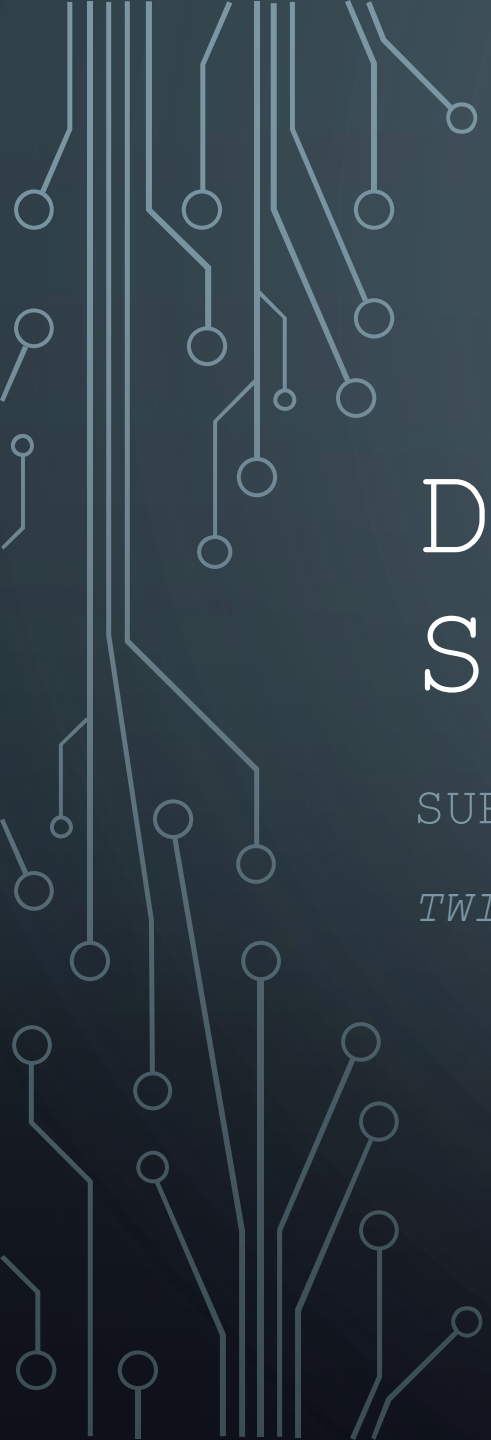


DOCKER CONTAINER SECURITY

SURAJ KHETANI

TWITTER - @R00TREAVER

CONTENTS

WHAT EVERY DEVELOPER SAYS

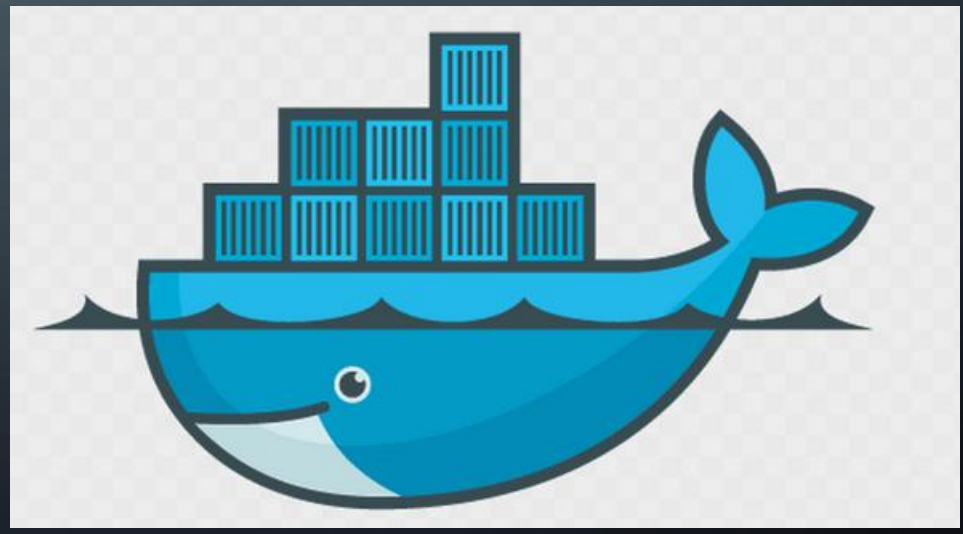
code works fine locally.
so you go upload. test. doesnt work.
Every f'in time.

WHAT IS DOCKER

Docker is a tool designed to make it easier to create, deploy, and run applications by using containers. Containers allow a developer to package up an application with all of the parts it needs, such as libraries and other dependencies, and ship it all out as one package.

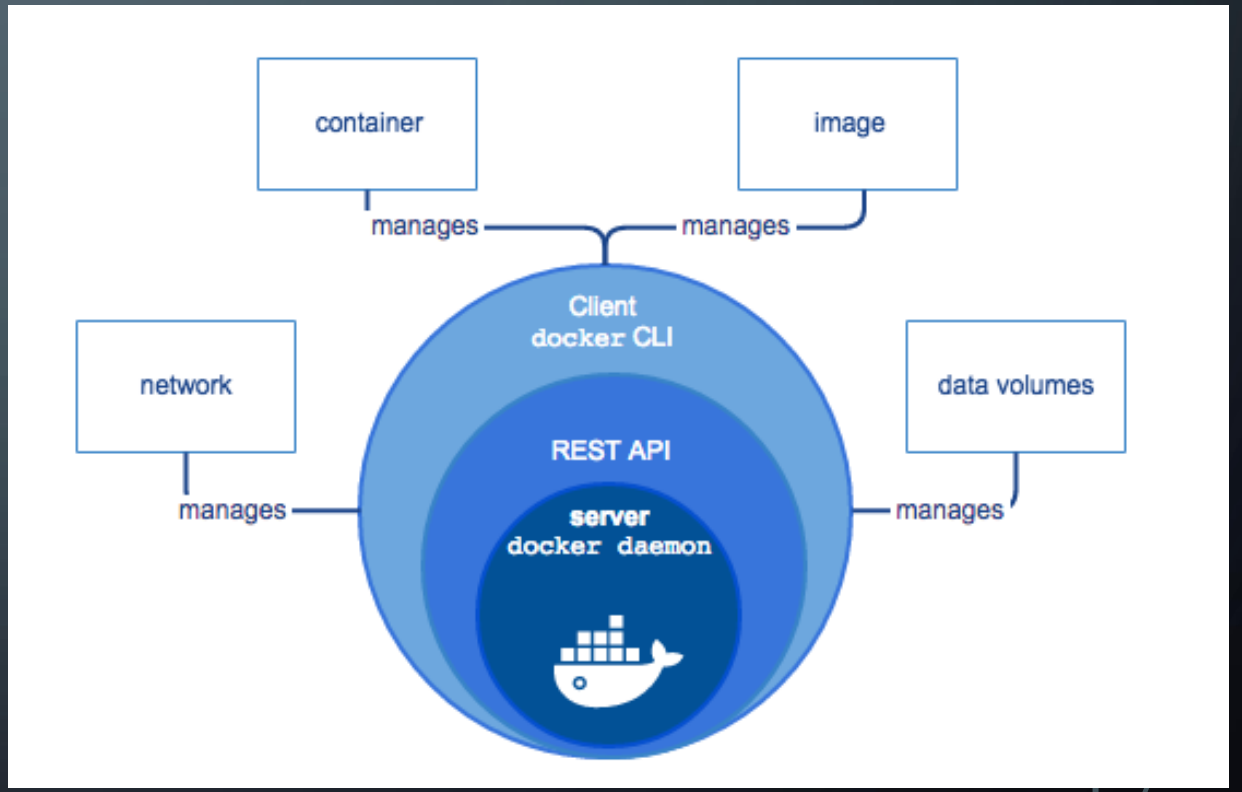
- Docker is currently the only ecosystem providing the full package:
  - Image management
  - Resource Isolation
  - File System Isolation
  - Network Isolation
  - Change Management
  - Process Management

Source: https://medium.com/@yannmjl/what-is-docker-in-simple-english-a24e8136b90b



BASICS OF DOCKER

- Docker Engine is a client-server application with these major components:
  - A CLI client (Docker)
  - A REST API
  - A server called the daemon process



A BRIEF HISTORY OF CONTAINERS

A BRIEF HISTORY OF CONTAINERS

1979 Unix V7

- During the development of Unix V7 in 1979, the **chroot** system call was introduced, changing the root directory of a process and its children to a new location in the filesystem. This advance was the beginning process isolation: segregating file access for each process. Chroot was added to **BSD** in 1982.

2000 FreeBSD Jails

- FreeBSD Jails allows administrators to partition a **FreeBSD** computer system into several independent, smaller systems – called "jails" – with the ability to assign an IP address for each system and configuration.
- Similar Jail was introduced in Linux VServer in 2001.

2004 Solaris Containers

- Combines system resource controls and boundary separation provided by zones, which were able to leverage features like snapshots and cloning from ZFS.

Source: https://blog.aquasec.com/a-brief-history-of-containers-from-1970s-chroot-to-docker-2016

A BRIEF HISTORY OF CONTAINERS [CONTD.]

2006
Process Containers

- It was designed for limiting, accounting and isolating resource usage (CPU, memory, disk I/O, network) of a collection of processes. It was renamed "Control Groups (cgroups)" a year later and eventually merged to Linux kernel 2.6.24.

2008
Linux Containers

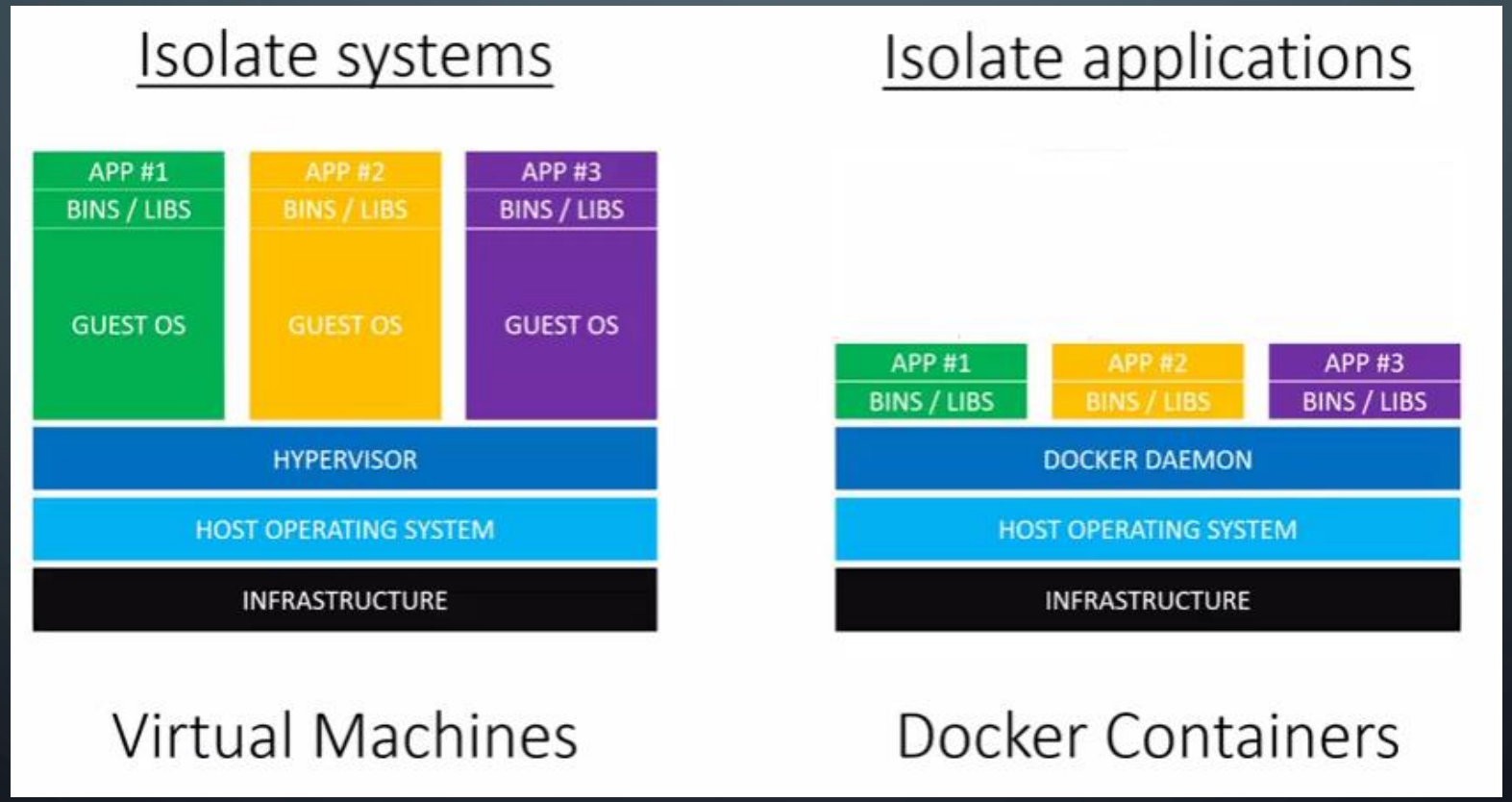
- The most complete implementation of Linux container manager. It was implemented using cgroups and Linux namespaces, and it works on a single Linux kernel without requiring any patches.

2013
Docker

- Docker used LXC in its initial stages and later replaced that container manager with its own library, libcontainer. But there's no doubt that Docker separated itself from the pack by offering an entire ecosystem for container management.

Source: https://blog.aquasec.com/a-brief-history-of-containers-from-1970s-chroot-to-docker-2016

CONTAINER VS VIRTUAL MACHINES



CONTAINER VS VIRTUAL MACHINES

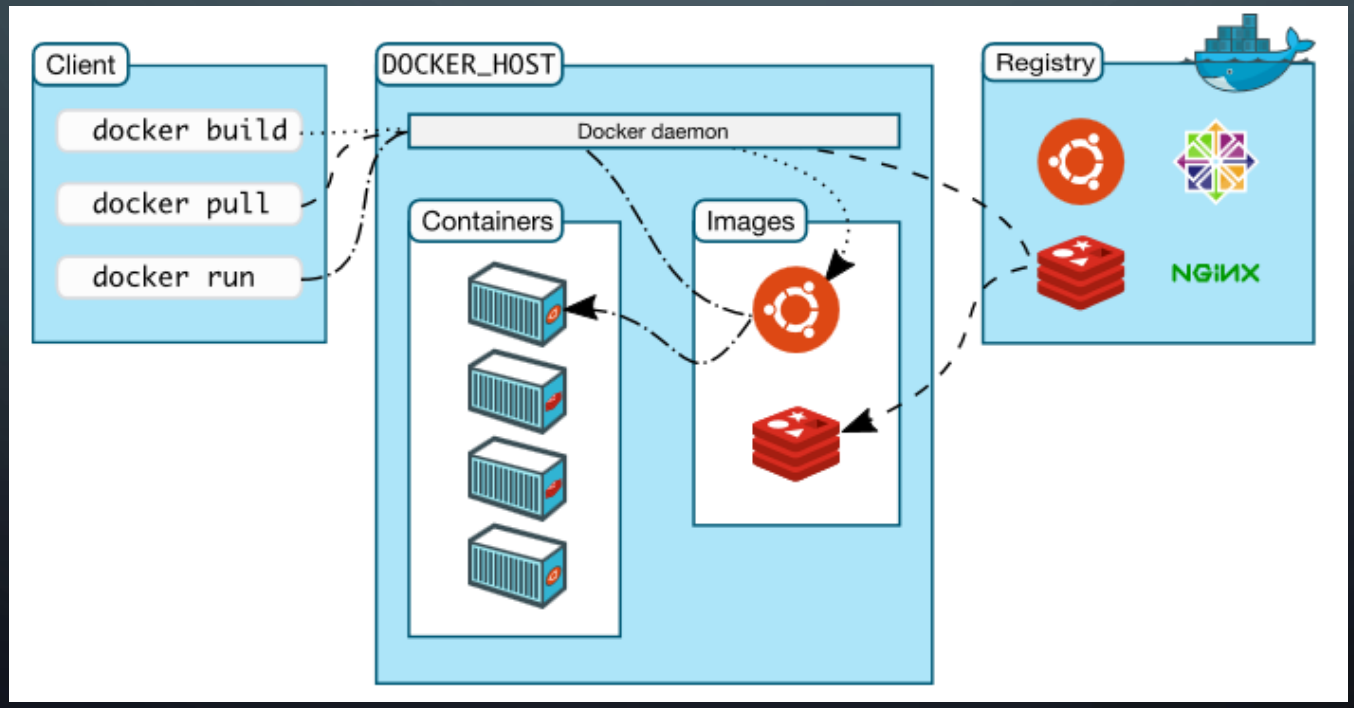
Average Start/Stop Times

Technology	Start Time	Stop Time
Docker Containers	< 50 ms	< 50 ms
Virtual Machines	30-45 sec	5-10 sec

Source: https://runnable.com/docker/why-use-docker

DOCKER ARCHITECTURE

- **The Docker client -** primary way that many Docker users interact with Docker
- **The Docker daemon -** listens for Docker API requests and manages Docker objects such as images, containers, networks, and volumes.
- **Docker registries -** A Docker *registry* stores Docker images. Eg: Docker Hub and Docker Cloud



DOCKER ARCHITECTURE

- **Docker objects**
  - **Images -** An *image* is a read-only template with instructions for creating a Docker container. To build your own image, you create a *Dockerfile*.
  - **Containers -** A container is a runnable instance of an image.
  - **Services -** Services allow you to scale containers across multiple Docker daemons, which all work together as a *swarm* with multiple *managers* and *workers*. By default, the service is load-balanced across all worker nodes.

DEMO – CREATING AND RUNNING DOCKER CONTAINERS

DEMO 1 - CREATING MY FIRST DOCKER IMAGE

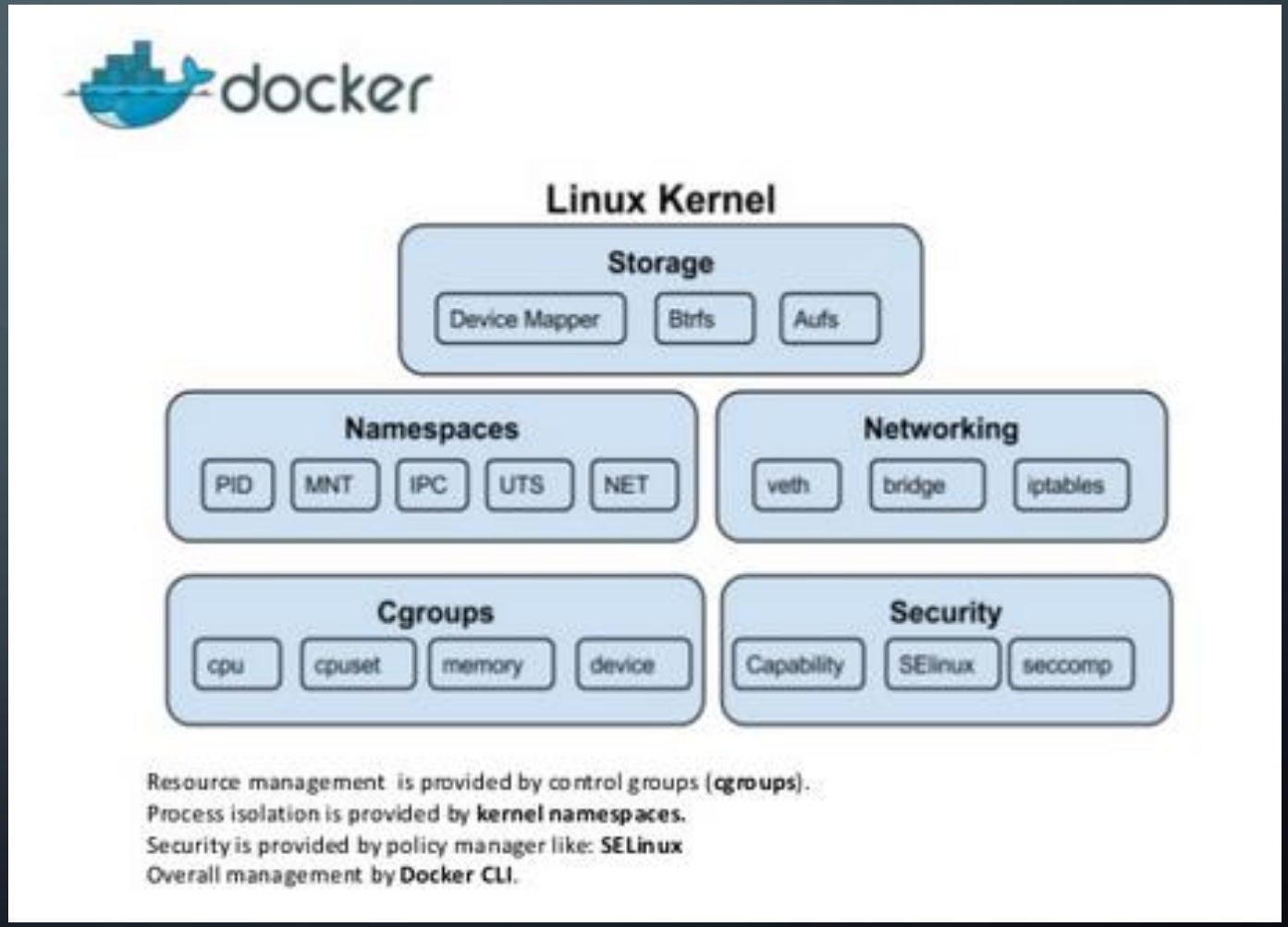
DEMO 2 - RUNNING MY FIRST DOCKER CONTAINER

BUILDING AND RUNNING DOCKER CONTAINERS

- Create *Dockerfile*
- Build the Docker image – docker build .
- Turns Docker image to container – docker run <image-id>
- Other ways to run containers:
  - Pull images from docker repo – docker pull <image-id>
  - Run the image: docker run <image-id>

DOCKER INTERNALS AND FEATURES

- Namespaces
- Control Groups
- Security
  - Capability
  - SELinux
  - seccomp



NAMESPACES

- Network Namespace – when containers are launched, a unique network interface and IP address is created.
  - docker run -it alpine ip addr show
- By changing the namespace to host, the container will share the same network interface and IP address of the host machine
  - docker run -it --net=host alpine ip addr show
- By changing the namespace to the host, the container can also see all other system processes running on the operating system
  - docker run -it --pid=host alpine ps aux

NAMESPACES

- docker run -it alpine ip addr show

```
suraj@rootreaver:~$ sudo docker run -it alpine ip addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
101: eth0@if102: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue state UP
    link/ether 02:42:ac:11:00:0e brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.14/16 brd 172.17.255.255 scope global eth0
       valid_lft forever preferred_lft forever
suraj@rootreaver:~$
```

NAMESPACES

- By changing the namespace to host, the container will share the same network interface and IP address of the host machine
  - docker run -it --net=host alpine ip addr show

```
suraj@rootreaver:~$ sudo docker run -it --net=host alpine ip addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc fq_codel state DOWN qlen 1000
    link/ether 18:db:f2:37:33:fe brd ff:ff:ff:ff:ff:ff
3: wlan0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP qlen 1000
    link/ether f0:d5:bf:a3:22:ad brd ff:ff:ff:ff:ff:ff
    inet 192.168.43.28/24 brd 192.168.43.255 scope global dynamic wlan0
       valid_lft 2288sec preferred_lft 2288sec
    inet6 fe80::68d9:9d81:bc83:56a3/64 scope link
       valid_lft forever preferred_lft forever
4: vmnet1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UNKNOWN qlen 1000
    link/ether 00:50:56:c0:00:01 brd ff:ff:ff:ff:ff:ff
    inet 172.16.176.1/24 brd 172.16.176.255 scope global vmnet1
       valid_lft forever preferred_lft forever
    inet6 fe80::250:56ff:fec0:1/64 scope link
       valid_lft forever preferred_lft forever
5: vmnet8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UNKNOWN qlen 1000
    link/ether 00:50:56:c0:00:08 brd ff:ff:ff:ff:ff:ff
    inet 172.16.43.1/24 brd 172.16.43.255 scope global vmnet8
       valid_lft forever preferred_lft forever
    inet6 fe80::250:56ff:fec0:8/64 scope link
       valid_lft forever preferred_lft forever
6: docker_gwbridge: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
    link/ether 02:42:a3:f7:55:51 brd ff:ff:ff:ff:ff:ff
    inet 172.18.0.1/16 brd 172.18.255.255 scope global docker_gwbridge
       valid_lft forever preferred_lft forever
    inet6 fe80::42:a3ff:fef7:5551/64 scope link
       valid_lft forever preferred_lft forever
7: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
    link/ether 02:42:db:40:20:3b brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:dbff:fe40:203b/64 scope link
       valid_lft forever preferred_lft forever
```

NAMESPACES

- docker run -it alpine ps aux

```
suraj@rootreaver:~$ sudo docker run -it alpine ps aux
PID   USER     TIME  COMMAND
    1 root      0:00 ps aux
```

NAMESPACES

- By changing the namespace to the host, the container can also see all other system processes running on the operating system
  - docker run -it --pid=host alpine ps aux

```
suraj@rootreaver:~$ sudo docker run -it --pid=host alpine ps aux
PID   USER     TIME  COMMAND
    1 root      0:06 {systemd} /sbin/init splash
    2 root      0:00 [kthreadd]
    4 root      0:00 [kworker/0:0H]
    6 root      0:00 [mm_percpu_wq]
    7 root      0:00 [ksoftirqd/0]
    8 root      0:09 [rcu_sched]
    9 root      0:00 [rcu_bh]
   10 root      0:00 [migration/0]
   11 root      0:00 [watchdog/0]
   12 root      0:00 [cpuhp/0]
   13 root      0:00 [cpuhp/1]
   14 root      0:00 [watchdog/1]
   15 root      0:00 [migration/1]
   16 root      0:00 [ksoftirqd/1]
   18 root      0:00 [kworker/1:0H]
   19 root      0:00 [cpuhp/2]
   20 root      0:00 [watchdog/2]
   21 root      0:00 [migration/2]
   22 root      0:00 [ksoftirqd/2]
   24 root      0:00 [kworker/2:0H]
   25 root      0:00 [cpuhp/3]
```

CGROUPS

- Control the resource utilization and keep a limit on the memory CPUs etc.
  - docker run -d --name wordpress --memory 100m alpine top
  - This would allow up to 100mb to the wordpress container
- Similarly --cpu-shares can be used to set a cap on cpu resource utilization
- docker stats --no-stream to verify the above implemented configuration

CGROUPS

- sudo docker run --name unrestricted-mem -d myfirstimage

```
suraj@rootreaver:~/myfirstimage$ sudo docker run --name unrestricted-mem -d myfirstimage
13cf5d77f6da1b4c8c822385c2b3782d8ef030fa5ea9048786a32546b1fb87
suraj@rootreaver:~/myfirstimage$ sudo docker stats --no-stream
CONTAINER ID        NAME                CPU %               MEM USAGE / LIMIT     MEM %               NET I/O             BLOCK I/O           PIDS
13cf5d77f6da        unrestricted-mem    0.02%               16.53MiB / 15.55GiB   0.10%               2.37kB / 0B         0B / 0B             1
9d635f0b1043        hopeful_raman       0.02%               16.46MiB / 15.55GiB   0.10%               3.18kB / 0B         0B / 0B             1
5fcdadca431d        gracious_wozniak    0.00%               1.438MiB / 15.55GiB   0.01%               4.2kB / 0B          73.7kB / 0B         2
719ce381d39b        dvna                0.00%               57.39MiB / 15.55GiB   0.36%               57.4MB / 928kB      17MB / 213kB        32
78a3de76409e        gracious_bell       0.00%               50.11MiB / 15.55GiB   0.31%               987kB / 6.64kB      0B / 131kB          21
e4af59943da0        determined_volhard  0.00%               49.93MiB / 15.55GiB   0.31%               1MB / 15.7kB        0B / 131kB          21
f871c8b0523d        epic_nightingale    0.00%               53.02MiB / 15.55GiB   0.33%               988kB / 7.99kB      85.3MB / 131kB      21
6d46c5d8eaf5        cranky_antonelli    0.10%               93.14MiB / 15.55GiB   0.58%               4.85kB / 0B         5.68MB / 331MB      37
b51f7471bf4f        jolly_germain       0.11%               93.15MiB / 15.55GiB   0.58%               4.85kB / 0B         262kB / 336MB       37
9ca4e1f31afb        pensive_jackson     0.11%               93MiB / 15.55GiB      0.58%               4.85kB / 0B         573kB / 329MB       37
9e14110eba3f        inspiring_curie     0.02%               16.54MiB / 100MiB     16.54%              4.85kB / 0B         0B / 0B             1
7df405cd5325        unruffled_lichterman 0.02%              16.41MiB / 15.55GiB   0.10%               5.15kB / 0B         0B / 0B             1
1afe220bbc1e        loving_sinoussi     0.02%               17.59MiB / 15.55GiB   0.11%               7.54kB / 2.18kB     0B / 0B             1
935ce91d1263        vibrant_benz        0.02%               17.61MiB / 15.55GiB   0.11%               7.93kB / 2.23kB     0B / 0B             1
658512d2daf4        wordpress_db_1      0.09%               179.6MiB / 15.55GiB   1.13%               7.33kB / 654B       262kB / 34.8MB      28
```

CGROUPS

- Control the resource utilization and keep a limit on the memory CPUs etc.
  - docker run -d --name restricted-mem --memory 100m myfirstimage
  - This would allow up to 100mb to the myfirstimage container

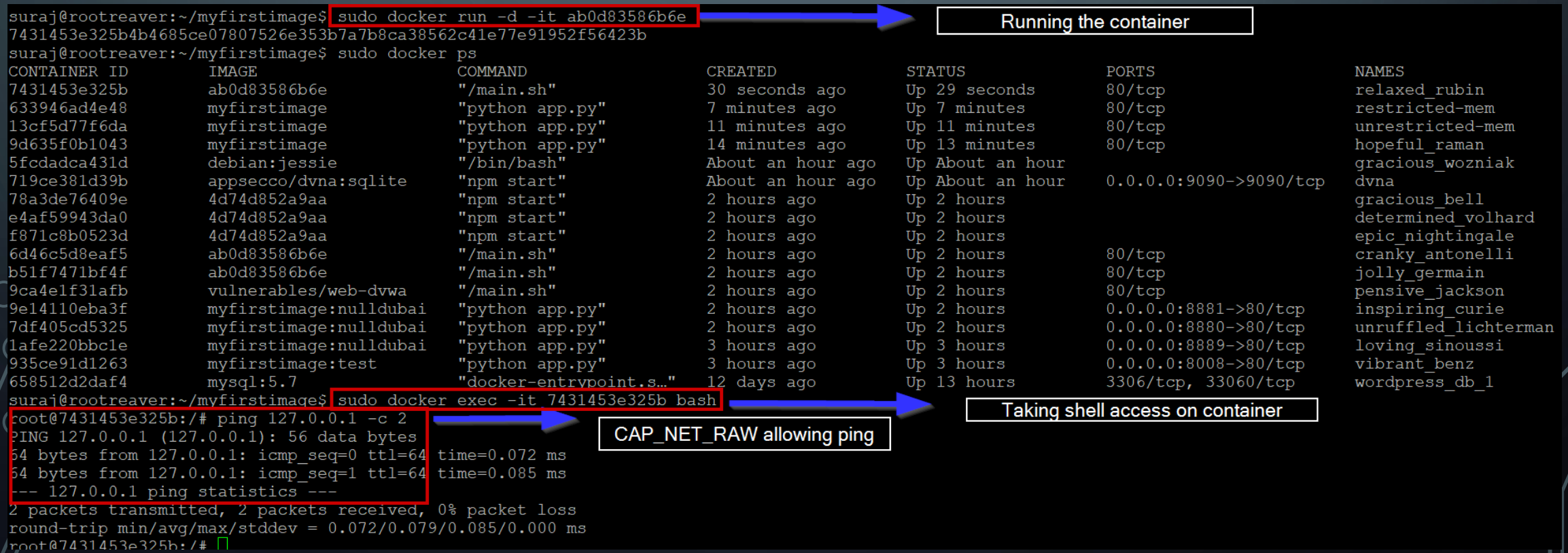
```
suraj@rootreaver:~/myfirstimage$ sudo docker run -d --name restricted-mem --memory 100m myfirstimage
WARNING: Your kernel does not support swap limit capabilities or the cgroup is not mounted. Memory limited without swap.
633946ad4e486898a76ea9a657ef0ebfba7345188d153d270f7f73dc4ee3e285
suraj@rootreaver:~/myfirstimage$ sudo docker stats --no-stream
CONTAINER ID        NAME                CPU %               MEM USAGE / LIMIT     MEM %               NET I/O             BLOCK I/O           PIDS
633946ad4e48        restricted-mem      0.02%               16.5MiB / 100MiB      16.50%              2.37kB / 0B         0B / 0B             1
13cf5d77f6da        unrestricted-mem    0.02%               16.52MiB / 15.55GiB   0.10%               3.18kB / 0B         0B / 0B             1
9d635f0b1043        hopeful_raman       0.02%               16.46MiB / 15.55GiB   0.10%               3.36kB / 0B         0B / 0B             1
5fcdadca431d        gracious_wozniak    0.00%               1.438MiB / 15.55GiB   0.01%               4.2kB / 0B          73.7kB / 0B         2
719ce381d39b        dvna                0.00%               57.39MiB / 15.55GiB   0.36%               57.4MB / 928kB      17MB / 213kB        32
78a3de76409e        gracious_bell       0.00%               50.11MiB / 15.55GiB   0.31%               987kB / 6.64kB      0B / 131kB          21
e4af59943da0        determined_volhard  0.00%               49.93MiB / 15.55GiB   0.31%               1MB / 15.7kB        0B / 131kB          21
f871c8b0523d        epic_nightingale    0.00%               53.02MiB / 15.55GiB   0.33%               988kB / 7.99kB      85.3MB / 131kB      21
6d46c5d8eaf5        cranky_antonelli    0.09%               93.14MiB / 15.55GiB   0.58%               4.85kB / 0B         5.68MB / 331MB      37
b51f7471bf4f        jolly_germain       0.11%               93.15MiB / 15.55GiB   0.58%               4.85kB / 0B         262kB / 336MB       37
9ca4e1f31afb        pensive_jackson     0.09%               93MiB / 15.55GiB      0.58%               4.85kB / 0B         573kB / 329MB       37
9e14110eba3f        inspiring_curie     0.02%               16.54MiB / 100MiB     16.54%              4.85kB / 0B         0B / 0B             1
7df405cd5325        unruffled_lichterman 0.02%              16.41MiB / 15.55GiB   0.10%               5.15kB / 0B         0B / 0B             1
1afe220bbc1e        loving_sinoussi     0.02%               17.59MiB / 15.55GiB   0.11%               7.61kB / 2.18kB     0B / 0B             1
935ce91d1263        vibrant_benz        0.02%               17.61MiB / 15.55GiB   0.11%               7.93kB / 2.23kB     0B / 0B             1
658512d2daf4        wordpress_db_1      0.06%               179.6MiB / 15.55GiB   1.13%               7.4kB / 654B        262kB / 34.8MB      28
suraj@rootreaver:~/myfirstimage$
```

SECURITY: CAPABILITIES

- Ability of the kernel to break down root privileges is Capability.
  - CAP_CHOWN – allows root user to make changes to file UIDs and GUIDs
  - CAP_DAC_OVERRIDE – allows roots user to bypass kernel permission on file read, write and execute
  - CAP_NET_RAW – used by ping command
- Drop capabilities – CAP_NET_RAW
  - sudo docker run --cap-drop NET_RAW -d -it ab0d83586b6e
  - sudo docker exec -it <container-id> sh

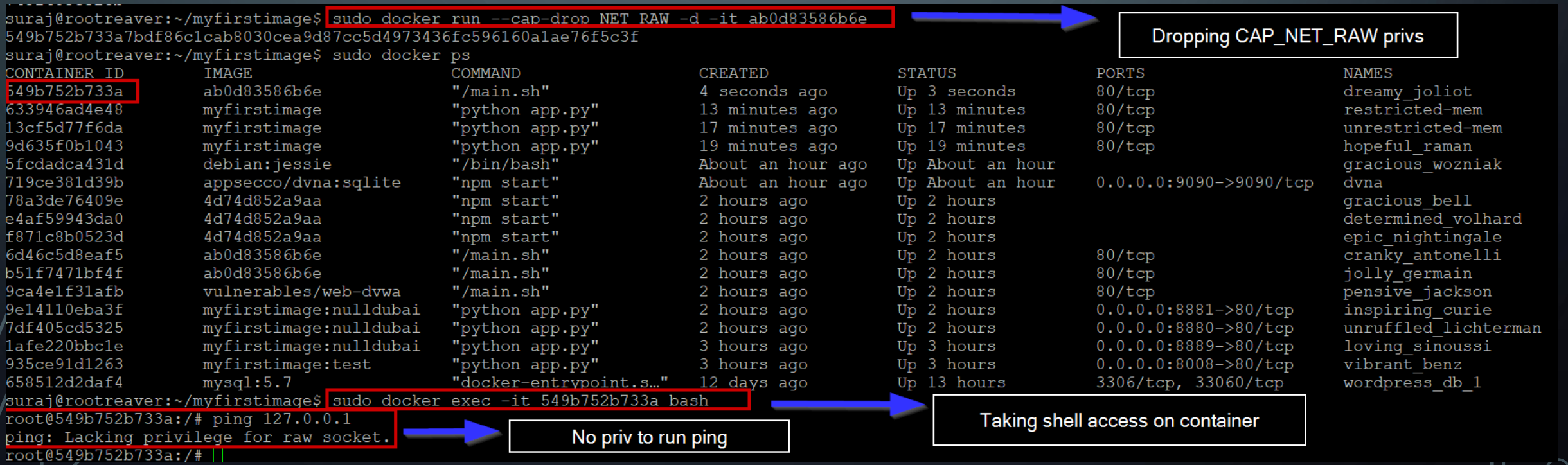
SECURITY: CAPABILITIES

- Before Dropping capabilities – CAP_NET_RAW
  - sudo docker run -d -it ab0d83586b6e
  - sudo docker exec -it <container-id> sh



SECURITY: CAPABILITIES

- Drop capabilities – CAP_NET_RAW
  - sudo docker run --cap-drop NET_RAW -d -it ab0d83586b6e
  - sudo docker exec -it <container-id> sh



SECURITY: SECCOMP

- SecComp defines which system calls should and should not be allowed to be executed by a container.
- They're defined in a JSON file that is applied when a container starts.

SECURITY: SECCOMP

- In this initial step we've defined seccomp permissions to disable allowing containers to run chmod, chown and chown32.
- Create json formatted file for defining seccomp policies

```
[root@host01 ~]# cat 1_chmod.json
{
    "defaultAction": "SCMP_ACT_ALLOW",
    "architectures": [
        "SCMP_ARCH_X86_64",
        "SCMP_ARCH_X86",
        "SCMP_ARCH_X32"
    ],
    "syscalls": [
        {
            "name": "chmod",
            "action": "SCMP_ACT_ERRNO",
            "args": []
        },
        {
            "name": "chown",
            "action": "SCMP_ACT_ERRNO",
            "args": []
        },
        {
            "name": "chown32",
            "action": "SCMP_ACT_ERRNO",
            "args": []
        }
    ]
}
```

SECURITY: SECCOMP

Running a container with the seccomp policy

```
[root@host01 ~]# docker run --rm -it \
>   --security-opt seccomp:1_chmod.json \
>   benhall/strace \
>   chmod 400 /etc/hostname
Unable to find image 'benhall/strace:latest' locally
latest: Pulling from benhall/strace
d0ca440e8637: Pull complete
e14a5bd01123: Pull complete
Digest: sha256:6c4d5e4752ae78f8ce68fafe8ddd207c7fb53a991bc6a8eca73bc36878c2d9c9
Status: Downloaded newer image for benhall/strace:latest
chmod: /etc/hostname: Operation not permitted
[root@host01 ~]#
```

ATTACKING COMMON SECURITY MISCONFIGURATIONS IN DOCKER

- Attacking insecure volume mounts
- Attacking container capabilities
- Attacking unauthenticated docker api

ATTACKING INSECURE VOLUME MOUNTS

- Demo

ATTACKING CONTAINER CAPABILITIES

- Demo

ATTACKING UNAUTHENTICATED DOCKER API

- Demo

DOCKER COMMAND CHEAT SHEET FOR ADMINS AND PENTESTERS

- service dockerd start – starts Docker daemon service
- docker ps – lists all running containers
- docker ps -a – lists all containers that have been stopped, running, created, etc
- docker run -name <*container-name*> -it <*image-name*>:<*tag*> /bin/bash – take an interactive tty shell inside a container
- docker log -f <*container-name*> - inspect docker logs
- docker inspect <*container-name*> or <*image-name*> -
- docker history <*container-name*> - lists changes done on the image
- docker network ls
- docker build <*dir*> .
- docker login
- docker secret ls
- docker commit c3f279d17e0a svendowideit/testimage:version3

NEXT TOPICS TO COVER

- Container Orchestration platform – Kubernetes and its (In)Security

REFERENCES AND FURTHER READING

- Attack demos inspired from Madhu Akulas' workshop from defcon
- https://www.katacoda.com
- https://docker.com
- http://docker-saigon.github.io/post/Docker-Internals/


PLEASE DON'T
ASK QUESTIONS
quickmeme.com