Deployment Note

Please make sure to read this note carefully as it has all the general information you need to deploy front + backend. However it does not substitute going through the videos as it mainly holds complimentary information to the videos, not all of the specific steps. When running through the videos, I'll often reference this note, as there are quite a few code snippets that are easier to copy & paste from here instead of copying them from the screen during the videos.

General Overview

There are two paths you can follow. The first one, which is only used to deploy the two codebases to the web, while using a new, completely blank database.
This one is the normal path to take, when working with a client, as you usually test locally and then hand over the project for your client to fill in the actual data/content.

However I did want to show you as well how you can migrate your local SQLite database to a Postgres database and then dump all the locally created data into your cloud database. But just be aware, that this involves quite a few steps and I don't necessarily see the benefit of hosting exactly the project from the course. Unless you've created a project you actually want to deliver locally on your own throughout this course, this path is not recommended.

On top of that, we'll be using Heroku for hosting the front- and backend in this guide. They **do not have a free tier** anymore, meaning that the server cost will be about 15$ per month.
If you're wondering why: There are two reasons:

1. I want to show you exactly what works in production and not just use some free tier service where we could get this potentially running but at the expense of performance and other potential issues.
2. When freelancing, your client will pay for hosting. And hosting for 15$ per month is quite cheap for a lot of companies. I've heard of a company that used to pay an agency 8.000$ per year for hosting, in a situation where our setup would have easily sufficed.

The server costs are billed by the hour, so if you just want to follow along to see that everything works and then delete the servers after a day, you'll have paid about 50 cents.

Tools

- You will need to install the Heroku CLI to run a few commands in order to deploy to the server.

- **If** you follow the path with database migration, you might find it easier to work with a tool that allows you to connect to your cloud database instead of just using the terminal. In the videos I'm using TablePlus for this. Click here if you want to download TablePlus (the basic version is free.

Code Snippets referenced in the lecture

All of the changes below are made within the Strapi folder. The changes to the frontend are minimal and explained in the video.
Here are a few links where you can find the source code:

- Repository of deployed Strapi Project
- Commit: Setup Strapi for Deployment
- Commit: Database Migration
- Commit: S3 Image Integration

Strapi Deployment

- After you created your Strapi Heroku Server, you will need to set some default env variables. Here's the code I'm using in the video:

```
heroku config:set APP_KEYS=$(openssl rand -base64 32)
heroku config:set API_TOKEN_SALT=$(openssl rand -base64 32)
heroku config:set ADMIN_JWT_SECRET=$(openssl rand -base64 32)
heroku config:set JWT_SECRET=$(openssl rand -base64 32)
```

- `config/env/production/server.js` file

```
module.exports = ({ env }) => ({
  proxy: true,
  url: env("MY_HEROKU_URL"),
  app: {
    keys: env.array("APP_KEYS"),
  },
});
```

Database Migration

1. Export the data from the sqlite database using `strapi export`

```
npm run strapi export
```

2. After installing PostgreSQL locally, you need to create a local db to use for strapi. To do this you can run the following code in terminal:

```
psql -U postgres
```

This command will prompt you for the password for the `postgres` user (the default superuser for PostgreSQL). If you haven't set a password or changed the username, it's usually `postgres` for local installations.

```
CREATE DATABASE surfcamp_db;
```

Now make Strapi use the newly created `surfcamp_db` database. In the video we do this by adjusting our database client within `config/database.js` to `postgres` and deleting the environment variables set in `.env` from the Strapi Quickstart.

```
DATABASE_CLIENT=sqlite
DATABASE_FILENAME=.tmp/data.db
```

To then also make your production Strapi environment use the correct database, you'll have to crate a new file in `config/env/production/database.js`:

```
const parse = require("pg-connection-string").parse;
const config = parse(process.env.DATABASE_URL);
module.exports = ({ env }) => ({
  connection: {
    client: "postgres",
    connection: {
      host: config.host,
      port: config.port,
      database: config.database,
      user: config.user,
      password: config.password,
      ssl: {
        rejectUnauthorized: false,
      },
    },
    debug: false,
  },
});
```

Make sure you also have `pg-connection-string` installed by running `npm i pg-connection-`

`string` inside of your strapi repository.

Image Upload S3 with Strapi

- `config/plugins.js` file within Strapi Repository:

```
module.exports = ({ env }) => ({
  upload: {
    config: {
      provider: "aws-s3",
      providerOptions: {
        accessKeyId: env("AWS_ACCESS_KEY_ID"),
        secretAccessKey: env("AWS_ACCESS_SECRET_KEY"),
        region: env("AWS_REGION", "eu-central-1"),
        params: {
          Bucket: env("AWS_BUCKET", "surfcamp-strapi-images"),
        },
      },
    },
  },
});
```

- `config/middlewares.js` file within Strapi:

```
module.exports = [
  "strapi::errors",
  "strapi::cors",
  "strapi::poweredBy",
  "strapi::logger",
  "strapi::query",
  "strapi::body",
  "strapi::favicon",
  "strapi::public",
  {
    name: "strapi::security",
    config: {
      contentSecurityPolicy: {
        useDefaults: true,
        directives: {
          "connect-src": ["'self'", "https:"],
          "img-src": [
```

```
            "'self'",
            "data:",
            "blob:",
            `${process.env.AWS_BUCKET}.s3.$
{process.env.AWS_REGION}.amazonaws.com`,
          ],
          "media-src": [
            "'self'",
            "data:",
            "blob:",
            `${process.env.AWS_BUCKET}.s3.$
{process.env.AWS_REGION}.amazonaws.com`,
          ],
          upgradeInsecureRequests: null,
        },
      },
    },
  },
];
```

Replace Image link in Database

Here is the query I use in the video to replace all local upload paths with the new s3 bucket upload path. You have to adjust the amazonaws link though, as the current query points towards my bucket! Also be aware that there are two instances of that link, you have to replace both!

SQL Query:

```
UPDATE files
SET url = REPLACE(url, '/uploads/', 'https://surfcamp-strapi-images.s3.eu-
central-1.amazonaws.com/');

UPDATE files
SET provider = 'aws-s3'
WHERE provider = 'local';

UPDATE files
SET formats = REPLACE(formats::text, '/uploads/', 'https://surfcamp-strapi-
images.s3.eu-central-1.amazonaws.com/')::jsonb
WHERE formats::text LIKE '%/uploads/%';
```