AWS Migration App Setup Guide

This document provides a step-by-step guide to set up the AWS Migration App on a local system for development and testing. It covers the tech stack, prerequisites, and instructions for configuring the backend, frontend, and PostgreSQL database using pgAdmin.

Overview

The AWS Migration App is a full-stack application designed to manage and guide AWS account migrations. The frontend provides a user-friendly interface, while the backend handles API logic and AWS interactions, with data stored in a PostgreSQL database.

Tech Stack

- **Frontend**:
  - React (TypeScript)
  - Vite (build tool and development server)
  - Tailwind CSS (styling)
  - Axios (HTTP requests)
  - React Router (routing)
  - Framer Motion (animations)
- **Backend**:
  - Python
  - FastAPI (web framework)
  - SQLAlchemy (ORM for database interactions)
  - boto3 (AWS SDK for Python)
- **Database**: PostgreSQL (managed via pgAdmin)

Prerequisites

Before starting, ensure the following are installed on your system:

1. **Node.js** (version 16 or higher) and **npm**
   - Download from nodejs.org.
   - Verify installation: `node --version` and `npm --version`.
2. **Python** (version 3.9 or higher) and **pip**
   - Download from python.org.
   - Verify installation: `python --version` and `pip --version`.
3. **PostgreSQL** (version 13 or higher) and **pgAdmin**
   - Download PostgreSQL from postgresql.org.
   - Download pgAdmin from pgadmin.org if not included with PostgreSQL.
   - Verify PostgreSQL is running (e.g., via Services on Windows or `sudo systemctl status postgresql` on Mac/Linux).

4. **Git**
   - Download from git-scm.com.
   - Verify installation: `git --version`.
5. **AWS Credentials**
   - Obtain an AWS Access Key ID and Secret Access Key from the AWS IAM console with permissions for services like RAM, GuardDuty, Cost Explorer, and IAM.
   - Keep these credentials secure and ready for configuration.

Backend Setup

The backend is a FastAPI application that handles API requests, interacts with AWS services, and stores data in a PostgreSQL database.

Step 1: Clone the Repository

Clone the project repository to your local system:

```
git clone repo url
cd server/Backend
```

Note: Replace `<repository-url>` with the actual repo URL.

Step 2: Create and Activate a Virtual Environment

Create a Python virtual environment to isolate dependencies:

```
python -m venv venv
```

Activate the virtual environment:

On Windows:
```
venv\Scripts\activate
```

On Mac/Linux:
```
source venv/bin/activate
```

Verify activation by seeing `(venv)` in your terminal prompt.

Step 3: Install Python Dependencies

Install the required Python packages listed in `requirements.txt`:

```
pip install -r requirements.txt
```

This installs FastAPI, SQLAlchemy, boto3, and other dependencies.

Step 4: Set Up PostgreSQL with pgAdmin

Configure the PostgreSQL database using pgAdmin for management:

1. **Launch pgAdmin**:
    - Open pgAdmin (installed with PostgreSQL or separately).
    - Connect to your PostgreSQL server using the `postgres` user and the password set during PostgreSQL installation.
2. **Create the Database**:
    - In pgAdmin's left sidebar, right-click **Databases** and select **Create > Database**.
    - Name the database `aws_migration` and click **Save**.

Step 5: Configure Environment Variables

Create a `.env` file in the `server/Backend` directory (or copy `.env.example` if provided) with the following content:

```
DATABASE_URL = url for postgres
POSTGRES_USER=user(postgres)
POSTGRES_PASSWORD=password
POSTGRES_HOST=localhost
POSTGRES_PORT=5432
POSTGRES_DB=aws_migration
```

Default profile config:

```
AWS_PROFILE=aws_profile
AWS_REGION=region
AWS_ACCESS_KEY_ID=aws_access_key
AWS_SECRET_ACCESS_KEY=your_aws_secret_key
AWS_SESSION_TOKEN=yoursessiontoken
```

- Replace `password` with the password.
- Replace `your_aws_access_key` and `your_aws_secret_key` with your AWS credentials.
- Ensure `region` matches your intended AWS region (e.g., `us-east-1`).

Step 6: Run Database Migrations

The project uses a migration script (`app/db/PG.py`) to set up the database schema. Run it:

```
python app/db/migrations.py
```

- Verify in pgAdmin that tables (e.g., for accounts, migration steps) are created in the `aws_migration` database under **Schemas > public > Tables**.
- If the script fails, check the `DATABASE_URL` in the `.env` file and ensure PostgreSQL is running.

Step 7: Start the Backend Server

Launch the FastAPI server:

```
uvicorn main:app --reload
```

- The API will be available at `http://localhost:8000`.
- The `--reload` flag enables auto-reloading for development, so changes to the backend code are reflected automatically.

Frontend Setup

The frontend is a React application built with Vite and TypeScript, styled with Tailwind CSS.

Step 1: Navigate to the Frontend Directory

From the `server/Backend` directory, navigate to the frontend:

```
cd ../../Frontend
```

Step 2: Install Node.js Dependencies

Install the required Node.js packages:

```
npm install
```

Step 3: Configure Environment Variables

Create a `.env` file in the `Frontend` directory (or copy `.env.example` if provided) with:

```
VITE_API_URL=http://localhost:8000
```

- This points the frontend to the backend API.
- Adjust the port if the backend is running on a different one.

Step 4: Start the Frontend Development Server

Launch the Vite development server:

```
npm run dev
```

- The app will be available at `http://localhost:5173` (or another port shown in the terminal).
- Open this URL in a browser to access the frontend.

Testing the Setup

To ensure the app is set up correctly:

1. **Access the Frontend**:
   - Open `http://localhost:5173` in a browser.
   - Log in or configure an AWS account to verify frontend-backend communication.
2. **Check the Backend**:
   - Visit `http://localhost:8000/docs` to access FastAPI's Swagger UI and test API endpoints.
3. **Verify the Database**:
   - In pgAdmin, navigate to the `aws_migration` database and check for tables under **Schemas > public > Tables**.
   - Confirm data persistence (e.g., accounts or migration steps) after interacting with the app.
4. **Test AWS Integration**:
   - Ensure the AWS credentials in the `.env` file have permissions for services used in `aws_services.py` (e.g., RAM, GuardDuty, Cost Explorer, IAM).
   - Test migration steps to confirm AWS API calls work as expected.

Troubleshooting

- **PostgreSQL Connection Issues**:
    - Verify PostgreSQL is running (e.g., via Services on Windows or `sudo systemctl status postgresql` on Mac/Linux).
    - Check the `DATABASE_URL` in `server/Backend/.env` matches your pgAdmin setup (username, password, database name, port).
- **CORS Issues**:
    - If the frontend cannot connect to the backend, ensure `main.py` includes CORS middleware allowing requests from `http://localhost:5173`.
- **AWS Credential Errors**:
    - Confirm the AWS credentials in `server/Backend/.env` are valid and have the necessary permissions.
    - Check AWS IAM policies for services like RAM, GuardDuty, and Cost Explorer.
- **Missing Tables**:
    - If tables are not created, re-run `python app/db/migrations.py` and verify the migration script's output.

Production Considerations

- **Secure Credentials**: Use a secure vault (e.g., AWS Secrets Manager) instead of `.env` files for sensitive data.
- **Backend**: Run the backend with a production server like Gunicorn (`gunicorn -w 4 -k uvicorn.workers.UvicornWorker main:app`).
- **Frontend**: Build for production with `npm run build` and serve the output (e.g., via Nginx or a static file server).
- **Database**: Use a managed PostgreSQL instance (e.g., AWS RDS) for scalability and reliability.

Last Updated: June 26, 2025