Automatiser la mise en production avec Kubernetes

Table des matières

Module 1 : Introduction aux Kubernetes

🎓 Objectifs pédagogiques :

- Comprendre ce qu'est Kubernetes, son rôle et son intérêt dans la mise en production moderne.
- Identifier les composants de base d'un cluster Kubernetes.
- Appréhender le fonctionnement général d'un déploiement de conteneurs avec Kubernetes.

🕮 1. Qu'est-ce que Kubernetes ?

Kubernetes (aussi appelé K8s) est une plateforme open source conçue pour **automatiser le déploiement, la mise à l'échelle et la gestion des applications conteneurisées**. Il est devenu la norme pour orchestrer les conteneurs Docker (et autres).

🔧 Créé à l'origine par Google, puis confié à la **Cloud Native Computing Foundation (CNCF)**.

🚀 Pourquoi Kubernetes ?

- Gère automatiquement les ressources (CPU, RAM) pour les conteneurs.
- Redémarre les services qui tombent en panne.
- Permet des mises à jour sans coupure (rolling update).
- Facilite le scaling (horizontal ou vertical).
- Fournit des abstractions pour simplifier la gestion (Pods, Services…).

🏭 2. Architecture de Kubernetes

Un cluster Kubernetes est constitué de deux types de machines :

🧠 Le Control Plane (plan de contrôle)

- **API Server** : point d'entrée principal ; communique avec `kubectl`.
- **Scheduler** : décide sur quel nœud exécuter un nouveau Pod.
- **Controller Manager** : s'assure que l'état actuel du cluster correspond à l'état désiré.
- **etcd** : base de données clé/valeur qui stocke l'état du cluster.

⚙ Les nœuds (Nodes)

Chaque node (machine physique ou virtuelle) exécute :

- **kubelet** : agent qui communique avec le control plane.
- **kube-proxy** : gère la mise en réseau des Pods.
- **Runtime de conteneurs** (ex: Docker, containerd, CRI-O)

---

3. Composants fondamentaux de Kubernetes

Composant	Description
Pod	Plus petite unité déployable. Contient un ou plusieurs conteneurs.
Deployment	Gère les Pods pour assurer un état stable.
Service	Permet de rendre des Pods accessibles via une IP stable.
Namespace	Isole des ressources dans le cluster.
ConfigMap & Secret	Gèrent la configuration de manière externe aux images.
Volume	Permet de persister les données au-delà du cycle de vie des Pods.

---

4. Cycle de vie d'un déploiement Kubernetes

1. Écriture d'un manifeste (YAML)
2. Application via `kubectl apply`
3. Kubernetes planifie le Pod sur un Node disponible
4. L'application est exécutée
5. Le Service rend le Pod accessible (interne ou externe)

---

5. Exemple de manifeste de Pod

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: mon-pod
spec:
  containers:
    - name: nginx-container
      image: nginx:latest
      ports:
        - containerPort: 80
```

☞ Ce fichier YAML crée un Pod contenant un conteneur **nginx** exposé sur le port 80.

---

6. Mini-lab pratique

Étapes à réaliser avec Minikube ou Kind (local) :

1. Installer Kubernetes via Minikube :

```
minikube start
```

2. Créer un fichier `pod.yaml` avec l'exemple ci-dessus.
3. Appliquer le manifeste :

```
kubectl apply -f pod.yaml
```

4. Vérifier que le Pod est en cours d'exécution :

```
kubectl get pods
```

5. Afficher les logs :

```
kubectl logs mon-pod
```

✓ 7. Quiz d'évaluation (à choix multiples)

1. Quel composant de Kubernetes orchestre les conteneurs ?
a) Docker
b) etcd
c) Scheduler ✓
d) Pod

2. Lequel de ces objets permet de regrouper des ressources ?
a) Deployment
b) Pod
c) Namespace ✓
d) Volume

3. Quel fichier est utilisé pour définir un Pod ?
a) .json
b) .yaml ✓
c) .ini
d) .xml

📌 À retenir :

- Kubernetes automatise la gestion des conteneurs.
- Le Pod est l'unité de base.
- Le Control Plane orchestre l'ensemble du cluster.
- L'interaction se fait principalement via des fichiers YAML.

Module 2 : Utilisation d'une plateforme Kubernetes

🎓 Objectifs pédagogiques :

- Installer et configurer un cluster Kubernetes local ou distant.
- Utiliser l'outil `kubectl` pour interagir avec le cluster.
- Comprendre les notions de **contexte**, **namespace**, et **accès sécurisé** à un cluster.

📄 1. Choisir sa plateforme Kubernetes

🔄 En local (idéale pour l'apprentissage) :

Outil	Avantages
Minikube	Facile à installer, compatible Windows/Linux/Mac
Kind	Léger, tourne dans Docker
K3s	Version allégée de Kubernetes

☁️ Dans le cloud (pour l'entreprise / mise en prod) :

Provider	Service
Google	GKE (Google Kubernetes Engine)
Amazon	EKS (Elastic Kubernetes Service)
Microsoft	AKS (Azure Kubernetes Service)

🛠️ 2. Installation de Minikube (exemple)

✅ Pré-requis :

- **`Docker`** ou VirtualBox installé

`kubectl` installé :

```bash
brew install kubectl      # macOS
sudo apt install kubectl  # Linux
```

🚀 Lancer un cluster local :

```
minikube start
```

🔍 Vérifier l'état :

```
kubectl get nodes
```

⚙ 3. L'outil `kubectl`

📌 Commandes de base :

```
kubectl get pods                # Liste les Pods
kubectl get deployments         # Liste les Deployments
kubectl get services            # Liste les Services
kubectl describe pod <name>     # Infos détaillées sur un Pod
kubectl logs <name>             # Affiche les logs du Pod
```

🔧 Appliquer un fichier de configuration :

```
kubectl apply -f mon-pod.yaml
```

🔁 Supprimer une ressource :

```
kubectl delete -f mon-pod.yaml
```

🌐 4. Connexion à un cluster cloud

Si tu utilises un cluster géré (ex: GKE) :

```
gcloud container clusters get-credentials my-cluster --zone europe-west1
kubectl config use-context gke_my-project_europe-west1_my-cluster
```

🔐 La configuration est enregistrée dans le fichier `~/.kube/config`.

📂 5. Gérer les contexts et namespaces

🔄 Lister les contextes disponibles :

```
kubectl config get-contexts
```

✅ Passer à un autre cluster :

```
kubectl config use-context <nom-contexte>
```

🎁 Travailler dans un namespace :

```
kubectl create namespace staging
kubectl apply -f mon-app.yaml -n staging
```

❓ 6. Mini-lab : Premier cluster et première commande

Objectif : Créer un cluster Minikube et y déployer un pod.

1. Lancer Minikube :

```
minikube start
```

2. Créer un manifeste `mon-pod.yaml` :

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: test-pod
spec:
  containers:
    - name: nginx
      image: nginx
      ports:
        - containerPort: 80
```

3. Déployer et vérifier :

```
kubectl apply -f mon-pod.yaml
kubectl get pods
minikube service list
```

✓ Quiz rapide

1. Quelle commande permet de voir les Pods actifs ?
a) `kubectl view pods`
b) `kubectl get pods` ✓
c) `k8s pods list`
d) `kubectl show pods`

2. Quel outil permet de lancer Kubernetes en local ?
a) Terraform
b) VirtualBox
c) Minikube ✓
d) Git

3. Quelle commande change le cluster actif ?
a) `kubectl set-cluster`
b) `kubectl config use-context` ✓
c) `kubectl switch`
d) `kubectl config set-namespace`

📌 À retenir :

- `kubectl` est l'interface principale pour interagir avec Kubernetes.
- Minikube est idéal pour apprendre et expérimenter localement.
- La configuration multi-cluster se gère via les **contextes**.
- Les namespaces isolent les ressources au sein d'un même cluster.

Module 3 : Préparation d'un environnement de pré-production

🎓 Objectifs pédagogiques :

- Comprendre l'intérêt d'un environnement de pré-production.
- Savoir isoler et configurer cet environnement dans un cluster Kubernetes.
- Simuler des conditions proches de la production pour valider l'application avant sa mise en ligne.

---

1. Qu'est-ce qu'un environnement de pré-production ?

La **pré-production (staging)** est une copie contrôlée de l'environnement de production. C'est un **environnement de test avancé**, dans lequel les développeurs et QA peuvent :

- Tester les déploiements avant production.
- Vérifier les performances et la stabilité.
- Identifier les erreurs qui ne sont pas visibles en développement local.
- Réaliser des démonstrations internes ou externes.

🔒 2. Séparer la pré-production dans Kubernetes

Utiliser les namespaces pour isoler les environnements :

```
kubectl create namespace preprod
```

Chaque ressource déployée en pré-production utilisera ce namespace :

```
kubectl apply -f app-deployment.yaml -n preprod
```

⚙ 3. Configurations spécifiques à la pré-production

🔄 Variables d'environnement :

Créer des `ConfigMap` et `Secret` propres à la pré-prod.

Exemple de ConfigMap :

yaml

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: app-config
  namespace: preprod
data:
  API_URL: "https://api.preprod.example.com"
```

Exemple de Secret :

yaml

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: db-secret
  namespace: preprod
type: Opaque
data:
  password: cHJlcHJvZF9wYXNz # base64
```

---

4. Ressources typiques à déployer en préprod

Ressource	Description
App (Pod/Deployment)	Version candidat au passage en prod
DB Mock ou anonymisée	Données réalistes sans contenu sensible
Monitoring léger	Pour capter les erreurs (Logs, Prometheus, etc.)
Service / Ingress	Accès réseau simulé comme en prod

🛡 5. Sécurité et limites

- **Limiter l'accès** aux ressources (ex: via RBAC).
- **Eviter les effets de bord** : jamais d'appel à des systèmes de production (ex: paiements réels).
- **Ne pas persister les données sensibles**.

🛠 6. Exemple pratique : Déploiement d'une app de test

1. Créer un namespace :

```
kubectl create namespace preprod
```

2. Fichier `app-preprod.yaml` :

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: webapp
  namespace: preprod
spec:
  replicas: 2
  selector:
    matchLabels:
      app: webapp
  template:
    metadata:
      labels:
        app: webapp
    spec:
      containers:
        - name: web
          image: nginx
          ports:
            - containerPort: 80
```

3. Appliquer le déploiement :

```
kubectl apply -f app-preprod.yaml
kubectl get pods -n preprod
```

4. Exposer l'application :

```
kubectl expose deployment webapp --type=NodePort --port=80 -n preprod
minikube service webapp -n preprod
```

---

7. Mini-lab – Mise en place complète d'un environnement préprod

Objectif : déployer une version préprod d'une app avec des paramètres isolés

- Création d'un namespace
- Déploiement d'un `Deployment` avec `ConfigMap` et `Secret`
- Exposition via un `Service`
- Test avec `curl` ou navigateur

✓ Quiz rapide

1. Quel est l'intérêt du namespace "preprod" ?
a) Limiter l'utilisation du cluster
b) Isoler les ressources d'un environnement ✓
c) Protéger contre les virus
d) Cacher le code

2. Quelle ressource stocke des données sensibles ?
a) ConfigMap
b) Pod
c) Secret ✓
d) Deployment

3. Quelle commande permet de déployer dans un namespace spécifique ?
a) `kubectl apply -f fichier.yaml --scope=preprod`
b) `kubectl deploy -n preprod`
c) `kubectl apply -f fichier.yaml -n preprod` ✓
d) `kubectl -n fichier.yaml`

📌 À retenir :

- La pré-production est essentielle pour tester l'application dans des conditions réalistes.
- Utilise un **namespace dédié** pour séparer les ressources.
- Configure des variables, secrets et endpoints spécifiques.
- N'expose **jamais de données réelles** dans cet environnement.

Module 4 : Déploiement de l'application dans la pré-production

Objectifs pédagogiques :

- Appliquer un processus de déploiement automatisé dans un namespace de pré-production.
- Utiliser des outils comme `kubectl`, `Helm` ou des pipelines CI/CD.
- Vérifier la stabilité du déploiement via des tests post-déploiement.

---

1. Étapes classiques d'un déploiement en préprod

1. **Préparation du namespace** (ex. `preprod`)
2. **Application des ConfigMaps et Secrets**
3. **Déploiement de l'application (Deployment, Service, Ingress)**
4. **Exécution de tests automatiques (smoke tests)**
5. **Validation manuelle ou automatisée**
6. **Tagging de l'image pour passage en prod (optionnel)**

---

2. Exemple simple : Déploiement d'une web app dans le namespace `preprod`

2.1 Fichier `webapp-deployment.yaml`

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: webapp
  namespace: preprod
spec:
  replicas: 2
  selector:
    matchLabels:
      app: webapp
  template:
    metadata:
      labels:
        app: webapp
    spec:
      containers:
        - name: web
          image: myregistry.io/webapp:v1.2.3
          envFrom:
            - configMapRef:
                name: app-config
            - secretRef:
                name: db-secret
          ports:
            - containerPort: 80
```

✅ 2.2 Fichier `webapp-service.yaml`

```yaml
apiVersion: v1
kind: Service
metadata:
  name: webapp
  namespace: preprod
spec:
  selector:
    app: webapp
  ports:
    - port: 80
      targetPort: 80
  type: ClusterIP
```

🎁 Déploiement :

```
kubectl apply -f webapp-deployment.yaml
kubectl apply -f webapp-service.yaml
```

🔨 3. Intégrer un outil : Déploiement avec Helm

Helm est un gestionnaire de packages pour Kubernetes. Il permet d'automatiser et versionner les déploiements.

```
helm install webapp ./mychart -n preprod
```

Tu peux facilement redéployer une nouvelle version :

```
helm upgrade webapp ./mychart --set image.tag=v1.2.4 -n preprod
```

🚦 4. Tests de vérification (Smoke Tests)

Une fois le déploiement terminé :

- Vérifie que les Pods sont **"Running"** :

```
kubectl get pods -n preprod
```

- Effectue une requête sur le Service :

```
kubectl port-forward svc/webapp 8080:80 -n preprod
curl http://localhost:8080
```

- Optionnel : script de **tests automatiques** post-déploiement :

```
./tests/smoke-test.sh
```

🧪 5. Mini-lab – Déploiement d'une app complète

🎯 Objectif : Tu déploies une app Node.js en préprod avec Helm.

Étapes :

1. Créer un `namespace preprod`
2. Créer un chart Helm (ou utiliser un existant)
3. Ajouter les variables de configuration
4. Déployer avec Helm
5. Vérifier le fonctionnement via port forwarding

✓ Quiz rapide

1. Quelle ressource Kubernetes déploie une application à plusieurs réplicas ?
a) Service
b) Pod
c) Deployment ✓
d) Volume

2. Quelle commande déploie une app avec Helm ?
a) `helm deploy myapp`
b) `helm install myapp ./chart` ✓
c) `helm start app`
d) `kubectl apply mychart`

3. À quoi servent les smoke tests ?
a) Tester la sécurité
b) Vérifier que les conteneurs sont bien lancés ✓
c) Faire un benchmark de performance
d) Compresser les logs

📌 À retenir :

- Déployer une app en préprod suit un process structuré : config ➜ ressources ➜ vérifications.
- Helm permet de versionner, templater et réutiliser les déploiements.
- Les tests post-déploiement sont essentiels pour valider le succès du processus.

Module 5 : Introduction à la QA et au bug report

- Comprendre le rôle de la QA dans le cycle DevOps.
- Connaître les types de tests à réaliser en préproduction.
- Apprendre à rédiger des rapports de bugs clairs et utiles.
- Collaborer efficacement entre développeurs et testeurs.

1. C'est quoi la QA (Quality Assurance) ?

La **QA**, c'est l'ensemble des **méthodes et outils** qui visent à garantir que :

- le produit fonctionne correctement,
- les fonctionnalités répondent aux spécifications,
- et que les **bugs sont détectés avant la mise en prod**.

La QA **ne se résume pas aux tests** ! Elle englobe aussi :

- la documentation fonctionnelle,
- la stratégie de tests,
- l'automatisation des scénarios critiques,
- le suivi des anomalies.

---

2. Les différents types de tests en QA

Type de test	But principal	Exemple
Tests unitaires	Tester une fonction isolée	Méthode de calcul de prix
Tests d'intégration	Tester plusieurs modules ensemble	Connexion API + DB
Tests fonctionnels	Vérifier les fonctionnalités utilisateur	"Créer un compte" fonctionne-t-il ?
Tests UI/UX	Tester l'apparence et l'ergonomie	Responsive mobile
Tests de performance	Mesurer la charge supportée par l'appli	Test avec JMeter
Tests de régression	Vérifier que rien n'a cassé entre versions	Anciennes fonctionnalités OK ?
Smoke tests	Vérifier que l'app démarre sans erreur	HTTP 200 sur les pages clés

---

3. Outils populaires en QA

Usage	Outils
Tests automatiques	Jest, Mocha, PyTest, Selenium, Cypress
Tests de charge	JMeter, k6
Bug tracking	Jira, GitHub Issues, Trello
Gestion de campagne QA	TestRail, Xray, Zephyr
QA collaborative	Notion, Miro (pour specs et workflows)

📝 4. Comment rédiger un bug report efficace

Un bon rapport de bug doit être :

1. **Clair**
2. **Reproductible**
3. **Priorisé**
4. **Documenté avec des preuves (logs, captures, vidéos, etc.)**

✅ Exemple de template :

markdown

```
🧠 Titre : Impossible de valider le formulaire d'inscription

🎯 Environnement :
- Version app : v1.3.0
- OS/Browser : Chrome 123 / Windows 10
- Namespace : preprod

📝 Étapes pour reproduire :
1. Aller sur https://preprod.monsite.fr/signup
2. Remplir le formulaire avec des infos valides
3. Cliquer sur "S'inscrire"

💥 Résultat observé :
- Erreur 500 retournée

✅ Résultat attendu :
- L'utilisateur est redirigé vers la page de confirmation

📎 Pièces jointes :
- Capture écran
- Log navigateur (console)
- Vidéo Loom
```

✅ 5. Collaboration QA / Dev

Pour que la QA soit utile :

- Les développeurs doivent **faciliter les tests** (ex : logs lisibles, environnements stables, endpoints préprod).
- Les testeurs doivent **être précis dans leurs retours** (ex : logs clairs, captures, contexte).
- Utiliser un **outil centralisé** (Jira, GitHub Projects...) pour suivre les anomalies.

✅ Quiz rapide

1. Le rôle principal de la QA est de...
a) Valider la performance uniquement
b) Écrire la documentation technique
c) Garantir la qualité globale ✅
d) Gérer le budget de l'équipe

2. Quel type de test vérifie qu'un bug n'est pas revenu ?
a) Test de performance
b) Test de régression ✅
c) Test d'intégration
d) Test manuel

3. Qu'est-ce qu'un bon bug report doit contenir ?
a) Des critiques sur le développeur
b) Des étapes floues
c) Des preuves, un contexte, et un résultat attendu ✅
d) Rien, le bug suffit

📌 À retenir :

- La QA n'est pas là pour "fliquer" les devs, mais pour **prévenir les problèmes**.
- Des tests bien choisis permettent de **gagner du temps** sur le long terme.
- Un bon bug report = une résolution rapide.
- **Communication claire** = QA + Devs + Product qui avancent ensemble.

Module 6 : Orchestration des conteneurs

🎓 Objectifs pédagogiques :

- Comprendre ce qu'est l'orchestration de conteneurs.
- Identifier les composants clés de Kubernetes pour l'orchestration.
- Apprendre comment Kubernetes assure la **haute disponibilité**, la **scalabilité** et la **récupération automatique**.

🎁 1. Qu'est-ce que l'orchestration de conteneurs ?

L'orchestration, c'est le fait de **gérer automatiquement** :

- le déploiement des conteneurs,
- leur mise à l'échelle,
- leur mise à jour,
- leur redémarrage en cas d'erreur,
- leur communication réseau,
- la gestion du stockage.

➞ **Kubernetes** est l'outil le plus utilisé pour cette tâche. Il agit comme **un chef d'orchestre** de tous tes conteneurs (Docker, CRI-O, etc.).

2. Les composants de base de Kubernetes pour l'orchestration

Composant	Rôle
Pod	Unité minimale d'exécution (1+ conteneurs)
Deployment	Décrit comment et combien de Pods lancer
ReplicaSet	Assure que le bon nombre de Pods est en cours d'exécution
Service	Permet la communication entre Pods
Scheduler	Assigne les Pods aux bons nœuds du cluster
Controller Manager	Gère les boucles de contrôle (failover, scale, etc.)
Kubelet	Agent sur chaque nœud qui exécute les Pods
Etcd	Base de données clé-valeur pour l'état du cluster

3. Exemple d'orchestration automatique

Cas pratique :

Tu veux 3 copies (réplicas) de ton appli web :

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: webapp
spec:
  replicas: 3
  selector:
    matchLabels:
      app: webapp
  template:
    metadata:
      labels:
        app: webapp
    spec:
      containers:
      - name: webapp
        image: myapp:v1
        ports:
        - containerPort: 80
```

Si un Pod tombe : **Kubernetes en redémarre un automatiquement.**

Si tu veux plus de charge :

```
kubectl scale deployment webapp --replicas=10
```

🔌 4. Gestion du réseau entre Pods

Kubernetes **crée un réseau plat** dans lequel :

- chaque Pod a une **adresse IP unique**,
- les **Services** permettent de localiser dynamiquement les Pods,
- **DNS interne** permet aux Pods de se découvrir (`webapp.default.svc.cluster.local`).

🔐 5. ConfigMaps, Secrets et Volumes : orchestration des ressources annexes

- Les **ConfigMaps** injectent des variables d'environnement.
- Les **Secrets** gèrent les infos sensibles (tokens, mots de passe).
- Les **Volumes** permettent le stockage persistant des données.

➞ Tous ces éléments sont **liés aux Pods** via le `spec`.

📈 6. Autoscaling

Kubernetes permet le **scaling automatique** des Pods selon :

- le **CPU**, la **RAM** consommés (`HorizontalPodAutoscaler`)
- le **nombre de requêtes** (via custom metrics)

```
kubectl autoscale deployment webapp --cpu-percent=50 --min=2 --max=10
```

🛡 7. Résilience automatique

Kubernetes applique le **principe du self-healing** :

- Si un Pod meurt → il est redémarré.
- Si un Node tombe → les Pods sont redistribués.
- Si une update échoue → rollback automatique.

✅ Quiz rapide

1. Quel composant Kubernetes s'occupe de redémarrer les Pods automatiquement ?
a) Etcd
b) Controller Manager ✅
c) Scheduler
d) Service

2. Quel fichier contient la définition du nombre de Pods à créer ?
a) Secret
b) ConfigMap

c) Deployment ✓
d) Service

3. À quoi sert un Service dans Kubernetes ?
a) Installer les Pods
b) Gérer les identifiants secrets
c) Exposer des Pods sur le réseau ✓
d) Stocker les logs

📌 À retenir :

- Kubernetes orchestre les conteneurs pour assurer leur disponibilité et leur gestion automatique.
- Il repose sur des ressources clés comme les Deployments, Pods, Services et ReplicaSets.
- Il est **auto-réparateur**, **scalable** et **flexible**, ce qui en fait la base de toute stratégie DevOps moderne.

Module 7 : Stratégie de déploiement dans Kubernetes

🎓 Objectifs pédagogiques :

- Comprendre les différentes **stratégies de mise à jour** dans Kubernetes.
- Savoir quand utiliser **Rolling Update, Recreate, Blue/Green ou Canary**.
- Appliquer ces stratégies via YAML ou CI/CD.

⚙ 1. Pourquoi une stratégie de déploiement ?

Imagine : tu déploies une nouvelle version... et boum 💥, une régression s'infiltre en prod.
Les **stratégies de déploiement** permettent de :

- **minimiser l'impact** d'un bug,
- assurer une **mise à jour progressive** ou isolée,
- **revenir facilement en arrière**.

🌀 2. Rolling Update – ⚙ Par défaut dans Kubernetes

🔄 Principe :

Les Pods de l'ancienne version sont **remplacés progressivement** par ceux de la nouvelle version.

✓ Avantages :

- Aucun downtime.
- Facile à implémenter.

✗ Inconvénient :

- Si la nouvelle version a un bug, tu risques une propagation rapide.

📄 Exemple dans un Deployment :

```yaml
strategy:
  type: RollingUpdate
  rollingUpdate:
    maxSurge: 1
    maxUnavailable: 1
```

🧨 3. Recreate – La version "brute"

🔄 Principe :

Tous les anciens Pods sont supprimés, puis les nouveaux sont lancés.

✓ Avantage :

- Pas de conflit de version entre services backend/db.

✗ Inconvénients :

- Downtime.
- Risqué pour des apps en production live.

```
yaml
strategy:
  type: Recreate
```

🌐 4. Blue/Green Deployment

🛰 Principe :

- Deux environnements (blue = actuel, green = nouveau).
- Le trafic est routé vers l'un ou l'autre.
- Le switch est **instantané** une fois validé.

✓ Avantages :

- Rollback très simple (retour à l'env. précédent).
- Tests possibles en prod avant switch.

✗ Inconvénients :

- Double ressources nécessaires.
- Doit être géré avec un **Service ou Ingress personnalisé**.

Exemple simplifié :

```
bash
# Green déployé sous un nouveau label
kubectl set image deployment/webapp webapp=myapp:v2 --record

# Modifier le service pour pointer vers le nouveau label
kubectl label pods --selector=app=webapp version=green --overwrite
```

🐤 5. Canary Deployment – Le plus progressif

🐓 Principe :

Tu exposes **seulement un petit pourcentage** d'utilisateurs à la nouvelle version (ex: 10%).

✓ Avantages :	✗ Inconvénients :
• Tests réels avec utilisateurs réels. • Bascule progressive.	• Complexité de gestion du routage. • Doit souvent être géré avec **Istio, Linkerd, Flagger ou Argo Rollouts**.

Ex. avec Flagger + Istio :

```yaml
spec:
  canaryAnalysis:
    interval: 1m
    threshold: 5
    steps:
      - setWeight: 10
      - pause:
          duration: 2m
      - setWeight: 50
      - pause:
          duration: 5m
```

---

6. Quelle stratégie pour quel besoin ?

Besoin	Stratégie recommandée
Mise à jour rapide, peu risquée	Rolling Update ✓
Migration base de données	Recreate
Validation avant bascule finale	Blue/Green
Déploiement progressif (prod)	Canary

✓ Quiz rapide

1. Quelle stratégie coupe temporairement l'accès à l'application ?
a) Rolling Update
b) Recreate ✓
c) Blue/Green
d) Canary

2. La stratégie Blue/Green consiste à...
a) Supprimer les anciens Pods avant de créer les nouveaux
b) Mettre à jour un pod à la fois
c) Maintenir deux environnements en parallèle ✓
d) Avoir une version par couleur d'utilisateur

3. Quelle stratégie permet un rollback instantané sans redéploiement ?
a) Recreate
b) Blue/Green ✓
c) Rolling Update
d) Canary

Module 8 : Mise à niveau d'un cluster Kubernetes

🎓 Objectifs pédagogiques :

- Comprendre **pourquoi et quand** faire une mise à jour de cluster.
- Identifier les étapes critiques d'un **upgrade sécurisé**.
- Connaître les outils et commandes nécessaires à une mise à niveau sans interruption.
- Appliquer les bonnes pratiques de test, backup, et rollback.

❓ 1. Pourquoi mettre à jour un cluster Kubernetes ?

Raisons principales	Détails
Sécurité	Patchs de vulnérabilités
Stabilité	Correction de bugs critiques
Nouvelles features	Support de nouvelles API, optimisations
Dépréciation	Certaines API deviennent obsolètes

💡 Les mises à jour suivent un **cycle de publication** :

- Une nouvelle version mineure tous les 3-4 mois.
- Les versions majeures sont compatibles jusqu'à N-2.

🛠️ 2. Préparer un upgrade : checklist avant toute chose

✅ Vérifie la compatibilité avec :

- La version actuelle (ex : de 1.26 vers 1.27 = OK)
- Les outils associés (Ingress controller, CNI, etc.)
- Les images Docker utilisées

✅ Analyse les objets obsolètes (ex : `apps/v1beta1`, `extensions/v1beta1`) :

```
kubectl get all --all-namespaces -o yaml | grep apiVersion
```

✅ Sauvegarde l'état du cluster :

```
ETCDCTL_API=3 etcdctl snapshot save snapshot.db
```

✅ Teste l'upgrade sur un **cluster de préproduction** ou avec **Kind / Minikube**.

🧰 3. Outils d'upgrade selon ton setup

Environnement	Outils recommandés
Kubeadm	`kubeadm upgrade`
Managed (GKE, EKS, AKS)	Console UI / CLI (`gcloud`, `eksctl`, etc.)
K3s / RKE	Scripts d'upgrade / `k3s upgrade`
Rancher	Interface Rancher (très user-friendly)

🔁 4. Processus type avec kubeadm (cluster on-prem)

🧪 Étapes :

1. **Mettre à jour kubeadm**

```
apt-get install -y kubeadm=1.27.3-00
kubeadm version
```

2. **Planifier l'upgrade**

```
kubeadm upgrade plan
```

3. **Lancer l'upgrade master**

```
kubeadm upgrade apply v1.27.3
```

4. **Mettre à jour kubelet et kubectl**

```
apt-get install -y kubelet=1.27.3-00 kubectl=1.27.3-00
systemctl restart kubelet
```

5. **Mettre à jour les nœuds un par un** :
   - Drain :

```
kubectl drain <node> --ignore-daemonsets
```

   - Upgrade :

```
kubeadm upgrade node
apt install -y kubelet=1.27.3-00
systemctl restart kubelet
```

   - Uncordon :

```
kubectl uncordon <node>
```

🧪 5. Tester le cluster après mise à jour

✓ Vérifie que tous les pods sont **Running**
✓ Teste les services exposés

✓ Contrôle les logs (`kubectl logs`)
✓ Vérifie les Custom Resource Definitions (CRDs) et les API versions

---

🆘 6. En cas de souci : rollback possible ?

- Si etcd a été sauvegardé, tu peux restaurer l'état précédent :

```
etcdctl snapshot restore snapshot.db
```

⚠ Pas de rollback natif dans Kubernetes ! C'est pour ça qu'on recommande **une phase de test + snapshot**.

✅ Quiz rapide

1. Quelle commande permet de planifier un upgrade avec kubeadm ?
a) `kubeadm update`
b) `kubeadm upgrade preview`
c) `kubeadm upgrade plan` ✅
d) `kubectl upgrade dry-run`

2. Quelle action est indispensable avant l'upgrade ?
a) Supprimer les pods manuellement
b) Sauvegarder ETCD ✅
c) Redémarrer tous les workers
d) Désactiver le kubelet

3. Peut-on faire un rollback automatique en cas d'échec ?
a) Oui, toujours
b) Oui mais uniquement sur GKE
c) Non, d'où l'importance du snapshot ✅
d) Seulement si on est sur version LTS

📌 À retenir :

- Toujours **préparer, tester et sauvegarder** avant d'upgrader.
- Utilise les outils adaptés à ton environnement (kubeadm, GKE, Rancher...).
- La mise à jour peut se faire sans downtime avec une bonne orchestration.
- Pas de rollback automatique : **le snapshot ETCD est ton assurance vie**.

Module 9 : Mise en production avec Kubernetes

🎓 Objectifs pédagogiques :

- Savoir préparer un déploiement **production-ready**.
- Définir les bonnes pratiques de mise en prod avec Kubernetes.
- Comprendre la surveillance, les alertes, le rollback et les tests post-déploiement.
- Garantir **disponibilité**, **sécurité** et **traçabilité**.

✅ 1. Prérequis avant mise en prod

Avant de lancer ton appli sur le cluster prod, assure-toi d'avoir :

Élément	Pourquoi c'est essentiel
✅ Liveness/Readiness Probes	Contrôle de santé des Pods
✅ Namespace dédié (ex: `prod`)	Isolation logique
✅ Autoscaling activé	Réponse à la charge
✅ Secrets chiffrés	Sécurité des tokens, clés API, etc.
✅ Ressources limitées	Empêcher un Pod d'engloutir la RAM/CPU
✅ Monitoring + Alerting	Savoir réagir aux incidents
✅ RBAC et NetworkPolicies	Sécurité et segmentation réseau

🚀 2. Déploiement : la checklist finale

🔄 Étapes typiques :

1. **Vérification de la version**
   - Check que l'image container est bien la dernière (`v1.4.2` par ex).
2. **Lancer le déploiement via GitOps ou CI/CD**
   - Ex: ArgoCD, FluxCD, GitLab CI avec `kubectl apply -f`
3. **Contrôle de l'état post-deploy**

```
kubectl rollout status deployment webapp
kubectl get pods -n prod
```

4. **Test des endpoints critiques**
   - **`/api/health`, `/status`, etc.**

⁉ 3. Intégrer les probes Kubernetes

Exemple :

```yaml
livenessProbe:
  httpGet:
    path: /healthz
    port: 8080
  initialDelaySeconds: 5
  periodSeconds: 10

readinessProbe:
  httpGet:
    path: /ready
    port: 8080
  initialDelaySeconds: 5
  periodSeconds: 5
```

➞ Ces probes garantissent que **seuls les pods sains** reçoivent du trafic.

🔒 4. Sécurité & stabilité

- **RBAC** : limite qui peut faire quoi (utilisateurs, services)
- **PodSecurityContext** : ne jamais tourner en root si possible
- **NetworkPolicy** : limite la communication inter-Pod
- **Secrets** stockés dans Kubernetes ou dans un Vault externe (ex : HashiCorp Vault)

📈 5. Monitoring et observabilité

Un cluster de production sans observabilité, c'est comme piloter un avion sans tableau de bord ✈

Outil	Fonction
Prometheus	Metrics & alertes
Grafana	Visualisation
Loki / EFK	Logs
Kiali / Jaeger	Traces (service mesh)
Sentry	Erreurs app front/back

🔁 6. Rollback : si tout déraille

```
kubectl rollout undo deployment webapp
```

✅ Tu peux revenir à la version précédente instantanément

💡 Bonus : active `revisionHistoryLimit` pour limiter l'historique

🛠️ 7. Bonnes pratiques de mise en prod

- **Déployer en semaine, le matin** (jamais le vendredi 17h 😅)
- **Déployer par petits lots**, en **rolling** ou **canary**
- **Activer les alertes Slack/Teams** sur les échecs de déploiement
- **Taguer et versionner** toutes les images (`app:v1.4.3`)
- **Loguer les dates, commits, personnes impliquées**

✅ Quiz final

1. Quel outil est utilisé pour monitorer les métriques dans Kubernetes ?
a) Fluentd
b) Prometheus ✅
c) Grafana
d) Docker Stats

2. Quelle commande permet un retour à une version précédente d'un déploiement ?
a) `kubectl delete deployment`
b) `kubectl rollback`
c) `kubectl rollout undo` ✅
d) `kubectl revert`

3. Que permet une `readinessProbe` ?
a) Savoir si l'app va bien au démarrage
b) Empêcher le Pod de crasher
c) Déterminer si le Pod est prêt à recevoir du trafic ✅
d) Supprimer le Pod s'il échoue

📌 À retenir :

- Une mise en prod réussie repose sur **préparation, sécurité, surveillance, et rollback possible**.
- Utilise les outils comme Prometheus, Grafana, RBAC, probes pour **assurer fiabilité & visibilité**.
- **Déploie progressivement**, et toujours en conditions maîtrisées.