Local PostgreSQL Setup Guide

This guide will help you set up PostgreSQL for local development of the AI Event Planner application.

Overview

Important: PostgreSQL is **required** for all environments (local development, testing, and production). SQLite is no longer supported.

Prerequisites

- macOS, Linux, or Windows
- Administrator/sudo access
- Basic command line knowledge

Installation

macOS

Option 1: Using Homebrew (Recommended)

```
# Install Homebrew if not already installed
/bin/bash -c "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"

# Install PostgreSQL
brew install postgresql@15

# Start PostgreSQL service
brew services start postgresql@15

# Add PostgreSQL to PATH (add to ~/.zshrc or ~/.bash_profile)
echo 'export PATH="/opt/homebrew/opt/postgresql@15/bin:$PATH"' >>
~/.zshrc
source ~/.zshrc
```

Option 2: Using Postgres.app

1. Download from https://postgresapp.com/
2. Move to Applications folder
3. Double-click to start
4. Click "Initialize" to create a new server

Linux (Ubuntu/Debian)

```
# Update package list
sudo apt update

# Install PostgreSQL
sudo apt install postgresql postgresql-contrib

# Start PostgreSQL service
sudo systemctl start postgresql
sudo systemctl enable postgresql

# Check status
sudo systemctl status postgresql
```

Windows

Option 1: Using Official Installer

1. Download from https://www.postgresql.org/download/windows/
2. Run the installer
3. Follow the setup wizard
4. Remember the password you set for the postgres user

Option 2: Using Docker (All Platforms)

```
# Pull PostgreSQL image
docker pull postgres:15

# Run PostgreSQL container
docker run --name eventplanner-postgres \
  -e POSTGRES_PASSWORD=postgres \
  -e POSTGRES_DB=eventplanner \
  -p 5432:5432 \
  -d postgres:15

# For development database
docker run --name eventplanner-postgres-dev \
  -e POSTGRES_PASSWORD=postgres \
  -e POSTGRES_DB=eventplanner \
  -p 5432:5432 \
  -d postgres:15

# For test database (on different port)
docker run --name eventplanner-postgres-test \
  -e POSTGRES_PASSWORD=postgres \
  -e POSTGRES_DB=eventplanner_test \
  -p 5433:5432 \
  -d postgres:15
```

Database Setup

1. Access PostgreSQL

```
# macOS/Linux
psql postgres

# Windows (from Command Prompt)
psql -U postgres

# Docker
docker exec -it eventplanner-postgres psql -U postgres
```

2. Create Application Database

```
-- Create the main database
CREATE DATABASE eventplanner;

-- Create a test database
CREATE DATABASE eventplanner_test;

-- Create a dedicated user (optional but recommended)
CREATE USER eventplanner_user WITH PASSWORD 'your_secure_password';

-- Grant privileges
GRANT ALL PRIVILEGES ON DATABASE eventplanner TO eventplanner_user;
GRANT ALL PRIVILEGES ON DATABASE eventplanner_test TO eventplanner_user;

-- Exit psql
\q
```

3. Verify Connection

```
# Test connection to main database
psql -h localhost -U postgres -d eventplanner

# Test connection with custom user
psql -h localhost -U eventplanner_user -d eventplanner

# Exit
\q
```

Application Configuration

1. Copy Environment Template

```
cp .env.example .env
```

2. Update .env File

Edit `.env` and set your database URL:

```
# Using default postgres user
DATABASE_URL=postgresql://postgres:postgres@localhost:5432/eventplanner

# Or using custom user
DATABASE_URL=postgresql://eventplanner_user:your_secure_password@localho
st:5432/eventplanner
```

3. Connection String Format

```
postgresql://username:password@host:port/database
```

Components:

- **username**: PostgreSQL user (default: `postgres`)
- **password**: User's password
- **host**: Server address (default: `localhost`)
- **port**: PostgreSQL port (default: `5432`)
- **database**: Database name (e.g., `eventplanner`)

4. Additional Parameters

For SSL connections (required for some cloud providers):

```
DATABASE_URL=postgresql://user:password@host:5432/db?sslmode=require
```

Running Migrations

After setting up PostgreSQL and configuring your `.env` file:

```
# Install dependencies
pip install -r requirements.txt

# Run database migrations
alembic upgrade head
```

```
# Or using the migration script
python scripts/migrate.py
```

Verification

1. Check Database Connection

```
python scripts/test_postgres_connection.py
```

2. List Tables

```
psql -h localhost -U postgres -d eventplanner -c "\dt"
```

3. Check Application Startup

```
# Start the application
uvicorn app.main:app --reload

# Check for successful database connection in logs
# Should see: "INFO: Connecting to PostgreSQL database for development
environment"
```

Common Issues and Solutions

Issue 1: Connection Refused

Error: `connection refused` or `could not connect to server`

Solutions:

```
# Check if PostgreSQL is running
# macOS (Homebrew)
brew services list

# Linux
sudo systemctl status postgresql

# Docker
docker ps | grep postgres

# Start the service if not running
brew services start postgresql@15  # macOS
```

```
sudo systemctl start postgresql    # Linux
docker start eventplanner-postgres # Docker
```

Issue 2: Authentication Failed

Error: `password authentication failed for user`

Solutions:

1. Verify password in connection string
2. Reset password:

```
psql postgres
ALTER USER postgres PASSWORD 'new_password';
```

Issue 3: Database Does Not Exist

Error: `database "eventplanner" does not exist`

Solution:

```
createdb eventplanner
# or
psql postgres -c "CREATE DATABASE eventplanner;"
```

Issue 4: Port Already in Use

Error: `port 5432 is already in use`

Solutions:

1. Use different port in Docker:

```
docker run -p 5433:5432 ...
# Update DATABASE_URL to use port 5433
```

2. Stop conflicting service:

```
# macOS
brew services stop postgresql@15

# Linux
sudo systemctl stop postgresql
```

Issue 5: Permission Denied

Error: `permission denied for database`

Solution:

```
-- Connect as postgres user
psql postgres

-- Grant permissions
GRANT ALL PRIVILEGES ON DATABASE eventplanner TO your_user;
GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA public TO your_user;
GRANT ALL PRIVILEGES ON ALL SEQUENCES IN SCHEMA public TO your_user;
```

Testing Setup

For running tests with PostgreSQL:

1. Create test database:

   ```
   createdb eventplanner_test
   ```

2. Update `.env.test`:

   ```
   DATABASE_URL=postgresql://postgres:postgres@localhost:5432/eventpla
   nner_test
   ```

3. Run tests:

   ```
   pytest
   ```

Docker Compose (Alternative)

Create `docker-compose.yml` for easier management:

```
version: '3.8'

services:
  postgres:
    image: postgres:15
    container_name: eventplanner-postgres
    environment:
      POSTGRES_USER: postgres
```

```
      POSTGRES_PASSWORD: postgres
      POSTGRES_DB: eventplanner
    ports:
      - "5432:5432"
    volumes:
      - postgres_data:/var/lib/postgresql/data
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U postgres"]
      interval: 10s
      timeout: 5s
      retries: 5

  postgres-test:
    image: postgres:15
    container_name: eventplanner-postgres-test
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: postgres
      POSTGRES_DB: eventplanner_test
    ports:
      - "5433:5432"
    volumes:
      - postgres_test_data:/var/lib/postgresql/data

volumes:
  postgres_data:
  postgres_test_data:
```

Start services:

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

Useful PostgreSQL Commands

```
# List all databases
psql -U postgres -l

# Connect to database
psql -U postgres -d eventplanner

# Inside psql:
\l          # List databases
\dt         # List tables
\du         # List users
\c dbname   # Connect to database
\q          # Quit

# Backup database
```

```
pg_dump -U postgres eventplanner > backup.sql

# Restore database
psql -U postgres eventplanner < backup.sql

# Drop database (careful!)
dropdb eventplanner
```

Production Considerations

When deploying to production:

1. **Use strong passwords**
2. **Enable SSL/TLS connections**
3. **Set up regular backups**
4. **Monitor database performance**
5. **Use connection pooling**
6. **Set appropriate resource limits**

For Azure PostgreSQL deployment, see `DEPLOYMENT_GUIDE.md`.

Additional Resources

- PostgreSQL Official Documentation
- PostgreSQL Tutorial
- SQLAlchemy Documentation
- Alembic Documentation

Support

If you encounter issues not covered here:

1. Check application logs for detailed error messages
2. Verify PostgreSQL logs: `/usr/local/var/log/postgresql@15.log` (macOS)
3. Review Azure deployment guides for cloud setup
4. Consult the project's issue tracker

Note: SQLite is no longer supported. All environments must use PostgreSQL.