Rook-Deployed Scalable NFS Clusters Exporting CephFS

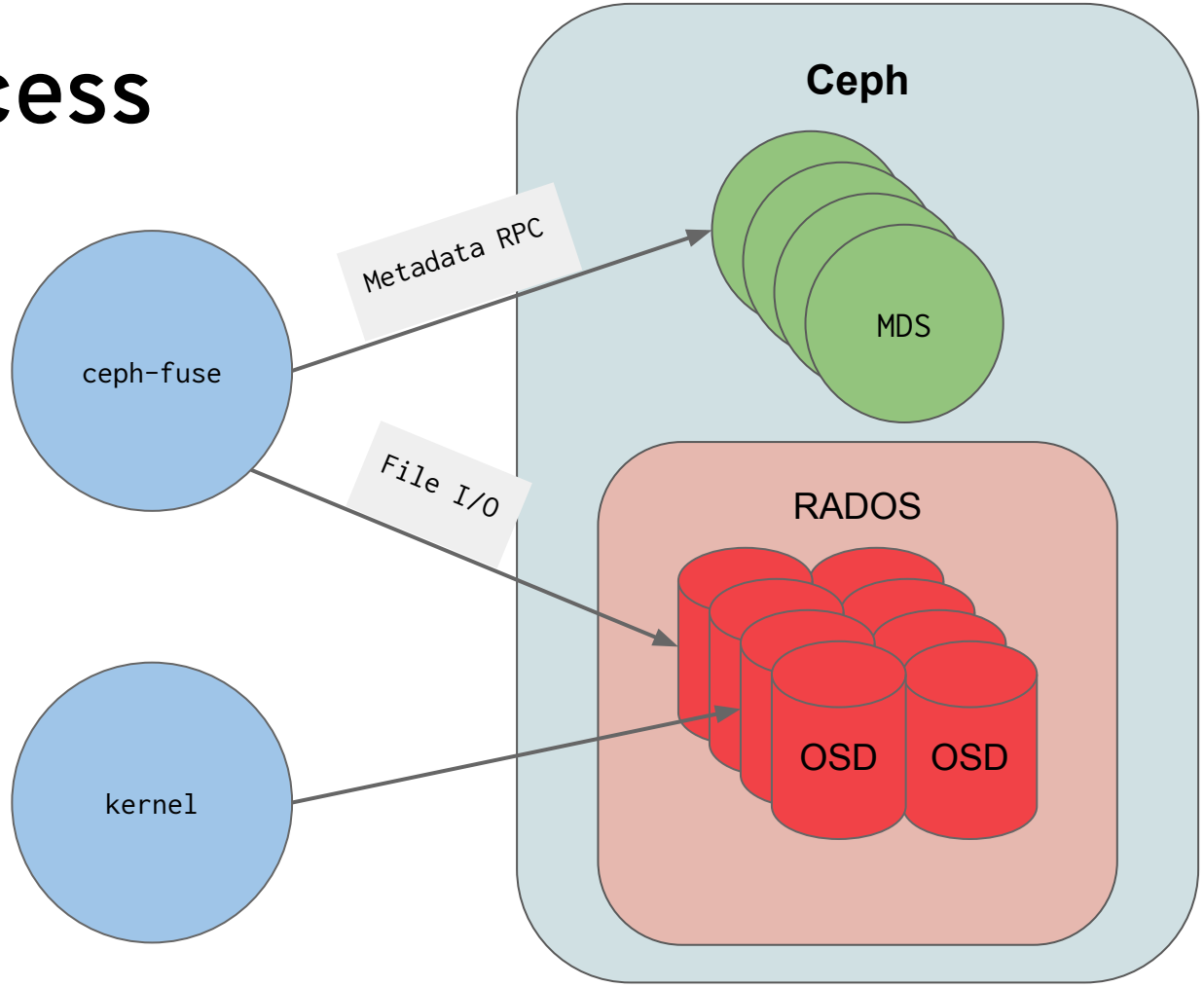
Jeff Layton
Principal Software Engineer

Patrick Donnelly
Senior Software Engineer

Red Hat

CephFS and Client Access

- CephFS is a POSIX distributed file system.
- Clients and MDS cooperatively maintain a distributed cache of metadata including inodes and directories
- MDS hands out capabilities (aka caps) to clients, to allow them delegated access to parts of inode metadata
- Clients directly perform file I/O on RADOS



Why Would You Need a Ceph/NFS Gateway?

Clients that can't speak Ceph properly

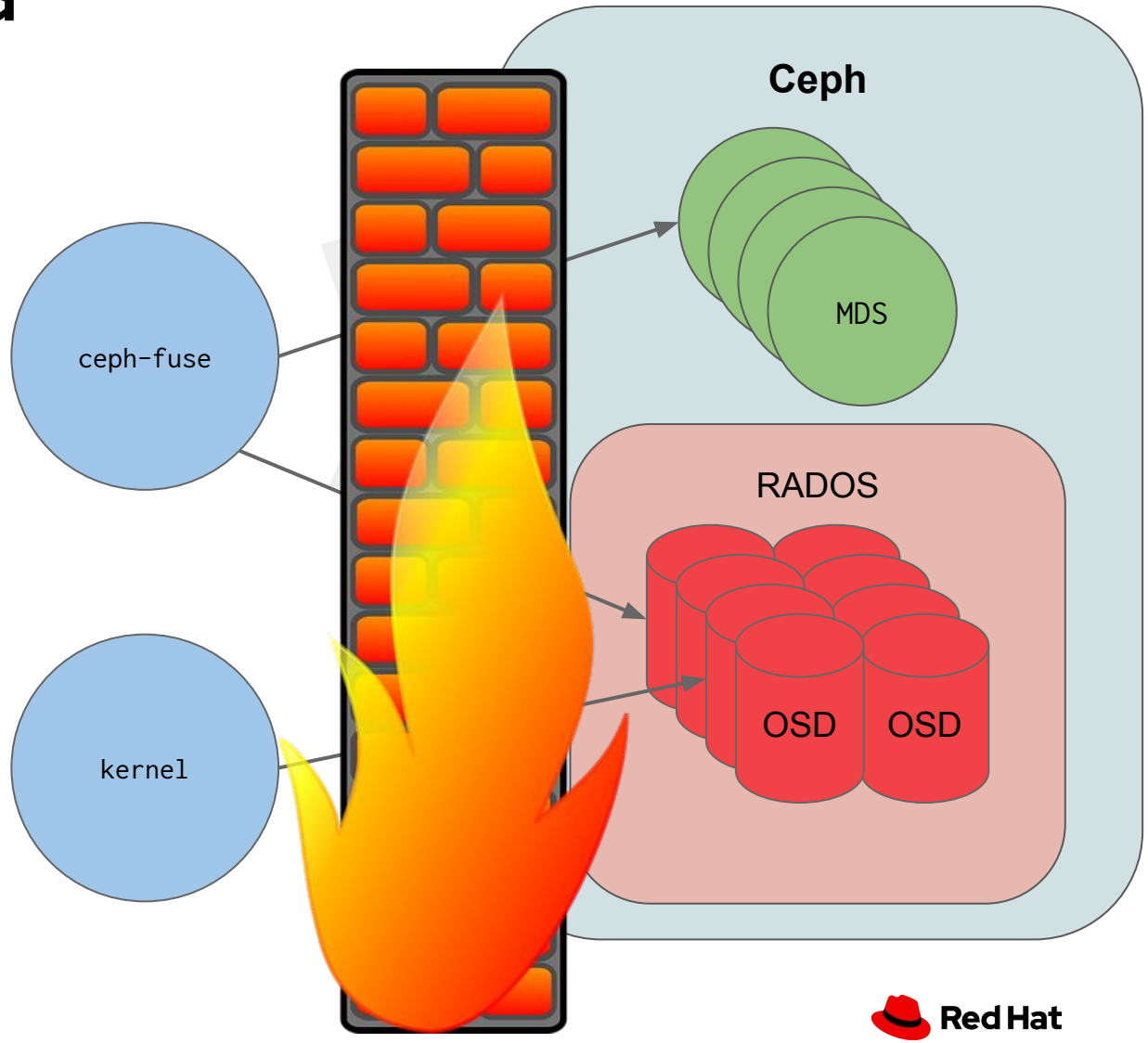
- old, questionable, or unknown ceph drivers (old kernel)
- 3rd party OSs

Security

- Partition Ceph cluster from less trusted clients
- GSSAPI (kerberos)

Openstack Manila

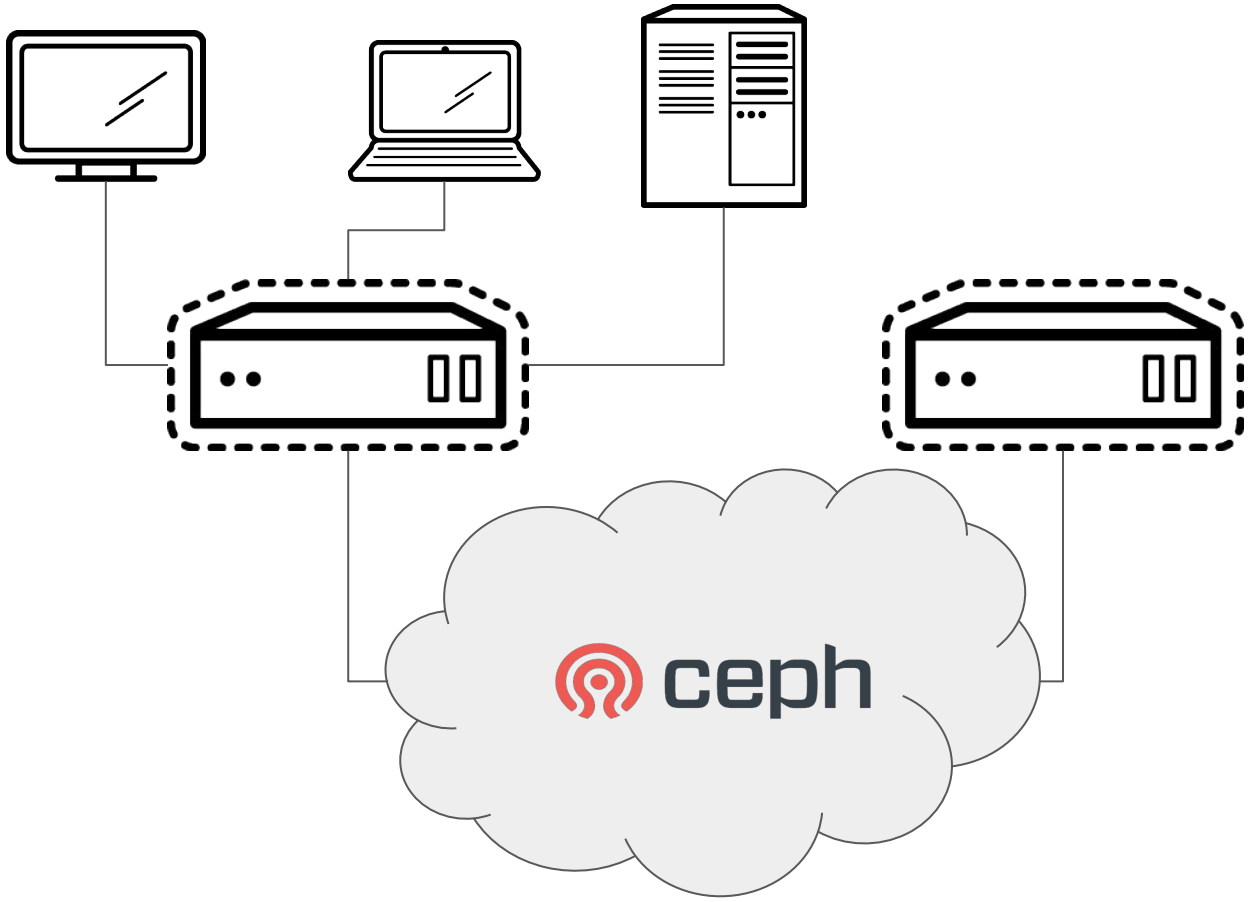
- Filesystem shared between multiple nodes
- ...but also tenant-aware
- ...and self-managed by tenant admins



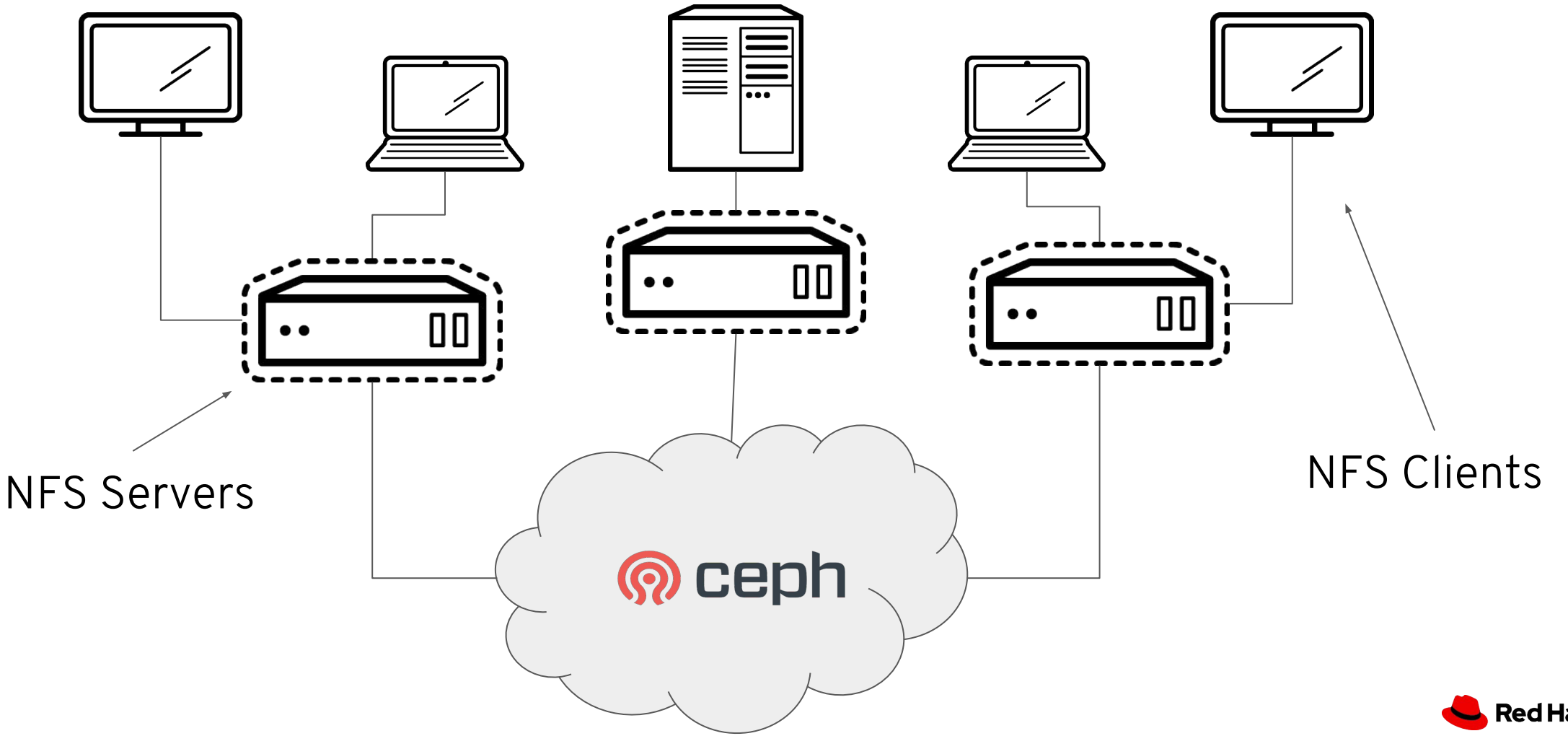
Red Hat

Active/Passive Deployments

- One active server at a time that “floats” between physical hosts
- Traditional “failover” NFS server, running under pacemaker/corosync
- Scales poorly + requires idle resources
- Available since Ceph Luminous



Goal: Active/Active Deployment



Goals and Requirements

Scale Out

Active/Active cluster of mostly independent servers. Keep coordination between them to a bare minimum.

Containerizable

Leverage container orchestration technologies to simplify deployment and handle networking. No failover of resources. Just rebuild containers from scratch when they fail.

NFSv4.1+

Avoid legacy NFS protocol versions, allowing us to rely on new protocol features for better performance, and possibility for pNFS later.

Ceph/RADOS for Communication

Avoid need for any 3rd party clustering or communication between cluster nodes. Use Ceph and RADOS to coordinate

Ganesha NFS Server

- Open-source NFS server that runs in userspace (LGPLv3)
- Plugin interface for exports and client recovery databases
  - well suited for exporting userland filesystems
- FSAL_CEPH uses **libcephfs** to interact with Ceph cluster
- Can use **librados** to store client recovery records and configuration files in RADOS objects
- Amenable to containerization
  - Store configuration and recovery info in RADOS
  - No need for writeable local fs storage
  - Can run in unprivileged container
  - Rebuild server from r/o image if it fails

NFS Protocol

- Based on ONC-RPC (aka SunRPC)
- Early versions (NFSv2 and v3) were stateless
  - with sidecar protocols to handle file locking (NLM and NSM)
- NFSv4 was designed to be stateful, and state is leased to the client
  - client must contact server at least once every lease period to renew (45-60s is typical)
- NFSv4.1 revamped protocol using a sessions layer to provide exactly-once semantics
  - added RECLAIM_COMPLETE operation (allows lifting grace period early)
  - more clustering and migration support
- NFSv4.2 mostly new features on top of v4.1

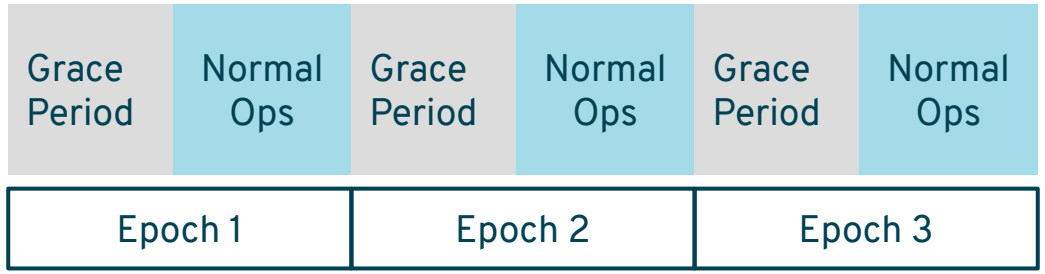
NFS Client Recovery

- After restart, NFS servers come up with no ephemeral state
  - Opens, locks, delegations and layouts are lost
  - Allow clients to reclaim state they held earlier for ~2 lease periods
  - Detailed state tracking on stable storage is quite expensive
  - Ganesha had support for storing this in RADOS for single-server configurations

- During the grace period:
  - No new state can be established. Clients may only reclaim old state.
  - Allow reclaim only from clients present at time of crash
    - Necessary to handle certain cases involving network partitions

- Must keep stable-storage records of which clients are allowed to reclaim after a reboot
  - Prior to a client doing its first OPEN, set a stable-storage record for client if there isn't one
  - Remove after last file is closed, or when client state expires
  - Atomically replace old client db with new just prior to ending grace period

NFS Server Reboot Epochs



- Consider each reboot the start of a new epoch
  - As clients perform first open (reclaim or regular), set a record for them in a database associated with current epoch
  - During grace period, allow clients that are present in the previous epoch's database to reclaim state
  - After transition from Grace to Normal operations, can delete any previous epoch's database
- The same applies in a cluster of servers!

New in Ganesha: Coordinating Grace Periods

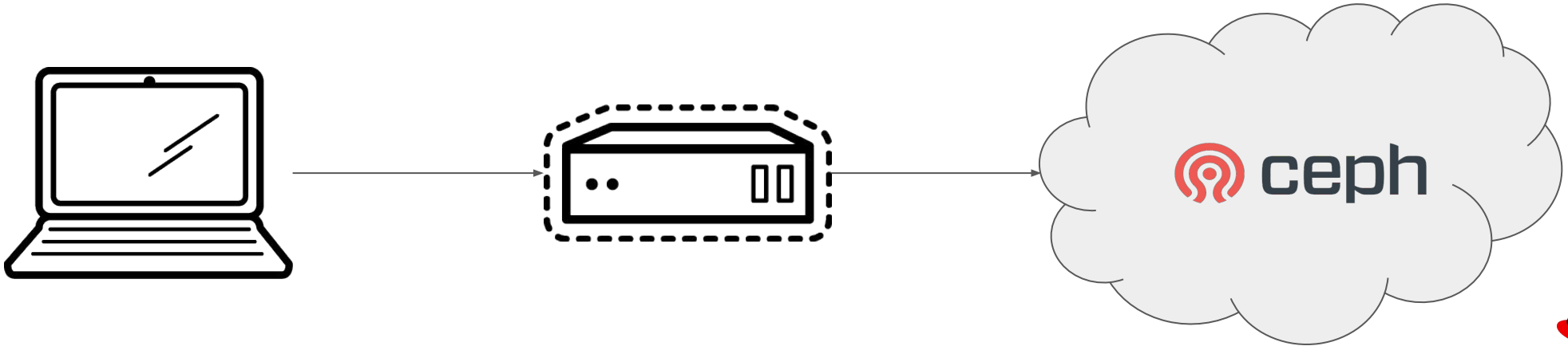
- Done via central database stored in a RADOS object
- Two 64-bit integers:
  - **Current Epoch (C):** Where should new records be stored?
  - **Recovery Epoch (R):** From what epoch should clients be allowed to reclaim?
  - **R == 0** means grace period is not in force (normal operation)
- Flags for each node indicating current need for a grace period and enforcement status
  - **NEED (N):** does this server currently require a grace period?
    - First node to set its NEED flag declares a new grace period
    - Last node to clear its NEED flag can lift the grace period
  - **ENFORCING (E):** is this node enforcing the grace period?
    - If all servers are enforcing then we know no conflicting state can be handed out (grace period is fully in force)

Simple logic to allow individual hosts to make decisions about grace enforcement based on state of all servers in cluster. No running centralized entity.

Challenge: Layering NFS over a Cluster FS

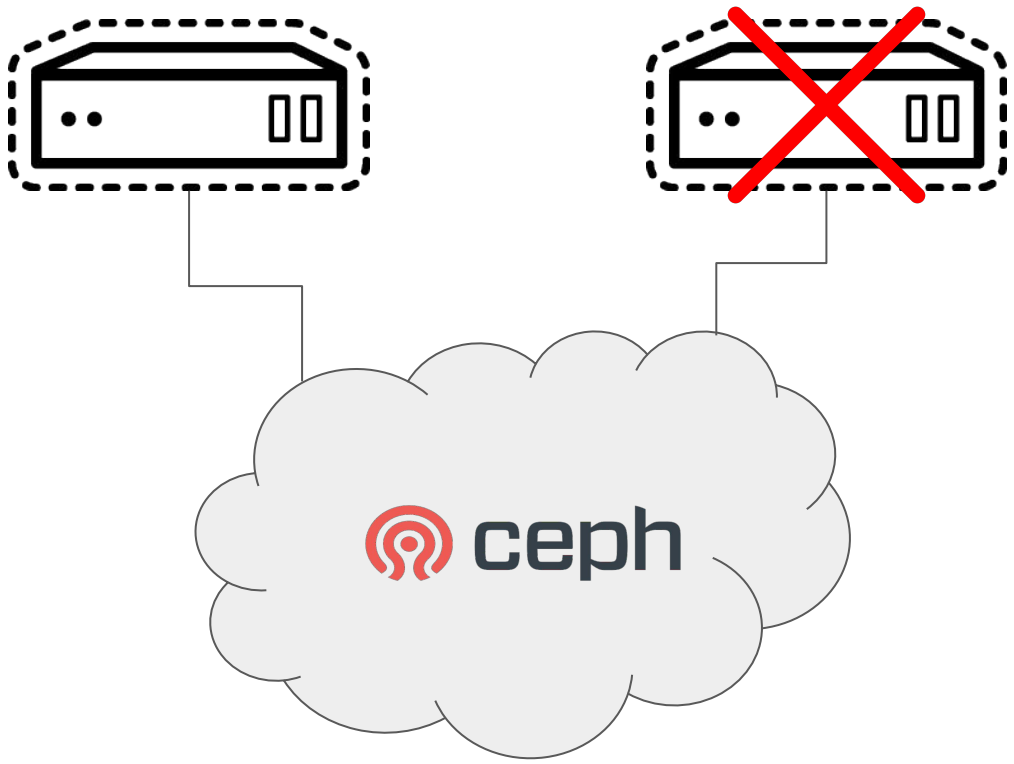
- Both protocols are stateful, and use lease-based mechanism. Ganesha acts as a proxy.

- Example: ganesha server may crash and a new one started:
  - Client issues request to do a open NFS reclaim
  - Ganesha performs an open request via libcephfs
  - Blocks, waiting for caps held by previous ganesha instance
  - Default MDS session timeout is 60s
  - Session could also die before other servers are enforcing grace period



New In Nautilus: Client Reclaim of Sessions

- When a CephFS client dies abruptly, the MDS keeps its session around for a little while (several minutes). Stateful info (locks, opens, caps, etc.) are tied to those sessions.
- Added new interfaces to libcephfs to allow ganesha to request that its old session be killed off immediately.
- May eventually add ability for session to take over old stateful objects, obviating need for grace period on surviving server heads. https://tracker.ceph.com/issues/36397



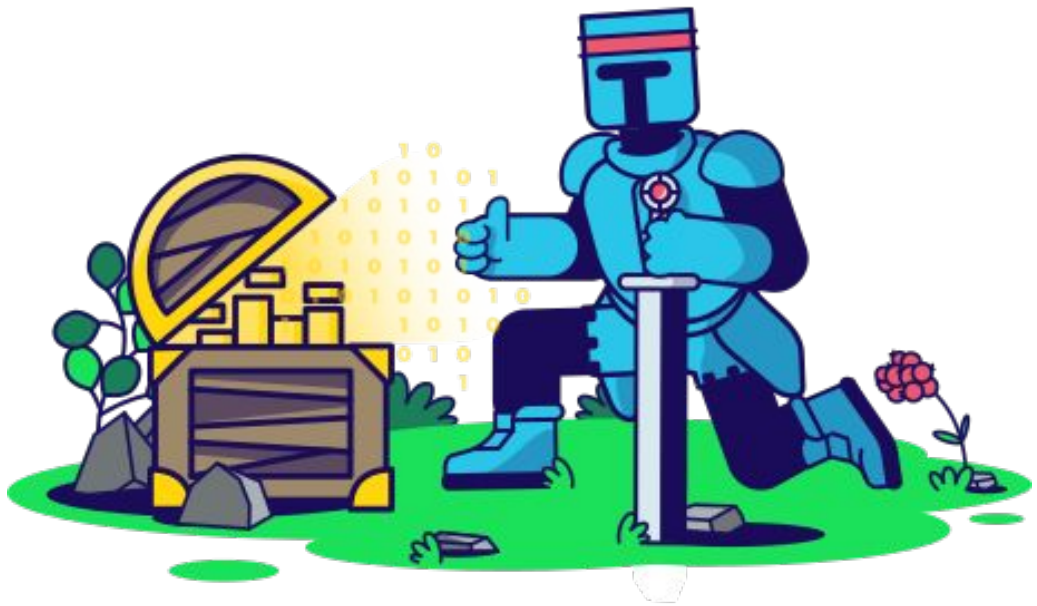
Challenge: Deploy and Manage NFS Clusters

- Dynamic NFS cluster creation/deletion
  - Scale in two directions:
    - active-active configuration of NFS clusters for a hot subvolumes
    - per-subvolume independent NFS clusters

- Handling failures + IP migration
  - Spawn replacement NFS-Ganesha daemon and migrate the IP address.
  - Note: NFS clients **cannot** modify the NFS server's IP address post-mount

Tying it all together with Rook and Kubernetes

- rook.io is cloud-native storage orchestrator that can deploy ceph clusters dynamically.
  - CephNFS resource type in k8s
- Rook operator handles cookie-cutter configuration of daemons
- Can scale NFS cluster size up or down (MDS too in the future!)
- Integration with orchestrator ceph-mgr module
  - Deploy new clusters
  - Scale node count up or down
  - Dashboard can add+remove exports



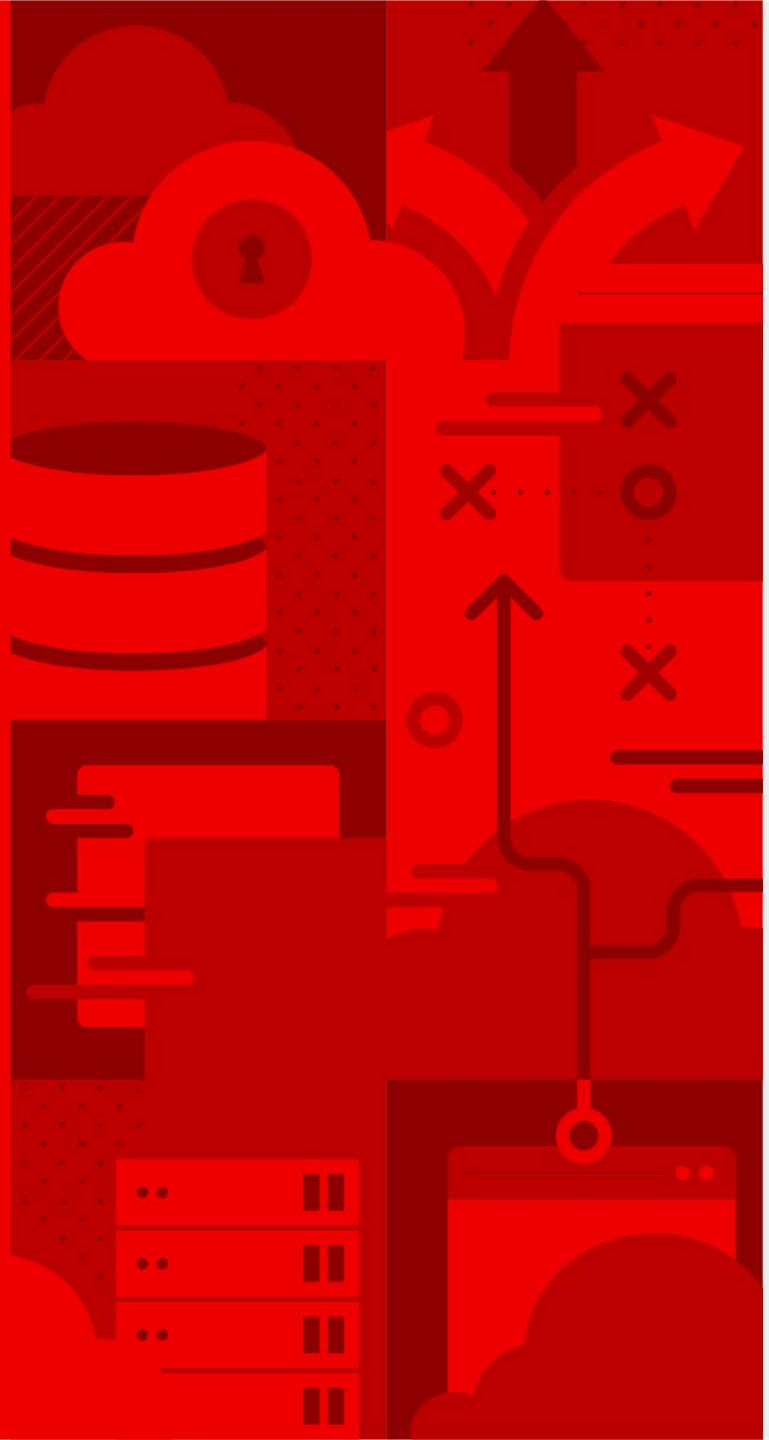
Vision for the Future

- Trivial creation/config of a managed volume with a persistent NFS Gateway.

```
$ ceph fs subvolume create vol_a subvol_1
$ ceph fs subvolume set vol_a subvol_1 <nfs-config…>
$ ceph fs subvolume set vol_a subvol_1 sharenfs true
```

- NFS-Ganesha:
  - Add support for NFSv4 migration to allow redistribution of clients among servers
  - Optimizing grace periods for subvolumes.

- SMB integration (samba)



Thank you

Jeff Layton jlayton@redhat.com

Patrick Donnelly pdonnell@redhat.com

Blog Demo: https://ceph.com/community/deploying-a-cephnfs-server-cluster-with-rook/

New in Nautilus: CephFS Improvements: https://ceph.com/community/nautilus-cephfs/

Rook: https://rook.io/

Ceph: https://ceph.com/

NFS Ganesha: https://www.nfs-ganesha.org/

Red Hat