Kuliah Tamu - Universitas Negeri Surabaya, 25 Nov 2021

Simplify Your Deployment Application with Docker

Heronimus Tresy Renata Adie

Site Reliability Engineer
PT Nodeflux Teknologi Indonesia

Heronimus Tresy Renata Adie (Roni)

Site Reliability Engineer

PT Nodeflux Teknologi Indonesia

- Web : heronimus.id
- Email : adie@heronimus.id
- LinkedIn : linkedin.com/in/heronimustra/

Contents

Disclaimer

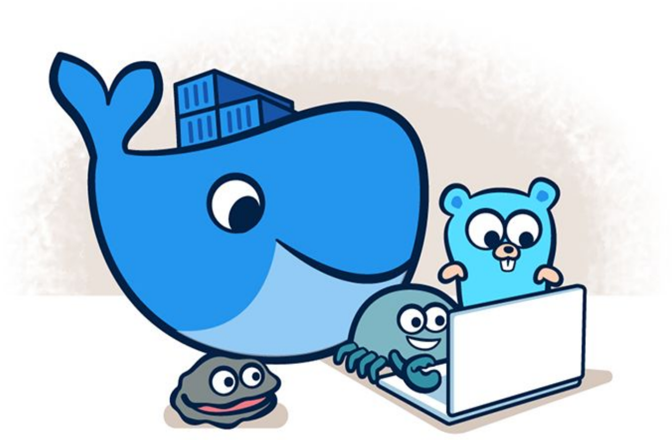
1. I try to present **basic introduction** in this session, let’s learn together!
2. Only as stepping stone, 1,5 hours isn’t enough not understand everything. **Follow up learning is a must**.
3. The presentation material is in English, but I’ll explain in Indonesian. Please ask right away if you confused with the terms used.
4. Don’t forget to **prepare your question**.

:)

Why Docker?

Why Docker exist? Why we must use Docker?

- What is deployment ?
- State before Docker.
- Problem Docker trying to solve.
- Docker use case (for Univ. Student).



What is “Deployment” ?

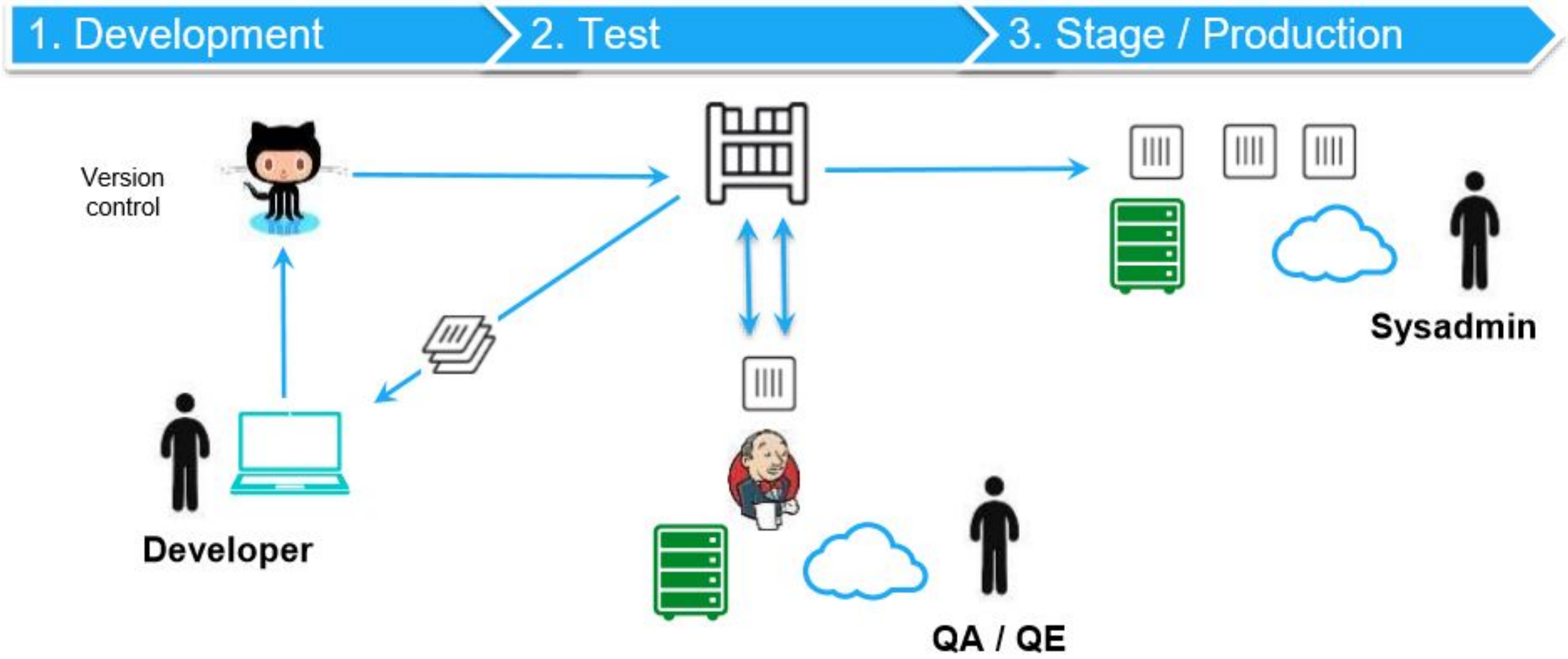
*“Simplify Your **Deployment** Application with Docker”*

+ **Deployment** is a process in getting software/application running properly.

+ **Involves**:
    - Configuration
    - Dependency
    - Installation
    - Multiple Environment (testing - production)

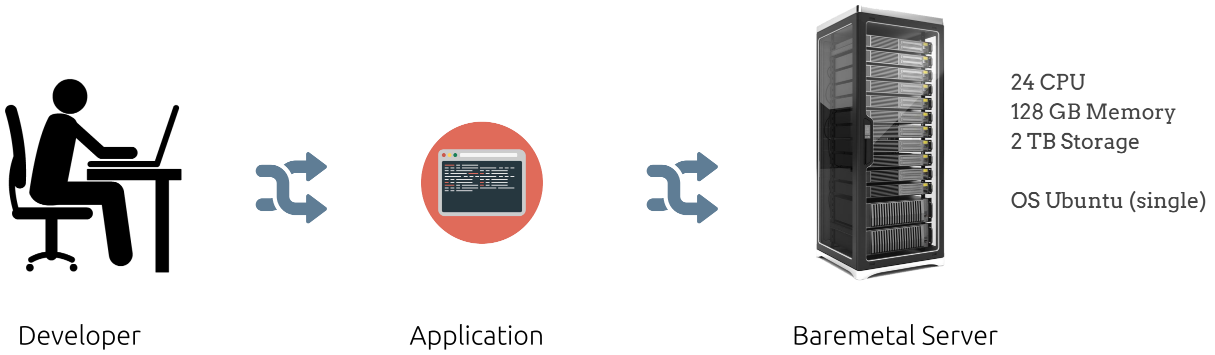
+ **Target**:
    - Server (On Premises & Cloud)

+ **Goals**:
    - Apps is running, customer can access, delivered.


1. Development
2. Test
3. Stage / Production
Version control
Developer
QA / QE
Sysadmin

State before Docker

- Directly to Server (Baremetal)



(+) Simple process

(-) Not Efficient
(-) Become complex on large scale application
(-) Not Isolated

State before Docker

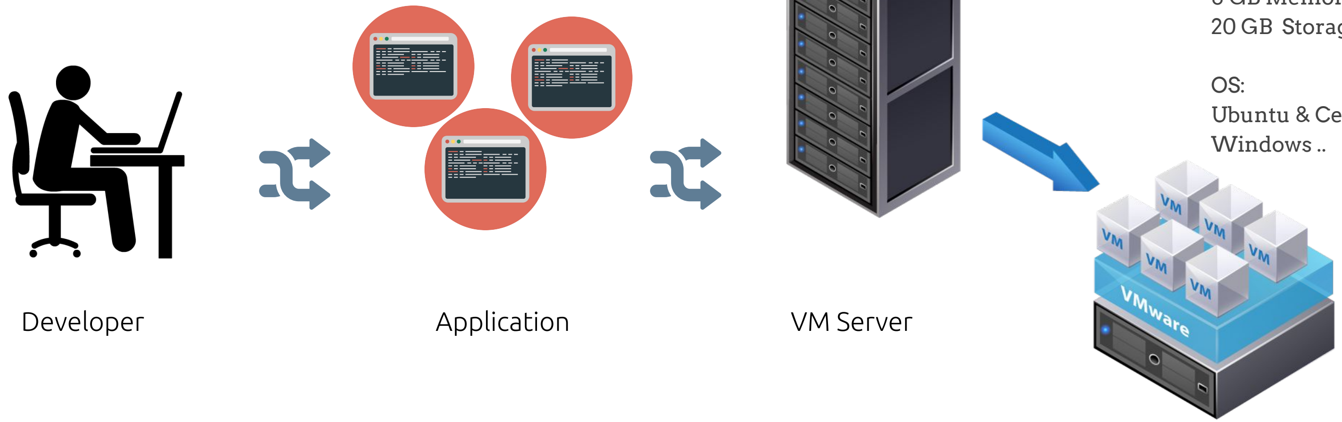
- Virtual Machine

24 CPU
128 GB Memory
2 TB Storage

OS Ubuntu (single)

@ Each VM:
4 CPU
8 GB Memory
20 GB Storage

OS:
Ubuntu & CentOS & Windows ..



(+) Resource Efficient
(+) Isolated

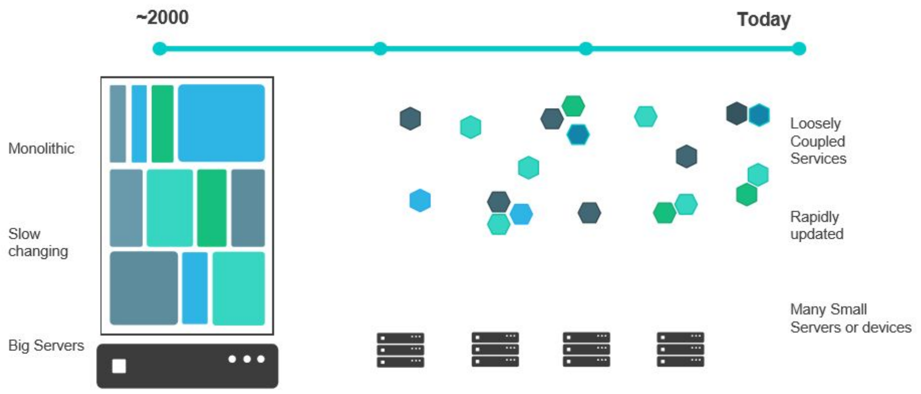
(-) Hard to maintain (configure many os)
(-) Complex to scale

Problem Trying to Solve

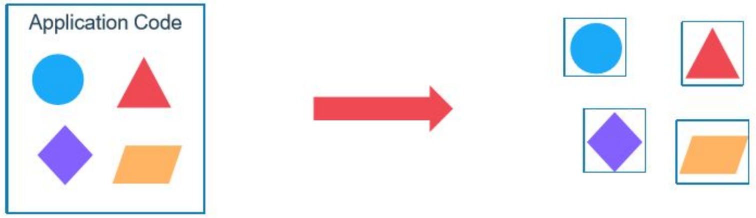
- Application is **evolving** (Monolith → Microservices)
- **Complex** configuration on **deployment** process

  (example: database, os version, dependency version)
- Environment **Isolation**
- **Scalability**
- **Efficiency**

Applications are transforming



Application Modernization



Developer Issues:

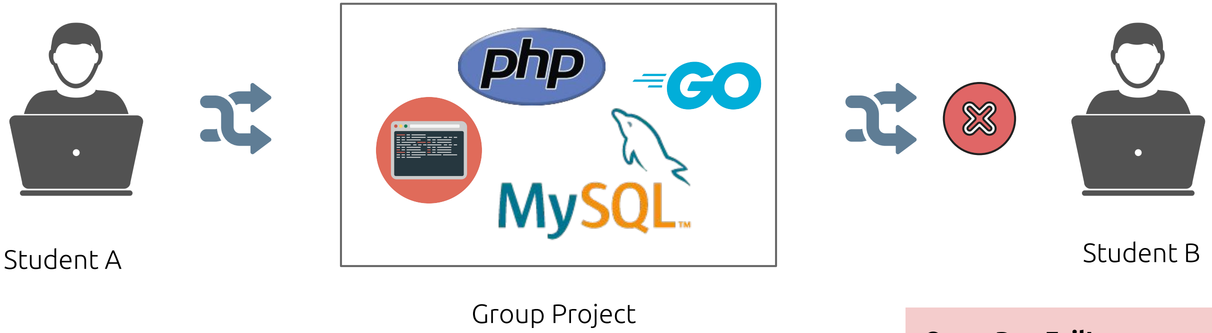
- Minor code changes require full re-compile and re-test
- Application becomes single point of failure
- Application is difficult to scale

Microservices: Break application into separate operations

12-Factor Apps: Make the app independently scalable, stateless, highly available by design

Docker Use Case (for Univ. Student)

- Problem when sharing project/task in a group

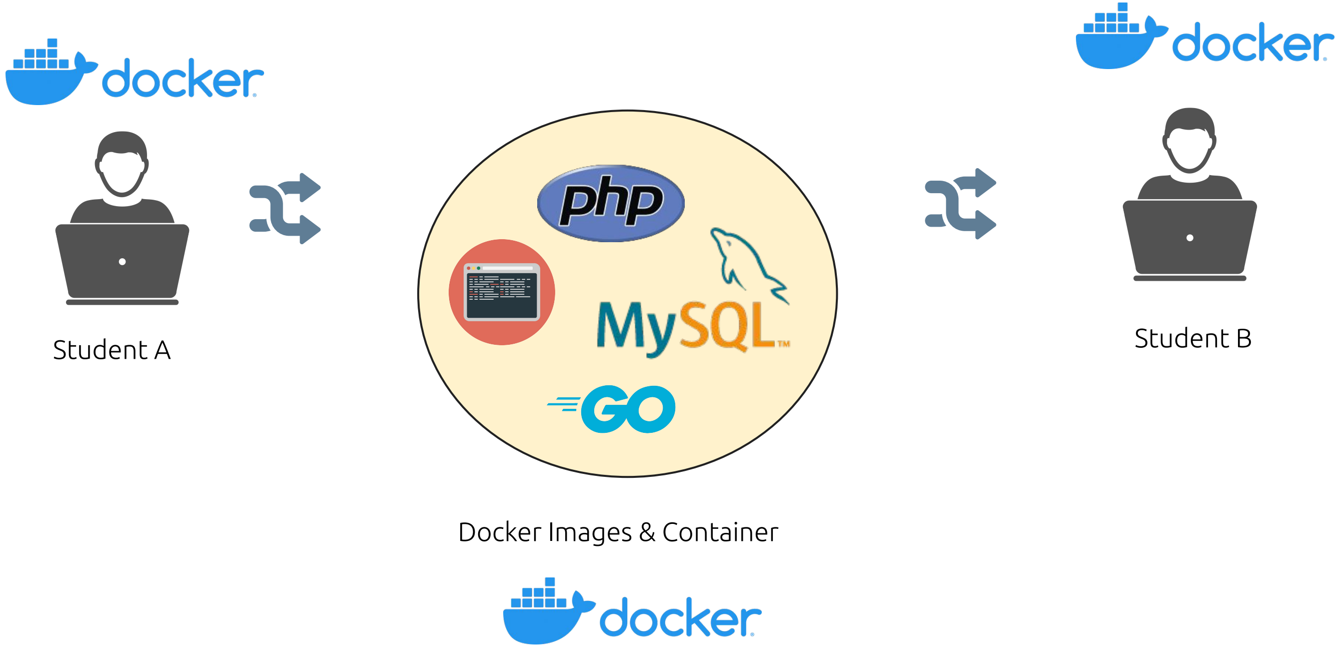


Opps, Run Fail!

- Version miss-match
- Different OS
- “magic” behaviour

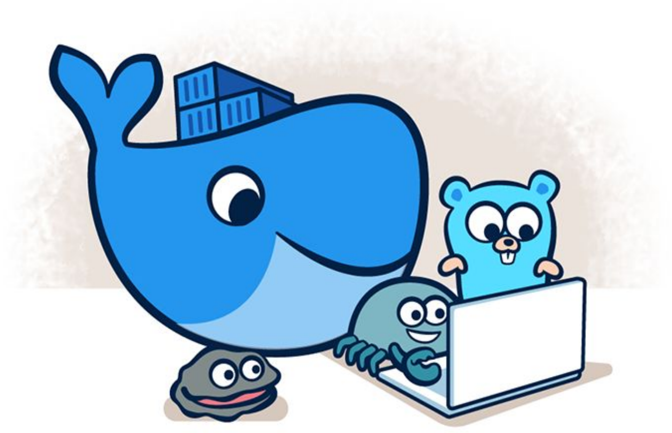
Docker Use Case (for Univ. Student)

- **Easy sharing** project/task in a group

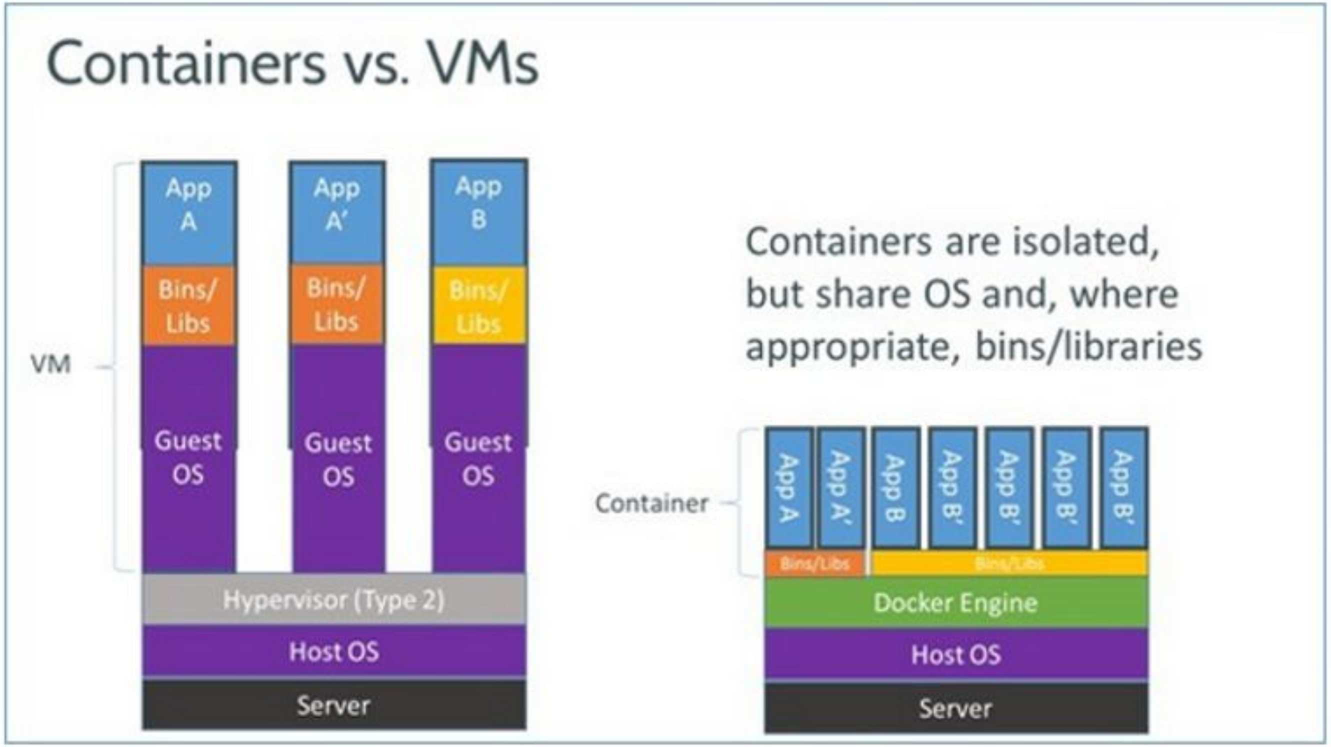


Inside Docker

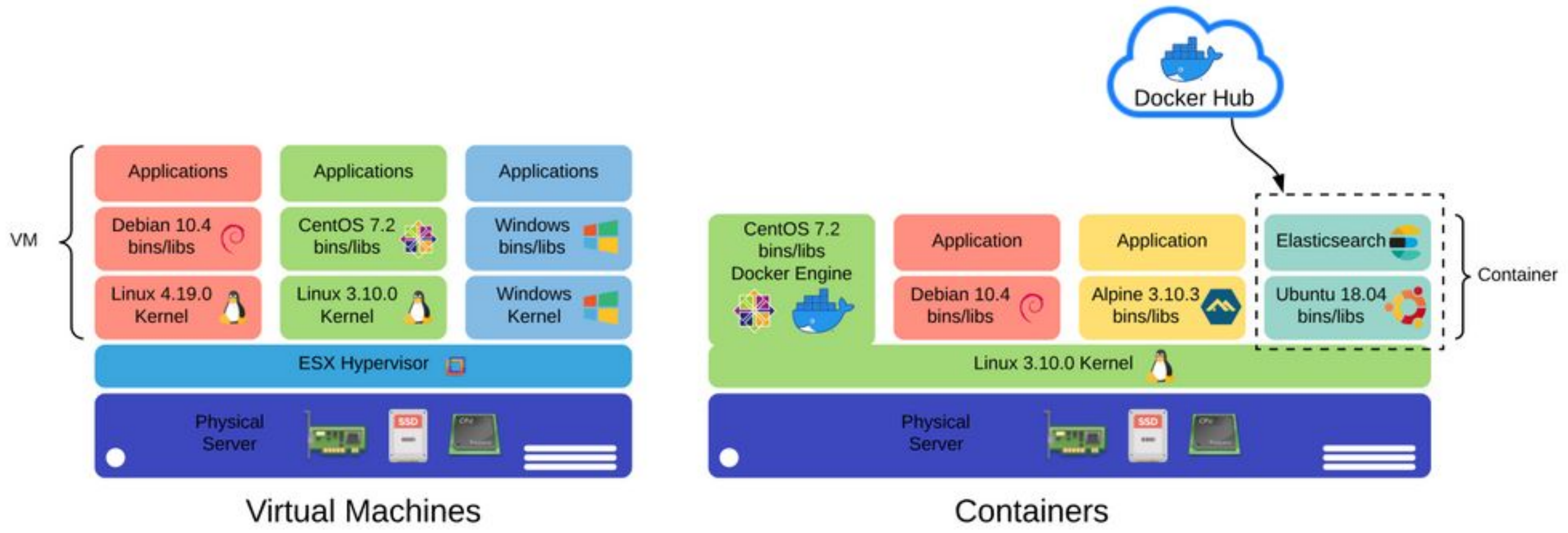
- Docker Architecture
- Dockerfile
- Images
- Container
- Registry
- Engine & Control Plane



Docker Architecture



Docker Architecture



Component: Docker Images

- **Docker Images** is **template** with instructions to creating application container.

  Template contain:
  - Configuration
  - Dependency
  - Apps files (.php .py .html … etc)

* Docker images is object that “**delivered**” for deployments.
* **Dockerfile** used to create docker images.

Component: Dockerfile

+ **Dockerfile** is **template** with instructions to creating application container.

Example: **Dockerfile** with php 7 & php 5

```
FROM php:7.4-cli

COPY /home/my-laptop/myapp /myapp

CMD [ "php", "/myapp/your-script.php" ]
```

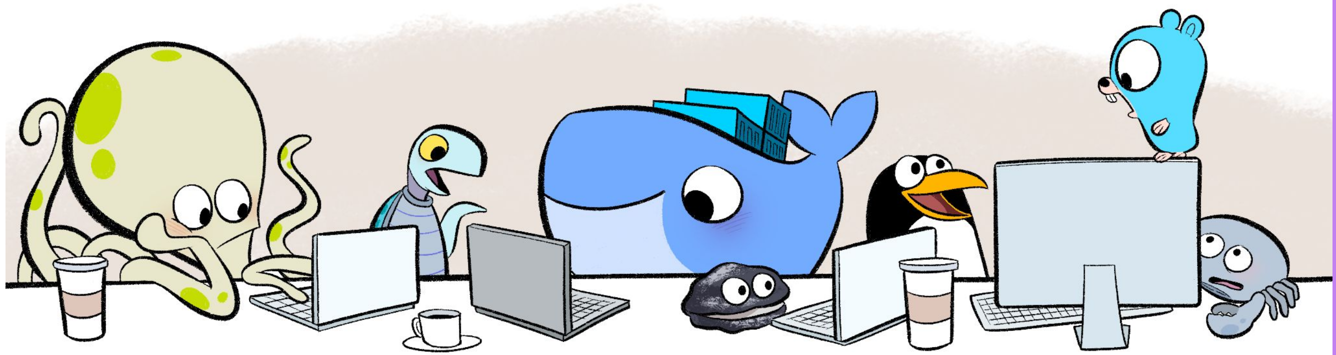
```
FROM php:5.6-cli

COPY /home/my-laptop/myapp /myapp

CMD [ "php", "/myapp/your-script.php" ]
```

```
FROM python:3

WORKDIR /usr/src/app

COPY requirements.txt ./

RUN pip install -r requirements.txt

COPY . .

CMD [ "python", "./my-app.py" ]
```

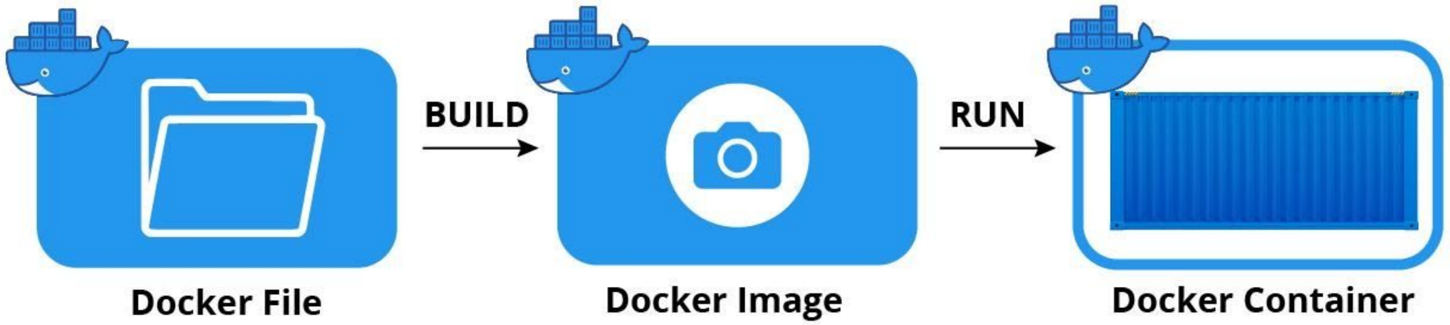
Component: Container

+ **Docker Container** is **runnable** instance of docker images.

  Container can be:
  - Start
  - Stop
  - Delete
  - Scale (create replica)



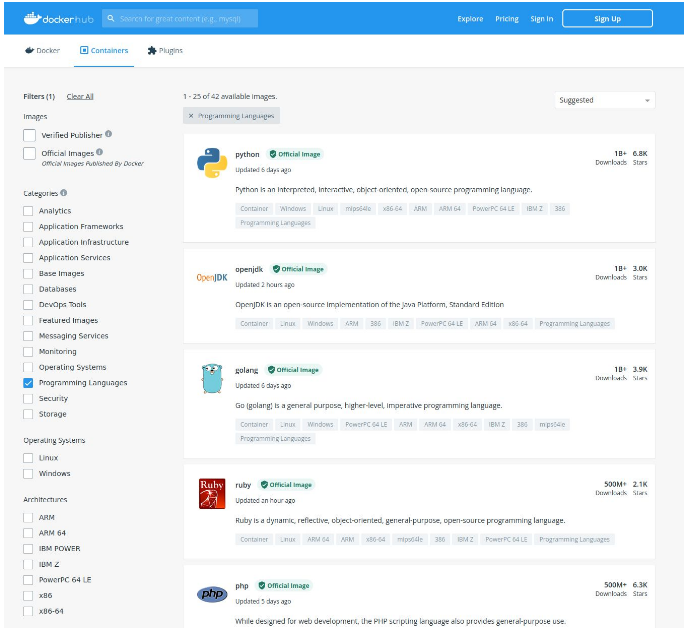
Docker Component



Docker Registry

- **Docker Registry** is **marketplace** for docker images.

  You can use **ready-to-use** template from others.



Filters (1) Clear All

Images
- [ ] Verified Publisher
- [ ] Official Images
  Official Images Published By Docker

Categories
- [ ] Analytics
- [ ] Application Frameworks
- [ ] Application Infrastructure
- [ ] Application Services
- [ ] Base Images
- [ ] Databases
- [ ] DevOps Tools
- [ ] Featured Images
- [ ] Messaging Services
- [ ] Monitoring
- [ ] Operating Systems
- [x] Programming Languages
- [ ] Security
- [ ] Storage

Operating Systems
- [ ] Linux
- [ ] Windows

Architectures
- [ ] ARM
- [ ] ARM 64
- [ ] IBM POWER
- [ ] IBM Z
- [ ] PowerPC 64 LE
- [ ] x86
- [ ] x86-64

1 - 25 of 42 available images.

× Programming Languages

Suggested

python Official Image — 1B+ Downloads, 6.8K Stars
Updated 6 days ago
Python is an interpreted, interactive, object-oriented, open-source programming language.
Container | Windows | Linux | mips64le | x86-64 | ARM | ARM 64 | PowerPC 64 LE | IBM Z | 386 | Programming Languages

openjdk Official Image — 1B+ Downloads, 3.0K Stars
Updated 2 hours ago
OpenJDK is an open-source implementation of the Java Platform, Standard Edition
Container | Linux | Windows | ARM | 386 | IBM Z | PowerPC 64 LE | ARM 64 | x86-64 | Programming Languages

golang Official Image — 1B+ Downloads, 3.9K Stars
Updated 6 days ago
Go (golang) is a general purpose, higher-level, imperative programming language.
Container | Linux | Windows | PowerPC 64 LE | ARM | ARM 64 | x86-64 | IBM Z | 386 | mips64le | Programming Languages

ruby Official Image — 500M+ Downloads, 2.1K Stars
Updated an hour ago
Ruby is a dynamic, reflective, object-oriented, general-purpose, open-source programming language.
Container | Linux | ARM 64 | ARM | x86-64 | mips64le | 386 | IBM Z | PowerPC 64 LE | Programming Languages

php Official Image — 500M+ Downloads, 6.3K Stars
Updated 5 days ago
While designed for web development, the PHP scripting language also provides general-purpose use.

Docker Engine and Control Plane

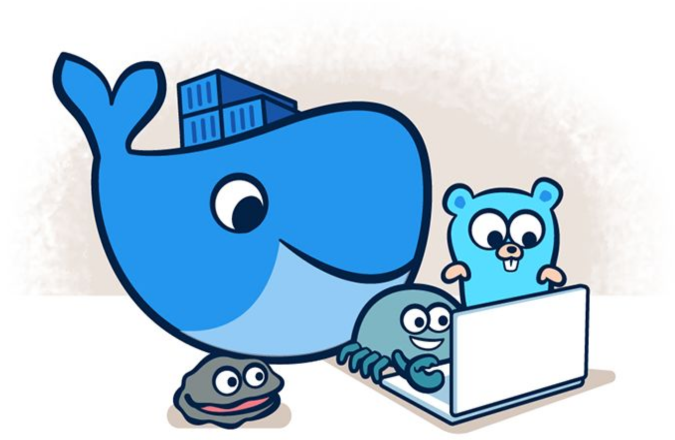
- **Docker Engine** is the main program/services that manage container.
- **Docker Control Plane** used to connect multiple node of docker.

Develop with Docker

Example:

- Simple HTML Web Server
- Laravel Web Apps

Example:
https://github.com/heronimus/sharing-unesa-docker



Simple HTML Web Server

index.html

```
<!doctype html>

<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Docker Nginx</title>
</head>

<body>
  <h2>Hello from Docker World</h2>
  <p>Kuliah Umum UNESA.</p>
</body>
</html>
```

Dockerfile

```
FROM nginx:latest

COPY ./index.html /usr/share/nginx/html/index.html
```

Build Command

```
docker build -t my-webserver .
```

Run Command

```
docker run -it --rm -p 8080:80 --name web my-webserver
```

Laravel Web Apps

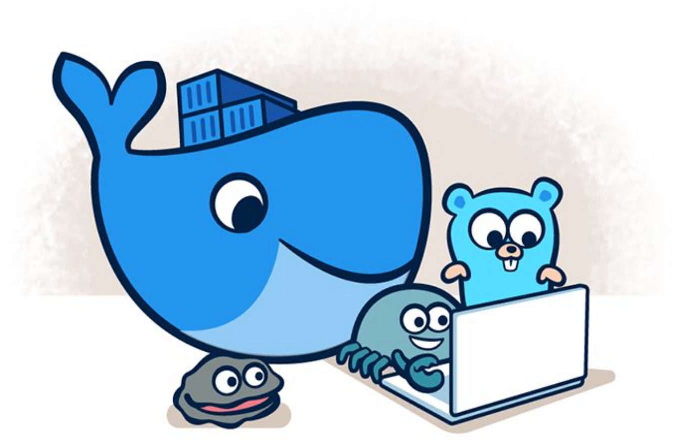
Dockerfile

```
FROM php:7.4
RUN apt-get update -y && apt-get install -y openssl zip unzip git libonig5 libonig-dev libzip-dev

## Install Composer
RUN curl -sS https://getcomposer.org/installer | php -- --install-dir=/usr/local/bin --filename=composer
RUN docker-php-ext-install pdo mbstring

## Copy Laravel Apps
WORKDIR /app
COPY ./laravel-app /app
RUN composer install

## Run PHP Serve
CMD php artisan serve --host=0.0.0.0 --port=8080
EXPOSE 8080
```

Extras

- IT always evolve ..
- Always learning, it’s free :)

- After familiar with docker, what’s next?
    - docker-compose
    - docker-swarm / kubernetes



Laravel Web Apps

Build Command

```
docker build -t my-laravel-image .
```

Run Command

```
docker run -it --rm -p 8181:8080 --name laravel my-laravel-image:latest
```