7 Storing your objects: S3

This chapter covers

- Transferring files to S3 using the terminal
- Integrating S3 into your applications with SDKs
- Archiving data at low costs
- Hosting a static website with S3
- Diving into the internals of the S3 object store

Storing data comes with two challenges: dealing with ever-increasing volumes of data and ensuring durability. Solving the challenges is hard, or even impossible, if using disks connected to a single machine. For this reason, this chapter covers a revolutionary approach: a distributed data store consisting of a large number of machines connected over a network. This way, you can store nearly unlimited amounts of data by adding additional machines to the distributed data store. And because your data is always stored on more than one machine, you dramatically reduce the risk of losing that data.

You will learn about how to store images, videos, documents, executables, or any other kind of data on Amazon S3 in this chapter. Amazon S3 is a simple-to-use, fully managed distributed data store provided by AWS. Data is managed as objects, so the storage system is called an *object store*. We will show you how to use S3 to back up your data, how to archive data at low cost, and how to integrate S3 into your own application for storing user-generated content, as well as how to host static websites on S3.

Not all examples are covered by the Free Tier

The examples in this chapter are not all covered by the Free Tier. A warning message appears when an example incurs costs. As for the other examples, as long as you follow the instructions and don't run them longer than a few days, you won't pay anything.

7.1 What is an object store?

Back in the old days, data was managed in a hierarchy consisting of folders and files. The file was the representation of the data. In an *object store*, data is stored as objects. Each object consists of a globally unique identifier, some metadata, and the data itself, as figure 7.1 illustrates. An object's *globally unique identifier* (GUID) is also known as its *key*; you can

address the object from different devices and machines in a distributed system using the GUID.

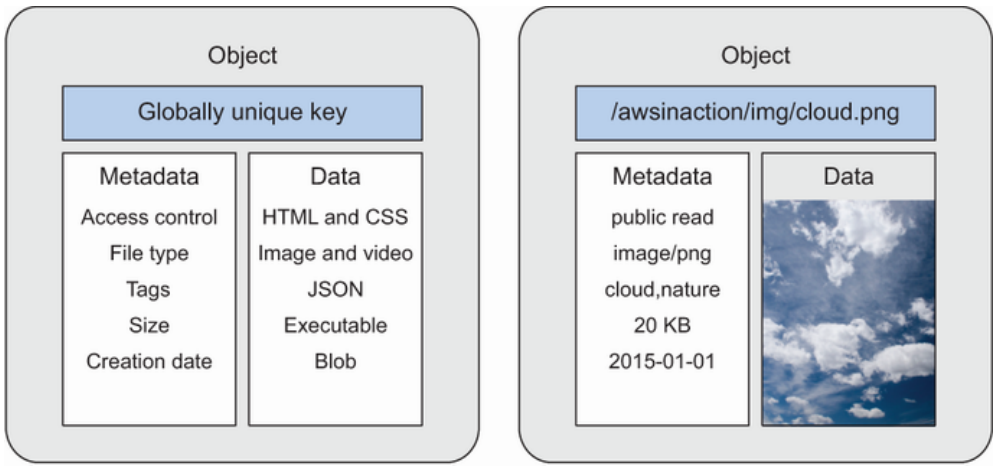

Figure 7.1 Objects stored in an object store have three parts: a unique ID, metadata describing the content, and the content itself (such as an image).

Typical examples for object metadata are:

- Date of last modification
- Object size
- Object's owner
- Object's content type

It is possible to request only an object's metadata without requesting the data itself. This is useful if you want to list objects and their metadata before accessing a specific object's data.

7.2 Amazon S3

Amazon S3 is a distributed data store, and one of the oldest services provided by AWS. *Amazon S3* is an acronym for *Amazon Simple Storage Service*. It's a typical web service that lets you store and retrieve data organized as objects via an API reachable over HTTPS. Here are some typical use cases:

- *Storing and delivering static website content* —For example, our blog **https://cloudonaut.io** is hosted on S3.
- *Backing up data*—For example, you can back up your photo library from your computer to S3 using the AWS CLI.
- *Storing structured data for analytics, also called a data lake*—For example, you can use S3 to store JSON files containing the results of performance benchmarks.
- *Storing and delivering user-generated content*—For example, we built a web application—with the help of the AWS SDK—that stores user uploads on S3.

Amazon S3 offers virtually unlimited storage space and stores your data in a highly available and durable way. You can store any kind of data, such as images, documents, and binaries, as long as the size of a single object doesn't exceed 5 TB. You have to pay for every GB you store in S3, and you also incur costs for every request and for all transferred data. As figure 7.2 shows, you can access S3 via the internet using HTTPS to upload and download objects. To access S3, you can use the Management Console, the CLI, SDKs, or third-party tools.

SFigure 7.2 Uploading and downloading an object to S3 via HTTPS

3 uses *buckets* to group objects. A bucket is a container for objects. It is up to you to create multiple buckets, each of which has a globally unique name, to separate data for different scenarios. By *unique*, we really mean unique—you have to choose a bucket name that isn't used by any other AWS customer in any other region. Figure 7.3 shows the concept.

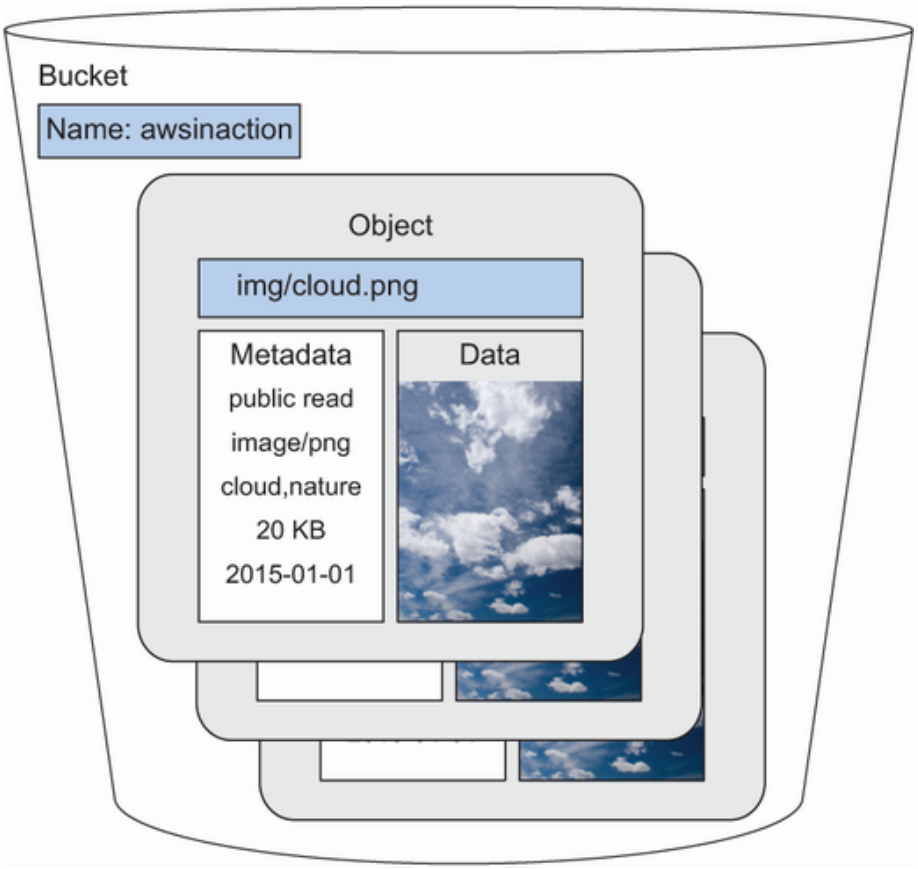

Figure 7.3 S3 uses buckets with globally unique names to group objects.

You will learn how to upload and download data to S3 using the AWS CLI next.

7.3 Backing up your data on S3 with AWS CLI

Critical data needs to be backed up to avoid loss. Backing up data at an offsite location decreases the risk of losing data, even during extreme conditions like natural disaster. But where should you store your backups? S3 allows you to store any data in the form of objects. The AWS object store is a perfect fit for your backup, allowing you to choose a location for your data as well as storing any amount of data with a pay-per-use pricing model.

In this section, you'll learn how to use the AWS CLI to upload data to and download data from S3. This approach isn't limited to offsite backups; you can use it in many other scenarios as well, such as the following:

- Sharing files with your coworkers or partners, especially when working from different locations
- Storing and retrieving artifacts needed to provision your virtual machines (such as application binaries, libraries, or configuration files)
- Outsourcing storage capacity to lighten the burden on local storage systems—in particular, for data that is accessed infrequently

First, you need to create a bucket for your data on S3. As we mentioned earlier, the name of the bucket must be unique among all other S3 buckets, even those in other regions and those of other AWS customers. To find a unique bucket name, it's useful to use a prefix or suffix that includes your company's name or your own name. Run the following command in the terminal, replacing `$yourname` with your name:

```
$ aws s3 mb s3://awsinaction-$yourname
```

Your command should look similar to this one:

```
$ aws s3 mb s3://awsinaction-awittig
```

In the unlikely event that you or another AWS customer has already created a bucket with this name, you will see the following error:

```
[... ] An error occurred (BucketAlreadyExists) [...]
```

In this case, you'll need to use a different value for `$yourname`.

Everything is ready for you to upload your data. Choose a folder you'd like to back up, such as your Desktop folder. Try to choose a folder with a total size less than 1 GB, to avoid long waiting times and exceeding the

Free Tier. The following command uploads the data from your local folder to your S3 bucket. Replace `$path` with the path to your folder and `$yourname` with your name. `sync` compares your folder with the /backup folder in your S3 bucket and uploads only new or changed files:

```
$ aws s3 sync $path s3://awsinaction-$yourname/backup
```

Your command should look similar to this one:

```
$ aws s3 sync /Users/andreas/Desktop s3://awsinaction-awittig/backup
```

Depending on the size of your folder and the speed of your internet connection, the upload can take some time.

After uploading your folder to your S3 bucket to back it up, you can test the restore process. Execute the following command in your terminal, replacing `$path` with a folder you'd like to use for the restore (don't use the folder you backed up) and `$yourname` with your name. Your Downloads folder would be a good place to test the restore process:

```
$ aws s3 cp --recursive s3://awsinaction-$yourname/backup $path
```

Your command should look similar to this one:

```
$ aws s3 cp --recursive s3://awsinaction-awittig/backup/ \
➥ /Users/andreas/Downloads/restore
```

Again, depending on the size of your folder and the bandwidth of your internet connection, the download may take a while.

Versioning for objects

By default, S3 versioning is disabled for every bucket. Suppose you use the following steps to upload two objects:

1. Add an object with key A and data 1.
2. Add an object with key A and data 2.

If you download the object with key A, you'll download data 2. The old data 1 doesn't exist any more.

You can change this behavior by turning on versioning for a bucket. The following command activates versioning for your bucket. Don't forget to replace `$yourname`:

```
$ aws s3api put-bucket-versioning --bucket awsinaction-$yourname \
➥ --versioning-configuration Status=Enabled
```

If you repeat the previous steps, the first version of object A consisting of data 1 will be accessible even after you add an object with key A and data 2. The following command retrieves all objects and versions:

```
$ aws s3api list-object-versions --bucket awsinaction-$yourname
```

You can now download all versions of an object.

Versioning can be useful for backing up and archiving scenarios. Keep in mind that the size of the bucket you'll have to pay for will grow with every new version.
You no longer need to worry about losing data. S3 is designed for 99.999999999% durability of objects over a year. For instance, when storing 100,000,000,000 objects on S3, you will lose only a single object per year on average.

After you've successfully restored your data from the S3 bucket, it's time to clean up. Execute the following command to remove the S3 bucket containing all the objects from your backup. You'll have to replace `$yourname` with your name to select the right bucket. `rb` removes the bucket; the `force` option deletes every object in the bucket before deleting the bucket itself:

```
$ aws s3 rb --force s3://awsinaction-$yourname
```

Your command should look similar to this one:

```
$ aws s3 rb --force s3://awsinaction-awittig
```

You're finished—you've uploaded and downloaded files to S3 with the help of the CLI.

Removing a bucket causes a `BucketNotEmpty` error

If you turn on versioning for your bucket, removing the bucket will cause a `BucketNotEmpty` error. Use the Management Console to delete the bucket in this case as follows:

1. Open the Management Console with your browser.
2. Go to the S3 service using the main navigation menu.
3. Select the bucket you want to delete.

4. Click the Empty button, and confirm permanently deleting all objects.
5. Wait until objects and versions have been deleted, and click the Exit button.
6. Select the bucket you want to delete.
7. Click the Delete button, and confirm deleting the bucket.

7.4 Archiving objects to optimize costs

In the previous section, you learned about backing up your data to S3. Storing 1 TB of data on S3 costs about $23 per month. Wouldn't it be nice to reduce the costs for storing data by 95%? Besides, by default, S3 comes with storage classes designed to archive data for long time spans.

Table 7.1 compares the storage class *S3 Standard* with storage classes intended for data archival.

Table 7.1 Differences between storing data with S3 and Glacier

	S3 Standard	S3 Glacier Instant Retrieval	S3 Glacier Flexible Retrieval	S3 Glacier Deep Archive
Storage costs for 1 GB per month in US East (N. Virginia)	$0.023	$0.004	$0.0036	$0.00099
Costs for 1,000 write requests	$0.005	$0.02	$0.03	$0.05
Costs for retrieving data	Low	High	High	Very High
Accessibility	Milliseconds	Milliseconds	1–5 minutes/3–5 hours/5–12 hours	12 hours/48 hours
Durability objective	99.999999999%	99.999999999%	99.999999999%	99.999999999%
Availability objective	99.99%	99.9%	99.99%	99.99%

The potential savings for storage costs are enormous. So what's the catch?

First, accessing data stored on S3 by using the storage classes S3 Glacier Instant Retrieval, S3 Glacier Flexible Retrieval, and S3 Glacier Deep Archive is expensive. Let's assume, you are storing 1 TB of data on S3 and decided to use storage type S3 Glacier Deep Archive. It will cost you about $120 to restore 1 TB of data stored in 1,000 files.

Second, fetching data from S3 Glacier Flexible Retrieval and S3 Glacier Deep Archive takes something between 1 minute and 48 hours, depending on the storage class and retrieval option.

Using the following example, we would like to explain what it means not to be able to access archived data immediately. Let's say you want to archive a document for five years. You do not expect to access the document more than five times during this period.

Example not covered by Free Tier

The following example is not covered by the Free Tier. Archiving and restoring data as shown in the example will cost you less than $1. You will find information on how to delete all resources at the end of the section. Therefore, we recommend completing this section within a few days

Start by creating an S3 bucket that you will use to archive documents, as shown next. Replace `$yourname` with your name to get a unique bucket name:

```
$ aws s3 mb s3://awsinaction-archive-$yourname
```

Next, copy a document from your local machine to S3. The `--storage-class` parameter overrides the default storage class with `GLACIER`, which maps to the S3 Glacier Flexible Retrieval storage class. Replace `$path` with the path to a document, and `$yourname` with your name. Note the key of the object:

```
$ aws s3 cp --storage-class GLACIER $path \
➥ s3://awsinaction-archive-$yourname/
```

For instance, I run the following command:

```
$ aws s3 cp --storage-class GLACIER \
➥ /Users/andreas/Desktop/taxstatement-2022-07-01.pdf \
➥ s3://awsinaction-archive-awittig/
```

The key point is that you can't download the object. Replace `$objectkey` with the object's key that you noted down after uploading the document, and `$path` with the Downloads folder on your local machine:

```
$ aws s3 cp s3://awsinaction-archive-$yourname/$objectkey $path
```

For example, I'm getting the following error when trying to download my document taxstatement-2022-07-01.pdf:

```
$ aws s3 cp s3://awsinaction-archive-awittig/taxstatement-2022-07-01.pdf
➥ ~/Downloads
warning: Skipping file s3://awsinaction-archive-awittig/
➥ taxstatement-2022-07-01.pdf. Object is of storage class GLACIER.
➥ Unable to perform download operations on GLACIER objects. You must
➥ restore the object to be able to perform the operation.
```

As mentioned in the error message, you need to restore the object before downloading it. By default, doing so will take three to five hours. That's why we will pay a little extra—just a few cents—for expedited retrieval. Execute the following command after replacing `$yourname` with your name, and `$objectkey` with the object's key:

```
$ aws s3api restore-object --bucket awsinaction-archive-$yourname \
➥ --key $objectkey \
➥ --restore-request Days=1,,GlacierJobParameters={"Tier"="Expedited"}
```

This results in the following command in my scenario:

```
$ aws s3api restore-object --bucket awsinaction-archive-awittig \
➥ --key taxstatement-2022-07-01.pdf
➥ --restore-request Days=1,,GlacierJobParameters={"Tier"="Expedited"}
```

As you are using expedited retrieval, you need to wait one to five minutes for the object to become available for download. Use the following command to check the status of the object and its retrieval. Don't forget to replace `$yourname` with your name, and `$objectkey` with the object's key:

```
$ aws s3api head-object --bucket awsinaction-archive-$yourname \
➥ --key $objectkey
{
    "AcceptRanges": "bytes",
    "Expiration": "expiry-date=\"Wed, 12 Jul 2023 ...\", rule-id=\"...\"
    "Restore": "ongoing-request=\"true\"",
    "LastModified": "2022-07-11T09:26:12+00:00",
```

```
    "ContentLength": 112,
    "ETag": "\"c25fa1df1968993d8e647c9dcd352d39\"",
    "ContentType": "binary/octet-stream",
    "Metadata": {},
    "StorageClass": "GLACIER"
}
```

① Restoration of the object is still ongoing.

Repeat fetching the status of the object until `ongoing-request` flips to `false`:

```
{
    "AcceptRanges": "bytes",
    "Expiration": "expiry-date=\"Wed, 12 Jul 2023 ...\", rule-id=\"...\"
    "Restore": "ongoing-request=\"false\", expiry-date=\"...\"",
    "LastModified": "2022-07-11T09:26:12+00:00",
    "ContentLength": 112,
    "ETag": "\"c25fa1df1968993d8e647c9dcd352d39\"",
    "ContentType": "binary/octet-stream",
    "Metadata": {},
    "StorageClass": "GLACIER"
}
```

① The restoration is finished, with no ongoing restore requests.

After restoring the object, you are now able to download the document using the next code snippet. Replace `$objectkey` with the object's key that you noted down after uploading the document, and `$path` with the Downloads folder on your local machine:

```
$ aws s3 cp s3://awsinaction-archive-$yourname/$objectkey $path
```

In summary, the Glacier storage types are intended for archiving data that you need to access seldom, which means every few months or years. For example, we are using the S3 Glacier Deep Archive to store a remote backup of our MacBooks. Because we store another backup of our data on an external hard drive, the chances that we need to restore data from S3 are very low.

> Cleaning up
> Execute the following command to remove the S3 bucket containing all the objects from your backup. You'll have to replace `$yourname` with your name to select the right bucket. `rb` removes the bucket; the `force` option deletes every object in the bucket before deleting the bucket itself:

```
$ aws s3 rb --force s3://awsinaction-archive-$yourname
```

You've learned how to use S3 with the help of the CLI. We'll show you how to integrate S3 into your applications with the help of SDKs in the next section.

7.5 Storing objects programmatically

S3 is accessible using an API via HTTPS. This enables you to integrate S3 into your applications by making requests to the API programmatically. Doing so allows your applications to benefit from a scalable and highly available data store. AWS offers free SDKs for common programming languages like C++, Go, Java, JavaScript, .NET, PHP, Python, and Ruby. You can execute the following operations using an SDK directly from your application:

- Listing buckets and their objects
- Creating, removing, updating, and deleting (CRUD) objects and buckets
- Managing access to objects

Here are examples of how you can integrate S3 into your application:

- *Allow a user to upload a profile picture.* Store the image in S3, and make it publicly accessible. Integrate the image into your website via HTTPS.
- *Generate monthly reports (such as PDFs), and make them accessible to users.* Create the documents and upload them to S3. If users want to download documents, fetch them from S3.
- *Share data between applications.* You can access documents from different applications. For example, application A can write an object with the latest information about sales, and application B can download the document and analyze the data.

Integrating S3 into an application is one way to implement the concept of a *stateless server*. We'll show you how to integrate S3 into your application by diving into a simple web application called Simple S3 Gallery next. This web application is built on top of Node.js and uses the AWS SDK for JavaScript and Node.js. You can easily transfer what you learn from this example to SDKs for other programming languages; the concepts are the same.

Installing and getting started with Node.js

Node.js is a platform for executing JavaScript in an event-driven environment so you can easily build network applications. To install Node.js, visit **https://nodejs.org** and download the package that fits your OS. All examples in this book are tested with Node.js 14.

After Node.js is installed, you can verify that everything works by typing `node --version` into your terminal. Your terminal should respond with something similar to `v14.*`. Now you're ready to run JavaScript examples like the Simple S3 Gallery.

Do you want to get started with Node.js? We recommend *Node.js in Action* (second edition) by Alex Young et al. (Manning, 2017), or the video course *Node.js in Motion* by PJ Evans (Manning, 2018).
Next, we will dive into a simple web application called the Simple S3 Gallery. The gallery allows you to upload images to S3 and displays all the images you've already uploaded. Figure 7.4 shows Simple S3 Gallery in action. Let's set up S3 to start your own gallery.

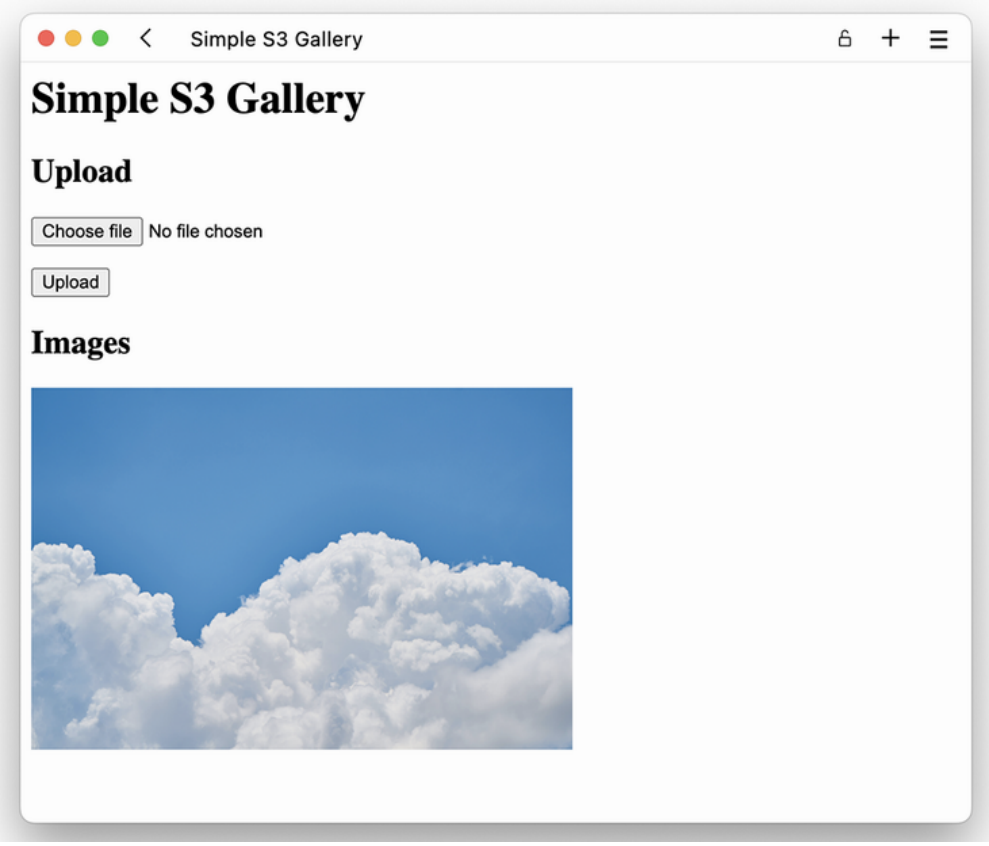

Figure 7.4 The Simple S3 Gallery app lets you upload images to an S3 bucket and then download them from the bucket for display.

7.5.1 Setting up an S3 bucket

To begin, you need to set up an empty bucket. Execute the following command, replacing `$yourname` with your name:

```
$ aws s3 mb s3://awsinaction-sdk-$yourname
```

Your bucket is now ready to go. Installing the web application is the next step.

7.5.2 Installing a web application that uses S3

You can find the Simple S3 Gallery application in /chapter07/gallery/ in the book's code folder. Switch to that directory, and run `npm install` in your terminal to install all needed dependencies.

To start the web application, run the following command. Replace `$yourname` with your name; the name of the S3 bucket is then passed to the web application:

```
$ node server.js awsinaction-sdk-$yourname
```

Where is the code located?

You can find all the code in the book's code repository on GitHub: **https://github.com/AWSinAction/code3**. You can download a snapshot of the repository at **https://github.com/AWSinAction/code3/archive/main.zip**.

After you start the server, you can open the gallery application. To do so, open http://localhost:8080 with your browser. Try uploading a few images.

7.5.3 Reviewing code access S3 with SDK

You've uploaded images to the Simple S3 Gallery and displayed images from S3. Inspecting parts of the code will help you understand how you can integrate S3 into your own applications. It's not a problem if you don't follow all the details of the programming language (JavaScript) and the Node.js platform; we just want you to get an idea of how to use S3 via SDKs.

Uploading an image to S3

You can upload an image to S3 with the SDK's `putObject()` function. Your application will connect to the S3 service and transfer the image via HTTPS. The next listing shows how to do so.

Listing 7.1 Uploading an image with the AWS SDK for S3

```
const AWS = require('aws-sdk');                                    ①
const uuid = require('uuid');
```

```
const s3 = new AWS.S3({                                    ②
  'region': 'us-east-1'
});
const bucket = process.argv[2];

async function uploadImage(image, response) {
  try {
    await s3.putObject({                                   ③
      Body: image,                                         ④
      Bucket: bucket,                                      ⑤
      Key: uuid.v4(),                                      ⑥
      ACL: 'public-read',                                  ⑦
      ContentLength: image.byteCount,                      ⑧
      ContentType: image.headers['content-type']           ⑨
    }).promise();
    response.redirect('/');
  } catch (err) {                                          ⑩
    console.error(err);
    response.status(500);
    response.send('Internal server error.');               ⑪
  }
}
```

① Loads the AWS SDK

② Instantiates the S3 client with additional configurations

③ Uploads the image to S3

④ Image content

⑤ Name of the bucket

⑥ Generates a unique key for the object

⑦ Allows everybody to read the image from bucket

⑧ Size of image in bytes

⑨ Content type of the object (image/png)

⑩ Catching errors

⑪ Returns an error with the HTTP status code 500

The AWS SDK takes care of sending all the necessary HTTPS requests to the S3 API in the background.

To display a list of images, the application needs to list all the objects in your bucket. This can be done with the S3 service's `listObjects()` function. The next code listing shows the implementation of the corresponding function in the server.js JavaScript file, acting as a web server.

Listing 7.2 Retrieving all the image locations from the S3 bucket

```
const bucket = process.argv[2];                    ①

async function listImages(response) {
  try {
    let data = await s3.listObjects({              ②
      Bucket: bucket                               ③
    }).promise();
    let stream = mu.compileAndRender(              ④
      'index.html',
      {
        Objects: data.Contents,
        Bucket: bucket
      }
    );
    stream.pipe(response);                         ⑤
  } catch (err) {                                  ⑥
    console.error(err);
    response.status(500);
    response.send('Internal server error.');
  }
}
```

① Reads the bucket name from the process arguments

② Lists the objects stored in the bucket

③ The bucket name is the only required parameter.

④ Renders an HTML page based on the list of objects

⑤ Streams the response

⑥ Handles potential errors

Listing the objects returns the names of all the images from the bucket, but the list doesn't include the image content. During the uploading process, the access rights to the images are set to public read. This means anyone can download the images with the bucket name and a random key. The following listing shows an excerpt of the index.html template,

which is rendered on request. The `Objects` variable contains all the objects from the bucket.

Listing 7.3 Template to render the data as HTML

```
[...]
<h2>Images</h2>
{{#Objects}}          ①
  <p><img src=        ②
➥ "https:/ /s3.amazonaws.com/{{Bucket}}/{{Key}}"
➥ width="400px" ></p>
{{/Objects}}
[...]
```

① Iterates over all objects

② Puts together the URL to fetch an image from the bucket

You've now seen the three important parts of the Simple S3 Gallery integration with S3: uploading an image, listing all images, and downloading an image.

> Cleaning up
> Don't forget to clean up and delete the S3 bucket used in the example. Use the following command, replacing `$yourname` with your name:
>
> ```
> $ aws s3 rb --force s3://awsinaction-sdk-$yourname
> ```

You've learned how to use S3 using the AWS SDK for JavaScript and Node.js. Using the AWS SDK for other programming languages is similar.

The next section is about a different scenario: you will learn how to host static websites on S3.

7.6 Using S3 for static web hosting

We started our blog **https://cloudonaut.io** in May 2015. The most popular blog posts, like "ECS vs. Fargate: What's the Difference?" (**http://mng.bz/Xa2E**), "Advanced AWS Networking: Pitfalls That You Should Avoid" (**http://mng.bz/yaxe**), and "CloudFormation vs. Terraform" (**https://cloudonaut.io/cloudformation-vs-terraform/**) have been read more than 200,000 times. But we did not need to operate any VMs to publish our blog posts. Instead, we used S3 to host our static website built

with a static site generator, Hexo (**https://hexo.io**). This approach provides a cost-effective, scalable, and maintenance-free infrastructure for our blog.

You can host a static website with S3 and deliver static content like HTML, JavaScript, CSS, images (such as PNG and JPG), audio, and videos. Keep in mind, however, that you can't execute server-side scripts like PHP or JSP. For example, it's not possible to host WordPress, a content management system based on PHP, on S3.

Increasing speed by using a CDN

Using a content-delivery network (CDN) helps reduce the load time for static web content. A CDN distributes static content like HTML, CSS, and images to nodes all around the world. If a user sends out a request for some static content, the request is answered from the nearest available node with the lowest latency. Various providers offer CDNs. Amazon CloudFront is the CDN offered by AWS. When using CloudFront, users connect to CloudFront to access your content, which is fetched from S3 or other sources. See the CloudFront documentation at **http://mng.bz/M0m8** if you want to set this up; we won't cover it in this book.

In addition, S3 offers the following features for hosting a static website:

- *Defining a custom index document and error documents*—For example, you can define index.html as the default index document.
- *Defining redirects for all or specific requests*—For example, you can forward all requests from /img/old.png to /img/new.png.
- *Setting up a custom domain for an S3 bucket*—For example, Andreas might want to set up a domain like mybucket.andreaswittig.info pointing to his bucket.

7.6.1 Creating a bucket and uploading a static website

First you need to create a new S3 bucket. To do so, open your terminal and execute the following command, replacing `$BucketName` with your own bucket name. As we've mentioned, the bucket name has to be globally unique. If you want to link your domain name to S3, you must use your entire domain name as the bucket name:

```
$ aws s3 mb s3://$BucketName
```

The bucket is empty; you'll place an HTML document in it next. We've prepared a placeholder HTML file. Download it to your local machine from the following URL: **http://mng.bz/aPyX**. You can now upload the file

to S3. Execute the following command to do so, replacing `$pathToPlaceholder` with the path to the HTML file you downloaded in the previous step and `$BucketName` with the name of your bucket:

```
$ aws s3 cp $pathToPlaceholder/helloworld.html \
➥ s3://$BucketName/helloworld.html
```

You've now created a bucket and uploaded an HTML document called helloworld .html. You need to configure the bucket next.

7.6.2 Configuring a bucket for static web hosting

By default, only you, the owner, can access files from your S3 bucket. Because you want to use S3 to deliver your static website, you'll need to allow everyone to view or download the documents included in your bucket. A *bucket policy* helps you control access to bucket objects globally. You already know from chapter 5 that policies are defined in JSON and contain one or more statements that either allow or deny specific actions on specific resources. Bucket policies are similar to IAM policies.

Download our bucket policy from the following URL: **http://mng.bz/gROG**. You need to edit the bucketpolicy.json file next, as shown in the following listing. Open the file with the editor of your choice, and replace `$BucketName` with the name of your bucket.

Listing 7.4 Bucket policy allowing read-only access to every object in a bucket

```
{
  "Version":"2012-10-17",
  "Statement":[
    {
      "Sid":"AddPerm",
      "Effect":"Allow",                                  ①
      "Principal": "*",                                  ②
      "Action":["s3:GetObject"],                         ③
      "Resource":["arn:aws:s3:::$BucketName/*"]          ④
    }
  ]
}
```

① Allows access...

② ...for anyone

③ Reads objects

④ Your bucket

You can add a bucket policy to your bucket with the following command. Replace `$BucketName` with the name of your bucket and `$pathToPolicy` with the path to the bucketpolicy.json file:

```
$ aws s3api put-bucket-policy --bucket $BucketName \
➥ --policy file://$pathToPolicy/bucketpolicy.json
```

Every object in the bucket can now be downloaded by anyone. You need to enable and configure the static web-hosting feature of S3 next. To do so, execute the following command, replacing `$BucketName` with the name of your bucket:

```
$ aws s3 website s3://$BucketName --index-document helloworld.html
```

Your bucket is now configured to deliver a static website. The HTML document helloworld.html is used as index page. You'll learn how to access your website next.

7.6.3 Accessing a website hosted on S3

You can now access your static website with a browser. To do so, you need to choose the right endpoint. The endpoints for S3 static web hosting depend on your bucket's region. For us-east-1 (US East N. Virginia), the website endpoint looks like this:

```
http://$BucketName.s3-website-us-east-1.amazonaws.com
```

Replace `$BucketName` with your bucket. So if your bucket is called `awesomebucket` and was created in the default region us-east-1, your bucket name would be:

```
http://awesomebucket.s3-website-us-east-1.amazonaws.com
```

Open this URL with your browser. You should be welcomed by a Hello World website.

Please note that for some regions, the website endpoint looks a little different. Check S3 endpoints and quotas at **http://mng.bz/epeq** for details.

Linking a custom domain to an S3 bucket

If you want to avoid hosting static content under a domain like awsinaction.s3-website-us-east-1.amazonaws.com, you can link a custom domain to an S3 bucket, such as `awsinaction.example.com`. All you have to do is to add a CNAME record for your domain, pointing to the bucket's S3 endpoint. The domain name system provided by AWS allowing you to create a CNAME record is called Route 53.

The CNAME record will work only if you comply with the following rules:

- *Your bucket name must match the CNAME record name.* For example, if you want to create a CNAME for awsinaction.example.com, your bucket name must be awsinaction.example.com as well.
- *CNAME records won't work for the primary domain name (such as* example.com*).* You need to use a subdomain for CNAMEs like awsinaction or www. If you want to link a primary domain name to an S3 bucket, you need to use the Route 53 DNS service from AWS.

Linking a custom domain to your S3 bucket works only for HTTP. If you want to use HTTPS (and you probably should), use AWS CloudFront together with S3. AWS CloudFront accepts HTTPS from the client and forwards the request to S3.

Cleaning up

Don't forget to clean up your bucket after you finish the example. To do so, execute the following command, replacing `$BucketName` with the name of your bucket:

```
$ aws s3 rb --force s3://$BucketName
```

7.7 Protecting data from unauthorized access

Not a week goes by without a frightening announcement that an organization has leaked confidential data from Amazon S3 accidentally. Why is that?

While reading through this chapter, you have learned about different scenarios for using S3. For example, you used S3 to back up data from your local machine. Also, you hosted a static website on S3. So, S3 is used to store sensitive data as well as public data. This can be a dangerous mix because a misconfiguration might cause a data leak.

To mitigate the risk, we recommend you enable Block Public Access for all your buckets as illustrated in figure 7.5 and shown next. By doing so, you will disable public access to all the buckets belonging to your AWS ac-

count. This will break S3 website hosting or any other form of accessing S3 objects publicly.

1. Open the AWS Management Console and navigate to S3.
2. Select Block Public Access Settings for This Account from the subnavigation menu.
3. Enable Block All Public Access, and click the Save Changes button.

Figure 7.5 Enable Block Public Access for all S3 buckets to avoid data leaks.

In case you really need buckets with both sensitive data and public data, you should enable Block Public Access not on the account level but for all buckets with sensitive data individually instead.

Check out our blog post “How to Avoid S3 Data Leaks?” at **https://cloudonaut.io/s3-security-best-practice/** if you are interested in further advice.

7.8 Optimizing performance

By default, S3 handles 3,500 writes and 5,500 reads per second. If your workload requires higher throughput, you need to consider the following when coming up with the naming scheme for the object keys.

Objects are stored without a hierarchy on S3. There is no such thing as a directory. All you do is specify an object key, as discussed at the beginning of the chapter. However, using a prefix allows you to structure the object keys.

By default, the slash character (`/`) is used as the prefix delimiter. So, `archive` is the prefix in the following example of object keys:

```
archive/image1.png
archive/image2.png
archive/image3.png
archive/image4.png
```

Be aware that the maximum throughput per partitioned prefix is 3,500 writes and 5,500 reads per second. Therefore, you cannot read more than 5,500 objects from the prefix `archive` per second.

To increase the maximum throughput, you need to distribute your objects among additional prefixes. For example, you could organize the objects from the previous example like this:

```
archive/2021/image1.png
archive/2021/image2.png
archive/2022/image3.png
archive/2022/image4.png
```

By doing so, you can double the maximum throughput when reading from archive/2021 and archive/2022 as illustrated in figure 7.6.

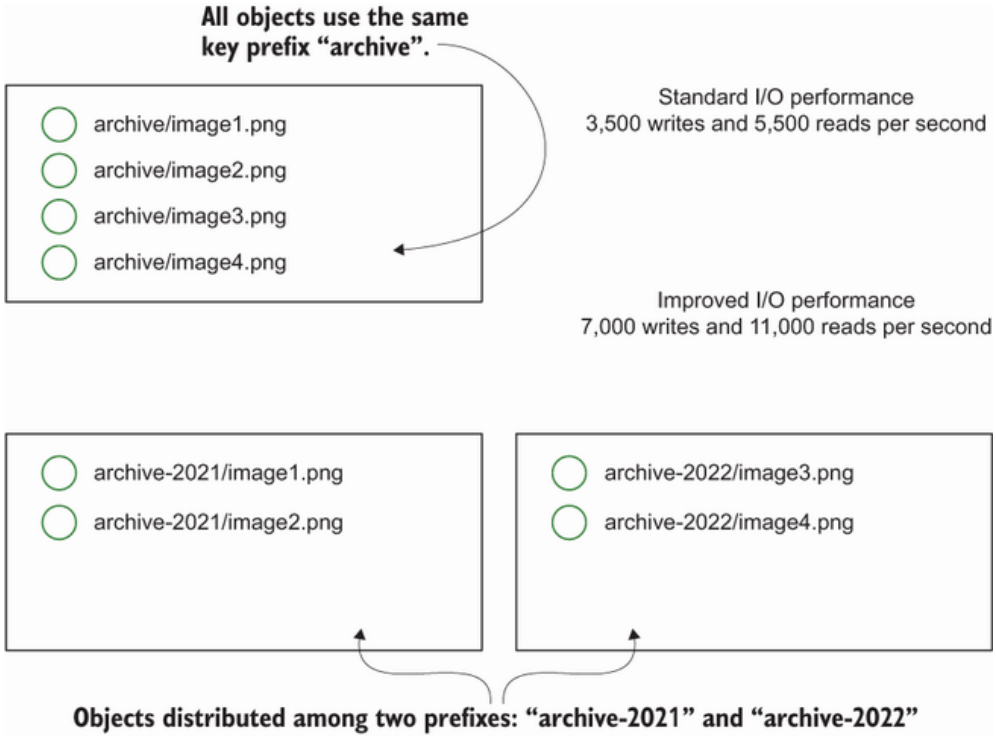

Figure 7.6 To improve I/O performance, distribute requests among multiple object key prefixes.

In summary, the structure of your object keys has an effect on the maximum read and write throughput.

Summary

- An object consists of a unique identifier, metadata to describe and manage the object, and the content itself. You can save images, documents, executables, or any other content as an object in an object store.
- Amazon S3 provides endless storage capacity with a pay-per-use model. You are charged for storage as well as read and write requests.
- Amazon S3 is an object store accessible only via HTTP(S). You can upload, manage, and download objects with the CLI, SDKs, or the Management Console. The storage classes Glacier Instant Retrieval, Glacier Flexible Retrieval, and Glacier Deep Archive are designed to archive data at low cost.
- Integrating S3 into your applications will help you implement the concept of a stateless server, because you don't have to store objects locally on the server.
- Enable Block Public Access for all buckets, or at least those buckets that contain sensitive information to avoid data leaks.
- When optimizing for high performance, make sure to use many different key prefixes instead of similar ones or the same prefix for all objects.