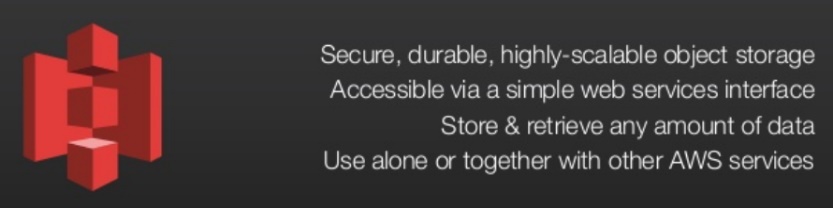
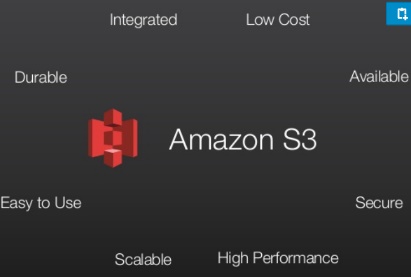
The need for **storage** is increasing every day, so building and maintaining your own repositories, therefore, becomes a tedious and tiresome job because knowing the amount of capacity you may need in the future is difficult to predict. You may either over-utilize it leading to an application failure because of not having sufficient space or you may end up buying stacks of storage which will then be under-utilized.

Keeping all these hassles in mind, Amazon came up with an internet storage service called *AWS S3.*

**

**What is AWS S3 ?**

Amazon Simple Storage Service (S3) is a storage for the internet. It is designed for large-capacity, low-cost storage provision across multiple geographical regions. Amazon S3 provides developers and IT teams with **Secure**, **Durable** and **Highly Scalable**object storage.

S3 is **Secure** because AWS provides:

* Encryption to the data that you store. It can happen in two ways:
  + Client Side Encryption
  + Server Side Encryption
* Multiple copies are maintained to enable regeneration of data in case of data corruption
* *Versioning,* wherein each edit is archived for a potential retrieval.

S3 is **Durable** because:

* It regularly verifies the integrity of data stored using checksums e.g. if S3 detects there is any corruption in data, it is immediately repaired with the help of replicated data.
* Even while storing or retrieving data, it checks incoming network traffic for any corrupted data packets.

S3 is **Highly Scalable**, since it automatically scales your storage according to your requirement and you only pay for the storage you use.

*The next question which comes to our mind is,*

**What kind and how much of data one can store in AWS S3?**

You can store virtually any kind of data, in any format, in S3 and when we talk about capacity, the volume and the number ofobjects that we can store in S3 are unlimited.

\**An object* is the fundamental entity in S3. It consists of data, key and metadata.

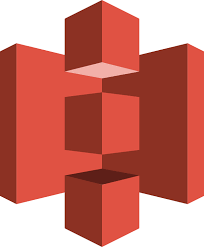
When we talk about data, it can be of two types-

* Data which is to be accessed frequently.
* Data which is accessed not that frequently.

Therefore, Amazon came up with 3 storage classes to provide its customers the best experience and at an affordable cost.

**Let’s understand the 3 storage classes with a “health-care” use case:**

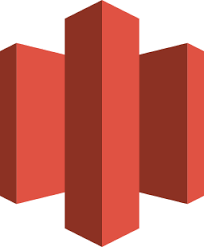
*1.Amazon S3 Standard* for frequent data access

This is suitable for performance sensitive use cases where the latency should be kept low. e.g. in a hospital, frequently  accessed data will be the data of admitted patients, which should be retrieved quickly.

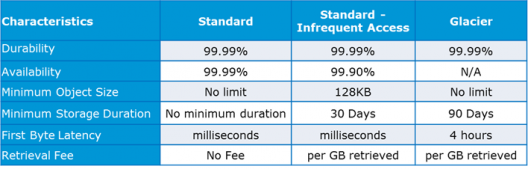
*2. Amazon S3 Standard* for infrequent data access

This is suitable for use cases where the data is long lived and less frequently accessed, i.e for data archival but still expects high performance. e.g. in the same hospital, people who have been discharged, their records/data will not be needed on a daily basis, but if they return with any complication, their discharge summary should be retrieved quickly.

*3.Amazon Glacier*

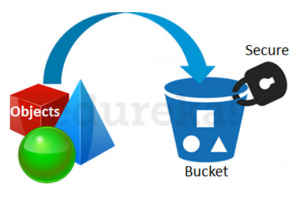
** Suitable for use cases where the data is to be archived, and high performance is not required, it has a lower cost than the other two services.e.g. in the hospital, patients’ test reports, prescriptions, MRI, X Ray, Scan docs etc. that are older than a year will not be needed in the daily run and even if it is required, lower latency is not needed.

**Specification Snapshot:**Storage Classes



**How is data organized in S3?**

Data in S3 is organized in the form of buckets.

****

* A Bucket is a logical unit of storage in S3.
* A Bucket contains objects which contain the data and metadata.

Before adding any data in S3 the user has to create a bucket which will be used to store objects.

**Where is your data stored geographically?**

You can self-choose where or in which region your data should be stored. Making a decision for the region is important and therefore it should be planned well.

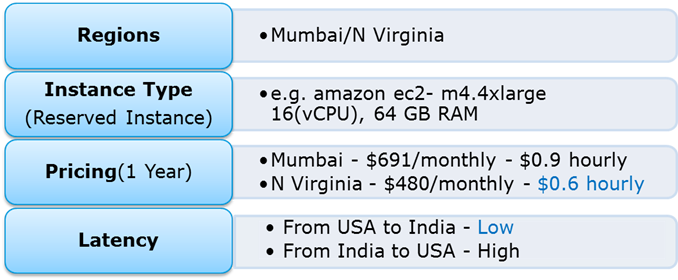
These are the 4 parameters to choose the optimal region –

* Pricing
* User/Customer Location
* Latency
* Service Availability

**Let’s understand this through an example:**

*Suppose there is a company which has to launch these storage instances to host a website for the customers in the US and India.*

*To provide the best experience, the company has to choose a region, which best fits its requirements.*



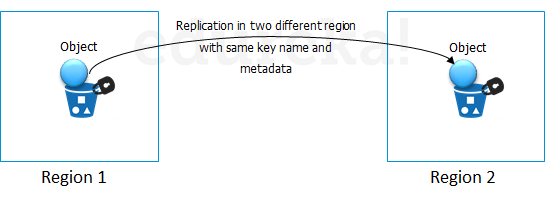
Now looking at the above parameters, we can clearly identify, that N Virginia will be the best region for this company because of the low latency and low price. Irrespective of your location, you can select any region which might suit your requirements, since you can access your S3 buckets from anywhere.

**Talking about regions, let’s see about the possibility of having a backup in some other availability region or you may want to move your data to some other region. Thankfully, this feature has been recently added to the AWS S3 system and is pretty easy to use.**

**Cross-region Replication**

As the name suggests, Cross-region Replicationenables user to either replicate or transfer data to some other location without any hassle.

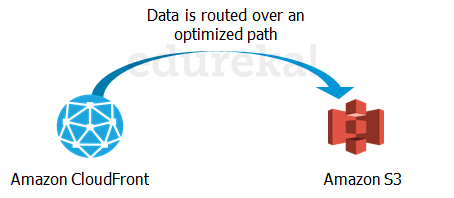
This obviously has a cost to it which has been discussed further in this article.



**How is the data transferred?**

Besides traditional transfer practices that is over the internet, AWS has 2 more ways to provide data transfer securely and at a faster rate:

* Transfer Acceleration
* Snowball

****

**Transfer Acceleration** enables fast, easy and secure transfers over long distances by exploiting Amazon’s CloudFront edge technology.

**CloudFront** is a caching service by AWS, in which the data from client site gets transferred to the nearest edge location and from there the data is routed to your AWS S3 bucket over an optimised network path.

The**Snowball** is a way of transferring your data physically. In this Amazon sends an equipment to your premises, on which you can load the data. It has a kindle attached to it which has your shipping address when it is shipped from Amazon. When data transfer is complete on the Snowball, kindle changes the shipping address back to the AWS headquarters where the Snowball has to be sent.

The Snowball is ideal for customers who have large batches of data move. The average turnaround time for Snowball is 5-7 days, in the same time Transfer Acceleration can transfer up to 75 TB of data on a dedicated 1Gbps line. So depending on the use case, a customer can decide.

Obviously, there will be some cost around it, let’s look at the overall costing around S3.

**Pricing**

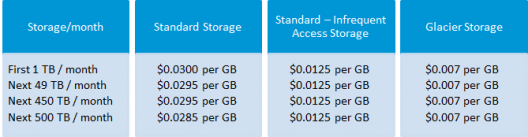
**“Isn’t anything free on AWS?”**

***Yes!***  As a part of the AWS Free Usage Tier, you can get started with AWS S3 for free. Upon sign up, new AWS customers receive 5 GB of Amazon S3 standard storage, 20,000 Get-Requests, 2,000 Put-Requests, and 15GB of data transfer-out each month for one year.

Over this limit, there is a cost attached, let’s understand how amazon charges you:

**How is S3 billed?**

Though having so many features, AWS S3 is affordable and flexible in its costing. It works on **Pay Per Use,** meaning, you only pay what you use. The table below is an example for pricing of S3 for a specific region:

****

**Source: aws.amazon.com for North Virginia region**

**Cross Region Replication** is billed in the following way:

If you replicate 1,000 1 GB objects (1,000 GB) between regions you will incur a request charge of $0.005 (1,000 requests x $0.005 per 1,000 requests) for replicating 1,000 objects and a charge of $20 ($0.020 per GB transferred x 1,000 GB) for inter-region data transfer. After replication, the 1,000 GB will incur storage charges based on the destination region.

**Snowball,** there are 2 variants:

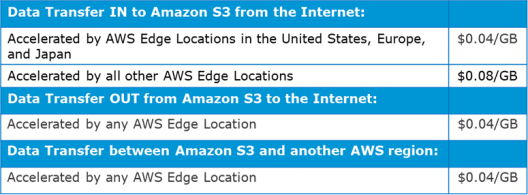
* Snowball 50 TB : 200$
* Snowball 80 TB:  250$

This is the fixed service fee that they charge.

Apart from this there are on-site, charges which are exclusive of shipping days, the shipping days are free.

The first 10 on-site days are also free, meaning when the Snowball reaches your premises from then, till the day it is shipped back, they are the on-site days. The day it arrives, and the day it is shipped gets counted as shipping days, therefore are free.

**Transfer Acceleration**pricing is shown in the following table:



**AWS S3 Use case: 1**

Industry “Media”

Let’s understand it through a real time use case to assimilate all what we have learnt so far:*IMDb Internet Movie Database* is a famous online database of information related to films, television programs and video games.

Let’s see how they exploit the AWS services:

* To get the lowest possible latency, all possible results for a search are pre-calculated with a document for every combination of letters in search. Each document is pushed to Amazon Simple Storage Service (S3) and thereby to **Amazon CloudFront**, putting the documents physically close to the users. The theoretical number of possible searches to calculate is mind-boggling—a 20-character search has 23 x 1030 combinations
* But in practice, using IMDb’s authority on movie and celebrity data can reduce the search space to about 150,000 documents, which Amazon S3 and **Amazon CloudFront** can distribute in just a few hours.

**AWS S3 Use case: 2**

**Project Statement –**Hosting a Static Website on Amazon S3

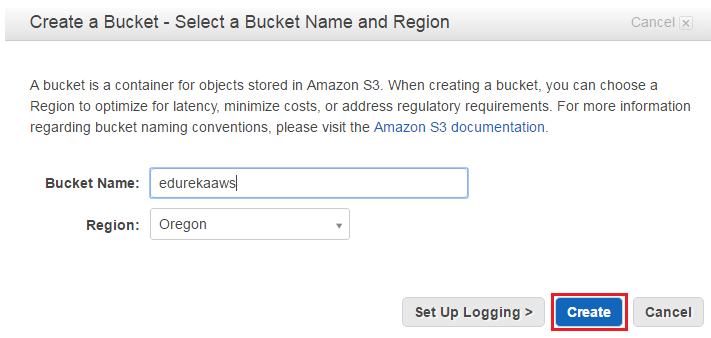
Let’s first understand: What is a static website?

In short, it’s a website comprised of only HTML, CSS, and/or JavaScript. That means server-side scripts aren’t supported, so if you want to host a Rails or PHP app, you’ll need to look elsewhere.

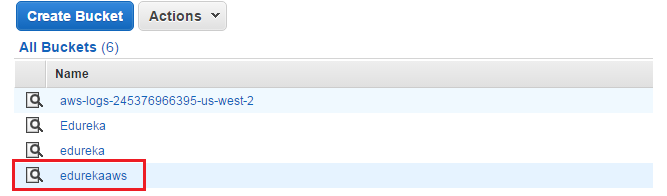
For simpler purposes, welcome to the wonderful world of hosting websites on AWS S3!

**Step 1: Create a bucket**

To create a bucket, navigate to S3 in the AWS Management Console and hit Create Bucket. You’ll be prompted to enter a name and a region.

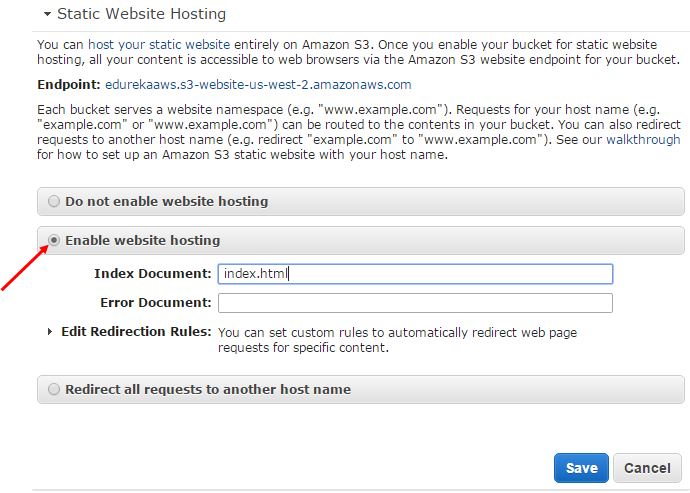
If you plan on using your own domain/sub-domain, use that for your bucket name. For the region, pick the one closest to you and hit Create. With any luck, you’ll see your new bucket appear in the console.

**Step 2: Verify the Created Bucket**



**Step3: Enable Website Hosting**

The only thing now left to do is, to enable Static Website Hosting. Just select it from the properties panel on the right.



**Step 4: Create a Html File**

Make sure you set the Index Document to index.html. You can also set an error page if you want. When you’re done, hit Save.

One nice thing about the AWS Management Console is that you can upload files to your bucket right from  your browser. Let’s start by creating one called **index.html**. This will be the contents of the home page:

<!doctype html>

<html>

<head>

<title>

Hello, S3!

</title>

<meta name="description" content="My first S3 website">

<meta charset="utf-8">

</head>

<body>

<h2>My first S3 website</h2>

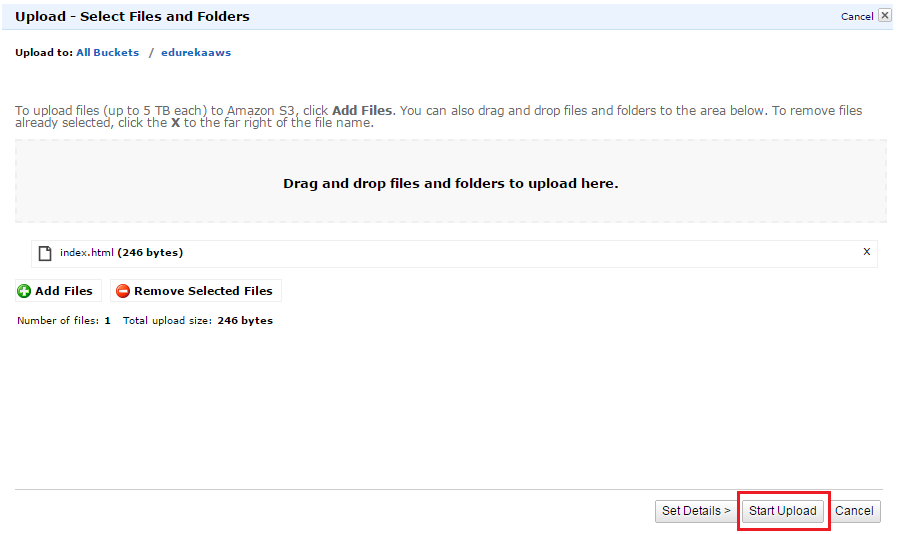
<p>I can't believe it was that easy!</p>

</body>

</html>

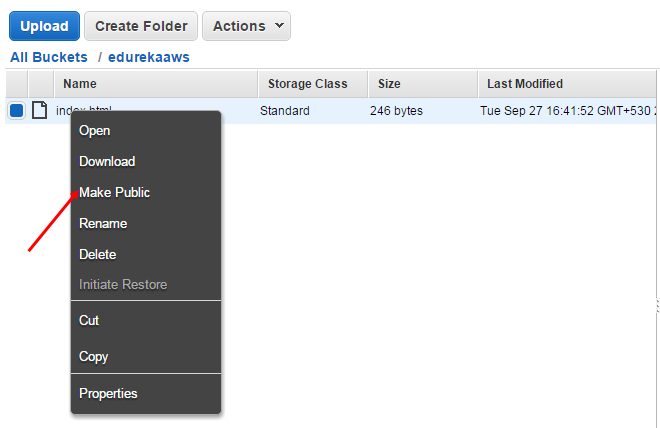
**Step 5: Upload the File in a Bucket**

To upload the file, select your new bucket and hit Start Upload button.

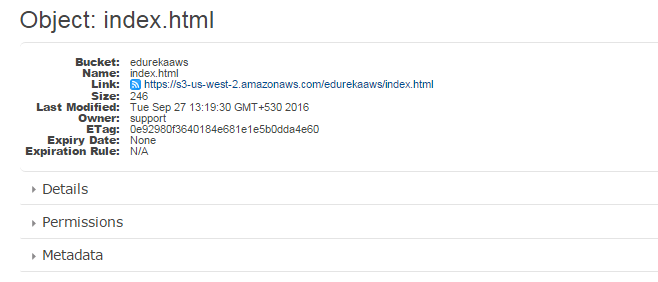
Once you’ve uploaded index.html, it will appear in your bucket. However, you won’t be able to see it in your browser yet because everything in AWS S3 is private by default.

**Step 6: Make the Html File Public**

i) To make index.html file public, right-click on index.html and select Make Public. (Remember to do this for any other files you upload to your website!)

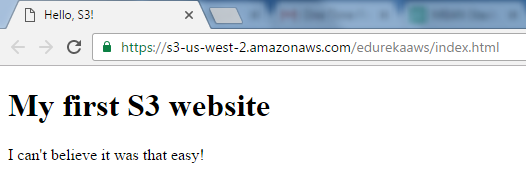


Now that your homepage is visible to the world, it’s time to test everything out!

ii) Now, select index.html in the console and go to the Properties tab.

**Step 7: Final Step to Verify the Result**

Clicking the link will take you to your new homepage.



**Congratulations!**You have just hosted a html website in AWS using S3.

AWS Simple Storage Service – S3 Overview

* Amazon S3 is a simple key, value object store designed for the Internet
* S3 provides unlimited storage space and works on the pay as you use model. Service rates gets cheaper as the usage volume increases
* S3 is an Object level storage (not a Block level storage) and cannot be used to host OS or dynamic websites.
* We can host [Hosting a Static Website on Amazon S3](https://docs.aws.amazon.com/AmazonS3/latest/dev/WebsiteHosting.html).
* S3 resources *for* ***e.g. buckets and objects*** are private by default

S3 Buckets & Objects

**Buckets**

* A bucket is a container for objects stored in S3 and help organize the S3 namespace.
* A bucket is owned by the AWS account that creates it and helps identify the account responsible for storage and data transfer charges. Bucket ownership is not transferable
* S3 bucket names are globally unique, regardless of the AWS region in which you create the bucket
* Even though S3 is a global service, buckets are created within a region specified during the creation of the bucket
* Every object is contained in a bucket
* There is no limit to the number of objects that can be stored in a bucket and no difference in performance whether you use many buckets to store your objects or a single bucket to store all your objects
* S3 data model is a flat structure i.e. there are no hierarchies or folders within the buckets. However, logical hierarchy can be inferred using the keyname prefix e.g. Folder1/Object1
* Restrictions
  + 100 buckets (soft limit) can be created in each of AWS account
  + Bucket names should be globally unique and DNS compliant
  + Bucket ownership is not transferable
  + Buckets cannot be nested and cannot have bucket within another bucket
* You can delete a empty or a non-empty bucket
* S3 allows retrieval of 1000 objects and provides pagination support

**Objects**

* Objects are the fundamental entities stored in S3 bucket
* Object is uniquely identified within a bucket by a keyname and version ID
* Objects consist of object data, metadata and others
  + **Key** is object name
  + **Value** is data portion is opaque to S3
  + **Metadata** is the data about the data and is a set of name-value pairs that describe the object *for e.g. content-type, size, last modified*. Custom metadata can also be specified at the time the object is stored.
  + **Version ID**is the version id for the object and in combination with the key helps to unique identify an object within a bucket
  + **Subresources** helps provide additional information for an object
  + **Access Control Information** helps control access to the objects
* Metadata for an object cannot be modified after the object is uploaded and it can be only modified by performing the copy operation and setting the metadata
* Objects belonging to a bucket reside in a specific AWS region never leave that region, unless explicitly copied using Cross Region replication
* Object can be retrieved as a whole or a partially
* With Versioning enabled, you can retrieve current as well as pervious versions of an object

Bucket & Object Operations

* Listing
  + S3 allows listing of all the keys within a bucket
  + A single listing request would return a max of 1000 object keys with pagination support using an indicator in the response to indicate if the response was truncated
  + Keys within a bucket can be listed using Prefix and Delimiter.
  + Prefix limits results to only those keys (kind of filtering) that begin with the specified prefix, and delimiter causes list to roll up all keys that share a common prefix into a single summary list result.
* Retrieval
  + Object can be retrieved as a whole
  + Object can be retrieved in parts or partially (specific range of bytes) by using the Range HTTP header.
  + Range HTTP header is helpful
    - if only partial object is needed for e.g. multiple files were uploaded as a single archive
    - for fault tolerant downloads where the network connectivity is poor
  + Objects can also be downloaded by sharing Pre-Signed urls
  + Metadata of the object is returned in the response headers
* Object Uploads
  + Single Operation – Objects of size 5GB can be uploaded in a single PUT operation
  + Multipart upload – can be used for objects of size > 5GB and supports max size of 5TB can is recommended for objects above size 100MB
  + Pre-Signed URLs can also be used shared for uploading objects
  + Uploading object if successful, can be verified if the request received a success response. Additionally, returned ETag can be compared to the calculated MD5 value of the upload object
* Copying Objects
  + Copying of object up to 5GB can be performed using a single operation and multipart upload can be used for uploads up to 5TB
  + When an object is copied
    - user-controlled system metadata *e.g. storage class* and user-defined metadata are also copied.
    - system controlled metadata *e.g. the creation date etc*is reset
  + Copying Objects can be needed
    - Create multiple object copies
    - Copy object across locations
    - Renaming of the objects
    - Change object metadata *for e.g. storage class, server-side encryption etc*
    - **Updating any metadata for an object requires all the metadata fields to be specified again**
* Deleting Objects
  + S3 allows deletion of a single object or multiple objects (max 1000) in a single call
  + For Non Versioned buckets,
    - the object key needs to be provided and object is permanently deleted
  + For Versioned buckets,
    - if an object key is provided, S3 inserts a delete marker and the previous current object becomes non current object
    - if an object key with a version ID is provided, the object is permanently deleted
    - if the version ID is of the delete marker, the delete marker is removed and the previous non current version becomes the current version object
  + Deletion can be MFA enabled for adding extra security
* Restoring Objects from Glacier
  + Objects must be restored before you can access an archived object
  + Restoration of an Object can take about 3 to 5 hours
  + Restoration request also needs to specify the number of days for which the object copy needs to be maintained.
  + During this period, the storage cost for both the archive and the copy is charged

# AWS Simple Storage Service – S3 Overview

* Amazon S3 is a simple key, value object store designed for the Internet
* S3 provides unlimited storage space and works on the pay as you use model. Service rates gets cheaper as the usage volume increases
* S3 is an Object level storage (not a Block level storage) and cannot be used to host OS or dynamic websites
* S3 resources for e.g. buckets and objects are private by default

## S3 Buckets & Objects

### ****Buckets****

* A bucket is a container for objects stored in S3 and help organize the S3 namespace.
* A bucket is owned by the AWS account that creates it and helps identify the account responsible for storage and data transfer charges. Bucket ownership is not transferable
* S3 bucket names are globally unique, regardless of the AWS region in which you create the bucket
* Even though S3 is a global service, buckets are created within a region specified during the creation of the bucket
* Every object is contained in a bucket
* There is no limit to the number of objects that can be stored in a bucket and no difference in performance whether you use many buckets to store your objects or a single bucket to store all your objects
* S3 data model is a flat structure i.e. there are no hierarchies or folders within the buckets. However, logical hierarchy can be inferred using the keyname prefix e.g. Folder1/Object1
* Restrictions
  + 100 buckets (soft limit) can be created in each of AWS account
  + Bucket names should be globally unique and DNS compliant
  + Bucket ownership is not transferable
  + Buckets cannot be nested and cannot have bucket within another bucket
* You can delete a empty or a non-empty bucket
* S3 allows retrieval of 1000 objects and provides pagination support

### ****Objects****

* Objects are the fundamental entities stored in S3 bucket
* Object is uniquely identified within a bucket by a keyname and version ID
* Objects consist of object data, metadata and others
  + **Key** is object name
  + **Value** is data portion is opaque to S3
  + **Metadata** is the data about the data and is a set of name-value pairs that describe the object for e.g. content-type, size, last modified. Custom metadata can also be specified at the time the object is stored.
  + **Version ID**is the version id for the object and in combination with the key helps to unique identify an object within a bucket
  + **Subresources** helps provide additional information for an object
  + **Access Control Information** helps control access to the objects
* Metadata for an object cannot be modified after the object is uploaded and it can be only modified by performing the copy operation and setting the metadata
* Objects belonging to a bucket reside in a specific AWS region never leave that region, unless explicitly copied using Cross Region replication
* Object can be retrieved as a whole or a partially
* With Versioning enabled, you can retrieve current as well as pervious versions of an object

## Bucket & Object Operations

* Listing
  + S3 allows listing of all the keys within a bucket
  + A single listing request would return a max of 1000 object keys with pagination support using an indicator in the response to indicate if the response was truncated
  + Keys within a bucket can be listed using Prefix and Delimiter.
  + Prefix limits results to only those keys (kind of filtering) that begin with the specified prefix, and delimiter causes list to roll up all keys that share a common prefix into a single summary list result.
* Retrieval
  + Object can be retrieved as a whole
  + Object can be retrieved in parts or partially (specific range of bytes) by using the Range HTTP header.
  + Range HTTP header is helpful
    - if only partial object is needed for e.g. multiple files were uploaded as a single archive
    - for fault tolerant downloads where the network connectivity is poor
  + Objects can also be downloaded by sharing Pre-Signed urls
  + Metadata of the object is returned in the response headers
* Object Uploads
  + Single Operation – Objects of size 5GB can be uploaded in a single PUT operation
  + Multipart upload – can be used for objects of size > 5GB and supports max size of 5TB can is recommended for objects above size 100MB
  + Pre-Signed URLs can also be used shared for uploading objects
  + Uploading object if successful, can be verified if the request received a success response. Additionally, returned ETag can be compared to the calculated MD5 value of the upload object
* Copying Objects
  + Copying of object up to 5GB can be performed using a single operation and multipart upload can be used for uploads up to 5TB
  + When an object is copied
    - user-controlled system metadata e.g. storage class and user-defined metadata are also copied.
    - system controlled metadata e.g. the creation date etc is reset
  + Copying Objects can be needed
    - Create multiple object copies
    - Copy object across locations
    - Renaming of the objects
    - Change object metadata for e.g. storage class, server-side encryption etc
    - **Updating any metadata for an object requires all the metadata fields to be specified again**
* Deleting Objects
  + S3 allows deletion of a single object or multiple objects (max 1000) in a single call
  + For Non Versioned buckets,
    - the object key needs to be provided and object is permanently deleted
  + For Versioned buckets,
    - if an object key is provided, S3 inserts a delete marker and the previous current object becomes non current object
    - if an object key with a version ID is provided, the object is permanently deleted
    - if the version ID is of the delete marker, the delete marker is removed and the previous non current version becomes the current version object
  + Deletion can be MFA enabled for adding extra security
* Restoring Objects from Glacier
  + Objects must be restored before you can access an archived object
  + Restoration of an Object can take about 3 to 5 hours
  + Restoration request also needs to specify the number of days for which the object copy needs to be maintained.
  + During this period, the storage cost for both the archive and the copy is charged

### ****Pre-Signed URLs****

* All buckets and objects are by default private
* Pre-signed URLs allows user to be able download or upload a specific object without requiring AWS security credentials or permissions
* Pre-signed URL allows anyone access to the object identified in the URL, provided the creator of the URL has permissions to access that object
* Creation of the pre-signed urls requires the creator to provide his security credentials, specify a bucket name, an object key, an HTTP method (GET for download object & PUT of uploading objects), and expiration date and time
* Pre-signed urls are valid only till the expiration date & time

### Multipart Upload

* Multipart upload allows the user to upload a single object as a set of parts. Each part is a contiguous portion of the object’s data.
* Multipart uploads supports 1 to 10000 parts and each Part can be from 5MB to 5GB with last part size allowed to be less than 5MB
* Multipart uploads allows max upload size of 5TB (10000 parts \* 5GB/part theoretically)
* Object parts can be uploaded independently and in any order. If transmission of any part fails, it can be retransmitted without affecting other parts.
* After all parts of the object are uploaded and complete initiated, S3 assembles these parts and creates the object.
* Using multipart upload provides the following advantages:
  + Improved throughput – parallel upload of parts to improve throughput
  + Quick recovery from any network issues – Smaller part size minimizes the impact of restarting a failed upload due to a network error.
  + Pause and resume object uploads – Object parts can be uploaded over time. Once a multipart upload is initiated there is no expiry; you must explicitly complete or abort the multipart upload.
  + Begin an upload before the final object size is known – an object can be uploaded as is it being created
* Three Step process
  + **Multipart Upload Initiation**
    - Initiation of a Multipart upload request to S3 returns a unique ID for each multipart upload.
    - This ID needs to be provided for each part uploads, completion or abort request and listing of parts call.
    - All the Object metadata required needs to be provided during the Initiation call
  + **Parts Upload**
    - Parts upload of objects can be performed using the unique upload ID
    - A part number (between 1 – 10000) needs to be specified with each request which identifies each part and its position in the object
    - If a part with the same part number is uploaded, the previous part would be overwritten
    - After the part upload is successful, S3 returns an ETag header in the response which must be recorded along with the part number to be provided during the multipart completion request
  + **Multipart Upload Completion or Abort**
    - On Multipart Upload Completion request, S3 creates an object by concatenating the parts in ascending order based on the part number and associates the metadata with the object
    - Multipart Upload Completion request should include the unique upload ID with all the parts and the ETag information
    - S3 response includes an ETag that uniquely identifies the combined object data
    - On Multipart upload Abort request, the upload is aborted and all parts are removed. Any new part upload would fail. However, any in progress part upload is completed and hence and abort request must be sent after all the parts upload have been completed
    - S3 should receive a multipart upload completion or abort request else it will not delete the parts and storage would be charged

## Virtual Hosted Style vs Path-Style Request

S3 allows the buckets and objects to be referred in Path-style or Virtual hosted-style URLs

### Path-style

* Bucket name is not part of the domain (unless you use a region specific endpoint)
* the endpoint used must match the region in which the bucket resides
* for e.g, if you have a bucket called *mybucket*that resides in the EU (Ireland) region with object named puppy.jpg, the correct path-style syntax URI is http://s3-eu-west-1.amazonaws.com/mybucket/puppy.jpg*.*
* A “PermanentRedirect” error is received with an HTTP response code 301, and a message indicating what the correct URI is for the resource if a bucket is accessed outside the US East (N. Virginia) region with path-style syntax that uses either of the following:
  + http://s3.amazonaws.com
  + An endpoint for a region different from the one where the bucket resides. For example, if you use http://s3-eu-west-1.amazonaws.com for a bucket that was created in the US West (N. California) region

### Virtual hosted-style

* S3 supports virtual hosted-style and path-style access in all regions.
* In a virtual-hosted-style URL, the bucket name is part of the domain name in the URL
* for e.g. http://bucketname.s3.amazonaws.com/objectname
* S3 virtual hosting can be used to address a bucket in a REST API call by using the HTTP Host header
* Benefits
  + attractiveness of customized URLs,
  + provides an ability to publish to the “root directory” of the bucket’s virtual server. This ability can be important because many existing applications search for files in this standard location.
* S3 updates DNS to reroute the request to the correct location when a bucket is created in any region, which might take time.
* S3 routes any virtual hosted-style requests to the US East (N.Virginia) region, by default, if the US East (N. Virginia) endpoint s3.amazonaws.com is used, instead of the region-specific endpoint (for example, s3-eu-west-1.amazonaws.com) and S3 redirects it with HTTP 307 redirect to the correct region.
* When using virtual hosted-style buckets with SSL, the SSL wild card certificate only matches buckets that do not contain periods.To work around this, use HTTP or write your own certificate verification logic.
* If you make a request to the http://bucket.s3.amazonaws.com endpoint, the DNS has sufficient information to route your request directly to the region where your bucket resides.

## S3 Pricing

* Amazon S3 costs vary by Region
* Charges in S3 are incurred for
  + Storage – cost is per GB/month
  + Requests – per request cost varies depending on the request type GET, PUT
  + Data Transfer
    - data transfer in is free
    - data transfer out is charged per GB/month (except in the same region or to Amazon CloudFront).

------------------------------------------------------------------------------------------------------------------

# AWS S3 Data Consistency Model

* S3 achieves high availability by replicating data across multiple servers within Amazon’s data centers.
* S3 provides **read-after-write consistency for PUTS of new objects**
  + For a PUT request, S3 synchronously stores data across multiple facilities before returning SUCCESS
  + A process writes a new object to S3 and will be immediately able to read the Object
  + A process writes a new object to S3 and immediately lists keys within its bucket. Until the change is fully propagated, the object might not appear in the list.
* S3 provides**eventual consistency for overwrite PUTS and DELETES** in all regions.
  + For updates and deletes to Objects, the changes are eventually reflected and not available immediately
  + if a process replaces an existing object and immediately attempts to read it. Until the change is fully propagated, S3 might return the prior data
  + if a process deletes an existing object and immediately attempts to read it. Until the deletion is fully propagated, S3 might return the deleted data
  + if a process deletes an existing object and immediately lists keys within its bucket. Until the deletion is fully propagated, S3 might list the deleted object.
* Updates to a single key are atomic. for e.g., if you PUT to an existing key, a subsequent read might return the old data or the updated data, but it will never write corrupted or partial data.
* S3 does not currently support object locking. for e.g. If two PUT requests are simultaneously made to the same key, the request with the latest time stamp wins. If this is an issue, you will need to build an object-locking mechanism into your application.
* Updates are key-based; there is no way to make atomic updates across keys. for e.g, you cannot make the update of one key dependent on the update of another key unless you design this functionality into your application.
* -------------------------------------------------------------------------------------------------------------------------------------------

# AWS S3 Subresources

* Amazon S3 Subresources provides support to store, and manage the bucket configuration information
* S3 subresources only exist in the context of a specific bucket or object
* S3 defines a set of subresources associated with buckets and objects.
* S3 Subresources are subordinates to objects; that is, they do not exist on their own, they are always associated with some other entity, such as an object or a bucket
* S3 supports various options to configure a bucket for e.g., bucket can be configured for website hosting, configuration added to manage lifecycle of objects in the bucket, and to log all access to the bucket.

## Bucket Subresources

### Lifecycle

Refer to My Blog Post about [S3 Object Lifecycle Management](http://jayendrapatil.com/aws-s3-object-lifecycle-management/)

### Static Website hosting

* S3 can be used for Static Website hosting with Client side scripts.
* S3 does not support server-side scripting
* S3, in conjunction with Route 53, supports hosting a website at the root domain which can point to the S3 website endpoint
* S3 website endpoints do not support https.
* For S3 website hosting the content should be made publicly readable which can be provided using a bucket policy or an ACL on an object
* User can configure the index, error document as well as configure the conditional routing of on object name
* Bucket policy applies only to objects owned by the bucket owner. If your bucket contains objects not owned by the bucket owner, then public READ permission on those objects should be granted using the object ACL.
* Requester Pays buckets or DevPay buckets do not allow access through the website endpoint. Any request to such a bucket will receive a 403 -Access Denied response

### Versioning

Refer to My Blog Post about [S3 Object Versioning](http://jayendrapatil.com/aws-s3-object-versioning/)

### Policy & Access Control List (ACL)

Refer to My Blog Post about [S3 Permissions](http://jayendrapatil.com/aws-s3-permisions/)

### CORS (Cross Origin Resource Sharing)

* All browsers implement the Same-Origin policy, for security reasons, where the web page from an domain can only request resources from the same domain.
* CORS allow client web applications loaded in one domain access to the restricted resources to be requested from another domain
* With CORS support in S3, you can selectively allow cross-origin access to your S3 resources
* CORS configuration rules identify the origins allowed to access the bucket, the operations (HTTP methods) that would be supported for each origin, and other operation-specific information

### Logging

* Logging, disabled by default, enables tracking access requests to S3 bucket
* Each access log record provides details about a single access request, such as the requester, bucket name, request time, request action, response status, and error code, if any.
* Access log information can be useful in security and access audits and also help learn about the customer base and understand the S3 bill
* S3 periodically collects access log records, consolidates the records in log files, and then uploads log files to a target bucket as log objects.
* If logging is enabled on multiple source buckets with same target bucket, the target bucket will have access logs for all those source buckets, but each log object will report access log records for a specific source bucket

### Tagging

* S3 provides the tagging subresource to store and manage tags on a bucket
* Cost allocation tags can be added to the bucket to categorize and track AWS costs
* AWS can generate a cost allocation report with usage and costs aggregated by the tags applied to the buckets

### Location

* When you create a bucket, AWS region needs to be specified where the S3 bucket will be created
* S3 stores this information in the location subresource and provides an API for retrieving this information

### Notification

* S3 notification feature enables notifications to be triggered when certain events happen in your bucket
* Notifications are enabled at Bucket level
* Notifications can be configured to be filtered by the prefix and suffix of the key name of objects. However, filtering rules cannot be defined with  overlapping prefixes, overlapping suffixes, or prefix and suffix overlapping
* S3 can publish the following events
  + New Objects created event
    - Can be enabled for PUT, POST or COPY operations
    - You will not receive event notifications from failed operations
  + Object Removal event
    - Can public delete events for object deletion, version object deletion or insertion of delete marker
    - You will not receive event notifications from automatic deletes from lifecycle policies or from failed operations.
  + Reduced Redundancy Storage (RRS) object lost event
    - Can be used to reproduce/recreate the Object
* S3 can publish events to the following destination
  + Amazon SNS topic
  + Amazon SQS queue
  + AWS Lambda
* For S3 to be able to publish events to the destination, S3 principal should be granted necessary permissions

### Cross Region Replication

* Cross-region replication is a bucket-level feature that enables automatic, asynchronous copying of objects across buckets in different AWS regions
* S3 can replicate all or a subset of objects with specific key name prefixes
* S3 encrypts all data in transit across AWS regions using SSL
* Object replicas in the destination bucket are exact replicas of the objects in the source bucket with the same key names and the same metadata.
* Cross Region Replication can be useful for the following scenarios :-
  + **Compliance requirement** to have data backed up across regions
  + **Minimize latency** to allow users across geography to access objects
  + **Operational reasons** compute clusters in two different regions that analyze the same set of objects
* Requirements
  + source and destination buckets must be versioning-enabled
  + source and destination buckets must be in different AWS regions
  + objects can be replicated from a source bucket to only one destination bucket
  + S3 must have permission to replicate objects from that source bucket to the destination bucket on your behalf.
  + If the source bucket owner also owns the object, the bucket owner has full permissions to replicate the object. If not, the source bucket owner must have permission for the S3 actions s3:GetObjectVersion and s3:GetObjectVersionACL to read the object and object ACL
  + If you are setting up cross-region replication in a cross-account scenario (where the source and destination buckets are owned by different AWS accounts), the source bucket owner must have permission to replicate objects in the destination bucket.
* Replicated & Not Replicated
  + Any new objects created after you add a replication configuration **are**replicated
  + S3 **does not** retroactively replicate objects that existed before you added replication configuration.
  + Only Objects created with SSE-S3 **are** replicated using server-side encryption using the Amazon S3-managed encryption key.
  + Objects created with server-side encryption using either customer-provided (SSE-C) or AWS KMS–managed encryption (SSE-KMS) keys **are not replicated**
  + S3 replicates **only** objects in the source bucket for which the bucket owner has permission to read objects and read ACLs
  + S3 **does not** replicate objects in the source bucket for which the bucket owner does not have permissions.
  + Any object ACL updates **are** replicated, although there can be some delay before Amazon S3 can bring the two in sync. This applies only to objects created after you add a replication configuration to the bucket.
  + Updates to bucket-level S3 subresources **are not** replicated, allowing different bucket configurations on the source and destination buckets
  + Only customer actions **are** replicated & actions performed by lifecycle configuration **are not** replicated
  + Objects in the source bucket that are replicas, created by another cross-region replication, **are not** replicated.

### Requester Pays

* By default, buckets are owned by the AWS account that created it (the bucket owner) and the AWS account pays for storage costs, downloads and data transfer charges associated with the bucket.
* Using Requester Pays subresource :-
  + Bucket owner specifies that the requester requesting the download will be charged for the download
  + However, the bucket owner still pays the storage costs
* Enabling Requester Pays on a bucket
  + disables anonymous access to that bucket
  + does not support BitTorrent
  + does not support SOAP requests
  + cannot be enabled for end user logging bucket

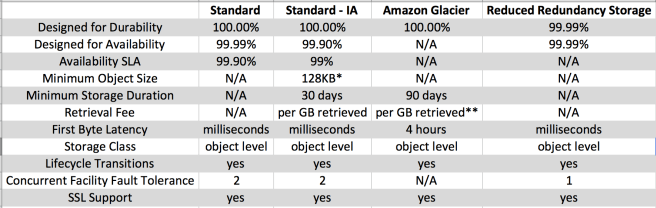
## Object Subresources

### Torrent

* Default distribution mechanism for S3 data is via client/server download
* Bucket owner bears the cost of Storage as well as the request and transfer charges which can increase linearly for an popular object
* S3 also supports the BitTorrent protocol
  + BitTorrent is an open source Internet distribution protocol
  + BitTorrent addresses this problem by recruiting the very clients that are downloading the object as distributors themselves
  + S3 bandwidth rates are inexpensive, but BitTorrent allows developers to further save on bandwidth costs for a popular piece of data by letting users download from Amazon and other users simultaneously
* Benefit for publisher is that for large, popular files the amount of data actually supplied by S3 can be substantially lower than what it would have been serving the same clients via client/server download
* Any object in S3 that is publicly available and can be read anonymously can be downloaded via BitTorrent
* torrent file can be retrieved for any publicly available object by simply adding a “?torrent” query string parameter at the end of the REST GET request for the object
* Generating the .torrent for an object takes time proportional to the size of that object, so its recommended to make a first torrent request yourself to generate the file so that subsequent requests are faster
* Torrent are enabled only for objects that are less than 5 GB in size.
* Torrent subresource can only be retrieve, and cannot be created, updated or deleted

# AWS S3 Storage Classes Overview

* Amazon S3 storage classes are designed to sustain the concurrent loss of data in one or two facilities
* S3 storage classes allows lifecycle management for automatic migration of objects for cost savings
* S3 storage classes support SSL encryption of data in transit and data encryption at rest
* S3 also regularly verifies the integrity of your data using checksums and provides auto healing capability



## Standard

* Storage class is ideal for performance-sensitive use cases and frequently accessed data and is designed to sustain the loss of data in a two facilities
* STANDARD is the default storage class, if none specified during upload
* Low latency and high throughput performance
* Designed for durability of 99.999999999% of objects
* Designed for 99.99% availability over a given year
* Backed with the [Amazon S3 Service Level Agreement](https://aws.amazon.com/s3/sla/) for availability.

## Standard IA

* S3 STANDARD\_IA (Infrequent Access) storage class is optimized for long-lived and less frequently accessed data for e.g. backups and older data where access is limited, but the use case still demands high performance
* STANDARD\_IA is designed to sustain the loss of data in a two facilities
* STANDARD\_IA objects are available for real-time access.
* STANDARD\_IA storage class is suitable for larger objects greater than 128 KB (smaller objects are charged for 128KB only) kept for at least 30 days.
* Same low latency and high throughput performance of Standard
* Designed for durability of 99.999999999% of objects
* Designed for 99.9% availability over a given year
* Backed with the [Amazon S3 Service Level Agreement](https://aws.amazon.com/s3/sla/) for availability

## Reduced Redundancy Storage – RRS

* Reduced Redundancy Storage (RRS) storage class is designed for noncritical, reproducible data stored at lower levels of redundancy than the STANDARD storage class, which reduces storage costs
* Designed for durability of 99.99% of objects
* Designed for 99.99% availability over a given year
* Lower level of redundancy results in less durability and availability
* RRS stores objects on multiple devices across multiple facilities, providing 400 times the durability of a typical disk drive,
* RRS does not replicate objects as many times as S3 standard storage and is designed to sustain the loss of data in a single facility.
* If an RRS object is lost, S3 returns a 405 error on requests made to that object
* S3 can send an event notification, configured on the bucket, to alert a user or start a workflow when it detects that an RRS object is lost which can be used to replace the lost object

## Glacier

* GLACIER storage class is suitable for archiving data where data access is infrequent and retrieval time of several (3-5) hours  is acceptable.
* GLACIER storage class uses the very low-cost Amazon Glacier storage service, but the objects in this storage class are still managed through S3
* Designed for durability of 99.999999999% of objects
* GLACIER cannot be specified as the storage class at the object creation time but has to be transitioned fromSTANDARD, RRS, or STANDARD\_IA to GLACIER storage class using lifecycle management.
* For accessing GLACIER objects,
  + object must be restored which can taken anywhere between 3-5 hours
  + objects are only available for the time period (number of days) specified during the restoration request
  + object’s storage class remains GLACIER
  + charges are levied for both the archive (GLACIER rate) and the copy restored temporarily (RRS rate)
* Vault Lock feature enforces compliance via a lockable policy.
* --------------------------------------------------------------------------------------------------------------------------------------------

# S3 Object Versioning

* S3 Object Versioning can be used to protect from unintended overwrites and deletions
* Versioning helps to keep multiple variants of an object in the same bucket and can be used to preserve, retrieve, and restore every version of every object stored in your Amazon S3 bucket.
* As Versioning maintains multiple copies of the same objects as whole and you accrue charges for multiple versions for e.g. for a 1GB file with 5 copies with minor differences would consume 5GB of S3 storage space and you would be charged for the same.
* Versioning is not enabled by default and has to be explicitly enabled for each bucket
* Versioning once enabled, cannot be disabled and can only be suspended
* Versioning enabled on a bucket applies to all the objects within the bucket
* Permissions are set at the version level. Each version has its own object owner; an AWS account that creates the object version is the owner. So, you can set different permissions for different versions of the same object.
* Irrespective of the Versioning, each object in the bucket has a version.
  + For Non Versioned bucket, the version ID for each object is null
  + For Versioned buckets, a unique version ID is assigned to each object
* With Versioning, version ID forms a key element to define uniqueness of an object within an bucket along with the bucket name and object key
* Object Retrieval
  + For Non Versioned bucket
    - An Object retrieval always return the only object available
  + For Versional bucket
    - An object retrieval returns the Current object.
    - Non Current object can be retrieved by specifying the version ID.
* Object Addition
  + For Non Versioned bucket
    - If an object with the same key is uploaded again it overwrites the object
  + For Versioned bucket
    - If an object with the same key is uploaded the new uploaded object becomes the Current version and the previous object becomes the Non current version.
    - A non current versioned object can be retrieved and restored hence protecting against **accidental overwrites**
* When an object in a bucket is deleted
  + For Non Versioned bucket
    - An object is permanently deleted and cannot be recovered
  + For Versioned bucket,
    - All versions remain in the bucket and Amazon inserts a delete marker which becomes the Current version
    - A non current versioned object can be retrieved and restored hence protecting against **accidental overwrites**
    - If a Object with a specific version ID is deleted, a permanent deletion happens and the object cannot be recovered
* Delete marker
  + Delete Marker object does not have any data or acl associated with it, just the key and the version ID
  + An object retrieval on a bucket with delete marker as the Current version would return a 404
  + Only a DELETE operation is allowed on the Delete Marker object
  + If the Delete marker object is deleted by specifying its version ID, the previous non current version object becomes the current version object
  + If a DELETE request is fired on the Bucket with Delete Marker as the current version, the Delete marker object is not deleted but an Delete Marker is added again
* Restoring Previous Versions
  + Copy a previous version of the object into the same bucket. Copied object becomes the current version of that object and all object versions are preserved – **Recommended** as you still keep all the versions
  + Permanently delete the current version of the object. When you delete the current object version, you, in effect, turn the previous version into the current version of that object.
* Versioning Suspended Bucket
  + Versioning can be suspended to stop accruing new versions of the same object in a bucket
  + Existing objects in your bucket do not change and only future requests behavior changes
  + For each new object addition, a object with version ID null is added.
  + For each object addition with the same key name, the object with the version ID null is overwritten
  + An object retrieval request will always return the current version of the object
  + A DELETE request on the bucket, would permanently delete the version ID null object and inserts a Delete Marker
  + A DELETE request does not delete anything if the bucket does not have an object with version ID null
  + A DELETE request can still be fired with a specific version ID for any previous object with version IDs stored
* MFA Delete
  + Additional security can be enabled by configuring a bucket to enable MFA (Multi-Factor Authentication) delete
  + MFA Delete can be enabled on a bucket to ensure that data in your bucket cannot be **accidentally deleted**
  + While the bucket owner, the AWS account that created the bucket (root account), and all authorized IAM users can enable versioning, but only the **bucket owner (root account) can enable MFA delete**.

# S3 Object Lifecycle Overview

* S3 Object lifecycle can be managed by using a lifecycle configuration, which defines how S3 manages objects during their lifetime.
* Lifecycle configuration enables simplification of object lifecycle management, for e.g. moving of less frequently access objects, backup or archival of data for several years or permanent deletion of objects, all transitions can be controlled automatically
* 1000 lifecycle rules can be configured per bucket
* S3 Object Lifecycle Management rules applied to an bucket are applicable to all the existing objects in the bucket as well as the ones that will be added anew
* S3 Object lifecycle management allows 2 types of behavior
  + **Transition** in which the storage class for the objects change
  + **Expiration** where the objects are permanently deleted
* Lifecycle Management can be configured with [Versioning](http://jayendrapatil.com/aws-s3-object-versioning/), which allows storage of one current object version and zero or more non current object versions
* Object’s lifecycle management applies to both Non Versioning and Versioning enabled buckets
* For Non Versioned buckets
  + Transitioning period is considered from the object’s creation date
* For Versioned buckets,
  + Transitioning period for current object is calculated for the object creation date
  + Transitioning period for non current object is calculated for the date when the object became a noncurrent versioned object
  + S3 uses the number of days since its successor was created as the number of days an object is noncurrent.
* S3 calculates the time by adding the number of days specified in the rule to the object creation time and rounding the resulting time to the next day midnight UTC. For e.g., if an object was created at 15/1/2016 10:30 AM UTC and you specify 3 days in a transition rule, which results in 18/1/2016 10:30 AM UTC and rounded of to next day midnight time 19/1/2016 00:00 UTC.
* Lifecycle configuration on MFA-enabled buckets is not supported.

## S3 Object Lifecycle Management Rules

1. STANDARD or REDUCED\_REDUNDANCY -> (128 KB & 30 days) -> STANDARD\_IA
   * Only objects with size more than 128 KB can be transitioned, as cost benefits for transitioning to STANDARD\_IA can be realized only for larger objects
   * Objects must be stored for at least 30 days in the current storage class before being transitioned to the STANDARD\_IA, as younger objects are accessed more frequently or deleted sooner than is suitable for STANDARD\_IA
2. STANDARD\_IA -> **X** -> STANDARD or REDUCED\_REDUNDANCY
   * Cannot transition from STANDARD\_IA to STANDARD or REDUCED\_REDUNDANCY
3. STANDARD or REDUCED\_REDUNDANCY or STANDARD\_IA -> GLACIER
   * Any Storage class can be transitioned to GLACIER
4. STANDARD or REDUCED\_REDUNDANCY -> (1 day) -> GLACIER
   * Transitioning from Standard or RRS to Glacier can be done in a day
5. STANDARD\_IA -> (30 days) -> GLACIER
   * Transitioning from Standard IA to Glacier can be done only after 30 days or 60 days from the object creation date or non current version date
6. GLACIER-> **X** -> STANDARD or REDUCED\_REDUNDANCY or STANDARD\_IA
   * Transition of objects to the GLACIER storage class is one-way
   * Cannot transition from GLACIER to any other storage class.
7. GLACIER -> (90 days) -> Permanent Deletion
   * Deleting data that is archived to Glacier is free, if the objects you delete are archived for three months or longer.
   * Amazon S3 charges a prorated early deletion fee, if the object is deleted or overwritten within three months of archiving it.
8. STANDARD or STANDARD\_IA or GLACIER -> **X**-> REDUCED\_REDUNDANCY
   * Cannot transition from any storage class to REDUCED\_REDUNDANCY.
9. Archival of objects to Amazon Glacier by using object lifecycle management is performed asynchronously and there may be a delay between the transition date in the lifecycle configuration rule and the date of the physical transition. However, AWS charges Amazon Glacier prices based on the transition date specified in the rule
10. For a versioning-enabled bucket
    * Transition and Expiration actions apply to current versions.
    * NoncurrentVersionTransition and NoncurrentVersionExpiration actions apply to noncurrent versions and works similar to the non versioned objects except the time period is from the time the objects became noncurrent
11. Expiration Rules
    * For Non Versioned bucket
      + Object is permanently deleted
    * For Versioned bucket
      + Expiration is applicable to the Current object only and does not impact any of the non current objects
      + S3 will insert a Delete Marker object with unique id and the previous current object becomes a non current version
      + S3 will not take any action if the Current object is a Delete Marker
      + If the bucket has a single object which is the Delete Marker (referred to as expired object delete marker), S3 removes the Delete Marker
    * For Versioned Suspended bucket
      + S3 will insert a Delete Marker object with version ID null and overwrite the any object with version ID null
12. When an object reaches the end of its lifetime, Amazon S3 queues it for removal and removes it asynchronously. There may be a delay between the expiration date and the date at which S3 removes an object.You are not charged for storage time associated with an object that has expired.
13. There are additional cost considerations if you put lifecycle policy to expire objects that have been in STANDARD\_IA for less than 30 days, or GLACIER for less than 90 days.

# S3 Permissions Overview

* By default, all S3 buckets, objects and related subresources are private
* User is the AWS Account or the IAM user who access the resource
* Bucket owner is the AWS account that created a bucket
* Object owner is the AWS account that uploads the object to a bucket, not owned by the account
* Only the Resource owner, the AWS account that creates the resource, can access the resource
* Resource owner can be
  + AWS account that creates the bucket or object owns those resources
  + If an IAM user creates the bucket or object, the AWS account of the IAM user owns the resource
  + If the bucket owner grants cross-account permissions to other AWS account users to upload objects to the buckets, the objects are owned by the AWS account of the user who uploaded the object and not the bucket owner except for the following conditions
    - Bucket owner can deny access to the object, as its still the bucket owner who pays for the object
    - Bucket owner can delete or apply archival rules to the object and perform restoration

## S3 Permissions Classification

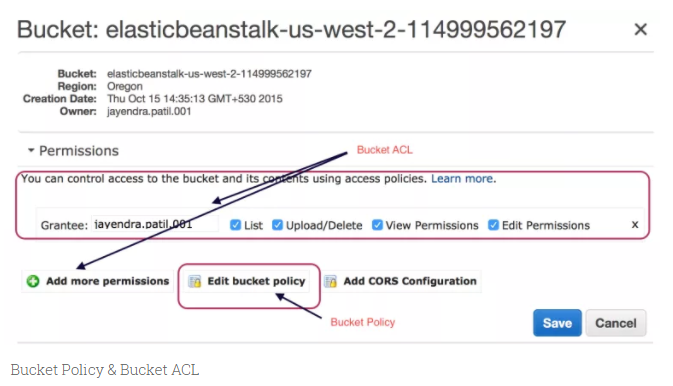
S3 permissions are classified into Resource based policies and User policies

### User policies

* User based policies use IAM with S3 to control the type of access a user or group of users has to specific parts of an S3 bucket the AWS account owns
* User based policy is always attached to an User, Group or a Role, anonymous permissions cannot be granted
* If an AWS account that owns a bucket wants to grant permission to users in its account, it can use either a bucket policy or a user policy

### Resource based policies

Bucket policies and access control lists (ACLs) are resource-based because they are attached to the Amazon S3 resources

Bucket Policy & Bucket ACL

#### **Bucket Policies**

* Bucket policy can be used to grant cross-account access to **other AWS accounts or IAM users in other accounts** for the bucket and objects in it.
* Bucket policies provide centralized, access control to buckets and objects based on a variety of conditions, including S3 operations, requesters, resources, and aspects of the request (e.g. IP address)
* If an AWS account that owns a bucket wants to grant permission to users in its account, it can use either a bucket policy or a user policy
* Permissions attached to a bucket apply to all of the objects in that bucket created and owned by the bucket owner
* Policies can either add or deny permissions across all (or a subset) of objects within a bucket
* Only the bucket owner is allowed to associate a policy with a bucket

#### **Access Control Lists (ACLs)**

* Each bucket and object has an ACL associated with it.
* An ACL is a list of grants identifying grantee and permission granted
* ACLs are used to grant basic read/write permissions on resources to **other AWS accounts**.
* ACL supports limited permissions set and
  + cannot grant conditional permissions, nor can you explicitly deny permissions
  + cannot be used to grant permissions for bucket subresources
* Permission can be granted to an AWS account by the email address or the canonical user ID (is just an obfuscated Account Id). If an email address is provided, S3 will still find the canonical user ID for the user and add it to the ACL.
* It is Recommended to use Canonical user ID as email address would not be supported
* **Bucket ACL**
  + Only recommended use case for the bucket ACL is to grant write permission to S3 Log Delivery group to write access log objects to the bucket
  + Bucket ACL will help grant write permission on the bucket to the Log Delivery group if access log delivery is needed to your bucket
  + **Only way you can grant necessary permissions to the Log Delivery group is via a bucket ACL**
* **Object ACL**
  + Object ACLs control only Object-level Permissions
  + Object ACL is the **only way to manage permission to an object in the bucket not owned by the bucket owner** i.e. If the bucket owner allows cross-account object uploads and if the object owner is different from the bucket owner, the only way for the object owner to grant permissions on the object is through Object ACL
  + If the Bucket and Object is owned by the same AWS account, Bucket policy can be used to manage the permissions
  + If the Object and User is owned by the same AWS account, User policy can be used to manage the permissions

### Amazon S3 Request Authorization

When Amazon S3 receives a request, it must evaluate all the user policies, bucket policies and acls to determine whether to authorize or deny the request.

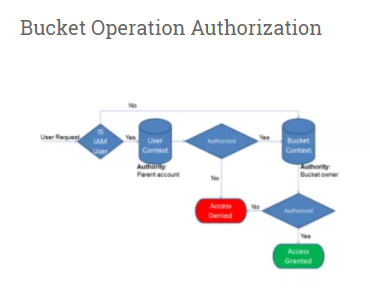
#### **S3 evaluates the policies in 3 context**

* **User context** is basically the context in which S3 evaluates the User policy that the parent AWS account (context authority) attaches to the user
* **Bucket context** is the context in which S3 evaluates the access policies owned by the bucket owner (context authority) to check if the bucket owner has not explicitly denied access to the resource
* **Object context** is the context where S3 evaluates policies owned by the Object owner (context authority)

#### **Analogy**

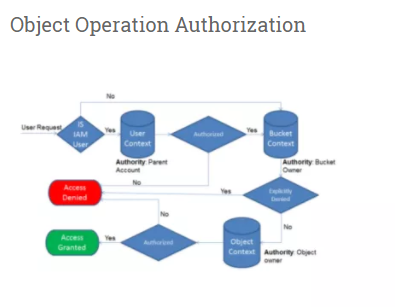
* Consider 3 Parents (AWS Account) A, B and C with Child (IAM User) AA, BA and CA respectively
* Parent A owns a Toy box (Bucket) with Toy AAA and also allows toys (Objects) to be dropped and picked up
* Parent A can grant permission (User Policy OR Bucket policy OR both) to his Child AA to access the Toy box and the toys
* Parent A can grant permissions (Bucket policy) to Parent B (different AWS account) to drop toys into the toys box. Parent B can grant permissions (User policy) to his Child BA to drop Toy BAA
* Parent B can grant permissions (Object ACL) to Parent A to access Toy BAA
* Parent A can grant permissions (Bucket Policy) to Parent C to pick up the Toy AAA who in turn can grant permission (User Policy) to his Child CA to access the toy
* Parent A can grant permission (through IAM Role) to Parent C to pick up the Toy BAA who in turn can grant permission (User Policy) to his Child CA to access the toy

#### **Bucket Operation Authorization**



1. If the requester is an IAM user, the user must have permission (User Policy) from the parent AWS account to which it belongs
2. Amazon S3 evaluates a subset of policies owned by the parent account. This subset of policies includes the user policy that the parent account attaches to the user.
3. If the parent also owns the resource in the request (in this case, the bucket), Amazon S3 also evaluates the corresponding resource policies (bucket policy and bucket ACL) at the same time.
4. Requester must also have permissions (Bucket Policy or ACL) from the bucket owner to perform a specific bucket operation.
5. Amazon S3 evaluates a subset of policies owned by the AWS account that owns the bucket. The bucket owner can grant permission by using a bucket policy or bucket ACL.
6. Note that, if the AWS account that owns the bucket is also the parent account of an IAM user, then it can configure bucket permissions in a user policy or bucket policy or both

#### **Object Operation Authorization**



1. If the requester is an IAM user, the user must have permission (User Policy) from the parent AWS account to which it belongs.
2. Amazon S3 evaluates a subset of policies owned by the parent account. This subset of policies includes the user policy that the parent attaches to the user.
3. If the parent also owns the resource in the request (bucket, object), Amazon S3 evaluates the corresponding resource policies (bucket policy, bucket ACL, and object ACL) at the same time.
4. If the parent AWS account owns the resource (bucket or object), it can grant resource permissions to its IAM user by using either the user policy or the resource policy.
5. S3 evaluates policies owned by the AWS account that owns the bucket.
6. If the AWS account that owns the object in the request is not same as the bucket owner, in the bucket context Amazon S3 checks the policies if the bucket owner has explicitly denied access to the object.
7. If there is an explicit deny set on the object, Amazon S3 does not authorize the request.
8. Requester must have permissions from the object owner (Object ACL) to perform a specific object operation.
9. Amazon S3 evaluates the object ACL.
10. If bucket and object owners are the same, access to the object can be granted in the bucket policy, which is evaluated at the bucket context.
11. If the owners are different, the object owners must use an object ACL to grant permissions.
12. If the AWS account that owns the object is also the parent account to which the IAM user belongs, it can configure object permissions in a user policy, which is evaluated at the user context.

**Permission Delegation**

* If an AWS account owns a resource, it can grant those permissions to another AWS account.
* That account can then delegate those permissions, or a subset of them, to users in the account. This is referred to as permission delegation.
* But an account that receives permissions from another account cannot delegate permission cross-account to another AWS account.
* If the Bucket owner wants to grant permission to the Object which does not belong to it to an other AWS account it cannot do it through cross-account permissions and need to define a IAM role which can be assumed by the AWS account to gain access
* ----------------------------------------------------------------------------------------------------------------------------------------

# AWS S3 Data Protection Overview

* Amazon S3 provides a S3 data protection using highly **durable** storage infrastructure designed for mission-critical and primary data storage.
* Objects are redundantly stored on multiple devices across multiple facilities in an S3 region.
* Amazon S3 PUT and PUT Object copy operations synchronously store the data across multiple facilities before returning SUCCESS.
* Once the objects are stored, S3 maintains its durability by quickly detecting and repairing any lost redundancy.
* S3 also regularly verifies the **integrity** of data stored using checksums. If Amazon S3 detects data corruption, it is repaired using redundant data.
* In addition, S3 calculates checksums on all network traffic to detect corruption of data packets when storing or retrieving data
* Data protection against accidental overwrites and deletions can be added by enabling Versioning to preserve, retrieve and restore every version of the object stored
* S3 also provides the ability to protect data in-transit (as it travels to and from S3) and at rest (while it is stored in S3)

## Data Protection

### Data in-transit

S2 allows protection of data in-transit by enabling communication via SSL or using client-side encryption

### Data at Rest

* S3 supports both client side encryption and server side encryption for protecting data at rest
* Using Server-Side Encryption, S3 encrypts the object before saving it on disks in its data centers and decrypt it when the objects are downloaded
* Using Client-Side Encryption, you can encrypt data client-side and upload the encrypted data to S3. In this case, you manage the encryption process, the encryption keys, and related tools.

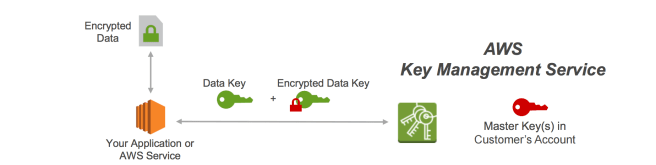
#### **Server-Side Encryption**

* Server-side encryption is about data encryption at rest
* Server-side encryption encrypts only the object data. Any object metadata is not encrypted.
* S3 handles the encryption (as it writes to disks) and decryption (when you access the objects) of the data objects
* There is no difference in the access mechanism for both encrypted or unencrypted objects and is handled transparently by S3

##### **Server-Side Encryption with Amazon S3-Managed Keys (SSE-S3)**

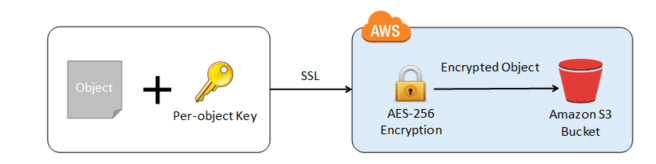
* Each object is encrypted with a unique data key employing strong multi-factor encryption.
* SSE-S3 encrypts the data key with a master key that is regularly rotated.
* S3 server-side encryption uses one of the strongest block ciphers available , 256-bit Advanced Encryption Standard (AES-256), to encrypt the data.
* Whether or not objects are encrypted with SSE-S3 can’t be enforced when they are uploaded using pre-signed URLs, because the only way you can specify server-side encryption is through the AWS Management Console or through an HTTP request header

##### **Server-Side Encryption with AWS KMS-Managed Keys (SSE-KMS)**



* SSE-KMS is similar to SSE-S3, but it uses AWS Key management Services (KMS) which provides additional benefits along with additional charges
  + KMS is a service that combines secure, highly available hardware and software to provide a key management system scaled for the cloud.
  + KMS uses customer master keys (CMKs) to encrypt the S3 objects.
  + Master key is never made available
  + KMS enables you to centrally create encryption keys, define the policies that control how keys can be used
  + Allows audit use of key usage to prove they are being used correctly, by inspecting logs in AWS CloudTrail
  + Allows keys to temporarily disabled and re-enabled
  + Allows keys to be rotated regularly
  + Security controls in AWS KMS can help meet encryption-related compliance requirements.
* SSE-KMS enables separate permissions for the use of an envelope key (that is, a key that protects the data’s encryption key) that provides added protection against unauthorized access of the objects in S3.
* SSE-KMS provides the option to create and manage encryption keys yourself, or use a default customer master key (CMK) that is unique to you, the service you’re using, and the region you’re working in.
* Creating and Managing your own CMK gives you more flexibility, including the ability to create, rotate, disable, and define access controls, and to audit the encryption keys used to protect your data.
* Data keys used to encrypt your data are also encrypted and stored alongside the data they protect and are unique to each object
* Process flow
  + An application or AWS service client requests an encryption key to encrypt data and passes a reference to a master key under the account.
  + Client requests are authenticated based on whether they have access to use the master key.
  + A new data encryption key is created, and a copy of it is encrypted under the master key.
  + Both the data key and encrypted data key are returned to the client.
  + Data key is used to encrypt customer data and then deleted as soon as is practical.
  + Encrypted data key is stored for later use and sent back to AWS KMS when the source data needs to be decrypted.

##### **Server-Side Encryption with Customer-Provided Keys (SSE-C)**



* Encryption keys can be managed and provided by the Customer and S3 manages the encryption, as it writes to disks, and decryption, when you access the objects
* When you upload an object, the encryption key is provided as a part of the request and S3 uses that encryption key to apply AES-256 encryption to the data and removes the encryption key from memory.
* When you download an object, the same encryption key should be provided as a part of the request. S3 first verifies the encryption key and if matches decrypts the object before returning back to you
* As each object and each object’s version can be encrypted with a different key, you are responsible for maintaining the mapping between the object and the encryption key used.
* SSE-C request must be done through HTTPS and S3 will reject any requests made over http when using SSE-C.
* For security considerations, AWS recommends to consider any key sent erroneously using http to be compromised and discarded or rotated
* S3 does not store the encryption key provided. Instead, it stores a randomly salted HMAC value of the encryption key which can be used to validate future requests. The salted HMAC value cannot be used to derive the value of the encryption key or to decrypt the contents of the encrypted object. That means, if you lose the encryption key, you lose the object.

#### **Client-Side Encryption**

Client-side encryption refers to encrypting data before sending it to Amazon S3 and decrypting the data after downloading it

##### **AWS KMS-managed customer master key (CMK)**

* Customer can maintain the encryption CMK with AWS KMS and can provide the CMK id to the client to encrypt the data
* Uploading Object
  + AWS S3 encryption client first sends a request to AWS KMS for the key to encrypt the object data
  + AWS KMS returns a  randomly generated data encryption key with 2 versions a plain text version for encrypting the data and cipher blob to be uploaded with the object as object metadata
  + Client obtains a unique data encryption key for each object it uploads.
  + AWS S3 encryption client uploads the encrypted data and the cipher blob with object metadata
* Download Object
  + AWS Client first downloads the encrypted object from Amazon S3 along with the cipher blob version of the data encryption key stored as object metadata.
  + AWS Client then sends the cipher blob to AWS KMS to get the plain text version of the same, so that it can decrypt the object data.

##### **Client-Side master key**

* Encryption master keys are completely maintained at Client-side
* Uploading Object
  + Amazon S3 encryption client ( for e.g. AmazonS3EncryptionClient *in the AWS SDK for Java*) locally generates randomly a one-time-use symmetric key (also known as a data encryption key or data key).
  + Client encrypts the data encryption key using the customer provided master key
  + Client uses this dataencryption key to encrypt the data of a single S3 object (for each object, the client generates a separate data key).
  + Client then uploads the encrypted data to Amazon S3 and also saves the encrypted data key and itsmaterial description  as object metadata (x-amz-meta-x-amz-key) in Amazon S3 by default
* Downloading Object
  + Client first downloads the encrypted object from Amazon S3 along with the object metadata.
  + Using the material description in the metadata, the client first determines which master key to use to decrypt the encrypted data key.
  + Using that master key, the client decrypts the data key and uses it to decrypt the object
* Client-side master keys and your unencrypted data are never sent to AWS
* If the master key is lost the data cannot be decrypted

# S3 Best Practices

## Performance

### Multiple Concurrent PUTs/GETs

* S3 scales to support very high request rates. If the request rate grows steadily, S3 automatically partitions the buckets as needed to support higher request rates.
* If the typical workload involves only occasional bursts of **100 requests per second and less than 800 requests per second**, AWS scales and handle it.
* If the typical workload involves request rate for a bucket to **more than 300 PUT/LIST/DELETE requests per second or more than 800 GET requests per second**, its recommended to open a support case to prepare for the workload and avoid any temporary limits on your request rate.
* S3 best practice guidelines can be applied only if you are routinely **processing 100 or more requests per second**
* Workloads that include a mix of request types
  + If the request workload are typically a mix of GET, PUT, DELETE, or GET Bucket (list objects), choosing **appropriate key names** for the objects ensures better performance by providing low-latency access to the S3 index
  + This behavior is driven by how S3 stores key names.
    - S3 maintains an index of object key names in each AWS region.
    - Object keys are stored lexicographically (UTF-8 binary ordering) across multiple partitions in the index i.e. S3 stores key names in alphabetical order.
    - Object keys are stored in across multiple partitions in the index and the key name dictates which partition the key is stored in
    - Using a sequential prefix, such as timestamp or an alphabetical sequence, increases the likelihood that S3 will target a specific partition for a large number of keys, overwhelming the I/O capacity of the partition.
  + Introduce **some randomness** in the key name prefixes, the key names, and the I/O load, will be distributed across multiple index partitions.
  + It also ensures scalability regardless of the number of requests sent per second.
* Workloads that are GET-intensive
  + Cloudfront can be used for performance optimization and can help by
    - distributing content with low latency and high data transfer rate.
    - caching the content and thereby reducing the number of direct requests to S3
    - providing multiple endpoints (Edge locations) for data availability
    - available in two flavors as Web distribution or RTMP distribution

### PUTs/GETs for Large Objects

* AWS allows Parallelizing the PUTs/GETs request to improve the upload and download performance as well as the ability to recover in case it fails
* For PUTs, **Multipart upload** can help improve the uploads by
  + performing multiple uploads at the same time and maximizing network bandwidth utilization
  + quick recovery from failures, as only the part that failed to upload needs to be re-uploaded
  + ability to pause and resume uploads
  + begin an upload before the Object size is known
* For GETs, **range http header** can help to improve the downloads by
  + allowing the object to be retrieved in parts instead of the whole object
  + quick recovery from failures, as only the part that failed to download needs to be retried.

### List Operations

* Object key names are stored lexicographically in Amazon S3 indexes, making it hard to sort and manipulate the contents of LIST
* S3 maintains a single lexicographically sorted list of indexes
* Build and maintain Secondary Index outside of S3 for e.g. DynamoDB or RDS to store, index and query objects metadata rather then performing operations on S3

## Security

* Use **Versioning**
  + can be used to protect from unintended overwrites and deletions
  + allows the ability to retrieve and restore deleted objects or rollback to previous versions
* Enable additional security by configuring a bucket to enable MFA (Multi-Factor Authentication) delete
* **Versioning does not prevent Bucket deletion** and must be backed up, as if accidentally or maliciously deleted the data is lost
* Use Cross Region replication feature to backup data to a different region
* When using VPC with S3, use VPC S3 endpoints as
  + are horizontally scaled, redundant, and highly available VPC components
  + help establish a private connection between VPC and S3 and the traffic never leaves the Amazon network

## Cost

* Optimize S3 storage cost by selecting an appropriate storage class for objects
* Configure appropriate lifecycle management rules to move objects to different storage classes and expire them

## Tracking

* Use Event Notifications to be notified for any put or delete request on the S3 objects
* Use CloudTrail, which helps capture specific API calls made to S3 from the AWS account and delivers the log files to an S3 bucket
* Use CloudWatch to monitor the Amazon S3 buckets, tracking metrics such as object counts and bytes stored and configure appropriate actions