Fog Computing
2025/26 - Autumn 2025 - UiO

Containers

Prof. Paulo Ferreira

paulofe@ifi.uio.no

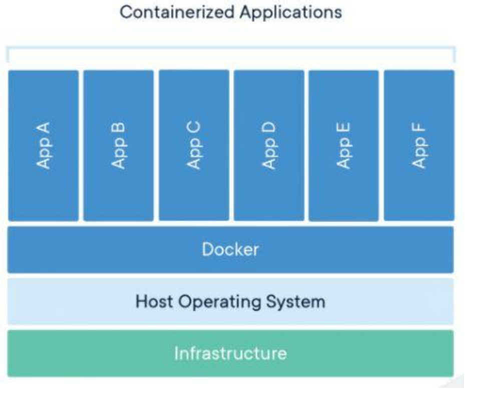
UiO/IFI/PT

1-**https://www.docker.com/; https://www.docker.com/resources/what-container**
2-**Docker Container Deployment in Fog Computing Infrastructures**, A. Ahmed and G. Pierre.
IEEE International Conference on Edge Computing (EDGE), San Francisco, CA, 2018, pp. 1-8
doi: 10.1109/EDGE.2018.00008.

© Paulo Ferreira

What is a Container?

- 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**
- A Docker **container image** is:
  - a **lightweight, standalone, executable package of software**
  - it **includes everything needed to run an application** (code, runtime, system tools, system libraries and settings)
- **Container images**:
  - **become containers at runtime**, and
  - in the case of Docker containers - **images become containers when they run on Docker Engine**
- **Containerized software** will always run the same, regardless of the infrastructure

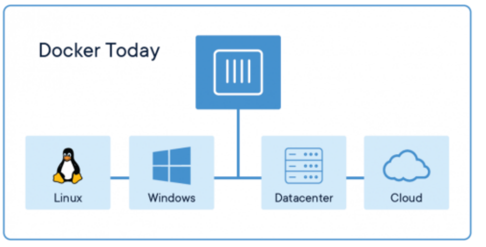


- Containers:
  - **isolate software** from its environment, and
  - ensure that it **works uniformly** despite differences (e.g., between development and instalation)

© Paulo Ferreira

Docker Containers

- **Docker containers that run on Docker Engine**:
  - **Standard:** Docker created the **industry standard** for containers, so they could be portable anywhere
  - **Lightweight: containers share the machine's OS system kernel and therefore do not require an OS per application,** driving higher server efficiencies and reducing server and licensing costs
  - **Secure: applications are safer in containers** and Docker provides the strongest **default isolation capabilities** in the industry



- It leveraged existing computing concepts around containers world, primitives known as **cgroups** and **namespaces**

© Paulo Ferreira 3

3

Other Container Management Solutions

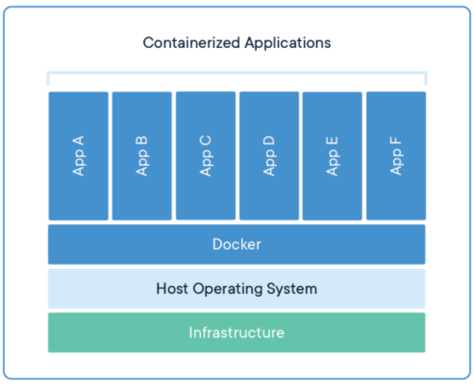
- **Docker Enterprise Edition**:
  - perhaps the **best known commercial container management solution**
  - it provides an **integrated, tested and certified platform** for apps running on enterprise Linux or Windows operating systems (obviously including cloud providers)
- But **there are many others**, and **several notable ones have a layer of proprietary software built around Kubernetes at the core**; examples of this type of management software product include:
  - **CoreOS's Tectonic** pre-packages all of the **open source** components required to build a Google-style infrastructure and **adds additional commercial features**, such as a management console, corporate SSO integration, and Quay, an enterprise-ready container registry
  - **Red Hat's Open Shift Container Platform** is an on-premises **private platform** as a service product, built around a core of application containers powered by Docker, with orchestration and management provided by Kubernetes, on a foundation of Red Hat Enterprise Linux
  - **Rancher Labs'** is a commercial **open source** solution designed to make it easy to deploy and manage containers in production on any infrastructure
  - **Kernel-based Virtual Machine (KVM)** is **an open source** option and is built into the Linux® kernel
  - **Microsoft Hyper-V, VMware Workstation and Oracle VirtualBox and VMware vSphere** are other examples

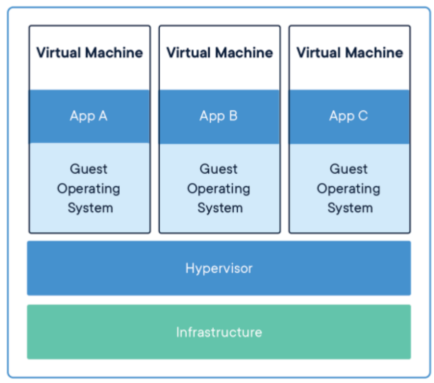
© Paulo Ferreira 4

4

Differences Between a VM and a Container (1/3)

- **Containers and virtual machines have similar resource isolation and allocation benefits** but **function differently**
- **Containers are more portable, efficient, smaller, and faster**

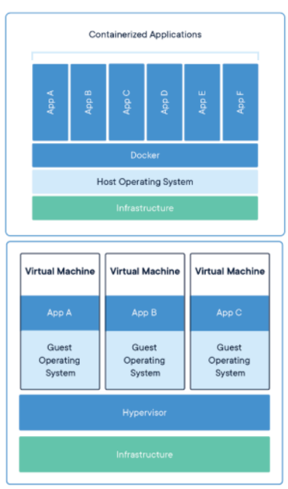




© Paulo Ferreira

Differences Between a VM and a Container (2/3)

- **Containers** are:
  - an **abstraction at the app layer that packages code and dependencies** together
- **Multiple containers** can:
  - **run on the same machine**, and
  - **share the OS kernel with other containers**, each running as **isolated processes** in user space
- **Containers**:
  - take up **less space than VMs** (container images are typically tens of MBs in size),
  - can **handle more applications and require fewer VMs and Operating systems**



- **Virtual machines** (VMs) are:
  - an **abstraction of physical hardware turning one server into many servers**
- The **hypervisor allows multiple VMs to run on a single machine**
- **Each VM includes**:
  - a **full copy of an operating system**, the **application**, **necessary binaries and libraries** - taking up **tens of GBs**
- **VMs can also be slow to boot**

- **Containers** and **VMs** used together provide a **great deal of flexibility** in deploying and managing app

© Paulo Ferreira

Differences Between a VM and a Container (3/3)

https://www.youtube.com/watch?v=a1LW8rDB874

© Paulo Ferreira

A Container in Docker (1/2)

- **Docker** is a popular framework to **build, package, and run applications** inside containers
- **Applications are packaged in the form of images**:
  - an **image** contains a part of a file system with the required libraries, executables, configuration files, etc.
- **Images are stored in centralized repositories**:
  - where they are **accessible from any computer**
- To **deploy a container**, Docker therefore:
  - **first downloads the image from the repository and locally installs it**
  - unless the image is already cached in the compute node
- **Starting a container from a locally-installed image** is as quick as starting the processes which constitute the container's application
- The **deployment time of any container is therefore dictated by**:
  - the **time it takes to download, decompress, verify**, and
  - **locally install the image before starting the container itself**

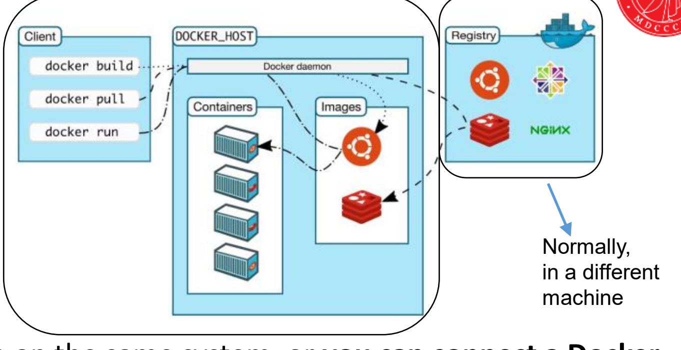
© Paulo Ferreira

A Container in Docker (2/2)

- 1) **Image structure**:
  - **Docker images** are composed of **multiple layers** stacked upon one another: **every layer may add, remove, or overwrite files present in the layers below itself**
  - this **enables developers to build new images very easily** by simply specializing pre-existing images
  - the same **layering strategy is also used to store file system updates performed by the applications after a container has started**:
    - upon every container deployment, **Docker creates an additional writable top-level layer which stores all updates following a Copy-on-Write (CoW) policy**
    - the **container's image layers themselves remain read-only**
- 2) **Container deployment process**:
  - **Docker images are identified with a name and a tag** representing a **specific version of the image**
  - **Docker users can start any container** by simply **giving its name and tag** using the command:
    - docker run IMAGE:TAG [parameters]
  - **Docker keeps a copy of the latest deployed images in a local cache**
  - **when a user starts a container**, **Docker checks its cache** and pulls the missing layers from the **docker registry** before starting the container

© Paulo Ferreira

Docker Architecture

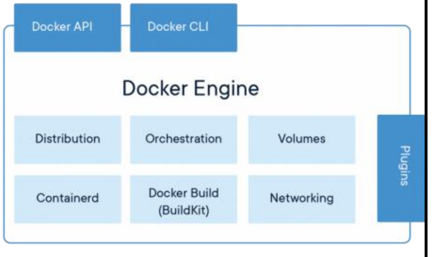


- Docker uses **a client-server architecture**
- The **Docker *client* talks to the Docker *daemon***, which does the heavy lifting of building, running, and distributing your Docker containers
- The **Docker client and daemon** *can* run on the same system, or **you can connect a Docker client to a remote Docker daemon**
- The **Docker client and daemon** communicate using a REST API, over UNIX sockets or a network interface
- **Another Docker client is Docker Compose**, that lets you work with applications consisting of a set of containers

© Paulo Ferreira

Docker has Several Products

- **Docker Hub**:
  - the world's leading **service for finding and sharing container images** with your team and the Docker community
- **Docker Desktop**:
  - the preferred **choice for millions of developers that are building containerized apps**
  - it is an application for the **building and sharing of containerized applications**
- **Container Runtime**:
  - **Docker Engine powers millions of applications worldwide**
  - it provides a **standardized packaging format for diverse applications**
- **Developer Tools**:
  - the **fastest way to securely build, test, and share cloud-ready modern applications** from your desktop
- **Kubernetes**:
  - it has **become the standard orchestration platform for containers**
  - all the **major cloud providers support it, making it the logical choice for organizations looking to move more applications to the cloud**

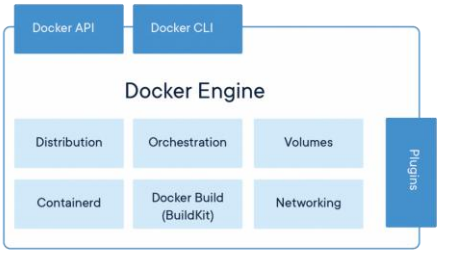


© Paulo Ferreira 11

11

Container Runtime

- **Docker Engine Sparked the Containerization Movement**
- **Docker Engine is the industry's de facto container runtime** that runs on various Linux (CentOS, Debian, Fedora, Oracle Linux, RHEL, SUSE, and Ubuntu) and Windows Server operating systems
- **Docker creates**:
  - **simple tooling**, and
  - a **universal packaging approach**, that
  - **bundles up all application dependencies inside a container which is then run on Docker Engine**
- **Docker Engine**:
  - **enables containerized applications to run anywhere consistently on any infrastructure**, solving "**dependency hell**" for developers and operations teams, and
  - eliminating the **"it works on my laptop!"** problem



© Paulo Ferreira 12

12

Application Development (1/2)

- **Before containers**:
  - **develop and install applications on each OS**
  - different OS imply different code and different installation steps

- **With containers**:
  - **just install the container**
  - it is an **isolated environment**
  - **packaged with all the needed configuration**
  - **same command independently of the specific OS**

© Paulo Ferreira

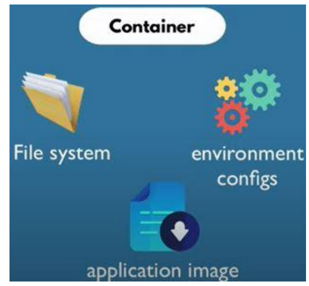
Application Deployment (2/2)

- **Before containers**:
  - the **development team** would give all the information to the **deployment team**
  - such information includes:
    - the **software**
    - **readme** files

- **With containers**:
  - **there is a single package** (no configuration needed)
  - the **deployment team just have to run a single command** to **get the container from the repository and install it**

© Paulo Ferreira

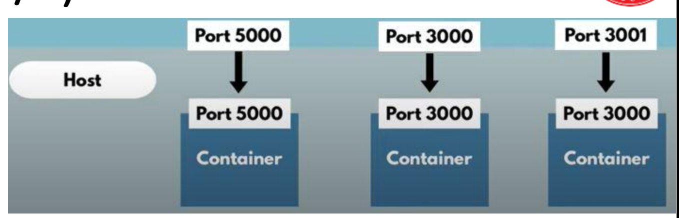
Images, Containers, and Layers



- An **image** is:
  - the **actual package** (e.g., containing the configuration)
- A **container** is:
  - an **image that is running**
  - so, it is a **running environment for the image**
- **Each container**:
  - **has an associated port (5000) that can be used to "talk" to the application running inside the container**
  - **contains a file system which is virtual**
- **Image vs container**:
  - an image is similar to a "executable" file in Unix
  - a container is similar to a "process" in Unix
- When you **download an image from some repository only the images that are needed are updated (**remember that **images are made of several layers)**

© Paulo Ferreira

Basic Docker Commands (1/2)



- The **basic commands** are:
  - pull, images, run, ps, start, stop, logs, exec
- **pull**:
  - **download an image from some repository**
- **images**:
  - **provides a list of the local images**
  - the **list shows the port which can be used to talk to the application**
- **run**:
  - **creates a container inside which an image executes**
  - **if the container does not exist locally**, this is equivalent to pull+start
  - option "-d" means **detached**
  - option "-p" **binds a local port** (see right-hand side of this slide)
  - option "--n" **gives a name to a container**

- Your computer has some **port available**
- There must be a **bound between a port in the local machine with the container port**:
  - then, **the port of the local computer can be used to talk to the application in the container**
  - the **biding** is done via the "run" command with the option"-p"

© Paulo Ferreira

Basic Docker Commands (2/2)

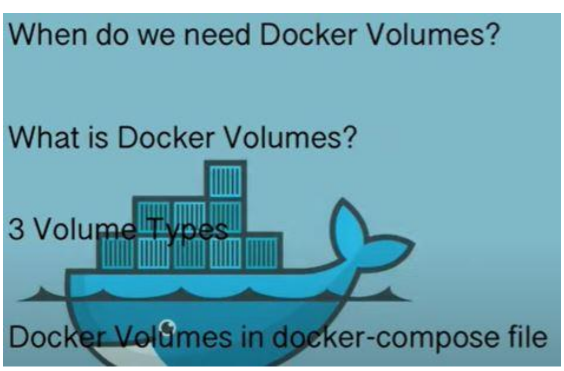
- **ps**:
  - provides the **status of each container that is running**
  - option "-a" **shows all existing containers** (running or not)
- **start**:
  - starts **running a container**
- **stop**:
  - **stops the running container** given its identifier (seen with ps)
- **logs**:
  - **see the log for a container** given its identifier
- **exec**:
  - get the **terminal associated with a running container**
  - option "-it" (meaning interactive terminal) given its identifier (or its name)
  - **allows the user to get into the container as a root user**

The **basic commands** are:
- pull, images, run,
  ps, start, stop, logs, exec

© Paulo Ferreira

18

Persisting Data in Docker with Volumes



- **A container contains a virtual file system**
- However, **when a container is re-started or removed**, the **data is lost** so that the container starts in a fresh state
- A **Volume in Docker supports persistence**
- A **Docker Volume of the local host is mounted into the virtual file system of a container**
- The idea is that **the data in the Docker Volume** is read into the virtual file system when a container is re-started

© Paulo Ferreira

38

Docker Volume Types

- There are **3 Docker Volume types**:
  - **host volume**; use "docker run" command use the option "-v"
  - e.g.: -v /home/mount/data:/var/lib/mysql/data (HOSTDIR:TARGETDIR)
  - it is up to you **to decide where on the host file system the reference is made**

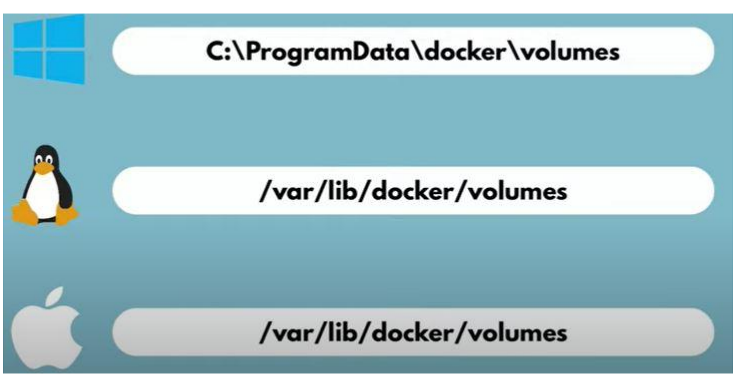
  - **anonymous volume**; use "docker run" command use the option "-v" but with a reference only to the virtual file system of the container
  - e.g.: -v :/var/lib/mysql/data (HOSTDIR:TARGETDIR)
  - so you **do not specify the mounting point in the host file system**; it is up to Docker to decide that (for each container a folder is generated in the host file that gets mounted)

  - **named volume**; use "docker run" command use the option "-v" but with the name of mounting folder on the host file system
  - so **you can reference the volume by name**

© Paulo Ferreira

39

Where are Volumes Stored ?



Docker for Mac
creates a Linux virtual machine
and stores all the Docker data here!

© Paulo Ferreira

41

Will containers eventually replace full-blown server virtualization? (1/2)

- That's **unlikely in the foreseeable future** for a number of important reasons
- **First**:
  - there is still a **widely held view that virtual machines offer better security than containers** because of the increased level of isolation that they provide
- **Second**:
  - the **management tools** that are available to orchestrate large numbers of containers are also not yet as comprehensive as software for managing virtualized infrastructure, such as VMware's vCenter or Microsoft's System Center
  - **companies that have made significant investments** in this type of software are unlikely to want to abandon their virtualized infrastructure without very good reason
- **Perhaps more importantly**:
  - virtualization and containers are also coming to be seen as **complementary technologies** rather than competing ones
  - that's because **containers can be run in lightweight virtual machines to increase isolation** and therefore security, and because **hardware virtualization makes it easier to manage the hardware infrastructure** (networks, servers and storage) that are needed to support containers

© Paulo Ferreira

Will containers eventually replace full-blown server virtualization? (2/2)

- **VMware encourages customers** who have invested in its virtual machine management infrastructure to run containers on its Photon OS container Linux distro inside lightweight virtual machines that can then be managed from vCenter:
  - this is VMware's "**container in a VM**" strategy
- But **VMware has also introduced what it calls vSphere Integrated Containers (VICs);** these containers can be deployed directly to a standalone ESXi host or deployed to vCenter Server as if they were virtual machines:
  - this is VMware's "**container as a VM**" strategy
- **Both approaches have their benefits**:
  - but what's important is that **rather than replacing virtual machines, it can often be useful to be able to use containers within a virtualized infrastructure**

© Paulo Ferreira

Conclusion

- What a **container** is
- **Containers** are **much faster than a VM**
- **Docker containers** are the most well known
- **Basic of containers in Docker** and some commands

- **Container orchestration** is the next step:
  - it **allows the management of containers** in a distributed setting
  - **kubernetes** is the most well known technology for this

© Paulo Ferreira