

Administrator guide

Table of Contents

Install.....	1
Configuration.....	1
Authorization.....	1
Logging.....	2
Changelog.....	2
Architecture.....	2
API.....	3

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 "EUDATAAuthZ" and "getAuthZParameters". The "getEUDATAAuthZ" 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 should 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 "getEUDATAAuthZ" as fall back.

Logging

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

```

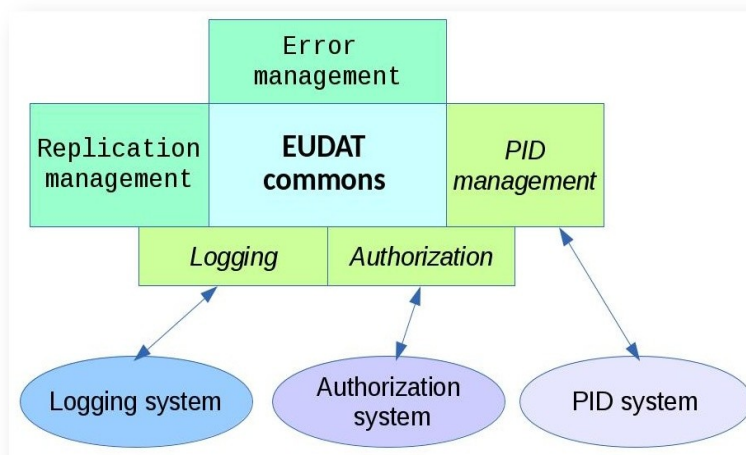
{
  "log_level": "DEBUG",
  "log_dir": "<iRODS_path>/modules/B2SAFE/log",
}

```

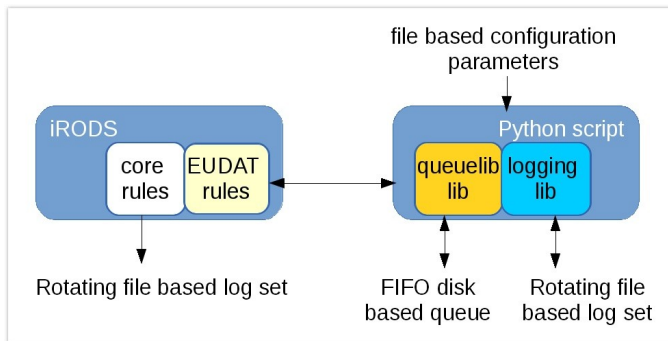
Changelog

See <module_home>/docs/changelog.txt

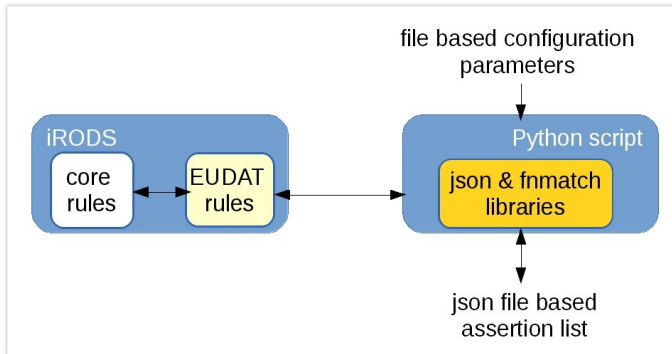
Architecture



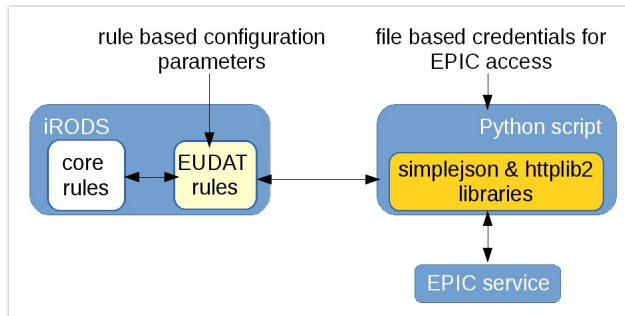
Logging:



Authorization:



PID management:



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	Replication management	Error management
EUDATCreatePID(*parent_pid, *path, *ror, *iCATCache, *newPID) Create PID	EUDATUpdateLogging(*status_transfer_success, *path_of_transferred_file, *target_transferred_file, *cause) Log a transfer event to the log file and, if it is a failure, to the FIFO queue	EUDATCatchErrorChecksum(*source,*destination) Catch error with Checksum
EUDATSearchPID(*path, *existing_pid) Search PID	EUDATCheckError(*path_of_transferred_file,*target_of_transferred_file) Perform error checks about the transfer	EUDATCatchErrorSize(*source,*destination) Catch error Size of file
EUDATSearchPIDchecksum(*path, *existing_pid) Search PID by checksum	EUDATTransferSingleFile(*path_of_transferred_file,*target_of_transferred_file) Transfer a single file	EUDATProcessErrorUpdatePID(*updfile) Process error update PID at Parent_PID. It will be processed during replication_workflow, called by updateMonitor.
EUDATUpdatePIDWithNewChild(*parentPID, *childPID) Update PID record field 10320/LOC	EUDATTransferUsingFailLog(*buffer_length) Retry to perform a certain number of failed transfers queued in the FIFO queue	EUDATCatchErrorDataOwner(*path,*status) Catch error Data Owner if user is not owner of Data from *path
EUDATGetRorPid(*pid, *ror) Get PID record field RoR's value	EUDATCheckReplicas(*source, *destination) Check whether two files are available and identical	
EUDATeIPiDeiChecksumMgmt(*path, *PID, *ePIDcheck, *iCATuse, *minTime) Create or update a PID, including checksum	EUDATTransferCollection(*path_of_transferred_coll,*target_of_transferred_coll,*incremental,*recursive) Transfer a whole collection	
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(*sourceCollection) 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		

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") {  
        EUDATAAuthZ("$userNameClient#$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 {  
    EUDATAAuthZ("$userNameClient#$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.

Replication

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

```
EUDATTransferSingleFile(*path_of_transferred_file,  
                        *target_of_transferred_file)
```

contained in the ruleset <B2SAFE_module>/rulebase/replication.re .

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_transferred_coll,  
                       *target_of_transferred_coll,  
                       *incremental,*recursive)
```

Which is included in the aforementioned ruleset.

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 option 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.