TPs Helm : Découverte des objets `.Release`, `.Chart` et `.Values`

Objectif

Ces TPs vous permettront de comprendre et d'utiliser les objets Helm fondamentaux `.Release`, `.Chart` et `.Values` pour personnaliser vos chartes Helm.

TP 1 : Utilisation de l'objet `.Release`

Prérequis

- Un cluster Kubernetes fonctionnel.
- Helm installé sur votre machine.

Objectif

Comprendre l'utilisation de l'objet `.Release` dans les templates Helm.

Énoncé

1. Créez une charte Helm avec `helm create tp-release`.
2. Modifiez le fichier `templates/configmap.yaml` pour inclure l'objet `.Release` comme suit :

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ .Release.Name }}-configmap
  labels:
    app: {{ .Release.Name }}
data:
  release-namespace: {{ .Release.Namespace }}
  release-service: {{ .Release.Service }}
```

1. Installez la charte avec une commande comme :

```
$ helm install my-release ./tp-release
```

1. Récupérez le ConfigMap créé avec `kubectl` et vérifiez les valeurs injectées.

Correction

Voici le ConfigMap attendu après installation :

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: my-release-configmap
  labels:
    app: my-release
data:
  release-namespace: default
  release-service: Helm
```

TP 2 : Utilisation de l'objet `.Chart`

Objectif

Comprendre et exploiter l'objet `.Chart` pour inclure des métadonnées de votre charte.

Énoncé

1. Modifiez le fichier `templates/configmap.yaml` pour inclure des informations de l'objet `.Chart` comme suit :

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ .Release.Name }}-configmap
  labels:
    app: {{ .Release.Name }}
data:
  chart-name: {{ .Chart.Name }}
  chart-version: {{ .Chart.Version }}
  chart-app-version: {{ .Chart.AppVersion }}
```

1. Installez la charte et vérifiez le ConfigMap créé.

Correction

Après l'installation, le ConfigMap contiendra :

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: my-release-configmap
  labels:
    app: my-release
```

```
data:
  chart-name: tp-release
  chart-version: 0.1.0
  chart-app-version: 1.0
```

TP 3 : Utilisation de l'objet `.Values`

Objectif

Découvrir comment personnaliser le comportement des chartes avec des valeurs dynamiques.

Énoncé

1. Ajoutez une section personnalisée dans le fichier `values.yaml` :

```
customConfig:
   key1: value1
   key2: value2
```

1. Modifiez le fichier `templates/configmap.yaml` pour inclure ces valeurs :

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ .Release.Name }}-configmap
  labels:
    app: {{ .Release.Name }}
data:
  key1: {{ .Values.customConfig.key1 }}
  key2: {{ .Values.customConfig.key2 }}
```

1. Installez la charte et vérifiez les valeurs dans le ConfigMap créé.

   Déployez la charte avec des valeurs personnalisées via un fichier :

```
customConfig:
  key1: newValue1
  key2: newValue2
```

Utilisez la commande suivante :

```
helm install my-release ./tp-release -f custom-values.yaml
```

Correction

Avec les valeurs par défaut, le ConfigMap contiendra :

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: my-release-configmap
  labels:
    app: my-release
data:
  key1: value1
  key2: value2
```

Avec le fichier de valeurs personnalisées, il contiendra :

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: my-release-configmap
  labels:
    app: my-release
data:
  key1: newValue1
  key2: newValue2
```

Conclusion

Ces TPs introduisent les objets fondamentaux de Helm et leur usage pratique. Une fois compris, vous pourrez les combiner pour créer des chartes puissantes et dynamiques.