

## MINI APPLICATION E-COMMERCE

### Développement et Déploiement d'une Cloud-Native Application



ENCADRÉ PAR :

- Pr ALLAKI DriSS



RÉALISÉ PAR :

- LAAZIZ Fatima Zahrae
- TAMLALTI Maryam

Année universitaire :  
2022/2023

# INTRODUCTION

Dans le cadre du développement de l'application Mcommerce, nous sommes chargés de mettre en place les services backend et le frontend en utilisant des technologies spécifiques. L'objectif est de créer une solution complète et performante, permettant aux utilisateurs de naviguer et d'effectuer des transactions de manière fluide.

Pour les services backend, nous avons choisi d'utiliser Node.js et le framework Express. Ces technologies offrent une grande flexibilité et une efficacité élevée dans le développement d'applications web. Nous allons développer les différents services nécessaires, en mettant l'accent sur le microservice "produit", qui sera basé sur une base de données MongoDB pour stocker les informations relatives aux produits.

L'application Mcommerce sera également dotée d'un frontend développé sous la forme d'une Single Page Application (SPA) en utilisant la librairie React. Cette approche permettra une expérience utilisateur dynamique et réactive, offrant une navigation fluide et une interactivité optimale.



# Table des matières :

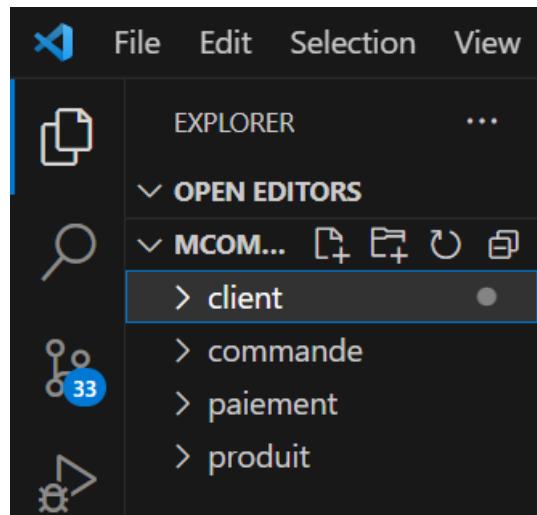
<b>I.</b>	<b>DEVELOPPEMENT DE L'APPLICATION : .....</b>	<b>4</b>
A.	FRONT-END.....	4
B.	BACKEND: .....	14
1.	Le microservice produit : .....	14
2.	Le microservice « commande » : .....	19
3.	Le microservice « paiement » : .....	21
<b>II.</b>	<b>CONTENEURISATION DE L'APPLICATION : .....</b>	<b>21</b>
A.	INTRODUCTION A DOCKER : .....	21
B.	LES DOCKERFILES UTILISES : .....	22
1.	Dockerfile du microservice client .....	22
2.	Dockerfile des micro-services du backend.....	23
<b>III.</b>	<b>ORCHESTRATION DES CONTENEURS EN UTILISANT KUBERNETES : .....</b>	<b>30</b>
<b>IV.</b>	<b>AUTOMATISATION AVEC GITLAB CI: .....</b>	<b>37</b>
A.	LE STAGE « BUILD » : .....	42
B.	LE STAGE « PACKAGE » : .....	43
C.	LE STAGE « DEPLOY » :.....	44
D.	LE STAGE « SAST » : .....	44
E.	LE STAGE « TEST » : .....	45
<b>V.</b>	<b>CREATION D'UN REPERTOIRE GITLAB POUR LE DEPLOIEMENT LOCAL D'UNE APPLICATION KUBERNETES : .....</b>	<b>48</b>
<b>VI.</b>	<b>UTILISATION DE L'OUTIL ARGOCD :.....</b>	<b>51</b>
<b>VII.</b>	<b>CONCLUSION .....</b>	<b>55</b>

# I. Développement de l'application :

## A. Front-end

On a pu développer le front-end avec ReactJS

Pour la structure de notre projet, on va avoir quatre micro-services : produit, commande, paiement, et client



Plaçons-nous dans le dossier client

```
client > src > JS App.js > App
1  import logo from './logo.svg';
2  import './App.css';
3  import Navbar from './component/Navbar';
4  import Home from './component/Home';
5  import Products from './component/Products'
6  import About from './component/About'
7
8  import {Routes, Route, BrowserRouter, Router} from 'react-router-dom';
9  import Product from './component/Product';
10
11 function App() {
12   return (
13     <div>
14
15       <Navbar/>
16       <Routes>
17
18         <Route path="/" element={<Home/>} />
19         <Route path="/products" element={<Products/>} />
20         <Route path="/about" element={<About/>} />
21         <Route path="/products/:id" element={<Product/>} />
22   
```

Il s'agit d'une application React utilisant React Router pour gérer la navigation entre différentes pages.

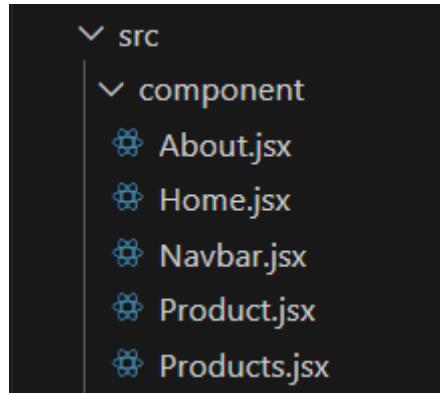
L'application commence par importer les dépendances nécessaires, y compris les composants personnalisés tels que Navbar, Home, Products et About. Ensuite, la fonction App est définie,

qui représente le composant principal de l'application. À l'intérieur de cette fonction, une structure JSX est retournée, qui comprend un composant Navbar et un composant Routes.

Le composant Navbar représente la barre de navigation de l'application et est rendu en haut de chaque page. Le composant Routes est utilisé pour définir les différentes routes de l'application. Chaque route est définie à l'aide du composant Route, spécifiant un chemin d'accès et un élément à rendre lorsque ce chemin d'accès est atteint.

Par exemple, si l'URL correspond au chemin d'accès "/", le composant Home est rendu

Maintenant parlons un peu des composants de base de notre application



Commençons par le composant Navbar

```
1 import React from 'react'
2 import {Link} from "react-router-dom";
3 import { useSelector } from 'react-redux';
4 export default function Navbar() {
5   const state = useSelector((state)=> state.handleCart)
6
7   return (
8     <div>
9       <nav class="navbar navbar-expand-lg bg-body-tertiary">
10      <div class="container">
11        <Link class="navbar-brand fw-bold fs-4" to="#">
12          MyStore
13        </Link>
14        <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarSupportedContent" data-bs-offset="0 0">
15          <span class="navbar-toggler-icon"></span>
16        </button>
17        <div class="collapse navbar-collapse" id="navbarSupportedContent">
18          <ul class="navbar-nav mx-auto mb-2 mb-lg-0">
19            <li class="nav-item">
20              <Link class="nav-link active" aria-current="page" to="/">Home</Link>
21            </li>
22            <li class="nav-item">
23              <Link class="nav-link" to="/products">Products</Link>
24            </li>
25          </ul>
26        </div>
27      </div>
28    </nav>
29  )
30}
```

Ce code représente un composant React fonctionnel appelé Navbar, qui est responsable de l'affichage de la barre de navigation de l'application. Lorsque le composant est rendu, il retourne une structure JSX représentant une barre de navigation avec des liens vers différentes pages de l'application.

Le composant utilise la fonction useSelector de React Redux pour accéder à l'état global de l'application et récupérer la variable handleCart, qui représente le panier de l'utilisateur. Le

nombre d'éléments dans le panier est affiché dans le bouton de panier en utilisant la longueur de cet état.

La barre de navigation est construite à l'aide d'éléments HTML tels que `<nav>`, `<div>`, `<ul>`, `<li>`, et `<a>`. Les liens de navigation sont créés à l'aide de la balise `<Link>` de React Router, qui permet de définir les URL de destination. Les classes CSS sont utilisées pour styliser les différents éléments de la barre de navigation.

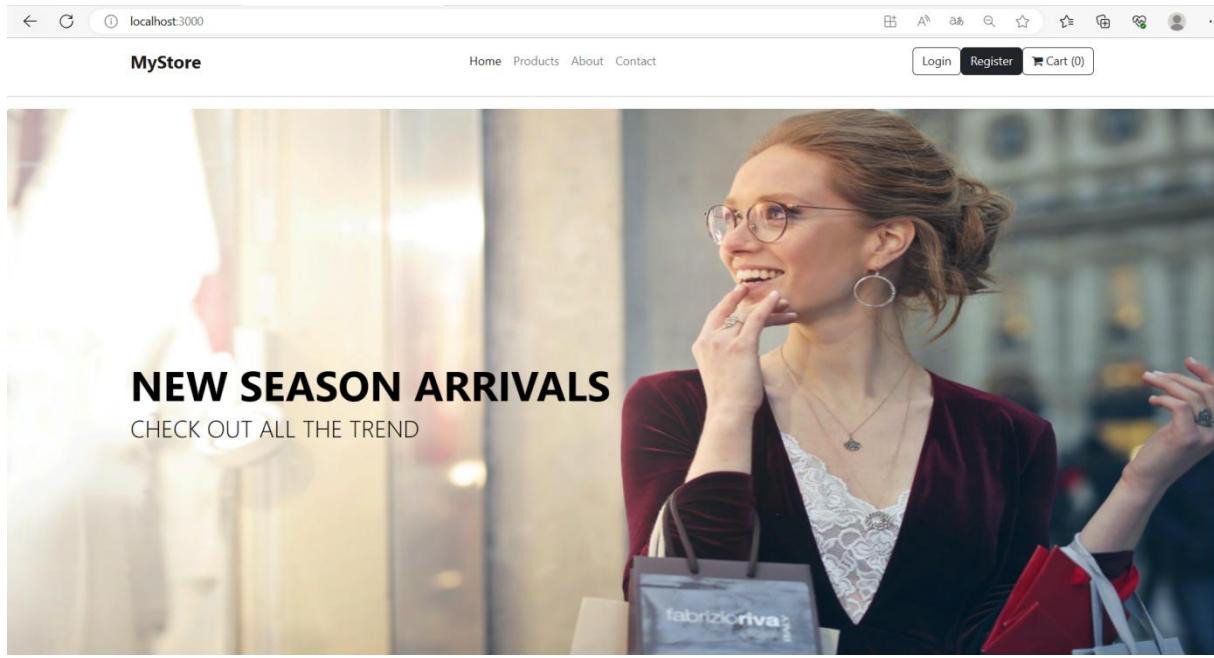
```
client > src > component > Home.jsx > ...
1  import React from "react";
2  import Navbar from "./Navbar";
3  import Product from "./Products";
4
5  export default function Home() {
6    return (
7
8      <div className="hero">
9        <div className="card text-bg-dark text-black border-0">
10          
11          <div className="card-img-overlay d-flex flex-column justify-content-center">
12            <div className="container">
13              <h5 className="card-title display-4 fw-bolder mb-0 ">NEW SEASON ARRIVALS</h5>
14              <p className="card-text lead fs-2">
15                CHECK OUT ALL THE TREND
16              </p>
17            </div>
18
19          </div>
20        </div>
21      </div>
22    );
23  }
24 }
```

Ce composant est responsable de l'affichage du contenu de la page d'accueil de l'application.

Lorsqu'il est rendu, ce composant retourne une structure JSX qui représente une carte avec une image de fond et du texte superposé. La carte est enveloppée dans un conteneur principal avec la classe "hero".

La carte elle-même est composée d'une image (`<img>`) dont la source est définie sur `"/assets/bg2.jpeg"`. Cette image sert de fond pour la carte.

Ensuite, un élément `<div>` avec la classe "card-img-overlay" est utilisé pour superposer du contenu sur l'image de fond. À l'intérieur de cet élément, on trouve un autre élément `<div>` avec la classe "container", qui agit comme un conteneur pour le contenu textuel.



Passant maintenant au component Products qui affiche une liste de produits à partir d'une API locale.

```
1  import React, { useState, useEffect } from "react";
2  import { Link } from "react-router-dom";
3
4
5
6  export default function Products() {
7    const [data, setData] = useState([]);
8    const [filter, setFilter] = useState(data);
9    const [loading, setLoading] = useState(false);
10   let componentMounted = true;
11
12  useEffect(() => {
13    const getProducts = async () => {
14      setLoading(true);
15      const response = await fetch("http://localhost:5000/api/products");
16      if (componentMounted) {
17        setData(await response.clone().json());
18        setFilter(await response.json());
19        setLoading(false);
20        console.log(filter);
21      }
22    };
23    getProducts();
24  }, []);
```

```

    const ShowProducts = () => {
  return (
    <>
      <div className="buttons d-flex justify-content-center">
        <button className="btn btn-outline-dark">All</button>
        <button className="btn btn-outline-dark">Men's clothing</button>
        <button className="btn btn-outline-dark">Women's clothing</button>
        <button className="btn btn-outline-dark">Jewelery</button>
      </div>
      {filter.map((product) => {
        return (
          <div className="col-md-3 mb-4" key={product.id}>
            <div className="card h-100 text-center p-4">
              <img
                src={product.image}
                className="card-img-top"
                alt={product.titre}
                height={250}
              />
              <div className="card-body">
                <h5 className="card-title mb-0">
                  {product.titre && product.titre.substring(0, 12)}
                </h5>
                <p className="card-text lead fw-bold">
                  ${product.prix}
                </p>
                <Link to={`/products/${product._id}`}>
                  Buy now
                </Link>
              </div>
            </div>
          </div>
        )
      )}
    </>
  )
}

```

```

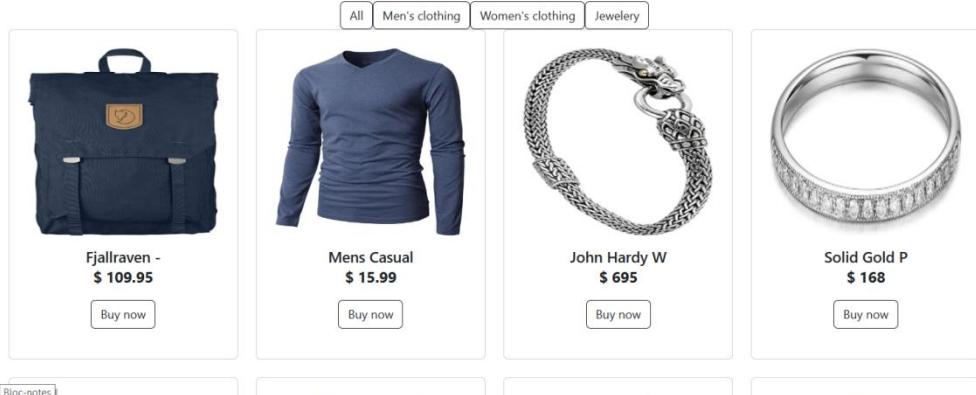
  return (
    <div>
      <div className="container my-5 py-5">
        <div className="row">
          <div className="col-12 mb-5">
            <h1 className="display-6 fw-bolder text-center">Latest Products</h1>
            <hr />
          </div>
        </div>
        <div className="row justify-content-center">
          {loading ? <Loading /> : <ShowProducts />}
        </div>
      </div>
    </div>
  );
}

```

Lorsque le composant est monté, il effectue une requête pour récupérer les données des produits, les stocke dans l'état data, et affiche les produits à l'aide de la fonction ShowProducts.

Les produits sont affichés dans des cartes avec des détails tels que l'image, le titre et le prix. Il existe également des boutons de filtrage pour sélectionner les produits par catégorie. Pendant le chargement des données, une indication de chargement est affichée, et une fois les données chargées, les produits sont affichés

## Latest Products



Le composant Product va nous afficher les détails d'un produit spécifique.

```
useEffect(() => [
  const getProduct = async () => {
    setLoading(true);
    const response = await fetch(`http://localhost:5000/api/products/${id}`);
    setProduct(await response.json());
    setLoading(false);
  };
  getProduct();
], [id]);

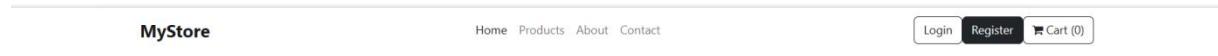
const Loading = () => {
  return (
    <>
      <div className="col-md-6">
        <Skeleton height={400} />
      </div>
      <div className="col-md-6">
        <Skeleton height={50} width={300} />
        <Skeleton height={75} />
        <Skeleton height={25} width={150} />
        <Skeleton height={50} />
        <Skeleton height={150} />
        <Skeleton height={50} width={100} />
        <Skeleton height={50} width={100} style={{marginLeft: 6}} />
      </div>
    </>
  );
};
```

```

const ShowProduct = () => {
  return (
    <>
      <div className="col-md-6">
        <img
          src={product.image}
          alt={product.titre}
          height="400px"
          width="400px"
        />
      </div>
      <div className="col-md-6">
        <h4 className="text-uppercase text-black-50">{product.category}</h4>
        <h1 className="display-5">{product.titre}</h1>
        <p className="lead fw-bolder">
          Rating {product.rating && product.rating.rate}
          <i className="fa fa-star"></i>
        </p>
        <h3 className="display-6 fw-bold my-4">{product.prix}</h3>
        <p className="lead">{product.description}</p>
        <Link
          to="#"
          className="btn btn-outline-dark px-4 py-2"
          onClick={() => handleCommandeSubmit(product)}
        >
          Commander
        </Link>
      </div>
    </>
  )
}

```

Le composant utilise le hook useParams de React Router pour récupérer l'identifiant du produit à partir de l'URL. Il effectue ensuite une requête pour récupérer les informations du produit à partir de l'API locale. Pendant le chargement des données, une interface de chargement est affichée à l'aide de la bibliothèque "react-loading-skeleton". Une fois les données chargées, les détails du produit sont affichés, tels que l'image, la catégorie, le titre, le prix et la description. Il y a également un bouton "Commander" qui va gérer la partie commande.



Fjallraven - Foldsack No. 1  
Backpack, Fits 15 Laptops

Rating ★

**109.95**

Your perfect pack for everyday use and walks in the forest. Stash your laptop (up to 15 inches) in the padded sleeve, your everyday

[Commander](#)

```

const handleCommandeSubmit = async (product) => [
  try {
    const commande = {
      id: 1,
      datePaiement: Date.now(),
      idProduct: product._id,
    };

    const response = await axios.post(
      "http://localhost:5001/api/commandes",
      [commande]
    );

    setOrderId(response.data[0]._id);
    setError(null);
    setCommandeModalisOpen(true);
  } catch (error) {
    console.error("Erreur lors de la création de la commande:", error);
    setOrderId(null);
    setError(error.message);
    setCommandeModalisOpen(true);
  }
];

```

```

  Commander
  </Link>
{orderId && (
  <Modal
    isOpen={commandeModalisOpen}
    onRequestClose={handleCommandeModalClose}
    style={customModalStyles}
  >
    <button onClick={handleCommandeModalClose}>Fermer</button>
    <br />
    <br />

    <p className="text-center text-success fw-bold">
      La Commande est passée avec succès et ID de la commande est:{ " " }
      {orderId}
    </p>
)
}

```

La fonction qui est appelée handleCommandeSubmit est exécutée lorsqu'un utilisateur soumet une commande pour un produit spécifique. La fonction envoie une requête POST à une API pour créer une nouvelle commande avec les informations nécessaires, telles que l'identifiant du produit, la date de paiement et l'identifiant de la commande. Si la requête est réussie, la fonction extrait l'identifiant de la commande de la réponse et le stocke dans l'état orderId. Si une erreur se produit lors de la requête, l'erreur est capturée et enregistrée dans l'état error. Dans les deux cas, la fonction met à jour l'état commandeModalisOpen pour ouvrir une fenêtre modale affichant le résultat de la commande (succès ou erreur). Cela permet à l'utilisateur de visualiser et de prendre des mesures appropriées en fonction de la réponse de la requête.

Le code fourni est un exemple de fichier Dockerfile utilisé pour créer une image Docker pour une application Node.js.

[Fermer](#)

**La Commande est passée avec succès et ID de  
la commande est: 647dee7895e6239cbd4b655a**

Payer ma commande

```

        <div style={customModalStyles.buttonContainer}>
          <button
            className="d-flex justify-content-center btn btn-secondary fixed-bottom p-3 text-center"
            onClick={handlePayment}
          >
            Payer ma commande
          </button>
        </div>
      </Modal>
    )
<Modal
  isOpen={paymentModalisOpen}
  onRequestClose={handleCommandeModalClose}
  style={customModalStyles}
>
  <button onClick={handleCommandeModalClose}>Fermer</button>
  <br />
  <br />

  <p className="text-center text-success fw-bold">
    La commande a été payée avec succès.
    <br />
    ID de la commande : {orderId}
    <br />
    Montant à payer : {product.prix}
  </p>
</Modal>
</div>

const handlePayment = async () => {
  try {
    await axios.post("http://localhost:5002/api/paiements", {
      commandeId: orderId,
      montant: product.prix,
    });

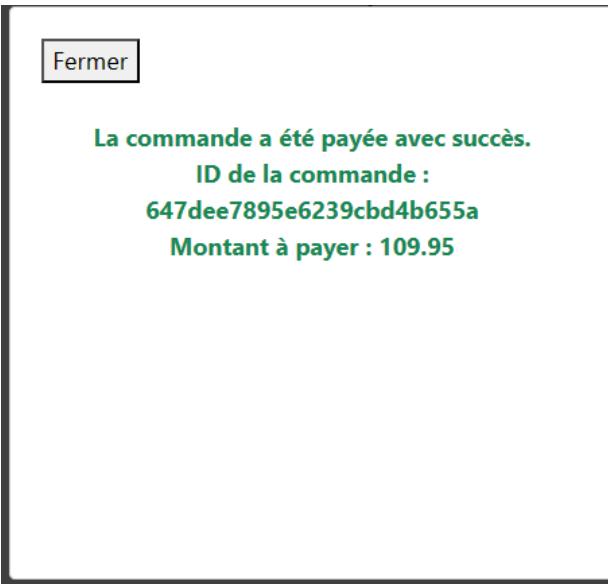
    setPaymentModalisOpen(true);
    console.log("Le paiement a été effectué avec succès");
  } catch (error) {
    console.error("Erreur lors du paiement :", error.message);
  }
};

```

La fonction **handlePayment** est une fonction asynchrone qui effectue une requête HTTP POST vers l'URL "http://localhost:5002/api/paiements" afin de réaliser un paiement. L'objectif de cette fonction est d'envoyer les informations nécessaires pour effectuer le paiement, telles que l'ID de la commande (orderId) et le montant à payer (product.prix).

Dans le bloc try, la fonction utilise axios.post pour envoyer la requête au serveur. Si la requête est réussie, c'est-à-dire qu'elle reçoit une réponse sans erreur, la fonction met à jour l'état paymentModalisOpen pour ouvrir le modal de paiement en définissant sa valeur sur true. Elle affiche également un message dans la console indiquant que le paiement a été effectué avec succès.

Cependant, si une erreur se produit lors de la requête, elle est capturée dans le bloc catch. Dans ce cas, l'erreur est affichée dans la console à l'aide de console.error, avec le message d'erreur fourni.



## B. Backend:

Le backend de notre application comporte trois microservices : produit, commande et paiement.

### 1. Le microservice produit :

- Afin de pouvoir récupérer tous les produits et chaque produit par son ID depuis le frontend, on a défini tout d'abord un schéma de modèle Mongoose pour un objet Item qu'on va utiliser par la suite:

```

produit > models > JS productjs > ...
1  const mongoose = require('mongoose');
2  const Schema = mongoose.Schema;
3
4  const ItemSchema = new Schema({
5      id :{
6          type:Number,
7          required:true
8      },
9      titre:{ 
10         type: String,
11         required:true
12     },
13     description:{ 
14         type: String,
15         required: false
16     },
17     image:{ 
18         type: String,
19         required: false
20     },
21     prix:{ 
22         type: Number,
23         required: true
24     },
25     category: { 
26         type: String,
27         required: false
28     },
29     rating: { 
30         type: Object,
31         required: false
32     }
33 });
34 module.exports = Item = mongoose.model('produit' , ItemSchema);

```

- Ensuite, on a créé une base de données sur MongoDB Atlas. Pour ce faire, on a créé un compte, puis on créé et configuré un nouveau cluster

The screenshot shows the MongoDB Atlas web interface. At the top, there's a navigation bar with 'Atlas', 'INPT', 'Access Manager', 'Billing', and 'All Clusters'. Below the navigation is a secondary navigation bar with 'Project 0', 'Data Services' (which is selected), 'App Services', and 'Charts'. On the left, a sidebar contains sections for 'DEPLOYMENT' (Database, Data Lake PREVIEW), 'SERVICES' (Triggers, Data API, Data Federation, Search), and 'SECURITY' (Quickstart, Backup, Database Access). The main content area is titled 'INPT>PROJECT0' and 'Database Deployments'. It features a search bar and a button for 'LaazizDB' with options to 'Connect', 'View Monitoring', 'Browse Collections', and more. Below this is a section for 'Visualize Your Data' with a note about building dashboards and charts. It displays metrics for reads (R 0), writes (W 0), connections (0), and network traffic (In 0.0 B/s, Out 0.0 B/s). Buttons for 'Dismiss' and 'Explore Charts' are at the bottom of this section.

Node.js

▼

5.5 or later

▼

## 2. Install your driver

Run the following on the command line

```
npm install mongodb
```



[View MongoDB Node.js Driver installation instructions.](#)

## 3. Add your connection string into your application code

View full code sample

```
mongodb+srv://laaziz2023:<password>@laazizdb.e5voy7h.mongodb.net/?retryWrites=true&w=majority
```



Replace `<password>` with the password for the `laaziz2023` user. Ensure any option params are [URL encoded](#).

Après avoir récupéré le lien de la connexion, en raison de sécurité et pour séparer la configuration du code source, on a enregistré cet URL comme variable d'environnement dans un fichier .env

```
produit > ⚡ .env
1   mongoURI = 'mongodb+srv://laaziz2023:<password>@laazizdb.e5voy7h.mongodb.net/?retryWrites=true&w=majority'|
```

Par la suite, on a défini des routes afin de communiquer avec le microservice client :

Deux routes sont définies : une pour obtenir tous les produits (/), et une pour obtenir un produit spécifique en utilisant son ID (/:id). Ces routes utilisent des fonctions asynchrones (`async`) pour permettre l'utilisation d'opérations asynchrones telles que `Item.find()` et `Item.findById()`.

Si une erreur se produit lors de la recherche, un message d'erreur est affiché et une réponse d'erreur est renvoyée. Les résultats de la recherche sont renvoyés sous forme de **JSON** dans la réponse.

```

produit > routes > api > js products.js > ...
1  const express = require('express');
2  const router = express.Router();
3  const Item = require('..../models/product'); // Mettez le chemin approprié vers votre modèle
4
5  // Définissez vos routes ici
6  router.get('/', async (req, res) => {
7    try {
8      const products = await Item.find(); // Utilisation de l'opération de recherche avec une fonction asyn
9      res.json(products);
10    } catch (error) {
11      console.error('Une erreur s\'est produite lors de la recherche des produits:', error);
12      res.status(500).json({ error: 'Erreur lors de la récupération des produits' });
13    }
14  });
15
16  router.get('/:id', async (req, res) => {
17    try {
18      const product = await Item.findById(req.params.id, 'image title prix description');
19      if (!product) {
20        return res.status(404).json({ error: 'Produit non trouvé.' });
21      }
22      res.json(product);
23    } catch (error) {
24      console.error('Une erreur s\'est produite lors de la récupération du produit :', error);
25      res.status(500).json({ error: 'Erreur lors de la récupération du produit.' });
26    }
27  });
28
29  module.exports = router;

```

A ce stade, à l'aide du fichier server.js on configure un serveur **Express** qui se connecte à notre base de données MongoDB en utilisant **Mongoose**. On utilise le middleware **express.json()** pour analyser les données **JSON** dans les requêtes, et un autre middleware qui définit les en-têtes appropriés pour permettre les requêtes provenant de 'http://localhost:3000'. (client) Cela permet d'autoriser les méthodes GET, POST, PUT et DELETE, d'autoriser le type de contenu 'Content-Type' et d'activer les informations d'authentification.

Puis, on définit les routes pour les fonctionnalités liées aux produits. Le serveur écoute sur le port 5000 et affiche un message de confirmation lorsque le serveur démarre avec succès.

```

produit > js server.js > ...
  5  const items = require('./routes/api/products');
  6
  7  const app = express();
  8
  9  app.use(express.json());
 10
 11  const db = process.env.mongoURI;
 12
 13  mongoose
 14    .connect(db)
 15    .then(() => console.log('MongoDB Connected ...'))
 16    .catch((err) => console.log(err));
 17
 18 // Add the CORS headers middleware
 19 app.use((req, res, next) => {
 20   res.setHeader('Access-Control-Allow-Origin', 'http://localhost:3000');
 21   res.setHeader('Access-Control-Allow-Methods', 'GET, POST, PUT, DELETE');
 22   res.setHeader('Access-Control-Allow-Headers', 'Content-Type, Authorization');
 23   res.setHeader('Access-Control-Allow-Credentials', 'true');
 24   next();
 25 });
 26
 27 app.use('/api/products', items);
 28
 29 const port = process.env.PORT || 5000;
 30
 31 app.listen(port, () => console.log(`Server started on port ${port}`));
 32

```

Ensuite, on a rempli la base de données à partir d'une API "fakestore" en utilisant la bibliothèque **Axios** pour effectuer une requête GET vers l'API et récupérer les produits.

```

produit > js data.js > i importData > b batchPromises > b batch.map() callback > b newItem
  10
 11  async function importData() {
 12    try {
 13      // Effectuer une requête GET vers l'API pour obtenir les produits
 14      const response = await axios.get('https://fakestoreapi.com/products');
 15      const products = response.data;
 16
 17      const batchSize = 100; // Nombre de documents à insérer par lot
 18      const delayBetweenBatches = 1000; // Décalage en millisecondes entre chaque lot
 19
 20      // Parcourir les produits et les insérer dans la base de données par lots
 21      for (let i = 0; i < products.length; i += batchSize) {
 22        const batch = products.slice(i, i + batchSize);
 23
 24        // Créer un tableau de promesses pour chaque document à insérer dans le lot
 25        const batchPromises = batch.map(async (product) => {
 26          const { id, title, description, image, price, category, rating } = product;
 27
 28          // Créer un nouvel objet Item en utilisant les données du produit
 29          const newItem = new Item({
 30            id,
 31            titre: title,
 32            description,
 33            image,
 34            prix: price,
 35            category,
 36            rating
 37          });
 38
 39          // Sauvegarder le nouvel objet Item dans la base de données
 40          return newItem.save();
 41        });
 42
 43        // Attendre que toutes les promesses du lot actuel soient résolues avant de passer au lot suivant
 44        await Promise.all(batchPromises);
 45

```

Après avoir exécuté la commande node data.js, on remarque que les données sont bien enregistrées dans notre base de données.

The screenshot shows the MongoDB Compass interface with the following details:

- Storage Size:** 44KB
- Logical Data Size:** 22.18KB
- Total Documents:** 40
- Indexes Total Size:** 36KB
- Find:** Selected tab
- Indexes:** Tab
- Schema Anti-Patterns:** Tab (0)
- Aggregation:** Tab
- Search Indexes:** Tab
- Charts:** Tab (●)
- INSERT DOCUMENT:** Button
- Filter:** Type a query: { field: 'value' }
- Apply:** Button
- More Options:** More Options ▶
- Document Preview:**

```
_id: ObjectId('6479ef8a8854c03ce9952205')
id: 1
titre: "Fjallraven - Foldsack No. 1 Backpack, Fits 15 Laptops"
description: "Your perfect pack for everyday use and walks in the forest. Stash your..."
image: "https://fakestoreapi.com/img/81fPKd-2AYL.AC SL1500.jpg"
```
- Pagination:** PREVIOUS, 1-20 of many results, NEXT, Chat icon

## 2. Le microservice « commande » :

Avec la même approche qu'on a suivie au niveau du microservice « produit », on définit un schéma de modèle Mongoose pour un objet CommandeSchema puis on connecte notre microservice avec notre base de données en utilisant le même URL.

```
commande > models > JS commande.js > ...
1 const mongoose = require('mongoose');
2 const Schema = mongoose.Schema;
3
4 const CommandeSchema = new Schema({
5   id :{
6     type:String,
7   },
8 },
9
10  idProduct : {
11    type: String,
12    required: true,
13  },
14
15  datePaiement:{
16    type: Date,
17    default: Date.now
18  },
19
20 });
21
22
23 module.exports = Commande = mongoose.model('commande' , CommandeSchema);
```

De plus, on définit les endpoints (**POST / :** | **GET / :** | **GET /:id** | **PUT /:id/paiement** | **DELETE /:id**) pour la gestion des commandes.

- Concernant, l'endpoint **POST / :** (utilisé par le microservice client), il crée une nouvelle commande en utilisant les données fournies dans le corps de la requête. Les commandes doivent être envoyées sous forme d'un tableau dans le corps de la

requête. Les données de chaque commande comprennent un identifiant (**id**) et un identifiant de produit (**idProduct**). Les commandes sont enregistrées dans la base de données à l'aide de la méthode **insertMany** de Mongoose. La réponse renvoie les commandes créées.

```
commande > routes > api > JS commandes.js > ...
1 const express = require('express');
2 const router = express.Router();
3 const Commande = require('../models/commande');
4
5 // Créer une commande
6 router.post('/', async (req, res) => {
7   try {
8     const commandes = req.body; // Obtenir la liste des commandes du corps de la requête
9
10    if (!Array.isArray(commandes)) {
11      return res.status(400).json({ error: 'Le corps de la requête doit contenir un tableau de commandes' });
12    }
13
14    const nouvellesCommandes = commandes.map(({ id, idProduct }) => ({ id, idProduct }));
15    const commandesCrees = await Commande.insertMany(nouvellesCommandes);
16
17    res.json(commandesCrees);
18  } catch (error) {
19    console.error('Erreur lors de la création des commandes:', error);
20    res.status(500).json({ error: 'Erreur lors de la création des commandes' });
21  }
22});
23
```

- ⊕ Et pour l'endpoint **PUT /:id/paiement** (utilisé par le microservice paiement): Il met à jour le montant du paiement d'une commande spécifique en utilisant son identifiant (**id**). Le nouveau montant est extrait du corps de la requête. La commande est recherchée dans la base de données, puis son champ de paiement est mis à jour avec le nouveau montant. Les modifications sont enregistrées dans la base de données.

```
51 // Mettre à jour le paiement d'une commande
52 router.put('/:id/paiement', async (req, res) => {
53   try {
54     const { id } = req.params;
55     const { montant } = req.body;
56
57     // Mettez en œuvre votre logique de mise à jour de l'état de la commande ici
58     // par exemple, vous pouvez rechercher la commande dans la base de données par son ID
59     // puis mettre à jour le champ de paiement de la commande avec le montant spécifié
60
61     const commande = await Commande.findById(id);
62
63     if (!commande) {
64       return res.status(404).json({ error: 'La commande n\'existe pas' });
65     }
66
67     commande.paiement = montant; // Mettez à jour le champ de paiement de la commande
68     await commande.save(); // Enregistrez les modifications dans la base de données
69
70     res.json({ message: 'Le paiement de la commande a été mis à jour avec succès' });
71   } catch (error) {
72     console.error('Erreur lors de la mise à jour du paiement de la commande:', error);
73     res.status(500).json({ error: 'Erreur lors de la mise à jour du paiement de la commande' });
74   }
75});
```

### 3. Le microservice « paiement » :

Ce microservice doit communiquer avec le frontend en utilisant le microservice commande pour marquer cette dernière comme payée. Pour cette raison on a configuré l'endpoint POST/ : qui effectue un paiement pour une commande en utilisant les données fournies dans le corps de la requête. Les données attendues sont l'identifiant de la commande (commandeId) et le montant du paiement (montant).

Le code envoie une requête HTTP de type PUT à l'aide de la bibliothèque axios au microservice commande à l'URL spécifiée avec le montant du paiement dans le corps de la requête.

- Si la réponse du microservice commande a un statut de 200, cela signifie que la mise à jour de la commande a réussi, ce qui indique que le paiement a été effectué avec succès. Une réponse JSON est renvoyée avec le statut "success" défini sur true et un message de succès.

- Si la réponse du microservice commande a un statut différent de 200, cela indique que la mise à jour de la commande a échoué. Une réponse JSON est renvoyée avec le statut "success" défini sur false et un message indiquant que la mise à jour de la commande a échoué.

```
paiement > routes > api > js paiements.js > router.get('/') callback
1  const express = require('express');
2  const axios = require('axios');
3  const router = express.Router();
4
5  const Paiement = require('../models/paiement');
6
7
8  // Effectuer un paiement pour une commande
9  router.post('/', async (req, res) => {
10    try {
11      const { commandeId, montant } = req.body; // Obtenir l'ID de la commande et le montant du corps de la requête
12
13      // Envoyer une requête au microservice commande pour marquer la commande comme payée
14      const response = await axios.put(`http://localhost:5001/api/commandes/${commandeId}/paiement`, {
15        montant,
16      });
17
18      if (response.status === 200) {
19        // Le paiement a réussi
20        res.json({ success: true, message: 'Le paiement a été effectué avec succès' });
21      } else {
22        // La mise à jour de la commande a échoué
23        res.json({ success: false, message: 'La mise à jour de la commande a échoué' });
24      }
25    } catch (error) {
26      console.error('Erreur lors du paiement de la commande:', error);
27      res.status(500).json({ error: 'Erreur lors du paiement de la commande' });
28    }
29  });
30
```

## II. Conteneurisation de l'application :

### A. Introduction à docker :

Docker est une plateforme open-source permettant de créer, déployer et exécuter des applications dans des conteneurs légers et portables. Les conteneurs sont des environnements isolés qui encapsulent toutes les dépendances nécessaires à l'exécution d'une application, y

compris le système d'exploitation, les bibliothèques et les fichiers de configuration. Ils offrent une méthode efficace et reproductible pour distribuer des applications, en garantissant que celles-ci fonctionnent de la même manière sur différents systèmes.

L'une des principales caractéristiques de Docker est sa capacité à virtualiser l'environnement d'exécution de l'application, sans avoir besoin de virtualiser tout le système d'exploitation. Cela signifie que les conteneurs sont légers, démarrage rapide et consomment moins de ressources système par rapport à une machine virtuelle traditionnelle.

Docker utilise des fichiers appelés Dockerfiles pour décrire les étapes nécessaires à la création d'une image Docker. Une image Docker est une version immuable et auto-suffisante d'une application, prête à être exécutée dans un conteneur. Les images Docker peuvent être partagées et distribuées via des registres Docker, ce qui facilite le déploiement des applications sur différents environnements.

L'utilisation de Docker présente de nombreux avantages. Elle simplifie le déploiement des applications, permet une gestion plus efficace des ressources et facilite la collaboration entre les développeurs et les équipes d'exploitation. En encapsulant les applications dans des conteneurs, Docker favorise également la portabilité et la scalabilité des applications, permettant aux développeurs de créer et de déployer des applications de manière cohérente, quel que soit l'environnement d'exécution.

## B. Les Dockerfiles utilisés :

### 1. Dockerfile du microservice client

```
client > 📄 Dockerfile > ...
1  FROM node:16-alpine as builder
2  WORKDIR /client
3  COPY ["package.json", "package-lock.json", "./"]
4  RUN npm install
5  COPY . .
6  RUN npm run build
7
8
9  FROM node:19.9.0-alpine
10 ENV NODE_ENV=production
11 WORKDIR /app
12 COPY ["package.json", "package-lock.json", "./"]
13 COPY . .
14 EXPOSE 5000
15 CMD ["node", "start"]
```

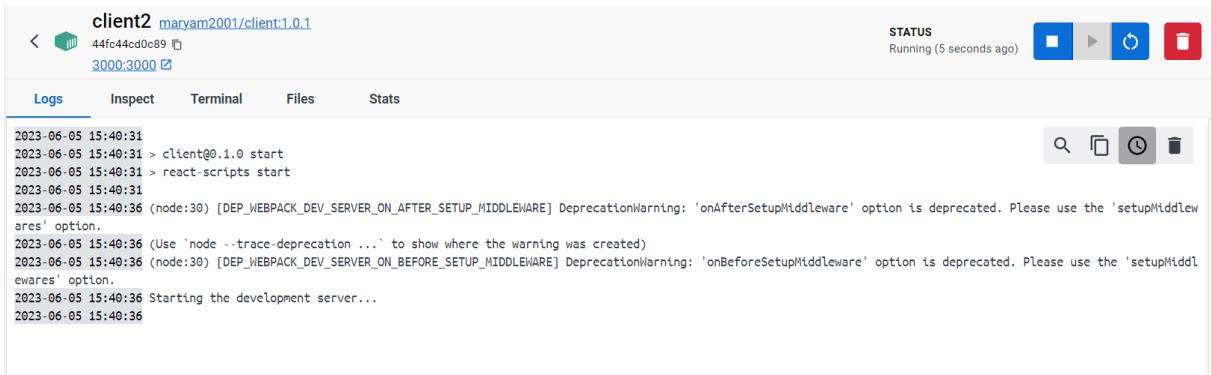
Une fois que le fichier dockerfile est écrit on va faire le build de l'image docker en utilisant la commande suivante :

```
PS C:\Users\Lenovo\Desktop\mcommerce> cd client
PS C:\Users\Lenovo\Desktop\mcommerce\client> docker build -t maryam2001/client:1.0.1 .
```

Ensuite on fait un run en attribuant au conteneur le nom client2 avec le port 3000

```
PS C:\Users\Lenovo\Desktop\mcommerce> docker run -d --name client2 -p 3000:3000 maryam2001/client:1.0.1
```

On accède ensuite à docker desktop pour visualiser le conteneur créé



On remarque que tout marche bien et l'application a pu s'exécuter.

## 2. Dockerfile des micro-services du backend

### a) Le micro-service Product :

Pour les quatre micro-services restants on va opter pour le même dockerfile qui est le suivant :

```
produit > Dockerfile > ...
1  FROM node:19.9.0-alpine
2  ENV NODE_ENV=production
3  WORKDIR /app
4  COPY ["package.json", "package-lock.json", "./"]
5  RUN npm install --${NODE_ENV}
6  COPY . .
7  EXPOSE 5000
8  COPY .env /app/.env
9  CMD ["node", "server.js"]
```

Le premier élément du fichier est la ligne `FROM node:19.9.0-alpine`, qui spécifie l'image de base à utiliser, en l'occurrence `node:19.9.0-alpine`. Cette image est basée sur Alpine Linux, une distribution légère de Linux, et contient l'environnement Node.js.

Ensuite, la ligne `ENV NODE_ENV=production` définit la variable d'environnement `NODE_ENV` sur "production". Cela indique à l'application Node.js qu'elle doit s'exécuter en mode production.

Le répertoire de travail est défini sur `/app` avec la ligne `WORKDIR /app`. Cela signifie que tous les commandes et fichiers seront exécutés et copiés dans ce répertoire.

Les commandes COPY sont utilisées pour copier les fichiers package.json, package-lock.json et .env (contenant les variables d'environnement) dans le répertoire de travail /app de l'image Docker.

Ensuite, la commande RUN npm install --\${NODE\_ENV} est exécutée pour installer les dépendances de l'application. L'option --\${NODE\_ENV} est utilisée pour installer les dépendances de production spécifiées dans le fichier package.json lorsque la variable d'environnement NODE\_ENV est définie sur "production".

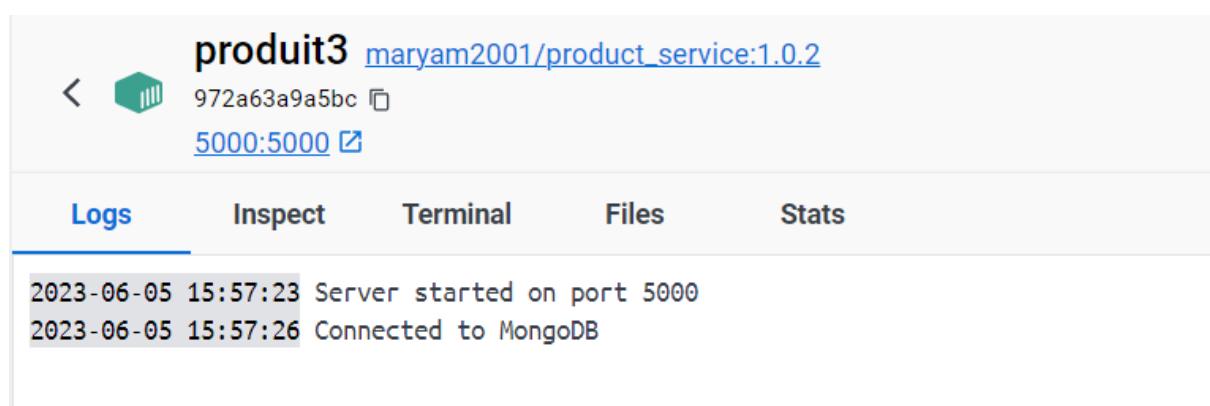
Enfin, la commande EXPOSE 5000 indique que le conteneur Docker écoutera sur le port 5000.

La dernière ligne CMD ["node", "server.js"] spécifie la commande à exécuter lorsque le conteneur est démarré. Dans ce cas, il exécute le fichier server.js à l'aide de Node.js pour lancer l'application.

Ensuite, On fait le build et le run afin de créer le conteneur.

```
Terminer le programme de commandes (O/N) ? o
PS C:\Users\Lenovo\Desktop\mcommerce\produit> docker build -t maryam2001/product_service:1.0.2
```

```
Terminer le programme de commandes (O/N) ? o
PS C:\Users\Lenovo\Desktop\mcommerce\produit> docker run -d --name produit3 --network product -p 5000:5000 maryam2001/product_service:1.0.2
```



On remarque que le micro-service product marche bien et on a pu se connecter à la base de données.

De même pour les micro-services commande et paiement, on passe par les mêmes étapes

b) *Le micro-service commande :*

```

commande > Dockerfile > ...
1  FROM node:19.9.0-alpine
2  ENV NODE_ENV=production
3  WORKDIR /app
4  COPY ["package.json", "package-lock.json", "./"]
5  RUN npm install --${NODE_ENV}
6  COPY . .
7  EXPOSE 5001
8  COPY .env /app/.env
9  CMD ["node", "server.js"]

```

○ PS C:\Users\Lenovo\Desktop\mcommerce\commande> docker build -t maryam2001/commande\_service:1.0.1.

● PS C:\Users\Lenovo\Desktop\mcommerce\produit> cd ../commande  
○ PS C:\Users\Lenovo\Desktop\mcommerce\commande> docker run -d --name commande\_service -p 5001:5001 maryam2001/commande\_service:1.0.1



### c) Le micro-service paiement :

```

# Use a lightweight Node.js image
FROM node:19.9.0-alpine

# Set the working directory inside the container
WORKDIR /app

# Copy the package.json and package-lock.json files to the working directory
COPY package*.json .

# Install dependencies
RUN npm install --only=production

# Copy the rest of the application code
COPY . .

# Expose the port on which the microservice will listen
EXPOSE 5002

# Start the microservice
CMD [ "node", "server.js" ]

```

PS C:\Users\Lenovo\Desktop\mcommerce\commande> cd ../paiement

○ PS C:\Users\Lenovo\Desktop\mcommerce\paiement> docker build -t maryam2001/paiement:1.0.0.

```

PS C:\Users\Lenovo\Desktop\mcommerce\commande> cd ..\paiement
PS C:\Users\Lenovo\Desktop\mcommerce\paiement> docker run -d --name paiement2 -p 3000:3000 maryam2001/paiement:1.0.1

```

**paiement2** [maryam2001/paiement:1.0.0](#)

70a074d3625a 5002:5002

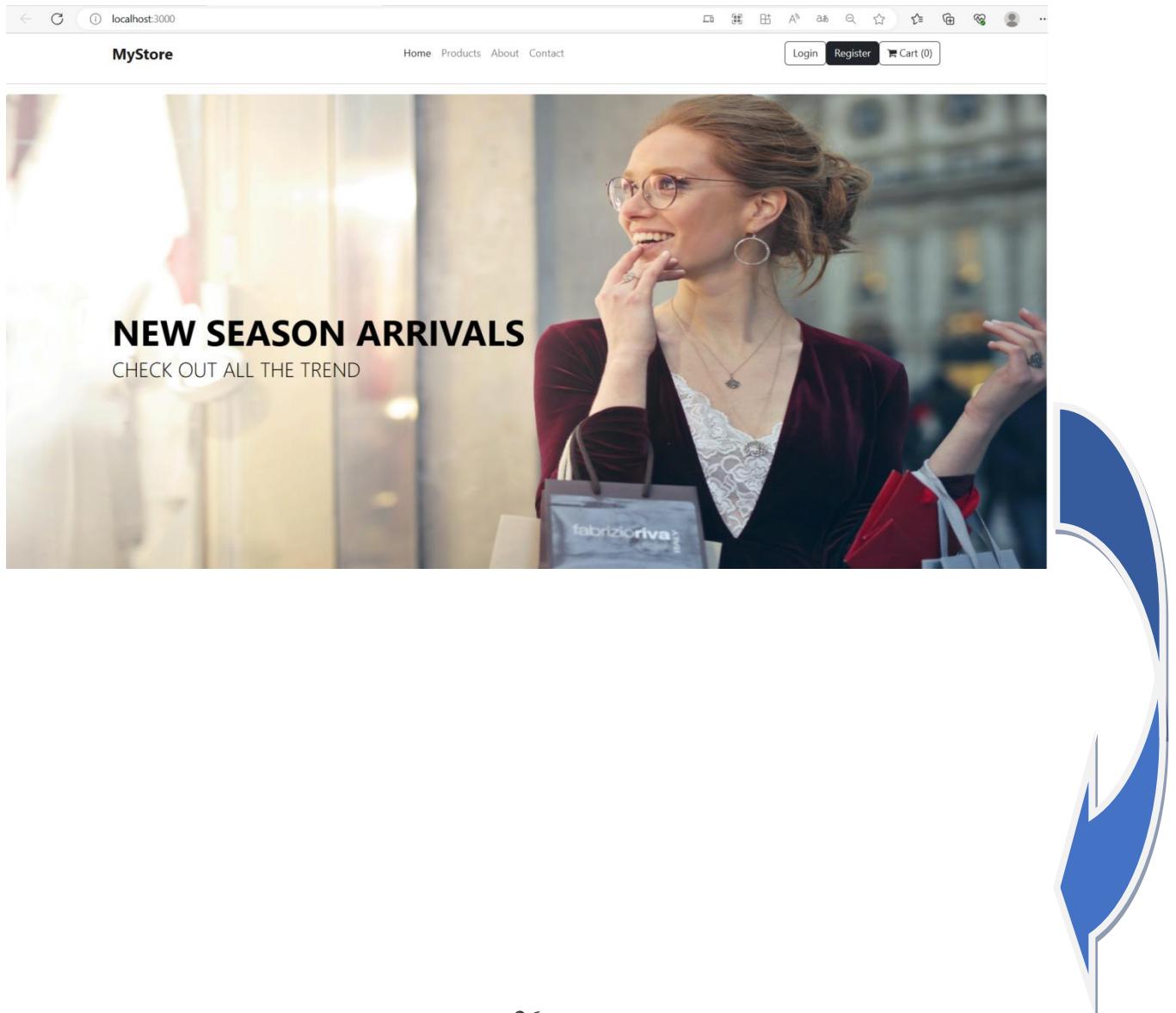
STATUS  
Running (0 seconds ago)

Logs Inspect Terminal Files Stats

2023-06-05 16:16:49 Server started on port 5002

client2	44fc44cd0c89	maryam2001/client:1.0.1	Running	3000:3000	4 hours ago	⋮
paiement2	70a074d3625a	maryam2001/paiement:1.0.0	Running	5002:5002	3 hours ago	⋮
commande_service	a52fcf8b701e	maryam2001/commande_service:1.0.1	Running	5001:5001	4 hours ago	⋮
produit3	972a63a9a5bc	maryam2001/product_service:1.0.2	Running	5000:5000	4 hours ago	⋮

Afin de visualiser le résultat, on va taper localhost :3000 pour voir est ce que le frontend marche bien et puis on vérifie est ce que les liens affichés vont nous amenés vers la bonne destination



localhost:3000/products

## Latest Products

All Men's clothing Women's clothing Jewelry

Fjallraven - \$ 109.95

Buy now

Mens Casual \$ 15.99

Buy now

John Hardy W \$ 695

Buy now

Solid Gold P \$ 168

Buy now

Diamond ring

Rose gold rings

Men's raglan shirt

Khaki jacket

localhost:3000/products/647a06f2355694040e9065ab

MyStore Home Products About Contact Login Register Cart (0)

John Hardy Women's Legends  
Naga Gold & Silver Dragon  
Station Chain Bracelet

Rating ★

**695**

From our Legends Collection, the Naga was inspired by the mythical water dragon that protects the ocean's pearl. Wear facing inward to be bestowed with love and abundance, or outward for protection.

Commander

[Fermer](#)

**La Commande est passée avec succès et ID de la commande est: 647dfdadd6fdff1e0c6d5580**

[Payer ma commande](#)

[Commander](#)

[Fermer](#)

**La commande a été payée avec succès.**

**ID de la commande :**

**647dee7895e6239cbd4b655a**

**Montant à payer : 109.95**

Sans oublier de publier les images crée dans notre docker hub afin de les utilisées par la suite

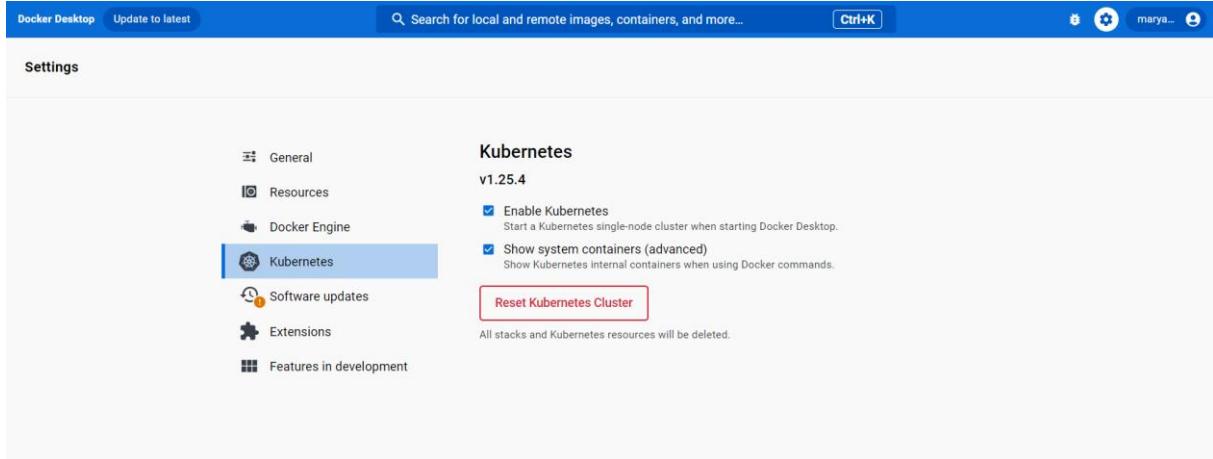
```
The push refers to repository [docker.io/maryam2001/commende_ser]
● 0a55e7f47bf5: Pushed
6ff7ea353927: Pushed
e22285e01b0f: Layer already exists
eaf06335f255: Layer already exists
ca875116cc25: Layer already exists
e394b5a78515: Layer already exists
9b2d5af7f709: Layer already exists
b240fe81adaa: Layer already exists
bb01bd7e32b5: Layer already exists
```

The screenshot shows the Docker Hub interface. At the top, there's a search bar with "Search Docker Hub" and a navigation bar with "Explore", "Repositories", "Organizations", "Help", and a dropdown menu. Below that, a user profile "maryam2001" is selected in a dropdown, along with a search bar for "Search by repository name" and a "Create repository" button. The main area displays four public repositories:

- maryam2001 / commande\_service**  
Contains: Image | Last pushed: 3 hours ago  
Inactive | 0 stars | 1 download | Public
- maryam2001 / product\_service**  
Contains: Image | Last pushed: a day ago  
Inactive | 0 stars | 1 download | Public
- maryam2001 / paiement**  
Contains: Image | Last pushed: a day ago  
Inactive | 0 stars | 0 download | Public
- maryam2001 / client**  
Contains: Image | Last pushed: a day ago  
Inactive | 0 stars | 0 download | Public

### III. Orchestration des conteneurs en utilisant Kubernetes :

La première chose à faire est qu'on va aller activer kubernetes dans docker desktop comme le montre la figure ci-dessous :



Ensuite on va créer un dossier kubernetes contenant quatre fichiers YAML chacun dédié à un micro-service.

Commençons par le fichier yaml du micro-service produit :

```
kubernetes > ! product.yaml
 1  # product.yaml
 2
 3  apiVersion: v1
 4  kind: Service
 5  metadata:
 6  |   name: product
 7  spec:
 8  |   selector:
 9  |     app: product
10  |   ports:
11  |     - port: 5000
12
13
14  ---
15
16  apiVersion: apps/v1
17  kind: Deployment
18  metadata:
19  |   name: product-deployment
20  spec:
21  |   replicas: 2
22  |   selector:
23  |     matchLabels:
24  |       app: product
25  |   template:
26  |     metadata:
27  |       labels:
28  |         app: product
29  |   spec:
30  |     containers:
31  |       - name: produit
32  |         image: maryam2001/product_service:1.0.2
33  |       ports:
34  |         - containerPort: 5000
35  |           name: product-port
36
```

Notre fichier product.yaml est utilisé pour déployer une application appelée "product" dans un environnement Kubernetes. Il définit un service et un déploiement qui permettent de gérer et d'exécuter cette application de manière scalable et fiable.

Le bloc de code du service spécifie les détails du service, notamment son nom "product" et le port 5000 sur lequel il sera exposé. Le sélecteur indique que ce service dirigera le trafic vers les pods ayant l'étiquette "app: product". Cela permet de faire correspondre les requêtes aux instances spécifiques de l'application.

Le bloc de code du déploiement définit les spécifications pour le déploiement de l'application. Il crée deux répliques (pods) de l'application pour assurer une haute disponibilité. Les pods sont identifiés par l'étiquette "app: product" définie dans le sélecteur du déploiement. Chaque pod exécute un conteneur nommé "produit" en utilisant une image spécifique "maryam2001/product\_service:1.0.2". Le port 5000 est exposé à l'intérieur du conteneur pour permettre la communication avec l'application.

Une fois fait on tape la commande suivante pour exécuter le fichier en question :

```
PS C:\Users\Lenovo\Desktop\mcommerce\kubernetes> kubectl apply -f product.yaml
service/product created
deployment.apps/product-deployment unchanged
PS C:\Users\Lenovo\Desktop\mcommerce\kubernetes>
```

On accède à docker desktop pour voir

	<a href="#">k8s POD_product-deployment-5797887f97-vm</a> 72fe8a19db63	registry.k8s.io/pause:3.8	Running	20 minutes ago	
	<a href="#">k8s POD_product-deployment-5797887f97-v2l</a> 7f39b5c0888a	registry.k8s.io/pause:3.8	Running	20 minutes ago	
	<a href="#">k8s produit_product-deployment-5797887f97-</a> ae3286c471ee	sha256:f6d307cdbfa6582b9db865d92a07680a	Running	13 minutes ago	
	<a href="#">k8s produit_product-deployment-5797887f97-</a> 21e24a3a5459	sha256:f6d307cdbfa6582b9db865d92a07680a	Running	13 minutes ago	

On remarque la création de deux pods (c'est ça ce qu'on a mentionné au niveau de réplicas)

On accède aux déploiements créés :

The screenshot shows the Docker Desktop interface with two deployment containers listed:

- k8s\_produit\_product-deployment-5797887f97**: sha256:f6d307cdbfa6582b9db865d92a07680a2383200a0c46c07f6cb30fb81be4844e. Status: Running (0 seconds ago). Logs: 2023-06-05 21:37:20 Server started on port 5000, 2023-06-05 21:37:22 Connected to MongoDB.
- v2hfl\_default\_5713b1ab-9ad5-4778-aefa-9c5af070b3ff\_0**: sha256:f6d307cdbfa6582b9db865d92a07680a2383200a0c46c07f6cb30fb81be4844e. Status: Running (9 seconds ago). Logs: 2023-06-05 21:37:57 Server started on port 5000, 2023-06-05 21:37:59 Connected to MongoDB.

On remarque que les deux fonctionnent bien et le serveur se lance.

On fait la même chose pour les autres micro-services en modifiant les noms des services et des déploiements ainsi que l'image docker à importer.

```

1  # commande.yaml
2
3  apiVersion: v1
4  kind: Service
5  metadata:
6    name: commande
7  spec:
8    selector:
9      app: commande
10   ports:
11     - port: 5001
12
13
14 ---
15
16  apiVersion: apps/v1
17  kind: Deployment
18  metadata:
19    name: commande-deployment
20  spec:
21    replicas: 2
22    selector:
23      matchLabels:
24        app: commande
25    template:
26      metadata:
27        labels:
28          app: commande
29    spec:
30      containers:
31        - name: commande
32          image: maryam2001/commande_service:1.0.1
33          ports:
34            - containerPort: 5001
35            name: commande-port
36

```

```

PS C:\Users\Lenovo\Desktop\mcommerce\kubernetes> kubectl apply -f commande.yaml
service/commande created
deployment.apps/commande-deployment created
PS C:\Users\Lenovo\Desktop\mcommerce\kubernetes>

```

The screenshot shows the Kubernetes Dashboard interface. The top section displays a table of pods:

	Name	Image	Status	Last Seen	Actions
□	k8s_POD_commande-deployment-76c9f6c7f5-1b7f00116608	registry.k8s.io/pause:3.8	Running	4 minutes ago	⋮ ⚙️
□	k8s_POD_commande-deployment-76c9f6c7f5-7bf132af372	registry.k8s.io/pause:3.8	Running	4 minutes ago	⋮ ⚙️
□	k8s_commande_commande-deployment-76c9f6c7f5-07269025293a	sha256:145605e025b46659162e2fba3fd9f7cf1	Running	4 minutes ago	⋮ ⚙️
□	k8s_commande_commande-deployment-76c9f6c7f5-b4d6bcd9ca	sha256:145605e025b46659162e2fba3fd9f7cf1	Running	4 minutes ago	⋮ ⚙️

The bottom section shows the logs for one of the pods:

```

k8s_commande_commande-deployment-76c9f6c7f5-b6hpv_default_3fec1174-36ea-4030-a887-6ec277ca8774_0
07269025293a

```

STATUS: Running (3 minutes ago)

Log tab is selected. Log output:

```

2023-06-05 21:46:29 Server started on port 5001
2023-06-05 21:46:31 Connected to MongoDB

```

k8s\_commande\_command... [sha256:145605e025b46659162e2fba3fd9f7cf1d9bbd3143af29637ecc8e7d3f77700e](#)

deployment-76c9f6c7f5-  
vml22\_default\_2579ccbb-  
af48-4857-8e82-  
ad5d67f56d56\_0  
b4d6b0db9ca

STATUS  
Running (2 minutes ago)

Logs Inspect Terminal Files Stats

2023-06-05 21:46:29 Server started on port 5001  
2023-06-05 21:46:31 Connected to MongoDB

```

1 # paiement.yaml
2
3 apiVersion: v1
4 kind: Service
5 metadata:
6   name: paiement
7 spec:
8   selector:
9     app: paiement
10  ports:
11    - port: 5002
12
13
14 ---
15
16 apiVersion: apps/v1
17 kind: Deployment
18 metadata:
19   name: paiement-deployment
20 spec:
21   replicas: 2
22   selector:
23     matchLabels:
24       app: paiement
25   template:
26     metadata:
27       labels:
28         app: paiement
29   spec:
30     containers:
31       - name: paiement
32         image: maryam2001/paiement:1.0.0
33         ports:
34           - containerPort: 5002
35             name: paiement-port

```

PS C:\Users\Lenovo\Desktop\mcommerce\kubernetes> kubectl apply -f paiement.yaml

- service/paiement created
- deployment.apps/paiement-deployment created

○ PS C:\Users\Lenovo\Desktop\mcommerce\kubernetes>

The screenshot shows two separate pod details pages from a Kubernetes interface.

**Pod 1 Details:**

- Name:** k8s\_POD\_paiement-deployment-65c8c864d6-s
- Image:** registry.k8s.io/pause:3.8
- Status:** Running
- Created:** 4 minutes ago

**Pod 2 Details:**

- Name:** k8s\_POD\_paiement-deployment-65c8c864d6-v
- Image:** registry.k8s.io/pause:3.8
- Status:** Running
- Created:** 4 minutes ago

**Logs Tab (for Pod 1):**

2023-06-05 21:58:40 Server started on port 5002

**Logs Tab (for Pod 2):**

2023-06-05 21:58:40 Server started on port 5002

```

kubernetes > ! client.yaml
1  # frontend.yaml
2
3  apiVersion: v1
4  kind: Service
5  metadata:
6    name: frontend
7  spec:
8    selector:
9      app: frontend
10   ports:
11     - port: 3000
12   type: LoadBalancer
13
14 ---
15
16  apiVersion: apps/v1
17  kind: Deployment
18  metadata:
19    name: frontend-deployment
20  spec:
21    replicas: 2
22    selector:
23      matchLabels:
24        app: frontend
25    template:
26      metadata:
27        labels:
28          app: frontend
29    spec:
30      containers:
31        - name: frontend
32          image: maryam2001/client:1.0.1
33          ports:
34            - containerPort: 3000
35

```

```

deployment.apps/frontend-deployment created
● PS C:\Users\Lenovo\Desktop\mcommerce\kubernetes> kubectl apply -f client.yaml
service/frontend created
deployment.apps/foreground-deployment created
○ PS C:\Users\Lenovo\Desktop\mcommerce\kubernetes>

```

□	<a href="#">k8s_POD_frontend-deployment-6968d9b847-x1</a> 843c42bd30c7	<a href="#">registry.k8s.io/pause:3.8</a>	Running	2 minutes ago		
□	<a href="#">k8s_POD_frontend-deployment-6968d9b847-4i</a> 0ccdf9d3852b	<a href="#">registry.k8s.io/pause:3.8</a>	Running	2 minutes ago		
□	<a href="#">k8s_frontend_frontend-deployment-6968d9b84</a> cd1e0c834959	<a href="#">sha256:21f19f56bb7b428cc5228aac5cab3661</a>	Running	2 minutes ago		
□	<a href="#">k8s_frontend_frontend-deployment-6968d9b84</a> 35d999233a51	<a href="#">sha256:21f19f56bb7b428cc5228aac5cab3661</a>	Running	2 minutes ago		

k8s\_frontend\_frontend-deployment-6968d9b847-  
xhx7t\_default\_7e296a63-  
60eb-4a6b-9d9d-  
68b6498c10b2\_0  
35d999233a51

STATUS  
Running (13 seconds ago)

**Logs** Inspect Terminal Files Stats

```
2023-06-05 22:05:59
2023-06-05 22:05:59 > client@0.1.0 start
2023-06-05 22:05:59 > react-scripts start
2023-06-05 22:05:59
2023-06-05 22:06:12 [DEP_WEBPACK_DEV_SERVER_ON_AFTER_SETUP_MIDDLEWARE] DeprecationWarning: 'onAfterSetupMiddleware' option is deprecated. Please use the 'setupMiddleware' option.
2023-06-05 22:06:12 (Use `node --trace-deprecation ...` to show where the warning was created)
2023-06-05 22:06:12 [DEP_WEBPACK_DEV_SERVER_ON_BEFORE_SETUP_MIDDLEWARE] DeprecationWarning: 'onBeforeSetupMiddleware' option is deprecated. Please use the 'setupMiddleware' option.
2023-06-05 22:06:13 Starting the development server...
2023-06-05 22:06:13
2023-06-05 22:06:43 Compiled with warnings.
2023-06-05 22:06:43
2023-06-05 22:06:43 [eslint]
2023-06-05 22:06:43 src/App.js
2023-06-05 22:06:43 Line 1:8: 'logo' is defined but never used no-unused-vars
2023-06-05 22:06:43 Line 8:24: 'BrowserRouter' is defined but never used no-unused-vars
2023-06-05 22:06:43 Line 8:39: 'Router' is defined but never used no-unused-vars
2023-06-05 22:06:43
2023-06-05 22:06:43 src/component/Home.jsx
2023-06-05 22:06:43 Line 2:8: 'Navbar' is defined but never used no-unused-vars
2023-06-05 22:06:43 Line 3:8: 'Product' is defined but never used no-unused-vars
2023-06-05 22:06:43
2023-06-05 22:06:43 src/component/Product.jsx
2023-06-05 22:06:43 Line 2:10: 'useSelector' is defined but never used no-unused-vars
2023-06-05 22:06:43 Line 45:10: 'error' is assigned a value but never used no-unused-vars
2023-06-05 22:06:43 Line 51:9: 'addProduct' is assigned a value but never used no-unused-vars
```

↓

k8s\_frontend\_frontend-deployment-6968d9b847-  
4mrkd\_default\_7887ab97-  
ed36-46d5-b602-  
355bf79ac8d7\_0  
cd1e0c834959

STATUS  
Running (2 minutes ago)

**Logs** Inspect Terminal Files Stats

```
2023-06-05 22:05:59
2023-06-05 22:05:59 > client@0.1.0 start
2023-06-05 22:05:59 > react-scripts start
2023-06-05 22:05:59
2023-06-05 22:06:13 Starting the development server...
2023-06-05 22:06:13
2023-06-05 22:06:43 Compiled with warnings.
2023-06-05 22:06:43
2023-06-05 22:06:43 [eslint]
2023-06-05 22:06:43 src/App.js
2023-06-05 22:06:43 Line 1:8: 'logo' is defined but never used no-unused-vars
2023-06-05 22:06:43 Line 8:24: 'BrowserRouter' is defined but never used no-unused-vars
2023-06-05 22:06:43 Line 8:39: 'Router' is defined but never used no-unused-vars
2023-06-05 22:06:43
2023-06-05 22:06:43 src/component/Home.jsx
2023-06-05 22:06:43 Line 2:8: 'Navbar' is defined but never used no-unused-vars
2023-06-05 22:06:43 Line 3:8: 'Product' is defined but never used no-unused-vars
2023-06-05 22:06:43
2023-06-05 22:06:43 src/component/Product.jsx
2023-06-05 22:06:43 Line 2:10: 'useSelector' is defined but never used no-unused-vars
2023-06-05 22:06:43 Line 45:10: 'error' is assigned a value but never used no-unused-vars
2023-06-05 22:06:43 Line 51:9: 'addProduct' is assigned a value but never used no-unused-vars
2023-06-05 22:06:43
2023-06-05 22:06:43 src/component/Products.jsx
2023-06-05 22:06:43 Line 24:6: React Hook useEffect has missing dependencies: 'componentMounted' and 'filter'. Either include them or remove the dependency array react-hooks/exhaustive-deps
2023-06-05 22:06:43
```

↓

On fait un kubectl get all pour voir l'état de tous les pods qu'on a.

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
service/frontend created
deployment.apps/frontend-deployment created
● C:/Users/Lenovo/Desktop/mcommerce/kubernetes> kubectl get all
NAME                                         READY   STATUS    RESTARTS   AGE
pod/commande-deployment-76c9fec7f5-p6hpv   1/1    Running   1          64m
pod/commande-deployment-76c9fec7f5-vml22   1/1    Running   2          64m
pod/frontend-deployment-6968db847-4mrkd     1/1    Running   0          45m
pod/frontend-deployment-6968db847-xbx7t     1/1    Running   0          45m
pod/paiement-deployment-65c8c864d6-srp98   1/1    Running   0          52m
pod/paiement-deployment-65c8c864d6-wngl1   1/1    Running   0          52m
pod/product-deployment-5797887f97-vzhfl    1/1    Running   0          80m
pod/product-deployment-5797887f97-vml2g    1/1    Running   0          80m

NAME           TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
service/commande   ClusterIP   10.96.162.27   <none>       5001/TCP   64m
service/frontend   LoadBalancer  10.100.231.163  localhost   3000:31441/TCP   45m
service/kubernetes ClusterIP   10.96.0.1      <none>       443/TCP    3h51m
service/paiement   ClusterIP   10.97.222.200   <none>       5002/TCP   52m
service/product    ClusterIP   10.96.246.186   <none>       5000/TCP   79m

NAME          READY  UP-TO-DATE  AVAILABLE  AGE
deployment.apps/commande-deployment   2/2    2          2          64m
deployment.apps/frontend-deployment  2/2    2          2          45m
deployment.apps/paiement-deployment  2/2    2          2          52m
deployment.apps/product-deployment   2/2    2          2          80m

NAME          DESIRED  CURRENT  READY  AGE
replicaset.apps/commande-deployment-76c9fec7f5  2        2        2      64m
replicaset.apps/frontend-deployment-6968db847   2        2        2      45m
replicaset.apps/paiement-deployment-65c8c864d6  2        2        2      52m
replicaset.apps/product-deployment-5797887f97   2        2        2      80m
○ PS C:/Users/Lenovo/Desktop/mcommerce/kubernetes>

```

On push ensuite le dossier yaml\_files contenant nos fichier dans un repo gitlab qu'on a appellé kubernetes

Name	Last commit	Last update
..		
.gitkeep	Add new directory	21 hours ago
client.yaml	Upload New File	21 hours ago
commande.yaml	commande microservice version 1.0.2	4 hours ago
paiement.yaml	paiement microservice version 1.0.2	2 hours ago
product.yaml	Upload New File	21 hours ago

## IV. Automatisation avec Gitlab CI:

Cette partie vise à **automatiser** le build, scans de sécurité, packaging et publication de l'image Docker de chaque service en plus du frontend avec **GitLab CI**.

Pour ce faire, on a accédé à l'outil d'intégration continue Gitlab CI.

- Tout d'abord, on sélectionne la fonctionnalité CI/CD

The screenshot shows the GitLab homepage with three main sections: "Agile with GitLab", "CI/CD", and "Collaboration".

- Agile with GitLab:** Manage your work with built-in agile features. Includes links to Issues, Issue boards, and Labels.
- CI/CD:** Build, test, and deploy applications using Continuous Integration and Delivery. Includes links to .gitlab-ci.yml reference, GitLab Runner, and Auto DevOps.
- Collaboration:** Configure and develop your software applications. Includes links to Projects, Repositories, and Merge requests.

- Puis, on importe nos microservices dans un seul repository.

The screenshot shows the "Create new project" interface. It includes four options: "Create blank project", "Create from template", "Import project", and "Run CI/CD for external repository". On the right, a sidebar shows a project structure under "main": "ecommerce /". The "Name" field is set to "client". Other visible folder names include "commande", "paiement", "produit", and ".gitlab-ci.yml".

- Ensuite, comme bonne pratique, on enregistre le mot de passe de notre registre de conteneurs en ligne, Docker Hub. Et on s'assure qu'il est enregistré sous forme d'une variable d'environnement protégée afin qu'il soit masqué pour des raisons de sécurité.

The screenshot shows the "CI/CD Settings" page for a project named "client". It displays information about variables, including a note that variables can have several attributes like Protected, Masked, and Expanded. A table lists variables, showing one entry: "Type" (String), "Key" (Docker\_Hub\_Password), "Value" (redacted), "Options" (Protected), and "Environments" (main). A message at the bottom states "There are no variables yet." A blue button labeled "Add variable" is visible at the bottom left.

Add variable

**Key**

**Value**

**Type**

**Environment scope**

**Flags**

Protect variable  
Export variable to pipelines running on protected branches and tags only.

Mask variable  
Variable will be masked in job logs. Requires values to meet regular expression requirements. [Learn more](#).

- A ce stade, et après avoir créé les fichiers de configurations de chaque microservices, on crée le fichier **.gitlab-ci.yml** qui englobe les chemins vers ces derniers, ainsi que les stages qui seront exécutés lors de lancement du pipeline.

📌 **.gitlab-ci.yml** 232 bytes

```

1 stages:
2   - test
3   - build
4   - package
5   - sast
6   - deploy
7   - dast
8
9
10 include:
11   - local: 'produit/.gitlab-ci.yml'
12   - local: 'commande/.gitlab-ci.yml'
13   - local: 'paiement/.gitlab-ci.yml'
14   - local: 'client/.gitlab-ci.yml'
15

```

- Concernant, les fichiers de configuration **.gitlab-ci.yml** de chaque microservice, ils contiennent généralement quatre stages : test, build, sast et deploy.

On va détailler chaque stage par la suite pour le cas du microservice produit :

Initialement, on spécifie l'image Docker à utiliser qui est dans notre cas **docker:latest**, puis on définit le supplémentaire service **docker:dind** (Docker-in-Docker) qui permet d'avoir un environnement Docker complet à l'intérieur du conteneur, ce qui facilite l'exécution de tâches Docker supplémentaires, telles que la construction et le déploiement de conteneurs. Enfin, on spécifie l'ordre d'exécution des étapes.

```

.gitlab-ci.yml  1.40 KiB
1  image: docker:latest
2
3  services:
4    - docker:dind
5
6
7  stages:
8    - test
9    - build
10   - package
11   - sast
12   - deploy
13   - dast
14

```

Au niveau du job "**product-node-build**" qui est dans l'étape "**build**" du pipeline CI. On utilise une image Docker contenant **Node.js** pour construire ce projet. On commence par installer les dépendances à l'aide de npm install, puis exécute le script de construction avec **npm run build**. Cela permet de générer les artefacts nécessaires pour le déploiement ou d'autres étapes ultérieures du pipeline.

```

32
33  product-node-build:
34    image: node:latest
35    stage: build
36    script:
37      - cd ./produit
38      - npm install
39      - npm run build
40

```

Ensuite, on définit le job "**product-docker-build**" qui se situe dans l'étape "**package**" du pipeline CI. Il se charge de construire une image Docker à partir du Dockerfile présent dans le répertoire *./produit*, puis de pousser cette image vers le registre Docker Hub en utilisant les informations d'identification fournies par la commande **docker login**.

```

product-docker-build:
  stage: package
  before_script:
    - docker login -u laaziz2001 -p $DOCKER_HUB_PASSWORD
  script:
    - docker build -t $DOCKER_IMAGE_NAME ./produit
    - docker push $DOCKER_IMAGE_NAME

```

Puis, on configure le job "**product-docker-deploy**" situé dans l'étape "**deploy**" du pipeline CI. Il utilise la commande **docker run** pour lancer un conteneur Docker à partir de l'image product déjà créée, expose le port 5000, attend 30 secondes pour permettre au conteneur de démarrer, puis affiche les journaux du conteneur.

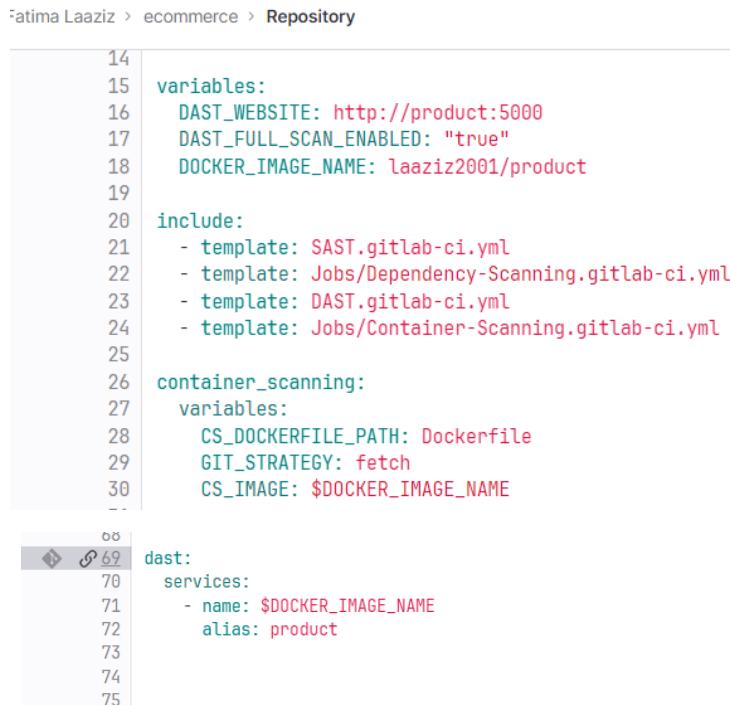
```

product-docker-deploy:
  stage: deploy
  before_script:
    - docker login -u laaziz2001 -p $DOCKER_HUB_PASSWORD
  script:
    - docker run -d -p 5000:5000 --name product $DOCKER_IMAGE_NAME

```

A la suite de cela, on ajoute des fonctionnalités de sécurité pour analyser l'image docker qu'on a créée.

- On définit des variables d'environnement pour spécifier des valeurs telles que l'URL du site web pour l'analyse de sécurité dynamique (**DAST\_WEBSITE**), l'activation de l'analyse complète (**DAST\_FULL\_SCAN\_ENABLED**) et le nom de l'image Docker utilisée (**DOCKER\_IMAGE\_NAME**).
- On inclut plusieurs templates prédéfinis pour ajouter des fonctionnalités spécifiques au pipeline. Ces templates concernent l'analyse statique de sécurité (**SAST**), l'analyse des dépendances, l'analyse de sécurité dynamique (DAST) et l'analyse des conteneurs.
- Et on configure l'étape de scan des conteneurs avec des variables spécifiques, notamment le chemin vers le fichier Dockerfile (**CS\_DOCKERFILE\_PATH**), la stratégie Git de récupération du code source (**GIT\_STRATEGY**) et l'image Docker à analyser (**CS\_IMAGE**).
- Enfin, on configure l'étape d'analyse de sécurité dynamique (DAST) du pipeline CI en spécifiant un service Docker supplémentaire à utiliser pendant cette étape. Le service est nommé en utilisant la variable **\$DOCKER\_IMAGE\_NAME** et est accessible avec l'alias **product**.



```

Fatima Laaziz > ecommerce > Repository

14
15 variables:
16   DAST_WEBSITE: http://product:5000
17   DAST_FULL_SCAN_ENABLED: "true"
18   DOCKER_IMAGE_NAME: laaziz2001/product
19
20 include:
21   - template: SAST.gitlab-ci.yml
22   - template: Jobs/Dependency-Scanning.gitlab-ci.yml
23   - template: DAST.gitlab-ci.yml
24   - template: Jobs/Container-Scanning.gitlab-ci.yml
25
26 container_scanning:
27   variables:
28     CS_DOCKERFILE_PATH: Dockerfile
29     GIT_STRATEGY: fetch
30     CS_IMAGE: $DOCKER_IMAGE_NAME
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
69
70   dast:
71     services:
72       - name: $DOCKER_IMAGE_NAME
73         alias: product
74
75

```

A la fin, on définit le job "**product-nodejsscan**" qui se situe dans l'étape "**sast**" du pipeline CI. Il utilise une image Docker contenant Node.js pour exécuter une analyse de sécurité statique sur notre projet Node.js produit. Il installe les dépendances, exécute **npm audit** pour

l'analyse et stocke le rapport de sécurité généré dans un fichier JSON. Ce rapport est ensuite collecté en tant qu'**artefact** pour référence ultérieure.

```
49  product-nodejsscan:
50    stage: sast
51    image: node:14
52    script:
53      - cd ./produit
54      - npm install
55      - npm audit --json > product-service-security-report.json
56    artifacts:
57      paths:
58        - produit/product-service-security-report.json
```

Après l'exécution du pipeline, on remarque que tous les jobs sont passés avec succès.

test	build	package	sast	deploy
✓ brakeman-sast	✓ commande-node-build	✓ Mcommerce-docker-build	✓ Mcommerce-nodejsscan	✓ Mcommerce-docker-deploy
✓ container_scanning	✓ paiement-node-build	✓ commande-docker-build	✓ commande-nodejsscan	✓ commande-docker-deploy
✓ flawfinder-sast	✓ product-node-build	✓ paiement-docker-build	✓ paiement-nodejsscan	✓ paiement-docker-deploy
✓ kics-lac-sast		✓ product-docker-build	✓ product-nodejsscan	✓ product-docker-deploy
✓ nodejs-scan-sast				
✓ phpcs-security-audit-sast				
✓ pmd-apex-sast				
✓ semgrep-sast				
✓ sobelow-sast				
✓ spotbugs-sast				

Voyons l'output des jobs dans chaque stage un peu plus en détails :

#### A. Le stage « build » :

Ce job s'est déroulé avec succès. Les dépendances ont été installées correctement, aucune vulnérabilité n'a été détectée et le projet a été construit avec succès.

```

24 Downloading artifacts
25 Downloading artifacts for container_scanning (4421640598)...
26 Downloading artifacts from coordinator... ok          host=storage.googleapis.com id=4421640598
   responseStatus=200 OK token=64_8VWJa
28 Executing "step_script" stage of the job script
29 Using docker image sha256:f03de6896e9e068cf6c241b5d668cc1f4b3c851a0b32939fe4ebacf145b212f5 fo
   r node:latest with digest node@sha256:14f0471d0478fbb9177d0f9e8c146dc872273cdccfc7fea93a27ed81
   fc6b0e96 ...
30 $ cd ./produit
31 $ npm install
32 up to date, audited 154 packages in 728ms
33 20 packages are looking for funding
34   run `npm fund` for details
35 found 0 vulnerabilities
36 $ npm run build
37 > produit@1.0.0 build
38 > npm install
39 up to date, audited 154 packages in 508ms
40 20 packages are looking for funding
41   run `npm fund` for details
42 found 0 vulnerabilities
44 Cleaning up project directory and file based variables
46 Job succeeded

```

## B. Le stage « package » :

L'output de ce job montre que le processus de construction de l'image Docker et son déploiement vers le référentiel Docker se sont déroulés avec succès. Aucune vulnérabilité n'a été détectée et l'image est prête à être utilisée dans les étapes suivantes du pipeline CI/CD.

```

30 $ docker login -u laaziz2001 -p $DOCKER_HUB_PASSWORD
31 WARNING! Using --password via the CLI is insecure. Use --password-stdin.
32 WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
33 Configure a credential helper to remove this warning. See
34 https://docs.docker.com/engine/reference/commandline/login/#credentials-store
35 Login Succeeded
36 $ docker build -t $DOCKER_IMAGE_NAME ./produit
37 #1 [internal] load build definition from Dockerfile
38 #1 transferring dockerfile: 246B done
39 #1 DONE 0.0s
40 #2 [internal] load .dockerignore

```

```
$ docker push $DOCKER_IMAGE_NAME
Using default tag: latest
The push refers to repository [docker.io/laaziz2001/product]
9b3d3e0e39b7: Preparing
dca885f17a40: Preparing
f18533b29f58: Preparing
5bdb2acdc609: Preparing
37b7f7a9d2a0: Preparing
e394b5a78515: Preparing
```

### C. Le stage « deploy » :

On remarque sur la figure ci-dessous que ce job tente de se connecter à Docker Hub, télécharge une image Docker, exécute un conteneur à partir de cette image.

```
38 $ docker login -u laaziz2001 -p $DOCKER_HUB_PASSWORD
39 WARNING! Using --password via the CLI is insecure. Use --password-stdin.
40 WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
41 Configure a credential helper to remove this warning. See
42 https://docs.docker.com/engine/reference/commandline/login/#credentials-store
43 Login Succeeded
44 $ docker run -d -p 5000:5000 --name product $DOCKER_IMAGE_NAME
45 Unable to find image 'laaziz2001/product:latest' locally
46 latest: Pulling from laaziz2001/product
47 8a49fdb3b6a5: Pulling fs layer
48 1197750296b3: Pulling fs layer
49 f352bc07f19b: Pulling fs layer
50 47be83a79857: Pulling fs layer
```

### D. Le stage « sast » :

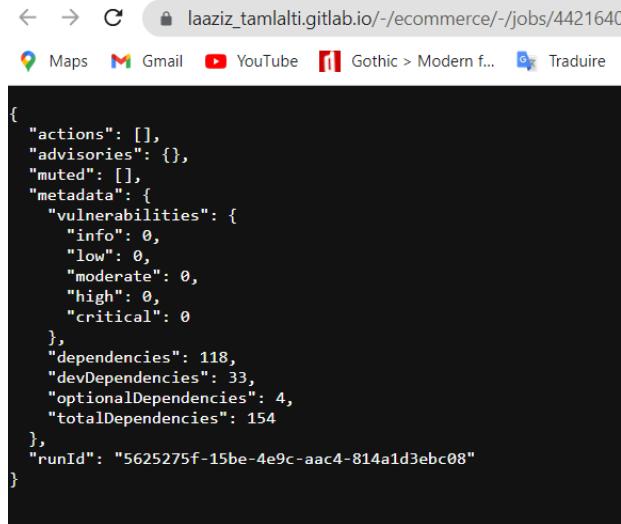
Au niveau de ce job, on observe l'installation des dépendances du projet produit Node.js, des avertissements liés à la version du lockfile et à l'absence du champ repository, ainsi que la génération d'un rapport de sécurité.

```

31 $ npm install
32 npm WARN read-shrinkwrap This version of npm is compatible with lockfileVersion@1, but packag
e-lock.json was generated for lockfileVersion@2. I'll try to do my best with it!
33 npm WARN produit@1.0.0 No repository field.
34 npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@2.3.2 (node_modules/fsevents):
35 npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@2.3.2: wanted
{"os":"darwin","arch":"any"} (current: {"os":"linux","arch":"x64"})
36 updated 153 packages and audited 154 packages in 18.931s
37 20 packages are looking for funding
38   run `npm fund` for details
39 found 0 vulnerabilities
40 $ npm audit --json > product-service-security-report.json
42 Uploading artifacts for successful job
43 Uploading artifacts...
44 produit/product-service-security-report.json: found 1 matching artifact files and directories
45 WARNING: Upload request redirected           location=https://gitlab.com/api/v4/jobs/4
421640620/artifacts?artifact_format=zip&artifact_type=archive new_url=https://gitlab.com
46 WARNING: Retrying...                         context=artifacts-uploader error=request
redirected
47 Uploading artifacts as "archive" to coordinator... 201 Created id=4421640620 responseStatus=
201 Created token=64_Pymzg
49 Cleaning up project directory and file based variables
51 Job succeeded

```

Le job se déroule sans erreurs et parvient à télécharger l'artefact généré, qui est le suivant :



The screenshot shows a browser window with the URL [https://laaziz\\_tamlalti.gitlab.io/-/ecommerce/-/jobs/4421640](https://laaziz_tamlalti.gitlab.io/-/ecommerce/-/jobs/4421640). The page displays a JSON object representing a security audit report. The JSON structure is as follows:

```

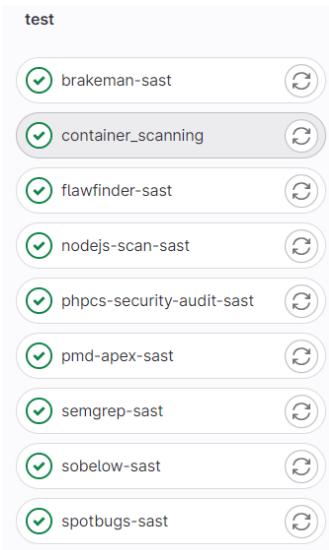
{
  "actions": [],
  "advisories": {},
  "muted": [],
  "metadata": {
    "vulnerabilities": {
      "info": 0,
      "low": 0,
      "moderate": 0,
      "high": 0,
      "critical": 0
    },
    "dependencies": 118,
    "devDependencies": 33,
    "optionalDependencies": 4,
    "totalDependencies": 154
  },
  "runId": "5625275f-15be-4e9c-aac4-814a1d3ebc08"
}

```

Ce rapport de sécurité est positif, car il ne signale aucune vulnérabilité ("info": 0, "low": 0, "moderate": 0, "high": 0, "critical": 0) dans les dépendances du projet. Cela indique que les packages utilisés sont à jour et ne présentent pas de risques connus en termes de sécurité.

## E. Le stage « test » :

Comme ce qu'on a déjà vu, ce stage comporte plusieurs jobs :



On peut voir par exemple, l'output du job container\_scanning qui est représenté par les figures suivantes :

```

51 [INFO] [2023-06-06 17:05:41 +0000] [container-scanning] > Scanning container from registry
laaziz2001/product for vulnerabilities with severity level UNKNOWN or higher, with gcs 6.0.1 a
nd Trivy Version: 0.36.1, advisories updated at 2023-06-06T12:10:02+00:00
52 +-----+-----+-----+
-----+
53 | STATUS | CVE SEVERITY | PACKAGE NAME | PACKAGE VERSION | CVE
DESCRIPTION | |
54 +-----+-----+-----+
-----+
55 | Unapproved | High | libcrypto3 | 3.1.0-r4 | Issue summary: Processing some
specially crafted ASN.1 object identifi |
56 | | | | | |
57 | | | | | data containin
g them may be very slow. |
58 | | | | | Impact summary: Applications t
hat use OBJ_obj2txt() directly, or use a |
59 | | | | | |
60 | | | | | the OpenSSL subsystems OCSP, P
KCS7/SMIME, CMS, CMP/CRMF or TS with no |
61 | | | | | |
62 | | | | | message |
63 | | | | | size limit may experience nota
ble to very long delays when processing |

```

```

-----+
155 [INFO] [2023-06-06 17:05:48 +0000] [container-scanning] > Scanning container from registry
laaziz2001/product for vulnerabilities with severity level UNKNOWN or higher, with gcs 6.0.1 a
nd Trivy Version: 0.36.1, advisories updated at 2023-06-06T12:10:02+00:00
156 [INFO] [2023-06-06 17:05:52 +0000] [container-scanning] > Scanning container from registry
laaziz2001/product for vulnerabilities with severity level UNKNOWN or higher, with gcs 6.0.1 a
nd Trivy Version: 0.36.1, advisories updated at 2023-06-06T12:10:02+00:00
158 Uploading artifacts for successful job 00:05
159 Uploading artifacts...
160 gl-container-scanning-report.json: found 1 matching artifact files and directories
161 gl-dependency-scanning-report.json: found 1 matching artifact files and directories
162 **/gl-sbom-*.cdx.json: found 1 matching artifact files and directories
163 WARNING: Upload request redirected location=https://gitlab.com/api/v4/jobs/4
423542393/artifacts?artifact_format=zip&artifact_type=archive new-url=https://gitlab.com
164 WARNING: Retrying... context=artifacts-uploader error=request
redirected
165 Uploading artifacts as "archive" to coordinator... 201 Created id=4423542393 responseStatus=
201 Created token=64_Ty8UB
166 Uploading artifacts...
167 gl-container-scanning-report.json: found 1 matching artifact files and directories
168 WARNING: Upload request redirected location=https://gitlab.com/api/v4/jobs/4
423542393/artifacts?artifact_format=raw&artifact_type=container_scanning new-url=https://gitla
b.com
169 WARNING: Retrying... context=artifacts-uploader error=request
redirected

```

On remarque que ce job se déroule avec succès et parvient à télécharger plusieurs artefacts générés

 passed Job #4423542393 in pipeline #891308949 for f1e9fce5 from main by  Fatima Laaziz 27 minutes ago

---

Artifacts

Name
 gl-container-scanning-report.json 
 gl-dependency-scanning-report.json 
 gl-sbom-report.cdx.json 

Pour le cas du rapport **gl-container-scanning-report.json** : Il identifie deux vulnérabilités de niveau élevé dans notre application. Les vulnérabilités sont liées aux paquets libcrypto3 et libssl3 dans la version 3.1.0-r4, utilisée sur une image Docker basée sur Alpine 3.18.0.

```

{
  "vulnerabilities": [
    {
      "id": "e8030482f8a2f55ae63effebe403772329c0b99",
      "severity": "High",
      "location": {
        "dependency": {
          "package": "libcrypto3",
          "version": "3.1.0-r4"
        },
        "operating_system": "alpine 3.18.0",
        "image": "laazizz2001/product"
      },
      "identifiers": [
        {
          "type": "cve",
          "name": "CVE-2023-2650",
          "value": "CVE-2023-2650"
        }
      ],
      "url": "https://www.openwall.com/lists/oss-security/2023/05/30/1"
    }
  ]
}

```

The screenshot shows a detailed report from laazizz.gitlab.io about a security vulnerability (CVE-2023-2650). The report discusses ASN.1 object identifiers and their impact on OpenSSL. It includes links to various sources like Openwall, CVE-2023-2650, and the OpenSSL git repository.

## V. Création d'un répertoire GitLab pour le déploiement local d'une application Kubernetes :

La création d'un répertoire GitLab pour le déploiement local d'une application Kubernetes permet de bénéficier de la gestion de version, de la collaboration, du suivi des modifications, de l'automatisation du déploiement et de la sauvegarde/reprise, ce qui facilite le processus de déploiement et la gestion de l'infrastructure Kubernetes.

Pour ce faire, on va créer un repository sur Gitlab-CI :

Name	Last commit
yaml_files	paiement microservice
README.md	Initial commit
argocd.yaml	Update argocd.yaml

Au sein de ce repository, on va créer un dossier yaml\_file contenant les fichiers YAML pour le déploiement et le service de chaque microservice.

Par exemple pour le cas du microservice commande, son fichier YAML se compose de deux parties principales :

### 1. Service :

- ❖ L'objet "**Service**" permet d'exposer l'application à d'autres services ou utilisateurs dans le cluster.
- ❖ Le service est nommé "**commande**" et utilise le port 5001.
- ❖ Le sélecteur "**app: commande**" garantit que ce service est associé aux pods étiquetés avec "app: commande".

```
1 # commande.yaml
2
3 apiVersion: v1
4 kind: Service
5 metadata:
6   name: commande
7 spec:
8   selector:
9     app: commande
10  ports:
11    - port: 5001
12
```

### 2. Déploiement :

- ❖ L'objet "**Deployment**" gère le déploiement des pods, qui exécutent l'application.
- ❖ Le déploiement est nommé "**commande-deployment**" et spécifie qu'il doit y avoir **deux répliques (pods)** de l'application.
- ❖ Le sélecteur "matchLabels" assure que les pods du déploiement sont étiquetés avec "app: commande".
- ❖ Dans la section "template", les pods sont configurés :
  - Ils portent l'étiquette "app: commande".
  - Le conteneur s'appelle "commande" et utilise l'image "maryam2001/commande\_service:1.0.1".Le port 5001 est exposé dans le conteneur sous le nom "commande-port".

```

15
16 apiVersion: apps/v1
17 kind: Deployment
18 metadata:
19   name: commande-deployment
20 spec:
21   replicas: 2
22   selector:
23     matchLabels:
24       app: commande
25   template:
26     metadata:
27       labels:
28         app: commande
29     spec:
30       containers:
31         - name: commande
32           image: maryam2001/commande_service:1.0.1
33           ports:
34             - containerPort: 5001
35               name: commande-port
36

```

Après avoir configuré les fichiers YAML des autres microservices.

Ce fichier YAML configure notre application dans Argo CD pour déployer les fichiers YAML d'une application provenant d'un référentiel Git spécifique. L'application sera déployée dans un cluster Kubernetes donné, dans un namespace spécifié. La synchronisation sera automatisée, et les ressources non déclarées dans les fichiers YAML de l'application seront supprimées lors de la synchronisation.

```

main ▾ kubernetes / argocd.yaml

argocd.yaml 358 bytes

1  apiVersion: argoproj.io/v1alpha1
2  Kind: Application
3  metadata:
4    name: product
5  spec:
6    project: default
7    source:
8      repoURL: https://gitlab.com/maryam2001/kubernetes.git
9      targetRevision: main
10     path: yaml_files/product.yaml
11   destination:
12     server: https://kubernetes.default.svc
13     namespace: default
14   syncPolicy:
15     automated:
16       prune: true
17

```

## VI. Utilisation de l'outil ArgoCD :

En intégrant **ArgoCD** dans le processus, les changements effectués dans le référentiel GitLab déclencheront une action de déploiement dans le pipeline CI de GitLab. ArgoCD sera responsable de détecter les modifications, de les appliquer dans le cluster Kubernetes local et de mettre à jour les microservices correspondants.

Cette approche garantit un déploiement continu et automatisé des microservices mis à jour, en s'appuyant sur les fichiers YAML de Kubernetes présents dans le référentiel GitLab. ArgoCD facilite la gestion et la synchronisation des déploiements, en assurant que le cluster Kubernetes local reflète les dernières modifications apportées au code source.

Pour réaliser cet objectif, on lance tout d'abord Docker desktop, puis en utilisant ce dernier on crée un cluster Kubernetes.

On passe ensuite à l'installation d'ArgoCD:

- On ouvre un terminal, on installe Helm et on ajoute le référentiel Helm ArgoCD :

```
PS C:\Users\Lenovo\Desktop\mcommerce> helm repo add argo https://argoproj.github.io/argo "argo" has been added to your repositories
```

- ```
PS C:\Users\Lenovo\Desktop\mcommerce> helm repo update
● Hang tight while we grab the latest from your chart repositories...
● ...Successfully got an update from the "argo" chart repository
Update Complete. Happy Helm-ing!
```

- On installe ArgoCD en utilisant Helm :

```
PS C:\Users\Lenovo\Desktop\mcommerce> helm install argocd argo/argo-cd
● NAME: argocd
LAST DEPLOYED: Tue Jun  6 09:46:55 2023
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
1. kubectl port-forward service/argocd-server -n default 8080:443
   and then open the browser on http://localhost:8080 and accept the certificate
2. enable ingress in the values file "server.ingress.enabled" and either
   - Set the "config.params."server.insecure"" in the values file and termina
ress/<option-2--multiple-ingress-objects-and-hosts
```

After reaching the UI the first time you can login with username: admin and the password:  
kubectl -n default get secret argocd-initial-admin-secret -o jsonpath="{.data.password}"  
(You should delete the initial secret afterwards as suggested by the Getting Started guide)

Après avoir lancé l'interface utilisateur, on clique sur le bouton "Nouvelle application" et on remplit les informations concernant notre application en ajoutant l'URL de notre repository sur Gitlab contenant les fichiers YAML.

Puis, on remarque que les fichiers YAML qu'on a créé précédemment sont déposés automatiquement au niveau d'ArgoCD

```

1 apiVersion: argoproj.io/v1alpha1
2 kind: Application
3 metadata:
4   name: product
5 spec:
6   project: default
7   source:
8     repoURL: https://gitlab.com/maryam2001/kubernetes.git
9     targetRevision: main
10    path: yaml_files/product.yaml
11   destination:
12     server: https://kubernetes.default.svc
13     namespace: default
14   syncPolicy:
15     automated:
16       prune: true
17
18

```

Cette interface nous montre que tous les pods sont sains

Au niveau de notre repository Gitlab CI qu'on a utilisé pour configurer notre pipeline, on ajoute un nouveau stage gitops-k8s qui effectue une modification sur notre déploiement.

```
34 gitops-k8s-deploy:
35   image: bitnami/git:latest
36   stage: gitops-k8s
37   before_script:
38     - git config --global user.email "mtamlalti@gmail.com"
39     - git config --global user.name "maryam2001"
40   script:
41     - git clone https://gitlab.com/maryam2001/kubernetes.git
42     - cd kubernetes/yaml_files/
43     - sed -i "s@maryam2001/commande_service:1.0.1@maryam2001/commande_service:$TAG@g" commande.yaml
44     - cat commande.yaml
45     - git add commande.yaml
46     - git commit -m "commande microservice version $TAG"
47     - git remote set-url origin https://maryam2001:$GIT_TOKEN@gitlab.com/maryam2001/kubernetes.git
48     - git push -uf origin main
49
```

On remarque que le commit s'est bien déroulé.

```
20 $ git remote set-url origin "${CI_REPOSITORY_URL}"
21 Executing "step_script" stage of the job script
22 Using docker image sha256:2af0249a4bf36e59f75486e6c92a1cfcd93622423c5fa89924afb7b4854df86f fo
r bitnami/git:latest with digest bitnami/git@sha256:89ac89173d93c88c91837c9bf8959c165a64a70437
ef607fe105c56ba84ac162 ...
23 $ git config --global user.email "mtamlalti@gmail.com"
24 $ git config --global user.name "maryam2001"
25 $ git clone https://gitlab.com/maryam2001/kubernetes.git
26 Cloning into 'kubernetes'...
27 $ cd kubernetes/yaml_files/
28 $ sed -i "s@maryam2001/commande_service:1.0.1@maryam2001/commande_service:$TAG@g" commande.ya
ml
29 $ cat commande.yaml
30 # commande.yaml
31 apiVersion: v1
32 kind: Service
33 metadata:
34   name: commande
35 spec:
36   selector:
37     app: commande
38   ports:
```

```

50 template:
51   metadata:
52     labels:
53       app: commande
54   spec:
55     containers:
56       - name: commande
57         image: maryam2001/commande_service:1.0.2
58       ports:
59         - containerPort: 5001
60           name: commande-port
61 $ git add commande.yaml
62 $ git commit -m "commande microservice version $TAG"
63 [main 492c442] commande microservice version 1.0.2
64 1 file changed, 1 insertion(+), 1 deletion(-)
65 $ git remote set-url origin https://maryam2001:$GIT_TOKEN@gitlab.com/maryam2001/kubernetes.git
66 $ git push -uf origin main
67 To https://gitlab.com/maryam2001/kubernetes.git
68   3a19565..492c442  main -> main
69 branch 'main' set up to track 'origin/main'.
70 Cleaning up project directory and file based variables
71 Job succeeded

```

commande-deployment-75f6f87c78-pjv9c ❤️

**SUMMARY**

|            |                                      |
|------------|--------------------------------------|
| KIND       | Pod                                  |
| NAME       | commande-deployment-75f6f87c78-pjv9c |
| NAMESPACE  | default                              |
| CREATED AT | 06/06/2023 18:59:44 (5 hours ago)    |
| IMAGES     | maryam2001/commande_service:1.0.2    |
| STATE      | Running                              |
| HEALTH     | ❤️ Healthy                           |
| LINKS      |                                      |

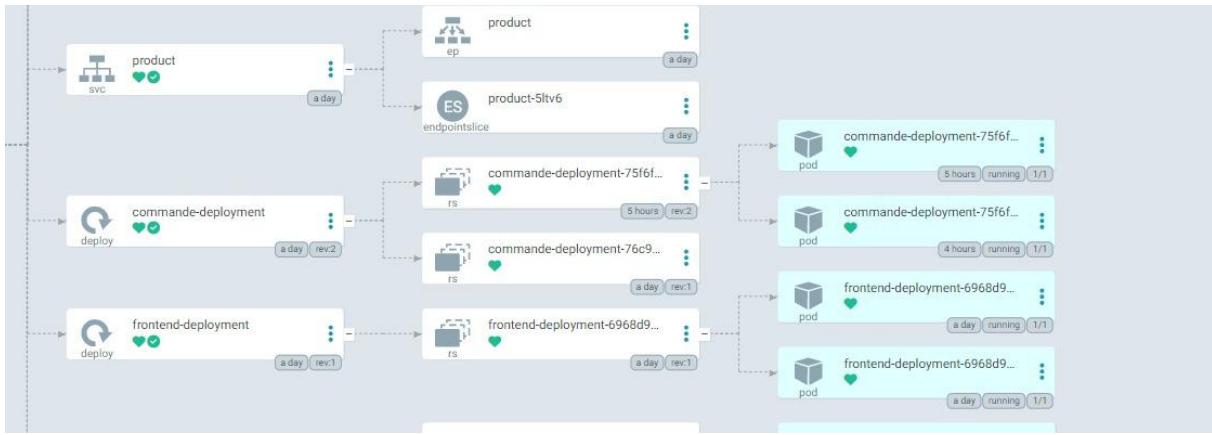
**Code**

```

15
16 apiVersion: apps/v1
17 kind: Deployment
18 metadata:
19   name: commande-deployment
20 spec:
21   replicas: 2
22   selector:
23     matchLabels:
24       app: commande
25   template:
26     metadata:
27       labels:
28         app: commande
29     spec:
30       containers:
31         - name: commande
32           image: maryam2001/commande_service:1.0.2
33         ports:
34           - containerPort: 5001
35             name: commande-port

```

Ainsi, les pods sont toujours healthy.



## VII. Conclusion

En conclusion, le projet de développement de l'application Mcommerce a permis de mettre en place une solution innovante en utilisant les principes et les technologies cloud-native. L'application, basée sur une architecture microservices, offre une expérience utilisateur engageante et une communication efficace entre les différents services. Grâce à l'utilisation de Node.js / Express pour le backend et React pour le frontend, l'application est performante et réactive. La conteneurisation avec Docker facilite le déploiement, tandis que l'utilisation de Kubernetes permet une gestion efficace de l'application dans un environnement de production. Dans l'ensemble, le projet démontre l'efficacité des approches cloud-native pour le développement et le déploiement d'applications modernes.