This page describes how to migrate from Amazon Simple Storage Service (Amazon S3) to Cloud Storage for users sending requests using an API.

If you are new to Cloud Storage and will not be using the API directly, consider using the Google Cloud Console to set and manage transfers (/storage-transfer/docs/create-manage-transfer-console). The Google Cloud Console provides a graphical interface to Cloud Storage that enables you to accomplish many of your storage tasks using just a browser, including migration of your data from Amazon S3 to Cloud Storage.

If you are an Amazon S3 user, you can easily migrate your applications that use Amazon S3 to use Cloud Storage. You have two migration options:

- **Simple Migration** (#migration-simple): This is this easiest way to get started with Cloud Storage if you are coming from Amazon S3 because it requires just a few simple changes to the tools and libraries you currently use with Amazon S3.
- **Full Migration** (#migration-full): A full migration from Amazon S3 to Cloud Storage requires a few extra steps compared to a simple migration, but the benefit is that you can use all the features of Cloud Storage, including multiple projects and OAuth 2.0 for authentication.

In a simple migration from Amazon S3 to Cloud Storage, you use your existing tools and libraries for generating authenticated REST requests to Amazon S3 to also send authenticated requests to Cloud Storage. The only steps you need to take to make requests to Cloud Storage are:

- Set a default Google project (#defaultproj).
- Get an HMAC key (/storage/docs/authentication/hmackeys).
- In your existing tools or libraries, make the following changes:
    - Change the request endpoint to use the Cloud Storage XML API request endpoint (/storage/docs/request-endpoints#typical).

- Replace the Amazon Web Services (AWS) access and secret key with the corresponding Cloud Storage access ID and secret (collectively called your Cloud Storage HMAC key).
- Make sure your `x-amz-` headers use supported Cloud Storage values. For example, `x-amz-storage-class` should use one of the available Cloud Storage storage classes (/storage/docs/storage-classes).

  When you use the Cloud Storage XML API in a simple migration scenario, specifying the `AWS` signature identifier in the `Authorization` header lets Cloud Storage know to expect `x-amz-*` headers and Amazon S3 ACL XML syntax in your request. Cloud Storage processes `x-amz-*` headers that have an `x-goog-*` equivalent, such as those listed in the headers table (#custommeta).

With these changes you can start using your existing tools and libraries to send keyed-hash message authentication code (HMAC) requests to Cloud Storage.

For example, the following samples demonstrate how to list Cloud Storage buckets using the Amazon S3 SDK:

Note that while most actions are available using the Amazon S3 V2 SDK, listing objects can only be performed using the Amazon S3 V1 list objects method. The following samples demonstrate such an object listing:

To use the Cloud Storage in a simple migration scenario, you must choose a default project. When you choose a default project, you are telling Cloud Storage the project to use for operations like `GET` service or `PUT` bucket.

To set a default project:

1. Open the **Cloud Storage Settings** page (https://console.cloud.google.com/storage/settings) in the Google Cloud Console (https://console.cloud.google.com/).
2. Select **Interoperability**.
3. Click **Make *PROJECT-ID* my default project**.

   If the project is already the default project, you see ***PROJECT-ID* is your default project for interoperable access**.

This project is now your default project. You can change your default project at any time by choosing a different project and following these steps.

Choosing a default project for a simple migration (interoperable access) does not impact using the XML API for a ful ion, where you specify the **`x-goog-project-id`** with each request.

To use the Cloud Storage XML API in a simple migration scenario, use Cloud Storage hash-based message authentication code (HMAC) keys (/storage/docs/authentication/hmackeys) for the credentials. Typically, you should create an HMAC key (/storage/docs/authentication/managing-hmackeys) that is associated with a service account, however you can alternatively use one associated with a user account.

ature is in a pre-release state and might change or have limited support. For more information, see the product launc s (/products/#product-launch-stages).

For operations in a simple migration scenario that require authentication, you include an `Authorization` request header just like you do for requests to Amazon S3. The `Authorization` header syntax for an Amazon S3 request is:

In a simple migration scenario, you only change the header to use your Google HMAC access ID and make sure the `Signature` you attach is calculated with your Google HMAC secret key:

The parts of the `Authorization` header are:

- ***ALGORITHM***: The signature algorithm and version that you are using. Using `AWS4-HMAC-SHA256` indicates that you are using an HMAC V4 signature and you intend to send `x-amz-*` headers. You can also use `GOOG4-HMAC-SHA256`, which indicates that you are using an HMAC V4 signature and you intend to send `x-goog-*` headers.
- ***GOOG-ACCESS-ID***: The access ID identifies the entity that is making and signing the request. In a simple migration, replace the Amazon Web Service (AWS) access key ID you use to access Amazon S3 with your Google HMAC access ID. Your Google HMAC access ID starts with `GOOG`.
- ***CREDENTIAL_SCOPE***: The credential scope is a string that has the following structure:

  - ***DATE***: Date formatted as YYYYMMDD.
  - ***LOCATION***: The region where the resource resides or will be created.
  - ***SERVICE***: The service name.
  - ***REQUEST_TYPE***: The request type.

  An example of an Amazon S3-style credential scope looks like:

  In a simple migration, you don't need to change credential scope if you are using `AWS4-HMAC-SHA256` for your ***ALGORITHM*** value. If you want to use `GOOG4-HMAC-SHA256`, replace `aws4_request` with `goog4_request`.
- ***SIGNED_HEADERS***: A semicolon-separated list of names of headers that must be included to sign this request. All headers should be lowercase and sorted by character code.

  An example of an Amazon S3-style signed header string looks like:

  In a simple migration, you don't need to make any changes to the signed header string.
- ***SIGNATURE***: The cryptographic hash of the string to sign that allowed the request to be authenticated. The signature is created by using HMAC-SHA256 as the hash function and a derived key from your secret and credential scope as the cryptographic key. The resulting digest

is then Base64 encoded. When Cloud Storage receives your signed request, it uses the access ID to look up your secret and verify that you created the signature. For more information on how to obtain an access and secret key, see Managing HMAC keys for service accounts (/storage/docs/authentication/managing-hmackeys).

In a simple migration, replace the AWS secret access key with your Google HMAC key secret to derive the cryptographic key.

This section describes the process of authenticating an XML API request in a simple migration scenario. While this section can be used to develop your own code to sign requests, it is mainly intended to be a review if you already have tools or libraries that sign your requests to Amazon S3. In this case, you continue to use those tools to access Cloud Storage using the XML API, with the changes shown here.

Signatures provides both identity and strong authentication without revealing your secret. Providing both identity and authentication in every request helps ensure that every request is processed under a specific user account and with the authority of that user account. This is possible because only you and the Cloud Storage system know your secret. When you make a request, the Cloud Storage system uses your secret to calculate the same signature for the request that you calculated when you made the request. If the signatures match, then the Cloud Storage system knows that only you could have made the request.

The following pseudocode shows how to create the signature for the Authorization header:

To create the signature, you use a cryptographic hash function known as HMAC-SHA256. HMAC-SHA256 is a hash-based message authentication code (MAC) and is described in RFC 4868 (https://tools.ietf.org/html/rfc4868). It requires two input parameters, both UTF-8 encoded: a signing key and a string-to-sign.

The signing key is derived from your Cloud Storage secret and is specific to the date, location, service, and request type associated with your request. Additionally, these values must match the values specified in the credential scope. The following pseudocode shows how to create the signing key:

where ***GOOG-SECRET-KEY*** is the secret for the HMAC key that is being used to make and sign the request.

The string-to-sign includes meta information about your request and about the canonical request (/storage/docs/authentication/canonical-requests) that you want to sign. The following pseudocode shows how to construct the string-to-sign, including the use of newlines between each element:

The string-to-sign has the following components:

- ***SigningAlgorithm***: This should be `AWS4-HMAC-SHA256` for a simple migration.
- ***RequestDateTime***: The current date and time, in ISO 8601 format: `YYYYMMDD'T'HHMMSS'Z'`.
- ***CredentialScope***: The credential scope of the request for signing the string-to-sign, as covered in the Authorization header (#authheader) section.
- ***HashedCanonicalRequest***: The hex-encoded SHA-256 hash of the canonical request (/storage/docs/authentication/canonical-requests). Use a SHA-256 hashing function to create a hash value of the canonical request. Your programming language should have a library for creating SHA-256 hashes. An example hash value looks like:

An example of a string-to-sign looks like:

The following examples upload an object named **/europe/france/paris.jpg** to a bucket named **my-travel-maps**, apply the predefined ACL `public-read`, and define a custom metadata header for reviewers. Here is the request to a bucket in Amazon S3:

Here is the request for a bucket in Cloud Storage:

Here is the corresponding canonical request that was created for this request:

Here is the corresponding string-to-sign that was created for this request:

This request did not provide a Content-MD5 header, so an empty string is shown in the second line of the message.

To support simple migrations, Cloud Storage accepts ACLs produced by Amazon S3. In a simple migration scenario, you use `AWS` as your signature identifier, which tells Cloud Storage to expect ACL syntax using Amazon S3 ACL XML syntax. You should ensure that the Amazon S3 ACLs you use map to the Cloud Storage ACL model. For example, if your tools and libraries use Amazon S3's ACL syntax to grant bucket `WRITE` permission, then they must also grant bucket `READ` permission because Cloud Storage permissions are concentric (/storage/docs/access-control/lists#concentric). You do not need to specify both `WRITE` and `READ` permission when you grant `WRITE` permission using the Cloud Storage syntax.

Cloud Storage supports Amazon S3 ACL syntax in the following scenarios:

- In a request to Cloud Storage to retrieve ACLs (for example, a `GET` Object or `GET` Bucket request), Cloud Storage returns Amazon S3 ACL syntax.
- In a request to Cloud Storage to apply ACLs (for example, a `PUT` Object or `PUT` Bucket request), Cloud Storage expects to receive Amazon S3 ACL syntax.

The `Authorization` header in a simple migration scenario uses `AWS` for the signature identifier, but with your Google access ID.

The following example shows a `GET` request to Cloud Storage to return the ACLs for an object.

The response to the request includes the ACL using Amazon S3 ACL syntax.

The following example shows a `PUT` request to Cloud Storage to set the ACLs for an object. The example shows a request body with Amazon S3 ACL syntax.

Finally, in a simple migration scenario, you can also use the `GOOG1` signature identifier in the `Authorization` header. In this case, you must use the Cloud Storage ACL syntax and ensure that all of your headers are Google headers, `x-goog-*`. While this is possible, we recommend that you move to a full migration as described below in order to realize all the benefits of Cloud Storage.

A full migration from Amazon S3 to Cloud Storage enables you to take advantage of all the features of Cloud Storage including:

- **Support for multiple projects**: Multiple projects allow you to have, in effect, many instances of the Cloud Storage service. This allows you to separate different functionality or services of your application or business as needed. For more information, see Projects (/storage/docs/projects).
- **OAuth 2.0 authentication**: OAuth 2.0 relies on SSL for security instead of requiring your application to do cryptographic signing directly and is easier to implement. With OAuth, your application can request access to data associated with a user's Google Account, and access can be scoped to several levels, including read-only, read-write, and full-control. For more information, see OAuth 2.0 Authentication (/storage/docs/authentication#oauth).

To migrate fully from Amazon S3 to Cloud Storage, make the following changes:

- Change any existing `x-amz-*` headers to corresponding `x-goog-*` headers.
- Change AWS Access Control List (ACL) XML to the corresponding Cloud Storage ACL XML (see Creating and managing access control lists (/storage/docs/access-control/create-manage-lists)).
- Set the x-goog-project-id (/storage/docs/xml-api/reference-headers#xgoogprojectid) header in your requests. Note that in a simple migration scenario, you chose a default project for all requests. This is not needed in a full migration.
- Get set up to use OAuth 2.0 authentication as described in OAuth 2.0 Authentication (/storage/docs/authentication#oauth). The first step is to register your application (where you will

be issuing requests from) with Google. Using OAuth 2.0 means that your `Authorization` header looks like this:

This section shows a few examples of access control to help you migrate from Amazon S3 to Cloud Storage. For an overview of access control in Cloud Storage, see Access Control (/storage/docs/access-control).

In Cloud Storage, there are several ways to apply ACLs to buckets and objects (see Creating and managing access control lists (/storage/docs/access-control/create-manage-lists)). Two of the ways you specify ACLs are analogous to what you do in Amazon S3:

- The `acl` query string parameter to apply ACLs for specific scopes.
- The `x-goog-acl` request header lets you apply predefined ACLs, which are sometimes known as canned ACLs.

You can use the `acl` query string parameter for a Cloud Storage request exactly the same way you would use it for an Amazon S3 request. The `acl` parameter is used in conjunction with the `PUT` method to apply ACLs to the following: an existing object, an existing bucket, or a bucket you are creating. When you use the `acl` query string parameter in a `PUT` request, you must attach an XML document (using Cloud Storage ACL syntax) to the body of your request. The XML document contains the individual ACL entries that you want to apply to the bucket or object.

The following example shows a `PUT` request to Amazon S3 that uses the `acl` query string parameter. ACLs are defined in an XML document sent in the request body. The `PUT` request changes the ACLs on an object named `europe/france/paris.jpg` that is in a bucket named `my-travel-maps`. The ACL grants jane@gmail.com `FULL_CONTROL` permission.

Here is the same request to Cloud Storage:

Note that Cloud Storage does not require an `<Owner/>` element in the ACL XML document. For more information, see Bucket and object ownership (/storage/docs/access-control/lists#ownership).

You can also retrieve bucket and object ACLs by using the `acl` query string parameter with the `GET` method. The ACLs are described in an XML document, which is attached to the body of the response. You must have `FULL_CONTROL` permission to apply or retrieve ACLs on an object or bucket.

You can use the `x-goog-acl` header in a Cloud Storage request to apply predefined ACLs to buckets and objects exactly the same way you would use the `x-amz-acl` header in an Amazon S3 request. You typically use the `x-goog-acl` (`x-amz-acl`) header to apply a predefined ACL to a bucket or object when you are creating or uploading the bucket or object. The Cloud Storage predefined ACLs are similar to Amazon S3 Canned ACLs (http://docs.aws.amazon.com/AmazonS3/latest/dev/ACLOverview.html#CannedACL), including private, public-read, public-read-write, as well as others. For a list of Cloud Storage predefined ACLs, see Predefined ACLs (/storage/docs/access-control/lists#predefined-acl).

The following example shows a `PUT` Object request that applies the `public-read` ACL to an object named `europe/france/paris.jpg` that is being uploaded into a bucket named `my-travel-maps` in Amazon S3.

Here is the same request to Cloud Storage:

You can also use the `x-goog-acl` header to apply a predefined ACL to an existing bucket or object. To do this, include the `acl` query string parameter in your request but do not include an XML document in your request. Applying a predefined ACL to an existing object or bucket is useful if you want to change from one predefined ACL to another, or you want to update custom ACLs to a predefined ACL. For example, the following `PUT` Object request applies the predefined ACL `private` to an object named `europe/france/paris.jpg` that is in a bucket named `my-travel-maps`.

For more information about managing ACLs, see Creating and managing access control lists (/storage/docs/access-control/create-manage-lists).

Cloud Storage supports the same standard HTTP request methods for reading and writing data to your buckets as are supported in Amazon S3. Therefore, the majority of your tools and libraries that you currently use with Amazon S3, work as-is with Cloud Storage. Cloud Storage supports the following request methods:

- Service request for `GET`.
- Bucket requests, including `PUT`, `GET`, `DELETE`.
- Object requests, including `GET`, `POST`, `PUT`, `HEAD`, and `DELETE`.

For more information, see XML API Reference Methods (/storage/docs/xml-api/reference-methods). Keep in mind that when you send requests to Cloud Storage, you need to change the request body, when applicable, to use the appropriate Cloud Storage syntax. For example, when you create a lifecycle configuration for a bucket, use the Cloud Storage lifecycle XML, which is different than the Amazon S3 lifecycle XML.

There are a few differences between Cloud Storage XML API and Amazon S3 which are summarized below, with suggested Cloud Storage alternatives:

Amazon S3 Functionality	Cloud Storage XML API Functionality

Multipart upload. POST /<object-name>, PUT /<object-name>	In the Cloud Storage XML API, you can upload a series of component objects, performing a separate upload for each component. Then you can compose the objects (/storage/docs/xml-api/put-object-compose) into a single composite object. ★ **Note:** While the JSON API offers a multipart upload (/storage/docs/json_api/v1/how-tos/multipart-upload) feature, this feature is used for sending metadata along with object data. It is not equivalent to S3's multipart upload feature.
GET/POST bucket query string parameters: • "policy" - Working with Amazon S3 bucket policies. • "website" - Configuring bucket websites. • "tagging" - Tagging buckets for cost allocation purposes. • "notification" - Notifying for bucket events. • "requestPayment" - Configuring who pays for the request and the data download from a bucket.	Alternatives: • "policy" - Cloud Storage ACLs, project team membership, and the ability to use multiple projects address many of the scenarios where bucket policies are used. • "website" - Use the gsutil web (/storage/docs/gsutil/commands/web) command to manage websites, or try the JSON API (see buckets (/storage/docs/json_api/v1/buckets) resource). • "tagging" - Use multiple projects to track different cost centers. For more about projects, see Managing Projects (https://developers.google.com/console/help/#managingprojects). • "notification" - Use gsutil or JSON API Pub/Sub Notifications (/storage/docs/pubsub-notifications). • "requestPayment" - Use multiple projects with different billing profiles to manage who pays for requests and downloaded data from a bucket. For more about configuring billing, see Billing (https://developers.google.com/console/help/#billing) in the Google APIs Console Help documentation.
Multiple object delete. POST /?delete	Use the gsutil rm (/storage/docs/gsutil/commands/rm) command to easily remove multiple objects. The rm command supports the "-m" option to perform parallel (multi-threaded/multi-processing) deletes. Alternatively, the JSON API supports sending batch requests (/storage/docs/json_api/v1/how-tos/batch) to reduce the number of HTTP connections your client makes.

Cloud Storage uses several standard HTTP headers as well as several custom (extension) HTTP headers. If you are transitioning from Amazon S3 to Cloud Storage, you can convert your custom Amazon S3 headers to the equivalent Cloud Storage custom header or similar functionality as shown in the tables below.

For many Amazon S3 headers, you simply need to replace the `x-amz` prefix with `x-goog`:

Amazon S3 Header	Cloud Storage Header
`x-amz-storage-class`	`x-goog-storage-class`
`x-amz-acl`	`x-goog-acl`
`x-amz-date`	`x-goog-date`
`x-amz-meta-*`	`x-goog-meta-*`
`x-amz-copy-source`	`x-goog-copy-source`
`x-amz-metadata-directive`	`x-goog-metadata-directive`
`x-amz-copy-source-if-match`	`x-goog-copy-source-if-match`
`x-amz-copy-source-if-none-match`	`x-goog-copy-source-if-none-match`
`x-amz-copy-source-if-unmodified-since`	`x-goog-copy-source-if-unmodified-since`
`x-amz-copy-source-if-modified-since`	`x-goog-copy-source-if-modified-since`

Several headers differ or do not apply in Cloud Storage:

Amazon S3 Header	Cloud Storage Header
`x-amz-server-side-encryption`	Not required. Cloud Storage automatically encrypts all data before it is written to disk. For more information, see Encryption (/storage/docs/encryption).
`x-amz-grant-*`	`x-goog-acl` (/storage/docs/xml-api/reference-headers#xgoogacl) with a predefined ACL value.
`x-amz-mfa`	Use OAuth 2.0 Authentication (/storage/docs/authentication#oauth).
`x-amz-website-redirect-location`, `x-amz-copy-source-range`	n/a

See HTTP headers and query string parameters for XML API (/storage/docs/xml-api/reference-headers) for a reference to Cloud Storage headers.

For discussions about XML API interoperability, see Stack Overflow using the tag google-cloud-storage (http://stackoverflow.com/questions/tagged/google-cloud-storage). See the Getting support (/storage/docs/getting-support) page for more information about discussion forums and subscribing to announcements.