

Chapter 6: Configuring a Database

By Tim Condon

数据库允许您在应用程序中持久化数据。在本章中，您将学习如何配置Vapor应用程序以与您选择的数据库集成。

本章和本书的大部分内容都使用**Docker**来托管数据库。**Docker**是一种容器化技术，允许您在计算机上运行独立映像，而无需虚拟机的开销。您可以启动不同的数据库，而不必担心安装相互干扰的依赖项或数据库。

为什么要使用数据库？

数据库提供了一种可靠，高效的存储和检索数据的方法。如果您的应用程序将信息存储在内存中，则在您停止应用程序时它会丢失。将存储与应用程序分离是一种很好的做法，因为这样可以跨多个实例来扩展您的应用程序，所有实例都由同一个数据库支持。不过呢，大多数托管解决方案都没有持久的文件存储。

选择数据库

Vapor有官方的Swift-native驱动程序：

- SQLite
- MySQL
- PostgreSQL

有两种类型的数据库：**relational (SQL)** 数据库和**non-relational (NoSQL)** 数据库。关系型数据库将其数据存储在有已定义列的结构化表中。它们可以有效地存储和查询那些结构已知的数据。您可以使用结构化查询语言 (SQL) 创建和查询表，该语言允许您从多个相关表中检索数据。例如，如果您有一个表中的宠物清单和另一个表中的所有者清单，您可以使用单个查询检索其所有者名称的宠物清单。目前Vapor只对关系型 (SQL) 数据库提供官方支持，但这在将来会发生变化。

虽然关系型数据库适用于固定结构，但如果必须更改该结构，则可能会出现问題。最近，NoSQL数据库作为存储大量非结构化数据的方式变得流行。例如，社交网络可以将设置，图像，位置，状态和指标存储在单个文档中。这比传统数据库具有更大的灵活性。

SQLite

SQLite是一个简单的基于文件的关系数据库系统。它旨在嵌入到应用程序中，对iOS应用程序等单进程应用程序非常有用。它依赖于文件锁来维护数据库完整性，因此它不适合写密集型应用程序。这也意味着它不能跨服务器使用。然而，它是测试和原型应用程序的良好数据库。

MySQL

MySQL是另一个开源的关系数据库，受LAMP Web应用程序堆栈 (Linux, Apache, MySQL, PHP) 的欢迎。由于其易用性以及大多数云提供商和网站构建者的支持，它已成为最受欢迎的数据库。

PostgreSQL

PostgreSQL - 经常缩写为Postgres--是一个开源的关系数据库系统，专注于可扩展性和标准，专为企业使用而设计。Postgres还支持几何原语，例如坐标。Fluent支持这些原语，并将嵌套类型（如词典）直接保存到Postgres中。

配置 Vapor

所有受支持的数据库遵循相同步骤来配置Vapor应用程序以使用数据库，如下所示。

- 将该数据库的**Fluent Provider**添加到应用程序的services中。
- 配置数据库。
- 配置 **migrations**。

Services是一种从容器创建和访问事物的方式，本质上是Vapor的依赖注入提供者。您在Vapor中与之交互的最常见容器是应用程序本身，请求和响应。您应该使用该应用程序来创建启动应用程序所需的服务。如果在处理请求时要hash密码，您应该使用请求和响应容器来创建服务，例如BCryptHasher的实例。

本章中的每个数据库配置都从第5章“Fluent and Persisting Models”中留下的**TILApp**开始。您还需要安装和运行Docker。访问 <https://www.docker.com/get-docker> 并按照说明进行安装。

SQLite

Vapor Toolbox提供的默认模板使用SQLite作为其数据库。第5章中的Acronym模型使用SQLite。但是要了解它是如何工作的，请在项目目录中打开**Package.swift**。它看起来类似于以下内容：

```
// swift-tools-version:4.0
import PackageDescription

let package = Package(
    name: "TILApp",
    dependencies: [
        // 💧 A server-side Swift web framework.
```

```
.package(url: "https://github.com/vapor/vapor.git",
         from: "3.0.0"),

// ✏ Swift ORM framework (queries, models, and relations)
// for building NoSQL and SQL database integrations.
.package(url: "https://github.com/vapor/fluent-sqlite.git",
         from: "3.0.0"),
],
targets: [
    .target(name: "App", dependencies: ["FluentSQLite",
                                        "Vapor"]),
    .target(name: "Run", dependencies: ["App"]),
    .testTarget(name: "AppTests", dependencies: ["App"]),
]
)
```

您可以看到您的应用依赖于FluentSQLite。您可能想知道数据库配置发生的位置。数据库配置发生在**Sources/App/configure.swift**：

```
// 1
try services.register(FluentSQLiteProvider())

// 2
var databases = DatabasesConfig()
try databases.add(database: SQLiteDatabase(storage: .memory),
                 as: .sqlite)
services.register(databases)

// 3
var migrations = MigrationConfig()
migrations.add(model: Acronym.self, database: .sqlite)
services.register(migrations)
```

这是它的作用：

1. 将FluentSQLiteProvider注册为服务，以允许应用程序通过Fluent与SQLite交互。
2. 创建一个DatabasesConfig类型，它注册一个SQLiteDatabase实例，在整个应用程序中标识为.sqlite。请注意，这使用.memory存储。这意味着数据库驻留在内存中，它不会持久保存到磁盘，并在应用程序终止时丢失。
3. 创建一个MigrationConfig类型，告诉应用程序为每个模型使用哪个数据库，如第5章“Fluent and Persisting Models”中所述。

如果您希望使用SQLite进行持久存储，请为SQLiteDatabase提供以下路径：

```
let database =
    try SQLiteDatabase(storage: .file(path: "db.sqlite"))
databases.add(database: database, as: .sqlite)
```

如果文件不存在，则在指定路径创建数据库文件。如果文件存在，SQLiteDatabase将使用它。

MySQL

要使用MySQL进行测试，请在Docker容器中运行MySQL服务器。在终端中输入以下命令：

```
docker run --name mysql -e MYSQL_USER=vapor \
    -e MYSQL_PASSWORD=password -e MYSQL_DATABASE=vapor \
    -p 3306:3306 -d mysql/mysql-server:5.7
```

这是它的作用：

- 运行一个名为**mysql**的新容器。
- 通过环境变量指定数据库名称、用户名和密码。
- 允许应用程序在其默认端口：3306上连接到MySQL服务器。
- 作为守护程序在后台运行服务器。
- 此容器使用名为**mysql/mysql-server**的Docker镜像。如果您的计算机上没有该镜像，Docker会自动下载该镜像。这也指定了使用版本5.7标记的镜像，该版本与Fluent兼容。

要检查数据库是否正在运行，请在终端中输入以下内容以列出所有活动容器：

```
docker ps
```

```
Tims-MBP:vapor tims$ docker run --name mysql -e MYSQL_USER=vapor -e MYSQL_PASSWORD=password -e MYSQL_DATABASE=vapor -p 3306:3306 -d mysql/mysql-server:5.7
Unable to find image 'mysql/mysql-server:5.7' locally
5.7: Pulling from mysql/mysql-server
b0efbbec3b2e: Pull complete
5053dccc7425: Pull complete
16b4ec95c155: Pull complete
8b211b61b1a0: Pull complete
Digest: sha256:eb3aa80c047efcb3e6bfcc3a28b80a2ec8c67b4315712b26679b0b22320f0b4a
Status: Downloaded newer image for mysql/mysql-server:5.7
38772026388b7d967e6a3190c84491be9f254e0eaa0bb66898e79f5aa911df96
Tims-MBP:vapor tims$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
38772026388b	mysql/mysql-server:5.7	"/entrypoint.sh mysql-"	Less than a second ago	Up 25 seconds (health: starting)	0.0.0.0:3306->3306/tcp, 33060/tcp	mysql

```
Tims-MBP:vapor tims$
```

现在MySQL正在运行，请设置您的Vapor应用程序。打开**Package.swift**，用以下内容替换其内容：

```
// swift-tools-version:4.0
import PackageDescription

let package = Package(
    name: "TILApp",
    dependencies: [
        .package(url: "https://github.com/vapor/vapor.git",
            from: "3.0.0"),

        // 1
        .package(url: "https://github.com/vapor/fluent-mysql.git",
            from: "3.0.0"),
    ],
    targets: [
        // 2
        .target(name: "App", dependencies: ["FluentMySQL",
            "Vapor"]),
        .target(name: "Run", dependencies: ["App"]),
        .testTarget(name: "AppTests", dependencies: ["App"]),
    ]
)
```

这是它的作用：

1. 将FluentMySQL指定为包依赖项。
2. 指定App target依赖于FluentMySQL以确保其正确链接。

在终端中，键入以下内容以重新生成Xcode项目（如果已打开项目，请先关闭项目）并引入新的依赖项：

```
vapor xcode -y
```

当Xcode打开时，打开**configure.swift**。要切换到MySQL，请使用以下代码替换内容：

```
// 1
import FluentMySQL
import Vapor

public func configure(
    _ config: inout Config,
    _ env: inout Environment,
    _ services: inout Services
) throws {
    // 2
    try services.register(FluentMySQLProvider())

    let router = EngineRouter.default()
    try routes(router)
    services.register(router, as: Router.self)
}
```

```

var middlewares = MiddlewareConfig()
middlewares.use(ErrorMiddleware.self)
services.register(middlewares)

var databases = DatabasesConfig()
// 3
let databaseConfig = MySQLDatabaseConfig(
    hostname: "localhost",
    username: "vapor",
    password: "password",
    database: "vapor")
let database = MySQLDatabase(config: databaseConfig)
databases.add(database: database, as: .mysql)
services.register(databases)
var migrations = MigrationConfig()
// 4
migrations.add(model: Acronym.self, database: .mysql)
services.register(migrations)
}

```

变化是：

1. 导入FluentMySQL。
2. 注册FluentMySQLProvider。
3. 使用提供给Docker的相同值设置MySQL数据库配置。
4. 更改Acronym migration以使用.mysql数据库。

最后，更改Acronym模型以遵循MySQLModel协议。打开**Acronym.swift**并使用以下代码替换其内容：

```

import Vapor
import FluentMySQL

final class Acronym: Codable {
    var id: Int?
    var short: String
    var long: String

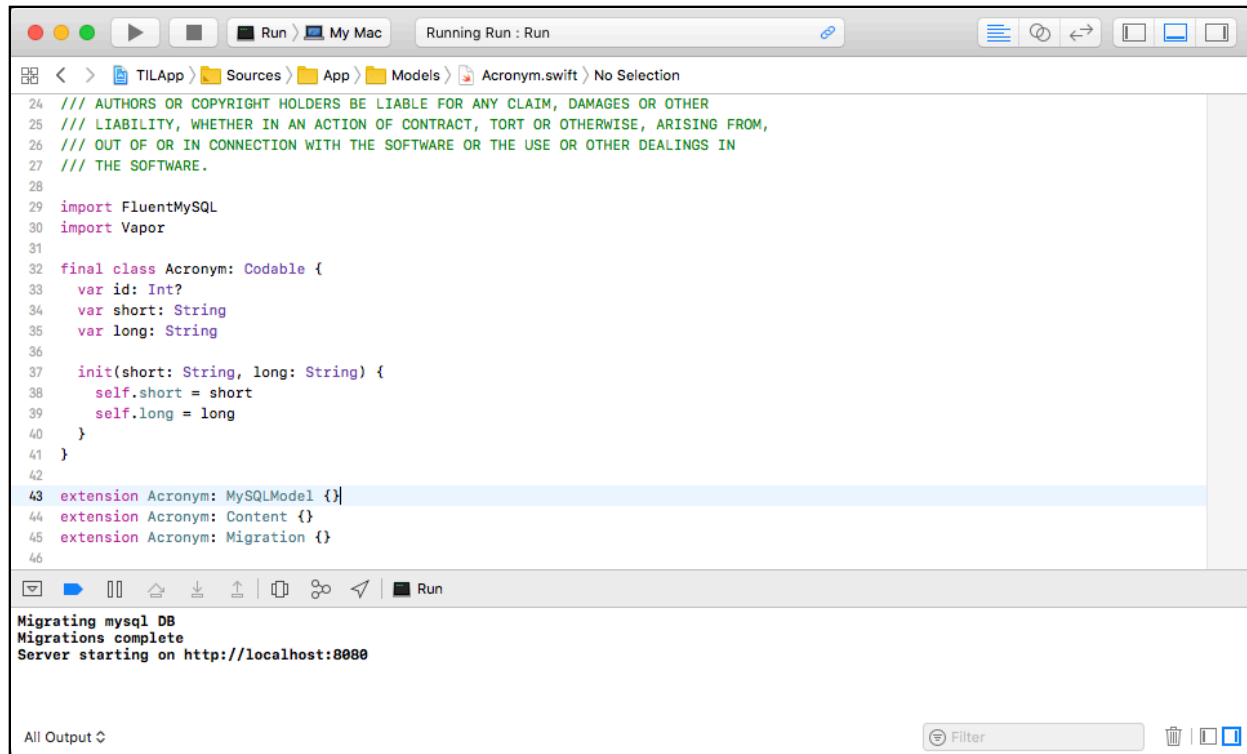
    init(short: String, long: String) {
        self.short = short
        self.long = long
    }
}

extension Acronym: MySQLModel {}
extension Acronym: Migration {}
extension Acronym: Content {}

```

确保选择了部署目标为**My Mac**的**Run**方案，然后构建并运行应用程序。

在控制台中查看migration消息。



The screenshot shows an IDE window with a Swift file named `Acronym.swift`. The code defines a `Codable` class `Acronym` with properties `id`, `short`, and `long`, and an `init` method. It also includes extensions for `MySQLModel`, `Content`, and `Migration`. The terminal at the bottom shows the output of running the application: `Migrating mysql DB`, `Migrations complete`, and `Server starting on http://localhost:8080`.

```
24 /// AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
25 /// LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
26 /// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
27 /// THE SOFTWARE.
28
29 import FluentMySQL
30 import Vapor
31
32 final class Acronym: Codable {
33     var id: Int?
34     var short: String
35     var long: String
36
37     init(short: String, long: String) {
38         self.short = short
39         self.long = long
40     }
41 }
42
43 extension Acronym: MySQLModel {}
44 extension Acronym: Content {}
45 extension Acronym: Migration {}
46
```

Migrating mysql DB
Migrations complete
Server starting on http://localhost:8080

PostgreSQL

要使用PostgreSQL进行测试，您将在Docker容器中运行Postgres服务器。打开终端并输入以下命令：

```
docker run --name postgres -e POSTGRES_DB=vapor \
-e POSTGRES_USER=vapor -e POSTGRES_PASSWORD=password \
-p 5432:5432 -d postgres
```

这是它的作用：

- 运行名为**postgres**的新容器。
- 通过环境变量指定数据库名称、用户名和密码。
- 允许应用程序在其默认端口：5432上连接到Postgres服务器。
- 作为守护程序在后台运行服务器。
- 此容器使用名为**postgres**的Docker镜像。如果您的计算机上没有该镜像，Docker会自动下载它。

要检查数据库是否正在运行，请在终端中输入以下内容以列出所有活动容器：

```
docker ps
```

```
Tims-MBP:vapor tims$ docker run --name postgres -e POSTGRES_DB=vapor -e POSTGRES_USER=vapor -e POSTGRES_PASSWORD=password -p 5432:5432 -d postgres
Unable to find image 'postgres:latest' locally
latest: Pulling from library/postgres
723254a2c089: Pull complete
39ec0e6c372c: Pull complete
ba1542fb91f3: Pull complete
c7195e642388: Pull complete
95424deca6a2: Pull complete
2d7d4b3a4ce2: Pull complete
fbde41d4a8cc: Pull complete
43a0cfa9789d: Pull complete
371d656a7cd4: Pull complete
6b98f92bd478: Pull complete
1899e8510879: Pull complete
5d421aa09a81: Pull complete
8423a5b1da74: Pull complete
Digest: sha256:92f5c1043096c56119f5d4a71a5ca382f652a4d02b814f6970c0021031422a2d
Status: Downloaded newer image for postgres:latest
882203d316180f81026d3309960a73dd778b1ec59c8042978ff58d353cdd109d
Tims-MBP:vapor tims$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
882203d31618	postgres	"docker-entrypoint.s..."	Less than a second ago	Up 4 seconds	0.0.0.0:5432->5432/tcp	postgres

```
Tims-MBP:vapor tims$
```

现在Postgres正在运行，请设置您的Vapor应用程序。打开**Package.swift**；用以下内容替换其内容：

```
// swift-tools-version:4.0
import PackageDescription

let package = Package(
    name: "TILApp",
    dependencies: [
        .package(url: "https://github.com/vapor/vapor.git",
            from: "3.0.0"),

        // 1
        .package(
            url: "https://github.com/vapor/fluent-postgresql.git",
            from: "1.0.0"),
    ],
    targets: [
        // 2
        .target(name: "App", dependencies: ["FluentPostgreSQL",
            "Vapor"]),
        .target(name: "Run", dependencies: ["App"]),
        .testTarget(name: "AppTests", dependencies: ["App"]),
    ]
)
```

这是它的作用：

1. 将FluentPostgreSQL指定为包依赖项。
2. 指定App target依赖于FluentPostgreSQL以确保其正确链接。

在终端中，键入以下内容，以重新生成Xcode项目（如果已打开项目，请先关闭项目）以引入新的依赖项：

```
vapor xcode -y
```

当Xcode打开时，打开**configure.swift**。要切换到PostgreSQL，请使用以下代码替换其内容：

```
// 1
import FluentPostgreSQL
import Vapor

public func configure(
    _ config: inout Config,
    _ env: inout Environment,
    _ services: inout Services
) throws {
    // 2
    try services.register(FluentPostgreSQLProvider())

    let router = EngineRouter.default()
    try routes(router)
    services.register(router, as: Router.self)

    var middlewares = MiddlewareConfig()
    middlewares.use(ErrorMiddleware.self)
    services.register(middlewares)

    // Configure a database
    var databases = DatabasesConfig()
    // 3
    let databaseConfig = PostgreSQLDatabaseConfig(
        hostname: "localhost",
        username: "vapor",
        database: "vapor",
        password: "password")
    let database = PostgreSQLDatabase(config: databaseConfig)
    databases.add(database: database, as: .psql)
    services.register(databases)

    var migrations = MigrationConfig()
    // 4
    migrations.add(model: Acronym.self, database: .psql)
    services.register(migrations)
}
```

变化是：

1. 导入 FluentPostgreSQL。
2. 注册 FluentPostgreSQLProvider。
3. 使用提供给Docker的相同值设置PostgreSQL数据库配置。

4. 更改Acronym migration以使用.psql数据库。

最后，更改Acronym模型以遵循PostgreSQLModel协议。打开**Acronym.swift**，并使用以下代码替换其内容：

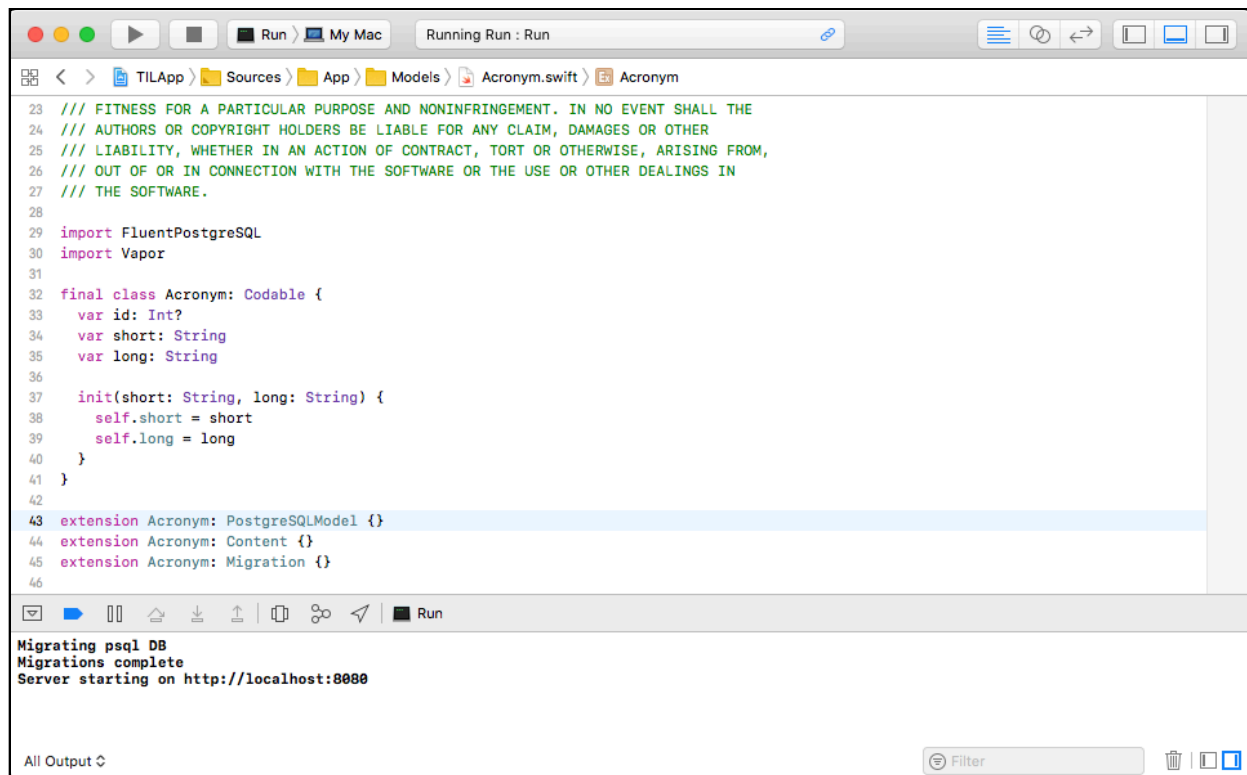
```
import Vapor
import FluentPostgreSQL

final class Acronym: Codable {
    var id: Int?
    var short: String
    var long: String

    init(short: String, long: String) {
        self.short = short
        self.long = long
    }
}

extension Acronym: PostgreSQLModel {}
extension Acronym: Migration {}
extension Acronym: Content {}
```

确保选择了部署目标为**My Mac**的**Run**方案，然后构建并运行应用程序。在控制台中查看migration消息。



The screenshot shows an IDE window with the file path `TILApp > Sources > App > Models > Acronym.swift`. The code in the editor is the same as shown in the previous block. The console at the bottom displays the following output:

```
Migrating psql DB
Migrations complete
Server starting on http://localhost:8080
```

然后去哪儿？

在本章中，您学习了如何为应用程序配置数据库。下一章介绍CRUD操作，以便您可以创建，检索，更新和删除您的acronym表数据。