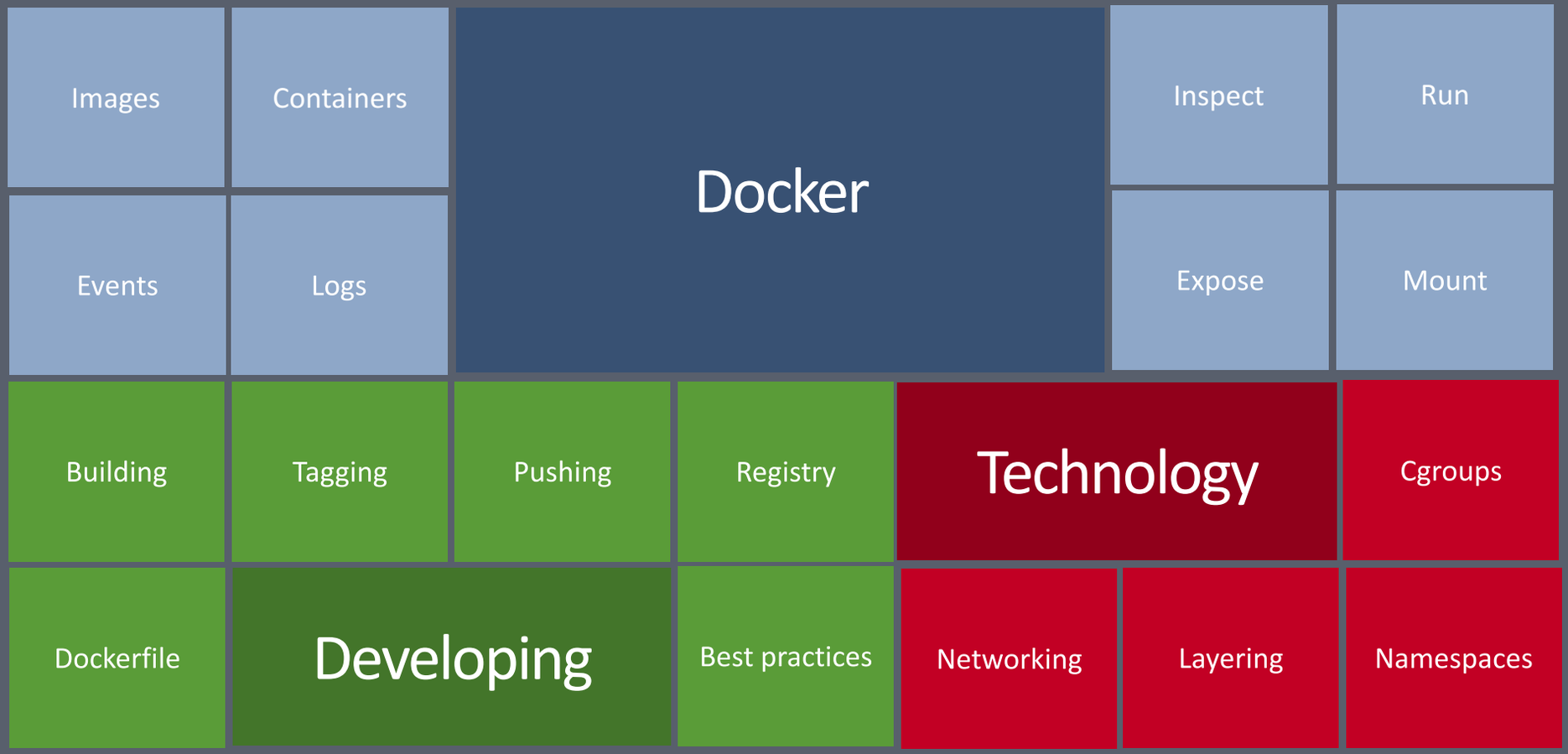
KRAMP

Docker

Arjen Wassink

Quintor

Inhoud


Images
Containers
Docker
Inspect
Run
Events
Logs
Expose
Mount
Building
Tagging
Pushing
Registry
Technology
Cgroups
Dockerfile
Developing
Best practices
Networking
Layering
Namespaces

docker

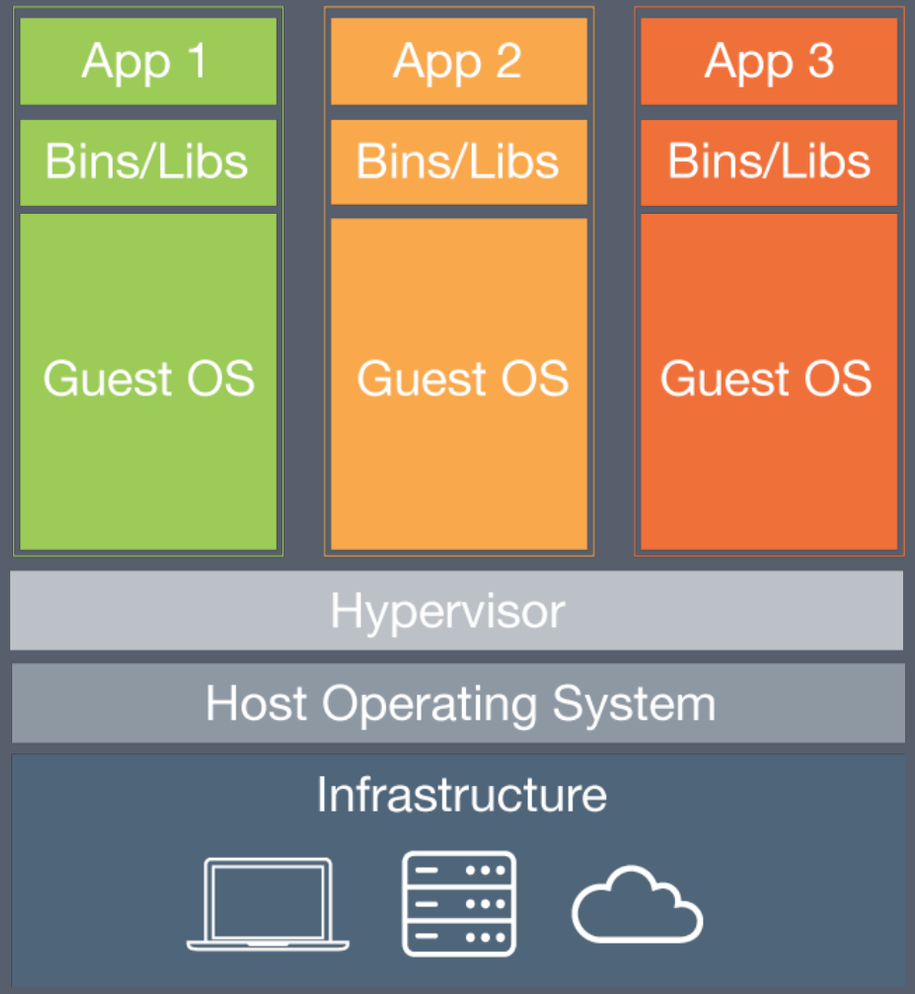
Build

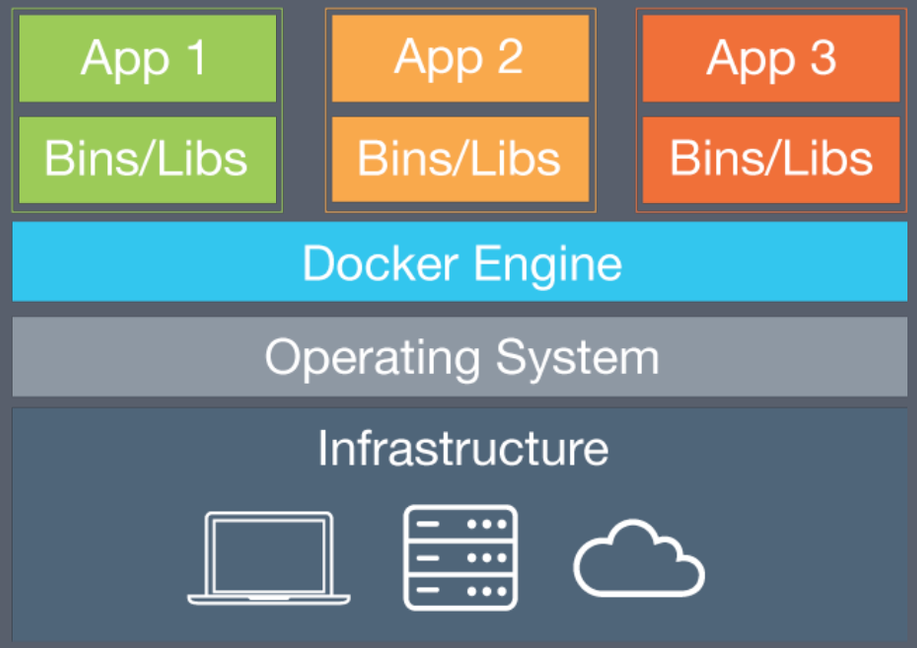
Ship

Run

Docker - Adopters

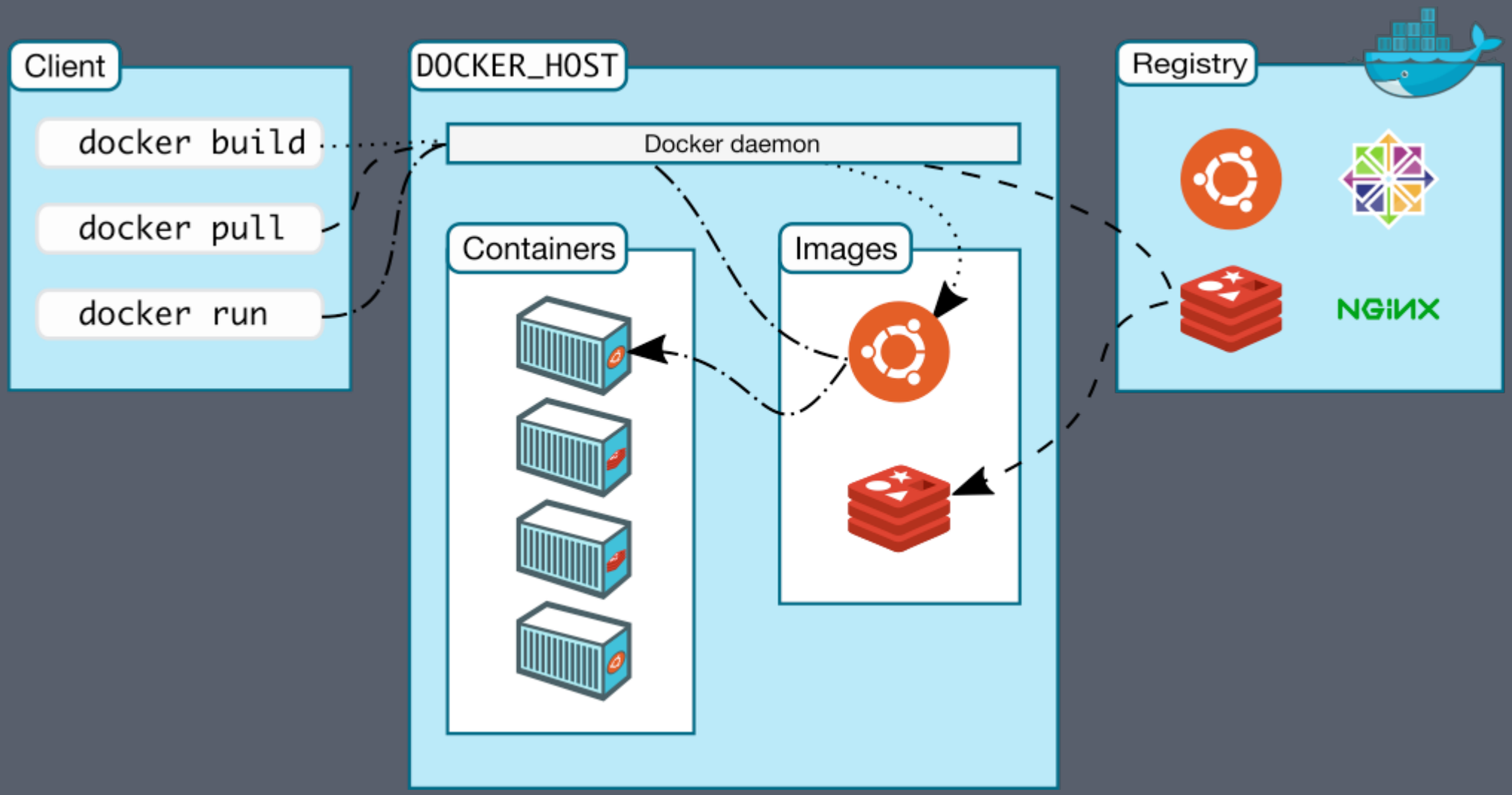
Containers vs. Virtual Machines





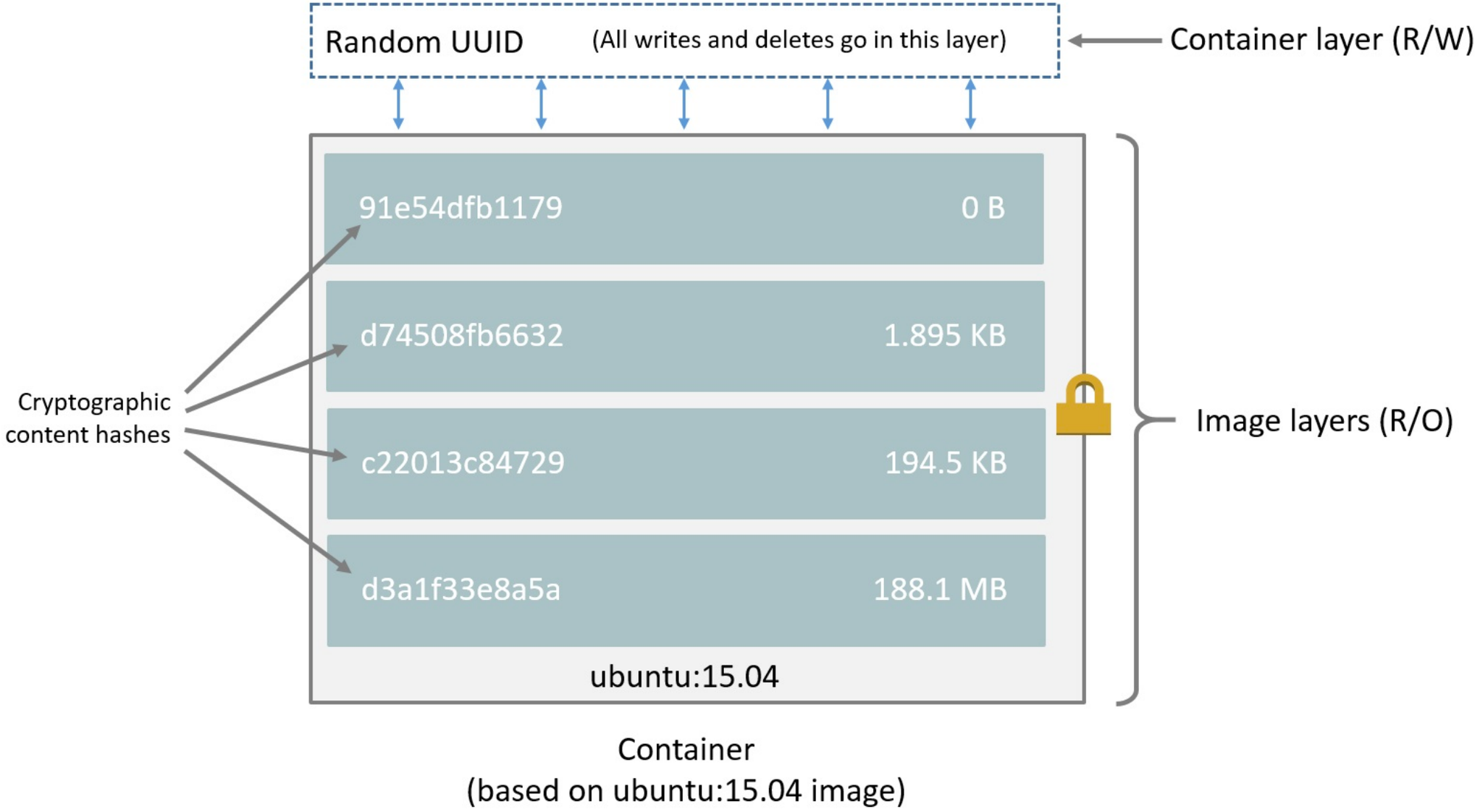
http://www.zdnet.com/article/what-is-docker-and-why-is-it-so-darn-popular/

Docker - Architecture

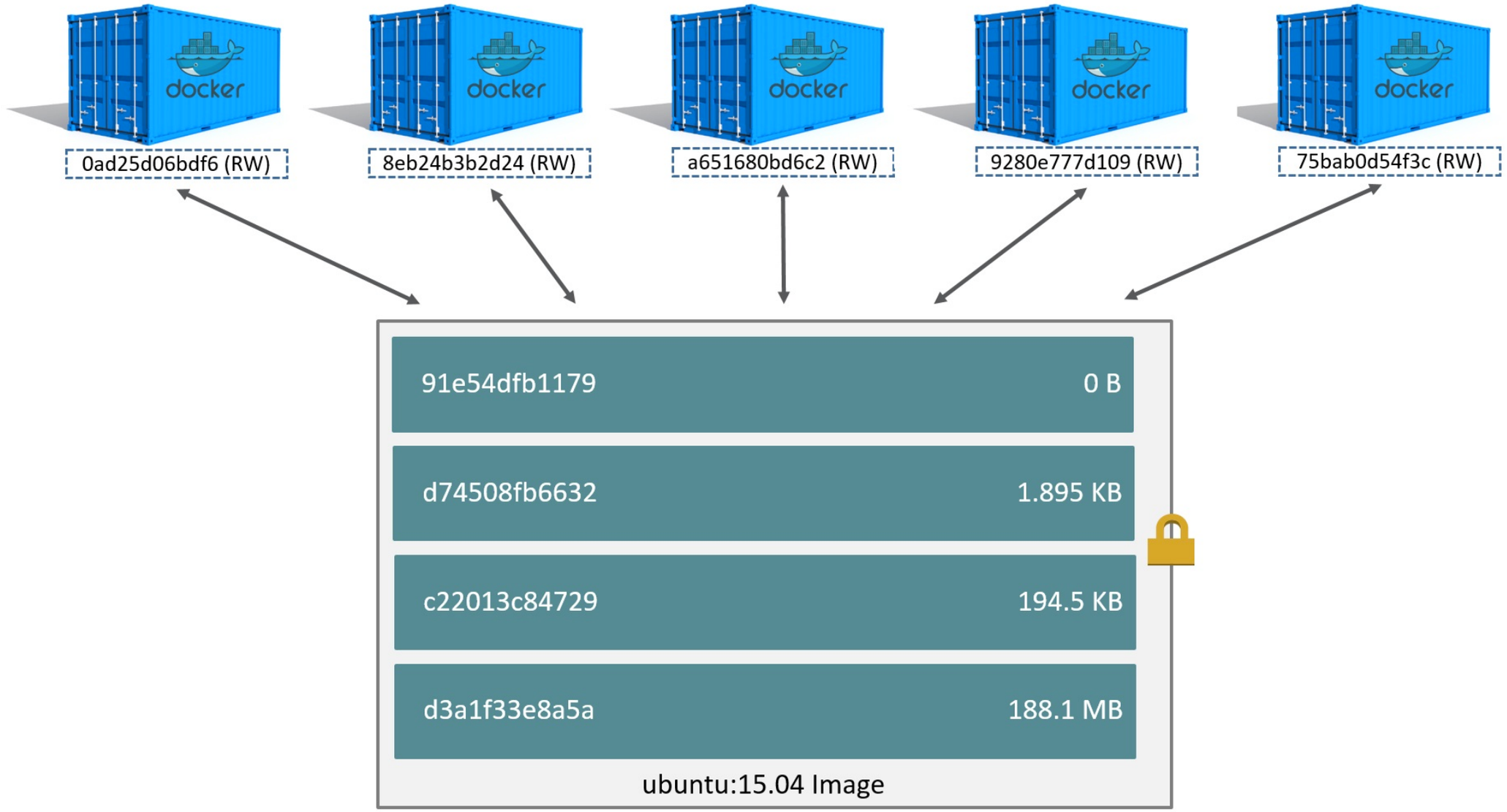


https://docs.docker.com/engine/introduction/understanding-docker/

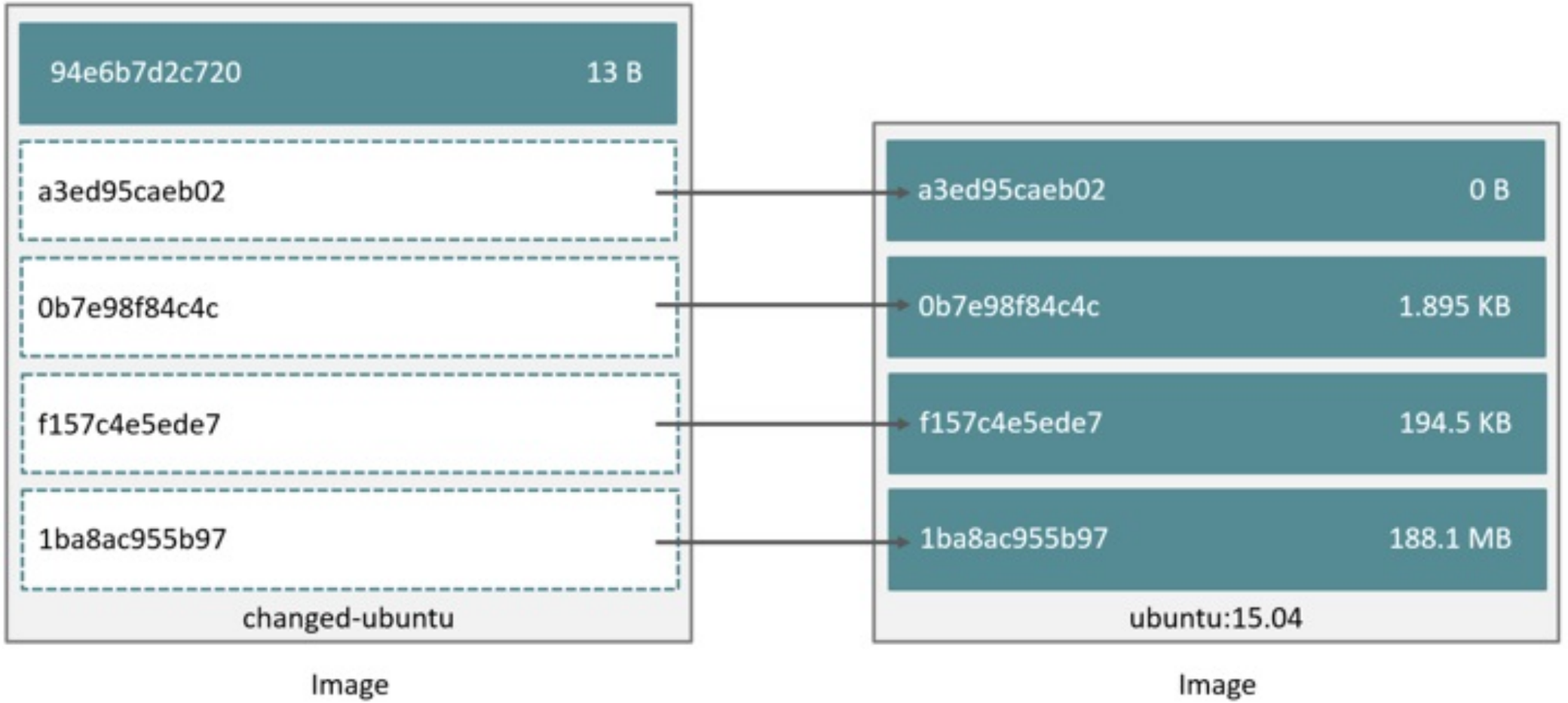
Docker – Images and containers



Docker – Images and containers



Docker - Image layering



Docker - Image layering

```
$ docker pull nginx
Using default tag: latest
latest: Pulling from library/nginx
6d827a3ef358: Pull complete
f8f2e0556751: Pull complete
5c9972dca3fd: Pull complete
451b9524cb06: Pull complete
Digest:
sha256:e6693c20186f837fc393390135d8a598a96a833917917789d63766cab6c59582
Status: Downloaded newer image for nginx:latest
$ docker pull debian:jessie
jessie: Pulling from library/debian
6d827a3ef358: Already exists
Digest:
sha256:72f784399fd2719b4cb4e16ef8e369a39dc67f53d978cd3e2e7bf4e502c7b793
Status: Downloaded newer image for debian:jessie
```

Running Containers

```
$ docker run [OPTIONS] IMAGE[:TAG|@DIGEST] [COMMAND] [ARG...]
```

- Detached vs foreground
  - Detached (-d)
    - To reattach to a detached container, use docker attach command.
  - Foreground (by default)
    - -a=[] : Attach to `STDIN`, `STDOUT` and/or `STDERR`
    - -t : Allocate a pseudo-tty
    - -i : Keep STDIN open even if not attached
- Container identification
  - UUID identifiers (short and long) come from the Docker daemon
    - Long: “f78375b1c487e03c9438c729345e54db9d20cfa2ac1fc3494b6eb60872e74778”
    - Short: “f78375b1c487 ”
  - Name - the daemon generates a random string name
    - Use the --name option to specify a specific but unique name for the container
  - PID equivalent (--cidfile)
    - have Docker write the container ID out to a file of your choosing

Running Containers

```
$ docker run -d nginx
294eb348eb80b4330cce24862973f1329ce837f0d73814779b97a3d594a23044
$ docker ps
CONTAINER ID   IMAGE   COMMAND       STATUS          NAMES
294eb348eb80   nginx   "nginx ..."   Up 10 seconds   youthful_minsky

$ docker run -d --name my_nginx nginx
ea8a87e02a00e80fed88cc48aaaf282eafc9473c601de75bcdb846c5d153d258
$ docker ps
CONTAINER ID   IMAGE   COMMAND       STATUS          NAMES
ea8a87e02a00   nginx   "nginx ..."   Up 7 seconds    my_nginx
294eb348eb80   nginx   "nginx ..."   Up 9 minutes    youthful_minsky
```

Running Containers – Command

```
$ docker run [OPTIONS] IMAGE[:TAG|@DIGEST] [COMMAND] [ARG...]
```

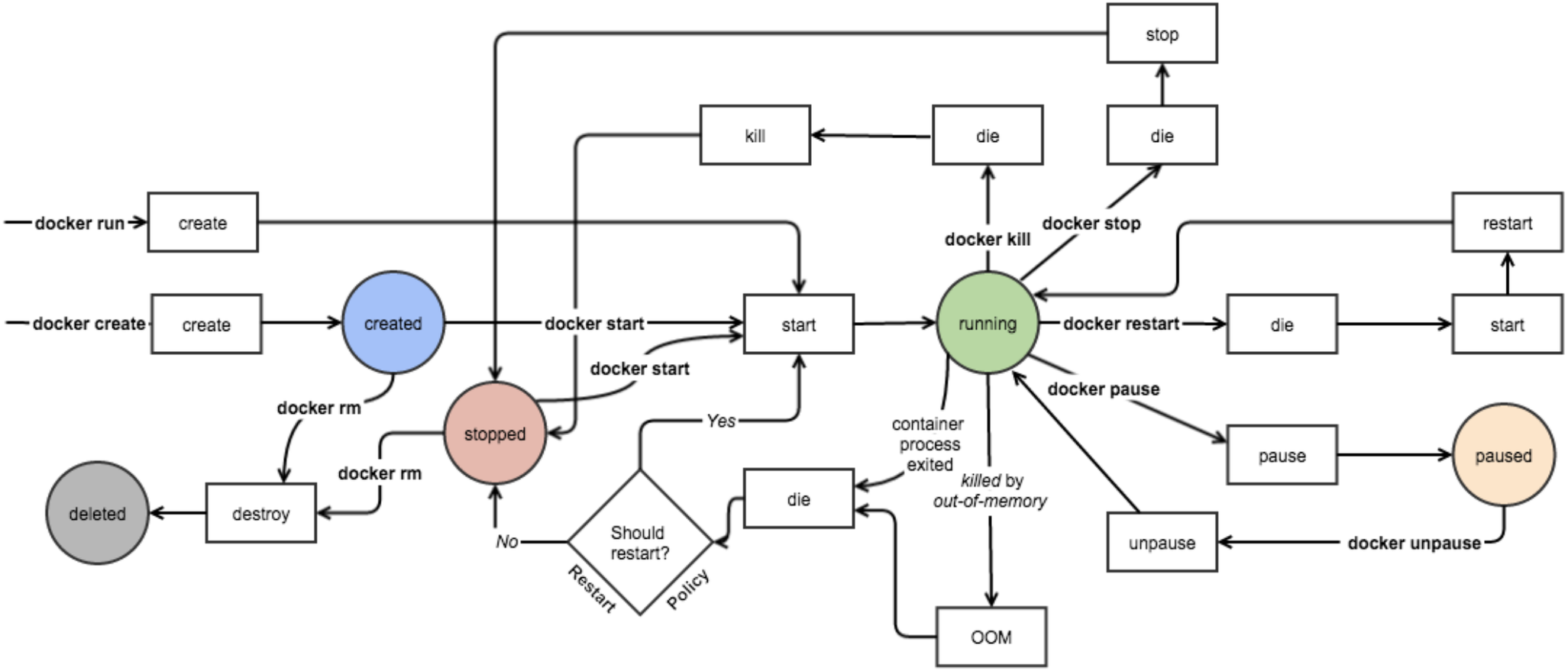
- CMD (default command or options)
  - The default COMMAND can be specified using the Dockerfile CMD instruction in the IMAGE
  - Override that CMD instruction just by specifying a new COMMAND
- ENTRYPOINT (default command to execute at runtime)
  - The ENTRYPOINT of an IMAGE specifies what executable to run when the container starts
  - --entrypoint="": Overwrite the default ENTRYPOINT set by the IMAGE
- ARG…
  - Arbitrary list of arguments for the actual command

Running Containers – Restart, Exit, Clean

```
$ docker run [OPTIONS] IMAGE[:TAG|@DIGEST] [COMMAND] [ARG...]
```

- Restart policies
  - --restart="no" (default) — Do not automatically restart the container when it exits.
  - --restart="on-failure[:max-retries]" — Restart only if the container exits with a non-zero exit status.
  - --restart="always" — Always restart the container regardless of the exit status.
  - --restart="unless-stopped" — Always restart the container regardless of the exit status, but do not start it on daemon startup if the container has been put to a stopped state before.
- Exit codes
  - *125* if the error is with Docker daemon *itself*
  - *126* if the *contained command* cannot be invoked
  - *127* if the *contained command* cannot be found
  - *Exit code* of *contained command* otherwise
- Clean up (--rm)
  - Automatically remove the container when it exits (incompatible with -d)

Container States & Runtime Events



Processes in a completely isolated container…

Running Containers – Ports and Volumes

```
$ docker run [OPTIONS] IMAGE[:TAG|@DIGEST] [COMMAND] [ARG...]
```

- Exposing ports
  - `-p, --publish list`
  - Publish a container's port(s) to the host (default [])
  - `[host_ip:]host_port:container_port`
  - Example: `$ docker run -d -p 8888:80 nginx`
- Sharing volumes
  - `-v, --volume list`
  - Bind mount a volume (default [])
  - `host_volume:container_volume[:ro|rw]`
  - Example: `$ docker run -d -v /some/content:/usr/share/nginx/html:ro nginx`

Helpful Docker Commands

```
$ docker exec [OPTIONS] CONTAINER COMMAND [ARG...]
```

- Run a command in a running container
- Example to enter a container with a shell: `$ docker exec -ti my_nginx bash`

```
$ docker logs [OPTIONS] CONTAINER
```

- Fetch the logs of a container
- Example: `$ docker logs my_nginx`

```
$ docker inspect [OPTIONS] NAME|ID [NAME|ID...]
```

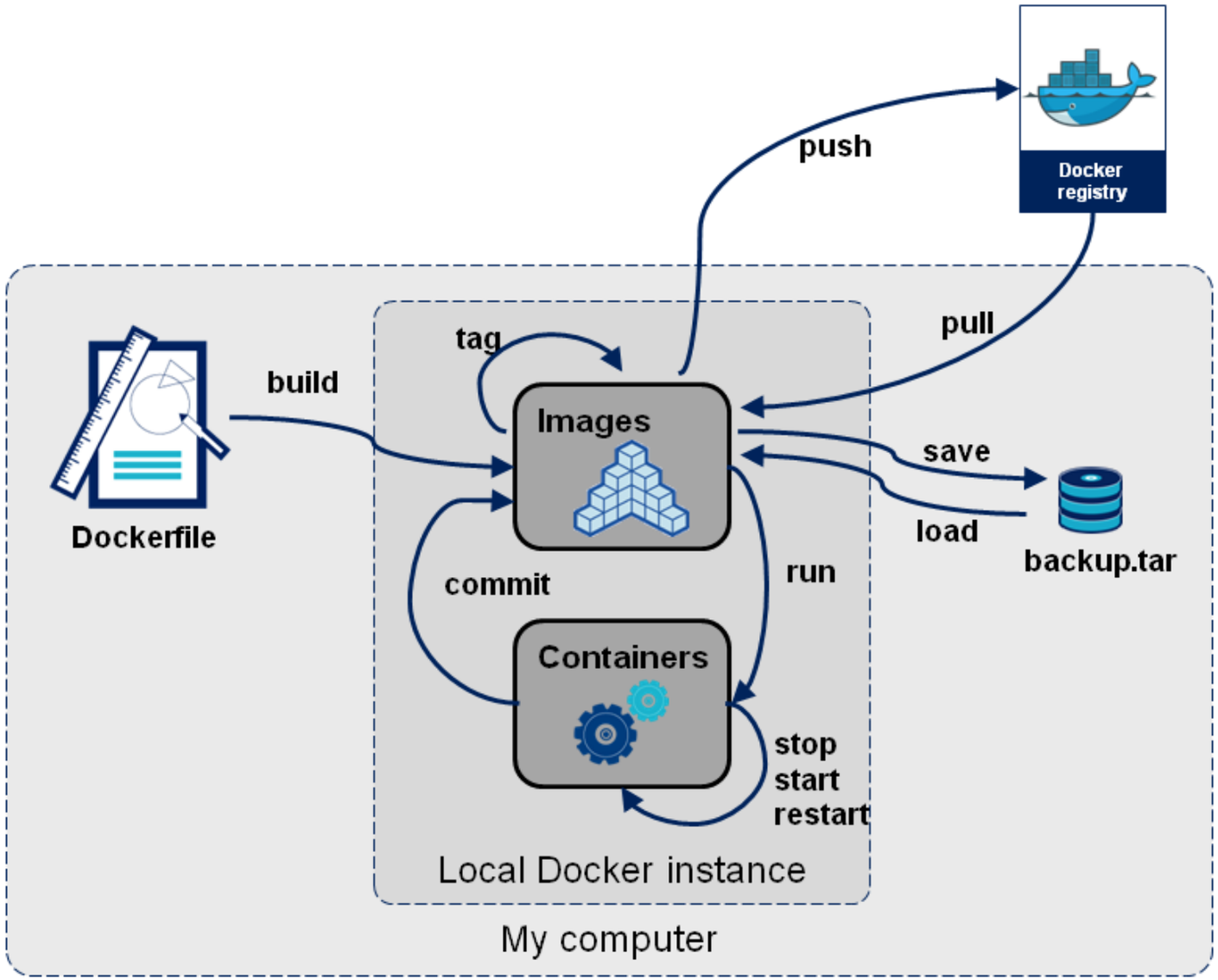
- Return low-level information on Docker objects
- Example: `$ docker inspect my_nginx`

Exercise 1 – Working with images and containers

https://github.com/awassink/docker-training

Building Docker Images

Developing Docker Images



Dockerfile

```
FROM debian:jessie

MAINTAINER NGINX Docker Maintainers "docker-maint@nginx.com"

ENV NGINX_VERSION 1.10.2-1~jessie

RUN apt-key adv --keyserver hkp://pgp.mit.edu:80 --recv-keys 573BFD6B3D8FBC641079A6ABABF5BD827BD9BF62 \
    && echo "deb http://nginx.org/packages/debian/ jessie nginx" >> /etc/apt/sources.list \
    && apt-get update \
    && apt-get install --no-install-recommends --no-install-suggests -y \
                        ca-certificates \
                        nginx=${NGINX_VERSION} \
                        nginx-module-xslt \
                        nginx-module-geoip \
                        nginx-module-image-filter \
                        nginx-module-perl \
                        nginx-module-njs \
                        gettext-base \
    && rm -rf /var/lib/apt/lists/*

# forward request and error logs to docker log collector
RUN ln -sf /dev/stdout /var/log/nginx/access.log \
    && ln -sf /dev/stderr /var/log/nginx/error.log

EXPOSE 80 443

CMD ["nginx", "-g", "daemon off;"]
```

Dockerfile - FROM

```
FROM IMAGE[:TAG|@DIGEST]
```

- The FROM instruction sets the Base Image for subsequent instructions.
- FROM must be the first non-comment instruction in the Dockerfile.
- FROM can appear multiple times within a single Dockerfile in order to create multiple images. Simply make a note of the last image ID output by the commit before each new FROM command.
- The tag or digest values are optional. If you omit either of them, the builder assumes a latest by default.
- FROM scratch - an explicitly empty image, especially for building images "FROM scratch"
  - Useful in the context of building base images and or super minimal images
  - FROM scratch is a no-op in the Dockerfile, and will not create an extra layer in your image
  - Scratch can’t be pulled from a registry!

Dockerfile - RUN

`RUN <command>` (*shell* form)

`RUN ["executable", "param1", "param2"]` (*exec* form)

- The RUN instruction will execute any commands in a new layer on top of the current image and commit the results.
  - The resulting committed image will be used for the next step in the Dockerfile.
- In the *shell* form, the command is run in a shell, which by default is /bin/sh -c on Linux
  - use a \ (backslash) to continue a single RUN instruction onto the next line
- The *exec* form makes it possible to avoid shell string munging, and to RUN commands using a base image that does not contain the specified shell executable

Dockerfile – COPY & ADD

```
COPY|ADD <src>... <dest>
COPY|ADD ["<src>",... "<dest>"]
```
(required for paths containing whitespace)

- The COPY or ADD instruction copies new files, directories or remote file URLs from <src> and adds them to the filesystem of the image at the path <dest>.
  - The ADD instruction allows you to use a URL as the <src> parameter
  - Another feature of ADD is the ability to automatically unpack compressed files
- The COPY instruction is the preferred over the ADD instruction.
- Multiple <src> resource may be specified but if they are files or directories then they must be relative to the source directory
- The <dest> is an absolute path, or a path relative to WORKDIR

Dockerfile - CMD

`CMD ["executable","param1","param2"]` (*exec* form, this is the preferred form)
`CMD ["param1","param2"]` (as *default parameters to ENTRYPOINT*)
`CMD command param1 param2` (*shell* form)

- The main purpose of a CMD is to provide defaults for an executing container.
- There can only be one CMD instruction in a Dockerfile
  - If you list more than one CMD then only the last CMD will take effect.

Dockerfile - ENTRYPOINT

`ENTRYPOINT ["executable", "param1", "param2"]` (*exec* form, preferred)
`ENTRYPOINT command param1 param2` (*shell* form)

- An ENTRYPOINT allows you to configure a container that will run as an executable.
  - Command line arguments to docker run <image> will be appended after all elements in an *exec* form ENTRYPOINT, and will override all elements specified using CMD.
- Only the last ENTRYPOINT instruction in the Dockerfile will have an effect.
- The *shell* form prevents any CMD or run command line arguments from being used
  - has the disadvantage that your ENTRYPOINT will be started as a subcommand of /bin/sh -c, which does not pass signals. This means that the executable will not be the container's PID 1 - and will *not* receive Unix signals - so your executable will not receive a SIGTERM from docker stop <container>.
- Use the *exec* form of ENTRYPOINT to set fairly stable default commands and arguments
  - Use CMD to set additional defaults that are more likely to be changed

Dockerfile - WORKDIR

```
WORKDIR /path/to/workdir
```

- The WORKDIR instruction sets the working directory for any RUN, CMD, ENTRYPOINT, COPY and ADD instructions that follow it in the Dockerfile.
- If the WORKDIR doesn’t exist, it will be created even if it’s not used in any subsequent Dockerfile instruction.
- Can be used multiple times in the one Dockerfile.

Dockerfile – ENV

```
ENV <key> <value>
ENV <key>=<value> ...
```

- The ENV instruction sets the environment variable <key> to the value <value>.
  - This value will be in the environment of all “descendant” Dockerfile commands
- The first form, ENV <key> <value>, will set a single variable to a value.
  - The entire string after the first space will be treated as the <value> - including characters such as spaces and quotes.
- The second form, ENV <key>=<value> ..., allows for multiple variables to be set at one time.

Dockerfile – EXPOSE & VOLUME

```
EXPOSE <port> [<port>...]
VOLUME ["/data"]
```

- The EXPOSE instruction informs Docker that the container listens on the specified network ports at runtime.
  - EXPOSE does not make the ports of the container accessible to the host.
- The VOLUME instruction creates a mount point with the specified name and marks it as holding externally mounted volumes from native host or other containers.
- The docker run command initializes the newly created volume with any data that exists at the specified location within the base image.

Best practices for writing Dockerfiles

- Containers should be ephemeral
- Use a .dockerignore file
- Avoid installing unnecessary packages
- Each container should have only one concern
- Minimize the number of layers
- Use the build cache

Docker – Build Image

```
docker build [OPTIONS] PATH | URL | -
```

- --build-arg list Set build-time variables (default [])
- -f, --file string Name of the Dockerfile (Default is 'PATH/Dockerfile')

Publishing Docker images

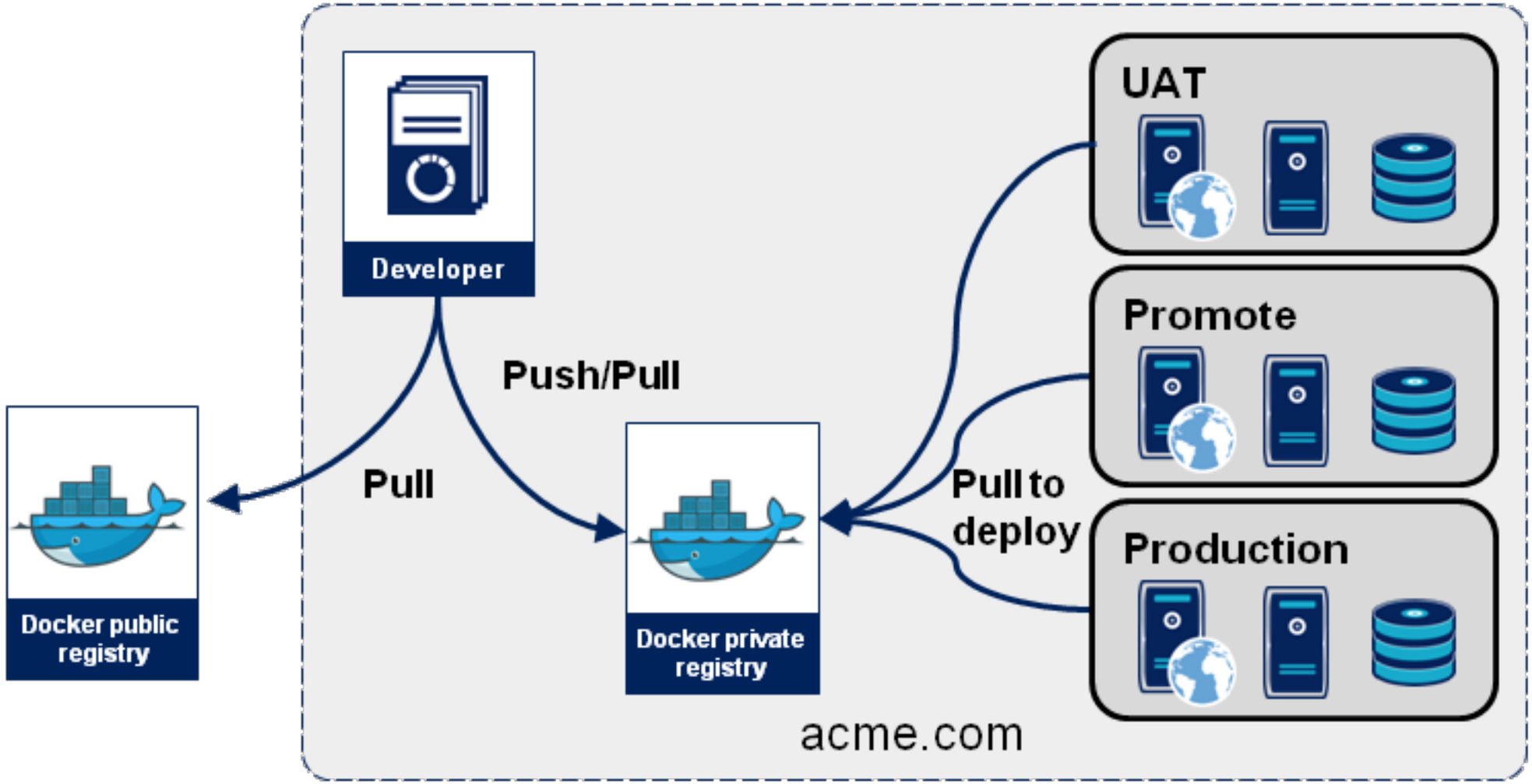


Image Naming

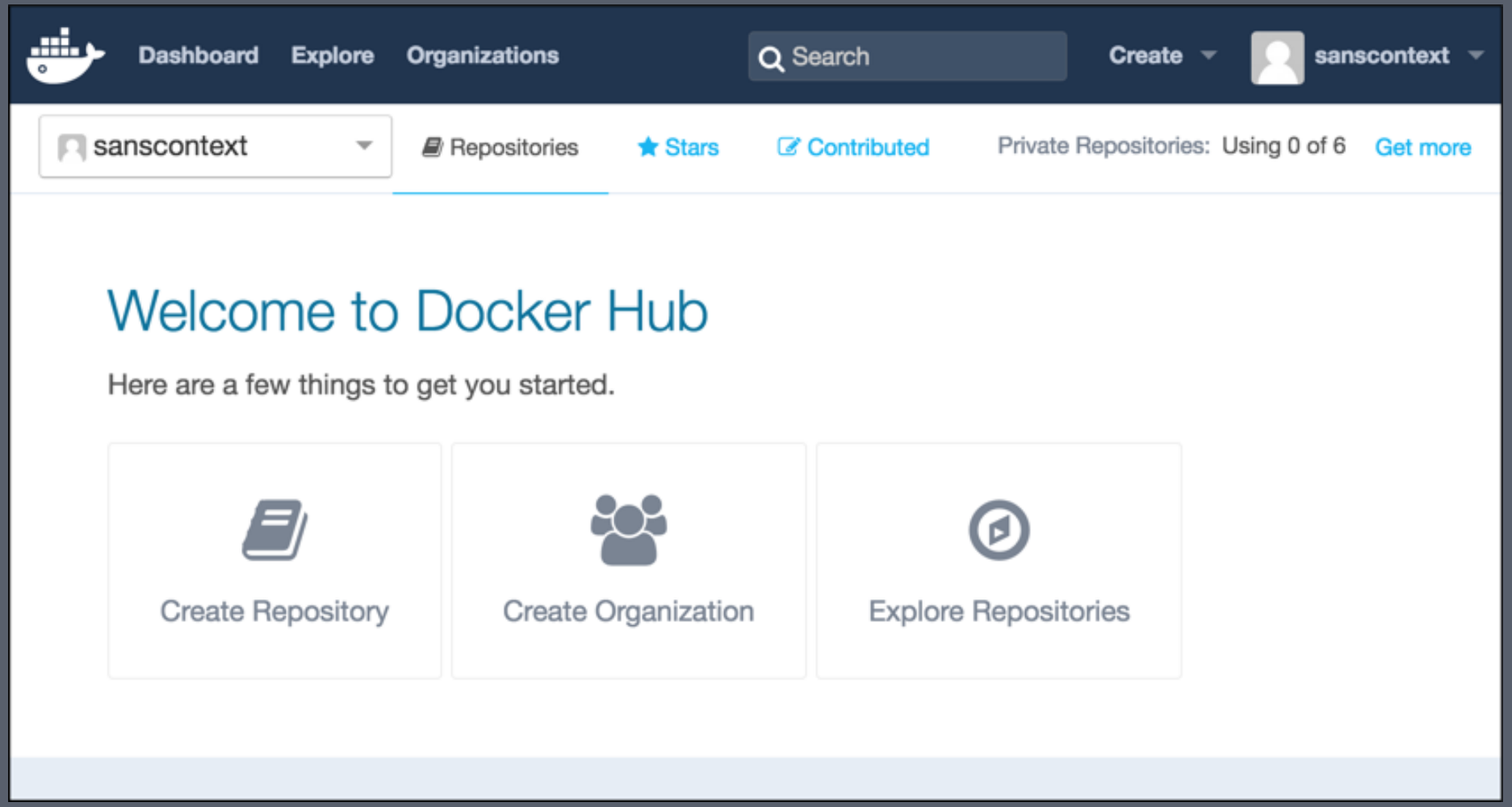
`[REGISTRY[:PORT]/][USER/]REPOSITORY[:TAG]`

- The default REGISTRY is index.docker.io which serves the images of hub.docker.com
- When using a private registry specify it's hostname and optional PORT number
- The USER indicates the optional registry account
- The REPOSITORY identifies the image
- The TAG identifies the image version

- Use docker tag to name an image
- Note that when pushing an image its name must match the targeted registry and user
- Also the user must be logged in to the registry. Use docker login

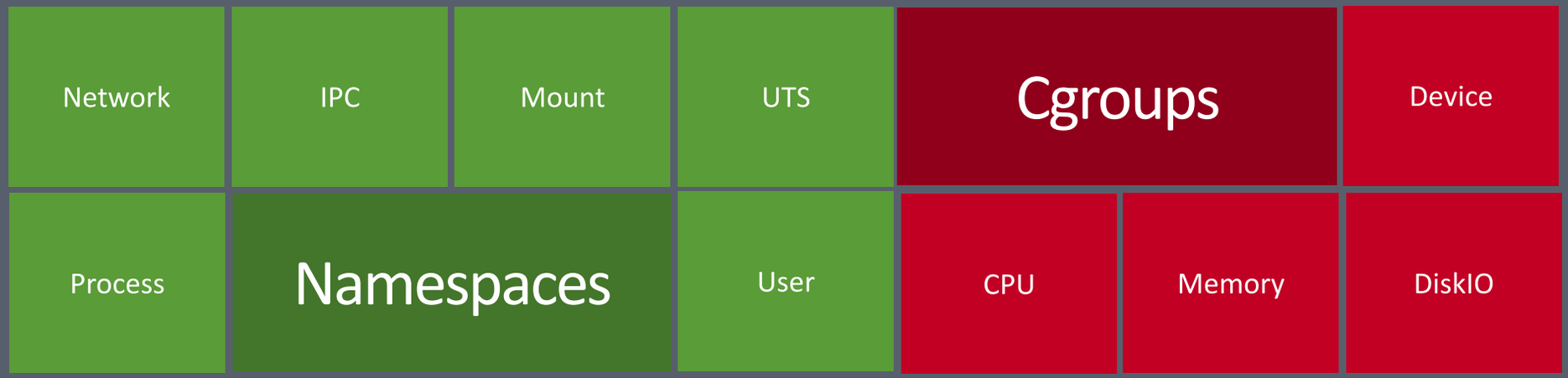
Docker Hub - a cloud-based registry service

Free to use to share your images



Exercise 2 – Building and publishing images

Linux Container Isolation



Running Containers – Process Isolation

```
$ docker run [OPTIONS] IMAGE[:TAG|@DIGEST] [COMMAND] [ARG...]
```

- PID isolation (--pid)
  - By default, all containers have the PID namespace enabled. => allows process ids to be reused including pid 1
  - --pid=”host” => use the host's PID namespace inside the container
  - --pid=“container:<name|id>” => joins another container's PID namespace
- UTS isolation (--uts)
  - By default, all containers have their own UTS namespace (and thus their own hostname and domain)
  - --uts=“host” => use the host's UTS namespace inside the container
- IPC isolation (--ipc)
  - By default, all containers have the IPC namespace enabled. => allows separation of the inter-process communications of named shared memory segments, semaphores and message queues
  - --ipc=”host” => use the host's IPC namespace inside the container
  - --ipc=“container:<name|id>” => joins another container's IPC namespace

Running Containers – Network Isolation

```
$ docker run [OPTIONS] IMAGE[:TAG|@DIGEST] [COMMAND] [ARG...]
```

- --network="bridge" (default)
  - By default, a bridge is setup on the host, commonly named docker0
  - A pair of veth interfaces will be created for the container: One will remain on the host attached to the bridge while the other will be placed inside the container's.
  - An IP address will be allocated for containers on the bridge's network and traffic will be routed though this bridge to the container.
- --network="none"
  - No networking in the container.
- --network="host"
  - Use the host's network stack inside the container.
- --network="container:<name|id>"
  - Use the network stack of another container, specified via its name or id.

Running Containers – Network Isolation

```
$ docker run [OPTIONS] IMAGE[:TAG|@DIGEST] [COMMAND] [ARG...]
```

- --network=“NETWORK”
  - Connects the container to a user created network (using docker network create command)
  - For overlay networks or custom plugins that support multi-host connectivity, containers connected to the same multi-host network but launched from different Engines can also communicate in this way.
- --dns[]
  - The --dns, --dns-search, and --dns-option options can be used to update /etc/resolv.conf inside the container.
- --add-host=""
  - The --add-host flag can be used to add additional lines to /etc/hosts.

Running Containers – Memory constraints

```
$ docker run [OPTIONS] IMAGE[:TAG|@DIGEST] [COMMAND] [ARG...]
```

- User memory constraints
  - -m, --memory="" Memory limit (format: <number>[<unit>]).
  - --memory-swap="" Total memory limit (memory + swap, format: <number>[<unit>]).
  - --memory-reservation="" Memory soft limit (format: <number>[<unit>])
- Swappiness constraint
  - By default, a container's kernel can swap out a percentage of anonymous pages. To set this percentage for a container, specify a --memory-swappiness value between 0 and 100. A value of 0 turns off anonymous page swapping.

Running Containers – CPU constraints

```
$ docker run [OPTIONS] IMAGE[:TAG|@DIGEST] [COMMAND] [ARG...]
```

- CPU share constraint
  - By default, all containers get the same proportion of CPU cycles.
  - A container process can take all CPU cycles of the host system, but when processes are competing, the CPU cycles are assigned proportionately.
  - To modify the proportion from the default of 1024, use the -c or --cpu-shares flag to set the weighting.
- CPU period and quota constraint
  - The default CPU CFS (Completely Fair Scheduler) period is 100ms.
  - We can use --cpu-period to set the period. Default value is: 100000
  - The --cpu-quota flag limits the container's CPU time. By default there is no quota (value of 0)
  - --cpu-quota=50000 => 50000 / 100000 = 50% CPU
  - --cpu-quota=100000 --cpu-period=50000 => 100000 / 50000 = 200% CPU
  - In addition to use these two options for setting CPU period constraints, it is possible to specify --cpus with a float number to achieve the same purpose. => --cpus=0.5
  - Note that the CFS can cause response latencies when the CPU time has been used before the specified period has been elapsed!

Running Containers – Disk IO constraints

```
$ docker run [OPTIONS] IMAGE[:TAG|@DIGEST] [COMMAND] [ARG...]
```

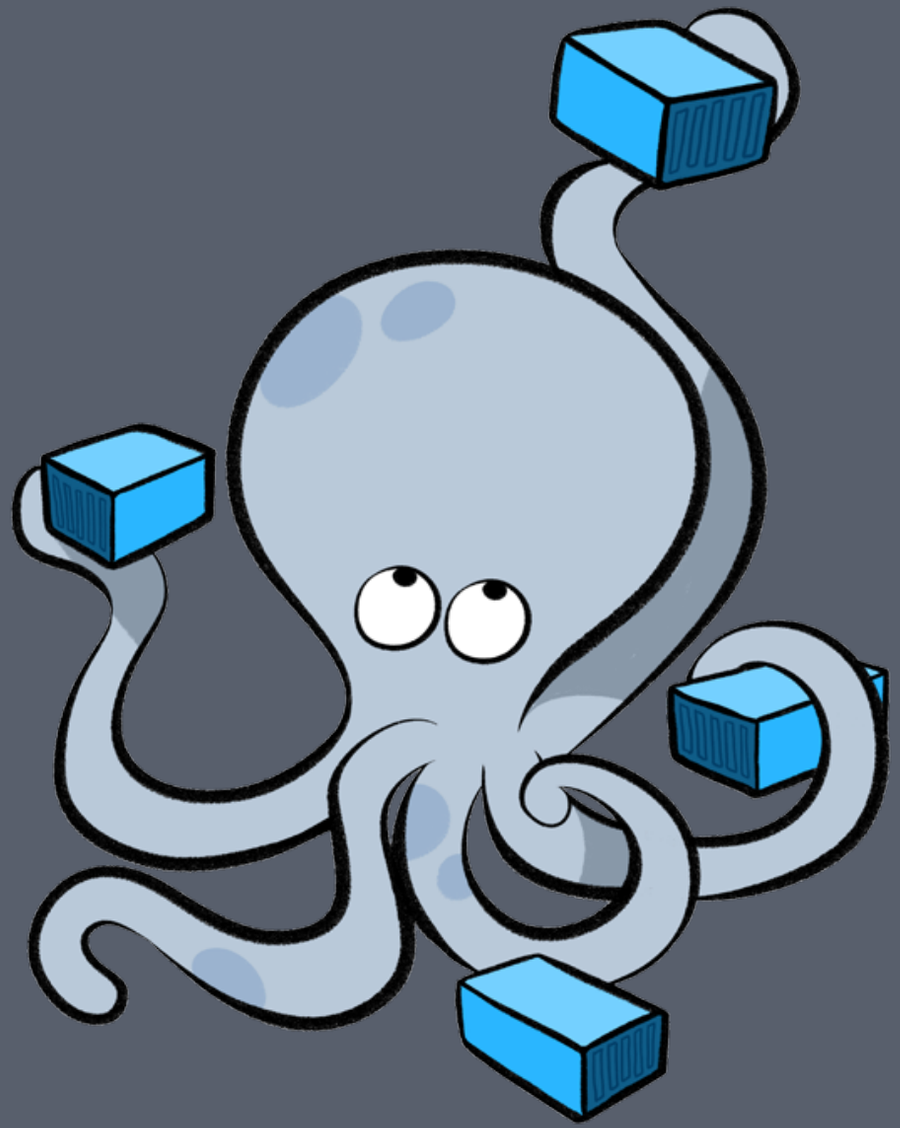
- Disk IO bandwidth (Blkio) constraint
  - By default, all containers get the same proportion of IO bandwidth (blkio). This proportion is 500.
  - change the container's blkio weight relative to all other running containers using the --blkio-weight flag.
  - The --device-read-bps flag limits the read rate (bytes per second) from a device.
  - The --device-write-bps flag limits the write rate (bytes per second)to a device.
  - The --device-read-iops flag limits read rate (IO per second) from a device.
  - The --device-write-iops flag limits write rate (IO per second) to a device.

Running Containers – Privilege/Capabilities

```
$ docker run [OPTIONS] IMAGE[:TAG|@DIGEST] [COMMAND] [ARG...]
```

- By default, Docker containers are “unprivileged” and cannot, for example, run a Docker daemon inside a Docker container.
- --privileged; Docker will enable to access to all devices on the host to allow the container nearly all the same access to the host as processes running outside containers on the host.
- If you want to limit access to a specific device or devices you can use the --device flag.
- have fine grain control over the capabilities using --cap-add and --cap-drop.

Docker Compose



- Tool for defining and running multi-container Docker applications.
- A Compose file configures your application’s services.
- With a single command, all the services are created and started.

Docker Compose – Use Cases

- Development environments
- Automated testing environments
- Single host deployments

Docker Compose File

```
version: '2'
services:
	db:
		image: mysql:5.7
		volumes:
			- db_data:/var/lib/mysql
		restart: always
		environment:
			MYSQL_ROOT_PASSWORD: wordpress
			MYSQL_DATABASE: wordpress
			MYSQL_USER: wordpress
			MYSQL_PASSWORD: wordpress
	wordpress:
		depends_on:
			- db
		image: wordpress:latest
		ports:
```

Docker Compose File

```
            environment:
                  MYSQL_ROOT_PASSWORD: wordpress
                  MYSQL_DATABASE: wordpress
                  MYSQL_USER: wordpress
                  MYSQL_PASSWORD: wordpress
      wordpress:
            depends_on:
                  - db
            image: wordpress:latest
            ports:
                  - "8000:80"
            restart: always
            environment:
                  WORDPRESS_DB_HOST: db:3306
                  WORDPRESS_DB_PASSWORD: wordpress
volumes:
      db_data:
```

Exercise 3 – Run Services with Docker Compose