Storing Stuff

Software Architecture

March 14, 2022

Brae Webb

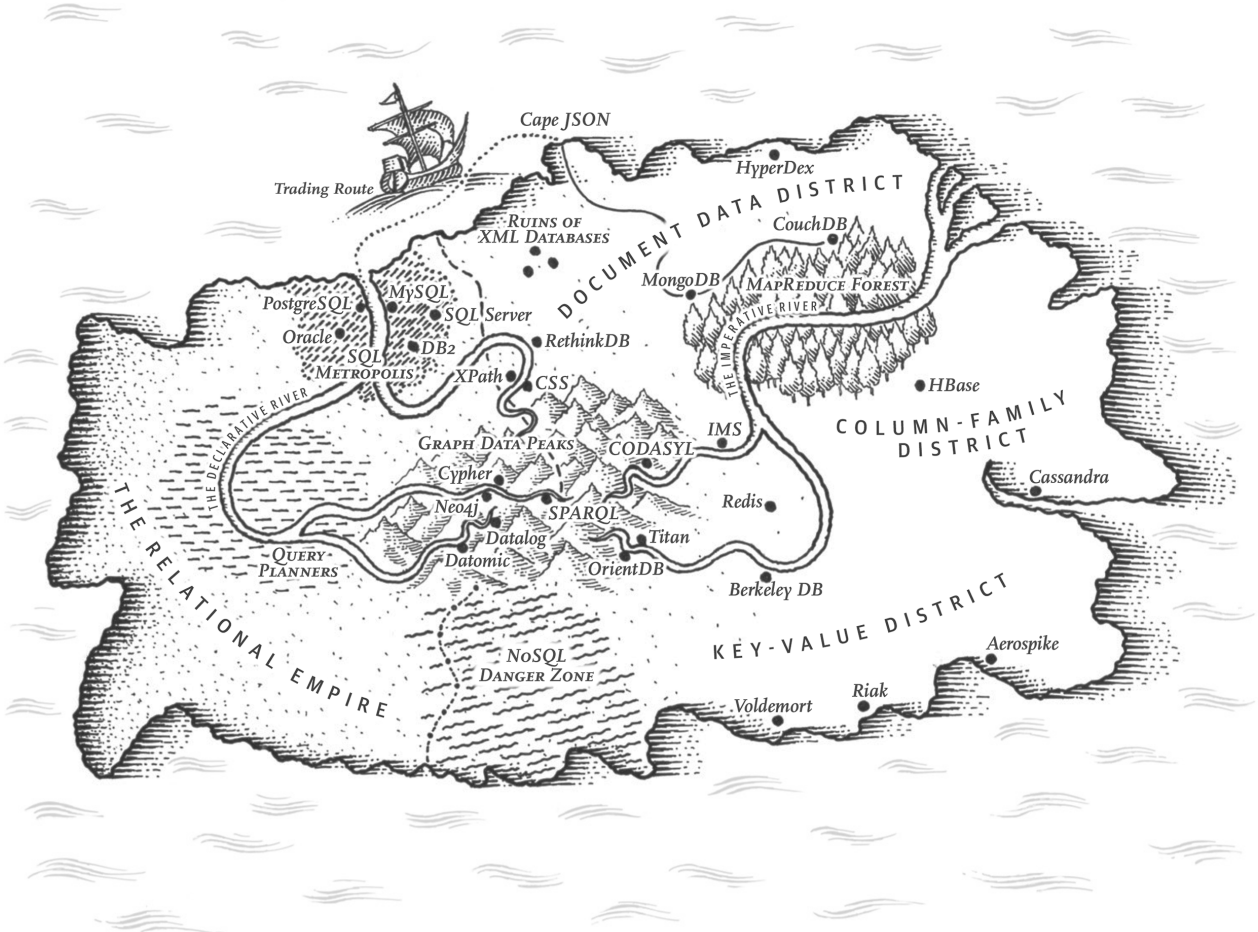


Figure 1: A map of data storage techniques from Designing Data-Intensive Applications [1].

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 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

Relational databases are the types which you will have been exposed too the most in your University career. These databases are exceptional useful with modeling the real world which is a highly connected environment. The data model that is suggested for this type of storage is a normalised approach where data duplication should be reduced.

Some popular offerings are below:

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

The AWS specific offerings of these services come in two differnt types, we have the traditional approach of server capacity (x cores, y ram) and we have a serverless approach. The serverless approach is a more dynamic database that can scale to large amounts of load when needed though at a cost per request.

3.1 ORM

ORM or Object Relational Mapping is a fairly common tool for programmers to use to make developing with databases much smoother. One fairly prevelant example of this is SQLAlchemy which is a very widly used database abstraction for python. With the power of SQLAlchemy we can move above the SQL dialect and perform actions using standard python code.

The benefits of ORMs are the ability to use the power of the programming language we are using instead of having large blocks of SQL text within our source code. The disadvantages come in when we need to do specific SQL work or where the abstractions cost is greater than the benefits.

4 Wide-Column Storage

Wide-Column databases are a form of NoSQL or Non-Relational datastores. In these datastores the data model design is focused more on having efficent queries at the cost of data duplication. A warning to the reader that these models are not flexible after creation, it is much easier to answer a new usecase in a relational model than a NoSQL model.

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

5 Key-Value Storage

Key-Value stores are very popular for cache or remote config usecases, some of the most notable are Redis and Memcached. These stores allow efficent lookup of values and are usually running in-memory.

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

6 Time Series Storage

Timeseries databases are highly focused storage which is tailed to retrieiving results by timestamp ranges. Many implementations also take advatange of the data model to allow efficent rollover of data and partitioning. One of the most popular timeseries databases is Prometheus which is used to store monitoring metrics.

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

7 Document Storage

Document databases are a subset of NoSQL databases with a focus on a flexible data model. MongoDB for instance allows the user to store JSON documents and perform queries on those documents.

- MongoDB.
- Apache CouchDB.
- Amazon DocumentDB.
- Amazon DynamoDB.

8 Graph Storage

Graph Databases are relational storage with a few enhancements to allow fast neighbour lookups. These databases also allow the implementing of Graph algorithms to query data.

- 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 relational 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 equlivant 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.

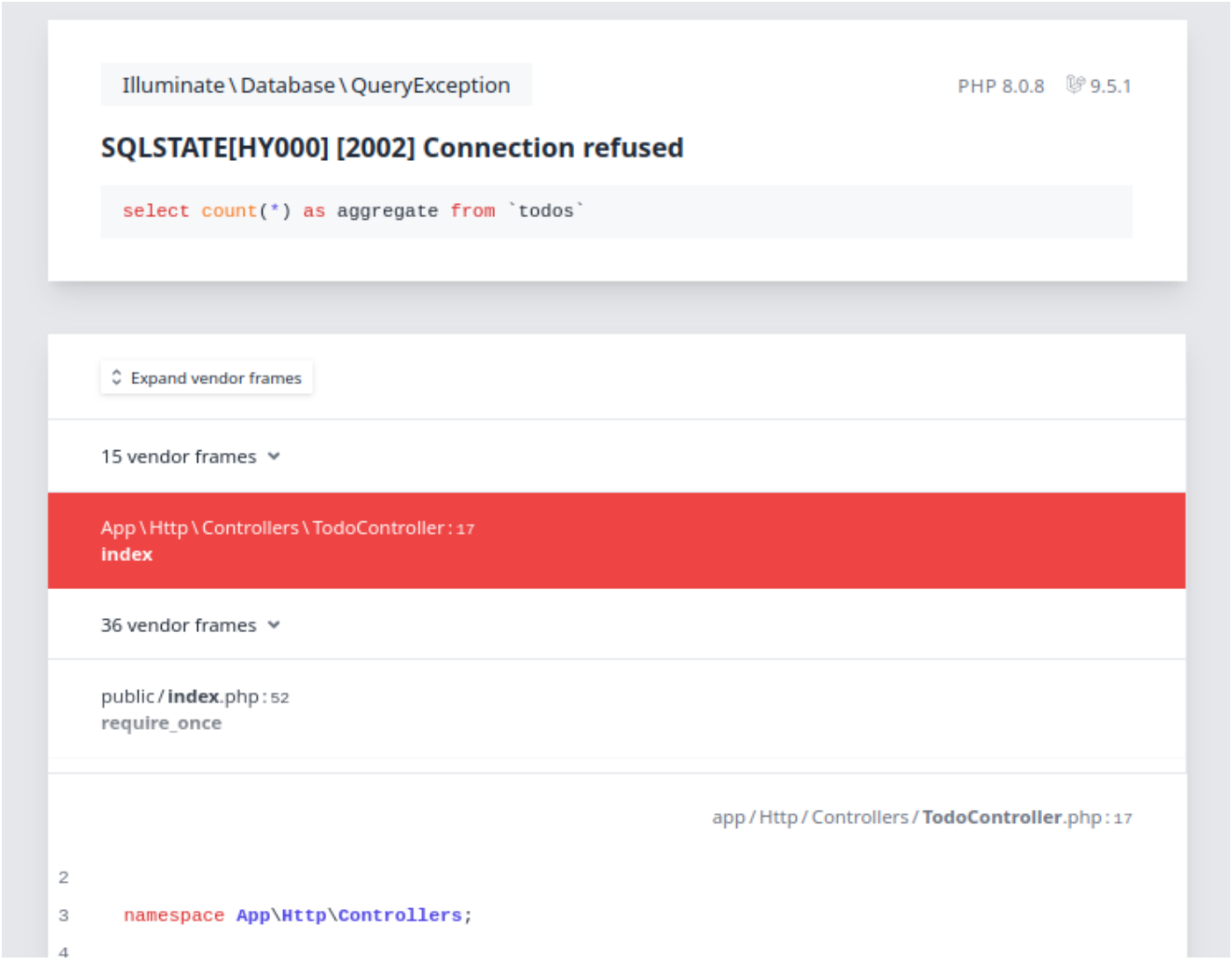
9.1 Locally

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

```
   » cat Dockerfile
FROM ubuntu:21.10
RUN apt-get update \
       && DEBIAN_FRONTEND=noninteractive apt install -y \
           php \
           php-mysql \
           php-xml \
           php-curl \
           curl \
           git \
           unzip
RUN curl -sS https://getcomposer.org/installer | php -- --install-dir=/usr/local/bin
   --filename=composer
COPY . /app
WORKDIR /app
RUN composer install
CMD ["php", "artisan", "serve", "--host=0.0.0.0"]
```

```yaml
version: '3.3'
services:
  backend:
    image: ghcr.io/csse6400/todo-app:latest
    ports:
      - '8000:8000'
    environment:
      APP_ENV: 'local'
      APP_KEY: 'base64:8PQEPYGlTm1t3aqWmlAw/ZPwCiIFvdXDBjk3mhsom/A='
      APP_DEBUG: 'true'
      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
```



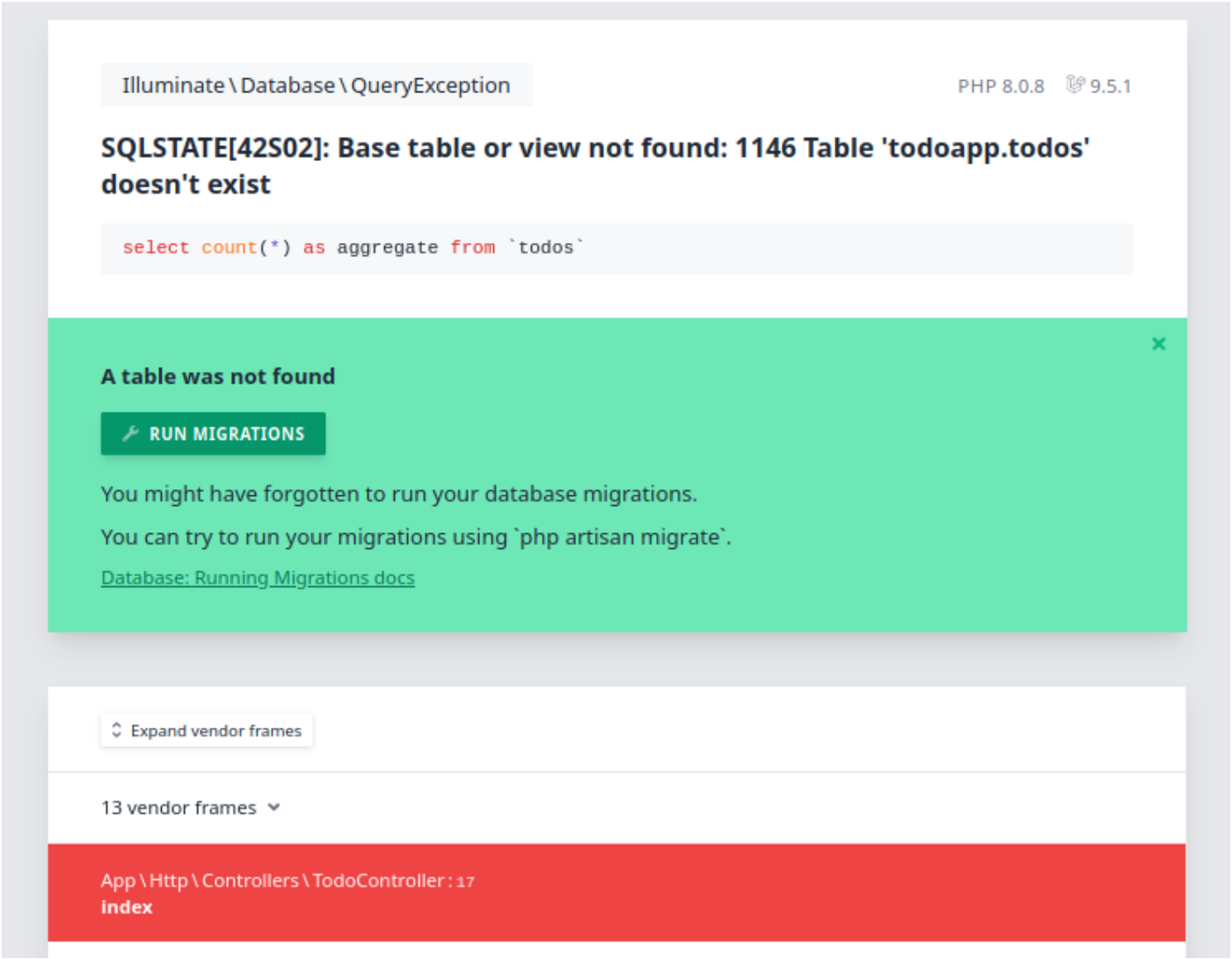
```
» cat main.tf
version: '3.3'
services:
  db:
    image: mysql:8-debian
    environment:
      MYSQL_DATABASE: 'todoapp'
      MYSQL_USER: 'todoapp'
      MYSQL_PASSWORD: 'password'
      MYSQL_ROOT_PASSWORD: 'password'
    ports:
      - '3306:3306'

  backend:
    image: ghcr.io/csse6400/todo-app:latest
    depends_on:
      - db
    ports:
      - '8000:8000'
    environment:
      APP_ENV: 'local'
      APP_KEY: 'base64:8PQEPYGlTm1t3aqWmlAw/ZPwCiIFvdXDBjk3mhsom/A='
      APP_DEBUG: 'true'
      LOG_LEVEL: 'debug'
      DB_CONNECTION: 'mysql'
      DB_HOST: 'db'
      DB_PORT: '3306'
      DB_DATABASE: 'todoapp'
      DB_USERNAME: 'todoapp'
      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




```
$ 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 are 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
FROM ubuntu:21.10
...
...
...
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.

```
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
version: '3.3'
services:
  db:
    ...

  backend:
    ...
    environment:
      ...
    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.