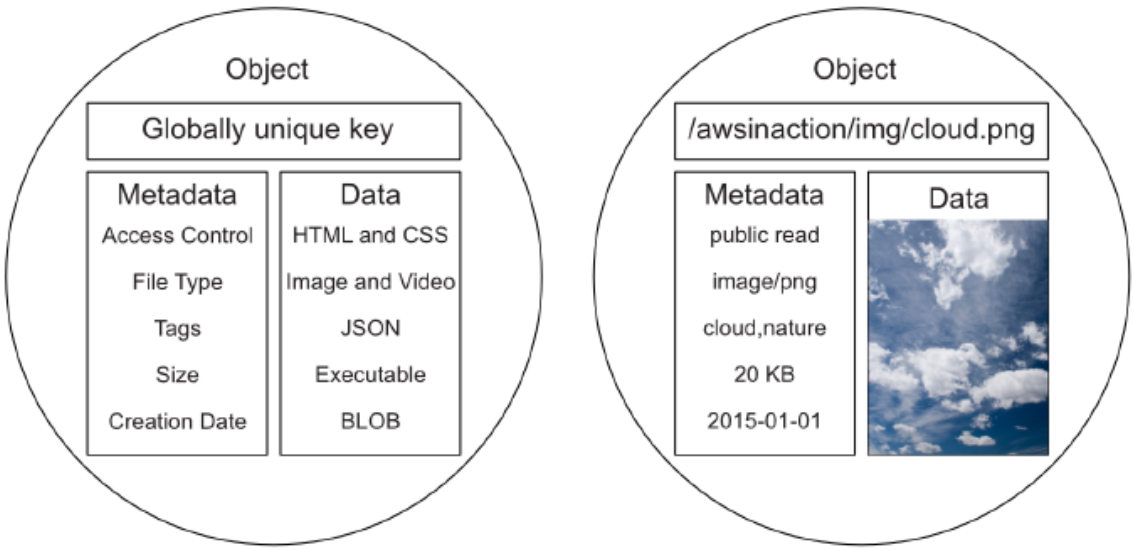
Introduction to S3

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. 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.

An object store, also known as object storage, is a data storage architecture that manages data as objects, as opposed to other storage architectures like file systems that manage data as a file hierarchy, or block storage that manages data as blocks within sectors and tracks. Object storage is particularly well-suited for storing large amounts of unstructured data, such as photos, videos, log files, backups, and other types of data that do not fit neatly into rows and columns.



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.

Amazon S3

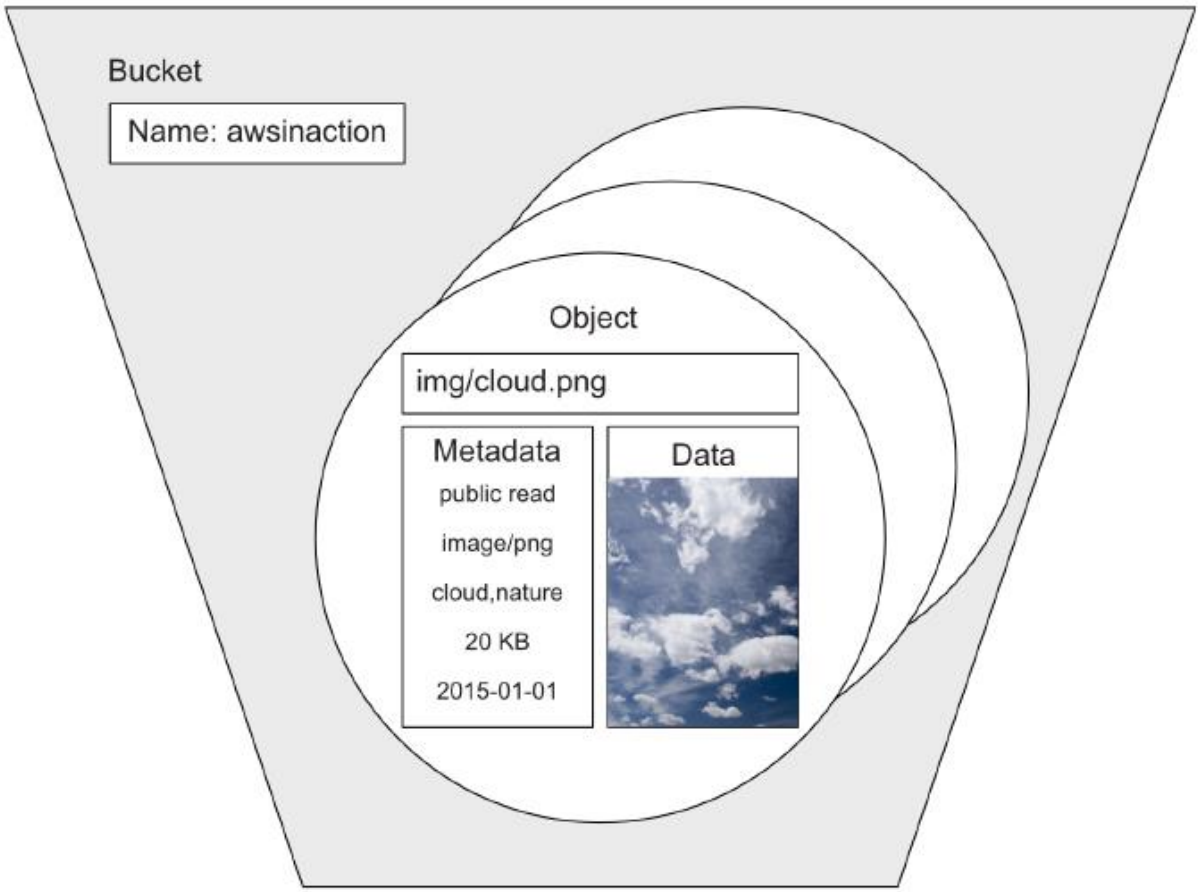
Amazon S3 is a distributed data store, and one of the oldest services provided by AWS. Amazon S3 is an acronym for Amazon Simple Storage Service. It's a typical web service that lets you store and retrieve data organized as objects via an API reachable over HTTPS.

Here are some typical use cases:

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

Amazon S3 offers 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.

S3 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.



Key Features of Amazon S3:

1. **Scalability:** Amazon S3 can automatically scale storage capacity up or down based on your needs, allowing you to handle virtually unlimited amounts of data without managing the underlying infrastructure.
2. **Durability and Availability:** S3 is designed for 99.999999999% (11 9's) durability by redundantly storing data across multiple devices in multiple facilities within an AWS Region. It offers 99.99% availability, ensuring data is accessible when needed.
3. **Data Management and Analytics:** S3 provides various features for data management, such as versioning, lifecycle policies, and replication. It also integrates with AWS analytics services like Amazon Athena, AWS Glue, and Amazon Redshift, enabling users to perform analytics directly on data stored in S3.
4. **Security:** S3 provides robust security features, including data encryption (both in transit and at rest), access control policies, and integration with AWS Identity and Access Management (IAM) for fine-grained access control.
5. **Storage Classes:** S3 offers multiple storage classes to optimize costs based on access frequency and performance needs:

I. **S3 Standard:** General-purpose storage for frequently accessed data.
II. **S3 Intelligent-Tiering:** Automatically moves data between two access tiers (frequent and infrequent) to optimize costs.
III. **S3 Standard-IA (Infrequent Access):** For data accessed less frequently but requiring rapid access.
IV. **S3 One Zone-IA:** Lower-cost option for infrequently accessed data that does not require multiple Availability Zone resilience.
V. **S3 Glacier:** Low-cost storage for data archiving with retrieval times ranging from minutes to hours.
VI. **S3 Glacier Deep Archive:** The lowest-cost storage for data that can be accessed within 12 hours.

6. **Performance:** S3 supports high-performance workloads with scalable throughput, low latency, and high availability. It can handle large numbers of simultaneous read/write requests.
7. **Data Consistency:** Amazon S3 provides strong read-after-write consistency for PUTs of new objects and eventual consistency for overwrite PUTs and DELETEs.

Components of Amazon S3:

1. **Buckets:** A bucket is a container for storing objects. Each object is stored in a bucket, and you can configure bucket settings such as permissions, versioning, and lifecycle policies.
2. **Objects:** An object consists of data and metadata. Each object is identified within a bucket by a unique key (name).
3. **Keys:** A key is the unique identifier for an object within a bucket. Each object in a bucket has exactly one key.
4. **Regions:** S3 buckets are created in specific AWS Regions. You can choose the region where your bucket resides to optimize latency, minimize costs, or address regulatory requirements.

Use Cases for Amazon S3:

1. **Backup and Restore:** Reliable and scalable storage for backing up data from various sources, including on-premises environments and other cloud services.
2. **Data Archiving:** Cost-effective long-term storage solutions with options like S3 Glacier and S3 Glacier Deep Archive.

3. **Content Storage and Distribution:** Storing and delivering static content such as images, videos, and documents for web applications.
4. **Big Data Analytics:** Storage for large datasets that can be processed using AWS analytics and machine learning services.
5. **Disaster Recovery:** Using cross-region replication and other features to ensure data is available and recoverable in case of disaster.

Interacting with S3 Via AWS CLI

Interacting with Amazon S3 via the AWS Command Line Interface (CLI) is a powerful way to manage your S3 resources programmatically. The AWS CLI provides a unified tool to manage your AWS services, including S3, and allows you to script operations, automate workflows, and integrate with other tools.

Backing up your data on S3 with AWS CLI

Critical data needs to be backed up to avoid loss. Backing up data at an off-site 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.

You can use S3 in many other scenarios as well:

- 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, configuration files, and so on).
- 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

If your bucket name conflicts with an existing bucket, you'll get an error like this one:

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

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

Everything is ready 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/ **

[CA] /Users/andreas/Downloads/restore

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

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.

Basic S3 Operations Using AWS CLI

1. Listing Buckets

   To list all buckets in your account: **aws s3 ls**

2. Creating a Bucket

   To create a new bucket: **aws s3 mb s3://my-bucket-name**

Replace my-bucket-name with your desired bucket name. Bucket names must be globally unique.

3. Listing Objects in a Bucket

   To list objects within a specific bucket: **aws s3 ls s3://my-bucket-name**

4. Uploading an Object

   To upload a file to a bucket: **aws s3 cp path/to/local/file s3://my-bucket-name**

5. Downloading an Object

   To download a file from a bucket: **aws s3 cp s3://my-bucket-name/file.txt path/to/local/directory**

6. Deleting an Object

   To delete a specific object from a bucket: **aws s3 rm s3://my-bucket-name/file.txt**

7. Deleting a Bucket

   To delete an empty bucket: **aws s3 rb s3://my-bucket-name**

   To delete a bucket and all its contents: **aws s3 rb s3://my-bucket-name --force**

Advanced S3 Operations

1. Syncing Directories

To synchronize a local directory with an S3 bucket: **aws s3 sync path/to/local/directory s3://my-bucket-name**

To synchronize an S3 bucket with a local directory: **aws s3 sync s3://my-bucket-name path/to/local/directory**

Types of Storage Classes on S3

Amazon S3 (Simple Storage Service) offers a range of storage classes designed to accommodate different use cases and cost requirements. Each storage class has specific features tailored to particular types of data and access patterns. Understanding these storage classes helps optimize both cost and performance based on how frequently you access your data and the level of durability and availability you require.

Types of Storage Classes in Amazon S3

1. S3 Standard
2. S3 Intelligent-Tiering
3. S3 Standard-IA (Infrequent Access)
4. S3 One Zone-IA
5. S3 Glacier
6. S3 Glacier Deep Archive

1. S3 Standard

- Designed for frequently accessed data.
- High durability and availability.

Use Cases:

- Data that requires low latency and high throughput.
- Frequently accessed applications like dynamic websites, content distribution, and big data analytics.

2. S3 Intelligent-Tiering

- Automatically moves data between two access tiers (frequent and infrequent) based on changing access patterns.

Use Cases:

- Data with unpredictable or changing access patterns.
- Ideal for datasets where access frequency is unknown or varies over time.

3. S3 Standard-IA (Infrequent Access)

- Designed for data that is accessed less frequently, but requires rapid access when needed.

Use Cases:

- Long-term storage, backups, and disaster recovery.
- Data that needs to be retained but is not frequently accessed.

4. S3 One Zone-IA

- Lower-cost option for infrequently accessed data that does not require multiple Availability Zone data resilience.

Use Cases:

- Data that can be recreated or is easily replaceable.
- Secondary backups, or data that can be easily recreated if lost.

5. S3 Glacier

- Low-cost storage for data archiving and long-term backup.
- Designed for data that is rarely accessed, with retrieval times ranging from minutes to hours.

Use Cases:

- Archival storage, compliance and regulatory data retention.
- Long-term data archiving that does not require instant access.

6. S3 Glacier Deep Archive

- Lowest-cost storage class designed for long-term data archiving.
- Suitable for data that is rarely accessed and requires longer retrieval times.

Use Cases:

- Data that needs to be retained for many years for compliance and regulatory requirements.
- Digital preservation and long-term backups where access time is not critical.

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 USD per month. Wouldn't it be nice, to reduce the costs for storing data by 95%? Besides, the 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 USD	0.004 USD	0.0036 USD	0.00099 USD
Costs for 1,000 write requests	0.005 USD	0.02 USD	0.03 USD	0.05 USD
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%

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 USD to restore 1 TB of data stored when 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 5 times during this period.

Start by creating an S3 bucket, that you will use to archive documents. 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 down, the key of the object.

**$ aws s3 cp --storage-class GLACIER $path **

[CA] s3://awsinaction-archive-$yourname/

For instance, I run the following command.

**$ aws s3 cp --storage-class GLACIER **

**[CA] /Users/andreas/Desktop/taxstatement-2022-07-01.pdf **

[CA] 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 do download my document taxstatement-2022-07-01.pdf.

**$ aws s3 cp s3://awsinaction-archive-awittig/taxstatement-2022-07-01.pdf **

[CA] ~/Downloads

warning: Skipping file s3://awsinaction-archive-awittig/

[CA]taxstatement-2022-07-01.pdf. Object is of storage class GLACIER.

[CA]Unable to perform download operations on GLACIER objects. You must

[CA]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 3 to 5 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 **

**[CA] --key $objectkey **

[CA] --restore-request Days=1„GlacierJobParameters={"Tier"="Expedited"}

That's resulting in the following command in my scenario.

**$ aws s3api restore-object --bucket awsinaction-archive-awittig **

[CA] --key taxstatement-2022-07-01.pdf

[CA] --restore-request Days=1„GlacierJobParameters={"Tier"="Expedited"}

As you are using expedited retrieval, you need to wait 1 to 5 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 **

[CA] --key $objectkey

After restoring the object, you are now able to download the document. 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. As we store another backup of our data on an external hard drive, chances that we need to restore data from S3 are very low.

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.

Setting up an S3 bucket

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

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

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

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

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

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 listing 7.1 shows how to do so.

Listing 7.1 Uploading an image with the AWS SDK for S3

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

const s3 = new AWS.S3({                            ❷
  'region': 'us-east-1'
});

const bucket = process.argv[2];

async function uploadImage(image, response) {
  try {
    await s3.putObject({                           ❸
      Body: image,                                 ❹
      Bucket: bucket,                              ❺
      Key: uuid.v4(),                              ❻
      ACL: 'public-read',                          ❼
      ContentLength: image.byteCount,              ❽
      ContentType: image.headers['content-type']   ❾
    }).promise();
    response.redirect('/');
  } catch (err) {                                  ❿
    console.error(err);
    response.status(500);
    response.send('Internal server error.');       ⓫
  }
}
```

❶ Load the AWS SDK.

❷ Instantiate s3 client with additional config.

❸ Uploads the image to S3

❹ Image content

❺ Name of the bucket

❻ Generates a unique key for the object

❼ Allows everybody to read the image from bucket

❽ Size of image in bytes

❾ Content type of the object (image/png)

❿ Catching errors

⓫ Return error with HTTP status code 500

Listing all the images in the s3 bucket

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. Listing 7.2 shows the implementation of the corresponding function in the server.js JavaScript file, acting as a web server.

❶ Reads the bucket name from the process arguments

❷ Lists the objects stored in the bucket

❸ The bucket name is the only requried parameter

❹ Renders a HTML page based on the list of objects

❺ Stream the response

❻ Handle potential errors

Listing 7.2 Retrieving all the image locations from the S3 bucket

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

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

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. Listing 7.3 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}}                    ❶
  <p><img src=                  ❷
[CA] "https://s3.amazonaws.com/{{Bucket}}/{{Key}}"
[CA] width="400px" ></p>
{{/Objects}}
[...]
```

❶ Iterates over all objects

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

Using S3 for static web hosting

Using S3 for static web hosting:

Using Amazon S3 for static web hosting is a cost-effective and scalable solution for hosting static websites, which include HTML, CSS, JavaScript, and image files but do not require server-side scripting. Here's a detailed guide on how to set up and configure S3 for static web hosting:

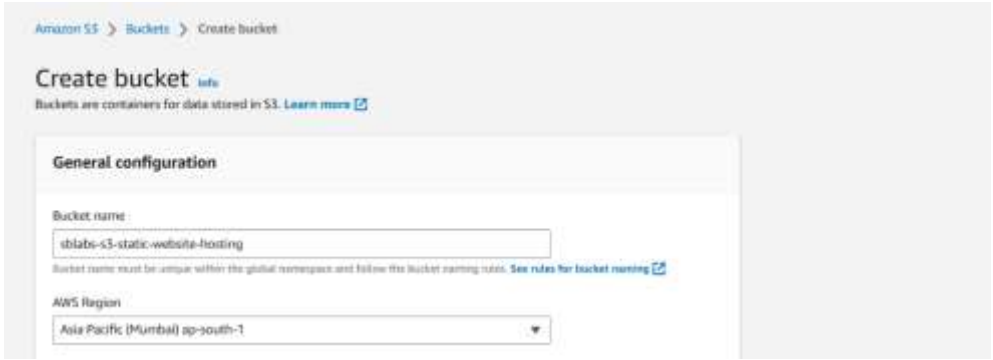
Steps to host static website in S3.

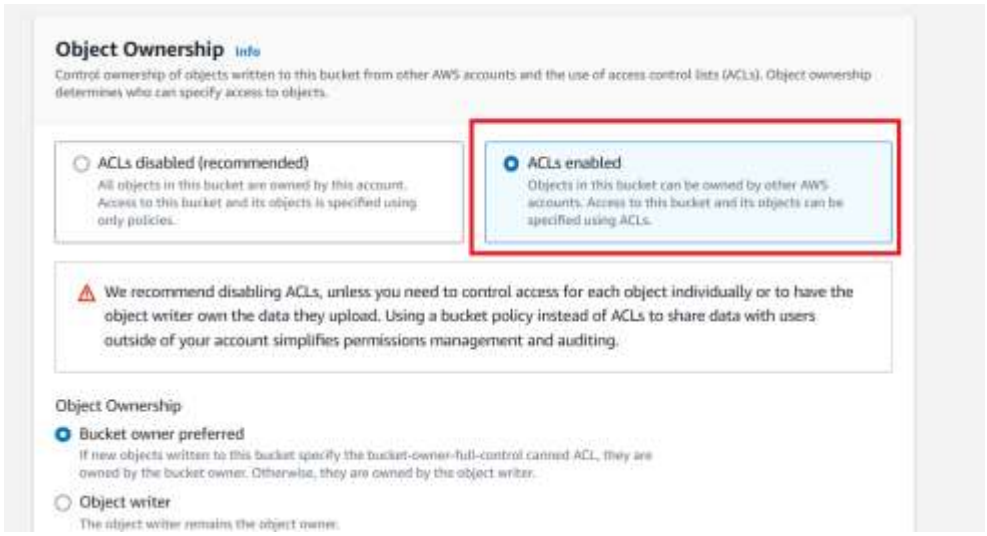
Step 1: Sign In to AWS Console

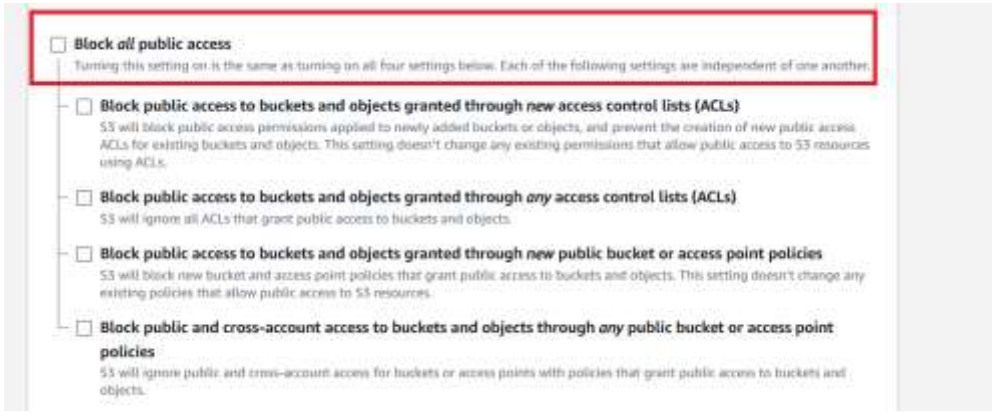
1. Open your web browser and navigate to the AWS Management Console.
2. Enter your AWS account credentials (username and password) to log in.

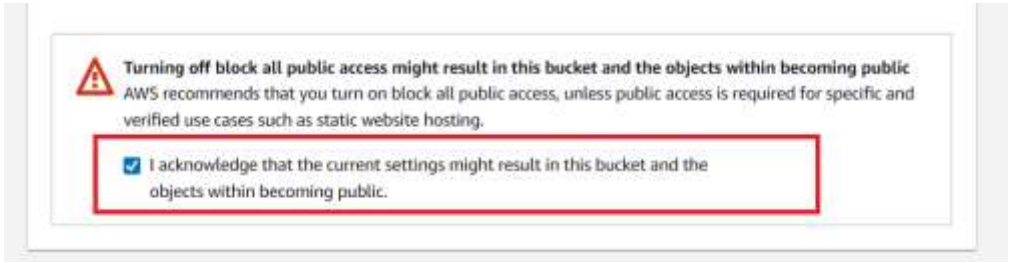
Step 2: Create an S3 Bucket

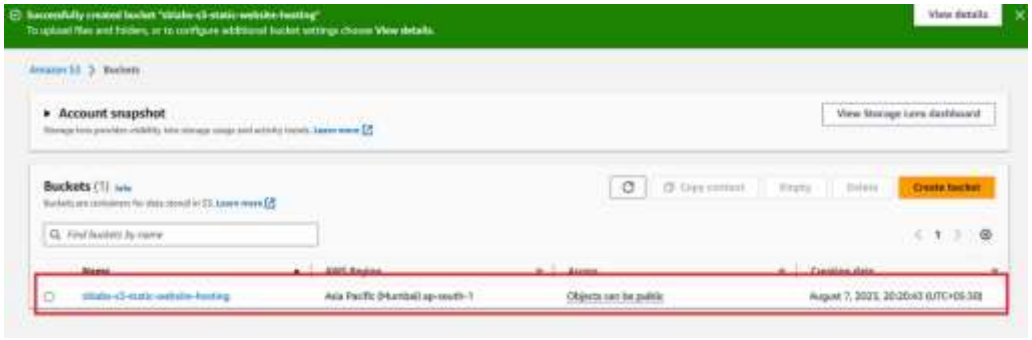
1. In the AWS Management Console, search for and select the "S3" service.
2. Click the "Create bucket" button.
3. Provide a globally unique name for your bucket (e.g., "my-static-website").
4. Choose a region that is geographically closest to your target audience for better performance.
5. Leave the default settings for the rest of the options and click "Create bucket."


Amazon S3 > Buckets > Create bucket
Create bucket Info
Buckets are containers for data stored in S3. Learn more
General configuration
Bucket name
sblabs-s3-static-website-hosting
Bucket name must be unique within the global namespace and follow the bucket naming rules. See rules for bucket naming
AWS Region
Asia Pacific (Mumbai) ap-south-1


Object Ownership Info
Control ownership of objects written to this bucket from other AWS accounts and the use of access control lists (ACLs). Object ownership determines who can specify access to objects.
ACLs disabled (recommended)
All objects in this bucket are owned by this account. Access to this bucket and its objects is specified using only policies.
ACLs enabled
Objects in this bucket can be owned by other AWS accounts. Access to this bucket and its objects can be specified using ACLs.
We recommend disabling ACLs, unless you need to control access for each object individually or to have the object writer own the data they upload. Using a bucket policy instead of ACLs to share data with users outside of your account simplifies permissions management and auditing.
Object Ownership
Bucket owner preferred
If new objects written to this bucket specify the bucket-owner-full-control canned ACL, they are owned by the bucket owner. Otherwise, they are owned by the object writer.
Object writer
The object writer remains the object owner.


Block all public access
Turning this setting on is the same as turning on all four settings below. Each of the following settings are independent of one another.
Block public access to buckets and objects granted through new access control lists (ACLs)
S3 will block public access permissions applied to newly added buckets or objects, and prevent the creation of new public access ACLs for existing buckets and objects. This setting doesn't change any existing permissions that allow public access to S3 resources using ACLs.
Block public access to buckets and objects granted through any access control lists (ACLs)
S3 will ignore all ACLs that grant public access to buckets and objects.
Block public access to buckets and objects granted through new public bucket or access point policies
S3 will block new bucket and access point policies that grant public access to buckets and objects. This setting doesn't change any existing policies that allow public access to S3 resources.
Block public and cross-account access to buckets and objects through any public bucket or access point policies
S3 will ignore public and cross-account access for buckets or access points with policies that grant public access to buckets and objects.

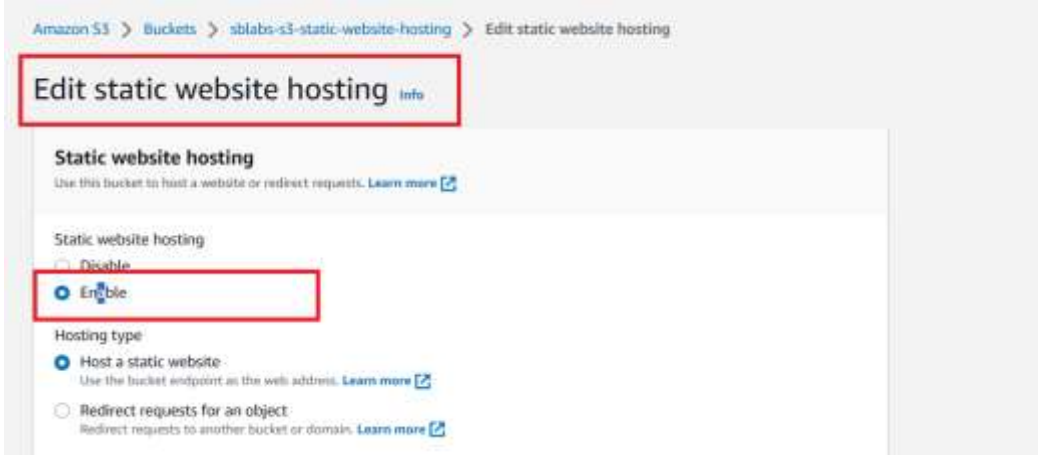

Turning off block all public access might result in this bucket and the objects within becoming public
AWS recommends that you turn on block all public access, unless public access is required for specific and verified use cases such as static website hosting.
I acknowledge that the current settings might result in this bucket and the objects within becoming public.



Bucket is created.

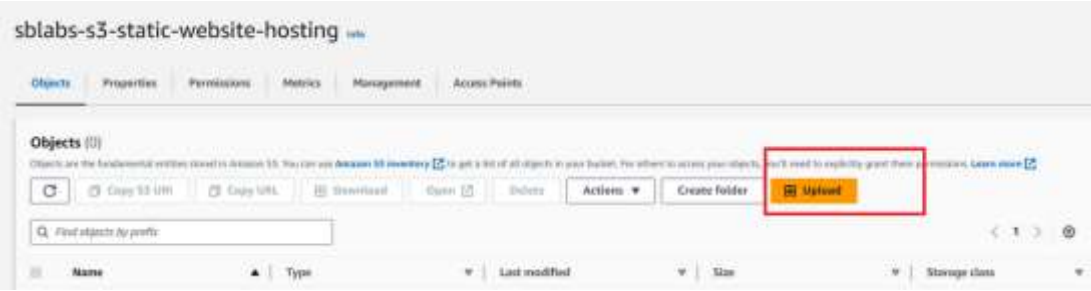
Step 3: Configure Bucket Properties

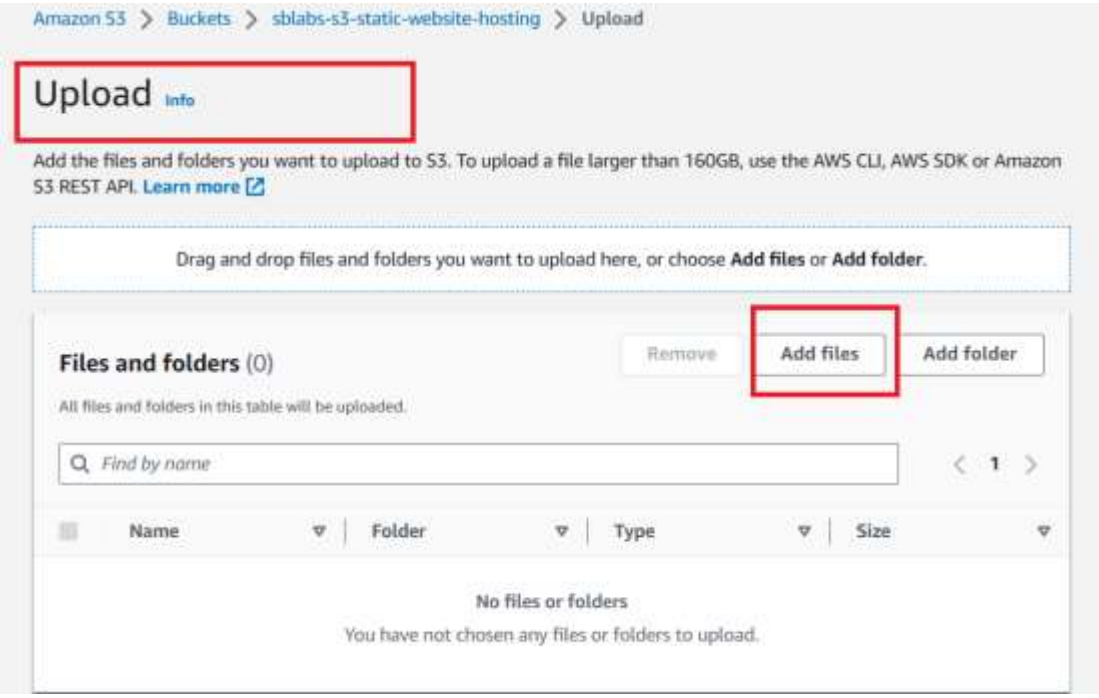
1. Once your bucket is created, select it from the list of buckets.
2. Go to the “Properties” tab and enable “Static website hosting.”
3. Enter the “Index document” (e.g., “index.html”) and “Error document” (e.g., “error.html”) if you have one.

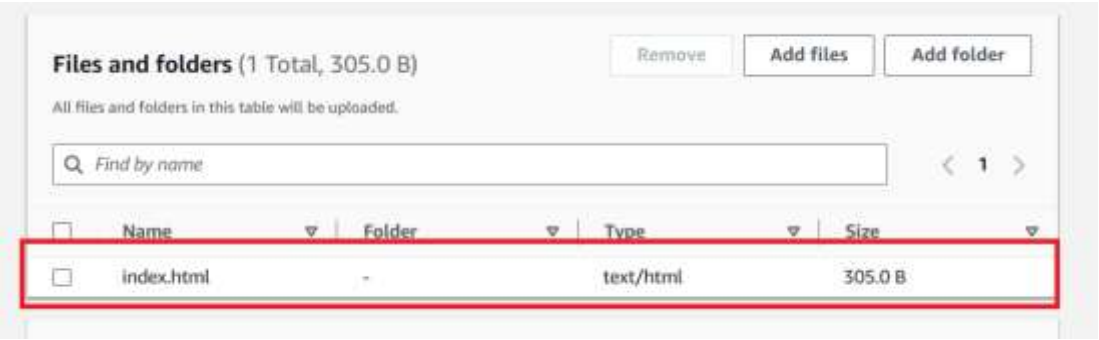


Step 4: Upload Your Website Files

1. Go to the “Overview” tab of your bucket.
2. Click the “Upload” button to add your static website files (HTML, CSS, JavaScript, images, etc.).
3. You can either drag and drop files or use the upload interface.



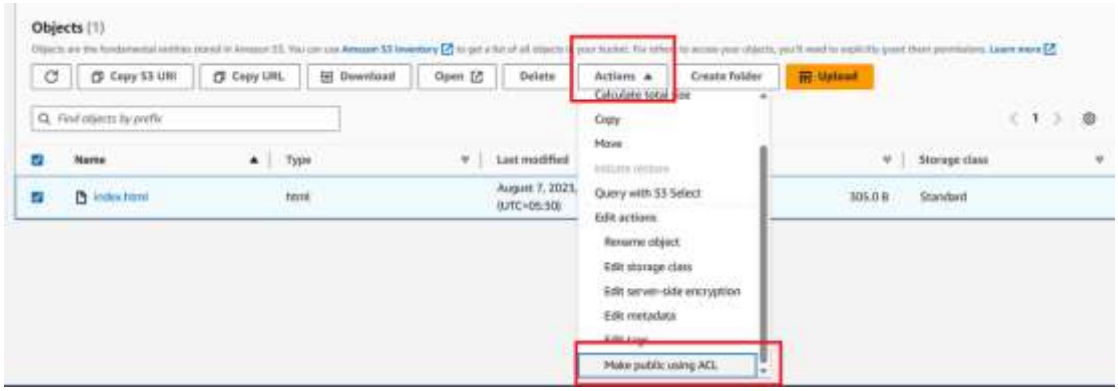




File index.html Uploaded.

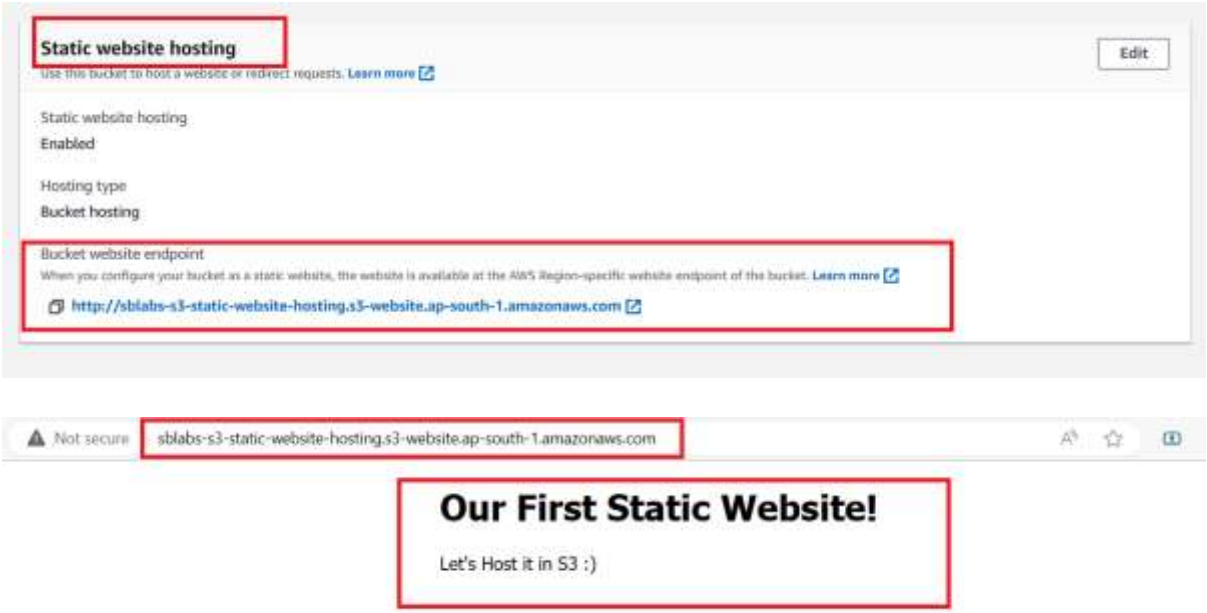
Step 5: Make Objects Public

1. Select the uploaded files in your bucket.
2. Go to the “Actions” dropdown and choose “Make public.”
3. Confirm that you want to make the files public. This is necessary to allow public access to your website content.



Step 6: Access Your Website

- After making file public, your static website should be accessible at the endpoint provided in the “Static website hosting” section of your bucket’s properties. It will look like this: http://your-bucket-name.s3-website-region.amazonaws.com.



Successfully hosted static website

Protecting data from unauthorized access:

For example, you used S3 to backup 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 as a misconfiguration might cause a data leak.

To mitigate the risk, we recommend to enable Block Public Access for all your buckets as illustrated in figure 7.5. By doing so, you will disable public access to all the buckets belonging to your AWS account. So 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 sub navigation.
3. Enable Block all public access and click the Save changes button.

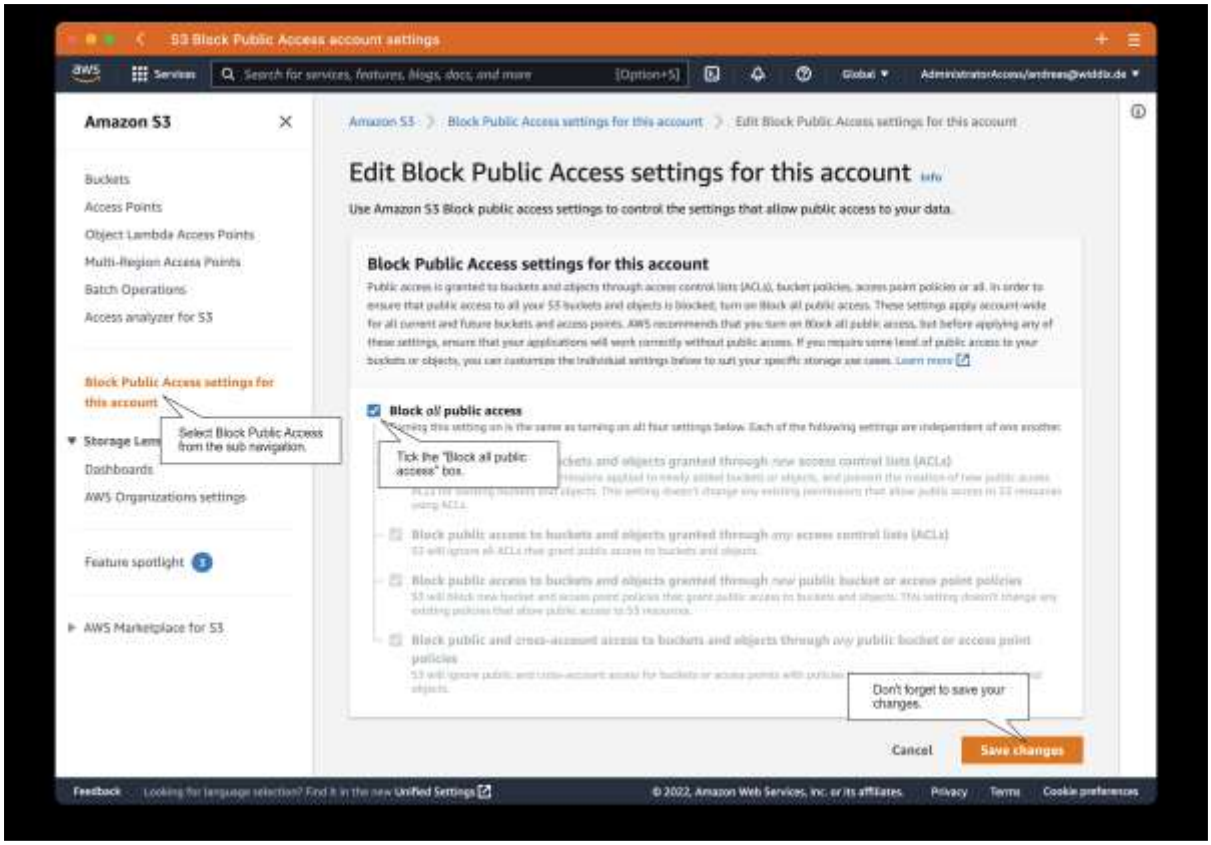


Figure 7.5 Enable block public access for all S3 buckets to avoid data leaks

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

Optimizing performance:

By default, S3 handles 3,500 writes and 5,500 reads per second. In case 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 to 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 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 example above 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.

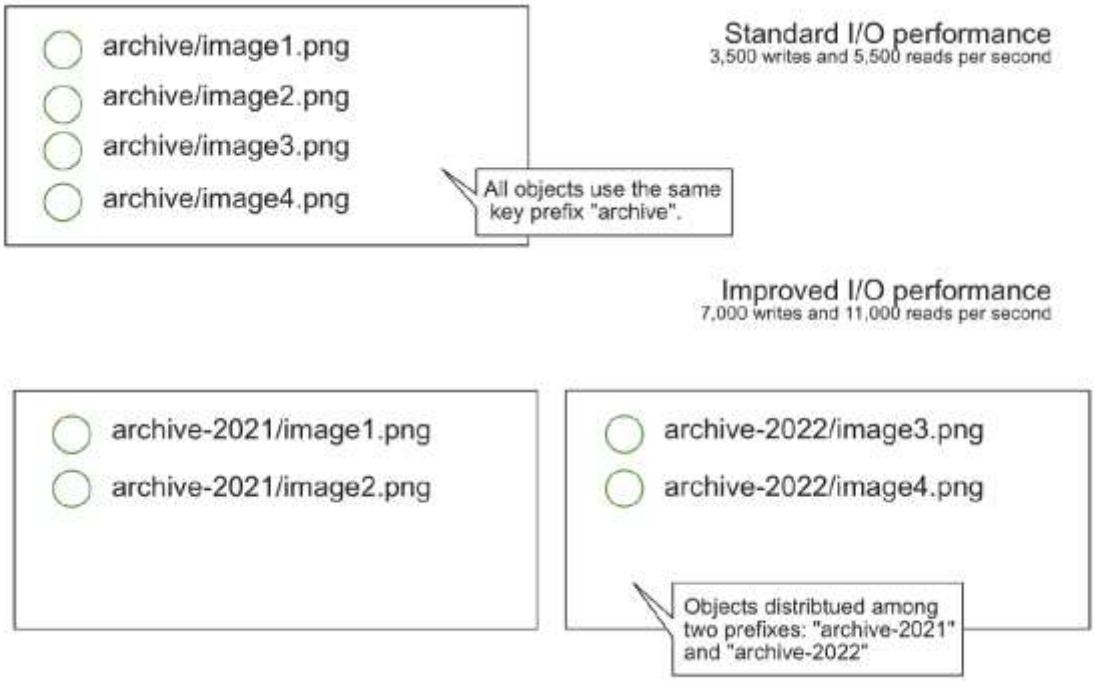


Figure 7.6 To improve I/O performance distribute requests among multiple object key prefixes

Migrate a website from local server to Cloud

Migrate a website from local server to Cloud

==========

1) Launch an EC2 Instance

2) Connect to EC2 Instance

3) clone the git repos

backend

==========

tinyurl.com/cs1bekmit

git clone https://github.com/procareer3fwd/realgrandebackend.git

frontend

==========

tinyurl.com/cs1fekmit

git clone https://github.com/procareer3fwd/realgrandefrontend.git

4) Update the Ubuntu

sudo apt update

5) Install the Docker

sudo apt -y install docker.io

6) Check the docker images

sudo docker images

7) change the directory to backend

cd realgrandebackend/

8) create an .env file

nano .env

9) Copy Paste the lines in .env file

MONGODBURL="mongodb+srv://fsd04.2hxrdca.mongodb.net/realgrande?retryWrites=true&w=majority"

DBUSERNAME=procareer3

DBPASSWORD=ISobjBDohsFqEAqg

FRONTENDURI="http://3.82.156.186"

10) Build the docker image for backend

sudo docker build -t backend_server .

11) Check for any images running in the container

sudo docker ps

12) Now run the backend_server docker image in the container

sudo docker run -d -p 2001:5000 backend_server

13) Now try to access the backend_server on the browser

<EC2_PUBLIC_IP_ADDRESS>:2001/api

14) Now change the directory to frontend

ubuntu@ip-172-31-1-17:~/realgrandebackend$ cd

ubuntu@ip-172-31-1-17:~$ cd realgrandefrontend/

ubuntu@ip-172-31-1-17:~/realgrandefrontend$

15) Now create a .env file

nano .env

16) Copy paste the line in .env file

REACT_APP_BACKEND_URL="http://3.82.156.186:2001/api"

17) Build the docker image for frontend

sudo docker build -t frontend .

18) Check for any images running in the container

sudo docker ps

19) Now run the backend_server docker image in the container

sudo docker run -d -p 80:3000 frontend

20) Now try to access the frontend on the browser

<EC2_PUBLIC_IP_ADDRESS>