**AWS's Simple Storage Service (S3)**

* is a secure, durable, highly-scalable object storage
* is accessible via a simple web services interface
* enables users to store and retrieve any amount of data
* can be used alone or together with other AWS services

**AWS S3 Characteristics**

* Durable
* Integrated
* Low Cost
* Available
* Secure (Data transfer: SSL)
* High Performance
* Scalable
* Easy to Use

**AWS S3 Use Cases**

* Static Website Hosting
* Big Data Analytics
* Cloud-native Application Data
* Disaster Recovery
* Backup & Archiving
* Content Storage & Distribution

**S3 Concepts**

**Buckets**

* containers for objects stored in S3
* organizes the S3 namespace at the highest level
* identifies the account responsible for charges
* plays a role in access control
* serves as the unit of aggregation for usage reporting

**Objects**

* fundamental entities stored in S3
* consists of data & metadata
  + data portion is opaque to S3 (the representation of the data is invisible to the system itself)
  + metadata is a set of name-value pairs that describe the object
  + each object is uniquely identified within a bucket by a key (name) and a version ID

**Keys**

* unique identifiers for an object within a bucket
* every object in a bucket has exactly one key
* maximum size: 1024 bytes UTF-8
* looks a lot like a URL or filepath
* combination of a bucket, key, & version ID uniquely identify each object
* Example key:<http://doc.s3.amazonaws.com/2006-03-01/AmazonS3.wsdl>
  + Bucket Name: doc
  + Object Key: 2006-03-01/AmazonS3.wsdl

**Regions**

* the geographical region where Amazon S3 will store the buckets that you create
* choose a region to optimize latency, minimize costs, or address regulatory requirements
* objects within a region never leave the region without a deliberate transfer/replication by the user

**S3 Fundamentals**

* an object store, not a file system
  + files systems scatter data across memory, reassembling it upon retrieval - including no metadata
  + object stores store object data contiguously within memory, and includes metadata during retrieval

* highly available & durable
  + write once, read many
    - S3 buckets are set up inside of Availability Zones (AZs) within a given Region
      * user writes an object
      * object being written hits a load balancer
      * load balancer directs the object to a web server
      * object is stored redundantly across storage in at least two AZs (replicated and protected)
      * indexes, which are located across multiple AZs, are updated
    - Redundancy is built into the underlying AWS system
      * multiple load balancers
      * multiple web servers
      * redundant storage across AZs
      * redundant indexing across AZs

* eventually consistent
  + the idea that given a short period of time, the system will be in a consistent state

* new objects
  + synchronously stores your data across multiple facilities before returning SUCCESS
    - exception in the US-STANDARD region

* potential update issues
  + Write-then-read: could report that key does not exist
  + Write-then-list: might not include key in list
  + Overwrite-then-read: old data could be returned

* potential delete issues
  + delete-then-read: old data could be returned
  + delete-then-list: deleted key could be included in list

* highly scalable data storage

* accessible via APIs (<http://aws.amazon.com/documentation/s3/>)

* accessible via software development kit (SDK) (<http://aws.amazon.com/tools/>)
  + support for Java, Node.js, .NET, Python, Go, Ruby, & PHP

* accessible via AWS CLI
  + Commands:
* **aws s3 ls**
  + List buckets

* **aws s3 cp s3-example-file.txt s3://aws-example-demo**
  + copy a file to an object

* **aws s3 ls s3://aws-example-demo**
  + list bucket contents

* **aws s3 rm s3://aws-example-demo/s3-example-file.txt**
  + delete an object

* **aws s3 sync . s3://aws-example-demo**
  + sync a directory with a bucket (in this case, the current working directory '.')

* **aws s3 rm s3://aws-example-demo --recursive**
  + delete bucket contents

* **aws s3 help**
  + AWS S3 CLI help

* **aws s3 cp help**
  + detailed help regarding a specific CLI command (in this case, the copy command 'cp')

* secure by default
  + IAM policies
    - fine grained control
    - administer as part of role based access
    - apply policies to S3 at role, user, & group level

* Bucket policies
  + fine grained
  + applies policies at the bucket level in S3
  + incorporates user restrictions without using IAM
  + Example policy (common for static web hosting):

**{**

**"Version": "2018-04-22",**

**"Statement": [**

**{**

**"Sid": "AddPerm",**

**"Effect": "Allow",**

**"Principal": "\*",**

**"Action": ["s3.GetObject"],**

**"Resource": ["arn:aws:s3:::examplebucket/\*"]**

**}**

**]**

**}**

* Example policy (restricting access to a specific IP address):

**{**

**"Version": "2018-04-22",**

**"Id": "S3PolicyId1"**

**"Statement": [{**

**"Sid": "IPAllow",**

**"Effect": "Allow",**

**"Principal": "\*",**

**"Action": ["s3.GetObject"],**

**"Resource": ["arn:aws:s3:::examplebucket/\*"],**

**"Condition": {**

**"IpAddress": {"aws:SourceIp": "54.240.143.0/24"},**

**"NotIpAddress": {"aws:SourceIp": "54.240.143.188/32"}**

**}**

**}]**

**}**

* other use cases for bucket policies:
  + granting permissions to multiple accounts with added conditions
  + restricting access to specific HTTP referer
  + granting permission to an Amazon CloudFront Origin Identity
  + adding a policy to require MFA authentication
  + granting cross-account permissions to upload objects while ensuring the bucket owner has full control
  + more: [docs.aws.amazon.com/AmazonS3/latest/dev/example-bucket-policies.html](http://docs.aws.amazon.com/AmazonS3/latest/dev/example-bucket-policies.html)

* ACLs
  + coarsely grained
  + applies access control rules at the bucket and/or object level in S3
  + can be used to grant basic read/write permissions to other AWS accounts
  + there are limits to managing permissions using ACLs
    - for example, you can grant permissions only to other AWS accounts, you cannot grant permissions to users in your account

* fast (upload & download)
  + throughput optimization
    - S3 automatically partitions objects based upon their key prefix
    - too many keys with similar prefixes can slow down object retrieval
      * reversing IDs can help distribute key prefixes better

**Storage Classes**

* Standard
  + affordable ($0.023/GB)
  + designed to sustain the concurrent loss of data between two facilities
  + provides 99.999999999% durability and 99.99% availability

* Reduced Redundancy Storage
  + reduces costs by storing data at lower levels of redundancy ($0.01/GB)
  + provides 99.99% durability and 99.99% availability

* Glacier
  + very low cost ($0.004/GB)
  + suitable for archiving data, where data is infrequently accessed and high retrieval latency is acceptable

* you can specify the storage class of an object when uploading or creating it

**aws s3 cp aws\_example.txt s3://aws-example-bucket/ --storage-class GLACIER**

* you can change the storage class of an object that is already stored in S3 by copying to the same key name in the same bucket

**aws s3 cp s3://aws-example-bucket/aws\_example.txt s3://aws-example-bucket/aws\_example.txt --storage-class STANDARD**

**Encryption & Other Security Features**

**Securing Data in Transit**

* securely upload or download data via SSL-encrypted endpoints using HTTPS
  + alternatively, use a client-encryption library such as the Amazon S3 Encryption Client to encrypt your data before uploading to Amazon S3

**Securing Data at Rest**

* S3 will automatically encrypt your data on write and decrypt your data on retrieval
  + uses AES 256-bit symmetric keys

* Three ways to manage encryption keys:

* SSE with Amazon S3 Key Management (SSE-SE)
  + S3 manages keys and encryption

* SSE with Customer-Provided Keys (SSE-C)
  + user provides key to encrypt upon writing and a key for decrypting upon retrieval
  + Amazon discards keys immediately after encryption/decryption

* SSE with AWS Key Management Service (SSE-KMS)
  + S3 will encrypt data using keys managed by the user, but stored in the KMS.
  + KMS provides an audit trail so users can see who used a key to access which object and when

**Multi-Factor Authentication for Delete**

* An extra layer of protection requiring valid AWS credentials and the code from an approved MFA authentication device.

**Website Hosting**

You can host an entire static website on Amazon S3 for a low-cost, highly-available hosting solution that can scale automatically to meet traffic demands. S3 reliably serves traffic and handles unexpected peaks without worrying about scaling any infrastructure, through the use of load balancers.