A Hello World iRODS Micro-Service

Ashu Guru

Introduction

This is a write up of creating an iRODS micro-service; the goal I had for this task was to identify and understand the steps & configurations involved in writing a micro-service and seeing it in action - for details regarding iRODS please refer to documentation at https://www.iRODS.org/. Based on my goal - the micro-service that I wrote is very simplistic (it simply writes a hello world message to a system log) however it provides you an overview of steps that will be involved in writing a useful micro-service. Also, please note that the information that is contained in this document is my understanding of iRODS after spending less than 4 days of reading and experimenting with it, which means that the information that is contained here is based on the limited knowledge I have gained so far.

Before I document the configurations, and codes involved in creating and registering a new micro-service let's look at figure 1. Figure 1 shows a high level diagram of an invocation of a micro-service by the iRODS rules engine. One way of looking at the micro-service and the iRODS rule engine is to think of it as a event based triggering system that can perform 'operations' on the data objects, and/or the iRODS system, and/or external resources based on events and its details, the iRODS micro-services are the means to customize and perform such *operations*. These micro-services are registered in rule definitions and the rule engine invokes them based on the condition specified for that rule. The events that may cause the invocation of the micro-service could be uploading of a data object, or deletion of a data object, etc. For a list of places in the iRODS workflow where a micro-service may be triggered please visit: https://www.irods.org/index.php/Default_iRODS_Rules.

Also you may refer to https://www.iRODS.org/index.php/Rule_Engine for a detailed diagram of a micro-service invocation.

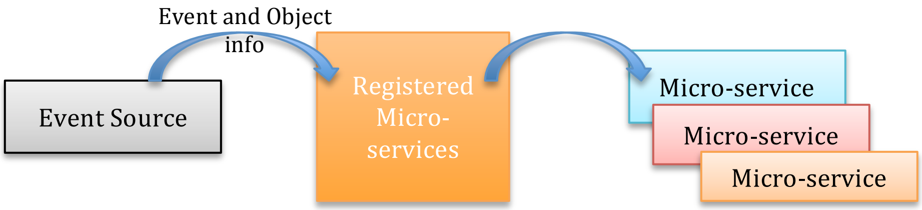


Figure: 1 Micro-service event workflow Overview

Figure 2 below shows the communication between the iRODS rule engine and the micro-service. A simplistic view of the communication layers is that the rule engine calls a defined C procedure, which exposes its functionality through an interface (commonly prefixed with msi). The arguments are passed through a structure msParam_t that is defined below:

```
typedef struct MsParam {
  char *label;
  char *type;        /* this is the name of the packing instruction in
                * rodsPackTable.h */
  void *inOutStruct;
  bytesBuf_t *inpOutBuf;
} msParam_t;
```

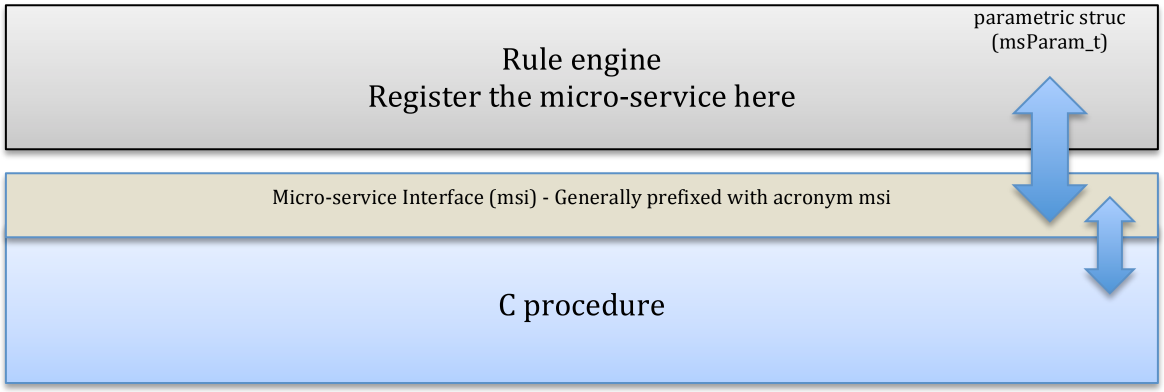

parametric struc
(msParam_t)
Rule engine
Register the micro-service here
Micro-service Interface (msi) - Generally prefixed with acronym msi
C procedure

Figure 2: Micro-service overview

Writing the micro-service

Lets now look at the steps for writing the micro-service:

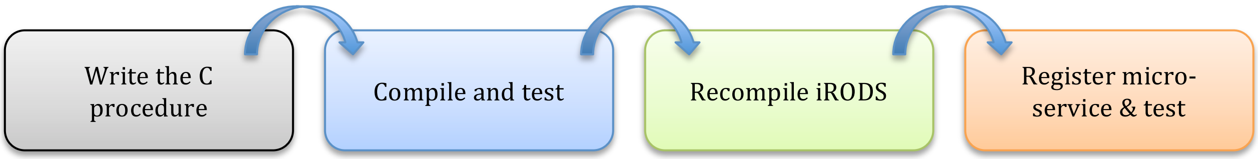


Figure 3: Steps involved in creating a micro-service

Write the C procedure

The C code below (lets call it test.c) has a function *writemessage* that writes a message to the system log. There is an interface to the function named *msiWritemessage* which exposes the method. The msi function takes a list of arguments of type *msParam_t* and a last argument of type *ruleExecInfo_t* for the result of operation.

```
#include <stdio.h>
#include <unistd.h>
#include <syslog.h>
#include <string.h>
#include "apiHeaderAll.h"


void writemessage(char arg1[], char arg2[]);
int msiWritemessage(msParam_t *mParg1, msParam_t *mParg2,  ruleExecInfo_t *rei);


void writemessage(char arg1[], char arg2[]) {
    openlog("slog", LOG_PID|LOG_CONS, LOG_USER);
    syslog(LOG_INFO, "%s %s from micro-service", arg1, arg2);
    closelog();
}

int msiWritemessage(msParam_t *mParg1, msParam_t *mParg2,  ruleExecInfo_t *rei)
{
  char *in1;
  int *in2;
  RE_TEST_MACRO ("    Calling Procedure");
  // the above line is needed for loop back testing using irule -i option
  if ( strcmp( mParg1->type, STR_MS_T ) == 0 )
  {
                in1 = (char*) mParg1->inOutStruct;
  }
  if ( strcmp( mParg2->type, INT_MS_T ) == 0 )
  {
                in2 = (int*) mParg2->inOutStruct;
  }
  writemessage(in1, in1);
  return rei->status;
}
```

Next we will make a folder structure in the *module* folder of iRODS home for placing this micro-service.

```
cd ~irods
mkdir modules/HCC
cd modules/HCC

mkdir microservices
mkdir rules
mkdir lib
mkdir clients
mkdir servers

mkdir microservices/src
mkdir microservices/include
mkdir microservices/obj
```

Next copy a few files from the properties module and modify them to fit the test.c micro-service

```
cp ../properties/Makefile .
cp ../properties/info.txt .
```

Listed below is my working copy of Makefile and the info.txt

```
#Makefile

ifndef buildDir
buildDir = $(CURDIR)/../..
endif

include $(buildDir)/config/config.mk
include $(buildDir)/config/platform.mk
include $(buildDir)/config/directories.mk
include $(buildDir)/config/common.mk

#
# Directories
#
MSObjDir =              $(modulesDir)/HCC/microservices/obj
MSSrcDir =              $(modulesDir)/HCC/microservices/src
MSIncDir =              $(modulesDir)/HCC/microservices/include

# Source files

OBJECTS =               $(MSObjDir)/test.o


# Compile and link flags
#
INCLUDES +=             $(INCLUDE_FLAGS) $(LIB_INCLUDES) $(SVR_INCLUDES)
CFLAGS_OPTIONS := $(CFLAGS) $(MY_CFLAG)
CFLAGS = $(CFLAGS_OPTIONS) $(INCLUDES) $(MODULE_CFLAGS)

.PHONY: all server client microservices clean
.PHONY: server_ldflags client_ldflags server_cflags client_cflags
.PHONY: print_cflags

# Build everytying
all:        microservices
            @true

# List module's objects and needed libs for inclusion in clients
client_ldflags:
            @true

# List module's includes for inclusion in the clients
client_cflags:
            @true

# List module's objects and needed libs for inclusion in the server
server_ldflags:
            @echo $(OBJECTS) $(LIBS)

# List module's includes for inclusion in the server
server_cflags:
            @echo $(INCLUDE_FLAGS)
```

info.txt

```
Name:               HCC
Brief:              HCC Test microservice
Description:        HCC Test microservice.
Dependencies:
Enabled:   yes
Creator:   Ashu Guru
Created:   December 2011
License:   BSD
```

As a next step we will define the micro-service header and micro-service table files in folder *microservices/include*. Since there is no header for this code I have that file all commented out and in the micro-service table is the entry for the table definition that is required to configure the micro-service with iRODS. The specifics to note are that the first argument is the label of the micro-service, the second argument is the count of *input* (so do not count the ruleExecInfo _t argument)arguments for your msi interface to the C function and the third argument is the name of the msi interface function.

File *microservices/include/microservices.table*

```
{ "msiWritemessage",2,(funcPtr) msiWritemessage },
```

We are done as far as the micro-service codes and its configuration is concerned.

Following is the directory tree structure for the *HCC* module that I have written:

```
bash-4.1$ pwd
/opt/iRODS/modules
bash-4.1$ tree HCC
HCC
 ├── clients
 ├── info.txt
 ├── lib
 ├── Makefile
 ├── microservices
 │   ├── include
 │   │   ├── microservices.header
 │   │   ├── microservices.table
 │   ├── obj
 │   └── src
 │     ├── test.c
 ├── rules
 └── servers
```

Next we make an entry for enabling the module in the file ~irods/config/config.mk so that the iRODS Makefile can pick it up. To do this open the file and add the module folder name (in my case HCC) to the variable MODULES.

Compile and test

We are now ready to compile and test our micro-service.

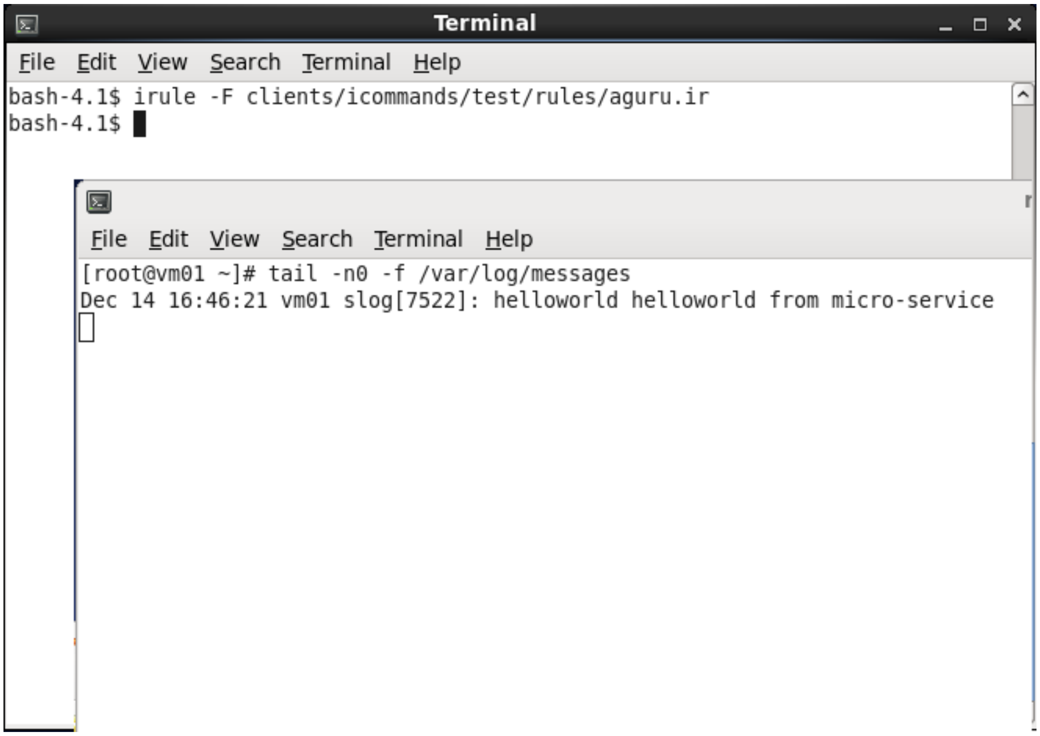
```
cd ~irods/modules/<YOURMODULENAME>
make
```

This should result in creation of an object file in the micro-service/obj folder.

Now we will test the micro-service manually, to accomplish this task we will create a client side rule file in the folder ~irods/ clients/icommands/test/rules
I have named the file aguru.ir and following are the contents of the file:

```
aguruTest||msiWritemessage(*A,*B)|nop
*A=helloworld%*B=testing
```

The above rule file now contains two lines the first line is the rules definition and the second line is the input parameters. To test this I will next invoke the micro-service and it should then write a message to the system log.


Terminal
File Edit View Search Terminal Help

```
bash-4.1$ irule -F clients/icommands/test/rules/aguru.ir
bash-4.1$
```

File Edit View Search Terminal Help

```
[root@vm01 ~]# tail -n0 -f /var/log/messages
Dec 14 16:46:21 vm01 slog[7522]: helloworld helloworld from micro-service
```

Recompile iRODS

First we need to make the entries for the headers and the msi table in the iRODS main micro-service action table (file ~irods/server/re/include/reAction.h). This should be done using the following commands:

```
rm server/re/include/reAction.h
make reaction
```

However, I had to manually add the line:

```
int msiWritemessage(msParam_t *mParg1, msParam_t *mParg2,  ruleExecInfo_t *rei);
```

to the file server/re/include/reAction.h file.

Now we are ready to recompile iRODS

```
cd ~irods
make test_flags
make modules
./irodsctl stop
make clean
make
./irodsctl start
./irodsctl status
```

Register Micro-service and Test

In this step we define a rule that will trigger the micro-service when a new data object is uploaded to iRODS. Open the file ~irods/server/config/reConfigs/core.re and add the following line below the *Test Rules* section.

```
acPostProcForPut {msiWritemessage("HelloWorld","String 2"); }
```

That is it… if you now put (iput) any file to iRODS there is a message that is added to the /var/log/messages file. Please note that we are not filtering this rule it is a catchall and applies to all put events.

References:
https://www.irods.org/
http://www.wrg.york.ac.uk/iread/compiling-and-running-irods-with-micros-services
http://technical.bestgrid.org/index.php/IRODS_deployment_plan