1. Kill a container. This abruptly exits from a container without even formally executing an exit.

Here is a container running.

```
osgdev@TG-DevOps-OS004:~$ docker container run -it --name app_doc
ubuntu:16.04
root@3c99d1886cbf:/#
```

Stop this container from another window.

```
osgdev@TG-DevOps-OS004:~/dockerlab$ docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
3c99d1886cbf        ubuntu:16.04        "/bin/bash"         About a
minute ago   Up 59 seconds                           app_doc

osgdev@TG-DevOps-OS004:~/dockerlab$ docker container stop app_doc
app_doc
```

Output on former window where the container was started: (Exit happened automatically)

```
osgdev@TG-DevOps-OS004:~$ docker container run -it --name app_doc
ubuntu:16.04
root@3c99d1886cbf:/#
root@3c99d1886cbf:/# exit
osgdev@TG-DevOps-OS004:~$
```

Now we shall kill this container.

```
osgdev@TG-DevOps-OS004:~$ docker container ls -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS                      PORTS               NAMES
3c99d1886cbf        ubuntu:16.04        "/bin/bash"         3 minutes ago
Exited (0) 2 minutes ago                        app_doc

osgdev@TG-DevOps-OS004:~$ docker start app_doc
app_doc

osgdev@TG-DevOps-OS004:~$ docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
3c99d1886cbf        ubuntu:16.04        "/bin/bash"         5 minutes ago
Up 5 seconds                            app_doc

osgdev@TG-DevOps-OS004:~$ docker container attach app_doc
root@3c99d1886cbf:/#
```

From another window, kill this container:

```
osgdev@TG-DevOps-OS004:~/dockerlab$ docker container kill app_doc
app_doc
```

The result of killing the container:

```
osgdev@TG-DevOps-OS004:~$ docker attach app_doc
root@3c99d1886cbf:/#
root@3c99d1886cbf:/# osgdev@TG-DevOps-OS004:~$
```

Note: observe there is no formal exit like it happened with docker stop command.

---

2. Restart a container:

Here we have a container running since last 28 seconds. We shall attach to the container.

```
osgdev@TG-DevOps-OS004:~$ docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
3c99d1886cbf        ubuntu:16.04        "/bin/bash"         10 minutes
ago      Up 28 seconds                          app_doc
osgdev@TG-DevOps-OS004:~$ docker container attach app_doc
root@3c99d1886cbf:/#
```

As we restart this container from another window

```
osgdev@TG-DevOps-OS004:~/dockerlab$ docker container restart app_doc
app_doc
```

Following happened automatically with container exits and back in start state 5 seconds ago

```
root@3c99d1886cbf:/#
root@3c99d1886cbf:/# exit
osgdev@TG-DevOps-OS004:~$ docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
3c99d1886cbf        ubuntu:16.04        "/bin/bash"         11 minutes
ago      Up 5 seconds                           app_doc
```

---

3. Suspend (Pause) a container. Container is not stopped, but it stop responding to any external events.

Here we have a running container:

```
osgdev@TG-DevOps-OS004:~$ docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
3c99d1886cbf        ubuntu:16.04        "/bin/bash"         14 minutes
ago      Up 3 minutes                           app_doc
```

You can execute command interactively through the shell:

```
osgdev@TG-DevOps-OS004:~$ docker container attach app_doc
```

```
root@3c99d1886cbf:/#
root@3c99d1886cbf:/# ls
bin   dev  home  lib64  mnt  proc  run   srv  tmp  var
boot  etc  lib   media  opt  root  sbin  sys  usr
root@3c99d1886cbf:/# pwd
/
root@3c99d1886cbf:/#
```

Let us pause this container:

```
osgdev@TG-DevOps-OS004:~/dockerlab$ docker container pause app_doc
app_doc
osgdev@TG-DevOps-OS004:~/dockerlab$ docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS                  PORTS               NAMES
3c99d1886cbf        ubuntu:16.04        "/bin/bash"         16 minutes
ago      Up 5 minutes (Paused)                          app_doc
```

Now any command you type in front of shell prompt, it simply don't appear there.

```
root@3c99d1886cbf:/# pwd
/
root@3c99d1886cbf:/#
```

Now we shall unpause the container

```
osgdev@TG-DevOps-OS004:~/dockerlab$ docker container unpause app_doc
app_doc
```

But the moment you unpause the container, you can see the commands that did not appear in previous steps.

```
root@3c99d1886cbf:/# pwd
/
root@3c99d1886cbf:/# lspwd
bash: lspwd: command not found
root@3c99d1886cbf:/#
```

You can also check that the container is running normally.

```
osgdev@TG-DevOps-OS004:~/dockerlab$ docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
3c99d1886cbf        ubuntu:16.04        "/bin/bash"         19 minutes
ago      Up 8 minutes                               app_doc
```

---

4. Execute a command remotely without interactive access. Its not necessary that you need to have interactive access (-it) and attach to container later to work with container.

Here we have a running container:

```
osgdev@TG-DevOps-OS004:~/dockerlab$ docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
3c99d1886cbf        ubuntu:16.04        "/bin/bash"         19 minutes
ago      Up 8 minutes                           app_doc

osgdev@TG-DevOps-OS004:~/dockerlab$ docker container exec app_doc ls /usr
bin
games
include
lib
local
sbin
share
src
osgdev@TG-DevOps-OS004:~/dockerlab$ docker container exec app_doc pwd
/
```

---

5. Wait for a container. Assuming that you need to wait for a container to terminate so that you can proceed with another activity.

Here is a running container, let us attach to it.

```
osgdev@TG-DevOps-OS004:~$ docker container start app_doc
app_doc
osgdev@TG-DevOps-OS004:~$ docker container attach app_doc
root@3c99d1886cbf:/#
root@3c99d1886cbf:/#
```

From another window execute a wait operation on this container. Observe that the shell hangs, and is not coming back to next command.

```
osgdev@TG-DevOps-OS004:~/dockerlab$ docker container wait app_doc
```

Now in the previous window where the container is running, make a formal exit and watch what happen in the window in which we are waiting for the container.

```
root@3c99d1886cbf:/#
root@3c99d1886cbf:/# exit 9
exit
osgdev@TG-DevOps-OS004:~$
```

Deliberately added an exit code 9. You can watch this exit code 9 is caught by waiting window.

```
osgdev@TG-DevOps-OS004:~/dockerlab$ docker container wait app_doc

9
```

```
osgdev@TG-DevOps-OS004:~/dockerlab$
```

---

6. Fetch container logs for all command line activity it did so far.

```
osgdev@TG-DevOps-OS004:~/dockerlab$ docker container logs app_doc
root@3c99d1886cbf:/#
root@3c99d1886cbf:/# exit
root@3c99d1886cbf:/#
root@3c99d1886cbf:/#
root@3c99d1886cbf:/# exit
root@3c99d1886cbf:/# exit
root@3c99d1886cbf:/# exit
root@3c99d1886cbf:/#
root@3c99d1886cbf:/# exit
root@3c99d1886cbf:/#
root@3c99d1886cbf:/# ls
bin   dev  home  lib64  mnt  proc  run   srv  tmp  var
boot  etc  lib   media  opt  root  sbin  sys  usr
root@3c99d1886cbf:/# pwd
/
root@3c99d1886cbf:/# lspwd
bash: lspwd: command not found
root@3c99d1886cbf:/# ls
bin   dev  home  lib64  mnt  proc  run   srv  tmp  var
boot  etc  lib   media  opt  root  sbin  sys  usr
root@3c99d1886cbf:/# exit
exit
root@3c99d1886cbf:/#
root@3c99d1886cbf:/# exit 9
exit
```

---

7. Checking the changes made to container environment. Changes are shown with file system of container.

```
osgdev@TG-DevOps-OS004:~$ docker container start app_doc
app_doc
osgdev@TG-DevOps-OS004:~$ docker container attach app_doc
root@3c99d1886cbf:/#
root@3c99d1886cbf:/# mkdir MASTER
root@3c99d1886cbf:/# touch testfile
root@3c99d1886cbf:/# ls
MASTER  boot  etc   lib    media  opt   root  sbin  sys       tmp  var
bin     dev   home  lib64  mnt    proc  run   srv   testfile  usr
root@3c99d1886cbf:/#

osgdev@TG-DevOps-OS004:~/dockerlab$ docker container diff app_doc
A /MASTER
C /root
A /root/.bash_history
```

```
A /testfile
```

Note: Folder "MASTER" and file "testfile" are added, while "/root" is changed with ".bash_history" added to /root.

---

8. Container is a running process. Here is the detail of the process.

```
osgdev@TG-DevOps-OS004:~/dockerlab$ docker container top app_doc
UID                 PID                 PPID                C
STIME               TTY                 TIME                CMD
root                12892               12877               0
11:14               pts/0               00:00:00            /bin/bash
```

Note: Process ID of app_doc container is 12892 (running the command /bin/bash) and its parent is 12877 running container service.

```
osgdev@TG-DevOps-OS004:~/dockerlab$ ps -ef | grep 12892
root     12892 12877  0 11:14 pts/0    00:00:00 /bin/bash

osgdev@TG-DevOps-OS004:~/dockerlab$ ps -ef | grep 12877
root     12877 12261  0 11:14 ?        00:00:00 docker-containerd-shim -
namespace moby -workdir
/var/lib/docker/containerd/daemon/io.containerd.runtime.v1.linux/moby/3c9
9d1886cbfc74450345b3339fcf5f063305ee528b95c8508789d45c79f1c87 -address
/var/run/docker/containerd/docker-containerd.sock -containerd-binary
/usr/bin/docker-containerd -runtime-root /var/run/docker/runtime-runc
```

Tracking the process tree upwards till init process. Just for your curiosity to look into docker internals.

```
osgdev@TG-DevOps-OS004:~/dockerlab$ ps -ef | grep 12261
root     12261 12229  0 Apr10 ?        00:01:23 docker-containerd --
config /var/run/docker/containerd/containerd.toml

osgdev@TG-DevOps-OS004:~/dockerlab$ ps -ef | grep 12229
root     12229     1  0 Apr10 ?        00:01:37 /usr/bin/dockerd -H fd://

osgdev@TG-DevOps-OS004:~/dockerlab$ ps -ef | grep 1
root         1     0  0 Mar29 ?        00:00:22 /sbin/init
```

---

9. Capture the events happening in Docker Daemon. Initially it hangs waiting for events. Later it would keep capturing events happening in Docker Daemon across the system irrespective of containers.

```
osgdev@TG-DevOps-OS004:~/dockerlab$ docker system events


osgdev@TG-DevOps-OS004:~$ docker container start app_doc
```

```
app_doc

2018-04-11T11:28:56.284348688+05:30 network connect
9af5ffc53ff5a67d5307d1323126091296053ac67908a9e969edb7a3661b496c
(container=3c99d1886cbfc74450345b3339fcf5f063305ee528b95c8508789d45c79f1c
87, name=bridge, type=bridge)
2018-04-11T11:28:56.593302546+05:30 container start
3c99d1886cbfc74450345b3339fcf5f063305ee528b95c8508789d45c79f1c87
(image=ubuntu:16.04, name=app_doc)

osgdev@TG-DevOps-OS004:~$ docker container attach app_doc
root@3c99d1886cbf:/#
root@3c99d1886cbf:/#

2018-04-11T11:30:22.303776070+05:30 container attach
3c99d1886cbfc74450345b3339fcf5f063305ee528b95c8508789d45c79f1c87
(image=ubuntu:16.04, name=app_doc)
2018-04-11T11:30:22.306731711+05:30 container resize
3c99d1886cbfc74450345b3339fcf5f063305ee528b95c8508789d45c79f1c87
(height=24, image=ubuntu:16.04, name=app_doc, width=81)
2018-04-11T11:30:22.308359577+05:30 container resize
3c99d1886cbfc74450345b3339fcf5f063305ee528b95c8508789d45c79f1c87
(height=23, image=ubuntu:16.04, name=app_doc, width=80)

root@3c99d1886cbf:/#
root@3c99d1886cbf:/# exit
exit
osgdev@TG-DevOps-OS004:~$

2018-04-11T11:31:13.522302185+05:30 container die
3c99d1886cbfc74450345b3339fcf5f063305ee528b95c8508789d45c79f1c87
(exitCode=0, image=ubuntu:16.04, name=app_doc)
2018-04-11T11:31:13.638766943+05:30 network disconnect
9af5ffc53ff5a67d5307d1323126091296053ac67908a9e969edb7a3661b496c
(container=3c99d1886cbfc74450345b3339fcf5f063305ee528b95c8508789d45c79f1c
87, name=bridge, type=bridge)

osgdev@TG-DevOps-OS004:~$ docker container restart app_doc
app_doc


2018-04-11T11:32:02.843689431+05:30 network connect
9af5ffc53ff5a67d5307d1323126091296053ac67908a9e969edb7a3661b496c
(container=3c99d1886cbfc74450345b3339fcf5f063305ee528b95c8508789d45c79f1c
87, name=bridge, type=bridge)
2018-04-11T11:32:03.190694022+05:30 container start
3c99d1886cbfc74450345b3339fcf5f063305ee528b95c8508789d45c79f1c87
(image=ubuntu:16.04, name=app_doc)
2018-04-11T11:32:03.190763039+05:30 container restart
3c99d1886cbfc74450345b3339fcf5f063305ee528b95c8508789d45c79f1c87
(image=ubuntu:16.04, name=app_doc)
^C
osgdev@TG-DevOps-OS004:~/dockerlab$
```

Note: Press ctrl-c to return to command prompt

---

10. Track the resource utilization of docker container.

Attach to a running container:

```
osgdev@TG-DevOps-OS004:~$ docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
3c99d1886cbf        ubuntu:16.04        "/bin/bash"         About an hour
ago   Up 6 minutes                            app_doc
osgdev@TG-DevOps-OS004:~$ docker container attach app_doc
root@3c99d1886cbf:/#
root@3c99d1886cbf:/#
```

Execute following command in another window where you can watch the status usage inteactively

```
osgdev@TG-DevOps-OS004:~/dockerlab$ docker container stats app_doc

CONTAINER ID        NAME                CPU %               MEM USAGE /
LIMIT   MEM %               NET I/O             BLOCK I/O           PIDS
3c99d1886cbf        app_doc             0.00%               484KiB /
3.859GiB   0.01%               4.93kB / 0B         106kB / 0B          1
^C
osgdev@TG-DevOps-OS004:~/dockerlab$
```

Press ctrl-c to return to command prompt.

---

11. Let us now launch a practical container with tomcat server running inside.

Note: docker container run is a versatile command, which creates container from a given image. It looks for the image locally in the machine, if not available then it will search in docker hub (https://hub.docker.com/). If found it will pull the image and create the container and also start the container.

-d flag is used to let the container run in background (daemon mode)
-P flag is used to expose the tomcat port (8080) and forward the same to any available port randomly chosen by docker daemon.

```
osgdev@TG-DevOps-OS004:~/dockerlab$ docker container run -d -P tomcat:8
Unable to find image 'tomcat:8' locally
8: Pulling from library/tomcat
c73ab1c6897b: Pull complete
1ab373b3deae: Pull complete
b542772b4177: Pull complete
0bcc3741ab14: Pull complete
```

```
421d624d778d: Pull complete
26ad58237506: Pull complete
8dbabc90b2b8: Pull complete
982930be204d: Pull complete
80869be51738: Pull complete
b71ce0f0260c: Pull complete
b18814a5c704: Pull complete
e3fbb69d7797: Pull complete
f2a4b7aaa851: Pull complete
Digest:
sha256:15f12b529a268986eb86224477f22ddfdf4a42383d6758ea14eaed10b3c8a8e9
Status: Downloaded newer image for tomcat:8
11d47ebc7e8d3d7de3894f546fc3320836873b7b8c6d18455f0c47136142c5d4

osgdev@TG-DevOps-OS004:~/dockerlab$ docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS                     NAMES
11d47ebc7e8d        tomcat:8            "catalina.sh run"   4 minutes ago
Up 4 minutes        0.0.0.0:32768->8080/tcp   optimistic_bardeen
```

You can now access the tomcat server using the following URL:
http://localhost:32768/

You may also locate the forwarded port of tomcat container using the following command.

```
osgdev@TG-DevOps-OS004:~/dockerlab$ docker container port
optimistic_bardeen
8080/tcp -> 0.0.0.0:32768
```

You can also specify a particular port number while creating a container.

```
osgdev@TG-DevOps-OS004:~/dockerlab$ docker container run -d -p 11022:8080
--name tomcat_app tomcat:8
0617c2cca00290d77a7a304e55ccffca60b2c6707d52355a8d2e6886b51bcc81

osgdev@TG-DevOps-OS004:~/dockerlab$ docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS                     NAMES
0617c2cca002        tomcat:8            "catalina.sh run"   About a
minute ago   Up About a minute   0.0.0.0:11022->8080/tcp   tomcat_app
11d47ebc7e8d        tomcat:8            "catalina.sh run"   4 hours ago
Up 4 hours          0.0.0.0:32768->8080/tcp   optimistic_bardeen
```

---

12. If you have a .war file you can copy the same into webapps folder inside this tomcat container.

Locate the path of webapps folder inside the tomcat container:

```
osgdev@TG-DevOps-OS004:~/dockerlab$ docker container exec
optimistic_bardeen pwd
```

```
/usr/local/tomcat

osgdev@TG-DevOps-OS004:~/dockerlab$ docker container exec
optimistic_bardeen ls
LICENSE
NOTICE
RELEASE-NOTES
RUNNING.txt
bin
conf
include
lib
logs
native-jni-lib
temp
webapps
work

osgdev@TG-DevOps-OS004:~/dockerlab$ docker container exec
optimistic_bardeen ls /usr/local/tomcat/webapps
ROOT
docs
examples
host-manager
manager
```

If you have any previously created war file (from early part of this DevOps Tools training), you may now copy the same inside webapps folder, using the following command.

```
osgdev@TG-DevOps-OS004:~/dockerlab$ ls
day12.log  NewApp1.war  screen.log  test.tar

osgdev@TG-DevOps-OS004:~/dockerlab$ docker container cp
/home/osgdev/dockerlab/NewApp1.war
optimistic_bardeen:/usr/local/tomcat/webapps

osgdev@TG-DevOps-OS004:~/dockerlab$ docker container exec
optimistic_bardeen ls /usr/local/tomcat/webapps
NewApp1
NewApp1.war
ROOT
docs
examples
host-manager
manager
```

Now you can access your webapplication using the following URL:

```
http://localhost:32768/NewApp1/
```

---

13. Docker Networking subcommands.

```
osgdev@TG-DevOps-OS004:~/dockerlab$ docker network

Usage:      docker network COMMAND

Manage networks

Options:


Commands:
  connect     Connect a container to a network
  create      Create a network
  disconnect  Disconnect a container from a network
  inspect     Display detailed information on one or more networks
  ls          List networks
  prune       Remove all unused networks
  rm          Remove one or more networks

Run 'docker network COMMAND --help' for more information on a command.

osgdev@TG-DevOps-OS004:~/dockerlab$ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
9af5ffc53ff5        bridge              bridge              local
2f17cc107ea7        host                host                local
23c983327ebe        none                null                local
```

---

14. Create a new container (ubuntu:14.04 image is used due to availability of ifconfig command in this image) and check whether it has any IP address.

```
osgdev@TG-DevOps-OS004:~/dockerlab$ docker container run -it ubuntu:14.04
root@a07f69f24918:/# ifconfig
eth0      Link encap:Ethernet  HWaddr 02:42:ac:11:00:02
          inet addr:172.17.0.2  Bcast:172.17.255.255  Mask:255.255.0.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:20 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:2897 (2.8 KB)  TX bytes:0 (0.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

---

15. While the container is running, check in another window the details of this container.

```
osgdev@TG-DevOps-OS004:~$ docker container inspect a07f69f24918
[
    {
        "Id":
"a07f69f249185723bede2d65ad95686ebd7c0a1e34b83353d2d470b44b4a6db4",
        "Created": "2018-04-11T13:33:32.813184597Z",
        "Path": "/bin/bash",

<<<<<<<<<<<<  OUTPUT RESPONSE IS EDITED OUT   >>>>>>>>>>>>>>>>>>>>>>>>

            "Networks": {
                "bridge": {
                    "IPAMConfig": null,
                    "Links": null,
                    "Aliases": null,
                    "NetworkID":
"9af5ffc53ff5a67d5307d1323126091296053ac67908a9e969edb7a3661b496c",
                    "EndpointID":
"4b5c72cdf9c7a99a7b941ae93f17e23864e7252767cc0631ae6118a661d64379",
                    "Gateway": "172.17.0.1",
                    "IPAddress": "172.17.0.2",
                    "IPPrefixLen": 16,
                    "IPv6Gateway": "",
                    "GlobalIPv6Address": "",
                    "GlobalIPv6PrefixLen": 0,
                    "MacAddress": "02:42:ac:11:00:02",
                    "DriverOpts": null
                }
            }
        }
    }
]
```

Note: The last part of response (JSON) shows that the network is "bridge" and its ID is "NetworkID": "9af5ffc53ff5a67d5307d1323126091296053ac67908a9e969edb7a3661b496c"

Compare this with list of networks we got in previous step. Our container is connected to bridge network.

```
osgdev@TG-DevOps-OS004:~/dockerlab/whale$ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
9af5ffc53ff5        bridge              bridge              local
2f17cc107ea7        host                host                local
23c983327ebe        none                null                local
```

---

16. Confirm connectivity of bridge network to our container, get the detil of the network.

```
osgdev@TG-DevOps-OS004:~$ docker network inspect bridge
[
    {
```

```
        "Name": "bridge",
        "Id":
"9af5ffc53ff5a67d5307d1323126091296053ac67908a9e969edb7a3661b496c",
        "Created": "2018-04-10T19:09:17.61577547+05:30",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "172.17.0.0/16",
                    "Gateway": "172.17.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {

"a07f69f249185723bede2d65ad95686ebd7c0a1e34b83353d2d470b44b4a6db4": {
                "Name": "gracious_kare",
                "EndpointID":
"4b5c72cdf9c7a99a7b941ae93f17e23864e7252767cc0631ae6118a661d64379",
                "MacAddress": "02:42:ac:11:00:02",
                "IPv4Address": "172.17.0.2/16",
                "IPv6Address": ""
            }
        },
        "Options": {
            "com.docker.network.bridge.default_bridge": "true",
            "com.docker.network.bridge.enable_icc": "true",
            "com.docker.network.bridge.enable_ip_masquerade": "true",
            "com.docker.network.bridge.host_binding_ipv4": "0.0.0.0",
            "com.docker.network.bridge.name": "docker0",
            "com.docker.network.driver.mtu": "1500"
        },
        "Labels": {}
    }
]
```


Note: Observe the container (a07f69f24918)being listed under containers. The network is using the subnet "172.17.0.0/16". Hence the container got the IP address "172.17.0.2/16"

---

17. Create another container to check the IP address allocated to second container.

```
osgdev@TG-DevOps-OS004:~$ docker container run -it ubuntu:14.04
root@31a385ad08fc:/# ifconfig
eth0      Link encap:Ethernet  HWaddr 02:42:ac:11:00:03
          inet addr:172.17.0.3  Bcast:172.17.255.255  Mask:255.255.0.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:13 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:1783 (1.7 KB)  TX bytes:0 (0.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

root@31a385ad08fc:/# ping 172.17.0.2
PING 172.17.0.2 (172.17.0.2) 56(84) bytes of data.
64 bytes from 172.17.0.2: icmp_seq=1 ttl=64 time=0.145 ms
64 bytes from 172.17.0.2: icmp_seq=2 ttl=64 time=0.081 ms
^C
--- 172.17.0.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.081/0.113/0.145/0.032 ms
```

From the first container:

```
osgdev@TG-DevOps-OS004:~/dockerlab$ docker container run -it ubuntu:14.04
root@a07f69f24918:/# ifconfig
eth0      Link encap:Ethernet  HWaddr 02:42:ac:11:00:02
          inet addr:172.17.0.2  Bcast:172.17.255.255  Mask:255.255.0.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:20 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:2897 (2.8 KB)  TX bytes:0 (0.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

root@a07f69f24918:/# ping 172.17.0.3
PING 172.17.0.3 (172.17.0.3) 56(84) bytes of data.
64 bytes from 172.17.0.3: icmp_seq=1 ttl=64 time=0.118 ms
64 bytes from 172.17.0.3: icmp_seq=2 ttl=64 time=0.099 ms
^C
--- 172.17.0.3 ping statistics ---
```

```
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.099/0.108/0.118/0.014 ms
```

---

18. Create a new network and let the new container shall connect to this new network.

```
osgdev@TG-DevOps-OS004:~$ docker network create new-net
9c2e31f9ea0fd6ac0b5f9c0fd65a640b070d63eff834e08b8944fcd3855bde8d
osgdev@TG-DevOps-OS004:~$ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
9af5ffc53ff5        bridge              bridge              local
2f17cc107ea7        host                host                local
9c2e31f9ea0f        new-net             bridge              local
23c983327ebe        none                null                local
osgdev@TG-DevOps-OS004:~$ docker container run -it --net new-net
ubuntu:14.04
root@159a0df34c38:/#
root@159a0df34c38:/# ifconfig
eth0      Link encap:Ethernet  HWaddr 02:42:ac:12:00:02
          inet addr:172.18.0.2  Bcast:172.18.255.255  Mask:255.255.0.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:71 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:11040 (11.0 KB)  TX bytes:0 (0.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

osgdev@TG-DevOps-OS004:~/dockerlab$ docker network inspect new-net
[
    {
        "Name": "new-net",
        "Id":
"9c2e31f9ea0fd6ac0b5f9c0fd65a640b070d63eff834e08b8944fcd3855bde8d",
        "Created": "2018-04-11T20:22:11.76184614+05:30",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.18.0.0/16",
                    "Gateway": "172.18.0.1"
                }
            ]
```

```
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {

"159a0df34c389fd60132ec440d59dc363e5485c1f115c291f16288aaf5aeecda": {
                "Name": "youthful_pare",
                "EndpointID":
"97a68e0d99339054e89b311c4a97fd349de0771d1e0d188d5eaf8eb5f37a9874",
                "MacAddress": "02:42:ac:12:00:02",
                "IPv4Address": "172.18.0.2/16",
                "IPv6Address": ""
            }
        },
        "Options": {},
        "Labels": {}
    }
]
```

---

19. As the new container created (159a0df34c38) in the previous step is connected only to new-net, you may also connect the same to bridge network.

```
osgdev@TG-DevOps-OS004:~/dockerlab$ docker network connect bridge
159a0df34c38
```

Check the IP address associated with the container:

```
root@159a0df34c38:/# ifconfig
eth0      Link encap:Ethernet  HWaddr 02:42:ac:12:00:02
          inet addr:172.18.0.2  Bcast:172.18.255.255  Mask:255.255.0.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:81 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:13163 (13.1 KB)  TX bytes:0 (0.0 B)

eth1      Link encap:Ethernet  HWaddr 02:42:ac:11:00:02
          inet addr:172.17.0.2  Bcast:172.17.255.255  Mask:255.255.0.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:23 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:3373 (3.3 KB)  TX bytes:0 (0.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
```

```
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

---

20. You can also disconnect the container from my-net network.

```
osgdev@TG-DevOps-OS004:~/dockerlab$ docker network disconnect new-net
159a0df34c38

root@159a0df34c38:/# ifconfig
eth1      Link encap:Ethernet  HWaddr 02:42:ac:11:00:02
          inet addr:172.17.0.2  Bcast:172.17.255.255  Mask:255.255.0.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:28 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:4521 (4.5 KB)  TX bytes:0 (0.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

---

21. The new-net is now not associated with any container. When the network is not connected to any of the network, it may be removed.

```
osgdev@TG-DevOps-OS004:~/dockerlab$ docker network inspect new-net
[
    {
        "Name": "new-net",
        "Id":
"9c2e31f9ea0fd6ac0b5f9c0fd65a640b070d63eff834e08b8944fcd3855bde8d",
        "Created": "2018-04-11T20:22:11.76184614+05:30",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.18.0.0/16",
                    "Gateway": "172.18.0.1"
```

```
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {},
        "Options": {},
        "Labels": {}
    }
]

osgdev@TG-DevOps-OS004:~/dockerlab$ docker network rm new-net
new-net

osgdev@TG-DevOps-OS004:~/dockerlab$ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
9af5ffc53ff5        bridge              bridge              local
2f17cc107ea7        host                host                local
23c983327ebe        none                null                local
```

Note: You may also use docker network prune to remove more than one network having no container connected to it.

---

22. Docker Volumes. You can create a new container and mount a volume inside the container and make it synchronize with a folder on the host machine.

```
osgdev@TG-DevOps-OS004:~/dockerlab$ mkdir SHARE

osgdev@TG-DevOps-OS004:~/dockerlab$ ls
day12.log  NewApp1.war  screen.log  SHARE  test.tar  whale

osgdev@TG-DevOps-OS004:~/dockerlab$ docker container run -it -v
/home/osgdev/dockerlab/SHARE/:/FROMHOST ubuntu:16.04
root@048e62898477:/# ls
FROMHOST  boot  etc   lib    media  opt   root  sbin  sys  usr
bin       dev   home  lib64  mnt    proc  run   srv   tmp  var
root@048e62898477:/# cd FROMHOST/
root@048e62898477:/FROMHOST# touch testfile
root@048e62898477:/FROMHOST# ls
testfile
root@048e62898477:/FROMHOST#
```

You can see the file in another window on the system folder SHARE

```
osgdev@TG-DevOps-OS004:~/dockerlab$ cd SHARE/
osgdev@TG-DevOps-OS004:~/dockerlab/SHARE$ ls
```

```
testfile
```

For this "testfile" created at root level inside the container, add a line from "SHARE"

```
osgdev@TG-DevOps-OS004:~/dockerlab/SHARE$ ls -l
total 0
-rw-r--r-- 1 root root 0 Apr 11 20:34 testfile
osgdev@TG-DevOps-OS004:~/dockerlab/SHARE$ sudo vi testfile
[sudo] password for osgdev:
osgdev@TG-DevOps-OS004:~/dockerlab/SHARE$ cat testfile
Put first line from system
```

You can view this inside the container, and you can add another line here.

```
root@048e62898477:/FROMHOST# cat >> testfile
Second line from container
^C
root@048e62898477:/FROMHOST#

View this on system folder SHARE

osgdev@TG-DevOps-OS004:~/dockerlab/SHARE$ cat testfile
Put first line from system
Second line from container
```

---

23. You can create another container and inherit the volume "FROMHOST", from the container where this volume is available and continue to sync with the "SHARE" folder of host machine.

```
osgdev@TG-DevOps-OS004:~$ docker container run -it --volumes-from
048e62898477 ubuntu:16.04
root@687925dc817b:/# ls
FROMHOST  boot  etc   lib    media  opt   root  sbin  sys  usr
bin       dev   home  lib64  mnt    proc  run   srv   tmp  var
root@687925dc817b:/# ls FROMHOST/
testfile
root@687925dc817b:/# cat ./FROMHOST/testfile
Put first line from system
Second line from container
root@687925dc817b:/#
```

---

24. Another practical use of volume based sharing with folder on host machine. Few steps back we created a tomcat container and copied war file inside the container into webapps folder. Here are the steps. This time we kept war file inside "web" folder.

```
osgdev@TG-DevOps-OS004:~/dockerlab$ docker container run -d -p 11022:8080
--name warcopy tomcat:8
32567b0da8848fd42a94b26c4e784dce6e29dce520fcf128d926fcd8518e59c0
```

```
osgdev@TG-DevOps-OS004:~/dockerlab$ mkdir web
osgdev@TG-DevOps-OS004:~/dockerlab$ mv ./NewApp1.war ./web/
osgdev@TG-DevOps-OS004:~/dockerlab$ ls ./web/
NewApp1.war

osgdev@TG-DevOps-OS004:~/dockerlab$ docker container cp
/home/osgdev/dockerlab/web/NewApp1.war warcopy:/usr/local/tomcat/webapps

osgdev@TG-DevOps-OS004:~/dockerlab$ docker container exec warcopy ls
/usr/local/tomcat/webapps
NewApp1
NewApp1.war
ROOT
docs
examples
host-manager
manager
```

We shall redo this activity with "web" folder being shared with "webapps" volume mounted at point /usr/local/tomcat. Note that original "webapps" folder whose content is listed in the above command is now overlayed with "webapps" volume mounted over that. Hence the original contents of "webapps" folder is not seen, instead the content of "web" folder on host machine is seen inside "webapps" volume.

```
osgdev@TG-DevOps-OS004:~/dockerlab$ docker container run -d -p 11055:8080
-v /home/osgdev/dockerlab/web:/usr/local/tomcat/webapps --name warshare
tomcat:8
48189932daf2d7332ff7b8dcb2882e50e50a259740f66cee0fedddd8a3d10656

osgdev@TG-DevOps-OS004:~/dockerlab$ docker container exec warshare ls
/usr/local/tomcat/webapps
NewApp1
NewApp1.war

osgdev@TG-DevOps-OS004:~/dockerlab$ ls web
NewApp1  NewApp1.war
```

You have these two containers listed below.

```
osgdev@TG-DevOps-OS004:~/dockerlab$ docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS                     NAMES
48189932daf2        tomcat:8            "catalina.sh run"   About an hour
ago   Up About an hour    0.0.0.0:11055->8080/tcp   warshare
32567b0da884        tomcat:8            "catalina.sh run"   About an hour
ago   Up About an hour    0.0.0.0:11022->8080/tcp   warcopy
```

---

25. Docker volume is another way of sharng information between containers. The information in the volume persists even after the containers are terminated.

```
osgdev@TG-DevOps-OS004:~/dockerlab$ docker volume

Usage:      docker volume COMMAND

Manage volumes

Options:


Commands:
  create      Create a volume
  inspect     Display detailed information on one or more volumes
  ls          List volumes
  prune       Remove all unused volumes
  rm          Remove one or more volumes

Run 'docker volume COMMAND --help' for more information on a command.
```

Your current machine may have many unwanted volumes. Delete all of them.

```
osgdev@TG-DevOps-OS004:~/dockerlab$ docker volume ls
DRIVER              VOLUME NAME
local
21838d9b146451894675bb2a7e7d948add33a9b5935763cf6eab85bc4c83e491
local
3ee9f70e7c3536cb4eb50966443530a1aae12ac003c049edc0f196928a367ccf
local
42586ab37c01411feee6330d97cb1301cedb4dcf03303909483ca744d9b0c894
local
45da15c30fb2be148cdd16b2604a10359c2d2fe2c7639d7a4089108e97a60c25
local
7ff4669a250badef0c5fa21c1eecd2337e82ea0a633ecac166ba9e15d222d335
local
d9a79cd7d0cbe7239eed0df9709847fc64f3726b7a69d3277e5076c132e85278

local               jenkins_home

osgdev@TG-DevOps-OS004:~/dockerlab$ docker volume prune
WARNING! This will remove all volumes not used by at least one container.
Are you sure you want to continue? [y/N] y
Deleted Volumes:
7ff4669a250badef0c5fa21c1eecd2337e82ea0a633ecac166ba9e15d222d335
d9a79cd7d0cbe7239eed0df9709847fc64f3726b7a69d3277e5076c132e85278
jenkins_home
foo
21838d9b146451894675bb2a7e7d948add33a9b5935763cf6eab85bc4c83e491
3ee9f70e7c3536cb4eb50966443530a1aae12ac003c049edc0f196928a367ccf
42586ab37c01411feee6330d97cb1301cedb4dcf03303909483ca744d9b0c894
45da15c30fb2be148cdd16b2604a10359c2d2fe2c7639d7a4089108e97a60c25

Total reclaimed space: 435.8MB

osgdev@TG-DevOps-OS004:~/dockerlab$ docker volume create foo
foo
```

```
osgdev@TG-DevOps-OS004:~/dockerlab$ docker volume ls
DRIVER              VOLUME NAME
local               foo

osgdev@TG-DevOps-OS004:~$ docker volume inspect foo
[
    {
        "CreatedAt": "2018-04-12T21:34:53+05:30",
        "Driver": "local",
        "Labels": {},
        "Mountpoint": "/var/lib/docker/volumes/foo/_data",
        "Name": "foo",
        "Options": {},
        "Scope": "local"
    }
]

osgdev@TG-DevOps-OS004:~/dockerlab$ docker container run -it -v
foo:/home/share_vol ubuntu:16.04
root@f03f5bc8d8ed:/# ls /home/
share_vol
root@f03f5bc8d8ed:/# cd /home/share_vol
```

In another window let us create one more container which will share the same volume. You can observe the testfile already existing in the shared volume.

```
osgdev@TG-DevOps-OS004:~$ docker container run -it -v foo:/opt/vol_res
ubuntu:16.04
root@e4714e1353f2:/# ls /opt
vol_res
root@e4714e1353f2:/# cd /opt/vol_res/
root@e4714e1353f2:/opt/vol_res# touch testfile
root@e4714e1353f2:/opt/vol_res# ls
testfile
```

Now in the previous container (f03f5bc8d8ed) you can see the testfile. Add some content to this file.

```
root@f03f5bc8d8ed:/home/share_vol# ls
testfile
root@f03f5bc8d8ed:/home/share_vol# cat >>testfile
some content to file
^C
root@f03f5bc8d8ed:/home/share_vol# cat testfile
some content to file
```

This may be seen in other container (e4714e1353f2). Add another line here.

```
root@e4714e1353f2:/opt/vol_res# cat testfile
some content to file
root@e4714e1353f2:/opt/vol_res# cat >> testfile
add more content
```

```
^C
root@e4714e1353f2:/opt/vol_res# cat testfile
some content to file
add more content
```

Now seen in container (f03f5bc8d8ed)

```
root@f03f5bc8d8ed:/home/share_vol# cat testfile
some content to file
add more content
```

As long as either of these containers are surviving, if you start another container with the same volume (foo) shared, the contents can be shared between the containers.

Terminate both the containers and they are also deleted.

```
root@f03f5bc8d8ed:/home/share_vol# exit
exit
osgdev@TG-DevOps-OS004:~/dockerlab$

root@e4714e1353f2:/opt/vol_res# exit
exit
osgdev@TG-DevOps-OS004:~$

osgdev@TG-DevOps-OS004:~$ docker container ls -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
```

Create new container sharing the same volume (foo)

```
osgdev@TG-DevOps-OS004:~$ docker container run -it -v foo:/home/new_vol
ubuntu:16.04
root@75010bf09fa5:/# ls /home/new_vol/
testfile
root@75010bf09fa5:/# cat /home/new_vol/testfile
some content to file
add more content
root@75010bf09fa5:/#
```

Actually this information is stored on host machine. If the storage is bigger, this may cause storage use even after the containers are removed. Hence need to be removed explicitly.

```
osgdev@TG-DevOps-OS004:~$ docker volume inspect foo
[
    {
        "CreatedAt": "2018-04-12T21:34:53+05:30",
        "Driver": "local",
        "Labels": {},
        "Mountpoint": "/var/lib/docker/volumes/foo/_data",
        "Name": "foo",
        "Options": {},
        "Scope": "local"
    }
```

```
]

osgdev@TG-DevOps-OS004:~$ sudo ls /var/lib/docker/volumes
[sudo] password for osgdev:
foo  metadata.db
osgdev@TG-DevOps-OS004:~$ sudo ls /var/lib/docker/volumes/foo
_data
osgdev@TG-DevOps-OS004:~$ sudo ls /var/lib/docker/volumes/foo/_data
testfile
osgdev@TG-DevOps-OS004:~$ sudo cat
/var/lib/docker/volumes/foo/_data/testfile
some content to file
add more content
```
