INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES LYON
Inserm
CNRS
Lyon 1
UNIVERSITÉ JEAN MONNET SAINT-ÉTIENNE
PRIMES UNIVERSITÉ DE LYON

Reprise de code

La plateforme AWESOMME

05 Novembre 2024

Luc Billaud [1] Frédéric Cervenansky [1] Tiphaine Diot [1]

[1]Univ.Lyon, INSA-Lyon, Université Claude Bernard Lyon 1, UJM-Saint Etienne, CNRS, Inserm, CREATIS UMR 5220, U1206, F-69XXX, LYON, France

Plan

Plan

La plateforme AWESOMME

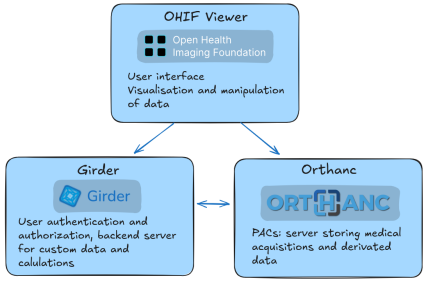
- Application Web pour la génération en masse de données expertisées en imagerie médicale

La plateforme AWESOMME

- Application Web pour la génération en masse de données expertisées en imagerie médicale
- Infrastructure basée sur 3 composants open-source

La plateforme AWESOMME

- Application Web pour la génération en masse de données expertisées en imagerie médicale
- Infrastructure basée sur 3 composants open-source
- Développée au sein du laboratoire CREATIS



Objectifs de la plateforme

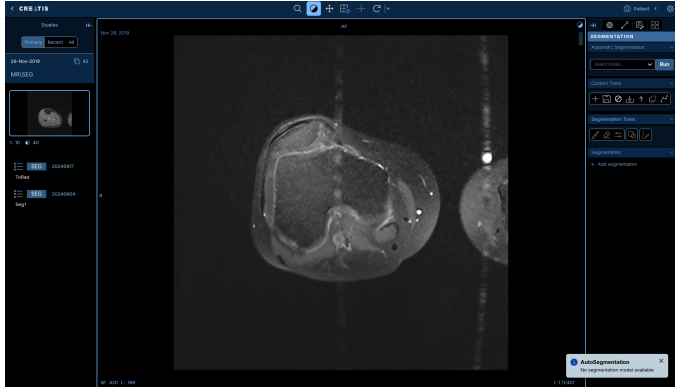
- Interface Web accessible, sécurisée et familière

Objectifs de la plateforme

- Interface Web accessible, sécurisée et familière
- Fonctionnalités d'analyse d'images et de visualisation avancée

Objectifs de la plateforme

- Interface Web accessible, sécurisée et familière
- Fonctionnalités d'analyse d'images et de visualisation avancée
- Environnement clinique



Motivations et étapes de la reprise de code

Motivations

Motivations et étapes de la reprise de code

Motivations

Motivations et étapes de la reprise de code

Motivations

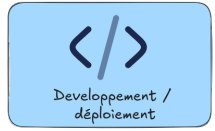
Motivations et étapes de la reprise de code

Motivations

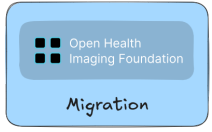
Etapes

Motivations et étapes de la reprise de code

Motivations



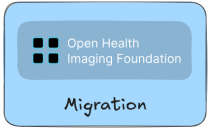
Etapes



Motivations et étapes de la reprise de code

Motivations

Etapes



Plan

Refactorisation

- Séparation claire des routes, de leurs dépendences et du code commun

Refactorisation

- Séparation claire des routes, de leurs dépendences et du code commun
- Principe de responsabilité unique

  Une classe ne doit changer que pour une seule raison

Refactorisation

- Séparation claire des routes, de leurs dépendences et du code commun
- Principe de responsabilité unique
  Une classe ne doit changer que pour une seule raison
- **Conséquences :** lisibilité et maintenabilité améliorées

Injection de dépendences

- Mise en place d'un système d'injection de dépendences avec **injector**

Injection de dépendences

- Mise en place d'un système d'injection de dépendences avec **injector**
- Gestion des cycles de vie des classes et injection automatique via le constructeur

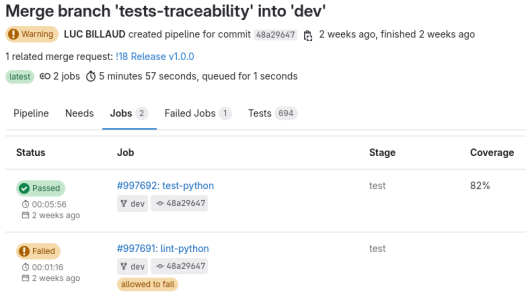
Injection de dépendences

- Mise en place d'un système d'injection de dépendences avec **injector**
- Gestion des cycles de vie des classes et injection automatique via le constructeur
- **Conséquences :** simplification de l'écriture des classes et des tests unitaires

Exemple : Injection de dépendences

```python
from girder_common.errors.group_not_found_error import GroupNotFoundError
from girder_common.services.group_service import GroupService
from injector import inject

# The GroupFetcher 's job is to fetch a group by its name and return it
# If no group with the provided name is found, it will raise a
# GroupNotFoundError


# The inject decorator tells that this class's instances can be created
# by injector with its dependencies automatically injected
@inject
class GroupFetcher:
    # This constructor can either be called directly (usually in tests)
    # or indirectly via injector and all the dependencies will be injected
    # automatically. The instance's lifecycle will also be managed by injector
    def __init__(self, group_service: GroupService) -> None:
        self._groupService = group_service

    # Our class's method. It uses the class's dependency
    def fetch_group_from_name(self, name):
        group = self._groupService.getByName(name)

        if group is None:
            raise GroupNotFoundError("Group not found")

        return group

```

Tests unitaires



- **Conséquences :** Plus d'assurance dans la justesse et la robustesse du code

Tests unitaires

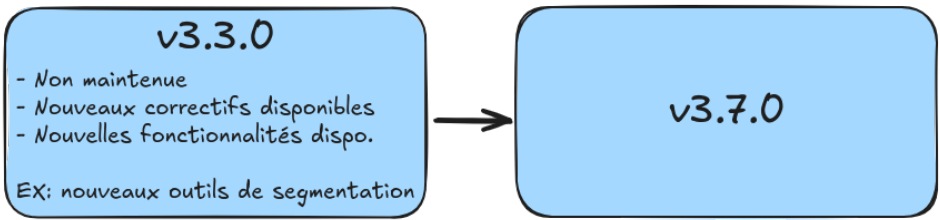
- **Conséquences :** Plus d'assurance dans la justesse et la robustesse du code
- Plus de tests fonctionnels sont nécessaires

Exemple : Tests unitaires

```python
import pytest
from unittest.mock import Mock
from girder_common.errors.group_not_found_error import GroupNotFoundError
from girder_common.utils.fetchers.group_fetcher import GroupFetcher

# The GroupFetcher 's job is to fetch a group by its name and return it
# If no group with the provided name is found, it will raise a
# GroupNotFoundError


# Here are our mocks. They mock the dependencies the GroupFetcher depends on
# We have 2 mocks for the same dependency to cover the multiple cases that can occur
# Here either a group is found or no group is found
@pytest.fixture()
def mock_group_service():
    mock = Mock()
    mock.getByName.return_value = {"_id": "test"}
    return mock


@pytest.fixture()
def mock_none_group_service():
    mock = Mock()
    mock.getByName.return_value = None
    return mock


# This function tests whether the fetch_group_from_name method as intended
# We control the dependencies and the method inputs to get a controlled result
# and we verify that it is actually returned by the method
def test_fetch_group_from_name_returns_group(mock_group_service):
    fetcher = GroupFetcher(mock_group_service)

    result = fetcher.fetch_group_from_name("myTestGroup")

    assert isinstance(result, dict)
    assert result["_id"] == "test"


# Here we test the case when no group is found and an error is raised
def test_fetch_group_from_name_raises_if_not_found(mock_none_group_service):
    fetcher = GroupFetcher(mock_none_group_service)

    with pytest.raises(GroupNotFoundError):
        fetcher.fetch_group_from_name("groupThatDoesntExist")
```

Plan

Problème de version de OHIF



Problème avec la migration de OHIF

Problème

- Nous avons modifié le core de OHIF pour l'adapter à nos besoins

Problème avec la migration de OHIF

Problème

- Nous avons modifié le core de OHIF pour l'adapter à nos besoins
- Le core de OHIF a été grandement modifié entre les versions v3.3 et v3.7

Problème avec la migration de OHIF

Problème

- Nous avons modifié le core de OHIF pour l'adapter à nos besoins
- Le core de OHIF a été grandement modifié entre les versions v3.3 et v3.7
- Le portage de nos modifications n'est donc pas évident

Problème avec la migration de OHIF

Problème

- Nous avons modifié le core de OHIF pour l'adapter à nos besoins
- Le core de OHIF a été grandement modifié entre les versions v3.3 et v3.7
- Le portage de nos modifications n'est donc pas évident

Solution

- Extraction des modifications faites en interne dans des dossiers séparés

Problème avec la migration de OHIF

Problème

- Nous avons modifié le core de OHIF pour l'adapter à nos besoins
- Le core de OHIF a été grandement modifié entre les versions v3.3 et v3.7
- Le portage de nos modifications n'est donc pas évident

Solution

- Extraction des modifications faites en interne dans des dossiers séparés
- Réduction du code du core modifié au maximum

Problème avec la migration de OHIF

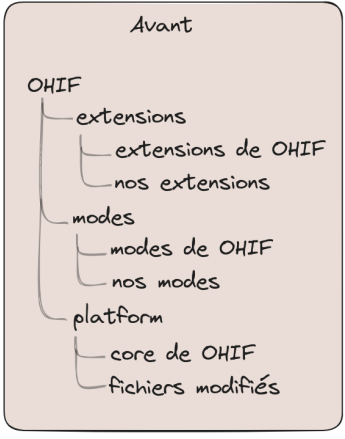
Problème

- Nous avons modifié le core de OHIF pour l'adapter à nos besoins
- Le core de OHIF a été grandement modifié entre les versions v3.3 et v3.7
- Le portage de nos modifications n'est donc pas évident

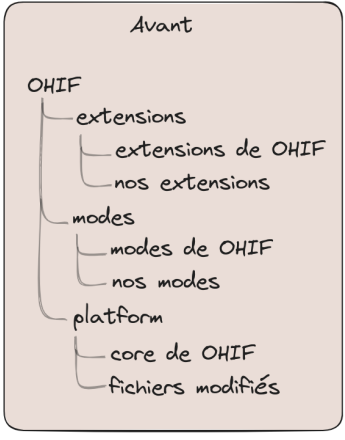
Solution

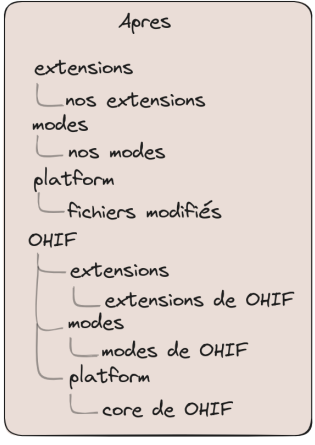
- Extraction des modifications faites en interne dans des dossiers séparés
- Réduction du code du core modifié au maximum
- Réinjection du code au lancement de l'application

Avec un schema

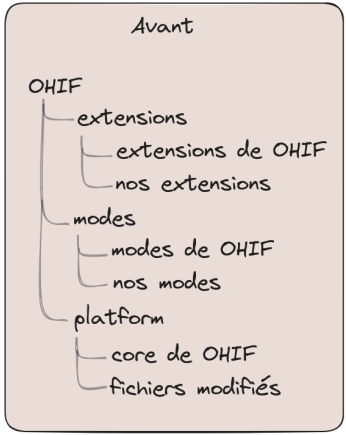


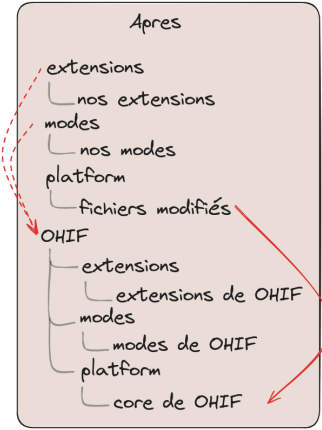
Avec un schema





Avec un schema





Conséquences

- Plus de clareté sur le code développé en interne

Conséquences

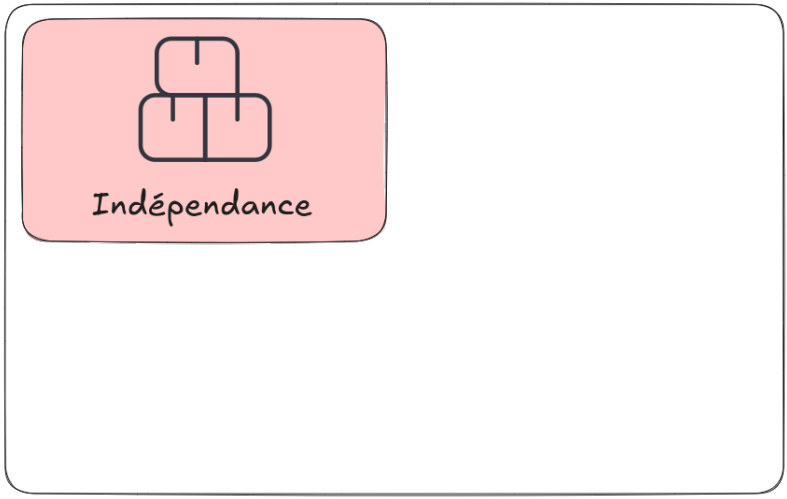
- Plus de clareté sur le code développé en interne
- Migrations simplifiées car moins d'interdépendance

Conséquences

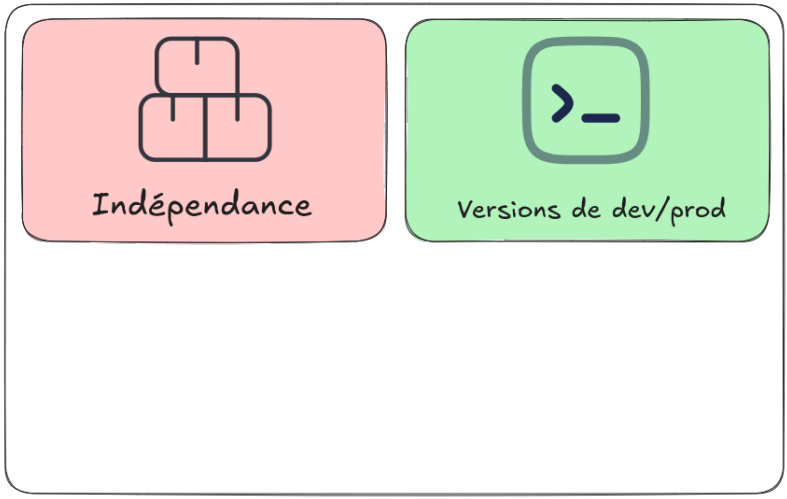
- Plus de clareté sur le code développé en interne
- Migrations simplifiées car moins d'interdépendance
- Migration vers la version v3.8.x sans accrocs

Plan

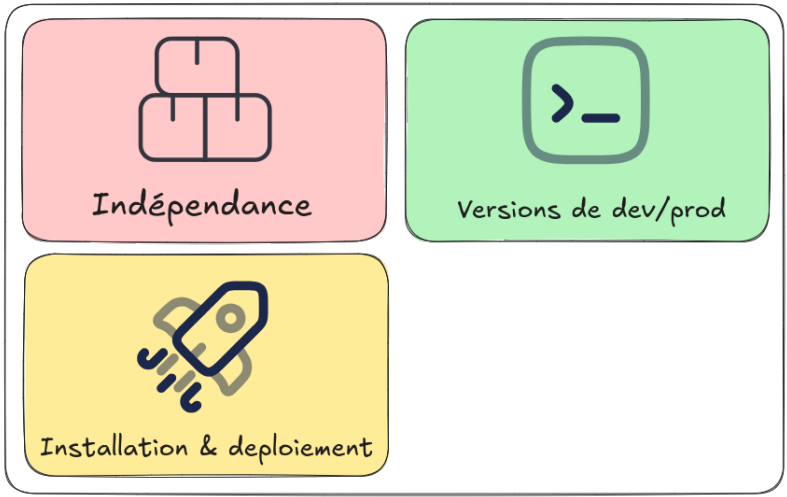
Avantages de Docker



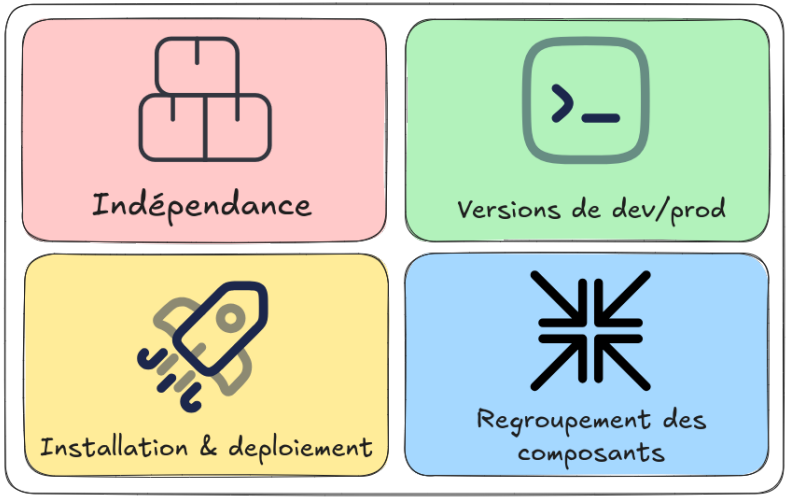
Avantages de Docker



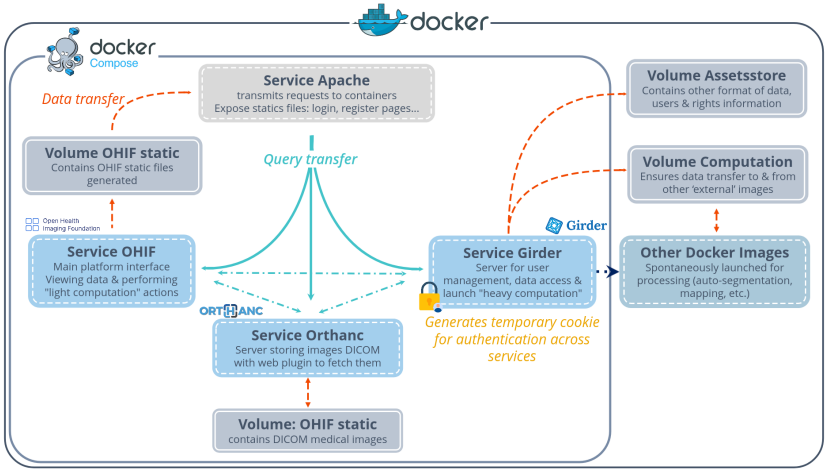
Avantages de Docker



Avantages de Docker



Nouvelle architecture du projet



Plan

Conclusion

3 étapes

- Refactorisation de Girder
  - Séparation du code
  - Injection de dépendences
  - Tests unitaires
- Migration de OHIF
  - Reorganisation du code
  - Migration
- Conteneurisation

Conclusion

3 étapes

- Refactorisation de Girder
  - Séparation du code
  - Injection de dépendences
  - Tests unitaires
- Migration de OHIF
  - Reorganisation du code
  - Migration
- Conteneurisation

Bénéfices

- Maintenabilité
- Robustesse
- Evolutions simplifiées
- Prise en main et developpement accélérés
- Deploiements simplifiés

Conclusion

Implications

- HERESP (Planification de doses - CLB)
- AVC Evualuation (120 patients- 60 experts)
- BoostData (60 patients - CLB, Lille, Paris)
- Prepanext (mode Educatif, Neurologie)
- Awesomme (130 patients, seg. aut., radiomiques, diagnostique)

Merci pour votre attention !