MODULE 3 · STUDY NOTES

Developing Storage Solutions

Amazon S3 for developers — buckets, objects, the PUT/GET/DELETE/SELECT operations, versioning, encryption and access control, and building with the AWS SDKs

Amazon Simple Storage Service (Amazon S3) is where a cloud developer keeps virtually anything — website assets, backups, IoT feeds, the raw material of a data lake — and this module is about wielding it from application code, not just clicking through the console. It builds from the ground up: what object storage is and why S3's eleven nines of durability matter, then the three components every developer must know (buckets, objects, and keys). From there it moves outward to the rules for creating and addressing buckets, the object operations that read and write data (PUT, GET, DELETE, and the SQL-driven SELECT), the versioning that guards against accidental loss, and finally the layers that protect data and control who may touch it: encryption in transit and at rest, identity-based and resource-based policies, presigned URLs, and CORS. The running thread is the café: Sofía is building a proof-of-concept static website hosted in S3 and locked down with a bucket policy — exactly the pattern the lab has you implement with the AWS CLI and the SDK for Python.

LEARNING OBJECTIVES

- Describe how Amazon Simple Storage Service (Amazon S3) can be used as a storage solution
- Identify the features and components of Amazon S3
- Describe two ways to protect data in Amazon S3
- Describe the function of the S3 object operations
- Explain how to manage access to Amazon S3 resources
- Develop with Amazon S3 by using the AWS software development kits (SDKs)

3.1 Introduction: Amazon S3 in a cloud application

The module opens with a business requirement from the café case study. Sofía has settled on a development environment and is ready to build. She wants a **proof-of-concept website** for the café — one that visually showcases the café's offerings to Frank and Martha — **without exposing the site to the outside world yet**. The solution highlighted in the course architecture is to host a **static website on Amazon S3** and protect the bucket with a **bucket policy** that limits the source of requests. That one scenario touches most of this module: creating a bucket, uploading objects, and applying an access policy — and it is exactly what you implement in the lab using the AWS CLI and the SDK for Python.

WHAT THIS MODULE COVERS

- **Introducing Amazon S3** — the service, its use cases, and its core components
- **Creating S3 buckets** — naming rules, Regions, URL styles, and prefixes
- **Working with S3 objects** — metadata and the PUT, GET, DELETE, and SELECT operations
- **Protecting data and managing access** — encryption, policies, presigned URLs, and CORS

3.2 Introducing Amazon S3

Amazon S3 is an object storage service designed to deliver virtually unlimited storage along with durability, availability, performance, and security. Developers reach for it to store backup files, Internet of Things (IoT) device data, big-data repositories, and much more. Its headline number is durability: S3 is **designed for 99.999999999% (11 nines) of durability**, which means that if you store 10,000,000 objects, on average you might lose a single object every 10,000 years. You can manage S3 from the console or **programmatically**, apply fine-grained access controls for compliance goals, and configure it to **respond to event triggers**.

> **EVENT NOTIFICATIONS**
>
> - S3 can generate notifications when an object is **created, deleted, or restored** — or lost from Reduced Redundancy Storage (RRS)
> - A notification can also fire when a **replication** event occurs
> - Targets: **AWS Lambda, Amazon SNS, or Amazon SQS**

S3 serves a wide range of use cases, from serving media to grounding analytics. The following table pairs each common use case with the S3 capability that delivers it.

TABLE 3.1 Common Amazon S3 use cases

USE CASE	HOW S3 DELIVERS IT
Content storage & distribution	Securely store images, video, and music and serve them directly or through Amazon CloudFront edge locations
Backup, restore & archive	Back up data and use storage lifecycle policies to move rarely accessed data to Amazon S3 Glacier
Data lakes & big data analytics	A centralized repository for structured and unstructured data at any scale, with query-in-place, analytics, and machine learning
Disaster recovery (DR)	Protect critical data without paying for a second physical site; Cross-Region Replication (CRR) asynchronously copies objects to buckets in other Regions
Static website hosting	Serve static pages and client-side scripts; S3 does not support server-side scripting such as PHP, JSP, or ASP.NET

The essential components of Amazon S3 are **buckets, objects, and keys**. A developer works with all three constantly, so it is worth pinning down exactly what each one is.

KEY TERMS

Bucket
A container for objects. Buckets organize the S3 namespace at the highest level, identify the account responsible for storage and data-transfer charges, play a role in access control, and are the unit of aggregation for usage reporting.

Object
The fundamental entity stored in S3 — any kind of file (text, video, photo, or another binary format) — consisting of the object data and its metadata (a set of name-value pairs that describe the object).

Object key
The value that uniquely identifies an object within a bucket. Each object in a bucket has exactly one key; an example key is preview.mp4.

TABLE 3.2 Region-specific reference formats

COMPONENT	ENDPOINT FORMAT	NOTE
Bucket	s3-<aws-region>.amazonaws.com/<bucket-name>/	Each bucket is Regional
Object	s3-<aws-region>.amazonaws.com/<bucket-name>/<object-key>	Adds the object key

Note that **each bucket is Regional**. You choose the AWS Region where S3 stores the buckets you create, and **objects stored in a Region never leave it** unless you explicitly transfer them to another Region. The Region is shown by its Region code — for example, a bucket created in the US West (Oregon) Region uses the code us-west-2.

3.3 Creating S3 buckets

When you create a bucket you give it a name, and the naming rules are strict because the **bucket namespace is shared by every AWS account**. A name must be **globally unique**: after a bucket is created, no other account can use that name in any Region until the bucket is deleted. To help guarantee uniqueness, teams often append the current date, the application name, or a randomly generated number to the name.

> BUCKET NAMING RULES
>
> - Must be **globally unique** across all AWS accounts
> - **3–63 characters** long
> - Only **lowercase letters, numbers, and hyphens (-)**
> - **No uppercase letters and no underscores (_)**
> - The name **cannot be changed** after the bucket is created

You must also **specify a Region** for the bucket, chosen with three deciding factors in mind: **latency, cost, and regulatory requirements** that could affect the data. As with the name, **a bucket's Region cannot be changed after creation**. When you connect to S3 programmatically, you connect to a URL that is the service **endpoint** for that Region.

You can access a bucket through the console or programmatically, and objects can be addressed with **virtual-host-style URLs** in which the bucket name is part of the domain name. This style is especially useful for **static website hosting**, which must be explicitly enabled and requires the bucket to allow **public read access**; a website-enabled bucket is served from a distinct **website endpoint**.

> TWO URL STYLES FOR REACHING A BUCKET
>
> - **Virtual-host-style** — the bucket name is part of the domain name, followed by the Regional S3 host and the object key (for example, cat.jpg in DOC-EXAMPLE-BUCKET); ideal for static website hosting
> - **Website endpoint** — a website-enabled bucket is served from a host that contains s3-website and the Region code; requires website hosting enabled and public read access

S3 has no true folders, but it supports the **folder concept** through **prefixes** — objects whose names begin with a common leading string. When you create a folder in the console, S3 actually creates an object whose name ends in a slash (/) and displays it as a folder. More importantly, when you query a bucket, **specifying a prefix limits the results to the keys that begin with that prefix**.

WORKED EXAMPLE – FILTERING RESULTS WITH A PREFIX

A bucket holds student English and math scores for 2021, with keys such as 2021/DOC-EXAMPLE-BUCKET/english/john.txt and 2021/DOC-EXAMPLE-BUCKET/math/maria.txt. To GET only the math-score keys for 2021, you specify the prefix 2021/DOC-EXAMPLE-BUCKET/math. S3 returns exactly the objects under that prefix — 2021/DOC-EXAMPLE-BUCKET/math/john.txt and 2021/DOC-EXAMPLE-BUCKET/math/maria.txt — and nothing from the english branch or the top-level summary.txt.

COMMON PITFALL

S3 has **no real folders**. What looks like a folder is a prefix convention — an object whose key ends in a slash. Querying and organization operate on **keys and prefixes**, not directories, so design your key names so that prefix filtering is useful.

3.4 Working with S3 objects

Every object is stored as **object data plus metadata**, and the metadata is a set of **key-value pairs** describing the object. There are two kinds, and the distinction matters as soon as you write code that sets or reads it.

SYSTEM-DEFINED METADATA	USER-DEFINED METADATA
S3 maintains it	*you assign it*
▪ S3 **controls** some values: object-creation date, object size, and object version ▪ You **can modify** others: storage-class configuration and server-side encryption ▪ S3 processes this system metadata as needed	▪ Extra information you attach during or after upload ▪ Keys **must begin with x-amz-meta-** (for example, x-amz-meta-alt-name) ▪ S3 **stores but does not process** it

The **PUT object** operation uploads an entire object to a bucket. You must have **write permission**, and S3 **always writes the whole object** — there are no partial updates. How you upload depends on the object's size.

TABLE 3.3 PUT: single upload vs. multipart upload

METHOD	WHEN TO USE	WHAT IT GIVES YOU
Single upload	Objects up to 5 GB in a single PUT operation	Simplest path — one request
Multipart upload	Recommended past 100 MB; required above 5 GB (up to 5 TB)	Parallel part uploads for throughput; resume after a failed part; parts sent independently and in any order, then assembled by S3

WORKED EXAMPLE – PUT WITH THE SDK FOR PYTHON (BOTO3)

A short snippet uploads core.css to a bucket. It imports the AWS SDK for Python (Boto3), then creates a client with boto3.client('s3', region_name='us-east-1'). It sets bucket_name and the local filename, then calls S3API.upload_file(filename, bucket_name, 'core.css', ExtraArgs={ ... }). The extra arguments set ContentType to text/css and CacheControl to max-age=0 — a cache value of zero tells web browsers **not to cache** the object, which is useful for files that change regularly.

The **GET object** operation retrieves objects and requires **read access**. You can fetch a **complete object** in one request or ask for a **range of bytes** — useful when network connectivity is poor or when your application can only process subsets of the object's data. A related operation, **SELECT**, retrieves object content with a **SQL expression**: you send the SQL plus a serialization format (JSON, CSV, or Apache Parquet), S3 parses the object into records, and it returns only the records that match. SELECT requires **s3:GetObject** permission, and because the filtering happens inside S3 it can improve query performance and reduce query cost substantially.

Versioning is a data-protection feature that keeps **multiple variants of an object in the same bucket**. Two objects can share a key but have different **version IDs** — for example, photo.gif with version 111111 and photo.gif with version 121212 — so you can preserve, retrieve, and restore each version and **recover from unintended overwrites, deletions, and application failures**. Versioning is **disabled by default**; once you enable it you can only **suspend it, never disable it**, and standard S3 rates apply to every version stored or requested. Versioned buckets also support **S3 Object Lock**, which prevents an object from being changed, overwritten, or deleted.

TABLE 3.4 DELETE object behavior

BUCKET STATE	WHAT DELETE DOES
Versioning disabled	Specify the object key to permanently delete a single object (or multiple objects) right away
Versioning enabled	Delete a specific version with a key and version ID; deleting by key alone adds a delete marker (retrieval returns 404), so you must delete each version to remove the object completely

WORKED EXAMPLE – THE MODULE'S SAMPLE EXAM QUESTION

Salespeople upload their sales figures daily. A solutions architect needs a durable storage solution for these documents that also protects against users accidentally deleting them. Which action protects against unintended user actions? The key phrases are **durable storage** and **protects against accidental deletion**. S3 is the durable store, and the feature that guards against accidental deletes is **versioning** — so the answer is to **store the data in an S3 bucket and enable versioning**. An EBS volume with weekly snapshots, two buckets in different Regions, and EC2 instance storage do not directly protect against a user deleting a document.

COMMON PITFALL

In a versioned bucket, a plain DELETE does **not** get rid of the object — it drops a **delete marker** and hides the object behind a 404, while every prior version (and its storage charges) remains until you delete each **version ID** explicitly.

3.5 Protecting data and managing access to Amazon S3 resources

Protecting data means guarding it **in transit** (as it travels to and from S3) and **at rest** (while it is stored in S3 data centers). S3 gives you options for both, and this section closes the module by adding the access controls that decide who may touch the data at all.

IN TRANSIT	AT REST
data on the move	*data in storage*
▪ **SSL/TLS-encrypted endpoints** accessed over HTTPS (clients must support TLS 1.0; AWS recommends TLS 1.2 or higher) ▪ **Client-side encryption** of the data before it is transmitted	▪ **Client-side encryption** — you encrypt before upload and manage the keys and tools ▪ **Server-side encryption** — S3 encrypts at the object level and decrypts on access, with three key-management options

TABLE 3.5 Server-side encryption options

OPTION	KEYS MANAGED BY	DETAILS
SSE-S3	Amazon S3	Each object gets a unique key using strong multi-factor encryption; the key is itself encrypted with a regularly rotated root key; uses AES-256
SSE-KMS	AWS KMS	Similar to SSE-S3 but uses an envelope key (which needs extra permissions) and adds an audit trail of when and by whom a key was used; use a default key or your own
SSE-C	You (customer-provided)	You manage the keys; S3 performs the encryption on write and the decryption on access

> **COMMON PITFALL**
>
> Enabling encryption is **not retroactive** — S3 encrypts only objects **added after** encryption is turned on. Objects already in the bucket stay unencrypted until you rewrite them.

By default, **all S3 resources are private** — only the AWS account that created a bucket or object can access it. You grant access by writing **access policies**, and S3 offers two types, distinguished by what the policy is attached to.

KEY TERMS

Identity-based policy
A policy attached to IAM users, groups, and roles in your account to grant them access to AWS resources, including S3.

Resource-based policy
A policy attached to the S3 resource itself; the two kinds are bucket policies and access control lists (ACLs).

Access control list (ACL)
A resource-based policy that grants basic read/write permissions at the bucket or object level. An object ACL is the only way to manage access to objects the bucket owner does not own, and ACLs are specified per object.

Bucket policy
A resource-based IAM policy that grants granular permissions on a bucket and its objects; it can supplement or replace ACLs and is the tool for managing cross-account permissions.

WORKED EXAMPLE – AN ANONYMOUS-READ BUCKET POLICY

A bucket policy is written in the IAM policy language. A classic example grants **anonymous read** access to every object in a bucket: a single statement with Effect set to Allow, a **wildcard principal** (meaning everyone), the action **s3:GetObject**, and a resource of the bucket's ARN followed by a wildcard so it covers all objects. This is the kind of policy that makes a static-website bucket publicly readable — and the reason you keep tight controls on everything else.

A **presigned URL** lets you give someone temporary permission to **PUT or GET a specific object** without changing any AWS security credentials or permissions. The URL simply carries the permissions of the user who created it, so it is the right tool when you want a user to upload or download one particular object and nothing more.

CREATING A PRESIGNED URL – WHAT YOU PROVIDE

- Your **security credentials**
- A **bucket name** and an **object key**
- An **HTTP method** — PUT to upload, GET to retrieve
- An **expiration date and time** — the URL is valid only until then, up to a **maximum of one week**
- The AWS CLI can generate presigned URLs for downloads from just the object address (s3://bucket-name/object-key)

Finally, **cross-origin resource sharing (CORS)** defines how a client web application loaded in **one domain** can interact with resources in a **different domain**. For example, JavaScript on http://www.example.com can use your bucket's resources only if CORS is enabled on the bucket; likewise, a page that loads a **web font** hosted in your bucket triggers a CORS check. You enable CORS by creating a **CORS configuration in JSON** that lists the allowed origins, the allowed operations (HTTP methods), and other rule details — up to **100 rules** — and you apply it from the console, the AWS SDKs, or the S3 REST API.

IN THE LAB – WORKING WITH AMAZON S3 (LAB 3.1, ~60 MINUTES)

- Connect to an IDE and configure the environment
- Create an S3 bucket by using the **AWS CLI**
- Set a **bucket policy** on the bucket by using the **SDK for Python**
- Upload objects to the bucket to build the website
- Test access to the website, then analyze the website code

Module summary

- Amazon S3 is an object storage service with virtually unlimited capacity, designed for 11 nines (99.999999999%) of durability, and able to fire event notifications to AWS Lambda, Amazon SNS, or Amazon SQS; use cases include content distribution, backup/archive, data lakes, DR, and static website hosting.
- The three core components are the bucket (a Regional, globally named container), the object (data plus metadata), and the object key (unique per object). Objects never leave their Region unless you explicitly transfer them.

- Bucket names are globally unique across all AWS accounts, 3–63 characters, and use only lowercase letters, numbers, and hyphens; neither the name nor the Region can change after creation, and the Region choice weighs latency, cost, and regulatory needs.
- Objects are addressed with virtual-host-style URLs (bucket name in the domain) or, for a website-enabled bucket, a website endpoint; prefixes imply folder structure and filter query results to matching keys.
- Object metadata is system-defined (some values S3-controlled, some modifiable) or user-defined (keys begin with x-amz-meta-, and S3 does not process it).
- The object operations: PUT writes whole objects (single ≤ 5 GB, multipart required above 5 GB up to 5 TB), GET reads a whole object or a byte range, DELETE removes objects, and SELECT queries contents with SQL.
- Versioning keeps multiple variants under one key with version IDs to recover from accidental changes; it is off by default and, once enabled, can only be suspended; a key-only DELETE in a versioned bucket adds a delete marker (404) rather than erasing versions.
- Protect data in transit with SSL/TLS over HTTPS or client-side encryption, and at rest with client-side or server-side encryption (SSE-S3, SSE-KMS, SSE-C); encryption applies only to objects added after it is enabled.
- S3 resources are private by default; grant access with identity-based or resource-based policies (bucket policies and ACLs), hand out temporary single-object access with presigned URLs (max one week), and enable CORS to allow cross-domain browser access.

Review questions

1. What are the three core components of Amazon S3, and what does each do?

Answer: The bucket (a Regional container that also identifies the billing account and anchors access control), the object (the stored data plus its metadata), and the object key (the value that uniquely identifies an object within a bucket — exactly one per object).

2. List the S3 bucket naming rules, and state what is immutable after creation.

Answer: Globally unique across all AWS accounts, 3–63 characters, and only lowercase letters, numbers, and hyphens — no uppercase or underscores. After creation, both the bucket name and its Region are fixed.

3. When must you use a multipart upload rather than a single PUT, and what does multipart add?

Answer: A single PUT handles objects up to 5 GB; multipart is required above 5 GB (max object 5 TB) and recommended past 100 MB. Multipart uploads parts in parallel for throughput, lets you resume after a failed part, and has S3 assemble the parts into the final object.

4. How do system-defined and user-defined object metadata differ?

Answer: System-defined metadata is maintained by S3 — it controls some values (creation date, size, version) and lets you modify others (storage class, server-side encryption). User-defined metadata is information you attach; its keys must start with x-amz-meta-, and S3 stores but does not process it.

5. In a versioning-enabled bucket, what happens when you DELETE with only the object key, and how do you actually remove the object?

Answer: S3 adds a delete marker that becomes the current version, so retrieving the key returns 404 while earlier versions remain. To remove the object completely, you must delete each individual version by its version ID.

6. Name S3's three server-side encryption options and the key difference among them, plus the one limitation of enabling encryption.

Answer: SSE-S3 (S3 manages the keys; AES-256), SSE-KMS (AWS KMS manages the keys, with an envelope key and an audit trail), and SSE-C (you provide the keys). Enabling encryption is not retroactive — only objects added afterward are encrypted.

---

7. What is the difference between identity-based and resource-based policies in S3, and what are the two resource-based mechanisms?

Answer: Identity-based policies attach to IAM users, groups, and roles to grant them access; resource-based policies attach to the S3 resource itself. The two resource-based mechanisms are bucket policies (granular, cross-account) and ACLs (basic read/write at the bucket or object level).

---

8. How does a presigned URL work, and what are its limits?

Answer: It grants temporary access to PUT or GET a single object using the permissions of the user who created it; you provide security credentials, a bucket name, an object key, an HTTP method, and an expiration. It is valid only until that expiration, with a maximum of one week.
