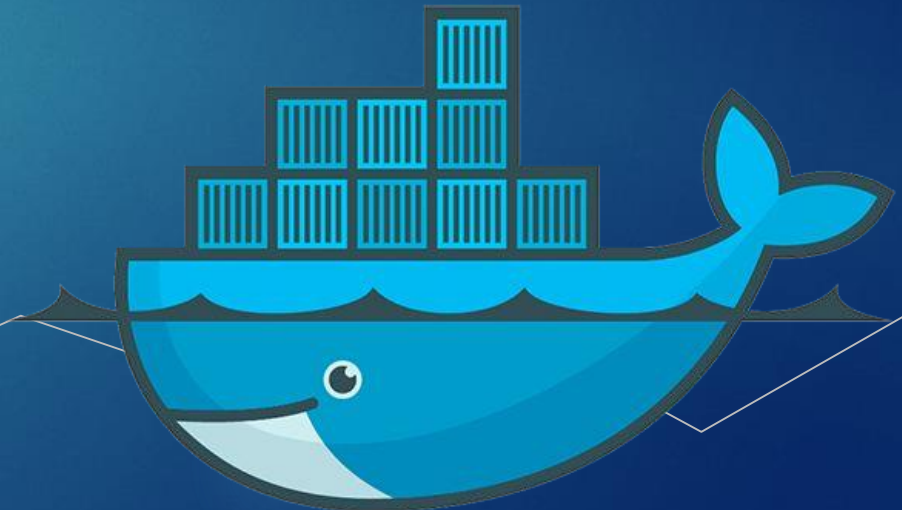




# Docker Advanced

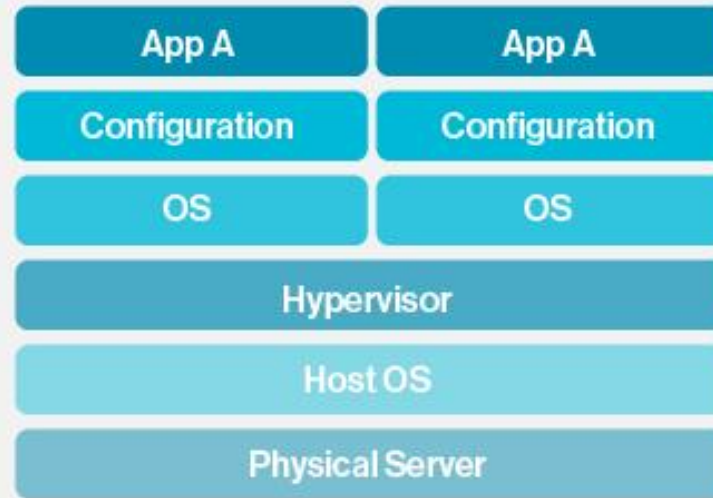
MARIUS MITROFAN

SENIOR IT CONSULTANT



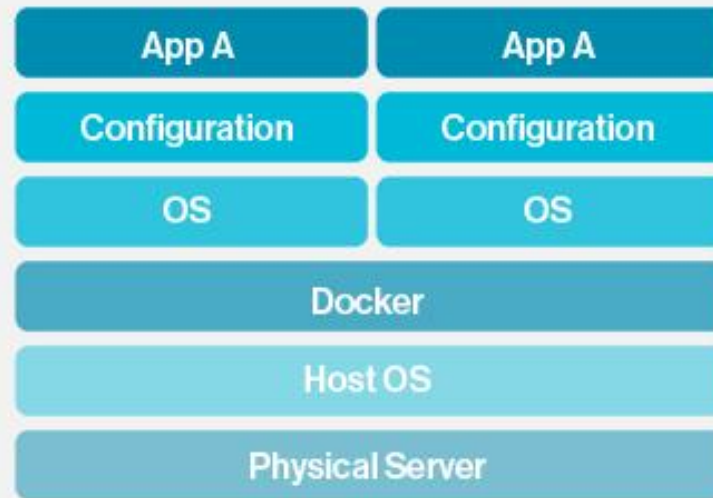
# Why use Docker?

► Docker provides this same capability without the overhead of a virtual machine. It lets you put your environment and configuration into code and deploy it. The same Docker configuration can also be used in a variety of environments. This decouples infrastructure requirements from the application environment.



## Virtual Machine Stack

- Tied to hypervisor version and location
- Each instance has its own OS

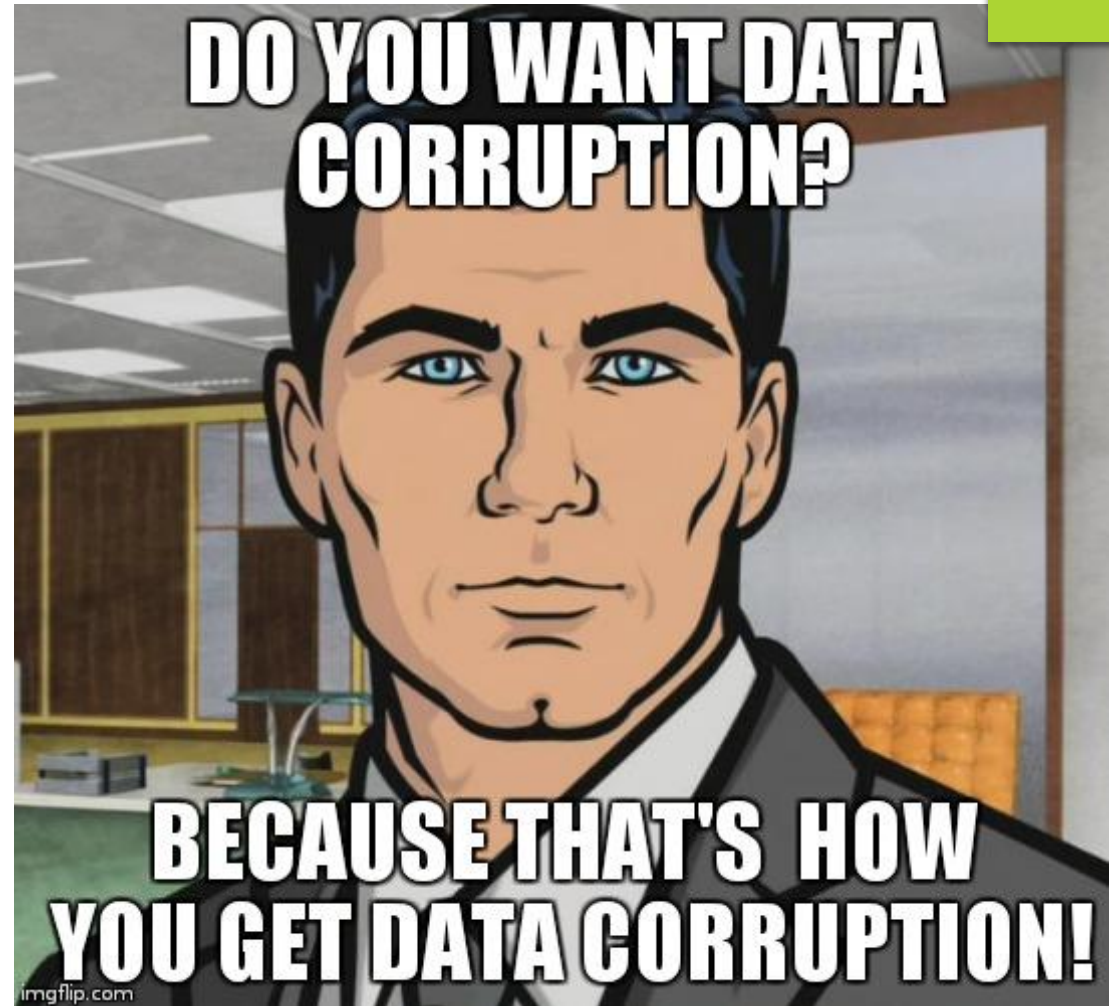


## Docker Stack

- Runs on host OS with host resources
- "Virtualize" just app and dependencies

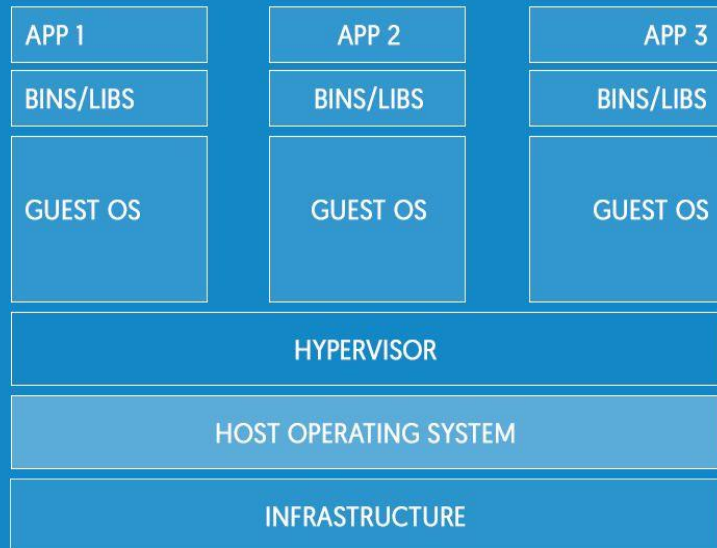
# When NOT to use Docker

- ▶ You don't have container expertise in-house.
- ▶ You need ultra-high security
- ▶ You use Windows Server
- ▶ You need traditional virtualization

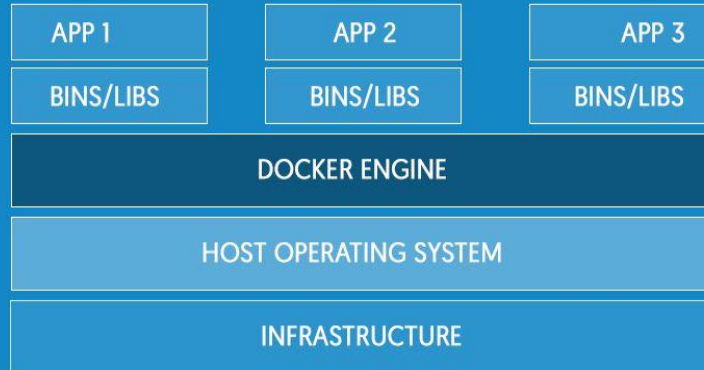


# Virtual Machines VS Containers

## VMs



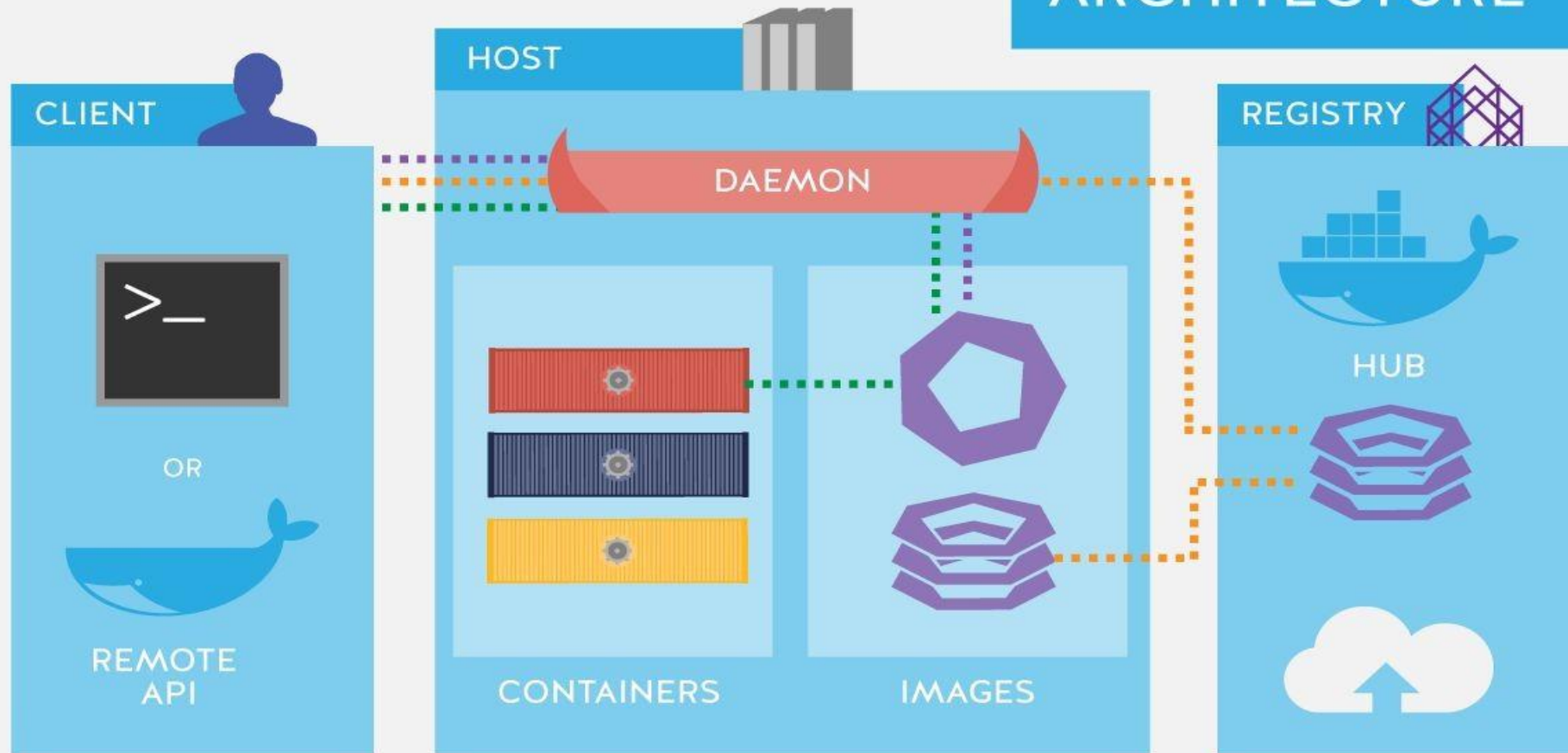
## Containers





# DOCKER ARCHITECTURE

BUILD PULL RUN



NORDICAPIS.COM

# How to install Docker

## Linux

<https://goo.gl/QbcSdv>

```
>> sudo apt-key adv --keyserver  
hkp://p80.pool.sks-  
keyservers.net:80 --recv-keys  
58118E89F3A912897C070ADBF7  
6221572C52609D
```

```
>> sudo apt-add-repository 'deb  
https://apt.dockerproject.org/re  
po ubuntu-xenial main'
```

```
>> sudo apt-get update && sudo  
apt-get install -y docker-engine
```

## Windows 7

<https://goo.gl/VYpPGi>

```
>> Install Docker Toolbox
```

```
>> Pray to God it works  
from the first try
```

## Windows 10


<https://goo.gl/3JgJkB>

```
>> Enable from BIOS Hyper-V
```

```
>> Install Hyper-V packages
```

```
>> Install Docker for  
Windows
```

```
>> Pray to God it works from  
the first try
```



```
$ docker run busybox echo 'Hello World'
Unable to find image 'busybox:latest' locally
latest: Pulling from library/busybox
7520415ce762: Pull complete
Digest: sha256:32f093055929dbc23dec4d03e09dfe971f5973a9ca5cf059cbfb644c206aa83f
Status: Downloaded newer image for busybox:latest
Hello World
```

```
$ docker network ls
```

NETWORK ID	NAME	DRIVER	SCOPE
0c3df316dcb0	bridge	bridge	local
0aac005761c0	host	host	local
bd8e190f5f97	none	null	local



```
$ docker run --name docker-nginx -p 80:80 -d nginx
```

```
Unable to find image 'nginx:latest' locally
```

```
latest: Pulling from library/nginx
```

```
693502eb7dfb: Pull complete
```

```
6decb850d2bc: Pull complete
```

```
c3e19f087ed6: Pull complete
```

```
Digest: sha256:52a189e49c0c797cfc5cbfe578c68c225d160fb13a42954144b29af3fe4fe335
```

```
Status: Downloaded newer image for nginx:latest
```

```
3897847ca9bcc329e94471f734036105fb7df55600fca20b8c8f653642c35135
```

```
$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
3897847ca9bc	nginx	"nginx -g 'daemon ...'"	5 seconds ago	Up 3 seconds	
0.0.0.0:80->80/tcp, 443/tcp		docker-nginx			

```
PS C:\Users\mariu\Documents\Training> docker search ubuntu
```

NAME	DESCRIPTION	STARS	OFFICIAL
ubuntu	Ubuntu is a Debian-based Linux operating s...	5750	[OK]
ubuntu-upstart	Upstart is an event-based replacement for ...	71	[OK]

```
$ docker history nginx
```

IMAGE	CREATED	CREATED BY	SIZE
6b914bbcb89e	3 weeks ago	/bin/sh -c #(nop) CMD ["nginx" "-g" "daem...	0 B
<missing>	3 weeks ago	/bin/sh -c #(nop) EXPOSE 443/tcp 80/tcp	0 B
<missing>	3 weeks ago	/bin/sh -c ln -sf /dev/stdout /var/log/ngi...	22 B
<missing>	3 weeks ago	/bin/sh -c apt-key adv --keyserver hkp://p...	58.8 MB
<missing>	3 weeks ago	/bin/sh -c #(nop) ENV NGINX_VERSION=1.11....	0 B
<missing>	3 weeks ago	/bin/sh -c #(nop) MAINTAINER NGINX Docker...	0 B
<missing>	3 weeks ago	/bin/sh -c #(nop) CMD ["/bin/bash"]	0 B
<missing>	3 weeks ago	/bin/sh -c #(nop) ADD file:41ac8d85ee35954...	123 MB

```
$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
busybox	latest	00f017a8c2a6	2 weeks ago	1.11 MB
nginx	latest	6b914bbcb89e	3 weeks ago	182 MB



```
$ docker exec -ti docker-nginx /bin/bash
root@3897847ca9bc:/# cd /usr/share/nginx/html/
root@3897847ca9bc:/usr/share/nginx/html# ls -altr
total 16
-rw-r--r-- 1 root root 612 Feb 14 15:36 index.html
-rw-r--r-- 1 root root 537 Feb 14 15:36 50x.html
drwxr-xr-x 3 root root 4096 Feb 28 15:15 ..
drwxr-xr-x 2 root root 4096 Feb 28 15:15 .
root@3897847ca9bc:/usr/share/nginx/html# cat index.html
<!DOCTYPE html><html><head><title>Welcome to nginx!</title></head>
<body><h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and working. Further configuration is
required.</p>
<p>For online documentation and support please refer to <a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at <a href="http://nginx.com/">nginx.com</a>.</p>
</body></html>
root@3897847ca9bc:/usr/share/nginx/html#
```

```
root@3897847ca9bc:/usr/share/nginx/html# apt-get update && apt-get install nano -y
```

```
root@3897847ca9bc:/usr/share/nginx/html# exit
```

```
$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
3897847ca9bc	nginx	"nginx -g 'daemon ...'"	21 minutes ago	Up 21 minutes	0.0.0.0:80->80/tcp, 443/tcp
docker-nginx					

```
$ docker commit 3897847ca9bc mmitrofan/cluj-training:latest
```

```
sha256:3873d61fc7e092c29cafd664e29733e4105beb50fdc6b13bcca6d9d6e1ad72b
```

```
$ docker run --name training-cluj-nginx -p 8080:80 -d mmitrofan/cluj-training
```

```
aeed99dd4b9823687670460d53d0ca458a85d94267ac1929ec8fdbda25135e39
```

```
$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
aeed99dd4b98	mmitrofan/cluj-training	"nginx -g 'daemon ...'"	6 seconds ago	Up 6 seconds
443/tcp, 0.0.0.0:8080->80/tcp training-cluj-nginx				
3897847ca9bc	nginx	"nginx -g 'daemon ...'"	24 minutes ago	Up 24 minutes
0.0.0.0:80->80/tcp, 443/tcp docker-nginx				

```
docker cp index.html docker-nginx:/usr/share/nginx/html/index.html
```

```
$ docker run --name docker-nginx -v C:/Users/mariu/Documents/Training:/usr/share/nginx/html:ro -p 80:80 -d nginx
392f38aa25dece03307e3298ce8d661cb182169fb0039f23f98c588a3e04cd25
```

```
$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
392f38aa25de	nginx	"nginx -g 'daemon ...'"	1 second ago	Up 1 second
0.0.0.0:80->80/tcp, 443/tcp		docker-nginx		

```
$ docker volume ls
```

DRIVER	VOLUME NAME
local	f6bb4c912465e83c724740e1d3d6f41c1cbe27ac22067275c7e32ba35f997261



```
$ docker network ls
```

NETWORK ID	NAME	DRIVER	SCOPE
0c3df316dcb0	bridge	bridge	local
0aac005761c0	host	host	local
bd8e190f5f97	none	null	local

```
$ docker network disconnect bridge training-cluj-nginx
```

```
$ docker network connect none training-cluj-nginx
```

```
$ docker inspect training-cluj-nginx
```

# Dockerfile

# Best practices

- ▶ Dockerfiles provide a simple syntax for building images.
- ▶ Here are a some tips and tricks that help to get the most out of Dockerfiles.

## Use the cache

- ▶ Each instruction in a Dockerfile commits the modification into a new image which will then be utilised as the base of the next instruction.
- ▶ If an image exists with an identical parent and instruction docker will utilise the image instead of executing the instruction, i.e. the cache.
- ▶ To effectively utilize the cache we require to keep the Dockerfiles consistent and only add the alterations at the end.

```
FROM      php:5.6.30-apache
ENV       TARGET    /var/www/html
ENV       CONFIG    /tmp
ENV       CHECK_DB  $CONFIG/database_exists.txt

RUN       apt-get update && \
          apt-get install -y \
          build-essential git apache2-dev libmcrypt-dev libicu-dev libssl-dev \
          libpcre3-dev libgtop2-dev libpng12-dev libjpeg62-turbo-dev libfreetype6-dev \
          unzip python-pip python-dev libmysqlclient-dev mysql-client --no-install-recommends && \
          rm -r /var/lib/apt/lists/*

RUN       docker-php-ext-install -j$(nproc) iconv json mcrypt intl phar ctype gd mysql && \
          pecl install xdebug && \
          docker-php-ext-enable --ini-name 0-apc.ini iconv json mcrypt intl phar ctype gd mysql

COPY      public-html/ /var/www/html
COPY      mybb-config/ /tmp/

RUN       pip install -U pip && \
          pip install MySQL-python && \
          chmod +x /tmp/entrypoint.sh

EXPOSE    80

ENTRYPOINT ["/bin/sh","/tmp/entrypoint.sh"]
```



# Use tags

- ▶ Pass the -t option to docker build so that the resulting image is tagged.
- ▶ A simple human readable tag will help to manage what each image was generated for.

# EXPOSE-ing ports

- ▶ Two of the primary concepts of docker are portability and repeatability.
- ▶ Images should be able to run on any host and as many times as required.
- ▶ With Dockerfiles we can map the private and public ports.

# CMD and ENTRYPOINT syntax

- ▶ Both ENTRYPOINT and CMD are straight forward but they consist a hidden, err, "feature" that can cause issues if we are not aware.
- ▶ Two different syntaxes are given below, supported for these instructions.

# FROM

- ▶ We can use various instructions available for use in a Dockerfile.
- ▶ We can use current Official Repositories as the basis for the image.
- ▶ FROM directive is possibly the most crucial amongst all others for Dockerfiles.
- ▶ FROM defines the base image to use to start the build process and it can be any image, comprising the ones someone have generated previously



# LABEL

- ▶ We can add labels to image to help organize images by project, to aid in automation, record licensing information, or for other reasons.
- ▶ For every label, we have to add a line beginning with LABEL and with one or more key-value pairs.

# RUN

- ▶ This command is the central executing directive for Dockerfiles.
- ▶ To make the Dockerfile more readable, maintainable, and understandable, split long or complex RUN statements on multiple lines separated with backslashes.

# CMD

- ▶ This instruction should be utilised to run the software contained by the image, along with any arguments.
- ▶ It should almost always be utilised in the form of CMD ["executable", etc].
- ▶ CMD should be given an interactive shell, such as python, bash and perl.
- ▶ For instance, CMD ["perl", "-deo"], CMD ["python"], or CMD ["php", "-a"].

# ADD vs COPY

- ▶ ADD and COPY are functionally similar, but COPY is more preferred because it is more transparent than ADD.
- ▶ COPY only supports the basic copying of local files into the container.
- ▶ Best use for ADD is local tar file auto-extracted into the image, as in `ADD rootfs.tar.xz /`.
- ▶ ADD can also target remote (`http / https / ftp`) locations

# ENTRYPOINT

- ▶ Best use for ENTRYPOINT is to set the image's main command, enabling that image to be run as though it was that command.
- ▶ An instance of an image for the command line tool s3cmd is given below:



# VOLUME

- ▶ This instruction should be utilised to expose any database storage area, configuration storage, or files/folders generated by the docker container.
- ▶ Utilise VOLUME for any mutable and/or user-serviceable parts of the image.

# USER

- ▶ The USER instruction sets the user name or UID to use when running the image and for any RUN, CMD and ENTRYPOINT instructions that follow it in the Dockerfile.
- ▶ If a service can run without privileges, we can use USER to change to a non-root user.
- ▶ Begin with generating the user and group in the Dockerfile with something such as RUN groupadd -r postgres && useradd -r -g postgres postgres.

# WORKDIR

- ▶ For reliability and clarity, we should always use absolute paths for the WORKDIR.
- ▶ Utilise WORKDIR instead of proliferating instructions such as `RUN cd ... && do-something`, which are hard to read, troubleshoot, and maintain.

# ONBUILD

- ▶ This command executes after the current Dockerfile build completes.
- ▶ It executes in any child image derived FROM the current image.

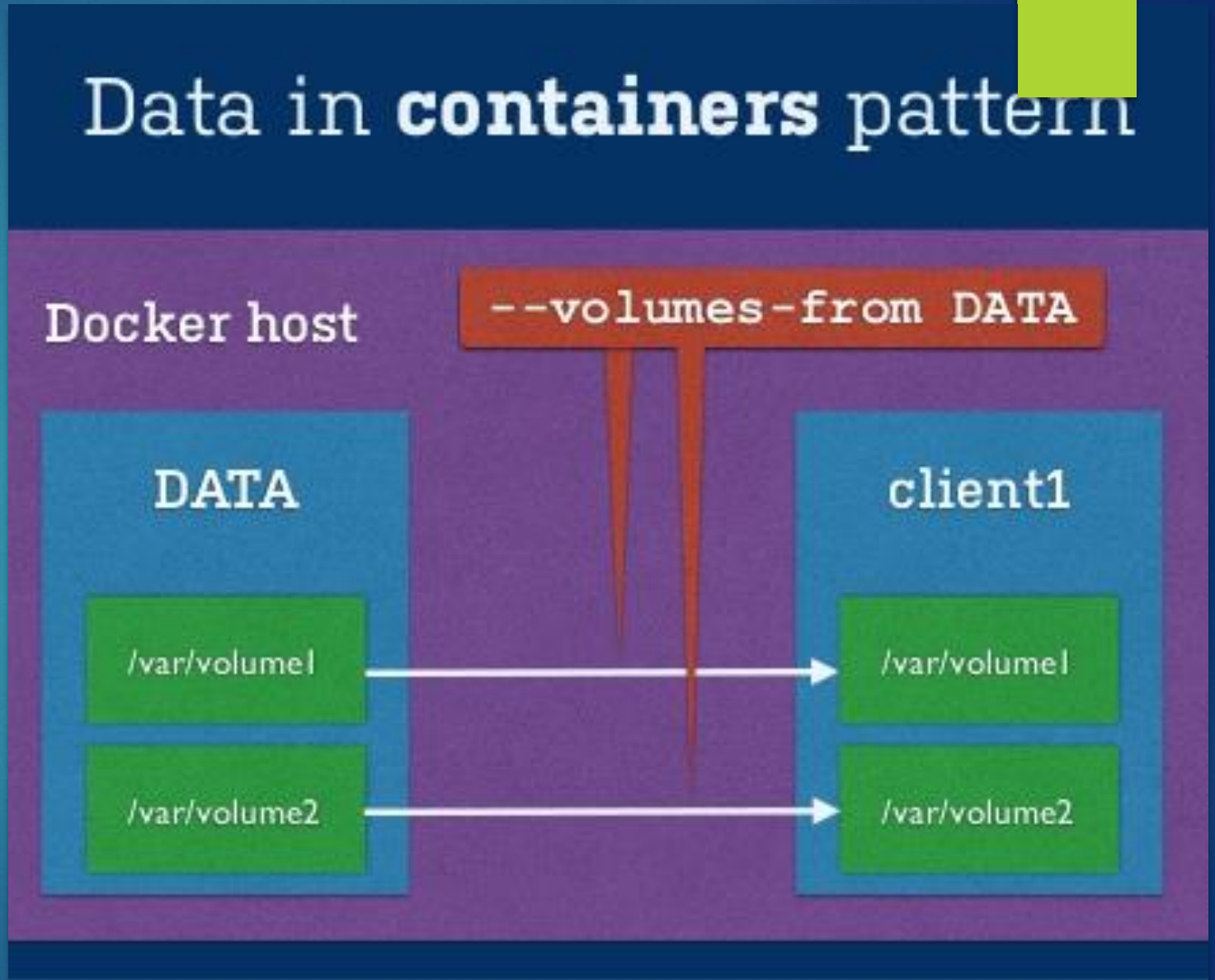
VOLUMES



# Methods to share data

There are two primary ways in which we can manage data with Docker Engine:

- ▶ Data volumes
- ▶ Data volume containers



# Data volumes

- ▶ It is a specially-designated directory within one or more containers that bypasses the Union File System.
- ▶ Data volumes provide many useful features for persistent or shared data:

# Important

- ▶ Modifications to a data volume are made directly.
- ▶ Changes to a data volume will not be comprised when we update an image.
- ▶ Data volumes persist even if the container itself is deleted.

# Locating a volume

- ▶ We can locate the volume on the host using the docker inspect command.
- ▶ The output will provide details on the container configurations comprising the volumes in a JSON format.



# Mount shared volumes into containers

- ▶ To mounting a host directory in container, some Docker volume plugins enable us to provision and mount shared storage, like iSCSI, NFS, or FC.
- ▶ An advantage of utilizing the shared volumes is that they are host-independent which means that a volume can be made available on any host that a container is started on as long as it has access to the shared storage backend, and has the plugin installed.
- ▶ One way to utilize volume drivers is via the Docker run command.



- ▶ Volume drivers create volumes by name, instead of by path like in the other examples.
- ▶ The following command generates a named volume, known as my-named-volume, by using the flocker volume driver and makes it available within the container at /webapp.
- ▶ The example given below to use, the local driver.

```
docker run -d -P \  
  --volume-driver=flocker \  
  -v my-named-volume:/webapp \  
  --name web training/webapp python app.py
```

# Mount a host file as a data volume

- ▶ The -v flag can also be utilized to mount a single file - instead of just directories - from the host machine.
- ▶ It will drop us into a bash shell in a new container, where we can check the bash history from the host.

```
docker run --rm -it \  
    -v ~/.bash_history:/root/.bash_history \  
    ubuntu /bin/bash
```

# Data containers vs Volumes

# Differences

- ▶ Data volumes are designed to persevere data, independent of the container's life cycle.
- ▶ Docker never automatically deletes volumes when we remove a container, nor will it "garbage collect" volumes that are no longer referenced by a container.
- ▶ Data volumes are generated by either using the `--volume` or `-v` flag in `docker run`, or by using the `VOLUME` instruction in the Dockerfile.
- ▶ Both options perform the same task: generate a volume in the container that is mapped to a directory on the host itself.

- ▶ If you do care about the volume's location from the host's perspective, you can use `docker run` with the `-v` flag to request mounting a particular file or directory from the host and, if necessary, mapping the host directory to a container directory.
- ▶ For example, `docker run -v /host/src/demo:/opt/demo` maps a volume so it is known in the host directory as `/host/src/demo` and in the container directory as `/opt/demo`.
- ▶ Data volume containers operate like data volumes but are designed to persist data that you want to share between containers or that you want to use from non-persistent containers.



- ▶ This docker create command starts a data volume container and exist immediately because, instead of an active process like other application containers, it is used as a container shell that references a newly created volume for other containers to use.
- ▶ After the data volume container exits, you can reference it from other containers by using the `--volumes-from` flag on subsequent containers.
- ▶ For example, `docker run -d --volumes-from datastore --name mywebapp demo/webapp` has access to a data volume container named `datastore`.
- ▶ Removing the `datastore` container or the containers that reference it does not delete the volume where your data is stored.

- ▶ To delete the volume from disk, you must explicitly call `docker rm -v` against the last container with a reference to the volume.
- ▶ This allows you to upgrade or effectively migrate data volumes between containers or migrate the containers using the data volume container with no worries of losing the data itself.
- ▶ If you use volumes with containers, you can back up a database or log that lives on the volume by using the same method that you used before implementing containers.
- ▶ After performing a backup, you can upgrade the image by shutting down the previous container and starting the updated one with the same volume or volumes, or you can perform any other routine updates and maintenance.

# Debugging Containers

```
$ docker run -d --name=logtest alpine /bin/sh -c "while true; do sleep 2; df -h; done"
```

```
Unable to find image 'alpine:latest' locally
```

```
latest: Pulling from library/alpine
```

```
627beaf3eaaf: Already exists
```

```
Digest: sha256:58e1a1bb75db1b5a24a462dd5e2915277ea06438c3f105138f97eb53149673c4
```

```
Status: Downloaded newer image for alpine:latest
```

```
3985e7855a013b561a933013c5800618189aad55bfd5004f0332626921a4cd18
```

```
$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
3985e7855a01	alpine	"/bin/sh -c 'while..."	17 seconds ago	Up 16 seconds	
logtest					

```
$ docker logs logtest
```

Filesystem	Size	Used	Available	Use%	Mounted on
overlay	58.8G	2.7G	53.1G	5%	/
tmpfs	990.1M	0	990.1M	0%	/dev
tmpfs	990.1M	0	990.1M	0%	/sys/fs/cgroup
/dev/sda1	58.8G	2.7G	53.1G	5%	/etc/resolv.conf
/dev/sda1	58.8G	2.7G	53.1G	5%	/etc/hostname
/dev/sda1	58.8G	2.7G	53.1G	5%	/etc/hosts
shm	64.0M	0	64.0M	0%	/dev/shm
tmpfs	990.1M	0	990.1M	0%	/proc/kcore
tmpfs	990.1M	0	990.1M	0%	/proc/timer_list
tmpfs	990.1M	0	990.1M	0%	/proc/sched_debug
tmpfs	990.1M	0	990.1M	0%	/sys/firmware

# LOGS

- ▶ The history is available even after the container exits, since its file system is still present on disk.
- ▶ The data is stored in a json file concealed under `/var/lib/docker`.
- ▶ We can get the complete path by using the inspect command.
- ▶ The log command takes options which enable us to follow this file, basically `tail -f`, as well as select how many lines the command returns.
- ▶ The command will return all lines by default.



# Override the ENTRYPOINT

- ▶ Every docker image has an command and entrypoint, whether it is defined in the dockerfile at build time or as an choice to the docker run command at run time.
- ▶ We can use entrypoint and command with many different ways, but there is one setup that follows best practices and gives us a lot of customization ability.
- ▶ An example for entrypoint are as follows: Run a django app. Firstly, define the python process that runs the app as the entry point, it will ensure that it is pid 1 inside the container.

# Process supervision – 1 per container

- ▶ It is a best practice that will definitely make debugging, and reasoning about the state of the container a lot easier.
- ▶ Docker containers are meant to run a single process which is PID 1 inside the sandbox.
- ▶ The main advantage of running one process per container is that it is easier to reason about the state of the container at run time.
- ▶ The container comes up when the process comes up and dies when it dies, and platforms such as Docker Compose and kubernetes.

# Docker Daemon

# Config

- ▶ If we are running the Docker daemon directly by running `dockerd` instead of using a process manager, we can append the configuration options to the `docker run` command directly.
- ▶ The other options can be passed to the Docker daemon to configure it, few of the daemon's options are given below:

Flag	Description
<code>-D, --debug=false</code>	Enable or disable debug mode. By default, this is false.
<code>-H, --host=[]</code>	Daemon socket(s) to connect to.
<code>--tls=false</code>	Enable or disable TLS. By default, this is false.

# Configure Docker with remote server

- ▶ Enable -D mode.
- ▶ Set tls to true with the server certificate and key specified by using --tlscert and --tlskey respectively.
- ▶ Listen for connections on tcp://192.168.59.3:2376.

```
$ dockerd \  
-D \  
--tls=true \  
--tlscert=/var/docker/server.pem \  
--tlskey=/var/docker/serverkey.pem \  
-H tcp://192.168.59.3:2376
```



# Docker Security

# Docker security

- ▶ The intrinsic security of the kernel and its support for namespaces and cgroups.
- ▶ The attack surface of the Docker daemon itself.
- ▶ Loopholes in the container configuration profile, either by default, or when customized by users.
- ▶ The “hardening” security features of the kernel and how they interact with containers.

# Docker security

- ▶ They are another key element of Linux Containers.
- ▶ Control groups implement resource accounting and limiting.
- ▶ Control groups provide several useful metrics, but they also help to make sure that each container gets its fair share of memory, CPU, disk I/O
- ▶ They do not play a role in averting one container from accessing or affecting the data and processes of another container, they are important to fend off few denial-of-service attacks

# Docker daemon attack surface

- ▶ Running containers with Docker implies running the Docker daemon.
- ▶ This daemon currently requires root privileges, and you should therefore be aware of some important details.
- ▶ Only trusted users should be allowed to control the Docker daemon.
- ▶ It is a direct consequence of some powerful Docker facets.

# Docker daemon attack surface

- ▶ Docker enable us to share a directory between the Docker host and a guest container and it also enable us to do so without limiting the access rights of the container.
- ▶ So we can start a container where the /host directory will be the / directory on the host and the container will be able to alter the host filesystem without any restriction.
- ▶ It is similar to how virtualization systems enable filesystem resource sharing.
- ▶ Nothing prevents from sharing the root filesystem with a virtual machine.



# Linux kernel capabilities

- ▶ By default, docker starts containers with a restricted set of abilities.
- ▶ The capabilities turn the binary “root/non-root” dichotomy into a fine-grained access control system.
- ▶ Processes that just require to bind on a port below 1024 do not have to run as root: they can just be granted the `net_bind_service` capability instead.
- ▶ There are many other proficiencies, for almost all the specific areas where root privileges are usually required.

# Docker security

- ▶ Average server requires to run a bunch of processes as root.
- ▶ Those typically comprise SSH, cron, syslogd, hardware management tools, network configuration tools, and much more.
- ▶ A container is very much different, because almost all of those tasks are handled by the infrastructure around the container:
- ▶ SSH access will be managed by a single server running on the Docker host.
- ▶ It should run as a user process, dedicated and tailored for the app that requires its scheduling service, rather than as a platform-wide facility

# Docker security

- ▶ Log management will also be handed to Docker, or by third-party services like Loggly or Splunk.
- ▶ Hardware management is irrelevant, which means we never need to run udevd or equivalent daemons within containers.
- ▶ Network management happens outside of the containers, it enforcing separation of concerns as much as possible, which means a container should never need to perform ifconfig, route, or ip commands.

# Private registries

# Private Registry

- ▶ Docker considers a private registry either secure or insecure.
- ▶ Registry is utilized for private registry, and myregistry:5000 is a placeholder an example for a private registry.
- ▶ A secure registry utilizes TLS and a copy of its CA certificate is placed on the Docker host at `/etc/docker/certs.d/myregistry:5000/ca.crt`.
- ▶ An insecure registry is either not utilizing TLS, or is utilizing TLS with a CA certificate not known by the Docker daemon.



# Private Registry

- ▶ By default, Docker assumes all, but local, registries are secure.
- ▶ Communicating with an insecure registry is not possible if Docker assumes that registry is secure.
- ▶ To communicate with an insecure registry, the Docker daemon needs `--insecure-registry` in one of the following two forms:
- ▶ The flag can be utilized multiple times to enable multiple registries to be marked as insecure.

```
--insecure-registry myregistry:5000  
--insecure-registry 10.1.0.0/16
```

```
docker run \  
    -d -p 5000:5000 \  
    --restart=always \  
    --name registry \  
    -v `pwd`/data:/var/lib/registry \  
    registry:2
```

# Microservices Architecture

Docker was developed by Docker Inc as a open-source engine for deployment of applications on containers.	By providing a logical layer that manages the lifecycle of the containers, we can focus our development on the applications themselves, leaving the implementation of the container management to Docker.
The Docker architecture consists of a client-server model, where we have a Docker client, that could be the command line one provided by Docker, or a consumer of the RESTFul API also provided on the toolset	The Docker server, also known as Docker daemon, which receives requests to create/start/stop containers, been responsible for the container management.

# Containers & Images

- ▶ Containers on Docker plays important role for processes running from instructions that were set on images.
- ▶ We can comprehend images as the building blocks from which containers are build.
- ▶ These images are distributed on repositories, also known as a Docker registry, which in turn are versioned by using GIT.
- ▶ The main repository for the distribution of Docker images is, the Docker Hub, managed by Docker itself.
- ▶ The code behind the Docker Hub is open, so it is perfectly possible to host the own image repository.



# Docker Union File System

- ▶ Docker utilizes the file system which is known as Union File System.
- ▶ In it each image is layered as a read-only layer.
- ▶ The layers are then overlapped on top of each other and finally on top of the chain a read-write layer is generated, for the use of the container.