B2Safe module tutorial

Table of Contents

Introduction

This document aims to provide some examples and typical usage patterns of the EUDAT iRODS rules and scripts included in the B2SAFE module. The details about the configuration of the module are described in the administrator guide.

Data Ingestion

How to add a persistent identifier (PID) to an object, as soon as it is uploaded to a specific collection?

```
acPostProcForPut{
    on ( $objPath like "/TestZone/home/test_collection/*" )
    {
        *parent_pid = “”
        *ror = “”
        *iCATCache = bool(“true”)
        *path = $objPath
        EUDATCreatePID(*parent_pid, *path, *ror, *iCATCache, *newPID)
    }
}
```

If the object is a copy, the ROR is the persistent identifier of the original, otherwise is not defined.

If the object is a copy of a copy, the parent PID is the persistent identifier of the intermediate copy, otherwise is not defined.

This rule creates a PID record in the PID registry, but it can also store the PID in the iCAT as metadata, setting the boolean parameter iCATCache to “true”. The iCAT in this way works as a local cache for PID values, improving the performance during search tasks and shielding the local system from network or PID registry failures.

The previous rule creates a new PID for each object, but what if the object is overwritten and we want to keep the same persistent identifier, just updating the checksum?

```
acPostProcForPut{
    on ( $objPath like "/TestZone/home/test_collection/*" )
    {
        *path = $objPath
        *ePIDcheck = bool(“true”)
        *iCATuse = bool(“true”)
        *minTime = “120”
        EUDATeiPIDeiChecksumMgmt(*path, *newPID, *ePIDcheck, *iCATuse, *minTime)
    }
```

```
}
```

This rule searches whether a PID already exists for this object and in this case update its checksum. The search can be performed both at PID registry level and at local cache (iCAT) level or just at one of the two. If the PID does not exist, it is created.
However the PID is created when the upload of the object is completed. If the object size is very big or the network bandwidth very small, it could take time, therefore the parameter “minTime” allows to set the amount of time in seconds to wait for, before searching for the PID.

Data Replication

In order to replicate a single file the best option is the rule:

```
EUDATTransferSingleFile(*path_of_transfered_file,
                        *target_of_transfered_file)
```

This rule already includes integrity checks (based on object's size and checksum), proper registration and update of PIDs associated to the source and to the target, and a logging mechanism to track replication errors.
In case the replication source is a collection, then it is more suitable the following rule:

```
EUDATTransferCollection(*path_of_transfered_coll,
                        *target_of_transfered_coll,
                        *incremental,*recursive)
```

It relies on `EUDATTransferSingleFile`, hence it has its same features.
However this rule is quite demanding in terms of memory consumption, therefore in case of collection containing more than 1000 objects, it is advised to execute it in delayed mode (see `<B2SAFE_module>/rules/testtransfercollection.r`), for example:

```
delay("<EF>1s REPEAT UNTIL SUCCESS OR 10 TIMES</EF>") {
        EUDATTransferCollection(*Path,*replicaPath,
                                bool("true"),bool("true"));
}
```

Note the two boolean flags set to true which mean respectively “perform it in a incremental way” and “perform it in a recursive way”. The different options available for the delay mode are listed here: https://irods.sdsc.edu/index.php/Rule_Execution_modes .
In this way it will be possible to recover the replication, even after a failure due to out of memory errors.

Authorization

If you want to implement an ACL for the execution of an external command, such as a python script, a C code executable or a shell command, you can use the iRODS hook:

```
acPreProcForExecCmd(*cmd, *args, *addr, *hint) {
    if (*cmd != "authZ_manager.py") {
        EUDATAuthZ("$userNameClient#$rodsZoneClient",
                   *cmd, *args, *response);
    }
```

```
}
```

This hook can be put in the ruleset `<irods_home>/server/config/reConfig/core.re` .

Then in the file `<irods_home>/modules/B2SAFE/conf/authz.map.json` can be added the suitable assertions. So for example if the objective is to implement:

`Only user guybrush#MIslandZone can execute the python script <irods_home>/server/bin/cmd/drink_grog.py`

Then just add the following assertion in the authorization map:

```
{ "subject": [ "guybrush#MIslandZone" ],
  "action":  [ "<irods_home>/server/bin/cmd/drink_grog.py" ],
  "target":  [ "*" ]
}
```

But if you want a more fine-grained ACL, you can also specify the allowed input arguments:
`Only user guybrush#MIslandZone can execute the python script <irods_home>/server/bin/cmd/drink_grog.py -in acid_battery`

```
{ "subject": [ "guybrush#MIslandZone" ],
  "action":  [ "<irods_home>/server/bin/cmd/drink_grog.py" ],
  "target":  [ "-in acid_battery" ]
}
```

In principle, the same mechanism can be applied directly to filter the execution of every rule. For example, adding a line before the rule invocation in this way:

```
acPostProcForPut {
    EUDATAuthZ("$userNameClient#$rodsZoneClient",
               "EUDATTransferSingleFile", “*”, *response);
    EUDATTransferSingleFile(*path,*replicaPath);
}
```

And the related assertion in the map:

```
{ "subject": [ "user#CompanyZone" ],
  "action":  [ "EUDATTransferSingleFile" ],
  "target":  [ "*" ]
}
```

However the authorization mechanism implies a certain overhead so it should be used carefully.

Authentication and Account management

How can we make accessible our B2Safe instance to EUDAT users and in general to other communities?

```
15 8 * * * remote_users_sync.py -r conf/remote.users.sync.conf syncto EUDAT >
/dev/null

*/30 * * * * user_sync.py conf/user.sync.conf sync

12 * * * * quota_stats.py conf/quota.stats.conf compute
```

The three lines above allows to configure the chrontab of the B2Safe machine to integrate the B2Safe

local instance with the EUDAT infrastructure.
In particular the first line gets the user and group information from the EUDAT central user DB and store it into a local json file (cache).
Then the second one reads such file and synchronizes its content with B2Safe.

These scripts could create new users and groups in B2Safe, which means allocate storage space without our approval, what if we want to control the storage space allocation?
The second scripts allows to merge the users and groups coming from remote sources, like the EUDAT central DB, with those ones coming from the local source, for example an LDAP instance. And it is possible to configure it so that the local source is the only one allowed to create new collections (projects), while the remote ones can only add and remove users or sub-groups.

How can we keep consistent the namespace?
The scripts can be configured to add a prefix to the new users coming from remote sources, such the EUDAT DB. Anyway if the user name is already taken, within the local B2Safe instance, an email will be sent to the administrator to notify the conflict.

IRODS quota mechanism allows to calculate the used space per user and per group. How the space allocated to the various collections can be tracked?
The script in the third lines allows to calculate the space allocated for each main collection. It is written to a json file. Note that this script works in combination with the user_sync script which should write to the same file the quota limit defined in the local system.