Author: Renaissance Computing Institute (RENCI)
Version: 3.0.1
Date: 2013-10-31

Table of Contents

1 Prerequisites

This tutorial covers two built-in examples: "Hello World!" and "CURL Get".

To run this tutorial:

1. Download E-iRODS binary package and Development Tools from http://www.eirods.org/download
2. Install E-iRODS
3. Install the E-iRODS Development Tools
4. Install package libboost1.xx-dev
5. If you want to run the curl based example, install the libcurl and libcurl-dev packages

After installing the E-iRODS dev tools, make a test directory on your system and copy the contents of /usr/share/eirods/examples/microservices into it. This will be our current working directory for the rest of the tutorial.

2 About microservices and microservice plugins

An iRODS microservice is a C function which takes as parameters:

- One argument of type ruleExecInfo_t* (required)
- Any number of arguments of type msParam_t* (optional)

Input and output values of a microservice are passed through its msParam_t* arguments.

A set of helper functions is commonly used in microservices to interface with the generic msParam_t* type:

- parseMspForXxx() for inputs
- fillXxxInMsParam() for outputs

Where 'Xxx' varies depending on the type of parameter ('Str', 'Int', 'DataObjInp', etc..).

For more information on iRODS microservices and related helper functions see https://www.irods.org/index.php/iRODS_Introduction sections 6 and 8:

- https://www.irods.org/index.php/iRODS_Introduction#iRODS_Micro-services
- https://www.irods.org/index.php/iRODS_Introduction#Extending_iRODS

E-iRODS allows microservices to be dynamically loaded as plugins, without the need to rebuild the iRODS server.

3 Generic template for writing a microservice plugin

```
// =-=-=-=-=-=-=-
// E-iRODS Includes
#include "msParam.h"
#include "reGlobalsExtern.h"
#include "eirods_ms_plugin.h"

// =-=-=-=-=-=-=-
// STL Includes
#include <iostream>

extern "C" {

    // =-=-=-=-=-=-=-
    // 1. Write a standard issue microservice
    int my_microservice( ..., ruleExecInfo_t* _rei ) {
        [...]
        return 0;
    }

    // =-=-=-=-=-=-=-
    // 2.  Create the plugin factory function which will
    //     return a microservice table entry
    eirods::ms_table_entry* plugin_factory() {

        // =-=-=-=-=-=-=-
        // 3. Allocate a microservice plugin which takes the number of function
        //    params as a parameter to the constructor, not including _rei. With
        //    N as the total number of arguments of my_microservice() we would have:
        eirods::ms_table_entry* msvc = new eirods::ms_table_entry( N-1 );

        // =-=-=-=-=-=-=-
        // 4. Add the microservice function as an operation to the plugin
        //    the first param is the name / key of the operation, the second
        //    is the name of the function which will be the microservice
        msvc->add_operation( "my_microservice", "my_microservice" );

        // =-=-=-=-=-=-=-
        // 5. Return the newly created microservice plugin
        return msvc;
    }

}; // extern "C"
```

4 "Hello World!" as a microservice plugin

Given the above template we can write a simple microservice plugin that returns a string to the client. That string needs to be passed out via a microservice parameter (msParam_t*). A simple microservice plugin would then look like this (`eirods_hello.cpp`):

```
// =-=-=-=-=-=-=-
// E-iRODS Includes
#include "msParam.h"
#include "reGlobalsExtern.h"
#include "eirods_ms_plugin.h"
```

```
// =-=-=-=-=-=-=-
// STL Includes
#include <iostream>

extern "C" {

    // =-=-=-=-=-=-=-
    // 1. Write a standard issue microservice
    int eirods_hello( msParam_t* _out, ruleExecInfo_t* _rei ) {
        std::string my_str = "Hello World!";
        fillStrInMsParam( _out, my_str.c_str() );
        return 0;
    }

    // =-=-=-=-=-=-=-
    // 2. Create the plugin factory function which will
    //    return a microservice table entry
    eirods::ms_table_entry* plugin_factory() {

        // =-=-=-=-=-=-=-
        // 3. Allocate a microservice plugin which takes the number of function
        //    params as a parameter to the constructor, not including _rei.
        eirods::ms_table_entry* msvc = new eirods::ms_table_entry( 1 );

        // =-=-=-=-=-=-=-
        // 4. Add the microservice function as an operation to the plugin
        //    the first param is the name / key of the operation, the second
        //    is the name of the function which will be the microservice
        msvc->add_operation( "eirods_hello", "eirods_hello" );

        // =-=-=-=-=-=-=-
        // 5. Return the newly created microservice plugin
        return msvc;
    }

}; // extern "C"
```

To run this example from your test directory type:

```
$ make hello
```

This should create a shared object: `libeirods_hello.so`

Copy `libeirods_hello.so` to the microservices plugin directory (as eirods):

```
$ sudo -u eirods cp -f libeirods_hello.so /var/lib/eirods/plugins/microservices/
```

Now that you have "loaded" your new microservice plugin you can test it with its corresponding rule:

```
$ irule -F eirods_hello.r
```

5 "CURL Get" as a microservice plugin

In this second example we are using libcurl to make a GET request and write the result to an E-iRODS object. For the full source see `eirods_curl_get.cpp`.

While the previous example simply returned an arbitrary string, this microservice is manipulating iRODS content and therefore needs an iRODS connection context. This connection context is provided to our microservice via its ruleExecInfo_t* parameter and is required when making iRODS API calls to create and write to iRODS objects. In our example this is done by the CURL write function that writes blocks of data to a new iRODS object, using `rsDataObjCreate()`, `rsDataObjWrite()`, and `rsDataObjClose()`. For this reason the iRODS connection context needs to be passed all the way to the CURL write function.

To keep our microservice code concise we define an irodsCurl object that maintains an iRODS connection context and a (reusable) CURL handler. We can then use the `irodsCurl::get()` method to make one or more requests to remote objects, e.g:

```
myCurl.get("www.example.com/file1.html", "/tempZone/home/example/file1.html");
myCurl.get("www.example.com/file2.html", "/tempZone/home/example/file2.html");
myCurl.get("www.example.com/file3.html", "/tempZone/home/example/file3.html");
etc...
```

To run this example from your test directory type:

```
$ make curl_get
```

This should create a shared object: `libeirods_curl_get.so`

Copy `libeirods_curl_get.so` to the microservices plugin directory (as eirods):

```
$ sudo -u eirods cp -f libeirods_curl_get.so /var/lib/eirods/plugins/microservices/
```

Edit the rulefile `eirods_curl_get.r` to make sure that the destination object path (*dest_object) is valid in your environment

Run the rulefile:

```
$ irule -F eirods_hello.r
```

Check if the new object is there:

```
$ ils -L PATH_OF_DEST_OBJECT
$ iget PATH_OF_DEST_OBJECT -
```

Note the - at the end of the iget command. This simply puts the retrieved file into stdout and will print the file contents to your screen.