

Monitoring, Backup/Disaster Recovery, Performance Testing, Capacity Planning

- [Monitoring](#)
- [Backup/Disaster Recovery](#)
 - [Process](#)
- [Performance Testing](#)
 - [Open Questions](#)
- [Capacity Planning](#)

Monitoring

See [NDS Labs Monitoring](#).

Backup/Disaster Recovery

1. GFS, Etcd "best effort" for beta
 2. Cluster config (using kubectl)
 3. Deploy tools provisioning
- Where? AWS, BW, TACC
 - How? Some script/Job/rsync
 - When? Daily rolling
 - Q
 - Hot backup of DBs – backupz + side car
 - GFS backup options, depends on # of users
 - Snapshots + diffs
 - Checkpointing
 - Replication to another GFS/geolocation

Process



Backup and DR Procedures

Backup DR Plan:

This backup/DR plan is intended for emergency use to recover the system to a recent and operational state quickly, with user/system state restored to the time of the backup in toto. It is not intended as a "user-backup" strategy where users can request accidentally deleted data due to user error or as a partial-state restoration mechanism.

Sources of data at risk:

Provisioning data (ansible/SAVED_VOLUME/provisioning image/version):

includes original state of cluster, images, sizes, addresses, etc.

Size is small and constant (kb-mb) : provisioning image reference, inventory, dump of all ansible variables post-provision

Kubernetes cluster info (cluster-dump, etcd-dump)

includes current cluster configuration from kubernetes admin view and the

raw data-dump of the full etcd configuration database

Size is small and grows linearly in small increments with #users

Global volume data (/var/glfs/global)

All system and user data stored in the cluster global filesystem

Size is variable and assumed to be in the GB to TB range

Retention:

At least one 'latest' state dump of the system is required, giving retention of one dump-period.

A rolling stack of N 'latest' dumps is preferable for safety in case a dump failure leaves the latest corrupted,

this increased retention to N times the dump-period, and requires N times storage capacity.

A rolling level-based strategy rolls the last N dumps, and every cycle promotes the last dump to a higher-level M

set where the period of M is N. Typical strategies include N=7, M=5 which saves a week of daily dumps and 5

rolling weekly copies, giving retention of 5 weeks with M+N storage required. The periods of M and N can be adjusted

to accomodate tradeoff between ability to recover 'fresh' data, storage requirements, and operational requirements.

Incremental vs full-copy:

To achieve better resource utilization a 2 layer N/M strategy start with a full-copy M, followed by N incremental dumps

against the most recent M. This minimizes the data transfer and storage requirements according to

how much data is recently changed. For slow changing systems the lower limit is M copies, for high-churn filesystems the upper limit approaches N+M for total storage required. This strategy is useful in low-change scenarios due to the savings, but adds complexity in the restore process which requires restoring the latest M, and then all the incremental N's since the last M.

DR storage requirements:

DR should be off-site, geographically dispersed as widely as possible, and sized to support the storage needs of the retention policy. Best practices for auditable systems require 3 geo-dispersed copies for regulatory compliance with FIPS/NIST, etc.

Other comments:

GlobalFS data dominates - the cluster and configuration information is of little use without the globalfs and it is quite simple to include the cluster and other data within the globalfs with the other data included.

Starting simple:

It is unclear what the filesystem usage and activity patterns will be, and the activity and patterns are subject to frequent change over time, requiring adaptation of the strategy. This suggests that a simple strategy such as a rolling stack N-way full-dump, where $N \geq 4$ (covers an extra day on 3-day weekend).

Monitoring:

Assuming that there is monitoring of both the cluster and backup system does not necessarily guarantee the integrity of the backups. A corrupted filesystem can still be dumped with the cluster and the backup system presenting as normal in the monitoring system.

Special data:

Some applications can create files that are problematic for file-level backup systems. These include files with "holes" of unallocated blocks (databases), device inodes (docker, vm's etc.), vfs binds, and hard-links. These can be translated in unrestoreable ways by filesystem-level programs and various tools handle these with varying degrees of "restorability".

Gluster backup options:

Gluster currently has no native "dump" program that can export restorable contents of the filesystem such as xfs-dump or dumpe2fs. The filesystem can be backed up with traditional file archiving and distribution tools:

rdist - originally developed as a mirroring tool for content distribution mirroring. It has options for incremental transfers using block hash differencing, optional removed file tracking, server mode, ssh tunnel transfer, retries, and performs decently on most special files like device inodes and links. It is widely used but not frequently as a backup solution.

tar/cpio/etc. - simple archiving tools that have been around a long time.

Gluster Geo-replication - Geo-replication allows multiple gluster clusters to operate in a replication-tree within a master-slave configuration. After peering setup to the slave cluster at the remote-site, replication is managed via the command-line to synchronize geo-replicas and to restore data from a slave to a master. Geo replication can operate securely with TLS, and synchronizes at the gluster brick-level for efficiency.

BareOS with gluster - BareOS is a descendant of the backula backup system originally developed as a client-server backup

solution for various operating systems. The latest gluster (v3.8) and BareOS (v16.2) have the capability to scrape

and backup a gluster filesystem within the BareOS infrastructure which can manage backup schedules, data storage, restores, etc. BareOS is highly customizable for many purposes.

Design Recommendation:

Use geo-replication as it is native to gluster and likely to be most efficient. File-level tools are easy but may have unpredictable results. BareOS/backula might be preferred in scenario where such infrastructure was in place, but it's not.

In the long-run, if we come to be managing backups of data across many systems, this would make sense.

Plan:

1. Bring up peer glusters on SDSC
2. Configure geo-replication for globalfs with slaves on SDSC
3. Test replication with some test data
4. Repeat for TACC
5. Setup alternating schedule to replicate nightly to SDSC TACC alternating
 1. Dump cluster config, etcd backup to /var/glfs/global/backup
 2. Start geo-replicate
 3. Wait for replication checkpoint status
 4. Turn replication off
6. Document replication performance
 1. Create large-sized file structure
 2. Replicate and time to slaves - baseline replication throughput

Performance Testing

- GFS
- "iassist" redux
 - est. per-user quotas?
 - what do we need on day 1

Open Questions

- How many beta users?
 - what is the workload?
- By what performance metrics do we judge pass/fail?
- How do we learn our limits?
 - Capacity planning / monitoring
- What happens when we need to:
 - add GFS bricks?
 - add kubernetes nodes?
- What constitutes a failure?
 - Dead node

Capacity Planning