What you don't know about containers *(and why it matters)*

Sean Scott

DOAG CloudLand 2024
Brühl, Germany
June 20, 2024

Sean Scott

Oracle ACE Director

Data on Kubernetes Community Ambassador

Managing Principal Consultant
Viscosity North America

Database Reliability Engineering

Business Continuity ·:· HA & DR

Automation ·:· Observability

Real Application Clusters ·:· Data Guard ·:· Sharding

Containerization ·:· Terraform ·:· Ansible

Exadata & Engineered Systems

AHF ·:· TFA ·:· GIMR ·:· CHA

Oracle ACE Program

500+ technical experts helping peers globally

The **Oracle ACE Program** recognizes and rewards community members for their technical and community contributions to the Oracle community

3 membership tiers

Oracle ACE Director

Oracle ACE Pro

Oracle ACE Associate

Oracle ACE

Nominate

yourself or someone you know:

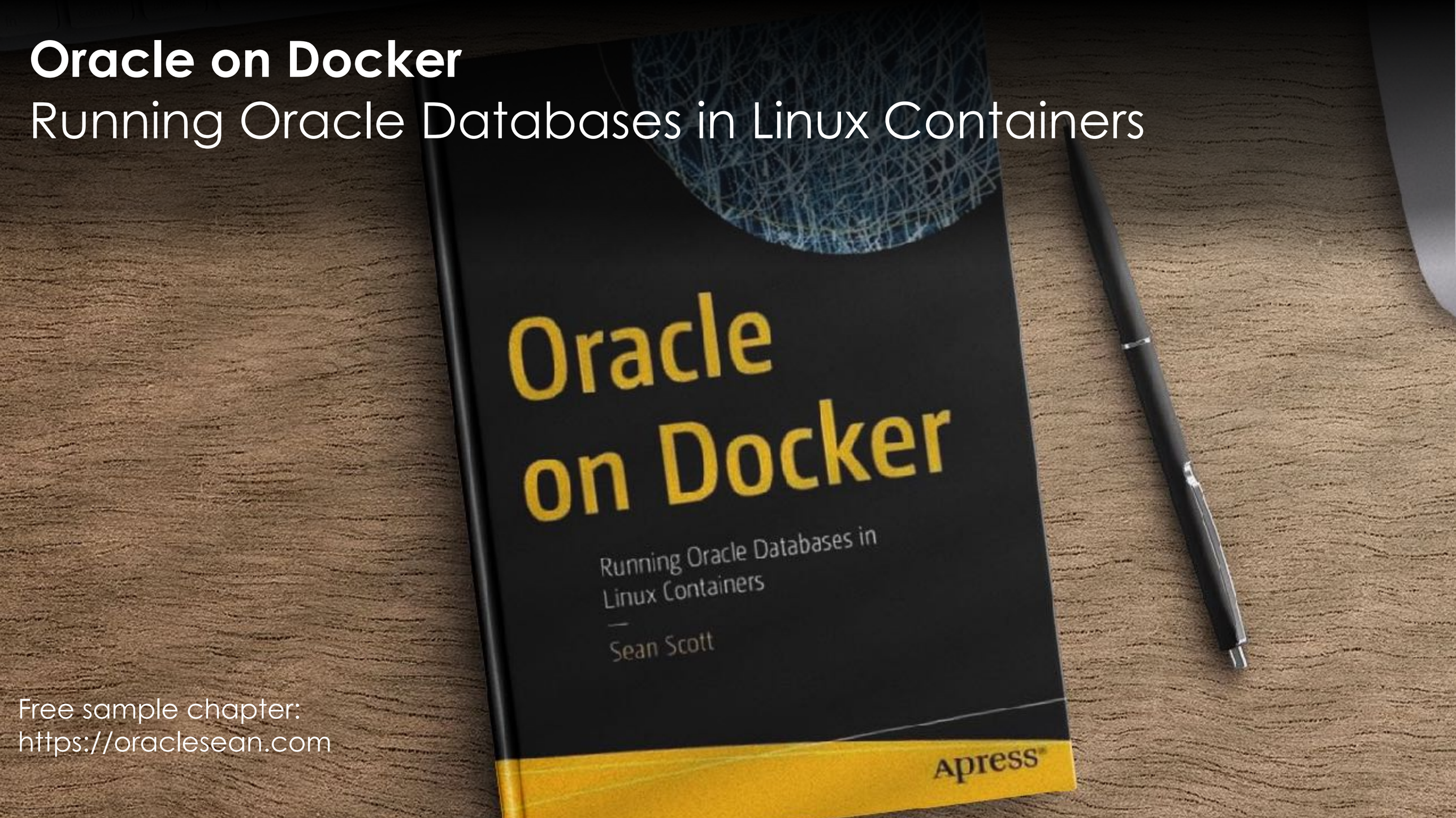
ace.oracle.com/nominate

For more details on Oracle ACE Program:
ace.oracle.com

Connect: **aceprogram_ww@oracle.com** Facebook.com/OracleACEs @oracleace

Oracle on Docker

Running Oracle Databases in Linux Containers



Free sample chapter:
https://oraclesean.com

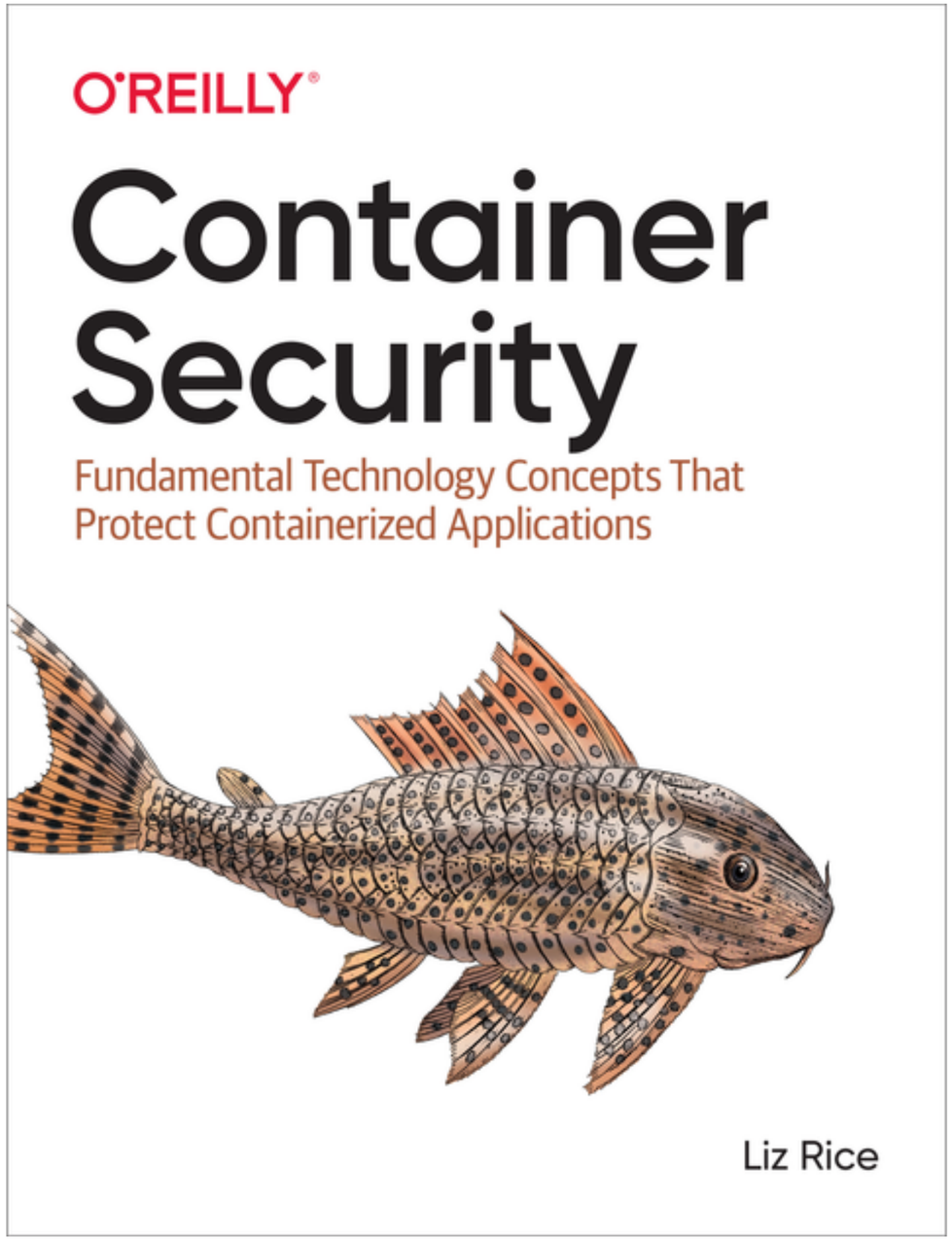
Lab Preparation

You will need:

- A Linux environment
- `sudo` permission (to `mount/umount`)
- Three (or more) terminal windows
- Optional: Docker/Podman

Copy/pull code examples and slides:

https://github.com/oraclesean/cloudland24



Container Security

Liz Rice

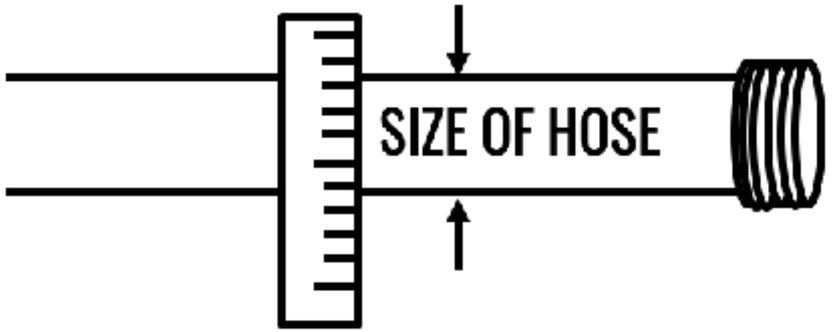
What Are Containers?

Containers are like lightweight Virtual Machines, right?

Electricity is like a water hose

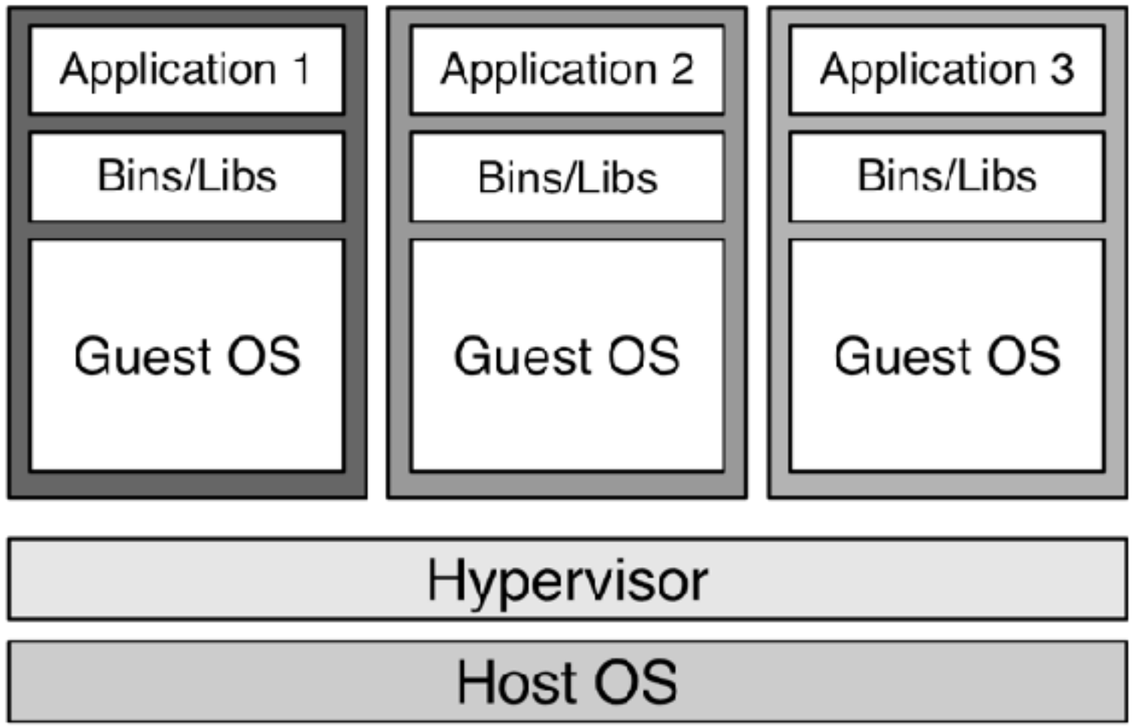
Voltage — Volts (V)

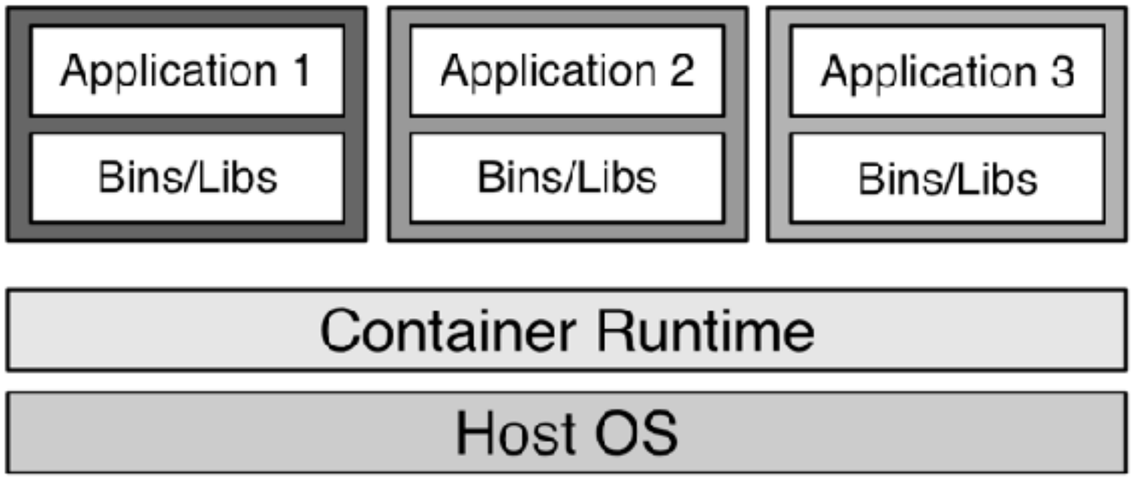
Current — Amps (A or I)



Resistance — Ohms (R or Ω)

FREEING ENERGY





Three full operating systems

Three application/executable directories

13 Oracle Databases on a Laptop

```
# dps
NAMES       IMAGE                          PORTS                                  NETWORKS    STATE
ORCL112     oraclesean/db:11.2.0.4-EE      1521/tcp, 5500/tcp, 8080/tcp           oracle-db   running
ORCL121     oraclesean/db:12.1-EE          1521/tcp, 5500/tcp, 8080/tcp           oracle-db   running
ORCL122     oraclesean/db:12.2-EE          1521/tcp, 5500/tcp, 8080/tcp           oracle-db   running
ORCL1914    oraclesean/db:19.14-EE         1521/tcp, 5500/tcp, 8080/tcp           oracle-db   running
ORCL19151   oraclesean/db:19.15.1-EE       1521/tcp, 5500/tcp, 8080/tcp           oracle-db   running
ORCL19152   oraclesean/db:19.15.2-EE       1521/tcp, 5500/tcp, 8080/tcp           oracle-db   running
ORCL1915    oraclesean/db:19.15-EE         1521/tcp, 5500/tcp, 8080/tcp           oracle-db   running
ORCL19161   oraclesean/db:19.16.1-EE       1521/tcp, 5500/tcp, 8080/tcp           oracle-db   running
ORCL1916    oraclesean/db:19.16-EE         1521/tcp, 5500/tcp, 8080/tcp           oracle-db   running
ORCL1917    oraclesean/db:19.17-EE         1521/tcp, 5500/tcp, 8080/tcp           oracle-db   running
ORCL216     oraclesean/db:21.6-EE          1521/tcp, 5500/tcp, 8080/tcp           oracle-db   running
ORCL217     oraclesean/db:21.7-EE          1521/tcp, 5500/tcp, 8080/tcp           oracle-db   running
ORCL218     oraclesean/db:21.8-EE          1521/tcp, 5500/tcp, 8080/tcp           oracle-db   running
```

2018 MacBook Pro 15", 2.2GHz 6-core Intel Core i7, 16GB, 1TB SSD

Virtual Machines: Bootable OS, often heavy

Containers: Support one application or service

Virtual Machines: Bootable OS, often heavy

A 100-page book of games & puzzles

Containers: Support one application or service

Virtual Machines: Bootable OS, often heavy

A 100-page book of games & puzzles

Containers: Support one application or service

A sheet of paper with a Tic-Tac-Toe grid

Games have structured playing surfaces and rules.

Services deliver games to players.

Images include the
game rules and playing surface.

Images are a *minimal filesystem & metadata*

for running a service.

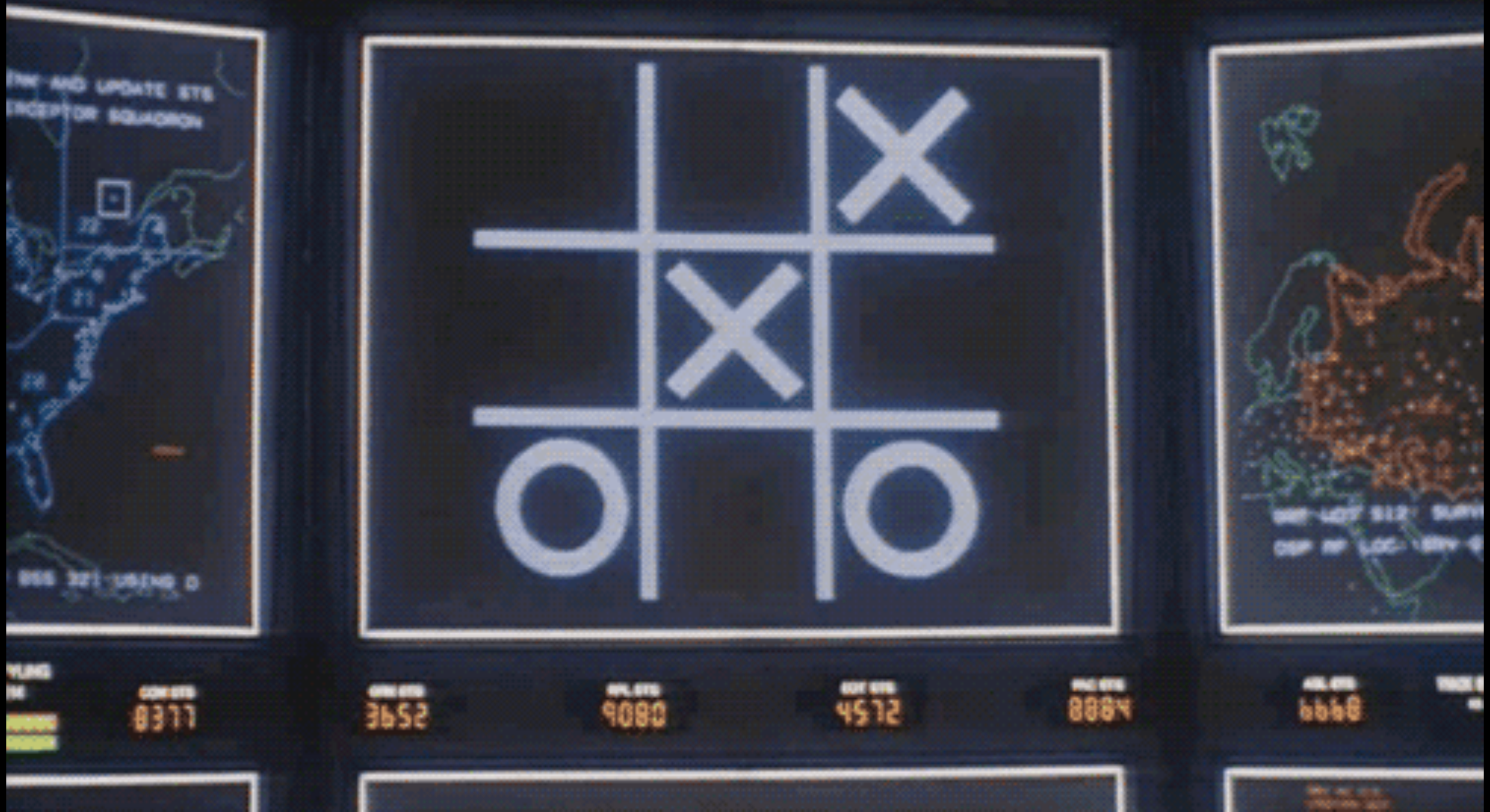
Running an image starts a **Container**.

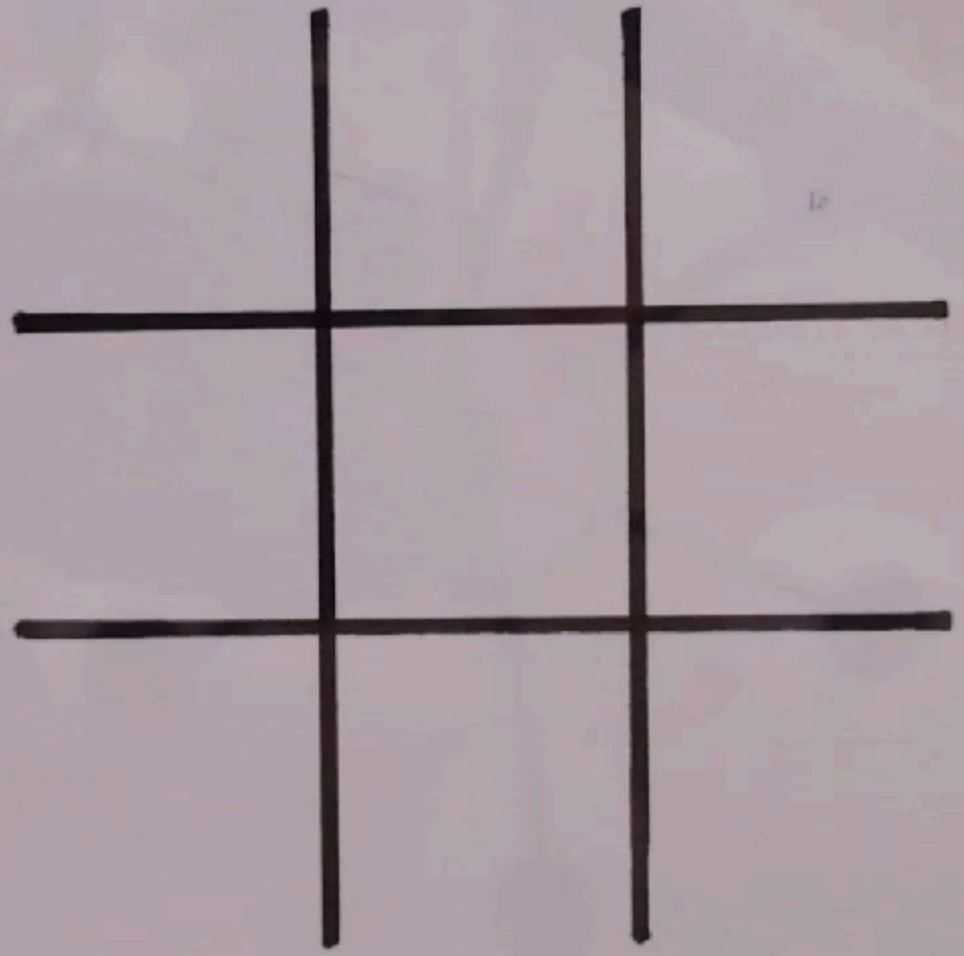
A container is:

an **isolated host process** and

a **Union Filesystem**.


SHALL WE PLAY A GAME?





A **Union Filesystem** has three layers:

A *Merge* or *Union* Layer.

An *Upper* Layer.

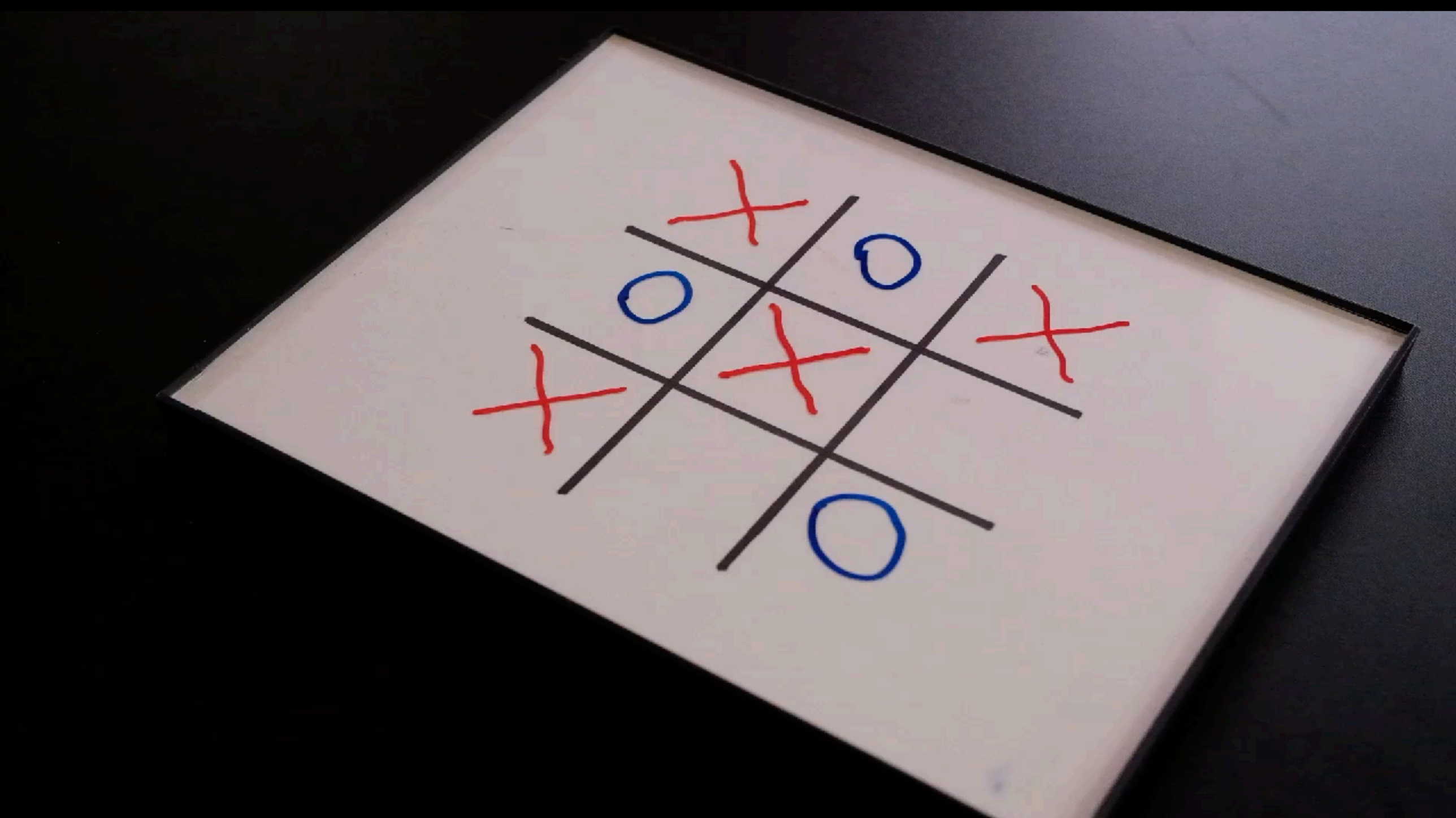
A *Lower* Layer.

A **Union Filesystem** has three layers:

A *Merge* or *Union* Layer: The **game.**

An *Upper* Layer: The **player's moves**.

A *Lower* Layer: The **board and rules**.



The **Upper** (merge) layer isn't "real."

The **Upper** (merge) layer isn't "real."

It's a perceived (calculated) view of **two overlapping layers**.

The **Upper** (merge) layer isn't "real."

It's a perceived (calculated) view of **moves** and **the board.**

Union Filesystem Lab

```
# Create some directories:

mkdir -p ~/workshop/
{image,con1_upper,con1_work,container1}

# cd into the main directory:

cd ~/workshop
```

```
# Set the architecture:

arch="$(arch)"
arch="${arch/arm64/aarch64}"
arch="${arch/amd64/x86_64}"

# wget Alpine:

wget -qO- https://dl-cdn.alpinelinux.org/alpine/v3.19/releases/"${arch}"/
alpine-minirootfs-3.19.1-"${arch}".tar.gz \
     | tar xvz -C ~/workshop/image
```

```
# Create an overlay filesystem

sudo mount -t overlay \
  -o lowerdir=image, \
     upperdir=con1_upper, \
     workdir=con1_work \
     overlay container1
```

```
# Create an overlay filesystem

sudo mount -t overlay \
  -o lowerdir=image, \
     upperdir=con1_upper, \
     workdir=con1_work \
     overlay container1
```

```
# Create an overlay filesystem

sudo mount -t overlay \
  -o lowerdir=image, \
     upperdir=con1_upper, \
     workdir=con1_work \
     overlay container1
```

```
# Create an overlay filesystem

sudo mount -t overlay \
  -o lowerdir=image, \
     upperdir=con1_upper, \
     workdir=con1_work \
     overlay container1
```

```
# Create an overlay filesystem

sudo mount -t overlay \
  -o lowerdir=image, \
     upperdir=con1_upper, \
     workdir=con1_work \
     overlay container1
```

```
# List files in different layers:

ls -l ~/workshop/image
ls -l ~/workshop/con1_upper
ls -l ~/workshop/container1
```

```
# Create files in different layers:

echo "AAA" > ~/workshop/image/AAA
echo "BBB" > ~/workshop/con1_upper/BBB
```

```
# Change a file in the container layer:

echo "aaa" > ~/workshop/con1_upper/AAA
```

```
# Check the contents of file AAA in each layer:

cat ~/workshop/image/AAA
cat ~/workshop/con1_upper/AAA
cat ~/workshop/container1/AAA
```

```
# Update PATH

PATH=$PATH:/bin
```

```
# "Unshare" namespaces and chroot to the "container":

unshare --fork --uts --pid --mount \
        --user --ipc --net \
        --map-root-user \
        chroot ~/workshop/container1 \
        /bin/sh
```

```
# "Unshare" namespaces and chroot to the "container":

unshare --fork --uts --pid --mount \
        --user --ipc --net \
        --map-root-user \
        chroot ~/workshop/container1 \
        /bin/sh
```

```
# "Unshare" namespaces and chroot to the "container":

unshare --fork --uts --pid --mount \
        --user --ipc --net \
        --map-root-user \
        chroot ~/workshop/container1 \
        /bin/sh
```

```
# "Unshare" namespaces and chroot to the "container":

unshare --fork --uts --pid --mount \
        --user --ipc --net \
        --map-root-user \
        chroot ~/workshop/container1 \
        /bin/sh
```

```
# "Unshare" namespaces and chroot to the "container":

unshare --fork --uts --pid --mount \
        --user --ipc --net \
        --map-root-user \
        chroot ~/workshop/container1 \
        /bin/sh
```

```
# "Unshare" namespaces and chroot to the "container":

unshare --fork --uts --pid --mount \
        --user --ipc --net \
        --map-root-user \
        chroot ~/workshop/container1 \
        /bin/sh
```

```
# "Unshare" namespaces and chroot to the "container":

unshare --fork --uts --pid --mount \
        --user --ipc --net \
        --map-root-user \
        chroot ~/workshop/container1 \
        /bin/sh
```

```
# "Unshare" namespaces and chroot to the "container":

unshare --fork --uts --pid --mount \
        --user --ipc --net \
        --map-root-user \
        chroot ~/workshop/container1 \
        /bin/sh
```

```
# "Unshare" namespaces and chroot to the "container":

unshare --fork --uts --pid --mount \
        --user --ipc --net \
        --map-root-user \
        chroot ~/workshop/container1 \
        /bin/sh
```

```
# "Unshare" namespaces and chroot to the "container":

unshare --fork --uts --pid --mount \
        --user --ipc --net \
        --map-root-user \
        chroot ~/workshop/container1 \
        /bin/sh
```

```
# "Unshare" namespaces and chroot to the "container":

unshare --fork --uts --pid --mount \
        --user --ipc --net \
        --map-root-user \
        chroot ~/workshop/container1 \
        /bin/sh
```

```
# Check the hostname:

hostname
```

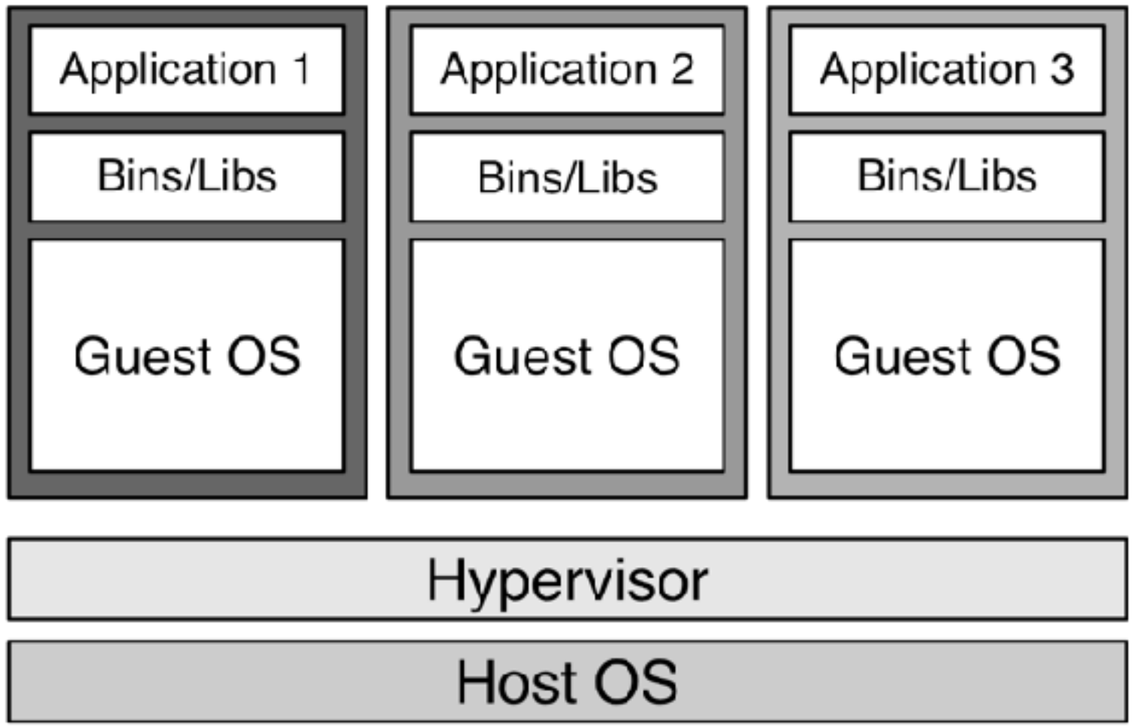
```
# Change the hostname

hostname alpine
hostname
```

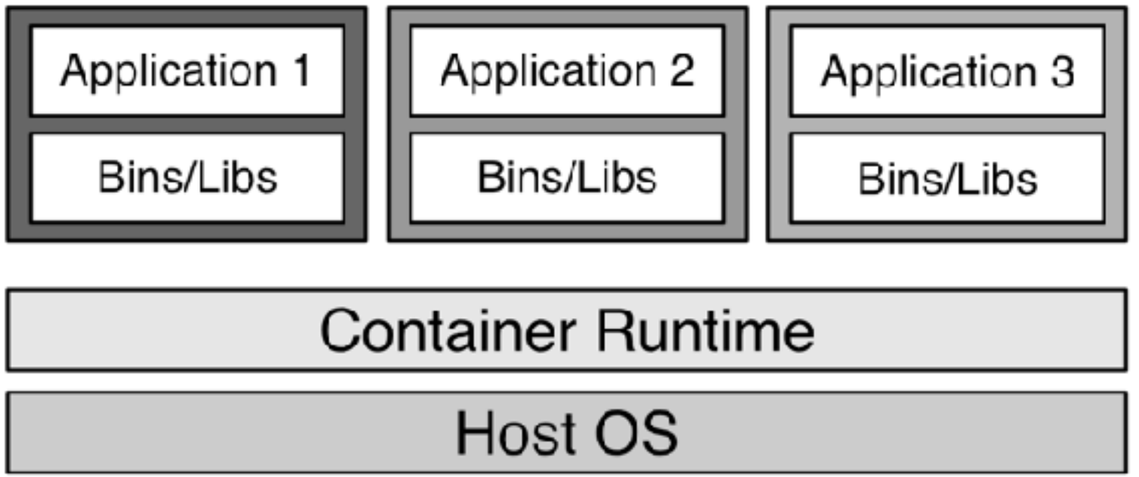
```
# Explore the "container"

cat /etc/os-release
uname -a
whoami
echo $$
pwd
```

```
# What files are here?

ls -l /
cat AAA
cat BBB
```

```
# Add a group and user:

addgroup -g 12345 workshop_g
adduser -H -S -u 54321 workshop_u -G workshop_g
```

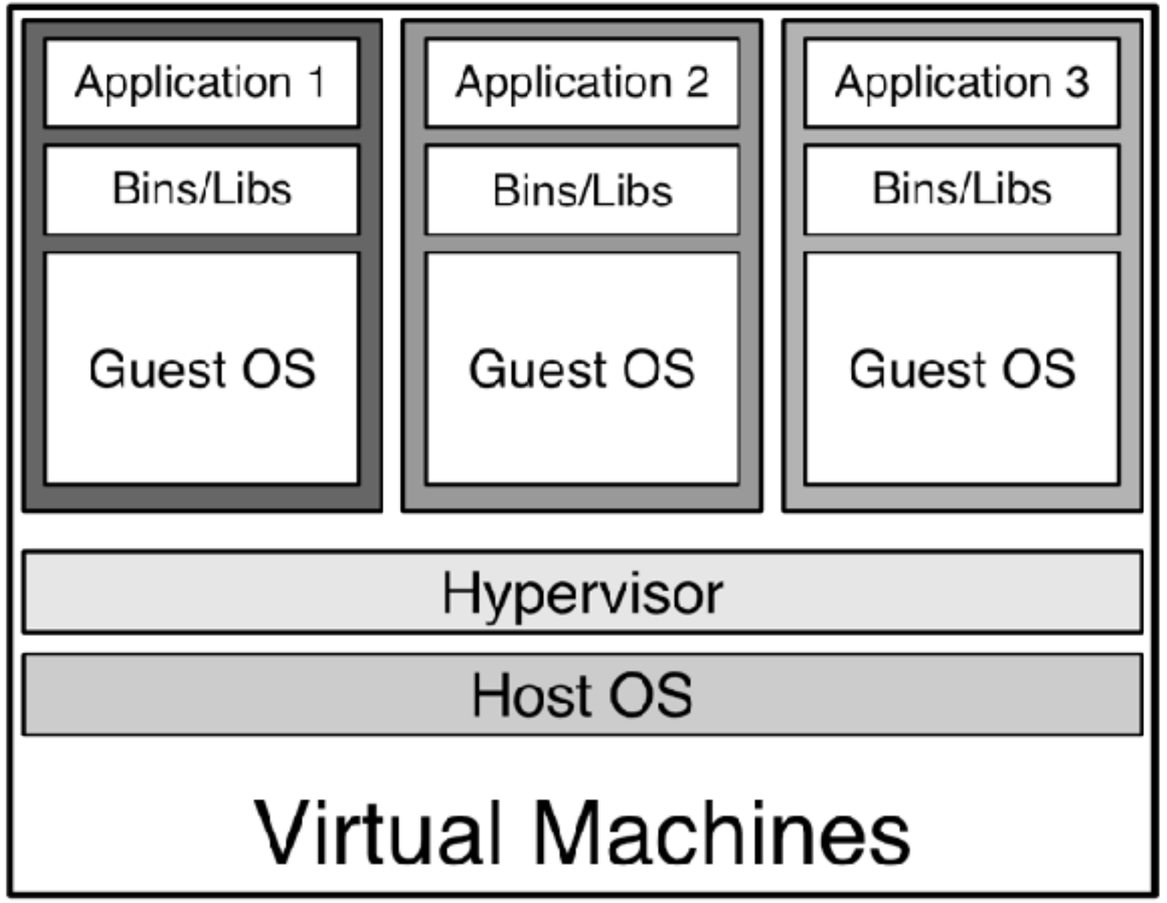
Containers are Magic!



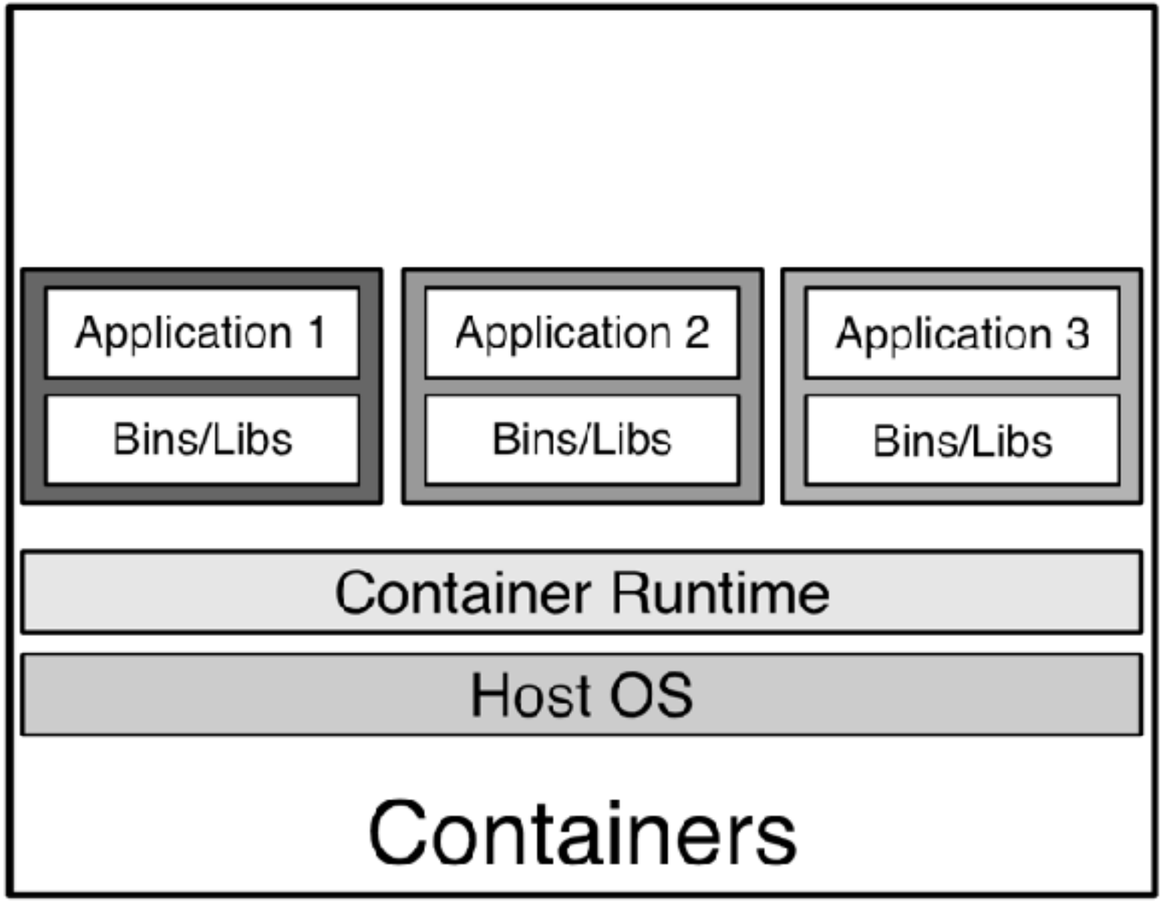


Three full operating systems

Three application/executable directories

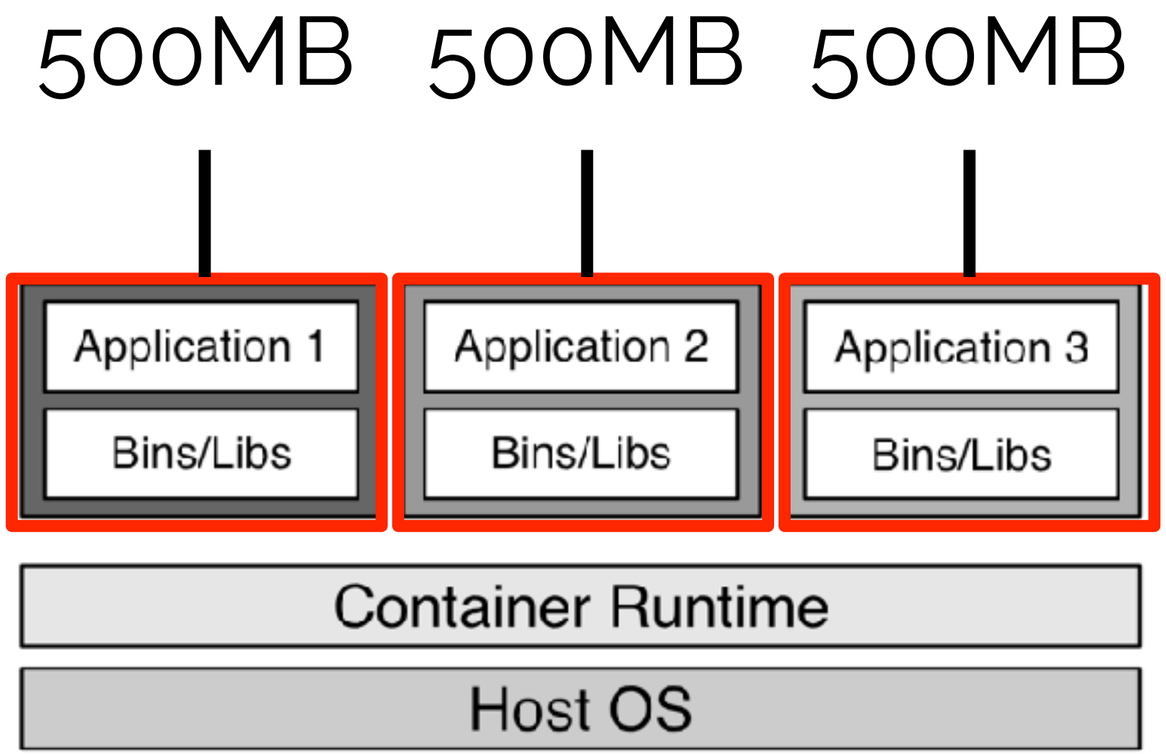


Three full operating systems



Three application/executable directories

How much space do three containers use?



500MB
+ 500MB
+ 500MB

1,500MB

Multiple identical containers will share an **image**.

```
# Prepare directories for a second container

mkdir -p ~/workshop/{con2_upper,con2_work,container2}

cd ~/workshop
```

```
# Create a second overlay filesystem

sudo mount -t overlay \
  -o lowerdir=image, \
     upperdir=con2_upper, \
     workdir=con2_work \
     overlay container2
```

```
# Create a second overlay filesystem

sudo mount -t overlay \
  -o lowerdir=image, \
     upperdir=con2_upper, \
     workdir=con2_work \
     overlay container2
```

```
# Create a second overlay filesystem

sudo mount -t overlay \
  -o lowerdir=image, \
     upperdir=con2_upper, \
     workdir=con2_work \
     overlay container2
```

```
# Set the environment

PATH=$PATH:/bin

# "Run" the "container"

unshare --fork --uts --pid --mount \
        --user --ipc --net \
        --map-root-user \
        chroot ~/workshop/container2 \
        /bin/sh
```

```
# What files are here?

ls -l /
cat AAA
cat BBB
```

```
# Manipulate files within the container:

echo "111" > /AAA
echo "222" > /BBB
```

```
# Check the contents of files from the host:

cat ~/workshop/image/AAA
cat ~/workshop/con2_upper/AAA
cat ~/workshop/con2_upper/BBB
```

Containers **persist** state (moves) in the run-time layer of its Union Filesystem. Containers are **stateful**.

```
# Exit and restart the container:

exit

unshare --fork --uts --pid --mount \
        --user --ipc --net \
        --map-root-user \
        chroot ~/workshop/container2 \
        /bin/sh
```

```
# Confirm the file contents:

ls -l /
cat AAA
cat BBB
```

Container Properties

Dropping a **Container** removes its **Layers.**

```
# "Drop" the container:

sudo umount container2

rm -fr ~/workshop/container2
rm -fr ~/workshop/con2_upper
rm -fr ~/workshop/con2_work
```

Containers are **Ephemeral.**

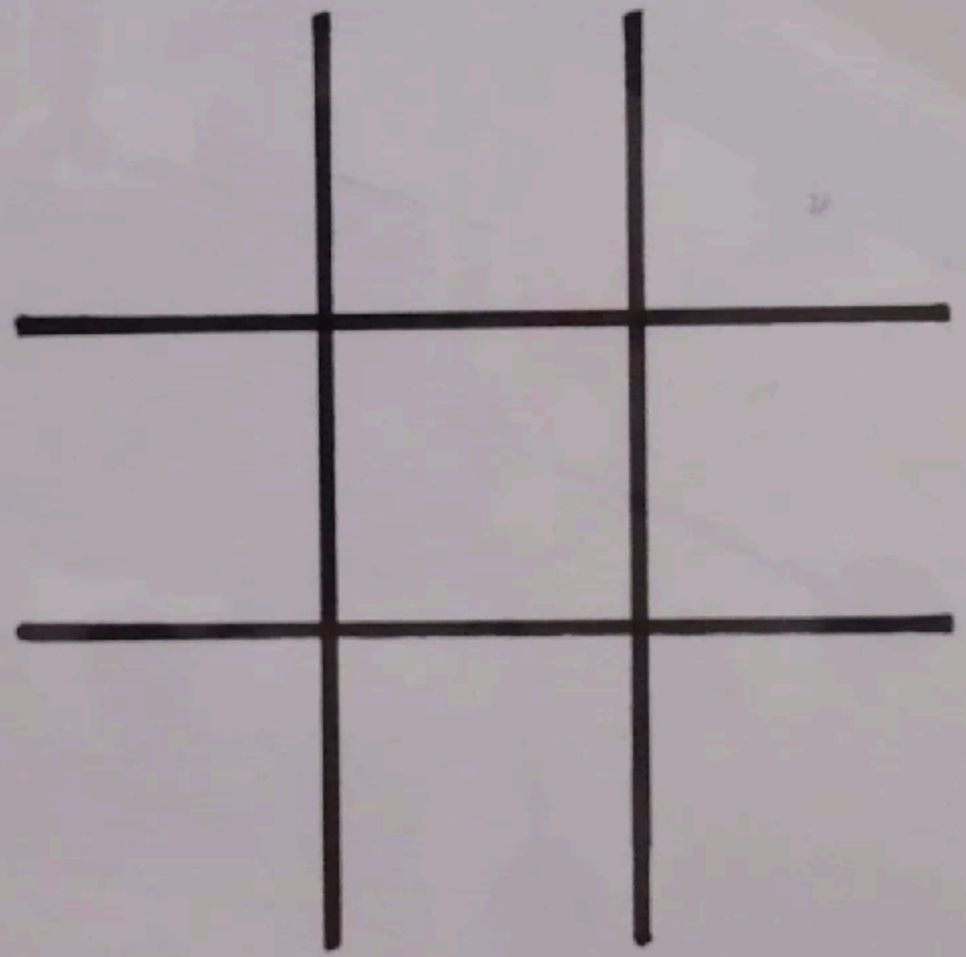
Ephemeral does not have a time limit.

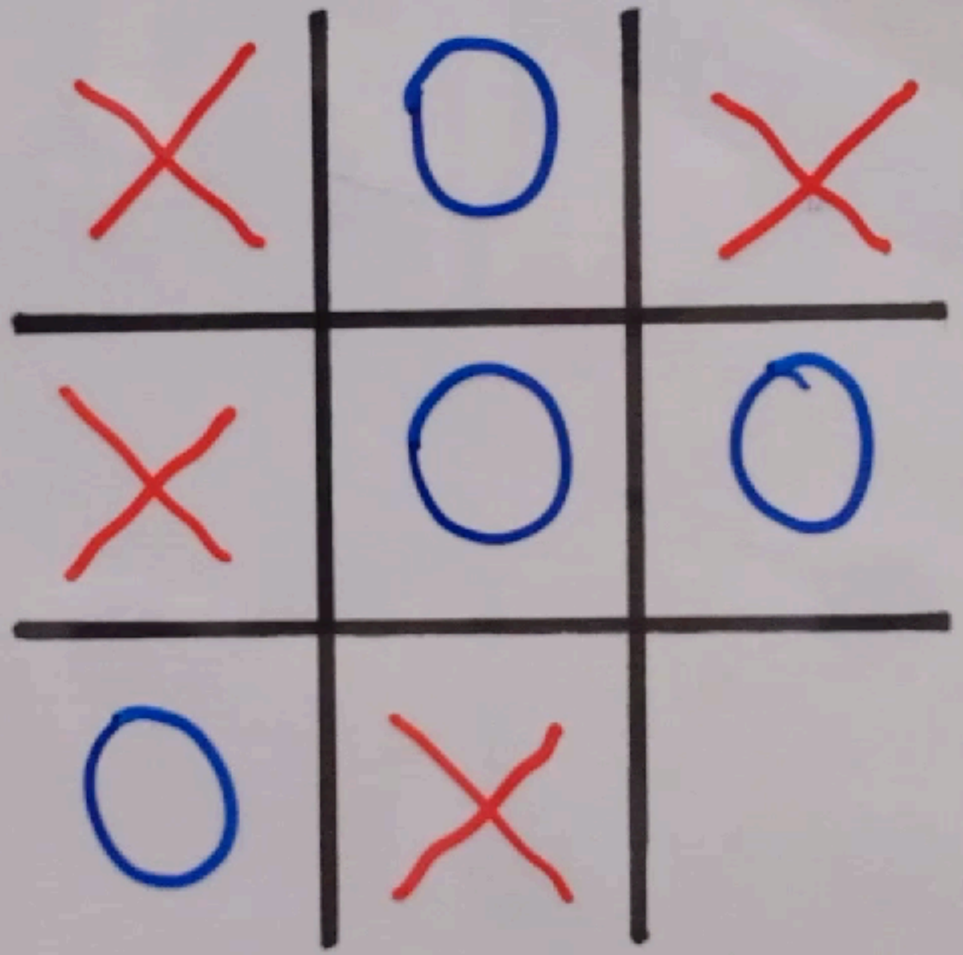
Ephemeral does not have a time limit.

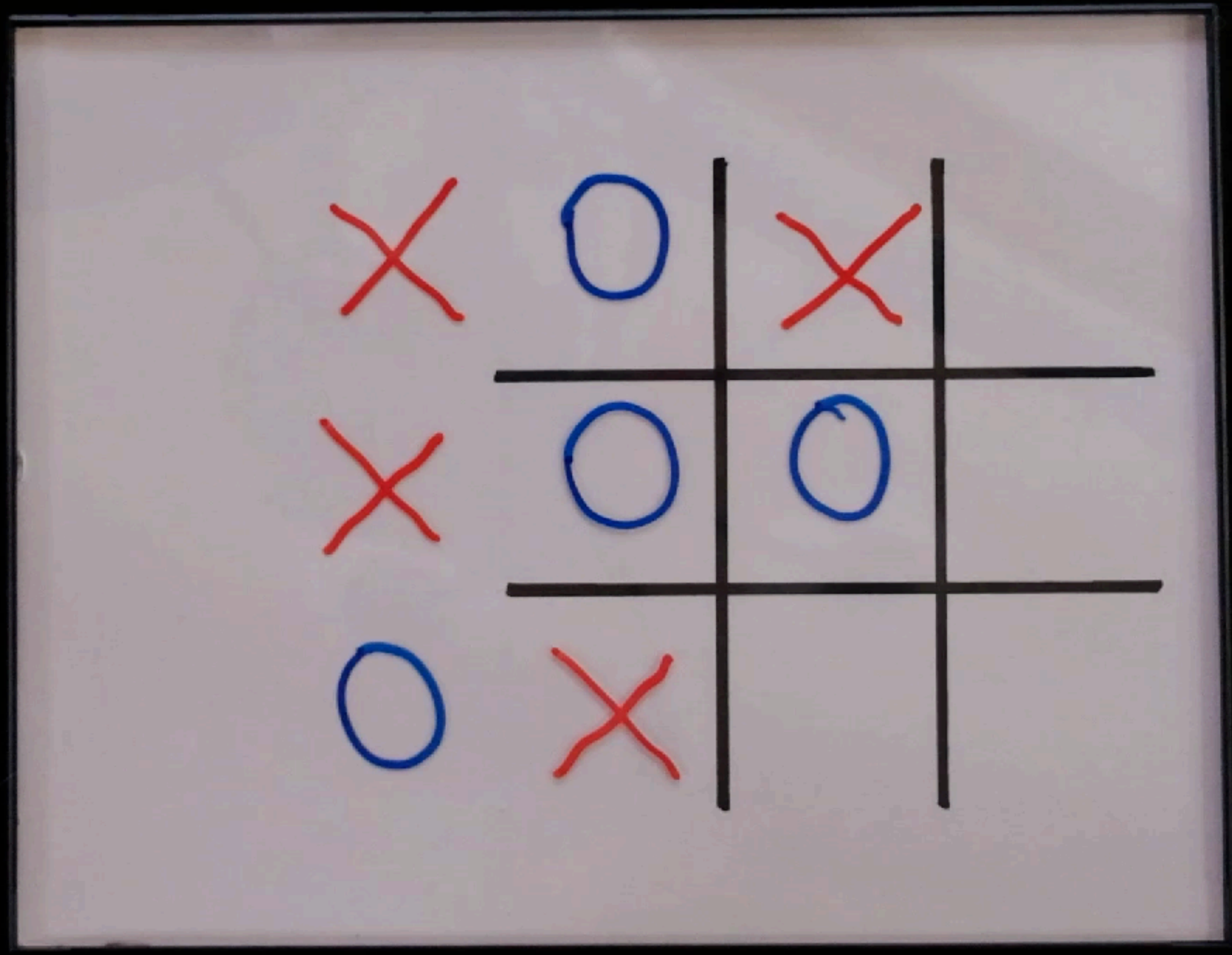
On a long enough scale,
everything is ephemeral.

Images are **immutable** and *never change*.

Stalemate!

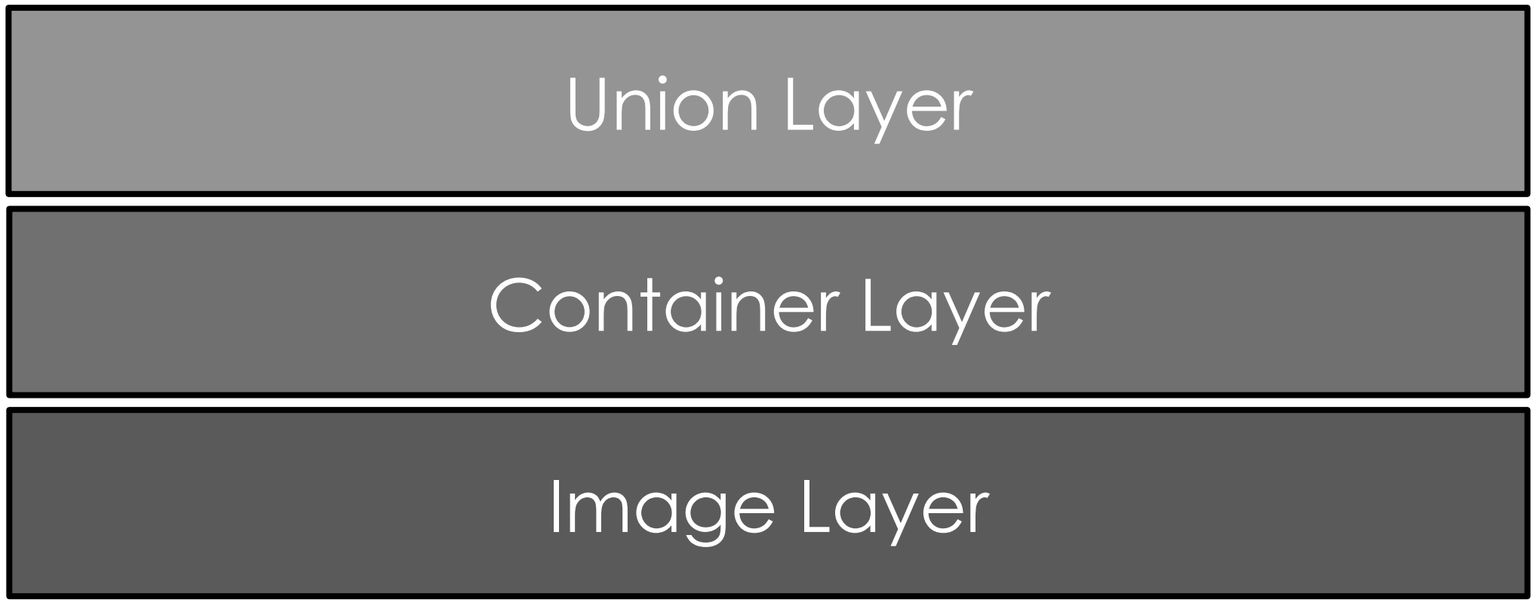


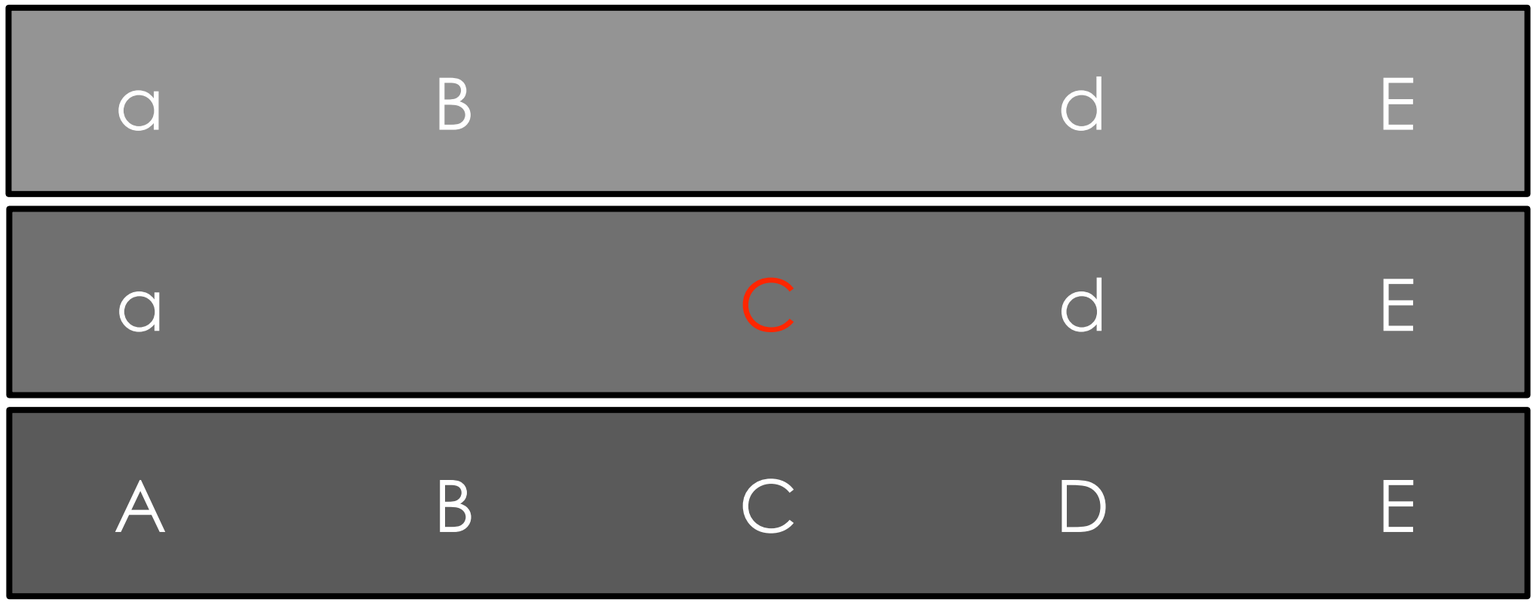




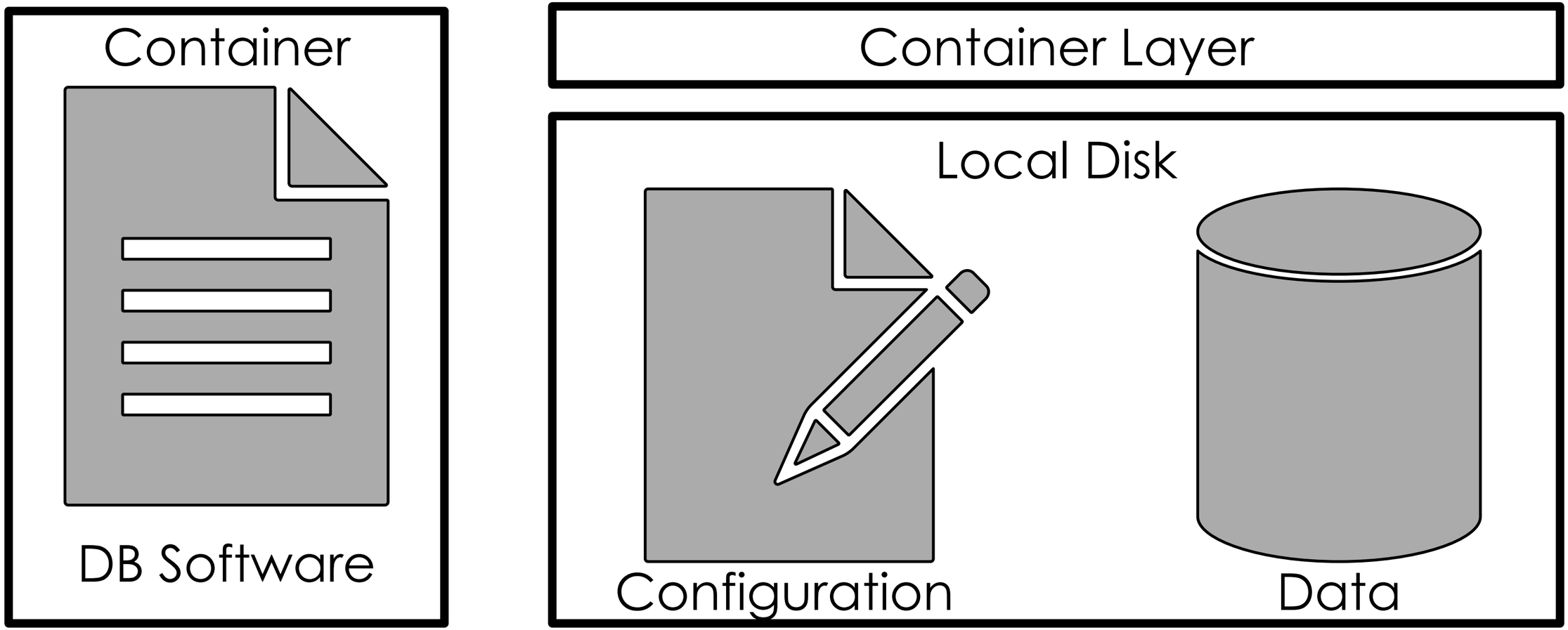
Volumes

Volumes "externalize" container directories.


Union Layer
Container Layer
Image Layer


a B d E
a C d E
A B C D E

Database Containers



Database Image Rule #1:

If there is a database, start it.

If not, create a new database.

Database Container Startup

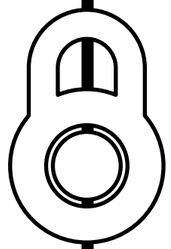
```
docker volume create ${CONTAINER_NAME}_data

docker run -d \
       --name ${CONTAINER_NAME} \
       -v /data/${CONTAINER_NAME}:<data_directory> \
       <image_name>
```

Default Volumes

Linux

```
ls -l /var/lib
drwx--x--- root root docker
```

Docker Desktop



Virtual Machine

`/var/lib/docker`

```
mkdir -p /data/${CONTAINER_NAME}

docker volume create \
        --opt type=none \
        --opt o=bind \
        --opt device=/data/${CONTAINER_NAME} \
        ${CONTAINER_NAME}_data
```

Database Container Startup

```
docker run -d \
       --name ${CONTAINER_NAME} \
       -v /data/${CONTAINER_NAME}:<data_directory> \
       <image_name>
```

Database Image Rule #1:

If there is a database, start it.

If not, create a new database.

Clone the Data Directory

```
cp -rp /data/${CONTAINER_NAME}/ \
       /data/clone
```

Clone the Data Directory

```
cp -rp /data/clone/ \
       /data/${CONTAINER_NAME}
```

Clone the Data Directory

```
cp -rp /data/clone/ \
       /data/${CONTAINER_NAME}

cp -rp /NFS/gold_data/clone/ \
       /data/${CONTAINER_NAME}
```

Clone the Data Directory

```
cp -rp /data/clone/ \
       /data/${CONTAINER_NAME}

cp -rp /NFS/gold_data/clone/ \
       /data/${CONTAINER_NAME}

cp -rp /NFS/gold_data/clone/2024-05-08.0900/ \
       /data/${CONTAINER_NAME}
```

Building Images

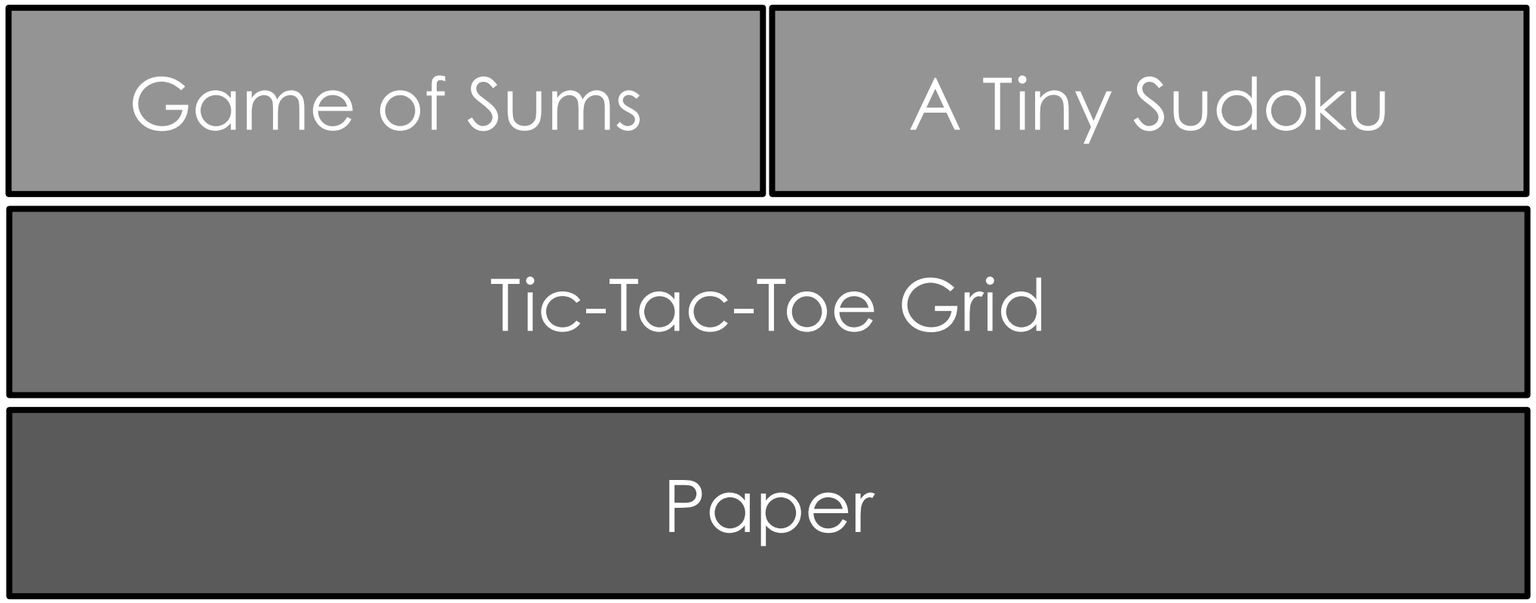
A **Containerfile** is a recipe for building images.

Tic-Tac-Toe Grid

Paper

Image layers are *modular* and *reusable*.

Layers can be **cached** and reused among multiple images.


Game of Sums
A Tiny Sudoku
Tic-Tac-Toe Grid
Paper

```
# Create some directories:

mkdir -p ~/workshop/
{image,con3_upper,con3_work,container3}

# cd into the main directory:
cd ~/workshop
```

```
# Create an overlay filesystem

sudo mount -t overlay \
  -o lowerdir=container1, \
     upperdir=con3_upper, \
     workdir=con3_work \
     overlay container3
```

```
# Create an overlay filesystem

sudo mount -t overlay \
  -o lowerdir=container1, \
     upperdir=con3_upper, \
     workdir=con3_work \
     overlay container3
```

```
# Create an overlay filesystem

sudo mount -t overlay \
  -o lowerdir=container1, \
     upperdir=con3_upper, \
     workdir=con3_work \
     overlay container3
```

```
# What files are in each layer?

ls -l ~/workshop/con3_upper
ls -l ~/workshop/con3_work
ls -l ~/workshop/container3
cat ~/workshop/container3/AAA
```

```
# Update PATH

PATH=$PATH:/bin
```

```
# "Unshare" namespaces and chroot to the "container":

unshare --fork --uts --pid --mount \
        --user --ipc --net \
        --map-root-user \
        chroot ~/workshop/container3 \
        /bin/sh
```

```
# Explore the container contents

ls -l
cat AAA
```

Cleanup

```
# Cleanup

sudo umount container3
sudo umount container1

cd ~

rm -fr ~/workshop
```

Questions

sean.scott@viscosityna.com
https://linktr.ee/oraclesean

Contact Me

sean.scott@viscosityna.com
https://linktr.ee/oraclesean