

Storing Stuff

Software Architecture

March 14, 2022

Brae Webb



Figure 1: A map of data storage techniques from Designing Data-Intensive Applications [1].

Warning

This document is still a work in progress.

1 This Week

This week our goal is to:

- explore the various techniques developers use to store data; and
- look at the storage options implementing these techniques on the AWS platform.

- run a small application using docker that requires a database.
- deploy a small application that requires a database in AWS using Terraform.

2 Introduction

Unfortunately, to build interesting software we often need to store and use data. The storage of data introduces a number of challenges to designing, creating, and maintaining our software. However, not all data storage techniques are created equal; the choice of data storage model can have a profound impact on our software's complexity and maintainability. In this practical, we want to take a superficial exploration of our island of data storage models. For a more in-depth treatment of data storage models that is outside the scope of this course, see the *Designing Data-Intensive Applications* book [1].

3 Relational Storage

- MySQL/MariaDB [Amazon RDS / Amazon Aurora].
- Postgres [Amazon RDS / Amazon Aurora].

3.1 ORM

Just mentioning the relational-object mismatch.

4 Wide-Column Storage

- Apache Cassandra [Amazon Keyspaces for Cassandra].
- Apache HBase.

5 Key-Value Storage

- Redis [Amazon ElastiCache for Redis].
- Memcached [Amazon ElastiCache for Memcached].
- Amazon DynamoDB.
- Amazon MemoryDB for Redis.

6 Time Series Storage

- Amazon Timestream.
- TimescaleDB (Postgres + Addon).
- Prometheus.

7 Document Storage

- MongoDB.
- Apache CouchDB.
- Amazon DocumentDB.

8 Graph Storage

- Amazon Neptune.
- Neo4J.
- Janus Graph.

9 Working with Docker

So far in the course we have introduced docker as a means to package software to make it easier to work with and deploy. Today we will be using it to run a small application locally that is a webserver + a database.

Info

You will need to have docker and docker-compose installed for this practical. Installation will depend on your operating system.

- docker compose: <https://docs.docker.com/compose/install/>
- docker engine: <https://docs.docker.com/get-docker/>

We also recommend installing the vscode docker plugin or the equivalent tools in IntelliJ IDEs.

Notice

For terminal examples in this section, lines that begin with a \$ indicate a line which you should type while the other lines are example output that you should expect. Not all of the output is captured in the examples to save on space.

9.1 Locally

We will be using a container that is built from the Dockerfile described below which can be found here: <https://github.com/CSSE6400/todo-app/blob/main/backend/Dockerfile>. □

```
» cat Dockerfile
```

```
1 FROM ubuntu:21.10
2 RUN apt-get update \
3     && DEBIAN_FRONTEND=noninteractive apt install -y \
4         php \
5         php-mysql \
```

```

6         php-xml \
7         php-curl \
8         curl \
9         git \
10        unzip
11 RUN curl -sS https://getcomposer.org/installer | php -- --install-dir=/usr/local/bin
    --filename=composer
12 COPY . /app
13 WORKDIR /app
14 RUN composer install
15 CMD ["php", "artisan", "serve", "--host=0.0.0.0"]

```

```

    » cat main.tf
1 version: '3.3'
2 services:
3     backend:
4         image: ghcr.io/csse6400/todo-app:latest
5         ports:
6             - '8000:8000'
7         environment:
8             APP_ENV: 'local'
9             APP_KEY: 'base64:8PQEPYG1Tm1t3aqWmlAw/ZPwCiIFvdXDBjk3mhsom/A='
10            APP_DEBUG: 'true'
11            LOG_LEVEL: 'debug'

```

```

$ docker-compose up
Creating network "p1_default" with the default driver
Creating p1_backend_1 ... done
Attaching to p1_backend_1
backend_1 | Starting Laravel development server: http://0.0.0.0:8000
backend_1 | [Sun Mar 20 07:56:23 2022] PHP 8.0.8 Development Server (http
    ://0.0.0.0:8000) started

```

SQLSTATE[HY000] [2002] Connection refused

```
select count(*) as aggregate from `todos`
```

↕ Expand vendor frames

15 vendor frames ▾

App \ Http \ Controllers \ TodoController : 17
index

36 vendor frames ▾

public / **Index.php** : 52
require_once

app / Http / Controllers / **TodoController.php** : 17

```
2
3 namespace App\http\controllers;
4
```

» cat main.tf

```
1 version: '3.3'
2 services:
3   db:
4     image: mysql:8-debian
5     environment:
6       MYSQL_DATABASE: 'todoapp'
7       MYSQL_USER: 'todoapp'
8       MYSQL_PASSWORD: 'password'
9       MYSQL_ROOT_PASSWORD: 'password'
10    ports:
11      - '3306:3306'
12
13   backend:
14     image: ghcr.io/csse6400/todo-app:latest
15     depends_on:
16       - db
17     ports:
18       - '8000:8000'
19     environment:
```

```
20 APP_ENV: 'local'
21 APP_KEY: 'base64:8PQEPYGlTm1t3aqWmlAw/ZPwCiIFvdXDBjk3mhsom/A='
22 APP_DEBUG: 'true'
23 LOG_LEVEL: 'debug'
24 DB_CONNECTION: 'mysql'
25 DB_HOST: 'db'
26 DB_PORT: '3306'
27 DB_DATABASE: 'todoapp'
28 DB_USERNAME: 'todoapp'
29 DB_PASSWORD: 'password'
```

```
$ docker-compose up
Starting p2_db_1 ... done
Starting p2_backend_1 ... done
Attaching to p2_db_1, p2_backend_1
db_1 | 2022-03-20 08:11:55+00:00 [Note] [Entrypoint]: Entrypoint ....
db_1 | 2022-03-20 08:11:55+00:00 [Note] [Entrypoint]: Switching t....
db_1 | 2022-03-20 08:11:55+00:00 [Note] [Entrypoint]: Entrypoint ....
db_1 | 2022-03-20T08:11:55.438996Z 0 [System] [MY-010116] [Server....
db_1 | 2022-03-20T08:11:55.445261Z 1 [System] [MY-013576] [InnoDB....
backend_1 | Starting Laravel development server: http://0.0.0.0:8000
db_1 | 2022-03-20T08:11:55.535803Z 1 [System] [MY-013577] [InnoDB....
db_1 | 2022-03-20T08:11:55.673757Z 0 [Warning] [MY-010068] [Serve....
db_1 | 2022-03-20T08:11:55.673784Z 0 [System] [MY-013602] [Server....
db_1 | 2022-03-20T08:11:55.674810Z 0 [Warning] [MY-011810] [Serve....
db_1 | 2022-03-20T08:11:55.684729Z 0 [System] [MY-010931] [Server....
db_1 | 2022-03-20T08:11:55.684756Z 0 [System] [MY-011323] [Server....
backend_1 | [Sun Mar 20 08:11:55 2022] PHP 8.0.8 Development Serv....
```

TODO: Going to 127.0.0.1:8000/api/v1/todo will show an error page

SQLSTATE[42S02]: Base table or view not found: 1146 Table 'todoapp.todos' doesn't exist

```
select count(*) as aggregate from `todos`
```

A table was not found[🔧 RUN MIGRATIONS](#)

You might have forgotten to run your database migrations.

You can try to run your migrations using `php artisan migrate`.

[Database: Running Migrations docs](#)

[↕ Expand vendor frames](#)

13 vendor frames ▾

App\Http\Controllers\TodoController:17
index

```
$ docker-compose exec backend php artisan migrate:fresh --seed
Dropped all tables successfully.
Migration table created successfully.
Migrating: 2022_03_19_041557_create_todos_table
Migrated: 2022_03_19_041557_create_todos_table (7.55ms)
Seeding: Database\Seeders\TodoSeeder
Seeded: Database\Seeders\TodoSeeder (6.56ms)
Database seeding completed successfully.
```

9.1.1 Exercise: Migrations at startup

So far when we run this application we have to perform the database migrations manually. To help us get up and running we are going to make a small modification to pre run the migrations when the web app starts. First we need to have a look at how the container is set to launch by default. In the Dockerfile attached at the start of the prac we see that we have defined the command to run on the last line with the CMD directive.

```

» cat Dockerfile
1 FROM ubuntu:21.10
2 ...
3 ...
4 ...
5 CMD ["php", "artisan", "serve", "--host=0.0.0.0"]

```

Info

When working with docker it can get confusing around the networking aspects. In this application I have specified that the server must listen on all network interfaces (0.0.0.0). Without this flag the default is 127.0.0.1 which even though its the localhost the forwarded traffic through the docker container would never reach it.

This command launches the laravel development server and listens on all interfaces on the host. We are going to override this in our docker-compose file so that we run the migrations then start the server. Add the following line to the docker-compose.yml that you have been developing during the prac.

```

1 command: sh -c "sleep 30 && php artisan migrate:refresh --seed && php artisan serve
    --host=0.0.0.0"

```

This new command does the following:

- Waits for the database to be ready in a simple way.
- Runs the migrations and seeds the database, as we have seen earlier.
- Starts the development server as the container originally did.

Example: redacted version of goal docker-compose.yml attached below.

```

» cat docker-compose.yml
1 version: '3.3'
2 services:
3   db:
4     ...
5
6   backend:
7     ...
8     environment:
9       ...
10    command: sh -c "sleep 10 && php artisan migrate:refresh --seed && php artisan
    serve --host=0.0.0.0"

```

Now when we launch the docker-compose we can see that our migrations were run in the output.


```
$ docker-compose up
...
...
backend_1 | Rolling back: 2022_03_19_041557_create_todos_table
backend_1 | Rolled back: 2022_03_19_041557_create_todos_table (8.28ms)
backend_1 | Migrating: 2022_03_19_041557_create_todos_table
backend_1 | Migrated: 2022_03_19_041557_create_todos_table (11.55ms)
backend_1 | Seeding: Database\Seeders\TodoSeeder
backend_1 | Seeded: Database\Seeders\TodoSeeder (44.77ms)
backend_1 | Database seeding completed successfully.
backend_1 | Starting Laravel development server: http://0.0.0.0:8000
backend_1 | [Sun Mar 20 12:08:41 2022] PHP 8.0.8 Development Server (http
: //0.0.0.0:8000) started
```

We can also bake this into the container by extending the original, it is fairly common to see projects in the wild that run a init script when the container launches. An exercise left for the reader is to build upon the provided docker container but create an init script.

9.2 AWS

Warning

This section is still being developed.

References

- [1] M. Kleppmann, *Designing Data-Intensive Applications: The big ideas behind reliable, scalable, and maintainable systems*. O'Reilly Media, Inc., March 2017.