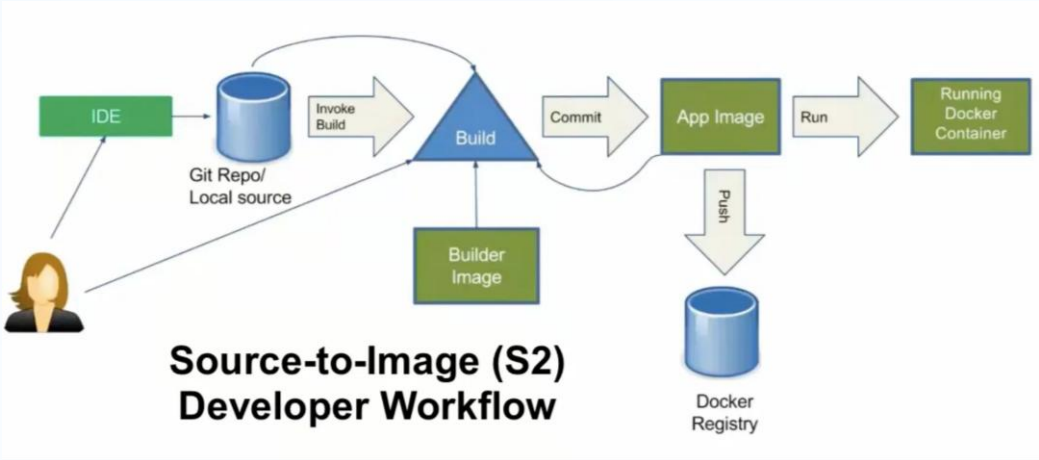
Module 5

Personnalisation de versions source-to-image

Fonctionnement technique détaillé



Objectifs du module détaillé

Comprendre en profondeur l'architecture et le fonctionnement technique de S2I

Maîtriser les composants internes, le workflow et les mécanismes de S2I

Analyser en détail les scripts S2I et leur rôle dans le processus de build

Étudier les scripts assemble, run, save-artifacts et les hooks personnalisables

Maîtriser les techniques avancées de personnalisation d'images S2I

Créer des images S2I optimisées pour des besoins spécifiques, notamment pour .NET Core

Optimiser les builds S2I pour améliorer les performances et la sécurité

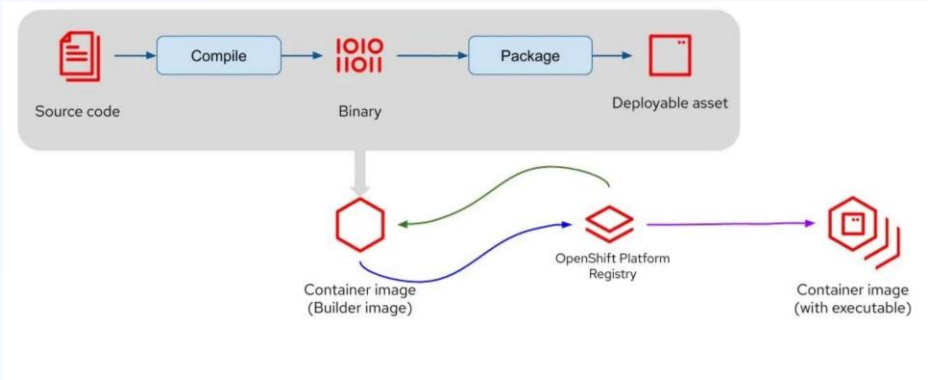
Implémenter des builds incrémentaux et des configurations avancées pour les environnements de production

Appliquer les connaissances acquises à travers des exercices pratiques

Réaliser 3 exercices pratiques utilisant l'application eShopOnWeb pour maîtriser S2I

Introduction à S2I

Source-to-Image (S2I) est un framework et un ensemble d'outils qui facilitent la création d'images de conteneurs reproductibles à partir du code source d'une application, en automatisant le processus de construction et de déploiement.



- **Automatisation complète du processus de build**

  Élimine les étapes manuelles en injectant le code source dans une image de base et en exécutant les scripts de construction

- **Abstraction de la complexité des conteneurs**

  Permet aux développeurs de se concentrer sur le code plutôt que sur la configuration des conteneurs

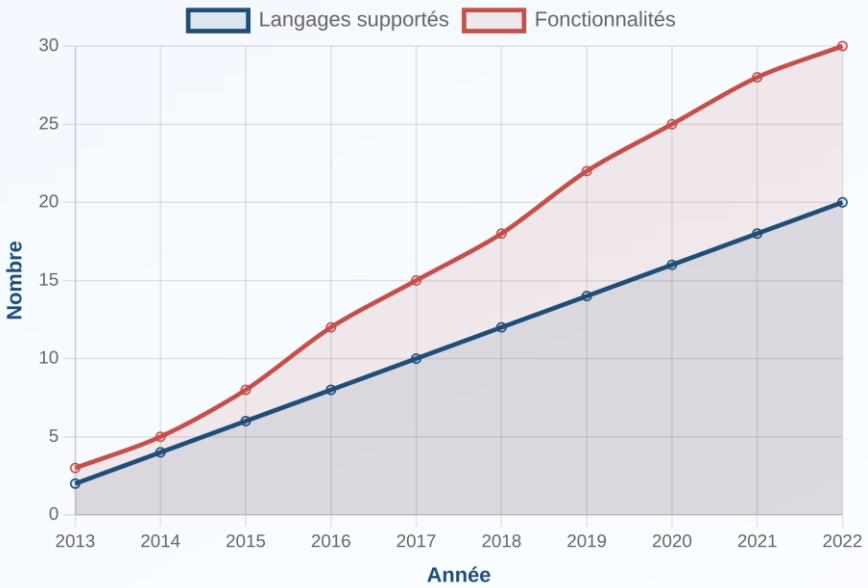
- **Intégration native avec OpenShift**

  Composant fondamental de la stratégie de build d'OpenShift, optimisé pour les workflows CI/CD

- **Extensibilité et personnalisation**

  Architecture modulaire permettant d'adapter le processus à des besoins spécifiques via des scripts et des hooks

Historique et évolution de S2I



2013 — **Création initiale**

Développement initial de S2I par Red Hat pour simplifier le processus de construction d'images de conteneurs à partir du code source

2014 — **Intégration à OpenShift v3**

S2I devient un composant fondamental de la stratégie de build d'OpenShift v3, remplaçant les cartridges d'OpenShift v2

2016 — **Open Source et standardisation**

Publication en tant que projet open source indépendant et standardisation des scripts et des interfaces

2018 — **Support multi-langages étendu**

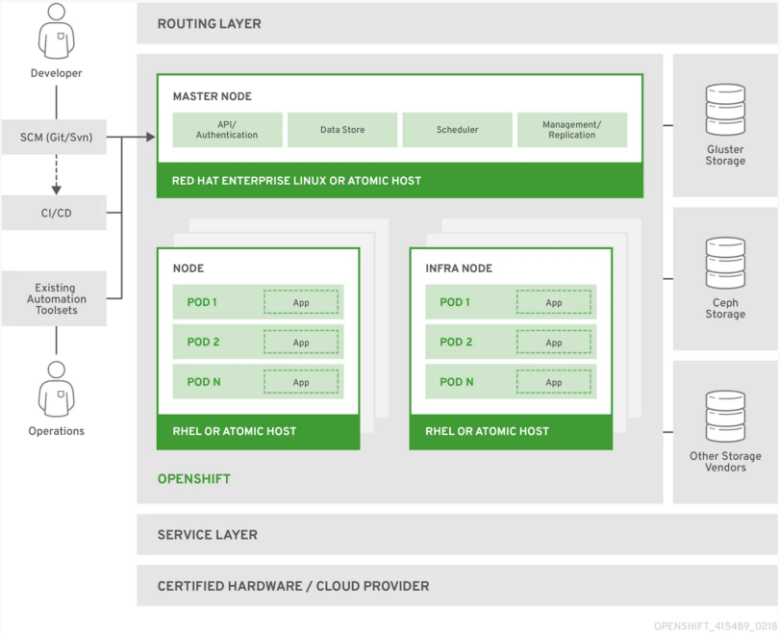
Extension du support à de nombreux langages et frameworks, dont .NET Core, Node.js, Python, Java, Ruby, PHP et Perl

2020+ — **Intégration CI/CD avancée**

Intégration avec les pipelines OpenShift, Tekton et autres outils CI/CD modernes, optimisations pour les builds incrémentaux

Architecture de S2I

L'architecture de Source-to-Image est conçue pour être modulaire et extensible, permettant une séparation claire des responsabilités entre les différents composants du système.



Builder Image

Image de base contenant l'environnement d'exécution et les scripts S2I nécessaires pour construire et exécuter l'application.

Scripts S2I

Ensemble de scripts (assemble, run, save-artifacts) qui définissent le comportement du processus de build et d'exécution.

BuildConfig

Ressource OpenShift qui définit les paramètres de construction, incluant la source, la stratégie et les déclencheurs.

Application Image

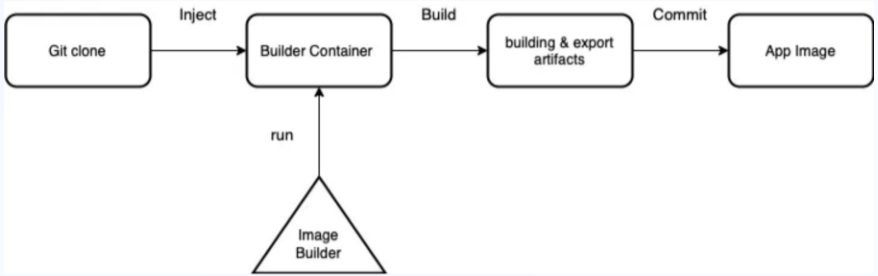
Image finale contenant l'application construite, prête à être déployée dans des conteneurs.

Workflow S2I

1 Récupération des sources

OpenShift clone le code source depuis le dépôt Git spécifié

Supporte Git, archives tar, ou répertoires locaux comme sources



2 Téléchargement de l'image builder

L'image S2I de base correspondant au langage est récupérée

Contient l'environnement d'exécution et les scripts S2I nécessaires

3 Exécution du script assemble

Compilation du code source et création des artefacts de l'application

Installation des dépendances, compilation, tests unitaires, etc.

4 Création de l'image finale

Une nouvelle image de conteneur est créée avec les artefacts générés

Inclut le script run qui sera exécuté au démarrage du conteneur

5 Publication et déploiement

L'image est poussée vers le registre interne et déployée dans OpenShift

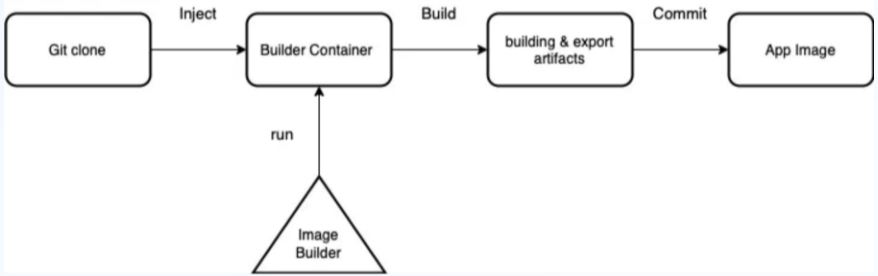
Déclenchement automatique des déploiements via ImageStream triggers

Étapes détaillées du processus S2I

1 Récupération du code source

Téléchargement du code source depuis le dépôt Git spécifié

Utilise git clone ou télécharge l'archive directement via HTTP/HTTPS



2 Téléchargement de l'image builder

Récupération de l'image S2I de base correspondant au langage/framework

Utilise docker/podman pull pour récupérer l'image depuis le registre

3 Exécution du script assemble

Compilation du code source et création des artefacts de l'application

Exécute /usr/libexec/s2i/assemble dans un conteneur temporaire

4 Création de l'image finale

Génération d'une nouvelle image contenant les artefacts compilés

Utilise docker/podman commit pour créer l'image avec les artefacts

5 Push vers le registre

Publication de l'image finale vers le registre interne d'OpenShift

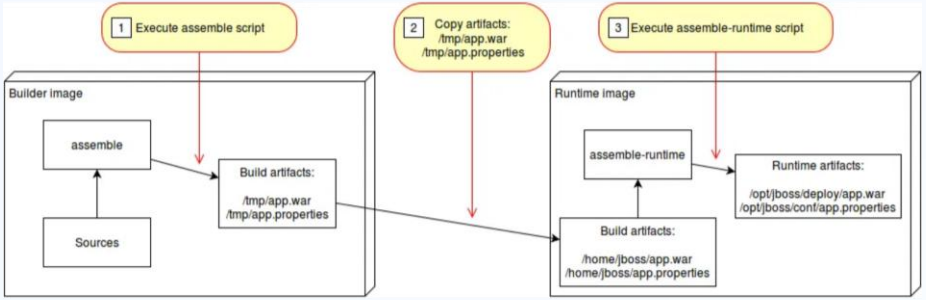
Utilise docker/podman push vers le registre configuré

6 Déploiement de l'application

Création des ressources OpenShift et démarrage des conteneurs

Exécute le script run (/usr/libexec/s2i/run) au démarrage du conteneur

Images de construction S2I



Structure d'une image de construction S2I

Organisation des répertoires

Structure standardisée avec des répertoires spécifiques pour les scripts et configurations.

`/usr/libexec/s2i/ | /opt/app-root/`

Scripts obligatoires

Ensemble de scripts requis pour le fonctionnement de S2I.

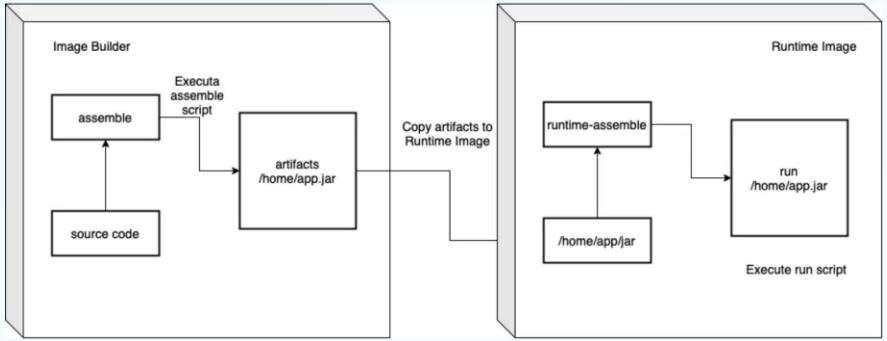
`assemble | run | usage`

Scripts optionnels

Scripts additionnels pour des fonctionnalités avancées.

`save-artifacts | assemble-runtime`

Scripts S2I



assemble

Responsable de la construction de l'application à partir du code source. Installe les dépendances, compile le code et prépare l'environnement d'exécution.

run

Exécuté au démarrage du conteneur pour lancer l'application. Configure l'environnement d'exécution et démarre le processus principal.

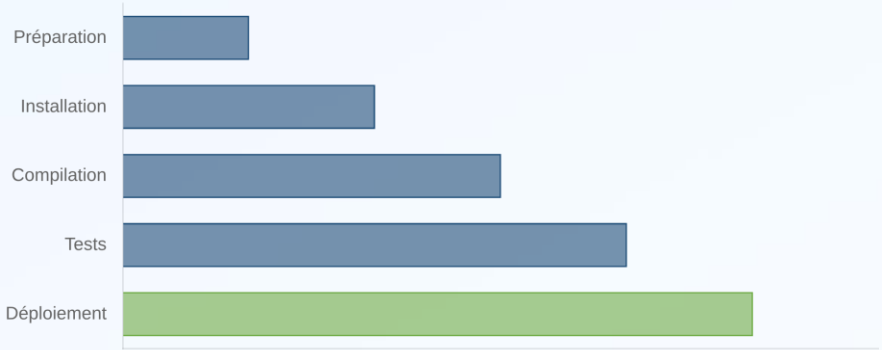
save-artifacts

Utilisé pour les builds incrémentaux. Sauvegarde les artefacts (comme les dépendances) qui peuvent être réutilisés lors des builds suivants.

usage

Affiche des informations d'aide sur l'utilisation de l'image builder, y compris les variables d'environnement supportées et les exemples d'utilisation.

Script assemble




```
#!/bin/bash
set -e

# Téléchargement des dépendances
echo "---> Installation des dépendances..."
npm install -s

# Compilation du code
echo "---> Compilation de l'application..."
npm run build

# Déplacement des artefacts
echo "---> Déplacement des artefacts..."
cp -Rf dist/. /opt/app-root/output/
```

Fonction principale

Le script assemble transforme le code source en artefacts exécutables pendant la phase de build.

Tâches typiques

- Installation des dépendances (npm, pip, maven)
- Compilation du code source
- Génération des artefacts de build
- Déplacement des fichiers vers les répertoires appropriés

Points de personnalisation

- Ajouter des étapes de build spécifiques
- Configurer des variables d'environnement
- Optimiser les artefacts générés

Points d'attention

- Doit retourner un code de sortie 0 en cas de succès
- Doit gérer correctement les erreurs
- Doit respecter les chemins standards de S2I

Script run

Le script **run** est exécuté au démarrage du conteneur et a pour responsabilité de lancer l'application. C'est l'équivalent de la commande CMD ou ENTRYPOINT dans un Dockerfile.

Exemple de script run pour Node.js

```
#!/bin/bash
set -e

# Définition des variables d'environnement
if [ -e "/opt/app-root/etc/environment" ]; then
  source /opt/app-root/etc/environment
fi

# Utilisation de nodemon en développement
if [ "$DEV_MODE" == true ]; then
  exec npx nodemon app.js
else
  # Exécution de l'application en production
  exec node app.js
fi
```

Démarrage de l'application

Responsable du lancement de l'application avec les paramètres appropriés.

Configuration de l'environnement

Configure les variables d'environnement nécessaires au fonctionnement.

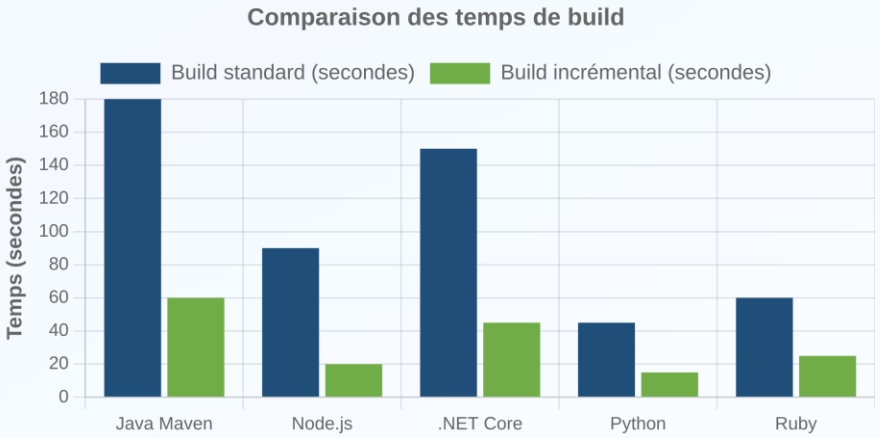
Gestion des modes d'exécution

Adapte le comportement selon le mode (développement, production).

Redirection des signaux

Assure la transmission des signaux système pour une terminaison propre.

Script save-artifacts et builds incrémentaux



Principe des builds incrémentaux

Mécanisme permettant de réutiliser des artefacts entre les builds successifs pour réduire considérablement le temps de construction.

Fonctionnement du script save-artifacts

Script exécuté avant le build pour sauvegarder les dépendances et artefacts réutilisables.

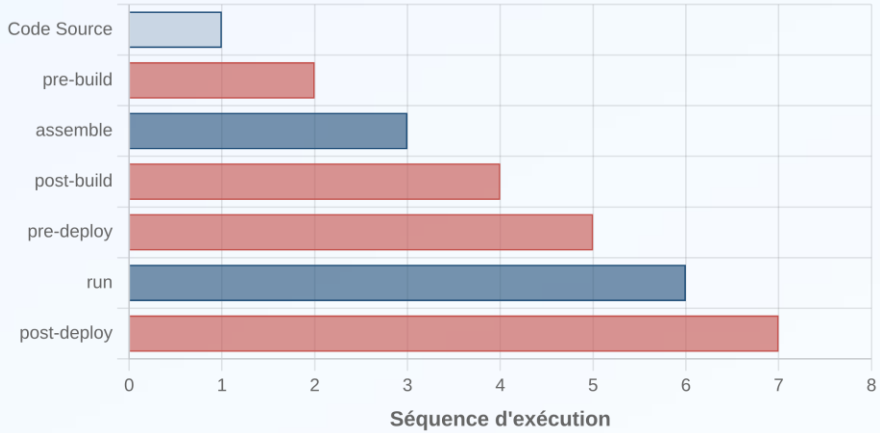
```
#!/bin/bash # Exemple pour Node.js cd $HOME tar cf - node_modules
```

Activation des builds incrémentaux

Configuration dans le BuildConfig pour activer les builds incrémentaux.

```
oc set env bc/nom-application INCREMENTAL=true
```

Hooks S2I



Les hooks S2I sont des points d'extension qui permettent de personnaliser le processus de build à différentes étapes sans modifier les scripts principaux.

→ pre-build

Exécuté avant le script assemble. Permet de préparer l'environnement ou effectuer des vérifications préliminaires.

→ post-build

Exécuté après le script assemble. Permet d'effectuer des opérations de nettoyage ou d'optimisation.

→ pre-deploy

Exécuté avant le déploiement. Permet de configurer l'environnement d'exécution ou d'effectuer des migrations.

→ post-deploy

Exécuté après le déploiement. Permet d'effectuer des vérifications de santé ou d'enregistrer l'application.

Pourquoi personnaliser une version S2I

Adaptation aux besoins spécifiques

Les images S2I standard ne couvrent pas tous les cas d'utilisation. La personnalisation permet d'adapter le processus aux exigences particulières de votre application.

Optimisation des performances

Réduire les temps de build en optimisant les scripts pour votre application spécifique et en éliminant les étapes inutiles.

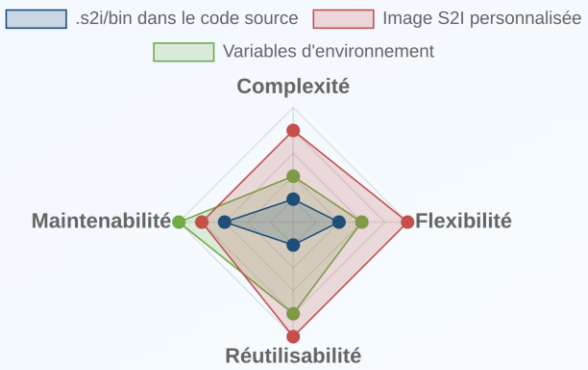
Renforcement de la sécurité

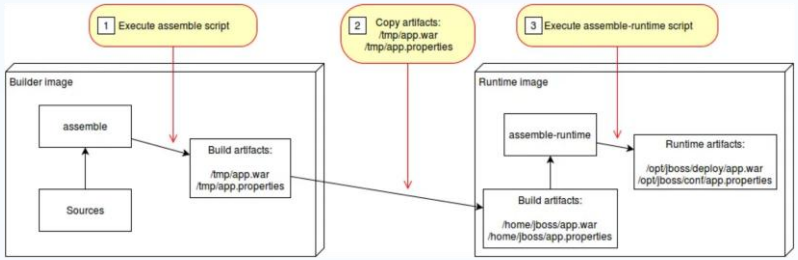
Intégrer des outils d'analyse de sécurité et des scanners de vulnérabilités directement dans le processus de build.

Intégration CI/CD avancée

Adapter les images S2I pour s'intégrer parfaitement dans vos pipelines CI/CD existants avec des hooks personnalisés.

Personnalisation d'une image S2I





La personnalisation d'une image S2I permet d'adapter le processus de build aux besoins spécifiques de votre application.

Personnalisation via le répertoire .s2i

Ajoutez un répertoire .s2i/bin dans votre code source contenant vos propres versions des scripts S2I.

Création d'une image S2I personnalisée

Créez votre propre image S2I en étendant une image existante et en modifiant les scripts pour répondre à vos besoins.

Configuration via variables d'environnement

Utilisez des variables d'environnement pour configurer le comportement des scripts S2I sans les modifier directement.

Création d'un Dockerfile pour S2I

Exemple de Dockerfile pour une image S2I Node.js

```
FROM registry.access.redhat.com/ubi8/nodejs-14

# Métadonnées de l'image
LABEL io.k8s.description="Plateforme pour Node.js pour applications web" \
      io.openshift.tags="builder,nodejs,nodejs14" \
      io.openshift.s2i.scripts-url="image:///usr/libexec/s2i"

# Variables d'environnement
ENV NODE_ENV=production \
    APP_ROOT=/opt/app-root

# Installation des dépendances globales
RUN npm install -g nodemon

# Copie des scripts S2I personnalisés
COPY ./s2i/bin/ /usr/libexec/s2i/

# Permissions
RUN chown -R 1001:0 ${APP_ROOT} && \
    chmod -R +x /usr/libexec/s2i

# Utilisateur non-root
USER 1001

# Port d'exposition
EXPOSE 8080

# Commande par défaut
CMD ["/usr/libexec/s2i/usage"]
```

Labels et métadonnées

Les labels identifient l'image comme builder S2I et fournissent des informations sur son utilisation.

- io.openshift.s2i.scripts-url : Emplacement des scripts
- io.k8s.description : Description de l'image

Structure des répertoires

Une structure cohérente est nécessaire pour S2I.

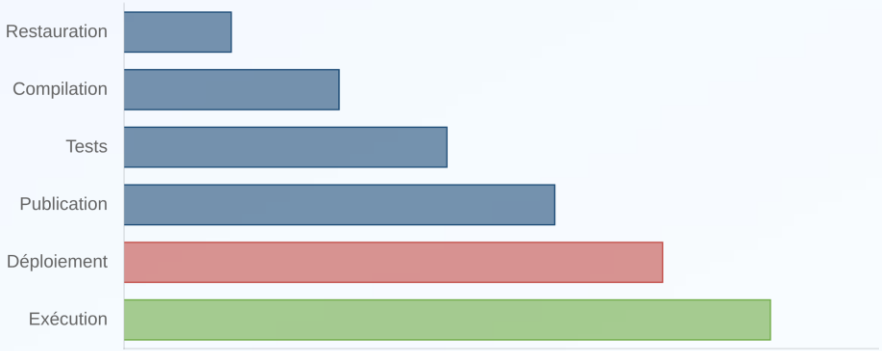
- /usr/libexec/s2i/ : Scripts S2I
- /opt/app-root/ : Racine de l'application

Sécurité et permissions

Les images S2I doivent être sécurisées et fonctionner avec un utilisateur non-root.

Utilisateur non-root (UID 1001)
Permissions appropriées sur les répertoires

S2I pour .NET Core



Structure du script assemble

Le script assemble pour .NET Core utilise la commande dotnet pour restaurer les dépendances et compiler l'application.

```
dotnet restore dotnet publish -c Release
```

Script run spécifique

Le script run configure les variables d'environnement spécifiques à ASP.NET Core et exécute l'application.

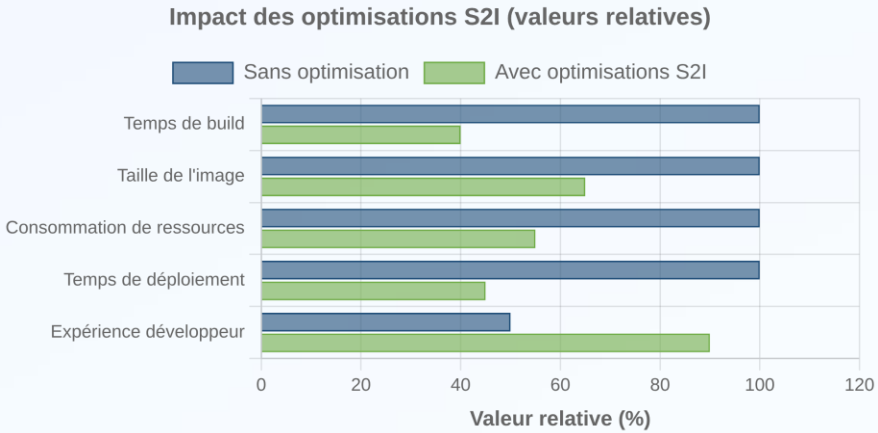
```
export ASPNETCORE_URLS="http://*:8080" exec dotnet ${APP_DLL_NAME}.dll
```

Personnalisation pour eShopOnWeb

Pour l'application eShopOnWeb, des personnalisations spécifiques peuvent être nécessaires pour gérer les dépendances frontend et les migrations de base de données.

```
# Dans .s2i/bin/assemble npm install && npm run build dotnet ef database update
```

Optimisation des builds S2I



Builds incrémentaux

Activez les builds incrémentaux pour réutiliser les dépendances et artefacts entre les builds successifs, réduisant ainsi considérablement le temps de construction.

Filtrage des fichiers sources

Utilisez un fichier .s2iignore pour exclure les fichiers non nécessaires au build, comme les tests, la documentation ou les fichiers temporaires.

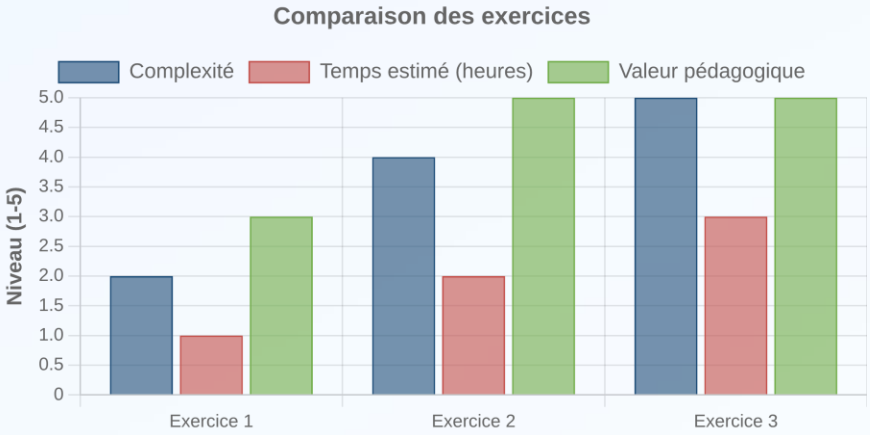
Mise en cache des dépendances

Implémentez des stratégies de mise en cache des dépendances pour éviter de les télécharger à chaque build, particulièrement important pour les projets volumineux.

Réduction de la taille des images

Optimisez la taille des images en supprimant les outils de développement inutiles en production et en utilisant des images de base plus légères.

Introduction aux travaux pratiques



Les travaux pratiques suivants vous permettront d'appliquer les concepts de S2I vus dans ce module. Vous utiliserez l'application eShopOnWeb comme base pour explorer différentes techniques de personnalisation et d'optimisation des builds S2I dans OpenShift.

Exercices pratiques

Exercice 1 : Utilisation de base de S2I

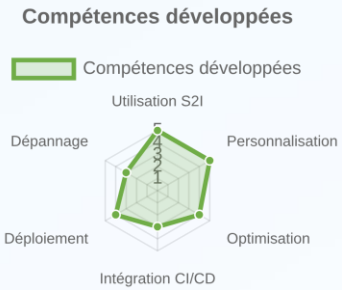
Déploiement de l'application eShopOnWeb avec l'image S2I .NET Core standard. Découverte du processus de base de Source-to-Image et analyse des artefacts générés.

Exercice 2 : Personnalisation d'une image S2I

Création d'une image S2I personnalisée pour eShopOnWeb avec des scripts assemble et run adaptés. Optimisations spécifiques pour .NET Core et gestion des dépendances frontend.

Introduction aux travaux pratiques

Exercices pratiques



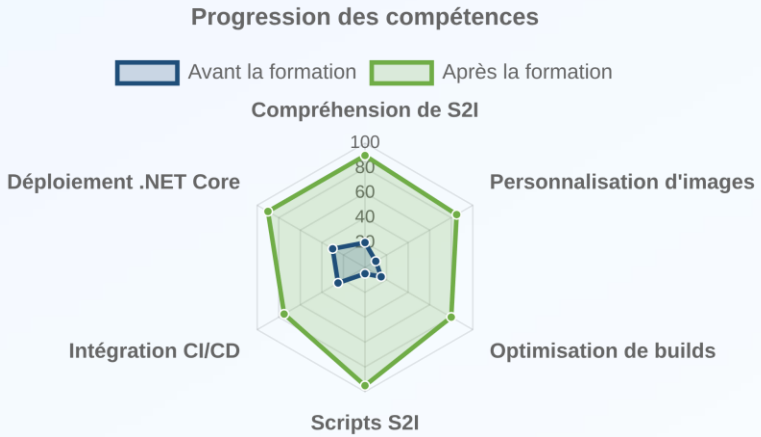
Exercice 3 : Déploiement avancé avec S2I

Configuration multi-environnements, intégration avec une base de données PostgreSQL, et mise en place d'un déploiement production-ready avec health checks et autoscaling.

Objectifs d'apprentissage

À la fin de ces exercices, vous serez capable de personnaliser des images S2I pour vos propres applications, d'optimiser le processus de build, et d'intégrer S2I dans un pipeline CI/CD complet.

Conclusion et récapitulatif



Prochaines étapes : Après avoir maîtrisé la personnalisation de S2I, vous pourrez explorer l'intégration avec des pipelines CI/CD, l'automatisation des déploiements multi-environnements, et la mise en place de stratégies avancées de déploiement comme le blue-green ou le canary.

Points clés à retenir

Comprendre le fonctionnement de S2I

S2I est un outil puissant qui simplifie la création d'images conteneur à partir du code source, en automatisant le processus de build et en appliquant les meilleures pratiques.

Maîtriser les scripts S2I

Les scripts assemble, run et save-artifacts sont les composants clés de S2I. Leur personnalisation permet d'adapter le processus de build aux besoins spécifiques de votre application.

Personnaliser pour optimiser

La personnalisation des images S2I permet d'optimiser les performances, d'améliorer la sécurité et d'adapter le processus de build aux spécificités de votre application et de votre organisation.

Appliquer à des cas réels

L'application des concepts S2I à des applications réelles comme eShopOnWeb démontre la flexibilité et la puissance de cette approche pour différents frameworks et langages.