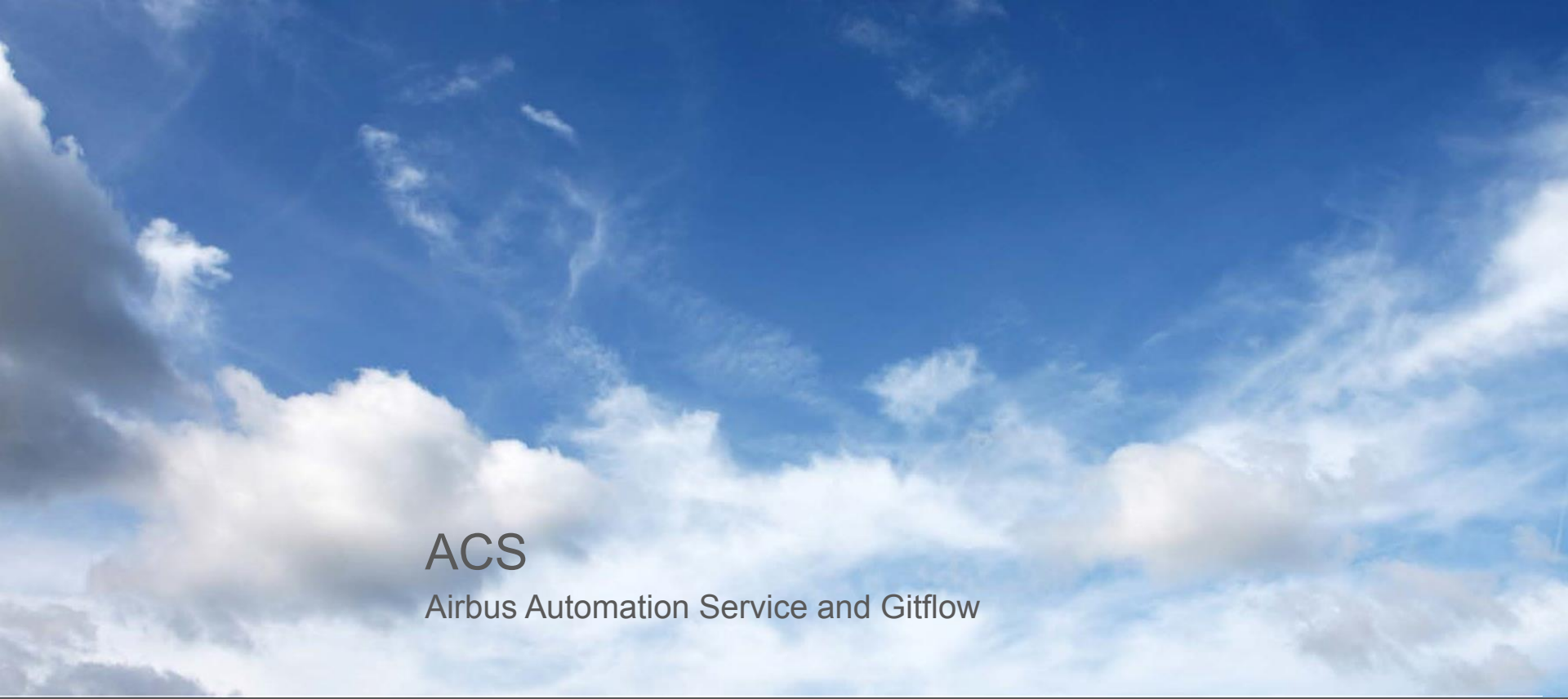


ACS

Airbus Automation Service and Gitflow

DevOps Team

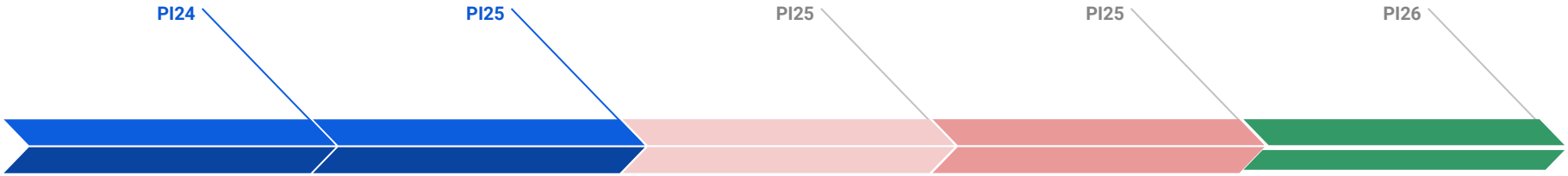
06/07/2022

AIRBUS

SUMMARY

1. Migration Strategy
2. Designed architecture for ACS
3. DevOps Toolchain configuration
4. ACS Gitflow
5. From Gerrit to Github
6. Working process on GitHub
7. Best practices
8. Migration to AAS follow-up

1. Migration Strategy



Workshops

With AAS architects

Architecture design

Adapt to CPAO context

Technical onboarding - POC with a full Use case

Include installation & configuration

Full test case

Migration of 3 Pilot modules

MODMGT,GLOBAL SEARCH, NX

Migration of all ACS Modules from the start of PI

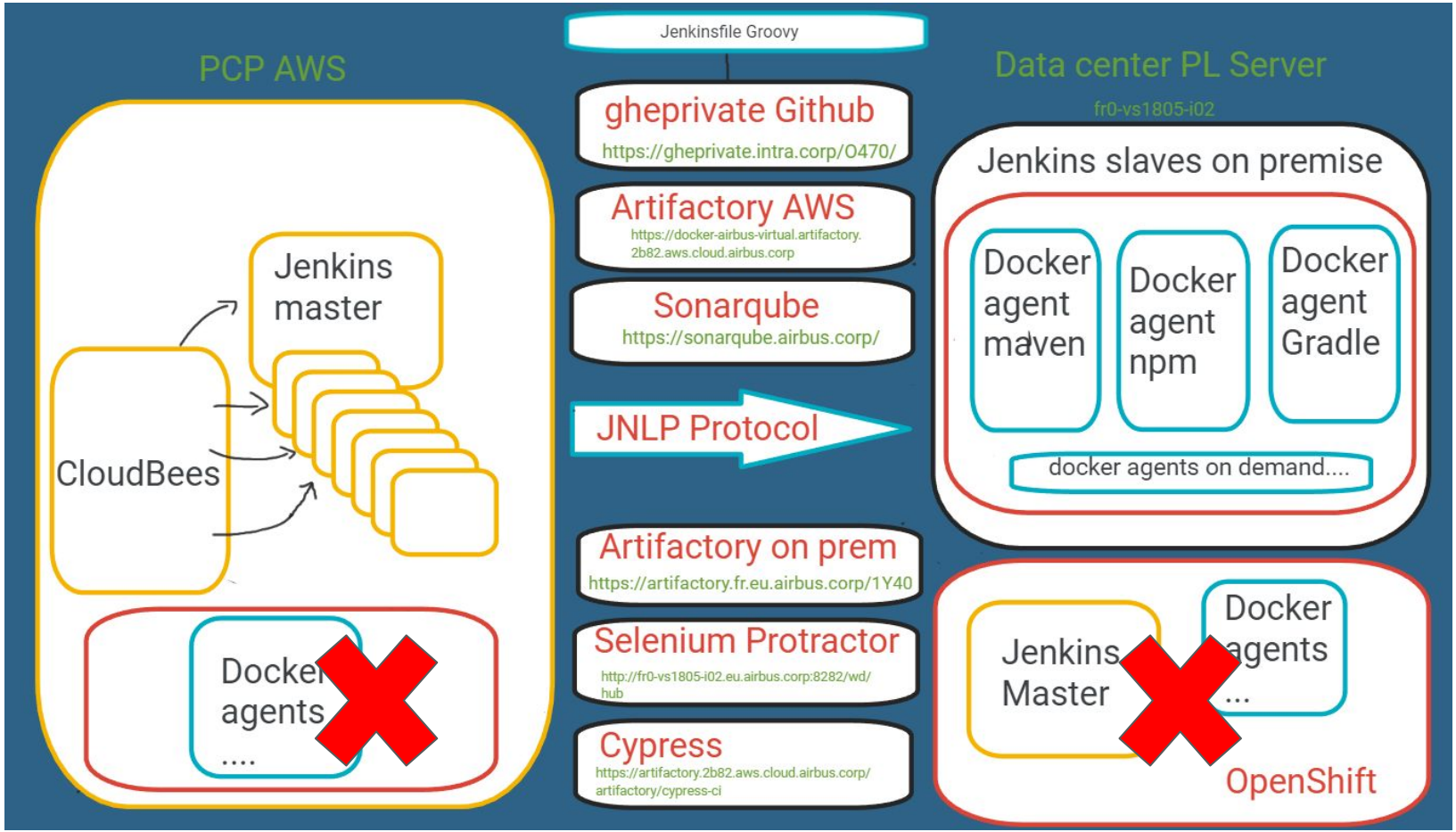
Everything must be done by the end of the PI. old PI must be locked

2. Designed architecture for ACS (1/4)

All the informations about Designed architecture for ACS:

https://confluence.airbus.corp/pages/viewpage.action?pageId=274697234

2. Designed architecture for ACS (2/4)



2. Designed architecture for ACS (3/4)

Shared information & assets:

- Access Management via ET4GPMG group groupe : 2J00-jenkins-o470-users
- JAC, workflow & pull requests
- Scalability
- Easy-to-reproduce environments
- Flexibility provided by Docker
- On premise Managing Platform Except for Jenkins Master
- Easy way of changing versions (maven, npm, gradle, openjdk, …)
- On going: Github actions

Documentation:

https://confluence.airbus.corp/display/2DUD/

2. Designed architecture for ACS (4/4): Migration Steps

1. Ask for AAS Jenkins instance user to DevOps Team (https://**o470-o470**-acs.jenkins.2b82-devops.aws.cloud.airbus.corp)
   > Sign-in via OneLogin 2J00-jenkins-o470-users
2. Ask for Github GHE PRIVATE access
   a. **Login for first time to** https://gheprivate.intra.corp/O470
   b. **Created Organization = O470**
   c. **Sign-in via OneLogin**
   d. **Ask for access as Member or Owner to DevOps Team**
3. Go to ACS job folder in Jenkins : https://o470-o470-acs.jenkins.2b82-devops.aws.cloud.airbus.corp/job/O470/
4. If not created yet, create your own project folder and sub-folders if necessary
   > Example : **O470/MODMGT/NOMENCLATURE_BACKEND/**
5. Create your own Jobs under created folder in above :
   a. **JOB type = Multibranch pipeline**
   b. **build configuration = Jenkinsfile**
   c. **no password in pipeline itself, must be managed by Jenkins Credentials (Token, Secret file, API key, SSH key)**
6. Create your own **private** repo on GHE Github & Clone your project from Gerrit to Github
7. Defined git workflow (strategie & owners, reviewers, …)
8. Under created Github project, add given Jenkinsfile Template to root folder
   a. Use Jenkins_slave_maven for your Maven project
   b. Use Jenkins_slave_NPM for your NPM project
   c. Use Jenkins_slave_gradle for your Gradle project

   Example of jenkinsfile : https://gheprivate.intra.corp/O470/O470-MODMGT-NOMENCLATURE/blob/master/Jenkinsfile
9. Create your own Sonarqube project under sonarqube.airbus.corp/projects
   a. must be **private** project https://sonarqube.airbus.corp/projects
   b. ensure that pushed data is not **classified**
10. Cypress integration done. **V 9.7.1**
11. Selenium protractor E2E used instance installed on Production Line : http://fr0-vs1805-i02.eu.airbus.corp:8282/wd/hub

All the informations about Designed architecture for ACS:

https://confluence.airbus.corp/display/ZO470CPAGT/2.+DevOps+Toolchain+configuration

Moving from Gerrit to Github

Gerrit is over …

Before moving to Github, be sure that

- Everyone is the team have access to https://gheprivate.intra.corp/O470
- The team won't push changes to Gerrit
- There is no change waiting for review on Gerrit
- Gerrit is on read only mode

If you don’t have access to github, please ask to devops team :)

All process describe in details here :
https://confluence.airbus.corp/display/ZO470CPAGT/4.+From+Gerrit+to+Github

Create the project under github and first push

Under https://gheprivate.intra.corp/O470, create a new project

- With the same name than the one on Gerrit.
- Set it private.

After, go to project page and copy the git url of the new project. This url will be used to pushed your existing master branch to github with the following command:

```
git push -f <git project url> master
```

Set up branches

After push done, create develop branch on project page. You should have now 2 branches (master & develop) !

Set up develop as default branch and add 2 rules on each branch

- Add a rule on master: protect master from direct actions on it: only code from pull request approved by administrators will be allowed.
- Add a rule on develop: Protect master from direct actions on it: only code from pull request approved will be allowed.

The new Gitflow

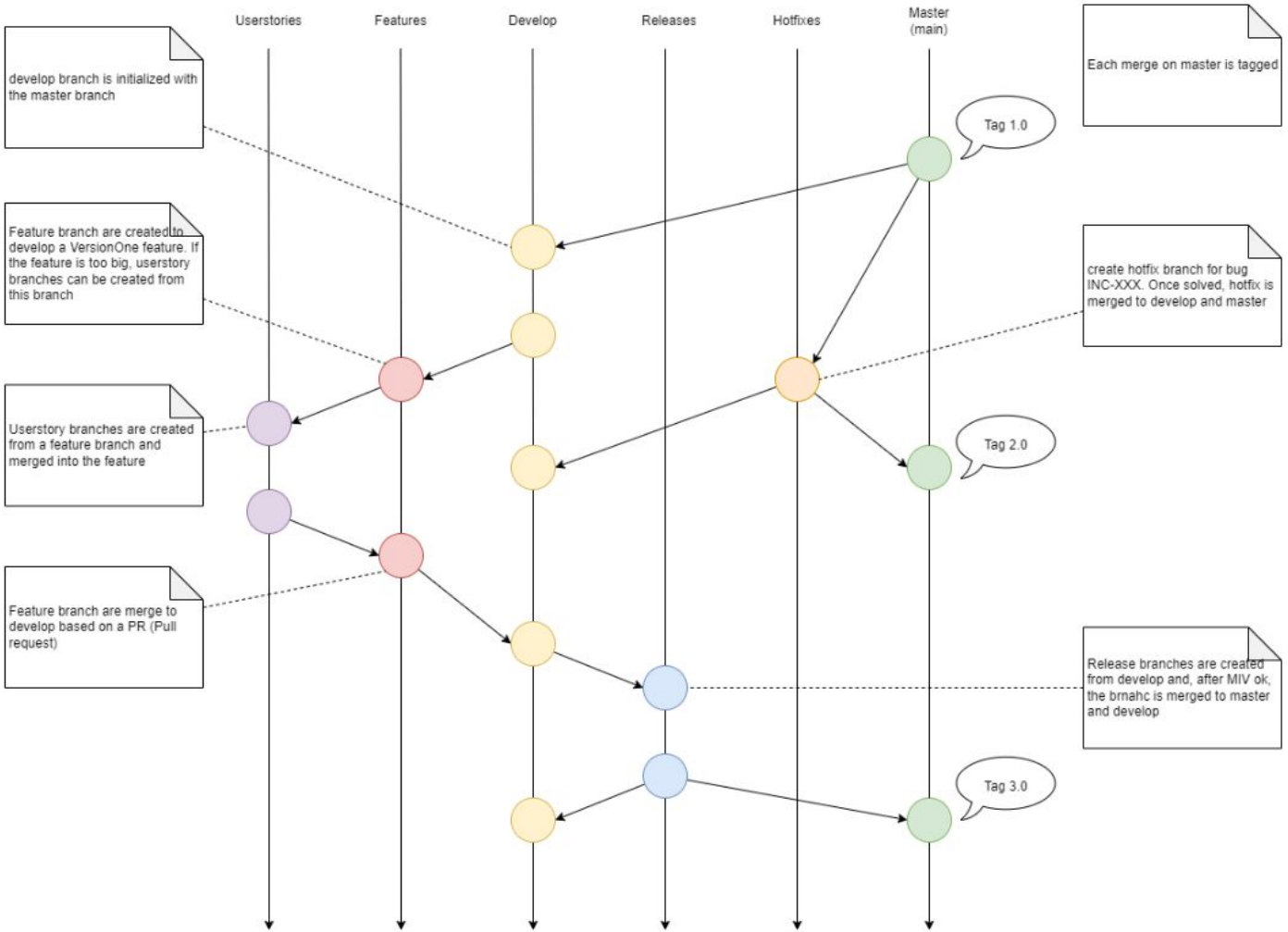
With new tool comes new power

With moving from Gerrit to Github, it’s time to define a new way of working for pushing, merging and reviewing source code !

We decided to use the standard Gitflow with some specificities :)

The principle changes from Gerrit are

- The main branches
- The supporting branches
- The Pull-request


Userstories
Features
Develop
Releases
Hotfixes
Master (main)
develop branch is initialized with the master branch
Feature branch are created to develop a VersionOne feature. If the feature is too big, userstory branches can be created from this branch
Userstory branches are created from a feature branch and merged into the feature
Feature branch are merge to develop based on a PR (Pull request)
Each merge on master is tagged
Tag 1.0
create hotfix branch for bug INC-XXX. Once solved, hotfix is merged to develop and master
Tag 2.0
Release branches are created from develop and, after MIV ok, the brnahc is merged to master and develop
Tag 3.0

Main branches

The master branch

This branch contains the production code and nobody can directly push in it. It is a protected branch and only merges from **releases** and **hotfixes** are authorized.

Master branch contains the tags of each release deployed.

The develop branch

This branch is used to merge the coming development for future versions of the modules. Pull request are requested to be able to merge code in it.

Supporting branches (1)

The features branch

The **features** branches are the branches where we develop a functionality. All **features** branches are prefixed with **ft/F-<featureID>** and could be deleted when merged. **Features** branches are created from **develop**.

If the feature from VersionOne is big and could last 2 or 3 iterations, you could use **user stories** branches to develop each user story and push them into the feature branch of the VersionOne's feature. At the end, the feature branch will be merges to develop from a pull-request.

The userstories branch

As said earlier, **userstories** branches are the branches are sub-branch use to develop a user story when the VersionOne feature is big.

All **userstories** branches are prefixed with **us/S-<userStoryID>** and could be deleted when merged into the **ft/F-<featureID>** branch.

Supporting branches (2)

The releases branch

The **releases** branches are the branches used for a release. They are created from **develop** and are prefixed **release-<nameOfTheRelease>**. This branch could contain fixes of the current release and will be merged when finish into **master** and **develop**.

The hotfixes branch

The **hotfixes** branches are the branches used for a hotfix on production. They are created from **master** and are prefixed **hotfixes-<IDofTheBug>**. This branch contains the fix of the current release and will be merged when finish into **master** and **develop**.

Respect the patterns

It is important to respect the pattern defined for the branch naming because it is used after in the Jenkinsfile

```
stage ('Build SNAPSHOT or RELEASE') {
     when {
       anyOf { branch 'master'; branch 'ft/F-*'; branch 'us/S-*' }
     }
     // build the branch !
}
```

```
stage ('Build and publish RELEASE') {
     when {
       anyOf { branch 'release-*'; branch 'hotfixes-*' }
     }
     // build and publish the release of the hotfix !
}
```

Daily work with Github

Using the Gitflow in your daily work

Here is the basic steps of Gitflow usage in your daily work.

Everything is describe in details in the following confluence page:

https://confluence.airbus.corp/display/ZO470CPAGT/5.+Working+process+on+GitHub

Starting a new development

Starting a new development is the first step ! Because we move from Gerrit to Github, it is better to reclone the project (for those who are not doing the migration) to avoid complications…

Do the clone command only once :)

```
git clone git@gheprivate.intra.corp:O470/O470-MODMGT-MOD-OPENING.git
```

Next step is to create a branch for the new development

```
cd <project>
git checkout develop
git pull
git checkout -b ft/F-XXX
```

Developing

Here, nothing changes !!

As usual, you write your unit tests, your integration tests, your cypress tests, your code and commit it when everything is tested and finished :)

You also verify that your code respect the Sonarqube standards and don’t forget that your commit message should follow few principles:

- Separate subject from body with a blank line.
- Limit the subject line to 50 characters.
- Capitalize the subject line.
- Do not end the subject line with a period.
- Use the imperative mood in the subject line.
- Wrap the body at 72 characters.
- Use the body to explain what and why vs. how.

Now with Github, you can also use markdown into your commit message !

Ending a new development and pushing it

Ending a new development needs 3 things:

- Merging the current feature branch with develop
- Pushing the feature branch
- Creating a Pull Request

```
git checkout develop
git pull
git checkout ft/F-XXX
git rebase develop
git push --set-upstream origin ft/F-XXX
```

Go the github to create the Pull Request: the url for the Pull Request is displayed in the logs after pushing to github :)

Url of the Pull Request

```
git push --set-upstream origin ft/S-X-adding-maven-sample
...
Enumerating objects: 11, done.
Counting objects: 100% (11/11), done.
Delta compression using up to 8 threads
Compressing objects: 100% (6/6), done.
Writing objects: 100% (8/8), 13.02 KiB | 666.00 KiB/s, done.
Total 8 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
remote:
remote: Create a pull request for ft/S-X-adding-maven-sample' on GitHub by visiting:
remote:
https://gheprivate.intra.corp/O470/O470-JMETER-PERFORMANCE-TESTS/pull/new/ft/S-X-adding-maven-sample
remote:
To gheprivate.intra.corp:O470/O470-JMETER-PERFORMANCE-TESTS.git
 * [new branch]      ft/S-X-adding-maven-sample -> ft/S-X-adding-maven-sample
```

The Pull Request

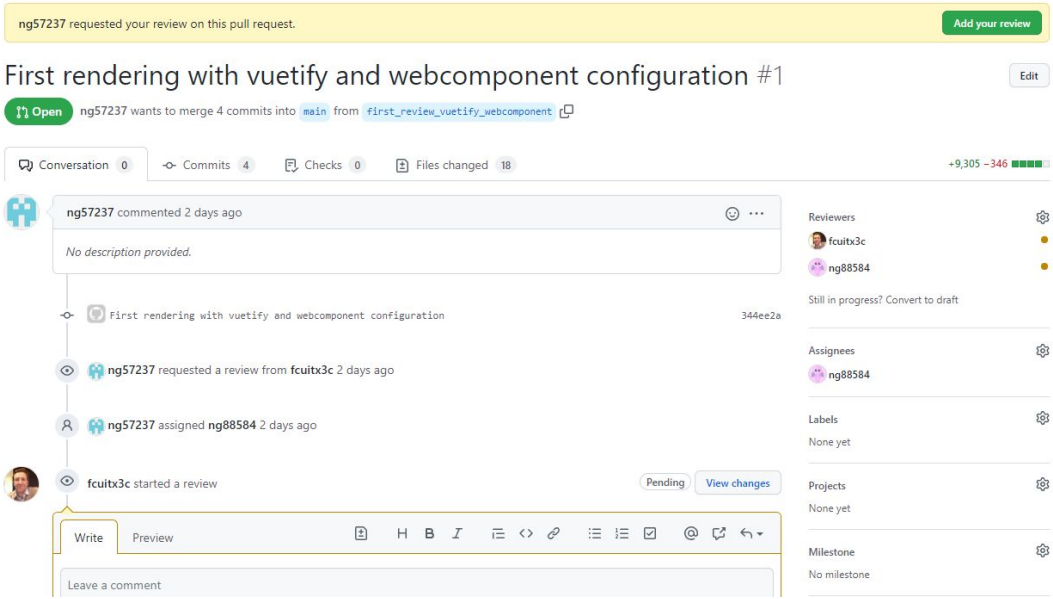
Nothing special here :)

You can add comments,

Review the commits and

The files, approve or

Request for changes,...



Migration to AAS follow-up

Here is the link to the follow-up sheet for github migration. It contains all the projects from Gerrit with name, status, …

Modules	Used	Migrated to AAS (GitHub/Jenkins)	Read only on Gerrit	Sonar AAS configured	Comments
A410					
A410-ANSIBLE					
A410-CADB					
ACS-DEV-ENVIRONEMENT					
ACS-MATERIAL-NX					Pushed on GitHub for FE migration preparation + pipelines migration

https://docs.google.com/spreadsheets/d/1CHVpYvQ6I52b05ASdCFb9QLuPnAQkbQb71F_LSPDRtA/

Do not forget to fill it !

Questions

Thanks