CYVERSE®

Transforming Science Through Data-Driven Discovery

Using iRODS Rules to Automate Trash Management Policy

Urvika Gola

THE UNIVERSITY OF ARIZONA

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

Agenda

1. Understand what, why and how we are implementing trash management logic in CyVerse Data Store.
2. Code Walkthrough - Overview of Pepview Tool
3. Code Walkthrough - AVU Metadata logic for trash management
4. Code Walkthrough - Example of custom rule logic in dynamic PEP
5. Automate trash purging

Goal

To automate the **trash management process** for Data Objects and Collections in iRODS using **dynamic policy enforcement points (PEPs)** and **microservices** invoked using the **rule engine plugin framework.**

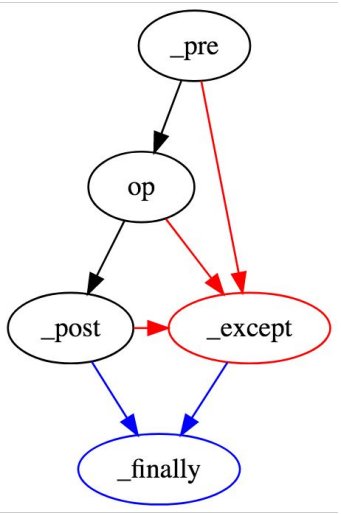
Significance

- Robustness - Enhance our existing solution which was based on `irmtrash`
- Data Retention Policy
- Efficient utilization of storage space
- User convenience, data Recovery
- To reduce the need for admin to manually monitor trash truncation on a regular basis

Key Components

- Dynamic PEPs
- Microservices
- Attribute-Value-Unit (AVU)

Dynamic PEPs



- For every operation that is called, four policy enforcement points are constructed (**pre, post, except, and finally**)
- By using some or all of the four distinct policy enforcement points we ensure that our trash management system adeptly handles **various data movement techniques.**

Image source - https://docs.irods.org/4.2.8/plugins/dynamic_policy_enforcement_points/

Step 1: Get a mapping between client operations and dynamic PEPs

Download Pepview Builder Tool from CyVerse git repo:
https://github.com/cyverse/irods-adm/blob/master/pepview-builder.sh
https://github.com/cyverse/irods-adm/blob/master/pep-fmt.awk

Run:
./pepview-builder.sh

Output:
pepview.re
pepview-default-peps.re

Step 1 contd:
Use pepview tool in server_config.json

```
"plugin_configuration": {
        "authentication": {},
        "database": {},
        "network": {},
        "resource": {},
        "rule_engines": [
            {
                "instance_name": "irods_rule_engine_plugin-
irods_rule_language-instance",
                "plugin_name": "irods_rule_engine_plugin-
irods_rule_language",
                "plugin_specific_configuration": {
                    "re_data_variable_mapping_set": [ "core" ],
                    "re_function_name_mapping_set": [ "core" ],
                    "re_rulebase_set": [
                        "pepview",
                        "pepview-default-peps",
                        "core"
                    ],
                    "regexes_for_supported_peps": [
                        "ac[^ ]*",
                        "msi[^ ]*",
                        "[^ ]*pep_[^ ]*_(pre|post|except|finally)"
                    ]
                },
```

Understanding pepview.re rule file

```
# continuation to the rule engine after writing out their call details. This
# will keep pepview from interfering with the normal policies.

pepview_CONTINUE = false


# This parameter controls which plugins pepview will log call details about. It
# accepts a list of plugin names. The supported plugins are 'api', 'auth',
# 'database', 'microservices', 'network', and 'resource'. The parameter can be
# assigned to the constant pepview_ALL_PLUGINS to have pepview log the call
# details for all supported plugins.

pepview_PLUGINS_SHOWN = list()
#pepview_PLUGINS_SHOWN = pepview_ALL_PLUGINS


# This parameter controls which plugin operations pepview will log call details
# about. It accepts a list of operation name wildcard patterns. The operation
# name for a dynamic PEP is derived from the PEP's name. A PEP name is a
# sequence of words separated by underscores. The first word is always "pep".
# The second is the name of the plugin invoking the PEP, and the last is the
# stage of the operation. The remaining words between the second and last form
# operation name. For example, if the name of the PEP is
# "pep_PLUGIN_doing_something_STAGE", the operation name is "doing_something".
# The parameter can be assigned to the constant pepview_ALL_OPS to have pepview
# display call details for all operations.

#pepview_OPS_SHOWN = list()
pepview_OPS_SHOWN = pepview_ALL_OPS


# This parameter controls which operation stages or phases pepview will log call
# details about. It accepts a list of stages. The stages are 'pre', 'post',
# 'except', and 'finally'. The parameter can be assigned to the constant
```

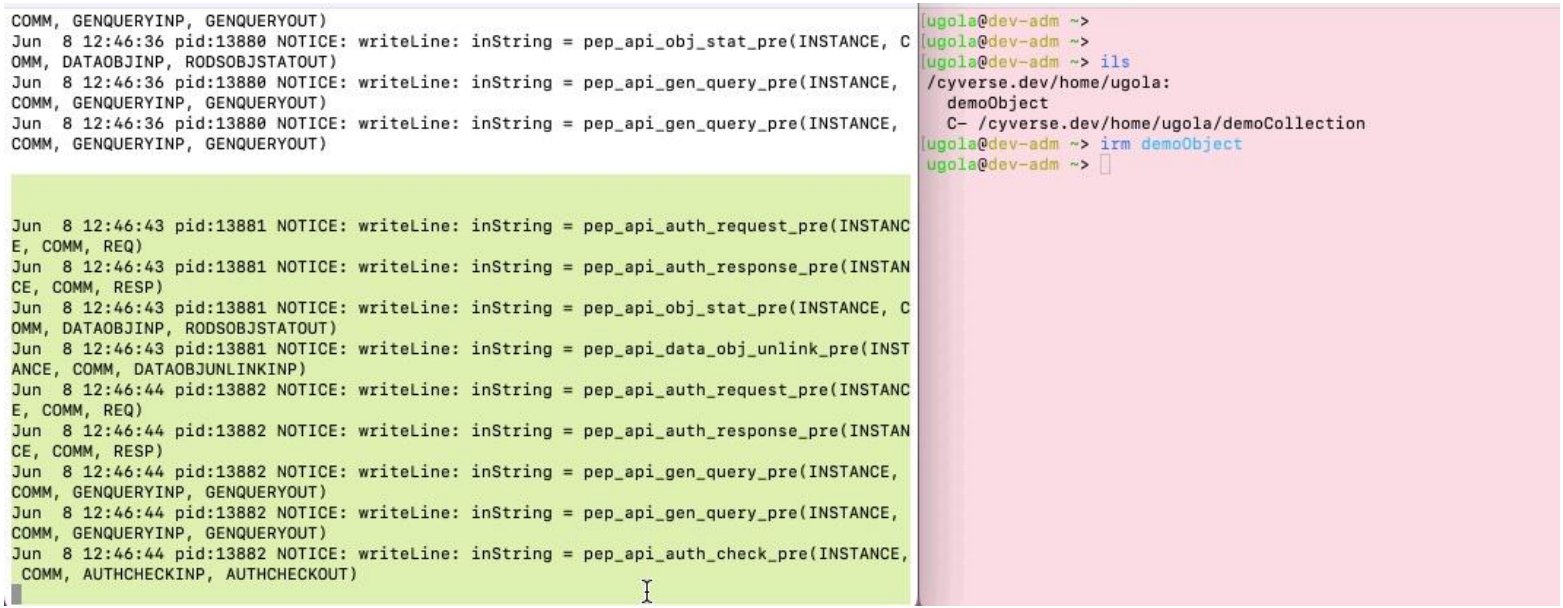
Understanding pepview.re rule file

```
# details for all supported plugins.

pepview_PLUGINS_SHOWN = list('api')
#pepview_PLUGINS_SHOWN = pepview_ALL_PLUGINS


# This parameter controls which plugin operations pepview will log call details
# about. It accepts a list of operation name wildcard patterns. The operation
# name for a dynamic PEP is derived from the PEP's name. A PEP name is a
# sequence of words separated by underscores. The first word is always "pep".
# The second is the name of the plugin invoking the PEP, and the last is the
# stage of the operation. The remaining words between the second and last form
# operation name. For example, if the name of the PEP is
# "pep_PLUGIN_doing_something_STAGE", the operation name is "doing_something".
# The parameter can be assigned to the constant pepview_ALL_OPS to have pepview
# display call details for all operations.

#pepview_OPS_SHOWN = list()
pepview_OPS_SHOWN = pepview_ALL_OPS


# This parameter controls which operation stages or phases pepview will log call
# details about. It accepts a list of stages. The stages are 'pre', 'post',
# 'except', and 'finally'. The parameter can be assigned to the constant
# pepview_ALL_STAGES to have pepview display call details for all stages.

pepview_STAGES_SHOWN = list('pre')
#pepview_STAGES_SHOWN = pepview_ALL_STAGES


# This parameter controls how the PEP rule arguments are displayed. A value of
# 'compact' means that the arguments will be displayed one per line. The ones
# with structured values will have all fields displayed on that line with fields
# separated by '++++'.  A value of 'expanded' is similar to 'compact', except
```

Understanding pepview.re rule file

```
# The second is the name of the plugin invoking the PEP, and the last is the
# stage of the operation. The remaining words between the second and last form
# operation name. For example, if the name of the PEP is
# "pep_PLUGIN_doing_something_STAGE", the operation name is "doing_something".
# The parameter can be assigned to the constant pepview_ALL_OPS to have pepview
# display call details for all operations.

#pepview_OPS_SHOWN = list()
pepview_OPS_SHOWN = pepview_ALL_OPS


# This parameter controls which operation stages or phases pepview will log call
# details about. It accepts a list of stages. The stages are 'pre', 'post',
# 'except', and 'finally'. The parameter can be assigned to the constant
# pepview_ALL_STAGES to have pepview display call details for all stages.

pepview_STAGES_SHOWN = list('pre')
#pepview_STAGES_SHOWN = pepview_ALL_STAGES


# This parameter controls how the PEP rule arguments are displayed. A value of
# 'compact' means that the arguments will be displayed one per line. The ones
# with structured values will have all fields displayed on that line with fields
# separated by '++++'.  A value of 'expanded' is similar to 'compact', except
# that the fields of structured values will also be diplsayed one per line. A
# value of 'none' (or any other value) means the argument values will not be
# displayed.

pepview_ARGS_DISPLAY = 'none'


# CONFIGURATION CONSTANTS
#
# No need to modify them
```

Using pepview tool



Using pepview tool

```
CE, COMM, RESP)
Jun  8 12:47:04 pid:13886 NOTICE: writeLine: inString = pep_api_obj_stat_pre(INSTANCE, C
OMM, DATAOBJINP, RODSOBJSTATOUT)
Jun  8 12:47:04 pid:13886 NOTICE: writeLine: inString = pep_api_specific_query_pre(INSTA
NCE, COMM, SPECIFICQUERYINP, GENQUERYOUT)
Jun  8 12:47:04 pid:13886 NOTICE: writeLine: inString = pep_api_specific_query_pre(INSTA
NCE, COMM, SPECIFICQUERYINP, GENQUERYOUT)
Jun  8 12:47:04 pid:13886 NOTICE: writeLine: inString = pep_api_gen_query_pre(INSTANCE,
COMM, GENQUERYINP, GENQUERYOUT)
Jun  8 12:47:04 pid:13886 NOTICE: writeLine: inString = pep_api_obj_stat_pre(INSTANCE, C
OMM, DATAOBJINP, RODSOBJSTATOUT)
Jun  8 12:47:04 pid:13886 NOTICE: writeLine: inString = pep_api_gen_query_pre(INSTANCE,
COMM, GENQUERYINP, GENQUERYOUT)
Jun  8 12:47:04 pid:13886 NOTICE: writeLine: inString = pep_api_gen_query_pre(INSTANCE,
COMM, GENQUERYINP, GENQUERYOUT)
```

```
Jun  8 12:47:08 pid:13887 NOTICE: writeLine: inString = pep_api_auth_request_pre(INSTANC
E, COMM, REQ)
Jun  8 12:47:08 pid:13887 NOTICE: writeLine: inString = pep_api_auth_response_pre(INSTAN
CE, COMM, RESP)
Jun  8 12:47:08 pid:13887 NOTICE: writeLine: inString = pep_api_obj_stat_pre(INSTANCE, C
OMM, DATAOBJINP, RODSOBJSTATOUT)
Jun  8 12:47:08 pid:13887 NOTICE: writeLine: inString = pep_api_rm_coll_pre(INSTANCE, CO
MM, RMCOLLINP, COLLOPRSTAT)
```

```
[ugola@dev-adm ~>
[ugola@dev-adm ~>
[ugola@dev-adm ~> ils
/cyverse.dev/home/ugola:
  demoObject
  C- /cyverse.dev/home/ugola/demoCollection
[ugola@dev-adm ~> irm demoObject
[ugola@dev-adm ~> ils
/cyverse.dev/home/ugola:
  C- /cyverse.dev/home/ugola/demoCollection
[ugola@dev-adm ~> irm -r demoCollection/
 ugola@dev-adm ~>
```

Covering various data movements to trash

- POSIX style open/write/close API calls - used by **Jargon** and **python-irodsclient** libraries
- Items removed using “**irm**” command
- Items moved directly to trash using “**imv**” command
- Items renamed in trash using “**imv**” command
- Items copied directly to trash using “**icp**” command
- Bulk upload directly to trash “**iput -b** <>”

Step 2: Adding AVU Metadata

- **Add** an AVU to a data object or a collection that stores the **epoch time at which it's moved to the trash.**
- Conversely, **remove** the AVU from the object or collection when **moved out of the trash.**
- **Automatically purge** data objects and collections in the trash for more than **30 days, based on the AVU value.**

Metadata (AVU) in action

```
[ugola@dev-adm ~>
[ugola@dev-adm ~> ils
 /cyverse.dev/home/ugola:
   demoObject
   C- /cyverse.dev/home/ugola/demoCollection
[ugola@dev-adm ~> imeta ls -d demoObject
 AVUs defined for dataObj /cyverse.dev/home/ugola/demoObject:
 attribute: ipc_UUID
 value: 9a64a500-064c-11ee-9a75-fa163e4a9b57
 units:
[ugola@dev-adm ~> imeta ls -C demoCollection/
 AVUs defined for collection /cyverse.dev/home/ugola/demoCollection:
 attribute: ipc_UUID
 value: 9d856c88-064c-11ee-9a75-fa163e4a9b57
 units:
[ugola@dev-adm ~> irm demoObject
[ugola@dev-adm ~> irm -r demoCollection/
[ugola@dev-adm ~> icd /cyverse.dev/trash/home/ugola/
[ugola@dev-adm ~> ils
 /cyverse.dev/trash/home/ugola:
   demoObject
   C- /cyverse.dev/trash/home/ugola/demoCollection
 ugola@dev-adm ~>
```

Metadata (AVU) in action

```
[ugola@dev-adm ~> icd /cyverse.dev/trash/home/ugola/
[ugola@dev-adm ~> ils
 /cyverse.dev/trash/home/ugola:
   demoObject
   C- /cyverse.dev/trash/home/ugola/demoCollection
[ugola@dev-adm ~> imeta ls -C demoCollection/
 AVUs defined for collection /cyverse.dev/trash/home/ugola/demoCollection:
 attribute: ipc::trash_timestamp
 value: 01686263748
 units:
 ----
 attribute: ipc_UUID
 value: 9d856c88-064c-11ee-9a75-fa163e4a9b57
 units:
[ugola@dev-adm ~> imeta ls -d demoObject
 AVUs defined for dataObj /cyverse.dev/trash/home/ugola/demoObject:
 attribute: ipc::trash_timestamp
 value: 01686263741
 units:
 ----
 attribute: ipc_UUID
 value: 9a64a500-064c-11ee-9a75-fa163e4a9b57
 units:
 ugola@dev-adm ~>
```

Metadata (AVU) in action

```
[ugola@dev-adm ~>
[ugola@dev-adm ~> ils
 /cyverse.dev/trash/home/ugola:
   demoObject
   C- /cyverse.dev/trash/home/ugola/demoCollection
[ugola@dev-adm ~> imv demoObject /cyverse.dev/home/ugola
[ugola@dev-adm ~> ils
 /cyverse.dev/trash/home/ugola:
   C- /cyverse.dev/trash/home/ugola/demoCollection
[ugola@dev-adm ~> icd
[ugola@dev-adm ~> ils
 /cyverse.dev/home/ugola:
   demoObject
[ugola@dev-adm ~> imeta ls -d demoObject
AVUs defined for dataObj /cyverse.dev/home/ugola/demoObject:
attribute: ipc_UUID
value: 9a64a500-064c-11ee-9a75-fa163e4a9b57
units:
ugola@dev-adm ~>
```

irods-fish

Working with iRODS command line tools within the fish shell.

https://github.com/cyverse/irods-fish

Dynamic PEPs used

- `pep_api_data_obj_unlink`
- `pep_api_data_obj_put`
- `pep_api_rm_coll`
- `pep_api_coll_create`
- `pep_api_data_obj_rename`
- `pep_api_data_obj_copy`
- `pep_api_data_obj_create`

Step 3: Example (pre)

```
pep_api_data_obj_unlink_pre(*INSTANCE, *COMM, *DATAOBJUNLINKINP) {
  if (errorcode(*DATAOBJUNLINKINP.forceFlag) != 0) {
    msiGetSystemTime(*timestamp, "");
    *dataObjPath = *DATAOBJUNLINKINP.obj_path;
    *timestampVar = _ipc_mkTimestampVar(/*dataObjPath);
    temporaryStorage.'*timestampVar' = *timestamp;
    msiModAVUMetadata("-d", *dataObjPath, "add", 'ipc::trash_timestamp', *timestamp);
  }
  msiSplitPath(*dataObjPath, *Coll, *File);
  foreach(*Row in SELECT DATA_ID
                  WHERE COLL_NAME = '*Coll'
                    AND DATA_NAME = '*File') {
                        *dataIdVar = _ipc_mkObjDataIdVar(/*dataObjPath);
                        temporaryStorage.'*dataIdVar' = *Row.DATA_ID;
  }
}
```

Step 3: Example (except)

```
pep_api_data_obj_unlink_except(*INSTANCE, *COMM, *DATAOBJUNLINKINP) {
  *dataObjPath = *DATAOBJUNLINKINP.obj_path;
  *timestampVar = _ipc_mkTimestampVar(/*dataObjPath);
  if (errorcode(temporaryStorage.'*timestampVar') == 0) {
    if (temporaryStorage.'*timestampVar' != "") {
      msiModAVUMetadata("-d", *dataObjPath, "rm", 'ipc::trash_timestamp',
temporaryStorage.'*timestampVar');
    }
  }
}
```

Step 3: Example (except)

```
pep_api_data_obj_unlink_post(*INSTANCE, *COMM, *DATAOBJUNLINKINP) {
  *dataObjPath = *DATAOBJUNLINKINP.obj_path;
  *timestampVar = _ipc_mkTimestampVar(/*dataObjPath);
  if (errorcode(temporaryStorage.'*timestampVar') == 0) {
    temporaryStorage.'*timestampVar' = "";
  }|
  *dataIdVar = _ipc_mkObjDataIdVar(/*dataObjPath);
  if (errorcode(temporaryStorage.'*dataIdVar') == 0) {
    *dataIdVarTemp = temporaryStorage.'*dataIdVar';
    foreach(*Row in SELECT COLL_NAME
                    WHERE DATA_ID = '*dataIdVarTemp') {
                      *collNameList = split(*Row.COLL_NAME, '/');
                      if (size(*collNameList) >= 5) {
                        *parentCollPath = "";
                        for (*i = 0; *i < 5; *i = *i + 1) {
                          *parentCollPath = *parentCollPath ++ "/" ++
elem(*collNameList, *i);
                        }
                        msiGetSystemTime(*timestamp, "");
                        msiModAVUMetadata("-C", *parentCollPath, "add",
'ipc::trash_timestamp', *timestamp);
                      }
    }
  }
}
```

Step 4: Automate trash clearing

Get the epoch timestamp to figure out which items are older than 30 days.

msiGetSystemTime(*timestamp, "");

2,592,000 is the number of seconds in 30 days. We subtract this value from the current timestamp to calculate the threshold time for items in the trash that are older than 30 days.

***int_month_timestamp = int(*timestamp) - 2592000;**

Step 4: Automate trash clearing (contd)

```
foreach(*Row in SELECT META_COLL_ATTR_VALUE, COLL_NAME
                    WHERE COLL_NAME like '/*zone/trash/%'
                      AND META_COLL_ATTR_NAME = 'ipc::trash_timestamp'
                        AND META_COLL_ATTR_VALUE <= *month_timestamp) {

                        *status = errorcode(msiRmColl(*Row.COLL_NAME, *FlagObj,
                                            *Status));

                        if (*status == 0) {
                               writeLine("serverLog", "Removed trash collection -
                               *Row.COLL_NAME");
                        } else {
                               writeLine("serverLog", "Unable to remove trash
                               collection - *Row.COLL_NAME");
                          *verdict = false;
                        }
}
```

Step 4: Automate trash clearing (contd)

```
if (*verdict) {
    *subject = ipc_ZONE ++ ' trash removal succeeded';
    *body = 'SSIA';
  } else {
    *subject = ipc_ZONE ++ ' trash removal failed';
    msiGetStderrInExecCmdOut(*out, *body);
    *body = 'View the irods logs for details';
  }

  if (0 != errorcode(msiSendMail(ipc_REPORT_EMAIL_ADDR, *subject, *body))) {
    writeLine('serverLog', 'Failed to mail trash removal report');
  }
  writeLine('serverLog', 'Completed trash removal');
}
```

Future scope

- Adapt this solution for **new** data movement methods using icommands, Jargon / irods-python-client APIs.
- Monitor logs for new errors and improve the rule logic.

Thank you