Z E B Y L

Nov 23, 2020

TO: **Dev-ops & Development Team**

Prepared by**: Dishant Agnihotri**

Re: **Production Deployments**

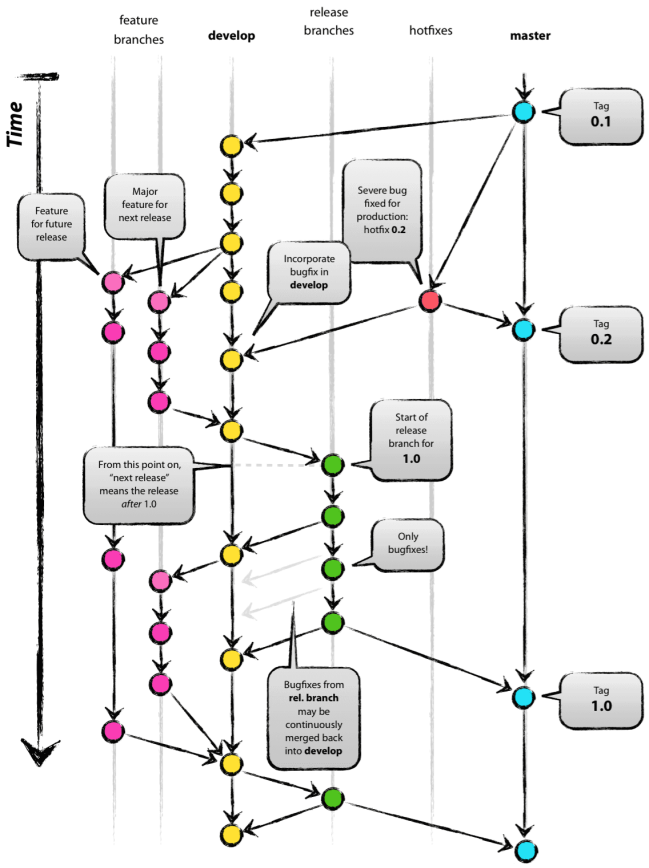
Please Note: This document considers we've not set up CI/CD pipelines or hooks.

Version Control using Centralized Git Workflow:

Centralized Workflow is important so that we can easily debug the issue, fast rollbacks and easy collaboration between multiple developers & dev-ops. The Centralized Workflow uses a central repository to serve as the entry for all changes to the project.

The **default production deployment branch is called master** and the **default development branch is develop**.

The architecture for the centralized workflow is explained using the below diagram:



Looking at the big picture we can see the possibilities for easier collaborations & versioning. Git Flow is an evolution of Feature Branch Workflow, so it reuses all principles of each pattern and includes more as “develop”, “hotfixes” and “release” branches.
Branches Explanation:

1. **For Developers:**
    a. **develop**: Develop a.k.a development is the default and main branch for the developers. No direct commit will be tolerated. Develop branch can only be used to cut “**feature**” branch for actually committing the latest code.

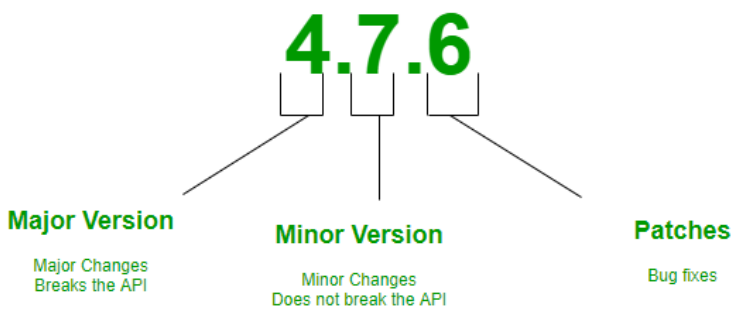
    b. **feature**: Feature branches are the developer friend. They will be used to push code & to raise PR’s. A single user can cut as much as PR they want. Each feature represents a module on which the developer is working and should be named accordingly.
        i. For example: A developer is working on the login page UI then it should be named as: feature/login-page-ui and another developer is

working on the register page then it should be written as feature/register-page-ui.

c. **hotfixes**: These branches can be directly cut from the master with version control naming and can be used to fix quick bugs. Developers can cut hotfixes.

2. **For Dev-ops:**
   a. **master**: Master is the default and main branch for the dev-ops. No direct commit will be tolerated. Master branch can only be used to merge releases and to cut hotfixes.

   b. **release** - Releases are the branches that need to always be cut from the develop with a version & to merge inside the master for deployment. A release needs to be cut once we want it to be deployed on the master.
      i. Please note: Summary of commits needs to be added inside the description so that it can be easily debugged in the near future.
      ii. **Version controlling**: Below image can help to understand how version controlling is done:



   c. **hotfixes**: These branches can be directly cut from the master with version control naming and can be used to fix quick issues. Dev-ops can cut hotfixes.
   d. **tags**: On successful production deployment, the tag needs to be assigned onto the master branch with the same version control name of the most recent release. Each tag will have the same commits summary as of the recent release.

For detailed documentation refer to this link:
https://nvie.com/posts/a-successful-git-branching-model/

Centralized Git Workflow based upon our setup

As we've used two different setups or repos for the same project then the below workflow will be used for centralizing git workflow:

1. **https://github.com/TextAspectInc/Development-Module**
    a. "master" branch will be the finalized branch from the development team. On completion, we'll notify the dev-ops team to use it as a production release.
    b. "feature" we'll cut feature branches for the development of the modules. On completion, we'll merge them to master.

2. **https://github.com/TextAspectInc/MH_SMS_Portal**
    a. "master" branch will be the main production branch.
    b. "develop" branch will be used to migrate code from "Development Module".
    c. Once the code is migrated and commit to "develop" then a release will be cut with the proper versioning & summary of commits in the description.
    d. The release will be merged to "master".
    e. Deployment starts.
    f. On successful deployment & developers testing, a tag needs to be generated with the same description as of the previous release.
    g. The same progress will keep on following for each release.

Important Tools Required

As we i.e. developers don't have access to the production server, we need the below setup for debugging & monitoring purposes:

1. **Apache JMeter:** (https://jmeter.apache.org/): For load type performance testing of new releases. This is a free tool. If we are installing UppTime then we don't need to have this one installed.
    a. **Please Note**: This should be done by the dev-ops team.

2. **Pingdom**: (https://www.pingdom.com/): For static content & release size of the new release. This is a free tool, please sign-up for a free account.
    a. **Please note**: We don't need to set up this tool on the server. We just need a report before and after deployment. As mentioned inside the Pre-Deployment Checklist & Post-Deployment Checklist.
    b. I have shared a drive to upload the reports for each release. Please create a folder for each release version & upload both the report with the name "pre-deployment report" & "post-deployment report"
    c. **Please note**: This should be done by the dev-ops team.

3. **Sentry:** (https://sentry.io/welcome/): For Error Tracking. This is a free tool.
    a. **Please note**: This should be done by the development team.

4. **UppTime** (https://upptime.js.org/): For uptime & log tracking. This is a free tool.
    a. This needs to be installed on the server.
    b. **Please note**: This should be done by the dev-ops team.
    c. Please add monitoring emails for "dishantagnihotri@gmail.com" & "rahul.kumar.mj5@gmail.com" **& pejman's sir.**

5. **Cloudflare (https://www.cloudflare.com/en-gb/):** Not required but recommended.

#TODO: @ Consult with Hassan.

Based on your needs, you can set up free tools provided by SolarWinds for all kinds of server needs. All important tools are free or come with generous free plan: https://www.solarwinds.com/

Pre-Deployment Checklist

1. **Deployment Objective Updations (Dev-ops)**
    a. Deployment objective document needs to be filled with the date & summary of commits that will be going to take effect into the release. The branch that will take effect should also be mentioned.
    b. The next release tag should also be mentioned with a reason why the tag is incremented.
    c. Server report needs to be added for the previous release with the data specified when the report has been taken.

d. **NOTE**: I have attached a sample document for the deployment objective.

2. **Git Workflow (Dev-ops)**
    a. All pending branches should be merged inside the develop branch. This will be done by the development team.
    b. Create a new release branch as specified inside the Centralized Git Workflow and the increment should be proper. All the commits summary needs to be added properly to the description.
    c. Once released, the branch can be merged to master and a tag needs to be generated on the master branch specifying the same number as of the release node. Please do note: All the descriptions specified inside the release branch should also be done inside the tag.

3. **Git Clean-up (Dev-ops & Development)**
    a. Once the tag is generated, the git needs to be cleaned for future deployments.
    b. All branches need to be deleted except the “active”, “master” & “develop” branches.
    c. This will be done by the development team on the “Development Module” & should be done on the “MH SMS Portal” by the dev-ops team.

4. **Building up of infrastructure code (Dev-ops)**
    a. Compiling & Minifying of JS & CSS bundles (if required)
    b. Before moving to production, the local build needs to be generated & verified.
    c. Docker image integrity needs to be verified (if required)

5. **Database Migrations (Dev-ops)**
    a. If the new release contains a new table schema or new field inputs then migrations have to be done.
    b. Migrations can never be done directly. **Back-up needs to be generated & renamed with today's date**.
    c. The backup needs to be stored where it can be accessed easily.
    d. Now, the new table or fields needs to be created.
    e. Test the website just to be sure, it doesn’t affect the current execution of the website.

6. **Basic To-do (Dev-ops & Developers)**
    a. Please verify upptime is working & have all working emails (required)

b. Clear the server cache (required)
c. CDN integrity verification (if required)
d. Stop the CRON jobs (required as they cause issues). Please do verify all-important jobs has been finished, as we can't pause jobs.

7. **Current Image Backup (Dev-ops)**
    a. Delete the previous deployed server image
    b. Make an image backup with the name of today's date. (required)
    c. Test the backup (if required)

Post-Deployment Checklist

1. **Use Incognito Window**
    a. Every deployment the test needs to be done on the incognito window.

2. **Notify the development team (Dev-ops)**
    a. Head up to the development team to inform them of the deployment in case anything happens that requires a follow-up.

3. **Basic To-do (Dev-ops)**
    a. Clear the server cache (required)
    b. Reset Nginx or apache (required)
    c. Clear the route cache (if setup)
    d. Monitor Exception Rates: Leverage an error-tracking platform to measure the number of exceptions occurring, and if the rate increases, consider rolling back the change.
        i. This will be done by the developers using sentry reporting. (if required)
    e. Checking up of database connection (required)
    f. Serve the latest codebase onto the CDN (if setup)
    g. Re-initialization of the CRON Jobs (required)
    h. Check integrity of load-balancer if we're using (if setup, yes)
    i. Gzipping (required)

4. **Monitor**:
    a. Monitor the overall performance of the website. Look for abnormalities in CPU, memory, network or disk usage using upptime.

b. Server report needs to be taken and added onto the “Deployment Objective Report” with the date when the report is generated. The drive is shared for uploading the report.

5. **Testing**:
    a. SMTP & 3rd party integration needs to be tested. (required)
    b. Tests need to be performed for the latest deployed update. This will be done by developers as well as dev-ops. (required)

6. **Security**
    a. Passwords need to be changed & updated every time a new release goes into the server. The password needs to be stored inside 1Password for all the team members to have access to it in the near future. (required)
    b. Environment variables need to be verified again & all-important passwords and keys like PayPal integration key, stripe integration key, database password or other needs payment & security related passwords need to be changed after each release.

7. **Post Mortem Meeting**
    a. Discuss how the deployment went during the next team meeting.

How to revert the old deployment

Git pull ~Head${1}