Docker 101

Denis Timofeev

Feb 07, 2019

Outline

Docker Purpose?

Purpose

Aims to solve common software problems and simplifies installing, running, publishing, and removing software. Accomplishes this by leveraging UNIX technology called containers

Container

Container — lightweight application isolation mechanism allowing the kernel to run process separate from the host system in its own isolated user space. The container has its own process list, can have its own network stack, file systems, and other resources, but shares the kernel with the host and other containers

What does Docker provide?

- Strict organization (no library mess) — dependencies packed with application
- Improved portability — language / environment / environment state
- Protecting machine — limits the access to resources
- Application abstraction — how to run instead of how to install

Containers History

- 1979: Unix V7 — *chroot*. Added to BSD in 1982
- 2000: FreeBSD Jails — partition system to independent systems
- 2001: Linux VServer — partition resources (file systems, network addresses, memory)
- 2004: Solaris Containers — zones
- 2006: Process Containers — Control Groups (cgroups)
- 2008: LXC — LinuX Containers (cgroups and Linux namespaces)
- 2013: Docker

Name

Containers (after Solaris 10 and Solaris Containers) — environment for an application which prevents that application from accessing protected resources

Docker is just a tool

Tool

Docker is a tool used to create, control, and manage containers. Docker adds API, image format, delivery, and sharing model to containers technology

Docker Glossary

Docker Container

Standardized, encapsulated environment running application

Docker Image

Docker image contains an application and all its dependencies. When a container is started, a read-write layer for that container is combined with the read-only image. Image is just a template for creating container (not virtual machine image)

Docker Registry

Docker Registry is a repository for Docker images with "pull" / "push" scheme

One More Time

Docker Image - Template

Docker images such as centos, ubuntu are not operating systems. Just a similar filesystem structure and set of built-in tools you find in OSes line Centos and Ubuntu

No Kernel

Docker image doesn't contain kernel, but there is still positibility of invoking non-existing system call. Docker requires kernel version > 3.10, so the posibility of not having backward compatible calls is very low

Docker Components

- The Docker daemon (dockerd) — controls container lifecycle
- Command-line client (docker) — sends commands to daemon
- Set of services and APIs — the Docker Engine API
- Docker Registry — publish/share images
- Docker Compose — declarative deployment
- Docker Swarm Mode — clusterization/orchestration (superseded by Kubernetes)

Outline

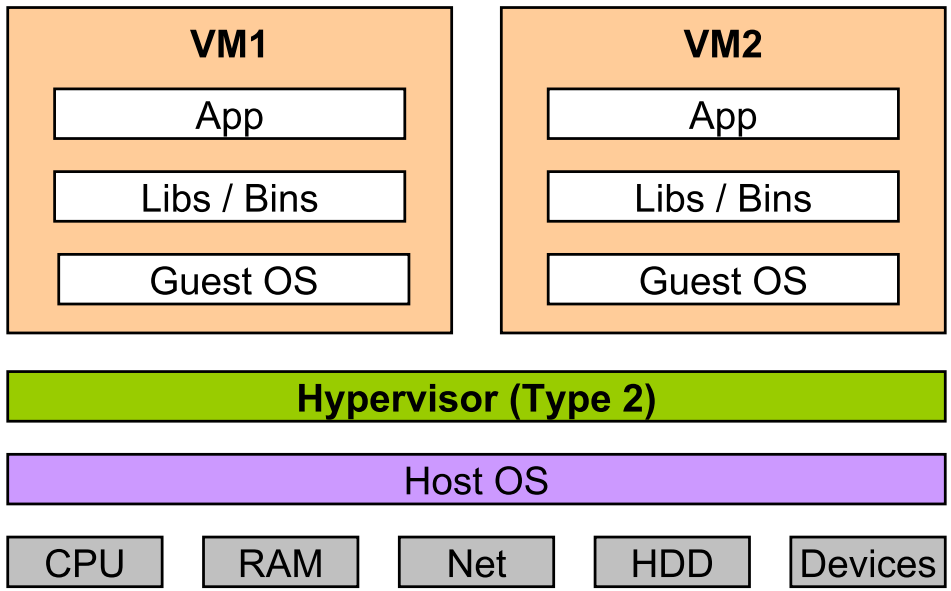
Under the hood

Container is a child process of docker daemon

- PID namespace — Process identifiers and capabilities
- UTS namespace — Host and domain name
- MNT namespace — File system access and structure
- IPC namespace — Process communication over shared memory
- NET namespace — Network access and structure
- USR namespace — User names and identifiers
- chroot — Controls the location of the file system root
- cgroups — Resource protection

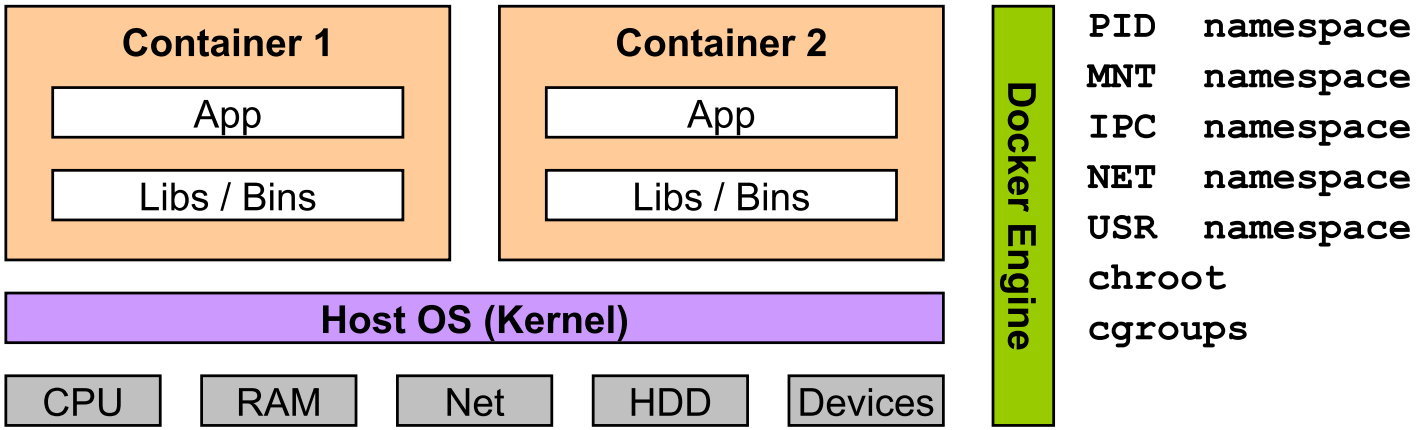
Docker vs VM

- Hypervisor (Type2)
- Guest OS



Docker vs VM

- ~~Hypervisor~~
- ~~Guest OS~~
- System calls to Linux Kernel



Outline

Docker hello-world

- Docker *"run"* creates new container each time

```
docker run hello-world

Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
d1725b59e92d: Pull complete
Digest: sha256:0add3ace90e
Status: Downloaded newer image

Hello from Docker!
```

- Run detached process — background mode

```
docker run --detach --name web nginx:latest
```

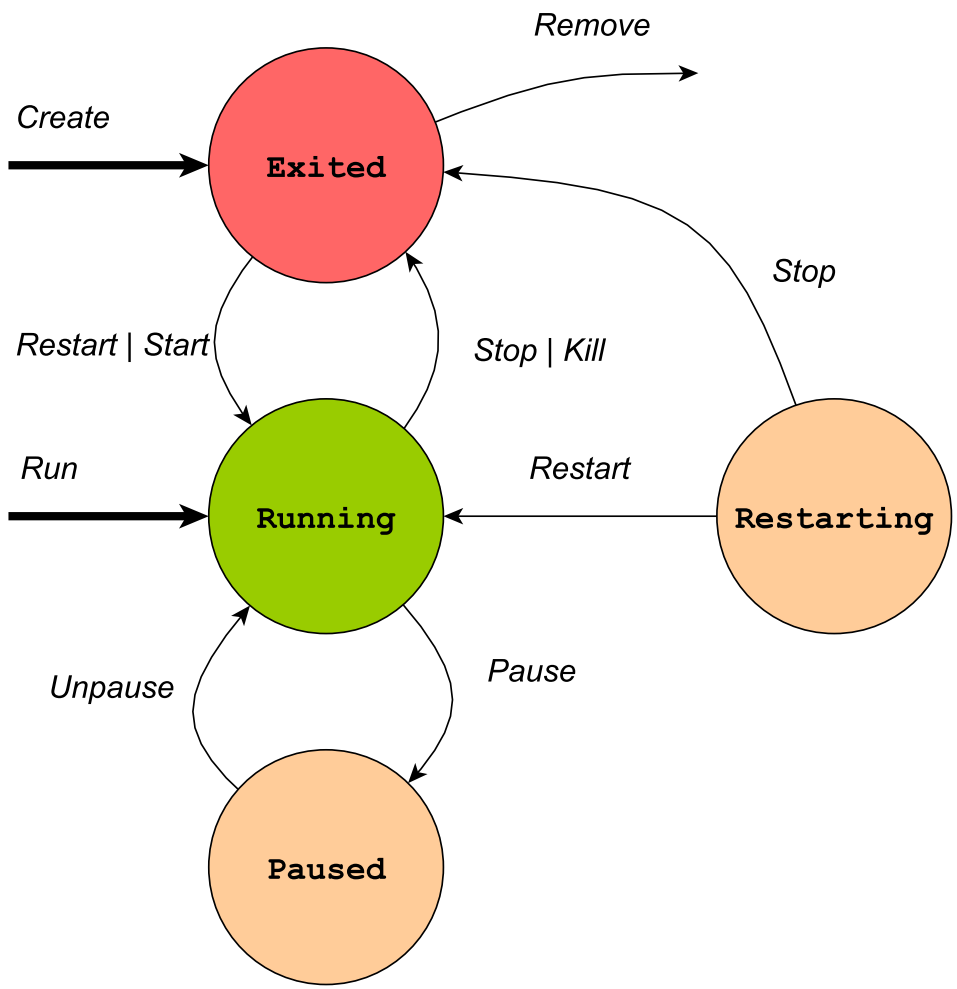
Docker interactive

- Docker "run -it" in interactive mode
- Attach to STDIN/STDOUT
- Attach terminal to STDIN/STDOUT

```
docker run --interactive --tty \
           busybox:latest /bin/sh
```

Docker Commands And Lifecycle

- create / start / stop / kill / restart / rm / exec / logs



Docker PS (-a)

- The container ID
- The image used
- The command executed in the container
- The time since the container was created
- The duration that the container has been running
- The network ports exposed by the container
- The name of the container

```
CONTAINER ID  IMAGE                     COMMAND             CREATED
379f56c48527  grafana/grafana:5.0.3     "/run.sh"           3 weeks ago
a58b3427eda6  prom/prometheus:v2.2.1    "/bin/prometheus"   3 weeks ago
0c64881769f9  stressy:latest          " java "    3 weeks ago
```

```
STATUS       PORTS       NAMES
Up 3 weeks   0.0.0.0:3002->3000/tcp   stressy-grafana
Up 3 weeks   0.0.0.0:9092->9090/tcp   stressy-prometheus
Exited (137)                          stressy
```

Docker Commands Examples

- Read Only Image + Volumes

```
docker run -d --name test \
       -v /run/lock/test/ --read-only \
      busybox:latest
```

- Environment variables

```
docker run --env TEST_ENV_VAR="test" \
           busybox:latest env
```

Docker Commands Examples

- Linking containers (Deprecated!!!)

```
docker run -d --name db -v /var/db/data db:latest

docker create --name=reader1 --link db \
              db-reader:latest

docker create --name=reader2 --link db \
              db-reader:latest
```

- Expose port

```
docker run --detach --name web -p 80 nginx:latest
```

Docker Commands Examples

- Override entrypoint

```
docker run -ti --entrypoint=sh nginx:latest
```

Troubleshooting

Usefull for troubleshooting to override entrypoint and "look" into container if it fails to start, for example

Docker Containers Automatic Restart

- Never restart (default) — no
- Restart when a failure is detected — on-failure
- Restart when a failure is detected unless it explicitely stopped — unless-stopped
- Always restart the container regardless of the condition — always

```
docker run --restart=always \
           busybox:latest env
```

Docker Container Commands Recap

- create — create a container from an image
- start — start an existing container
- run — create a new container and start it
- ps — list running containers
- logs — print container logs
- stop — gracefully stop running container
- kill — stop main process in container container abruptly
- rm — remove stopped container

Outline

Dockerfile

Dockerfile

Set of instructions which say how to build Docker image consisting of base image, filesystem layers, environment variable and instructions to run application

Dockerfile

```
# base image
FROM docker.in.devexperts.com/ria/ria-openjdk:8u151-jdk-alpine3.7

# add metadata for searching and identification
LABEL "maintainer"="dtimofeev@devexperts.com"
LABEL "appserver"="Tomcat"

# set environment variables
ENV CATALINA_HOME /usr/local/tomcat
ENV PATH $CATALINA_HOME/bin:$PATH
RUN mkdir -p "$CATALINA_HOME"
WORKDIR $CATALINA_HOME

ENV TOMCAT_NATIVE_LIBDIR $CATALINA_HOME/native-jni-lib
ENV LD_LIBRARY_PATH ${LD_LIBRARY_PATH:+$LD_LIBRARY_PATH:}$TOMCAT_NATIVE_LIBDIR

ENV TOMCAT_MAJOR 8
ENV TOMCAT_VERSION 8.0.41

ENV TOMCAT_TGZ_URL http://archive.apache.org/dist/tomcat/tomcat-$TOMCAT_MAJOR/
 v$TOMCAT_VERSION/bin/apache-tomcat-$TOMCAT_VERSION.tar.gz

RUN apk --update add tar

RUN wget -O tomcat.tar.gz "$TOMCAT_TGZ_URL" \
    && tar -xvf tomcat.tar.gz --strip-components=1 \
    && rm tomcat.tar.gz*
```

Several Dockerfile commands

- FROM — specifies base image
- LABEL — provides metadata
- ENV — sets environmental variables
- RUN — runs command and creates image layer
- COPY — copies files and directories to the image
- ADD — copies files and directories to the image. Unpacks .tar files
- CMD — sets a command/arguments for an executing container. Parameters can be overridden. There can be only one CMD
- ARG — defined a variable to pass to Docker at build-time
- EXPOSE — exposes a port
- VOLUME — creates a directory mount point to access and store persistent data
- ENTRYPOINT — configures command which runs each time a container is started

Docker Image Build

```
Removing intermediate container 4e76a27d0987
Step 3/13 : ENV CATALINA_HOME /usr/local/tomcat
 ---> Running in 9e95902b7445
 ---> 6ad788b83172
Removing intermediate container 9e95902b7445
Step 4/13 : ENV PATH $CATALINA_HOME/bin:$PATH
 ---> Running in b4dbb019a07f
 ---> 2245940b88fe
Removing intermediate container b4dbb019a07f
Step 5/13 : RUN mkdir -p "$CATALINA_HOME"
 ---> Running in c33fddbaa99e
 ---> d580a8d4986c
Removing intermediate container c33fddbaa99e
Step 6/13 : WORKDIR $CATALINA_HOME
 ---> 418034d9a7d6
Removing intermediate container 35f2929b5b51
```

Troubleshooting build

In case of error during build process

```
Removing intermediate container 4e76a27d0987
Step 3/13 : ENV CATALINA_HOME /usr/local/tomcat
 ---> Running in 9e95902b7445
 ---> 6ad788b83172
Removing intermediate container 9e95902b7445
Step 4/13 : ENV PATH $CATALINA_HOME/bin:$PATH
 ---> Running in b4dbb019a07f
      ERROR!!!
```

Troubleshoot

you can launch command in interactive mode in the last container before the error step

```
docker run --rm -ti 6ad788b83172 /sh
```

Keeping images small

- Multi-goal commands — creates only one additional layer

```
RUN wget -O tomcat.tar.gz "$TOMCAT_TGZ_URL"
    && tar -xvf tomcat.tar.gz --strip-components=1
    && rm tomcat.tar.gz*
```

Keeping images small

- Multi-stage builds

```
FROM golang:1.7.3 as builder
WORKDIR /go/src/github.com/alexellis/href-counter/
RUN go get -d -v golang.org/x/net/html
COPY app.go .
RUN CGO_ENABLED=0 GOOS=linux go build -a -installsuffix cgo -o app .

# use empty image
FROM alpine:latest
RUN apk --no-cache add ca-certificates
WORKDIR /root/
# copy from builder container
COPY --from=builder /go/src/github.com/alexellis/href-counter/app .
CMD ["./app"]
```

Docker Image Layers

```
docker history docker.in.devexperts.com/ria/tomcat
IMAGE          CREATED         CREATED BY                                      SIZE
35cc64ce1494   5 months ago    wget -O tomcat.tar.gz $TOMCAT_                  13.3 MB
<missing>      5 months ago    apk --update add tar                            1.76 MB
<missing>      5 months ago    ENV TOMCAT_TGZ_URL=http...                      0 B
<missing>      5 months ago    ENV TOMCAT_VERSION=8.0.41                       0 B
<missing>      5 months ago    ENV TOMCAT_MAJOR=8                              0 B
<missing>      5 months ago    ENV LD_LIBRARY_PATH=/us...                      0 B
<missing>      5 months ago    ENV TOMCAT_NATIVE_LIBDI...                      0 B
<missing>      5 months ago    WORKDIR /usr/local/tomcat                       0 B
<missing>      5 months ago    mkdir -p $CATALINA_HOME                         0 B
<missing>      5 months ago    ENV PATH=/usr/local/tom...                      0 B
<missing>      5 months ago    ENV CATALINA_HOME=/usr/...                      0 B
<missing>      5 months ago    MAINTAINER Denis Timofe...                      0 B
<missing>      5 months ago    ENTRYPOINT [/usr/bin/d...                       0 B
<missing>      5 months ago    apk add --no-cache dumb-init                    137 kB
<missing>      5 months ago    MAINTAINER Denis Timofe...                      0 B
<missing>      6 months ago    set -x  && apk add --no-cache                   97.4 MB
...
```

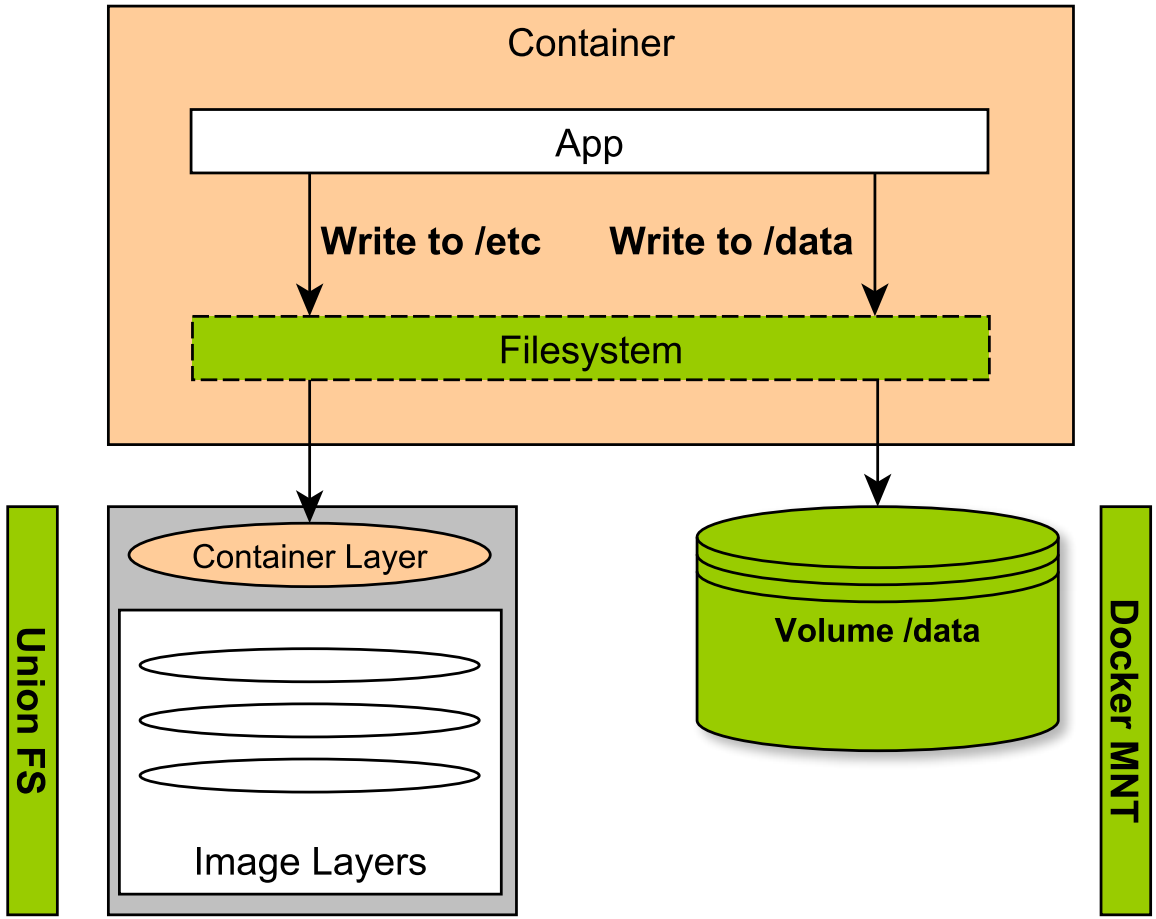
Docker Image Commands Recap

- build — build an image
- login — login to a remote repository
- push — push an image to a remote repository
- images — list all downloaded/built images
- history — intermediate image info
- inspect — info about the image
- rmi — delete image

Outline

Docker Bind Mount Point

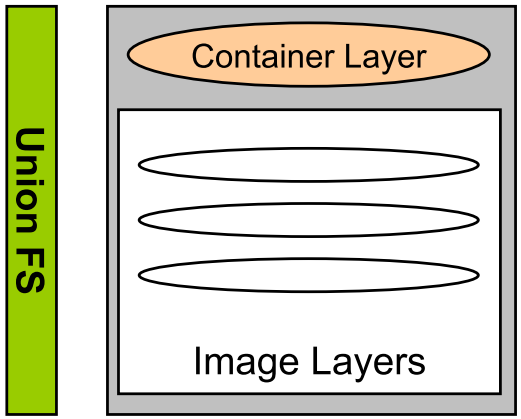
```
docker run -d -v ./host-folder:/data:ro alpine...
```



Union FS

Union FileSystem

Docker Storage Drivers: overlay2, aufs, devicemapper, btrfs, zfs, vfs



Memmory-mapped files

If your application uses memmory-mapped files it's better to place them in bind-mount volume

Docker Bind Mount File

```
docker run -d

-v ./folder/test.txt:/folder/text.txt:ro
        alpine...
```

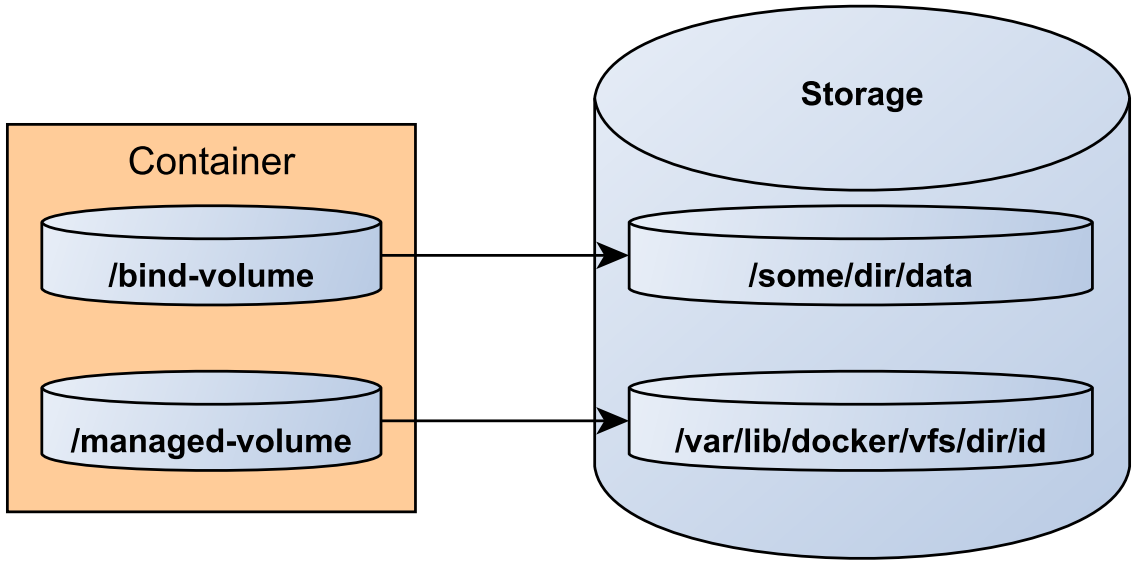
Existing file

File MUST exist in image which is used for container. Otherwise new directory will be created and named with filename

Docker Managed Volume

```
docker run -d --volume /var/lib/data --name data-
  shared alpine...

docker run -d --volumes-from data-shared busybox...
```

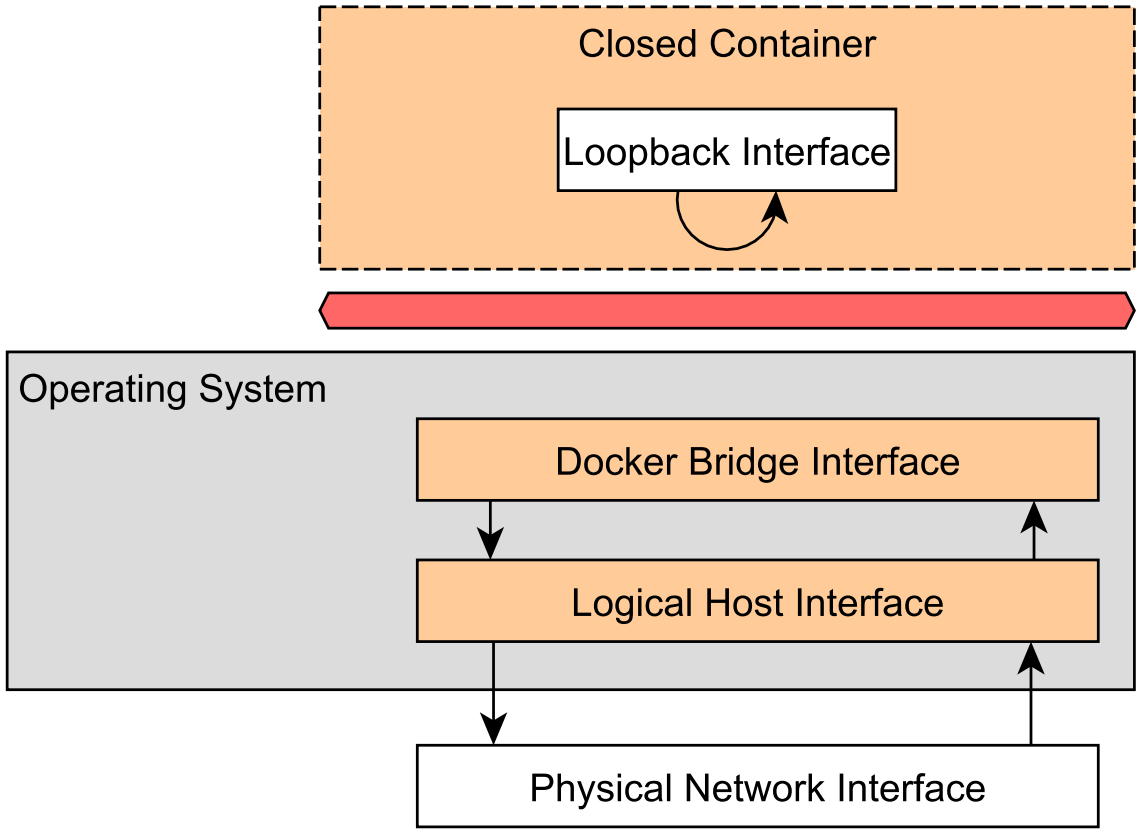


Outline

Closed Container

```
docker run --rm --net none busybox ip addr
```

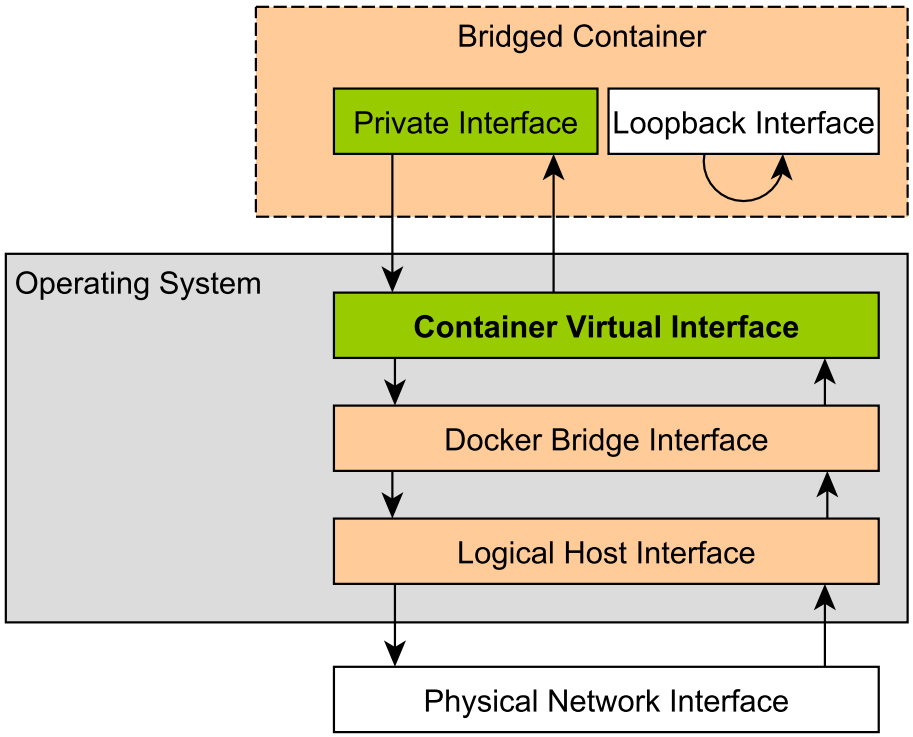
- Doesn't allow any network traffic



Bridged Container

```
docker run --rm --net bridge busybox ip addr
docker run --rm --net bridge -p 8080:80 busybox ip addr
docker run --rm --net bridge --hostname=busy busybox ip addr
```

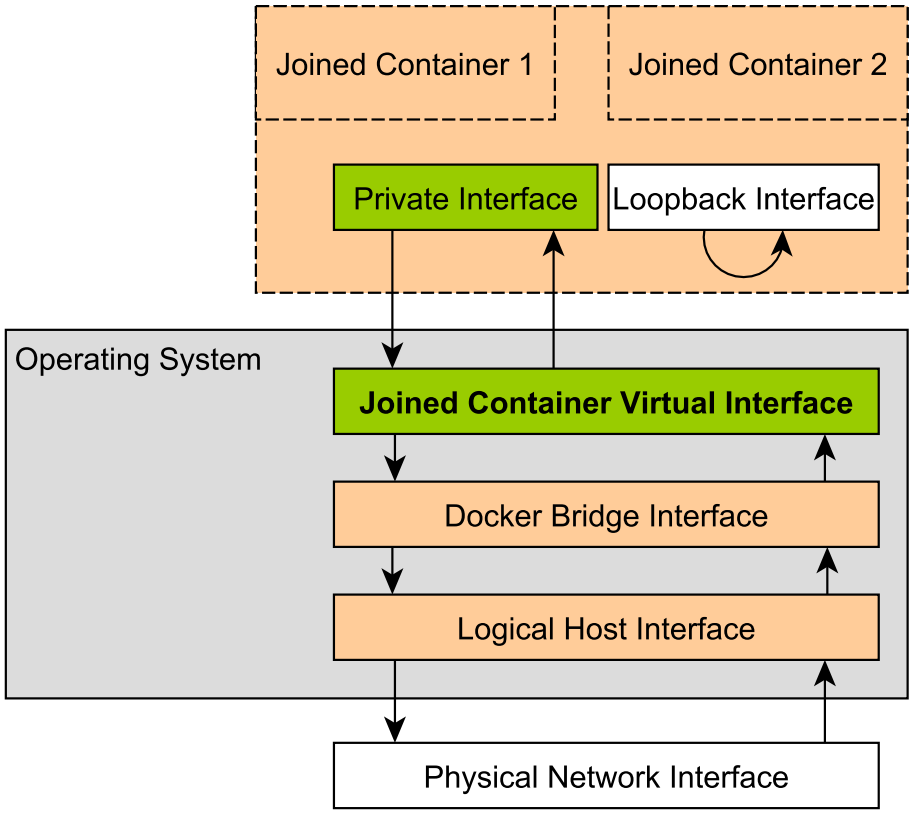
- Allows access to network / containers can be found using their ip addresses



Joined Container

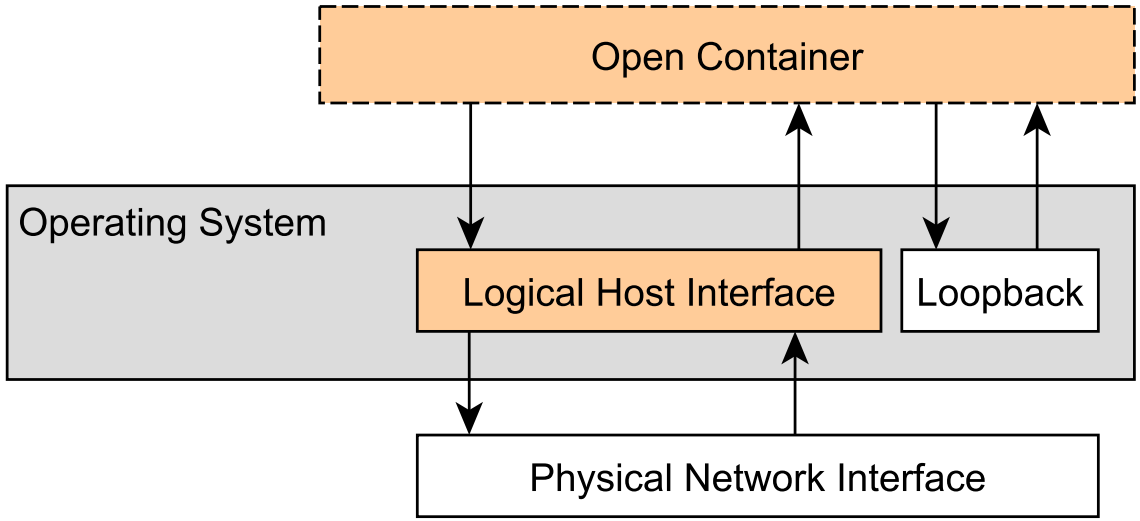
```
docker run --rm --net none --name foo busybox ip addr
docker run --rm --net container:foo --name bar busybox ip addr
```

- Allows access to container's network stack



Open Container

```
docker run --rm --net host busybox ip addr
```



Containers Linking

- Sets env variables describing target container's endpoint (env sharing)
- Adds link allias to the DNS override list
- If inter-container communication (ICC) is disabled adds specific firewall rules to allow communication between containers ("expose" flag)

```
docker run -d --name db -v /var/db/data db:latest

docker create --name=reader1 --link db db-reader:latest
docker create --name=reader2 --link db db-reader:latest
```

Links are deprecated

Use user-defined bridges instead

User-defined bridges

```
docker network create my-net

docker create --name my-nginx --network my-net \
              --publish 8080:80 nginx:latest
```

- Better isolation and interoperability between containerized applications (less open ports)
- Automatic DNS resolution between containers
- Containers can be attached and detached from user-defined networks on the fly (stop/recreate on defaul bridge)
- Linked containers on the user-defined bridge network doesn't share environment variables

User-defined bridges

User-defined network subnet

Always set subnet for user-defined network. By default docker chooses subnet itself and can choose one conflicting with your infrastructure. Hello, IP tables!

```
docker network create stress-test
       --driver=bridge \
       --ipam-driver=default \
       --subnet=172.32.0.0/12
```

Outline

Resource Allowance

- Memmory — the amount of memmory container can use (memory)
- CPU — CPU shares container can use (cpu-shares)
- CPU set — assign CPU for container - cache friendly (cpuset-cpus)
- Devices — allow container to use host's devices (device)
- Shared Memmory — allow IPC (ipc)
- User space — user option

Outline

Feature Authorizations

OS capabilities

When a process makes system calls, the capabilities of that process are checked for the required capability. The required call will succeed if the process has the requireds capability and fail otherwise

- SYS_NICE — Modify priority of processes
- SYS_RESOURCE — Override resource limits
- SYS_TIME — Modify the system clock
- SYS_TTY_CONFIG — Configure TTY devices
- ...

Defaults

By default Docker drops a set of capabilities to enforce isolation from administrative functions (cap-drop/cap-add)

Outline

Full privileges mode

- Use –privileged flag
- Still partially isolated — network namespace still works
- Use –net host to drop network isolation

God mode

Process looks like "uncontainerized" one but you still get all the tooling from Docker related to container management