iRODS Client: C++ REST API

Jason Coposky
Renaissance Computing Institute
(RENCI)
UNC Chapel Hill
jasonc@renci.org

Terrell Russell
Renaissance Computing Institute
(RENCI)
UNC Chapel Hill
unc@terrellrussell.com

ABSTRACT

The iRODS C++ REST API has been discussed for years, but is now ready to show to others. This paper will explore the different aspects of what is possible with the REST API today and invite discussion about what else it may need.

Keywords

iRODS, data management, administration, REST, API

INTRODUCTION

iRODS itself is a protocol and an API, by default running on port 1247/1248, with a server implementation written in C++. Since iRODS was designed, the world has largely standardized on HTTP REST as a means to quickly set up and maintain connections between different services on the Internet. This iRODS REST API is written in C++, designed to be run alongside an iRODS Server, and provides a wrapper around the iRODS protocol, giving new accessibility and development speed to REST-familiar developers of web services and other applications.

It runs under the same linux service account as the iRODS Server, accesses `/etc/irods/server_config.json`, and uses the active authenticated `rodsadmin` iRODS account.

This initial REST API implementation requires to be run alongside an iRODS Server v4.2.0 or greater. It provides endpoints for `/access`, `/admin`, `/auth`, `/configuration`, `/list`, `/query`, `/stream`, and `/zone_report`. Additional functionality may be added if community usage generates new use cases.

MOTIVATION

The iRODS community is diverse and demands relatively unique applications depending on the science or industry domain (microscopy, genomics, agriculture, archives, etc.) or the specific roles of its users (instrument service account, metadata curator, policy administrator, etc.). These different use cases sometimes render general search and browse applications (like Metalnx[1]) too powerful and/or too confusing. Many organizations now have the capacity to develop their own specific, customized applications built with modern web-development frameworks, but these frameworks all assume available HTTP server endpoints with which to interact.

iRODS needs to provide these type of endpoints through an easy to use, easy to deploy, fast, and lightweight REST API.

An initial iRODS Consortium application to consume these endpoints is the new Zone Management Tool[2].

iRODS UGM 2021 June 8-11, 2021, Virtual

IMPLEMENTATION

This API is based on the iRODS C++ API and acts as a proxied mid-tier application layer between web applications and an iRODS server. It utilizes standard JSON Web Tokens (JWT)[3] for authentication and authorization. The REST API provides a single executable per endpoint which is designed to be compatible with containerized deployments.

This API is designed to follow the Hypermedia as the Engine of Application State (HATEOAS)[4] model where the client does not need to keep state and the API itself will share URLs in its responses dictating the next relevant links.

CONFIGURATION

The REST API provides an executable for each individual API endpoint. These endpoints may be grouped behind a reverse proxy in order to provide a single port for access.

Two configuration template files are placed in `/etc/irods` in a packaged deployment and need to be copied and edited before activation. The `/etc/irods/irods_client_rest_cpp.json.template` is the REST API template and `/etc/irods/irods_client_rest_cpp_reverse_proxy.conf.template` is an nginx[5] template.

/etc/irods/irods_client_rest_cpp.json

The REST API service relies on a configuration file in `/etc/irods` that dictates which port is used for each endpoint, as well as how many threads it can run concurrently and its network timeout in seconds.

```
{
    "irods_rest_cpp_access_server" : {
        "port" : 8080,
        "threads" : 4,
        "maximum_idle_timeout_in_seconds" : 10
    },
    "irods_rest_cpp_admin_server" : {
        "port" : 8087,
        "threads" : 4,
        "maximum_idle_timeout_in_seconds" : 10
    },
    "irods_rest_cpp_auth_server" : {
        "port" : 8081,
        "threads" : 4,
        "maximum_idle_timeout_in_seconds" : 10
    },
    "irods_rest_cpp_get_configuration_server" : {
        "port" : 8088,
        "threads" : 4,
        "maximum_idle_timeout_in_seconds" : 10,
        "api_key" : "default_api_key"
    },
    "irods_rest_cpp_put_configuration_server" : {
        "port" : 8089,
        "threads" : 4,
        "maximum_idle_timeout_in_seconds" : 10
    },
    "irods_rest_cpp_list_server" : {
```

```
        "port" : 8082,
        "threads" : 4,
        "maximum_idle_timeout_in_seconds" : 10
    },
    "irods_rest_cpp_query_server" : {
        "port" : 8083,
        "threads" : 4,
        "maximum_idle_timeout_in_seconds" : 10
    },
    "irods_rest_cpp_stream_get_server" : {
        "port" : 8084,
        "threads" : 4,
        "maximum_idle_timeout_in_seconds" : 10
    },
    "irods_rest_cpp_stream_put_server" : {
        "port" : 8085,
        "threads" : 4,
        "maximum_idle_timeout_in_seconds" : 10
    },
    "irods_rest_cpp_zone_report_server" : {
        "port" : 8086,
        "threads" : 4,
        "maximum_idle_timeout_in_seconds" : 10
    }
}
```

/etc/nginx/sites-available/irods_client_rest_cpp_reverse_proxy.conf

The nginx reverse proxy relies on a configuration file (sometimes deployed to `/etc/nginx/sites-available`) which defines how to route incoming paths to the separate REST API ports.

```
server {
    listen 80;

    add_header 'Access-Control-Allow-Origin' '*' always;
    add_header 'Access-Control-Allow-Headers' '*' always;
    add_header 'Access-Control-Allow-Methods' 'AUTHORIZATION,ACCEPT,GET,POST,OPTIONS,PUT,DELETE' always;

    location /irods-rest/1.0.0/access {
        if ($request_method = 'OPTIONS') {
            return 204;
        }
        proxy_pass http://localhost:8080;
    }

    location /irods-rest/1.0.0/admin {
        if ($request_method = 'OPTIONS') {
            return 204;
        }
        proxy_pass http://localhost:8087;
    }
```

```
location /irods-rest/1.0.0/auth {
    if ($request_method = 'OPTIONS') {
        return 204;
    }
    proxy_pass http://localhost:8081;
}

location /irods-rest/1.0.0/configuration {
    if ($request_method = 'OPTIONS') {
        return 204;
    }

    if ($request_method = GET ) {
        proxy_pass http://localhost:8088;
    }

    if ($request_method = PUT ) {
        proxy_pass http://localhost:8089;
    }
}

location /irods-rest/1.0.0/list {
    if ($request_method = 'OPTIONS') {
        return 204;
    }
    proxy_pass http://localhost:8082;
}

location /irods-rest/1.0.0/query {
    if ($request_method = 'OPTIONS') {
        return 204;
    }
    proxy_pass http://localhost:8083;
}

location /irods-rest/1.0.0/stream {
    if ($request_method = 'OPTIONS') {
        return 204;
    }

    if ($request_method = GET ) {
        proxy_pass http://localhost:8084;
    }

    if ($request_method = PUT ) {
        proxy_pass http://localhost:8085;
    }
}

location /irods-rest/1.0.0/zone_report {
```

```
        if ($request_method = 'OPTIONS') {
            return 204;
        }
        proxy_pass http://localhost:8086;
    }
}
```

AUTHENTICATION

This REST API relies on the use of JSON Web Tokens (JWT) to communicate identity, authentication information, authorization information, and in the future, role-based information.

A JWT is generated by invoking the `/auth` endpoint. This JWT must be reused by the client as an Authentication header for use with subsequent requests to other endpoints.

ENDPOINTS

The following endpoints describe the entirety of the REST API at this time. Additional endpoints or changes to these initial endpoints could change the way an application may have to interact with the API. This initial look at the REST API could change before version 1.0.0.

/access

This endpoint provides a service for the generation of an iRODS ticket to a given logical path (either a collection or a data object). A ticket can grant read or write permission to an otherwise anonymous user.

`POST` Parameters:

- path: The url-encoded logical path to a collection or data object for which access is desired

Example Usage:

```
curl -X POST -H "Authorization: ${TOKEN}" \
"http://localhost/irods-rest/1.0.0/access?path=%2FtempZone%2Fhome%2Frods%2Ffile0"
```

Returns:

An iRODS ticket token within the X-API-KEY header, and a URL for streaming the object.

```
{
  "headers": [
    "X-API-KEY: CS11B8C4KZX2BIl"
  ],
  "url": "/irods-rest/1.0.0/stream?path=%2FtempZone%2Fhome%2Frods%2Ffile0&offset=0&limit=33064"
}
```

/admin

The administration interface to the iRODS Catalog which allows the creation, removal and modification of users, groups, resources, and other entities within the zone.

`POST` Parameters:

- action: dictates the action to be taken (add, modify, or remove)
- target: the subject of the action: user, zone, resource, childtoresc, childfromresc, token, group, rebalance, unusedAVUs, specificQuery
- arg2: generic argument, could be user name, resource name, depending on the value of action and target
- arg3: generic argument, see above
- arg4: generic argument, see above
- arg5: generic argument, see above
- arg6: generic argument, see above
- arg7: generic argument, see above

Example Usage:

```
curl -X POST -H "Authorization: ${TOKEN}" \
"http://localhost/irods-rest/1.0.0/admin?action=add&target=resource&arg2=ufs0&
arg3=unixfilesystem&arg4=/tmp/irods/ufs0&arg5=&arg6=tempZone&arg7="
```

Returns:

"Success" or an iRODS exception

/auth

This endpoint provides an authentication service for the iRODS zone.

Currently only native iRODS authentication is supported, as `Basic` or `Native`.

`POST` Parameters:

- None

Example Usage:

```
export BASE64USERPASS=$(echo -n "rods:secret" | base64 -)
export TOKEN=$(curl -X POST -H "Authorization: Basic ${BASE64USERPASS}" \
"http://localhost:80/irods-rest/1.0.0/auth")
```

Returns:

An encrypted JWT which contains everything necessary to interact with the other endpoints. This token is expected in the Authorization header for the other endpoints.

/configuration

This endpoint will return a JSON structure holding the configuration for an iRODS server. This endpoint takes a known API key for authorization which is configured in `/etc/irods/irods_client_rest_cpp.json`.

`GET` Parameters:

Example Curl Command:

```
curl -X GET -H "X-API-KEY: ${API_KEY}" \
"http://localhost/irods-rest/1.0.0/configuration"
```

Returns:

A JSON array of objects whose key is the file name and whose contents is the configuration file.

```
{
    "host_access_control_config.json": {
        <SNIP>
    },
    "hosts_config.json": {
        <SNIP>
    },
    "irods_client_rest_cpp.json": {
        <SNIP>
    },
    "server_config.json": {
        <SNIP>
    }
}
```

This endpoint will write the url-encoded JSON to the specified files in `/etc/irods`.

`PUT` Parameters:

cfg - a url encoded JSON string of the format

```
[
    {
        "file_name": "test_rest_cfg_put_1.json",
        "contents" : {
            "key0" : "value0",
            "key1" : "value1"
        }
    },
    {
        "file_name": "test_rest_cfg_put_2.json",
        "contents" : {
            "key2" : "value2",
            "key3" : "value3"
```

```
        }
    }
]
```

Example Usage:

```
export CONTENTS="%5B%7B%22file_name%22%3A%22test_rest_cfg_put_1.json%22%2C
%20%22contents%22%3A%7B%22key0%22%3A%22value0%22%2C%22key1%22%20%3A%20
%22value1%22%7D%7D%2C%7B%22file_name%22%3A%22test_rest_cfg_put_2.json%22%2C
%22contents%22%3A%7B%22key2%22%20%3A%20%22value2%22%2C%22key3%22%20%3A
%20%22value3%22%7D%7D%5D"
curl -X PUT -H "Authorization: ${TOKEN}" \
"http://localhost/irods-rest/1.0.0/configuration?cfg=${CONTENTS}"
```

Returns:

None

/list

This endpoint provides a recursive listing of a collection, or stat, metadata, and access control information for a given data object.

Method : GET

Parameters: path : The url encoded logical path which is to be listed stat : Boolean flag to indicate stat information is desired permissions : Boolean flag to indicate access control information is desired metadata : Boolean flag to indicate metadata is desired offset : number of records to skip for pagination limit : number of records desired per page

Example Curl Command:

```
curl -X GET -H "Authorization: ${TOKEN}" \
"http://localhost/irods-rest/1.0.0/list?path=%2FtempZone%2Fhome%2Frods&stat=0&
permissions=0&metadata=0&offset=0&limit=100"
```

Returns:

A JSON structured response within the body containing the listing, or an iRODS exception

```
{
  "_embedded": [
    {
      "logical_path": "/tempZone/home/rods/subcoll",
      "type": "collection"
    },
    {
      "logical_path": "/tempZone/home/rods/subcoll/file0",
      "type": "data_object"
```

```
    },
    {
      "logical_path": "/tempZone/home/rods/subcoll/file1",
      "type": "data_object"
    },
    {
      "logical_path": "/tempZone/home/rods/subcoll/file2",
      "type": "data_object"
    },
    {
      "logical_path": "/tempZone/home/rods/file0",
      "type": "data_object"
    }
  ],
  "_links": {
    "first": "/irods-rest/1.0.0/list?path=%2FtempZone%2Fhome%2Frods&stat=0&
    permissions=0&metadata=0&offset=0&limit=100",
    "last": "/irods-rest/1.0.0/list?path=%2FtempZone%2Fhome%2Frods&stat=0&
    permissions=0&metadata=0&offset=UNSUPPORTED&limit=100",
    "next": "/irods-rest/1.0.0/list?path=%2FtempZone%2Fhome%2Frods&stat=0&
    permissions=0&metadata=0&offset=100&limit=100",
    "prev": "/irods-rest/1.0.0/list?path=%2FtempZone%2Fhome%2Frods&stat=0&
    permissions=0&metadata=0&offset=0&limit=100",
    "self": "/irods-rest/1.0.0/list?path=%2FtempZone%2Fhome%2Frods&stat=0&
    permissions=0&metadata=0&offset=0&limit=100"
  }
}
```


/query

This endpoint provides access to the iRODS General Query language, which is a generic query service for the iRODS catalog.

`GET` Parameters:

- query_string: A url-encoded general query
- query_limit: Number of desired rows
- row_offset: Number of rows to skip for paging
- query_type: Either 'general' or 'specific'

Example Usage:

```
curl -X GET -H "Authorization: ${TOKEN}" \
"http://localhost/irods-rest/1.0.0/query?query_limit=100&
row_offset=0&query_type=general&query_string=SELECT%20COLL_NAME%2C
%20DATA_NAME%20WHERE%20COLL_NAME%20LIKE%20%27%2FtempZone%2Fhome%2Frods%25%27"
```

Returns:

A JSON structure containing the query results


```
{
  "_embedded": [
    [
      "/tempZone/home/rods",
      "file0"
    ],
    [
      "/tempZone/home/rods/subcoll",
      "file0"
    ],
    [
      "/tempZone/home/rods/subcoll",
      "file1"
    ],
    [
      "/tempZone/home/rods/subcoll",
      "file2"
    ]
  ],
  "_links": {
    "first": "/irods-rest/1.0.0query?query_string=SELECT%20COLL_NAME%2C
    %20DATA_NAME%20WHERE%20COLL_NAME%20LIKE%20%27%2FtempZone%2Fhome%2Frods%25%27&
    query_limit=100&row_offset=0&query_type=general",
    "last": "/irods-rest/1.0.0query?query_string=SELECT%20COLL_NAME%2C
    %20DATA_NAME%20WHERE%20COLL_NAME%20LIKE%20%27%2FtempZone%2Fhome%2Frods%25%27&
    query_limit=100&row_offset=0&query_type=general",
    "next": "/irods-rest/1.0.0query?query_string=SELECT%20COLL_NAME%2C
    %20DATA_NAME%20WHERE%20COLL_NAME%20LIKE%20%27%2FtempZone%2Fhome%2Frods%25%27&
    query_limit=100&row_offset=0&query_type=general",
    "prev": "/irods-rest/1.0.0query?query_string=SELECT%20COLL_NAME%2C
    %20DATA_NAME%20WHERE%20COLL_NAME%20LIKE%20%27%2FtempZone%2Fhome%2Frods%25%27&
    query_limit=100&row_offset=0&query_type=general",
    "self": "/irods-rest/1.0.0query?query_string=SELECT%20COLL_NAME%2C
    %20DATA_NAME%20WHERE%20COLL_NAME%20LIKE%20%27%2FtempZone%2Fhome%2Frods%25%27&
    query_limit=100&row_offset=0&query_type=general"
  },
  "count": "4",
  "total": "4"
}
```


/stream

Stream data into and out of an iRODS data object

`PUT` and `GET` Parameters:

- path: The url-encoded logical path to a data object
- offset: The offset in bytes into the data object

- limit: The maximum number of bytes to read

Example `PUT` Usage:

```
curl -X PUT -H "Authorization: ${TOKEN}" -d"This is some data" \
"http://localhost/irods-rest/1.0.0/stream?
path=%2FtempZone%2Fhome%2Frods%2FfileX&offset=0&limit=1000"
```

Returns:

Nothing, or iRODS Exception

Example `GET` Usage:

```
curl -X GET -H "Authorization: ${TOKEN}" \
"http://localhost/irods-rest/1.0.0/stream?
path=%2FtempZone%2Fhome%2Frods%2FfileX&offset=0&limit=1000"
```

Returns:

The data requested in the body of the response

/zone_report

Requests a JSON formatted iRODS Zone report, containing all configuration information for every server in the zone.

`POST` Parameters:

- None

Example Usage:

```
curl -X POST -H "Authorization: ${TOKEN}" \
"http://localhost/irods-rest/1.0.0/zone_report"
```

Returns:

JSON formatted Zone Report

```
{
  "schema_version": "file:///var/lib/irods/configuration_schemas/v3/zone_bundle.json",
  "zones": [
    {
    <snip>
    }]
}
```

FUTURE WORK

This initial look at the new iRODS C++ REST API covers the majority of our use cases and design goals. We expect there to be some changes as we become more familiar with JWT usage and deployment patterns. Additional functionality is expected to grow around configuration management, as iRODS has never allowed outside processes to manipulate its configuration before. Handling zone-wide configuration in a single location has never been possible, so care must be taken to make sure there are sufficient protections in place (dry-run, rollback, etc.).

SUMMARY

The iRODS C++ REST API can now provide new access and increased speed of client development to modern web development frameworks. The Zone Management Tool (ZMT) is being built alongside this new API and the needs of each are driving the development process.

The functionality included in this first release is sufficient for full-featured applications to be rapidly built and deployed for the public, or for specific internal positions in large organizations.

REFERENCES

[1] Zhou, Bo; Draughn, Kory; Coposky, Jason; Russell, Terrell; Conway, Mike; iRODS Client: Metalnx 2.4.0 with GalleryView (2021)
`https://irods.org/uploads/2021/Zhou-iRODS-Metalnx_2.4.0_with_GalleryView-slides.pdf`

[2] Zhou, Bo; Russell, Terrell; iRODS Client: Zone Management Tool (ZMT) (2021).
`https://irods.org/uploads/2021/Zhou-iRODS-Zone_Management_Tool_ZMT-paper.pdf`

[3] Jones, M.; Bradley, J.; Sakimura, N.; JSON Web Token (JWT): RFC 7519 (2015).
`https://datatracker.ietf.org/doc/html/rfc7519`

[4] Hypermedia as the Engine of Application State (HATEOAS). `https://en.wikipedia.org/wiki/HATEOAS`

[5] nginx: HTTP and reverse proxy server. `http://nginx.org/en/`