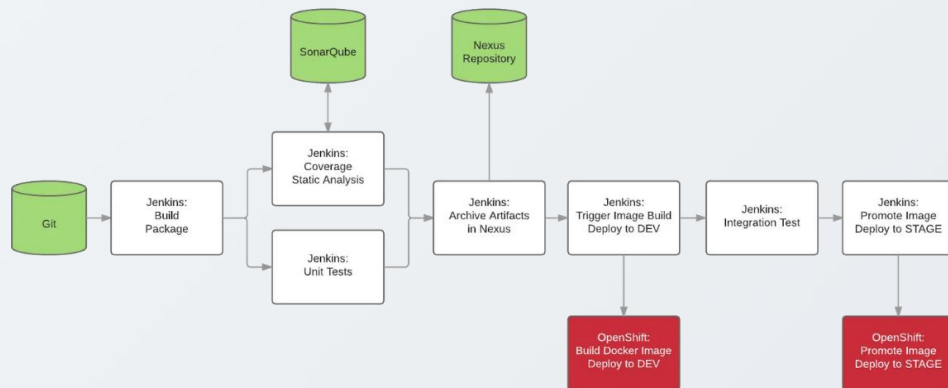


Module 4

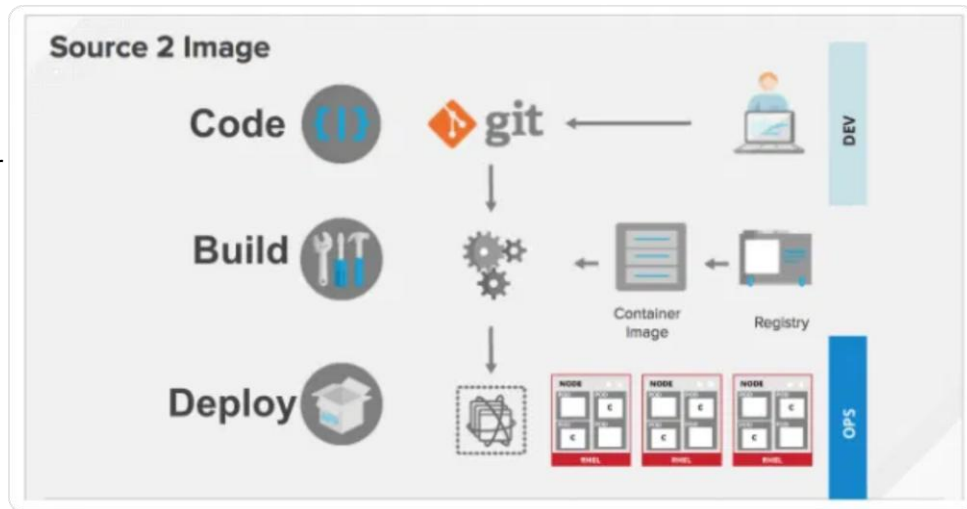
Construction d'applications

Processus de construction et crochets de version



Objectifs du module

- Comprendre le **processus de construction** dans OpenShift et ses différentes stratégies
- Maîtriser la technologie **Source-to-Image (S2I)** pour la création d'images de conteneurs
- Implémenter des **crochets de version** après soumission pour automatiser le versioning
- Intégrer le processus de construction dans un **pipeline CI/CD** complet
- Appliquer les **bonnes pratiques** pour optimiser les builds et le versioning



Ce module vous permettra de maîtriser les mécanismes de construction d'applications dans OpenShift, depuis la création d'images jusqu'à leur versioning automatisé, en passant par l'optimisation des processus de build.

Introduction au processus de construction

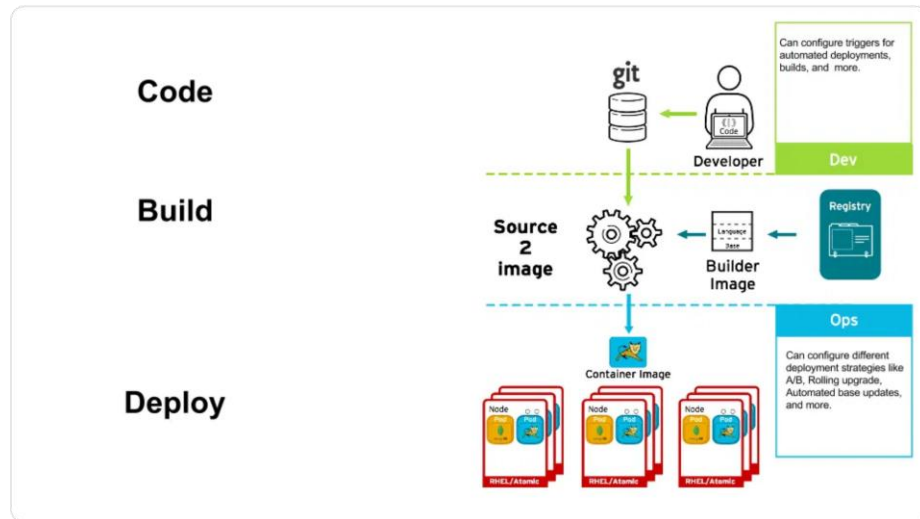
Qu'est-ce que le processus de construction ?

Le processus de construction dans OpenShift est le mécanisme qui transforme votre **code source** en **images de conteneurs** exécutables, prêtes à être déployées.

Étapes du processus

- 1 **Source** : Récupération du code source depuis un dépôt Git, un répertoire local ou une image existante
- 2 **Build** : Compilation du code et création de l'image selon la stratégie définie
- 3 **Push** : Publication de l'image dans le registre interne ou externe
- 4 **Deploy** : Déploiement de l'application à partir de l'image construite

OpenShift utilise le concept de **BuildConfig** pour définir comment les applications sont construites. Plusieurs stratégies de construction sont disponibles, notamment **Source-to-Image (S2I)**, **Docker**, **Custom** et **Pipeline**.







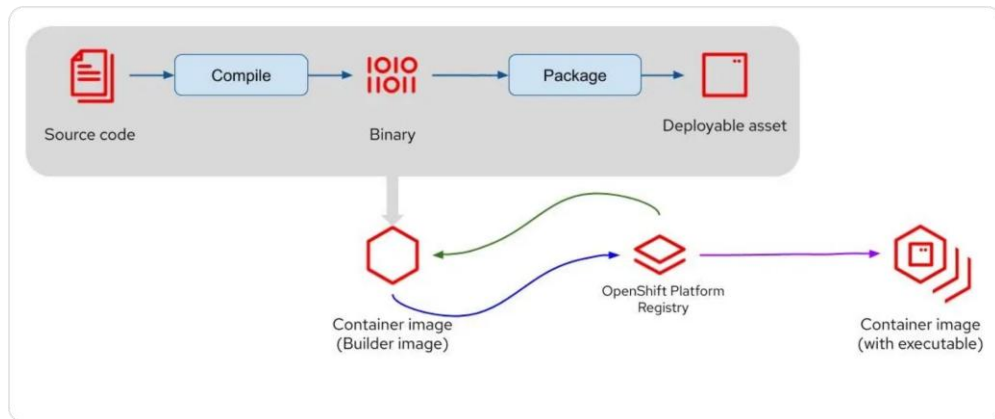
Source-to-Image (S2I)

Qu'est-ce que S2I ?

Source-to-Image (S2I) est un framework qui facilite la création d'images de conteneurs reproductibles à partir du code source. Il automatise le processus d'intégration du code applicatif dans une image de conteneur.

Caractéristiques principales

-  **Simplicité** : Pas besoin de connaître Docker ou de créer des Dockerfiles
-  **Sécurité** : Images standardisées et sécurisées par défaut
-  **Reproductibilité** : Builds cohérents et prévisibles
-  **Extensibilité** : Possibilité de créer des images builder personnalisées



Fonctionnement de S2I

S2I fonctionne en deux phases principales : **build** (combinaison du code source avec l'image builder) et **run** (exécution de l'application assemblée).

OpenShift fournit des images builder pour les langages courants comme Java, Node.js, Python, PHP, Ruby, etc

BuildConfig et stratégies de construction

Qu'est-ce qu'un BuildConfig ?

Un **BuildConfig** est une ressource OpenShift qui définit le processus de construction d'une image de conteneur à partir du code source.

La source du code (Git, binaire, image)

La stratégie de construction à utiliser

Les déclencheurs de build

Stratégies de construction

Source-to-Image (S2I)

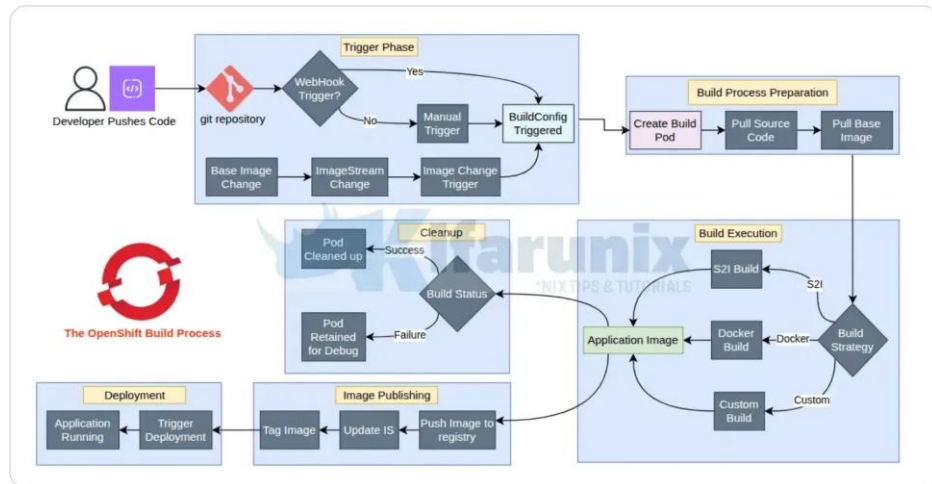
Injecte le code source dans une image de base

Docker

Utilise un Dockerfile pour construire l'image

Custom

Utilise une image de construction personnalisée



```
kind: BuildConfig
apiVersion: build.openshift.io/v1
metadata:
  name: "sample-build"
spec:
  source:
    git:
      uri: "https://github.com/user/app"
  strategy:
    sourceStrategy:
      from:
        kind: "ImageStreamTag"
        name: "nodejs:14"
  output:
    to:
      kind: "ImageStreamTag"
      name: "app:latest"
```

Gestion des images de base

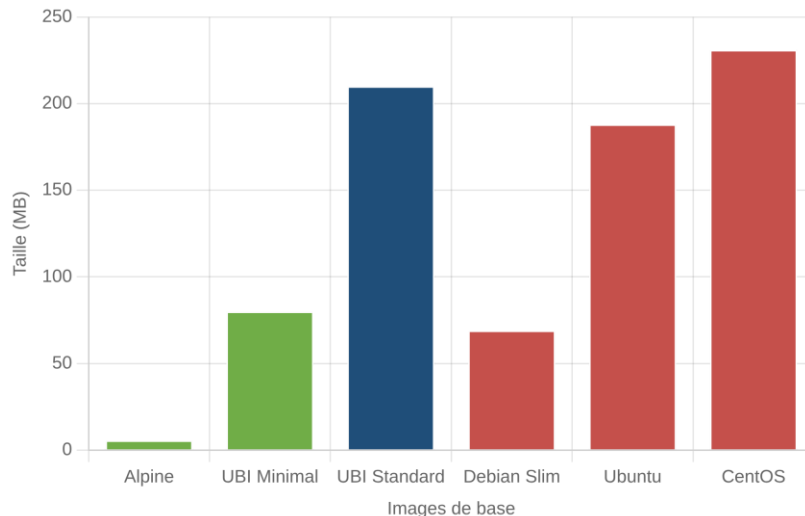
Importance des images de base

Les **images de base** sont le fondement de vos conteneurs. Leur choix impacte directement la sécurité, les performances et la taille de vos applications conteneurisées.

Bonnes pratiques

- Sécurité** : Utilisez des images officielles et maintenez-les à jour pour réduire les vulnérabilités
- Taille** : Préférez les images minimalistes (Alpine, UBI) pour réduire l'empreinte et accélérer les déploiements
- Versioning** : Utilisez des tags spécifiques plutôt que "latest" pour garantir la reproductibilité
- Standardisation** : Créez des images de base d'entreprise pour uniformiser les environnements
- Automatisation** : Mettez en place des scans de sécurité automatisés pour vos images

OpenShift propose des **images UBI (Universal Base Image)** optimisées pour la sécurité et la compatibilité avec les environnements Red Hat, tout en étant librement distribuables dans vos conteneurs.



Comparaison des tailles d'images de base populaires (MB)

Optimisation des builds

Techniques d'optimisation



Multi-stage builds

Utiliser plusieurs étapes de construction pour réduire la taille finale de l'image



.dockerignore

Exclure les fichiers inutiles du contexte de construction



Optimisation des couches

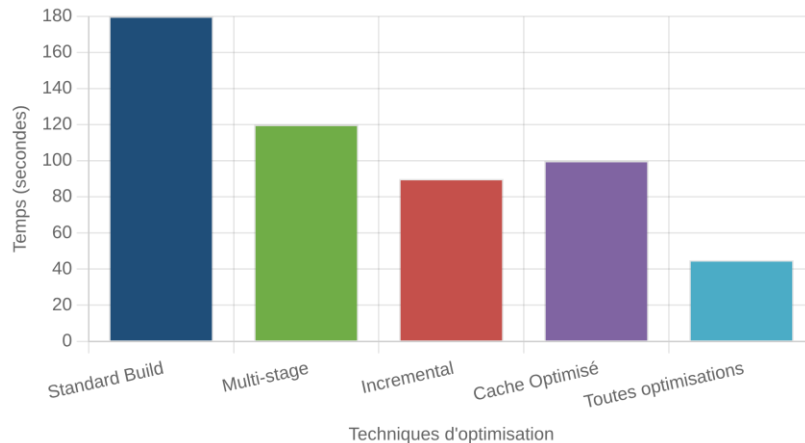
Combiner les commandes RUN pour réduire le nombre de couches



Builds incrémentaux

Réutiliser les artefacts des builds précédents pour accélérer le processus

Impact sur les performances



Conseil

Utilisez la commande `oc start-build --incremental` pour activer les builds incrémentaux et réduire considérablement le temps de construction.


L'optimisation des builds est essentielle pour améliorer la **vitesse de développement**, réduire la **consommation de ressources** et garantir la **cohérence** entre les environnements de développement et de production.


Introduction aux crochets de version


Qu'est-ce qu'un crochet de version ?

Les **crochets de version** (version hooks) sont des mécanismes qui permettent d'exécuter des actions automatisées à différentes étapes du processus de construction et de déploiement dans OpenShift.

Rôle dans le cycle de vie

 **Automatisation** : Déclenchement automatique d'actions lors d'événements spécifiques du cycle de construction

 **Versioning** : Application automatique de tags de version aux images construites

 **Validation** : Exécution de tests ou vérifications après la construction

Point clé

Les crochets de version permettent d'intégrer le **versioning sémantique** dans votre pipeline CI/CD, assurant ainsi une traçabilité complète de vos déploiements et facilitant les rollbacks en cas de problème.



Types de crochets

Principaux types de crochets



Pre-commit hooks

Exécutés **avant** la validation du code source. Utilisés pour les vérifications de qualité, les tests unitaires et la validation du format.



Post-commit hooks

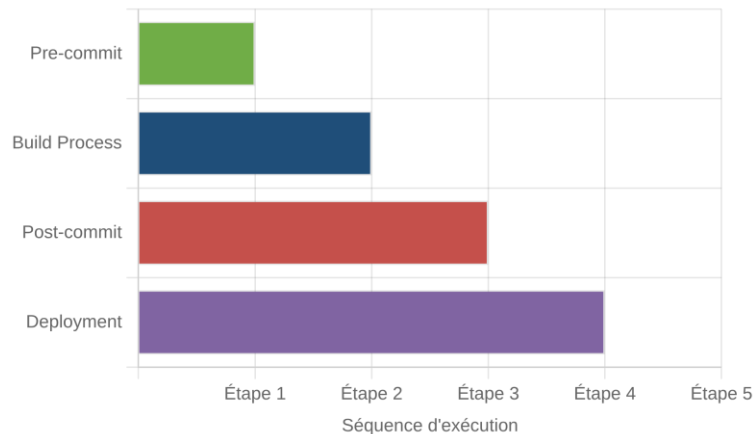
Exécutés **après** la construction de l'image. Idéaux pour l'application de tags de version, les tests d'intégration et les notifications.

```
spec:
  postCommit:
    script: "npm test && ./apply-version-tag.sh"
```



Webhook triggers

Déclencheurs externes qui initient une construction lors d'événements comme un push Git ou une mise à jour d'image.



Les **post-commit hooks** sont particulièrement utiles pour le versioning automatique, car ils s'exécutent après la validation réussie de la construction, garantissant ainsi que seules les images fonctionnelles reçoivent un tag de version.

Mise en œuvre des tags de version

Versioning sémantique

Le **versioning sémantique** (SemVer) est une convention de nommage qui utilise un format MAJOR.MINOR.PATCH pour identifier clairement la nature des changements dans une version.

Implémentation dans OpenShift

- 1 **Configuration du BuildConfig** : Ajoutez un crochet post-commit pour appliquer des tags après la construction
- 2 **Script de versioning** : Créez un script qui détermine la version appropriée basée sur les commits ou les métadonnées
- 3 **Application des tags** : Utilisez la commande `oc tag` pour appliquer les tags de version à l'image construite
- 4 **Propagation** : Configurez les déploiements pour utiliser les images avec les tags spécifiques

```
postCommit:
script: |
  VERSION=$(./get-version.sh)
  oc tag $OPENSHIFT_BUILD_NAMESPACE/$OPENSHIFT_BUILD_NAME \
    $OPENSHIFT_BUILD_NAMESPACE/$OPENSHIFT_BUILD_NAME:$VERSION
```

Recommended minimum for annotations:




```
# Exemple de script get-version.sh
#!/bin/bash
# Extraire la version depuis package.json
VERSION=$(grep "version" package.json | cut -d'"' -f4)


# Ajouter un suffixe basé sur le hash du commit
COMMIT=$(git rev-parse --short HEAD)
echo "${VERSION}-${COMMIT}"
```


L'utilisation de tags de version après soumission permet de **tracer précisément** quelle version du code est déployée, **facilite les rollbacks** en cas de problème, et **améliore la gouvernance** des déploiements dans un environnement d'entreprise.

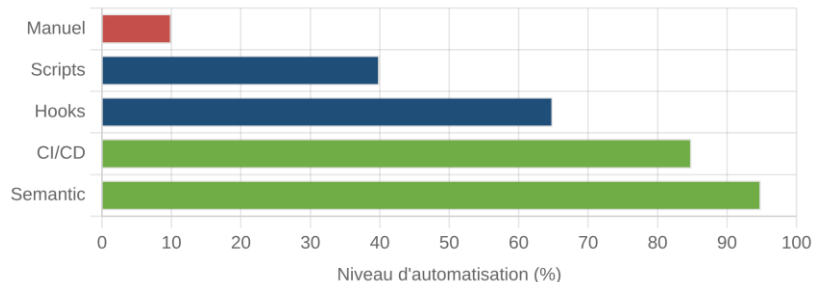
Automatisation du versioning

Techniques d'automatisation

 **Post-commit hooks** : Exécuter des scripts après la construction pour appliquer des tags

 **Versioning sémantique** : Automatiser l'incrémentation selon le type de changement

 **Webhooks** : Déclencher des actions de versioning via des webhooks Git



Exemple de post-commit hook

```
kind: BuildConfig
apiVersion: build.openshift.io/v1
metadata:
  name: "app-build"
spec:
  source:
    git:
      uri: "https://github.com/user/app"
  strategy:
    sourceStrategy:
      from:
        kind: "ImageStreamTag"
        name: "nodejs:14"
    postCommit:
      script: |
        VERSION=$(grep -m1 version package.json | \
          awk -F: '{ print $2 }' | \
          sed 's/["', ]//g')
        oc tag app:latest app:$VERSION
```

Extraction de version depuis Git

```
#!/bin/bash
# Extraire la version depuis les tags Git
GIT_TAG=$(git describe --tags --abbrev=0)

# Appliquer le tag à l'image
oc tag ${IMAGE_STREAM}:latest ${IMAGE_STREAM}:${GIT_TAG}
```

Bonnes pratiques de versioning

Versioning sémantique

Le **versioning sémantique** (SemVer) est une convention de numérotation des versions qui suit le format

MAJEUR.MINEUR.CORRECTIF :

⚠ **MAJEUR** : Changements incompatibles avec les versions précédentes

⊕ **MINEUR** : Ajouts de fonctionnalités rétrocompatibles

🛠 **CORRECTIF** : Corrections de bugs rétrocompatibles

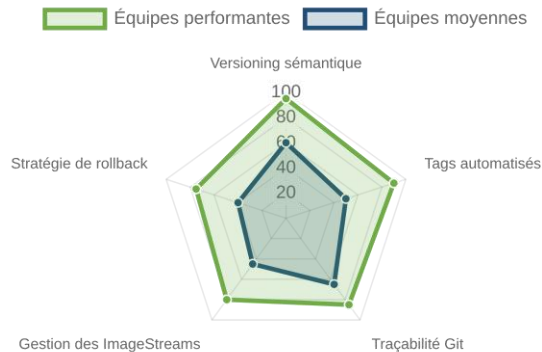
Recommandations

🔗 **Automatisation** : Automatisez le processus de versioning avec des crochets post-commit

🕒 **Traçabilité** : Associez chaque version à un commit Git spécifique

Dans OpenShift, utilisez les **ImageStreams** pour gérer efficacement les différentes versions de vos images. Cela facilite les déploiements, les rollbacks et la promotion entre environnements.

Recommended minimum for annotations:



Adoption des pratiques de versioning

Intégration CI/CD

Intégration des crochets dans un pipeline CI/CD

Les **crochets de version** s'intègrent parfaitement dans un pipeline CI/CD pour automatiser le processus de versioning et garantir la traçabilité des déploiements.



Déclenchement automatique : Les builds sont déclenchés automatiquement par des webhooks Git lors des commits



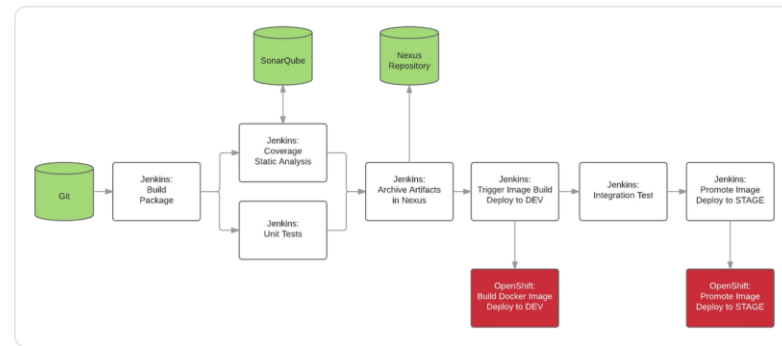
Versioning automatisé : Les post-commit hooks appliquent des tags de version basés sur les conventions de commit



Tests automatisés : Les tests sont exécutés avant l'application des tags pour garantir la qualité



Promotion entre environnements : Les images taguées sont promues automatiquement entre les environnements



Exemple d'intégration

Dans un pipeline CI/CD complet, les crochets de version peuvent être utilisés pour :

Générer des numéros de version basés sur les commits

Appliquer des tags aux images dans le registre

Mettre à jour les manifestes de déploiement

Déclencher des déploiements dans les environnements cibles

L'intégration des crochets de version dans un pipeline CI/CD permet de **standardiser le processus de versioning** d'**améliorer la traçabilité** des déploiements et de **faciliter les rollbacks** en cas de problème. Cette approche s'inscrit dans une démarche DevOps mature et favorise la collaboration entre les équipes de développement et d'exploitation.

Introduction aux travaux pratiques

Mise en pratique des concepts

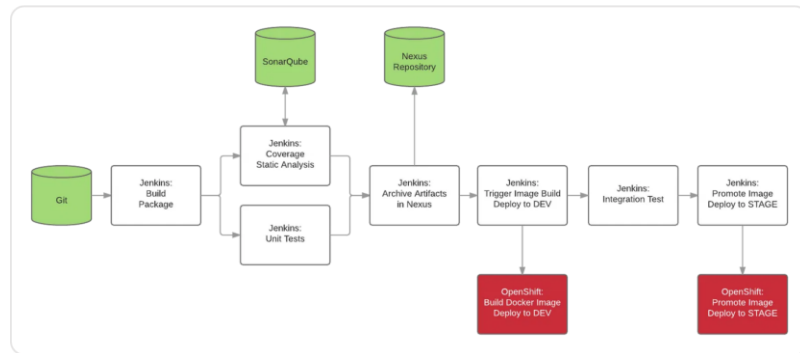
Nous allons mettre en pratique les concepts vus dans ce module à travers un exercice complet sur la **construction OpenShift** et la mise en œuvre de **tags de version après soumission**

Objectifs de l'exercice

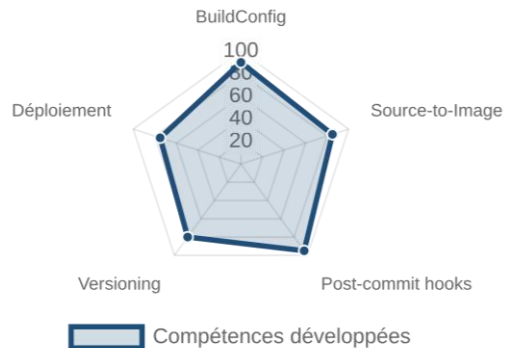
- ✓ Configurer un BuildConfig avec une stratégie S2I pour une application Node.js
- ✓ Implémenter un crochet post-commit pour appliquer des tags de version
- ✓ Automatiser le processus de versioning avec un script
- ✓ Configurer un déploiement pour utiliser les images versionnées

Prérequis

Accès à un cluster OpenShift avec droits d'administration et outils CLI (oc) installés.



Compétences développées dans ce TP



Conclusion et récapitulatif

Points clés du module

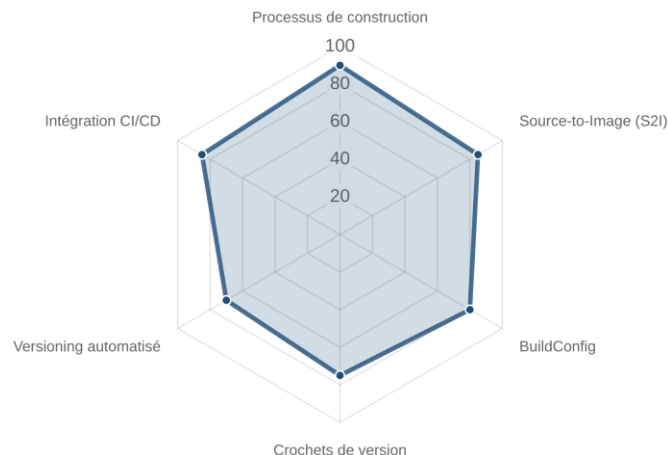
- ⚙️ Le **processus de construction** dans OpenShift transforme votre code source en images de conteneurs prêtes à être déployées.
- 🔗 La technologie **Source-to-Image (S2I)** simplifie la création d'images reproductibles sans nécessiter de Dockerfile.
- 🏷️ Les **crochets post-commit** permettent d'automatiser l'application de tags de version après la construction.
- 🔄 Le **versioning sémantique** (MAJEUR.MINEUR.CORRECTIF) assure une traçabilité claire des changements dans vos applications.

Prochaines étapes

Dans le prochain module, nous explorerons les stratégies avancées de déploiement et la gestion des applications sur OpenShift.

Questions ?

N'hésitez pas à poser vos questions sur les concepts abordés dans ce module ou sur l'exercice pratique à venir.



Compétences acquises dans ce module