NFS

New in version Jewel.

Ceph Object Gateway namespaces can now be exported over file-based access protocols such as NFSv3 and NFSv4, alongside traditional HTTP access protocols (S3 and Swift).

In particular, the Ceph Object Gateway can now be configured to provide file-based access when embedded in the NFS-Ganesha NFS server.

LIBRGW

The librgw.so shared library (Unix) provides a loadable interface to Ceph Object Gateway services, and instantiates a full Ceph Object Gateway instance on initialization.

In turn, librgw.so exports rgw_file, a stateful API for file-oriented access to RGW buckets and objects. The API is general, but its design is strongly influenced by the File System Abstraction Layer (FSAL) API of NFS-Ganesha, for which it has been primarily designed.

A set of Python bindings is also provided.

NAMESPACE CONVENTIONS

The implementation conforms to Amazon Web Services (AWS) hierarchical namespace conventions which map UNIX-style path names onto S3 buckets and objects.

The top level of the attached namespace consists of S3 buckets, represented as NFS directories. Files and directories subordinate to buckets are each represented as objects, following S3 prefix and delimiter conventions, with ‘/’ being the only supported path delimiter [1].

For example, if an NFS client has mounted an RGW namespace at “/nfs”, then a file “/nfs/mybucket/www/index.html” in the NFS namespace corresponds to an RGW object “www/index.html” in a bucket/container “mybucket.”

Although it is generally invisible to clients, the NFS namespace is assembled through concatenation of the corresponding paths implied by the objects in the namespace. Leaf objects, whether files or directories, will always be materialized in an RGW object of the corresponding key name, “<name>” if a file, “<name>/” if a directory. Non-leaf directories (e.g., “www” above) might only be implied by their appearance in the names of one or more leaf objects. Directories created within NFS or directly operated on by an NFS client (e.g., via an attribute-setting operation such as chown or chmod) always have a leaf object representation used to store materialized attributes such as Unix ownership and permissions.

SUPPORTED OPERATIONS

The RGW NFS interface supports most operations on files and directories, with the following restrictions:

- Links, including symlinks, are not supported
- NFS ACLs are not supported
    - Unix user and group ownership and permissions *are* supported
- Directories may not be moved/renamed
    - files may be moved between directories
- Only full, sequential *write* i/o is supported
    - i.e., write operations are constrained to be **uploads**
    - many typical i/o operations such as editing files in place will necessarily fail as they perform non-sequential stores
    - some file utilities *apparently* writing sequentially (e.g., some versions of GNU tar) may fail due to infrequent non-sequential stores
    - When mounting via NFS, sequential application i/o can generally be constrained to be written sequentially to the NFS server via a synchronous mount option (e.g. -osync in Linux)
    - NFS clients which cannot mount synchronously (e.g., MS Windows) will not be able to upload files

SECURITY

The RGW NFS interface provides a hybrid security model with the following characteristics:

- NFS protocol security is provided by the NFS-Ganesha server, as negotiated by the NFS server and clients
  - e.g., clients can by trusted (AUTH_SYS), or required to present Kerberos user credentials (RPCSEC_GSS)
  - RPCSEC_GSS wire security can be integrity only (krb5i) or integrity and privacy (encryption, krb5p)
  - various NFS-specific security and permission rules are available
    - e.g., root-squashing
- a set of RGW/S3 security credentials (unknown to NFS) is associated with each RGW NFS mount (i.e., NFS-Ganesha EXPORT)
  - all RGW object operations performed via the NFS server will be performed by the RGW user associated with the credentials stored in the export being accessed (currently only RGW and RGW LDAP credentials are supported)
    - additional RGW authentication types such as Keystone are not currently supported

CONFIGURING AN NFS-GANESHA INSTANCE

Each NFS RGW instance is an NFS-Ganesha server instance *embeddding* a full Ceph RGW instance.

Therefore, the RGW NFS configuration includes Ceph and Ceph Object Gateway-specific configuration in a local ceph.conf, as well as NFS-Ganesha-specific configuration in the NFS-Ganesha config file, ganesha.conf.

CEPH.CONF

Required ceph.conf configuration for RGW NFS includes:

- valid [client.radosgw.{instance-name}] section
- valid values for minimal instance configuration, in particular, an installed and correct `keyring`

Other config variables are optional, front-end-specific and front-end selection variables (e.g., `rgw data` and `rgw frontends`) are optional and in some cases ignored.

A small number of config variables (e.g., `rgw_namespace_expire_secs`) are unique to RGW NFS.

GANESHA.CONF

A strictly minimal ganesha.conf for use with RGW NFS includes one EXPORT block with embedded FSAL block of type RGW:

```
EXPORT
{
     Export_ID={numeric-id};
     Path = "/";
     Pseudo = "/";
     Access_Type = RW;
     SecType = "sys";
     NFS_Protocols = 4;
     Transport_Protocols = TCP;

     # optional, permit unsquashed access by client "root" user
     #Squash = No_Root_Squash;

     FSAL {
             Name = RGW;
             User_Id = {s3-user-id};
             Access_Key_Id ="{s3-access-key}";
             Secret_Access_Key = "{s3-secret}";
     }
}
```

`Export_ID` must have an integer value, e.g., “77”

`Path` (for RGW) should be “/”

`Pseudo` defines an NFSv4 pseudo root name (NFSv4 only)

`SecType = sys;` allows clients to attach without Kerberos authentication

`Squash = No_Root_Squash;` enables the client root user to override permissions (Unix convention). When root-squashing is enabled, operations attempted by the root user are performed as if by the local “nobody” (and “nogroup”) user on the NFS-Ganesha server

The RGW FSAL additionally supports RGW-specific configuration variables in the RGW config section:

```
RGW {
    cluster = "{cluster name, default 'ceph'}";
    name = "client.rgw.{instance-name}";
    ceph_conf = "/opt/ceph-rgw/etc/ceph/ceph.conf";
    init_args = "-d --debug-rgw=16";
}
```

`cluster` sets a Ceph cluster name (must match the cluster being exported)

`name` sets an RGW instance name (must match the cluster being exported)

`ceph_conf` gives a path to a non-default ceph.conf file to use

OTHER USEFUL NFS-GANESHA CONFIGURATION:

Any EXPORT block which should support NFSv3 should include version 3 in the NFS_Protocols setting. Additionally, NFSv3 is the last major version to support the UDP transport. To enable UDP, include it in the Transport_Protocols setting. For example:

```
EXPORT {
 ...
   NFS_Protocols = 3,4;
   Transport_Protocols = UDP,TCP;
 ...
}
```

One important family of options pertains to interaction with the Linux idmapping service, which is used to normalize user and group names across systems. Details of idmapper integration are not provided here.

With Linux NFS clients, NFS-Ganesha can be configured to accept client-supplied numeric user and group identifiers with NFSv4, which by default stringifies these–this may be useful in small setups and for experimentation:

```
NFSV4 {
    Allow_Numeric_Owners = true;
    Only_Numeric_Owners = true;
}
```

TROUBLESHOOTING

NFS-Ganesha configuration problems are usually debugged by running the server with debugging options, controlled by the LOG config section.

NFS-Ganesha log messages are grouped into various components, logging can be enabled separately for each component. Valid values for component logging include:

```
*FATAL* critical errors only
*WARN* unusual condition
*DEBUG* mildly verbose trace output
*FULL_DEBUG* verbose trace output
```

Example:

```
 LOG {

       Components {
               MEMLEAKS = FATAL;
               FSAL = FATAL;
               NFSPROTO = FATAL;
               NFS_V4 = FATAL;
               EXPORT = FATAL;
               FILEHANDLE = FATAL;
               DISPATCH = FATAL;
               CACHE_INODE = FATAL;
```

```
                CACHE_INODE_LRU = FATAL;
                HASHTABLE = FATAL;
                HASHTABLE_CACHE = FATAL;
                DUPREQ = FATAL;
                INIT = DEBUG;
                MAIN = DEBUG;
                IDMAPPER = FATAL;
                NFS_READDIR = FATAL;
                NFS_V4_LOCK = FATAL;
                CONFIG = FATAL;
                CLIENTID = FATAL;
                SESSIONS = FATAL;
                PNFS = FATAL;
                RW_LOCK = FATAL;
                NLM = FATAL;
                RPC = FATAL;
                NFS_CB = FATAL;
                THREAD = FATAL;
                NFS_V4_ACL = FATAL;
                STATE = FATAL;
                FSAL_UP = FATAL;
                DBUS = FATAL;
        }
        # optional: redirect log output
#       Facility {
#               name = FILE;
#               destination = "/tmp/ganesha-rgw.log";
#               enable = active;
        }
}
```

RUNNING MULTIPLE NFS GATEWAYS

Each NFS-Ganesha instance acts as a full gateway endpoint, with the limitation that currently an NFS-Ganesha instance cannot be configured to export HTTP services. As with ordinary gateway instances, any number of NFS-Ganesha instances can be started, exporting the same or different resources from the cluster. This enables the clustering of NFS-Ganesha instances. However, this does not imply high availability.

When regular gateway instances and NFS-Ganesha instances overlap the same data resources, they will be accessible from both the standard S3 API and through the NFS-Ganesha instance as exported. You can co-locate the NFS-Ganesha instance with a Ceph Object Gateway instance on the same host.

RGW VS RGW NFS

Exporting an NFS namespace and other RGW namespaces (e.g., S3 or Swift via the Civetweb HTTP front-end) from the same program instance is currently not supported.

When adding objects and buckets outside of NFS, those objects will appear in the NFS namespace in the time set by `rgw_nfs_namespace_expire_secs`, which defaults to 300 seconds (5 minutes). Override the default value for `rgw_nfs_namespace_expire_secs` in the Ceph configuration file to change the refresh rate.

If exporting Swift containers that do not conform to valid S3 bucket naming requirements, set `rgw_relaxed_s3_bucket_names` to true in the [client.radosgw] section of the Ceph configuration file. For example, if a Swift container name contains underscores, it is not a valid S3 bucket name and will be rejected unless `rgw_relaxed_s3_bucket_names` is set to true.

CONFIGURING NFSV4 CLIENTS

To access the namespace, mount the configured NFS-Ganesha export(s) into desired locations in the local POSIX namespace. As noted, this implementation has a few unique restrictions:

- NFS 4.1 and higher protocol flavors are preferred
    - NFSv4 OPEN and CLOSE operations are used to track upload transactions
- To upload data successfully, clients must preserve write ordering
    - on Linux and many Unix NFS clients, use the -osync mount option

Conventions for mounting NFS resources are platform-specific. The following conventions work on Linux and some Unix platforms:

From the command line:

```
mount -t nfs -o nfsvers=4.1,noauto,soft,sync,proto=tcp <ganesha-host-name>:/ <mount-point>
```

In /etc/fstab:

```
<ganesha-host-name>:/ <mount-point> nfs noauto,soft,nfsvers=4.1,sync,proto=tcp 0 0
```

Specify the NFS-Ganesha host name and the path to the mount point on the client.

CONFIGURING NFSV3 CLIENTS

Linux clients can be configured to mount with NFSv3 by supplying `nfsvers=3` and `noacl` as mount options. To use UDP as the transport, add `proto=udp` to the mount options. However, TCP is the preferred transport:

```
<ganesha-host-name>:/ <mount-point> nfs noauto,noacl,soft,nfsvers=3,sync,proto=tcp 0 0
```

Configure the NFS Ganesha EXPORT block Protocols setting with version 3 and the Transports setting with UDP if the mount will use version 3 with UDP.

NFSV3 SEMANTICS

Since NFSv3 does not communicate client OPEN and CLOSE operations to file servers, RGW NFS cannot use these operations to mark the beginning and ending of file upload transactions. Instead, RGW NFS starts a new upload when the first write is sent to a file at offset 0, and finalizes the upload when no new writes to the file have been seen for a period of time, by default, 10 seconds. To change this timeout, set an alternate value for `rgw_nfs_write_completion_interval_s` in the RGW section(s) of the Ceph configuration file.

REFERENCES

[1] http://docs.aws.amazon.com/AmazonS3/latest/dev/ListingKeysHierarchy.html