

SUSE® Storage hands-on session

Ceph with SUSE®

Adam Spiers

Senior Software Engineer
aspiers@suse.com

Thorsten Behrens

Senior Software Engineer
tbehrens@suse.com



Agenda

- Brief intro to SUSE Storage / Ceph
- Deployment: theory and practice
- Resiliency tests
- Calamari web UI
- Playing with RBD
- Pools: theory and practice
- CRUSH map
- Tiering and erasure coding

Why would you use the product?

More data to store

- business needs
- more data driven processes
- more applications
- e-commerce



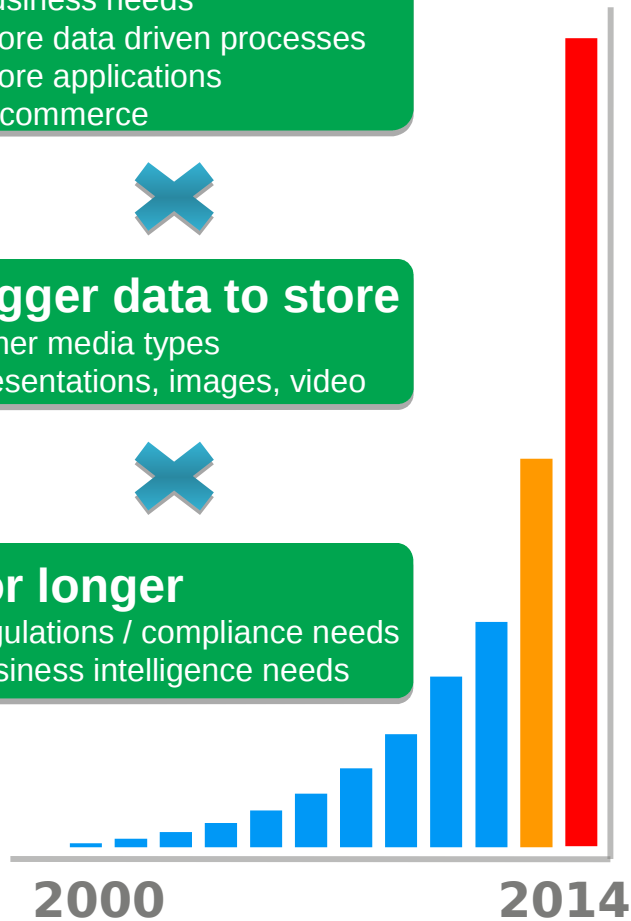
Bigger data to store

- richer media types
- presentations, images, video



For longer

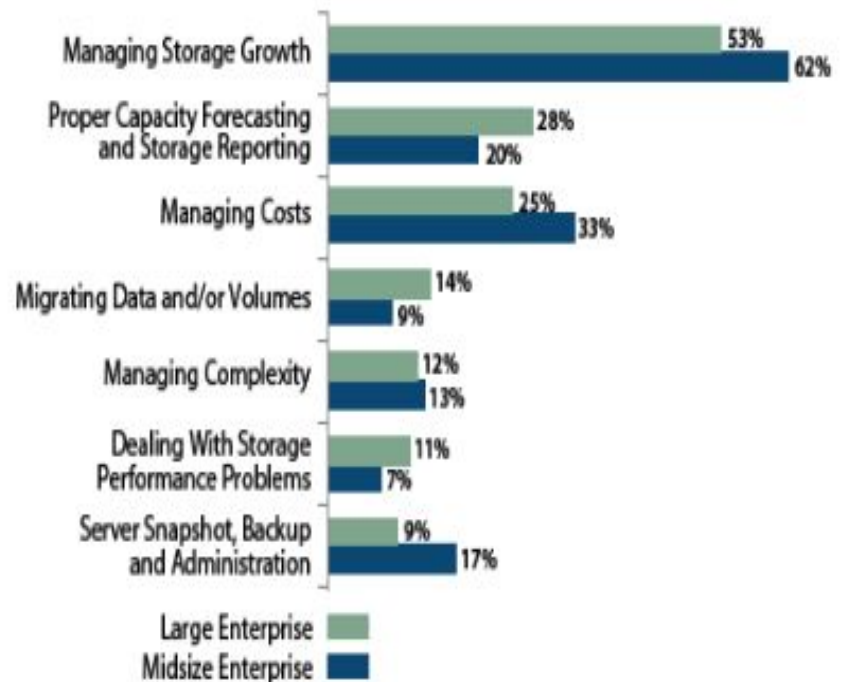
- regulations / compliance needs
- business intelligence needs



Top Seven Pain Points:

What are your top storage-related pain points?

Large Enterprise Sample: n=180; Midsize Enterprise Sample: n=69.



The InfoPro Storage Study 1H12 – 451 Research



Technical Ceph overview

Unified Data Handling for 3 Purposes

Object Storage
(Like Amazon S3)

- RESTful Interface
- S3 and SWIFT APIs

Block Device

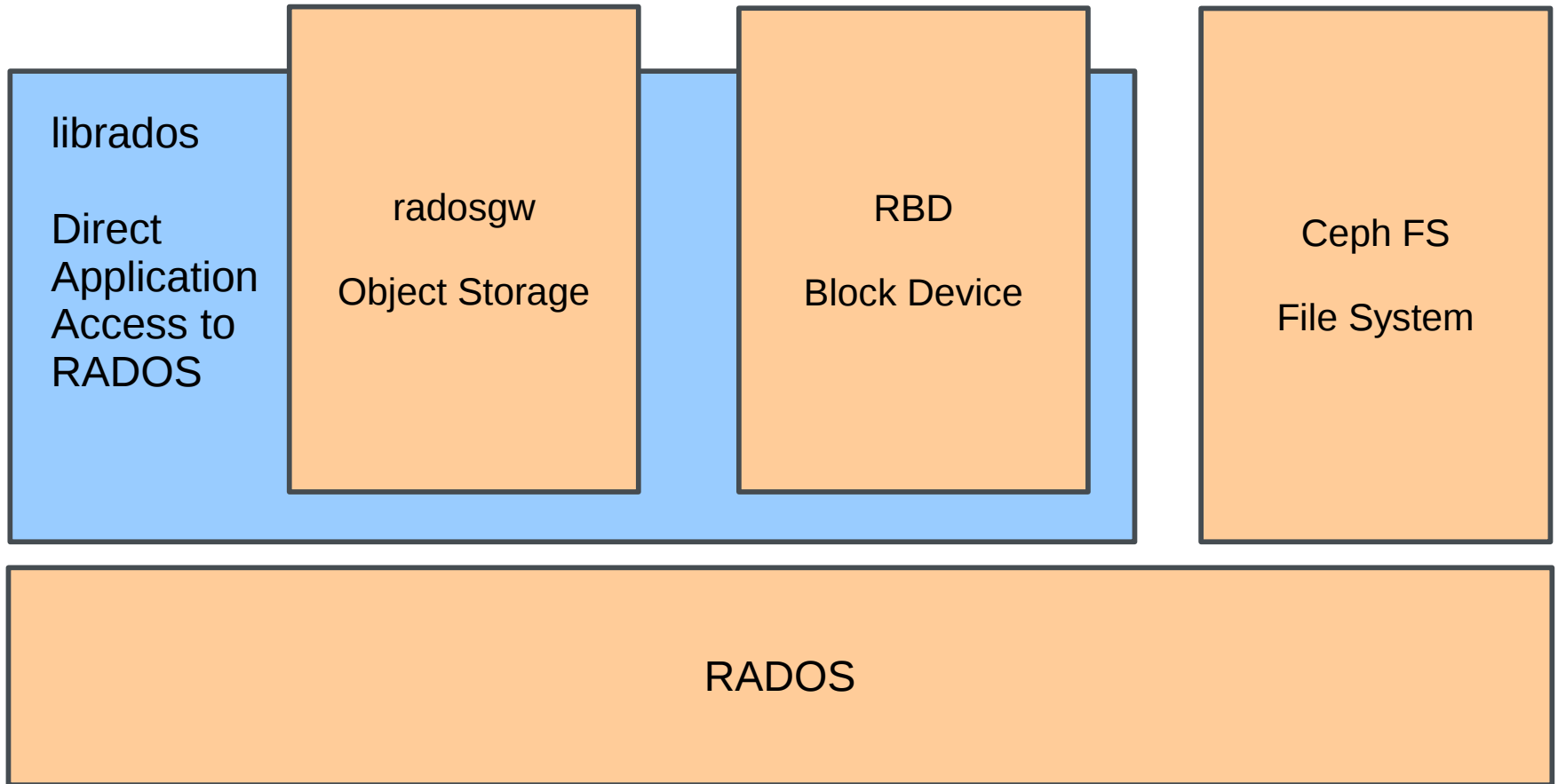
- Block devices
- Up to 16 EiB
- Thin Provisioning
- Snapshots

File System

- POSIX Compliant
- Separate Data and Metadata
- For use e.g. with Hadoop

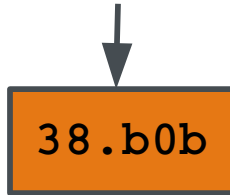
Autonomous, Redundant Storage Cluster

Component Names

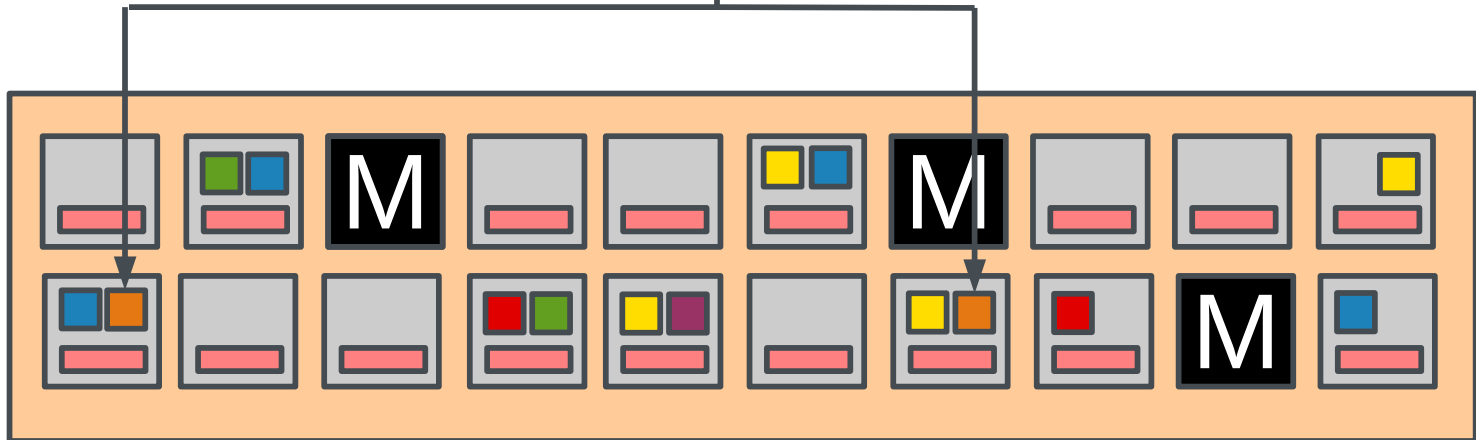


CRUSH in Action: reading

swimmingpool/rubberduck

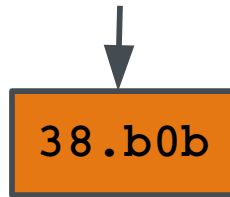


Reads could be serviced by any of the replicas (parallel reads improve throughput)

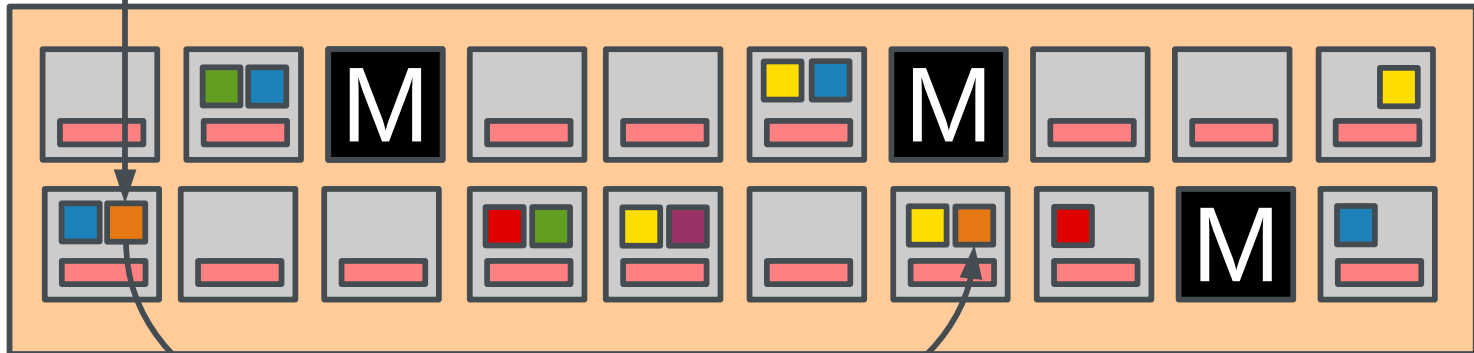


CRUSH in Action: writing

swimmingpool/rubberduck



Writes go to one OSD, which then propagates the changes to other replicas



Brief intro to SUSE Storage / Ceph

SUSE Storage

- SUSE Storage is based upon Ceph
- SUSE Storage 1.0 is soon to be released
 - Based upon Ceph Firefly release
 - This workshop will use this release

SUSE Storage architectural benefits

- Exabyte scalability
 - No bottlenecks or single points of failure
- Industry-leading functionality
 - Remote replication, erasure coding
 - Cache tiering
 - Unified block, file and object interface
 - Thin provisioning, copy on write
- 100% software based; can use commodity hardware
- Automated management
 - Self-managing, self-healing



Expected use cases

- Scalable cloud storage
 - Provide block storage for the cloud
 - Allowing host migration
- Cheap archival storage
 - Using erasure encoding (like RAID5/6)
- Scalable object store
 - This is what Ceph is built upon

More exciting things about Ceph

- Tunable for multiple use cases:
 - for performance
 - for price
 - for recovery
- Configurable redundancy:
 - at the disk level
 - at the host level
 - at the rack level
 - at the room level
 - ...

A little theory

- Only two main components:
 - "mon" for cluster state
 - OSD (Object Storage Daemon) for storing data
- Hash-based data distribution (CRUSH)
 - (Usually) No need to ask where data is
 - Simplifies data balancing
- Ceph clients communicate with OSD directly

Deploying SUSE Storage

Deploying Ceph with `ceph-deploy`

- `ceph-deploy` is a simple command line tool
- Makes small scale setups easy
- In this workshop, run as `ceph@ceph_deploy`

Workshop setup

- Each environment contains 5 VM instances running on AWS
 - one admin node to run `ceph-deploy` and Calamari
 - three Ceph nodes doubling as mons / OSDs
 - each with 3 disks for 3 OSDs to serve data
 - one client node

About Ceph layout

- Ceph needs 1 or more `mon` nodes
 - In production 3 nodes are the minimum
- Ceph needs 3 or more `osd` nodes
 - Can be fewer in testing
 - Each `osd` should manage a minimum of 15 Gb
 - Smaller is possible

Ceph in production

- Every OSD has an object journal
- SSD journals are recommended best practice
- Tiered storage can improve performance
 - An SSD tier can dramatically improve performance

Ceph in cost-effective production

Erasur encoding can greatly reduce storage costs

- Similar approach to RAID5, RAID6
- Data chunks and coding chunks
- Negative performance impact
- To use block devices, a cache tier is required

ceph-deploy usage

Accessing demo environment

5 VMs on AWS EC2, accessed via `ssh`:

```
$ ssh-add .ssh/id_rsa
$ ssh ceph_deploy
$ ssh ceph1
$ ssh ceph2
$ ssh ceph3
$ ssh ceph_client
```

Using ceph-deploy

First we must install and set up `ceph-deploy` as root:

- Install `ceph-deploy`

```
$ ssh ceph_deploy
```

```
$ sudo zypper in ceph-deploy
```

ceph-deploy working directory

Recommendation: `ceph-deploy` creates important files in the directory it is run from.

So it is best to run `ceph-deploy` in an empty directory, and with a separate (i.e. `non-root`), which is `ceph` for us.

Install ceph using ceph-deploy

First Ceph needs to be installed on the nodes:

```
$ ceph-deploy install \  
  $NODE1 $NODE2 $NODE3 $CLIENT
```

Setting up the mon nodes

Deploy keys and config for onto the Ceph cluster:

```
$ ceph-deploy new $NODE1 $NODE2 $NODE3
```

- This will:
 - log into to each node,
 - create the keys,
 - ceph config file `ceph.conf`
- These files will be in the current working dir
- One should inspect the initial `ceph.conf` file

Looking at `ceph.conf`

- Many tuning options can be set in `ceph.conf`
- Identical on all ceph nodes
- Good idea to set up `ceph.conf` properly now
- Older versions needed many sections in `ceph.conf`
- Newer versions need very few options
 - In most production setups, `public` and `private` networks would be used
- see `cat ceph.conf` for the canonical one

Looking at `ceph.conf` part 2

Most settings are in the `global` section of `ceph.conf`

For example, sometimes explicit networks need to be setup for Ceph:

```
public network = 10.121.0.0/16
```

The options are well documented on the Ceph web site.

Ceph and "size"

By default Ceph replicates every file stored 3 times.

If running a smaller cluster with only 2 OSDs, the default of 3 replications need to be reduced to 2 by adding the following line in the `global` section of `ceph.conf`:

```
osd pool default size = 2
```

Creating the mon daemons

Create the initial mon service on created nodes:

```
$ ceph-deploy mon create-initial
```

Creating the osd daemons

- Setup and prepare disks for Ceph:

```
$ ceph-deploy osd prepare \  
  $NODE1:xvd{b,c,d}  
$ ceph-deploy osd prepare \  
  $NODE2:xvd{b,c,d}  
$ ceph-deploy osd prepare \  
  $NODE3:xvd{b,c,d}
```

Note: The device name changed due to AWS.

Install calamari bits

For a graphical web management interface, the following needs to be done:

```
$ sudo zypper in calamari-clients
$ sudo calamari-ctl initialize
$ ceph-deploy calamari connect --master \  
`hostname` $NODE1
$ ceph-deploy calamari connect --master \  
`hostname` $NODE2
$ ceph-deploy calamari connect --master \  
`hostname` $NODE3
```

There's now a working Ceph setup!

Check out the cluster:

```
$ ceph-deploy disk list ceph1  
$ ssh ceph1
```

Ceph administration works via the `root` account:

```
$ sudo bash
```


Explore the Ceph cluster

- Look at the disks:

```
# parted --list
```

- Notice Ceph journal and data partitions
- Notice file system used under Ceph journal

```
# ceph df
```

Looking at the Ceph cluster

```
# ceph osd tree
# id weight type name up/down reweight
-1 0.08995 root default
-2 0.02998 host $NODE1
0 0.009995 osd.0 up 1
1 0.009995 osd.1 up 1
2 0.009995 osd.2 up 1
-3 0.02998 host $NODE2
3 0.009995 osd.3 up 1
4 0.009995 osd.4 up 1
5 0.009995 osd.5 up 1
...
```

OSD weighting

Each OSD has a weight:

- The higher the weight, the more likely data will be written
- Weight of zero will drain an OSD
 - This is a good way to drain an OSD

Monitoring the Ceph cluster

```
# ceph status
```

Is the ceph cluster healthy?

```
# ceph health  
# ceph mon stat  
# ceph osd stat  
# ceph pg stat  
# ls /var/log/ceph
```

Continuous messages:

```
# ceph -w
```

Updating ceph.conf

On ceph_deploy:

```
$ vi ceph.conf
```

Add the following lines:

```
[mon]  
mon_clock_drift_allowed = 0.100
```

Updating ceph.conf (continued)

```
$ ceph-deploy --overwrite-conf config  
push ceph{1,2,3}
```

On all nodes:

```
$ ssh ceph1 sudo rcceph restart  
$ ssh ceph2 sudo rcceph restart  
$ ssh ceph3 sudo rcceph restart
```

Working with Ceph services

As root on ceph3:

```
$ ssh ceph3  
$ sudo bash  
# rcceph
```

(look at the options)

```
# rcceph status
```

Simulating maintenance work

```
# systemctl status ceph-mon@ceph3.service  
# systemctl status ceph-osd@6.service  
# systemctl stop ceph-mon@ceph3.service
```

Use `ceph status` and other Ceph options to see what happens.

```
# systemctl start ceph-mon@ceph3.service  
# systemctl stop ceph-osd@6.service  
# systemctl start ceph-osd@6.service
```


Using Calamari to explore the Ceph cluster

- Point a browser at calamari:

```
# xdg-open `sed -ne " s/$ADMIN *\  
(.*\) /http://\1/p" /etc/hosts`
```

- Log in
- (Hosts requesting to be managed by Calamari)
- Click *Add*
- Explore cluster using the browser
- Stop a `mon` on a node and check Calamari
- Don't forget to restart the `mon`!
- Stop an `osd` on a node and check Calamari
- Don't forget to restart the `osd`!



RADOS Block Devices (RBD)

Ceph's RADOS Block Devices (RBD)

Ceph's RADOS Block Devices (RBD) can interact with OSDs using kernel modules or the `librbd` library.

This page discusses how to use the kernel module.

Still, for the config and shell utilities, ceph needs to be installed on the host - without admin rights for the cluster though. Login to host `ceph_client`:

```
$ ssh ceph_client
$ sudo bash
# zypper in ceph
```

Block device creation

To create a block device image, on any of the ceph or client nodes:

```
# rbd create {image-name} --size \  
  {megabytes} --pool {pool-name}  
# rbd create media0 --size 500 --pool rbd
```

Retrieve image information -

```
# rbd --image media0 -p rbd info
```

Map a block device

```
# rbd map media0
```



Block device management

Show mapped block devices, benchmark it quickly:

```
# rbd showmapped
# rados -p media0 bench 300 write -t 400
```

Mount the block device and perform some read/write operations.

```
# mkfs.ext3 /dev/rbd1
# mount /dev/rbd1 /mnt
# dd if=/dev/urandom of=/mnt/test.avi \
  bs=1M count=250
```

Find a file inside an rbd device

Find the pg the file ended up in:

```
# ceph osd map rbd test.avi
  osdmap e53 pool 'rbd' (2) object
'test.avi' -> pg 2.ac8bd444 (2.4) -> up
([5,0,7], p5) acting ([5,0,7], p5)
```

Find out which node hosts the primary osds of a file:

```
# ceph osd tree
-> -3 0.02998 host ip-10-81-16-108
   3 0.009995 osd.3 up 1
   4 0.009995 osd.4 up 1
   5 0.009995 osd.5 up 1
```



Find a file inside an rbd device (cnt.)

The file is here:

```
# ls -al /var/lib/ceph/osd/\  
ceph-5/current/2.4_head
```

Ceph stores objects sparsely, i.e. facilitates thin provisioning e.g. for VM images.

Verifying replication

Take out the node that has the file's primary osd:

```
# ceph osd out 4
```

Check the data is still there:

```
# dd of=/dev/nul if=/mnt/test.avi \  
bs=1M count=250
```


Cleaning up

Unmount the block device and unmap the block device:

```
# umount /mnt  
# rbd unmap /dev/rbd/rbd/media0
```

Remove the block device image

```
# rbd rm {image-name} -p {pool-name}  
# rbd rm media0 -p rbd
```

Ceph pools

What are pools?

- "Pools" are logical partitions for storing objects
 - Define data resilience
 - Replication "size"
 - Erasure encoding and details

Pool Properties

- Have Placement Groups
 - Number of hash buckets to store data
 - Typically approximately 100 per OSD / Terabyte
- Have mapping to CRUSH map rules
 - CRUSH map defines data distribution
- Have ownership
- Have quotas

Basic pool usage

Login to any of the ceph nodes:

```
$ ssh ceph1  
$ sudo bash
```

To list pools:

```
# ceph osd lspools
```

- Three default pools (removable)
 - are defaults for tools
- `rbd` tools default to using the `rbd` pool

Adding a pool

```
# ceph osd pool create \  
{pool-name} {pg-num} [{pgp-num}] \  
[replicated] [crush-ruleset-name]
```

Example:

```
# ceph osd pool create \  
suse_demo_pool 512
```

Explaining pg-num

- "pg-num" is number of chunks data is placed in
- Used for tracking groups of objects and their distribution
- Default value of 8
 - Too low even for test system

pg-num recommended value

- With less than 5 OSDs, set `pg-num` to 128
- Between 5 and 10 OSDs, set `pg-num` to 512
- Between 10 and 50 OSDs, set `pg-num` to 4096

Trade-offs in pg-num value

- Too many
 - More peering
 - More resources used
- Too few
 - Large amounts of data per pool group
 - Slower recovery from failure
 - Slower re-balancing

Setting quotas on pools

```
# ceph osd pool set-quota \  
  {pool-name} [max_objects {obj-count}] \  
  [max_bytes {bytes}]
```

Example:

```
# ceph osd pool set-quota data \  
  max_objects 10000
```

Set to 0 to remove pool quota.

Show pool usage

```
# rados df
```

Shows stats on all pools

Get/set pool attributes

pool properties are a set of key value pairs. We mentioned "size" is number of replicas.

```
# ceph osd pool get suse_demo_pool size
size: 3
# ceph osd pool set suse_demo_pool size 2
size: change from 3 to 2
# ceph osd pool get suse_demo_pool size
size: 2
```

Pool snapshots

To make a snapshot:

```
# ceph osd pool mksnap suse_demo_pool  
testsnap
```

To remove a snapshot:

```
# ceph osd pool rmsnap suse_demo_pool  
testsnap
```

Removing pools

CAUTION! removing pools will remove all data stored in the pool!

```
# ceph osd pool delete \  
  suse_demo_pool suse_demo_pool \  
  --yes-i-really-really-mean-it
```

Ceph CRUSH map

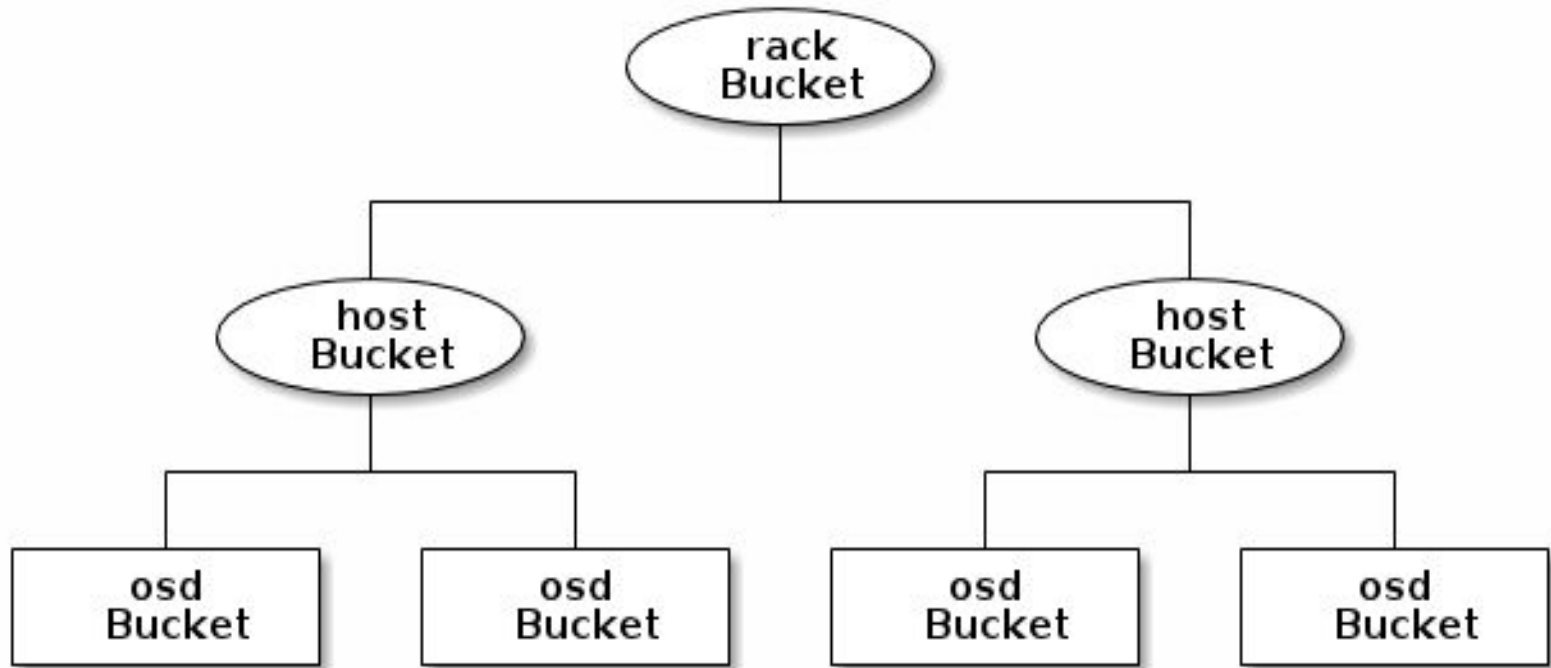
Ceph CRUSH map overview

- Controlled, scalable, decentralized placement of replicated data
- CRUSH map decides where data is distributed
- Buckets
- Rules map pools to the crushmap

Buckets

- Group OSDs into groups for replication purposes
 - type 0 OSD (usually a disk but could be smaller)
 - type 1 host (usually a disk but could be smaller)
 - type 2 chassis (eg blade)
 - type 3 rack ...
 - type 7 room
 - type 8 datacenter
 - type 9 region
 - type 10 root

Buckets can contain buckets



Inktank Storage, Inc., CC-BY-SA



CRUSH map rules

- How to use buckets
 - Pick bucket of type
 - Should buckets inside bucket be used?
- How many replicas to store (size)

Modifying a CRUSH map

- Ceph has a default one
- Decompile the CRUSH map and edit it directly
 - This is good for complex changes
 - more likely to make errors
 - Syntax checking happens on re-compilation
- Or: Setting up via using command line
 - Best way for normal use as each step is validated

Example of decompiling a CRUSH map

Getting CRUSH map from ceph:

```
# ceph osd getcrushmap -o  
crush.running.map
```

Decompile CRUSH map to text:

```
# crushtool -d crush.running.map -o  
map.txt
```

Edit:

```
# vim map.txt
```

Example of decompiling a CRUSH map (cnt.)

Re-compile binary CRUSH map:

```
# crushtool -c map.txt -o crush.new.map
```

Setting the CRUSH map for ceph:

```
# ceph osd setcrushmap -i crush.new.map
```

Example of adding a rack bucket

```
# ceph osd crush add-bucket rack1 rack
# ceph osd crush add-bucket rack2 rack
```

Racks are currently empty:

```
# ceph osd tree
# id weight type name up/down reweight
-6 0 rack rack2
-5 0 rack rack1
-1 11.73 root default
-2 5.46 host test1
0 1.82 osd.0 up 1
...
```



Example of moving a OSD

Syntax:

```
# ceph osd crush set {id} {name} {weight}  
pool={pool-name} [{bucket-type}={bucket-  
name} ...]
```

Example:

```
# ceph osd crush set osd.0 1.0  
root=default datacenter=dc1 room=room1  
row=foo rack=bar host=foo-bar-1
```

There's a huge amount of options to play with.



Adding an OSD to a rack bucket

So with the new types, now add the OSDs to the rack bucket:

```
# ceph osd crush move ip-10-...  
rack=rack1  
# ceph osd crush move ip-10-...  
rack=rack2  
moved item id -3 name 'test2' to location  
{rack=rack1} in crush map
```

Adding an OSD to a rack bucket (cnt.)

You can now see the bucket 'tree'

```
# ceph osd tree
-6 0.81 rack rack2
 2 1.82 osd.2 up 1
 3 1.82 osd.3 up 1
-5 10.92 rack rack1
 0 1.82 osd.0 up 1
 1 1.82 osd.1 up 1
```

Putting crushmap and pools together

- CRUSH map distributes ***where*** data is distributed
 - Use rules to map pools to buckets
- Pools describe ***how*** data is distributed
 - Specifying size and replication mode # Tiering and storage

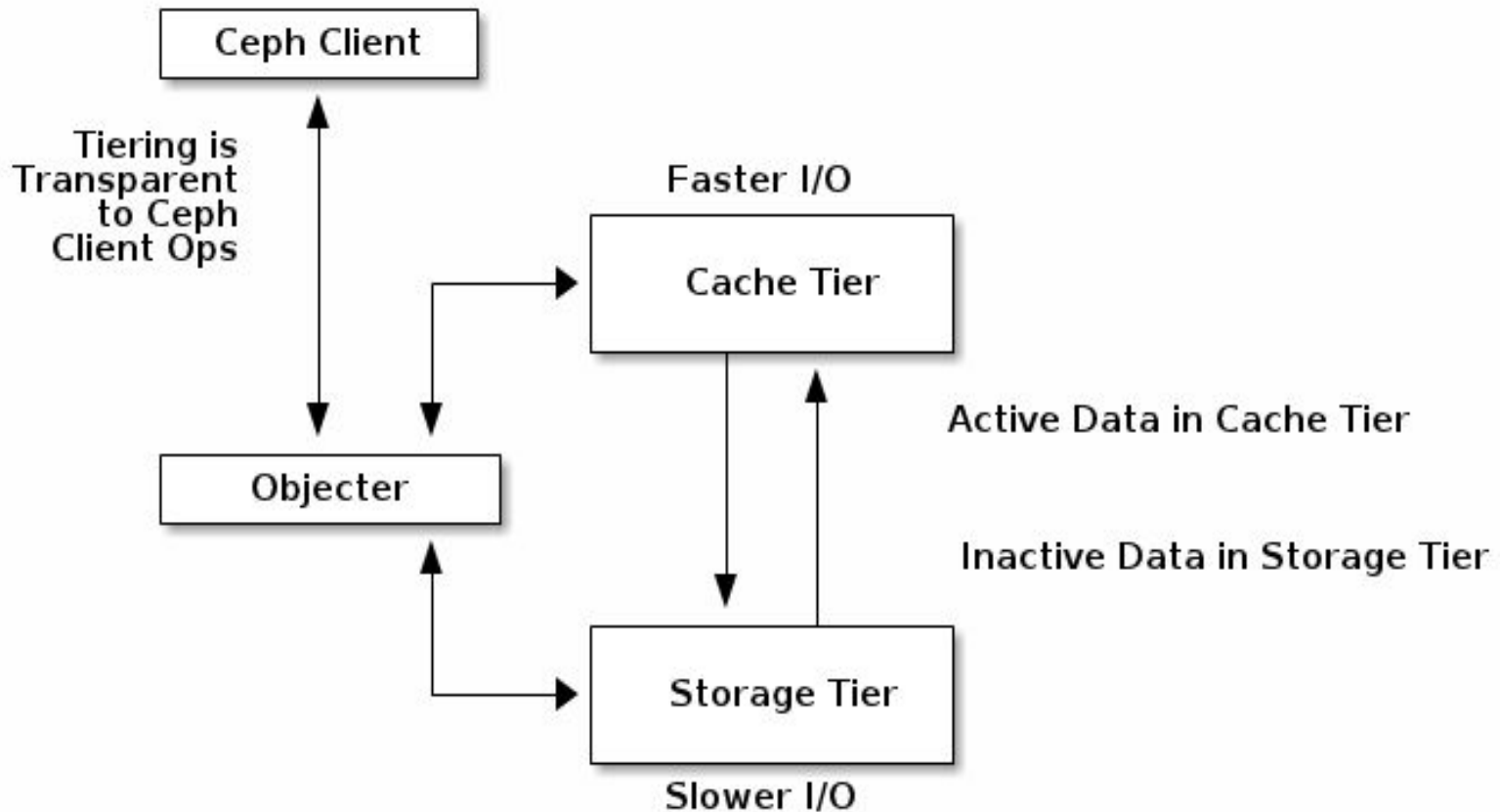
Why do we want tiered storage?

- Faster storage is more expensive
 - What's the price per terabyte for flash disks?
- Active data is usually a subset of data
 - So cost savings if managed automatically
- Ceph-specific
 - Erasure encoded storage cannot provide block devices
 - Via a cache tier it can

Is tiered storage simple?

- In ceph it's "just" a cache service.
 - We expect further Tiering options to be developed.
- Caching is not complex in theory.
 - http://en.wikipedia.org/wiki/Cache_%28computing%29
- Caching is subtle in practice.
 - "Hot" adjustment is possible
 - "Hot" removal is possible.
- ceph tiering summary
 - Performance will benefit!
 - You can tune it over time.

Tiered storage diagram.



Inktank Storage, Inc., CC-BY-SA



Setting up a cache

Setting up a cache tier involves associating a backing storage pool with a cache pool

```
# ceph osd tier add {storagepool} \  
  {cachepool}
```

For example:

```
# ceph osd tier add cold-storage \  
  hot-storage
```

Cache tier mode overview

- We need to decide on type.
 - writeback
 - For caching writes
 - readonly
 - For caching reads
 - forward
 - While removing write
 - Allow flushing
 - none
 - To disable cache

Setting the tier mode

To set the cache mode, execute the following:

```
# ceph osd tier cache-mode {cachepool} \  
  {cache-mode}
```

For example:

```
# ceph osd tier cache-mode hot-storage \  
  writeback
```

And also for a writeback cache

- One additional step for writeback
 - redirect traffic to the cache.

```
# ceph osd tier set-overlay \  
{storagepool} {cachepool}
```

For example:

```
# ceph osd tier set-overlay cold-  
storage \  
hot-storage
```

Configuring a Cache Tier Options.

- Cache tiers are "like" pools.
 - many configuration options.
 - Options are set just like pools.
 - get "key"
 - Options include all pool settings.
 - size
 - ..
- Example

```
# ceph osd pool set {cachepool} {key} \  
{value}
```

Bloom filter option

Binning accesses over time allows Ceph to determine whether a Ceph client accessed an object at least once, or more than once over a time period (cage vs. temperature).

Ceph's production cache tiers use a Bloom Filter for the `hit_set_type`:

```
# ceph osd pool set {cachepool} \  
hit_set_type bloom
```

For example:

```
# ceph osd pool set hot-storage \  
hit_set_type bloom
```



Example settings for a cache Tier

The "hit_set_count" and "hit_set_period" define how much time each HitSet should cover, and how many such HitSets to store. Currently there is minimal benefit for hit_set_count bigger than 1 since the agent does not yet act intelligently on that information.

```
# ceph osd pool set {cachepool} \  
hit_set_count 1  
# ceph osd pool set {cachepool} \  
hit_set_period 3600  
# ceph osd pool set {cachepool} \  
target_max_bytes 10000000000000
```

RAM and settings for a cache Tier

- All "hit_set_count" HitSets are loaded into RAM.
 - When the agent is active
 - flushing cache objects
 - evicting cache objects

The longer the "hit_set_period" and the "higher the count", the more RAM the osd daemon consumes.

Cache Sizing

The cache tiering agent performs two main functions:

- Flushing:
 - The agent identifies modified (or dirty) objects and forwards them to the storage pool for long-term storage
- Evicting:
 - The agent identifies objects that haven't been modified (or clean) and evicts the least recently used among them from the cache

Relative Sizing Introduction

The cache tiering agent can flush or evict objects relative to the size of the cache pool. When the cache pool consists of a certain percentage of modified (or dirty) objects, the cache tiering agent will flush them to the storage pool.

Relative Sizing Dirty Ratio

To set the "cache_target_dirty_ratio", execute the following:

```
# ceph osd pool set {cachepool} \  
  cache_target_dirty_ratio {0.0..1.0}
```

For example, setting the value to 0.4 will begin flushing modified (dirty) objects when they reach 40% of the cache pool's capacity:

```
# ceph osd pool set hot-storage \  
  cache_target_dirty_ratio 0.4
```

Relative Sizing Full Ratio

When the cache pool reaches a certain percentage of its capacity, the cache tiering agent will evict objects to maintain free capacity. To set the `cache_target_full_ratio`, execute the following:

```
# ceph osd pool set {cachepool} \  
  cache_target_full_ratio {0.0..1.0}
```

For example, setting the value to 0.8 will begin flushing unmodified (clean) objects when they reach 80% of the cache pool's capacity:

```
# ceph osd pool set hot-storage \  
  cache_target_full_ratio 0.8
```



Absolute Sizing

The cache tiering agent can flush or evict objects based upon the total number of bytes or the total number of objects. To specify a maximum number of bytes, execute the following:

```
# ceph osd pool set {cachepool} \  
target_max_bytes {#bytes}
```

For example, to flush or evict at 1 TB, execute the following:

```
# ceph osd pool hot-storage \  
target_max_bytes 1000000000000
```



Absolute Sizing (cnt.)

To specify the maximum number of objects, execute the following:

```
# ceph osd pool set {cachepool} \  
target_max_objects {#objects}
```

For example, to flush or evict at 1M objects, execute the following:

```
# ceph osd pool set hot-storage \  
target_max_objects 1000000
```

Relative / Absolute Cache Sizing Limits

- You can specify "Relative" and "Absolute" Limits.
 - will trigger when either limit happens.
- You don't need to set "Relative" and "Absolute" Limits.
 - Will depend on your work load.

Cache Age Flushes

One can specify the minimum age of an object before the cache tiering agent flushes a recently modified (or dirty) object to the backing storage pool:

```
# ceph osd pool set {cachepool} \  
  cache_min_flush_age {#seconds}
```

For example, to flush modified (or dirty) objects after 10 minutes, execute the following:

```
# ceph osd pool set hot-storage \  
  cache_min_flush_age 600
```

Cache Age Eviction

The minimum age of an object can be specified before it will be evicted from the cache tier:

```
# ceph osd pool {cache-tier} \  
  cache_min_evict_age {#seconds}
```

For example, to evict objects after 30 minutes, execute the following:

```
# ceph osd pool set hot-storage \  
  cache_min_evict_age 1800
```

Removing a Cache Tier

- Procedure dependent on type.
 - For writeback cache.
 - For read-only cache.

Removing a Read-Only Cache

- read-only cache does not have modified data
 - Easier to remove
 - No modified data
 - So you can just disable.

Removing a Read-Only Cache (Disable)

Change the cache-mode to none to disable it.

```
# ceph osd tier cache-mode {cachepool} \  
none
```

For example:

```
# ceph osd tier cache-mode hot-storage \  
none
```

Removing a Read-Only Cache from the backing pool.

Remove the cache pool from the backing pool.

```
# ceph osd tier remove {storagepool} \  
  {cachepool}
```

For example:

```
# ceph osd tier remove cold-storage \  
  hot-storage
```

Removing a Writeback Cache

Since a writeback cache may have modified data, one must take steps to ensure that no recent changes to objects are lost in the cache, before it is disabled and removed.

Forward a Writeback Cache

Change the cache mode to forward so that new and modified objects will flush to the backing storage pool.

```
# ceph osd tier cache-mode {cachepool} \  
forward
```

For example:

```
# ceph osd tier cache-mode hot-storage \  
forward
```

Inspection of a Writeback Cache

Ensure that the cache pool has been flushed. This may take a few minutes:

```
# rados -p {cachepool} ls
```

Flushing of a Writeback Cache

If the cache pool still has objects, flush them manually. For example:

```
# rados -p {cachepool} cache-flush-evict-  
all
```

Remove Forward on Writeback Cache

Remove the overlay so that clients will not direct traffic to the cache.

```
# ceph osd tier remove-overlay \  
  {storagetier}
```

For example:

```
# ceph osd tier remove-overlay \  
  cold-storage
```


Remove Writeback Cache Final

Finally, remove the cache tier pool from the backing storage pool.

```
# ceph osd tier remove {storagepool} \  
  {cachepool}
```

For example:

```
# ceph osd tier remove cold-storage \  
  hot-storage
```

Tiered Storage Summary

- You want Tiered Storage in production.
 - As Storage is never fast enough.
- It will increase RAM usage for OSD daemons.
 - So be careful with your settings.
- You can use use RBD to access Erasure encoded storage.
 - But only with a cache tier.
- You will need to tune this to your sites workload.
 - To get the best for your budget.
 - You can adjust Tiering.
 - While ceph is running.
 - While clients are running.



Credits

- These tiering instructions were developed from:
 - <http://ceph.com/docs/master/rados/operations/cache-tiering/>

Erasure coding and ceph

Introduction

We set this up on 3 hosts with a lot of disks. This requires setting up a lot disks as a base system. Then setting up two rule sets, then multiple pools.

Prerequisites

You will need a ceph cluster with quite a few disks. Erasure coding does require a significant list of disks to make an erasure encoded pool.

An erasure encoded pool cannot be accessed directly using rbd. For this reason we need a cache pool and an erasure pool. This not only allows supporting rbd but increases performance.

In a production system we would recommend using faster media for the cache and slower media for the erasure encoded pool.

For our test install

We don't have enough hosts to test erasure encoding across hosts you probably want to set up erasure encoding across disks.

Make the following change to each rule set.

```
-step chooseleaf firstn 0 type host  
+step chooseleaf firstn 0 type osd
```

This will set the redundancy on a per OSD rather than per host basis.

Initial setup

We made two groupings of of OSD following two different selection rules one for ssd disks one for hdd.

Please see "row ssd" to understand this.

To show the pool.

```
# ceph osd lspools
0 data,1 metadata,2 rbd,3 e1,4 e2,6
ECtemppool,8 ecpool,9 ssd,
```


Setting up Erasure encoded pool

Erasure encoding background

Erasure coding makes use of a mathematical equation to achieve data protection. The entire concept revolves around the following equation:

$n = k + m$ where ,

k = The number of chunks original data divided into.

m = The extra codes added to original data chunks to provide data protection. For ease of understanding, it can be considered the reliability level.

n = The total number of chunks created after erasure coding process.

Erasure encoding background (cnt.)

In continuation to erasure coding equation, there are couple of more terms which are :

Recovery: To perform recovery operation we would require any k chunks out of n chunks and thus can tolerate failure of any m chunks

Reliability Level : We can tolerate failure upto any m chunks.

Encoding Rate (r) : $r = k / n$, where r smaller 1

Storage Required : $1 / r$

Erasure encoding Profiles

Erasure encoding will not be directly setup with ceph, instead there'll be an erasure encoding profile, using the profile to create the pool.

Setting up an erasure encoding profile

To create an erasure encoded profile.

```
# ceph osd erasure-code-profile set EC-temp-pool
```

To list erasure encoded profiles.

```
# ceph osd erasure-code-profile ls  
EC-temp-pool  
profile1
```

To delete erasure encoded profile.

```
# ceph osd erasure-code-profile rm EC-temp-pool
```



Erasure encoding Profiles (cnt.)

To show an erasure encoded profile.

```
# ceph osd erasure-code-profile get EC-  
temp-pool  
directory=/usr/lib64/ceph/erasure-code  
k=2  
m=1  
plugin=jerasure  
technique=reed_sol_van
```

Using the force option to set all properties of a profile:

```
# ceph osd erasure-code-profile set EC-  
temp-pool ruleset-failure-domain=osd k=4  
m=2 --force
```



Erasure encoding Profiles (cnt.)

The following plugins exists.

- jerasure (jerasure)
- Intel/Intelligent Storage Acceleration Library (isa)
- Locally repairable erasure (lrc)

The choice of plugin is primarily decided by the workload hardware and use cases, particularly on recovery from disk failure.

Creating a pool from an erasure encoding profile

To create a pool

```
# ceph osd pool create <Pool_name>  
<pg_num> <pgp_num> erasure  
<EC_profile_name>
```

For example:

```
# ceph osd pool create ECtemppool 128 128  
erasure EC-temp-pool  
pool 'ECtemppool' created
```


Creating a pool from an erasure encoding profile (cnt.)

To validate the list of available pools using rados.

```
# rados lspools  
data  
metadata  
rbd  
ECtemppool
```

Creating a pool from an erasure encoding profile (cnt.)

To verify this worked:

```
# ceph osd dump | grep -i erasure
pool 22 'ECTemppool' erasure size 6
min_size 2 crush_ruleset 1 object_hash
rjenkins pg_num 128 pgp_num 128
last_change 2034 owner 0 flags hashpspool
stripe_width 4096
```

Writing directly to the erasure encoding pool

- writing to an erasure encoding pool with the rados interface
- To list the content of the pool: `# rados -p ECtemp pool ls`
- To put a file in: `# rados -p ECtemp pool put object.1 testfile.in`
- To get a file: `# rados -p ECtemp pool get object.1 testfile.out`
- To use rbd interface we need another pool

Summary of Erasure encoding.

- Makes storage much cheaper.
 - With no reduction in reliability.
- Makes storage slower.
- Makes recovery slower.
- Requires more CPU power.
- You will probably want to add a cache tier.
 - To maximize the performance.
- Can access via RADOS.
- For RBD access requires a cache tier.
 - But you probably want one anyway.

Credits

These instructions are based upon:

- <http://karan-mj.blogspot.de/2014/04/erasure-coding-in-ceph.html>
- <http://docs.ceph.com/docs/master/rados/operations/pools/>

SUSE Storage Roundtable (OFOR7540):
- Thu, 14:00

And many thanks to Owen Syngé
for content and live support!



Corporate Headquarters
Maxfeldstrasse 5
90409 Nuremberg
Germany

+49 911 740 53 0 (Worldwide)
www.suse.com

Join us on:
www.opensuse.org

Unpublished Work of SUSE LLC. All Rights Reserved.

This work is an unpublished work and contains confidential, proprietary and trade secret information of SUSE LLC. Access to this work is restricted to SUSE employees who have a need to know to perform tasks within the scope of their assignments. No part of this work may be practiced, performed, copied, distributed, revised, modified, translated, abridged, condensed, expanded, collected, or adapted without the prior written consent of SUSE. Any use or exploitation of this work without authorization could subject the perpetrator to criminal and civil liability.

General Disclaimer

This document is not to be construed as a promise by any participating company to develop, deliver, or market a product. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. SUSE makes no representations or warranties with respect to the contents of this document, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. The development, release, and timing of features or functionality described for SUSE products remains at the sole discretion of SUSE. Further, SUSE reserves the right to revise this document and to make changes to its content, at any time, without obligation to notify any person or entity of such revisions or changes. All SUSE marks referenced in this presentation are trademarks or registered trademarks of Novell, Inc. in the United States and other countries. All third-party trademarks are the property of their respective owners.

