iRODS Client: NFSRODS 1.0

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

An update from last year's preview, this v1.0 release of NFSRODS[1] now provides multi-user support for NFSv4 ACLs by handling calls from `nfs4_setfacl` and `nfs4_getfacl`. It also supports sssd for easier AD/LDAP integration and secure connections to iRODS via SSL. NFSRODS v1.0 can provide a direct NFSv4.1[2] mount point to iRODS[3] users in enterprise environments.

Keywords

iRODS, client, NFS, NFSv4, data management

INTRODUCTION

Since 2019's initial implementation[4], the iRODS Consortium has worked to complete the implementation of an NFSv4.1 server that provides a complete lossless bidirectional permission mapping of multiple owners of collections and data objects in iRODS to NFSv4 ACLs.

Along the way, as interest and usage increased in the community, additional features were requested and implemented. Easy Active Directory (AD) and LDAP support is included via support for sssd.

NFSRODS v1.0 is nearly feature complete. With this release, the roadmap for NFSRODS includes adding support for parallel file transfer and hard links.

ARCHITECTURE

The core of NFSRODS has two components. The NFS server side of NFSRODS is provided by NFS4J[5] 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[6] is used to implement an iRODS VirtualFileSystem. Jargon implements the iRODS protocol and communicates as an iRODS client.

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 communicates 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` file. Therefore, these must be kept in sync across the different machines in the

iRODS UGM 2020 June 9-12, 2020, Virtual

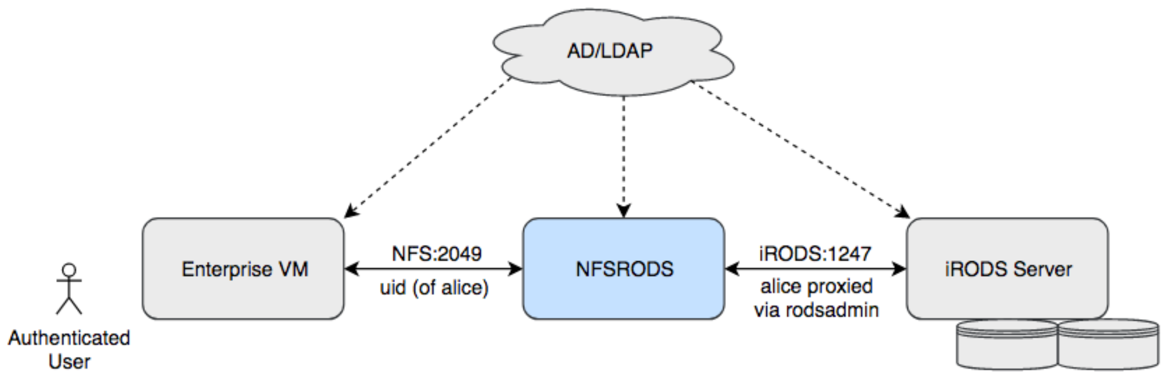

Figure 1. NFSRODS assumes an authenticated user without sudo access within the Enterprise VM.

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

In iRODS, multiple users and groups can be given different permissions on a collection or data object. Unix does not provide this capability and therefore, iRODS permissions cannot be mapped into traditional Unix permissions[7] without losing information. To get around this, NFSRODS uses NFSv4 ACLs.

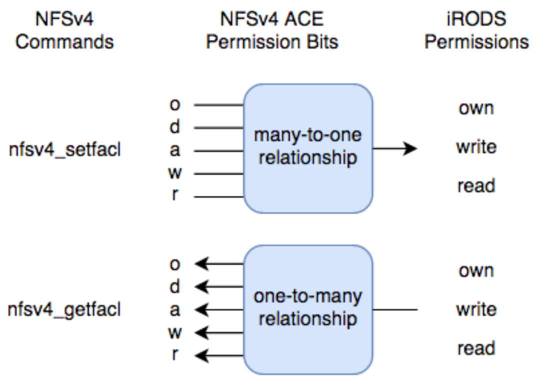

Figure 2. Bidirectional mapping of NFSv4 Commands and iRODS Permissions.

NFSv4 ACLs provide more than enough control for reflecting iRODS permissions in Unix. To manage permissions through NFSRODS, you'll need to install the package that contains `nfs4_getfacl` and `nfs4_setfacl`. On Ubuntu 16.04, that package would be `nfs4-acl-tools`. With these commands, you can view and modify all permissions in iRODS.

The order of Access Control Entries (ACEs) within an ACL does not matter in NFSRODS. When NFSRODS has to decide whether a user is allowed to execute an operation, it takes the highest level of permission for that user (including groups the user is a member of).

Using nfs4_setfacl

When using `nfs4_setfacl`, it is important to remember the following:

- Domain names within the user and group name field are ignored.
- Special ACE user/group names (e.g. OWNER, GROUP, EVERYONE, etc.) are not supported.
- Unsupported permission bits are ignored.
- The highest permission level provided is what NFSRODS will set as the permission.

Below is the permissions translation table used by NFSRODS when `nfs4_setfacl` is invoked. The list is in descending order of iRODS permissions.

NFSv4 ACE Permission Bit	NFSv4 ACE Permission Bit Name	iRODS Permission
o	ACE4_WRITE_OWNER	own
a	ACE4_APPEND_DATA	write
w	ACE4_WRITE_DATA	write
r	ACE4_READ_DATA	read

Table 1. NFSRODS v1.0 Mapping of NFSv4 ACE Permission Bits to iRODS Permissions

A simple example is as follows:

```
$ nfs4_setfacl -a A::john@:ro foo.txt
```

NFSRODS will see that the `ACE4_READ_DATA` and `ACE4_WRITE_OWNER` bits are set. It then maps these to appropriate iRODS permissions and takes the max of those. NFSRODS will then set `john`'s permission on `foo.txt` to `own`.

Using nfs4_getfacl

Using this command is much simpler. When invoked, it returns the list of iRODS permissions on an object as an ACL. The mapping used for translation is shown below.

iRODS Permission	NFSv4 ACE Permission Bits
own	rwado
write	rwa
read	r

Table 2. NFSRODS v1.0 Mapping of iRODS Permissions to NFSv4 ACE Permission Bits

nfs4_setfacl Whitelist

NFSRODS offers a whitelist for granting `nfs4_setfacl` permission to particular users.

If a user is in the whitelist or in a group in the whitelist, they can run `nfs4_setfacl` on the specified logical path or any collection or data object below it, regardless of their iRODS permissions on that collection or data object.

A rodsadmin can add a user to the whitelist by adding a specific iRODS AVU (metadata) on the user.

```
$ imeta add -u <username> irods::nfsrods::grant_nfs4_setfacl <logical_path_prefix>
```

The following example demonstrates adding `alice#tempZone` to the whitelist with a prefix of `/tempZone/project_a/lab/notes`:

```
$ imeta add -u alice irods::nfsrods::grant_nfs4_setfacl /tempZone/project_a/lab/notes
$ imeta ls -u alice
AVUs defined for user alice#tempZone:
attribute: irods::nfsrods::grant_nfs4_setfacl
value: /tempZone/project_a/lab/notes
units:
```

A user can set permissions via `nfs4_setfacl` on a collection or data object if any of the following are true:

1. The user is an iRODS administrator (i.e. rodsadmin).
2. The user has own permission on the collection or data object.
3. The user is a member of a group that has own permission on the collection or data object.
4. The user is in the whitelist with a prefix that covers the collection or data object.
5. The user is a member of a group in the whitelist with a prefix that covers the collection or data object.

USAGE

Deployment of NFSRODS v1.0 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 for the NFSRODS 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 for cached user information.
        "user_information_refresh_time_in_milliseconds": 3600000,

        // The refresh time for cached stat information.
        "file_information_refresh_time_in_milliseconds": 1000,

        // The refresh time for cached user access information.
        "user_access_refresh_time_in_milliseconds": 1000,

        // Specifies whether the force flag should be applied when overwriting
        // an existing file. If this option is false, an error will be reported
        // back to the client.
        "allow_overwrite_of_existing_files": true
    },

    // 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",

        // Defines the target resource for new data objects.
        "default_resource": "demoResc",

        // Enables/disables SSL/TLS between NFSRODS and the iRODS server.
        //
        // The following options are available:
        // - CS_NEG_REQUIRE: Only use SSL/TLS.
        // - CS_NEG_DONT_CARE: Use SSL/TLS if the iRODS server is not set to CS_NEG_REFUSE.
        // - CS_NEG_REFUSE: Do NOT use SSL/TLS.
        "ssl_negotiation_policy": "CS_NEG_REFUSE",

        // The total amount of time before an idle connection times out.
        // Defaults to 600 seconds.
        "connection_timeout_in_seconds": 600,

        // 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 for some administrative actions. iRODS will still apply policies
        // based on the requesting user's account, not the proxy admin account.
        "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 mountpoint will expose the virtual filesystem, and some cache settings (`user_information_refresh_time_in_milliseconds`, `file_information_refresh_time_in_milliseconds`, and `user_access_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. Also found here are the `ssl_negotiation_policy` and the `connection_timeout_in_seconds` setting for idle connections to refresh.

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 `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 </full/path/to/etc/passwd/formatted/file>:/etc/passwd: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 mount of the host system's `/etc/passwd`-formatted file.

It is important to note that the volume-mounted `/etc/passwd`-formatted file is expected to contain all of the users planning to use NFSRODS. The users defined in this file MUST be defined in iRODS as well. Their usernames must match the names defined in this file exactly as this is how NFSRODS matches users to the correct account in iRODS.

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

SSL

If you want to connect NFSRODS to an iRODS Zone that is using SSL, a certificate file can be mounted for use within the container:

```
             -v </full/path/to/certificate.crt>:/nfsrods_ssl.crt:ro
```

The container will load any cert it finds at `/nfsrods_ssl.crt` within the container into the OpenJDK keystore.

sssd Integration

As an alternative to an /etc/passwd-formatted file, the default NFSRODS container also supports libnss-sss. It can be used by configuring sssd on the container host and binding the sssd socket into the container.

```
$ docker run -d --name nfsrods \
             -p <public_port>:2049 \
             -v </full/path/to/nfsrods_config>:/nfsrods_config:ro \
             -v /var/lib/sss:/var/lib/sss \
             nfsrods
```

Using sssd, NFSRODS can use any sssd domain for ID mapping, including AD or LDAP. If sssd and /etc/passwd are used together, passwd will be consulted first.

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 v1.0 represents a nearly feature-complete release and has been deployed into production in multiple enterprise environments.

The only major features remaining to be added are hard link support and parallel file transfers in and out of iRODS. Parallel transfer may be possible with the upcoming release of iRODS 4.2.9 where multiple streams can operate on port 1247, reading or writing to a single data object.

NFSRODS v1.0 incorporates the NFStest[8] suite but should be paired with a performance testing model to characterize its overhead. Anecdotal feedback suggests best and usable performance when the NFSRODS server is co-hosted with the iRODS catalog provider. This makes sense as it reduces additional network hops.

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 v1.0 provides this compatibility layer and has been deployed into production in multiple enterprise environments. 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.

REFERENCES

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

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

[3] 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)

[4] Draughn, K., Russell, T., et al: NFSRODS: Presenting iRODS as NFSv4.1. 6pp. 2019 iRODS User Group Meeting. (2019) https://irods.org/uploads/2019/Draughn-iRODS-NFSRODS-paper.pdf

[5] NFS4J. https://github.com/dCache/nfs4j

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

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

[8] NFStest http://wiki.linux-nfs.org/wiki/index.php/NFStest