iRODS Capabilities

- Packaged and supported solutions
- Require configuration not code
- Derived from the majority of use cases observed in the user community

- Storage Tiering
- Auditing
- Provenance
- Data Integrity
- Automated Ingest
- Indexing
- Compliance
- Publishing
Storage Tiering Overview

Periodically, the storage tiering policy discovers data objects in violation via a default query and schedules their migration to the next tier group.

After 1800 seconds, any data objects in violation are automatically replicated to tier 1, and then once at rest, they are trimmed from tier 0.

After 9000 seconds, any data objects in violation are automatically replicated to tier 2, and then once at rest, they are trimmed from tier 1.

The default query that determines which data objects are in violation can be overridden by adding a new metadata attribute irods::storage_tier_query with a value that defines the custom query.

FEDERATE SECURELY

OTHER ORGANIZATION

DATA SOURCES

YOUR ORGANIZATION

UNIFIED NAMESPACE
Data Object Access Time

The default policy for tiering is based on the last time of access for a given data object which is applied as metadata

```cpp
irods::access_time <unix timestamp>
```

Dynamic Policy Enforcement Points for RPC API are used to apply the metadata

```cpp
pep_api_data_obj_close_post
pep_api_data_obj_put_post
pep_api_data_obj_get_post
pep_api_phy_path_reg_post
```
Configuring a Tier Group

Tier groups are entirely driven by metadata

- The attribute identifies the resource as a tiering group participant
- The value defines the group name
- The unit defines the position within the group

```
imeta set -R <resc0> irods::storage_tiering::group example_group 0
imeta set -R <resc1> irods::storage_tiering::group example_group 1
imeta set -R <resc2> irods::storage_tiering::group example_group 2
```

- Tier position, or index, can be any value - order will be honored
- Configuration must be performed at the root of a resource composition
- A resource may belong to many tiering groups
Configuring Tiering Time Constraints

Tiering violation time is configured in seconds

Configure a tier to hold data for 30 seconds

```
imeta set -R <resc> irods::storage_tiering::time 30
```

Configure a tier to hold data for 30 days

```
imeta set -R <resc> irods::storage_tiering::time 2592000
```

The final tier in a group does not have a storage tiering time - it will hold data indefinitely
Verification of Data Migration

When data is found to be in violation:

- Data object is replicated to the next tier
- New replica integrity is verified (in one of three ways)
- Source replica is trimmed

'catalog' is the default verification for all resources

```
imeta set -R <resc> irods::storage_tiering::verification catalog
```

For verification, this setting will determine if the replica is properly registered within the catalog after replication.
Verification of Data Migration

Filesstem verification is more expensive as it involves a potentially remote file system stat.

```
imeta add -R <resc> irods::storage_tiering::verification filesystem
```

This option will stat the remote replica on disk and compare the file size with that of the catalog.
Checksum verification is the most expensive as file sizes may be large

```
imeta add -R <resc> irods::storageTiering::verification checksum
```

Compute a checksum of the data once it is at rest, and compare with the value in the catalog.

Should the source replica not have a checksum one will be computed before the replication is performed.
Configuring the restage resource

When data is in a tier other than the lowest tier, upon access the data is restaged back to the lowest tier.

This flag identifies the tier for restage:

```
imeta add -R <resc> irods::storageTiering::minimumRestageTier true
```

Users may not want data restaged back to the lowest tier, should that tier be very remote or not appropriate for analysis.

Consider a storage resource at the edge serving as a landing zone for instrument data.
Preserving Replicas

Some users may not wish to trim a replica from a tier when data is migrated, such as to allow data to be archived and also still available on fast storage.

To preserve a replica on any given tier, attach the following metadata flag to the root resource.

```
imeta set -R <resc> irods::storage_tiering::preserve_replicas true
```
Custom Violation Query

Admins may specify a custom query which identifies violating data objects

```
imeta set -R <resc> irods::storage_tiering::query "SELECT DATA_NAME, COLL_NAME, DATA_RESC_ID WHERE META_DATA_ATTR_NAME = 'irods::access_time' AND META_DATA_ATTR_VALUE < 'TIME_CHECK_STRING' AND DATA_RESC_ID IN ('10021', '10022')"
```

Any number of queries may be attached to a resource in order provide a range of criteria by which violating data may be identified

- could include user applied metadata
- could include externally harvested metadata
Custom Violating Specific Query

More complex SQL may be required to identify violating objects. Users may configure Specific Queries and attach those to a given tier within a group.

Create a specific query in SQL

```sql
iadmin asq "select distinct R_DATA_MAIN.data_name, R_COLL_MAIN.coll_name,
R_DATA_MAIN.resc_id from R_DATA_MAIN, R_COLL_MAIN, R_OBJT_METAMAP r_data_metamap,
R_META_MAIN r_data_meta_main where R_DATA_MAIN.resc_id IN (10021, 10022) AND
r_data_meta_main.meta_attr_name = 'archive_object' AND r_data_meta_main.meta_attr_value
= 'true' AND R_COLL_MAIN.coll_id = R_DATA_MAIN.coll_id AND R_DATA_MAIN.data_id =
r_data_metamap.object_id AND r_data_metamap.meta_id = r_data_meta_main.meta_id order by
R_COLL_MAIN.coll_name, R_DATA_MAIN.data_name" archive_query
```

Configure the specific query

```bash
imeta set -R <resc> irods::storage_tiering::query archive_query specific
```
Limiting violating query results

When working with large sets of data, throttling the amount of data migrated at one time can be helpful.

In order to limit the results of the violating queries attach the following metadata attribute with the value set as the query limit.

```
imeta set -R <resc> irods::storageTiering::object_limit LIMIT_VALUE
```
Logging data transfer

In order to record the transfer of data objects from one tier to the next, the storage tiering plugin on the ICAT server can be configured by setting "data_transfer_log_level" : "LOG_NOTICE" in the plugin_specific_configuration.

In /etc/irods/server_config.json add the configuration to the storage_tiering plugin instance:

```json
{
    "instance_name": "irods_rule_engine_plugin-storage_tiering-instance",
    "plugin_name": "irods_rule_engine_plugin-storage_tiering",
    "plugin_specific_configuration": {
        "data_transfer_log_level" : "LOG_NOTICE"
    }
}
```
Storage Tiering Metadata Vocabulary

All default metadata attributes are configurable

```
"plugin_specific_configuration": {
    "access_time_attribute" : "irods::access_time",
    "storage_tiering_group_attribute" : "irods::storage_tiering::group",
    "storage_tiering_time_attribute" : "irods::storage_tiering::time",
    "storage_tiering_query_attribute" : "irods::storage_tiering::query",
    "storage_tiering_verification_attribute" : "irods::storage_tiering::verification",
    "storage_tiering_restage_delay_attribute" : "irods::storage_tiering::restage_delay",
    "default_restage_delay_parameters" : "<PLUSET>1s</PLUSET><EF>1h DOUBLE UNTIL SUCCESS OR 6 TIMES</EF>",
    "time_check_string" : "TIME_CHECK_STRING"
}
```

Should there be a preexisting vocabulary in your organization, it can be leveraged by redefining the metadata attributes used by the storage tiering framework.
Getting Started
Installing Tiered Storage Plugin

As the **ubuntu** user

Install the package repository

```bash
wget -qO - https://packages.irods.org/irods-signing-key.asc | sudo apt-key add -
echo "deb [arch=amd64] https://packages.irods.org/apt/ $(lsb_release -sc) main" | \
sudo tee /etc/apt/sources.list.d/renCI-irods.list
sudo apt-get update
```

Install the storage tiering package

```bash
ubuntu@hostname:~$ sudo apt-get install irods-rule-engine-plugin-storage-tiering
```
Configuring the rule engine plugin

As the **irods** user

Edit `/etc/irods/server_config.json`

```
"rule_engines": [
    {
        "instance_name": "irods_rule_engine_plugin-storage_tiering-instance",
        "plugin_name": "irods_rule_engine_plugin-storage_tiering",
        "plugin_specific_configuration": {
        }
    },
    {
        "instance_name": "irods_rule_engine_plugin-irods_rule_language-instance",
        "plugin_name": "irods_rule_engine_plugin-irods_rule_language",
        "plugin_specific_configuration": {
            <snip>
        },
        "shared_memory_instance": "irods_rule_language_rule_engine"
    },
    ...
]
```

Note - Make sure **storage_tiering** is the only rule engine plugin listed above **irods_rule_language**.
Three Tier Group with Random Resources
Make some resources

As the **irods** user

```
  iadmin mkresc rnd0 random
  iadmin mkresc rnd1 random
  iadmin mkresc rnd2 random
  iadmin mkresc st_ufs0 unixfilesystem `hostname`:/tmp/irods/st_ufs0
  iadmin mkresc st_ufs1 unixfilesystem `hostname`:/tmp/irods/st_ufs1
  iadmin mkresc st_ufs2 unixfilesystem `hostname`:/tmp/irods/st_ufs2
  iadmin mkresc st_ufs3 unixfilesystem `hostname`:/tmp/irods/st_ufs3
  iadmin mkresc st_ufs4 unixfilesystem `hostname`:/tmp/irods/st_ufs4
  iadmin mkresc st_ufs5 unixfilesystem `hostname`:/tmp/irods/st_ufs5
  iadmin addchildtoresc rnd0 st_ufs0
  iadmin addchildtoresc rnd0 st_ufs1
  iadmin addchildtoresc rnd1 st_ufs2
  iadmin addchildtoresc rnd1 st_ufs3
  iadmin addchildtoresc rnd2 st_ufs4
  iadmin addchildtoresc rnd2 st_ufs5
```
Make some resources

Check the results

```bash
demoResc:unixfilesystem
  rnd0:random
    st_ufs0:unixfilesystem
    st_ufs1:unixfilesystem
  rnd1:random
    st_ufs2:unixfilesystem
    st_ufs3:unixfilesystem
  rnd2:random
    st_ufs4:unixfilesystem
    st_ufs5:unixfilesystem
```
Create a tier group named example_group, adding the metadata to the root resources

imeta set -R rnd0 irods::storage_tiering::group example_group 0
imeta set -R rnd1 irods::storage_tiering::group example_group 1
imeta set -R rnd2 irods::storage_tiering::group example_group 2
Set the Tiering Time Constraints

Configure tier 0 to hold data for 30 seconds

```
imeta set -R rnd0 irods::storage_tiering::time 30
```

Configure tier 1 to hold data for 60 seconds

```
imeta set -R rnd1 irods::storage_tiering::time 60
```

Tier 2 does not have a storage tiering time as it will hold data indefinitely
Testing the tiering

Stage some data into storage tier 0

iput -R rnd0 /tmp/stickers.jpg

Check the results

irods@hostname:~$ imeta ls -d stickers.jpg
AVUs defined for dataObj stickers.jpg:
attribute: irods::access_time
value: 1526134799
units:

irods@hostname:~$ ils -l
/tempZone/home/rods:
  rods 0 rnd0;st_ufs0 2157087 2018-05-11.11:51 &
stickers.jpg
Sample Tiering rule

JSON ingested by the Tiering Plugin

- run once until success or six failures

```
{
    "rule-engine-instance-name": "irods_rule_engine_plugin-tiered_storage-instance",
    "rule-engine-operation": "apply_storage_tiering_policy",
    "delay-parameters": "<PLUSET>1s</PLUSET><EF>1h DOUBLE UNTIL SUCCESS OR 6 TIMES</EF>",
    "storage-tier-groups": [
        "example_group_g2",
        "example_group"
    ]
}
```
INPUT null
OUTPUT ruleExecOut

In production this would be persistently on the delay queue
Launching the sample Tiering rule

Run the rule from the terminal

```bash
irule -r irods_rule_engine_plugin-storage_tiering-instance -F example_tiering_invocation.r
```

Check the delay queue

```bash
irods@hostname:~$ iqstat
id   name
10038  {"rule-engine-operation":"apply_storage_tiering_policy","storage-tier-groups":["example_group_g2","example_group"]}
```

Wait for the delay execution engine to fire...

Check the resource for stickers.jpg

```bash
irods@hostname:~$ ils -l
/tempZone/home/rods:
   rods           2  rnd1;st_ufs2    2157087 2018-05-12.10:22 &
   stickers.jpg
```
Launching the Tiering rule once more

The time for violation is 60 seconds for rnd1

```
irule -r irods_rule_engine_plugin-storage_tiering-instance -F example_tiering_invocation.r
```

Check the delay queue

```
irods@hostname:~$ iqstat
id  name
10038  "rule-engine-operation":"apply_storage_tiering_policy","storage-tier-groups":["example_group_g2","example_group"]
```

Wait for the delay execution engine to fire...

Check the resource for stickers.jpg

```
irods@hostname:~$ ils -l
/tempZone/home/rods:  
  rods 3 rnd2;st_ufs4 2157087 2018-05-12.10:22 & stickers.jpg
```
Restaging Data

Fetching data when it is not in the lowest tier will automatically trigger a restaging of the data.

The object will be replicated back to the lowest tier, honoring the verification policy.
Setting a Minimum Restage Tier

In order to flag a resource as the target resource for restaging data, we add metadata

```
imeta set -R rnd1 irods::storage_tiering::minimum_restage_tier true
```

The object will be replicated to this tier instead of the lowest tier

```
irods@hostname:$ irule
irods@hostname:$ irule
irods@hostname:$ ils -l
/tempZone/home/rods:
   rods 6 rnd2;st_ufs5 2157087 2018-05-15.15:10 &
   stickers.jpg
irods@hostname:$ iget -f stickers.jpg
irods@hostname:$ iqstat
   id  name
   10044 {"destination-resource":"rnd1","object-path":"/tempZone/home/rods/stickers.jpg","preserve-replicas":false,"rule-engine-operation":"migrate_object_to_resource","source-resource":"rnd2","verification-type":"catalog"}
irods@hostname:$ ils -l
/tempZone/home/rods:
   rods 7 rnd1;st_ufs2 2157087 2018-05-15.15:10 &
   stickers.jpg
```
Preserving replicas on a given storage tier

If we want to preserve replicas on a tier we can set a metadata flag

```
imeta set -R rnd1 irods::storage_tiering::preserve_replicas true
```

When the staging rule is invoked the replica on the rnd1 tier will not be trimmed after replication

```
irods@hostname:~$ !irule
irule -r irods_rule_engine_plugin-storage_tiering-instance -F example_tiering_invocation.r
```

```
irods@hostname:~$ ils -l
/tempZone/home/rods:
    rods       1  rnd1;st_ufs2  2157087 2018-05-15.15:28 &
    stickers.jpg
    rods       2  rnd2;st_ufs5  2157087 2018-05-15.15:28 &
    stickers.jpg
```

A replica is preserved for analysis while another is safe in the archive tier
Custom violation queries

Any number of queries may be attached to a tier to identify violating objects.

The resource ids of the leaf resources are necessary for the query, not the root.

```
  irods@hostname:~$ ilsresc -l st_ufs0
  resource name: ufs0
  id: 10019
  ...
  irods@hostname:~$ ilsresc -l st_ufs1
  resource name: ufs1
  id: 10020
  ....
```
Custom violation queries

Craft a query that replicates the default behavior

```sql
imeta set -R rnd0 irods::storage_tiering::query "SELECT DATA_NAME, COLL_NAME, DATA_RESC_ID WHERE META_DATA_ATTR_NAME = 'irods::access_time' AND META_DATA_ATTR_VALUE < 'TIME_CHECK_STRING' AND DATA_RESC_ID IN ('10019', '10020')" [general]
```

- Compare data object access time against TIME_CHECK_STRING
- TIME_CHECK_STRING macro is replaced with the current time by the plugin before the query is submitted
- Check DATA_RESC_ID against the list of child resource ids
- Columns DATA_NAME, COLL_NAME, DATA_RESC_ID must be queried in that order
- By default all queries are of the type general, which is optional
Custom violation queries

Identify the leaf resource ids for the middle tier

```
irods@hostname:~$ ilsresc -l st_ufs2
resource name: ufs2
id: 10021
...
```
```
irods@hostname:~$ ilsresc -l st_ufs3
resource name: ufs3
id: 10022
...
```
Custom violation queries

Craft a query that uses metadata to identify violating objects

Add a Specific Query to iRODS, using new resource ids

```
admin asq "select distinct R_DATA_MAIN.data_name, R_COLL_MAIN.coll_name,
R_DATA_MAIN.resc_id from R_DATA_MAIN, R_COLL_MAIN, R_OBJT_METAMAP r_data_metamap,
R_META_MAIN r_data_meta_main where R_DATA_MAIN.resc_id IN (10021, 10022) AND
r_data_meta_main.meta_attr_name = 'archive_object' AND r_data_meta_main.meta_attr_value =
'true' AND R_COLL_MAIN.coll_id = R_DATA_MAIN.coll_id AND R_DATA_MAIN.data_id =
r_data_metamap.object_id AND r_data_metamap.meta_id = r_data_meta_main.meta_id order by
R_COLL_MAIN.coll_name, R_DATA_MAIN.data_name" archive_query
```

Attach the query to the middle tier

```
imeta set -R rnd1 irods::storage_tiering::query archive_query specific
```

This query must be labeled specific via the units
Testing the queries

Starting over with stickers.jpg

```bash
irods@hostname:~$ irm -f stickers.jpg
irods@hostname:~$ iput -R rnd0 /tmp/stickers.jpg
```

Wait for it...

```bash
irods@hostname:~$ irule -r irods_rule_engine_plugin-storage_tiering-instance -F example_tiering_invocation.r
irods@hostname:~$ iqstat
id    name
10065 {"rule-engine-operation":"apply_storage_tiering_policy","storage-tier-groups":
["example_group_g2","example_group"]}
irods@hostname:~$ ils -l
/tempZone/home/rods:
  rods     1  rnd1;st_ufs3  2157087 2018-05-18 13:38 & stickers.jpg
```
Testing the queries

The file stopped at rnd1 as the time-based default query is now overridden

```
irods@hostname:~$ ils -l
/tempZone/home/rods:
   rods           1 rnd1;st_ufs3  2157087 2018-05-18.13:38 & stickers.jpg
```

Now set the metadata flag to archive the data object

```
irods@hostname:~$ imeta set -d /tempZone/home/rods/stickers.jpg archive_object true
```
Testing the queries

The metadata is set, run the tiering rule

```bash
irods@hostname:~$ irule -r irods_rule_engine_plugin-storage_tiering-instance -F example_tiering_invocation.r
irods@hostname:~$ iqstat
id   name
10065 "rule-engine-operation":"apply_storageTieringPolicy","storage-tier-groups":
["example_group_g2","example_group"]
irods@hostname:~$ ils -l
/tempZone/home/rods:
    rods  1   rnd1;st_ufs3  2157087 2018-05-18.13:38 & stickers.jpg
    rods  2   rnd2;st_ufs4  2157087 2018-05-18.14:16 & stickers.jpg
```

The preservation flag is still set so we have two replicas
Another example

Three Tier Groups with Common Archive
We will create data flow from instrument to archive
Create some more resources

```bash
iadmin mkresc tier2 unixfilesystem `hostname`:/tmp/irods/tier2
iadmin mkresc tier0_A unixfilesystem `hostname`:/tmp/irods/tier0_A
iadmin mkresc tier1_A unixfilesystem `hostname`:/tmp/irods/tier1_A
iadmin mkresc tier0_B unixfilesystem `hostname`:/tmp/irods/tier0_B
iadmin mkresc tier1_B unixfilesystem `hostname`:/tmp/irods/tier1_B
iadmin mkresc tier0_C unixfilesystem `hostname`:/tmp/irods/tier2_C
iadmin mkresc tier1_C unixfilesystem `hostname`:/tmp/irods/tier1_C
```
Create Tier Groups

Tier Group A

```
imeta set -R tier0_A irods::storage_tiering::group tier_group_A 0
imeta set -R tier1_A irods::storage_tiering::group tier_group_A 1
imeta add -R tier2  irods::storage_tiering::group tier_group_A 2
```  

Tier Group B

```
imeta set -R tier0_B irods::storage_tiering::group tier_group_B 0
imeta set -R tier1_B irods::storage_tiering::group tier_group_B 1
imeta add -R tier2  irods::storage_tiering::group tier_group_B 2
```  

Tier Group C

```
imeta set -R tier0_C irods::storage_tiering::group tier_group_C 0
imeta set -R tier1_C irods::storage_tiering::group tier_group_C 1
imeta add -R tier2  irods::storage_tiering::group tier_group_C 2
```
Set Tier Time Constraints

Tier 0

```
imeta set -R tier0_A irods::storage_tiering::time 30
imeta set -R tier0_B irods::storage_tiering::time 45
imeta set -R tier0_C irods::storage_tiering::time 15
```

Tier 1

```
imeta set -R tier1_A irods::storage_tiering::time 60
imeta set -R tier1_B irods::storage_tiering::time 120
imeta set -R tier1_C irods::storage_tiering::time 180
```

Tier 2 has no time constraints
Creating an automated periodic rule

Create a new rule file to periodically apply the storage tiering policy - foo.r

```json
{
   "rule-engine-instance-name": "irods_rule_engine_plugin-storage_tiering-instance",
   "rule-engine-operation": "apply_storage_tiering_policy",
   "delay-parameters": "<PLUSET>1s</PLUSET><EF>REPEAT FOR EVER</EF>",
   "storage-tier-groups": [
      "tier_group_A",
      "tier_group_B",
      "tier_group_C"
   ]
}
```

INPUT null
OUTPUT ruleExecOut

Launch the new rule

```bash
irods@hostname:~$ irule -r irods_rule_engine_plugin-storage_tiering-instance -F foo.r

irods@hostname:~$ iqstat
id  name
10096  "rule-engine-operation":"apply_storage_tiering_policy","storage-tier-groups": ["tier_group_A","tier_group_B","tier_group_C"]"
```

Stage data into all three tiers and watch

```
# Stage data to different tiers
irods@hostname:~$ iput -R tier0_A /tmp/stickers.jpg stickers_A.jpg
irods@hostname:~$ iput -R tier0_B /tmp/stickers.jpg stickers_B.jpg
irods@hostname:~$ iput -R tier0_C /tmp/stickers.jpg stickers_C.jpg

# List the files
irods@hostname:~$ ils -l
/tempZone/home/rods:
  rods          0    tier0_A     2157087  2018-05-18.21:03 & stickers_A.jpg
  rods          0    tier0_B     2157087  2018-05-18.21:08 & stickers_B.jpg
  rods          0    tier0_C     2157087  2018-05-18.21:13 & stickers_C.jpg
  rods           1  rnd1;st_ufs3  2157087  2018-05-18.13:38 & stickers.jpg
  rods           2  rnd2;st_ufs4  2157087  2018-05-18.14:16 & stickers.jpg
```
Wait for it...

And then again...

All newly ingested stickers_X all now reside in tier2
Questions?