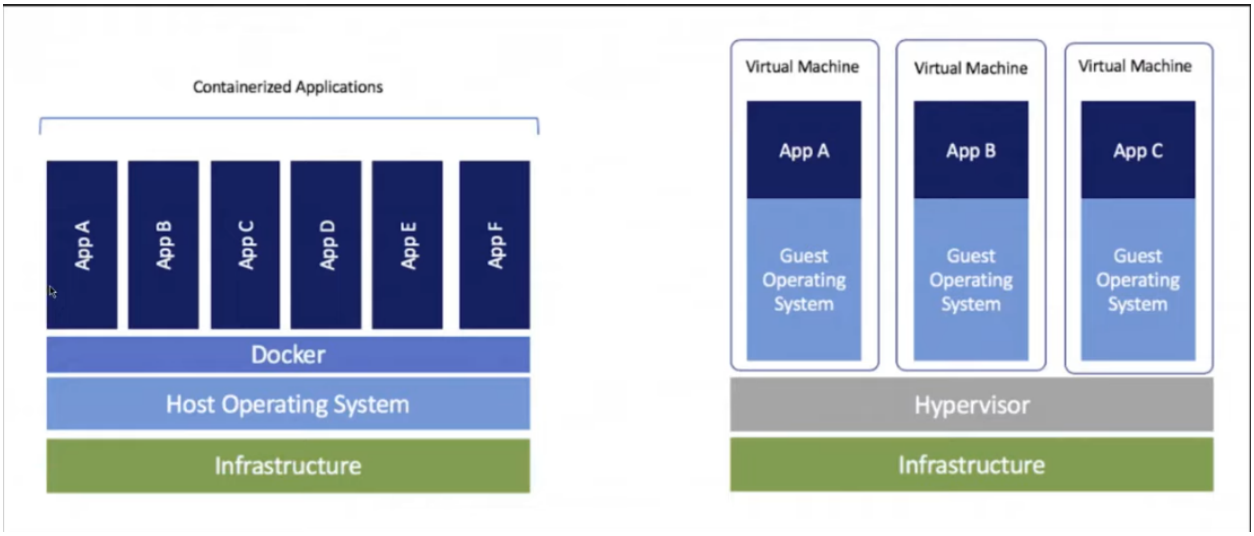
Docker Security Essentials

Link: https://www.youtube.com/watch?v=KINjI1tlo2w&list=PLBf0hzazHTGNv0-GVWZoveC49pIDHEHbn

Docker helps to build and deploy applications and services using a container.

They are better then using a VM because they use host operating system and kernel



Since the docker uses a host operating system, it should be primarily secured first.

A website similar to the OWASP top 10: https://www.cvedetails.com/vulnerabilities-by-types.php

Minimal operating systems can be run as the host operating systems for hosting docker applications. If using Linux we should use linux best security practices, have all the packages updated and upgraded. Few examples of such operating systems are: Clear Linux and Ubuntu Core

1. Practice of running the images with an unprivileged user

   Usually the docker image is run as the root user and we have to prevent this in production env. To prevent this we can create a low-privilege user.

   In a new Dockerfile, we can add:

```
FROM ubuntu:18.04

LABEL maintainer="Sam Sepiol"

RUN groupadd -r local_users && useradd -r -g local_users sathvik

#ENVIRONMENT VARIABLES
ENV HOME /home/sathvik
ENV DEBIAN_FRONTEND=noninteractive

~
```

Here a new group is created called `local_users` and the a user `sathvik` the is added to the group `local_users`.

This docker file should be run with the new user. The docker image need to be built first,

```
docker build . -t test
```

Then the docker image can be run by the low privilege user we just created in the new users.

```
docker run -u sathvik -it --rm <image id> /bin/bash
```

Here the user can still escalate privileges and gain access the root user and cause harm or make the system more vulnerable. So we remove the access to the root login even if the user have credentials for the application by changing the shell.

```
FROM ubuntu:18.04

LABEL maintainer="Sam Sepiol"

RUN groupadd -r local_users && useradd -r -g local_users sathvik
RUN chsh -s /usr/sbin/nologin root

#ENVIRONMENT VARIABLES
ENV HOME /home/sathvik
ENV DEBIAN_FRONTEND=noninteractive

~
```

Similar to the previous step build the image and then run it

```
docker build . -t test
```

```
docker run -u sathvik -it --rm <image id> /bin/bash
```

2. Not running docker container in privileged mode.
    a. A docker image can be run in privileged mode by

    ```
    docker run -it --rm --privileged <container id> /bin/bash
    ```

    b. Why even run a docker image in privileged mode?

    A Docker image running on privileged mode is a Docker image that has been granted root access to the host system. This means that the image can access and modify all files and devices on the host system, including the kernel.

    Privileged mode is typically used for running special-purpose containers, such as those that need to access hardware devices or perform other privileged operations. For example, a container that needs to run a Docker daemon or a Kubernetes master node would need to be running in privileged mode.

    c. We can prevent it by adding a `no-new-privilege` to the options for the docker image.

    ```
    docker run -it --rm --security-opt=no-new-privileges <container id>
    ```

    d. We can run the image with the low privileged user to reduce the vulnerabilities

    ```
    docker run -it -u sathvik --rm --security-opt=no-new-privileges <container id> /bin/bash
    ```

    e. We can add and remove capabilities to a docker image when it runs:

    ```
    docker run -u sathvik -it --rm --cap-drop all --cap-add <CAPABILITY> <image id> /bin/bash
    ```

    Capabilities of a Linux system: https://man7.org/linux/man-pages/man7/capabilities.7.html

3. Prevent docker from writing to the file system (Read-Only Mode)
    a. What is `--read-only` ?

    Read-only in a container means that the container's root filesystem cannot be modified. This means that the container cannot create, edit, or delete files in its root filesystem.

    b. An image can be run in read-only mode by the following command:

    ```
    docker run -u sathvik --read-only -it --rm <container id> /bin/bash
    ```

c. Even as a root user any modifications, addition and deletion can’t happen in the docker container.

d. If the application within the container is dynamic(the application keeps modifying the data) then we can specify a temporary file system where the application can perform the changes. Only the root user can perform the changes in the temp file system.

```
docker run -it --rm --read-only --tmpfs /opt <container id> /bin/bash
```

4. Disabling Inter-container communication

    a. Isolation of docker container could be done to use it with different networks like: Bridged, host etc.

    b. To check the type of networks available for the docker, check:

    ```
    docker network ls
    ```

    By default all the containers use a bridged connection.

    c. To inspect about the briged network and see which subnet the bridge network lies on we can check it by:

    ```
    docker network inspect bridge
    ```

```
└─$ docker network inspect bridge
[
    {
        "Name": "bridge",
        "Id": "182fd300a7e4799b0a7d3782b8359e2eed0cb94ee873e4acc5f6c47db6760eef",
        "Created": "2023-10-08T17:37:00.409694448+05:30",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "172.17.0.0/16"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {},
        "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": {}
    }
]
```

In the options we have `enable_icc`, ICC means inter-container communication. To change this we need to create out own network.

d. Disabling inter-container communication

   i. Create a new bridged network

```
docker network create --driver bridge -o "com.docker.network.bridge.enable_icc="false" test-network
```

   ii. Check for any errors and if the network is added

```
docker network ls
```

   iii. Running a docker image with specific network configuration

```
docker run -it --network test-net <container id> /bin/bash
```

4. Auditing tools
    1. Docker bench security: https://github.com/docker/docker-bench-security