Rapport

Projet Programmation Distribuee

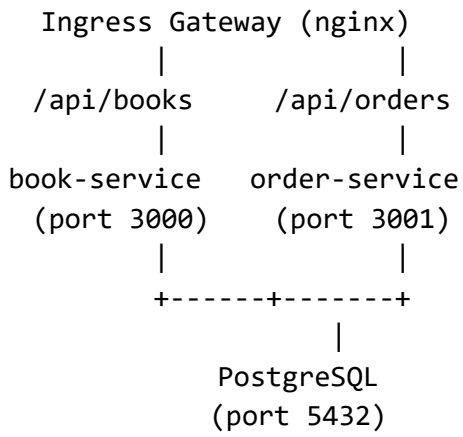
Online Bookstore - Architecture Microservices

Etudiants : Aya CHIHOUB, Nour EL Imene KHELASSI
Formation : Master 1 VMI
Date : Mars 2026
Professeur : Benoit Charroux

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

1. Architecture du Projet

Le projet est compose de deux microservices communiquant entre eux, deployes dans un cluster Kubernetes (Minikube) avec une base de donnees PostgreSQL et un Ingress Gateway pour le routage des requetes.



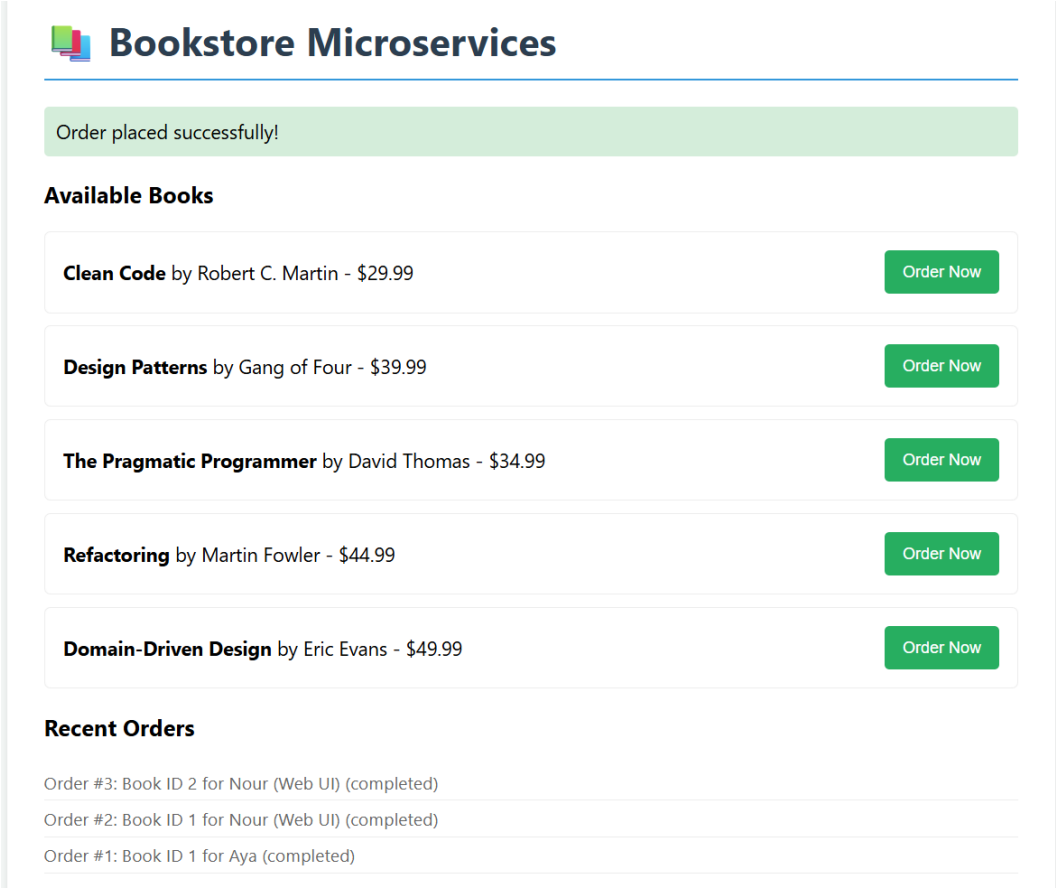
Technologies utilisees

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

2. Interface Web (Dashboard)

Une interface web a ete developpee pour permettre aux utilisateurs de visualiser les livres disponibles, passer des commandes et consulter les commandes recentes. L'interface communique avec les microservices via les APIs REST.

Interface



3. Microservice : book-service

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

Endpoints REST API

Methode	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

```
                    _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-Or
                    igin: *
                    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
```

order-service

Le order-service gere les commandes de livres. Lorsqu'une commande est creee, il appelle le book-service via le DNS interne Kubernetes (http://book-service:3000) pour verifier que le livre existe avant d'enregistrer la commande en base de donnees. L'image Docker est disponible sur Docker Hub : nourno/order-service:latest

Endpoints REST API

Methode	Endpoint	Description
GET	/api/orders	Liste toutes les commandes
POST	/api/orders	Cree 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 requete POST /api/orders avec un book_id, le order-service appelle http://book-service:3000/api/books/{book_id} pour verifier l'existence du livre avant de creer 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
```

5. Docker

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

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

6. Kubernetes

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

Ressources deployees

Ressource	Nom	Details
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'execution

```
                                                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_distribuee> 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
```

7. Base de Donnees PostgreSQL

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

Table	Creee par	Colonnes principales
books	book-service (Aya)	id, title, author, price, stock, created_at
orders	order-service (Nour)	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   STORAG
ECLASS   VOLUMEATTRIBUTESCLASS   AGE
postgres-pvc   Bound    pvc-fd06a8df-cb2b-48b4-a856-fec5b7a63451   1Gi        RWO            standa
rd       <unset>                 2d4h
```

8. Ingress Gateway

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

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

Ingress

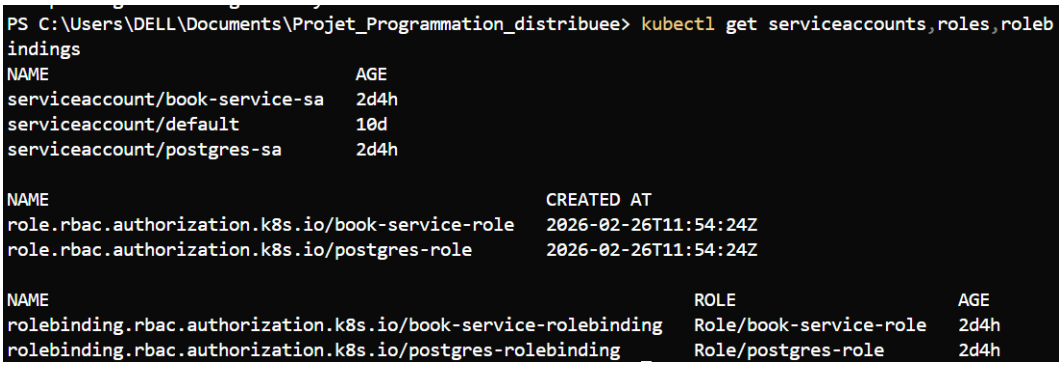
```
PS C:\Users\DELL\Documents\Projet_Programmation_distribuee> kubectl get ingress
NAME                CLASS   HOSTS            ADDRESS        PORTS   AGE
bookstore-ingress   nginx   *                192.168.49.2   80      2m33s
example-ingress     nginx   myservice.info   192.168.49.2   80      10d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

9. Securite - RBAC

Des controles d'acces ont ete 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

Capture d'ecran : RBAC



10. GitHub

Le code source complet est disponible sur GitHub :

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

11. 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)
- Interface web pour la visualisation et la gestion des commandes