

Mini Rapport

Projet Programmation Distribuee

Online Bookstore - Architecture Microservices

Etudiantes : Aya CHIHOUB, Nour El Imene KHELASSI

Formation : Master 1 VMI

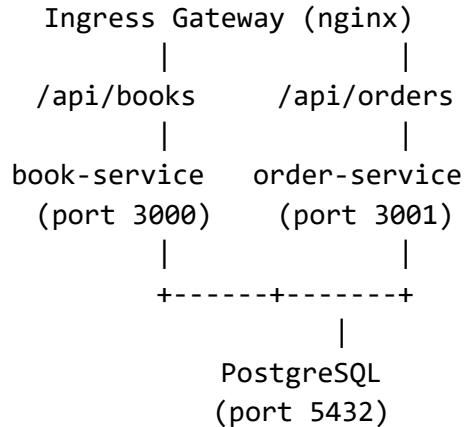
Date : Fevrier 2026

Professeur : Benoit Charroux

Repository GitHub : github.com/Aya-chihoub/Projet-Programmation-Distribuee-M1

1. Architecture du Projet

Le projet est composé de deux microservices communiquant entre eux, déployés dans un cluster Kubernetes (Minikube) avec une base de données PostgreSQL et un Ingress Gateway pour le routage des requêtes.



Technologies utilisées

Technologie	Utilisation
Node.js + Express	Microservices REST API
PostgreSQL 16	Base de données
Docker	Conteneurisation
Kubernetes (Minikube)	Orchestration
Ingress (nginx)	Gateway / Routage
RBAC	Sécurité du cluster
Docker Hub	Registry des images
GitHub	Gestion du code source

2. Microservice : book-service

Le book-service gère le catalogue de livres de la librairie en ligne. Il expose une API REST complète (CRUD) et se connecte à PostgreSQL pour la persistance des données. L'image Docker est disponible sur Docker Hub : ayach10/book-service:latest

Endpoints REST API

Méthode	Endpoint	Description
GET	/api/books	Liste tous les livres
GET	/api/books/:id	Recupere un livre par ID
POST	/api/books	Cree un nouveau livre
PUT	/api/books/:id	Met a jour un livre
DELETE	/api/books/:id	Supprime un livre
GET	/health	Health check pour Kubernetes

Reponse API book-service

```
PS C:\Users\DELL\Documents\Projet_Programmation_distribuee> curl http://localhost:3000/api/books
```

```
Security Warning: Script Execution Risk  
Invoke-WebRequest parses the content of the  
web page. Script code in the web page  
might be run when the page is parsed.
```

```
RECOMMENDED ACTION:
```

```
Use the -UseBasicParsing switch to  
avoid script code execution.
```

```
Do you want to continue?
```

```
[Y] Yes [A] Yes to All [N] No  
[L] No to All[S] Suspend [?] Help  
(default is "N"):y
```

```
StatusCode      : 200  
StatusDescription : OK  
Content        : [{"id":1,"title":"Clean  
Code","author":"Robert  
C. Martin","price":"29.  
99","stock":15,"created
```

```

    _at": "2026-02-26T11:50:
30.134Z"}, {"id": 2, "titl
e": "Design Patterns", "a
uthor": "Gang of Four", "
price": "39.99", ...
RawContent : HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Connection: keep-alive
Keep-Alive: timeout=5
Content-Length: 637
Content-Type:
application/json;
charset=utf-8
Date: Sat, 28 Feb 2026
16:38:55 GMT...
Forms : {}
Headers : {[Access-Control-Allow-Origin, *],
[Connection,
keep-alive],
[Keep-Alive,
timeout=5],
[Content-Length,
637]...}
Images : {}

```

```

InputFields : {}
Links : {}
ParsedHtml : System.__ComObject
RawContentLength : 637

```

3. Microservice : order-service

Le order-service gère les commandes de livres. Lorsqu'une commande est créée, il appelle le book-service via le DNS interne Kubernetes (<http://book-service:3000>) pour vérifier que le livre existe avant d'enregistrer la commande en base de données. L'image Docker est disponible sur Docker Hub : [nourno/order-service:latest](https://hub.docker.com/r/nourno/order-service)

Endpoints REST API

Méthode	Endpoint	Description
GET	/api/orders	Liste toutes les commandes
POST	/api/orders	Créer une commande (appelle book-service)
GET	/health	Health check pour Kubernetes

Communication inter-services

Le order-service communique avec le book-service via le DNS interne de Kubernetes. Quand un client envoie une requête POST /api/orders avec un book_id, le order-service appelle `http://book-service:3000/api/books/{book_id}` pour vérifier l'existence du livre avant de créer la commande.

Reponse API order-service

```
PS C:\Users\DELLcurl http://localhost:3001/api/ordersibuee>

Security Warning: Script Execution Risk
Invoke-WebRequest parses the content of the web page. Script code in the web page
might be run when the page is parsed.
    RECOMMENDED ACTION:
        Use the -UseBasicParsing switch to avoid script code execution.

    Do you want to continue?

[Y] Yes [A] Yes to All [N] No [L] No to All [S] Suspend [?] Help
(default is "N"):y

StatusCode      : 200
StatusDescription : OK
Content         : [{"id":1,"book_id":1,"customer_name":"Aya","quantity":2,"stat
us":"completed","created_at":"2026-02-28T16:29:13.229Z"}]
RawContent      : HTTP/1.1 200 OK
                  Access-Control-Allow-Origin: *
                  Connection: keep-alive
                  Keep-Alive: timeout=5
                  Content-Length: 118
                  Content-Type: application/json; charset=utf-8
                  Date: Sat, 28 Feb 2026 16:40:05 GMT...
Forms           : {}
Headers         : {[Access-Control-Allow-Origin, *], [Connection, keep-alive],
                  [Keep-Alive, timeout=5], [Content-Length, 118]...}
Images          : {}
InputFields     : {}
Links           : {}
ParsedHtml      : System.__ComObject
RawContentLength : 118
```

4. Docker

Chaque microservice possède son propre Dockerfile basé sur node:20-alpine (book-service) et node:24-alpine (order-service). Les images sont construites et publiées sur Docker Hub.

Service	Image Docker Hub	Port
book-service	ayach10/book-service:latest	3000
order-service	nourno/order-service:latest	3001

5. Kubernetes

L'ensemble de l'application est déployé dans un cluster Kubernetes local via Minikube. Chaque composant dispose de son propre Deployment et Service.

Ressources déployées

Ressource	Nom	Détails
Deployment	book-service	2 replicas
Deployment	order-service	1 replica
Deployment	postgres	1 replica
Service	book-service	ClusterIP, port 3000
Service	order-service	ClusterIP, port 3001
Service	postgres	ClusterIP, port 5432
PersistentVolumeClaim	postgres-pvc	1Gi de stockage
Ingress	bookstore-ingress	Routage /api/books et /api/orders

Pods en cours d'exécution

kubectl get pods					
NAME	READY	STATUS	RESTARTS	AGE	
book-service-6b656dbcc6-2wht4	1/1	Running	0	2d4h	
book-service-6b656dbcc6-d4dlx	1/1	Running	1 (2d4h ago)	2d4h	
myservice-746766f779-6gdwn	1/1	Running	0	10d	
myservice-746766f779-b7cj4	1/1	Running	0	10d	
order-service-6c7cbc67f-qdpqr	1/1	Running	0	4m13s	
postgres-6cd57587b6-z9kwb	1/1	Running	1 (2d4h ago)	2d4h	

Services Kubernetes

PS C:\Users\DELL\Documents\Projet_Programmation_distribuée> kubectl get svc					
NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
book-service	ClusterIP	10.96.193.120	<none>	3000/TCP	2d4h
kubernetes	ClusterIP	10.96.0.1	<none>	443/TCP	10d
myservice	NodePort	10.106.126.108	<none>	8080:30099/TCP	10d
order-service	ClusterIP	10.99.187.148	<none>	3001/TCP	4m55s
postgres	ClusterIP	10.99.51.212	<none>	5432/TCP	2d4h

6. Base de Données PostgreSQL

PostgreSQL 16 est déployé dans le cluster Kubernetes avec un PersistentVolumeClaim de 1Gi pour assurer la persistance des données. La base 'bookstore' contient deux tables :

Table	Créé par	Colonnes principales
books	book-service	id, title, author, price, stock, created_at
orders	order-service	id, book_id, customer_name, quantity, status, created_at

PersistentVolumeClaim

```
PS C:\Users\DELL\Documents\Projet_Programmation_distribuee> kubectl get pvc
NAME      STATUS  VOLUME          CAPACITY  ACCESS MODES  STORAGE
ECLASS   VOLUME ATTRIBUTES CLASS     AGE
postgres-pvc Bound   pvc-fd06a8df-cb2b-48b4-a856-fec5b7a63451  1Gi       RWO          standard
rd        <unset>          2d4h
```

7. Ingress Gateway

Un Ingress NGINX est configurer pour router les requetes entrantes vers les microservices correspondants :

Chemin	Service cible	Port
/api/books	book-service	3000
/api/orders	order-service	3001

Ingress

```
PS C:\Users\DELL\Documents\Projet_Programmation_distribuee> kubectl get ingress
NAME      CLASS  HOSTS          ADDRESS    PORTS  AGE
bookstore-ingress  nginx  *           192.168.49.2  80    2m33s
example-ingress    nginx  myservice.info  192.168.49.2  80    10d
```

8. Securite - RBAC

Des controles d'accès ont été mis en place via RBAC (Role-Based Access Control) de Kubernetes pour limiter les permissions de chaque composant :

Ressource	Nom	Permissions
ServiceAccount	book-service-sa	Attache au deployment book-service
ServiceAccount	postgres-sa	Attache au deployment postgres
Role	book-service-role	Lecture pods, services, configmaps
Role	postgres-role	Lecture pods, PVCs
RoleBinding	book-service-rolebinding	Lie book-service-sa au role
RoleBinding	postgres-rolebinding	Lie postgres-sa au role

RBAC

```
PS C:\Users\DELL\Documents\Projet_Programmation_distribuee> kubectl get serviceaccounts,roles,rolebindings
NAME                                     AGE
serviceaccount/book-service-sa          2d4h
serviceaccount/default                  10d
serviceaccount/postgres-sa             2d4h

NAME                                         CREATED AT
role.rbac.authorization.k8s.io/book-service-role 2026-02-26T11:54:24Z
role.rbac.authorization.k8s.io/postgres-role    2026-02-26T11:54:24Z

NAME                                     ROLE           AGE
rolebinding.rbac.authorization.k8s.io/book-service-rolebinding Role/book-service-role 2d4h
rolebinding.rbac.authorization.k8s.io/postgres-rolebinding     Role/postgres-role   2d4h
```

9. GitHub

Le code source complet est disponible sur GitHub :

<https://github.com/Aya-chihoub/Projet-Programmation-Distribuee-M1>

10. Conclusion

Ce projet nous a permis de mettre en pratique les concepts de la programmation distribuee a travers la realisation d'une application microservices complete. Nous avons integre les technologies suivantes :

- Developpement de microservices REST avec Node.js et Express
- Conteneurisation avec Docker et publication sur Docker Hub
- Orchestration avec Kubernetes (Minikube)
- Base de donnees PostgreSQL avec persistance (PVC)
- Communication inter-services via DNS interne Kubernetes
- Routage des requetes via Ingress Gateway (NGINX)
- Securisation du cluster avec RBAC (ServiceAccounts, Roles, RoleBindings)