iRODS S3 API: Presenting iRODS as S3

Terrell Russell, Ph.D.
@terrellrussell
Executive Director, iRODS Consortium

June 13-16, 2023
iRODS User Group Meeting 2023
Chapel Hill, NC

Overview

- History
  - Desire / Justification
  - Research
- Status / Implementation
  - Architecture
  - Configuration
- Next Steps

History - Desire / Justification

- Everyone just wants to talk to an S3 endpoint
  - Easy
  - Comprehensible
  - Lots of existing clients
  - Decoupled authentication
- Aligns with our Protocol Plumbing efforts
  - Accessible / Approachable
  - Maintainable

History - Desire / Justification

- Present iRODS as the S3 protocol
- Reuse - don't reinvent
- Load Balancer friendly
- Maintainable

History - Desire / Justification

- iRODS S3 Working Group
  - https://github.com/irods-contrib/irods_working_group_s3
  - Initial email 20210731
  - Four options

History - Desire / Justification

1. Update and maintain https://github.com/bioteam/minio-irods-gateway

This minio-based front end already exists and has been demonstrated, but has not been updated since its debut. I do not know the extent that it has been used in production work, perhaps John can talk about that. This uses the GoRODS client library, which will continue to need to be updated to wrap the latest iRODS C API as new releases of the server come out. Presumably, BioTeam / John would continue to own/maintain GoRODS and the minio-irods-gateway.

2. minio-irods-gateway converts to use https://github.com/cyverse/go-irodsclient

Illyoung has produced and is actively developing a pure Go iRODS client library. This new library could be 'swapped' for the GoRODS calls in the minio-irods-gateway. Presumably, then Arizona / Illyoung would own/maintain a fork of the minio-irods-gateway.

3. Add irods/gateway-irods.go to upstream https://github.com/minio/minio/tree/master/cmd/gateway

Someone / all of us would port / implement the same work from Option 2 (convert to pure Go), but work with the minio community to get iRODS to be an officially supported gateway for the MinIO server directly.

4. New C++ implementation - https://github.com/irods/irods_client_s3_cpp

The iRODS Consortium would work to implement the S3 specification directly with a new C++ client. This could be more performant (in the long run), but requires the most work and answers to open questions.

"I am leaning towards **Option 3** as the best option both from a cost/benefit perspective, as well as exposure to a larger community and confidence that 'it just works'."

History - Research

- Investigated the 4 options, in 4 phases
    - Options 1-3 … MinIO, Go, TicketBooth
    - Option 4 … C++ proof of concept
    - Multipart
    - Handshake / Authentication

History - Research - Phase 1

- **Option 1** - MinIO with GoRODS (wrapping C)
  - Limited, not maintainable (**Jul 2021**)
- **Option 2** - MinIO with pure go-irodsclient
  - Needed to add (anonymous) ticket functionality
    - Implemented TicketBooth/BoxOffice (**Oct 2021**)
      - But would need admin credentials
      - Might as well just use C++ REST API (**Nov 2021**)
  - Lacks multi-user functionality (**Feb 2022**)
    - Auth code is in MinIO core - gateway code fires 'too late'
- **Option 3** - Get work into upstream MinIO
  - MinIO announced deprecation of gateway (**May 2022**)
    - Too hard / not worth supporting 'legacy' POSIX

History - Research - Phase 2

- **Option 4** - New C++ Implementation
  - Removes dependency on other codebase(s)
  - 1 collection -> 1 bucket
  - Framework selection (**Aug 2022**)
    - Pistache
    - Oat++
    - Drogon
    - Boost.Beast (**Nov 2022**)
  - Initial endpoints working (**Jan 2023**)
    - User mapping
    - Bucket mapping

History - Research - Phase 2 - Alternate Illyoung Universes

- Add S3 protocol support to SFTPGo (**Aug 2022**)
- Searching for existing S3 server in Go (**Sept 2022**)
- Add iRODS backend to Zenko (**Oct 2022**)
- Add JuiceFS frontend to iRODS (**Nov 2022**)
- Add JuiceFS frontend to SFTPGo (**Nov 2022**)
- GarageHQ frontend to iRODS (**Jan 2023**)
- In-memory IBM s3mem-go as inspiration (**Mar 2023**)

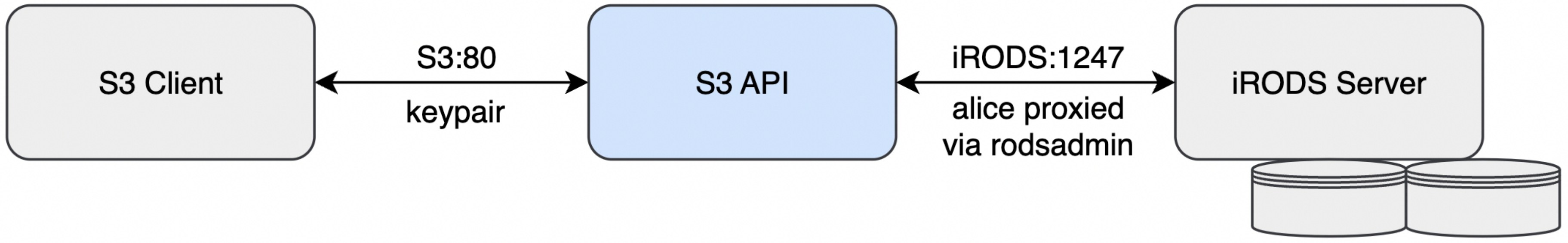
History - Research - Phase 3 (**Feb-Mar 2023**)

- **Multipart Options**
  - **a. Multiobject** - write all parts individually to iRODS, then complete triggers copy/concatenate/whatever
    - pro - relatively simple
    - con - lots of extra policy, could trigger replication to multiple continents (just a config option)
      - requires API plugin for concatenate()
  - **b. Store-and-forward** - write it all down in the bridge, then send it to iRODS
    - pro - simple, no extra policy
    - con - slow/delayed, need POTENTIALLY HUGE disk
  - **c. Efficient store-and-forward** - write down / hold non-contiguous parts in bridge - send contiguous parts to iRODS when ready
    - pro - elegant, single write
    - con - more complexity, need biggish disk
    - maybe off the table because a client can re-send the same numbered part and it should overwrite the earlier same part
      - OR... new thread! offset, overwrite, who cares, same size, magic/perfect, just works, don't look at me...
  - **d. Store-and-register** - write it all down where iRODS can see it, then just register it in iRODS
    - pro - simple, fastest
    - con - just reg policy?, adds dependency on co-visibility of bridge and iRODS
      - cannot continue on failure (incomplete writes)
        - iRODS doesn't know what happened
        - Client has no way to recover

History - Research - Phase 4 (**June 2023**)

- Saved Multipart for later
- Demo working with simple boto
- HMAC / Signature not working for other client interactions

Status / Implementation - Architecture



- Single binary
- Requires rodsadmin credentials
- Two configuration files
  - irods_environment.json
  - config.json

Status / Implementation - Architecture

- Implemented Endpoints
  - CopyObject
  - DeleteObject
  - GetObject
  - HeadObject
  - ListObjectsV2
  - PutObject
- Under Discussion
  - CompleteMultipartUpload
  - CreateMultipartUpload
  - UploadPart
- Maybe Later?
  - UploadPartCopy
  - ListObjects
  - DeleteObjects
  - GetObjectAcl
  - PutObjectAcl
  - GetObjectTagging
  - PutObjectTagging

Status / Implementation - Configuration

Not Yet Real...

```
{
    // Defines S3 options that affect how the
    // client-facing component of the server behaves.
    "s3_server": {
        // ...
    },

    // Defines iRODS connection information.
    "irods_client": {
        // ...
    }
}
```

Status / Implementation - Configuration - s3_server

```
"s3_server": {
    "host": "0.0.0.0",
    "port": 80,

    "log_level": "warn",

    "bucket_mapping": {
        # local / static (JSON)
        # external / dynamic - iRODS or third party API
    },

    "user_mapping": {
        # local / static (JSON)
        # external / dynamic - iRODS or third party API
    },

    "threads": 3
    }
}
```

Status / Implementation - Configuration - irods_client

```
"irods_client": {
    "host": "<string>",
    "port": 1247,
    "zone": "<zone>",

    "proxy_rodsadmin": {
        "username": "<string>",
        "password": "<string>"
    },

    "connection_pool": {
        "size": 6,
        "refresh_timeout_in_seconds": 600
    },
}
```

Next Steps

- Configuration consolidation
- HMAC cleanup / Confirmation
- Packaging / Initial release
- Testing
  - Use as S3 resource
  - Stress / Performance
- Additional plugins
  - Other bucket mappings
  - Other user mappings

Questions?

Thank you.