Universidad Latina de Costa Rica

BIT-28 Sistemas Operativos II

Laboratorio #2 – Jaime Fung Delgado

**Contenedores**

**Instrucciones**

Trabaje individualmente. Siga los siguientes puntos paso a paso. Debe trabajar en Linux (Ubuntu). Debe entregar un reporte al final de la clase en el Moodle.

Para el presente laboratorio debe contar con al menos una máquina Ubuntu 18.

**Objetivos**

¿Qué son los contenedores?

Comprender las diferencias entre Docker y LXD

Instalar Docker

Administrar contenedores Docker

Administrar Docker con interfaz gráfica Portainer

**Introducción**

Los contenedores permiten empaquetar aplicaciones y sus dependencias en entornos aislados y reproducibles, con menor sobrecarga que la virtualización tradicional. En este laboratorio se busca entender los conceptos fundamentales, distinguir entre Docker y LXD, y practicar la instalación y administración de contenedores en Linux.

**¿Qué son los contenedores?**

Un contenedor es una forma de virtualización a nivel de sistema operativo que permite ejecutar aplicaciones junto con todas sus dependencias en un entorno aislado y portable. A diferencia de una máquina virtual, que requiere emular un hardware completo y ejecutar un sistema operativo invitado encima del host, los contenedores comparten el mismo kernel de Linux y aprovechan funciones como *namespaces* y *cgroups* para proveer aislamiento y control de recursos.

En términos prácticos, un contenedor se comporta como una "caja" en la que vive una aplicación junto con sus bibliotecas, configuraciones y binarios necesarios para funcionar. Esto garantiza que, sin importar en qué sistema Linux se ejecute el contenedor, el resultado será el mismo, lo que reduce el problema de compatibilidad conocido como “funciona en mi máquina, pero no en la tuya”.

Las características más destacadas de los contenedores son:

* Ligereza: al no necesitar un sistema operativo invitado, consumen menos recursos que una máquina virtual.
* Rapidez: pueden iniciarse en segundos, lo que favorece la escalabilidad y el despliegue continuo.
* Portabilidad: el mismo contenedor puede ejecutarse en diferentes entornos (desarrollo, pruebas, producción).
* Reproducibilidad: garantizan que el software siempre se ejecute en el mismo entorno.
* Escalabilidad: permiten ejecutar múltiples instancias de manera eficiente.

Además, los contenedores han transformado la forma en que las empresas despliegan software, ya que son la base de arquitecturas modernas como microservicios y orquestadores como Kubernetes. Gracias a su eficiencia, son ampliamente utilizados en la nube, en CI/CD, y en aplicaciones que requieren portabilidad y rapidez.

**Docker vs LXD**

LXD (pronunciado *Lex-D*) tiene sus orígenes en LXC (*Linux Containers*, pronunciado *Lex-C*). LXC fue una de las primeras implementaciones de contenedores en Linux, basada en las capacidades del kernel como *cgroups* (grupos de control) y *namespaces* para aislar procesos y administrar recursos. Esta tecnología fue diseñada con el objetivo de proveer entornos que se comportan de manera similar a un sistema operativo completo, pero sin la sobrecarga de una máquina virtual.

LXD surge como una evolución de LXC, desarrollado inicialmente por Canonical (la empresa detrás de Ubuntu). Aunque está estrechamente asociado a Ubuntu, hoy en día se puede instalar en cualquier distribución de Linux que soporte *snap*. LXD no reemplaza a LXC, sino que lo extiende y lo mejora al añadir una capa de administración más completa, ofreciendo funcionalidades adicionales como:

* **Instantáneas**: permiten guardar estados del contenedor y restaurarlos cuando sea necesario.
* **Migración en vivo**: posibilidad de mover contenedores de un host a otro con interrupciones mínimas.
* **Mejor gestión de recursos**: herramientas de configuración avanzadas que permiten controlar memoria, CPU, redes y almacenamiento.

La manera más sencilla de entender LXD es considerarlo como una capa de gestión sobre LXC, que hace que trabajar con contenedores de sistema sea mucho más amigable y potente.

**Diferencias clave entre Docker y LXD**

* **Modelo de virtualización**: Docker se enfoca en *application containers*, es decir, contenedores diseñados para ejecutar un proceso o servicio específico, mientras que LXD se orienta a *system containers*, que se asemejan más a máquinas virtuales ligeras capaces de ejecutar múltiples procesos y un sistema init completo como *systemd*.
* **Transaccionalidad**: Docker trabaja con imágenes basadas en capas inmutables que se apilan unas sobre otras, lo que permite reconstruir aplicaciones fácilmente. LXD, en cambio, maneja imágenes de sistemas operativos completos y tiende a ser más estático en este aspecto.
* **Compatibilidad multiplataforma**: Docker funciona en Linux, macOS y Windows, lo que lo hace más universal. LXD, por su parte, está más enfocado en entornos Linux.
* **Repositorio de imágenes**: Docker cuenta con Docker Hub, una plataforma donde millones de imágenes están disponibles públicamente. Esto acelera el desarrollo y despliegue. LXD tiene su propio repositorio de imágenes, aunque mucho más limitado y enfocado en sistemas base.
* **Uso recomendado**: Docker resulta ideal para empaquetar aplicaciones, microservicios y despliegues en la nube. LXD, por su parte, es más útil para crear entornos de prueba completos, simular servidores y ejecutar aplicaciones tradicionales que requieren un sistema completo.

Docker y LXD no son excluyentes: ambos cumplen el objetivo de aislar aplicaciones en entornos seguros y portables, aunque con enfoques distintos. Docker es más adecuado para arquitecturas modernas basadas en microservicios, mientras que LXD resulta útil en entornos donde se requiere un sistema operativo completo en un contenedor. Aprender ambos proporciona al administrador de sistemas mayor flexibilidad para elegir la herramienta adecuada en cada escenario.

**Instalación de Docker en Ubuntu**

1. **Instalacion**

# Configurar apt el repositorio de Docker.

# Agregar la clave GPG oficial de Docker:

sudo apt-get update

sudo apt-get install ca-certificates curl

sudo install -m 0755 -d /etc/apt/keyrings

sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.asc

sudo chmod a+r /etc/apt/keyrings/docker.asc

# Agregar el repositorio a las fuentes de Apt:

echo \

"deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc] https://download.docker.com/linux/ubuntu \

$(. /etc/os-release && echo "${UBUNTU\_CODENAME:-$VERSION\_CODENAME}") stable" | \

sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

sudo apt-get update

# Instalar los paquetes de Docker.

sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin

# Verifique que la instalación sea exitosa ejecutando la hello-world imagen:

sudo docker run hello-world

# Crear el grupo docker

sudo groupadd docker

#Agrega tu usuario al grupo Docker

sudo usermod -aG docker $USER

# comando para activar los cambios en los grupos

newgrp Docker

# Verifique que pueda ejecutar comandos de docker sin sudo.

docker run hello-world

1. **Administración básica de contenedores**

Descargar una imagen:

docker pull apache

Ejecutar un contenedor:

docker run -d -p 8080:80 apache

Ver contenedores en ejecución:

docker ps

Detener un contenedor:

docker stop <ID>

Eliminar un contenedor:

docker rm <ID>

1. **Portainer (interfaz gráfica para Docker)**

**Instalar y ejecutar con un solo contenedor:**

docker volume create portainer\_data

docker run -d -p 9443:9443 --name portainer \

--restart=always \

-v /var/run/docker.sock:/var/run/docker.sock \

-v portainer\_data:/data \

portainer/portainer-ce:latest

**Acceder desde el navegador en:**

https://localhost:9443

**Uso de Docker**

Un contenedor es básicamente un proceso que corre de forma aislada y con su propio sistema de archivos. Ese sistema lo proporciona una **imagen Docker**, que incluye todo lo necesario para ejecutar una aplicación (código, librerías y dependencias).

Las **imágenes** se descargan de **Docker Hub** o se pueden crear con un **Dockerfile**. Están hechas de capas, y esas capas se pueden reutilizar entre diferentes imágenes.

Cuando un contenedor se elimina, se pierden los datos que estaban dentro. Para evitarlo se usan **volúmenes persistentes**, que permiten guardar información fuera del contenedor.

**Ejemplo: Servidor web con base de datos**

1. Crear una red propia para los contenedores:

docker network create misitio-net

1. Crear un contenedor de base de datos con volumen persistente:

docker run -d --name mysql11 \

--network misitio-net \

-e MYSQL\_ROOT\_PASSWORD=1234 \

-e MYSQL\_DATABASE=misitiodb \

-v websitedbvolume:/var/lib/mysql \

mariadb:latest

1. Confirmar volumen:

docker volume ls

docker volume inspect websitedbvolume

1. Crear directorio para los archivos web:

mkdir public\_html

1. Levantar el contenedor con el servidor web:

docker run -d --name sitio \

--network misitio-net \

-v $(pwd)/public\_html:/var/www/html \

-p 8080:80 \

php:apache

**Acceder con:**

http://localhost:8080