Branching Strategy

Date: 10/31/2023

Driver: Harivardhan Pyaram (PRFT)

Approvers:

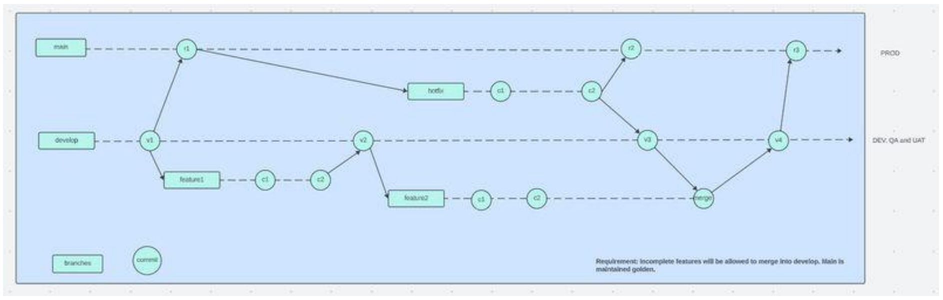
- Ed Barr
- Matthew Crowley
- Suchita Chanda
- Krishnachandra Epili

Participants:

- John Hodgkinson (PRFT)
- Surendar Madasani (PRFT)
- Gerardo Leyton (PRFT)
- Alyssa Blomquist (PRFT)

Decision:

The Org will use Gitflow Deployment Strategy.


main
r1
r2
r3
PROD
hotfix
c1
c2
develop
v1
v2
v3
v4
DEV, QA and UAT
feature1
c1
c2
feature2
c1
c2
merge
branches
commit
Requirement: Incomplete features will be allowed to merge into develop. Main is maintained golden.

Options Considered:

Gitflow Deployment Strategy

After further discussion, the Team decided on a combined strategy where we maintain a `develop` branch to deploy to all lower environments but PROD deployments are done only from the `main` branch. Feature branches are used to build new features and merge into `develop`. And `release` branches are created to merge code into `main`. Hotfix branches are required to be merged into both `develop` and `main`.

Pros

- Only one branch needs to be golden, but we still have another branch for CI/CD.
- Hotfix Releases have a very short turnaround.
- Automation takes care of Code Promotions until UAT.
- Fast feedback loops if something fails.
- Incomplete features can go through multiple merges into `develop` and can be pushed through lower environments.

Cons

- There will be code drift between `develop` and `main` if all code in `develop` isn't promoted to `main` regularly.
- The Team Leads have to be very diligent before approving PRs into `main` because it isn't an automated process.
- Feature development teams will have to track all the commits merging into `develop` that form their complete feature so they can create the next PR into `main`.

Main Branch Deployment Strategy (Feature Branch Deployment Strategy)

Only main branch is used to deploy to all environments.

Pros

- Only one branch needs to be golden.
- Frequent Releases with Short Production cycles.
- Automation takes care of deployments to higher environments.
- Fast feedback loops if something fails.

Cons

- All deployments happen from a single branch.
- Timed releases to environments are not possible.
- Releases are pushed in order unless failures block a deploy.
- Team doesn't have much control on deploys to higher environments beyond merging the PR.
- Any code fix has to go through all environments.

Demo

Repo Location

The repo contains two things:

- Python Script - Looks up the values of DB_URL, DB_USERNAME and DB_PASSWORD in the environment variables and prints what it sees to console. (Simulates a deployment script.)

- Workflow YAMLs under `.github/workflows` : A YAML defining the workflow for Github Actions to download the code and run the deployment script. Uses the same code to deploy to multiple environments in a linear fashion. Important keywords to track are `environment` and `needs`.

Environments

There is an entry created for every environment that the repository will be deploying to. Environment specific information is stored in variables and any protected information is stored in secrets. These are both accessible in Github Actions Workflow.

`prod` Environment also has an extra Required Reviewers rule that blocks deployments from deploying to Prod without an approval from defined users/teams.

Protected Branches

Only the `main` branch is protected and considered golden.

Branch Protections: https://github.com/Commonwealthcare/gitflow-strategy-demo/settings/branch_protection_rules/43346531

Workflows

A single workflow takes care of deploying to all the environments. Any deployments that do not succeed will block promotion to higher environments. Prod deployments will wait for approval. Each deployment stage will only access the values in the `environment` defined for it.

Pull Requests

Only one PR is needed per code change to get it to all environments.

Release Branch Deployments

Each environment has its associated release branch.

Pros

- Every release requires PR approvals and there's clear track of what code is going into each environment.
- Approvals can be given in a certain timeframe to trigger timed releases to certain environments.
- Code can be bundled together for release to higher environments.
- Targeted bug fixes to an environment are possible but creates code drift unless cherry-picked.

Cons

- Every PR requires at least one approver and some manual work to merge.

- There's no automated promotion to higher environments.
- If something fails in the higher environment, someone needs to track that back to the relevant team and restart the whole process.

Repo Location

The repo contains two things:

- Python Script - Looks up the values of DB_URL, DB_USERNAME and DB_PASSWORD in the environment variables and prints what it sees to console. (Simulates a deployment script.)
- Workflow YAMLs under .github/workflows:
    - Release Workflows: Multiple YAMLs defining the workflow for Github Actions to download the code from a specific branch and run the deployment script against a specified environment. Important keywords to track are `environment` and `branches`.
    - PR Workflows: A PR Workflow that blocks PRs from merging into specific environment branches unless the source branch matches requirements.

Environments

There is an entry created for every environment that the repository will be deploying to. Environment specific information is stored in variables and any protected information is stored in secrets. These are both accessible in Github Actions Workflow.

prod Environment also has an extra Required Reviewers rule that blocks deployments from deploying to Prod without an approval from defined users/teams.

Protected Branches

The `main` branch and any branches starting with `release-` are protected and all these branches should be considered golden.

Workflows

Multiple workflows deploying code from a specific branch to specific environments. Prod deployments will wait for approval. Each deployment stage will only access the values in the environment defined for it.

Pull Requests

One PR per environment is needed for code changes to be promoted. The PRs will have a PR build breaker that blocks the PR from merging if the source branch isn't the lower environment branch. Higher environment PRs can include multiple commits from the previous branch.