Contenido

[1. Autores del trabajo, planificación y entrega 2](#_Toc445388848)

[1.1 Autores 2](#_Toc445388849)

[1.2 Planificación 2](#_Toc445388850)

[1.3 Entrega 2](#_Toc445388851)

[2. Descripción de las tecnologías 3](#_Toc445388852)

[2.1 Descripción de la tecnología 1 3](#_Toc445388853)

[2.2 Descripción de la tecnología 2 3](#_Toc445388854)

[3. Criterios de comparación 10](#_Toc445388855)

[3.1 Categoría A: Nombre 10](#_Toc445388856)

[3.1.1 Criterio A.1: Nombre 10](#_Toc445388857)

[3.1.2 Criterio A.2: Nombre 11](#_Toc445388858)

[3.1.n Criterio A.n: Nombre 11](#_Toc445388859)

[3.2 Categoría B: Nombre 11](#_Toc445388860)

[3.2.1 Criterio B.1: Nombre 11](#_Toc445388861)

[3.2.2 Criterio B.2: Nombre 11](#_Toc445388862)

[3.2.n Criterio B.n: Nombre 11](#_Toc445388863)

[3.3 Categoría Z: Nombre 11](#_Toc445388864)

[3.3.1 Criterio Z.1: Nombre 11](#_Toc445388865)

[3.3.2 Criterio Z.2: Nombre 11](#_Toc445388866)

[3.3.n Criterio Z.n: Nombre 11](#_Toc445388867)

[4. Evaluación de los criterios por tecnología 12](#_Toc445388868)

[4.1 Evaluación de los criterios para la tecnología 1 12](#_Toc445388869)

[4.2 Evaluación de los criterios para la tecnología 2 12](#_Toc445388870)

[5. Comparación de las tecnologías 13](#_Toc445388871)

[6. Recomendaciones 15](#_Toc445388872)

[6.1 Situación 1 15](#_Toc445388873)

[6.1.1 Descripción de la situación 15](#_Toc445388874)

[6.1.2 Recomendación de tecnología a utilizar 15](#_Toc445388875)

[6.2 Situación 2 15](#_Toc445388876)

[6.2.1 Descripción de la situación 15](#_Toc445388877)

[6.2.2 Recomendación de tecnología a utilizar 15](#_Toc445388878)

# 1. Autores del trabajo, planificación y entrega

## 1.1 Autores

Grupo M4

* Diego Cárdenas Cuadrado (Coordinador)
* Adrián Blanco Domínguez
* Alejandro Martínez Pantín
* Marcos Rodríguez Castillo
* Roberto Sánchez Leal

## 1.2 Planificación

[Enlace planificación GanttPro](https://app.ganttpro.com/shared/token/15957a85a42133e47210c13c34c9917d2e19765437b506b44fd1cbedd3b3b25a)

## 

## 1.3 Entrega

[Enlace repositorio GitHub](https://github.com/DiegoUAH/TG2)

# 2. Descripción de las tecnologías

## 2.1 OpenVZ

Es una tecnología de virtualización a nivel de sistema operativo para Linux. Permite que un servidor físico ejecute múltiples instancias de sistemas operativos aislados, conocidos como Servidores Privados Virtuales (VPS), Entornos Virtuales (EV) o Contenedores.

Mientras que las tecnologías de virtualización como VMware y Xen proporcionan una virtualización completa y pueden ejecutar múltiples sistemas operativos y versiones diferentes de kernel, en OpenVZ tanto el servidor anfitrión como los VPS deben ser Linux, aunque las distribuciones de GNU/Linux pueden ser diferentes en cada Contenedor. Sin embargo, la virtualización a nivel de sistema operativo de OpenVZ proporciona mejor rendimiento, escalabilidad, densidad, administración de recursos dinámicos, y facilidad de administración que las máquinas virtuales o que otras tecnologías de virtualización de servidores.

OpenVZ es la base de Virtuozzo (para Linux) que es un software comercial desarrollado por SWsoft Inc. OpenVZ es un producto de software libre y licenciado bajo los términos de la licencia GNU GPL versión 2.

OpenVZ consiste del **núcleo** y de **herramientas a nivel de usuario**.

**Núcleo:**

El núcleo de OpenVZ es un núcleo Linux modificado que agrega soporte para contenedores. El núcleo modificado proporciona virtualización, aislamiento, administración de recursos y Puntos de comprobación.

**Virtualización y aislamiento:**

Cada contenedor es una entidad separada, y se comporta en gran medida como lo haría un servidor físico real. De manera que tiene sus propios archivos, usuarios y grupos, árbol de procesos, red, dispositivos y objetos de comunicación entre procesos.

**Administración de recursos:**

Como todos los contenedores usan el mismo kernel, la administración de recursos es de suprema importancia. Con la administración de recursos de OpenVZ se consigue que cada contenedor permanezca dentro de sus límites y no afecte al resto.

La administración de recursos de OpenVZ se compone de cuatro componentes: cuota de disco de dos niveles, planificador de CPU justo, planificador de E/S de disco y monitor de usuarios. Estos recursos pueden cambiarse durante el tiempo de ejecución de un contenedor sin necesidad de reiniciar el sistema.

*Cuota de disco de dos niveles:*

El dueño (root) del sistema anfitrión (OpenVZ) puede configurar la cuota de disco por contenedor, en términos de bloques de disco e inodos (aproximadamente igual al número de archivos). Éste es el primer nivel de cuota de disco. También, un dueño de contenedor (root) puede usar las herramientas usuales de cuota dentro de su propio contenedor para definir cuotas de disco estándar de UNIX por usuario y por grupo.

Si se desea ampliar el espacio a un contenedor, solamente hay que incrementar la cuota de dsico. No es necesario redimensionar las particiones de disco, etc.

*Planificador de CPU:*

El planificador de CPU en OpenVZ es una implementación de dos niveles de la estrategia de programación de fair-share (cuota justa).

En el primer nivel, el planificador decide a qué contenedor debe asignar la división de tiempo de la CPU, en función de los valores de las cpuunits por contenedor. En el segundo nivel, el planificador de Linux estándar decide qué proceso ejecutar en ese contenedor, utilizando las prioridades de proceso estándar de Linux.

Es posible establecer diferentes valores para las CPUs en cada contenedor. El tiempo real de la CPU se distribuirá proporcionalmente a estos valores.

Además de lo anterior, OpenVZ proporciona formas de:

* Establecer límites estrictos de la CPU, como el 10% del tiempo total de la CPU.
* Limitar el número de núcleos de CPU disponibles para el contenedor.
* Enlazar un contenedor a un conjunto específico de CPUs.

*Planificador de E/S:*

Al igual que el planificador de CPU, el planificador de E/S en OpenVZ también tiene dos niveles, utilizando el programador de E/S CFA de Jens Axboe en su segundo nivel.

A cada contenedor se le asigna una prioridad de E/S y el planificador distribuye el ancho de banda de E/S disponible de acuerdo con las prioridades asignadas. Por lo tanto, ningún contenedor único puede saturar un canal de E/S.

*Monitor de usuarios:*

El monitor de usuarios es un grupo de contadores por Contenedor, límites, y garantías. Hay un conjunto de alrededor de 20 parámetros que se eligen cuidadosamente para cubrir todos los aspectos de la operación de Contenedor, de manera que ningún Contenedor por sí solo pueda abusar de cualquier recurso el cual es limitado por todo el nodo y así hacer daño a otros EVs.

Los recursos contabilizados y controlados son principalmente memoria y objetos en el kernel tales como los segmentos de memoria compartidos IPC, buffers de red etc. Cada recurso tiene cinco valores asociados con este: uso actual, uso máximo (por el tiempo de vida de un EV), contador de barrera, de límite y de falla. El significado de barrera y límite es dependiente del parámetro. Si cualquier recurso alcanza el contador de límite o de falla, entonces el dueño del EV puede ver si algo malo está pasando.

**Punto de comprobación y migración:**

La característica de la migración en vivo y punto de comprobación se liberó para OpenVZ a mediados de abril de 2006. Esta permite migrar un EV desde un servidor físico a otro sin necesidad de apagar/reiniciar un EV. El proceso se conoce como punto de comprobación: un EV se congela y todo su estado se guarda en un archivo en disco. Este archivo puede ser transferido a otra máquina y un EV puede descongelarse (restaurarse) allí. La demora es de unos pocos segundos.

Dado que cada pieza de estado de EV (incluyendo conexiones de red abiertas) se guarda, desde la perspectiva del usuario parece una demora en la respuesta: una transacción de base de datos toma más tiempo que el usual, cuando continúa como normal el usuario no nota que su base de datos está ya corriendo en otra máquina. Esta característica hace posible escenarios tales como actualizar un servidor sin necesidad de reiniciarlo: si la base de datos necesita más memoria o recursos de CPU, sencillamente se debe comprar un servidor mejor y más nuevo y migrar en vivo el EV a éste, migrar todos los EVs a otro, apagarlo, agregar memoria, arrancarlo de nuevo y migrar todos los EVs de nuevo.

**Herramientas en el nivel del usuario:**

OpenVZ viene con herramientas de línea de comandos para administrar Contenedores (vzctl), además de herramientas para administrar software en Contenedores (vzpkg).

**vzctl:** Esta es una simple herramienta de línea de comandos de alto nivel para administrar un Contenedor

Las plantillas son imágenes precreadas que se usan para crear un nuevo EV. Estas plantillas permiten crear contenedores en pocos segundos.

**Características distintivas:**

**Escalabilidad:**

Ya que OpenVZ emplea un modelo de kernel único, es tan escalable como kernel Linux 2.6, lo que significa que soporta hasta 64 CPUs y hasta 64 GiB de RAM. Un único contenedor se puede escalar hasta el equipo físico entero, usando todos los CPUs y toda la RAM.

De hecho, algunas personas están usando OpenVZ con único Entorno Virtual. Esto es extraño a primera vista, pero dado el hecho de que un EV único usa todo los recursos del hardware con rendimiento nativo, y tiene agregados beneficios tales como independencia del hardware, administración de recursos y migración en vivo, esto es una opción obvia en muchos escenarios.

**Densidad:**

OpenVZ es capaz de alojar cientos de Entornos Virtuales en hardware decente. Las principales limitaciones son RAM y CPU.

**Administración masiva:**

Un propietario (root) de un servidor físico OpenVZ (también conocido como Nodo de Hardware) puede ver todos los procesos y archivos de EV. Esto hace la administración masiva de escenarios posible. Considerar que VMware o Xen se usan para consolidación de servidores: para aplicar una actualización de seguridad a unos 10 servidores virtuales se debe iniciar una sesión en cada uno y ejecutar el procedimiento de actualización - el mismo que se haría con diez servidores físicos.

En el caso de OpenVZ, se puede ejecutar un simple script de intérprete de comandos que actualice todo (o sólo algunos seleccionados) EVs a la vez.

**Escenarios de uso:**

Los siguientes escenarios de uso son comunes para todas las tecnologías de virtualización. Sin embargo, una característica única de la virtualización en el nivel de SO como OpenVZ es que no se tiene que gastar gran cantidad de tiempo de procesamiento por la virtualización, esto hace los escenarios más atractivos.

**Seguridad:**

Se pone cada servicio de red (como Apache, Servidor de Correo, Servidor DNS etc.) en un Entorno Virtual separado. En caso de que un cracker encuentre un agujero en la seguridad de uno de las aplicaciones y entre, todo lo que puede hacer es en ese mismo servicio; dado que todos los otros servicios están en EVs separados no puede acceder a ellos.

**Consolidación de Servidores:**

Actualmente, la mayoría de los servidores están infra-utilizados. Usando OpenVZ, tales servidores se pueden consolidar migrándolos a Entornos Virtuales. Se ahorra en espacio de racks, consumo de electricidad, y esfuerzo de administración.

**Hosting:**

Aparentemente, la virtualización en el nivel de SO es el único tipo de virtualización que las empresas de hosting pueden pagar y usar para vender Entornos Virtuales baratos a sus clientes. Notar que cada EV tiene acceso completo de root, lo que significa que el dueño del EV puede reinstalar cualquier cosa, y aún usar cosas tales como iptables (reglas de cortafuego).

**Desarrollo y Pruebas:**

Usualmente los desarrolladores y testers necesitan acceder a un grupo de distribuciones de Linux, y necesitan reinstalarlas desde cero con frecuencia. Con OpenVZ, pueden tener todo en un sólo equipo, sin ninguna necesidad de reiniciar, con rendimiento nativo, y un nuevo EV se puede crear en sólo un minuto. Clonar un VE es también muy simple: solamente se necesita copiar el área del EV y el archivo de configuración.

**Educativo:**

Varios alumnos pueden compartir un EV. Cada uno puede trabajar con una diferente distribución de Linux. Un nuevo EV se puede (re)crear en un minuto.

## 2.2 Docker

Docker es un proyecto de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores de software, proporcionando una capa adicional de abstracción y automatización de virtualización a nivel de sistema operativo en Linux. Docker utiliza características de aislamiento de recursos del kernel de Linux, tales como cgroups y espacios de nombres (namespaces) para permitir que "contenedores" independientes se ejecuten dentro de una sola instancia de Linux, evitando la sobrecarga de iniciar y mantener máquinas virtuales.

El soporte del kernel de Linux para los espacios de nombres aísla de vista, en su mayoría, una aplicación del entorno operativo, incluyendo árboles de proceso, red, ID de usuario y sistemas de archivos montados, mientras que los cgroups del kernel proporcionan aislamiento de recursos, incluyendo la CPU, la memoria, el bloque de E/S y de la red. Desde la versión 0.9, Docker incluye la biblioteca libcontainer como su propia manera de utilizar directamente las facilidades de virtualización que ofrece el kernel de Linux, además de utilizar las interfaces abstraídas de virtualización mediante libvirt, LXC (Linux Containers) y systemd-nspawn.

Otro de los puntos fuertes de Docker es que Docker Engine (el servidor de Docker encargado de ejecutar los contenedores), por defecto es capaz de conectarse a Docker Hub (registro de imágenes público) donde se encuentran disponibles miles de imágenes ya creadas y descargables a demanda. Un análisis en mayo de 2015 mostró las siguientes organizaciones como las principales contribuyentes de Docker: Red Hat (mayores contribuyentes, aún más que el equipo de Docker en sí), el equipo de Docker, IBM, Google y Cisco Systems.

También es posible crear un registro privado de imágenes donde se almacenarían las imágenes asociadas a los servicios desarrollados por una compañía.

Los contenedores de Docker aportan muchos beneficios a los desarrolladores y administradores de sistemas, es un cambio de filosofía radical que requiere la implicación de las dos partes. Simplifica enormemente el ciclo de vida completo de la aplicación así como el despliegue entre los diferentes entornos.

Los contenedores están destinados a ser completamente estandarizados. Esto significa que el contenedor se conecta al host y a cualquier lado fuera del contenedor usando interfaces definidas. Una aplicación contenerizada no se debe involucrar ni preocupar sobre los detalles de los recursos o arquitectura del host. Esto simplifica los supuestos del desarrollo sobre el ambiente operativo. Del mismo modo, que el host, cada contenedor es una caja negra. No se preocupan de los detalles de la aplicación en el interior.

Uno de los beneficios de la abstracción entre el sistema host y los contenedores es eso, a partir de un diseño correcto de aplicación, escalar puede ser simple y directo. Un diseño orientado a servicio combinado con aplicaciones contenerizadas proporcionan las bases para una fácil escalabilidad. Un developer puede ejecutar algunos contenedores en su estación de trabajo, mientras que este sistema puede escalar horizontalmente en ensayo o área de pruebas. Cuando los contenedores suben a producción, estos pueden escalar nuevamente.

Los contenedores permiten al desarrollador agrupar una aplicación o un componente de la misma junto a todas sus dependencias como unidad. El sistema host no tiene que preocuparse por las dependencias necesarias para ejecutar una aplicación específica. Mientras pueda correr Docker, será capaz de correr los contenedores de Docker. Esto hace que la administración de dependencias sea fácil y simplifica la administración de versiones de la aplicación. Los sistemas host y los equipos de operaciones dejan de ser responsables de administrar las dependencias necesarias para la aplicación ya que deben estar contenidas dentro del propio contenedor.

Algunas ventajas de Docker:

* Las instancias se arrancan en pocos segundos.
* Es fácil de automatizar e implantar en entornos de integración continua.
* Existen multitud de imágenes que pueden descargarse y modificarse libremente.
* Ambientes Replicables.

Como inconvenientes cabe destacar:

* Sólo puede usarse de forma nativa en entornos Unix aunque se puede virtualizar gracias a boot2docker tanto en OSX como en Windows.
* Las imágenes sólo pueden estar basadas en versiones de Linux modernas (kernel 3.8 mínimo).
* Como es relativamente nuevo, puede haber errores de código entre versiones.

**Dockerfiles:**

Los archivos de Docker (Dockerfiles) permiten a los usuarios definir las acciones exactas necesarias para crear una nueva imagen de contenedor. Esto te permite escribir la ejecución de tu ambiente como si fuera código, almacenándola en un controlador de versiones si es necesario. El mismo archivo de Docker en el mismo entorno siempre producirá una imagen de contenedor idéntica.

Debido a que es posible crear imágenes de contenedores a través de un proceso iterativo, es muy frecuente agregar los pasos de configuración dentro de un Dockerfile para que se conozcan los pasos necesarios. Los Dockerfiles son simples archivos de compilación que describen como crear la imagen del contenedor desde un punto de partida conocido.

Beneficios de los Dockerfiles:

*Fácil versionamiento*: Los Dockerfiles por si solos pueden ser rastreables a través de un controlador de versiones para comparar sus cambios y revertir cualquier error.

*Predictibilidad*: Crear una imagen desde un Dockerfile ayuda a reducir el error humano desde el proceso de creación de la imagen.

*Trazabilidad*: Si tu planeas compartir tus imágenes, es una buena idea proporcionar el Dockerfile que crea una imagen como una forma para que otros usuarios puedan auditar el proceso. Esto básicamente proporciona el historial de la línea de comandos con los pasos tomados para crear la imagen.

*Flexibilidad*: Crear imágenes desde Dockerfile permite sobrescribir los valores predeterminados que las compilaciones proporcionan. Esto significa que no tienes que proporcionar las opciones de tiempo de arranque para que la imagen funcione según lo previsto.

**Arquitectura de aplicaciones contenerizadas:**

Cuando se diseñan aplicaciones para ser implementadas en contenedores, una de las principales áreas de preocupación es la actual arquitectura de la aplicación.

Generalmente, las aplicaciones contenerizadas funcionan mejor cuando se implementan en un diseño orientado a servicio.

Las aplicaciones orientadas a servicio dividen la funcionalidad del sistema en componentes discretos que se comunican unos con los otros a través de interfaces bien definidas. La tecnología de contenedores por si misma se encarga de este tipo de diseño debido a que permite a cada componente escalar o crecer de manera independiente.

# 3. Criterios de comparación

En los sub-apartados de este apartado se deben indicar cada uno de los criterios (también llamados factores, propiedades, características, indicadores, etc.), que se usarán en la comparación.

Los criterios deben organizarse en categorías (al menos 3 categorías). El número de criterios totales no puede ser inferior a 20.

Las categorías dependerán del tipo de tecnología, pueden ser categorías como “General”, “Utilidades”, “Rendimiento”, etc.

Los criterios a definir en cada categoría también dependerán del tipo de tecnologías a comparar. En el siguiente apartado hay algunos ejemplos.

## 3.1 Categoría A: Nombre

### 3.1.1 Criterio A.1: Nombre

Por cada criterio hay que indicar el nombre, una breve descripción, y el tipo de valor a asignar al criterio. El número de criterios no puede ser inferior a 20.

Por ejemplo, si se comparan dos herramientas CASE, un criterio podría ser el autor, entonces en este apartado se podría incluir este texto:

*3.1.1 Criterio A.1: Autor de la herramienta*

*Nombre del criterio: Autor.*

*Descripción: Nombre de la persona, institución o empresa que ha creado la herramienta.*

*Tipo de valor: Texto libre.*

Otro ejemplo de criterio con valor Si/No:

*3.1.1 Criterio A.1: Generación de código Java*

*Nombre del criterio: Generación código Java.*

*Descripción: Indica si la herramienta CASE incluye funcionalidad para generar archivos fuente .java a partir de diagramas de clase.*

*Tipo de valor: Booleano (Si/No).*

Otro ejemplo de criterio con valor numérico:

*3.1.1 Criterio A.1: Número de proyectos simultáneos*

*Nombre del criterio: Proyectos simultáneos.*

*Descripción: Indica el número máximo de proyectos de modelado que deja tener abiertos la herramienta de forma simultánea.*

*Tipo de valor: Numérico.*

### 3.1.2 Criterio A.2: Nombre

### 3.1.n Criterio A.n: Nombre

## 3.2 Categoría B: Nombre

### 3.2.1 Criterio B.1: Nombre

### 3.2.2 Criterio B.2: Nombre

### 3.2.n Criterio B.n: Nombre

## 3.3 Categoría Z: Nombre

### 3.3.1 Criterio Z.1: Nombre

### 3.3.2 Criterio Z.2: Nombre

### 3.3.n Criterio Z.n: Nombre

# 4. Evaluación de los criterios por tecnología

## 4.1 Evaluación de los criterios para la tecnología 1

Debe incluir al menos una tabla con la siguiente estructura.

|  |  |
| --- | --- |
| CRITERIOS | EVALUACIÓN |
| Criterio A.1: Nombre |  |
| Criterio A.2: Nombre |  |
| Criterio A.n: Nombre |  |
| … |  |
| Criterio B.1: Nombre |  |
| Criterio B.2: Nombre |  |
| … |  |

Y algunos comentarios aclaratorios sobre aquellos criterios cuyo valor indicado en la tabla no sea suficiente para entenderlo.

La tabla anterior es obligatoria y deben completarla los autores del trabajo, aunque se pueden incluir otros gráficos o tablas complementarias copiadas y pegadas desde diversas fuentes de información, siempre que debajo de cada uno se indique la fuente (al menos la URL).

## 4.2 Evaluación de los criterios para la tecnología 2

# 5. Comparación de las tecnologías

Debe incluir al menos una tabla resumen, en sección de página horizontal, cruzando los criterios y los valores de cada tecnología. Con una columna de comentarios sobre la comparación

Esta tabla anterior es obligatoria y deben completarla los autores del trabajo, aunque se pueden incluir otros gráficos o tablas complementarias copiadas y pegadas desde diversas fuentes de información, siempre que debajo de cada uno se indique la fuente (al menos la URL).

|  |  |  |  |
| --- | --- | --- | --- |
| CRITERIOS | TECNOLOGÍA A | TECNOLOGÍA B | COMENTARIOS |
| A.1 |  |  |  |
| A.2 |  |  |  |
| … |  |  |  |
| B.1 |  |  |  |
| B.2 |  |  |  |
| … |  |  |  |

# 6. Recomendaciones

Deben platearse posibles situaciones de uso, y recomendar justificadamente una u otra tecnología en función de la situación. Al menos 2 situaciones diferentes.

## 6.1 Situación 1

### 6.1.1 Descripción de la situación

*Una posible situación en el caso de comparar dos herramientas CASE, podría ser el caso de una empresa de desarrollo muy interesada en tecnologías open source, que programa sólo en Java, con equipos de desarrollo pequeños, que utiliza UML como notación, etc, etc*…

### 6.1.2 Recomendación de tecnología a utilizar

Debe indicarse la tecnología propuesta para esa situación.

Debe incluirse una tabla como la siguiente, mostrando las ventajas, respecto a los criterios, que ofrece cada tecnología en esa situación concreta.

Incluir sólo los criterios sobre los que se aprecien ventajas de una de las tecnologías frente a otra. No incluir criterios que no sean relevantes para la decisión (por ejemplo, el criterio “autor” seguramente no será relevante).

|  |  |  |
| --- | --- | --- |
| Criterios relevantes para la decisión | Ventajas tecnología 1 | Ventajas tecnología 2 |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

## 6.2 Situación 2

### 6.2.1 Descripción de la situación

### 6.2.2 Recomendación de tecnología a utilizar

---------------------------

(Hay que cumplir la estructura básica indicada de secciones. Pero si se desea se pueden añadir otras secciones como anexos. Por ejemplo, alguna encuesta de opinión realizada sobre las tecnologías, etc.)