XRootD S3 Gateway for WLCG Storage

Andrew Hanushevsky[1*] and *Wei* Yang[1]

[1]SLAC National Accelerator Laboratory, 2575 Sand Hill Rd, Menlo Park USA

Abstract. The S3 Gateway is a server based application that provides a bridge between protocols and security used in the HEP community to S3 protocols and its associated security model. This allows the use of common copy tools based on the HEP security models to store or download data from S3 based storage. This storage can reside in a public or private cloud. This paper details the motivation for implementing such a service and how it can address certain problem when dealing with storage only accessible via S3 protocol.

1 Introduction

S3 protocol based storage is the de facto storage model for cloud computing. S3 storage essentially adopts a data depot where objects (i.e. files) are dumped into buckets to be later fully retrieved. As such, S3 is very compatible with HEP. workflows that use a copy-to-scratch data processing model (e.g. reconstruction). Using local S3 based storage (e.g. Ceph [1]) can provide significant cost savings and it potentially allows experiments to uniformly access public cloud resources where someone else manages the infrastructure.

However, S3 protocol along with its security model is not one that is common to HEP. Generalizing access to S3 storage would likely come at a high cost to implement an additional way to access storage across all workflows.

This dilemma was the motivation for the S3 Gateway – a mechanism to allow experiments to use existing infrastructure (i.e. protocols and security model) to access S3 protocol based resources.

1.1 S3 Protocol

The S3 protocol [2] was released by Amazon Web Services (AWS) in March 2006. It was launched as part of the Amazon S3 (Simple Storage Service) offering, which provided developers with a scalable and reliable cloud storage platform. At the time AWS was the largest cloud service provider. When Amazon introduced S3, along with the Elastic Compute Cloud (EC2), it quickly propelled it to be the dominant cloud provider. Consequently, S3 protocol became a de facto standard.

Today, practically every cloud storage provider supports at least the basic S3 protocol elements which include full file upload/download and byte range read capability.

*email : abh@slac.stanford.edu

Additionally, certain object oriented file systems also support S3 protocol in addition to their native APIs (e.g. Ceph).

S3 being so ubiquitous makes for a strong case for providing a bridge from current widely used HEP storage protocols (e.g. https and xroot) to providers offering an S3 interface.

1.1.1 S3 Security

S3 protocol authentication relies on Transport Layer Security (TLS) along with a unique user/key combination. While various cloud providers call the user-key combination different names, they all resolve to effectively the same thing. The general security model requires the presentation of an "Access Key" and a “Secret Key”. This model used by Amazon, MinIO [3] and Ceph, among others. Google cloud storage [4] provides access via what it calls an HMAC key but it is essentially a combination of an access and secret key. MinIO also calls its model username and password. For all practical purposes it is identical the previous models for arbitrary usernames.

This particular security model is neither compatible with x509 nor JWT Tokens (i.e. SciTokens [5] or WLCG Tokens [6]) which are widely used in the HEP community. Even for cloud providers that support x509, HEP x509 is not compatible because of HEP specific extension (i.e. VOMS [7]) and the CA’s used by HEP are generally not recognized by commercial cloud providers. Using S3 security on an experimental scale would require deploying a new security management infrastructure which would be yet another cost element in computing.

Finally, most Cloud Storage providers also support "Signed URL” [8] [9] which simply uses the secret key to sign a URL along with the access key and possible restrictions all specified in the CGI string suffix. The server needs to know the access key, normally provided as a CGI token, to find the associated secret key and validate the signature. Signed URL’s are meant to be issued by the owner of the data and not a central service.

2 S3 Gateway

The S3 Gateway is meant to provide a bridge between the HEP security model and the S3 security model. It is similar to a proxy server where the front-end of the proxy provides a certain security model and a set of protocols that are converted to another protocol and security model required to communicate with an origin server (e.g. an S3 data server). This is a common commercial solution to allow an existing infrastructure to interoperate with other systems. This is used to eliminate the cost of migrating to some other infrastructure, especially if the interoperation is not a significant part of the operation. This is the model adopted by the S3 Gateway.

2.1 Implementation

The S3 Gateway is implemented using the XRootD framework [10]. This framework allows building data services using a plug-in mechanism. That is, by interconnecting pre-existing modules one can configure a basic service that only needs one or two additional plug-ins to fully realize the desired service. In this case, the S3 Gateway implementation used a pre-existing proxy service configuration that only required writing an additional plug-in to provide an S3 protocol backend for the data flow.

The use of a proxy server as a template corresponded to the notion of protocol and security model transformation to a backend server (i.e. the origin server) that provides the desired resource; in this case data.

The proxy server communicates with the origin server using the XRootD client. The XRootD client itself can use any protocol to communicate with the requested server via a plug-in mechanism. While the default is to use xroot protocol the client is also capable of communicating using http protocol by loading a specialized plug-in, xrdcl-http [11]. This plug-in is widely used to provide caching proxy server (e.g. Xcache) a way to fetch data from http based origins. Since S3 protocol is http based most of the work is done already completed.

However, S3 protocol is an extension of http and the xrdcl-http plug-in required a relatively simple modification to enable S3 protocol. This was possible because xddcl-http plug-in relies on the CERN developed and supported Davix library [12]. This library provides http-based data services including basic S3 support.

2.1.1 Authentication and Authorization

The security scheme of the S3 Gateway is relatively straightforward and did not need additional development. The XRootD framework provides the necessary elements to authenticate a client using x509 (with or without VOMS), SciTokens, WLCG tokens, and Kerberos5 among other authentication protocols. The client's identity is then matched against configured capabilities using the default built-in authorization framework. If the client is not authorized for the requested operation against a file path, the request is rejected. Otherwise, the request proceeds to the proxy plug-in; which routes the request to the S3 endpoint. The S3 endpoint's authorization scheme is used by the Davix library with the appropriate username and password extracted from the environment that is preset during server start-up. In effect, the server executes the request on behalf of the client using its own credentials. This effectively provides the translation between HEP security and that required by the S3 endpoint.

2.1.2 Check summing

Most cloud providers automatically allow you to test the integrity of the copy using CRC or MD5, sometime both. Neither checksum is used by many experiments; instead opting for Adler32. In order to provide equivalent functionality, the data sent to an S3 instance would need to be read back and the Adler32 checksum computed. This would not only impact performance but also be costly as egress charges would be incurred.

Our solution to this problem was to implement a cloud function (e.g. Google Cloud Function [13] or AWS Lambda [14]) that computes the checksum in the cloud itself when a checksum is requested. The checksum is recorded with the S3 object and never needs to be recomputed. Since the execution is in the cloud all data transfer occurs in the cloud as well. The only charge is the CPU cost which is relatively trivial.

2.2 Performance

Ingress and egress performance was measured against Google Cloud Storage and Amazon Simple Storage Service instances. The gateway node was a 12 core Intel based CPU with 24 GB of RAM, a mirrored SSD drive, and a 100gb LAN/WAN interface. The local data source and sink for the tests was provided by a LAN connected Erasure Encoded file system capable of reaching I/O speeds of at least 16 GB/sec.

The ingress test (i.e. the gateway sends data to the S3 instance) was an FTS managed transfer consisting of 3,129 files of various sizes equalling 1.36 TB of data. The transfers consisted from 50 to 230 concurrent streams over a 35 minute period, All transfers succeeded. The average transfer rate was approximately 312 MB/sec with peaks approaching 2 GB/sec. This output speeds closely matched the input speed of the source file system indicating that the gateway was not a bottleneck. The performance was essentially the same for Google and Amazon.

The egress test (i.e. the gateway reads data from the S3 instance) differed only in the amount of data read in order to minimize egress costs. In this case the FTS managed transfer using 312 files totalling 136 GB. Files were transferred using from 50 to 230 concurrent streams. All transfers succeeded. The average transfer rate was approximately 90 MB/sec with peaks approaching 390 MB/sec. We could not account for the performance discrepancy between the Ingress and egress tests. It could have been that the ingress test did not checksum the data while the egress test did. It could also be the case that some artificial rate limiting was in effect for the egress test. More study is needed to resolve this.

Even with the uneven ingress and egress tests, we concluded that the S3 Gateway would not be a bottleneck for typical workloads.

2.3 Future work

While Davix supports HTTP and S3 data transfers, it does not support PUT streaming. That is, streaming file uploads must first create a local file which is then transferred to the S3 end-point. This not only requires additional resources but also makes gateway management more complex. It also makes deploying the gateway in the public cloud more problematic since a local disk storage charge would apply. This can be costly when the gateway is heavily used.

We are in the process of enhancing Davix to use multi-part uploads to support direct streaming. This would eliminate a local staging disk requirement.

3 Conclusion

The S3 Gateway is an enabling technology that broadens storage resource access without incurring administrative overhead to manage the secure access to those resources.

For instance, it becomes much easier for an arbitrary group of researchers to pool their resources to leverage a cost-effective contract from a public storage service provider and allow secure access to the storage pool with a minimum amount of administrative overhead.

Since data access is compliant to existing access models, client overhead is equally minimized. Furthermore, because the S3-Gateway provides compatible access the storage resources can be easily integrated into HEP federated storage systems.

References

1. *Welcome to Ceph*, Accessed November 2023, https://docs.ceph.com/
2. *Amazon Simple Storage Service Documentation*, Accessed November 2023, https://docs.aws.amazon.com/s3/
3. *MinIO Object Storage for Kubernetes*, Accessed November 2023, https://docs.min.io/
4. *Google Cloud Storage*, Accessed November 2023, https://cloud.google.com/storage/docs
5. *GitHub: scitokens*s, Accessed November 2023, https://github.com/scitokens/scitokens
6. *Token-based AuthN/Z for WLCG*, Accessed November 2023, https://wlcg-authz-wg.github.io/wlcg-authz-docs/token-based-authorization/

7. *Virtual Organization Membership Service*, Accessed November 2023, https://wiki.italiangrid.it/twiki/bin/view/VOMS/WebHome
8. *Signed URLs*, Accessed November 2023, https://cloud.google.com/storage/docs/access-control/signed-urls
9. *Signed URLs*, Accessed November 2023, https://docs.aws.amazon.com/AmazonS3/latest/userguide/ShareObjectPreSignedURL.html
10. *Welcome to the XRootD webpage*, Accessed November 2023, http://xrootd.org
11. *GitHub: xrdcl-http*, Accessed November 2023, https://github.com/xrootd/xrootd/tree/master/src/XrdClHttp
12. *GitHub: Davix*, Accessed November 2023, https://github.com/cern-fts/davix
13. *Cloud Functions documentation*, Accessed November 2023, https://cloud.google.com/functions/docs
14. *Serverless Function, FaaS Serverless - AWS Lambda - AWS*, Accessed November 2023, https://aws.amazon.com/lambda/