Atelier 01 — Installation de l'environnement React dans Docker

Objectif de l'atelier

À la fin de cet atelier, chaque **participant** disposera d'un environnement de développement **ReactJS fonctionnel et isolé dans Docker**, prêt pour les ateliers suivants.

Ils comprendront :

- Les **principes de Docker** (images, conteneurs, volumes, orchestration).
- Comment créer un environnement ReactJS stable et portable.
- Pourquoi on utilise **Vite** pour créer une application React moderne.
- Le rôle des commandes npm install, npm run dev, et la logique du cycle de développement React.

Avant de commencer :

1. Installer Visual studio code (Dans le dossier ressources)
2. Installer Docker (Dans le dossier Ressources) ou bien Télécharger depuis : https://www.docker.com/products/docker-desktop/

Solution possible pour le démarrage de Docker sous Windows

Activer WSL2 (Windows uniquement)

```
dism.exe /online /enable-feature /featurename:Microsoft-Windows-Subsystem-Linux /all /norestart
dism.exe /online /enable-feature /featurename:VirtualMachinePlatform /all /norestart
wsl --update
wsl --install
```

Ensuite Installer Docker Desktop

- Suivre l'installation puis redémarrer.
- Vérifier avec les commandes suivantes:
    - docker --version
    - docker-compose --version

1. Introduction à Docker

- **Pourquoi Docker ?**

Docker est un outil de **virtualisation légère** qui permet d'exécuter des applications dans des **conteneurs isolés**.
Chaque conteneur contient tous les éléments nécessaires au fonctionnement de l'application (Node.js, React, dépendances...).

Avantages pour le développement ReactJS :

- 💡 Tous les participants travaillent dans **le même environnement** (peu importe Windows, Linux, macOS).
- ⚙️ Pas d'installation manuelle complexe de Node.js ni de conflits de versions.
- 🧱 Facile à **reconstruire ou supprimer** sans affecter le système local.
- 🚀 Portable : le projet peut être exécuté sur n'importe quelle machine équipée de Docker.

🔹 Concepts Docker essentiels

Élément	Description
Image	Modèle ou "photo" d'un environnement logiciel prêt à l'emploi (ex. : node:22-alpine).
Conteneur	Instance en exécution d'une image.
Dockerfile	Fichier décrivant les étapes de construction d'une image personnalisée.
Volume	Dossier partagé entre le conteneur et le disque local (pour sauvegarder le code).
docker-compose.yml	Fichier permettant de lancer plusieurs conteneurs ou configurer des services facilement.

2. Versions utilisées (stables à octobre 2025)

Logiciel	Version	Rôle
Docker Desktop	27.3.1	Moteur de conteneurisation
Docker Compose	2.29	Orchestration de services
Node.js	22 LTS (long-term support)	Environnement JavaScript
npm	10.9+	Gestionnaire de paquets Node
Vite	6.0	Outil moderne de création d'app React ***(A1)***
React	18.3.1	Bibliothèque JavaScript pour interfaces web

Ces versions sont stables, récentes et compatibles entre elles.
Le conteneur Docker téléchargera automatiquement ces versions exactes.

3. Pourquoi utiliser Vite pour créer l'application React ?

Avant 2022, les développeurs utilisaient souvent **Create React App (CRA)**.
Aujourd'hui, **Vite** est préféré car il est :

- **Beaucoup plus rapide** au démarrage et au rechargement (grâce à ESBuild).
- **Plus léger** : moins de dépendances et structure claire.
- **Compatible avec React, Vue, Svelte, etc.**

- **Hot Reload instantané** (actualisation automatique du navigateur).
- Prêt pour TypeScript, ESM, PostCSS, etc.

On peut toujours utiliser create-react-app, mais c'est **désormais déconseillé** car :

- Il est **non maintenu activement**.
- Il est **lent** et installe beaucoup de dépendances inutiles.
  👉 Donc dans cette formation, **on utilisera Vite**.

4. Structure du projet

Nous allons travailler sur le **disque local C:** de chaque participant.

Arborescence prévue pour chaque Application React :

```
C:\ReactProjects
 ├── Dockerfile
  ├──FirstApp
       ├── docker-compose.yml
       ├──   App      (Application React )
```

Pour **FirstApp** , lancer ces commandes :

```
mkdir C:\ReactProjects
cd C:\ReactProjects
type NUL > Dockerfile
mkdir FirstApp
cd FirstApp
mkdir app
type NUL > docker-compose.yml
exit
```

5. Contenu du fichier Dockerfile

Crée un fichier Dockerfile dans C:\ ReactProjects :

```
# Utiliser la dernière version stable de Node.js
FROM node:22-alpine
# Créer le dossier de travail à l'intérieur du conteneur
WORKDIR /usr/src/app

# Exposer le port de Vite
EXPOSE 5173
```

Explications :

- node:22-alpine → version légère et rapide de Node.js (Linux Alpine).

- WORKDIR → dossier où tout se passe dans le conteneur.
- EXPOSE 5173 → port de développement utilisé par Vite.

6. Contenu du fichier docker-compose.yml

```yaml
services:
  reactapp:
    container_name: FirstApp_container
    build:
      context: .
      dockerfile: ../Dockerfile
    ports:
      - "5173:5173"
    volumes:
      - ./app:/usr/src/app
    working_dir: /usr/src/app
    environment:
      - NODE_ENV=development
    command: tail -f /dev/null
```

Explication :

Structure générale

services:

- Déclare la section des **services** que Docker Compose va gérer.
- Chaque service correspond à un **conteneur**.

Définition du service reactapp

reactapp:

- Nom logique du service.
- Tu utiliseras ce nom pour lancer, arrêter ou référencer le conteneur (docker compose up reactapp, etc.).

container_name: FirstApp_container

- Définit un nom personnalisé pour le conteneur, au lieu d'un nom généré automatiquement.
- Le conteneur s'appellera **FirstApp_container**.

Configuration de l'image

build:

context: .

dockerfile: ../Dockerfile

- build: indique que l'image doit être construite à partir d'un Dockerfile.
- context: . → le dossier **courant** est utilisé comme contexte de build (tous les fichiers accessibles pendant le build).
- dockerfile: ../Dockerfile → le Dockerfile se trouve **dans le dossier parent** (..) et non dans le dossier actuel.

Configuration réseau

ports:

- "5173:5173"

- Expose le port du conteneur sur la machine hôte.
- host:container
- Ici, tu pourras accéder à l'application React via **http://localhost:5173**

Volumes (synchronisation fichiers)

volumes:

- ./app:/usr/src/app

- Monte le dossier local ./app dans le conteneur à l'emplacement /usr/src/app.
- Permet le **hot reload** : les modifications faites dans ton éditeur sont immédiatement visibles dans le conteneur.
- Sans ce volume, tu devrais reconstruire l'image à chaque modification.

Répertoire de travail

working_dir: /usr/src/app

- Définit le dossier **où les commandes seront exécutées** dans le conteneur.
- Par exemple, si tu fais npm install, il s'exécutera dans /usr/src/app.

Variables d'environnement

environment:

- NODE_ENV=development

- Définit l'environnement d'exécution.

- development active les fonctions de dev (ex : React Fast Refresh).

Commande de démarrage

```
    command: tail -f /dev/null
```

- Cette commande permet au conteneur de **rester actif** sans s'arrêter.
- tail -f /dev/null ne fait rien mais empêche le conteneur de quitter.
- Utile quand on veut **entrer dans le conteneur à la main** avec :
- docker exec -it FirstApp_container bash

Plus tard, tu remplaceras probablement cette commande par :

command: npm run dev

pour lancer réellement React.

Résumé

Élément	Rôle
build	Construit l'image depuis le Dockerfile parent
ports	Permet d'accéder à React via localhost:5173
volumes	Synchronise les fichiers locaux avec le conteneur
working_dir	Définit le dossier de travail interne
environment	Configure l'application en mode développement
command	Maintient le conteneur en vie sans exécuter React

7. Lancer l'environnement pour FirstApp

Ouvrez un terminal **dans C:\ReactProjects\FirstApp**, puis exécutez :

```
docker-compose up -d --build
```

- --build → force la reconstruction de l'image.
- -d → exécute en arrière-plan.

Ensuite, entrez dans le conteneur :

```
docker exec -it FirstApp_container sh
```

8. Lancer l'application react

Crée ton projet React :

npm create vite@latest . **-- --template**

```
/usr/src/app # npm create vite@latest . --template

> npx
> create-vite .

|
◇  Select a framework:
|  React
|
◇  Select a variant:
|  JavaScript
|
◇  Use rolldown-vite (Experimental)?:
|  No
|
◆  Install with npm and start now?
|  ● Yes / ○ No
└
```

Si le serveur Vite lancé, arrête-le (CTR + C)

Ensuite quitter de container avec la commande :

```
exit
```

9. Modifier le fichier vite.config.js

```
import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react'

// https://vitejs.dev/config/
export default defineConfig({
  plugins: [react()],
  server: {
    host: true,           // équivaut à --host
    watch: {
      usePolling: true,   // <== active le mode "polling"
      interval: 1000,     // <== vérifie les changements toutes les 1s
    },
  },
})
```

Puis arrêter et démarrer le container, directement avec le docker ou bien exécuter ces deux commandes

```
docker-compose down
docker-compose up -d --build
```

Ensuite, entrez dans le conteneur :

```
docker exec -it FirstApp_container sh
```

Lancer cette commande :

```
npm run dev -- --host
```

10. Accéder à l'application

Une fois npm run dev lancé dans le conteneur,
ouvre ton navigateur sur ton poste de travail :

👉 http://localhost:5173

Tu verras l'écran d'accueil de Vite + React 🎉
Cela prouve que ton environnement est fonctionnel !

11. Vérification & test

Lancer une invite de commande et essayer les commandes suivantes

Commandes utiles :

```
docker ps              # voir si le conteneur tourne
docker-compose down    # arrêter l'environnement
```

12. Conclusion

- Docker permet d'avoir un **environnement ReactJS stable, portable et reproductible**.
- Vite offre un **outillage rapide et moderne** pour la création d'applications React.
- La commande npm run dev démarre le **serveur de développement local** de Vite.
- Cet environnement servira de **base à tous les ateliers suivants (J1 → J5)**.

Annexe 1

En 2025, **plusieurs outils** permettent de **créer et configurer une application ReactJS**. Ils ont évolué au fil des années, chacun ayant ses avantages selon le **niveau**, **le besoin en performance** et **le type de projet**.

Vite (recommandé)

- 🧩 **Description :** Outil moderne et ultra-rapide pour le développement front-end (créé par Evan You, le créateur de Vue.js).
- ⚙️ **Principe :** Utilise **ESBuild** et **Rollup** pour un démarrage quasi instantané et un hot reload rapide.
- 💡 **Pourquoi on le choisit :**
  - Temps de démarrage extrêmement rapide.
  - Moins de configuration.
  - Support natif de React, Vue, Svelte, etc.
  - Idéal pour les formations et petits projets.
- 📦 **Commande :**
- npm create vite@latest myProjectReact -- --template react

Create React App (CRA) ***(ancienne méthode officielle)***

- 🧩 **Description :** Outil officiel de Facebook pour générer un projet React préconfiguré (Webpack, Babel, ESLint, etc.).
- 🚫 **Statut :** Depuis 2023, **déprécié** — non recommandé pour les nouveaux projets car lent et non maintenu activement.
- 💡 **Intérêt pédagogique :**
  - Permet de comprendre comment fonctionnaient les projets React avant Vite.
- 📦 **Commande :**
- npx create-react-app myProjectReact

Next.js

- 🧩 **Description :** Framework React complet développé par **Vercel**, pour les projets professionnels.
- 🌐 **Fonctionnalités clés :**
    - Rendu côté serveur (SSR)
    - Génération statique (SSG)
    - Routage automatique
    - API intégrée
- 💡 **Idéal pour :** Applications web complexes, SEO, sites dynamiques.
- 📦 **Commande :**
- npx create-next-app@latest myProjectReact

Remix

- 🧩 **Description :** Framework React moderne orienté **performances et expérience utilisateur**, aussi développé par l'équipe React Router.
- 💡 **Avantages :**
    - Gestion native du chargement de données et des formulaires.
    - Excellente performance pour les applications web à routes multiples.
- 📦 **Commande :**
- npx create-remix@latest

Parcel

- 🧩 **Description :** Bundler rapide qui nécessite **zéro configuration**.
- 💡 **Avantages :**
    - Pas besoin de fichier de configuration.
    - Support automatique de React, TypeScript, Sass...
- 📦 **Exemple :**
- npm install -g parcel
- parcel index.html

Webpack (manuel)

- 🧩 **Description :** Bundler historique utilisé avant CRA.
- 💡 **Utilité :**
  - Excellent pour apprendre les concepts bas niveau (entry, output, loaders, plugins...).
  - Requiert beaucoup de configuration manuelle.
- 💡 **À utiliser uniquement** pour un atelier avancé ou de démo technique.

Comparatif rapide

Outil	Rapidité	Configuration	Usage recommandé	Statut
Vite	⚡ ⚡ ⚡	Très simple	Formations, petits projets	✅ Actuel
Create React App	⚡	Très simple	Ancien apprentissage	❌ Déprécié
Next.js	⚡ ⚡	Moyenne	Applications complexes	✅ Populaire
Remix	⚡ ⚡	Moyenne	Applications modernes UX	✅ Populaire
Parcel	⚡ ⚡	Très simple	Petits projets rapides	✅ Option alternative
Webpack	⚡	Difficile	Approche pédagogique avancée	⚙️ Technique

💬 En résumé (à expliquer à vos participants)

Aujourd'hui, **Vite est la méthode recommandée** pour créer une application React, car elle combine **simplicité, rapidité et compatibilité**.

D'autres outils comme **Next.js** ou **Remix** peuvent être utilisés dans des contextes plus professionnels ou avancés, mais **Vite reste parfait pour débuter** et assurer la portabilité dans Docker.

Annexe 2 : Export et import les images de Docker

🧱 1️⃣ Solution la plus simple : Exporter et importer les images Docker

💾 Sur ta machine (formateur) :

1. Liste les images que tu veux partager :

```
docker images
```

2. Sauvegarde-les dans un fichier .tar :

```
docker save -o firstapp-reactapp_images.tar firstapp-reactapp
```

👉 Cela crée un seul fichier contenant toutes les images nécessaires.

3. Copie le fichier sur une **clé USB** ou un **disque externe**.

💻 Sur les machines des apprenants :

1. Copie le fichier monerp_images.tar sur leurs ordinateurs.
2. Importer les images dans Docker :

```
docker load -i firstapp-reactapp_images.tar
```

3. Vérifie qu'elles sont bien là :

```
docker images
```

4. Copier le dossier **ReactProjects** de ressources sous **C :**
5. Avec l'invite de commande accéder au dossier **C:\ReactProjects\FirstApp** et lancer la commande suivante :

```
docker-compose up -d --build
```