

Docker y un poco de CDK

Curso Introductorio

Tomás GARCÍA-POZUELO BARRIOS

tomas.garcia-pozuelobarrios@soprasteria.com

Sopra Steria

Mayo 2019



Introducción a la formación

- Alcance de esta formación / presentación
 - ⇒ Introducción a Docker, CDK y tendencias tecnologías. Conceptos, Bases, etc...
Como un todo. Docker no tiene sentido sin las ultimas tendencias, ni viceversa.
 - ⇒ No es una formación súper práctica. Mayormente conceptos y teoría básica.
 - ⇒ Para profundizar no hay nada mejor que ser friki y echarle horas. Google es tu amigo (o stackoverflow).

Docker. Introducción (1/2)

- ¿Qué es Docker?
- Características y objetivos de Docker.

Docker. Introducción (1/2)

- ¿Qué es Docker?

From Wikipedia

Docker is a collection of interoperating [software-as-a-service](#) and [platform-as-a-service](#) offerings that employ [operating-system-level virtualization](#)

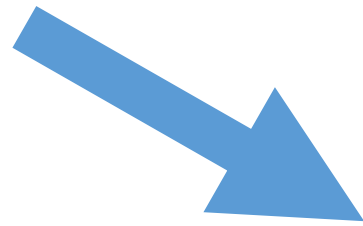
to cultivate development and delivery of software inside standardized software packages called [containers](#).

- Características y objetivos de Docker.

Docker. Introducción (1/2)

⇒ PseudoVirtualización

- ¿Qué es Docker?

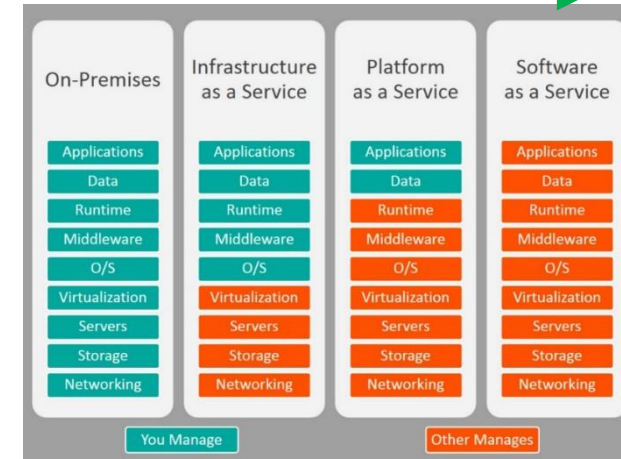


From Wikipedia

Docker is a collection of interoperating [software-as-a-service](#) and [platform-as-a-service](#) offerings that employ [operating-system-level virtualization](#)

to cultivate development and delivery of software inside standardized software packages called [containers](#).

- Características y objetivos de Docker.



Docker. Introducción (1/2)

⇒ SAAS / PAAS / IAAS
⇒ PseudoVirtualización

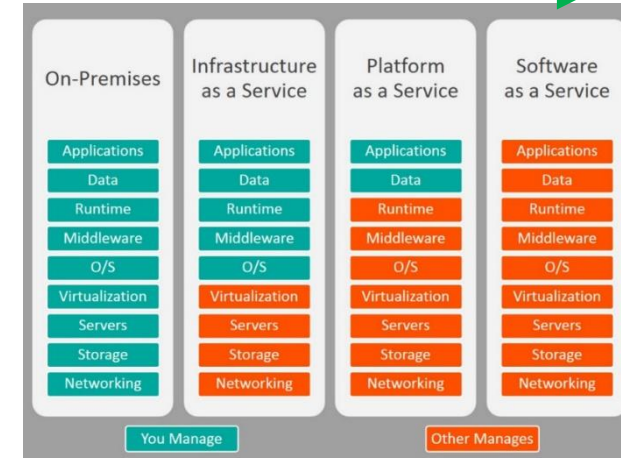
- ¿Qué es Docker?

From Wikipedia

Docker is a collection of interoperating [software-as-a-service](#) and [platform-as-a-service](#) offerings that employ [operating-system-level virtualization](#)

to cultivate development and delivery of software inside standardized software packages called [containers](#).

- Características y objetivos de Docker.



Docker. Introducción (1/2)

⇒ SAAS / PAAS / IAAS
⇒ PseudoVirtualización

• ¿Qué es Docker?

From Wikipedia

Docker is a collection of interoperating [software-as-a-service](#) and [platform-as-a-service](#) offerings that employ [operating-system-level virtualization](#)

to cultivate development and delivery of software inside standardized software packages called [containers](#).

• Características y objetivos de Docker.

Paquetes de software estandarizados

⇒ Esto quiere decir que son independientes de donde corran, corren igual.

⇒ Con esto se facilita inmensamente la tarea del deploy de las aplicaciones. Sean en local / remoto / producción /... ¡Dónde sea!

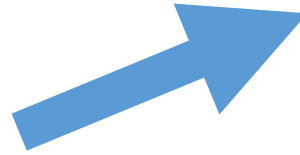
Docker. Introducción (2/2)

- ¿Por qué Docker?
 - Desarrolladores. Desarrollo sin tener que montar ningún entorno en local.
 - Empresas. Para entregas ágiles y automatizadas.
 - Curva de aprendizaje muuuy rápida. Gente con experiencia en DevOps / Linux lo puede coger en un par de días.
- Sobre todo:
 - Porque facilita infinitamente la faena de hacer un cluster con aplicaciones basadas en microservicios.

Un poco de historia

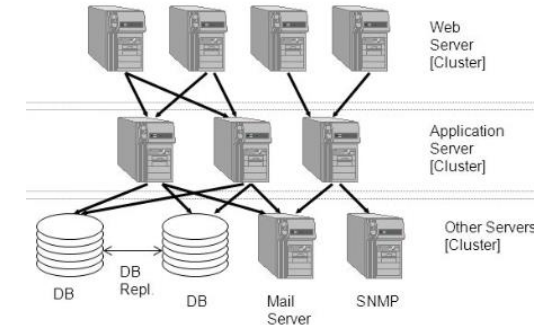
Un poco de historia

- En los tiempos de los dinosaurios...



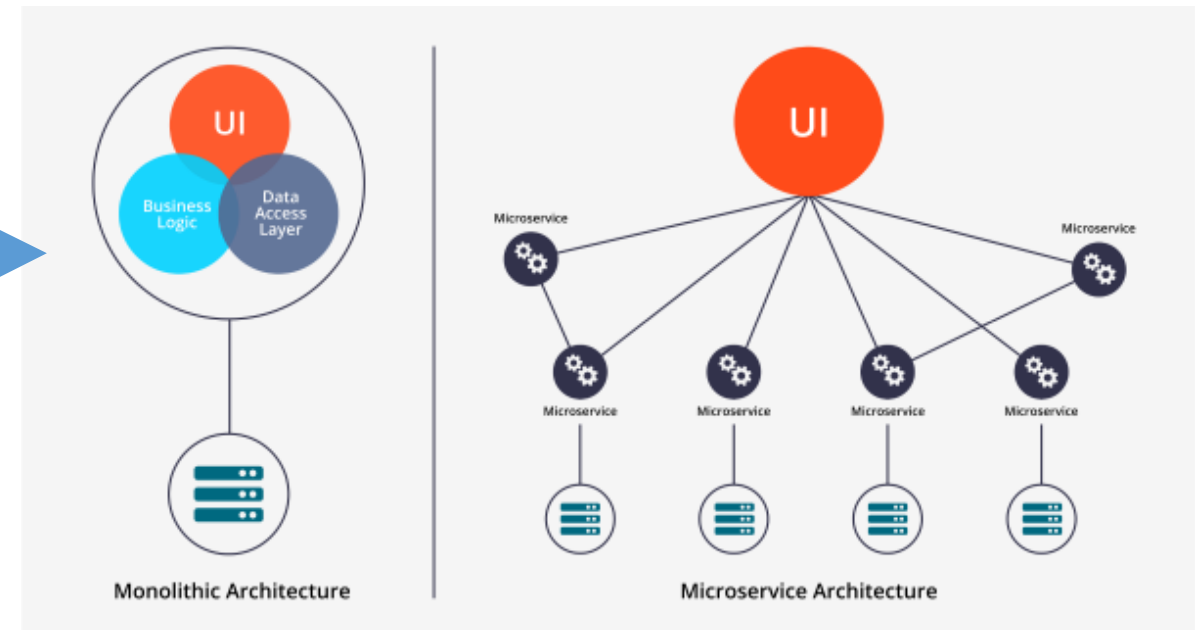
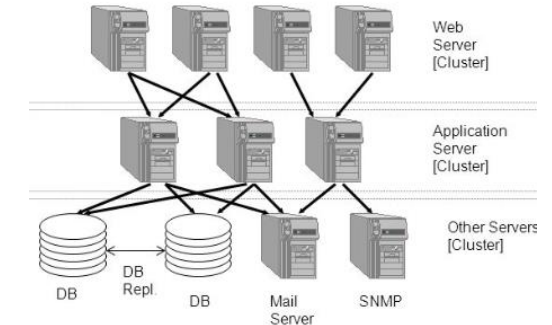
Un poco de historia

- En los tiempos de los dinosaurios...
- Clusters / Facebook / otros
- Más recientemente VMWare ... virtualización



Un poco de historia

- En los tiempos de los dinosaurios...
- Clusters / Facebook / otros
- Más recientemente VMWare ... virtualización
- Docker: Creación (2013)
- Implantación (2014-2016)
- AWS / Azure / Google Cloud
- Microservicios



Algunos datos de interés (de Docker)

- Fue creado por dotCloud, una empresa basada en Paris / Berlín en el 2013 por un yankee (de NYC) llamado Solomon Hykes
- En septiembre, siete meses después del lanzamiento, RedHat lo implanta. IBM dos meses más tarde.
- En 2014 AWS lo incluye en su Elastic Computing Cloud (Super Hito).
- En 2015 se convierte en el proyecto estrella de GitHub, el que más estrellas tiene ever ever.
- En 2016 se porta a Windows
- Actualmente está ampliamente implantado, sobre todo en nuevos builds. En el mundo startup es el estándar.



Docker. ¿Por qué ha triunfado? Pros y Contras

- Como SCRUM. Moda.
- Pros:
 - En un mundo ideal, es la panacea del pobre y sufrido administrador de sistemas.
 - Un clic en un botón y mi aplicación está desplegada y funcionando.
=> Cambio de máquina, no tengo que hacer nada más que bajar mi imagen y darle a run!
 - OpenSource.
 - Si se hace bien puede ahorrar muchísimas horas de trabajo en puesta a punto de nuevos entornos.
 - La gran razón de su triunfo: ¡¡¡¡Microservicios / Escalabilidad / Tolerancia a errores!!!!
- Contras:
 - Dificultad de Docker. No todo es tan bonito.
 - Tiene un coste de mantenimiento elevado. El, relativamente nuevo perfil de moda, llamado DevOps.
 - Muchas veces obviado como gratuito por el cliente/responsables. Esto no siempre es así.
 - Muchos sistemas interconectados (Jenkins / Nexus / Docker / GitLab ...). Basta que uno caiga para que no se pueda desplegar en integracion / preprod /...

Ley de Murphy => Suele pasar siempre el día de entrega.

=> Ejemplo -> JBoss 4 a JBoss 5 – Ejemplo de como de horrible puede ser mantener Docker con aplicaciones viejunas

Docker. ¿Por qué ha triunfado? Pros y Contras

En realidad ha triunfado porque es perfecto para el conjunto de nuevas tecnologías.

Docker. ¿Por qué ha triunfado? Pros y Contras

En realidad ha triunfado porque es perfecto para el conjunto de nuevas tecnologías.

DOCKER



REST / Microservicios / ... /
CDK / GitLab / ...

Docker. ¿Por qué ha triunfado? Pros y Contras

En realidad ha triunfado porque es perfecto para el conjunto de nuevas tecnologías.

DOCKER



REST / Microservicios / ... /
CDK / GitLab / ...

Y las nuevas tecnologías son perfectas para Docker.

REST / Microservicios / ... /
CDK / GitLab / ...



DOCKER

Docker. ¿Por qué ha triunfado? Pros y Contras

En realidad ha triunfado porque es perfecto para el conjunto de nuevas tecnologías.

DOCKER



REST / Microservicios / ... /
CDK / GitLab / ...

Y las nuevas tecnologías son perfectas para Docker.

REST / Microservicios / ... /
CDK / GitLab / ...



DOCKER

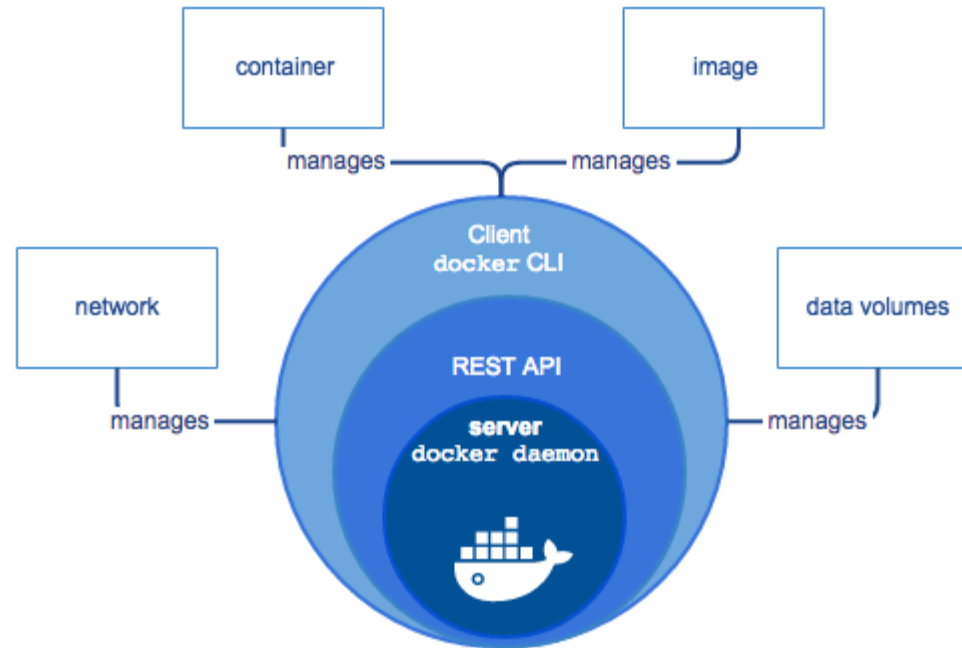


ES LA PAREJA IDEAL!!! AMOR VERDADERO!!!

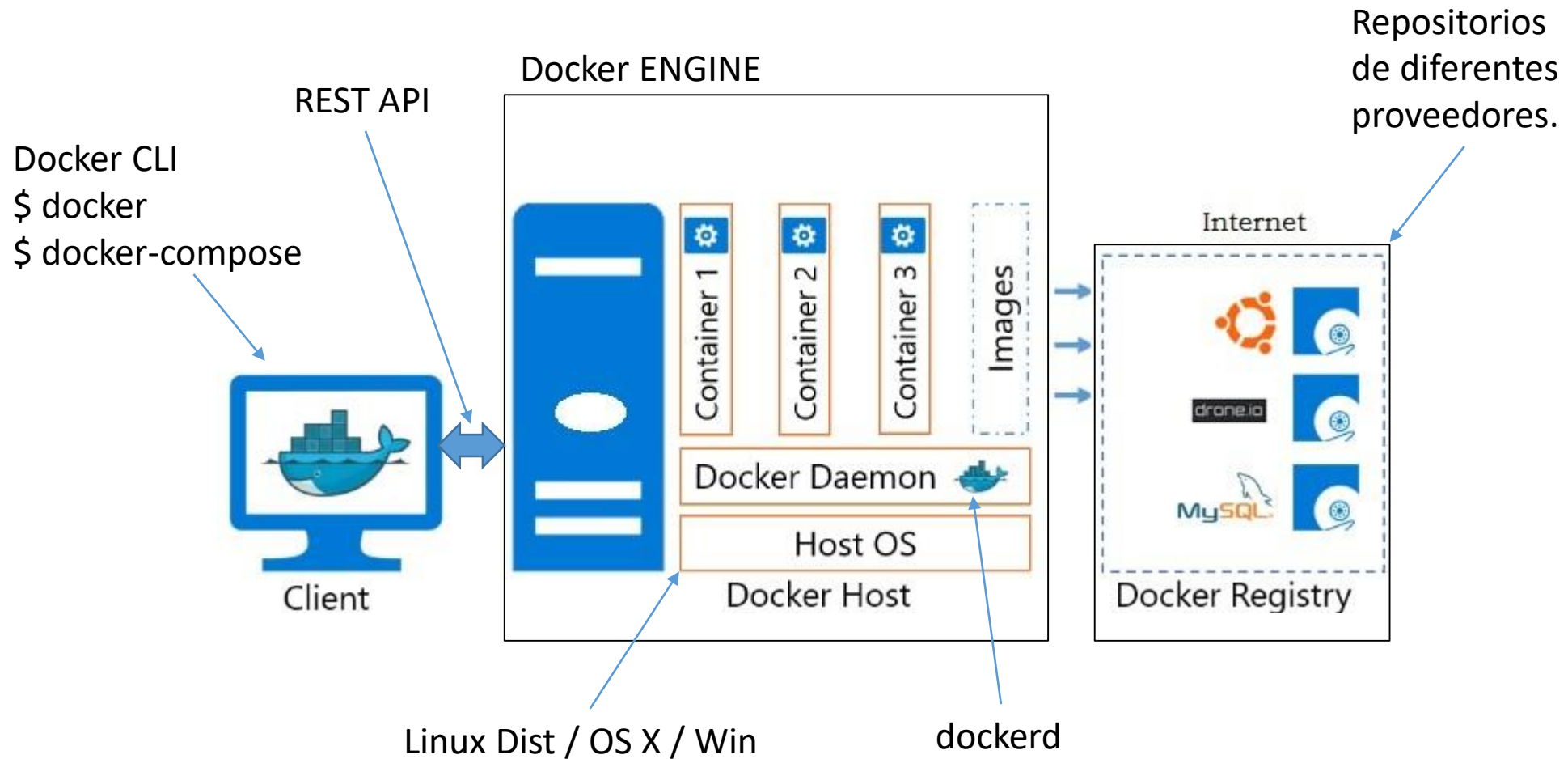


Arquitectura de Docker (1/3). Como funciona

- Diagrama de la arquitectura simple (Oficial) del servicio Linux y el cliente (o clientes).



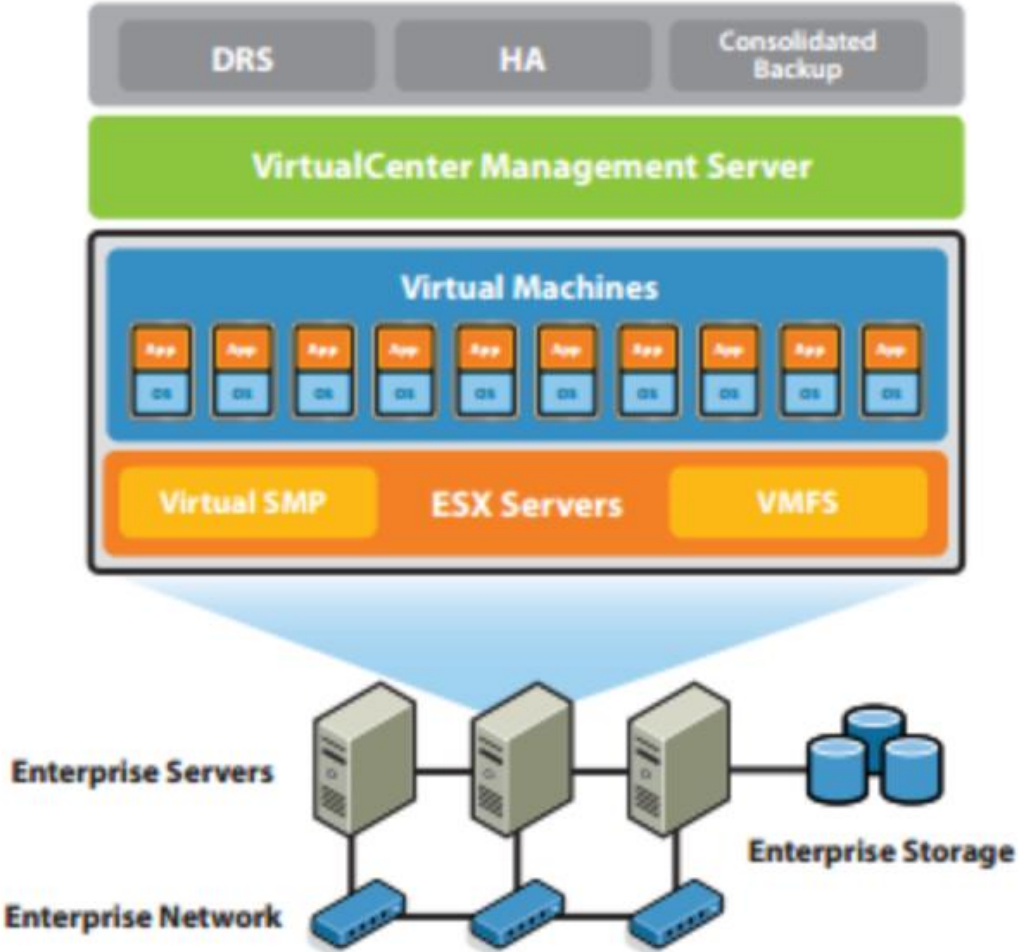
Arquitectura de Docker (2/3). Otra visión



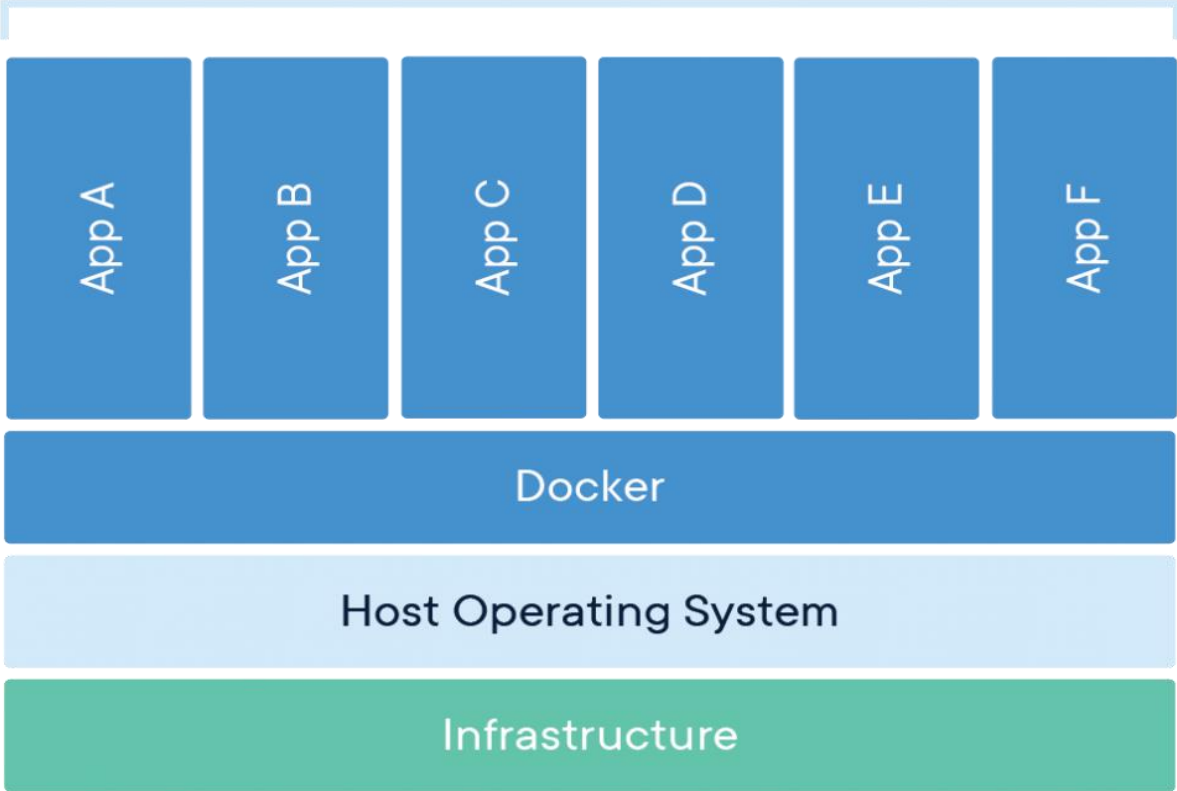
VMWARE

Vs

DOCKER



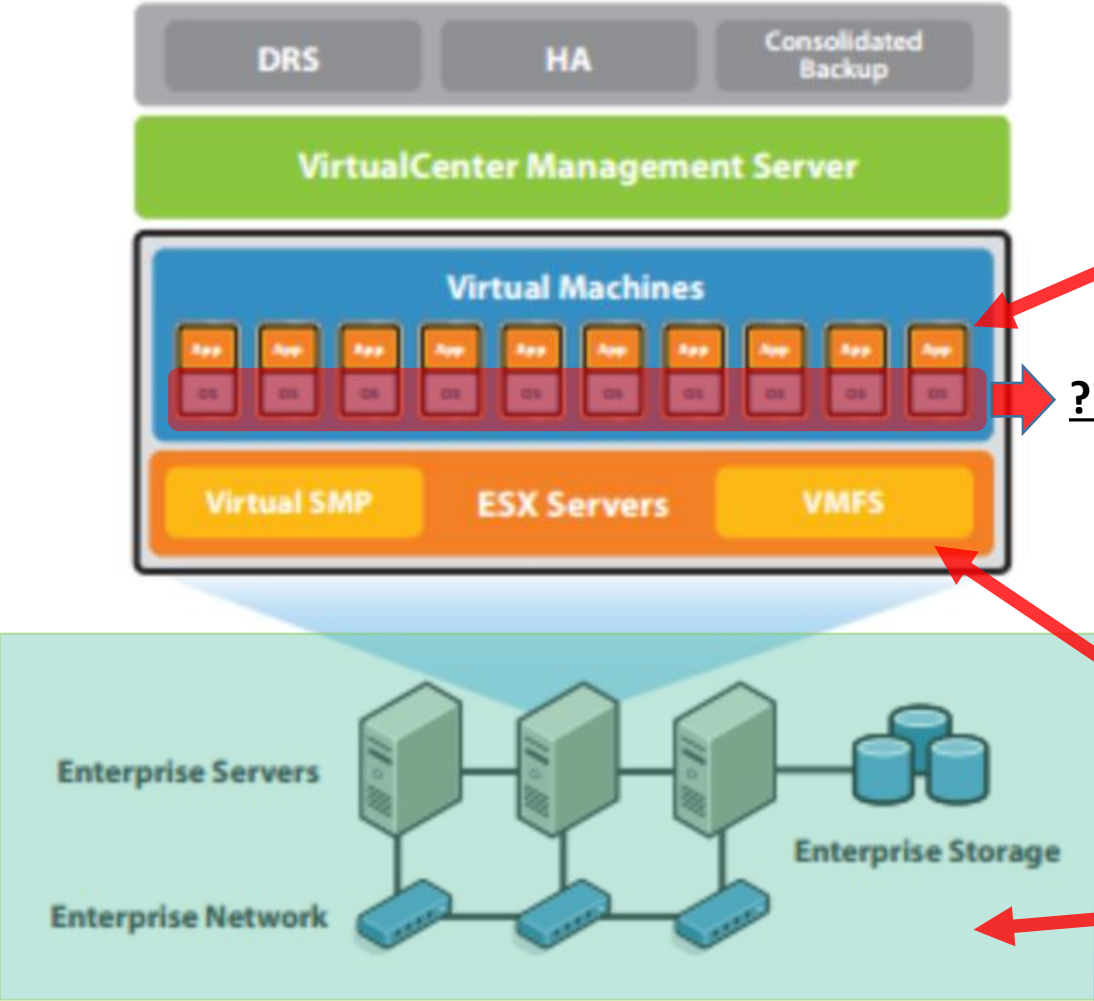
Containerized Applications



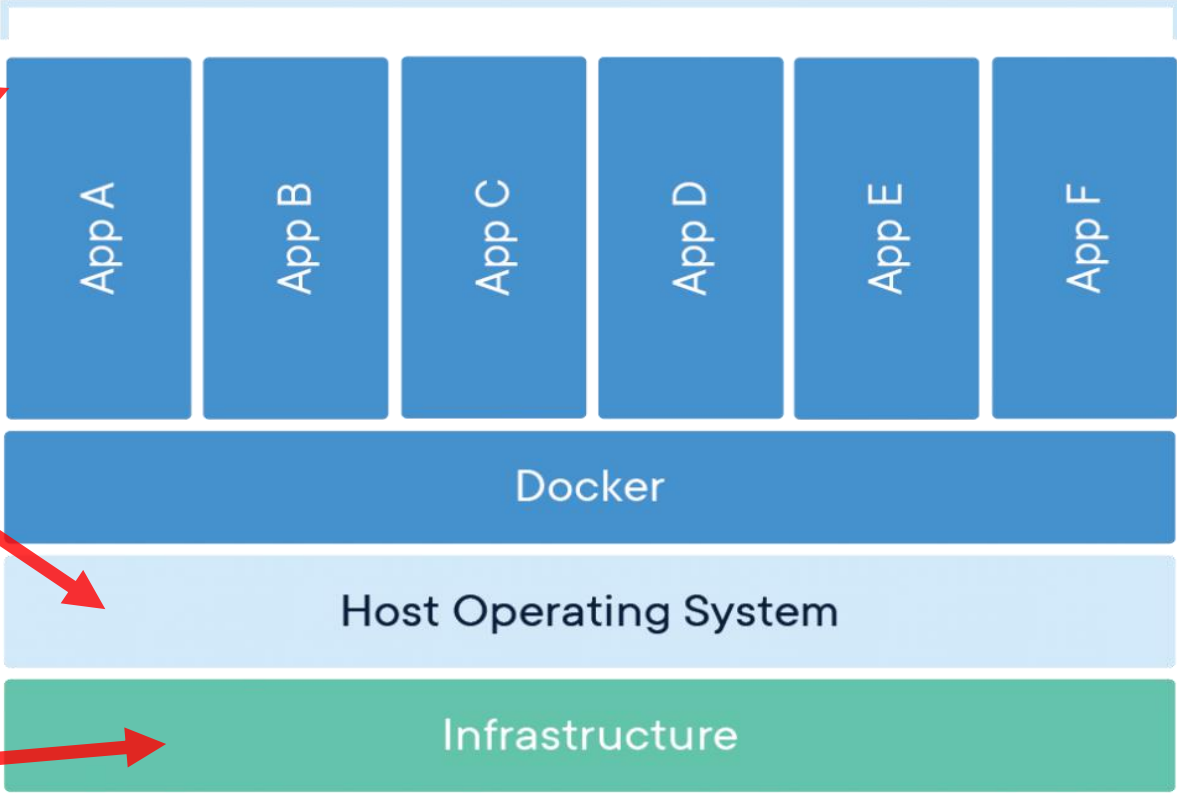
VMWARE

Vs

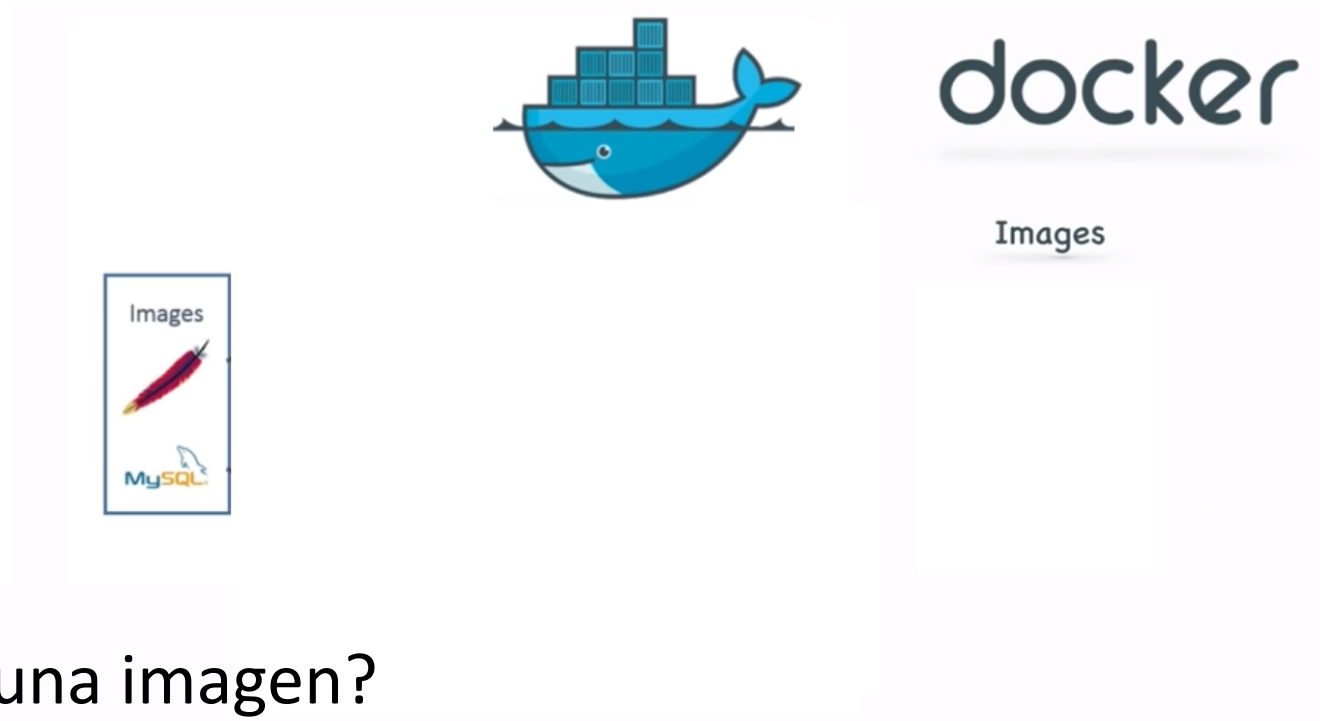
DOCKER



Containerized Applications



Conceptos (1/8). Images (1/3)



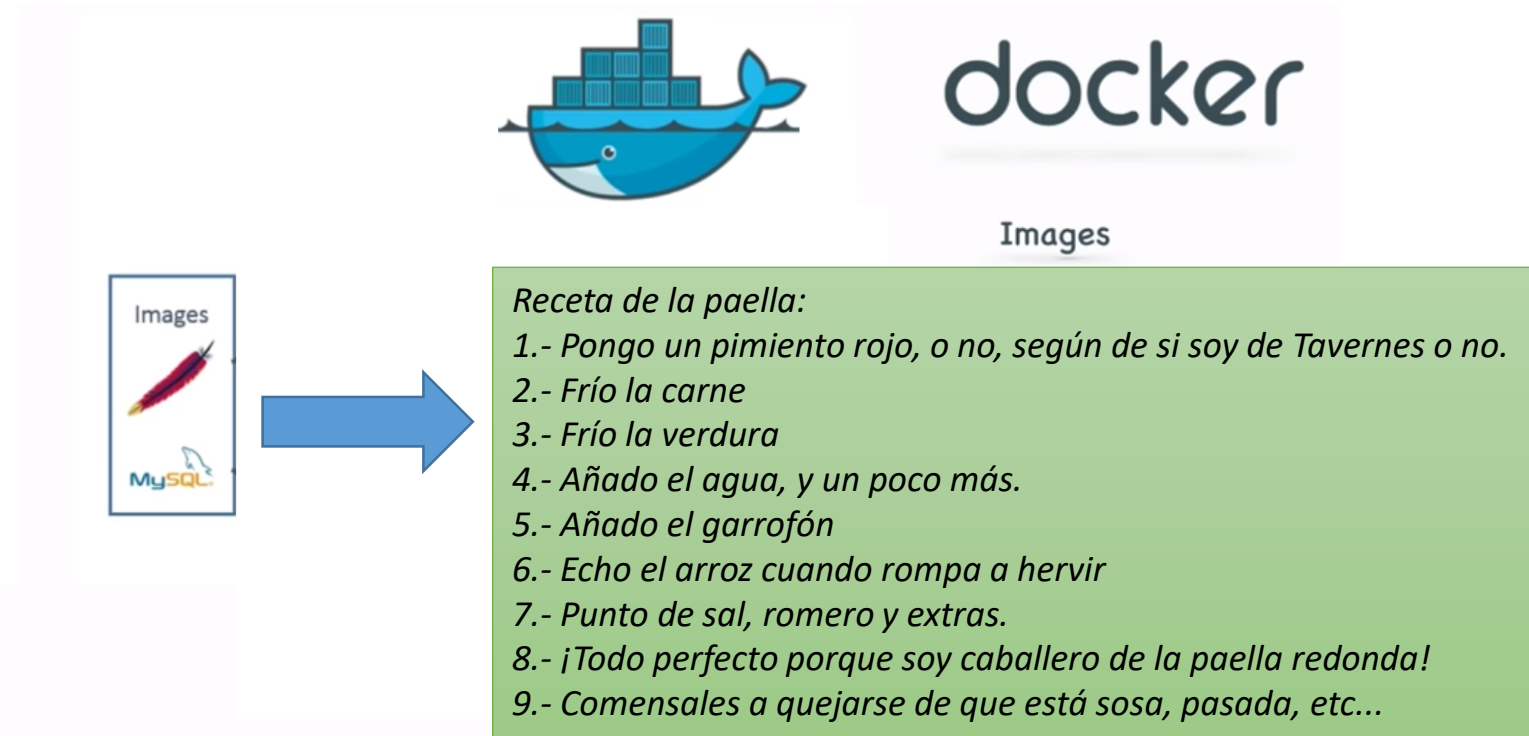
- ¿Qué es una imagen?

Dicho de una manera simple

⇒ *Receta de cocinado para construir una aplicación*

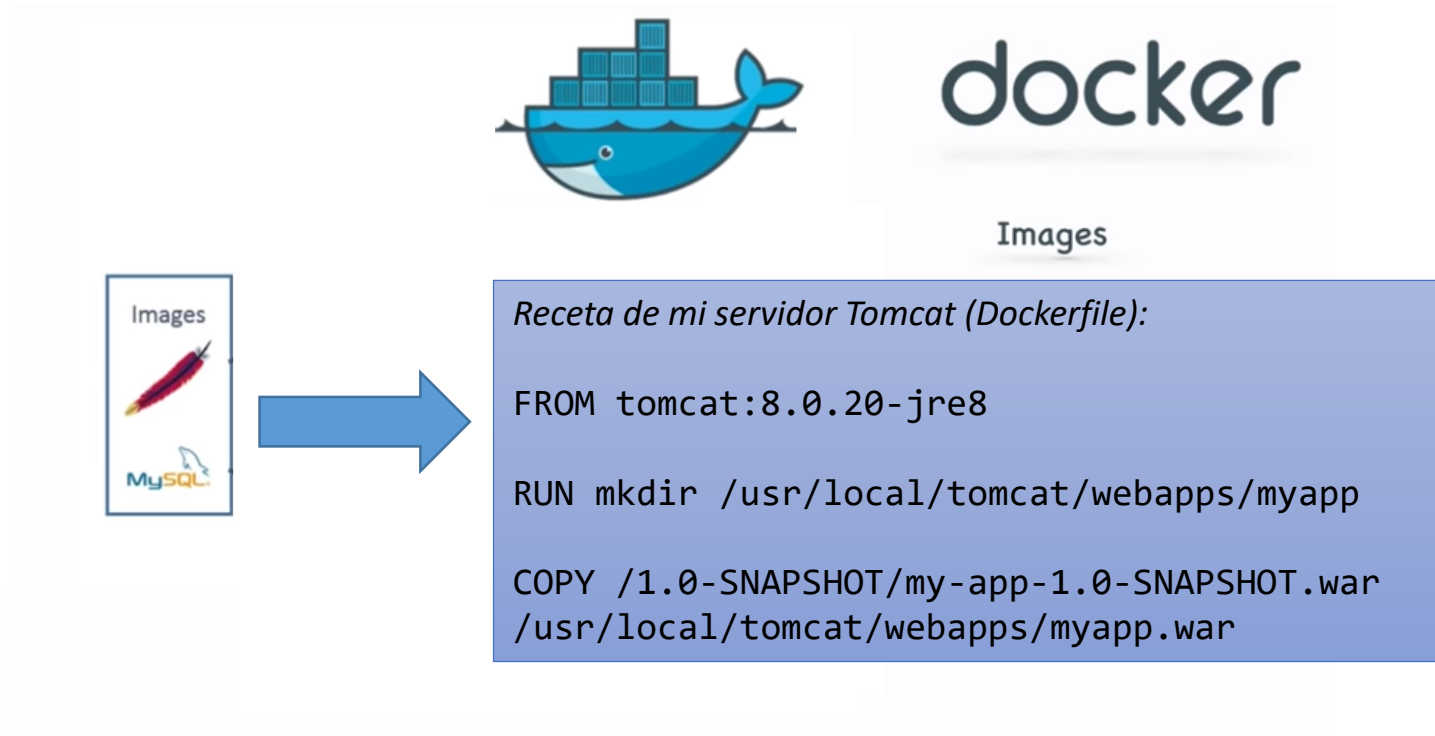
⇒ *Para los del CS → Una PTI.*

Conceptos (1/8). Images (2/3)



- Receta de cocinado para construir una aplicación (Paella)

Conceptos (1/8). Images (3/3)



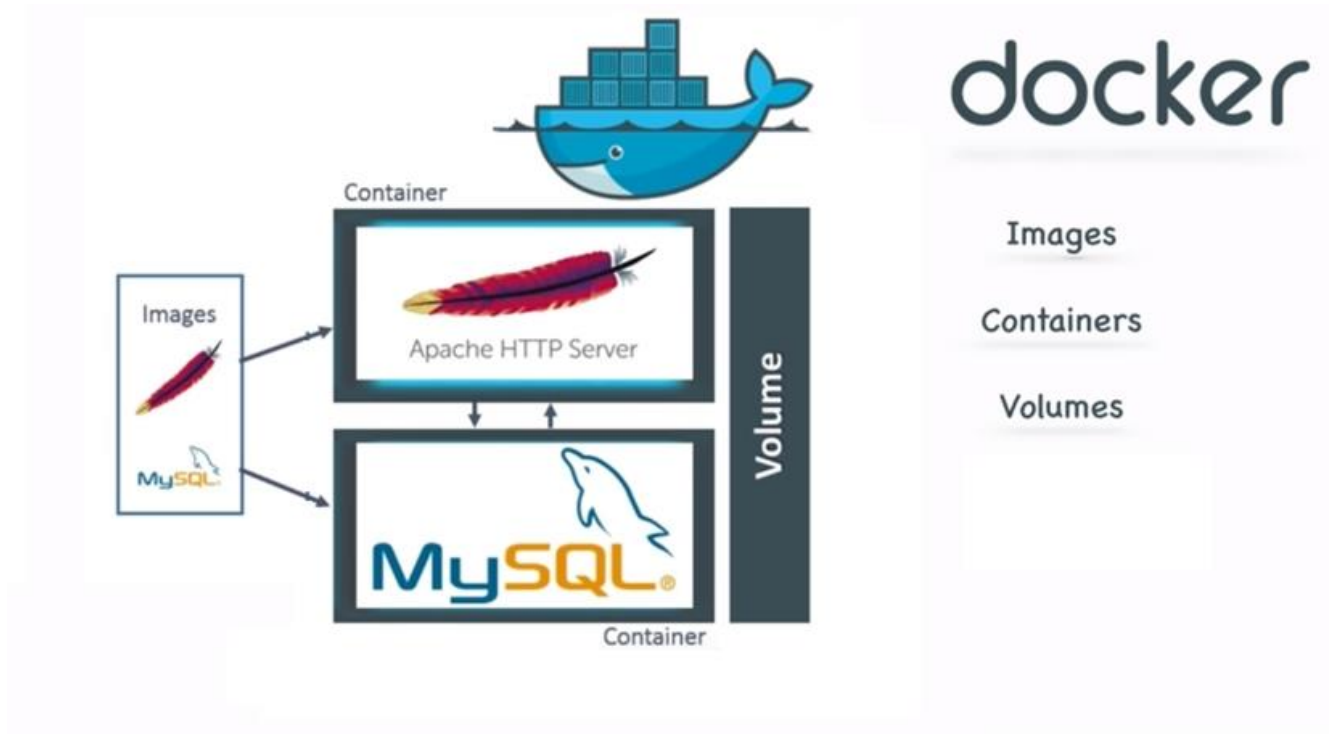
- Receta de cocinado para construir una aplicación (Servidor Tomcat)
- Con más detalle más tarde

Conceptos (2/8). Containers



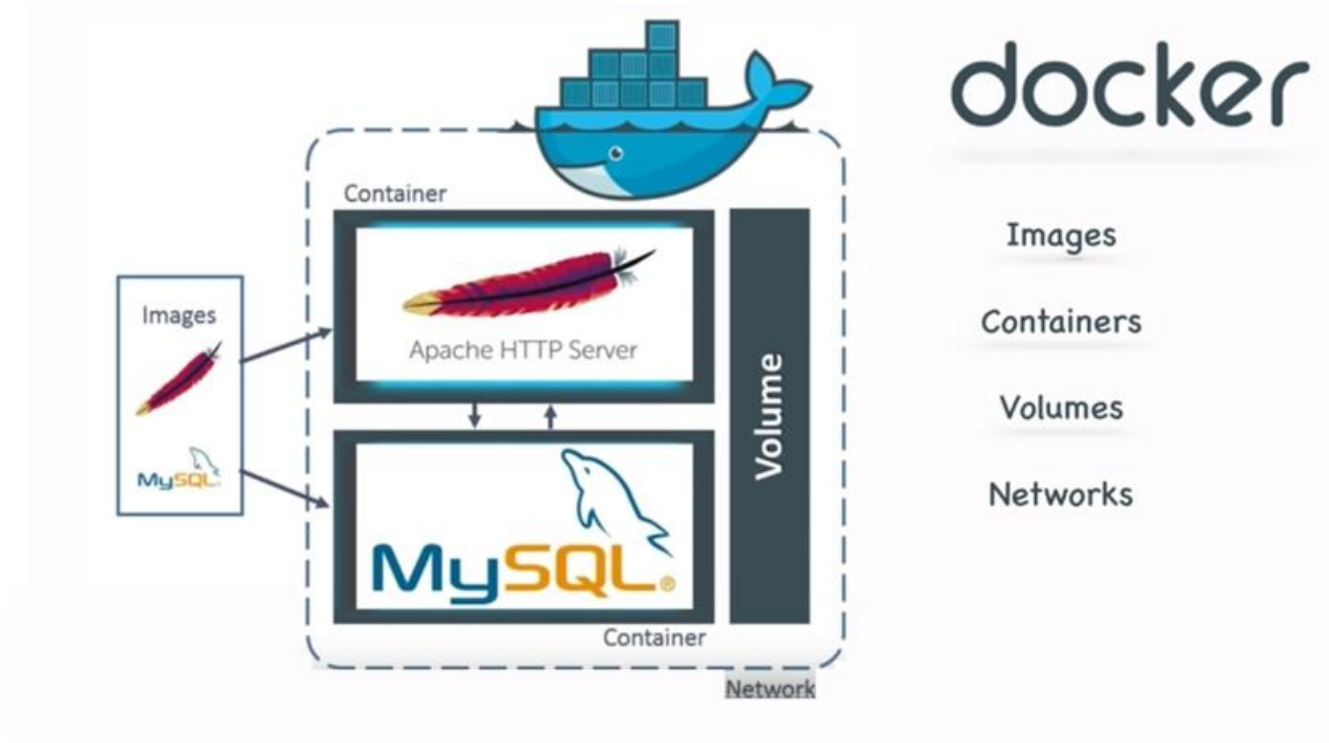
- Container – Instancias de una imagen, es lo que más se usa trabajando con docker. Los contenedores no son persistentes.
- Analogía: Image es una class, Container es un objeto, una instancia.
- Hay infinitas políticas de deploy de los contenedores.
 - Alta disponibilidad,...
 - Escalabilidad,...
 - Según necesidad de cada aplicación.

Conceptos (3/8). Volúmenes



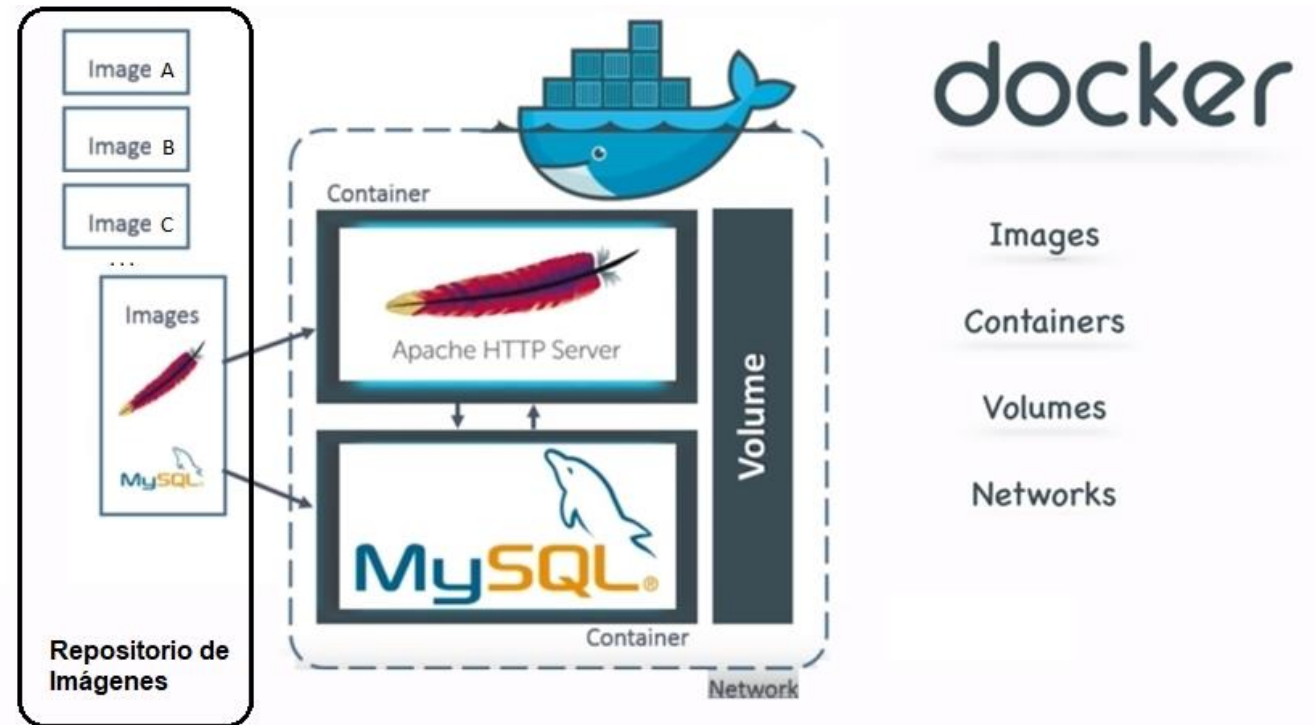
- Discos duros virtuales donde se guardan los datos del contenedor.
- En general están alojados en la misma máquina que el contenedor pero se pueden crear en red, en local, distribuidos, replicados, etc.
- Por defecto no son persistentes => Muerte del contenedor = Muerte de los datos
- Se pueden hacer persistentes => BDDs, directorios compartidos, ...

Conceptos (4/8). Networks



- Al igual que los volúmenes, las redes son redes virtuales. (Privadas)
- Son para que el contenedor pueda verse con otros contenedores que estén en la misma red.
- En resumen: Es una abstracción de una red de verdad.

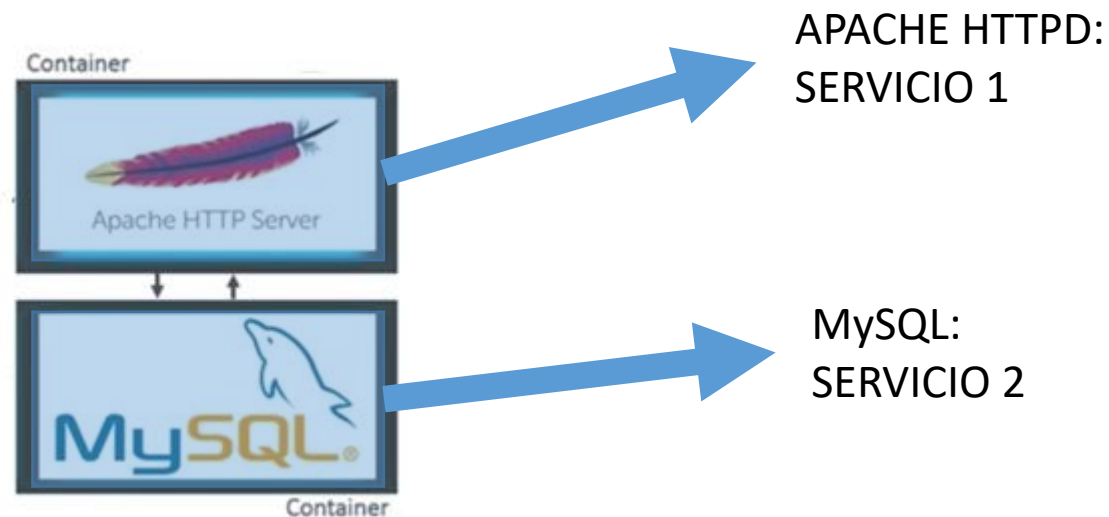
Conceptos (5/8). Repositorio de imágenes



- Repositorio de imágenes más común es: Docker Hub (<https://hub.docker.com/>)
- Similar a GitHub. Puedes instalártelo en local, hacer tu propio repositorio.
- Imagina que solo necesitas una BDD MySQL sin nada especial, nada más fácil que desplegar un contenedor desde la imagen que se aloja en el Docker Hub oficial de MySQL.
- En resumen: Un repositorio no es más que el libro de recetas de la abuela.

Conceptos (6/8). Services

- Es lo que su nombre indica: Un servicio
- Ejs: Una BDD, Un servidor de apps (tomcat), un NAS, un interfaz de Docker, etc etc etc. Es un componente que cumple una función.
- Una imagen, por lo general, tiene un servicio. Que se instancia en un container.

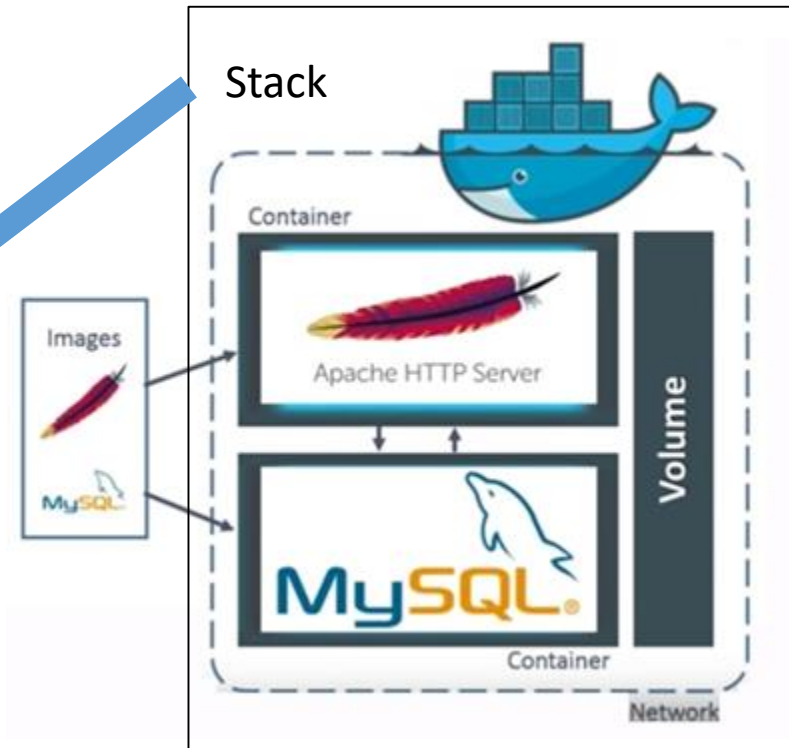


Conceptos (7/8). Stacks

- Básicamente, y sin explayarse mucho, una pila (stack) es un conjunto de servicios, volúmenes, networks,...
- Se definen, generalmente, con un fichero YAML: docker-compose.yml

docker-compose.yml

```
version: '3.3'
services:
  application:
    image: myApp-app-frontend:1.0
    depends_on:
      - database
    build: ./myApp-frontend
    ports:
      - "8380:8380"
    networks:
      - my_network
  database:
    image: myApp_database:1.0
    build: ./myApp-database
    ports:
      - "5432:5432"
    volumes:
      - myApp_vol_database:/var/lib/pgsql/9.3/data
    networks:
      - my_network
networks:
  my_network:
    driver: overlay
    attachable: true
volumes:
  myApp_vol_database:
```



Conceptos (8/8). Swarm

From Wikipedia

Docker Swarm or simply **Swarm** is an open-source container orchestration platform and is the native clustering engine for and by **Docker**. Any software, services, or tools that run with **Docker** containers run equally well in **Swarm**. ... **Swarm** turns a pool of **Docker** hosts into a virtual, single host.

- En una sola palabra: Cluster
- El Swarm es el que “orquestará” todos los servicios según las necesidades del Stack (del conjunto de servicios) y cómo estén configurados.
- Orchestration => ¿qué es?
- Docker Swarm vs Kubernetes
- No entro en mucho detalle porque Kubernetes es el estándar en orquestación. Sobre todo por la adopción de AWS.
 - A no ser que pase algo raro lo más normal es que Docker Swarm acabe por quedar en un segundo plano.

Docker. Instalación

- Muy fácil en Linux / Mac OS X
- En Windows fácil también... aunque...
 - Consumo bestial de recursos en Windows -> ¿Por qué razón?
- Instalación en Unix / Windows / OS X -> Fácil, lo más difícil es configurar el maravilloso proxy, por estar en un entorno corporativo como Sopra.
- Instalación ya hecha. Vbox.

COMANDOS INSTALACIÓN (en UBUNTU)

INSTALL WITH THE REPOSITORY

```
sudo apt-get update
```

```
sudo apt-get install apt-transport-https ca-certificates curl gnupg-agent software-properties-common
```

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

```
sudo apt-key fingerprint 0EBFCD88
```

```
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
```

INSTALL DOCKER en Ubuntu a partir del repositorio de paquetes

```
sudo apt-get update
```

```
sudo apt-get install docker-ce docker-ce-cli containerd.io
```

vim ~/.docker/config.json

```
{
  "proxies":
  {
    "default":
    {
      "httpProxy": "http://alca.proxy.corp.sopra:8080",
      "httpsProxy": "https://alca.proxy.corp.sopra:8080",
      "noProxy": "localhost,127.0.0.1"
    }
  }
}
```

-- INSTALAR EL COMPOSE

```
vim ~/.curlrc
proxy = alca.proxy.corp.sopra:8080
```

```
sudo curl -L "https://github.com/docker/compose/releases/download/1.24.0/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
sudo mv /usr/local/bin/docker-compose /usr/bin/docker-compose
sudo chmod +x /usr/bin/docker-compose
```

sudo vim /etc/environment

```
HTTP_PROXY=http://alca.proxy.corp.sopra:8080
HTTPS_PROXY=http://alca.proxy.corp.sopra:8080
NO_PROXY=localhost,127.0.0.1
http_proxy=http://alca.proxy.corp.sopra:8080
https_proxy=http://alca.proxy.corp.sopra:8080
no_proxy=localhost,127.0.0.1
```

Ejercicio simple. Docker. Python y PHP.
REST (Representational state transfer)

```
-----  
docker run hello-world  
-----  
  
hostname ==> tgp-VirtualBox  
docker run --interactive --tty ubuntu bash ==> Arranca el contenedor (de manera interactiva)  
  
docker container ls          ==> Nada porque no hay contenedores rulando  
docker container ls --all    ==> Salen los Exited(X) también  
docker container prune      ==> Limpia los contenedores que estén parados  
docker container ls --all    ==> Ya no sale nada  
  
-----  
  
docker run --detach --tty ubuntu bash  
docker container ls          ==> Se ve el contendor (rulando como detach)  
docker container stop <id_container> ==> Para el contenedor arrancado antes  
-----
```

Unos comandos para iniciarse con Docker, una vez instalado.

Hello world, se ejecuta y sale.

Mostrar contenedores.

Levantar (crear y ejecutar) un contenedor.

```
-----  
--          TUTORIAL 1  
-----
```

```
mkdir tutorial  
cd tutorial  
mkdir product  
cd product
```

```
vim api.py
```

```
-----  
FICHERO api.py (Copiar / pegar, cuidado con la indentación)  
-----
```

```
# Product service
```

```
from flask import Flask  
from flask_restful import Resource, Api
```

```
app = Flask(__name__)  
api = Api(app)
```

```
class Product(Resource):  
    def get(self):  
        return {  
            'products': ['Ice cream',  
                        'Chocolate',  
                        'Fruit']  
        }
```

```
api.add_resource(Product, '/')
```

```
if __name__ == '__main__':  
    app.run(host='0.0.0.0', port=80, debug=True)
```

```
-----  
vim requirements.txt  
-----
```

```
FICHERO requirements.txt  
-----
```

```
Flask==0.12  
flask_restful==0.3.5
```

Primeros pasos del tutorial.

- Crear una estructura de directorios y sus ficheros

/tutorial/

product/

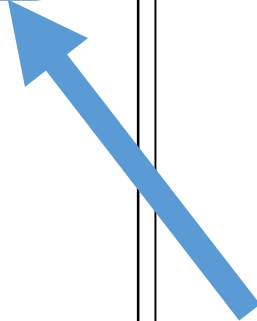
api.py

requirements.txt

- Copiar / Pegar el contenido en cada uno de los ficheros.

- Básicamente es un controlador REST en Python.

Docker container ls



- Crear el Dockerfile y el docker-compose.yml
/tutorial/
 - product/
 - api.py
 - requirements.txt
- Dockerfile**
- /docker-compose.yml**
- Copiar / Pegar el contenido en los ficheros.
- Hacer docker-compose up
 - Bajará las imágenes padre del repo de Docker
 - Construirá la imagen de Python a partir del Dockerfile
 - Instanciará un contenedor de la imagen y lo ejecutará.
 - Ir a <http://localhost:5001> y veremos nuestro JSON con la respuesta del controlador.
 - Hacer Ctrl+C para pararlo, volver a ejecutar con detach. Para que lo ejecute en un hilo aparte.
 - Hacer un docker container ls para ver el contenedor funcionando.

```
-----  
--          TUTORIAL 2  
-----
```

```
mkdir website  
cd website  
vim index.php  
-----
```

```
<html>  
  <head>  
    <title>My Shop</title>  
  </head>  
  <body>  
    <h1>Welcome to my shop</h1>  
    <ul>  
      <?php  
        $json = file_get_contents('http://product-service');  
        $obj = json_decode($json);  
  
        $products = $obj->products;  
        foreach ($products as $product) {  
            echo "<li>$product</li>";  
        }  
      <?>  
    </ul>  
  </body>  
</html>  
-----
```

- Crear una nueva carpeta para nuestro front en PHP.
- Crear el fichero index.php y copiar pegar el contenido.

/tutorial/

product/

api.py

requirements.txt

Dockerfile

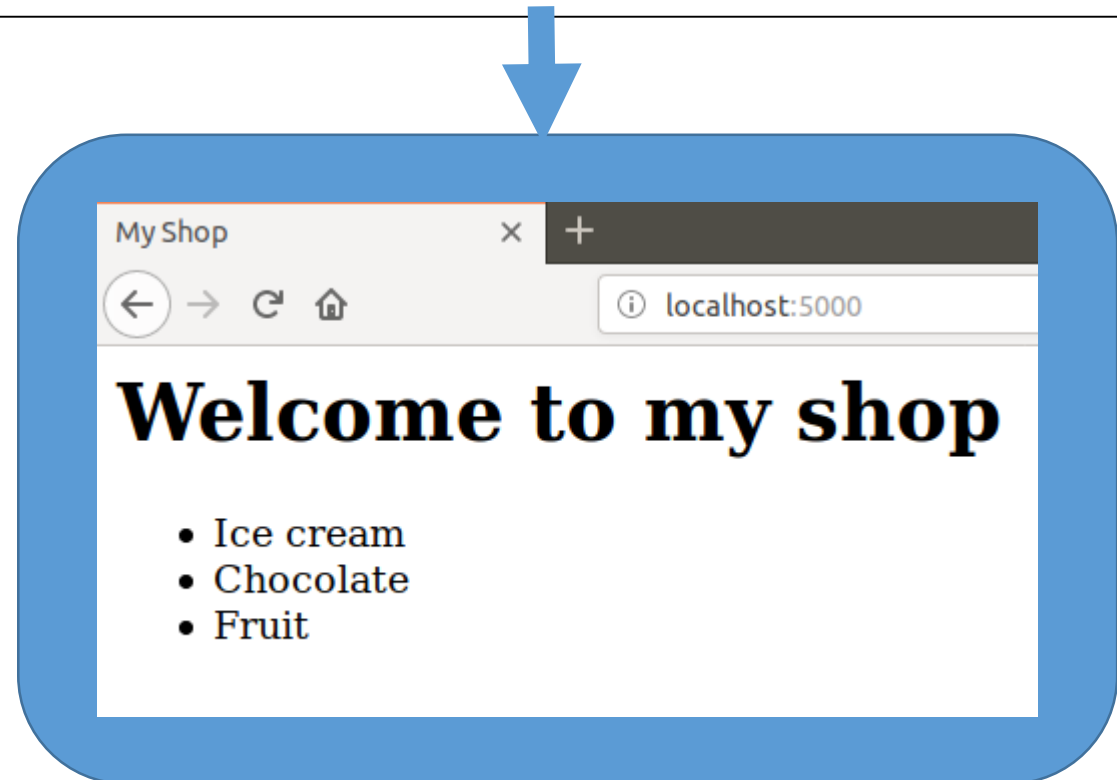
/docker-compose.yml

/website/

index.php

```
-----  
cd ..  
vim docker-compose.yml  
-----  
version: '3'  
  
services:  
  product-service:  
    build: ./product  
    volumes:  
      - ./product:/usr/src/app  
    ports:  
      - 5001:80  
  
  website:  
    image: php:apache  
    volumes:  
      - ./website:/var/www/html  
    ports:  
      - 5000:80  
    depends_on:  
      - product-service  
-----
```

- Editar el docker-compose.yml creado antes.
- Añadir nuestro nuevo servicio (website) a nuestra stack.
- Volver a lanzar el stack con: docker-compose up
- Ahora al ir a <http://localhost:5000> veremos nuestra página web que hace una petición REST al controlador Python y muestra los resultados en una lista.



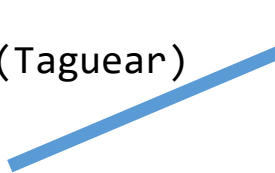
Docker (v1.18). Comandos básicos (1/2)

Servicio (Daemon) (CentOS/Linux)

```
sudo systemctl daemon-reload  
sudo systemctl start/stop/restart docker
```

Comandos súper básicos

```
docker ps / docker container ls / docker image ls  
docker volume ls  
docker volumen rm <id_volumen>  
docker network ls  
docker pull  
docker build  
    => docker build -t my_container (Taguear)  
docker run <imagen>  
    => docker run my_image -it bash  
docker logs <id_container>  
docker rm  
docker rm i  
docker stop  
docker --help
```



Comandos de parada masiva

```
kill all running containers with  
docker kill $(docker ps -q)
```

```
delete all stopped containers with  
docker rm $(docker ps -a -q)
```

```
delete all images with  
docker rmi $(docker images -q)
```

```
docker exec -it <id_container> bash
```

Compose / Stacks / Swarm

```
docker-compose up / YAML file / ...  
(hay que instalarlo aparte)  
docker swarm init/join/leave  
docker swarm --help
```

Docker. Comandos básicos (2/2) => Portainer (1/2)

- Es básicamente un GUI para manejar Docker.
- Exactamente lo mismo que los comandos de consola, pero más amigable para gente que quiera tener vida social. Se comunica mediante el interfaz REST de la misma manera que lo hace Docker CLI
- Una vez instalado Docker, se instala haciendo una instancia de la imagen de Portainer. Algo tan simple como:

```
$ docker pull portainer/portainer  
$ docker run -d -p 9000:9000 -v /var/run/docker.sock:/var/run/docker.sock portainer/portainer
```

- Repositorio (libro de recetas) de Portainer:

<https://hub.docker.com/r/portainer/portainer/>

No es necesario añadirlo ni nada porque como está alojado en .docker.com se puede descargar sin más.

Docker. Comandos básicos (2/2) => Portainer

Doktor. (A mi me parece flojo)

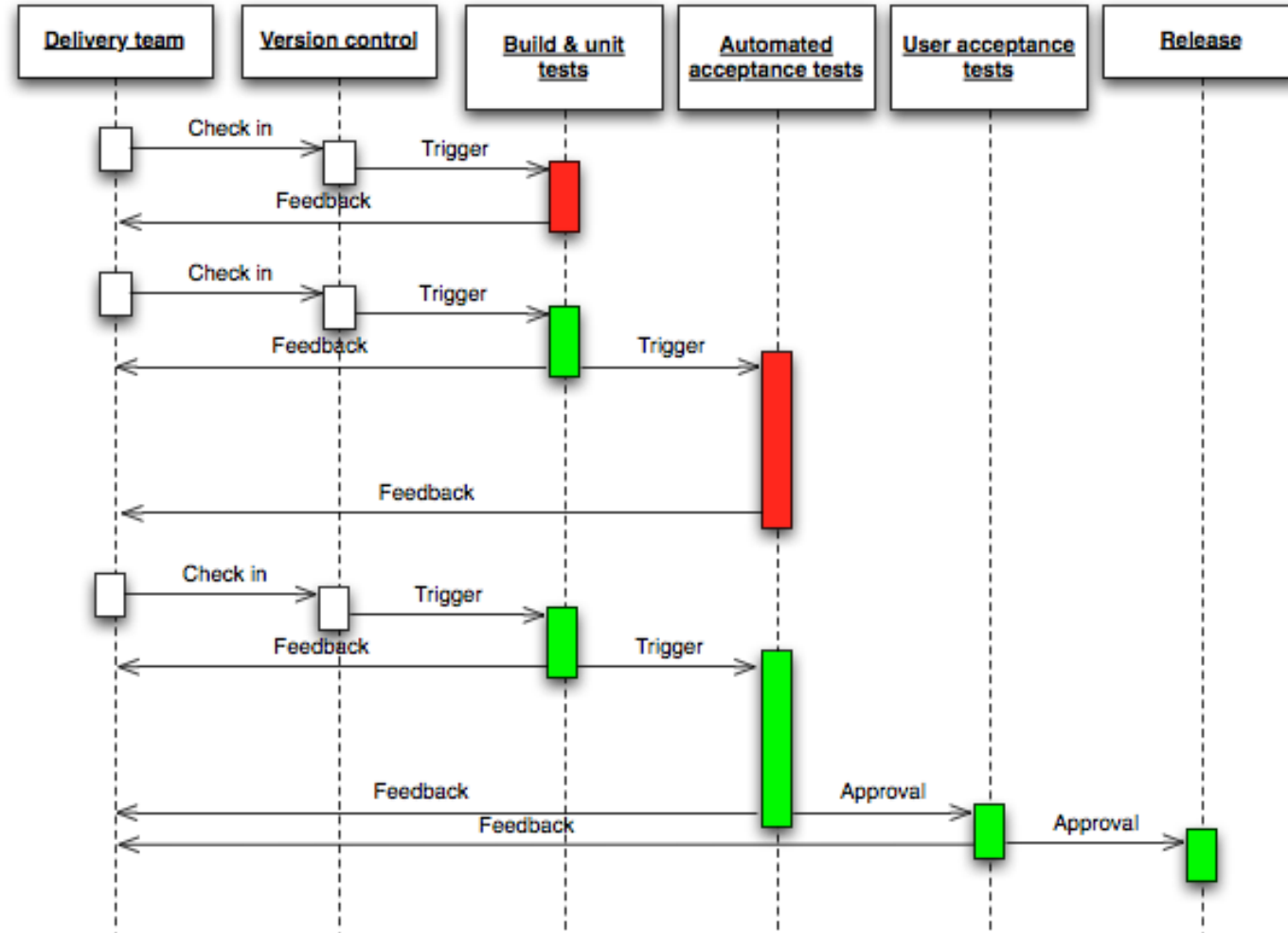
The screenshot shows the Docker management interface (Doktor) with a dark header. The main content area displays a group named "GroupCCCD" created on Nov 18, 2014 by Yvonnick Esnault. Below the group name are buttons for "Deploy a service", "Edit this group", and "Delete Group". The "Services" section shows a log for the container "/GroupCCCD-soapower" with various system messages. Below the logs is a table of services.

Name	Service - Image	Status	Commands	Daemon	CPU	Mem	Parameter	Variables
/GroupCCCD-phabricator	yesnault/docker-phabricator	Stopped		UP details			Hostname:GroupCCCD-phabricator-NodeA AAA:BBB	VAR_GROUP2:qsf
/GroupCCCD-soapower ffb123de3e53	soapower yesnault/docker-soapower:215	Running	Top Logs echo uname env	UP details	13	18 372 Mo	CpuShares:1000 Memory:2g Hostname:GroupCCCD-soapower-NodeA AAA:BBB	AA:val VAR_GROUP2:qsf

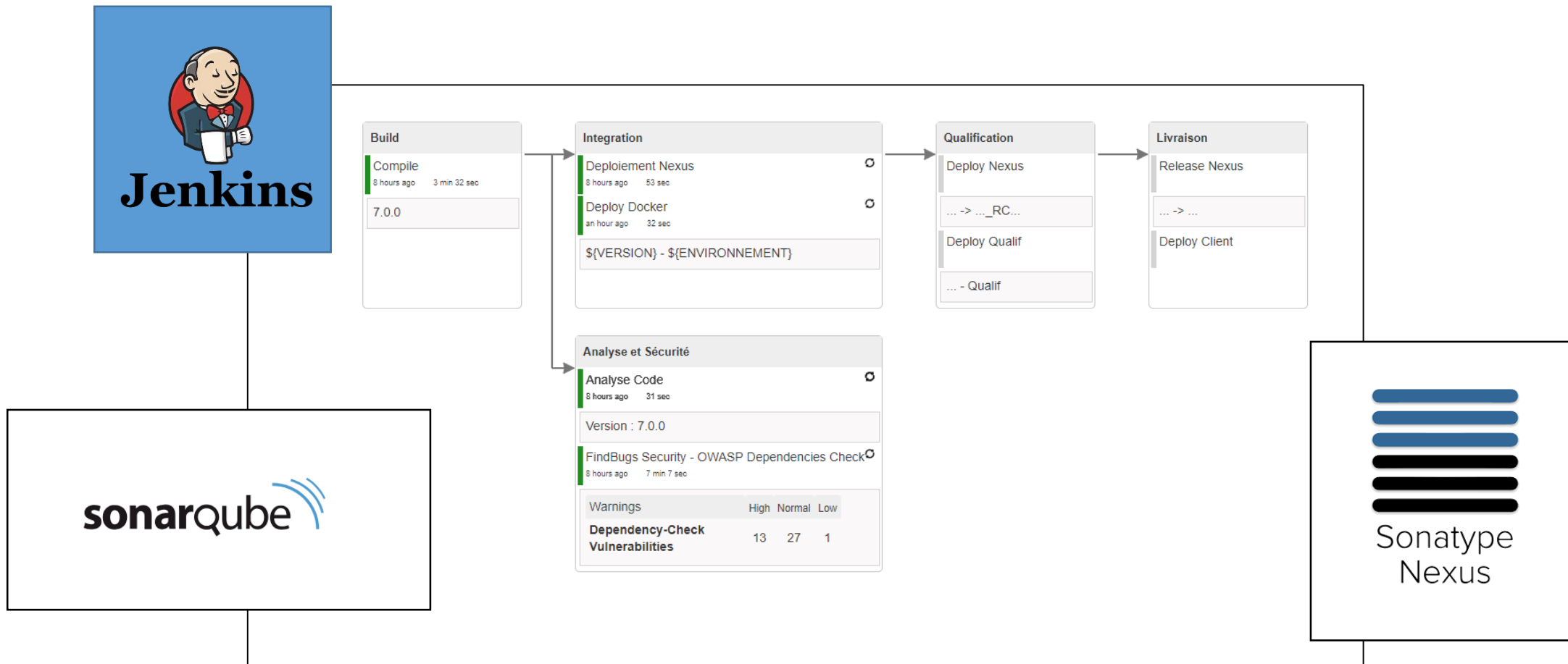
Portainer. (Tomás' Seal of Approval)

The screenshot shows the Portainer dashboard with a dark sidebar. The main content area displays the "Dashboard" section with "Endpoint summary". The "Cluster information" section shows "Nodes in the cluster" as 4. Below this are several statistics cards: "14 Stacks", "26 Services", "48 Containers" (with 27 running and 19 stopped), "37 Images" (with 17.8 CB), and "10 Volumes". The bottom right corner shows "54 Networks".

Docker + CDK. Caso de estudio genérico.



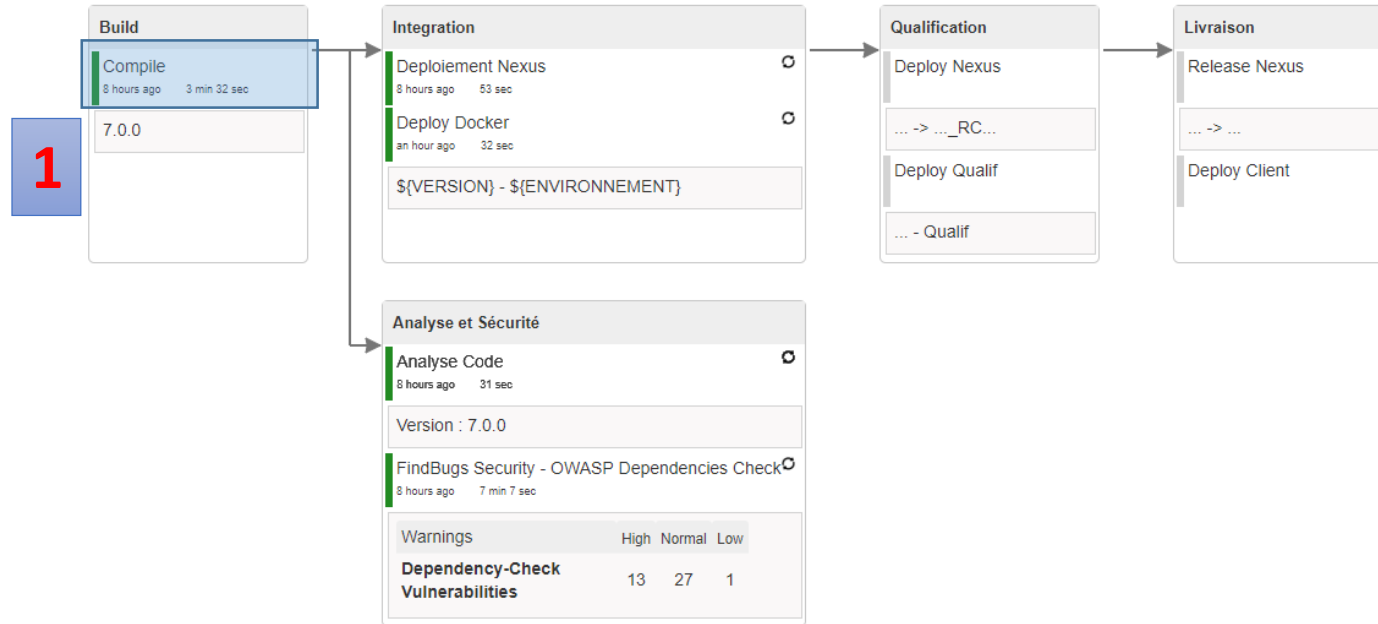
Docker + CDK. Caso de estudio genérico.



Vamos a ver de manera teórica cómo funciona un CDK con:

- Jenkins
- SVN / GitLab
- Docker
- Nexus
- SonarQube

Docker + CDK. Caso de estudio genérico.



Docker + CDK. Caso de estudio genérico.

INPUT: **`\${VERSION}`**



1

```
svn update / git pull (master)  
mvn clean install
```

Build

Compile
8 hours ago 3 min 32 sec

7.0.0

Integration

Deploiement Nexus
8 hours ago 53 sec

Deploy Docker
an hour ago 32 sec

`\${VERSION}` - `\${ENVIRONNEMENT}`

Qualification

Deploy Nexus

... -> ..._RC...

Deploy Qualif

... - Qualif

Livraison

Release Nexus

... -> ...

Deploy Client

Analyse et Sécurité

Analyse Code
8 hours ago 31 sec

Version : 7.0.0

FindBugs Security - OWASP Dependencies Check
8 hours ago 7 min 7 sec

Warnings	High	Normal	Low
Dependency-Check Vulnerabilities	13	27	1

INPUT: **`\${VERSION}`**

Generalmente sacada directamente desde GitLab o SVN.

También se puede meter a mano

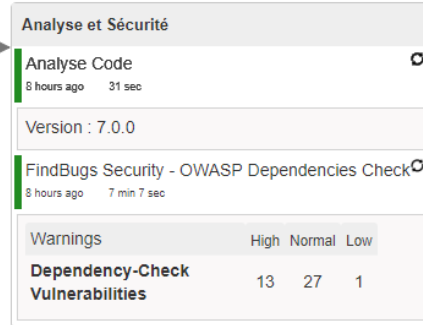
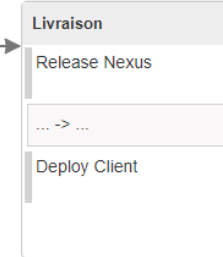
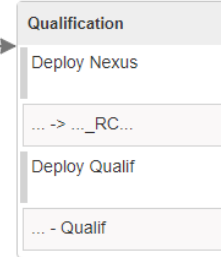
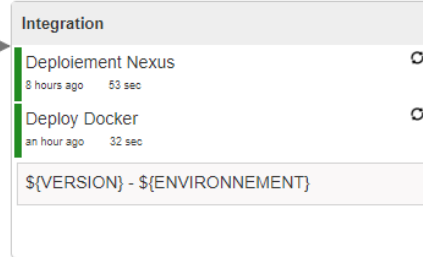
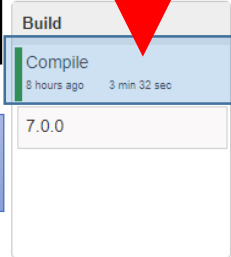
Docker + CDK. Caso de estudio genérico.

INPUT: **`${VERSION}`**



```
svn update / git pull (master)
mvn clean install
```

1



INPUT: **`${VERSION}`**

Generalmente sacada directamente desde GitLab o SVN.

También se puede meter a mano

Detectar cambios en el repositorio de código, SVN (trunk) o GitLab (en la rama master)

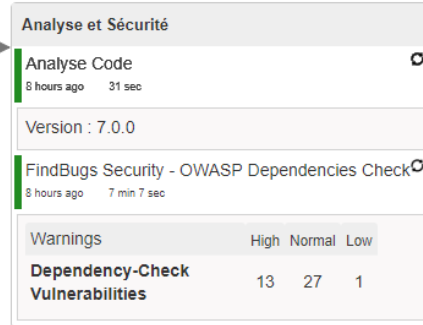
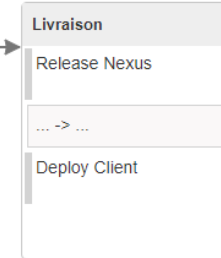
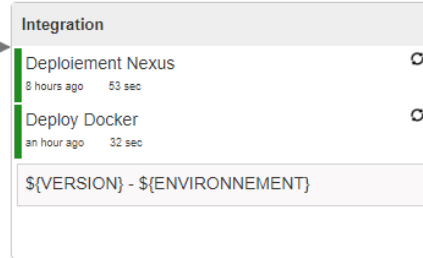
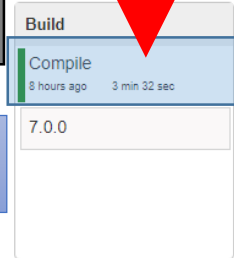
Docker + CDK. Caso de estudio genérico.

INPUT: **`${VERSION}`**



1

```
svn update / git pull (master)
mvn clean install
```



Warnings	High	Normal	Low
Dependency-Check Vulnerabilities	13	27	1

INPUT: **`${VERSION}`**

Generalmente sacada directamente desde GitLab o SVN.

También se puede meter a mano

Detectar cambios en el repositorio de código, SVN (trunk) o GitLab (en la rama master)

Generalmente con `-DskipTests` (no nos interesan los tests en este step)

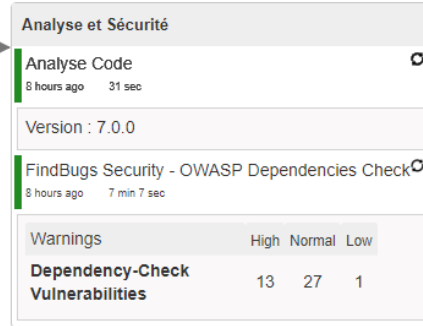
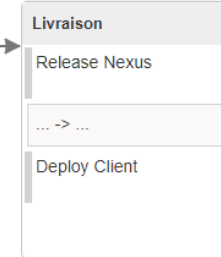
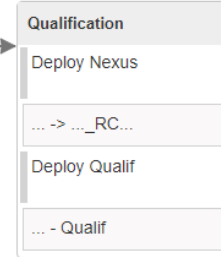
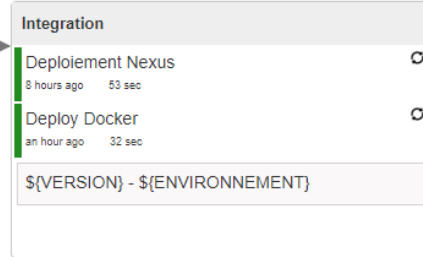
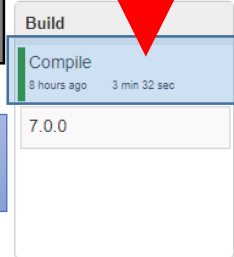
Docker + CDK. Caso de estudio genérico.

INPUT: **`${VERSION}`**



```
svn update / git pull (master)
mvn clean install
```

1



INPUT: **`${VERSION}`**

Generalmente sacada directamente desde GitLab o SVN.

También se puede meter a mano

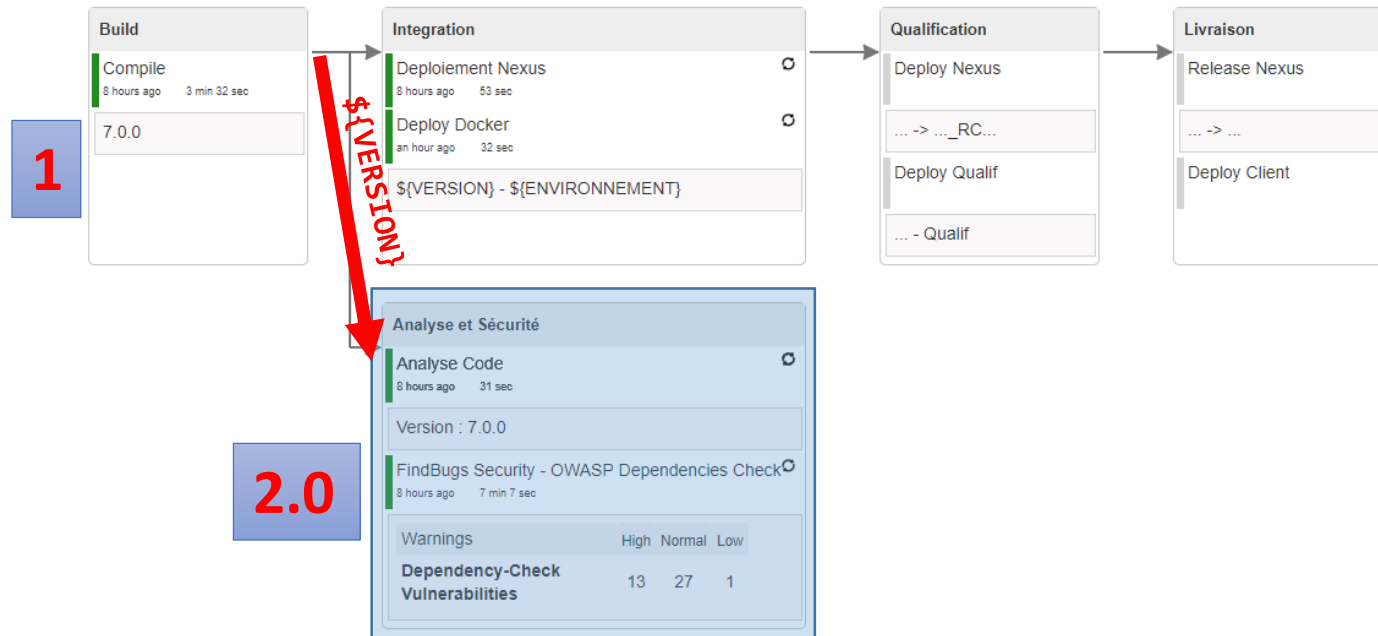
Detectar cambios en el repositorio de código, SVN (trunk) o GitLab (en la rama master)

Generalmente con `-DskipTests` (no nos interesan los tests en este step)

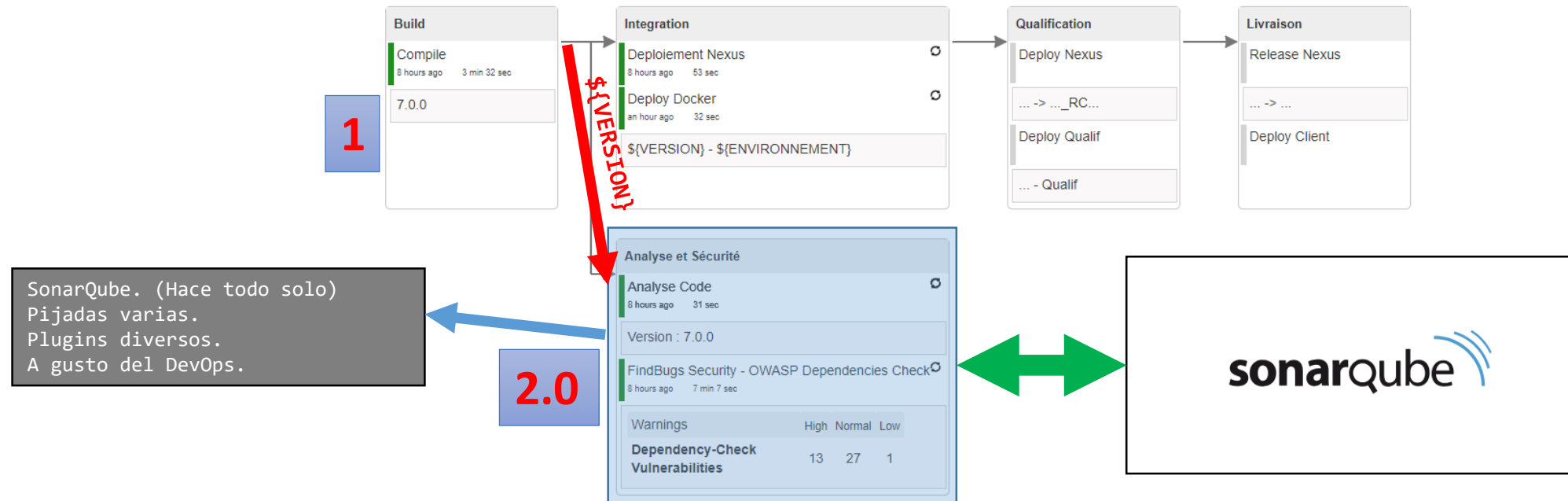
Si la compilación peta, Jenkins se configura para que mande emails a RT.

Y el culpable de la petada **a pagar ronda de cafés.**

Docker + CDK. Caso de estudio genérico.



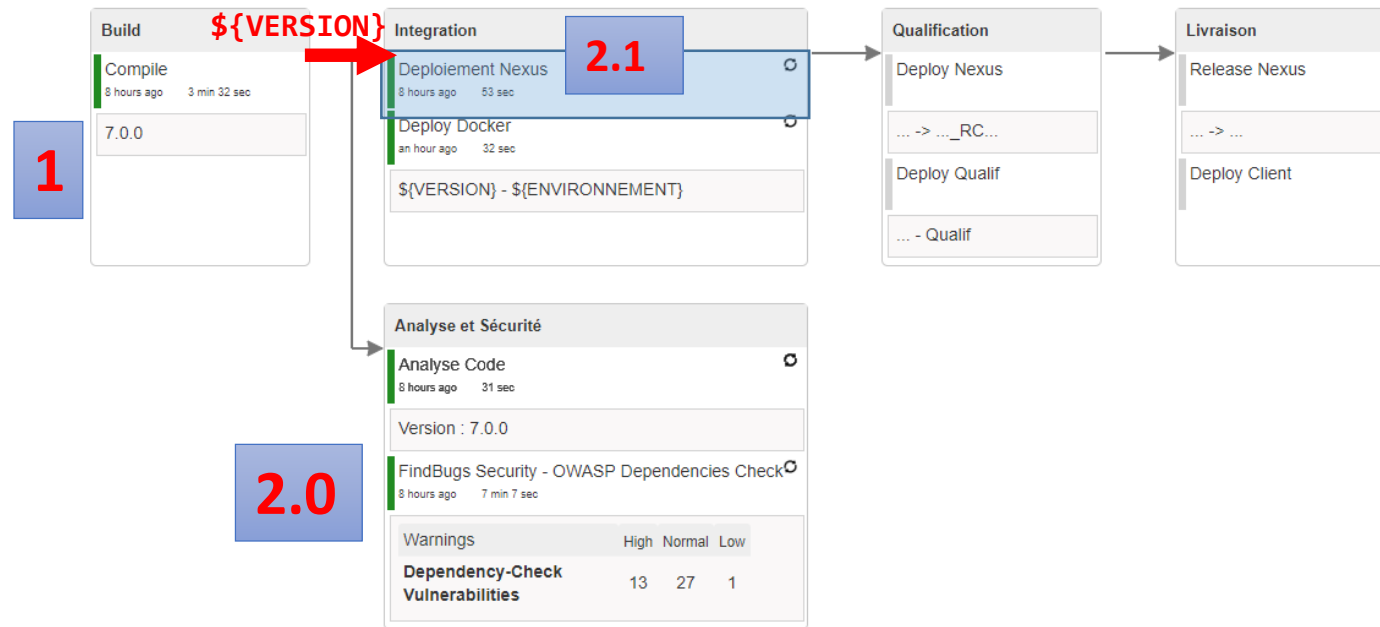
Docker + CDK. Caso de estudio genérico.



Aquí, en esta etapa, es donde se pasan todos los tests unitarios. Se saca calidad de código, etc.

Si hay problemas de seguridad o algo que no entre en las pautas del proyecto, se configura para que mande emails a RT.

Docker + CDK. Caso de estudio genérico.

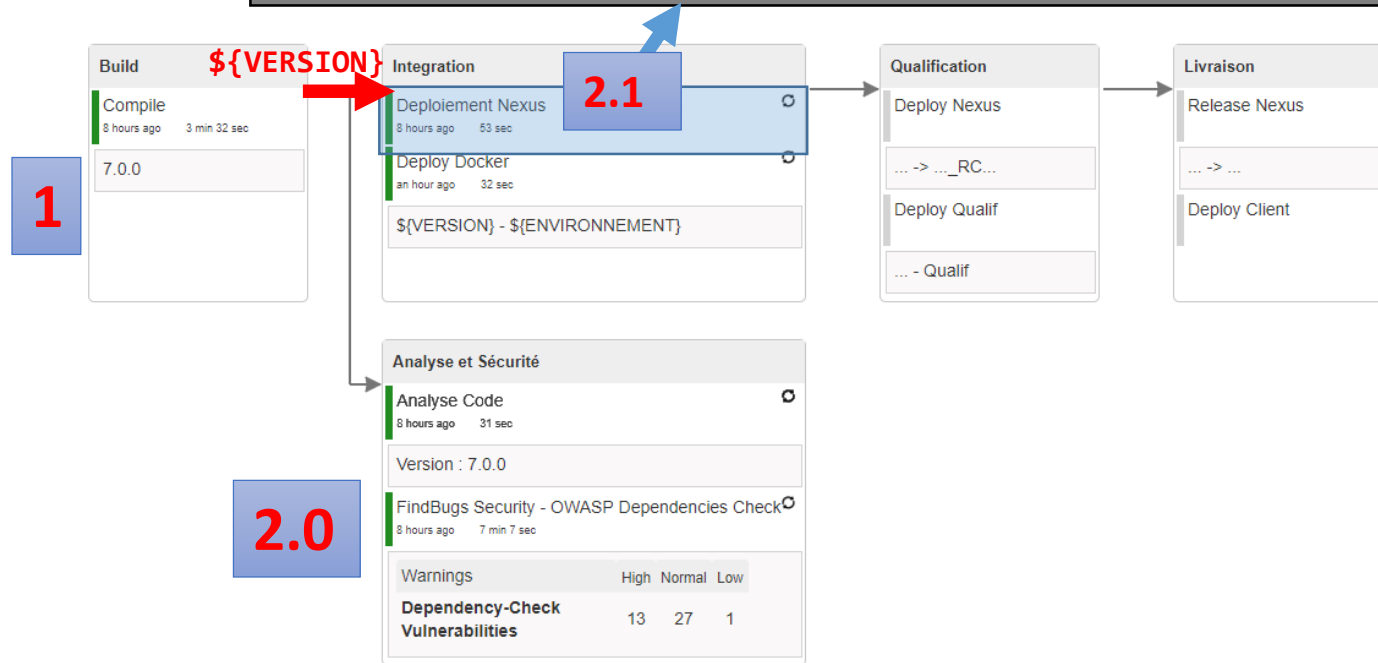


Docker + CDK. Caso de estudio genérico.

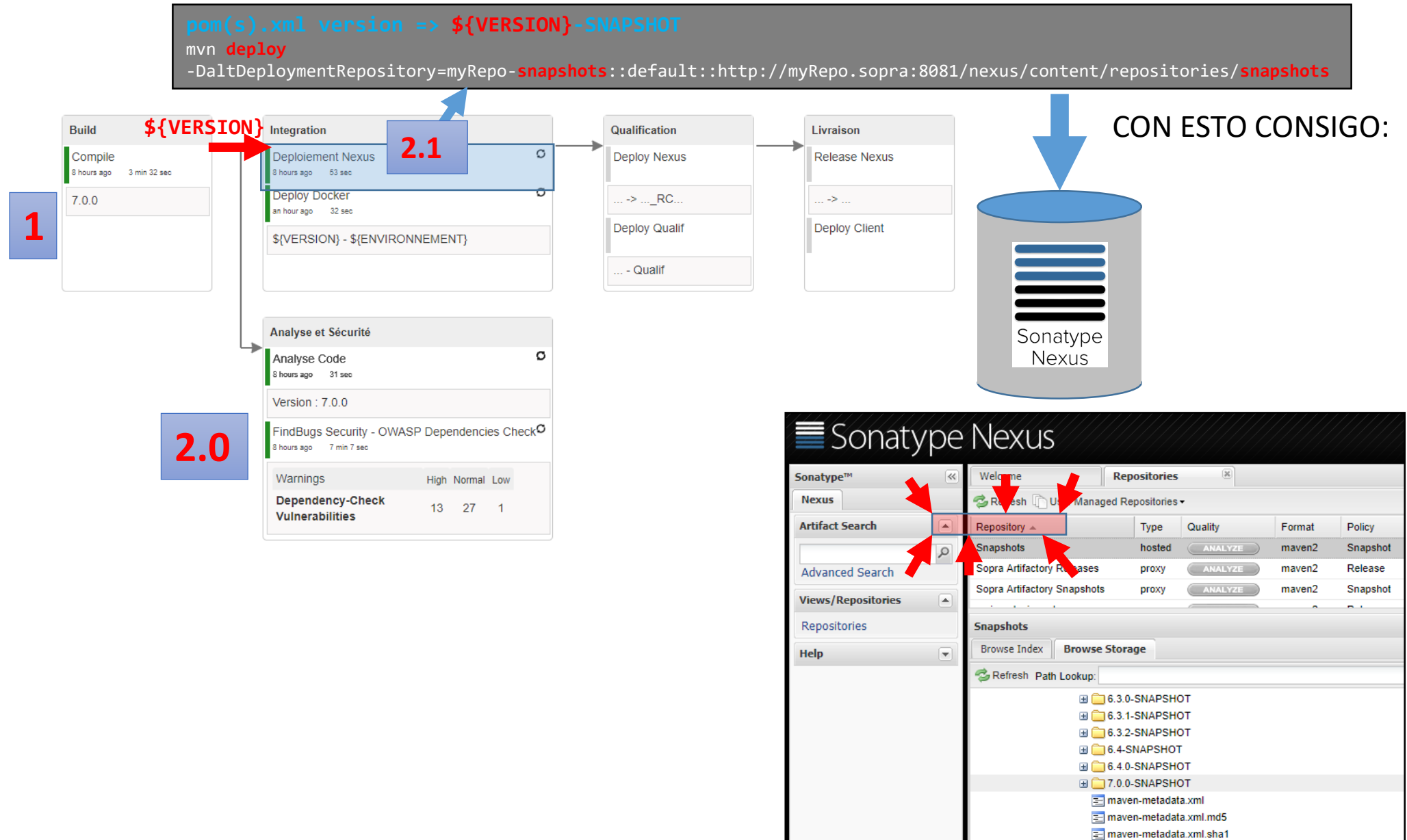
```
pom(s).xml version => ${VERSION}-SNAPSHOT
```

```
mvn deploy
```

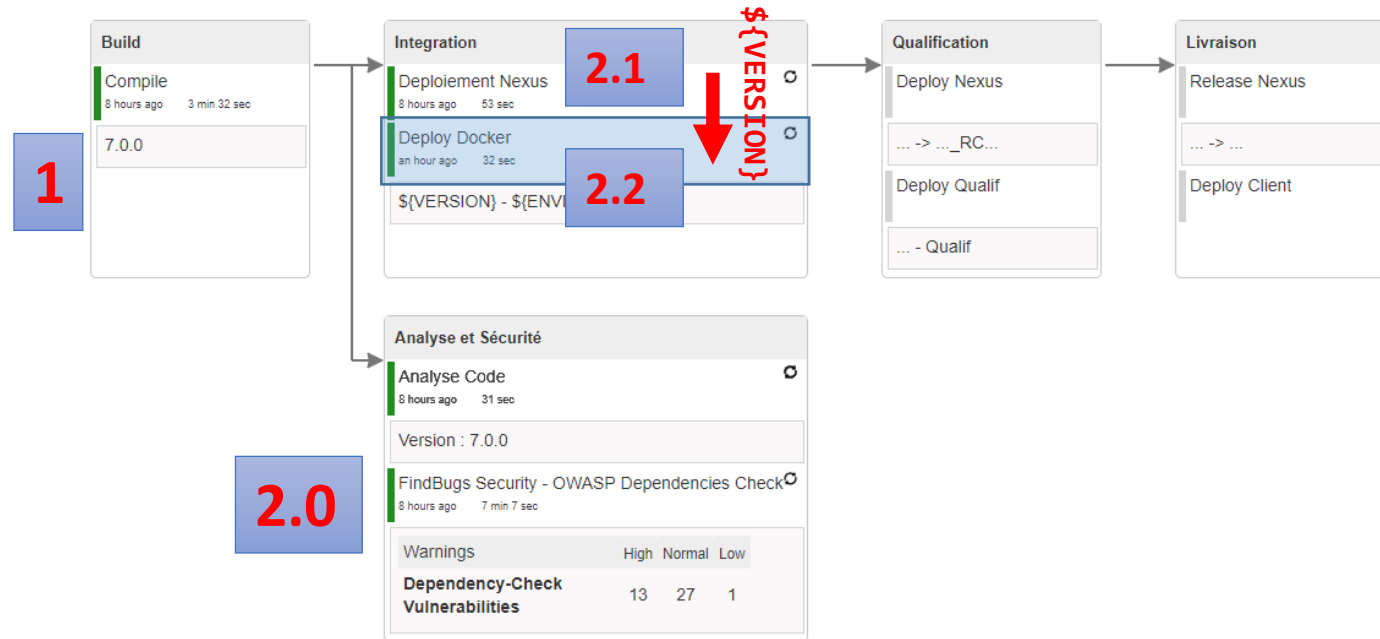
```
-DaltDeploymentRepository=myRepo-snapshots::default::http://myRepo.sopra:8081/nexus/content/repositories/snapshots
```



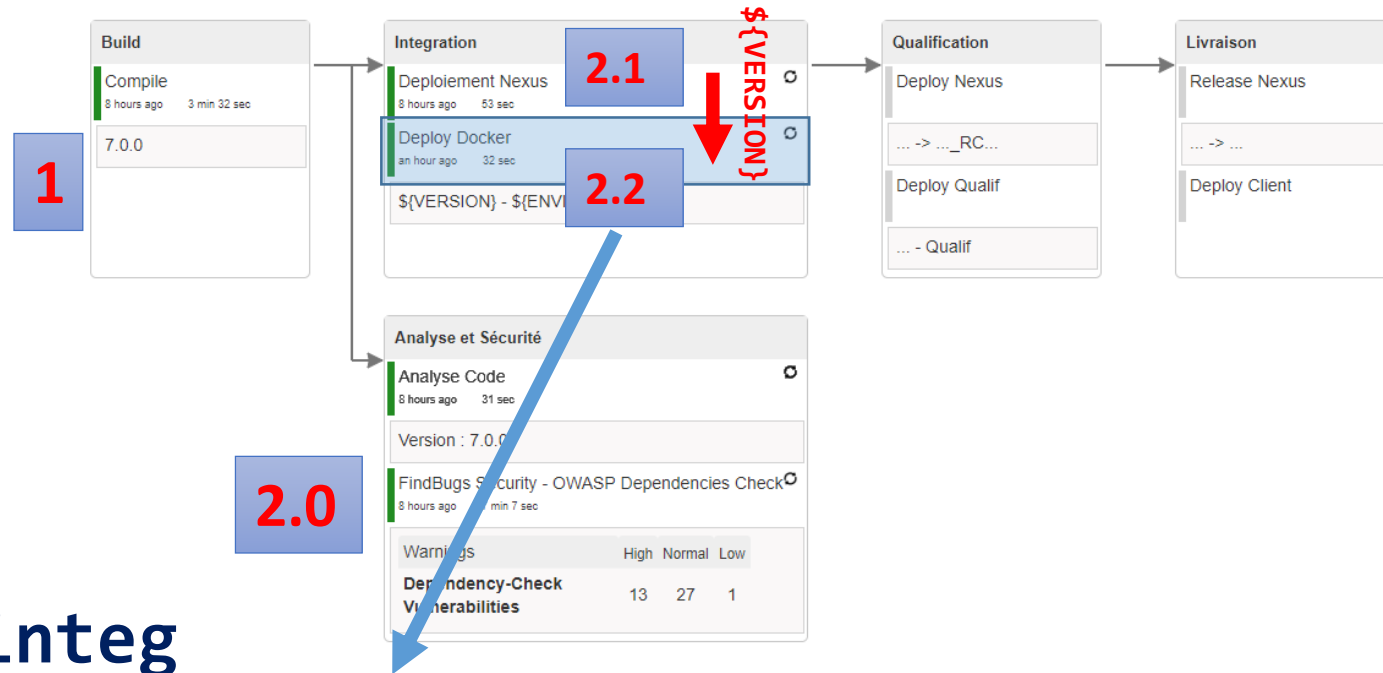
Docker + CDK. Caso de estudio genérico.



Docker + CDK. Caso de estudio genérico.



Docker + CDK. Caso de estudio genérico.



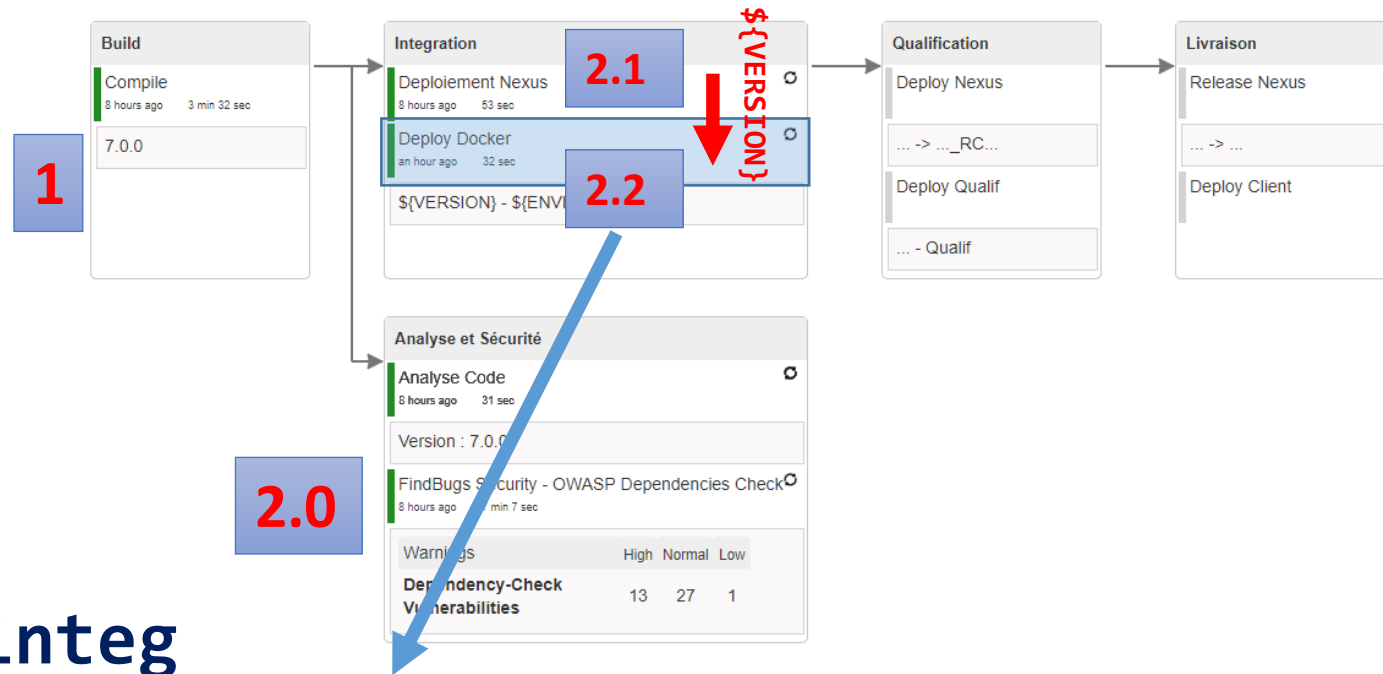
`${ENVIRONEMENT}` → **integ**

Update del repositorio (SVN o Git). Para los scripts.

```
chmod +x *.sh  
chmod +x *.yaml
```

```
./myApp-deploy-stack.sh ${VERSION}
```

Docker + CDK. Caso de estudio genérico.



Update del repositorio (SVN o Git). Para los scripts.

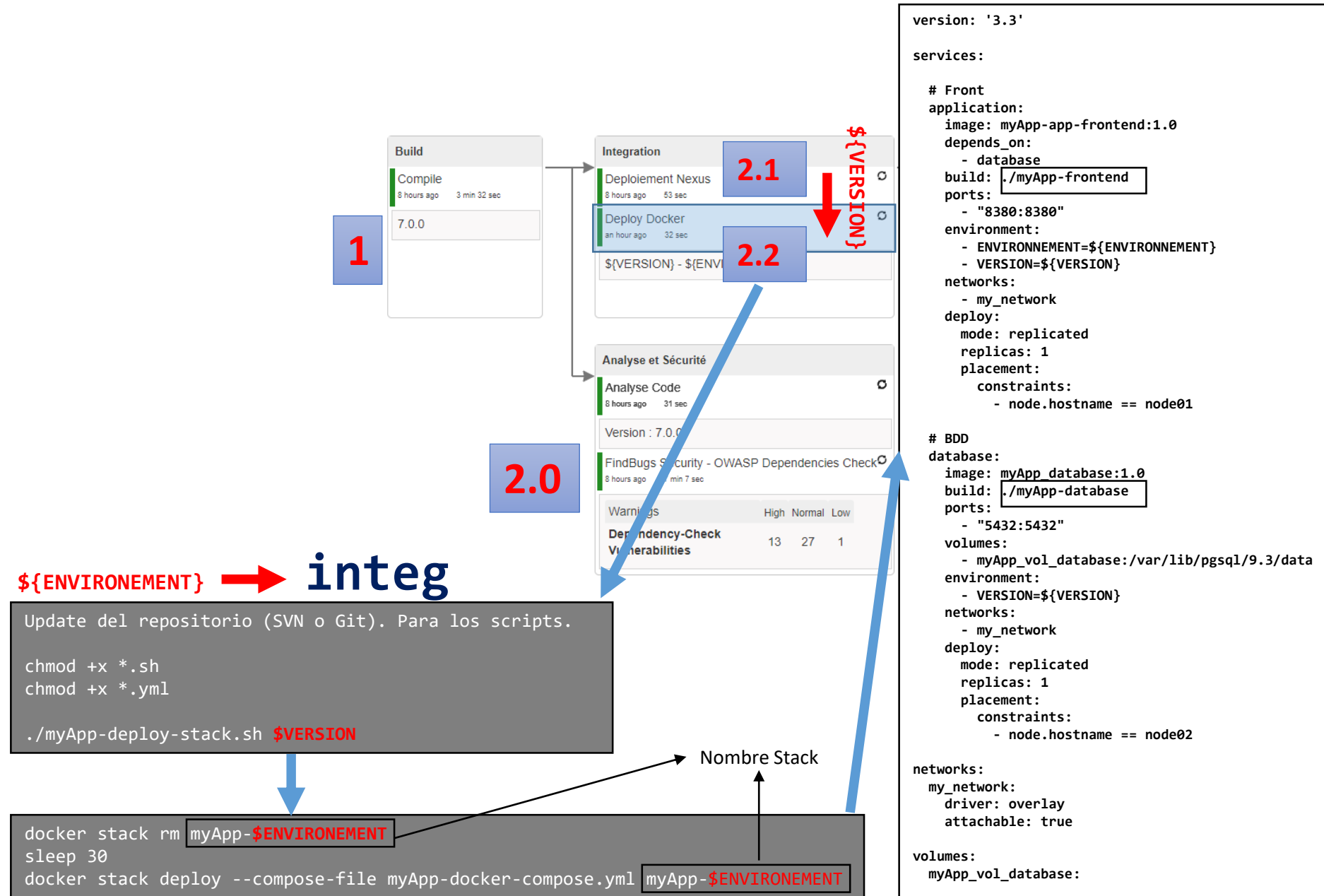
```
chmod +x *.sh  
chmod +x *.yaml
```

```
./myApp-deploy-stack.sh 2.0
```

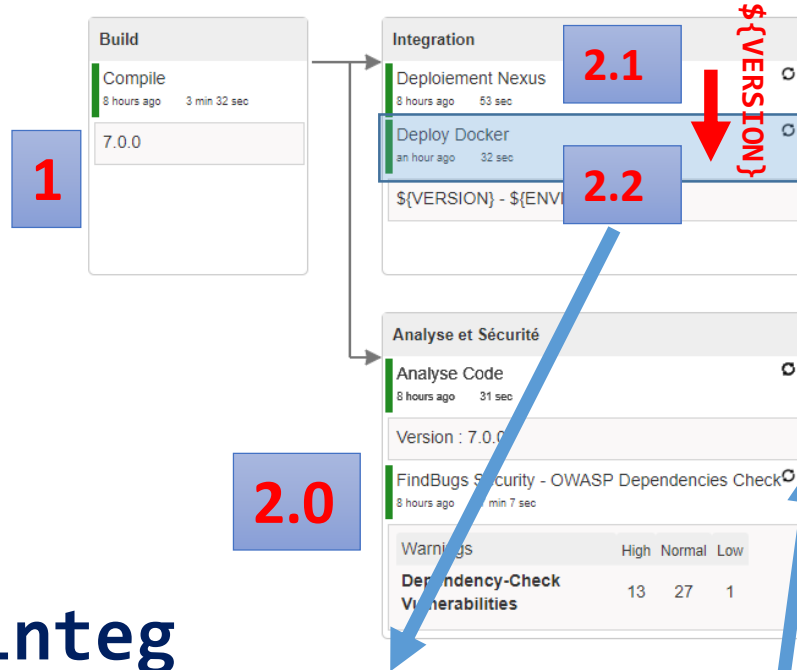
```
docker stack rm myApp-2.0  
sleep 30  
docker stack deploy --compose-file myApp-docker-compose.yaml myApp-2.0
```

Nombre Stack

Docker + CDK. Caso de estudio genérico.



Docker + CDK. Caso de estudio genérico.



```
version: '3.3'

services:

  # Front
  application:
    image: myApp-app-frontend:1.0
    depends_on:
      - database
    build: ./myApp-frontend
    ports:
      - "8380:8380"
    environment:
      - ENVIRONNEMENT=${ENVIRONNEMENT}
      - VERSION=${VERSION}
    networks:
      - my_network
    deploy:
      mode: replicated
      replicas: 1
      placement:
        constraints:
          - node.hostname == node01

  # BDD
  database:
    image: myApp_database:1.0
    build: ./myApp-database
    ports:
      - "5432:5432"
    volumes:
      - myApp_vol_database:/var/lib/postgresql/9.3/data
    environment:
      - VERSION=${VERSION}
    networks:
      - my_network
    deploy:
      mode: replicated
      replicas: 1
      placement:
        constraints:
          - node.hostname == node02

networks:
  my_network:
    driver: overlay
    attachable: true

volumes:
  myApp_vol_database:
```

```
FROM tomcat:8

MAINTAINER SopraSteria

ENV NEXUS_URL http://my nexus.sopra:8081/nexus
ENV ENVIRONNEMENT $ENVIRONNEMENT
ENV VERSION $VERSION

# répertoire de travail
WORKDIR /tmp

# création de l'environnement de base
RUN mkdir -p "$CATALINA_HOME/myApp-conf"

# nettoyage
RUN rm -rf "$CATALINA_HOME/work/*"
"$CATALINA_HOME/temp/*"

# Copie du entrypoint et modification des droits
COPY docker-entrypoint.sh /usr/bin/
RUN chmod 744 /usr/bin/docker-entrypoint.sh

ENTRYPOINT ["/usr/bin/docker-entrypoint.sh"]
```

`${ENVIRONNEMENT}` → integ

Update del repositorio (SVN o Git). Para los scripts.

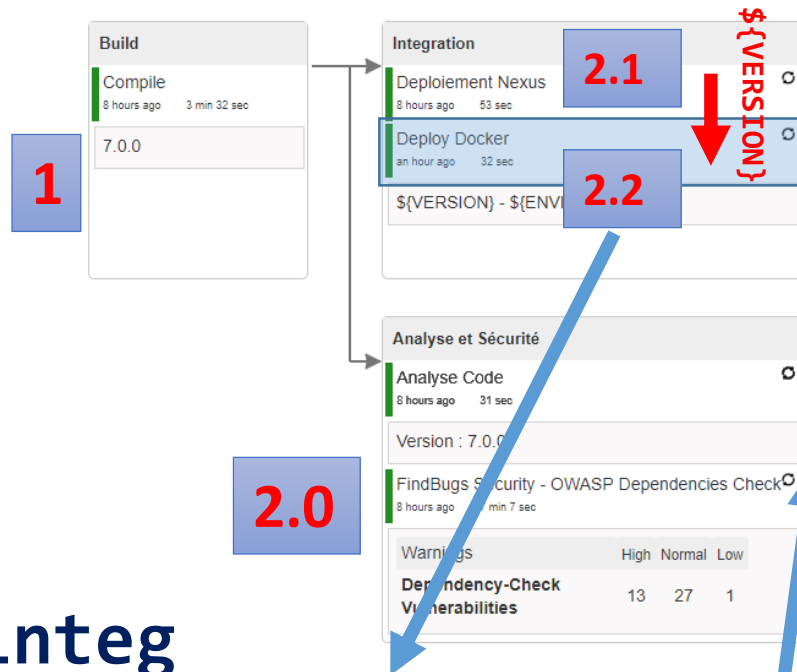
```
chmod +x *.sh
chmod +x *.yaml
```

```
./myApp-deploy-stack.sh $VERSION
```

```
docker stack rm myApp-$ENVIRONNEMENT
sleep 30
docker stack deploy --compose-file myApp-docker-compose.yml myApp-$ENVIRONNEMENT
```

Nombre Stack

Docker + CDK. Caso de estudio genérico.



```
version: '3.3'

services:
  # Front
  application:
    image: myApp-app-frontend:1.0
    depends_on:
      - database
    build: ./myApp-frontend
    ports:
      - "8380:8380"
    environment:
      - ENVIRONNEMENT=${ENVIRONNEMENT}
      - VERSION=${VERSION}
    networks:
      - my_network
    deploy:
      mode: replicated
      replicas: 1
      placement:
        constraints:
          - node.hostname == node01

  # BDD
  database:
    image: myApp_database:1.0
    build: ./myApp-database
    ports:
      - "5432:5432"
    volumes:
      - myApp_vol_database:/var/lib/postgresql/9.3/data
    environment:
      - VERSION=${VERSION}
    networks:
      - my_network
    deploy:
      mode: replicated
      replicas: 1
      placement:
        constraints:
          - node.hostname == node02

networks:
  my_network:
    driver: overlay
    attachable: true

volumes:
  myApp_vol_database:
```

```
FROM tomcat:8

MAINTAINER SopraSteria

ENV NEXUS_URL http://my nexus.sopra:8081/nexus
ENV ENVIRONNEMENT $ENVIRONNEMENT
ENV VERSION $VERSION

# répertoire de travail
WORKDIR /tmp

# création de l'environnement de base
RUN mkdir -p "$CATALINA_HOME/myApp-conf"

# nettoyage
RUN rm -rf "$CATALINA_HOME/work/*"
"$CATALINA_HOME/temp/*"

# Copie du entrypoint et modification des droits
COPY docker-entrypoint.sh /usr/bin/
RUN chmod 744 /usr/bin/docker-entrypoint.sh

ENTRYPOINT ["/usr/bin/docker-entrypoint.sh"]
```

```
#!/bin/sh

# détermination du repo à utiliser
if [[ $VERSION == *"SNAPSHOT"* ]]; then
  repo="snapshots"
else
  repo="releases"
fi

# on télécharge le war et les config
wget -O aubette.war
"$NEXUS_URL/service/local/artifact/maven/content?r=$
repo&g=sopra.myApp&a=myApp-web&e=war&v=$VERSION"

# dépôt de l'artefact dans webapps
mv aubette.war "$CATALINA_HOME/webapps"

echo "Sleep 10 pour attendre la BDD"
sleep 10

cd /srv/tomcat/bin \
&& ./catalina.sh run
```

Update del repositorio (SVN o Git). Para los scripts.

```
chmod +x *.sh
chmod +x *.yml
```

```
./myApp-deploy-stack.sh $VERSION
```

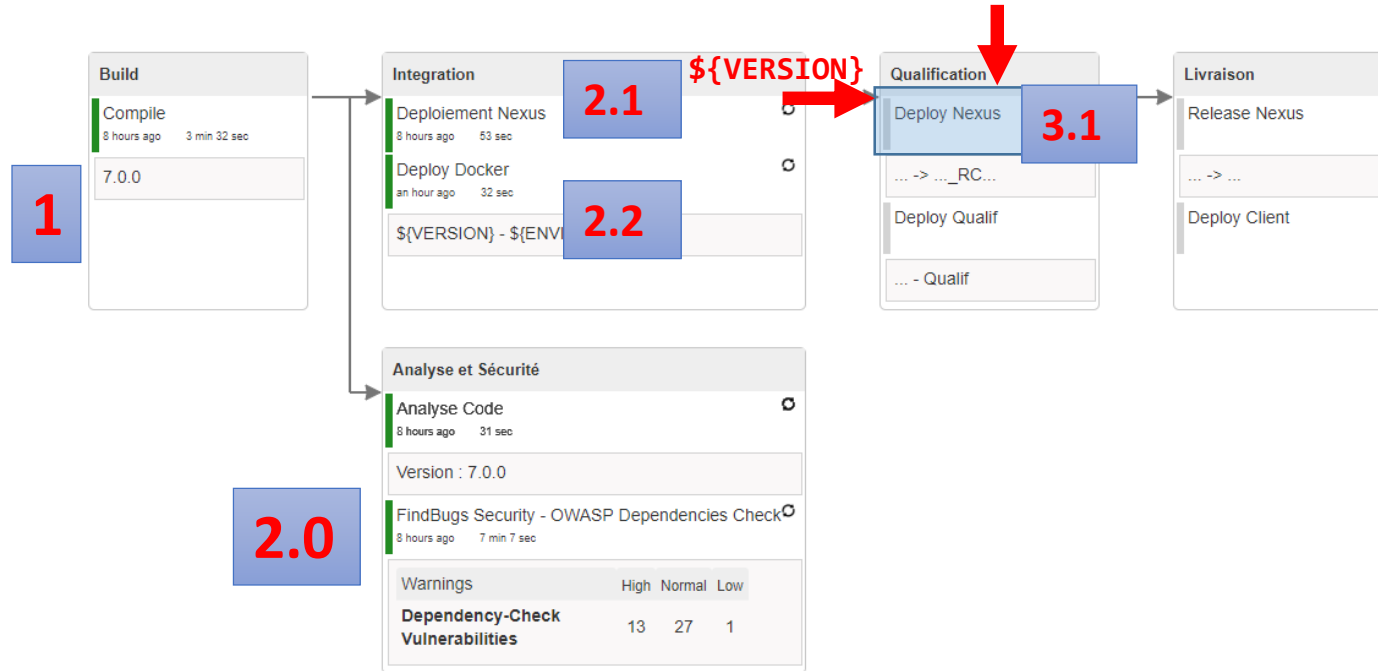
```
docker stack rm myApp-$ENVIRONNEMENT
sleep 30
docker stack deploy --compose-file myApp-docker-compose.yml myApp-$ENVIRONNEMENT
```

Nombre Stack

Docker + CDK. Caso de estudio genérico.

Generalmente tiene que aprobarse manualmente.

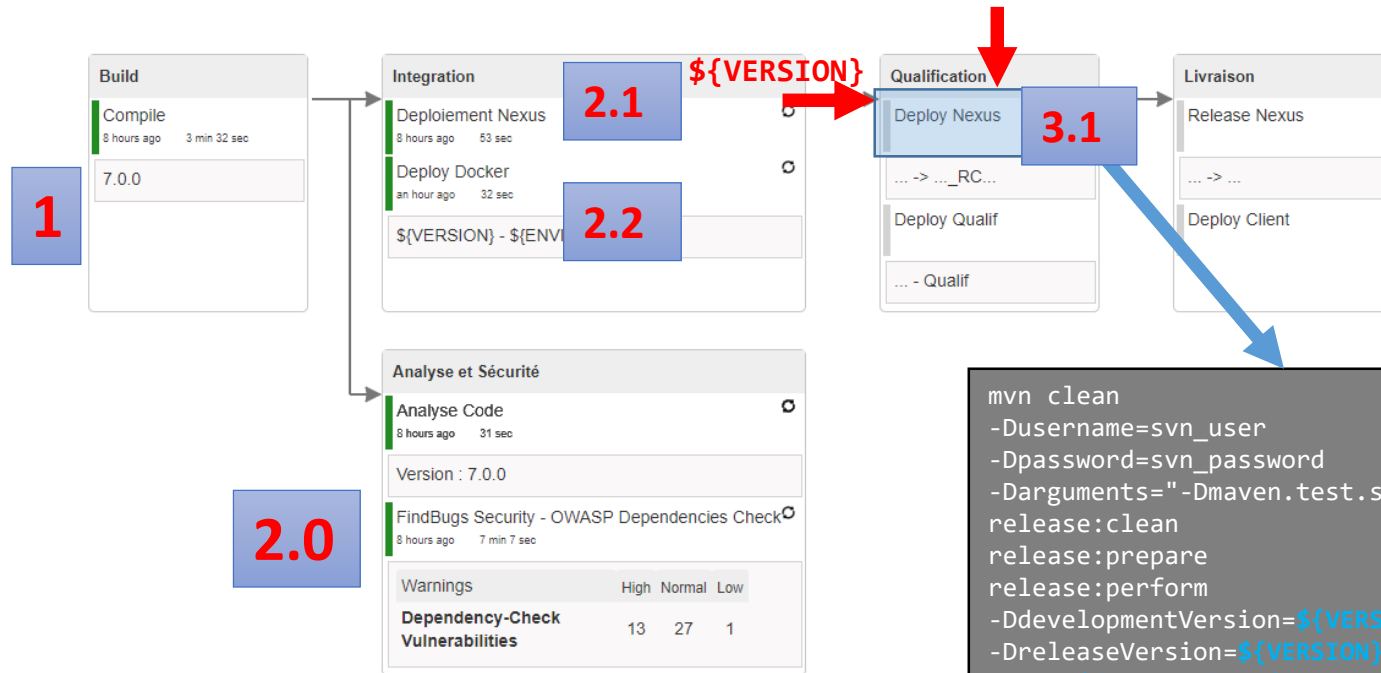
INPUT: **`${NUM_PASSE_QUALIF}`**



Docker + CDK. Caso de estudio genérico.

Generalmente tiene que aprobarse manualmente.

INPUT: **`${NUM_PASSE_QUALIF}`**

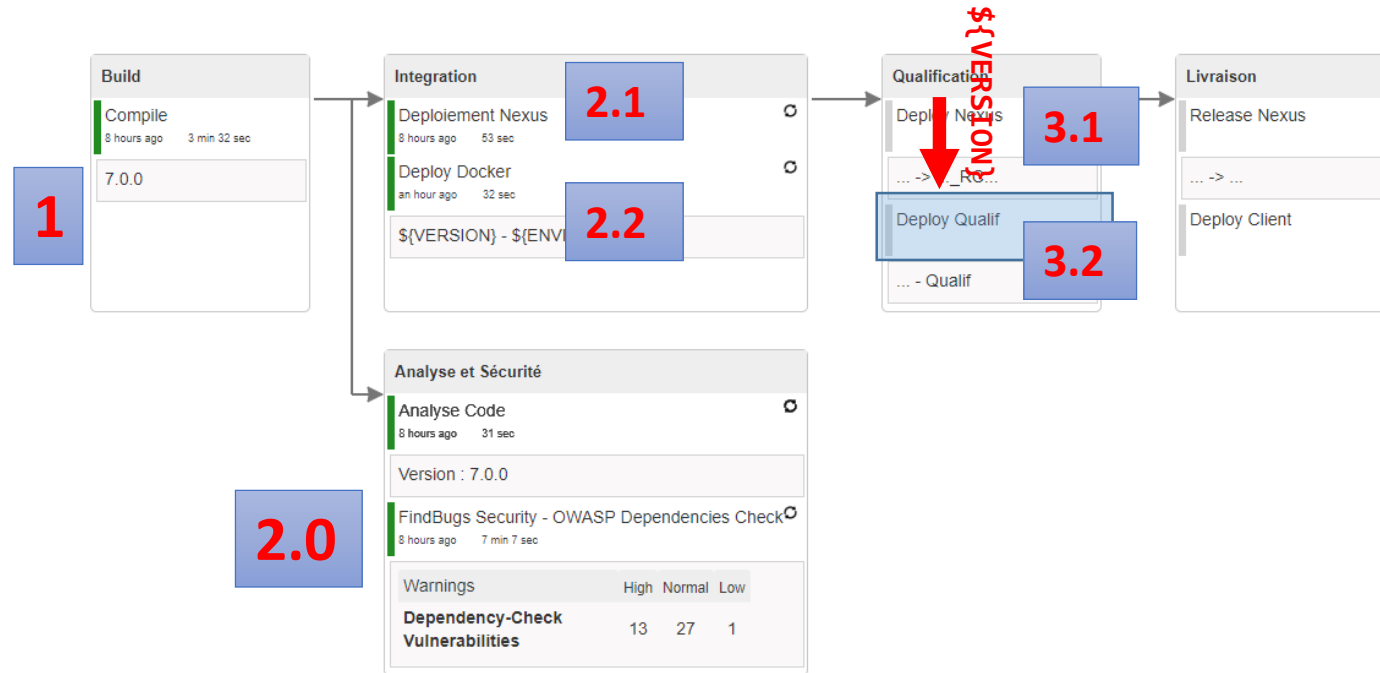


```
mvn clean
-Dusername=svn_user
-Dpassword=svn_password
-Darguments="-Dmaven.test.skip=true -DskipITs -Dmaven.javadoc.skip=true"
release:clean
release:prepare
release:perform
-DdevelopmentVersion=${VERSION}-SNAPSHOT
-DreleaseVersion=${VERSION}-RC${NUM_PASSE_QUALIF}
-Dtag=${VERSION}-RC${NUM_PASSE_QUALIF}
-Dgoals=deploy
```

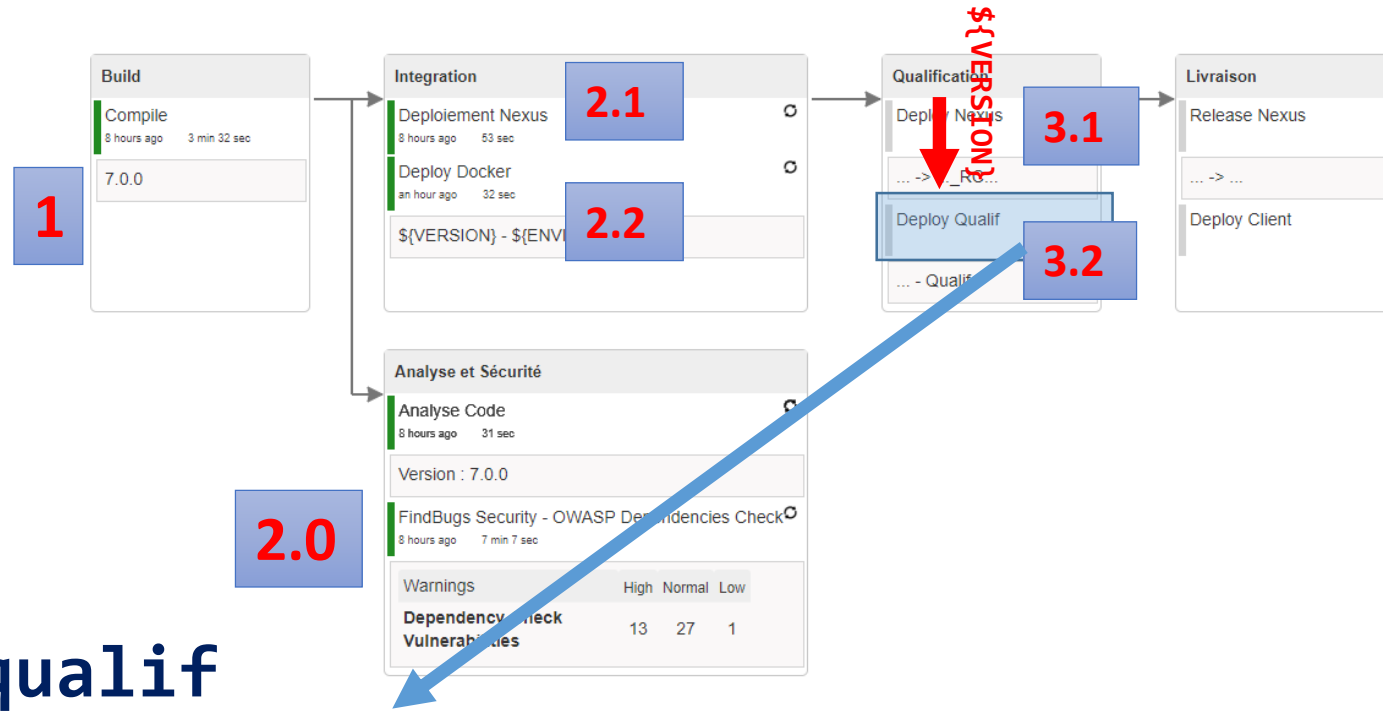
Sube un artifact (JAR / WAR / EAR) al nexus (deploy)
Hace un tag del código en SVN (o GitLab)



Docker + CDK. Caso de estudio genérico.



Docker + CDK. Caso de estudio genérico.



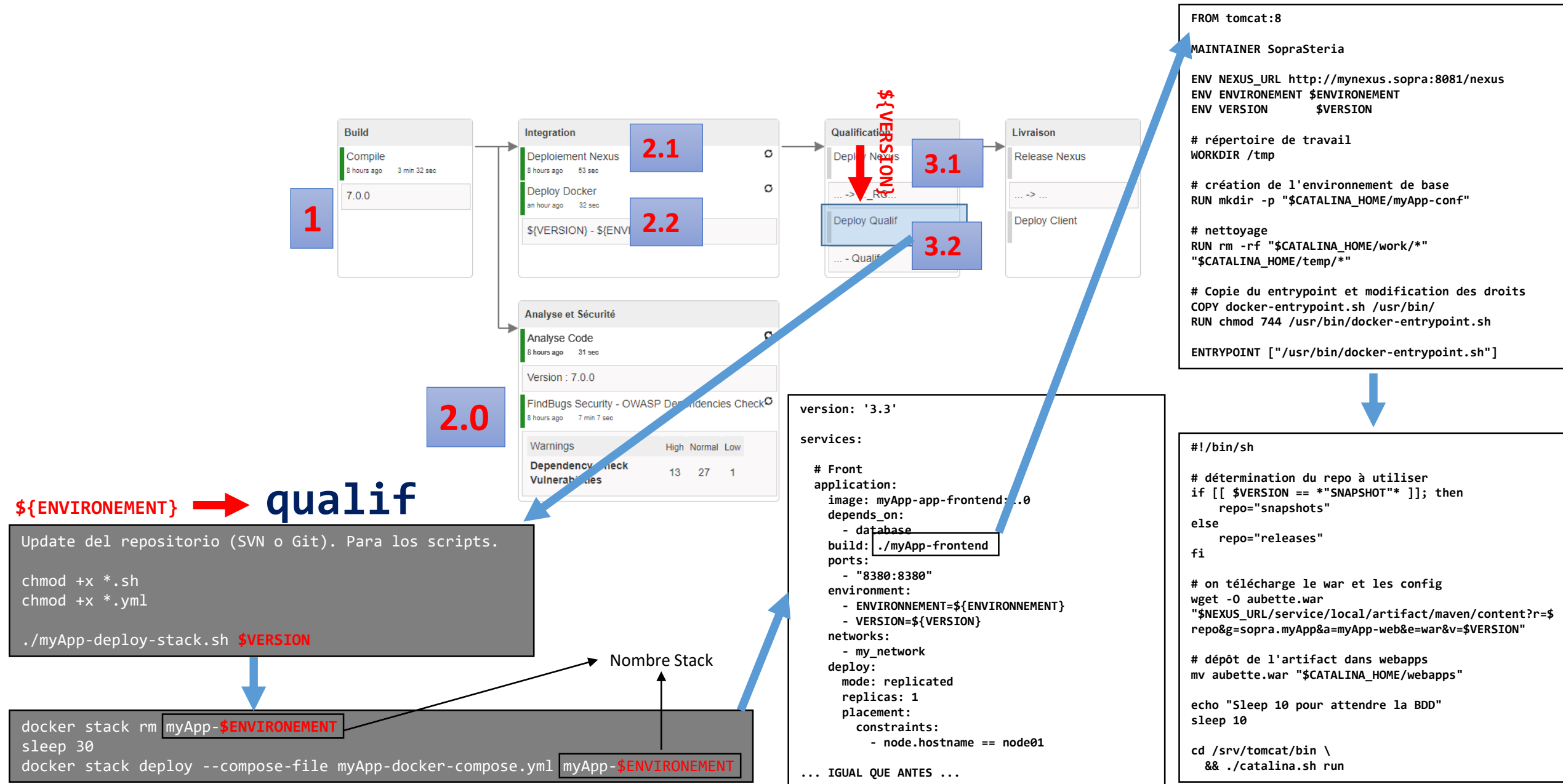
`\${ENVIRONEMENT}` → qualif

Update del repositorio (SVN o Git). Para los scripts.

```
chmod +x *.sh  
chmod +x *.yaml
```

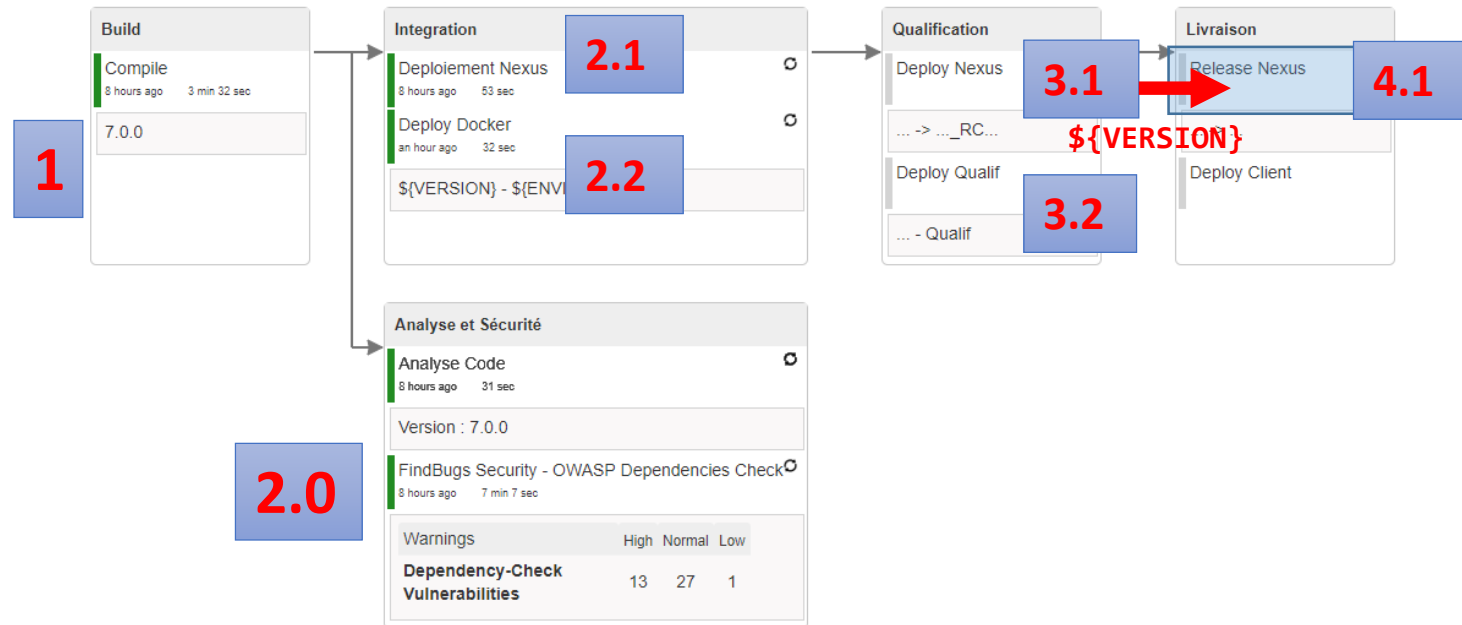
```
./myApp-deploy-stack.sh `${VERSION}`
```

Docker + CDK. Caso de estudio genérico.



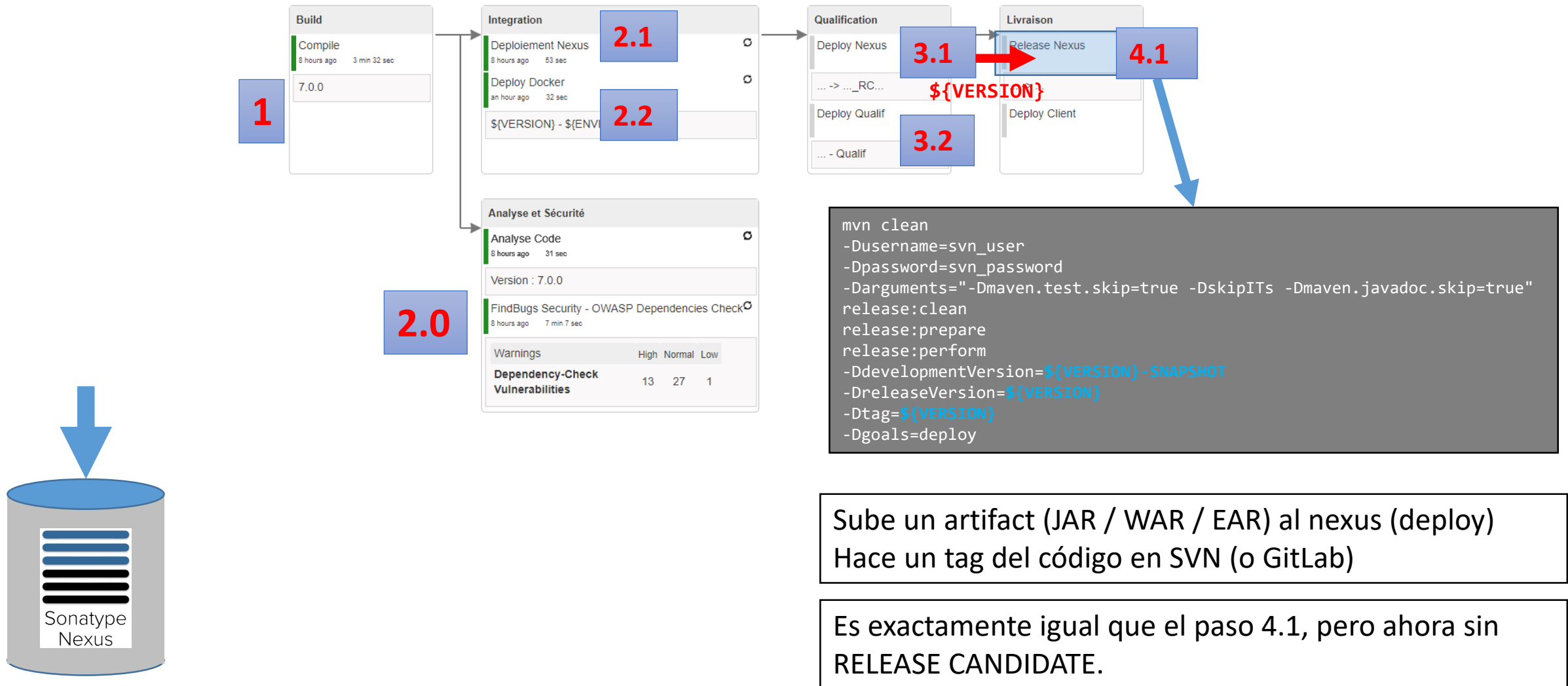
Docker + CDK. Caso de estudio genérico.

Obligatoriamente tiene que aprobarse manualmente.
No vamos a hacer entregas al cliente a cada cambio en el SVN.

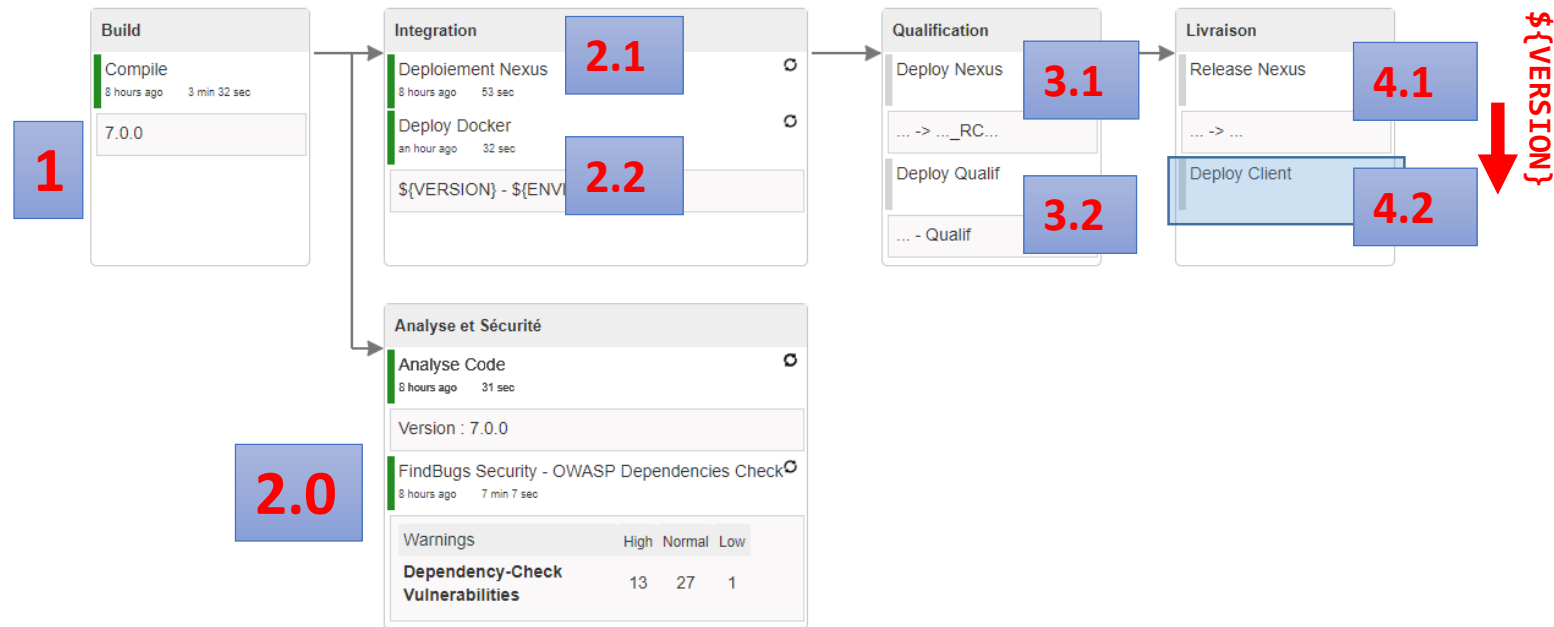


Docker + CDK. Caso de estudio genérico.

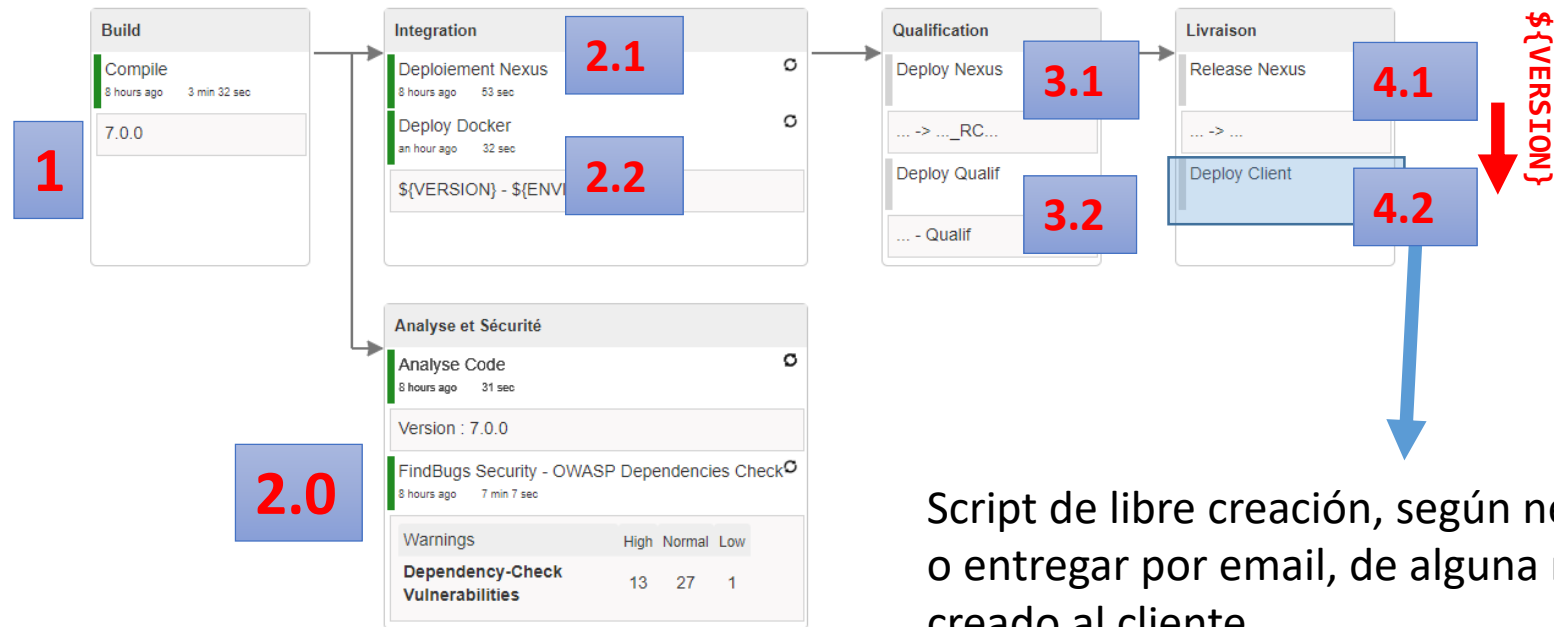
Obligatoriamente tiene que aprobarse manualmente.
No vamos a hacer entregas al cliente a cada cambio en el SVN.



Docker + CDK. Caso de estudio genérico.

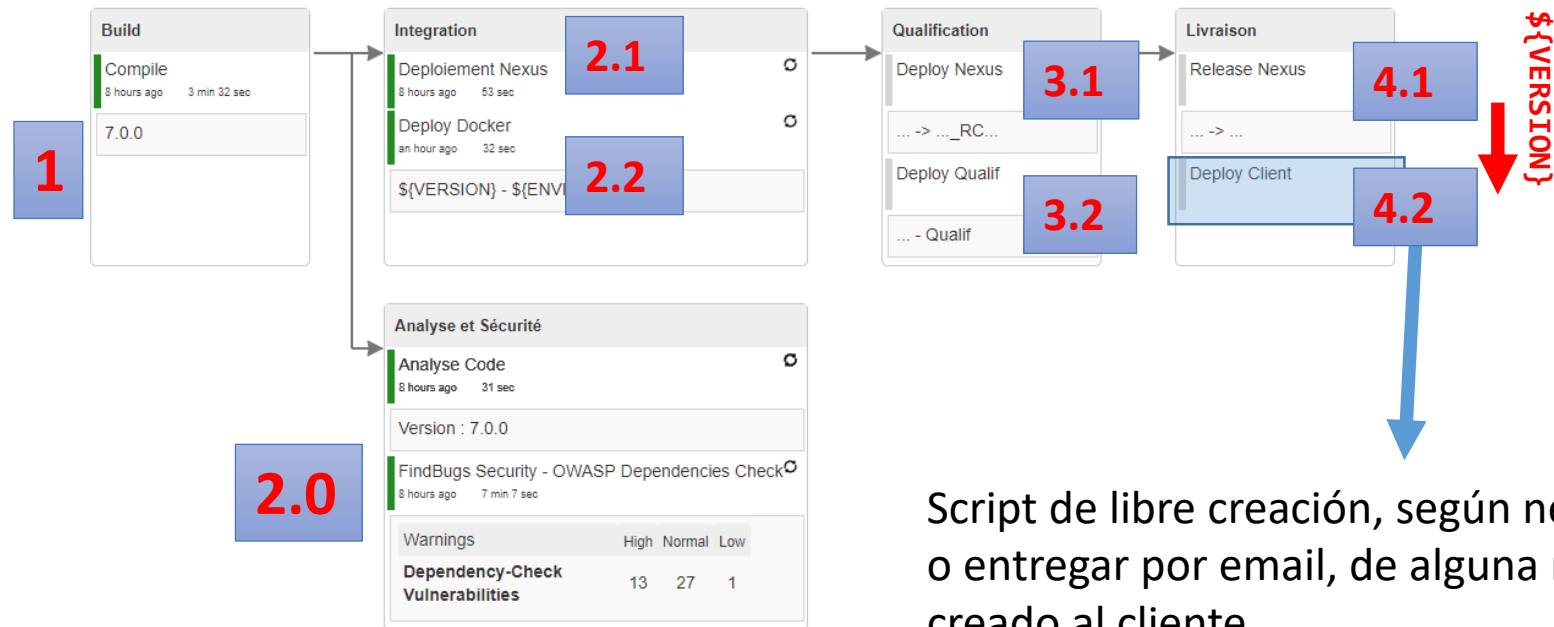


Docker + CDK. Caso de estudio genérico.



Script de libre création, según necesidades, para subir o entregar por email, de alguna manera el artefacto creado al cliente.

Docker + CDK. Caso de estudio genérico.



Script de libre création, según necesidades, para subir o entregar por email, de alguna manera el artefacto creado al cliente.

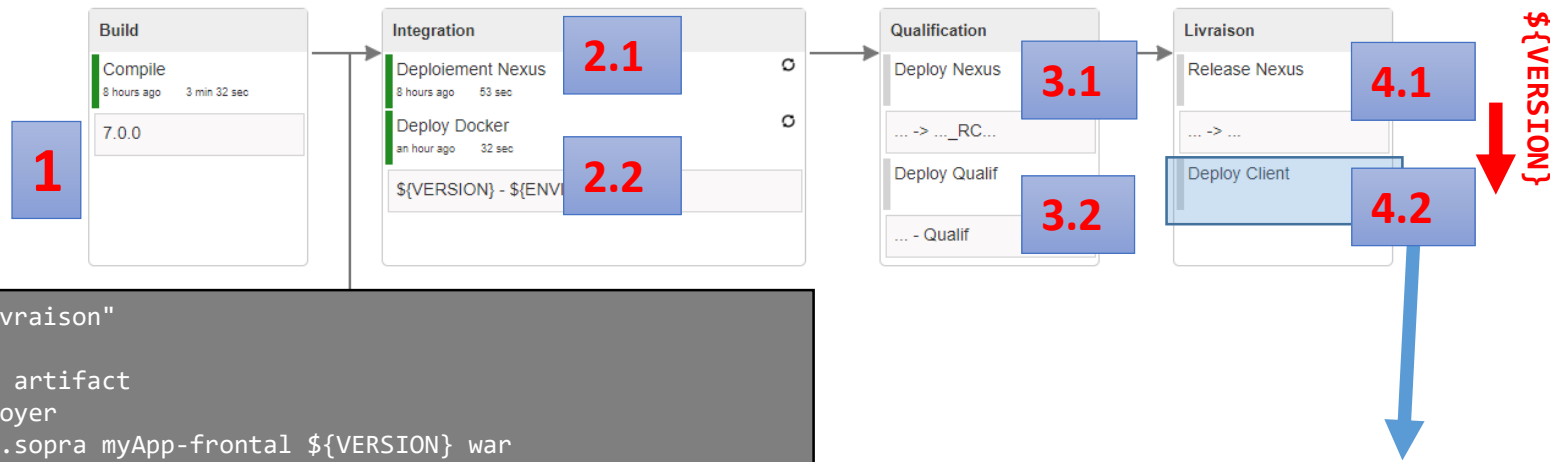
Un ejemplo sería:

```
#!/bin/sh

chmod +x *.sh

./myApp-livraison.sh ${VERSION}
```

Docker + CDK. Caso de estudio genérico.



```
echo " --> Debut livraison"

# telechargement du artefact
cd /apps/nexus-deployer
./download.sh myApp.sopra myApp-frontal ${VERSION} war

mkdir -p myApp-war/front

cp myApp-frontal.war myApp-war/front1

cd myApp-war \
  && zip -r myApp-war-${VERSION}.zip * \
  && mv myApp-war-${VERSION}.zip ../myApp-war.zip \
  && cd .. \
  && ./upload.sh myApp.sopra myApp-war ${VERSION} zip

# nettoyage
rm -rf *.war
rm -rf *.zip

echo " --> Transfert terminé"
```

Script de libre création, según necesidades, para subir o entregar por email, de alguna manera el artefacto creado al cliente.

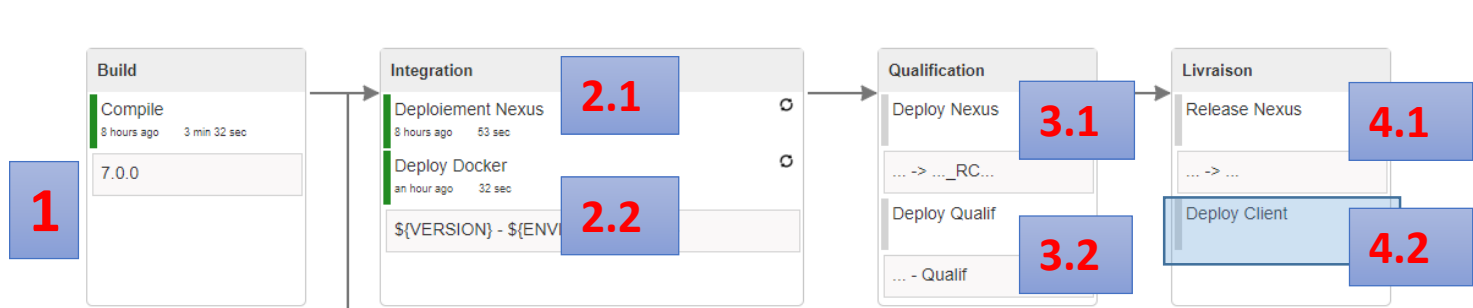
Un ejemplo sería:

```
#!/bin/sh

chmod +x *.sh

./myApp-livraison.sh $VERSION
```


Docker + CDK. Caso de estudio genérico.



2.0

Analyse et Sécurité

Analyse Code (8 hours ago, 31 sec)

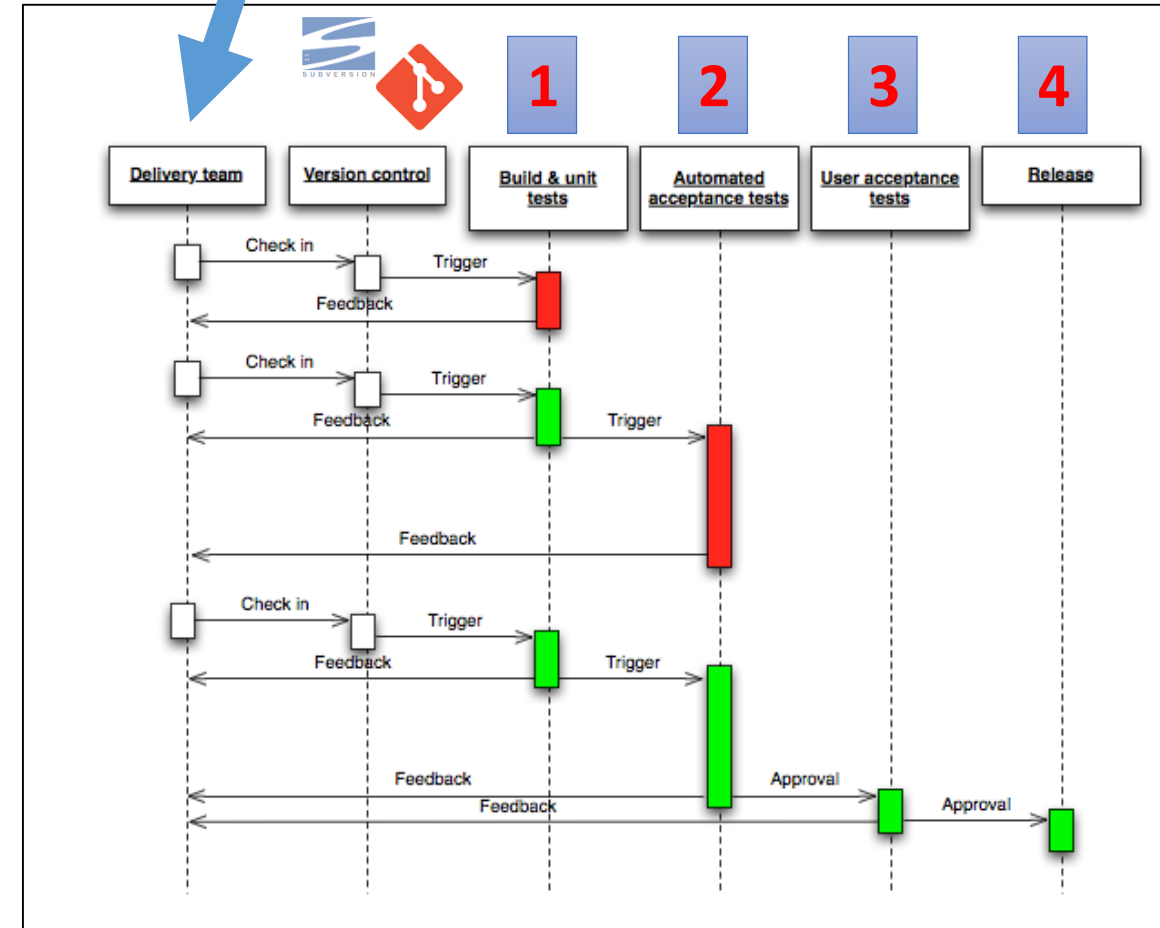
Version : 7.0.0

FindBugs Security - OWASP Dependencies Check (8 hours ago, 7 min 7 sec)

Warnings	High	Normal	Low
Dependency-Check Vulnerabilities	13	27	1

Developers!
Developers!
Developers!

Algún funcional también...



Recomendación personal (ejercicio para casa)

1. Instalarse un Ubuntu / CentOS. En nativo o máquina virtual
2. Instalar Jenkins
3. Instalar Nexus
4. Instalar Docker
5. Instalar GitLab en local (o SVN si se siente más cómodo, pero altamente recomendado GitLab)
6. Instalar SonarQube (Opcional, todo el mundo sabe que nadie mira la cobertura de código...)
7. Hacer una app de ejemplo muy sencilla (como la vista aquí)
8. Crear todo el pipeline en jenkins

Otros temas relacionados con Docker.

- Kubernetes
 - ¿Qué es? Brevísima introducción a la Orchestration
- Docker con AWS ¿qué es AWS? ¿Por qué está triunfando como la coca-cola?
- ¿qué sentido tiene Docker para un desarrollador? ¿Para qué puede estar bien Docker en local? Desarrollo con un contenedor ya hecho.

END

eso fue todo amigos



Ahora ya podéis montar una startup

SUGERENCIAS Y COMENTARIOS SOBRE EL CURSO

tomas.garcia-pozuelobarrios@soprasteria.com