Midi de la bidouille
DOCKER

Louise-Amélie Schmitt & David Sherman

EQUIPE
PLEIADE
BORDEAUX SUD-OUEST

2017-05-23

OUTLINE

1. Prise en main

   images ; docker ; Dockerfile

2. Cas d'utilisation

   Exemples : Mimoza, CI GitLab & Jenkins

3. Retour sur l'expérience

   pérenne vs. ephémère ; service complet ; variants

2. Cas d'utilisation (1)

Cas 1: Conteneuriser une app classique

Acteur principal : Développeur

Préconditions :

- Disposer d'un logiciel indépendant (ligne de commande ou serveur mono-port)
- Disposer d'un dépôt Git pour le logiciel

Scénario de succès :

1. Identifier le processus principal et définir le ou les entrypoint
2. Identifier les ports IP qui doivent être exposés
3. identifier les volumes qui peuvent être exposés
4. Écrire un Dockerfile qui compile et installe le logiciel dans le conteneur
5. Installer le Dockerfile à la racine du dépôt Git

Extensions :

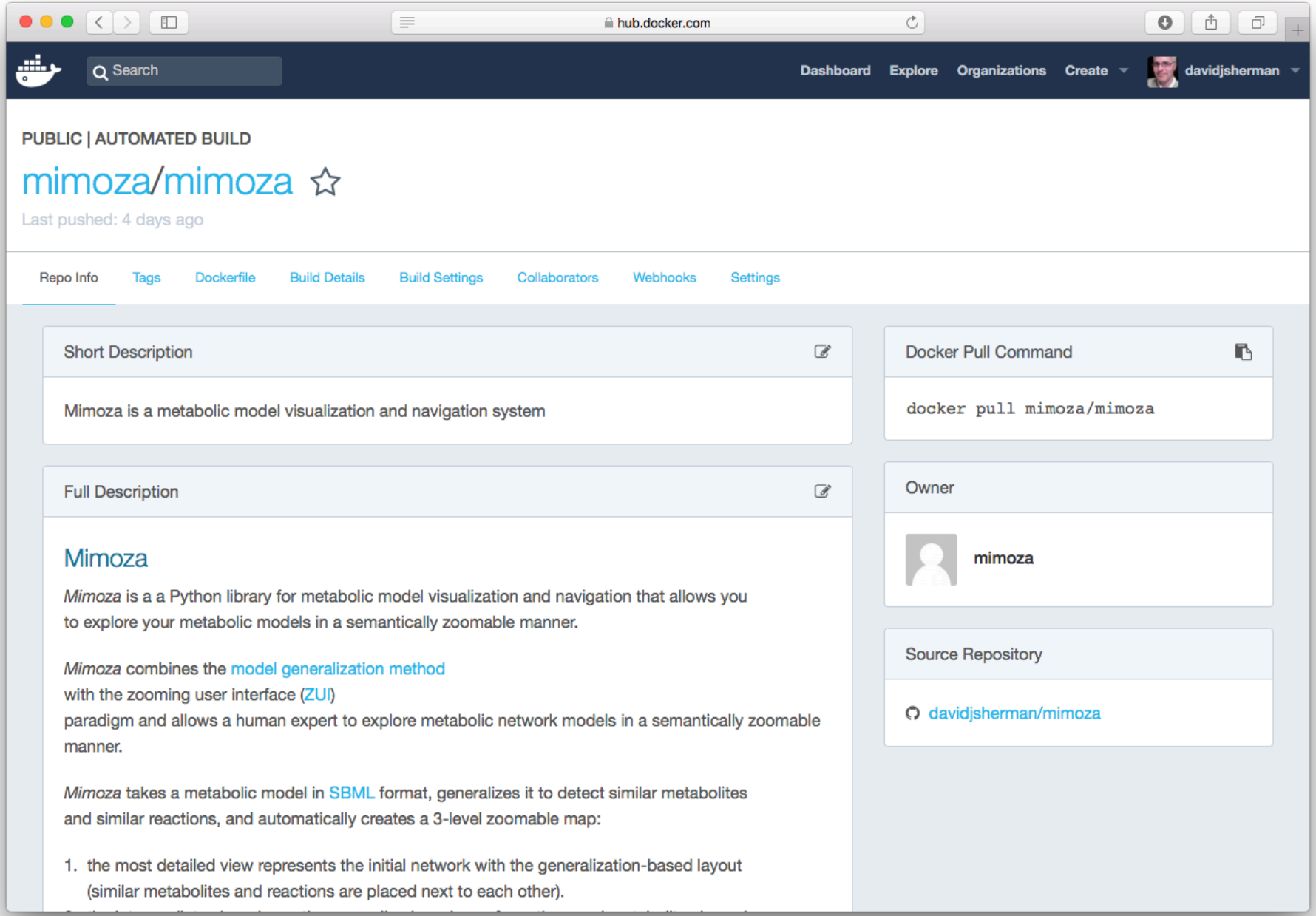
(1.) Si plusieurs processus principaux, utiliser *dumb-init* ou *phusion/baseimage-docker*

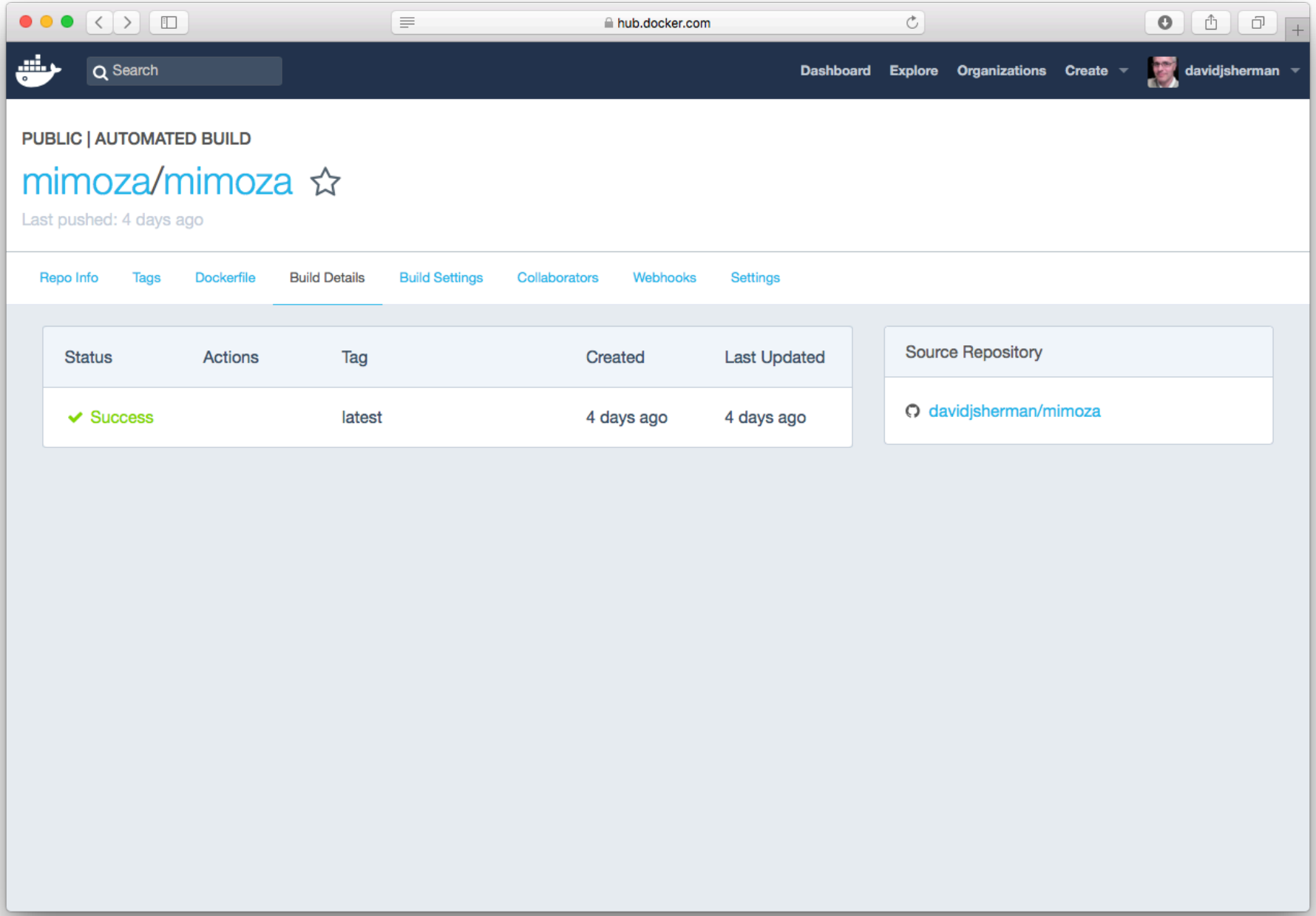
(4.) S'il est difficile de nettoyer l'environnement, utiliser un *build container*

(5.) Éventuellement, automatiser la construction du conteneur

Dockerfile

```
# Docker image for Mimoza, the metabolic model generalizer
# The entrypoint runs Mimoza with command line arguments; provide a volume for input data and results
#   docker run --volume $JOBDIR:/tmp mimoza:latest --model $MODEL.xml

FROM ubuntu

# Python3 needs an UTF-8 locale, http://bugs.python.org/issue19846
ENV LANG C.UTF-8

# Build-time environmental variable so that apt doesn't complain
ARG DEBIAN_FRONTEND=noninteractive

# Create the development environment
RUN apt update && \
    apt install -y git python3-setuptools python3-dev python3-pip python3
RUN pip3 install --no-cache-dir --upgrade pip

# Copy in the Mimoza source code then install
COPY . /app
WORKDIR /app
RUN pip3 install --no-cache-dir .

# Clean up the development environment to reduce image size
RUN apt remove -y build-essential python3.5-dev && \
    apt autoremove –y

# Mimoza jobs will be run in /tmp by a nonprivileged user (allgo.inria.fr)
RUN useradd mimozaUSER mimoza
WORKDIR /tmp

# The entrypoint runs Mimoza with command line arguments
ENTRYPOINT ["/usr/bin/python3", "/app/mimoza.py"]
```


hub.docker.com
Search
Dashboard
Explore
Organizations
Create
davidjsherman
PUBLIC | AUTOMATED BUILD
mimoza/mimoza
Last pushed: 4 days ago
Repo Info
Tags
Dockerfile
Build Details
Build Settings
Collaborators
Webhooks
Settings
Short Description
Mimoza is a metabolic model visualization and navigation system
Full Description
Mimoza
Mimoza is a a Python library for metabolic model visualization and navigation that allows you to explore your metabolic models in a semantically zoomable manner.
Mimoza combines the model generalization method with the zooming user interface (ZUI) paradigm and allows a human expert to explore metabolic network models in a semantically zoomable manner.
Mimoza takes a metabolic model in SBML format, generalizes it to detect similar metabolites and similar reactions, and automatically creates a 3-level zoomable map:
1. the most detailed view represents the initial network with the generalization-based layout (similar metabolites and reactions are placed next to each other).
Docker Pull Command
docker pull mimoza/mimoza
Owner
mimoza
Source Repository
davidjsherman/mimoza


hub.docker.com
Search
Dashboard
Explore
Organizations
Create
davidjsherman
PUBLIC | AUTOMATED BUILD
mimoza/mimoza
Last pushed: 4 days ago
Repo Info
Tags
Dockerfile
Build Details
Build Settings
Collaborators
Webhooks
Settings
Status
Actions
Tag
Created
Last Updated
Success
latest
4 days ago
4 days ago
Source Repository
davidjsherman/mimoza

2. Cas d'utilisation (2)

Cas 2 : Améliorer l'intégration continue

Acteur principal : Développeur

Préconditions :

- Disposer d'une définition de pipeline Jenkins ou GitLab runner
- Disposer d'un dépôt Git pour le logiciel

Scénario de succès :

1. Identifier l'environnement pour chaque étape dans le pipeline
2. Choisir ou définir un conteneur pour chaque environnement
3. Mettre les images de conteneurs à disposition du système CI
4. Réécrire le pipeline pour se servir des conteneurs à la place des *build nodes*

Extensions :

(2.) Il est possible que certains services pérennes doivent être conteneurisés (par exemple, une base de données pour les tests)

(2.) Éviter de faire de l'installation ou la configuration à la volée : l'idée est que l'environnement de l'étape est **immuable**

(3.) Soit sur Docker Hub, soit dans le dépôt du système CI

```
stage('Package') {
    // Packages are only built for the master branch
    when {
        expression {
            return env.BRANCH == 'master'
        }
    }
    steps {
        parallel (
            "debian-pack" : {
                node('docker') {
                    script {
                        docker.image('aseba/buildfarm').inside {
                            unstash 'source'
                            sh '''
                                (cd externals/dashel && debuild -i -us -uc -b && sudo dpkg -i ../libdashel*.deb)
                                (cd externals/enki && debuild -i -us -uc -b && sudo dpkg -i ../libenki*.deb)
                                (cd aseba && debuild -i -us -uc -b)
                                exit 0
                            '''
                        }
                    }
                    archiveArtifacts artifacts: 'aseba*.deb', fingerprint: true, onlyIfSuccessful: true
                }
            },
            "macos-pack" : {
                node('macos') {
                    unstash 'build-aseba-macos'
                    git branch: 'inherit-env', url: 'https://github.com/davidjsherman/aseba-osx.git'
                    ...
```

.gitlab-ci.yml

```
# Build docker image and upload to repository

# variables:
#  CI_REGISTRY_IMAGE: "mimoza/mimoza”
#  other variables are supplied by GitLab project configuration

# Official docker image.
image: docker:latest

services:
  - docker:dind

# see https://docs.gitlab.com/ce/ci/yaml/README.html for all available options

before_script:
  - echo "Before script section”

after_script:
  - echo "After script section”

build:
  stage: build
  script:
    - export IMAGE_TAG=$(echo -en $CI_BUILD_REF_NAME | tr -c '[:alnum:]_.-' '-')
    - echo "Building $IMAGE_TAG”
    - docker login -u ”$CI_BUILD_USER" -p "$CI_BUILD_TOKEN" $CI_REGISTRY
    - docker build --pull -t "$CI_REGISTRY_IMAGE:$IMAGE_TAG" .
    - docker push "$CI_REGISTRY_IMAGE:$IMAGE_TAG"
```

2. Cas d'utilisation (3)

Cas 3 : Refactoriser en microservices

Acteur principal : Architecte logiciel

Préconditions :

- Disposer d'un logiciel monolithique

Scénario de succès :

1. Identifier les services et périmètres de responsabilité
2. Identifier ou définir les conteneurs pour ces services ⚠
3. Réécrire les composants logiciels en services indépendants ⚠
4. Réécrire les appels aux services pour utiliser les conteneurs ⚠
5. Définir une orchestration des conteneurs (Docker Compose, Kubernetes)

Extensions: