Lab 1 - Architecture Kubernetes Multi-Namespaces

Objectif

Mettre en place une architecture Kubernetes organisée avec des namespaces dédiés pour un projet type "Random", en suivant les bonnes pratiques de séparation des responsabilités.

Contexte

Vous devez préparer un cluster Kubernetes pour héberger une application composée de plusieurs couches : backend, base de données, frontend, jobs batch et ordonnanceur.

Prérequis

- Cluster Kubernetes accessible (minikube, k3s, ou cluster cloud)
- kubectl configuré
- Droits administrateur sur le cluster

Architecture Cible

```
Cluster Kubernetes
├── random-backend      (Services API)
├── random-db           (PostgreSQL)
├── random-frontend     (Interface utilisateur)
├── random-jobs         (Pods Airflow workers)
└── random-scheduler    (Airflow scheduler)
```

Exercices

Exercice 1 : Création des Namespaces (15 min)

Créer les 5 namespaces nécessaires au projet Random. Chaque namespace doit être dédié à une fonction spécifique de l'architecture.

Livrables :

- Fichier YAML contenant tous les namespaces
- Capture d'écran de la commande `kubectl get namespaces`

Exercice 2 : Configuration des ResourceQuotas (20 min)

Limiter les ressources pour éviter qu'un namespace consomme toutes les ressources du cluster.

Contraintes :

- random-backend : CPU 4-8 cores, Memory 8-16Gi, max 10 pods
- random-jobs : CPU 8-16 cores, Memory 16-32Gi, max 20 pods
- random-db : CPU 2-4 cores, Memory 4-8Gi, max 5 pods
- random-frontend : CPU 1-2 cores, Memory 2-4Gi, max 8 pods

- random-scheduler : CPU 1-2 cores, Memory 2-4Gi, max 3 pods

Livrables :

- Fichier YAML avec tous les ResourceQuotas
- Vérification avec `kubectl describe quota`

Exercice 3 : LimitRanges par Défaut (15 min)

Définir des limites par défaut pour les pods sans spécifications de ressources dans chaque namespace.

Exigences :

- Définir des default requests et limits cohérents avec les ResourceQuotas
- Adapter les valeurs selon la fonction de chaque namespace
- Les jobs Spark (random-jobs) nécessitent plus de ressources

Livrables :

- Fichier YAML avec les LimitRanges pour tous les namespaces

Exercice 4 : Network Policies (30 min)

Sécuriser la communication entre namespaces selon les règles suivantes :

- La base de données (random-db) ne doit être accessible que depuis random-backend et random-jobs sur le port 5432
- Le backend (random-backend) ne doit être accessible que depuis random-frontend sur le port 8080
- Le scheduler (random-scheduler) doit pouvoir accéder à random-jobs
- Tout autre trafic doit être bloqué par défaut

Prérequis :

- Labelliser correctement tous les namespaces avant d'appliquer les policies

Livrables :

- Fichier YAML avec toutes les NetworkPolicies
- Documentation expliquant chaque règle
- Test de connectivité entre namespaces

Exercice 5 : RBAC - Contrôle d'Accès (25 min)

Créer des ServiceAccounts, Roles et RoleBindings pour chaque composant.

Permissions requises :

- random-backend-sa : lecture seule (get, list, watch) sur pods, services, configmaps, secrets
- random-jobs-sa : création/suppression de Jobs, lecture sur pods, configmaps, secrets
- random-scheduler-sa : permissions étendues pour orchestrer les jobs
- random-frontend-sa : lecture seule minimale
- random-db-sa : lecture sur secrets uniquement

Livrables :

- Fichiers YAML pour tous les ServiceAccounts, Roles et RoleBindings
- Tests de permissions avec `kubectl auth can-i`

Exercice 6 : Labels et Annotations (10 min)

Standardiser les labels pour faciliter la gestion et la traçabilité.

Labels obligatoires :

- env: production
- app: random
- component: (backend|database|frontend|jobs|scheduler)

Annotations obligatoires :

- description: Description du rôle du namespace
- contact: Email de l'équipe responsable
- Pour random-db : alerte sur la surveillance du PVC

Livrables :

- Script bash appliquant tous les labels et annotations
- Documentation des conventions de nommage

Exercice 7 : Script d'Automatisation (20 min)

Créer un script bash qui :

- Crée tous les namespaces
- Applique tous les ResourceQuotas
- Applique tous les LimitRanges
- Configure le RBAC
- Applique les labels et annotations
- Vérifie que tout est correctement déployé

Livrables :

- Script `setup-namespaces.sh`
- Script `verify-namespaces.sh`
- Script `cleanup-namespaces.sh` pour tout supprimer

Points Critiques

 ALERTE IMPORTANTE (Issue du document de passation) : "Il est crucial de surveiller la saturation du PVC de la base de données PostgreSQL de Random en risque d'avoir une interruption des services"

Actions à documenter :

- Comment configurer un StorageClass avec expansion automatique
- Où placer des alertes de monitoring sur l'utilisation du PVC
- Quelle stratégie de backup prévoir

Validation Finale

Checklist :

- [ ] 5 namespaces créés et labellisés
- [ ] ResourceQuotas configurés et vérifiés
- [ ] LimitRanges définis pour tous les namespaces

- [ ] NetworkPolicies appliquées et testées
- [ ] ServiceAccounts et RBAC configurés
- [ ] Labels et annotations standardisés
- [ ] Scripts d'automatisation fonctionnels
- [ ] Documentation complète

Tests de Validation :

- Déployer un pod test dans chaque namespace et vérifier les limites
- Tester la connectivité réseau entre namespaces
- Vérifier les permissions RBAC avec différents ServiceAccounts
- Simuler une saturation de quota et observer le comportement

QCM - Évaluation des Connaissances

Question 1

Pourquoi est-il important de séparer la base de données dans un namespace dédié ?

- A) Pour des raisons esthétiques
- B) Pour isoler les ressources critiques et appliquer des politiques de sécurité spécifiques
- C) Parce que Kubernetes l'exige
- D) Pour réduire les coûts

Question 2

Quelle est la différence principale entre un ResourceQuota et un LimitRange ?

- A) Aucune différence, ce sont des synonymes
- B) ResourceQuota limite les ressources totales du namespace, LimitRange limite les ressources par pod
- C) LimitRange est obsolète, il faut utiliser ResourceQuota
- D) ResourceQuota est pour le CPU, LimitRange pour la mémoire

Question 3

Dans une NetworkPolicy, que signifie une règle sans ingress ni egress définis ?

- A) Tout le trafic est autorisé
- B) Tout le trafic est bloqué
- C) Seul le trafic HTTP est autorisé
- D) La policy est invalide

Question 4

Quel paramètre d'un StorageClass permet l'extension automatique d'un PVC ?

- A) autoExpand: true
- B) allowVolumeExpansion: true
- C) dynamicProvisioning: true
- D) expandable: true

Question 5

Pourquoi utiliser un StatefulSet plutôt qu'un Deployment pour PostgreSQL ?

- A) C'est plus moderne
- B) Pour garantir une identité stable et un ordre de démarrage prévisible
- C) Les Deployments ne supportent pas les volumes
- D) C'est obligatoire pour les bases de données

Question 6

Dans le contexte RBAC, quelle est la portée d'un Role (vs ClusterRole) ?

- A) Un Role s'applique à tout le cluster
- B) Un Role s'applique uniquement au namespace où il est défini
- C) Un Role s'applique uniquement aux ServiceAccounts
- D) Aucune différence, les deux sont identiques

Question 7

Que se passe-t-il si un pod demande plus de ressources que le LimitRange autorise ?

- A) Le pod est créé avec les limites du LimitRange
- B) Le pod est rejeté et ne démarre pas
- C) Un warning est émis mais le pod démarre quand même
- D) Le scheduler attend que des ressources se libèrent

Question 8

Selon le document de passation, quel est le risque critique à surveiller pour random-db ?

- A) La saturation CPU
- B) La saturation du PVC (stockage)
- C) Le nombre de connexions
- D) La latence réseau

Question 9

Quelle commande permet de labelliser un namespace existant ?

- A) kubectl label namespace key=value
- B) kubectl set label namespace key=value
- C) kubectl annotate namespace key=value
- D) kubectl tag namespace key=value

Question 10

Pourquoi utiliser `volumeBindingMode: WaitForFirstConsumer` dans un StorageClass ?

- A) Pour accélérer le provisioning
- B) Pour que le volume soit créé dans la même zone que le pod qui l'utilise
- C) Pour économiser des ressources
- D) C'est obligatoire pour les PVC

Réponses

1. B | 2. B | 3. B | 4. B | 5. B | 6. B | 7. B | 8. B | 9. A | 10. B