Introducción a contenedores

Despliegue de aplicaciones web

Tabla de Contenidos

Servidores de aplicaciones

Un servidor de aplicaciones es un software que proporciona un entorno en el que las aplicaciones web pueden ejecutarse y servirse a través de Internet. Su función principal es gestionar las solicitudes de los clientes, procesar la lógica de la aplicación y devolver las respuestas al cliente. Algunos servidores de aplicaciones populares incluyen Apache Tomcat o Microsoft IIS.

- **Apache Tomcat** Servidor de aplicaciones web orientado a ejecutar aplicaciones Java Servlet o JSP. Su principal diferencia con Apache HTTP Server es que está diseñado específicamente para alojar aplicaciones Java y proporcionar un entorno para su ejecución a través de HTTP o HTTPS.
- **Microsoft Information Services (IIS)** Servidor web desarrollado por Microsoft para sistemas Windows. IIS proporciona un entorno de servidor web para alojar y administrar sitios web y aplicaciones web en servidores Windows.

Instalación y configuración básica de un servidor de aplicaciones

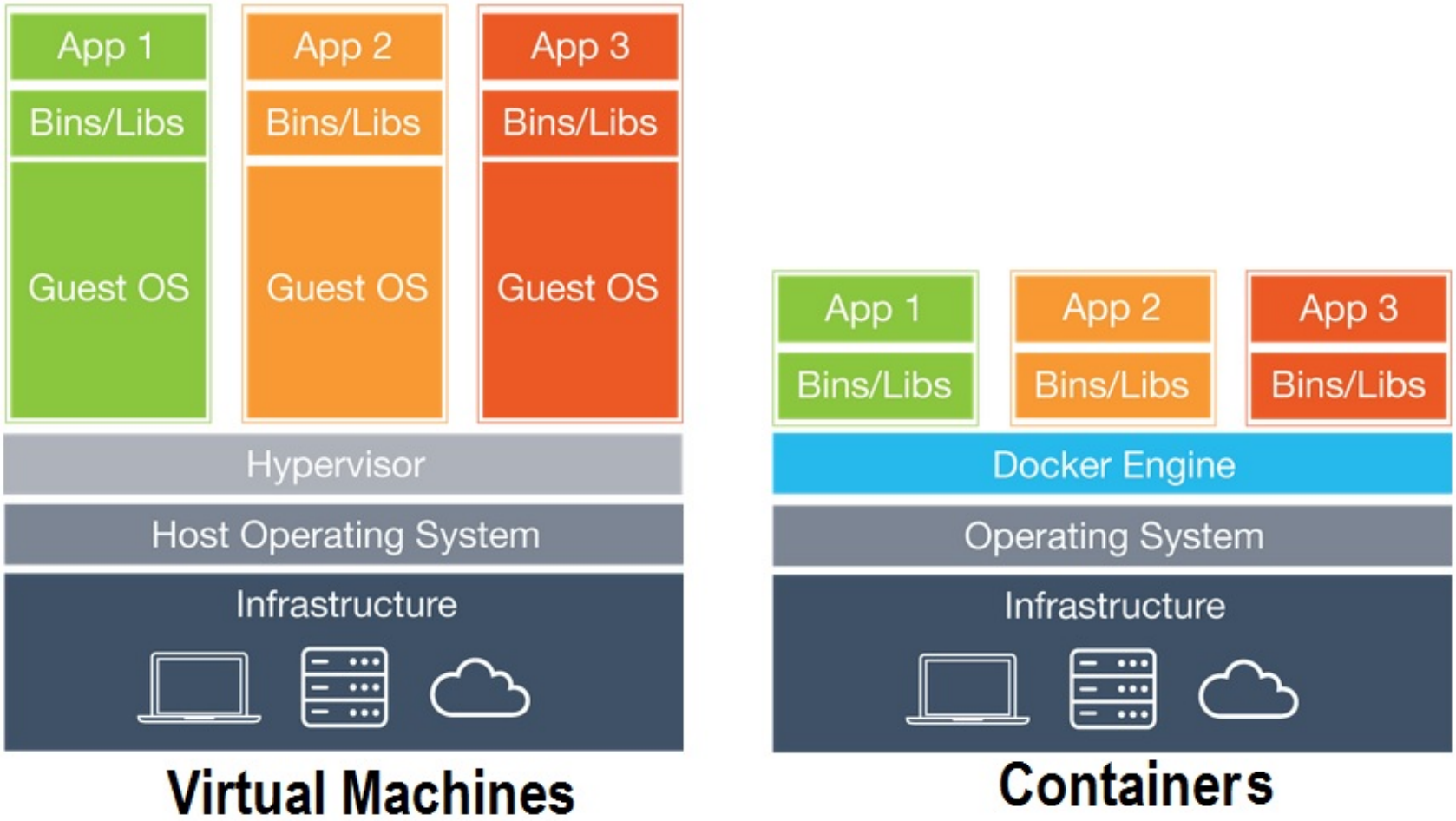
1. **Seleccionar un servidor de aplicaciones** Descarga el software desde el sitio web oficial del servidor de aplicaciones.
2. **Instalación** Sigue las instrucciones de instalación específicas para tu servidor de aplicaciones y sistema operativo.
3. **Configuración inicial** Configura opciones básicas como el puerto en el que escuchará el servidor, la gestión de logs y otros parámetros según las necesidades de tu aplicación.
4. **Despliegue de aplicaciones** Coloca tu aplicación web en el directorio adecuado del servidor.
5. **Inicio del servidor** Inicia el servidor de aplicaciones. Esto generalmente se hace ejecutando un comando o script específico.
6. **Pruebas** Accede a tu aplicación en un navegador web usando la dirección y el puerto configurados previamente para verificar que todo funcione correctamente.

Contenedores

Los contenedores son entornos de ejecución ligeros y aislados que encapsulan aplicaciones y sus dependencias.

- **Aislamiento** Los contenedores proporcionan un nivel de aislamiento para las aplicaciones, lo que significa que pueden ejecutarse de manera independiente sin interferir con otras aplicaciones o el sistema operativo subyacente.
- **Portabilidad** Los contenedores son portátiles y pueden ejecutarse en cualquier entorno que admita el motor de contenedores utilizado.
- **Eficiencia** Los contenedores comparten el kernel del sistema operativo del host, lo que los hace más eficientes en recursos que las máquinas virtuales tradicionales. Puedes ejecutar múltiples contenedores en un solo servidor.

Contenedores



Docker

Docker es una plataforma de código abierto que permite a los desarrolladores crear, distribuir y ejecutar aplicaciones en contenedores. Los contenedores son entornos de ejecución ligeros y aislados que encapsulan aplicaciones y sus dependencias, lo que facilita la implementación consistente y portátil de aplicaciones en diferentes entornos.

- **Contenedores** Docker utiliza contenedores para empaquetar aplicaciones y sus dependencias en imágenes. Estas imágenes se pueden ejecutar de manera coherente en cualquier sistema que tenga Docker instalado.
- **Portabilidad** Las imágenes de Docker son portátiles y se pueden ejecutar en estaciones de trabajo locales, servidores en la nube o en cualquier lugar donde Docker esté disponible.
- **Dockerfile** Para crear imágenes de Docker personalizadas, los desarrolladores utilizan archivos de configuración llamados Dockerfiles. Estos archivos especifican las instrucciones para construir una imagen de contenedor.

Docker

- **DockerHub** Registro en línea de imágenes de Docker públicas y privadas. Los desarrolladores pueden compartir sus imágenes en Docker Hub o usar imágenes de la comunidad.
- **Orquestación** Para administrar un gran número de contenedores en clústeres, Docker ofrece herramientas como **Docker Compose** para aplicaciones multi-contenedor y Docker Swarm para orquestación de contenedores. **Kubernetes** también es ampliamente utilizado en combinación con Docker para la orquestación a gran escala.
- **Desarrollo y CI/CD** Docker se utiliza comúnmente en entornos de desarrollo y despliegue continuo (CI/CD) para crear entornos de desarrollo reproducibles y facilitar la entrega rápida de aplicaciones.
- **Escalabilidad** Docker permite escalar aplicaciones fácilmente mediante la creación de múltiples instancias de contenedores que se ejecutan en paralelo, lo que facilita la gestión de cargas de trabajo en crecimiento.

Instalando Docker

Docker es una plataforma multiplataforma que se puede ejecutar en sistemas UNIX, incluyendo Linux, así como en Windows. Para instalarlo en Windows, basta con ejecutar el instalador más reciente de Docker. Sin embargo, en sistemas como Ubuntu, es necesario seguir algunos pasos adicionales para su instalación:

1. Instalar tal y como pone en su **manual oficial**.
2. Crear un grupo de usuarios **docker**.

```
sudo groupadd docker
```

3. Añadir a tu usuario al grupo de docker.

```
sudo usermod -aG docker $USER
```

4. Activa los cambios en los grupos.

```
newgrp docker
```

5. Configurar docker para iniciar con el sistema

```
sudo systemctl enable docker.service
sudo systemctl enable containerd.service
```

Gestión básica

1. **Verificar la instalación** Utilizar comandos como `docker --version` para asegurar que docker se instaló correctamente y que versión se instaló.
2. **Ejecución de contenedores** Utilizar comando tales como `docker run` para crear y ejecutar contenedores basados en imágenes de aplicaciones. Prueba para verificar que funciona correctamente a ejecutar `docker run hello-world`.
3. **Gestión de imágenes** Usa comandos como `docker pull` para descargar imágenes de aplicaciones y `docker build` para crear tus propias imágenes personalizadas a partir del `Dockerfile`.

Comandos útiles

- `docker pull` Descarga una imagen de docker desde Docker Hub:

```
docker pull nombre_de_la_imagen:etiqueta
```

- `docker images` Lista todas las imágenes de docker disponibles en el sistema.
- `docker run` Crea y ejecuta un nuevo contenedor a partir de una imagen:

```
docker run -d --name mi_contenedor nombre_de_la_imagen
docker run -dit --name my-apache-app -p 8080:80 -v "$PWD
    ":/usr/local/apache2/htdocs/ httpd:2.4
```

- `docker ps` Lista los contenedores en ejecución.
- `docker ps -a` Lista todos los contenedores, incluidos los que están detenidos.

Comando útiles

- `docker start` Inicia un contenedor detenido:

```
docker stop mi_contenedor
```

- `docker stop` Detiene un contenedor en ejecución:

```
docker stop mi_contenedor
```

- `docker restart` Reinicia un contenedor:

```
docker restart mi_contenedor
```

- `docker rm` Elimina un contenedor:

```
docker rm mi_contenedor
```

Comando útiles

- `docker rmi` Elimina una imagen de docker:

```
docker rmi nombre_de_la_imagen:etiqueta
```

- `docker exec` Ejecuta un comando en un contenedor en ejecución:

```
docker exec -it mi_contenedor comando
```

- `docker logs` Ver los registros de un contenedor en ejecución:

```
docker logs mi_contenedor
```

- `docker build` Crea una nueva imagen de docker a partir de un Dockerfile:

```
docker build -t nombre_de_la_imagen:etiqueta
    ruta_del_Dockerfile
```

Comando útiles

- `docker network` Administra las redes de docker. Puedes crear redes personalizadas para conectar contenedores:

```
# Crea una red personalizada
docker network create mi_red
# Inicia contenedores y los conecta a la red que
    acabamos de crear
docker run -d --name contenedor_web1 --network mi_red
    imagen_web1
docker run -d --name contenedor_web2 --network mi_red
    imagen_web2
# Verificamos la conectividad ejecutando un ping
docker exec -it contenedor_web1 ping contenedor_web2
```

Comandos útiles

- `docker volume` Administra volúmenes de docker para persistencia de datos entre contenedores:

```
# Crea un volumen
docker volume create datos_mysql
# Inicia un contenedor con el volumen que acabamos de
    crear
docker run -d --name mysql-container -e
    MYSQL_ROOT_PASSWORD=pass -v datos_mysql:/var/lib/
    mysql mysql:latest
```

- `docker inspect` Muestra los detalles y metadatos de un contenedor o imagen:

Ejemplo de comando

Si tuviéramos que crear un contenedor basado en la imagen oficial de `PostgreSQL`, exponer el puerto `5432` en tu sistema host, darle un nombre personalizado `pgcontainer`, conectarlo a la red personalizada `red` y utilizar un volumen llamado `datapostgre` para persistencia de datos, se haría con el siguiente comando:

```
docker run -d -p 5432:5432 --name pgcontainer --network red
    -v datapostgre:/var/lib/postgresql/data postgres:latest
```

- `docker run` Este es el comando principal para crear y ejecutar un contenedor.
- `-d` Esta opción indica que el contenedor se ejecutará en segundo plano (modo daemon).
- `-p 5432:5432` Aquí, se están mapeando los puertos del contenedor al sistema host. El formato es `-p <puerto_host>:<puerto_contenedor>`. Estamos haciendo que el puerto 5432 del sistema host se comunique con el puerto 5432 del contenedor PostgreSQL.

Ejemplo de comando

- `--name pgcontainer` Establece un nombre personalizado para el contenedor como `pgcontainer`.
- `--network red` Conecta el contenedor a una red personalizada llamada `red`. Esto permite la comunicación entre contenedores en la misma red.
- `-v datapostgre:/var/lib/postgresql/data` Monta un volumen llamado `datapostgre` en el directorio `/var/lib/postgresql/data` del contenedor PostgreSQL. Esto asegura la persistencia de los datos de la base de datos.
- `postgres:latest` Esto especifica la imagen de Docker a partir de la cual se creará el contenedor. En este caso, estamos utilizando la imagen oficial de PostgreSQL y la etiqueta `latest`disponible en Docker Hub.

Dockerfile

Un Dockerfile es un archivo de texto plano que contiene una serie de instrucciones que Docker utiliza para construir una imagen de contenedor. Estas imágenes son la base para crear y ejecutar contenedores Docker.

- **Instrucción `FROM`** especifica la imagen base desde la cual se construirá la nueva imagen. Por ejemplo, puedes usar una imagen oficial de un sistema operativo como `ubuntu`, una imagen de una aplicación como `httpd`, o incluso otra imagen personalizada que hayas creado previamente. Desde aquí se recomienda el uso de `alpine`

```
FROM ubuntu:20.04
```

- **Instrucción `COPY` o `ADD`** Estas instrucciones se utilizan para copiar archivos desde el sistema de archivos del host al sistema de archivos del contenedor. Esto es útil para agregar archivos de código fuente, configuración o cualquier otro recurso necesario.

```
COPY superappgalaxial /app
```

Dockerfile

- **Instrucción** `RUN` Ejecuta comandos en el contenedor durante la construcción de la imagen. Puedes utilizarlo para instalar paquetes, configurar el entorno y realizar otras tareas de configuración.

```
RUN apt-get update && apt-get install -y nginx
```

- **Instrucción** `WORKDIR` Esta instrucción establece el directorio de trabajo (working directory) para las instrucciones posteriores en el Dockerfile. Es útil para especificar el directorio en el que se ejecutarán comandos.

```
WORKDIR /app
```

- **Instrucción** `EXPOSE` Especifica el puerto en el que el contenedor escuchará las conexiones. Sin embargo, esto no hace que el puerto sea accesible desde fuera del contenedor. Es más como una documentación de puertos.

```
EXPOSE 80
```

Dockerfile

- **Instrucción `CMD` o `ENTRYPOINT`** Estas instrucciones definen el comando o entrada predeterminados que se ejecutarán cuando se inicie un contenedor a partir de la imagen. Puedes tener solo una de estas instrucciones en tu Dockerfile.

```
CMD ["nginx", "-g", "daemon off;"]
```

Una vez que hayas creado un `Dockerfile`, puedes utilizar el comando `docker build` para construir una imagen a partir de él

```
docker build -t superappgalaxial:1.0 .
```

El `.` al final del comando indica que Docker debe buscar el Dockerfile en el directorio actual.

Una vez que la imagen se ha construido, puedes usarla para crear y ejecutar contenedores Docker. Los Dockerfiles son una parte fundamental de la infraestructura de contenedores y permiten definir el entorno de tu aplicación de una manera reproducible y controlada.

Ejemplo Dockerfile para nginx

```
FROM ubuntu:20.04

COPY mi_aplicacion /app

RUN apt-get update && apt-get install -y nginx

WORKDIR /app

EXPOSE 80

CMD ["nginx", "-g", "daemon off;"]
```

** **NGINX** es un servidor web de código abierto y un servidor de proxy inverso. Es muy utilizado por su alta velocidad, eficiencia y capacidad para manejar cargas de trabajo pesadas. Se puede utilizar junto a apache para manejar algunas tareas de servicio web y balancear la carga.*