Продолжаю цикл статей про кластерные решения, который был начат с установки kubernetes. Я расскажу как установить и настроить кластер **ceph**, так же покажу, как им потом пользоваться. Статья с практическими примерами от и до — поднятие кластера и подключение дисков к конечным серверам.

Содержание:

Цели статьи

1. Рассказать своими словами о том, что такое Ceph, для чего нужен и какие есть аналоги.
2. Развернуть тестовый кластер ceph с помощью ansible-playbook.
3. Показать работу с кластером — подключение rbd дисков, запись данных в cephfs.
4. Проверить поведение кластера при отказе одного из узлов.

Что такое Ceph

Ceph — программная объектная отказоустойчивая сеть хранения данных. Для чего нужна Ceph? Она реализует возможность файлового и блочного доступа к данным. Оба этих варианта я рассмотрю в своей статье. Это бесплатное программное обеспечение, которое устанавливается на linux системы и может состоять из огромного количества узлов.

Для того, чтобы потестировать ceph, достаточно трех практически любых компьютеров или виртуальных машин. Моя тестовая лаборатория, на которой я буду писать статью, состоит из 4-х виртуальных машин с операционной системой Centos 7 со следующими характеристиками.

CPU	2
RAM	4G
DISK	/dev/sda 50G, /dev/sdb 50G

Это минимальная конфигурация для ceph, с которой стоит начинать тестирование. Из трех машин будет собран кластер ceph, а к четвертой я буду монтировать диски для проверки работоспособности.

Архитектура Ceph

Кластерная система хранения данных ceph состоит из нескольких демонов, каждый из которых обладает своей уникальной функциональностью. Расскажу о них кратко своими словами.

1. **MON**, Ceph monitor — монитор кластера, который отслеживает его состояние. Все узлы кластера сообщают мониторам информацию о своем состоянии. Когда вы монтируете хранилища кластера к целевым серверам, вы указываете адреса мониторов. Сами мониторы не хранят непосредственно данные.

2. **OSD**, Object Storage Device — элемент хранилища, который хранит сами данные и обрабатывает запросы клиентов. OSD являются основными демонами кластера, на которые ложится большая часть нагрузки. Данные в OSD хранятся в виде блоков.
3. **MDS**, Metadata Server Daemon — сервер метаданных. Он нужен для работы файловой системы CephFS. Если вы ее не используете, то MDS вам не нужен. К примеру, если кластер ceph предоставляет доступ к данным через блочное устройство RBD, сервер метаданных разворачивать нет необходимости. Разделение метаданных от данных значительно увеличивает производительность кластера. К примеру, для листинга директории нет необходимости дергать OSD. Данные берутся из MDS.
4. **MGR**, Manager Daemon — сервис мониторинга. До релиза Luminous был не обязательным компонентом, теперь — неотъемлемая часть кластера. Демон обеспечивает различный мониторинг кластера — от собственного дашборда до выгрузки метрик через json. Очень удобно. Мониторинг кластера не представляет особых сложностей.

Кластер ceph состоит из пулов для хранения данных. Каждый pool может обладать своими настройками. Пулы состоят из Placement Groups (PG), в которых хранятся объекты с данными, к которым обращаются клиенты.

В каждом кластере ceph имеется понятие фактора репликации — это уровень избыточности данных, или по простому — сколько копий данных будет храниться на разных дисках. Фактор репликации можно задавать разным для каждого пула и менять на лету.

Аналоги

Вообще говоря, Ceph достаточно уникальное кластерное решение, прямых аналогов которого нет. Но есть некоторые системы, схожие по решаемым проблемам. Основным аналогом Ceph является GlusterFS, которую я рассмотрю ниже отдельно. Так же к аналогам можно отнести следующие кластерные системы хранения данных:

- Файловая система ocfs2.
- OpenStack Swift.
- Sheepdog.
- HDFS (Hadoop Distributed File System)
- LeoFS

Список не полный. Это только то, что вспомнил я из того, что слышал. Сразу оговорюсь, что перечисленные системы мне практически не знакомы. Список привожу только для того, что, чтобы вам было проще потом самим найти о них информацию и сравнить. Они не являются полными аналогами ceph, но в каких-то вариантах могут подойти больше, нежели он. К примеру, Sheepdog намного проще система и потребности только в хранилище для виртуальных машин может закрывать лучше, чем ceph.

Теперь рассмотрим отдельно GlusterFS.

CephFS vs GlusterFS

Я не буду строить из себя эксперта и пытаться что-то объяснить в том, в чем не разбираюсь. На хабре есть очень подробная статья о сравнении Cephfs с GlusterFS от человека, который использовал обе системы. Если вам интересна тема подобного сравнения, то внимательно прочитайте. Я приведу краткие выводы, которые вынес сам из этой статьи.

1. Glusterfs менее стабильная и у нее больше критических багов, которые приводят к повреждению данных.
2. Ceph более гибкая и функциональная система.
3. В ceph можно добавлять диски любого размера и каждый будет иметь зависимый от размера вес. Данные будут размещаться по дискам почти равномерно. В glusterfs диски добавляются парами или тройками в зависимости от фактора репликации. В итоге в glusterfs не получится просто вытащить старые диски меньшего объема и поставить новые больше. Вам нужно будет менять диски сразу у всей группы, выводя ее из работы. В ceph такой проблемы нет, можно спокойно заменять старые диски на новые большего объема.
4. Архитектурно ceph быстрее и надежнее обрабатывает отказы дисков или серверов.
5. GlusterFS лучше масштабируется. Есть примеры огромных инсталляций.
6. В случае фактора репликации 2 у GlusterFS есть отличное решение с внешним арбитром.

Подводя итог статьи автор отмечает, что CephFS более сложное но и более функциональное решение. Я так понял, он отдает предпочтение ему.

Опыт использования

Своего опыта использования Ceph в production у меня нет. Я его хорошо протестировал и понял, что готов к тому, чтобы внедрять и использовать. Так что как и в предыдущем разделе поделюсь сторонними материалами на эту тему, которые изучал сам и они мне показались полезными.

Во-первых, очень интересная и популярная статья — Ceph. Анатомия катастрофы. Автор очень подробно рассматривает, как развивается ситуация в случае деградации кластера и дает советы по проектированию, чтобы восстановление прошло с наименьшими потерями. К слову, я тестировал на своем кластере отказ одной из трех нод и наблюдал примерно ту же картину, что описывает автор. Кластер начинает очень-очень сильно тормозить и привести его в чувство не так просто. После ввода в работу упавшей ноды, кластер вставал колом. Без подготовки и тренировки вся заявленная отказоустойчивость кластера ceph может вылететь в трубу и вы получите остановленный сервис. Там же в статье автор дает совет не отключать swap. А, к примеру, при установке Kubernetes, его наоборот нужно обязательно отключить.

Еще одна статья, которой хотел бы с вами поделиться на эту тему — А вот вы говорите Ceph... а так ли он хорош? Автор обращает внимание на огромное количество настроек ceph, большая часть из которых не понятно, что означает и за что отвечает :) Так же он делится своим опытом эксплуатации большого и нагруженного кластера. Некоторые вещи удивляют, и становится не понятно, как это ставить в production. Тем не менее люди ставят и это вроде как даже нормально работает. Но однозначно это путь смелых! В комментариях автор явно не ответил, но стало понятно, что его кластер не имеет бэкапа! Как вам такое? Сложнейшая распределенная система, которая может превратиться в тыкву и нет бэкапов. Админам таких систем нужно доплачивать за риск. Я бы не стал работать в такой должности. Мне спокойная жизнь дороже даже повышенной оплаты за обслуживание таких систем.

Как я понял из этих статей, нагруженный ceph требует очень внимательного мониторинга и обслуживания. Требуется постоянно дежурный инженер и готовые инструкции на наиболее популярные инциденты — замена дисков, отказ сервера, переполнение osd и т.д. Все это нужно заранее тестировать и документировать.

Подготовка настроек

Ну что же, с теоретической частью закончили, начинаем практику. Мы будем устанавливать ceph с помощью официального playbook для ansible. Клонируем к себе репозиторий.

```
# git clone https://github.com/ceph/ceph-ansible
```

Переключаемся на последнюю стабильную ветку 3.2.

```
# cd ceph-ansible
# git checkout stable-3.2
```

Проверяем зависимости в файле *requirements.txt*:

1. ansible~=2.6,<2.7
2. netaddr

Установим ansible и модуль питона netaddr через pip. А так же нужен будет модуль notario.

```
# pip install ansible==2.6.0.0 notario netaddr
```

И обновим еще пару модулей.

```
# pip install --upgrade chardet urllib3
```

Теперь готовим инвентарь для palybook. Создаем в корне репозитория папку *inventory*.

```
# mkdir inventory
```

В ней создаем файл *hosts* примерно следующего содержания.

```
[mons]
kub-ingress-1 monitor_address=10.1.4.39
kub-node-1 monitor_address=10.1.4.32
kub-node-2 monitor_address=10.1.4.33

[osds]
kub-node-1
kub-node-2
kub-ingress-1

[mgrs]
kub-node-1
kub-node-2
kub-ingress-1

[mdss]
kub-node-1
kub-node-2
kub-ingress-1
```

Для примера я взял 3 рабочие ноды из своей статьи про установку кластера Kubernetes. Я использую ту же тестовую лабу. По сути, это просто 3

виртуальных сервера с двумя жесткими дисками:

- sda под систему
- sdb под данные кластера ceph

В инвентаре несколько групп серверов:

1. mons — мониторинг
2. osds — демон хранения
3. mgrs — менеджер
4. mds — сервер метаданных

У меня все сервера равнозначные, поэтому на всех будет стоять все.

Дальше нам нужно указать некоторые параметры кластера. Делаем это в файле *all.yml* в директории *inventory/group_vars*.

```
ceph_origin: repository
ceph_repository: community
ceph_stable_release: luminous
public_network: "10.1.4.0/24"
cluster_network: "10.1.4.0/24"

ntp_service_enabled: true
ntp_daemon_type: ntpd

osd_objectstore: bluestore
osd_scenario: lvm
devices:
  - /dev/sdb

ceph_conf_overrides:
  global:
    osd_pool_default_pg_num: 64
```

```
    osd_pool_default_pgp_num: 64
    osd_journal_size: 5120
    osd_pool_default_size: 3
    osd_pool_default_min_size:  2
```

Небольшие пояснения к некоторым переменным:

- ceph_origin — откуда будет выполняться установка. В моем случае из репозитория.
- ceph_repository — название репозитория. В данном случае community, есть еще rhcs от redhat.
- ceph_stable_release — название последнего стабильного релиза.
- public_network — сеть, откуда будут приходить запросы в кластер.
- cluster_network — сеть для общения самого кластера. У меня небольшой тестовый кластер, поэтому сеть общая. Для больших кластеров cluster_network надо делать отдельной с хорошей пропускной способностью.
- osd_objectstore — тип хранения данных, bluestore — общая рекомендация для использования.
- osd_scenario — как будут храниться данные, в данном случае в lvm томах.
- devices — устройства, которые будет использовать ceph. Если их не указать, будут использованы все незадействованные диски.
- osd_pool_default_pg_num — кол-во placement groups. Стандартная формула для расчета этой штуки — (OSD * 100) / кол-во реплик. Результат должен быть округлён до ближайшей степени двойки. Если получилось 700, округляем до 512.
- osd_pool_default_pgp_num — настройка для размещения pg, служебная штука, рекомендуется ее выставлять такой же, как количество pg.
- osd_journal_size — размер журнала в мегабайтах. Я оставил дефолтное значение в 5 Гб, но если у вас маленький тестовый кластер можно уменьшить, или наоборот увеличить в больших кластерах.
- osd_pool_default_size — количество реплик данных в нашем кластере, 3 — минимально необходимое для отказоустойчивости.
- osd_pool_default_min_size — количество живых реплик, при которых пул еще работает. Если будет меньше, запись блокируется.

С инвентарем и параметрами разобрались. У нас почти все готово для установки ceph. Осталось создать файл site.yml в корневой папке репозитория, скопировав site.yml.sample.

```
# cp site.yml.sample site.yml
```

В файле оставляем все значения дефолтными.

Вычисление Placement Groups (PG)

Самая большая трудность в вычислении PG это необходимость соблюсти баланс между количеством групп на OSD и их размером. Чем больше PG на одной OSD, тем больше вам надо памяти для хранения информации об их расположении. А чем больше размер самой PG, тем больше данных будет перемещаться при балансировке.

Получается, что если у вас мало PG, они у вас большого размера, надо меньше памяти, но больше трафика уходит на репликацию. А если больше, то все наоборот. Теоретически считается, что для хранения 1 Тб данных в кластере надо 1 Гб оперативной памяти.

Как я уже кратко сказал выше, примерная формула расчета PG такая — Total PGs = (Number OSD * 100) / max_replication_count. Конкретно в моей установке по этой формуле получается цифра 100, которая округляется до 128. Но если задать такое количество pg, то роль ansible отработает с ошибкой:

```
Error ERANGE: pg_num 128 size 3 would mean 768 total pgs, which exceeds max 750 (mon_max_pg_per_osd 250 * num_in_osds 3)
```

Суть ошибки в том, что максимальное количество pg становится больше, чем возможно, исходя из параметра mon_max_pg_per_osd 250. То есть не более 250 на один OSD. Я не стал менять этот дефолтный параметр, а просто установил количество pg_num 64. Более подробная формула есть на официальном сайте — https://ceph.com/pgcalc/.

Существует проблема выделения pg и состоит она в том, что у нас в кластере обычно несколько пулов. Как распределить pg между ними? В общем случае поровну, но это не всегда эффективно, потому что в каждом пуле может храниться разное количество и типов данных. Поэтому для распределения pg по пулам стараются учитывать их размеры. Для этого тоже есть примерная формула — pg_num_pool = Total PGs * % of SizeofPool/TotalSize.

Количество PG можно изменять динамически. К примеру, если вы добавили новые OSD, то вы можете увеличить и количество PG в кластере. В последней версии ceph, которая еще не lts, появилась возможность уменьшения Placement Groups.

> Нужно понимать одну важную вещь — изменение количества PG приводит к ребалансингу всего кластера. Нужно быть уверенным, что он к этому готов.

Установка ceph

Запускаем установку Ceph с помощью playbook ansible.

```
# cd ~/ceph-ansible/
# ansible-playbook -u root -k -i inventory/hosts site.yml -b --diff
```

Если вы используете авторизацию по паролю, то скорее всего получите ошибку подключения к ssh. Я с этим столкнулся. Когда вывел расширенный лог ошибок плейбука, увидел, что ansible подключается по ssh с использованием публичного ключа, который я не задаю. Изменить это можно в конфиге ansible.cfg, который лежит в корне репозитория. Находим там строку:

```
ssh_args = -o ControlMaster=auto -o ControlPersist=600s -o PreferredAuthentications=publickey
```

и удаляем параметр для publickey, чтобы получилось вот так:

```
ssh_args = -o ControlMaster=auto -o ControlPersist=600s
```

После этого заново запускайте развертывание ceph. Оно должно пойти без ошибок. В конце увидите примерно такое сообщение.

Если будут какие-то ошибки и счетчик failed не будет равен нулю, то разбирайте ошибки, исправляйте их и запускайте роль заново, пока она не закончится без ошибок.

После установки Ceph, можно проверить статус кластера командой, которую нужно выполнить на одной из нод кластера.

```
# ceph status
```

```
mon: 3 daemons, quorum kub-node-1,kub-node-2,kub-ingress-1
```

Вы должны увидеть примерно то же самое, что на скриншоте выше. 3 монитора работают в кворуме, друг друга видят, все хорошо.

```
mgr: kub-node-2(active), standbys: kub-node-1, kub-ingress-1
```

Менеджеры работают в режиме активный и ждущие — 1 активен, остальные ожидают.

```
usage: 3.03GiB used, 147GiB / 150GiB avail
```

3 Гб данный ceph занял под свои нужды. Так же нужно учитывать, что доступное пространство в 150 Гб это общее пространство кластера, которое будет расходоваться в зависимости от заданной репликации. Записать 150 Гб данных туда не получится. Если мы запишем 1 Гб данных, они реплицируются на все ноды и будут реально занимать 3 Гб в кластере.

Кластер ceph установлен. Дальше разберем, как с ним работать.

Основные команды

Пройдемся по основным командам ceph, которые вам пригодятся при эксплуатации кластера. Основная — обзор состояния кластера:

```
# ceph -s
```

Традиционный ключ -w к команде для отслеживания изменений в реальном времени:

```
# ceph -w
```

Посмотреть список пулов в кластере:

```
# rados lspools
```

Статистика использования кластера:

```
# ceph df
```

Список всех ключей учетных записей кластера:

```
# ceph auth list
```

Состояние кворума:

```
# ceph quorum_status
```

Просмотр дерева OSD:

```
# ceph osd tree
```

Статистика OSD:

```
# ceph osd dump
```

Статистика PG:

```
# ceph pg stat
```

Список PG:

```
# ceph pg dump
```

Создание или удаление OSD:

```
# ceph osd create || ceph osd rm
```

Создание или удаление пула:

```
# ceph osd pool create || ceph osd pool delete
```

Тестирование производительности OSD:

```
# ceph tell osd.1 bench
```

Использование кластера ceph

Для начала давайте посмотрим, какие пулы у нас уже есть в кластере ceph.

```
# rados lspools
```

```
cephfs_data
cephfs_metadata
```

Это дефолтные пулы для работы cephfs, которые были созданы в момент установки кластера. Сейчас подробнее на этом остановимся. Ceph представляет для клиента различные варианты доступа к данным:

- файловая система **cephfs**;
- блочное устройство **rbd**;
- объектное хранилище с доступом через s3 совместимое api.

Я рассмотрю два принципиально разных варианта работы с хранилищем — в виде cephfs и rbd. Основное отличие в том, что cephfs позволяет монтировать один и тот же каталог с данными на чтение и запись множеству клиентов. RBD же подразумевает монопольный доступ к выделенному хранилищу. Начнем с cephfs. Для этого у нас уже все готово.

Подключение Cephfs

Как я уже сказал ранее, для работы cephfs у нас уже есть pool, который можно использовать для хранения данных. Я сейчас подключу его к одной из нод кластера, где у меня есть административный доступ к нему и создам в пуле отдельную директорию, которую мы потом смонтируем на другой сервер.

Монтируем pool.

```
# mount.ceph 10.1.4.32:/ /mnt/cephfs -o name=admin,secret=`ceph auth get-key client.admin`
```

В данном случае 10.1.4.32 адрес одного из мониторов. Их надо указывать все три, но сейчас я временно подключаю пул просто чтобы создать в нем каталог. Достаточно и одного монитора. Я использую команду:

```
# ceph auth get-key client.admin
```

для того, чтобы получить ключ пользователя admin. С помощью такой конструкции он нигде не засвечивается, а сразу передается команде mount. Проверим, что у нас получилось.

```
 # df -h | grep cephfs
10.1.4.32:/ 47G 0 47G 0% /mnt/cephfs
```

Смонтировали pool. Его размер получился 47 Гб. Напоминаю, что у нас в кластере 3 диска по 50 Гб, фактор репликации 3 и 3 гб заняты под служебные нужды. По факту у нас есть 47 Гб свободного места для использования в кластере ceph. Это место делится поровну между всеми пулами. К примеру, когда у нас появятся rbd диски, они будут делить этот размер вместе с cephfs.

Создаем в cephfs директорию data1, которую будем монтировать к другому серверу.

```
# mkdir /mnt/cephfs/data1
```

Теперь нам нужно создать пользователя для доступа к этой директории.

```
# ceph auth get-or-create client.data1 mon 'allow r' mds 'allow r, allow rw path=/data1' osd 'allow rw pool=cephfs_data'
```

```
[client.data1]
        key = AQBLRDBePhITJRAAFpGaJlGmqOj9RCXhMdIQ+w==
```

На выходе получите ключ от пользователя. Что я сделал в этой команде:

1. Создал клиента data1;
2. Выставил ему права к разным сущностям кластера (mon, mds, osd);
3. Дал права на запись в директорию data1 в cephfs.

Если забудете ключ доступа, посмотреть его можно с помощью команды:

```
# ceph auth get-key client.data1
```

Теперь идем на любой другой сервер в сети, который поддерживает работу с cephfs. Это практически все современные дистрибутивы linux. У них поддержка ceph в ядре. Монтируем каталог кластера ceph, указывая все 3 монитора.

```
# mount -t ceph 10.1.4.32,10.1.4.33,10.1.4.39:/ /mnt -o name=data1,secret='AQBLRDBePhITJRAAFpGaJlGmqOj9RCXhMdIQ+w=='
```

Проверяем, что получилось.

```
# df -h | grep mnt
10.1.4.32,10.1.4.33,10.1.4.39:/   47G     0   47G   0% /mnt
```

Каталог data1 на файловой системе cephfs подключен. Можете попробовать на него что-то записать. Этот же файл вы должны увидеть с любого другого клиента, к которому подключен этот же каталог.

Теперь настроим автомонтирование диска cephfs при старте системы. Для этого надо создать конфиг файл */etc/ceph/data1.secret* следующего содержания.

```
AQBLRDBePhITJRAAFpGaJlGmqOj9RCXhMdIQ+w==
```

Это просто ключ пользователя data1. Добавляем в */etc/fstab* подключение диска при загрузке.

```
10.1.4.32,10.1.4.33,10.1.4.39:/ /mnt ceph name=data1,secretfile=/etc/ceph/data1.secret,_netdev,noatime 0 0
```

Не забудьте в конце файла fstab сделать переход на новую строку, иначе сервер у вас не загрузится. Теперь проверим, все ли мы сделали правильно. Если у вас уже смонтирован диск, отмонтируйте его и попробуйте автоматически смонтировать на основе записи в fstab.

```
# umount /mnt
# mount -a
```

На этом по поводу cephfs все. Можно пользоваться. Переходим к блочным устройствам rbd.

Ceph RBD

Теперь давайте создадим rbd диск в кластере ceph и подключим его к целевому серверу. Для этого идем в консоль на любую ноду кластера. Создаем pool для rbd дисков.

```
# ceph osd pool create rbdpool 32
```

rbdpool	название пула, может быть любым
32	32 — кол-во pg (placement groups) в пуле

Проверим список пулов кластера.

```
# rados lspools
```

Создаем в этом пуле rbd диск на 10G.

```
# rbd create disk1 --size 10G --pool rbdpool
```

Добавим пользователя с разрешениями на использование этого пула. Делается точно так же, как в случае с cephfs, что мы проделали ранее.

```
# ceph auth get-or-create client.rbduser mon 'allow r, allow command "osd blacklist"' osd 'allow rwx pool=rbdpool'
```

В консоли увидите ключ пользователя rbduser для подключения пула rbdpool.

Перемещаемся на целевой сервер, куда мы будем подключать rbd диск кластера ceph. Для подключения blockdevice нам необходимо поставить программное обеспечение из репозитория ceph-luminous на сервер, где будет использоваться rbd.

```
# yum install centos-release-ceph-luminous
# yum install ceph-common
```

Так же запишем на целевой сервер ключ клиента, который имеет доступ к пулу с дисками. Создаем файл */etc/ceph/ceph.client.rbduser.keyring* следующего содержания.

```
[client.rbduser]
        key = AQCW5DFeXNy2JRAAWnCU/DpZhuNHmNcI5l1sEQ==
```

Там же создаем конфигурационный файл */etc/ceph/ceph.conf*, где нам необходимо указать ip адреса мониторов ceph.

```
mon host = 10.1.4.32,10.1.4.33,10.1.4.39
```

> В конце конфигурационного файла должен быть переход на новую строку. Если его не сделать, диск не смаппится, будет ошибка.

Пробуем подключить блочное устройство.

```
# rbd map disk1 --pool rbdpool --id rbduser
rbd: sysfs write failed
RBD image feature set mismatch. You can disable features unsupported by the kernel with "rbd feature disable rbdpool/disk1
object-map fast-diff deep-flatten".
In some cases useful info is found in syslog - try "dmesg | tail".
rbd: map failed: (6) No such device or address
```

Скорее всего получите такую же или подобную ошибку. Суть ее в том, что текущее ядро поддерживает не все возможности образа RBD, поэтому их нужно отключить. Как это сделать показано в подсказке. Для отключения нужен администраторский доступ в кластер. Так что идем на любую ноду пула и выполняем там предложенную команду.

```
# rbd feature disable rbdpool/disk1 object-map fast-diff deep-flatten
```

Не должно быть никаких ошибок, как и любого вывода после работы команды. Возвращаемся на целевой сервер и пробуем подключить rbd диск еще раз.

```
# rbd map disk1 --pool rbdpool --id rbduser
/dev/rbd0
```

Все в порядке. В системе появился новый диск — **/dev/rbd0**. Создадим на нем файловую систему и подмонтируем к серверу.

```
# mkfs.xfs -f /dev/rbd0
# mkdir /mnt/rbd
# mount /dev/rbd0 /mnt/rbd
# df -h | grep rbd
```

Можно попробовать туда что-то записать и посмотреть на скорость.

```
# dd if=/dev/zero of=/mnt/rbd/testfile bs=10240 count=100000
```

```
100000+0 records in
100000+0 records out
1024000000 bytes (1.0 GB) copied, 6.88704 s, 149 MB/s
```

Не знаю, что я измерил :) На самом деле это скорость одиночного sata диска, на котором установлена система сервера, которому я подключил диск. Так понимаю, запись вся ушла в буфер, а потом началась синхронизация по кластеру.

Настроим теперь автоматическое подключение rbd диска при старте системы. Для начала надо настроить mapping диска. Для этого создаем конфиг файл */etc/ceph/rbdmap* следующего содержания.

```
rbdpool/disk1           id=rbduser,keyring=/etc/ceph/ceph.client.rbduser.keyring
```

Запускаем скрипт rbdmap и добавляем в автозапуск.

```
# systemctl enable --now rbdmap
Created symlink from /etc/systemd/system/multi-user.target.wants/rbdmap.service to /usr/lib/systemd/system/rbdmap.service.
```

Проверяем статус.

```
# systemctl status rbdmap
```

Осталось добавить монтирование блочного устройства rbd в */etc/fstab*.

```
/dev/rbd/rbdpool/disk1 /mnt/rbd xfs noauto,noatime 0 0
```

И не забудьте в конце сделать переход на новую строку. После этого перезагрузите сервер и проверьте, что rbd диск кластера ceph нормально

подключается.

Проверка надежности и отказоустойчивости

Расскажу, какие проверки отказоустойчивости ceph делал я. Напомню, что у меня кластер состоит всего из трех нод, да еще на sata дисках на двух разных гипервизорах. Многого тут не натестируешь :) Диски собраны в raid1, никакой нагрузки помимо ceph на серверах не было. Я просто выключал одну ноду. При этом в работе кластера не было никаких заметных изменений. С ним можно было нормально работать, писать и читать данные. Самое интересное начиналось, когда я запускал обратно выключенную ноду.

В этот момент запускался ребалансинг кластера и он начинал жутко тормозить. Настолько жутко, что в эти моменты я даже не мог зайти на ноды по ssh или напрямую с консоли, чтобы посмотреть, что именно там тормозит. Виртуальные машины вставали колом. Я пытался их отключать и включать по очереди, но ничего не помогало. В итоге я выключил все 3 ноды и стал включать их по одной. Очевидно, что и нагрузки никакой я не давал, так как кластер был не в состоянии обслуживать внешние запросы.

Включил сначала одну ноду, убедился, что она загрузилась и показывает свой статус. Запустил вторую. Дождался, когда полностью синхронизируются две ноды, потом включил третью. Только после этого все вернулось в нормальное состояние. При этом никаких действий с кластером я не производил. Только следил за статусом. Он сам вернулся в рабочее состояние. Данные все оказались на месте. Меня это приятно удивило, с учетом того, что я жестко выключал зависшие виртуалки несколько раз.

Как я понял, если у вас есть возможность снять с кластера нагрузку, то в момент деградации особых проблем у вас не будет. Это актуально для кластеров с холодными данными, например, под бэкапы или другое долгосрочное хранение. Там можно тормознуть задачи и дождаться ребаланса. Особых проблем с эксплуатацией ceph быть не должно. А вот если у вас идет постоянная работа с кластером, то вам нужно все внимательно проектировать, изучать, планировать, тестировать и т.д. Точно должен быть еще один тестовый кластер и доскональное понимание того, что вы делаете.

Заключение

> Не понравилась статья и хочешь научить меня администрировать? Пожалуйста, я люблю учиться. Комментарии в твоем распоряжении. Расскажи, как сделать правильно!

Надеюсь, моя статья про описание, установку и эксплуатацию ceph была полезна. Постарался объяснять все простым языком для тех, кто как и я, только

начинает знакомство с ceph. Мне система очень понравилась именно тем, что ее можно так легко разворачивать и масштабировать. Берешь обычные серверы, раскатываешь ceph, ставишь фактор репликации 3 и не переживаешь за свои данные. Думаю, использовать его под бэкапы, docker registry или некритичное видеонаблюдение.

Переживать начинаешь, когда в кластер идет непрерывная высокая нагрузка. Но тут, как и в любых highload проектах, нет простых решений. Надо во все вникать, во всем разбираться и быть всегда на связи. Меня не привлекают такие перспективы :)