Déploiement d'applications sur PLMshift

Philippe Depouilly
Laurent Facq
Sandrine Layrisse

Journées Mathrice, IHP, mars 2019

mathrice

Plan

mathrice

Introduction

Les objectifs

- Mettre à disposition de la communauté une ressource numérique pour déployer des applications web le plus simplement possible
- Sans les prérequis de l'administration système
- Permettre la co-construction de services communs
- Faciliter le passage du savoir et des compétences

La solution

- Openshift intégré dans la PLM = PLMshift

mathrice

La solution : Openshift

Un système élégant de mise en ligne de services qui permet de déployer des applications sur un principe simple de réservation de ressources (CPU, mémoire, disque, réseau, outillage de contrôle, etc.)

- basés sur des containers de type Docker
- orchestrés par kubernetes (K8s)
- avec des mécanismes intégrés de sécurité (selinux, cgroups, segmentation réseau, etc.)
- avec de la réplication (scaling), du redéploiement à la demande (hook), de la redondance, de la reprise d'activité, ...

mathrice

Eléments de base d'Openshift

Système basé sur docker qui s'appuie sur

- des images
  - une image = une arborescence système (récupérable par un docker pull)
  - cette image est réputée immutable (ne peut pas et ne doit pas être modifiée dans le temps)
  - hébergées dans une bibliothèque d'images : registry (OpenShift offre un registry interne qui va offrir une fonctionnalité d'*Image Streams*)
- des containers
  - un container = un environnement système isolé lancé par la commande d'exécution d'une image (docker run nom_image)
  - un container = un processus
- des déploiements
  - un déploiement = une application à base d'un ou plusieurs containers ayant des besoins de communication inter-processus ou du stockage pérenne

mathrice

Eléments de base d'Openshift

Système qui exécute des PODs

- un pod = un environnement d'exécution de container(s) docker
- un pod :
  - est constitué d'une (*) ou plusieurs instances d'un container applicatif
  - accède à des ressources de stockage
  - a un réseau IP unique (*)
  - possède des options qui régissent l'exécution du ou des containers

(*) à minima

mathrice

Eléments de base d'Openshift

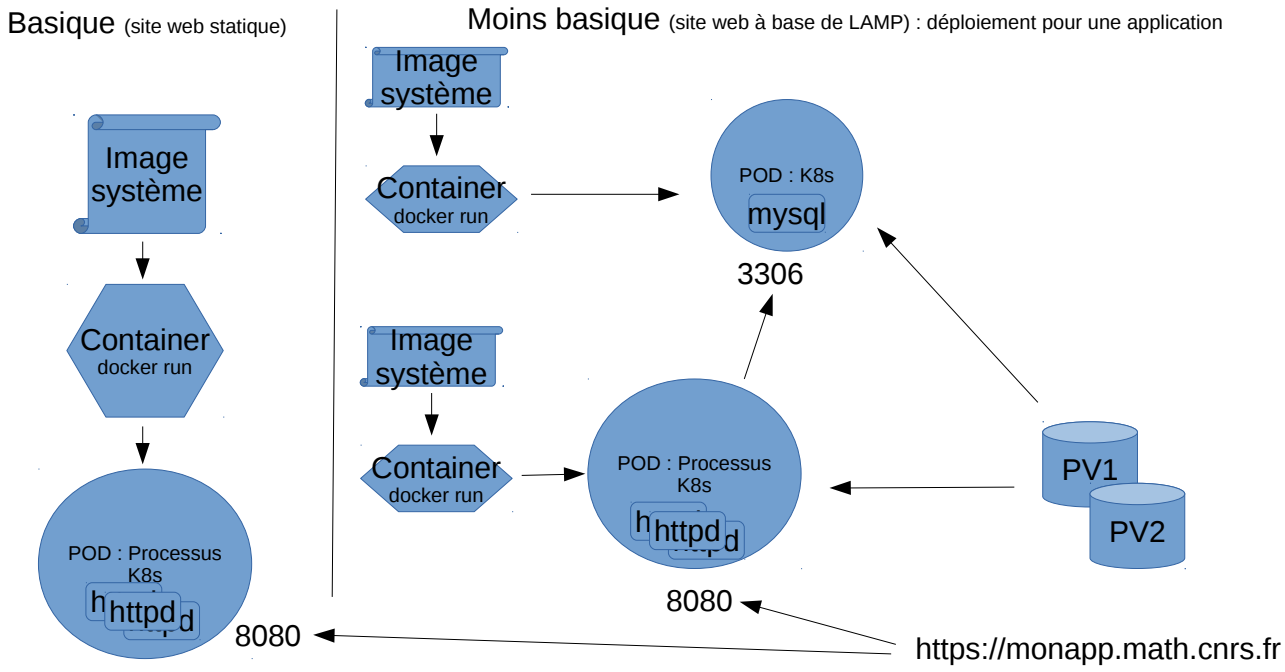
Et qui offre autour des PODs

- une connectivé réseau (essentiellement en HTTPS) : la route
- des interconnexions réseau entre PODs : les services (un POD apache se connecte au service mysql d'un POD DB)
- du stockage : PVC (Persistent Volume Claim), un point de montage disque résilient (GlusterFS) à la demande
- des cycles de vie d'images de containers (créer une image qui fusionne un serveur HTTP avec un repository GIT contenant des pages HTML)

mathrice

Eléments de base - schéma de principe général

Schéma de principe général d'openshift : les déploiements



Eléments de base - TP1

Instancier un POD Apache

- Commencez par créer un projet avec "Create Project"
- Cliquez sur "Browse catalog", puis sur "Apache HTTP Server (httpd)"
- Lisez la description de cette application puis cliquez sur "next"
- Indiquer une source de données (vous pouvez utiliser l'exemple fourni)
- Donner un nom à votre instance = Application Name
- Cliquez sur "advanced options"
- Voyez comment sécuriser la route (en cliquant sur Secure Route)

Quand l'application est en ligne, découvrez les menus de gauche de l'application et le menu Edit ou Edit Yaml du "Deployment Config"

mathrice

Eléments de base - Cycle de vie et tuning d'une application

- Une application est décrite par un fichier au format JSON ou YAML
- Scaling : nombre de replicas ou auto-scaler
  - Menu "Applications - Deployments - Configuration"
- Filtrage : Règles "egress" comparables à des iptables en OUTPUT
- Hook : des triggers pour relancer l'appli avec de nouvelles données, ou suite aux mises à jour d'un élément du déploiement
  - Menu "Builds - Builds - appli-img - actions - edit - show advanced options - Triggers"
- Opérations de maintenance : suppression des inutiles, occupation quota, sauvegardes restorations

mathrice

Structure d'une application - vue globale

Une application est l'instanciation dans le temps d'un ou plusieurs POD avec

- des services réseaux,
- du stockage persistant,
- des services de contrôle (readyness probe, liveness probe, crons, etc.)
- des mesures (metrics)
- etc.

Openshift instancie ces objects à travers un fichier déclaratif qui s'enrichit des ressources allouées et est rejouable.

mathrice

Structure d'une application - vue globale

```
$ oc project mon_projet
$ oc get dc,is,bc,services,routes,pvc -o json
{
    "apiVersion": "v1",
    "items": [
        {
            "apiVersion": "v1",
            "kind": "DeploymentConfig",
            ...
        },
         {
             "apiVersion": "v1",
             "kind": "ImageStreamTag",
             ...
        },
        {
             "apiVersion": "v1",
             "kind": "BuildConfig",
             ...
        },
        {
             "apiVersion": "v1",
             "kind": "Services",
             ...
        },
        {
             "apiVersion": "v1",
             "kind": "Routes",
             ...
        },
        {
             "apiVersion": "v1",
             "kind": "PersistentVolumeClaim",
             ...
        }
    ]
}
```

<==== Définition du contexte de déploiement d'un POD
basé sur l'image d'un container exposant un port réseau et
utilisant un stockage persistant

<==== Registry contenant l'image du container à jour

<==== Instructions de fabrication de l'image du container

<==== Définition d'un port réseau de service du POD

<==== Accès Internet à ce service depuis une route
sous la forme https://....apps.math.cnrs.fr

<==== Demande d'un stockage persistant pour le POD

Structure d'une application - le DeployementConfig

```
{
  "kind": "DeploymentConfig",
  "apiVersion": "v1",
  "metadata": {
    "name": "limesurvey",
    "labels": {
      "app": "monapp"
    }
  },
  "spec": {
    "strategy": {
      "type": "Recreate"
    },
    "triggers": [
      {
        "type": "ConfigChange"
      },
      {
        "type": "ImageChange",
        "imageChangeParams": {
          "automatic": true,
          "containerNames": [
            "limesurvey"
          ],
          "from": {
            "kind": "ImageStreamTag",
            "name": "monlamp:latest"
          }
        }
      }
    ],
    "replicas": 1,
    "selector": {
      "app": "monapp",
      "deploymentconfig": "limesurvey"
    },
```

```
    "template": {
      "metadata": {
        "labels": {
          "app": "monapp",
          "deploymentconfig": "limesurvey"
        }
      },
      "spec": {
        "volumes": [
          {
            "name": "data",
            "persistentVolumeClaim": {
              "claimName": "limesurvey-data"
            }
          }
        ],
        "containers": [
          {
            "name": "limesurvey",
            "image": "monlamp",
            "ports": [
              {
                "containerPort": 8080,
                "protocol": "TCP"
              }
            ],
            "resources": {
              "limits": {
                "memory": "1Gi"
              }
            },
            "volumeMounts": [
              {
                "name": "data",
                "mountPath": "/opt/app-root/src/livedatas"
              }
            ]
```

Structure d'une application - le BuildConfig

```
{
  "kind": "BuildConfig",
  "apiVersion": "v1",
  "metadata": {
    "name": "monlamp",
    "labels": {  "app": "monapp"    }
  },
  "spec": {
    "triggers": [
      { "type": "ConfigChange" },
      { "type": "ImageChange" }
    ],
    "source": {
      "type": "Git",
      "git": {
        "uri": "https://github.com/LimeSurvey/LimeSurvey.git",
        "ref": "master"
      }
    },
    "strategy": {
      "type": "Source",
      "sourceStrategy": {
        "from": {
          "kind": "ImageStreamTag",
          "namespace": "openshift",
          "name": "php:7.1"
        }
      }
    },
    "output": {
      "to": {
        "kind": "ImageStreamTag",
        "name": "monlamp:latest"
      }
    }
  }
}
```

```
{
      "kind": "ImageStream",
      "apiVersion": "v1",
      "metadata": {
        "name": "monlamp",
        "labels": {
          "app": "monapp"
        }
      }
    },
```

Structure d'une application - les fonctionnalités

```
{
        "kind": "Service",
        "apiVersion": "v1",
        "metadata": {
          "name": "web",
          "labels": {
            "app": "monapp"
          }
        },
        "spec": {
          "ports": [
            {
              "name": "8080-tcp",
              "protocol": "TCP",
              "port": 8080,
              "targetPort": 8080
            }
          ],
          "selector": {
            "app": "monapp",
            "deploymentconfig": "limesurvey"
          }
        }
      },
      {
        "kind": "Route",
        "apiVersion": "v1",
        "metadata": {
          "name": "https-route",
          "labels": {
            "app": "monapp"
          }
        },
```

```
    (suite de Route...)
    "spec": {
      "host": "",
      "to": {
        "kind": "Service",
        "name": "web",
        "weight": 100
      },
      "port": {
        "targetPort": 8080
      },
      "tls": {
        "termination": "edge",
        "insecureEdgeTerminationPolicy": "Allow"
      }
    }
  },
{
    "kind": "PersistentVolumeClaim",
    "apiVersion": "v1",
    "metadata": {
      "name": "lamp-data",
      "labels": {
        "app": "monapp"
      }
    },
    "spec": {
      "accessModes": [
        "ReadWriteMany"
      ],
      "resources": {
        "requests": {
          "storage": "1Gi"
        }
      }
    }
  }
```

Développer des applications

Développer des applications = assembler des briques, réclamer des ressources, interconnecter des services

A l'aide du CLI Openshift (commande oc), il est possible de construire une application très simplement

```
oc new-project un_nom_unique

# une application LAMP du catalogue
oc new-app cakephp-mysql-example

# La même depuis le dépôt GitHub source
oc new-app https://github.com/sclorg/cakephp-ex

# Une application LAMP à la main
oc new-app  php  mysql -e MYSQL_USER=moi -e MYSQL_DATABASE=base -e MYSQL_PASSWORD=moi
```

Lire impérativement la documentation
https://docs.openshift.com/container-platform/latest/dev_guide/application_lifecycle/new_app.html

Développer des applications

Développer des applications = personnaliser des images (format docker) pour un déploiement

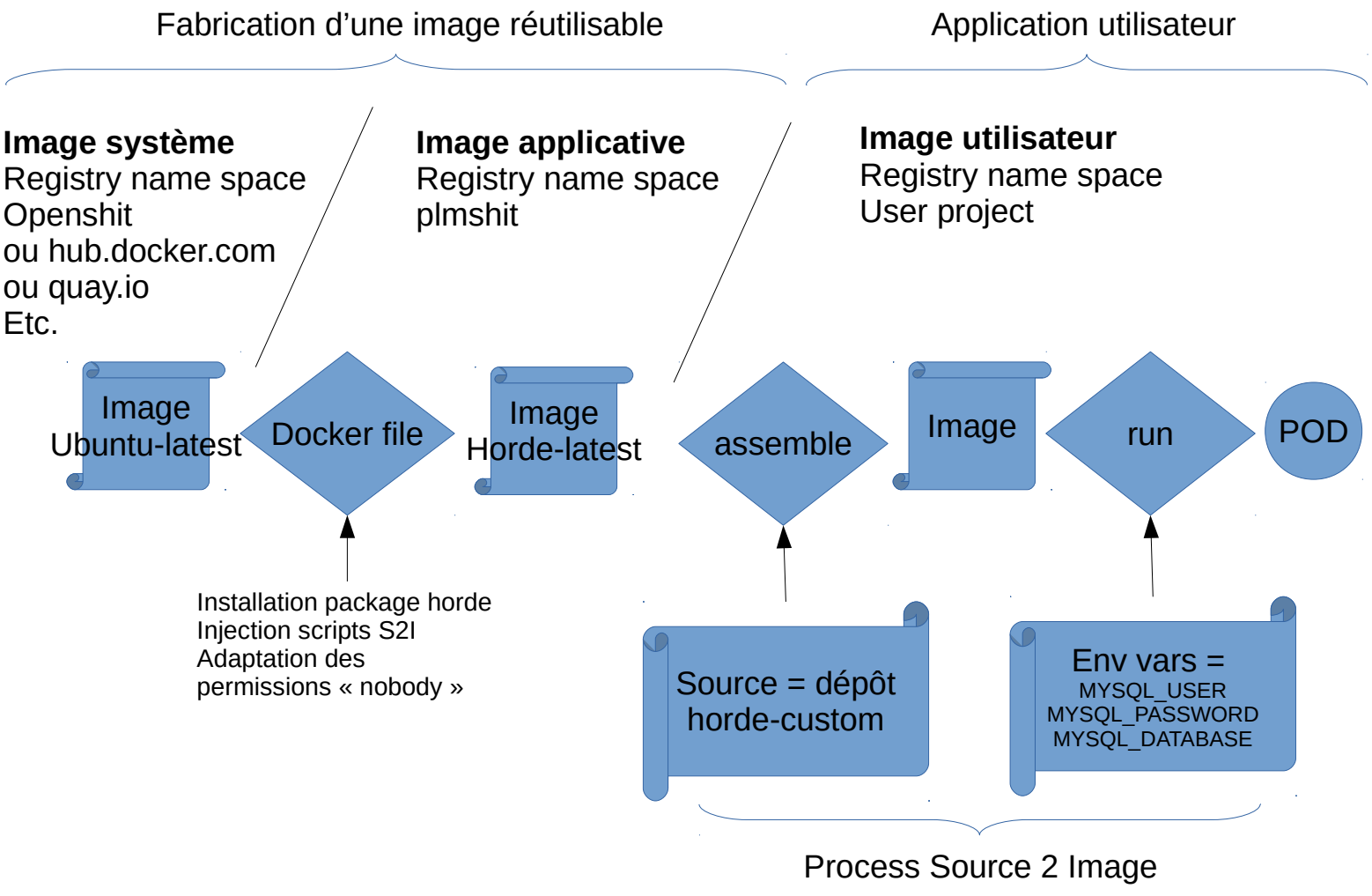
Principe : à partir d'une image source, obtenir une image personnalisée prête à être déployée

- Composition de l'image personnalisée
  - un **BuildConfig** avec une recette
    - **strategy Docker** : dockerfile
    - **strategy Source** : s2i (source2image)
  - un **Image Stream** = stockage de l'image de référence dans le projet utilisateur (registry local au projet)

Lire impérativement la documentation
`https://docs.okd.io/latest/dev_guide/builds/index.html`

mathrice

Eléments de base - schéma de principe d'un déploiement



Exemple de BuildConfig basé sur une image Docker

```
{
  "kind": "BuildConfig",
  "apiVersion": "v1",
  "metadata": {
    "name": "application",
    "labels": {
      "app": "application"
    }
  },
  "spec": {
    "source": {
      "type": "Git",
      "git": {
        "uri": "https://plmlab.math.cnrs.fr/plmshift/...",
        "ref": "master"
      },
      "contextDir": "Docker"
    },
    "strategy": {
      "type": "Docker",
      "dockerStrategy": {
        "from": {
          "kind": "DockerImage",
          "name": "ubuntu:18.04"
        }
      }
    },
```

```
    "output": {
      "to": {
        "kind": "ImageStreamTag",
        "name": "application:latest"
      }
    },
    {
      "kind": "ImageStream",
      "apiVersion": "v1",
      "metadata": {
        "name": "application",
        "labels": {
          "app": "application"
        }
      }
    },
```

Exemple de BuildConfig basé sur Source2Image

```
{
  "kind": "BuildConfig",
  "apiVersion": "v1",
  "metadata": {
    "name": "monlamp",
    "labels": {  "app": "monapp"    }
  },
  "spec": {
    "triggers": [
      { "type": "ConfigChange" },
      { "type": "ImageChange" }
    ],
    "source": {
      "type": "Git",
      "git": {
        "uri": "https://github.com/LimeSurvey/LimeSurvey.git",
        "ref": "master"
      }
    },
    "strategy": {
      "type": "Source",
      "sourceStrategy": {
        "from": {
          "kind": "ImageStreamTag",
          "namespace": "openshift",
          "name": "php:7.1"
        }
      }
    },
    "output": {
      "to": {
        "kind": "ImageStreamTag",
        "name": "monlamp:latest"
      }
    },
    {
      "kind": "ImageStream",
      "apiVersion": "v1",
      "metadata": {
        "name": "monlamp",
        "labels": {
          "app": "monapp"
        }
      }
    },
```

Eléments d'un BuildConfig

Principe :

On part d'une image Docker quelconque (quay, docker.io, etc.) ou bien d'une image stockée dans le registry interne :
from Dockerfile ou from ImageStreamTag

On décrit les modifications dans une recette Docker ou bien on fusionne avec le contenu d'un dépôt GIT tiers :
Strategy Type Docker ou Strategy Type Source

L'image résultante est déposée dans le registry réservé au projet :
to : ImageStreamTag

mathrice

Eléments d'un BuildConfig

A savoir !!!

- Un POD s'exécute toujours (pour un utilisateur normal) sous un ID arbitraire de type NOBODY
- Il faut toujours penser à configurer les services (mysql, http, nodejs, etc.) pour qu'ils se lancent sous un utilisateur arbitraire
- Pour aider à la configuration, cet utilisateur a TOUJOURS comme groupe primaire : 0 (root)
- Les images fournies par Openshift se terminent par convention par **USER 1001**, c'est donc sous cet utilisateur que les commandes suivantes du Buildconfig s'exécuteront (impossible de faire yum install ou apt-get install)

mathrice

Eléments d'un BuildConfig : recette Docker

Docker Strategy

- Utilise simplement la commande **docker build** avec un fichier Dockerfile fourni (via un dépôt GIT ou directement injecté dans le JSON/YAML)
- Dans la stratégie, l'attribut **from DockerImage** ou **from ImageStreamTag** est prépondérant sur le champ **FROM** du fichier Dockerfile
- Rappel : les images dédiées à Openshift sont configurées pour travailler avec un UID d'un simple utilisateur, donc à moins de revenir à root, il n'est plus possible de modifier l'espace système (plus de yum install ou apt-get install possible...)

voir :
`https://docs.okd.io/3.11/creating_images/guidelines.html`

mathrice

Exemple de Dockerfile

```
FROM centos:7
RUN yum -y install mypackage && yum clean all -y

ENV APP_ROOT=/opt/app-root
ENV PATH=${APP_ROOT}/bin:${PATH} HOME=${APP_ROOT}

COPY ./run.sh /
RUN chmod 755 /run.sh && chmod g=u /etc/passwd # Important pour uid_entrypoint

ADD myfile /test/myfile # J'ajoute mes fichiers plutôt à la fin du Dockerfile

USER 1001 # A partir de là, plus de modification système possible
WORKDIR ${APP_ROOT}

# Set the default port for applications built using this image
EXPOSE 8080

### user name recognition at runtime w/ an arbitrary uid - for OpenShift deployments
ENTRYPOINT [ "uid_entrypoint" ]
# Modify the usage script in your application dir to inform the user how to run this image.
CMD ["/run.sh"]
```

uid_entrypoint ressemble à ceci :

```
if ! whoami &> /dev/null; then
  if [ -w /etc/passwd ]; then
    echo "${USER_NAME:-default}:x:$(id -u):0:${USER_NAME:-default} user:${HOME}:/sbin/nologin" >> /etc/passwd
  fi
fi
```

Eléments d'un BuildConfig : recette Source2Image

Source Strategy

- Part d'une image Docker ou un ImageStreamTag (comme pour la stratégie Docker)
- Clone un dépôt GIT personnalisé, fabrique un tar du contenu
- Déploie ce tar dans /tmp/src (par défaut, paramétrable)
- Appelle un script shell **assemble** pour intégrer le contenu du dépôt GIT depuis /tmp/src dans l'image produite
- Dépose le résultat dans un ImageStreamTag du projet
- Lorsque l'image est exécutée dans un POD, Openshift exécute alors un script shell **run**

mathrice

Eléments d'un BuildConfig : recette Source2Image

Exemple de script assemble

```
#!/bin/bash -e

# Ce script présuppose que mon dépôt GIT contient
# un dossier sources à rappatrier dans mon image destination

mkdir -p /tmp/src/html
mv /tmp/src/html /opt/app-root/html
rm -rf /tmp/src
```

Exemple de script run

```
#!/bin/bash -e

# Pour les images Apache Openshift, le DocumentRoot est
# /opt/app-root/src/
# Je recopie donc le contenu html dans le DocumentRoot

rsync -av /opt/app-root/html/ /opt/app-root/src/

# Je lance Apache en mode FOREGROUND pour logger sur la console (container immutable)
exec httpd -D FOREGROUND
```

mathrice

Eléments d'un BuildConfig : recette Source2Image

Les scripts shell (??**assemble** et **run**??) pour Source2Image sont disponibles pour le container sous plusieurs formes

- Un script passé en argument via **–scripts-url URL**
- Un script dans le dossier **.s2i** du dépôt GIT source
- Un script à l'URL spécifié par un label Docker **io.openshift.s2i.scripts-url**

Le label io.openshift.s2i.scripts-url peut être spécifié dans l'image Docker source ou bien dans un attribut du BuildConfig.
`https://github.com/openshift/source-to-image/blob/master/docs/builder_image.md`

mathrice

Exemple de Dockefile incluant les scripts S2I

```
FROM centos:7
RUN yum -y install mypackage && yum clean all -y

ENV APP_ROOT=/opt/app-root
ENV PATH=${APP_ROOT}/bin:${PATH} HOME=${APP_ROOT}
RUN chmod g=u /etc/passwd # Important pour uid_entrypoint
LABEL io.k8s.description="Docker Image for application and S2I builder" \
    # this label tells s2i where to find its mandatory scripts
    # (run, assemble, save-artifacts)
    io.openshift.s2i.scripts-url="image:///usr/libexec/s2i"
COPY ./s2i/bin/ /usr/libexec/s2i # ./s2i/bin contient ici au moins les scripts assemble, run et usage

ADD myfile /test/myfile

USER 1001 # A partir de là, plus de modification système possible
WORKDIR ${APP_ROOT}

# Set the default port for applications built using this image
EXPOSE 8080

### user name recognition at runtime w/ an arbitrary uid - for OpenShift deployments
ENTRYPOINT [ "uid_entrypoint" ]
# Modify the usage script in your application dir to inform the user how to run this image.
CMD ["/usr/libexec/s2i/usage"]
```

Déployer sa propre application

Le catalogue propose des "templates" qui sont des applications paramétrisées. Ces paramètres sont réutilisables dans l'ensemble des fonctionnalités de l'application en étant converties en variables d'environnement.
Consultez les fichiers "templates" du dépôt public de `https://plmlab.math.cnrs.fr/plmshift/shiny`

mathrice

Déployer sa propre application via un dépôt GIT privé

Il est possible d'associer un secret (identifiant/mot de passe ou token ou clé privée SSH) à un BuildConfig de telle sorte de récupérer le contenu privé du dépôt.
voir le README de
`https://plmlab.math.cnrs.fr/plmshift/shiny-custom`
et
`https://docs.okd.io/latest/dev_guide/builds/build_inputs.html#source-clone-secrets`

mathrice

Déployer sa propre application - TP2

Instancier un POD de votre choix du "Service Catalog" parmi ceux proposés ici : horde-custom, Shiny-R Application Server, Wordpress, Wordpress Kit CNRS

- Suivez les instructions du README du dépôt PLMlab associé (dans les dépôts de `https://plmlab.math.cnrs.fr/plmshift`)
- augmentez le nombre de replicas de votre application
- Créez un webhook pour relancer l'appli lors d'une modification des données

mathrice

Déployer sa propre application - TP3

Clonez le dépôt `https://plmlab.math.cnrs.fr/plmshift/lamp`

- Regardez le fichier templates/lamp.json
- Editez les fichiers des dossiers php-pre-start et httpd-cfg
- oc create -f templates/lamp.json
- Dans la console Web sélectionnez "From Project" puis "LAMP (with Repository)"

mathrice

Proposition de co-construction

- Le projet (namespace) **plmshift** accueille un registry/ImageStream ("namespace": "plmshift") public. Dans ce namespace sont maintenus des BuildConfig (et ImageStream correspondant) afin de proposer des images d'applications pour la communauté.
- Dans le Service Catalog sont ajoutés des templates utilisant ces ImagesStreams. Les dépôts GIT correspondants sont hébergés dans `https://plmlab.math.cnrs.fr/plmshift`

mathrice