iRODS Client: AWS Lambda Function for S3 1.0

Terrell Russell
Renaissance Computing
Institute (RENCI)
UNC Chapel Hill
unc@terrellrussell.com

ABSTRACT

Under development for less than six months, this new AWS Lambda[1] function updates an iRODS Catalog with events occurring in one or more S3 buckets. Files created, renamed, or deleted in S3 appear quickly in iRODS.

The following AWS configurations are supported with the 1.0 release:

- S3 -> Lambda -> iRODS

- S3 -> SNS -> Lambda -> iRODS

- S3 -> SQS -> Lambda -> iRODS

Keywords

iRODS, S3, AWS, lambda, data management

INTRODUCTION

As the iRODS Server has continued to improve its reliability, stability, and speed, additional functionality is being demanded of the clients that connect to the iRODS Server. One of the additional bits of functionality requested over the past few years has been a better ”out-of-the-box” onboarding, or ingest, experience for existing data.

In 2018, the Automated Ingest Capability[2], a Python-based client provided and supported by the iRODS Consortium, was introduced. It provided a parallel, distributed solution for quickly scanning and re-scanning an existing filesystem (and later, S3[3]) and registering or PUT-ting files into iRODS. It works very well, but it requires a bit of configuration and computing overhead, and cannot see the ”negative space”, or when a file is removed from the filesystem.

This Lambda function represents an alternative approach. It reduces the complexity involved in keeping up with a changing filesystem and improves the coverage of removed files.

DESIGN GOALS

The design goals of this new approach to getting files registered into iRODS were three-fold. First, it needed to play nicely with the universe of tools that already know how to write to S3 directly. Second, it must allow those updates within the S3 namespace to smoothly flow into the iRODS Catalog. And third, it would trigger iRODS automated data management due to crossing the policy boundary (the iRODS API).

A tool that could provide all three of these goals would solve a number of use cases the Consortium is seeing in the community. Everyone who begins to consider iRODS as a solution already has a lot of data in existing systems. Many of these existing systems are now in the ”cloud”, most commonly in Amazon S3 buckets.

iRODS UGM 2020 June 9-12, 2020, Virtual

Considerations

In looking at other clients and existing work, we realized that Amazon's Lambda service could provide the "place" to run the new client code. Lambda can run Python code, and iRODS already provides a well-tested and robust Python client library (python-irodsclient[4]).

Success with this approach would be defined as near-real-time, asynchronous updates to the iRODS Catalog. We were interested not only in visibility of create and rename operations, but also delete operations.

IMPLEMENTATION

As a single Python file, the implementation is straightforward and self-contained. The `lambda_handler` function captures variables from its configuration and environment, parses the event coming from S3, and then selects whether to perform a registration into the iRODS Catalog or to perform a delete from the iRODS Catalog based on the parsed S3 event.

The S3 Events that trigger an iRODS registration are:

- `ObjectCreated:Put`
- `ObjectCreated:Copy`
- `ObjectCreated:CompleteMultipartUpload`

The S3 Events that trigger an iRODS deletion are:

- `ObjectRemoved:Delete`
- `ObjectRemoved:DeleteMarkerCreated`

Rename operations sent to S3 are decoupled into independent `ObjectRemoved` and `ObjectCreated` events. This is discussed later in the Limitations section.

At this time, each firing of the Lambda function reacts to a single S3 event.

CONFIGURATION OPTIONS

Inputs to the Lambda function come from three places.

First, the S3 Event payload itself provides the event type, bucket name, key name, and for `ObjectCreated` events, the file size. This is not configuration, but supplies important context for the functioning of the Lambda.

Second, the Python runtime environment provides configuration information to the Lambda including `IRODS_COLLECTION_PREFIX`, `IRODS_ENVIRONMENT_SSM_PARAMETER_NAME`, and optionally, `IRODS_MULTIBUCKET_SUFFIX`. These must be defined by the owner of the Lambda.

And third, the iRODS connection information required by the Lambda is stored in the `AWS Systems Manager > Parameter Store`
as a JSON object of the type `SecureString` under the name that matches the environment variable `IRODS_ENVIRONMENT_SSM_PARAMETER_NAME`. This information must be defined and protected separately because it contains the password of the configured iRODS user.

The `SecureString` is expected to have the form:

```
{
    "irods_default_resource": "s3Resc",
    "irods_host": "irods.example.org",
    "irods_password": "rods",
    "irods_port": 1247,
    "irods_user_name": "rods",
    "irods_zone_name": "tempZone"
}
```

iRODS is assumed to have its associated S3 Storage Resource(s) configured with `HOST_MODE=cacheless_attached`. There should be no compound resources involved in the relevant resource hierarchy.

The Lambda function must be configured to trigger on all `ObjectCreated` and `ObjectRemoved` events for a connected S3 bucket. Defining and maintaining the AWS Policies can vary widely in practice and is beyond the scope of this document.

A well-configured Lambda function will update the iRODS Catalog in near-real-time as events flow from the S3 bucket(s).

The following AWS configurations are supported at this time:

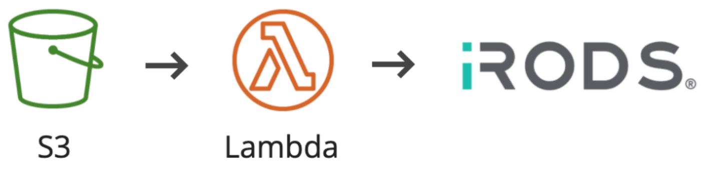

Figure 1. S3 to Lambda to iRODS - Direct Connection

Figure 1 shows the simplest and most straightforward configuration. Events from S3 flow directly to the Lambda function and triggered immediately.

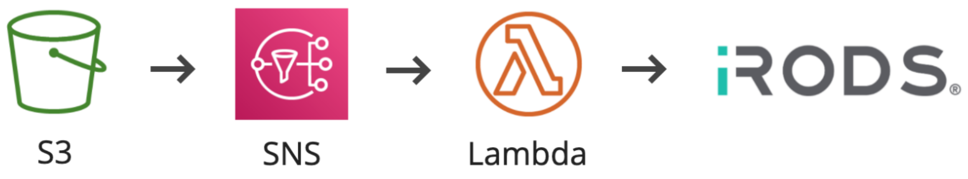

Figure 2. S3 to Simple Notification Service (SNS) to Lambda to iRODS

Figure 2 shows how the Lambda can be triggered as one of many services being notified by S3 Events coming out of a bucket as the Simple Notification Service (SNS) can be configured to send its Events to multiple endpoints. This is useful for adding Lambda to an existing ecosystem of cloud microservices and APIs.

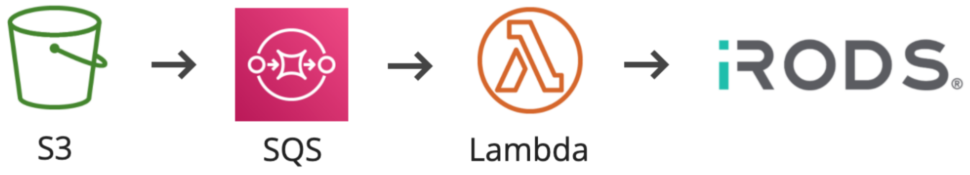

Figure 3. S3 to Simple Queue Service (SQS) to Lambda to iRODS

Figure 3 shows how the Lambda can be triggered in a more robust store-and-forward configuration. The Simple Queue Service (SQS) provides retry functionality for failed operations as well as provides the ability to operate on multiple events at once to save operational costs.

SSL Support

SSL to iRODS is supported by placing a certificate in a relative path within the Lambda package.

If the Lambda needs to be configured to connect with an SSL-enabled iRODS Zone, the following additional keys need to be included in the environment in the Parameter Store:

```
{
    "irods_client_server_negotiation": "request_server_negotiation",
    "irods_client_server_policy": "CS_NEG_REQUIRE",
    "irods_encryption_algorithm": "AES-256-CBC",
    "irods_encryption_key_size": 32,
    "irods_encryption_num_hash_rounds": 16,
    "irods_encryption_salt_size": 8,
    "irods_ssl_verify_server": "cert",
    "irods_ssl_ca_certificate_file": "irods.crt"
}
```

The `irods_ssl_ca_certificate_file` is a relative path to a certificate file (or certificate chain file) within the Lambda package.

Multi-Bucket Support

This Lambda function can also be configured to receive events from multiple sources at the same time.

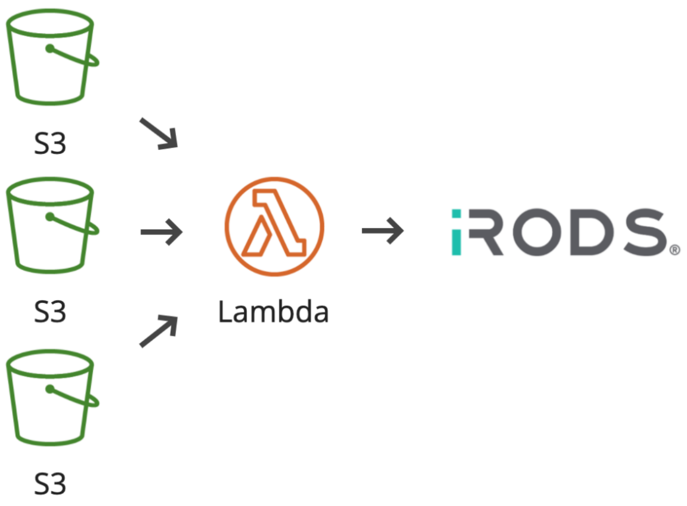

Figure 4. Multi-bucket S3 to Lambda to iRODS

Figure 4 shows how the Lambda can be triggered by an event in multiple different S3 buckets. This is made possible by having the target iRODS Resource be determined at runtime. This configuration reduces the number of Lambda functions that need to be deployed and maintained within an organization.

If the `irods_default_resource` is NOT defined in the environment in the Parameter Store, then the Lambda function will derive the name of a target iRODS Resource.

By default, the Lambda function will append `_s3` to the incoming bucket name found in the S3 event.

For example, if the incoming event comes from bucket `example_bucket`, then the iRODS resource that would be targeted would be `example_bucket_s3`.

However, if `IRODS_MULTIBUCKET_SUFFIX` is defined as `-S3Resc`, the the iRODS resource that would be targeted would be `example_bucket-S3Resc`.

LIMITATIONS AND FUTURE WORK

This first release of the Lambda function meets all the design goals defined earlier. However, some limitations have been documented.

The first is that S3 is decoupled from the Lambda itself. A rename operation sent to S3 is decomposed into a create event and a delete event. Because of this decoupling and without the ability to confirm that two events are related to one another, iRODS must treat this as a new data object. This means any metadata AVUs associated with the now-deleted data object is lost.

It is possible this could be remedied with a comparison of full checksums of the object to be deleted and any new incoming objects within a certain window of time, but this would be slow, and therefore, more expensive to operate.

The second limitation is that SQS configuration is limited to `batch_size = 1`. Operating on more than one message at a time would increase performance (event throughput) and reduce the cost of running this Lambda at AWS. However, it is unclear how the Lambda could signal partial success at this time. What happens to the failed events?

It is possible this could be remedied with an atomic database operations API at the iRODS Server level. If any events fail to be processed, the entire batch could be safely returned to the SQS queue with no side effects on the iRODS Catalog.

SUMMARY

The AWS Lambda function for S3, a new iRODS client written in Python, has been developed relatively rapidly and is already in multiple production deployments. It handles both creates and deletes within an S3 bucket and updates the associated iRODS Catalog in near-real-time.

I would like to thank Bristol Myers Squibb for providing the pre-release testing environment and conformance test scenarios that drove much of this initial work.

REFERENCES

[1] Russell, Terrell: iRODS Client AWS Lambda S3 (2020). `https://github.com/irods/irods_client_aws_lambda_s3`

[2] Xu, Hao; King, Alan; Russell, Terrell; Coposky, Jason; de Torcy, Antoine; iRODS Capability: Automated Ingest (2018) `https://irods.org/uploads/2018/Xu-RENCI-Automated_Ingest-paper.pdf`

[3] Amazon S3 (2006) `https://en.wikipedia.org/wiki/Amazon_S3`

[4] Python iRODS Client `https://github.com/irods/python-irodsclient`