAMESPACE}
            🎯 Corp ID: ${params.CORP_ID}
            🏷️  Version: ${params.IMAGE_VERSION}

            Get your application URL:
            kubectl get ingress ${params.CORP_ID}-ingress -n ${params.NAMESPACE}
            ✅ ========================================
            """
          } else {
            echo "✅ Action '${params.ACTION}' completed successfully!"
          }
        }
      }
      failure {
        echo """
        ❌ ========================================
        ❌ DEPLOYMENT FAILED!
        ❌ ========================================
        Check the logs above for errors.

        Debug commands:
        kubectl get pods -n ${params.NAMESPACE}
        kubectl logs -n ${params.NAMESPACE} -l corp-id=${params.CORP_ID}
        ❌ ========================================
        """
      }
    }
}
```

Generic Pipeline Templates

Multi-Student CI/CD Pipeline

Create a single pipeline that all students can use:

groovy

```
pipeline {
  agent any

  parameters {
    // Student Identity
    string(name: 'CORP_ID', defaultValue: 'a000000', description: '🆔 Your Corp ID (e.g., a643580)')

    // JFrog Configuration
    choice(name: 'JFROG_ENDPOINT',
        choices: ['jfrog-1', 'jfrog-2', 'jfrog-3', 'jfrog-4', 'jfrog-5'],
        description: '🏢 Your assigned JFrog endpoint')
    string(name: 'JFROG_REPO', defaultValue: 'fse1team1', description: '📦 JFrog repository name')

    // Git Configuration
    string(name: 'GIT_REPO', description: '📂 Git repository URL')
    string(name: 'GIT_BRANCH', defaultValue: 'feature/docker', description: '🌿 Branch to build')

    // Build Configuration
    string(name: 'IMAGE_VERSION', defaultValue: '1.0', description: '🏷️ Image version tag')
    booleanParam(name: 'BUILD_ALL', defaultValue: true, description: '🏗️ Build all components?')

    // Deployment Target
    choice(name: 'DEPLOY_TARGET',
        choices: ['none', 'docker-compose', 'kubernetes'],
        description: '🎯 Where to deploy after build?')

    // Database Configuration
    string(name: 'RDS_ENDPOINT', description: '🗄️ RDS endpoint from Day 1')
    string(name: 'DB_USERNAME', defaultValue: 'admin', description: '👤 Database username')
    password(name: 'DB_PASSWORD', defaultValue: 'LA2025fmr', description: '🔐 Database password')

    // Kubernetes Configuration (if deploying to K8s)
    string(name: 'K8S_NAMESPACE', defaultValue: 'default', description: '📦 Kubernetes namespace')
    string(name: 'EKS_CLUSTER', defaultValue: 'my-eks-cluster', description: '☸️ EKS cluster name')
    choice(name: 'AWS_CREDS',
        choices: ['aws-creds-generic'],
        description: '🔑 AWS credentials')
  }

  environment {
    // JFrog URL mapping
```

```
    JFROG_URLS = [
        'jfrog-1': 'leapfse1.jfrog.io',
        'jfrog-2': 'leapfse2.jfrog.io',
        'jfrog-3': 'leapfse3.jfrog.io',
        'jfrog-4': 'leapfse4.jfrog.io',
        'jfrog-5': 'leapfse5.jfrog.io'
    ]
    JFROG_URL = "${JFROG_URLS[params.JFROG_ENDPOINT]}"
}

stages {
    stage('🎬 Initialize') {
        steps {
            script {
                echo """
                ╔═══════════════════════════════════════╗
                ║  🚀 LEAP CI/CD PIPELINE STARTED       ║
                ╚═══════════════════════════════════════╝

                👤 Corp ID: ${params.CORP_ID}
                🏢 JFrog: ${env.JFROG_URL}
                📦 Repo: ${params.JFROG_REPO}
                🏷️  Version: ${params.IMAGE_VERSION}
                🎯 Deploy: ${params.DEPLOY_TARGET}
                """
            }
        }
    }

    stage('📥 Checkout Code') {
        steps {
            git branch: "${params.GIT_BRANCH}", url: "${params.GIT_REPO}"
        }
    }

    stage('🏗️  Build & Push Images') {
        steps {
            script {
                withCredentials([usernamePassword(
                    credentialsId: "${params.JFROG_ENDPOINT}",
                    usernameVariable: 'JFROG_USER',
```

```
            passwordVariable: 'JFROG_PASS'
        )]) {
            sh """
                echo ${JFROG_PASS} | docker login ${env.JFROG_URL} -u ${JFROG_USER} --password-stdin
            """

            def components = ['frontend', 'midtier', 'backend', 'fmts']

            components.each { component ->
                if (params.BUILD_ALL || params."BUILD_${component.toUpperCase()}") {
                    echo "🏗️ Building ${component}..."
                    def imageName = "${params.CORP_ID}-${component}"
                    def fullImage = "${env.JFROG_URL}/${params.JFROG_REPO}/${imageName}:${params.IMAGE_

                    sh """
                        cd ${component}
                        docker build -t ${imageName}:${params.IMAGE_VERSION} .
                        docker tag ${imageName}:${params.IMAGE_VERSION} ${fullImage}
                        docker push ${fullImage}
                        echo "✅ ${component} → ${fullImage}"
                    """
                }
            }
        }
    }
}

stage('🐳 Deploy to Docker Compose') {
    when {
        expression { params.DEPLOY_TARGET == 'docker-compose' }
    }
    steps {
        script {
            // Use Docker Compose deployment logic from previous pipeline
            echo "🚀 Deploying to Docker Compose..."
            // ... (code from Docker Compose pipeline)
        }
    }
}
```

```
        stage('☸️ Deploy to Kubernetes') {
            when {
                expression { params.DEPLOY_TARGET == 'kubernetes' }
            }
            steps {
                script {
                    // Use Kubernetes deployment logic from previous pipeline
                    echo "🚀 Deploying to Kubernetes..."
                    // ... (code from Kubernetes pipeline)
                }
            }
        }
    }

    post {
        success {
            echo """
            ✅ ═══════════════════════════════════════════════════
            ✅   PIPELINE COMPLETED SUCCESSFULLY!
            ✅ ═══════════════════════════════════════════════════
            """
        }
        failure {
            echo """
            ❌ ═══════════════════════════════════════════════════
            ❌   PIPELINE FAILED!
            ❌ ═══════════════════════════════════════════════════
            """
        }
        always {
            sh "docker logout ${env.JFROG_URL} || true"
        }
    }
}
```

Student Quick Start Guide

For Students Using Local Jenkins

1. **Access Jenkins:**
    - Open browser: `http://localhost:8080`

    - Login with your credentials
2. **Select Your Pipeline:**
    - Click on "Generic-CI-CD-Pipeline"
    - Click "Build with Parameters"
3. **Fill in Your Details:**

```
CORP_ID: a643580
JFROG_ENDPOINT: jfrog-1 (your assigned endpoint)
JFROG_REPO: fse1team1
GIT_REPO: https://github.com/yourrepo/a643580-projectName
GIT_BRANCH: feature/docker
IMAGE_VERSION: 1.0
DEPLOY_TARGET: docker-compose (or kubernetes)
RDS_ENDPOINT: a643580-rds.cj6ui28e0bu9.ap-south-1.rds.amazonaws.com
```

4. **Click "Build"** and monitor the console output

For Students Using Kubernetes Jenkins

1. **Access Jenkins:**
    - Via Port Forward: `http://localhost:9090`
    - Or via ALB: `http://<alb-dns-name>`
2. **Same pipeline usage as above**

Troubleshooting

Common Issues

1. JFrog Authentication Failed

Error:

```
Error response from daemon: Get "https://leapfse1.jfrog.io/v2/": unauthorized
```

Solution:

```bash
```

```
# Verify credentials in Jenkins
# Re-create JFrog credentials with correct password
# Ensure credential ID matches pipeline parameter
```

2. AWS Authentication Failed

Error:

```
error: You must be logged in to the server (Unauthorized)
```

Solution:

bash

```bash
# Verify AWS credentials in Jenkins
# Check IAM permissions for EKS access
# Update kubeconfig:
aws eks update-kubeconfig --region ap-south-1 --name my-eks-cluster
```

3. Docker Build Failed

Error:

```
ERROR: Cannot connect to Docker daemon
```

Solution:

bash

```
# For local Jenkins:
sudo systemctl start docker
sudo usermod -aG docker jenkins
sudo systemctl restart jenkins

# For K8s Jenkins:
# Ensure Docker-in-Docker is configured in pod
```

4. RDS Connection Failed

Error:

```
ORA-01017: invalid username/password; logon denied
```

Solution:

- Verify RDS endpoint is correct
- Check database credentials match
- Ensure RDS security group allows EKS security group
- Test connectivity:

bash

```
nc -zv a######-rds.cj6ui28e0bu9.ap-south-1.rds.amazonaws.com 1521
```

5. Kubernetes Deployment Timeout

Error:

```
error: timed out waiting for the condition
```

Solution:

bash

```
# Check pod status
kubectl get pods -n <namespace>

# Check pod logs
kubectl logs <pod-name> -n <namespace>

# Check events
kubectl get events -n <namespace> --sort-by='.lastTimestamp'

# Common causes:
# - Image pull error (check JFrog secret)
# - Resource limits too low
# - Liveness probe failing
```

6. Port Conflicts (Docker Compose)

Error:

```
port is already allocated
```

Solution:

bash

```
# Find conflicting container
docker ps | grep <port>

# Stop it
docker stop <container-id>

# Or change port in docker-compose.yml
ports:
  - "4201:4200"  # Use different host port
```

Best Practices

1. Version Management

groovy

```groovy
// Use semantic versioning
IMAGE_VERSION: "1.0.0"    // Initial release
IMAGE_VERSION: "1.1.0"    // Feature update
IMAGE_VERSION: "1.0.1"    // Bug fix

// Tag with git commit
IMAGE_VERSION: "${GIT_COMMIT.take(7)}"

// Tag with build number
IMAGE_VERSION: "${BUILD_NUMBER}"
```

2. Credentials Security

- ✅ Always use Jenkins credentials store
- ✅ Never hardcode passwords in Jenkinsfile
- ✅ Use AWS Secrets Manager for production
- ✅ Rotate credentials regularly
- ❌ Never commit credentials to Git

3. Pipeline Organization

```
jenkins/
├── pipelines/
│   ├── ci-build.groovy
│   ├── cd-docker-compose.groovy
│   ├── cd-kubernetes.groovy
│   └── generic-pipeline.groovy
├── shared-libraries/
│   ├── vars/
│   │   ├── buildDockerImage.groovy
│   │   ├── deployToK8s.groovy
│   │   └── deployToCompose.groovy
└── config/
    ├── jfrog-endpoints.json
    └── aws-regions.json
```

4. Resource Management

groovy

```groovy
// Clean up after builds
post {
    always {
        // Remove local images
        sh "docker image prune -f"

        // Logout from registries
        sh "docker logout ${JFROG_URL}"

        // Clean workspace
        cleanWs()
    }
}
```

5. Notifications

groovy

```
post {
    success {
        // Send email notification
        emailext(
            to: "${params.CORP_ID}@example.com",
            subject: "✅ Build Successful: ${params.CORP_ID}-${BUILD_NUMBER}",
            body: "Your deployment is ready!"
        )
    }
    failure {
        // Send failure notification
        emailext(
            to: "${params.CORP_ID}@example.com",
            subject: "❌ Build Failed: ${params.CORP_ID}-${BUILD_NUMBER}",
            body: "Check Jenkins console: ${BUILD_URL}"
        )
    }
}
```

Advanced Configurations

Multi-Environment Deployment

groovy

```
parameters {
    choice(name: 'ENVIRONMENT',
           choices: ['dev', 'staging', 'production'],
           description: 'Deployment environment')
}

environment {
    NAMESPACE = "${params.ENVIRONMENT}-${params.CORP_ID}"
    REPLICAS = "${params.ENVIRONMENT == 'production' ? '3' : '1'}"
}
```

Automated Testing

groovy

```
stage('Run Tests') {
    steps {
        script {
            sh """
                # Backend tests
                cd backend
                mvn test

                # Frontend tests
                cparams.JFROG_ENDPOINT}",
                        usernameVariable: 'JFROG_USER',
                        passwordVariable: 'JFROG_PASS'
                    )]) {
                        sh """
                            echo ${JFROG_PASS} | docker login ${env.JFROG_URL} -u ${JFROG_USER} --password-stdin
                        """
                    }
                }
            }
        }

        stage('Build Frontend') {
            when {
                expression { params.BUILD_FRONTEND == true }
            }
            steps {
                script {
                    echo "🏗️ Building Frontend..."
                    def imageName = "${params.CORP_ID}-frontend"
                    def fullImageName = "${env.JFROG_URL}/${params.JFROG_REPO}/${imageName}:${params.IMAGE_V

                    sh """
                        cd frontend
                        docker build -t ${imageName}:${params.IMAGE_VERSION} .
                        docker tag ${imageName}:${params.IMAGE_VERSION} ${fullImageName}
                        docker push ${fullImageName}
                        echo "✅ Frontend pushed: ${fullImageName}"
                    """
                }
            }
        }
```

```
stage('Build Midtier') {
    when {
        expression { params.BUILD_MIDTIER == true }
    }
    steps {
        script {
            echo "🏗️ Building Midtier..."
            def imageName = "${params.CORP_ID}-midtier"
            def fullImageName = "${env.JFROG_URL}/${params.JFROG_REPO}/${imageName}:${params.IMAGE_V

            sh """
                cd midtier
                docker build -t ${imageName}:${params.IMAGE_VERSION} .
                docker tag ${imageName}:${params.IMAGE_VERSION} ${fullImageName}
                docker push ${fullImageName}
                echo "✅ Midtier pushed: ${fullImageName}"
            """
        }
    }
}

stage('Build Backend') {
    when {
        expression { params.BUILD_BACKEND == true }
    }
    steps {
        script {
            echo "🏗️ Building Backend..."
            def imageName = "${params.CORP_ID}-backend"
            def fullImageName = "${env.JFROG_URL}/${params.JFROG_REPO}/${imageName}:${params.IMAGE_V

            sh """
                cd backend
                docker build -t ${imageName}:${params.IMAGE_VERSION} .
                docker tag ${imageName}:${params.IMAGE_VERSION} ${fullImageName}
                docker push ${fullImageName}
                echo "✅ Backend pushed: ${fullImageName}"
            """
        }
    }
```

```
    }

    stage('Build FMTS') {
        when {
            expression { params.BUILD_FMTS == true }
        }
        steps {
            script {
                echo "🏗️ Building FMTS..."
                def imageName = "${params.CORP_ID}-fmts"
                def fullImageName = "${env.JFROG_URL}/${params.JFROG_REPO}/${imageName}:${params.IMAGE_V

                sh """
                    cd fmts
                    docker build -t ${imageName}:${params.IMAGE_VERSION} .
                    docker tag ${imageName}:${params.IMAGE_VERSION} ${fullImageName}
                    docker push ${fullImageName}
                    echo "✅ FMTS pushed: ${fullImageName}"
                """
            }
        }
    }

    stage('Cleanup') {
        steps {
            script {
                echo "🧹 Cleaning up local images..."
                sh """
                    docker image prune -f
                """
            }
        }
    }
}

post {
    success {
        echo """
        ✅ ========================================
        ✅ BUILD SUCCESSFUL!
        ✅ ========================================
```

```
        📦 Images pushed to: ${env.JFROG_URL}/${params.JFROG_REPO}/
        🏷️  Version: ${params.IMAGE_VERSION}
        👤 Corp ID: ${params.CORP_ID}
        ✅ ==========================================
        """
      }
      failure {
        echo """
        ❌ ==========================================
        ❌ BUILD FAILED!
        ❌ ==========================================
        Please check the logs above for errors.
        ❌ ==========================================
        """
      }
      always {
        echo "🔓 Logging out from Docker registries..."
        sh "docker logout ${env.JFROG_URL} || true"
      }
    }
  }
```

Usage Instructions

1. Go to Jenkins Dashboard
2. Click on your CI pipeline
3. Click **Build with Parameters**
4. Fill in your details:
   - **CORP_ID:** Your Corp ID (e.g., `a643580`)
   - **JFROG_ENDPOINT:** Select your assigned endpoint (e.g., `jfrog-1`)
   - **JFROG_REPO:** Your team repository (e.g., `fse1team1`)
   - **GIT_REPO:** Your Git repository URL
   - **GIT_BRANCH:** Branch to build (default: `feature/docker`)
   - **IMAGE_VERSION:** Version tag (e.g., `1.0`, `1.1`)
   - Select which components to build
5. Click **Build**

CD Pipeline: Docker Compose

Generic Docker Compose Deployment Pipeline

groovy

```
pipeline {
    agent any

    parameters {
        string(name: 'CORP_ID', defaultValue: 'a000000', description: 'Your Corp ID')
        choice(name: 'JFROG_ENDPOINT', choices: ['jfrog-1', 'jfrog-2', 'jfrog-3', 'jfrog-4', 'jfrog-5'], description: 'Select your
        string(name: 'JFROG_REPO', defaultValue: 'fse1team1', description: 'Your JFrog repository')
        string(name: 'IMAGE_VERSION', defaultValue: '1.0', description: 'Image version to deploy')
        string(name: 'RDS_ENDPOINT', defaultValue: '', description: 'RDS endpoint (e.g., a######-rds.cj6ui28e0bu9.ap-sou
        string(name: 'DB_USERNAME', defaultValue: 'admin', description: 'Database username')
        password(name: 'DB_PASSWORD', defaultValue: 'LA2025fmr', description: 'Database password')
        string(name: 'DEPLOYMENT_HOST', defaultValue: 'localhost', description: 'Host to deploy (localhost or remote IP)'
        choice(name: 'ACTION', choices: ['deploy', 'stop', 'restart', 'status'], description: 'Deployment action')
    }

    environment {
        JFROG_URL_1 = 'leapfse1.jfrog.io'
        JFROG_URL_2 = 'leapfse2.jfrog.io'
        JFROG_URL_3 = 'leapfse3.jfrog.io'
        JFROG_URL_4 = 'leapfse4.jfrog.io'
        JFROG_URL_5 = 'leapfse5.jfrog.io'

        JFROG_URL = "${params.JFROG_ENDPOINT == 'jfrog-1' ? env.JFROG_URL_1 :
                      params.JFROG_ENDPOINT == 'jfrog-2' ? env.JFROG_URL_2 :
                      params.JFROG_ENDPOINT == 'jfrog-3' ? env.JFROG_URL_3 :
                      params.JFROG_ENDPOINT == 'jfrog-4' ? env.JFROG_URL_4 :
                      env.JFROG_URL_5}"
    }

    stages {
        stage('Prepare') {
            steps {
                script {
                    echo "🎯 Preparing Docker Compose deployment..."
                    echo "Corp ID: ${params.CORP_ID}"
                    echo "JFrog: ${env.JFROG_URL}"
                    echo "Version: ${params.IMAGE_VERSION}"
                    echo "Action: ${params.ACTION}"
                }
            }
        }
```

```
        stage('Docker Login') {
            steps {
                script {
                    withCredentials([usernamePassword(
                        credentialsId: "${params.JFROG_ENDPOINT}",
                        usernameVariable: 'JFROG_USER',
                        passwordVariable: 'JFROG_PASS'
                    )]) {
                        sh """
                            echo ${JFROG_PASS} | docker login ${env.JFROG_URL} -u ${JFROG_USER} --password-stdin
                        """
                    }
                }
            }
        }

        stage('Generate Docker Compose') {
            steps {
                script {
                    echo "📝 Generating docker-compose.yml..."

                    writeFile file: 'docker-compose.yml', text: """
version: '3.8'

services:
  frontend:
    image: ${env.JFROG_URL}/${params.JFROG_REPO}/${params.CORP_ID}-frontend:${params.IMAGE_VERSION
    container_name: ${params.CORP_ID}-frontend
    ports:
      - "4200:4200"
    networks:
      - app-network
    depends_on:
      - midtier
    restart: unless-stopped

  midtier:
    image: ${env.JFROG_URL}/${params.JFROG_REPO}/${params.CORP_ID}-midtier:${params.IMAGE_VERSION}
    container_name: ${params.CORP_ID}-midtier
    ports:
```

```
    - "3000:3000"
  environment:
    - NODE_ENV=production
    - BACKEND_URL=http://backend:8080
    - FMTS_URL=http://fmts:5000
  networks:
    - app-network
  depends_on:
    - backend
    - fmts
  restart: unless-stopped

 backend:
  image: ${env.JFROG_URL}/${params.JFROG_REPO}/${params.CORP_ID}-backend:${params.IMAGE_VERSION
  container_name: ${params.CORP_ID}-backend
  ports:
    - "8080:8080"
  environment:
    - SPRING_DATASOURCE_URL=jdbc:oracle:thin:@${params.RDS_ENDPOINT}:1521/ORCL
    - SPRING_DATASOURCE_USERNAME=${params.DB_USERNAME}
    - SPRING_DATASOURCE_PASSWORD=${params.DB_PASSWORD}
    - SPRING_DATASOURCE_DRIVER_CLASS_NAME=oracle.jdbc.OracleDriver
    - SPRING_JPA_DATABASE_PLATFORM=org.hibernate.dialect.Oracle12cDialect
  networks:
    - app-network
  restart: unless-stopped

 fmts:
  image: ${env.JFROG_URL}/${params.JFROG_REPO}/${params.CORP_ID}-fmts:${params.IMAGE_VERSION}
  container_name: ${params.CORP_ID}-fmts
  ports:
    - "5000:5000"
  environment:
    - NODE_ENV=production
  volumes:
    - fmts-uploads:/app/uploads
  networks:
    - app-network
  restart: unless-stopped

networks:
```

```
  app-network:
    driver: bridge

volumes:
  fmts-uploads:
    driver: local
"""
                sh "cat docker-compose.yml"
            }
        }
    }

    stage('Deploy') {
        when {
            expression { params.ACTION == 'deploy' }
        }
        steps {
            script {
                echo "🚀 Deploying application..."
                sh """
                    docker-compose pull
                    docker-compose up -d
                    sleep 10
                    docker-compose ps
                """
            }
        }
    }

    stage('Stop') {
        when {
            expression { params.ACTION == 'stop' }
        }
        steps {
            script {
                echo "🛑 Stopping application..."
                sh "docker-compose down"
            }
        }
    }
```

```
stage('Restart') {
    when {
        expression { params.ACTION == 'restart' }
    }
    steps {
        script {
            echo "🔄 Restarting application..."
            sh """
                docker-compose restart
                sleep 5
                docker-compose ps
            """
        }
    }
}

stage('Status') {
    when {
        expression { params.ACTION == 'status' }
    }
    steps {
        script {
            echo "📊 Checking application status..."
            sh """
                docker-compose ps
                echo "\\n=== Container Logs ==="
                docker-compose logs --tail=20
            """
        }
    }
}

stage('Health Check') {
    when {
        expression { params.ACTION == 'deploy' || params.ACTION == 'restart' }
    }
    steps {
        script {
            echo "🏥 Running health checks..."
            sh """
                echo "Waiting for services to be healthy..."
```

```
            sleep 15

            echo "✅ Checking Frontend..."
            curl -f http://localhost:4200 || echo "⚠️ Frontend not responding"

            echo "✅ Checking Midtier..."
            curl -f http://localhost:3000/health || echo "⚠️ Midtier not responding"

            echo "✅ Checking Backend..."
            curl -f http://localhost:8080/actuator/health || echo "⚠️ Backend not responding"

            echo "✅ Checking FMTS..."
            curl -f http://localhost:5000/health || echo "⚠️ FMTS not responding"
          """
        }
      }
    }
  }

  post {
    success {
      echo """
      ✅ ========================================
      ✅ DEPLOYMENT SUCCESSFUL!
      ✅ ========================================
      🎯 Action: ${params.ACTION}
      🌐 Frontend: http://localhost:4200
      🔧 Midtier: http://localhost:3000
      📦 Backend: http://localhost:8080
      📁 FMTS: http://localhost:5000
      ✅ ========================================
      """
    }
    failure {
      echo """
      ❌ ========================================
      ❌ DEPLOYMENT FAILED!
      ❌ ========================================
      Check logs: docker-compose logs
      ❌ ========================================
      """
```

```
        }
        always {
            sh "docker logout ${env.JFROG_URL} || true"
        }
    }
}
```

CD Pipeline: Kubernetes/EKS

Generic Kubernetes Deployment Pipeline

groovy

```
pipeline {
    agent any

    parameters {
        string(name: 'CORP_ID', defaultValue: 'a000000', description: 'Your Corp ID')
        choice(name: 'JFROG_ENDPOINT', choices: ['jfrog-1', 'jfrog-2', 'jfrog-3', 'jfrog-4', 'jfrog-5'], description: 'Select your
        string(name: 'JFROG_REPO', defaultValue: 'fse1team1', description: 'Your JFrog repository')
        string(name: 'IMAGE_VERSION', defaultValue: '1.0', description: 'Image version to deploy')
        string(name: 'NAMESPACE', defaultValue: 'default', description: 'Kubernetes namespace')
        string(name: 'AWS_REGION', defaultValue: 'ap-south-1', description: 'AWS Region')
        string(name: 'EKS_CLUSTER_NAME', defaultValue: 'my-eks-cluster', description: 'EKS Cluster Name')
        string(name: 'RDS_ENDPOINT', defaultValue: '', description: 'RDS endpoint')
        string(name: 'DB_USERNAME', defaultValue: 'admin', description: 'Database username')
        password(name: 'DB_PASSWORD', defaultValue: 'LA2025fmr', description: 'Database password')
        choice(name: 'ACTION', choices: ['deploy', 'delete', 'status', 'rollback'], description: 'Deployment action')
        choice(name: 'AWS_CREDS', choices: ['aws-creds-generic', 'aws-creds-a643580'], description: 'AWS Credentials to u
    }

    environment {
        JFROG_URL_1 = 'leapfse1.jfrog.io'
        JFROG_URL_2 = 'leapfse2.jfrog.io'
        JFROG_URL_3 = 'leapfse3.jfrog.io'
        JFROG_URL_4 = 'leapfse4.jfrog.io'
        JFROG_URL_5 = 'leapfse5.jfrog.io'

        JFROG_URL = "${params.JFROG_ENDPOINT == 'jfrog-1' ? env.JFROG_URL_1 :
                      params.JFROG_ENDPOINT == 'jfrog-2' ? env.JFROG_URL_2 :
                      params.JFROG_ENDPOINT == 'jfrog-3' ? env.JFROG_URL_3 :
                      params.JFROG_ENDPOINT == 'jfrog-4' ? env.JFROG_URL_4 :
                      env.JFROG_URL_5}"
    }

    stages {
        stage('Prepare') {
            steps {
                script {
                    echo "🎯 Preparing Kubernetes deployment..."
                    echo "Corp ID: ${params.CORP_ID}"
                    echo "Namespace: ${params.NAMESPACE}"
                    echo "EKS Cluster: ${params.EKS_CLUSTER_NAME}"
                    echo "Action: ${params.ACTION}"
```

```
        }
    }
}

stage('Configure AWS & kubectl') {
    steps {
        script {
            withCredentials([[$class: 'AmazonWebServicesCredentialsBinding',
                        credentialsId: "${params.AWS_CREDS}"]]) {
                sh """
                    # Configure AWS CLI
                    aws configure set region ${params.AWS_REGION}

                    # Update kubeconfig for EKS
                    aws eks update-kubeconfig \\
                        --region ${params.AWS_REGION} \\
                        --name ${params.EKS_CLUSTER_NAME}

                    # Verify connection
                    kubectl cluster-info
                    kubectl get nodes
                """
            }
        }
    }
}

stage('Create Namespace') {
    when {
        expression { params.ACTION == 'deploy' }
    }
    steps {
        script {
            sh """
                kubectl create namespace ${params.NAMESPACE} --dry-run=client -o yaml | kubectl apply -f -
            """
        }
    }
}

stage('Create Docker Registry Secret') {
```

```
when {
    expression { params.ACTION == 'deploy' }
}
steps {
    script {
        withCredentials([usernamePassword(
            credentialsId: "${
```