

Installation et configuration de Helm

Nous allons voir comment installer Helm CLI sur Linux.

À ce jour, la dernière version de Helm est la 3.17.2. Les exemples présentés dans cet article utilisent la version 3.17.2. Vous pouvez trouver la dernière māj de l'installation sur <https://helm.sh/docs/intro/install/>

Via les gestionnaires de paquets

La communauté Helm permet d'installer Helm via les gestionnaires de paquets du système d'exploitation.

(Debian/Ubuntu)

```
sudo apt-get install curl gpg apt-transport-https --yes
curl -fsSL https://packages.buildkite.com/helm-linux/helm-
debian/gpgkey | gpg --dearmor | sudo tee
/usr/share/keyrings/helm.gpg > /dev/null
echo "deb [signed-by=/usr/share/keyrings/helm.gpg]
https://packages.buildkite.com/helm-linux/helm-debian/any/ any main"
| sudo tee /etc/apt/sources.list.d/helm-stable-debian.list
sudo apt-get update
sudo apt-get install helm
```

dnf/yum (fedora)

```
sudo dnf install helm
```

Configurer Helm avec votre cluster

De la même manière que kubectl est configuré pour communiquer avec un cluster spécifique, Helm utilise également le fichier ‘~/.kube/config’ pour savoir sur quel cluster exécuter les commandes.

```
# To find out which context (and therefore cluster) is currently
being used
kubectl config current-context
# To view the entire ~/.kube/config file
kubectl config view
# View helm charts installed in all namespaces on the cluster
helm list -A
```

```

admuser@k8s-console:~$ kubectl config current-context
kubernetes-admin@kubernetes
admuser@k8s-console:~$ kubectl config view
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: DATA+OMITTED
    server: https://192.168.56.20:6443
    name: kubernetes
contexts:
- context:
    cluster: kubernetes
    user: kubernetes-admin
    name: kubernetes-admin@kubernetes
current-context: kubernetes-admin@kubernetes
kind: Config
preferences: {}
users:
- name: kubernetes-admin
  user:
    client-certificate-data: DATA+OMITTED
    client-key-data: DATA+OMITTED
admuser@k8s-console:~$ helm version
version.BuildInfo{Version:"v3.19.0", GitCommit:"3d8990f0836691f0229297773f3524598f46bda6", GitTreeState:"clean", GoVersion:"go1.24.7"}
admuser@k8s-console:~$ helm list -A
NAME      NAMESPACE   REVISION   UPDATED           STATUS      CHART          APP VERSION
monitoring  monitoring  1          2025-09-03 00:05:04.376396693 +0000 UTC deployed  kube-prometheus-stack-77.3.0  v0.85.0
admuser@k8s-console:~$ helm list -A
NAME      NAMESPACE   REVISION   UPDATED           STATUS      CHART          APP VERSION
monitoring  monitoring  1          2025-09-03 00:05:04.376396693 +0000 UTC deployed  kube-prometheus-stack-77.3.0  v0.85.0

```

Sortie des commandes de configuration kubectl

Ajout et mise à jour des dépôts Helm

Les Helm Charts publiques sont stockées dans les dépôts Helm. Avant d'utiliser une Helm Chart publique, nous devons ajouter certaines métadonnées de ce référentiel à notre Helm local.

```
# Adds a Helm repository locally
helm repo add bitnami https://charts.bitnami.com/bitnami
```

Où la commande *helm repo add* enregistre-t-elle les informations ? Exécutez *helm env*.

```

admuser@k8s-console:~$ helm repo add bitnami https://charts.bitnami.com/bitnami
"bitnami" has been added to your repositories
admuser@k8s-console:~$ helm env
HELM_BIN="helm"
HELM_BURST_LIMIT="100"
HELM_CACHE_HOME="/home/admuser/.cache/helm"
HELM_CONFIG_HOME="/home/admuser/.config/helm"
HELM_DATA_HOME="/home/admuser/.local/share/helm"
HELM_DEBUG="false"
HELM_KUBEAPISERVER=""
HELM_KUBEASGROUPS=""
HELM_KUBEASUSER=""
HELM_KUBECFILE=""
HELM_KUBECONTEXT=""
HELM_KUBEINSECURE_SKIP_TLS_VERIFY="false"
HELM_KUBETLS_SERVER_NAME=""
HELM_KUBETOKEN=""
HELM_MAX_HISTORY="10"
HELM_NAMESPACE="default"
HELM_PLUGINS="/home/admuser/.local/share/helm/plugins"
HELM_QPS="0.00"
HELM_REGISTRY_CONFIG="/home/admuser/.config/helm/registry/config.json"
HELM_REPOSITORY_CACHE="/home/admuser/.cache/helm/repository"
HELM_REPOSITORY_CONFIG="/home/admuser/.config/helm/repositories.yaml"

```

Sortie de helm repo add et helm env

Le chemin indiqué par HELM_REPOSITORY_CONFIG est l'endroit où les informations du référentiel sont sauvegardées.

Pour obtenir la liste de tous les dépôts ajoutés localement, exécutez la commande *helm repo list*. On obtient l'information à partir de la valeur de HELM_REPOSITORY_CONFIG.

Pour ma part, ce sera le dossier /home/admuser/.config/helm/repositories.yaml.

Pour lister tous les Charts du référentiel, exécutez *helm search repo bitnami*.

NAME	URL	CHART VERSION	APP VERSION	DESCRIPTION
bitnami/airflow	https://charts.bitnami.com/bitnami	25.0.2	3.0.5	Apache Airflow is a tool to express and execute...
bitnami/apache		11.4.29	2.4.65	Apache HTTP Server is an open-source HTTP serve...
bitnami/apisix		6.0.0	3.13.0	Apache APISIX is high-performance, real-time AP...
bitnami/appsmith		7.0.3	1.85.0	Appsmith is an open source platform for buildin...
bitnami/argo-cd		11.0.0	3.1.1	Argo CD is a continuous delivery tool for Kuber...
bitnami/argo-workflows		13.0.6	3.7.1	Argo Workflows is meant to orchestrate Kuberne...
bitnami/aspnet-core		7.0.35	9.0.8	ASP.NET Core is an open-source framework for we...
bitnami/cadvisor		0.1.13	0.53.0	cAdvisor (Container Advisor) is an open-source ...
bitnami/cassandra		12.3.11	5.0.5	Apache Cassandra is an open source distributed ...
bitnami/cert-manager		1.5.14	1.18.2	cert-manager is a Kubernetes add-on to automate...

Sortie de helm repo list et helm search repo

En tant que dépôts publics, ils reçoivent régulièrement des mises à jour de la part des contributeurs respectifs. Pour mettre à jour les Charts dans le référentiel, il suffit de mettre à jour l'ensemble du référentiel.

```
# Update a repository
helm repo update bitnami
```

Exploration de la structure d'un Helm Chart

Pour bien comprendre le fonctionnement des Helm Charts, il est important de se familiariser avec leur structure.

Un Helm Chart est plus qu'un simple paquet de fichiers YAML : c'est un répertoire bien organisé qui définit une application Kubernetes en combinant des modèles, une configuration et des métadonnées.

Dans cette section, on va décomposer les composants clés, y compris la disposition générale du répertoire chart, le rôle du fichier *values.yaml* pour la configuration, et comment Helm utilise Go templating pour générer dynamiquement des manifestes Kubernetes.

Structure du répertoire des Charts

Le répertoire des Charts par défaut est le suivant, où demo-helm est le nom du Chart que j'ai défini. Nous le créerons dans la section suivante.

Voyons brièvement à quoi sert chaque fichier ou dossier qui s'y trouve.

```
# demo-helm/
#   ├── .helmignore # Contains patterns to ignore when packaging Helm charts.
#   ├── Chart.yaml # Information about your chart
#   ├── values.yaml # The default values for your templates
#   ├── charts/     # Charts that this chart depends on
#   └── templates/  # The template files
```

- ```
└── tests/ # The test files
```
- Le fichier `.helmignore` fonctionne de la même manière que `.gitignore` ou `.dockerignore`. Si nous voulons exclure certains fichiers ou répertoires de notre Helm Chart, nous pouvons dresser la liste de leurs modèles dans le fichier `.helmignore`. Tous les fichiers ou répertoires qui correspondent à ces modèles seront ignorés lors de l'exécution des commandes Helm.
  - `Chart.yaml` contient des métadonnées sur la carte Helm, telles que le nom de la carte, la description, la version de la carte et la version de l'application.
  - `values.yaml` contient les valeurs de configuration par défaut qui transforment les modèles Go de la Helm Chart en manifestes Kubernetes. Voici quelques valeurs courantes définies dans ce fichier :
    - Le dépôt d'images et la balise.
    - Indicateurs booléens permettant d'activer ou de désactiver des ressources spécifiques.
    - Numéros de port.
    - Autres paramètres spécifiques à l'application.
  - Le répertoire `charts/` contient les dépendances de la carte. Si notre Helm Chart dépend d'autres cartes (appelées cartes secondaires), elles seront placées ici.
  - Le dernier est le dossier `templates/`, qui contient les fichiers modèles Go utilisés pour générer les manifestes Kubernetes. Ces modèles définissent les ressources qui seront déployées, telles que `Deployment`, `ServiceAccount`, `Service`, `Ingress` et `ConfigMap`.

### Comprendre `values.yaml`

Le fichier `values.yaml` est le fichier de configuration par défaut, fournissant un emplacement central pour définir les valeurs d'entrée que les modèles référencent lors du rendu des manifestes Kubernetes.

Lors de l'installation d'un Helm Chart, Helm lit le fichier `values.yaml` et injecte son contenu dans les modèles du Chart via le moteur de création de modèles. Cela nous permet de paramétriser la configuration de l'application, comme les versions des images de conteneurs, les limites de ressources, le nombre de répliques, les paramètres d'entrée, et plus encore, sans modifier directement les modèles.

Voici un extrait de la page `values.yaml` par défaut d'un Helm Chart nouvellement créé :

```
Default values for demo-helm.
This is a YAML-formatted file.
Declare variables to be passed into your templates.

This will set the replicaset count more information can be found here:
https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/
replicaCount: 1

This sets the container image more information can be found here:
https://kubernetes.io/docs/concepts/containers/images/
image:
 repository: nginx
 # This sets the pull policy for images.
 pullPolicy: IfNotPresent
```

```

Overrides the image tag whose default is the chart appVersion.
tag: ""

This is for the secrets for pulling an image from a private repository more
information can be found here: https://kubernetes.io/docs/tasks/configure-pod-
container/pull-image-private-registry/
imagePullSecrets: []
This is to override the chart name.
nameOverride: ""
fullnameOverride: ""

This section builds out the service account more information can be found here:
https://kubernetes.io/docs/concepts/security/service-accounts/
serviceAccount:
 # Specifies whether a service account should be created
 create: true
 # Automatically mount a ServiceAccount's API credentials?
 automount: true

```

Ces valeurs sont référencées dans les fichiers modèles à l'aide de la notation `.Values.replicaCount` ou `.Values.image.repository`. Helm permet de modifier ces valeurs par défaut lors de l'installation ou de la mise à jour, en utilisant un fichier de valeurs personnalisé ou en passant des valeurs individuelles en ligne.

```

Install a helm chart with the default values from values.yaml
helm install demo-helm ./demo-helm

Values defined in custom-values.yaml will override those in values.yaml.
Nonetheless, values in values.yaml not found in custom-values.yaml will
still be applied.
"-f" flag is short for the "--values" flag
helm install demo-helm ./demo-helm -f custom-values.yaml

Overwrite default values using the "--set" flag
helm install demo-helm ./demo-helm --set replicaCount=3 --set
image.tag=latest

```

## Modélisation avec les modèles Go

L'une des fonctionnalités les plus puissantes de Helm est son moteur de templating, qui permet une configuration dynamique et réutilisable des manifestes Kubernetes. Les modèles Go sont au cœur de cette fonctionnalité, et le même langage de création de modèles est utilisé dans de nombreux autres outils et frameworks basés sur Go, tels que [Terraform](#) et [Docker](#).

Au lieu de coder les valeurs directement dans nos fichiers YAML, Helm nous permet de définir des modèles qui font référence à des entrées dynamiques - le plus souvent à partir de `values.yaml`. Par exemple, dans le modèle `deployment` ci-dessous :

```

apiVersion: apps/v1
kind: Deployment
metadata:
 name: {{ .Values.name }}
spec:
 replicas: {{ .Values.replicas }}

```

`{{ .Values.name }}` et `{{ .Values.replicas }}` sont des expressions du modèle Go. Où `.Values` fait référence soit au contenu du fichier `values.yaml`, soit au fichier défini dans l'indicateur `--values`, soit à des valeurs passées en ligne. Ces espaces réservés sont remplacés par des valeurs réelles lorsque le Chart est rendu.

Par exemple, le modèle `deployment` ci-dessous utilise des valeurs telles que `replicaCount` et `image.repository`.

```

! values.yaml U ×
helm-charts > demo-helm > ! values.yaml > ...
1 # Default values for demo-helm.
2 # This is a YAML-formatted file.
3 # Declare variables to be passed into your templates.
4
5 # This will set the replicaset count more information can be found here: https://...
6 replicaCount: 1
7
8 # This sets the container image more information can be found here: https://...
9 image:
10 repository: nginx
11 # This sets the pull policy for images.
12 pullPolicy: IfNotPresent
13 # Overrides the image tag whose default is the chart appVersion.
14 tag: ""
15
16 # This is for the secrets for pulling an image from a private repository more info...
17 imagePullSecrets: []
18 # This is to override the chart name.
19 nameOverride: ""
20 fullnameOverride: ""
21
22 # This section builds out the service account more information can be found here:...
23 serviceAccount:
24 # Specifies whether a service account should be created
25 create: true
26 # Automatically mount a ServiceAccount's API credentials?
27 automount: true
28 # Annotations to add to the service account
29 annotations: {}
30 # The name of the service account to use.
31 # If not set and create is true, a name is generated using the fullname template
32 name: ""
33

! deployment.yaml U ×
helm-charts > demo-helm > templates > ! deployment.yaml > [map] > apiVersion > apps/v1
1 apiVersion: apps/v1
2 kind: Deployment
3 metadata:
4 name: {{ include "demo-helm.fullname" . }}
5 labels:
6 {{- include "demo-helm.labels" . | nindent 4 }}
7 spec:
8 {{- if not .Values.autoscaling.enabled }}
9 replicas: {{ .Values.replicaCount }}
10 {{- end }}
11 selector:
12 matchLabels:-
13 template:
14 metadata:-
15 spec:
16 {{- with .Values.imagePullSecrets }}
17 imagePullSecrets:-
18 {{- end }}
19 serviceAccountName: {{ include "demo-helm.serviceAccountName" . }}
20 {{- with .Values.podSecurityContext }}
21 securityContext:-
22 {{- end }}
23 containers:
24 - name: {{ .Chart.Name }}
25 {{- with .Values.securityContext }}
26 securityContext:
27 {{- toYaml . | nindent 12 }}
28 {{- end }}
29 image: "{{ .Values.image.repository }}{{ .Values.image.tag }}| default .Chart.AppVersi...
30 imagePullPolicy: {{ .Values.image.pullPolicy }}
31 ports:
32 - name: http
33 containerPort: {{ .Values.service.port }}
34
35
36
37
38
39
40
41
42
43
44
45

```

### Mapping entre `values.yaml` et `deployment.yaml`

Le modèle `serviceAccount` utilise quant à lui `serviceAccount.create`. La totalité de la ressource `serviceAccount` sera omise si la valeur est fausse.

```

! values.yaml U ×
helm-charts > demo-helm > ! values.yaml > ...
19 nameOverride: ""
20 fullnameOverride: ""
21
22 # This section builds out the service account more information can be found here:...
23 serviceAccount:
24 # Specifies whether a service account should be created
25 create: true
26 # Automatically mount a ServiceAccount's API credentials?
27 automount: true
28 # Annotations to add to the service account
29 annotations: {}
30 # The name of the service account to use.
31 # If not set and create is true, a name is generated using the fullname template
32 name: ""
33

! serviceaccount.yaml U ×
helm-charts > demo-helm > templates > ! serviceaccount.yaml > [map]
1 {{- if .Values.serviceAccount.create -}}
2 apiVersion: v1
3 kind: ServiceAccount
4 metadata:
5 name: {{ include "demo-helm.serviceAccountName" . }}
6 labels:
7 {{- include "demo-helm.labels" . | nindent 4 }}
8 {{- with .Values.serviceAccount.annotations }}
9 annotations:
10 {{- toYaml . | nindent 4 }}
11 {{- end }}
12 automountServiceAccountToken: {{ .Values.serviceAccount.automount }}
13 {{- end }}
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45

```

### Correspondance entre `values.yaml` et `serviceaccount.yaml`

Pour illustrer davantage, disons que vous avez la configuration suivante dans votre `values.yaml`.

```

replicaCount: 2
image:
 repository: nginx
 tag: "1.21.1"

```

Votre modèle `deployment.yaml` (sous `templates/deployment.yaml`) pourrait ressembler à ceci :

```

apiVersion: apps/v1
kind: Deployment
metadata:
 name: {{ .Release.Name }}
spec:

```

```

replicas: {{ .Values.replicaCount }}
selector:
 matchLabels:
 app: {{ .Release.Name }}
template:
 metadata:
 labels:
 app: {{ .Release.Name }}
spec:
 containers:
 - name: nginx
 image: "{{ .Values.image.repository }}:{{ .Values.image.tag }}"
 ports:
 - containerPort: 80

```

Lorsque Helm rend le manifeste Kubernetes, soit pendant helm template, helm install, soit pendant helm upgrade, il le fait en injectant les valeurs de values.yaml.

```
helm template <release name> <chart name OR path to chart folder>
```

```
helm template my-nginx <chart name OR path to chart folder>
```

Nous obtiendrons alors le manifeste ci-dessous :

```

apiVersion: apps/v1
kind: Deployment
metadata:
 name: my-nginx
spec:
 replicas: 2
 selector:
 matchLabels:
 app: my-nginx
 template:
 metadata:
 labels:
 app: my-nginx
 spec:
 containers:
 - name: nginx
 image: "nginx:1.21.1"
 ports:
 - containerPort: 80

```

## **Création de votre première carte de barre**

La façon la plus simple d'apprendre Helm est de construire nous-mêmes une carte. Nous allons parcourir la création d'un diagramme de base, comprendre sa structure et le personnaliser pour déployer notre application sur Kubernetes.

### **En utilisant helm create**

Au lieu d'utiliser les cartes Helm publiques des dépôts Helm, que nous avons abordées plus haut dans la sous-section "Ajouter et mettre à jour les dépôts Helm", créons-en une nous-mêmes.

```
Creates a helm chart with the name "demo-helm"
helm create demo-helm
```

La structure du répertoire sera la suivante :

```
demo-helm/
├── .helmignore # Contains patterns to ignore when packaging Helm charts.
├── Chart.yaml # Information about your chart
├── values.yaml # The default values for your templates
└── charts/ # Charts that this chart depends on
└── templates/ # The template files
└── tests/ # The test files
```

```
~/Documents/GitLab/demo-eks-helm/helm-charts [main !4 ?2] ...
[> ls -l
total 0

~/Documents/GitLab/demo-eks-helm/helm-charts [main !10 ?3] ...
[> helm create demo-helm
Creating demo-helm

~/Documents/GitLab/demo-eks-helm/helm-charts [main !10 ?4] ...
[> ls -l
total 0
drwxr-xr-x 7 kennyang staff 224 Apr 10 22:36 demo-helm

~/Documents/GitLab/demo-eks-helm/helm-charts [main !10 ?4] ...
[> cd demo-helm

~/Documents/GitLab/demo-eks-helm/helm-charts/demo-helm [main]
[> tree -L 2
.
├── Chart.yaml
└── templates
 ├── NOTES.txt
 ├── _helpers.tpl
 ├── deployment.yaml
 ├── hpa.yaml
 ├── ingress.yaml
 ├── service.yaml
 └── serviceaccount.yaml
 └── tests
└── values.yaml

4 directories, 9 files
```

Résultat de la commande `helm create`

## Personnalisation des modèles

Explorons le contenu du dossier `templates` pour mieux le comprendre avant de le modifier.

```
~/Documents/GitLab/demo-eks-helm/helm-charts/demo-helm/templates [main !10 ?4]
[> ls -l
total 56
-rw-r--r-- 1 kennyang staff 1752 Apr 10 22:36 NOTES.txt
-rw-r--r-- 1 kennyang staff 1802 Apr 10 22:36 _helpers.tpl
-rw-r--r-- 1 kennyang staff 2390 Apr 10 22:36 deployment.yaml
-rw-r--r-- 1 kennyang staff 997 Apr 10 22:36 hpa.yaml
-rw-r--r-- 1 kennyang staff 1094 Apr 10 22:36 ingress.yaml
-rw-r--r-- 1 kennyang staff 367 Apr 10 22:36 service.yaml
-rw-r--r-- 1 kennyang staff 393 Apr 10 22:36 serviceaccount.yaml
drwxr-xr-x 3 kennyang staff 96 Apr 10 22:36 tests
```

Templates of  
Kubernetes  
Resources

Contenu du dossier du modèle

Le fichier `_helpers.tpl` contient des aides de modèle Go réutilisables (semblables à des fonctions) qui peuvent être appelées à partir d'autres modèles dans le diagramme, ce qui favorise la réutilisation et la maintenance du code.

L'image ci-dessous montre comment les sections du fichier `_helpers.tpl` correspondent au fichier `service.yaml`.

```

_helpers.tpl U >
helm-charts > demo-helm > templates > _helpers.tpl > ...
13 {{- define "demo-helm.fullname" -}}
14 {{- if .Values.fullnameOverride -}}
15 {{- .Values.fullnameOverride | trunc 63 | trimSuffix "-" -}}
16 {{- else -}}
17 {{- $name := default .Chart.Name .Values.nameOverride -}}
18 {{- if contains $name .Release.Name -}}
19 {{- .Release.Name | trunc 63 | trimSuffix "-" -}}
20 {{- else -}}
21 {{- printf "%s-%s" .Release.Name $name | trunc 63 | trimSuffix "-" -}}
22 {{- end -}}
23 {{- end -}}
24 {{- end -}}
25
26 /* Create chart name and version as used by the chart label.
27 */
28 {{- define "demo-helm.chart" -}}
29 {{- printf "%s-%s" .Chart.Name .Chart.Version | replace "+" "_" | trunc 63 | trimSuffix "-" -}}
30 {{- end -}}
31
32 /* Common labels */
33 {{- define "demo-helm.labels" -}}
34 helm.sh/chart: {{ include "demo-helm.chart" . }}
35 {{- include "demo-helm.selectorLabels" . }}
36 {{- if .Chart.AppVersion -}}
37 app.kubernetes.io/version: {{ .Chart.AppVersion | quote -}}
38 {{- end -}}
39 app.kubernetes.io/managed-by: {{ .Release.Service -}}
40 {{- end -}}
41
42 /* Selector labels */
43 {{- define "demo-helm.selectorLabels" -}}
44 app.kubernetes.io/name: {{ include "demo-helm.name" . }}
45 app.kubernetes.io/instance: {{ .Release.Name -}}
46 {{- end -}}
47
48
49
50
51

```

```

service.yaml U >
helm-charts > demo-helm > templates > service.yaml > (map) > apiVersion
1 apiVersion: v1
2 kind: Service
3 metadata:
4 name: {{ include "demo-helm.fullname" . }}
5 labels:
6 {{- include "demo-helm.labels" . | indent 4 -}}
7 spec:
8 type: {{ .Values.service.type -}}
9 ports:
10 - port: {{ .Values.service.port -}}
11 targetPort: http
12 protocol: TCP
13 name: http
14 selector:
15 {{- include "demo-helm.selectorLabels" . | indent 4 -}}
16

```

#### Mapping entre `_helpers.tpl` et `service.yaml`

Nous pouvons exécuter la commande `helm template` pour obtenir le rendu `service.yaml`. La commande valide que le Chart ne contient pas d'erreurs et rend les fichiers de manifeste.

```
helm template <release name> <chart name OR path to chart folder>
helm template demo-helm ./demo-helm
```

```

~/Documents/GitLab/demo-eks-helm/helm-charts main !4 ?3
> helm template demo-helm ./demo-helm

Source: demo-helm/templates/serviceaccount.yaml
apiVersion: v1
kind: ServiceAccount
metadata:
 name: demo-helm
 labels:
 helm.sh/chart: demo-helm-0.1.0
 app.kubernetes.io/name: demo-helm
 app.kubernetes.io/instance: demo-helm
 app.kubernetes.io/version: "1.16.0"
 app.kubernetes.io/managed-by: Helm
automountServiceAccountToken: true

Source: demo-helm/templates/service.yaml
apiVersion: v1
kind: Service
metadata:
 name: demo-helm
 labels:
 helm.sh/chart: demo-helm-0.1.0
 app.kubernetes.io/name: demo-helm
 app.kubernetes.io/instance: demo-helm
 app.kubernetes.io/version: "1.16.0"
 app.kubernetes.io/managed-by: Helm
spec:
 type: ClusterIP
 ports:
 - port: 80
 targetPort: http
 protocol: TCP
 name: http
 selector:
 app.kubernetes.io/name: demo-helm
 app.kubernetes.io/instance: demo-helm

Source: demo-helm/templates/deployment.yaml
apiVersion: apps/v1
kind: Deployment

```

### Sortie de la commande `helm template`

Nous pouvons compléter ou modifier le contenu du fichier `_helpers.tpl` ou de l'un des modèles de ressources pour répondre à nos besoins. Par exemple, pour ajouter une étiquette de sélection à la ressource service via le fichier `_helpers.tpl`:

```

{{- define "demo-helm.selectorLabels" -}}
app.kubernetes.io/name: {{ include "demo-helm.name" . }}
app.kubernetes.io/instance: {{ .Release.Name }}
app.kubernetes.io/custom: "testing" # Add this line
{{- end }}

```



```

! _helpers.tpl U X
helm-charts > demo-helm > templates > ! _helpers.tpl > ...
46 Selector labels
47 */
48 {{- define "demo-helm.selectorLabels" -}}
49 app.kubernetes.io/name: {{ include "demo-helm.name" . }}
50 app.kubernetes.io/instance: {{ .Release.Name }}
51 app.kubernetes.io/custom: "testing" # Add this line
52 {{- end }}
53

```

Ajouter une étiquette de sélection dans `_helpers.tpl`

Les guillemets utilisés autour de la valeur testing doivent être des guillemets droits ("testing") et non des guillemets frisés ("testing"). Bien que helm template ne nous envoie pas d'erreur, helm install le fera.

- Enregistrez le fichier \_helpers.tpl et exécutez à nouveau la commande helm template. Vous devriez voir le sélecteur ajouté à la ressource de service.

```
~/Documents/GitLab/demo-eks-helm/helm-charts main !4 ?3 ...
> helm template demo-helm ./demo-helm
Source: demo-helm/templates/serviceaccount.yaml
apiVersion: v1
kind: ServiceAccount
metadata:
 name: demo-helm
 labels:
 helm.sh/chart: demo-helm-0.1.0
 app.kubernetes.io/name: demo-helm
 app.kubernetes.io/instance: demo-helm
 app.kubernetes.io/custom: "testing" # Add this line
 app.kubernetes.io/version: "1.16.0"
 app.kubernetes.io/managed-by: Helm
automountServiceAccountToken: true

Source: demo-helm/templates/service.yaml
apiVersion: v1
kind: Service
metadata:
 name: demo-helm
 labels:
 helm.sh/chart: demo-helm-0.1.0
 app.kubernetes.io/name: demo-helm
 app.kubernetes.io/instance: demo-helm
 app.kubernetes.io/custom: "testing" # Add this line
 app.kubernetes.io/version: "1.16.0"
 app.kubernetes.io/managed-by: Helm
spec:
 type: ClusterIP
 ports:
 - port: 80
 targetPort: http
 protocol: TCP
 name: http
 selector:
 app.kubernetes.io/name: demo-helm
 app.kubernetes.io/instance: demo-helm
 app.kubernetes.io/custom: "testing" # Add this line

Source: demo-helm/templates/deployment.yaml
apiVersion: apps/v1
kind: Deployment
```

*Mise à jour de la sortie de la commande helm template*

Remarquez qu'en plus d'être ajouté au sélecteur, il a également été ajouté aux étiquettes de toutes les ressources. En effet, le site {{- define "demo-helm.labels" -}} contient le site {{ include "demo-helm.selectorLabels" . }}. Ainsi, tout ce qui se trouve sur le site selectorLabels sera inclus dans les étiquettes .

Un autre exemple est que nous pouvons modifier le site deployment.yaml pour exposer un autre port sur le conteneur. Par défaut, lorsque nous créons un Chart à partir de la commande helm create, le modèle de déploiement n'expose qu'un port sur le conteneur.

- Ajoutez la configuration suivante au site deployment.yaml.

```

- name: https
 containerPort: 443
 protocol: TCP

```

```

deployment.yaml U ×
1 apiVersion: apps/v1
2 spec:
3 replicas: {{ .Values.replicaCount }}
4 template:
5 metadata:
6 securityContext:
7 {{- toYaml . | nindent 12 }}
8 {{- end }}
9 image: "{{ .Values.image.repository }}:{{ .Values.image.tag }}"
10 imagePullPolicy: {{ .Values.image.pullPolicy }}
11 ports:
12 - name: http
13 containerPort: {{ .Values.service.port }}
14 protocol: TCP
15 - name: https
16 containerPort: 443 Added this configuration
17 protocol: TCP
18 {{- with .Values.livenessProbe }}
19 livenessProbe:
20 {{- toYaml . | nindent 12 }}
21 {{- end }}
22 {{- with .Values.readinessProbe }}
23 readinessProbe:
24 {{- toYaml . | nindent 12 }}
25 {{- end }}
26 {{- with .Values.resources }} ...

```

*Ajout d'un port au conteneur dans deployment.yaml*

Pour des raisons de simplicité, la valeur de containerPort est codée en dur à 443. Dans la sous-section suivante, nous verrons comment utiliser le site values.yaml pour le paramétriser .

- Enregistrez le fichier deployment.yaml et exécutez à nouveau la commande helm template. Le port doit maintenant être ajouté au conteneur dans la ressource de déploiement.

Comme l'indentation est cruciale dans le formatage YAML, le modèle pour les ressources Kubernetes inclut la fonction nindent pour indenter les blocs requis en conséquence.

```

apiVersion: v1
kind: Service
metadata:
 name: {{ include "demo-helm.fullname" . }}
 labels:
 {{- include "demo-helm.labels" . | nindent 4 }}
spec:
 type: {{ .Values.service.type }}
 ports:
 - port: {{ .Values.service.port }}
 targetPort: http
 protocol: TCP
 name: http
 selector:
 {{- include "demo-helm.selectorLabels" . | nindent 4 }}

```

### Définition des valeurs par défaut

Maintenant que nous avons couvert la majeure partie du contenu du dossier template, nous sommes prêts à passer à values.yaml.

```

~/Documents/GitLab/demo-eks-helm/helm-charts/demo-helm [main !4 ?3] ...
> tree -L 2
.
├── Chart.yaml
├── charts
├── templates
│ ├── NOTES.txt
│ ├── _helpers.tpl
│ ├── deployment.yaml
│ ├── hpa.yaml
│ ├── ingress.yaml
│ ├── service.yaml
│ ├── serviceaccount.yaml
│ └── tests
└── values.yaml ←

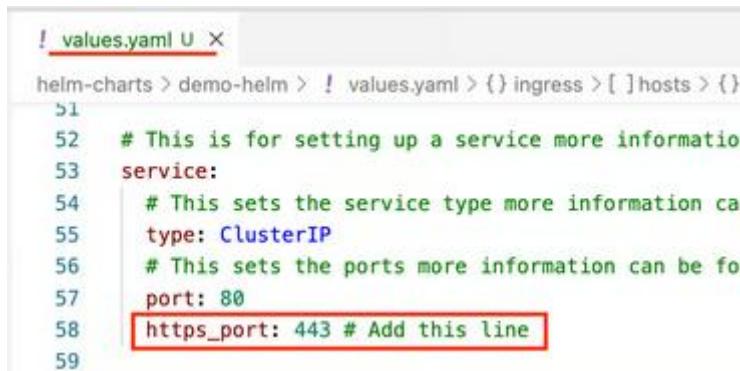
4 directories, 9 files

```

### Répertoire des cartes de barre

Dans la sous-section précédente, nous avons codé en dur la valeur de containerPort à 443. Paramétrons-le.

- Dans values.yaml, sous la clé service, ajoutez la clé imbriquée https\_port avec la valeur 443.



```

! values.yaml U X
helm-charts > demo-helm > ! values.yaml > {} ingress > []hosts > {}
51
52 # This is for setting up a service more information
53 service:
54 # This sets the service type more information can
55 type: ClusterIP
56 # This sets the ports more information can be fou
57 port: 80
58 https_port: 443 # Add this line
59

```

### Ajouter une clé imbriquée

- Modifiez le site deployment.yaml pour remplacer le code en dur 443 par {{ .Values.service.port }}.



```

! deployment.yaml U X
helm-charts > demo-helm > templates > ! deployment.yaml > (e){map}
1 apiVersion: apps/v1
7 spec:
9 replicas: {{ .Values.replicaCount }}
14 template:
15 metadata:
42 imagePullPolicy: {{ .Values.image.pullPolicy }}
43 ports:
44 - name: http
45 containerPort: {{ .Values.service.port }}
46 protocol: TCP
47 - name: https
48 containerPort: {{ .Values.service.https_port }} ←
49 protocol: TCP
50 {{- with .Values.livenessProbe }}
51 livenessProbe:
52 {{- toYaml . | nindent 12 }}

```

### Paramétriser le numéro de port

- Pour valider et rendre les ressources Kubernetes :

```
helm template <release name> <chart name OR path to chart folder>
helm template demo-helm ./demo-helm
```

Vous devriez voir que la valeur 443 est correctement définie sur le site containerPort.

```
spec:
 serviceAccountName: demo-helm
 containers:
 - name: demo-helm
 image: "nginx:1.16.0"
 imagePullPolicy: IfNotPresent
 ports:
 - name: http
 containerPort: 80
 protocol: TCP
 - name: https
 containerPort: 443
 protocol: TCP
 livenessProbe:
 httpGet:
 path: /
```

*Valeur du port https rendu*

Si d'autres endroits du tableau de bord nécessitent la valeur de ce containerPort, utilisez simplement {{ .Values.service.https\_port }}.

### Déploiement et gestion des versions

Maintenant que nous avons créé notre propre diagramme Helm, l'étape suivante consiste à le déployer sur un cluster Kubernetes.

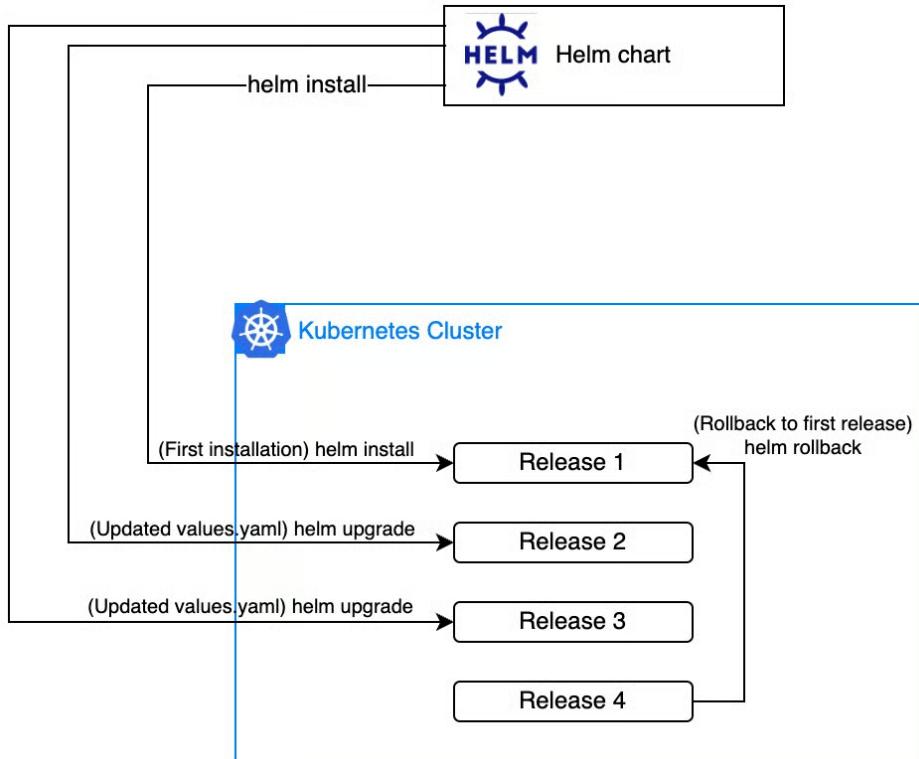
Helm gère le cycle de vie des ressources Kubernetes déployées en tant que version. Dans cette section, nous allons explorer comment Helm gère les déploiements, les mises à niveau, les retours en arrière et le suivi de l'historique des versions, ce qui nous donne un contrôle et une visibilité puissants sur nos charges de travail Kubernetes.

#### Qu'est-ce qu'une libération ?

Dans Helm, une version est une **instance spécifique** d'une Helm Chart qui a été déployée dans un cluster Kubernetes.

Chaque fois que nous installons un Chart à l'aide de helm install, Helm crée une nouvelle version avec son propre nom, sa propre configuration et sa propre version. De même, lorsque nous mettons à niveau une version à l'aide de helm upgrade, Helm crée une nouvelle révision de cette version avec le même nom, mais la configuration et la version peuvent être différentes. Essentiellement, une version est une nouvelle version de la même version avec des paramètres ou un contenu Chart mis à jour.

Ainsi, lorsque nous disons "installer une carte Helm", nous faisons en réalité référence à la "création d'une version Helm".



*Illustration visuelle des rejets - image de l'auteur.*

Bien que seule la révision la plus récente d'une version soit activement déployée dans le cluster, Helm conserve l'historique complet des révisions précédentes. Cela signifie que nous pouvons facilement revenir à un état antérieur en utilisant `helm rollback`.

Lorsque nous effectuons un rollback, Helm ne restaure pas l'ancienne version telle quelle, mais crée une nouvelle révision qui reproduit la configuration de la version précédente sélectionnée. Chaque révision est numérotée et les retours en arrière incrémentent le numéro de révision, ce qui permet de conserver un historique cohérent et vérifiable.

### Installation d'une carte avec `helm install`

Voyons comment déployer une version de Helm sur un cluster Kubernetes. Nous allons installer deux diagrammes Helm : celui que nous avons créé dans la section précédente et le diagramme bitnami/apache du dépôt public Bitnami que nous avons déjà ajouté à notre configuration Helm locale.

| helm search repo bitnami |               |             |                                                    |
|--------------------------|---------------|-------------|----------------------------------------------------|
| NAME                     | CHART VERSION | APP VERSION | DESCRIPTION                                        |
| bitnami/airflow          | 22.7.2        | 2.10.5      | Apache Airflow is a tool to express and execute... |
| bitnami/apache           | 11.3.5        | 2.4.63      | Apache HTTP Server is an open-source HTTP serve... |
| bitnami/apisix           | 4.2.2         | 3.12.0      | Apache APISIX is high-performance, real-time AP... |
| bitnami/appsmith         | 5.2.2         | 1.64.0      | Appsmith is an open source platform for buildin... |
| bitnami/argo-cd          | 7.3.4         | 2.14.9      | Argo CD is a continuous delivery tool for Kuber... |

Tableau Helm bitnami/apache

Lors de l'installation d'une carte, nous devons spécifier un nom de publication - qui peut être n'importe quoi et ne doit pas nécessairement correspondre au nom de la carte. Toutefois, il est recommandé d'utiliser un nom étroitement lié au Chart afin de permettre aux utilisateurs de comprendre plus facilement ce que le communiqué représente en un coup d'œil.

- Tout d'abord, pour la Helm Chart que nous avons créée :

```
Install the helm chart (in other words, create a helm release) into the
default namespace
helm install <release name> <chart name OR path to chart folder>
helm install datacamp-demo-helm ./demo-helm
OR
helm upgrade --install datacamp-demo-helm ./demo-helm

If installing to a specific namespace, add the -n flag
Install the chart to the "app" namespace
helm install my-amazing-helm-demo ./demo-helm -n app
```

```
~/Documents/GitLab/demo-eks-helm/helm-charts [main !4 ?3] . . .
[> helm install datacamp-demo-helm ./demo-helm
NAME: datacamp-demo-helm
LAST DEPLOYED: Thu Apr 10 23:26:21 2025
NAMESPACE: default
STATUS: deployed
REVISION: 1
NOTES:
1. Get the application URL by running these commands:
 export POD_NAME=$(kubectl get pods --namespace default -l "app.kubernetes.io/name=demo-helm,app.kub
 export CONTAINER_PORT=$(kubectl get pod --namespace default $POD_NAME -o jsonpath="{.spec.container
 echo "Visit http://127.0.0.1:8080 to use your application"
 kubectl --namespace default port-forward $POD_NAME 8080:$CONTAINER_PORT
```

Déployer un Helm Chart personnalisé

```
To validate the Kubernetes resources
Get pods, service, and serviceaccount resources in the default
namespace
kubectl get pods,svc,sa
```

```
~/Documents/GitLab/demo-eks-helm/helm-charts [main !4 ?3] . . .
[> k get pods,svc,sa
NAME READY STATUS RESTARTS AGE
pod/datacamp-demo-helm-6bf5c9f89-cdgwl 1/1 Running 0 2m9s

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE
service/datacamp-demo-helm ClusterIP 10.105.156.171 <none> 80/TCP 2m9s
service/kubernetes ClusterIP 10.96.0.1 <none> 443/TCP 31d

NAME SECRETS AGE
serviceaccount/datacamp-demo-helm 0 2m9s
serviceaccount/default 0 31d
```

*Valider les ressources déployées à partir de la Helm Chart personnalisée*

- Deuxièmement, pour le Chart bitnami/apache:

```
Install the helm chart into the default namespace
helm install apache bitnami/apache
OR
helm upgrade --install apache bitnami/apache
```

```

~ .: > helm install apache bitnami/apache
NAME: apache
LAST DEPLOYED: Tue Apr 8 09:30:54 2025
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
CHART NAME: apache
CHART VERSION: 11.3.5
APP VERSION: 2.4.63

Did you know there are enterprise versions of the Bitnami catalog? For enhanced secure software supply chain features, unlimited pulls from Docker, LTS support, or applicat
or Tanzu Application Catalog. See https://www.arrow.com/globalecs/na/vendors/bitnami for more information.

** Please be patient while the chart is being deployed **

1. Get the Apache URL by running:

** Please ensure an external IP is associated to the apache service before proceeding **
** Watch the status using: kubectl get svc --namespace default -w apache **

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

WARNING: You did not provide a custom web application. Apache will be deployed with a default page. Check the README section "Deploying your custom web application" in http
/bitnami/apache/README.md#deploying-a-custom-web-application.

WARNING: There are "resources" sections in the chart not set. Using "resourcesPreset" is not recommended for production. For production installations, please set the follow
eeds:
- resources
+info https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/
```

### Résultat de l'installation de helm

```

~ .: > helm upgrade --install apache bitnami/apache
Release "apache" does not exist. Installing it now.
NAME: apache
LAST DEPLOYED: Tue Apr 8 09:34:59 2025
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
CHART NAME: apache
CHART VERSION: 11.3.5
APP VERSION: 2.4.63

Did you know there are enterprise versions of the Bitnami catalog? For enhanced secure software supply chain features, unlimited pulls from Docker, LTS support, or applicat
or Tanzu Application Catalog. See https://www.arrow.com/globalecs/na/vendors/bitnami for more information.

** Please be patient while the chart is being deployed **

1. Get the Apache URL by running:

** please ensure an external IP is associated to the apache service before proceeding **
** Watch the status using: kubectl get svc --namespace default -w apache **

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

WARNING: You did not provide a custom web application. Apache will be deployed with a default page. Check the README section "Deploying your custom web application" in http
/bitnami/apache/README.md#deploying-a-custom-web-application.

WARNING: There are "resources" sections in the chart not set. Using "resourcesPreset" is not recommended for production. For production installations, please set the follow
eeds:
- resources
+info https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/
```

### Sortie de helm upgrade --install

Pour afficher les versions de Helm qui ont été déployées :

```
List all helm releases in all namespaces
helm list -A
~/Documents/GitLab/demo-eks-helm/helm-charts [main !4 ?3] .: > helm list -A
NAME NAMESPACE REVISION UPDATED STATUS CHART APP VERSION
apache default 1 2025-04-10 23:32:19.028379 +0800 +08 deployed apache-11.3.5 2.4.63
datacamp-demo-helm default 1 2025-04-10 23:42:03.559833 +0800 +08 deployed demo-helm-0.1.0 1.16.0
```

### Liste de tous les communiqués de Helm

### Mise à niveau du gouvernail

Après le déploiement initial d'une carte Helm, les mises à jour sont inévitables, qu'il s'agisse de changer la balise de l'image, d'ajuster le nombre de répliques, d'activer ou de désactiver des ressources optionnelles ou de modifier les valeurs de configuration.

Helm propose la commande `helm upgrade` pour appliquer ces changements à une version existante sans avoir à la redéployer à partir de zéro. Cette commande nous permet de mettre à jour notre application en cours d'exécution en re-rendant les modèles avec la nouvelle configuration et en appliquant les changements au cluster Kubernetes, en gérant de manière transparente les mises à jour avec un minimum d'interruption.

L'application déployée via notre diagramme de barre personnalisé exécute une image nginx avec la balise 1.16.0. Nous pouvons déterminer la balise en exécutant la commande `helm get manifest`.

```
Output the rendered Kubernetes manifest that was installed by the
helm chart
helm get manifest <release name>
helm get manifest datacamp-demo-helm
```

Dans le manifeste de la ressource deployment, vous devriez voir ce qui suit :

```
spec:
 serviceAccountName: datacamp-demo-helm
 containers:
 - name: demo-helm
 image: "nginx:1.16.0"
 imagePullPolicy: IfNotPresent
 ports:
 - name: http
 containerPort: 80
 protocol: TCP
 - name: https
```

#### Valeur de l'image

D'où vient l'étiquette 1.16.0? Notre valeur `image.tag` in `values.yaml` est vide.

Examinez le modèle `deployment.yaml`: la clé `image` a pour valeur " `"{{ .Values.image.repository }}:{{ .Values.image.tag | default .Chart.AppVersion }}"`. Lorsque le `image.tag` n'est pas défini dans le `values.yaml`, Helm obtiendra la valeur du `appVersion` définie dans le `Charts.yaml`.

```
! Chart.yaml U X
helm-charts > demo-helm > ! Chart.yaml > apiVersion
1 apiVersion: v2
2 name: demo-helm
3 description: A Helm chart for Kubernetes
4
5 # A chart can be either an 'application' or a 'library'
6 #
7 # Application charts are a collection of templates that
8 # to be deployed.
9 #
10 # Library charts provide useful utilities or functions
11 # a dependency of application charts to inject those into
12 # pipeline. Library charts do not define any template
13 type: application
14
15 # This is the chart version. This version number should
16 # go to the chart and its templates, including the app version
17 # Versions are expected to follow Semantic Versioning
18 version: 0.1.0
19
20 # This is the version number of the application being
21 # incremented each time you make changes to the application
22 # follow Semantic Versioning. They should reflect the
23 # It is recommended to use it with quotes.
24 appVersion: "1.16.0"
25
```

### *appVersion dans Chart.yaml*

- Mettons à jour l'adresse values.yaml de notre Chart personnalisé, et mettons à jour l'adresse image.tag en "1.26".

```
! values.yaml U X
helm-charts > demo-helm > ! values.yaml > {} image > tag
7
8 # This sets the container image more information can be found here:
9 image:
10 repository: nginx
11 # This sets the pull policy for images.
12 pullPolicy: IfNotPresent
13 # Overrides the image tag whose default is the chart appVersion.
14 tag: "1.26"
15
```

### *Mise à jour de image.tag de "" à "1.26"*

- Ensuite, exécutez la commande helm upgrade.

```
helm upgrade <release name> <chart name or path to chart>
helm upgrade datacamp-demo-helm ./demo-helm
```

```
Alternatively, you can update the value via the --set flag
helm upgrade datacamp-demo-helm ./demo-helm --set
```

```
~/Documents/GitLab/demo-eks-helm/helm-charts [main !4 ?3]
> helm upgrade datacamp-demo-helm ./demo-helm
Release "datacamp-demo-helm" has been upgraded. Happy Helming!
NAME: datacamp-demo-helm
LAST DEPLOYED: Fri Apr 11 00:03:01 2025
NAMESPACE: default
STATUS: deployed
REVISION: 2
NOTES:
1. Get the application URL by running these commands:
 export POD_NAME=$(kubectl get pods --namespace default -l "app.kubernetes.io/name=datacamp-demo-helm")
 export CONTAINER_PORT=$(kubectl get pod --namespace default $POD_NAME -o jsonpath='{.status.containerStatuses[0].containerID}' | sed -e 's@^.*://@@' | awk '{print $NF}')
 echo "Visit http://127.0.0.1:$CONTAINER_PORT to use your application"
 kubectl --namespace default port-forward $POD_NAME 8080:$CONTAINER_PORT
```

*La sortie de la commande helm upgrade*

Examinez la ressource de déploiement ; la balise image doit être 1.26. En coulisses, lors d'une mise à niveau, Helm effectue les opérations suivantes :

- Rendu des modèles:** Helm redessine les modèles du Chart en utilisant les valeurs (mises à jour) transmises avec la commande.
- Déterminer les changements:** Helm compare les nouveaux manifestes rendus avec les manifestes actuels dans le cluster. Dans la dernière section de l'article, nous étudierons la possibilité de vérifier les modifications avant de les appliquer.
- Mise à jour sélective:** Seules les ressources qui ont changé (comme deployment, configmaps ou secrets ) sont mises à jour.
- Suivi de l'historique des versions:** Helm enregistre la nouvelle version et la configuration, ce qui nous permet de revenir en arrière si nécessaire.

```
To view the release history. It also shows the revision numbers.
helm history <release name>
helm history datacamp-demo-helm
```

```
~/Documents/GitLab/demo-eks-helm/helm-charts [main !4 ?3]
[> helm history datacamp-demo-helm
REVISION UPDATED STATUS CHART APP VERSION DESCRIPTION
1 Fri Apr 11 00:02:12 2025 superseded demo-helm-0.1.0 1.16.0 Install complete
2 Fri Apr 11 00:03:01 2025 deployed demo-helm-0.1.0 1.16.0 Upgrade complete
```

*Sortie de la commande helm history*

### Retour en arrière avec helm rollback

Nous venons de mettre à jour une version de Helm, mais peu de temps après, nous avons remarqué que l'application ne se comportait pas comme prévu. Bien que le dépannage soit important, dans un environnement de production, la priorité immédiate est souvent de stabiliser le système. C'est là qu'intervient la fonction de retour en arrière de Helm.

Helm permet de revenir facilement à une version précédente, dont la qualité est connue, à l'aide d'une seule commande, ce qui nous permet de récupérer rapidement les déploiements qui ont échoué et de minimiser les temps d'arrêt.

- Nous allons annuler la mise à niveau effectuée ci-dessus.

```
The first argument of the rollback command is the name of a
release, and the second is a revision (version) number. If this
```

argument is omitted or set to 0, it will roll back to the previous release.

```
helm rollback <release name> <revision number>
helm rollback datacamp-demo-helm 0
~/Documents/GitLab/demo-eks-helm/helm-charts [main !4 ?3] ...
[> helm rollback datacamp-demo-helm 0
Rollback was a success! Happy Helming!
```

```
~/Documents/GitLab/demo-eks-helm/helm-charts [main !4 ?3] ...
[> helm history datacamp-demo-helm
REVISION UPDATED STATUS CHART APP VERSION DESCRIPTION
1 Fri Apr 11 00:02:12 2025 superseded demo-helm-0.1.0 1.16.0 Install complete
2 Fri Apr 11 00:03:01 2025 superseded demo-helm-0.1.0 1.16.0 Upgrade complete
3 Fri Apr 11 00:07:56 2025 deployed demo-helm-0.1.0 1.16.0 Rollback to 1
```

*La sortie de la commande helm rollback*

Examinez le déploiement datacamp-demo-helm, et vous devriez voir l'image nginx renvoyée à 1.16.0.

```
kubectl get deploy datacamp-demo-helm -oyaml
```

### Désinstallation d'une version

Pour désinstaller la version Helm, en d'autres termes, pour supprimer toutes les ressources Kubernetes déployées à partir de la version, exécutez la commande helm uninstall.

```
helm uninstall <release name>
helm uninstall datacamp-demo-helm
helm uninstall apache
~/Documents/GitLab/demo-eks-helm/helm-charts [main !4 ?3] ...
[> helm uninstall datacamp-demo-helm
release "datacamp-demo-helm" uninstalled

~/Documents/GitLab/demo-eks-helm/helm-charts [main !4 ?3] ...
[> helm list -A
NAME NAMESPACE REVISION UPDATED STATUS CHART APP VERSION
apache default 1 2025-04-10 23:32:19.028379 +0800 +08 deployed apache-11.3.5 2.4.63
```

*Résultat de la commande helm uninstall*

### Personnalisation des tableaux de bord pour différents environnements

L'un des principaux atouts de Helm est sa flexibilité. Qu'il s'agisse d'un déploiement en développement, en staging ou en production, Helm permet de personnaliser facilement le comportement de notre Chart par le biais de la configuration.

Dans cette section, nous verrons comment adapter un diagramme Helm à différents environnements à l'aide d'outils tels que values.yaml, de dérogations spécifiques à l'environnement et des drapeaux --set et -f de Helm.

### Utilisation de plusieurs fichiers values.yaml

Helm offre un moyen flexible de gérer les configurations spécifiques à l'environnement à l'aide de plusieurs fichiers values.yaml .

Au lieu de maintenir un site values.yaml unique et complexe qui tente de prendre en compte tous les paramètres possibles pour dev, staging et production, ou des instructions conditionnelles complexes dans nos modèles pour prendre en compte les différents environnements, nous pouvons les répartir dans des fichiers distincts, chacun adapté à un environnement spécifique.

Par exemple, notre répertoire de Charts peut comprendre les éléments suivants

```
my-chart/
├── values.yaml # default values
├── values-dev.yaml # development-specific overrides
├── values-staging.yaml # staging-specific overrides
└── values-prod.yaml # production-specific overrides
```

### Remplacer les valeurs au moment de l'installation

Helm nous permet de remplacer les valeurs par défaut définies dans values.yaml en utilisant soit le drapeau --set pour les remplacements en ligne, soit le drapeau -f pour fournir un fichier de valeurs externe. Le drapeau -f est l'abréviation du drapeau --values.

Pour installer ou mettre à jour notre version de Helm avec l'option -f :

```
For installing on dev
helm install <release name> <chart name or path to chart> -f values-
dev.yaml

For upgrading on dev
helm upgrade <release name> <chart name or path to chart> -f values-
dev.yaml

For installing on staging
helm install <release name> <chart name or path to chart> -f values-
staging.yaml

For upgrading on staging
helm upgrade <release name> <chart name or path to chart> -f values-
staging.yaml
```

Pour installer ou mettre à jour notre version de Helm avec le drapeau --set :

```
For installation
helm install <release name> <chart name or path to chart> --set
image.tag=1.26
```

```
For upgrading
helm upgrade <release name> <chart name or path to chart> --set
image.tag=1.26
```

Si --set et -f sont tous deux utilisés lors de l'installation (ou de la mise à jour), Helm applique les valeurs dans l'ordre suivant :

1. La base values.yaml (si présente)
2. Valeurs du fichier spécifié avec -f
3. Les valeurs en ligne transmises avec --set

Cela signifie que si la même clé apparaît à la fois dans le fichier -f et dans l'indicateur --set, c'est la valeur de --set qui prévaut .

Par exemple, si nous avons défini image.tag comme étant 1.22 dans values-dev.yaml, mais que nous avons défini image.tag comme étant 1.26 via le drapeau --set, la valeur image.tag qui sera utilisée pour rendre le manifeste Kubernetes sera 1.26.

```
E.g. we defined image.tag to be 1.22 in values-dev.yaml
helm install <release name> <chart name OR path to chart> --set
image.tag=1.26 -f values-dev.yaml
```

### Gestion des secrets

Lorsque vous déployez des applications sur Kubernetes à l'aide de Helm, la manipulation de données sensibles, telles que les clés API, les identifiants de base de données ou les certificats TLS, nécessite une attention particulière. Bien que Helm prenne en charge la ressource Kubernetes Secret, les valeurs de la ressource secrète sont simplement encodées en base64, ce qui peut être décodé par n'importe qui.

```
apiVersion: v1
kind: Secret
metadata:
 name: my-secret
 namespace: default
type: Opaque
data:
 key1: c3VwZXJzZWNyZXQ= # ← This is base64 encoded
 key2: dG9wc2VjcmV0 # ← This is base64 encoded
Decode the value of key1
echo "c3VwZXJzZWNyZXQ=" | base64 -d

Decode the value of key2
echo "dG9wc2VjcmV0" | base64 -d
```

Le stockage de secrets directement dans un fichier values.yaml ou en texte brut dans notre tableau **n'est pas sécurisé et doit être évité**, en particulier lorsque les fichiers sont soumis à un contrôle de version.

Pour gérer les secrets en toute sécurité dans les diagrammes Helm dans différents environnements, envisagez d'utiliser des outils spécialement conçus à cet effet. Deux solutions largement adoptées sont les secrets scellés et les secrets externes.

### Secrets scellés

Sealed Secrets est un projet open-source développé par Bitnami qui aide à gérer les données sensibles dans Kubernetes de manière sécurisée et adaptée à GitOps. Il introduit une définition de ressource personnalisée (CRD) appelée SealedSecret, qui nous permet de chiffrer les secrets avant de les déposer dans un dépôt Git.

Contrairement aux ressources Kubernetes secrets ordinaires, qui stockent les données en codage base64 et ne sont pas chiffrées, les ressources SealedSecret contiennent des valeurs chiffrées qui ne peuvent être déchiffrées que par le contrôleur Sealed Secrets s'exécutant dans le cluster Kubernetes cible. Cela signifie que les secrets chiffrés peuvent être stockés en toute sécurité dans des systèmes de contrôle de version tels que Git.

Pour configurer notre cluster et notre environnement local afin d'utiliser Sealed Secrets :

- Installez le CRD SealedSecret et le contrôleur côté serveur (qui fonctionne comme un pod) dans l'espace de noms kube-system.

```
kubectl apply -f https://github.com/bitnami-labs/sealed-
secrets/releases/download/v0.29.0/controller.yaml
#OR
helm install sealed-secret bitnami/sealed-secrets -n kube-system
```

```
> kubectl apply -f https://github.com/bitnami-labs/sealed-secrets/releases/download/v0.29.0/controller.yaml
deployment.apps/sealed-secrets-controller created
customresourcedefinition.apiextensions.k8s.io/sealedsecrets.bitnami.com created
service/sealed-secrets-controller created
role.rbac.authorization.k8s.io/sealed-secrets-key-admin created
serviceaccount/sealed-secrets-controller created
rolebinding.rbac.authorization.k8s.io/sealed-secrets-service-proxier created
role.rbac.authorization.k8s.io/sealed-secrets-service-proxier created
service/sealed-secrets-controller-metrics created
rolebinding.rbac.authorization.k8s.io/sealed-secrets-controller created
clusterrolebinding.rbac.authorization.k8s.io/sealed-secrets-controller created
clusterrole.rbac.authorization.k8s.io/secrets-unsealer created
```

*Installez SealedSecret sur notre cluster via kubectl*

```
> helm install sealed-secret bitnami/sealed-secrets -n kube-system
NAME: sealed-secret
LAST DEPLOYED: Tue Apr 15 21:51:26 2025
NAMESPACE: kube-system
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
CHART NAME: sealed-secrets
CHART VERSION: 2.5.9
APP VERSION: 0.29.0

Did you know there are enterprise versions of the Bitnami catalog? For enhanced secure software supply chain features, unlimited pu
um or Tanzu Application Catalog. See https://www.arrow.com/globalecs-na/vendors/bitnami for more information.

** Please be patient while the chart is being deployed **

Watch the SealedSecret controller status using the command:
 kubectl get deploy -w --namespace kube-system -l app.kubernetes.io/name=sealed-secrets,app.kubernetes.io/instance=sealed-secret
Once the controller is up and ready, you should be able to create sealed secrets.
```

*Installez SealedSecret sur notre cluster via Helm*

- Vérifiez les ressources :

```
[> k get deploy,secrets,crd -n kube-system
NAME READY UP-TO-DATE AVAILABLE AGE
deployment.apps/coredns 2/2 2 2 32d
deployment.apps/sealed-secrets-controller 1/1 1 1 5h59m

NAME TYPE DATA AGE
secret/sealed-secrets-keyvb269 kubernetes.io/tls 2 5h59m

NAME CREATED AT
customresourcedefinition.apiextensions.k8s.io/sealedsecrets.bitnami.com 2025-04-12T03:18:18Z
```

*Vérifier les ressources en installant les secrets scellés*

- Installez le CLI kubeseal sur notre système local.

```
Linux x86_64:
curl -OL "https://github.com/bitnami-labs/sealed-
secrets/releases/download/v0.29.0/kubeseal-0.29.0-linux-
amd64.tar.gz"
tar -xvzf kubeseal-0.29.0-linux-amd64.tar.gz kubeseal
sudo install -m 755 kubeseal /usr/local/bin/kubeseal

macOS
brew install kubeseal
```

Pour créer la ressource SealedSecret CRD:

```

Pass the secret manifest to kubeseal for it to generate the
SealedSecret CRD resource containing the encrypted secrets. Saves
the SealedSecret CRD manifest in sealed-mysecret.yaml.
kubectl create secret generic mysecret --dry-run=client --from-
literal key1=supersecret --from-literal key2=topsecret --output json
| kubeseal | tee sealed-mysecret.yaml

Or if you have the secret manifest saved in a JSON file
kubeseal -f mysecret.yaml -w sealed-mysecret.yaml
> kubectl create secret generic mysecret --dry-run=client --from-literal key1=supersecret --from-literal key2=topsecret --output json | kubeseal | tee sealed-mysecret.yaml
{
 "kind": "SealedSecret",
 "apiVersion": "bitnami.com/v1alpha1",
 "metadata": {
 "name": "mysecret",
 "namespace": "default",
 "creationTimestamp": null
 },
 "spec": {
 "template": {
 "metadata": {
 "name": "mysecret",
 "namespace": "default",
 "creationTimestamp": null
 }
 },
 "encryptedData": {
 "key1": "AgB1Ua/G$JVBqAsz/uM2Fa5VlsYfb04Z8B9hSf8+2csDz1oC0S0Bc0uCgWKGdD>Dg24F7XsVP9PoEYjHbs81mUSVWwLWxJvJFMGjHxa24hn1CLT8q1Q1Vry-PvKkQvJcbkTFm0t424uYsRTLnI05Hcvosq33p1IwOPV1HBMye/VA2Re-/jQf4GrATWjq1fIFhA6vxno1ZQ7L5ZpzsCaF5gYHwwmMTbZ8hzjPsejUxj8r3phUA5gR0uAvt0x3w1p2rMjuA+y+o24H1q10R0wrvf1gJ1xT3El1vNhuAdgMdcaApwKtkvbwJ68mYHNV1LHWsv8pNTbvhpfmZzrY1CmbQ78PrTqClccn0eA32V5PcfQUm+yc28MN/tw.9TkcLq0xGv4x3x2NSeF9CoLEEz8oYMoRYW6KRjvfz38ee1r811htknKcIye0Pw&t4ems+19Wmm+Ab4qKEfegxU/nDF8+s40hb2tsM+1wkvJWFYFnqYosLC/N07Vluaoa6Dz1B09nP28+87ohYN+PK08vozsCDjWDhsIS0mxxZaJzJQTDF-GU+C4Qltnz3Rx51Nxzw==",
 "key2": "AgDrechT8b9VVF1kq1BsrlpmM12wxDGp0G19L9F+0Mz5Xcm0Gr1e5yHyQoJxDZFx25BSFGfIUoR316gePtckP34910a0ULCzdSVt3LGQs7G48TSbk1plqlqULWE5UJEPyAhxZFMlYB9p/uuQ7JDvmJsTn3s63PwqyCHt1GpA+aYcUQdpCxylKukz19zxx0ce6yVe5y8faZsJ1Y0h0uH5MBvHIApmDZK217paZoghuHf5fsexVc7txLu/aubcASC8MvXUhAjcrVtYHMaFp/p1Lcgv057VATG88buYmfLh/Cz0GpX6xFxus+2BQs2zyLpdYsocAiRL0fjp-Jxj2JCS19c31w/WnnWzsvxuHOjB4XFYDv+r746PHB80XcnVi:zboz2Q8VD6sJf3nnX0RDbxlpkryisnfwCbwyMSLhbbCEH6GSz+Efk9TSEU35XsghwIE6xjnH88fy23TQFzQ1SXoz1dCJgOX1BCtM1oC3mrL0knR8un0VdTazBze6RDKj1MuIrqH132VcF5fp2ae/zHhmY0Ex800BLY6/UwMCfnpG2blfxEPsca4s3JUg4="
 }
 }
}

```

### Générer et enregistrer la ressource CRD SealedSecret

Appliquez le fichier sealed-mysecret.yaml. Lorsqu'une ressource SealedSecret est créée dans le cluster, le contrôleur déchiffre automatiquement les données scellées et crée une ressource Kubernetes secrets standard à partir de celles-ci.

```

~/Documents/GitLab/demo-eks-helm [main !4 ?5]
[> k get secrets
No resources found in default namespace.

~/Documents/GitLab/demo-eks-helm [main !4 ?5]
[> k create -f sealed-mysecret.yaml
sealedsecret.bitnami.com/mysecret created

~/Documents/GitLab/demo-eks-helm [main !4 ?5]
[> k get SealedSecret
NAME AGE
mysecret 10s

~/Documents/GitLab/demo-eks-helm [main !4 ?5]
[> k get secrets
NAME TYPE DATA AGE
mysecret Opaque 2 13s

~/Documents/GitLab/demo-eks-helm [main !4 ?5]
[> k get secrets mysecret -oyaml
apiVersion: v1
data:
 key1: c3VwZXJzZWNyZXQ=
 key2: dG9wc2VjcmV0
kind: Secret
metadata:
 creationTimestamp: "2025-04-12T13:26:01Z"
 name: mysecret
 namespace: default
 ownerReferences:
 - apiVersion: bitnami.com/v1alpha1
 controller: true
 kind: SealedSecret
 name: mysecret
 uid: 05706373-aa26-4558-b05f-367070d173ce
 resourceVersion: "54027"
 uid: 2f132D4-50b7-4715-982c-60373622fc28
 type: Opaque

```

## Créer la ressource CRD SealedSecret

Cette approche garantit que les informations sensibles restent sécurisées tout au long du pipeline CI/CD.

## Opérateur de Secrets Externes (OSE)

External Secrets Operator (ESO) est un opérateur Kubernetes qui nous permet de récupérer et d'injecter en toute sécurité dans Kubernetes Secrets des secrets provenant de systèmes de gestion de secrets externes, tels que AWS Secrets Manager, HashiCorp Vault, Google Secret Manager, Azure Key Vault, et d'autres.

Au lieu de créer ou de mettre à jour manuellement les secrets Kubernetes, ESO récupère les dernières valeurs secrètes de ces fournisseurs externes via leurs API et s'assure que notre cluster Kubernetes dispose toujours d'informations d'identification ou de données sensibles à jour.

L'ESO introduit un ensemble de ressources personnalisées -ExternalSecret et ClusterSecretStore- qui font abstraction des complexités liées à l'interaction avec des backends secrets externes. Ces ressources définissent les secrets à récupérer et leur provenance, ce qui facilite la gestion des cycles de vie des secrets d'une manière déclarative et native pour Kubernetes.

Pour cet exemple, nous allons récupérer un secret à partir d'AWS Secrets Manager.

- Tout d'abord, créez un rôle IAM qui sera utilisé par le pod external-secrets pour récupérer le secret dans AWS Secrets Manager.

The screenshot shows the AWS IAM Roles page. A new role named 'AWS-EKS-ESO' has been created. The 'Summary' tab is selected, displaying details like ARN, Creation date (April 13, 2025), Last activity (none), and Maximum session duration (1 hour). The 'Permissions' tab is active, showing one attached policy: 'ESO-IAM-Policy'. This policy is a customer-managed policy. The 'Trust relationships' and 'Tags' tabs are also visible.

**Permissions policies (1)**  
You can attach up to 10 managed policies.

| Policy name    | Type             | Attached entities |
|----------------|------------------|-------------------|
| ESO-IAM-Policy | Customer managed | 1                 |

**Permissions boundary (not set)**

## Rôle IAM pour le pod external-secrets

La politique d'autorisation des rôles IAM est la suivante :

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Sid": "ESO",
 "Effect": "Allow",
 "Action": [
 "secretsmanager:GetSecretValue"
],
 "Resource": "*"
 }
]
}
```

- Ensuite, créez un secret dans AWS Secrets Manager.

The screenshot shows the AWS Secrets Manager interface for a secret named 'demo-datacamp-ESO'. The 'Overview' tab is selected. In the 'Secret details' section, it shows the encryption key as 'aws/secretsmanager', the secret name as 'demo-datacamp-ESO', and the secret ARN as 'arn:aws:secretsmanager:ap-southeast-1:717240872783:secret:demo-datacamp-ESO-DA4RvB'. The 'Secret value' section contains two key-value pairs: 'username' with value 'top\_secret\_username' and 'password' with value 'top\_secret\_password'. There are 'Key/value' and 'Plaintext' tabs, and buttons for 'Close' and 'Edit'.

### AWS Secrets Manager secret

Pour configurer notre cluster afin d'utiliser ESO :

helm repo add external-secrets <https://charts.external-secrets.io>

```
Replace the ARN of the IAM role
helm install external-secrets external-secrets/external-secrets \
-n external-secrets \
--create-namespace \
--version 0.15.1 \
--set serviceAccount.annotations."eks\.amazonaws\.com/role-
arn"=""
```

```

~/Documents/GitLab/demo-eks-helm [main !5 ?5] . .
> helm repo add external-secrets https://charts.external-secrets.io
"external-secrets" has been added to your repositories

~/Documents/GitLab/demo-eks-helm [main !5 ?5] . .
> helm search repo external-secrets
NAME CHART VERSION APP VERSION DESCRIPTION
external-secrets/external-secrets 0.15.1 v0.15.1 External secret management for Kubernetes

~/Documents/GitLab/demo-eks-helm [main !5 ?5] . .
> helm install external-secrets external-secrets/external-secrets \
 -n external-secrets \
 --create-namespace \
 --version 0.15.1 \
 --set serviceAccount.annotations."eks\\.amazonaws\\.com/role-arn"="arn:aws:iam::717240872783:role/AWS-EKS-ESO"
NAME: external-secrets
LAST DEPLOYED: Sun Apr 13 17:12:39 2025
NAMESPACE: external-secrets
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
external-secrets has been deployed successfully in namespace external-secrets!

In order to begin using ExternalSecrets, you will need to set up a SecretStore
or ClusterSecretStore resource (for example, by creating a 'vault' SecretStore).

More information on the different types of SecretStores and how to configure them
can be found in our Github: https://github.com/external-secrets/external-secrets

```

### Résultat de l'installation des secrets externes

- Vérifiez l'installation :

```

kubectl get pods,crd -n external-secrets
~/Documents/GitLab/demo-eks-helm [main !5 ?5] . .
> kubectl get pods,crd -n external-secrets
NAME READY STATUS RESTARTS AGE
pod/external-secrets-9c44fbf86-wdzdt 1/1 Running 0 116s
pod/external-secrets-cert-controller-7cdbbb6d5-vl7hw 1/1 Running 0 116s
pod/external-secrets-webhook-cbbb45647-qdtjq 1/1 Running 0 116s

NAME CREATED AT
customresourcedefinition.apextensions.k8s.io/acraccessstokens.generators.external-secrets.io 2025-04-13T09:12:41Z
customresourcedefinition.apextensions.k8s.io/clusterexternalsecrets.external-secrets.io 2025-04-13T09:12:41Z
customresourcedefinition.apextensions.k8s.io/clustergenerators.generators.external-secrets.io 2025-04-13T09:12:42Z
customresourcedefinition.apextensions.k8s.io/clusterpushsecrets.external-secrets.io 2025-04-13T09:12:41Z
customresourcedefinition.apextensions.k8s.io/clustersecretstores.external-secrets.io 2025-04-13T09:12:42Z
customresourcedefinition.apextensions.k8s.io/crinodes.vpcresources.k8s.aws 2025-04-13T07:58:14Z
customresourcedefinition.apextensions.k8s.io/ecriauthorizationtokens.generators.external-secrets.io 2025-04-13T09:12:42Z
customresourcedefinition.apextensions.k8s.io/etconfigs.crd.k8s.amazonaws.com 2025-04-13T08:00:20Z
customresourcedefinition.apextensions.k8s.io/externalsecrets.external-secrets.io 2025-04-13T09:12:42Z
customresourcedefinition.apextensions.k8s.io/fakes.generators.external-secrets.io 2025-04-13T09:12:41Z
customresourcedefinition.apextensions.k8s.io/gcraccessstokens.generators.external-secrets.io 2025-04-13T09:12:42Z
customresourcedefinition.apextensions.k8s.io/generatorstates.generators.external-secrets.io 2025-04-13T09:12:41Z
customresourcedefinition.apextensions.k8s.io/githubaccesstokens.generators.external-secrets.io 2025-04-13T09:12:42Z
customresourcedefinition.apextensions.k8s.io/grafanas.generators.external-secrets.io 2025-04-13T09:12:42Z
customresourcedefinition.apextensions.k8s.io/passwords.generators.external-secrets.io 2025-04-13T09:12:42Z
customresourcedefinition.apextensions.k8s.io/policyendpoints.networking.k8s.aws 2025-04-13T07:58:14Z
customresourcedefinition.apextensions.k8s.io/pushsecrets.external-secrets.io 2025-04-13T09:12:42Z
customresourcedefinition.apextensions.k8s.io/quayaccesstokens.generators.external-secrets.io 2025-04-13T09:12:41Z
customresourcedefinition.apextensions.k8s.io/secretstores.external-secrets.io 2025-04-13T09:12:41Z
customresourcedefinition.apextensions.k8s.io/secretgroupolicies.vpcresources.k8s.aws 2025-04-13T07:58:14Z
customresourcedefinition.apextensions.k8s.io/stssessiontokens.generators.external-secrets.io 2025-04-13T09:12:42Z
customresourcedefinition.apextensions.k8s.io/uuids.generators.external-secrets.io 2025-04-13T09:12:42Z
customresourcedefinition.apextensions.k8s.io/vaultdynamicsecrets.generators.external-secrets.io 2025-04-13T09:12:42Z
customresourcedefinition.apextensions.k8s.io/webhooks.generators.external-secrets.io 2025-04-13T09:12:41Z

```

### Vérifier les ressources provenant de l'install de l'OEN

- Ensuite, pour déployer la ressource ClusterSecretStore, enregistrez ce qui suit dans cluster-secret-store.yaml et créez la ressource.

```
apiVersion: external-secrets.io/v1beta1
```

```

kind: ClusterSecretStore
metadata:
 name: "cluster-secret-store"
spec:
 provider:
 aws:
 service: SecretsManager
 region: ap-southeast-1
 auth:
 jwt:
 serviceAccountRef:
 name: "external-secrets"
 namespace: "external-secrets"
kubectl create -f cluster-secret-store.yaml
kubectl get ClusterSecretStore
La ressource personnalisée ClusterSecretStore est un SecretStore à l'échelle du cluster qui peut être référencé par ExternalSecrets à partir de n'importe quel espace de noms. Il fait référence à l'adresse external-secrets serviceAccount pour s'authentifier auprès d'AWS Secrets Manager.

```

```

~/Documents/GitLab/demo-eks-helm main !5 ?4
[> k create -f cluster-secret-store.yaml
clustersecretstore.external-secrets.io/cluster-secret-store created

~/Documents/GitLab/demo-eks-helm main !5 ?4
[> k get ClusterSecretStore
NAME AGE STATUS CAPABILITIES READY
cluster-secret-store 20s Valid ReadWrite True

```

*Valider la création de la ressource ClusterSecretStore*

- Enfin, pour extraire le secret demo-datacamp-ESO d'AWS Secrets Manager, déployez la ressource personnalisée ExternalSecret. Nous allons extraire la valeur de la clé password. Enregistrez ce qui suit sur external-secret.yaml et créez la ressource.

```

apiVersion: external-secrets.io/v1beta1
kind: ExternalSecret
metadata:
 name: demo-datacamp-secret
 namespace: default # Change to your target namespace
spec:
 refreshInterval: 1h # How often to refresh the secret
 secretStoreRef:
 name: cluster-secret-store # This references your
ClusterSecretStore
 kind: ClusterSecretStore
 target:
 name: demo-datacamp-k8s-secret # The name of the resulting
Kubernetes Secret
 creationPolicy: Owner # ESO manages lifecycle of the K8s secret
 data:
 - secretKey: password # Key name in the Kubernetes Secret
 remoteRef:
 key: demo-datacamp-ESO # AWS Secrets Manager secret name

```

```

 property: password # Key in the AWS secret JSON object
kubectl create -f external-secret.yaml
kubectl get externalsecrets
kubectl get secrets
~/Documents/GitLab/demo-eks-helm [main !5 ?5] . . .
[> k create -f external-secret.yaml
externalsecret.external-secrets.io/demo-datacamp-secret created

~/Documents/GitLab/demo-eks-helm [main !5 ?5] . . .
[> k get externalsecrets
NAME STORETYPE STORE REFRESH INTERVAL STATUS READY
demo-datacamp-secret ClusterSecretStore cluster-secret-store 1h SecretSynced True

~/Documents/GitLab/demo-eks-helm [main !5 ?5] . . .
[> k get secrets
NAME TYPE DATA AGE
demo-datacamp-k8s-secret Opaque 1 16s

~/Documents/GitLab/demo-eks-helm [main !5 ?5] . . .
[> k get secrets demo-datacamp-k8s-secret -oyaml
apiVersion: v1
data:
 password: dG9wX3NlY3JldF9wYXNzd29yZA==
kind: Secret
metadata:
 annotations:
 reconcile.external-secrets.io/data-hash: 09bb452ca3203207fbebe4d0e2c99c37
 creationTimestamp: "2025-04-13T08:59:11Z"
 labels:
 reconcile.external-secrets.io/created-by: fff58accc3ee4e99bdb209478eb97518
 reconcile.external-secrets.io/managed: "true"
 name: demo-datacamp-k8s-secret
 namespace: default
 ownerReferences:
 - apiVersion: external-secrets.io/v1beta1
 blockOwnerDeletion: true
 controller: true
 kind: ExternalSecret
 name: demo-datacamp-secret
 uid: 675b7db5-9310-430d-bf3f-154b389caed6
 resourceVersion: "11914"
 uid: fa32c34a-e4c2-487f-aa36-7bae2ed23806
type: Opaque

~/Documents/GitLab/demo-eks-helm [main !5 ?5] . . .
[> echo "dG9wX3NlY3JldF9wYXNzd29yZA==" | base64 -d
top_secret_password

```

Vérifier que le secret a été récupéré avec succès

L'image ci-dessus montre que la valeur top\_secret\_password de la clé password a été récupérée avec succès dans AWS Secrets Manager et qu'un secret Kubernetes contenant cette valeur a été créé.

### **Meilleures pratiques pour le développement des cartes Helm**

Au fur et à mesure que nous utilisons Helm au-delà de l'installation de cartes de base, il est important de suivre les meilleures pratiques qui favorisent la maintenabilité, la réutilisation et la fiabilité.

Cette section couvre les pratiques de développement clés, telles que l'utilisation des plugins Helm pour étendre les fonctionnalités, la structuration des Charts pour la réutilisation, la validation de nos Charts à l'aide d'outils d'analyse et la gestion efficace des versions et des dépendances.

## Plugins Helm

Les plugins Helm sont un moyen puissant d'étendre les fonctionnalités de base de Helm et de les adapter à votre flux de travail. Ils nous permettent d'ajouter des commandes personnalisées et d'intégrer des outils externes sans modifier Helm lui-même.

Les plugins peuvent être installés à partir d'un dépôt Git, d'une archive ou d'un répertoire local. Une fois installés, ils se comportent comme les commandes natives de Helm. Les plugins Helm sont installés localement sur notre machine, et non sur le cluster Kubernetes. Cela signifie qu'ils améliorent les capacités de notre CLI Helm local et ne nécessitent aucun changement ou déploiement sur le cluster Kubernetes. Chaque utilisateur ou système utilisant Helm doit installer les plugins souhaités indépendamment.

Un plugin commun qui est largement utilisé, en particulier sur les pipelines CI/CD, est helm-diff. Il compare une version Helm à une Helm Chart et montre les changements qui seraient appliqués. Cette fonction est particulièrement utile pour prévisualiser les mises à niveau.

```
Install the plugin
helm plugin install https://github.com/databus23/helm-diff

List installed plugins
helm plugin list

View the changes for an upgrade
helm diff upgrade datacamp-demo-helm ./demo-helm --set
image.tag="1.26"
~/Documents/GitLab/demo-eks-helm/helm-charts [main !5 ?5]
> helm plugin install https://github.com/databus23/helm-diff
Downloading https://github.com/databus23/helm-diff/releases/latest/download/helm-diff-macos-arm64.tgz
Preparing to install into /Users/kennyang/Library/helm/plugins/helm-diff
Installed plugin: diff

~/Documents/GitLab/demo-eks-helm/helm-charts [main !5 ?5]
> helm plugin list
NAME VERSION DESCRIPTION
diff 3.11.0 Preview helm upgrade changes as a diff
drift 0.1.1 Identifies configurations deviated from helm charts, mostly due to inplace edits of kubernetes workload
secrets 4.6.3 This plugin provides secrets values encryption for Helm charts secure storing
```

*Sortie des commandes du plugin helm*

```

~/Documents/GitLab/demo-eks-helm/helm-charts [main !5 ?5]
[> helm diff upgrade datacamp-demo-helm ./demo-helm --set image.tag="1.26"
default, datacamp-demo-helm, Deployment (apps) has changed:
Source: demo-helm/templates/deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
 name: datacamp-demo-helm
 labels:
 helm.sh/chart: demo-helm-0.1.0
 app.kubernetes.io/name: demo-helm
 app.kubernetes.io/instance: datacamp-demo-helm
 app.kubernetes.io/version: "1.16.0"
 app.kubernetes.io/managed-by: Helm
spec:
 replicas: 1
 selector:
 matchLabels:
 app.kubernetes.io/name: demo-helm
 app.kubernetes.io/instance: datacamp-demo-helm
 template:
 metadata:
 labels:
 helm.sh/chart: demo-helm-0.1.0
 app.kubernetes.io/name: demo-helm
 app.kubernetes.io/instance: datacamp-demo-helm
 app.kubernetes.io/version: "1.16.0"
 app.kubernetes.io/managed-by: Helm
 spec:
 serviceAccountName: datacamp-demo-helm
 containers:
 - name: demo-helm
 image: "nginx:1.16.0"
 + image: "nginx:1.26"
 imagePullPolicy: IfNotPresent
 ports:
 - name: http
 containerPort: 80
 protocol: TCP
 - name: https
 containerPort: 443

```

*Sortie de la commande helm diff*

## Structurer des Charts réutilisables

Au fur et à mesure que notre utilisation de Helm augmente, il devient crucial de structurer nos Charts pour qu'ils puissent être réutilisés. Un diagramme Helm bien structuré favorise la cohérence, simplifie la maintenance et permet des modèles de déploiement modulaires entre les projets. Voici deux conseils :

### Utiliser des modèles et des aides

Comme le montre la sous-section "Personnalisation des modèles" de la section "Création de votre premier Chart à barre", nous avons exploité le fichier \_helpers.tpl pour ajouter un sélecteur supplémentaire à la ressource deployment et des étiquettes à toutes les ressources.

L'utilisation de modèles d'aide permet d'éviter les doublons et fournit ainsi une source unique de vérité pour les extraits couramment réutilisés. Cela améliore la cohérence de votre tableau et facilite la maintenance - toutes les modifications futures ne doivent être effectuées qu'à un seul endroit.

### Diviser les grands Charts en sous-groupes

Si notre application est constituée de plusieurs composants faiblement couplés (par exemple, une application web, une base de données et Redis), envisagez de diviser le diagramme en sous-groupes. Placez-les dans le dossier charts/ et gérez-les comme des dépendances via Chart.yaml.

dependencies:

- name: redis
- version: 17.1.2
- repository: <https://charts.bitnami.com/bitnami>

Lancez l'application helm dependency update pour extraire ces sous-Charts. Cela permet de gérer chaque composant de manière indépendante et favorise la réutilisation dans différents projets.

### Mise en place d'une chartreuse et validation des Charts

Avant de déployer un Helm Chart sur un cluster Kubernetes, il est important de s'assurer que le Chart est bien structuré, syntaxiquement correct et logique. Helm fournit des outils intégrés pour nous aider à détecter rapidement les problèmes par le biais de linting et de tests.

La commande helm lint analyse notre tableau pour y déceler les erreurs courantes et les violations des meilleures pratiques. Il vérifie la syntaxe des modèles, la structure des fichiers Charts et la présence de champs obligatoires dans Chart.yaml.

```
helm lint ./demo-helm
```

```
~/Documents/GitLab/demo-eks-helm/helm-charts main !5 ?5 ...
[> helm lint ./demo-helm
==> Linting ./demo-helm
[INFO] Chart.yaml: icon is recommended

1 chart(s) linted, 0 chart(s) failed
```

*Sortie de la commande helm lint*

Si nous modifions la valeur de name dans Chart.yaml de "demo-helm" à "demo\_helm", il y aura un problème où helm lint sera bloqué.

```
! Chart.yaml U X
helm-charts > demo-helm > ! Chart.yaml > ...
1 apiVersion: v2
2 name: demo_helm
3 description: A Helm chart for Kubernetes
4
```

Modification du "-" en "\_"

```
~/Documents/GitLab/demo-eks-helm/helm-charts main !5 ?5 ...
[> helm lint ./demo-helm
==> Linting ./demo-helm
[INFO] Chart.yaml: icon is recommended
[WARNING] templates/deployment.yaml: object name does not conform to Kubernetes naming requirements: "test-release-demo_helm": metadata.name: Invalid value: "test-release-demo_helm": a lower
st of lower case alphanumeric characters, '-' or '.', and must start and end with an alphanumeric character (e.g. 'example.com', regex used for validation is '[a-z0-9]([a-z0-9]*[a-z0-9])?(\\.
[WARNING] templates/service.yaml: object name does not conform to Kubernetes naming requirements: "test-release-demo_helm": metadata.name: Invalid value: "test-release-demo_helm": a DNS-1035
alphanumeric characters or '-', start with an alphabetic character, and end with an alphanumeric character (e.g. 'my-name', or 'abc-123', regex used for validation is '[a-z]([a-z0-9]*[a-z0-
[WARNING] templates/serviceaccount.yaml: object name does not conform to Kubernetes naming requirements: "test-release-demo_helm": metadata.name: Invalid value: "test-release-demo_helm": a lo
onist of lower case alphanumeric characters, '-' or '.', and must start and end with an alphanumeric character (e.g. 'example.com', regex used for validation is '[a-z0-9]([a-z0-9]*[a-z0-9])?
')
[WARNING] templates/tests/test-connection.yaml: object name does not conform to Kubernetes naming requirements: "test-release-demo_helm-test-connection": metadata.name: Invalid value: "test-r
: a lowercase RFC 1123 subdomain must consist of lower case alphanumeric characters, '-' or '.', and must start and end with an alphanumeric character (e.g. 'example.com', regex used for vali
z0-9])?([a-z0-9]([a-z0-9]*[a-z0-9])?)*)"
1 chart(s) linted, 0 chart(s) failed
```

helm lint : envoi de messages d'avertissement

En lignant et en testant nos diagrammes avant de les déployer, nous réduisons le risque d'échec des versions et de mauvaises configurations en production. Ces pratiques sont essentielles pour maintenir des diagrammes Helm de haute qualité qui se comportent de manière prévisible dans tous les environnements.

## Gestion des versions et des dépendances

Au fur et à mesure que nos cartes Helm évoluent et dépendent d'autres cartes, il devient essentiel de gérer les versions de manière cohérente et de suivre les dépendances avec précision. Helm fournit des mécanismes intégrés pour prendre en charge cela grâce au versionnage sémantique, au fichier Chart.lock et aux dépendances Chart.yaml.

Helm utilise le Semantic Versioning pour les versions des cartes. La version d'une carte est définie dans Chart.yaml sous la clé version:

```
! Chart.yaml ×
helm-charts > demo-helm > ! Chart.yaml > ...
1 apiVersion: v2
2 name: demo_helm
3 description: A Helm chart for Kubernetes
4
5 # A chart can be either an 'application' or a 'library' chart.
6 #
7 # Application charts are a collection of templates that can be packaged into versioned archives
8 # to be deployed.
9 #
10 # Library charts provide useful utilities or functions for the chart developer. They're included as
11 # a dependency of application charts to inject those utilities and functions into the rendering
12 # pipeline. Library charts do not define any templates and therefore cannot be deployed.
13 type: application
14
15 # This is the chart version. This version number should be incremented each time you make changes
16 # to the chart and its templates, including the app version.
17 # Versions are expected to follow Semantic Versioning (https://semver.org/)
18 version: 0.1.0
19
20 # This is the version number of the application being deployed. This version number should be
21 # incremented each time you make changes to the application. Versions are not expected to
22 # follow Semantic Versioning. They should reflect the version the application is using.
23 # It is recommended to use it with quotes.
24 appVersion: "1.16.0"
```

### Version Chart

Le versionnage sémantique suit le format : MAJOR.MINOR.PATCH

- Changements majeurs
- MINOR - ajouts de fonctionnalités rétrocompatibles
- PATCH - corrections de bogues rétrocompatibles

La version du Chart doit être incrémentée chaque fois que des modifications sont apportées au Chart lui-même, y compris à ses modèles, à ses valeurs ou même à la version de l'application. Parallèlement, la version de l'application reflète la version de l'application réelle déployée par le Chart, généralement alignée sur la balise d'image utilisée dans le Chart. Il doit être mis à jour chaque fois que l'image de l'application est modifiée.

Une bonne gestion des versions permet aux équipes de savoir si une mise à niveau est sûre ou si elle nécessite des précautions supplémentaires. Il permet également à Helm de gérer les chemins de mise à niveau et les dépendances de manière plus fiable.

Dans Helm 3, que nous utilisons depuis le début de cet article, les dépendances sont définies directement dans le fichier Chart.yaml, dans la section dependencies. Il n'est pas inclus par défaut lorsque nous créons le Chart. Il remplace le fichier requirements.yaml de Helm 2.

```
! Chart.yaml u ×
helm-charts > demo-helm > ! Chart.yaml > [] dependencies > {} 0
24 appVersion: 1.10.0
25
26 dependencies:
27 - name: elasticsearch
28 version: 21.5.0
29 repository: https://charts.bitnami.com/bitnami
30
```

#### Dépendances définies dans Chart.yaml

Le fichier Chart.lock est automatiquement généré lorsque vous exécutez helm dependency update. Il agit comme un fichier de verrouillage dans de nombreux gestionnaires de paquets, en enregistrant les versions exactes de chaque dépendance telle que définie dans la section dependencies de Chart.yaml. Cela garantit des constructions reproductibles et un comportement cohérent entre les environnements et les équipes.

```
! Chart.lock u ×
helm-charts > demo-helm > ! Chart.lock > [] dependencies
1 dependencies:
2 - name: elasticsearch
3 repository: https://charts.bitnami.com/bitnami
4 version: 21.5.0
5 digest: sha256:0354de9f957f4a6bce2916c23bd49ab13e1b34a5d5b900dfaf895df32fc3065b
6 generated: "2025-04-14T23:02:28.405891+08:00"
7
```

#### Contenu de Chart.lock

Si vous connaissez Terraform, le concept est similaire au fichier .terraform.lock.hcl, qui verrouille les versions des fournisseurs. Bien que les formats et les écosystèmes sous-jacents diffèrent, tous deux verrouillent les versions des dépendances afin de garantir la stabilité et la reproductibilité.

## Conclusion

Les cartes Helm offrent un moyen puissant et standardisé de packager, configurer et déployer des applications sur Kubernetes. Tout au long de cet article, nous avons exploré ce qu'est un diagramme Helm, comment il est structuré et comment il simplifie le processus de déploiement dans différents environnements.

Nous avons passé en revue les principales étapes :

- **Création de votre premier Helm Chart** en utilisant helm create.
- **Personnalisation des configurations** à l'aide de values.yaml et de paramètres spécifiques à l'environnement.

- **Déployer et gérer les versions** à l'aide de commandes telles que helm install, helm upgrade, et helm rollback.
- **Suivre les meilleures pratiques** telles que les diagrammes de linting, la structuration de composants réutilisables, le traitement sécurisé des secrets et la gestion efficace des versions et des dépendances.

Au fur et à mesure que vous prenez de l'assurance en travaillant avec Helm, prenez le temps d'explorer le vaste écosystème de [cartes Helm publiques](#), hébergées par des communautés et des fournisseurs. Ces tableaux peuvent accélérer vos déploiements et servir de références précieuses pour votre propre développement.