Problem Statement: CI/CD Pipeline for Multi-branch GitHub Repository Using Jenkins, Webhooks/Polling, and Docker Integration

Context:

You are tasked with automating the continuous integration (CI) and continuous deployment (CD) process for a multi-branch GitHub repository using Jenkins. The repository contains two primary branches:

- **Main branch**: Represents the production-ready codebase.
- **Dev branch**: Represents the continuously evolving and developing branch.

The system needs to be designed so that:

1. **Dev Branch**:
    - Continuous development happens here.
    - Any changes to this branch must trigger an automated build and testing pipeline in Jenkins.
    - Upon successful completion of the pipeline, a Docker image tagged as `dev` should be built and uploaded to GitHub Packages.
2. **Main Branch**:
    - Code from the dev branch is merged into the main branch through pull requests.
    - When the PR is merged, a Jenkins pipeline should:
        - Check for code changes.
        - Run the build and test pipeline.
        - Create a Docker image, allowing parameters for deployment stage and image tag (for the production version).
        - If deployment is enabled, the tagged Docker image should be uploaded to Docker Hub.

Requirements:

1. **GitHub Setup**:
    - You have a **GitHub repository** with two branches: `main` (production) and `dev` (development).
    - The repository uses **GitHub webhooks** (or polling) to notify Jenkins of changes in branches.
    - Each branch contains its own `Jenkinsfile` for defining its respective CI/CD pipeline.
    - A **pull request (PR)** is created whenever dev branch changes are ready to be merged into the main branch.
2. **Jenkins Configuration**:
    - A **Jenkins Multibranch Pipeline** job should be set up to work with the GitHub repository, automatically detecting new branches and Jenkinsfiles.

- The pipeline should poll the repository at regular intervals to detect any new changes.
- **Dev Branch Pipeline**:
    - The pipeline triggered by any changes in the `dev` branch should execute the following steps:
        - Clone the repository.
        - Build the project.
        - Run unit and integration tests.
        - On successful completion of tests, create a Docker image tagged with `dev`.
        - Push the Docker image to **GitHub Packages**.
- **Main Branch Pipeline**:
    - After the dev branch code is merged into the main branch via a PR, the main branch pipeline should be triggered.
    - It should:
        - Verify the code changes.
        - Build and test the application (similar to the dev branch).
        - Allow **pipeline parameters** for:
            - Enabling or disabling the **deployment stage**.
            - Providing a **tag** for the production Docker image.
        - If the deployment is enabled, it should:
            - Build a Docker image with the provided tag.
            - Push the Docker image to **Docker Hub**.

Deliverables:

1. **GitHub Repository**:
    - The GitHub repository must contain two branches:
        - `main` for production-ready code.
        - `dev` for active development.
    - Each branch must have its own `Jenkinsfile` that defines the respective pipeline steps for that branch.
2. **Jenkins Pipeline**:
    - A multibranch pipeline job on Jenkins that:
        - Detects changes to the `dev` branch and triggers an automated build and test pipeline, followed by Docker image creation and upload.
        - Detects changes (via PR merge) to the `main` branch and runs a parameterized pipeline that builds, tests, and optionally deploys the application.
3. **Docker**:
    - On successful completion of the `dev` pipeline, a Docker image tagged `dev` must be uploaded to **GitHub Packages (https://ghcr.io)**.
    - On completion of the `main` pipeline, if deployment is enabled, a Docker image with the specified tag must be uploaded to **Docker Hub**.

4. **Webhook/Polling**:
    - GitHub repository must have a webhook configured (or Jenkins polling set up) to ensure that Jenkins is notified of any changes to either branch.

Bonus (Optional):

- Implement a **Slack notification** system in Jenkins to notify the team of the success or failure of builds and deployments.
- Include a **branch protection rule** for the main branch to ensure that PRs need to pass the Jenkins pipeline checks before merging.