Avance de proyecto: Análisis de seguridad de Docker

Maximiliano Osorio

Agenda

1 Motivación y descripción del problema

2 Avances
- Marco teorico
- Implementación
- Discusión y posibles ataques
  - Secure image repository
  - ARP-spoofing attack

3 Conclusiones preliminares

4 Trabajo restante

5 Referencias

Motivación

Motivación

Motivación

django web frontend	?	?	?	?	?	?
node.js async API	?	?	?	?	?	?
background workers	?	?	?	?	?	?
SQL database	?	?	?	?	?	?
distributed DB, big data	?	?	?	?	?	?
message queue	?	?	?	?	?	?
	my laptop	your laptop	QA	staging	prod on cloud VM	prod on bare metal

FIGURE: Representation flow shop scheduling problem. (CC-By-Sa)

Motivación

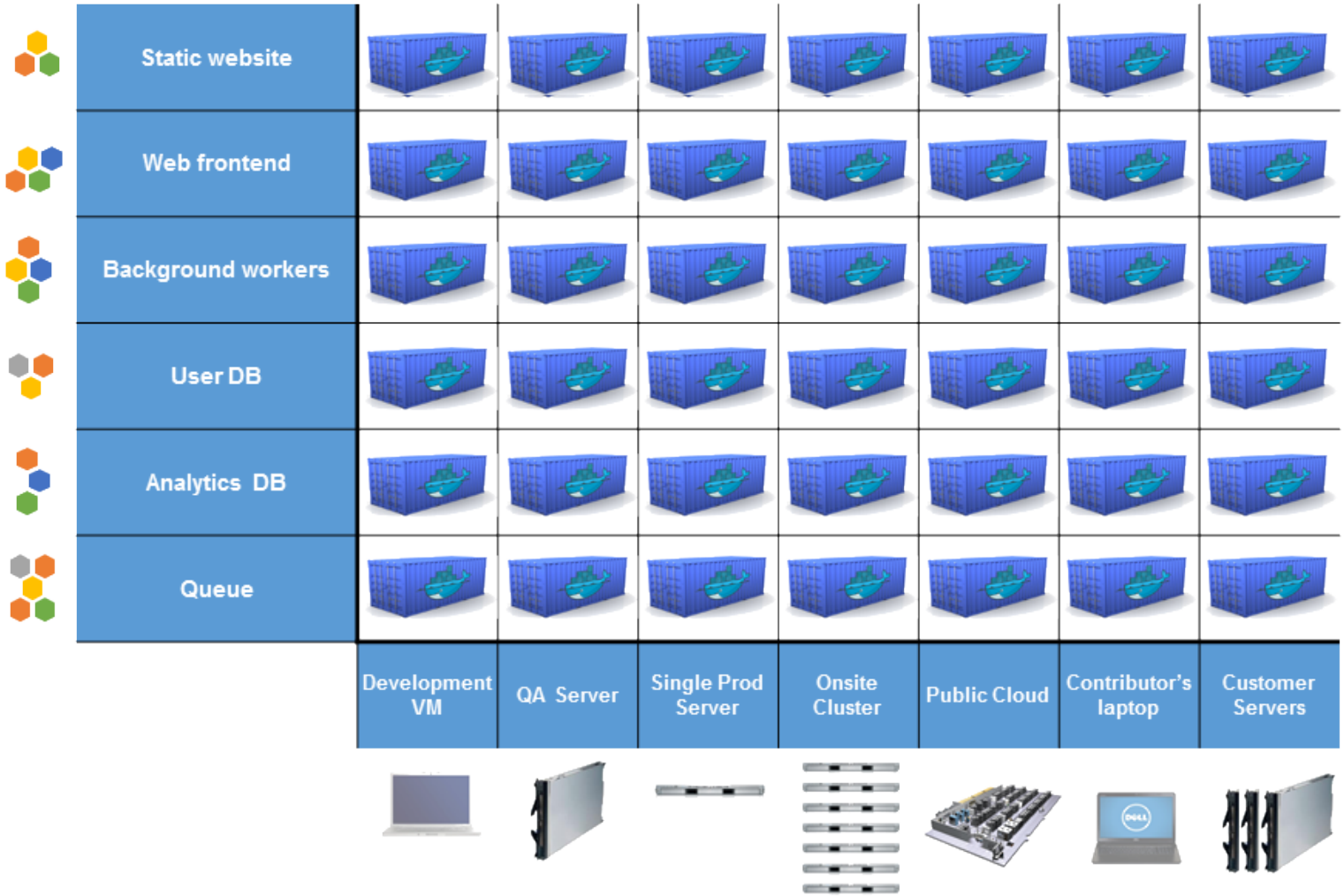

FIGURE: Representation flow shop scheduling problem. (CC-By-Sa)

Motivación

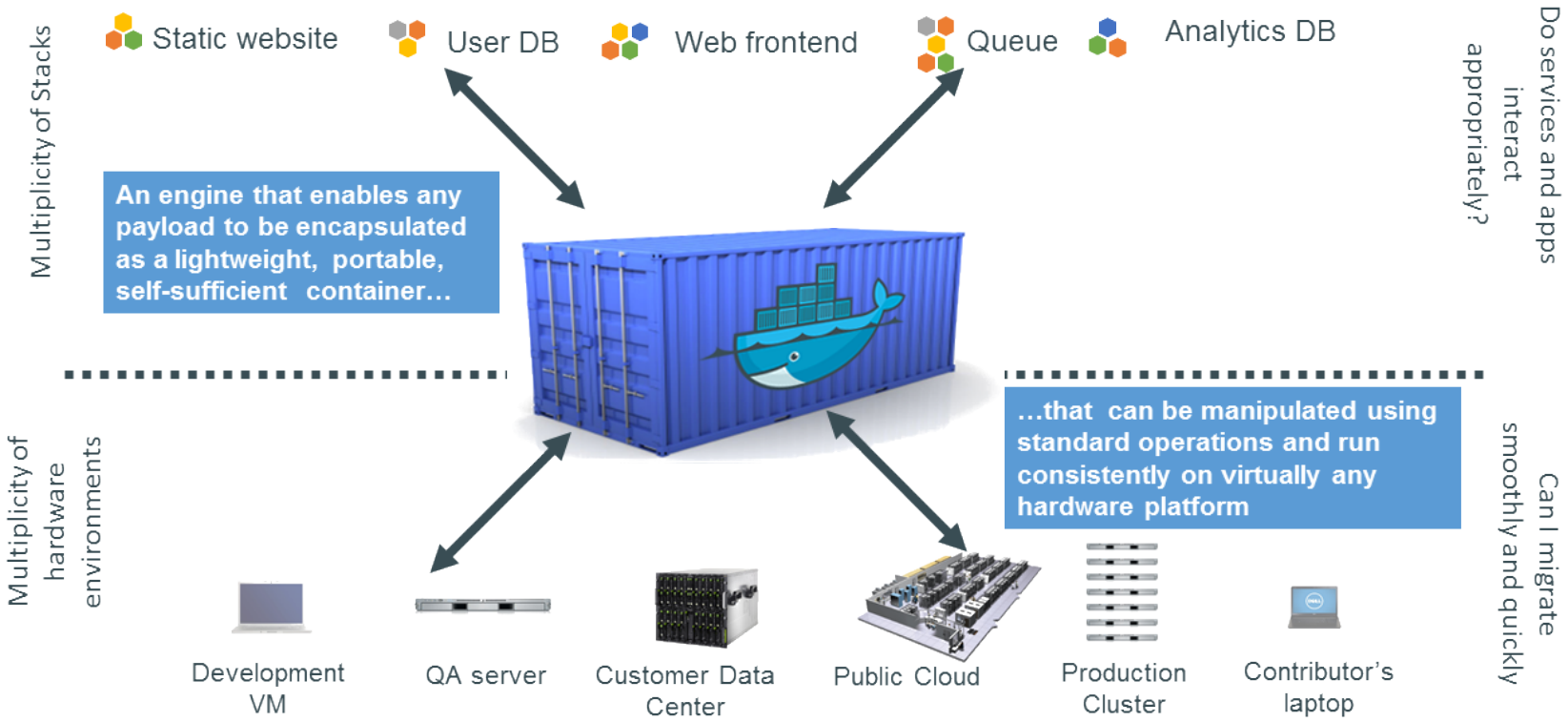

FIGURE: Representation flow shop scheduling problem. (CC-By-Sa)

Docker

- Docker es un proyecto open-source que utiliza la tecnología de los containers (libcontainer) para “construir, migrar y correr aplicaciones distribuidas". Los containers se basan en *container-based virtualization*
- ¿Qué son los containers ?
  - Desde lejos, parecen ser como VM.
  - Puedo instalar aplicaciones, tengo root, tengo red, montar filesystems, etc.
  - Pero son ambientes virtuales livianos, rápidos de iniciar (boot en ms), facil de migrar, deterministas y otros.

¿Problema ?

¿Es seguro correr aplicaciones en Linux Containers ?

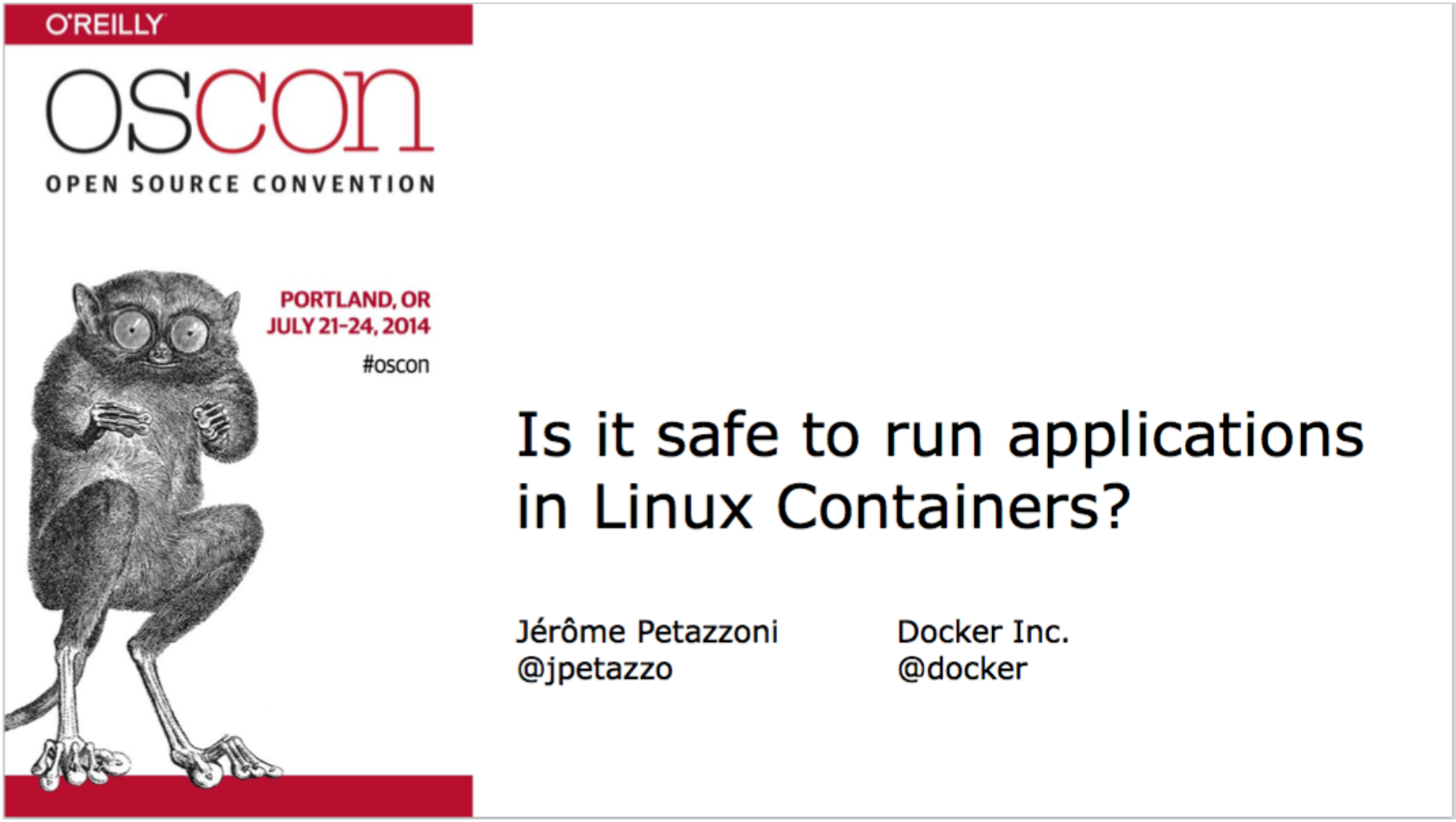

FIGURE: De : Jerome Petazzoni, Docker

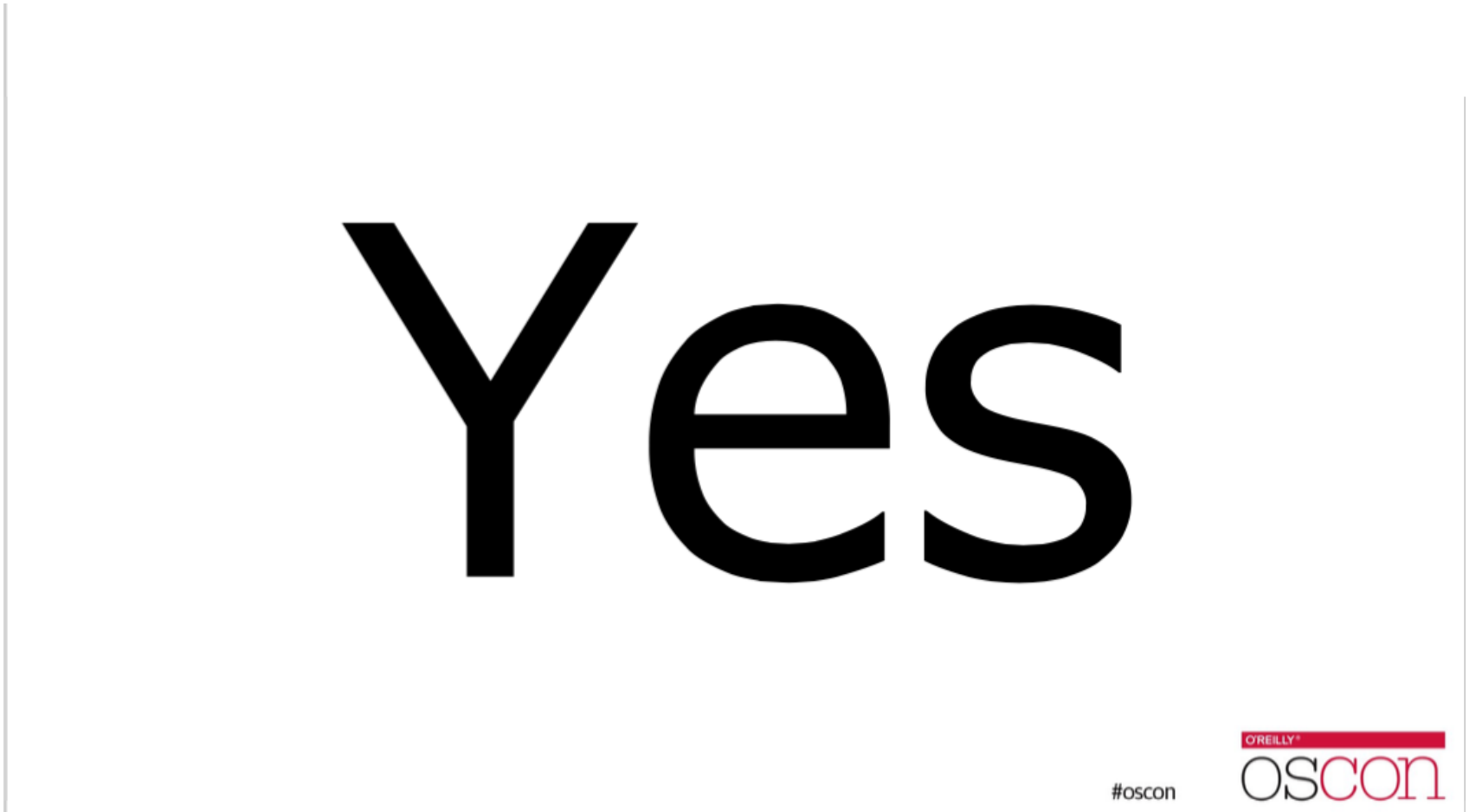

FIGURE: De : Jerome Petazzoni, Docker

```
/* shocker: docker PoC VMM-container breakout (C) 2014 Sebastian Krahmer
 *
 * Demonstrates that any given docker image someone is asking
 * you to run in your docker setup can access ANY file on your host,
 * e.g. dumping hosts /etc/shadow or other sensitive info, compromising
 * security of the host and any other docker VM's on it.
 *
 * docker using container based VMM: Sebarate pid and net namespace,
 * stripped caps and RO bind mounts into container's /. However
 * as its only a bind-mount the fs struct from the task is shared
 * with the host which allows to open files by file handles
 * (open_by_handle_at()). As we thankfully have dac_override and
 * dac_read_search we can do this. The handle is usually a 64bit
 * string with 32bit inodenumber inside (tested with ext4).
 * Inode of / is always 2, so we have a starting point to walk
 * the FS path and brute force the remaining 32bit until we find the
 * desired file (It's probably easier, depending on the fhandle export
 * function used for the FS in question: it could be a parent inode# or
 * the inode generation which can be obtained via an ioctl).
 * [In practise the remaining 32bit are all 0 :]
```

#oscon O'REILLY OSCON

FIGURE: De : Jerome Petazzoni, Docker

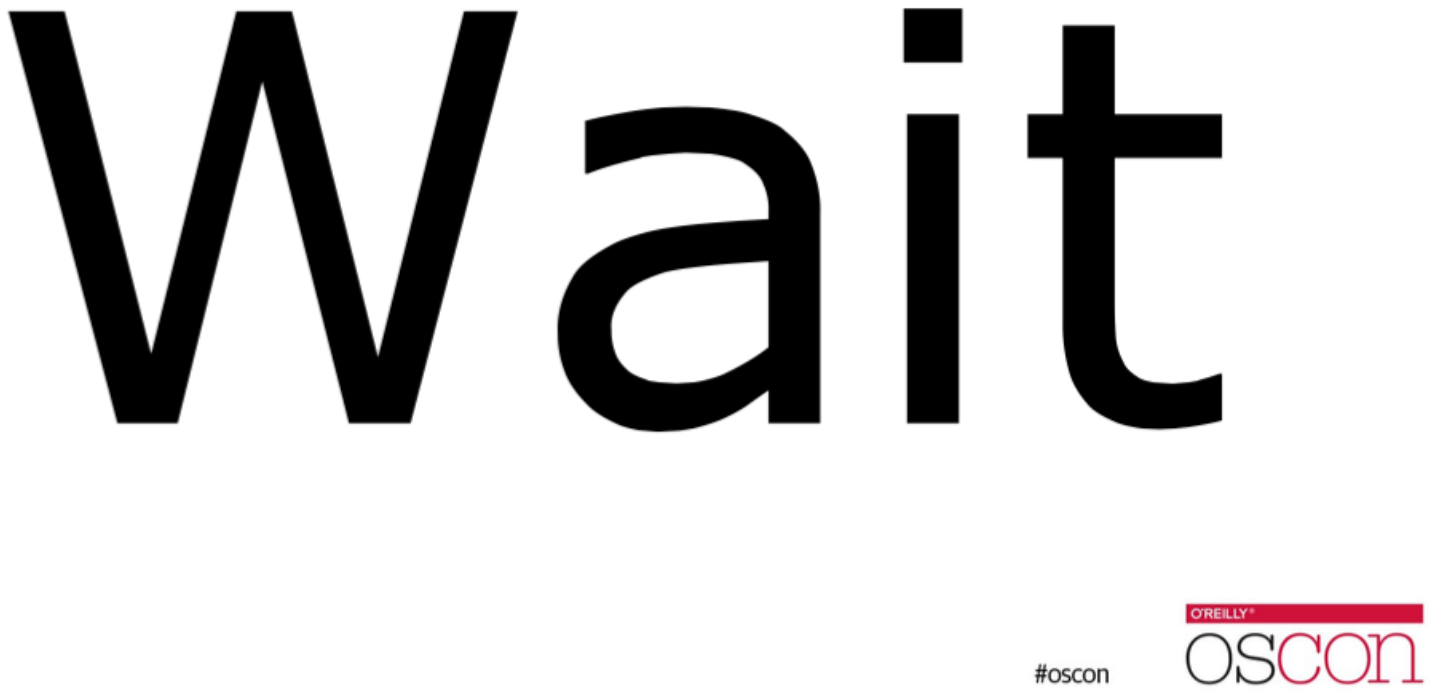

FIGURE: De : Jerome Petazzoni, Docker

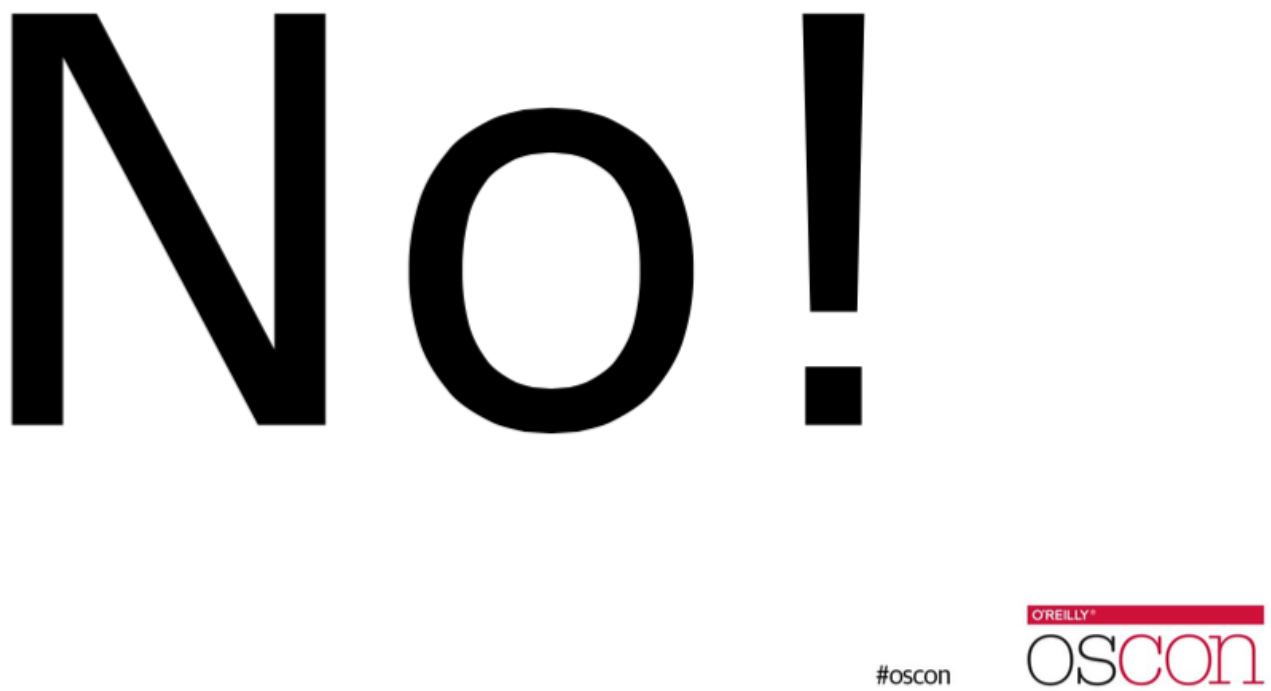

FIGURE: De : Jerome Petazzoni, Docker

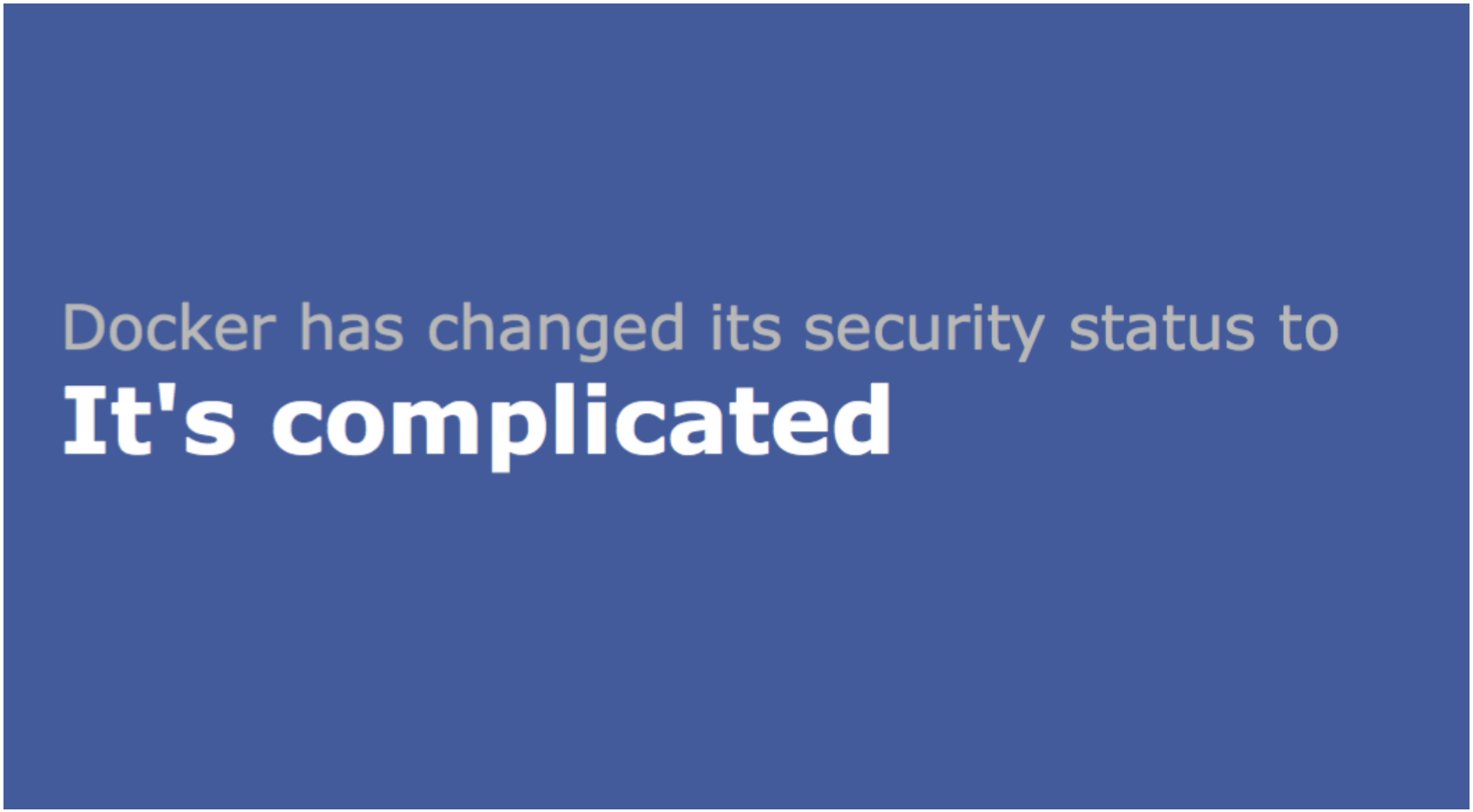

FIGURE: De : Jerome Petazzoni, Docker

Agenda

1 Motivación y descripción del problema

2 Avances
- Marco teorico
- Implementación
- Discusión y posibles ataques
  - Secure image repository
  - ARP-spoofing attack

3 Conclusiones preliminares

4 Trabajo restante

5 Referencias

Componentes

- Docker registry
- Docker client
- Docker
  - Docker Images
  - Docker Containers

Docker registry

- Repositorios públicos (Docker Hub) y privados de imágenes.
- Permite subir y bajar imágenes con sistemas operativos o aplicaciones ya configuradas.
- Imágenes son verificadas autenticidad y integridad.
- Son usadas para construir Docker containers.

Docker image

- Template read-only que son obtenidas desde registry.
- Son usadas para construir Docker containers.
- Los cambios realizados se hacen a través de capas.

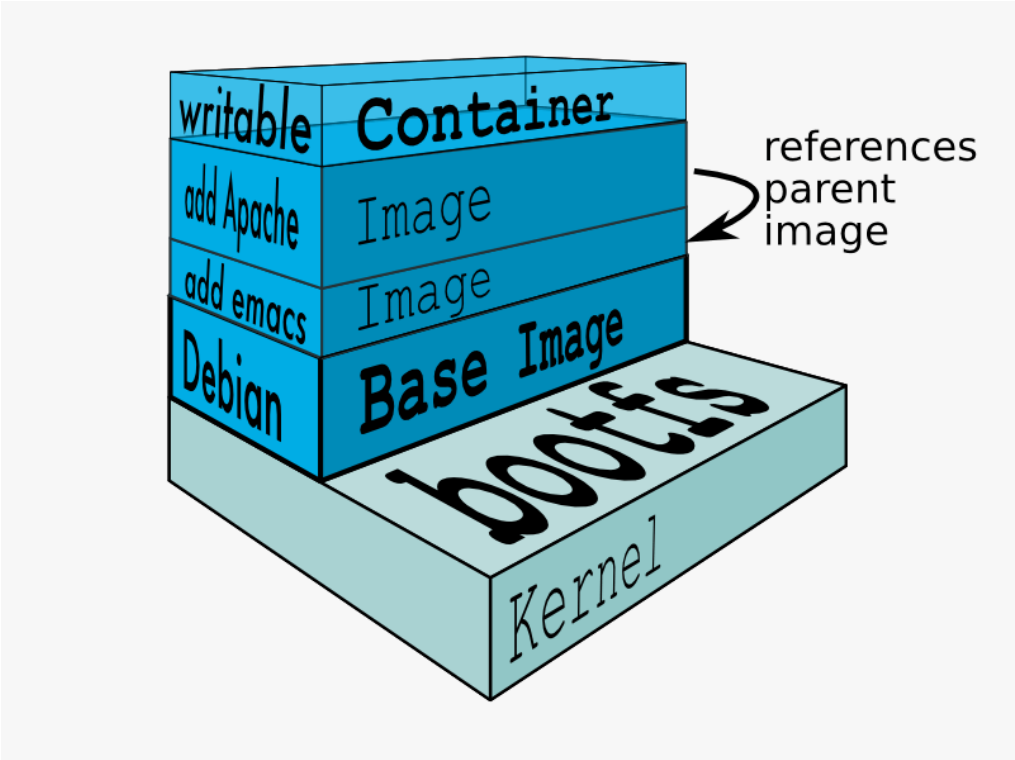

FIGURE: Representación *union file system*

Docker registry

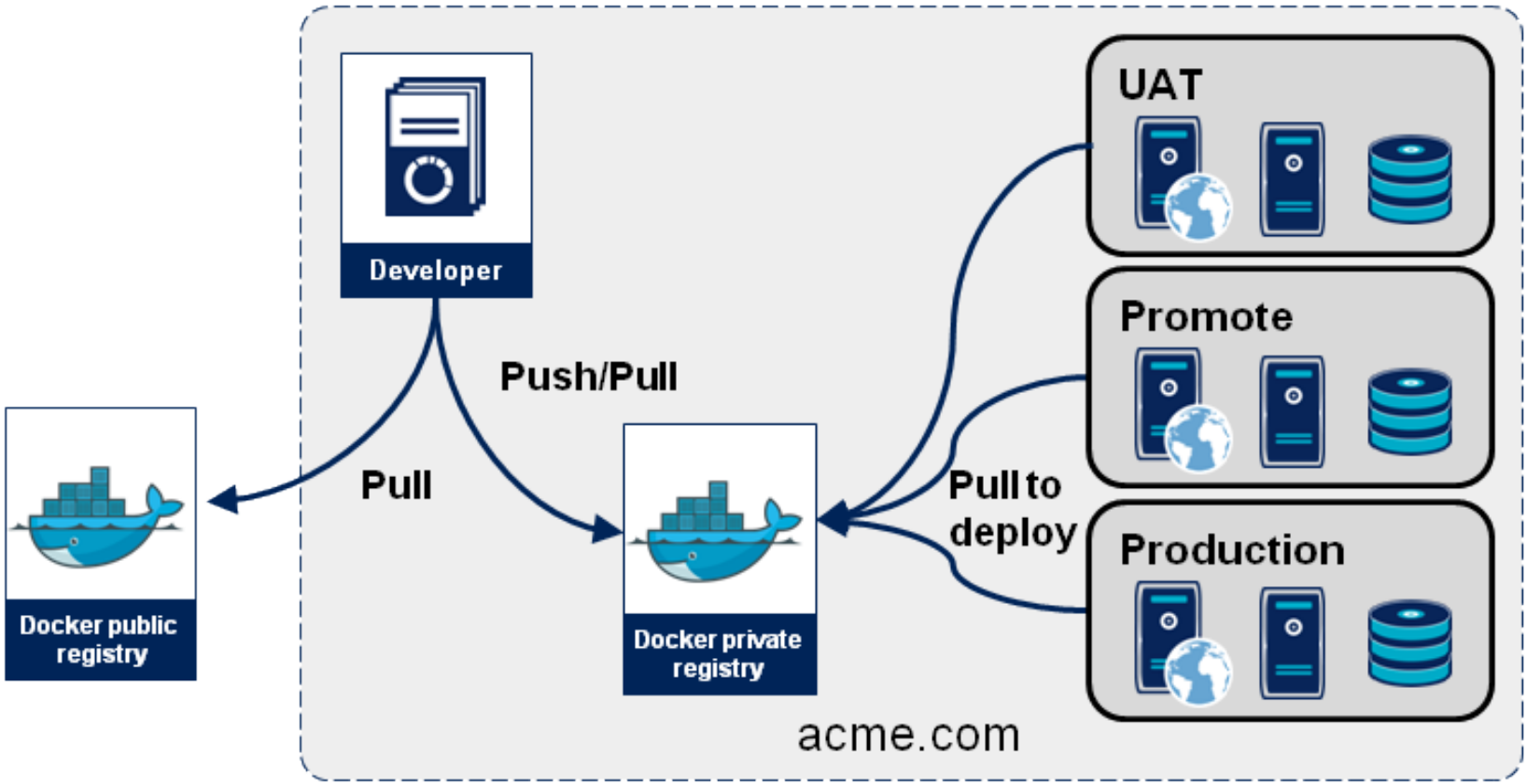

FIGURE: Representación *Dinámica de un Docker registry*

Docker containers

- Docker *container* consiste de un sistema operativo, con archivos de usuarios y metadata.
- Es construido en base a la imagen.

Ejemplo

```
docker run -ti ubuntu /bin/bash
```

Docker containers

- *Docker client* le informa al Docker que debe correr un *container*.
- El comando **/bin/bash** será el *init 0* del container
- **Traer la imagen :** Docker verifica la existencia de la imagen ubuntu y sino existe en el host, entonces Docker descarga de un *registry* ya sea privado o publico. Si la imagen existe entonces crea el container.
- **Asignar un filesystem y montar una capa read-write :** El container es creado en el **filesystem** y se añade una capa en modo **read-write** a la imagen.
- **Crear la red y conectar con el bridge interface :** Crea la interfaz de red que permite que el Docker container pueda hablar con el host a través del bridge (docker0).
- **Asignar un IP :** Asigna una ip del pool al container
- **Capturar el *output*, *input* y errores**.

Aislamiento

- **Limite de recursos :** Cgroups controlan la cantidad de recursos como CPU, memoria y disk I/O
- **Process :** Utiliza PID namespaces (kernel $\geq$ 2.6.32, el container solo puede ver los procesos del container.
- **Filesystem :** Mismo mecanismo y mismo resultado que con los procesos. Existen device que deben ser montados (ej. /sys).
- **Device :** Cgroups permite usar algunos devices [1] y bloquea la posibilidad de crear y usar otros devices.
- **IPC :** Los procesos corriendo en los *containers* utilizan *IPC namespaces* que permite la creación de un *IPC* separado y independiente para cada *container*

1. /dev/console, /dev/null, /dev/zero, /dev/full, /dev/tty*, /dev/urandom, /dev/random, /dev/fuse

Aislamiento

- **Network :** Para cada *container* , Docker crea una red independiente usando **network namespaces**, compuesta de su propia IP, rutas, **network devices**. Por defecto, la conexión se realiza gracias al host que provee un **Virtual Ethernet bridge** en la máquina, llamado docker0 que automaticamente realiza un **forward** de los paquetes entre las interfaces del container.

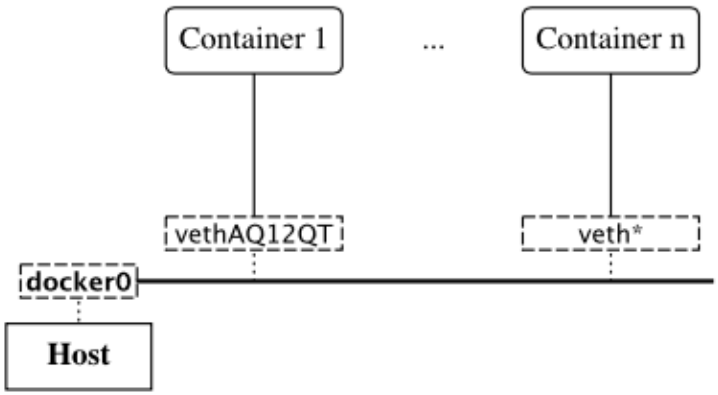

FIGURE: Network

Ambiente de prueba

Actualmente la implementación base se encuentra lista. Utilizando un nodo Centos con project-atomic y Docker 1.5.0. Se habilitó SElinux para realizar pruebas con un *Mandatory Access Control*

Versión
Docker version 1.5.0, build a8a31ef/1.5.0

Secure image repository

Basado en la idea de Fernandez, Monge y Hashizume de *Secure virtual machine image repository* [6] se investigó sobre si Docker hub cumple con los patrones especificados.

- Desde Docker $\geq$ 1.3.0 se completa las defensas descritas : Authenticator-Authorizer, Secure Channel, Security Logger/Auditor y filter.
- Rudenberg [15] y Jay [7] alertan que el proceso de traer un imagen no es seguro. CVE-2014-9356, CVE-2014-9357, CVE-2014-9358 [8]
- Al probar realizar las pruebas, se determina que no afectan a la última versión de Docker.

ARP-spoofing attack

- El modelo de red permite comunicación layer-2.
- Por defecto los containers se pueden comunicar, aunque existe una opcion –icc que aisla al container de los otros (red).
- –icc funciona a nivel de iptables (layer-3)

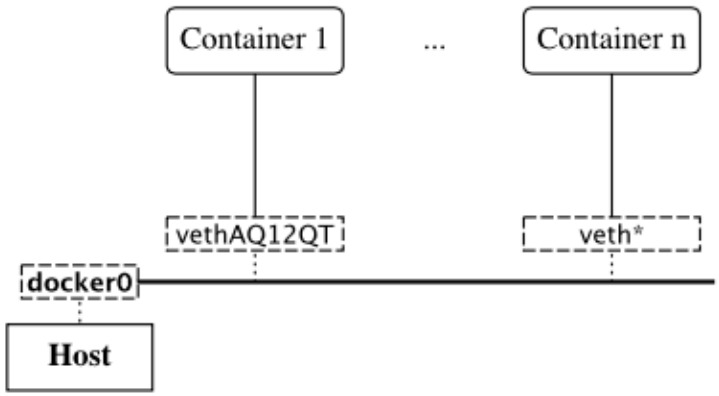

FIGURE: Network

Solución ARP-spoofing attack

nyatec propone dos soluciones para esto : [12].

- Eliminar la capacidad de NET_RAW para evitar que el *container* puede crear *PF_PACKET sockets* que son necesarios para el ARP spoofing attack.
- La utilización de etables para filtrar los *Ethernet frame* con direcciones de destino incorrecto.

La utilización de SElinux también es una solución comprobado en la prueba de concepto.

Conclusiones preliminares

- Docker es un sistema seguro siendo utilizado con la configuración por defecto aunque si existen vulnerabilidades como se nombro anteriormente.
- Utilización de herramientas complementarías como SElinux aumentan el grado de seguridad.

Trabajo restante

- Completar investigación con **Linux Capabilities** y **Mandatory Access Control**
- Alcances de ataques en un ambiente de Docker con SElinux (MAC) integrado.
- Contestar la pregunta ¿Es posible asegurar el host o los *containers* si el atacante salió del *container* ?.

Referencias I


Thanh Bui.
Analysis of docker security.
arXiv preprint arXiv :1501.02967, 2015.


Docker.
7.2. advantages of using docker, 2015.


Docker.
Understanding docker, May 2015.


Docker.
Use cases, 2015.


Wes Felter, Alexandre Ferreira, Ram Rajamony, and Juan Rubio.
An updated performance comparison of virtual machines and linux containers.
technology, 28 :32, 2014.


EduardoB. Fernandez, Raul Monge, and Keiko Hashizume.
Building a security reference architecture for cloud systems.
Requirements Engineering, pages 1–25, 2015.


Trevor Jay.
Before you initiate a "docker pull", Dic 2014.

Referencias II


LVM.
docker-io : multiple vulnerabilities, Dic 2014.


LVM.
Pid namespaces in the 2.6.24 kernel, 2015.


Victor Marmol, Rohit Jnagal, and Tim Hockin.
Networking in containers and container clusters.


Dirk Merkel.
Docker : lightweight linux containers for consistent development and deployment.
Linux Journal, 2014(239) :2, 2014.


nyantec.
Docker networking considered harmful, Mar 2015.


Pradeep Padala, Xiaoyun Zhu, Zhikui Wang, Sharad Singhal, Kang G Shin, et al.
Performance evaluation of virtualization technologies for server consolidation.
HP Labs Tec. Report, 2007.

Referencias III


Nathan Regola and J-C Ducom.
Recommendations for virtualization technologies in high performance computing.
In *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, pages 409–416. IEEE, 2010.


Jonanthan Rudenberg.
Docker image insecurity, Dic 2014.


Daniel Walsh.
Bringing new security features to docker, sep 2014.