CONTENIDOS

1 DESPLIEGUE DE APLICACIONES EN CONTENEDORES (2 HORAS)

- Introducción a los contenedores
- Arquitectura de microservicios
- Tecnologías subyacentes y diferencias entre ellas: docker, cri-o, LXC, ...
- Ciclo de vida en el despliegue de aplicaciones con docker

2 INTRO A KUBERNETES (2 HORAS)

- Características, historia, estado actual del proyecto kubernetes (k8s)
- Arquitectura básica de k8s
- Alternativas para instalación simple de k8s: minikube, kubeadm, k3s
- Instalación con minikube
- Instalación y uso de kubectl
- Despliegue de aplicaciones con k8s

3 DESPLIEGUE DE APLICACIONES CON K8S (1:30 HORAS)

- Pods
- ReplicaSet: Tolerancia y escalabilidad
- Deployment: Actualizaciones y despliegues automáticos

4 COMUNICACIÓN ENTRE SERVICIOS Y ACCESO DESDE EL EXTERIOR (1:30 HORAS)

- Services
- DNS
- Ingress
- Ejemplos de uso y despliegues

5 CONFIGURACIÓN DE APLICACIONES (1 HORA)

- Variables de entorno
- ConfigMaps
- Secrets
- Ejemplo de despliegue parametrizado

6 ALMACENAMIENTO (1:30 HORAS)

- Consideraciones sobre el almacenamiento
- PersistentVolume
- PersistentVolumeClaim
- Ejemplo de despliegue con volúmenes

CONTENIDOS

7 OTROS TIPOS DE DESPLIEGUES (1:30 HORAS)

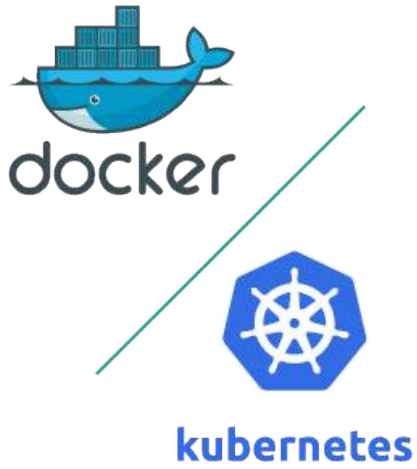
- StatefulSet
- DaemonSet
- AutoScale
- Helm

8 ADMINISTRACIÓN BÁSICA (1 HORA)

- Namespaces
- Usuarios
- RBAC
- Cuotas y límites

9 INSTALACIÓN PASO A PASO (4 HORAS)

- Consideraciones previas: Requisitos hardware, arquitectura física y lógica, entornos y herramientas para el despliegue
- Instalación completa componente a componente en múltiples nodos



1 MÓDULO. DESPLIEGUE DE APLICACIONES EN CONTENEDORES .

Índice

- Introducción a los contenedores
- Arquitectura de microservicios
- Tecnologías subyacentes y diferencias entre ellas: docker, cri-o, LXC, ...
- Ciclo de vida en el despliegue de aplicaciones con docker

Introducción a los contenedores

- ¿Para qué sirve un sistema operativo?¿Qué es un proceso?
- ¿Qué es un contenedor y para qué se utiliza?
- Precedentes en linux
  - chroot
  - OpenVZ
  - Linux vservers
- Precedentes en otros sistemas operativos: FreeBSD Jails, Solaris Zones, etc.

Introducción a los contenedores

- El gran hito: inclusión de cgroups y namespaces en el kérnel linux (a partir de 2007)
  - **cgroups** (límite de memoria, cpu, I/O o red para un proceso y sus hijos)

    https://wiki.archlinux.org/index.php/Cgroups
  - **namespaces**: proporcionan un punto de vista diferente a un proceso (interfaces de red, procesos, usuarios, etc.)

    http://laurel.datsi.fi.upm.es/~ssoo/SOA/namespaces.html
- Todo esto unido a la expansión de linux en el centro de datos ha provocado la explosión en el uso de contenedores de los últimos años.

Cgroups, namespaces, and beyond: what are containers made from?

Arquitectura de microservicios. Aplicación monolítica

- Todos los componentes en el mismo nodo
- Escalado vertical
- Arquitectura muy sencilla
- Suele utilizarse un solo lenguaje de programación (o un conjunto pequeño de ellos)
- No pueden utilizarse diferentes versiones de un lenguaje de programación a la vez
- Interferencias entre componentes en producción
- Complejidad en las actualizaciones. Puede ocasionar paradas en producción
- Infraestructura estática y fija por años
- Típicamente la aplicación no es tolerante a fallos

Arquitectura de microservicios. Aplicación distribuida

- Idealmente un componente por nodo
- Escalado horizontal
- Arquitectura más compleja
- Menos interferencias entre componentes
- Mayor simplicidad en las actualizaciones
- Diferentes enfoques no excluyentes: SOA, cloud native, microservicios, ...

Arquitectura de microservicios. SOA

- SOA: Service Oriented Architecture
- Servicios independientes
- Múltiples tecnologías, lenguajes y/o versiones interactuando
- Comunicación vía SOAP
- Uso de XML, XSD y WSDL
- Colas de mensajes
- Se relaciona con aplicaciones corporativas
- Se le achaca mucha complejidad y no ha terminado de extenderse

Arquitectura de microservicios. Cloud Native Application

- Énfasis en la adaptación de la infraestructura a la demanda
- Uso extensivo de la elasticidad: Infraestructura dinámica
- Aplicaciones resilientes
- Elasticidad horizontal
- Automatización
- Puede ser complejo de implementar en una aplicación

Arquitectura de microservicios. Microservicios

- Deriva del esquema SOA
- No existe una definición formal, ni WSDL, XSD, etc. Solución más pragmática
- Servicios llevados a la mínima expresión (idealmente un proceso por nodo): Microservicios
- Comunicación vía HTTP REST y colas de mensajes, ¿gRPC?
- Relacionado con procesos ágiles de desarrollo, facilita enormemente la actualizaciones de versiones, llegando incluso a la entrega continua o despliegue continuo.
- Suele implementarse sobre contenedores
- Aumento de la latencia entre componentes
- Puede utilizar características de “cloud native application”

Arquitectura de microservicios. Ejemplo

- OpenStack
  - Cada componente es un microservicio que puede ejecutarse en un nodo independiente
  - kolla-ansible: Despliegue de OpenStack con ansible en múltiples contenedores docker
  - https://elatov.github.io/2018/01/openstack-ansible-and-kolla-on-ubuntu-1604/

Introducción a los contenedores. LXC

- Características
  - Espacios de nombres del kérnel
  - Apparmor y SELinux
  - Chroots (pivot root)
  - Kernel capabilities
  - CGroups
- Comienza su desarrollo en 2008
- Licencia LGPL
- Desarrollado principalmente por Canonical
- http://linuxcontainers.org

Introducción a los contenedores. LXC

- Pertenece a los denominados contenedores de sistemas
- Gestiona contenedores directamente sin “adornos” y a bajo nivel
- No compite con docker sino con otros sistemas de virtualización
- No hay nuevos conceptos, es otro sistema de virtualización en la que todos los contenedores tienen el mismo kérnel
- No hay nuevos paradigmas de uso
- Utiliza pivot root para definir el directorio raíz del contenedor en un directorio
- No hay que definir un LXCFile ni nada que se parezca ;)
- Para acceder al contenedor utilizamos ssh(!)
- Instalación simple: apt install lxc

Introducción a los contenedores. Docker

- “docker”: estibador
- Pertenece a los denominados contenedores de aplicaciones
- Gestiona contenedores a alto nivel proporcionando todas las capas y funcionalidad adicional
- Nuevo paradigma. Cambia completamente la forma de desplegar y distribuir una aplicación
- Docker: build, ship and run
- Lo desarrolla la empresa Docker, Inc.
- Instalación y gestión de contenedores simple
- El contenedor ejecuta un comando y se para cuando éste termina, no es un sistema operativo al uso, ni pretende serlo

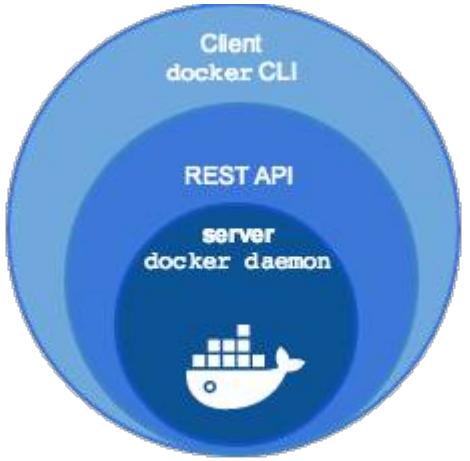
Introducción a los contenedores. ¿Cómo funciona docker?

- Nos descargamos una imagen disponible de un registro público o privado (p. ej. docker hub)
  - docker pull ubuntu:latest
- Podemos ver en qué consiste esta imagen:
  - docker inspect ubuntu:latest
- En este caso está compuesta de varias capas, como podemos ver con:
  - docker history ubuntu:latest
- Cuando ejecutamos un contenedor, estamos instanciando una imagen
  - docker run -d -p 8080:80 nginx
- Podemos ver las características del contenedor creado
  - docker inspect <nombre o ID>
- Podemos ver los logs del contenedor creado
  - docker logs <nombre o ID>
- Para“acceder” al contenedor necesitamos que ejecute una shell:
  - docker run -it ubuntu /bin/bash

Introducción a los contenedores. ¿Cómo funciona docker?

- Desde docker 1.13 se puede ejecutar un comando en un contenedor que esté corriendo:
  - docker exec -it <nombre o ID> /bin/bash
- Dockerfile: Añadir capas a un contenedor base
  - Ejemplo1
- ENV
  - Variables de entorno
  - Pueden ser modificadas con --env VARIABLE=VALOR
- RUN, CMD, ENTRYPOINT
  - RUN Ejecuta el comando y añade una nueva capa
  - ENTRYPOINT Comando principal a ejecutar. Se puede modificar con --entrypoint
  - CMD Comando por defecto. Si existe ENTRYPOINT se añade como argumento. Se puede modificar al ejecutar
    - docker history ubuntu (CMD es /bin/bash)
    - docker run -it ubuntu /bin/dash
  - Ejemplo2

Introducción a los contenedores. Software relacionado con docker

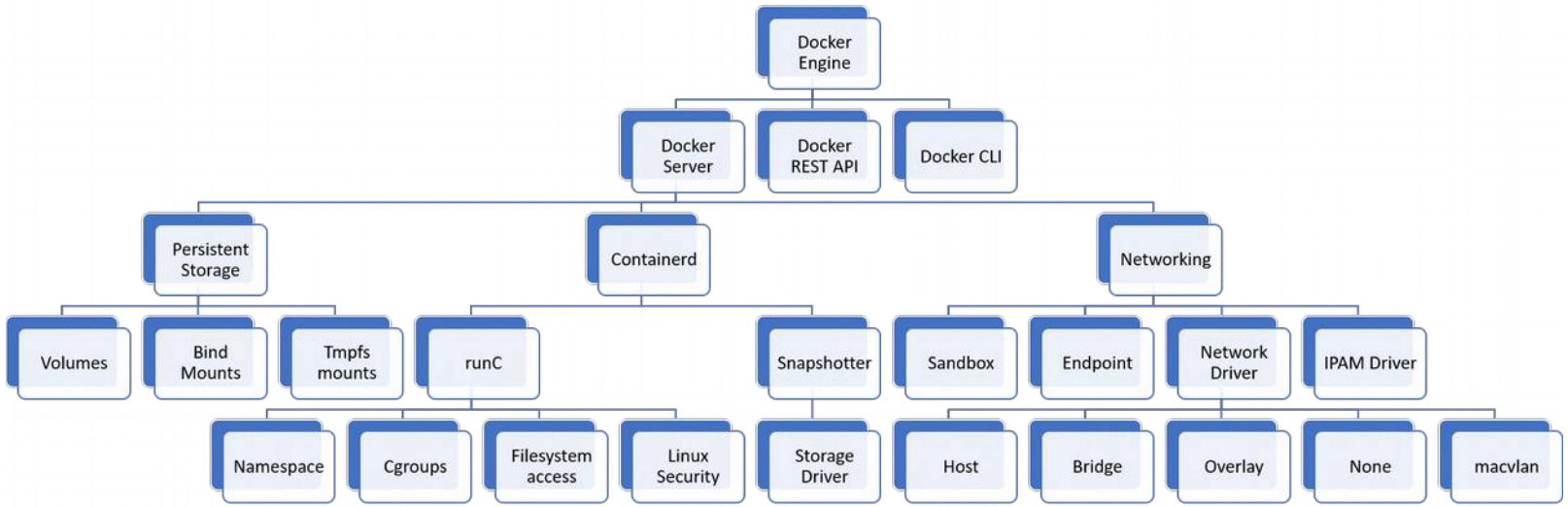


- docker engine
  - docker API
  - demonio docker
  - docker CLI
- docker-machine
  - Gestiona múltiples docker engine
- docker compose
  - Para definir aplicaciones que corren en múltiples contenedores
  - Ejemplo: https://github.com/bitnami/bitnami-docker-wordpress/blob/master/docker-compose.yml
- docker swarm
  - Orquestador de contenedores

Introducción a los contenedores. Evolución del software de docker

- El dilema de docker inc. entre el éxito y el negocio
- OCI: Open Containers Initiative
  - runtime-spec: http://www.github.com/opencontainers/runtime-spec
  - image-spec: http://www.github.com/opencontainers/image-spec
- El cambio en docker
  - moby
  - docker CE
  - docker EE
  - runc
  - containerd

Introducción a los contenedores. Arquitectura de docker engine



- Si vamos a utilizar un orquestador diferente a docker swarm, ¿necesitamos docker engine o containerd?
- runc es equivalente a lxc
- Tanto runc como containerd son proyectos de software libre hoy en día independientes de docker inc.

Alternativas a docker

- rkt, inicialmente desarrollado por CoreOS. Actualmente dentro de la Cloud Native Computing Foundation: https://github.com/rkt/rkt y enfocado a ser una alternativa a containerd
- cri-o. Creado por Red Hat como alternativa a containerd https://cri-o.io/ y pensado solo para funcionar integrado en kubernetes
- Pouch. Desarrollado por Alibaba. https://pouchcontainer.io/#/
- ¿Micromáquinas virtuales?
  - Kata containers. MVs ligeras para para proporcionar mayor aislamiento. https://katacontainers.io/
  - Nemu: https://github.com/intel/nemu
  - Firecraker (AWS): https://github.com/firecracker-microvm/firecracker

Introducción a los contenedores. Ciclo de vida en el despliegue de aplicaciones con docker

- Docker ha tenido un gran éxito en el desarrollo de software en su uso como plataforma de desarrollo, pruebas y puesta en producción y como alternativa en el empaquetamiento, distribución y despliegue de aplicaciones
- Ciclo de vida tipo en docker
  - Se crea un Dockerfile con las dependencias necesarias: Sistema base, runtime, etc.
  - Se añade la propia aplicación, típicamente desde un repositorio git
  - Se construye la imagen del contenedor con la aplicación y se sube a un registro público o privado
  - Cada vez que se crea una nueva versión de la aplicación, se crea una nueva imagen de docker
  - ¿Qué hacemos con los cambios entre versiones?
  - ¿Cómo hacemos los cambios en producción?
  - Respuestas muy diferentes dependiendo de las características y requisitos de la aplicación
    - Aplicación en un contenedor
    - Aplicación en múltiples contenedores
    - Aplicación totalmente distribuida en múltiples nodos

Introducción a los contenedores. Limitaciones de docker

- ¿Cómo se balancea la carga entre múltiples contenedores iguales?
- ¿Cómo se conectan contenedores que se ejecuten en diferentes demonios de docker?
- ¿Se puede hacer una actualización de una aplicación sin interrupción?
- ¿Se puede variar a demanda el número de réplicas de un determinado contenedor?
- ¿Es posible mover la carga entre diferentes nodos?
- Las respuestas a estas preguntas y otras similares tiene que venir de un orquestador de contenedores

Introducción a los contenedores. ¿kubernetes o docker swarm?

- Hace algunos años había tres alternativas para la orquestación. Todos proyectos de software libre
  - Docker swarm
  - Apache Mesos
  - Kubernetes
- Hoy en día se acepta unánimemente que el vencedor ha sido kubernetes. ¿Por qué?
  - Proyecto de fundación con gran cantidad de empresas implicadas
  - Los recelos iniciales del control de Google se disiparon con la CNCF
  - La versión inicial estaba prácticamente lista para poner en producción
  - Se ha desarrollado un interesante ecosistema de aplicaciones complementarias