swiftdocs Documentation

Release 0.1

timeredbull

September 20, 2012

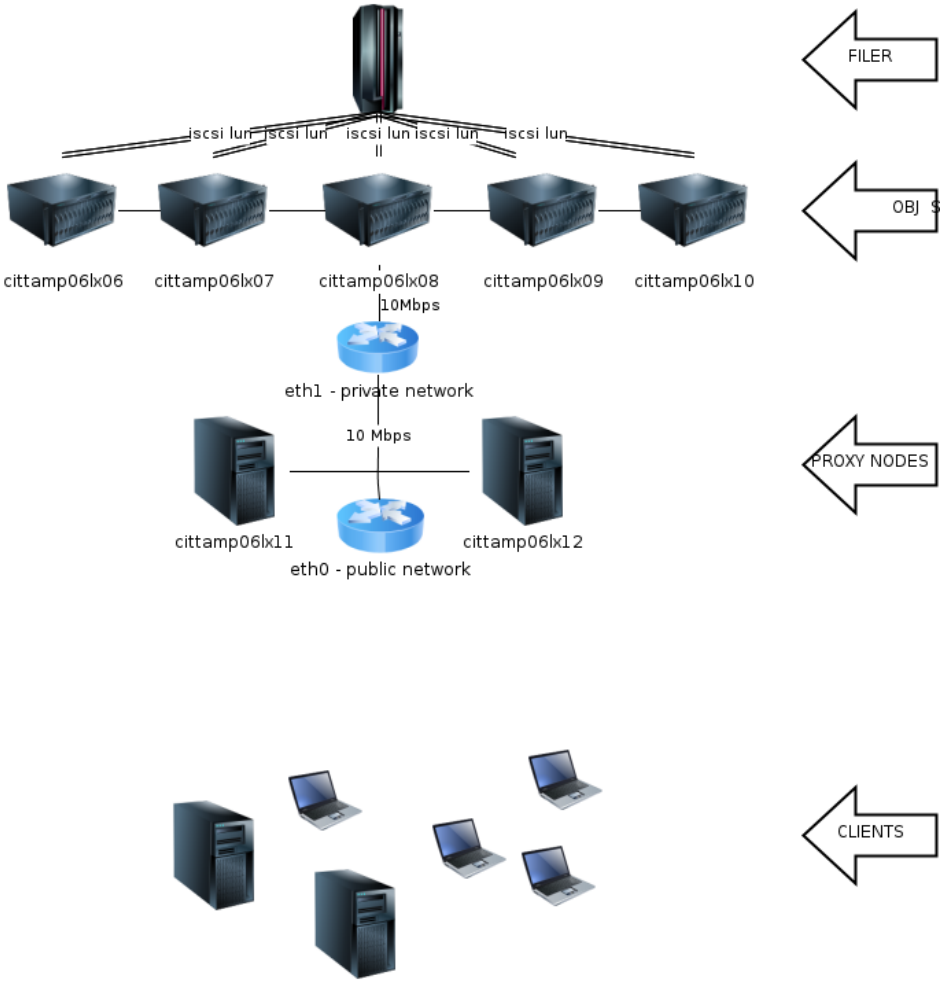
CONTENTS

Conteúdo:

CHAPTER

ONE

HARDWARE UTILIZADO



Object-writer nodes

- **6x ProLiant BL460c G6**
  - 2x 10Gbps Nic
  - 2x 300GB 10KRpm SAS HDD
  - 24GB RAM
  - 24 CPUs Xeon(R) L5640 @ 2.27GHz

- **6x 10GB iSCSI LUNs**
  - NetAPP Filer

 # Proxy nodes

- **2x ProLiant BL460c G6**
  - 2x 10Gbps Nic
  - 2x 300GB 10KRpm SAS HDD
  - 24GB RAM
  - 24 CPUs Xeon(R) L5640 @ 2.27GHz

No matter where you go, there you are.

—Buckaroo Banzai

CHAPTER

TWO

INSTALAÇÃO DO SWIFT

O Swift é um sistema de armazenamento distribuido e altamente redundante, projetado para atender uma única Região (links de baixa latência entre os servidores do Swift). O cluster do Swift conta com dois tipos de nó (nodes), os Swift Object Servers e os Swift Proxy Servers. A funcao dos primeiros é o armazenamento dos bancos de dados de configuração do Swift, metadados, contas de usuário e objetos propriamente ditos, enquanto os nós do segundo tipo prestam para o encaminhamento das requisições e serviço dos dados do cluster.

O Swift foi instalado segundo a documentação em Swift documentation. Os endpoints resultantes dessa instalação são:

```
https://swift.cumulus.dev.globoi.com/v1/AUTH_<tenant_id>/       (Interface de estáticos)
http://swift.cumulus.dev.globoi.com:8080/v1/AUTH_<tenant_id>/   (Interface Swift)
```

As áreas de dados do Swift são designadas segundo os tenentes do Keystone. Cada "tenant" tem uma área discriminada pelo seu próprio UUID segundo a fórmula acima. Essas áreas podem ser divididas em "containers" (buckets, na terminologia AWS), e utilizadas para armazenamento de backups e conteúdo estático.

2.1 Object Storage Nodes:

Os Storage Nodes são os reais trabalhadores da infraestrutra do Swift. É neles que os dados de usuário e de configuração sáo armazenados e é neles que é garantida a redundância e disponibilidade dos dados.

2.1.1 ARMAZENAMENTO

Visando dedicar todo o storage local para uso pelo Swift, os object nodes foram instalados com o root filesystem sediado em uma LUN iSCSI, servida, para a cloud de LAB, pelo Filer de desenvolvimento (riofd06). Essa LUN contendo a instalação inicial do nó foi clonado em outras 6 LUNs, uma para cada "Object Storage". As LUNs foram clonadas e mapeadas como a seguir, para os seus respectivos hosts:

LUN GOLDEN

```
- riofd06:/vol/vol342/swift_obj_root (LUN Golden)
```

LUNs CLONE - ROOTFS

```
- riofd06:/vol/vol342/cittamp06lx06_root
- riofd06:/vol/vol342/cittamp06lx07_root
- riofd06:/vol/vol342/cittamp06lx08_root
- riofd06:/vol/vol342/cittamp06lx09_root
- riofd06:/vol/vol342/cittamp06lx10_root
- riofd06:/vol/vol342/cittamp06lx11_root
```

As LUNs foram mapeadas para "igroups" (agrupamento de initiators iSCSI, no filer) contendo apenas o servidor "dono" da LUN e nomeado segundo o hostname do servidor:

```
cittamp06lx06 (iSCSI) (ostype: linux):
    iqn.1994-05.com.redhat:cittamp06lx06 (logged in on: vif1)
cittamp06lx07 (iSCSI) (ostype: linux):
    iqn.1994-05.com.redhat:cittamp06lx07 (logged in on: vif1)
cittamp06lx08 (iSCSI) (ostype: linux):
    iqn.1994-05.com.redhat:cittamp06lx08 (logged in on: vif1)
cittamp06lx09 (iSCSI) (ostype: linux):
    iqn.1994-05.com.redhat:cittamp06lx09 (logged in on: vif1)
cittamp06lx10 (iSCSI) (ostype: linux):
    iqn.1994-05.com.redhat:cittamp06lx10 (logged in on: vif1)
cittamp06lx11 (iSCSI) (ostype: linux):
    iqn.1994-05.com.redhat:cittamp06lx11 (logged in on: vif1)
```

Status final das LUNs após o mapeamento:

```
LUN path                              Mapped to          LUN ID  Protocol
----------------------------------------------------------------------------
/vol/vol342/cittamp06lx06_root      cittamp06lx06           0     iSCSI
/vol/vol342/cittamp06lx07_root      cittamp06lx07           0     iSCSI
/vol/vol342/cittamp06lx08_root      cittamp06lx08           0     iSCSI
/vol/vol342/cittamp06lx09_root      cittamp06lx09           0     iSCSI
/vol/vol342/cittamp06lx10_root      cittamp06lx10           0     iSCSI
```

Para garantir o boot sem maiores configurações recomenda-se que a LUN de boot seja a primeira LUN apresentada aos servidores (LUN0), conforme acima.

Nessa instalaçáo foi utilizado o "kickstart" (em riofb02a:/admfiler/2a/1/unix/tftpboot/pxelinux.cfg/centos6_64_SwiftWriter), para a instalação da LUN "golden", modelo para as demais. É basicamente uma instalação mínima de CentOS 6.3 x86_64, com os pacotes e dependências do Openstack Swift, na versão empacotada com a release Essex do Openstack (1.4.8). Nessa LUN estão incluidas as configurações para os 5 iniciais nós do cluster, número mínimo de hosts recomendado para produção). Quando for necessário adicionar-se mais nós no cluster, pode-se clonar a LUN GOLDEN em uma nova LUN, criar um novo iGroup contendo o novo servidor, e mapear a LUN para esse recém criado igroup. Para informar o Swift sobre o novo nó, siga os procedimentos descritos em *Procedimentos de Mudança:*

Os dois HDDs locais disponíveis nos servidores, foram integralmente particionados e formatados com o FS XFS, pela necessidade de Extended Attributes (XATTRS), conforme sugerido na documentação do Swift, e montados no mount point default de consumo do Swift, com os parâmetros recomendados na documentação, tanto para criação quanto para a montagem:

```
Criação:
mkfs.xfs -i size=1024 /dev/sda1
mkfs.xfs -i size=1024 /dev/sdb1

Montagem (/etc/fstab):
/dev/sda1 /srv/node/sda1 xfs noatime,nodiratime,nobarrier,logbufs=8 0 0
/dev/sdb1 /srv/node/sdb1 xfs noatime,nodiratime,nobarrier,logbufs=8 0 0
```

A disponibilidade dos dados é garantida por um esquema de réplicas, onde cada objeto é gravado em n-réplicas em zonas distintas, de modo a garantir a disponibilidade dos dados em caso de falha de até (n-replicas -1) zonas.

Cada servidor de objetos é uma "Zona" para o swift. As zonas são domínios de falha que devem ser configuradas do modo a ter-se tanta independência de recursos quanto possível (racks distintos, alimentação elétrica distinta, etc). O cluster deve ser disposto de modo a minimizar os efeitos de uma falha que afete mais de um nó do cluster ao mesmo tempo, sendo que a disponibilidade dos dados é garantida pelo número de réplicas configurado inicialmente,conforme já descrito.

As réplicas são feitas por intermédio do protocolo rSync. Cada servidor de objetos tem configurado em seu super-server (xinetd) um servidor de rsync que disponibliza os HDDs de cada zona para receber réplicas das demais zonas. O rsync foi configurado segundo sugerido na documentação do swift:

/etc/xinetd.d/rsyncd.conf

```
# default: off
# description: The rsync server is a good addition to an ftp server, as it \
#       allows crc checksumming etc.
service rsync
{
        disable = no
        flags           = IPv4
        socket_type     = stream
        wait            = no
        user            = root
        server          = /usr/bin/rsync
        server_args     = --daemon
        log_on_failure  += USERID
}
```

/etc/rsyncd.conf

```
uid = swift
gid = swift
log file = /var/log/rsyncd.log
pid file = /var/run/rsyncd.pid
address = 192.168.33.xx

[account]
max connections = 2                <- Valor sugerido pela documentação.
path = /srv/node/
read only = false
lock file = /var/lock/account.lock

[container]
max connections = 2
path = /srv/node/
read only = false
lock file = /var/lock/container.lock

[object]
max connections = 2
path = /srv/node/
read only = false
lock file = /var/lock/object.lock
```

Criacao dos rings:

Uma vez configurados os servidores (object, account e container), precisamos definir e informar ao Swift como particionar os discos, quantas réplicas fazer de cada objeto, etc. Essas configurações devem ser feitas com o utilitário "swift-ring-builder".

```
node-1$ cd /etc/swift
node-1$ swift-ring-builder account.builder   create <PARTITION_POWER> <REPLICAS> <MOVE_RESTRICTION_H>
node-1$ swift-ring-builder container.builder create <PARTITION_POWER> <REPLICAS> <MOVE_RESTRICTION_H>
node-1$ swift-ring-builder object.builder    create <PARTITION_POWER> <REPLICAS> <MOVE_RESTRICTION_H>
```

```
node-1$ swift-ring-builder object.builder    add z<ZONE>-<STORAGE_LOCAL_NET_IP>:6000/<DEVICE> <DISK_S
node-1$ swift-ring-builder container.builder add z<ZONE>-<STORAGE_LOCAL_NET_IP>:6001/<DEVICE> <DISK_S
node-1$ swift-ring-builder account.builder   add z<ZONE>-<STORAGE_LOCAL_NET_IP>:6002/<DEVICE> <DISK_S

node-1$ swift-ring-builder account.builder

PARTITION_POWER: 2^<PARTITION_POWER> = tamanho aproximado da partição.
REPLICAS: número de réplicas que cada objeto terá no cluster.
MOVE_RESTRICTION_H: número de horas a restringir que uma partição seja movida mais de uma vez.
ZONE: número sequencial da zona.
STORAGE_LOCAL_NET_IP: IP na rede de interconexão de baixa latência entres os nós.
DEVICE: label que identifica o disco na árvore de montagem '/srv/node'.
DISK_SIZE_GB: peso do disco, convencionado no número de GB do disco.
```

2.1.2 REDE

As interfaces de rede dos servidores foram configuradas como a seguir:

eth0 - Interface de acesso público (10.170.0.0/24 - DHCP)

eth1 - Interface de acesso privado - interconexão entre os Swift Object Servers e os Swift Proxy Servers (192.168.33.0/24 - Estatica em função do IP na eth0)

2.2 Proxy Nodes:

pacotes: openstack-swift-essex-proxy-essex-1.4.8-b3000, memcached-1.4.4-3.el6.x86_64

2.2.1 Descrição:

Os proxy-nodes são os responsáveis por receber as requisições clientes do Swift. Pode-se ter tantos proxy-nodes quantos necessários em função da demanda, balanceados por um VIP. Todo tráfego é HTTP/HTTPS.

2.2.2 Cacheamento automatico de estáticos:

Para fins de testes, os proxy-nodes implementados no LAB Cumulus, são balanceados por um Varnish, com cacheamento default em 120 segundos para _todos_os_objetos_ servidos, indiscriminadamente, pela interface de estáticos. Essa configuração visa amortecer quaisquer picos de acesso via interface de estáticos. Os acessos internos do Swift, via porta 8080, são apenas balanceados e nunca cacheados (pipe).

2.2.3 Cacheamento de metadados:

Para fins de cacheamento de meta-dados para uso interno, o Swift usa instâncias de "memcache" em cada um de seus nós proxy. Cada proxy deve ser configurado para "enxergar" os memcaches dos demais nós de modo a criar uma rede redundante de processos memcached.

CHAPTER

THREE

CONFIGURAÇÕES SWIFT

Cada cluster Swift deve ter um "Unique Identifier" (swift_hash_path_suffix), que o diferencie de outros clusters e que seja consistente entre os nós de cada cluster. Esse UUID deve ser armazenado no arquivo de configuração /etc/swift/swift.conf.

/etc/swift/swift.conf:

```
[swift-hash]
# random unique string that can never change (DO NOT LOSE)
swift_hash_path_suffix =  d9fa0ad2ded1f0db
```

OBS.: O hash acima pode ser facilmente regerado com o oneliner :

```
od -t x8 -N 8 -A n \< /dev/random
```

/etc/swift/{object-server.conf\container-server.conf\account-server.conf}:

```
[DEFAULT]
bind_ip = 192.168.33.26    <- Endereço privado de interconexão do cluster
workers = 24               <- Número de threads = número de CPUs do host
log_facility = LOG_LOCAL3  <- Facility padrão do lognit para Python

[pipeline:main]
pipeline = object-server <- (ou container-server, ou account-server)

[app:object-server]      <- (ou container-server, ou account-server)
use = egg:swift#object   <- (ou swift#container, ou swift#account)

[object-replicator]

[object-updater]

[object-auditor]
```

/etc/swift/proxy-server.conf:

/etc/sysconfig/memcached:

CHAPTER

FOUR

TUNNINGS DO SO

Visando privilegiar o throughput de I/O nos servidores na função de "object writers", alguns tunning foram feitos no ambiente a saber:

Hardware

Os servidores de objetos tiveram suas configurações de BIOS setadas para privilegiar I/O em detrimento de Acesso de memória:

```
Advanced Options – Advanced Performance Tunning Options – QPI Bandwidth Optimization(RTID) – Optimize
```

Esses tunnings são específicos para o Hardware utilizado na solução de POC no Laboratório - HP ProLiant BL460c G6. Outros tunnings, com as mesmas finalidades, devem ser prospectados nos novos modelos adquiridos para o Swift.

Software

Os tunnings abaixo foram aplicados para o SO :

/etc/sysctl.conf

O parametro tcp_syscookies foi mantido habilitado nos nós de proxy.

CHAPTER

FIVE

LOGGING:

Os processos do Swift enviam os logs, por default, para o rsyslog local. Para garantir a consolidação dos logs, os nós Swift Object Servers foram configurados para enviar seus logs para o lognit, como a seguir:

rsyslog.conf

CHAPTER

SIX

CHECKS DE POS-INSTALAÇÃO:

Além da verificação dos logs, minutos após a instalação, é necessária a verificação do correto funcionamento da API REST.

6.1 Validação

Para validar a correta instalação de um cluster de Swift, basta que, após configuradas as variáveis de ambiente de autenticação:

```
node-1$ export OS_USERNAME=login
node-1$ export OS_PASSWORD=senha
node-1$ export OS_AUTH_URL=http://keystone.dominio:5000/v2.0

node-1$ swift stat
   Account: AUTH_uuid_do_tenant
Containers: 0
   Objects: 0
     Bytes: 0
Accept-Ranges: bytes
X-Trans-Id: tx010d347fcf634e23925e17d0967f4ed0
```

6.2 Validação de upload/Criação do arquivo de healthcheck

Como uma primeira ação administrativa, cria-se, sob o tenant do adminstrador, um container com o nome "healthcheck", que será usado na monitoração do servico. Para que isso seja possível, além de criar o container precisaremos setar uma ACL básica que permita o accesso ao recurso pela monitoração:

```
node-1$ swift post -r '.r:\*'  healthcheck

node-1$ swift post -m 'web-listings: true' healthcheck

node-1$ echo OK > index.html

node-1$ swift upload healthcheck index.html

node-1$ curl -I http://swift.cumulus.dev.globoi.com:8080/v1/AUTH_uuid_do_tenant/healthcheck/index.htr
HTTP/1.1 200 OK
Last-Modified: Wed, 19 Sep 2012 20:10:50 GMT
Etag: d36f8f9425c4a8000ad9c4a97185aca5
X-Object-Meta-Mtime: 1348085441.15
```

```
Accept-Ranges: bytes
Content-Length: 3
Content-Type: text/html
X-Trans-Id: tx911b480689a645bcb097940cbb20e666
Date: Thu, 20 Sep 2012 12:07:45 GMT
```

CHAPTER

SEVEN

ADMINISTRAÇÃO DO SERVIÇO SWIFT

7.1 Monitoração do serviço:

Caso os *Checks de pos-instalação:* tenham sido observados, o Swift terá no mínimo os seguintes pontos de monitoraçao:

Interface principal da API REST

Todas os endpoints devem retornar 200 OK:

```
http://swift.cumulus.dev.globoi.com:8080/v1/AUTH_uuid_do_tenant/healthcheck/index.html

http://swift.cumulus.dev.globoi.com/v1/AUTH_uuid_do_tenant/healthcheck/index.html

http://swift.cumulus.dev.globoi.com/admin/healthcheck/index.html
```

7.2 Procedimentos de Mudança:

O Swift possui mecanismos de escalabilidade que permitem que sejam adicionados mais discos aos atuais servidores de objetos, ou mesmo mais servidores ao cluster atual. Esses procedimentos são bem simples, embora cautela seja recomendada durante sua condução para minimizar os impactos de performance que tais atividades impõem no cluster, a saber:

7.2.1 Adição de discos aos storage nodes

Para adicionar-se mais discos ou aumentar a capacidade dos discos nos nodes de storage, é preciso recriar os arquivos de configuração e rebalancear-se o cluster, tal e qual feito na instalação. Como a adição/remoção de zones implica na redistribuição de partições pelo cluster, é aconselhável que esse procedimento seja feito paulatinamente, através do progressivo aumento do peso (weight) do novo/antigo dispositivo.

O procedimento em si é análogo ao de criação do cluster, descrito em : *Criacao dos rings:*

Ex. Adição de um disco de 3TB. a uma zona

```
1. Adição de 25% dos 3000 = 750

   node-1$ mount /dev/sdc1 /srv/nodes/sdc1
   node-1$ swift-ring-builder account.builder add z1-node-1:6002/sdc1 750
   node-1$ swift-ring-builder container.builder add z1-node-1:6001/sdc1 750
   node-1$ swift-ring-builder object.builder add z1-node-1:6000/sdc1 750
   node-1$ swift-ring-builder account.builder rebalance
```

```
   node-1$ swift-ring-builder container.builder rebalance
   node-1$ swift-ring-builder object.builder rebalance

   node-1$ scp account.ring.gz node-2:/etc/swift/account.ring.gz
   node-1$ scp container.ring.gz node-2:/etc/swift/container.ring.gz
   node-1$ scp account.ring.gz node-2:/etc/swift/account.ring.gz
   node-1$ scp account.ring.gz node-3:/etc/swift/account.ring.gz
   node-1$ scp container.ring.gz node-3:/etc/swift/container.ring.gz
   node-1$ scp account.ring.gz node-3:/etc/swift/account.ring.gz
   ...
   node-1$ scp account.ring.gz node-N:/etc/swift/account.ring.gz
   node-1$ scp container.ring.gz node-N:/etc/swift/container.ring.gz
   node-1$ scp account.ring.gz node-N:/etc/swift/account.ring.gz

2. Aguarde até que o I/O no cluster esteja estabilizado, aumente em 25% o peso do disco, e
   repita o procedimento acima, até que o peso seja 100%, ou seja, os 3000 para um HDD de 3GB
```

Upgrade de discos do cluster

O aumento da capacidade do cluster pela substituição de HDDs pequenos e/ou lentos por outros maiores/mais rápidos, também deve ser feita de forma gradual de modo a não gerar um alto assédio de I/O no cluster. Recomenda-se que o disco antigo seja removido pela gradual redução de seu peso no cluster, em passos de 25% (sugeridos) até que o mesmo chegue a 0. Após a remoção física do disco antigo e instalação do novo disco, o procedimento de adição gradual já descrito acima, *Adição de discos aos storage nodes*, deve ser observada.

Adição de nodes ao cluster

A adição de um novo nó de storage ao cluster também se dá pela reconfiguração dos arquivos "*.builder" como explanado acima. Recomenda-se que _todos_ os discos do novo nó sejam adicionados gradualmente, de modo a minimizar o I/O internodes.

7.3 Procedimentos de Recuperação de desastres:

7.3.1 Falha de um ou mais Swift Object Servers ::

blah blah

7.3.2 Falha de um ou mais Swift Proxy Servers ::

blah blah

CHAPTER

EIGHT

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*