

# E-COMMERCE MICROSERVICE APP

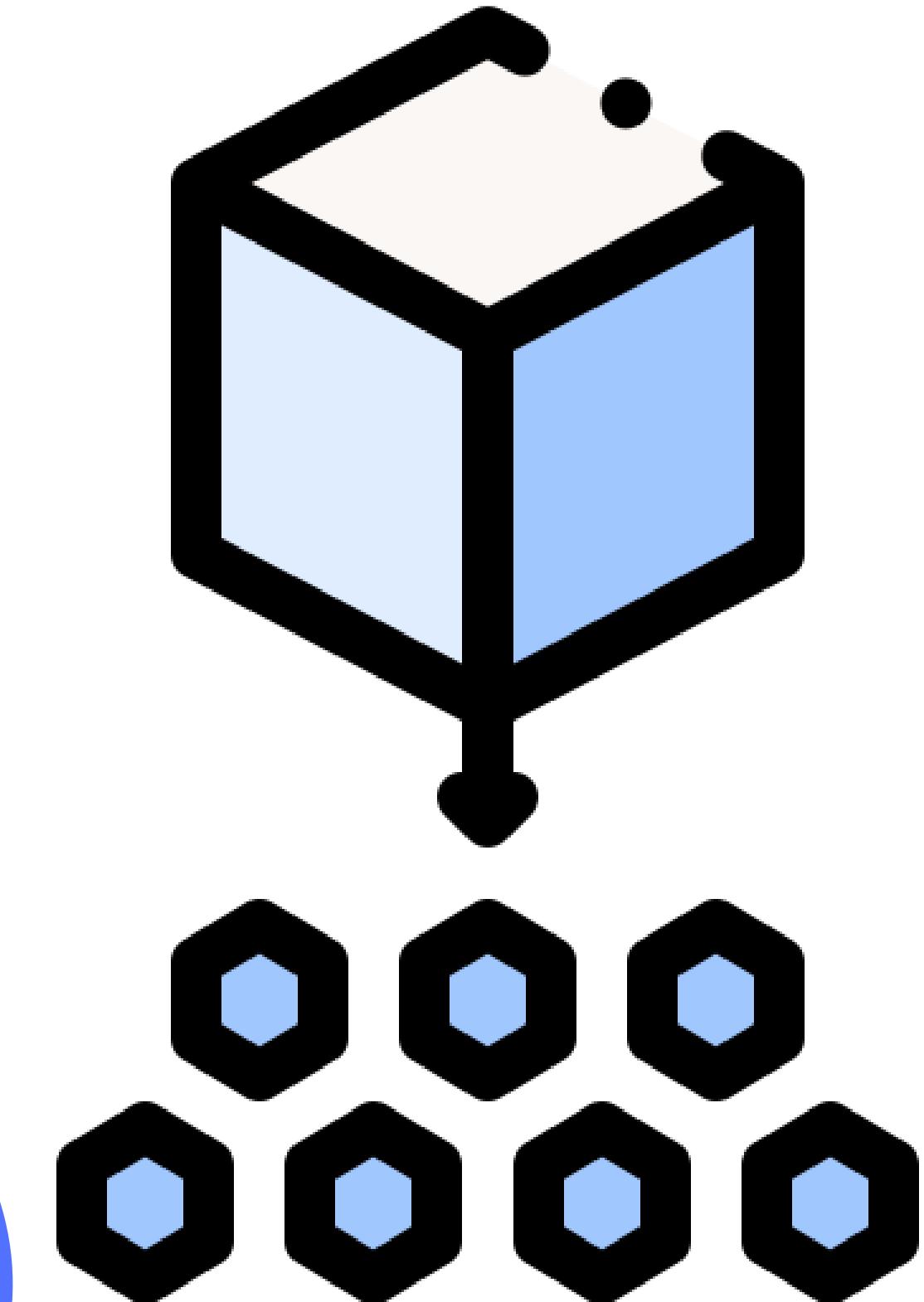


*Présenté par :*

- Asmae ELAZRAK
- Riad ELHAJJAJI

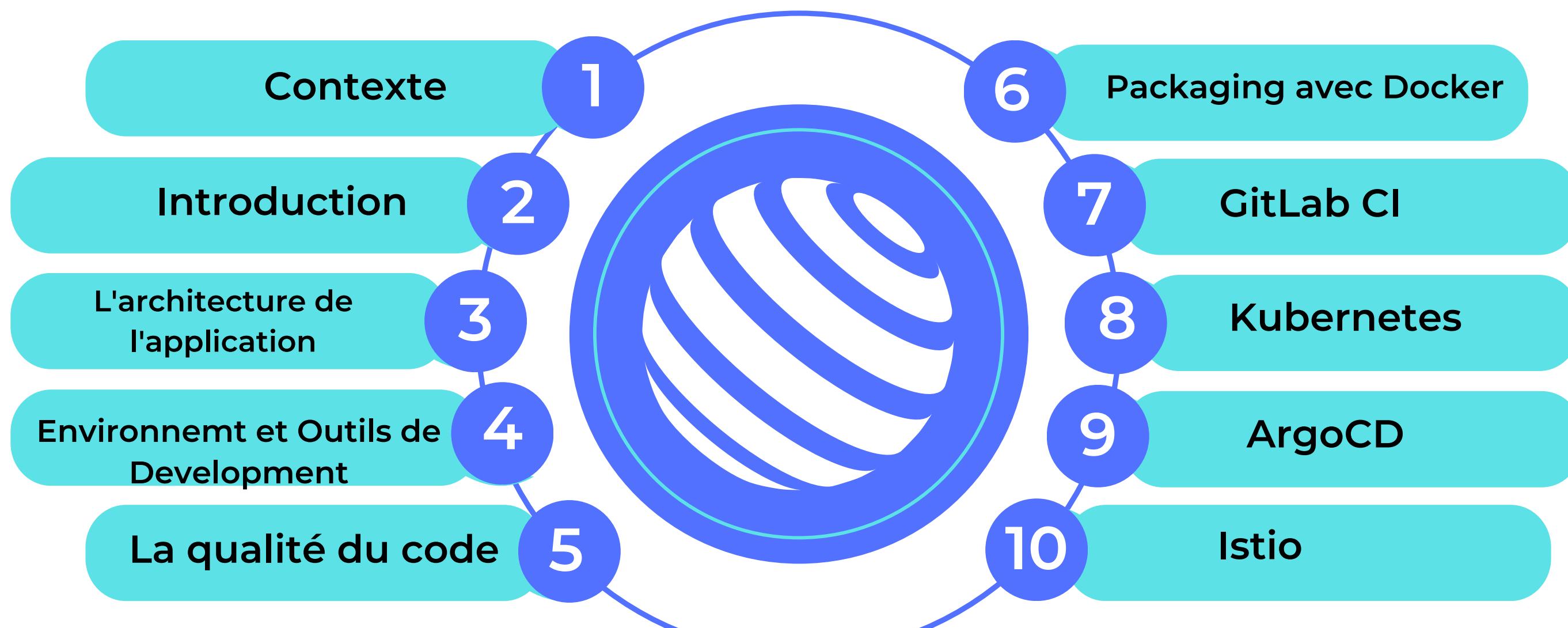
*Encadré par :*

- Pr. Driss ALLAKI



# TABLE OF CONTENT

---



# CONTEXTE

KATA est une jeune petite entreprise de développement de solutions et de services logiciels. Pour les quelques mois à venir, KATA veut saisir une nouvelle opportunité du marché, et compte se focaliser sur le développement d'une nouvelle solution avec une idée disruptive. Il s'agit d'une application engageante avec un fort potentiel d'utilisation à l'échelle par des milliers de clients. Conscients du manque d'expérience et d'expertise qu'ils ont par rapport au développement de ce type de projets, ils vous ont recruté afin de réaliser un PoC (Proof of Concept) leur permettant de découvrir et d'évaluer l'efficacité des concepts, pratiques et technologies Cloud Native. Le sujet de notre mission concerne le développement et le déploiement d'une application microservices nommée Mcommerce en utilisant les pratiques et technologies cloud-native (conteneurisation, orchestration des conteneurs, automatisation des déploiements, etc.).



# INTRODUCTION

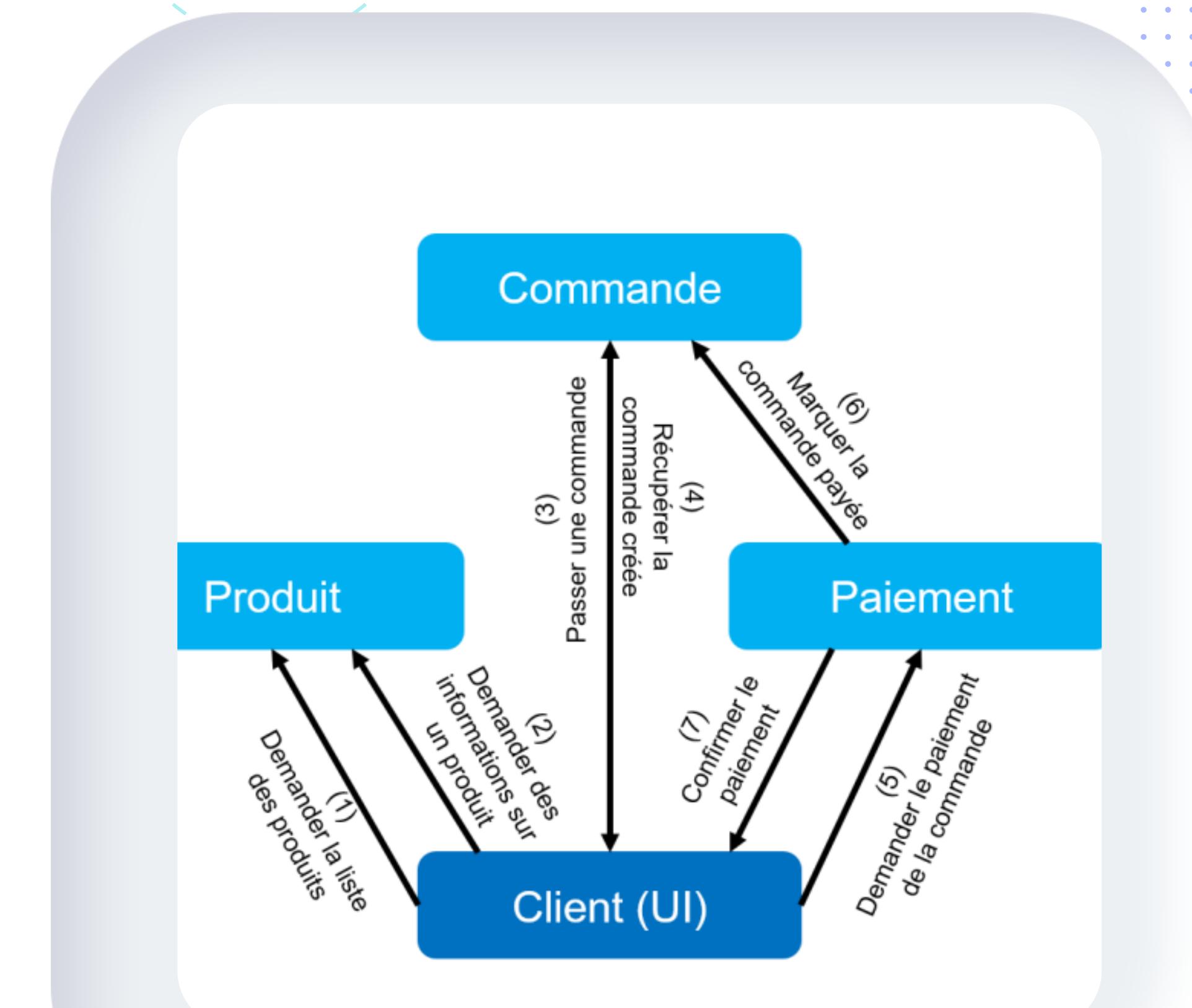
L'application développée, nommée Mcommerce, est basée sur une architecture microservices. Les services backend ont été construits en utilisant les technologies Node.js et Express, tandis que le frontend a été développé sous forme de Single Page Application (SPA) avec la librairie React. La communication entre le frontend et le backend s'effectue via une API REST et RabbitMq.

Dans le cadre de ce projet, nous avons également créé les fichiers Dockerfile nécessaires pour chaque service et en utilisant GitLab CI, nous avons automatisé le processus de build, les scans de sécurité, le packaging et la publication des images Docker correspondantes.

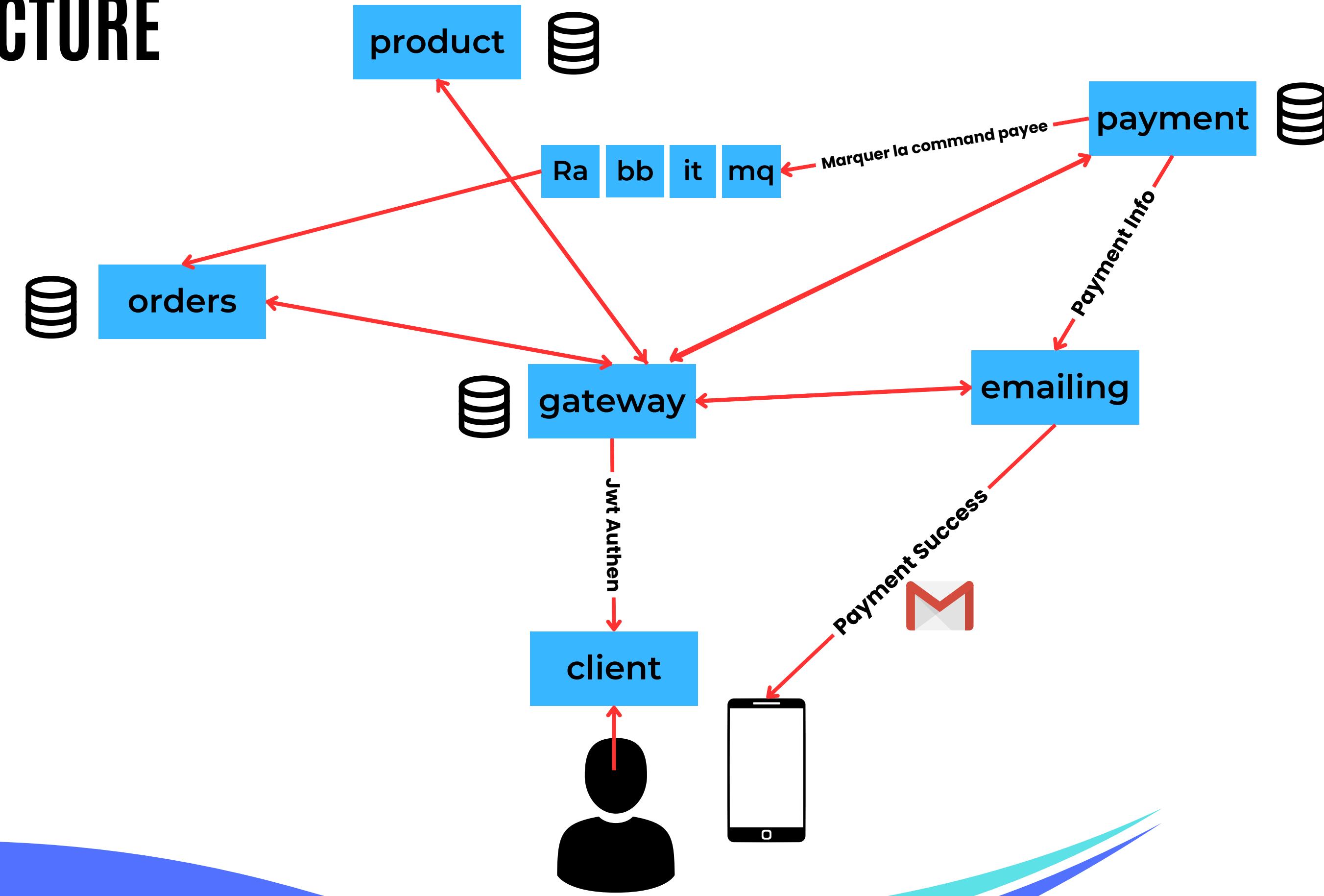
De plus, nous avons mis en place un répertoire GitLab contenant les fichiers YAML de Kubernetes, qui nous ont permis de déployer l'application dans un cluster Kubernetes en local. Afin de faciliter la gestion des déploiements, nous avons utilisé l'outil ArgoCD pour détecter les changements au niveau du repository GitLab.

Dans cette présentation on va explorer les différentes étapes que nous avons suivies pour développer et déployer cette application cloud-native, en mettant en évidence les avantages et les défis rencontrés.

# L'architecture de l'application



# ARCHITECTURE



# Front End SPA ??

Dans le projet, on est demandé d'utiliser React Js pour créer une SPA, Mais on a utilisé Next js qui est basé sur React Js Malgré Next.js n'est pas strictement un framework de monopage SPA. mais il est hybrides. Il vous permet de créer des pages Web dynamiques et interactives à l'aide des composants React et du rendu côté serveur (SSR).

Bien que Next.js puisse être utilisé pour créer des applications monopage (SPA), il offre également des fonctionnalités pour le rendu côté serveur et la génération de sites statiques. Avec le rendu côté serveur, le contenu HTML initial est généré sur le serveur et envoyé au client, ce qui offre de meilleurs avantages en matière de référencement et de performances par rapport au rendu traditionnel côté client. Next.js prend également en charge la génération de sites statiques, où les pages HTML sont pré-rendues au moment de la construction, permettant des sites Web rapides et évolutifs.

# Exploration de l'application



# landing page

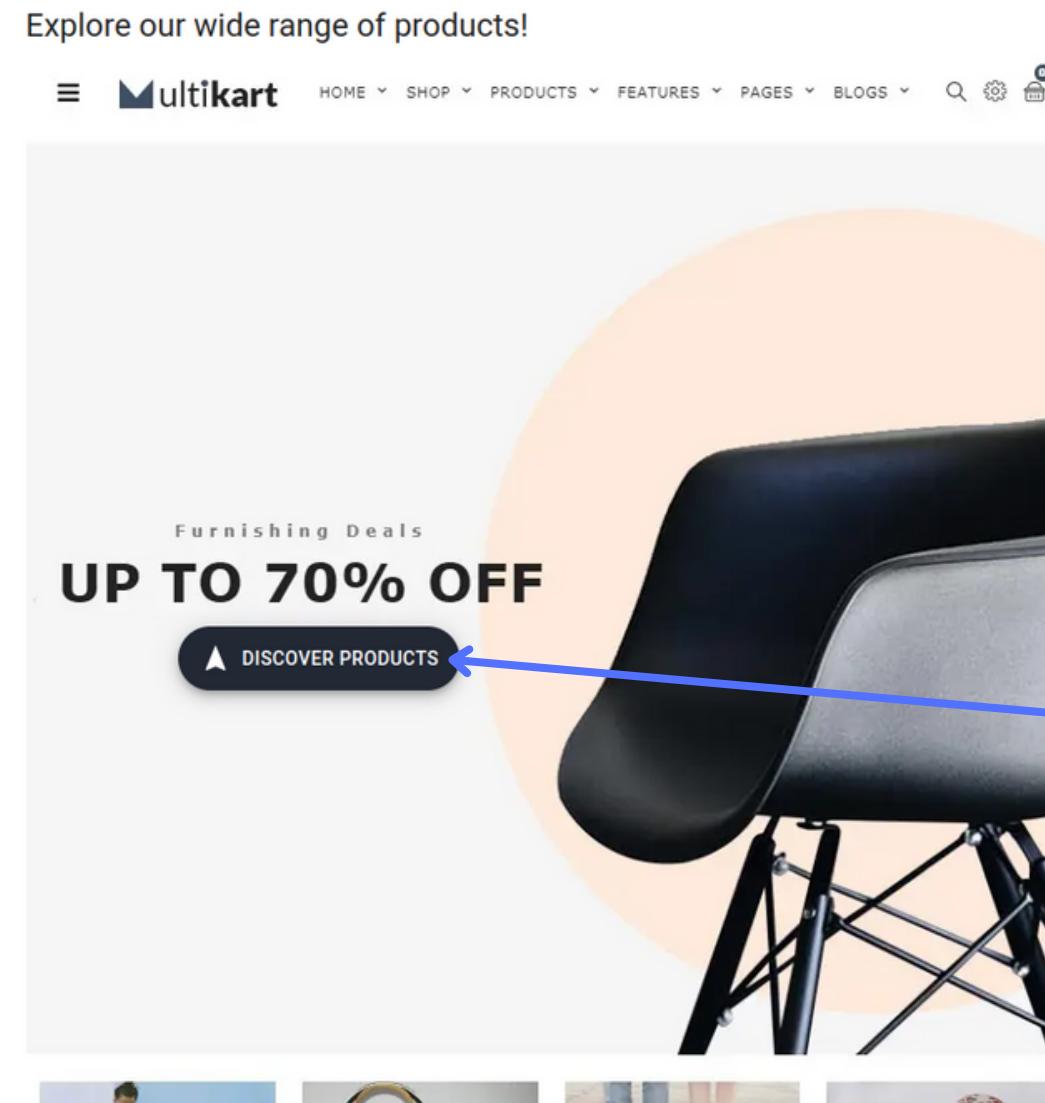
barre de navigation



Panier



Log In, Log Out



naviguer vers les produits

# page des produits

The screenshot shows a grid of 15 product cards on a website. The products include various electronic devices and office equipment. Each card displays the product image, name, description, rating, price, and a 'View' button.

| Product Image | Product Name  | Description  | Rating | Price    | Action               |
|---------------|---|--|--------|----------|----------------------|
|               | Epson L3100 Multi-function Color Printer (Black, Ink Tank)      | Similar features as Epson L3110, Epson L3101 printer Printer Tvoe - Ink Tank: gamer      | 4.6 ⭐  | 11199 DH | <a href="#">View</a> |
|               | Dell Inspiron 15 3000 Series 3511 Laptop, 15.6" FHD Touchscreen | Care Instructions: Wash with mild detergent. do not bleach.drv in shade Fit gamer        | 4.4 ⭐  | 1900 DH  | <a href="#">View</a> |
|               | 2022 Newest Dell Inspiron 3510 Laptop, 15.6 HD Display, Intel   | Care Instructions: Wash with mild detergent. do not bleach.drv in shade Fit gamer        | 4.4 ⭐  | 1900 DH  | <a href="#">View</a> |
|               | HP 15.6-inch Laptop, 11th Generation Intel Core i5-1135G7,      | Care Instructions: Wash with mild detergent. do not bleach.drv in shade Fit gamer        | 4.4 ⭐  | 1900 DH  | <a href="#">View</a> |
|               | APPLE iPhone 12 Pro (Gold, 128 GB)                              | A14 Bionic rockets past every other smartphone chip. The Pro camera developer            | 4.5 ⭐  | 11990 DH | <a href="#">View</a> |
|               | realme Narzo 30 5G (Racing Blue, 128 GB)                        | High-speed internet, smooth gaming, and stunning photos. the realme Narzo Pro developer  | 4.5 ⭐  | 15990 DH | <a href="#">View</a> |
|               | POCO M3 (Power Black, 64 GB)                                    | High-speed internet, smooth gaming, and stunning photos. the realme Narzo Pro developer  | 4.5 ⭐  | 11499 DH | <a href="#">View</a> |
|               | MOTOROLA G60 (Dynamic Gray, 128 GB)                             | The moto g60 enables you to capture spectacular selfies wherever you are. developer      | 4.5 ⭐  | 17990 DH | <a href="#">View</a> |
|               | Mi 11X (Lunar White, 128 GB)                                    | The Mi 11X from Xiaomi is the most affordable of three new models in the Mi photographer | 4.2 ⭐  | 27990 DH | <a href="#">View</a> |
|               | OPPO Reno6 Pro 5G (Aurora, 256 GB)                              | The OPPO Reno6 Pro 5G is not only easy on the eyes but also equipped with photographer   | 4.5 ⭐  | 55900 DH | <a href="#">View</a> |
|               |   |  |        |          |                      |
|               |   |  |        |          |                      |
|               |   |  |        |          |                      |
|               |   |  |        |          |                      |
|               |   |  |        |          |                      |

→ Les produits sont rendu dans le cote serveur

# details d'un produit

placer le curseur  
pour zoomer une  
partie du produit

The screenshot shows a product detail page for an Epson L3100 Multi-function Color Printer (Black, Ink Tank). The page includes a navigation bar with 'SUD-E-COMMERCE', a search bar, and links for 'Home', 'Products', and 'About us'. A user's email 'riadelhajjaji@gmail.com' and a shopping cart icon with '3' items are also visible. On the left, there are four thumbnail images of the printer, with a purple arrow pointing to the second one from the top. The main image shows the printer printing a document. To the right of the printer image, the product title 'Epson L3100 Multi-function Color Printer (Black, Ink Tank)' is displayed. Below it, the original price '11199 DH', discounted price '1176 DH', and a '25% OFF' discount are shown, along with a note 'inclusive of all taxes'. The price '11199 DH' is highlighted in yellow. A green box indicates 'undefined units available'. A yellow button labeled '4.6★' is also present. In the center, a price box displays '11199 DH' in yellow, a quantity selector set to '1', and a 'BUY NOW' button. Purple arrows point from the text 'details du produits' to the product title and from 'prix, acheter ou ajouter au panier' to the price box. At the bottom, there are sections for 'Product details' containing placeholder text and 'Reviews' with a feedback entry from 'riad@gmail.com'.

details du produits

plaer le curseur pour zoomer une partie du produit

Epson L3100 Multi-function Color Printer (Black, Ink Tank)

11199 DH    1176 DH    ( 25% OFF )  
inclusive of all taxes

undefined units available

4.6★

11199 DH

- 1 +

BUY NOW

Product details

Reviews

Give your feedback

riad@gmail.com  
Printer Is Nice.\* Quality Is Awesome. \*

details du  
produits

prix, acheter ou  
ajouter au panier

# Commentaire !!!

## SSR dans la page produits

```
export const getServerSideProps: GetServerSideProps<ProductPageProps> = async () => {
  try {
    // Fetch products from an API or database
    const response = await fetch(
      `${process.env.NODE_ENV === 'production'
        ? process.env.NEXT_PUBLIC_API_URL
        : 'http://localhost:5005/api'}/products`
    );

    // Handle non-successful HTTP response
    if (!response.ok) {
      throw new Error('Failed to fetch products');
    }
    const products: Product[] = await response.json();
    return {
      props: {
        products,
      },
    };
  } catch (error) {
    console.error('Error fetching products:', error);
    return {
      props: {
        products: [], // Return an empty array or any default value for products
      },
    };
  }
};
```

## SSR dans la page d'un produit

```
export const getServerSideProps: GetServerSideProps<ProductPageProps> = async (context: GetServerSidePropsContext) => {
  const { productId } = context.params as { productId: string };
  try {
    // Fetch products from an API or database
    const response = await fetch(
      `${process.env.NODE_ENV === 'production'
        ? process.env.NEXT_PUBLIC_API_URL
        : 'http://localhost:5005/api'}/products/${productId}`
    );

    // Handle non-successful HTTP response
    if (!response.ok) {
      throw new Error('Failed to fetch products');
    }

    const product: Product = await response.json();

    return {
      props: {
        product,
      },
    };
  } catch (error) {
    console.error('Error fetching products:', error);
    return {
      props: {
        product: {} as Product, // Return an empty array or any default value for products
      },
    };
  }
};
```

Dans la page products (src/pages/produit/index.tsx) serait préférable d'utiliser static site generation (SSG) de même dans la page de détails de produit (/page/product/[productId]) on devait utiliser SSG et Incremental Static Regeneration (ISR), au lieu de server side rendering (SSR). Mais cela suppose que lors du build du **Front End** les autres microservices doivent être en exécutions pour générer les pages des produits sinon le build génère des erreurs ,ce qui n'est possible lors d'exécution du pipeline parce que les autres microservices ne sont pas encore déployés, l'existence d'une solution à ce problème est sûre mais on avait pas le temps pour la chercher et l'implémenter , Pourtant, Au cas où on a trouvé la solution les méthodes pour faire le SSG et ISR dans les deux cas seraient comme dans les écrans suivants .

# Commentaire !!!!

## SSG dans le details d'un produits

```
export const getStaticPaths: GetStaticPaths = async () => {
  try {
    // Call an external API endpoint to get products
    const res = await fetch(`${process.env.NEXT_PUBLIC_API_URL}/products`);
    if (!res.ok) {
      throw new Error("Failed to fetch products");
    }
    const products: Product[] = await res.json();
    // Get the paths we want to pre-render based on products
    const paths = products.map((product) => ({
      params: { productId: product._id },
    }));
    return { paths, fallback: false };
  } catch (error) {
    // Handle the error gracefully, e.g., log the error or return a fallback value
    console.error("Error in getStaticPaths:", error);
    return { paths: [], fallback: false };
  }
};

export const getStaticProps: GetStaticProps = async ({  
  params,  
}: GetStaticPropsContext) => {  
  const productId = params?.productId;  
  const res = await fetch(  
    `${process.env.NEXT_PUBLIC_API_URL}/products/${productId}`);  
  const product: Product = await res.json();  
  return {  
    props: {  
      product,  
    },  
  };  
};
```

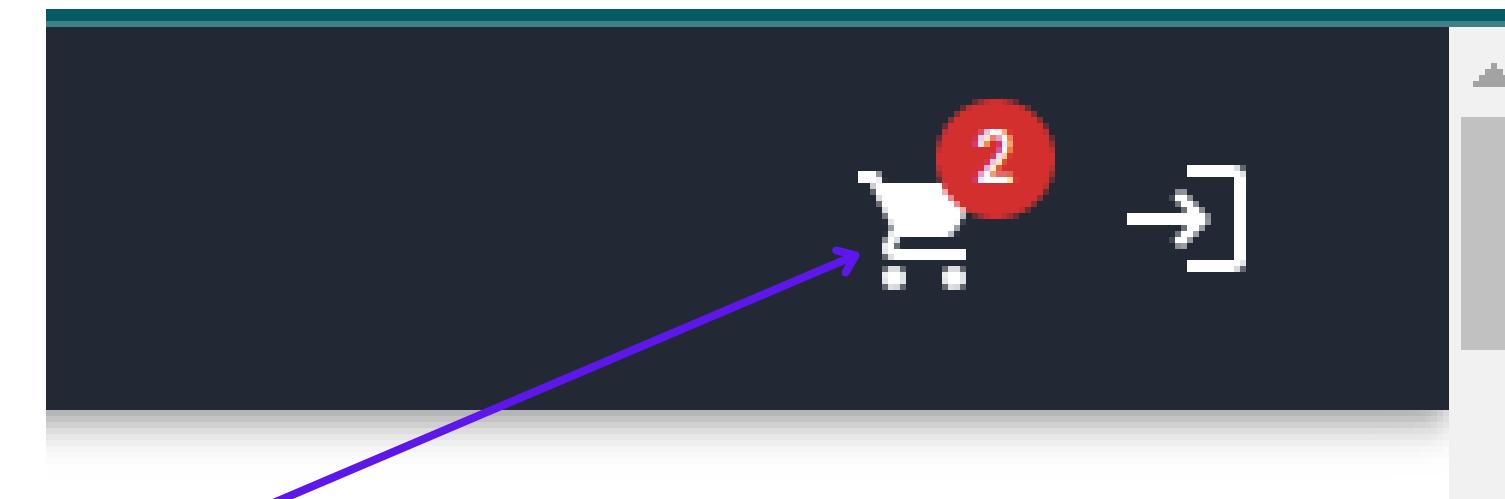
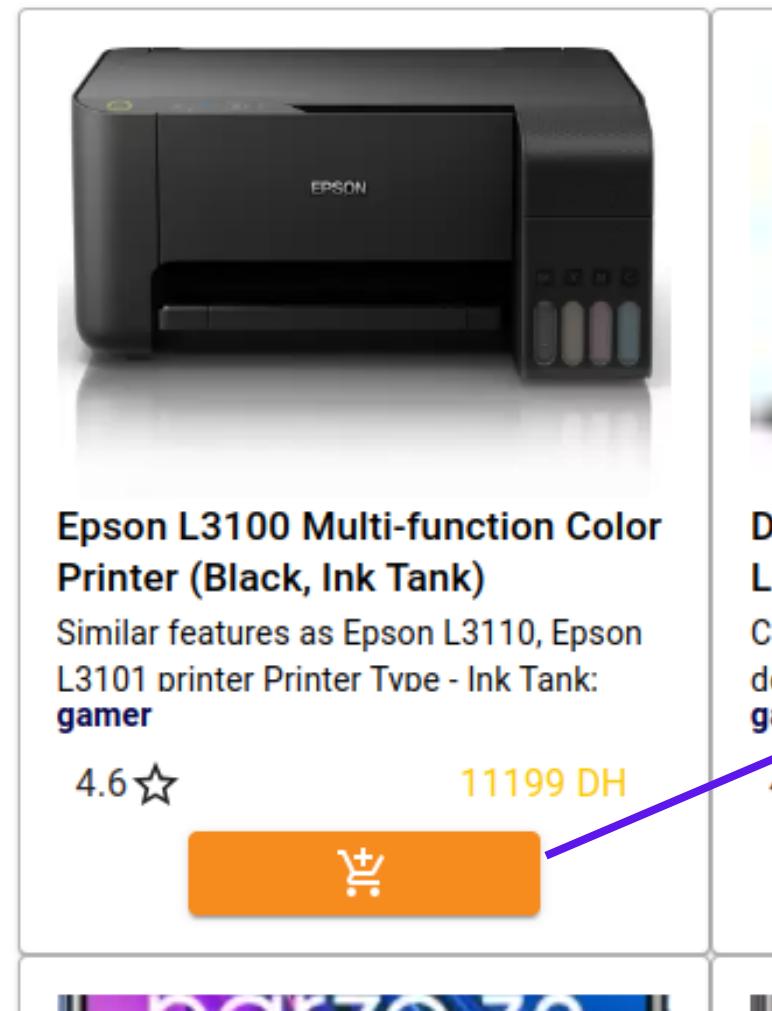
## SSG dans les produits

```
export const GetStaticProps: GetStaticProps<  
  ProductPageProps  
> = async () => {  
  try {  
    // Fetch products from an API or database  
    const response = await fetch(  
      `${  
        process.env.NODE_ENV === 'production'  
        ? process.env.NEXT_PUBLIC_API_URL  
        : 'http://localhost:5005/api'  
      }/products`  
    );  
  
    // Handle non-successful HTTP response  
    if (!response.ok) {  
      throw new Error('Failed to fetch products');  
    }  
  
    const products: Product[] = await response.json();  
  
    return {  
      props: {  
        products,  
      },  
    };  
  } catch (error) {  
    console.error('Error fetching products:', error);  
    return {  
      props: {  
        products: [], // Return an empty array or any default value for products  
      },  
    };  
  }  
};
```

avec les fonctions `getStaticPaths` et  
`getStaticProps`

avec la fonction `getStaticProps`

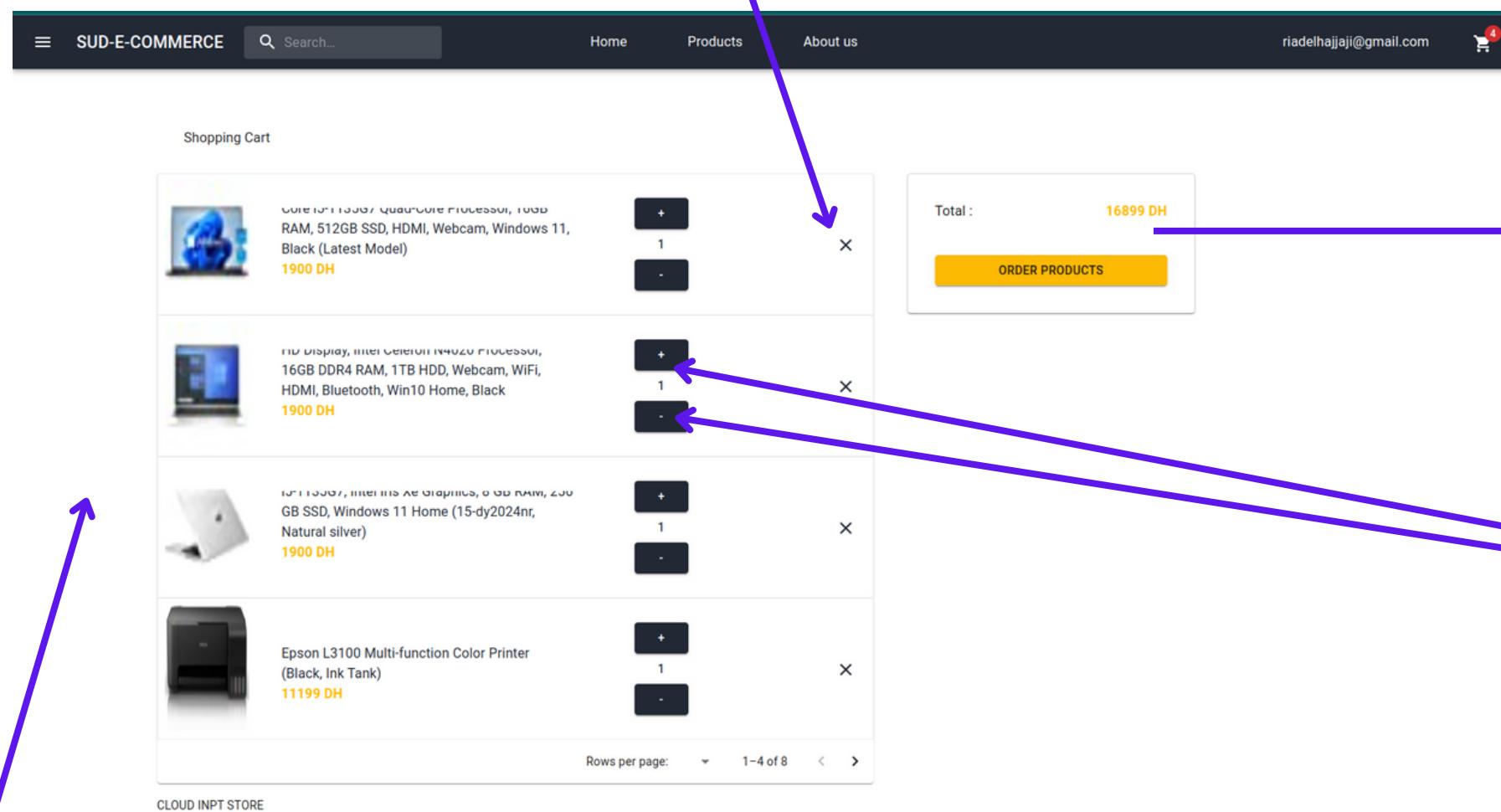
# product card



Ajout d'un produit au panier ,ceci se fait par un dispatch de Redux

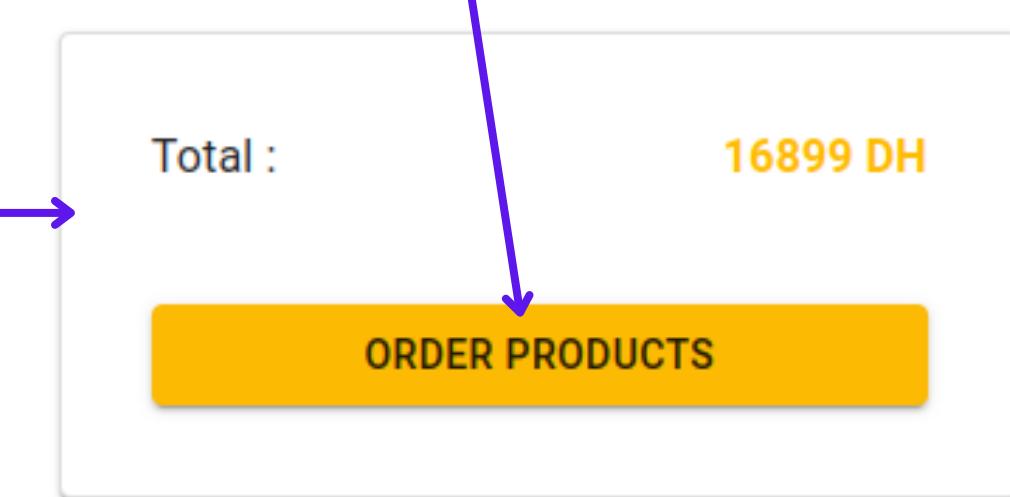
# panier

## Supprimer un produit du panier



produits ajoutés dans le panier

## passer une commande



Augmenter ou diminuer le nombre d'unité d'un produit

# payment

informations de la carte bancaire (pas réelles)

Customer :  
Full Name:  
Username: riadelhajjaji@gmail.com  
Order: 647dbe04172ad82084da13c8

Total à payer :  
**16899 DH**

Credit Card Number  
1234 4234 3435 2342

Credit Owner

PAY

SUD CLOUD INPT 2023.

numero de la commande

Customer :  
Full Name:  
Username: riadelhajjaji@gmail.com  
Order: 647dbe04172ad82084da13c8

Transaction Number : 647dbe250c32e27cc13f1837

Payment Successed !

SUD CLOUD INPT 2023.

payment avec succès.

payment en cours ....

Customer :  
Full Name:  
Username: riadelhajjaji@gmail.com  
Order: 647dbe04172ad82084da13c8

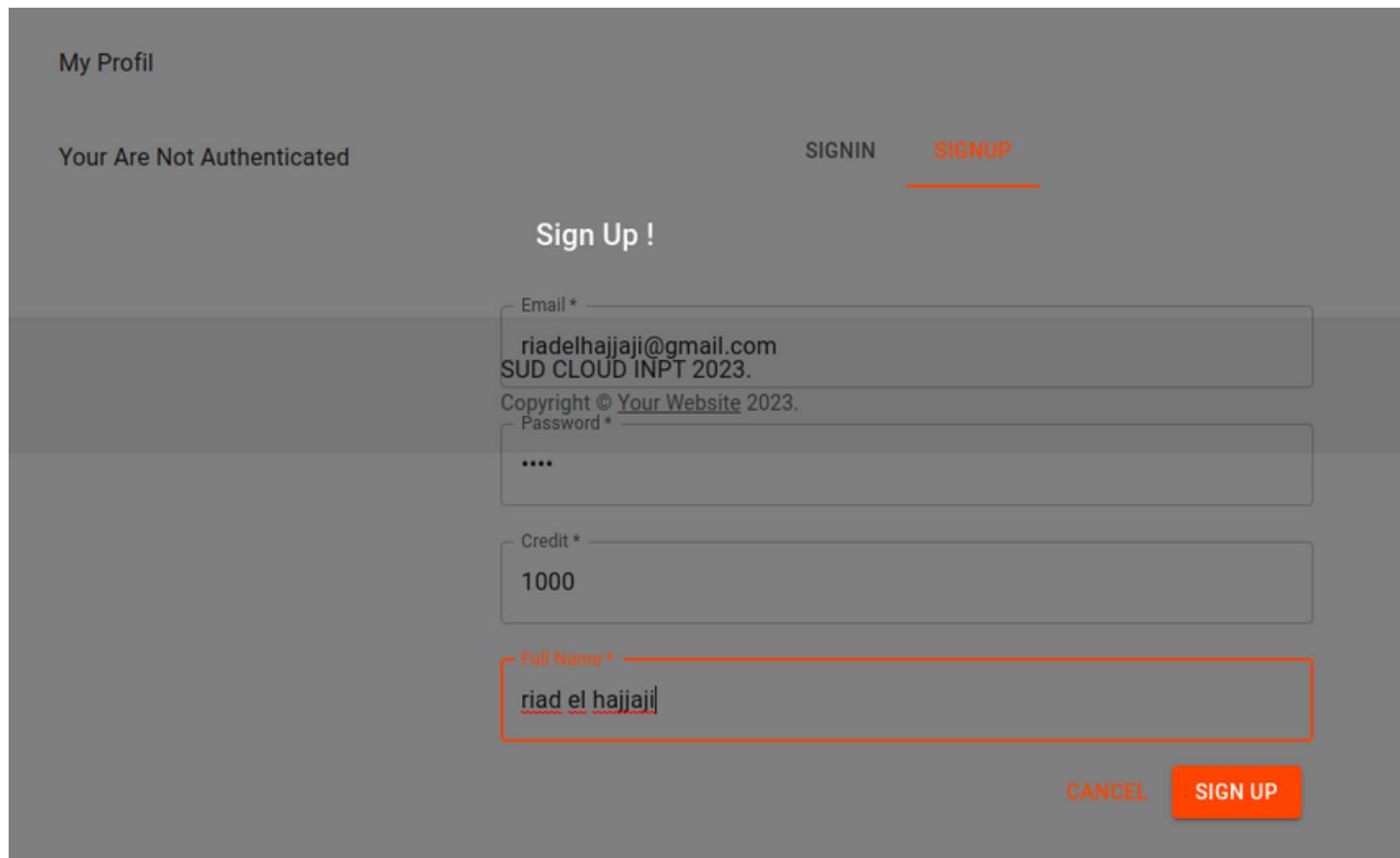
Transaction Number : 647dbe250c32e27cc13f1837

SUD CLOUD INPT 2023.  
Copyright © Your Website 2023.

numero de la transaction

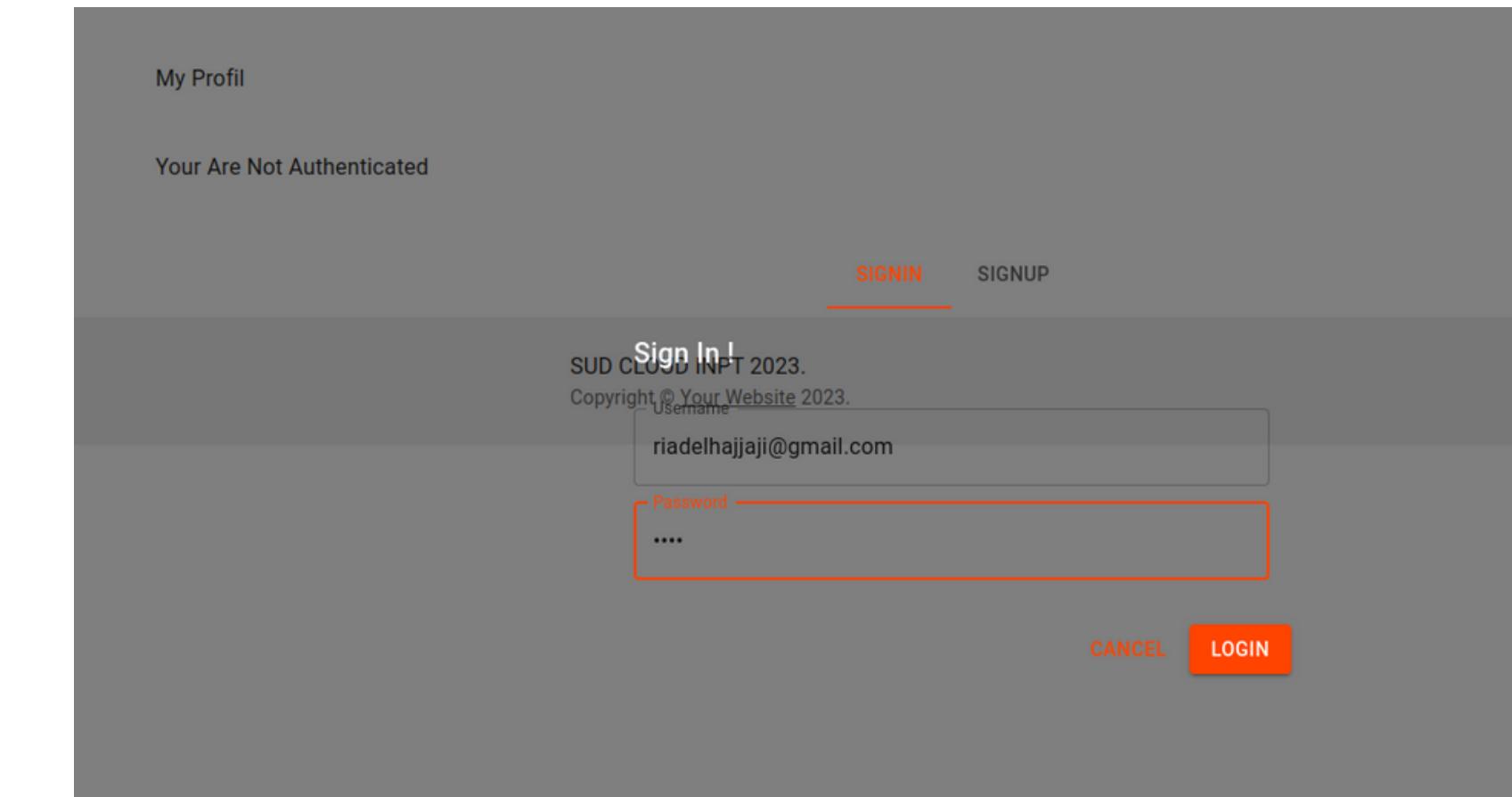
# Sign Up et Sign In

Sign Up se fait par une validation front end et back end , les champs sont email, password, credit et nom complet



The screenshot shows a "Sign Up" form. At the top left is a "My Profil" section with a blue circular icon. Below it, a message says "Your Are Not Authenticated". There are two buttons: "SIGNIN" (grey) and "SIGNUP" (orange, underlined). The main form area has a title "Sign Up !". It contains four input fields: "Email \*" with value "riadelhajjaji@gmail.com", "SUD CLOUD INPT 2023.", "Copyright © Your Website 2023.", "Password \*" with value "\*\*\*\*"; "Credit \*" with value "1000"; and "Full Name \*" with value "riad el hajjaji". At the bottom are "CANCEL" and "SIGN UP" buttons.

Sign In se fait par une validation front end et back end , les champs sont email, password



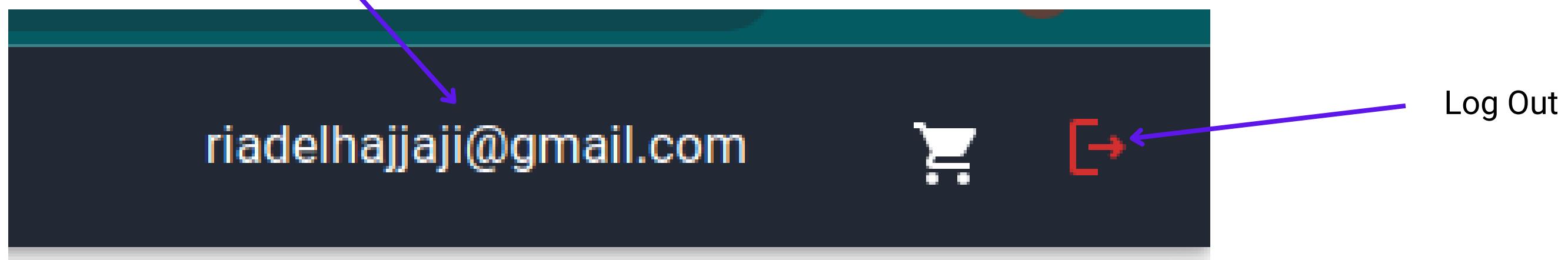
The screenshot shows a "Sign In" form. At the top left is a "My Profil" section with a blue circular icon. Below it, a message says "Your Are Not Authenticated". There are two buttons: "SIGNIN" (orange, underlined) and "SIGNUP" (grey). The main form area has a title "Sign In !". It contains three input fields: "Username" with value "riadelhajjaji@gmail.com", "Password" with value "\*\*\*\*", and "Credit" (disabled). At the bottom are "CANCEL" and "LOGIN" buttons.

les composants n'est pas clairs car y avait un probleme dans le theme MUI !!

- L'authentification est fait avec **JWT** .
- Pour faire le payement on doit s'authentifier .
- Le champ **Email** doit contenir l'email au on va recevoir l'email de succes du payment de la part de microservice **emailing**.

# profil d'utilisateur

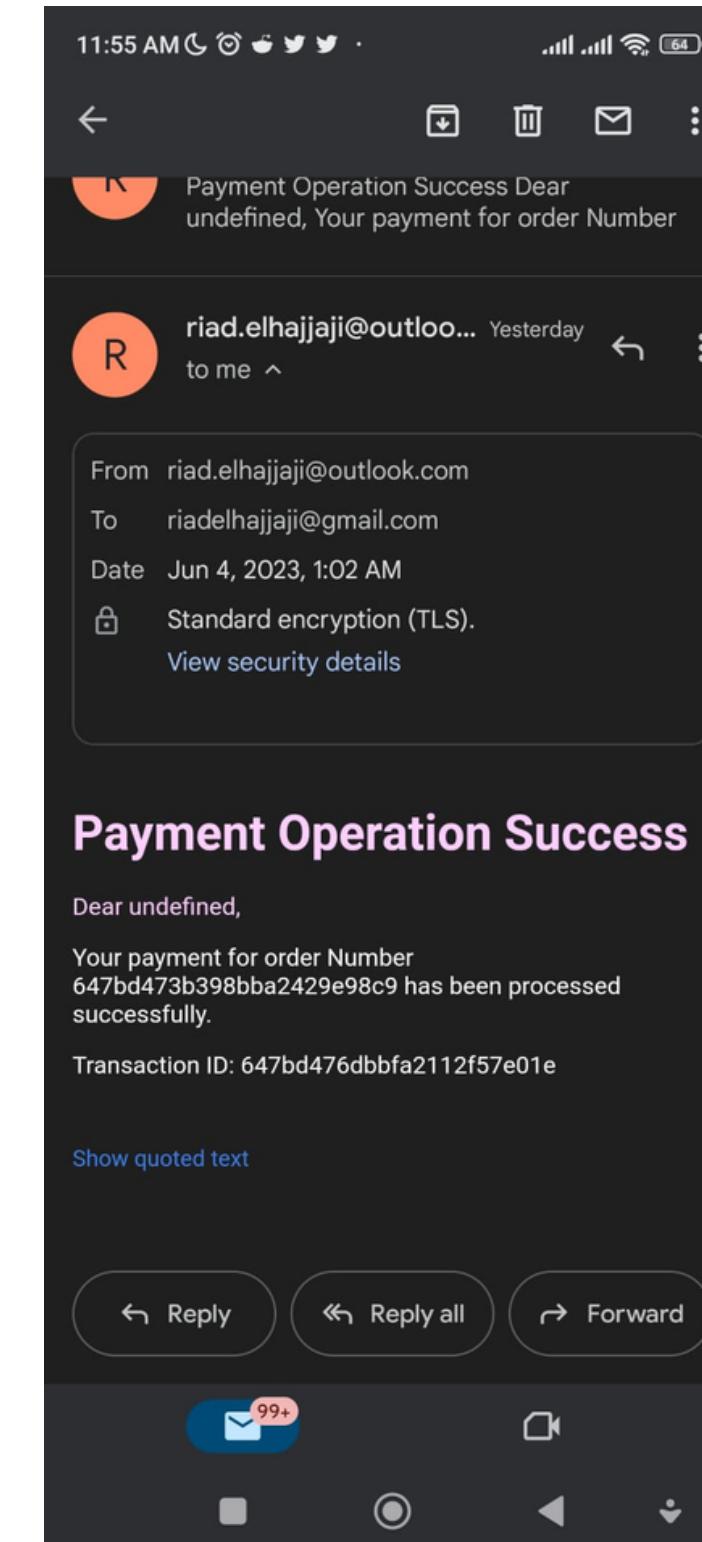
le username qui est déjà authentifié



- Le profil de l'utilisateur et le token sont stockés avec les cookies .
- Le token est valide pour 2h après l'authentification.

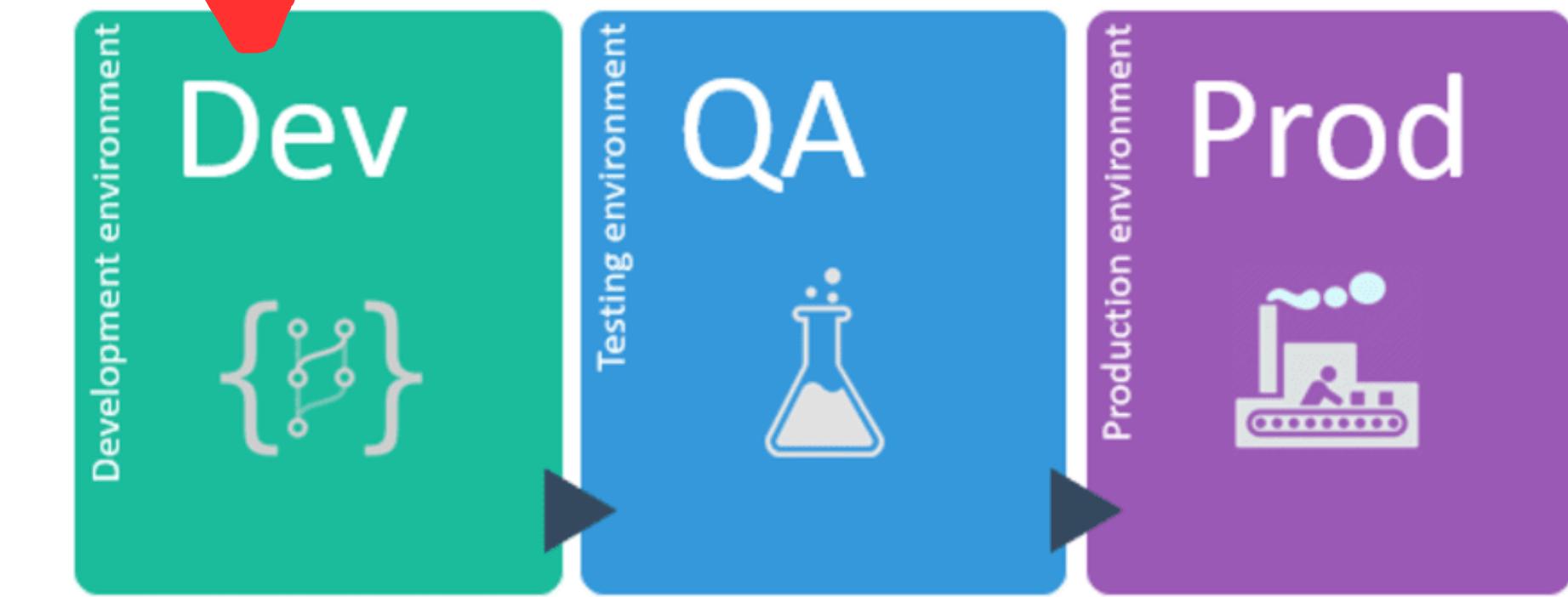
# Informer le client du payment fait

- Le microservice **emailing** utilise **nodeMailer** pour envoyer l'email.
- On a utiliser **outlouk** car **gmail** ne marche pas toujours
- Username et password d'outlook sont des secrets dans kubernetes
- Apres le payment le service emailing envoyer une requeute **PUT** qui modifie le champ **orderPayed** de **false** a **true** pour marquer la commande comme payee.



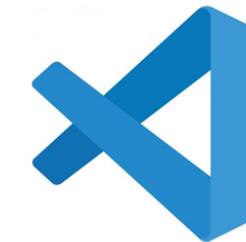
le wifi de l'INPT bloque l'envoie de l'email !!

# Environnement et Outils de Development

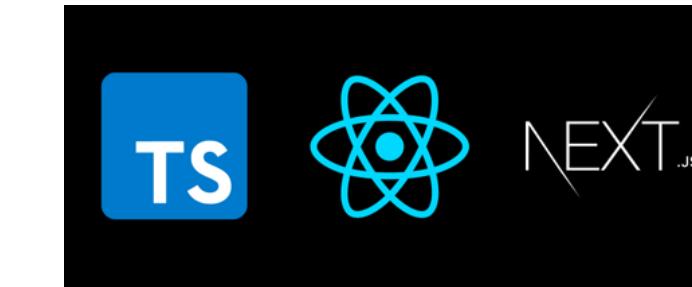


# Frameworks et Outils de Development

- Editeur : Vs code



- Front-End: NextJs, ReactJS, Typescript , Material UI

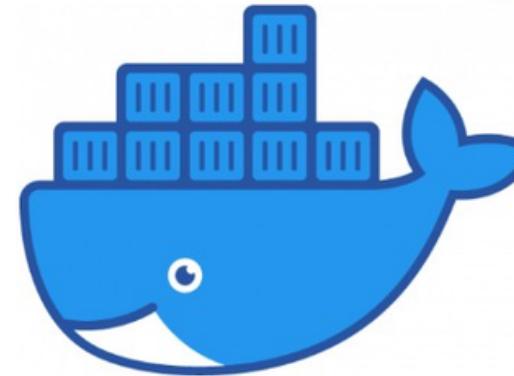


- Backend-End: Express ,Typescript



# Frameworks et Outils de Development

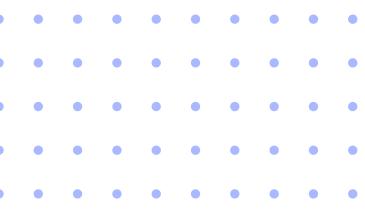
- **MongoDb container : pour simuler MongoDB**



- **RabbitMq Container : pour simuler rabbitMq**



- **Postman : pour tester**



## Choix des Frameworks et Outils

### Next.js

Next.js est préféré à React pour les sites Web de commerce électronique car il offre un rendu côté serveur (SSR) intégré et une génération de site statique (SSG), améliorant les performances et le référencement. Il simplifie le routage et la navigation, fournit des itinéraires d'API pour la logique côté serveur et améliore la productivité des développeurs grâce à des fonctionnalités telles que le rechargement de modules à chaud et le fractionnement de code.

### Material UI

Material-UI is ideal for e-commerce apps, offering a wide range of components and a cohesive design system. With its ready-to-use components, responsive design, and customization options, developers can create visually appealing and user-friendly interfaces for product listings, shopping carts, and checkout processes. It enables the development of modern and engaging e-commerce experiences that provide a seamless shopping journey for customers.

# Choix des Frameworks et Outils

## TypeScript

TypeScript is widely adopted in the development community due to its numerous advantages. It enhances the JavaScript programming language by adding static typing, which improves code quality and reduces bugs during development. TypeScript offers better tooling and editor support, enabling developers to catch errors early and write more reliable code. It also provides improved documentation and code readability, making it easier for teams to collaborate on projects. TypeScript's compatibility with existing JavaScript codebases and seamless integration with popular frameworks and libraries make it a preferred choice for building robust and scalable applications across various domains and industries.

## Redux

Redux is a popular JavaScript library used for managing the state of an application. It provides a predictable state container that helps to organize and manage the state in a consistent way across different components of an application. Its features are :

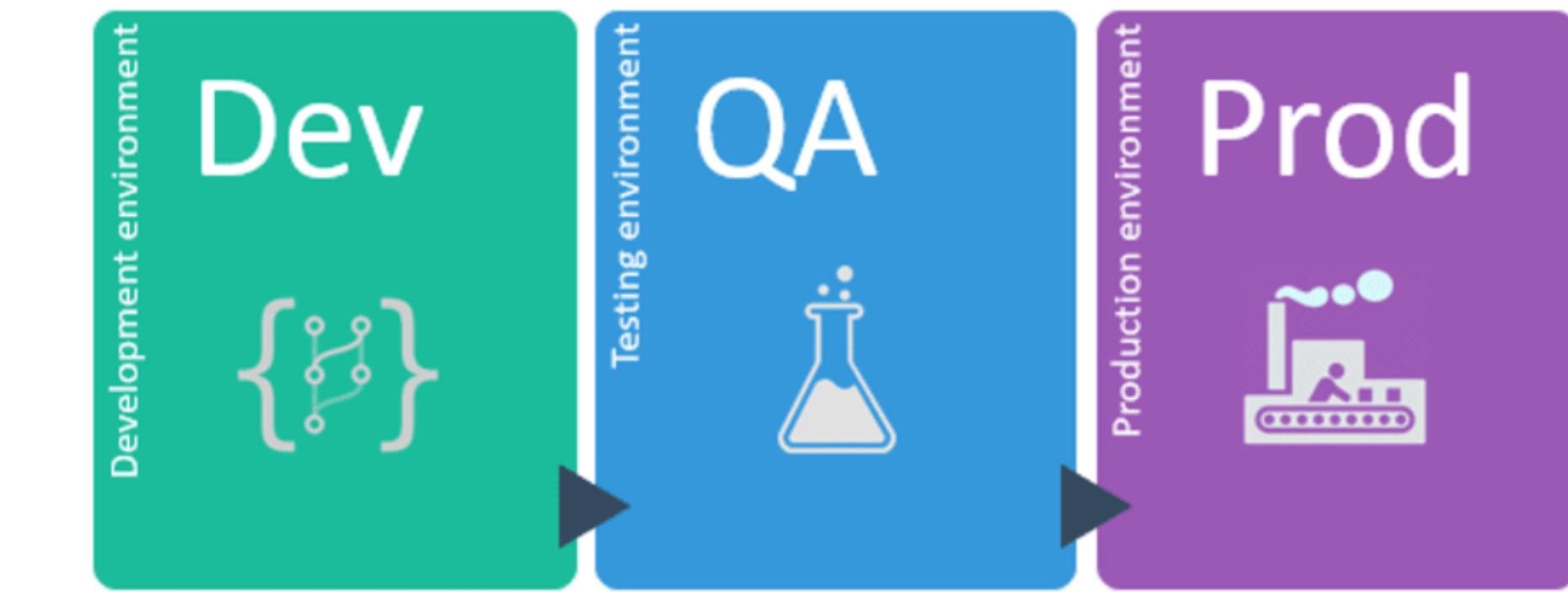
- 1. Centralized State**
- 3. Ease of Debugging**
- 5. Ecosystem and Community Support**
- 2. Management Predictable State Updates**
- 4. Scalability**
- 6. Testability**

# Choix des Outils de Development

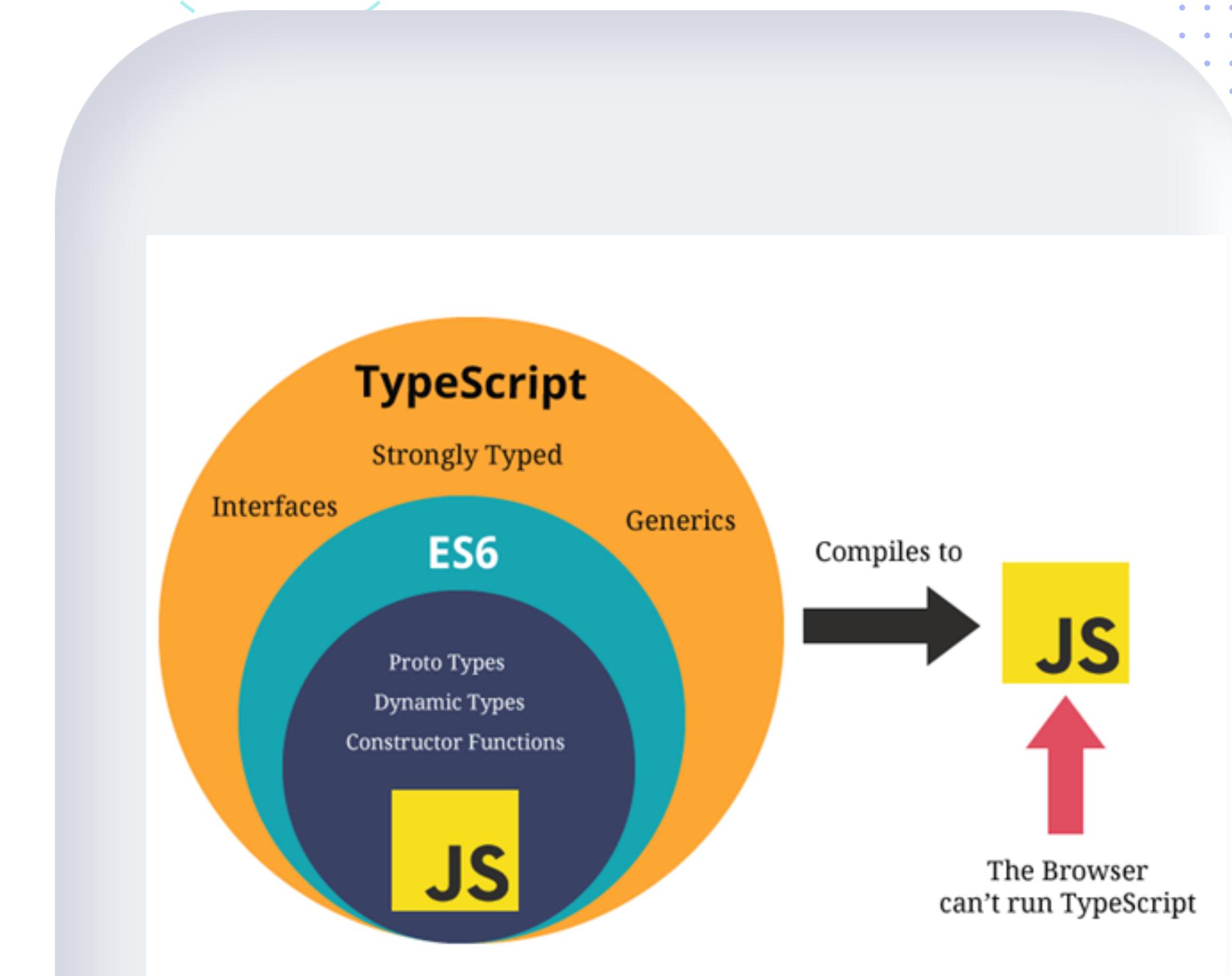
## Node JS

- Event-driven, non-blocking I/O: Node.js follows an event-driven, non-blocking I/O model, which means it can handle a large number of concurrent connections without blocking the execution of other operations. This makes it highly scalable and efficient, particularly for applications that involve heavy I/O operations, such as e-commerce apps
- NPM ecosystem: Node.js has a vast ecosystem of open-source packages available through the Node Package Manager (NPM). NPM provides a repository of reusable libraries and modules that can be easily integrated into your applications. This accelerates development by allowing developers to leverage existing solutions rather than building everything from scratch.
- Microservices architecture: Node.js is well-suited for building microservices-based architectures. Its lightweight and modular nature enables you to create small, independent services that can be easily scaled and deployed. Node.js also supports asynchronous programming, which aligns well with the event-driven nature of microservices.
- DevOps tooling: Node.js is supported by various DevOps tools and frameworks, making it easier to automate deployment, monitoring, and management processes. Tools like Docker, Kubernetes, and CI/CD pipelines can be used seamlessly with Node.js applications to streamline the development and deployment workflow.

# Code Quality



# Types Checking



# TYPESCRIPT

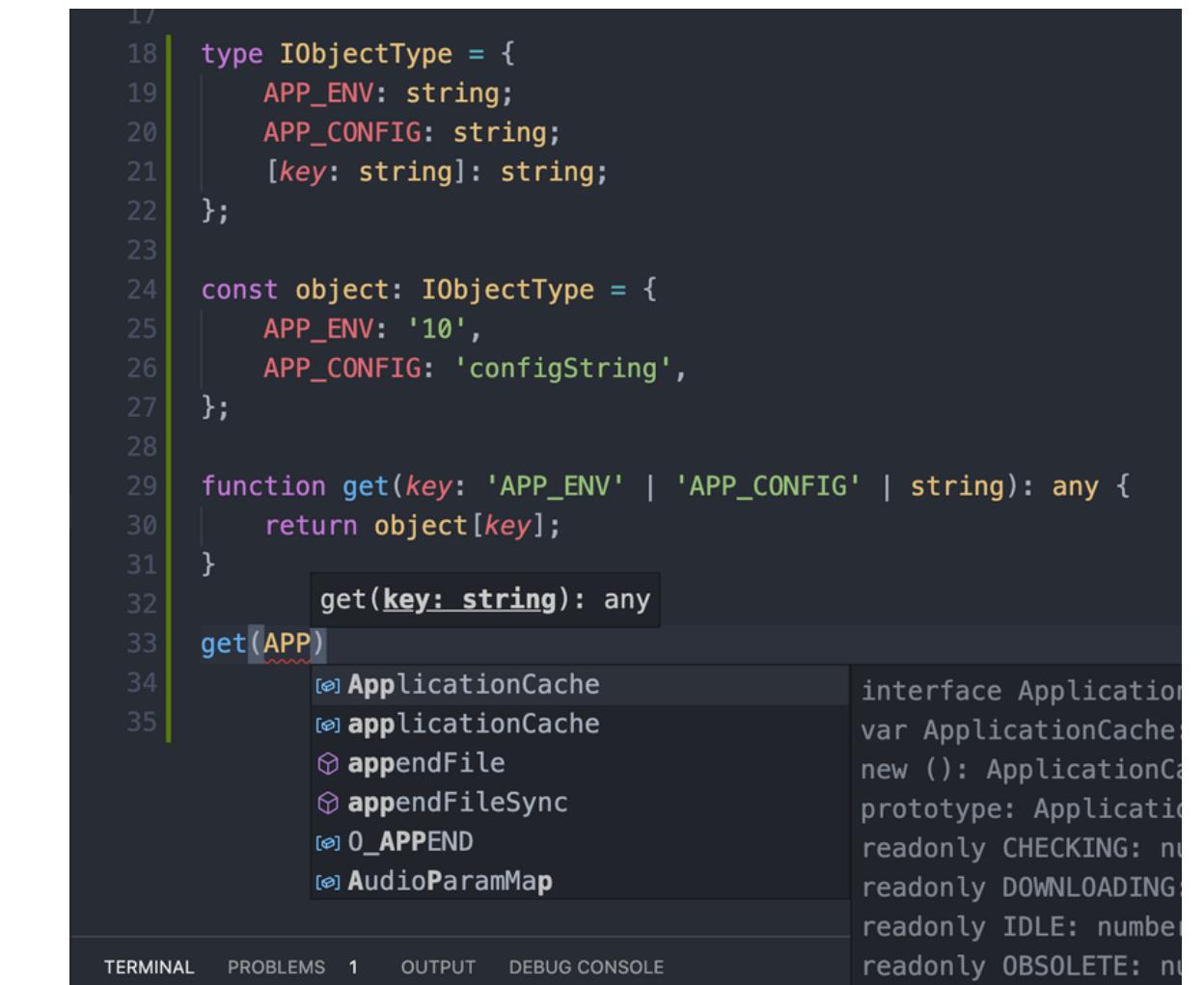
TypeScript utilise la vérification du type au moment de la compilation. Ce qui signifie qu'il vérifie si les types spécifiés correspondent avant d'exécuter le code, et non pendant l'exécution du code.

## vérification du type

```
const parsed = parseEmailAddress("example.com");

if (parsed.success) {
    // Here, parsed has the following type:
    // { success: true; value: string}
    parsed.value; // OK
    parsed.error; // Error
} else {
    // Here, parsed has the following type:
    // { success: false; error: string}
    parsed.value; // Error
    parsed.error; // OK
}
```

## autocomplete

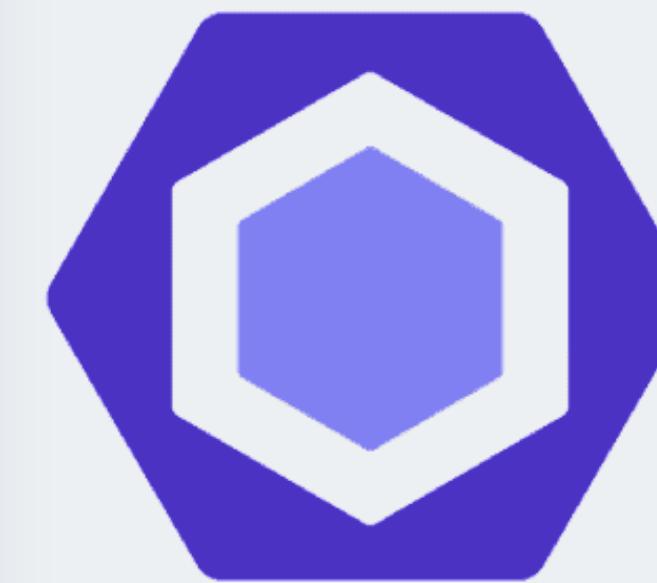


A screenshot of a code editor showing a TypeScript file. The code defines a type `IObjectType` and a function `get`. The `get` function takes a key of type `'APP_ENV' | 'APP_CONFIG' | string` and returns a value of type `any`. Below the code, there is an `ApplicationCache` interface definition. An autocomplete dropdown is open over the `get` call, showing suggestions like `ApplicationCache`, `applicationCache`, `appendFile`, `appendFileSync`, `O_APPEND`, and `AudioParamMap`.

```
17
18 type IObjectType = {
19     APP_ENV: string;
20     APP_CONFIG: string;
21     [key: string]: string;
22 };
23
24 const object: IObjectType = {
25     APP_ENV: '10',
26     APP_CONFIG: 'configString',
27 };
28
29 function get(key: 'APP_ENV' | 'APP_CONFIG' | string): any {
30     return object[key];
31 }
32     get(key: string): any
33 get(APP)
34     [e] ApplicationCache
35     [e] applicationCache
      appendFile
      appendFileSync
      [e] O_APPEND
      [e] AudioParamMap
interface ApplicationCache {
    var ApplicationCache;
    new (): ApplicationCache;
    prototype: ApplicationCache;
    readonly CHECKING: number;
    readonly DOWNLOADING: number;
    readonly IDLE: number;
    readonly OBSOLETE: number;
}
```

TERMINAL PROBLEMS 1 OUTPUT DEBUG CONSOLE

# Code Linting



# ESLint

# LINTING

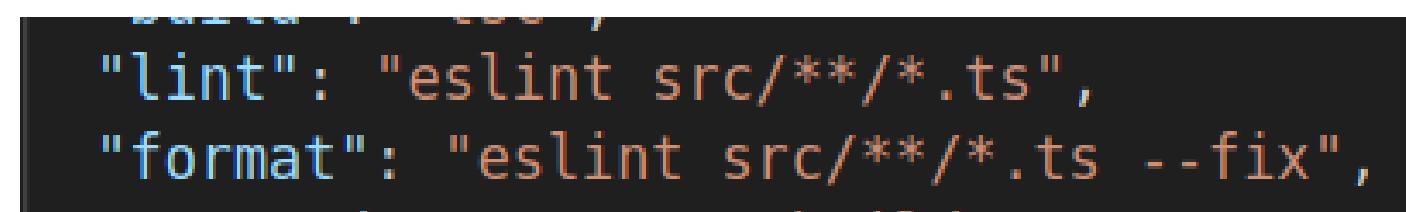


▶ ESLint Improve code quality, catch errors, enforce coding standards

## Fichier de config de Eslint dans la racine du projet



## Scripts de package.json pour utiliser eslint



Le fichier contient Les regles (rules) de linting de Front-End

# LINTING

Le fichier contient Les regles (rules) de linting de Back-End

```
"rules": {
    "no-console": "warn",
    "no-debugger": "error",
    "@typescript-eslint/strict-boolean-expressions": "error",
    "@typescript-eslint/prefer_READONLY": "error",
    "@typescript-eslint/explicit-function-return-type": [
        "error",
        { "allowExpressions": true }
    ],
    "@typescript-eslint/consistent-type-definitions": [ "error", "interface" ],
    "@typescript-eslint/no-unused-vars": [
        "error",
        { "argsIgnorePattern": "^_" }
    ],
    "@typescript-eslint/await-thenable": "error",
    "@typescript-eslint/no-explicit-any": "error",
    "@typescript-eslint/no-floating-promises": "error",
    "@typescript-eslint/no-misused-promises": "error",
    "@typescript-eslint/promise-function-async": "error",
    "import/extensions": [
        "error",
        "ignorePackages",
        {
            "js": "never",
            "ts": "never"
        }
    ],
    "import/no-extraneous-dependencies": [
        "error",
        {
            "optionalDependencies": true
        }
    ]
},
```

Utiliser les commandes définies dans **scripts** pour faire le Linting avec **Eslint**

```
• riad@riad-Inspiron-3501:~/etude/allaki/micsrvce-project/services/mclientui$ npm run lint

> streaming@0.1.0 lint
> next lint

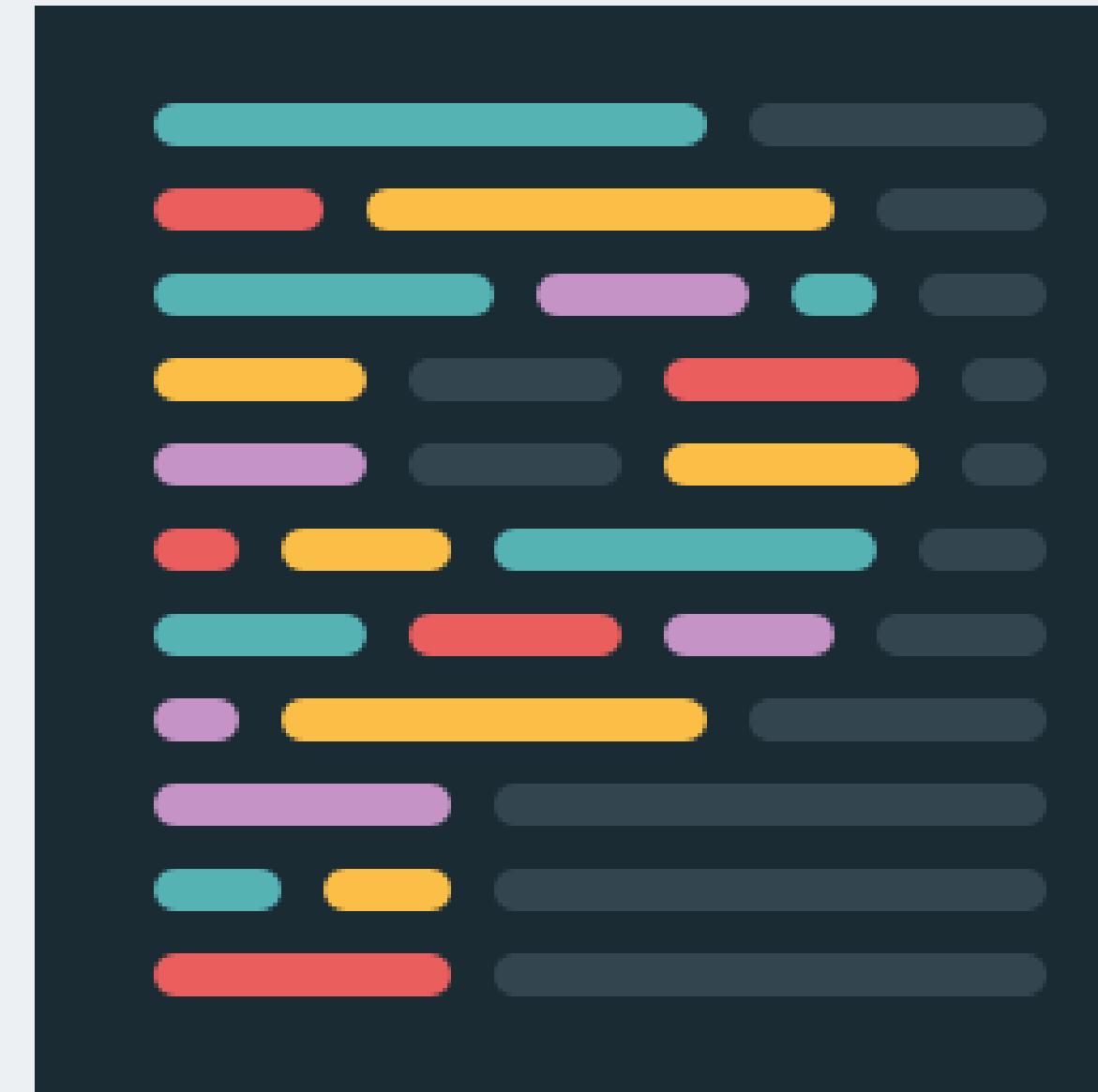
info  - Loaded env from /home/riad/etude/allaki/micsrvce-project/services/mclientui/.env
warn  - The Next.js plugin was not detected in your ESLint configuration. See https://nextjs.org/docs/basic-features/eslint#migrating

./pages/product/[productId].tsx
217:5  Warning: Unexpected console statement.  no-console

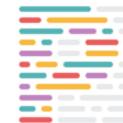
info  - Need to disable some ESLint rules? Learn more here: https://nextjs.org/docs/basic-features/eslint#disabling-rules
```

- dans le screenshot eslint detecte errors et warning qui refletent une violation des regles du syntax

# Code Formatting



# CODE FORMATING



Prettier **Consistent code formatting, enhances code readability, and enforces a unified coding style.**

Fichier de config de Prettier dans la racine du projet

```
└─ .gitignore  
└─ .prettierrc  
└─ Dockerfile
```

le package suivant est pour combiner entre Eslint et Prettier

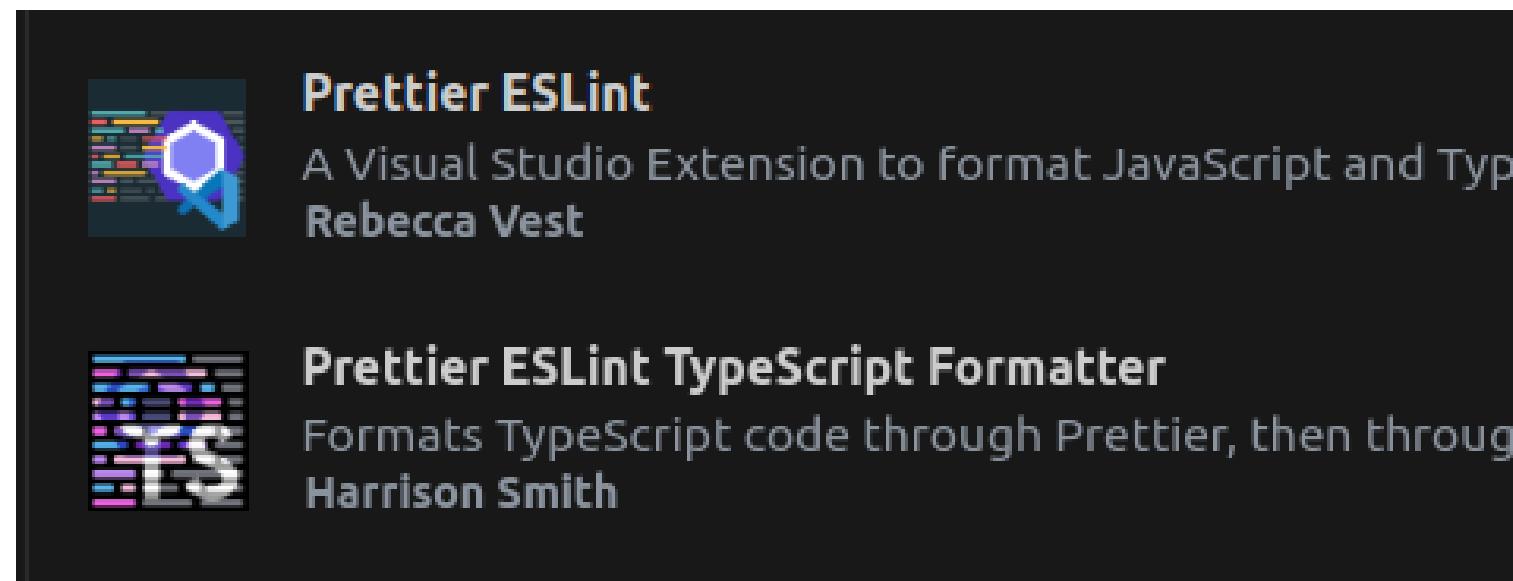
```
"eslint-plugin-prettier": "^4.2.1",
```

Le fichier contient Les regles de formattage

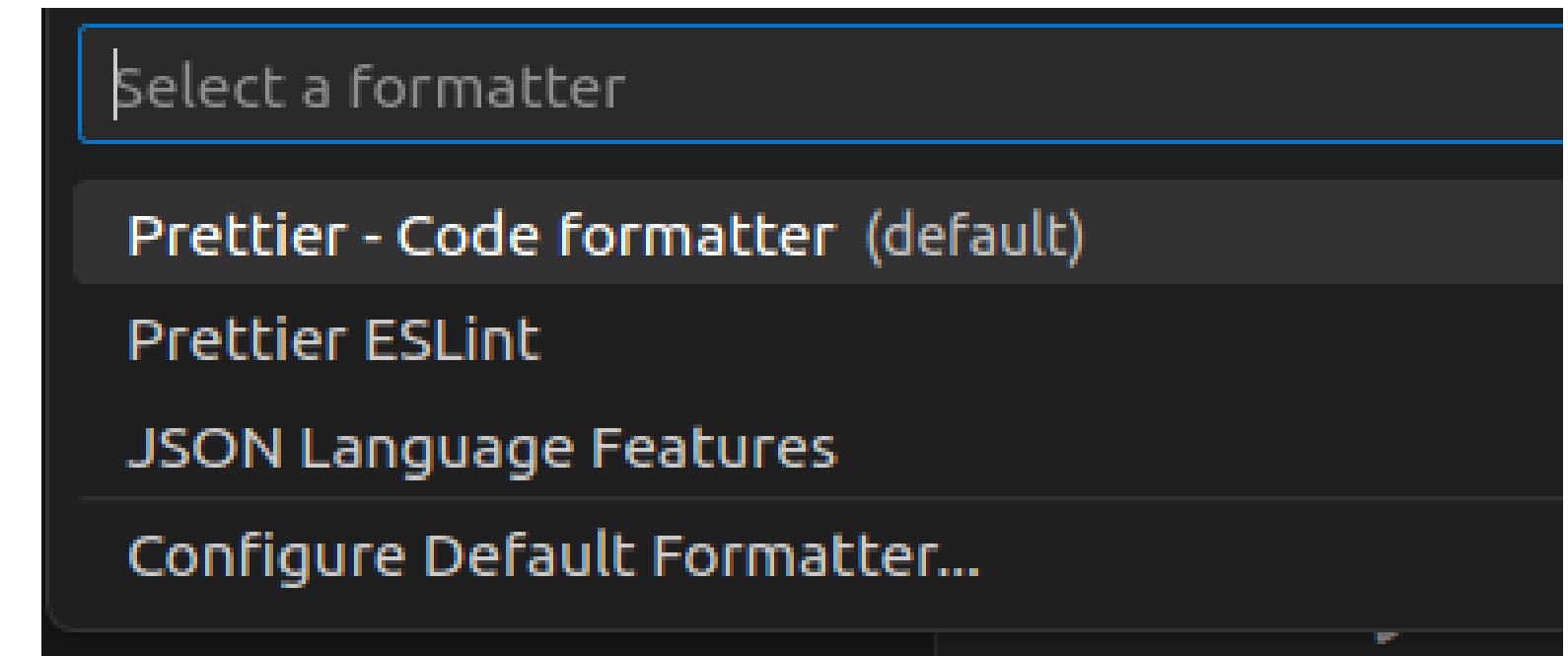
```
You, 19 hours ago | 1 author (You)  
{  
  "endOfLine": "lf",  
  "printWidth": 80,  
  "trailingComma": "es5",  
  "singleQuote": true,  
  "arrowParens": "avoid",  
  "proseWrap": "preserve",  
  "quoteProps": "as-needed",  
  "bracketSameLine": false,  
  "bracketSpacing": true,  
  "tabWidth": 2  
}
```

# CODE FORMATING

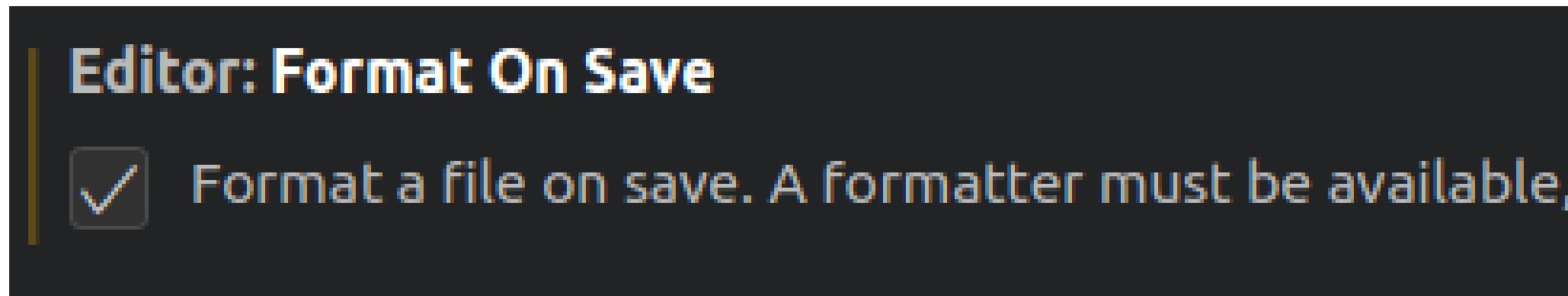
Installer les plugin nécessaire



Configurer Vs Code pour utiliser **Prettier Eslint** comme **Formatter**

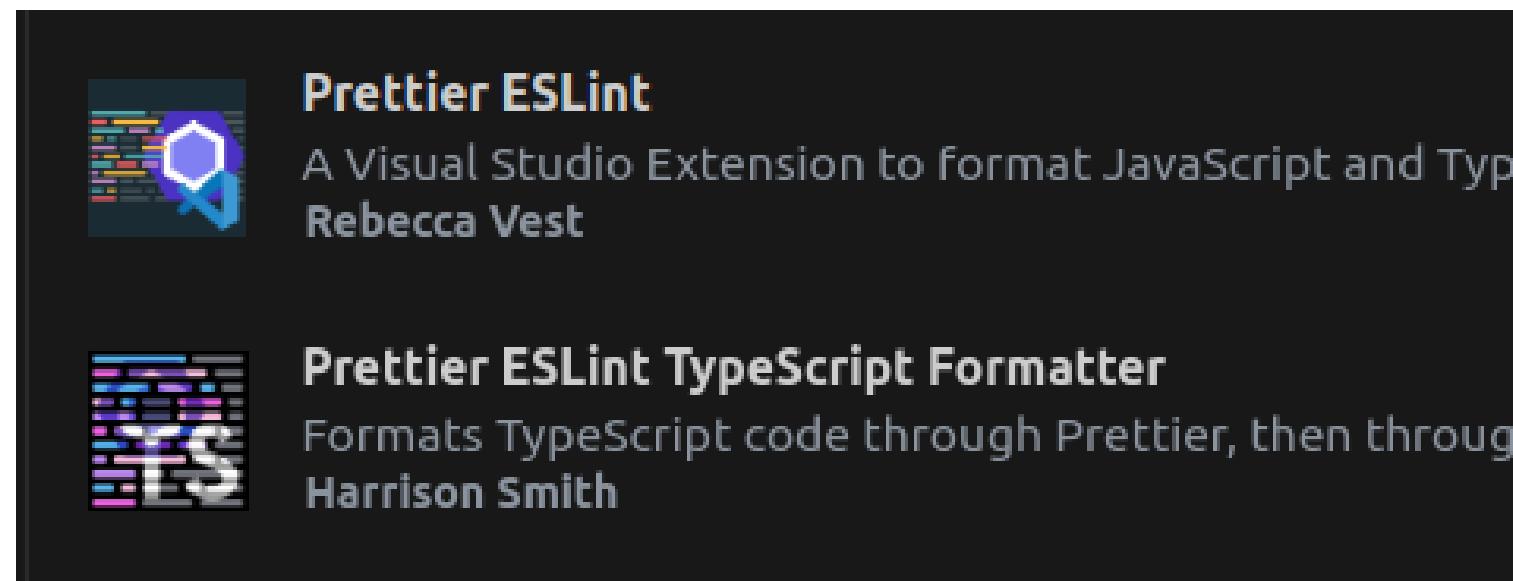


Activer l'option Format On Save pour appliquer le formattage automatiquement apres sauvegarde

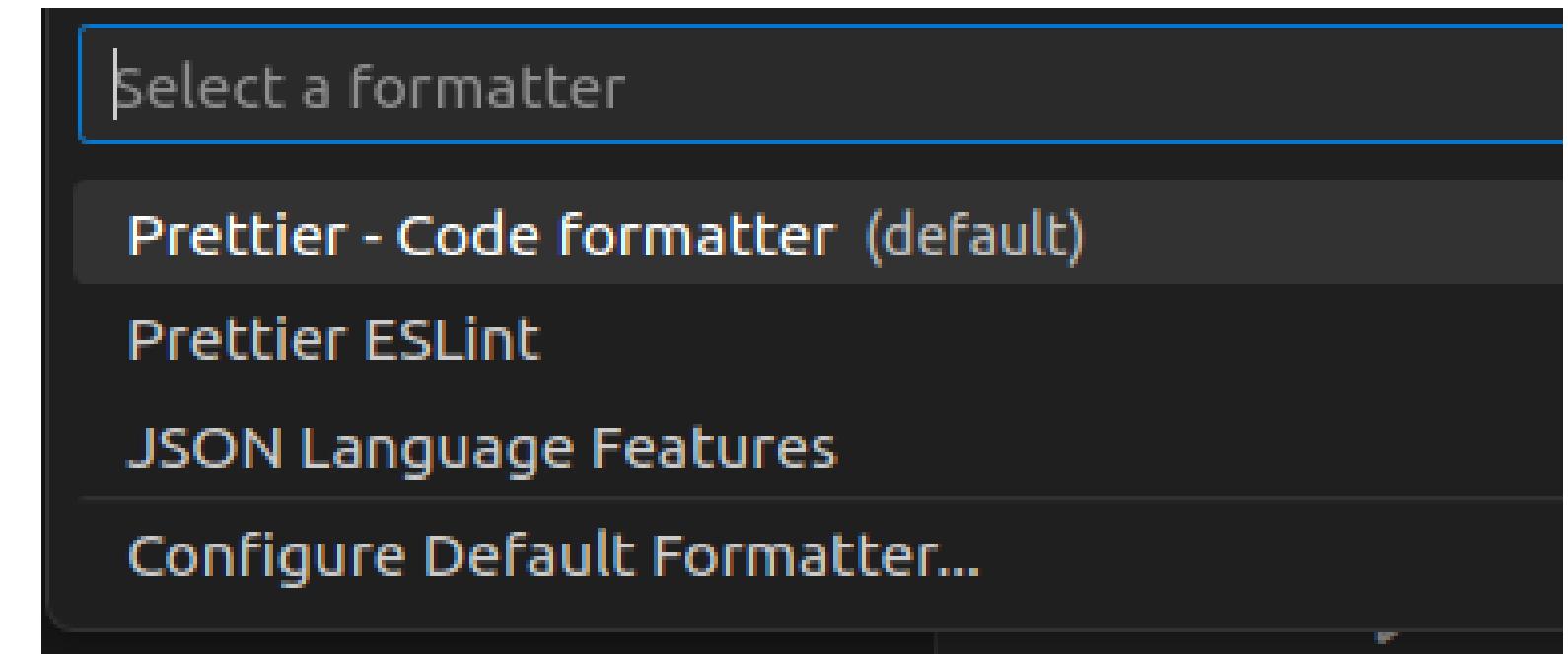


# CODE FORMATING

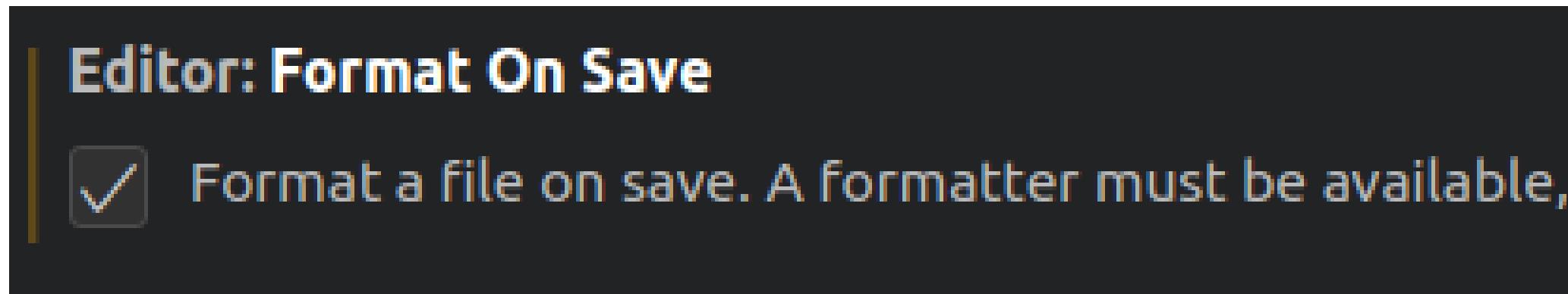
Installer les plugin nécessaire



Configurer Vs Code pour utiliser **Prettier Eslint** comme **Formatter**



Activer l'option Format On Save pour appliquer le formattage automatiquement apres sauvegarde



# CODE FORMATING

**pretty-quick** : Un outil Idéal pour formater un fichier entier sur vos fichiers modifiés (changed) / staged files, d'une maniere automatique pour avoir code consistent.

Installez-le avec husky :

- npx husky-init
- npm install --save-dev pretty-quick
- npx husky set .husky/pre-commit  
"npx pretty-quick --staged"

```
"pretty-quick": "pretty-quick",
"postinstall": "husky install"
```

Formatter le code du project :

```
no changes added to commit (use git add and/or git commit -a)
• riad@riad-Inspiron-3501:~/etude/allaki/micsrvce-project/services/mclientui$ npm run pretty-quick

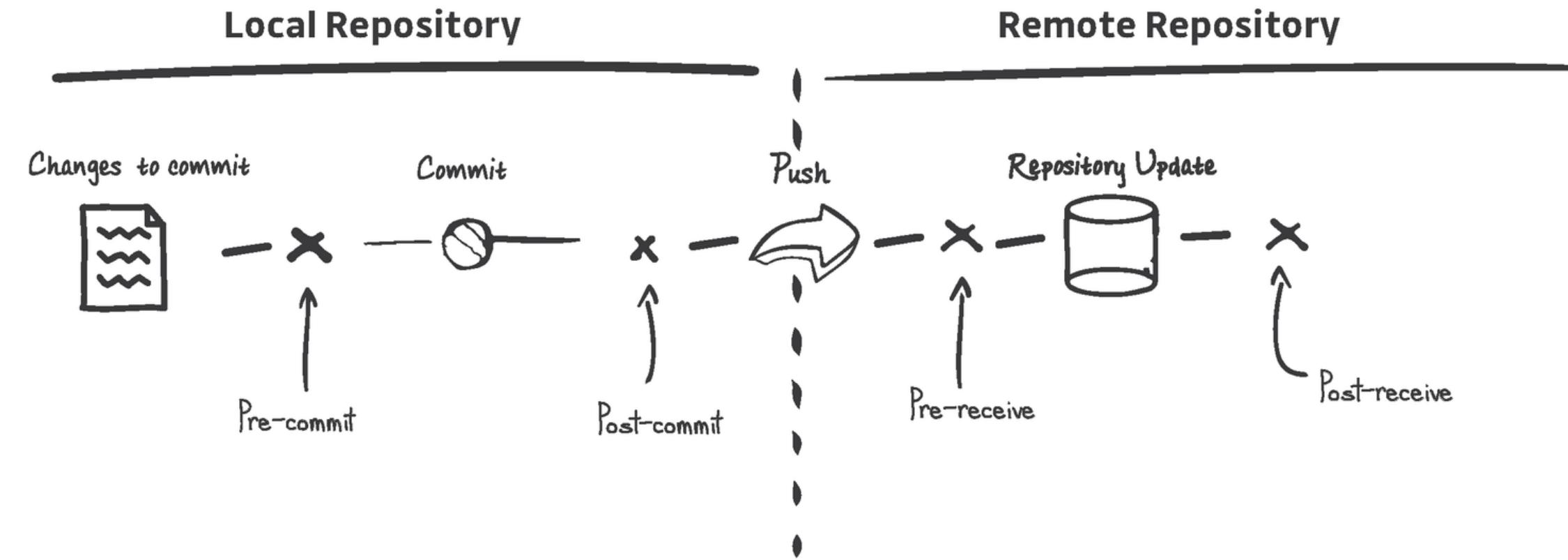
> streaming@0.1.0 pretty-quick
> pretty-quick

🔍 Finding changed files since git revision null.
⌚️ Found 2 changed files.
✅ Everything is awesome!
riad@riad-Inspiron-3501:~/etude/allaki/micsrvce-project/services/mclientui$
```

# Pre-Commit Hooks



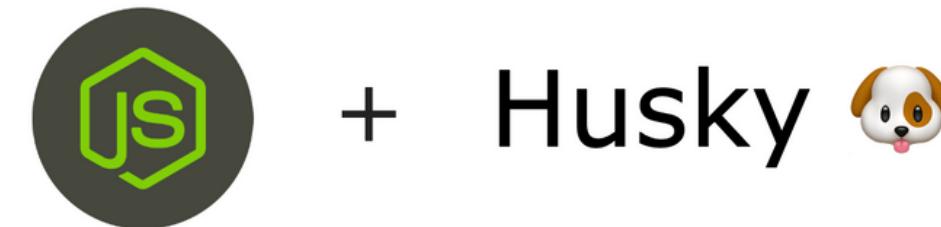
# PRE-COMMIT HOOKS



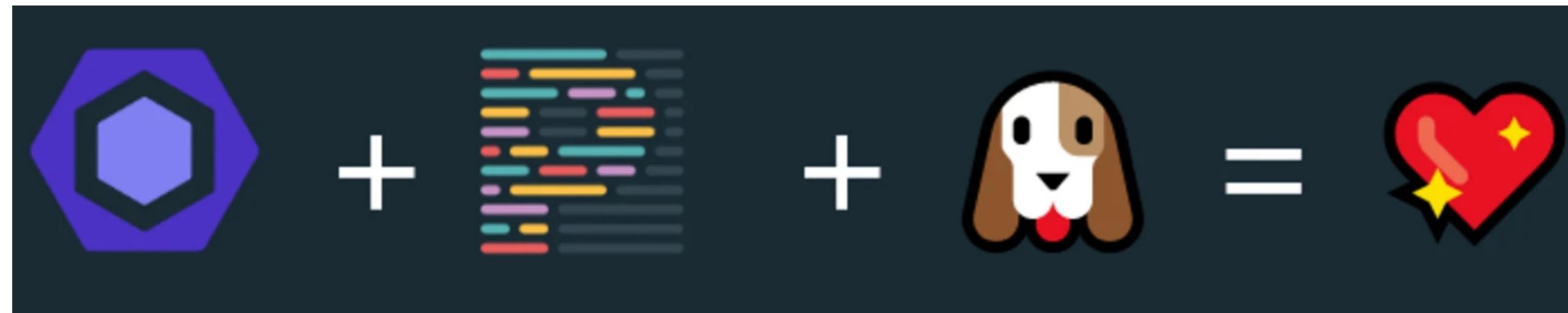
**Hooks de pré-commit :** Scripts automatisés qui s'exécutent avant la validation des modifications de code dans un système de contrôle de version. Ils vous permettent d'appliquer des normes de qualité du code, d'exécuter des tests ou d'effectuer d'autres vérifications pour garantir que le code validé respecte certains critères avant d'être ajouté au référentiel. Les hooks de pré-commit aident à détecter les problèmes potentiels dès le départ et garantissent que seul un code propre, bien formaté et validé est validé.

# PRE-COMMIT HOOKS

Husky est une librairie Javascript permettant de faciliter la création et le partage des hooks au sein d'un projet.



*Husky improves your commits and more dog woof!*



# PRE-COMMIT HOOKS

1. Au debut, On installe husky :

```
npm install husky --save-dev
```

2. On install les Pre-commit par la commande :

```
./node_modules/.bin/husky install
```

```
✓ mclientui
  ✓ .husky
    > _ 
$ pre-commit
  ↵ next
```

3. Dans le fichier **.husky/pre-commit** on ajoute les actions qui vont s'exécuter avant chaque commit.

- On va faire code linting avec **Eslint**, et code formating avec **pretty-quick**

```
#!/usr/bin/env sh
. "$(dirname -- "$0")/_husky.sh"
echo "linting before commit..."
npm run lint
npm run pretty-quick
```

# PRE-COMMIT HOOKS

4. Dans `package.json`, On ajoute cette ligne pour indiquer que le fichier pre-commit va s'executer lorsqu'on essaye de faire un commit

```
  "husky": {  
    "hooks": {  
      "pre-commit": "pretty-quick --staged"  
    }  
  },
```

**--staged flag** : Pre-commit mode. Under this flag only staged files will be formatted, and they will be re-staged after formatting

# PRE-COMMIT HOOKS

## 5. On va violer une regles de Eslint , Puis, On va essayer de commiter.

- Ici on utilise un `console.log` dans le code ce qui n'est une bonne pratique parmis celles definies dans Eslint rules

```
4 import NavigationIcon from '@mui/icons-material/NavigationIcon';
5
6 export default function ExploreproductsButton() {
7   console.log(3);          You, 1 second ago • Uncommitt
8   return (
9     Link to products

```

# PRE-COMMIT HOOKS

6. On remarque que le linting et le formatage du code s'execute avant commit puis il fait exit pour afficher les erreurs de syntax (console.log)

```
info  - Need to disable some ESLint rules? Learn more here: https://nextjs.org/docs/basic-features/eslint#disabling-rules
④ riad@riad-Inspiron-3501:~/etude/allaki/micsrvce-project/services/mclientui$ git commit -m "commit with linting error"
linting before commit...

> streaming@0.1.0 lint
> next lint

info  - Loaded env from /home/riad/etude/allaki/micsrvce-project/services/mclientui/.env
warn  - The Next.js plugin was not detected in your ESLint configuration. See https://nextjs.org/docs/basic-features/eslint#migr
g

./pages/product/[productId].tsx
217:5 Warning: Unexpected console statement.  no-console

info  - Need to disable some ESLint rules? Learn more here: https://nextjs.org/docs/basic-features/eslint#disabling-rules

> streaming@0.1.0 pretty-quick
> pretty-quick

🔍 Finding changed files since git revision null.
⌚️ Found 2 changed files.
✅ Everything is awesome!
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   common/components/homePage/ExploreProductsButton.tsx
    modified:   pages/product/index.tsx

no changes added to commit (use "git add" and/or "git commit -a")
riad@riad-Inspiron-3501:~/etude/allaki/micsrvce-project/services/mclientui$
```

# PRE-COMMIT HOOKS

## 7. Si on commit **sans** violation de Eslint rules.

- il n'y a pas d'erreurs dans le commit se passe correctement

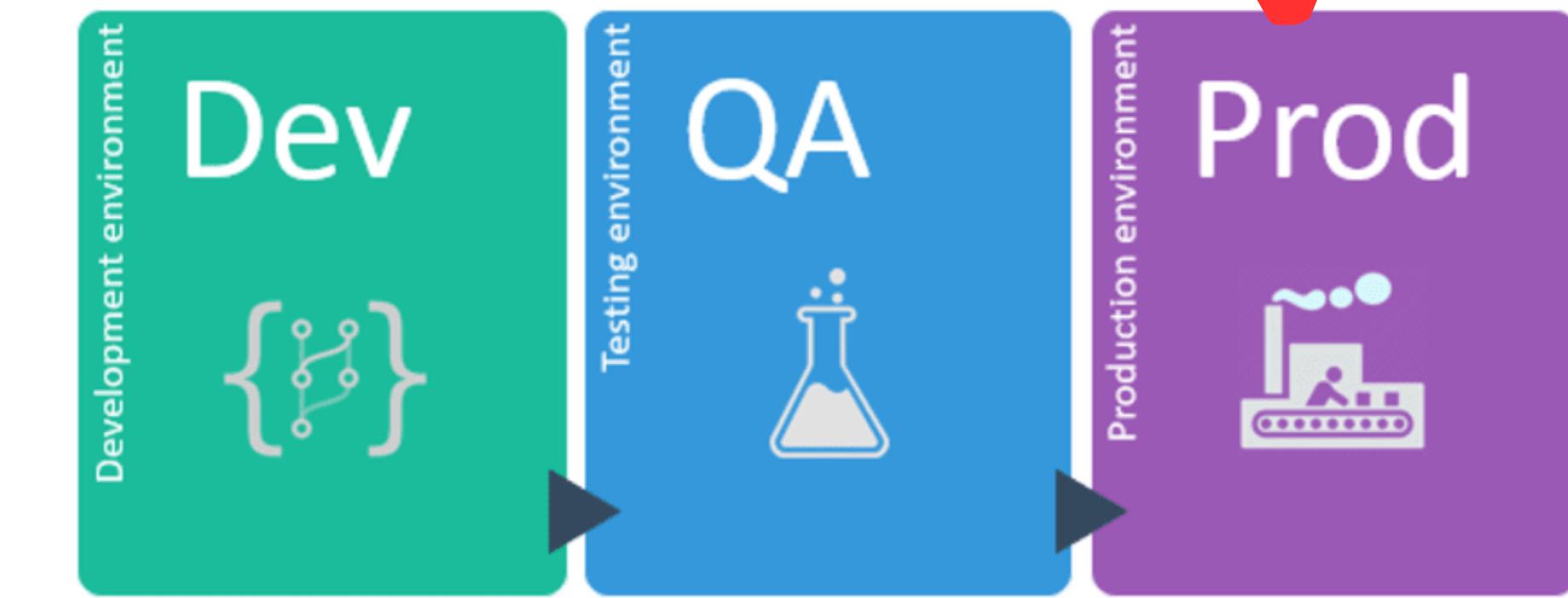
```
● riad@riad-inspiron-3501:~/etude/allaki/micsrvce-project/services/mproducts$ git add .
● riad@riad-Inspiron-3501:~/etude/allaki/micsrvce-project/services/mproducts$ git commit -m "commit without linting errors"
linting before commit...

> micsrvce-project@1.0.0 lint
> eslint src/**/*.ts

> micsrvce-project@1.0.0 pretty-quick
> pretty-quick

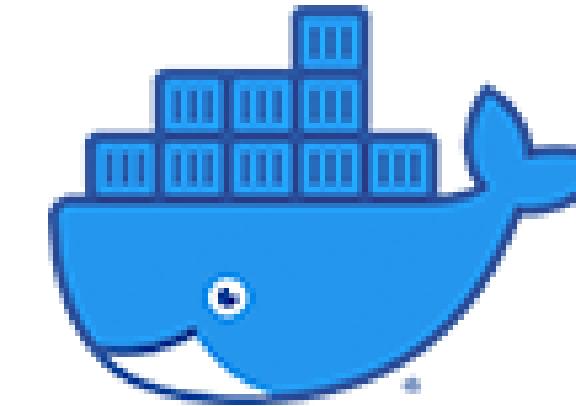
🔍 Finding changed files since git revision null.
⌚️ Found 0 changed files.
✅ Everything is awesome!
[main fabf620b] commit without linting errors
  4 files changed, 60 insertions(+), 70 deletions(-)
    rename .eslintrc.js => .eslintrc.json (51%)
● riad@riad-Inspiron-3501:~/etude/allaki/micsrvce-project/services/mproducts$
```

# Deployment



# Packaging avec Docker

# Dockerfile



# CRÉATION DU FICHIER DOCKERFILE

## Le choix de l'image de base:

Avant de commencer la création de nos fichiers Dockerfile, nous allons choisir l'image de base la plus sécurisée. Pour cela, nous avons utilisé l'outil Snyk afin de nous assurer que l'image sélectionnée est bien sécurisée.

```
$ npm install -g snyk  
$ snyk auth  
$ snyk container test node:16.17.0-bullseye-slim --file=Dockerfile
```

# CRÉATION DU FICHIER DOCKERFILE

## Le choix de l'image de base:

On peut constater que l'image que nous avons utilisée n'est pas sécurisée, et pour résoudre ce problème, nous allons utiliser la version recommandée par Snyk.

```
Organization: asmae20
Package manager: deb
Target file: Dockerfile
Project name: docker-image|node
Docker image: node:16.17.0-bullseye-slim
Platform: linux/amd64
Base image: node:16.17.0-bullseye-slim
Licenses: enabled

Tested 97 dependencies for known issues, found 73 issues.

Base Image          Vulnerabilities  Severity
node:16.17.0-bullseye-slim  73           1 critical, 12 high, 12 medium, 48 low

Recommendations for base image upgrade:

Minor upgrades
Base Image          Vulnerabilities  Severity
node:16.20-bullseye-slim  48           0 critical, 0 high, 0 medium, 48 low

Major upgrades
Base Image          Vulnerabilities  Severity
node:20.0-bullseye-slim  48           0 critical, 0 high, 0 medium, 48 low

Alternative image types
Base Image          Vulnerabilities  Severity
node:18-slim          48           0 critical, 0 high, 0 medium, 48 low
node:20.0-buster-slim  63           0 critical, 3 high, 1 medium, 59 low
node:20.0-buster       359          0 critical, 7 high, 8 medium, 344 low
```



# CRÉATION DU FICHIER DOCKERFILE

---

On prend comme exemple le Dockerfile du microservice "product" et on va l'analyser .

Ce Dockerfile est divisé en deux étapes : la première pour construire l'application avec toutes les dépendances de développement, et la seconde pour créer une image de production légère qui ne contient que les artefacts de construction nécessaires à l'exécution de l'application.

```
# Stage 1: Build the application
FROM node:16.20-bullseye-slim AS build

# Set environment variables
ENV NODE_ENV=development
ENV MONGO_URI=mongodb://mongodb-service/microservices
ENV PORT=5001

# Set the working directory
WORKDIR /app

# Copy package.json and package-lock.json to install dependencies
COPY package*.json .

RUN npm install

# Copy the source code and build the application
COPY . .

RUN npm run build

# Stage 2: Create the production image
FROM node:16.20-bullseye-slim

# Set environment variables
ENV NODE_ENV=production
ENV PORT=5001

# Set the working directory
WORKDIR /app

# Copy the build artifacts from the previous stage
COPY --from=build /app/build ./build

# Install only production dependencies
COPY package*.json .
RUN npm ci --only=production

# Expose the port on which the application listens
EXPOSE 5001

# Use a non-root user for security reasons
USER node

# Start the application
CMD ["node", "build/server.js"]
```

# CRÉATION DU FICHIER DOCKERFILE

Le choix de l'image de base:

Et maintenant, l'image est bien sécurisée.

```
Organization:      asmae20
Package manager:  deb
Target file:       Dockerfile
Project name:     docker-image|node
Docker image:     node:16.20-bullseye-slim
Platform:          linux/amd64
Base image:        node:16.20-bullseye-slim
Licenses:          enabled

Tested 97 dependencies for known issues, found 48 issues.

According to our scan, you are currently using the most secure version of the selected base image
```

# CRÉATION DU FICHIER DOCKERFILE

On définit la variable d'environnement `NODE_ENV` avec la valeur "development". La variable est couramment utilisée dans les applications Node.js pour spécifier l'environnement dans lequel elles s'exécutent

```
# Set environment variables
ENV NODE_ENV=development
ENV MONGO_URI=mongodb://mongodb-service/microservices
ENV PORT=5001
```

Après on définit L'URI qui spécifie l'emplacement de la base de données à laquelle l'application doit se connecter. Et la variable d'environnement `PORT` avec le numéro de port sur lequel l'application écoutera les requêtes entrantes

# CRÉATION DU FICHIER DOCKERFILE

Cette ligne définit le répertoire de travail à l'intérieur du conteneur. Tous les commandes ultérieures seront exécutées à partir de ce répertoire.

```
# Set the working directory  
WORKDIR /app
```

Cette ligne copie les fichiers package.json et package-lock.json depuis le répertoire local vers le répertoire de travail du conteneur. Ces fichiers sont nécessaires pour installer les dépendances de l'application.

```
# Copy package.json and package-lock.json to install dependencies  
COPY package*.json ./
```

Cette commande exécute la commande npm install pour installer les dépendances de l'application en fonction des fichiers package.json et package-lock.json.

```
RUN npm install
```

# CRÉATION DU FICHIER DOCKERFILE

Ensuite nous entrons dans la deuxième étape de création de l'image de production. Nous utilisons à nouveau l'image de base node:16.20-bullseye-slim.

```
# Stage 2: Create the production image
FROM node:16.20-bullseye-slim
```

Après on définit les variables d'environnement pour l'exécution de l'application en mode production. Ici, nous définissons NODE\_ENV sur "production" et PORT sur 5001.

```
# Set environment variables
ENV NODE_ENV=production
ENV PORT=5001
```

# CRÉATION DU FICHIER DOCKERFILE

Et on copie les artefacts de construction (le répertoire "build") depuis l'étape de construction précédente (étiquetée "build") vers le répertoire "build" du conteneur de production. Cela signifie que seul le code construit et nécessaire pour l'exécution de l'application est inclus dans l'image de production

```
# Copy the build artifacts from the previous stage  
COPY --from=build /app/build ./build
```

et on copie à nouveau les fichiers package.json et package-lock.json, puis on exécute la commande npm ci --only=production pour installer uniquement les dépendances nécessaires à l'exécution de l'application en mode production. Cela permet de réduire la taille de l'image finale

```
# Install only production dependencies  
COPY package*.json ./  
RUN npm ci --only=production
```

# CRÉATION DU FICHIER DOCKERFILE

Et on indique que le conteneur expose le port 5001, sur lequel l'application écoute les requêtes.

```
# Expose the port on which the application listens  
EXPOSE 5001
```

et on définit l'utilisateur du conteneur comme "node" plutôt que "root", pour des raisons de sécurité et pour réduire les risques potentiels.

```
# Use a non-root user for security reasons  
USER node
```

la commande de démarrage de l'application lorsqu'elle est exécutée dans le conteneur. Dans cet exemple, il exécute le fichier "server.js" situé dans le répertoire "build" de l'application à l'aide de Node.js.

```
# Start the application  
CMD ["node", "build/server.js"]
```

# CRÉATION DU FICHIER DOCKERFILE

On utilise le même Dockerfile pour l'ensemble des services, en modifiant uniquement le port sur lequel l'application écoute les requêtes. et pour le frontend on modifie la commande de démarrage de l'application

```
# Expose the desired port (replace 3000 with your Next.js port)
EXPOSE 3000
# Run the Next.js application
CMD ["npm", "run", "start"]
```

# CRÉATION DU FICHIER DOCKERFILE

Et pour s'assurer que le fichier Dockerfile que nous avons créé est fonctionnel, nous construisons l'image et exécutons le conteneur.

```
HP@Asmae-EL MINGW64 ~/Desktop/microservice-commandes (main)
$ docker build -t order-service:1.0 .
#1 [internal] load build definition from Dockerfile
#1 sha256:43d544408ae165dbaf51328d661e50194061f393d7f5a244fa63df11936654ee
#1 transferring dockerfile: 963B 0.0s done
#1 DONE 0.1s

#2 [internal] load .dockerignore
#2 sha256:10f59d1349dd9ded1ca46b2cd0f69169f51fd19212fbef2f0c50ff5b2b8dd187
#2 transferring context: 34B done
#2 DONE 0.1s

#3 [internal] load metadata for docker.io/library/node:16.20-bullseye-slim
#3 sha256:7e2b59d1fa716baaa51e969bf093e9386706b31c2d352c8ca3551595cc3a0ef4
#3 DONE 0.0s

#4 [1/6] FROM docker.io/library/node:16.20-bullseye-slim
#4 sha256:c3043047158f5d834d66cae9231dd2fd367e48ab3b85e2f460da3b436ffab073
#4 DONE 0.0s

#6 [internal] load build context
#6 sha256:a94d8bdfbdd8fe62b58e0151b55dca4de3b23dfb70e3be474fb50d28a6f918c
#6 transferring context: 1.99kB 0.0s done
#6 DONE 0.1s

#5 [2/6] WORKDIR /app
#5 sha256:295cb79ef9fd716fbe6a1b62eca36b4c807c37667cdc3a41192e2987790ddec2
#5 CACHED

#7 [3/6] COPY package*.json .
#7 sha256:c366610d69ce182a388b8436a8b9f98d79430a0ba49f61c0e9555b26c06e07fb
#7 CACHED

#8 [4/6] RUN npm ci --only=production && npm cache clean --force
#8 sha256:2804d5d6fe80093ea6d21c32a81819ad82c81298084209294c9c1ec4b4a060c3
#8 1.231 npm WARN config only Use `--omit=dev` to omit dev dependencies fro
#8 6.388 npm WARN deprecated @types/mongoose@5.11.97: Mongoose publishes it
#8 6.712 npm WARN deprecated @types/dotenv@8.2.0: This is a stub types defi
#8 16.75
#8 16.75 added 105 packages, and audited 106 packages in 16s
#8 16.75
```

```
HP@Asmae-EL MINGW64 ~/Desktop/microservice-commandes (main)
$ docker logs order-service-container
> micsrvce-project@1.0.0 start:prod
> node build/server.js

Server listening on 5002
Connected to database
```

# GitLab CI





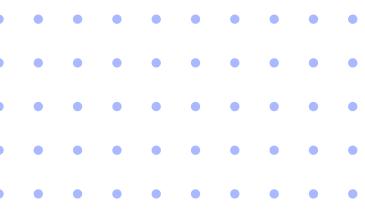
# GITLAB CI

Dans cette étape, nous allons utiliser GitLab CI (Continuous Integration) pour automatiser le processus de build, de scans de sécurité, packaging et de publication des images Docker de chaque service et du frontend. GitLab CI est une solution intégrée fournie par GitLab qui permet d'automatiser les tâches liées au développement, au test et au déploiement d'applications.

L'objectif est de configurer des pipelines GitLab CI qui se déclenchent automatiquement lors de certaines actions, telles que le push de code vers le référentiel GitLab. Ces pipelines exécutent différentes étapes déjà mentionnées.

En automatisant ces étapes clés avec GitLab CI, on gagne en efficacité et en fiabilité dans le processus de construction, de test et de déploiement de nos images Docker. Cela facilite également la collaboration et la gestion du cycle de vie notre applications, en assurant un suivi précis des versions et des déploiements.

Pour chaque service et le frontend, on crée un fichier de configuration GitLab CI (`.gitlab-ci.yml`) qui définit les étapes nécessaires pour le processus d'automatisation.



# Création d'un repo distant avec Gitlab

On commence tout d'abord par la création des repos distant pour chaque microservice

Your work / Projects / New project / Create blank project

### Create blank project

Create a blank project to store your files, plan your work, and collaborate on code, among other things.

**Project name**  
product-microservice

Must start with a lowercase or uppercase letter, digit, emoji, or underscore. Can also contain dots, pluses, dashes, or spaces.

**Project URL**  
https://gitlab.com/ Asmae20

Want to organize several dependent projects under the same namespace? [Create a group](#).

**Project deployment target (optional)**  
Select the deployment target

**Visibility Level**  
 Private  
 Public  
The project can be accessed without any authentication.

**Project Configuration**

Initialize repository with a README  
Allows you to immediately clone this project's repository. Skip this if you plan to push up an existing repository.

Enable Static Application Security Testing (SAST)  
Analyze your source code for known security vulnerabilities. [Learn more](#).

**Create project** **Cancel**

ASMAE ELAZRAK > product-microservice

P **product-microservice**

Project ID: 46348102

0 Commit 1 Branch 0 Tags 0 Bytes Project Storage

Configure SAST in '.gitlab-ci.yml', creating this file if it does not already exist  
ASMAE ELAZRAK authored 59 minutes ago

main / product-microservice / +

README CI/CD configuration Add LICENSE Add CHANGELOG Add CONTRIBUTING Add Kubernetes cluster A

Configure Integrations

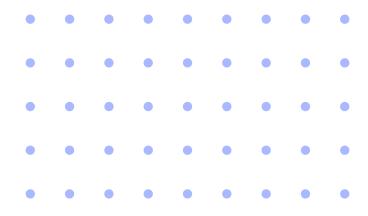
| Name           | Last commit  |
|----------------|--|
| .gitlab-ci.yml | Configure SAST in '.gitlab-ci.yml', creating this file if it ... |
| README.md      | Initial commit   |

README.md

## product-microservice

### Getting started

To make it easy for you to get started with GitLab, here's a list of recommended next steps.



Ensuite pour qu'on puisse faire le merger par n'importe qu'il utilisateur il faut autoriser les push forcés car par défaut dans GitLab, la branche main du projet est protégée.

Protect a branch

Branch:

Select branch or create wildcard ▾

Wildcards such as `*-stable` or `production/*` are supported.

Allowed to merge:

Select ▾

Allowed to push and merge:

Select ▾

Allowed to force push:

Allow all users with push access to [force push](#).

[Protect](#)

| Branch                       | Allowed to merge | Allowed to push and merge | Allowed to force push <a href="#">?</a>                       |
|------------------------------|------------------|---------------------------|---|
| main <a href="#">default</a> | Maintainers ▾    | Maintainers ▾             | <input checked="" type="checkbox"/> <a href="#">Unprotect</a> |

Nous allons maintenant connecter le référentiel local au référentiel distant afin de pusher le code vers le référentiel distant

```
HP@Asmae-EL MINGW64 ~/Desktop/microservice-products (main)
$ git remote set-url origin https://gitlab.com/Asmae20/product-microservice.git

HP@Asmae-EL MINGW64 ~/Desktop/microservice-products (main)
$ git remote
origin

HP@Asmae-EL MINGW64 ~/Desktop/microservice-products (main)
$ |
```

```
HP@Asmae-EL MINGW64 ~/Desktop/microservice-products (main)
$ git push -uf origin main
Enumerating objects: 25, done.
Counting objects: 100% (25/25), done.
Delta compression using up to 8 threads
Compressing objects: 100% (19/19), done.
Writing objects: 100% (25/25), 46.03 KiB | 7.67 MiB/s, done.
Total 25 (delta 2), reused 18 (delta 0), pack-reused 0
To https://gitlab.com/Asmae20/product-microservice.git
 + c318130...3b035ad main -> main (forced update)
Branch 'main' set up to track remote branch 'main' from 'origin'.

HP@Asmae-EL MINGW64 ~/Desktop/microservice-products (main)
$ |
```

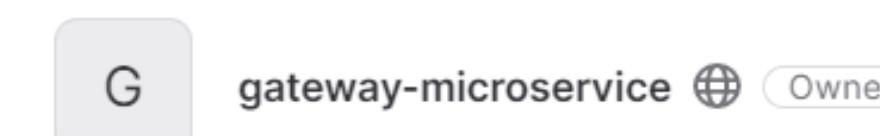
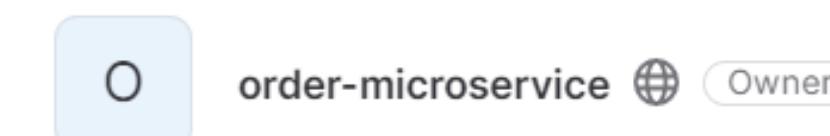
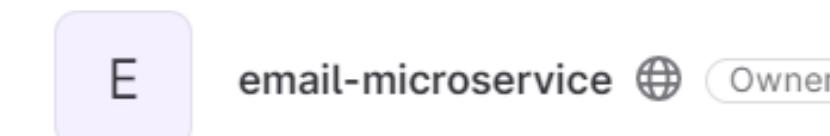
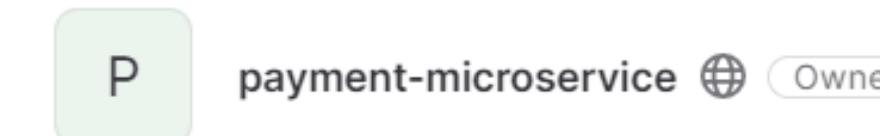
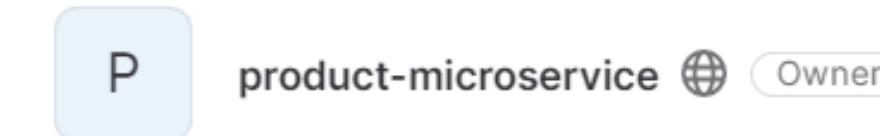
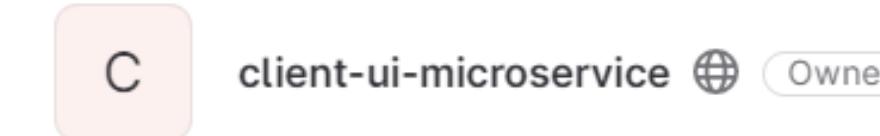


| ASMAE ELAZRAK > product-microservice                            |              |             |
|---|--------------|-------------|
| <a href="#">Dockerfile</a><br>ASMAE ELAZRAK authored 1 week ago |              |             |
| main product-microservice / +                                   |              |             |
| Name  | Last commit  | Last update |
| src   | first commit | 2 weeks ago |
| .dockerignore   | Dockerfile   | 1 week ago  |
| .eslintignore   | first commit | 2 weeks ago |
| .eslintrc.js  | first commit | 2 weeks ago |
| .gitignore  | first commit | 2 weeks ago |
| Dockerfile  | Dockerfile   | 1 week ago  |
| nodemon.json  | first commit | 2 weeks ago |
| package-lock.json   | first commit | 2 weeks ago |
| package.json  | Dockerfile   | 1 week ago  |
| tsconfig.json   | first commit | 2 weeks ago |



Nous suivons le même processus pour tous les autres microservices.

## Personal projects





Ensuite, nous procédons à la création du fichier `.gitlab-ci.yml` pour chaque microservice.

main ✓ client-ui-microservice / `.gitlab-ci.yml` Find file Blame History Permalink

`.gitlab-ci.yml` 1.12 KiB Edit Replace Delete

```
1 image: docker:latest
2 services:
3   - docker:dind
4
5 stages:
6   - sast
7   - build
8   - package
9   - deploy-k8s
10
11 variables:
12   TAG: $CI_COMMIT_MESSAGE
13
14 sast:
15   stage: sast
16   include:
17     - template: Security/SAST.gitlab-ci.yml
18
19 docker-build:
20   stage: package
21   before_script:
22     - docker login -u asmaael -p $DOCKER_HUB_PASS
23   script:
24     - docker build -t asmaael/client-ui-microservice:$TAG .
25     - docker push asmaael/client-ui-microservice:$TAG
26
27 gitops-k8s-deploy:
28   image: bitnami/git:latest
29   stage: deploy-k8s
30   before_script:
31     - git config --global user.email "elazrakasmae@gmail.com"
32     - git config --global user.name "Asmae20"
33   script:
34     - git clone https://gitlab.com/Asmae20/client-ui-microservice.git
35     - cd client-ui-microservice/file-k8s/
36     - sed -i "s/client-ui-microservice:1.0/client-ui-microservice:$TAG/g" client-ui-deployment.yml
37     - cat client-ui-deployment.yml
38     - git add client-ui-deployment.yml
39     - git commit -m "microservice version $TAG"
40     - git remote set-url origin https://Asmae20:$GIT_PASSWORD@gitlab.com/Asmae20/client-ui-microservice.git
41     - git push -uf origin main
42
43 after_script:
44   - docker logout
```

# LE CONTENU DU FICHIER .GITLAB-CI.YML

Cette partie spécifie l'image Docker utilisée pour les étapes du pipeline. L'image docker:latest est utilisée et le service Docker-in-Docker (docker:dind) est activé pour permettre l'exécution des commandes Docker dans les étapes suivantes.

Cette section définit les différentes étapes du pipeline, à savoir "sast" pour l'analyse statique de sécurité, "build" pour la construction de l'image Docker, "package" pour l'empaquetage de l'application, et "deploy-k8s" pour le déploiement sur Kubernetes.

```
image: docker:latest
services:
  - docker:dind
stages:
  - sast
  - build
  - package
  - deploy-k8s
```

# LE CONTENU DU FICHIER .GITLAB-CI.YML

Cette partie définit une variable \$TAG qui est utilisée pour stocker le message de validation (\$CI\_COMMIT\_MESSAGE) afin d'être utilisé comme balise (tag) pour l'image Docker.

**variables:**

**TAG: \$CI\_COMMIT\_MESSAGE**

Cette section définit l'étape "sast" (analyse statique de sécurité) en incluant un modèle de configuration pré-défini (Security/SAST.gitlab-ci.yml) pour exécuter des analyses de sécurité sur le code source

**sast:**

**stage: sast**

**include:**

**- template: Security/SAST.gitlab-ci.yml**

# LE CONTENU DU FICHIER .GITLAB-CI.YML

Cette partie définit l'étape "docker-build" pour la construction de l'image Docker. Avant d'exécuter le script, la commande docker login est utilisée pour s'authentifier auprès du registre Docker. Ensuite, la commande docker build est utilisée pour construire l'image avec la balise spécifiée par \$TAG, et docker push est utilisée pour pousser l'image vers le registre Docker.

```
docker-build:  
  stage: package  
  before_script:  
    - docker login -u asmaeel -p $DOCKER_HUB_PASS  
  script:  
    - docker build -t asmaeel/client-ui-microservice:$TAG .  
    - docker push asmaeel/client-ui-microservice:$TAG
```

# LE CONTENU DU FICHIER .GITLAB-CI.YML

Cette partie définit l'étape "gitops-k8s-deploy" pour le déploiement sur Kubernetes. L'image bitnami/git:latest est utilisée pour cette étape. Le script clone le référentiel distant, effectue des modifications dans le fichier de déploiement Kubernetes (client-ui-deployment.yml), puis effectue un commit et un push vers le référentiel distant pour déployer la nouvelle version du microservice.

```
gitops-k8s-deploy:  
  image: bitnami/git:latest  
  stage: deploy-k8s  
  before_script:  
    - git config --global user.email "elazrakasmae@gmail.com"  
    - git config --global user.name "Asmae20"  
  script:  
    - git clone https://gitlab.com/Asmae20/client-ui-microservice.git  
    - cd client-ui-microservice/file-k8s/  
    - sed -i "s/client-ui-microservice:1.0/client-ui-microservice:$TAG/g" client-ui-deployment.yml  
    - cat client-ui-deployment.yml  
    - git add client-ui-deployment.yml  
    - git commit -m "microservice version $TAG"  
    - git remote set-url origin https://Asmae20:$GIT_PASSWORD@gitlab.com/Asmae20/client-ui-microservice.git  
    - git push -uf origin main
```

# LE CONTENU DU FICHIER .GITLAB-CI.YML

Cette section spécifie les commandes à exécuter après l'exécution de toutes les étapes du pipeline. Dans ce cas, la commande docker logout est utilisée pour se déconnecter du registre Docker.

```
after_script:  
  - docker logout
```

# LE CONTENU DU FICHIER .GITLAB-CI.YML

Pour exécuter ce fichier, il est nécessaire d'ajouter les variables requises. Dans notre cas, nous allons ajouter deux variables

| Type   | ↑ Key   | Value   | Options             | Environments  |   |
|--|---|---|---------------------|---|---|
| Variable   | DOCKER_HUB_PASS  | *****  | Protected, Expanded | All (default)  |  |
| Variable   | GIT_PASSWORD     | *****  | Protected, Expanded | All (default)  |  |
| <a href="#">Add variable</a> <a href="#">Reveal values</a> |   |   |                     |   |   |

# LE CONTENU DU FICHIER .GITLAB-CI.YML

Et voilà notre pipeline est bien passé .

ASMAE ELAZRAK > payment-microservice > Pipelines > #882137125

passed

Pipeline #882137125 triggered 1 hour ago by ASMAE ELAZRAK

1.1

⌚ 4 jobs for main

in 2 minutes and 27 seconds, using 3.21 compute credits, and was queued for 0 seconds

-o 09d3d2d5 ↗

∅ No related merge requests found.

Pipeline   Needs   Jobs 4   Tests 0

sast

✓ nodejs-scan-sast



✓ semgrep-sast



package

✓ docker-build



deploy-k8s

✓ gitops-k8s-deploy



# LE CONTENU DU FICHIER .GITLAB-CI.YML

Et voilà, on peut constater que toutes les images sont présentes sur Docker Hub.

The screenshot shows the Docker Hub interface with the search bar set to 'asmaeel'. Five repositories are listed:

- asmaeel / client-ui-microservice**: Contains: Image | Last pushed: 4 days ago. Status: Inactive. Stars: 0. Downloads: 9. Public.
- asmaeel / payment-microservice**: Contains: Image | Last pushed: 4 days ago. Status: Inactive. Stars: 0. Downloads: 3. Public.
- asmaeel / product-microservice**: Contains: Image | Last pushed: 4 days ago. Status: Inactive. Stars: 0. Downloads: 20. Public.
- asmaeel / email-microservice**: Contains: Image | Last pushed: 4 days ago. Status: Inactive. Stars: 0. Downloads: 8. Public.
- asmaeel / order-microservice**: Contains: Image | Last pushed: 4 days ago. Status: Inactive. Stars: 0. Downloads: 4. Public.
- asmaeel / gateway-microservice**: Contains: Image | Last pushed: 5 days ago. Status: Inactive. Stars: 0. Downloads: 7. Public.

# Kubernetes



# kubernetes



# KUBERNETES

Une fois que les pipelines GitLab CI ont été configurés pour chaque service et le frontend, nous pouvons passer à l'étape suivante : la création d'un répertoire GitLab spécifique qui contiendra les fichiers YAML nécessaires pour le déploiement de l'application dans un cluster Kubernetes local.

Pour cela, nous allons ajouter les fichiers de configuration Kubernetes nécessaires, tels que les déploiements, les services, les ingress, les secrets, etc. Ces fichiers YAML définissent comment l'application doit être déployée et configurée dans un environnement Kubernetes.

Chaque microservice et le frontend auront leurs propres fichiers YAML correspondants. Ces fichiers spécifient les ressources Kubernetes nécessaires pour déployer et exécuter chaque composant de l'application de manière isolée et cohérente.

En créant un répertoire GitLab dédié aux fichiers YAML de Kubernetes, nous centralisons et gérions de manière organisée les configurations de déploiement de l'application. Cela facilite également la reproductibilité, la maintenance et la collaboration entre les membres de l'équipe qui travaillent sur le déploiement de l'application dans le cluster Kubernetes local.



# K8S FILES

Nous utiliserons le microservice "gateway" comme exemple pour analyser le contenu du fichier YAML.

ASMAE ELAZRAK > gateway-microservice

Update file  
ASMAE ELAZRAK authored 4 days ago

main + v gateway-microservice / file-k8s / + v

History Find file Web IDE v Clone v

| Name                    | Last commit | Last update |
|-------------------------|-------------|-------------|
| ..                      |             |             |
| gateway-deployment.yml  | Update file | 4 days ago  |
| gateway-service.yml     | Update file | 6 days ago  |
| mongodb-statefulset.yml | Update file | 6 days ago  |



# K8S FILES

Nous commençons par créer le fichier "mongodbsatefulset nécessaire à l'exécution du microservice.

main ▾ gateway-microservice / file-k8s / mongodb-statefulset.yml

mongodb-statefulset.yml 1.49 KiB

Edit Find file Blame History Permalink

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: mongodb
  namespace: m-commerce
spec:
  serviceName: mongodb
  replicas: 1
  selector:
    matchLabels:
      app: mongodb
  template:
    metadata:
      labels:
        app: mongodb
    spec:
      containers:
        - name: mongodb
          image: mongo:latest
          command: ["mongod"]
          args: [--bind_ip_all, --noauth]
          ports:
            - containerPort: 27017
          livenessProbe:
            exec:
              command:
                - echo
                - -e
                - '--eval "db.adminCommand('ping')"'
                - \\
                  - xargs
                  - mongo
            initialDelaySeconds: 15
            periodSeconds: 20
          readinessProbe:
            exec:
              command:
                - echo
                - -e
                - '--eval "db.adminCommand('ping')"'
                - \\
                  - xargs
                  - mongo
            initialDelaySeconds: 5
            periodSeconds: 10
          volumeMounts:
            - name: mongo-data
              mountPath: /data/db
          volumeClaimTemplates:
            - metadata:
                name: mongo-data
```

# K8S FILES

Cette partie spécifie les métadonnées du StatefulSet, y compris son nom et l'espace de noms dans lequel il sera déployé.

Dans cette partie, nous définissons les spécifications du StatefulSet. Nous spécifions le nom du service associé, le nombre de réplicas (dans ce cas, 1) et le sélecteur utilisé pour faire correspondre les pods.

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: mongodb
  namespace: m-commerce

spec:
  serviceName: mongodb
  replicas: 1
  selector:
    matchLabels:
      app: mongodb
```

# K8S FILES

La partie template contient les spécifications du pod à créer. Nous définissons les métadonnées du pod, y compris les labels. Ensuite, nous spécifions le conteneur principal du pod, avec son nom, l'image Docker utilisée (mongo:latest), la commande et les arguments à exécuter dans le conteneur, ainsi que les ports à exposer.

```
template:  
  metadata:  
    labels:  
      app: mongodb  
  spec:  
    containers:  
      - name: mongodb  
        image: mongo:latest  
        command: ["mongod"]  
        args: [--bind_ip_all, --noauth]  
    ports:  
      - containerPort: 27017
```

# K8S FILES

Cette partie définit les sondes de disponibilité pour le pod. La sonde de disponibilité (livenessProbe) vérifie si le pod est en état de fonctionnement, tandis que la sonde de préparation (readinessProbe) vérifie si le pod est prêt à recevoir du trafic. Dans ce cas, nous utilisons une sonde exécutable qui exécute une commande spécifique (echo '--eval "db.adminCommand("ping")"' | xargs mongo) pour vérifier la disponibilité du service MongoDB.

```
livenessProbe:  
  exec:  
    command:  
      - echo  
      - -e  
      - '--eval "db.adminCommand('ping')"'  
      - \|  
      - xargs  
      - mongo  
  initialDelaySeconds: 15  
  periodSeconds: 20  
readinessProbe:  
  exec:  
    command:  
      - echo  
      - -e  
      - '--eval "db.adminCommand('ping')"'  
      - \|  
      - xargs  
      - mongo  
  initialDelaySeconds: 5  
  periodSeconds: 10
```

# K8S FILES

Cette partie spécifie les montages de volumes pour le conteneur. Dans ce cas, nous montons un volume nommé mongo-data sur le chemin /data/db du conteneur. Cela permet de stocker les données de MongoDB de manière persistante.

Nous définissons un modèle de demande de volume (volumeClaimTemplates) pour le StatefulSet. Il spécifie les caractéristiques du volume persistant réclamé, y compris les modes d'accès (ReadWriteOnce) et les ressources demandées (1Gi de stockage).

## volumeMounts:

- **name: mongo-data**

## mountPath: /data/db

## volumeClaimTemplates:

- **metadata:**

**name: mongo-data**

## spec:

**accessModes: [ "ReadWriteOnce" ]**

## resources:

## requests:

**storage: 1Gi**

# K8S FILES

Cette partie spécifie les métadonnées du Service, y compris son nom et l'espace de noms dans lequel il sera déployé. nous définissons les spécifications du Service. Nous utilisons le sélecteur app: mongodb pour faire correspondre les pods, et nous exposons le port 27017 en utilisant le protocole TCP et en le mappant au port 27017 des pods.

```
---  
apiVersion: v1  
kind: Service  
metadata:  
  name: mongodb-service  
  namespace: m-ecommerce  
spec:  
  selector:  
    app: mongodb  
  ports:  
    - protocol: TCP  
      port: 27017  
      targetPort: 27017
```



# K8S FILES

Maintenant, nous allons examiner de plus près le fichier "deployment" du microservice Gateway pour comprendre sa configuration et son fonctionnement :

gateway-deployment.yaml 1.49 KiB

Edit Replace Delete

```
1 apiVersion: v1
2 kind: ConfigMap
3 metadata:
4   name: mongodb-conf
5   namespace: m-ecommerce
6 data:
7   host: mongodb-service
8   port: "27017"
9 ---
10
11 apiVersion: v1
12 kind: Secret
13 metadata:
14   name: jwt-secret
15   namespace: m-ecommerce
16 data:
17   token_key: MTMyMTMyNAo=
18
19
20
21 ---
22 apiVersion: apps/v1
23 kind: Deployment
24 metadata:
25   name: gateway-deployment
26   namespace: m-ecommerce
27 spec:
28   selector:
29     matchLabels:
30       app: gateway-deployment
31   replicas: 1
32   template:
33     metadata:
34       labels:
35         app: gateway-deployment
36   spec:
37     containers:
38       - name: gateway-deployment
39         image: asmaael/gateway-microservice:1.4
40         ports:
41           - containerPort: 5005
42             name: gateway-port
43         env:
44           - name: DB_HOST
45             valueFrom:
46               configMapKeyRef:
47                 name: mongodb-conf
48                 key: host
49           - name: DB_PORT
50             valueFrom:
51               configMapKeyRef:
52                 name: mongodb-conf
53                 key: port
54           - name: MONGO_URI
55             value: "mongodb://mongodb-service/microservices"
56           - name: ORDERS_SERVER
57             value: "http://order-service:5002/api"
58           - name: PRODUCTS_SERVER
59             value: "https://product-service:5001/api"
```

# K8S FILES

Ce bloc définit un ConfigMap nommé "mongodb-conf" dans l'espace de noms "m-commerce". Il contient deux clés (host et port) avec leurs valeurs respectives. Ce ConfigMap sera utilisé pour fournir la configuration du service MongoDB à d'autres composants.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: mongodb-conf
  namespace: m-commerce
data:
  host: mongodb-service
  port: "27017"
---
```

# K8S FILES

Ce bloc définit un Secret nommé "jwt-secret" dans l'espace de noms "m-eCommerce". Le Secret contient une clé "token\_key" qui stocke une valeur encodée. Ce Secret est utilisé pour stocker de manière sécurisée des informations sensibles, dans ce cas, une clé JWT.

```
apiVersion: v1
kind: Secret
metadata:
  name: jwt-secret
  namespace: m-eCommerce
data:
  token_key: MTMyMTMyNAo=
```

# K8S FILES

- Nous spécifions le sélecteur pour le déploiement, en utilisant l'étiquette "app: gateway-deployment" pour sélectionner les pods correspondants.
- Nous définissons le nombre de répliques du déploiement à 1, ce qui signifie qu'un seul pod sera déployé pour le microservice Gateway.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: gateway-deployment
  namespace: m-ecommerce
spec:
  selector:
    matchLabels:
      app: gateway-deployment
  replicas: 1
```

# K8S FILES

La section "template" contient les métadonnées et les spécifications pour le pod créé par le déploiement.

- Dans la section "containers", nous définissons le conteneur pour le microservice Gateway.
- Nous spécifions l'image Docker à utiliser pour le conteneur, en utilisant "asmaeel/gateway-microservice:1.4".
- Le conteneur écoute le port 5005 et nous lui donnons le nom "gateway-port".

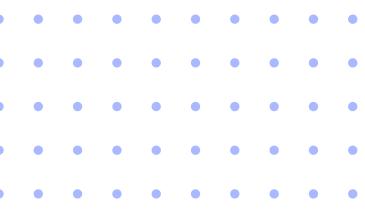
```
template:  
  metadata:  
    labels:  
      app: gateway-deployment  
  
  spec:  
    containers:  
      - name: gateway-deployment  
        image: asmaeel/gateway-microservice:1.4  
        ports:  
          - containerPort: 5005  
            name: gateway-port
```

# K8S FILES

- Nous définissons plusieurs variables d'environnement pour le conteneur, telles que "DB\_HOST" et "DB\_PORT", qui récupèrent leurs valeurs à partir de la ConfigMap "mongodb-conf". Nous avons également des variables d'environnement pour l'URL de la base de données MongoDB, ainsi que les URL des services "orders-service", "products-service" et "payment-service".
- Enfin, nous utilisons une Secret "jwt-secret" pour récupérer la clé de jeton JWT à utiliser dans le microservice Gateway.

env:

```
- name: DB_HOST
  valueFrom:
    configMapKeyRef:
      name: mongodb-conf
      key: host
- name: DB_PORT
  valueFrom:
    configMapKeyRef:
      name: mongodb-conf
      key: port
- name: MONGO_URI
  value: "mongodb://mongodb-service/microservices"
- name: ORDERS_SERVER
  value: "http://order-service:5002/api"
- name: PRODUCTS_SERVER
  value: "http://product-service:5001/api"
- name: PAYMENT_SERVER
  value: "http://payment-service:5003/api"
- name: JWT-Secret
  valueFrom:
    secretKeyRef:
      name: jwt-secret
      key: token_key
```



Continuons maintenant en examinant le fichier de configuration "service" du microservice Gateway :

Le service est configuré avec les informations suivantes :

- Le nom du service est "gateway-service", ce qui permet de l'identifier de manière unique.
- Le service est créé dans l'espace de noms "m-ecommerce", ce qui permet de regrouper les ressources liées au microservice dans cet espace de noms spécifique.
- Le service est étiqueté avec "app: gateway-service", ce qui facilite l'identification et la sélection de ce service en utilisant ces étiquettes.
- Le sélecteur du service est défini sur "app: gateway-deployment", ce qui permet au service de cibler les pods associés au microservice Gateway.
- Le service écoute sur le port 5005 en utilisant le protocole TCP. Cela signifie que les requêtes TCP entrantes sur le port 5005 seront redirigées vers les pods du microservice Gateway.
- Le service utilise le port cible 5005 à l'intérieur des pods du microservice Gateway.

ASMAE ELAZRAK > gateway-microservice

 Update file  
ASMAE ELAZRAK authored 6 days ago

main ▾ gateway-microservice / file-k8s / gateway-service.yml

 gateway-service.yml 237 bytes

```
1 apiVersion: v1
2 kind: Service
3 metadata:
4   name: gateway-service
5   namespace: m-ecommerce
6   labels:
7     app: gateway-service
8 spec:
9   selector:
10    app: gateway-deployment
11   ports:
12     - protocol: TCP
13       port: 5005
14       targetPort: 5005
```

# K8S FILES

Pour chaque autre microservice, nous suivons la même approche que pour le microservice Gateway. Nous créons un objet de type "Service" dans Kubernetes pour chaque microservice afin de les exposer et de permettre aux autres composants de communiquer avec eux.

Cependant, pour le frontend, nous ajoutons une fonctionnalité supplémentaire appelée "Ingress". L'Ingress est un objet Kubernetes qui permet de gérer l'accès externe à nos services depuis l'extérieur du cluster Kubernetes. Il agit comme une passerelle ou un point d'entrée unique pour les requêtes entrantes.

L'ajout de l'Ingress pour le frontend nous permet de configurer des règles d'acheminement basées sur les noms d'hôtes ou les chemins d'URL. Cela signifie que nous pouvons rediriger les requêtes vers le frontend en fonction de l'URL demandée. Par exemple, dans notre cas nous pouvons définir une règle qui redirige les requêtes pour "m-commerce.com" vers le service frontend.



L'Ingress est utilisé pour gérer l'accès externe à nos services depuis l'extérieur du cluster Kubernetes.

- La section "metadata" spécifie les métadonnées de l'Ingress, telles que son nom, son espace de noms et les annotations associées.
- Les annotations fournissent des informations supplémentaires et des configurations spécifiques à l'Ingress. Dans cet exemple, nous utilisons les annotations "kubernetes.io/ingress.class" pour spécifier la classe d'Ingress utilisée (dans ce cas, "nginx"), "nginx.ingress.kubernetes.io/add-base-url" pour ajouter l'URL de base à l'acheminement des requêtes, et "nginx.ingress.kubernetes.io/rewrite-target" pour spécifier la cible de réécriture des requêtes.
- La section "spec" définit les règles de l'Ingress. Dans cet exemple, nous avons une seule règle qui spécifie le nom d'hôte "m-commerce.com". Toutes les requêtes entrantes avec cet hôte seront gérées par cette règle.
- Sous la règle, nous spécifions un chemin "/" avec le type de chemin "Prefix". Cela signifie que toutes les requêtes commençant par "/" seront redirigées vers le service backend spécifié.
- Dans la section "backend", nous spécifions le service "client-ui" comme backend de destination pour ces requêtes. Le port 3000 du service sera utilisé.

ASMAE ELAZRAK > client-ui-microservice > Repository

main ▾

client-ui-microservice / file-k8s / **client-ui-ingress.yml**



Update file

ASMAE ELAZRAK authored 3 days ago

**client-ui-ingress.yml** 510 bytes

```
1 apiVersion: networking.k8s.io/v1
2 kind: Ingress
3 metadata:
4   name: app-ingress
5   namespace: m-eCommerce
6   annotations:
7     kubernetes.io/ingress.class: "nginx"
8     nginx.ingress.kubernetes.io/add-base-url: "true"
9     nginx.ingress.kubernetes.io/rewrite-target: /
10
11 spec:
12   rules:
13     - host: m-commerce.com
14       http:
15         paths:
16           - path: /
17             pathType: Prefix
18           backend:
19             service:
20               name: client-ui
21               port:
22                 number: 3000
```



Il faut ajouter cette commande pour que Ingress fonctionne, et également vérifier que l'adresse IP correspond au nom d'hôte spécifié.

```
HP@Asmae-EL MINGW64 ~/Desktop/microservice-client-ui/file-k8s (main)
$ minikube addons enable ingress
* ingress est un addon maintenu par Kubernetes. Pour toute question, contactez mir
Vous pouvez consulter la liste des mainteneurs de minikube sur : https://github.co
* Après que le module est activé, veuiller exécuter "minikube tunnel" et vos resso
  - Utilisation de l'image registry.k8s.io/ingress-nginx/controller:v1.5.1
  - Utilisation de l'image registry.k8s.io/ingress-nginx/kube-webhook-certgen:v202
  - Utilisation de l'image registry.k8s.io/ingress-nginx/kube-webhook-certgen:v202
* Vérification du module ingress...

X Fermeture en raison de MK_ADDON_ENABLE : run callbacks: running callbacks: [sud
-f /etc/kubernetes/addons/ingress-deploy.yaml: Process exited with status 1
stdout:
namespace/ingress-nginx unchanged
serviceaccount/ingress-nginx unchanged
serviceaccount/ingress-nginx-admission unchanged
role.rbac.authorization.k8s.io/ingress-nginx unchanged
role.rbac.authorization.k8s.io/ingress-nginx-admission unchanged
clusterrole.rbac.authorization.k8s.io/ingress-nginx unchanged
clusterrole.rbac.authorization.k8s.io/ingress-nginx-admission unchanged
rolebinding.rbac.authorization.k8s.io/ingress-nginx unchanged
rolebinding.rbac.authorization.k8s.io/ingress-nginx-admission unchanged
clusterrolebinding.rbac.authorization.k8s.io/ingress-nginx unchanged
clusterrolebinding.rbac.authorization.k8s.io/ingress-nginx-admission unchanged
```

```
# localhost name resolution is handled within DN
#           127.0.0.1      localhost
#           ::1            localhost
127.0.0.1 m-ecommerce.com
192.168.49.2 asmae-k8s.com
```

The AT&T logo, which consists of a stylized globe with horizontal blue and white stripes.

Et voilà, on peut constater que tous les microservices sont correctement déployés et fonctionnels.

# ArgoCD



argo

# ARGOCD

Dans cette étape, nous introduisons l'outil ArgoCD pour faciliter le déploiement de nos microservices mis à jour dans notre cluster local de Kubernetes. ArgoCD est un outil de déploiement continu spécifiquement conçu pour Kubernetes. Il offre des fonctionnalités avancées telles que la détection automatique des changements dans un référentiel GitLab contenant les fichiers YAML de nos services.

L'idée ici est d'intégrer ArgoCD à notre pipeline CI existant dans GitLab. Lorsque des modifications sont détectées dans le référentiel GitLab, ArgoCD prendra le relais et déclenchera le processus de déploiement des microservices mis à jour dans notre cluster Kubernetes local.

En utilisant ArgoCD, nous bénéficions de plusieurs avantages. Tout d'abord, cela nous permet d'automatiser le déploiement de nos microservices, éliminant ainsi la nécessité d'interventions manuelles et réduisant les erreurs humaines potentielles. De plus, ArgoCD offre une gestion centralisée du déploiement, nous permettant de garder une trace précise des versions déployées et de gérer facilement les mises à jour.

# ARGO CD

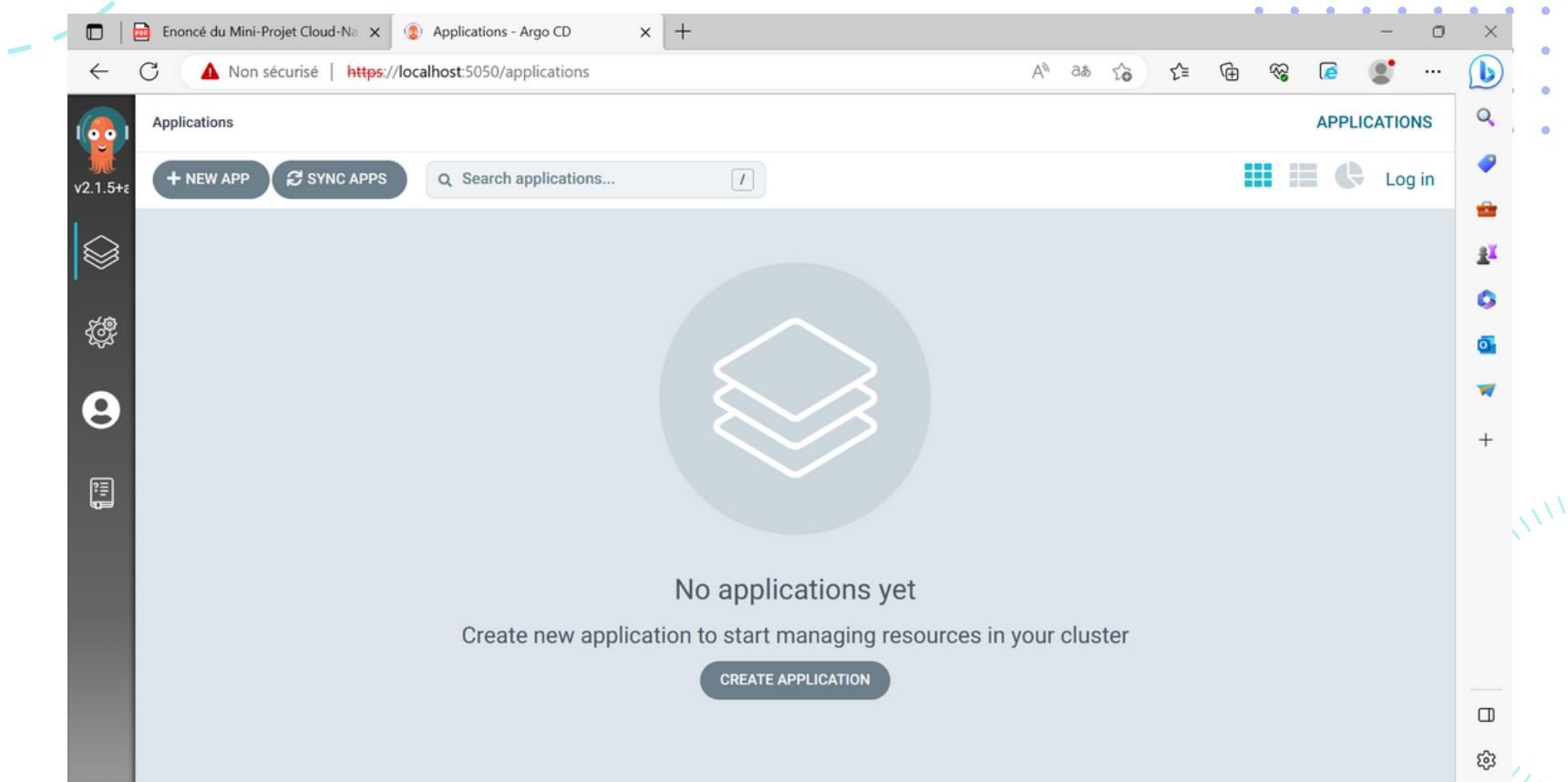
Pour installer correctement Agocd, il est nécessaire de suivre les étapes indiquées dans les images ci-dessous.

```
HP@Asmae-EL MINGW64 ~/Desktop/microservice-products (main)
$ kubectl create namespace argocd
namespace/argocd created

HP@Asmae-EL MINGW64 ~/Desktop/microservice-products (main)
$ kubectl get namespaces
NAME          STATUS  AGE
argocd        Active  3s
default       Active  3h
kube-node-lease  Active  3h
kube-public   Active  3h
kube-system   Active  3h
m-commerce    Active  123m
```

```
HP@Asmae-EL MINGW64 ~/Desktop/microservice-products (main)
$ kubectl get pods -n argocd
NAME                           READY   STATUS    RESTARTS   AGE
argocd-application-controller-0  1/1     Running   0          3m49s
argocd-dex-server-5477b5f4df-xd4f1 1/1     Running   0          3m50s
argocd-redis-99f9c4888-cs947      1/1     Running   0          3m50s
argocd-repo-server-56cb78f958-xrfm5 1/1     Running   0          3m50s
argocd-server-58c689786c-5gvqc    1/1     Running   0          3m49s

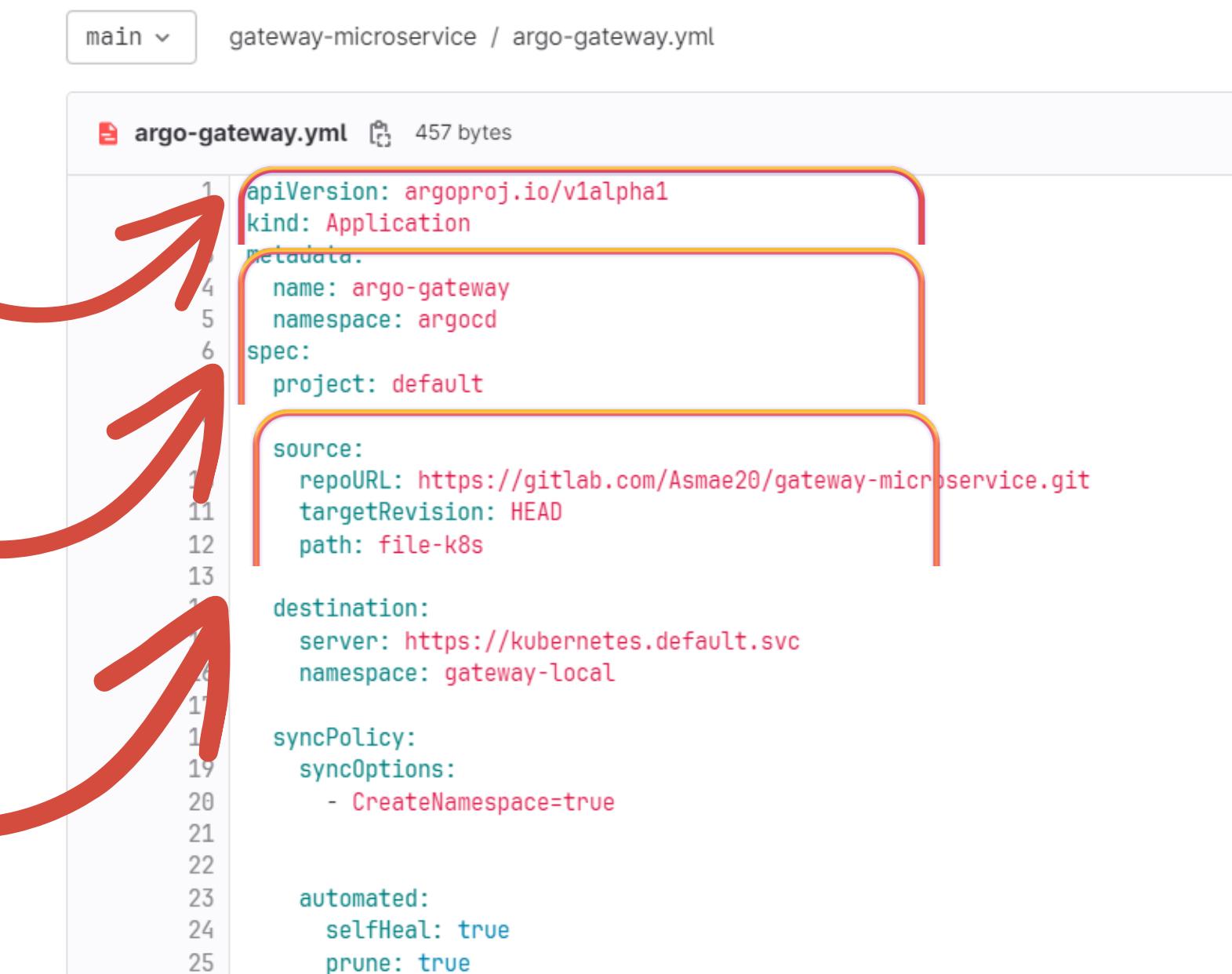
HP@Asmae-EL MINGW64 ~/Desktop/microservice-products (main)
$
```



# ARGOCD

Après avoir créé l'application que nous souhaitons afficher dans ArgoCD manuellement en écrivant ce code...

- Le code spécifie la version de l'API utilisée par ArgoCD, qui est "argoproj.io/v1alpha1". La ressource définie est une application ArgoCD.
- La section "metadata" contient les informations sur l'application, telles que son nom ("argo-gateway") et le namespace dans lequel elle sera déployée ("argocd").
- Dans la section "spec", le nom du projet ArgoCD auquel l'application appartient est défini comme "default".
- La section "source" spécifie la configuration de la source de déploiement. L'URL du dépôt Git où se trouve le code de l'application est définie comme "
- La révision cible est définie comme "HEAD", ce qui signifie la dernière révision disponible.
- Le chemin vers les fichiers Kubernetes de l'application est spécifié comme "file-k8s".



```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: argo-gateway
  namespace: argocd
spec:
  project: default
  source:
    repoURL: https://gitlab.com/Asmae20/gateway-microservice.git
    targetRevision: HEAD
    path: file-k8s
  destination:
    server: https://kubernetes.default.svc
    namespace: gateway-local
  syncPolicy:
    syncOptions:
      - CreateNamespace=true
  automated:
    selfHeal: true
    prune: true
```

# ARGOCD

Après avoir créé l'application que nous souhaitons afficher dans ArgoCD manuellement en écrivant ce code...

- La section "destination" indique la configuration de la destination du déploiement. Le serveur de destination est défini comme "Le code spécifie la version de l'API utilisée par ArgoCD, qui est "argoproj.io/v1alpha1". La ressource définie est une application ArgoCD." et le namespace de destination est défini comme "gateway-local".
- La section "syncPolicy" contient la politique de synchronisation de l'application. L'option "CreateNamespace=true" indique que le namespace doit être créé si nécessaire.
- La sous-section "automated" spécifie les options de synchronisation automatique. Les options "selfHeal" et "prune" sont définies sur "true", ce qui signifie que l'application se guérira automatiquement et supprimera les ressources obsolètes lors de la synchronisation.

**destination:**  
server: <https://kubernetes.default.svc>  
namespace: gateway-local

**syncPolicy:**  
**syncOptions:**  
- CreateNamespace=true

**automated:**  
selfHeal: true  
prune: true

# ARGOCD

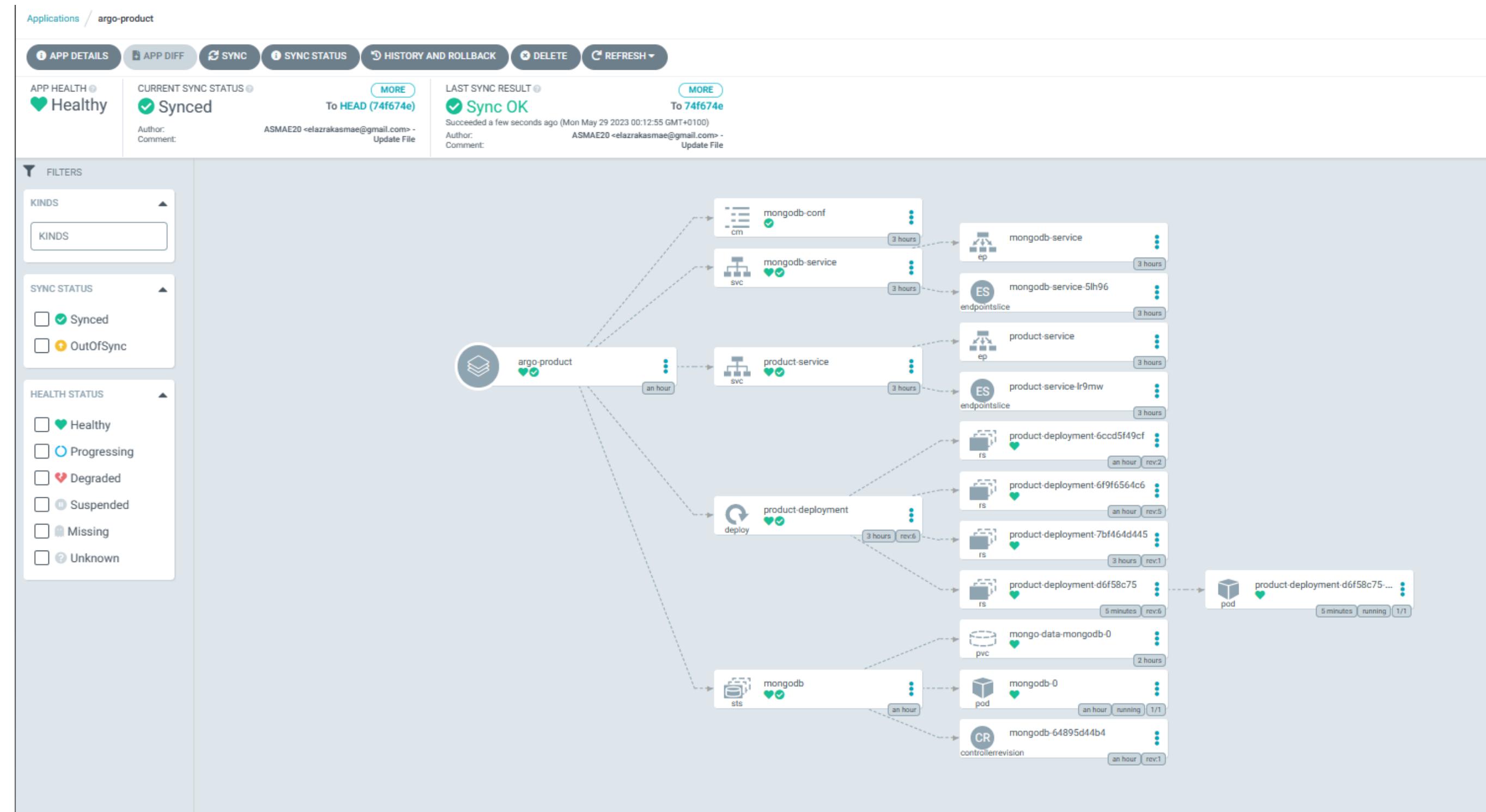
Après avoir exécuté le fichier, nous pouvons constater que notre microservice est affiché dans ArgoCD

The screenshot shows the ArgoCD application dashboard. On the left, there's a sidebar with icons for new app, sync apps, search, filters, sync status, and health status. The main area shows a list of applications. One application, 'argo-product', is highlighted. Its details are shown in a modal: Project: default, Label: Synced, Status: Progressing (OutOfSync), Repository: https://gitlab.com/Asmae20..., Target: HEAD, Path: file-k8s, Destination: in-cluster, Name: product-local. There are buttons for SYNC, COPY, and DELETE.

This screenshot shows the logs for a specific pod named 'product-deployment-d6f58c75-cg7ls'. The logs tab is selected. The log output shows two entries: 'Server listening on 5001' and 'Connected to database'. There are buttons for SYNC, DELETE, and other log-related actions like COPY, FOLLOW, DOWNLOAD, and a filter string.

This screenshot shows the summary for the same pod. It provides a quick overview of the pod's details: Kind: Pod, Name: product-deployment-d6f58c75-cg7ls, Namespace: m-commerce, Created At: 05/29/2023 00:07:56, Images: asmaael/product-microservice:1.5, and State: Running.

# ARGOCD



# ARGOCD

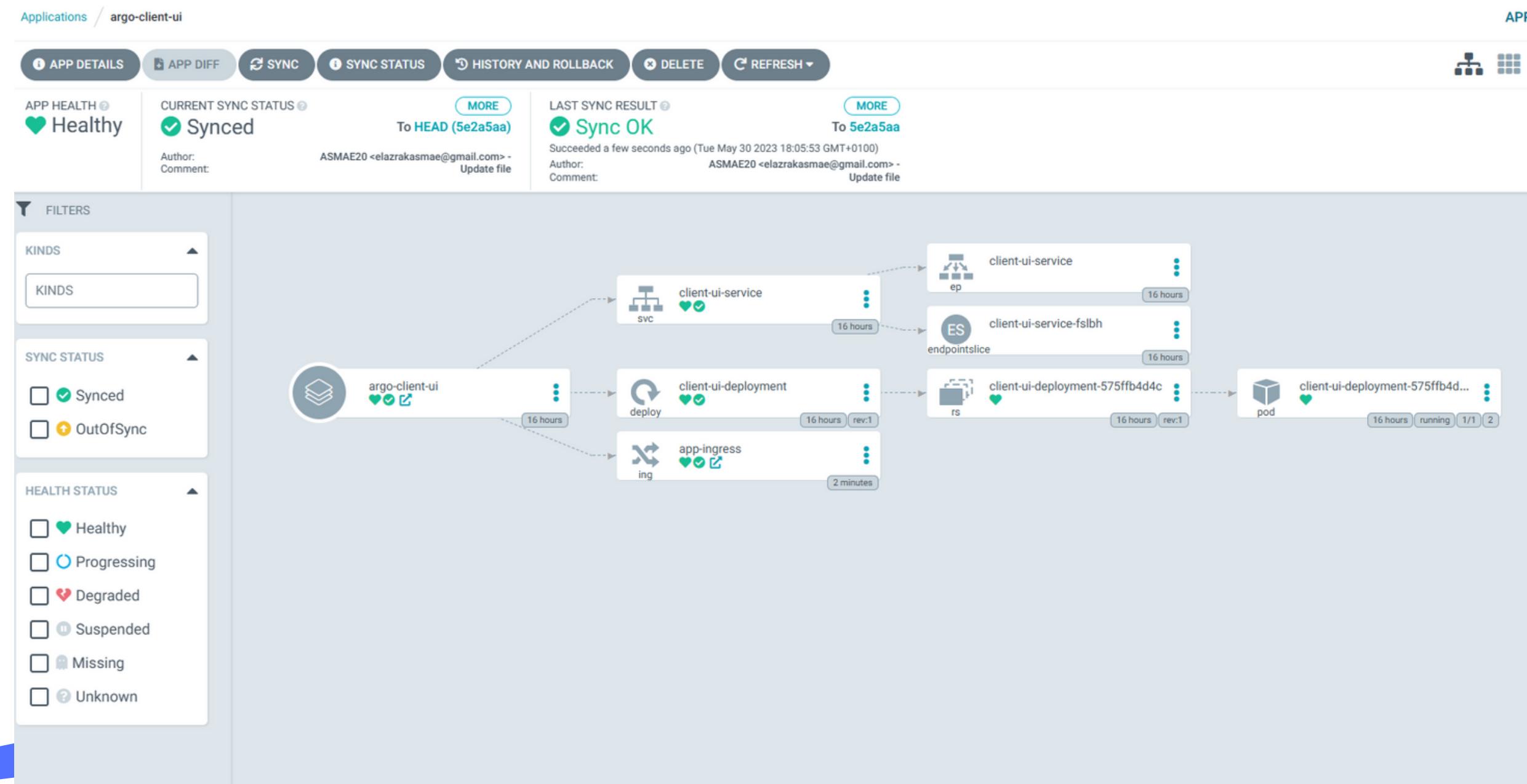
Et nous répétons le même processus pour tous les autres microservices.

The screenshot shows the ArgoCD web interface with the following details:

- Header:** Applications, APPLICATIONS, Log in
- Filters (Left Sidebar):**
  - SYNC STATUS:** Unknown (0), Synced (2), OutOfSync (4)
  - HEALTH STATUS:** Unknown (0), Progressing (0), Suspended (0), Healthy (6), Degraded (0), Missing (0)
  - LABELS:** Labels input field
  - PROJECTS:** Projects input field
  - CLUSTERS:** Clusters input field
- Applications (Main Area):**
  - argo-client-ui**: Project: default, Labels: None, Status: Healthy, Synced, Syncing, Repository: https://gitlab.com/Asmae20/client-ui..., Target R...: HEAD, Path: file-k8s, Destination: in-cluster, Names...: client-ui-local. Buttons: SYNC, REFRESH, DELETE.
  - argo-email**: Project: default, Labels: None, Status: Healthy, Synced, Syncing, Repository: https://gitlab.com/Asmae20/email-mi..., Target R...: HEAD, Path: file-k8s, Destination: in-cluster, Names...: email-local. Buttons: SYNC, REFRESH, DELETE.
  - argo-gateway**: Project: default, Labels: None, Status: Healthy, OutOfSync, Syncing, Repository: https://gitlab.com/Asmae20/gateway..., Target R...: HEAD, Path: file-k8s, Destination: in-cluster, Names...: gateway-local. Buttons: SYNC, REFRESH, DELETE.
  - argo-order**: Project: default, Labels: None, Status: Healthy, OutOfSync, Syncing, Repository: https://gitlab.com/Asmae20/order-mi..., Target R...: HEAD, Path: file-k8s, Destination: in-cluster, Names...: order-local. Buttons: SYNC, REFRESH, DELETE.
  - argo-payment**: Project: default, Labels: None, Status: Healthy, OutOfSync, Syncing, Repository: https://gitlab.com/Asmae20/payment..., Target R...: HEAD, Path: file-k8s, Destination: in-cluster, Names...: payment-local. Buttons: SYNC, REFRESH, DELETE.
  - argo-product**: Project: default, Labels: None, Status: Healthy, OutOfSync, Syncing, Repository: https://gitlab.com/Asmae20/product..., Target R...: HEAD, Path: file-k8s, Destination: in-cluster, Names...: product-local. Buttons: SYNC, REFRESH, DELETE.
- Bottom Right:** Items per page: 10

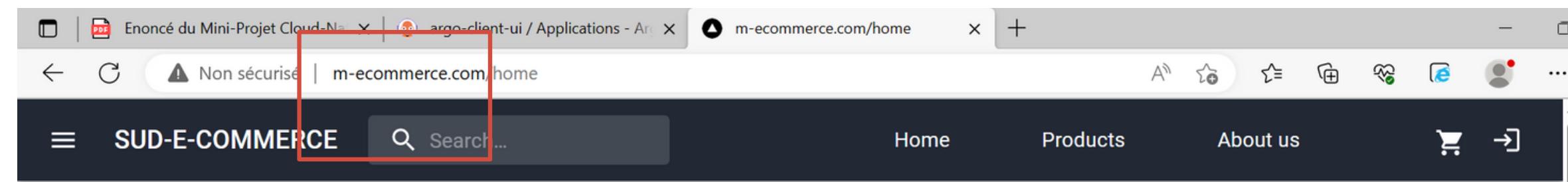
# ARGOCD

Et pour le frontend, étant donné que nous exposons le pod vers l'extérieur, nous avons un schéma similaire à celui montré dans cette image



# ARGOCD

Et voilà, on peut accéder à l'application via le nom de domaine

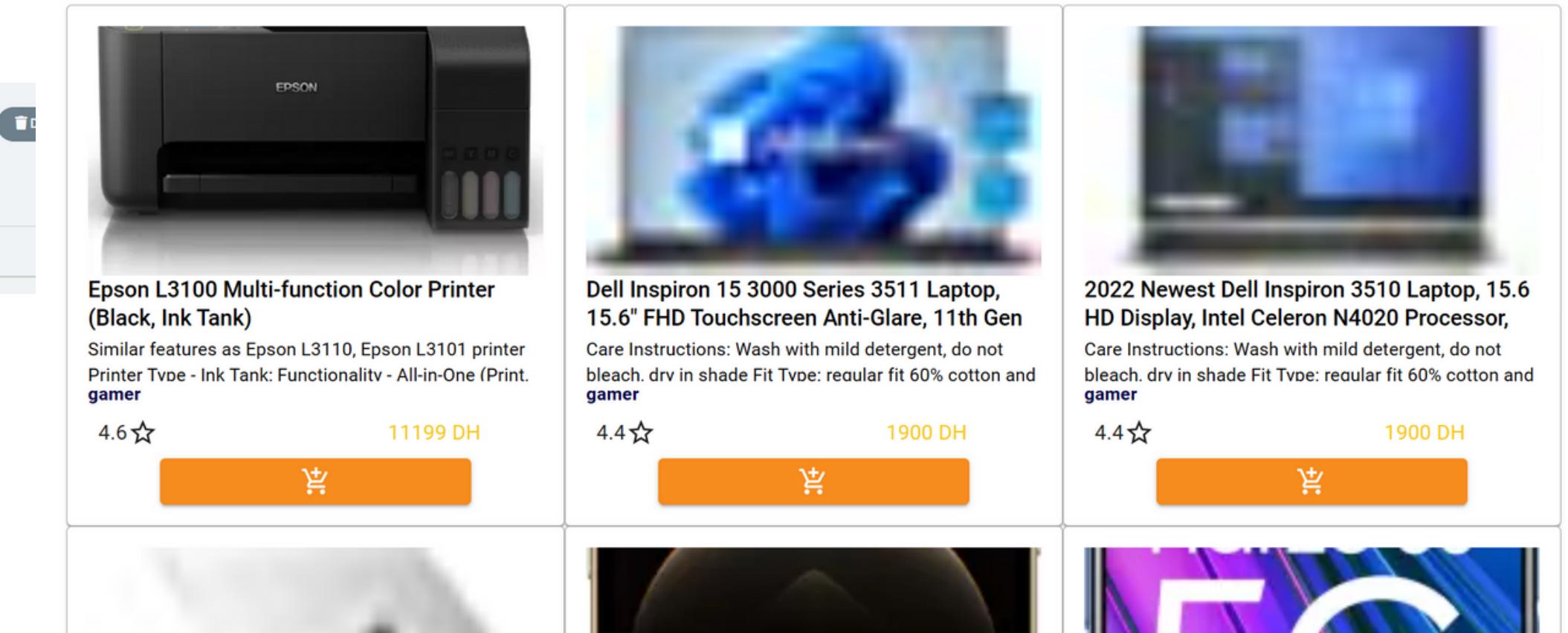
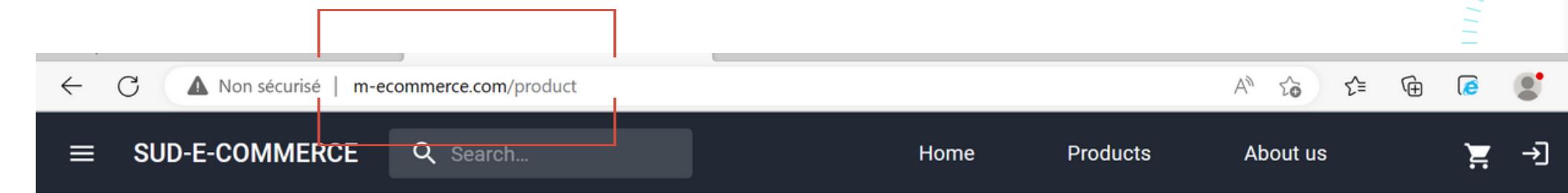
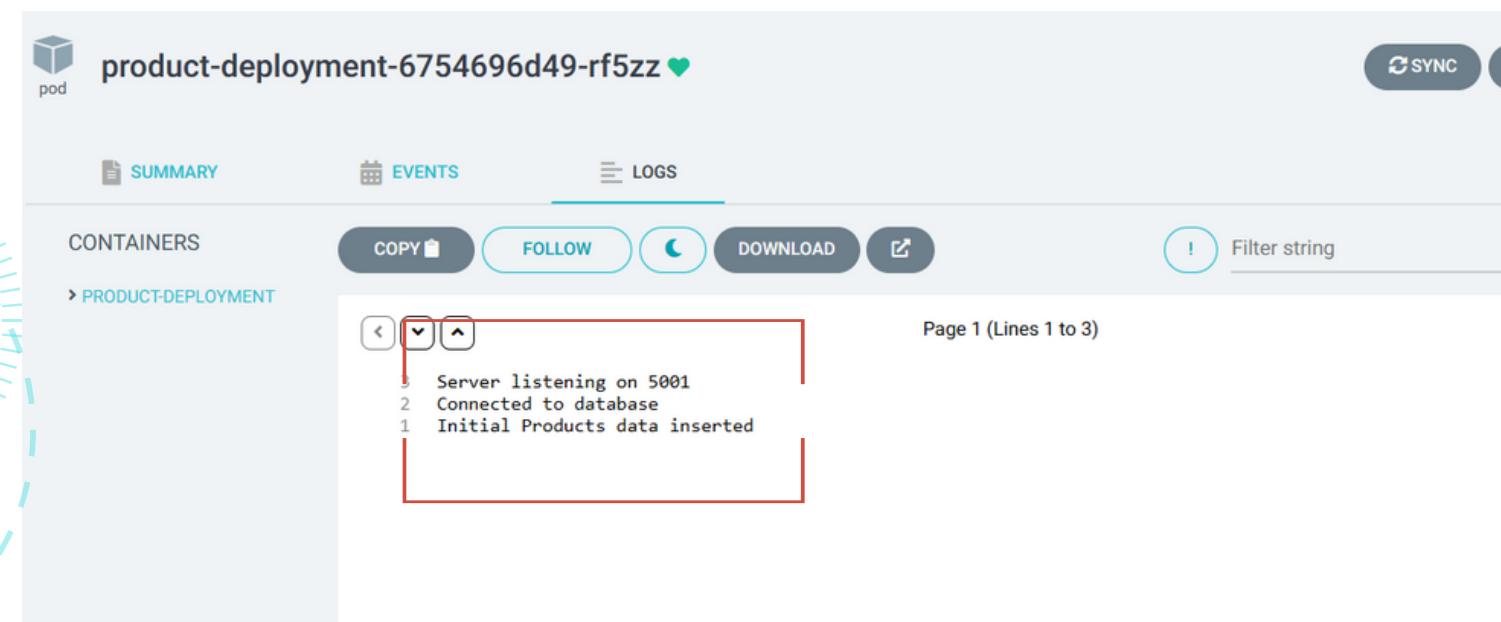


Explore our wide range of products!



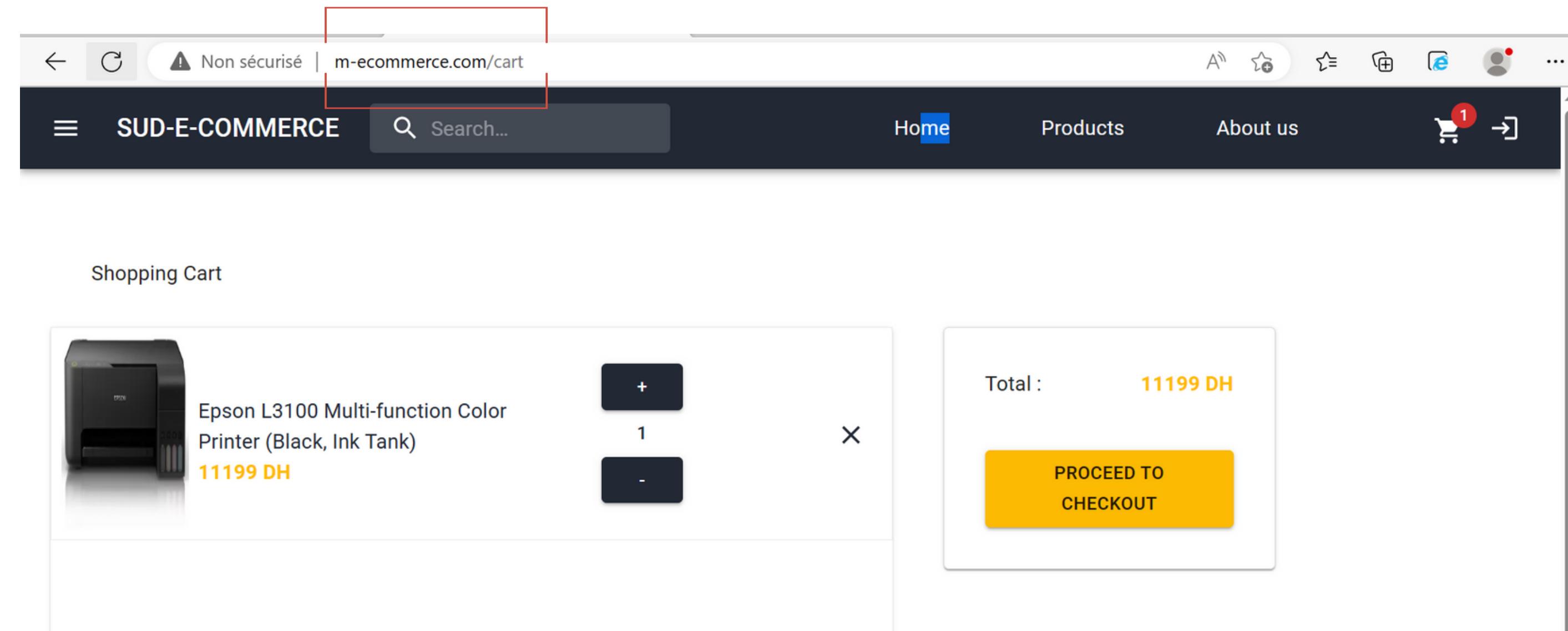
# ARGOCD

La même chose pour product et les autres microservices



# ARGOCD

La mem chose pour la commande



Istio



Istio

# STRATÉGIE DE DÉPLOIEMENT (CANARY DEPLOYS)

Dans le monde en constante évolution des déploiements logiciels, l'utilisation du service mesh Istio se révèle être une solution prometteuse pour garantir une stratégie de déploiement robuste, notamment avec des pratiques telles que les déploiements en mode Canary.

La stratégie de déploiement Canary, également connue sous le nom de déploiement progressif ou de déploiement en étapes, est une approche permettant de tester une nouvelle version d'une application ou d'un microservice de manière contrôlée et graduée. Elle consiste à acheminer une partie du trafic vers la nouvelle version tout en maintenant une partie du trafic vers l'ancienne version.

Dans notre cas précis, nous allons suivre une démarche similaire en modifiant le code de notre application et en créant une nouvelle version. Nous allons ensuite utiliser Istio pour acheminer un pourcentage spécifique des requêtes vers cette nouvelle version, tout en maintenant le reste du trafic vers l'ancienne version

# STRATÉGIE DE DÉPLOIEMENT (CANARY DEPLOYS)

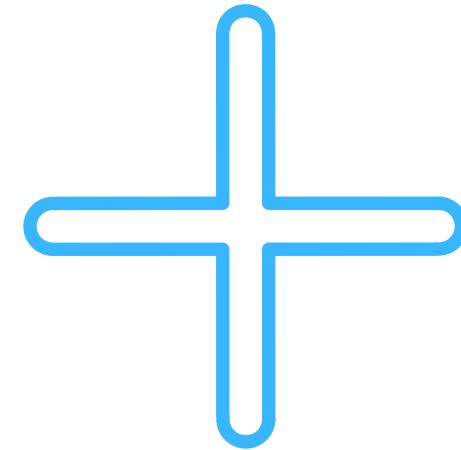
Nous apportons des modifications dans le code en remplaçant l'image et en modifiant une phrase spécifique.

```
1 import type { NextPage } from "next";
2 // import Head from "next/head";
3 import { Container, Typography } from "@mui/material";
4 import Link from "next/link";
5 import Image from "next/image";
6 import HomeImage from "../public/images/home2.png";  
7
8 const Home: NextPage = () => [  
9   <Container sx={{ pt: 9 }}>  
10    <header />  
11
12    <Container component="main">
13      <Typography variant="h5" gutterBottom>
14        Explore our wide range of products version 2!
15      </Typography>
16      <Image src={HomeImage} />
17
18      <Link href="/product"> explore our products</Link>
19    </Container>
20  </Container>
21];
22
23 export default Home;
24
```

# STRATÉGIE DE DÉPLOIEMENT (CANARY DEPLOYS)

Après on ajoute un autre déploiement avec l'image de la nouvelle version

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: client-ui-v1
  namespace: m-ecommerce
spec:
  selector:
    matchLabels:
      app: client-ui
  replicas: 1
  template:
    metadata:
      labels:
        app: client-ui
        version: v1
    spec:
      containers:
        - name: client-ui
          image: asmaael/client-ui-microservice:1.5
          resources:
            limits:
              cpu: 500m
              memory: 256Mi
            requests:
              cpu: 200m
              memory: 128Mi
          ports:
            - containerPort: 3000
              name: client-ui-port
      env:
        - name: NEXT_PUBLIC_API_URL
          value: "http://gateway-service:5005/api"
```

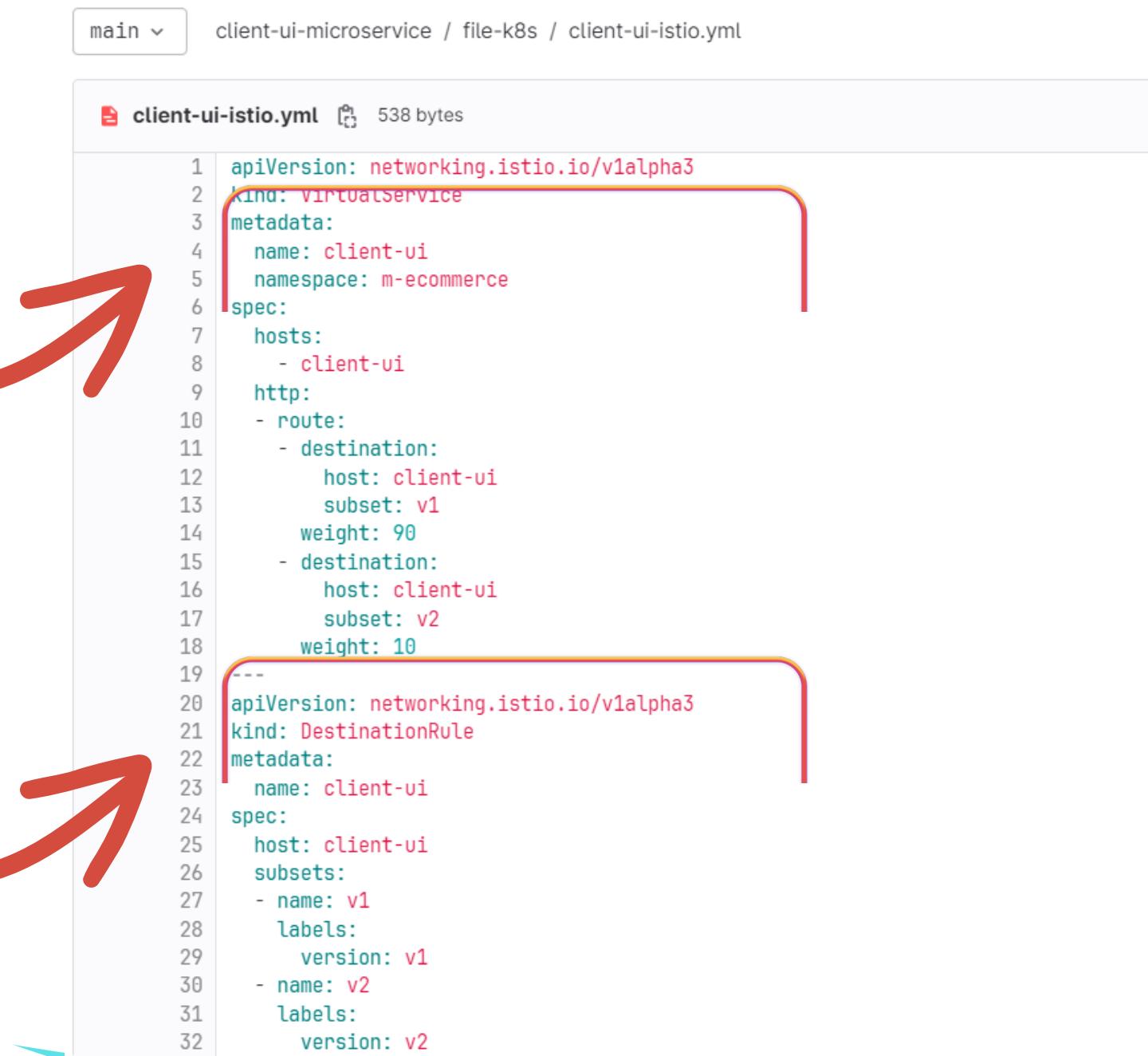


```
---  
apiVersion: apps/v1
kind: Deployment
metadata:
  name: client-ui-v2
  namespace: m-ecommerce
spec:
  selector:
    matchLabels:
      app: client-ui
  replicas: 1
  template:
    metadata:
      labels:
        app: client-ui
        version: v2
    spec:
      containers:
        - name: client-ui
          image: asmaael/client-ui-microservice:1.6
          resources:
            limits:
              cpu: 500m
              memory: 256Mi
            requests:
              cpu: 200m
              memory: 128Mi
          ports:
            - containerPort: 3000
              name: client-ui-port
      env:
        - name: NEXT_PUBLIC_API_URL
          value: "http://gateway-service:5005/api"
```

# STRATÉGIE DE DÉPLOIEMENT (CANARY DEPLOYS)

Après on définit un VirtualService et une DestinationRule dans Istio pour la ressource "client-ui" dans le namespace "m-ecommerce".

- Le VirtualService contrôle la distribution du trafic vers différentes versions (subsets) de "client-ui" en utilisant des poids de répartition. Dans cet exemple, 90% du trafic est dirigé vers la version "v1" de "client-ui", tandis que 10% du trafic est dirigé vers la version "v2" de "client-ui".
- Le DestinationRule définit les sous-ensembles de "client-ui" correspondant aux différentes versions. Les sous-ensembles sont étiquetés avec les noms "v1" et "v2", et ils permettent d'appliquer des politiques de trafic spécifiques à chaque version.



```
main ▾ client-ui-microservice / file-k8s / client-ui-istio.yml

client-ui-istio.yml 538 bytes
client-ui-istio.yml
1 apiVersion: networking.istio.io/v1alpha3
2 kind: VirtualService
3 metadata:
4   name: client-ui
5   namespace: m-ecommerce
6 spec:
7   hosts:
8     - client-ui
9     http:
10       route:
11         - destination:
12           host: client-ui
13             subset: v1
14             weight: 90
15         - destination:
16           host: client-ui
17             subset: v2
18             weight: 10
19
20 --- apiVersion: networking.istio.io/v1alpha3
21 kind: DestinationRule
22 metadata:
23   name: client-ui
24 spec:
25   host: client-ui
26   subsets:
27     - name: v1
28       labels:
29         version: v1
30     - name: v2
31       labels:
32         version: v2
```

# STRATÉGIE DE DÉPLOIEMENT (CANARY DEPLOYS)

Après on fait l'autoscalling pour les deux version

```
HP@Asmae-EL MINGW64 ~/Desktop/microservice-client-ui/file-k8s (main)
$ kubectl autoscale deployment client-ui-v1 --cpu-percent=50 --min=1 --max=10 -n m-e
commerce
horizontalpodautoscaler.autoscaling/client-ui-v1 autoscaled

HP@Asmae-EL MINGW64 ~/Desktop/microservice-client-ui/file-k8s (main)
$ kubectl autoscale deployment client-ui-v2 --cpu-percent=50 --min=1 --max=10 -n m-e
commerce
horizontalpodautoscaler.autoscaling/client-ui-v2 autoscaled

HP@Asmae-EL MINGW64 ~/Desktop/microservice-client-ui/file-k8s (main)
$ |
```

# STRATÉGIE DE DÉPLOIEMENT (CANARY DEPLOYS)

On récupère des informations sur les objets HorizontalPodAutoscaler (HPA) dans le namespace "m-ecommerce".

L'objet HorizontalPodAutoscaler est utilisé pour ajuster automatiquement le nombre de pods d'un déploiement en fonction de la charge ou de la demande de l'application

```
HP@Asmae-EL MINGW64 ~/Desktop/microservice-client-ui/file-k8s (main)
$ kubectl get hpa -n m-ecommerce
NAME          REFERENCE           TARGETS      MINPODS   MAXPODS   REPLICAS
client-ui-v1  Deployment/client-ui-v1  <unknown>/50%  1         10        1
              43s
client-ui-v2  Deployment/client-ui-v2  <unknown>/50%  1         10        1
              30s
```

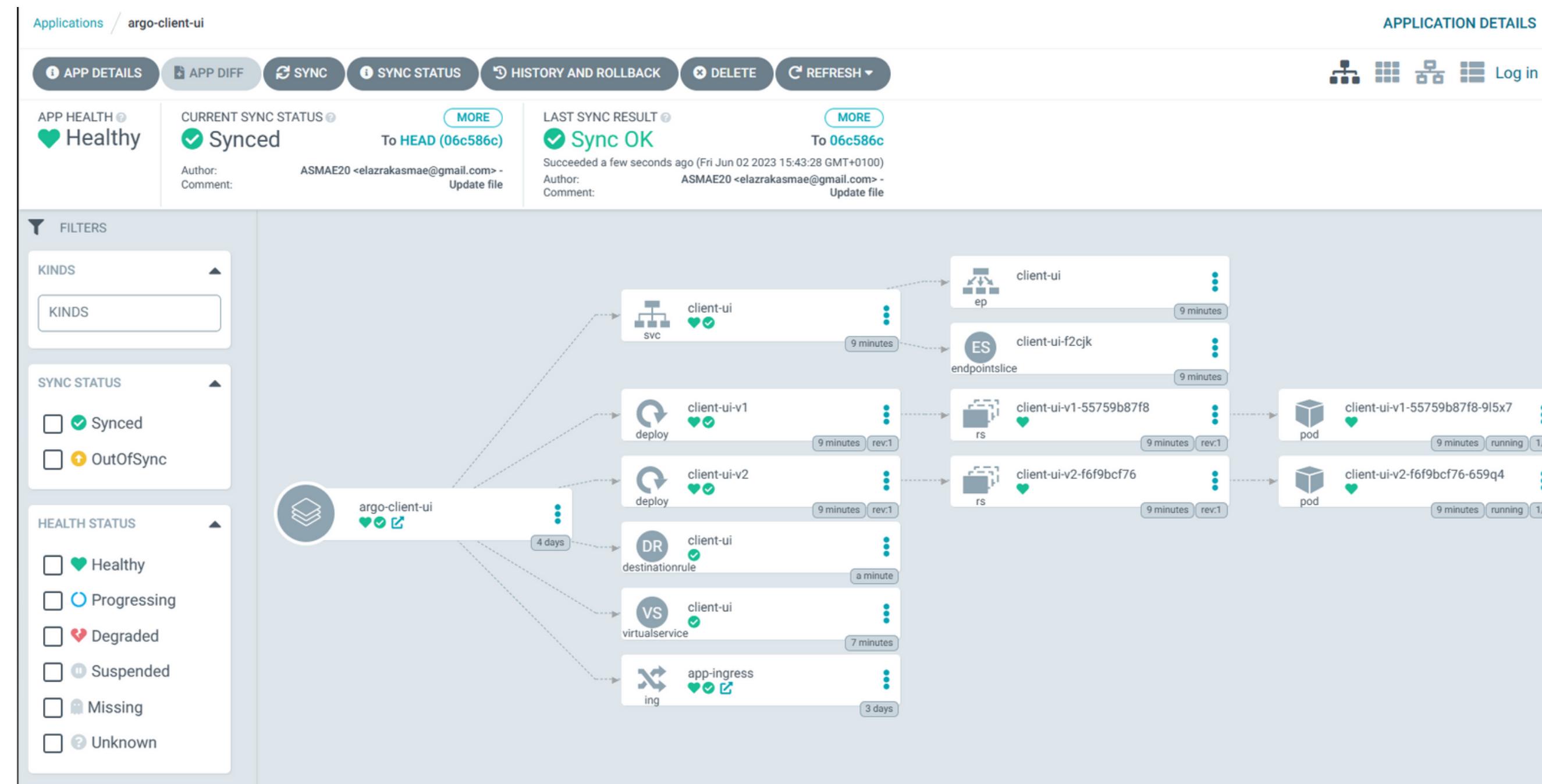
Et on vérifie que les pods fonctionnent

```
HP@Asmae-EL MINGW64 ~/Desktop/microservice-client-ui/file-k8s (main)
$ kubectl get pods -n m-ecommerce | grep client-ui
client-ui-deployment-65697fcdd-zrp5g  1/1     Running   2 (20m ago)  28m
client-ui-v1-55759b87f8-915x7         1/1     Running   0          4m14s
client-ui-v2-f6f9bcf76-659q4         1/1     Running   0          4m14s

HP@Asmae-EL MINGW64 ~/Desktop/microservice-client-ui/file-k8s (main)
```

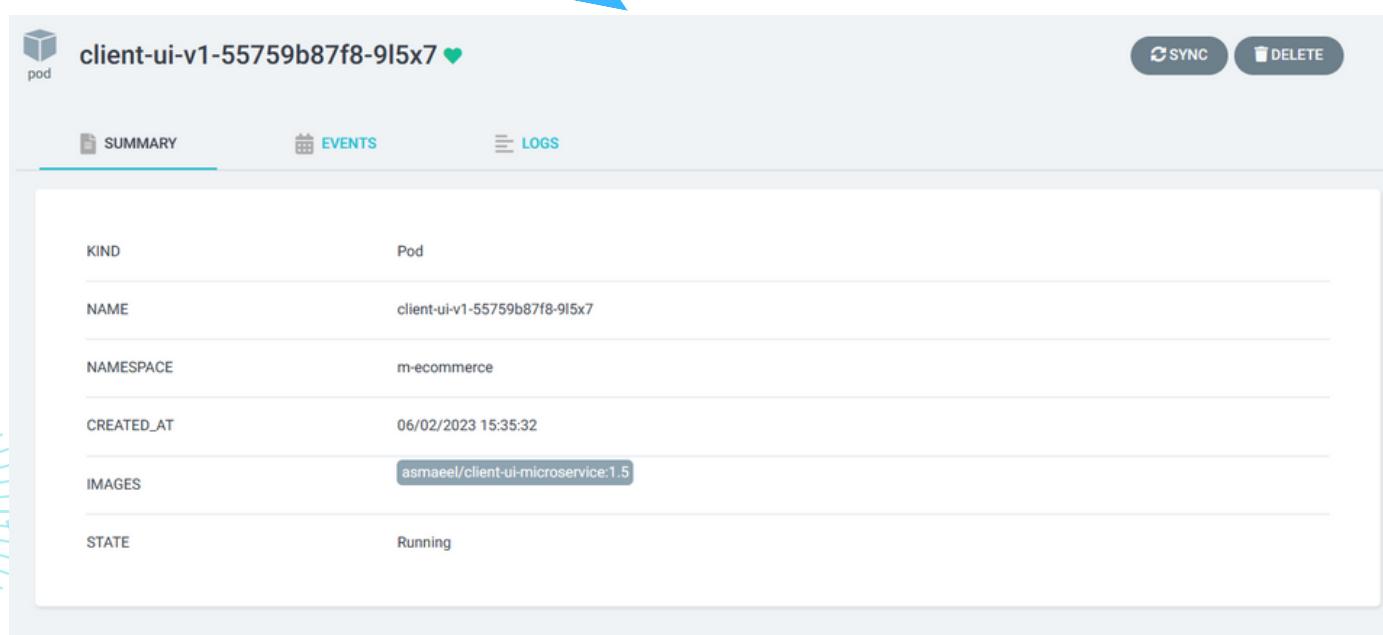
# STRATÉGIE DE DÉPLOIEMENT (CANARY DEPLOYS)

Après avoir relancé l'application dans ArgoCD, nous obtenons ce nouveau schéma.

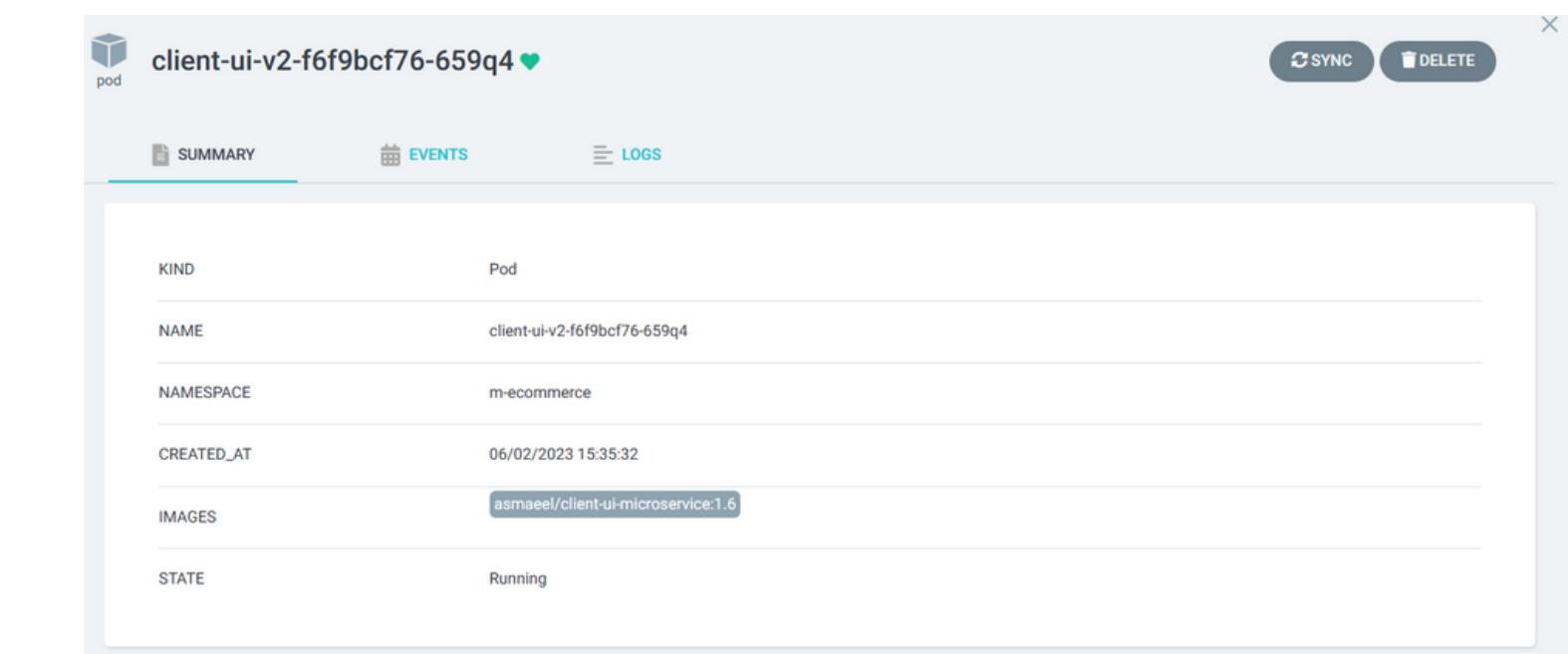


# STRATÉGIE DE DÉPLOIEMENT (CANARY DEPLOYS)

Après avoir relancé l'application dans ArgoCD, nous obtenons ce nouveau schéma.

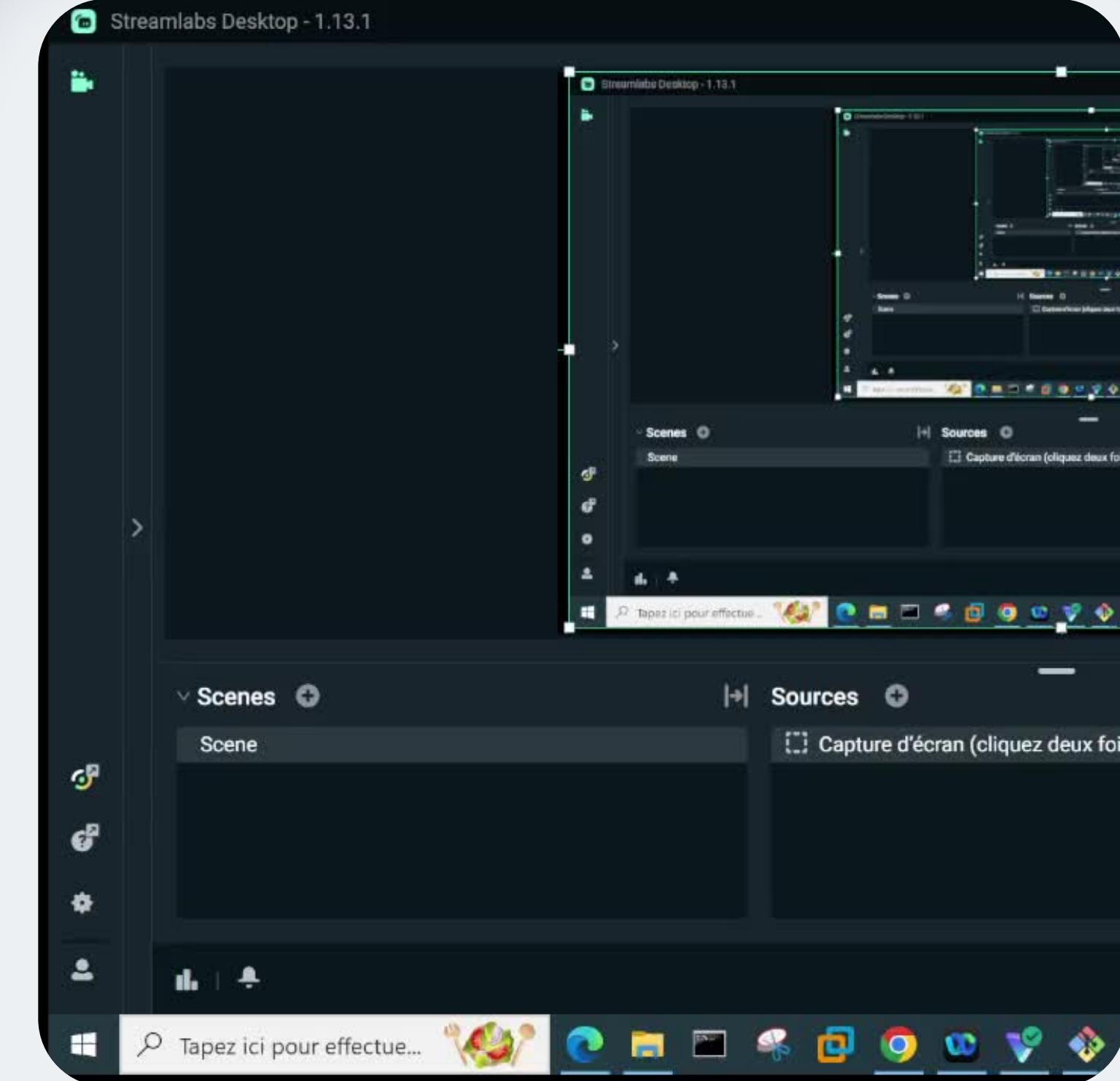


A screenshot of a Kubernetes pod summary interface. The pod name is "client-ui-v1-55759b87f8-9l5x7". The "SUMMARY" tab is selected. The pod kind is listed as "Pod". The pod name is "client-ui-v1-55759b87f8-9l5x7". It is located in the "m-commerce" namespace and was created at "06/02/2023 15:35:32". The image used is "asmaeel/client-ui-microservice:1.5" and the state is "Running". There are "SYNC" and "DELETE" buttons at the top right.



A screenshot of a Kubernetes pod summary interface. The pod name is "client-ui-v2-f6f9bcf76-659q4". The "SUMMARY" tab is selected. The pod kind is listed as "Pod". The pod name is "client-ui-v2-f6f9bcf76-659q4". It is located in the "m-commerce" namespace and was created at "06/02/2023 15:35:32". The image used is "asmaeel/client-ui-microservice:1.6" and the state is "Running". There are "SYNC" and "DELETE" buttons at the top right.

# Demo Canary deploys



# Istio Observability



# OUTILS D'OBSERVABILITÉ DE ISTIO

Il existe plusieurs addons ou outils d'observabilité que vous pouvez utiliser pour améliorer les capacités d'observabilité de votre architecture de microservices. Ces addons fournissent des fonctionnalités supplémentaires et des visualisations pour la surveillance, la traçabilité et les journaux. Voici quelques addons d'observabilité populaires qui peuvent être intégrés à Istio :

- Prometheus : Prometheus est un outil de surveillance et d'alerte. Il collecte et stocke les données de métriques provenant des composants et des services Istio, et fournit un puissant langage de requête pour analyser et visualiser les métriques. Prometheus peut vous aider à surveiller la santé et les performances de vos microservices.
- Grafana : Grafana est une plateforme de visualisation et d'analyse qui peut être intégrée à Prometheus pour créer des tableaux de bord interactifs et personnalisables. Elle vous permet de créer des représentations visuelles de vos données de métriques, de configurer des alertes et de surveiller les performances et le comportement de vos microservices en temps réel.
- Jaeger : Jaeger est un système de traçabilité distribuée qui peut être intégré à Istio pour fournir des capacités de traçage de bout en bout. Il vous aide à comprendre le flux des requêtes entre vos microservices, à identifier les problèmes de latence et à résoudre les problèmes liés aux interactions entre les services. Jaeger offre une interface de visualisation complète pour explorer et analyser les traces.
- Kiali : Kiali est une console de gestion pour Istio qui offre des fonctionnalités d'observabilité et de visualisation. Il fournit une interface graphique pour visualiser la topologie du maillage de services, surveiller les flux de trafic et analyser les configurations d'Istio. Kiali vous aide à comprendre les relations entre les services, à suivre le trafic des requêtes et à identifier les problèmes potentiels.

**Manifest kubernetes pour mise en place des pod d'observabilité, qui vient avec le package d'installation istio**

```
riad@riad-Inspiron-3501:~/istio-1.17.2/samples$ cd addons/
riad@riad-Inspiron-3501:~/istio-1.17.2/samples/addons$ ls
extras  grafana.yaml  jaeger.yaml  kiali.yaml  prometheus.yaml  README.md
riad@riad-Inspiron-3501:~/istio-1.17.2/samples/addons$
```

# OUTILS D'OBSERVABILITÉ DE ISTIO

Les pods d'observabilité, sont en marche

```
No resources found in system-istio namespace.  
riad@riad-Inspiron-3501:~$ kubectl get pods -n istio-system  
NAME          READY   STATUS    RESTARTS   AGE  
grafana-69f9b6bfdc-xzsvn   1/1     Running   2 (154m ago)   22h  
istio-egressgateway-676bf68b54-8fm6h   1/1     Running   2 (154m ago)   22h  
istio-ingressgateway-8d56c999d-h5449   1/1     Running   2 (154m ago)   22h  
istiod-dbf5ff64-xx5sl      1/1     Running   2 (154m ago)   22h  
jaeger-cc4688b98-s6x8p       1/1     Running   2 (154m ago)   22h  
kiali-594965b98c-zddsl      1/1     Running   2 (154m ago)   22h  
prometheus-5f84bbfcfd-jf4x5     2/2     Running   4 (154m ago)   22h  
riad@riad-Inspiron-3501:~$
```

Pour lancer le dashboard d'un outil on execute la command suivant avec nom de l'outil : istioctl dashboard tool-name

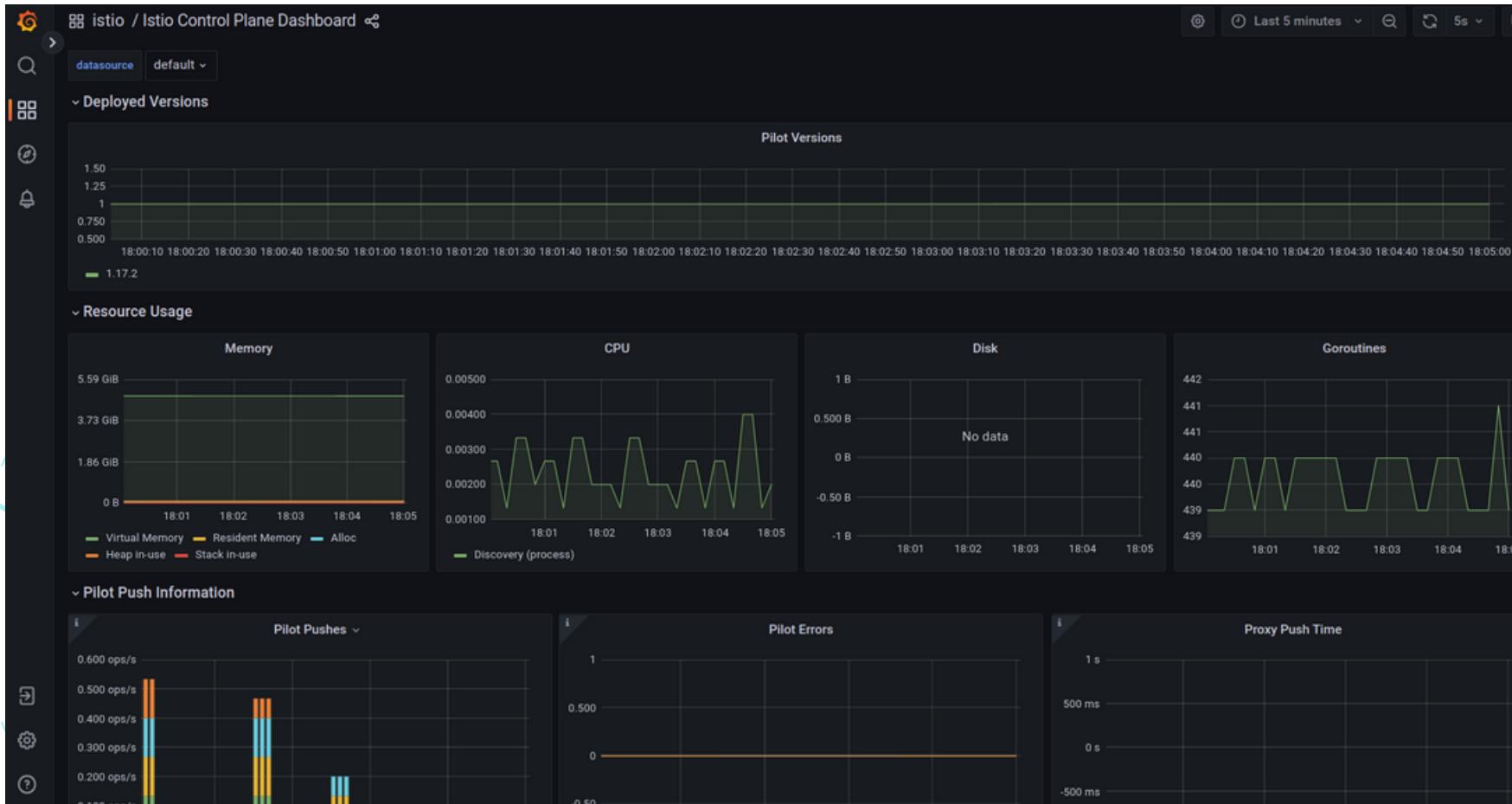
```
riad@riad-Inspiron-3501:~/istio-1.17.2/bin$ istioctl dashboard grafana  
http://localhost:3000
```

Dans Grafana on peut voir les metriques en tem reel

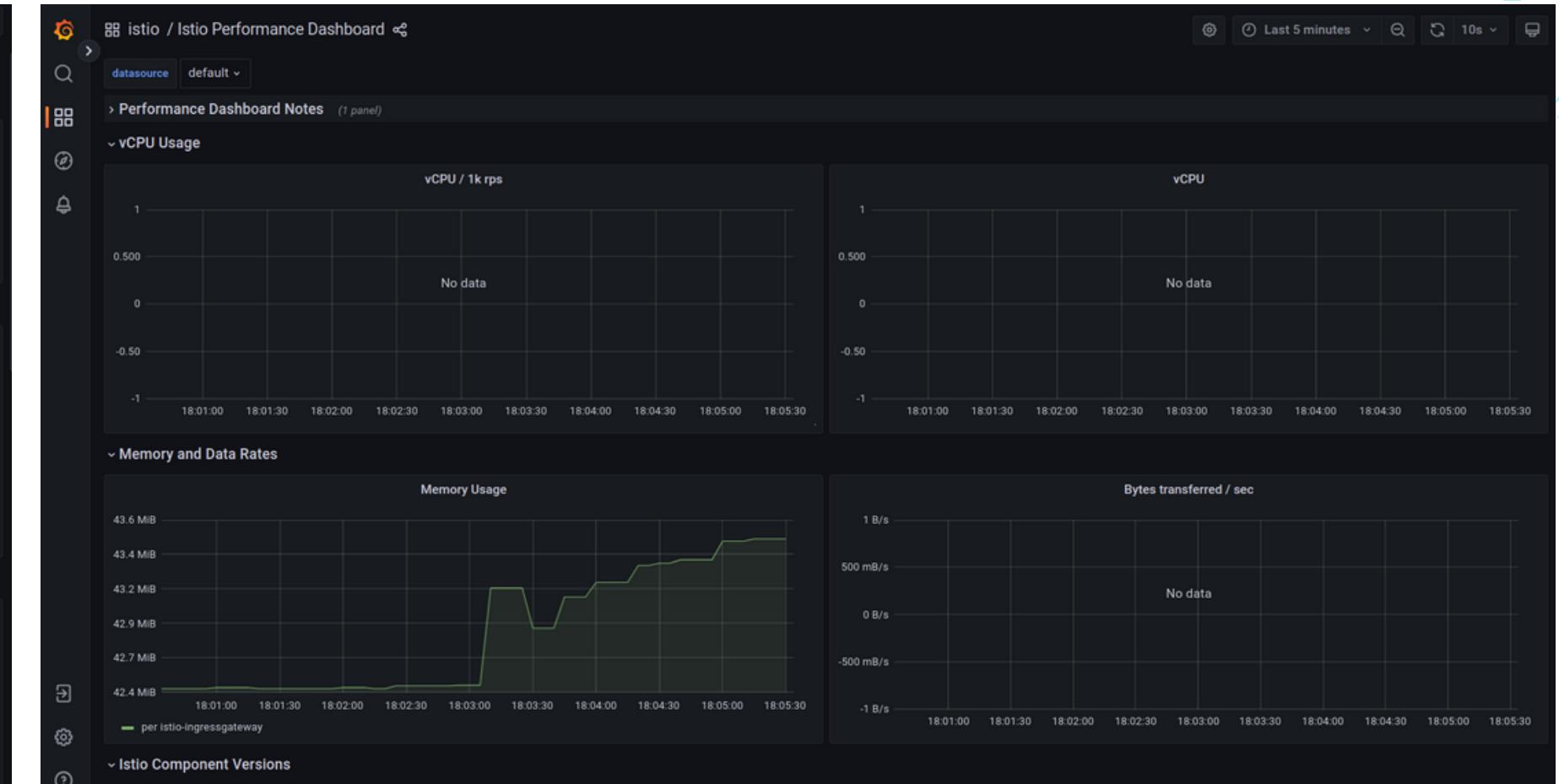
The screenshot shows the Grafana web interface with a dark theme. The top navigation bar includes a back arrow, forward arrow, refresh icon, and a search bar with the URL 'localhost:3000/dashboards/f/WN8wUDlVk/istio'. Below the header, there's a sidebar with icons for dashboard management, search, and other settings. The main area is titled 'Dashboards / istio' and contains a list of six items under the 'Dashboards' tab. Each item has a small preview thumbnail, the name, and a 'istio' tag icon. The names are: Istio Control Plane Dashboard, Istio Mesh Dashboard, Istio Performance Dashboard, Istio Service Dashboard, Istio Wasm Extension Dashboard, and Istio Workload Dashboard.

# OUTILS D'OBSERVABILITÉ DE ISTIO

## Observer Istio Control Plane



## Observer Istio performance



# OUTILS D'OBSERVABILITÉ DE ISTIO

## Prometheus dashboard

The screenshot shows the Prometheus web interface with a search bar containing 'apiserver\_request\_total'. The results table displays a list of metrics with their values. The table includes columns for metric name, labels, and value. The first few rows are as follows:

| Metric                  | Labels  | Value |
|-------------------------|---|-------|
| apiserver_request_total | code="0", component="apiserver", instance="192.168.49.2:8443", job="kubernetes-apiservers", resource="pods", scope="resource", subresource="portforward", verb="POST", version="v1"   | 1905  |
| apiserver_request_total | code="200", component="apiserver", group="admissionregistration.k8s.io", instance="192.168.49.2:8443", job="kubernetes-apiservers", resource="mutatingwebhookconfigurations", scope="cluster", verb="LIST", version="v1"    | 4     |
| apiserver_request_total | code="200", component="apiserver", group="admissionregistration.k8s.io", instance="192.168.49.2:8443", job="kubernetes-apiservers", resource="mutatingwebhookconfigurations", scope="cluster", verb="WATCH", version="v1"   | 86    |
| apiserver_request_total | code="200", component="apiserver", group="admissionregistration.k8s.io", instance="192.168.49.2:8443", job="kubernetes-apiservers", resource="validatingwebhookconfigurations", scope="cluster", verb="LIST", version="v1"  | 3     |
| apiserver_request_total | code="200", component="apiserver", group="admissionregistration.k8s.io", instance="192.168.49.2:8443", job="kubernetes-apiservers", resource="validatingwebhookconfigurations", scope="cluster", verb="WATCH", version="v1" | 62    |
| apiserver_request_total | code="200", component="apiserver", group="admissionregistration.k8s.io", instance="192.168.49.2:8443", job="kubernetes-apiservers", resource="validatingwebhookconfigurations", scope="resource", verb="GET", version="v1"  | 14    |
| apiserver_request_total | code="200", component="apiserver", group="admissionregistration.k8s.io", instance="192.168.49.2:8443", job="kubernetes-apiservers", resource="validatingwebhookconfigurations", scope="resource", verb="PUT", version="v1"  | 4     |
| apiserver_request_total | code="200", component="apiserver", group="apiextensions.k8s.io", instance="192.168.49.2:8443", job="kubernetes-apiservers", resource="customresourcedefinitions", scope="cluster", verb="LIST", version="v1"                | 4     |
| apiserver_request_total | code="200", component="apiserver", group="apiextensions.k8s.io", instance="192.168.49.2:8443", job="kubernetes-apiservers", resource="customresourcedefinitions", scope="cluster", verb="WATCH", version="v1"               | 61    |
| apiserver_request_total | code="200", component="apiserver", group="apiregistration.k8s.io", instance="192.168.49.2:8443", job="kubernetes-apiservers", resource="apiservices", scope="cluster", verb="LIST", version="v1"                            | 2     |
| apiserver_request_total | code="200", component="apiserver", group="apiregistration.k8s.io", instance="192.168.49.2:8443", job="kubernetes-apiservers", resource="apiservices", scope="cluster", verb="WATCH", version="v1"                           | 41    |
| apiserver_request_total | code="200", component="apiserver", group="apiregistration.k8s.io", instance="192.168.49.2:8443", job="kubernetes-apiservers", resource="apiservices", scope="resource", verb="DELETE", version="v1"                         | 2     |
| apiserver_request_total | code="200", component="apiserver", group="apis", instance="192.168.49.2:8443", job="kubernetes-apiservers", resource="controllerrevisions", scope="cluster", verb="LIST", version="v1"                                      | 1     |
| apiserver_request_total | code="200", component="apiserver", group="apis", instance="192.168.49.2:8443", job="kubernetes-apiservers", resource="controllerrevisions", scope="cluster", verb="WATCH", version="v1"                                     | 22    |
| apiserver_request_total | code="200", component="apiserver", group="apis", instance="192.168.49.2:8443", job="kubernetes-apiservers", resource="daemonsets", scope="cluster", verb="LIST", version="v1"   | 5     |
| apiserver_request_total | code="200", component="apiserver", group="apis", instance="192.168.49.2:8443", job="kubernetes-apiservers", resource="daemonsets", scope="cluster", verb="WATCH", version="v1"  | 42    |
| apiserver_request_total | code="200", component="apiserver", group="apis", instance="192.168.49.2:8443", job="kubernetes-apiservers", resource="daemonsets", scope="resource", subresource="status", verb="PUT", version="v1"                         | 2     |
| apiserver_request_total | code="200", component="apiserver", group="apis", instance="192.168.49.2:8443", job="kubernetes-apiservers", resource="daemonsets", scope="resource", verb="PUT", version="v1"   | 1     |

## Requête Prometheus pour collecter les métriques

The screenshot shows the Metrics Explorer interface with a search bar and a list of metrics. The metrics listed include various metrics related to the Kubernetes API server, such as 'apiserver\_admission\_controller\_admission\_duration\_seconds\_bucket', 'apiserver\_admission\_step\_admission\_duration\_seconds\_bucket', and 'apiserver\_audit\_event\_total'.

| Metric  |
|---|
| apiserver_openapi_v2_regeneration_count                           |
| apiserver_openapi_v2_regeneration_duration                        |
| apiserver_unavailable_apiservice                                  |
| apiextensions_openapi_v2_regeneration_count                       |
| apiextensions_openapi_v3_regeneration_count                       |
| apiserver_admission_controller_admission_duration_seconds_bucket  |
| apiserver_admission_controller_admission_duration_seconds_count   |
| apiserver_admission_controller_admission_duration_seconds_sum     |
| apiserver_admission_step_admission_duration_seconds_bucket        |
| apiserver_admission_step_admission_duration_seconds_count         |
| apiserver_admission_step_admission_duration_seconds_sum           |
| apiserver_admission_step_admission_duration_seconds_summary       |
| apiserver_admission_step_admission_duration_seconds_summary_count |
| apiserver_admission_step_admission_duration_seconds_summary_sum   |
| apiserver_admission_webhook_admission_duration_seconds_bucket     |
| apiserver_admission_webhook_admission_duration_seconds_count      |
| apiserver_admission_webhook_admission_duration_seconds_sum        |
| apiserver_admission_webhook_fail_open_count                       |
| apiserver_admission_webhook_rejection_count                       |
| apiserver_admission_webhook_request_total                         |
| apiserver_audit_event_total                                       |

# OUTILS D'OBSERVABILITÉ DE ISTIO

## KIALI DASHBOARD

m-commerce

2 labels  
Istio config N/A  
8 applications 8

No inbound traffic

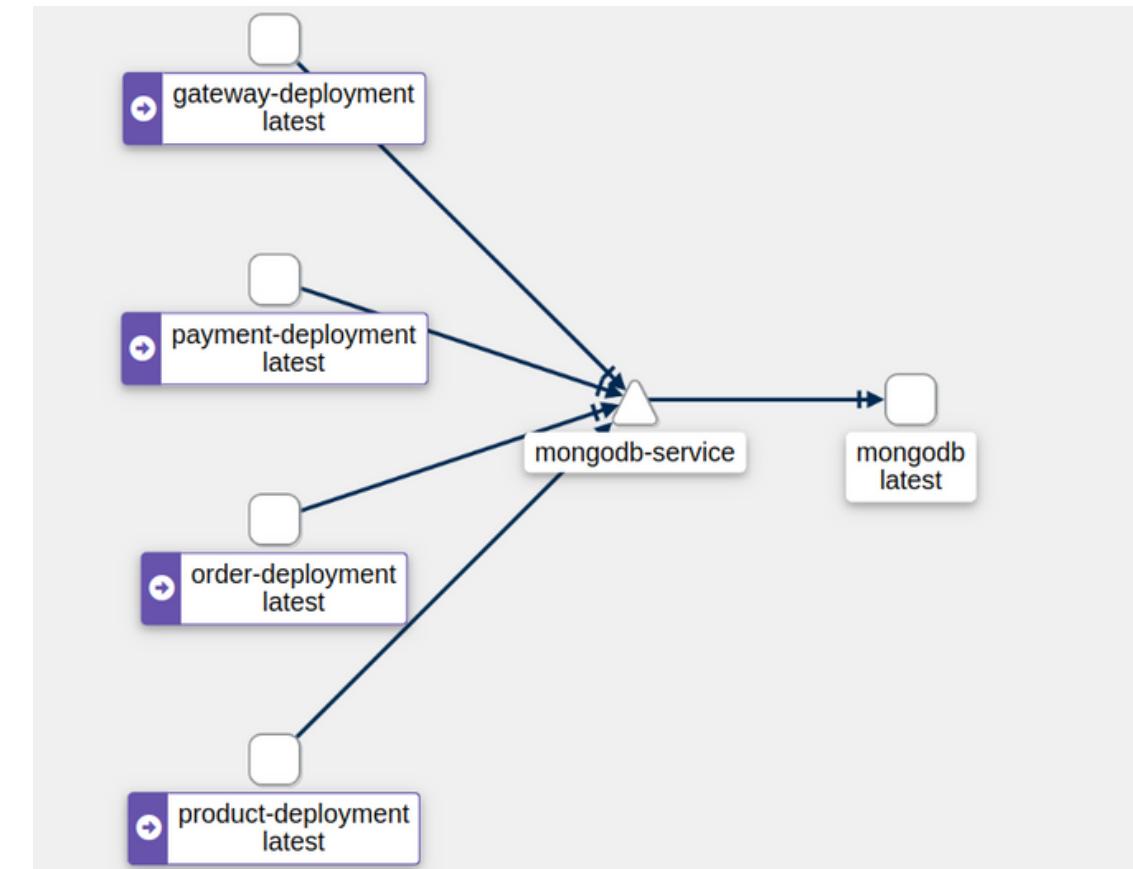
### KIALI workloads

Namespace: m-commerce

Last 1m | Every 10s | Refresh

| Workload Name | Filter by Workload Name |               |             |                        |                     |
|---------------|-------------------------|---------------|-------------|------------------------|---------------------|
| Health        | Name                    | Namespace     | Type        | Labels                 | Details             |
| ✓             | client-deployment       | NS m-commerce | Deployment  | app=client-deployment  | ✗ Missing Version ⓘ |
| ✓             | email-deployment        | NS m-commerce | Deployment  | app=email-deployment   | ✗ Missing Version ⓘ |
| ✓             | gateway-deployment      | NS m-commerce | Deployment  | app=gateway-deployment | ✗ Missing Version ⓘ |
| ✓             | mongodb                 | NS m-commerce | StatefulSet | app=mongodb            | ✗ Missing Version ⓘ |
| ✓             | order-deployment        | NS m-commerce | Deployment  | app=order-deployment   | ✗ Missing Version ⓘ |
| ✓             | payment-deployment      | NS m-commerce | Deployment  | app=payment-deployment | ✗ Missing Version ⓘ |
| ✓             | product-deployment      | NS m-commerce | Deployment  | app=product-deployment | ✗ Missing Version ⓘ |
| ✓             | rabbitmq                | NS m-commerce | Deployment  | app=rabbitmq           | ✗ Missing Version ⓘ |

## KIALI graph





# LIEN DU REPO

---

Product repo : <https://gitlab.com/Asmae20/product-microservice>

Gateway repo: <https://gitlab.com/Asmae20/gateway-microservice>

Payment repo : <https://gitlab.com/Asmae20/payment-microservice>

Cient-ui repo: <https://gitlab.com/Asmae20/client-ui-microservice>

Email repo : <https://gitlab.com/Asmae20/email-microservice>

Order repo: <https://gitlab.com/Asmae20/order-microservice>

Lien du video demo du canary strategy : [https://drive.google.com/file/d/18G6U8i-vB0U40ccDlcBojN-EuZcEppbx/view?usp=drive\\_link](https://drive.google.com/file/d/18G6U8i-vB0U40ccDlcBojN-EuZcEppbx/view?usp=drive_link)

MERCI D'AVOIR SUIVI CETTE  
PRÉSENTATION

