

Compute Services

Boris TEABE

`boris.teabe@inp-toulouse.fr`

Goals

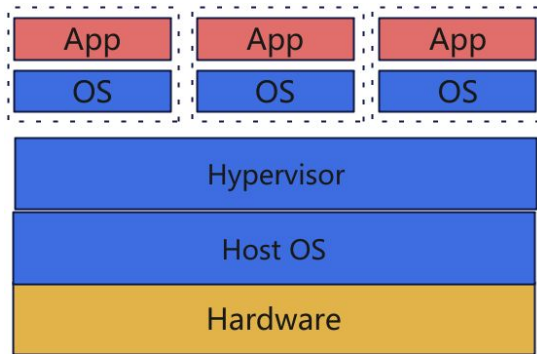
1. Bare Metals
2. Virtualization
3. **Containers**
4. Unikernels/MicroVM

Containers

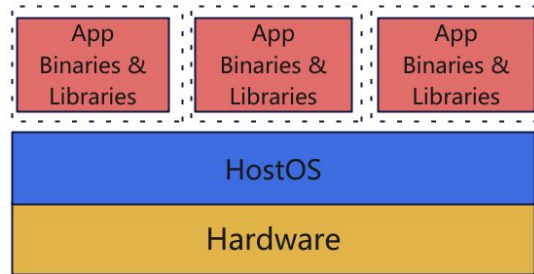
- Lightweight Virtualization
 - Represent virtualization solutions aiming at providing, vs. traditional virtual machines:
 - Lower memory footprint: 0 to a few MB per virtualized instance
 - Lower disk footprint: in the order of KBs/MBs
 - Faster boot times: in the order of micro/milliseconds
- **Simple examples are containers and unikernels**

Containers

- **Containers: process-level sandboxing technologies**
 - Enforced by the operating system
 - **Called OS-level virtualization**



Traditionnal virtualization



Containers

Containers

- The OS restricts the visibility on system resources for a process or a set of processes
- The OS also controls hardware resource allocation/usage between such isolated processes
 - CPU cores, memory, disk/network bandwidth, etc.
- A container is much lighter than a VM
 - Per-container system memory/disk footprint close to 0
 - Boot time is that of spawning a process, i.e micro seconds
- Still containers are not an ideal form of virtualization
 - **Security issues**

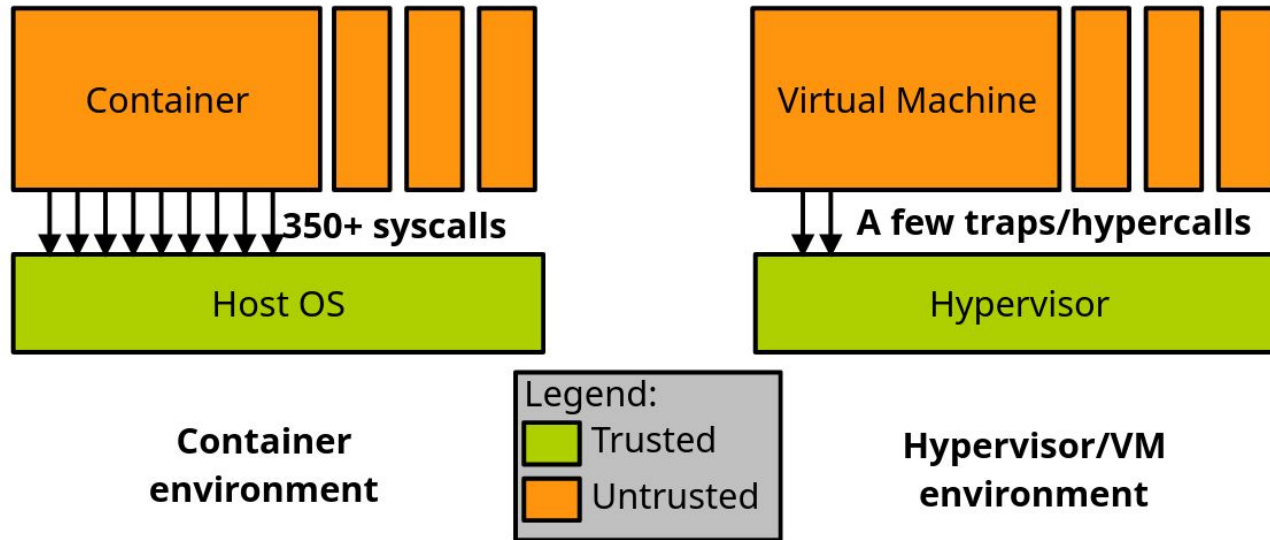
Containers

- **Uses cases**
- **Software development/testing/deployment**
 - Develop, build and test in a controlled, identical environment
 - Deploy in the same environment as the development one (repeatability)
 - Can be deployed on any server/cloud supporting containers independently of the host configuration
- **Lightweight (low cost) & elastic virtualization**
 - Containers consume few resources and can be brought up/destroyed very fast
 - Cloud services such as Gmail and Facebook make extensive use of containers
 - Function as a Service (e.g. AWS Lambda)

Containers

- **Software Development with Containers**
- **Package application programs and dependencies**
 - One of the main benefits: ease of development/testing/deployment
 - “Shipping containers”
- **Developing and running application X requires a complex set of dependencies**
 - Libraries sources and/or binaries (ex: glibc, etc.)
 - Build tools (ex: cmake, autotools, etc.)
 - System tools (ex: perl, grep, etc.)
 - All of these with sometimes very specific versions that may not be compatible with that of application Y that we also want to build & run
- **One solution would be to build and run application X and Y each in their own VM**
 - Too heavy, does not scale to a high number of apps
 - **Containers can help!**

Containers and security



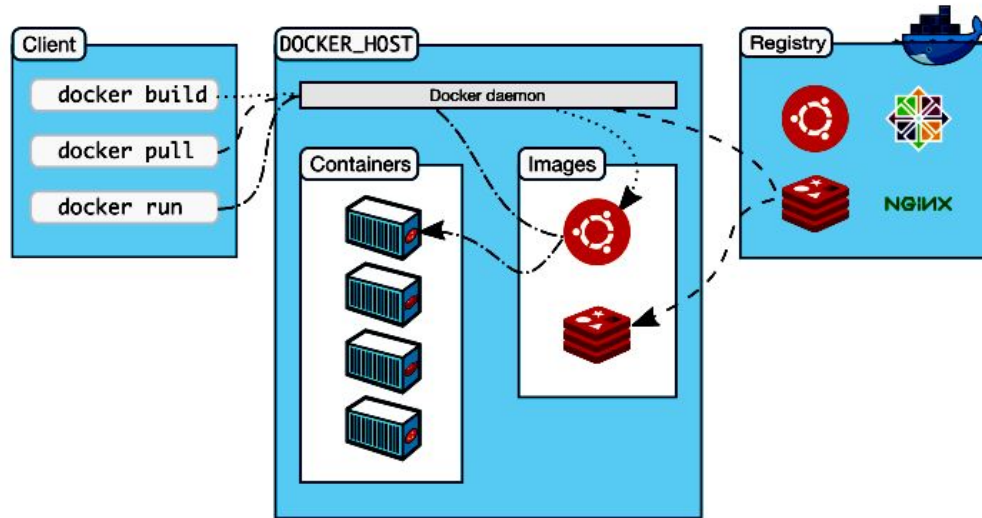
Containers and security

- The isolation enforced by a host OS between containers is not trusted to be as strong as that enforced between VMs by a hypervisor
 - Due to the size and complexity of the interface between a container and the host OS: **the system call interface**
- Attempts at securing containers:
 - Run containers within virtual machines...



Docker

- Virtualization system Allow building very light VMs (containers)
- Set of user-friendly tools for managing containers
- Client-server architecture



Docker

- **The image of a VM**
 - Docker relies on Union File System for the representation of images
 - An image is represented as a set of layers
 - Each layer describes a modification of the file system (like diff)
- **Advantages of this representation**
 - Allows building a file system
 - From a standard image
 - With small additional data (tens of Mb instead of hundreds of Mb)
 - Efficiently
- **The same set of standard images can be reused**
- **The modification of a file system does not generate a full file system (only a layer)**
 - Only diffs are saved
- **Docker allows sharing images**
 - <https://hub.docker.com>

Docker Core Technical Concepts

Namespaces

- **Namespaces** = isolation of resources per container
 - **PID** → isolate process IDs
 - **NET** → isolate network interfaces, IPs, ports
 - **MNT** → isolate file system mount points
 - **IPC** → isolate inter-process communication
 - **UTS** → isolate hostnames/domain names
 - **USER** → isolate user/group IDs

Docker Core Technical Concepts

Control Groups

CGroupe = Resource Management

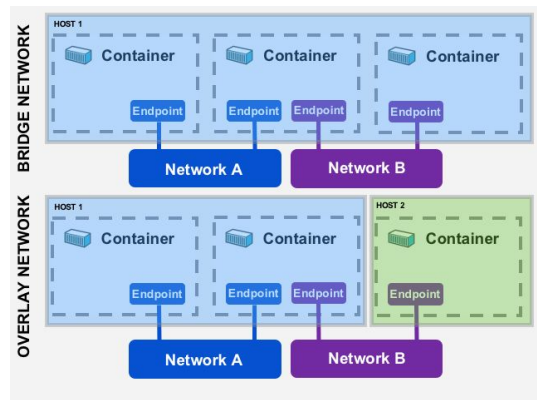
- Track and limit container resource usage
- **CPU**: pin processes to cores, enforce quotas
- **Memory**: enforce limits, prevent out-of-memory kill of host
- **I/O**: throttle disk/network access
- Enable fair resource sharing across container

Docker Core Technical Concepts

Container Networking

Connectivity models

- Bridge mode
 - Private container IPs, NAT via host
- Host mode
 - Shares host network stack, higher perf, less isolation
- Overlay network
 - Multi-host virtual networks (used in Kubernetes, swarm)



Docker

- **Some basic commands**
- Installation under Linux
 - `wget -qO- https://get.docker.com/ | sh`
- Starting a container
 - `docker run -it ubuntu bash`
- Lookup the image
 - If the image is not in the local registry, download from the hub
 - Ubuntu: pre-existing image in the hub
- Build the Linux file system
- Start the container
- Configure the IP address of the container
 - Also communication between outside and the container

Docker

- List local images
- docker images

```
hagimont@hagimont-pc:~$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
ubuntu              latest             cd6d8154f1e1       12 days ago
84.1MB
hagimont@hagimont-pc:~$
```

- Log in the hub
 - docker login/logout
- Lookup an image in the hub
 - docker search hagimont

```
hagimont@hagimont-pc:~$ docker search hagimont
NAME                DESCRIPTION          STARS     OFFICIAL
AUTOMATED
hagimont/docker-whale      0
hagimont/hagi            my repo             0
lwapet/projet_docker     ENSP - hagimont Daniel  0
hagimont@hagimont-pc:~$
```


Docker

- Creation of an image
- From a container instance
 - Start the container (from an initial standard image)
 - Modify the file system (apt-get install ...)
 - Commit the instance with a new image name
 - `docker commit c8744fe9eab6 ubuntu:hagi`

```
hagimont@hagimont-pc:~$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
c8744fe9eab6        ubuntu             "bash"             4 seconds ago      Up 2 seconds
hagimont@hagimont-pc:~$ docker commit c8744fe9eab6 ubuntu:hagi
sha256:58bf3876c787780770f7c75740c675bf1c3ab4f34d128f48f8c5163e6a0df422
hagimont@hagimont-pc:~$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
ubuntu              hagi               58bf3876c787       6 seconds ago      84.1MB
ubuntu              latest             cd6d8154f1e1       12 days ago        84.1MB
hagimont@hagimont-pc:~$
```

Docker

- **Creation of an image**
- **From a Dockerfile**
 - `mkdir foo`
 - `cd foo`
 - Create a file Dockerfile
 - `# This is a comment`
 - `FROM ubuntu`
 - `RUN apt-get update && apt-get install -y apache2`
- `docker build -t hagimont/ubapache:v2 .`

```
hagimont@hagimont-pc:~/foo$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
hagimont/ubapache	v2	904d4cc37cdd	About a minute ago	222MB
ubuntu	latest	cd6d8154f1e1	12 days ago	84.1MB

```
hagimont@hagimont-pc:~/foo$
```

Docker

- **Management of images in the hub**
 - You must be logged in
 - Save the image in the hub
 - *docker push hagimont/ubapache:v2*
- **Download an image from the hub**
 - *docker pull hagimont/ubapache:v2*
- **Tag an image (versioning)**
 - *docker tag id_image training/sinatra:thetag*

Docker

- **Goal of data volumes**
 - make visible in one or more containers a directory or file from the host file system
 - Allows file sharing between several containers
- **Persistent even after container destruction**
- Any modification is immediately effective
- **Command:**
 - *docker run -it -v /tmp/host_file:/tmp/container_f*
- Port redirection
- Example of link: host → container
 - *docker run -d -p 80:5000 hagimont/apache*
- Any connection on port 80 of the host is forwarded to port 5000 of the container