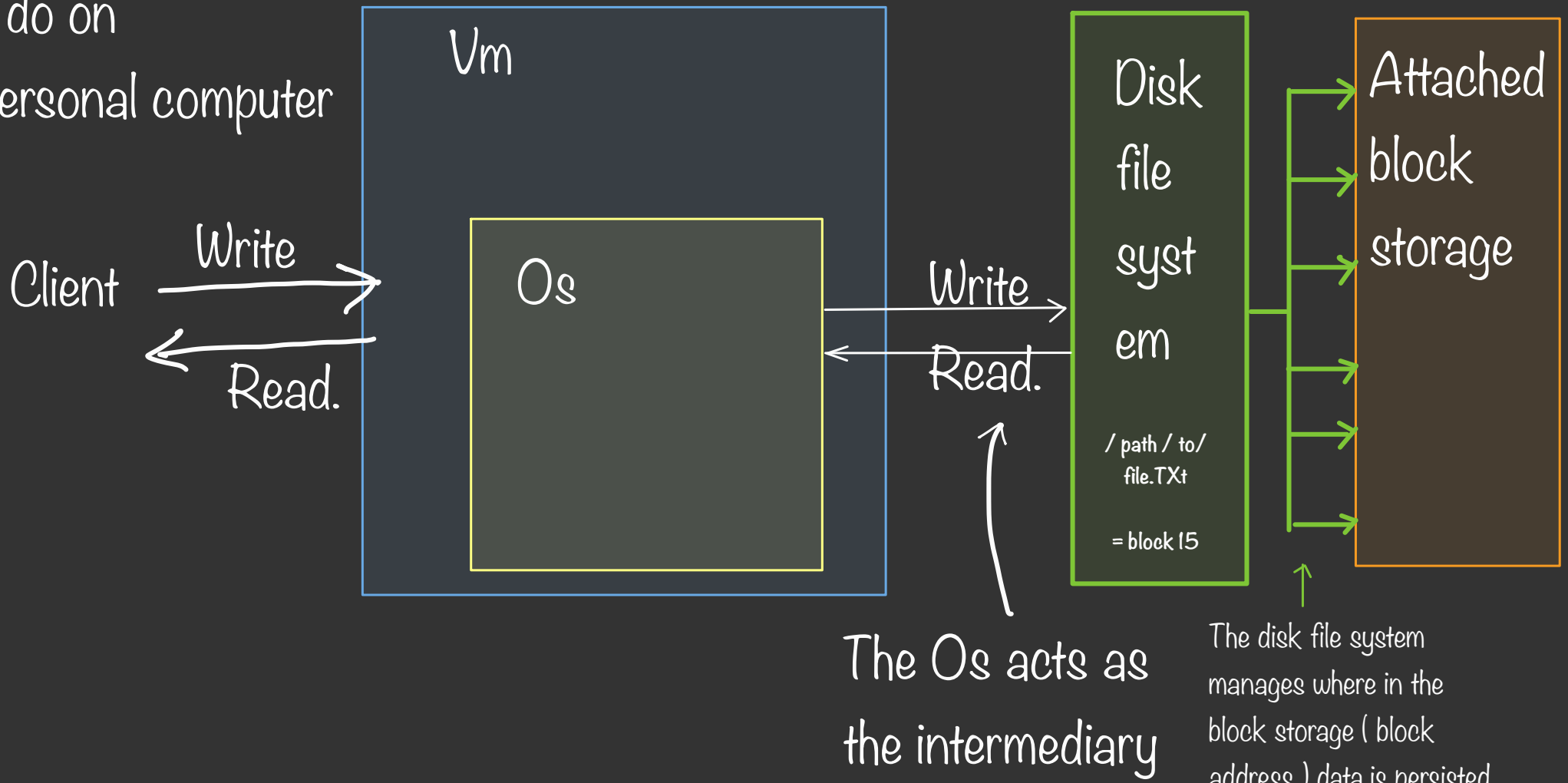
Lecture 7 slides

Block level storage with a disk system can be used store files as a

we do on
a personal computer



Different file systems - deep dive later.

Why block storage?

Systems / applications that use system calls to access files
Like a database - block storage is faster can't use rest based object store
For file system - there are extra path resolution that happens to identify a block,

Why is Bs **faster**?

Bs uses limited metadata and unique identifiers assigned to each block for faster read / write.
Reduced data transfer because of metredata. → ultra low latency Db operations → EBS is a good volume for Db applications.
Block storage models allow multiple paths to underlying data
But file system based models allow only one.

Allows for **frequent modifications** and is also **scalable.**

1. New blocks can be added in real time. To manage capacity
2. Instead of modifying whole file, only the block is modified.

Data grouping..

Frequently accessed data → warm blocks, others in cold blocks
Warm blocks can be kept on s sds while Cold on low cost hard drives.

Lifetime - E Bs volumes are not part of the vm instances and are attached through the network
They continue to exit even after vm is terminated.

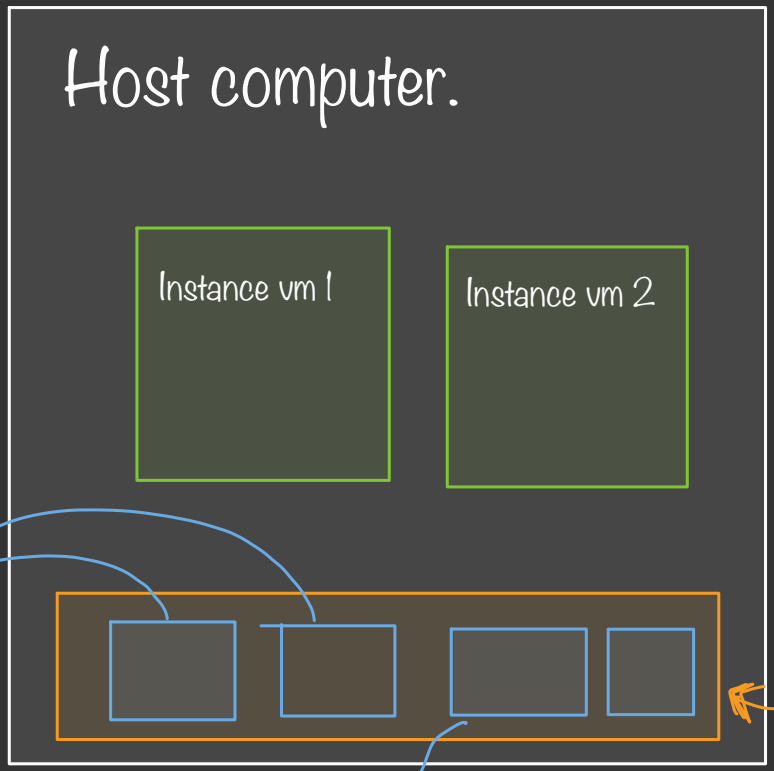
Availability.: designed for 5 nines availability. Automatically replicated within az to provide high availability and durability.

Snapshots: Ę Bs takes point in time snapshots

- Only the delta is saved and charges are only for the delta.
- These are saved internally in s 3.→ can be moved to archive tier for 75% cheaper
- can be restored but are expensive.

Instance store

- Physical hard disk attached to underlying vm.
- large number of I/O ops. As compared to network attached.,
- caching - buffering
- temp storage only - will be deleted if instance goes down or rebooted.



[Cost included along with vm]

Multiple ephemeral volumes

Instance store.

Further points

- does not have out of box solution for data encryption
- helps avoid noisy neighbor problem

Elastic File System EFS - Not present in GCP

- Provides simple and **scalable** file storage for use with EC2 instances
- Supports NFS versions 4 and 4.1 which makes it **easier to migrate legacy applications to AWS**
- Highly available and durable because it **stores data across multiple AZs** in a given region - **Cannot be used as Boot Volume!!** as OS data has to be in the same machine

Storage classes in EFS

- Multiple storage classes based on performance, data access frequency and cost requirements of workloads
- Regional file systems - stores data across multiple AZs - resilient
- One Zone file system - For workloads that don't require multi AZ resilience

EFS Lifecycle Management

- EFS automatically moves files across classes
- We can **set lifecycle policy**, EFS automatically handles this movement

CANNOT BE USED AS ROOT VOLUME!!

Accessing Data in EFS

- Once mounted to a VM, EFS provides standard File system interface
- **Multiple VMs can access EFS at the same time** - needs access control here
- **Designed to handle multi threaded applications** and apps that **concurrently access data** from multiple VMs that require substantial levels of aggregate IOPS

OBJECT STORAGE SERVICE

There are a few operations that we can complete with just the metadata, this is where HTTP methods like HEAD and Options come in to the picture. The cost model of S3 is heavily based on the bandwidth of data in and data out, so when we just work with metadata - this leads to lesser data flow across the network thereby leads to **cost savings**

General Overview

- When dealing with files move to OSS
- Each object has a **globally unique identifier**, **metadata** and the data itself
- By globally unique it is literally unique throughout AWS's storage -across customers and regions
- There is no level of abstraction on top of these objects, they are the primary entity in the OSS - this allows for **granular permissions, access and security policies directly on each of these objects**

How does S3 work internally

Physical Storage :

- Distributed environment across multiple devices and facilities
- Automatically migrated across different storage classes based on lifecycle policy

GLobal Uniqueness:

- Stored across a network of data centers - each object's unique id ensures, it can be located and retrieved from anywhere in the AWS network

Replication:

- Cross region : Compliance, redundancy, availability, durability
- Same Region: Replicate within same region in different buckets - Maybe for stricter compliance

Data Consistency Model For S3 and consistency in general

To maintain strong consistency, instead of rewriting existing objects, a new copy of the object is created / **the new object replaces the old object**. This way whenever a fetch call is made, the latest available object is made available.

In case we have **read replicas that are lagging behind the primary , there might be cases where we get outdated data**, but we always get the latest one when we create new object.... Need to figure out why ? - How is the read replica able to get the latest data in this case, where did the lag go ?

S3 Storage Classes / tiers

- Standard - General Purpose Storage for frequently accessed data
- Intelligent Tiering - Automatic cost savings for data with changing / unknown access patterns
- Infrequently Accessed -
- Glacier - Long lived data that is accessed rarely
- Archive - Data that is very rarely accessed

The cost model changes based on the tier that is being used, the two parameters are data stored and data retrieved. They sort of have an inverse relationship across the tiers. **Keeping the standard to the left and the archive to the right. As we move from standard to archive, while data storage cost reduces, data retrieval cost increases.**

Other features and important considerations

Logging:

- Access Log Record - contains details - request type, resources specified and the time and date of request being processed
- It is important in object store because rather than processing requests through our application, it is done on the Object Storage Server directly through HTTP requests, so we need to know what was requested when

Event **Notifications**:

- Object created
- Object removed
- Reduced Redundancy server, **object lost** event

Versioning:

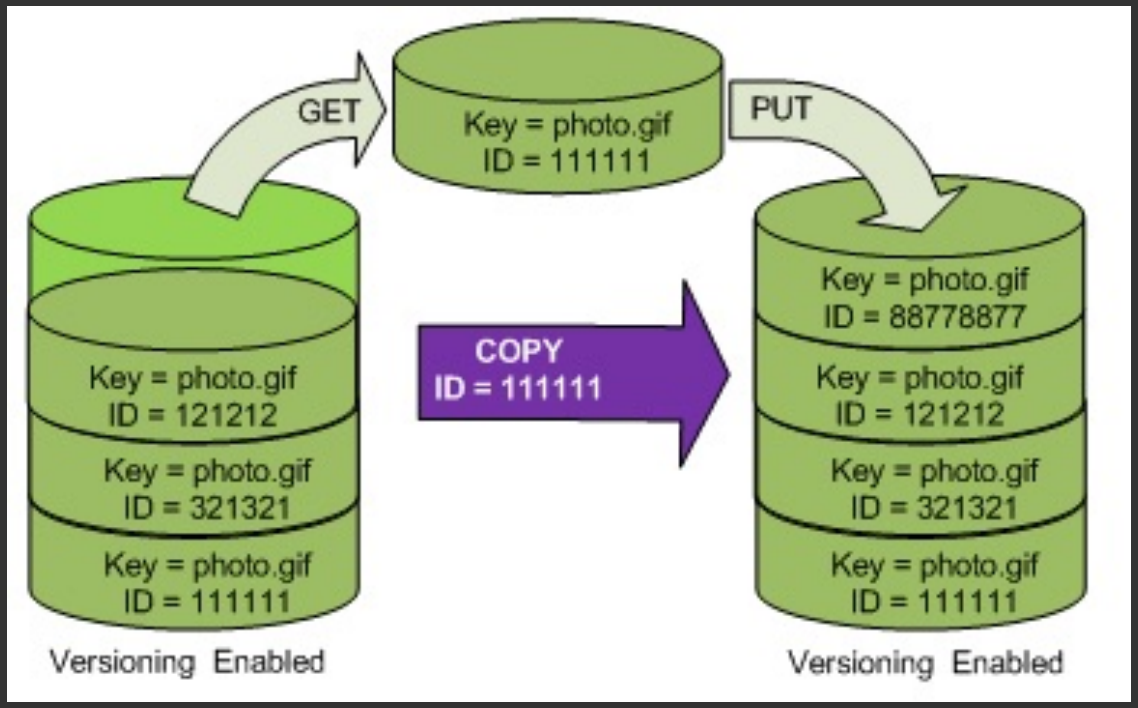
- **Disabled** by **default** for every bucket
- Optional feature and costs money
- In the get request, only the latest version of the object is fetched

In a versioning enabled bucket, we can perform the following **operations**:

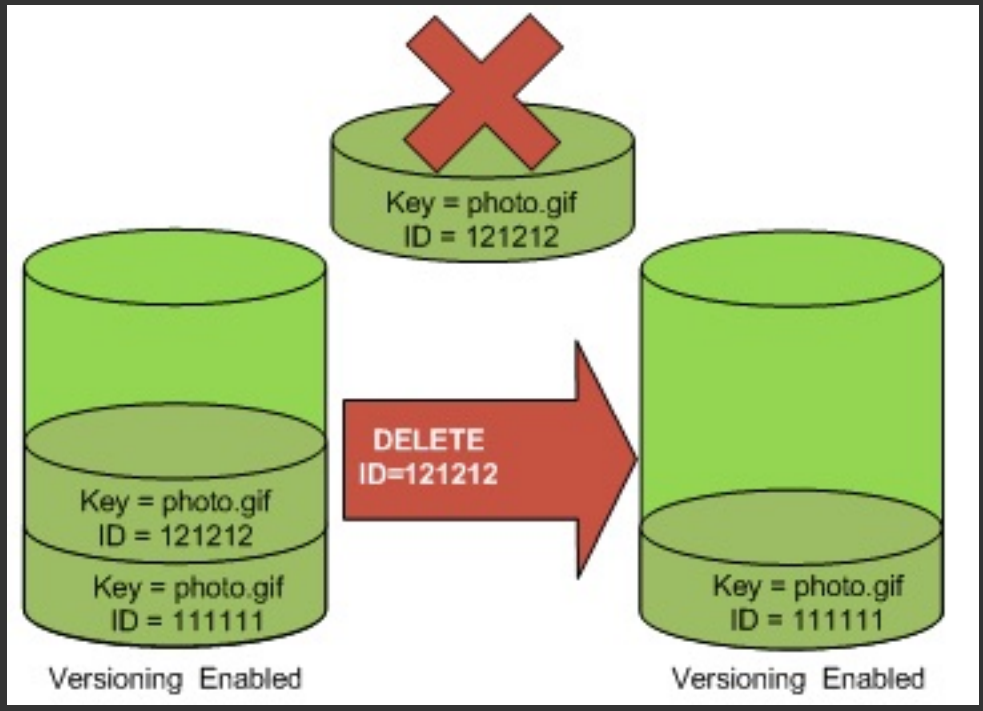
- Updating the object / updating the current version
- Restoring previous version
- Deleting object versions
- Retrieving object versions
- Listing objects

Restoring to a previous version can happen in two ways. One is by deleting the current version, or by copying over a previous version and making it the current version.

1. Restoring a Previous version



2. Deleting the current version

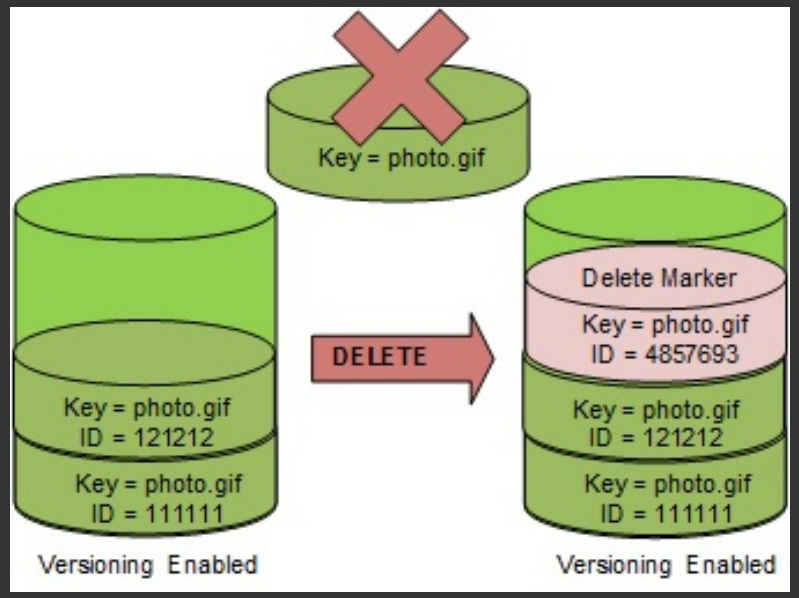


Deleting objects

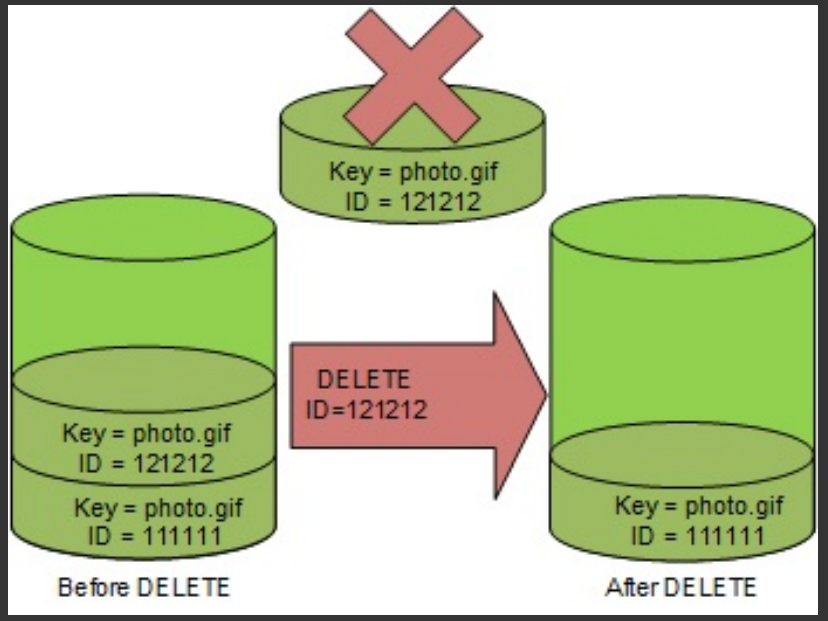
It is unclear what happens to the object after a delete call in a non version controlled bucket. We know for a fact that it get's hidden from the UI.

For a version controlled bucket two things can happen when we delete an object.

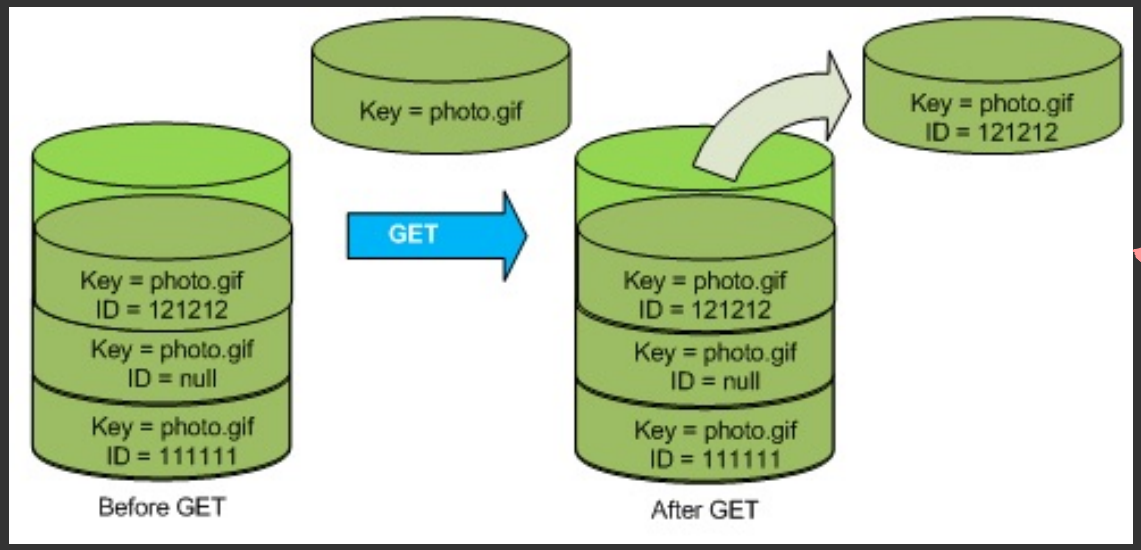
.delete object



.delete object versionId

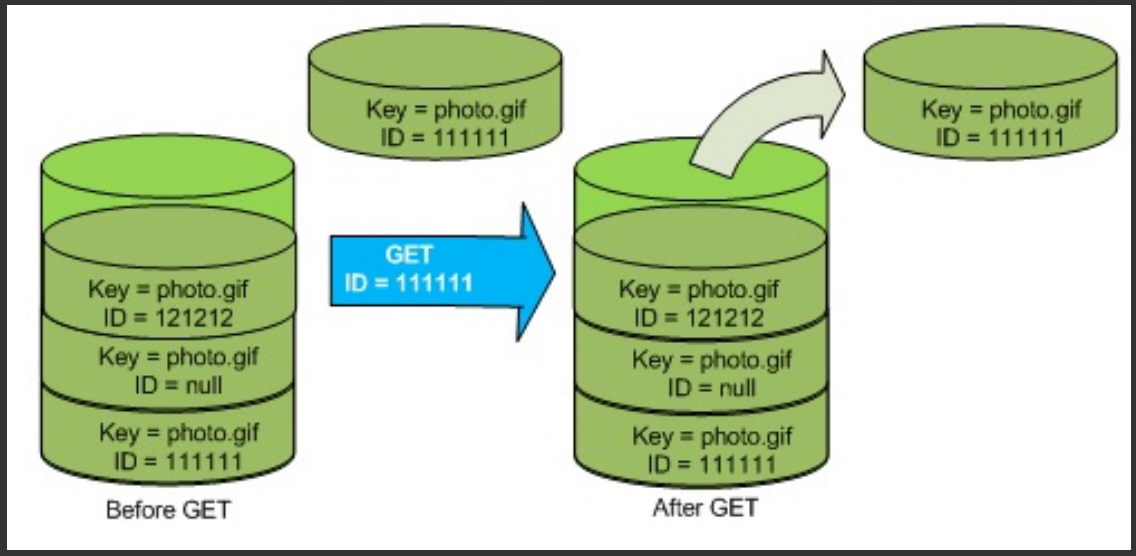


Retrieving Objects From a version controlled bucket



Get Without mentioning version id

Get with version id mentioned



Tagging:

- Can be applied to S3 objects any time
- This let's us create IAM policies, sets up S3 lifecycle policies and customize storage metrics through tags
- Further these tags also can then manage transition between storage classes and expire objects in the background

Why can't S3 be used as a file system

- S3 is not POSIX compliant
- Cannot perform system operations, but rather interact through REST API calls
- Has a flat namespace

In computing, a namespace is a set of signs (names) that are used to identify and refer to objects of various kinds. A namespace ensures that all of a given set of objects have unique names so that they can be easily identified.

Namespaces are commonly structured as hierarchies to allow reuse of names in different contexts. As an analogy, consider a system of naming of people where each person has a given name, as well as a family name shared with their relatives. If the first names of family members are unique only within each family, then each person can be uniquely identified by the combination of first name and family name; there is only one Jane Doe, though there may be many Janes. Within the namespace of the Doe family, just "Jane" suffices to unambiguously designate this person, while within the "global" namespace of all people, the full name must be used.