

A close-up photograph of a traditional dreamcatcher. It features a circular web made of black thread with several red, white, and blue beads attached. Long, light-colored feathers hang from the bottom and sides of the web. The background is a soft, out-of-focus yellow.

Module 4AMS – Microservices

Jour 4 – Web Services, Docker & Avancés

Yassen ABARJI – 16/05/2025

Objectifs pédagogiques

- Documenter une API avec Swagger / OpenAPI
- Déployer un microservice avec Spring Boot & Docker
- Exécuter plusieurs microservices sur des ports distincts
- Comprendre et implémenter un WebSocket
- Consolider les acquis par un atelier final

Partie 1 – REST avancé & OData

Objectif :

Comprendre comment enrichir une API REST avec des requêtes dynamiques et propres.

Contenu :

- Structuration REST (ressources, verbes HTTP)
- Paramètres : path, query, header, body
- Introduction à OData (filtrage, tri, pagination dynamiques)

 Exemple :

```
@GetMapping("/products")
public List<Product> list(
    @RequestParam String category,
    @RequestParam(defaultValue="asc") String sort
) {
    // Logique ici
}
```

Exercice 1 – Ressource Student

Créer un microservice Student avec :

- GET /students (liste complète)
- GET /students/{id} (par ID)
- POST /students (création)
- PUT /students/{id} (modification)
- DELETE /students/{id} (suppression)
-
- bonus : GET /students?promotion=2025

Utiliser @PathVariable et @RequestParam dans les contrôleurs.

Partie 2 – Documentation avec Swagger

Objectif :

Documenter son API automatiquement avec OpenAPI 3 (Swagger).

Étapes :

- Ajouter springdoc-openapi-ui au pom.xml
- Lancer le projet → accéder à /swagger-ui.html
- Utiliser @Operation, @Parameter, @Schema pour enrichir

 Exemple :

```
@Operation(summary = "Lister les produits", description = "Retourne tous les produits disponibles")
@GetMapping("/products")
public List<Product> all() {
    // code
}
```

Exercice 2 – Swagger

Objectif : Ajouter une documentation lisible à votre microservice.

- 1 Ajouter la dépendance springdoc dans pom.xml
 - 2 Annoter vos méthodes avec @Operation
 - 3 Vérifier que tous vos endpoints apparaissent
 - 4 Bonus : ajouter des descriptions de paramètres et réponses
- ✓ Utile pour générer une doc de projet claire et pro

Partie 3 – Déploiement local & ports

Objectif :

Faire fonctionner plusieurs microservices en parallèle

- ◉ Spring Boot embarque un serveur Tomcat
- ◉ On peut spécifier un port pour chaque microservice

```
# application.properties  
server.port=8082
```

Exécution :

- UserService sur 8081
- ProductService sur 8082

Vérifier le fonctionnement via navigateur ou Postman

Partie 4 – Dockerisation

Objectif :

Conteneuriser une application Spring Boot avec Docker

Pourquoi Docker ?

- Isoler chaque microservice
- Faciliter la livraison et l'exécution

Dockerfile type :

```
FROM openjdk:17
COPY target/myapp.jar app.jar
ENTRYPOINT ["java", "-jar", "/app.jar"]
```

Commandes à connaître :

```
docker build -t myservice .
docker run -p 8080:8080 myservice
```

Exercice 3 – Docker

Créer un Dockerfile pour votre projet actuel.

1 Générer le JAR avec Maven

```
mvn clean package
```

1 Construire l'image Docker

2 Lancer le conteneur

✓ Vérifier que l'API est accessible sur le port exposé

Partie 5 – WebSocket

Objectif :

Découvrir les communications temps réel avec Spring

- REST = client demande / WebSocket = serveur pousse
- Utiliser Spring Messaging (STOMP)
- Créer un canal /topic/messages

💡 Exemple :

```
@MessageMapping("/notify")
@SendTo("/topic/messages")
public String notifyAll(String msg) {
    return "Nouveau message : " + msg;
}
```

✓ Tester avec un client JS ou une page HTML simple



Atelier final – Consolidation

Objectif :

Mettre en pratique tous les concepts vus dans le module.

À réaliser en binôme/trio :

- ◉ Swagger documenté
- ◉ Dockerfile fonctionnel
- ◉ REST complet (CRUD + query)
- ◉ Exécution sur port défini
- ◉ Bonus : ajout WebSocket ou Actuator

⌚ Durée : 1h30 à 2h



Checklist projet - Soutenance

- ✓ API REST complète (CRUD + filtres)
- ✓ Swagger visible et structuré (/swagger-ui.html)
- ✓ Dockerfile + README pour build/run
- ✓ Logs fonctionnels
- ✓ Monitoring actif (optionnel)
- ✓ Chaque microservice sur un port propre
- ✓ Démo fluide du système complet
- ✓ Bonus : WebSocket ou Eureka (si applicable)

🎓 Fin du module

Bravo pour votre engagement !

- ✓ REST + Spring Boot
- ✓ Sécurité + Monitoring
- ✓ Swagger + Docker

🎯 Vous êtes prêts pour la soutenance finale !

Merci à tous 🎉