Docker

Kurzgesagt - In a Nutshell

About Me

- Apprenticeship System Admin/Engineer
- CS Degree at BFH
- Consulting Years
  - Public Cloud Provider, Telco Provider, Medtech
  - DevOps / Automation Engineer, Software Engineer
  - .NET Core, Java Spring Boot and a lot of Tooling
- Securiton
  - Intrusion Alarm System
  - Software Engineer
  - C++, Go and a lot of Tooling

Christian Nydegger
LinkedIn

Intro

Goals of today's lecture

- You can classify Docker
- You know the basic concepts of Docker
- You can apply those concepts
- You know about Docker-Compose

What are Containers?

What are Containers?

A container is a ***standard unit of software*** that ***packages up code and all its dependencies*** so the application runs quickly and reliably from one computing environment to another.

- Wait.. that just sounds like an ordinary software package?
  - Yes but it is so much more
- A Software package can be
  - an executable like MS Word
  - a library like Flask or Numpy
  - a simple python script
- An application often depends on libraries

What are Containers?

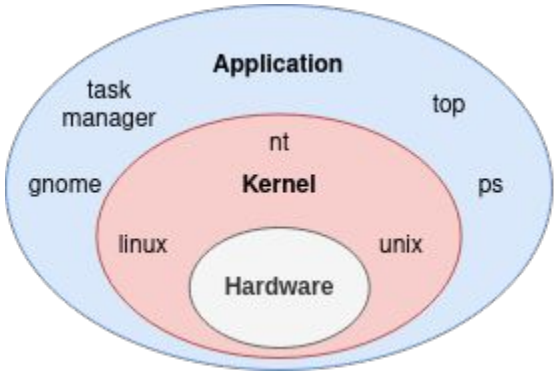
OS-level ***virtualization*** is an operating system (***OS***) paradigm in which the ***kernel*** allows the existence of ***multiple isolated*** user space ***instances, called containers***..

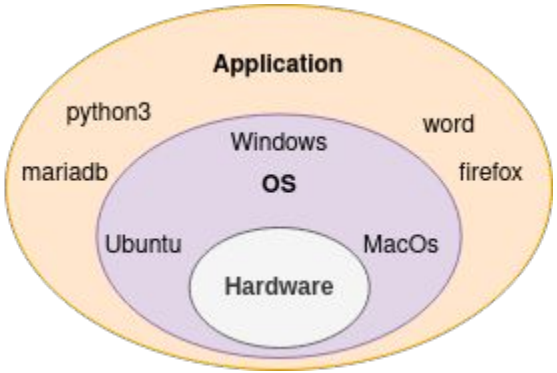
- So it is a virtual machine?
  - Yes, in a way :)

What are Containers?

What defines an Operating System?

- Kernel abstracts Hardware
- OS is more than a Kernel
  - It also includes various tools like

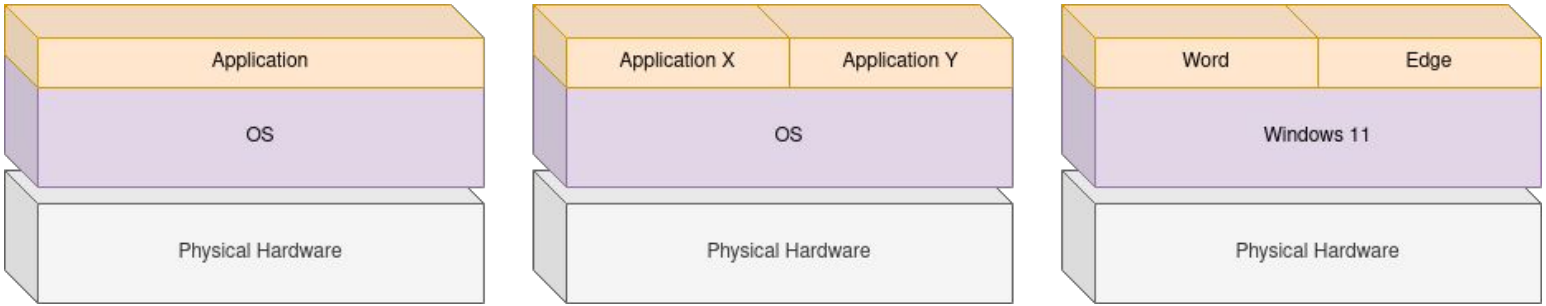




What are Containers?

Bare Metal Machine

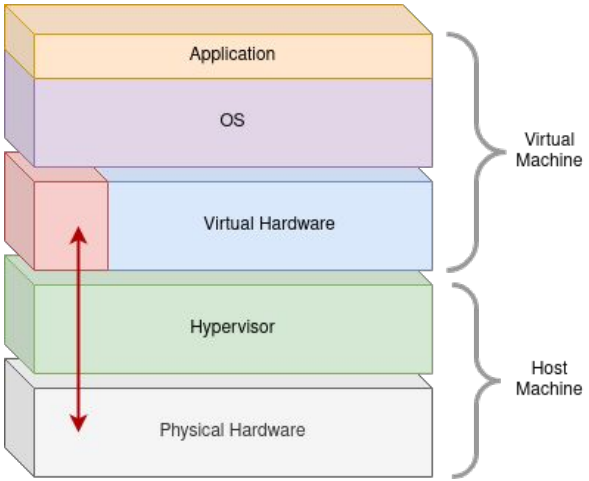
- Applications can
  - pack up all dependencies
    - Lots of duplications but no interference
  - make us of shared dependencies
    - No/less duplications but interference

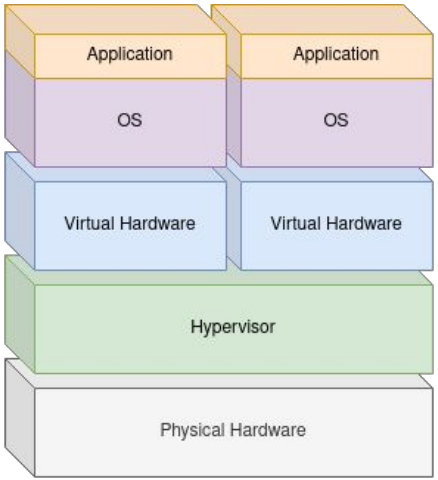


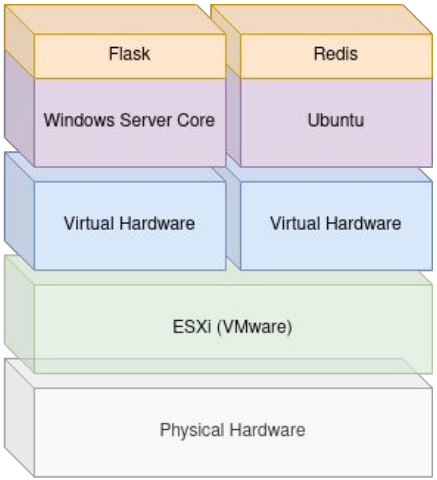
What are Containers?

“Traditional” Virtual Machine

- Isolated Instance with its own Hardware and OS



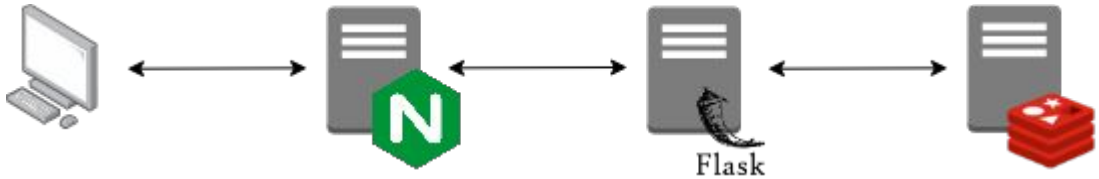




What are Containers?

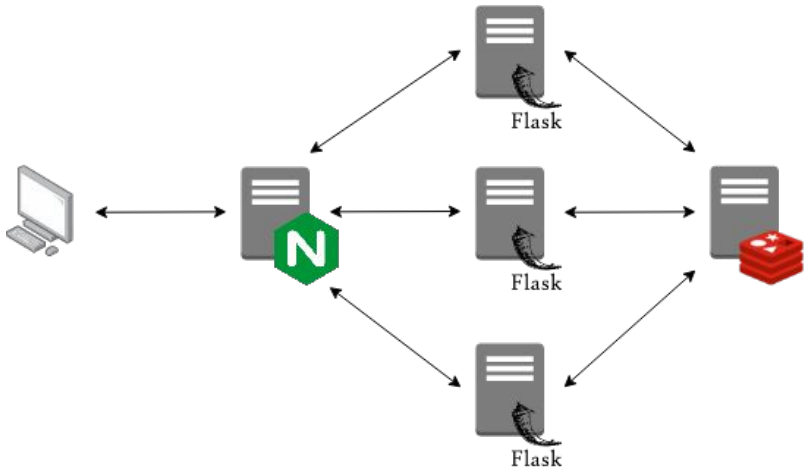
Project Setup

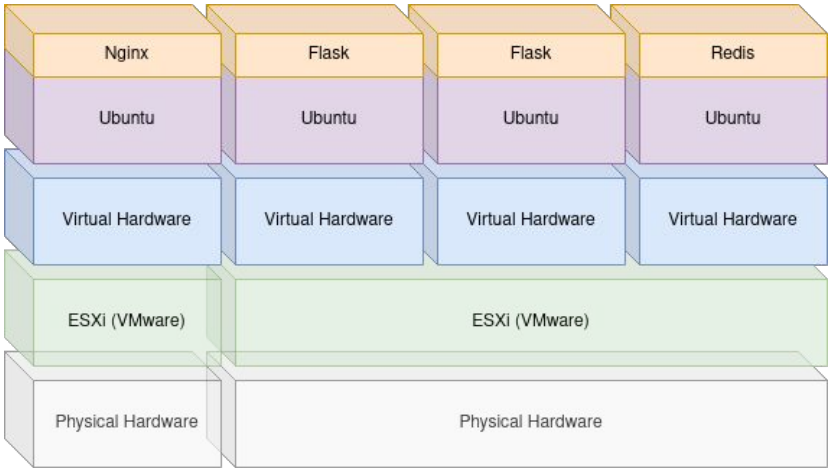
- Simple Web Application Setup
- Nginx as Load Balancer
- Flask to implement Rest Service
- Redis as Persistence Layer



What are Containers?

“Traditional” Virtual Machine





What are Containers?

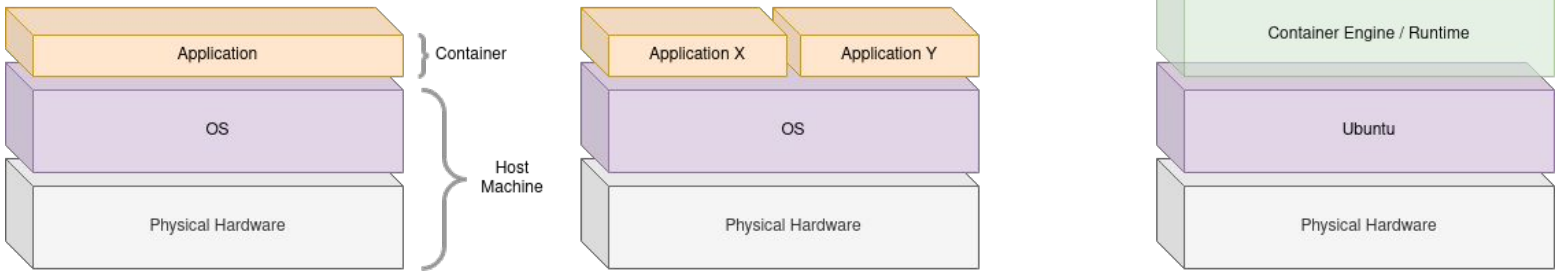
VM vs. Bare Metal Machine

- Isolated Instance
- Improved Resource Economy
- Horizontal vs. Vertical Scalability
  - Memory Slots are limited :)
- Virtualization Overhead
  - Loss of Performance
  - Lots of Duplications

What are Containers?

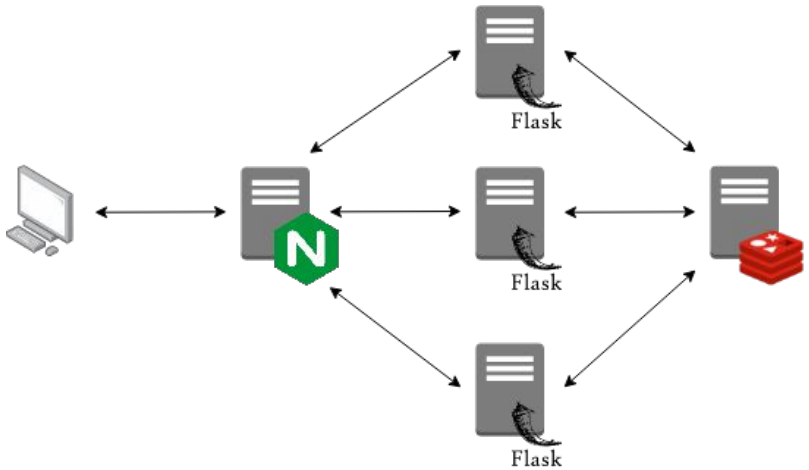
Container

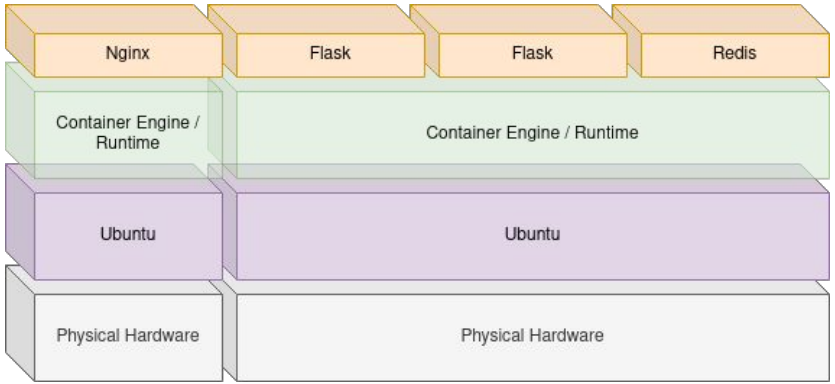
- Isolated Instance
- No overhead of Virtual Hardware or Multiple Operating Systems
  - Isolation is achieved by kernel features not virtualized hardware



What are Containers?

Container





What are Containers?

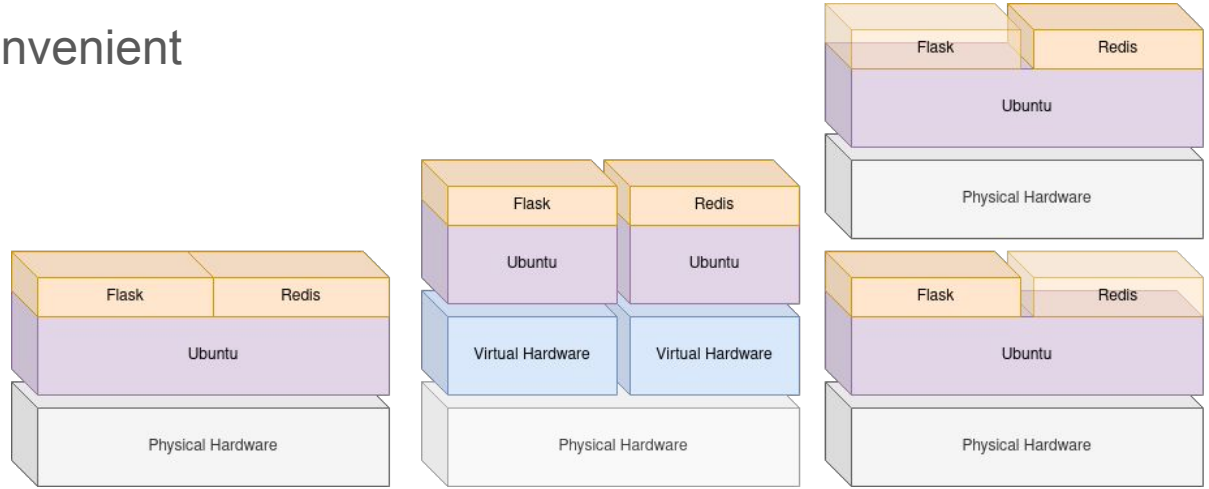
Container vs. VM

- Less overhead
  - Isolation not by virtualized hardware
- Increased Performance
  - Direct Hardware Access
  - There is a tendency
- Smaller Footprint
  - There is a tendency

What are Containers?

Bare Metal vs. Virtual Machine vs. Container

- More flexibility
- More efficient
- More convenient



Break

If there are any questions, feel free to approach me

Docker Intro

- Set of Tools to work with Containers
- Alternatives
  - Podman
  - LXC
- Why Docker?
  - Well established
  - Big Community
- Terminology
  - Container
  - Image
  - Dockerfile
  - Registry

Docker Intro

Container

- Runtime instance of a Docker Image
- Can be compared to an Object

Docker Intro

Image

Docker images are the basis of containers. An Image is an ordered collection of root filesystem changes and the corresponding execution parameters for use within a container runtime. An image typically contains a union of layered filesystems stacked on top of each other. An image ***does not have state and it never changes***.

- Blueprint to instantiate Containers from
- Can be compared to a Class

Docker Intro

Dockerfile

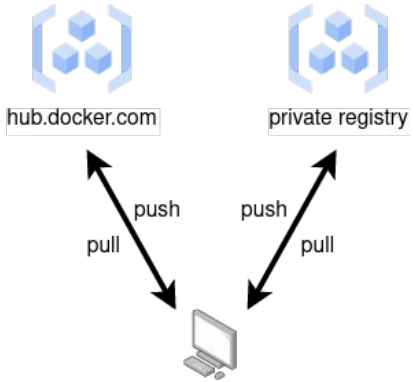
A Dockerfile is a text document that contains all the commands you would normally execute manually in order to build a Docker image.

- Instructions for Docker to build Image
- Declares how the Docker Image looks like
- A human readable representation of the Docker Image

Docker Intro

Registry

- Hosts Docker Images
  - Can be searched by *docker search*
- Default is hub.docker.com
  - Can be accessed by browser
- Private registry can be setup
  - Available as an Image itself



Docker Intro

Demo

Docker Intro

Process

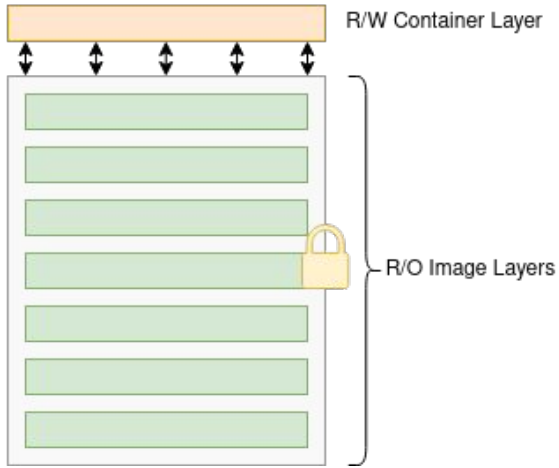
- Write Dockerfile
- Build Image from it
- Instantiate Image to run Container
- Push Image to Registry if desired

Docker Intro

Layer Concept

- Image consists of ReadOnly Layers
- Container ReadWrite Layer represents Container State

```
CMD [ "python", "./main.py" ]
COPY main.py ./
RUN pip install --no-cache-dir -r requirements.txt
COPY requirements.txt ./
WORKDIR /app
EXPOSE 5000
FROM python:3-alpine
```



Docker Intro

What not to do

- Treat a Container like a Virtual Machine
- Upgrade Containers
  - internals
  - Upgrade Dockerfile and rebuild Image instead
- Reuse Containers
  - Run a new container instead
  - If a container is gone, let it rest
- Run multiple Services in on Container
  - Run a container for each service instead

Break

If there are any questions, feel free to approach me

What problem might occur with Docker?

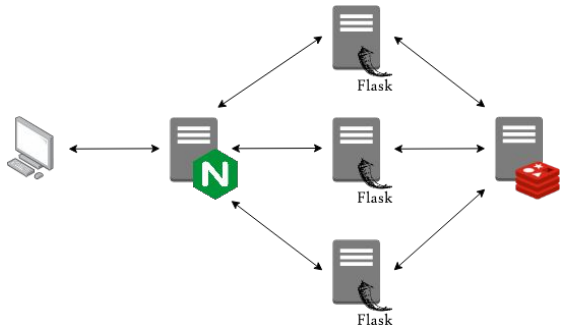
Docker-Compose Intro

Compose is a tool for defining and running multi-container *Docker* applications.

- Compose File
  - Instructions for Compose to configure and run individual Services
- Similar command set as Docker
  - Application level:
    - Up, Down, Build, ..
  - Container level:
    - Start, Stop, Run, ..

Docker-Compose Intro

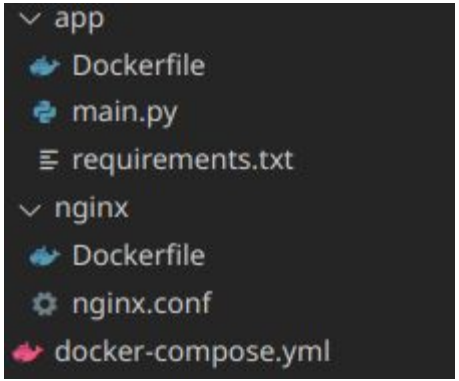
- CLI
  - Instantiate individual Containers with *docker run*
  - Very inconvenient and error prone
- Script
  - Essentially wrap individual commands in a bash script
  - Technically possible
  - Scripting vs. declaring
- Compose File
  - Declare your multi container application



Docker-Compose Intro

```yaml
version: '3.7'

services:
 app:
   # refers the directory to build image from
   build: app
   # container is only started once redis was started
   depends_on:
     - redis
 nginx:
   # explictly sets a container name instead of defering it
   container_name: nginx
   # refers the directory to build image from
   build: nginx
   # declares port mapping, equivalent to docker run -p 80:80 ..
   ports:
     - 80:80
   # container is only started once app was started
   depends_on:
     - app
 # more services if necessary
```



Docker-Compose Intro

Demo

Your Task

Dockerize a small web application

The goal is to implement a tiny rest service similar to the examples discussed during the lecture. It can be a simple ping or something a bit more sophisticated. The only requirement for the service is that the persistence layer is used. The example discussed during the lecture implemented a simple hit count stored in a redis store.

Other requirements are:

- The rest service and all its dependencies **must** be packed in a Docker Image.
- The redis store **must** be run as a container
- The application **can** be managed with docker-compose
- The application **can** be load balanced
- It is **recommended** to us redis and flask but **not a must**

Deliverables:

- All relevant files
- Either pack them up or push them to a git repository

Q&A