Q2:

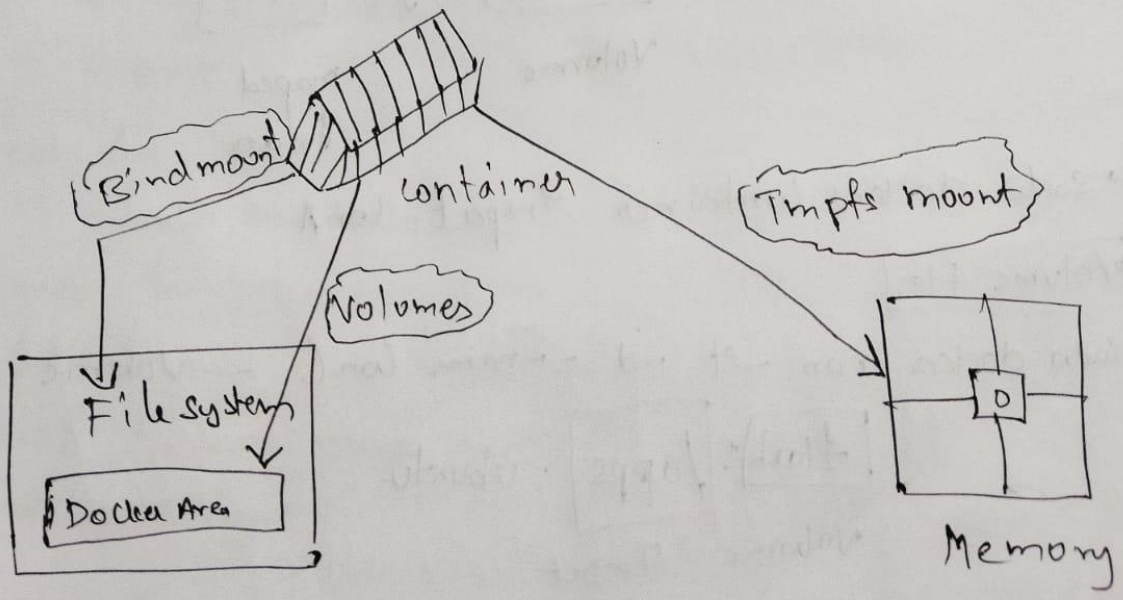
→ Advantages to Docker Storage

Normally if you wanted to store data in a Docker container it would be stored in the writable layer of the Docker container, but this is not an efficient way to store data. So we make use of different Docker storage types. These storage types have a lot of advantages over the default storage method.

- Persistent Data
- Transfer Data easily
- Increase container performance.

→ Types of Storage Types ←

1. Volumes (with docker)
2. Bind mount (without docker)
3. Tmpfs mount (Temp docker)



```
        }
    }
}
]
[root@localhost files]#
```

Docker Volumes

Docker volumes are basically persistent storage locations for the containers. They are managed by docker completely. They can be easily attached and removed from containers, you can backup your volumes also. This is the most used type of data storage.

→ Sudo docker volume create <name>
→ Sudo docker volume ls
→ Sudo docker volume inspect <name>
→ Sudo docker volume rm <batman>
→ Sudo docker volume crate batman
supermon
flash
→ Sudo docker volume prune (delete all volumes)

mount flag

→ Sudo docker run -it -d --name conA **--mount** source = **batman**, target = **/apps** ubuntu

batman → Volume
/apps → maped folder

→ Sudo docker container inspect conA

Volume flag

→ Sudo docker run -it -d --name conC --volume **flash**:**/apps** ubuntu

flash → Volume
/apps → Target

Readonly Volumes

→ sudo docker Volume create SupermanRo
→ sudo docker run -it -d --name Con1 --mount
Source = SupamanRo, target = /apps, readonly ubuntu

* Bind mount *

Bind mount's aren't managed by Docker and are mapped to a host system directory.

→ sudo docker run -it -d --name ConAB ubuntu
→ sudo docker run -it -d --name ConABC --mount
type = bind, Source = $(pwd), target = /apps ubuntu

→ sudo docker container inspect ConABC
→ docker exec -it ConABC bash
↳ cd apps/
↳ touch hello.txt
↳ Exit

→ cd fun
→ ls (will se hello.txt)
→ touch fun.txt
→ cat >> fun.txt → "Hellow word"
→ sudo docker ~~exec~~ exec -it ConABC bash
→ readonly bound mount ←
→ sudo docker run -it -d --name ConABCD --mount type = bind,
Source = $(pwd)/funs, target = /apps, readonly ubuntu
→ sudo docker ps

* Tmpfs mount *

This type of storage map to the host system (Linux) memory. Tmpfs is not persistent like volumes and tmpfs and get removed when the containers they are attached to are stopped. They only ever get mapped to a single container in their lifetime. Allow you store more temporary data without affecting a container's efficiency

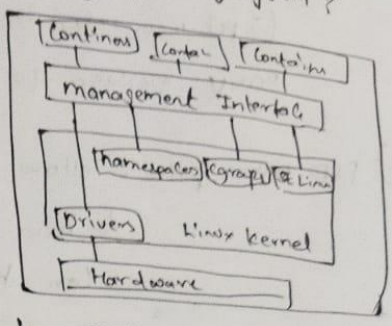
→ docker run -it -d --name <temporary Container>
--mount type=tmpfs, target=/apps ubuntu
(Flag)

↳ sudo docker ps

→ docker container inspect <temporary Container>
name

Q) what are the feature of underlying operating system?

A) Several components are needed for Linux containers to function correctly, most of them are provided by Linux kernel.



* kernel namespaces ensure process isolation
* Cgroups are employed to control the system resources.
* SELinux is used to assure separation b/w the host and the containers and also b/w the individual containers.
* management Interface forms a higher layer that interacts with the aforementioned kernal components and provides tools for construction and maganent of containers

* NAmespaces *

- Types of namespaces

(1) moont namespace: Isolate the set of file system mount points seen by a group of processes so that processes in different moont namespaces can have different views of the file system hierarchy

(2) UTS namespace: Isolate two system identifiers - nodename and domainname return by the name uname() system call.

(iii) IPC namespace: Isolate certain Interprocess communication (IPC) resources, such as System V IPC objects and POSIX message queues.

(iv) PID namespaces: allow process in different containers to have the same PID. So each container can have its own init (PID 1) process that manages various system initialization tasks as well as container life cycle.

→ `ps -e2 | grep systemd$`

(v) Network namespace: provide isolation of network controllers, system resources associated with networking, firewall and routing tables.

→ allows containers to use separate VN stacks, loopback devices and process space.

* Control Groups (Cgroup)

The kernel uses cgroup to group process of system resource. management, Cgroup allocate CPU time, System memory, network bandwidth, or combination of these among user-defined group of tasks. In Linux Cgroups are managed by systemd slice, scope, and service unit.

SELinux: provides Secure Separation of containers by applying SELinux policy and labels it. Integrates with virtual devices by using the sVirt technology.

* Secure Containers with SELinux:

From the Security point of view, there is a need of isolate the host system from a container and to isolate containers from each other. The kernel features used by containers mainly Cgroups and namespaces, by themselves provide a certain level of Security. Cgroup ensure that a single container cannot exhaust a large amount of system resources thus preventing some DDOS attacks.

- By virtue of namespace, the /dev directory created within containers is a private to each container.

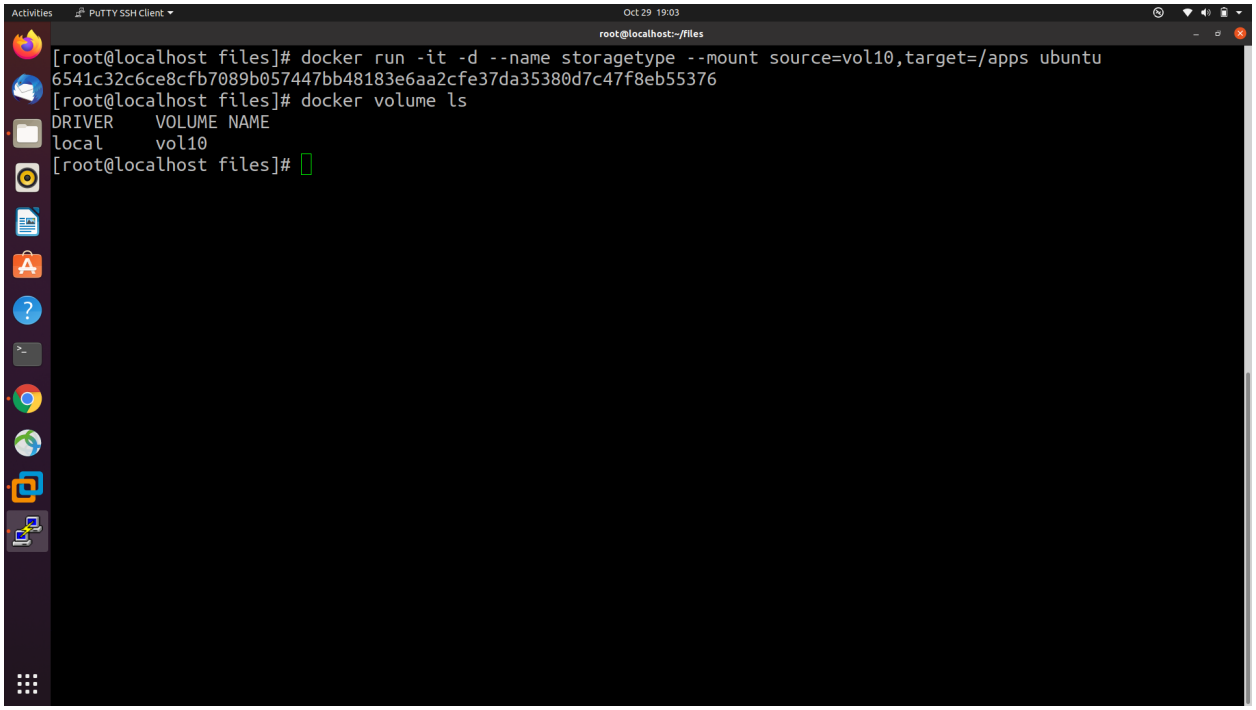
→ `ps -eZ | grep virtd_lxc_t`

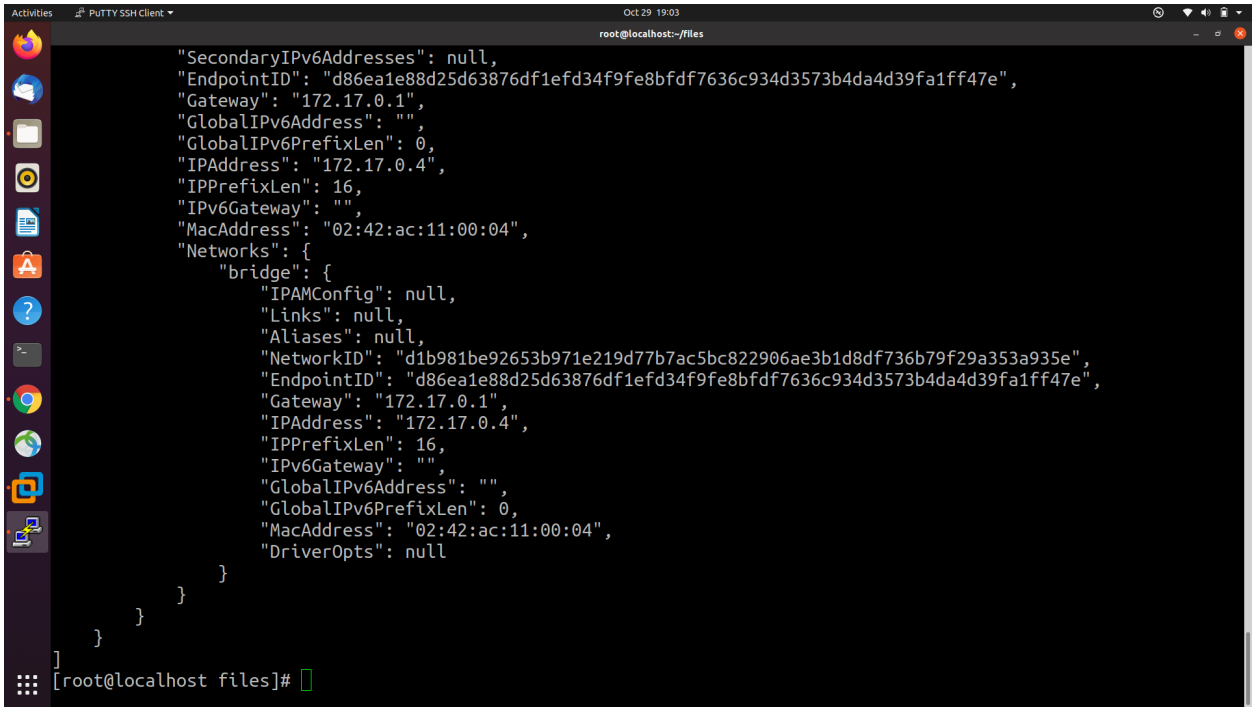
Note: You might note SELinux appears to be disabled inside the container even though it is running in enforce mode on the host system.

→ ~~getenforce~~ gen

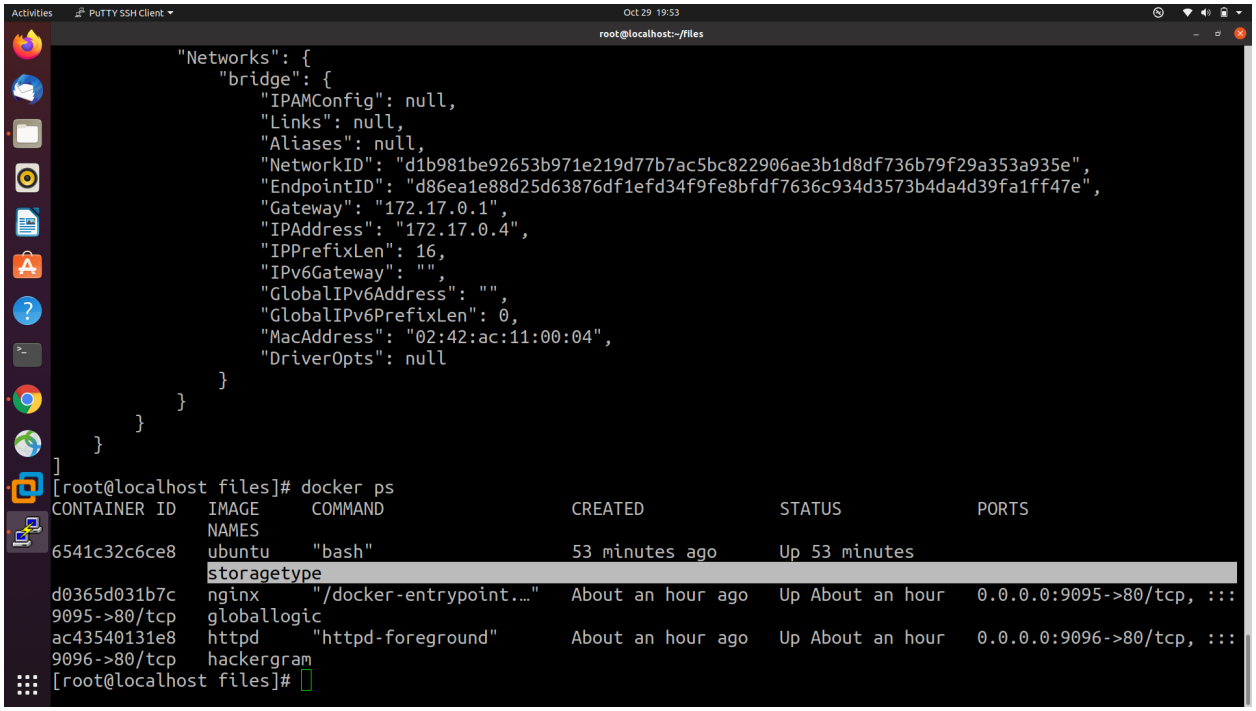
→ `getenforce` command on host and in the containers

This is to prevent utilities that have SELinux awareness such as `setenforce`, to perform any SELinux activity inside the container.


Activities
PuTTY SSH Client
Oct 29 19:03
root@localhost:~/files
[root@localhost files]# docker run -it -d --name storagetype --mount source=vol10,target=/apps ubuntu
6541c32c6ce8cfb7089b057447bb48183e6aa2cfe37da35380d7c47f8eb55376
[root@localhost files]# docker volume ls
DRIVER VOLUME NAME
local vol10
[root@localhost files]#


Activities
PuTTY SSH Client
Oct 29 19:03
root@localhost:~/files
"SecondaryIPv6Addresses": null,
"EndpointID": "d86ea1e88d25d63876df1efd34f9fe8bfdf7636c934d3573b4da4d39fa1ff47e",
"Gateway": "172.17.0.1",
"GlobalIPv6Address": "",
"GlobalIPv6PrefixLen": 0,
"IPAddress": "172.17.0.4",
"IPPrefixLen": 16,
"IPv6Gateway": "",
"MacAddress": "02:42:ac:11:00:04",
"Networks": {
"bridge": {
"IPAMConfig": null,
"Links": null,
"Aliases": null,
"NetworkID": "d1b981be92653b971e219d77b7ac5bc822906ae3b1d8df736b79f29a353a935e",
"EndpointID": "d86ea1e88d25d63876df1efd34f9fe8bfdf7636c934d3573b4da4d39fa1ff47e",
"Gateway": "172.17.0.1",
"IPAddress": "172.17.0.4",
"IPPrefixLen": 16,
"IPv6Gateway": "",
"GlobalIPv6Address": "",
"GlobalIPv6PrefixLen": 0,
"MacAddress": "02:42:ac:11:00:04",
"DriverOpts": null
}
}
}
}
]
[root@localhost files]#


Activities
PuTTY SSH Client
Oct 29 19:04
root@localhost:~/files
b7cd-init/diff:/var/lib/docker/overlay2/4a0904244161fdac55787af52f0efc5ec00bb4a21f0256f79ec601a9aef39049/diff",
"MergedDir": "/var/lib/docker/overlay2/7eed1eac2505502dd1932c11fc06141399d43df62a9286151517e2d9aa9
9b7cd/merged",
"UpperDir": "/var/lib/docker/overlay2/7eed1eac2505502dd1932c11fc06141399d43df62a9286151517e2d9aa99
b7cd/diff",
"WorkDir": "/var/lib/docker/overlay2/7eed1eac2505502dd1932c11fc06141399d43df62a9286151517e2d9aa99b
7cd/work"
},
"Name": "overlay2"
},
"Mounts": [
{
"Type": "volume",
"Name": "vol10",
"Source": "/var/lib/docker/volumes/vol10/_data",
"Destination": "/apps",
"Driver": "local",
"Mode": "z",
"RW": true,
"Propagation": ""
}
],
"Config": {
"Hostname": "6541c32c6ce8",
"Domainname": "",
"User": "",
"AttachStdin": false,
"AttachStdout": false,
"AttachStderr": false,
"Tty": true,


Activities
PuTTY SSH Client
Oct 29 19:53
root@localhost:~/files
"Networks": {
"bridge": {
"IPAMConfig": null,
"Links": null,
"Aliases": null,
"NetworkID": "d1b981be92653b971e219d77b7ac5bc822906ae3b1d8df736b79f29a353a935e",
"EndpointID": "d86ea1e88d25d63876df1efd34f9fe8bfdf7636c934d3573b4da4d39fa1ff47e",
"Gateway": "172.17.0.1",
"IPAddress": "172.17.0.4",
"IPPrefixLen": 16,
"IPv6Gateway": "",
"GlobalIPv6Address": "",
"GlobalIPv6PrefixLen": 0,
"MacAddress": "02:42:ac:11:00:04",
"DriverOpts": null
}
}
}
}
]
[root@localhost files]# docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS
NAMES
6541c32c6ce8 ubuntu "bash" 53 minutes ago Up 53 minutes
storagetype
d0365d031b7c nginx "/docker-entrypoint.…" About an hour ago Up About an hour 0.0.0.0:9095->80/tcp, :::
9095->80/tcp globallogic
ac43540131e8 httpd "httpd-foreground" About an hour ago Up About an hour 0.0.0.0:9096->80/tcp, :::
9096->80/tcp hackergram
[root@localhost files]#