Administrator guide

Table of Contents

Install

See <module_home>/install.txt .

Configuration

Authorization

- iRODS 3.3.x is required.
- in case you are using the version 3.3.0 please apply the patch in "rsExecCmd.patch" placed the dir "patches".
- in the rule file "eudat.re": there are two rules called "EUDATAuthZ" and "getAuthZParameters". The "getEUDATAuthZ" calls an external python script placed in iRODS_home/server/bin/cmd and called "authZ.manager.py". Which requires a configuration file placed in iRODS_home/modules/B2SAFE/cmd and called "authz.map.json". The script provides just a couple of methods: "test" and "check", which returns a boolean value of True if the authorization is granted, False otherwise. The authorization decision is based on the file "authz.map.json", which contains triplets (subject, action, target) called assertions. So, for example, passing to the script in input a request like:

  ```
  testuser#testzone,
  read,
  /iRODS_home/modules/B2SAFE/cmd/credentials
  ```

  It will be accepted if the json file contains:

  ```
      "assertion 1":
        { "subject":
           [ "testuser#testzone" ],
          "action":
           [ "read" ],
          "target":
           [ "/iRODS_home/modules/B2SAFE/cmd/credentials" ]
        }
  ```

  Or even:

```
    "assertion 1":
       { "subject":
          [ "*#testzone" ],
         "action":
          [ "read" ],
         "target":
          [ "/iRODS_home/modules/B2SAFE/cmd/*" ]
       }
```

Because it supports the wild characters in the same way a shell do.

- in the rulebase file "core.re" the hook shoul be configured using the patch "corere.patch" placed in the folder "patches" of the module.
- The entry point for rules specific for certain external executables should be called inside the "getEUDATAuthZ" as fall back.

Logging

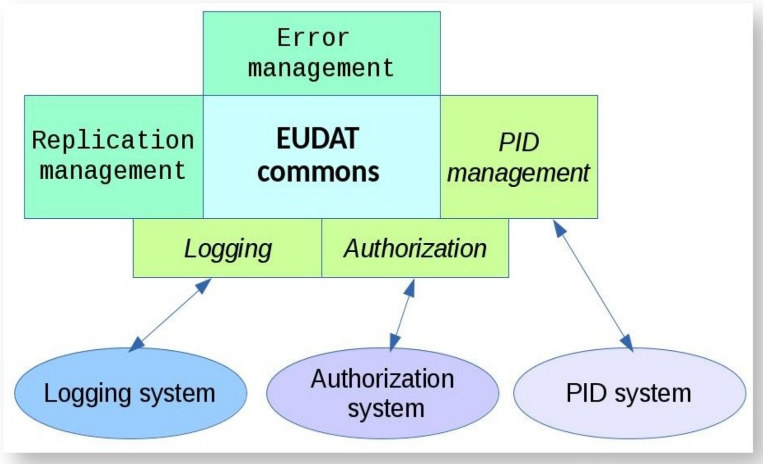
Just configure the logging level (INFO, DEBUG, ERROR) and the path to the logging directory:

```
{
"log_level": "DEBUG",
"log_dir": "/<iRODS path>/modules/B2SAFE/log",
}
```

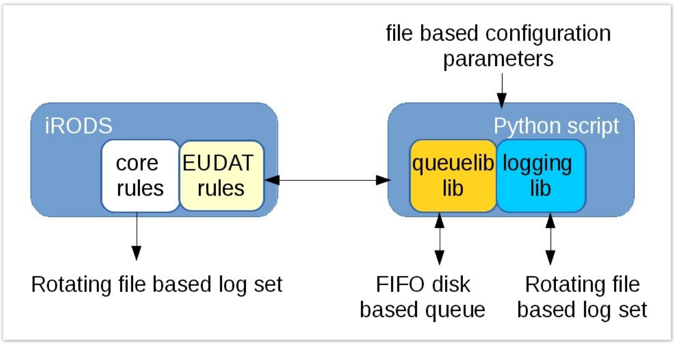
Changelog

See <module_home>/docs/changelog.txt

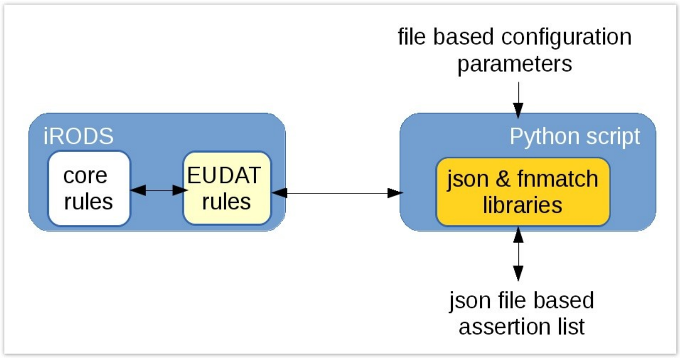
Architecture



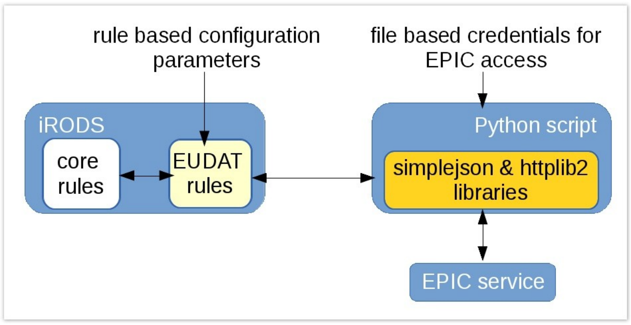
Logging:


file based configuration parameters
iRODS
core rules
EUDAT rules
Python script
queuelib lib
logging lib
Rotating file based log set
FIFO disk based queue
Rotating file based log set

Authorization:


file based configuration parameters
iRODS
core rules
EUDAT rules
Python script
json & fnmatch libraries
json file based assertion list

PID management:


rule based configuration parameters
file based credentials for EPIC access
iRODS
core rules
EUDAT rules
Python script
simplejson & httplib2 libraries
EPIC service

API (EUDAT rules)

Commons

EUDATiCHECKSUMretrieve(*path, *checksum)

Get an existent checksum from iCAT

EUDATiCHECKSUMget(*path, *checksum)

Get, if exist or create if not, a checksum from iCAT

EUDATgetObjectTimeDiff(*filePath, *age)

Calculate the difference between the creation time and the modification time of an object (in seconds).

EUDATfileInPath(*path,*subColl)

Check if a file is in a given path

EUDATCreateAVU(*Key,*Value,*Path)

Logging

EUDATLog(*message, *level)

Log an event

EUDATQueue(*action, *message, *number)

Log a failure to a FIFO queue

Authorization

EUDATAuthZ(*user, *action, *target, *response)

Authorization policy decision point

Create a metadata triplet on iCAT

PID management

EUDATCreatePID(*parent_pid, *path, *ror, *iCATCache, *newPID)

Create PID

EUDATSearchPID(*path, *existing_pid)

Search PID

EUDATSearchPIDchecksum(*path, *existing_pid)

Search PID by checksum

EUDATUpdatePIDWithNewChild(*parentPID, *childPID)

Update PID record field 10320/LOC

EUDATGetRorPid(*pid, *ror)

Get PID record field RoR's value

EUDATeiPIDeiChecksumMgmt(*path, *PID, *ePIDcheck, *iCATuse, *minTime)

Create or update a PID, including checksum

EUDATiPIDcreate(*path, *PID)

Create a PID as iCAT metadata

EUDATiFieldVALUEretrieve(*path, *FNAME, *FVALUE)

Get a metadata value from iCAT

EUDATePIDcreate(*path, *PID)

Create a PID as EPIC service record

EUDATePIDsearch(*field, *value, *PID)

Search a PID on the EPIC service

EUDATeCHECKSUMupdate(*PID)

Update the PID record field checksum

EUDATeURLupdate(*PID, *newURL)

Update the PID record field URL

EUDATePIDremove(*path)

Delete a PID

EUDATeiPIDeiChecksumMgmtColl(*sourceColl)

Walk through the collection. For each object, it creates a PID and stores its value and the object checksum in the iCAT.

EUDATiRORupdate(*source, *pid)

Add the ROR field of the PID of the object to

Replication management

EUDATUpdateLogging(*status_transfer_success, *path_of_transfered_file, *target_transfered_file, *cause)

Log a transfer event to the log file and, if it is a failure, to the FIFO queue

EUDATCheckError(*path_of_transfered_file,*target_of_transfered_file)

Perform error checks about the transfer

EUDATTransferSingleFile(*path_of_transfered_file,*target_of_transfered_file)

Transfer a single file

EUDATTransferUsingFailLog(*buffer_length)

Retry to perform a certain number of failed transfers queued in the FIFO queue

EUDATCheckReplicas(*source, *destination)

Check whether two files are available and identical

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

Transfer a whole collection

Error management

EUDATCatchErrorChecksum(*source,*destination)

Catch error with Checksum

EUDATCatchErrorSize(*source,*destination)

Catch error Size of file

EUDATProcessErrorUpdatePID(*updfile)

Process error update PID at Parent_PID. It will be processed during replication_workflow, called by updateMonitor.

EUDATCatchErrorDataOwner(*path,*status)

Catch error Data Owner if user is not owner of Data from *path iCAT

EUDATeParentUpdate(*PID, *PFName, *PFValue)

Update the EUDAT ROR or PPID field in the PID record

Best Practices

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/cmd/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 {
    getEUDATAuthZ("$userNameClient#$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.