Docker存储模式与命令汇总

文档说明：

- OS与Docker版本：**CentOS 7.2**、***Docker 1.13.1***

配置**Docker**日志驱动程序：**3**种方法

- 方法1：

```
# vim /usr/lib/systemd/system/docker.service
  ExecStart= ... --log-driver=<log_driver> ...
// 配置docker守护进程systemd的unit单元文件，该配置对新运行的所有容器生效。
# systemctl daemon-reload
// 重新加载systemd的unit单元文件
# systemctl restart docker.service
// 重启docker守护进程使配置生效
```

- 方法2：

```
# vim /etc/sysconfig/docker
  OPTIONS='... --log-driver=<log_driver> ...'
// 更改Docker主配置文件，配置日志驱动程序选项，该配置对新运行的所有容器生效。
# systemctl restart docker.service
// 重启docker守护进程使配置生效
```

- 方法3：

```
# docker run ... --log-driver=<log_driver> ...
// 运行容器时指定日志驱动程序类型，但为一次性临时命令，退出容器或重启后将失效。
```

```
[root@cloud-ctl ~]# docker run -d --name base-centos ocp-registry.domain12.example.com:5000/centos:6.8 /bin/bash -c 'while true; do echo hello world;
 sleep 1; done'
b9aa9d513e3dd94cf23686ec1923b6024e64cf2f2e1883410b128f4a421ade03
[root@cloud-ctl ~]# docker inspect -f '{{ .HostConfig.LogConfig.Type }}' base-centos
journald     # docker守护进程默认日志驱动
[root@cloud-ctl ~]# tail -n 3 /var/log/messages      # 查看容器的日志驱动为journald，可在Docker宿主机日志文件/var/log/messages中查看容器日志。
Aug 21 16:53:05 cloud-ctl journal: hello world
Aug 21 16:53:06 cloud-ctl journal: hello world
Aug 21 16:53:07 cloud-ctl journal: hello world
[root@cloud-ctl ~]# docker logs --tail 3 base-centos
hello world
hello world
hello world
[root@cloud-ctl ~]# docker rm --force base-centos     # 强制删除容器，否则创建相同名称容器时将报错。
base-centos
[root@cloud-ctl ~]# docker run -d --name base-centos --log-driver=json-file ocp-registry.domain12.example.com:5000/centos:6.8 /bin/bash -c 'while tru
e; do echo hello world; sleep 1; done'
fbc2d2a65a4b172b8ba5297d4468b0f192b46939ba91371bbdf0ad1caa120e3c      # 运行容器时指定日志驱动为json-file
[root@cloud-ctl ~]# docker inspect -f '{{ .HostConfig.LogConfig.Type }}' base-centos
json-file
[root@cloud-ctl ~]# tail -n 3 /var/log/messages
Aug 21 16:54:06 cloud-ctl kernel: docker0: port 1(vethf9916a2) entered forwarding state
Aug 21 16:54:07 cloud-ctl avahi-daemon[839]: Registering new address record for fe80::1cc5:82ff:fee5:a1a2 on vethf9916a2.*.
Aug 21 16:54:07 cloud-ctl avahi-daemon[839]: Registering new address record for fe80::1cc5:82ff:fee5:a1a2 on vethf9916a2.*.
[root@cloud-ctl ~]# docker logs --tail 3 base-centos
hello world
hello world
hello world
[root@cloud-ctl ~]# tail -n 3 /var/lib/docker/containers/fbc2d2a65a4b172b8ba5297d4468b0f192b46939ba91371bbdf0ad1caa120e3c/fbc2d2a65a4b172b8ba5297d446
8b0f192b46939ba91371bbdf0ad1caa120e3c-json.log
{"log":"hello world\n","stream":"stdout","time":"2018-08-21T08:56:45.492046275Z"}  # 使用json-file日志驱动时，容器日志以JSON格式位
{"log":"hello world\n","stream":"stdout","time":"2018-08-21T08:56:46.495333653Z"}  于/var/lib/docker/containers/<CONTAINER_ID>/<CONTAINER_ID>-json.log中。
{"log":"hello world\n","stream":"stdout","time":"2018-08-21T08:56:47.498106299Z"}
[root@cloud-ctl ~]#                                                               # docker守护进程的根目录可自定义
```

```
[root@cloud-ctl ~]# docker run -d --name base-centos --log-driver=syslog ocp-registry.domain12.example.com:5000/centos:6.8 /bin/bash -c 'while true;
do echo hello world; sleep 1; done'
d13687d7204d665e2acd9d2bcd600c42a0a3fa8138366e2494d80c2ec4263543      # 运行容器时指定日志驱动为syslog
[root@cloud-ctl ~]# docker inspect -f '{{ .HostConfig.LogConfig.Type }}' base-centos
syslog
[root@cloud-ctl ~]# tail -n 3 /var/log/messages
Aug 21 17:00:21 cloud-ctl d13687d7204d[1596]: hello world  # 使用syslog日志驱动时，容器日志位于/var/log/messages中。
Aug 21 17:00:22 cloud-ctl d13687d7204d[1596]: hello world
Aug 21 17:00:22 cloud-ctl d13687d7204d[1596]: hello world  # CONTAINER_ID与日志信息
[root@cloud-ctl ~]# docker logs --tail 3 base-centos
"logs" command is supported only for "json-file" and "journald" logging drivers (got: syslog) # 只有日志驱动为json-file与journald时，才能使用docker logs命令。
[root@cloud-ctl ~]#
```

```
[student@workstation ~]$ docker logs mysql-db
You must either specify the following environment variables:
  MYSQL_USER (regex: '^[a-zA-Z0-9_]+$')
  MYSQL_PASSWORD (regex: '^[a-zA-Z0-9_~!@#$%^&*()-=<>,.?;:|]+$')
  MYSQL_DATABASE (regex: '^[a-zA-Z0-9_]+$')
Or the following environment variable:
  MYSQL_ROOT_PASSWORD (regex: '^[a-zA-Z0-9_~!@#$%^&*()-=<>,.?;:|]+$')
Or both.
Optional Settings:
  MYSQL_LOWER_CASE_TABLE_NAMES (default: 0)
  MYSQL_LOG_QUERIES_ENABLED (default: 0)
  MYSQL_MAX_CONNECTIONS (default: 151)
  MYSQL_FT_MIN_WORD_LEN (default: 4)
  MYSQL_FT_MAX_WORD_LEN (default: 20)
  MYSQL_AIO (default: 1)
  MYSQL_KEY_BUFFER_SIZE (default: 32M or 10% of available memory)
  MYSQL_MAX_ALLOWED_PACKET (default: 200M)
  MYSQL_TABLE_OPEN_CACHE (default: 400)
  MYSQL_SORT_BUFFER_SIZE (default: 256K)
  MYSQL_READ_BUFFER_SIZE (default: 8M or 5% of available memory)
  MYSQL_INNODB_BUFFER_POOL_SIZE (default: 32M or 50% of available memory)
  MYSQL_INNODB_LOG_FILE_SIZE (default: 8M or 15% of available memory)
  MYSQL_INNODB_LOG_BUFFER_SIZE (default: 8M or 15% of available memory)
For more information, see https://github.com/sclorg/mysql-container
[student@workstation ~]$
```

查看**docker**守护进程的存储驱动：2种方法

- `# lsmod | grep dm_mod`

  `// 查看devicemapper存储驱动的内核模块是否加载`

- 方法1：`# ls -l /sys/class/misc/device-mapper`
- 方法2：`# grep 'device-mapper' /proc/devices`

配置**Docker devicemapper**存储驱动：

- 虚拟机的快照实际上是虚拟磁盘的快照，要实现快照功能，虚拟磁盘必须支持写时复制机制。
- qcow2格式的虚拟磁盘支持写时复制技术。
- 写时复制（copy-on-write，**CoW**）机制：

1. 针对于修改已存在文件的场景
2. 即当文件需要进行修改时，再对该文件进行复制，而不是对虚拟机磁盘镜像或容器镜像
的整体复制，增加磁盘性能与效率。
3. 用例：
若不同容器使用相同的容器镜像，在不同容器中需修改同一个文件，devicemapper存储
驱动将在不同容器的**顶层可读写层**中复制该文件的副本，再分别对该文件的副本进行修改，
修改的内容通过**用时分配机制**（allocate-on-demand）获得新的块并写入数据。
因此在不同容器中的相同文件可**相互隔离互不影响**，修改的内容保存于容器的顶层可读
写层中。

- 用时分配（allocate-on-demand）机制：

1. 针对于创建新文件或文件发生修改的场景
2. 即在文件创建或修改前不分配新的块，而在文件创建或修改后，分配新的块并写入数据。

- Docker devicemapper存储驱动基本概念：

1. Docker graphdriver驱动（存储驱动）包括：
a. aufs：Ubuntu文件级存储
b. overlay *or* ***overlay2***：文件级存储
c. ***devicemapper***：块级存储
d. btrfs、zfs
2. RHEL与CentOS中可使用 overlay/**overlay2** 与 **devicemapper** 作为存储驱动，并且两者
都已纳入Linux内核主线（mainline）。
3. RHEL与CentOS中默认使用 overlay/overlay2 存储驱动。

```
[root@cloud-ctl ~]# df -Th
Filesystem                     Type      Size  Used Avail Use% Mounted on     # 在未运行容器的情况下，不使用默认存储驱动overlay2。
/dev/mapper/rootvg-lv0         xfs        28G   19G  9.1G  68% /
devtmpfs                       devtmpfs  7.8G     0  7.8G   0% /dev
tmpfs                          tmpfs     7.8G  144K  7.8G   1% /dev/shm
tmpfs                          tmpfs     7.8G  9.2M  7.8G   1% /run
tmpfs                          tmpfs     7.8G     0  7.8G   0% /sys/fs/cgroup
/dev/sr0                       iso9660   4.1G  4.1G     0 100% /mnt/centos7.2-dvd
/dev/sda1                      xfs       497M  214M  284M  43% /boot
/dev/sdb2                      xfs        30G   11G   20G  35% /mnt/virtual-imgs
/dev/mapper/isovg-lviso        xfs        20G  8.1G   12G  41% /mnt/iso-backup
/dev/loop0                     iso9660   4.3G  4.3G     0 100% /var/www/html/centos7.4-dvd
/dev/loop1                     iso9660   3.8G  3.8G     0 100% /var/www/html/rhel7.4-dvd
/dev/mapper/dockervg-lvregistry xfs       20G  252M   20G   2% /mnt/docker-registry
tmpfs                          tmpfs     1.6G   32K  1.6G   1% /run/user/0
/dev/loop2                     iso9660   6.7G  6.7G     0 100% /var/www/html/suse15-dvd
[root@cloud-ctl ~]# docker info | grep Storage
  WARNING: You're not using the default seccomp profile
Storage Driver: overlay2       # docker守护进程存储驱动为overlay2
[root@cloud-ctl ~]# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
[root@cloud-ctl ~]# docker ps -a
CONTAINER ID        IMAGE                                                COMMAND                  CREATED             STATUS                      POR
TS                  NAMES
85c2755bea37        kube-registry.domain12.example.com:5000/centos:6.8   "/bin/bash -c 'whi..."   16 hours ago        Exited (137) 14 hours ago
                    base-centos
[root@cloud-ctl ~]# docker start base-centos     # 启动docker容器，同时将使用overlay2存储驱动加载容器镜像。
base-centos
[root@cloud-ctl ~]# df -Th | grep overlay        # 查看overlay2存储驱动的文件系统使用情况
overlay                        overlay    28G   19G  9.1G  68% /var/lib/docker/overlay2/2fb6af3a6e1147a7a2d94604c588ca4c09b75ab75220d9bdbd58a34f9a11
1791/merged
[root@cloud-ctl ~]#
```

4. devicemapper存储驱动为**块级别的存储**，直接对块进行读写操作，提高磁盘利用率与性能，
适用于I/O密集的场景。

- **Docker devicemapper存储驱动原理：**

1. devicemapper存储驱动使用LVM的**精简设置**（thin-provisioning）与**快照技术**。

2. LVM的精简配置创建的两个LV块设备分别用于存储数据与元数据，两者共同组成

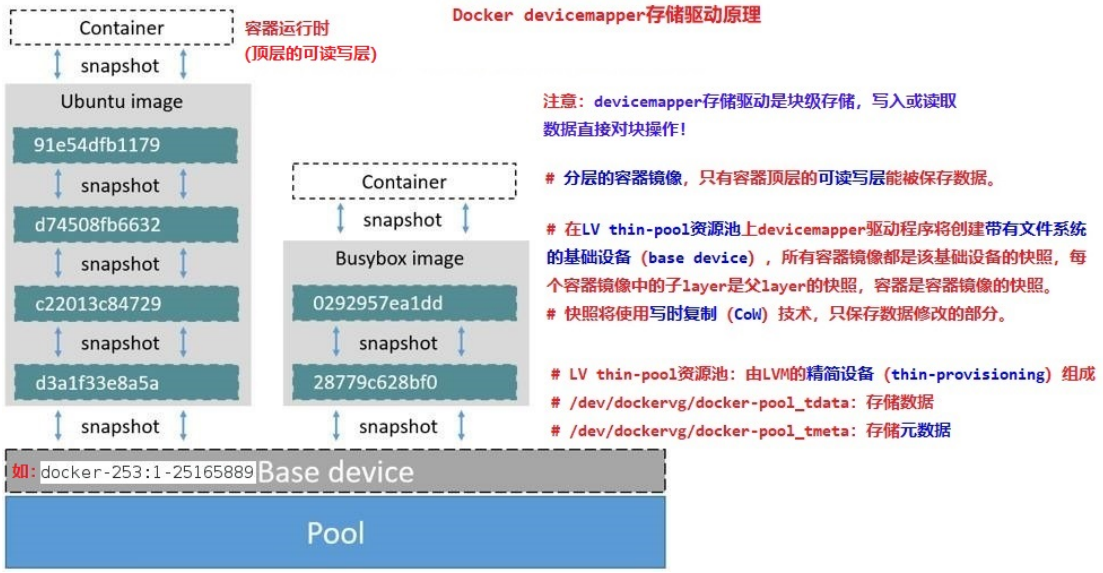
存储资源池（pool）。

3. 存储资源池之上devicemapper驱动程序创建带有文件系统的基础设备（base device）。

4. 容器镜像作为该基础设备的快照，容器镜像内的各层layer，其子layer为父layer的快照，

使用CoW机制实现。

5. 运行的容器为容器镜像的快照，所有的修改都发生在容器镜像的顶层可读写层中。




```
sdc                                   8:32   0   40G  0 disk
└─sdc1                                8:33   0   10G  0 part
  └─rootvg-lv0                      253:0    0 27.5G  0 lvm  /
sdd                                   8:48   0   20G  0 disk
└─sdd1                                8:49   0   20G  0 part
  ├─dockervg-docker--pool_tmeta     253:3    0   24M  0 lvm   # LV thin-pool资源池存储元数据
  │ └─dockervg-docker--pool         253:6    0    8G  0 lvm
  │   ├─docker-253:1-25165889-735cd040042fd601e2913da845c39d829661d757f0301909c1bbdcd049095f43   # 基础设备中对应的相应容器的基础设备
                                    253:7    0   10G  0 dm   /var/lib/docker.bak/devicemapper/mnt/735cd040042fd601e2913da845c39d829661d757f0301909c1bbdcd
  │   ├─docker-253:1-25165889-7386a79a6900056893ee81096f920843c41a0eb60001d4e5acf1cd29b0b0f10e
                                    253:8    0   10G  0 dm   /var/lib/docker.bak/devicemapper/mnt/7386a79a6900056893ee81096f920843c41a0eb60001d4e5acf1cd2
  │   └─docker-253:1-25165889-e7758da5fd9e2078a1306658cd43de7e5829b5bcce373f5eb18543a3eed3166d
                                    253:9    0   10G  0 dm   /var/lib/docker.bak/devicemapper/mnt/e7758da5fd9e2078a1306658cd43de7e5829b5bcce373f5eb18543a
  └─dockervg-docker--pool_tdata     253:4    0    8G  0 lvm   # LV thin-pool资源池存储数据                    # 容器的基础设备在Docker根目录中的设备挂载目录，容器的rootfs为该基础设备rootfs的快照。
    └─dockervg-docker--pool         253:6    0    8G  0 lvm
      ├─docker-253:1-25165889-735cd040042fd601e2913da845c39d829661d757f0301909c1bbdcd049095f43
                                    253:7    0   10G  0 dm   /var/lib/docker.bak/devicemapper/mnt/735cd040042fd601e2913da845c39d829661d757f0301909c1bbdcd
      ├─docker-253:1-25165889-7386a79a6900056893ee81096f920843c41a0eb60001d4e5acf1cd29b0b0f10e
                                    253:8    0   10G  0 dm   /var/lib/docker.bak/devicemapper/mnt/7386a79a6900056893ee81096f920843c41a0eb60001d4e5acf1cd2
      └─docker-253:1-25165889-e7758da5fd9e2078a1306658cd43de7e5829b5bcce373f5eb18543a3eed3166d
                                    253:9    0   10G  0 dm   /var/lib/docker.bak/devicemapper/mnt/e7758da5fd9e2078a1306658cd43de7e5829b5bcce373f5eb18543a
```

```
[root@kube-master docker.bak]# ls -l      # 查看使用devicemapper存储驱动的Docker根目录结构
total 4
drwx------ 7 root root 4096 Sep 18 15:57 containers     # 容器信息的存储目录
drwx------ 4 root root   33 Aug 25 16:39 devicemapper   # devicemapper存储驱动基础设备存储目录
drwx------ 3 root root   26 Aug 25 16:16 image          # 容器镜像的存储与元数据目录
drwxr-x--- 3 root root   19 Aug 25 16:16 network
drwx------ 4 root root   32 Aug 25 16:16 plugins
drwx------ 2 root root    6 Aug 25 16:16 swarm
drwx------ 2 root root    6 Aug 31 09:42 tmp
drwx------ 2 root root    6 Aug 25 16:16 trust
drwx------ 2 root root   25 Aug 25 16:16 volumes
[root@kube-master docker.bak]# cd devicemapper/ ; ls -l
total 8
drwx------ 2 root root 4096 Sep 18 15:57 metadata       # 容器基础设备元数据的存储目录
drwxr-xr-x 5 root root  222 Sep 18 15:57 mnt            # 容器基础设备的挂载点，其中包含每个容器的rootfs。
[root@kube-master devicemapper]# cd mnt/ ; ls -l
total 0
drwxr-xr-x 3 root root 30 Aug 25 16:39 735cd040042fd601e2913da845c39d829661d757f0301909c1bbdcd049095f43
drwxr-xr-x 3 root root 30 Aug 25 16:39 7386a79a6900056893ee81096f920843c41a0eb60001d4e5acf1cd29b0b0f10e    # 容器基础设备的挂载点
drwxr-xr-x 3 root root 30 Aug 25 16:40 e7758da5fd9e2078a1306658cd43de7e5829b5bcce373f5eb18543a3eed3166d
[root@kube-master mnt]# ls -l 735cd040042fd601e2913da845c39d829661d757f0301909c1bbdcd049095f43/
total 4
-rw-------  1 root root  64 Aug 25 16:39 id
drwxr-xr-x 22 root root 263 Sep 18 15:55 rootfs
[root@kube-master mnt]# docker inspect fe61d9bb394a | grep -B4 -A3 735cd040042fd601e2913da845c39d829661d757f0301909c1bbdcd049095f43
        "GraphDriver": {
            "Name": "devicemapper",
            "Data": {
                "DeviceId": "391",                          # 指定容器的容器基础设备名称
                "DeviceName": "docker-253:1-25165889-735cd040042fd601e2913da845c39d829661d757f0301909c1bbdcd049095f43",
                "DeviceSize": "10737418240"
            }                   # 默认情况下，为容器基础设备提供10G存储。
        },
[root@kube-master mnt]# ls 735cd040042fd601e2913da845c39d829661d757f0301909c1bbdcd049095f43/rootfs/
bin  dev  etc  home  lib  lib64  lost+found  media  mnt  opt  proc  root  run  sbin  selinux  srv  sys  tmp  usr  var
```

- **Docker storage存储配置文件与创建过程：**

1. /etc/sysconfig/docker-storage-setup

```
# STORAGE_DRIVER=overlay2          # 默认的docker守护进程存储驱动为overlay2
STORAGE_DRIVER=devicemapper
DEVS=/dev/sdd                      # 单独指定磁盘
VG=dockervg
~   # 指定devicemapper存储驱动创建的资源池（LV thin-pool）的VG名称。
~
```

2. /etc/sysconfig/docker-storage

```
DOCKER_STORAGE_OPTIONS="--storage-driver devicemapper --storage-opt dm.fs=xfs --storage-opt dm.thinpooldev=/dev/mapper/dockervg-docker--pool --storag
e-opt dm.use_deferred_removal=true --storage-opt dm.use_deferred_deletion=true "
~
~ # docker守护进程的存储驱动配置参数：devicemapper
```

```
[root@master ~]# grep -vE '^#|^$' /usr/lib/docker-storage-setup/docker-storage-setup      注意：使用`man docker-storage-setup`查看docker-storage的配置参数！
STORAGE_DRIVER=devicemapper
DATA_SIZE=40%FREE
MIN_DATA_SIZE=2G
CHUNK_SIZE=512K
GROWPART=false
AUTO_EXTEND_POOL=yes
POOL_AUTOEXTEND_THRESHOLD=60
POOL_AUTOEXTEND_PERCENT=20
DEVICE_WAIT_TIMEOUT=60
WIPE_SIGNATURES=false
DOCKER_ROOT_VOLUME=no
DOCKER_ROOT_VOLUME_SIZE=40%FREE
[root@master ~]# cat /etc/sysconfig/docker-storage-setup
# Edit this file to override any configuration options specified in
# /usr/lib/docker-storage-setup/docker-storage-setup.
#
# For more details refer to "man docker-storage-setup"
DEVS=/dev/vdb
VG=docker-vg
SETUP_LVM_THIN_POOL=yes
[root@master ~]# cat /etc/sysconfig/docker-storage
DOCKER_STORAGE_OPTIONS="--storage-driver devicemapper --storage-opt dm.fs=xfs --storage-opt dm.thinpooldev=/dev/mapper/docker--vg-docker--pool --stor
age-opt dm.use_deferred_removal=true "
[root@master ~]# blkid
/dev/vda1: UUID="d85d7be1-03aa-433a-a847-030104cc3ce2" TYPE="xfs"
/dev/vdb1: UUID="ySZrAp-CTev-Y697-Mef4-YxmF-YVvj-71EDY1" TYPE="LVM2_member"
```

3. Docker devicemapper配置过程：

```
[root@cloud-ctl ~]# docker-storage-setup    # 必须停止运行docker守护进程，配置存储驱动参数后执行该命令。
INFO: Device node /dev/sdd1 exists.
  Physical volume "/dev/sdd1" successfully created.
  Volume group "docker-vg" successfully created          # devicemapper存储驱动创建报错，需更改配置文件中参数。
INFO: Storage is already configured with overlay2 driver. Can't configure it with devicemapper driver. To override, remove /etc/sysconfig/docker-stor
age and retry.
[root@cloud-ctl ~]# cat /etc/sysconfig/docker-storage-setup
# STORAGE_DRIVER=overlay2
STORAGE_DRIVER=devicemapper
DEVS=/dev/sdd
VG=docker-vg
[root@cloud-ctl ~]# cat /etc/sysconfig/docker-storage
DOCKER_STORAGE_OPTIONS="--storage-driver overlay2 "       # 更改overlay2为devicemapper，再运行docker-storage-setup命令即可。
```

```
[root@cloud-ctl ~]# cat /etc/sysconfig/docker-storage
DOCKER_STORAGE_OPTIONS="--storage-driver devicemapper "  # docker守护进程devicemapper存储驱动配置文件，将其配置为devicemapper。
[root@cloud-ctl ~]# docker-storage-setup
INFO: Device /dev/sdd is already partitioned and is part of volume group docker-vg     # 更改devicemapper存储驱动配置文件后，重新
  Using default stripesize 64.00 KiB.                                                  执行docker-storage-setup命令创建devicemapper
  Rounding up size to full physical extent 24.00 MiB                                   存储驱动的LV thin-pool资源池。
  Thin pool volume with chunk size 512.00 KiB can address at most 126.50 TiB of data.
  Logical volume "docker-pool" created.
  Logical volume docker-vg/docker-pool changed.
[root@cloud-ctl ~]# vgs
  VG        #PV #LV #SN Attr   VSize   VFree
  docker-vg   1   1   0 wz--n- <20.00g  12.00g   # devicemapper存储驱动的LV thin-pool资源池的VG，VG容量不会全部使用完毕，剩余容量将在
  dockervg    1   1   0 wz--n- <20.00g      0    LV thin-pool达到自动扩容百分比时自动扩展。
  isovg       1   1   0 wz--n- <50.00g <30.00g
  rootvg      3   1   0 wz--n-  27.50g      0
[root@cloud-ctl ~]# lvs
  LV          VG        Attr       LSize   Pool Origin Data%  Meta%  Move Log Cpy%Sync Convert
  docker-pool docker-vg twi-a-t---  <7.95g             0.00   0.15    # LV thin-pool资源池创建用于存储数据的LV与用于存储元数据的LV
  lvregistry  dockervg  -wi-ao---- <20.00g
  lviso       isovg     -wi-ao----  20.00g
  lv0         rootvg    -wi-ao----  27.50g
[root@cloud-ctl ~]#
```

注意：

1. 配置docker守护进程devicemapper存储驱动时，存储分配情况：
   a. 分配独立的磁盘用于创建LV thin-pool资源池
   b. 建议创建独立的文件系统用于存储容器与镜像快照