Helm

Helm ajoute un langage de création de modèles au-dessus du YAML standard de Kubernetes.

Helm et Helm Chart

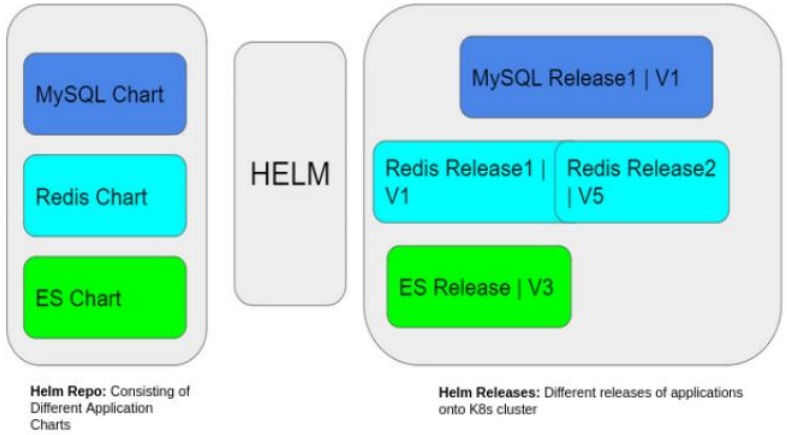
- Helm est un gestionnaire de paquets pour Kubernetes.
- Il permet de déployer des charts. Les charts sont des applications packagées prêtes à être installées.
- Il s'agit d'une collection de toutes nos ressources/manifestes d'application préconfigurés et versionnés qui peuvent être déployés comme une seule unité.
- Helm aide de trois manières principales :
  - Augmente la productivité
  - Réduit la duplication et la complexité des déploiements de microservices
  - Permet l'adaptation des applications cloud natives

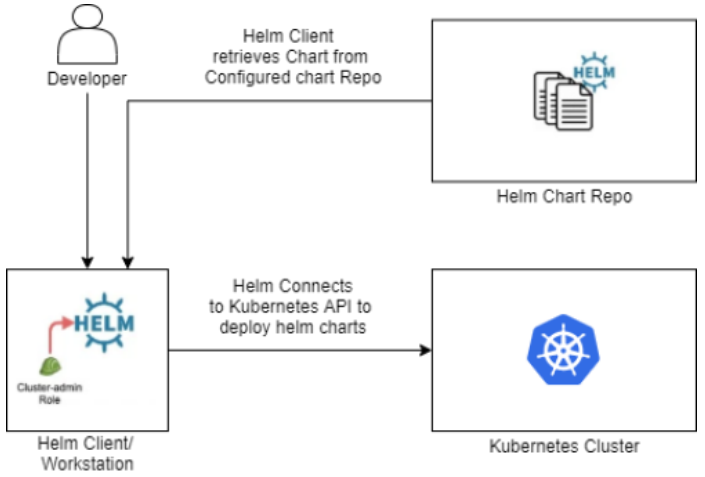
Pourquoi utiliser Helm et les Charts Helm ?

- Tâches fastidieuses avec Kubernetes :
  - Écriture et maintenance des manifests YAML pour chaque déploiement
  - Gestion des dépendances des applications
- Solution :
  - Helm simplifie la gestion des déploiements Kubernetes
  - Création d’un package unique déployable sur le cluster
- Avantages :
  - Emballage et déploiement d’applications avec leurs dépendances
  - Installation simple d’applications (MySQL, Redis, Prometheus, Grafana, etc.) via la commande helm install
  - Désinstallation facile des applications du cluster

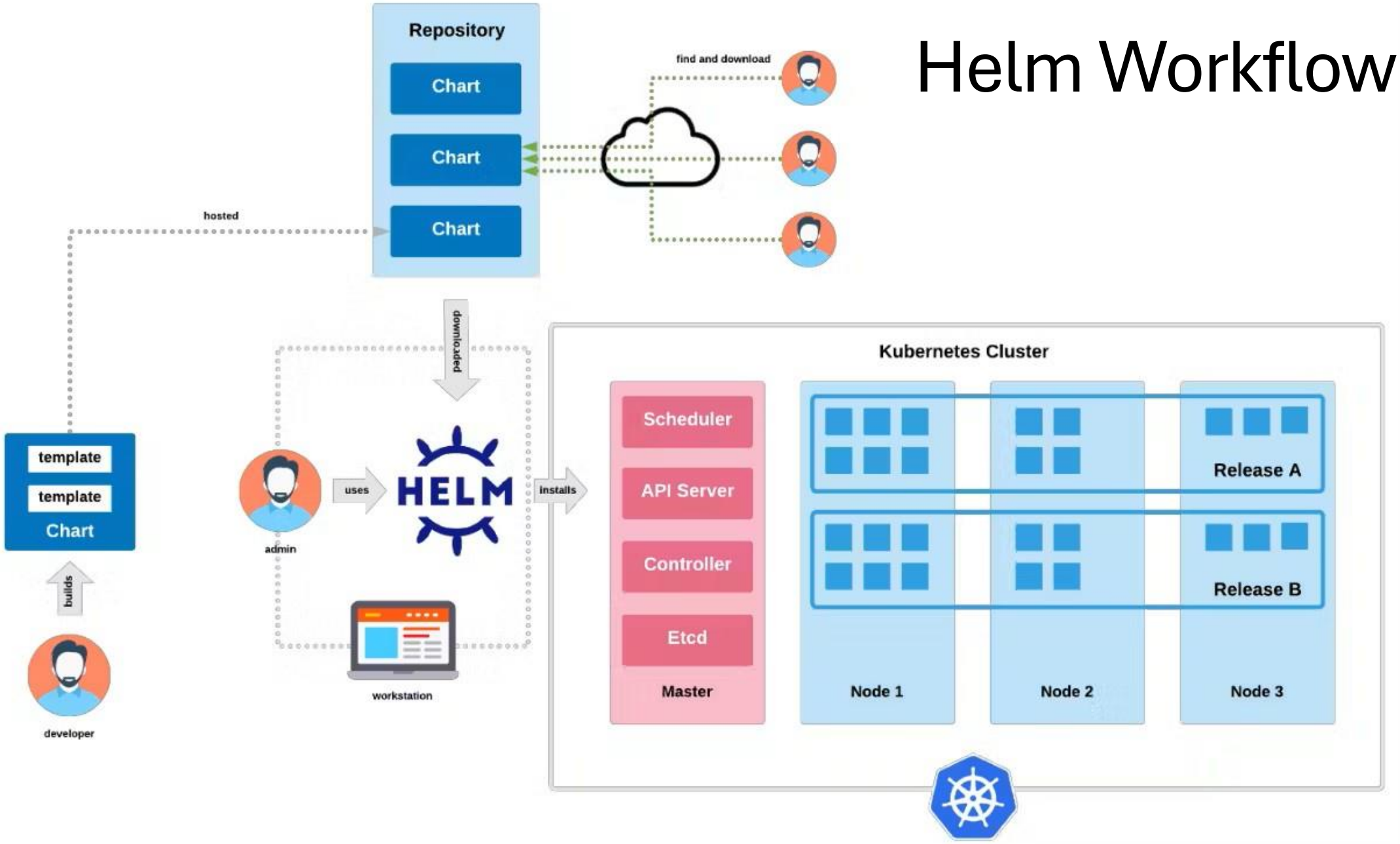
Composants de Helm

- **Helm Client** : **Interface de ligne de commande (CLI)** pour la gestion des charts (installation, màj, suppression de versions).
- **Charts** : **Sont les éléments fondamentaux des packages Helm.**
- Ensemble de fichiers (manifestes, des templates et des métadonnées Kubernetes) regroupés pour définir une application et ses dépendances.
- Permettent aux utilisateurs de créer des packages, de gérer les versions et de partager des applications dans un format standardisé.
- **Repository : Servent d'emplacements centralisés pour le stockage et le partage de charts.**
- Les utilisateurs peuvent publier leurs charts dans des repos ou exploiter les repos existants gérés par la communauté Helm.
- Les repos simplifient le processus de découverte et de distribution des charts, favorisant ainsi la collaboration et la réutilisabilité.
- **Helm Release**: une release Helm est une instance d'un chart exécuté dans un cluster K8s.



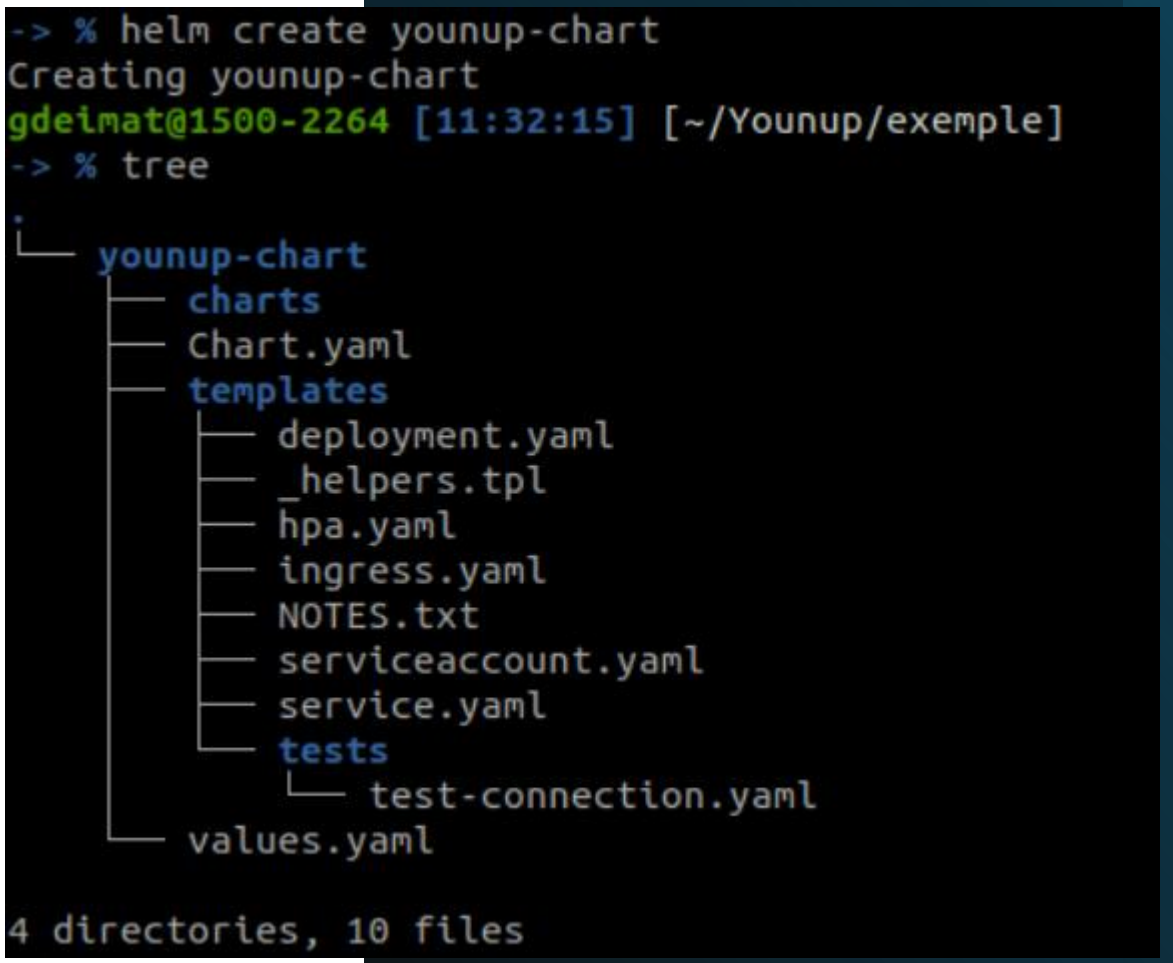


Helm Workflow



Flux de travail de l'architecture Helm

1. **Création du Chart :**
   - Un développeur crée un chart Helm, qui ressemble à un livre de recettes pour une application Kubernetes.
   - Ce chart contient tous les ingrédients (manifests Kubernetes) et les instructions de préparation (templates).
2. **Stockage du Chart :**
   - Le chart est ensuite stocké dans un dépôt de charts.
3. **Préparation à l'installation :**
   - Un utilisateur décide de déployer une application avec Helm.
   - Il récupère le chart souhaité depuis le dépôt.
4. **Personnalisation des valeurs :**
   - L'utilisateur peut personnaliser le chart en modifiant le fichier ‘values.yaml’.
   - Cela revient à ajuster la recette selon les préférences.
5. **Rendu du template :**
   - Helm prend le chart et les valeurs personnalisées, et les exécute dans son moteur de templates.
   - Ce processus est comme un robot cuisinier qui interprète la recette personnalisée.



Flux de travail de l'architecture Helm

6.Génération des manifests Kubernetes :

Le processus de templating produit les manifests Kubernetes finaux.

Ce sont les instructions exactes dont Kubernetes a besoin, comme une liste d'ingrédients et un plan de préparation détaillé.

7.Installation :

Helm envoie ces manifests à Kubernetes.

Kubernetes crée alors toutes les ressources nécessaires (pods, services, etc.).

8.Création de la release :

Helm crée une release, qui est une capture instantanée de ce qui a été installé.

9.Stockage de la release :

Helm stocke les informations sur la release dans son historique de releases.

Cet historique est généralement conservé dans des secrets Kubernetes.

10.Mise à jour / Rétablissement :

Si des changements sont nécessaires, Helm peut mettre à jour la release avec de nouvelles valeurs ou version du chart.

Si quelque chose va mal, Helm peut revenir à une version précédente.

11.Désinstallation :

Lorsqu'elle n'est plus nécessaire, Helm peut désinstaller la release, en nettoyant toutes les ressources Kubernetes associées.

Exemples de Commandes Helm

- Installer un chart :

```
helm install my-release bitnami/nginx
```

- Lister les releases :

```
helm list
```

- Mettre à jour une release :

```
helm upgrade my-release bitnami/nginx
```

- Supprimer une release :

```
helm uninstall my-release
```

```
gdeimat@1500-2264 [14:13:45] [~/Younup/exemple]
-> % helm install my-release oci://registry-1.docker.io/bitnamicharts/drupal
Pulled: registry-1.docker.io/bitnamicharts/drupal:14.1.3
Digest: sha256:1f7d821ae502dbef86c4de7a2f3bb583c30e595541729908059fcd319d8f3835
NAME: my-release
LAST DEPLOYED: Wed May 17 14:15:31 2023
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
CHART NAME: drupal
CHART VERSION: 14.1.3
APP VERSION: 10.0.9** Please be patient while the chart is being deployed **

1. Get the Drupal URL:

  NOTE: It may take a few minutes for the LoadBalancer IP to be available.
        Watch the status with: 'kubectl get svc --namespace default -w my-release-drupal'

  export SERVICE_IP=$(kubectl get svc --namespace default my-release-drupal --template "{{ range (index .status.loadBalancer.ingress 0) }}{{ . }}{{ end }}")
  echo "Drupal URL: http://$SERVICE_IP/"

2. Get your Drupal login credentials by running:
```

Installation d'un package Helm

Pour déployer le chart Helm de la dernière version de Drupal sur son cluster Kubernetes il suffit de lancer la commande

```
helm install my-release oci://registry-1.docker.io/bitnamicharts/drupal
```

Gestion des Variables et des Templates

- Fichier values.yaml :

```
replicaCount: 3
image:
  repository: nginx
  tag: 1.21
```

- Exemple de template dans le fichier deployment.yaml :

```
spec:
  replicas: {{ .Values.replicaCount }}
  containers:
    - name: nginx
      image: "{{
.Values.image.repository }}:{{
.Values.image.tag }}"
```

Pipelines Helm et DevOps

En fonction de la taille de votre entreprise et de votre niveau d'implication avec Helm, vous devez décider quelle pratique vous convient le mieux

Problèmes résolus par Helm :

Gestion des configurations complexes.	Versionnement des déploiements.	Réutilisabilité et partage de modèles (Charts).

Avantages pour les pipelines CI/CD :

Cohérence dans les environnements.	Facilité d'intégration avec les outils DevOps comme Jenkins, GitLab CI/CD, etc.

Architecture d’un pipeline Helm

- Étapes principales :
  - Linting du Chart Helm (Validation syntaxique et règles).
  - Construction des Charts (Versionnage et empaquetage).
  - Tests automatisés (Helm test, ou tests Kubernetes).
  - Déploiement sur l’environnement cible (Dev, Staging, Prod).
  - Rollback en cas de problème.

Intégration avec les outils CI/CD

Exemple avec Jenkins :

- Pipeline Jenkins typique pour Helm :
  - Cloner le dépôt contenant le Chart.
  - Valider avec helm lint.
  - Tester localement ou dans un cluster.
  - Déployer avec helm install ou helm upgrade.

Exemple avec GitLab CI/CD :

- Utilisation de runners pour exécuter les commandes Helm.
- Gestion des secrets avec Vault ou Kubernetes.

Déployer à partir d’un chart non empaqueté

Il s'agit du pipeline le plus simple pour Helm. Le chart Helm se trouve dans le même référentiel Git que le code source de l'application.

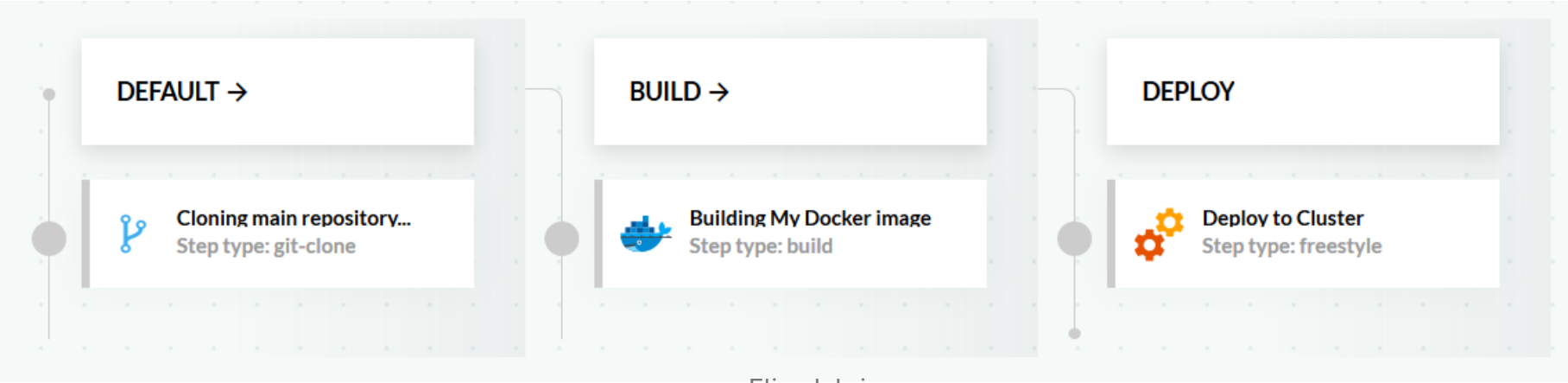
- Les étapes sont les suivantes :
  - Le code/Dockerfile/Chart est extrait de Git
  - L'image Docker est créée (et poussée vers le registre Docker par défaut)
  - Le graphique est déployé directement sur un cluster Kubernetes

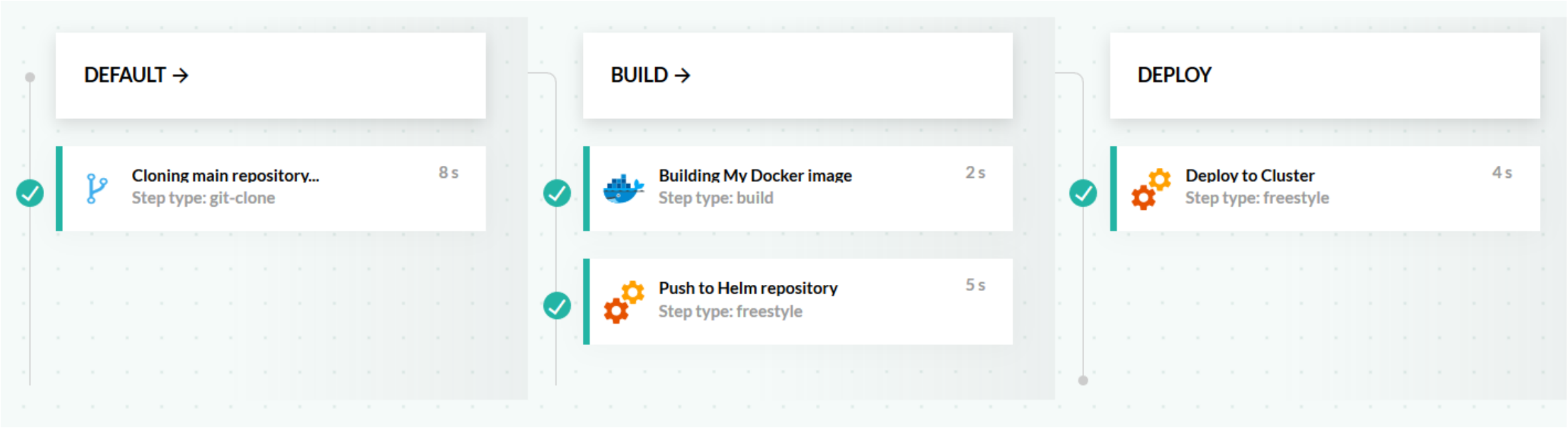
Notez que dans ce pipeline, aucun référentiel Helm n'est impliqué.

Exemple concret - Déploiement avec Helm

Cas d’usage : Application web (front-end + back-end)

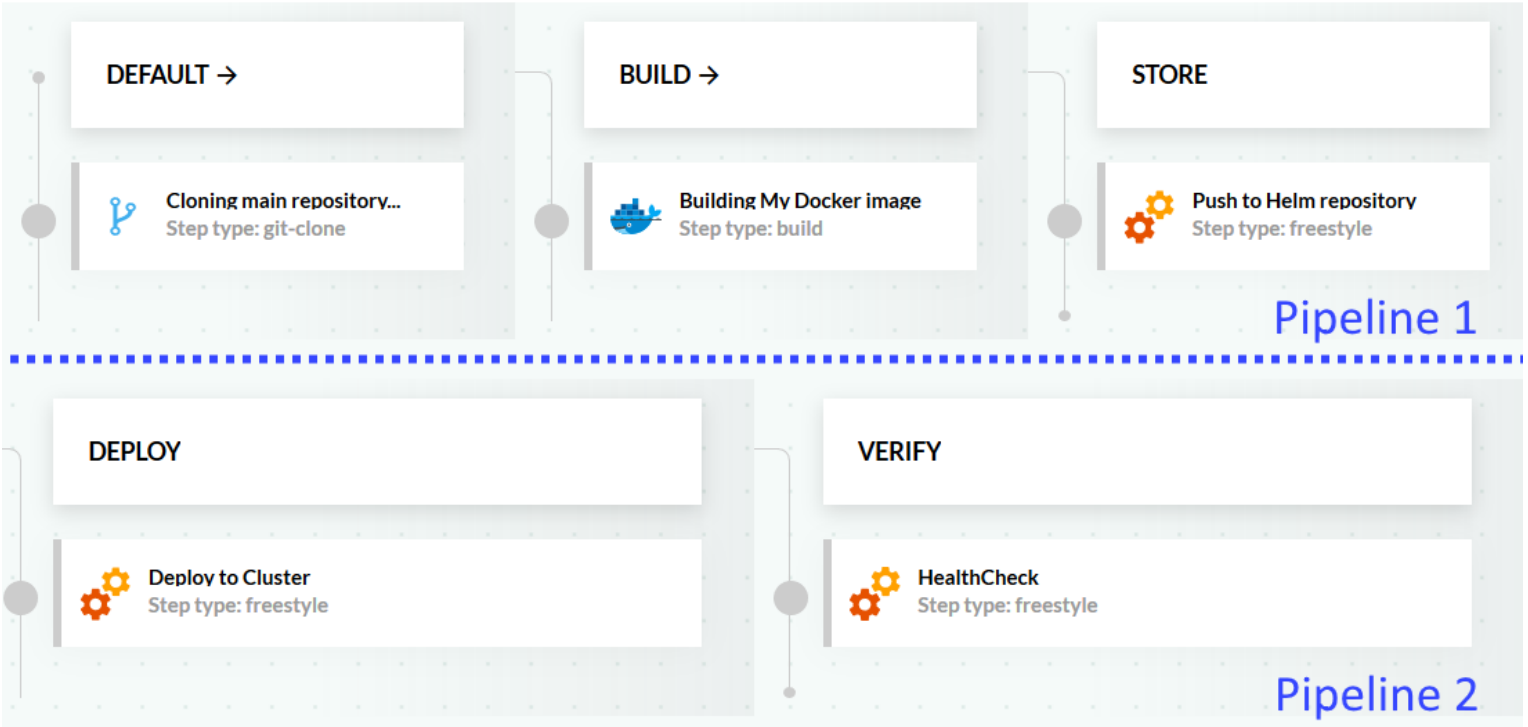
1. Préparation du Chart Helm :
   Structure du Chart (templates/, values.yaml).
2. Pipeline CI/CD :
   `helm lint → helm package → helm install.`
3. Déploiement en staging puis en production.





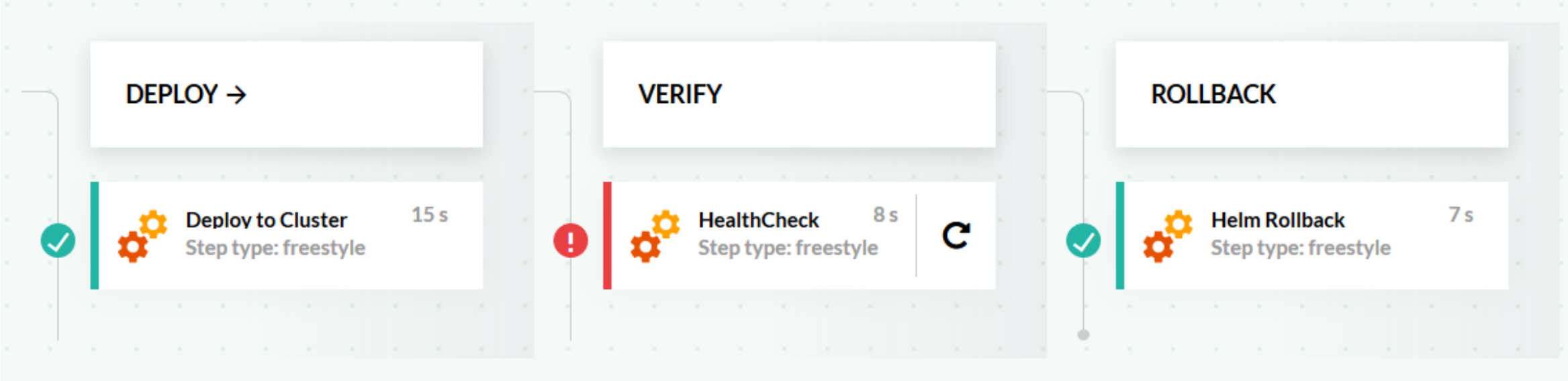
Package/push puis déployer

- Il s'agit de l'approche recommandée lors de l'utilisation de Helm.
- Tout d'abord, vous empaquetez et transférez le chart Helm dans un référentiel, puis vous le déployez sur votre cluster.
- De cette façon, votre référentiel Helm affiche un registre des applications qui s'exécutent sur votre cluster.
- Vous pouvez également réutiliser les charts pour les déployer dans d'autres environnements.



Pipelines Helm séparés

- Plusieurs entreprises ont deux processus différents pour l'empaquetage et la publication.
- Dans ce cas, vous pouvez créer deux pipelines.
- L'un empaquète le chart Helm et le télécharge dans un référentiel Helm, et
- L'autre déploie le chart Helm vers un cluster.



Utilisation des Rollbacks Helm

- Helm a la capacité native de restaurer une version à n'importe quelle révision précédente .

- Une utilisation plus avancée consisterait à annuler automatiquement une version si elle « échoue ».

Promotion des charts entre les repos et les environnements

- La version d’un chart Helm est différente de la version de l'application qu'elle contient.
- Cela signifie que vous pouvez suivre les versions sur le Helm chart lui-même séparément des applications qu'il définit.