Storing Stuff

Software Architecture

March 14, 2022

Brae Webb

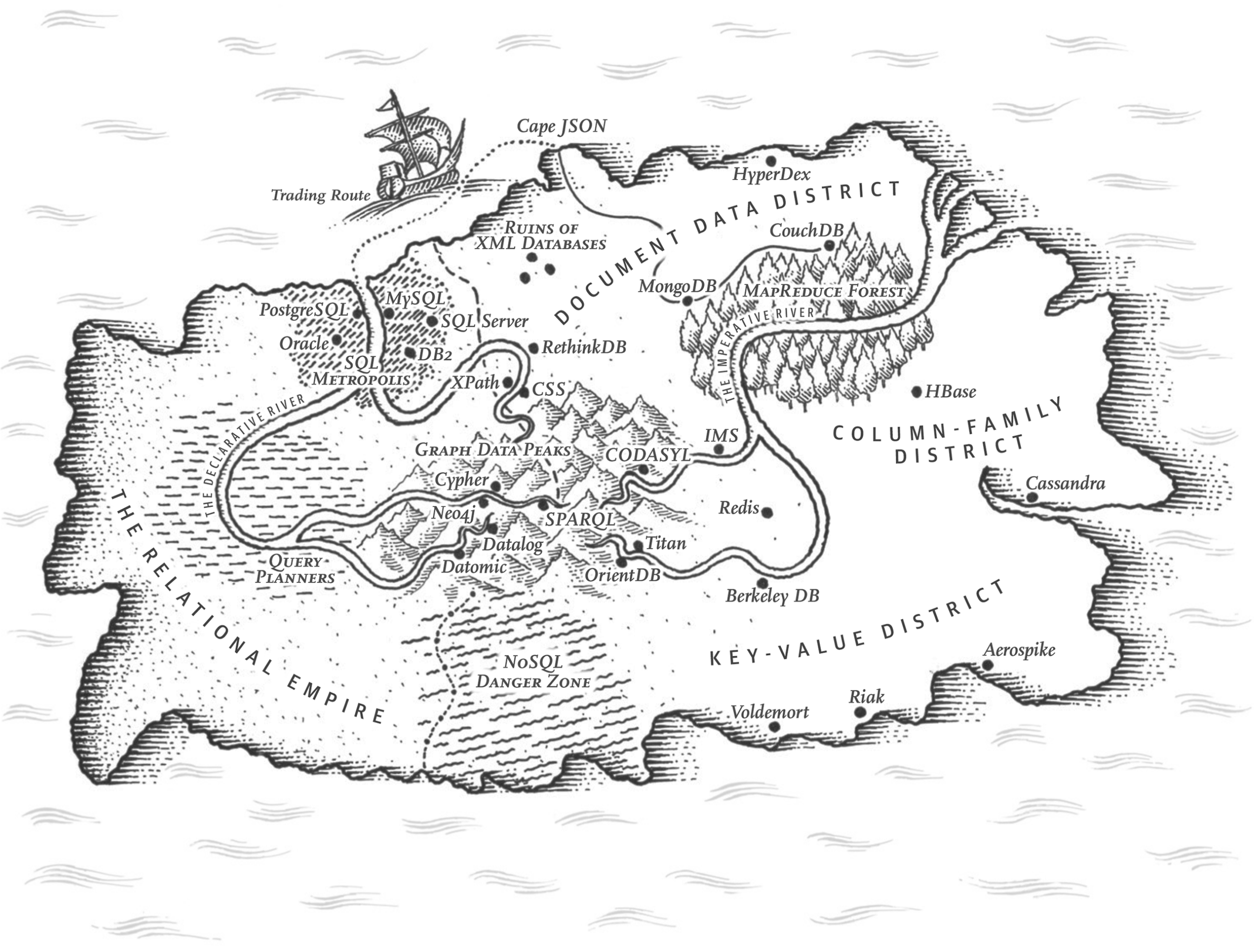

Figure 1: A map of data storage techniques from Designing Data-Intensive Applications [1].

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

- 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 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

```
» 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"]
```

TODO: .. write the docker compose file.

```
» cat main.tf
version: '3.3'
services:
  backend:
    image: ghcr.io/csse6400/todo-app:latest
    restart: always
    ports:
      - '8000:8000'
    environment:
      APP_ENV: 'local'
      APP_KEY: 'base64:8PQEPYGlTm1t3aqWmlAw/ZPwCiIFvdXDBjk3mhsom/A='
      APP_DEBUG: 'true'
      LOG_LEVEL: 'debug'
```

TODO: Include photo showing that we couldnt connect to DB

```
» cat main.tf
version: '3.3'
services:
  db:
    image: mysql:8-debian
    restart: always
    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
    restart: always
    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'
```

9.1.1 Optional Exercise: Fixing the container

In a typical application you dont want to have to enable debug and access an endpoint which is broken inorder to run the database migrations. One way for us to fix this is to run the database migrations when our application starts. Luckily for us the application we have is based off of Laravel which has a cli which can do this for us.

TODO: Walk through building a new container with a script that runs the migrations first.

9.2 AWS

TODO: .. write the terraform to deploy instead.

References

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