Bonus: Storing Uploaded Images In The Cloud (AWS S3)

As explained in the previous lecture, storing uploaded files (or any other files that are generated at runtime) on the local filesystem is not a great idea - because those files will simply not be available in the running NextJS applications.

Instead, it's recommended that you store such files (e.g., uploaded images) via some cloud file storage - like AWS S3.

AWS S3 is a service provided by AWS which allows you to store and serve (depending on its configuration) files. You can get started with this service for free but you should check out its pricing page to avoid any unwanted surprises.

In this lecture, I'll explain how you could use AWS S3 to store uploaded users images & serve them on the NextJS website.

1) Create an AWS account

In order to use AWS S3, you need an AWS account. You can create one here.

2) Create a S3 bucket

Once you created an account (and you logged in), you should navigate to the S3 console to create a so-called "bucket".

"Buckets" are containers that can be used to store files (side-note: you can store any files - not just images).

Every bucket must have a globally unique name, hence you should become creative. You could, for example, use a name like *<your-name>-nextjs-demo-users-image*.

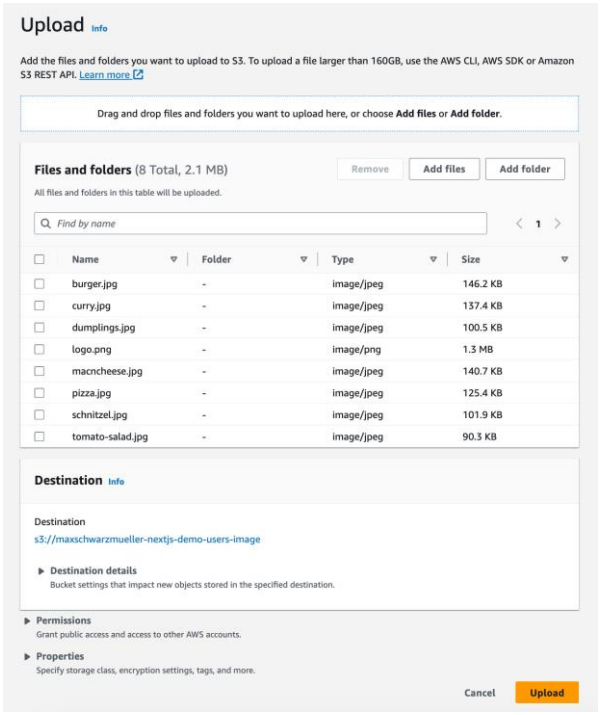
I'll use *maxschwarzmueller-nextjs-demo-users-image* in this example here.

When creating the bucket, you can confirm all the default settings - the name's the only thing you should set.

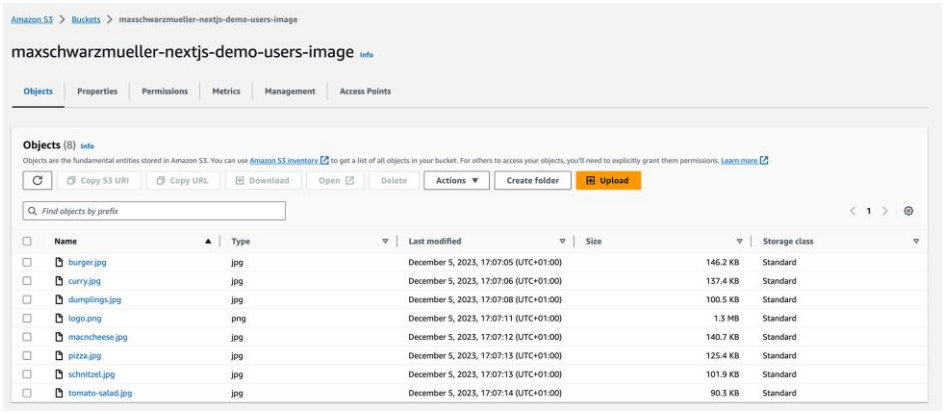
3) Upload the dummy image files

Now that the bucket was created, you can already add some files to it => The dummy images that were previously stored locally in the `public/images` folder.

To do that, select your created bucket and click the "Upload" button. Then drag & drop those images into the box and confirm the upload.



Thereafter, all those images should be in the bucket:



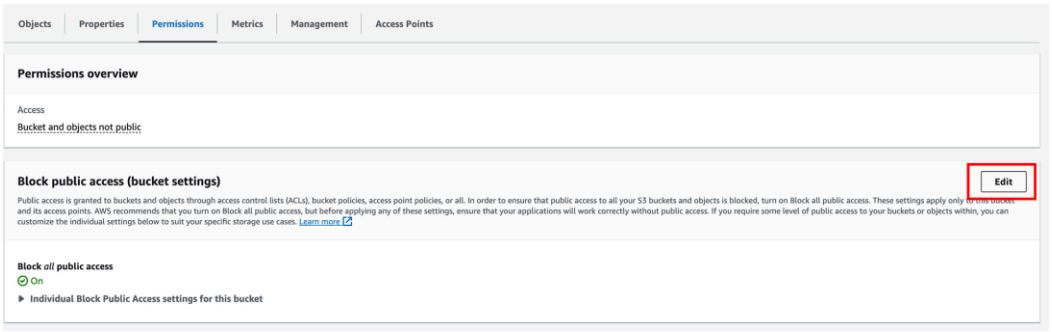
4) Configure the bucket for serving the images

Now that you uploaded those dummy images, it's time to configure the bucket such that the images can be loaded from the NextJS website.

Because, by default, this is **not possible**! By default, S3 buckets are "locked down" and the files in there are secure & not accessible by anyone else.

But for our purposes here, we must update the bucket settings to make sure the images can be viewed by everyone.

To do that, as a first step, click on the "Permissions" tab and "Edit" the "Block public access" setting:



Then, disable the "Block all public access" checkbox (and with it, all other checkboxes) and select "Save Changes".

Type "confirm" into the confirmation overlay once it pops up.

That's not all though - as a next (and final step), you must add a so-called "Bucket Policy". That's an AWS-specific policy document that allows you to manage the permissions of the objects stored in the bucket.

You can add such a "Bucket Policy" right below the "Block all public access" area, still on the "Permissions" tab:

Click "Edit" and insert the following bucket policy into the box:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "PublicRead",
            "Effect": "Allow",
            "Principal": "*",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": [
                "arn:aws:s3:::DOC-EXAMPLE-BUCKET/*"
            ]
        }
    ]
}
```

Replace `DOC-EXAMPLE-BUCKET` with your bucket name (*maxschwarzmueller-nextjs-demo-users-image* in my case).

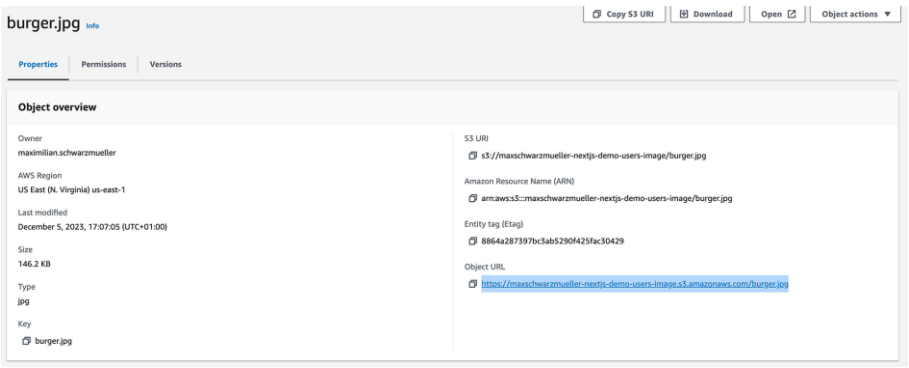
Then, click "Save Changes".

Now the bucket is configure to grant access to all objects inside of it to anyone who has a URL pointing to one of those objects.

Therefore, you should now of course not add any files into the bucket that you don't want to share with the world!

To test if everything works, click on one of the images you uploaded (in the bucket).

Then click on the "Object URL" - if opening it works (and you can see the image), you configured everything as needed.



5) Update the NextJS code to use those S3 images

Now that the images are stored + served via S3, it's time to also load them from there in your NextJS app.

As a first step, you can delete the `public/images` folder (so that an empty `public/` folder remains).

Now, if you also delete the `.next` folder in the NextJS project and you then visit `localhost:3000/meals`, you should see a bunch of meals without images.

To bring them back, as a first step, edit the database data by updating the `initdb.js` file: Change all the image property values from `image: '/images/burger.jpg'`, to `image: 'burger.jpg'` (and do that for all meals).

Alternatively, you find an updated `initdb.js` file attached.

Next, go to the `components/meals/meal-item.js` file (which contains the `MealItem` component) and update the `<Image>` `src`:

```
<Image
  src={`https://maxschwarzmueller-nextjs-demo-users-image.s3.amazonaws.com/${image}`}
  alt={title}
  fill
/>
```

Of course, use your S3 URL / bucket name!

The new `src` value is a string that contains the S3 URL to your bucket objects (i.e., the URL you previously clicked for testing purposes - without the image file name at the end). The actual image name that should be loaded is then dynamically inserted via `${image}`.

Note: This will only work if the images stored in the S3 bucket have the names referenced in the `initdb.js` file!

You should also update the `app/meals/[mealSlug]/page.js` file and make sure that the image on this page is also fetched from S3:

```
<Image
  src={`https://maxschwarzmueller-nextjs-demo-users-image.s3.amazonaws.com/${meal.image}`}
  alt={meal.title}
  fill
/>
```

Of course, use your S3 URL / bucket name!

Now, to reset the database data, you should delete your `meals.db` file (i.e., delete the SQLite database file) and re-run `node initdb.js` to re-initialize it (with the updated image values).

If you do that, and you then restart the development server (`npm run dev`), you'll notice that you now get an error when visiting the `/meals` page:

```
Error: Invalid src prop (https://maxschwarzmueller-nextjs-demo-users-image.s3.amazonaws.com/burger.jpg) on `next/image`, hostname "maxschwarzmueller-nextjs-demo-users-image.s3.amazonaws.com" is not configured under images in your `next.config.js`
```

6) Allowing S3 as an image source

You get this error because, by default, NextJS does not allow external URLs when using the `<Image>` component.

You explicitly have to allow such a URL in order to get rid of this error.

That's done by editing the `next.config.js file`:

```
const nextConfig = {
  images: {
    remotePatterns: [
      {
        protocol: 'https',
        hostname: 'maxschwarzmueller-nextjs-demo-users-image.s3.amazonaws.com',
        port: '',
        pathname: '/**',
      },
    ],
  },
};
```

Of course, use your S3 URL / bucket name!

This `remotePatterns` config allows this specific S3 URL as a valid source for images.

With the config file updated + saved, you should now be able to visit `/meals` and see all those images again.

7) Storing uploaded images on S3

Now that we can see those dummy images again, it's finally time to also "forward" user-generated (i.e., uploaded) images to S3.

This can be done with help of a package provided by AWS - the `@aws-sdk/client-s3` package. This package provides functionalities that allow you to interact with S3 - e.g., to store files in a specific bucket.

Install that package via `npm install @aws-sdk/client-s3`.

Then, go to your `lib/meals.js` file and import the AWS S3 SDK (at the top of the file):

```
import { S3 } from '@aws-sdk/client-s3';
```

Next, initialize it by adding this line (e.g., right above the line where the db object is created):

```
const s3 = new S3({
  region: 'us-east-1'
});
const db = sql('meals.db'); // <- this was already there!
```

Almost there!

Now, edit the `saveMeal()` function and remove all code that was related to storing the image on the local file system.

Instead, add this code:

```
s3.putObject({
  Bucket: 'maxschwarzmueller-nextjs-demo-users-image',
  Key: fileName,
  Body: Buffer.from(bufferedImage),
  ContentType: meal.image.type,
});
```

Of course, use your S3 URL / bucket name!

Also make sure to save the image filename under `meal.image`:

```
meal.image = fileName;
```

The final `saveMeal()` function should look like this:

```
export async function saveMeal(meal) {
  meal.slug = slugify(meal.title, { lower: true });
  meal.instructions = xss(meal.instructions);

  const extension = meal.image.name.split('.').pop();
  const fileName = `${meal.slug}.${extension}`;

  const bufferedImage = await meal.image.arrayBuffer();

  s3.putObject({
    Bucket: 'maxschwarzmueller-nextjs-demo-users-image',
    Key: fileName,
    Body: Buffer.from(bufferedImage),
    ContentType: meal.image.type,
  });


  meal.image = fileName;

  db.prepare(
```

```
    `
    INSERT INTO meals
      (title, summary, instructions, creator, creator_email, image, slug)
    VALUES (
      @title,
      @summary,
      @instructions,
      @creator,
      @creator_email,
      @image,
      @slug
    )
  `
  ).run(meal);
}
```

8) Granting the NextJS backend AWS access permissions

Now, there's just one last, yet very important, step missing: Granting your NextJS app S3 access permissions.

We did configure S3 to serve the bucket content to everyone.

But we did not (and should not!) configure it to allow everyone to write to the bucket or change the bucket contents.

But that's what our NextJS app (via the S3 AWS SDK) now tries to do!

To grant our app appropriate permissions, you must set up AWS access keys for your app.

This is done by adding a `.env.local` file to your root NextJS project. This file will automatically be read by NextJS and the environment variables configured in there will be made available to the backend (!) part of your app.

You can learn more about setting up environment variables for NextJS apps here: https://nextjs.org/docs/app/building-your-application/configuring/environment-variables.

In this `.env.local` file, you must add two key-value pairs:

```
AWS_ACCESS_KEY_ID=<your aws access key>
AWS_SECRET_ACCESS_KEY=<your aws secret access key>
```

You get those access keys from inside the AWS console (in the browser). You can get them by clicking on your account name (in the top right corner of the AWS console) and then "Security Credentials".

Scroll down to the "Access Keys" area and create a new Access Key. Copy & paste the values into your `.env.local` file and **never share these keys with anyone!** Don't commit them to Git or anything like that!

You can learn more about them here: https://docs.aws.amazon.com/IAM/latest/UserGuide/id_credentials_access-keys.html

With all that done, finally, you should be able to create new meals, upload images and see them on `/meals`. Even in production! Because now, the images are stored on S3!

You find the finished, adjusted code attached to this lecture. Please note that the `.env.local` file is not included - you must add it (and use your own credentials) if you want to run the attached code.