

DÉVELOPPEMENT ET DÉPLOIEMENT D'UNE CLOUD-NATIVE APPLICATION

FURNISHOP

Par OUSSAKEL Khadija



Encadré par :
Mr. ALLAKI Driss

Agenda

1

Introduction

2

Contexte général

3

User interfaces

4

Microservice frontend

- SPA avec React
- Conteneurisation du microservice
- Gitlab CI/CD pipeline
- Déploiement dans un cluster k8s

5

Microservice produit

- Développement avec Node.js/express et MongoDB
- Conteneurisation du microservice
- Gitlab CI/CD pipeline
- Déploiement dans un cluster k8s

6

Microservice commande

- Développement avec Node.js/express et MongoDB
- Conteneurisation du microservice
- Gitlab CI/CD pipeline
- Déploiement dans un cluster k8s

7

Microservice paiement

- Développement avec Node.js/express et MongoDB
- Conteneurisation du microservice
- Gitlab CI/CD pipeline
- Déploiement dans un cluster k8s

8

Gestion et déploiement des applications Kubernetes de manière continue avec ArgoCD

9

Conclusion

Introduction

Dans le développement d'une application cloud-native, l'utilisation de différents outils et technologies est cruciale pour simplifier le déploiement et la gestion. Parmi ces outils essentiels, on retrouve GitLab CI/CD, Docker, Kubernetes (K8s) et ArgoCD. En intégrant ces technologies et outils, nous serons en mesure de créer une application **FurniShop** solide, évolutive et facilement déployable, garantissant une expérience utilisateur fluide et une gestion efficace des microservices.

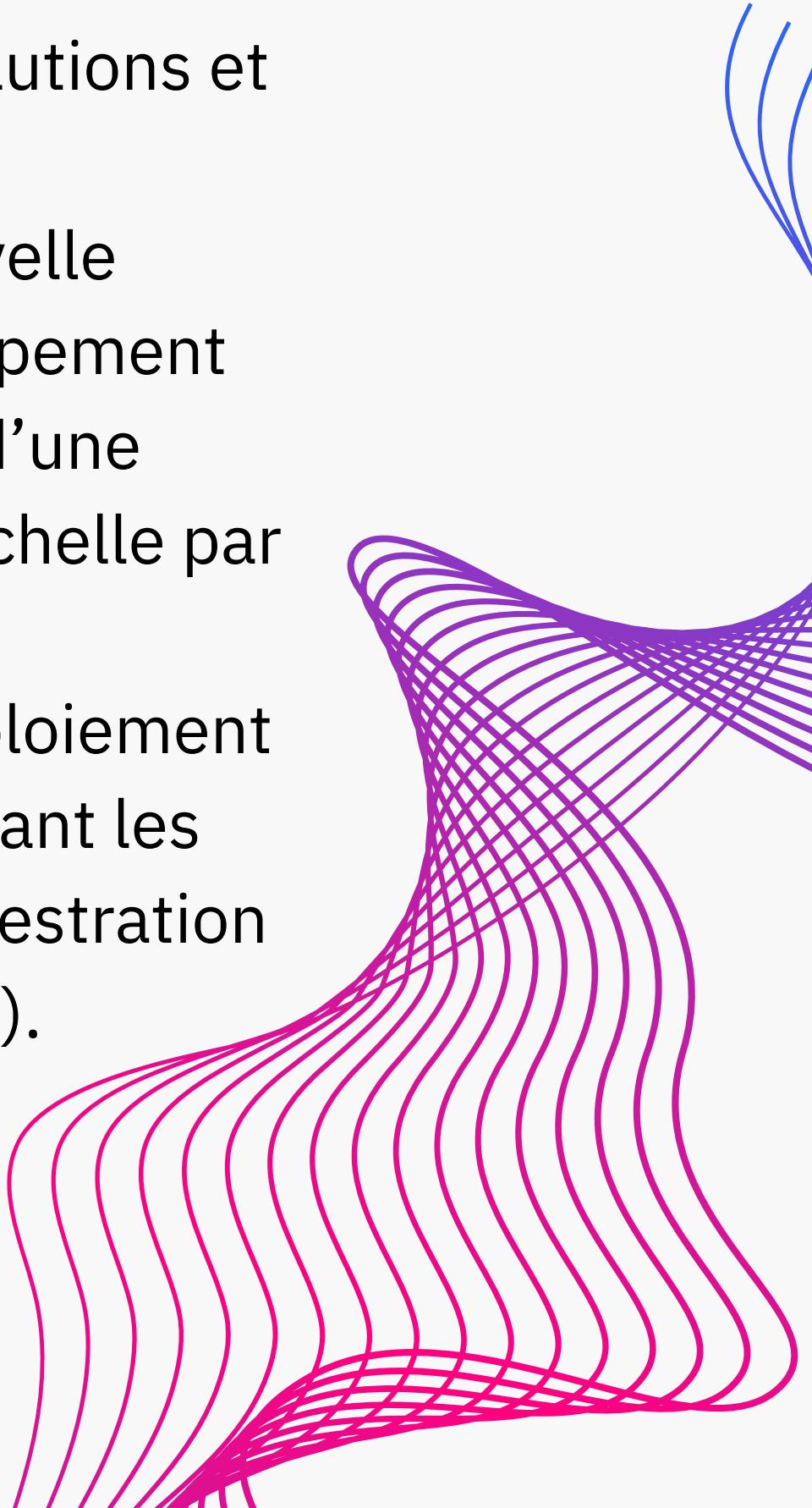


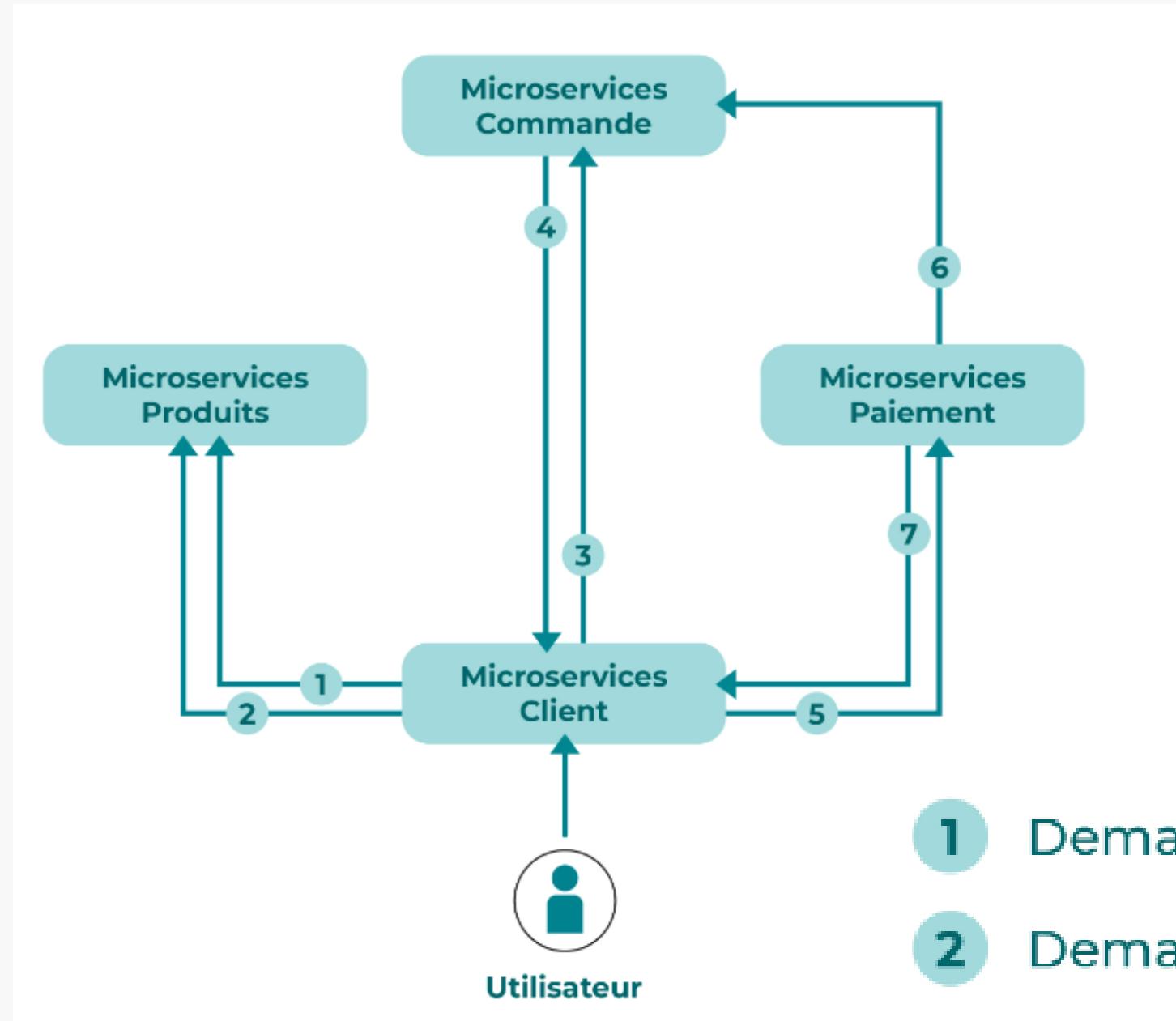
Contexte général

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.

Le sujet de notre mission concerne le développement et le déploiement d'une application microservices nommée Furnishop en utilisant les pratiques et technologies cloud-native (conteneurisation, orchestration des conteneurs, automatisation des déploiements, etc.).





- 1 Demande la liste des produits
- 2 Demande les informations sur un produit
- 3 Passe une commande
- 4 Récupère la commande créée
- 5 Demande le paiement de la commande
- 6 Demande que la commande soit marquée comme payée
- 7 Confirme le paiement

User interfaces

La page d'accueil de l'application web FurniShop présente initialement un agencement comprenant le logo, le menu, le slogan et une image:

The screenshot shows a web browser window with the address bar displaying "localhost:3000". The header features the "FurniShop" logo on the left and a navigation menu with "HOME", "ABOUT", and "CONTACT" links on the right. Below the header, a large slogan reads "Elevate Your Space with FurniShop". A paragraph of text follows, stating: "At FurniShop, we believe that your home should be a reflection of your personality and lifestyle, which is why we offer furniture pieces that are not only functional but also aesthetically pleasing." A "Shop now" button is located below the text. To the right, there is a grid of six smaller images showcasing modern furniture pieces, such as a sofa, a chair, and a side table. A large orange callout box with the word "Modern" in white text is positioned at the bottom right of the grid.

← → ⌛ ⓘ localhost:3000

FurniShop

HOME ABOUT CONTACT

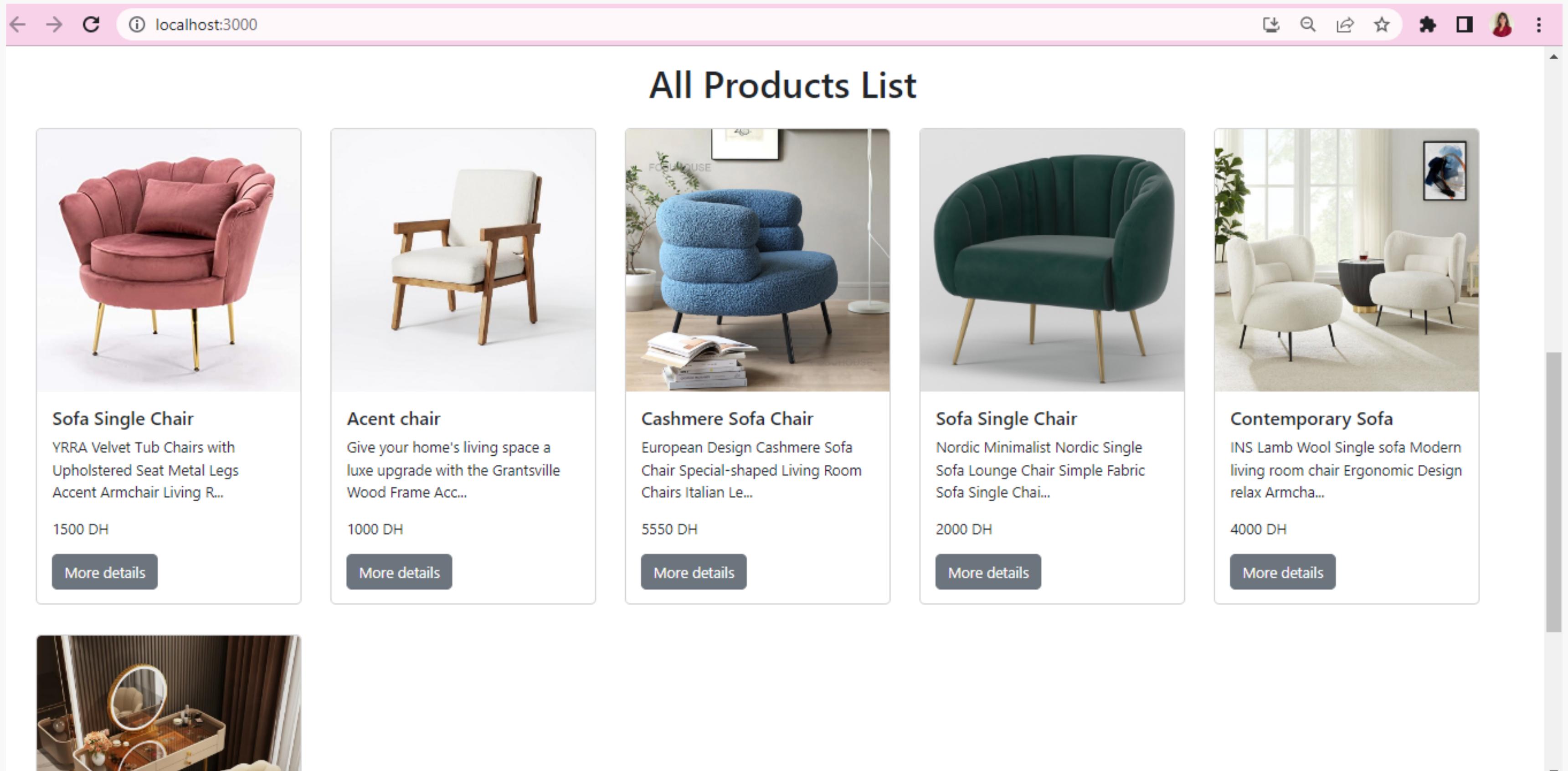
Elevate Your Space with FurniShop

At FurniShop, we believe that your home should be a reflection of your personality and lifestyle, which is why we offer furniture pieces that are not only functional but also aesthetically pleasing.

Shop now

Modern

Lorsqu'on fait défiler la page vers le bas, on découvre les différentes fournitures disponibles. Chaque fourniture est accompagnée de son nom, d'une description, du prix et d'un bouton "More details" :

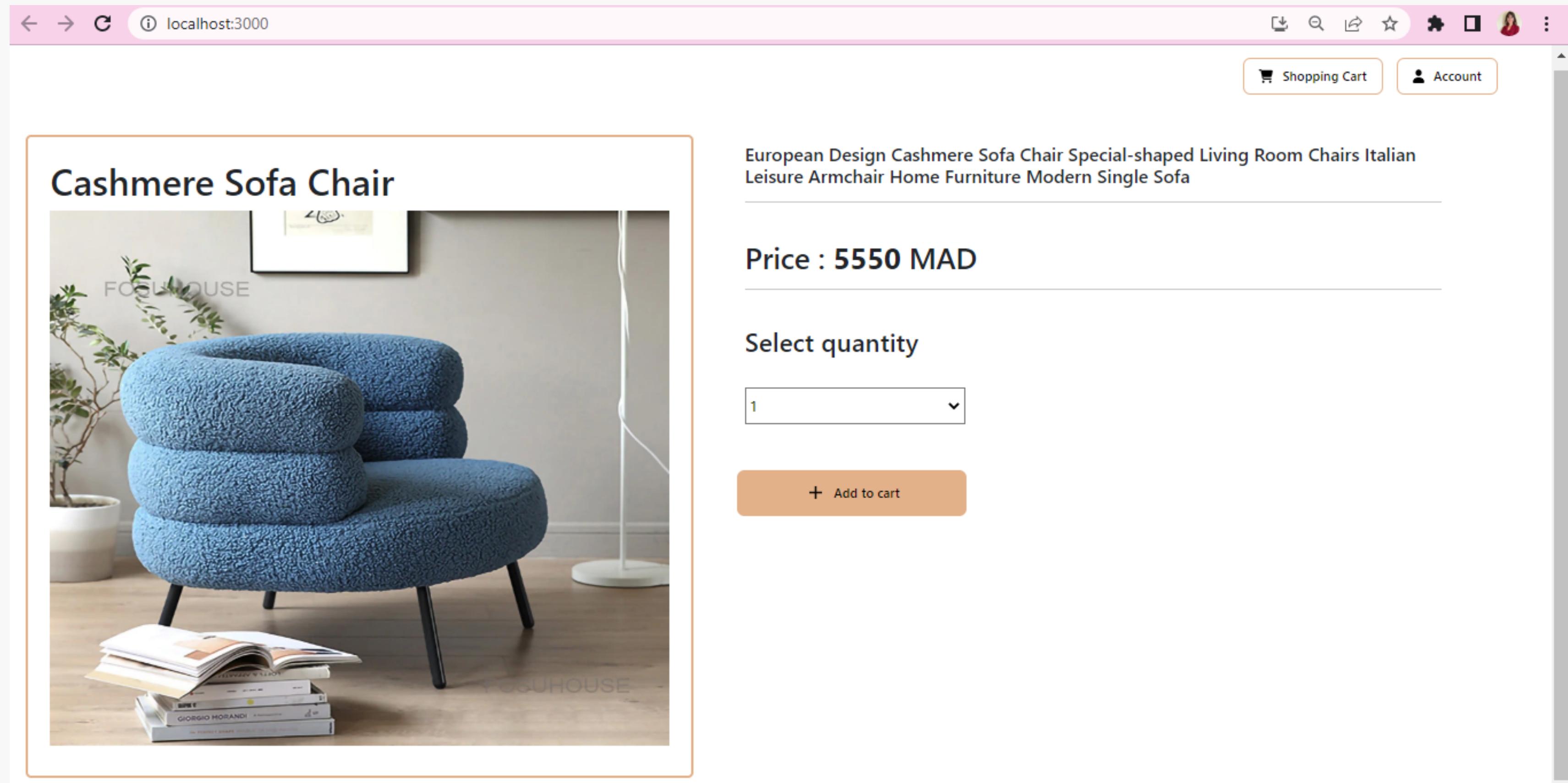


The screenshot shows a web browser window with a pink header bar. The address bar displays "localhost:3000". The main content area is titled "All Products List" in bold blue text. Below the title, there are five product cards arranged horizontally. Each card contains an image of a chair, the product name, a brief description, the price, and a "More details" button.

Product Image	Product Name	Description	Price	Action
	Sofa Single Chair	YRRA Velvet Tub Chairs with Upholstered Seat Metal Legs Accent Armchair Living R...	1500 DH	More details
	Accent chair	Give your home's living space a luxe upgrade with the Grantsville Wood Frame Acc...	1000 DH	More details
	Cashmere Sofa Chair	European Design Cashmere Sofa Chair Special-shaped Living Room Chairs Italian Le...	5550 DH	More details
	Sofa Single Chair	Nordic Minimalist Nordic Single Sofa Lounge Chair Simple Fabric Sofa Single Chai...	2000 DH	More details
	Contemporary Sofa	INS Lamb Wool Single sofa Modern living room chair Ergonomic Design relax Armcha...	4000 DH	More details

A small portion of another product card is visible at the bottom left, showing a vanity setup with a round mirror and a wooden cabinet.

En cliquant sur le bouton "En savoir plus", on accède à des informations plus détaillées sur le produit sélectionné:



The screenshot shows a web browser window with a pink header bar. The address bar displays "localhost:3000". The main content area shows a product page for a "Cashmere Sofa Chair". On the left, there is a large image of a blue, textured sofa chair with black legs, positioned in a room with a potted plant and some books on the floor. To the right of the image, the product title "Cashmere Sofa Chair" is displayed in bold. Below the title is a detailed description: "European Design Cashmere Sofa Chair Special-shaped Living Room Chairs Italian Leisure Armchair Home Furniture Modern Single Sofa". The price is listed as "Price : 5550 MAD". A dropdown menu for selecting quantity is set to "1". At the bottom is a large orange "Add to cart" button with a plus sign.

Cashmere Sofa Chair

European Design Cashmere Sofa Chair Special-shaped Living Room Chairs Italian Leisure Armchair Home Furniture Modern Single Sofa

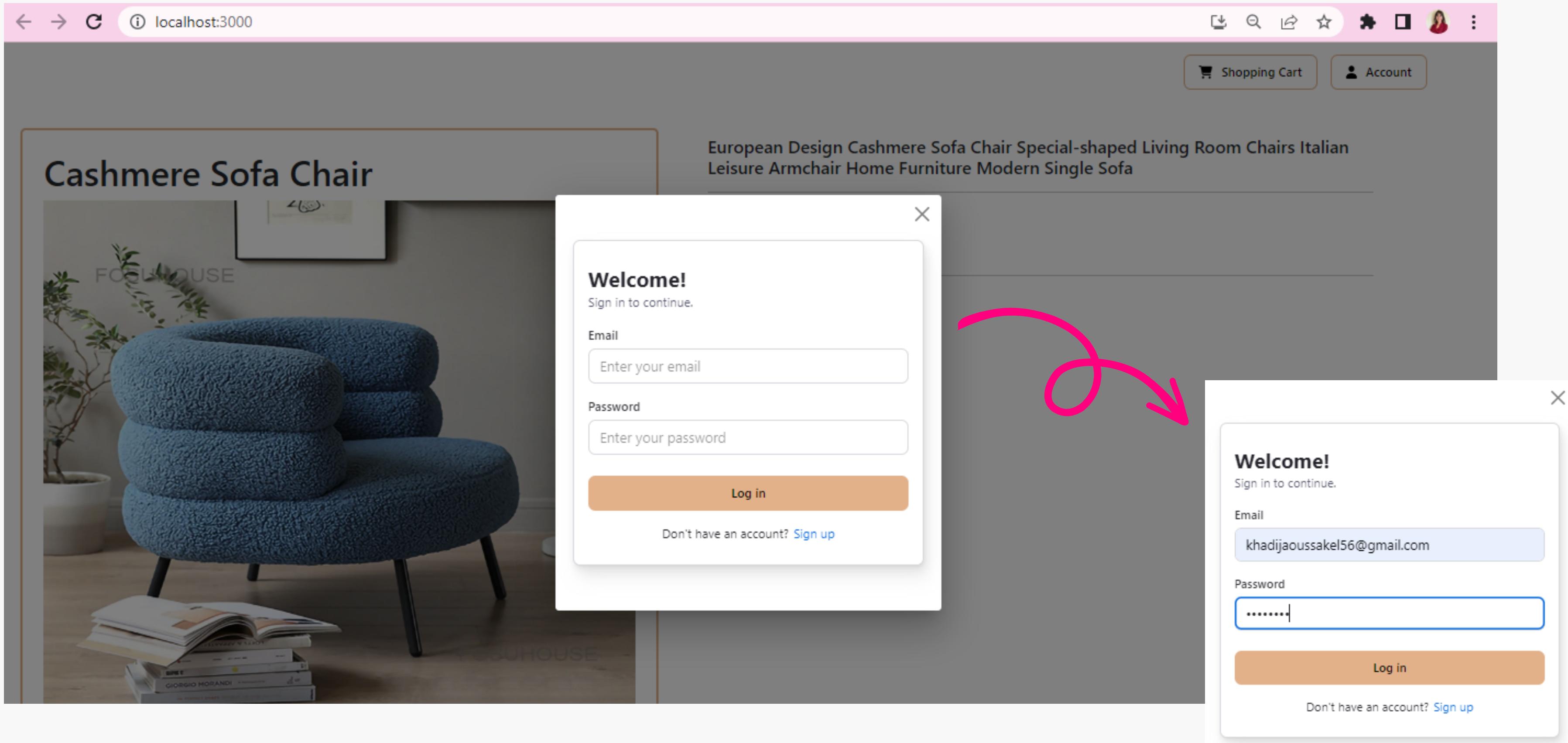
Price : 5550 MAD

Select quantity

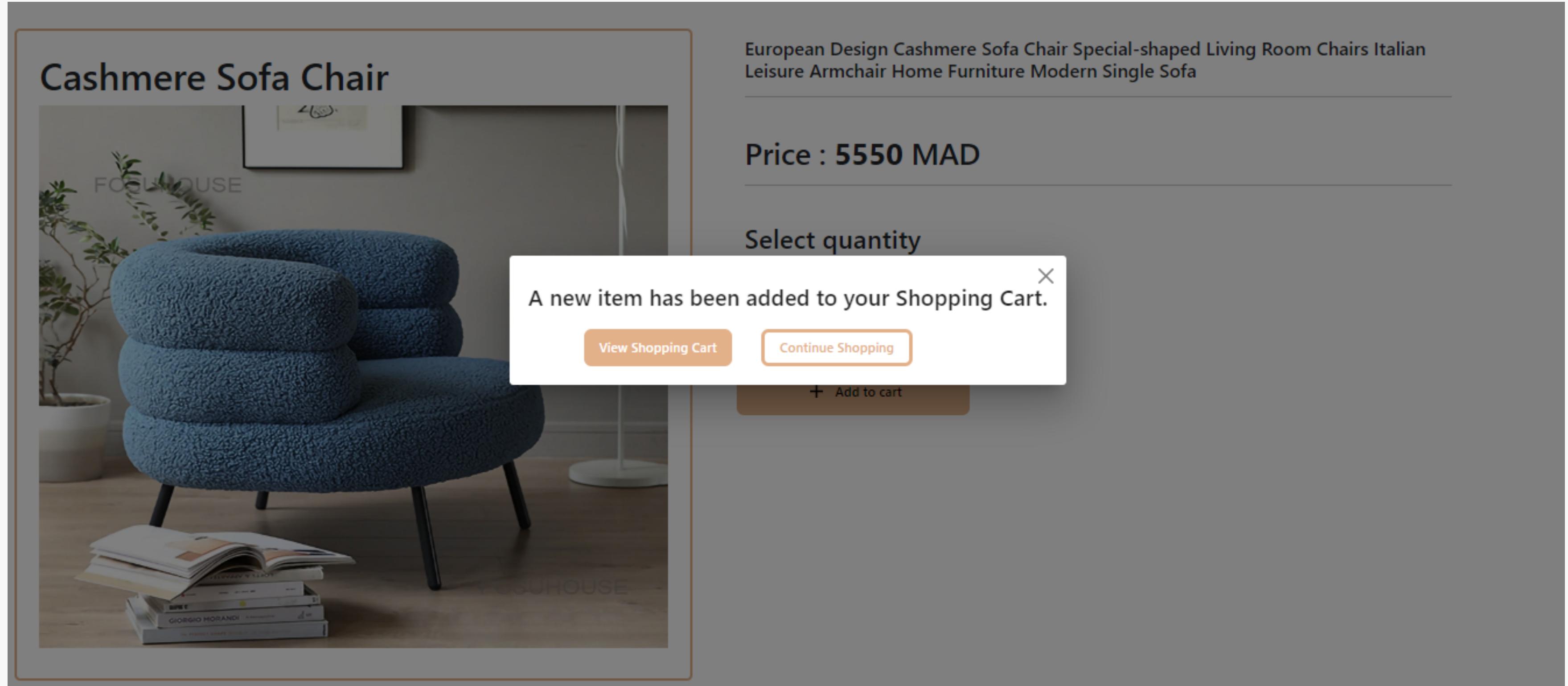
1

+ Add to cart

Lorsque vous vous trouvez sur la page du produit, en cliquant sur le bouton "Add to cart", un formulaire apparaît vous demandant de renseigner votre adresse e-mail et votre mot de passe:



Une fois que vous vous êtes connecté avec succès, le produit sera ajouté à votre panier et une fenêtre contextuelle s'affichera indiquant : "A new item has been added to your shopping cart" :



En cliquant sur "View shopping cart", vous pourrez consulter la liste des produits que vous avez ajoutés, avec la quantité souhaitée et le prix total de votre commande:

The screenshot shows a web browser window with a pink header bar containing navigation icons and the URL "localhost:3000". The main content area displays a product page for a "Cashmere Sofa Chair". A modal window titled "Shopping cart" is overlaid on the page, listing three items:

Product	Quantity	Price
Acent chair	1	1000 MAD
Acent chair	3	3000 MAD
Cashmere Sofa Chair	2	11100 MAD

The total amount is displayed as "Total Amount: 15100 MAD". A "Proceed to checkout" button is at the bottom of the cart modal. The background of the page shows a large blue cashmere sofa chair and some decorative plants.

Pour retirer un produit de votre liste d'achats, il vous suffit de cliquer sur la croix (X) située à droite du produit:

The screenshot shows a web browser window with a pink header bar containing navigation icons and the URL "localhost:3000". The main content area displays a product page for a "Cashmere Sofa Chair". The page features a large image of a blue, textured sofa chair, a smaller image of an accent chair, and descriptive text: "European Design Cashmere Sofa Chair Special-shaped Living Room Chairs Italian Leisure Armchair Home Furniture Modern Single Sofa". A modal dialog box titled "Shopping cart" is overlaid on the page, listing items from the user's cart:

Item	Quantity	Price
Acent chair	3	3000 MAD
Cashmere Sofa Chair	2	11100 MAD

The total amount is displayed as "Total Amount: 14100 MAD". At the bottom of the modal is a "Proceed to checkout" button.

En cliquant sur le bouton "Proceed to checkout", vous confirmez votre commande:

The screenshot shows a web browser window with the URL `localhost:3000` in the address bar. The page content is a product listing for a "Cashmere Sofa Chair". A modal dialog box is displayed, containing a success message from the server: "localhost:3000 indique your payment was successful". Below this message is a blue "OK" button. In the background, a larger modal titled "Shopping cart" lists the items in the user's cart. The cart contains two items: an "Acent chair" (Quantity: 3) and a "Cashmere Sofa Chair" (Quantity: 2). The total amount is displayed as 14100 MAD. At the bottom of the cart modal is a prominent orange "Proceed to checkout" button.

localhost:3000 indique
your payment was successful

OK

Shopping cart

Acent chair
Quantity: 3
Price: 3000 MAD

Cashmere Sofa Chair
Quantity: 2
Price: 11100 MAD

Total Amount: 14100 MAD

Proceed to checkout

Et votre panier d'achat sera vidé:

The screenshot shows a web browser window with a pink header bar. The address bar displays "localhost:3000". The main content area features a large image of a blue Cashmere Sofa Chair. To the right of the image, the product title is "European Design Cashmere Sofa Chair Special-shaped Living Room Chairs Italian Leisure Armchair Home Furniture Modern Single Sofa" and the price is "Price : 5550 MAD". A modal dialog box titled "Shopping cart" is overlaid on the page. The dialog contains the text "Total Amount: 0 MAD" and a "Proceed to checkout" button. The browser's toolbar at the top includes icons for back, forward, search, and user profile.

← → ⌛ ⓘ localhost:3000

Shopping Cart Account

Cashmere Sofa Chair

European Design Cashmere Sofa Chair Special-shaped Living Room Chairs Italian Leisure Armchair Home Furniture Modern Single Sofa

Price : 5550 MAD

X

Shopping cart

Total Amount: 0 MAD

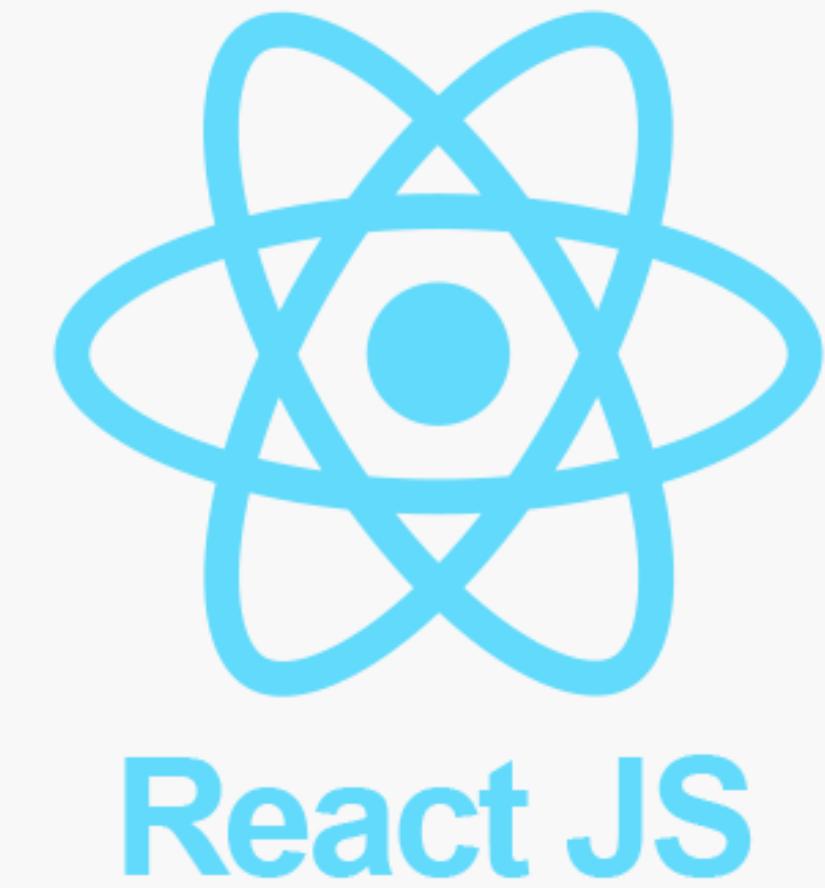
Proceed to checkout

Microservice frontend

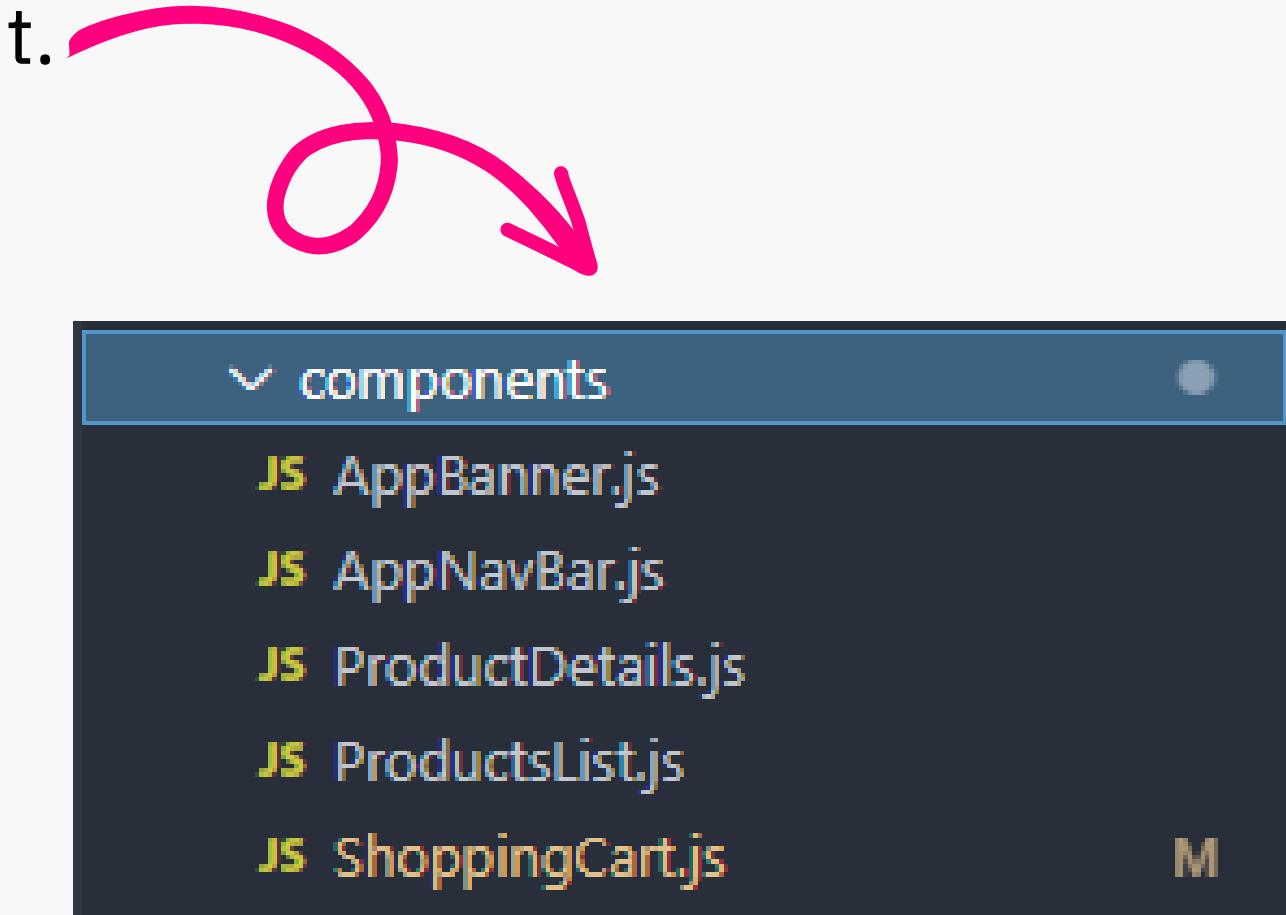
SPA avec React, Dockerfile, Gitlab CI/CD, Kubernetes

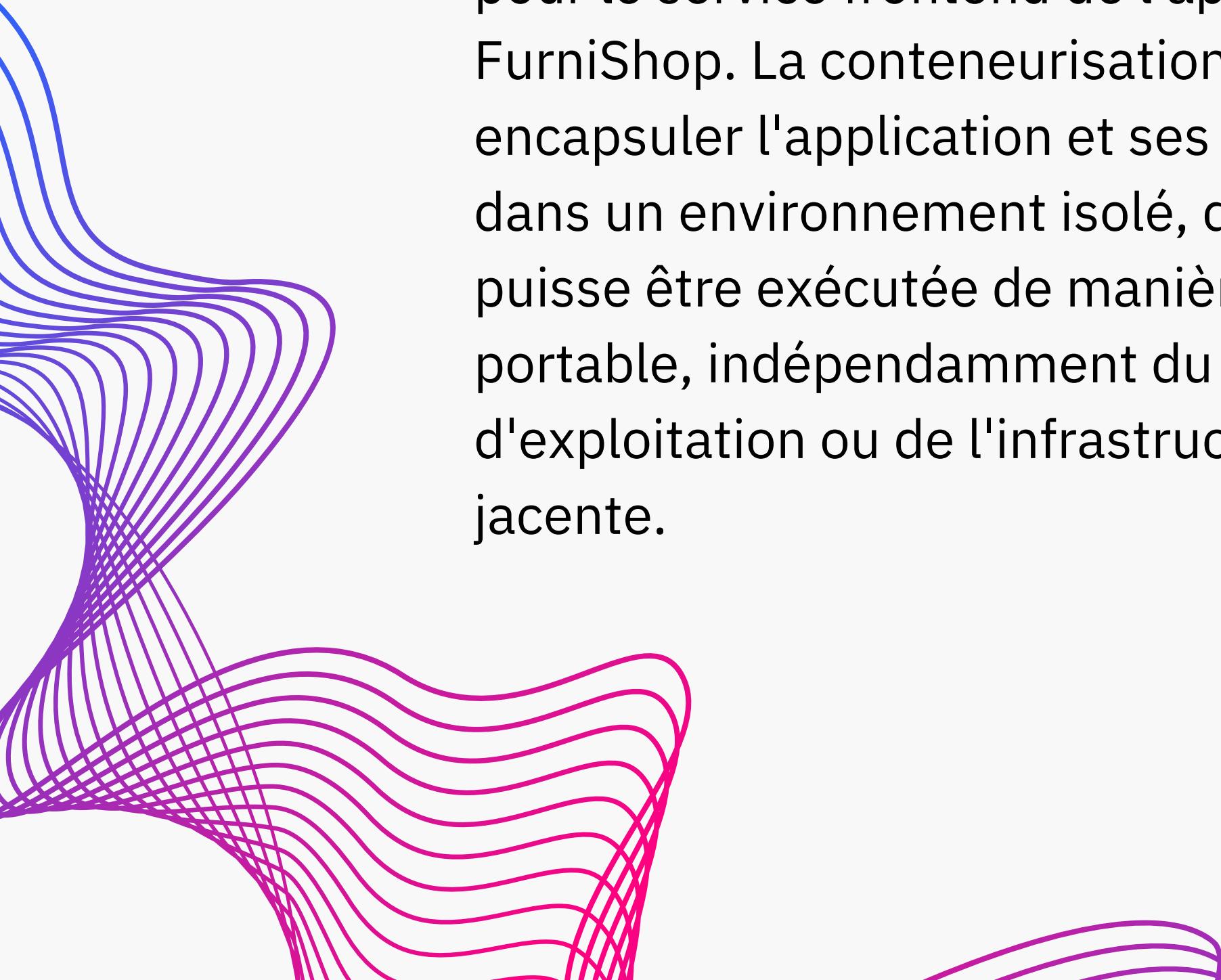


Le service frontend de l'application est développé en utilisant la bibliothèque React, qui permet de créer une application web en tant que SPA (Single Page Application). Une SPA est une application web qui fonctionne en chargeant dynamiquement son contenu, ce qui signifie que seule une seule page HTML est chargée initialement, puis les mises à jour de contenu sont gérées de manière fluide sans rechargement de la page.

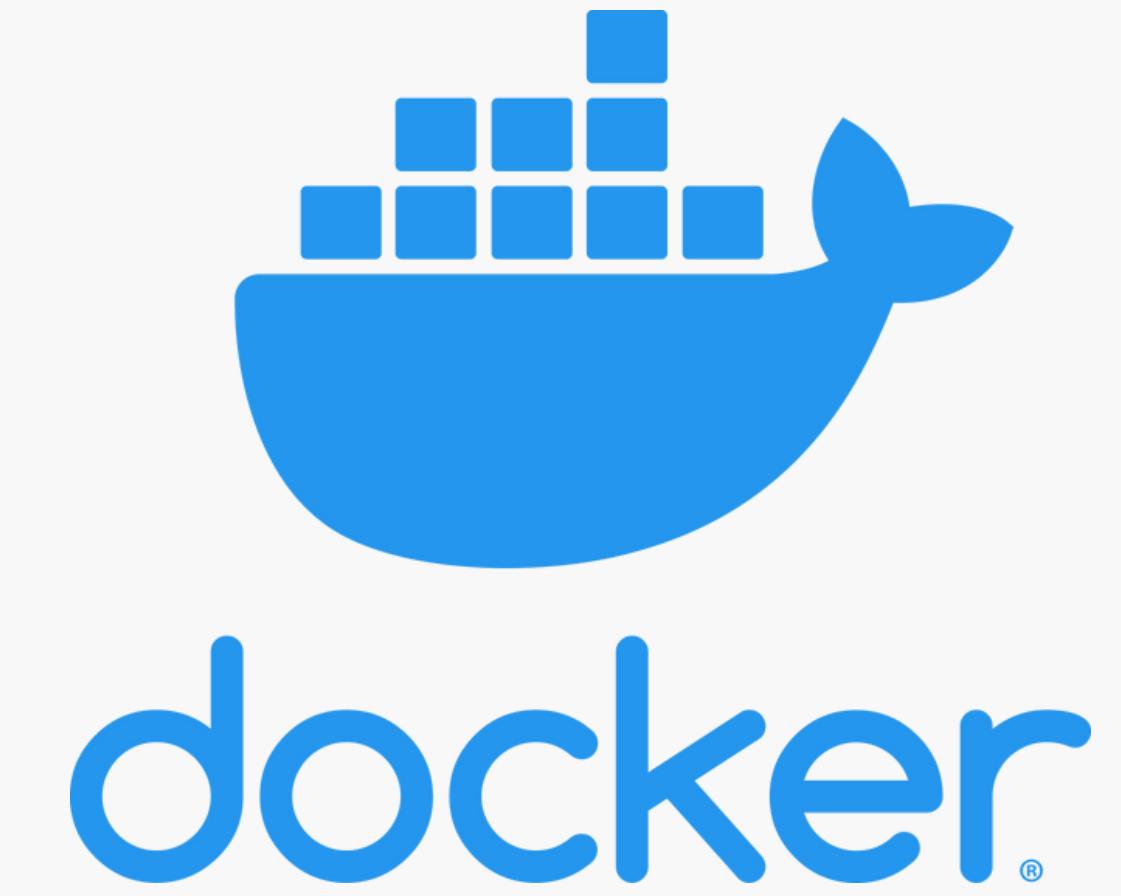


Dans le dossier src , on crée un nouveau répertoire nommé components où on va mettre nos composants React.Les composants dans React permettent une approche modulaire et réutilisable du développement d'interface utilisateur. Ils permettent de diviser une application en unités plus petites et gérables, facilitant ainsi la maintenance, les tests et les mises à jour. Les composants peuvent être réutilisés à plusieurs endroits dans une application, ce qui favorise la cohérence et l'efficacité du développement.





On a utilisé Docker pour créer un conteneur pour le service frontend de l'application FurniShop. La conteneurisation consiste à encapsuler l'application et ses dépendances dans un environnement isolé, de sorte qu'elle puisse être exécutée de manière cohérente et portable, indépendamment du système d'exploitation ou de l'infrastructure sous-jacente.



On a créé le fichier Dockerfile, qui définit les étapes pour construire l'image Docker du service frontend, tout en respectant les bonnes pratiques.

Cette ligne spécifie l'image de base à utiliser pour le conteneur Docker. Dans ce cas, on a choisi l'image node:16-alpine, qui est une version légère de l'image Node.js

En utilisant l'utilisateur node à l'intérieur du conteneur, on applique le principe du moindre privilège en limitant les permissions et en réduisant les risques de sécurité.

On utilise npm ci au lieu de npm install pour garantir une installation plus rapide et reproductible des dépendances. L'option --only=production permet d'installer uniquement les dépendances nécessaires pour l'environnement de production, ce qui réduit la taille de l'image.

```
client > ⚙ dockerfile > ...
1 FROM node:16-alpine
2 USER node
3 ENV NODE_ENV=production
4 WORKDIR /client
5 COPY --chown=node:node package.json package-lock.json .
6 RUN npm ci --only=production
7 COPY --chown=node:node .
8 EXPOSE 3000
9 CMD ["npm", "run", "start"]
10
```

la commande npm run start est utilisée pour démarrer l'application frontend.

L'utilisation de --chown=node:node garantit que les fichiers appartiennent à l'utilisateur node à l'intérieur du conteneur.

On a créé également le fichier `.dockerignore` qui est utilisé pour spécifier les fichiers et répertoires à exclure lors de la construction de l'image Docker à partir du contexte de construction.

On a inclus le répertoire `node_modules` dans le fichier `.dockerignore` pour optimiser la taille de l'image Docker. Le répertoire `node_modules` contient toutes les dépendances installées pour notre application frontend, et il peut être assez volumineux.

Et puisque les dépendances sont déjà installées à l'intérieur du conteneur à l'aide de la commande `npm ci --only=production`, il n'est pas nécessaire de copier le répertoire `node_modules` depuis le contexte de construction dans l'image Docker.

```
client > ➜ .dockerignore
  1 node_modules
```

On construit l'image Docker à partir du Dockerfile présent dans le répertoire courant et on lui attribue le nom "kouss/client" avec la balise "1.0.0"

```
PS C:\Users\HP\Desktop\FurniShop\client> docker build -t kouss/client:1.0.0 .
● [+] Building 129.7s (11/11) FINISHED
  => [internal] load build definition from Dockerfile
  => => transferring dockerfile: 271B
  => [internal] load .dockerignore
  => => transferring context: 34B
  => => transferring context: 6.55kB
  => CACHED [1/5] FROM docker.io/library/node:16-alpine@sha256:8d18e32fa398763c07407e9d407c64e6a3c2a18e9fa97362cfde669a6b6161ef
  => => resolve docker.io/library/node:16-alpine@sha256:8d18e32fa398763c07407e9d407c64e6a3c2a18e9fa97362cfde669a6b6161ef
  => [2/5] WORKDIR /client
  => [3/5] COPY --chown=node:node package.json package-lock.json .
  => [4/5] RUN npm ci --only=production
  => [5/5] COPY --chown=node:node . .
  => exporting to image
  => => exporting layers
  => => writing image sha256:ef0a167e35f814cf8544e418c174b613884f1d2a23b3f67828c8f806f0b79bce
  => => naming to docker.io/kouss/client:1.0.0
```

On exécute un conteneur à partir de l'image "kouss/client:1.0.0" et effectue un mappage de port pour permettre l'accès au service du conteneur sur le port 3000 de l'hôte.

```
PS C:\Users\HP\Desktop\FurniShop\paiement> docker run -p 3000:3000 kouss/client:1.0.0

> client@0.1.0 start
> react-scripts start

(node:25) [DEP_WEBPACK_DEV_SERVER_ON_AFTER_SETUP_MIDDLEWARE] DeprecationWarning: 'onAfterSetupMiddleware' option is deprecated. Please use the
  'setupMiddlewares' option.
(Use `node --trace-deprecation ...` to show where the warning was created)
(node:25) [DEP_WEBPACK_DEV_SERVER_ON_BEFORE_SETUP_MIDDLEWARE] DeprecationWarning: 'onBeforeSetupMiddleware' option is deprecated. Please use t
he 'setupMiddlewares' option.
Starting the development server...

Compiled successfully!

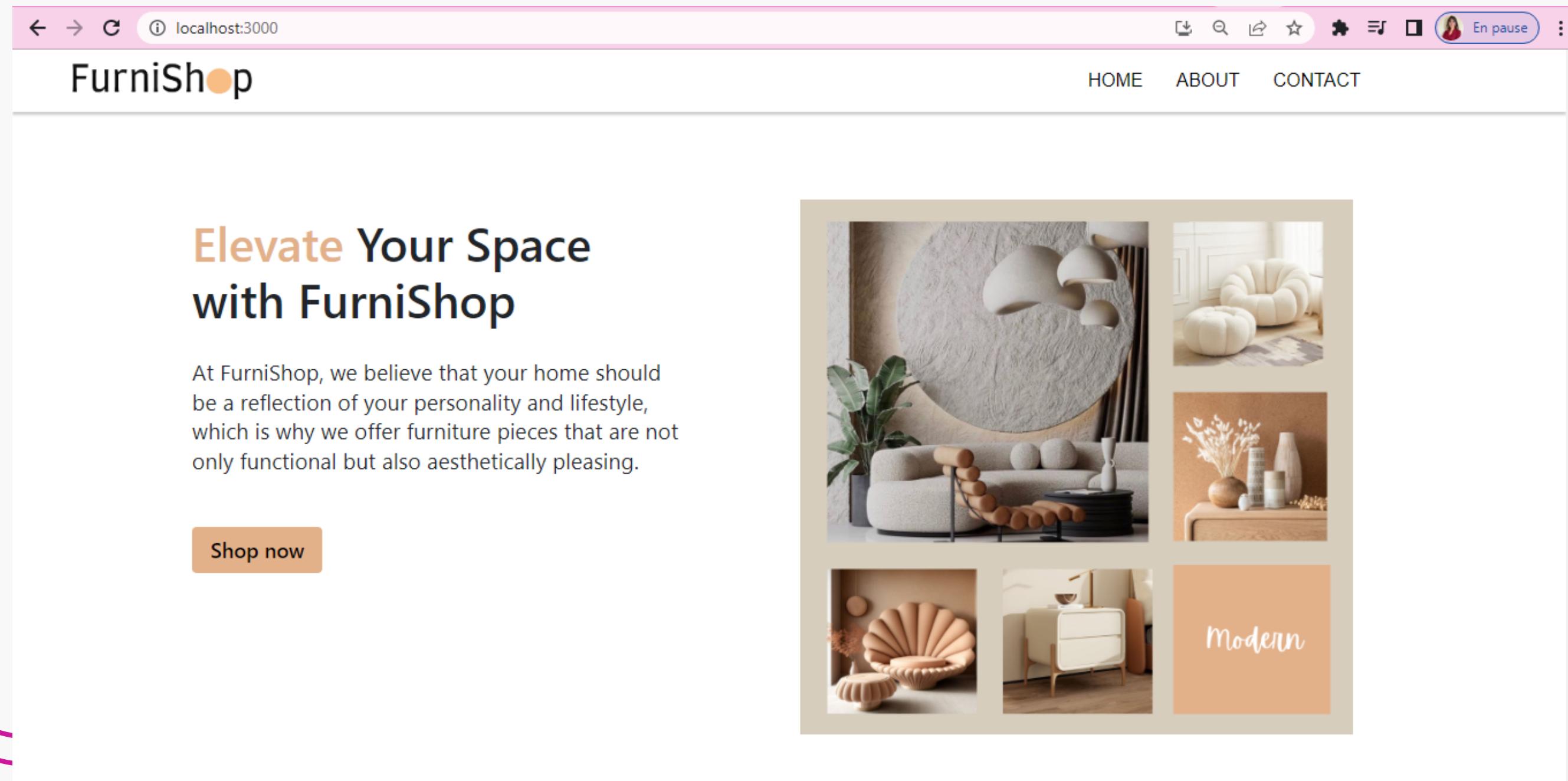
You can now view client in the browser.

  Local:          http://localhost:3000
  On Your Network: http://172.17.0.6:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
Compiling...
Compiled successfully!
webpack compiled successfully
```

Cela permet d'accéder à l'application FurniShop en utilisant l'URL <http://localhost:3000>.



The screenshot shows a web browser window with the URL "localhost:3000" in the address bar. The page title is "FurniShop". The main heading reads "Elevate Your Space with FurniShop". Below it, a paragraph states: "At FurniShop, we believe that your home should be a reflection of your personality and lifestyle, which is why we offer furniture pieces that are not only functional but also aesthetically pleasing." A "Shop now" button is visible. To the right, there is a grid of images showing various furniture pieces like a sofa, a chair, and a side table, along with a decorative text box labeled "modern". The browser interface includes standard navigation buttons and a user profile icon.

← → ⌛ i localhost:3000

FurniShop

Elevate Your Space with FurniShop

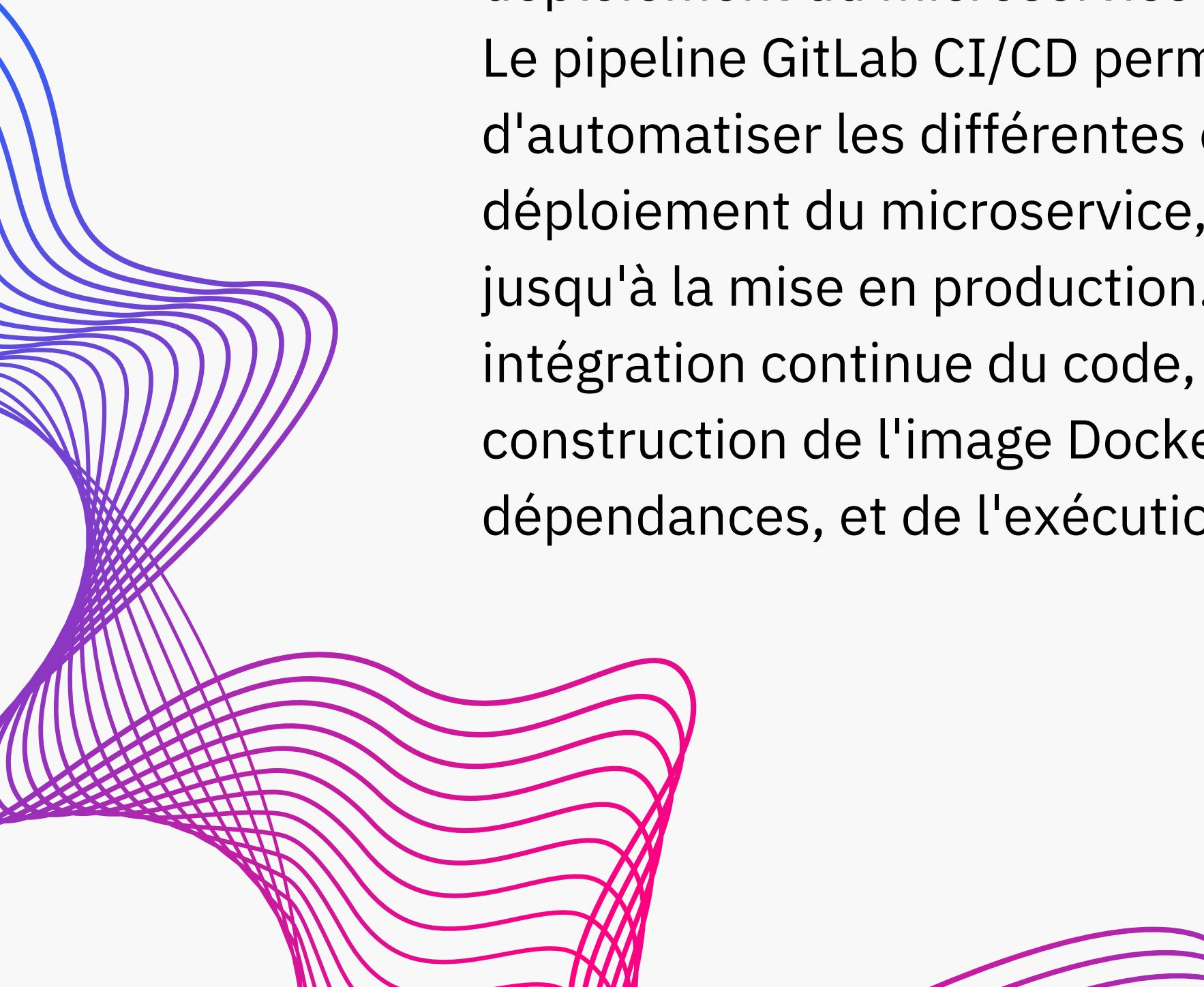
At FurniShop, we believe that your home should be a reflection of your personality and lifestyle, which is why we offer furniture pieces that are not only functional but also aesthetically pleasing.

Shop now

modern

HOME ABOUT CONTACT

En pause



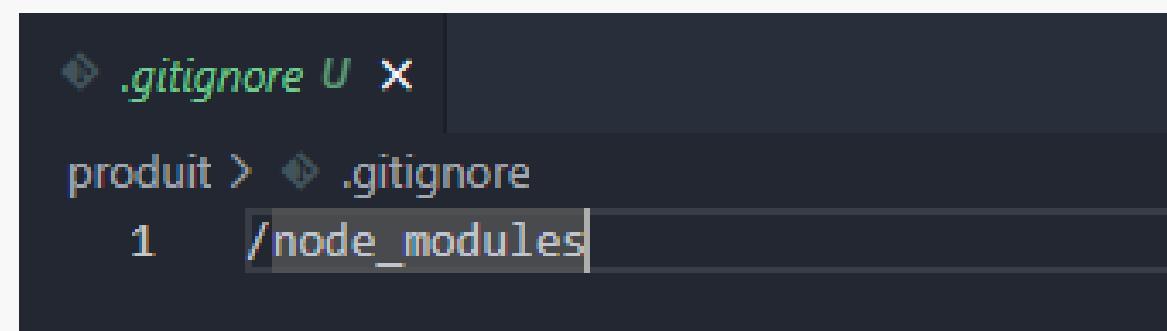
On utilise GitLab CI/CD pour automatiser le déploiement du microservice à l'aide d'un pipeline. Le pipeline GitLab CI/CD permet d'orchestrer et d'automatiser les différentes étapes nécessaires au déploiement du microservice, depuis la construction jusqu'à la mise en production. Il assure une intégration continue du code, des tests, de la construction de l'image Docker, de la gestion des dépendances, et de l'exécution des déploiements.



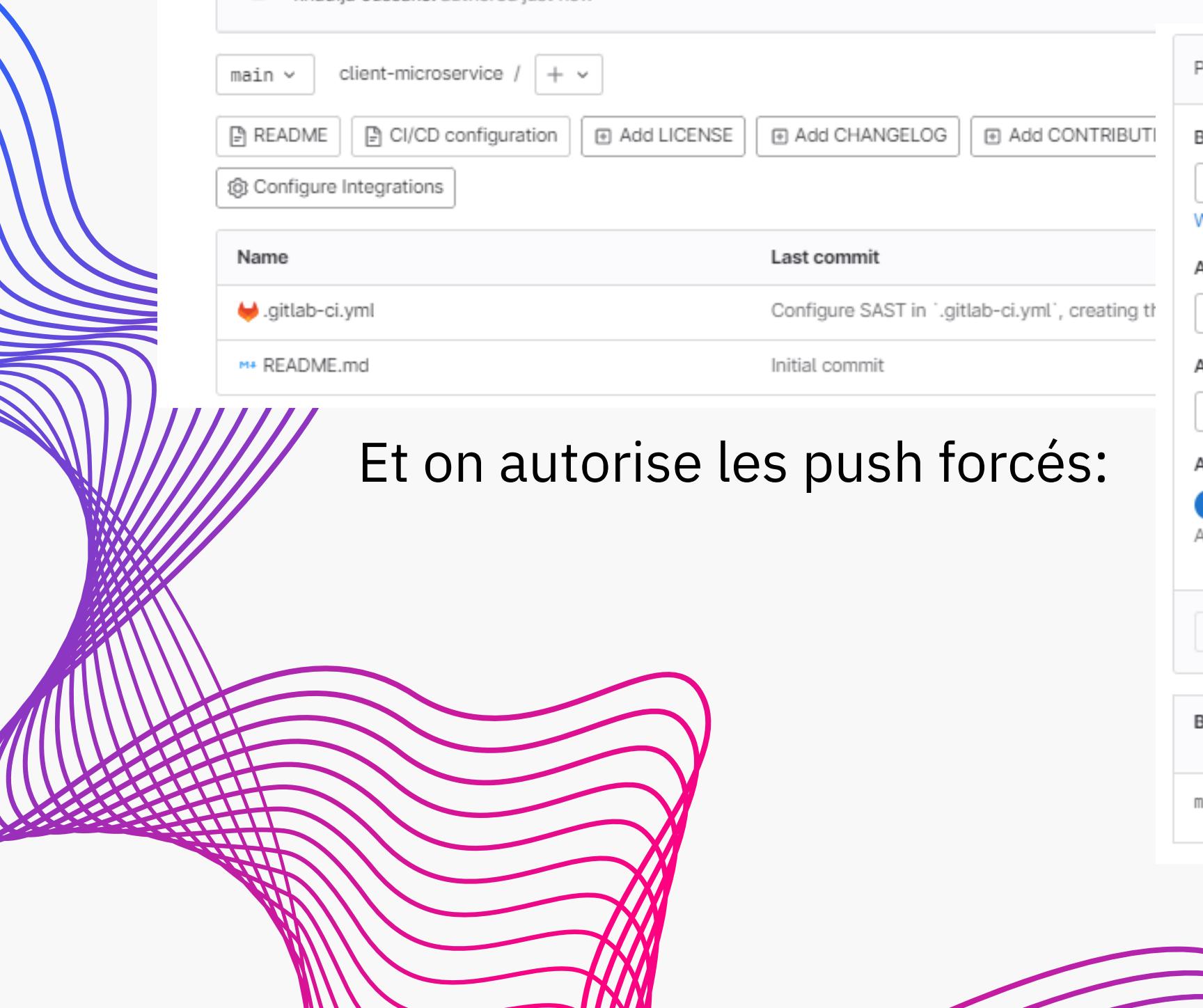
Tout d'abord on initialise un nouveau dépôt Git dans le répertoire client. Puis on ajoute tous les fichiers et répertoires modifiés dans le répertoire de travail actuel au suivi de version Git. Enfin, on effectue un commit pour créer un instantané des modifications ajoutées:

```
PS C:\Users\HP\Desktop\FurniShop\client> git init
Initialized empty Git repository in C:/Users/HP/Desktop/FurniShop/client/.git/
PS C:\Users\HP\Desktop\FurniShop\client> git add .
warning: LF will be replaced by CRLF in .gitignore.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in README.md.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in package-lock.json.
```

```
PS C:\Users\HP\Desktop\FurniShop\client> git commit -m "client microservice"
[master (root-commit) c20a795] client microservice
 25 files changed, 19021 insertions(+)
 create mode 100644 .dockerignore
 create mode 100644 .gitignore
 create mode 100644 README.md
 create mode 100644 dockerfile
 create mode 100644 package-lock.json
 create mode 100644 package.json
 create mode 100644 public/favicon.ico
 create mode 100644 public/index.html
 create mode 100644 public/logo192.png
```



On crée un nouveau référentiel GitLab avec le nom "client-microservice":



The screenshot shows the GitLab interface for a project named "client-microservice". The top navigation bar includes a profile icon, the project name, a "Project ID: 46387474" link, and social sharing icons for Star (0) and Fork (0). Below the header, a message from "khadija oussakel" indicates a commit to ".gitlab-ci.yml". The main content area displays the repository structure under the "main" branch, showing files like ".gitlab-ci.yml" and "README.md". On the right, a "Protect a branch" dialog is open, specifically for the "main" branch. It contains settings for merging, pushing, and force pushing. A large blue "Protect" button is visible at the bottom of this dialog. Below the dialog, a table summarizes the current protection status for the "main" branch.

Branch	Allowed to merge	Allowed to push and merge	Allowed to force push
main (default)	Maintainers	Maintainers	<input checked="" type="checkbox"/>

Et on autorise les push forcés:

On ajoute une nouvelle connexion distante à notre dépôt Git en utilisant la commande 'git remote add origin' et on pousse les modifications vers la branche distante. L'option -f est utilisée pour forcer le push des modifications vers la branche distante.

```
PS C:\Users\HP\Desktop\FurniShop\client> git remote add origin https://gitlab.com/khadijaoussakel/client-microservice.git
PS C:\Users\HP\Desktop\FurniShop\client> git branch -M main
PS C:\Users\HP\Desktop\FurniShop\client> git push -uf origin main
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
To https://gitlab.com/khadijaoussakel/client-microservice.git
 + 290abb6...c20a795 main -> main (forced update)
branch 'main' set up to track 'origin/main'.
```

The screenshot shows a GitLab repository named 'client microservice' created by 'khadija oussakel' 21 minutes ago. The repository has a single commit with the SHA 'c20a795'. The repository structure includes a 'main' branch, a 'client-microservice' folder, and several files: README, CI/CD configuration, Add LICENSE, Add CHANGELOG, Add CONTRIBUTING, Add Kubernetes cluster, and Add Wiki. A 'Configure Integrations' button is also present. The file list shows the following entries:

Name	Last commit	Last update
public	client microservice	21 minutes ago
src	client microservice	21 minutes ago
.dockerignore	client microservice	21 minutes ago
.gitignore	client microservice	21 minutes ago
README.md	client microservice	21 minutes ago
dockerfile	client microservice	21 minutes ago
package-lock.json	client microservice	21 minutes ago
package.json	client microservice	21 minutes ago

On crée une variable d'environnement qui comportera le mot de passe de Docker Hub. Cela offre une couche de sécurité supplémentaire en évitant d'écrire le mot de passe directement dans les scripts ou les fichiers sources.

Type	↑ Key	Value	Options	Environments	
Variable	DOCKER_HUB_PASSWORD 	***** 	Protected	All (default) 	

Maintenant on crée le fichier gitlab-ci.yml avec quatre stages :

On inclut le fichier **Security/SAST.gitlab-ci.yml** pour gérer l'analyse statique du code source.

La section **cache** spécifie les chemins des fichiers ou des répertoires à mettre en cache entre les exécutions du pipeline. Dans ce cas, le répertoire `node_modules/` est mis en cache pour accélérer les builds futurs.

L'étape sast est définie dans la section `securityScan`. Cette étape est **responsable de l'analyse statique du code source**.

Les artefacts produits par cette étape sont un rapport de sécurité au format JSON nommé `gl-sast-report.json`. En plus si l'analyse statique de sécurité ne réussit pas, le pipeline entier sera considéré comme échoué

Le cache spécifique à la branche est utilisé pour stocker les dépendances installées dans le répertoire `node_modules/`. Le script exécute la commande `npm ci` pour installer les dépendances du projet.

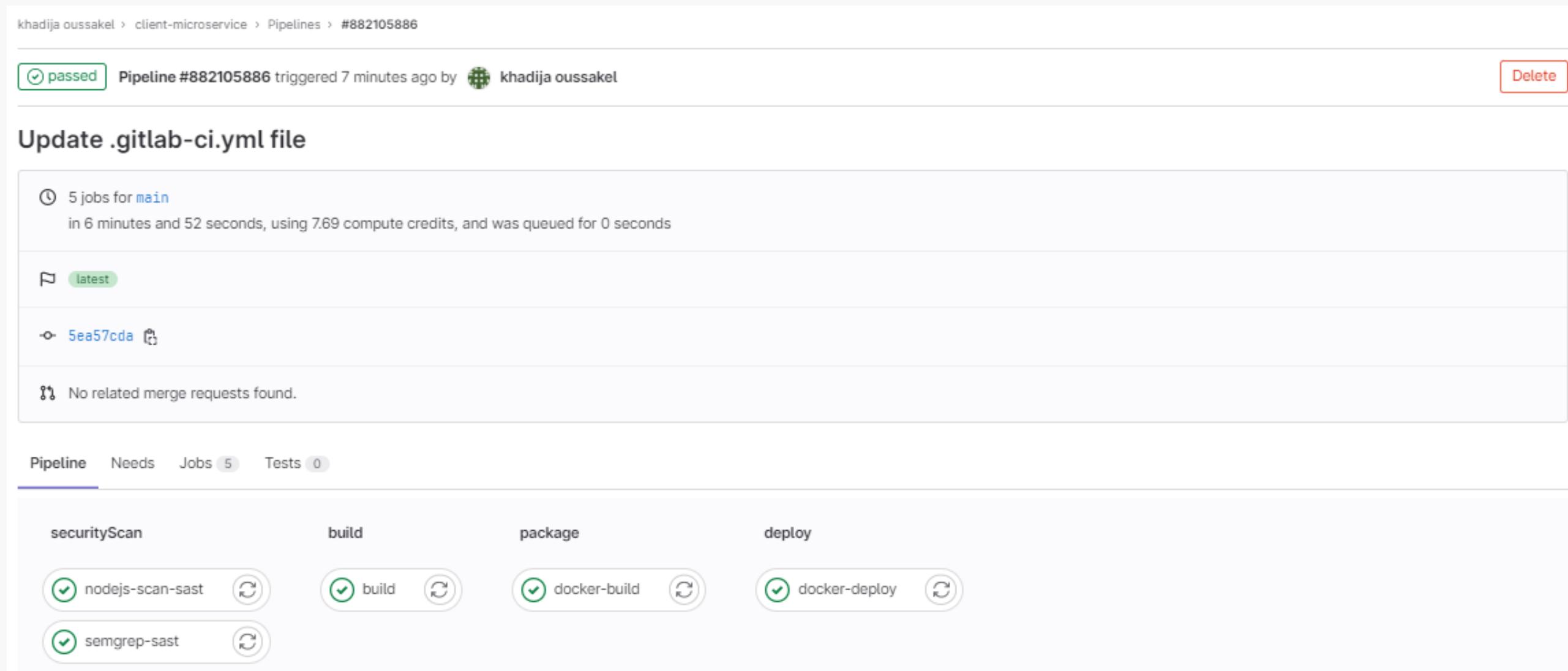
```
image: docker:latest
services:
  - docker:dind
stages:
  - securityScan
  - build
  - package
  - deploy
include:
  - template: Security/SAST.gitlab-ci.yml
cache:
  paths:
    - node_modules/
sast:
  stage : securityScan
  artifacts:
    reports:
      sast: gl-sast-report.json
    allow_failure: false
build:
  image: node:16
  stage: build
  cache:
    key: "$CI_COMMIT_REF_SLUG" # Utilisez une clé de cache spécifique à la branche
    paths:
      - node_modules/
  script:
    - npm ci                                # Utiliser le cache pour télécharger les dépendances
    - npm run build
```

L'étape docker-build est définie dans la section package. Avant de construire l'image Docker, le script exécute quelques commandes de préparation, notamment la connexion à Docker Hub, l'installation de curl et jq, et l'installation de trivy (un outil de sécurité pour les conteneurs). Ensuite, le script utilise Docker pour tirer l'image de base node:16, construire une nouvelle image nommée kouss/client:1.0.0 **en utilisant le cache de l'image de base, exécute une analyse de sécurité sur l'image avec trivy**, puis pousse l'image construite vers Docker Hub.

L'étape docker-deploy est définie dans la section deploy. Le script exécute une commande Docker pour déployer l'image kouss/client:1.0.0 dans un conteneur nommé client, en le reliant au port 3000 sur la machine hôte.


```
36 docker-build:
37   stage: package
38   before_script:
39     - docker login -u kouss -p $DOCKER_HUB_PASSWORD
40     - apk add --no-cache curl jq
41     - curl -sSfL https://raw.githubusercontent.com/aquasecurity/trivy/main/contrib/install.sh | sh -s -- -b /usr/local/bin
42   script:
43     - docker pull node:16 # Tirer l'image de base avant de construire pour utiliser le cache
44     - docker build --cache-from=node:16 -t kouss/client:1.0.0 . # Utiliser le cache de l'image
45     - trivy image kouss/client:1.0.0
46     - docker push kouss/client:1.0.0
47
48
49 docker-deploy:
50   stage : deploy
51   script :
52     - docker run -d -p 3000:3000 --name client kouss/client:1.0.0
53
```

Après la création du fichier `gitlab-ci.yml`, le pipeline se déclenche automatiquement. Dans notre cas, le pipeline a été exécuté avec succès:



The screenshot shows a successful CI pipeline run on the GitLab interface. The pipeline, titled "Update .gitlab-ci.yml file", was triggered 7 minutes ago by "khadija oussakel". It consists of 5 jobs for the "main" branch, completed in 6 minutes and 52 seconds, using 7.69 compute credits, and was queued for 0 seconds. The latest job, "5ea57cda", is shown with a green checkmark. There are no related merge requests found.

Pipeline Details:

- 5 jobs for main
- in 6 minutes and 52 seconds, using 7.69 compute credits, and was queued for 0 seconds
- latest: 5ea57cda
- No related merge requests found.

Job Statuses:

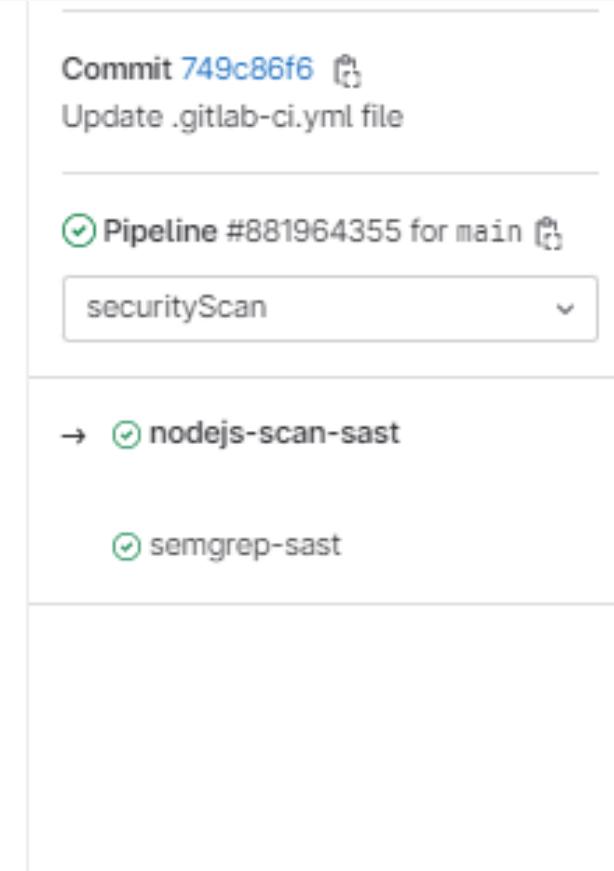
- securityScan: nodejs-scan-sast (green checkmark), semgrep-sast (green checkmark)
- build: build (green checkmark)
- package: docker-build (green checkmark)
- deploy: docker-deploy (green checkmark)

Navigation:

- Pipeline
- Needs
- Jobs 5
- Tests 0

Résultat de l'exécution du Job sast:

```
31 $ /analyzer run
32 [INFO] [NodeJsScan] [2023-05-29T12:34:29Z] ► GitLab NodeJsScan analyzer v4.0.2
33 [INFO] [NodeJsScan] [2023-05-29T12:34:29Z] ► Detecting project
34 [INFO] [NodeJsScan] [2023-05-29T12:34:29Z] ► Analyzer will attempt to analyze all projects in the repository
35 [INFO] [NodeJsScan] [2023-05-29T12:34:29Z] ► Running analyzer
36 [INFO] [NodeJsScan] [2023-05-29T12:34:43Z] ► Creating report
38 Saving cache for successful job
39 Creating cache default-protected...
40 WARNING: node_modules/: no matching files. Ensure that the artifact path is relative to the working directory (/builds/khadijaoussakel/client-microservice)
41 Archive is up to date!
42 Created cache
44 Uploading artifacts for successful job
45 Uploading artifacts...
46 gl-sast-report.json: found 1 matching artifact files and directories
47 Uploading artifacts as "sast" to coordinator... 201 Created id=4367525777 responseStatus=201 Created token=64_RcqS5
49 Cleaning up project directory and file based variables
51 Job succeeded
```

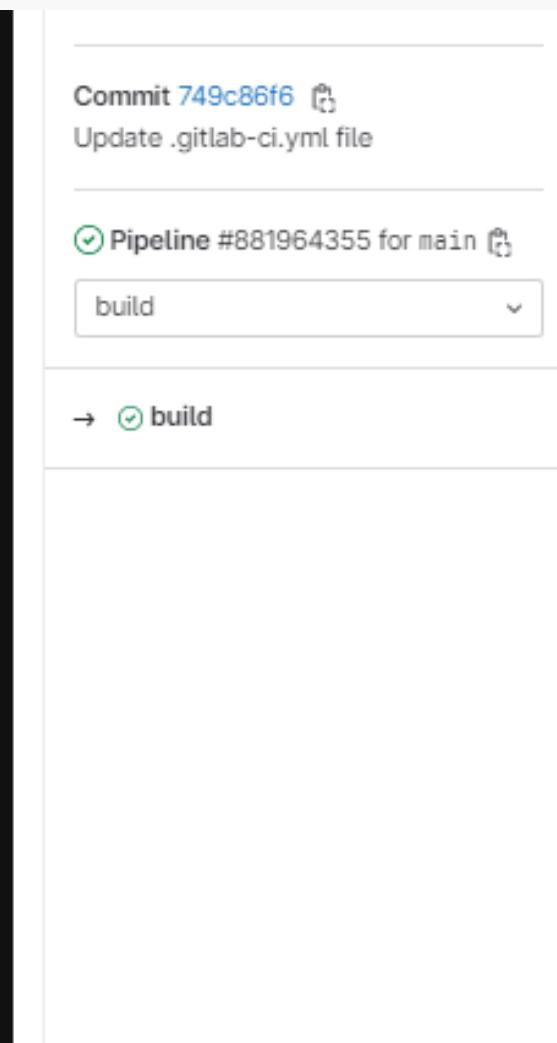


Le fichier gl-sast-report.json:

```
{} gl-sast-report (1).json ● ! commande.yaml JS App.js M, M dockerfile U ! paiem
C: > Users > HP > Downloads > {} gl-sast-report (1).json > ...
1 {"version": "15.0.4", "vulnerabilities": [],
2 "dependency_files": [],
3 "scan": {"analyzer": {"id": "nodejs-scan", "name": "NodeJsScan",
4 "url": "https://gitlab.com/gitlab-org/security-products/analyzers/nodejs-scan",
5 "vendor": {"name": "GitLab"}, "version": "4.0.2"}, "scanner": {"id": "njsscan",
6 "name": "njsscan", "url": "https://github.com/ajinabraham/njsscan",
7 "vendor": {"name": "GitLab"}, "version": "0.3.4"}, "type": "sast",
8 "start_time": "2023-06-05T23:20:17", "end_time": "2023-06-05T23:20:30",
9 "status": "success"}}
```

Résultat de l'exécution du Job build:

```
43 $ npm run build
44 > client@0.1.0 build
45 > react-scripts build
46 Creating an optimized production build...
47 Compiled successfully.
48 File sizes after gzip:
49   198 kB  build/static/js/main.cf94f418.js
50   28.17 kB build/static/css/main.a4204810.css
51   1.78 kB  build/static/js/787.c4e7f8f9.chunk.js
52 The project was built assuming it is hosted at /.
53 You can control this with the homepage field in your package.json.
54 The build folder is ready to be deployed.
55 You may serve it with a static server:
56   npm install -g serve
57   serve -s build
58 Find out more about deployment here:
59   https://cra.link/deployment
60 Saving cache for successful job
61 Creating cache main-protected...
62 node_modules/: found 87034 matching artifact files and directories
63 Uploading cache.zip to https://storage.googleapis.com/gitlab-com-runners-cache/project/46387474/main-protected
64 Created cache
65 Cleaning up project directory and file based variables
66 Job succeeded
```



Lors de l'exécution du job docker-build, le scan de l'image Docker a détecté deux vulnérabilités de degré moyen:

```
200 $ trivy image kouss/client
201 2023-05-29T15:11:27.532Z      INFO  Need to update DB
202 2023-05-29T15:11:27.532Z      INFO  DB Repository: ghcr.io/aquasecurity/trivy-db
203 2023-05-29T15:11:27.532Z      INFO  Downloading DB...
204 28.34 MiB / 37.40 MiB [=====>] 75.78% ? p/s 37.40 MiB / 37.40 MiB [->] 100.00% ? p/s 37.40 MiB / 37.40 MiB [=====>] 100.00% 15.10 MiB p/s ETA 0s37.40 MiB / 37.40 MiB [->] 100.00% 15.10 MiB p/s ETA 0s37.40 MiB / 37.40 MiB [->] 100.00% 15.10 MiB p/s ETA 0s37.40 MiB / 37.40 MiB [->] 100.00% 34.60 MiB p/s 1.3s2023-05-29T15:11:29.184Z
INFO  Vulnerability scanning is enabled
205 2023-05-29T15:11:29.184Z      INFO  Secret scanning is enabled
206 2023-05-29T15:11:29.184Z      INFO  If your scanning is slow, please try '--scanners vuln' to disable secret scanning
207 2023-05-29T15:11:29.184Z      INFO  Please see also https://aquasecurity.github.io/trivy/v0.41/docs/secret/scanning/#recommendation for faster secret detection
208 2023-05-29T15:11:40.188Z      INFO  Detected OS: alpine
209 2023-05-29T15:11:40.188Z      INFO  Detecting Alpine vulnerabilities...
210 2023-05-29T15:11:40.189Z      INFO  Number of language-specific files: 1
211 2023-05-29T15:11:40.189Z      INFO  Detecting node-pkg vulnerabilities...
212 kouss/client (alpine 3.17.3)
213 =====
214 Total: 2 (UNKNOWN: 0, LOW: 0, MEDIUM: 2, HIGH: 0, CRITICAL: 0)
215
216  Library | Vulnerability | Severity | Installed Version | Fixed Version | Title
217
218  libcrypto3 | CVE-2023-1255 | MEDIUM | 3.0.8-r3 | 3.0.8-r4 | Input buffer over-read in AES-XTS implementation on 64 bit ARM
219
220
221  libssl3
222
223
224
225
```

Duration: 2 minutes 44 seconds
Finished: 1 minute ago
Queued: 0 seconds
Timeout: 1h (from project) ?
Runner: #12270859 (xS6Vzvpoq) 5-green.shared.runners-manager.gitlab.com/default

Commit [5ea57cda](#) ↻
Update .gitlab-ci.yml file

Pipeline #882105886 for main ↻

package

→ docker-build

Suite de résultat de l'exécution du Job docker-build :

```
226 $ docker push kouss/client
227 Using default tag: latest
228 The push refers to repository [docker.io/kouss/client]
229 0dcfa0425cf4: Preparing
230 1655949f5ca4: Preparing
231 ee7b1cb6f13d: Preparing
232 3cddc64f59e2: Preparing
233 b951f8a113f5: Preparing
234 1d8bcb7a961e: Preparing
235 f1417ff83b31: Preparing
236 1d8bcb7a961e: Waiting
237 f1417ff83b31: Waiting
238 b951f8a113f5: Layer already exists
239 3cddc64f59e2: Layer already exists
240 1d8bcb7a961e: Layer already exists
241 f1417ff83b31: Layer already exists
242 ee7b1cb6f13d: Pushed
243 1655949f5ca4: Pushed
244 0dcfa0425cf4: Pushed
245 latest: digest: sha256:48fef43d23f13cddcf40759b804418ee2d053232a3a6b87a341a78c8be24086d size: 1786
246 Saving cache for successful job
247 Creating cache default-protected...
248 WARNING: node_modules/: no matching files. Ensure that the artifact path is relative to the working directory (/builds/khadijaoussakel/client-microservice)
249 Archive is up to date!
250 Created cache
251 Cleaning up project directory and file based variables
252 Job succeeded
```

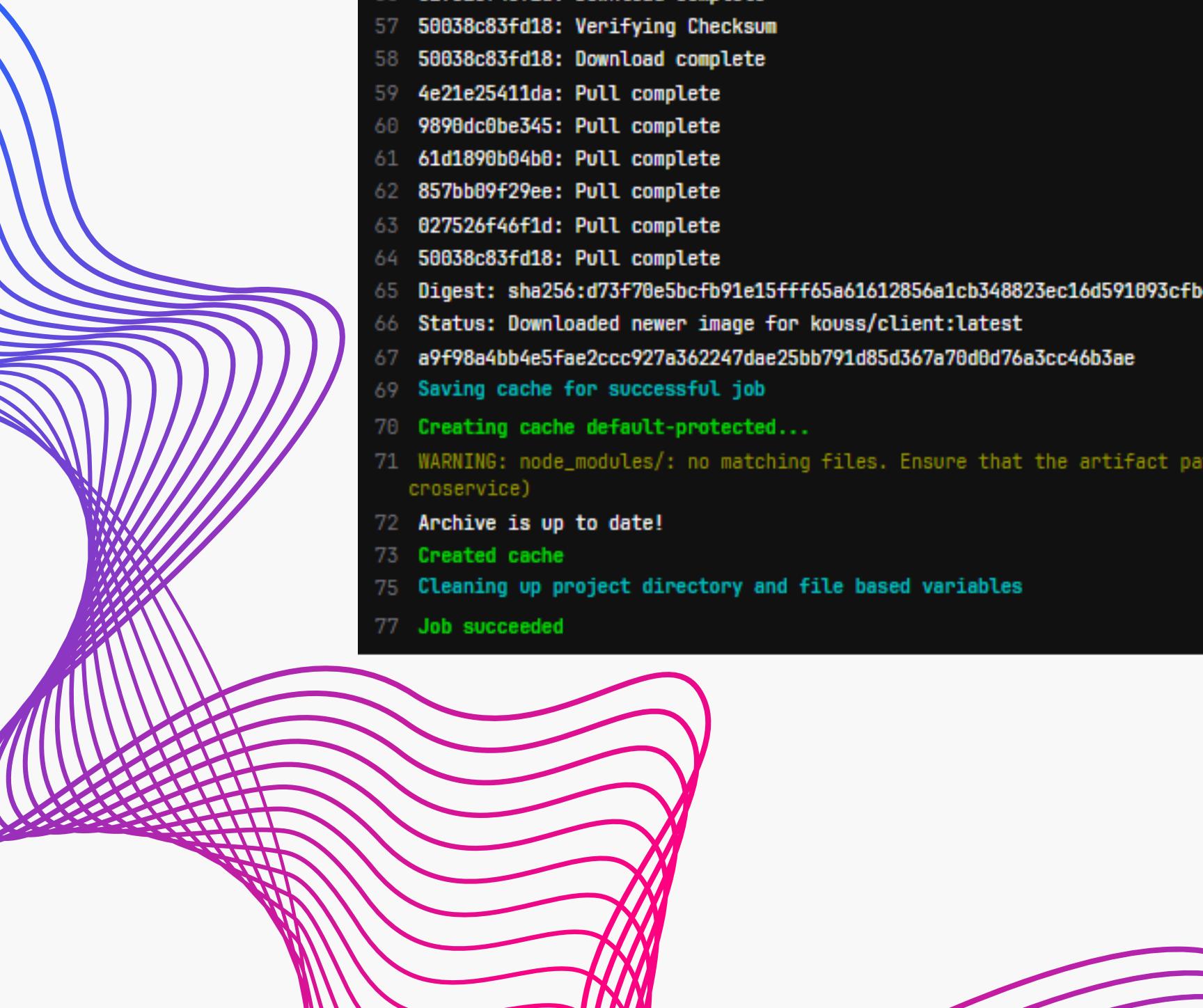
Timeout: 1h (from project) [?](#)
Runner: #12270859 (xS6Vzpv0q) 5-green.shared.runners-manager.gitlab.com/default

Commit [Sea57cda](#) [diff](#)
Update .gitlab-ci.yml file

[Pipeline #882105886 for main](#) [diff](#)
package

→ [docker-build](#)

Résultat de l'exécution du Job docker-deploy:



```
56 027526f46f1d: Download complete
57 50038c83fd18: Verifying Checksum
58 50038c83fd18: Download complete
59 4e21e25411da: Pull complete
60 9890dc0be345: Pull complete
61 61d1890b04b0: Pull complete
62 857bb09f29ee: Pull complete
63 027526f46f1d: Pull complete
64 50038c83fd18: Pull complete
65 Digest: sha256:d73f70e5bcfb91e15fff65a61612856a1cb348823ec16d591093cfb4bda32eb2
66 Status: Downloaded newer image for kouss/client:latest
67 a9f98a4bb4e5fae2ccc927a362247dae25bb791d85d367a70d0d76a3cc46b3ae
69 Saving cache for successful job
70 Creating cache default-protected...
71 WARNING: node_modules/: no matching files. Ensure that the artifact path is relative to the working directory (/builds/khadijaoussakel/client-microservice)
72 Archive is up to date!
73 Created cache
75 Cleaning up project directory and file based variables
77 Job succeeded
```

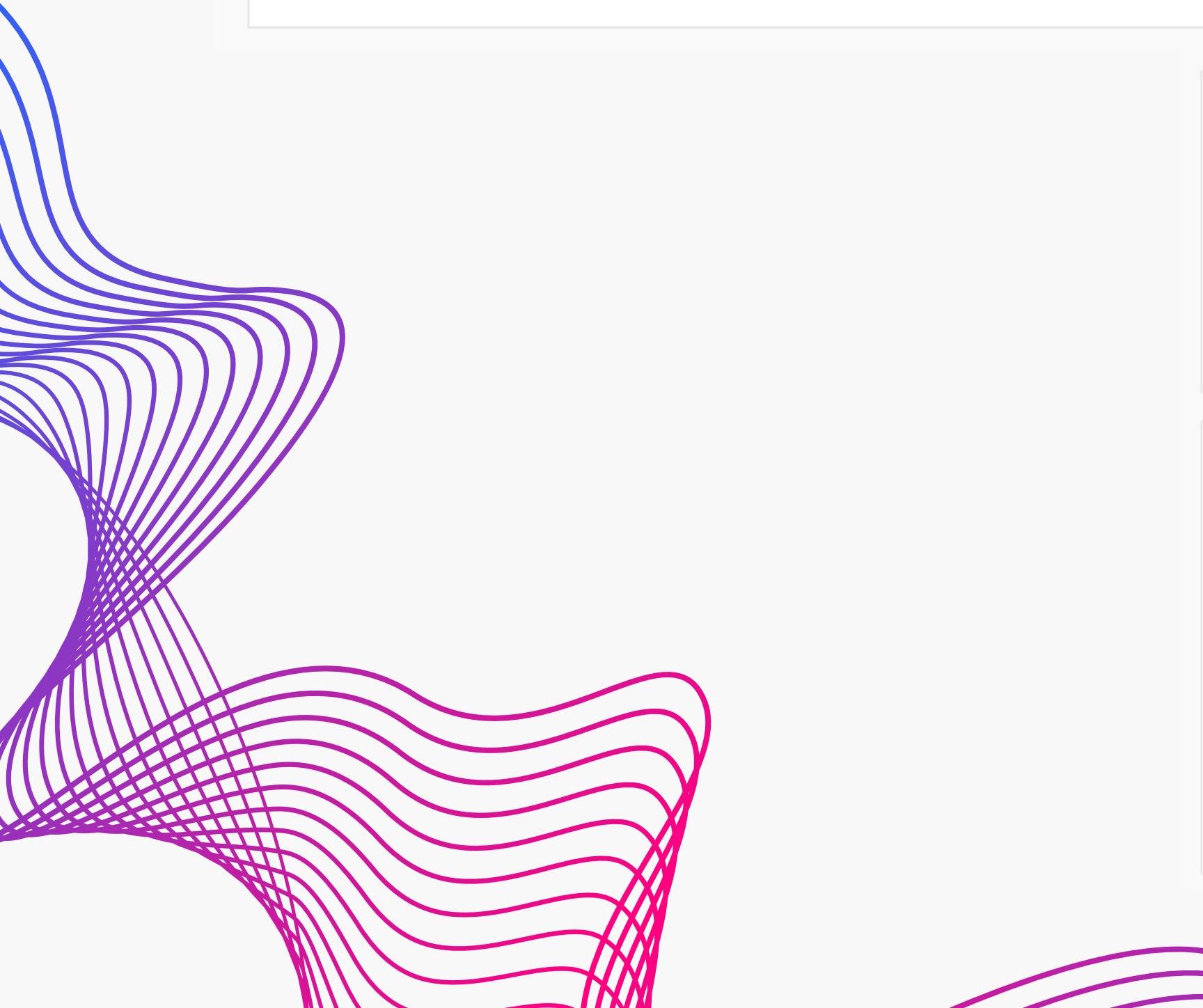


✓ Pipeline #881964355 for main []

deploy

→ docker-deploy

L'image construite "kouss/client" a été bien poussée vers Docker Hub :



The screenshot shows the Docker Hub repository page for "kouss/client".

Repository Summary:

- Name: kouss / client
- Contains: Image | Last pushed: 7 minutes ago
- Status: Inactive
- Stars: 0
- Downloads: 6
- Type: Public

Description:
This repository does not have a description.

Last pushed: 8 minutes ago

Tags:
This repository contains 1 tag(s).

Tag	OS	Type	Pulled	Pushed
latest		Image	7 minutes ago	8 minutes ago

[See all](#) [Go to Advanced Image Management](#)



En utilisant Kubernetes dans le processus de déploiement automatisé, l'orchestration des conteneurs est simplifiée et optimisée. Cela permet de bénéficier de la mise à l'échelle, de la résilience et de la gestion avancée des applications offertes par Kubernetes, tout en intégrant ces fonctionnalités dans le pipeline GitLab CI/CD pour un déploiement continu et automatisé du microservice.



kubernetes

La sortie "docker desktop" obtenue en exécutant la commande kubectl config current-context indique que le contexte actuellement sélectionné dans la configuration de kubectl est associé à Docker Desktop.

```
● PS C:\Users\HP\Desktop\FurniShop\k8s> kubectl config current-context  
docker-desktop
```

Ensuite on crée un namespace "furnishop". Un namespace dans Kubernetes est une abstraction logique qui permet de diviser un cluster en plusieurs environnements virtuels isolés.

```
● PS C:\Users\HP\Desktop\FurniShop\k8s> kubectl create namespace furnishop  
namespace/furnishop created
```

On crée le Kubernetes manifest (fichier YAML) nécessaire pour déployer le microservice frontend avec un déploiement (Deployment) et un service (Service) dans un cluster Kubernetes:

```
k8s > ! clientyaml > {} spec > {} template > {} spec > [ ]containers > {} 0 > {} resources > {} requests  
      io.k8s.api.apps.v1.Deployment (v1@deployment.json) | io.k8s.api.core.v1.Service (v1@service.json)  
1  apiVersion: v1  
2  kind: Service  
3  metadata:  
4    name: frontend-service  
5  spec:  
6    ports:  
7      - port: 3000  
8    selector:  
9      app: frontend  
10   ---  
  
11  ---  
12  apiVersion: apps/v1  
13  kind: Deployment  
14  metadata:  
15    name: frontend-deployment  
16  spec:  
17    replicas: 2  
18    selector:  
19      matchLabels:  
20        app: frontend  
21    template:  
22      metadata:  
23        labels:  
24          app: frontend  
25    spec:  
26      containers:  
27        - name: frontend  
28          image: kouss/client:1.0.0  
29          ports:  
30            - containerPort: 3000  
31            name: http-port  
32          resources:  
33            limits:  
34              cpu: "0.5"  
35              memory: "1Gi"  
36            requests:  
37              cpu: "0.2"  
38              memory: "256Mi"
```

"replicas: 2" indique que nous voulons deux répliques du microservice frontend.

On spécifie les limites et les demandes de ressources pour le conteneur.

En utilisant ce fichier YAML avec la commande `kubectl apply -f frontend.yaml`, le déploiement avec deux répliques du microservice frontend sera créé, ainsi que le service pour acheminer le trafic vers ces pods:

```
● PS C:\Users\HP\Desktop\FurniShop\k8s> kubectl apply -f client.yaml -n furnishop
service/frontend-service created
deployment.apps/frontend-deployment created
```

```
● PS C:\Users\HP\Desktop\FurniShop\k8s> kubectl get all -n furnishop
NAME                                         READY   STATUS    RESTARTS   AGE
pod/frontend-deployment-6c77fd6dcc-fcx5s   1/1     Running   0          26s
pod/frontend-deployment-6c77fd6dcc-m7qcq   1/1     Running   0          26s

NAME           TYPE      CLUSTER-IP      EXTERNAL-IP      PORT(S)      AGE
service/frontend-service   ClusterIP   10.104.30.192   <none>        3000/TCP   26s

NAME           READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/frontend-deployment   2/2     2           2           26s

NAME           DESIRED  CURRENT   READY   AGE
replicaset.apps/frontend-deployment-6c77fd6dcc  2        2         2       26s
```

Microservice produit

Node.js/Express et MongoDB, Dockerfile, Gitlab CI/CD, Kubernetes



En combinant Node.js, Express et MongoDB, le microservice "produit" peut être développé de manière efficace et flexible. Node.js fournit une plateforme d'exécution rapide et évolutive pour exécuter le code JavaScript du microservice. Express simplifie la création des routes et des fonctionnalités web nécessaires pour l'API du microservice. MongoDB permet de stocker les données des produits de manière flexible et de les interroger efficacement grâce à son modèle de données orienté documents.

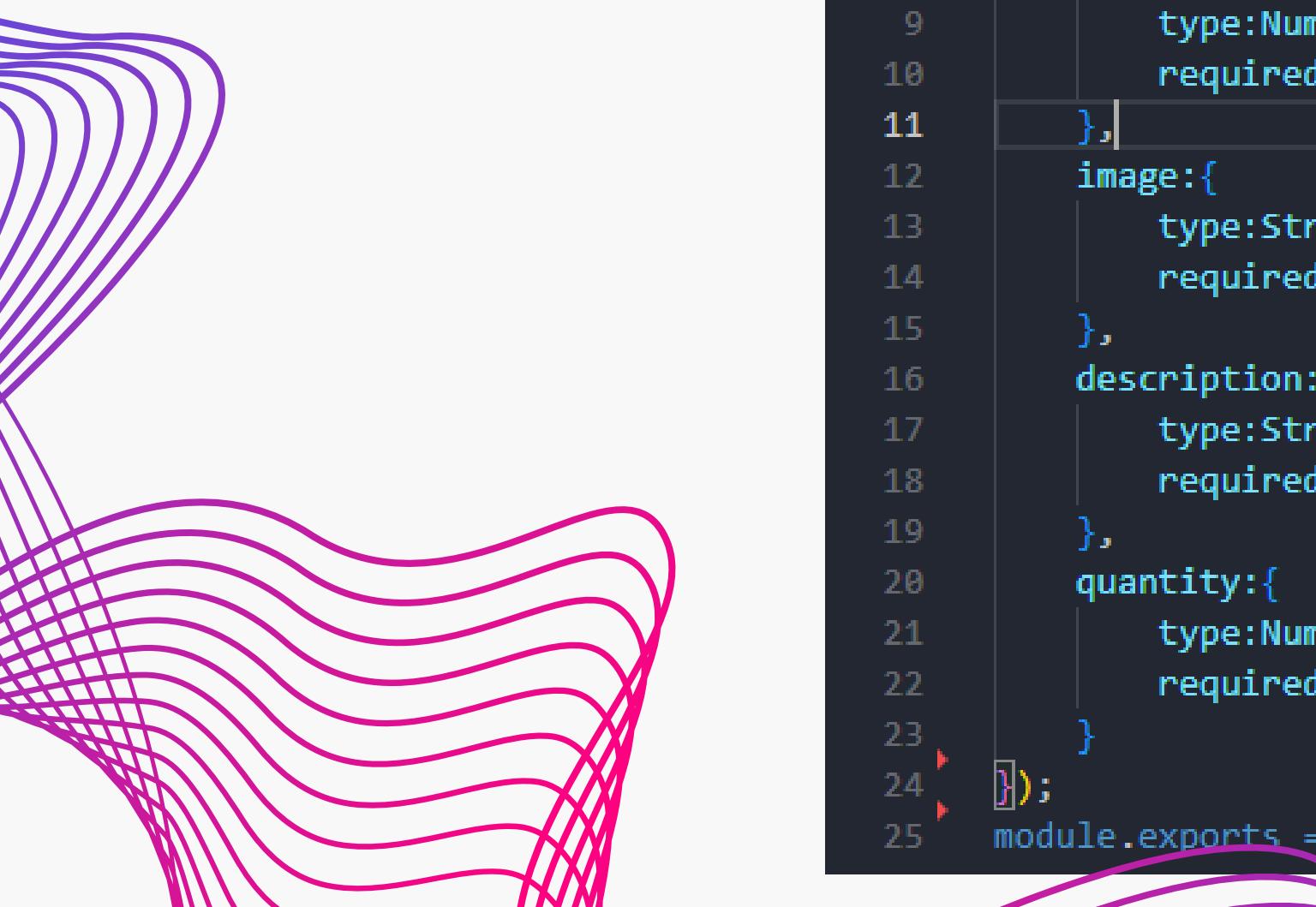


On crée le fichier server.js qui configure et démarre le serveur Express pour le microservice, établit la connexion avec la base de données MongoDB, définit les routes pour les items et écoute les requêtes entrantes sur le port 5000.



```
JS server.js M, M ●
produit > JS server.js > ...
1  const express = require('express');
2  const mongoose = require('mongoose');
3  const items = require('./routes/api/items');
4  const app = express();
5  app.use(express.json());
6  var cors = require('cors')
7
8  app.use(cors())
9  const db = require('./config/keys').mongoURI
10
11 mongoose.connect(db)
12   .then( ()=>console.log('MongoDB connected ...') )
13   .catch( err=>console.log(err));
14
15 app.use('/api/items' , items);
16
17 const port = process.env.PORT || 5000 ;
18 app.listen(port , ()=>console.log(`Server started on port ${port}`)) ;
19
```

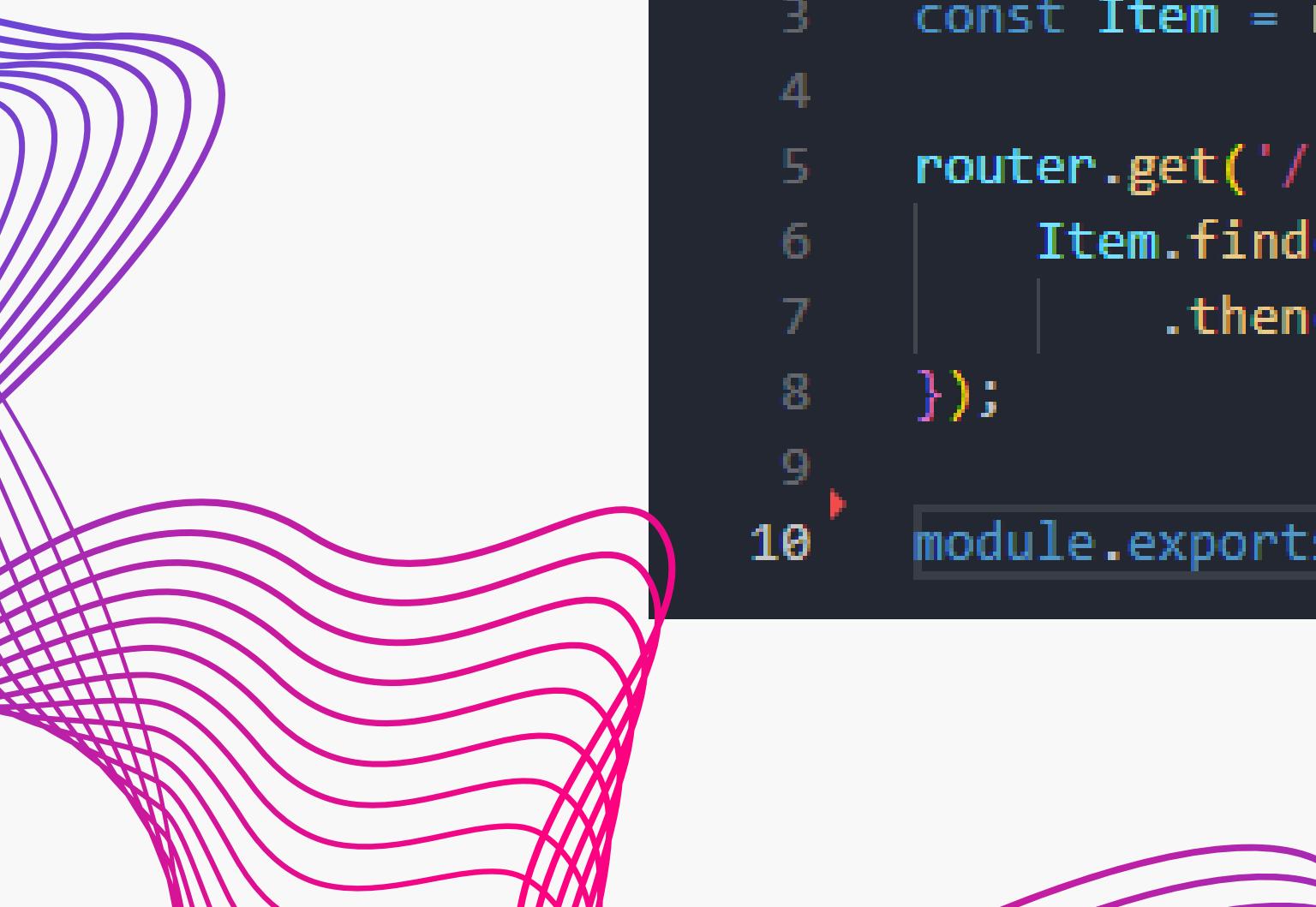
Ensute on crée le fichier item.js dans le dossier models qui définit le schéma de données pour le modèle Item utilisé dans le microservice.



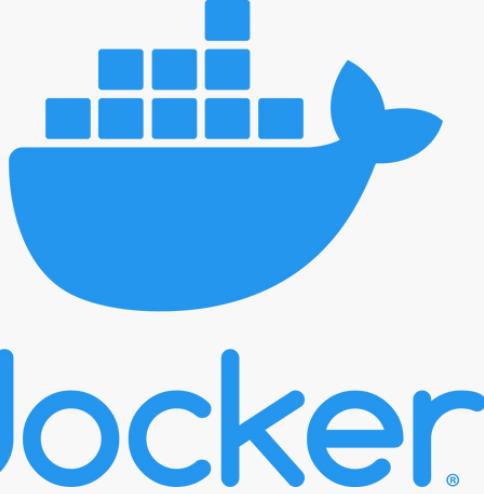
```
JS item.js M, M X

produit > models > JS item.js > [o] ItemSchema
1  const mongoose = ...require('mongoose');
2  const Schema = mongoose.Schema ;
3  const ItemSchema = new Schema([
4    name:{
5      type:String,
6      required:true
7    },
8    price:{
9      type:Number,
10     required:true
11  },
12    image:{
13      type:String,
14      required:true
15  },
16    description:{
17      type:String,
18      required:true
19  },
20    quantity:{
21      type:Number,
22      required:true
23  }
24]);
25 module.exports = Item = mongoose.model('item', ItemSchema);
```

Dans le fichier items.js, on définit la route GET pour récupérer tous les items du microservice. Lorsque la route est appelée, le fichier utilise le modèle Item pour récupérer les items depuis la base de données MongoDB et les renvoie en tant que réponse JSON.



```
JS items.js M, M X
produit > routes > api > JS items.js > ...
1 const express = require('express');
2 const router = express.Router();
3 const Item = require('../models/item');
4
5 router.get('/', (req,res)=>{
6   Item.find()
7     .then( items=>res.json(items) )
8 );
9
10 module.exports = router ;
```



On a créé le fichier Dockerfile, qui définit les étapes pour construire l'image Docker du service produit, tout en respectant les bonnes pratiques.

Cette ligne spécifie l'image de base à utiliser pour le conteneur Docker. Dans ce cas, on a choisi l'image node:16-alpine, qui est une version légère de l'image Node.js

En utilisant l'utilisateur node à l'intérieur du conteneur, on applique le principe du moindre privilège en limitant les permissions et en réduisant les risques de sécurité.

On utilise npm ci au lieu de npm install pour garantir une installation plus rapide et reproductible des dépendances. L'option --only=production permet d'installer uniquement les dépendances nécessaires pour l'environnement de production, ce qui réduit la taille de l'image.

```
produit > ⚡ dockerfile > ...
1 FROM node:16-alpine
2 USER node
3 ENV NODE_ENV=production
4 WORKDIR /produit
5 COPY --chown=node:node package.json package-lock.json .
6 RUN npm ci --only=production
7 COPY --chown=node:node .
8 EXPOSE 5000
9 CMD ["node", "server.js"]
10
11
```

il exécute la commande "node server.js", ce qui démarre l'application Node.js en exécutant le fichier "server.js".

L'utilisation de --chown=node:node garantit que les fichiers appartiennent à l'utilisateur node à l'intérieur du conteneur.



On a créé également le fichier `.dockerignore` qui est utilisé pour spécifier les fichiers et répertoires à exclure lors de la construction de l'image Docker à partir du contexte de construction.

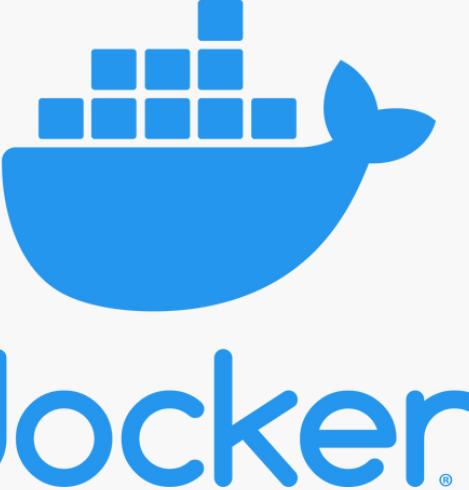
On a inclus le répertoire `node_modules` dans le fichier `.dockerignore` pour optimiser la taille de l'image Docker.

```
client > ➜ .dockerignore
1 node_modules
```



On construit l'image Docker à partir du Dockerfile présent dans le répertoire courant et on lui attribue le nom "kouss/produit" avec la balise "1.0.0"

- PS C:\Users\HP\Desktop\FurniShop\produit> docker build -t kouss/produit:1.0.0 .
[+] Building 10.6s (11/11) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 272B
=> [internal] load .dockerignore
=> => transferring context: 52B
=> [internal] load metadata for docker.io/library/node:16-alpine
=> [auth] library/node:pull token for registry-1.docker.io
=> CACHED [1/5] FROM docker.io/library/node:16-alpine@sha256:8d18e32fa398763c07407e9d407c64e6a3c2a18e9fa97362cfde669a6b6161ef
=> => resolve docker.io/library/node:16-alpine@sha256:8d18e32fa398763c07407e9d407c64e6a3c2a18e9fa97362cfde669a6b6161ef
=> [internal] load build context
=> => transferring context: 6.66MB
=> [2/5] WORKDIR /produit
=> [3/5] COPY --chown=node:node package.json package-lock.json ./
=> [4/5] RUN npm ci --only=production
=> [5/5] COPY --chown=node:node . .
=> exporting to image
=> => exporting layers
=> => writing image sha256:dd97b24a824dc8bda786421d8c801e70451f88b4ace437371b02321eb133a0ca
=> => naming to docker.io/kouss/produit:1.0.0



On exécute un conteneur à partir de l'image "kouss/produit:1.0.0" et effectue un mappage de port pour permettre l'accès au service du conteneur sur le port 5000 de l'hôte.

```
PS C:\Users\HP\Desktop\FurniShop> docker run -p 5000:5000 kouss/produit:1.0.0
Server started on port 5000
MongoDB connected ...
[]
```



GitLab

Pour le microservice "produit", nous créons un nouveau référentiel GitLab avec le nom "produit-microservice". Ensuite, nous effectuons les étapes similaires à celles du microservice frontend pour pousser les modifications vers la branche distante. Cela inclut l'ajout des fichiers nécessaires, la création du fichier `.gitignore`

The screenshot shows a GitLab repository interface for a project named "client microservice".

Commit Details:
A single commit by "khadija oussakel" is shown, authored 21 minutes ago. The commit hash is "c20a795c".

File Navigation:
The repository path is "main / client-microservice /". There are buttons for "Find file", "Web IDE", "Download", and "Clone".

File Actions:
Buttons for "README", "CI/CD configuration", "Add LICENSE", "Add CHANGELOG", "Add CONTRIBUTING", "Add Kubernetes cluster", and "Add Wiki".

Integrations:
A button for "Configure Integrations".

File List:
A table lists the files in the repository, all last updated 21 minutes ago by "client microservice".

Name	Last commit	Last update
public	client microservice	21 minutes ago
src	client microservice	21 minutes ago
.dockerignore	client microservice	21 minutes ago
.gitignore	client microservice	21 minutes ago
README.md	client microservice	21 minutes ago
dockerfile	client microservice	21 minutes ago
package-lock.json	client microservice	21 minutes ago
package.json	client microservice	21 minutes ago

Maintenant on crée le fichier `gitlab-ci.yml` avec quatre stages similaire à celui du microservice frontend:

On inclut le fichier **Security/SAST.gitlab-ci.yml** pour gérer l'analyse statique du code source.

La section **cache** spécifie les chemins des fichiers ou des répertoires à mettre en cache entre les exécutions du pipeline. Dans ce cas, le répertoire `node_modules/` est mis en cache pour accélérer les builds futurs.

L'étape sast est définie dans la section `securityScan`. Cette étape est **responsable de l'analyse statique du code source**.

Les artefacts produits par cette étape sont un rapport de sécurité au format JSON nommé `gl-sast-report.json`. En plus si l'analyse statique de sécurité ne réussit pas, le pipeline entier sera considéré comme échoué

Le cache spécifique à la branche est utilisé pour stocker les dépendances installées dans le répertoire `node_modules/`. Le script exécute la commande `npm ci` pour installer les dépendances du projet.

```

.gitlab-ci.yml 1.22 KiB
1 image: docker:latest
2 services:
3   - docker:dind
4
5 stages:
6   - securityScan
7   - build
8   - package
9   - deploy
10
11 include:
12   - template: Security/SAST.gitlab-ci.yml
13
14 cache:
15   paths:
16     - node_modules/
17
18 sast:
19   stage : securityScan
20   artifacts:
21     reports:
22       sast: gl-sast-report.json
23       allow_failure: false
24
25 build:
26   image: node:16
27   stage: build
28   cache:
29     key: "$CI_COMMIT_REF_SLUG" # Utilisez une clé de cache spécifique à la branche
30     paths:
31       - node_modules/
32   script:
33     - npm ci           # Utiliser le cache pour télécharger les dépendances

```



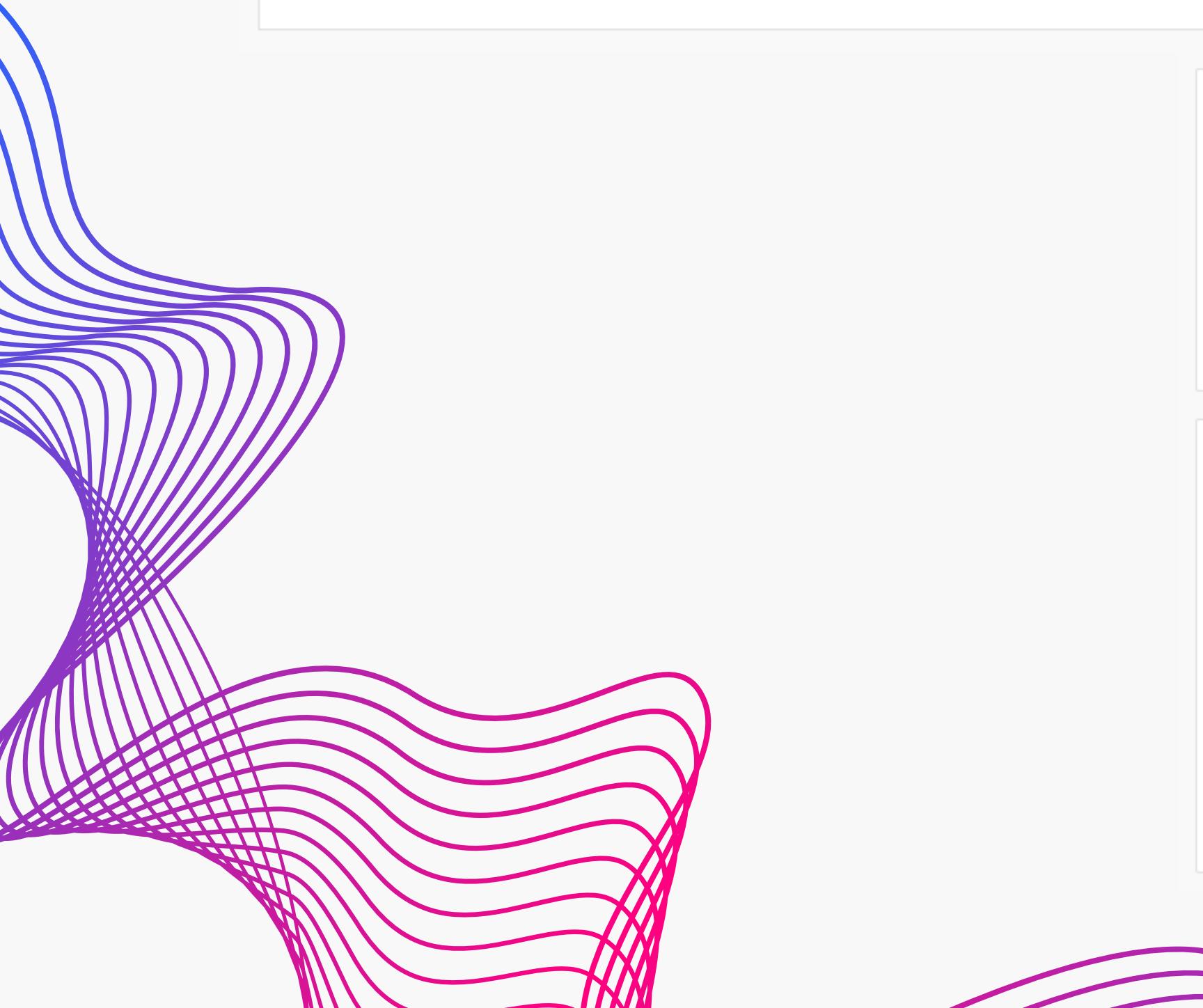
GitLab

L'étape docker-build est définie dans la section package. Avant de construire l'image Docker, le script exécute quelques commandes de préparation, notamment la connexion à Docker Hub, l'installation de curl et jq, et l'installation de trivy (un outil de sécurité pour les conteneurs). Ensuite, le script utilise Docker pour tirer l'image de base node:16, construire une nouvelle image nommée kouss/produit:1.0.0 **en utilisant le cache de l'image de base, exécute une analyse de sécurité sur l'image avec trivy**, puis pousse l'image construite vers Docker Hub.

L'étape docker-deploy est définie dans la section deploy. Le script exécute une commande Docker pour déployer l'image kouss/produit:1.0.0 dans un conteneur nommé client, en le reliant au port 5000 sur la machine hôte.

```
35 docker-build:
36   stage: package
37   before_script:
38     - docker login -u kouss -p $DOCKER_HUB_PASSWORD
39     - apk add --no-cache curl jq
40     - curl -sSfL https://raw.githubusercontent.com/aquasecurity/trivy/main/contrib/install.sh | sh -s -- -b /usr/local/bin
41   script:
42     - docker pull node:16 # Tirer l'image de base avant de construire pour utiliser le cache
43     - docker build --cache-from=node:16 -t kouss/produit:1.0.0 . # Utiliser le cache de l'image
44     - trivy image kouss/produit:1.0.0
45     - docker push kouss/produit:1.0.0
46
47
48 docker-deploy:
49   stage : deploy
50   script :
51     - docker run -d -p 5000:5000 --name produit kouss/produit:1.0.0
52
53
```

L'image construite "kouss/produit" a été bien poussée vers Docker Hub :



The screenshot shows the Docker Hub repository page for "kouss / produit".

Repository Summary:

- Name: kouss / produit
- Contains: Image
- Last pushed: 17 minutes ago
- Status: Inactive
- Stars: 0
- Downloads: 1
- Type: Public

Description:
This repository does not have a description.

Last pushed: 17 minutes ago

Tags:
This repository contains 1 tag(s).

Tag	OS	Type	Pulled	Pushed
latest	🐧	Image	17 minutes ago	17 minutes ago

[See all](#) [Go to Advanced Image Management](#)



kubernetes

Tout d'abord on s'assure qu'on dispose des droits nécessaires pour créer des secrets dans le cluster Kubernetes.

```
PS C:\Users\HP\Desktop\FurniShop\k8s> kubectl auth can-i create secrets  
yes
```

On crée un secret Kubernetes pour stocker la valeur de mongoURI. Cette approche permet de sécuriser les informations sensibles, comme les informations de connexion à la base de données, en les stockant dans des secrets plutôt que de les exposer directement dans le fichier YAML du déploiement.

```
PS C:\Users\HP\Desktop\FurniShop\k8s> kubectl create secret generic mongodb-credentials --from-literal=mongoURI='mongodb+srv://khadijaoussakel  
56:oussakel@oussakel.8loey8k.mongodb.net/?retryWrites=true&w=majority' -n furnishop  
secret/mongodb-credentials created
```

Cela crée un secret nommé mongodb-credentials contenant la valeur de mongoURI

```
● PS C:\Users\HP\Desktop\FurniShop\k8s> kubectl get secrets mongodb-credentials -n furnishop  
○ NAME          TYPE      DATA  AGE  
mongodb-credentials  Opaque    1     70s  
PS C:\Users\HP\Desktop\FurniShop\k8s>
```



kubernetes

On crée le Kubernetes manifest (fichier YAML) nécessaire pour déployer le microservice produit avec un déploiement (Deployment) et un service (Service) dans un cluster Kubernetes:

```
34  ---  
35  apiVersion: v1  
36  kind: Service  
37  metadata:  
38    name: produit-backend-service  
39  spec:  
40    ports:  
41      - port: 5000  
42    selector:  
43      app: produit-backend  
44
```

En spécifiant cette configuration, le conteneur du déploiement pourra accéder à la valeur du **secret mongodb-credentials** et l'utiliser comme valeur de la variable d'environnement MONGO_URI.

On spécifie **les limites et les demandes de ressources** pour le conteneur.

```
k8s > ! produit.yaml > {} spec > {} template > {} spec > [ ] containers > {} o > [ ] ports  
io.k8s.api.core.v1.Service(v1@service.json) | io.k8s.api.apps.v1.Deployment(v1@deployment.json)  
1  apiVersion: apps/v1  
2  kind: Deployment  
3  metadata:  
4    name: produit-backend  
5  spec:  
6    replicas: 2  
7    selector:  
8      matchLabels:  
9        app: produit-backend  
10   template:  
11     metadata:  
12       labels:  
13         app: produit-backend  
14   spec:  
15     containers:  
16       - name: produit-backend  
17         image: kouss/produit:1.0.0  
18         ports:  
19           - containerPort: 5000  
20         env:  
21           - name: MONGO_URI  
22             valueFrom:  
23               secretKeyRef:  
24                 name: mongodb-credentials  
25                 key: mongoURI  
26     resources:  
27       limits:  
28         cpu: 500m  
29         memory: 512Mi  
30       requests:  
31         cpu: 200m  
32         memory: 256Mi
```



kubernetes

En utilisant ce fichier YAML avec la commande `kubectl apply -f produit.yaml`, le déploiement avec **deux répliques** du microservice produit sera créé, ainsi que le service pour acheminer le trafic vers ces pods:

```
● PS C:\Users\HP\Desktop\FurniShop\k8s> kubectl apply -f produit.yaml -n furnishop
deployment.apps/produit-backend created
service/produit-backend-service created
PS C:\Users\HP\Desktop\FurniShop\k8s>
```

```
PS C:\Users\HP\Desktop\FurniShop\k8s> kubectl get all -n furnishop
● NAME                                         READY   STATUS    RESTARTS   AGE
pod/frontend-deployment-6c77fd6dcc-x727s   1/1     Running   0          3s
pod/frontend-deployment-6c77fd6dcc-zsskl   1/1     Running   0          3s
pod/produit-backend-5f55947798-56hmg       1/1     Running   0          8m4s
pod/produit-backend-5f55947798-hkxf4        1/1     Running   0          8m4s

NAME                           TYPE      CLUSTER-IP      EXTERNAL-IP      PORT(S)      AGE
service/frontend-service       ClusterIP  10.96.242.187  <none>           3000/TCP    3s
service/produit-backend-service ClusterIP  10.108.3.32   <none>           5000/TCP    8m4s

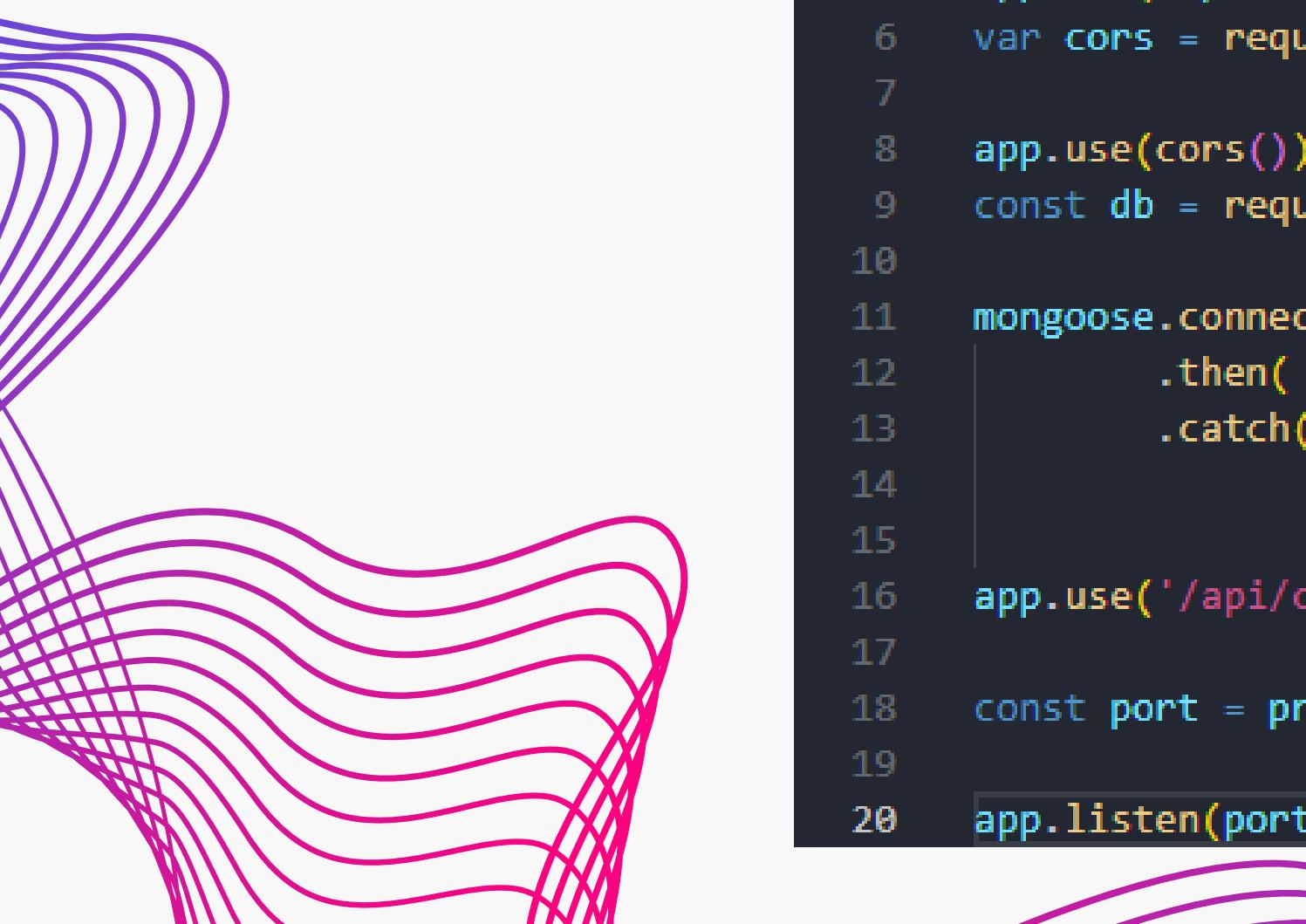
NAME                           READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/frontend-deployment  2/2     2           2           3s
deployment.apps/produit-backend       2/2     2           2           8m4s

NAME                           DESIRED  CURRENT  READY   AGE
replicaset.apps/frontend-deployment-6c77fd6dcc  2        2        2       3s
replicaset.apps/produit-backend-5f55947798       2        2        2       8m4s
```

Microservice commande

Node.js/Express et MongoDB, Dockerfile, Gitlab CI/CD, Kubernetes

Le fichier server.js configure et démarre un serveur Express pour une application web. Il importe les modules nécessaires tels que Express et Mongoose, et définit les routes pour les commandes. Il configure également la connexion à la base de données MongoDB. Le middleware CORS est utilisé pour gérer les demandes d'origine croisée. Le serveur écoute sur le port 8000:



```
JS server.js ●

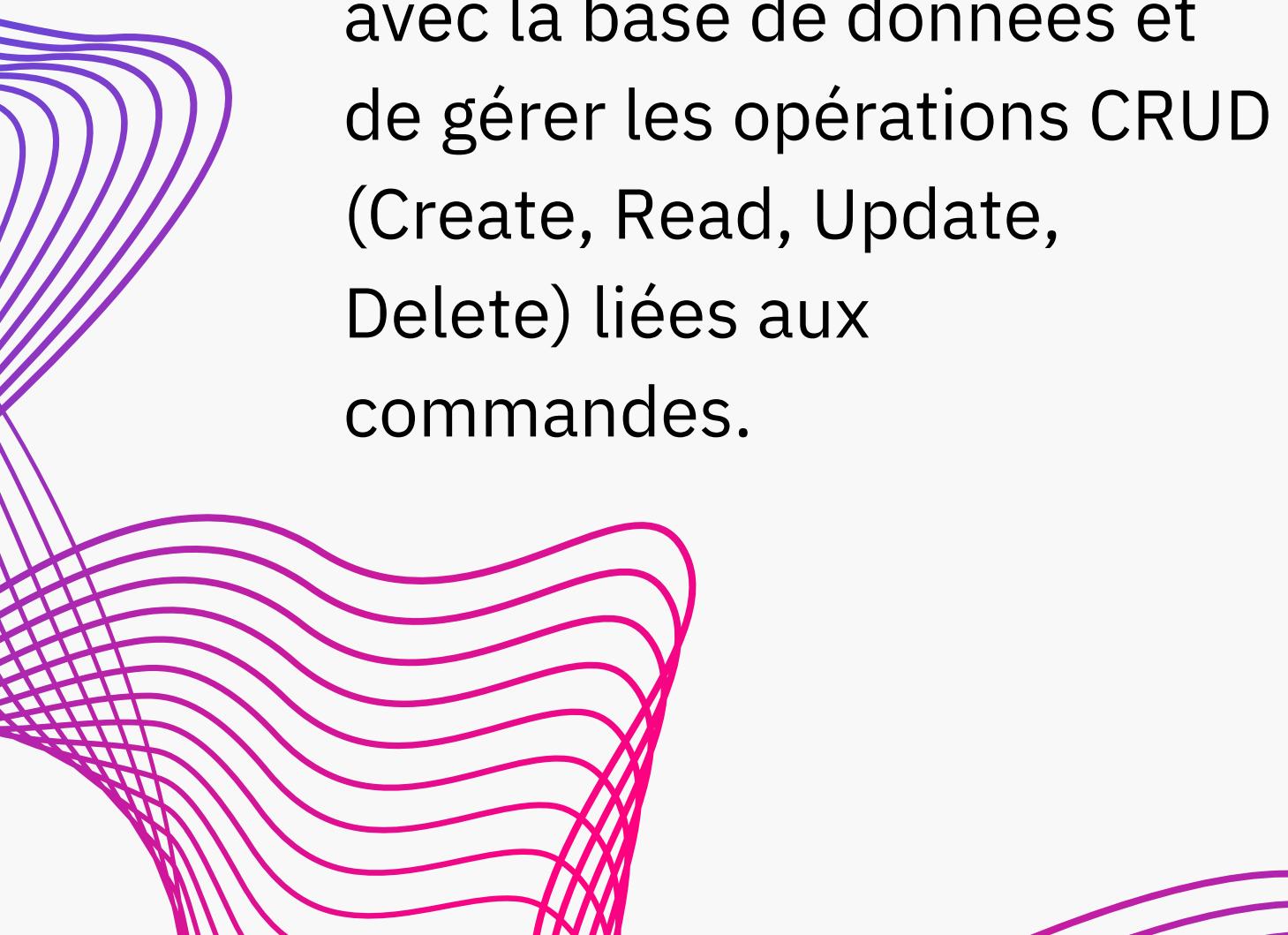
commande > JS server.js > ...

1  const express = require('express');
2  const mongoose = require('mongoose');
3  const commandes = require('../routes/api/commandes');
4  const app = express();
5  app.use(express.json());
6  var cors = require('cors')
7
8  app.use(cors())
9  const db = require('../config/keys').mongoURI
10
11 mongoose.connect(db)
12   .then( ()=>console.log('MongoDB connected ...') )
13   .catch( err=>console.log(err));
14
15
16 app.use('/api/commandes' , commandes);
17
18 const port = process.env.PORT || 8000 ;
19
20 app.listen(port , ()=>console.log(`Server started on port ${port}`) ) ;
```

le fichier commande.js définit le schéma de données pour une commande avec des champs tels que l'identifiant de l'article, la date de commande, la quantité et l'état de paiement:



```
JS commande.js ●
commande > models > JS commande.js > ...
1  const mongoose = require('mongoose');
2  const Schema = mongoose.Schema ;
3
4  const CommandeSchema = new Schema({
5    itemId:{
6      type:Object,
7      required:true
8    },
9    dateCommande:{
10      type:Date,
11      default:Date.now
12    },
13    quantity:{
14      type:Number,
15      required:true
16    },
17    commandePayee:{
18      type:Boolean,
19      default:false
20    }
21  });
22 });
23
24 module.exports = Commande = mongoose.model('commande', CommandeSchema);
```



le fichier commandes.js définit les routes pour obtenir, créer, mettre à jour et supprimer des commandes dans une application. Ces routes permettent d'interagir avec la base de données et de gérer les opérations CRUD (Create, Read, Update, Delete) liées aux commandes.

```
JS commandes.js ●

commande > routes > api > JS commandes.js > ...

1  const express = require('express');
2  const router = express.Router();
3  const Commande = require('../models/commande');
4  router.get('/', (req,res)=>{
5      Commande.find()
6          .then( commandes=>res.json(commandes) )
7  });
8  router.post('/', (req,res)=>{
9      const newCommande = new Commande({
10          itemId:req.body.itemId,
11          quantity : req.body.quantity,
12      });
13      newCommande.save()
14          .then( commande => res.json(commande) )
15  });
16  router.put('/:id', (req, res, next) => {
17      Commande.findByIdAndUpdate(req.params.id, req.body)
18          .then(() => Commande.findById(req.params.id))
19          .then(commande => res.send(commande))
20          .catch(next);
21  });
22  router.delete('/:id' , (req,res)=>{
23      Commande.findByIdAndRemove(req.params.id)
24          .then( () => res.json( {success:true} ) )
25          .catch(err => res.status(404).json( {success:false} ) )
26  });
}
```



On a créé le fichier Dockerfile, qui définit les étapes pour construire l'image Docker du service commande, tout en respectant les bonnes pratiques.

Cette ligne spécifie l'image de base à utiliser pour le conteneur Docker. Dans ce cas, on a choisi l'image node:16-alpine, qui est une version légère de l'image Node.js

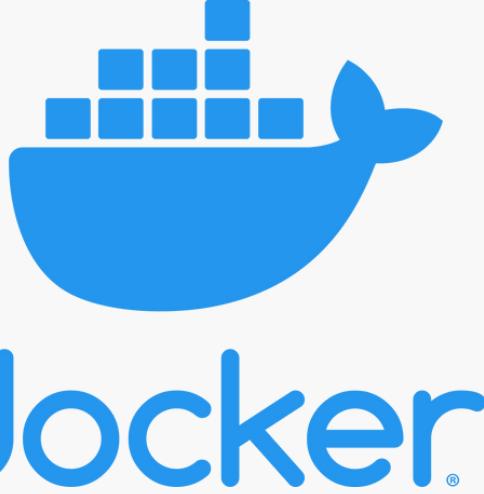
En utilisant l'utilisateur node à l'intérieur du conteneur, on applique le principe du moindre privilège en limitant les permissions et en réduisant les risques de sécurité.

On utilise npm ci au lieu de npm install pour garantir une installation plus rapide et reproductible des dépendances. L'option --only=production permet d'installer uniquement les dépendances nécessaires pour l'environnement de production, ce qui réduit la taille de l'image.

```
commande > ⚡ dockerfile > ...
1 FROM node:16-alpine
2 USER node
3 ENV NODE_ENV=production
4 WORKDIR /commande
5 COPY --chown=node:node package.json package-lock.json .
6 RUN npm ci --only=production
7 COPY --chown=node:node .
8 EXPOSE 8000
9 CMD ["node", "server.js"]
10
```

il exécute la commande "node server.js", ce qui démarre l'application Node.js en exécutant le fichier "server.js".

L'utilisation de --chown=node:node garantit que les fichiers appartiennent à l'utilisateur node à l'intérieur du conteneur.



On a créé également le fichier `.dockerignore` qui est utilisé pour spécifier les fichiers et répertoires à exclure lors de la construction de l'image Docker à partir du contexte de construction.

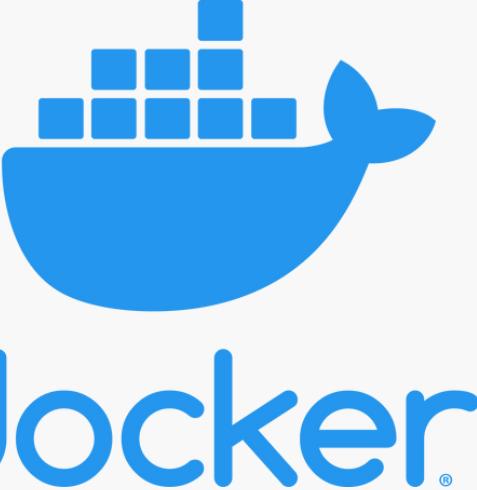
On a inclus le répertoire `node_modules` dans le fichier `.dockerignore` pour optimiser la taille de l'image Docker.

```
commande > 🐳 .dockerignore
1   node_modules
```



On construit l'image Docker à partir du Dockerfile présent dans le répertoire courant et on lui attribue le nom "kouss/commande" avec la balise "1.0.0"

```
PS C:\Users\HP\Desktop\FurniShop\commande> docker build -t kouss/commande:1.0.0 .
● [+] Building 25.0s (11/11) FINISHED
  => [internal] load build definition from Dockerfile
  => => transferring dockerfile: 271B
  => [internal] load .dockerignore
  => => transferring context: 52B
  => [internal] load metadata for docker.io/library/node:16-alpine
  => [auth] library/node:pull token for registry-1.docker.io
  => [internal] load build context
  => => transferring context: 108.00kB
  => [1/5] FROM docker.io/library/node:16-alpine@sha256:8d18e32fa398763c07407e9d407c64e6a3c2a18e9fa97362cfde669a6b6161ef
  => => resolve docker.io/library/node:16-alpine@sha256:8d18e32fa398763c07407e9d407c64e6a3c2a18e9fa97362cfde669a6b6161ef
  => CACHED [2/5] WORKDIR /commande
  => [3/5] COPY --chown=node:node package.json package-lock.json ./
  => [4/5] RUN npm ci --only=production
  => [5/5] COPY --chown=node:node . .
  => exporting to image
  => => exporting layers
  => => writing image sha256:bdf62e32778fab6cee72f8cdb7dbfb0f2ba90dc49c1644311d2ef755d61e9e52
  => => naming to docker.io/kouss/commande:1.0.0
```



On exécute un conteneur à partir de l'image "kouss/commande:1.0.0" et effectue un mappage de port pour permettre l'accès au service du conteneur sur le port 8000 de l'hôte.

```
PS C:\Users\HP\Desktop\FurniShop> docker run -p 8000:8000 kouss/commande:1.0.0
Server started on port 8000
MongoDB connected ...
[]
```



GitLab

On crée un nouveau référentiel GitLab avec le nom "commande-microservice" pour pousser le projet vers la branche distante :

The screenshot shows a GitLab repository page for a project named "commande-microservice". The repository has a Project ID of 46407907. It contains 1 Commit, 1 Branch, 0 Tags, and 0 Bytes Project Storage. The commit history shows a single commit from "khadija oussakel" authored 2 minutes ago, with the commit hash 75057e32. The repository structure includes files like README, CI/CD configuration, LICENSE, CHANGELOG, CONTRIBUTING, Kubernetes cluster, and Wiki. A "Configure Integrations" button is also present. The file list on the right shows the following details:

Name	Last commit	Last update
config	commande microservice	2 minutes ago
models	commande microservice	2 minutes ago
routes/api	commande microservice	2 minutes ago
.dockerignore	commande microservice	2 minutes ago
.gitignore	commande microservice	2 minutes ago
dockerfile	commande microservice	2 minutes ago
package-lock.json	commande microservice	2 minutes ago
package.json	commande microservice	2 minutes ago
server.js	commande microservice	2 minutes ago

Maintenant on crée le fichier `gitlab-ci.yml` avec quatre stages similaire à celui du microservice produit:

On inclut le fichier **`Security/SAST.gitlab-ci.yml`** pour gérer l'analyse statique du code source.

La section **cache** spécifie les chemins des fichiers ou des répertoires à mettre en cache entre les exécutions du pipeline. Dans ce cas, le répertoire `node_modules/` est mis en cache pour accélérer les builds futurs.

L'étape `sast` est définie dans la section `securityScan`. Cette étape est **responsable de l'analyse statique du code source**.

Les artefacts produits par cette étape sont un rapport de sécurité au format JSON nommé `gl-sast-report.json`. En plus si l'analyse statique de sécurité ne réussit pas, le pipeline entier sera considéré comme échoué

Le cache spécifique à la branche est utilisé pour stocker les dépendances installées dans le répertoire `node_modules/`. Le script exécute la commande `npm ci` pour installer les dépendances du projet.

```

.gitlab-ci.yml 1.22 KiB
1  image: docker:latest
2  services:
3    - docker:dind
4
5  stages:
6    - securityScan
7    - build
8    - package
9    - deploy
10
11 include:
12   - template: Security/SAST.gitlab-ci.yml
13
14 cache:
15   paths:
16     - node_modules/
17
18 sast:
19   stage : securityScan
20   artifacts:
21     reports:
22       sast: gl-sast-report.json
23       allow_failure: false
24
25 build:
26   image: node:16
27   stage: build
28   cache:
29     key: "$CI_COMMIT_REF_SLUG" # Utilisez une clé de cache spécifique à la branche
30     paths:
31       - node_modules/
32   script:
33     - npm ci           # Utiliser le cache pour télécharger les dépendances

```



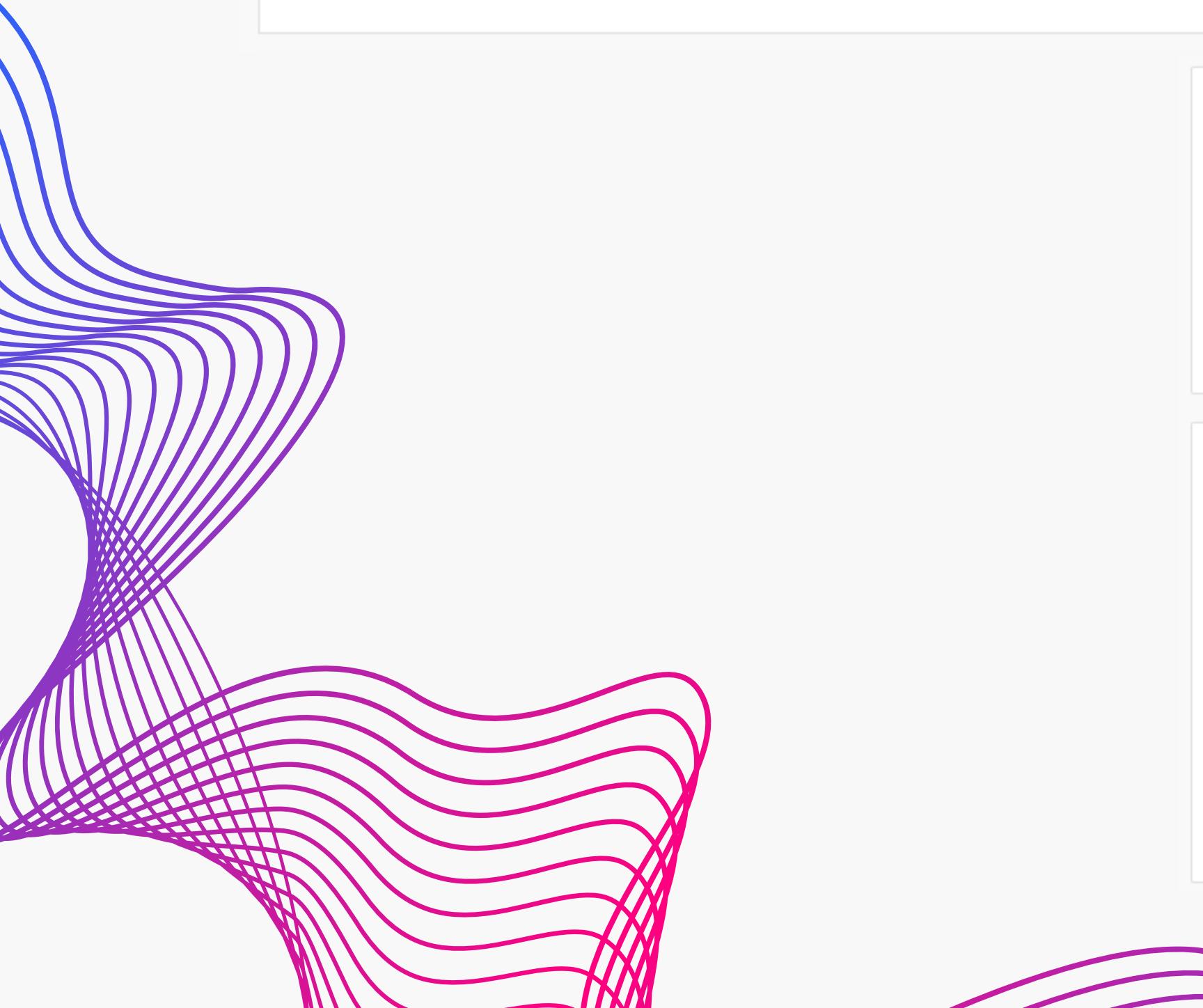
GitLab

La section "package" du script définit l'étape "docker-build". Avant de procéder à la construction de l'image Docker, certaines commandes de préparation sont exécutées. Cela comprend la connexion à Docker Hub, l'installation des outils curl et jq, ainsi que l'installation de trivy, un outil de sécurité pour les conteneurs. Ensuite, le script utilise Docker pour récupérer l'image de base "node:16". **En utilisant le cache de l'image de base**, il construit une nouvelle image nommée "kouss/commande:1.0.0". Ensuite, **une analyse de sécurité est effectuée sur l'image à l'aide de trivy**. Enfin, l'image construite est poussée vers Docker Hub.

L'étape docker-deploy est définie dans la section deploy. Le script exécute une commande Docker pour déployer l'image kouss/commande:1.0.0 dans un conteneur nommé client, en le reliant au port 8000 sur la machine hôte.

```
35 docker-build:
36   stage: package
37   before_script:
38     - docker login -u kouss -p $DOCKER_HUB_PASSWORD
39     - apk add --no-cache curl jq
40     - curl -sSfL https://raw.githubusercontent.com/aquasecurity/trivy/main/contrib/install.sh | sh -s -- -b /usr/local/bin
41   script:
42     - docker pull node:16 # Tirer l'image de base avant de construire pour utiliser le cache
43     - docker build --cache-from=node:16 -t kouss/commande:1.0.0 . # Utiliser le cache de l'image
44     - trivy image kouss/commande:1.0.0
45     - docker push kouss/commande:1.0.0
46
47 docker-deploy:
48   stage : deploy
49   script :
50     - docker run -d -p 8000:8000 --name commande kouss/commande:1.0.0
51
52
```

L'image construite "kouss/commande" a été bien poussée vers Docker Hub :



A screenshot of a Docker Hub repository page for the image "kouss / commande".

Repository Summary:

- Name: kouss / commande
- Type: Image
- Last pushed: 3 minutes ago
- Status: Inactive
- Stars: 0
- Downloads: 1
- Public

Description:
This repository does not have a description.

Last pushed: 3 minutes ago

Tags:
This repository contains 1 tag(s).

Tag	OS	Type	Pulled	Pushed
latest	Ubuntu	Image	3 minutes ago	3 minutes ago

[See all](#) [Go to Advanced Image Management](#)



kubernetes

On crée le Kubernetes manifest (fichier YAML) nécessaire pour déployer le microservice commande avec un déploiement (Deployment) et un service (Service) dans un cluster Kubernetes:

```
34  ---  
35  apiVersion: v1  
36  kind: Service  
37  metadata:  
38    name: commande-backend-service  
39  spec:  
40    ports:  
41      - port: 8000  
42    selector:  
43      app: commande-backend  
44
```

En spécifiant cette configuration, le conteneur du déploiement pourra accéder à la valeur du **secret mongodb-credentials** et l'utiliser comme valeur de la variable d'environnement MONGO_URI.

On spécifie **les limites et les demandes de ressources** pour le conteneur.

```
k8s > ! commande.yaml > {} spec > {} template > {} spec > [ ] containers > {} 0 > {} resources >  
io.k8s.core.v1.Service(v1@service.json) | io.k8s.api.apps.v1.Deployment(v1@deployment.json)  
1  apiVersion: apps/v1  
2  kind: Deployment  
3  metadata:  
4    name: commande-backend  
5  spec:  
6    replicas: 2  
7    selector:  
8      matchLabels:  
9        app: commande-backend  
10   template:  
11     metadata:  
12       labels:  
13         app: commande-backend  
14   spec:  
15     containers:  
16       - name: commande-backend  
17         image: kouss/commande:1.0.0  
18         ports:  
19           - containerPort: 8000  
20         env:  
21           - name: MONGO_URI  
22             valueFrom:  
23               secretKeyRef:  
24                 name: mongodb-credentials  
25                 key: mongoURI  
26         resources:  
27           limits:  
28             cpu: 500m  
29             memory: 512Mi  
30           requests:  
31             cpu: 200m
```



kubernetes

En utilisant ce fichier YAML avec la commande `kubectl apply -f commande.yaml`, le déploiement avec **deux répliques** du microservice commande sera créé, ainsi que le service pour acheminer le trafic vers ces pods:

```
PS C:\Users\HP\Desktop\FurniShop\k8s> kubectl apply -f commande.yaml -n furnishop
deployment.apps/commande-backend created
service/commande-backend-service created
```

```
PS C:\Users\HP\Desktop\FurniShop\k8s> kubectl get all -n furnishop
● NAME                                         READY   STATUS    RESTARTS   AGE
  pod/commande-backend-847f84cf56-lbwvdv      1/1     Running   0          104s
  pod/commande-backend-847f84cf56-lwtkx       1/1     Running   0          104s
  pod/frontend-deployment-6c77fd6dcc-x727s   1/1     Running   2 (107s ago) 5m51s
  pod/frontend-deployment-6c77fd6dcc-zsskl   1/1     Running   2 (103s ago) 5m51s
  pod/produit-backend-5f55947798-56hmg       1/1     Running   0          13m
  pod/produit-backend-5f55947798-hkxf4        1/1     Running   0          13m

NAME                           TYPE      CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
service/commande-backend-service ClusterIP  10.109.254.104 <none>        8000/TCP  104s
service/frontend-service         ClusterIP  10.96.242.187 <none>        3000/TCP  5m51s
service/produit-backend-service ClusterIP  10.108.3.32  <none>        5000/TCP  13m

NAME                                         READY   UP-TO-DATE  AVAILABLE   AGE
deployment.apps/commande-backend            2/2     2           2           104s
deployment.apps/frontend-deployment        2/2     2           2           5m51s
deployment.apps/produit-backend             2/2     2           2           13m
```

Microservice paiement

Node.js/Express et MongoDB, Dockerfile, Gitlab CI/CD, Kubernetes

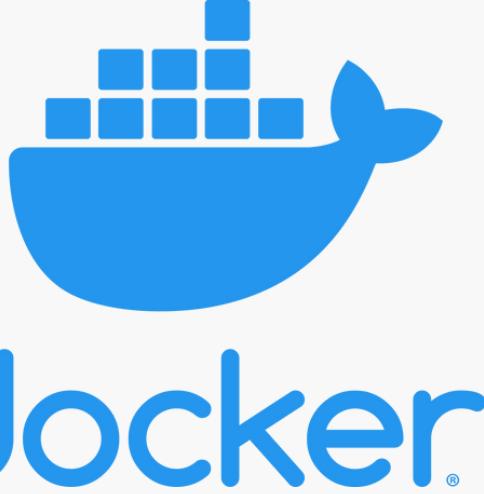
On crée les fichiers servers.js , paiement.js et paiements.js similaires aux fichiers créés pour les microservices produit et commande:

```
JS paiements.js M X
paiement > routes > api > JS paiements.js > ...
1  const express = require('express');
2  const router = express.Router();
3  const Paiement = require('../..../models/paiement');
4
5  router.get('/', (req,res)=>{
6      Paiement.find()
7          .then( paiements=>res.json(paiements) )
8  });
9
10 router.post('/', (req,res)=>{
11     const newPaiement = new Paiement({
12         commandes:req.body.commandes,
13         montant : req.body.montant,
14         numeroCarte : req.body.numeroCarte,
15     });
16     newPaiement.save()
17         .then( paiement => res.json(paiement) )
18 });
19
20 module.exports = router ;
```

```
JS server.js ●
paiement > JS server.js > [0] db
1  const express = require('express');
2  const mongoose = require('mongoose');
3  const commandes = require('./routes/api/commandes');
4  const app = express();
5  app.use(express.json());
6  var cors = require('cors')
7
8  app.use(cors())
9  const db = require('./config/keys').mongoURI
10
11 mongoose.connect(db)
12     .then( ()=>console.log('MongoDB connected ...') )
13     .catch( err=>console.log(err));
14
15 app.use('/api/paiements' , commandes);
16 const port = process.env.PORT || 8080 ;
17 app.listen(port , ()=>console.log(`Server started on port ${port}`)) ;
```



```
JS paiement.js M, M ●
paiement > models > JS paiement.js > ...
1  const mongoose = require('mongoose');
2  const Schema = mongoose.Schema ;
3  const PaiementSchema = new Schema({
4      commandes: [
5          {
6              type: Schema.Types.ObjectId,
7              ref: 'Commande',
8              required: true
9          }
10     ],
11     montant:{
12         type:Number,
13         required:true
14     },
15     numeroCarte:{
16         type:Number,
17         default:2113448900709
18     },
19 });
20 module.exports = Paiement = mongoose.model('paiement', PaiementSchema);
```



Ensuite on crée le dockerfile, le fichier .dockerignore et on construit l'image Docker à partir du Dockerfile présent dans le répertoire courant et on lui attribue le nom "kouss/paiement" avec la balise "1.0.0"

```
paiement > 🐳 dockerfile > ...
1  FROM node:16-alpine
2  USER node
3  ENV NODE_ENV=production
4  WORKDIR /paiement
5  COPY --chown=node:node package.json package-lock.json ../
6  RUN npm ci --only=production
7  COPY --chown=node:node . .
8  EXPOSE 8080
9  CMD ["node", "server.js"]
10
```

```
paiement > 🐳 .dockerignore
1  node_modules
```

On exécute un conteneur à partir de l'image "kouss/paiement:1.0.0"

```
PS C:\Users\HP\Desktop\FurniShop\paiement> docker build -t kouss/paiement:1.0.0 .
[+] Building 22.5s (10/10) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 271B
=> [internal] load .dockerignore
=> => transferring context: 52B
=> [internal] load metadata for docker.io/library/node:16-alpine
=> CACHED [1/5] FROM docker.io/library/node:16-alpine@sha256:8d18e32fa398763c07407e9d407c64e6a3c2a18e9fa97362cfde669a6b6161ef
=> => resolve docker.io/library/node:16-alpine@sha256:8d18e32fa398763c07407e9d407c64e6a3c2a18e9fa97362cfde669a6b6161ef
=> [internal] load build context
=> => transferring context: 111.61kB
=> [2/5] WORKDIR /paiement
=> [3/5] COPY --chown=node:node package.json package-lock.json ../
=> [4/5] RUN npm ci --only=production
=> [5/5] COPY --chown=node:node . .
=> exporting to image
=> => exporting layers
=> => writing image sha256:ddc1f586f409fae34905be4e745c1617d34d0fc1476594a5131cf89b45064a08
=> => naming to docker.io/kouss/paiement:1.0.0
```

```
PS C:\Users\HP\Desktop\FurniShop> docker run -p 8080:8080 kouss/paiement:1.0.0
Server started on port 8080
MongoDB connected ...
```



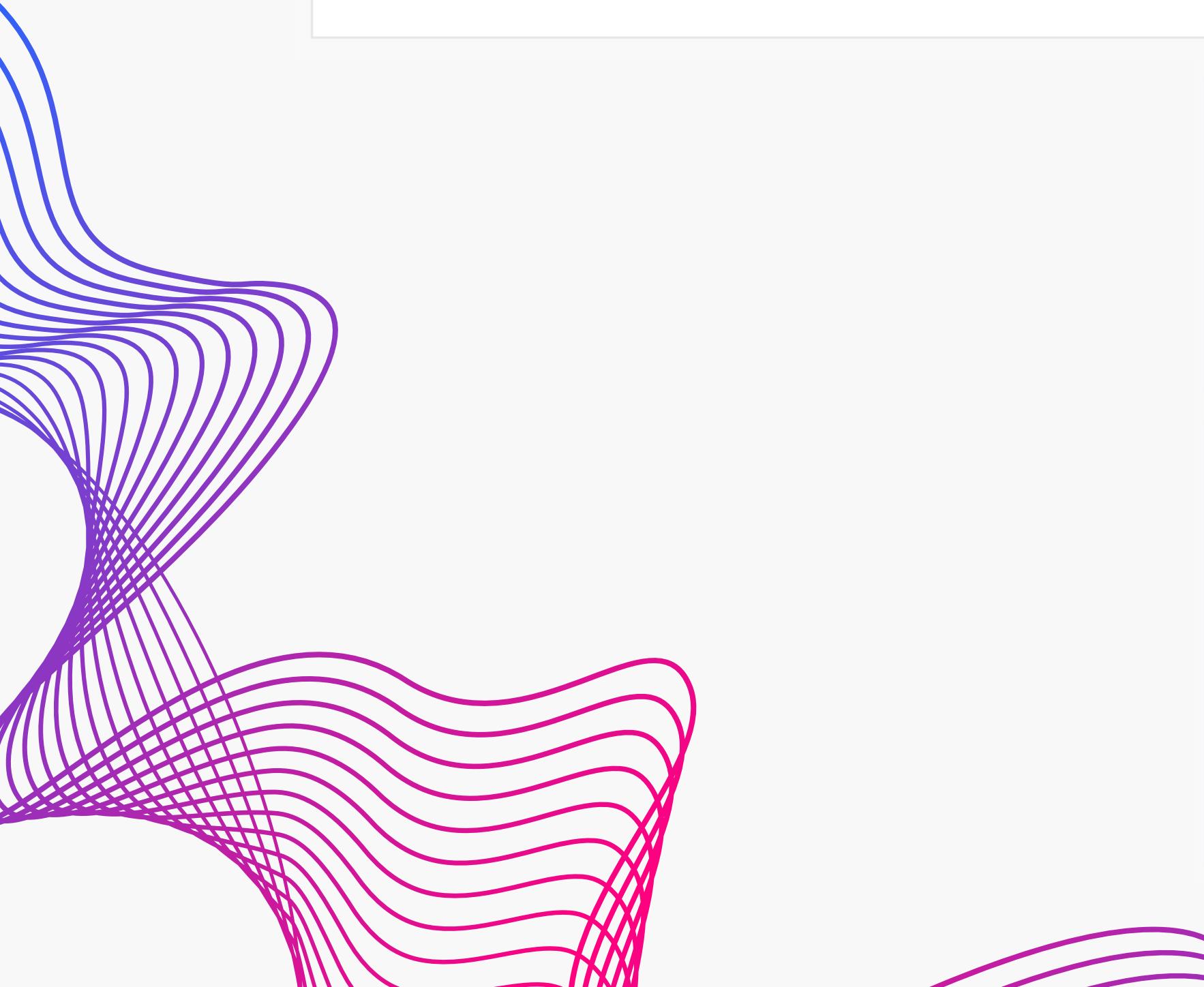
GitLab

Maintenant on crée le fichier `gitlab-ci.yml` avec quatre stages similaire à celui des microservices précédents:

📌 `.gitlab-ci.yml` 1.23 KiB Open in

```
1 image: docker:latest
2 services:
3   - docker:dind
4
5 stages:
6   - securityScan
7   - build
8   - package
9   - deploy
10
11 include:
12   - template: Security/SAST.gitlab-ci.yml
13
14 cache:
15   paths:
16     - node_modules/
17
18 sast:
19   stage : securityScan
20   artifacts:
21     reports:
22       sast: gl-sast-report.json
23   allow_failure: false
24
25 build:
26   image: node:16
27   stage: build
28   cache:
29     key: "$CI_COMMIT_REF_SLUG" # Utilisez une clé unique pour chaque branche
30     paths:
31       - node_modules/
32   script:
33     - npm ci           # Utiliser le cache pour les dépendances
34
35 docker-build:
36   stage: package
37   before_script:
38     - docker login -u kouss -p $DOCKER_HUB_PASSWORD
39     - apk add --no-cache curl jq
40     - curl -sSfL https://raw.githubusercontent.com/aquasecurity/trivy/main/contrib/install.sh | sh -s -- -b /usr/local/bin
41   script:
42     - docker pull node:16 # Tirer l'image de base avant de construire pour utiliser le cache
43     - docker build --cache-from=node:16 -t kouss/paiement:1.0.0 . # Utiliser le cache de l'image
44     - trivy image kouss/paiement:1.0.0
45     - docker push kouss/paiement:1.0.0
46
47 docker-deploy:
48   stage : deploy
49   script :
50     - docker run -d -p 8080:8080 --name paiement kouss/paiement:1.0.0
```

L'image construite "kouss/paiement" a été bien poussée vers Docker Hub :



The screenshot shows a Docker Hub repository page for the image "kouss / paiement".

Repository Summary:

- Name: kouss / paiement
- Contains: Image
- Last pushed: 3 minutes ago
- Status: Inactive
- Stars: 0
- Downloads: 1
- Type: Public

Description:

This repository does not have a description.

Last pushed: 3 minutes ago

Tags:

This repository contains 1 tag(s).

Tag	OS	Type	Pulled	Pushed
latest	Ubuntu	Image	2 minutes ago	3 minutes ago

[See all](#) [Go to Advanced Image Management](#)



kubernetes

Enfin on crée le Kubernetes manifest (fichier YAML) nécessaire pour déployer le microservice paiement avec un déploiement (Deployment) et un service (Service) dans un cluster Kubernetes:

```
34  ---
35  apiVersion: v1
36  kind: Service
37  metadata:
38    name: paiement-backend-service
39  spec:
40    ports:
41      - port: 8080
42    selector:
43      app: paiement-backend
```

```
k8s > ! paiement.yaml > {} spec > {} template > {} spec > [ ] containers > {} 0 > [ ] env > {} 0 > []
          io.k8s.api.core.v1.Service (v1@service.json) | io.k8s.api.apps.v1.Deployment (v1@deployment.json)
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: paiement-backend
5  spec:
6    replicas: 2
7    selector:
8      matchLabels:
9        app: paiement-backend
10   template:
11     metadata:
12       labels:
13         app: paiement-backend
14     spec:
15       containers:
16         - name: paiement-backend
17           image: kouss/paiement:1.0.0
18           ports:
19             - containerPort: 8080
20           env:
21             - name: MONGO_URI
22               valueFrom:
23                 secretKeyRef:
24                   name: mongodb-credentials
25                   key: mongoURI
26       resources:
27         limits:
28           cpu: 500m
29           memory: 512Mi
30         requests:
31           cpu: 200m
32           memory: 256Mi
```



kubernetes

En utilisant ce fichier YAML avec la commande kubectl apply -f paiement.yaml, le déploiement avec **deux répliques** du microservice paiement sera créé, ainsi que le service pour acheminer le trafic vers ces pods:

```
● PS C:\Users\HP\Desktop\FurniShop\k8s> kubectl apply -f paiement.yaml -n furnishop
deployment.apps/paiement-backend created
service/paiement-backend-service created
```

```
PS C:\Users\HP\Desktop\FurniShop\k8s> kubectl get all -n furnishop
● NAME                                     READY   STATUS    RESTARTS   AGE
pod/commande-backend-847f84cf56-1bwvdv   1/1     Running   0          7m20s
pod/commande-backend-847f84cf56-1wtkx    1/1     Running   0          7m20s
pod/frontend-deployment-6c77fd6dcc-x727s  1/1     Running   4 (2m42s ago) 11m
pod/frontend-deployment-6c77fd6dcc-zsskl  1/1     Running   4 (2m45s ago) 11m
pod/paiement-backend-548757b8b9-pxmw2   1/1     Running   0          2m31s
pod/paiement-backend-548757b8b9-sn1gb   1/1     Running   0          2m31s
pod/produit-backend-5f55947798-56hmg   1/1     Running   0          19m
pod/produit-backend-5f55947798-hkxf4   1/1     Running   0          19m

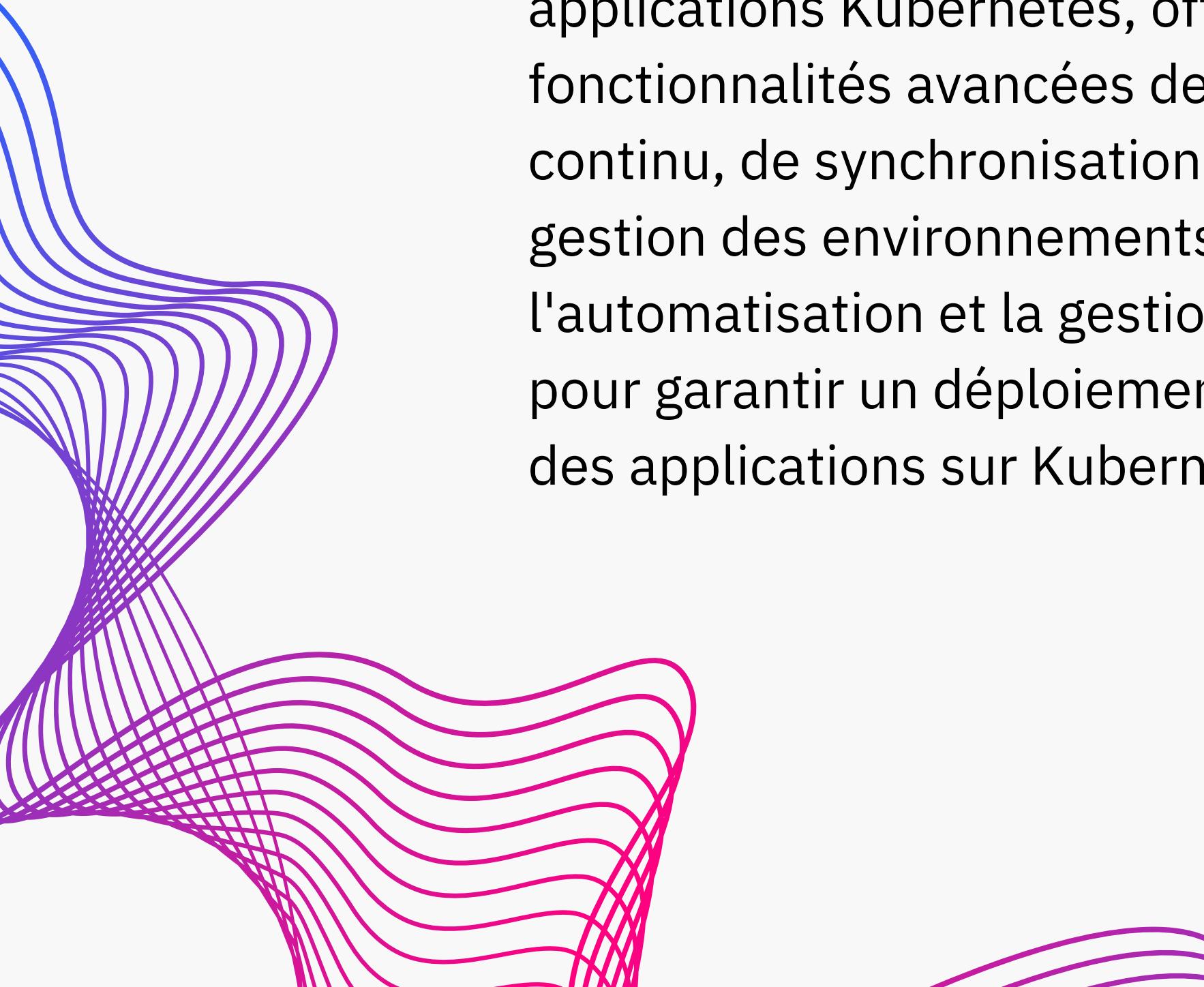
NAME                           TYPE      CLUSTER-IP        EXTERNAL-IP   PORT(S)   AGE
service/commande-backend-service ClusterIP 10.109.254.104 <none>       8000/TCP  7m20s
service/frontend-service          ClusterIP 10.96.242.187 <none>       3000/TCP  11m
service/paiement-backend-service ClusterIP 10.105.131.13 <none>       8080/TCP  2m31s
service/produit-backend-service  ClusterIP 10.108.3.32  <none>       5000/TCP  19m

NAME                           READY   UP-TO-DATE  AVAILABLE   AGE
deployment.apps/commande-backend 2/2     2           2           7m20s
deployment.apps/frontend-deployment 2/2     2           2           11m
deployment.apps/paiement-backend  2/2     2           2           2m31s
deployment.apps/produit-backend  2/2     2           2           19m

NAME                           DESIRED  CURRENT  READY   AGE
replicaset.apps/commande-backend-847f84cf56  2        2        2       7m20s
replicaset.apps/frontend-deployment-6c77fd6dcc 2        2        2       11m
replicaset.apps/paiement-backend-548757b8b9   2        2        2       2m31s
replicaset.apps/produit-backend-5f55947798  2        2        2       19m
```

Gestion et déploiement des applications Kubernetes de manière continue

Kubernetes, ArgoCD



ArgoCD est un outil puissant pour la gestion des applications Kubernetes, offrant des fonctionnalités avancées de déploiement continu, de synchronisation en continu et de gestion des environnements. Il facilite l'automatisation et la gestion des flux de travail pour garantir un déploiement cohérent et fiable des applications sur Kubernetes.



Pour déployer ArgoCD sur Kubernetes, on crée d'abord un namespace pour ArgoCD :

```
● PS C:\Users\HP\Desktop\FurniShop> kubectl create namespace argocd  
namespace/argocd created
```

En utilisant cette commande, la configuration spécifiée dans le fichier YAML sera appliquée dans le namespace "argocd", déployant ainsi ArgoCD sans authentification sur votre cluster Kubernetes:

```
● PS C:\Users\HP\Desktop\FurniShop> kubectl apply -n argocd -f https://raw.githubusercontent.com/Codefresh-contrib/gitops-certification-examples/main/argocd-noauth/install.yaml  
customresourcedefinition.apiextensions.k8s.io/applications.argoproj.io created  
customresourcedefinition.apiextensions.k8s.io/appprojects.argoproj.io created  
serviceaccount/argocd-application-controller created  
serviceaccount/argocd-dex-server created  
○ serviceaccount/argocd-redis created  
○ serviceaccount/argocd-server created  
● role.rbac.authorization.k8s.io/argocd-application-controller created  
role.rbac.authorization.k8s.io/argocd-dex-server created  
role.rbac.authorization.k8s.io/argocd-server created  
clusterrole.rbac.authorization.k8s.io/argocd-application-controller created  
clusterrole.rbac.authorization.k8s.io/argocd-server created  
rolebinding.rbac.authorization.k8s.io/argocd-application-controller created  
rolebinding.rbac.authorization.k8s.io/argocd-dex-server created  
rolebinding.rbac.authorization.k8s.io/argocd-redis created  
rolebinding.rbac.authorization.k8s.io/argocd-server created  
clusterrolebinding.rbac.authorization.k8s.io/argocd-application-controller created
```

On vérifie que tous les pods ArgoCD sont en cours d'exécution :

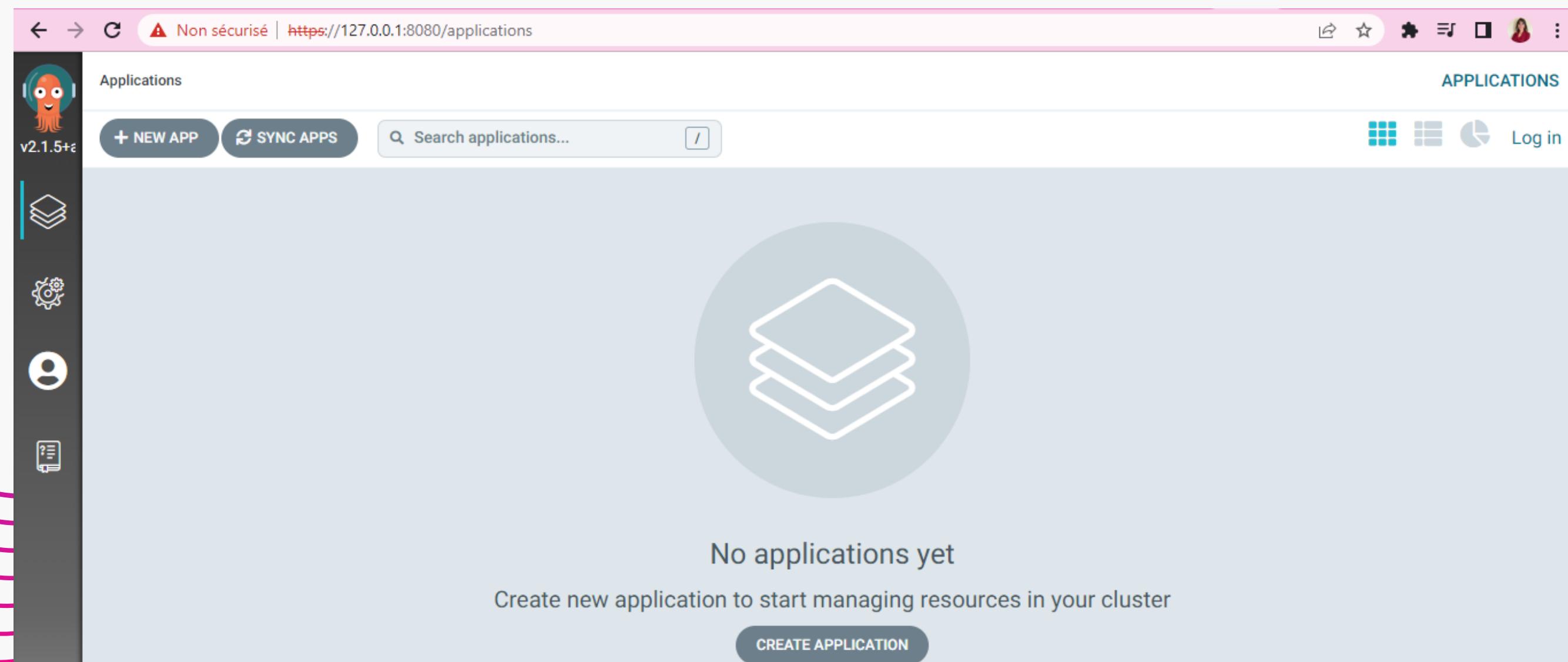
PS C:\Users\HP\Desktop\FurniShop> kubectl get pods -n argocd					
● NAME	READY	STATUS	RESTARTS	AGE	
argocd-application-controller-0	1/1	Running	0	14m	
argocd-dex-server-7f99d787b9-c9n98	1/1	Running	0	14m	
argocd-redis-7bf6866966-txc6w	1/1	Running	0	14m	
argocd-repo-server-75f5cbdf58-kzrn1	1/1	Running	0	14m	
argocd-server-5dfc8bfdb7-r87zq	1/1	Running	0	14m	

On vérifie l'adresse IP externe attribuée au service ArgoCD :

PS C:\Users\HP\Desktop\FurniShop> kubectl get svc -n argocd					
● NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
argocd-dex-server	ClusterIP	10.111.155.102	<none>	5556/TCP,5557/TCP,5558/TCP	16m
argocd-metrics	ClusterIP	10.101.243.5	<none>	8082/TCP	16m
argocd-redis	ClusterIP	10.110.6.83	<none>	6379/TCP	16m
argocd-repo-server	ClusterIP	10.101.174.104	<none>	8081/TCP,8084/TCP	16m
argocd-server	ClusterIP	10.107.87.89	<none>	80/TCP,443/TCP	16m
argocd-server-metrics	ClusterIP	10.101.73.118	<none>	8083/TCP	16m

On accède à l'interface utilisateur d'ArgoCD dans le navigateur en utilisant l'adresse IP externe obtenue à l'étape précédente:

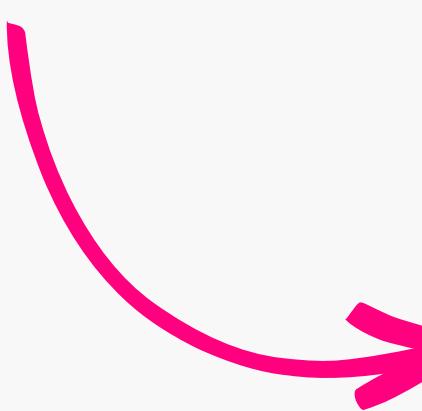
```
PS C:\Users\HP\Desktop\FurniShop> kubectl port-forward -n argocd svc/argocd-server 8080:443
Forwarding from 127.0.0.1:8080 -> 8080
Forwarding from [::1]:8080 -> 8080
```



On se connecte à l'interface utilisateur d'ArgoCD en utilisant le nom d'utilisateur par défaut admin et on obtient le mot de passe en exécutant la commande suivante :

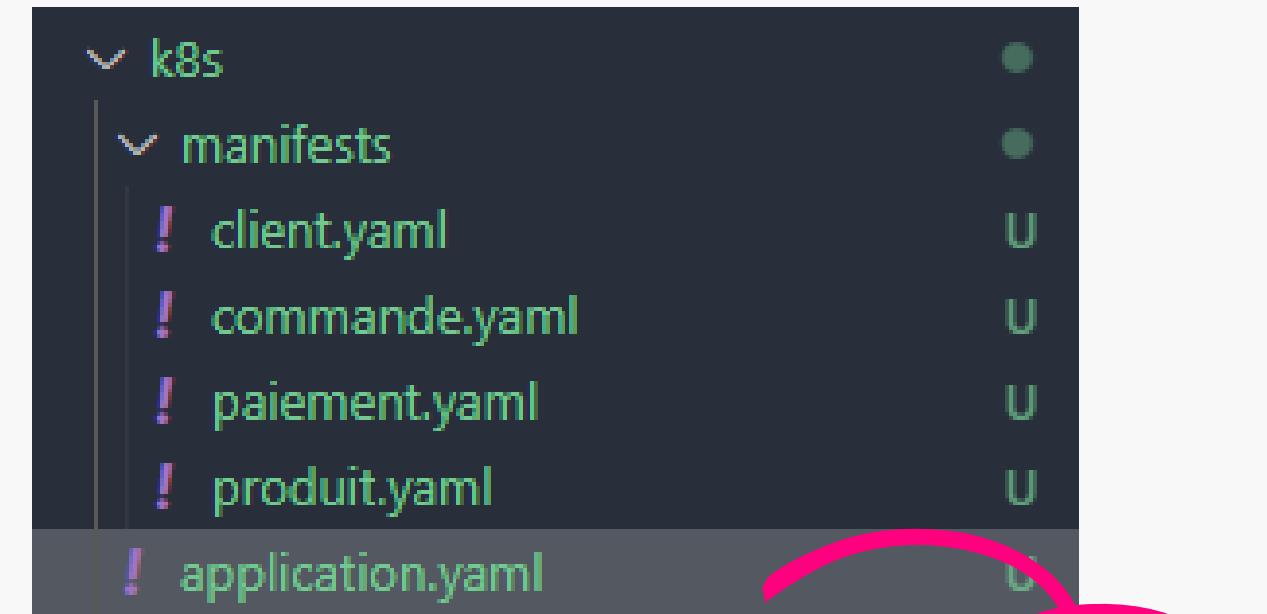
```
PS C:\Users\HP\Desktop\FurniShop> kubectl get secret argocd-initial-admin-secret -n argocd -o yaml
● apiVersion: v1
  data:
    password: QjFFcDFyTC04ckg2TGd50Q==
  kind: Secret
  metadata:
    creationTimestamp: "2023-06-04T17:01:04Z"
    name: argocd-initial-admin-secret
    namespace: argocd
    resourceVersion: "87603"
    uid: 1b7f5f0b-6504-4a77-b22d-16e8e5a0b917
  type: Opaque
```

```
PS C:\Users\HP\Desktop\FurniShop> echo QjFFcDFyTC04ckg2TGd50Q== | base64 -d
B1Ep1rL-8rH6Lgy9
```



The image shows a screenshot of a web browser displaying the Argo sign-in page. The page has a light gray background with a white header containing the word "argo" in orange. Below the header, there are two input fields: one for "Username" containing "admin" and another for "Password" containing "B1Ep1rL-8rH6Lgy9". A teal "SIGN IN" button is located at the bottom right of the form. The overall design is clean and modern.

On crée le fichier yaml "application.yaml" qui définit une application ArgoCD nommée "argo-app" et déploie les fichiers de configuration situés dans le référentiel Git "<https://gitlab.com/khadijaoussakel/kubernetes.git>" vers le serveur Kubernetes par défaut, dans le namespace "furnishop". La politique de synchronisation est configurée pour inclure la création du namespace cible, ainsi que la correction automatique de l'état de l'application et la suppression des ressources obsolètes.



```
k8s > ! application.yaml > {} spec > {} syncPolicy > {} automated
1   apiVersion: argoproj.io/v1alpha1
2   kind: Application
3   metadata:
4     name: argo-app
5     namespace: argocd
6   spec:
7     project: default
8
9   source:
10    repoURL: https://gitlab.com/khadijaoussakel/kubernetes.git
11    targetRevision: HEAD
12    path: manifests
13
14   destination:
15    server: https://kubernetes.default.svc
16    namespace: furnishop
17
18   syncPolicy:
19     syncOptions:
20       - CreateNamespace=true
21
22   automated:
23     selfHeal: true
24     prune: true
```

On pousse les modifications vers la branche distante du référentiel GitLab avec le nom "kubernetes". Ce répertoire GitLab contient les fichiers YAML de Kubernetes nécessaires pour déployer l'ensemble de l'application dans un cluster Kubernetes en local et le fichier application.yaml configuré précédemment:

```
PS C:\Users\HP\Desktop\FurniShop\k8s> git add .
PS C:\Users\HP\Desktop\FurniShop\k8s> git commit -m "add argocd configuration"
[main 6c714e8] add argocd configuration
  5 files changed, 24 insertions(+)
  create mode 100644 application.yaml
  rename client.yaml => manifests/client.yaml (100%)
  rename commande.yaml => manifests/commande.yaml (100%)
  rename paiement.yaml => manifests/paiement.yaml (100%)
  rename produit.yaml => manifests/produit.yaml (100%)
PS C:\Users\HP\Desktop\FurniShop\k8s> git push -uf origin main
Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Delta compression using up to 8 threads.
Compressing objects: 100% (9/9), done.
Writing objects: 100% (9/9), 1.60 KiB
Total 9 (delta 2), reused 0 (delta 0)
To https://gitlab.com/khadijaoussakel/FurniShop
 + fd0c68e...6c714e8 main -> main (forced update)
branch 'main' set up to track 'origin/main'.
```

The screenshot shows a GitLab repository interface for a project named "kubernetes". The repository has a Project ID of 46563477. It contains 12 commits, 1 branch, 0 tags, and 43 KB of project storage. The most recent commit, "add argocd configuration" by khadija oussakel, was authored 1 minute ago. The commit hash is 6c714e8d. The repository has 0 stars and 0 forks. Below the commit details, there are buttons for CI/CD configuration, Add README, Add LICENSE, Add CHANGELOG, Add CONTRIBUTING, Add Kubernetes cluster, and Add Wiki. There is also a "Configure Integrations" button. A table below lists the files in the repository, showing their names, last commit, and last update time. The files listed are "manifests" and "application.yaml", both of which were last updated 1 minute ago.

Name	Last commit	Last update
manifests	add argocd configuration	1 minute ago
application.yaml	add argocd configuration	1 minute ago

On tape la commande "kubectl apply -f application.yaml" pour appliquer la configuration de l'application ArgoCD:

```
PS C:\Users\HP\Desktop\FurniShop\k8s> kubectl apply -f application.yaml
application.argoproj.io/argo-app created
```

Sur l'interface ArgoCD, on peut voir l'application nouvellement créée, ainsi que son état de synchronisation.

← → C Non sécurisé | https://127.0.0.1:8080/applications?proj=&sync=&health=&namespace=&cluster=&labels=

v2.1.5+e

Applications

+ NEW APP SYNC APPS Search applications... /

FILTERS

SYNC STATUS

- Unknown 0
- Synced 1
- OutOfSync 0

HEALTH STATUS

- Unknown 0
- Progressing 0
- Suspended 0
- Healthy 1
- Degraded 0

argo-app

Project: default

Labels:

Status: Healthy Synced

Repo...: https://gitlab.com/khadijaoussake...

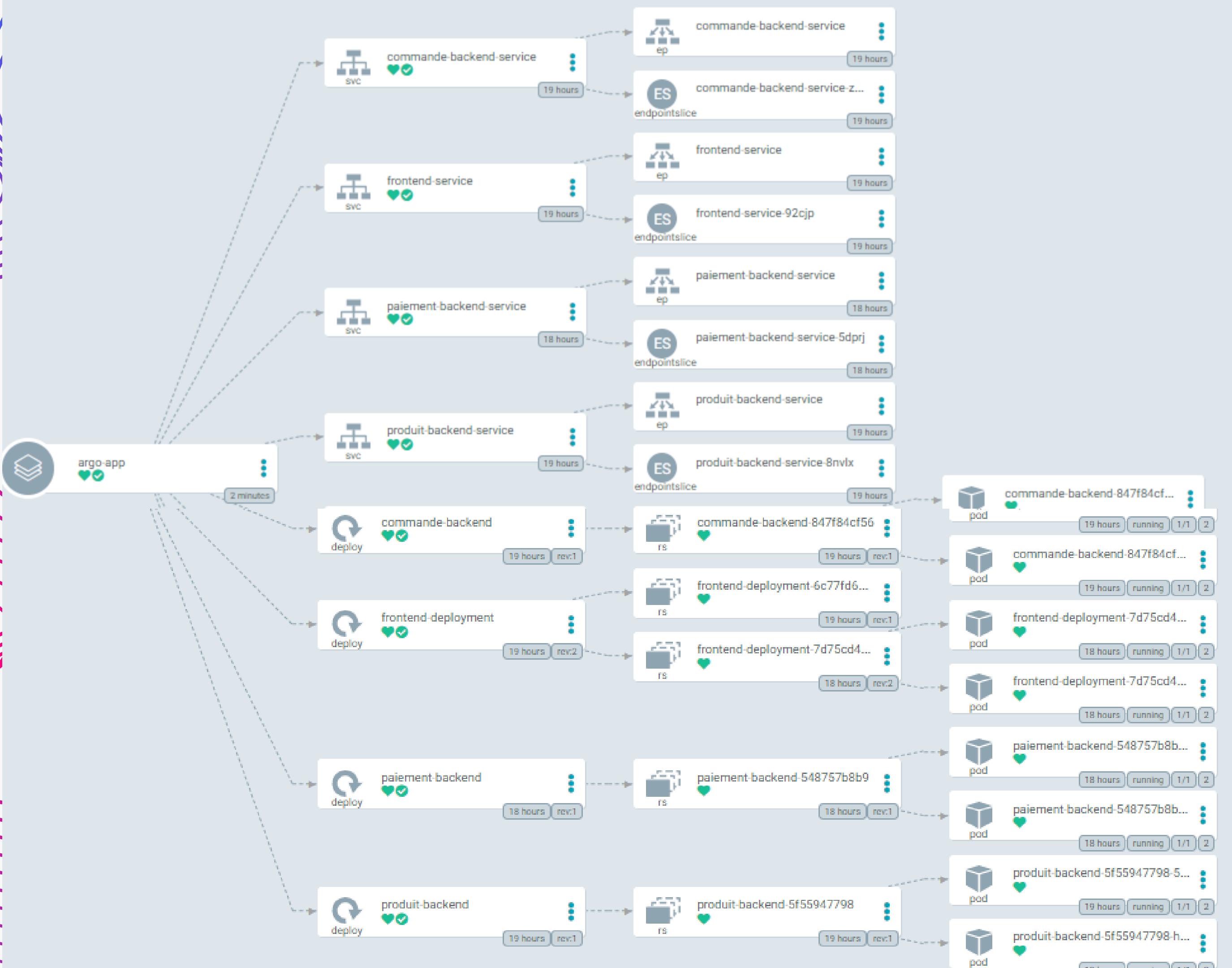
Target ...: HEAD

Path: manifests

Destin...: in-cluster

Name...: furnishop

SYNC C X

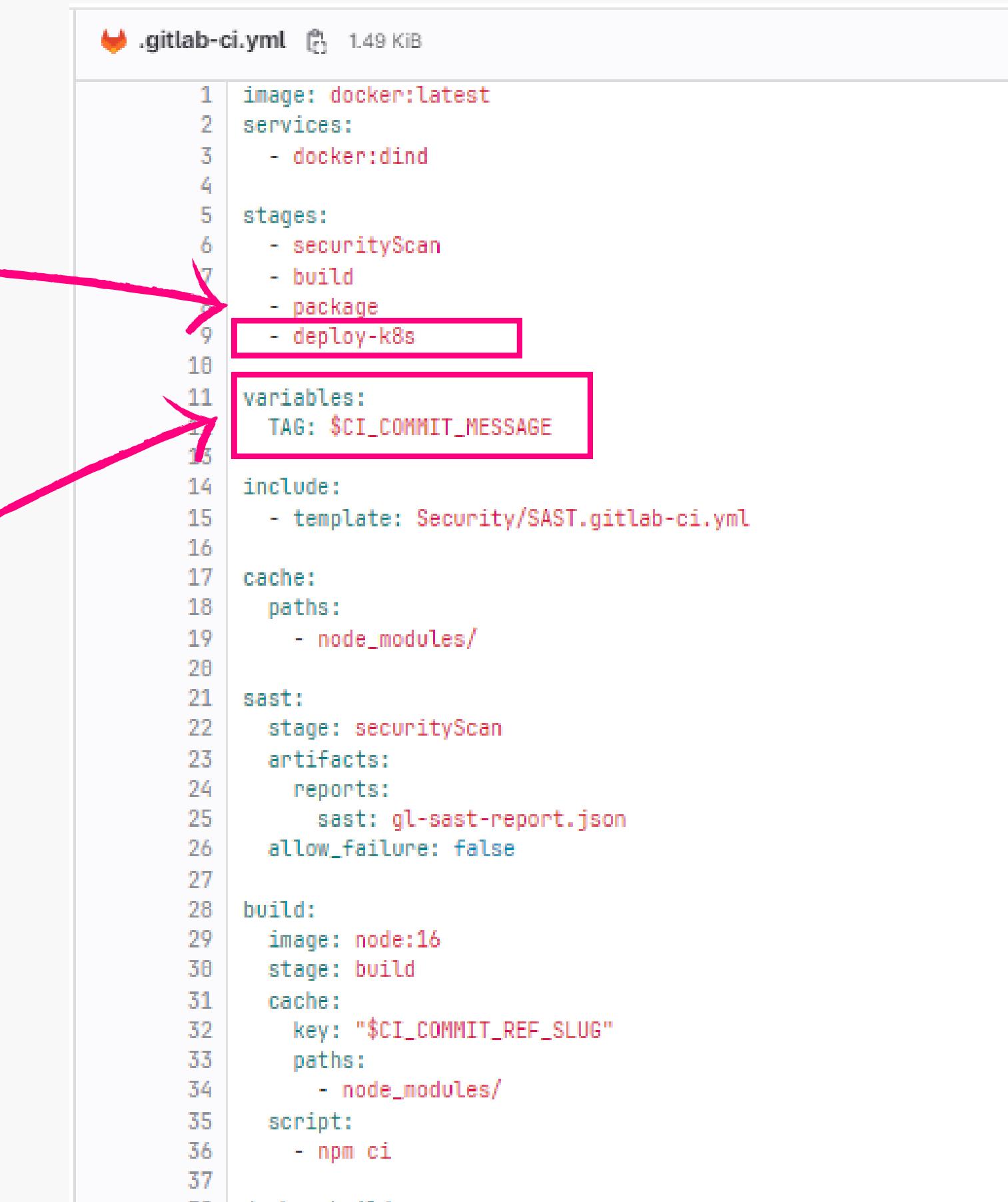


On crée une nouvelle variable d'environnement qui comportera le mot de passe de Git. Cela offre une couche de sécurité supplémentaire en évitant d'écrire le mot de passe directement dans les scripts.

Type	↑ Key	Value	Options	Environments	
Variable	DOCKER_HUB_PASSWORD 	***** 	Protected	All (default) 	
Variable	GIT_PASSWORD 	***** 	Protected	All (default) 	
Add variable Reveal values					

On a ajouté un autre stage "deploy-k8s" pour le déploiement sur Kubernetes avec GitOps.

La section "variables" définit une variable nommée "TAG" qui est utilisée pour stocker le message de validation actuel (CI_COMMIT_MESSAGE) qui sera utilisé comme balise lors de la construction de l'image Docker.



The image shows a screenshot of a .gitlab-ci.yml file. Two specific sections are highlighted with pink boxes and arrows pointing from the explanatory text above:

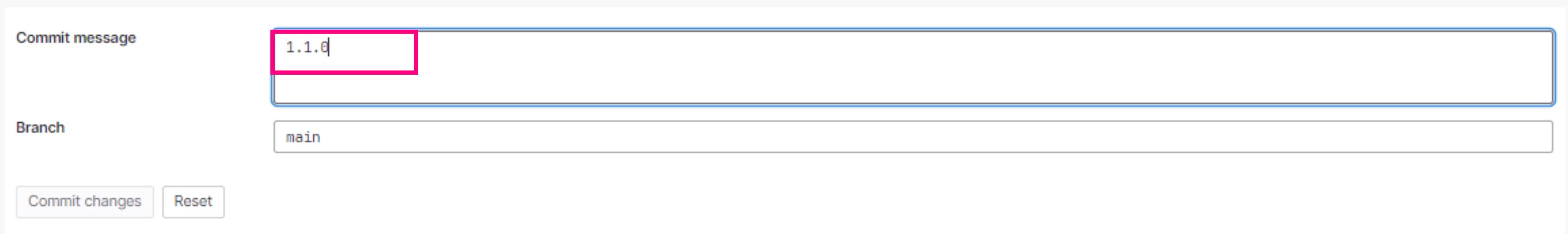
- A pink box highlights the "deploy-k8s" stage at line 9: `- deploy-k8s`
- A pink box highlights the "variables" section at line 11: `TAG: $CI_COMMIT_MESSAGE`

```
1 image: docker:latest
2 services:
3   - docker:dind
4
5 stages:
6   - securityScan
7   - build
8   - package
9   - deploy-k8s
10
11 variables:
12   TAG: $CI_COMMIT_MESSAGE
13
14 include:
15   - template: Security/SAST.gitlab-ci.yml
16
17 cache:
18   paths:
19     - node_modules/
20
21 sast:
22   stage: securityScan
23   artifacts:
24     reports:
25       sast: gl-sast-report.json
26   allow_failure: false
27
28 build:
29   image: node:16
30   stage: build
31   cache:
32     key: "$CI_COMMIT_REF_SLUG"
33     paths:
34       - node_modules/
35   script:
36     - npm ci
37
```

La section "gitops-k8s-deploy" définit une étape de pipeline pour le déploiement sur Kubernetes. Elle utilise l'image "bitnami/git:latest" pour exécuter les commandes Git. Le script clone le référentiel contenant des fichiers de configuration Kubernetes, modifie le fichier "produit.yaml" avec la balise spécifiée, effectue un commit et un push vers le référentiel.

```
38 docker-build:
39   stage: package
40   before_script:
41     - docker login -u kouss -p $DOCKER_HUB_PASSWORD
42     - apk add --no-cache curl jq
43     - curl -sSfL https://raw.githubusercontent.com/aquasecurity/trivy/main/contrib/install.sh | sh -s -- -b /usr/local/bin
44   script:
45     - docker pull node:16
46     - docker build --cache-from=node:16 -t kouss/produit:$TAG .
47     - trivy image kouss/produit:$TAG
48     - docker push kouss/produit:$TAG
49
50
51 gitops-k8s-deploy:
52   image: bitnami/git:latest
53   stage: deploy-k8s
54   before_script:
55     - git config --global user.email "kouix.maroc@gmail.com"
56     - git config --global user.name "khadijaoussakel"
57   script:
58     - git clone https://gitlab.com/khadijaoussakel/kubernetes.git
59     - cd kubernetes/manifests/
60     - sed -i "s/produit:1.0.0/produit:$TAG/g" produit.yaml
61     - cat produit.yaml
62     - git add produit.yaml
63     - git commit -m "microservice version $TAG"
64     - git remote set-url origin https://khadijaoussakel:$GIT_PASSWORD@gitlab.com/khadijaoussakel/kubernetes.git
65     - git push -uf origin main
```

On fait passer la balise qu'on souhaite utiliser lors de la construction de l'image Docker dans le Commit message :



La balise de l'image a bien été changée dans le fichier product.yaml

```
36 $ cat produit.yaml
37 apiVersion: apps/v1
38 kind: Deployment
39 metadata:
40   name: produit-backend
41 spec:
42   replicas: 2
43   selector:
44     matchLabels:
45       app: produit-backend
46   template:
47     metadata:
48       labels:
49         app: produit-backend
50   spec:
51     containers:
52       - name: produit-backend
53         image: kouss/produit:1.1.0
54       ports:
55         - containerPort: 5000
56       env:
57         - name: MONGO_URI
58         valueFrom:
59           secretKeyRef:
60             name: mongodb-
61             key: mongoURI
62       resources:
63         limits:
```

```
79 $ git add produit.yaml
80 $ git commit -m "microservice version $TAG"
81 [main a1edd1c] microservice version 1.1.0
82 1 file changed, 1 insertion(+), 1 deletion(-)
83 $ git remote set-url origin https://khadijaoussakel:$GIT_PASSWORD@gitlab.com/khadijaoussakel/kubernetes.git
84 $ git push -uf origin main
85 To https://gitlab.com/khadijaoussakel/kubernetes.git
86   eda520f..a1edd1c  main -> main
87 branch 'main' set up to track 'origin/main'.
88 Saving cache for successful job
89 Creating cache default-protected...
90 WARNING: node_modules/: no matching files. Ensure that the artifact path is relative to the working directory (/builds/khadijaoussakel/produit-microservice)
91 Archive is up to date!
92 Created cache
93 Cleaning up project directory and file based variables
94 Job succeeded
```

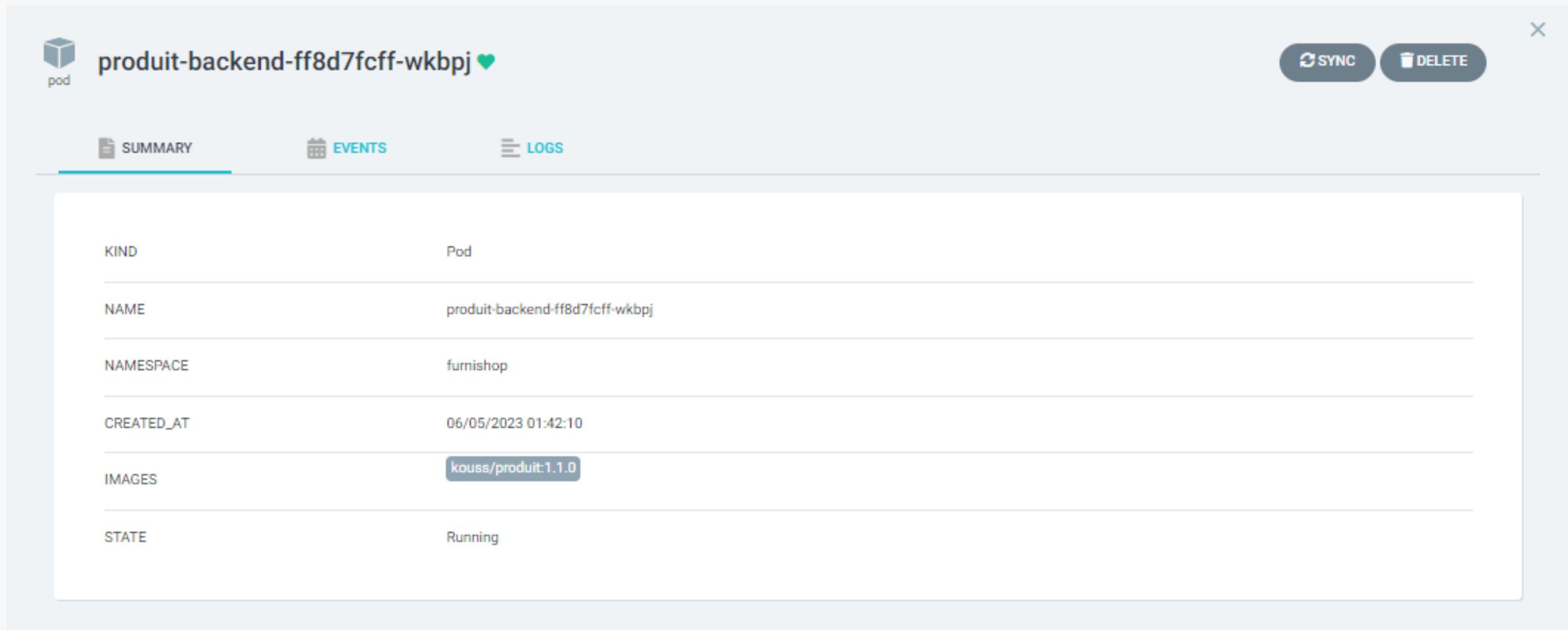
Finished: 4 minutes ago
Queued: 0 seconds
Timeout: 1h (from project) 
Runner: #12270845 (JLgUopmMV) 1-green.shared.runners-manager.gitlab.com/default

Commit 49f20270 
1.1.0

✓ Pipeline #889048608 for main 
deploy-k8s 
→ gitops-k8s-deploy

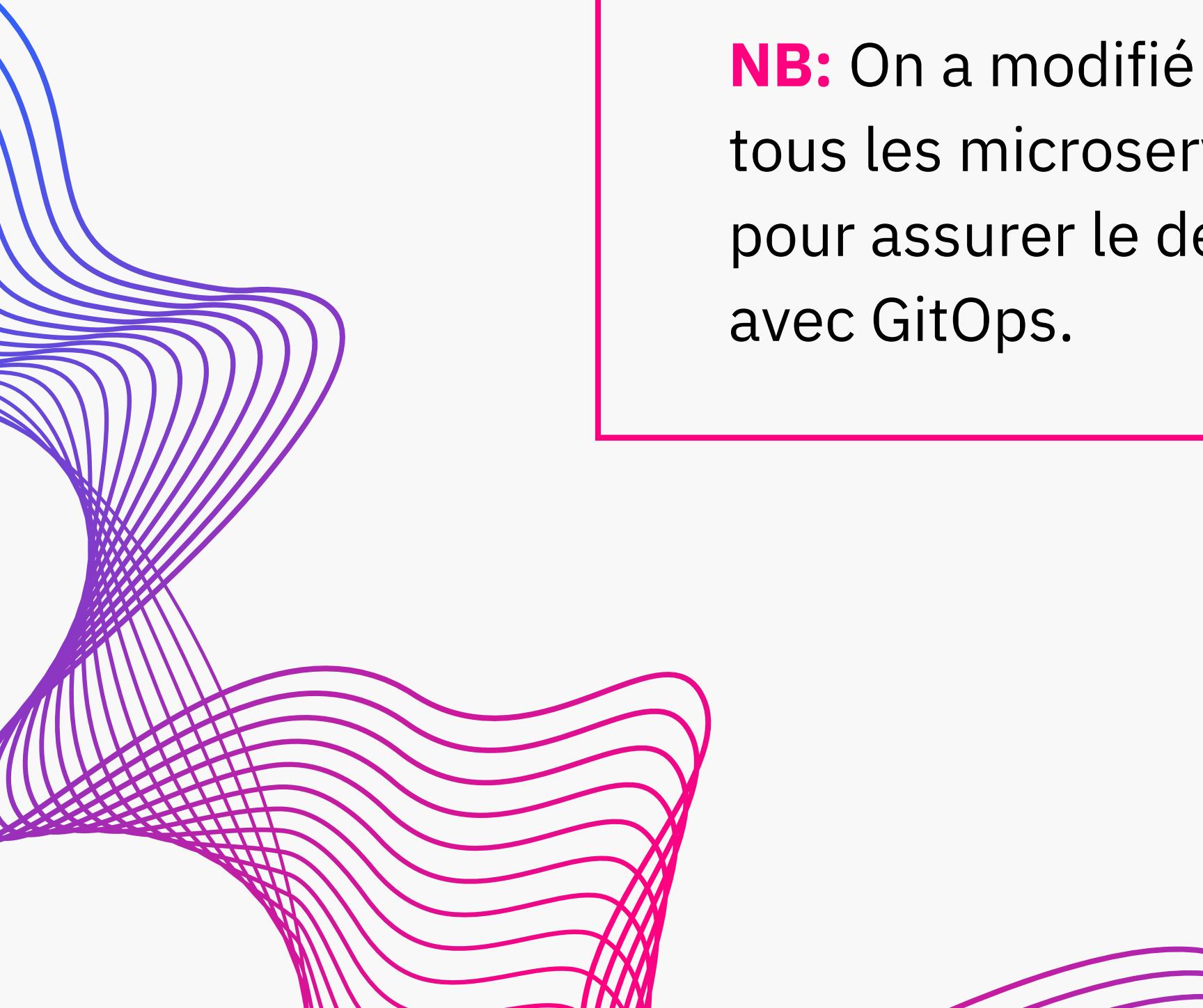
→ gitops-k8s-deploy

En utilisant le message de validation "1.1.0", ArgoCD détectera cette modification et déclenchera une mise à jour de l'application correspondante, en déployant la nouvelle version avec la balise "1.1.0".



The screenshot shows a detailed view of a Kubernetes pod named "produit-backend-ff8d7fcff-wkbpj". The pod is of type "Pod" and is currently running. It was created on "06/05/2023 01:42:10" and is using the image "kouss/produit:1.1.0". The pod is located in the "furnishop" namespace. There are tabs for "SUMMARY", "EVENTS", and "LOGS", with "SUMMARY" being the active tab. At the top right, there are "SYNC" and "DELETE" buttons.

KIND	Pod
NAME	produit-backend-ff8d7fcff-wkbpj
NAMESPACE	furnishop
CREATED_AT	06/05/2023 01:42:10
IMAGES	kouss/produit:1.1.0
STATE	Running

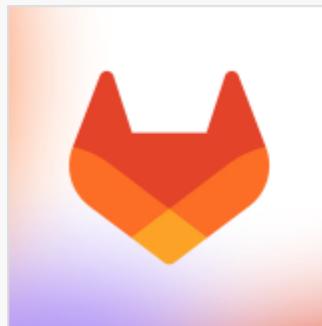


NB: On a modifié le fichier `.gitlab-ci.yml` de tous les microservices de la même manière pour assurer le déploiement sur Kubernetes avec GitOps.

Conclusion

Le développement de l'application FurniShop a impliqué l'utilisation de technologies telles que Node.js/Express, MongoDB, React, Docker, Kubernetes et ArgoCD. En utilisant ces technologies, nous étions en mesure de créer des services backend et frontend performants et évolutifs. L'utilisation de Docker a facilité la conteneurisation des services, tandis que Kubernetes a permis de gérer efficacement le déploiement et la mise à l'échelle des applications. Sans oublier GitLab CI qui a accéléré le cycle de développement. L'utilisation d'ArgoCD nous a permis de mettre en place une gestion automatisée des déploiements dans Kubernetes, assurant ainsi une livraison continue et une facilité de maintenance. En combinant ces technologies et ces outils, on a pu créer une application FurniShop robuste, scalable et facilement déployable, offrant une expérience utilisateur fluide et une gestion efficace des microservices.

Les liens de GitLab repositories



khadija oussakel / client-microservice · GitLab

GitLab.com



**khadija oussakel / commande-microservice ·
GitLab**

GitLab.com



**khadija oussakel / produit-microservice ·
GitLab**

GitLab.com



**khadija oussakel / paiement-microservice ·
GitLab**

GitLab.com



khadija oussakel / kubernetes · GitLab

GitLab.com

