# **CI/CD Pipeline & Deployment Strategy for the Banking Application**

This document describes the CI/CD pipeline setup for building and deploying the Account and Transaction services using Github Actions.

The pipeline will

* Trigger on new commits to the main branch (git)
* Build (and Test) the services
* Push the Docker images to Docker Hub
* Deploy the services to remote server using Docker Compose.

## Github Actions pipeline files

1. Create a Dockerfile each for account & transaction services

|  |
| --- |
| # Use Node.js official image  FROM node:18  # Set working directory  WORKDIR /app  # Copy package.json and install dependencies  COPY package.json package-lock.json ./  RUN npm install  # Copy the rest of the application code  COPY . .  # Expose the service port  EXPOSE 5001  # Run the server  CMD ["node", "server.js"] |

1. Create docker-compose.yml file for deployment at root of directory.

|  |
| --- |
| version: "3.8"  services:    account\_management:      image: mydockerhubuser/account\_management:latest      ports:        - "5000:5000"      environment:        - DATABASE\_URL=postgresql://postgres:password@db:5432/banking    transaction\_service:      image: mydockerhubuser/transaction\_service:latest      ports:        - "5001:5001"      environment:        - DATABASE\_URL=mongodb://mongo:27017/bankings    db:      image: postgres      environment:        POSTGRES\_USER: postgres        POSTGRES\_PASSWORD: password        POSTGRES\_DB: banking      ports:        - "5432:5432"    mongo:      image: mongo      ports:        - "27017:27017" |

1. Github repository setup:

- Create/Use the Github repository and add new repository “banking-app”

- In VS Code, run “*git init*” in the root folder so that both account & transaction services are in the same repo.

- Run “*git remote add origin <github repo url>*”

- Add .**gitignore** file to skip files/folders like **node\_modules**, **logs**, etc.

- Run “*git add .*” to add/stage all the files of account and transaction services.

- Run “*git commit -m “<commit message>*”

- Run “*git push –set-upstream origin master*”

1. Create **deploy.xml** file (.github/workflows/deploy.xml) in root of project as below:

|  |
| --- |
| name: CI/CD Pipeline for Account & Transaction Services  on:    push:      branches:        - main  # Trigger on code push to the main branch  jobs:    build-and-test:      name: Build & Test Services      runs-on: ubuntu-latest      steps:        - name: Checkout Repository          uses: actions/checkout@v3        - name: Set up Node.js          uses: actions/setup-node@v3          with:            node-version: 18        - name: Install Dependencies for Account Service          run: |            cd account\_management            npm install        - name: Run Tests for Account Service          run: |            cd account\_management            npm run test        - name: Install Dependencies for Transaction Service          run: |            cd transaction\_service            npm install        - name: Run Tests for Transaction Service          run: |            cd transaction\_service            npm run test    build-and-push:      name: Build & Push Docker Images      runs-on: ubuntu-latest      needs: build-and-test  # Only runs if tests pass      steps:        - name: Checkout Repository          uses: actions/checkout@v3        - name: Log in to Docker Hub          run: echo "${{ secrets.DOCKER\_PASSWORD }}" | docker login -u "${{ secrets.DOCKER\_USERNAME }}" --password-stdin        - name: Build & Push Account Service Image          run: |            cd account\_management            docker build -t mydockerhubuser/account\_management:latest .            docker push mydockerhubuser/account\_management:latest        - name: Build & Push Transaction Service Image          run: |            cd transaction\_service            docker build -t mydockerhubuser/transaction\_service:latest .            docker push mydockerhubuser/transaction\_service:latest    deploy:      name: Deploy to Server      runs-on: ubuntu-latest      needs: build-and-push  # Runs after Docker images are pushed      steps:        - name: Deploy via SSH          uses: appleboy/ssh-action@v0.1.6          with:            host: ${{ secrets.SERVER\_HOST }}            username: ${{ secrets.SERVER\_USER }}            password: ${{ secrets.SERVER\_PASSWORD }}            script: |              cd ~/banking-app              git pull origin main              docker-compose pull              docker-compose up -d --remove-orphans |

#### CI/CD Stages in the above config:

**Stage 1: Build & Test**

* Checks out the repo
* Installs the dependencies for both the projects
* Runs unit tests.

**Stage 2: Build & Push Docker images**

* Build Docker images of both services.
* Push the images to Docker Hub

**Stage 3: Deploy to Server**

* Connect to remote server using SSH
* Pull the code (git pull) to get latest
* Pull latest Docker images
* Deploy using Docker Compose command.

**Placeholders used**

Before running the above, the following needs to be setup in Github repository:

|  |  |
| --- | --- |
| **Secret Name** | **Description** |
| DOCKER\_USERNAME | Dockerhub username |
| DOCKER\_PASSWORD | Dockerhub password |
| SERVER\_HOST | Public IP of the server |
| SERVER\_USER | SSH username |
| SERVER\_PASSWORD | SSH password |

## Deployment Strategy

There are various strategies to deploy applications into production environments. Prominent ones are:

* Recreate
* Rolling
* Blue-Green
* Canary
* A/B Testing
* Shadow

#### Recreate Deployment:

The Recreate strategy is a deployment where the current version say, A is shut down then the new version say, B is deployed. This strategy implies downtime in between the shutdown of version A and version B being up.

#### Rolling or Ramped Strategy:

The Ramped or Rolling strategy involves slowly rolling out the new version by replacing the instances of existing version one after another until all the instances are new versions. This strategy takes time as it is a slow deployment and would require supporting different versions of APIs at same time.

#### Blue/Green Strategy:

This strategy differs from Rolling in that the versions A (Blue) and B (Green) are active at the same time. After testing the new version extensively, the traffic from current version is switched to the new version. The rollout/rollback is instant and there is no downtime of the application.

#### Canary Strategy:

Canary Strategy consists of gradually shifting production traffic from version A to version B, usually based on weights. For example, 90% of traffic goes to current version and 10% to new version. Once the new version is tested with this traffic and is deemed ready, rest of the traffic is shifted to new version. This is also a slow rollout and users can have differing experiences based on which version they access.

#### A/B Testing:

A/B Testing strategy consists of routing a subset of users to new version under specific conditions. It is usually used to make specific business decisions based on user details. But can be used for deployments as well. Specific conditions like Browser type/version, User’s device, geolocation, etc. are used to rollout the versions. This does bring issues with troubleshooting issues as the version accessed by the user has to be known.

#### Shadow Strategy:

Shadow Strategy consists of deploying new version along with the current version and ensuring the traffic is forked/copied to hit new version as well. This is helpful to do performance and functional testing of new version before rollout. This does add the complexity of setup and the need to have special requirements for traffic direction to new version as well.

Summing up, comparing the above deployment strategies, the most suitable strategy for the Banking application would be one which would ensure High-availability, instant rollout/rollback ability for critical services and best user experience. So **Blue/Green Deployment strategy** would be the most apt one as it also has zero/no downtime.

#### How it would work:

Two identical deployments with

* **Blue** environment (Active): runs the current stable version of the application
* **Green** environment (Idle/Inactive): runs the new version to be rolled out.

Deployment process steps:

* Deploy the new version (**Green**) while keeping existing (**Blue**) live.
* Perform testing in **Green** environment.
* If everything works fine, switch the traffic (using Load Balancer/API Gateway)

**Blue**->**Green**.

* **Blue** environment is now Idle but available for rollback in case of issues with **Green**.