MODULE 4 · STUDY NOTES

# Adding a Storage Layer with Amazon S3

*How Amazon S3 object storage works — buckets and objects, storage classes and lifecycle rules, durability and availability, versioning, and the access controls that keep a storage layer secure*

Amazon S3 is the storage layer of a cloud architecture — an object storage service that holds virtually unlimited amounts of unstructured data as objects inside buckets. This module builds S3 from the ground up: what object storage is and how it differs from block and file storage; the anatomy of buckets, objects, keys, and prefixes; and the durability, availability, and performance that make S3 dependable. It then turns to how architects actually use S3 — media hosting, static websites, analytics data stores, and backup — and how to move data in and out efficiently with multipart upload, S3 Transfer Acceleration, and AWS Transfer Family. The heart of the module is managing content over its life: choosing the right storage class, automating movement with lifecycle rules, and protecting objects with versioning. Finally, it covers designing a secure, well-placed storage layer — encryption, access control, Region selection, and cost — and maps every choice back to the AWS Well-Architected Framework. Throughout, the architect's job is the same: start from the business need and choose S3 configurations that optimize cost while meeting performance and compliance requirements, and apply security best practices that protect the data against unwanted access and accidental loss.

LEARNING OBJECTIVES

- Define Amazon Simple Storage Service (Amazon S3) and how it works
- Recognize the problems that Amazon S3 can solve
- Describe how to move data to and from Amazon S3
- Manage the storage of content efficiently by using Amazon S3
- Recommend the appropriate use of Amazon S3 based on requirements
- Configure a static website on Amazon S3
- Use the AWS Well-Architected Framework principles when designing a storage layer with Amazon S3

## 4.1 Defining Amazon S3

Storage comes in three basic types, and you will likely work with all of them: **block storage** (data in fixed-size blocks that applications and file systems manage), **file storage** (a hierarchical shared file system, like a company network drive), and **object storage** (each file kept as an object based on its attributes and metadata, where an update rewrites the *entire* object). Amazon S3 is an object storage service.

**TABLE 4.1** The three types of storage

| TYPE | HOW DATA IS STORED | HOW IT WORKS |
|---|---|---|
| **Block** | In fixed-size blocks, each with a unique identifier | Applications and file systems control how blocks are accessed, combined, and modified; blocks can span systems and OSes |
| **File** | In a hierarchical structure | A shared file system that users, applications, and services access — like a centralized network drive |
| **Object** | As objects based on attributes and metadata | Each object = data + metadata + object key; updating rewrites the whole object |

**Amazon S3 stores virtually unlimited amounts of unstructured data** as objects in a **bucket** that you define. Every bucket name must be **globally unique across all Regions and all AWS accounts**, and objects have a globally unique URL (a universal namespace). A single object ranges from **0 bytes to 5 TB**, but total storage is effectively unlimited. Each object has five consistent characteristics.

### THE FIVE CHARACTERISTICS OF AN S3 OBJECT

- **Key** — the name you assign; you use it to retrieve the object, and it holds the full path relative to the bucket root (S3 does not really have directories).
- **Version ID** — together with the key, uniquely identifies an object in a bucket.
- **Value** — the actual content, any sequence of bytes; it is **immutable**, so to change it you re-upload the object.
- **Metadata** — name-value pairs describing the object (user-defined plus S3 system metadata); **sub-resources** hold additional object-specific information.

A **bucket** is a container for objects: it organizes the S3 namespace at the highest level, identifies the account responsible for charges, plays a role in access control, and is the unit for usage reporting. An **object** is a file plus the metadata that describes it (such as a content type that tells a browser how to render the file), uniquely identified by its **object key** — each object has exactly one key. Each bucket is **Regional**: you choose the Region, and objects never leave it unless you explicitly transfer them. A Region-specific endpoint takes the form `https://s3-<aws-region>.amazonaws.com/<bucket-name>/<object-key>`.

### WORKED EXAMPLE – PREFIXES IMPLY A FOLDER STRUCTURE

S3 has no true folders, but it supports the folder concept with **shared name prefixes**. Suppose a bucket named graphics-bucket holds objects organized by year, such as photos/2022/catpiano.jpg and photos/2021/lakefront.png. A GET or LIST query with the prefix **photos/2022** returns only the objects whose keys begin with that 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.

**TABLE 4.2** Key benefits of Amazon S3

| BENEFIT | WHAT IT MEANS |
|---|---|
| **Durability** | Data is not lost — S3 Standard is designed for 11 nines (99.999999999%). Objects are stored redundantly across multiple Availability Zones, S3 self-heals lost redundancy, and integrity is verified with checksums |
| **Availability** | Data is reachable when needed — S3 Standard is designed for 4 nines (99.99%); capacity to store data is effectively unlimited |
| **High performance** | Thousands of transactions per second when uploading and retrieving; automatically scales to high request rates |

> **COMMON PITFALL**
>
> Keep durability and availability straight. **Durability** (11 nines) is the probability an object is not lost over a year; **availability** (4 nines) is the probability you can reach it right now. A design can be extremely durable yet briefly unavailable — they measure different things.

## 4.2 Using Amazon S3

Once you know what S3 offers, the question is how customers put it to work. It handles **spikes in demand**, hosts **static sites**, serves as a **data store for analysis**, and supports **disaster recovery and backup** — four common use cases that combine these features into robust architectures.

**TABLE 4.3** Four common Amazon S3 use cases

| USE CASE | HOW S3 FITS |
|---|---|
| **Media hosting** | Store and distribute video, photos, and music; every object has a unique HTTP URL, and S3 can be the origin store for a CloudFront CDN. Its elasticity suits fast-growing sites with user-generated content |
| **Static website** | Host HTML, images, videos, and client-side scripts with no server to run — low cost, high scalability and availability |
| **Computation and analytics** | A data store for financial transaction analysis, clickstream analytics, or media transcoding; horizontal scaling supports many concurrent transactions |
| **Backup and archive** | Highly durable, scalable backup from on-premises data centers and EC2 instances; cross-Region replication and Glacier extend it |

Two use cases carry exam-relevant detail. To host a **static website**, configure an S3 bucket for website hosting, attach a **bucket policy** that allows access to the objects, and upload your content (which may include client-side scripts such as JavaScript). S3 does **not support server-side scripting** (PHP, JSP, ASP.NET), so a dynamic site needs other AWS services; the static-site URL takes the form http://bucket-name.s3-website.Region.amazonaws.com. For **backup and archive**, S3's durability makes it a strong backup target for on-premises and EC2 data — **cross-Region replication** automatically and **asynchronously** copies new objects to buckets in other Regions, and you can move long-term data to **Amazon S3 Glacier**.

**WORKED EXAMPLE – AN ANALYTICS DATA-INTEGRATION PATTERN**

S3 often anchors large-scale analytics. A temporary compute cluster is spun up — an **EC2 Spot Fleet** when Spot prices are low, or an **Amazon EMR** cluster — which extracts raw data from an S3 bucket (and another source), transforms it, and loads the processed data into a **different S3 bucket**. The compute is then terminated to save cost, and a tool such as **Amazon QuickSight** harvests insights from the processed data. S3's horizontal scaling and concurrent transactions make it well suited to this role.

**COMMON PITFALL**

Hosting a static website is the one common case where you deliberately allow public read access to a bucket. For almost every other use case — data an application consumes, or sensitive backups — you should **never** grant public access. And remember S3 serves only static content: it does not run server-side scripts, so dynamic sites need additional AWS services.

## 4.3 Moving data to and from Amazon S3

When you upload a file to S3 it is stored as an object, and there is **no limit to the number of objects** in a bucket. Uploading requires **write permission** for the bucket. Uploaded objects are **encrypted by default**: during upload S3 automatically applies server-side encryption with S3-managed keys (SSE-S3), and during download the object is decrypted.

**TABLE 4.4** Options for uploading objects to Amazon S3

| METHOD | USE IT TO | NOTES |
|---|---|---|
| **AWS Management Console** | Move data in or out with a wizard-based, drag-and-drop approach | Maximum file size 160 GB |
| **AWS CLI** | Upload or download from a terminal command or a script | Multipart upload, up to 5 TB |
| **AWS SDKs** | Upload objects programmatically using wrapper libraries | Multipart upload, up to 5 TB |
| **Amazon S3 REST API** | Send a PUT request to upload data in a single operation | Multipart upload, up to 5 TB |

To upload a file larger than the console's 160 GB limit, use the CLI, SDKs, or REST API, which upload an object in parts using the **multipart upload** API — a single object can be up to **5 TB**. You upload the object as a set of independent parts (in any order); if a part fails, you retransmit just that part, and S3 assembles them. S3 does this for you behind the scenes when you use the console or SDKs.

**WHY MULTIPART UPLOAD HELPS**

- **Improve throughput** — parts upload in parallel.
- **Recover quickly from network issues** — a smaller part size limits the cost of restarting a failed part.
- **Pause and resume** — there is no expiry once initiated; you must explicitly complete or stop the upload.
- **Begin before you know the final size** — you can upload an object as you are creating it.
- AWS recommends multipart upload when an object reaches **100 MB or greater**.

**S3 Transfer Acceleration** is a **bucket-level** feature for fast, secure transfers of files over long distances between your client and a bucket. It routes data through the globally distributed **CloudFront edge locations** onto an optimized

network path into S3, improving speed by **50–500%** on average for cross-country transfers of larger objects. Consider it when customers upload to a centralized bucket from around the world, when you move gigabytes to terabytes across continents regularly, or when you cannot saturate your available internet bandwidth uploading to S3.

**AWS TRANSFER FAMILY**

- A **fully managed, serverless** service that scales in real time; no file-transfer infrastructure to run and no upfront cost — you pay only for use.
- Transfers files into and out of **Amazon S3** storage or **Amazon EFS** file systems.
- Supports the **SFTP** (SSH File Transfer Protocol) v3, **FTPS**, **FTP**, and **AS2** protocols.
- Use cases — S3: data lakes for third-party uploads, subscription-based data distribution, internal transfers; EFS: data distribution, supply chain, content management, web-serving apps.

**COMMON PITFALL**

The 160 GB limit is a **console** limit, not an S3 limit. To move larger files — up to the 5 TB object maximum — use the CLI, SDKs, or REST API, which upload in parts via **multipart upload**. When you use the console or SDKs, S3 performs multipart upload for you automatically.

## 4.4 Storing content with Amazon S3

S3 provides several **storage classes** aligned to different requirements, trading off retrieval latency, resilience, and cost. **General purpose:** S3 Standard for frequently accessed data. **Intelligent tiering:** S3 Intelligent-Tiering automatically moves objects to the most cost-effective access tier based on access frequency, with no retrieval fees or operational overhead. **Infrequent access:** S3 Standard-IA (millisecond access, lower storage cost, higher retrieval cost) and S3 One Zone-IA (a single Availability Zone, for lower-cost, re-creatable data). **Archive:** S3 Glacier Instant Retrieval (millisecond retrieval), S3 Glacier Flexible Retrieval (asynchronous retrieval in minutes, 1–2×/year), and S3 Glacier Deep Archive (lowest cost, 7–10-year regulatory retention). Every class shares **11 nines of durability**, millisecond first-byte latency, and lifecycle eligibility; all provide 99.9% availability (One Zone-IA 99.5%) backed by a 99% SLA. (S3 on Outposts extends object storage to on-premises Outposts hardware.)

**TABLE 4.5** S3 storage classes – charge structure

| STORAGE CLASS | AZS | MIN CAPACITY CHARGE | MIN DURATION | RETRIEVAL FEE |
|---|---|---|---|---|
| **S3 Standard** | ≥3 | N/A | N/A | N/A |
| **S3 Intelligent-Tiering** | ≥3 | N/A | N/A | N/A |
| **S3 Standard-IA** | ≥3 | 128 KB | 30 days | Per GB |
| **S3 One Zone-IA** | 1 | 128 KB | 30 days | Per GB |
| **S3 Glacier Instant Retrieval** | ≥3 | 128 KB | 90 days | Per GB |
| **S3 Glacier Flexible Retrieval** | ≥3 | N/A | 90 days | Per GB |
| **S3 Glacier Deep Archive** | ≥3 | N/A | 180 days | Per GB |

An **S3 Lifecycle configuration** is a set of rules defining the actions S3 applies to a group of objects. **Transition actions** move objects to another storage class — for example, to Standard-IA 30 days after creation or to Glacier a year later

(transition requests carry a cost). **Expiration actions** define when objects expire, and S3 deletes them on your behalf. Once a policy is set, **data transfers automatically without any application changes**, so you pay less as data ages. You can set rules per object or per bucket.

> **WORKED EXAMPLE – THE CAFÉ WEBSITE LIFECYCLE**
>
> In the café lab, versioning is enabled, so older copies of the site accumulate and drive up storage cost. Two lifecycle rules keep that in check: the first automatically **transitions previous versions of every object to S3 Standard-IA after 30 days** (cheaper storage for files that are no longer current), and the second **deletes previous versions after 365 days** (so you stop paying to store copies you will never restore). The application never changes — S3 applies the rules on your behalf.

**Versioning** protects objects from accidental overwrites and deletes. You enable it at the **bucket level**; by default it is **disabled** (the version ID is null). A bucket has three states: **enabled** (a same-key upload keeps the old object and adds a new version ID; a delete adds a delete marker but leaves prior versions retrievable), **disabled** (the default — overwrites, and deletes are permanent), and **suspended** (existing versions are kept, but new writes behave as if versioning were off). Once enabled, versioning **cannot be disabled — only suspended**. There is no charge for versioning itself, but each version is a full copy that adds to storage cost.

**TABLE 4.6** Versioning operations, step by step (versioning enabled)

| OPERATION | WHAT AMAZON S3 DOES |
|---|---|
| **PUT with the same key** | Keeps the original object and adds a new version ID; both versions stay retrievable |
| **DELETE (no version ID)** | Inserts a delete marker as the current version; prior versions remain retrievable by version ID |
| **GET (no version ID)** | Returns the most recent version — but returns 404 Not Found when the current version is a delete marker |
| **GET with a version ID** | Returns that specific version even if it is not the current one |
| **DELETE with a version ID** | Permanently removes that version with no delete marker; only the bucket owner can do this |

**Cross-origin resource sharing (CORS)** defines how client web applications loaded in one domain can interact with resources in another. You create a **CORS configuration** — an XML document identifying the origins you allow, the HTTP methods supported for each, and other details — and when a browser sends a preflight request, S3 applies the matching rule. A classic example is hosting a **web font** that a webpage in another domain needs to load.

The **S3 data consistency model** provides **strong read-after-write and list consistency automatically** for all new and existing objects in **all AWS Regions**: after a successful PUT, any following GET or LIST returns the data that was written. This benefits big data workloads and simplifies migrating on-premises analytics (no need for extra infrastructure such as S3Guard). One caveat: while objects are strongly consistent, **bucket configurations are only eventually consistent** — a just-deleted bucket might briefly still appear in a list-buckets result.

> **COMMON PITFALL**
>
> Two versioning gotchas trip up exam takers. First, once you **enable** versioning you can never disable it — only **suspend** it. Second, do not confuse consistency levels: S3 **objects** are strongly read-after-write consistent in every Region, but bucket-level *configuration* changes are only eventually consistent.

## 4.5 Designing with Amazon S3

Design starts with **secure defaults**. S3 buckets and the objects in them are **private and protected by default** — the only entities that can access a new, unmodified bucket are the account administrator and the AWS account root user. Buckets also have **encryption configured by default**, with **SSE-S3** as the default. When a use case requires sharing S3 data, you must actively **manage and control access** and follow the **principle of least privilege**. Encryption encodes legible data so it is unreadable without the secret key; you can optionally use **AWS Key Management Service (AWS KMS)** to manage keys.

**TABLE 4.7** Server-side vs. client-side encryption

| ASPECT | SERVER-SIDE ENCRYPTION (SSE) | CLIENT-SIDE ENCRYPTION |
|---|---|---|
| **Who encrypts** | S3 encrypts objects before saving to disk and decrypts them on download | You encrypt the data on the client, then upload the ciphertext |
| **How to enable** | Turn on the bucket's default encryption option | You manage the process, the keys, and the tools yourself |
| **Key options** | SSE-S3 (default, S3-managed keys), SSE-KMS, DSSE-KMS (dual-layer), and SSE-C (customer-provided keys) | Your own keys, kept separate from the store that holds the data |

**TABLE 4.8** Tools for protecting buckets and objects

| TOOL | WHAT IT DOES |
|---|---|
| **Block Public Access** | Overrides any other policy or permission to keep buckets from being publicly accessible — the simplest guard against accidental exposure |
| **IAM policies** | Specify which users or roles can access specific buckets and objects (authenticate via IAM) |
| **Bucket policies** | Rule-based access attached to a bucket; can grant cross-account or public access, or use deny statements to restrict — the preferred method over ACLs |
| **Access control lists (ACLs)** | Older, less-common access rules on buckets and objects; predate IAM — do not set them too permissively |
| **S3 access points** | Named network endpoints attached to a bucket, each with names and permissions tailored to a specific application |
| **Presigned URLs** | Grant time-limited access to an object with a temporary URL |
| **AWS Trusted Advisor** | A bucket-permission check that flags buckets granting global (public) access |

These tools support **three general approaches to configuring access** — from fully private by default, to controlled access for specific users and resources, to public access, which is appropriate only for special cases like a static website and **not recommended** for data, backups, or anything sensitive.

**TABLE 4.9** Three approaches to configuring access

| APPROACH | WHO CAN ACCESS | WHEN TO USE IT |
|---|---|---|
| **Default (private)** | Only the account administrator and AWS account root user | Every new, unmodified bucket |
| **Access policy applied (controlled)** | Specific users or resources you explicitly grant; others are denied | The common case for most workloads |
| **Public access** | Anyone, publicly | Special cases like a static website; not recommended for data |

**TABLE 4.10** Considerations when choosing a Region

| CONSIDERATION | WHY IT MATTERS |
|---|---|
| **Data privacy and compliance** | Data is subject to the laws where it is stored; regulations such as HIPAA dictate how and where data can live |
| **Proximity of users to data** | Even small latency differences affect customer experience — choose the Region closest to your users |
| **Service and feature availability** | Not all AWS services and features exist in every Region; availability expands over time |
| **Cost-effectiveness** | Prices vary by Region, and transferring data out of a Region has a cost |

**Amazon S3 Inventory** helps you manage storage: use it to **audit and report on the replication and encryption status** of your objects for compliance and to speed up big data jobs. It is a **scheduled alternative to the synchronous List API**, outputting CSV, ORC, or Parquet files (daily or weekly) that you can query with Amazon Athena or Redshift Spectrum. On **cost**, S3 is pay-for-what-you-use with no minimum fee.

### HOW AMAZON S3 COSTS WORK

- **Storage** — gigabytes stored per month, priced by Region and storage class.
- **Requests** — PUT, COPY, POST, LIST, or lifecycle transitions to move data into any storage class.
- **No charge** for data transferred *in* from the internet, the first **100 GB/month out** to the internet, between buckets or to any service in the **same Region**, and out to CloudFront (Intelligent-Tiering adds a small monitoring charge but no retrieval fees).

### WORKED EXAMPLE – CONTROLLED ACCESS FOR CAFÉ EMPLOYEES

The café owner wants employees — not the public — to reach internal documents (cooking instructions, vendor information, payroll, compliance training). The answer is **controlled access**: create a folder (prefix) for each content type, then write **IAM policies** — per group or per user — that allow or deny access based on each employee's role. Because buckets are private by default, only the people you explicitly grant access can see the files.

### COMMON PITFALL

**Block Public Access overrides everything.** If it is enabled, even a permissive bucket policy or ACL will not expose the bucket, so turn it on for every bucket you do not intend to make public. For deliberate sharing, prefer **bucket policies over ACLs** and combine them with IAM for least-privilege access.

## 4.6 Applying the AWS Well-Architected Framework principles to storage

The AWS Well-Architected Framework has **six pillars**, each with best practices to consider when architecting. Four are most relevant to an S3 storage layer: **Security, Reliability, Performance Efficiency, and Cost Optimization** — and S3 already does much of this work for you, engineered so architects need not build durability, redundancy, or default encryption themselves.

**TABLE 4.11** Well-Architected best practices met by Amazon S3

| PILLAR | BEST PRACTICE | HOW AMAZON S3 SUPPORTS IT |
|---|---|---|
| **Security** | Protect data at rest — enforce encryption and access control | Encrypted (SSE-S3) and private by default; IAM and bucket policies for least privilege; versioning; Block Public Access |
| **Performance Efficiency** | Architecture selection — know the services, factor in cost | Massive unstructured object storage; Transfer Acceleration and multipart upload; storage classes and Intelligent-Tiering matched to access patterns |
| **Cost Optimization** | Cost-effective resources — analyze cost across usage over time | Storage classes with lifecycle rules; Intelligent-Tiering; S3 Inventory to audit how storage is used |
| **Reliability** | Failure management — use fault isolation across locations | 11 nines durability and 4 nines availability; objects stored redundantly across multiple Availability Zones; checksums verify integrity |

COMMON PITFALL

On a scenario that asks how to protect S3 documents from accidental deletion or overwrite, the answer is **versioning** — not a second bucket, not Infrequent Access, and not S3 Inventory. A second bucket guards against a Regional outage but not user error; Infrequent Access only lowers cost; and S3 Inventory reports on objects but cannot prevent a deletion.

## Module summary

- Amazon S3 is object storage for virtually unlimited unstructured data — files kept as objects (data + metadata + key) in buckets, unlike block or file storage. Bucket names are globally unique across all AWS accounts, each bucket is Regional, an object can reach 5 TB, and the namespace is flat (prefixes imply folders).
- S3 Standard is designed for 11 nines of durability and 4 nines of availability, stores objects redundantly across multiple AZs, scales to thousands of transactions per second, and is strongly read-after-write consistent in all Regions.
- Common use cases: media hosting (unique URLs, CloudFront origin), static websites (no server-side scripting), analytics data stores, and backup/archive with asynchronous cross-Region replication and Glacier.
- Move data with the console (160 GB max), CLI, SDKs, or REST API; use multipart upload (100 MB+, up to 5 TB), S3 Transfer Acceleration for long distances, and AWS Transfer Family (SFTP, FTPS, FTP, AS2).
- Manage content over its life: match the storage class to the access pattern, automate transitions and expirations with lifecycle rules, and protect objects with versioning — suspend-only once enabled, at one storage copy per version.

- S3 is private and SSE-S3-encrypted by default; control access with Block Public Access, IAM and bucket policies (over ACLs), access points, presigned URLs, and Trusted Advisor; choose a Region for compliance, proximity, availability, and cost; and map it to four Well-Architected pillars.

## Review questions

**1. What is object storage, and how is a single S3 object structured?**

*Answer:* Object storage keeps whole files as objects in a flat namespace. Each object is the value (data), metadata (name-value pairs), and an object key, plus a version ID and sub-resources; values are immutable, so you re-upload to change one.

**2. How is durability different from availability, and what is S3 Standard designed for?**

*Answer:* Durability (11 nines, 99.999999999%, for S3 Standard) means objects are not lost — via redundant storage across multiple Availability Zones and checksums. Availability (4 nines, 99.99%) means you can reach the data when needed.

**3. A customer needs archive storage that is rarely accessed but must be retrieved in milliseconds when it is. Which storage class fits?**

*Answer:* S3 Glacier Instant Retrieval — the lowest-cost archive class with millisecond retrieval (for example, medical images). S3 Glacier Flexible Retrieval and Deep Archive retrieve asynchronously in minutes to hours.

**4. What are the two kinds of actions in an S3 Lifecycle configuration, and what does each do?**

*Answer:* Transition actions move objects to another storage class (e.g., Standard-IA after 30 days); expiration actions delete objects on your behalf when they expire — automatically, with no application changes.

**5. In a versioning-enabled bucket, what happens on a same-key PUT, a DELETE without a version ID, and a GET when the current version is a delete marker?**

*Answer:* The PUT keeps the old object and adds a new version ID (both retrievable); the DELETE inserts a delete marker but leaves prior versions retrievable by version ID; a GET whose current version is a delete marker returns a 404 Not Found error.

**6. By default, are S3 buckets public or private, and are objects encrypted? Name three tools you would use to control sharing.**

*Answer:* Both buckets and objects are private by default — only the account administrator and root user have access until it is explicitly granted — and every new object is encrypted at rest with SSE-S3 by default. Control sharing with any of: Block Public Access, IAM policies, bucket policies (preferred over ACLs), ACLs, S3 access points, presigned URLs, and AWS Trusted Advisor's bucket-permission check.