### **Guide de Démarrage et de Configuration de l'Environnement de Développement Local**

* **Version :** 1.0
* **Date :** 21 juin 2025
* **Propriétaire :** Ingénieur DevOps

### **Partie 1 : Prérequis Fondamentaux - Les Outils de Base**

#### **1.1. Introduction**

L'objectif de ce guide est de vous permettre de configurer un environnement de développement complet, conteneurisé et reproductible pour le projet SkillForge AI. En suivant ces étapes, vous disposerez d'un environnement local stable, identique à celui de tous les autres membres de l'équipe.

Cela permet de réduire le temps de configuration initial et d'éliminer les problèmes du type "ça marche sur ma machine".

#### **1.2. Systèmes d'Exploitation Recommandés**

Pour garantir la compatibilité des outils et des scripts, l'un des systèmes d'exploitation suivants est requis :

* **Linux (Recommandé) :** Toute distribution moderne basée sur Debian (comme Ubuntu 22.04+) ou Fedora.
* **macOS (Recommandé) :** Version 12 (Monterey) ou supérieure, compatible avec les puces Apple Silicon (M1/M2/M3) et Intel.
* **Windows 10/11 (Supporté) :** L'utilisation de **WSL 2 (Windows Subsystem for Linux)** est **obligatoire**. Le développement directement sur l'environnement Windows natif (CMD, PowerShell) n'est pas supporté pour ce projet.

#### **1.3. Installation et Configuration de Git**

Git est notre système de contrôle de version. Il est probable que vous l'ayez déjà, mais vérifiez sa présence et sa configuration.

1. **Installation :**
   * Sur Debian/Ubuntu : sudo apt update && sudo apt install git
   * Sur macOS (avec Homebrew) : brew install git
2. **Configuration Initiale Obligatoire :**
   * Ouvrez un terminal et exécutez les commandes suivantes en remplaçant les informations par les vôtres. Ceci est indispensable pour que vos commits vous soient correctement attribués.

git config --global user.name "Votre Nom Complet"

git config --global user.email "votre.email@fournisseur.com"

#### **1.4. Éditeur de Code : Visual Studio Code**

VS Code est l'éditeur de code standard pour ce projet afin d'assurer que tout le monde bénéficie des mêmes outils de formatage et de linting.

1. **Installation :** Téléchargez et installez la version stable depuis le [site officiel de Visual Studio Code](https://code.visualstudio.com/).
2. **Installation des Extensions Obligatoires :** Pour une expérience de développement optimale et cohérente, les extensions suivantes doivent être installées. Vous pouvez les trouver dans l'onglet "Extensions" de VS Code ou les installer via la ligne de commande.

| Nom de l'Extension | Identifiant (ID) | Justification |
| --- | --- | --- |
| Python | ms-python.python | Support complet de Python : linting, débogage, IntelliSense. |
| Ruff | charliermarsh.ruff | Linter et formateur ultra-rapide pour Python, configuré pour notre projet. |
| Prettier - Code formatter | esbenp.prettier-vscode | Formatage automatique et standardisé du code Front-End (JS, TS, CSS, JSON). |
| ESLint | dbaeumer.vscode-eslint | Intègre les règles de linting pour le code JavaScript et TypeScript. |
| Docker | ms-azuretools.vscode-docker | Intégration avancée avec Docker : gestion des conteneurs, images, etc. |
| Remote - Containers | ms-vscode-remote.remote-containers | Permet d'ouvrir un dossier à l'intérieur d'un conteneur de développement. |
| Terraform | hashicorp.terraform | coloration syntaxique et autocomplétion pour les fichiers Terraform. |
| YAML | redhat.vscode-yaml | Validation et autocomplétion pour les fichiers YAML (ex: GitHub Actions). |

Commande d'installation rapide :

*for extension in ms-python.python charliermarsh.ruff esbenp.prettier-vscode dbaeumer.vscode-eslint ms-azuretools.vscode-docker ms-vscode-remote.remote-containers hashicorp.terraform redhat.vscode-yaml; do code --install-extension $extension; done*

#### **1.5. Moteur de Conteneurisation : Docker Desktop**

Docker est le cœur de notre environnement local. Il nous permet d'exécuter tous les services du projet (base de données, API, etc.) dans des conteneurs isolés.

1. **Installation :** Téléchargez et installez **Docker Desktop** depuis le [site officiel de Docker](https://www.docker.com/products/docker-desktop/).
2. **Configuration pour Windows :** Si vous êtes sur Windows, assurez-vous que Docker Desktop est configuré pour utiliser le **backend WSL 2**. Cette option est généralement proposée lors de l'installation.
3. **Vérification de l'Installation :** Une fois l'installation terminée et Docker Desktop lancé, ouvrez un terminal et exécutez les commandes suivantes pour confirmer que tout fonctionne :
   * Vérifier la version de Docker : docker --version
   * Vérifier la version de Docker Compose : docker compose version
   * Lancer un conteneur de test : docker run hello-world

Si cette dernière commande affiche un message de bienvenue de "hello-world", votre installation de Docker est fonctionnelle.

### **Partie 2 : Configuration des Environnements d'Exécution (Runtimes)**

#### **2.1. Gestion des Versions de Python avec pyenv**

pyenv est un outil qui nous permet d'installer et de gérer plusieurs versions de Python sur une même machine. Il nous assure que le projet utilisera toujours la version spécifiée, évitant ainsi les problèmes de compatibilité.

1. **Installation de pyenv :**

**Sur macOS (avec Homebrew) :**

**brew update**

**brew install pyenv**

**Sur Linux :** La méthode la plus simple est d'utiliser le script d'installation officiel :

curl https://pyenv.run | bash

**2. Configuration du Shell :**

* Après l'installation, vous devez ajouter les lignes suivantes à la fin de votre fichier de configuration de shell (~/.zshrc si vous utilisez Zsh, ou ~/.bash\_profile / ~/.bashrc si vous utilisez Bash) pour que pyenv soit chargé automatiquement.

export PYENV\_ROOT="$HOME/.pyenv"

[[ -d $PYENV\_ROOT/bin ]] && export PATH="$PYENV\_ROOT/bin:$PATH"

eval "$(pyenv init -)"

* **Important :** Redémarrez votre terminal pour que ces changements prennent effet.

**3. Installation de la Version Python du Projet :**

* Le monorepo du projet contient un fichier nommé .python-version qui spécifie la version exacte à utiliser (ex: 3.11.5).
* Installez cette version spécifique avec pyenv

# Remplacez 3.11.5 par la version indiquée dans le fichier .python-version

pyenv install 3.11.5

Une fois dans le répertoire du projet, pyenv détectera automatiquement le fichier .python-version et utilisera la bonne version de Python.

**Vérification :** Naviguez dans le répertoire du projet cloné (étape que nous verrons dans la partie 3) et exécutez :

python --version

# Doit afficher : Python 3.11.5

#### **2.2. Gestion des Versions de Node.js avec nvm**

nvm (Node Version Manager) est l'équivalent de pyenv pour l'écosystème Node.js, requis pour le développement Front-End.

1. **Installation de nvm :**
   * Exécutez le script d'installation via curl ou wget. La commande suivante est la plus courante :

curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.7/install.sh | bash

* Le script d'installation ajoutera automatiquement les lignes de configuration nécessaires à votre fichier de shell.

**Installation de la Version Node.js du Projet :**

* Le projet contient un fichier .nvmrc qui spécifie la version de Node.js à utiliser (ex: lts/iron ou une version spécifique comme 20.11.0).
* **Important :** Redémarrez votre terminal après l'installation de nvm.
* Naviguez dans le répertoire du projet et exécutez simplement : nvm install

nvm lira automatiquement le fichier .nvmrc, téléchargera et installera la version requise. Ensuite, activez-la : nvm use

Vérification : node --version # Doit afficher la version spécifiée dans le fichier .nvmrc

#### **2.3. Installation de la CLI Terraform**

Bien que la plupart des déploiements soient gérés par le pipeline CI/CD, avoir la CLI Terraform en local est indispensable pour inspecter l'infrastructure (terraform plan) ou pour y contribuer.

1. **Installation :**
   * **Sur macOS (avec Homebrew) :**

brew tap hashicorp/tap

brew install hashicorp/tap/terraform

**Sur Linux (Debian/Ubuntu) :** Suivez les étapes officielles de HashiCorp pour ajouter leur dépôt et installer Terraform avec apt. C'est la méthode la plus propre.

wget -O- https://apt.releases.hashicorp.com/gpg | sudo gpg --dearmor -o /usr/share/keyrings/hashicorp-archive-keyring.gpg

echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg] https://apt.releases.hashicorp.com $(lsb\_release -cs) main" | sudo tee /etc/apt/sources.list.d/hashicorp.list

sudo apt update && sudo apt install terraform

Vérification : terraform --version

### **Partie 3 : Accès à la Plateforme Cloud (GCP) et au Code (GitHub)**

Cette section vous guidera pour configurer un accès sécurisé à notre dépôt de code et à notre projet cloud. À la fin de cette partie, votre machine ne sera plus une île isolée, mais un poste de travail connecté et authentifié, prêt à interagir avec le projet.

#### **3.1. Configuration de l'Accès Sécurisé à GitHub**

Nous utilisons des clés SSH pour communiquer avec GitHub. C'est une méthode bien plus sécurisée et pratique que l'authentification par mot de passe.

1. **Vérifier l'existence d'une clé SSH :**
   * Ouvrez votre terminal et tapez la commande suivante pour lister les fichiers dans votre répertoire SSH : ls -al ~/.ssh
2. Cherchez des fichiers nommés id\_rsa.pub, id\_ecdsa.pub, ou, de préférence, id\_ed25519.pub. Si l'un de ces fichiers existe, vous pouvez passer à l'étape 4.

**Générer une nouvelle clé SSH (si nécessaire) :**

* Si vous n'avez pas de clé, générez-en une nouvelle avec l'algorithme ed25519 (plus moderne et sécurisé) : ssh-keygen -t ed25519 -C "[votre.email@fournisseur.com](mailto:votre.email@fournisseur.com)"
* Appuyez sur Entrée pour accepter l'emplacement par défaut.
* Il est **fortement recommandé** de saisir une "passphrase" (mot de passe) pour protéger votre clé.

**Ajouter la clé à l'agent SSH :**

* Pour éviter de saisir votre passphrase à chaque interaction, ajoutez votre clé à l'agent SSH :

eval "$(ssh-agent -s)"

ssh-add ~/.ssh/id\_ed25519

**Ajouter la clé publique à votre compte GitHub :**

* Copiez le contenu de votre clé **publique** dans votre presse-papiers.

# Sur macOS

pbcopy < ~/.ssh/id\_ed25519.pub

# Sur Linux (nécessite xclip)

sudo apt install xclip

xclip -selection clipboard < ~/.ssh/id\_ed25519.pub

* Rendez-vous sur GitHub, dans les paramètres de votre compte, à la section [SSH and GPG keys](https://github.com/settings/keys).
* Cliquez sur "New SSH key", donnez-lui un titre reconnaissable (ex: "Laptop Pro M3"), et collez votre clé dans le champ "Key". Cliquez sur "Add SSH key".

**Tester la connexion :**

* Vérifiez que tout fonctionne en tapant : ssh -T git@github.com
* Vous devriez voir un message de bienvenue : Hi <votre-username>! You've successfully authenticated...

**Cloner le Dépôt du Projet :**

* Vous êtes maintenant prêt à cloner le code source du projet. Naviguez dans votre répertoire de travail et exécutez : git clone git@github.com:SkillForgeAI/monorepo.git

*(Note: le nom du dépôt SkillForgeAI/monorepo est un exemple et sera remplacé par le nom réel du dépôt)*.

#### **3.2. Configuration du SDK Google Cloud (gcloud CLI)**

La CLI gcloud est notre outil principal pour interagir avec notre projet GCP depuis le terminal. Elle est notamment indispensable pour permettre à Docker de télécharger nos images privées.

1. **Initialisation du SDK :**
   * Dans votre terminal, lancez la configuration initiale : gcloud init
2. Suivez les instructions :
   * Connectez-vous à votre compte Google lorsque le navigateur s'ouvre.
   * Choisissez le projet GCP correspondant au projet, par exemple skillforge-ai-dev.
   * Vous pouvez sauter la configuration de la région/zone par défaut pour le moment.

**Authentification de l'Utilisateur :**

* Pour vous assurer que vous êtes bien authentifié, vous pouvez exécuter :

gcloud auth login

**Configurer les "Application Default Credentials" (ADC) :**

* C'est une étape cruciale pour que le code que vous exécutez en local puisse s'authentifier auprès des APIs Google (ex: pour parler à Google Cloud Storage).

gcloud auth application-default login

**Autoriser Docker à accéder à notre registre d'images (Artifact Registry) :**

* C'est l'étape qui vous permettra de lancer le projet avec Docker Compose. Cette commande configure Docker pour qu'il utilise les identifiants gcloud pour s'authentifier.

# Remplacez <gcp-region> par la région de notre Artifact Registry (ex: "europe-west1")

gcloud auth configure-docker europe-west1-docker.pkg.dev

* Un message de succès confirmera que la configuration est terminée.

**Vérification Finale :**

* Pour voir votre configuration active, exécutez : gcloud config list
* Vérifiez que les champs account et project correspondent à votre compte et au projet SkillForge.

Félicitations ! Votre machine est maintenant un poste de travail de confiance, capable de communiquer de manière sécurisée avec nos deux plateformes centrales.

### **Partie 4 : Lancement du Projet en Local avec Docker Compose**

Cette dernière étape est la plus gratifiante. Grâce à la configuration que nous avons mise en place, vous allez pouvoir lancer tous les microservices, les bases de données et les dépendances du projet avec seulement deux ou trois commandes.

#### **4.1. Le Cœur de l'Environnement Local : docker-compose.yml**

À la racine de notre monorepo se trouve un fichier nommé docker-compose.yml. Ce fichier est la recette de notre environnement de développement local. Il est orchestré par l'outil docker compose pour définir et lier tous les services nécessaires à l'exécution de l'application.

Ce fichier définit :

* **Les Services :** Chaque composant de notre architecture (frontend, user-service, project-service, postgres-db, redis-cache, etc.).
* **Les Images Docker :** Il indique à Docker d'utiliser soit une image publique (comme postgres:16), soit de construire une image personnalisée à partir d'un Dockerfile présent dans le code.
* **Les Réseaux :** Il crée un réseau privé virtuel pour nos conteneurs, leur permettant de communiquer entre eux en utilisant simplement leur nom de service (ex: le user-service peut atteindre la base de données à l'adresse postgres-db:5432).
* **Les Volumes :** C'est une partie cruciale pour le développement. Nous "montons" nos répertoires de code source locaux directement à l'intérieur des conteneurs. Ainsi, lorsque vous modifiez un fichier dans VS Code, la modification est instantanément répercutée dans le conteneur en cours d'exécution, permettant un rechargement à chaud (Hot Reloading) sans avoir à reconstruire l'image.

#### **4.2. Gestion des Secrets Locaux - Le Fichier .env**

Notre docker-compose.yml a besoin de secrets (mots de passe, clés d'API, etc.) pour fonctionner, mais nous ne pouvons pas les stocker dans Git. Nous utilisons donc un fichier .env local.

1. **Localisez le Template :** À la racine du projet, vous trouverez un fichier nommé .env.template. Ce fichier liste toutes les variables d'environnement dont le projet a besoin.
2. **Créez votre Fichier d'Environnement :** Copiez ce template pour créer votre propre fichier .env. Ce fichier est ignoré par Git et ne sera jamais partagé.

cp .env.template .env

**Remplissez les Valeurs :** Ouvrez le nouveau fichier .env dans VS Code et remplissez les valeurs manquantes. Pour le développement local, des valeurs simples sont suffisantes.

* Exemple :

### **Contenu du Fichier .env.template**

Voici le contenu exhaustif que doit contenir le fichier .env.template à la racine de notre monorepo.

# ===================================================================

# CONFIGURATION DE LA BASE DE DONNEES POSTGRESQL (pour Docker Compose)

# ===================================================================

# Utilisateur pour la base de données locale

POSTGRES\_USER=skillforge\_user

# Mot de passe pour l'utilisateur PostgreSQL local. DOIT être changé.

POSTGRES\_PASSWORD=CHANGEME\_MOT\_DE\_PASSE\_LOCAL\_TRES\_FORT

# Nom de la base de données locale

POSTGRES\_DB=skillforge\_db

# Port sur lequel la base de données sera accessible depuis votre machine

POSTGRES\_PORT=5432

# URL de connexion complète, utilisée par les services applicatifs pour se connecter à la BDD.

# Le mot de passe sera lu depuis la variable ci-dessus.

# Le host 'postgres-db' est le nom du service dans docker-compose.

DATABASE\_URL=postgresql+psycopg://skillforge\_user:${POSTGRES\_PASSWORD}@postgres-db:5432/skillforge\_db

# ===================================================================

# CONFIGURATION DU BROKER DE MESSAGES REDIS (pour Docker Compose)

# ===================================================================

# Host du service Redis dans docker-compose

REDIS\_HOST=redis-cache

# Port standard de Redis

REDIS\_PORT=6379

# ===================================================================

# CONFIGURATION DES SERVICES BACK-END (Commune à tous les microservices)

# ===================================================================

# Clé secrète pour la signature des jetons JWT.

# NE PAS UTILISER CETTE VALEUR. Générez la vôtre avec la commande : openssl rand -hex 32

JWT\_SECRET\_KEY=CHANGEME\_A\_GENERER\_AVEC\_LA\_COMMANDE\_OPENSSL\_RAND

# Algorithme de signature des JWT. HS256 est utilisé en local pour sa simplicité (clé symétrique).

# En production, nous utiliserons RS256 (clé asymétrique), comme défini dans le CDC Back-End.

JWT\_ALGORITHM=HS256

# Durée de validité d'un jeton d'accès en minutes.

ACCESS\_TOKEN\_EXPIRE\_MINUTES=60

# ===================================================================

# CONFIGURATION DU FRONT-END (pour l'environnement Vite)

# ===================================================================

# URL de base de l'API Back-End que l'application Front-End doit appeler.

# En local, elle pointe vers le port de l'API Gateway ou d'un des services.

VITE\_API\_BASE\_URL=http://localhost:8000

# ===================================================================

# CONFIGURATION POUR LE DÉVELOPPEMENT LOCAL (GCP)

# ===================================================================

# ID de votre projet Google Cloud de développement/test.

GCP\_PROJECT\_ID=skillforge-ai-dev

# Nom du bucket GCS utilisé pour les uploads de fichiers en local (si testé hors Docker).

GCS\_BUCKET\_UPLOADS=skillforge-dev-uploads

Détail de Chaque Variable d'Environnement

| Variable | Service(s) concerné(s) | Description et Exemple de Valeur (Locale) |
| --- | --- | --- |
| POSTGRES\_USER | PostgreSQL, Tous les services Back-End | Nom de l'utilisateur pour la base de données locale. skillforge\_user est une bonne valeur par défaut. |
| POSTGRES\_PASSWORD | PostgreSQL, Tous les services Back-End | Mot de passe pour l'utilisateur PostgreSQL local. Doit être changé pour une valeur complexe (ex: Th1sIsMyS3cr3tP@ss). |
| POSTGRES\_DB | PostgreSQL, Tous les services Back-End | Nom de la base de données qui sera créée au premier lancement. skillforge\_db est la valeur standard. |
| POSTGRES\_PORT | Docker Compose | Port de votre machine locale qui sera mappé sur le port de la base de données dans le conteneur. 5432 est le standard. |
| DATABASE\_URL | Tous les services Back-End, Alembic | L'URL de connexion complète utilisée par SQLAlchemy. Le format est fixe, seul le mot de passe est injecté. |
| REDIS\_HOST | Services utilisant Redis | Nom d'hôte du service Redis, tel que défini dans docker-compose.yml. redis-cache est une bonne valeur. |
| REDIS\_PORT | Services utilisant Redis | Port standard de Redis. 6379. |
| JWT\_SECRET\_KEY | Services d'authentification | Clé secrète critique pour signer les jetons. Doit être générée avec openssl rand -hex 32 et être unique pour chaque développeur. |
| JWT\_ALGORITHM | Services d'authentification | Algorithme de signature. HS256 est utilisé en local. La production utilisera RS256. |
| ACCESS\_TOKEN\_EXPIRE\_MINUTES | Services d'authentification | Durée de vie d'un jeton JWT. 60 minutes est un bon compromis entre sécurité et expérience utilisateur. |
| VITE\_API\_BASE\_URL | Front-End | L'URL que le code JavaScript du navigateur utilisera pour joindre le Back-End. http://localhost:8000. |
| GCP\_PROJECT\_ID | Outils CLI, Code utilisant les SDKs GCP | L'ID de votre projet GCP de développement. Essentiel pour que les SDKs sachent à quel projet s'adresser. |
| GCS\_BUCKET\_UPLOADS | Services gérant les fichiers | Nom du bucket GCS où les fichiers sont stockés. En local, peut pointer vers un bucket de test. |

Ce modèle .env.template fournit une structure complète et "production-like" pour que chaque développeur puisse configurer un environnement local robuste et sécurisé, tout en respectant la séparation stricte des secrets entre les environnements.

#### **4.3. Commandes Unifiées pour le Lancement**

Toutes les commandes suivantes doivent être exécutées depuis la racine du monorepo.

* **Construire les Images (Première fois ou si un Dockerfile change) :**
  + Cette commande lit tous les Dockerfile de nos services et construit les images Docker locales.

docker compose build

**Démarrer l'Environnement Complet :**

* C'est la commande que vous utiliserez au quotidien. Elle démarre tous les conteneurs en arrière-plan (mode "detached").

docker compose up

**Arrêter l'Environnement :**

* Cette commande arrête et supprime proprement tous les conteneurs liés au projet.

docker compose down

* **Astuce :** Utilisez docker compose down -v pour supprimer également les volumes (comme les données de la base de données), ce qui est utile pour repartir d'un état totalement propre.

**Consulter les Logs :**

* Pour voir les logs en temps réel d'un service spécifique (très utile pour le débogage) :

# Exemple pour le user-service

docker compose logs -f user-service

#### **4.4. Vérification de l'Environnement**

Une fois que vous avez lancé docker compose up -d, voici comment vérifier que tout fonctionne :

1. **Vérifiez les Conteneurs :**
   * Exécutez docker compose ps. Vous devriez voir la liste de tous nos services avec un statut running ou up.
2. **Accédez à l'Application Front-End :**
   * Ouvrez votre navigateur et rendez-vous à l'adresse **http://localhost:5173**. La page de connexion de SkillForge AI devrait s'afficher.
3. **Accédez à la Documentation de l'API Back-End :**
   * Rendez-vous à l'adresse **http://localhost:8000/docs**. La documentation interactive de l'API (Swagger/OpenAPI) de l'un de nos services Back-End devrait être visible.
4. **(Optionnel) Se Connecter à la Base de Données :**
   * Vous pouvez utiliser un client de base de données comme DBeaver, TablePlus ou la CLI psql pour vous connecter à la base de données PostgreSQL qui tourne en local sur le port 5432 (localhost:5432), en utilisant les identifiants que vous avez définis dans votre fichier .env.

### **. La Configuration Réseau Privé Virtuel (Communication entre Conteneurs)**

Pour que les conteneurs puissent communiquer entre eux en utilisant leurs noms de service (ex: le user-service qui appelle postgres-db), nous définissons un réseau personnalisé dans Docker Compose.

**La Procédure :**

1. À la fin du fichier docker-compose.yml, nous déclarons un bloc networks.
2. Nous créons un réseau, que nous nommerons skillforge-net.
3. Dans la définition de chaque service qui a besoin de communiquer, nous ajoutons une section networks pour le rattacher à skillforge-net.

Une fois qu'ils sont sur le même réseau, Docker gère un DNS interne qui permet à chaque conteneur de résoudre les noms des autres conteneurs en adresses IP internes. C'est ce qui rend la communication transparente.

### **2. Le Montage des Volumes (Hot Reloading)**

Pour que vos modifications de code soient prises en compte instantanément sans reconstruire l'image Docker, nous utilisons le montage de volumes.

**La Procédure :**

1. Dans la définition d'un service (ex: user-service), nous ajoutons une section volumes.
2. Nous utilisons la syntaxe [CHEMIN\_SUR\_VOTRE\_MACHINE]:[CHEMIN\_DANS\_LE\_CONTENEUR].
3. Ceci a pour effet de "synchroniser" le dossier de votre machine avec le dossier à l'intérieur du conteneur. Quand vous sauvegardez un fichier dans VS Code, le fichier à l'intérieur du conteneur est instantanément mis à jour.
4. Le "Hot Reloading" est ensuite activé par un outil à l'intérieur du conteneur qui surveille ces changements de fichiers. Pour notre Back-End Python, c'est l'option --reload de uvicorn. Pour notre Front-End, c'est le serveur de développement de Vite qui s'en charge nativement.

### **Le Fichier docker-compose.yml Complet et Commenté**

Voici un exemple complet de ce à quoi ressemblerait notre fichier docker-compose.yml, intégrant tous ces concepts.

—--------

# Version de la syntaxe Docker Compose. '3.8' ou supérieure est recommandée.

version: '3.8'

services:

# --- SERVICE DE BASE DE DONNEES ---

postgres-db:

# Utilise l'image officielle de PostgreSQL version 16.

image: postgres:16

container\_name: postgres-db-skillforge

# Le nom d'hôte 'postgres-db' sera résolvable par les autres services sur le même réseau.

hostname: postgres-db

# Charge les variables d'environnement (mots de passe, etc.) depuis le fichier .env.

env\_file:

- .env

# Mappe le port 5432 du conteneur au port 5432 de votre machine locale.

ports:

- "${POSTGRES\_PORT}:5432"

# Utilise un "volume nommé" pour que les données de la BDD persistent même si le conteneur est supprimé.

volumes:

- postgres\_data:/var/lib/postgresql/data

# Rattache ce service à notre réseau personnalisé.

networks:

- skillforge-net

# Stratégie de redémarrage en cas d'échec.

restart: unless-stopped

# --- SERVICE DE CACHE ---

redis-cache:

image: redis:7-alpine

container\_name: redis-cache-skillforge

hostname: redis-cache

ports:

- "6379:6379"

networks:

- skillforge-net

restart: unless-stopped

# --- EXEMPLE DE MICROSERVICE BACK-END (ex: user-service) ---

user-service:

container\_name: user-service-skillforge

# Indique à Docker Compose de construire l'image à partir du Dockerfile situé dans ce répertoire.

build:

context: ./apps/backend/user-service

dockerfile: Dockerfile

# Charge les variables d'environnement.

env\_file:

- .env

# Mappe le port 8000 du conteneur au port 8000 de votre machine.

ports:

- "8000:8000"

# === MONTAGE DU VOLUME POUR LE HOT RELOADING ===

# Le répertoire './apps/backend/user-service' de votre machine est monté

# dans le répertoire '/app' à l'intérieur du conteneur.

volumes:

- ./apps/backend/user-service:/app

# Commande pour lancer le serveur FastAPI avec l'option --reload, qui surveille les changements de fichiers.

command: uvicorn app.main:app --host 0.0.0.0 --port 8000 --reload

# Ce service ne démarrera qu'une fois que la base de données et Redis seront prêts.

depends\_on:

- postgres-db

- redis-cache

networks:

- skillforge-net

restart: unless-stopped

# --- SERVICE FRONT-END ---

frontend:

container\_name: frontend-skillforge

build:

context: ./apps/frontend

dockerfile: Dockerfile

env\_file:

- .env

ports:

- "5173:5173"

# === MONTAGE DU VOLUME POUR LE HOT RELOADING ===

# Le principe est le même que pour le back-end.

# La deuxième ligne est une astuce pour éviter que les node\_modules locaux n'écrasent

# ceux installés dans le conteneur, ce qui pourrait causer des problèmes.

volumes:

- ./apps/frontend:/app

- /app/node\_modules

networks:

- skillforge-net

restart: unless-stopped

# --- DECLARATION DES RESSOURCES EXTERNES ---

# Déclaration de notre réseau personnalisé.

networks:

skillforge-net:

driver: bridge

# Déclaration de notre volume de données pour PostgreSQL.

volumes:

postgres\_data:

driver: local

—-

En plaçant ce fichier à la racine de votre projet et en exécutant docker compose up, vous orchestrerez un écosystème complet où chaque service peut se parler par son nom et où chaque ligne de code que vous modifiez est instantanément reflétée.

### **Partie 5 : Configuration Avancée (Optionnelle) - Reverse Proxy Nginx**

La configuration de la Partie 4, où chaque service est accessible sur un port différent (localhost:5173, localhost:8000, etc.), est parfaitement fonctionnelle pour la majorité des tâches de développement.

Cependant, cette partie s'adresse à ceux qui souhaitent une configuration plus avancée qui imite la manière dont notre application fonctionnera en production, où un seul point d'entrée (le Load Balancer) distribue le trafic en fonction de l'URL.

**Pourquoi utiliser cette configuration ?**

* **Point d'Entrée Unique :** Accéder à toute l'application via une seule adresse, http://localhost, comme le ferait un utilisateur final.
* **Routage par Chemin :** Tester de manière fiable des logiques qui dépendent du chemin de l'URL (CORS, authentification, etc.), car Nginx se chargera d'aiguiller les requêtes /api/\* vers le Back-End et le reste vers le Front-End.

#### **5.1. Fichier de Configuration Nginx**

1. À la racine de votre monorepo, créez un nouveau dossier nommé nginx.
2. À l'intérieur de ce dossier nginx, créez un fichier nommé nginx.conf.
3. Copiez le contenu suivant dans votre fichier nginx/nginx.conf :

<!-- end list -->

# nginx/nginx.conf

# Définit le contexte pour les événements réseau.

events {

worker\_connections 1024; # Nombre de connexions simultanées par worker.

}

# Définit le contexte pour le protocole HTTP.

http {

# Définit un "groupe de serveurs" pour notre Front-End.

# Le nom 'frontend' correspond au nom du service dans docker-compose.

upstream frontend {

server frontend:5173;

}

# Définit un "groupe de serveurs" pour notre Back-End.

# 'user-service' est le nom du service API dans docker-compose.

upstream backend {

server user-service:8000;

}

# Définit notre serveur principal qui écoute sur le port 80.

server {

listen 80;

# Toutes les requêtes commençant par /api/ sont redirigées vers le Back-End.

location /api/ {

proxy\_pass http://backend;

proxy\_set\_header Host $host;

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

}

# Toutes les autres requêtes (catch-all) sont redirigées vers le Front-End.

location / {

proxy\_pass http://frontend;

proxy\_set\_header Host $host;

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

}

}

}

#### **5.2. Intégration à Docker Compose**

Maintenant, ajoutons Nginx comme un nouveau service dans notre fichier docker-compose.yml. Ouvrez le fichier et ajoutez ce bloc à la liste des services :

# A ajouter dans le fichier docker-compose.yml

# --- SERVICE DE REVERSE PROXY ---

nginx-proxy:

# Utilise l'image officielle et légère de Nginx.

image: nginx:1.25-alpine

container\_name: nginx-proxy-skillforge

# Mappe le port 80 de votre machine locale au port 80 du conteneur.

# C'est désormais notre point d'entrée principal.

ports:

- "80:80"

# Monte notre fichier de configuration personnalisé à l'intérieur du conteneur.

# L'option ':ro' le monte en lecture seule (bonne pratique de sécurité).

volumes:

- ./nginx/nginx.conf:/etc/nginx/nginx.conf:ro

# S'assure que Nginx ne démarre qu'après le front et le back.

depends\_on:

- user-service

- frontend

networks:

- skillforge-net

restart: unless-stopped

Après avoir ajouté ce service, lancez docker compose up -d --build. Vous pouvez maintenant accéder à votre application en tapant simplement **http://localhost** dans votre navigateur. Nginx se chargera de la redirection.

### **Conclusion Finale du Guide**

Félicitations ! Vous avez terminé la configuration complète de votre environnement de développement pour le projet SkillForge AI.

Vous disposez maintenant d'un système robuste et conteneurisé qui vous permet de :

* Lancer l'intégralité de la plateforme avec une seule commande.
* Développer des fonctionnalités avec un rechargement à chaud instantané.
* Travailler dans un environnement standardisé et identique à celui de vos coéquipiers.
* Interagir de manière sécurisée avec les ressources cloud du projet.

Vous avez toutes les clés en main pour commencer à construire, tester et innover.

**Bon développement !**