

Contents

Préparer son poste (à faire AVANT la formation)	1
Vue d'ensemble	1
1. Système	1
Windows 10/11 : WSL2 (Ubuntu)	1
macOS	2
Linux (Ubuntu/Debian)	2
2. Git, identité et clé SSH GitHub	2
3. VS Code	2
4. Docker + Compose v2	3
Windows (WSL2) et macOS : Docker Desktop	3
Linux : Docker Engine	3
5. Node.js 20	3
6. GitHub CLI gh (facultatif)	3
Vérifier que tout marche	3
Problèmes fréquents	4

Préparer son poste (à faire AVANT la formation)

Durée : 45 à 60 minutes (le téléchargement de Docker est le plus long). À la fin, vous lancez la section « Vérifier que tout marche » : si tout est conforme, vous êtes prêt.

Vous êtes bloqué ? Notez le message d'erreur **exact** (copier-coller ou capture) et prévenez le formateur avant le jour 1.

Vue d'ensemble

Outil	Pourquoi
Git + compte GitHub + clé SSH	versionner le code, travailler à plusieurs
VS Code + extensions	éditer Dockerfile, YAML, workflows
Docker (Desktop ou Engine) + Compose v2	lancer l'application dans des conteneurs
Node.js 20	lancer les tests/lint de TaskFlow sur le poste
gh (facultatif)	GitHub en ligne de commande

1. Système

Windows 10/11 : WSL2 (Ubuntu)

Tout le travail de la formation se fait **dans Ubuntu (WSL2)**, pas dans PowerShell. 1. PowerShell **en administrateur** : `wsl --install -d Ubuntu-24.04` puis redémarrer. 2. Au premier lancement d'Ubuntu : choisir un nom d'utilisateur et un mot de passe Linux. 3. Vérifier : `wsl -l -v` doit afficher Ubuntu-24.04 ... VERSION 2. 4. Travaillez dans `/home/<vous>/` (et **pas** dans `/mnt/c/...`, très lent et source de bugs de droits). 5. Dans Ubuntu : `sudo apt update && sudo apt install -y git curl unzip ca-certificates openssl`

macOS

Installer Homebrew (<https://brew.sh>) puis `brew install git`.

Linux (Ubuntu/Debian)

`sudo apt update && sudo apt install -y git curl unzip ca-certificates openssl`

2. Git, identité et clé SSH GitHub

```
git config --global user.name "Prénom Nom"
git config --global user.email "votre-email-github@exemple.fr"
git config --global init.defaultBranch main
git config --global pull.rebase false
git config --global core.editor "code --wait"      # facultatif (après installation de VS Code)
git config --global --list
```

Clé SSH pour GitHub (évite de retaper un mot de passe/jeton à chaque `git push`) :

```
ssh-keygen -t ed25519 -C "votre-email-github@exemple.fr"      # Entrée à chaque question
cat ~/.ssh/id_ed25519.pub                                     # c'est la clé PUBLIQUE
```

Sur GitHub : **Settings > SSH and GPG keys > New SSH key** > coller le contenu affiché.

Test :

```
ssh -T git@github.com
# Hi <pseudo>! You've successfully authenticated, but GitHub does not provide shell access
```

(Au premier essai, répondre yes à la question sur l’empreinte.) **Ne partagez jamais** `id_ed25519` (sans `.pub`).

3. VS Code

Installer depuis <https://code.visualstudio.com>. Sous Windows, installer aussi l’extension **WSL** : depuis le terminal Ubuntu, `code .` ouvre VS Code relié à WSL.

Extensions (menu Extensions, `Ctrl+Shift+X`) :

Extension	Identifiant
Docker	ms-azuretools.vscode-docker
YAML (Red Hat)	redhat.vscode-yaml
GitHub Actions	github.vscode-github-actions
GitLens	eamodio.gitlens
(Windows) WSL	ms-vscode-remote.remote-wsl

En ligne de commande : `code --install-extension ms-azuretools.vscode-docker` (idem pour les autres).

4. Docker + Compose v2

Windows (WSL2) et macOS : Docker Desktop

1. Installer Docker Desktop (<https://www.docker.com/products/docker-desktop/>).
2. Windows : *Settings* > *General* : « Use the WSL 2 based engine » coché ; *Settings* > *Resources* > *WSL integration* : activer **Ubuntu-24.04**.
3. Lancer Docker Desktop et attendre « Engine running ».
4. Sans licence professionnelle ? Docker Desktop est gratuit pour l'enseignement et les petites structures ; en cas de doute, demandez au formateur.

Linux : Docker Engine

Suivre la procédure officielle de votre distribution : <https://docs.docker.com/engine/install/ubuntu/> (dépôt apt Docker, paquets docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin). Puis :

```
sudo usermod -aG docker $USER      # autorise docker sans sudo
newgrp docker                       # ou fermer/ouvrir la session
```

5. Node.js 20

Avec **nvm** (Linux/macOS/WSL) :

```
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.40.1/install.sh | bash
# fermer et rouvrir le terminal, puis :
nvm install 20
nvm use 20
node -v      # v20.x.x
```

Alternative : **fnm** (curl -fsSL <https://fnm.vercel.app/install> | bash, puis fnm install 20).

6. GitHub CLI gh (facultatif)

- macOS : brew install gh — Ubuntu/WSL : voir https://github.com/cli/cli/blob/trunk/docs/install_linux.md
- Puis gh auth login (GitHub.com > SSH > suivre l'assistant). Test : gh auth status.

Vérifier que tout marche

Copiez ces commandes dans le terminal (Ubuntu/WSL sous Windows).

Commande	Résultat attendu (les numéros peuvent varier)
git --version	git version 2.4x.x
git config user.name && git config user.email	vos nom et e-mail
ssh -T git@github.com	Hi <pseudo>! You've successfully authenticated...
node -v && npm -v	v20.x.x puis 10.x.x
docker --version	Docker version 2x.x.x
docker compose version	Docker Compose version v2.x.x (ou v5.x : toute version ≥ 2 convient)
docker run --rm hello-world	Hello from Docker!
docker run --rm -p 8080:80 -d --name test-nginx nginx:1.27-alpine puis curl -s localhost:8080 \ head -n 4	début de la page Welcome to nginx!
docker rm -f test-nginx	test-nginx
code --version	un numéro de version
gh --version (facultatif)	gh version 2.x.x

Ports 80 et 443 : en jour 4, le « serveur » (proxy Nginx) publie les ports 80 (HTTP) et 443 (HTTPS). Ils doivent être libres :

```
ss -ltn | grep -E ':(80|443)\b'      # Linux / WSL : aucune ligne = libres
lsof -iTCP:80 -sTCP:LISTEN           # macOS
```

Sinon, voir « port déjà pris » ci-dessous.

Résolution de taskflow.localhost : les navigateurs modernes (Chrome, Firefox, Edge) considèrent tout *.localhost comme 127.0.0.1 automatiquement : rien à configurer. Pour curl (ou si un outil ne le résout pas) :

```
curl -k --resolve taskflow.localhost:443:127.0.0.1 https://taskflow.localhost/health
```

(-k = accepter le certificat auto-signé ; --resolve = associer le nom à 127.0.0.1.) Sous Windows avec le navigateur Windows et le serveur dans WSL2 : localhost est transféré à WSL2, ça fonctionne ; sinon utiliser https://localhost.

Problèmes fréquents

Symptôme	Cause	Solution
Cannot connect to the Docker daemon	Docker pas lancé	Lancer Docker Desktop (attendre « running ») ; Linux : sudo systemctl start docker
permission denied ... /var/run/docker.sock (Linux)	Pas dans le groupe docker	sudo usermod -aG docker \$USER puis newgrp docker ou reconnexion ; vérifier avec id

Symptôme	Cause	Solution
docker: command not found dans WSL	Intégration WSL non activée	Docker Desktop > Settings > Resources > WSL integration > cocher Ubuntu, Apply
Bind for 0.0.0.0:80 failed: port is already allocated / address already in use	Un autre programme (Apache, Nginx, IIS, Skype, autre conteneur) utilise le port	Trouver : <code>ss -ltnp \ grep :80</code> . Soit l'arrêter, soit changer les ports de la prod locale : dans <code>~/taskflow-prod/.env</code> , <code>HTTP_PORT=8081</code> et <code>HTTPS_PORT=8443</code> , puis <code>docker compose up -d</code> ; l'URL devient <code>https://taskflow.localhost:8443</code>
Port 80 refusé sans autre programme (Linux rootless / Podman)	Ports < 1024 réservés	Utiliser <code>HTTP_PORT=8081 / HTTPS_PORT=8443</code> (pas 8080 : déjà pris par le front de développement)
Dossiers montés vides, très lents (Windows)	Projet sous /mnt/c	Cloner dans <code>/home/<vous>/... côté WSL</code>
git push demande un mot de passe	Dépôt cloné en HTTPS	<code>git remote set-url origin git@github.com:<pseudo>/<repo>.git</code>
Permission denied (publickey)	Clé non ajoutée sur GitHub / pas chargée	Rejouer la section 2 ; <code>ssh -vT git@github.com</code> pour voir les clés essayées
node: command not found après installation de nvm	Terminal non rechargé	Fermer/rouvrir le terminal ; <code>source ~/.bashrc</code>
WSL très gourmand en mémoire	Réglage par défaut	Fichier <code>%UserProfile%\wslconfig</code> : <code>[wsl2] puis memory=4GB ; wsl --shutdown</code>
Avertissement « connexion non sécurisée » sur <code>https://taskflow.localhost</code>	Normal : certificat auto-signé	« Avancé » > « Continuer ». Sur un vrai serveur : Let's Encrypt