Connexion à une base de données PostgreSQL en utilisant FastAPI et Docker Compose

Fichier : .dockerignore

```
.env
Dockerfile
docker-compose.yml
```

Fichier : .env

```
# API Configuration
API_PORT=8000

# PostgreSQL Configuration
POSTGRES_HOST=postgres
POSTGRES_PORT=5434
POSTGRES_DB=postgres
POSTGRES_USER=postgres
POSTGRES_PASSWORD=postgres
```

Fichier : docker-compose.yaml

```
services:
  postgres:
    image: postgres:latest
    container_name: postgres-server
    ports:
      - "${POSTGRES_PORT:-5434}:5434"  # Modifié pour utiliser le port 5434
    environment:
      - POSTGRES_DB=${POSTGRES_DB:-postgres}
      - POSTGRES_USER=${POSTGRES_USER:-postgres}
      - POSTGRES_PASSWORD=${POSTGRES_PASSWORD:-postgres}
    command: -p 5434  # Important: configure PostgreSQL pour écouter sur le port
5434
    volumes:
      - postgres-data:/var/lib/postgresql/data

  api:
    build: .
    container_name: fastapi-app
    ports:
      - "${API_PORT:-8000}:8000"
    depends_on:
      - postgres
    env_file:
      - .env

volumes:
  postgres-data:
```

Fichier : Dockerfile

```
FROM python:3.9-slim

WORKDIR /app

COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

COPY . .

CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]
```

Fichier : main.py

```
from fastapi import FastAPI, HTTPException
import os
from dotenv import load_dotenv
import sqlalchemy
from sqlalchemy import text

# Charger les variables d'environnement depuis le fichier .env
load_dotenv()

app = FastAPI(title="PostgreSQL Version API")

# PostgreSQL connection parameters from .env
POSTGRES_HOST = os.getenv("POSTGRES_HOST", "localhost")
POSTGRES_PORT = os.getenv("POSTGRES_PORT", "5432")
POSTGRES_USER = os.getenv("POSTGRES_USER", "postgres")
POSTGRES_PASSWORD = os.getenv("POSTGRES_PASSWORD", "postgres")
POSTGRES_DB = os.getenv("POSTGRES_DB", "postgres")

# Créer l'URL de connexion
DATABASE_URL = f"postgresql://{POSTGRES_USER}:
{POSTGRES_PASSWORD}@{POSTGRES_HOST}:{POSTGRES_PORT}/{POSTGRES_DB}"

@app.get("/")
async def root():
    return {
        "message": "API is running. Access /version to get PostgreSQL version.",
        "config": {
            "postgres_host": POSTGRES_HOST,
            "postgres_port": POSTGRES_PORT,
            "postgres_db": POSTGRES_DB,
            "postgres_user": POSTGRES_USER
        }
    }

@app.get("/version")
async def get_postgres_version():
    try:
        # Créer un moteur SQLAlchemy
        engine = sqlalchemy.create_engine(DATABASE_URL)

        # Se connecter à la base de données
        with engine.connect() as connection:
            # Exécuter une requête pour obtenir la version
            result = connection.execute(text("SELECT version()"))
            version = result.scalar()

        # Retourner la version
        return {
            "postgres_version": version,
            "connection_status": "successful",
            "connected_to": f"{POSTGRES_HOST}:{POSTGRES_PORT}"
        }
    except Exception as e:
        raise HTTPException(status_code=500, detail=f"Failed to connect to
PostgreSQL: {str(e)}")
```

Fichier : requirements.txt

```
fastapi==0.104.1
uvicorn==0.23.2
psycopg2-binary==2.9.6
sqlalchemy==2.0.19
python-dotenv==1.0.0
```

Fichier : main.py avec swagger

```
from fastapi import FastAPI, HTTPException
import os
from dotenv import load_dotenv
import sqlalchemy
from sqlalchemy import text
from pydantic import BaseModel
from typing import Dict, Any

# Charger les variables d'environnement depuis le fichier .env
load_dotenv()

# Définition de modèles pour les réponses API
class ConfigInfo(BaseModel):
```

```python
    postgres_host: str
    postgres_port: str
    postgres_db: str
    postgres_user: str
    database_url: str

class RootResponse(BaseModel):
    message: str
    config: ConfigInfo

class VersionResponse(BaseModel):
    postgres_version: str
    connection_status: str
    connected_to: str

# Création de l'application avec des métadonnées pour la documentation
app = FastAPI(
    title="PostgreSQL Version API",
    description="Une simple API pour se connecter à PostgreSQL et récupérer sa
version",
    version="1.0.0",
    contact={
        "name": "Développeur",
        "email": "dev@example.com",
    },
    license_info={
        "name": "MIT",
    },
)

# PostgreSQL connection parameters from .env
POSTGRES_HOST = os.getenv("POSTGRES_HOST", "localhost")
POSTGRES_PORT = os.getenv("POSTGRES_PORT", "5434")  # Port 5434 comme demandé
POSTGRES_USER = os.getenv("POSTGRES_USER", "postgres")
POSTGRES_PASSWORD = os.getenv("POSTGRES_PASSWORD", "postgres")
POSTGRES_DB = os.getenv("POSTGRES_DB", "postgres")

# Créer l'URL de connexion
DATABASE_URL = f"postgresql://{POSTGRES_USER}:
{POSTGRES_PASSWORD}@{POSTGRES_HOST}:{POSTGRES_PORT}/{POSTGRES_DB}"

@app.get(
    "/",
    response_model=RootResponse,
    summary="Informations de base",
    description="Retourne un message de bienvenue et la configuration actuelle de
connexion à PostgreSQL."
)
async def root():
    return {
        "message": "API is running. Access /version to get PostgreSQL version.",
        "config": {
            "postgres_host": POSTGRES_HOST,
            "postgres_port": POSTGRES_PORT,
            "postgres_db": POSTGRES_DB,
            "postgres_user": POSTGRES_USER,
            "database_url": DATABASE_URL
        }
    }

@app.get(
    "/version",
    response_model=VersionResponse,
    summary="Version de PostgreSQL",
    description="Se connecte au serveur PostgreSQL et récupère sa version.",
    responses={
        200: {
            "description": "Version récupérée avec succès",
            "content": {
                "application/json": {
                    "example": {
                        "postgres_version": "PostgreSQL 15.3 on x86_64-pc-linux-
gnu, compiled by gcc (GCC) 10.2.1 20210110, 64-bit",
                        "connection_status": "successful",
                        "connected_to": "postgres:5434"
                    }
                }
            }
        },
        500: {
            "description": "Erreur de connexion à PostgreSQL",
            "content": {
                "application/json": {
                    "example": {
                        "detail": "Failed to connect to PostgreSQL: Connection
refused"
                    }
                }
            }
        }
    }
)
async def get_postgres_version():
    try:
        # Afficher les informations de connexion pour le débogage
        print(f"Connecting to PostgreSQL at {POSTGRES_HOST}:{POSTGRES_PORT}")

        # Créer un moteur SQLAlchemy
        engine = sqlalchemy.create_engine(DATABASE_URL)

        # Se connecter à la base de données
        with engine.connect() as connection:
            # Exécuter une requête pour obtenir la version
            result = connection.execute(text("SELECT version()"))
            version = result.scalar()
        # Retourner la version
        return {
            "postgres_version": version,
            "connection_status": "successful",
            "connected_to": f"{POSTGRES_HOST}:{POSTGRES_PORT}"
        }
    except Exception as e:
        raise HTTPException(status_code=500, detail=f"Failed to connect to 
PostgreSQL: {str(e)}")
```

Connexion à une base de données ClickHouse en utilisant FastAPI, Python et Docker Compose

Fichier : .env

```
# API Configuration
API_PORT=8000

# ClickHouse Configuration
CLICKHOUSE_HOST=clickhouse
CLICKHOUSE_HTTP_PORT=8123
CLICKHOUSE_TCP_PORT=9000
CLICKHOUSE_DB=keyce_db
CLICKHOUSE_USER=keyce_user
CLICKHOUSE_PASSWORD=keyce_2025_password
```

Fichier : .dockerignore

```
.env
Dockerfile
docker-compose.yml
```

Fichier : requirements.txt

```
fastapi==0.104.1
uvicorn==0.23.2
clickhouse-driver==0.2.6
python-dotenv==1.0.0
```

Fichier : docker-compose.yaml

```
services:
  clickhouse:
    image: clickhouse/clickhouse-server:latest
    container_name: clickhouse-server
    ports:
      - "${CLICKHOUSE_HTTP_PORT:-8123}:8123"
      - "${CLICKHOUSE_TCP_PORT:-9000}:9000"
    environment:
      - CLICKHOUSE_DB=${CLICKHOUSE_DB:-default}
      - CLICKHOUSE_USER=${CLICKHOUSE_USER:-default}
      - CLICKHOUSE_PASSWORD=${CLICKHOUSE_PASSWORD:-default}

  api:
    build: .
    container_name: fastapi-app
    ports:
      - "${API_PORT:-8000}:8000"
    depends_on:
      - clickhouse
    env_file:
      - .env
```

Fichier : Dockerfile

```
FROM python:3.9-slim

WORKDIR /app

COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

COPY . .

CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]
```

Fichier : main.py

```
from fastapi import FastAPI, HTTPException
from clickhouse_driver import Client
import os
from dotenv import load_dotenv

# Charger les variables d'environnement depuis le fichier .env
load_dotenv()

app = FastAPI(title="ClickHouse Version API")

# ClickHouse connection parameters from .env
CLICKHOUSE_HOST = os.getenv("CLICKHOUSE_HOST", "localhost")
CLICKHOUSE_PORT = os.getenv("CLICKHOUSE_TCP_PORT", "9000")
CLICKHOUSE_USER = os.getenv("CLICKHOUSE_USER", "default")
CLICKHOUSE_PASSWORD = os.getenv("CLICKHOUSE_PASSWORD", "default")
CLICKHOUSE_DB = os.getenv("CLICKHOUSE_DB", "default")

@app.get("/")
async def root():
    return {
        "message": "API is running. Access /version to get ClickHouse version.",
        "config": {
            "clickhouse_host": CLICKHOUSE_HOST,
            "clickhouse_port": CLICKHOUSE_PORT,
            "clickhouse_db": CLICKHOUSE_DB,
            "clickhouse_user": CLICKHOUSE_USER
        }
    }

@app.get("/version")
async def get_clickhouse_version():
    try:
        # Create a ClickHouse client
        client = Client(
            host=CLICKHOUSE_HOST,
            port=int(CLICKHOUSE_PORT),
            user=CLICKHOUSE_USER,
            password=CLICKHOUSE_PASSWORD,
            database=CLICKHOUSE_DB
        )

        # Query ClickHouse version
        result = client.execute("SELECT version()")

        # Return the version
        return {
            "clickhouse_version": result[0][0],
            "connection_status": "successful",
            "connected_to": f"{CLICKHOUSE_HOST}:{CLICKHOUSE_PORT}"
        }
    except Exception as e:
        raise HTTPException(status_code=500, detail=f"Failed to connect to
ClickHouse: {str(e)}")
```

Fichier : main.py avec swagger

```
from fastapi import FastAPI, HTTPException
from clickhouse_driver import Client
import os
from dotenv import load_dotenv
from pydantic import BaseModel
from typing import Dict, Any

# Charger les variables d'environnement depuis le fichier .env
load_dotenv()

# Définition de modèles pour les réponses API
class ConfigInfo(BaseModel):
    clickhouse_host: str
    clickhouse_port: str
```

```python
    clickhouse_db: str
    clickhouse_user: str

class RootResponse(BaseModel):
    message: str
    config: ConfigInfo

class VersionResponse(BaseModel):
    clickhouse_version: str
    connection_status: str
    connected_to: str

# Création de l'application avec des métadonnées pour la documentation
app = FastAPI(
    title="ClickHouse Version API",
    description="Une simple API pour se connecter à ClickHouse et récupérer sa
version",
    version="1.0.0",
    contact={
        "name": "Développeur",
        "email": "dev@example.com",
    },
    license_info={
        "name": "MIT",
    },
)

# ClickHouse connection parameters from .env
CLICKHOUSE_HOST = os.getenv("CLICKHOUSE_HOST", "localhost")
CLICKHOUSE_PORT = os.getenv("CLICKHOUSE_TCP_PORT", "9000")
CLICKHOUSE_USER = os.getenv("CLICKHOUSE_USER", "default")
CLICKHOUSE_PASSWORD = os.getenv("CLICKHOUSE_PASSWORD", "default")
CLICKHOUSE_DB = os.getenv("CLICKHOUSE_DB", "default")

@app.get(
    "/",
    response_model=RootResponse,
    summary="Informations de base",
    description="Retourne un message de bienvenue et la configuration actuelle de
connexion à ClickHouse."
)
async def root():
    return {
        "message": "API is running. Access /version to get ClickHouse version.",
        "config": {
            "clickhouse_host": CLICKHOUSE_HOST,
            "clickhouse_port": CLICKHOUSE_PORT,
            "clickhouse_db": CLICKHOUSE_DB,
            "clickhouse_user": CLICKHOUSE_USER
        }
    }

@app.get(
    "/version",
    response_model=VersionResponse,
    summary="Version de ClickHouse",
    description="Se connecte au serveur ClickHouse et récupère sa version.",
    responses={
        200: {
            "description": "Version récupérée avec succès",
            "content": {
                "application/json": {
                    "example": {
                        "clickhouse_version": "23.8.1.94",
                        "connection_status": "successful",
                        "connected_to": "clickhouse:9000"
                    }
                }
            }
        },
        500: {
            "description": "Erreur de connexion à ClickHouse",
            "content": {
                "application/json": {
                    "example": {
                        "detail": "Failed to connect to ClickHouse: Connection
refused"
                    }
                }
            }
        }
    }
)
async def get_clickhouse_version():
    try:
        # Create a ClickHouse client
        client = Client(
            host=CLICKHOUSE_HOST,
            port=int(CLICKHOUSE_PORT),
            user=CLICKHOUSE_USER,
            password=CLICKHOUSE_PASSWORD,
            database=CLICKHOUSE_DB
        )

        # Query ClickHouse version
        result = client.execute("SELECT version()")

        # Return the version
        return {
            "clickhouse_version": result[0][0],
            "connection_status": "successful",
            "connected_to": f"{CLICKHOUSE_HOST}:{CLICKHOUSE_PORT}"
        }
    except Exception as e:
        raise HTTPException(status_code=500, detail=f"Failed to connect to
ClickHouse: {str(e)}")
```

Connexion à une base de données PostgreSQL et en utilisant JavaScript et Docker Compose

Fichier : .dockerignore

```
node_modules
dist
.env
Dockerfile
docker-compose.yml
```

Fichier : .env

```
# API Configuration
PORT=3000

# PostgreSQL Configuration
POSTGRES_HOST=db
POSTGRES_PORT=5434 # Modifié le port à 5433
POSTGRES_USER=postgres
POSTGRES_PASSWORD=postgres
POSTGRES_DB=postgres
```

Fichier : docker-compose.yaml

```yaml
services:
  api:
    build: .
    ports:
      - "3000:3000"
    depends_on:
      - db
    env_file:
      - .env
  db:
    image: postgres:latest
    container_name: postgres-server
    ports:
      - "${POSTGRES_PORT:-5434}:5434"  # Modifié pour utiliser le port 5434
    environment:
      - POSTGRES_DB=${POSTGRES_DB:-postgres}
      - POSTGRES_USER=${POSTGRES_USER:-postgres}
      - POSTGRES_PASSWORD=${POSTGRES_PASSWORD:-postgres}
    command: -p 5434  # Important: configure PostgreSQL pour écouter sur le port
5434
    volumes:
      - postgres-data:/var/lib/postgresql/data
volumes:
  postgres-data:
```

Fichier : Dockerfile

```
FROM node:18
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
EXPOSE 3000
CMD ["node", "index.js"]
```

Fichier : index.js

```
import express from "express";
import dotenv from "dotenv";
import pkg from 'pg';

const { Pool } = pkg;

dotenv.config();

const app = express();
const port = process.env.PORT || 3000;

// Configuration PostgreSQL
const POSTGRES_HOST = process.env.POSTGRES_HOST;
const POSTGRES_PORT = parseInt(process.env.POSTGRES_PORT, 10); // Convertir le
port en nombre
const POSTGRES_USER = process.env.POSTGRES_USER;
const POSTGRES_PASSWORD = process.env.POSTGRES_PASSWORD;
const POSTGRES_DB = process.env.POSTGRES_DB;

const pool = new Pool({
    host: POSTGRES_HOST,
    port: POSTGRES_PORT,
    user: POSTGRES_USER,
    password: POSTGRES_PASSWORD,
    database: POSTGRES_DB,
});

app.get("/", (req, res) => {
    res.json({
        message: "API is running. Access /version to get PostgreSQL version.",
        config: {
            postgres_host: POSTGRES_HOST,
            postgres_port: POSTGRES_PORT,
            postgres_db: POSTGRES_DB,
            postgres_user: POSTGRES_USER,
        }
    });
});

app.get("/version", async (req, res) => {
    try {
        const client = await pool.connect();
        const result = await client.query("SELECT version()");
        client.release();

        res.json({
            postgres_version: result.rows[0].version,
            connection_status: "successful",
            connected_to: `${POSTGRES_HOST}:${POSTGRES_PORT}`
        });
    } catch (error) {
        res.status(500).json({ error: `Failed to connect to PostgreSQL:
${error.message}` });
    }
});

app.listen(port, () => {
    console.log(`Server is running on port ${port}`);
});
```

Fichier : package.json

```
{
  "name": "javascript_api",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "type": "module",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "dotenv": "^16.4.7",
    "express": "^4.21.2",
    "pg": "^8.13.3",
    "swagger-ui-express": "^5.0.1",
    "yamljs": "^0.3.0"
  },
  "devDependencies": {
    "@types/express": "^5.0.0",
    "@types/node": "^22.13.10"
  }
}
```

Fichier : index.js avec swagger

```
import express from "express";
import dotenv from "dotenv";
import pkg from 'pg';
import swaggerUi from 'swagger-ui-express';
import YAML from 'yamljs';

// Charger le fichier Swagger
const swaggerDocument = YAML.load('./swagger.yaml');

const { Pool } = pkg;

dotenv.config();

const app = express();
const port = process.env.PORT || 3000;

// Configuration PostgreSQL
const POSTGRES_HOST = process.env.POSTGRES_HOST;
const POSTGRES_PORT = parseInt(process.env.POSTGRES_PORT, 10); // Convertir le
port en nombre
const POSTGRES_USER = process.env.POSTGRES_USER;
const POSTGRES_PASSWORD = process.env.POSTGRES_PASSWORD;
const POSTGRES_DB = process.env.POSTGRES_DB;

const pool = new Pool({
    host: POSTGRES_HOST,
    port: POSTGRES_PORT,
    user: POSTGRES_USER,
    password: POSTGRES_PASSWORD,
    database: POSTGRES_DB,
});

// Serve Swagger UI
app.use("/api-docs", swaggerUi.serve, swaggerUi.setup(swaggerDocument));

app.get("/", (req, res) => {
    res.json({
        message: "API is running. Access /version to get PostgreSQL version.",
        config: {
            postgres_host: POSTGRES_HOST,
            postgres_port: POSTGRES_PORT,
            postgres_db: POSTGRES_DB,
            postgres_user: POSTGRES_USER,
        }
    });
});

app.get("/version", async (req, res) => {
    try {
        const client = await pool.connect();
        const result = await client.query("SELECT version()");
        client.release();

        res.json({
            postgres_version: result.rows[0].version,
            connection_status: "successful",
            connected_to: `${POSTGRES_HOST}:${POSTGRES_PORT}`
        });
    } catch (error) {
        res.status(500).json({ error: `Failed to connect to PostgreSQL:
${error.message}` });
    }
});

app.listen(port, () => {
    console.log(`Server is running on port ${port}`);
});
```

Fichier : swagger.yaml

```
openapi: 3.0.0
info:
  title: PostgreSQL API
  description: API pour interagir avec PostgreSQL et obtenir la version.
  version: 1.0.0
servers:
  - url: http://localhost:3000
paths:
  /:
    get:
      summary: "Get API information"
      responses:
        '200':
          description: "API information retrieved successfully"
          content:
            application/json:
              schema:
                type: object
                properties:
                  message:
                    type: string
                  config:
                    type: object
```

```yaml
                    properties:
                      postgres_host:
                        type: string
                      postgres_port:
                        type: integer
                      postgres_db:
                        type: string
                      postgres_user:
                        type: string
  /version:
    get:
      summary: "Get PostgreSQL version"
      responses:
        '200':
          description: "PostgreSQL version retrieved successfully"
          content:
            application/json:
              schema:
                type: object
                properties:
                  postgres_version:
                    type: string
                  connection_status:
                    type: string
                  connected_to:
                    type: string
        '500':
          description: "Failed to connect to PostgreSQL"
          content:
            application/json:
              schema:
                type: object
                properties:
                  error:
                    type: string
```

Lancer l'application avec Docker Compose

Vous pouvez démarrer votre environnement en utilisant Docker Compose.

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

7. Conclusion

- Vous avez appris à configurer un environnement Docker avec PostgreSQL, ClickHouse et FastAPI.

- Vous avez vu comment interagir avec ces bases de données via des API FastAPI et comment utiliser JavaScript pour interroger ces API.
- Cette architecture vous permettra de développer des applications robustes avec une gestion efficace des données provenant de PostgreSQL et ClickHouse.
