Kubernetes For Storm

Noviembre 2020

Ponentes

José Domingo Muñoz Rodríguez
Alberto Molina Coballes

Índice del curso

https://github.com/iesgn/kubernetes-storm

Unidad 1

Despliegue de aplicaciones en contenedores

Índice

- Contenedores
- Microservicios
- Tecnologías subyacentes. LXC, docker

Contenedores

Contenedores

- ¿Para qué sirve un sistema operativo?¿Qué es un proceso?
- Compartir o no compartir, ésa es la cuestión: .so., dependencias
- ¿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.

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
- **cgroupsv2** (rootless containers)

  https://medium.com/nttlabs/cgroup-v2-596d035be4d7
- **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?

Despliegue de apliacaciones

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
- Normalmente usan infraestructura estática y fija por años
- Típicamente la aplicación no es tolerante a fallos

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, ..

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

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

Aplicación con o sin estado

- En una aplicación con estado la operación se realiza en un contexto de un estado anterior
- Una aplicación con estado típicamente requerirá que se interactúe con el mismo servidor
- Suelen hacer uso de algún tipo de almacenamiento permanente
- Para escalar aplicaciones con estado se pueden utilizar clusters de alta disponibilidad
- Ejemplo: Webmail

- En una aplicación sin estado, cada operación es independiente de las anteriores o posteriores
- Puede interactuar cada vez con un servidor diferente
- No es necesario que se utilice almacenamiento permanente o puede ser solo de lectura
- Para escalar aplicaciones sin estado se pueden utilizar clusters de balanceo de carga
- Ejemplo: Búsqueda en un buscador de Internet

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
- Muy adecuada para aplicaciones sin estado
- Aumento de la latencia entre componentes
- Puede utilizar características de “cloud native application”

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/

Tecnologías de 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

LXC

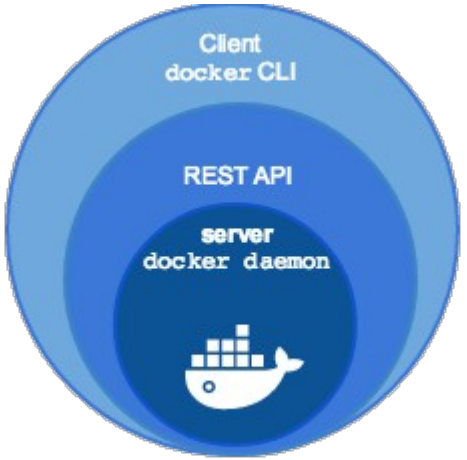
- Pertenece a los denominados contenedores de sistemas
- Gestiona contenedores directamente sin “adornos” y a bajo nivel
- No compite con docker sino con otros sistemas de virtualización
- No hay nuevos conceptos, es otro sistema de virtualización en la que todos los contenedores tienen el mismo kérnel
- 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
- LXD: LXC + demonio + CLI unificado + imágenes

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
- Escrito en go
- Software libre (ha ido cambiando con el tiempo)

Docker

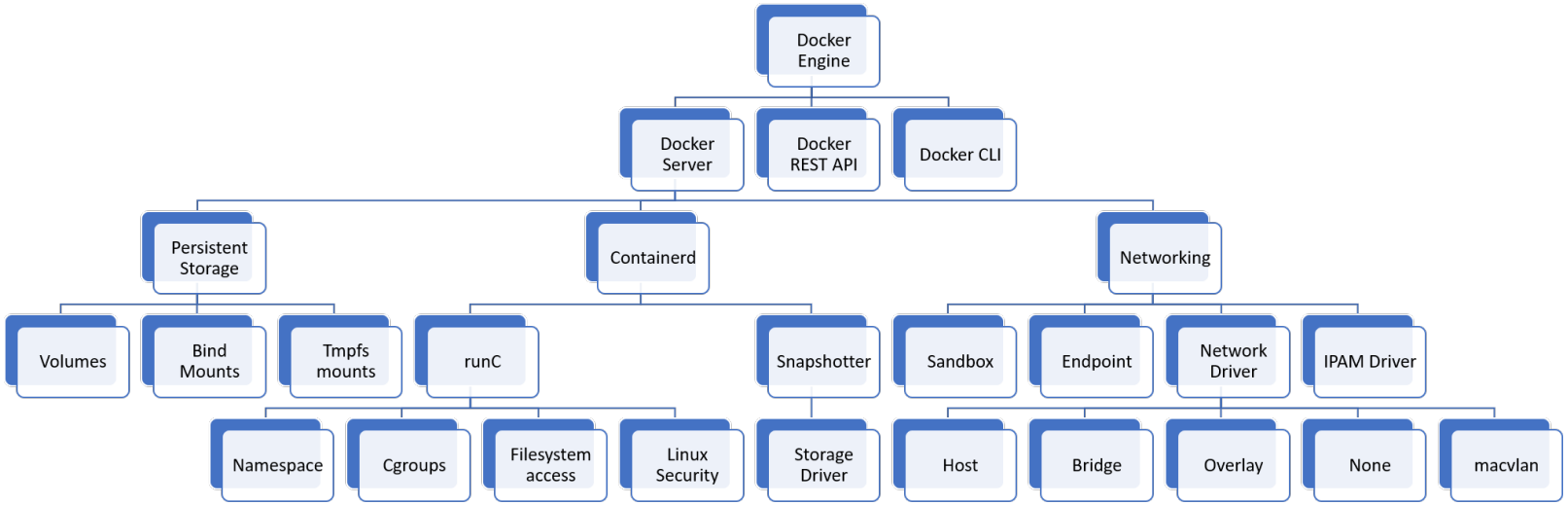
- docker engine
  - demonio docker
  - docker API
  - docker CLI
- docker-machine
  - Gestiona múltiples docker engine
- docker compose
  - Para definir aplicaciones que corren en múltiples contenedores
  - Ejemplo: https://github.com/bitnami/bitnami-docker-wordpress/blob/master/docker-compose.yml
-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



Docker. Evolución del proyecto

- 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
- distribution-spec: https://github.com/opencontainers/distribution-spec
- El cambio en docker
  - Moby (proyecto de comunidad) ← docker.io de debian
  - docker CE (docker engine proporcionado por Docker inc)
  - docker EE (docker engine + servicios de Docker inc)
  - runc (OCI) https://github.com/opencontainers/runc
  - containerd (CNCF) https://github.com/containerd/containerd

Docker. Componentes



- 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
- Podman. Creado por Red Hat como alternativa a docker https://podman.io
- Pouch. Desarrollado por Alibaba como alternativa a docker. 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

Ciclo de vida de una aplicación en 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

Limitaciones 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
- ¿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

¿Qué orquestador de contenedores?

- Hace algunos años había tres alternativas para la orquestación. Todos proyectos de software libre
  - Docker swarm
  - Apache Mesos
  - Hashicorp Nomad
  - Kubernetes
  - ...
- Hoy en día se acepta generalmente 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
- El resto de proyectos siguen activos, como alternativas más sencillas a k8s o en su propio nicho