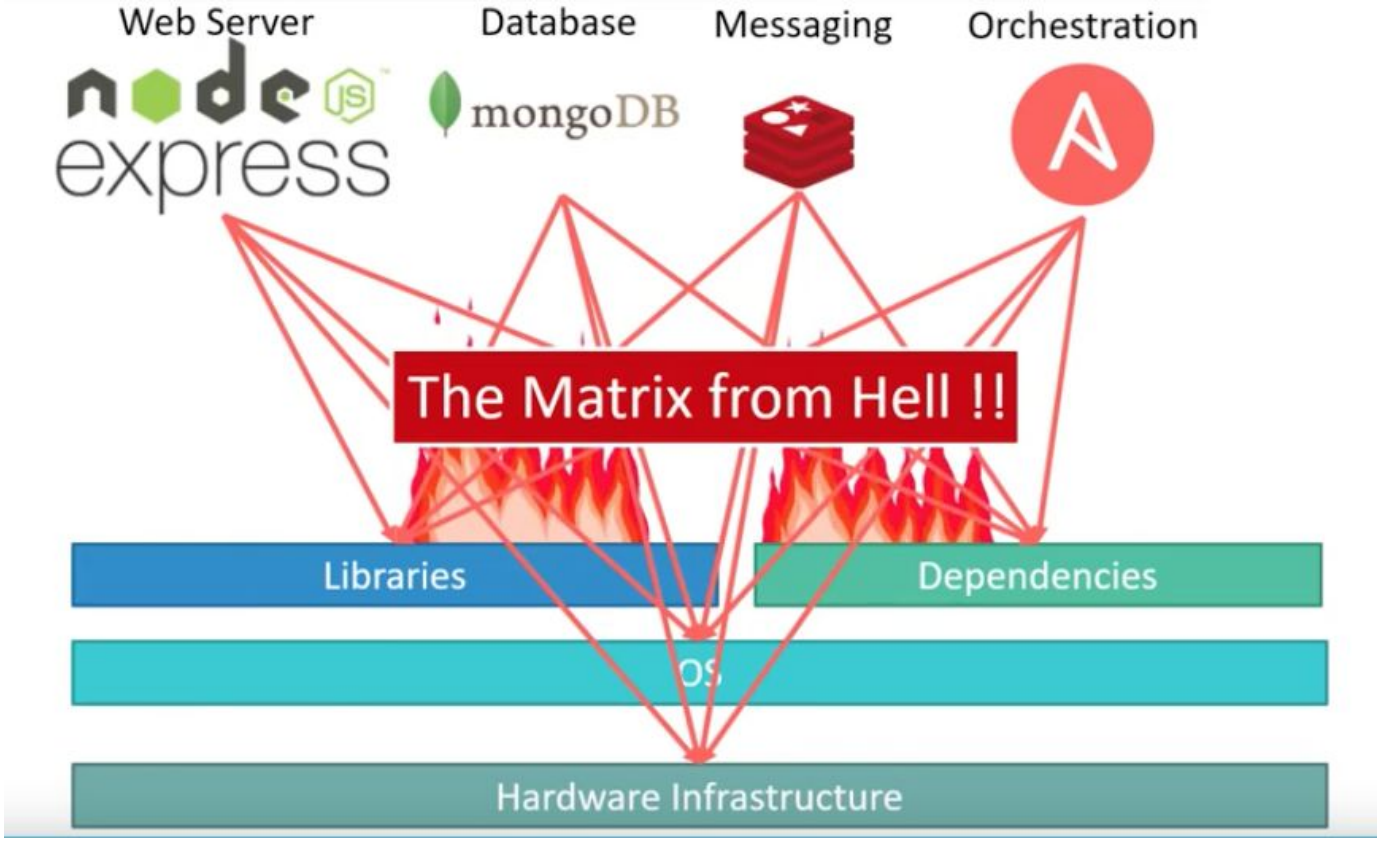
CC Lecture - Containers

- *Open Data Management & the Cloud*
- (Data Science & Scientific Computing / UniTS – DMG)

Dependency hell!

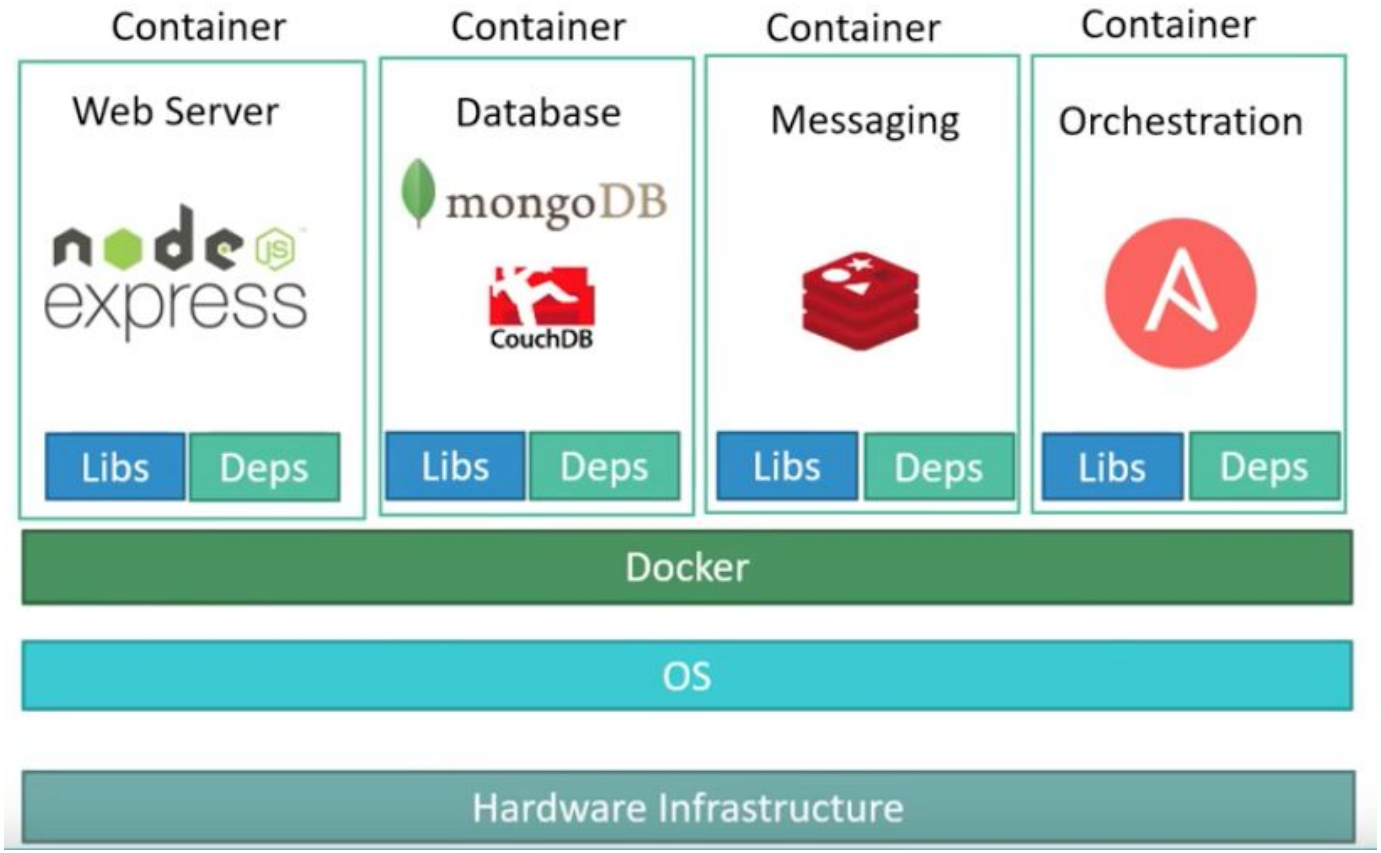


What are containers?

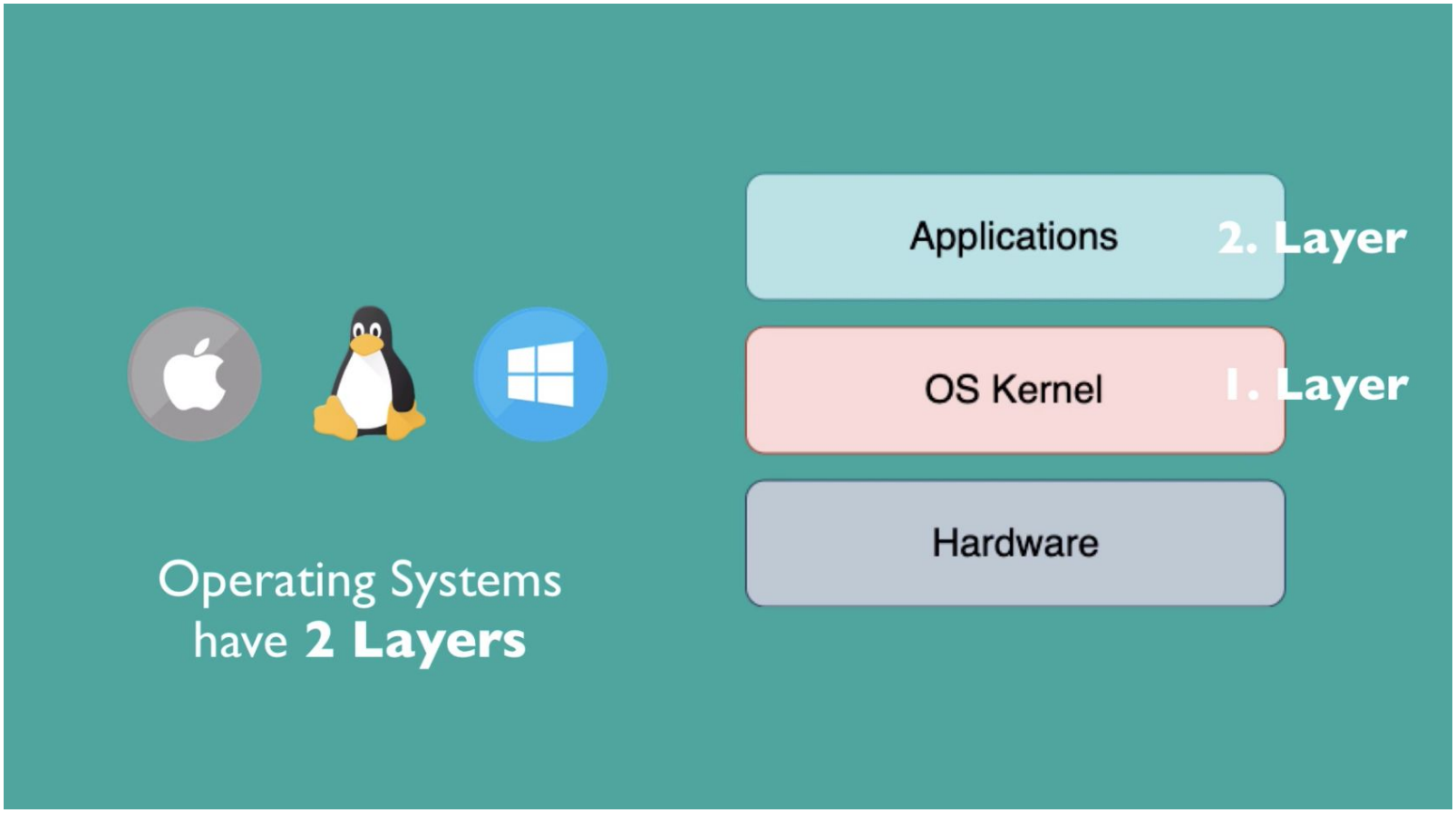
★ A container is a standard unit of software that packages up code and all its dependencies in processes isolated from resources so the application runs quickly and reliably from one computing environment to another.

★ Containers creates portable isolated environments at application level and not at server level.

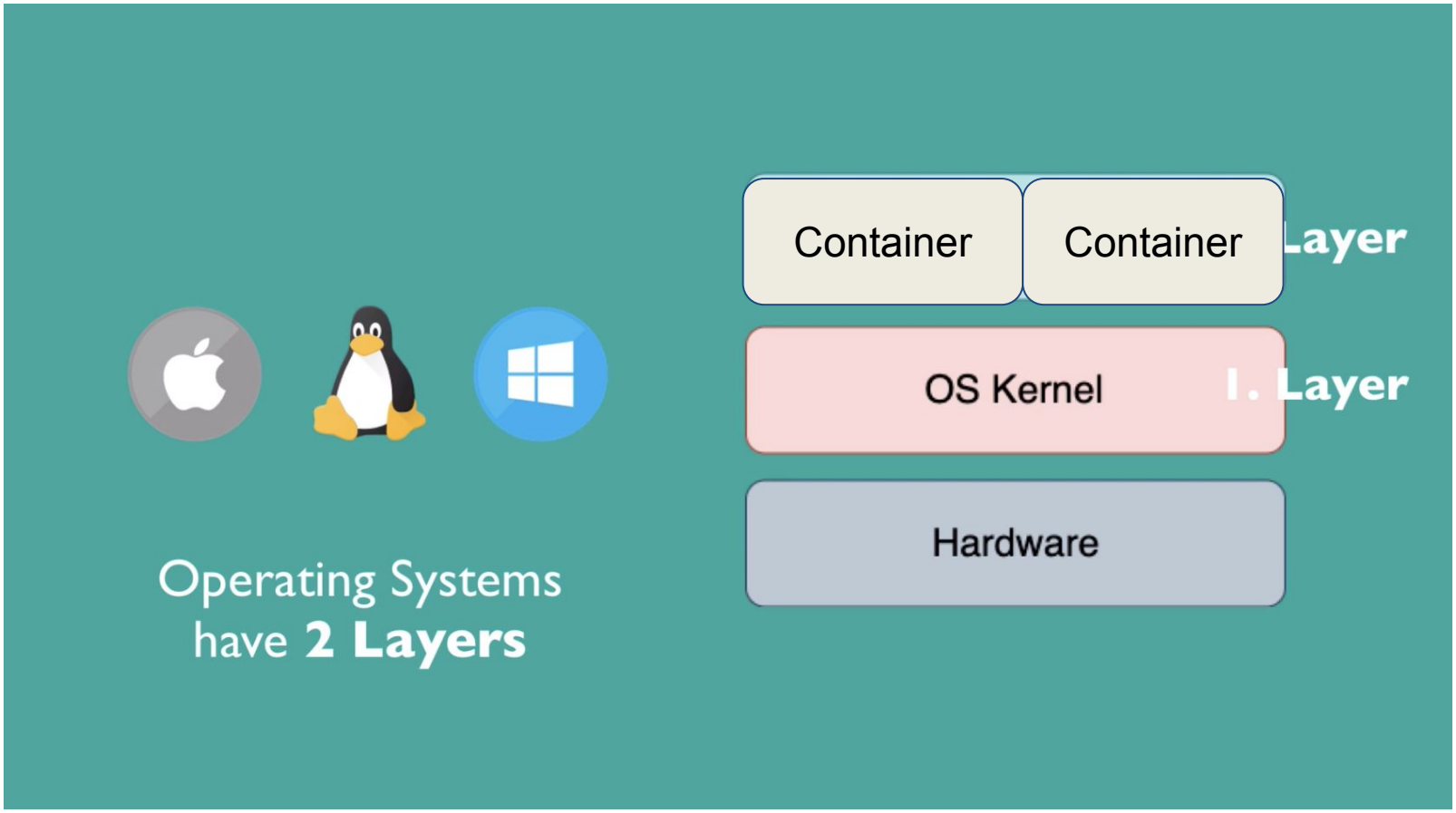
Dependency hell with containers



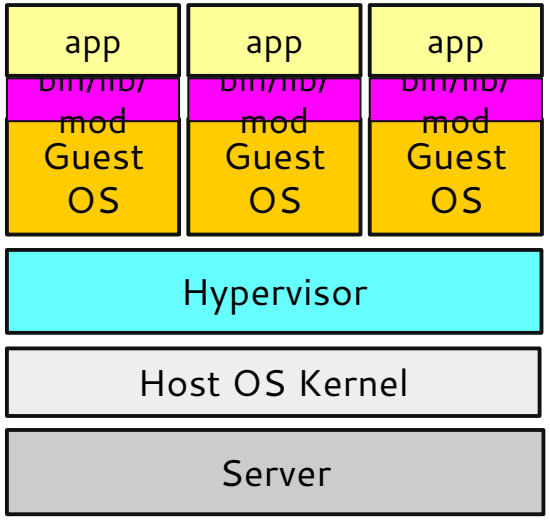
How does a container work?



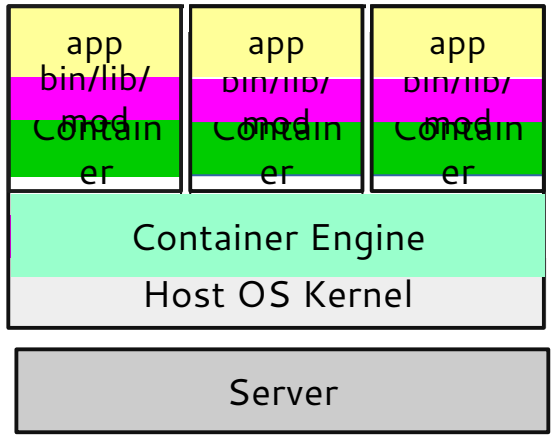
How does a container work?



Containers vs Virtual Machines



Virtual Machine



Container

★ Virtual Machines are able to run different Operative Systems

★ Container Engine: abstraction and isolation level between OS and applications environment. Enables and disables containers. Packages applications and their environment

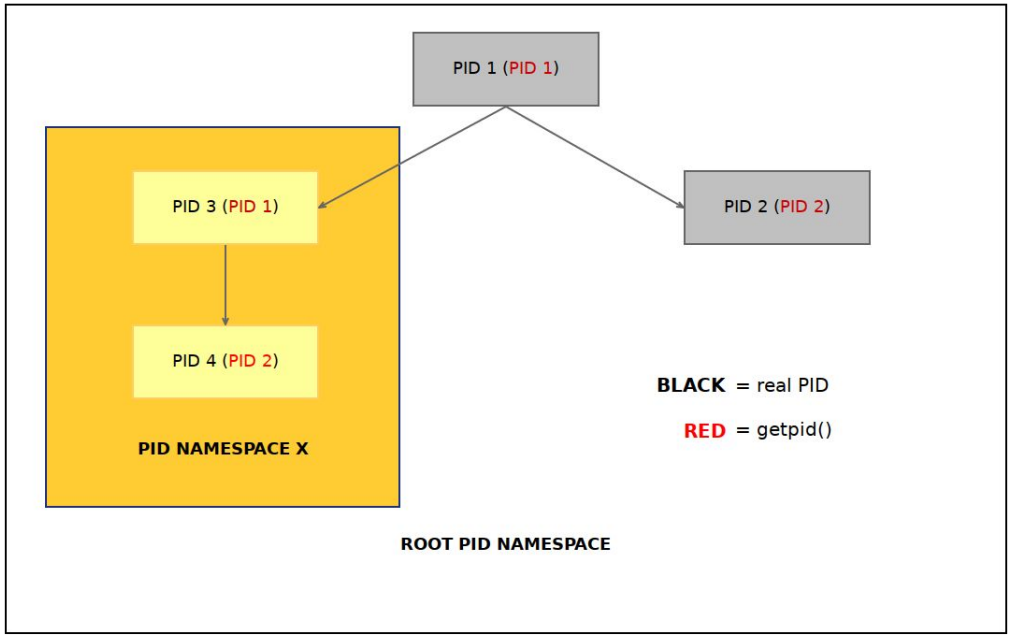
★ Containers virtualize at the OS level, with multiple containers running atop the OS kernel directly. Containers are far more lightweight: they share the OS kernel, start much faster, and use a fraction of the memory compared to booting an entire OS

Containers technology

★ Process Isolation and sandbox

★ **A namespace:** wraps a global system resource in an abstraction that makes it appear to the processes within the namespace that they have their own isolated instance of the global resource. Changes to the global resource are visible to other processes that are members of the namespace, but are invisible to other processes.

★ **Cgroups:** limit, police and account the resource usage for a set of processes.



Containers advantages (1)

★ **Isolation** Containers virtualize CPU, memory, storage, and network resources at the OS-level, providing developers with a sandboxed view of the OS logically isolated from other applications. Developers, using containers, are able to create predictable environments isolated from other applications.

★ **Productivity enhancement** Containers can include software dependencies needed by the application (specific versions of programming language runtimes, software libraries) guaranteed to be consistent no matter where the application is deployed. All this translates to productivity: developers and IT operations teams spend less time debugging and diagnosing differences in environments, and more time shipping new functionality for users.

Containers advantages (2)

★ **Deployment simplicity** containers allow your application as a whole to be packaged, abstracting away the operating system, the machine, and even the code itself, so development and deployment are easier because containers are able to run virtually anywhere (Linux, Windows, and Mac operating systems; virtual machines or bare metal; developer's machine or data centers on-premises; public cloud).

★ **Easy portability** Docker image format for containers further helps with portability. Docker V2 image manifest is a specification for container images that allows multi-architecture images and supports content-addressable images

Containers advantages (3)

★ **Operational efficiency and reliability** Containers are perfect for Service Oriented Architectures/Applications because each service limited to specific resources can be containerized. Separate services can be considered as black boxes.

- This arises efficiency because each container can be health checked and started/stopped when needed independently from others
- Reliability arises because separation and division of labor allows each service to continue running even if others are failing, keeping the application as a whole more reliable

Containers advantages (4)

★ **Easy Versioning** A new container can be packaged for each new application version including all needed dependencies, modules and libraries at the “right” version

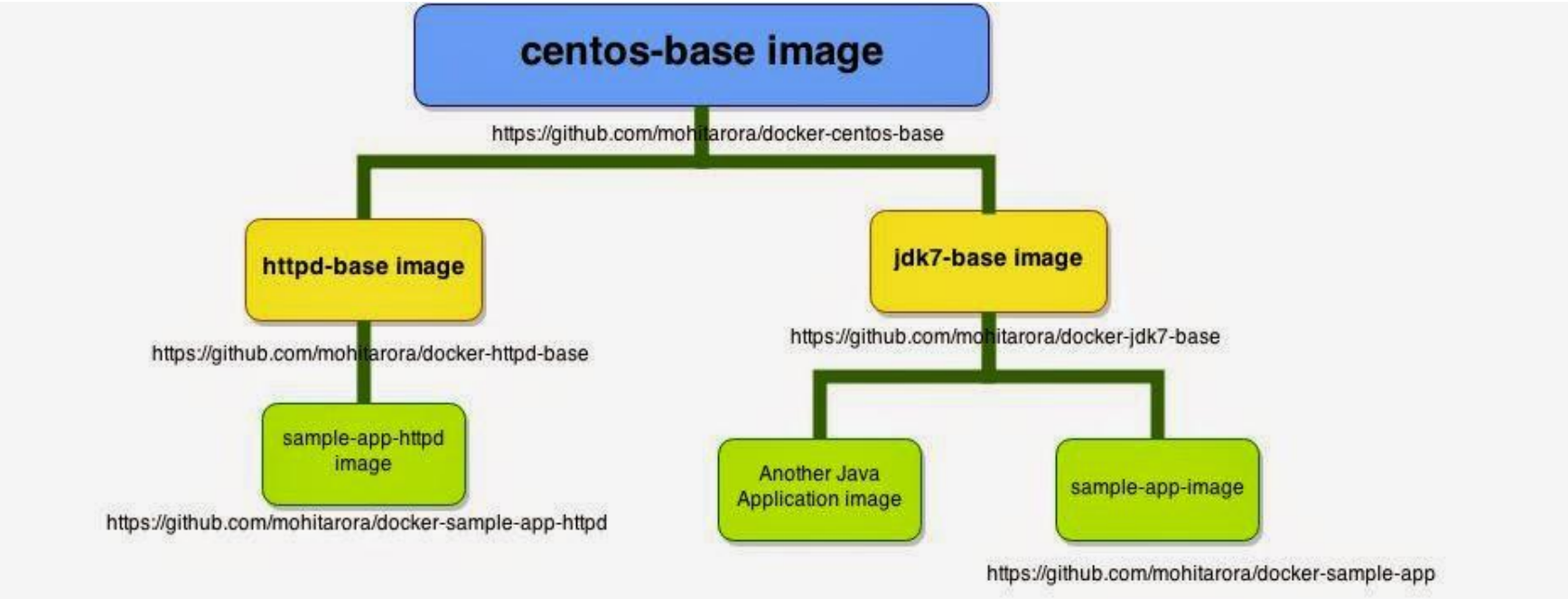
★ **Security** Containers add an additional layer of security since the applications aren't running directly on the host operating system. There are security constraint if application running inside containers have root privileges

What are container images?

★ Container images are lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.

★ Container images become containers at runtime

Container image hierarchy



Multiple software version issue (1)

★ Java example

- /usr/lib/jvm/java-1.7.0-openjdk
- /usr/lib/jvm/java-1.8.0-openjdk

=> /etc/alternatives/java_sdk -> /usr/lib/jvm/java-1.7.0-openjdk

★ Python example

- /usr/lib64/python2.7
- /usr/lib64/python3

=> virtualenv
=> conda and anaconda

Multiple software version issue (2)

$ python -V

Python 2.7.12 :: Continuum Analytics, Inc.

- If you install a packege SomePackage, it will install on python 2.7 path
- /usr/lib64/python2.7/site-packages/orca
- If later you will need to build a software with python3 and this software need an orca library, you will not be able to compile because orca library is installed for python2.7
- Moreover, this is not an easy problem to debug
- Other issue: unintentionally upgrade an application that shouldn't be upgraded breaking another application

One solution: Virtual Environments

★ Python “Virtual Environments” allow Python packages to be installed in an isolated location for a particular application, rather than being installed globally.

★ Virtual environments have their own installation directories and they don’t share libraries with other virtual environments.

The final solution: Containers

- Dockerfile to create a container packaging a specific python version and the application

Containers disadvantages

★ Difficult management in case of hosted containers high number

★OS sharing is prone to errors/instability

Container formats

★ There are many container formats available. Examples:

- Docker is a popular, open-source container format that is supported on Google Cloud Platform and by Google Kubernetes Engine.
- Singularity: Singularity containers can be used to package entire scientific workflows, software and libraries, and even data

==> Singularity software can import your Docker images without having Docker installed or being a superuser

Docker

- ★ Docker is a containerization platform that packages your application and all its dependencies together in the form of a docker container to ensure that your application works seamlessly in any environment.
- ★ Docker Container is a standardized unit which can be created on the fly to deploy a particular application or environment. It could be an Ubuntu container, CentOs container, etc. to full-fill the requirement from an operating system point of view.

How to use Docker

Verify your PC supports virtualization and specifically HyperV

Install from docker.com

Simple commands:

> docker version

> docker images

Run a simple “hello world!” program:

> docker run hello-world

Docker verifies if it is a local image with this name and if so, it runs it,

elsewhere it tries to automatically download it:

Docker image creation

```
[bertocco@firiel containers]$ docker run hello-world
Unable to find image 'hello-world:latest' locally
Trying to pull repository docker.io/library/hello-world ...
latest: Pulling from docker.io/library/hello-world
d1725b59e92d: Pull complete
Digest:
sha256:0add3ace90ecb4adbf7777e9aacf18357296e799f81cabc9fde47
0971e499788
Status: Downloaded newer image for docker.io/hello-world:latest
```

Image creation welcome message

Hello from Docker!
This message shows that your installation appears to be working correctly.
To generate this message, Docker took the following steps:
1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub. (amd64)
3. The Docker daemon created a new container from that image which runs the executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash
Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/
For more examples and ideas, visit:
https://docs.docker.com/get-started/

Docker simple commands

> docker run --help (i=interactive t=terminal)
> docker run -it ubuntu (run ubuntu, minimized version, also in windows shell)
Now we are in ubuntu shell. Try `ls` to check the filesystem
> docker run –it –name ubuntu my_ubuntu (gives a name)
> docker ps (shows available containers)
> docker attach my_ubuntu (to enter, use also the container ID)
> exit (exits and closes, verify `docker ps`)
> docker images (shows all downloaded images)
> docker rmi (-f) hello-world

How to create your image

https://github.com/HermantNET/hello-system

Branch: **master** ▾ **hello-system** / **index.js**

 **HermantNET** added server

1 contributor

13 lines (9 sloc) | 301 Bytes Raw

```
1   var http = require('http');
2
3   const PORT = 80;
4
5   function requestHandler(req, res) {
6           res.end(`Hello ${process.platform}`);
7   }
8
9   var server = http.createServer(requestHandler);
10
11  server.listen(PORT, function(){
12          console.log(`${process.env.NODE_ENV} server listening on port: ${PORT}. CTRL-C to exit.`);
13  });
```

Dockerfile

> git clone https://github.com/HermantNET/hello-system
Navigate to docker hub: https://hub.docker.com
Look for the Node package (we are working on a Node application)
“How to use this image” => “Create your dockerfile”
> cat .Dockerfile (the name is important)

```
# specify the node base image with your desired version node:<version>
FROM node:4-onbuild
# replace this with your application's default port
EXPOSE 80
```

Run your image (1)

You can then build and run the Docker image:

> docker build t hello-system .

> docker run -it --rm --name my_running-app hello-system

Bind a local port:

docker run -it p <localport>:<containerport> <name> <imagename>

> docker run -it -p 3377:80 –name hi –hello-system

Check: localhost:3377

Run your image (2)

If you prefer Docker Compose:

```
version: "2"
services:
 node:
  image: "node:8"
  user: "node"
  working_dir: /home/node/app
  environment:
   - NODE_ENV=production
  volumes:
   - ./:/home/node/app
  expose:
   - "80"
  command: "npm start"
```

You can then run using Docker Compose:

```
$ docker-compose up -d
```

Docker Compose example copies your current directory (including node_modules) to the container. It assumes that your application has a file named package.json defining start script.

Dockerfiles

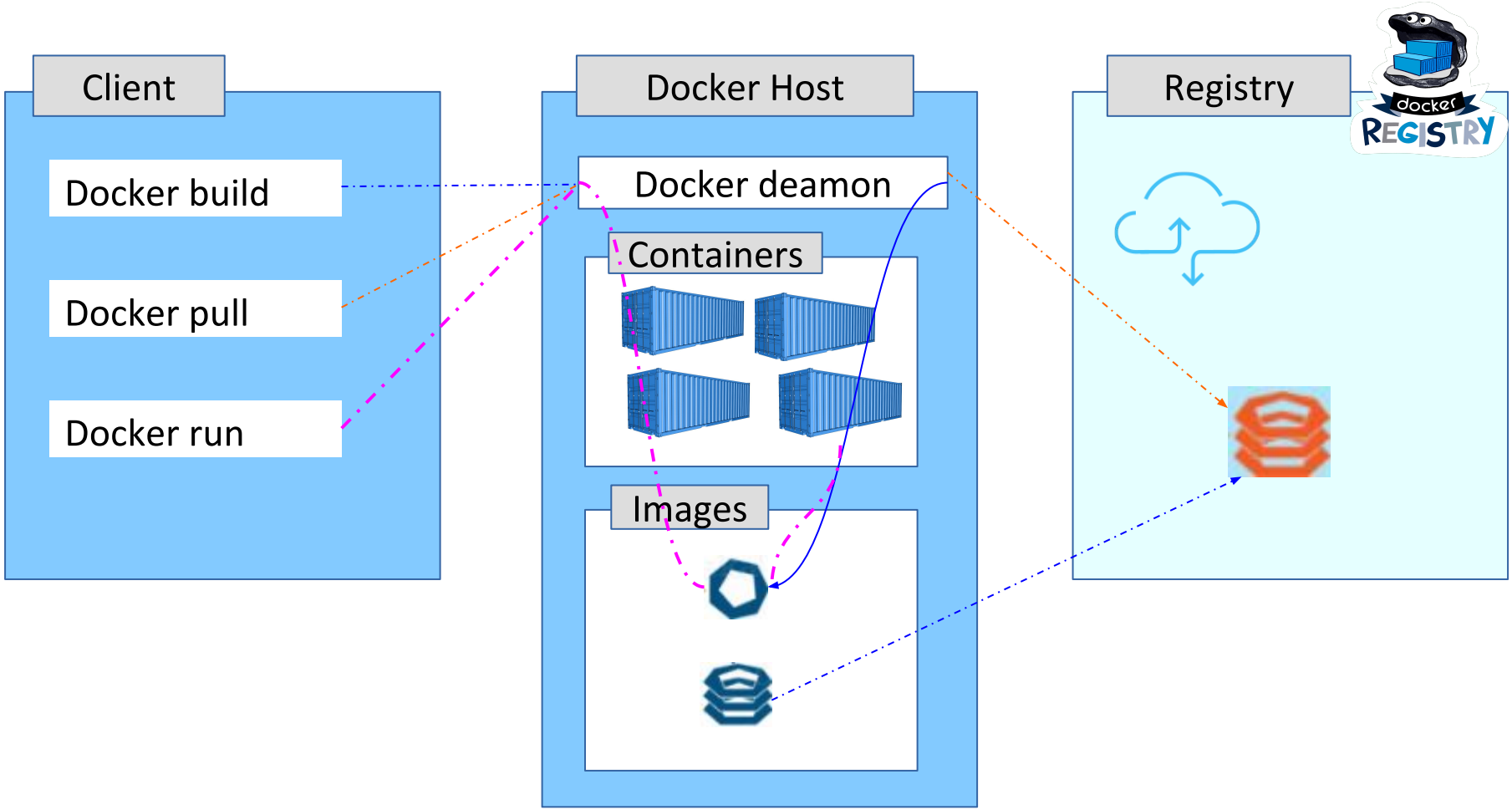
```
anouk:CONTAINER morgan$ more esoclimi/Dockerfile
FROM centos
RUN yum install -y  vim wget epel-release
RUN yum install -y gsl gsl-devel gcc gcc-gfortran make autoconf patch automake
RUN yum install -y mpich-3.2 mpich-3.2-devel mpich-3.2-autoload environment-mod
les
RUN yum install -y python-devel numpy python-matplotlib python2-pip
RUN pip install --upgrade pip
ENV MPI_INCLUDE=/usr/include/mpich-3.2-x86_64
ENV MPI_PYTHON_SITEARCH=/usr/lib64/python2.7/site-packages/mpich-3.2
ENV MPI_LIB=/usr/lib64/mpich-3.2/lib
ENV MPI_BIN=/usr/lib64/mpich-3.2/bin
ENV MPI_COMPILER=mpich-3.2-x86_64
ENV MPI_SYSCONFIG=/etc/mpich-3.2-x86_64
ENV MPI_SUFFIX=_mpich-3.2
ENV MPI_MAN=/usr/share/man/mpich-3.2
ENV MPI_HOME=/usr/lib64/mpich-3.2
ENV MPI_FORTRAN_MOD_DIR=/usr/lib64/gfortran/modules/mpich-3.2-x86_64
ENV PATH="/usr/lib64/mpich-3.2/bin:${PATH}"
RUN pip install mpi4py
RUN pip install astropy
RUN pip install ipython
WORKDIR /home/mpi4py
```

Docker Registry & Docker Hub

★ Docker Registry is where the Docker Images are stored. The Registry can be either a user's local repository or a public repository like a Docker Hub allowing multiple users to collaborate in building an application. Even with multiple teams within the same organization can exchange or share containers by uploading them to the Docker Hub.

★ Docker Hub is Docker's very own cloud repository similar to GitHub.

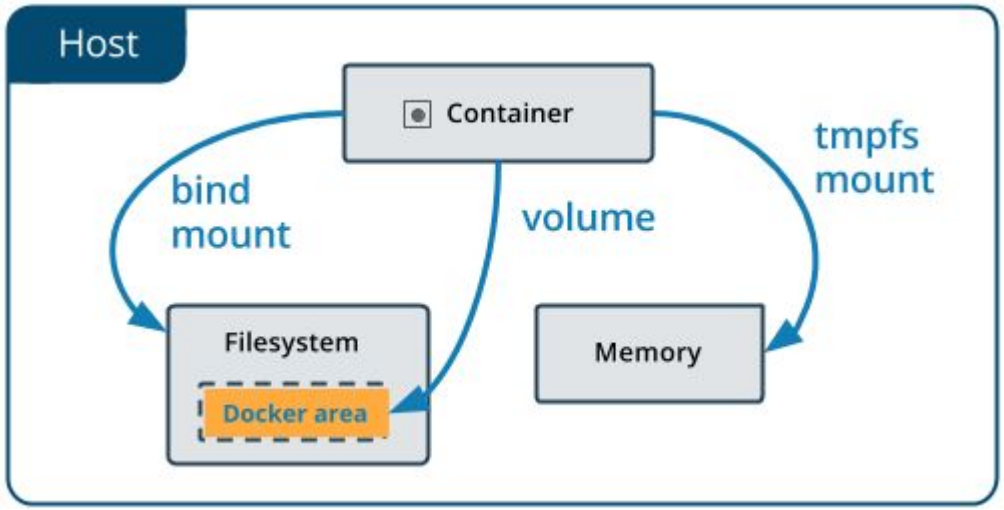
Docker Architecture



Docker volumes

★ Volumes are the preferred mechanism for persisting data generated by and used by Docker containers.

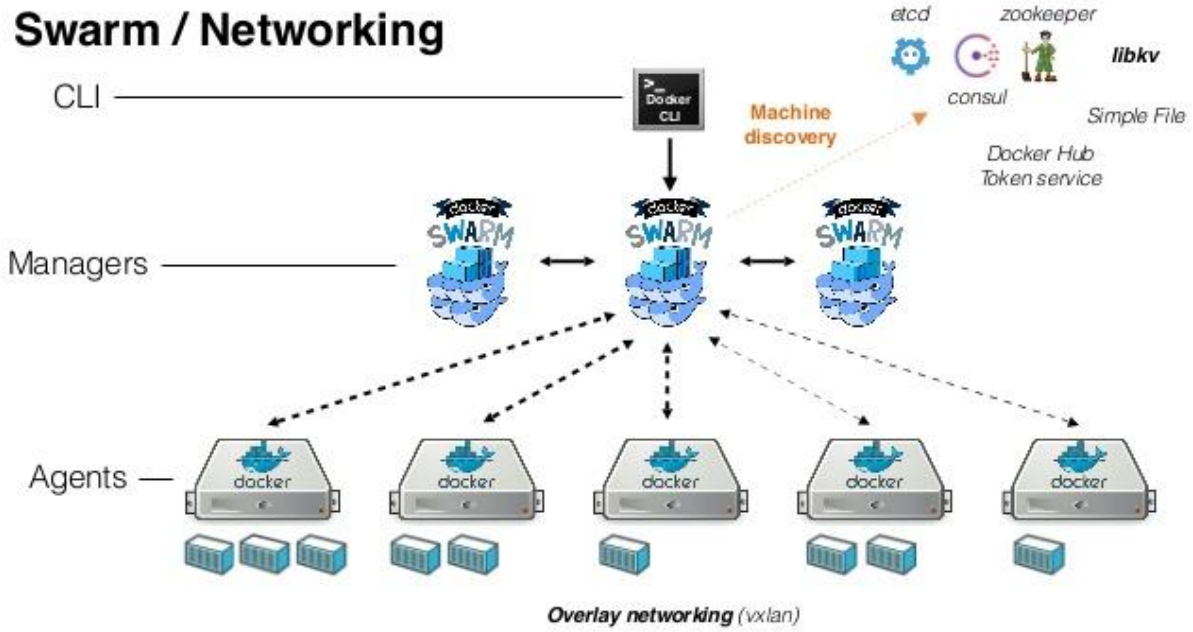
★ Volumes are completely managed by Docker.




```
docker run -d --name devtest  -v myvol2:/app   nginx:latest
```

Docker swarm

★ The cluster management and orchestration features embedded in the Docker Engine are built using swarmkit




```
$ docker swarm init
```

Docker scaling

★ Docker specialization is massive horizontal scaling each environment can be composed from many different containers and each container can be a database, a JS installation, if you need more resources, Docker responds with instances of what you need

Kubernates

Kubernetes, or k8s (k, 8 characters, s), or “kube” is an open source platform that automates Linux container operations.

Groups of hosts running Linux containers can be clustered spanning hosts across public, private, or hybrid clouds and Kubernetes helps to easily and efficiently manage those clusters.

Kubernates features

Kubernates is a platform to schedule and run containers on clusters of physical or virtual machines

- ★ Orchestrate containers across multiple hosts.
- ★ Make better use of hardware to maximize resources needed to run the enterprise apps.
- ★ Control and automate application deployments and updates.
- ★ Mount and add storage to run stateful apps.
- ★ Scale containerized applications and their resources on the fly.
- ★ Declaratively manage services, which guarantees the deployed applications are always running how you deployed them.
- ★ Health-check and self-heal your apps with autoplacement, autorestart, autoreplication, and autoscaling.

Kubernates in production