Facultad de Ingeniería
Carrera Ingeniería Informática

Procedimiento para el uso de la tecnología de contenedores Docker en la División Territorial de ETECSA en Cienfuegos

Trabajo de diploma para optar por el título de Ingeniero Informático

Autora:
Beatriz Reyes Fundora

Tutores:
Msc. Denis Morejón López. DTCF
Msc. Viviana R. Toledo Rivero. Univ. Cfgos

Cienfuegos, Cuba
Curso 2021

Agradecimientos

*A **mi mamá** por haberme dado la vida..., por estar ahí siempre que lo he necesitado, por luchar junto a mí y pelear en cada dura batalla que la vida me ha puesto, por haberme apoyado en este largo camino de convertirme en una profesional...*

*A **mi abuela** por ser mi motor impulsor y mi ejemplo a seguir como mujer trabajadora, y por estar siempre a mi lado en los momentos más difíciles.*

*A **mi esposo y su familia** por apoyarme en esta dura batalla de formación, por ayudarme a levantarme en cada uno de mis tropiezos, y por aguantar todo mi mal genio.*

*A **mi padre, hermano, tío y abuelo** por estar siempre al pendiente de mí y de mi futuro ayudándome y protegiéndome.*

*A **mis amigas Laura, Annaliet, Diana y su mamá Maira** por siempre apoyarme y brindarme sus casas y su amistad.*

*A **mis tutores Viviana y Denis** por su dedicación y guía durante todos estos años para lograr la tan anhelada meta de ser ingeniera.*

*A **mis profesores y compañeros** por contribuir a lo largo de este tiempo en mi formación*

*A **TODOS** los que de una forma u otra pusieron su granito de arena en este proceso tan importante de mi vida.*

Dedicatoria

A toda mi Familia

Resumen

Las tecnologías de contenedores se están imponiendo en la actualidad en todo el mundo para mejorar la eficiencia en el uso de los recursos de hardware dedicados al hospedaje de servidores virtuales; por lo cual con este proyecto se creó un procedimiento para desplegar la tecnología de contenedores Docker junto al sistema de virtualización utilizado en la División Territorial de ETECSA en Cienfuegos, facilitando así el trabajo de programadores y administradores de red con esta tecnología. Actualmente se utiliza una virtualización total o de máquinas virtuales y se pretende el uso de los contenedores Docker, lo cual permite a diferencia de las máquinas virtuales y los contenedores Linux que se puedan desplegar más servidores por cada servidor físico del que se disponga; así ahorrando más energía, aspecto vital para la situación económica de nuestro país. También permite utilizar menos recursos de hardware, lo cual brinda rapidez en la gestión de procesos y agilidad en el trabajo; y permite el despliegue de múltiples versiones de una misma aplicación, propiciando a los desarrolladores la implementación, prueba y despliegue de sus aplicaciones o proyectos. El uso de tecnologías de contenedores, como DOCKER resulta interesante y novedosa para desplegar nuevos servicios y facilitar el trabajo de programadores y administradores de red en la DT ETECSA en Cienfuegos.

Palabras clave: contenedores, servidores, máquinas virtuales, procedimiento, DOCKER.

Summary

Container technologies are now being imposed around the world to improve efficiency in the use of hardware resources dedicated to hosting virtual servers; Therefore, with this project, a procedure was created to deploy the Docker container technology together with the virtualization system used in the Territorial Division of ETECSA in Cienfuegos, thus facilitating the work of programmers and network administrators with this technology. Currently a total virtualization or virtual machine virtualization is used and the use of Docker containers is intended, which, unlike virtual machines and Linux containers, allows more servers to be deployed for each physical server available; thus saving more energy, a vital aspect for the economic situation of our country. It also allows you to use fewer hardware resources, which provides speed in process management and agility at work; and allows the deployment of multiple versions of the same application, enabling developers to implement, test and deploy their applications or projects. The use of container technologies, such as DOCKER, is interesting and novel to deploy new services and facilitate the work of programmers and network administrators at DT ETECSA in Cienfuegos.

Keywords: containers, servers, virtual machines, procedure, DOCKER.

Índice

Índice de tablas

Índice de figuras

Introducción

En la actualidad casi no se puede tener una discusión respecto a Cloud Computing sin llegar al concepto de “Contenedores” (Containers). Organizaciones de todos los segmentos de negocio hoy quieren entender que son los contenedores, que significan para las aplicaciones en la nube y como pueden usarlos.[1]

Cada vez se desarrollan menos aplicaciones monolíticas y más basadas en módulos o en micro servicios, que permiten un desarrollo más ágil, rápido y a la vez portable. Las tendencias tecnológicas siguen evolucionando y cada día van apareciendo nuevos conceptos que debemos aprender.[2]

Antes de ahondar en el concepto de contenedores, volvamos unos años atrás para recordar el nacimiento de la virtualización. A medida que el hardware se hacía más poderoso, nos encontramos con que el software no ocupaba todas las capacidades de la máquina física donde se encontraba siendo ejecutada. Dado lo anterior se crearon recursos virtuales para simular el hardware base sobre el cual se ejecuta el software, permitiendo que múltiples aplicaciones puedan ser ejecutadas al mismo tiempo, cada una usando una fracción de los recursos del hardware físico disponible. A esta simulación que permite compartir recursos la denominamos comúnmente virtualización.

Al escuchar la palabra virtualización pensamos inmediatamente en máquinas virtuales, pero es importante entender que este es solo un tipo de virtualización.

Dentro de los tipos de virtualización existentes, estos se pueden diferenciar en dos grandes ramas: la virtualización de plataforma la cual consiste en la creación de una máquina virtual utilizando una combinación de hardware y software, y la virtualización de recursos que es la que involucra la simulación de recursos, como volúmenes de almacenamiento, espacios de nombres y recursos de red.[2]

En la División Territorial de ETECSA en Cienfuegos (DTCF) se utiliza la plataforma de virtualización Proxmox VE (Proxmox Virtual Enviroment).

Proxmox VE es una solución de código abierto para virtualización de servidores con Licencia GPL v2 y de gratis distribución. El producto pertenece a una empresa austriaca llamada: “Proxmox Company “que en el 2009 obtuvo el premio “Constantinus Award 2009 Open Source “por la calidad de su trabajo con Proxmox VE. Su producto insignia es en cambio el “Proxmox Mail Gateway”, por el que hay que pagar y que es una máquina virtual con toda una solución de servicio de correo antivirus, antispam, y reportería integrada.[3]

Tras un intenso estudio realizado hace unos trece años, por internet y por listas de discusión de administradores de sistemas, para encontrar la tecnología de virtualización conveniente, dado que el país y en concreto la empresa se encontraba en tránsito hacia el uso de la tecnología de código abierto, expresado en la resolución 27 de 2008 del presidente ejecutivo de la empresa, dio lugar a utilizar, este producto Proxmox VE (Proxmox Virtual Enviroment) el cual resultó ser el elegido para su despliegue en la DTCF, y es el mismo sistema que se utiliza hasta la actualidad; por lo cual la técnica de virtualización utilizada en la empresa es, la virtualización total o máquinas virtuales que es sobre la que se basa este producto, y también los contenedores Linux.

Las tecnologías de contenedores suelen a diferencia de la tecnología de máquinas virtuales utilizar menos recursos de hardware: ahorra almacenamiento en disco (si desea utilizar una máquina virtual completa, deberá tener el almacenamiento de la máquina virtual multiplicado por la cantidad de máquinas virtuales que desee), con Docker se puede compartir el almacenamiento entre todos los contenedores y si tiene 1000 contenedores, es posible que solo tenga un poco más de 1 GB de espacio para el sistema operativo de contenedores. Los contenedores son más livianos que un sistema virtualizado completo ya que no tienen que levantar un sistema operativo independiente y demoran solo unos pocos segundos o menos en ejecutarse.[5]

En ETECSA actualmente el esquema de virtualización no permite a los programadores trabajar simultáneamente con varias versiones de la misma aplicación para facilitar la implementación, prueba y despliegue de sus aplicaciones, tampoco el despliegue de

varios servidores por cada servidor físico, ni el despliegue de varios sistemas operativos, lo que provoca la ineficiente gestión de procesos y servicios informáticos de la empresa, así como el derroche de recursos de hardware. Por todo lo anterior, se define como:

Problema: La ausencia de un mecanismo de virtualización que permita el uso eficiente de la explotación de servicios informáticos sobre recursos de hardware.

Objeto de estudio

La tecnología de contenedores de software.

Campo de acción

La tecnología de contenedores de software en los servicios informáticos internos de la DTCF.

Para darle solución al mismo se plantea el siguiente **Objetivo General:** Determinar el estado del arte sobre el uso de la tecnología de contenedores Docker como alternativa para el despliegue y migración de los servicios informáticos internos de la División Territorial de ETECSA en Cienfuegos a dicha tecnología.

Del cual se derivan los siguientes **Objetivos Específicos:**

- Analizar las características de la virtualización y el esquema de virtualización utilizado en ETECSA.

- Analizar las características y arquitectura de la tecnología de contenedores, en particular Docker.

- Determinar un procedimiento para el uso de la tecnología de contenedores Docker en la DTCF.

El presente trabajo ha sido estructurado en 3 capítulos: capítulo 1: donde se describen las características de la virtualización y las ramas en las que se divide la misma, con sus respectivos tipos; la tecnología de contenedores con una descripción de algunos de sus tipos y el esquema de virtualización utilizado en la División Territorial de ETECSA en Cienfuegos; capitulo 2, en el que se describen las características, ventajas, arquitectura y componentes de la tecnología de contenedores Docker, se realiza una comparación entre contenedores Docker y máquinas virtuales, y se detalla el procedimiento para el uso de la tecnología de contenedores Docker en la DTCF y el capítulo 3: donde se realizan una serie de pruebas al procedimiento para el uso de la tecnología de contenedores Docker.

1 – Características de la virtualización, tecnología de contenedores y el esquema de virtualización utilizado en la DTCF.

1.1 – Introducción

En este capítulo se aborda lo referente a las características de la virtualización y se enfatiza en las ramas de la virtualización y sus tipos. Se presenta lo referido a la tecnología de contenedores y se describe la plataforma y el tipo de virtualización utilizado en la División territorial de ETECSA en Cienfuegos en la actualidad.

1.2 – Virtualización

La virtualización en informática es una técnica de abstracción para romper la relación directa entre distintos componentes computacionales sin que como resultado se afecte el servicio que reciben los usuarios finales. Por ejemplo, estamos en presencia de virtualización cuando una aplicación puede ejecutarse en cualquier sistema operativo o hardware, o cuando un sistema operativo puede ejecutarse en diferentes servidores con independencia del cambio de hardware (ver Anexo 1), o incluso cuando se define en un sistema operativo un medio de almacenamiento aparentemente local y en realidad está ubicado en otra máquina. [4]

Una Máquina Virtual es un contenedor de software perfectamente aislado que puede ejecutar sus propios sistemas operativos y aplicaciones como si fuera un ordenador físico (ver Anexo 2).

Una Máquina Física es una máquina que contiene piezas hardware (físicas) que ofrecen una operatividad a muy bajo nivel (nivel físico) en una Arquitectura conocida y donde se obtienen los mejores y más eficaces resultados en cuanto a velocidad final de la máquina (ver Anexo 2).[6]

Entre los conceptos básicos de la virtualización encontramos anfitrión, invitado e hipervisor:

Anfitrión (host)

Es el Sistema Operativo (SO) que ejecuta el software de virtualización. Es el encargado de controlar el hardware real.

Invitado o huésped (guest)

Es el Sistema Operativo virtualizado. Puede haber varios SO invitados en un mismo anfitrión. Los invitados no deben interferir ni entre ellos ni con el anfitrión.

Hipervisor

Hipervisor o Virtual Machine Manager (VMM) se le denomina al software de virtualización. Este se ejecuta como parte del sistema operativo anfitrión o es el anfitrión. Permiten que diferentes sistemas operativos, tareas y configuraciones de software coexistan en una misma máquina física. Abstraen los recursos físicos de la máquina anfitriona para las distintas máquinas virtuales. Garantizan un nivel de aislamiento entre los invitados. Proporcionan una interfaz única para el hardware.

1.3 – Ramas de la virtualización

La virtualización se divide en dos grandes ramas: Virtualización de plataforma y Virtualización de recursos.

1.3.1 – Virtualización de plataforma

Consiste en la creación de una máquina virtual utilizando una combinación de hardware y software. Se lleva a cabo a través de un software de virtualización. Dicho software actúa de host o anfitrión y simula un determinado entorno computacional (máquina virtual). En esta máquina virtual se instala un software guest o invitado, normalmente un sistema operativo completo. Instalado de la misma manera que si lo estuviera en una máquina real. La simulación debe ser suficientemente robusta como para soportar todas las interfaces externas del software invitado, incluidos, en algunos casos, drivers de hardware.[2]

En la virtualización de plataforma se encuentran disimiles tipos de virtualización. Entre ellos se encuentran:

- **Emulación o simulación.**

La máquina virtual simula un hardware completo. Admite sistema operativo (SO) invitados sin modificar para arquitecturas CPU completamente diferentes a la CPU del SO anfitrión.

Un emulador permite ejecutar programas en una plataforma diferente para la que fueron escritos (ver Anexo 3).

Ejemplos:

Bochs, PearPC (emulador PowerPC para x86), QEMU sin aceleración, MAME (emulador de hardware de máquinas recreativas).

Ventajas:

Simula hardware que no está físicamente disponible.

Desventajas:

Bajo rendimiento, alto coste de computación.

• **Virtualización nativa o completa.**

La máquina virtual simula un hardware suficiente para poder permitir a un sistema operativo invitado sin modificar, correr de forma aislada sobre el mismo tipo de CPU que la máquina anfitriona. En virtualización nativa, tanto el sistema anfitrión como el sistema operativo invitado se ejecutan sobre la misma CPU. Se consigue un alto rendimiento, ya que no es necesario emular todo el entorno (ver Anexo 4).

Ejemplos:

Parallels Workstation, Parallels Desktop for Mac, VirtualBox, Microsoft Hyper-V, VMware Workstation, VMwareServer (formerly GSX Server), KVM+QEMU, Parallels Desktop, QEMU, Microsoft Virtual PC, Microsoft Virtual Server, Win4LinPro, Xen + Intel VT-x.

[2]

Ventajas:

Flexibilidad y un alto rendimiento.

Desventajas:

No se pueden emular otras arquitecturas.

• **Virtualización asistida por hardware.**

Es un caso especial de la virtualización completa en la que se cuenta con ayuda del procesador. Intel con su tecnología VT-x y AMD con AMD-V proporcionan ayuda por hardware al software de virtualización.

Como **ejemplos** de plataformas de virtualización adaptadas a este hardware tenemos:

KVM, VMware Workstation, VMware Fusion, Microsoft Hyper-V, Microsoft Virtual PC, Xen, Parallels Desktop for Mac, VirtualBox y Parallels Workstation.

Incluidas en 2005 y 2006 por Intel y AMD. Añaden soporte hardware para la virtualización.

• Intel Virtualization Technology (Intel VT-x), codename Vanderpool.

• AMD Virtualization (AMD-V), codename Pacifica.

Permiten a los hipervisores un rendimiento mayor en modo virtualización completa. De esta forma la virtualización completa es mucho más fácil de implementar y ofrece un mayor rendimiento. Aunque el procesador la incluya, hay que activarla en BIOS. A estas extensiones x86 también se les denomina, de forma neutral en cuanto al fabricante, como HVM (Hardware Virtual Machine).

• **Paravirtualización.**

La máquina virtual no necesariamente simula un hardware, sino que ofrece un API especial que solo puede utilizarse en un sistema operativo invitado modificado. Las llamadas del sistema operativo invitado al hipervisor se denominan hypercalls (ver Anexo 5).

Ejemplos:

Xen en CPU estándar.

Ventajas:

Mayor rendimiento que la virtualización nativa, no se necesita de una CPU con soporte para virtualización.

Desventajas:

Hay que modificar el SO invitado.

• **Virtualización a nivel de sistema operativo.**

El SO anfitrión virtualiza el hardware a nivel de SO. Esto permite que varios SO virtuales se ejecuten de forma aislada en un mismo servidor físico. El SO invitado ejecuta el mismo el mismo kernel que el anfitrión, de hecho, son el mismo SO.

Básicamente se consigue la virtualización instanciando la imagen del SO (tal como lo ven las aplicaciones), no existe un hipervisor (ver Anexo 6).

Ejemplos:

FreeBSD jails, Solaris Containers, OpenVZ, Linux-VServer, LXC (Linux Containers), AIX Workload Partitions, Parallels Virtuozzo Containers, y iCore Virtual Accounts.

Ventajas:

Muy rápida, la capa de virtualización es muy ligera, rendimiento muy cercano al nativo.

Desventajas:

Muy difícil de implementar un aislamiento completo. No se pueden virtualizar diferentes SO.

1.3.2 – Virtualización de Recursos

Es el tipo de virtualización que involucra la simulación de recursos, como volúmenes de almacenamiento, espacios de nombres y recursos de red.[2]

Algunos ejemplos de virtualización de recursos son:

- **RAID y volume managers** combinan muchos discos en un gran disco lógico.
- **Virtualización de almacenamiento** refiere al proceso de abstraer el almacenamiento lógico del almacenamiento físico, y es comúnmente usado en SANs (Storage Area Network). Los recursos de almacenamientos físicos son agregados al storage pool, del cual es creado el almacenamiento lógico. Múltiples dispositivos de almacenamiento independientes, que pueden estar dispersos en la red, aparecen ante el usuario como un dispositivo de almacenamiento independiente del lugar físico, monolítico y que puede ser administrado centralmente.
- **Canales virtuales** aseguran que las redes sean libres de bloqueos mediante multiplexación del tiempo los enlaces físicos, obteniendo varios enlaces virtuales.

- **Channel bonding y el equipamiento de red** utilizan para trabajar múltiples enlaces combinados mientras ofrecen un enlace único y con mayor amplitud de banda.
- **Red privada virtual (VPN) y Traducción de dirección de red (NAT)** y tecnologías de red similares crean una red virtual dentro o a través de subredes.
- **Cluster, multiprocesadores, multi-cores y servidores** virtuales usan las tecnologías anteriormente mencionadas para combinar múltiples y diferentes computadoras en una gran meta computadora única, rápida e independiente.
- **Particionamiento** es la división de un solo recurso, como el espacio en disco o ancho de banda de la red, en un número más pequeño y con varios recursos del mismo tipo más fáciles de utilizar.
- **Encapsulación** es el ocultamiento de los recursos complejos mediante la creación de un interfaz simple. Por ejemplo, muchas veces los procesadores incorporan memoria caché o segmentación para mejorar el rendimiento, pero estos elementos no son reflejados en su interfaz virtual externa. Interfaces virtuales similares que ocultan implementaciones complejas se encuentran en los discos, módems, router y otros dispositivos inteligentes.

1.4- Tecnología de Contenedores

La virtualización ha revolucionado la informática. La distribución de los recursos de una máquina física en varias máquinas virtuales ha entrado en el panorama de las tecnologías de la información en forma de virtualización de hardware.

Una alternativa a la virtualización de hardware la constituye la virtualización del sistema operativo, por la cual diversas aplicaciones de un servidor se implementan en entornos virtuales aislados, los llamados contenedores, que funcionan en el mismo sistema operativo (virtualización basada en contenedores). [7]

Como las máquinas virtuales, cada una con su sistema operativo, los contenedores también brindan la posibilidad de usar en paralelo diversas aplicaciones, cada una con

sus requisitos, en un mismo sistema físico. Como los contenedores no disponen de ningún sistema operativo propio, esta técnica de virtualización se diferencia por una instalación mucho más sencilla y un menor consumo.

En el contexto de software, el término contenedor es “una referencia generalizada para cualquier partición virtual que no sea una máquina virtual basada en un hipervisor.

Los contenedores permiten empaquetar y aislar aplicaciones con todas las librerías y dependencias que necesitan para poder ejecutarse. [7]

Un contenedor de software, en su concepción más básica, puede considerarse como aplicación para el servidor. Para poder instalar una aplicación, el contenedor se carga en el ordenador en un formato portable o imagen que incluye todos los datos necesarios para su funcionamiento y, en el ordenador, se inicia en un entorno virtual. Prácticamente todos los sistemas operativos soportan la implementación de contenedores de aplicaciones: en Windows se utiliza el software Virtuozzo creado por Parallels, FreeBSD tiene el entorno de virtualización Jails y Linux soporta contenedores en la forma de OpenVZ y LXC (Linux Containers). [8]

Intel en su artículo Linux Containers Streamline Virtualization and Complement Hypervisor-Based Virtual Machines, comenta que “los enfoques alternativos para la partición de recursos que no se basen en hipervisores, fueron desarrollados por Unix y a medida que los sistemas operativos tipo Unix crecieron a finales de 1970 y principios de 1980 también fueron impulsados por diferentes organizaciones que incluían laboratorios Bell, AT&T, Digital Equipment Corporation, Sun Microsystems e instituciones académicas. Estas tecnologías desarrolladas se refirieron colectivamente como virtualización por sistema operativo, proporcionando enfoques livianos de virtualización, actuando como los fundamentos de los contenedores Linux de hoy en día”. [7]

En el Anexo 7 de este documento se representan la evolución de las técnicas de partición de recursos basados en hipervisores (ver Anexo 7) y a continuación podemos ver la descripción de algunas de estas técnicas de partición de recursos.

Características de algunas de las técnicas de partición de recursos basadas en hipervisores.

- Llamada al sistema chroot

Este es un fundamento crítico en la virtualización basada en contenedores. Tal como lo define Intel “el mecanismo chroot puede redefinir el directorio raíz para cualquier programa en ejecución, previniendo con eficacia que el programa sea capaz de nombrar o acceder recursos fuera del árbol del directorio raíz”. Esta característica fue introducida en la versión 7 de Unix por los laboratorios Bell en 1979 y como parte de 4.2BSD por la Universidad de California, Berkeley en 1983. [7]

- FreeBSD Jails

Es parecido en concepto que chroot, pero con un enfoque en la seguridad. Según Intel “las definiciones de FreeBSD Jails pueden restringir explícitamente el acceso fuera del entorno de espacio aislado por entidades tales como archivos, procesos, y cuentas de usuario. [7]

- Solaris Containers (Solaris Zone)

Este mecanismo se hizo basado en las capacidades de chroot y FreeBSD Jails. Según Intel las Zones, “son instancias de servidores individuales que pueden coexistir dentro de una sola instancia de sistema operativo”. Sun Microsystems introdujeron esta característica con el nombre de Solaris Containers como parte de Solaris 10 en el año 2005, luego Oracle oficialmente cambio el nombre a Solaris Zones con el lanzamiento de Solaris 11 en el año 2011. [7]

- Linux Containers (LXC)

En Linux, Contenedores Linux (LXC), el bloque de construcción que formó la base de las tecnologías de contenerización fue agregada al kernel en el 2008. LXC combinó el uso del kernel cgroups (que permite aislar y rastrear el uso de recursos) y namespaces (que permite a los grupos separarse, por lo que no pueden verse "unos a otros") para implementar aislamiento de procesos ligeros.

Los Contenedores Linux son capacidades de nivel de sistema operativo que hacen posible ejecutar múltiples contenedores Linux aislados, en un host de control (el host

LXC). Los contenedores de Linux sirven como una alternativa ligera a las máquinas virtuales, ya que no requieren los hipervisores.

Los contenedores de Linux son un conjunto de tecnologías que juntas forman un contenedor (de Docker), estos conjuntos de tecnologías se llaman, Namespaces que permite a la aplicación que corre en un contenedor de Docker tener una vista de los recursos del sistema operativo, Cgroups que permite limitar y medir los recursos que se encuentran disponibles en el sistema operativo y Chroot que permite tener en el contenedor una vista de un sistema "falso" para el mismo, es decir, crea su propio entorno de ejecución con su propio root y home. Docker, que se basa en la tecnología LXC, ha proporcionado a los desarrolladores una plataforma para ejecutar sus aplicaciones y, al mismo tiempo, ha habilitado a las personas de operaciones con una herramienta que les permite implementar el mismo contenedor en servidores de producción o centros de datos. Docker y LXC están destinados más a la zona de pruebas, la contenedorización y el aislamiento de recursos.

- Docker

James Turnbull en su libro The Docker Book, describe a Docker como "un motor de código abierto que automatiza el despliegue de aplicaciones en contenedores". Docker fue desarrollado por la empresa del mismo nombre Docker, Inc., anteriormente conocida como dotCloud Inc. Docker utiliza una capa de abstracción para poder acceder a las capacidades de virtualización (cgroups, namespaces, capabilities, etc.) del kernel de Linux. [7]

1.5 – Tipo de virtualización utilizada en la División Territorial de ETECSA en Cienfuegos

Cuando el país y en concreto la empresa estuvo en tránsito hacia el uso de tecnologías de código abierto, expresado en la resolución 27 de 2008 en la Empresa de Telecomunicaciones de Cuba (ETECSA) ya se usaba la técnica de virtualización, pero aún en un porciento muy bajo, y empleando casi siempre soluciones informáticas privativas, como variantes de la familia VMware. Por otra parte, el uso de servidores instalados a la manera tradicional (sin virtualización) dificultaba y dilataba la capacidad

de respuestas de los administradores de sistemas ante la ocurrencia de averías de hardware o software en los mismos, a la vez que encarecían el inventario informático al tener que dedicar servidores reales a aplicaciones específicas cuando no se podían mezclar en una misma máquina.

Para darle solución a este problema y lograr el uso de la técnica de virtualización más adecuada en un porciento más elevado, en ese momento en la empresa se realizó un estudio donde se midieron diversos criterios de selección de una herramienta de virtualización. Tras una búsqueda en base a requisitos tales como: código abierto, fácil de instalar y usar, una interfaz administrativa vía web y otras; se implementa un sistema basado en código abierto que permite crear máquinas virtuales y gestionarlas de manera que den solución al problema de la baja disponibilidad de los servicios vitales en una organización y al uso irracional de recursos de hardware y para lograr la disminución notable del tiempo de restablecimiento de un servicio virtualizado. La búsqueda de la herramienta de virtualización, da como resultado el uso de este producto nombrado Proxmox VE (Proxmox Virtual Enviroment) el cual resulta ser conveniente.

Proxmox VE es una solución de código abierto para virtualización de servidores con Licencia GPL v2 (El free software clásico) y de gratis distribución. El producto pertenece a una empresa austriaca llamada: “Proxmox Company “que en el 2009 obtuvo el premio “Constantinus Award 2009 Open Source “por la calidad de su trabajo con Proxmox VE. Su producto insignia es en cambio el “Proxmox Mail Gateway”, por el que hay que pagar y que es algo así como una máquina virtual con toda una solución de servicio de correo antivirus, antispam, y reportería integrada.

Proxmox VE no es una nueva forma de virtualizar, sino un producto que integra herramientas ya maduras como: QEMU, KVM, Open VZ, RSYNC.

Proxmox VE es la herramienta actualmente utilizada en la empresa, utilizando como técnica de virtualización la virtualización total o de máquinas virtuales ya que puede realizar tanto “virtualización total” como “paravirtualización”. En tal sentido maneja máquinas totalmente virtualizadas, a las que se suele llamar simplemente máquinas virtuales o contenedores virtuales (respondiendo a la técnica de Paravirtualización) (ver Anexo 8).

Con esta solución se crean, en una red de servidores físicos, distintas máquinas virtuales independientes con sistemas operativos diversos (Linux y Windows), donde corren los servicios de vital importancia para la organización gracias a la elevada fiabilidad y capacidad de restauración del sistema. Se implementan las posibilidades que incluye el producto de realizar operaciones sobre las máquinas virtuales como la planificación de salvas hacia servidores remotos, el movimiento en caliente de máquinas virtuales entre servidores físicos, el uso de redes locales virtuales, y el empleo de formas más eficientes de virtualización como la que se implementa a nivel de sistema operativo con contenedores virtuales.[3]

1.6 – Conclusiones

En este capítulo se define el concepto de virtualización, sus ramas y sus tipos; y se realiza un análisis de la virtualización total o de máquinas virtuales y la tecnología de contenedores que son los utilizado por la DTCF determinando que resulta un mecanismo ineficiente y poco ágil, ya que consume más espacio de almacenamiento y no es tan rápido en el proceso de ejecución puesto que tiene que levantar un sistema operativo completo con bibliotecas, dependencias, etc.

2 – Características, arquitectura y componentes de Docker y procedimiento para el uso de la tecnología de contenedores Docker

2.1 – Introducción

En este capítulo se aborda lo referente a las características, arquitectura, ventajas y componentes de Docker. También se realiza una comparación entre contenedores Docker y máquinas virtuales. Se presenta y crea el procedimiento para el uso de la tecnología de contenedores Docker en la División Territorial de ETECSA en Cienfuegos que va desde su instalación hasta el uso de sus principales comandos y componentes.

2.2 – Docker

Docker es un proyecto de código abierto que automatiza la implementación de aplicaciones dentro de los contenedores de software.[9]

Este contenedor empaqueta todo lo necesario para que uno o más procesos (servicios o aplicaciones) funcionen: código, herramientas del sistema, bibliotecas del sistema, dependencias, etc. Esto garantiza que siempre se podrá ejecutar, independientemente del entorno en el que se quiera desplegar. No hay que preocuparse de qué software ni versiones tiene la máquina, ya que la aplicación se ejecutará en el contenedor.

Estos contenedores de aplicaciones son similares a las máquinas virtuales ligeras, ya que se pueden ejecutar de forma aislada entre sí y con el host en ejecución. [9]

Docker es una plataforma abierta para los desarrolladores y administradores de sistemas para la construcción, envío y ejecución de aplicaciones. Consta de Motor de Docker, una Herramienta ligera y portátil y Docker Hub que es un servicio en la nube para compartir aplicaciones que permite la automatización de flujos de trabajo. Docker permite ensamblar aplicaciones rápidamente y elimina la fricción entre los entornos de desarrollo, control de calidad y producción. Como resultado, se puede enviar más rápido y ejecutar la misma aplicación, sin cambios, en los ordenadores portátiles, los centros de datos de máquinas virtuales, y cualquier nube. [10]

Salomon Hykes comenzó Docker como un proyecto interno dentro de dotCloud, empresa enfocada a PaaS (plataforma como servicio). Fue liberado como código abierto en marzo de 2013.

Con el lanzamiento de la versión 0.9 (en marzo de 2014) Docker dejó de utilizar LXC como entorno de ejecución por defecto y lo reemplazó con su propia librería, libcontainer (escrita en Go), que se encarga de hablar directamente con el kernel. Actualmente es uno de los proyectos con más estrellas en GitHub, con miles de bifurcaciones (forks) y miles de colaboradores.[7]

Con la tecnología de Docker se puede virtualizar un Linux con todas las aplicaciones que se necesiten dentro de nuestro sistema operativo Linux, para "empaquetarlo y desplegarlo" en cualquier otro Linux sin necesidad más que de introducir un par de comandos.

Docker implementa un alto nivel API para proporcionar contenedores ligeros que ejecuten los procesos de manera aislada. Basándose en la cima de las instalaciones que ofrece el kernel Linux un contenedor Docker, a diferencia de como una máquina virtual tradicional lo hace; no exige o incluye un sistema operativo independiente. En su lugar, se basa en la funcionalidad del núcleo y utiliza aislamiento de recursos (CPU, memoria, bloque de E / S, red, etc.) y los espacios de nombres separados para aislar completamente la vista de la aplicación de la del sistema operativo. Docker accede a la virtualización que el kernel Linux ofrece, ya sea directamente o a través de la biblioteca “libcontainer”. [10]

2.2.1 – Características de Docker

Las principales características de Docker son:

- Autogestión de los contenedores.
- Fiabilidad
- Aplicaciones libres de las dependencias instaladas en el sistema anfitrión.
- Capacidad para desplegar varios contenedores.
- Contenedores muy livianos que facilitan su almacenaje, transporte y despliegue.
- Capacidad para desplegar una amplia gama de aplicaciones.

- Compatibilidad Multi-Sistema es decir que podremos desplegar nuestros contenedores en múltiples plataformas.
- Podremos compartir nuestros contenedores a través del repositorio Docker Hub.[11]

2.2.2 – Ventajas de la virtualización con Docker

Docker es una herramienta de virtualización que a diferencia de las máquinas virtuales suele utilizarse para levantar máquinas independientes con sistemas operativos ligeros y en muy poco tiempo.

Los entornos de ejecución de Docker son mucho más ligeros, y se aprovecha mucho mejor el hardware, además de permitir levantar muchos más contenedores que máquinas virtuales en la misma máquina física. Mientras que una máquina virtual puede tardar un minuto o más en arrancar y tener disponible nuestra aplicación, un contenedor Docker se levanta y responde en unos pocos segundos (o menos, según la imagen). El espacio ocupado en disco es muy inferior con Docker al no necesitar que instalemos el sistema operativo completo.[12]

Docker y sus contenedores son capaces de compartir un solo núcleo y bibliotecas de aplicaciones, esto ayuda a que presenten una carga más baja de sistema que las máquinas virtuales.

Docker es una herramienta de virtualización que a diferencia de las máquinas virtuales permite a desarrolladores y administradores de sistemas probar aplicaciones o servicios en un entorno seguro e igual al de producción, reduciendo los tiempos de pruebas y adaptaciones entre los entornos de prueba y producción.

Las principales ventajas de usar contenedores Docker son:

- Las instancias se inician en pocos segundos.
- Son fácilmente replicables.
- Es fácil de automatizar y de integrar en entornos de integración continua.
- Consumen menos recursos que las máquinas virtuales tradicionales.

- Mayor rendimiento que la virtualización tradicional ya que corre directamente sobre el Kernel de la máquina en la que se aloja, evitando al hipervisor.
- Ocupan mucho menos espacio.
- Permite aislar las dependencias de una aplicación de las instaladas en el host.
- Existe un gran repositorio de imágenes ya creadas sobre miles de aplicaciones, que además pueden modificarse libremente.

Por todo esto Docker ha entrado con mucha fuerza en el mundo del desarrollo, ya que permite desplegar las aplicaciones en el mismo entorno que tienen en producción o viceversa, permite desarrollarlas en el mismo entorno que tendrán en producción.

2.2.3 – Arquitectura de la tecnología de contenedores Docker

Docker usa una arquitectura cliente-servidor. El cliente de Docker habla con el demonio de Docker que hace el trabajo de crear, correr y distribuir los contenedores. Ambos pueden ejecutarse en el mismo sistema, o se puede conectar un cliente a un demonio Docker remoto. El cliente Docker y el demonio se comunican vía sockets o a través de una RESTfull API (ver Anexo 9).[12]

El cliente de Docker (Docker Client): es la principal interfaz de usuario para Docker. Él acepta comandos del usuario y se comunica con el demonio Docker.

El demonio Docker (Docker Engine): corre en una máquina anfitriona (host). El usuario no interactúa directamente con el demonio, en su lugar lo hace a través del cliente Docker. El demonio Docker levanta los contenedores haciendo uso de las imágenes, que pueden estar en local o en el Docker Registry.
Cada contenedor se crea a partir de una imagen y es un entorno aislado y seguro dónde se ejecuta nuestra aplicación.

2.2.4 – Componentes de Docker

Los componentes de Docker se dividen en dos grupos: componentes base y componentes del flujo de trabajo. A continuación, se explican cada uno de ellos.

Componentes base

A continuación, se describen los componentes núcleo de Docker, los cuales son el servidor y el cliente, lo cuales en conjunto se les conoce como Docker Engine.

Docker Daemon

También se le conoce como Servidor de Docker. La documentación oficial lo define como “el proceso persistente que crea y gestiona los objetos Docker, tales como imágenes, contenedores, redes y volúmenes de datos”.El servidor Docker está integrado en el mismo binario que se usa para ejecutar los procesos del cliente, por ello tiende a confundirse. El Docker daemon sin embargo necesita privilegios root para poder ejecutarse, mientras que el cliente no.

Docker Client

Cualquier software o herramienta que hace uso de la API del demonio Docker, pero suele ser el comando docker, que es la herramienta de línea de comandos para gestionar Docker Engine. Éste cliente puede configurarse para hablar con un Docker local o remoto, lo que permite administrar nuestro entorno de desarrollo local como nuestros servidores de producción.

Docker Engine

Es el demonio que se ejecuta dentro del sistema operativo (Linux) y que expone una API para la gestión de imágenes, contenedores, volúmenes o redes. Sus funciones principales son:

- La creación de imágenes Docker.
- Publicación de imágenes en Docker Registry.
- Descarga de imágenes desde Docker Registry.
- Ejecución de contenedores usando las imágenes.
- Gestión de contenedores en ejecución (pararlo, arrancarlo, ver logs, ver estadísticas).

Componentes de flujo de trabajo

Para entender el funcionamiento interno de Docker es necesario conocer sobre las imágenes, contenedores y registros Docker (ver Anexo 10).

Docker Images

Son plantillas de sólo lectura que contienen el sistema operativo base (más adelante entraremos en profundidad) dónde correrá nuestra aplicación, además de las dependencias y software adicional instalado, necesario para que la aplicación funcione correctamente. Las plantillas son usadas por Docker Engine para crear los contenedores Docker.

Docker Containers

El contenedor de Docker aloja todo lo necesario para ejecutar un servicio o aplicación. Cada contenedor es creado de una imagen base y es una plataforma aislada. Un contenedor es simplemente un proceso para el sistema operativo, que se aprovecha de él para ejecutar una aplicación. Dicha aplicación sólo tiene visibilidad sobre el sistema de ficheros virtual del contenedor.

Docker Registres

Los registros de Docker guardan las imágenes. Pueden ser repositorios públicos o privados. El registro público lo provee el Hub de Docker, que sirve tantas imágenes oficiales cómo las subidas por usuarios con sus propias aplicaciones y configuraciones.

Así tenemos disponibles para todos los usuarios imágenes oficiales de las principales aplicaciones (MySQL, MongoDB, Apache, Tomcat, etc.), así como no oficiales de infinidad de aplicaciones y configuraciones. Docker Hub ha supuesto una gran manera de distribuir las aplicaciones. Es un proyecto open Source que puede ser instalado en cualquier servidor.

Docker File

Un Dockerfile es simplemente un fichero de texto que nos permite definir las instrucciones a seguir por Docker para construir una imagen. Es un archivo de configuración que se utiliza para crear imágenes. En dicho

archivo indicamos qué es lo que queremos que tenga la imagen, y los distintos comandos para instalar las herramientas.

Comandos de gestión

Docker Network

Hay varias formas en las que los contenedores se pueden conectar entre ellos y con el host. Las principales redes son las siguientes:

None

None es sencillamente que el contenedor no tiene interfaz de red, aunque sí que tiene interfaz de loopback. Este modo se utiliza para contenedores que no utilizan la red, como pueden ser contenedores para pruebas que no necesiten comunicación externa o contenedores que pueden dejarse preparados para ser conectados a una red posteriormente.

Bridge

Un bridge de linux proporciona una red interna en el host a través de la cual los contenedores se pueden comunicar. Las direcciones IP asignadas en esta red no son accesibles desde fuera del host. La red “bridge” aprovecha las iptables para hacer NAT y mapeado de puertos. La red bridge es la red por defecto de Docker. La creación del contenedor incluye los siguientes pasos respecto a la red:

1. Se proporciona al host una red bridge.
2. Un namespace para cada contenedor es proporcionado en ese bridge.
3. Las interfaces de red de los contenedores se mapean a las interfaces privadas del bridge.
4. Las iptables con NAT se usan para mapear entre cada contenedor y la interfaz pública del host.

Host

En este caso el contenedor comparte su namespaces de red con el host, lo que se traduce en un mejor rendimiento ya que elimina la necesidad de NAT, pero esto también provoca conflictos de puertos. Por lo tanto, el contenedor tiene acceso a todas las interfaces de red del host. La red “host” es la que se utiliza por defecto en “Mesos”.

Overlay

Utiliza los túneles de red para la comunicación de contenedores en diferentes hosts. Esto permite que dos contenedores que estén en hosts diferentes se comporten como si se encontraran en el mismo host, ya que hacen túneles de subredes de un host a otro. La tecnología de tunneling que utiliza Docker es VXLAN (Virtual Extensible Local Area Network), que se incluye de forma nativa desde el lanzamiento de la versión 1.9. Las redes multi-host requieren de parámetros adicionales cuando lanzamos el demonio de Docker, así como un registro key-value. Este tipo de red es la utilizada en la orquestación de contenedores distribuidos en diferentes hosts, ya que no se podrían comunicar entre sí por la red “bridge” comentada anteriormente.

Docker Swarm

Docker Swarm nos permite gestionar un grupo de hosts de Docker como un solo host de Docker virtual. Swarm utiliza la API estándar de Docker, por lo que cualquier herramienta que se comunique con el Docker Daemon puede utilizar Docker Swarm, que permite la escalabilidad a varios hosts.

Para desplegar una imagen de una aplicación cuando Docker Engine está en modo Swarm hay que crear un servicio. Normalmente el servicio será la imagen de un microservicio dentro de una aplicación más grande. Por ejemplo, un servicio puede ser un servidor HTTP, una base de datos, o cualquier otro tipo de programa ejecutable que se desee correr en un entorno distribuido.

Cuando se crea un servicio hay que especificar qué imagen de contendor usar, así como que comandos se ejecutarán en esos contenedores. También se definen las diferentes opciones de configuración del servicio como pueden ser:

- El puerto por el que Docker Swarm hará el servicio accesible desde el exterior.
- La red Overlay que permitirá conectar al servicio con otros servicios del clúster.
- Límites y reservas de memoria y CPU.
- Políticas de actualizaciones.
- Número de réplicas de la imagen que corren en el clúster.

En el momento de despliegue del servicio en Swarm, el gestor de Swarm acepta la definición como el estado deseado del servicio y distribuye el servicio en los nodos del clúster (dependiendo del número de réplicas). Estos procesos se ejecutan independientemente en cada uno de los nodos del clúster y se llaman task.

Docker Compose

Es un proyecto open Source que permite definir aplicaciones multi-contenedor de una manera sencilla. Es una alternativa más cómoda al uso del comando docker run, para trabajar con aplicaciones con varios componentes. Es una buena herramienta para gestionar entornos de desarrollo y de pruebas o para procesos de integración continua.

Docker Machine

Es un proyecto open Source para automatizar la creación de máquinas virtuales con Docker instalado, en entornos Mac, Windows o Linux, pudiendo administrar así un gran número de máquinas Docker. Incluye drivers para Virtualbox, que es la opción aconsejada para instalaciones de Docker en local, en vez de instalar Docker directamente en el host.

Esto simplifica y facilita la creación o la eliminación de una instalación de Docker, facilita la actualización de la versión de Docker o trabajar con distintas instalaciones a la vez. Usando el comando docker-machine podemos iniciar, inspeccionar, parar y reiniciar un host administrado, actualizar el Docker Client y el Docker Damon, y configurar un cliente para que hable con el host anfitrión. A través de la consola de administración podemos administrar y correr comandos Docker directamente desde el host. Éste comando docker-machine automáticamente crea hosts, instala Docker Engine en ellos y configura los clientes Docker.

Herramientas para el trabajo con Docker

Kubernetes

Kubernetes es una herramienta de código abierto desarrollada por Google para gestionar aplicaciones en el entorno de un clúster. Permite tratar grupos de contenedores de Docker como unidades con su dirección IP propia y escalarlos a tu

antojo. Puede programar y ejecutar contenedores de aplicaciones en grupos de máquinas físicas o virtuales. Sin embargo, Kubernetes también permite a los desarrolladores "cortar el cable" a máquinas físicas y virtuales, pasando de una infraestructura centrada en el host a una infraestructura centrada en el contenedor, que proporciona todas las ventajas y beneficios inherentes a los contenedores. Kubernetes proporciona la infraestructura para construir un entorno de desarrollo verdaderamente centrado en contenedores.

Kubernetes satisface una serie de necesidades comunes de aplicaciones que se ejecutan en producción, como:

- Ubicar conjuntamente los procesos de ayuda, facilitar las aplicaciones compuestas y preservar el modelo de una aplicación por contenedor,
- Montaje de sistemas de almacenamiento,
- Distribuyendo secretos,
- Control de salud de la aplicación,
- Replicando instancias de aplicaciones,
- Autoescalamiento horizontal,
- Nombramiento y descubrimiento,
- Balanceo de carga,
- Actualizaciones continuas,
- Monitoreo de recursos,
- Registro de acceso e ingestión,
- Soporte para introspección y depuración, y
- Identidad y autorización.

Esto proporciona la simplicidad de la Plataforma como un Servicio (PaaS) con la flexibilidad de la Infraestructura como un Servicio (IaaS), y facilita la portabilidad entre los proveedores de infraestructura.[13]

Kubernetes tiene terminología básica que deberemos conocer, antes que nada:

Componentes en nodo master

Primero veamos los componentes que se encuentran dentro del nodo master en el diagrama

- **etcd:** Es un storage que se encarga de guardar datos de forma persistente, liviana y confiable; Distribuida en forma clave-valor y desarrollada por CoreOS. Contiene la configuración/información de los nodos. Además representa el estado de éstos en punto de tiempo determinado y es accesible solamente por la Api Server de Kubernetes, debido a que contiene o puede contener información sensible.
- **Api Server:** es quien sirve a la API de Kubernetes, enviando los datos en formato JSON a través de HTTP. Es quien valida y recibe todas las peticiones y actualiza los objetos que se encuentran en etcd.
- **Scheduler:** Es quien se responsabiliza de distribuir la carga. Él se encarga de verificar los recursos utilizado por los nodos y ajustar el uso con los recursos que tenga disponible, es por ello que es uno de los componentes más importantes.
- **Controller Manager:** se puede considerar como un proceso o demonio que se encuentra corriendo continuamente y que además se encarga de recolectar la información y enviarla a la API Server.

Componentes de los Nodos

Ahora veamos para qué se utilizan los componentes encontrados en los Nodos de Kubernetes.

- **Docker:** aquí encontramos a Docker, que se utiliza para correr nuestras aplicaciones en un ambiente encapsulado.
- **Kubelet:** es un proceso que se encarga de mantener el estado de trabajo y el nodo mismo. Lee las configuraciones necesarias de etcd y recibe los comandos del nodo maestro que necesite para trabajar así como también envía el estado del nodo cada pocos segundos. Además también se encarga de los port forwarding, configuraciones de la red, etc.
- **Kubernetes-proxy:** es quien se encarga de que los servicios se encuentren disponibles para el exterior y también del balanceo de carga y enrutamiento de tráfico por dirección de IP.

Conceptos Importantes

Además de la arquitectura y sus componentes, hay otros conceptos básicos que se deben tener en cuenta y con los cuales nos tenemos que ir familiarizando para poder trabajar cómodamente con Kubernetes.

- **Cluster:** Es un grupo de máquinas u ordenadores, que pueden ser físicos o bien virtuales, que se encuentran unidos muchas veces por una red y que se comportan como una.
- **Nodo:** es básicamente una máquina que se está ejecutando en Kubernetes, en las cuales se pueden programar Pods.
- **Pod:** es un grupo de uno o más containers que comparten el almacenamiento y también sus configuraciones necesarias para ser ejecutados.
- **Replication Controller:** Es quién maneja los fallos antes comentados y recrea en caso de ser necesario los pods. Además, también se asegura de que el número de réplicas del pod se esté ejecutando.
- **Service:** Es una abstracción que define un conjunto lógico de pods y también las reglas y lógica para acceder a ellos.

2.2.5 – Comparación entre contenedores Docker, máquinas virtuales y contenedores Linux

Cada tecnología tiene sus aplicaciones y sus ventajas según las necesidades y circunstancias de cada desarrollo. En la actualidad los contenedores en general y Docker en particular se están convirtiendo en una tecnología indispensable y cada vez se utilizan para más cosas, no solo para desplegar aplicaciones en producción, sino también para crear entornos de desarrollo replicables entre todos los miembros de un equipo, asegurar que las aplicaciones se van a ejecutar igual en todos los entornos (desarrollo, pruebas y producción), etc.

Docker sin duda ofrece ventajas muy importantes en todas las fases de desarrollo de software. Los contenedores Docker a diferencia de las máquinas virtuales y los contenedores Linux (LXC) tienen muchas ventajas que lo hacen un mecanismo de virtualización menos pesado en cuanto a almacenamiento, más ágil y portable,

debido a que pueden ser creados y lanzados en pocos segundos, y las máquinas virtuales y LXC pueden tardar varios minutos para crearse y ponerse en marcha; los contenedores Docker abstraen las aplicaciones del sistema operativo y las máquinas virtuales se encargan de abstraer el hardware y los LXC aíslan procesos y recursos; en docker todas las aplicaciones tienen sus propias dependencias, que incluyen tanto los recursos de software y hardware y utilizan todas las bibliotecas del sistema operativo sobre el cual se ejecutan, mientras que las máquinas virtuales y los LXC se crean con su kernel, su entorno, sus bibliotecas, sus dependencias, etc; Docker tampoco necesita que instalemos un sistema operativo completo para su ejecución a diferencia de las máquinas virtuales y los LXC que si necesitan un sistema operativo instalado (ver Anexo 11) (ver Anexo 12) .

2.2.6 – Requisitos mínimos para la instalación de Docker

Para poder instalar la tecnología de contenedores Docker se deben conocer ciertos requisitos los cuales deben ser cumplidos para así lograr el correcto funcionamiento de la misma. A continuación, se muestran los requisitos mínimos para la instalación de Docker tomando como referencia los sistemas operativos Windows y Linux puesto que son los que se utilizan en la División Territorial de ETECSA en Cienfuegos.

Tabla 1. Requisitos mínimos para la instalación de Docker

Windows	**Linux**
• Para instalación de Docker se necesita contar con Windows 7 x64 en adelante que admita la tecnología de virtualización de hardware, y esté habilitada. • Acceder a la página oficial de Docker y descargar los componentes necesarios para su instalación.	• Docker solo funciona en una instalación de Linux de 64 bits. • Para instalar Docker en Linux se ha tomado tres S.O usados frecuentemente como son Centos, Ubuntu, Fedora. • En Centos debe contar con la versión 7x64 en adelante. • En Ubuntu se debe contar con la versión 14.04 x64 en adelante.

	• En Fedora se debe contar con la versión 22 x64 en adelante.
Requisitos mínimos de hardware	
• Memoria RAM de 1GB. • Espacio en disco de 20 GB.	

2.3 – Procedimiento para el uso de la tecnología de contenedores Docker

En ETECSA y en cualquier empresa o entidad con condiciones similares la secuencia de pasos a seguir para el uso de la tecnología de contenedores Docker seria:

1. Verificar que exista conexión a internet.
2. Seleccionar el(los) equipo(s) que tengan los requisitos mínimos para la instalación de Docker.
3. Instalación de la tecnología de contenedores Docker.
4. Crear el repositorio para su explotación por parte de los usuarios.

A continuación, se detalla información útil para la implementación en la práctica de este procedimiento; desde los principales comandos a utilizar y el proceso de instalación hasta el trabajo con sus componentes.

2.3.1 – Principales comandos para el trabajo con la tecnología de contenedores Docker

Docker cuenta con una serie de comandos que nos permiten el interactuar con cada uno de sus componentes facilitando así el uso y explotación de cada una de sus prestaciones que lo han llevado a convertirse en una de las tecnologías de contenedores más populares.

Antes de comenzar a usar Docker, se deben estudiar y comprender los comandos que el mismo ofrece para realizar diferentes operaciones, logrando así una mayor fluidez en el trabajo y un mejor entendimiento del funcionamiento de esta tecnología (ver Anexo 13) (ver Anexo 14) (ver Anexo 15).

Escribiendo docker en la terminal aparece una lista de las opciones disponibles.

Si se requiere saber cómo funciona un comando concreto se debe escribir el comando “docker COMANDO --help”.

Ejemplo

docker save --help

Usage: docker save [OPTIONS] IMAGE [IMAGE...]

Save one or more images to a tar archive (streamed to STDOUT by default)

--help Print usage

-o, --output Write to a file, instead of STDOUT

Para el uso de cualquier comando de Docker podemos seguir la siguiente sintaxis:

Uso: Docker [OPCIONES] COMANDO

Por ejemplo, para crear una imagen llamada *dockerbuild-example:1.0.0* partir de un Dockerfile en el directorio de trabajo actual se utilizan los siguientes comandos:

$ ls Dockerfile Dockerfile2

$ docker build -t dockerbuild-example:1.0.0 .

$ docker build -t dockerbuild-example-2:1.0.0 -f Dockerfile2

Por ejemplo, Para ejecutar un contenedor de forma interactiva, pase las opciones -it :

$ docker run -it ubuntu:14.04 bash root@8ef2356d919a:/# echo hi hi root@8ef2356d919a:/#

Ejemplo, para establecer límite de memoria y desactivar el límite de intercambio se ejecuta el comando:

$ docker run -it -m 300M --memory-swap -1 ubuntu:14.04 /bin/bash

2.3.2 – Instalación en Linux

A continuación, se procederá a la instalación de un motor de Docker en Ubuntu. Para ello será necesario los siguientes requisitos: conexión a internet y un pc con una distribución Linux.

1. Llamar a la terminal: inicia Ubuntu y abre la terminal, por ejemplo, con la combinación de teclas [STRG] + [ALT] + [T].
2. Para ejecutar acciones en la terminal como administrador hay que anteponer el comando sudo a la llamada al programa.
3. Actualizar listas de paquetes: utiliza el siguiente comando para actualizar el índice local de paquetes de tu sistema operativo y confirma con [ENTER].

$ apt-get update

Una vez que te has identificado como usuario raíz con tu contraseña, Ubuntu comienza con el proceso de actualización, mostrando su estado en la terminal.

4. Docker se puede descargar como un paquete DEB para proseguir con su instalación manual. O descargar el paquete de instalación necesario en el proceso de instalación directamente:

Para realizar la descarga como paquete deb y realizar su instalación manual obtenemos la instalación del sitio https://mirrors.aliyun.com/docker-ce/linux

Descarga los archivos DEB de la versión de Ubuntu que se desea.

https://mirrors.aliyun.com/docker-ce/linux/ubuntu/dists/bionic/pool/stable/amd64/

Y se escoge la versión más actualizada del producto.

Luego de copiar los archivos descargados en su computadora pueden cargar la plataforma de contenedores desde el propio repositorio del sistema operativo.

El siguiente comando se utiliza para instalar uno de los paquetes de Docker ofrecidos por la comunidad Ubuntu:

```
$ apt-get install -y docker.io
```

Para comprobar que docker está instalado correctamente procedemos a ejecutar el comando.

```
$ docker info
```

5. Para realizar su instalación y descargar la instalación en el mismo proceso seguimos los siguientes pasos (para este proceso es necesario la conexión a internet).

Como primer paso procedemos a modificar el sources.list para agregar los repositorios alternos de Docker con el comando nano /etc/apt/sources.list y colocamos dentro de el la siguiente informacion de acuerdo a la distribucion linux escogida.

[Ubuntu]

Agregamos:

deb [arch=amd64] https://download.docker.com/linux/ubuntu bionic stable

Alterno:

deb [trusted=yes] https://mirrors.tuna.tsinghua.edu.cn/docker-ce/linux/ubuntu bionic stable

[Debian]

Agregamos:

deb [arch=amd64] https://download.docker.com/linux/debian buster stable

Alterno:

deb [trusted=yes] https://mirrors.tuna.tsinghua.edu.cn/docker-ce/linux/debian buster stable

Si algun repositorio pide llaves con los sigiuientes comandos podran ser añadidas:

[Ubuntu]

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | apt-key add -

Alterno:

curl -fsSL https://mirrors.tuna.tsinghua.edu.cn/docker-ce/linux/ubuntu/gpg | apt-key add -

[Debian]

curl -fsSL https://download.docker.com/linux/debian/gpg | apt-key add -

Alterno:

curl -fsSL https://mirrors.tuna.tsinghua.edu.cn/docker-ce/linux/debian/gpg | apt-key add -

Se actualiza el sistema con los comandos:

apt update

apt upgrade

Una vez ya el sistema totalmente actualizado, se procede a instalar las dependencias necesarias para que corra Docker en el sistema.

apt install apparmor-utils apt-transport-https avahi-daemon jq network-manager socat qrencode

apt install curl net-tools ca-certificates software-properties-common dbus

Finalmente se procede a instalar docker

apt install docker-ce docker-compose docker-ce-cli containerd.io

Ahora con docker ya instalado solo queda configurarlo para que use un mirror diferente al de dockerhub y así poder bajar las imágenes sin restricciones.
Crear en /etc/docker/ el fichero daemon.json con el comando cat> daemon.json
Para modificar el contenido del fichero y colocar la informacion se ejecuta el comando nano /etc/docker/daemon.json.
Agregar el siguiente contenido.

```
{
   "registry-mirrors": [
   "https://rw21enj1.mirror.aliyuncs.com",
   "https://dockerhub.azk8s.cn",
   "https://reg-mirror.qiniu.com",
   "https://hub-mirror.c.163.com",
   "https://docker.mirrors.ustc.edu.cn",
   "https://1nj0zren.mirror.aliyuncs.com",
   "https://quay.io",
   "https://docker.mirrors.ustc.edu.cn",
   "http://f1361db2.m.daocloud.io",
   "https://registry.docker-cn.com"
   ]
}
```

Se reincia para que se vean los cambios.

```
systemctl daemon-reload
systemctl restart docker
```

6. Este paso a continuación es para si nos encontramos detrás de un proxy padre y es opcional.
Crear la carpeta docker.service.d

```
mkdir -p /etc/systemd/system/docker.service.d
```

Creamos el fichero http-proxy.conf

```
nano /etc/systemd/system/docker.service.d/http-proxy.conf
```

Dentro agregamos:

[Service]

Environment="HTTP_PROXY=http://user:password@proxyip:port/"

Environment="HTTPS_PROXY=http://user:password@proxyip:port/"

Environment="NO_PROXY= hostname.example.com,localhost,127.0.0.1"

Recargamos la configuración y reiniciamos el servicio:

systemctl daemon-reload

systemctl restart docker

Para comprobar que docker está instalado correctamente procedemos a ejecutar el comando.

docker info

2.3.3 – Instalación en Windows

A continuación, se procederá a la instalación de un motor de Docker en Windows. Para ello será necesario los siguientes requisitos: para la instalación de Docker se necesita contar con Windows 7 x64 en adelante que admita la tecnología de virtualización de hardware, y esté habilitada. También contar almeno con 1Gb disponible de RAM y 20 Gb de espacio en disco.

1. Para la instalación descargaremos el fichero de ejecución https://github.com/boot2docker/windows-installer/releases/tag/v1.5.0 o también podemos ir a la página Docker CE for Windows, y en ella encontraremos los instaladores para Windows 10.

2. Simplemente se ejecuta el archivo que se ha descargado de la página web, abrirá una ventana con la instalación de Docker.

3. Nos preguntará el lugar donde instalaremos Docker.

Escoger el destino donde se desea instalar y dar click en siguiente (next) para continuar con la instalación.

4. Lo siguiente que preguntará el programa será qué queremos instalar, lo que hace Docker en Windows es, crear una máquina virtual a la cual le instala un sistema operativo Linux “Boot2Docker”, ya que Docker necesita utilizar el kernel de Linux, para esto utiliza el programa de virtualización VirtualBox, también instala la herramienta msys-git que utiliza como terminal para conectarse a la máquina virtual que creará.

Se señalan los programas que se necesitan instalar para su correcta ejecución para ello deben quedar activadas las opciones Boot2Docker, VirtualBox, MSYS-git y dar siguiente para continuar.

Luego especifica la ruta de la instalación los programas que se decidieron instalar, si todo está correcto, se procede a dar install para ahora si instalar el docker.

5. Luego para su ejecución creará un acceso directo en el escritorio.

6. Al ejecutar la aplicación, lo que hace es verificar si existe la máquina virtual en Docker si no existe, la crea, y si existe, se inicia la máquina virtual y se conecta mediante la creación de una clave de conexión con ssh.

2.3.4 – Trabajo con los componentes del flujo de trabajo de la tecnología de contenedores Docker

La tecnología de contenedores Docker cuenta con una serie de componentes que forman parte de su flujo de trabajo, conforman estos componentes las Imágenes, los Contenedores, los archivos Dockerfile y el repositorio de Docker que puede ser el Docker Hub o el Docker Registry.

2.3.4.1 – Imagen

Las imágenes son plantillas de sólo lectura que se usan como base para lanzar un contenedor. Una imagen Docker se compone de un sistema de archivos en capas una sobre la otra. En la base tiene un sistema de archivos de arranque, bootfs (parecido al sistema de archivos de Linux) sobre el que arranca la imagen base.

Cada imagen, también conocida como repositorio, es una sucesión de capas. Es decir, al arrancar un contenedor se hace sobre una imagen, a la cual se llama imagen base. Con el contenedor corriendo, cada vez que realizamos un cambio en el contenedor Docker añade una capa encima de la anterior con los cambios, pero dichas modificaciones no serán persistentes, los cambios no los hacemos en la imagen (recuerde que es de sólo lectura), por lo que se debe guardar creando una nueva imagen con los cambios.

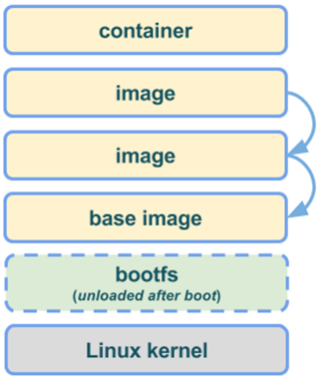


Figura 1. Representación del funcionamiento de las imágenes

Como se ve en la imagen anterior cuando un contenedor se pone en marcha a partir de una imagen, Docker monta un sistema de ficheros de lectura escritura en la parte superior de las capas. Aquí es donde los procesos del contenedor Docker serán ejecutados (container).

Cuando Docker crea un contenedor por primera vez, la capa inicial de lectura y escritura está vacía. Cuando se producen cambios, éstos se aplican a esta capa, por ejemplo, si se desea cambiar un archivo, entonces ese archivo se

copia desde la capa de sólo lectura inferior a la capa de lectura y escritura. Seguirá existiendo la versión de sólo lectura del archivo, pero ahora está oculto debajo de la copia.

Con el conocimiento básico previamente adquirido de Docker, iniciaremos con la creación de una imagen en base Ubuntu.

El posterior desarrollo estará soportado sobre Ubuntu Bionic.

Para crear estas imágenes utilizaremos la herramienta Debootstrap, instalándola, ya que no es propiamente de Ubuntu.

1. Para subir los privilegios y convertirse en súper usuario (root) ejecutamos el siguiente comando.

```
$ sudo su
```

2. Como buena práctica sobre el trabajo en Linux, recomendamos siempre actualizar el sistema de paquetes antes de comenzar.

```
$ apt update
```

3. Si el debootstrap no está instalado procedemos a su instalación con el comando.

```
$ apt install debootstrap
```

4. Iniciamos el proceso de creación de la imagen

```
$ debootstrap bionic[1]bionic [2]file:/home/beatriz/Repositorios/bionic[3]
```

Con el código anterior creamos una imagen en base Ubuntu Bionic, con nombre bionic partiendo de un repositorio localhost, el que hace referencia a la ubicación donde se encuentre el repositorio o mirror de Ubuntu Bionic en nuestra propia máquina, este puede ser intercambiado por un repositorio nacional (repositorio.cu) en caso que no poseamos el mirror en nuestra propia máquina.

---

[1] **bionic:** es la versión de Linux que se está usando

[2] **file:/home/beatriz/Repositorios/bionic:** es la dirección en memoria de donde está ubicado el repositorio

[3] **bionic:** es el nombre de la imagen a ser creada

La ejecución de este comando puede demorar unos 6 o 7 minutos.

La ejecución anterior nos generará un directorio con la estructura de carpetas como la de cualquier sistema operativo Ubuntu (apt, etc, opt, var…), se procede a empaquetar estos directorios para importarles en Docker como una imagen propia

5. Antes de proceder editamos el fichero /bionic/etc/apt/sources.list con el objetivo de que todo el proceso de instalación de paquetes en la imagen se haga desde los servidores locales o nacionales según nuestro interés.

$ nano bionic/etc/apt/sources.list

Es importante aclarar que no estamos refiriéndonos al fichero sources.list del sistema que contiene la instalación de Docker, sino al referenciado en el directorio de la imagen bionic creada con anterioridad.

En el archivo se coloca la dirección del repositorio

deb file:///home/beatriz/Repositorios/bionic bionic main universe multiverse restricted

6. Se empaqueta en un tar el directorio creado mediante debootstrap, luego se crea una imagen Docker limpia y se importa el contenido de la imagen bionic hacia ella quedando la imagen personalizada, creada desde servidores locales o nacionales, sin la necesidad de utilizar servicios de Docker Hub.

$ tar -cf bionic.tar bionic

$ docker import bionic.tar ubuntu18_04: debootstrap

7. Esta imagen ya puede ser ejecutada desde Docker normalmente, así como reutilizable para cualquier configuración personalizada. La ejecución de la imagen la realizamos con el siguiente comando.

$ docker run bionic cat /etc/lsb-release

8. Luego para ver si fue creada procedemos a listar las imágenes con el comando.

$ docker images

Las imágenes, como hemos visto, son plantillas de sólo lectura, que usamos de base para lanzar contenedores. Por tanto, lo que hagamos en el contenedor sólo persiste en ese contenedor, las modificaciones no las hacemos en la imagen.

Si queremos que dichos cambios sean permanentes, debemos crear una nueva imagen con el contenedor personalizado.

Vamos a realizar un ejemplo. Tenemos la imagen base de Ubuntu.

Lanzamos un contenedor con esa imagen base en el modo interactivo que vimos anteriormente.

Ahora vamos a guardar los cambios realizados en la imagen. Tenemos que salir del contenedor y ejecutar el comando “commit”.

Para poder utilizar esta imagen con los cambios, tenemos que crear una nueva imagen, con el comando:

$ docker commit –m[4] “Instalando Git” –a[5] “María Reyes” c906d56c7aa5 [6] git/ubuntu:v1[7]

Con commit creamos una nueva imagen en nuestro repositorio local.

Una vez que tenemos disponible una imagen podemos ejecutar cualquier contenedor.

2.3.4.2 – Contenedor

Los contenedores son creados a partir de una imagen ya existente. Un contenedor específico, solo puede existir una vez, pero se puede crear múltiples contenedores de una misma imagen.

Iniciaremos con la creación de un contenedor, el cual usa la imagen creada anteriormente.

---

[4] **-m:** añadimos un comentario.

[5] **-a:** autor de la imagen.

[6] **c906d56c7aa5:** identificador del contenedor.

[7] **git/ubuntu: v1:** el nombre que le damos a la imagen.

1. Para subir los privilegios y convertirse en súper usuario (root) ejecutamos el siguiente comando.

$ sudo su

2. Actualizamos el sistema de paquetes.

$ apt update

Para la creación de un contenedor es imprescindible primeramente tener creada una imagen.

Podemos crear un contenedor con el comando run.

Sintaxis:

docker run [OPTIONS] IMAGE [COMMAND] [ARG....]

Entre las opciones se encuentran (lista completa aquí):

-a, --attach: para conectarnos a un contenedor que está corriendo.

-d, --detach: corre un contenedor en segundo plano.

-i, --interactive: habilita el modo interactivo.

--name: le pone nombre a un contenedor.

El comando run primero crea una capa del contenedor sobre la que se puede escribir y a continuación ejecuta el comando especificado. Con este comando hemos ejecutado un contenedor, sobre la imagen Ubuntu, que ha ejecutado el comando echo. Cuando ha terminado de ejecutar el comando que le hemos pedido se ha detenido. Los contenedores están diseñados para correr un único servicio, aunque podemos correr más si hiciera falta. Cuando ejecutamos un contenedor con run debemos especificarle un comando a ejecutar en él, y dicho contenedor sólo se ejecuta durante el tiempo que dura el comando que especifiquemos, funciona como un proceso.

Algo a tener en cuenta es que, si la imagen que estamos poniendo en el comando run no la tuviéramos en local, Docker primero la descargaría y la guardaría en local y luego seguiría con la construcción capa por capa del contenedor.

3. Iniciamos el proceso de creación del contenedor.

$ docker run -it ubuntu18_04:debootstrap[8]

Con el código anterior se crea un contenedor interactivo.

-t: ejecuta una terminal.

-i: nos comunicamos con el contenedor en modo interactivo.

4. Luego para ver si fue creado procedemos a listar los contenedores que han sido creados con el comando.

$ docker ps –a

5. Docker genera automáticamente un nombre al azar por cada contenedor que creamos. Si queremos especificar un nombre en particular podemos hacerlo con el parámetro --name. Para crear un contenedor con su nombre ejecutamos el comando.

$ docker run --name Ubuntu Bionic [9]–it ubuntu18_04: debootstrap [10]

Se pueden limitar los recursos usados por los contenedores usando una serie de comandos (ver Anexo 16) (ver Anexo 17).

Algunos ejemplos de creación de contenedores con limitación de recursos serían:

root@docker:/# docker run -i -t -m 500m ubuntu /bin/bash

root@fbc27c5774aa:/#

Hemos creado un contenedor con un límite de memoria de 500 megas.

2.3.4.3 – Dockerfile

Un Dockerfile es un archivo legible por el demonio Docker, que contiene una serie de instrucciones para automatizar el proceso de creación de un contenedor.

El comando docker build irá siguiendo las instrucciones del Dockerfile y armando la imagen. El Dockerfile puede encontrarse en el directorio en el que estemos o en un repositorio.

---

[8] **ubuntu18_04: debootstrap:** nombre de la imagen

[9] **Ubuntu_Bionic:** nombre del contenedor

[10] **ubuntu18_04: debootstrap:** nombre de la imagen

El demonio de Docker es el que se encarga de construir la imagen siguiendo las instrucciones líneas por líneas y va lanzando los resultados por pantalla. Cada vez que ejecuta una nueva instrucción se hace en una nueva imagen, son imágenes intermedias, hasta que muestra el ID de la imagen resultante de ejecutar todas las instrucciones. El daemon irá haciendo una limpieza automática de las imágenes intermedias.

Estas imágenes intermedias con la caché de Docker son necesarias pues si por alguna razón la creación de la imagen falla (por un comando erróneo, por ejemplo), cuando lo corregimos y volvemos a construir la imagen a partir del Dockerfile, el demonio no iniciará todo el proceso, sino que usará las imágenes intermedias y continuará en el punto dónde falló.

En el Anexo 18 se puede ver una tabla que agrupa los comandos para trabajar con un Dockerfile y la descripción de cada uno de ellos (ver Anexo 18).

Ejemplo de la creación de un Dockerfile:

```
FROM debian
MAINTAINER maria <marcabgom@gmail.com>
ENV HOME /root
RUN apt-get update
RUN apt-get install -y nano wget curl unzip lynx apache2 php5 libapache2-mod-php5 php5-mysql
RUN echo "mysql-server mysql-server/root_password password root" | debconf-set-selections
RUN echo "mysql-server mysql-server/root_password_again password root" | debconf-set-selections
RUN apt-get install -y mysql-server
ADD https://es.wordpress.org/wordpress-4.2.2-es_ES.zip/var/www/wordpress.zip
ENV HOME /var/www/html/
RUN rm /var/www/html/index.html
RUN unzip /var/www/wordpress.zip -d /var/www/
```

```
RUN cp -r /var/www/wordpress/* /var/www/html/
RUN chown -R www-data:www-data /var/www/html/
RUN rm /var/www/wordpress.zip
ADD /script.sh /script.sh
 RUN ./script.sh
EXPOSE 80
CMD ["/bin/bash"]
ENTRYPOINT ["./script.sh"]
```

2.3.4.4 – Docker Hub y Docker Registry

El concepto de Docker Hub o Docker Registry inicia con la necesidad de registrar en la nube imágenes de docker dentro de un repositorio, con la tendencia de migrar imágenes docker que contengan sus propias configuraciones empresariales o personales.

Un Registro es un lugar donde almacenar imágenes de contenedores que luego utilizarán en los engines para crear nuestros contenedores.

Los repositorios registros o hub en docker pueden ser públicos o privados; existe un registro oficial de docker y también se puede tener un registro propio ya sea a nivel personal o empresarial.

Un Registro propio nos permitirá que los distintos motores docker que tengamos puedan descargar las imágenes que desarrollemos y que no necesariamente queramos que sean públicas. El Registro es una de las piezas clave a la hora de crear nuestros entornos Docker en cuanto empezamos a crear nuestras propias imágenes (minuto 1).

Desde Docker Inc. se ofrece un servicio de Registry que los usuarios de docker suelen usar a diario: Docker Hub. Este servicio se ofrece como SaaS y tiene una capa de uso gratuita. En este plan gratuito tenemos opción de un Registry privado y uno público.[14]

A pesar de Docker Inc. tener para ofrecer estos servicios es una buena práctica crear un registro propio. Los motivos más evidentes son los siguientes:

- Tener el Registro en nuestra infraestructura nos ahorra ancho de banda y nos da mejor tiempo de acceso/descarga.
- Tener una copia “controlada” de imágenes públicas nos protege ante cambios inesperados, o desaparición, de las mismas.
- Tener tantos namespaces privados como queramos, con las ACL que queramos y la auth conectada contra nuestro sistema de auth preferido.

Creando un repositorio en Docker Hub

1. Para crear un registro en Docker Hub se debe registrar un usuario en la página https://hub.docker.com/ utilizando un correo electrónico.
2. Luego de haber llenado los parámetros y confirmado la activación de la cuenta con el correo requiere la autenticación para ingresar al docker hub.

   Una vez registrados y autenticados presentará una ventana donde se pueden realizar varias tareas como:

   - Crear un repositorio.
   - Buscar imágenes de docker existentes construidas por otros usuarios de docker.
   - Mostrar una lista de todas las instrucciones de Git push y pull que se han realizado en las imágenes del repositorio creado.

   Por cada imagen subida se obtienen sus detalles como el peso y las descargas realizadas por los usuarios, esto hace énfasis de que si se publica una imagen docker en un repositorio de docker Hub será visible para otros usuarios y podrán obtenerla solo con el nombre anticipando la instrucción Git pull.
3. Para crear un repositorio en docker Hub se debe hacer clic en la opción crear repositorio que se encuentra en la parte superior derecha.
4. Después de haber ingresado se muestra la ventana donde se ingresan los parámetros de reconocimiento de la imagen que se desea realizar la instrucción Git push o subida de la imagen docker que tenemos localmente.

   Al establecer los parámetros de construcción de la imagen en el repositorio de docker hub también se determina si es pública o privada.

Luego se da clic en crear y está listo para poder subir la imagen al repositorio. Construyendo de esta manera un recipiente que contendrá eventualmente la imagen de docker, la pregunta es cómo realizar este procedimiento, y la respuesta es sencilla usando las instrucciones de Git push y pull.

5. Para realizar este proceso se debe ejecutar el servidor en Ubuntu que previamente fue instalado docker.
6. Después de tener inicializado los servicios de docker se puede visualizar las imágenes de docker existentes con el comando docker images.
7. Al ejecutar dicha instrucción se mostrará el tamaño de la imagen, el tag, el id de la imagen seguido del nombre y adicionalmente la fecha de la última iteración de la imagen de docker.
8. Para lograr introducir las instrucciones de Git push y pull se deben realizar algunos pasos en este caso se hace referencia al tag de la imagen que desea subir al repositorio de docker hub, antes de realizar este proceso debemos tener bien definido los siguientes parámetros: Nombre de usuario, Nombre Repositorio, Email.
9. Una vez definido estas credenciales se puede realizar una referencia de la imagen de docker al repositorio con un nombre de usuario de la siguiente forma:

   $ docker tag [IMAGE ID] [Nombre de usuario]/[Nombre Repositorio]:latest

   Al ejecutarlo no se aprecia mayor cambio porque solo realiza una referencia de nombre hacia un repositorio existente en docker Hub.
10. El siguiente paso para lanzar la imagen al repositorio virtual es realizar una autenticación en docker usando la instrucción docker login.
11. Luego se debe ejecutar la siguiente la siguiente instrucción:

    $ docker push [Nombre de usuario]/[Nombre Repositorio]
12. Finalmente se tiene subida la imagen en el repositorio docker Hub.

El beneficio de este procedimiento es poder realizar migraciones de contenedores ya configurados y simplemente realizar instrucciones de Git para subirlas u obtenerlas.

De este modo la imagen que se alojó en el repositorio de docker hub se mostrará estará listo para realizar las instrucciones de Git push y pull.

De cierto modo esto no termina aquí ya que muchos usuarios están construyendo varias imágenes en sus propios repositorios; además la mayoría de ellos tienden a compartir sus imágenes docker haciendo del uso de esta herramienta más sencilla y con mucho soporte entre la comunidad de docker hub.

Para comenzar a hacer búsquedas de imágenes en la comunidad de usuarios de docker hub se debe realizar lo siguiente:

- Establecer qué imagen se desea obtener, ejemplo se requiere de una imagen de postgres en la versión 9.2 y para ello se usó la opción de búsqueda ubicada en la parte superior de la página y se debe ingresar el nombre de la imagen que se desea adquirir.
- Luego de establecer la búsqueda se muestra una lista de todas las imágenes con ese nombre y versión que se estableció.
- Para ver la información del contenedor como sus credenciales y otros parámetros se debe ingresar a la opción detalles que despliega toda la información acerca de la imagen docker.

Creando un Docker Registry

Un Registry es un lugar donde almacenar imágenes de contenedores que luego serán utilizados en los engines para crear los contenedores.[14]

Un Registry propio permitirá que los distintos motores docker existentes puedan descargar las imágenes que se desarrollen y que no necesariamente se requiera que sean públicas.

Se puede utilizar Docker Hub, tanto para descargar imágenes de contenedores, como para subir imágenes para compartirlas con otras personas. O simplemente, para tenerlas ahí, a disposición para cuando sea necesario utilizarla. Sin embargo, Docker Hub no es mas que otro servicio. Se trata de un servicio Open Source, con

lo que también se puede tener instalado en el equipo. Y por supuesto,también se puede tener instalado como un contenedor. Así, Docker Registry es un servidor de imágenes docker. Se trata de un servidor que lo puedes tener instalado donde lo necesites, si es que lo necesitas. Desde una Raspberry Pi, a un NAS o algo mucho mas potente. Esto ya depende de para que se necesite y por supuesto, para lo que se quiera utilizar.
Evidentemente, para una empresa de desarrollo de software se trata de una excelente solución. Una forma de tenerlo todo bajo control, y solo liberar las imágenes cuando el producto sea estable o cuando se considere.

Así, en este subepigrafe se mostrara los pasos para crer un Docker Registry, o un servidor de imágenes docker, como puedes tenerlo instalado, y que puedes hacer con él.
Algunas de las razones para tener un servidor de imágenes docker local son:

- Permite controlar donde se están guardando las imágenes.
- Controlar todo el proceso de distribución de imágenes, sin necesidad de recurrir a terceros.
- Con todo ello se integra el almacenamiento y distribución de imágenes dentro del flujo de trabajo.

Entre los motivos mencionados para montar un registry uno de ellos era el hecho de tenerlo “cerca” de nuestra infraestructura. El punto exacto ya dependerá de nuestras necesidades**.** Podemos tenerlo en nuestra cloud privada o en la cloud pública. Podemos tener uno para toda la organización o implementar uno por proyecto de suficiente entidad y que se ejecute dentro de la infraestructura del mismo. Se pretende que sea sencillo de montar y que la decisión no dependa de este factor. Una cosa muy importante a tener en cuenta es la necesidad de almacenamiento.

Para comenzar con la creación del Registry se ejecutará un contenedor de la imagen registry y se creará los certificados que permitan accederlo en nuestra red

local. Supongase que el registry estará en la IP 10.10.1.37 y nuestro cliente es el 10.10.1.110.

1. Descargar y configurar la imagen

   Según la documentación oficial de docker, el puerto por defecto para el registro es el 5000, vamos a seguir usando el mismo por lo que en algunos pasos lo omitiremos de los comandos.

   $ docker run -d -p 5000:5000 --restart=always --name registry \-v /mnt/registry:/var/lib/registry registry:2

   De esta forma tendremos un contenedor que puede ser accedido solo de manera local, esto para fines didácticos está bien, pero es inútil para una aplicación real; un registry tiene que poder enviar/recibir imágenes de clientes al menos en su red local. Si ejecutaste el comando anterior puedes detener la imagen y borrarla con los siguientes comandos:

   $ docker stop registry && docker rm registry

2. Crear certificados en el servidor

   Vamos a crear los certificados necesarios para ello. Necesitas tener instalado openssl y editar el archivo

   /etc/ssl/openssl.cnf

   Agregamos una regla en la sección [v3_ca] Para tener algo como lo siguiente:

   [v3_ca]

   subjectAltName = IP:10.10.1.37

   Vamos a crear una carpeta en nuestro home llamada certs, después de ello creamos nuestras llaves:

   $ openssl req -newkey rsa:4096 -nodes -sha256 -keyout ~/certs/domain.key -x509 -days 1365 -out ~/certs/domain.crt

3. Ahora vamos a ejecutar un contenedor con los certificados que creamos:

```
$ docker run -d \
  --restart=always \
  --name registry \
  -v ~/certs:/certs \
  -e REGISTRY_HTTP_ADDR=0.0.0.0:443 \
  -e REGISTRY_HTTP_TLS_CERTIFICATE=/certs/domain.crt \
```

```
-e REGISTRY_HTTP_TLS_KEY=/certs/domain.key \
-e REGISTRY_STORAGE_DELETE_ENABLED=true \
-p 443:443 \
registry:2
```

Podemos mirar el contenido de nuestro registry a través de la siguiente URL: https://10.10.1.37/v2/_catalog

4. Certificados en los clientes

   Si intentamos conectarnos al registry desde un cliente, tendremos una salida como la siguiente:

   [paklei@CentOS ~]$ docker login 10.10.1.37
   Username: a
   Password:
   Error response from daemon: Get https://10.10.1.37/v2/: x509: certificate signed by unknown authority

   Vamos a configurar nuestro cliente para que confíe en el certificado self-signed que generamos previamente en el servidor. En cada uno de los clientes que se conectarán al registry necesitamos copiar el certificado generado en nuestro host (el archivo domain.crt que se encuentra en la carpeta certs que creamos previamente), para ello necesitamos crear el siguiente archivo:

   /etc/docker/certs.d/10.10.1.37/ca.crt

   -----BEGIN CERTIFICATE-----
   muchasLetras
   HolaMundo!!
   MuchosNúmeros
   OtroTantoMásDeLetras
   ....PuntosSuspensivosPorQueNo?
   -----END CERTIFICATE-----

Con lo anterior podemos conectarnos a nuestro registry

5. Ahora vamos a probar que podamos hacer un push y pull desde nuestro cliente. Vamos a descargar la imagen de alpine:latest, crear un tag con la nomenclatura necesaria y el push.

   $ docker pull alpine:latest

   $ docker tag alpine:latest 10.10.1.37/alpine-test-push

   $ docker push 10.10.1.37/alpine-test-push

6. Para consultar que nuestro registry tenga la imagen podemos ejecutar en GET en nuestro navegador y agregar la excepción del certificado self-signed, deberíamos mirar un response como el siguiente:

   https://10.10.1.37/v2/_catalog

   {"repositories":["alpine-test-push"]}

7. Ahora sabemos que nuestro registry puede recibir las imágenes que le sean enviadas por el cliente 10.10.1.100, vamos a comprobar que nuestro cliente puede hacer pull sin problemas de nuestro registry, para ello vamos a borrar las imágenes de alpine y alpine-test-push y después intentaremos hacer el pull del registry.

   $ docker image remove alpine:latest

   $ docker image remove 10.10.1.37/alpine-test-push:latest

   $ docker pull 10.10.1.37/alpine-test-push

   $ docker images

8. De esta forma tenemos un registry que puede aceptar cualquier push/pull de un cliente que cuente con el certificado correcto.

2.3.5 – Trabajo con comandos de gestión de la tecnología de contenedores Docker

2.3.5.1 – Docker Swarm

Antes de empezar definiremos el entorno que utilizaremos en la creación del de un clúster Swarm de contenedores Docker.

Trabajaremos con 3 nodos, de los cuales el node-1 tendrá el rol de “master” y los nodos node-2 y node-3 serán “workers”.

1. Crearemos los nodos a través de Docker Machine con los siguientes comandos:

   $ docker-machine create –d virtualbox node-1

   $ docker-machine create –d virtualbox node-2

   $ docker-machine create –d virtualbox node-3

2. A continuación nos conectaremos al node-1 por SSH.
3. Con el siguiente comando creamos el swarm y le añadimos el nodo 1 con el rol de “master”:

   $ docker swarm init –advertise-addr 192.168.99.101

   La dirección IP corresponde a la del node-1.

4. Con el comando “$ docker node ls” podemos ver la información de todos los nodos.
5. Para añadir los nodos al clúster como workers nos conectamos a cada uno de ellos por SSH y ejecutamos el siguiente comando:

   $ docker swarm join --token SWMTKN-1-0fv35ch2t4j7lt7jbyahmvhzld1r02femqoo6b0btrqtkx12yh-317kcrd3f9s64i0bb39uqv09z 192.168.99.101:2377

6. Ahora si nos conectamos al nodo master podremos ver la información de los nodos adjuntos al clúster.

   $ docker node ls

7. A continuación procedemos a la creación de un servicio, Nginx en nuestro caso.

   En el node-1 ejecutamos el comando $docker service create nginx

   Este comando nos creará un servicio Nginx con una sola réplica.

8. Con $ docker service ls podremos ver los servicios que se están ejecutando y cuantas instancias hay de cada uno.
9. El comando “$docker service ps web” nos da información de en qué nodo está corriendo el servicio especificado.

Si accedemos a un navegador y ponemos la dirección IP de cualquiera de los nodos deberíamos ser capaces de acceder a una web.Esto se debe a que un servicio ejecutándose en un nodo es accesible por todos los nodos del clúster.

Otra característica de Docker Swarm es la escalabilidad, con la que podemos conseguir instancias adicionales del mismo servicio. Si quisiéramos escalar este servicio web a 4 réplicas tan solo tendríamos que utilizar el siguiente comando.

$ docker service scale web=4

Podemos observar que hay cuatro tasks corriendo, distribuidas por los tres nodos.

Si quisiéramos que un nodo no tuviera tareas asignadas haríamos que su estado pasara de ACTIVE a DRAIN.

$ docker node update –availability drain node-2

Podemos ver como se ha parado la réplica del node-2 y se ha reiniciado inmediatamente en el node-1.

Para finalizar podemos ver el estado del servicio con el siguiente comando:

$ docker service inspect --pretty web

2.3.5.2 – Docker Network

En este apartado veremos el uso de las redes en Docker, como sabemos Docker tiene 4 redes por defecto, para verlas podemos utilizar el siguiente comando:

$docker network ls

La red Bridge es la red por defecto de Docker, a no ser que especifiques el tipo de red Docker siempre desplegará el contendor en la red Bridge.

Ahora ejecutaremos un contenedor llamado test:

$docker run –itd –name=test ubuntu

Para averiguar la IP asignada a nuestro contenedor tan solo tenemos que ejecutar el comando inspect que nos da información de la red.

$docker network inspect bridge

Para quitar el contenedor de la red tan solo tenemos que desconectarlo. En el comando “disconnect” tenemos que especificar el tipo de red y el nombre del contenedor o en su defecto su container ID.

```
$docker network disconnect bridge test
```

Así mismo no se puede eliminar la red Bridge. Las redes son maneras naturales de aislar o conectar a los contenedores de otros contenedores, por lo que se convierte una herramienta muy útil en Docker.

Como hemos visto en la parte teórica, Docker Engine soporta de forma nativa redes bridge y overlay. La red bridge está limitada a un solo host mientras que la overlay incluye varios hosts. En este caso crearemos una red bridge:

```
$docker network create –d bridge bridge_test
```

El parámetro -d nos permite especificar el uso del driver bridge para nuestra nueva red. Si no ponemos este parámetro Docker siempre utilizará la red bridge.

Comprobamos que se ha creado la red correctamente con el siguiente comando:

```
$ docker network ls
```

Si inspeccionamos la red veremos que está vacía.

```
$docker network inspect bridge_test
```

Para crear aplicaciones web que funcione de forma segura debemos utilizar las redes, que son por definición, proporciona aislamiento para los contenedores.

Ahora añadiremos un contendor que ejecuta una base de datos PostgreSQL a la red bridge_test:

```
$ docker run -d --net=bridge_test --name db training/postgres
```

Si inspeccionamos la red bridge_test veremos que tiene el contenedor db añadido, otra forma de verlo es inspeccionar el contenedor db para ver a qué red está conectado.

```
$ docker inspect db
```

Ahora arrancaremos otro contenedor que ejecuta un servidor web Nginx.

```
$ docker run -d --name web nginx
```

Se puede saber en qué red está corriendo la aplicación Nginx si se inspecciona el contenedor, si se especifica el parámetro "--format" puede ser encontrado más fácilmente, ejemplo:

$ docker inspect --format='{{json .NetworkSettings.Networks}}' web

Podemos observar que la red es bridge por defecto y la dirección IP en este caso es 172.17.0.2.

Ahora accedemos al contenedor que ejecuta la base de datos e intentamos llegar al contendor web mediante un ping.

$docker exec –it db bash

root@1987dcc89a12:/# ping 172.17.0.2

Al estar en diferentes redes vemos que no hay conectividad entre sí.

Para solucionar este problema simplemente tenemos que conectar el contenedor "web" a la red que hemos creado anteriormente "bridge_test", que es donde está el contenedor "db".

$docker network connect bridge_test web

Ahora intentaremos hacer ping otra vez desde el contenedor "db", esta vez especificando el nombre en vez de la dirección IP, al contenedor web y vemos que esta vez funciona correctamente.

$docker exec –it db bash

root@1987dcc89a12:/# ping web

En este caso también podemos observar que la dirección IP del contenedor ha cambiado, ya que ahora no se encuentra en la red bridge, si no en la red bridge_test.

2.3.5.3 – Docker Compose

Como hemos visto anteriormente Docker Compose es una herramienta que permite definir y ejecutar aplicaciones Docker con múltiples contenedores. Para ello utiliza un fichero YAML para configurar los servicios de la aplicación.

Un ejemplo de fichero YAML es el siguiente, donde encontramos la definición de un servicio web Nginx que consta de 3 instancias y que tiene la CPU limitada al 10% y la memoria RAM a 30MB. Estas limitaciones están referidas al host, no a cada uno de los nodos.

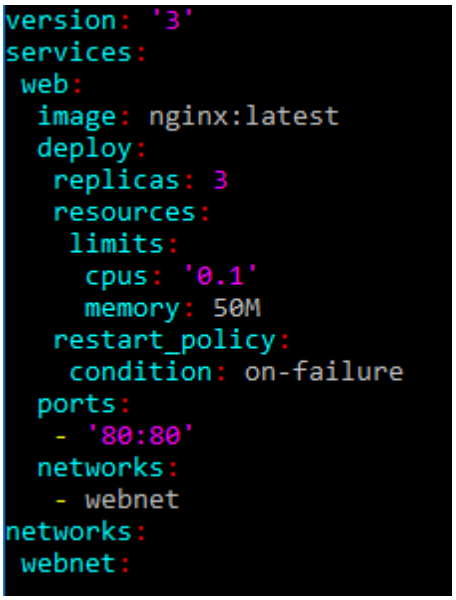


Figura 2. Ejemplo de Fichero YAML

A la hora de escribir este tipo de ficheros hay que ir con mucho cuidado ya que las tabulaciones están prohibidas y la jerarquía, que se marca con espacios, es muy importante que esté bien definida.

Este fichero lo guardaremos con el nombre compose-test.yml. Antes de hacer el "deploy" de este comando iniciaremos el clúster de Docker Swarm en el nodo master con el comando visto anteriormente "$ docker swarm init".

Ahora podremos desplegar nuestra aplicación.

```
$ docker stack deploy -c compose-test.yml web
```

Para obtener más información de la aplicación ejecutamos:

```
$ docker service ls
```

Para ver las tasks del servicio ejecutamos el comando:

```
$ docker service ps web
```

Como hemos visto antes podemos inspeccionar el servicio, así como ver los contenedores corriendo en este servicio.

También es posible escalar la aplicación mediante la modificación del fichero YML, tan solo tendremos que ejecutar otra vez el comando “deploy” para que el cambio se haga efectivo.

Por último, para parar la aplicación ejecutaremos el siguiente comando.

```
$ docker stack rm web
```

2.3.5.4 – Docker Machine

Con docker-machine podemos aprovisionar y administrar múltiples Docker remotos, además de aprovisionar clústers Swarm tanto en entornos Windows, Mac como Linux. Es una herramienta que nos permite instalar el demonio Docker en hosts virtuales, y administrar dichos hosts con el comando docker-machine. Además, esto podemos hacerlo en distintos proveedores (VirtualBox, OpenStack, VMware, etc.). Usando el comando docker-machine podemos iniciar, inspeccionar, parar y reiniciar los hosts administrados, actualizar el cliente y demonio Docker y configurar un cliente Docker para que interactúe con el host. Con el cliente podemos correr dicho comando directamente en el host (ver Anexo 19).

Probamos crear una máquina virtual llamada **prueba** con VirtualBox. Lo hacemos desde una CMD o una terminal Linux con el comando:

```
$ docker-machine create --driver virtualbox prueba
```

En el Anexo 20 podemos ver una tabla que recoge los principales comandos de docker-machine que podemos ejecutar (ver Anexo 20).

Todos estos comandos y el resto que se encuentran en la documentación oficial, están orientados a la administración de las máquinas sobre las que corre el demonio Docker.

A las máquinas podemos acceder a través de la consola de VirtualBox, o través de sh, pero también tenemos la opción de ejecutar comandos en nuestra máquina creada directamente desde nuestra terminal o CMD. Esto es conectar nuestro

cliente Docker con el demonio Docker de la máquina virtual. Esto lo hacemos declarando las variables de entorno de nuestra máquina para indicar a docker que debe ejecutar los siguientes comandos en una máquina en concreto.

2.3.6 – Trabajo con herramientas para el uso de la tecnología de contenedores Docker

2.3.6.1 – Kubernetes

Kubernetes es un proyecto open source de Google para la gestión de aplicaciones en contenedores, en especial los contenedores Docker, permitiendo programar el despliegue, escalado, monitorización de los contenedores, etc.

A pesar de estar diseñado para contenedores Docker, Kubernetes puede supervisar servidores de la competencia como Amazon o Rackspace.

Permite empaquetar las aplicaciones en contenedores y trasladarlos fácil y rápidamente a cualquier equipo para ejecutarlas.

Kubernetes es similar a Docker Swarm, que es la herramienta nativa de Docker para construir un clúster de máquinas. Fue diseñado para ser un entorno para la creación de aplicaciones distribuidas en contenedores. Es un sistema para la construcción, funcionamiento y gestión de sistemas distribuidos.[10]

Requerimientos

- En los hipervisores Linux, VirtualBox o KVM.
- En Windows VirtualBox o Hypervisors Hyper-V

INSTALACIÓN

Podemos instalar Kubernetes de distintas formas dependiendo de las características del entorno en el que queramos instalar. Por ejemplo si vamos a usar Vagrant, Cloud, Ansible,OpenStack, un sistema operativo u otro, máquinas virtuales, servidores, etc.

Por tanto en la documentación oficial nos ofrecen una serie de guías que pueden adaptarse a nuestras necesidades. Hay algunas soluciones que requieren sólo

unos cuantos comandos (ver Anexo 21) y otras que requieren un mayor esfuerzo para la configuración. También disponemos del repositorio oficial de GitHub de Kubernetes, en el que tenemos infinidad de releases y contribuciones que pueden servirnos para nuestro escenario.

Si nos vamos al repositorio Kubernetes/kubernetes, tenemos una subcarpeta llamada cluster, donde encontramos scripts que nos generan clústers automáticamente, incluyendo la red, DNS, nodos y los distintos componentes del Master.

En este ejemplo vamos a trener un escenario con 3 máquinas.

10.0.0.20 .Master: Ubuntu 18.04

10.0.0.21 Minion1: Ubuntu 18.04

10.0.0.22 Minion2: Ubuntu 18.04

1.- Configuración de repositorios para la instalación de paquetes.

Esto lo hacemos en todos los nodos del clúster (Master y Minions).

Creamos el fichero /etc/yum.repos.d/virt7-docker-common-release.repo con el siguiente contenido:

```
[virt7-docker-common-release]
name=virt7-docker-common-release
baseurl=http://cbs.centos.org/repos/virt7-docker-common-release/x86_64/os/
gpgcheck=0
```

2.- Configuración DNS

En todos los nodos del clúster configuramos el fichero /etc/hosts, para evitar el uso de un servidor DNS.

```
127.0.0.1   master
 ::1        master
10.0.0.21   minion1
10.0.0.22   minion2
```

3.- Instalación de Kubernetes y Docker

• MASTER

Instalamos el paquete kubernetes, que entre otras dependencias, instalará Docker, y los paquetes flannel y etcd.

```
[root@master ~]# yum install kubernetes flannel etcd
```

• MINIONS

Instalamos:

```
[root@minion2 ~]# yum install kubernetes flannel
```

4.- Firewall

Desactivamos el Firewall en todos los nodos del clúster, si lo tuviéramos instalado.

```
[root@master ~]# systemctl stop firewalld.service
[root@master ~]# systemctl disable firewalld.service
[root@master ~]# sed -i s/SELINUX=enforcing/SELINUX=disabled/g /etc/selinux/config
[root@master ~]# setenforce 0
```

5.- Configuración de Kubernetes en el MASTER

• Kubernetes Config

Configuramos el fichero /etc/kubernetes/config para que la línea KUBE_MASTER apunte a la IP del Master.

```
###
# kubernetes system config
#
# The following values are used to configure various aspects of all
# kubernetes services, including
#
#   kube-apiserver.service
#   kube-controller-manager.service
#   kube-scheduler.service
#   kubelet.service
#   kube-proxy.service
```

# logging to stderr means we get it in the systemd journal
KUBE_LOGTOSTDERR="--logtostderr=true"

# journal message level, 0 is debug
KUBE_LOG_LEVEL="--v=0"

# Should this cluster be allowed to run privileged docker containers
KUBE_ALLOW_PRIV="--allow-privileged=false"

# How the controller-manager, scheduler, and proxy find the apiserver
KUBE_MASTER="--master=http://10.0.0.20:8080"

• Kubernetes apiserver

Modificamos el fichero /etc/kubernetes/apiserver., en las líneas:

ANTIGUO → KUBE_API_ADDRESS="--insecure-bind-address=127.0.0.1"

NUEVO → KUBE_API_ADDRESS="--address=0.0.0.0"

ANTIGUO → KUBE_ETCD_SERVERS="--etcd-servers=http://127.0.0.1:2379"

NUEVO → KUBE_ETCD_SERVERS="--etcd-servers=http://10.0.0.20:2379"

Comentar la siguiente línea:

#KUBE_ADMISSION_CONTROL="--admissioncontrol=NamespaceLifecycle,NamespaceExists,LimitRanger,SecurityContextDen y,ServiceAccount,ResourceQuota"

• Etcd

Verificamos el puerto de etcd (2379) y lo configuramos para que se puedan unir desde 0.0.0.0:2379

```
# nano /etc/etcd/etcd.conf
ETCD_NAME=default
ETCD_DATA_DIR="/var/lib/etcd/default.etcd"
ETCD_LISTEN_CLIENT_URLS="http://0.0.0.0:2379"
ETCD_ADVERTISE_CLIENT_URLS="http://0.0.0.0:2379"
```

• Reiniciar servicios y los habilitamos para que se inicien en el arranque.

```
[root@master ~]# for SERVICES in etcd kube-apiserver kube-controllermanager kube-scheduler; do
> systemctl restart $SERVICES
> systemctl enable $SERVICES
> systemctl status $SERVICES
> done
```

6.- Definición de una red FLANNEL

Vamos a crear una red para que se asigne a cada nuevo contenedor del clúster. Normalmente se trata de una red 172.17.0.0/16.

Creamos un archivo .json en cualquier carpeta del MASTER, con el siguiente contenido:

```
[root@master ~]# nano flannel-config.json
{
        "Network": "10.0.10.0/16",
          "SubnetLen": 24,
          "Backend": {
                  "Type": "vxlan",
                   "VNI": 1
        }
}
```

En el fichero de configuración /etc/sysconfig/flanneld vemos la URL Y el lugar del archivo para etcd.

```
# Flanneld configuration options
```

```
# etcd url location.  Point this to the server where etcd runs
FLANNEL_ETCD="http://127.0.0.1:2379"

# etcd config key.  This is the configuration key that flannel queries
# For address range assignment
FLANNEL_ETCD_KEY="/atomic.io/network"

# Any additional options that you want to pass
#FLANNEL_OPTIONS=" "
```

Por tanto tenemos que añadir el fichero .json que creamos antes en la ruta /atomic.io/network.

```
[root@master ~]# etcdctl set /atomic.io/network/config < flannel
config.json
        {
                "Network": "10.0.10.0/16",
                  "SubnetLen": 24,
                  "Backend": {
                          "Type": "vxlan",
                           "VNI": 1
                }
        }
```

Lo verificamos:

```
[root@master ~]# etcdctl get /atomic.io/network/config
        {
                "Network": "10.0.10.0/16",
                  "SubnetLen": 24,
                  "Backend": {
                          "Type": "vxlan",
                           "VNI": 1
```

}

}

7.- Configuración de Kubernetes en los MINIONS

Realizamos la misma configuración en los dos minions.

• Kubernetes config

Ponemos la información del Master en la línea KUBE_MASTER del fichero /etc/kubernetes/config.

kube-proxy.service

logging to stderr means we get it in the systemd journal KUBE_LOGTOSTDERR="--logtostderr=true"

journal message level, 0 is debug

KUBE_LOG_LEVEL="--v=0"

Should this cluster be allowed to run privileged docker containers KUBE_ALLOW_PRIV="--allow-privileged=false"

How the controller-manager, scheduler, and proxy find the apiserver KUBE_MASTER="--master=http://10.0.0.20:8080"

• Kubelet

Aquí proporcionamos la información sobre el servidor de la API y el hostname.

#nano /etc/kubernetes/kubelet

###

kubernetes kubelet (minion) config

The address for the info server to serve on (set to 0.0.0.0 or "" for all interfaces)

KUBELET_ADDRESS="--address=0.0.0.0"

The port for the info server to serve on

```
# KUBELET_PORT="--port=10250"

# You may leave this blank to use the actual hostname
KUBELET_HOSTNAME="--hostname-override=minion2"

# location of the api-server
KUBELET_API_SERVER="--api-servers=http://10.0.0.20:8080"

# pod infrastructure container
KUBELET_POD_INFRA_CONTAINER="--pod-infra-container
image=registry.access.redhat.com/rhel7/pod-infrastructure:latest"
# Add your own!
KUBELET_ARGS=" "
```

- Red FLANNEL

  Modificamos sólo una línea dónde informamos del servidos Master.

```
#nano /etc/sysconfig/flanneld
# Flanneld configuration options
# etcd url location.  Point this to the server where etcd runs
FLANNEL_ETCD="http://10.0.0.20:2379"
# etcd config key.  This is the configuration key that flannel queries
# For address range assignment
FLANNEL_ETCD_KEY="/atomic.io/network"
# Any additional options that you want to pass
#FLANNEL_OPTIONS=" "
```

- Reiniciamos servicios en los Minions.

```
# for SERVICES in kube-proxy kubelet docker flanneld; do
systemctl restart $SERVICES
systemctl enable $SERVICES
systemctl status $SERVICES
done
```

- Comprobamos la red.

```
[root@minion2 ~]# ip -4 a|grep inet
inet 127.0.0.1/8 scope host lo
inet 10.0.0.22/24 brd 10.0.0.255 scope global dynamic eth0
inet 172.17.0.1/16 scope global docker0
inet 10.0.81.0/16 scope global flannel.1
```

También podemos hacer una consulta a etcd.

```
[root@minion1 ~]# curl -s http://10.0.0.20:2379/v2/keys/atomic.io/network/subnets | python -mjson.tool
{
    "action": "get",
    "node": {
        "createdIndex": 14,
        "dir": true,
        "key": "/atomic.io/network/subnets",
        "modifiedIndex": 14,
        "nodes": [
        {
        "createdIndex": 14,
        "expiration": "2016-06-18T00:53:49.78188336Z",
        "key": "/atomic.io/network/subnets/10.0.65.0-24",
        "modifiedIndex": 14,
        "ttl": 86172,
                                                    "value":
"{\"PublicIP\":\"10.0.0.21\",\"BackendType\":\"vxlan\",\"BackendData\":{\"VtepMAC\":\"52:db:30:ee:7d:8f\"}}"
        },
        {
            "createdIndex": 24,
            "expiration": "2016-06-18T00:54:18.551834481Z",
            "key": "/atomic.io/network/subnets/10.0.81.0-24",
```

"modifiedIndex": 24,
"ttl": 86201,
"value":
"{\"PublicIP\":\"10.0.0.22\",\"BackendType\":\"vxlan\",\"BackendData\":
{\"VtepMAC\":\"ba:95:a5:05:19:59\"}}"
}
]
}
}

Podemos verificar los nodos del clúster si en el Master ejecutamos:

kubectl get nodes

Hasta aquí la instalación. Los siguientes pasos para la creación de Pods y servicios son comunes a las distintas configuraciones. Primero podemos crear los servicios y luego los Pods o al revés.

CREANDO PODS y RC

Ahora vamos a explicar cómo crear Pods y Replications Controller. Podemos hacerlo de dos maneras, con un fichero (yaml, json) o por línea de comandos. Vemos las dos.

1.- Por línea de comandos Lo hacemos con el comando kubectl, que es el que se encarga de interactuar con la API de Kubernetes.

kubectl run webserver-nginx –image=nginx --replicas=3 --port=80

run: sirve para arrancar un pod.

my-nginx: es el nombre que va ha recibir.

--image: la imagen base para construir el pod.

--replicas: número de pods que se han de crear.

--port: el puerto en el que escucha.

Comprobamos los pods que tenemos con kubectl get pods.

Si nos fijamos en el estado pone que se está creando. Esperamos un poco para que temrinen de crearse y vemos el cambio de estado en pocos segundos.

Vemos que los pods se crean todos con el mismo nombre y un identificador único al final. Para ver información de un pod:

[root@master ~]# kubectl describe pod webserver-nginx-124355620-24z2p

Nos ofrece muchísima información, entre ella:

- ID del contenedor.
- Imagen base
- ID imagen
- Estado
- Fecha de creación
- Puerto
- IP
- Nodo

Para borrar un pod usamos:

kubectl delete pod <nombre_pod>

Vemos que lo borra, pero que automáticamente se crea uno nuevo con un nuevo nombre y su edad es menor que la del resto.

Cuando creamos un pod con el comando run, se crea automaticamente un "deployment" que administra los pods. El deployment tendrá el mismo nombre que los pods.

Para eliminar los pods permanentemente debemos eliminar primero el deployment, para lo que nos hace falta su nombre (lo conseguimos con el comando de la imagen anterior). Ejecutamos:

kubectl delete deployment DEPLOYMENT_NAME

Ahora comprobamos

Vemos que en el estado los está eliminando. Al cabo de unos segundos no aparece nada al ejecutar el mismo comando. http://kubernetes.io/docs/user-guide/pods/single-container/

Cuando creamos uno o varios pods es recomendable crear un replication controller que se encarga de que dicho grupo esté siempre disponible, actualizado, etc. (en versiones anteriores, la creación por línea de comandos de un pod, generaba automáticamente un RC, pero eso ya no es así, ahora se crea el deployment). Si hay muchos pods, eliminará algunos, si hay pocos los creará. Es recomendable crear siempre un RC aunque sólo tengamos un Pod.

Una vez creado un RC, te permite:

• Escalarlo: puedes escoger el número de réplicas que tiene un pod de forma dinámica.

• Borrar el RC: podemos borrar sólo el RC o hacerlo junto a los pods de los que se encarga.

• Aislarlo: Los pods pueden no pertenecer a un RC cambiando los labels. Si un pod es removido de esta manera, será reemplazado por otro creado por el RC.

Los Replication Controller sólo se pueden crear con un fichero .yml. así que lo vemos en el punto siguiente junto a la creación de los Pods por fichero.

1.- Fichero yml

Creamos un fichero .yml con la definición de nuestro RC, que debe tener la siguiente estructura (recordemos que también podría ser a partir de un fichero en formato json).

```
{
  "apiVersion": "v1",
  "kind": "ReplicationController",
  "metadata": {
    "name": "",
    "labels": "",
    "namespace": ""
  },
  "spec": {
    "replicas": int,
    "selector": {
      "":""
    },
    "template": {
      "metadata": {
        "labels": {
          "":""
        }
      },
      "spec": {
        // See 'The spec schema' below
      }
    }
  }
}
```

Figura 3. Archivo YALM

Donde.

• Kind: siempre tiene que ser ReplicationController.

• ApiVersion: actualmente la v1.

• Metadata: contiene metadatos del RC como.

Name: se requiere si no se especifica generateName. Debe ser un nombre único dentro del espacio de nombres y un valor compatible con RFC1035.

Labels: opcional. Las etiquetas son claves arbitrarias que se usan para agrupar y “ser visto” por recursos y servicios.

Éste campo es muy importante ya que será el campo en el que se fije un RC para saber qué pods tiene que gestionar.

GenerateName: se requiere si “name” no está definido. Tiene las mismas reglas de validación que “name”.

Namespace: opcional. El espacio de nombres del replication controller.

Annotations: opcional. Un mapa de clave/valor que puede ser utilizado por herramientas externas para almacenar y recuperar metadatos sobre objetos.

• Spec: especificaciones. Debe contener:

Replicas: el número de Pods a crear.

Selector: un mapa de clave/valor asignao al conjunto de Pods que este RC administra. Debe coincidir con la clave/valor del campo labels en la sección template.

Template: contiene

metadata: los metadatos con los labels para los pods.

Labels: el esquema que define la configuración de los pods.

spec: vemos cómo debe ser su estructura.

La definición de cada campo lo vemos en la documentación oficial.

Ahora creamos nuestro fichero yaml para crear un replication controller.

$ nano /opt/kubernetes/examples/nginx/nginx-rc.yaml

Ahora creamos el RC con el siguiente comando:

$ kubectl create -f <path.yml>

[root@master ~]# kubectl create -f /opt/kubernetes/examples/nginx/nginxrc.yaml replicationcontroller "rc-nginx" created

Podemos comprobar el estado de nuestro RC con el comando:

kubectl describe rc rc-nginx

Podemos escalar un RC, es decir, que añada pods, con el comando:

kubectl scale rc <nombre_rc> --replicas=<numero>

Tenemos que tener en cuenta que el número será el total de pods, no el número a sumar. Es decir si tenemos 3 pods y queremos dos más, el número tendrá que ser 5. Tenemos que esperar un poco para verlos ya que los va escalando uno a uno.

Para eliminar un RC.

kubectl delete rc <nombre_rc>

Ahora vamos a ver cómo tiene que ser un fichero yaml de creación de un pod.

```
apiVersion: v1
kind: Pod
metadata:
  name: ""
  labels:
    name: ""
  namespace: ""
  annotations: []
  generateName: ""
spec:
  ? "// See 'The spec schema' for details."
  : ~
```

Figura 4. Archivo YALM

Donde la definición de cada campo es la misma que en el fichero de creación de un RC, por lo que no lo repetimos.

Nuestro fichero yaml quedará:

nano /opt/kubernetes/examples/nginx/pod-nginx.yaml

apiVersion: v1

kind: Pod

metadata:

name: my-nginx

```
# Especificamos que el pod tenga un label con clave "app" y valor "nginx" que será lo que vea el RC para saber que tiene que gestionarlo.
        labels:
        app: nginx
spec:
        containers:
                - name: nginx
                image: nginx
                ports:
                - containerPort: 80
        restartPolicy: Always
```

Ejecutamos el comando:

kubectl create -f /op t/kubernetes/examples/nginx/pod-nginx.yaml

Vemos el plod creado.

Vemos si tenemos algún RC o deployment.

No se ha creado ninguno, por lo que esta metodología no es la más adecuada para crear nuestros pods.

CREANDO SERVICIOS

Como hemos explicado anteriormente, podemos crear un servicio (services), y dentro de él podemos tener infinidad de contenedores, dónde Kubernetes hará de balanceador de carga con lo que se llama Replication Controller.

Los pods son volátiles, son creados y destruidos trivialmente. Su ciclo de vida es manejado por los Replication Controllers.

Cada Pod tiene su propia dirección IP, que incluso puede cambiar a lo largo de su vida, lo que supone un problema a la hora de comunicarse con otros Pods. Entonces ¿cómo pueden comunicarse?

Lo hacen con lo que se llama “service”. Un service define un grupo lógico de pods y una política de acceso a los mismos. Los pods apuntan a un servicio por la propiedad label. De esta manera un servicio en Pod que es destruido seguirá siendo accesible en otro Pod gracias a los Labels, incluso si está expuesto desde fuera del clúster.

Una buena práctica es la de crear el servicio primero y luego el RC, por lo que vamos a eliminar el que creamos anteriormente.

Creamos un servicio que será expuesto al exterior.

```
# nano /opt/kubernetes/examples/nginx/svc-nginx.yaml
        apiVersion: v1
        kind: Service
        metadata:
                name: my-nginx-service
        spec:
                type: NodePort
                selector:
                        app: nginx
        ports:
                - port: 80
                protocol: TCP
                name: http
```

Levantamos el servicio con el comando:

```
# kubectl create -f /opt/kubernetes/examples/nginx/svc-nginx.yaml
```

Lo podemos ver con:

Ahora creamos el RC que se va a encargar de los Pods que sirven el service.

```
# kubectl create -f /opt/kubernetes/examples/nginx/nginx-rc.yaml
```

Vemos que se van creando.

Una vez creados.

Podemos ver datos del servicio, como la IP del clúster, con el comando:

```
# kubectl describe service <nombre_servicio>
```

Las Ips de Endpoints son las ips de los minions que sirven el servicio, en este caso nginx.

Nos conectamos a otra máquina que está en la misma red y con curl accedemos a nginx.

2.4 – Conclusiones

En este capítulo se recogen tanto las características como la arquitectura de los contenedores Docker. También quedan explicados aspectos importantes que van desde su proceso de instalación hasta sus componentes y el funcionamiento de cada uno de ellos para así facilitar la utilización de este programa en la DTCF por parte de los trabajadores de tecnologías de la información y dando lugar así a un procedimiento que va a poder ser usado por los programadores y administradores de red de la DTCF.

3 – Desarrollo de pruebas al procedimiento para el uso de la tecnología de contenedores Docker

3.1 – Introducción

En este capítulo se realizarán varias pruebas al procedimiento expuesto en el epígrafe 2.3 del capítulo anterior, además usando los conocimientos adquiridos sobre la tecnología de contenedores Docker en los capítulos y epígrafes anteriores.

3.2 – Escenario sobre el cual se realizaron las pruebas

Las pruebas que a continuación se muestran fueron realizadas en el sistema operativo Linux en su versión Ubuntu 18.04 x64 sin necesidad del uso de una máquina virtual y teniendo conexión a internet.

3.3 – Prueba 1: Instalación de la tecnología de contenedores Docker en sistema operativo Linux

A continuación, se procederá a la instalación de un motor de Docker en Ubuntu. Para ello tendremos como requisitos un pc con Ubuntu 18.04 instalado y conexión a internet.

Modificamos el sources.list para agregar los repositorios alternos de Docker.

```
root@laptop:/home/beatriz# nano /etc/apt/sources.list
```

Agregamos la información q está señalada con un rectángulo rojo:

```
 GNU nano 2.9.3          /etc/apt/sources.list

#deb http://cu.archive.ubuntu.com/ubuntu/ bionic main restricted
#deb http://cu.archive.ubuntu.com/ubuntu/ bionic-updates main restric$
#deb http://cu.archive.ubuntu.com/ubuntu/ bionic universe
#deb http://cu.archive.ubuntu.com/ubuntu/ bionic-updates universe
#deb http://cu.archive.ubuntu.com/ubuntu/ bionic multiverse
#deb http://cu.archive.ubuntu.com/ubuntu/ bionic-updates multiverse
#deb http://cu.archive.ubuntu.com/ubuntu/ bionic-backports main restr$
#deb http://security.ubuntu.com/ubuntu bionic-security main restricted
#deb http://security.ubuntu.com/ubuntu bionic-security universe
#deb http://security.ubuntu.com/ubuntu bionic-security multiverse
################################
## Ubuntu 18.04 (bionic beaver) #
################################
#deb http://repositorio.cfg.etecsa.cu/repositorio/ubuntu/bionic bioni$
deb file:///home/beatriz/Repositorios/bionic bionic main universe mul$
deb [arch=amd64] https://download.docker.com/linux/ubuntu bionic stab$
deb [trusted=yes] https://mirrors.tuna.tsinghua.edu.cn/docker-ce/linu$
```

Agregamos las llaves del repositorio

```
root@laptop:/home/beatriz# curl -fsSL https://download.docker.com/linu
x/ubuntu/gpg | apt-key add -
```

```
root@laptop:/home/beatriz# curl -fsSL https://mirrors.tuna.tsinghua.ed
u.cn/docker-ce/linux/ubuntu/gpg | apt-key add -
```

Finalmente actualizamos nuestro sistema:

```
root@laptop:/home/beatriz# apt update
```

```
root@laptop:/home/beatriz# apt upgrade
```

Una vez ya nuestro sistema totalmente actualizado, procedemos a instalar las dependencias necesarias para que corra Docker en nuestro sistema.

```
root@laptop:/home/beatriz# apt install apparmor-utils apt-transport-ht
tps avahi-daemon jq network-manager socat qrencode
```

```
root@laptop:/home/beatriz# apt install curl net-tools ca-certificates
software-properties-common dbus
```

Finalmente instalamos docker

```
root@laptop:/home/beatriz# apt install docker-ce docker-compose docker
-ce-cli containerd.io
```

Ahora con nuestro docker ya instalado solo nos queda configurarlo para que use un mirror diferente al de dockerhub y así poder bajar las imágenes sin restricciones.

Crear en /etc/docker/ el fichero daemon.json

```
root@laptop:/home/beatriz# cd /etc/docker/
```

```
root@laptop:/etc/docker# cat> daemon.json
```

Agregar el siguiente contenido.

Reinciamos para que se vean los cambios

```
root@laptop:/etc/docker# systemctl daemon-reload
```

```
root@laptop:/home/beatriz# systemctl start docker
```

Este paso a continuación es para cuando se estas detras de un proxy como en la DTCF.

Crear la carpeta docker.service.d

```
root@laptop:/home/beatriz# mkdir -p /etc/systemd/system/docker.service
.d
```

Creamos el fichero http-proxy.conf

```
root@laptop:/home/beatriz# nano /etc/systemd/system/docker.service.d/h
ttp-proxy.conf
```

Dentro agregamos:

```
/etc/systemd/system/docker.service.d/http-proxy.conf

Environment="HTTP_PROXY=http://user:password@proxyip:port/"
Environment="HTTPS_PROXY=http://user:password@proxyip:port/"
Environment="NO_PROXY= hostname.example.com,localhost,127.0.0.1"
```

Recargamos la config y reinciamos el servicio:

```
root@laptop:/etc/docker# systemctl daemon-reload
```

```
root@laptop:/home/beatriz# systemctl start docker
```

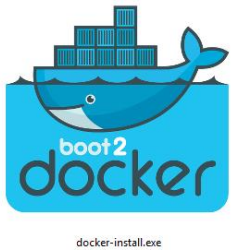
Ahora comprobaremos que todo está bien.

```
root@laptop:/etc/docker# docker info
```

Nos dará una salida similar a la siguiente:

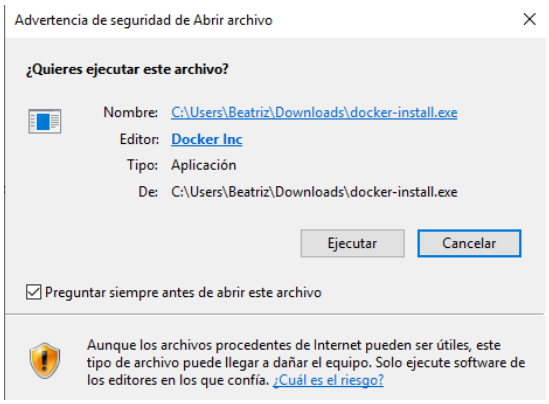
3.4 – Prueba 2: Instalación de la tecnología de contenedores Docker en sistema operativo Windows

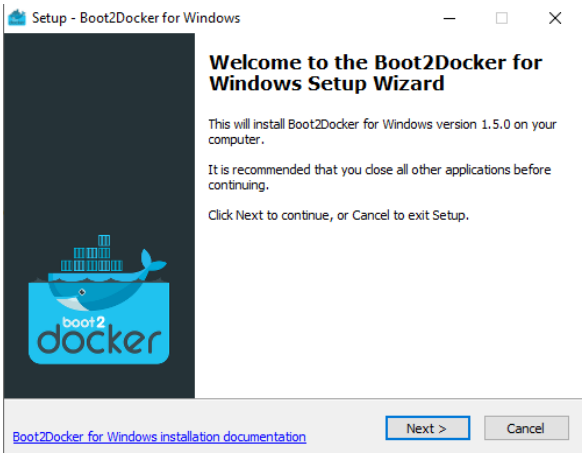
Para la instalación descargaremos el fichero de ejecución https://github.com/boot2docker/windows-installer/releases/tag/v1.5.0 o también podemos ir a la página Docker CE for Windows, y en ella encontraremos los instaladores para Windows 10.



docker-install.exe

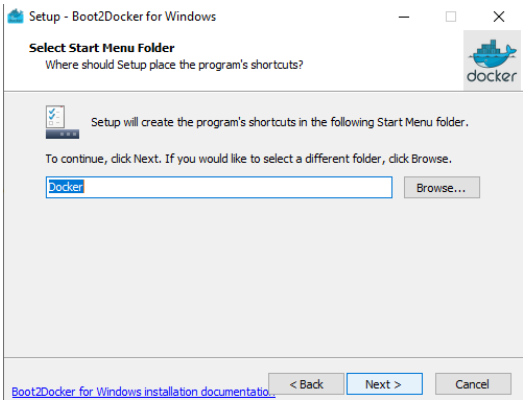
Simplemente se ejecuta el archivo que se ha descargado de la página web, abrirá una ventana con la instalación de Docker.





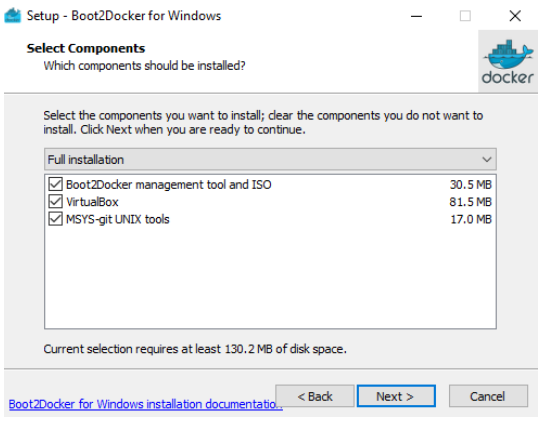
Nos preguntara el lugar donde instalaremos Docker.

Escoger el destino donde se desea instalar y dar click en siguiente (next) para continuar con la instalación.

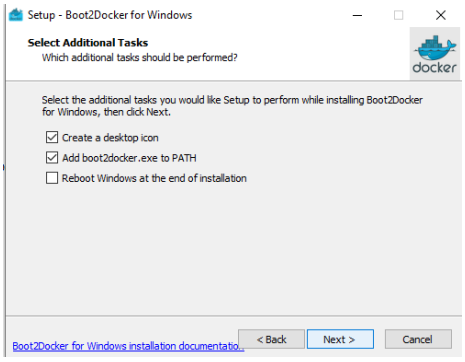


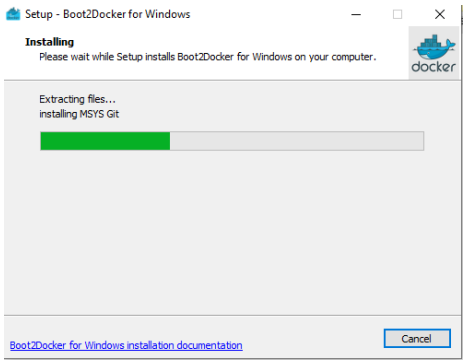
Lo siguiente que preguntará el programa será que queremos instalar, lo que hace Docker en Windows es, crear una máquina virtual a la cual le instala un sistema operativo Linux “Boot2Docker”, ya que Docker necesita utilizar el kernel de Linux, para esto utiliza el programa de virtualización VirtualBox, también instala la herramienta msys-git que utiliza como terminal para conectarse a la máquina virtual que creara.

Se señalan los programas que se necesitan instalar para su correcta ejecución para ello deben quedar activadas las opciones Boot2Docker, VirtualBox, MSYS-git y dar siguiente para continuar.



Luego para su ejecución creara un acceso directo en el escritorio.





Al ejecutar la aplicación, lo que hace es verificar si existe la máquina virtual en Docker si no existe, la crea, y si existe, se inicia la máquina virtual y se conecta mediante la creación de una clave de conexión con ssh.

3.5 – Prueba 3: Creación de un repositorio local con Docker

Corresponde el turno para crear nuestro docker registry local. Alternativas para disponer de un registry hay varias tales como: Nexus, Harbor etc. Implementaremos registry oficial de Hub Network por los pocos recursos que usa. La imagen que usaremos implementa un docker registry HTTP API v2. Para nuestro proyecto crearemos una carpeta llamada registry.

```
root@laptop:/home/beatriz# mkdir registry
root@laptop:/home/beatriz# cd registry
```

Creamos el fichero docker-compose.yml y dentro agregamos:

```
root@laptop:/home/beatriz/registry# touch docker-compose.yml
root@laptop:/home/beatriz/registry# nano docker-compose.yml
```

```
 GNU nano 2.9.3            docker-compose.yml

version: '3.5'

services:
     registry:
        image: docker.uclv.cu/registry:2.7.1
        container_name: registry
        networks:
            - hub_network
        restart: always
        ports:
            - '5000:5000'
        volumes:
            - ./registry-data:/var/lib/registry
            - ./config.yml:/etc/docker/registry/config.yml
        environment:
         REGISTRY_PROXY_REMOTEURL: "https://docker.uclv.cu"
         # Si navegas por proxy descomenta las líneas de abajo y pon $
         #HTTP_PROXY: ip:puerto
         #HTTPS_PROXY:  ip:puerto
         #NO_PROXY: localhost,127.0.0.1,10.0.0.7
         #no_proxy: localhost,127.0.0.1,10.0.0.7

networks:
  hub_network:
    driver: bridge
```

Nuestro docker compose hace referencia a config.yml el cual tendremos que crear.

```
root@laptop:/home/beatriz/registry# touch config.yml
```

```
 GNU nano 2.9.3                 config.yml                 Modificado

version: 0.1

log:
  fields:
    service: registry
storage:
  cache:
    blobdescriptor: inmemory
  filesystem:
    rootdirectory: /var/lib/registry
delete:
  enabled: true
http:
  addr: :5000
  headers:
    X-Content-Type-Options: [nosniff]
health:
  storagedriver:
    enabled: true
    interval: 10s
    threshold: 3
```

Levantamos el servicio:

```
root@laptop:/home/beatriz/registry# sudo docker-compose up -d
Creating network "registry_hub_network" with driver "bridge"
Pulling registry (docker.uclv.cu/registry:2.7.1)...
2.7.1: Pulling from registry
5a428f9f83b0: Pulling fs layer
90cad49de35d: Downloading [>
90cad49de35d: Downloading [=======>
90cad49de35d: Downloading [=============>
5a428f9f83b0: Downloading [>
        ]  32.77kB/2.817MB
90cad49de35d: Downloading [====================>
5a428f9f83b0: Downloading [=>
5a428f9f83b0: Downloading [=>
        ]   98.3kB/2.817MB
90cad49de35d: Downloading [========================================>
5a428f9f83b0: Downloading [==>
        ]  131.1kB/2.817MBplete
5a428f9f83b0: Pull complete
90cad49de35d: Pull complete
5215d0b40846: Pull complete
429305b6c15c: Pull complete
5f7e10a4e907: Pull complete
Digest: sha256:265d4a5ed8bf0df27d1107edb00b70e658ee9aa5acb3f37336c5a17
db634481e
Status: Downloaded newer image for docker.uclv.cu/registry:2.7.1
Creating registry ...
Creating registry ... done
root@laptop:/home/beatriz/registry#
```

Para usar tu nuevo servidor registry agrega esta líneas en daemon.json.

```
root@laptop:/home/beatriz/registry# nano /etc/docker/daemon.json
```

```
  GNU nano 2.9.3          /etc/docker/daemon.json

{
"registry-mirrors": [
    "https://rw21enj1.mirror.aliyuncs.com",
    "https://dockerhub.azk8s.cn",
    "https://reg-mirror.qiniu.com",
    "https://hub-mirror.c.163.com",
    "https://docker.mirrors.ustc.edu.cn",
    "https://1nj0zren.mirror.aliyuncs.com",
    "https://quay.io",
    "https://docker.mirrors.ustc.edu.cn",
    "http://f1361db2.m.daocloud.io",
    "https://registry.docker-cn.com",
    "http://registry:5000", "https://docker.uclv.cu"],
    ]
"insecure-registries":["http://registry:5000", "https://docker.uclv.c$
}
```

Reiniciamos el servicio.

```
root@laptop:/home/beatriz/registry# sudo service docker restart
```

Para que el host pueda resolver el nombre del registry que asignamos debemos cargarlo al fichero /etc/hosts.

3.6 – Prueba 4: Crear imagen con Docker

Para subir los privilegios y convertirse en súper usuario (root) ejecutamos el siguiente comando.

```
beatriz@laptop:~$ sudo su
[sudo] contraseña para beatriz:
```

Iniciamos el proceso de creación de la imagen

```
root@laptop:/home/beatriz#  debootstrap bionic bionic file:/home/beatr
iz/Repositorios/bionic
```

Con el código anterior creamos una imagen en base Ubuntu Bionic, con nombre bionic partiendo de un repositorio localhost, el que hace referencia a la ubicación donde se encuentre el repositorio o mirror de Ubuntu Bionic en nuestra propia máquina, este puede ser intercambiado por un repositorio nacional (repositorio.cu) en caso que no poseamos el mirror en nuestra propia máquina. La ejecución de este comando puede demorar unos 6 o 7 minutos.

Antes de proceder editamos el fichero /bionic/etc/apt/sources.list con el objetivo de que todo el proceso de instalación de paquetes en la imagen se haga desde los servidores locales o nacionales.

```
root@laptop:/home/beatriz# nano bionic/etc/apt/sources.list
```

En el archivo se coloca la dirección del repositorio

```
 GNU nano 2.9.3                                  bionic/etc/apt/sources.list

deb file:///home/beatriz/Repositorios/bionic bionic main universe multiverse restricted
```

Se empaqueta en un tar el directorio creado mediante debootstrap, luego se crea una imagen Docker limpia y se importa el contenido de la imagen bionic hacia

ella quedando la imagen personalizada, creada desde servidores locales o nacionales, sin la necesidad de utilizar servicios de Docker Hub.

```
root@laptop:/home/beatriz# tar -cf bionic.tar bionic
root@laptop:/home/beatriz# docker import bionic.tar ubuntu18_04:debootstrap
```

Esta imagen ya puede ser ejecutada desde Docker normalmente, así como reutilizable para cualquier configuración personalizada. La ejecución de la imagen la realizamos con el siguiente comando.

```
root@laptop:/home/beatriz# docker run bionic cat /etc/lsb-release
```

Luego para ver si fue creada procedemos a listar las imágenes con el comando.

```
root@laptop:/home/beatriz# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
ubuntu18_04         debootstrap         15f3127fdfcd        About a minute ago  296MB
<none>              <none>              1a95b8d01ef1        5 minutes ago       296MB
```

3.7 – Prueba 5: Reusar imagen

Si queremos que los cambios realizados anteriormente en la imagen sean permanentes, debemos crear una nueva imagen con el contenedor personalizado.

Vamos a realizar un ejemplo.

Tenemos la imagen base de Ubuntu.

```
root@laptop:/home/beatriz# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
ubuntu18_04         debootstrap         15f3127fdfcd        About a minute ago  296MB
<none>              <none>              1a95b8d01ef1        5 minutes ago       296MB
```

Lanzamos un contenedor con esa imagen base en el modo interactivo que vimos anteriormente.

```
root@laptop:/home/beatriz# docker run ubuntu18_04:debootstrap bin bash
```

Y listamos los contenedores y vemos señalado en rojo el que acaba de ser creado.

```
root@laptop:/home/beatriz# docker ps -a
CONTAINER ID        IMAGE                      COMMAND                  CREATED             STATUS
e5828fc14e9c        ubuntu18 04:debootstrap    "bin bash"               19 seconds ago      Created
rasekhar
1caa53ed7f71        ubuntu18_04:debootstrap    "bin bash"               8 hours ago         Created
ic
26f0ad938852        ubuntu18_04:debootstrap    "/bin/bash"              9 hours ago         Created
off
a25a49dd11dd        ubuntu18_04:debootstrap    "bin bash"               9 hours ago         Created
nyder
3ef6f4234b28        ubuntu18_04:debootstrap    "cat /etc/lsb-release"   9 hours ago         Created
ompson
98e5de816fef        ubuntu18_04:debootstrap    "cat /etc/lsb-release"   9 hours ago         Created
ht
```

Ahora vamos a guardar los cambios realizados en la imagen. Tenemos que salir del contenedor y ejecutar el comando “commit”.

Para poder utilizar esta imagen con los cambios, tenemos que crear una nueva imagen, con el comando:

```
root@laptop:/home/beatriz# docker commit -m "ubuntu bionic" -a "Beatriz Reyes" e5828fc14e9c ubuntu:deb
```

Y listamos las imágenes para ver si se creó correctamente:

```
root@laptop:/home/beatriz# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
ubuntu              deb                 8c6fe09ac814        13 seconds ago      296MB
ubuntu18_04         debootstrap         15f3127fdfcd        10 hours ago        296MB
<none>              <none>              1a95b8d01ef1        10 hours ago        296MB
```

3.8 – Prueba 6: Crear imagen a partir de un Dockerfile

En este epígrafe se lanzará un contenedor haciendo uso del siguiente Dockerfile, que se ha colocado en una carpeta llamada /wordpress.

Se crea el directorio wordpress.

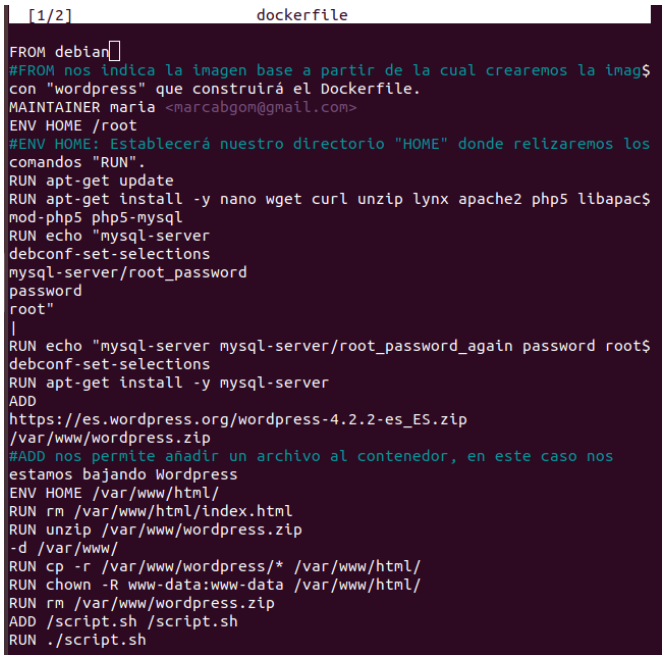
```
root@laptop:/home/beatriz/registry/docker_docs/documentacionimagenesdo
cker# mkdir wordpress
```

Dentro de la carpeta wordpress se crea el archivo Dockerfile.

```
root@laptop:/home/beatriz/registry/docker_docs/documentacionimagenesdo
cker/wordpress# touch dockerfile
```

Se edita el archivo Dockerfile para ponerle dentro las instrucciones.

```
root@laptop:/home/beatriz/registry/docker_docs/documentacionimagenesdo
cker/wordpress# nano etc dockerfile
```



```
[1/2]                    dockerfile

FROM debian
#FROM nos indica la imagen base a partir de la cual crearemos la imag$
con "wordpress" que construirá el Dockerfile.
MAINTAINER maria <marcabgom@gmail.com>
ENV HOME /root
#ENV HOME: Establecerá nuestro directorio "HOME" donde relizaremos los
comandos "RUN".
RUN apt-get update
RUN apt-get install -y nano wget curl unzip lynx apache2 php5 libapac$
mod-php5 php5-mysql
RUN echo "mysql-server
debconf-set-selections
mysql-server/root_password
password
root"
|
RUN echo "mysql-server mysql-server/root_password_again password root$
debconf-set-selections
RUN apt-get install -y mysql-server
ADD
https://es.wordpress.org/wordpress-4.2.2-es_ES.zip
/var/www/wordpress.zip
#ADD nos permite añadir un archivo al contenedor, en este caso nos
estamos bajando Wordpress
ENV HOME /var/www/html/
RUN rm /var/www/html/index.html
RUN unzip /var/www/wordpress.zip
-d /var/www/
RUN cp -r /var/www/wordpress/* /var/www/html/
RUN chown -R www-data:www-data /var/www/html/
RUN rm /var/www/wordpress.zip
ADD /script.sh /script.sh
RUN ./script.sh
```

```
[1/2]                    dockerfile

debconf-set-selections
RUN apt-get install -y mysql-server
ADD
https://es.wordpress.org/wordpress-4.2.2-es_ES.zip
/var/www/wordpress.zip
#ADD nos permite añadir un archivo al contenedor, en este caso nos
estamos bajando Wordpress
ENV HOME /var/www/html/
RUN rm /var/www/html/index.html
RUN unzip /var/www/wordpress.zip
-d /var/www/
RUN cp -r /var/www/wordpress/* /var/www/html/
RUN chown -R www-data:www-data /var/www/html/
RUN rm /var/www/wordpress.zip
ADD /script.sh /script.sh
RUN ./script.sh
#ADD nos añadirá en este caso la configuración de la base de datos que
una vez realizada el despliegue tendremos que poner para su utilizaci$
EXPOSE 80
#EXPOSE indica los puertos TCP/IP por los que se pueden acceder a los
servicios del contenedor 80 "HTTP".
CMD ["/bin/bash"]
#CMD nos establece el comando del proceso de inicio que se usará.
ENTRYPOINT ["./script.sh"]
```

3.9 – Prueba 7: Creación de un contenedor

Iniciaremos con la creación de un contenedor, el cual usa la imagen creada anteriormente.

1. Para subir los privilegios y convertirse en súper usuario (root) ejecutamos el siguiente comando.

```
beatriz@laptop:~$ sudo su
[sudo] contraseña para beatriz:
```

2. Actualizamos el sistema de paquetes.

```
root@laptop:/home/beatriz# apt update
```

Para la creación de un contenedor es imprescindible primeramente tener creada una imagen.

Podemos crear un contenedor con el comando run.

Sintaxis:

docker run [OPTIONS] IMAGE [COMMAND] [ARG....]

Entre las opciones se encuentran (lista completa aquí):

-a, --attach: para conectarnos a un contenedor que está corriendo.

-d, --detach: corre un contenedor en segundo plano.

-i, --interactive: habilita el modo interactivo.

--name: le pone nombre a un contenedor.

El comando run primero crea una capa del contenedor sobre la que se puede escribir y a continuación ejecuta el comando especificado. Con este comando hemos ejecutado un contenedor, sobre la imagen Ubuntu, que ha ejecutado el comando echo. Cuando ha terminado de ejecutar el comando que le hemos pedido se ha detenido. Los contenedores están diseñados para correr un único servicio, aunque podemos correr más si hiciera falta. Cuando ejecutamos un contenedor con run debemos especificarle un comando a ejecutar en él, y dicho contenedor solo se ejecuta durante el tiempo que dura el comando que especifiquemos, funciona como un proceso.

Algo a tener en cuenta es que, si la imagen que estamos poniendo en el comando run no la tuviéramos en local, Docker primero la descargaría y la guardaría en local y luego seguiría con la construcción capa por capa del contenedor.

3. Iniciamos el proceso de creación del contenedor.

```
root@laptop:/home/beatriz#  docker run -it ubuntu18_04: debootstrap
```

Con el código anterior se crea un contenedor interactivo.

-t: ejecuta una terminal.

-i: nos comunicamos con el contenedor en modo interactivo.

4. Luego para ver si fue creado procedemos a listar los contenedores que han sido creados con el comando.

```
root@laptop:/home/beatriz# docker ps -a
CONTAINER ID        IMAGE                      COMMAND                  CREATED             STATUS
e5828fc14e9c        ubuntu18 04:debootstrap    "bin bash"               19 seconds ago      Created
rasekhar
1caa53ed7f71        ubuntu18_04:debootstrap    "bin bash"               8 hours ago         Created
ic
26f0ad938852        ubuntu18_04:debootstrap    "/bin/bash"              9 hours ago         Created
off
a25a49dd11dd        ubuntu18_04:debootstrap    "bin bash"               9 hours ago         Created
nyder
3ef6f4234b28        ubuntu18_04:debootstrap    "cat /etc/lsb-release"   9 hours ago         Created
ompson
98e5de816fef        ubuntu18_04:debootstrap    "cat /etc/lsb-release"   9 hours ago         Created
ht
```

6. Docker genera automáticamente un nombre al azar por cada contenedor que creamos. Si queremos especificar un nombre en particular podemos hacerlo con el parámetro --name. Para crear un contenedor con su nombre ejecutamos el comando.

```
root@laptop:/home/beatriz# docker run --name Ubuntu_Bionic it ubuntu18_04: debootstrap
```

3.10 – Conclusiones

Luego de la realización de varias pruebas como fueron instalar la tecnología de contenedores docker en el sistema operativo Linux y Windows, y realizar pruebas usando la tecnología y aplicando conocimientos obtenidos en el análisis de este documento; teniendo como guía un procedimiento que permite seguir una serie de pasos lógicos que llevan a lograr el correcto funcionamiento de tecnología de contenedores Docker, se puede afirmar que este procedimiento está listo para su despliegue en la DTCF.

Conclusiones

Al llegar a esta etapa, según los objetivos definidos en el presente trabajo, se arriban a las siguientes conclusiones:

- De las características, arquitectura, componentes y utilización de la tecnología de contenedores, en particular Docker se concluye que:
    - Esta herramienta tiene múltiples prestaciones; desde crear imágenes y contenedores, hasta manejar redes y clústers basándonos en los mismos contenedores Docker.
    - Se puede trabajar la tecnología de contenedores docker con proyectos de código abierto como Kubernetes y Docker Machine facilitando así la explotación de todas las prestaciones de esta herramienta.
- Del análisis de las características de la virtualización y del esquema de virtualización utilizado en la DTCF se obtiene que usar docker brinda prestaciones que el actual sistema no tiene como son:
    - Desplegar más servidores por cada servidor físico del que se dispone, lo que propicia el ahorro de energía
    - Reutilizar las bibliotecas lo que proporciona mejor aprovechamiento del espacio
    - Permitir el uso de varias versiones de una misma aplicación facilitando el trabajo de los programadores.
- Docker reúne condiciones adecuadas para su implementación en la DTCF tales como:
    - Rapidez de ejecución
    - Ligereza
    - Portabilidad
    - Autosuficiencia

Recomendaciones

Continuar el estudio de la tecnología de contenedores Docker para:

- Continuar la prueba de la tecnología de contenedores Docker en la DTCF y llevarla a su optimo funcionamiento.

Referencias bibliográficas

[1] M. C. Christian Sfeir, «Máquinas Virtuales vs Contenedores, ¿Qué son y cómo elegir entre estas tecnologías?», FayerWayer, 29-jun-2016. [En línea]. Disponible en: https://www.fayerwayer.com/2016/06/maquinas-virtuales-vs-contenedores-que-son-y-como-elegir-entre-estas-tecnologias/.

[2] Josmell Chavarri, «Contenedores Docker Vs Máquinas Virtuales. – Guayoyo – Medium», Guayoyo, 19-sep-2017. [En línea]. Disponible en: https://medium.com/guayoyo/contenedores-docker-vs-m%C3%A1quinas-virtuales-2f434b4b5031.

[3] IES Gonzalo Nazareno Dos Hermanas (Sevilla) IES Los Albares Cieza (Murcia) IES La Campi˜na Arahal (Sevilla) IES Ingeniero de la Cierva Murcia, «Virtualización de Servidores Conceptos básicos». .

[4] Denis Morejón López, «VIRTUALIZACIÓN DE SERVIDORES UTILIZANDO PROXMOX VE». .

[5] «¿En qué se diferencia Docker de una máquina virtual?», *DOKRY*, 2017. [En línea]. Disponible en: https://www.dokry.com/2459.

[6] «DIFERENCIAS DE UNA MAQUINA VIRTUAL DE UNA REAL». .

[7] William Estuardo Salazar Hernández, «IMPLEMENTACIÓN DE ARQUITECTURA DE MICROSERVICIOS UTILIZANDO VIRTUALIZACIÓN POR SISTEMA OPERATIVO», TRABAJO DE GRADUACIÓN, Universidad de San Carlos de Guatemala, Guatemala, 2017.

[8] «Docker container las ventajas de los contenedores web - 1&1 IONOS», Docker y otros container: más allá de la virtualización, 21-jun-2019. [En línea]. Disponible en: https://www.ionos.es/digitalguide/servidores/know-how/docker-container-las-ventajas-de-los-contenedores-web/.

[9] Aprendizaje Docker. .

[10] María Cabrera Gómez de la Torre, «GESTIÓN DE CONTENEDORES DOCKER-KUBERNETES». .

[11] MILTON ANDRÉS SIMBAÑA ALARCÓN, «ESTUDIO DEL CONTENEDOR CLOUD DOCKER Y PROPUESTA DE IMPLEMENTACIÓN PARA LA PLATAFORMA CLOUD FICA», TRABAJO DE GRADO PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN SISTEMAS COMPUTACIONALES, UNIVERSIDAD TÉCNICA DEL NORTE, barra-Ecuador, 2016.

[12] Luis Ángel Sánchez Lasso, «Contenedor de aplicaciones: Docker». 23-jun-2015.

[13] Aprendizaje kubernetes. .

[14] jlatorre, «Montando un docker registry “Como Dios Manda”», Montando un Docker Registry Blog Irontec, 17-feb-2017. [En línea]. Disponible en: https://blog.irontec.com/montando-un-docker-registry-como-dios-manda/.

Bibliografía

By Young Kim, «Cómo configurar un registro de Docker privado en Ubuntu 18.04 DigitalOcean», Cómo configurar un registro de Docker privado en Ubuntu 18.04, 09-ene-2020. [En línea]. Disponible en: https://www.digitalocean.com/community/tutorials/how-to-set-up-a-private-docker-registry-on-ubuntu-18-04-es.

Yandex, «¿Como hacer motor de docker empezar otra vez?», Como hacer motor de docker empezar otra vez_ _ 16.04 _ EnMiMaquinaFunciona.com. [En línea]. Disponible en: https://www.enmimaquinafunciona.com/pregunta/138919/como-hacer-motor-de-docker-empezar-otra-vez.

Brian Hogan, «Cómo instalar y usar Docker en Ubuntu 18.04 DigitalOcean», Cómo instalar y usar Docker en Ubuntu 18.04, 26-abr-2019. [En línea]. Disponible en: https://www.digitalocean.com/community/tutorials/como-instalar-y-usar-docker-en-ubuntu-18-04-1-es.

Raul Unzue Pulido, «Crear servidores de docker con Docker-Machine en VMware vSphere - Blog Virtualizacion», Crear servidores de docker con Docker-Machine en VMware vSphere, 23-jul-2019. [En línea]. Disponible en: https://www.maquinasvirtuales.eu/crear-servidores-de-docker-con-docker-machine-en-vmware-vsphere/.

Ivonne Karina Farías Alejandro, «Definición de un ambiente de construcción de aplicaciones empresariales a través de Devops, Microservicios y Contenedores», Titulación de Ingeniero en Informática, Universidad Técnica Particular de Loja, Centro Universitario Salinas, 2017.

Ing. José Manuel De Paz Estrada, «DISEÑO E IMPLEMENTACIÓN DE UNA ARQUITECTURA ESCALABLE BASADA EN MICROSERVICIOS PARA UN SISTEMA DE GESTIÓN DE APRENDIZAJE CON CARACTERÍSTICAS DE RED SOCIAL», Maestría de Tecnologías de la Información y Comunicación, Universidad de San Carlos de Guatemala, Guatemala, 2017.

Alexander Rivas Alpizar, «Docker a lo cubano Sysadmins de Cuba», Docker a lo cubano, abril-2020. [En línea]. Disponible en: https://www.sysadminsdecuba.com/2020/04/docker-a-lo-cubano/.

Juan Carlos Rubio, «Docker básico».

Raul Unzue Pulido, «Docker comandos básicos - Blog VMware y Citrix», Docker comandos básicos, 21-oct-2018. [En línea]. Disponible en: https://www.maquinasvirtuales.eu/docker-comandos-basicos/.

Chabir Atrahouch Echarrouti, «Docker Compose, Machine y Swarm - Adictos al trabajo», Docker Compose, Machine y Swarm, diciembre-2015. [En línea]. Disponible en: https://www.adictosaltrabajo.com/2015/12/03/docker-compose-machine-y-swarm/.

Sébastien Goasguen, Docker Cookbook, November 2015. O'Reilly, 2015.

«docker Documentation». 10-nov-2018.

ETECSA-wiki, «Docker - ETECSA-wiki», Docker. [En línea]. Disponible en: https://wiki.etecsa.cu/index.php?title=Docker&oldid=3982.

I. M. Aidan Hobson Sayers, Docker in Practice. Shelter Island, NY: MANNING, 2016.

JOSÉ LUIS PACHECO LAJE, «ESTUDIO COMPARATIVO ENTRE UNA ARQUITECTURA CON MICROSERVICIOS Y CONTENEDORES DOCKERS Y UNA ARQUITECTURA TRADICIONAL (MONOLÍTICA) CON COMPROBACIÓN APLICATIVA», PROYECTO DE TITULACIÓN, UNIVERSIDAD DE GUAYAQUIL, GUAYAQUIL – ECUADOR, 2018.

Santiago Gimeno Martínez, «Evaluación de plataformas virtuales: Estudio comparativo», Tesis de máster, Universidad Politécnica de Valencia, Valencia, 2008.

William Estuardo Salazar Hernández, «IMPLEMENTACIÓN DE ARQUITECTURA DE MICROSERVICIOS UTILIZANDO VIRTUALIZACIÓN POR SISTEMA

OPERATIVO», TRABAJO DE GRADUACIÓN, Universidad de San Carlos de Guatemala, Guatemala, 2017.

B. C. Salvador González, «Introducción a docker», 03-ene-2016.

@javierprovecho, «Introducción a Docker - Parte 1».

«Introduction to Docker», nov-2013.

«Introduction to Docker». Docker Inc, 2014.

«Kubernetes», Orquestación de contenedores para producción. [En línea]. Disponible en: https://kubernetes.io/es/.

campusMVP, «Los beneficios de utilizar Docker y contenedores a la hora de programar campusMVP», Los beneficios de utilizar Docker y contenedores a la hora de programar, 09-oct-2018. [En línea]. Disponible en: https://www.campusmvp.es/recursos/post/los-beneficios-de-utilizar-docker-y-contenedores-a-la-hora-de-programar.aspx.

LIC. ENRIQUE CARBONELL MUELA, «Modelo para el desarrollo de la infraestructura guiado por pruebas empleando el paradigma de infraestructura como código», Trabajo para optar por el Título Académico Máster en Ciencia de la Computación, Universidad Central «Marta Abreu» de Las Villas, Santa Clara, 2018.

Javier Nogués García, «Orquestación de contenedores con Kubernetes», Universidad Carlos III Madrid, Leganés, 2018.

jgarzas, «Para los que empiezan crear y ejecutar una imagen propia en un contenedor Docker (2_2) - Javier Garzas», Para los que empiezan: crear y ejecutar una imagen propia en un contenedor Docker (2/2), 06-nov-2015. [En línea]. Disponible en: https://www.javiergarzas.com/2015/11/para-los-que-empiezan-crear-y-ejecutar-una-imagen-propia-en-un-contenedor-docker-12.html.

Rafael Rodriguez Gayoso, «Recetas Docker Documentation». 09-nov-2017.

B. Y. M. R. E. Bach. JOSEPH JOSAFAT RAMOS ARPASIBach. JOSEPH JOSAFAT RAMOS ARPASI, «Sistema de gestión documental para la Oficina de Cooperación

Nacional e Internacional de la Universidad Nacional del Altiplano Puno – 2017», PARA OPTAR EL TÍTULO PROFESIONAL DE: INGENIERO ESTADÍSTICO E INFORMÁTICO, UNIVERSIDAD NACIONAL DEL ALTIPLANO - PUNO, PUNO – PERÚ, 2017.

Pol Ponsico Martín, «Tecnología de Contenedores Docker», Tesis de Grado, Universidad Politècnica de Catalunya, Barcelona, 2017.

E. V. Julio Gómez, VIRTUALIZACION DE SERVIDORES DE TELEFONIA IP EN GNU/LINUX. .

Glosario de términos

Amazon Web Services(AWS): es una colección de servicios de computación en la nube (también llamados servicios web) que en conjunto forman una plataforma de computación en la nube, ofrecidas a través de Internet por Amazon.com.

BIOS: En informática, Basic Input Output System, en español "sistema básico de entrada y salida", también conocido como "System BIOS", "ROM BIOS" o "PC BIOS", es un estándar de facto que define la interfaz de firmware para computadoras IBM PC compatibles.

Cloud Computing: La computación en la nube, conocida también como servicios en la nube, informática en la nube, nube de cómputo o nube de conceptos, es un paradigma que permite ofrecer servicios de computación a través de una red, que usualmente es Internet.

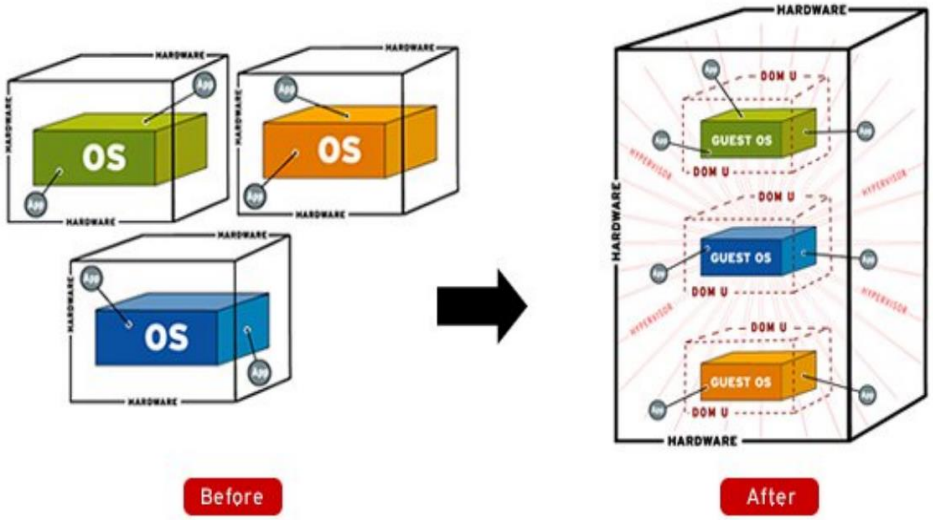
CPU: La unidad central de procesamiento o unidad de procesamiento central, es el hardware dentro de un ordenador u otros dispositivos programables, que interpreta las instrucciones de un programa informático mediante la realización de las operaciones básicas aritméticas, lógicas y de entrada/salida del sistema.

Hipervisor: Un hipervisor o monitor de máquina virtual es una plataforma que permite aplicar diversas técnicas de control de virtualización para utilizar, al mismo tiempo, diferentes sistemas operativos (sin modificar o modificados, en el caso de Paravirtualización) en una misma computadora. Es una extensión de un término anterior, «supervisor», que se aplicaba a los kernels de los sistemas operativos.

Microsoft Azure: Es un servicio en la nube ofrecida como servicio y alojado en los Data Centers de Microsoft. Anunciada en el Professional Developers Conference de Microsoft (PDC) del 2008 en su versión beta, pasó a ser un producto comercial el 1 de enero de 2010.

Anexos

Anexo 1 – Ejemplo de Virtualización



Anexo 2 – Representación de una máquina física y una máquina virtual

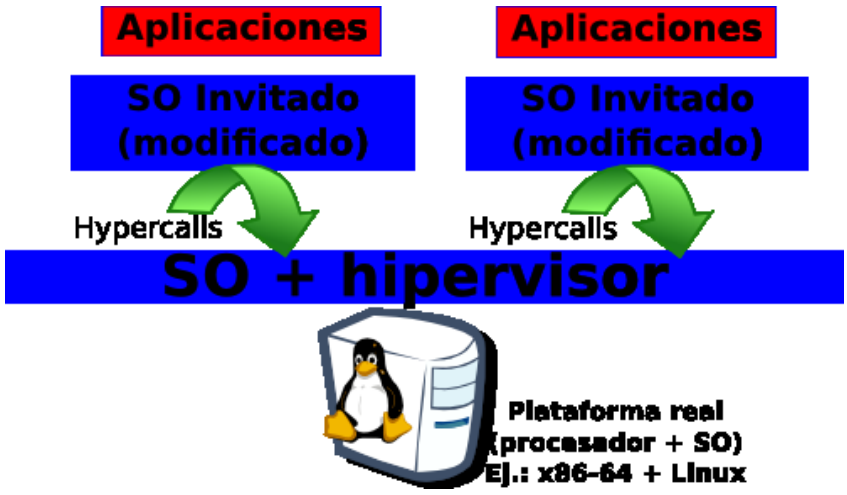
Máquina Física

Máquinas Virtuales

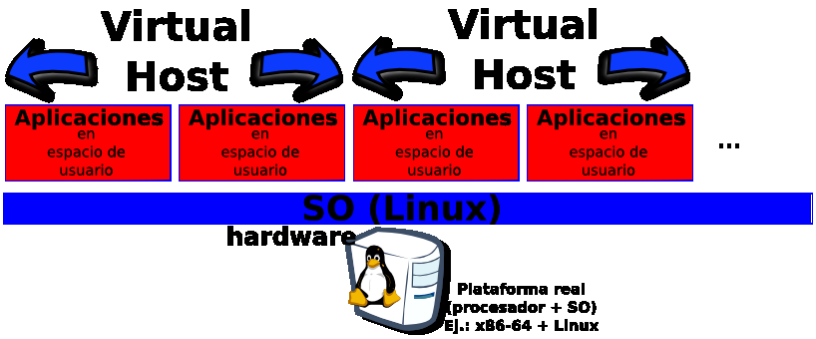
Anexo 3 – Arquitectura de la emulación

Anexo 4 – Arquitectura de la virtualización completa

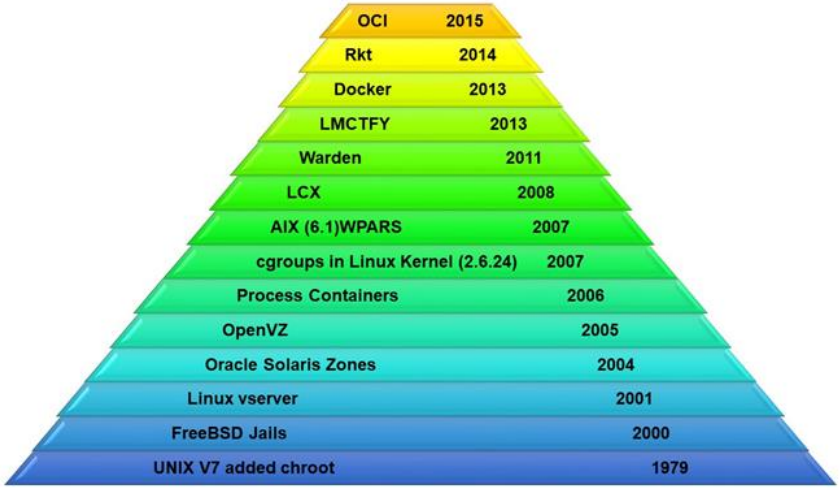
Anexo 5 – Arquitectura de la Paravirtualización



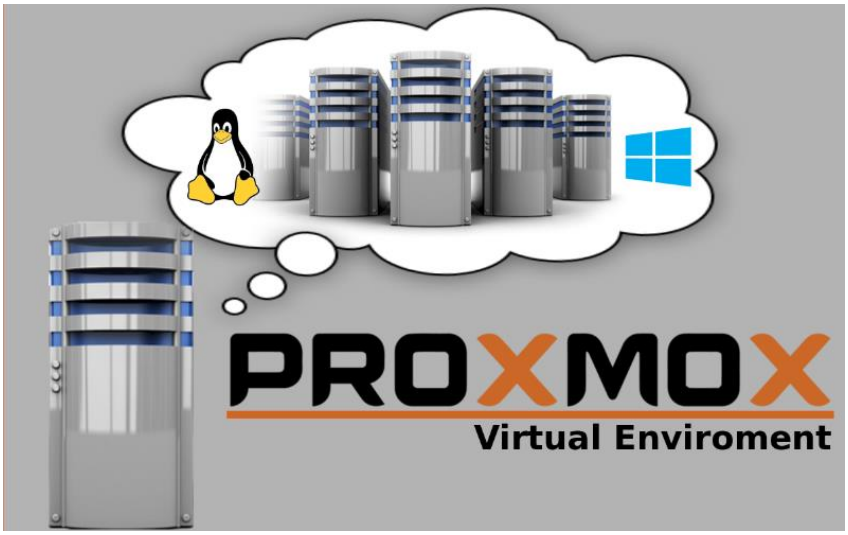
Anexo 6 – Arquitectura de la Virtualización a nivel de sistema operativo



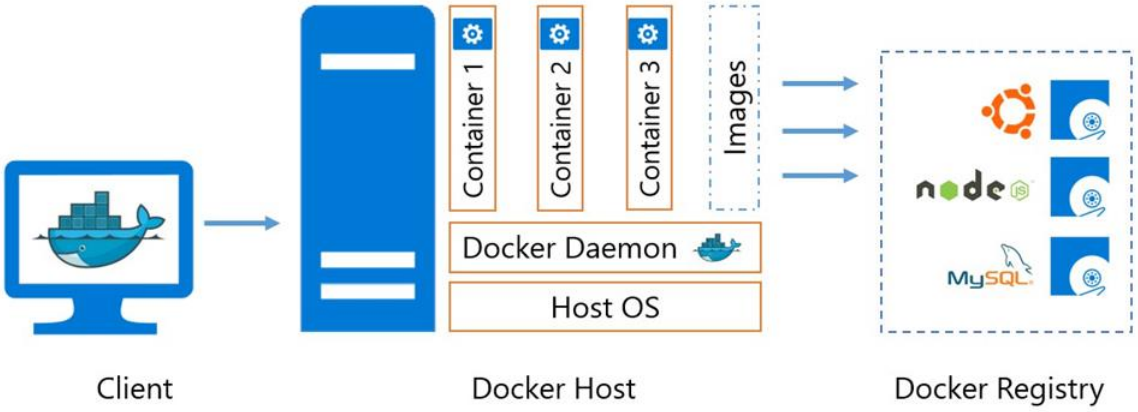
Anexo 7 – Evolución de las técnicas de partición basadas en un hipervisor



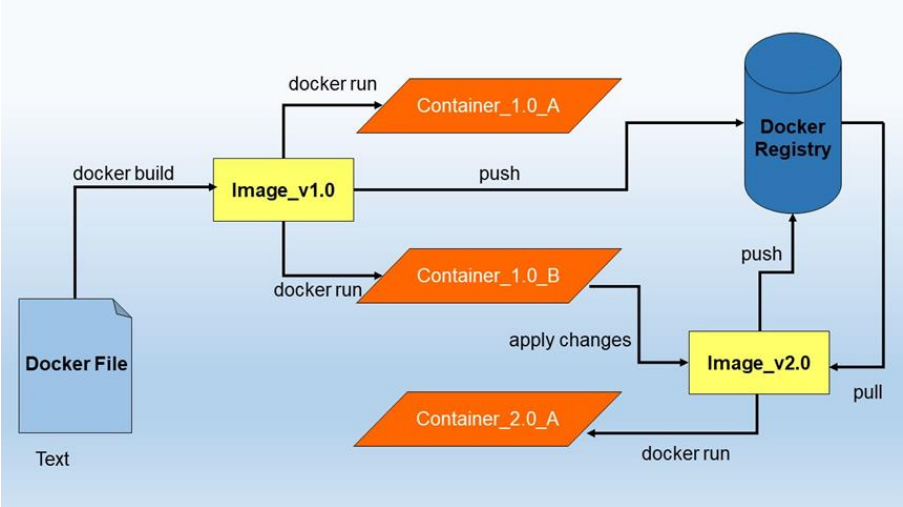
Anexo 8 – Representación del esquema de virtualización utilizado en la División Territorial de ETECSA en Cienfuegos



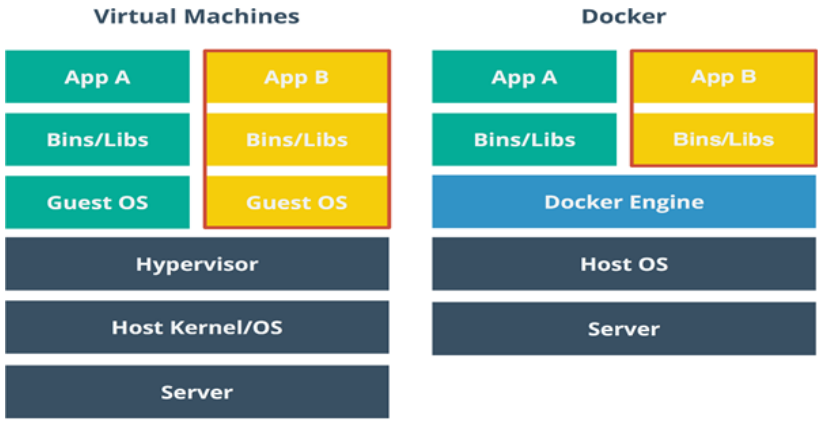
Anexo 9 – Representación gráfica de la Arquitectura de Docker



Anexo 10 – Esquema del funcionamiento de Docker y sus componentes



Anexo 11 – Comparación esquemática de Docker y una Máquina Virtual



Anexo 12 – Comparación entre Contenedores Docker, Máquinas Virtuales y Contenedores Linux

	Contenedores Docker	Máquinas Virtuales	Contenedores Linux
Objetivo	Abstraen las aplicaciones del sistema operativo.	Abstraen el hardware del SO.	Aislar procesos y recursos.
Rapidez	Puede ser creado y lanzado sólo en unos pocos segundos.	Puede tardar hasta varios minutos para crearse y poner en marcha.	Se pueden usar mucho más rápido, que los canales de desarrollo, que deben replicarlos entornos de prueba tradicionales.
Tiempo de Inicio y Detención	Menores a 50ms.	Inician en 30-45 segundos, y se detienen en 10 segundos o menos.	Se lanzan en 25 segundos.
Portabilidad	Todas las aplicaciones tienen sus propias dependencias, que incluyen tanto los recursos de software y hardware.	Se crean las máquinas virtuales con su kernel, su entorno, sus librerías, sus dependencias, etc.	Realiza el desacople de las aplicaciones del sistema operativo y es posible ejecutar cualquier contenedor a partir de la instalación de un entorno mínimo. Pero solo para máquinas con núcleo Linux.
Seguridad	Ofrece grupos de control de acceso al	La abstracción a nivel de hardware físico, limitan la	Utiliza una mezcla de elementos de seguridad que son el

	demonio que controla la virtualización.	superficie de ataque al hipervisor.	espacio de nombres, control de accesos mandatorios y grupos de control.
Sistema Operativo	No necesita que instalemos el sistema operativo completo.	Necesita un Sistema Operativo instalado.	Necesita un Sistema Operativo instalado.
Recursos	Son capaces de compartir un solo núcleo y compartir bibliotecas de aplicaciones.	Se desperdician recursos innecesariamente.	No encapsulan procesos sino que son sistemas enteros
Asignación de Recursos	Es Docker Engine, el encargado de asignar lo que sea necesario.	Debemos indicar de antemano cuántos recursos físicos le debemos dedicar.	Utilizan los recursos asignados al contenedor.
Almacenamiento	Poco menos de 1GB para algunas distribuciones de Linux con lo mínimo necesario y Hasta 10GB en el caso de un sistema Windows completo	Permite especificar el espacio en disco que se va a utilizar.	Hace uso de los recursos de almacenamiento que fueron asignados al contenedor.
Memoria RAM	Memoria RAM reservada, que puede ir desde 1 hasta varios GB, dependiendo de nuestras necesidades.	Aunque nuestra aplicación no haga uso en realidad de los GB de RAM reservados: no podrán ser utilizados por otras máquinas virtuales ni por nadie más.	Hace uso de los recursos que fueron asignados al contenedor.

Anexo 13 – Principales comandos de Docker

Comando	Descripción
attach	Para acceder a la consola de un contenedor que está corriendo.
build	Construye un contenedor a partir de un Dockerfile.
commit	Crea una nueva imagen de los cambios de un contenedor.
cp	Copia archivos o carpetas desde el sistema de ficheros de un contenedor al host.
create	Crea un nuevo contenedor.
daemon	Crea un proceso demonio.
diff	Comprueba cambios en el sistema de ficheros de un contenedor.
events	Muestra eventos en tiempo real del estado de un contenedor.
exec	Ejecuta un comando en un contenedor activo.
export	Exporta el contenido del sistema de ficheros de un contenedor a un archivo .tar.
history	Muestra el historial de una imagen.
images	Lista las imágenes que tenemos descargadas y disponibles
import	Crea una nueva imagen del sistema de archivos vacío e importa el contenido de un fichero .tar.
info	Muestra información sobre los contenedores, imágenes, versión de docker.
inspect	Muestra informaciones de bajo nivel del contenedor o la imagen.
kill	Detiene a un contenedor activo.
load	Carga una imagen desde un archivo .tar.
login	Para registrarse en un servidor de registro de Docker, por defecto.
logout	Se desconecta del servidor de registro de Docker.
logs	Obtiene los registros de un contenedor.
network connect	Conecta un contenedor a una red.
network create	Crea una nueva red con un nombre especificado por el usuario.

network disconnect	Desconecta un contenedor de una red.
network inspect	Muestra información detallada de una red.
network ls	Lista todas las redes creadas por el usuario.
network rm	Elimina una o más redes
pause	Pausa todos los procesos dentro de un contenedor.
port	Busca el puerto público, el cual está mateado, y lo hace privado.
ps	Lista los contenedores.
pull	Descarga una imagen o un repositorio del servidor de registros Docker.
push	Envía una imagen o un repositorio al servidor de registros de Docker.
rename	Renombra un contenedor existente.
restart	Reinicia un contenedor activo.
rm	Elimina uno o más contenedores.
rmi	Elimina una o más imágenes.
run	Ejecuta un comando en un nuevo contenedor.
save	Guarda una imagen en un archivo .tar
search	Busca una imagen en el índice de Docker.
start	Inicia un contenedor detenido.
stats	Muestra el uso de los recursos de los contenedores.
stop	Detiene un contenedor.
tag	Etiqueta una imagen en un repositorio.
top	Busca los procesos en ejecución de un contenedor.
unpause	Reanuda un contenedor pausado.
update	Actualiza la configuración de uno o más contenedores.
version	Muestra la versión de Docker instalada.
volume create	Crea un volumen.
volume inspect	Devuelve información de bajo nivel de un volumen.
volume ls	Lista los volúmenes.
volume rm	Elimina un volumen.
wait	Bloquea hasta detener un contenedor, entonces muestra su código de salida.

Anexo 14 – Opciones para el uso de los comandos de Docker

Opciones	Descripción
--config string	Ubicación de los archivos de configuración del cliente
-D, --debug	Habilita el modo de depuración
-H, --host list	Daemon socket (s) a los que conectarse
-l, --log-level string	Establece el nivel de registro
--tls	Utilice TLS; implícito por –tlsverify
--tlscacert string	Certificados de confianza firmados solo por esta CA
--tlscert string	Ruta al archivo de certificado TLS
--tlskey string	Ruta al archivo de claves TLS
--tlsverify	Use TLS y verifique el control remoto
-v, --version	Imprime la información de la versión y sale

Anexo 15 – Comandos de Gestión de Docker

Comando	Descripción
builder	Administrar compilaciones
config	Administrar las configuraciones de Docker
container	Gestionar contenedores
engine	Administra el motor de la ventana acoplable
image	Administrar imágenes
network	Administrar redes
node	Gestionar nodos de Swarm
plugin	Administrar complementos
secret	Gestionar secretos de Docker
service	Gestionar servicios
stack	Administrar pilas de Docker
swarm	Administrar enjambre
system	Administrar Docker
trust	Gestionar la confianza en las imágenes de Docker
volume	Gestionar volúmenes

Anexo 16 – Ejemplos de comandos para limitar los recursos de hardware de un contenedor

-m, --memory=""	Límite Memoria (formato: [], donde unidad= b, k, m or g)
--memory-swap=""	Total límite de memoria (memory + swap, formato: [], donde unidad = b, k, m or g)
--memory-reservation=""	Límite flexible de memoria (formato: [], donde unidad= b, k, m or g)
--kernel-memory=""	Límite memoria Kernel (formato: [], donde unidad= b, k, m or g)
-c, --cpu-shares=0	CPU (peso relativo)
--cpu-period=0	Limitar Período CPU CFS (Completely Fair Scheduler)
--cpuset-cpus=""	CPUs en donde permitir ejecución (0-3, 0,1)
--cpuset-mems=""	Nodos de memoria en donde permitir ejecución (0-3, 0,1)
--cpu-quota=0	Limitar cuota CPU CFS (Completely Fair Scheduler)
--blkio-weight=0	Bloquear Peso IO (Peso relativo) aceptar valor de peso entre 10 y 1000.
--oom-kill-disable=false	Desahabilitar OOM Killer para el contenedor
--memory-swappiness=""	Configurar el comportamiento d Swappines del contenedor

Anexo 17 – Ejemplos de comandos para limitar el espacio en memoria de un contenedor

memory=inf, memory-swap=inf (default)	No hay límites de memoria para el contenedor
memory=L<inf, memory-swap=inf	(Especificar memoria y configurar memory-swap como -1) El contenedor no puede usar más de la memoria especificada por L, pero puede usar toda la memoria swap que necesite
memory=L<inf, memory-swap=2*L	(Especificar memoria sin memory-swap) El contenedor no puede usar mas de la memoria especificada por L, y usar el doble como swap
memory=L<inf, memory-swap=S<inf, L<=S	(Especificar memoria y memory-swap) El contenedor no puede usar más de la memoria especificada por L, y la memoria Swap especificada por S

Anexo 18 – Principales comandos para el uso del Dockerfile

Comando	Descripción	Sintaxis
FROM	Indicamos una imagen base para construir el contenedor y opcionalmente un tag (si no la indicamos docker asumirá “latest” por defecto, es decir buscará la última versión). Lo que hace docker al leer esto es buscar en la máquina local una imagen que se llama, si no la encuentra la descargará de los repositorios.	FROM <imagen> FROM <imagen>:<tag>
MAINTAINER	Es información del creador y mantenedor de la imagen, usuario, correo, etc.	MAINTAINER <nombre> <correo> <cualquier_info>
RUN	Ejecuta directamente comandos dentro del contenedor y luego aplica/persiste los cambios creando una nueva capa encima de la anterior con los cambios producidos de la ejecución del comando y se hace un commit de los resultados. Posteriormente sigue con la siguiente instrucción.	RUN <comando> → modo shell, /bin/sh -c RUN [“ejecutable”,” parámetro1”,” parámetro2”] → modo ejecución, que permite correr comandos en imágenes base que no tengan /bin/sh o hacer uso de otra shell.
ENV	Establece variables de entorno del contenedor. Dichas variables se pasarán a todas las instrucciones RUN que se ejecuten posteriores a la declaración de las variables. Podemos especificar varias	ENV <key> <value> <key> <value> ENV <key>=<value> <key>=<value> Si queremos sustituir una variable, aunque esté definida en el Dockerfile, al ejecutar un contenedor podemos

	variables de entorno en una sola instrucción ENV.	especificarla y tomará dicho valor, tenga el que tenga en el Dockerfile. docker run -env <key>=<valor> También podemos pasar variables de entorno con el comando “docker run” utilizando el parámetro -e, para especificar que dichas variables sólo se utilizarán en tiempo de ejecución. Podemos usar estas variables en otras instrucciones llamándolas con $nombre_var o $ {nombre_var}. Por ejemplo: ENV DESTINO_DIR /opt/app WORKDIR $DESTINO_DIR.
ADD	Esta instrucción copia los archivos o directorios de una ubicación especificada en <fuente> y los agrega al sistema de archivos del contenedor en la ruta especificada en <destino>. En fuente podemos poner una URL, docker se encargará de descargar el archivo y copiarlo al destino. Si el archivo origen está comprimido, lo descomprime en el destino cómo si usáramos “tar -x”.	ADD <fuente>..<destino> ADD [“fuente”, ...” destino”] ¿El parámetro <fuente> acepta caracteres comodín tipo ?,*, etc. Si el destino no existe, Docker creará la ruta completa incluyendo cualquier subdirectorio. Los nuevos archivos y directorios se crearán con los permisos 0755 y un UID y GID de 0. También tenemos que tener en cuenta que, si los archivos o directorios agregados por una instrucción ADD cambian, entonces se invalida la caché para las siguientes instrucciones del Dockerfile.

COPY	Es igual que ADD, sólo que NO admite URLs remotas y archivos comprimidos como lo hace ADD.	
WORKDIR	Permite especificar en qué directorio se va a ejecutar una instrucción RUN, CMD o ENTRYPOINT. Puede ser usada varias veces dentro de un Dockerfile. Si se da una ruta relativa, esta será la ruta relativa de la instrucción WORKDIR anterior.	Podemos usar variables de entorno previamente configuradas, por ejemplo: ENV rutadir /ruta WORKDIR $rutadir
USER	Sirve para configurar el nombre de usuario a usar cuando se lanza un contenedor y para la ejecución de cualquier instrucción RUN, CMD o ENTRYPOINT posteriores.	
VOLUME	Crea un punto de montaje con un nombre especificado que permite compartir dicho punto de montaje con otros contenedores o con la máquina anfitriona. Es un directorio dentro de uno o más contenedores que no utiliza el sistema de archivos del contenedor, aunque se integra en el mismo para proporcionar varias funcionalidades útiles para que los datos sean persistentes y se puedan compartir con facilidad. Esto se hace para que cuando usemos el contenedor podamos tener acceso externo a un determinado directorio del contenedor.	Esto permite añadir datos, BBDD o cualquier otro contenido sin comprometer la imagen. El valor puede ser pasado en formato JSON o como un argumento, y se pueden especificar varios volúmenes. VOLUME [“/var/tmp”] VOLUME /var/tmp

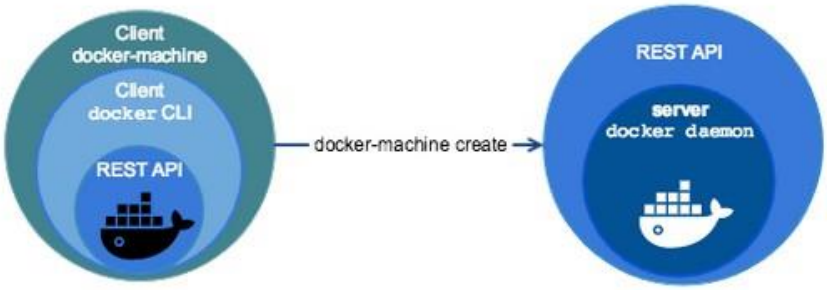
	Las características de estos volúmenes son: • Los volúmenes pueden ser compartidos y reutilizados entre los contenedores. • Un contenedor no tiene que estar en ejecución para compartir sus volúmenes. • Los cambios en un volumen se hacen directamente. • Los cambios en un volumen no se incluirán al actualizar una imagen. • Los volúmenes persisten incluso cuando dejan de usarlos los contenedores.	
LABEL	Añade metadatos a una imagen Docker. Se escribe en el formato etiqueta=" valor". Se pueden añadir varios metadatos separados por un espacio en blanco.	LABEL version="1.0" LABEL localizacion=" Barbate" tipo=" BBDD" Podemos inspeccionar las etiquetas en una imagen usando el comando docker inspect. $ docker inspect <nombre_imagen>/<tag>
STOPSIGNAL	Le indica al sistema una señal que será enviada al contenedor para salir. Puede ser un número válido permitido por el Kernel (por ejemplo 9) o un nombre de señal en el formato SIGNAME (por ejemplo SIGKILL).	

ARG	Define una variable que podemos pasar cuando estemos construyendo la imagen con el commando docker build, usando el flag --build-arg <varname>=<value>. Si especificamos un argumento en la construcción que no está definido en el Dockerfile, nos dará un error. El autor del Dockerfile puede definir una o más variables. Y también puede definir un valor por defecto para una variable, que se usará si en la construcción no se especifica otro.	ARG user1 ARG user1=someuser ARG user2 Se deben usar estas variables de la siguiente forma: docker build –build-arg <variable>=<valor> …. Docker tiene un conjunto de variables predefinidas que pueden usarse en la construcción de la imagen sin que estén declaradas en el Dockerfile: HTTP_PROXY http_proxy HTTPS_PROXY https_proxy FTP_PROXY ftp_proxy NO_PROXY no_proxy
ONBUILD	Añade triggers a las imágenes. Un disparador se utiliza cuando se usa una imagen como base de otra imagen. El disparador inserta una nueva instrucción en el proceso de construcción como si se especificara inmediatamente después de la instrucción FROM. Por ejemplo, tenemos un Dockerfile con la instrucción ONBUILD, y creamos una imagen a partir de este Dockerfile, por ejemplo, IMAGEN_padre. Si escribimos un nuevo Dockerfile, y la sentencia FROM apunta a IMAGEN_PADRE, cuando	Un ejemplo de uso: FROM Ubuntu:14.04 MAINTAINER mcgomez RUN apt-get update && apt-get install -y apache2 ONBUILD ADD ./var/www/ EXPOSE 80 CMD [“D”,”FOREGROUND”]

	construyamos una imagen a partir de este Dockerfile, IMAGEN_HIJO, veremos en la creación que se ejecuta el ONBUILD que teníamos en el primer Dockerfile. Los disparadores ONBUILD se ejecutan en el orden especificado en la imagen padre y sólo se heredan una vez, si construimos otra imagen a partir de la IMAGEN_HIJO, los disparadores no serán ejecutados en la construcción de la IMAGEN_NIETO. Hay algunas instrucciones que no se pueden utilizar en ONBUILD, como son FROM, MAINTAINER y ONBUILD, para evitar recursividad.	
EXPOSE	Se utiliza para asociar puertos, permitiéndonos exponer un contenedor al exterior (internet, host, etc.). Esta instrucción le específica a Docker que el contenedor escucha en los puertos especificados. Pero EXPOSE no hace que los puertos puedan ser accedidos desde el host, para esto debemos mapear los puertos usando la opción -p en docker run.	Por ejemplo: EXPOSE 80 443 docker run centos:centos7 -p 8080:80
CMD	Esta instrucción es similar al comando RUN, pero con la diferencia de que se ejecuta cuando instanciamos o	

	arrancamos el contenedor, no en la construcción de la imagen. Sólo puede existir una única instrucción CMD por cada Dockerfile y puede ser útil para ejecutar servicios que ya estén instalados o para correr archivos ejecutables especificando su ubicación.	
ENTRYPOINT	Cualquier argumento que pasemos en la línea de comandos mediante docker run serán anexados después de todos los elementos especificados mediante la instrucción ENTRYPOINT, y anulará cualquier elemento especificado con CMD. Esto permite pasar cualquier argumento al punto de entrada.	ENTRYPOINT [“ejecutable”,” parámetro1”,” parámetro2”] → forma de ejecución ENTRYPOINT comando parámetro1 parámetro 2 → forma shell ENTRYPOINT nos permite indicar qué comando se va a ejecutar al iniciar el contenedor, pero en este caso el usuario no puede indicar otro comando al iniciar el contenedor. Si usamos esta opción es porque no esperamos que el usuario ejecute otro comando que el especificado.
ARCHIVO DOCKERIGNORE	Es recomendable colocar cada Dockerfile en un directorio limpio, en el que sólo agreguemos los ficheros que sean necesarios. Pero es posible que tengamos algún archivo en dicho directorio, que cumpla alguna función pero que no queremos que sea agregado a la imagen. Para esto usamos. dockerignore, para que docker build excluya esos archivos durante la creación de la imagen.	Un ejemplo de. dockerignore */prueba* */*/prueba prueba?

Anexo 19 – Representación del funcionamiento de una Docker Machine



Anexo 20 – Comandos para el uso de las Docker Machine

Descripción	Comandos
Crear una máquina	$ docker-machine create --driver <nombre_driver> <nombre_máquina> $ docker-machine create -d <nombre_driver> <nombre_máquina>
Iniciar una máquina	$ docker-machine start <nombre_máquina>
Listar las máquinas y ver su estado.	$ docker-machine ls
Detener una máquina	$ docker-machine stop <nombre_máquina>
Obtener la ir de una máquina	$ docker-machine ir <nombre_máquina>
Actualizar la máquina con la última versión de Docker.	$ docker-machine upgrade <nombre_máquina>
Inspeccionar una máquina.	Nos muestra información sobra la máquina en formato JSON, por defecto (podemos especificarlo con la opción --format (-f). $ docker-machine inspect [options] <nombre_máquina>

Anexo 21 – Comandos para el uso de Kubernetes

Información a solicitar	Comando a ejecutar
¿Qué despliegues existen?	kubectl get deployments
¿Cuántos pods hay?	kubectl get pods
¿Qué servicios hay?	kubectl get services
¿Qué nodos están activos?	kubectl get nodes