CYVERSE®

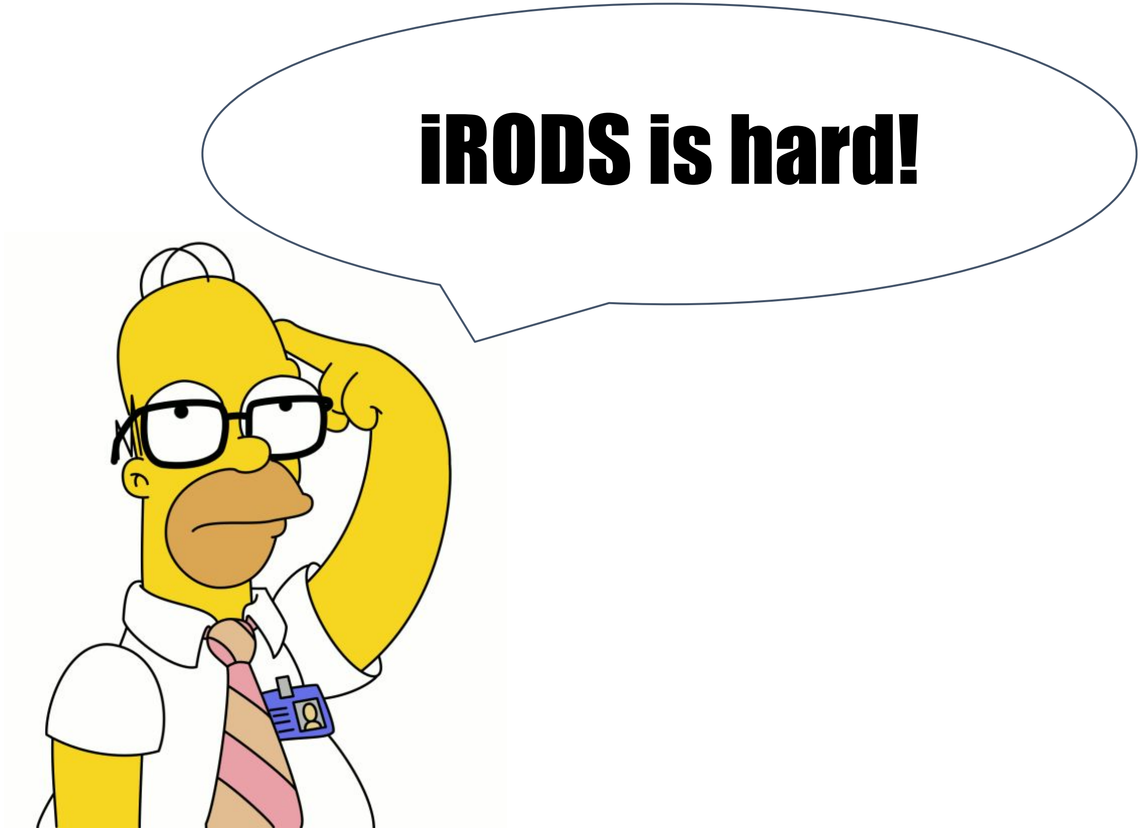
Transforming Science Through Data-Driven Discovery

10 Years at CyVerse: Some iRODS Administration ~~Practices~~ Solutions

Tony Edgin

iRODS UGM 2023

CyVerse is supported by the National Science Foundation under Grant Nos. DBI-0735191, DBI-1265383, and DBI-1743442


iRODS is hard!

iRODS is Powerful, but Difficult to Master

- Federation
- Distribution
- Resource Composition
- Plugin architecture that supports custom plugins

  microservices, resources, authentication, networking, databases, rule engines, APIs

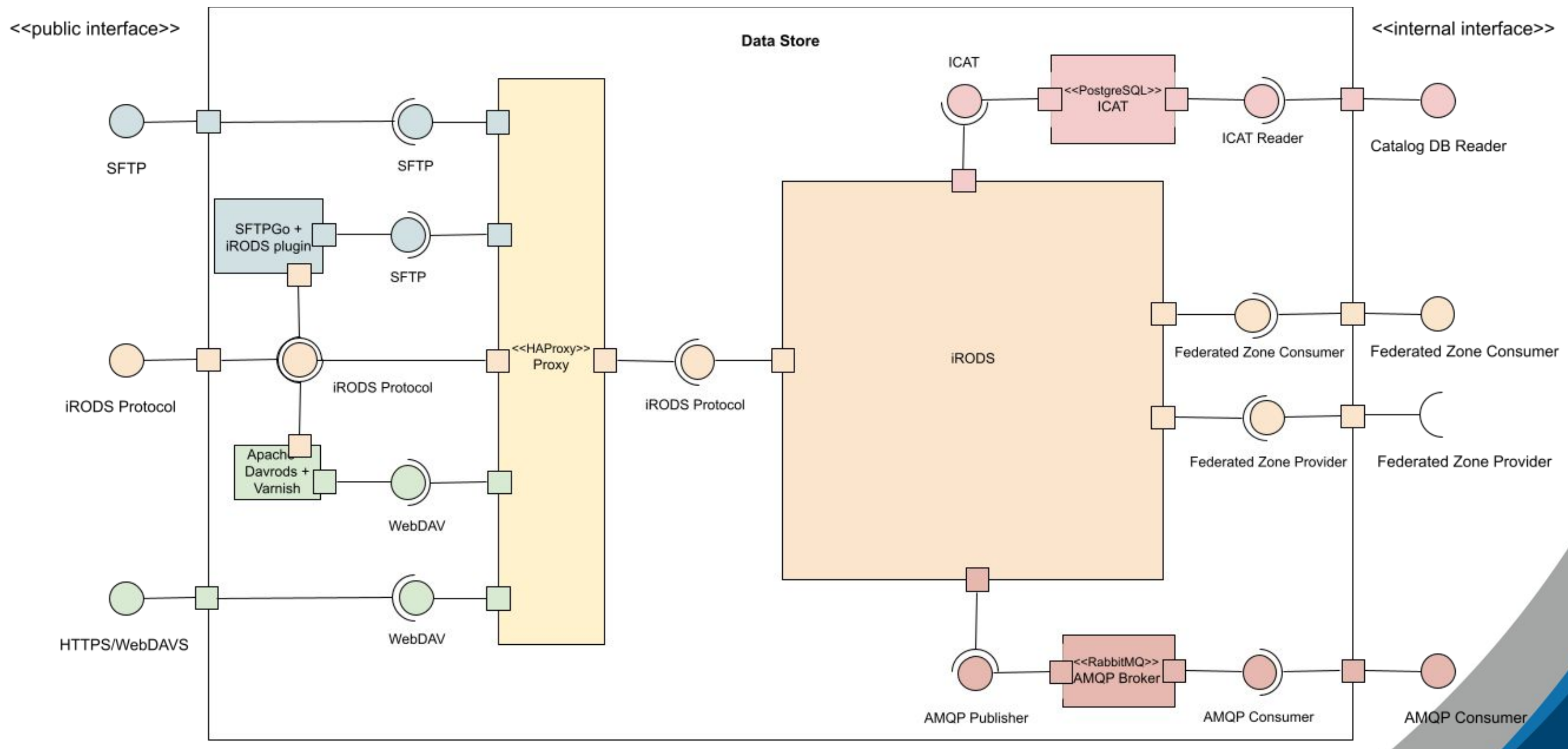
- 1000+ PEPs for attaching policy logic
- Sparse, and incomplete documentation
- Few online tutorials

Sharing knowledge within the community is important for using it effectively.

Agenda

1. Very brief introduction to CyVerse Data Store
2. Go over a few of our iRODS solutions
    a. Creating service accounts
    b. Determining data residency
    c. Downloading large sets of small files
3. Propose an iRODS administration interest group

CyVerse Data Store Built on iRODS



CyVerse's iRODS Zone by the Numbers

- 1 catalog provider and 42 resource servers
- 1 primary and 1 standby PostgreSQL server for ICAT
- 4300 lines of iRODS rule language based policy
  - Global policies
    checksumming, permission assignment, data residency, asynchronous replication, user storage usage tracking, storage freespace tracking, trash removal, service accounts, event publishing
  - 4 service specific rule bases
  - 7 project specific rule bases
- 120,000 users
- 500 million data objects consuming 12 PiB of storage
- 16 TiB uploaded and 440 TiB downloaded monthly
- 40 concurrent sessions on average

Service Accounts

Problem

Need to manage services that connect to iRODS differently than people

- A person has metadata we track, e.g., employer, ethnicity, etc.
- A service performs a task on behalf of a person, i.e., lacks agency
- A person can own data, a service does not (excluding state data)

Service Accounts

iRODS custom account types

Account types are managed through the token system in *user_type* namespace

New account type can be added through admin interface

```
iadmin at user_type ACCOUNT-TYPE '' DESCRIPTION
```

by convention, second value holds type's description

Rule logic is used to differentiate allowed behavior

Limitations:

- Cannot be given admin privileges, e.g., *cannot proxy for a user*
- Cannot create one that acts as a group

Service Accounts

CyVerse service account type

```
iadmin at user_type service '' 'iRODS service account'
```

```
iadmin mkuser SERVICE-NAME service
```

Policies

- No home or trash collection
- Does not belong to public group
- Data object created by service, owned by user invoking service (*planned*)

Custom account types are usable in iRODS 4.2.8, but not 4.2.12. `iadmin mkuser` *does not recognize them.*

Service Accounts

Weaknesses of solution

Service account cannot proxy for a user

- Prevents track user actions for provenance
- User must explicitly grant service access to data
- Ownership of data generated by service not well defined

iRODS feature request

Have a built-in service account type able to proxy for a user

Or

Have a PEP for controlling an account type's ability to proxy

Data Residency

Problem

We have general purpose, project-specific, and service-specific storage requirements.

- Some projects provide own storage server dedicated to their data
- One GPU-based service requires colocated storage, project can subscribe to service to have data hosted on the service storage
- Remaining data needs copies at both UArizona and TACC
  - Data proximity to compute – users can run analyses on data at either site
  - Data recoverability

Data Residency

Resource organization

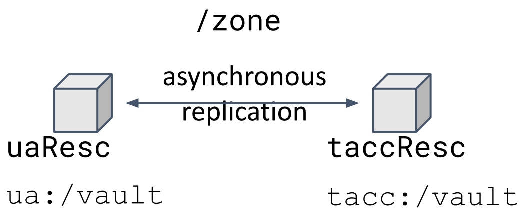
General storage

- Bidirectional asynchronous replication between sites
  - Synchronous degraded transfer too much
  - Acceptable risk of data loss
- Root coordinating resource for each site

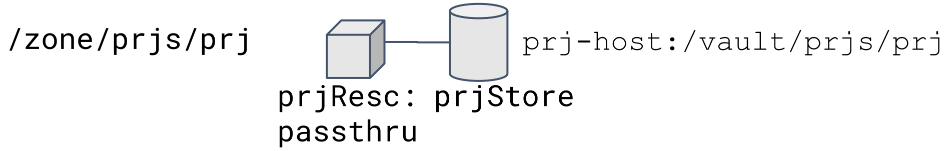
Each project and service

- Storage server configured as resource server
- Single storage resource
- passthru root resource

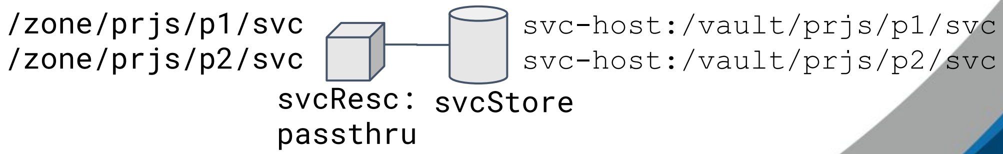
General resources



Resource for project *prj*



Resource for service *svc* hosting for projects *p1* and *p2*



Data Residency

Choosing where to store new data object (*wrong approach*)

Condition the rule logic on embedded project paths

- Lots of repeated code
- Order dependent
- Adding/removing project ⟶ redeploy rules

```
def acSetRescSchemeForCreate(_, cb, rei):
  obj = session_vars.get_map(rei).get(
    'data_object')
  path = obj.get('object_path')

  # choose general resc based on irods server
  resc = _choose_general_resc(cb)

  # projects
  if path.startswith('/zone/projects/prj_1/'):
    resc = 'prj1Resc'
   ⋮
  elif path.startswith(
    '/zone/projects/prj_n/'
  ):
    resc = 'prjNRes'

  # service
  elif (
    path.startswith('/zone/projects/1/svc/') or
     ⋮
    path.startswith('/zone/projects/m/svc/')
  ):
    resc = 'svcResc'

  res = cb.msiSetDefaultResc(resc, 'forced')
  return res['code']
```

Data Residency

Choosing where to store new data object (*better approach*)

Attach AVU to resource to associate project path

```
imeta add -R RESC \
  hosted-collection COLL
```

Rule logic use AVUs to determine resource

- No repeated code
- Order independent
- Adding/removing project ⟶ modify AVUs, not rule logic
  "*Configuration, not code*" - Jason Coposky

```
def acSetRescSchemeForCreate(_, cb, rei):
  obj = session_vars.get_map(rei).get(
    'data_object')
  path = obj.get('object_path')

  # choose general resc based on irods server
  resc = _choose_general_resc( cb)

  cols = (
    'ORDER_DESC(META_RESC_ATTR_VALUE)' ,
    'RESC_NAME')
  cond = (
    "META_RESC_ATTR_NAME ="
    " 'hosted-collection'")
  for rec in genquery.Query(cb, cols, cond):
    if path.startswith(rec[0]):
      resc = rec[1]
      break

  res = cb.msiSetDefaultResc(resc, 'forced')
  return res['code']
```

Data Residency

Choosing where to replicate data object

Replication resource determined by primary resource

Attach AVU to primary resource to associate replica resource

```
imeta set -R PRIMARY-RESC replica-resource REPLICA-RESC
```

Rule logic use AVUs to determine resource

- Attached to PEP `acSetRescSchemeForRepl`
- Similar to logic for choosing primary replica
- If primary has no `replica-resource` AVU, data object not replicated

Downloading Large Set of Small Files

Problem

A user needs to download multiple TB data set consisting of 100k+ files

Downloading set of small files takes a lot longer than downloading single large with same volume.

E.g., for me, downloading 1 1000 MiB file takes only 13 s, but downloading 1000 1 MiB files takes 471 s. 36x longer!

This is a general data transfer problem, not just iRODS

Downloading Large Set of Small Files

Common solutions

- Use tar pipe

  ```
  ssh remote tar --create dataset/ | tar --extract
  ```

  Weakness

  Downloading a single very large file can be problematic, network issues, cache overruns, etc.

- Use tar + split

  ```
  ssh remote 'tar --create dataset/ | split --bytes=100G - dataset.'
  scp 'remote:dataset.*' .
  cat dataset.* | tar --extract
  ```

Downloading Large Set of Small Files

iRODS solution

Use ibun

```
ibun -c -D tar dataset.tar dataset/
iget dataset.tar - | tar --extract
```

Weaknesses

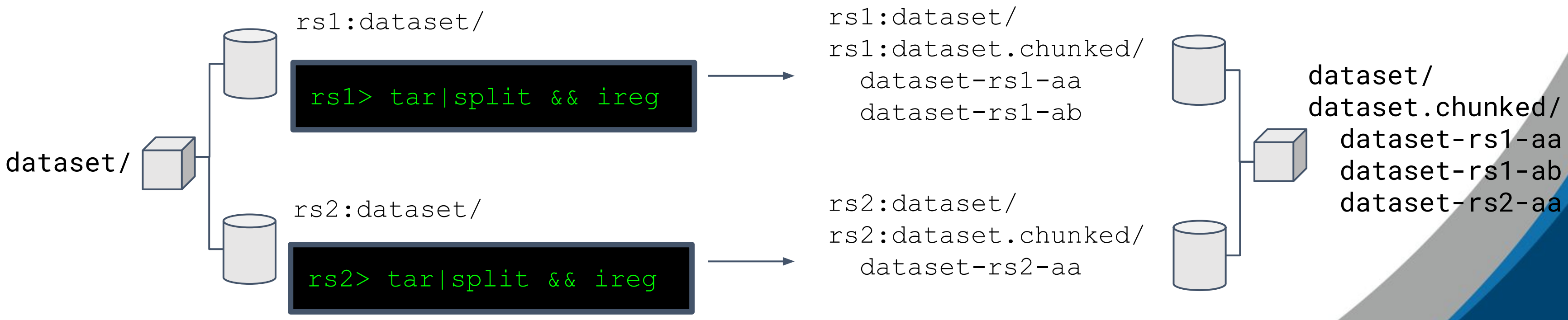
1. If data set distributed across multiple servers, ibun replicates them all to single server first – slow!
2. Very large file transfer can be problematic

Downloading Large Set of Small Files

Another iRODS solution

tar + split + ireg data set on each resource

1. tar+split data set's folder on each resource server
2. Use ireg to register chunks generated by split into iRODS
3. Use iget to download all chunks
4. For each resource server's set of chunks, use cat+tar to recover



```
rs1> tar|split && ireg
```

```
rs2> tar|split && ireg
```

Downloading Large Set of Small Files

Preliminary implementation

Two step process requiring admin user

1. Admin runs script that generates the data set chunks on each resource server and registers them
2. User runs second script that downloads the chunks and extracts the data set

Example usage

One project transferred 100 TB data set (6 million files) 4000 km over a 10 Gbit/s connection. Entire process took a little over 2 days.

Source code

https://github.com/cyverse/irods-adm/tree/master/chunk-transfer

Downloading Large Set of Small Files

Potential solution that doesn’t require admin user

1. Implement chunking and registration using server-side command script
2. Create iRODS rule that uses msiExecCmd microservice to execute command script on each resource server
3. Create client-side script that uses irule to invoke rule on provided data set, then downloads chunks and extracts data set

How about an iRODS Administration Interest Group?

- Goal is to improve the iRODS administration experience by
  - Developing and documenting best practices
  - Defining new features for iRODS
- Focus on solving administration problems
- Topic and discussion format
- Intended to fit in the space between iRODS-chat and Trirods
- *Interested? Join me for a BOF today at lunch.*