Dynamic platform, where are our µservices ?

#docker #dns #scalability...

Wifi : hola#4321

IPPON DEVOXX France

Florian GARCIA

@Im_flog

@ippontech

#java #spring
#microservices
#docker #kafka

Jeremie MONSINJON

@jmonsinjon

@ippontech

#java
#microservices
#elastic #docker

Christophe LABOUISSE

@XtlCnslt

#java #spring
#microservices
#docker #aws

Plan

1. Microservices? Discovery? Why?
2. Java Discovery
3. Docker basics
4. Docker network
5. Docker Compose
6. Docker Swarm
7. Bonus: Docker Stack

Microservices

Success keys

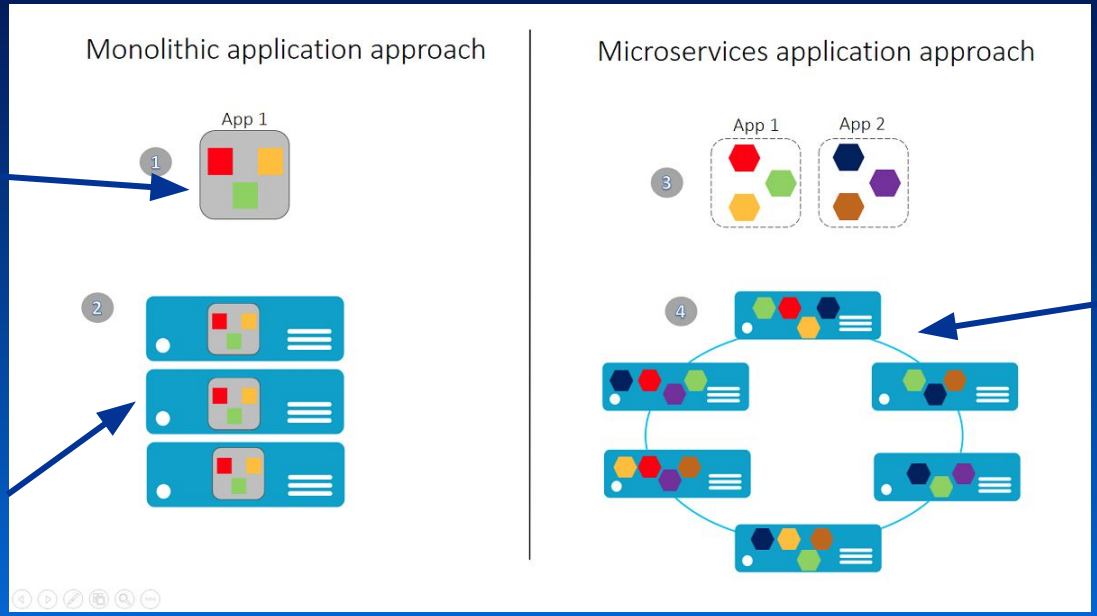
- Suite of small services
- Running in its own process
- Communicating with lightweight mechanisms
- Built around business capabilities
- Independently deployable

James Lewis and Martin Fowler

Complexity appears

Internal libraries calls

No internal network to manage



HTTP calls

Dynamic platform

- We need Scalability !
- We don’t care about the infra (cloud)
- We don’t care where are our applications
- We just want to know how many are “alive”
- But the applications need to know each other’s location !

The disco(very)

- Use of a “register” to store address and port for each microservice
- Client side register
- Common implementations:
  - Query the registry before each call
  - DNS for each service
- Central concept in a microservices architecture

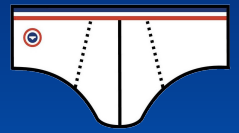
Disco(very) in Java

- Shared Key-Value Store
  - Zookeeper, Consul, etcd ...
- Custom mechanisms
  - Netflix Eureka, Serf ...
- DNS management
  - Spotify Apollo
  - Kong

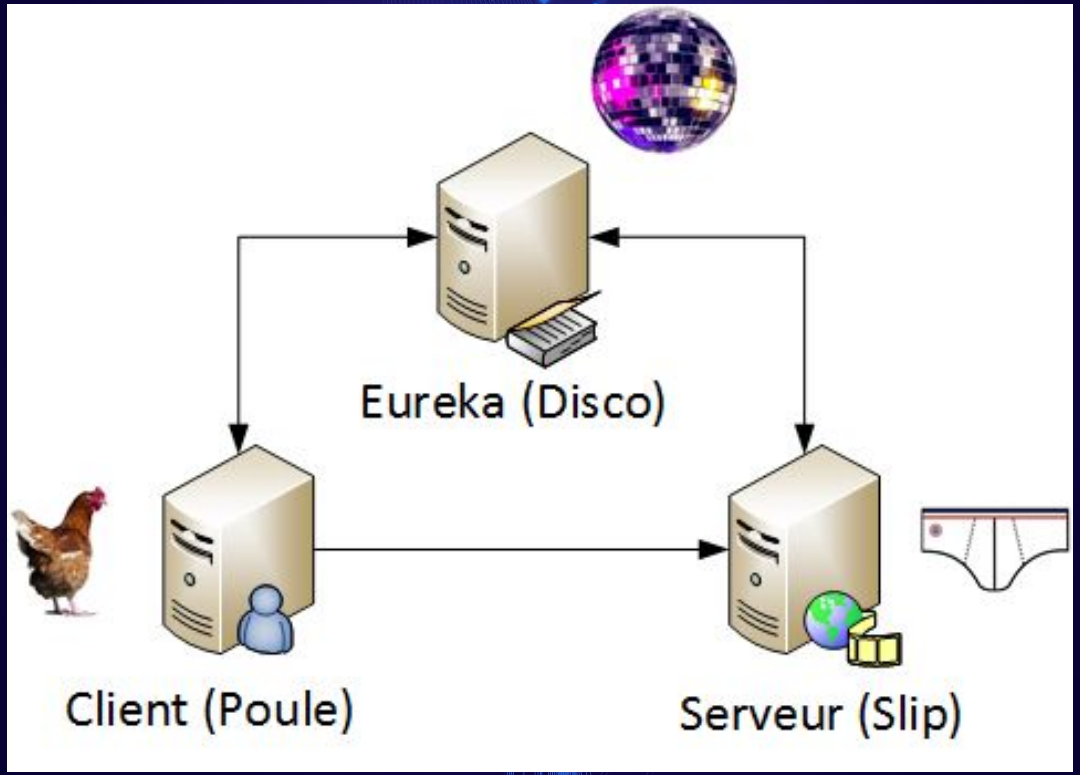
We will use Eureka, because it is very easy !

Context

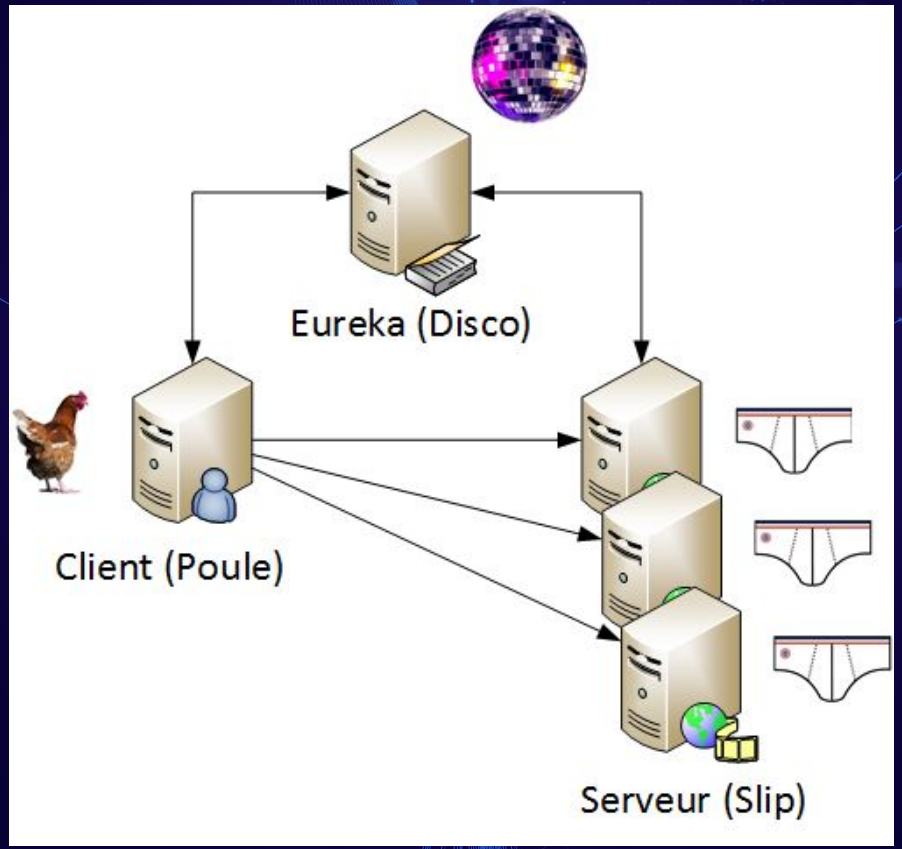
- Java / SpringBoot
- 3 services
  - **Slip**: listen http requests and for each one, wait 100ms before answering $HOSTNAME
  - **Poule**: Parallelize 10 calls to slip permanently
  - **Disco**: Manage the services discovery with Eureka



The app

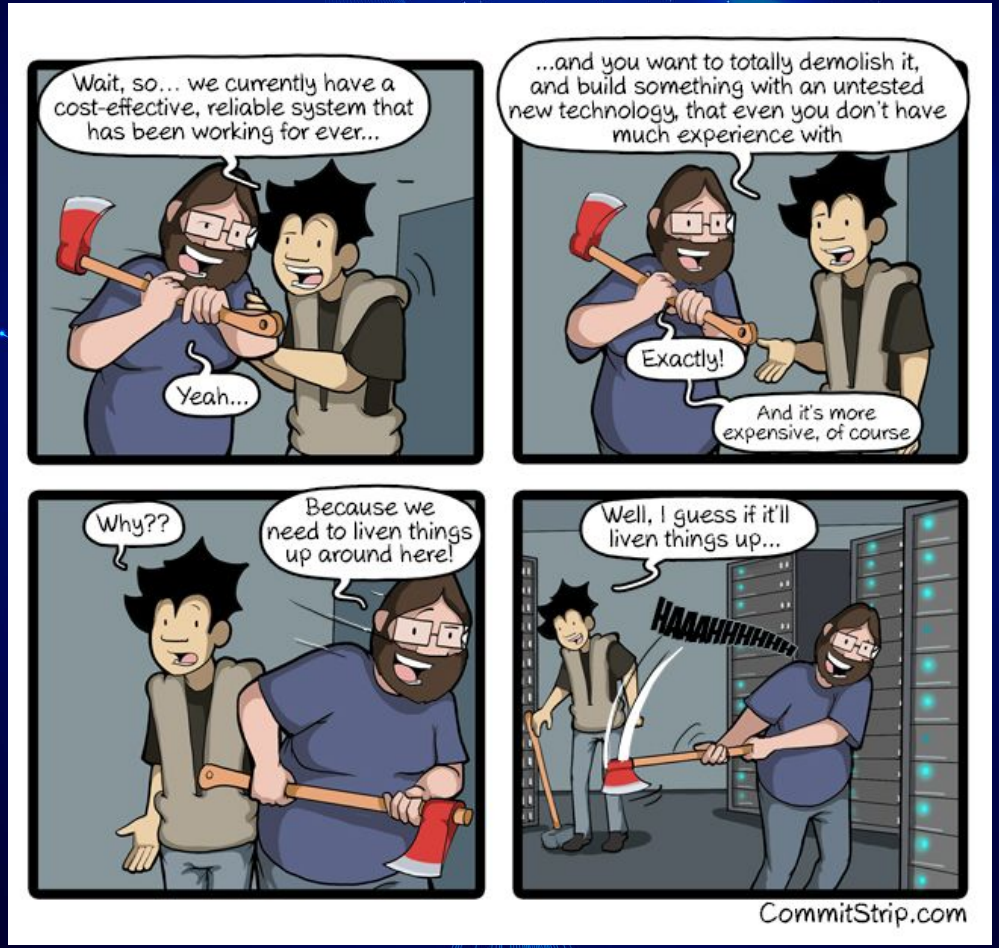


Scaling




Java discovery
demo

So you heard Docker was great?


Wait, so... we currently have a cost-effective, reliable system that has been working for ever...
Yeah...
...and you want to totally demolish it, and build something with an untested new technology, that even you don't have much experience with
Exactly!
And it's more expensive, of course
Why??
Because we need to liven things up around here!
Well, I guess if it'll liven things up...
HAAAHHHHHHH
CommitStrip.com

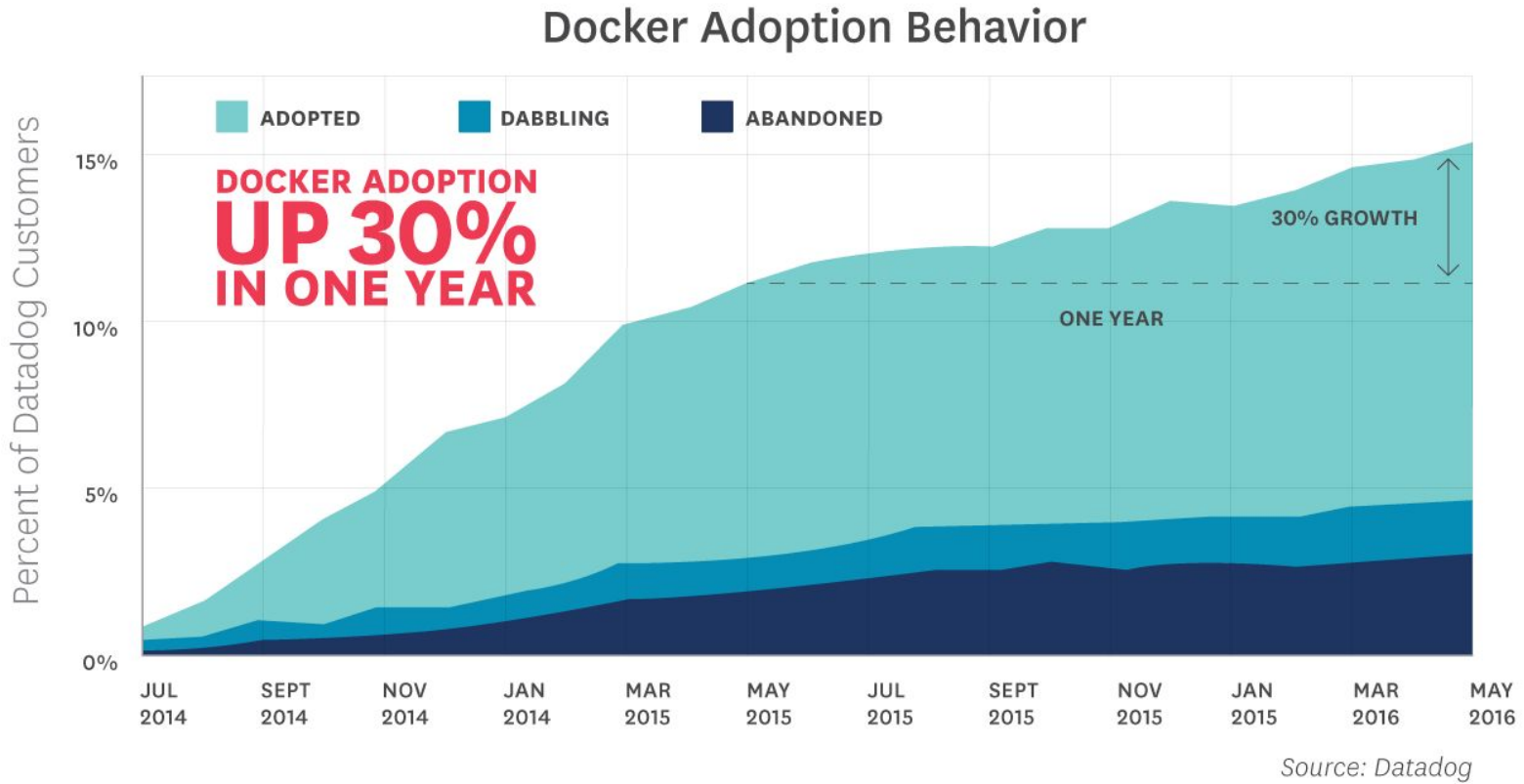
Docker basics

Docker history

- From dotCloud to Docker inc
- Founded by Solomon Hykes
- Open source project since March 2013
- 29 july 2016, docker 1.12 release
- On github:
  - 41175 stars
  - 10000 forks
  - 2000+ contributors
  - 17000+ pull requests

General information

- We will use Docker 1.13/17.03.1-ce
  - new management command groups
  - docker ps → docker container ls
- New Docker versioning scheme : YY.mm.p
- Two editions: community and enterprise
- More details: https://blog.docker.com/2017/03/docker-enterprise-edition/


Docker Adoption Behavior
ADOPTED
DABBLING
ABANDONED
DOCKER ADOPTION
UP 30%
IN ONE YEAR
30% GROWTH
ONE YEAR
Percent of Datadog Customers
15%
10%
5%
0%
JUL 2014
SEPT 2014
NOV 2014
JAN 2014
MAR 2015
MAY 2015
JUL 2015
SEPT 2015
NOV 2015
JAN 2015
MAR 2016
MAY 2016
Source: Datadog

Multiplicity of Goods





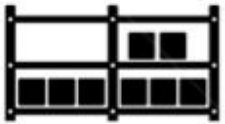


Do I worry about how goods interact (e.g. coffee beans next to spices)

Multiplicity of methods for transporting/storing



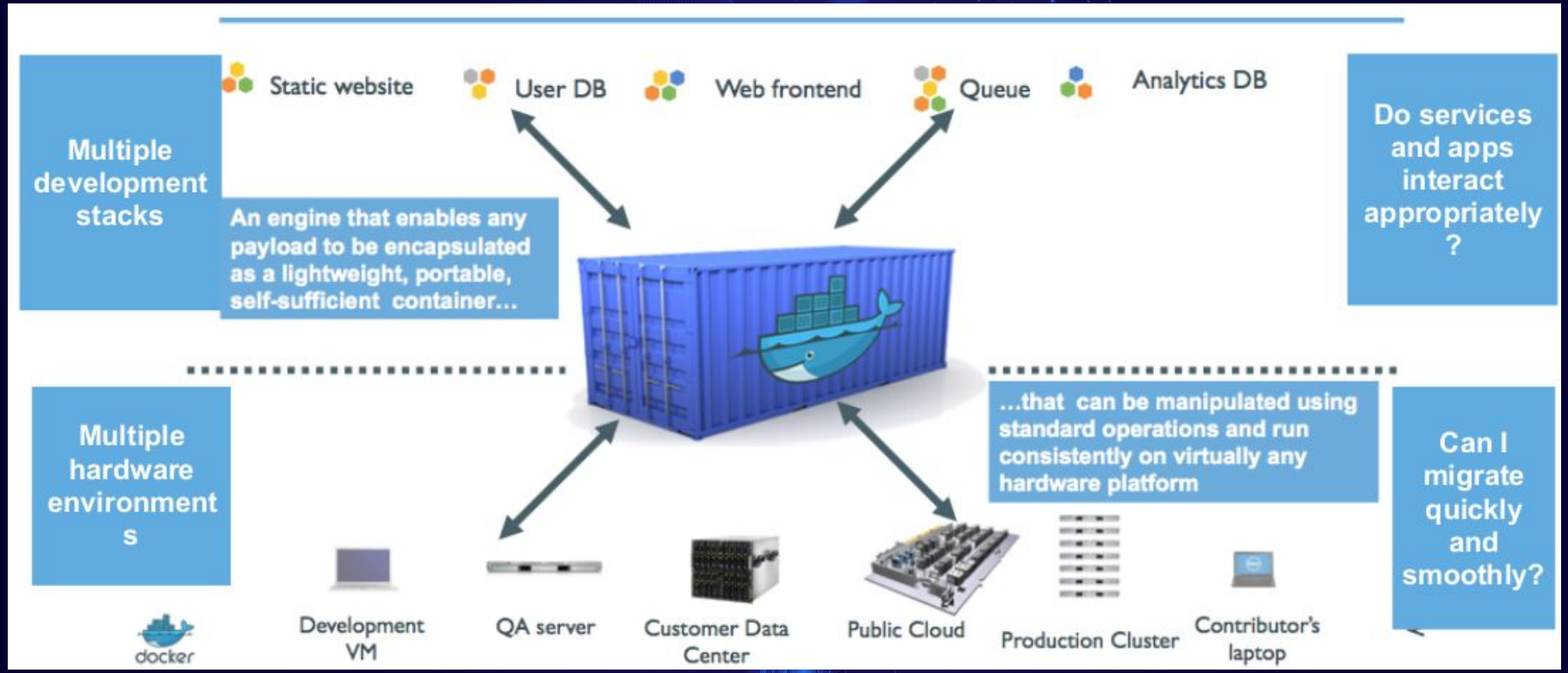



Can I transport quickly and smoothly (e.g. from boat to train to truck)

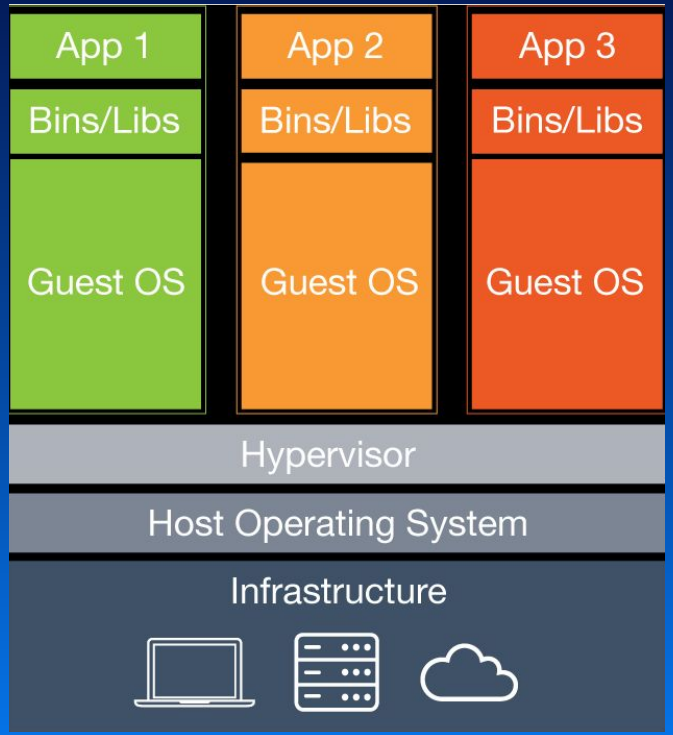
IPPON DEVOXX France

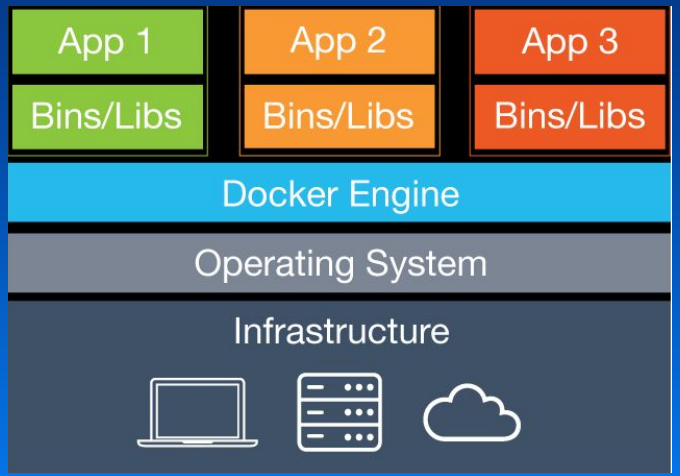

Multiplicity of Goods
A standard container that is loaded with virtually any goods, and stays sealed until it reaches final delivery.
Do I worry about how goods interact (e.g. coffee beans next to spices)
…in between, can be loaded and unloaded, stacked, transported efficiently over long distances, and transferred from one mode of transport to another
Multiplicity of methods for transporting/storing
Can I transport quickly and smoothly (e.g. from boat to train to truck)
docker


Multiple development stacks
Static website
User DB
Web frontend
Queue
Analytics DB
Do services and apps interact appropriately ?
An engine that enables any payload to be encapsulated as a lightweight, portable, self-sufficient container…
…that can be manipulated using standard operations and run consistently on virtually any hardware platform
Multiple hardware environments
Can I migrate quickly and smoothly?
docker
Development VM
QA server
Customer Data Center
Public Cloud
Production Cluster
Contributor's laptop

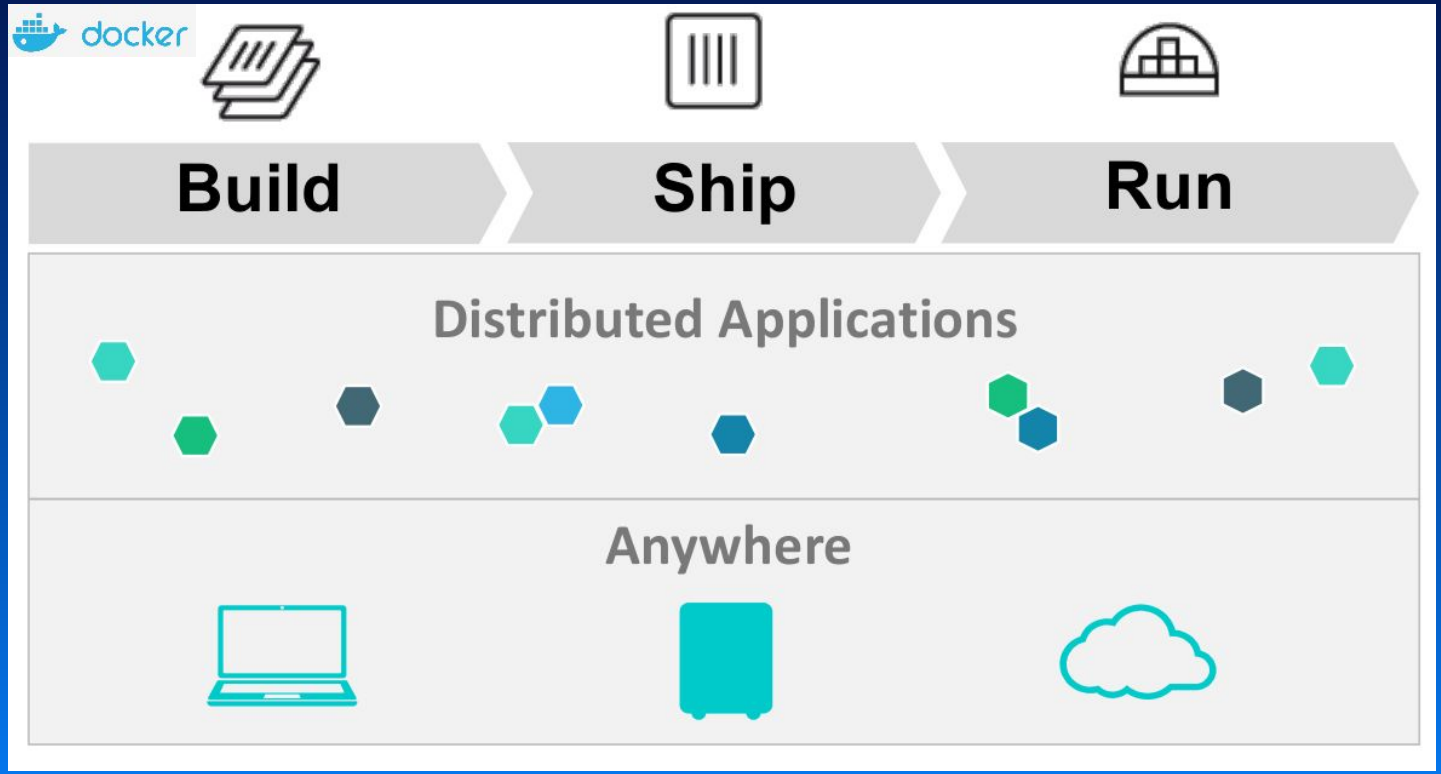

Hey guys, you're kind of quiet. Anyone want to troll us to wake us up?
Ah yeah. I've got an idea, but it's not really trolling...
Try anyway
Do you agree that Docker is just a kind of light-weight VM?
?!
NO
Pfff
You're crazy!
!
Docker isn't a VIRTUAL MAC
You can do "ONE CONTAINER per PROCESS" or "ONE CONTAINER per APP"
Just look at it for a second, God dammit!
for A SINGLE PROCESS ONLY!
Of course!
How could you SAY that?!
CommitStrip.com

Virtual machine VS Docker



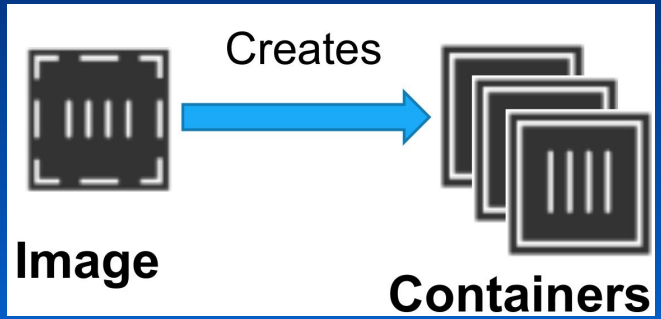


The Docker mission



Base concepts

- Cgroups & Namespaces isolation
- Images & Containers



- Container = single process
- Container is “ephemeral”

#Docker ?

Why would we change ?

• For the hype ?	Nope
• Standardize deployments ?	Yes
• Reproducibility ?	Yes
• Security & Isolation ?	Kinda
• To scale ?	No
• First steps to the cloud ?	Kinda

Building docker images

Introduction to Dockerfile

- Configuration file
- Like a cooking recipe
- Contains all instructions for building an Image
- Can be easily integrated into a CI pipeline

Main Dockerfile instructions

- **FROM**
  - Base image to use with its version (latest by default)
  - Must be the first instruction in Dockerfile
- **RUN**
  - Execute an instruction, a command into the base image
- **COPY**
  - Copy a host source file into the image destination path
- **EXPOSE**
  - Specify the application port to expose outside the container
- **ENTRYPOINT**
  - Command that will be always executed on Container startup

Build a Dockerfile

- Command
  - docker image build -t [image_name:tag] [context_path]
- Context path
  - Directory used for building
  - Files from the context can be copied inside the image
  - Path is relative from current path
  - Type “.” for using the current directory
  - Docker daemon will build the image from the file name “Dockerfile” by default (-f for a specific file name)

Exercise steps (15 min)

1. Create an image for each “service”
    1.1. Java Runtime Environment
        1.1.1. openjdk:8-jre-alpine
    1.2. Port 8080 exposition
    1.3. Run the service on startup
    1.4. Build the image

Cheat sheet

- Dockerfile
    - FROM
    - COPY
    - RUN
    - EXPOSE
    - ENTRYPOINT
- docker image
    - ls
    - build
    - tag
    - pull
    - push
- [“sh”, “-c”, “java -jar ~/path/service.jar”]

Exercise solution

```
FROM openjdk:8-jre-alpine
# Expose a port outside the container
EXPOSE 8080
# Specify the command to run on startup
ENTRYPOINT [ "sh", "-c", "java -jar /service.jar" ]
# Copy the built jar into the Docker image
COPY poule-0.0.1.jar /service.jar
```

Docker network

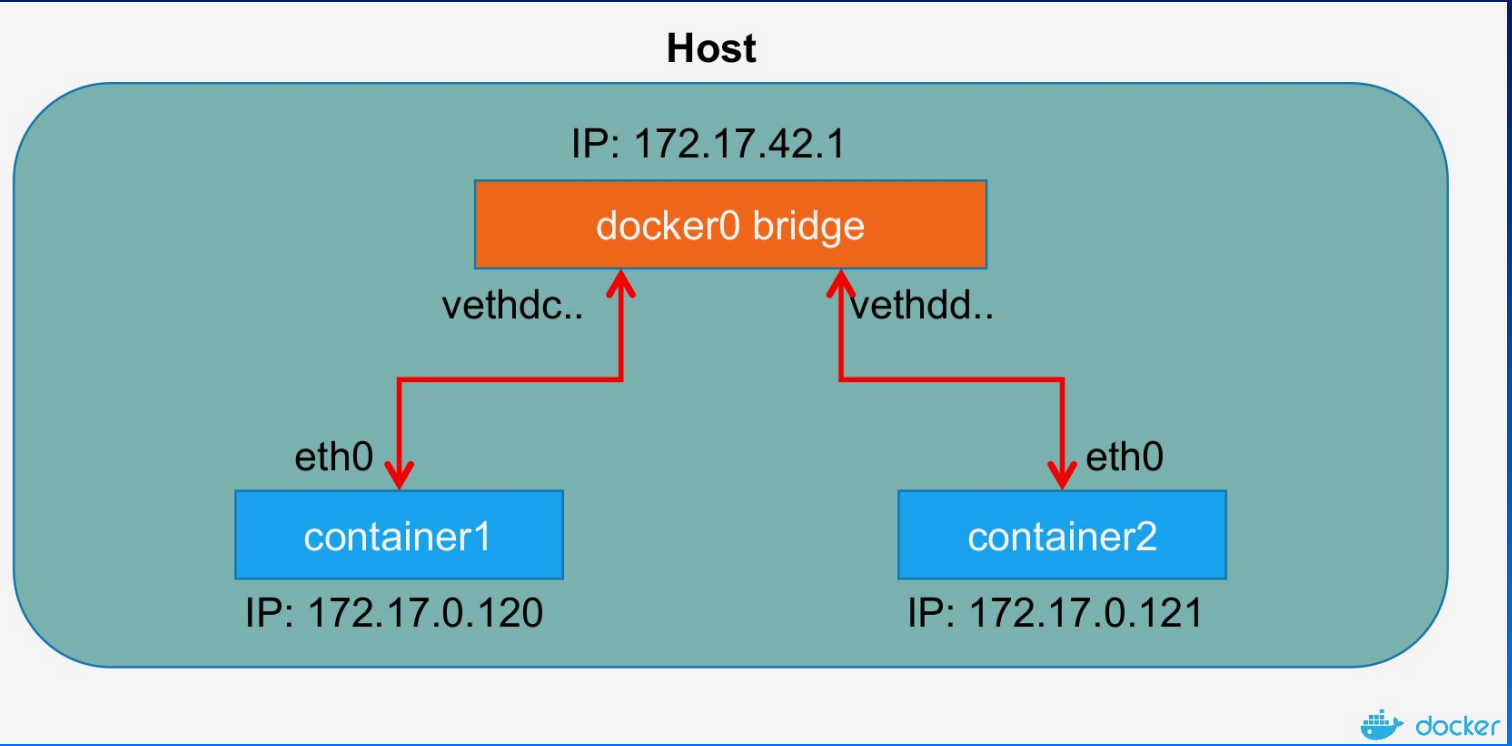
Network with Docker

- 3 network mods available
  - host: use directly the host network
  - bridge: use a virtual Ethernet bridge interface

```
docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
 link/ether 02:42:aa:29:08:70 brd ff:ff:ff:ff:ff:ff
 inet 172.17.42.1/16 scope global docker0
    valid_lft forever preferred_lft forever
 inet6 fe80::42:aaff:fe29:870/64 scope link
    valid_lft forever preferred_lft forever
```

  - none: can't access to host network
- **1 Docker bridge network = 1 host network interface**
- Can create custom bridge sub-networks

Bridge network



Docker bridge network

Exercise steps (5 min)

1. Run a “slip” container, named “slip”, in detached mode
2. Run a “poule” container and observe output console

Cheat sheet

- docker container
  - ls
  - run
    - --name [name]
    - -it
    - -d
    - -p [host]:[container]
  - inspect

Exercise solution

1. `docker container run --name slip -d slip`
2. `docker container run -it -p 8080:8080 poule`
3. `...`

Doesn't work :(

```
... 7 more
feign.RetryableException: sleep executing GET http://sleep:8080/request
        at feign.FeignException.errorExecuting(FeignException.java:67)
        at feign.SynchronousMethodHandler.executeAndDecode(SynchronousMethodHandler.java:104)
        at feign.SynchronousMethodHandler.invoke(SynchronousMethodHandler.java:76)
        at feign.ReflectiveFeign$FeignInvocationHandler.invoke(ReflectiveFeign.java:103)
        at com.ippon.jug.pull.Schedduler.$Proxy87.getRequest(Unknown Source)
        at com.ippon.jug.pull.Schedduler.Worker.run(Worker.java:21)
        at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1142)
        at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:617)
        at java.lang.Thread.run(Thread.java:745)
Caused by: java.net.UnknownHostException: sleep
        at java.net.AbstractPlainSocketImpl.connect(AbstractPlainSocketImpl.java:184)
        at java.net.SocksSocketImpl.connect(SocksSocketImpl.java:392)
        at java.net.Socket.connect(Socket.java:589)
        at sun.net.NetworkClient.doConnect(NetworkClient.java:175)
        at sun.net.www.http.HttpClient.openServer(HttpClient.java:432)
```

Exercise steps (5 min)

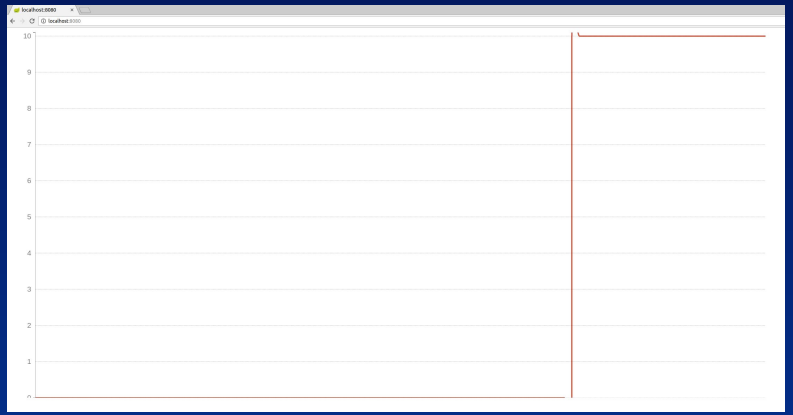
1. Create a custom bridge network
2. Run a “slip” and “poule” containers using this network
3. Observe result

Cheat sheet

- docker network
  - ls
  - create
  - inspect
- docker container
  - ls
  - run
    - --name [name]
    - -it
    - -d
    - -p [host]:[container]
    - --network
  - inspect

Exercise solution

1. docker network create --driver bridge disco
2. docker container run --network disco --name slip -d slip
3. docker container run --network disco -it -p 8080:8080 poule
4. docker network inspect disco



```
jmonsinjon$ docker network inspect --format '{{json .Containers}}' disco | jq .
{
  "2c54aa64fb44da4d8d6c1cae524b4f5f56bbf8124f44bc0908e85c4d76741c77": {
    "Name": "sleep",
    "EndpointID": "79112b068c3537f3052360207cae1f9abf0bd8355e473572b5e09a5cee9972b4",
    "MacAddress": "02:42:ac:12:00:02",
    "IPv4Address": "172.18.0.2/16",
    "IPv6Address": ""
  },
  "c841fb0d953f0810decb6c9e9469f66093a53284ce470626d70582009ccae534": {
    "Name": "elated_lamarr",
    "EndpointID": "228f0d8f616adf97536cd102b7297d8f86f3b9776b410ed9e92fc4d7d823b19a",
    "MacAddress": "02:42:ac:12:00:03",
    "IPv4Address": "172.18.0.3/16",
    "IPv6Address": ""
  }
}
```

Docker Compose

Docker-Compose

- External Docker tool
- Describes all services that make up an “application”
  - Network
  - Volumes
  - Environment variables
  - etc.
- Simple YML syntax
- Bridge network contains DNS

Compose file exemple

```
version: "3"
services:
  slip:
    build: ~/slip-src/

  poule:
    image: poule:latest
    ports:
      - "8080:8080"
```

Use the Dockerfile inside the folder to build and run an image

Use the local “pull:latest” image or pull from Docker hub

Bind container port to host.
Just specify container port for dynamic mapping

Exercise steps (15 min)

1. Write a compose file to launch “slip” and “poule” services
2. Run your application
3. Observe results
   3.1. Docker-Compose
   3.2. Web browser

Cheat sheet

- docker-compose file
  - version
  - services
    - build
    - image
    - ports
    - volumes
- docker-compose
  - -p [context_name]
  - up (-d)
  - start
  - stop
  - rm
  - down
  - scale

Exercise solution

1.

```
version: "3"
services:
  slip:
    image: slip:latest
  poule:
    image: poule:latest
    ports:
      - "8080:8080"
```

2. `docker-compose up -d`

Service scaling

Exercise steps (5 min)

1. Scale up your “slip” service
2. Observe result on your web browser
3. Kill the first slip container
4. Observe result on your web browser

Cheat sheet

- docker-compose
  - -p [context_name]
  - up (-d)
  - start
  - stop
  - rm
  - down
  - scale
- docker container
  - ls
  - start
  - stop
  - kill
  - inspect

Exercise solution

1. `docker-compose scale slip=3`
2. `Nothing happened on web ui`
3. `docker container kill dockertraining_slip_1`
4. `Nothing happened on web ui`

Performances aren't increased but failover is managed !

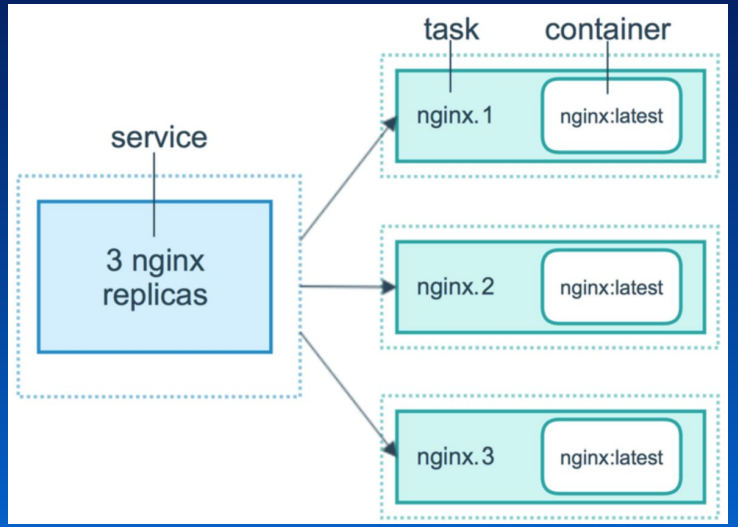
Docker Swarm

Orchestrators

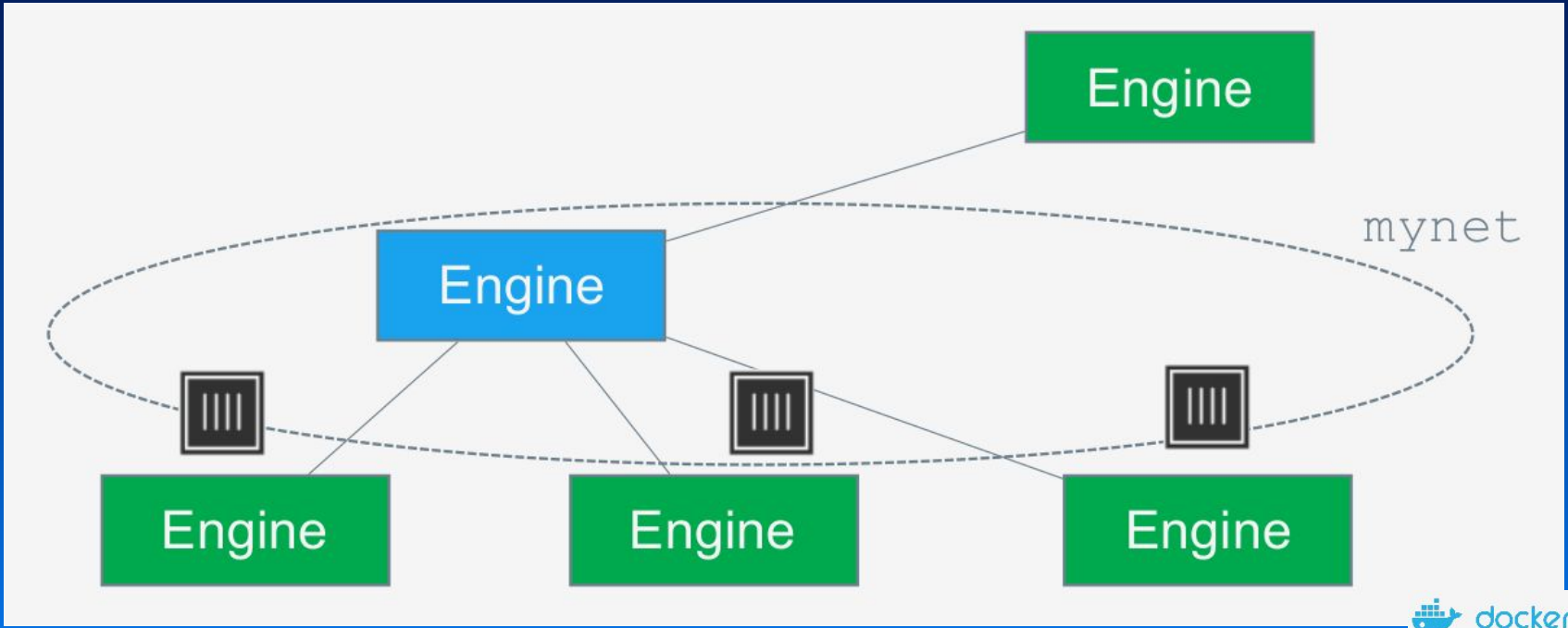
- Manage a cluster of servers
- Distribute containers according to constraints
- Ensures that active services match an expected state
- Main actors
  - Mesos (*Apache*)  / Marathon (*Mesosphere*) 
  - Kubernetes (*Cloud Native Computing Foundation*) 
  - Swarm Mode (*Docker*)  *(Not old Swarm !)*

Docker swarm to the rescue !

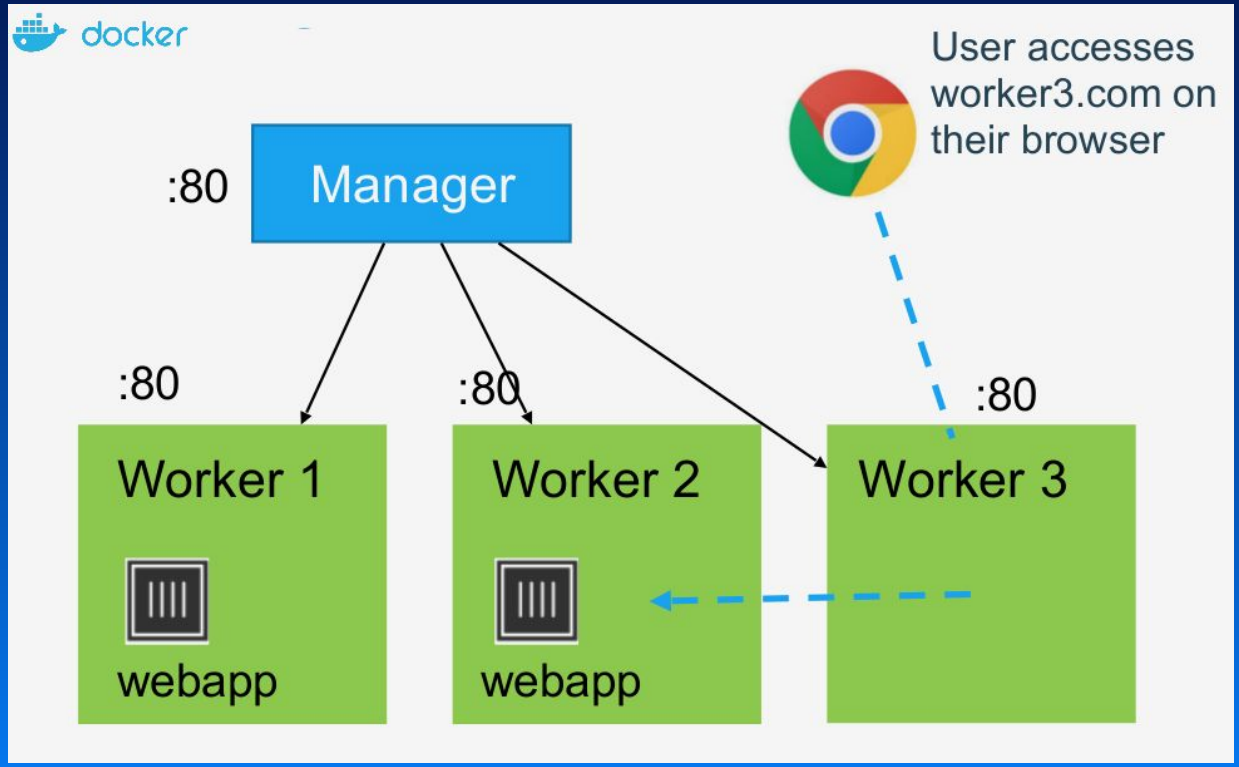
- The concept of “service”
- Embedded discovery
- Load balancing
- Backups and disaster recovery
- Rolling updates
- …



The overlay network



Routing Mesh



Docker swarm commands

- docker swarm
  - **init**: Initialize a Swarm cluster (only on one node!)
  - **join**: Join the current node inside a Swarm cluster
  - **leave**: Leave the Swarm cluster
- docker node (only managers)
  - **ls**: List cluster nodes (docker engines)
  - **promote**: Promote a worker node to manager node
  - **demote**: Demote a manager node to a worker node
  - **rm**: Remove a node from the Swarm cluster

Create your docker swarm cluster

Exercise steps (5 min)

1. Init a Swarm cluster on the first node
2. Join your others nodes as worker in the Swarm cluster

http://labs.play-with-docker.com

Cheat sheet

- docker swarm
  - init
  - join
- docker node
  - ls
  - promote
  - demote
  - ps

Exercise solution

1. Manager node
   1.1. `docker swarm init`
   1.2. `docker swarm join-token worker`
2. Worker node
   2.1. `docker swarm join --token SWMTKN-1-3swbrypttthl3pohkx3f3gxkrqv9jjpzb5thvrmfq327zmrfpc-90h5m0slpxf0yqsmffs7b0mq6 192.168.0.100:2377`
3. `docker container run -it -p 8080:8080 -v /var/run/docker.sock:/var/run/docker.sock julienbreux/docker-swarm-gui:latest`

Deploy services on your cluster

Exercise steps (15 min)

1. Create an overlay network
2. Create and start services on this network
3. Observe results
4. Scale up slip service to 3
5. Observe results again

Cheat sheet

- docker network
  - create
    - --driver
    - --subnet
- docker service
  - create
    - -p
    - --name
    - --replicas
  - rm
  - ls
  - ps
  - scale
  - update

Exercise solution

1. docker network create --driver overlay disco
2. docker service create --network disco --name slip ippontrain/slip:v0.0.1
3. docker service create --network disco -p 8080:8080 --name poule ippontrain/poule:v0.0.1
4. Observe results:
   - 4.1. Docker Swarm GUI
   - 4.2. Poule interface ([any_swarm_node_ip]:8080)
5. docker service scale slip=3

Stratégie de déploiement

- To ensure an high availability service, you can customize your update strategy.
- docker service update --update-*
- Modify configuration but doesn’t re-deploy your services
- update-*
  - delay: Delay between updates
  - parallelism: Maximum number of tasks updated simultaneously
  - failure-action: Action on update failure

Exercice steps (5 min)

1. Change the slip service update strategy
   1.1. 5s between each instance update
   1.2. whatever !
2. Update slip image to : v0.0.2
3. Observe results

Cheat sheet

- docker service
  - create
  - rm
  - ls
  - ps
  - scale
  - update
    - --image
    - --rollback

Exercise solution

1. `docker service update --update-delay 5s slip`
2. `docker service update --image ippontrain/slip:v0.0.2 slip`
3. Observe UI | `watch docker service ps slip`

Health checks

- In order to enable the service monitoring Swarm introduces the *health checks*
- The health status is in addition of the *normal* status
- New instruction in Dockerfile: **HEALTHCHECK**
  - **HEALTHCHECK NONE**
    - Will deactivate health check for the containers based on the image
  - **HEALTHCHECK [OPTIONS] CMD command**
    - Command will be run periodically
    - Options:
      - `--interval=DURATION` (default: `30s`)
      - `--timeout=DURATION` (default: `30s`)
      - `--retries=N` (default: `3`)

Exercice steps (10 min)

1. Update slip image to : v0.0.3
2. Observe results
3. ...
4. Guess you should rollback !
5. Observe results

Cheat sheet

- docker service
  - create
  - rm
  - ls
  - ps
  - scale
  - update
    - --image
    - --rollback

Exercise solution

1. `docker service update --image ippontrain/slip:v0.0.2 slip`
3. `docker service update --image ippontrain/slip:v0.0.3 slip`
4. `docker service inspect slip`
5. Why ?

```
HEALTHCHECK --interval=2s --retries=3 \
CMD exit $(/usr/bin/curl -fs http://localhost:8080/dockerHealth | 0)
```

6. `docker service update --rollback slip`
7. everything is back to normal on the GUI

Docker ~~1.13~~ 17.03.1-ce features

Bundle and stack deploy

- Deploy a list of services on multiple servers
- From Compose file or Bundle file (.DAB)
- Create a “stack” on a Swarm cluster
- Automatic overlay network instead of bridge network
- Enable use of specific “stack” commands
- Secret management

From compose to stack

Exercise steps (10 min)

1. Copy your compose file
2. Deploy the stack
3. Observe results

Cheat sheet

- docker network
  - create
    - --driver
    - --subnet
- docker service
  - create
    - -p
    - --replicas
  - rm
  - ls
  - ps
  - scale
  - update

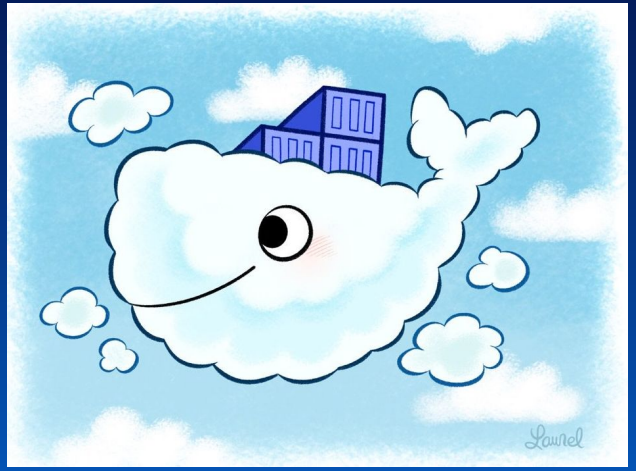
Exercise solution

2. docker stack deploy -c ~/[compose_file_path] [project_name]
3. Observe results:
   3.1. Docker Swarm GUI
   3.2. Pull interface ([any_swarm_node_ip]:8080)

Conclusion

Happy whale

- Docker ecosystem is rich
- Easy to use
- Powerful
- Moves fast
- The hype



Sad whale

- Moves fast
- Swarm is a little bit young
  - Few networking issues
  - Lack of production feedback
  - No distributed storage
- Docker is not the only answer
  - RKT
  - Kubernetes, Mesos/Marathon, Rancher, ...

github.com/ImFlog/docker-discovery

PARIS
BORDEAUX
NANTES
LYON
MARRAKECH
WASHINGTON DC
NEW-YORK
RICHMOND
MELBOURNE

contact@ippon.fr
www.ippon.fr - www.ippon-hosting.com - www.ippon-digital.fr
@ippontech
-
01 46 12 48 48