# Storing Stuff

Software Architecture

March 14, 2022

Teacher Version

Brae Webb

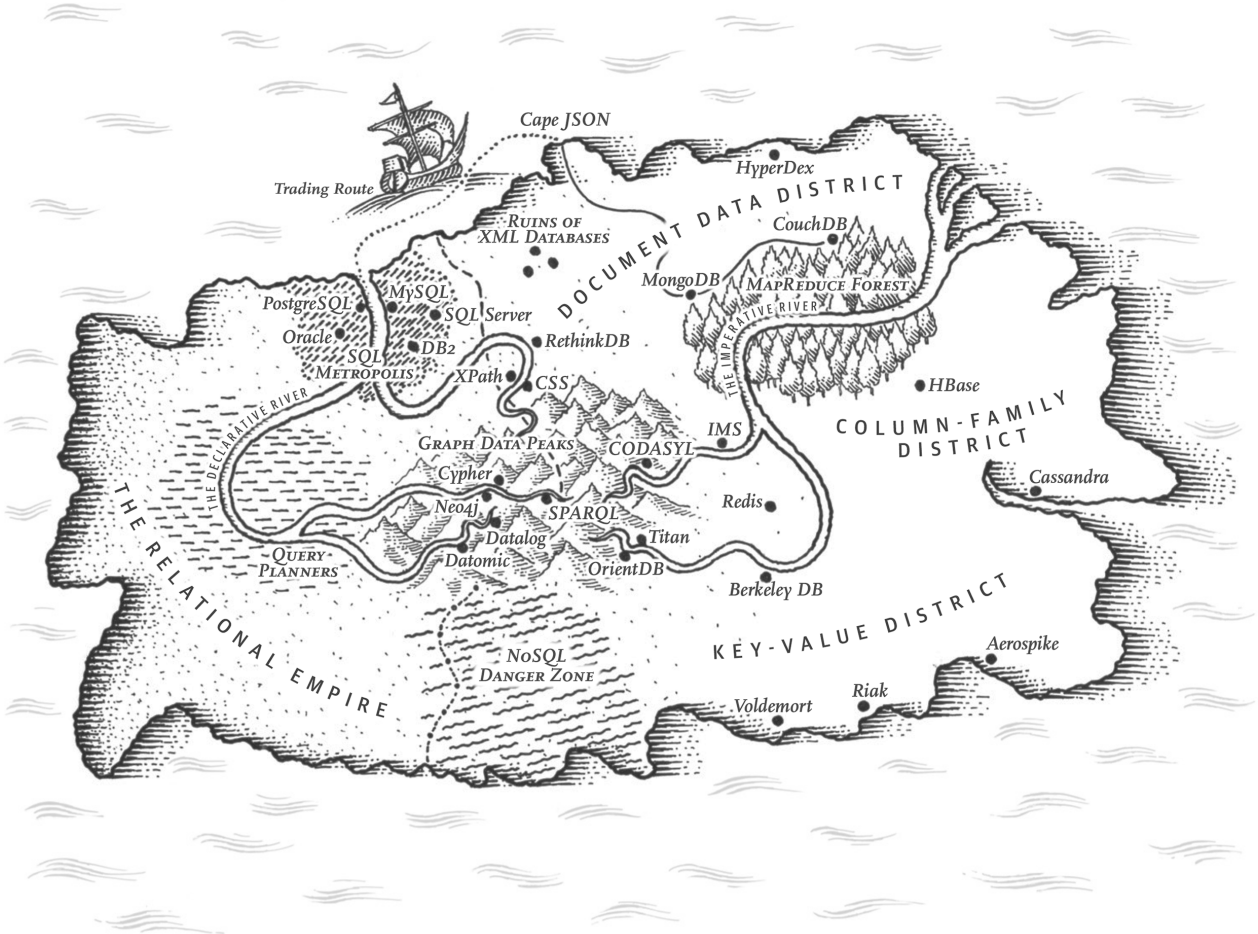



Figure 1: A map of data storage techniques from Designing Data-Intensive Applications [1].

## 1 This Week

This week our goal is to:

- explore the various techniques developers use to store data;
- investigate the storage options implementing these techniques on the AWS platform;
- run a small application using docker that requires a database; and
- deploy the application in AWS using Terraform.

# 2 Databases and Data Models

Unfortunately, to build interesting software we often need to store and use data. The storage of data introduces a number of challenges when 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 Chapter 2 of the *Designing Data-Intensive Applications* book [1].

For the teacher

Discuss the following different storage technologies and mention some use cases of when you would choose each one. Discuss some popular implementations of each.

Aim for no more than 30 minutes of discussion.

## 2.1 Relational Storage

Relational databases what have been exposed to the most in your University career — think MySQL, Postgres, Oracle DB, etc. This type of database is good at modelling the real world which is often a highly connected environment.

Some popular offerings are below:

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

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

### 2.1.1 ORM

Object Relational Mapping (ORM) is a fairly common tool for programmers to use to make developing with databases smoother. One fairly prevalent example of this is SQLAlchemy which is a very widely used database abstraction for python. SQLAlchemy allows us to move to a higher level of abstraction than SQL queries and perform database actions using standard python code.

The benefits of ORMs are the ability to model database objects in our existing programming language 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.

## 2.2 Wide-Column Storage

For the teacher

Examples of big apps that depend on this technology is Netflix https://netflixtechblog.com/netflixs-viewing-data-how-we-know-where-you-are-in-house-of-cards-608dd61077da.

Wide-Column databases are a form of NoSQL or non-relational data stores. In these data stores the data model design is focused more on having efficient 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 use case in a relational model.

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

## 2.3 Key-Value Storage

Key-Value stores are very popular for cache or remote config use cases, some of the most notable are Redis and Memcached. These stores allow efficient lookup of values via keys and are usually stored in-memory.

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

## 2.4 Time Series Storage

For the teacher

Something to mention here is that relations are usually not utilised between tables in time series databases.

Time series databases are highly focused storage which is tailored to retrieving results by timestamp ranges. Many implementations also take advantage of the data model to allow efficient rollover of data and partitioning. One of the most popular time series databases is Prometheus which is used to store monitoring metrics.

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

## 2.5 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. One advantage of document databases is that they match a programmers existing mental model of storing data in formats such as JSON.

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

### 2.6 Graph Storage

For the teacher

If you havnt experienced graph databases, a good usecase is “recommendation systems”, which use the connected nature of items to figure out what to suggest to a person.Another example is the https://neo4j.com/blog/analyzing-panama-papers-neo4j/ Panama Papers.

Graph Databases are relational storage with a few enhancements to allow fast neighbour look-ups. These databases also allow the implementation of graph algorithms to query data.

- Amazon Neptune.
- Neo4J.
- Janus Graph.

## 3 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 consists of a web server and a relational database.

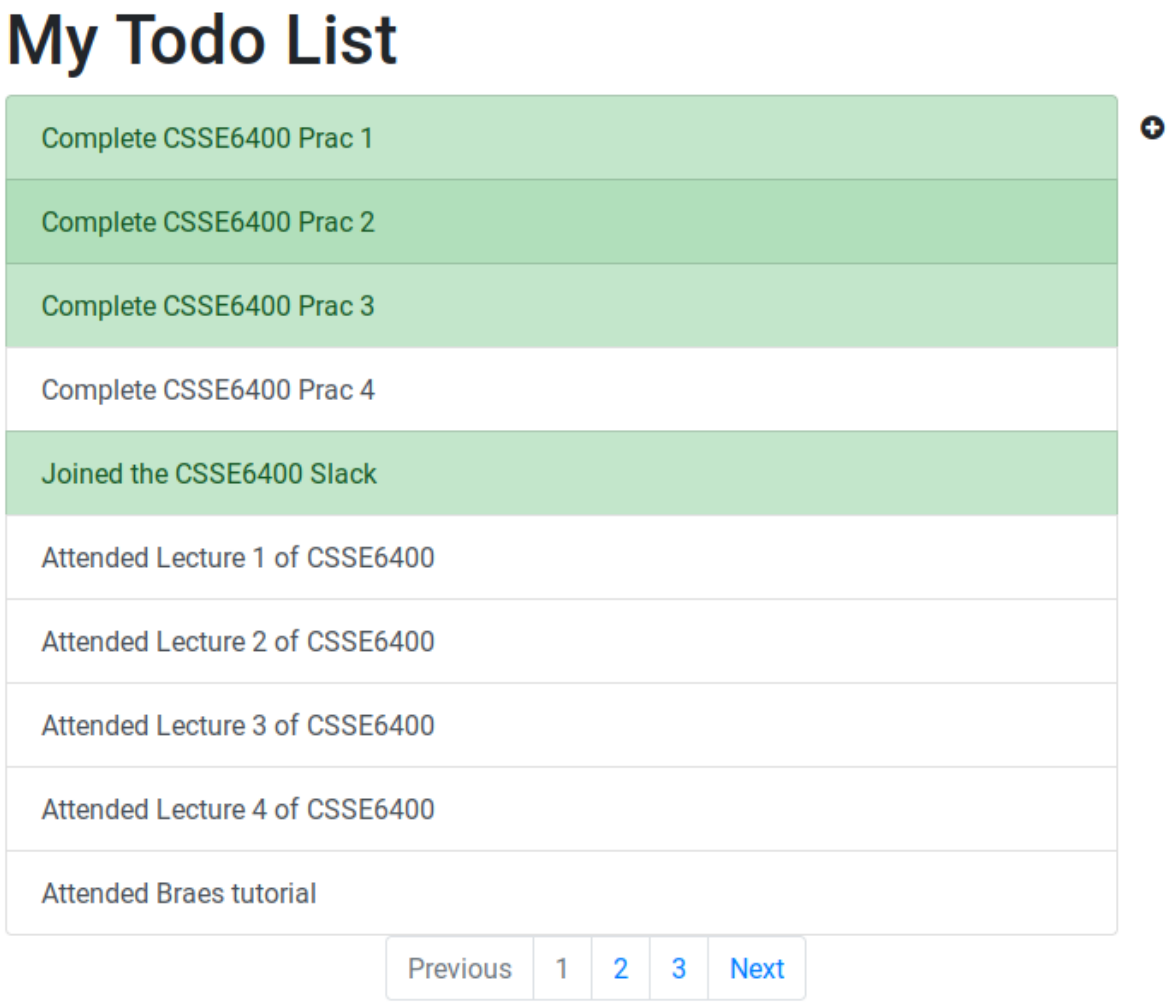



Figure 2: Sample Todo App made by Brae Webb

**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.

**For the teacher**

Wait for students to get docker-compose installed, they should have docker from their tutorials but some may be missing it.

**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.

## 3.1 Locally

**For the teacher**

Mention that the dockerfile exists but no need to get the repo. We will not be building the container ourselves. Instead use one that is published on the github, shown further down in the docker-compose.

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
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"]
```

Our goal for today is to have a running instance of the Todo App locally including the database. To get started we need to make a new directory for our work and create a Docker compose file.

```
$ mkdir prac4 && cd prac4
$ touch docker-compose.yml
```

Docker Compose is a small helper utility that allows us to more easily run docker applications without needing to remember a lot of command line parameters. Instead we define how we want our docker container to run through a YAML config file. Insert the following into your docker-compose.yml file.

```
   » cat docker-compose.yml
```

```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'
```

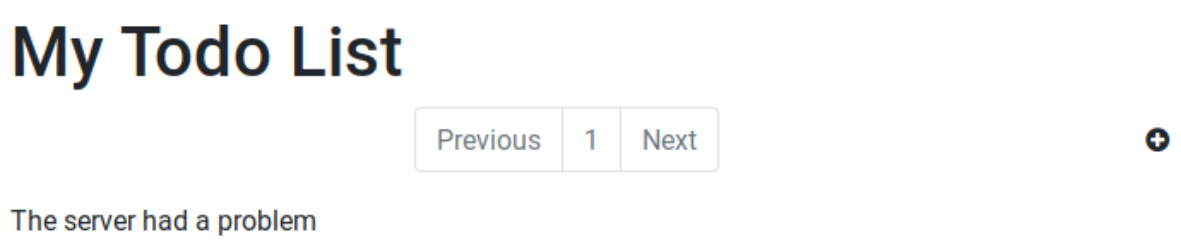
**For the teacher**

Feel free to show students dockerhub and where on github this container is stored. URL for this container is here https://github.com/CSSE6400/todo-app/pkgs/container/todo-app

A few things to point out in the file. We have defined a single service called backend which uses a pre-made docker image from ghcr.io/csse6400/todo-app with the tag of latest. We then have exposed this onto our machine on port 8000 and have passed a few environment variables.

```
$ 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
```

Now we head to our browser and go to http://127.0.0.1:8000, you should be presented with the following screen.




To investigate this error lets hit one of the endpoints for our api. Head over to http://127.0.0.1:8000/api/v1/todo which should list all the todos. Once we reach that page we have a clearer idea of whats gone wrong. The page you should see is shown in Figure 3 and the quick summary is that our App is complaining that we haven't given it a database.

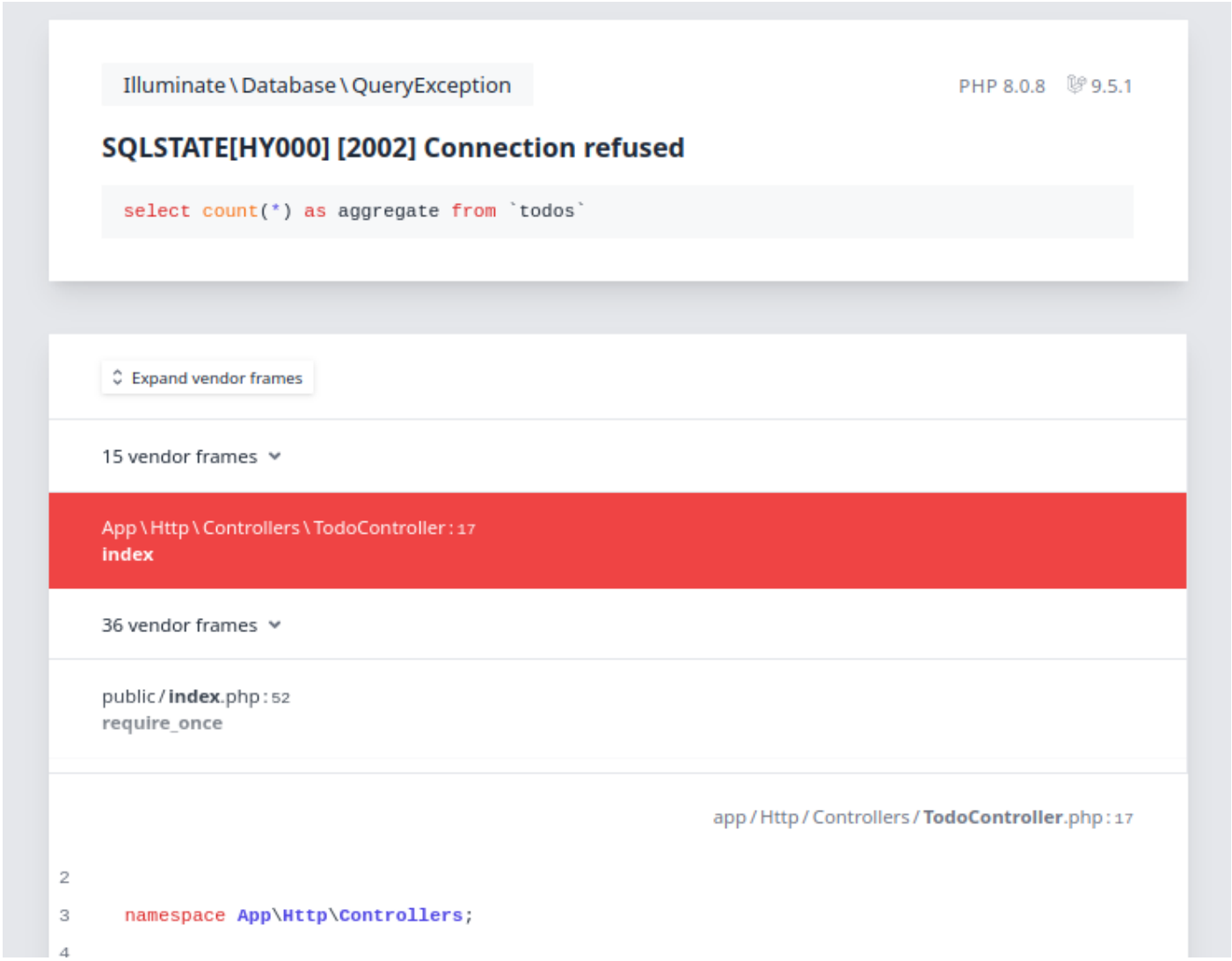



Figure 3: The expected error page when accessing `http://127.0.0.1:8000/api/v1/todo`.

To fix this lets add a popular relation database, MySQL, to our docker-compose file. Edit your docker compose file to match as shown below.

```
   » 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'
```

Now we have two services for our app and we have added a few more environment variables for our backend to know how to connect to the database. The `DB_HOST` variable uses a feature of docker compose where you can refer to other services by their name. This makes it easy for us to setup communication between these two services.

From the same shell let's re-run our containers, you may need to CTRL+C to stop the current running containers. Once they have shutdown, run the up command again.

```
$ 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....
```

Now when we go to http://127.0.0.1:8000/api/v1/todo we see a different error message, as shown in Figure 4. This error is complaining that we have a database that we can connect to but the todos table doesn't exist.

To populate the database our application comes with database migration files. One way would be to click the "RUN MIGRATIONS" button shown on the error page but we also want to pre-populate our database with some dummy data as well.

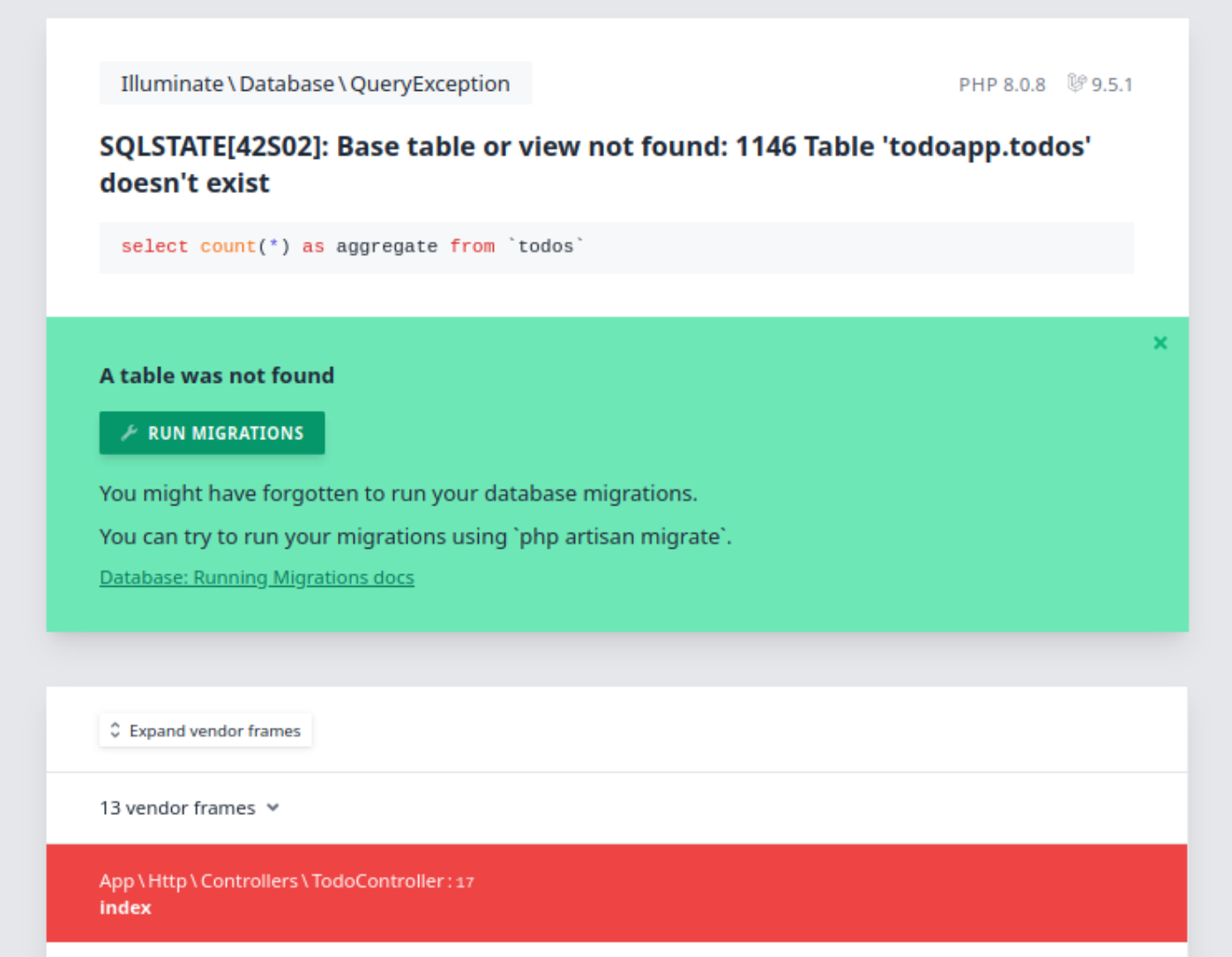
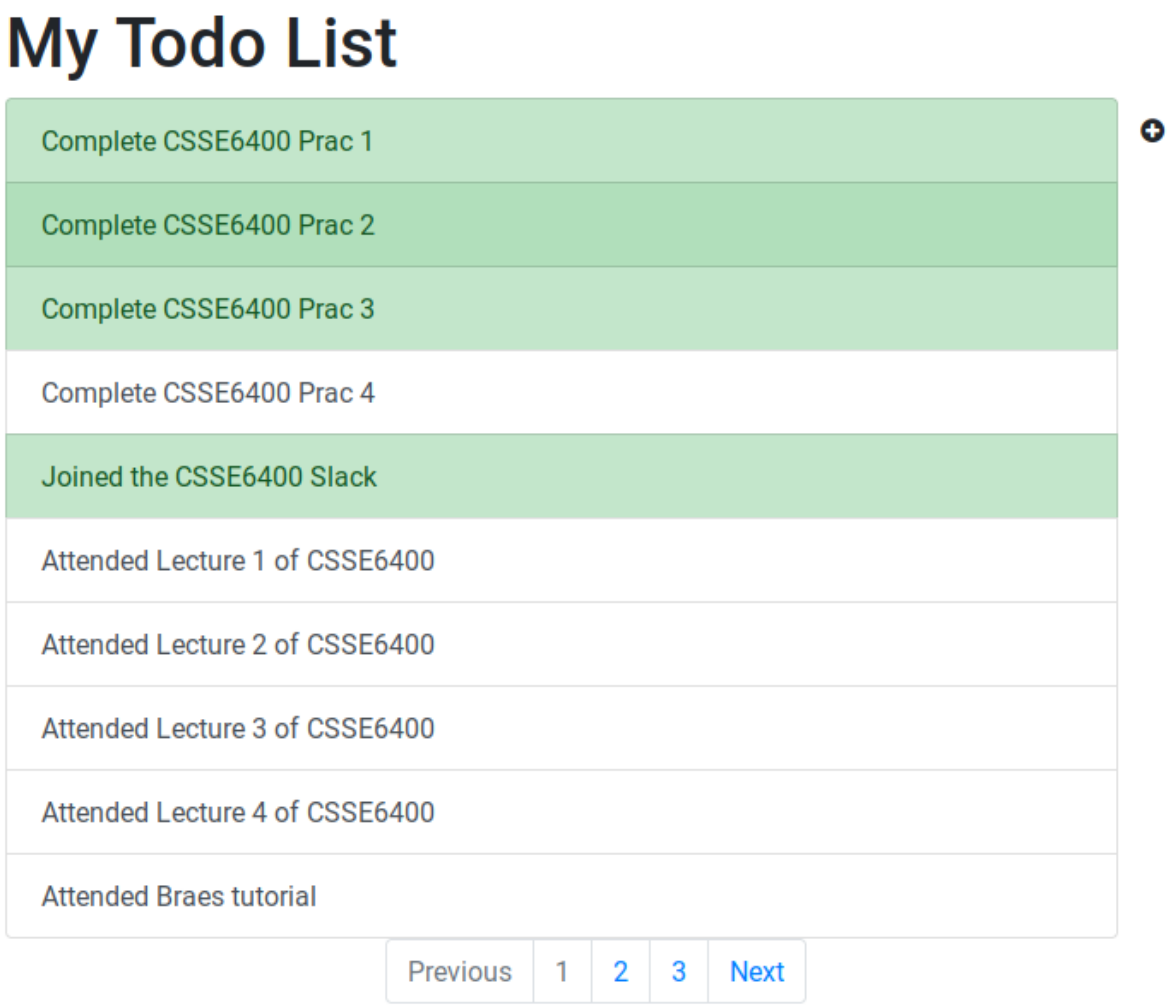



Figure 4: The expected error page when accessing `http://127.0.0.1:8000/api/v1/todo` after creating a database.

To do this we are going to jump into the running container and execute the migrations ourselves. Start by opening a new terminal so that we can leave the docker containers running. In this new terminal go to the same directory that we were just in and run the following command:

```
$ 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.
```

Now with this run we can check back at our web app and you should see a fully functional todo app.




**Info**

These migrations are performed by Laravel which is a popular PHP web framework. In the database it has created a table to keep track of which migrations have already been run so that it will skip them latter. To enable this functionality you will have to edit migrate:fresh to just migrate.

For a production instance you would also typically remove the –seed parameter as you do not want to insert dummy data into your production database.

### 3.1.1 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 practical 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 practical.

```
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: condensed version of the 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.

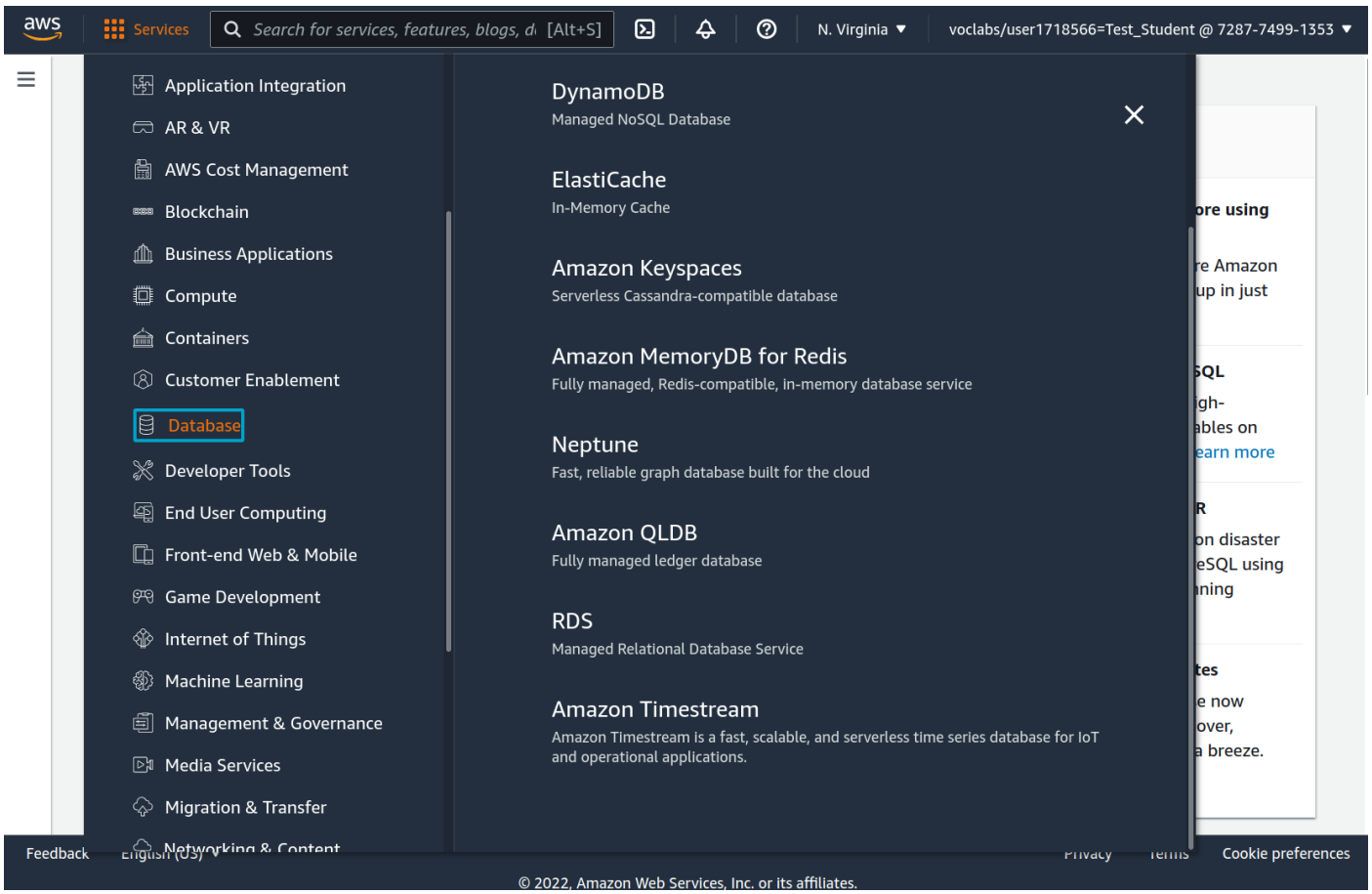
```
$ docker-compose up
...
...
backend_1 | Rolling back: 2022_03_19_041557_create_todos_table
backend_1 | Rolled back: 2022_03_19_041557_create_todos_table (8.28ms)
backend_1 | Migrating: 2022_03_19_041557_create_todos_table
backend_1 | Migrated: 2022_03_19_041557_create_todos_table (11.55ms)
backend_1 | 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
```

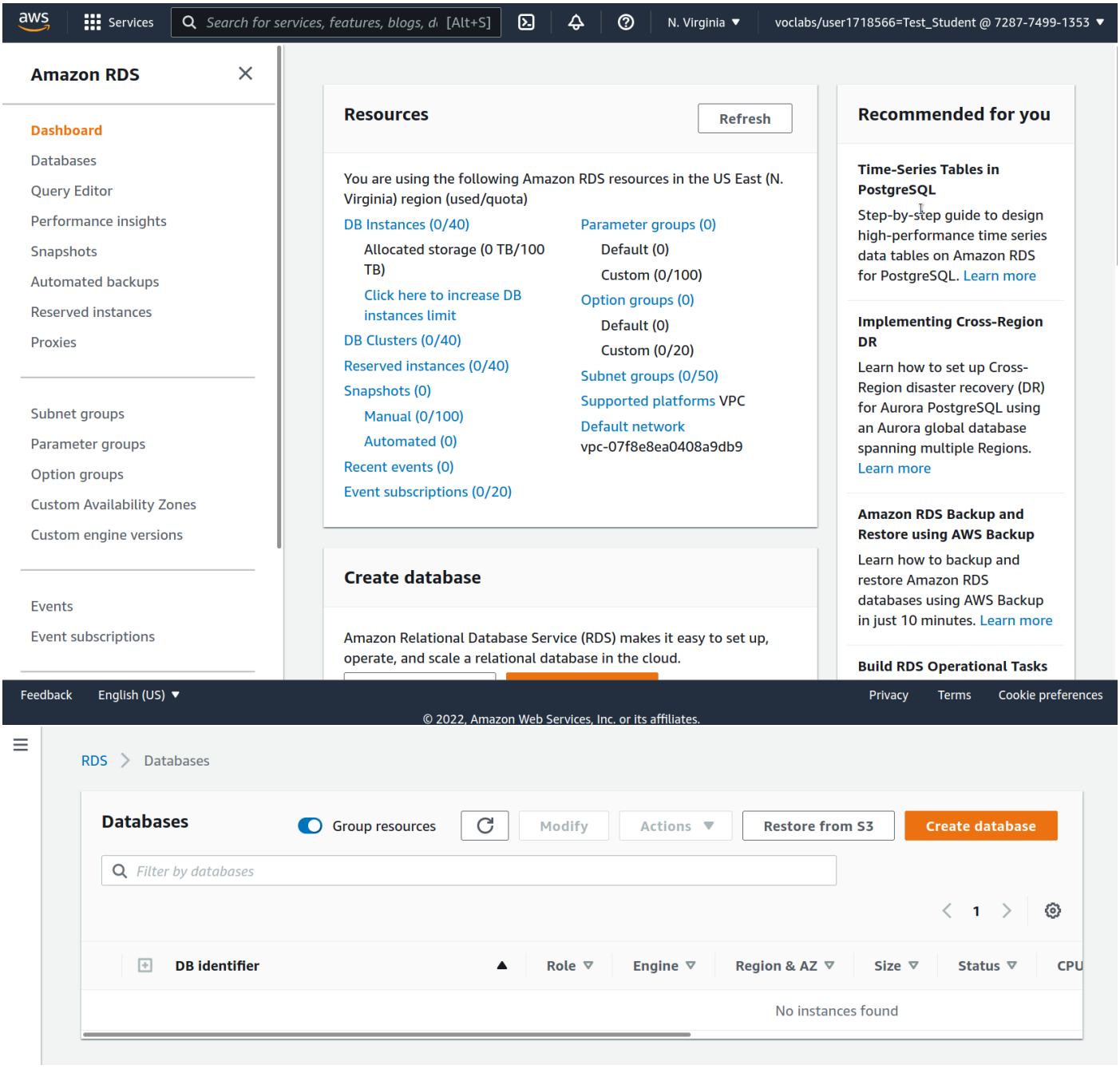
We can also bake this into the container by extending the original, it is fairly common to see projects in the wild that run a init script when the container launches. An exercise left for the reader is to build upon the provided docker container by including an init script.

## 3.2 AWS

**Warning**

This section is still being developed.




aws
Services
Search for services, features, blogs, d [Alt+S]
N. Virginia
voclabs/user1718566=Test_Student @ 7287-7499-1353
Amazon RDS
Dashboard
Databases
Query Editor
Performance insights
Snapshots
Automated backups
Reserved instances
Proxies
Subnet groups
Parameter groups
Option groups
Custom Availability Zones
Custom engine versions
Events
Event subscriptions
Resources
Refresh
You are using the following Amazon RDS resources in the US East (N. Virginia) region (used/quota)
DB Instances (0/40)
Allocated storage (0 TB/100 TB)
Click here to increase DB instances limit
DB Clusters (0/40)
Reserved instances (0/40)
Snapshots (0)
Manual (0/100)
Automated (0)
Recent events (0)
Event subscriptions (0/20)
Parameter groups (0)
Default (0)
Custom (0/100)
Option groups (0)
Default (0)
Custom (0/20)
Subnet groups (0/50)
Supported platforms VPC
Default network
vpc-07f8e8ea0408a9db9
Create database
Amazon Relational Database Service (RDS) makes it easy to set up, operate, and scale a relational database in the cloud.
Recommended for you
Time-Series Tables in PostgreSQL
Step-by-step guide to design high-performance time series data tables on Amazon RDS for PostgreSQL. Learn more
Implementing Cross-Region DR
Learn how to set up Cross-Region disaster recovery (DR) for Aurora PostgreSQL using an Aurora global database spanning multiple Regions. Learn more
Amazon RDS Backup and Restore using AWS Backup
Learn how to backup and restore Amazon RDS databases using AWS Backup in just 10 minutes. Learn more
Build RDS Operational Tasks
Feedback
English (US)
Privacy
Terms
Cookie preferences
© 2022, Amazon Web Services, Inc. or its affiliates.
RDS
Databases
Databases
Group resources
Modify
Actions
Restore from S3
Create database
Filter by databases
1
DB identifier
Role
Engine
Region & AZ
Size
Status
CPU
No instances found

## Choose a database creation method Info

- (•) **Standard create**
  You set all of the configuration options, including ones for availability, security, backups, and maintenance.
- ( ) **Easy create**
  Use recommended best-practice configurations. Some configuration options can be changed after the database is created.

## Engine options

Engine type Info

- ( ) Amazon Aurora
- (•) MySQL
- ( ) MariaDB
- ( ) PostgreSQL
- ( ) Oracle
- ( ) Microsoft SQL Server

Edition

(•) MySQL Community

**Known issues/limitations**
Review the Known issues/limitations to learn about potential compatibility issues with specific database versions.

Version

MySQL 8.0.27

## Templates

Choose a sample template to meet your use case.

- ○ **Production**
  Use defaults for high availability and fast, consistent performance.
- ○ **Dev/Test**
  This instance is intended for development use outside of a production environment.
- ◉ **Free tier**
  Use RDS Free Tier to develop new applications, test existing applications, or gain hands-on experience with Amazon RDS. **Info**

## Availability and durability

Deployment options **Info**

The deployment options below are limited to those supported by the engine you selected above.

- ◉ Single DB instance (not supported for Multi-AZ DB cluster snapshot)
  Creates a single DB instance with no standby DB instances.
- ○ Multi-AZ DB instance (not supported for Multi-AZ DB cluster snapshot)
  Creates a primary DB instance and a standby DB instance in a different AZ. Provides high availability and data redundancy, but the standby DB instance doesn't support connections for read workloads.
- ○ Multi-AZ DB Cluster - new
  Creates a DB cluster with a primary DB instance and two readable standby DB instances, with each DB instance in a different Availability Zone (AZ). Provides high availability, data redundancy and increases capacity to serve read workloads.

## Settings

DB instance identifier **Info**

Type a name for your DB instance. The name must be unique across all DB instances owned by your AWS account in the current AWS Region.

todoapp-manual

The DB instance identifier is case-insensitive, but is stored as all lowercase (as in "mydbinstance"). Constraints: 1 to 60 alphanumeric characters or hyphens. First character must be a letter. Can't contain two consecutive hyphens. Can't end with a hyphen.

▼ **Credentials Settings**

Master username **Info**

Type a login ID for the master user of your DB instance.

todoapp

1 to 16 alphanumeric characters. First character must be a letter.

☐ Auto generate a password

Amazon RDS can generate a password for you, or you can specify your own password.

Master password **Info**

••••••••••••••••••••

Constraints: At least 8 printable ASCII characters. Can't contain any of the following: / (slash), '(single quote), "(double quote) and @ (at sign).

Confirm password **Info**

••••••••••••••••••••

## DB instance class

DB instance class **Info**

○ Standard classes (includes m classes)

○ Memory optimized classes (includes r and x classes)

◉ Burstable classes (includes t classes)

db.t2.micro
1 vCPUs 1 GiB RAM Not EBS Optimized ▼

◯ Include previous generation classes

## Storage

Storage type Info

General Purpose SSD (gp2)
Baseline performance determined by volume size

Allocated storage

20 GiB

(Minimum: 20 GiB. Maximum: 16,384 GiB) Higher allocated storage **may improve** IOPS performance.

You might see better baseline performance with your selected volume size by specifying General Purpose SSD storage. Learn more about using Provisioned IOPS storage for consistent performance.

### Storage autoscaling Info

Provides dynamic scaling support for your database's storage based on your application's needs.

☑ Enable storage autoscaling
Enabling this feature will allow the storage to increase once the specified threshold is exceeded.

Maximum storage threshold Info

Charges will apply when your database autoscales to the specified threshold

1000 GiB

Minimum: 21 GiB. Maximum: 16,384 GiB

## Connectivity

Virtual private cloud (VPC) Info
VPC that defines the virtual networking environment for this DB instance.

Default VPC (vpc-07f8e8ea0408a9db9)

Only VPCs with a corresponding DB subnet group are listed.

After a database is created, you can't change its VPC.

Subnet group Info
DB subnet group that defines which subnets and IP ranges the DB instance can use in the VPC you selected.

default-vpc-07f8e8ea0408a9db9

Public access Info

- Yes
  Amazon EC2 instances and devices outside the VPC can connect to your database. Choose one or more VPC security groups that specify which EC2 instances and devices inside the VPC can connect to the database.
- No
  RDS will not assign a public IP address to the database. Only Amazon EC2 instances and devices inside the VPC can connect to your database.

VPC security group
Choose a VPC security group to allow access to your database. Ensure that the security group rules allow the appropriate incoming traffic.

- Choose existing
  Choose existing VPC security groups
- Create new
  Create new VPC security group

New VPC security group name

todoapp-manual

Availability Zone Info

No preference

▼ Additional configuration

Database port Info
TCP/IP port that the database will use for application connections.

3306

## Database authentication

Database authentication options Info

- (•) Password authentication
  Authenticates using database passwords.
- ( ) Password and IAM database authentication
  Authenticates using the database password and user credentials through AWS IAM users and roles.
- ( ) Password and Kerberos authentication
  Choose a directory in which you want to allow authorized users to authenticate with this DB instance using Kerberos Authentication.

## ▼ Additional configuration

Database options, backup enabled, backtrack disabled, Enhanced Monitoring disabled, maintenance, CloudWatch Logs, delete protection disabled.

### Database options

Initial database name Info

todoapp

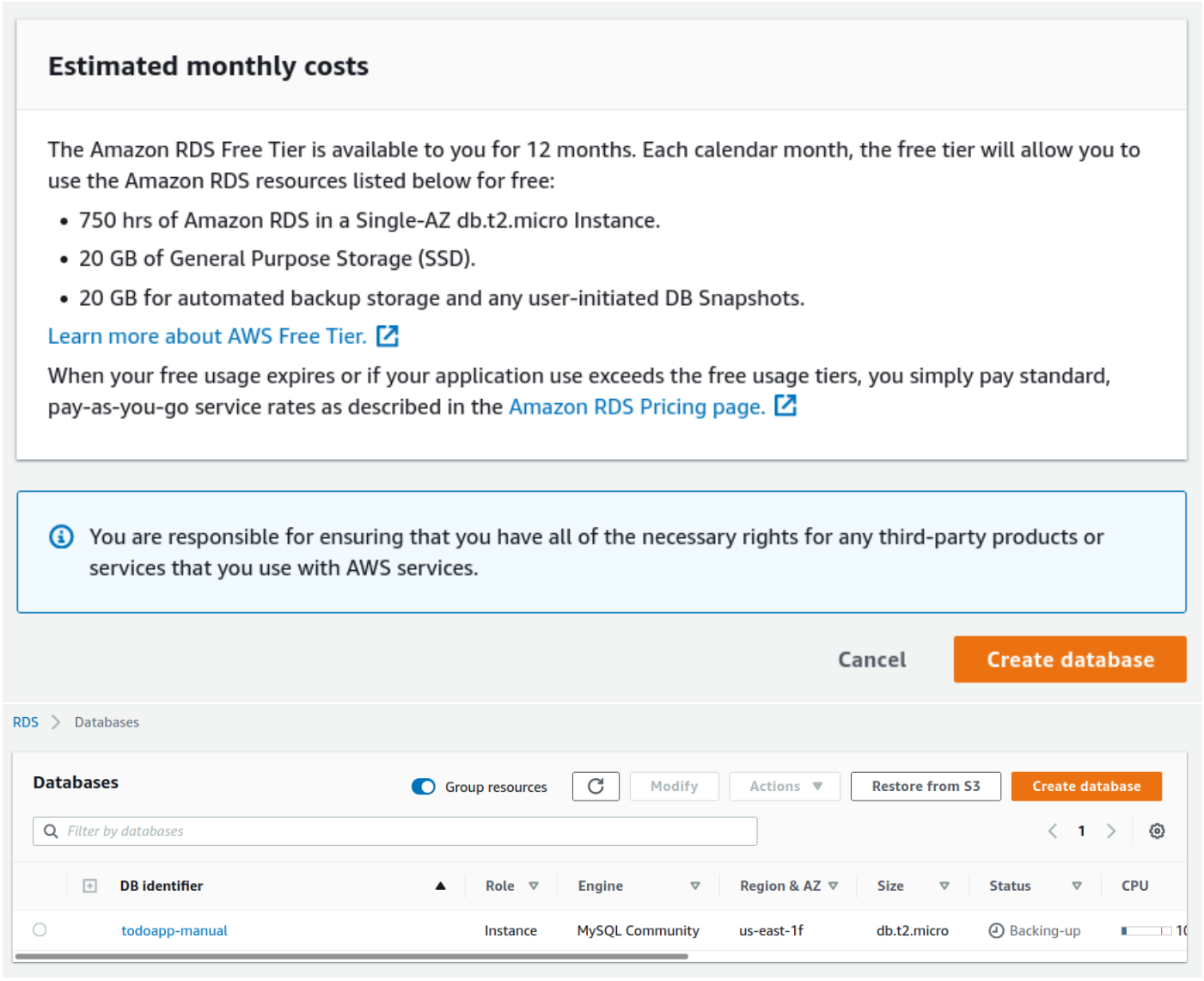
If you do not specify a database name, Amazon RDS does not create a database.

DB parameter group Info

default.mysql8.0

Option group Info

default:mysql-8-0




# References

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