

MODULE 6

# Création d'applications à partir de modèles

Templates OpenShift et déploiements multi-conteneurs

---



Templates



Multi-conteneurs



Paramètres



Réutilisation

 Durée estimée : 3 heures

# Introduction : Pourquoi les templates ?

## Qu'est-ce qu'un template OpenShift ?

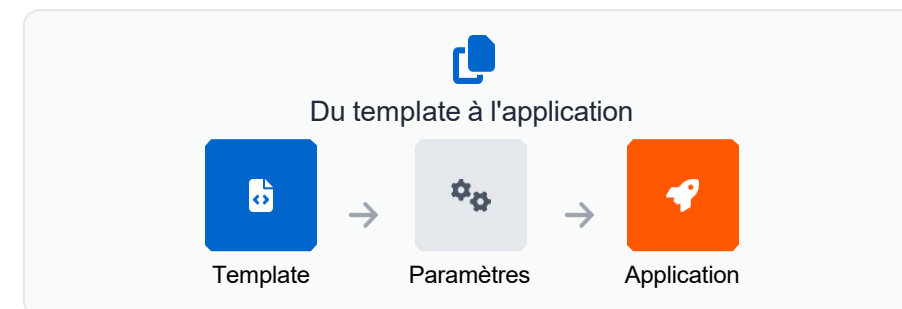
Un template est un modèle paramétrable qui définit un ensemble de ressources OpenShift pouvant être instanciées en une seule opération, servant de blueprint pour des déploiements répétables.

## Contexte d'utilisation

- ✓ Déploiement récurrent d'applications similaires
- ✓ Environnements de développement, test et production
- ✓ Standardisation des déploiements d'entreprise
- ✓ Applications multi-tiers ou multi-composants

## Avantages clés

-  **Réutilisabilité**  
Créez une fois, déployez plusieurs fois
-  **Paramétrage flexible**  
Adaptation aux contextes
-  **Déploiement accéléré**  
Gain de temps considérable
-  **Standardisation**  
Cohérence des déploiements



# Structure générale d'un template OpenShift

## Anatomie d'un template OpenShift

Un template OpenShift est défini dans un fichier YAML (ou JSON) avec une structure standardisée pour créer des templates efficaces et réutilisables.

## Éléments clés d'un template



apiVersion & kind

template.openshift.io/v1, Template



parameters

Paramètres personnalisables



metadata

Nom, description, annotations



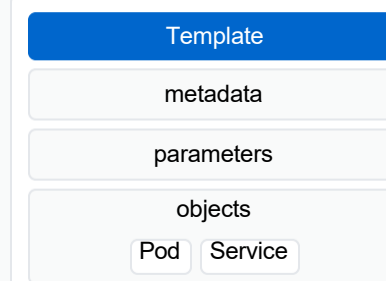
objects

Ressources OpenShift à créer

## Exemple de fichier YAML

```
apiVersion: template.openshift.io/v1
kind: Template
metadata:
  name: webapp-mysql-example
  annotations:
    description: Application web avec MySQL
parameters:
- name: APP_NAME
  value: webapp
```

### Schéma simplifié



### Conseils d'utilisation

- Structurez vos templates par fonctionnalité
- Paramétrez tous les éléments variables
- Utilisez des noms descriptifs
- Ajoutez des étiquettes et annotations

💡 À retenir : Un template permet de définir, paramétrer et déployer plusieurs ressources OpenShift en une seule opération.

# Les paramètres dans un template

## Types de paramètres



### Paramètres obligatoires

Paramètres sans valeur par défaut, requis lors de l'instanciation.



### Paramètres avec valeurs par défaut

Paramètres optionnels avec valeur prédéfinie si non fournie.



### Paramètres sécurisés

Masqués dans l'interface (mots de passe, clés API, tokens).

## Exemple pratique

parameters:

- name: APP\_NAME  
description: Nom de l'application required: true
- name: MYSQL\_VERSION  
description: Version MySQL value:  
"8.0"
- name: DB\_PASSWORD  
description: Mot de passe BDD generate:  
expression  
from: "[a-zA-Z0-9]{16}"  
required: true

## Bonnes pratiques

Descriptions claires et concises



Valeurs par défaut pertinentes



Génération de valeurs sécurisées pour les données sensibles



### Astuce



Utilisez `generate: expression` pour créer des valeurs sécurisées automatiquement.

## Objets générés par un template

### Types d'objets disponibles

Un template OpenShift peut générer une variété d'objets Kubernetes/OpenShift dans une seule opération.

**Pod**

Unité d'exécution

**Service**

Accès réseau

**Route**

Exposition externe

**ConfigMap**

Config non-sensible

**Secret**

Données sensibles

**Et plus...**

DC, PVC, Network...

#### Avantages multi-ressources

- ✓ Déploiement cohérent
- ✓ Relations automatiques
- ✓ Versionnement complet
- ✓ Gestion simplifiée

### Structure YAML d'objets multiples

```
objects:
- apiVersion: v1 kind:
  Secret
  metadata: {name: ${APP_NAME}-secret}

- apiVersion: v1 kind:
  Service
  metadata: {name: ${APP_NAME}}

- apiVersion: v1 kind:
  Route
  metadata: {name: ${APP_NAME}}

- apiVersion: apps.openshift.io/v1 kind:
  DeploymentConfig
  # ...
```



Un template peut générer tous les objets nécessaires pour une application complète en une seule opération

**Cas d'usage typiques**

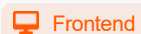
Application 3-tiers, microservices, CI/CD

## Cas d'usage des templates OpenShift



### Applications multi-tiers

Templates pour architectures front-end, back-end, et base de données en une seule opération.



Frontend



API



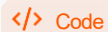
DB

→ Déploiement 80% plus rapide



### Environnements de test

Déploiement rapide d'environnements identiques pour tous les développeurs.



Code



DB



QA



### Workflows CI/CD

Infrastructure CI/CD standardisée au sein de l'organisation.




Jenkins



SonarQube



Nexus

Impact réel en entreprise 

**70%**

Réduction du temps

**90%**

Cohérence

**40%**

Moins d'erreurs

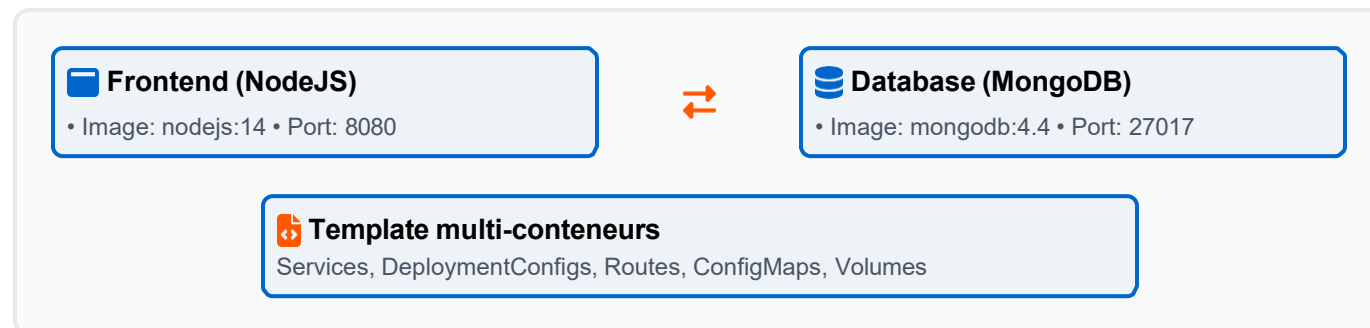
**Conseil** : Créez un catalogue de templates organisationnel pour standardiser les déploiements.

# Templates multi-conteneurs & bonnes pratiques

## Pourquoi utiliser des templates multi-conteneurs ?

Les templates multi-conteneurs permettent de déployer des applications complexes comportant plusieurs services interdépendants en une seule opération, garantissant cohérence et simplification du cycle de vie.

## Exemple : Application Web + Base de données



## Composants du template

- ✓ Déploiements (frontend, DB)
- ✓ Route pour accès externe
- ✓ Services pour exposition
- ✓ Volumes persistants

### Points clés pour le déploiement

Un template multi-conteneurs bien conçu permet un déploiement cohérent, une gestion centralisée et une plus grande fiabilité des déploiements complexes en environnement de production.

## Bonnes pratiques

### Structure modulaire

Organisation logique pour faciliter maintenance et évolution.

### Paramétrage riche

Variables configurables : versions, réplicas, ressources.

### Sécurité intégrée

Secrets pour données sensibles et permissions limitées.

### Documentation

Annotations claires pour paramètres et utilisation.

### Conseils pour l'évolutivité

- ✓ Scaling automatique (HPA)
- ✓ Déploiement par étapes
- ✓ MAJ indépendantes
- ✓ Priorisation des ressources

### Pour aller plus loin

- > Intégrer des métriques (Prometheus)
- > Déploiement CI/CD avec templates et Helm

## Travaux pratiques : Votre premier template personnalisé


### Objectif de l'atelier

Créer un template OpenShift personnalisé pour déployer une application web avec sa base de données.

### Outils recommandés


 CLI OpenShift


 Éditeur YAML

 Console web

### Conseils pratiques

- ✓ Travail en binôme
- ✓ Paramètres documentés
- ✓ Templates simples d'abord
- ✓ Noms génériques

 Temps estimé  
45-60 minutes

 N'hésitez pas à consulter la documentation OpenShift pour plus d'exemples.

1

#### Analyse d'un template

Examinez le template fourni :  
`oc get template django-psql-example -n openshift -o yaml`

2

#### Création du template

Modifiez pour votre cas d'usage :

- Paramètres personnalisés
- Objets (ConfigMap, Route...)

3

#### Déploiement

Importez votre template :  
`oc create -f mon-template.yaml`

Déployez une instance :  
`oc new-app --template=nom-template -p PARAM1=valeur`

4

#### Vérification

Validez le fonctionnement :

- État des pods ( `oc get pods` )
- Accès via la route créée
- Consultation des logs



# Synthèse & Ouverture

## Points clés à retenir

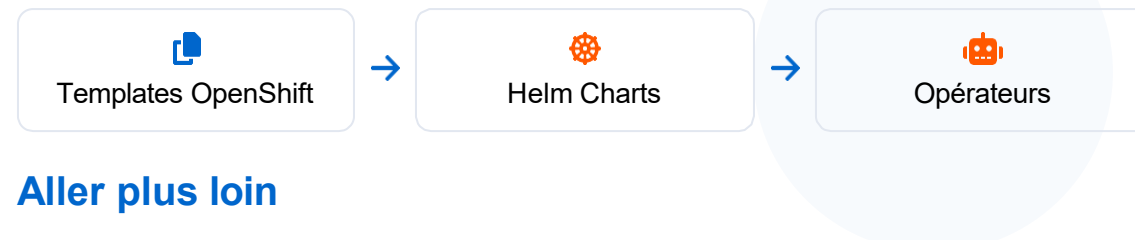
- ✓ Templates : modèles paramétrables pour ressources OpenShift
- ✓ Structure YAML : apiVersion, kind, metadata, parameters, objects
- ✓ Paramétrage flexible et multi-conteneurs pour applications complexes

## FAQ

Q: Quelle différence entre un template et un déploiement simple ?

R: Un template regroupe plusieurs ressources paramétrables en un seul objet.

## Évolution des méthodes



## Aller plus loin



### Helm Charts

- > Versionnement et dépendances
- > Repository centralisé de charts



### Opérateurs

- > Automatisation des opérations
- > Gestion du cycle de vie complet

*Les templates sont une excellente introduction aux méthodes DevOps avancées*



# Formation RedHat OpenShift

Applications de conteneurisation

---

 3 jours (21 heures)

 Formation Openshift

Formateur

Ahmed ZIAD

Dates

21-23 Juillet

Lieu

France