ceph-disk 源码分析

一、ceph-disk 简介

是 ceph 官方发布的一个用于部署 osd 数据以及 journal 分区或目录的工具。由于涉及数据盘以及 journal 盘的分区，所以需要 root 权限，默认被安装在/usr/sbin 目录下。

使用 ceph-disk 部署 osd，数据分区和 journal 分区将自动挂载到正确的目录，即使机器重启也能保证工作。通过 ceph-disk 来部署 osd，能大大提升 ceph 部署运维的自动化，并降低操作出错的几率。

创建 osd 的时候，分为两大步：ceph-disk prepare 和 ceph-disk activate，假设/dev/sdb 是 osd 要使用的数据盘，/dev/sda 是 ssd，osd 要使用的 journal 分区在这里创建，则创建 osd 并上线的命令如下 (这里所用的 ceph-disk 版本为 0.94.7)：

```
ceph-disk prepare --cluster ceph --fs-type xfs /dev/sdb /dev/sda1
ceph-disk activate --mark-init sysvinit /dev/sdb
```

一般在部署 osd 的时候，并没有直接调用 ceph-disk 命令，而是使用 ceph-deploy 或者 ansible 自动部署工具，在这些工具最终也是调用的 ceph-disk 来完成 osd 的部署的。一般在集群的 osd 出现故障需要将某个 ods 移除并重新加入集群的时候会直接用到 ceph-disk 命令。

在部署完 ceph 集群后，比较常用的 ceph-disk 参数有 list、activate、activate-all 和 zap。下面就从 ceph - disk 的源码入手，着重分析这些参数的执行步骤及流程，另外还附加介绍下系统重启时是如何自动挂载 osd 的数据盘并启动 osd 进程的。

二、ceph-disk list

`ceph-disk list` 命令的作用是列出系统中的磁盘，分区和所有的 ceph osd：

```
[root@ceph1 ~]# ceph-disk list
/dev/dm-0 other, xfs, mounted on /
/dev/dm-1 swap, swap
/dev/rbd0 other, xfs, mounted on /mnt
/dev/sda :
 /dev/sda2 other, LVM2_member
 /dev/sda1 other, xfs, mounted on /boot
```

```
/dev/sdb :
 /dev/sdb2 ceph journal, for /dev/sdb1
 /dev/sdb1 ceph data, active, cluster ceph, osd.0, journal /dev/sdb2
/dev/sdc :
 /dev/sdc2 ceph journal, for /dev/sdc1
 /dev/sdc1 ceph data, active, cluster ceph, osd.1, journal /dev/sdc2
```

在 ceph-disk 的源码中，执行 list 操作的函数是 `main_list`，流程如下：

1. 获取每块磁盘的分区信息

在 `main_list` 函数中，首先是获取`/sys/block` 下所有的块设备，然后再获取每个块设备的分区信息，比如以 `sda` 为例，找到 `/sys/block/sda` 目录下所有以 `sda` 开头的目录即可取得它的分区信息

```
for name in os.listdir('/sys/block'):
    ...
    dev_part_list[name] = list_partitions(name)

def list_partitions(basename):
    """
    Return a list of partitions on the given device name
    """
    partitions = []
    for name in os.listdir(os.path.join('/sys/block', basename)):
        if name.startswith(basename):
            partitions.append(name)
    return partitions
```

最后取得的分区信息保存在 `partmap` 这个字典中，以磁盘为 key，所有分区的组成的列表为值

```
{'sda': ['sda1', 'sda2'], 'sdb': ['sdb1', 'sdb2'], 'sdc': ['sdc1', 'sdc2']}
```

2. 得到每个分区的 `uuid`

即执行 `sgdisk -i` 命令获取每块磁盘所有分区的 `Partition unique GUID`

```
def get_partition_uuid(dev):
    (base, partnum) = split_dev_base_partnum(dev)
```

```
    out, _ = command(['sgdisk', '-i', partnum, base])
    for line in out.splitlines():
        m = re.match('Partition unique GUID: (\S+)', line)
        if m:
            return m.group(1).lower()
    return None
```

然后获取的 uuid 保存在以 uuid 为 key，分区号为值的字典 `uuid_map` 中

```
{'1cc9ba9e-d4d9-4225-b5a1-b31a8d53220b': '/dev/sdb1',
 'ddb026b7-3c52-401c-9ad1-07e29f40ba2b':'/dev/sda1',
 'd2608cea-d9e5-46ef-8292-1099a17d1e2e': '/dev/sdc1',
 'ca0eb69c-9717-4304-9a6c-30a3c1b673ca': '/dev/sdb2',
 '96fa9f69-0d8e-4a0a-8daf-56fb2b895fd4': '/dev/sda2',
 '7f9de3fa-c27c-46c8-800e-f988b7bcba0d': '/dev/sdc2'}
```

3. 获取 osd journal 的 uuid

得到分区的类型，使用命令 `blkid -p -o udev /dev/sdx` 获取该分区的 `ID_PART_ENTRY_TYPE` 值，得到该值后与 ceph-disk 中预定义的分区类型 UUID 进行比较，源码中预定义的分区类型 UUID 为：

```
...
OSD_UUID = '4fbd7e29-9d25-41b8-afd0-062c0ceff05d'
JOURNAL_UUID = '45b0969e-9b03-4f30-b4c6-b4b80ceff106'
...
```

如果分区类型为 OSD_UUID，接着得到文件系统类型，如果文件系统类型不为空，就尝试将该分区挂载到一个临时目录并读取该临时目录下的 `journal_uuid` 这个文件，这个文件中存储的是每个分区的 `Partition unique GUID`，然后将这个 osd 的 journal_uuid 在 journal_map 中

```
if ptype == OSD_UUID:
    fs_type = get_dev_fs(dev)
    if fs_type is not None:
        try:
            tpath = mount(dev=dev, fstype=fs_type, options='')
            try:
                journal_uuid = get_oneliner(tpath, 'journal_uuid')
                if journal_uuid:
                    journal_map[journal_uuid.lower()] = dev
```

```
            finally:
                unmount(tpath)
        except MountError:
            pass
```

4. 输出分区信息

根据该分区是常规的文件系统，osd 分区，journal 分区输出相应的信息

```
for base, parts in sorted(partmap.iteritems()):
    if parts:
        print '%s :' % get_dev_path(base)
        for p in sorted(parts):
            list_dev(get_dev_path(p), uuid_map, journal_map)
    else:
        list_dev(get_dev_path(base), uuid_map, journal_map)
```

三、ceph-disk prepare

`ceph-disk prepare` 的作用是准备 osd 的 data 分区和 journal 分区，根据配置文件中设置的文件系统属性新建分区和文件系统，并在 data 分区中写入集群的信息，下面以前面的 `ceph-disk prepare --cluster ceph --fs-type xfs /dev/sdb /dev/sda1` 为例来说明 prepare 操作的具体流程：

1. 检测 data 和 journal 分区，得到所需的配置参数

首先是检测 data 和 journal 分区是否是块设备以及是否已经被挂载

```
dmode = os.stat(args.data).st_mode
if stat.S_ISBLK(dmode):
    verify_not_in_use(args.data, True)

...
if args.journal and os.path.exists(args.journal):
    jmode = os.stat(args.journal).st_mode
    if stat.S_ISBLK(jmode):
        verify_not_in_use(args.journal, False)
```

然后获取需要的配置参数，这里的配置参数可以从命令行中手动指定，也可以从配置文件中获取。比如--

`fs_type` 这个配置是在命令行上指定的，那么就会优先使用这个参数，如果没有指定，才从配置文件中获取。在这一步中，获取的配置主要有：

- fsid: 集群的 UUID
- osd_mkfs_type: osd 分区所用的文件系统类型
- osd_mkfs_options_{fstype}: 制作文件系统时所用的属性，这里的 fstype 根据命令行传入的`--fs_type` 的参数或者从配置文件中获取的 `osd_fs_type` 来选择相应的文件系统类型
- osd_mount_options_{fstype}：挂载 data 分区时所用的属性，一般要在配置文件中指定
- osd_journal_size: 指定 journal 的大小，默认是 5G

上述的这些配置，如果在命令行或者配置文件中都没有指定的话，就获取 ceph 的默认配置，比如 `osd_journal_size`：

```
journal_size = get_conf_with_default(
    cluster=args.cluster,
    variable='osd_journal_size',
)
...
```

`get_conf_with_default` 这个函数会通过命令 `ceph-osd --cluster ceph --show-config-value=osd_journal_size` 来获取 `journal_size` 的值。

2. 创建 journal 文件或分区

journal 可以使用一个文件或者一个分区

如果 journal 是一个文件，则调用 `prepare_journal_file` 来创建 journal 文件：

```
if not os.path.exists(journal):
    LOG.debug('Creating journal file %s with size 0 (ceph-osd will resize and allocate)',
    with file(journal, 'wb') as journal_file:  # noqa
        pass
```

如果 journal 是块设备，则调用 `prepare_journal_dev` 来创建 journal 分区，在 `prepare_journal_dev` 函数中，主要是用 `sgdisk` 命令创建 journal 分区：

```
...
command_check_call(
    [
        'sgdisk',
        '--new={part}'.format(part=journal_part),
        '--change-name={num}:ceph journal'.format(num=num),
        '--partition-guid={num}:{journal_uuid}'.format(
            num=num,
            journal_uuid=journal_uuid,
            ),
        '--typecode={num}:{uuid}'.format(
            num=num,
            uuid=ptype_tobe,
            ),
        '--mbrtogpt',
        '--',
        journal,
        ]
    )
...
```

3. 创建 osd 数据的目录或分区

如果指定存放 osd 数据的是目录，则调用 `prepare_dir` 函数创建 osd 数据目录以及一些标示文件，如 `ceph_fsid`,`magic`，`whoami` 等

如果指定存放 osd 数据的是块设备，则调用 `prepare_dev` 函数创建一个新分区并修改改分区的卷标，分区 uuid 等信息：

```
command_check_call(
    [
        'sgdisk',
        '--largest-new=1',
        '--change-name=1:ceph data',
        '--partition-guid=1:{osd_uuid}'.format(
            osd_uuid=osd_uuid,
            ),
        '--typecode=1:%s' % ptype_tobe,
        '--',
        data,
    ],
)
```

四、ceph-disk activate

`ceph-disk activate` 的作用是激活分区，挂载 osd 数据分区并且启动 osd 服务

1. 激活分区

首先根据配置或命令行参数获取挂载分区的配置参数，在 `ceph-disk list` 一节已经介绍过了，然后调用 `mount` 函数将分区挂载到一个临时的目录中，返回该临时目录，然后调用 `activate` 函数写入一些标示文件

```
...
path = mount(dev=dev, fstype=fstype, options=mount_options)

osd_id = None
cluster = None
try:
    (osd_id, cluster) = activate(path, activate_key_template, init)
...
```

比如，在部署 osd 的时候，调用 `ceph-disk activate` 时会加入`--mark-init sysvinit` 或者`--mark-init systemd`，则在 activate 函数中就会写入以 sysvinit 或者 systemd 为名称的一个空文件

2. 挂载 osd 数据分区

先 mount 数据分区到 osd 的数据目录，然后再将之前挂载的临时目录卸载掉

```
command_check_call(
    [
        '/bin/mount',
        '-o',
        mount_options,
        '--',
        dev,
        osd_data,
        ],
    )
command_check_call(
    [
        '/bin/umount',
        '-l',   # lazy, in case someone else is peeking at the
```

```
                # wrong moment
        '--',
        path,
        ],
    )
```

3. 启动 osd 服务

如果在命令行参数中没有指定--mark-init 参数的话，就调用 ceph-osd 命令来启动 osd 服务进程，有的话就调用 start_daemon，在 start_daemon 函数中，根据系统的 init 类型使用不同的命令来启动服务和允许开机启动，比如 init 类型为 systemd 的话，就调用 systemctl 命令：

```
...
elif os.path.exists(os.path.join(path, 'systemd')):
    command_check_call(
        [
            'systemctl',
            'enable',
            'ceph-osd@{osd_id}'.format(osd_id=osd_id),
        ],
    )
    command_check_call(
        [
            'systemctl',
            'start',
            'ceph-osd@{osd_id}'.format(osd_id=osd_id),
        ],
    )
...
```

osd 数据盘自动挂载

在刚部署 osd 的时候，就有这么个疑问，系统是如何在开机启动时自动将 osd 的数据盘挂载的，经过一番探索之后，终于发现玄机了。原来系统是通过 udev 规则来实现启动时挂载数据盘的。这个 udev 规则在为 95-ceph-osd.rules，存在于 ceph 的 rpm 包中，安装时拷贝到系统的/usr/lib/udev/rules.d/目录下，其中的部分规则如下：

```
...
# activate ceph-tagged partitions
```

```
ACTION=="add", SUBSYSTEM=="block", \
  ENV{DEVTYPE}=="partition", \
  ENV{ID_PART_ENTRY_TYPE}=="4fbd7e29-9d25-41b8-afd0-062c0ceff05d", \
  RUN+="/usr/sbin/ceph-disk-activate /dev/$name"

# activate ceph-tagged partitions
ACTION=="add", SUBSYSTEM=="block", \
  ENV{DEVTYPE}=="partition", \
  ENV{ID_PART_ENTRY_TYPE}=="45b0969e-9b03-4f30-b4c6-b4b80ceff106", \
  RUN+="/usr/sbin/ceph-disk activate-journal /dev/$name"
...
```

这些规则在系统启动的时候会被执行，可以看到在 `RUN+=` 那行里执行了 `ceph-disk activate` 操作，这样就把分区类型的 UUID 为 `OSD_UUID` 的分区挂载到相应的目录并启动了 osd 服务