

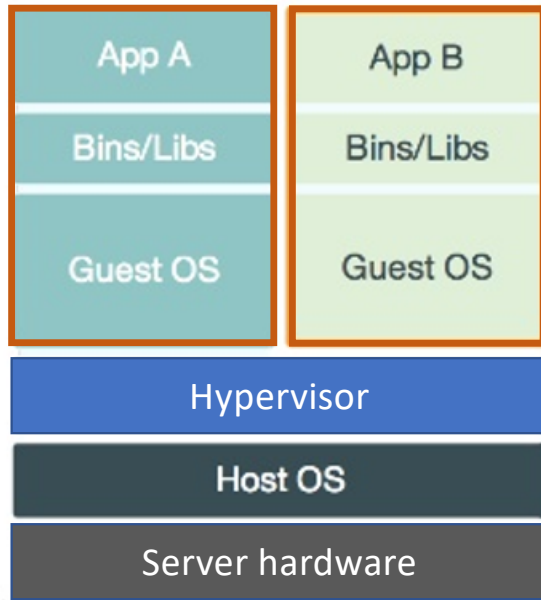
Containers

Antonio Brogi

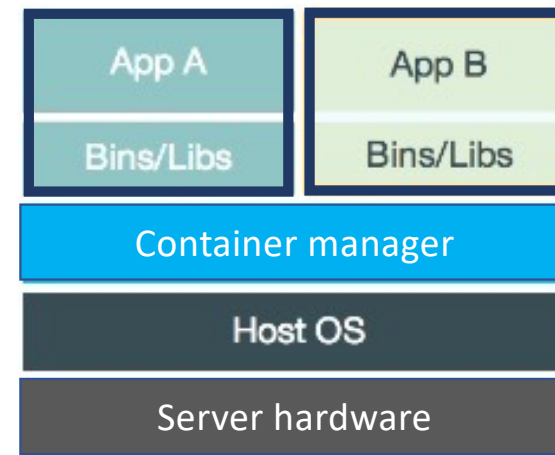
Department of Computer Science
University of Pisa

Virtualization and containers

From VMs to containers



- Server virtualization
- Many virtual machines (each with its OS) running on same physical server
- ("Type 1" hypervisor loaded directly on HW)



Key idea of containers: Exploit OS kernel's capability of allowing multiple isolated user-space instances

- + lighter (require less resources, from Gb to Mb)
- + faster to start (from mins to secs)
- share same OS

How old are containers?

- For decades, UNIX *chroot* command provided a simple form of filesystem isolation
- 1998 – FreeBSD *jail* utility extended *chroot* sandboxing to processes
- 2005 – Google started developing *CGroups* for Linux kernel and began moving its infrastructure to containers
- 2008 – Linux Containers (LXC) provided a complete containerization solution

2013 - Docker added the missing pieces - *portable images* and *friendly UI* – to the containerization puzzle, and containers entered the mainstream

The Docker platform consisted of:

- **Docker Engine** (for creating and running containers)
- **Docker Hub** (for distributing containers)

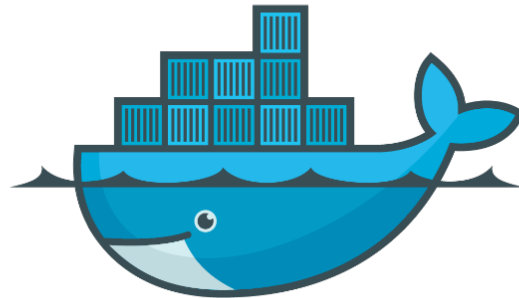
Virtualization and containers

Docker

What is Docker?

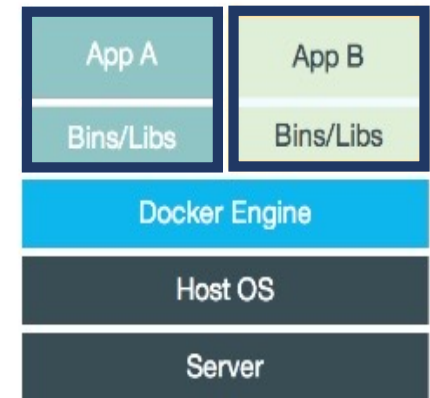
Docker is a platform that allows us
to run applications in an isolated environment

Docker allows us to develop and run portable applications
by exploiting *containers*



Docker

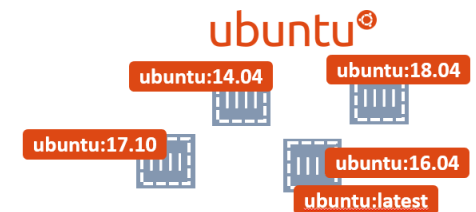
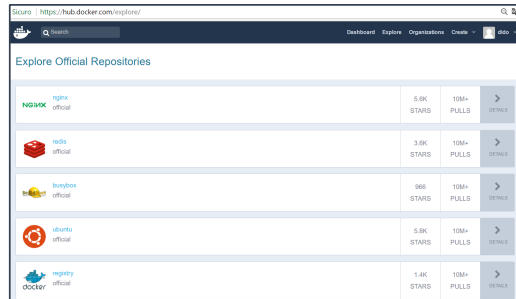
- Docker exploits container-based virtualization to run multiple isolated guest instances on the (same) OS
- Software components are packaged into ***images***, which are exploited as read-only templates to create and run ***containers***
- External ***volumes*** can be mounted to ensure data persistence



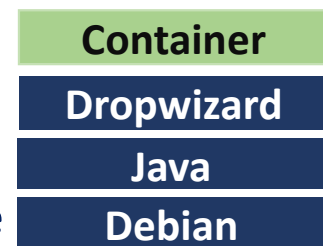
“Build, ship, and run any app, anywhere”

Docker images

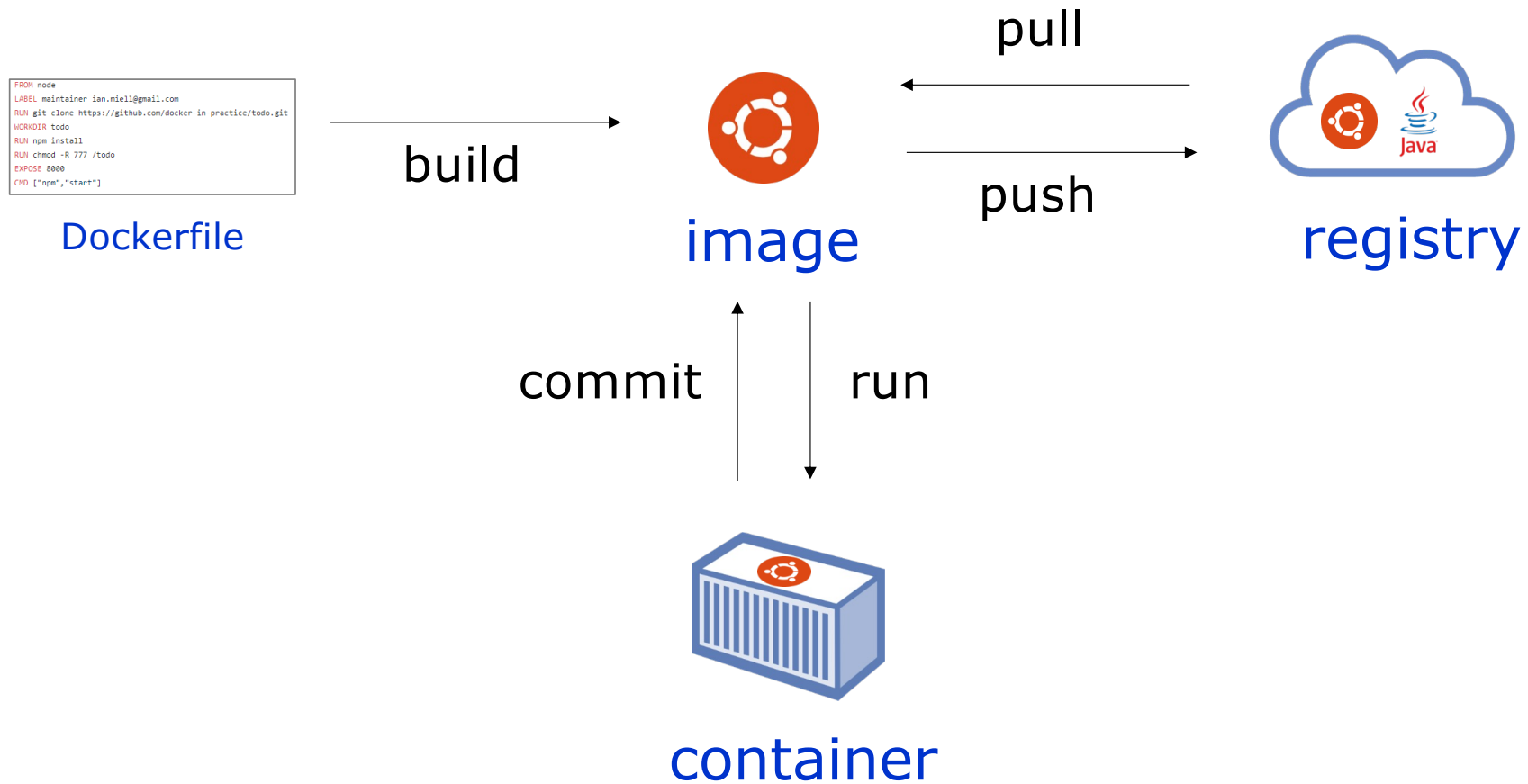
- (read-only) templates used to create containers
- stored in a (private or public) Docker **registry**
 - registry structured in repositories
 - each repository contains a set of images, for different versions of a software
 - images identified by pairs **repository:tag**
 - [**Docker Hub**](https://hub.docker.com/)



- images structured into (read-only) layers
- each layer is in turn an image, lowest layer called base image
- running container can write in new top layer, changes can be committed into new image



Docker commands



Learn Docker in 12 min



<https://www.youtube.com/watch?v=YFI2mCHdv24>



Advantages of Docker

- same environment
- sandbox projects
- it just works

Containers

- lighter and faster to start than VMs
- share OS
- images defined via Dockerfile
- build and run

Demo

install Docker

simple php "hello world" app

index.php

```
1 <?php
2
3 echo "Hello, World"
```

Dockerfile

find php image from Docker Hub

```
1 FROM php:7.0-apache
2 COPY src/ /var/www/html
3 EXPOSE 80
```

>docker build -t hello-world .

>docker run -p 80:80 hello-world

using a volume to share folder between host and container
one process per container

Virtualization and containers

Docker

Docker Compose

Docker Compose in 12 min



<https://www.youtube.com/watch?v=Qw9zIE3t8Ko>



//product service

api.py

```
3 from flask import Flask
4 from flask_restful import Resource, Api
5
6 app = Flask(__name__)
7 api = Api(app)
8
9 class Product(Resource):
10     def get(self):
11         return {
12             'products': ['Ice cream',
13                         'Chocolate',
14                         'Fruit',
15                         'Eggs']
16         }
17
18 api.add_resource(Product, '/')
19
20 if __name__ == '__main__':
21     app.run(host='0.0.0.0', port=80, debug=True)
```

requirements.txt

```
1 Flask==0.12
2 flask-restful==0.3.5
```

Dockerfile

```
1 FROM python:3-onbuild
2 COPY . /usr/src/app
3 CMD ["python", "api.py"]
```

docker-compose.yml

```
1 version: '3'
2
3 services:
4     product-service:
5         build: ./product
6         volumes:
7             - ./product:/usr/src/app
8         ports:
9             - 5001:80
10
11     website:
12         image: php:apache
13         volumes:
14             - ./website:/var/www/html
15         ports:
16             - 5000:80
17         depends_on:
18             - product-service
```

>docker-compose up
>docker ps
>docker-compose stop

//website

index.php

```
1 <html>
2     <head>
3         <title>My Shop</title>
4     </head>
5
6     <body>
7         <h1>Welcome to my shop</h1>
8         <ul>
9             <?php
10                 $json = file_get_contents('http://product-service');
11                 $obj = json_decode($json);
12
13                 $products = $obj->products;
14                 foreach ($products as $product) {
15                     echo "<li>$product</li>";
16                 }
17             <?php
18             </ul>
19         </body>
20 </html>
```

Virtualization and containers

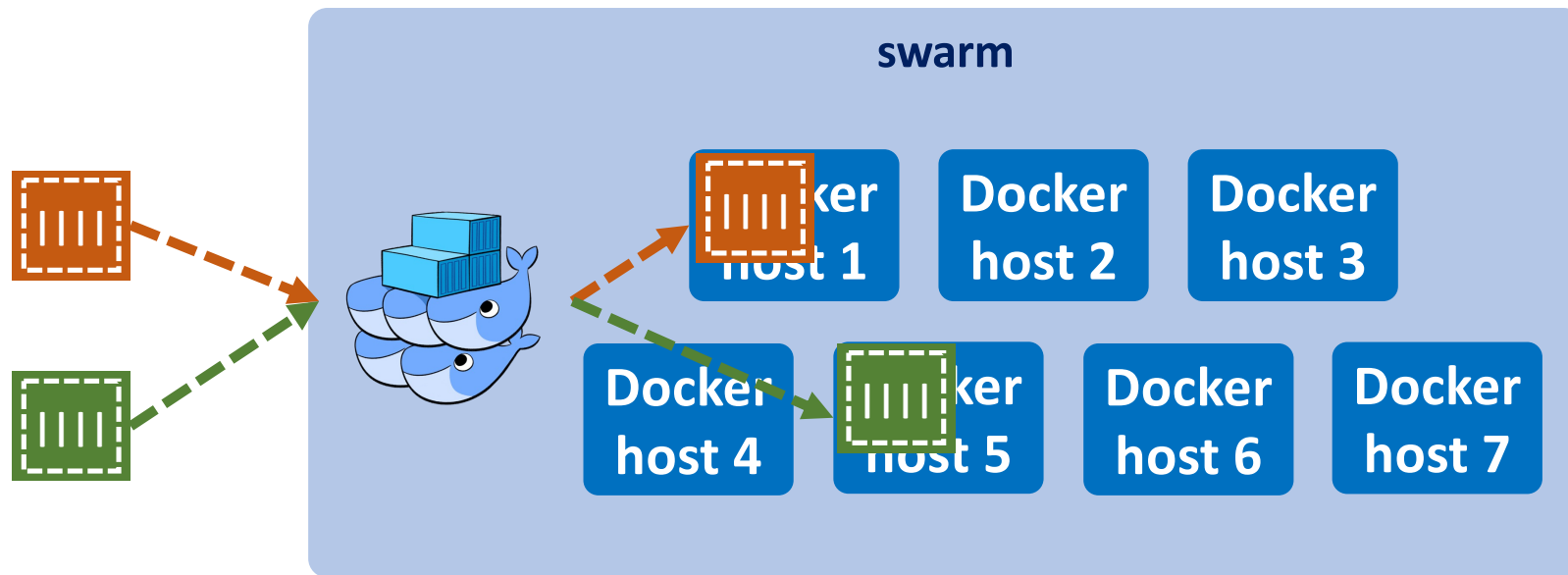
Docker

Docker Compose

Swarm mode

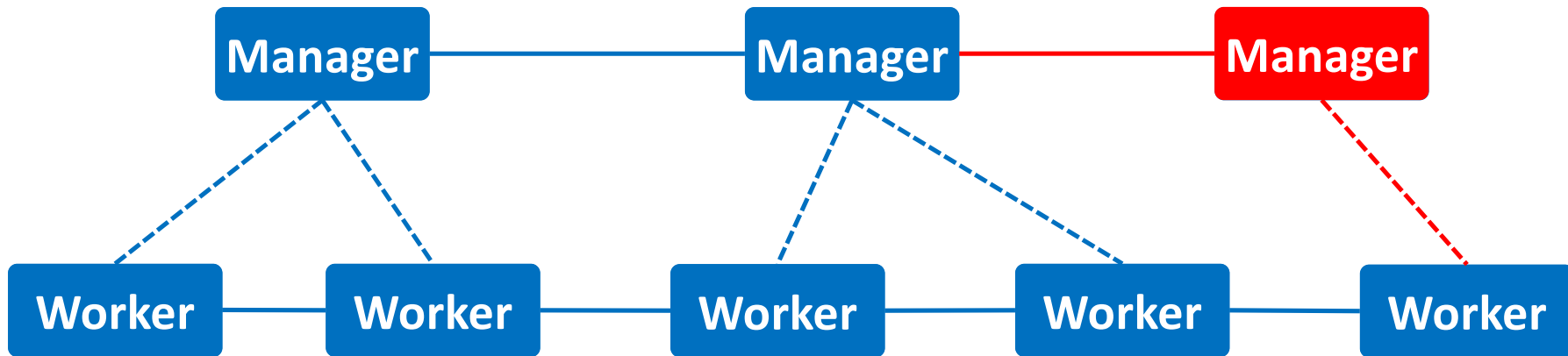
Swarm mode

Docker includes ***swarm mode*** for managing a cluster of Docker hosts called a *swarm*

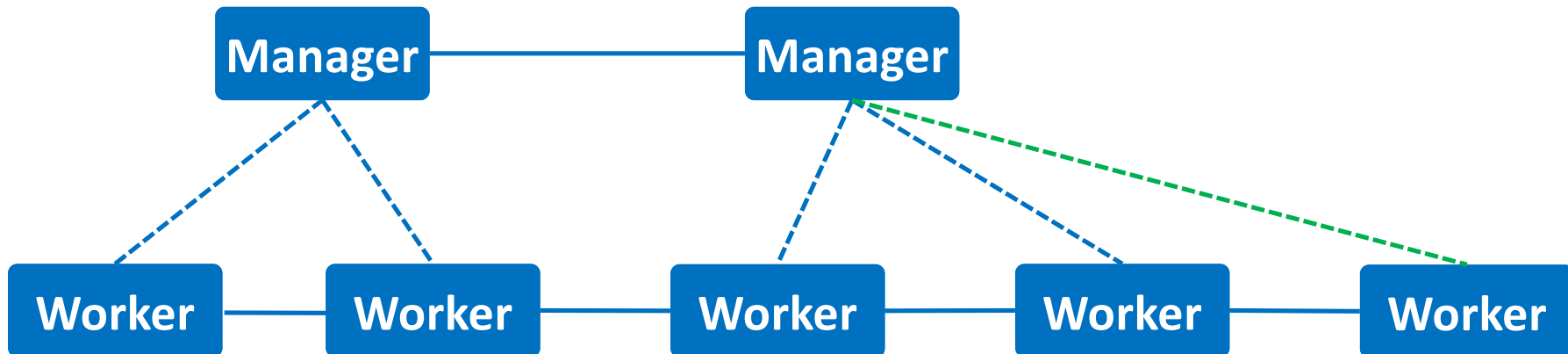


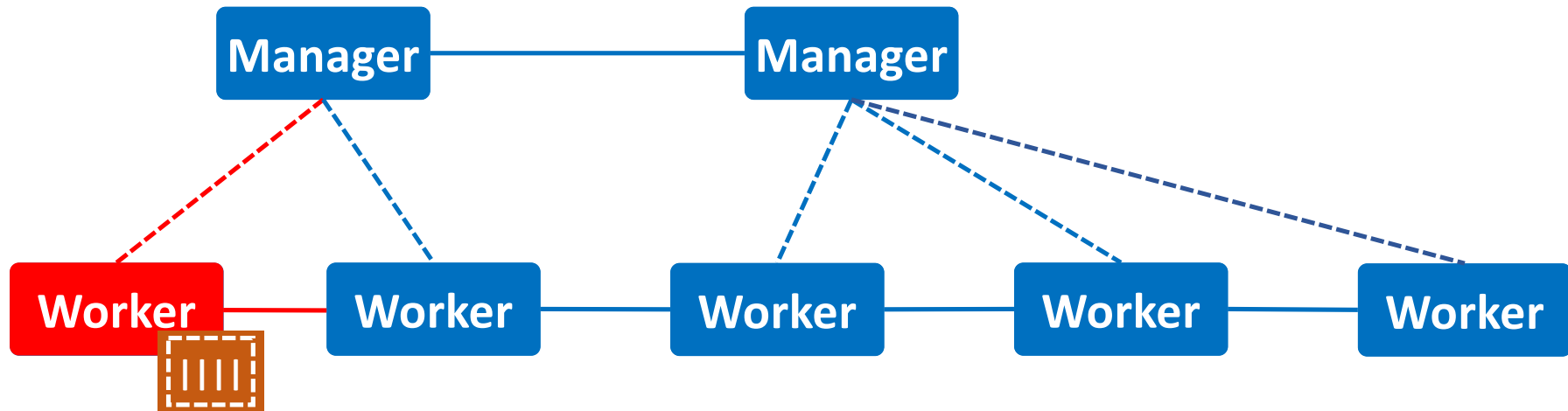
Swarm mode

- Swarm nodes can act as managers (delegating tasks workers) and workers (executing the tasks assigned to them)
- You can define the desired state of the various services in your application stack, including the number of tasks to run in each service
- Swarm manager nodes assign each service in the swarm a unique DNS name and load balances running containers
- Swarm manager nodes constantly monitor the cluster state and reconcile any differences between the actual state and your expressed desired state
 - For example, if you set up a service to run 10 replicas of a container, and a worker machine hosting two of those replicas crashes, two new replicas are created and assigned to workers that are running and available



Manager fails → manager's workers are assigned to another manager





Worker fails → worker's containers are assigned to another worker

