iRODS

Pluggable Rule Engine

CurateGear2016

Terrell Russell, Ph.D.

@terrellrussell

Senior Data Scientist, iRODS Consortium

Renaissance Computing Institute (RENCI), UNC-Chapel Hill


iRODS
CONSORTIUM
renci

iRODS Consortium

The iRODS Consortium was created to ensure the sustainability of iRODS and to further its adoption and continued evolution. To this end, the Consortium works to standardize the definition, development, and release of iRODS-based data middleware technologies, evangelize iRODS among potential users, promote new advances in iRODS, and expand the adoption of iRODS-based data middleware technologies through the development, release, and support of an open-source, mission-critical, production-level distribution of iRODS.

Current Members:

renci

D·I·C·E

SEAGATE

DDN

NOVARTIS

IBM

Complete GENOMICS

wellcome trust sanger institute

UCL

NASA

EMC²

cleversafe

NICS

Four Major Areas of Deployment

- Health Care & Life Science
- Oil & Gas
- Media & Entertainment
- Archives & Records Management

Open Source Data Management Middleware

- iRODS **enables data discovery** using a metadata catalog that describes every file, every directory, and every storage resource in the data grid.
- iRODS **automates data workflows**, with a rule engine that permits any action to be initiated by any trigger on any server or client in the grid.
- iRODS **enables secure collaboration**, so users only need to log in to their home grid to access data hosted on a remote grid.
- iRODS **implements data virtualization**, allowing access to distributed storage assets under a unified namespace, and freeing organizations from getting locked in to single-vendor storage solutions.

Open Source Data Management Middleware

- iRODS **enables data discovery** using a metadata catalog that describes every file, every directory, and every storage resource in the data grid.
- iRODS **automates data workflows**, with a rule engine that permits any action to be initiated by any trigger on any server or client in the grid.
- iRODS **enables secure collaboration**, so users only need to log in to their home grid to access data hosted on a remote grid.
- iRODS **implements data virtualization**, allowing access to distributed storage assets under a unified namespace, and freeing organizations from getting locked in to single-vendor storage solutions.

Pluggable Rule Engine

- Part of iRODS 4.2, Spring 2016
- Rule Engine Plugins are written in C++
- Allows rules to be written in any language (both interpreted and compiled)
- Multiple rule engines can run concurrently, allowing calls from one language to another
- Today's demo:
  - (existing) iRODS Rule Language
  - Javascript
  - Python

Pluggable Rule Engine

Rule Engine Plugin	LOC (w/ comments)
iRODS Rule Language	228
Javascript	222
Python	252
Auditing (C++)	157

Defined operations:

- start
- stop
- rule_exists
- exec_rule
- (probably also) prepend_to_rulebase

Policy Enforcement Points (PEPs)

A Policy Enforcement Point (or PEP) is a location in the iRODS Server code where policy can be inserted by an organization. Policy takes the form of a set of rules that are executed by the rule engine.

Consider, for example, a rule to transfer ownership of data objects to the project manager when a user is deleted; the trigger (or PEP) is the deletion of the user.

Rules could be written to extract metadata or pre-process data whenever a file is uploaded to a storage device. Or, upon access to particular data objects, a rule can create a log of the event, send an email notification to the project manager, or perform some other task you need to occur as a result of the data's access.

We will be using acPostProcForPut().

This is the location in the code that is hit after a file has been uploaded into iRODS. This location in the code has a full picture of the context in which it is executing. It knows the username, the filename, the location on disk, etc.

Policy Enforcement Points (PEPs)

audit_pep_auth_agent_auth_response_post
audit_pep_auth_agent_auth_response_pre
audit_pep_auth_agent_start_post
audit_pep_auth_agent_start_pre
audit_pep_database_check_auth_post
audit_pep_database_check_auth_pre
audit_pep_database_gen_query_access_control_setup_post
audit_pep_database_gen_query_access_control_setup_pre
audit_pep_database_gen_query_post
audit_pep_database_gen_query_pre
audit_pep_database_get_rcs_post
audit_pep_database_get_rcs_pre
audit_pep_database_mod_data_obj_meta_post
audit_pep_database_mod_data_obj_meta_pre
audit_pep_database_reg_data_obj_post
audit_pep_database_reg_data_obj_pre
audit_pep_exec_microservice_post
audit_pep_exec_microservice_pre
audit_pep_exec_rule_post
audit_pep_exec_rule_pre
audit_pep_resource_resolve_hierarchy_post
audit_pep_resource_resolve_hierarchy_pre
audit_pep_resource_stat_post
audit_pep_resource_stat_pre
audit_pep_resource_write_post
audit_pep_resource_write_pre

Three Rule Bases, Part 1 - iRODS Rule Language

```
# existing iRODS Rule Language - custom.re

irodsFunc0() {
    writeLine("serverLog", "custom.re - BEGIN - irodsFunc0()");
    writeLine("serverLog", "custom.re -- Hi Curate Gear!");
    writeLine("serverLog", "custom.re - END   - irodsFunc0()");
}

irodsFunc1(*foo) {
    writeLine("serverLog", "custom.re - BEGIN - irodsFunc1(foo): [*foo]");
    pyFunc1("called from custom.re");
    writeLine("serverLog", "custom.re - END   - irodsFunc1(foo)");
}

add_metadata_to_objpath(*str, *objpath, *objtype) {
    msiString2KeyValPair(*str, *kvp);
    msiAssociateKeyValuePairsToObj(*kvp, *objpath, *objtype);
}

getSessionVar(*name, *output) {
    *output = eval("str($"++*name++")");
}
```

Three Rule Bases, Part 2 - Javascript

```
/* Javascript - core.js */

function jsFunc0(callback) {
    callback.writeLine("serverLog", "JAVASCRIPT - BEGIN - jsFunc0(callback)");
    callback.writeLine("serverLog", "JAVASCRIPT -- Happy New Year!");
    callback.writeLine("serverLog", "JAVASCRIPT - END   - jsFunc0(callback)");
}

function jsFunc1(foo, callback) {
    callback.writeLine("serverLog", "JAVASCRIPT - BEGIN - jsFunc1(foo, callback)");
    callback.writeLine("serverLog", "  - with parameter foo[" + foo + "]");
    try {
        callback.doesnotexist("nope");
    } catch(e) {
        throw e + " -- ERROR HANDLING FTW!";
    }
    callback.writeLine("serverLog", "JAVASCRIPT - END   - jsFunc1(foo, callback)");
}

/*********************************
 * DEMO 1 - Just Print To rodsLog *
 *********************************/
function XXXacPostProcForPut(callback) {
    callback.writeLine("serverLog", "JAVASCRIPT - BEGIN - acPostProcForPut()");
    callback.irodsFunc0();
    callback.pyFunc0();
    callback.writeLine("serverLog", "JAVASCRIPT - END   - acPostProcForPut()");
}
```

Three Rule Bases, Part 3 - Python

```
# Python - core.py

import datetime

def pyFunc0(rule_args, callback):
    callback.writeLine('serverLog', 'PYTHON - BEGIN - pyFunc0(callback)')
    callback.writeLine('serverLog', 'PYTHON -- ' + str(datetime.datetime.now()))
    callback.writeLine('serverLog', 'PYTHON - END   - pyFunc0(callback)')

def pyFunc1(rule_args, callback):
    callback.writeLine('serverLog', 'PYTHON - BEGIN - pyFunc1(rule_args, callback)')
    for arg in (rule_args):
        callback.writeLine('serverLog', 'PYTHON -- arg=[' + arg + ']')
    callback.writeLine('serverLog', 'PYTHON - END   - pyFunc1(rule_args, callback)')

#####################################
# DEMO 2 - Parameters and Error Handling #
#####################################
def XXXacPostProcForPut(rule_args, callback):
    callback.writeLine('serverLog', 'PYTHON - BEGIN - acPostProcForPut()')
    callback.irodsFunc1("called from python, apples")
    callback.jsFunc1("called from python, bananas")
    session_vars = ['userNameClient', 'dataSize',]
    for s in session_vars:
        v = callback.getSessionVar(s, 'dummy')[1]
        callback.writeLine('serverLog', s + ' :: ' + v)
    callback.writeLine('serverLog', 'PYTHON - END   - acPostProcForPut()')

###############################
# DEMO 3 - EXIF Extraction Demo #
###############################
import os
import sys
from PIL import Image
from PIL.ExifTags import TAGS
def XXXacPostProcForPut(rule_args, callback):
    phypath = callback.getSessionVar('filePath', 'dummy')[1]
    callback.writeLine('serverLog', phypath)
    objpath = callback.getSessionVar('objPath', 'dummy')[1]
    callback.writeLine('serverLog', objpath)
    exiflist = []
    for (k, v) in Image.open(phypath)._getexif().iteritems():
        exifpair = '%s=%s' % (TAGS.get(k), v)
        exiflist.append(exifpair)
    exifstring = "%".join(exiflist)
    callback.add_metadata_to_objpath(exifstring, objpath, '-d')
    callback.writeLine('serverLog', 'PYTHON - acPostProcForPut() EXIF complete')
```

DEMO - Baseline

iput a file into iRODS

```
$ iput puppies.jpg
```

rodsLog:

```
Jan 13 13:00:04 pid:26540 NOTICE: Agent process 30793 started for puser=rods and cuser=rods from #.#.#
Jan 13 13:00:04 pid:30793 NOTICE: readAndProcClientMsg: received disconnect msg from client
Jan 13 13:00:04 pid:30793 NOTICE: Agent exiting with status = 0
Jan 13 13:00:04 pid:26540 NOTICE: Agent process 30793 exited with status 0
```

DEMO 1 - Just Print to rodsLog

overload the acPostProcForPut() policy enforcement point

```
# remove prepended XXX in core.js DEMO 1 function name
function acPostProcForPut(callback) {...}
```

remove, then iput the same file into iRODS

```
$ irm -rf puppies
$ iput puppies.jpg
```

rodsLog:

```
Jan 13 13:01:34 pid:26540 NOTICE: Agent process 30936 started for puser=rods and cuser=rods from #.#.#
Jan 13 13:01:35 pid:30936 NOTICE: readAndProcClientMsg: received disconnect msg from client
Jan 13 13:01:35 pid:30936 NOTICE: Agent exiting with status = 0
Jan 13 13:01:35 pid:26540 NOTICE: Agent process 30936 exited with status 0

Jan 13 13:01:35 pid:26540 NOTICE: Agent process 30943 started for puser=rods and cuser=rods from #.#.#
Jan 13 13:01:35 pid:30943 NOTICE: writeLine: inString = JAVASCRIPT - BEGIN - acPostProcForPut()
Jan 13 13:01:35 pid:30943 NOTICE: writeLine: inString = custom.re - BEGIN - irodsFunc0()
Jan 13 13:01:35 pid:30943 NOTICE: writeLine: inString = custom.re -- Hi Curate Gear!
Jan 13 13:01:35 pid:30943 NOTICE: writeLine: inString = custom.re - END   - irodsFunc0()
Jan 13 13:01:35 pid:30943 NOTICE: writeLine: inString = PYTHON - BEGIN - pyFunc0(callback)
Jan 13 13:01:35 pid:30943 NOTICE: writeLine: inString = PYTHON -- 2016-01-13 13:01:35.522276
Jan 13 13:01:35 pid:30943 NOTICE: writeLine: inString = PYTHON - END   - pyFunc0(callback)
Jan 13 13:01:35 pid:30943 NOTICE: writeLine: inString = JAVASCRIPT - END   - acPostProcForPut()
Jan 13 13:01:35 pid:30943 NOTICE: readAndProcClientMsg: received disconnect msg from client
Jan 13 13:01:35 pid:30943 NOTICE: Agent exiting with status = 0
Jan 13 13:01:35 pid:26540 NOTICE: Agent process 30943 exited with status 0
```

DEMO 2 - Parameters and Error Handling

overload the acPostProcForPut() policy enforcement point

```
# remove prepended XXX in core.py DEMO 2 function name
def acPostProcForPut(rule_args, callback):
```

remove, then iput the same file into iRODS

```
$ irm -rf puppies
$ iput puppies.jpg
```

rodsLog:

```
Jan 13 13:02:01 pid:26540 NOTICE: Agent process 30986 started for puser=rods and cuser=rods from #.#.#.#
Jan 13 13:02:01 pid:30986 NOTICE: readAndProcClientMsg: received disconnect msg from client
Jan 13 13:02:01 pid:30986 NOTICE: Agent exiting with status = 0
Jan 13 13:02:01 pid:26540 NOTICE: Agent process 30986 exited with status 0

Jan 13 13:02:01 pid:26540 NOTICE: Agent process 30993 started for puser=rods and cuser=rods from #.#.#.#
Jan 13 13:02:01 pid:30993 NOTICE: writeLine: inString = PYTHON - BEGIN - acPostProcForPut()
Jan 13 13:02:01 pid:30993 NOTICE: writeLine: inString = custom.re - BEGIN - irodsFunc1(foo): [called from python, apples]
Jan 13 13:02:01 pid:30993 NOTICE: writeLine: inString = PYTHON - BEGIN - pyFunc1(rule_args, callback)
Jan 13 13:02:01 pid:30993 NOTICE: writeLine: inString = PYTHON -- arg=[called from custom.re]
Jan 13 13:02:01 pid:30993 NOTICE: writeLine: inString = PYTHON - END   - pyFunc1(rule_args, callback)
Jan 13 13:02:01 pid:30993 NOTICE: writeLine: inString = custom.re - END   - irodsFunc1(foo)
Jan 13 13:02:01 pid:30993 NOTICE: writeLine: inString = JAVASCRIPT - BEGIN - jsFunc1(foo, callback)
Jan 13 13:02:01 pid:30993 NOTICE: writeLine: inString =   - with parameter foo[called from python, bananas]
Jan 13 13:02:01 pid:30993 ERROR: [-]    iRODS/server/re/src/rules.cpp:674:int actionTableLookUp(irods::ms_table_entry &,
        [-]     iRODS/server/re/src/irods_ms_plugin.cpp:110:irods::error irods::load_microservice_plugin(ms_table &, cons
                [-]     iRODS/lib/core/include/irods_load_plugin.hpp:145:irods::error irods::load_plugin(PluginType *&, c
Jan 13 13:02:01 pid:30993 ERROR: -1102000 -- ERROR HANDLING FTW!
Jan 13 13:02:01 pid:30993 NOTICE: writeLine: inString = userNameClient :: rods
Jan 13 13:02:01 pid:30993 NOTICE: writeLine: inString = dataSize :: 95891
Jan 13 13:02:01 pid:30993 NOTICE: writeLine: inString = PYTHON - END   - acPostProcForPut()
Jan 13 13:02:01 pid:30993 NOTICE: readAndProcClientMsg: received disconnect msg from client
Jan 13 13:02:01 pid:30993 NOTICE: Agent exiting with status = 0
Jan 13 13:02:01 pid:26540 NOTICE: Agent process 30993 exited with status 0
```

DEMO 3 - EXIF Extraction Demo - 1 of 2

overload the acPostProcForPut() policy enforcement point

```
# remove prepended XXX in core.py DEMO 3 function name
def acPostProcForPut(rule_args, callback):
```

remove, then iput the same file into iRODS

```
$ irm -rf puppies
$ iput puppies.jpg
```

rodsLog:

```
Jan 13 13:02:50 pid:26540 NOTICE: Agent process 31039 started for puser=rods and cuser=rods from #.#.#
Jan 13 13:02:50 pid:31039 NOTICE: readAndProcClientMsg: received disconnect msg from client
Jan 13 13:02:50 pid:31039 NOTICE: Agent exiting with status = 0
Jan 13 13:02:50 pid:26540 NOTICE: Agent process 31039 exited with status 0

Jan 13 13:02:50 pid:26540 NOTICE: Agent process 31046 started for puser=rods and cuser=rods from #.#.#
Jan 13 13:02:50 pid:31046 NOTICE: writeLine: inString = /var/lib/irods/iRODS/Vault/home/rods/puppies.j
Jan 13 13:02:50 pid:31046 NOTICE: writeLine: inString = /tempZone/home/rods/puppies.jpg
Jan 13 13:02:51 pid:31046 NOTICE: writeLine: inString = PYTHON - acPostProcForPut() EXIF complete
Jan 13 13:02:51 pid:31046 NOTICE: readAndProcClientMsg: received disconnect msg from client
Jan 13 13:02:51 pid:31046 NOTICE: Agent exiting with status = 0
Jan 13 13:02:51 pid:26540 NOTICE: Agent process 31046 exited with status 0
```

DEMO 3 - EXIF Extraction Demo - 2 of 2

list the newly extracted and associated metadata

```
$ imeta ls -d puppies.jpg
AVUs defined for dataObj puppies.jpg:
<snip>
----
attribute: DateTimeOriginal
value: 2012:10:31 18:37:26
units:
----
attribute: ExifImageHeight
value: 518
units:
----
attribute: ExifImageWidth
value: 777
units:
----
attribute: ExposureTime
value: (1, 40)
units:
----
attribute: Flash
value: 16
units:
----
<snip>
----
attribute: Make
value: Canon
units:
----
attribute: MeteringMode
value: 5
units:
----
attribute: Model
value: Canon EOS REBEL T3i
units:
----
attribute: Orientation
value: 1
units:
----
<snip>
```

verify with third-party jhead

```
$ jhead puppies.jpg
File name    : puppies.jpg
File size    : 95891 bytes
File date    : 2016:01:11 22:16:02
Camera make  : Canon
Camera model : Canon EOS REBEL T3i
Date/Time    : 2012:10:31 18:37:26
Resolution   : 777 x 518
Flash used   : No
Focal length : 18.0mm  (35mm equivalent: 188mm)
CCD width    : 3.45mm
Exposure time: 0.025 s  (1/40)
Aperture     : f/3.5
ISO equiv.   : 2500
Whitebalance : Auto
Metering Mode: pattern
Exposure     : program (auto)
JPEG Quality : 94
```

Discussion

- Powerful abstraction
- Invites new developers
- Provides migration path
- Requires new documentation
- Requires broader security model
- Requires careful consideration

Questions?

Hao Xu, Jason Coposky, Ben Keller, Terrell Russell (2015).
Pluggable Rule Engine Architecture.
iRODS User Group Meeting 2015 Proceedings, pp. 29-34
http://irods.org/wp-content/uploads/2015/09/UMG2015_P.pdf

irods.org
github.com/irods
@irods

Terrell Russell
@terrellrussell