

Configurazione di tiered storage

Marica Antonacci

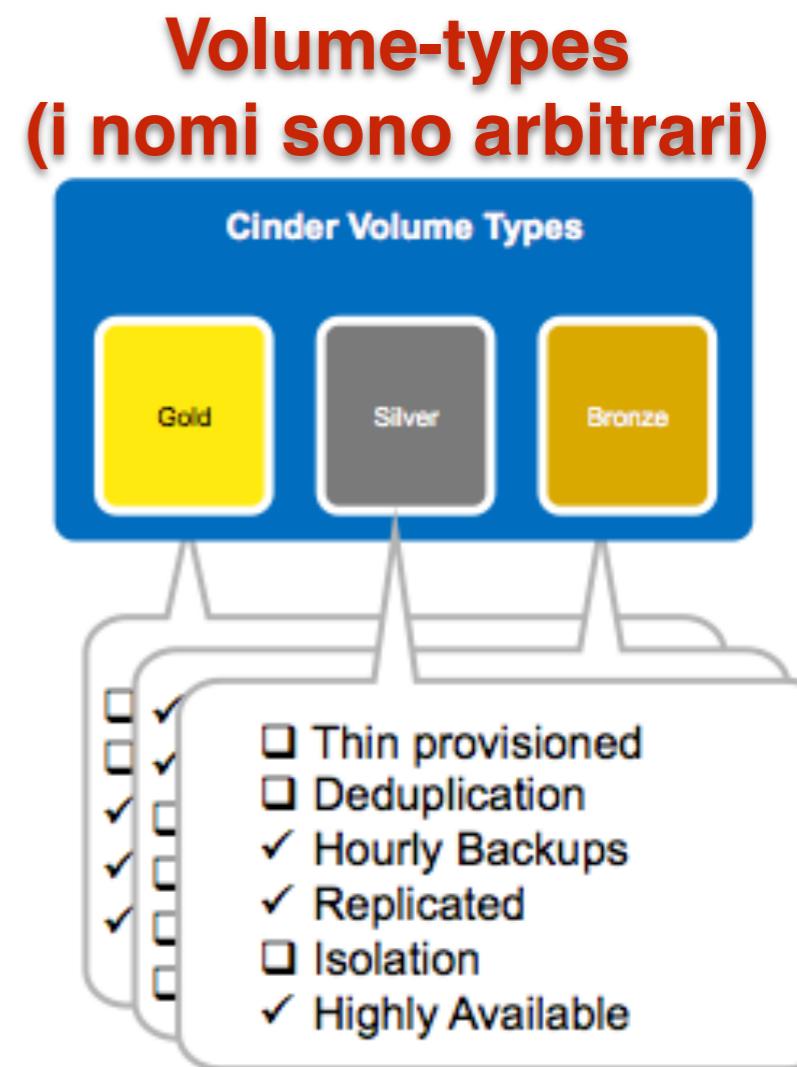
*Scuola di Cloud Storage
Bari, 9-11 Dicembre 2015*

Outline

- QoS in Openstack
- Multi-tier storage configuration example
- Cache tiering in Ceph
 - setup & configuration

Quality of Service in Openstack

- I Volume type possono essere usati per fornire agli utenti differenti livelli (tier) di storage:
 - ✓ performance diverse (p.e. HDD tier, mixed HDD-SDD tier, o SSD tier),
 - ✓ resilienza (selezionando differenti livelli di RAID, o replica)
 - ✓ specifiche features (p.e. compressione, data-deduplication, etc.).



Uso dei volume_type

- I volume-type possono essere utilizzati per controllare dove i volumi verranno allocati:

```
# cinder create --volume_type lvm --display_name my-test 1
```

- Ogni volume-type contiene un set di coppie chiave-valore chiamati **extra-specs**. Queste informazioni sono usate dal Cinder-Scheduler per prendere decisioni sul placement dei volumi in base alle capabilities dei backend disponibili

```
# cinder type-create lvm
# cinder type-key lvm set volume_backend_name=LVM_iSCSI
# cinder extra-specs-list
```

ID	Name	extra_specs
0c91c16f-f3ab-493c-960b-75eb3f10d90e	services	{u'volume_backend_name': u'LVM_SERVICES'}
143b23ca-e47c-401f-aa0b-8b9a408c72b7	data	{u'volume_backend_name': u'CEPH_DATA'}
5c1d93c7-64a9-496a-b0dc-2f675bddb057	encrypted-data	{u'volume_backend_name': u'CEPH_DATA_ENCR'}

Multi-tier configuration example

- Per esempio, assumiamo di avere 2 pool su Ceph che utilizzano storage device differenti:
- il pool “*cinder-sata*” usa un rack SATA
- il pool “*cinder-ssd*” usa un rack SSD

```
# Multi backend options

# Define the names of the groups for multiple volume backends
enabled_backends=rbd-sata,rbd-ssd

# Define the groups as above
[rbd-sata]
volume_driver=cinder.volume.driver.RBDDriver
rbd_pool=cinder-sata
volume_backend_name=RBD_SATA
# if cephX is enable
#rbd_user=cinder
#rbd_secret_uuid=<None>
[rbd-ssd]
volume_driver=cinder.volume.driver.RBDDriver
rbd_pool=cinder-ssd
volume_backend_name=RBD_SSD
# if cephX is enable
#rbd_user=cinder
#rbd_secret_uuid=<None>
```

Creiamo in cinder 2 **volume-types**, ognuno associato ad un backend. L’utente potrà richiedere un volume di tipo **Gold** (SSD) o **Silver** (SATA) in base al suo use-case

QoS e Rate limiting

- Feature introdotta a partire da **Havana**
- Implementa il supporto **QoS** in Nova e Cinder (sfruttando il rate limiting già supportato in KVM e QEMU attraverso libvirt) - utile nel caso in cui lo storage non espone questa funzionalità
- Il limiting può quindi essere realizzato dal “frontend” (hypervisor) o dal “backend” (storage subsystem) o entrambi
- **Backend**: campi specifici definiti dal vendor:
 - ❖ HP 3PAR (IOPS, tput: min, max; latency, priority)
 - ❖ Solidfire (IOPS: min, max, burst)
 - ❖ NetApp* (QoS Policy Group)
 - ❖ Huawei* (priority)

**defined through extra specs*

Rate limiting options

- **Frontend** QoS options:
 - throughput
 - `total_bytes_sec`: the total allowed bandwidth for the guest per second
 - `read_bytes_sec`: sequential read limitation
 - `write_bytes_sec`: sequential write limitation
 - IOPS
 - `total_iops_sec`: the total allowed IOPS for the guest per second
 - `read_iops_sec`: random read limitation
 - `write_iops_sec`: random write limitation
- Il file di definizione della VM a cui viene agganciato il volume con `qos-specs` conterrà un campo xml extra “`<iotune>`” nella sezione `<disk>`. Es.

```
<iotune>
  <read_iops_sec>2000</read_iops_sec>
  <write_iops_sec>1000</write_iops_sec>
</iotune>
```

Rate limiting: cinder CLI

create qos specs

```
$ cinder qos-create <name> <key=value>  
[<key=value> ...]
```

```
$ cinder qos-create high-iops consumer="front-end" read_iops_sec=2000  
write_iops_sec=1000  
+-----+  
| Property | Value |  
+-----+  
| consumer | front-end |  
| id | c38d72f8-f4a4-4999-8acd-a17f34b040cb |  
| name | high-iops |  
| specs | {u'write_iops_sec': u'1000', u'read_iops_sec': u'2000'} |  
+-----+
```

Associate qos specs with specific volume type

```
$ cinder qos-associate <qos_specs> <volume_type_id>
```

Esempi: extra-specs + qos-specs

Mettiamo insieme un po' tutto:

- volume-types,
- extra-specs,
- qos-specs

Volume Type	Extra Specs	QoS Specs
Gold	{netapp:disk_type=SSD, netapp_thick_provisioned=True}	{}
Silver	{}	{total_iops_sec=500}
Bronze	{volume_backend_name=lvm}	{total_iops_sec=100}

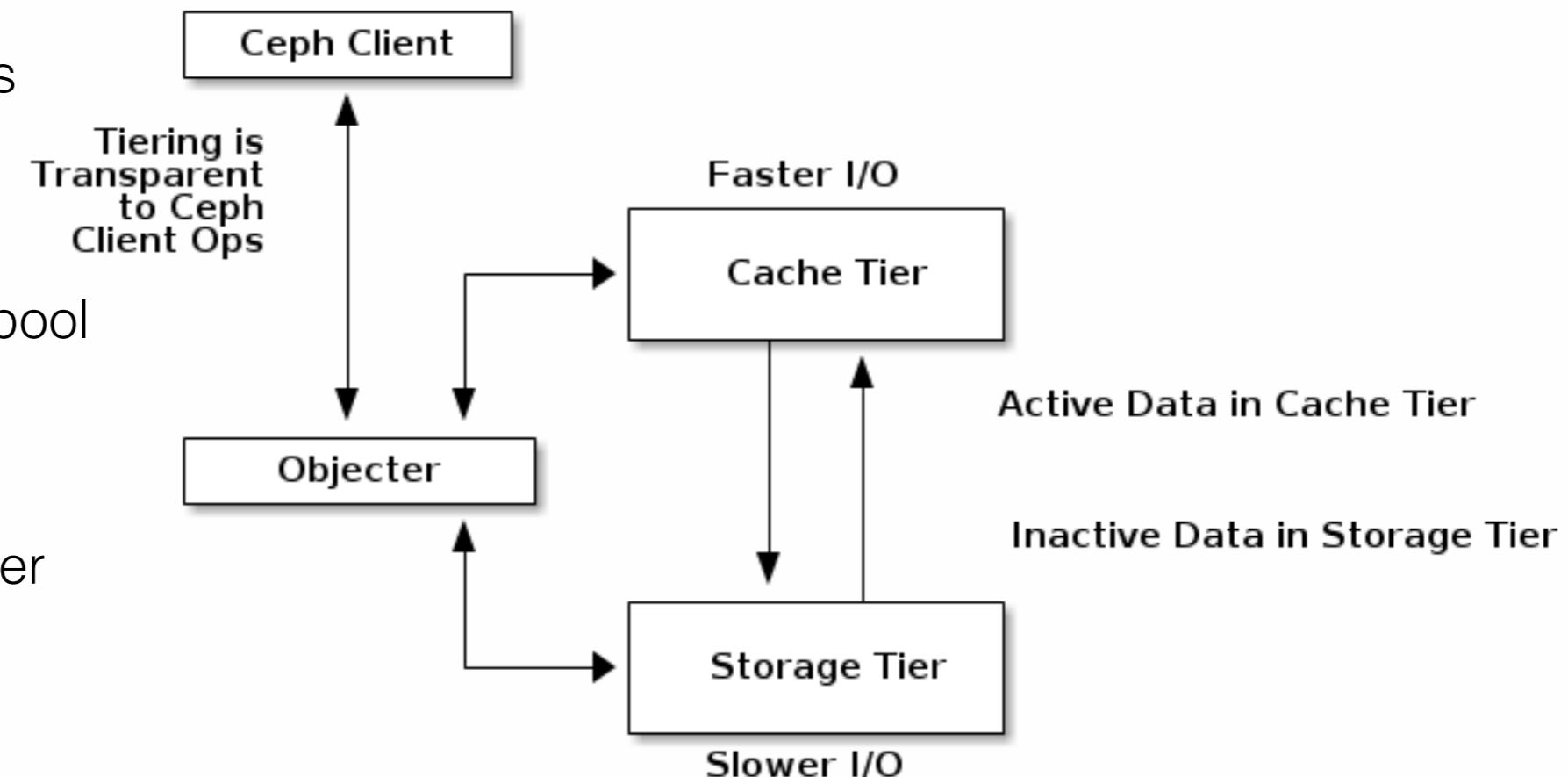
QoS “dinamico”

- **Volume-Retype**: consente di cambiare il tipo di volume dopo la sua creazione.
 - Questa funzionalità è utile per esempio per modificare il livello di QoS dinamicamente (nel caso in cui un volume sia sottoposto ad utilizzo pesante nel tempo e si renda necessario il passaggio ad un tier che offra un servizio migliore).

Cache tiering set-up

- To enable cache tiering, you need to have two pools: the **base tier pool** and the **cache tier pool**.
- The cache tiering setup involves the following five steps:

- ✓ Adding SSDs as OSDs
- ✓ Edit the crush map
- ✓ Create the cache tier pool
- ✓ Create the cache tier
- ✓ Configure the cache tier



Important note: Once we enabled the cache tiering, Ceph clients with kernel 3.13 stopped working with 'feature set mismatch' error —> kernel upgrade was needed.

Cache tier configuration

- **hit_set_type**: enable hit set tracking for the cache pool; the production-grade cache tier uses bloom filters
- **hit_set_count**: the number of hits set to store for a cache pool
- **hit_set_period**: the duration of the hit set period in seconds to store for a cache pool
- **target_max_bytes**: the maximum number of bytes after the cache-tiering agent starts flushing/evicting objects from a cache pool
- **target_max_objects**: the maximum number of objects after which a cache-tiering agent starts flushing/evicting objects from a cache pool

Cache tier configuration

- **cache_min_flush_age** and **cache_min_evict_age**: the time in seconds a cache-tiering agent will take to flush and evict objects from a cache tier to a storage tier
- **cache_target_dirty_ratio**: the percentage of cache pool containing dirty (modified) objects before the cache-tiering agent flushes them to the storage tier
- **cache_target_full_ratio**: the percentage of cache pool containing unmodified objects before the cache-tiering agent flushes them to the storage tier

Cache tier

```
root@cephsrv01:~# time rados put -p rbd filetest1 obj500M  
real    0m6.358s  
user    0m0.392s  
sys     0m0.636s  
root@cephsrv01:~#  
root@cephsrv01:~#  
root@cephsrv01:~# time rados put -p test filetest2 obj500M  
real    0m12.045s  
user    0m0.412s  
sys     0m0.644s
```

**cache tiering
enabled for
pool rbd**

Monitor disk I/O on
per-disk basis

```
iostat -xd <update interval in seconds>
```

Where are my objects?

```
root@cephsrv01:~# ceph osd map rbd filetest1  
osdmap e466 pool 'rbd' (0) object 'filetest1' -> pg 0.25f30571 (0.171) -> up ([0,15,6], p0) acting ([0,15,6], p0)  
root@cephsrv01:~#  
root@cephsrv01:~# ceph osd map test filetest2  
osdmap e466 pool 'test' (6) object 'filetest2' -> pg 6.57f4c5aa (6.1aa) -> up ([9,4,15], p9) acting ([9,4,15], p9)  
root@cephsrv01:~#  
root@cephsrv01:~#  
root@cephsrv01:~# find /var/lib/ceph/osd/ -name filetest* -exec ls -l {} \;  
-rw-r--r-- 1 root root 524288000 May 25 22:14 /var/lib/ceph/osd/ceph-5/current/1.71_head/filetest1_head_25F30571_1  
-rw-r--r-- 1 root root 524288000 May 25 22:15 /var/lib/ceph/osd/ceph-4/current/6.1aa_head/filetest2_head_57F4C5AA_6  
root@cephsrv01:~#  
root@cephsrv01:~# ls -l obj500M  
-rw-r--r-- 1 root root 524288000 May 25 14:02 obj500M
```

osd.[5,11,17] —> cache tier

Test tools for RBD

- **FIO** (Flexible I/O Tester) with **librbd** support

https://telekomcloud.github.io/ceph/2014/02/26/ceph-performance-analysis_fio_rbd.html

```
$ git clone git://git.kernel.dk/fio.git  
$ cd fio  
$ ./configure  
[...]  
Rados Block Device engine yes  
[...]  
$ make
```

needs
librbd-dev

```
[global]  
ioengine=rbd  
clientname=admin  
pool=rbd  
rbdname=fio_test  
invalidate=0      # mandatory  
rw=randwrite  
bs=4k  
  
[rbd_iodepth32]  
iodepth=32
```

Preparatory step:

```
$ rbd -p rbd create fio_test —size 2048
```

Test tools for RBD (2)

- IOzone Filesystem Benchmark

```
$ iozone -r $BS -I -i 0 -i 1 -i 2 -t $THREADS -s $FILESIZE  
-i 0=write/rewrite, 1=read/re-read, 2=random-read/write
```

- Preparatory steps:

```
$ rbd -p rbd create iozone_test —size 2048  
$ rbd -p rbd map iozone_test  
/dev/rbd0  
$ mkfs.ext4 /dev/rbd0  
$ mount /dev/rbd0 /ceph-test
```

Cache flush & evict

- During tests we needed to manually clean the cache:
 1. `ceph osd tier cache-mode {cachepool} forward`
 2. `rados -p {cachepool} cache-flush-evict-all`
- Check the status with “`rados df`”
- When the cache is empty, restore the **writeback** cache mode

Ceph tutorial: cluster set-up

