**Simple-Storage-Service-(S3)**

**S3 Security**

**S3 is private by default!** The only identity which has any initial access to an S3 bucket is the account root user of the account which owns that bucket.

**S3 Bucket Policy**

This is a **resource policy**

* controls who has access to that resource
* can allow or deny access from different accounts
* Explicit “Principal” section in a policy
* can allow or deny anonymous principals
  + this is explicitly declared in the bucket policy itself.

Different from an **identity policy**

* controls what that identity can access
* can only be attached to identities in your own account
  + no way of giving an identity in another account access to a bucket.

Each bucket can only have one policy, but it can have multiple statements.

**ACLs (Legacy)**

A way to apply a subresource to objects and buckets. These are legacy and AWS does not recommend their use. They are inflexible and allow simple permissions.

**Block Public Access**

**S3 Exam PowerUp**

When to use Identity Policy or Bucket Policy:

Identity

* Controlling high mix of different resources.
  + Not every service supports resource policies.
* Want to manage permissions all in one place, use IAM.
* Must have access to all accounts accessing the information.

Bucket

* Managing permissions on a specific product.
* If you need anonymous or cross account access.

ACLs: NEVER - unless you must.

**S3 Static Hosting**

Normal access is via AWS APIs. This allows access via HTTP using a web browser.

When you enable static website hosting you need two HTML files:

* index document
  + default page returned from a website
  + entry point for most websites
* error document
  + similar to index, but only when something goes wrong

Static website hosting creates a **website endpoint**.

This is influenced by the bucket name and region it is in. This cannot be changed.

You can use a custom domain for a bucket, but then the bucket name matters. The name of the bucket must match the domain.

**Offloading**

Instead of using EC2 to host an entire website, the compute service can generate a HTML file which points to the resources hosted on a static bucket. This ensures the media is retrieved from S3 and not EC2.

**Out-of-band pages**

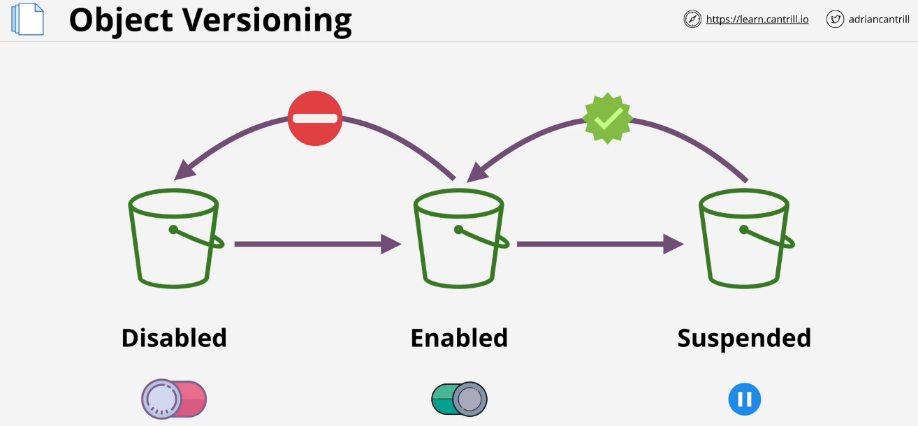
This may be an error page to display maintenance if the server goes offline. We could then change our DNS and move customers to a backup website on S3.

**S3 Pricing**

* Cost to store data, per GB / month fee
  + Prorated for less than a GB or month.
* Data transfer fee
  + Data in is always free
  + Data out is a per GB charge
* Each operation has a cost per 1000 operations.
  + Can add up for static website hosting with many requests.

**Object Versioning and MFA Delete**

**Without Versioning:**

* Each object is identified solely by the object key, it's name.
* If you modify an object, the original object is replaced.
* The attribute, **ID of object**, is set to **null**.

**Versioning:**

* This is off by default.
* Once it is turned on, it cannot be turned off.
* Versioning can be suspended and enabled again.
* This allows for multiple versions of objects within a bucket.
* Objects which would modify objects **generate a new version** instead.

The latest or current version is always returned when an object version is not requested.

When an object is deleted, AWS puts a **delete marker** on the object and hides all previous versions. You could delete this marker to enable the item.

To delete an object, you must delete all the versions of that object using their version marker.

**MFA Delete**

Enabled within version configuration in a bucket. This means MFA is required to change bucket versioning state. MFA is required to delete versions of an object.

In order to change a version state or delete a particular version of an object, you need to provide the serial number of your MFA token as well as the code it generates. These are concatenated and passed with any API calls.

**S3 Performance Optimization**

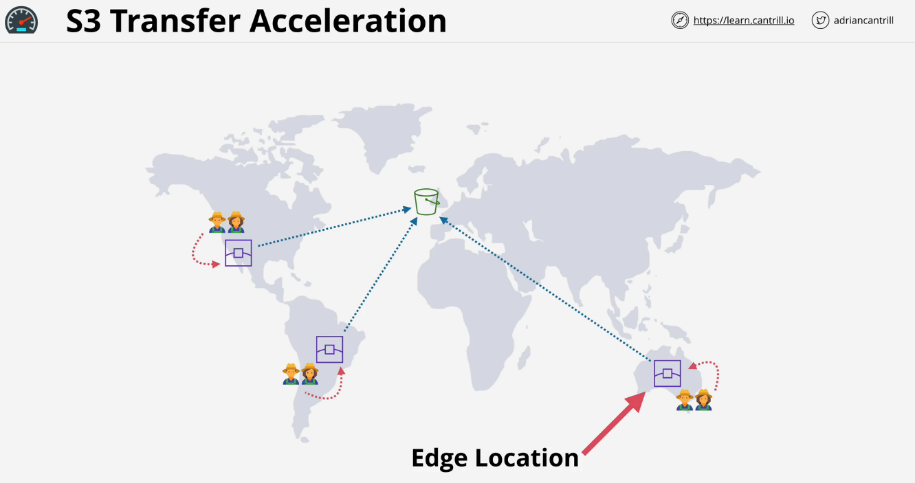
**Single PUT Upload**

* Objects uploaded to S3 are sent as a single stream by default.
* If the stream fails, the upload fails and requires a restart of the transfer.
* Single PUT upload up to 5GB

**Multipart Upload**

* Data is broken up into smaller parts.
* The minimum data size is 100 MB.
* Upload can be split into maximum of 10,000 parts.
  + Each part can range between 5MB and 5GB.
  + Last leftover part can be smaller than 5MB as needed.
* Parts can fail in isolation and restart in isolation.
* The risk of uploading large amounts of data is reduced.
* Improves transfer rate to be the speed of all parts.

**S3 Accelerated Transfer**

* Off by default.
* Uses the network of AWS edge locations to speed up transfer.
* Bucket name cannot contain periods.
* Name must be DNS compatible.
* Benefits improve the larger the location and distance.
  + The worse the start, the better the performance benefits.

**Encryption 101**

**Encryption at Rest**

* An example is a password on a laptop
  + If the laptop is stolen, the data is already encrypted and useless.
* Commonly within cloud environments. Even if someone could find and access the base storage device, they can't do anything with it.
* Only one entity involved

**Encryption in Transit**

* An encryption tunnel outside the raw data.
* Anyone looking from the outside will only see a stream of scrambled data.
* Used when there are multiple parties or systems at play.

**Terms**

* plaintext: unencrypted data not limited to text
* Algorithm: Blowfish, AES, RC4, DES, RC5 & RC6
* key: a password
* ciphertext: encrypted data generated by an algorithm from plaintext and a key

**Symmetric Encryption**

The key is handed from one entity to another before the data. This is difficult because the key needs to be transferred securely. If the data is time sensitive, the key needs to be arranged beforehand.

**Asymmetric Encryption**

* **public key:** cannot decrypt data but can generate ciphertext
* **private key:** can decrypt data encrypted by the public key

The public key is uploaded to cloud storage. The data is encrypted and sent back to the original entity. The private key can decrypt the data.

This is secure because stolen public keys can only encrypt data. Private keys must be handled securely. Example- SSL, SSH, TLS etc

**Signing**

Encryption by itself does not prove who encrypted the data.

1. An entity can encrypt a message with their private key.
2. Their public key is hosted in an accessible location.
3. The receiving party can use the public key to confirm who sent the message.

**Steganography**

Encryption is obvious when used. There is no denying that the data was encrypted. Someone could force you to decrypt the data packet.

A file can be hidden in an image or other file. If it difficult to find the message unless you know what to look for.

One party would take another party's public key and encrypt some data to create ciphertext. That ciphertext can be hidden in another file so long as both parties know how the data will be hidden.

**Key Management Service (KMS)**

* Regional service
  + Every region is isolated when using KMS.
* Public service
  + Occupies the AWS public zone and can be connected to from anywhere.
* Create, store, and manage keys.
  + Can handle both symmetric and asymmetric keys.
* KMS can perform cryptographic operations itself.
* Keys never leave KMS.
* Keys use **FIPS 140-2 (L2)** security standard.
  + Some features are compliant with Level 3.
  + All features are compliant with Level 2.

**CMKs - Customer Master Keys**

* Managed by KMS and used within cryptographic operations.
* AWS services, applications, and the user can all use them.
* Think of them as a container for the actual physical master keys.
* These are all backed by **physical** key material.
* You can generate or import the key material.
* CMKs can be used for up to **4KB of data**.

It is logical and contains

* Key ID: unique identifier for the key
* Creation Date
* Key Policy: a type of resource policy
* Description
* State of the Key: active or not

**Data Encryption Key (DEK)**

* Generated by KMS using the CMK and GenerateDataKey operation.
* Used to encrypt data larger than 4KB in size.
* Linked to a specific CMK so KMS can tell that a specific DEK was generated with a specific CMK.

KMS does not store the DEK, once provided to a user or service, it is discarded. KMS doesn't actually perform the encryption or decryption of data using the DEK or anything past generating them.

When the DEK is generated, KMS provides two version.

* Plaintext Version - This can be used immediately.
* Ciphertext Version - Encrypted version of the DEK.
  + This is encrypted by the CMK that generated it.
  + In the future it can be decrypted by KMS using the CMK assuming you have the permissions.

**Architecture**

1. DEK is generated right before something is encrypted.
2. The data is encrypted with the plaintext version of the DEK.
3. Discard the plaintext data version of the DEK.
4. The encrypted DEK is stored next to the ciphertext generated earlier.

**KMS Key Concepts**

* **Customer Master Keys (CMK)** are isolated to a region.
  + Never leave the region or KMS.
  + Cannot extract a CMK.
  + **AWS managed CMKs**
    - Created automatically by AWS when using a service such as S3 which uses KMS for encryption.
  + **Customer managed CMKS**
    - Created explicitly by the customer.
    - Much more configurable, for example the key policy can be edited.
    - Can allow other AWS accounts access to CMKS

**All CMKs support key rotation.**

* AWS automatically rotates the keys every 1095 days (3 years)
* Customer managed keys rotate every year.

**CMK itself contains:**

* Current backing key, physical material used to encrypt and decrypt
* Previous backing keys created from rotating that material

KMS can create an alias which is a shortcut to a particular CMK. Aliases are also per region. You can create a MyApp1 alias in all regions but they would be separate aliases, and in each region it would be pointing potentially at a different CMK.

**Key Policy (resource policy)**

* Every CMK has one.
* Customer managed CMKs can adjust the policy.
* **Unlike other policies, KMS has to be explicitly told that keys trust the AWS account that they're in.**
* The trust isn't automatic so be careful when adjusting key policies.
* You always need a key policy in place so the key trusts the account and so that the account can manage it by applying IAM permission policies to IAM users in that account.
* In order for IAM to work, IAM is trusted by the account, and the account needs to be trusted by the key.
* It sets up this chain of trust from the key to the account to IAM and then to an IAM user, if they're granted any identity permissions.

**KMS Key Demo**

Linux/macOS commands

aws kms encrypt \

--key-id alias/catrobot \

--plaintext fileb://battleplans.txt \

--output text \

--query CiphertextBlob \

--profile iamadmin-general | base64 \

--decode > not\_battleplans.enc

aws kms decrypt \

--ciphertext-blob fileb://not\_battleplans.enc \

--output text \

--profile iamadmin-general \

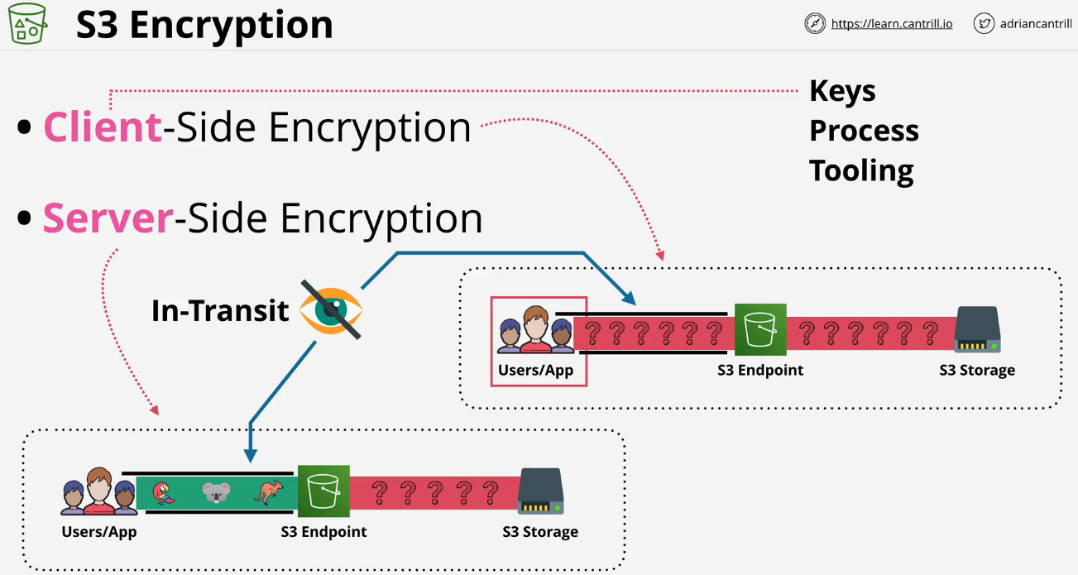
--query Plaintext | base64 --decode > decryptedplans.txt

**Object Encryption**

Buckets aren't encrypted, **objects are**. Multiple objects in a bucket can use a different encryption methods.

Two main methods of encryption S3 is capable of supporting. Both types are encryption at rest. Data sent from a user to S3 is automatically encrypted in transit outside of these methods.

**Client-Side encryption**

* Objects being encrypted by the client before they leave.
* Data being sent the whole time it is sent as cypher text.
* AWS has no way to see into the data.
* The encryption burden is on the customer and not AWS.

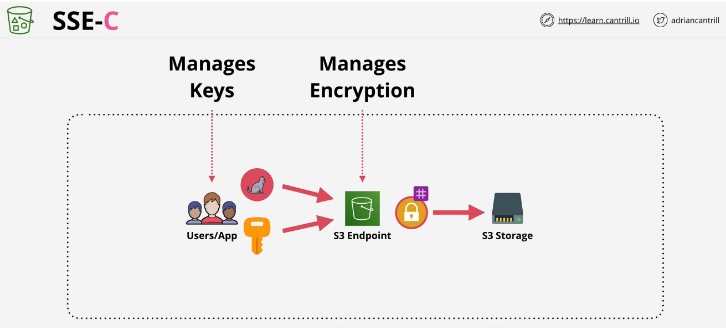
**Server-Side encryption**

* Data is encrypted in transit using HTTPS
* Data inside the tunnel is still in its original unencrypted form.
* Data reaches S3 server in plain text form.
* After S3 sees the data, it is then encrypted.
* AWS will handle some or all of these processes.

**SSE-C (Server-side encryption with customer provided keys)**

* Customer is responsible for the keys themselves.
* S3 services manages the actual encryption and decryption
  + Offloads CPU requirements for encryption.
* Customer still needs to generate and manage the key.
* S3 will see the unencrypted object throughout this process.

**SSE-C Encryption Steps**

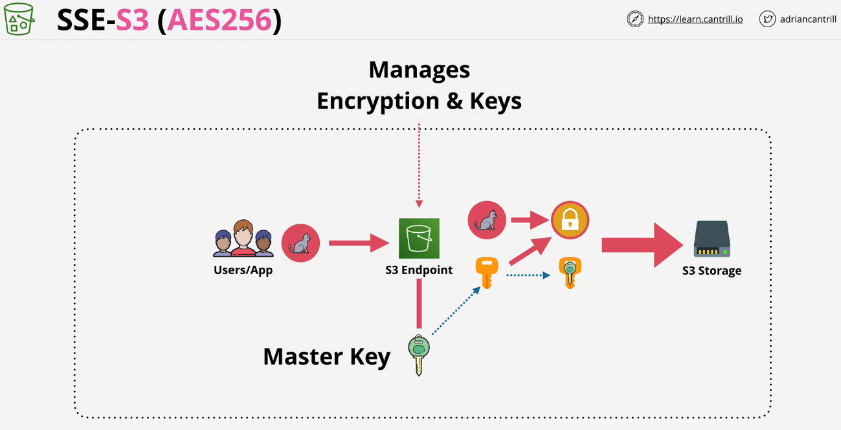
1. When placing an object in S3, you provide encryption key and plaintext object
2. Once the key and object arrive, it is encrypted.
3. A hash of the key is taken and attached to the object. The hash can identify if the specific key was used to encrypt the object.
4. The key is then discarded after the hash is taken.
5. The encrypted and one-way hash are stored persistently on storage.

To decrypt the object, you must tell S3 which object to decrypt and provide it with the key used to encrypt it. If the key that you supply is correct, the proper hash, S3 will decrypt the object, discard the key, and return the plaintext version of the object.

**SSE-S3 AES256 (Server-side encryption with Amazon S3 managed keys)**

AWS handles both the encryption and decryption process as well as the key generation and management. This provides very little control over how the keys are used, but has little admin overhead.

SSE-S3 Encryption Steps

1. When putting data into S3, only need to provide plaintext.
2. S3 generates fully managed and rotated **master key** automatically.
3. S3 generates a key specific for each object that is uploaded.
4. The master key is used to encrypt the specific object key, and the unencrypted version of that key is discarded.
5. The encrypted file and encrypted key are stored side by side in S3.

Three Problems with this method:

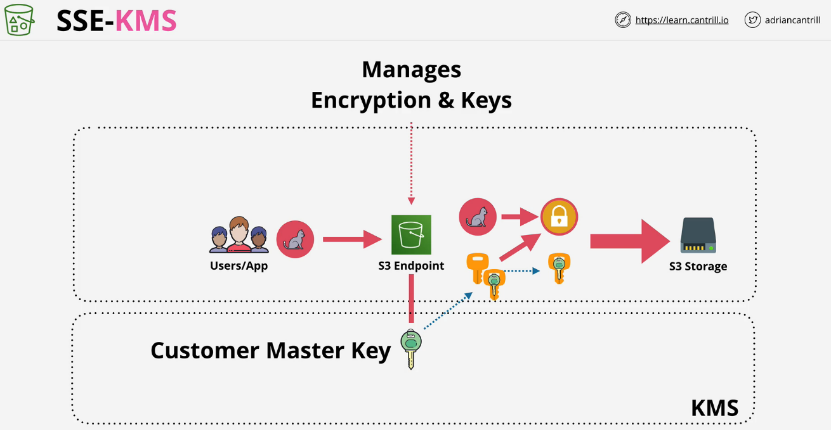
* Not good for regulatory environment where keys and access must be controlled.
* No way to control key material rotation.
* No role separation. A full S3 admin can decrypt data and open objects.

**SSE-KMS (Server-side encryption w/ customer master keys stored in AWS KMS)**

Much like SSE-S3, where AWS handles both the keys and encryption process. KMS handles the master key and not S3. The first time an object is uploaded, S3 works with KMS to create an AWS managed CMK. This is the default key which gets used in the future.

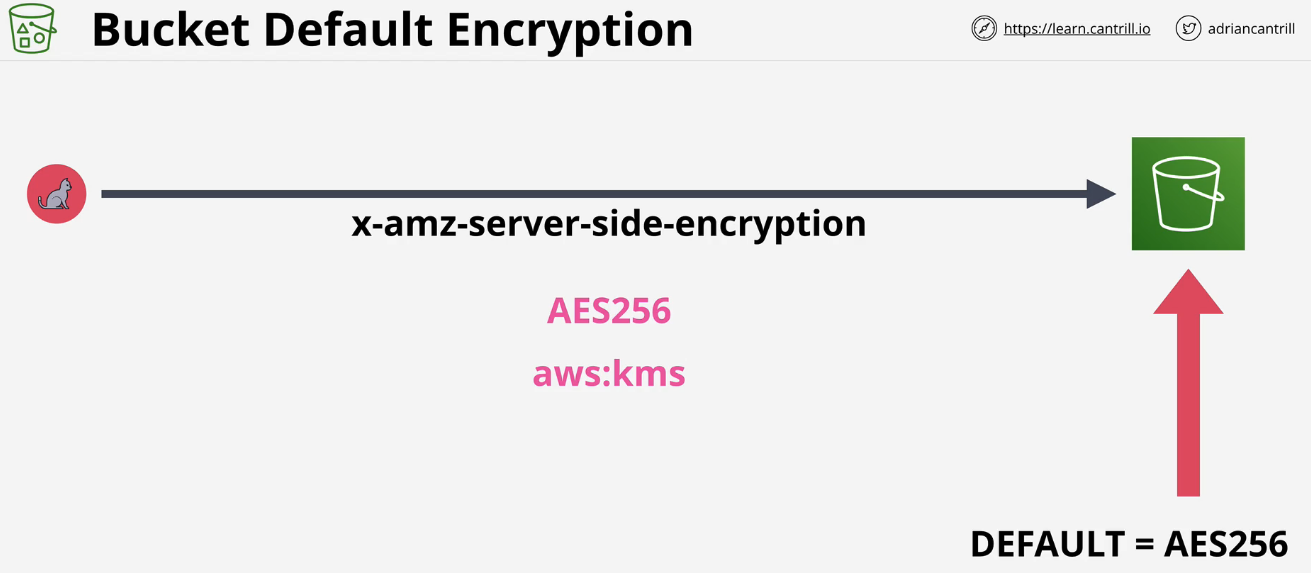
Every time an object is uploaded, S3 uses a dedicated key to encrypt that object and that key is a data encryption key which KMS generates using the CMK. The CMK does not need to be managed by AWS and can be a customer managed CMK.

SSE-KMS Encryption Steps

1. S3 is provided a plaintext version of the data encryption key as well as an encrypted version.
2. The data is encrypted with the plaintext key and the key discarded.
3. The encrypted key is stored alongside the encrypted object.

When uploading an object, you can create and use a customer managed CMK. This allows the user to control the permissions and the usage of the key material. In regulated industries, this is reason enough to use SSE-KMS You can also add logging and see any calls against this key from CloudTrail.

The best benefit is the role separation. To decrypt any object, you need access to the CMK that was used to generate the unique key that encrypted them. The CMK is used to decrypt the data encryption key for that object. That decrypted data encryption key is used to decrypt the object itself. If you don't have access to KMS, you don't have access to the object.



**S3 Object Storage Classes**

Picking a storage class can be done while uploading a specific object. The default is S3 standard. Once an object is uploaded to a specific class, it can be easily changed as long as some conditions are met.

Objects in S3 are stored in a specific region.

**S3 Standard**

* Default AWS storage class that's used in S3, should be user default as well.
* S3 Standard is region resilient, and can tolerate the failure of an AZ.
* Objects are replicated to at least 3+ AZs when they are uploaded.
* 99.999999999% durability
* 99.99% availability
* Offers low latency and high throughput.
* No minimums, delays, or penalties.
* Billing is storage fee, data transfer fee, and request based charge.

All of the other storage classes trade some of these compromises for another.

**S3 Standard-IA**

* Designed for less frequent rapid access when it is needed.
* Cheaper rate to store data you will rarely need, but if you do need it, you need it quickly.
* ~54% cheaper than S3 standard.
* Minimum 128KB charge for each object.
  + Cost benefits might be negated for smaller objects.
* 30 days minimum duration charge per object.
* Retrieval fee for every GB of data retrieved from this class.
* 99.9% availability, slightly lower than standard S3.

Designed for data that isn't accessed often, long term storage, backups, disaster recovery files. The requirement for data to be safe is most important.

**One Zone-IA**

* Designed for data that is accessed less frequently but needed quickly.
* 80% of the base cost of Standard-IA.
* Same minimum size and duration fee as Standard-IA
* Data is only stored in a single AZ, no 3+ AZ replication.
* 99.5% availability, lower than Standard-IA

Great choice for secondary copies of primary data or backup copies.

If data is easily creatable from a primary data set, this would be a great place to store the output from another data set.

Suitable for data which is easily replaced (most cost effective)

**S3 Glacier**

* No immediate access to objects, retrieval in minutes or hours.
* Make a request to access objects then after a duration, you get access.
  + Retrieval time anywhere from 1 min - 12 hrs
* Secure, durable, and low cost storage for archival data.
* 17% of the base cost of S3 standard
* 99.999999999% durability
* 99.99% availability
* 3+ AZ replication
* 40KB minimum object capacity charge
* **90** days minimum storage duration charge.
* Cannot use to make data public or download normally, because data can’t be retrieved/downloaded immediately, time is required to fetch data.
* **Retrieval methods:**
  + **Expedited**: 1 - 5 minutes, but is the most expensive
  + **Standard**: 3 - 5 hours to restore.
  + **Bulk**: 5 - 12 hours. Has the lowest cost and is good for a large set of data.

**S3 Glacier Deep Archive**

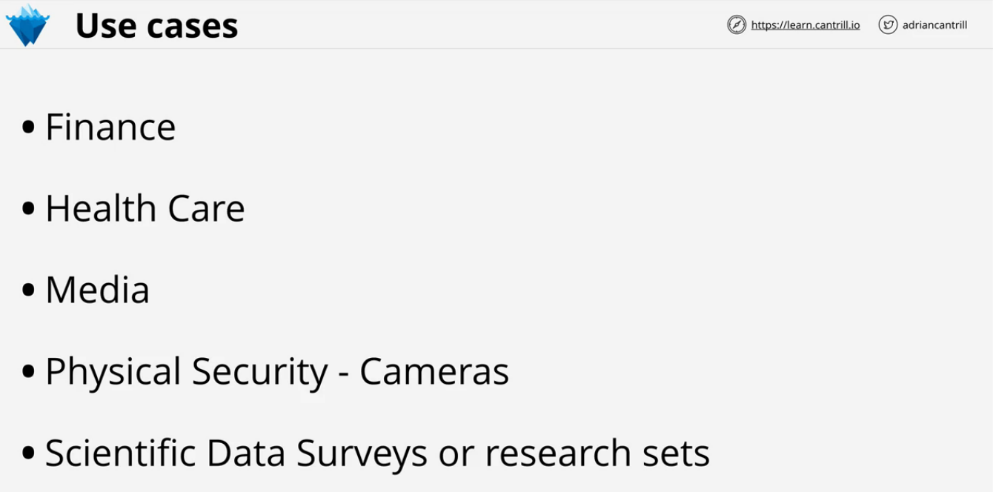
* Designed for long term backups and as a tape-drive replacement.
* 4.3% of the base cost of S3 standard
* **180** days minimum storage duration charge.
* **Retrieval methods:**
  + **Standard**: 12 hours to restore.
  + **Bulk**: 48 hours.
* Cannot use to make data public or download normally, because data can’t be retrieved/downloaded immediately, time is required to fetch data.

**S3 Intelligent-Tiering**

* Combination of standard and standard IA.
* Uses automation to remove overhead of moving objects.
* Additional fee of $0.0025 per 1,000 objects tracked.
* If an object is not accessed for 30 days, it will move into Standard-IA.

This is good for objects that are unknown their access pattern.

Uncertain access and low admin overhead



**Object Lifecycle Management**

Intelligent-Tiering is used for objects where access patterns is unknown. A lifecycle configuration is a set of **rules** that consists of **actions**.

**Transition Actions**

Change the storage class over time such as:

* Move an object from S3 to IA after 90 days
* After 180 days move to Glacier
* After one year move to Deep Archive

Objects must flow downwards, they can't flow in the reverse direction.

**Expiration Actions**

Once an object has been uploaded and changed, you can purge older versions after 90 days to keep costs down.

**S3 Replication**

There are two types of S3 replication available.

* Cross-Region Replication (CRR)
  + Allows the replication of objects from a source bucket to a destination bucket in **different** AWS regions.
* Same-Region Replication (SRR)
  + Allows the replication of objects from a source bucket to a destination bucket in the **same** AWS region.

Architecture for both is similar, only difference is if both buckets are in the same account or different accounts.

The replication configuration is applied to the source bucket and configures S3 to replicate from this source bucket to a destination bucket. It also configures the IAM role to use for the replication process. The role is configured to allow the S3 service to assume it based on its trust policy. The role's permission policy allows it to read objects on the source bucket and replicate them to the destination bucket.

When different accounts are used, the role is not by default trusted by the destination account. If configuring between accounts, you must add a bucket policy on the destination account to allow the IAM role from the source account access to the bucket.

**S3 Replication Options**

* Which objects are replicated.
  + Default is all source objects, but can select a smaller subset of objects.
* Select which storage class the destination bucket will use.
  + Default is the same type of storage, but this can be changed.
* Define the ownership of the objects.
  + The default is they will be owned by the same account as the source bucket.
  + If the buckets are in different accounts, the objects in the destination could be owned by the source account and not allowed access.
* Replication Time Control (RTC)
  + Adds a guaranteed level of SLA within 15 minutes for extra cost.
  + This is useful for buckets that must be in sync the whole time.

**Important Replication Tips**

* Replication is not retroactive.
  + If you enable replication on a bucket that already has objects, the old objects will not be replicated.
* Both buckets must have versioning enabled.
* It is a one way replication process only.
* Replication by default can handle objects that are unencrypted or SSE-S3.
  + With configuration it can handle SSE-KMS, but KMS requires more configuration to work.
  + It cannot replicate objects with SSE-C because AWS does not have the keys necessary.
* Source bucket owner needs permissions to objects. If you grant cross-account access to a bucket. It is possible the source bucket account will not own some of those objects.
* Will not replicate system events, glacier, or glacier deep archive.
* No deletes are replicated.

**Why use replication**

SRR - Log Aggregation SRR - Sync production and test accounts SRR - Resilience with strict sovereignty requirements CRR - Global resilience improvements CRR - Latency reduction

**S3 Presigned URL**

A way to give another person or application access to a object inside an S3 bucket using your credentials in a safe way.

IAM admin can make a request to S3 to generate a presigned URL by providing:

* security credentials
* bucket name
* object key
* expiry date and time
* indicate how the object or bucket will be accessed

S3 will create a presigned URL and return it. This URL will have encoded inside it the details that IAM admin provided. It will be configured to expire at a certain date and time as requested by the IAM admin user.

**S3 Presigned URL Exam PowerUp**

* You can create a presigned URL for an object you have do not have access to. The object will not allow access because your user does not have access.
* When using the URL the permission that you have access to, match the identity that generated it at the moment the item is being accessed.
* If you get an access deny it means the ID never had access, or lost it.
* Don't generate presigned URLs with an IAM role.
  + The role will likely expire before the URL does.

**S3 Select and Glacier Select**

This provides a ways to retrieve parts of objects and not the entire object.

If you retrieve a 5TB object, it takes time and consumes 5TB of data. Filtering at the client side doesn't reduce this cost.

S3 and Glacier select lets you use SQL-like statements to select part of the object which is returned in a filtered way. The filtering happens at the S3 service itself saving time and data.