**Pre-conditions:**

|  |  |
| --- | --- |
| Delivered service | Financial data processing & Billing system |
| Code Language | Python |
| Team setup | 5 Dev; 1 DQ, 3 DevOps |
| Architecture preferences: | Docker, Jenkins, Git |
| Service preferences | Open Source |
| Development approach | Agile |
| Release frequency | 1 per 2 weeks in pre-prod; 1 a month - prod |

**Brunch strategy:** For this project **I** recommend using Git flow merging strategy. It includes two main branches - develop branch and master brunch (for new releases brunch). Its possible to create new brunch for every feature.

**Why such branch strategy:**

* Code in master brunch is clean and tested
* Master branch stores release history
* Developer can work on development the same feature
* Develop brunch stores history for all commits, it easy to switch between features brunches.

**Example:**

* Create brunch for development

*git branch dev*

* Create brunch for new feature

*Git checkout -b feature\_1*

* Add changes in brunch feature

*Git add.*

* Commit new changes in brunch feature

*git commit -m “*[*new*](https://github.com/anastaviki/ci-cd/commit/0d48fff08a402956b78e924a1a8602f0ad4dab24) *feature was added”*

* Push new changes in brunch feature

*git push  feature*

* Merge brunch feature to dev brunch

*git checkout dev*

*git merge feature*

*git push  dev*

*-*Create release branch and push it

*git checkout -b release*

Add release changes to branch and commit changes

*git add .*

*git commit -m “*[*release*](https://github.com/anastaviki/ci-cd/commit/0d48fff08a402956b78e924a1a8602f0ad4dab24) *changes were added”*

*git push  release*

*-*Merge release branch into main

*git checkout main*

*git merge release*

*git push main*

After every release developers need to update dev brunch according release brunch before they start working with it.

Developers should not forget pull changes to feature brunches before pushing changes to dev brunch.