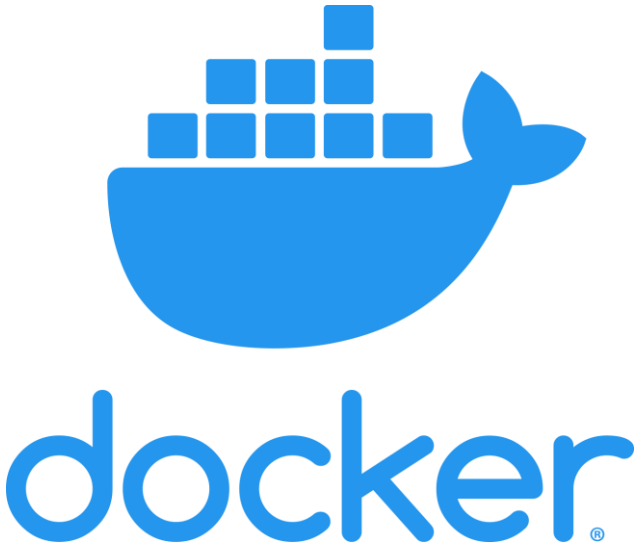
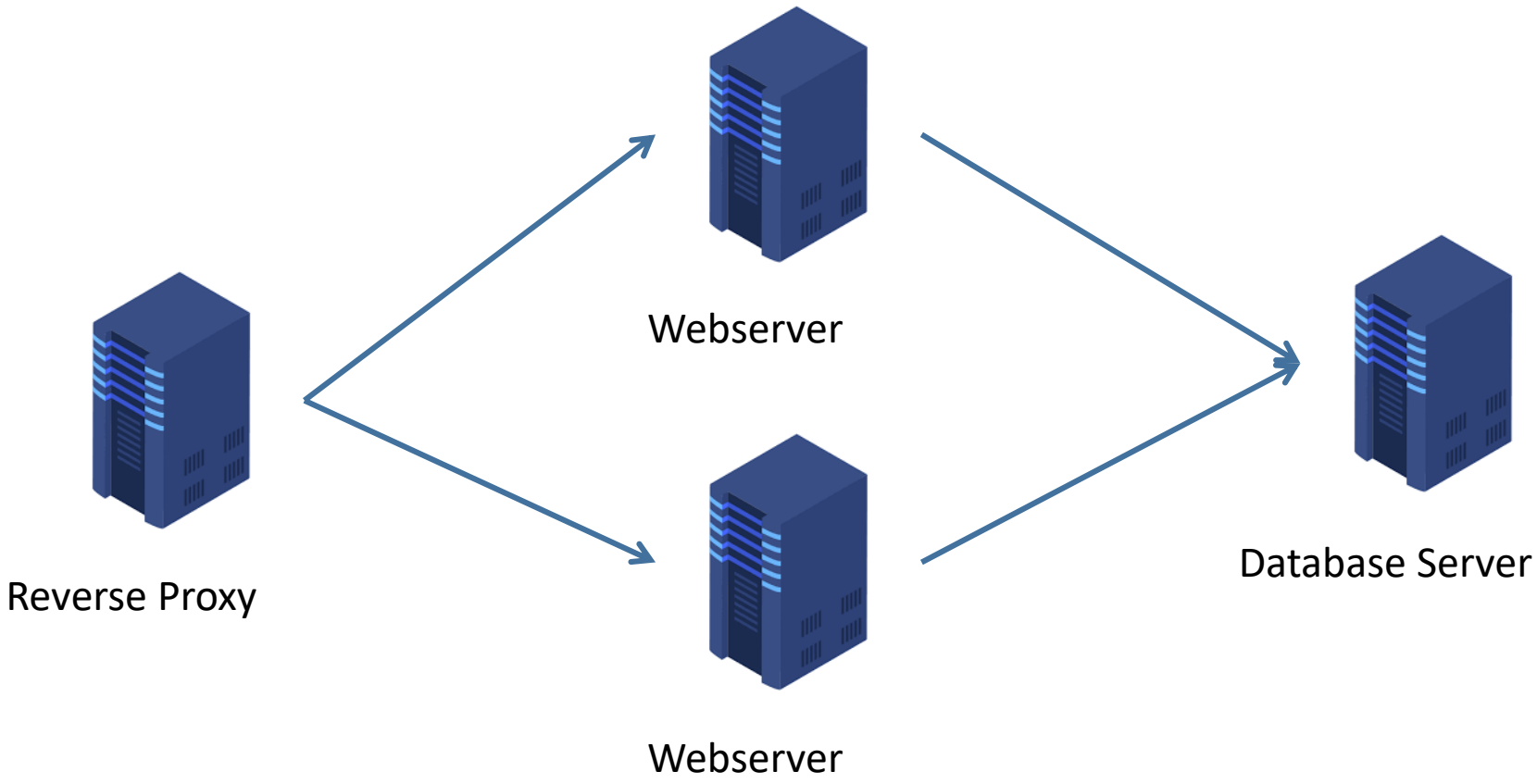
A very informal introduction to Docker

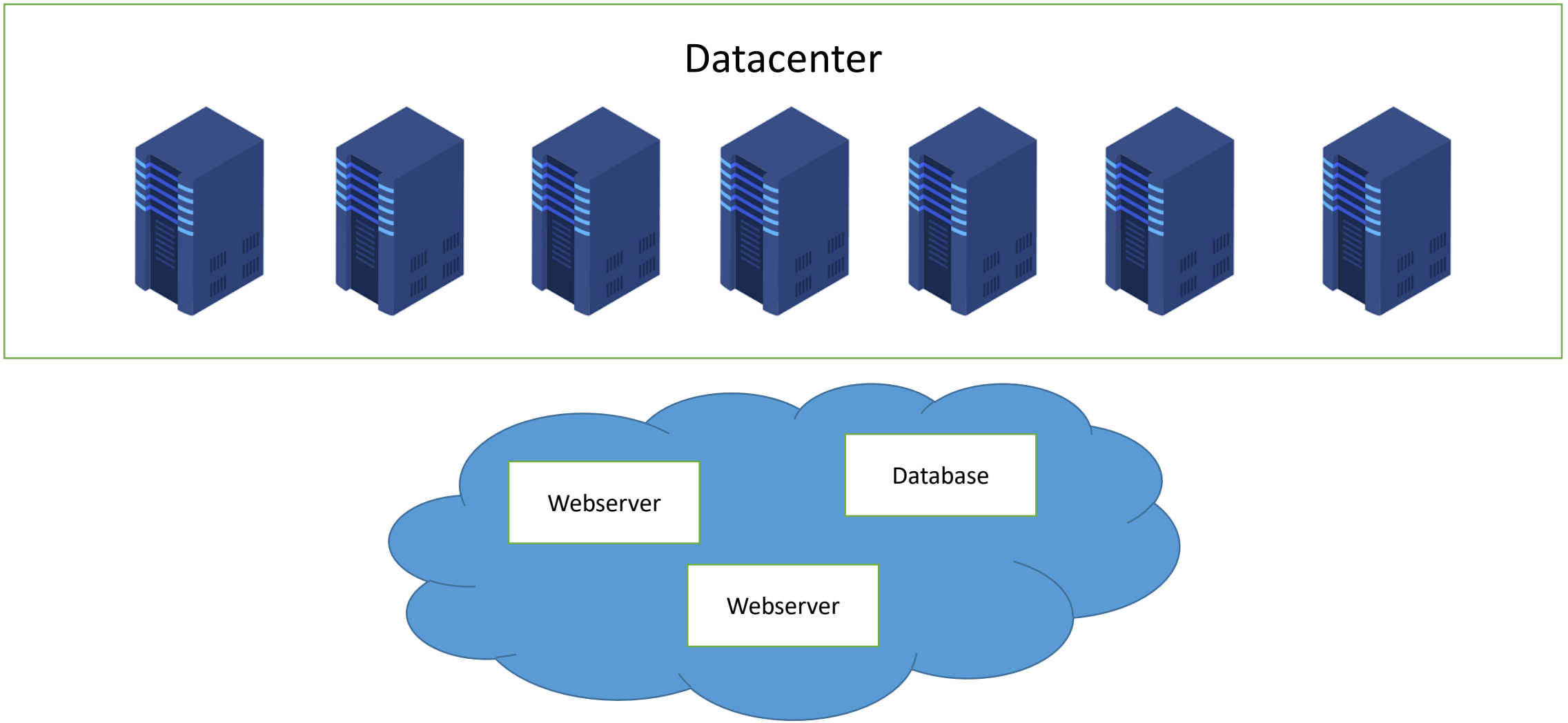
Federico Galatolo



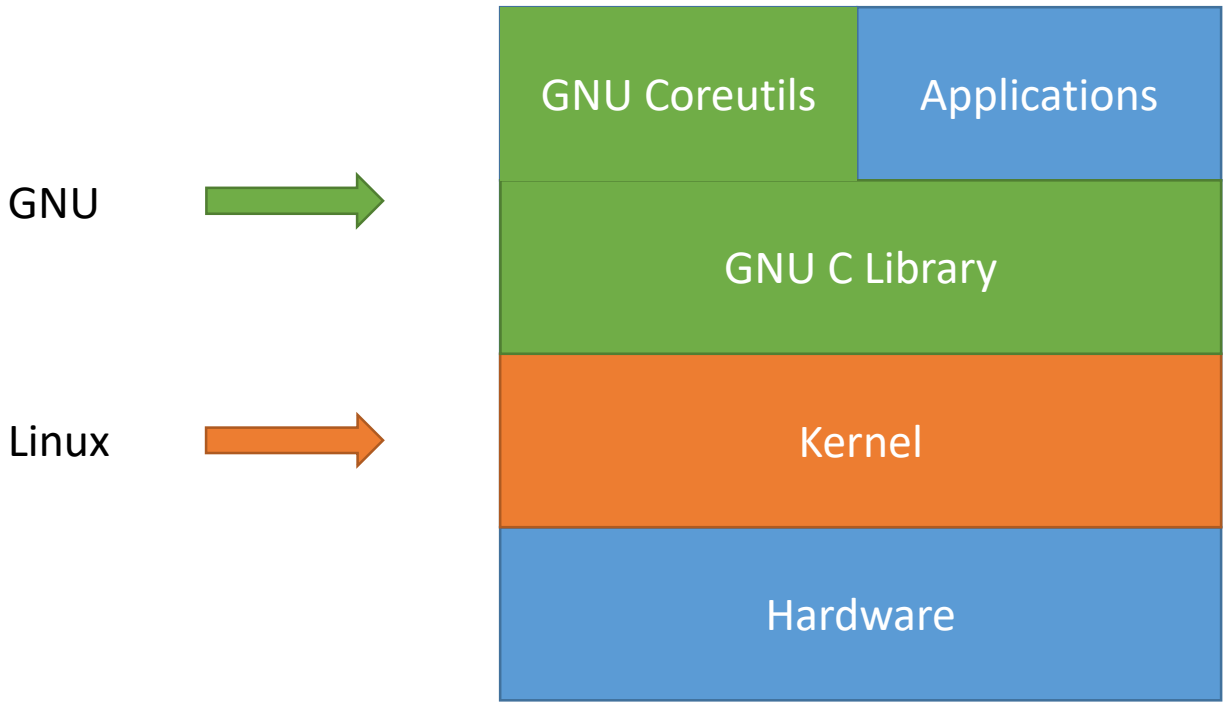
Once upon a time...



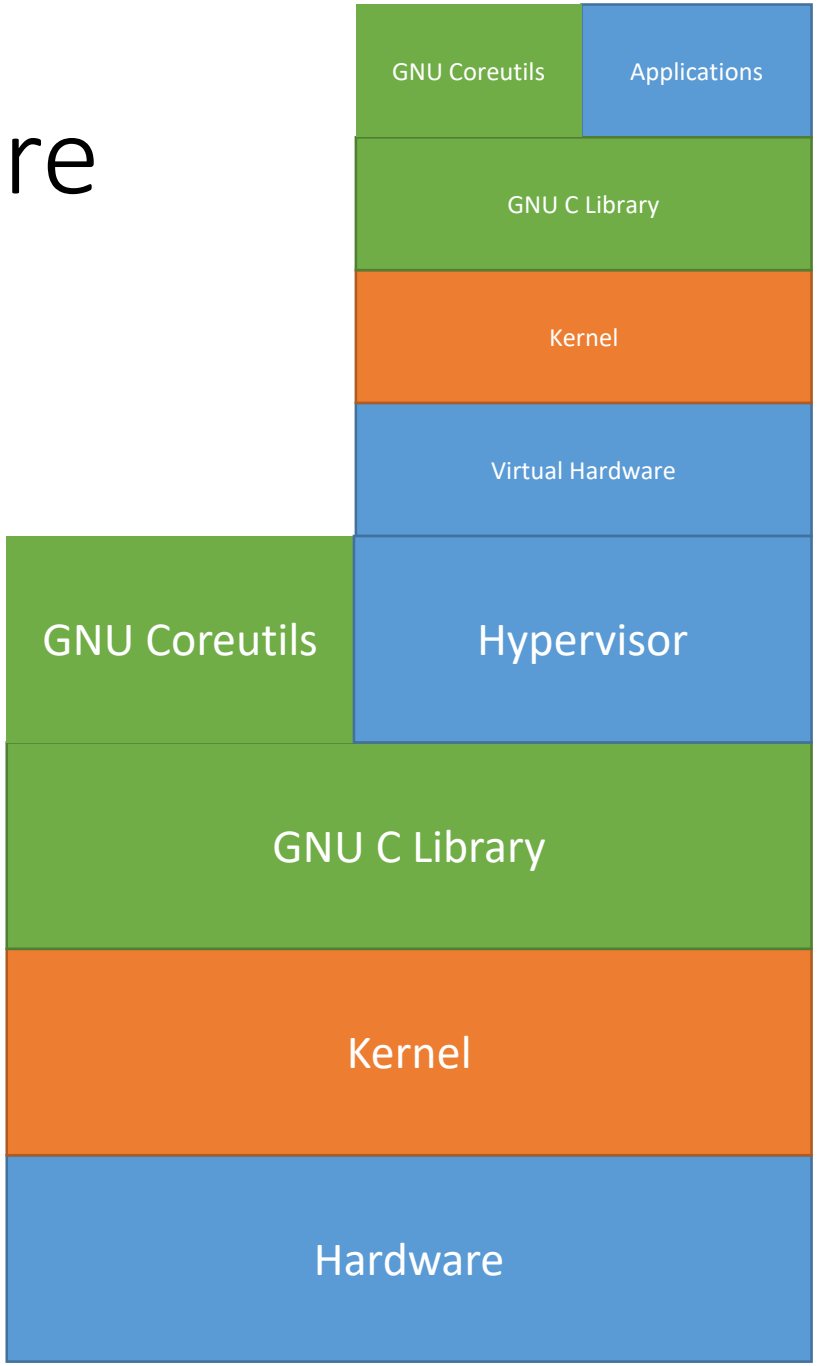
And then was the cloud...



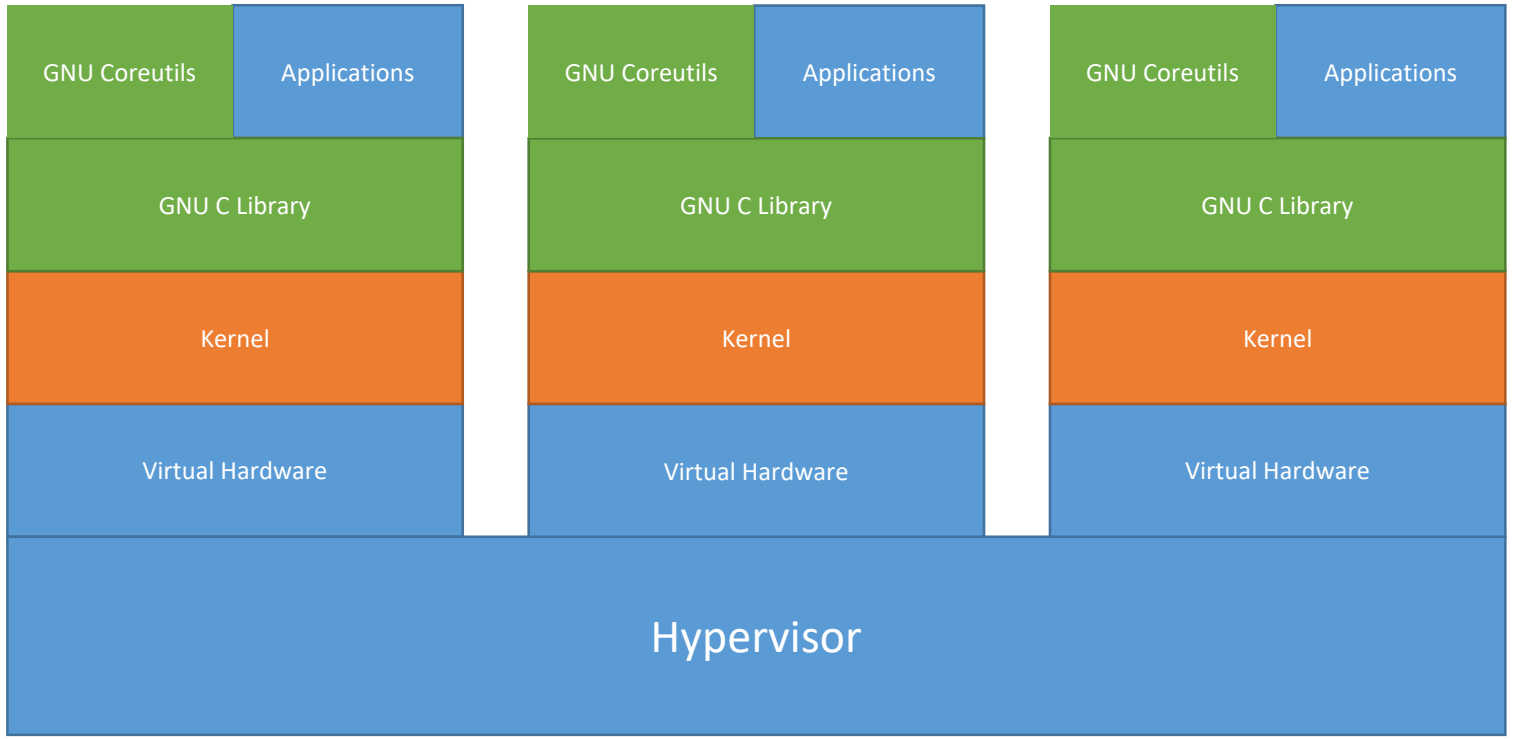
OS Architecture



VM Architecture

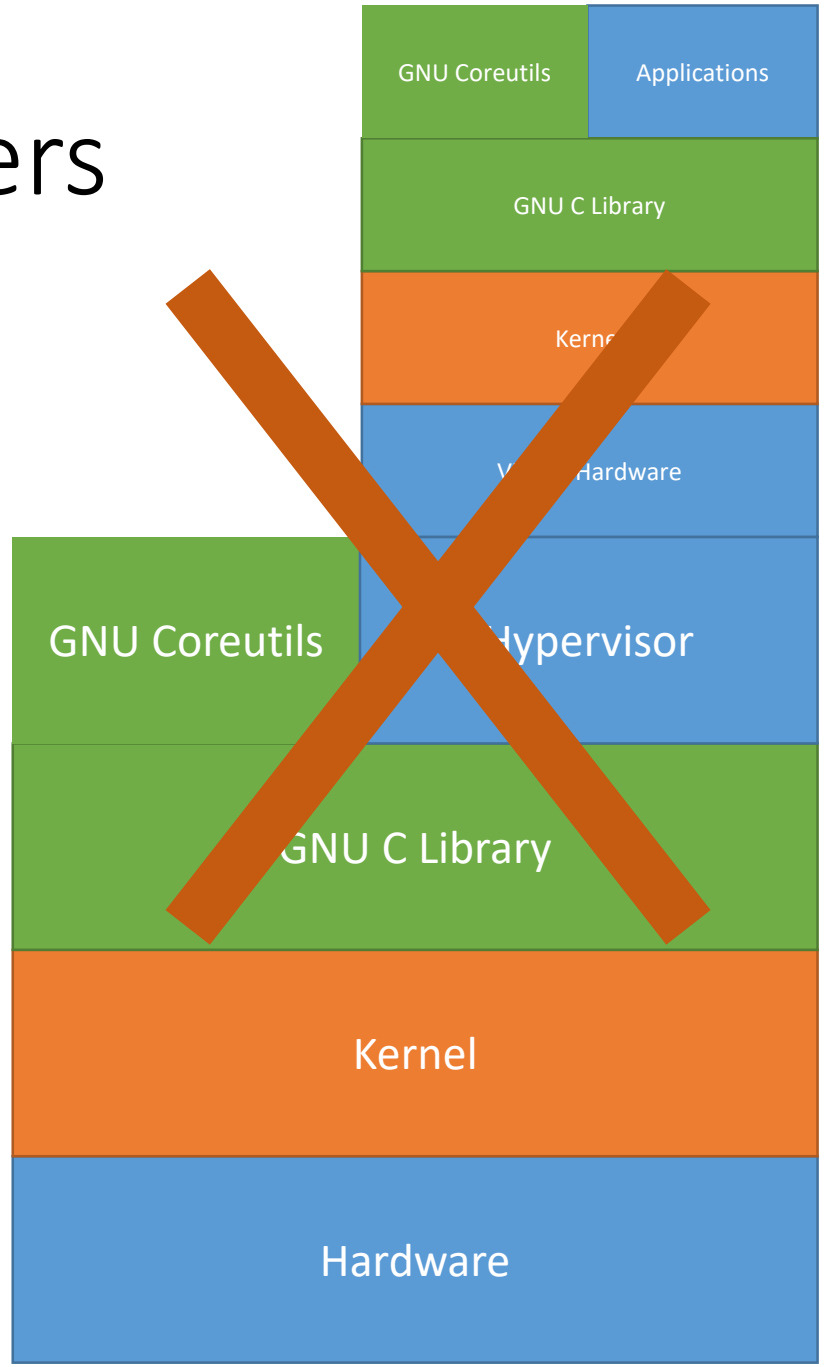


VMs Architecture

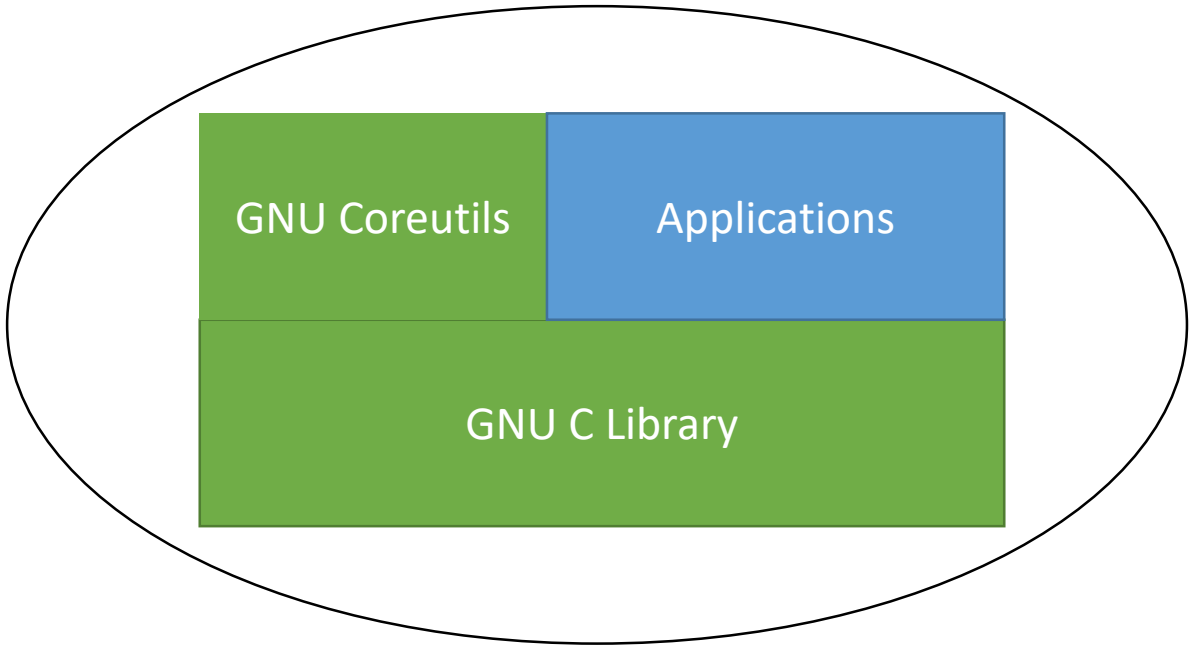


Too much overhead

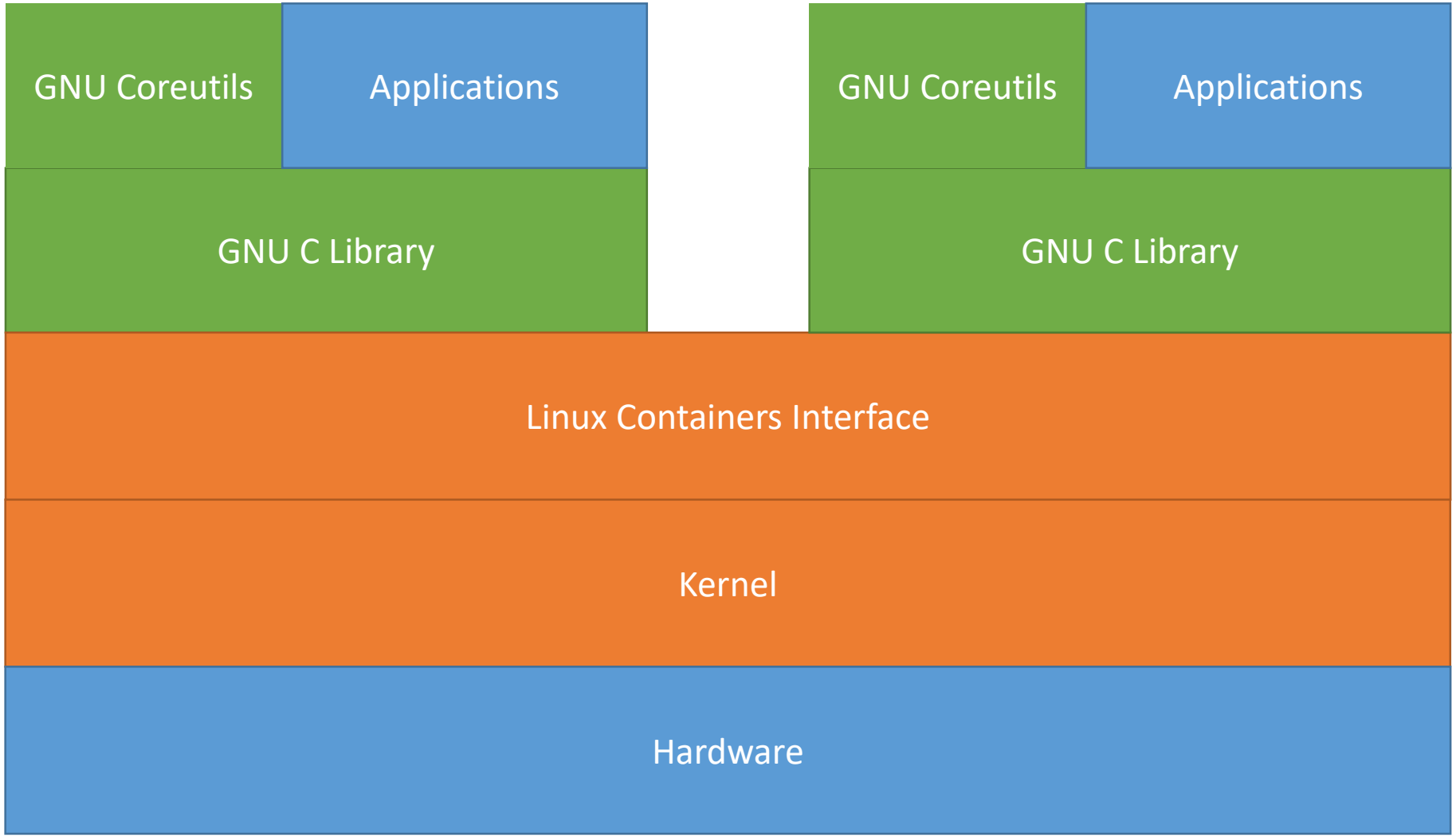
Linux Containers



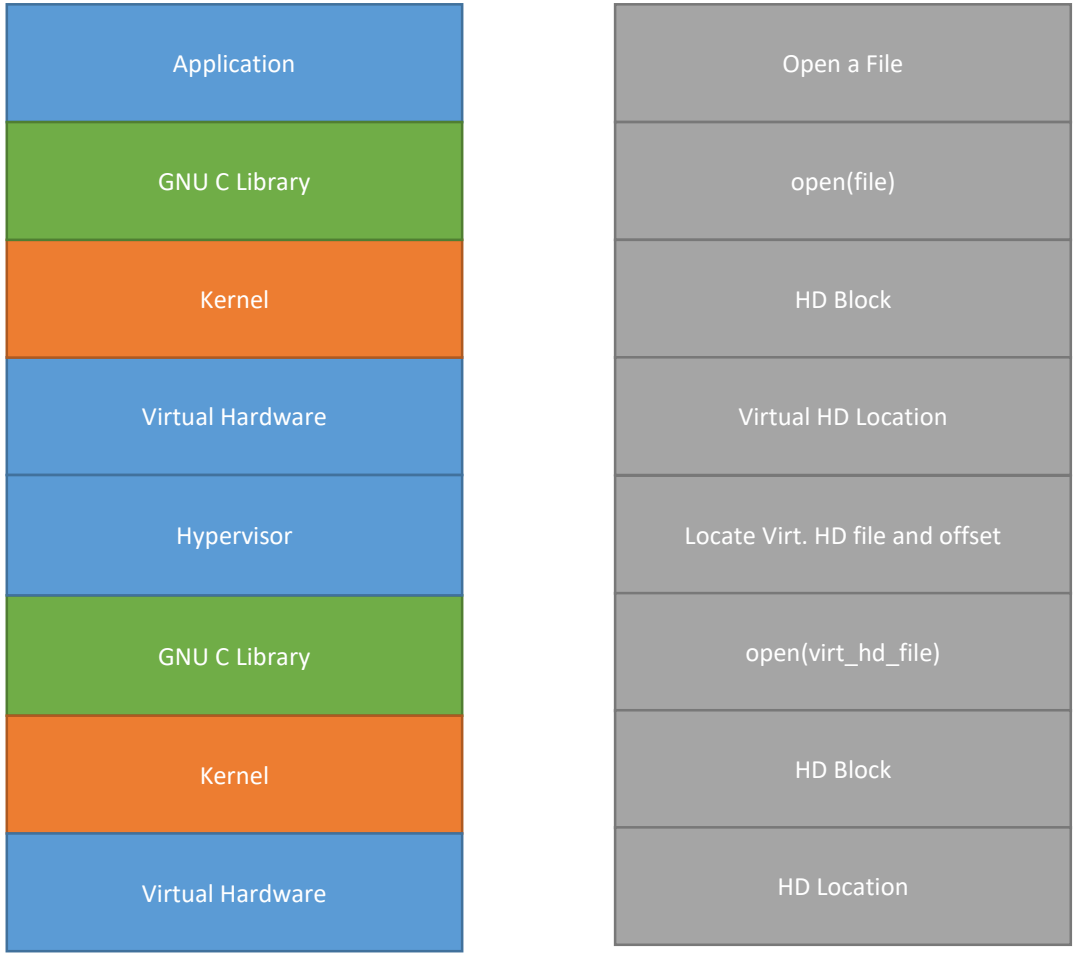
Linux Container



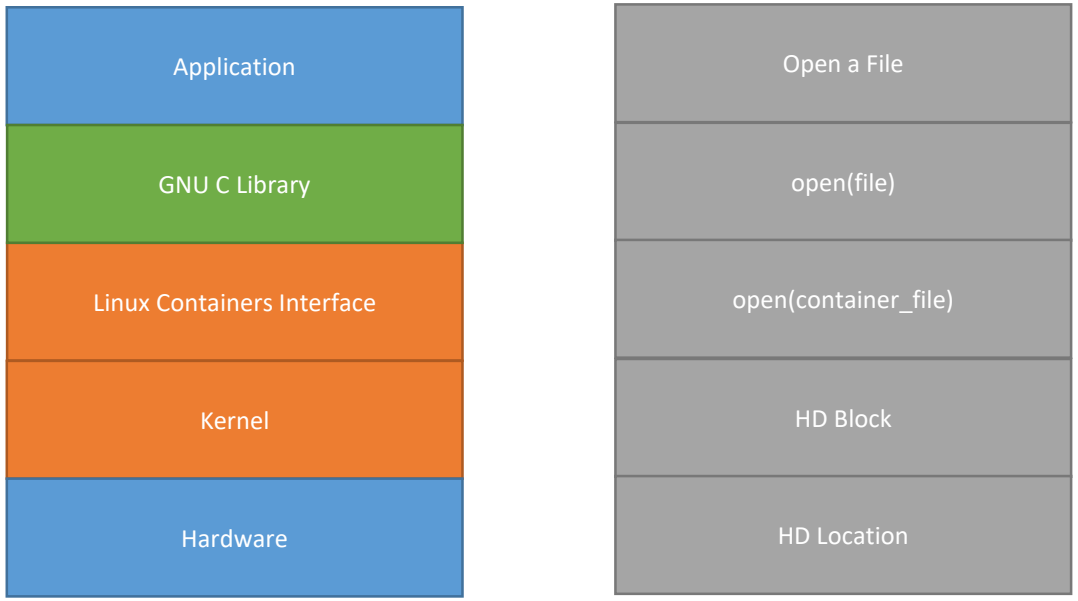
Linux Containers Architecture



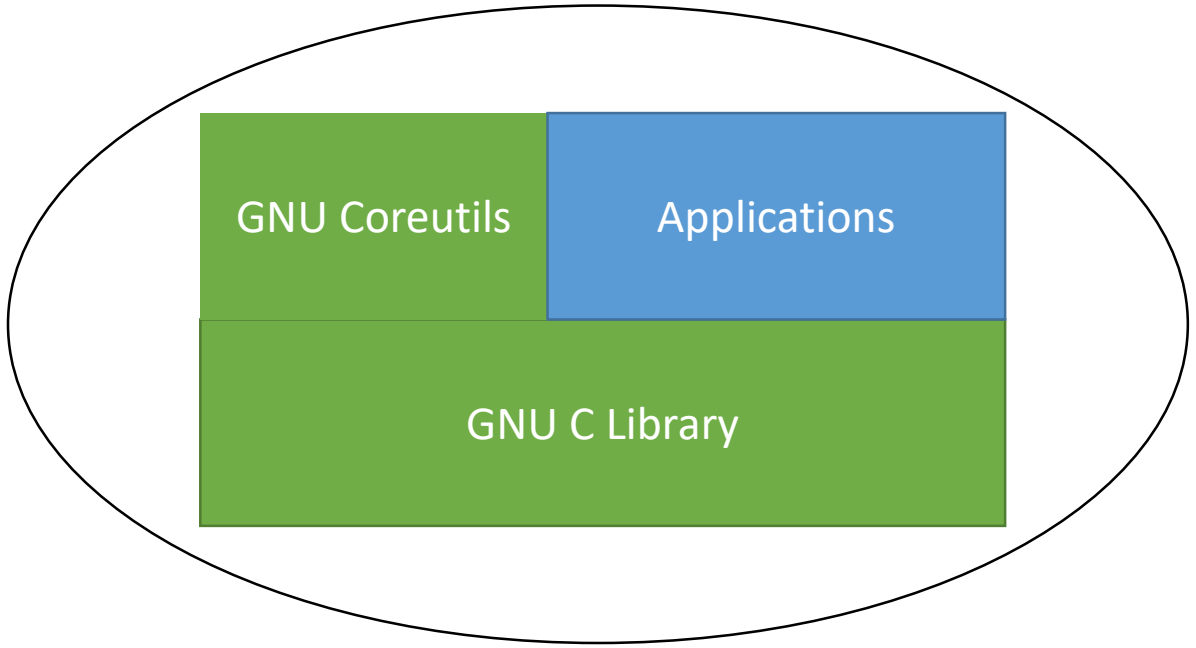
Containers VS VMs: Open a File in VM



Containers VS VMs: Open a File in Container



Linux Containers



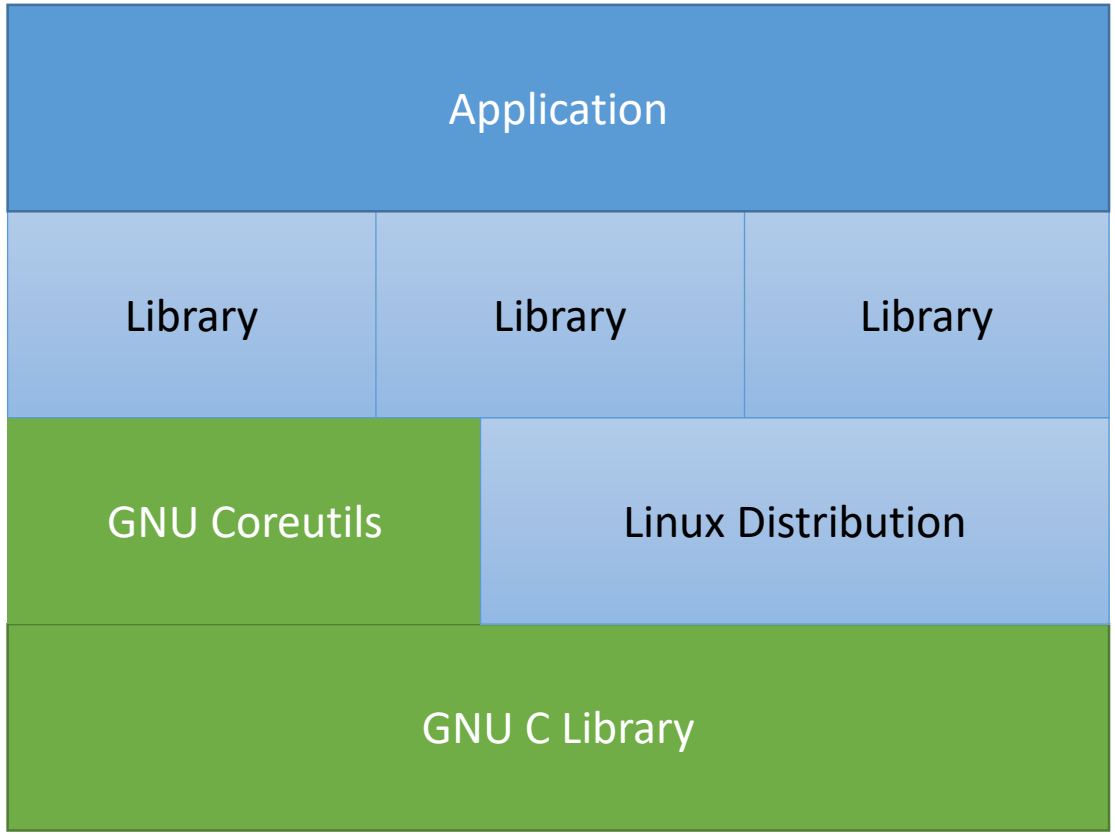
Basically zero overhead and host-like performances

VM-like isolation

Lightweight and portable

But can we do better?

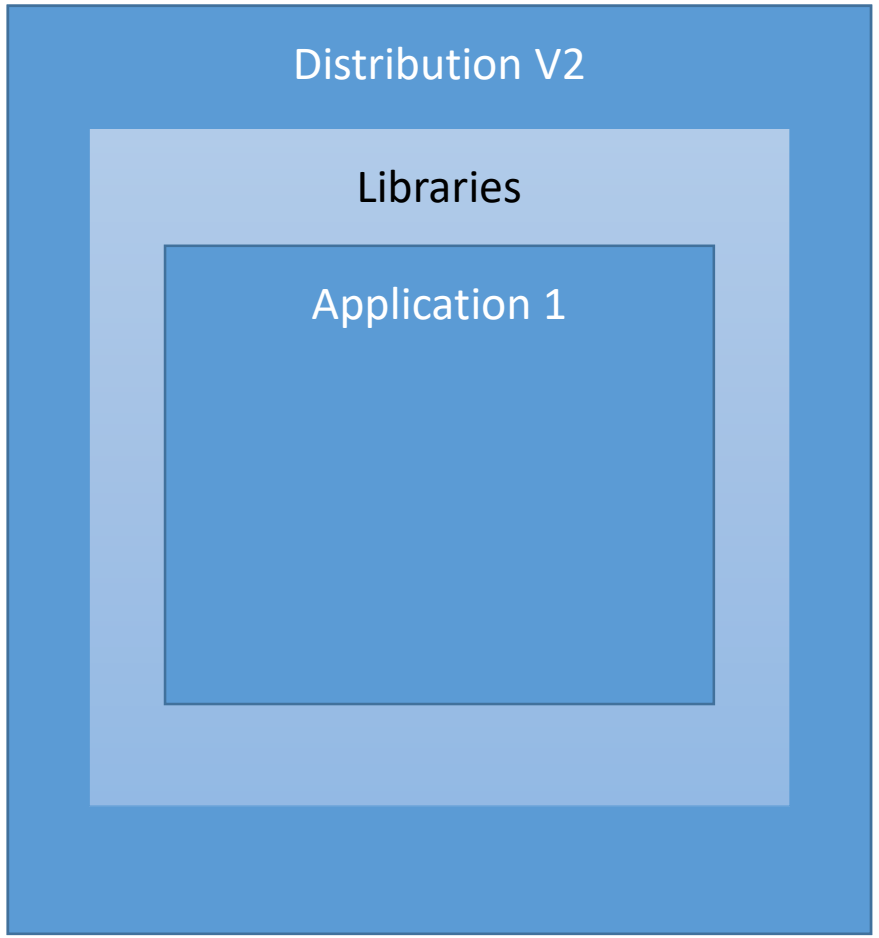
Linux Containers

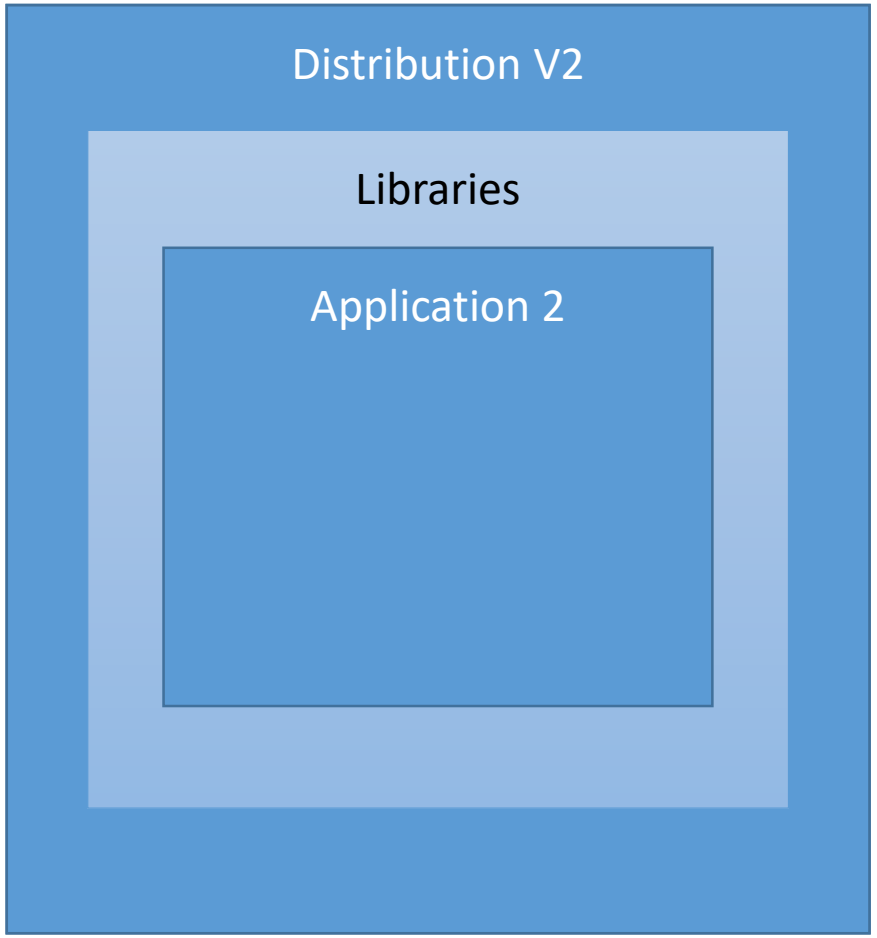


What if some applications use the same linux distribution or libraries?

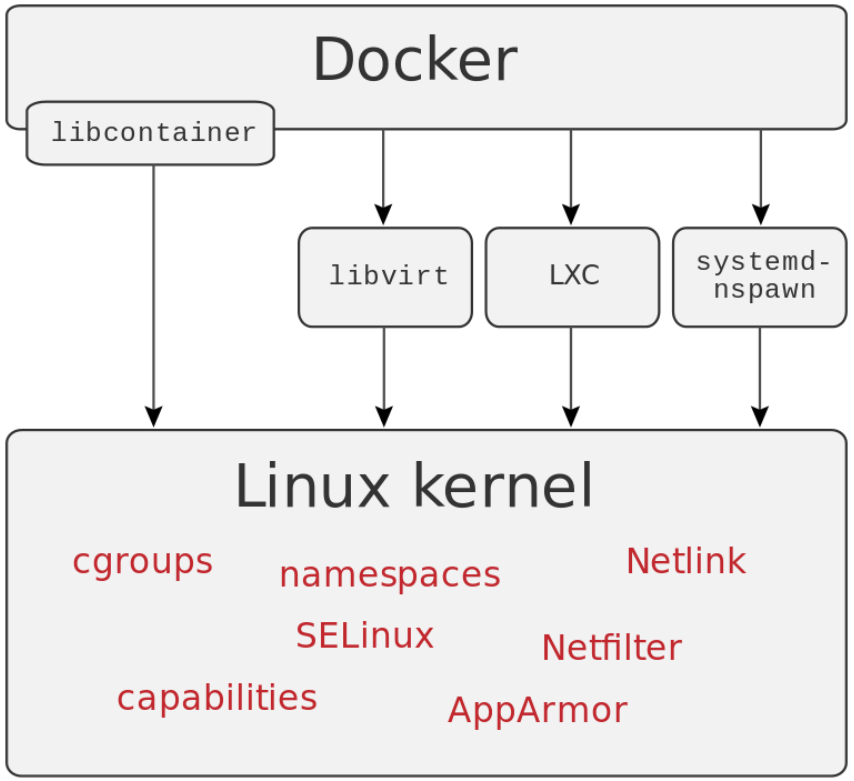
How to handle updates?

Wouldn't be nice if ...?





Docker



Docker is an interface to easily manage containers

It can use different containers backends (even full virtualization)

Handle containers in a onion-like structure

Time to get your hads dirty

Download and install docker:

https://docs.docker.com/install/

Dockerfile(1)

The Dockerfile is a file with which you can specify a docker image.

It is a plaintext file representing a sequence of steps needed to create your image.

Each command creates a “layer”

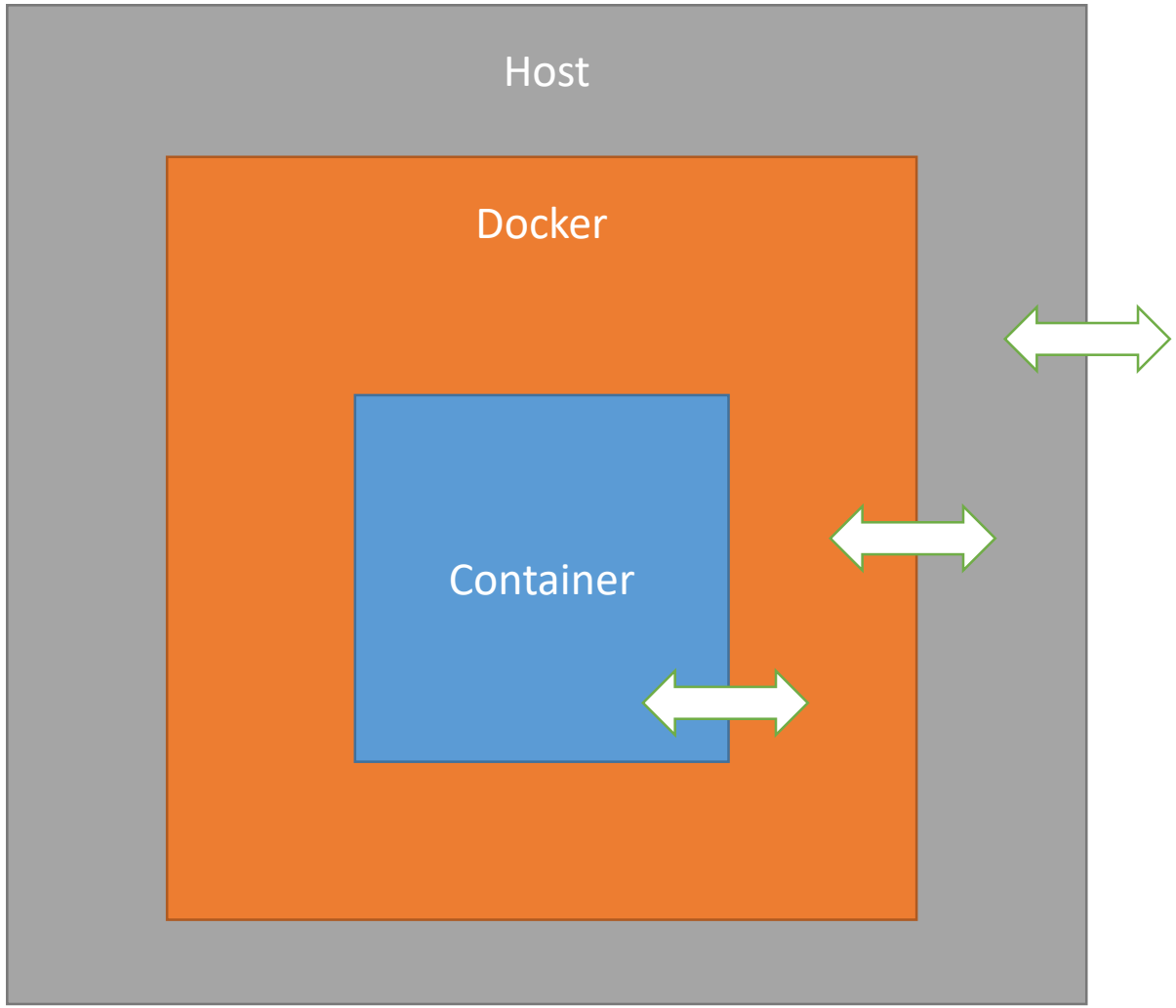
- FROM <image>
  - Use <image> as base image
- RUN <cmd>
  - Run the command <cmd>
- CMD <cmd>
  - Specify the command to run your application

Docker: build and run a container

- Build a container
  - docker build -t <container name> <Dockerfile path>
- Run a container
  - docker run <container name>

Hello world Example

Docker EXPOSE



- EXPOSE <port>
- docker -p <host_ip>:<host_port>:<container_port>/<protocol>
- iptables?

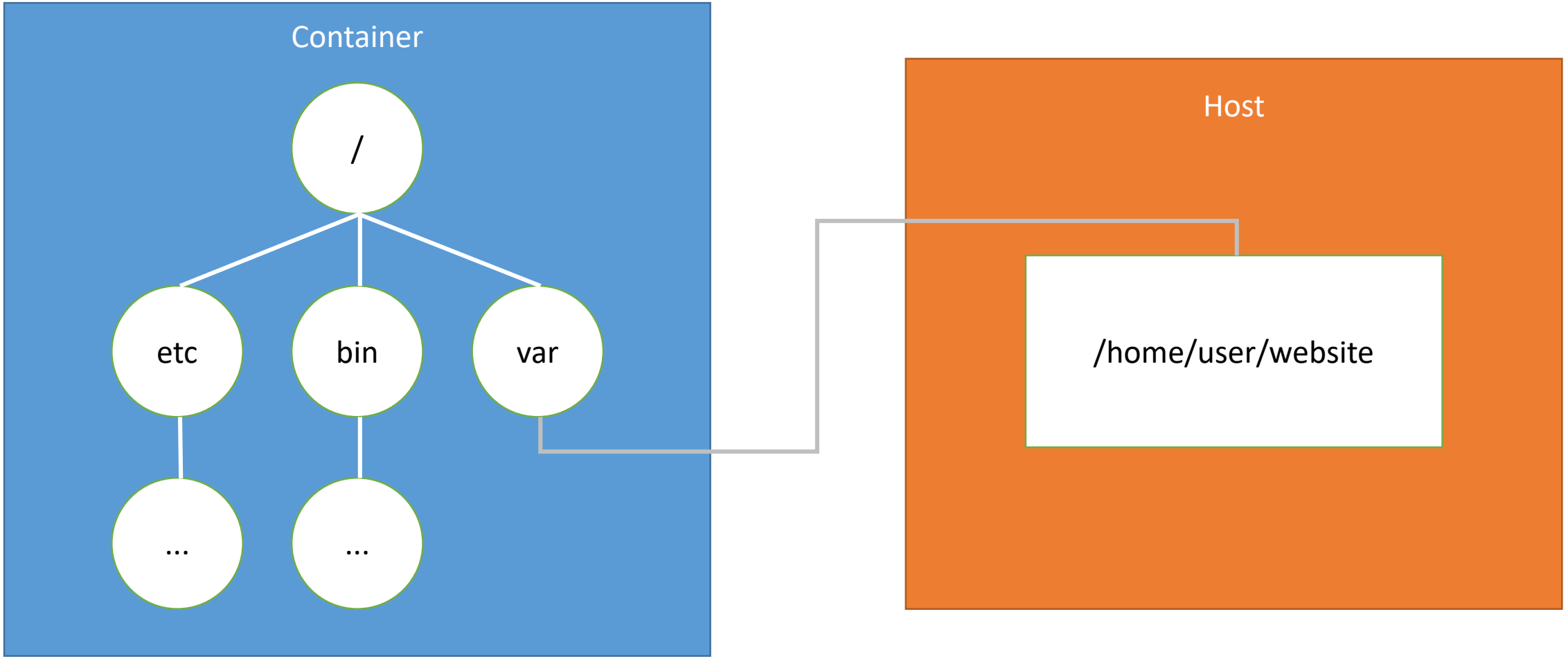
Dockerfile(2)

- COPY <host src> <container dst>
  - Copy a file/folder from host <host src> to container <container dst>
- EXPOSE <port>/<protocol>
  - Expose the container port <port> to docker

Hello world v2 Example
Echo Server Example

Docker Volumes

Volumes are a way to mount host folders in container ones



Docker Volumes

- docker -v <host_path>:<container_path>
  - Mount <host_path> host folder into <container_path> container folder

- Docker volumes are way more complex than this. But for now this is enough

Hello world v3 Example
Python webserver Example

Docker CLI isnt enough

Docker CLI interface is amazing!

Manage lifecycle of container with a bunch of batch scripts?

Maybe there is a better way

Docker Compose

With Docker Compose you can define and control an **entire architecture** with **one yaml file**

- Services
- Volumes
- Networks
- Connections
- Dependencies
- …

YAML

YAML is a human readable serialization language, easier than XML and JSON.

- field: value
- - for list elements
- indentation spaces for objects

That's all

YAML: Example

```xml
<root>
  <todo>
    <name>Docker lecture</name>
    <done>false</done>
  </todo>
  <todo>
    <name>Docker slides</name>
    <done>true</done>
  </todo>
</root>
```

XML

```json
{
    "todo":[
        {
            "name": "Docker lecture",
            "done": false
        },
        {
            "name": "Docker slides",
            "done": true
        }
    ]
}
```

JSON

```yaml
todo:
- name: Docker lecture
  done: false
- name: Docker slides
  done: true
```

YAML

YAML: Example 2

```json
{
  "phd-students": [
    {
      "federico": {
        "name": "Federico Galatolo",
        "job": "PhD Student",
        "skills": [
          "linux",
          "python"
        ]
      }
    },
    {
      "manilo": {
        "name": "Manilo Monaco",
        "job": "Developer",
        "skills": [
          "matlab",
          "python"
        ]
      }
    }
  ]
}
```

```yaml
phd-students:
-   federico:
      name: Federico Galatolo
      job: PhD Student
      skills:
        - linux
        - python
-   manilo:
      name: Manilo Monaco
      job: Developer
      skills:
        - matlab
        - python
```

Docker Compose

- version: <docker-compose-version>
- services:
  - service-name:
    - image: <docker-image>
    - build: <path-to-dockerfile>
    - ports:
      - <host-port>:<container-port>
      - ...
    - volumes:
      - <host-path>:<container-path>
      - ...
    - environment:
      - ENV_VAR=value
      - ...
    - deploy:
      - replicas: <number_of_replicas>
      - ....

Works only in "swarm" mode

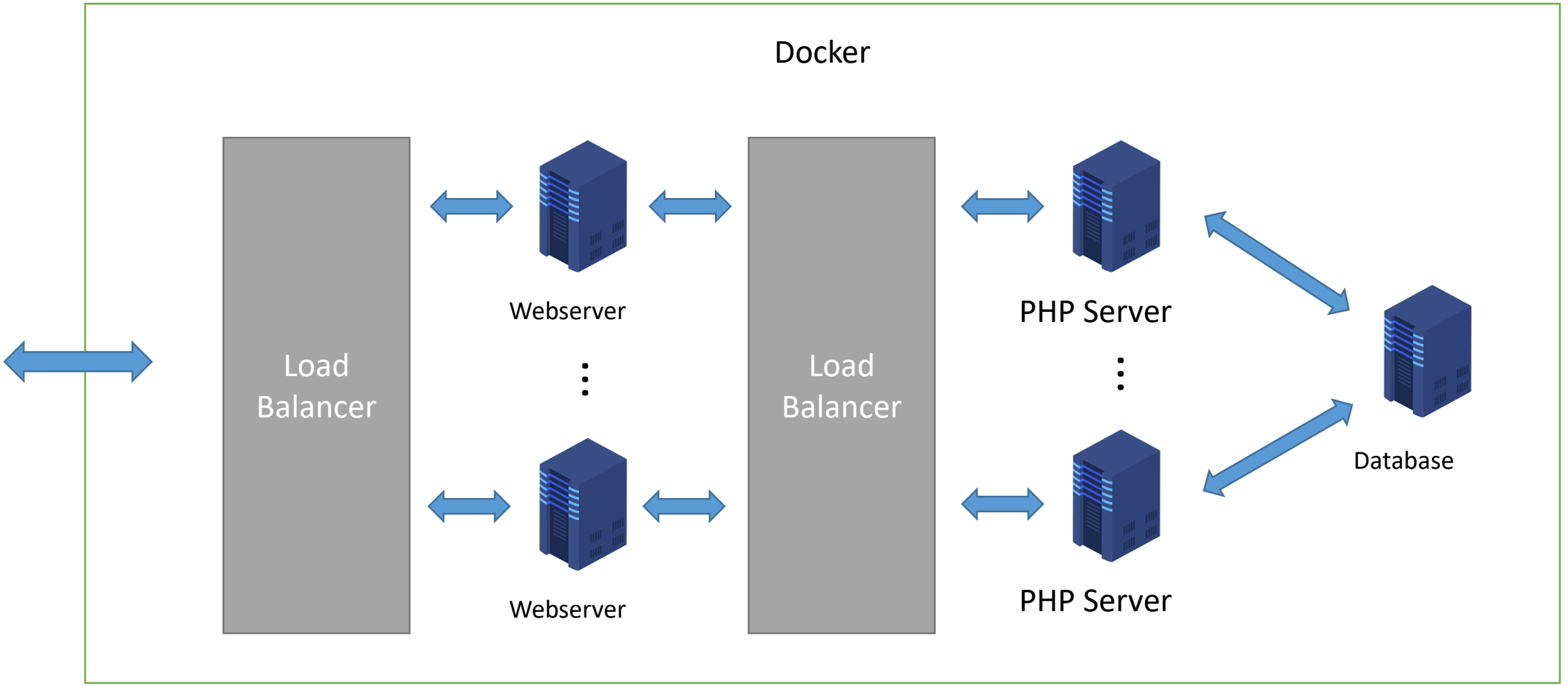
Docker Compose CLI

- Uses the file “docker-compose.yml” by default, you can specify a different yaml file with -f
- docker-compose up
  - Starts all the containers
- docker-compose stop
  - Stops all the containers
- docker-compose build
  - Builds all the containers that use the “build” keyword
- You can start the containers in detached mode with -d
- You can add the flag --compatibility to use “swarm” features

Docker internal DNS

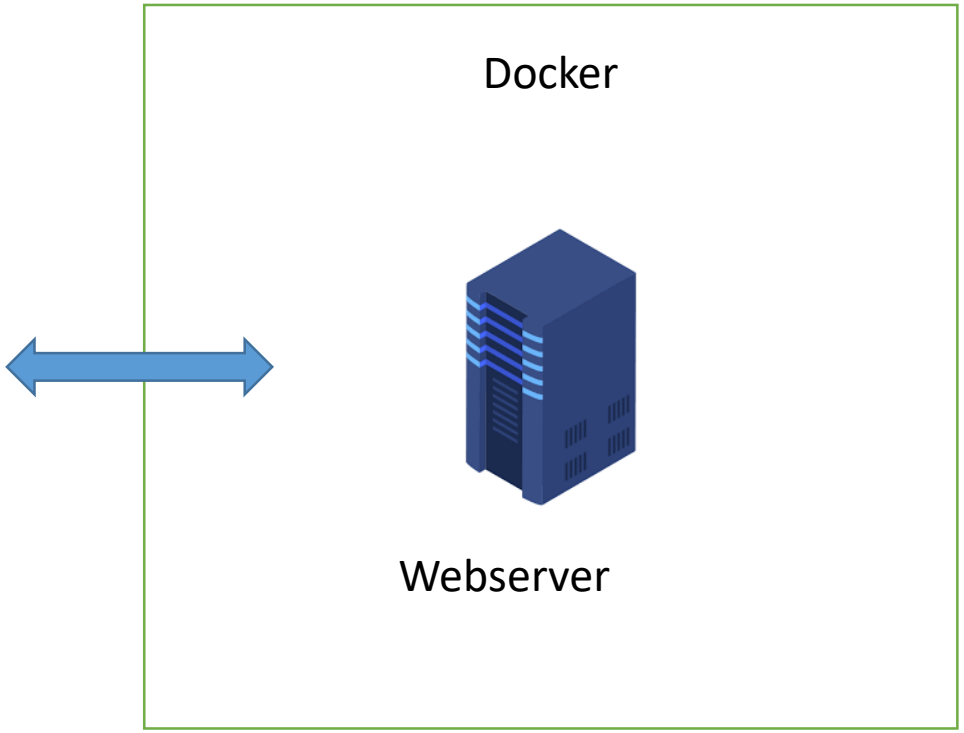
- Docker SDN (Software Defined Network) has its own DNS resolver
- You can use the container name to resolve its ip
- If you scale a container (with docker-compose scale o replicas) the internal DNS will round-robin all the containers
  - Do NOT use this as redundancy but always use a proper reverse proxy

Let's build a LEMP stack



linux-nginx-php-mysql

Let's build a LEMP stack: Webserver



Let's build a LEMP stack: Webserver

https://hub.docker.com/_/nginx

Default configuration file location: /etc/nginx/conf.d/default.conf

OT: nginx

nginx (pronounced “Engine-X”) is an high performance

- Web Server
- Reverse Proxy
- Load Balancer
- HTTP Cache
- ...

Very easy to configure and manage. Used by 30% of the websites worldwide

OT: nginx basic webserver

```
server {
    listen <port>;
    location <regex> {
        root <path>;
    }
    location <regex> {
        root <path>;
    }
}
```

```
server {
    listen 80;
    location / {
        root /var/www;
        location /images/ {
            root /data/images;
    }
}
```

OT: nginx basic reverse proxy

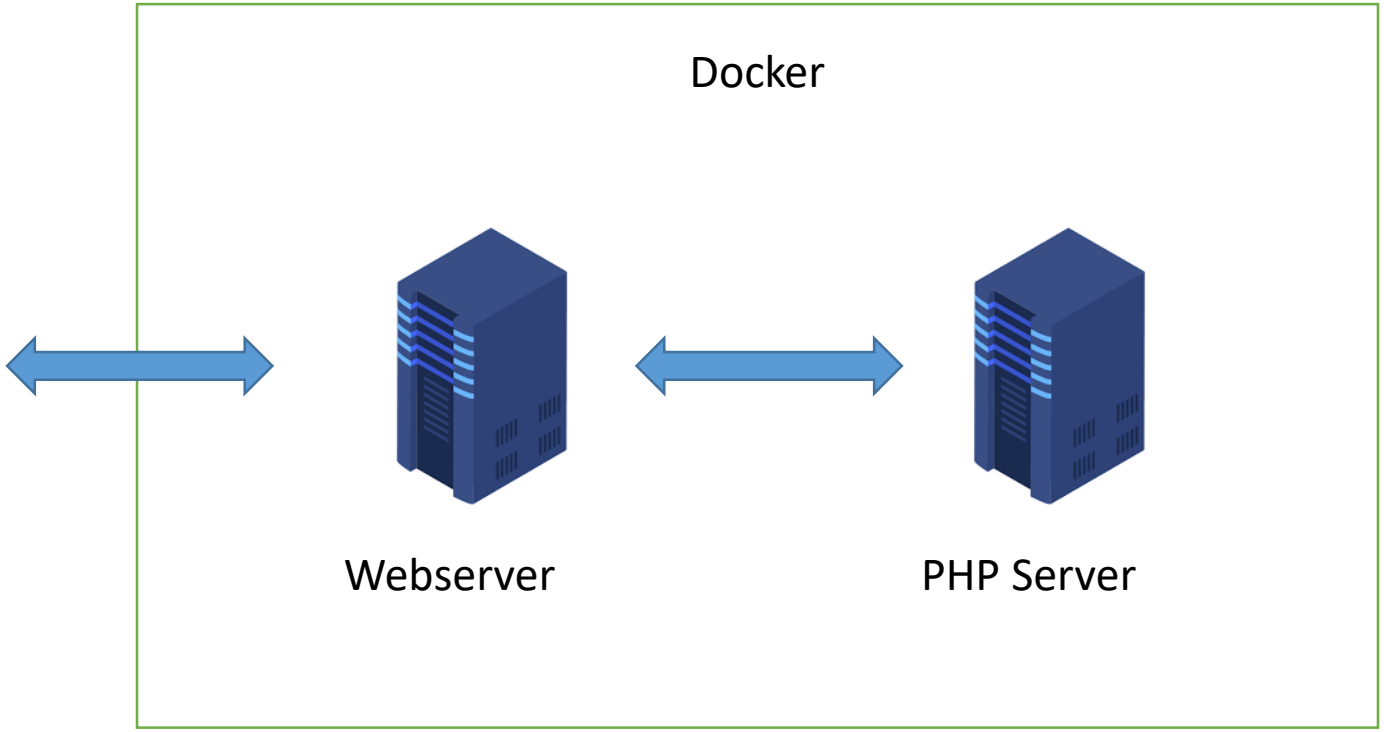
```
upstream <name> {
   server host1:port;
   server host2:port;
   ....
}

server {
   listen <port>
   location <regex> {
      proxy_pass <proto>://<name>
   }
}
```

```
upstream revhttp {
   server http-1:8080;
   server http-2:8080;
}

server {
   listen 80
   location / {
      proxy_pass http://revhttp;
   }
}
```

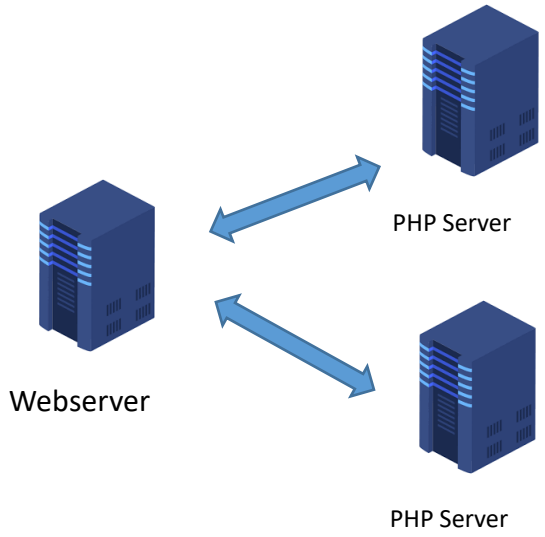
Let's build a LEMP stack: PHP



php-fpm

OT: Why php-fpm?

Apache php_mod	php-fpm reverse proxy
PHP execute in the same machine of the webserver	PHP execute in a different machine than the webserver
PHP interpreter is always loaded	PHP interpreter is loaded only for php content
Scale websever = scale php	Scale webserver only or php only (or both)
Apache is slow	php-fpm is webserver agnostic



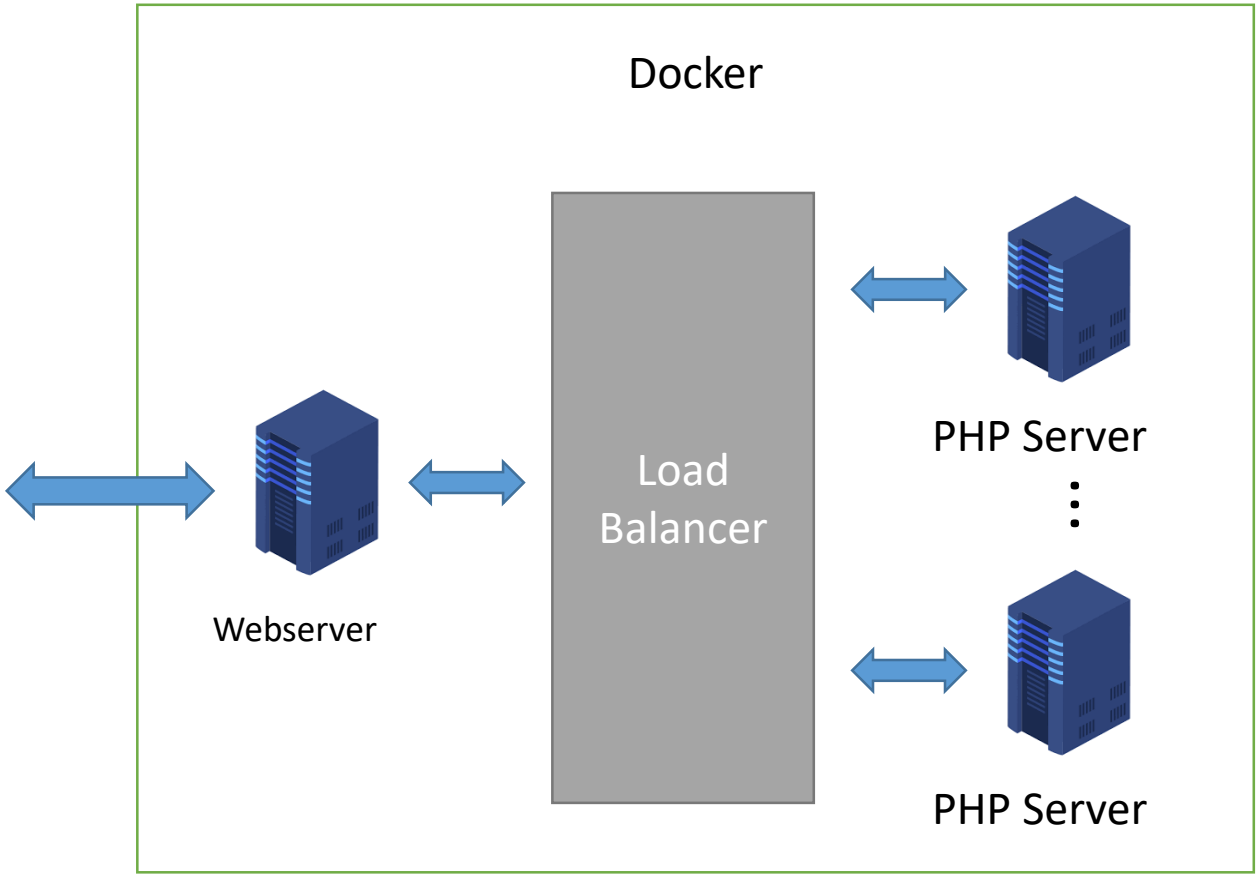
How php-fpm with nginx?

```
server {
    listen 80;
    location / {
        root /website;
        location ~ \.php$ {
            try_files $uri =404;
            fastcgi_split_path_info ^(.+\.php)(/.+)$;
            fastcgi_pass php:9000;
            fastcgi_index index.php;
            include fastcgi_params;
            fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
            fastcgi_param PATH_INFO $fastcgi_path_info;
        }
    }
}
```

Assuming php-fpm running on port 9000 of host “php”

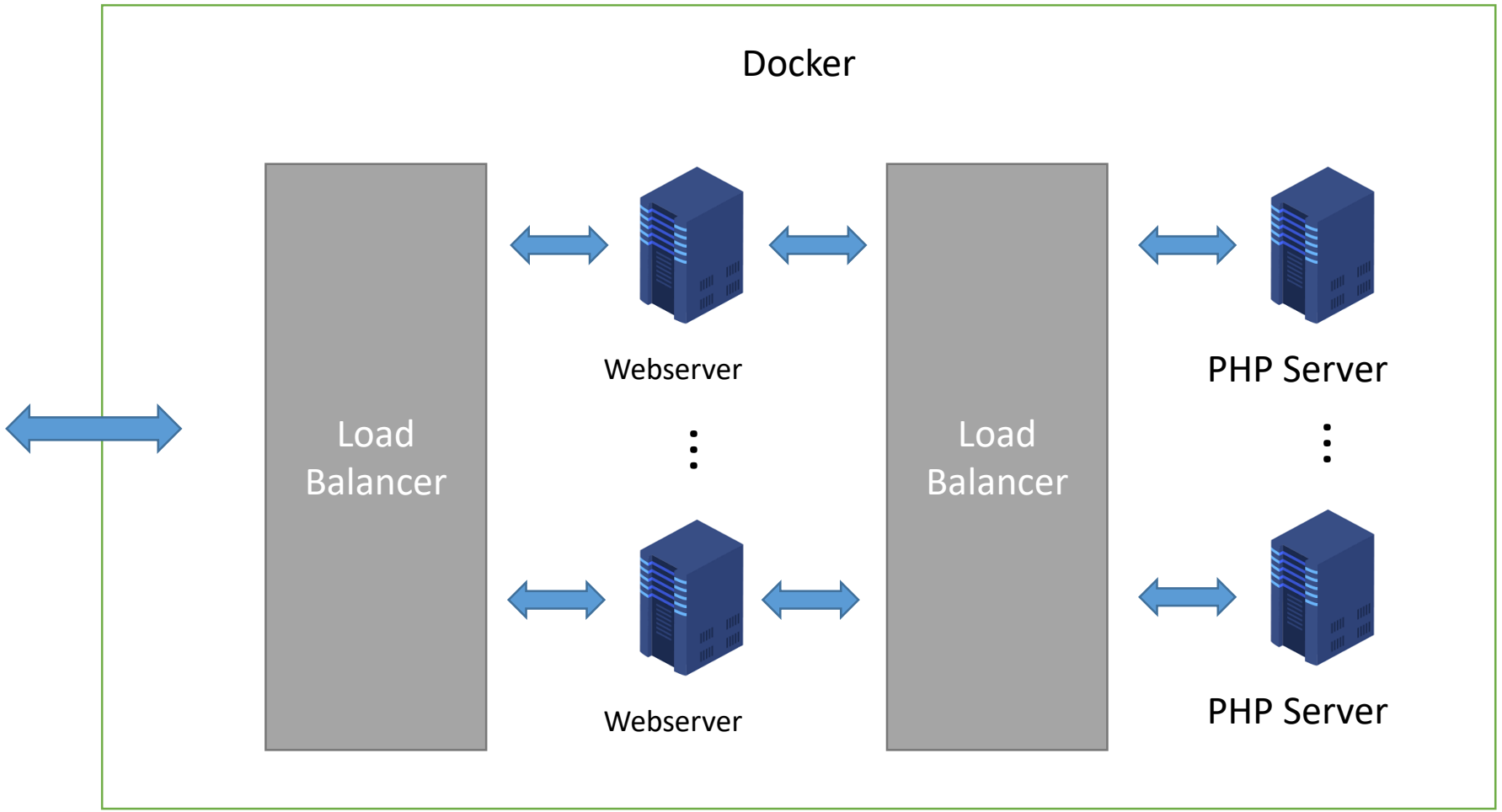
fastcgi_pass beheaves like proxy_pass it accepts both an host or an upstream

Let's build a LEMP stack: PHP Load Balancer



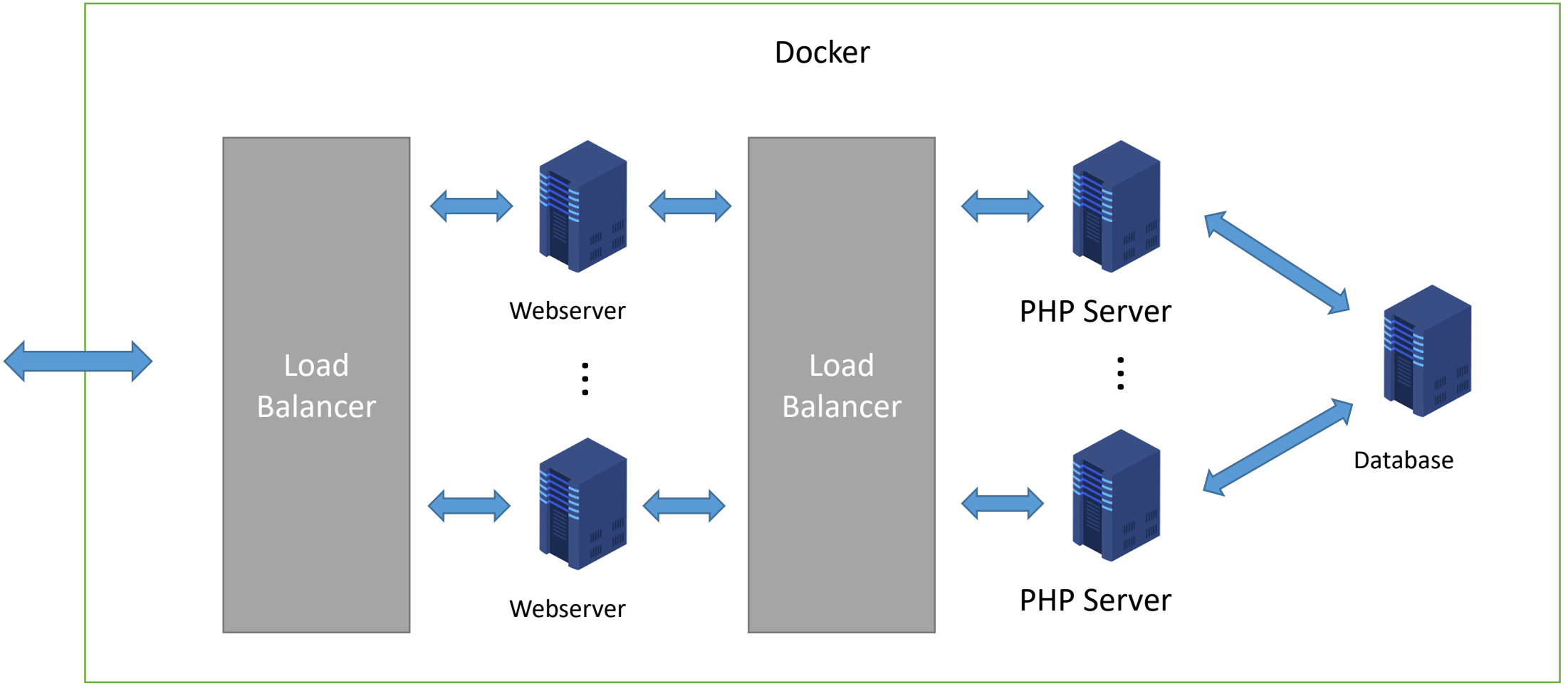
You just need to scale the php container and add a couple of lines to the webserver configuration

Let's build a LEMP stack: HTTP Load Balancer



Scale the webserver container, add another container just for the HTTP reverse proxy and expose **its port** to the host.

Let's build a LEMP stack: MySQL Server



Let's build a LEMP stack: MySQL Server

php:7-fpm **does not** have mysqli installed:

- Create your own container with a Dockerfile
  - Start from php:7-fpm
  - Execute docker-php-ext-install mysqli
  - Use this image in your docker-compose (with “build”)

Use the mysql image mysql:5.7

- set the environment variable MYSQL_ROOT_PASSWORD