

La gestion de version Intégration Continue GitLab



TABLE DES MATIERES

<i>Section 1 Généralités.....</i>	<i>3</i>
1.1 Le DevOps et l'intégration continue.....	3
1.2 But du document.....	5
<i>Section 2 Présentation de GitLab.....</i>	<i>6</i>
2.1 Fonctionnalités de GitLab.....	6
2.2 Concurrents de GitLab.....	6
<i>Section 3 Planifier notre environnement.....</i>	<i>7</i>
3.1 Créer un projet.....	7
3.2 Ajouter des labels et des boards.....	7
3.3 Créer un epic.....	9
3.4 Créer une user story.....	10
3.5 Associer des issues.....	10
3.6 Créer un Sprint.....	10
<i>Section 4 Intégrer le code en continu.....</i>	<i>12</i>
4.1 Cloner le projet sur notre ordinateur en local.....	12
4.2 Activer l'intégration continue.....	14
4.2.1 Mise en place.....	14
4.2.2 Introduire une erreur.....	16
4.2.3 Créer une issue suite à ce bug.....	17
4.2.4 Créer un merge Request.....	17
<i>Section 5 Mesurer la qualité du code.....</i>	<i>19</i>
<i>Section 6 Packager pour déployer.....</i>	<i>21</i>
<i>Section 7 La livraison continue.....</i>	<i>22</i>
7.1 But.....	22
7.2 Codifier l'infrastructure avec l'Infrastructure-as-Code.....	22
7.3 Le déploiement de l'application.....	23

Section 1 Généralités

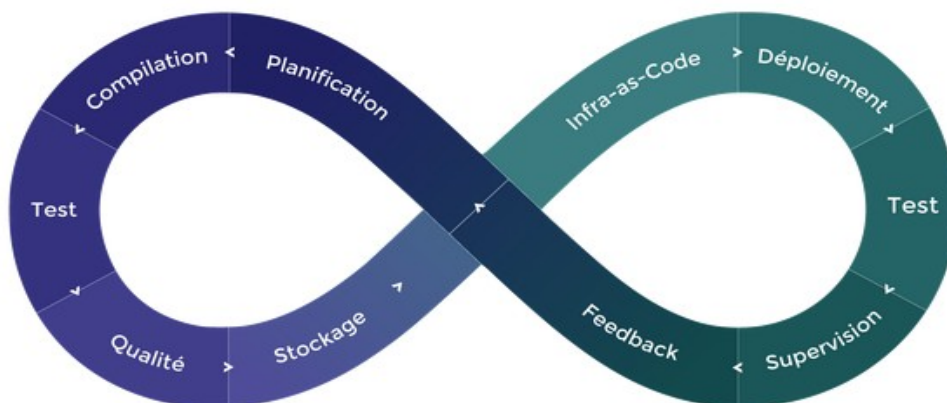
1.1 Le DevOps et l'intégration continue

L'intégration continue est une des composantes de la démarche DevOps

« Le DevOps est un ensemble de pratiques qui met l'emphase sur l'automatisation des processus entre les équipes de développement, et les équipes en charge du maintien en conditions opérationnelles de l'application développée. » (source openclassroom)

L'intégration et la livraison continues permettent de :

- Réduire le temps de dev et la mise en production
- Réduire les erreurs et les régressions
- Permettre une continuité de services des applications, car temps réduit



Cycle d'intégration et de livraison continues avec la démarche DevOps

Les 5 piliers du DevOps - acronyme CALMS: source [Openclassroom](#)

1. **Culture** : le DevOps est une affaire de culture d'entreprise entre les dev et les ops, avant d'être un ensemble d'outils ou même de pratiques métier. Il faut que les dev et les ops travaillent ensemble, aient une vision commune, soient tournés vers les mêmes objectifs.
2. **Automatisation** : tout ce qui peut être automatisé doit l'être ! Ceci se fait par des **déploiements** programmés et fréquents, des environnements créés **à la demande** et des **tests automatisés**.
3. **Lean** : dans la méthodologie DevOps, concentrez-vous sur la création de valeur et limitez le **gaspillage** de ressources. Pour cela, utilisez la **Value Stream Map** pour maîtriser votre chaîne de valeur.
4. **Mesure** : Afin de tout automatiser et tout optimiser, il faut mesurer tout ce que vous faites. Pour ça, mettez en place les **KPI** nécessaires pour superviser vos déploiements !
5. **Sharing** : Dans le DevOps, les équipes de dev et d'ops **partagent des moments et les responsabilités**. C'est le cœur de l'objectif du DevOps !

Liens utiles :

<https://openclassrooms.com/fr/courses/6093671-decouvrez-la-methodologie-devops>

<https://docs.gitlab.com/>

chaîne gitlab <https://www.youtube.com/watch?v=q5E-scBPYFA&list=PLn6POgpkIwWrRoZZXv0xf71mvT4E0QDOF&index=1>

<https://www.youtube.com/watch?v=URcMBXjlr24>

1.2 But du document

Ce document fournit un aperçu des possibilités offertes par l'environnement GitLab.

Il s'inspire et présente une petite partie du cours openclassroom

<https://openclassrooms.com/fr/courses/2035736-mettez-en-place-lintegration-et-la-livraison-continues-avec-la-demarche-devops>

Le parti pris est d'utiliser l'outil dans une démarche de gestion de projet de type Agile/Scrum. L'outil peut être configuré pour d'autres types de gestion.

Nous allons découvrir comment, au travers d'un code existant sur un github :

- Planifier notre développement.
- Compiler et intégrer notre code.
- Tester notre code.
- Mesurer la qualité de notre code.
- Gérer les livrables de notre application.

Section 2 Présentation de GitLab

2.1 Fonctionnalités de GitLab

- Dépôts git
- gestion de tickets
- organisation du travail (comme trello)
- rédaction de wiki et pages web
- intégration continue CI
- déploiement continu CD
- sécurité applicative
- API pour de nombreux services

Couvre une bonne partie des besoins DevOps

Gitlab Community edition est gratuit

2.2 Concurrents de GitLab

<https://about.gitlab.com/devops-tools/>

Section 3 Planifier notre environnement

3.1 Créer un projet

Create or import your first project

Projects help you organize your work. They contain your file repository, issues, merge requests, and so much more.

Create	Import
<p>Group name</p> <input type="text" value="imieMS2D"/>	
<p>Project name</p> <input type="text" value="spring-petclinic-microservices"/>	

GitLab is better with colleagues!



Congratulations on creating your project, you're almost there!

How about inviting a colleague or two to join you?

Username or email address

Select members or type email addresses

Select a role

[Read more](#) about role permissions

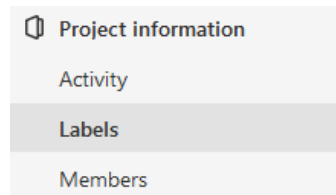
Access expiration date (optional)

Cancel

Invite

3.2 Ajouter des labels et des boards

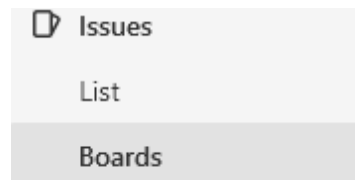
Aller dans la partie Project information / Labels



Créer les labels suivants : Epic, User Story, Bug, Ready, Rejected, High, Medium, Low, In Review, To Do, Doing

Vous avez le choix des couleurs.

Dans la partie Issues/Boards



Ajouter les listes To Do, Doing, In Review par le bouton Create list.

Créer un nouveau board par le bouton New board

Nommer ce board « Product backlog »

Title

List options

Configure which lists are shown for anyone who visits this board

- ☒ Show the Open list
- ☒ Show the Closed list

Sur la nouvelle page, après avoir cliqué sur le bouton Create Board, il faut maintenant ajouter les colonnes Low, Medium, High et Rejected afin de pouvoir classer les epics sur ce board

Nous avons maintenant tous les outils nécessaires afin de commencer notre travail sur le projet. Le workflow de développement se déroule comme suit :

1. Le product owner travaille avec les utilisateurs finaux afin de définir un **Product Backlog**. Ce Product Backlog est constitué de différentes **epics**.

2. Ces **epics** doivent faire apparaître plusieurs critères, comme par exemple des wireframes ou autres écrans définissant l'application. Une epic en high passe en sprint backlog lorsque son périmètre est délimité avec un label **Ready**, et le développement est confirmé avec le client. Ce n'est qu'à ce moment que l'epic est discutée avec l'équipe et décomposée en user stories, décrivant la feature à développer.
3. Les **user stories** doivent respecter la Definition of Ready définie plus tôt, lors du cycle de développement. Une user story n'est ready que si elle a bien été écrite, et contient tous les éléments nécessaires à son développement durant le sprint.
4. Une fois que le sprint backlog a été décomposé en user stories, nous pouvons alors commencer la planification du sprint backlog.

Pour ce cours, nous allons créer **deux issues** :

- **une epic** ;
- **une user story**.

3.3 Créer un epic

Aller dans Issue / List et choisir New Issue

Saisir ceci :

Titre

[EPIC] Gestion des vétérinaires

Description

Dans le projet *PetClinic*, il faut ajouter la gestion des vétérinaires.

Implémentation de :

- Création des vétérinaires
- Recherche des vétérinaires
- Mise à jour des vétérinaires
- Suppression des vétérinaires

Tâches à faire :

- [x] Création des vétérinaires
- [x] Recherche des vétérinaires
- [x] Mise à jour des vétérinaires
- [] Suppression des vétérinaires
- [] Supprimer les vétérinaires
- [x] Suppression de la base de données

Choisir le label Epic

Appui sur Create issue

3.4 Créer une user story

Titre

[User story] Suppression du vétérinaire

Description

En tant qu'administrateur, je dois pouvoir supprimer un vétérinaire afin que celui-ci ne s'affiche plus dans l'application

Choisir le bon label.

Lister les issues par Issues / List

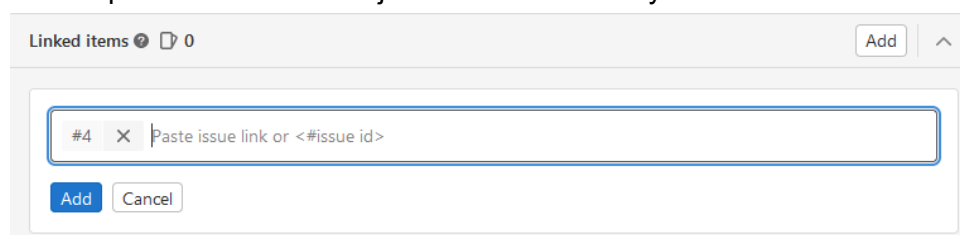
Noter le n° de la User Story

3.5 Associer des issues

On souhaite associer la user story à son epic.

Afficher l'epic.

Dans la partie Linked items ajouter le n° de la story comme ceci :



Si nous retournons maintenant sur le menu Boards, nous allons voir que sur les deux boards créés (Development et Product), dans la partie Open, nous avons nos deux issues créées précédemment

3.6 Créer un Sprint

Créer d'abord une échéance (milestone)

New Milestone

Title

Sprint 1

Start Date

2022-10-27

Due Date









2022-11-02

Clear start date

Description

Write

Preview

B I        

Description de mon 1er Sprint :

Supports [Markdown](#)

Create milestone

Cancel

Après le create Milestone, la vue permet de montrer un Burndown chart (licence premium nécessaire).

Il faut associer des issues à ce milestone. Aller dans la vue de l'issue Suppression du vétérinaire et affecter le milestone Sprint 1

Affecter une charge et simuler un avancement :

Dans le commentaire ajouter

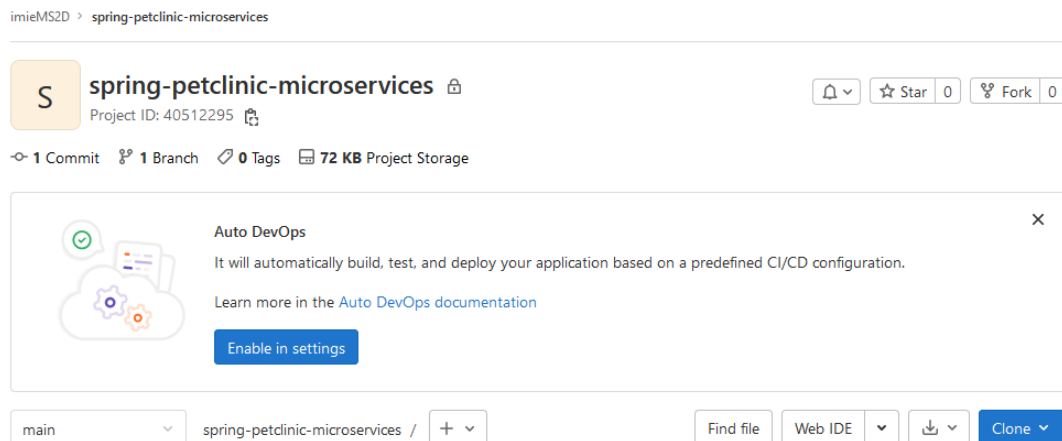
/estimate 3d

/spent 10

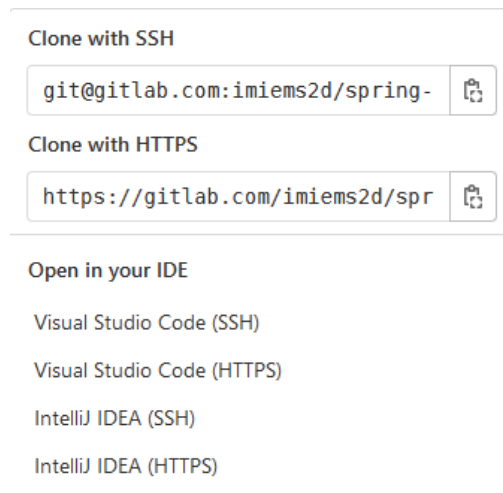
Section 4 Intégrer le code en continu

4.1 Cloner le projet sur notre ordinateur en local

Aller sur la page d'accueil gitlab du projet



Appuyer sur Clone



et copier le texte correspondant au Clone with HTTPS

Ce projet a une arborescence telle que certains chemins de fichiers sont trop longs pour Windows.

Dans une fenêtre cmd avec les droits d'administrateur taper :

```
git config --system core.longpaths true
```

Dans une autre fenêtre console, se déplacer dans un répertoire de base recevant le projet.

CI/CD Gitlab

Taper les commandes suivantes. Le texte en rouge vient de la copie précédente

```
git clone https://[votre-nom-d-utilisateur]@gitlab.com/imiems2d/spring-petclinic-  
microservices.git
```

(ancienne commande, il y a un an

```
git clone https://gitlab.com/[votre-nom-d-utilisateur]/spring-petclinic-microservices.git  
cd spring-petclinic-microservices )
```

Nous récupérons ainsi le contenu de notre projet gitlab, presque vide (seul un readme.md, **le supprimer**).

Il faut indiquer que l'on souhaite récupérer le contenu d'un autre projet, commande :

```
git remote add upstream  
https://gitlab.com/laurentgrangeau/spring-petclinic-  
microservices.git  
puis  
git pull upstream main --allow-unrelated-histories
```

Constater que le projet est récupéré en local.

Ce projet est complet, y compris la configuration pour le CI/CD :

- des fichiers dockerfile décrivent les images dockers du projet
- le fichier docker-compose.yml décrit les conteneurs à mettre en place pour l'application complète
- le fichier .gitlab-ci.yml décrit des jobs du pipeline CI/CD de GitLab

Ceci fait que le projet fonctionne directement sur GitLab. Mais pédagogiquement non intéressant.

Par conséquent **renommer le fichier .gitlab-ci.yml en .gitlab-ci.sav**

Faire un push de ce contenu vers notre gitlab. Il faut d'abord faire un commit

```
git add .  
git commit -m "commit après recup projet du web"
```

```
git push origin main  
ou
```

`git push origin HEAD:main` si la précédente commande ne fonctionne pas

Suite à ces opérations le contenu du projet doit être visible sous GitLab

4.2 Activer l'intégration continue

4.2.1 Mise en place

Il faut implémenter un « pipeline d'intégration continue », un mécanisme d'outils appelés les uns après les autres, de façon automatique.

Les étapes sont les suivantes :



Cliquer sur Build / Pipeline editor

Y placer par copier coller les parties build et test du fichier `.gitlab-ci.sav`

Voici ce que l'on doit obtenir :

CI/CD Gitlab

```
stages:
  - build
  - test

cache:
  paths:
    - .m2/repository
  key: "$CI_JOB_NAME"

build_job:
  stage: build
  image: eclipse-temurin:17-jdk-alpine
  script:
    - ./mvnw compile
      -Dhttps.protocols=TLSv1.2
      -Dmaven.repo.local=$CI_PROJECT_DIR/.m2/repository
    -
Dorg.slf4j.simpleLogger.log.org.apache.maven.cli.transfer.Slf4jMavenTransferLi
stener=WARN
    -Dorg.slf4j.simpleLogger.showDateTime=true
    -Djava.awt.headless=true
    --batch-mode --errors --fail-at-end --show-version -DinstallAtEnd=true -
DdeployAtEnd=true

test_job:
  stage: test
  image: eclipse-temurin:17-jdk-alpine
  script:
    - ./mvnw test
      -Dhttps.protocols=TLSv1.2
      -Dmaven.repo.local=$CI_PROJECT_DIR/.m2/repository
    -
Dorg.slf4j.simpleLogger.log.org.apache.maven.cli.transfer.Slf4jMavenTransferLi
stener=WARN
    -Dorg.slf4j.simpleLogger.showDateTime=true
    -Djava.awt.headless=true
    --batch-mode --errors --fail-at-end --show-version -DinstallAtEnd=true -
DdeployAtEnd=true
```

Le clic sur

A rectangular button with rounded corners, a light gray background, and a thin gray border. The text "Commit changes" is centered in a medium gray font.

Crée un commit et exécute le pipeline

Aller sur la vue des Build / Pipelines

En principe (!) le build_job et test_job doivent s'exécuter sans erreur.

Pipeline	Created by	Stages
Update .gitlab-ci.yml file #1294735453  main  fe7d63c6  		 

Puis

4.2.2 Introduire une erreur

Nous allons introduire une erreur dans la code et voir le comportement de GitLab

Dans la fenêtre console

`git pull` pour récupérer en local le projet

créer une branche pour y mettre notre modification

`git checkout -b refactor-customers`

Ensuite, **éditez le fichier** Java "`spring-petclinic-customers-service\src\main\java\org\springframework\samples\petclinic\customers\CustomersServiceApplication.java`"

et introduire une erreur, un `' ;'` manquant par exemple.

Faire un commit

`git commit -a -m "Refactorisation du code des clients"`

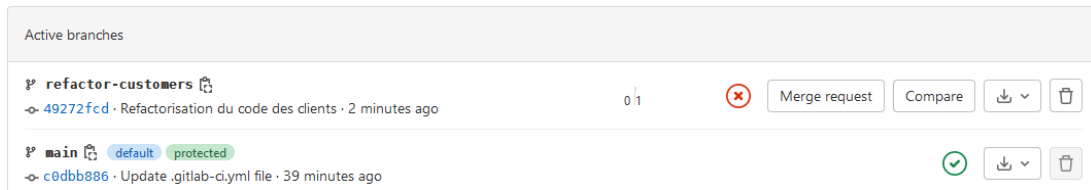
Mettre à jour GitLab

`git push origin refactor-customers`

Laisser le temps d'exécution du pipeline.

Regarder le résultat sur GitLab, vue des branches

CI/CD Gitlab



La branche refactor-customers est vue en échec, croix rouge.

Cliquer sur la croix rouge puis Build_job, la page montre les erreurs.

4.2.3 Créer une issue suite à ce bug

Dans l'écran précédent qui montre le listing avec erreur, clic sur New Issue

Laisser les champs remplis par défaut, choisir un responsable (assignee), milestone, labels Bug et ToDo

Aller dans l'issue et constater que l'on retrouve facilement les traces de l'erreur.

Le board Development montre cette issue

4.2.4 Créer un merge Request

Pour corriger ce bug automatiquement, nous allons créer une merge request, c'est-à-dire demander à commiter les changements sur notre branche dans la branche principale main.

Pour ce faire, il faut aller dans le menu Merge Requests et cliquer sur New Merge Request.

Saisir comme suit :

New merge request

Source branch	Target branch
<div>imiems2d/spring-petclinic-micro... refactor-customers</div>	<div>imiems2d/spring-petclinic-micro... master</div>
<div> Refactorisation du code des clients Daniel ROCHES authored 25 minutes ago 49272fcd </div>	<div> commit après recup projet du web albert authored 4 hours ago 351467ed </div>
<div>Compare branches and continue</div>	

Appui sur Compare branch.

Dans la vue qui suit, affecter milestone à Sprint 1, Description à Close #no bug

CI/CD Gitlab

Corriger le bug depuis votre poste, hors GitLab.

Puis

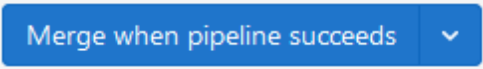
```
git commit -a -m "Correction du bug client"
```

```
git push origin refactor-customers
```

Vous allez constater qu'une fois le code pushé sur la branche refactor-customers, le pipeline de build se lance afin de vérifier que le code que nous avons envoyé fonctionne.

Une fois que le pipeline s'est terminé en succès, nous pouvons alors enlever le statut Work in progress en cliquant sur le bouton Resolve WIP status, pour ensuite cliquer sur le bouton Merge

Fait automatiquement si le choix suivant avait été fait dans le Merge request



Merge when pipeline succeeds

Section 5 Mesurer la qualité du code

Nous implémentons maintenant l'analyse de la qualité de code.

Depuis votre poste (pas GitLab) modifier gitlab-ci.yml

ajouter – quality dans stages

ajouter ceci en bas du fichier (copier / coller depuis .gitlab-ci.sav)

```
code_quality_job:
  stage: quality
  allow_failure: true
  image: docker:stable
  services:
    - docker:dind
  script:
    - mkdir codequality-results
    - docker run
      --env CODECLIMATE_CODE="$PWD"
      --volume "$PWD":/code
      --volume /var/run/docker.sock:/var/run/docker.sock
      --volume /tmp/cc:/tmp/cc
      codeclimate/codeclimate analyze -f html > ./codequality-results/index.html
  artifacts:
    paths:
      - codequality-results/
```

Réaliser un commit puis un push.

Le pipeline se met en route côté GitLab.

Il y a 2 façons de voir le résultat :

Le résultat de l'analyse de code est visible depuis la liste des pipelines



Il faut cliquer sur



puis récupérer le .zip

regarder le fichier index.html

CI/CD Gitlab

ou aller sur la vue du Job et clic sur Browse

Job artifacts

These artifacts are the latest. They will not be deleted (even if expired) until newer artifacts are available.

Keep

Download

Browse

Section 6 Packager pour déployer

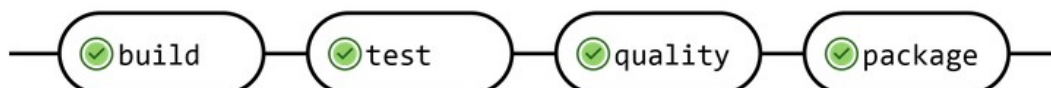
Il est possible d'étendre le pipeline pour y inclure de déploiement.

Dans gitlab-ci.yml

- ajouter – package dans stages
- ajouter le code suivant en fin de fichier .gitlab-ci.yml

```
package_job:
  stage: package
  image: eclipse-temurin:17-jdk-alpine
  services:
    - docker:dind
  variables:
    DOCKER_HOST: tcp://docker:2375
  script:
    - apk add --no-cache docker-cli-compose
    - docker login -u gitlab-ci-token -p $CI_JOB_TOKEN $CI_REGISTRY
    - ./mvnw install -PbuildDocker -DskipTests=true -DpushImage
      -Dhttps.protocols=TLSv1.2
      -Dmaven.repo.local=$CI_PROJECT_DIR/.m2/repository
    -
Dorg.slf4j.simpleLogger.log.org.apache.maven.cli.transfer.Slf4jMavenTransferLi
stener=WARN
-Dorg.slf4j.simpleLogger.showDateTime=true
-Djava.awt.headless=true
--batch-mode --errors --fail-at-end --show-version -DinstallAtEnd=true -
DdeployAtEnd=true
    - docker compose build
    - docker compose push
```

Nous avons donc un projet GitLab qui implémente le pipeline suivant



D'autres fonctionnalités sont expliquées sur le site openclassrom.

Section 7 La livraison continue

7.1 But

Nous allons voir comment livrer votre code en continu pour le mettre en production rapidement, et de manière fiable.

La **livraison continue** est une discipline où l'application est construite de manière à pouvoir être mise en production à n'importe quel moment.

Nous allons mettre en place **5 étapes** :

1. La codification de l'infrastructure avec l'**Infrastructure-as-Code**.
2. Le **déploiement** de votre application.
3. Le **test** de votre application en environnement de test.
4. La **supervision** de l'application.
5. La mise en place de **notifications** d'alerte.

7.2 Codifier l'infrastructure avec l'Infrastructure-as-Code

L'**Infrastructure-as-Code** est une pratique qui consiste à **décrire une infrastructure avec du code**. Ce code est alors stocké avec le code de l'application, et fait partie intégrante de cette dernière.

En fait ici c'est Docker qui assure cette fonction avec le fichier existant `docker/Dockerfile`

L'application va être déployée avec Docker Compose en utilisant le fichier `docker-compose.yml`

7.3 Le déploiement de l'application

Plusieurs stratégies de déploiement existent. Le lien suivant en décrit quelques unes :

<https://www.akuiteo.com/blog/les-strategies-de-dploiement-dun-logiciel-erp>

Pour notre application le but est ici de déployer l'application sur Play-With-Docker. Play-With-Docker permet d'utiliser des containers Docker et de les lancer.

La démarche est la suivante :

Aller sur le site <https://labs.play-with-docker.com/>

Se connecter. Une session de 4 heures est lancée.

Clic sur



Choisir

3 Managers and 2 Workers



Dans la clé SSH proposée copier la partie en bleu :

Memory

CPU

SSH

ssh **ip172-19-0-59-chpndp89ec4g00esot70**@direct.labs.play-with-d



Dans .gitlab-ci.yml :

Ajouter après le bloc stages, avec la chaîne précédemment copiée:

variables:

PLAYWD: ip172-19-0-114-cg2s48o1k7jg00dhgtb0

Ajouter le stage

- deploy

CI/CD Gitlab

Ajouter en fin de fichier :

`deploy_staging_job:`

`stage: deploy`

`image: alpine`

`script:`

- `apk add --no-cache docker-cli-compose`
- `export DOCKER_HOST=tcp://$PLAYWD.direct.labs.play-with-docker.com:2375`
- `docker compose down`
- `docker compose up -d`

`environment:`

`name: staging`

`url: http://$PLAYWD-8080.direct.labs.play-with-docker.com`

ATTENTION : vous pourriez avoir l'idée de commenter des stages secondaires ... comme quality. Ne le faites pas car des données de sortie de ce stage sont utilisés dans deploy.

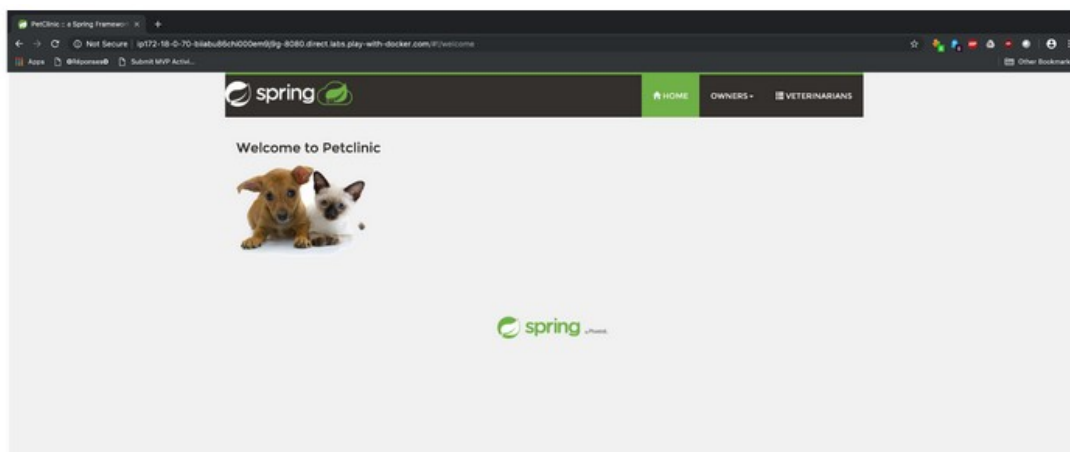
Archiver par commit et push vers GitLab

Dans GitLab, si tout s'est bien passé, dans Deployments / Environments doit apparaître la ligne

> staging

Open Stop

Un clic sur Open lance l'application



Quand la session de 4 heures expire, une nouvelle session produit de nouveaux identifiants.

Ils peuvent être passés au dernier moment sur GitLab par :

CI/CD Gitlab

clic sur

Run pipeline

Saisir le nom de la variable et son contenu

Variables

Variable	
PLAYWD	ip172-19-0-59-chpndp89ec4g00esot70

Clic sur Run pipeline

C'est reparti pour 4 heures ...

Avertissement :

A cette date (17/05/24) une régression fait que le stage deploy échoue car le fichier docker-compose.yml utilise une fonction « dockerize » qui ne fait pas partie de l'image de base spring-community ...

Correction à venir.