Technical Guide: Introduction to the S3 API

All you need to know to get started with object storage and S3 API compatibility. An independently authored guide, written by Chris Evans and sponsored by Cloudian Inc. Release 1, April 2016.

INTRODUCTION

The public cloud and Amazon Web Services in particular have seen massive growth over the last few years. In April 2015, Amazon broke out the revenue figures of AWS for the first time, showing that the subsidiary was a $7.3 billion business with over 1 million active customers, accounting for 8% of Amazon's total revenue. An 8-K submission in April 2016 indicated that the business is expected to reach $10 billion in revenue during 2016. At the heart of AWS is S3, the Simple Storage Service, an online object store that is now ten years old and stores trillions of objects (latest figures published in 2013 showed 2 trillion objects, with the amount stored doubling each year).

S3 has been remarkably successful and is the foundation for many well known services such as Dropbox and Pinterest. Part of the reason for this success has been the flexibility of object stores compared to standard block and file protocols. From a user perspective, these protocols provide little in the way of capabilities for controlling how the data is stored and managed (they only support basic I/O commands like read, write, open and close).

S3 on the other hand is all about object level management and manipulation, with S3 you can describe how you want to store objects, encrypt them, present them (even as content for a website) and much more. Each object is validated during I/O operations unlike a file system (NFS/SMB) which does data integrity checking only at the entire file system level.

In addition to the management capabilities is the relative ease in which data can be stored in the system. The underlying storage infrastructure isn't exposed to the customer. Instead access is provided through a set of programming interfaces, commonly called the S3 API. It's through a combination of features with the simplicity and ubiquity of this API that S3 has been so successful.

WHAT IS S3?

The S3 API is an application programming interface that provides the capability to store, retrieve, list and delete objects (or binary files) in S3. When first released in 2006, the S3 API supported REST, SOAP and BitTorrent protocols as well as development through an SDK for common programming languages such as Java .NET, PHP and Ruby. Storing and retrieving data is remarkably simple; objects are grouped into logical containers called buckets and accessed through a flat hierarchy that simply references the object name, bucket name and the AWS region holding the data. When using the REST protocol, these pieces combine with a URL that provides a unique reference for the object. Actions on the object are executed with simple PUT and GET commands that encapsulate the data and response into the HTTP header and body. The SDKs provide the ability to obfuscate some of the details of using HTTP-based calls.

S3 features are reflected in the API and have matured over time to include:

- **Metadata** – this includes system metadata and additional information created by the user when the object is stored.
- **Multi-tenancy** – S3 is divided into many customers, each of which sees an isolated, secure view of their data.
- **Security & Policy** – access is controlled at the account, bucket and object level.
- **Lifecycle Management** – objects can be both versioned and managed across multiple tiers of storage over the object lifetime.
- **Atomic Updates** – objects are uploaded, updated or copied in a single transaction/instruction.
- **Search** – accounts and buckets can be searched with object-level granularity.
- **Logging** – all transactions can be logged within S3 itself.
- **Notifications** – changes to data in S3 can be used to generate alerts.
- **Replication** – data can be replicated between AWS locations.
- **Encryption** – data is encrypted in flight and can be optionally encrypted at rest using either system or user generated keys.
- **Billing** – service charges are based on capacity of data stored and data accessed.

Due to it's longevity in the market and maturity of features, the S3 API has arguably become the 'de facto' standard for object-based storage interfaces (with perhaps Swift a close 2nd in certain markets). In addition to their own proprietary APIs, pretty much every object storage vendor in the market place supports S3 in some form or other. Having support for S3 provides a number of benefits:

- **Standardisation** – users/customers that have already written for S3 can use an on-premises object store simply by changing the object location in the URL (assuming security configurations are consistent). All of their existing code should work with little or no modification.
- **Maturity** – S3 offers a wealth of features (as already discussed) that cover pretty much every feature needed in an object store. Obviously there are some gaps (including object locking, full consistency and bucket nesting), which could be implemented as a superset by object storage vendors.
- **Knowledge** – end users who are looking to deploy object stores don't have to go to the market and acquire specific platform skills. Instead they can use resources that are already familiar with S3, whether they are individuals or companies.

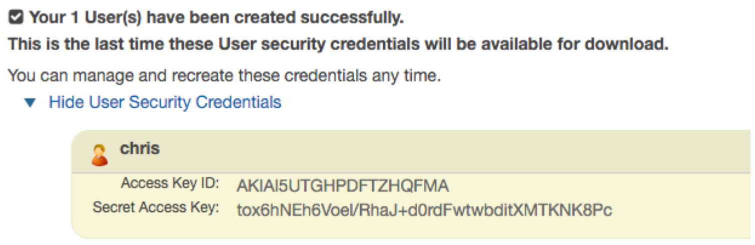
S3 AS THE DE-FACTO STANDARD

The current S3 API Developer Guide runs to 625 pages and has updates monthly, so vendors' claims of compatibility could mean many things. Both Eucalyptus and SwiftStack claim S3 API support, however looking at the specific feature support we see many gaps, especially around bucket-related features and object-based ACLs (rather an important security requirement). When establishing security credentials, AWS currently uses two versions for signing (v2 and v4), each of which provide slightly different functionality (such as being able to verify the identity of the requestor). We will go into the specifics of these features later.

As well as features/functionality, there are questions of compatibility in terms of performance and the way in which the S3 interface is implemented. Some vendors will translate S3 API calls into their own native API, rather than processing them directly. This can lead to performance issues where on-premises object stores don't behave and respond with the same error codes or response levels expected when using S3 directly.

S3 PRIMER

Before going further into the S3 API, it's worth spending a few moments looking at some of the terms used when talking about object storage. *Permissions*, providing access to data within S3 are granted to *buckets* (collections of data), *objects* (the actual content stored) or attributes of buckets and objects (like archiving policies). Permissions can be set through policies (either at the user or bucket level) or through specific *Access Control Lists* (ACLs). We'll come back and discuss ACLs in a bit more detail in a moment, however in general Object ACLs provide more granular permissions over individual objects, whereas bucket and user policies provide more general grouping capabilities. Bucket policies have to be used when enabling permission from one S3 account to another.



Your 1 User(s) have been created successfully.
This is the last time these User security credentials will be available for download.

You can manage and recreate these credentials any time.

▼ Hide User Security Credentials

chris

Access Key ID:	AKIAI5UTGHPDFTZHQFMA
Secret Access Key:	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

Users are individuals or services/applications that require access to S3 resources. Each user has a user ID/password that allows them to access features like the AWS console, however all S3 API requests are made using *access keys* generated by the user. These consist of an *access ID* and a *secret key* that are combined to make an access signature. The use of access keys in this way removes the need for the user to hard code user IDs/passwords into API requests that could be intercepted over the network.

"Data access in S3 is simple; objects are grouped in buckets and accessed using store, retrieve, delete and list commands."

Users can be placed into *groups*, to allow group-level permissions to be assigned, however groups are not hierarchical and have a flat structure. Permissions can also be grouped into *roles*, that can then be assigned to users or groups. This idea will be familiar to Active Directory users where specific tasks (like data operator) can be attributed to a specific user without necessarily providing access to the actual content.

Policies provide a mechanism to assign permissions either to users or resources and the choice of which method to use is down to the user themselves.

SECURITY AND ACCESS MANAGEMENT

When S3 was first introduced, there were only two main methods of accessing S3 content, either anonymously (in which case access was open to all) or using an authenticated request generated from access keys of the root AWS account owner. In September 2010, Amazon extended the security model to allow S3 resources to be controlled through the IAM feature – Identity and Access Management.

There are now five main access routes into S3 content; these are:

- *Anonymous* – resources can be made freely available for access by anyone, without the need to supply credentials.
- *Account Access Keys* – using a signature generated from an account access ID/secret key pair.
- *IAM User Access Keys* – using a signature generated from an IAM user's access ID/secret key pair.
- *Temporary Credentials* – using temporary credentials generated by an IAM user.
- Using *Multi-Factor Authentication* (MFA) with one of the above methods.

AWS best practices recommend not using the root account credentials. The root account has by default access to every resource within the account, whereas IAM users by default have no access. AWS also recommends creating one or more IAM administrator users and using policies/roles to manage the granting of permissions to buckets and objects within them.

"S3 provides a comprehensive security and identity management capability, that will be familiar to developers and users using directory services such as Active Directory"

Access Control Lists (ACLs) provide a mechanism for assigning permissions that was available before the introduction of IAM. Generally, AWS recommends using IAM policies however if ACLs are already in use, then there's no need to stop using them. There are specific times when ACLs need to be used over IAM policies, such as applying permissions to individual objects within a bucket. There are also limits on the size of bucket policies, which may also mean using ACLs to get around this restriction.

From the discussion so far, the terms and concepts of S3 security will be familiar to anyone who has worked with other directory services such as Microsoft's Active Directory or generic LDAP.

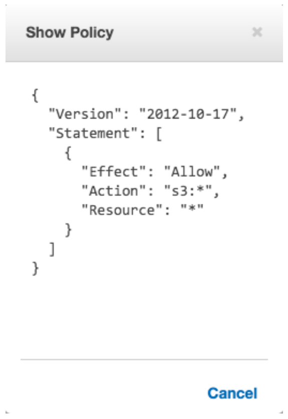
AUTHORISATION REQUESTS

Non-anonymous (authenticated) access requests made to S3 are signed using the access keys of the user. The keys themselves are not directly used in an access request, instead a signature is generated using a combination of the keys and part of the data in the request (such as the parameters of a query).

Two versions of signing are currently supported, known as Signature Version 2 and Signature Version 4. Version 4 is supported in all AWS regions; version 2 is supported in AWS regions that were created before 30 January 2014 and is therefore the legacy version. The changes from version 2 to version 4 are focused around improving the security of requests and making it more difficult to spoof or steal credentials. For example, a signing key generated from the user's secret key is used for signing messages rather than the secret key itself. Version 4 requests can also region specific and reference a regional rather than global endpoint.

PERMISSIONS AS CODE

Policy definitions are created using the Access Policy Language. This is a JSON-like coding structure that defines the policy and its attributes. Policy definitions don't have to be created by coding – they can be generated using the AWS Management Console, the CLI, API or PowerShell. However, the ability to set permissions with code does reflect one interesting aspect of AWS, and that's the ability to programmatically manage access to resources in S3. This makes it possible to extract definitions from one account, region or bucket and easily apply it to another. The ability to provide mobility to security definitions is a powerful tool and shouldn't be underestimated when evaluating object storage vendors claiming to provide 100% S3 API compatibility.



```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "s3:*",
      "Resource": "*"
    }
  ]
}
```

SUPPORTING EXTERNAL USERS

What happens if the users that you want to provide access to aren't based within AWS itself? This scenario is perfectly reasonable; imagine an object store used to retain medical records. From a compliance perspective it would be much more preferable to use some kind of external authorisation process that maps to an existing identity management system. S3 can enable that through temporary credentials that use Federation. The process allows integration with any OpenID or SAML 2.0 compatible authorisation process, such as Microsoft's ADFS (Active Directory Federation

Services). Another example could be providing access to data stored in AWS from a SharePoint environment.

BENEFITS OF MATURITY

So adding this all up, what benefits do we get from the way security has been developed in S3? Some specific advantages include:

"Using a standard object interface like S3 means developers don't have to learn multiple APIs in order to support multiple platforms. In most cases, code can be reused with little or no modification."

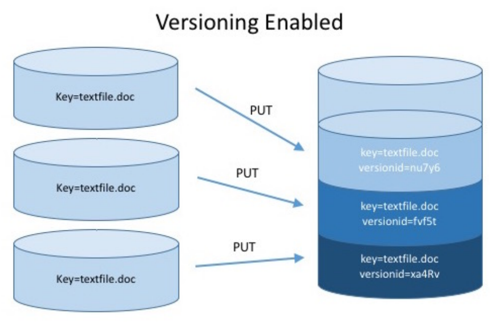
- **Granularity** – permissions can be defined onto items as small as an object.
- **Scale** – group policies and roles allow attributes to be defined across multiple users, objects and buckets.
- **Delegation** – permissions can be delegated out through roles, assigning access to actions as well as object policies.
- **Multi-tenancy** – permissions can be divided up within a single AWS account or can be federated or delegated to another AWS account or group of external users.
- **Audit** – policies can be applied to audit the access of users across an account.
- **Mobility** – both data and policy can be moved between AWS accounts, regions or even to external object store providers that fully support the S3 API.

ADVANCED FEATURES

Versioning allows multiple copies of an object to be stored within the same bucket. This can be used to provide a historical record/audit trail, or to protect against overwriting or deletion. The versioning feature is turned on at the bucket level, with a bucket being configured to one of three states; *unversioned*, *versioning enabled* or *versioning suspended*. All buckets start in the unversioned state and once enabled for a bucket, versioning cannot be turned off, only suspended.

Versions of an object are tracked with a VersionID, a characteristic of all objects. When versioning is not enabled, the VersionID of an object is stored as a null value. With versioning enabled, each PUT (update) request for an object stores the object with a unique VersionID, a randomly generated character string of up to 1024 bytes in length.

As objects are updated and stored with a unique VersionID, each copy of the object is retained in the bucket and can be accessed by name (in which case it retrieves the latest or current copy) or by name and VersionID. If an object is deleted from a bucket, GET (read) requests return an error, unless the VersionID is also included. A deleted object can be restored by issuing a COPY request and including a specific VersionID.


Versioning Enabled
Key=textfile.doc
Key=textfile.doc
Key=textfile.doc
PUT
PUT
PUT
key=textfile.doc versionid=nu7y6
key=textfile.doc versionid=fvf5t
key=textfile.doc versionid=xa4Rv

AWS offers no opportunity for cost reduction on the de-duplication of updated objects, so if only a small portion of an object is updated, each version occupies the full capacity – 5 copies of an object occupy 5x the space and cost, even when an object is deleted. This means implementing versioning can quickly become expensive in AWS and this provides object store vendors one opportunity to reduce costs while still offering S3 API compatibility. Object storage with de-duplication can represent a significant saving over AWS when versioning is in place.

TIERING

AWS offers multiple categories or classes of storage within S3. Each class offers the same level of durability (risk of data loss/corruption) but comes with varying levels of availability and access times. S3 tiers are: *Standard*, *Standard – Infrequent Access* (IA) and *Glacier*.

Data is moved between classes using Lifecycle management, which determines how data is managed from creation to destruction. Lifecycle policies determine how data within a bucket is managed between the multiple storage layers in S3. For example, a policy can be established to move all items in the *"archive/"* folder to Glacier after 365 days (known as a *Transition*) and then to delete the content after 10 years (an *Expiration*) from the date of creation.

In addition to the *Transition* and *Expiration* actions, S3 provides the ability to perform the same actions on versioned buckets with *NoncurrentVersionTransition* and *NoncurrentVersionExpiration* respectively. These features allow older versions of objects to be actioned in a different way to the current versions, so the inactive copies can be moved out more quickly to cheaper storage.

Obviously AWS has very rigid storage classes, however the S3 API does provide the ability to specify generic storage tiers as part of the migration action, as the target for migration is simply defined by the *StorageClass* parameter. This provides object storage vendors the capability to implement much richer data management policies that cater for many tiers of storage which can be defined outside of the API, without deviating from the standards of the API itself.

REPLICATION

The S3 API provides the ability to replicate data between AWS regions. AWS defines a region as a resilient group of geographically close data centres. S3 provides resiliency between data centres within a region, but there may be good reasons for protecting data between regions, for example, for compliance or to put data closer to an application and reduce latency overhead.

Replication is implemented at the bucket level and acts as an asynchronous (or eventually consistent) task. All data within a bucket, or a subset defined by prefix can be replicated, with each object, including Object ID, versions and metadata transferred to the target bucket. AWS only permits a one-to-one relationship between a source and target bucket and unless specified, the target bucket will be based on the same storage class as the source.

Replication, versioning and region resilience provide the capability to protect against most data loss situations, including user error, application data corruption and hardware failure. What these features don't provide is a guarantee that data is definitely safely stored in more than one location when an initial object PUT (save) request is issued. This may cause compliance issues for some businesses and so object store vendors have the ability to improve replication capabilities by implementing synchronous replication where appropriate for the right type of data.

LOGGING AND BILLING

S3 offers the capability to log all actions made through the S3 API. The data is stored in another AWS product called CloudTrail, that tracks all API calls, not just those performed with S3. Obviously the ability to trace activity is an important piece of any storage infrastructure and provides the basis for auditing and compliance. CloudTrail is offered by AWS at a very low cost but isn't free. In addition, data logged is stored in an S3 bucket owned by the customer and retaining large amounts of logging data will accrue costs over time.

Billing is managed at the account level, however tags can be associated with buckets to align charging to business units. The use of tags is entirely freeform and not interpreted in any way by S3, therefore the account owner is responsible for ensuring that billing maps to data owners within a single account.

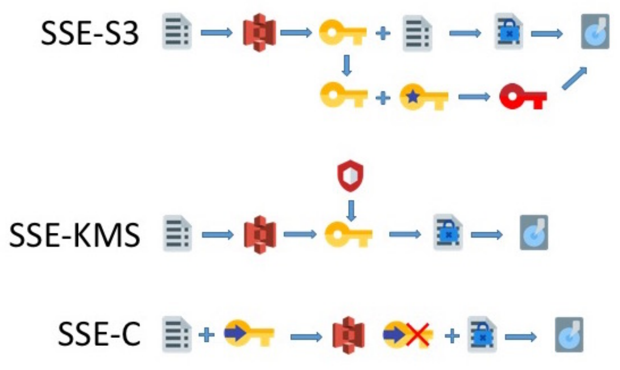
ENCRYPTION

Previously we discussed the security aspects of controlling access to data. S3 provides two other additional features to ensure data is protected for access only by authorised parties - data-at-rest encryption and encryption of data-in-flight.

Data at rest within S3 can be encrypted using either Server-side encryption (SSE) or Client-side encryption. Server-side encryption provides three options to the user; encryption using S3 managed keys (known as SSE-S3), encryption with keys stored in S3's key management service (KMS), known as SSE-KMS and encryption using customer-provided keys (SSE-C). In all three options AWS manages the encryption process, however each provides different levels of protection. SSE-KMS, for example provides more detailed audit tracking and permissions to control the use of keys. SSE-C takes key management away from AWS, putting responsibility into the hands of the user, including ensuring that keys are properly managed and stored.

"Encryption of data is an important feature of S3. Customers can choose to use AWS encryption and key management or supply their own encryption keys. Data can also be encrypted by the user before storing in S3."

It's important to note that when using the S3 REST API, the user is responsible for the encryption process when looking after their own key management (SSE-C). However, encryption using SSE-S3 or SSE-KMS can be specified in the request header on a PUT (save) request. AWS also provides the option to use an encryption client in the SDKs for Java, Ruby and .NET that take away some of the overhead of the encryption process.



Client-side encryption provides two options and requires the user to encrypt data before sending it to AWS. This can be achieved using AWS KMS managed keys, in which case the client only needs to use the KMS master key. Alternatively, data can be encrypted by the client before sending to S3.

Data in flight is protected using SSL when storing and retrieving content via the API. However, when using S3 as a website endpoint, HTTPS isn't supported and customer have to use CloudFront.

FEATURE RICHNESS

Many of the features discussed here address the requirements typically seen in storage environments and are delivered as services rather than specific parts of the infrastructure. In most cases, the S3 implementation provides basic levels of functionality. A good example of this is data protection, which is implied by not explicitly managed or configured by the end user. These kinds of features can be extended and improved by object storage vendors with their on-premises or cloud-based products. These enhancements can be added without directly affecting the S3 protocol definitions, while providing the storage administrators with additional functionality typically expected in enterprise environments.

CONCLUSIONS

There are some constraints around the implementation of S3 that can have an impact for users. These include:

- **Complexity** – for users new to AWS, the idea of security rules through code could prove a little challenging, however anyone experienced with LDAP and AD will be used to the idea of using code to access credentials.
- **Eventual Consistency** – IAM like every other AWS feature is eventually consistent across locations. This means it is theoretically possible to expose data as new security rules are put in place (hence AWS' best practice to assume default permission of no permissions).
- **Scale Limits** – there are some IAM limits in place; accounts are limited to 100 groups, 5000 users, 250 roles and users can only appear in 10 groups. This may prove an issue with some customers (especially those who don't want to use temporary credentials to program around the problem).

Some of the scale limits may be removable through discussions with Amazon, but naturally the ability to do this will be influenced by the size (and spend) of the customer. One of the features of Amazon's Web Services business is the low cost achieved by efficiency at scale. This is at odds with providing customers a bespoke service.

VALUE ADD

The relative simplicity of some storage-related S3 features means vendors supporting the S3 interface have the opportunity to deliver value into their object store product that provide S3 compatibility. Some of these include:

- **Data Optimisation** – AWS provides no cost reduction features based on using tools like de-duplication and compression. It's likely that Amazon are using these features behind the scenes to keep their overall prices low, however customers are used to on-premises storage savings that these features offer. At scale, S3 can be a comparatively expensive option.
- **Data Protection** – AWS provides limited data protection, mainly based around the idea of eventual consistency. Object store vendors can provide more enterprise-class data protection, either using traditional RAID or more likely with erasure coding techniques that scale better. More important here is the ability for the protection methods to be implemented at a more granular level, for instance on buckets or accounts.

VALIDATION

Value add services provide vendors with the option of delivering more comprehensive object storage services compared to AWS S3. However, the basis is still the API itself and compatibility is a key issue, especially with some of the more detailed aspects of IAM implementation. There aren't many tools available for testing S3 compatibility, however some open source code is available, such as S3 Compatibility Tests available on GitHub. CloudBerry Lab also has both freeware and paid versions of CloudBerry Explorer, which provides access to S3 and other object stores for the Windows platform.

FURTHER READING

More information can be found on S3 and Object Storage at the Architecting IT Blog (link). This ongoing series provides S3 and object storage information, including the results of testing against common object storage platforms. For more details on AWS S3, the Amazon website provides documentation and API reference material:

- Amazon Simple Storage Service Documentation
- Amazon Simple Storage Service API Reference
- Amazon Simple Storage Service Developer Guide

CLOUDIAN HYPERSTORE

This paper is brought to you in partnership with Cloudian, an object store vendor. HyperStore is an appliance or software-based object storage solution that offers 100% S3 compatibility while delivering additional features and functionality. For more details, check out the following links:

- HyperStore Appliance
- HyperStore Operating Environment

Further details on Cloudian's product offerings, information on S3 and other reports and solution briefs can be found on Cloudian's website (link).

THE AUTHOR

Chris M Evans has worked in the technology industry since 1987, starting as a systems programmer on the IBM mainframe platform, while retaining an interest in storage. After working abroad, he co-founded an Internet-based music distribution company during the .com era, returning to consultancy in the new millennium. In 2009 he co-founded Langton Blue Ltd (www.langtonblue.com), a boutique consultancy firm focused on delivering business benefit through efficient technology deployments. Chris writes a popular blog at http://blog.architecting.it, attends many conferences and invitation-only events and can be found providing regular industry contributions through Twitter (@chrismevans) and other social media outlets.

LANGTON BLUE & ARCHITECTING IT

For additional technical background or other advice on the use of storage in the enterprise, contact enquiries@langtonblue.com for more information. Langton Blue Ltd is hardware and software independent, working for the business value to the end customer. Contact us to discuss how we can help you transform your business through effective use of technology.

Website: www.langtonblue.com
Email: enquiries@langtonblue.com
Twitter: @langtonblue
Phone: (0) 330 220 0128

Langton Blue Ltd
133 Houndsditch
London
EC3A 7BX
United Kingdom