Using S3 as a Web Server

In this lesson, you will learn how to use S3 as a web server by getting familiar with bucket endpoints and website endpoints.

WE'LL COVER THE FOLLOWING

There are two ways of using S3 as a web server:

- bucket endpoints
- website endpoints

Bucket endpoints and website endpoints

A *bucket endpoint* allows direct access to S3 objects using HTTPS. AWS automatically activates this endpoint when you create an S3 bucket. For example, a file called `test.txt` in the bucket `gojko` will be available from https://gojko.s3.amazonaws.com/test.txt. Access to the bucket endpoint is controlled by IAM. When uploading a file to S3, you can make it publicly readable (as was done with `test.txt`), so anyone can access it using a web browser. You can also mark the file as private, so others will need a pre-signed download policy to access it. You used this approach for conversion results in the previous chapter.

A *website endpoint* is an optional feature of S3 that can perform some basic web workflows, such as redirecting users or showing index or error pages. This endpoint has a different URL from the bucket endpoint, usually a subdomain of `s3-website-us-east-1.amazonaws.com` in the `us-east-1` region, or a similar service for other regions. To use a website endpoint, you need to activate it after creating the bucket. You can activate a website endpoint for an S3 bucket from the AWS Web Console, using the *Static web site hosting* section of the bucket properties page. Of course, it's possible to activate it

using CloudFormation as well, using the `WebsiteConfiguration` properties of a bucket.

> ### Set a custom domain name
>
> S3 will automatically assign a domain name to a website endpoint. It's not possible to set a custom domain name, but you can put a CDN between the users and the website endpoint and configure a custom domain name in the CDN. This is the usual approach for creating nice web domains for serverless applications.

Requests for the bucket endpoint only work with URLs matching an exact resource path, so asking for the root object (`/`) will not automatically show `index.html`. You can set up the website endpoint to send back an index file if users ask for the root object.

Another major difference between website endpoints and bucket endpoints is that website endpoints work using HTTP, while bucket endpoints work using HTTPS. This might sound like a security problem, but for real-world production usage, you should put a content distribution network in front of the S3 website endpoint, so the CDN provides HTTPS access to users. If you use Amazon's CDN, then the CDN talks to the S3 website endpoint only internally in the AWS network, so the risk of man-in-the-middle attacks between the CDN and the origin isn't that big.

You'll need another bucket to host the files from your `web-site` directory. Users will expect to see a landing page when they type a domain in their web browsers, so you should create a new S3 bucket for static assets and set up a website endpoint. Another resource is added to the `Resources` section on the template, with the code from the following listing (indented so that `WebAssetsS3Bucket` aligns with other resource names, such as `UploadS3Bucket`).

```
WebAssetsS3Bucket:
  Type: AWS::S3::Bucket
  Properties:
    WebsiteConfiguration:
      ErrorDocument: 404.html
      IndexDocument: index.html
```

Line 26 to Line 31 of code/ch11/template.yaml

Notice the two files referenced in lines 5 and 6. That is how the website endpoint knows how to respond to root object requests and what to send back in case of missing files. You already have the index file, so a simple error page is added to your `web-site` directory. It will be called `404.html`.

```
<html>
  <body>
    <h1>Page not found</h1>
  </body>
</html>
```

code/ch11/web-site/404.html

When you configure a website endpoint for a bucket, AWS creates a separate website URL, which you can access in the SAM template using the `WebsiteURL` property. To be able to discover the website after you deploy the application, let's create another output in the application template. The code from the following listing is added to the `Outputs` section of your template, indented so that `WebUrl` aligns with other outputs.

```
WebUrl:
  Description: "Public web URL"
  Value: !GetAtt WebAssetsS3Bucket.WebsiteURL
```

Line 177 to Line 179 of code/ch11/template.yaml

You're ready to move on to the next lesson where you can learn about cross-origin resource sharing!