October 4, 2022

Docker te permite construir, distribuir y ejecutar cualquier aplicación en cualquier lado

1 Problemáticas del desarrollo de software

Una razón por la que Docker es tan popular es que cumple la promesa de "desarrollar una vez, ejecutar en cualquier lugar". Docker ofrece una forma sencilla de empaquetar una aplicación y sus dependencias de tiempo de ejecución en un solo contenedor; también proporciona una abstracción en tiempo de ejecución que permite que el contenedor se ejecute en diferentes versiones del kernel de Linux.
Usando Docker, un desarrollador puede crear una aplicación en contenedor en su estación de trabajo, y luego implementar fácilmente el contenedor en cualquier servidor habilitado para Docker. No es necesario volver a probar o volver a sintonizar el contenedor para el entorno del servidor, ya sea en la nube o en las instalaciones.

1.1 Construir

Escribir código en la máquina del desarrollador. (Compile, que no compile, arreglar el bug, compartir código, etc.).

- Entorno de desarrollo (paquetes)
- Dependencias (Frameworks, bibliotecas)
- Equivalencia de entornos de desarrollo (compartir el código)
- Equivalencia con entornos productivos (pasar a producción)
- Servicios externos (integración con otros servicios ejem: base de datos)

1.2 Distribuir

Llevar la aplicación donde se va a desplegar (Transformarse en un artefacto)

- Output de build heterogéneo (múltiples compilaciones)
- Acceso a servidores productivos (No tenemos acceso al servidor)
- Ejecución nativa vs virtualizada
- Entornos Serverless

1.3 Virtualización

Permite atacar en simultáneo los tres problemas del desarrollo de software profesional.
Virtualización.

- PESO: En el orden de los GBs. Repiten archivos en común. Inicio lento
- COSTO DE ADMINISTRACION: Necesita mantenimiento igual que cualquier otra computadora.
- MULTIPLES DE FORMATO: VDI, VMDK, VHD, raw, etc

Containerización
Que es un contenedor ?
Es una agrupación de procesos.
Es una entidad lógica, no tiene el limite estricto de las máquinas virtuales, emulación del sistema operativo simulado por otra más abajo.

Ejecuta sus procesos de forma nativa.
Los procesos que se ejecutan adentro de los contenedores ven su universo como el contenedor lo define, no pueden ver mas allá del contenedor, a pesar de estar corriendo en una maquina más grande.
No tienen forma de consumir más recursos que los que se les permite. Si esta restringido en memoria ram por ejemplo, es la única que pueden usar.
A fines prácticos los podemos imaginar cómo maquinas virtuales, pero NO lo son. Máquinas virtuales livianas.
Docker corre de forma nativa solo en Linux.
Sector del disco: Cuando un contenedor es ejecutado, el daemon de docker le dice, a partir de acá para arriba este disco es tuyo, pero no puedes subir mas arriba.
Docker hace que los procesos adentro de un contenedor este aislados del resto del sistema, no le permite ver más allá.
Cada contenedor tiene un ID único, también tiene un nombre.

- Flexibles
- Livianos
- Portables
- Bajo acoplamiento
- Escalables
- Seguros

Virtualizacion vs Containerización

- Virtualización: A diferencia de un contenedor, las máquinas virtuales ejecutan un sistema operativo completo, incluido su propio kernel.
- Containerización: Un contenedor es un silo aislado y ligero para ejecutar una aplicación en el sistema operativo host. Los contenedores se basan en el kernel del sistema operativo host (que puede considerarse la fontanería del sistema operativo), y solo puede contener aplicaciones y algunas API ligeras del sistema operativo y servicios que se ejecutan en modo de usuario.

¿Cuál es el principal problema que intenta resolver la virtualización?
Atacar en simultaneo los tres grandes problemas de la ingeniería del software.
Los contenedores son la estandarización para construir y desplegar software.

1.4 Distribuir

Llevar la aplicación donde se va a desplegar (Transformarse en un artefacto)

- Output de build heterogéneo (múltiples compilaciones)
- Acceso a servidores productivos (No tenemos acceso al servidor)
- Ejecución nativa vs virtualizada
- Entornos Serverless

1.5 Ejecutar

Implementar la solución en el ambiente de producción (Subir a producción)

- Dependencia de aplicación (paquetes, runtime)
- Compatibilidad con el entorno productivo (sistema operativo poco amigable con la solución)
- Disponibilidad de servicios externos (Acceso a los servicios externos)
- Recursos de hardware (Capacidad de ejecución - Menos memoria, procesador más debil)

1.6 Instalar y usar Docker en Ubuntu

- sudo apt update
- sudo apt install apt-transport-https ca-certificates curl software-properties-common
- curl -fsSL https://download.docker.com/linux/ubuntu/gpg — sudo apt-key add -
- sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu focal stable"1cm
- sudo apt update
- apt-cache policy docker-ce
- sudo apt install docker-ce
- sudo systemctl status docker

1.7 Ejecutar el comando Docker sin sudo (opcional)

sudo usermod -aG docker $USERsu-$USER id -nG sudo usermod -aG docker username

2 Comandos Docker

- Arrancar un contenedor asignándole un nombre -¿ docker run ¡image_container¿ -¿ docker run hello-world
- Arrancar un contenedor asignándole un nombre -¿ docker run –name ¡name_container¿ ¡image_container¿ -¿ docker run –name contenedor_test ubuntu
- Arrancar un contenedor con una terminal interactiva. Pasándole una shell para acceder al contenedor -¿ docker run -it ¡image_container¿ ¡shell¿ -¿ 0docker run -it ubuntu bash
- Arrancar un contenedor. Mapeando un puerto del host a un puerto del contenedor. -¿ docker run -p ¡host_port¿:¡container_port¿ ¡image_container¿ -¿ docker run -p 8080:80 nginx
- Igual que el ejemplo anterior pero dejándolo en segundo plano. -¿ docker run -p 8080:80 -d nginx
- Arrancar un contenedor. Que tras terminar su periodo de vida. Será eliminado automáticamente. -¿ docker run –rm ¡image_container¿, docker run -p 8080:80 -d –rm nginx
- Arrancar un contenedor con un volume -¿ docker run -v ¡volume_name¿:¡mount_point¿ :¡options¿ ¡image_container¿ docker run -v test:/apps:rw nginx * Volume -¿ test * Punto de montaje en el contenedor -¿ /apps * Opciones -¿ rw (Lectura y escritura)

- Arrancar un contenedor con un bind mount. docker run -v ¡shared$_folder$ >:< $mount_point$ >:< $options$ >< $image_container$ > $ejemplo * Rutadelhostacompartir- >$ $/home/application * Puntodemontajeenelcontenedor- > /apps * Opciones- > ro(Sololectura)$

  docker run -v /home/application:/apps:ro ubuntu

- Arrancar un contenedor con tmpfs. docker run –mount type=tmpfs,destination=¡mount$_point$ > $,tmpfs - mode =< permisos >, tmpfs - size =< bytes_size > < image_container >$

  Ejemplo * Punto de montaje en el contenedor -¿ /temporal * Permisos -¿ Todos los permisos solo para el propietario. * Tamaño del FS -¿ 21474836480 bytes = 20G docker run –mount type=tmpfs,destination=/temporal,tmpfs-mode=700,tmpfs-size=21474836480 nginx

- Lista de los contenedores activos. -¿ docker ps
- Lista de todos los contenedores activos e inactivos del sistema. -¿ docker ps -a
- Lista los ID de todos los contenedores. -¿ docker ps -aq
- Inspeccionar la data de un contenedor, por su ID -¿ docker inspect ¡id_container¿
- Inspeccionar la data de un contenedor, por su nombre -¿ docker inspect ¡name_container¿
- Aplicando filtros. Por ejemplo buscando las variables de entorno: -¿ docker inspect -f ' json .Config.Env ' ¡name_container¿
- Ver los logs del contenedor. -¿ docker logs ¡name_container¿
- Eliminar un contenedor por ID -¿ docker rm ¡name_container¿
- Eliminar un contenedor aunque este arriba. Forzándolo -¿ docker rm -f ¡id_container¿
- Eliminar todos los contenedores que no esten arriba a la vez. -¿ docker rm $(docker ps -aq)