How to improve the developer experience in Heptapod/GitLab

Automate the dull works to focus on development

Noé Gaumont: ngaumont@logilab.fr
Logilab presentation and our code management
What problems are we facing within Logilab?
Existing solutions
Tools designed and used internally:
  - assign-bot to have review
  - gitlab-ci-template to have fine-tuned job definition
  - release-new to have proper release (to pypi)
  - cube-doctor to update dependencies
Logilab

- small IT service and consulting company (~25 persons) focused on:
  - Web Semantic,
  - Scientific computing,
  - Python training.
- Work almost only with open source softwares
- Contribute either with code or money.
Purpose:

- a semantic web application framework,
- developed in Python (LGPL),
- explicit data model,
- reusable components (called cubes),
- a front agnostic way to serve your data.

Our go-to tech stack is composed of:

- JavaScript
- CSS
- HTML
- Python
- PostgreSQL
- RDF
- Docker
Code organisation within Logilab

- **mercurial**, a vcs written in python
- **Heptapod**, a friendly fork of GitLab with mercurial support
- **multi-repo** approach:
  - CubicWeb
  - public cubes (~200 repo)
  - private cubes (mainly client related)
  - opensource project (~ 40 repos)
  - internal projects
Both are hard, problems occurs only faster within multi-repo architecture:
Problems caused by the multi-repo approach

Keeping one repo tidy is simple, keeping several hundred repos is hard.

Both are hard, problems occurs only faster within multi-repo architecture:

- Make sure all the test/lint are green
- Assign a reviewer for each Merge request
- Coherent code everywhere
- Maintain good practices in CI configuration
- Properly release everything to Pypi, npm, …
- Up-to-date docker images with correct tag
Existing solutions

Within Github, there is the probot solution such as:

- Auto-Assign
- Dependabot
- Release-drafter
- PR-triage
Existing solutions

Within GitLab, custom CI or danger-bot which is good for:

- Coding style
- Database review
- Documentation review
- Merge request metrics
- Reviewer roulette
- Single codebase effort
A tale of a CubicWeb MergeRequest

Classic use case: A new feature introduced in CubicWeb induced a deprecation warning.

Goal: The MR is reviewed, the tests pass and the repo depending on CubicWeb are updated, all version are published.
A scheduled job of assign-bot will assign a reviewer based on user preferences:

<table>
<thead>
<tr>
<th>Username</th>
<th>max_auto_review_per_week</th>
<th>max_auto_review_per_day</th>
</tr>
</thead>
<tbody>
<tr>
<td>naumont</td>
<td>15</td>
<td>3</td>
</tr>
<tr>
<td>username_2</td>
<td>ZZZ</td>
<td>WWW</td>
</tr>
</tbody>
</table>

It will also comment on the MR inactive for one week.
Running the correct set of tests

A coherent set of up-to-date jobs are run, thanks to a shared definition, `gitlab-ci-template`:

It takes care of:

- reducing the CI load by using custom image
- defining rules to prevent CI on some branches
- define some stages

```yaml
einclude:
  - project: "open-source/gitlab-ci-templates"
    ref: "branch/default"
    file:
      - "templates/no-duplicated-ci-pipelines.yml"  # use workflow to avoid duplicated pipelines
      - "templates/lint/black.yml"  # will do the equivalent of 'tox -e black'
      - "templates/tests/py3.yml"  # will do the equivalent of 'tox -e py3'
      - "templates/create-release-on-heptapod.yml"  # this will create a release on heptapod
      - "templates/build-docker-image.yml"
      # uncomment and uses to customize/extend the configuration here if needed
      # (it needs to be at the same level than "- project")
      - ".gitlab-ci-extended.yml"
```

It is always possible to customise the jobs within `.gitlab-ci-extended.yml`
A coherent set of up-to-date jobs are run, thanks to a shared definition, `gitlab-ci-template`:

It takes care of:

- reducing the CI load by using custom image
- defining rules to prevent CI on some branches
- define some stages

```yaml
include:
- project: "open-source/gitlab-ci-templates"
  ref: "branch/default"
  file:
    - "templates/no-duplicated-ci-pipelines.yml"  # use workflow to avoid duplicated pipelines
    - "templates/lint/black.yml"  # will do the equivalent of 'tox -e black'
    - "templates/tests/py3.yml"  # will do the equivalent of 'tox -e py3'
    - "templates/create-release-on-heptapod.yml"  # this will create a release on heptapod
    - "templates/build-docker-image.yml"

# uncomment and uses to customize/extend the configuration here if needed
# (it needs to be at the same level than "- project")
- ".gitlab-ci-extended.yml"
```

It is always possible to customise the jobs within `.gitlab-ci-extended.yml`
It uses:

- a custom docker image with all the dependencies already installed
- store some artifacts generated during the tests
- make the jobs interruptible
Once the MR is accepted, another job defined in `gitlab-ci-template` is used to build and publish a Docker image:

```yaml
image_build_latest:
  [...]  
  image:
    name: gcr.io/kaniko-project/executor:debug
    entrypoint: [""
  script:
    - echo "{auths":{"$CI_REGISTRY":{"username":"$CI_REGISTRY_USER","password":"$CI_REGISTRY_PASS
    - /kaniko/executor
    --context $CI_PROJECT_DIR
    --dockerfile $CI_PROJECT_DIR/Dockerfile
    --destination $CI_REGISTRY_IMAGE:$CI_COMMIT_HG_SHORT_SHA
    --destination $CI_REGISTRY_IMAGE:latest
  rules:
    - if: 'CI_COMMIT_REF_NAME == "branch/default"'

image_build_tag:
  [...]  
  image:
    name: gcr.io/kaniko-project/executor:debug
    entrypoint: [""
  script:
    - echo "{auths":{"$CI_REGISTRY":{"username":"$CI_REGISTRY_USER","password":"$CI_REGISTRY_PASS
```

The image tag is **correctly tag** inside the job.
Once the MR is accepted, another job defined in `gitlab-ci-template` is used to build and publish a Docker image:

```plaintext
image_build_latest:
  [...]
  image:
    name: gcr.io/kaniko-project/executor:debug
    entrypoint: [""
  script:
    - echo "{"auths":{"$CI_REGISTRY":{"username":"$CI_REGISTRY_USER","password":"$CI_REGISTRY_PASSWORD"}}" /kaniko/executor
    - /kaniko/executor
    - --context $CI_PROJECT_DIR
    - --dockerfile $CI_PROJECT_DIR/Dockerfile
    - --destination $CI_REGISTRY_IMAGE:$CI_COMMIT_HG_SHORT_SHA
    - --destination $CI_REGISTRY_IMAGE:latest
  rules:
    - if: '$CI_COMMIT_REF_NAME == "branch/default"'

image_build_tag:
  [...]
  image:
    name: gcr.io/kaniko-project/executor:debug
    entrypoint: [""
  script:
    - echo "{"auths":{"$CI_REGISTRY":{"username":"$CI_REGISTRY_USER","password":"$CI_REGISTRY_PASSWORD"}}"
```

The image tag is **correctly tag** inside the job.
Once the MR is accepted, another job defined in `gitlab-ci-template` is used to build and publish a Docker image:

```yaml
image:
  name: gcr.io/kaniko-project/executor:debug
entrypoint: [""
script:
  - echo "{{"auths":{"$CI_REGISTRY":{"username":"$CI_REGISTRY_USER","password":"$CI_REGISTRY_PASS
  - /kaniko/executor
  - --context $CI_PROJECT_DIR
  - --dockerfile $CI_PROJECT_DIR/Dockerfile
  - --destination $CI_REGISTRY_IMAGE:$CI_COMMIT_HG_SHORT_SHA
  - --destination $CI_REGISTRY_IMAGE:latest
rules:
  - if: '$CI_COMMIT_REF_NAME == "branch/default"
image_build_tag:
  [...] image:
  name: gcr.io/kaniko-project/executor:debug
  entrypoint: [""
script:
  - echo "{{"auths":{"$CI_REGISTRY":{"username":"$CI_REGISTRY_USER","password":"$CI_REGISTRY_PASS
  - /kaniko/executor
  - --context $CI_PROJECT_DIR
```

The image tag is **correctly** tagged inside the job.
Pushing the image to the registry

Once the MR is accepted, another job defined in `gitlab-ci-template` is used to build and publish a Docker image:

```
image_build_tag:
  [...]  
image:
  name: gcr.io/kaniko-project/executor:debug
  entrypoint: [""
  script:
    - echo "\"{\"auths\":\"${CI_REGISTRY}\":\"${CI_REGISTRY_USER}\",\"password\":\"${CI_REGISTRY_PASSWORD}\",\"\"}
    - /kaniko/executor
    --context $CI_PROJECT_DIR
    --dockerfile $CI_PROJECT_DIR/Dockerfile
    --destination $CI_REGISTRY_IMAGE:$CI_COMMIT_HG_SHORT_SHA
  rules:
    - if: "$CI_COMMIT_REF_NAME == "branch/default"
      when: never
    - if: "$CI_MERGE_REQUEST_ID"
      when: always
      --context $CI_PROJECT_DIR
      --dockerfile $CI_PROJECT_DIR/Dockerfile
      --destination $CI_REGISTRY_IMAGE:$CI_COMMIT_HG_SHORT_SHA
      --destination $CI_REGISTRY_IMAGE:$CI_COMMIT_TAG
```

The image tag is **correctly tagged** inside the job.
A new version is released

The MR is accepted and a new version has to be released. `release-new` is used for that:

```bash
$ release-new
Automatic release guesser decided to release the version '3.35.0' (minor)
Are you ok with that? [Y/n]: Y
🎉 Congratulation, we have made a new minor release 3.35.0 \o/

Now you need to hg push the new commits
```

Under the hood, it:

- detects the version number using semantic versioning
- updates the version number in the source
- generates a changelog from commit message and let you modify it
- update packaging MANIFEST.in if needed
- create a commit and then tag it
Once pushed, a job on the tag releases to pypi, with `gitlab-ci-template`.

```yaml
pypi-publish:
  [...]
  - if: '$CI_MERGE_REQUEST_ID'
    when: never
  - if: '$CI_COMMIT_TAG'
    when: on_success
script:
  - tox -e pypi-publish
```
Repositories are updated

cube-doctor will update repositories:

- Rebase the MR in same project
- In other project, create MR updating the package version
- Create MR for lint configurations

Manually, it can run automatic refactoring (handle deprecation warning).
Conclusion

- GitLab/Heptapod are a great tools battery-included:
  - scheduled job
  - registry
  - shared job definition.
- A Good CI jobs helps a lot to keep repositories clean and up-to-date.
- However, manual effort are still needed. In our case:
  - forge.extranet.logilab.fr/open-source/assign-bot
  - forge.extranet.logilab.fr/open-source/gitlab-ci-template
  - forge.extranet.logilab.fr/open-source/release-new
  - forge.extranet.logilab.fr/cubicweb/cube-doctor

Noé Gaumont : ngaumont@logilab.fr