NFSRODS: Presenting iRODS as NFSv4.1

Kory Draughn
Renaissance Computing Institute (RENCI)
UNC Chapel Hill
korydraughn@renci.org

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

Alek Mieczkowski
Renaissance Computing Institute (RENCI)
UNC Chapel Hill
info@irods.org

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

Mike Conway
NIH / NIEHS
mike.conway@nih.gov

ABSTRACT

NFSRODS[1] is a new iRODS[2] client that presents the iRODS logical namespace as NFSv4.1[3]. This allows administrators to expose the iRODS namespace, for both reads and writes, to any software or users who do not know how to speak the iRODS protocol. Existing scripts, tools, or hardware can traverse the mountpoint as expected, but while doing so, they also invoke all the server-side policy that iRODS provides and enforces. The v0.8.0 release has preliminary iRODS multi-owner access mapped to traditional Unix permissions, but will be updated for NFSv4 ACLs before v1.0.

Keywords

iRODS, client, NFS, NFSv4, data management

INTRODUCTION

NFSRODS v0.8.0 is informed by earlier work from Brazil in 2016. NFS-RODS[4] reported that the mapping from NFS calls and attributes to iRODS attributes was the hardest part. In the end, this NFSv3[5] implementation was not expressive enough to present a lossless interface into the iRODS namespace. iRODS needed an NFSv4 server to work with before more work could be done.

As a summer project in 2018, the iRODS Consortium began an investigation and early coding for what would become today's NFSRODS. It was envisioned to be built atop NFS4J[6] and be a pure Java application to be run in a JVM, probably within a Docker[7] container. The summer ended with a prototype that included Kerberos[8] for authentication and handled the protocol translation from NFSv4.1 to iRODS pretty well. Deploying this version was very complicated and required deep knowledge of Kerberos internals and configuration.

It was decided to trust the deployment environment for authentication and remove the Kerberos layer. Deployment became much simpler and early testing with stakeholders in enterprise environments suggested this was the correct path forward.

At the time of publication, the NFSRODS server is released as v0.8.0 and provides a lossy permission mapping for multiple iRODS owners into the standard traditional Unix permission model.

Future work will use NFSv4 ACLs to provide a lossless bidirectional mapping between the multiple ownership and permission models of NFSv4.1 and iRODS. This will be released as NFSRODS version 1.0.

iRODS UGM 2019 June 25-28, 2019, Utrecht, Netherlands

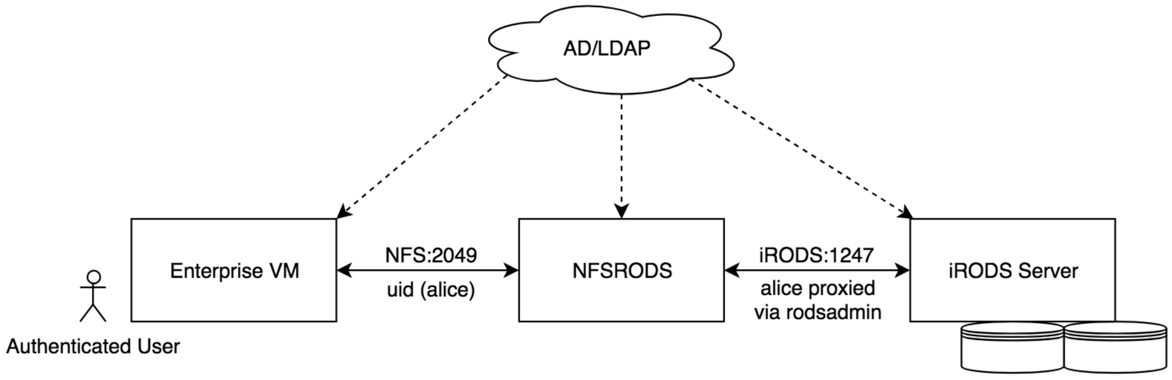

Figure 1. NFSRODS assumes an authenticated user without sudo access within the Enterprise VM.

ARCHITECTURE

The core of NFSRODS has two components. The NFS server side of NFSRODS is provided by NFS4J and is largely lifted directly from the open source project. NFS4J has a plugin architecture for its VirtualFileSystem and allows for other technologies to provide the filesystem interface. The iRODS Jargon client library[9] is used to implement an iRODS VirtualFileSystem. Jargon implements the iRODS protocol and communicates as an iRODS client.

With Kerberos removed, the security model of NFSRODS deployment makes a few assumptions about its environment. First, the usernames and UIDs must be consistent from the mountpoint, to the NFSRODS server, to within the iRODS catalog. The NFS connection between the mountpoint and NFSRODS communites which user is requesting access by unix UID. If `alice` (UID 509) makes a request to `ls` within the mountpoint, the NFSRODS server sees a request from UID 509 only. The NFSRODS server must be able to map the incoming UID to an iRODS username. This is done by the OS and uses the standard `/etc/passwd` and `/etc/shadow` files. Therefore, these must be kept in sync across the different machines in the system. It is assumed that this will be handled via external systems, most usually LDAP. The NFSRODS server maps the incoming request to an iRODS request which uses the matching username. It is assumed that the mechanism keeping the UIDs and usernames consistent is also keeping the list of users within the iRODS catalog consistent.

With this model, it is very important to note that any user with `sudo` rights on the Enterprise VM can become any other user, and therefore gain access to iRODS as that other user. It is recommended that there be no `sudo` rights available on the Enterprise VM where the mountpoint is accessible to the end user.

PERMISSIONS

Traditional Unix permissions[10] are based on rights granted to the three classes of `owner`, `group`, and `world`. Settings for each of these three classes can be `read`, `write`, `execute`, or none. A particular file can have one owner, be part of one group, and then have additional permissions set through the `world` class. This is a very rich and well-understood mechanism for sharing rights to files.

iRODS has a multi-owner model with a linear permission system. Permissions can be set to `OWN`, `WRITE`, `READ`, or none. `OWN` has all the rights of `WRITE` and can delete files and set permissions for others. `WRITE` has all the permissions of `READ` and can write the data or metadata for a data object. `READ` can see the contents of a data object.

Mapping between these two different permissions systems is not straightforward and has proven the most complex part of building NFSRODS.

iRODS Permission	Collection as Directory	Data Object as File
OWN	`drwx-----x`	`-rw-------`
WRITE	`d--x---rwx`	`-------rw-`
READ	`d--x---r-x`	`-------r--`
NULL	`d--x-----x`	`----------`

Table 1. NFSRODS v0.8.0 Mapping of iRODS Permissions to traditional Unix permissions

NFSRODS v0.8.0's mapping of iRODS Permissions to traditional Unix permissions is shown in Table 1. If an iRODS Collection is viewed from within an NFSRODS mountpoint, the directory is shown with the current user as Unix `owner` if the user is **an** iRODS owner. If this is the case, then the traditional Unix permissions are shown with full permissions `rwx`. If an iRODS data object is viewed from within an NFSRODS mountpoint, then the traditional Unix permissions are shown with permissions `rw`.

Non-owner (aka `WRITE` and `READ`) permissions are mapped, per viewer, as world permissions. Collections are always mapped with the `execute` bit for world, to allow traversal through the namespace.

This mapping works for non-admin and non-sysadmin users of NFSRODS. Access is granted correctly and disallowed correctly across the various combinations of multi-ownership.

However, this presentation of iRODS permissions through the use of `world` permissions proved too alarming and/or too strange for enterprise sysadmins and became the main impetus for moving towards NFSRODS v1.0 with NFSv4 ACLs capable of expressing multi-ownership.

Other limitations of this permissions system include a lack of `chmod` and no workable mappings for groups. The projection of iRODS permissions out to the NFSv4.1 mountpoint works, but there is no way for a file owner to set permissions for others from within a mountpoint and be reflected back into the iRODS Catalog. As a preliminary release, group permissions were not mapped out to the mountpoint at all. Both of these will be addressed with NVSv4 ACLs in NFSRODS v1.0.

USAGE

Deployment of NFSRODS v0.8.0 requires some preparation and then three steps.

The preparation includes making sure that the necessary user UIDs and usernames are available for the different components (Enterprise VM, NFSRODS server, and within the iRODS Catalog). The three steps include configuration, the `docker run` command, and setting up the mountpoint.

Configuration

Configuration for NFSRODS includes three configuration files, two of which do not need changes from the distributed examples. The `exports` and `log4j.properties` files can be used as is.

The `server.json` file needs to be updated to point to the correct iRODS server:

```
{
    // This section defines options needed by the NFS server.
    "nfs_server": {
        // The port number within the container to listen for NFS requests.
        "port": 2049,

        // The path within iRODS that will represent the root collection.
        // We recommend setting this to the zone. Using the zone as the root
        // collection allows all clients to access shared collections and data
        // objects outside of their home collection.
        "irods_mount_point": "/tempZone",

        // The refresh time (in minutes) for cached user information.
        "user_information_refresh_time_in_minutes": 60,

        // The refresh time (in milliseconds) for cached stat information.
        "file_information_refresh_time_in_milliseconds": 1000
    },

    // This section defines the location of the iRODS server being presented
    // by NFSRODS. The NFSRODS server can only be configured to present a single zone.
    "irods_client": {
        "host": "hostname",
        "port": 1247,
        "zone": "tempZone",

        // Because NFS does not have any notion of iRODS, you must define the
        // target resource for new data objects.
        "default_resource": "demoResc"
    },

    // An administrative iRODS account is required to carry out each request.
    // The account specified here is used as a proxy to connect to the iRODS
    // server. iRODS will still apply policies based on the client's account,
    // not the proxy account.
    "irods_proxy_admin_account": {
        "username": "rods",
        "password": "rods"
    }
}
```

The `nfs_server` section of the configuration file defines the settings for the NFSv4 side of NFSRODS. This includes the `port` number to expose as NFS (default 2049), the `irods_mount_point` to define how deep within iRODS the mount-point will expose the virtual filesystem, and some cache settings (`user_information_refresh_time_in_minutes` and `file_information_refresh_time_in_milliseconds`) for how long the NFSRODS server will keep a local copy of information found from the underlying unix system or the iRODS catalog.

The `irods_client` section of the configuration file defines the settings for the iRODS client side of NFSRODS (`host`, port, and zone). The default_resource setting will define where any newly created files within the mountpoint are physically created within iRODS.

NFSRODS occasionally needs to take action within iRODS that it would not be able to take without a higher privilege level. In these cases, NFSRODS uses the proxy mechanism of iRODS to request actions on behalf of the requesting user. The irods_proxy_admin_account is used to configure a rodsadmin username and password.

Docker

Starting NFSRODS requires a single docker run command of the form:

```
$ docker run -d --name nfsrods \
             -p <public_port>:2049 \
             -v </full/path/to/nfsrods_config>:/nfsrods_config:ro \
             -v /etc/passwd:/etc/passwd:ro \
             -v /etc/shadow:/etc/shadow:ro \
             nfsrods
```

The options launch the image known as nfsrods, put the container into daemon mode, and define the name of the running container (nfsrods), the port mapping from the outside world into the container, the volume mount to the configuration files, and the volume mounts of the host system's /etc/passwd and /etc/shadow files.

Restarting the NFSRODS server will not affect existing mountpoints other than the requirement to re-fetch any lost cache information.

Mountpoint

Once the NFSRODS server is running, the standard mount command can be used to mount the remote filesystem and provide a location for regular users to get access to the iRODS namespace:

```
$ sudo mkdir <mount_point>
$ sudo mount -o sec=sys,port=<public_port> <hostname>:/ <mount_point>
```

Note the hostname is the hostname where NFSRODS is running and the :/ after the hostname express to the mount command to mount the entire namespace provided by NFSRODS.

If you do not receive any errors after mounting, then a unix user with a properly mapped UID and username should be able to access the mount point like so:

```
$ cd <mount_point>/path/to/collection_or_data_object
```

FUTURE WORK

NFSRODS v0.8.0 is a work in progress. It is known that the lossy nature of the mapping between the iRODS Permission Model and traditional Unix permissions is insufficient for enterprise usage.

In addition to the permissions model, NFSRODS needs a test suite that expresses its intended usage, reduced debug logging, as well as performance measurements to characterize its overhead.

A lossless mapping between iRODS and NFSv4 ACLs with multi-ownership has been developed and will be implemented by the end of 2019. It is expected that NFSRODS v1.0 will include the lossless mapping and be ready for production deployments.

SUMMARY

The demand for a virtual filesystem with included policy and well-understood semantics is very strong. iRODS provides that abstraction and capability. However, it takes a lot of engineering effort to teach existing tools and workflows to speak the iRODS protocol. It is more likely that tools can read and write into a mountpoint provided by a compatibility layer between POSIX and iRODS.

NFSRODS provides this compatibility layer and v0.8.0 is the first proof of concept. Existing tools can read and write into the iRODS namespace without any changes to their own code, and iRODS organizational policy is enforced on the server.

The lossy mapping between iRODS and traditional Unix permissions will be addressed in an upcoming v1.0 release where iRODS and NFSv4 ACLs will be losslessly mapped to one another.

REFERENCES

[1] iRODS Client NFSRODS. `https://github.com/irods/irods_client_nfsrods`

[2] Xu, H., Russell, T., Coposky, J., et al: iRODS Primer 2: Integrated Rule-Oriented Data System. In: Synthesis Lectures on Information Concepts, Retrieval, and Services. 131pp. Morgan Claypool. (2017)

[3] Haynes, T., Noveck, D.: Network File System (NFS) Version 4 Protocol (2015) `https://tools.ietf.org/html/rfc7530`

[4] NFS-RODS: A Tool for Accessing iRODS via the NFS Protocol (2016) `https://irods.org/uploads/2016/06/NFSRODS-slides.pdf`

[5] Callaghan, B., Pawlowski, B., Staubach, P.: NFS Version 3 Protocol Specification (1995) `https://tools.ietf.org/html/rfc1813`

[6] NFS4J. `https://github.com/dCache/nfs4j`

[7] Carl Boettiger, C.: An introduction to Docker for reproducible research (2015) `https://doi.org/10.1145/2723872.2723882`

[8] Kohl, J., Neuman, C.: The Kerberos Network Authentication Service (V5) (1993) `https://tools.ietf.org/html/rfc1510`

[9] Jargon - iRODS Java client library. `https://github.com/DICE-UNC/jargon`

[10] Traditional Unix permissions. `https://en.wikipedia.org/wiki/File_system_permissions#Traditional_Unix_permissions`