UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

SISTEMA INFORMÁTICO CLIENTE-SERVIDOR PARA EJECUCIÓN DE ESCRITORIO REMOTO CON ACELERACIÓN GRÁFICA

Félix Rausell Martínez

Tutor: Francisco José Martínez Zaldívar

Trabajo Fin de Grado presentado en la Escuela Técnica Superior de Ingenieros de Telecomunicación de la Universitat Politècnica de València, para la obtención del Título de Graduado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Curso 2017-18

Valencia, 4 de julio de 2018

Escuela Técnica Superior de Ingeniería de Telecomunicación
Universitat Politècnica de València
Edificio 4D. Camino de Vera, s/n, 46022 Valencia
Tel. +34 96 387 71 90, ext. 77190
www.etsit.upv.es

VLC/ CAMPUS VALENCIA, INTERNATIONAL CAMPUS OF EXCELLENCE

EMAS Gestión ambiental verificada REG.NO. ES-CV- 000030

Resumen

El objetivo de este proyecto es el estudio sobre el desarrollo y despliegue de un sistema informático de modelo cliente-servidor dedicado a ofrecer un entorno de escritorio remoto con aceleración gráfica.

Consiste en una serie de instancias virtuales independientes ejecutándose en un clúster de computadoras haciendo uso de sus recursos gráficos y a los que se accede a través de la red mediante el protocolo VNC con el propósito de ejecutar aplicaciones CAE (Computer-Aided Engineering) en equipos de bajas prestaciones remotamente. Estas computadoras están supervisadas por otra que haría las veces de servidor Proxy y de balanceador de carga, dirigiendo las conexiones automaticámente basándose en las instancias que han sido menos utilizadas.

Para llevarse a cabo se realizan pruebas sobre la viabilidad de introducir el sistema en un entorno de producción real tales como consumo de recursos de hardware y ancho de banda con distintas configuraciones de compresión y cifrado. Esto último está supeditado a la opinión subjetiva de los usuarios en cuanto a usabilidad, estableciendo así un sistema de puntuación MOS (Mean Opinion Score).

Palabras clave

- Escritorio remoto
- Protocolo VNC
- SSL/TLS
- Clúster
- Balanceador de carga

Resum

L'objectiu d'aquest projecte és l'estudi sobre el desenvolupament i desplegament d'un sistema informàtic de model client-servidor dedicat a oferir un entorn d'escriptori remot amb acceleració gràfica.

Consisteix en una sèrie d'instàncies virtuals independents executant-se en un clúster d'ordinadors fent ús dels seus recursos gràfics i als quals s'accedeix a través de la xarxa mitjançant el protocol VNC amb el propòsit d'executar aplicacions CAE (Computer-Aided Engineering) en equips de baixes prestacions remotament. Aquests ordinadors estan supervisats per una altre que faria les vegades de servidor intermediari i de balanç de càrrega, dirigint les connexions automàticament basant-se en les instàncies que han estat menys utilitzades.

Per dur-se a terme es realitzen proves sobre la viabilitat d'introduir el sistema a un entorn de producción real com ara consum de recursos de hardware i ample de banda amb diferents configuracions de compressió i xifrat. Això últim està supeditat a l'opinió subjectiva dels usuaris pel que fa a usabilitat, establint així un sistema de puntuació MOS (Mean Opinion Score).

Paraules clau

- Escriptori remot
- Protocol VNC
- SSL/TLS
- Clúster
- Balancejador de càrrega

Abstract

The objective of this project is about the study of the development and deployment of a client-server model computer system dedicated to providing a graphically accelerated remote desktop environment.

It consists of a series of independent virtual instances running on a cluster of computers using their graphic resources and accessed through the network using the VNC protocol for the purpose of running CAE (Computer-Aided Engineering) applications on low-performance computers remotely. These computers are monitored by another computer that would act as a proxy server and load balancer, automatically routing connections based on the least used instances because users can work in any instance indifferently.

Tests are conducted on the feasibility of introducing the system into a real production environment such as hardware resources and bandwidth consumption with different compression and encryption configurations. This depends of the subjective opinion of users regarding usability, thus establishing a MOS (Mean Opinion Score) scoring system.

Keywords

- Remote desktop
- VNC protocol
- SSL/TLS
- Cluster
- Load-Balancer

Índice

Capítulo 1. Introducción

De forma general se puede definir la "nube" como un modelo de consumir tecnología como servicio (aplicaciones, computación y almacenamiento), sin necesidad de inversiones, de forma segura y accesible desde cualquier lugar.

En los últimos tiempos, han aflorado una serie de servicios informáticos basados en la "nube", esto es, ejecutándose en máquinas remotas a las que los usuarios acceden a través de la red, que ha sido bien recibida en entornos empresariales dado que posee una serie de ventajas:

- Reduce costes, tanto iniciales como de mantenimiento posterior.
- Aumenta la disponibilidad de los servicios informáticos, 24x7
- Evita inversiones en activos tales como hardware y software.
- Permite ajustar el servicio y posee una gran escalabilidad

Estos servicios en la nube abarcan muchos conceptos como el IaaS (*Infraestructure as a Service*) donde se ofrece sustituir la infraestructura física informática por servicios de máquinas virtuales de alta disponibilidad y redundancia o el PaaS *(Platform as a Service*), donde se ofrecen entornos para poder desplegar y administrar aplicaciones.

Más recientemente ha surgido un concepto nuevo debido a las altas prestaciones en cuanto a ancho de banda que nos ofrecen las conexiones de Internet que es el DaaS (*Desktop as a Service*), en el cual, en vez de trabajar sobre un escritorio en la máquina localmente, este se ejecuta en un equipo remoto lo cual confiere beneficios en términos de control y seguridad de los datos que se traten (generalmente confidenciales en el caso de las empresas).

1.1 Motivación

En este trabajo se intenta desarrollar un sistema de escritorio remoto que posea la capacidad de utilizar la potencia gráfica de una GPU para poder ser utilizada por los usuarios para desarrollar labores con software Computer-Aided Engineering (Fig. 1). Esto es debido a que es realizado para su uso en un entorno de trabajo dedicado a la ingeniería mecánica.

Para poder desarrollar correctamente su labor, el equipo de ingeniería ha de utilizar actualmente estaciones de trabajo con un elevado coste. En éstas, se han de realizar tareas que requieren de grandes recursos de capacidad de cálculo y aceleración gráfica

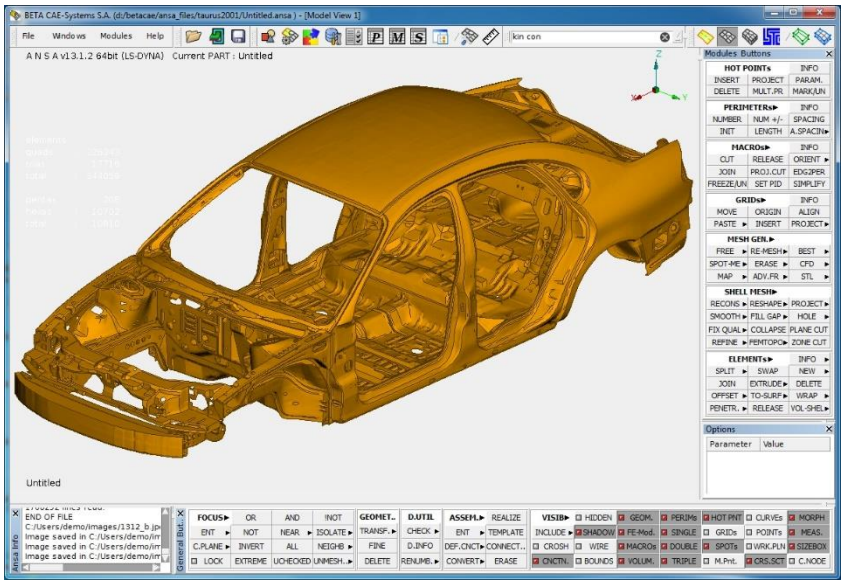

Figura 1 Interfaz gráfica de software Computer-Aided-Engineering

Para controlar los costes a la hora de añadir más miembros a un equipo de ingeniería, se ha planteado la posibilidad de utilizar un sistema de cliente-servidor donde el servidor realizaría el trabajo con carga gráfica y el cliente se ejecutaría en un equipo de bajas prestaciones.

Como cada usuario habría de trabajar en una instancia independiente del software se pensó en virtualización para el desarrollo, generando una serie de servidores en paralelo.

Pero, en principio, las soluciones de virtualización existentes no se desarrollan con el objetivo de un buen rendimiento gráfico, por lo que se decanta por el desarrollo de una solución basándose en tecnologías surgidas a partir de la expansión de la computación en la "nube", los contenedores.

1.2 Objetivo

El objetivo de este trabajo es implementar un sistema cliente-servidor de escritorio remoto cuya característica principal sea la capacidad de ofrecer aceleración gráfica, el cual permita a un número de usuarios trabajar desde ordenadores de bajas prestaciones (comúnmente conocido como *Thin Client*) con software que necesita unos requisitos específicos de hardware.

Para ello se fijan una serie de objetivos parciales:

- Diseñar un software de tipo servidor que pueda proporcionar un sistema de escritorio remoto capaz de ejecutar otros subprogramas cuya interfaz gráfica requiera de aceleración gráfica 3D. Más concretamente, software de ayuda a la ingeniería (o CAE) que hace uso de la tecnología OpenGL.
- Introducir varias réplicas del software en un número de servidores según requerimientos y dotarlo de un sistema de autogestión mediante balanceo de carga y de monitorización de las posibles necesidades para un correcto mantenimiento.
- Asegurar que el sistema cuenta con los requisitos de seguridad necesarios para el tratamiento de información de tipo confidencial, esto es, que la información viaje cifrada entre el cliente y el servidor.
- Cada servidor debe ser ejecutado sobre un ordenador anfitrión cuyo sistema operativo sea Fedora y, preferiblemente, hacer uso de contenedores Docker. El hardware de aceleración grafica consistirá en una GPU NVIDIA de la familia QUADRO, especialmente diseñadas para el correcto funcionamiento del software requerido.

1.3 Planificación del trabajo

Dados los requisitos, se organiza el trabajo en una serie de etapas ya que consta de bastantes elementos independientes y hay que conseguir la correcta integración entre ellos. Por suerte, todo el software necesario es *open source*, habiendo una gran cantidad de documentación en la red.

En primera instancia, se busca información en la red sobre la viabilidad del trabajo propuesto, se comparan distintas alternativas y se determina el software con el que se va a llevar a cabo.

En este trabajo vamos a necesitar los siguientes elementos:

- Una pantalla virtual que sea capaz de ejecutar instrucciones OpenGL, para ello se opta por Xvfb, programa en principio pensado para automatizar tests de software con interfaz gráfica pero que nos sirve para este cometido. Para poder ejecutar instrucciones OpenGL y debido a las limitaciones de esta tecnología a la hora de ejecutarse remotamente se opta por una solución denominada VirtualGL, la cual hace que el rendimiento gráfico corra del cargo del servidor en vez del cliente.

- Un contenedor Docker donde se encapsulará la pantalla virtual y un servidor VNC, que será el protocolo que utilizaremos para transmitir nuestro escritorio remoto a través de la red. Este contenedor se replicaría tantas veces como usuarios queramos introducir en el sistema. Este contenedor será el servidor de nuestro sistema y está estructurado de la forma que se observa en la siguiente figura (Fig. 2).

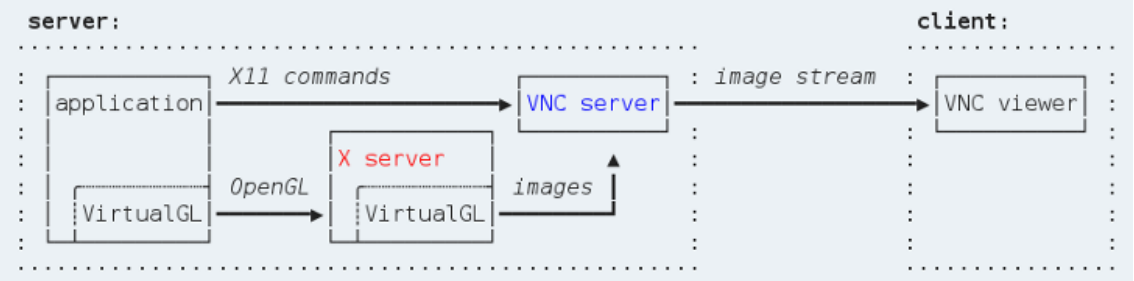

Figura 2 Esquema de comunicación VNC entre cliente-servidor con VirtualGL

El elemento marcado en rojo se encargará del *renderizado* 3D mientras que el marcado en azul se encargará del *renderizado* 2D, de este modo se evitarán los problemas a la hora de transportar el escritorio a través de la red.

- Un proxy que hará la función de balanceador de carga y redirigirá los clientes a los servidores VNC que se encuentren sin uso. En este caso nos hemos decantado por HAProxy como proxy TCP y Nginx como proxy HTTP.

Durante los siguientes capítulos se expondrá en qué consiste cada uno de estos elementos y las características que poseen para que nos hayamos decantado por ellos para el desarrollo de nuestro sistema y su posterior implementación.

1.4 Estructura del trabajo

El trabajo se encuentra organizado de la forma expuesta a continuación:

Primero, se describen las partes de software que se han utilizado para el desarrollo del trabajo.

En el capítulo 2 se habla de la pantalla virtual que hemos de desarrollar. Para ello se expone el sistema de ventanas X, el cual es el método para reproducir gráficos usado por las distribuciones GNU/Linux desde hace muchos años. Explicaremos su funcionamiento y las tecnologías usadas para poder desarrollar el escritorio remoto.

En el capítulo 3 se habla del contenedor donde se han de ejecutar la pantalla virtual y el servidor VNC. Hace uso de Docker, nuestra solución para "*virtualizar*" (aunque no hacemos exactamente eso) nuestros escritorios remotos. Docker será la herramienta con la que generaremos servicios independientes.

El capítulo 4 trata sobre VNC, la tecnología de escritorio remoto usada en el trabajo. Hablaremos del protocolo, su funcionamiento y seguridad. Además del software que usaremos como servidor y como cliente.

El capítulo 5 consistirá en una escueta descripción del software proxy que utilizaremos para facilitar a los usuarios la conexión con los distintos servidores. Este software se encargará de repartir a los clientes según la carga de los equipos anfitriones.

En el capítulo 6 se trata el desarrollo del trabajo. Se explica el código y la configuración de los equipos que se utiliza para conseguir múltiples instancias de escritorio remoto, así como su gestión, mantenimiento, etc.

El capítulo 7 trata sobre las soluciones contempladas a nivel de infraestructura informática para desplegar el sistema, su coste y su período de amortización.

El capítulo 8 habla de los resultados obtenidos en función de las variables que nos interesan para determinar la viabilidad del proyecto, que son el ancho de banda consumido en las distintas configuraciones del servidor VNC y la experiencia de usuario.

En el capítulo 9 se exponen las conclusiones y los planes a futuro del proyecto.

Capítulo 2. Pantalla virtual

2.1 Sistema de ventanas X

Para entender el proceso para conseguir un escritorio remoto con aceleración gráfica, hay que presentar el sistema de ventanas X. Es un sistema de gestión de ventanas usado en los sistemas GNU/Linux como el que vamos a basarnos que sirve para dotar a estos sistemas operativos de una interfaz gráfica con el que pueda interactuar un usuario.

El sistema de ventanas X distribuye el procesamiento de aplicaciones especificando enlaces cliente-servidor. El servidor provee servicios para acceder a la pantalla, teclado y ratón, mientras que los clientes son las aplicaciones que utilizan estos recursos para interacción con el usuario. De este modo mientras el servidor se ejecuta de manera local, las aplicaciones pueden ejecutarse remotamente desde otras máquinas[1].

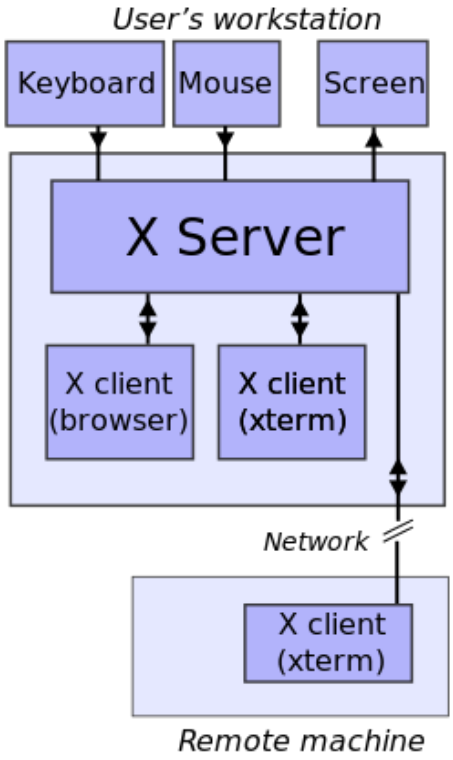

Figura 3 Esquema del servidor X y sus entradas/salidas

X no es un gestor de ventanas ya que necesita de uno para controlar el manejo de ventanas. Esto trae la ventaja de que hace de X estrictamente un sistema gráfico, de tal modo que un cliente X podría estar enviando un gráfico a una pantalla, a una impresora o a cualquier otro hardware, flexibilizando la salida gráfica.

El problema que nos encontramos al plantear este proyecto es el uso de la tecnología OpenGL, la cual, aunque es posible ser ejecutada directamente en un cliente remoto, requiere de unas capacidades de red que es muy difícil que se den (ancho de banda, latencia, baja congestión…) por lo tanto habrá que realizar un *workaround* para conseguir un modo estable y resistente a problemas en la red como los mencionados anteriormente.

En nuestro caso, al realizarse en servidores *headless*, esto quiere decir, sin un monitor físico conectado, habrá que establecer para nuestro servidor X una "pantalla" virtual que será donde se

reproduzcan las interfaces gráficas de los programas que ejecutemos para que posteriormente sean representados en los clientes a través de una comunicación VNC.

2.2 Xvfb

Para el uso de software que dependa de una interfaz gráfica haremos uso de Xvfb ya que el servidor no se encuentra conectado a ningún monitor.

Xvfb o X virtual framebuffer es un servidor X que ejecuta todas las operaciones gráficas en memoria, sin mostrar nada por pantalla. Desde el punto de vista del cliente, Xvfb actúa exactamente como cualquier otro servidor X, sirviendo las peticiones y enviando excepciones y errores habituales, solo que no sale nada por ninguna pantalla. Este servidor virtual no requiere que la máquina tenga ningún dispositivo gráfico. Tan solo es necesario la capacidad de acceder a la red.

Su funcionamiento es muy sencillo, simplemente hay que lanzarlo mediante una instrucción de línea de comandos:

```
Xvfb :1 –screen 0 1920x1080x24
```

Con esta instrucción generamos una "pantalla" virtual cuya resolución son 1920 píxeles de ancho por 1080 píxeles de alto y 24 bits de profundidad de color, esto último es necesario para el correcto funcionamiento de OpenGL.

A continuación le decimos al SO que la dirección de la pantalla que va a usar en este caso es :1, ya que :0 es la dirección por defecto.

```
export DISPLAY=:1
```

Con esta última instrucción, el software con interfaz gráfica se representará sobre nuestra "pantalla" virtual.

2.3 OpenGL

El software CAE que queremos ejecutar muestra modelos en 3D de productos industriales compuestos por millones de polígonos y para poder representarse en entornos GNU/Linux, los desarrolladores optaron por el uso de OpenGL.

OpenGL (Open Graphics Library) es una especificación estándar que define una API multilenguaje y multiplataforma para escribir aplicaciones que produzcan gráficos 2D y 3D. Se usa ampliamente en CAD, realidad virtual, representación científica, visualización de información y simulación de vuelo.

Básicamente OpenGL consiste en una serie de librerías y rutinas de clases por lo cual, OpenGL no es un paquete de software de renderizado y modelado como Blender o 3D Max, básicamente es una API de bajo nivel que proporciona una interfaz de hardware de gráficos. No es por lo tanto ningún lenguaje de programación, sino tan sólo un conjunto de librerías que son utilizadas a través de lenguajes de programación para conseguir un interfaz software entre las aplicaciones y el hardware gráfico.

Se desarrolló para unificar a la gran variedad de fabricantes de GPUs que se encontraban desarrollando sus propias soluciones de representación de gráficos en 3D, ayudando así a los desarrolladores de aplicaciones que ya no habrían de preocuparse por la tarjeta gráfica que hubiera instalada en el ordenador del usuario final.

El problema que se presenta es que OpenGL resulta muy problemático su uso a través de la red debido al alto volumen de recursos que consumiría, por lo cual se había de buscar una alternativa a su uso en clientes remotos. Esa alternativa se llama VirtualGL.

2.4 VirtualGL

VirtualGL[2] es un kit de herramientas de código abierto que proporciona a cualquier software de visualización remota Unix o Linux la capacidad de ejecutar aplicaciones OpenGL con aceleración gráfica 3D. Algunas soluciones de visualización remota no se pueden utilizar con aplicaciones OpenGL. Otros obligan a las aplicaciones OpenGL a utilizar un *renderizador* lento, exclusivamente software, con un detrimento del rendimiento y de la compatibilidad. El método tradicional de mostrar aplicaciones OpenGL en un servidor X remoto (*renderizado* indirecto) es compatible con la aceleración de hardware 3D, pero este método hace que todos los comandos OpenGL y los datos 3D que se envían a través de la red se *rendericen* en el equipo cliente (Fig. 4). Por lo tanto no se ajustaba a los requisitos que se necesitaban.

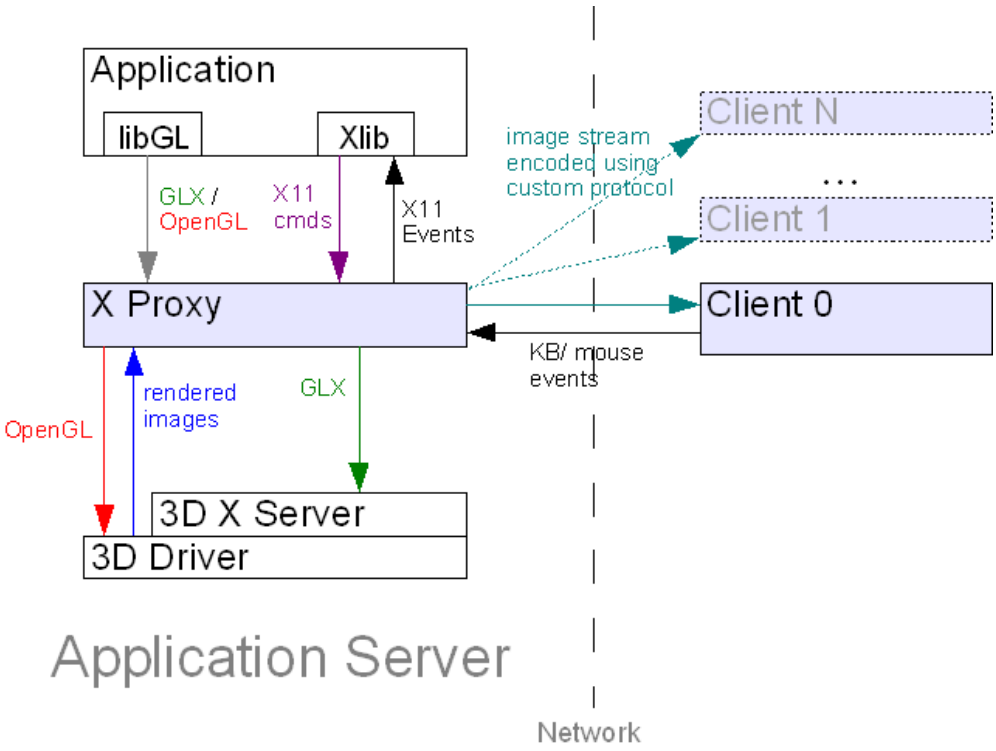

Figura 4 Esquema del funcionamiento de OpenGL a través de la red

Con VirtualGL, los comandos OpenGL y los datos 3D se redirigen a una unidad de procesamiento gráfico (GPU) en el servidor de aplicaciones, y sólo las imágenes 3D renderizadas se envían a la máquina cliente.

Normalmente, una aplicación OpenGL enviaría todos sus comandos de dibujo y datos, tanto 2D como 3D, a un servidor X, que puede estar ubicado a través de la red desde el servidor de

aplicaciones. VirtualGL, sin embargo, emplea una técnica llamada "split rendering" para forzar a los comandos y datos 3D de la aplicación a pasar a una GPU en el servidor de aplicaciones. VirtualGL logra esto precargando un objeto compartido dinámico (DSO) en la aplicación OpenGL en tiempo de ejecución. Este DSO intercepta un puñado de comandos GLX, OpenGL y X11 necesarios para realizar el *renderizado* dividido. Cuando la aplicación intenta utilizar una ventana X para el *renderizado* de OpenGL, VirtualGL intercepta la petición, crea un búfer de píxeles 3D ("Pbuffer") correspondiente en la memoria de vídeo del servidor de aplicaciones y utiliza el búfer para el renderizado de OpenGL. Cuando la aplicación intercambia los búferes de dibujo OpenGL o refresca el búfer de comandos de OpenGL para indicar que ha terminado de *renderizar* un *frame*, VirtualGL lee los píxeles del búfer y los envía al cliente (Fig. 5).

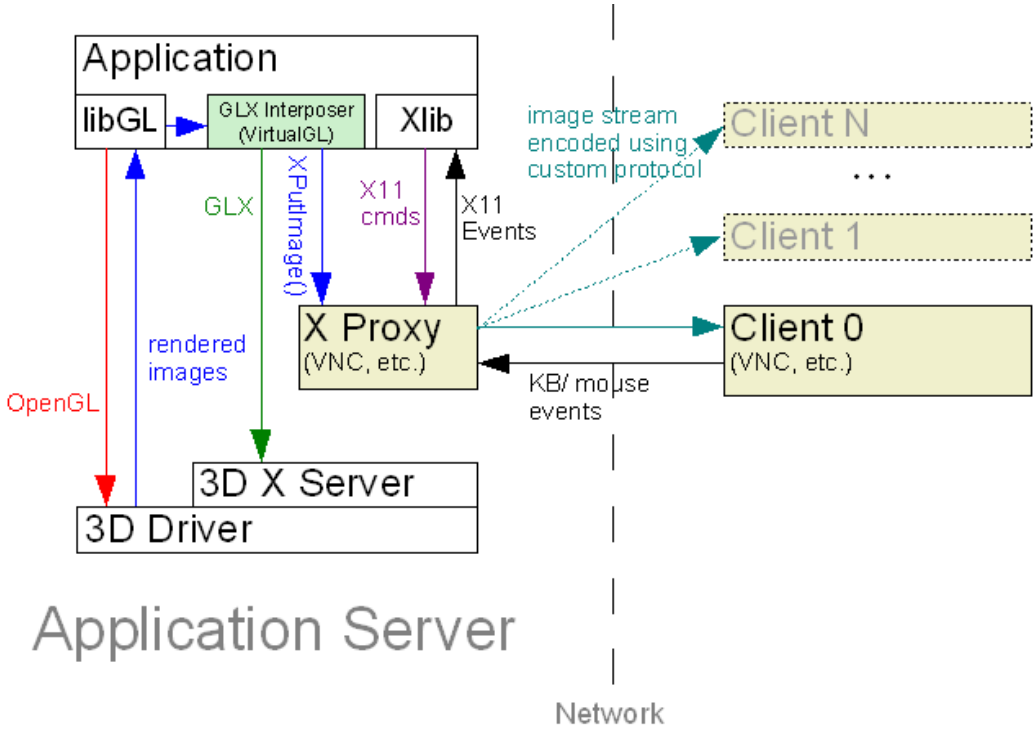

Figura 5 Esquema de funcionamiento de VirtualGL a través de la red

2.5 Controlador NVIDIA

Para poder utilizar la GPU NVIDIA de la que obtendremos la aceleración gráfica necesaria para nuestro objetivo se necesita el controlador que en este caso, se ofrece en Internet para el sistema operativo usado.

El controlador consiste en software diseñado para un sistema operativo específico. El sistema operativo utiliza el controlador para comunicarse con un dispositivo de hardware determinado. Son librerías (archivos) adicionales para el sistema operativo que controlan el hardware en este caso una tarjeta gráfica del fabricante NVIDIA donde las actualizaciones se dan con el paso del tiempo por los resultados que se van encontrando con cada aplicación nueva. Es absolutamente necesario, dado que sin él, el ordenador no sabe el "idioma" con el que comunicarse con la gráfica.

Este fabricante de GPU ha sido elegido por el buen rendimiento que dan sus controladores en sistemas operativos GNU/Linux como los que vamos a usar para el desarrollo del proyecto, además, se encuentra en la vanguardia del desarrollo de tecnología como el cálculo en paralelo o la Inteligencia Artificial por lo que resulta interesante contar con una GPU de estas características para futuros proyectos

Capítulo 3. Contenedores

Los contenedores son una alternativa a las máquinas virtuales que aportan una mayor flexibilidad que éstas ya que usan los recursos del sistema anfitrión de una forma escalada, por lo tanto no hay que dedicarle exclusivamente recursos a cada uno de ellos. Esto proporciona una gran escalabilidad a la hora de plantearse diseñar servicios que requieran de un alto rendimiento.

3.1 Docker

Al principio de plantearse el trabajo, se pensó en una solución basada en máquinas virtuales pero al investigar sobre el rendimiento de estas a la hora de representar gráficos en 3D complejos, se decidió decantarse por hacer uso de Docker.

Docker es una herramienta diseñada para facilitar la creación, implementación y ejecución de aplicaciones mediante el uso de contenedores. Los contenedores permiten a un desarrollador empaquetar una aplicación con todas las partes que necesita, como bibliotecas y otras dependencias, y enviar todo en un solo paquete. De este modo, gracias al contenedor, el desarrollador puede estar seguro de que la aplicación se ejecutará en cualquier otra máquina Linux o Windows, independientemente de cualquier configuración personalizada que la máquina pueda tener y que pueda diferir de la máquina utilizada para escribir y probar el código.

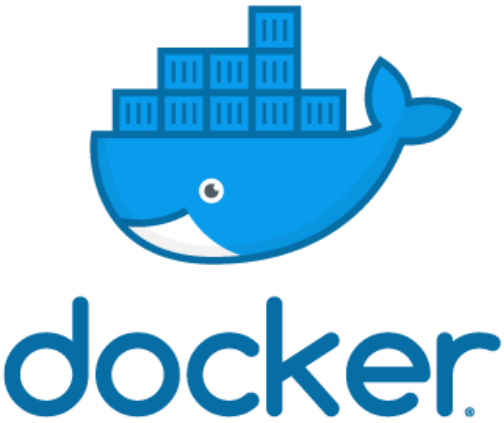

Figura 6 Logotipo de Docker

En cierto modo, Docker es un poco como una máquina virtual[3]. Pero a diferencia de una máquina virtual, en lugar de crear todo un sistema operativo virtual, Docker permite que las aplicaciones usen el mismo kernel de Linux que el sistema en el que se ejecutan (Fig. 7) y sólo requiere que las aplicaciones se envíen con cosas que no se estén ejecutando en el equipo anfitrión. Esto proporciona un aumento significativo del rendimiento y reduce el tamaño de la aplicación.

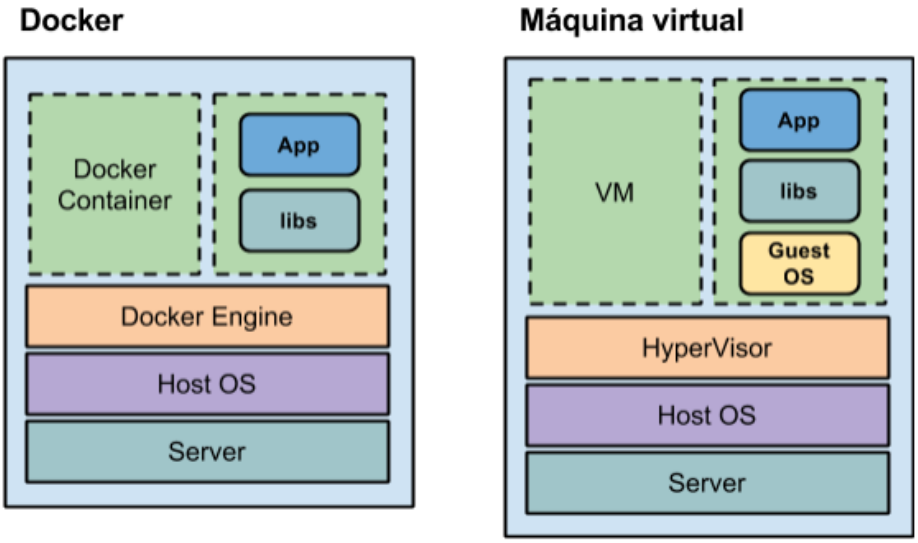

Figura 7 Comparativa entre Docker y máquinas virtuales

Docker es una herramienta diseñada para beneficiar tanto a los desarrolladores como a los administradores de sistemas, por lo que forma parte de muchas cadenas de herramientas *DevOps* (desarrolladores + operaciones). Para los desarrolladores, significa que pueden concentrarse en escribir código sin preocuparse por el sistema en el que se ejecutará en última instancia. También les permite tener una ventaja utilizando uno de los miles de programas ya diseñados para ejecutarse en un contenedor Docker como parte de su aplicación. Para el personal de operaciones, Docker ofrece flexibilidad y reduce potencialmente el número de sistemas necesarios debido a su pequeño tamaño y menor sobrecarga.

Docker proporciona encapsulación de hardware y software permitiendo que varios contenedores se ejecuten en el mismo sistema al mismo tiempo, cada uno con su propio conjunto de recursos (CPU, memoria, etc.) y su propio conjunto dedicado de dependencias (versión de biblioteca, variables de entorno, etc.). Docker también proporciona una implementación de Linux portátil: Los contenedores Docker pueden ejecutarse en cualquier sistema Linux con kernel 3.10 o posterior. Todas las principales distribuciones de Linux son compatibles con Docker desde 2014. La encapsulación y la implementación portátil son valiosas tanto para los desarrolladores que crean y prueban aplicaciones como para el personal de operaciones que las ejecuta en los centros de datos.

3.1.1 Características

Docker ofrece muchas más funciones importantes[4], de las cuales haremos uso de las siguientes:

- Con la potente herramienta de línea de comandos de Docker, **docker build**, crearemos imágenes de Docker a partir de código fuente y binarios, utilizando la descripción proporcionada en un "**Dockerfile**".
- La arquitectura de componentes de Docker permite utilizar una imagen de contenedor como base para otros contenedores.

- Permite realizar una especie de NAT entre el equipo anfitrión y el contenedor, de tal forma que se pueden asignar puertos del anfitrión para que el contenedor pueda ser accedido desde los demás elementos de la red.
- Docker Hub es un servicio de repositorio que facilita el uso compartido de imágenes de docker de forma pública o privada, utilizaremos imágenes ya existentes (Ubuntu en este caso) para desarrollar nuestra imagen del servidor.
- Los contenedores pueden acceder en modo privilegiado a los recursos en un sistema (a la tarjeta gráfica NVIDIA en este trabajo).
- Posee una API por la que puede ser manejado por programas externos, lo cual nos facilitará la posterior gestión y mantenimiento

3.2 Nvidia-docker

Una vez habiéndonos decidido por Docker como el método utilizado para encapsular nuestros servidores VNC nos encontramos con un problema.
Los contenedores Docker son agnósticos a las plataformas, pero también al hardware. Esto plantea un problema cuando se utiliza hardware específico, como las GPU NVIDIA, que requieren módulos de núcleo y bibliotecas de nivel de usuario para funcionar. Por lo tanto, Docker no es compatible de forma nativa con las GPU NVIDIA dentro de los contenedores.

Para solucionar este problema se realiza una instalación completa de los controladores NVIDIA dentro del contenedor y el mapa en los dispositivos de caracteres correspondientes a las GPU NVIDIA en el momento del lanzamiento. Esta solución es frágil porque la versión del controlador host debe coincidir exactamente con la versión del controlador instalado en el contenedor. Este requisito redujo drásticamente la portabilidad de estos primeros contenedores, socavando una de las características más importantes de Docker.

Para facilitar la portabilidad de las imágenes Docker que aprovechan las GPU NVIDIA, se ha desarrollado nvidia-docker, un proyecto de código abierto alojado en Github que proporciona los dos componentes críticos necesarios para los contenedores portátiles basados en la GPU:

- imágenes CUDA (para cálculo en paralelo) agnósticas a los controladores
- un set de instrucciones de línea de comandos Docker que monta los componentes del modo de usuario del controlador y las GPU (dispositivos de caracteres) en el contenedor en el momento del lanzamiento.

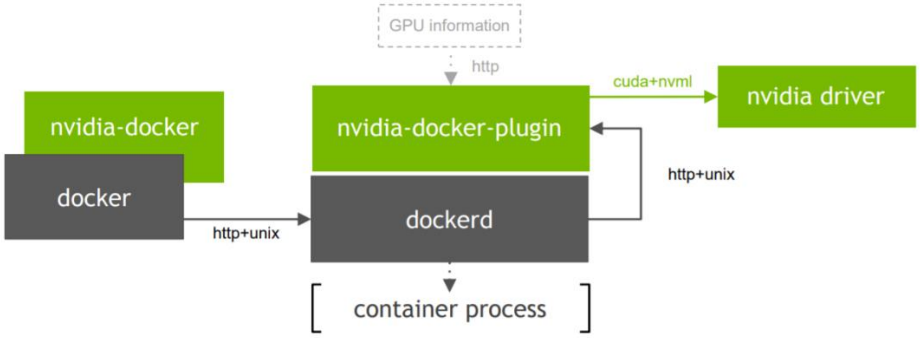

Figura 8 Esquema de funcionamiento de nvidia-docker

3.3 Portainer

Una vez que hubiéramos desarrollado el contenedor de Docker, había que buscar una solución *user-friendly* para el usuario que se encargue del manejo de los contenedores. Se llegó a la conclusión de que para la gestión y monitorización de nuestros contenedores Docker en uno o varios equipos anfitriones, usaremos una aplicación web llamada Portainer.

Portainer es una aplicación web open source, que permite gestionar de forma muy fácil sus contenedores Docker. Básicamente, permite hacer todo lo que permite el cliente docker, pero en una interfaz web bonita e intuitiva.

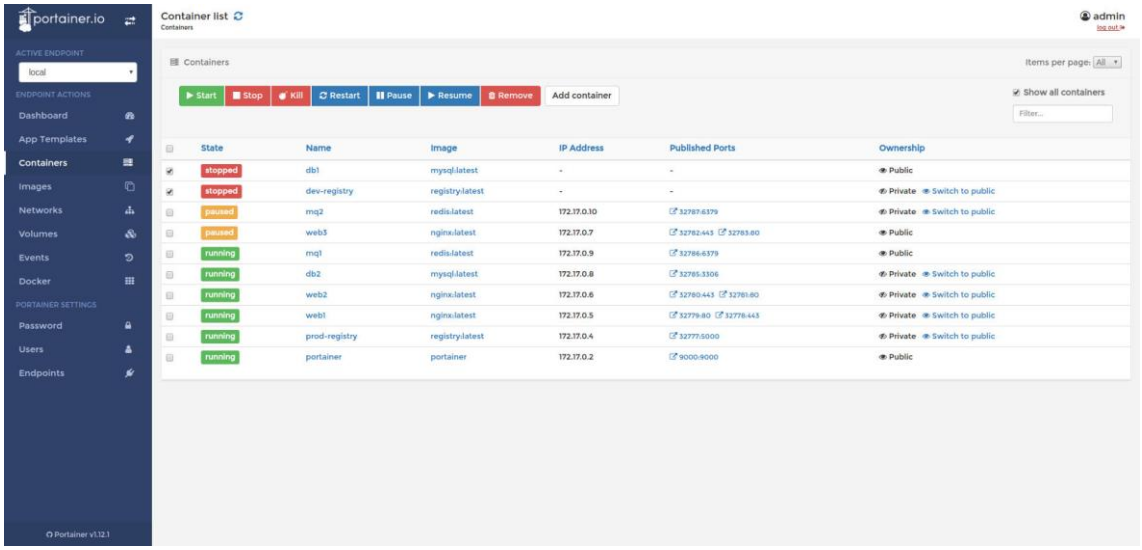

Figura 9 Ejemplo de listado de contenedores en Portainer

Capítulo 4. VNC

A la hora de determinar que protocolo usar para el transporte por red de nuestro escritorio remoto se llegó a la conclusión de que el único *open source* que podría cumplir los requisitos de rendimiento demandados era VNC aún a costa de un mayor consumo de ancho de banda en la red.

VNC[5] son las siglas en inglés de Virtual Network Computing. VNC es un programa de software libre basado en una estructura cliente-servidor que permite observar las acciones del ordenador servidor remotamente a través de un ordenador cliente. VNC utiliza el protocolo RFB para transmitir datos de píxeles de pantalla desde un equipo a otro a través de una red y enviar a cambio eventos de control.

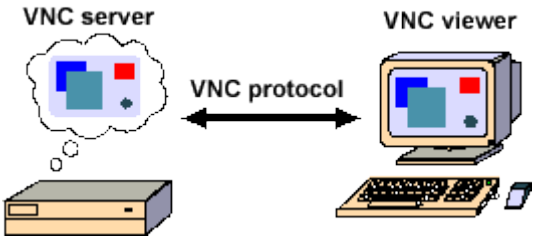

Figura 10 Componentes de un sistema VNC

El servidor VNC captura el escritorio del equipo en tiempo real y lo envía al visor VNC para su visualización. El visor VNC recopila su entrada (ratón, teclado o táctil) y la envía para que el servidor VNC la inserte y consiga de forma efectiva el control remoto.

VNC es independiente de la plataforma, un cliente VNC de un sistema operativo pueden conectarse a un servidor VNC del mismo sistema operativo o de cualquier otro. Hay clientes y servidores tanto para muchos sistemas operativos basados en GUI como para java. Varios clientes pueden conectarse a un servidor VNC al mismo tiempo. Los usos populares de esta tecnología incluyen ayuda técnica remota y acceso a los archivos presentes en el ordenador del trabajo desde la computadora de la casa o viceversa.

Un sistema de VNC (Fig. 10) se compone de un cliente, un servidor, y un protocolo de comunicación (RFB), los cuales exponemos a continuación.

4.1 Servidor VNC

El servidor VNC es el programa en el equipo que comparte su pantalla. El servidor de forma pasiva permite al cliente tomar el control de la misma.

Cuando ejecutamos el servidor de VNC, se crea un nuevo escritorio (nueva pantalla X) a la cual se puede acceder de forma remota con el cliente de VNC. Se pueden ejecutar tantos servidores VNC como permita la memoria del sistema, pudiendo varios usuarios acceder de forma simultánea, cada uno a su escritorio independiente, al contrario que la versión del servidor VNC para Windows que sólo permite acceder al escritorio principal.

4.1.1 X0vncserver

Este cliente forma parte de la plataforma TigerVNC, la cual se usará también como cliente. Consiste en un servidor VNC cuya característica principal es que replica una pantalla ya existente y la convierte al protocolo VNC. En nuestro caso hará una réplica de la pantalla virtual, con lo cual se conseguirán reproducir las interfaces gráficas en los equipos clientes a pesar de que el servidor no posea una pantalla física. Por motivos de seguridad limitaremos el número de usuarios simultáneos a uno para cada uno de los servidores.

4.2 Cliente VNC

El cliente VNC es el programa que controla e interactúa con el servidor. Básicamente el cliente VNC es el punto final remoto donde se sienta el usuario (es decir, la pantalla más el teclado y/o el puntero).

El énfasis en el diseño del protocolo RFB es hacer que necesite muy pocos requerimientos el cliente. De esta manera, los clientes pueden ejecutar en la más amplia gama de hardware, y la tarea de implementar un cliente se hace lo más simple posible.

El protocolo también convierte al cliente en *stateless*[6]. Si un cliente se desconecta de un servidor determinado y posteriormente se vuelve a conectar a ese mismo servidor, se conserva el estado de la interfaz de usuario. Además, se puede utilizar un punto final de cliente diferente para conectarse al mismo servidor RFB. En el nuevo punto final, el usuario verá exactamente la misma interfaz gráfica de usuario que en el punto final original. En efecto, la interfaz a las aplicaciones del usuario se vuelve completamente móvil. Dondequiera que exista una conectividad de red adecuada, el usuario puede acceder a sus propias aplicaciones personales, y el estado de estas aplicaciones se mantiene entre accesos desde diferentes ubicaciones. Esto proporciona al usuario una visión familiar y uniforme de la infraestructura informática allá donde vaya.

4.2.1 TigerVNC

TigerVNC[7] es una implementación de alto rendimiento, plataforma-neutral de VNC (Virtual Network Computing), sus componentes cliente/servidor permiten que los usuarios lancen e interactúen con interfaces gráficas en máquinas remotas. TigerVNC proporciona los niveles de funcionamiento necesarios para que funcionen características como 3D y video e intenta mantener un *look and feel* y reutiliza componentes, donde sea posible, en las distintas plataformas que soporta. TigerVNC también proporciona extensiones para métodos avanzados de autentificación y encriptación usando TLS (transport Layer Security).

Figura 11 Logotipo de TigerVNC

Estas son las características necesarias en cuanto a términos de rendimiento y seguridad que se plantearon en los requisitos iniciales, es por ello que nos decantamos por este cliente que además es multiplataforma e, incluso, se ha desarrollado una versión Java por lo que un amplio abanico de configuraciones y hardware distintos pueden ser usados con este cliente.

TigerVNC posee algunas opciones para reducir el ancho de banda que consume, cualidad que nos resulta interesante en el desarrollo de nuestro trabajo

Como podemos observar en la siguiente ilustración (Fig. 12), el cliente TigerVNC puede ajustar la profundidad en bits del color para obtener así un consumo significantemente menor de ancho de banda. Sin embargo, OpenGL no puede funcionar a una profundidad de bits menor que 24 por lo que no podremos reducir el ancho de banda por este método.

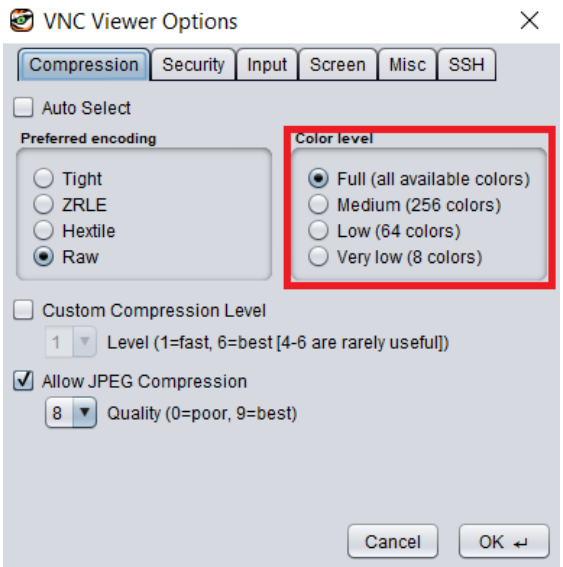

Figura 12 Opciones de profundidad de bits del color en TigerVNC

Por otra parte permite las compresiones Zlib y JPEG (Fig. 13). La primera reduce el ancho de banda a cambio de un uso más intensivo del procesador que, en nuestro caso, no resulta ningún problema; la segunda reduce la calidad de imagen recibida, lo cual puede llevar a una mala experiencia de usuario por lo que se decide no usar esta opción

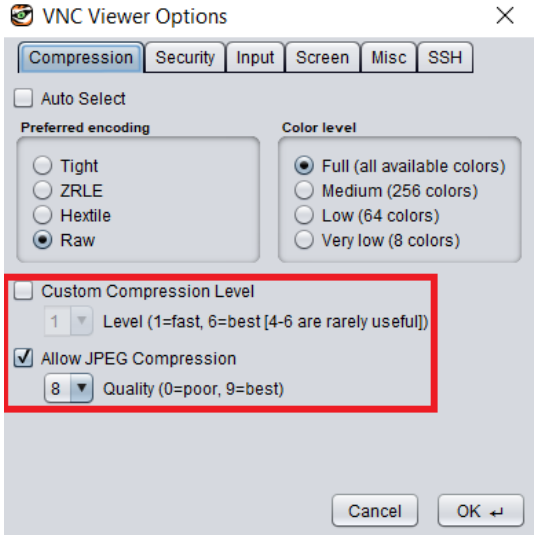

Figura 13 Opciones de compresión en TigerVNC

4.2.2 noVNC

En ocasiones no se puede acceder a un ordenador con una aplicación cliente instalada en él, para resolver esta posibilidad se desarrolla otra solución utilizando noVNC como cliente.

NoVNC[8] es un cliente VNC escrito completamente en HTML5 de manera que nos permite conectarnos a un servidor remoto sin necesidad de instalar ningún software adicional en nuestro ordenador, simplemente utilizando un navegador web que respete los estándares actuales (Google Chrome o Firefox, por ejemplo).

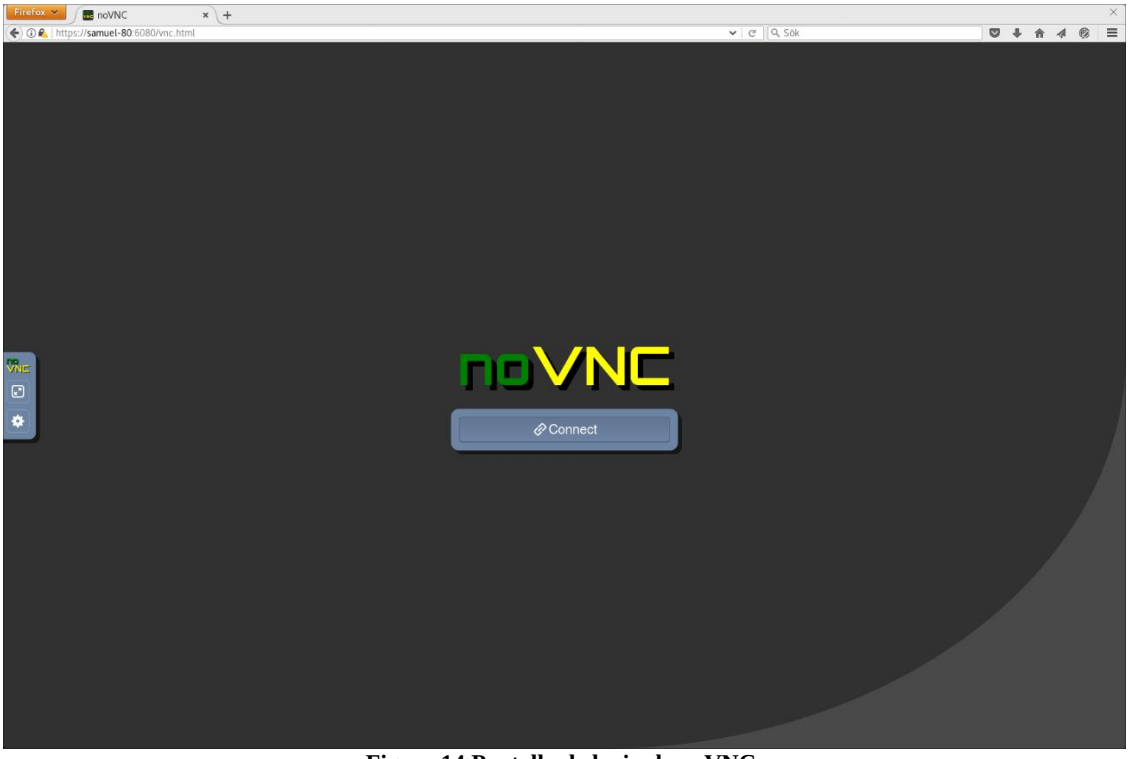

Figura 14 Pantalla de login de noVNC

Para ser capaz de convertir los datos de un servidor VNC, el cliente noVNC hace uso de la tecnología Websocket[8].

WebSocket es una tecnología que proporciona un canal de comunicación bidireccional y full-duplex sobre un único socket TCP. Está diseñada para ser implementada en navegadores y servidores web, pero puede utilizarse por cualquier aplicación cliente/servidor.

El uso de esta tecnología proporciona una funcionalidad similar a la apertura de varias conexiones en distintos puertos, pero multiplexando diferentes servicios WebSocket sobre un único puerto TCP.

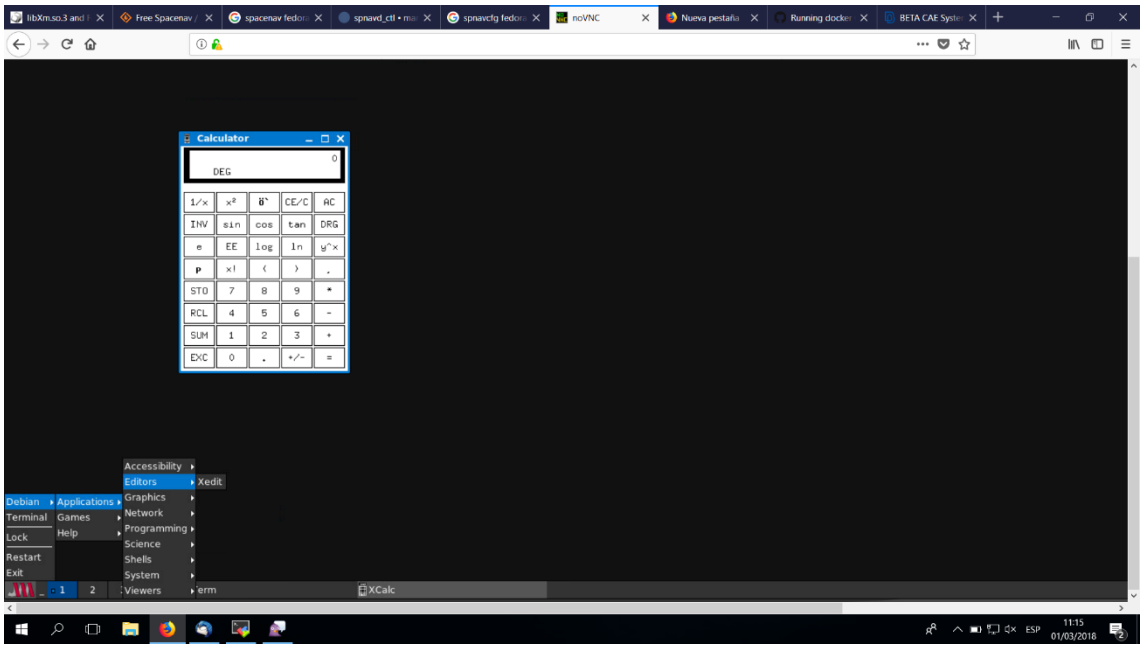

Figura 15 Escritorio remoto en noVNC

4.3 Protocolo RFB

El protocolo VNC (RFB o *Remote Framebuffer Protocol*) es muy simple, basado en una primitiva gráfica del servidor al cliente (("Put a rectangle of pixel data at the specified X,Y position", en español "Póngase un rectángulo de datos de píxel en la posición X,Y especificada) y mensajes de eventos desde el cliente al servidor.

El servidor envía pequeños rectángulos del *framebuffer* para el cliente. En su forma más simple, el protocolo VNC puede utilizar una gran cantidad de ancho de banda, por lo que han sido diseñados varios métodos para reducir la sobrecarga de comunicación. VNC por defecto usa el puerto TCP 5900+N[6], donde N es el número de la pantalla (por lo general: 0 para una pantalla física). Varias implementaciones también inician un servidor básico HTTP para proporcionar un visor VNC, que permite la conexión a través de cualquier navegador web.

Téngase en cuenta que la máquina donde se ejecuta el servidor VNC no necesita tener una pantalla física. Además, la pantalla que muestra VNC no es necesariamente la misma pantalla vista por un usuario en el servidor. En computadores Unix/Linux que soporten múltiples sesiones simultáneas X11, VNC puede ser configurado para servir a una sesión particular existente de X, o para iniciar una propia. También es posible ejecutar múltiples sesiones de VNC desde el mismo ordenador.

En Microsoft Windows la sesión VNC servida (proporcionada) es siempre la sesión del usuario actual.

4.3.1 Establecimiento de la conexión

El inicio de la comunicación entre servidor y cliente es algo realmente simple cuando hablamos de este protocolo. Los pasos pueden verse en la siguiente figura (Fig. 16):

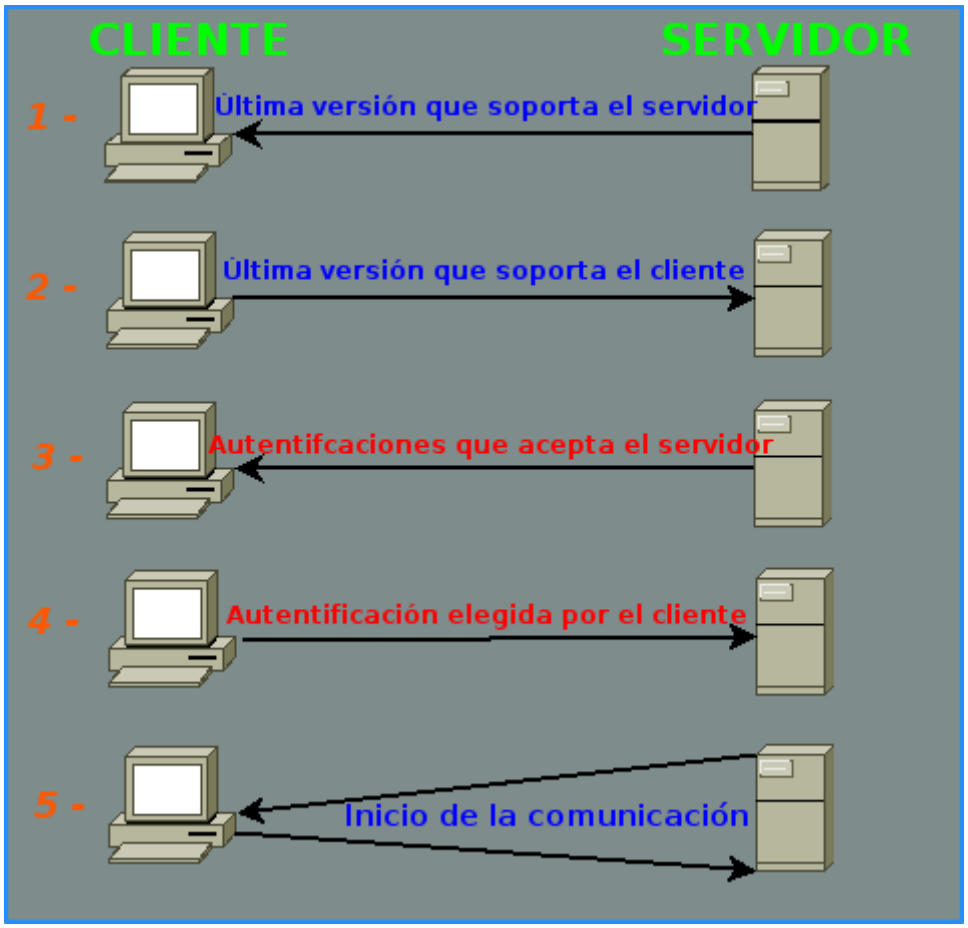

Figura 16 Establecimiento de conexión VNC

Consta de tres partes:

Primer Fase (Negociación del protocolo)

Cuando el cliente conecta al servidor, este le envía una cadena de 12 bytes con el último protocolo que soporta.

No. de bytes	Valor
12	"RFB 003.008\n" (hex 52 46 42 20 30 30 33 2e 30 30 38 0a)

Tabla 1 Valor hexadecimal de la versión del protocolo RFB

El cliente responde con la última versión de protocolo soportada y eligen usar la versión más baja. En nuestro caso tanto el cliente como el servidor soportan la versión de RFB 3.8 por lo que esa será la utilizada.

Segunda Fase (Negociación de la autenticación)

Una vez se tiene claro el protocolo se empieza la fase de negociación de autenticación. La secuencia de datos que se intercambia varía según la versión de protocolo que se utiliza.

En esta fase el servidor envía una serie de bytes con la siguiente estructura:

- 1 byte diciendo el número de autenticaciones que soporta.
- N bytes como array de esas autenticaciones.

Tercera Fase (Comunicación Bidireccional)

Ahora que ya se han autenticado correctamente, puede empezar la conexión al escritorio remoto. Se nos abrirá una ventana con el otro sistema y podremos ver todo lo que hace el que esté usando dicha máquina. Al mismo tiempo se sincronizan ratón y teclado con lo que si movemos el ratón o escribimos algo, aparecerá reflejado en el host servidor. Dicho de otra manera, estamos compartiendo el sistema e incluso la sesión.

4.4 Seguridad

Como se ha indicado en el apartado anterior el cliente y el servidor negocian la autentificación, es entonces cuando observamos en el listado de opciones que soporta TigerVNC para ver si alguna se ajusta a nuestras necesidades de privacidad.

El cliente y el servidor soportan varias opciones de autentificación, necesarias para dotar de seguridad a los datos transportados por red.

Código	Proveedor	Firma	Descripción
1	"STDV"	"NOAUTH__"	None
2	"STDV"	"VNCAUTH_"	VNC Authentication
19	"VENC"	"VENCRYPT"	VeNCrypt Security
20	"GTKV"	"SASL____"	Simple Authentication and Security Layer (SASL)
129	"TGHT"	"ULGNAUTH"	Unix Login Authentication
130	"TGHT"	"XTRNAUTH"	External Authentication

Tabla 2 Opciones de autentificación en VNC

Sin embargo, algunas de ellas poseen algunas vulnerabilidades que las hacen desaconsejables para nuestro propósito de proteger la conexión de extremo a extremo.

Los tipos de autentificación soportados son los siguientes:

- **None** El tráfico viaja sin cifrar y el servidor no solicita credenciales. Bajo esta premisa no puede ser esta la opción seleccionada de ningún modo.

- **VncAuth** Ninguna encriptación, autentificación estándar de VNC. Tiene el problema de que envía la contraseña en texto plano, de forma que un atacante

- **Plain** Sin encriptaciones, pregunta por nombre de usuario y contraseña de un usuario UNIX existente en el servidor.

- **TLSNone** Ninguna autenticación, los datos se encriptan usando GNUTLS. No se verifica la identidad del servidor. Es vulnerable a ataques del tipo *man-in-the-middle.*

- **TLSVnc** Autenticación estándar de VNC con la encriptación como TLSNone.

- **TLSPlain** Autenticación como Plain con cifrado como TLSNone. Está presente en los clientes, pero no está disponible el verificador de contraseñas.

- **X509None** Ninguna autenticación, los datos son encriptados usando GNUTLS. La identidad del servidor se verifica utilizando certificados X509. La clave y los certificados se especifican utilizando los parámetros de línea de comandos x509cert y x509key en el servidor. El visor de Unix y Windows acepta el parámetro de línea de comandos x509ca y x509crl para especificar opcionalmente el certificado de la CA a verificar y el fichero de revocación de certificados CRL, donde se añaden los certificados que hayan sido comprometidos.

- **X509Plain** Autenticación sencilla como Plain con cifrado como X509None. Está presente en los clientes, pero no está disponible el verficador de contraseñas.

En nuestro caso y dado que las credenciales se solicitan una vez conectado al servidor VNC optamos por la solución de autentificación **X509None**, la cual nos proporciona confidencialidad entre extremos debido al cifrado y autentificación del servidor ya que generamos nuestros propios certificados.

4.4.1 Certificados .X509

Uno de los requisitos del proyecto es que el tráfico entre cliente y servidor sea seguro. Para ello se usaran certificados .X509 en principio autofirmados, lo cual es suficiente para autentificar el servidor VNC. Estos certificados son los que usa TLS/SSL (Secure Socket Layer) que es una tecnología de seguridad estándar global que permite la comunicación cifrada entre un navegador web y un servidor web. Es utilizado por millones de empresas e individuos para reducir el riesgo de que información confidencial (por ejemplo, números de tarjetas de crédito, nombres de usuario, contraseñas, correos electrónicos, etc.) sea robada o manipulada por hackers y ladrones de identidad. En esencia, SSL permite una "conversación" privada sólo entre las dos partes.

Para crear esta conexión segura, se instala un certificado .X509 (también llamado "certificado digital") en un servidor y tiene dos funciones:

- Autentifica la identidad del servidor (esto garantiza a los clientes que no están en un servidor falso).
- Encripta los datos que se transmiten.

Por lo tanto, para asegurarnos que el tráfico entre cliente y servidor está cifrado y que el servidor es genuino, nuestro cliente habrá de contar con dos archivos (Fig. 17):

- **Certificado:** El cliente usará el mismo certificado que se encuentra en el servidor de este modo sabemos que el servidor es auténtico y se usará para descifrar el tráfico que le llegue.
- **CRL (Certificate Revocation List):** El CRL es un listado de certificados que han sido comprometidos y que por lo tanto deben dejar de ser utilizados. Este listado debería estar siempre actualizado ya que con un certificado que haya sido sustraído se podría desarrollar un servidor falso para robar información del usuario, siendo un problema de seguridad bastante grave.

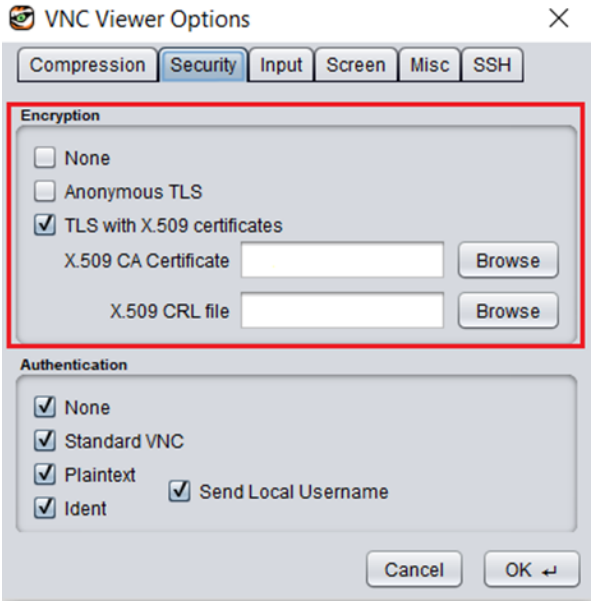

Figura 17 Opciones de cifrado en TigerVNC

4.5 Codificaciones

En definitiva, VNC consiste básicamente en enviar al cliente a través de la red el valor de todos y cada uno de los píxeles que queremos representar. Aunque hoy en día las redes han evolucionado lo suficiente para que esto no resulte un problema, no siempre ha sido así, es por lo que existe una cantidad muy elevada de distintas codificaciones[9] para indicar el valor de cada uno de los píxeles al cliente a cambio de perder calidad en la representación de la imagen.

Código	Proveedor	Firma	Descripción
0	"STDV"	"RAW_____"	Raw Encoding
1	"STDV"	"COPYRECT"	CopyRect Encoding
2	"STDV"	"RRE_____"	RRE Encoding
4	"STDV"	"CORRE____"	CoRRE Encoding
5	"STDV"	"HEXTILE_"	Hextile Encoding
6	"TRDV"	"ZLIB____"	ZLib Encoding
7	"TGHT"	"TIGHT___"	Tight Encoding
8	"TRDV"	"ZLIBHEX_"	ZLibHex Encoding
-32	"TGHT"	"JPEGQLVL"	JPEG Quality Level Pseudo-encoding
-223	"TGHT"	"NEWFBSIZ"	DesktopSize Pseudo-encoding (New FB Size)
-224	"TGHT"	"LASTRECT"	LastRect Pseudo-encoding
-232	"TGHT"	"POINTPOS"	Pointer Position
-239	"TGHT"	"RCHCURSR"	Cursor Pseudo-encoding (Rich Cursor)
-240	"TGHT"	"X11CURSR"	X Cursor Pseudo-encoding
-256	"TGHT"	"COMPRLVL"	Compression Level Pseudo-encoding
-305	"GGI_"	"GII_____"	gii Pseudo-encoding
-512	"TRBO"	"FINEQLVL"	JPEG Fine-Grained Quality Level Pseudo-encoding
-768	"TRBO"	"SSAMPLVL"	JPEG Subsampling Level Pseudo-encoding

Tabla 3 Opciones de codificación en VNC

En nuestro caso vamos a exponer únicamente las codificaciones que soporta TigerVNC (Fig. 18), dado que serán con las que hagamos las pruebas de consumo de ancho de banda.

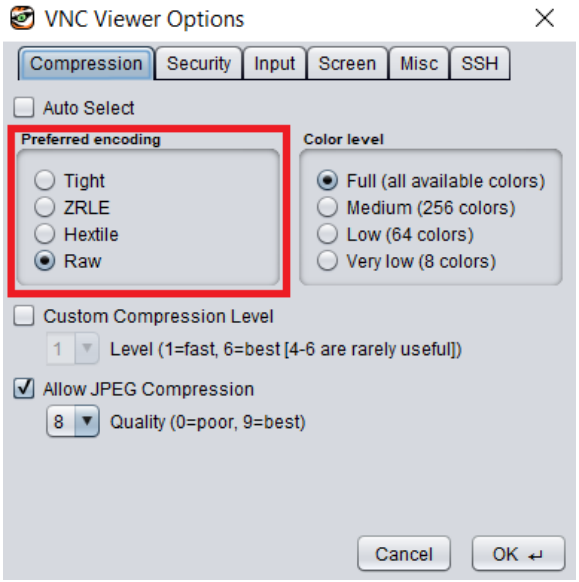

Figura 18 Opciones de codificación en TigerVNC

4.5.1 Codificación Raw

El tipo de codificación más simple son los datos de píxeles sin procesar. En este caso, los datos consisten en valores de píxeles de anchura*altura (donde anchura y altura son los valores de anchura y altura del rectángulo). Los valores simplemente representan el orden de líneas de escaneo de izquierda a derecha. Todos los clientes de RFB deben de poder manejar datos de píxeles en esta codificación sin procesar, y los servidores RFB deberían producir solamente la codificación Raw a menos que el cliente pida específicamente algún otro tipo de codificación.

Nº de bytes	Tipo [Valor]	Descripción
Anchura*altura*BytesPorPixel	Array de PÍXELES	píxeles

Tabla 4 Codificación Raw

4.5.2 Codificación Hextile

Hextile (Hexágono) es una variación de RRE (rise-and-run-length encoding) . Los rectángulos se dividen en 16x16 baldosas, permitiendo especificar las dimensiones de los subrectángulos en 4 bits cada uno, 16 bits en total. El rectángulo está dividido en baldosas comenzando por arriba a la izquierda y yendo de izquierda a derecha, de arriba abajo.

Los contenidos codificados de las baldosas simplemente se suceden en orden predeterminado. Si la anchura de todo el rectángulo no es un múltiplo exacto de 16, el ancho de la última baldosa de cada fila será proporcionalmente menor. Del mismo modo, si la altura del rectángulo entero no es un múltiplo exacto de 16, entonces la altura de cada baldosa en la última fila también será más pequeña.

Cada baldosa se codifica como datos Raw de pixel, o como variación sobre RRE. Cada baldosa tiene un valor de píxel de fondo. No es necesario especificar explícitamente el valor del píxel de fondo para cada una si es el mismo fondo que la anterior. Si todos los subrectángulos de una baldosa tienen el mismo valor de píxel, se puede especificar una vez el valor de píxel para toda la baldosa. Al igual que con el fondo, el valor del píxel del primer plano puede dejarse sin especificar, lo que significa que es transportado de la baldosa anterior. El valor del píxel del primer plano puede no ser igual si la baldosa anterior estaba codificada en Raw o tenía el bit SubrectsColored activado. Puede, sin embargo, ser trasladado de una baldosa anterior con el bit AnySubrects.

Los datos consisten en cada baldosa codificada en orden. Cada baldosa comienza con un byte de tipo subcodificación, que es una máscara compuesta por un número de bits:

Nº de bytes	Tipo [Valor]	Descripción
1	U8	Máscara de subcodificación:
	[1]	Raw
	[2]	BackgroundSpecified
	[4]	ForegroundSpecified
	[8]	AnySubrects
	[16]	SubrectsColored

Tabla 5 Byte de subcodificación

Si se activa el bit Raw, después los otros bits son irrelevantes. Se usan los valores del pixel de la anchura*altura (donde la anchura y la altura son las anchura y altura de la baldosa). De lo contrario, los otros bits de la máscara son los siguientes:

BackgroundSpecified

Si se activa, le sigue un número de bytes necesarios para representar un pixel y especifica el color de fondo de esta baldosa. La primera baldosa no codificada en Raw en una fila debe tener este bit activo. Si este bit no está configurado, entonces el color del fondo es el mismo que la de la última baldosa.

ForegroundSpecified

Si se activa, le sigue un número de bytes necesarios para representar un pixel y especifica el color de primer plano que se usará para todos los subrectángulos en esta baldosa.

Si este bit está activado, el bit SubrectsColored debe ser cero.

AnySubrects

Si está configurado, un solo byte sigue y da el número de subrectángulos que siguen. Si no está configurado, no hay subrectángulos(es decir, toda la baldosa es sólo del color de fondo).

SubrectsColored

Si se activa, entonces cada subrectángulo es precedido por un valor de píxel que da el color de ese subrectángulo, así que un subrectángulo es:

Nº de bytes	Tipo [Valor]	Descripción
BytesPorPíxel	PIXEL	Valor de píxel de subrectángulo
1	U8	Posición X e Y
1	U8	Altura y anchura

Tabla 6 Subrectángulo con bit SubrectColored activo

Si no está configurado, todos los subrectángulos son del mismo color (el color de primer plano). Si el bit ForegroundSpecified no estaba configurado, entonces el color del primer plano es el mismo que el de la última baldosa. Un subrectángulo es:

Nº de bytes	Tipo [Valor]	Descripción
1	U8	Posición X e Y
1	U8	Altura y anchura

Tabla 7 Subrectángulo con bit SubrectColored inactivo

La posición y el tamaño de cada subrectángulo se especifican en dos bytes, posición x-e-y y ancho-y-alto. Los 4 bits más significativos de la posición x e y especifique la posición X, la menos significativa especifique la posición Y. Los 4 bits más significativos de ancho y altura especifican el ancho menos 1, los menos significativos especifican la altura menos 1.

4.5.3 ZRLE

ZRLE significa Zlib Run-Length Encoding y es una codificación con compresión zlib. El rectángulo comienza con un valor de longitud de 4 bytes y es seguido por ese número de bytes de datos comprimidos en zlib. Se utiliza un único "stream" zlib para un protocolo RFB determinado de modo que los rectángulos ZRLE deben ser codificados y decodificados estrictamente en orden.

Nº de bytes	Tipo [Valor]	Descripción
4	U32	Longitud
Longitud	Array U8	zlibData

Tabla 8 Codificación ZRLE

Los datos zlibData cuando no están comprimidos representan las baldosas de izquierda a derecha, de arriba abajo con un tamaño de baldosa de 64x64 píxeles. Si el ancho del rectángulo no es un múltiplo exacto de 64, entonces el ancho de la última baldosa en cada fila es menor, y si la altura del rectángulo no es un múltiplo exacto de 64, entonces el parámetro de la altura de cada baldosa en la última fila es menor.

4.5.4 Codificación Tight

La codificación tight (o apretada) proporciona la compresión eficiente para los datos de pixel. Para reducir la complejidad de la implementación, el ancho de cualquier rectángulo codificado apretado no puede exceder los 2048 píxeles. Si se desea un rectángulo más ancho, debe dividirse en varios rectángulos y cada uno debe codificarse por separado.

El primer byte de cada rectángulo fuertemente codificado es un byte de control de compresión y los cuatro bits menos significativos del byte de control de compresión informan al cliente qué flujos de compresión zlib deben ser reajustados antes de decodificar el rectángulo. Cada bit es independiente y corresponde a un flujo zlib separado que debe ser reseteado:

Nº de bytes	Tipo [Valor]	Descripción
1	U8	Control de compresiónión:
	[1]	Reset stream 0
	[2]	Reset stream 1
	[4]	Reset stream 2
	[8]	Reset stream 3

Tabla 9 Primer byte del rectángulo

Uno de los tres métodos de compresión posibles está soportado en la codificación Tight. Estos son BasicCompression, FillCompression y JpegCompression. Si el bit 7 (el bit más significativo) del byte de control de compresión es 0, entonces el tipo de compresión es BasicCompression. En ese caso, los bits 7-4 (los cuatro bits más significativos) del control de compresión deben interpretarse como sigue:

Bits	Valor binario	Descripción
5-4	00	Usar stream 0
	01	Usar stream 1
	10	Usar stream 2
	11	Usar stream 3
6	0	-
	1	read-filter-id
7	0	BasicCompression

Tabla 10 Tipo de compresión BasicCompression

De lo contrario, si el bit 7 del control de compresión está ajustado a 1, entonces el método de compresión es FillCompression o JpegCompression, dependiendo de otros bits del mismo byte:

Bits	Valor binario	Descripción
7-4	1000	FillCompression
	1001	JpegCompression

Tabla 11 Métodos de compresión alternativos

La codificación Tight utiliza un nuevo tipo TPIXEL (Tight pixel). Esto es lo mismo que un PIXEL para el formato de píxeles acordado, el número de bits por píxel es 32, la profundidad es 24 y todos los bits que componen las intensidades roja, verde y azul son exactamente 8 bits de ancho. En este caso un TPIXEL tiene sólo 3 bytes de longitud, donde el primer byte es el componente rojo, el segundo byte es el componente verde y el tercer byte es el componente azul del valor del color del píxel.

Los datos que siguen al byte de control de compresión dependen del método de compresión.

FillCompression

Si el tipo de compresión es FillCompression, entonces el único valor de píxel sigue, en formato TPIXEL. Este valor se aplica a todos los píxeles del rectángulo.

JpegCompression

Si el tipo de compresión es JpegCompression, el siguiente flujo de datos se ve así:

Nº de bytes	Tipo [Valor]	Descripción
1-3		Longitud
Longitud	Array U8	zlibData

Tabla 12 Tipo de compresión JpegCompression

La longitud se codifica en una forma compacta de 1, 2 o 3 bits.

BasicCompression

Si el tipo de compresión es BasicCompression y el bit 6 (el bit read-filter-id) del byte de control de compresión se estableció en 1, entonces el siguiente (segundo) byte especifica el filter-id que le indica al decodificador qué tipo de filtro utilizó el codificador para *preprocesar* los datos de píxeles antes de la compresión. El byte de identificación del filtro puede ser uno de los siguientes:

Nº de bytes	Tipo [Valor]	Descripción
1	U8	filter-id
	[1]	CopyFilter
	[2]	PaletteFilter
	[4]	GradientFilter

Tabla 13 Tipos de filtro

CopyFilter

Cuando el CopyFilter está activo, los valores de píxeles sin procesar en formato TPIXEL se comprimen.

PaletteFilter

El PaletteFilter convierte datos de píxeles de color verdadero a colores indexados y una paleta que puede constar de 2..256 colores. Si el número de colores es 2, cada píxel se codifica en 1 bit, de lo contrario se utilizan 8 bits para codificar un píxel. La codificación de 1 bit se realiza de tal manera que los bits más significativos corresponden a los píxeles más a la izquierda, y cada fila de píxeles está alineada con el límite del byte. Cuando se utiliza el PaletteFilter, la paleta se envía antes de los datos de píxeles. La paleta comienza con un byte sin signo cuyo valor es el número de colores de la paleta menos 1 (es decir, 1 significa 2 colores, 255 significa 256 colores en la paleta). Luego sigue la paleta misma que consiste en valores de píxeles en formato TPIXEL.

GradientFilter

El GradientFilter procesa previamente los datos de los píxeles con un simple algoritmo que convierte cada componente de color en una diferencia entre una intensidad "prevista" y la intensidad real. Esta técnica no afecta al tamaño de los datos sin comprimir, pero ayuda a comprimir mejor las imágenes de tipo fotográfico.

Después de filtrar los datos de píxeles con uno de los tres filtros anteriores, se comprimen con la biblioteca zlib. Pero si el tamaño de los datos después de aplicar el filtro pero antes de la compresión es inferior a 12, entonces los datos se envían tal cual, sin comprimir. Se pueden utilizar cuatro flujos zlib separados (0..3) y el decodificador debe leer el ID real del flujo desde el byte de control de compresión.

Capítulo 5. Proxy

Dado que nos interesa que los usuarios puedan desarrollar su labor sin importar el servidor VNC al que se encuentren conectados y que el número de usuarios conectados a los distintos servidores no es uniforme a lo largo del tiempo, es bastante útil plantearse utilizar un proxy (también conocido como agente) que determine la dirección final a la que encaminar una petición del cliente. De este modo, los usuarios sólo han de conocer la dirección del proxy sin importar cual de los servidores se encuentra sin usar y sin tener que conocer las direcciones finales de cada uno de los servidores disponibles. Esta característica se conoce como balanceo de carga

Para ello utilizaremos dos software de proxy distintos, HAProxy para el cliente TigerVNC (funcionado sobre TCP) y Nginx para el cliente noVNC (que funciona sobre HTTP).

5.1 HAProxy

HAProxy[10] viene de High Availability Proxy, es software libre de código abierto que ofrece un balanceo de carga y alta disponibilidad para aplicaciones basadas en TCP y HTTP. Este tipo de tecnología es imprescindible para los servidores con una alta carga de proceso o que generan un gran tráfico.

Como características que nos interesan para su aplicación, HAProxy soporta SSL nativamente, mantenimiento de conexión, protección DDoS y mucho más. Nos decantamos por este proxy porque permite la redirección de tráfico SSL sin que sea necesario analizarlo (Fig. 19), por lo que actúa simplemente redirigiendo la dirección IP y el puerto de destino, controlando además que servidores se encuentran libres, con esta opción el tráfico va cifrado durante todo su trayecto.

Además, proporciona un sistema de estadísticas a través de una interfaz web que nos ofrece información tal como transferencia de datos, número de conexiones, estado de los servidores, etc.

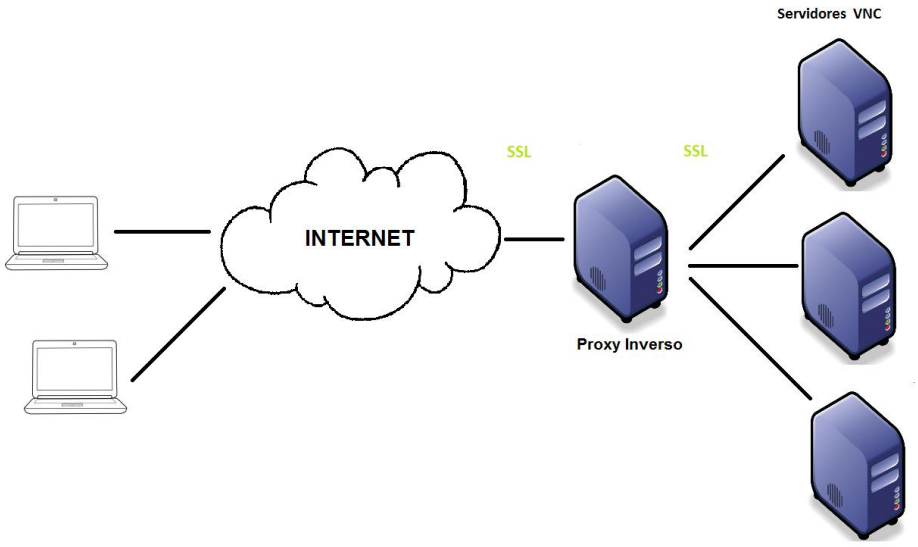

Figura 19 Esquema de funcionamiento de HAProxy con servidores VNC

5.2 Nginx

Para los clientes noVNC nos interesa implementar Nginx como proxy. Nginx es un servidor web de código abierto que, desde su éxito inicial como servidor web, ahora también es usado como proxy inverso, cache de HTTP, y balanceador de carga.

Figura 20 Logotipo de Nginx

La habilidad de actuar como proxy inverso HTTP es la que nos interesa de este software, actúa recuperando recursos web en nombre de un cliente desde uno o más servidores. Estos recursos son entonces devueltos al cliente como si se originaran en el propio servidor Web, esto es, el proxy se conecta a los servidores noVNC haciendo las peticiones que realiza el cliente y, al recibirlas del servidor, devuelve la respuesta al cliente que había hecho la solicitud.

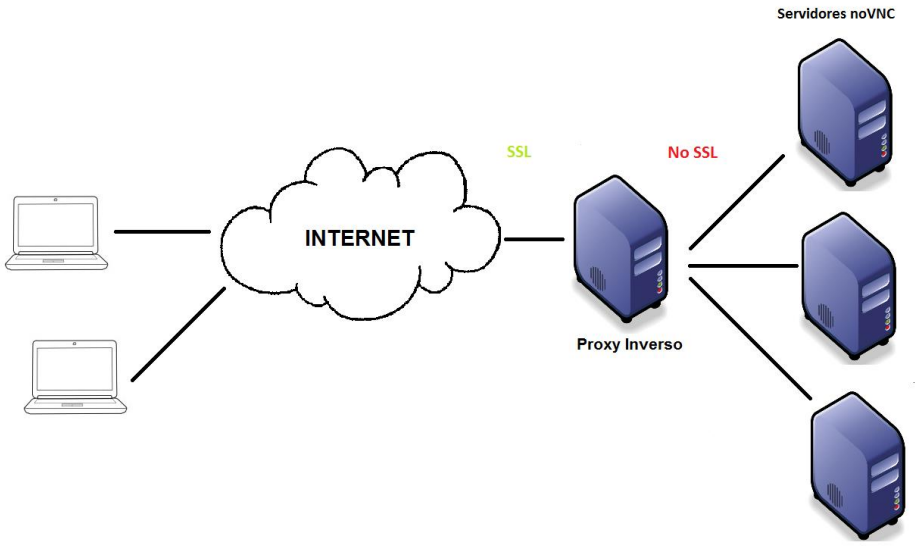

Figura 21 Esquema de funcionamiento de Nginx con servidores noVNC

En nuestro caso, también se encargará de gestionar el tráfico SSL con el cliente, mientras que en la comunicación con el servidor el tráfico será no SSL (Fig. 21), por lo que se recomienda su uso en el mismo equipo en el que se encuentra el servidor noVNC, para que no viaje tráfico sin cifrar por la red.

Capítulo 6. Despliegue de servidores

Con los elementos que hemos ido describiendo en los capítulos anteriores es como vamos a desarrollar nuestro trabajo, de tal forma que se divide en tres partes:

Primero hay que configurar el equipo anfitrión para que pueda ejecutar nuestros contenedores.

Acto seguido hay que construir la imagen de los contenedores y ejecutar el número deseado de ellos.

Para finalizar, se realiza la configuración de los proxys para redirigir

6.1 Configuración de equipo anfitrión

Partiendo de un equipo con Fedora como sistema operativo y con controlador NVIDIA ya instalado, necesitaremos que este sea capaz de ejecutar nuestros servidores.

Para ello necesitaremos que este equipo, que será el anfitrión de nuestros servidores, sea capaz de ejecutar instrucciones OpenGL y enviarlas rimero instalamos y activamos los componentes que hemos descrito en capítulos anteriores.

Instalamos Docker haciendo uso de los paquetes que se encuentran en los repositorios de Fedora, para ello utilizamos el gestor de paquetes **dnf**

```
dnf install docker
```

Activamos y establecemos su lanzamiento al arranque haciendo uso de los comandos de **systemd**

```
systemctl enable docker && systemctl start docker
```

Como nvidia-docker no se encuentra en los repositorios oficiales de Fedora, hemos de buscar e instalar un archivo .rpm, el tipo de archivos de instalación de Fedora. Por suerte, este tipo de archivos es de los más comunes en las distribuciones GNU/Linux y se encuentra en el repositorio git que sus desarrolladores tienen en Github. Habrá que descargarlo directamente de Internet haciendo uso de la herramienta **wget**, comúnmente usada para estos propósitos.

```
wget https://github.com/NVIDIA/nvidia-
docker/releases/download/v1.0.1/nvidia-docker-1.0.1-1.x86_64.rpm
```

Instalamos el paquete que hemos descargado haciendo uso de **rpm**, una herramienta de línea de comandos que sirve específicamente para ello.

```
rpm -ivh nvidia-docker-1.0.1-1.x86_64.rpm
```

Volvemos a activarlo y a establecer su inicio al arrancar el sistema operativo como hemos hecho anteriormente.

```
systemctl enable nvidia-docker && systemctl start nvidia-docker
```

Por último instalamos VirtualGL, encontrándose este también los repositorio oficiales y actuamos como anteriormente utilizando el comando dnf.

```
dnf install VirtualGL
```

Para poder utilizar VirtualGL hay Lanzamos la herramienta de configuración y responderemos que no a las preguntas que nos haga.

```
 vglserver_config

1) Configure server for use with VirtualGL
2) Unconfigure server for use with VirtualGL
X) Exit
Choose:
1
Restrict 3D X server access to vglusers group (recommended)?
[Y/n]
n
Restrict framebuffer device access to vglusers group (recommended)?
[Y/n]
n
Disable XTEST extension (recommended)?
[Y/n]
n
```

Hemos respondido que no a las preguntas dado que no vamos a añadir a los usuarios al grupo vglusers ya que esto evitará problemas a la hora de la administración de las cuentas de usuario por terceros, mientras que la extensión XTEST es necesaria para tratar con nuestra pantalla virtual.

Reiniciamos la máquina y ya la tendremos preparada para ejecutar nuestro software.

6.2 Despliegue de los contenedores

Una vez que nuestro equipo anfitrión se encuentra debidamente configurado, ha llegado el momento de desplegar nuestros contenedores de Docker. Para ello y según las distintas necesidades del usuario se han creado dos versiones principales con dos subversiones cada una (cuatro contenedores distintos en total).

- **VNC**: Versión a la que se acede mediante un cliente VNC remotamente
  - **vnc:** Solamente sirve para ejecutar el software CAE
  - **vnc_desk:** Ejecuta un gestor de ventanas desde el que se puede lanzar el software CAE y otro software de tipo ofimático (Firefox, Thunderbird, Libreoffice…)

- **noVNC**: Versión a la que se acede mediante un navegador web
  - **novnc:** Solamente sirve para ejecutar el software CAE
  - **novnc_desk:** Ejecuta un gestor de ventanas desde el que se puede lanzar el software CAE y otro software de tipo ofimático (Firefox, Thunderbird, Libreoffice…)

6.2.1 Versiones VNC

Los contenedores se describen mediante el uso de un archivo de configuración llamado Dockerfile. Estos Dockerfiles consisten básicamente en los pasos que hay que dar desde una instalación "limpia" de un sistema operativo.

En nuestro caso hemos optado por Ubuntu 17.10 como SO base debido a la mayor cantidad de software presente en sus repositorios, lo cual nos ofrece un amplio abanico de posibles configuraciones en muy pocos comandos.

Para la versión **sin escritorio**, el Dockerfile es el siguiente:

```
FROM ubuntu:17.10

ENV DEBIAN_FRONTEND noninteractive

RUN apt-get update
RUN apt-get install -y x-window-system
RUN apt-get install -y binutils
RUN apt-get install -y mesa-utils
RUN  apt-get update && apt-get install -y module-init-tools wget
supervisor tigervnc-scraping-server xvfb python-pip jwm expect yad
libjpeg62 libtspi1 xprintidle && \
  rm -f /usr/share/applications/x11vnc.desktop && \
  apt-get remove -y python-pip && \
  wget https://bootstrap.pypa.io/get-pip.py && \
  python get-pip.py && \
  pip install supervisor-stdout && \
  apt-get -y clean

ADD nvidia-driver.run /tmp/nvidia-driver.run
```

```
RUN sh /tmp/nvidia-driver.run -a -N --ui=none --no-kernel-module
RUN rm /tmp/nvidia-driver.run

RUN  wget
https://sourceforge.net/projects/virtualgl/files/2.5.2/virtualgl_2.5.2
_amd64.deb && \
    dpkg -i virtualgl_2.5.2_amd64.deb && rm virtualgl_2.5.2_amd64.deb

USER root

EXPOSE 5900

COPY etc /etc
COPY usr /usr
ENV DISPLAY :1
RUN rm /etc/localtime && ln -s /usr/share/zoneinfo/Europe/Berlin
/etc/localtime
WORKDIR /root

ENTRYPOINT ["/usr/bin/entrypoint.sh"]
```

Los pasos que se han dado son muy sencillos, consiste en la instalación del sistema de ventanas X, los drivers de NVIDIA, VirtualGL y los componentes necesarios para que funcione el servidor VNC[11] además de sus dependencias y la copia de una serie de archivos de configuración ya preestablecidos para que el contenedor pueda insertarse en el entorno de trabajo ya existente como si fueran un equipo independiente más.

Por otro lado, para la versión **con escritorio**:

```
FROM ubuntu:17.10

ENV DEBIAN_FRONTEND noninteractive

RUN apt-get update
RUN apt-get install -y x-window-system
RUN apt-get install -y binutils
RUN apt-get install -y mesa-utils
RUN apt-get update && apt-get install -y module-init-tools wget
supervisor tigervnc-scraping-server xvfb python-pip jwm expect yad
libjpeg62 libtspi1 terminator firefox thunderbird gimp libreoffice
gthumb evince dbus-x11 xfe parole xprintidle && \
  rm -f /usr/share/applications/x11vnc.desktop && \
  apt-get remove -y python-pip && \
  wget https://bootstrap.pypa.io/get-pip.py && \
  python get-pip.py && \
  pip install supervisor-stdout && \
  apt-get -y clean

ADD nvidia-driver.run /tmp/nvidia-driver.run
RUN sh /tmp/nvidia-driver.run -a -N --ui=none --no-kernel-module
RUN rm /tmp/nvidia-driver.run
```

```
RUN  wget
https://sourceforge.net/projects/virtualgl/files/2.5.2/virtualgl_2.5.2
_amd64.deb && \
    dpkg -i virtualgl_2.5.2_amd64.deb && rm virtualgl_2.5.2_amd64.deb

USER root

EXPOSE 5900

COPY etc /etc
COPY usr /usr
ENV DISPLAY :1
RUN mkdir /usr/tmp && chmod 777 /usr/tmp
RUN rm /etc/localtime && ln -s /usr/share/zoneinfo/Europe/Berlin
/etc/localtime
WORKDIR /root

ENTRYPOINT ["/usr/bin/entrypoint.sh"]
```

Es la misma configuración que en el caso anterior pero instalando además un gestor de ventanas muy ligero y diverso software ofimático (Fig. 22), por si el usuario desea ejecutar otros programas aparte del software dedicado a la ingeniería.

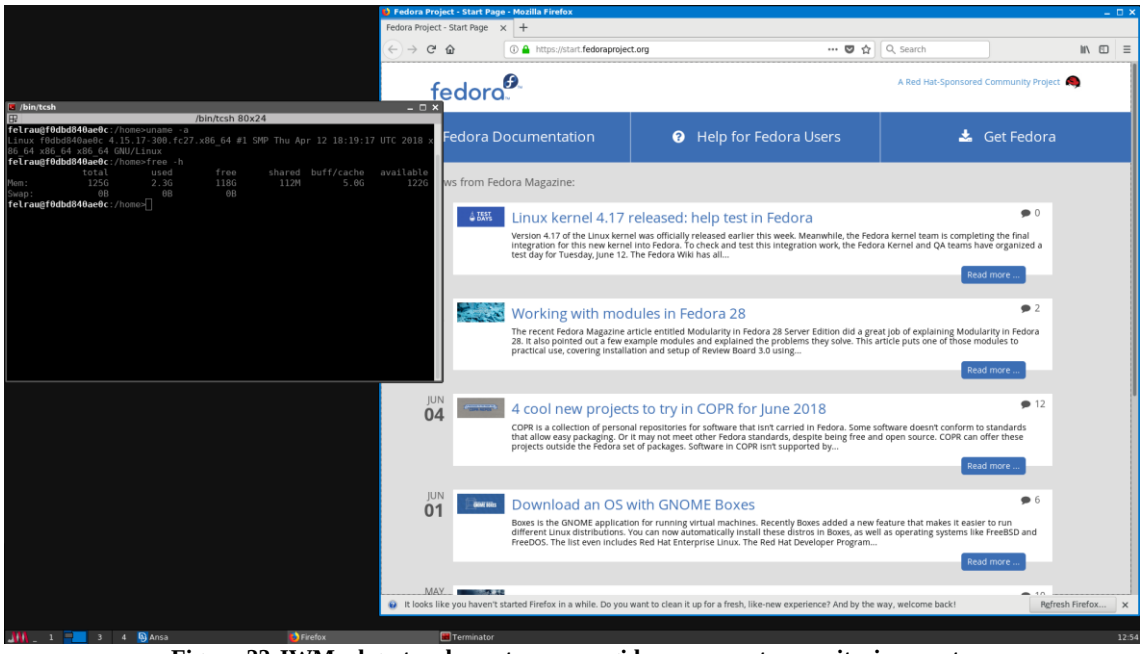

Figura 22 JWM, el gestor de ventanas escogido para nuestro escritorio remoto

Como se puede observar ambos Dockerfile acaban con la opción ENTRYPOINT, esta consiste en un script que deseamos que se ejecute al lanzar el contenedor, en nuestro caso para ambas versiones el script es el siguiente:

```
/usr/bin/entrypoint.sh

#!/bin/bash

while test $# -gt 0
do

case $1 in
      geometry)
      shift
      echo "---- Changing geometry to $1 ----"
      sed -i s/1920x1080/$1/ /etc/supervisor/conf.d/xvfb.conf
      shift
      ;;
      *)
      echo "Invalid argument $1"
      exit 1
      ;;
esac
done
echo "------ Starting ThinClient container ------"
supervisord -c /etc/supervisor/supervisord.conf
```

En pocas palabras, el script espera que se le introduzcan parámetros adicionales por si hay que cambiar alguna característica del contenedor, a modo de prueba se añadió la opción de cambiar la resolución de la pantalla virtual y por la forma en que está escrito, es fácilmente ampliable.

El script termina ejecutando un programa llamado **supervisor**[12], el cual sirve para lanzar una serie de comandos al arranque del contenedor.

```
/etc/supervisor/supervisord.conf

; supervisor config file

[unix_http_server]
file=/var/run/supervisor.sock   ; (the path to the socket file)
chmod=0700                       ; sockef file mode (default 0700)

[supervisord]
nodaemon=true
logfile=/var/log/supervisor/supervisord.log ; (main log file;default
$CWD/supervisord.log)
pidfile=/var/run/supervisord.pid ; (supervisord pidfile;default
supervisord.pid)
childlogdir=/var/log/supervisor            ; ('AUTO' child log dir,
default $TEMP)

; the below section must remain in the config file for RPC
; (supervisorctl/web interface) to work, additional interfaces may be
; added by defining them in separate rpcinterface: sections
[rpcinterface:supervisor]
supervisor.rpcinterface_factory =
supervisor.rpcinterface:make_main_rpcinterface
```

```
[supervisorctl]
serverurl=unix:///var/run/supervisor.sock ; use a unix:// URL  for a
unix socket

; The [include] section can just contain the "files" setting.  This
; setting can list multiple files (separated by whitespace or
; newlines).  It can also contain wildcards.  The filenames are
; interpreted as relative to this file.  Included files *cannot*
; include files themselves.

[include]
files = /etc/supervisor/conf.d/*.conf
```

Este es el archivo de configuración inicial, el cual indica que hay que añadir todos los archivos .conf que se encuentran en el directorio /etc/superviso/conf.d/, los cuales son los siguientes:

```
/etc/supervisor/conf.d/xvfb.conf

[program:xvfb]
command=bash -l -c "Xvfb :1 -screen 0 1920x1080x24 -ac"
user=root
environment=HOME=/root,USER=root
directory=/root
autorestart=true
stdout_logfile=/var/log/supervisor/xvfb.log
redirect_stderr=true
priority=10
```

Este archivo sirve para ejecutar el programa que nos proporciona la pantalla virtual, está configurada a una resolución de 1920x1080 píxeles con una profundidad de color de 24 bits, el equivalente a color "real".

```
/etc/supervisor/conf.d/x0vncserver.conf

[program:x0vncserver]
command=bash -l -c "x0vncserver -DisconnectClients=off -NeverShared=on
-securitytypes=X509none -display :1"
environment=HOME=/root ,USER=root
directory=/root
autorestart=true
stdout_logfile=/var/log/supervisor/x0vnc.log
redirect_stderr=true
priority=20
```

Este es el archivo que ejecuta el servidor VNC, está configurado con el tipo de seguridad X509none, el cual hace uso de certificados SSL para autentificar el servidor. También limita el número de clientes que se pueda conectar a uno, para que un segundo usuario no pueda observar ni modificar el trabajo que esté realizando el que se encuentre conectado. Por último, indica que la pantalla a replicar es la número :1, la que habíamos configurado como nuestra pantalla virtual.

```
/etc/supervisor/conf.d/xlogin.conf

[program:login]
command=bash -l -c "login.exp $(yad --title=Login --form --
field=Username --field=Password:H --center --separator=' ' --
image=/path/to/Logo.png)"
user=root
environment=HOME=/root,USER=root
directory=/root
autorestart=true
stdout_logfile=/var/log/supervisor/xlogin.log
redirect_stderr=true
priority=30
```

Este archivo ejecuta un script que sirve para que un usuario introduzca sus credenciales en una interfaz gráfica que es relativamente sencillo. Utiliza un intérprete de línea de comandos llamado **expect**, el cual sirve para automatizar comandos que requieren de la interacción del usuario.

El comando **login**, que sirve para autentificarse en sistemas operativos GNU/Linux es uno de estos comandos por motivos de seguridad y para que nuestra interfaz gráfica de autentificación funcione, es necesario hacerlo de este modo:

```
login.exp

#!/usr/bin/expect

set user [lindex $argv 0]
set password [lindex $argv 1]

spawn login $user

expect Password:
send "$password\n"
expect Quit.
send "exit\n"
interact
```

Este script espera que introduzcamos nuestras credenciales en el *prompt* gráfico (Fig. 23) que mostramos a continuación y le pasa el nombre de usuario y contraseña al comando login.

Figura 23 Interfaz gráfica para identificarse en el servidor

Finalmente, cambiamos la distribución de teclado a español de España con el siguiente archivo:

```
/etc/supervisor/conf.d/keyboard.conf

[program:keyboard]
command=bash -l -c "setxkbmap es"
user=root
environment=HOME=/root,USER=root
directory=/root
autorestart=true
stdout_logfile=/var/log/supervisor/keyboard.log
redirect_stderr=true
priority=110
```

6.2.2 Versiones noVNC

Las versiones noVNC son prácticamente iguales a las VNC, básicamente se añade el cliente noVNC a la hora de construir la imagen del contenedor.

Para la versión **sin escritorio**:

```
FROM ubuntu:17.10

ENV DEBIAN_FRONTEND noninteractive

RUN apt-get update
RUN apt-get install -y x-window-system
RUN apt-get install -y binutils
RUN apt-get install -y mesa-utils
RUN apt-get install -y module-init-tools wget supervisor tigervnc-
scraping-server xvfb python-pip tcsh rpcbind websockify novnc net-
tools jwm yad expect && \
  rm -f /usr/share/applications/x11vnc.desktop && \
  apt-get remove -y python-pip && \
  wget https://bootstrap.pypa.io/get-pip.py && \
  python get-pip.py && \
  pip install supervisor-stdout && \
  apt-get -y clean

ADD nvidia-driver.run /tmp/nvidia-driver.run
RUN sh /tmp/nvidia-driver.run -a -N --ui=none --no-kernel-module
RUN rm /tmp/nvidia-driver.run

RUN  wget
https://sourceforge.net/projects/virtualgl/files/2.5.2/virtualgl_2.5.2
_amd64.deb && \
    dpkg -i virtualgl_2.5.2_amd64.deb && rm virtualgl_2.5.2_amd64.deb

USER root

EXPOSE  6080
```

```
COPY etc /etc
COPY usr /usr

RUN cd /usr/share/novnc/ && ln -s vnc_auto.html index.html && cat
/etc/multivnc.key /etc/multivnc.pem > utils/self.pem

ENV DISPLAY :1
RUN rm /etc/localtime && ln -s /usr/share/zoneinfo/Europe/Berlin
/etc/localtime
WORKDIR /root

ENTRYPOINT ["/usr/bin/entrypoint.sh"]
```

En este caso obtenemos el cliente noVNC del repositorio oficial de Ubuntu, dado que esa versión funciona correctamente si sólo hemos de usar el software CAE.

Por otro lado para la versión con escritorio:

```
FROM ubuntu:17.10

ENV DEBIAN_FRONTEND noninteractive

RUN apt-get update
RUN apt-get install -y x-window-system
RUN apt-get install -y binutils
RUN apt-get install -y mesa-utils
RUN apt-get update && apt-get install -y git module-init-tools
terminator wget supervisor tigervnc-scraping-server xvfb python-pip
tcsh rpcbind websockify net-tools jwm yad expect firefox thunderbird
libreoffice evince tnef gimp gthumb xfe dbus-x11 parole xprintidle
libjpeg62 libtspi1 && \
  rm -f /usr/share/applications/x11vnc.desktop && \
  apt-get remove -y python-pip && \
  wget https://bootstrap.pypa.io/get-pip.py && \
  python get-pip.py && \
  pip install supervisor-stdout && \
   apt-get -y clean

ADD nvidia-driver.run /tmp/nvidia-driver.run
RUN sh /tmp/nvidia-driver.run -a -N --ui=none --no-kernel-module
RUN rm /tmp/nvidia-driver.run

RUN  wget
https://sourceforge.net/projects/virtualgl/files/2.5.2/virtualgl_2.5.2
_amd64.deb && \
    dpkg -i virtualgl_2.5.2_amd64.deb && rm virtualgl_2.5.2_amd64.deb

USER root

EXPOSE  6080

WORKDIR /root
```

```
RUN mkdir /usr/share/novnc && git clone
https://github.com/novnc/noVNC.git && cp -r noVNC/* /usr/share/novnc
&& rm -rf noVNC
COPY etc /etc
COPY usr /usr

RUN mkdir /usr/tmp && chmod 777 /usr/tmp
RUN rm /etc/localtime && ln -s /usr/share/zoneinfo/Europe/Berlin
/etc/localtime
RUN
cd /usr/share/novnc/ && ln -s vnc.html index.html && cat
/etc/multivnc.key /etc/multivnc.pem > utils/self.pem

ENV DISPLAY :1

ENTRYPOINT ["/usr/bin/entrypoint.sh"]
```

En esta versión hemos utilizado la versión del cliente noVNC que se encuentra en su repositorio oficial de Github dado que Linux malinterpreta alguna teclas como AltGr cuando el cliente usa Windows como sistema operativo, problema solucionado en esta versión.

Los archivos de ENTRYPOINT y de supervisor son los mismos que en las otras versiones con una salvedad, y es que el cliente noVNC también ha de ser lanzado desde supervisor:

```
/usr/bin/entrypoint.sh

#!/bin/bash

while test $# -gt 0
do

case $1 in
      geometry)
      shift
      echo "---- Changing geometry to $1 ----"
      sed -i s/1920x1080/$1/ /etc/supervisor/conf.d/xvfb.conf
      shift
      ;;
      *)
      echo "Invalid argument $1"
      exit 1
      ;;
esac
done
echo "------ Starting ThinClient container ------"
supervisord -c /etc/supervisor/supervisord.conf
```

A continuación se puede observar que los archivos de configuración son como los de las versiones VNC.

```
/etc/supervisor/supervisord.conf

; supervisor config file

[unix_http_server]
file=/var/run/supervisor.sock   ; (the path to the socket file)
chmod=0700                       ; sockef file mode (default 0700)

[supervisord]
nodaemon=true
logfile=/var/log/supervisor/supervisord.log ; (main log file;default
$CWD/supervisord.log)
pidfile=/var/run/supervisord.pid ; (supervisord pidfile;default
supervisord.pid)
childlogdir=/var/log/supervisor            ; ('AUTO' child log dir,
default $TEMP)

; the below section must remain in the config file for RPC
; (supervisorctl/web interface) to work, additional interfaces may be
; added by defining them in separate rpcinterface: sections
[rpcinterface:supervisor]
supervisor.rpcinterface_factory =
supervisor.rpcinterface:make_main_rpcinterface

[supervisorctl]
serverurl=unix:///var/run/supervisor.sock ; use a unix:// URL  for a
unix socket

; The [include] section can just contain the "files" setting.  This
; setting can list multiple files (separated by whitespace or
; newlines).  It can also contain wildcards.  The filenames are
; interpreted as relative to this file.  Included files *cannot*
; include files themselves.

[include]
files = /etc/supervisor/conf.d/*.conf
```

El archivo que ejecuta la pantalla virtual:

```
/etc/supervisor/conf.d/xvfb.conf

[program:xvfb]
command=bash -l -c "Xvfb :1 -screen 0 1920x1080x24 -ac"
user=root
environment=HOME=/root,USER=root
directory=/root
autorestart=true
stdout_logfile=/var/log/supervisor/xvfb.log
redirect_stderr=true
priority=10
```

El que ejecuta el servidor VNC:

```
/etc/supervisor/conf.d/x0vncserver.conf

[program:x0vncserver]
command=bash -l -c "x0vncserver -DisconnectClients=off -NeverShared=on
-securitytypes=X509none -display :1"
environment=HOME=/root ,USER=root
directory=/root
autorestart=true
stdout_logfile=/var/log/supervisor/x0vnc.log
redirect_stderr=true
priority=20
```

La interfaz gráfica para introducir las credenciales:

```
/etc/supervisor/conf.d/xlogin.conf

[program:login]
command=bash -l -c "login.exp $(yad --title=Login --form --
field=Username --field=Password:H --center --separator=' ' --
image=/path/to/Logo.png)"
user=root
environment=HOME=/root,USER=root
directory=/root
autorestart=true
stdout_logfile=/var/log/supervisor/xlogin.log
redirect_stderr=true
priority=30
```

El script para autentificarse también es el mismo:

```
login.exp

#!/usr/bin/expect

set user [lindex $argv 0]
set password [lindex $argv 1]

spawn login $user

expect Password:
send "$password\n"
expect Quit.
send "exit\n"
interact
```

Y el archivo de configuración de la distribución del teclado:

```
/etc/supervisor/conf.d/keyboard.conf

[program:keyboard]
command=bash -l -c "setxkbmap es"
user=root
environment=HOME=/root,USER=root
directory=/root
autorestart=true
stdout_logfile=/var/log/supervisor/keyboard.log
redirect_stderr=true
priority=110
```

El siguiente archivo de configuración es el único que difiere con respecto al desarrollo de las versiones VNC. En él se lanza un script que ejecuta el cliente noVNC en el puerto 6080.

```
/etc/supervisor/conf.d/novnc.conf

[program:noVNC]
command=bash -l -c "/usr/share/novnc/utils/launch.sh"
environment=USER=root
autorestart=true
stdout_logfile=/var/log/supervisor/noVNC.log
redirect_stderr=true
priority=100
```

6.2.3 Instalación

Para que se pueda utilizar la GPU en nuestro contenedor Docker se ha de instalar la misma versión del controlador NVIDIA que se halla en el ordenador anfitrión, para ello haremos uso de un script de línea de comandos bastante simple.

Para las versiones VNC:

```
#!/bin/sh

# Obtine la versión correspondiente del controlador, e.g. 340.24
nvidia_version=$(cat /proc/driver/nvidia/version | head -n 1 | awk '{
print $8 }')

# Hay que usar la misma versión en el contenedor y en el equipo
anfitrión

if test ! -f nvidia-driver.run; then
nvidia_driver_uri=http://us.download.nvidia.com/XFree86/Linux-
x86_64/${nvidia_version}/NVIDIA-Linux-x86_64-${nvidia_version}.run
wget -O nvidia-driver.run $nvidia_driver_uri
fi

docker build -t nvidia-opengl .
```

Para las versiones noVNC:

```
#!/bin/sh

# Obtine la versión correspondiente del controlador, e.g. 340.24
nvidia_version=$(cat /proc/driver/nvidia/version | head -n 1 | awk '{
print $8 }')

# Hay que usar la misma versión en el contenedor y en el equipo
anfitrión

if test ! -f nvidia-driver.run; then
nvidia_driver_uri=http://us.download.nvidia.com/XFree86/Linux-
x86_64/${nvidia_version}/NVIDIA-Linux-x86_64-${nvidia_version}.run
wget -O nvidia-driver.run $nvidia_driver_uri
fi

docker build -t nvidia-opengl-novnc .
```

Grosso modo, este script crea como variable el número de versión del controlador NVIDIA y se lo traslada a otra variable que consiste en la dirección de Internet del repositorio público de NVIDIA donde se encuentra un script que instalará esa versión en particular.

Por último crea la imagen del contenedor aprovechando que ya se posee una copia del script de instalación del controlador.

6.2.4 Ejecución

Una vez tenemos la imagen creada en nuestro ordenador anfitrión procedemos a ejecutarla teniendo que pasarle una serie de parámetros a la instrucción **docker run** para que se cumplan las siguientes condiciones:

- Ha de ejecutarse en modo privilegiado para poder tener acceso a la GPU.
- Ha de reiniciarse siempre, ya sea debido a un fallo o a un reinicio inesperado del equipo anfitrión.
- Han de declararse los distintos recursos (físicos y virtuales) de la GPU a los que puede tener acceso.
- Ha de exponerse en la interfaz de red del ordenador anfitrión el puerto correspondiente al servidor VNC del contenedor. El puerto origen (en el contenedor) y el puerto destino (en el anfitrión) no tienen porque ser el mismo, por lo tanto, en un equipo anfitrión se pueden exponer varios puertos correspondientes al mismo puerto en distintos contenedores.

Todas estas características se pueden conseguir a través del siguiente comando para las versiones VNC:

```
nvidia-docker run --privileged \
--device=/dev/nvidia0:/dev/nvidia0 \
--device=/dev/nvidiactl:/dev/nvidiactl \
--device=/dev/nvidia-uvm:/dev/nvidia-uvm \
```

```
-p 590X:5900 \
--restart=always \
nvidia-opengl
```

Y para las versiones noVNC:

```
nvidia-docker run --privileged \
--device=/dev/nvidia0:/dev/nvidia0 \
--device=/dev/nvidiactl:/dev/nvidiactl \
--device=/dev/nvidia-uvm:/dev/nvidia-uvm \
-p 608X:6080 \
--restart=always \
nvidia-opengl-novnc
```

Se puede corroborar que han sido correctamente generadas las imágenes con el siguiente comando (Fig. 24):

```
docker images

REPOSITORY           TAG                 IMAGE ID            CREATED             SIZE
nvidia-opengl-novnc  latest              317dc3e31ecb        8 minutes ago       2.37 GB
nvidia-opengl        latest              ab073692e4ca        7 days ago          2.3 GB
```

Figura 24 Imágenes de contenedores Docker creados

6.2.5 Gestión y monitorización

Lanzar contenedores mediante el uso de scripts no parece una forma muy amigable de gestionarlos. Por eso decidimos instalar el gestor de contenedores Portainer, el cual nos permitirá de una forma sencilla comprobar los contenedores que se están ejecutando en este momento, lanzar más instancias, cambiarles la configuración, etc.

Para la gestión y monitorización del correcto funcionamiento de los contenedores Docker haremos uso de Portainer, que, como hemos expuesto anteriormente, es una interfaz web diseñada para este menester.

Se descarga y ejecuta el contenedor oficial de Portainer que se encuentra en Docker Hub usando el siguiente comando:

```
docker run --name portainer -d \
    -p 9000:9000 \
    -v /var/run/docker.sock:/var/run/docker.sock \
    -v portainer_data:/data \
    portainer/portainer
```

Esto descarga y despliega un contenedor con Portainer ya configurado para ejecutarse y exponer su puerto para poder acceder mediante navegador. Una vez ejecutado encontraremos la interfaz de Portainer dirigiendo nuestro navegador web a la dirección del equipo anfitrión en el puerto 9000.

Cuando introducimos nuestros credenciales podemos acceder a un listado de contenedores (Fig. 25) y el estado en que se encuentran (parado, funcionando, etc.) y entonces decidir que hacemos con el contenedor en cuestión

Figura 25 Listado de contenedores Docker en estado "funcionando"

Como interesa ejecutar varios servidores VNC en el mismo equipo duplicaremos los contenedores de las versiones que se necesiten el número de veces que se requiera.

Para desplegar más contenedores del tipo que deseamos, elegimos el contenedor en cuestión de la lista y en la interfaz donde se encuentran sus características elegimos la opción de duplicar/editar (Fig. 26).

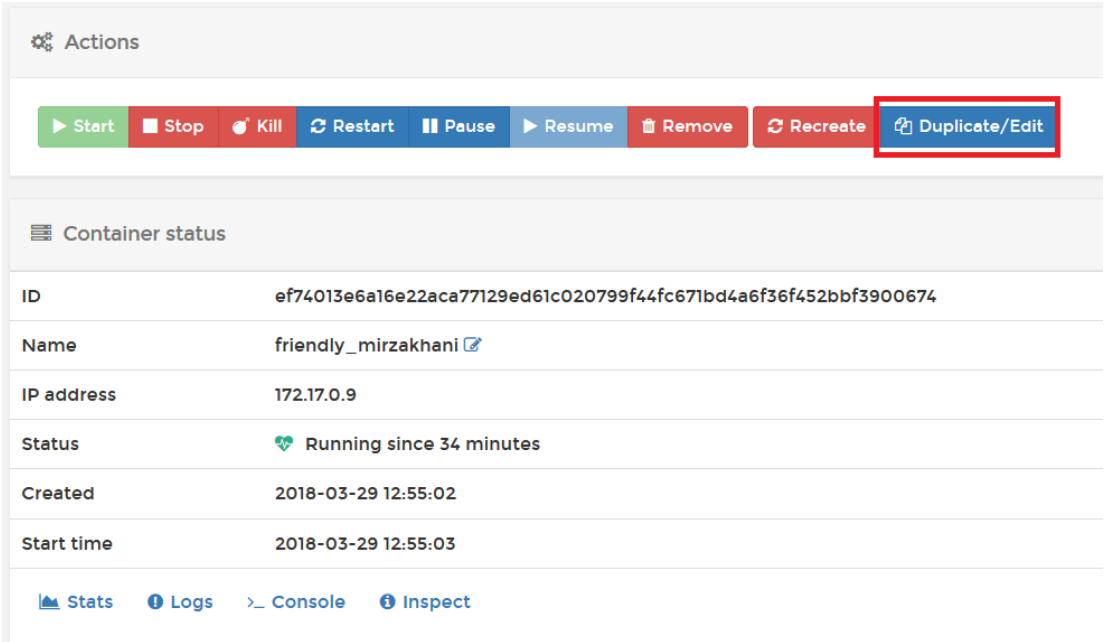

Figura 26 Opciones del contenedor soportadas por Portainer

Las únicas variables que no pueden compartir dos contenedores de Docker distintos es el nombre y el puesto en el que se han expuesto en el anfitrión (Fig. 27), los nombres se pueden generar de forma aleatoria por lo que no supone ningún problema, pero sí que hay que conocer los puertos que se hayan ocupados por otro contenedor y por eso se opta por utilizar los puertos consecutivos al 5900.

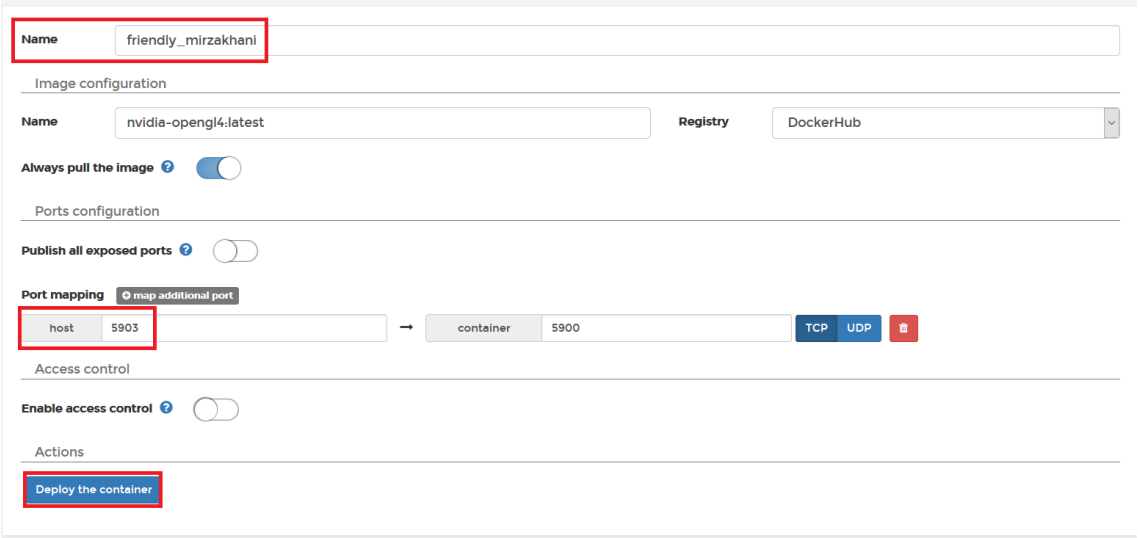

Figura 27 Nombre y puerto *mapeado* en el anfitrión

Durante las pruebas mientras se estaba desarrollando y depurando las imágenes de los contenedores, éstos han llegado a un punto en el que eran inusables. En informática es imposible estar seguro de que no ocurrirá ningún fallo. No obstante, si ocurriera alguno, Portainer ofrece la posibilidad de volver a crear el contenedor (Fig. 28), el cual se restaurará a su estado original.

Esto hace muy fácil recuperar rápidamente contenedores que no funcionen correctamente.

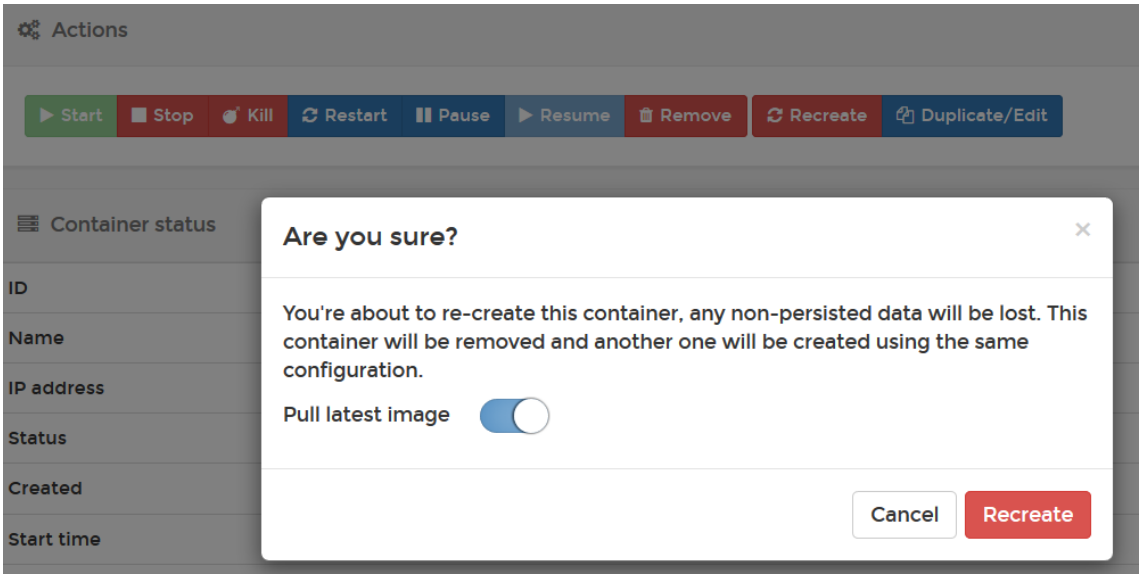

Figura 28 Reconstrucción de un contenedor

Con el comando de la siguiente ilustración (Fig. 29) se puede comprobar cómo hemos desplegado los contenedores cada uno a un puerto determinado del equipo anfitrión. En esta ocasión se han ejecutado 4 contenedores de la versión VNC y 2 de la versión noVNC, acto seguido se procede a configurar los proxys para redirigir los clientes a los distintos servidores que se encuentren sin usar.

```
docker ps

CONTAINER ID        IMAGE                          PORTS                    NAMES
8f90fb58e330        nvidia-opengl-novnc:latest     0.0.0.0:6081->6080/tcp   priceless_blackwell
e410c180d8b9        nvidia-opengl-novnc:latest     0.0.0.0:6080->6080/tcp   hungry_wozniak
dcb37b22157a        nvidia-opengl:latest           0.0.0.0:5904->5900/tcp   hopeful_jones
b33d9daeb3f6        nvidia-opengl:latest           0.0.0.0:5903->5900/tcp   flamboyant_bhabha
f0dbd840ae0c        nvidia-opengl:latest           0.0.0.0:5902->5900/tcp   affectionate_darwin
a7848e895b56        nvidia-opengl:latest           0.0.0.0:5901->5900/tcp   friendly_payn
```

Figura 29 Listado de contenedores ejecutándose

6.3 Configuración de los proxys

En este punto, se procede a configurar los proxys para redirigir a los clientes hasta las distintas direcciones donde se encuentran los servidores VNC, este proxy hace las veces de balanceador de carga, ya que tiene constancia de los servidores que no se encuentran en uso y reparte las conexiones mediante una política predeterminada.

6.3.1 HAProxy

HAProxy será la solución que usaremos para nuestros servidores VNC, al configurarse en modo TCP actúa haciendo de *pasarela SSL*[13], esto quiere decir que redirige el tráfico cifrado sin analizar su contenido por lo que el tráfico se encuentra cifrado en todo el trayecto. Además, se ha configurado para que se mantengan las conexiones activas dada la naturaleza del protocolo usado, consistente en conexiones largas y necesidad de respuesta rápida.

Por último, el balanceador de carga se ha configurado en el modo "menos conectado", por lo cual ante una nueva petición del cliente, el proxy lo redirigirá al servidor que haya recibido menos clientes en ese momento y se encuentre sin uso.

```
haproxy.cfg

global
  log 127.0.0.1   local0
  log 127.0.0.1   local1 notice

frontend vnc
  bind *:5900
  mode tcp
  option tcpka
  option tcp-smart-accept
  default_backend vnc-backend

backend vnc-backend
```

```
   mode tcp
   option tcpka
   option tcp-smart-connect
   balance leastconn

  server vnc1 192.168.X.X:5901 check maxconn 1
  server vnc2 192.168.X.X:5902 check maxconn 1
  server vnc3 192.168.X.X:5903 check maxconn 1
  server vnc4 192.168.X.X:5904 check maxconn 1
```

Los clientes podrán conectarse a cualquier servidor a través del puerto 5900 del equipo donde se encuentra el proxy. El número de usuarios permitidos por servidor es 1 por motivos de seguridad y este será el número de máximas conexiones permitidas para cada servidor.

6.3.2 Nginx

En cuanto a los servidores noVNC haremos uso de nginx ya que lo que nos interesa de este proxy es su función de proxy inverso HTTP[14], para ello se instala el certificado .X509 en el proxy y éste matendrá el tráfico cifrado con el cliente. Entre el servidor y el proxy el tráfico estará sin cifrar ya que el proxy ha de hacer peticiones de recursos al servidor en nombre del cliente y estas peticiones no pueden viajar cifradas.

En este caso el número de conexiones permitidas a cada servidor ha de ser 3 ya que se solicitan 3 recursos:

- La página de login
- Establecer una conexión websocket (mediante websockify)
- La página que muestra el escritorio

```
ngingx.conf

events {

}
http {
      upstream novnc {
      zone novnc 32k;
      least_conn;
      server 192.168.X.X:6080 max_conns=3;
      server 192.168.X.X:6081 max_conns=3;
      }

server {    listen 6080 default_server;
    ssl    on;
    ssl_certificate    /path/to/cert.pem;
    ssl_certificate_key    /path/to/key.pem;
    root /usr/share/nginx/html;
    index index.html index.htm;

    gzip on;
```

```
    gzip_types text/plain application/javascript text/xml text/css;
    gzip_proxied any;


    location = /websockify {
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
        proxy_pass http://novnc;
    }

    location / {
        try_files $uri @proxy;
    }

    location @proxy {
        proxy_set_header X-Real-IP  $remote_addr;
        proxy_set_header X-Forwarded-For $remote_addr;
        proxy_set_header Host $host;

        proxy_pass http://novnc;
        max_ranges 0;
    }

}
}
```

Los clientes podrán conectarse a cualquier servidor a través del puerto 6080 del equipo donde se encuentra el proxy.

Capítulo 7. Presupuesto

Ante la diversidad de hardware disponible sobre los que se puede realizar la infraestructura destinada al sistema cliente-servidor se propusieron dos soluciones acerca de cómo puede nuestro clúster de equipos anfitriones llevarse a cabo.

7.1 Soluciones propuestas

Crear el clúster usando un número reducido de mainframes, esto es, hardware específico capaz de ser equipado con varios procesadores y tarjetas gráficas que pudieran ejecutar un número muy elevado de instancias del servidor VNC o utilizando un número elevado de ordenadores equivalentes a las estaciones de trabajo como las que se están empleando en la actualidad en el equipo de ingeniería, a esta última opción la denominaremos como multinodo.

A continuación se exponen los pros y contras de cada una de las soluciones ideadas.

7.1.1 Mainframes

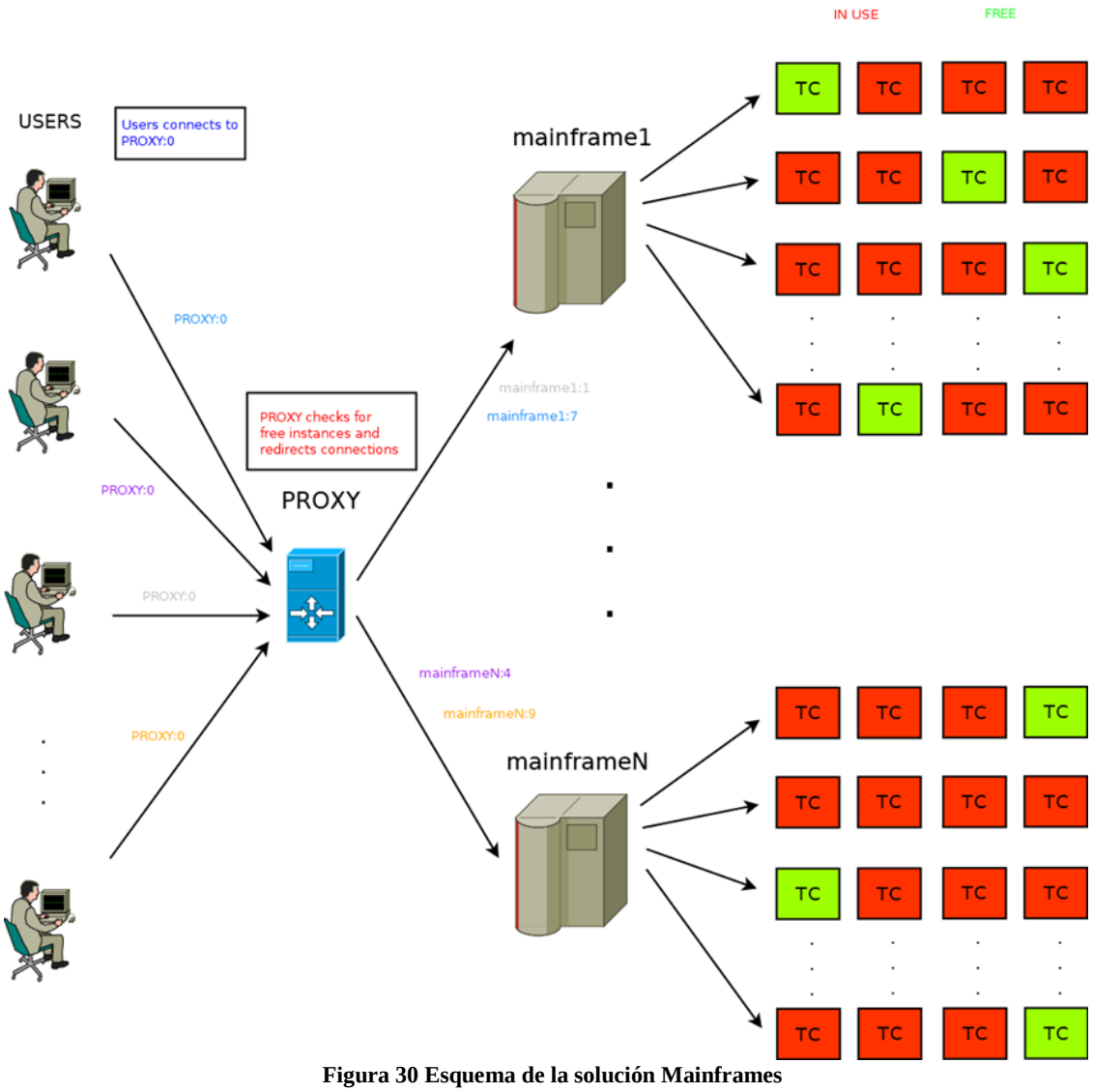

Figura 30 Esquema de la solución Mainframes

Tal y como hemos descrito anteriormente (Fig. 30), consiste en un reducido número de equipos muy potentes como anfitriones de los servidores VNC. Este montaje plantea una serie de ventajas e inconvenientes.

Ventajas:

- Número reducido de equipos a configurar
- Menor consumo energético (que la opción multinodo)

Inconvenientes:

- Necesario hardware específico
- Uso de ancho de banda por equipo más alto
- Peor tolerancia a fallos
- Los procesadores de equipos mainframe están ideados para aplicaciones críticas, no están pensados para alto rendimiento

Para cada uno de estos mainframes consideramos que podrían ejecutarse sin problema unas 20 instancias de servidor VNC. Se determina que su tolerancia a fallos es baja porque en caso de un funcionamiento incorrecto, afectaría a más usuarios que la opción siguiente.

7.1.2 Multinodo

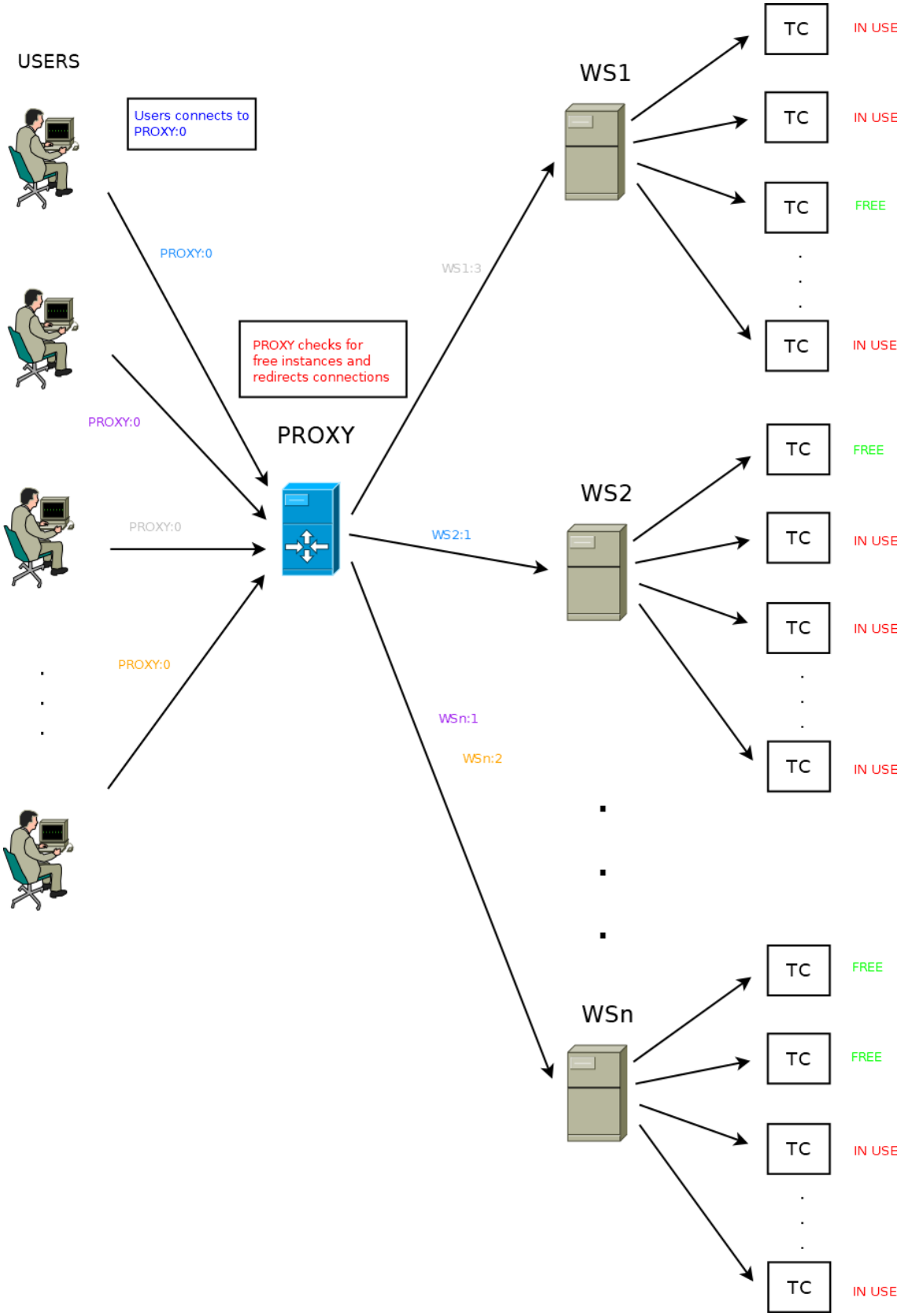

Figura 31 Esquema de la solución Multinodo

Esta opción consiste en un elevado número de equipos de características parecidas a una estación de trabajo ya disponibles (Fig. 31). Tal y como con la opción anterior se ha determinado una serie de ventajas e inconvenientes.

Ventajas:

- Mayor tolerancia a fallos
- Fácilmente escalable
- Cada nodo tiene el hardware "equilibrado" para la ejecución de software CAE
- Pueden usarse gabinetes de rack o equipos de sobremesa

Inconvenientes:

- La configuración es más difícil
- Ocupa más espacio físico

En este caso estimamos que cada uno de los nodos podría soportar sin mucho problema un número de instancias del servidor VNC en torno a 10. Un fallo en alguno de estos nodos afectaría a menos usuarios y sería más fácilmente subsanable repartiendo su carga entre los nodos restantes.

7.2 Solución escogida

Tras reflexionar sobre las opciones con las que se podría desplegar el sistema, nos decantamos por la opción de **multinodo** basándonos en el menor número de desventajas que representa su implementación, además de resultar la configuración más óptima para el tipo de sistema que queremos implementar dada la naturaleza del software que queremos ejecutar, es más fácilmente escalable y más tolerante a fallos.

7.3 Coste

Para nuestro sistema, hemos determinado los elementos que se requerirían para cada uno de los nodos que formarían parte. Para ello se establece el hardware que debería usarse tanto en el servidor como en el cliente.

Para el servidor:

El servidor consistirá en un equipo de altas prestaciones, con un procesador de elevado número de núcleos, una cantidad muy elevada de memoria RAM y una tarjeta gráfica especializada en software de diseño e ingeniería. Para ello se puede optar por dos factores de forma distintos del gabinete, este puede ser *enrackable*, es decir, que puede insertarse en un rack de comunicaciones o un equipo de sobremesa.

Para los clientes:

Para los clientes se podrá optar por equipos portátiles o sobremesa de bajas prestaciones, lo suficiente para poder ejecutar el cliente VNC y diverso software ofimático. Cada una de estas opciones iría acompañada de un monitor de 24", suficiente para trabajar cómodamente.

En la siguiente tabla se muestran los precios desglosados de cada uno de estos elementos:

		Precio	Precio (+IVA)	Modelo	Características
Servidor	**Rack**	8.985,00 €	10.871,85 €	HPE ProLiant DL380 Gen10	Intel Xeon-S 4114 10-Core (2.20GHz 13.75MB) NVIDIA Quadro P4000 Graphics Accelerator 128GB (4 x 32GB) PC4-2666V-R DDR4 RDIMM
	Estación de trabajo	4.557,00 €	5.513,97 €	Z6 G4	Intel xeon 4116 2.1 2400 Mhz 12C 128 GB (4*32gb) DDR4 2666 ECC NVIDIA Quadro P4000 8 gb (4) DP GFX
Cliente	**Portátil**	609,00 €	736,89 €	Dell Vostro 5568	i5 7200U 8GB Ram
	Desktop	549,00 €	664,29 €	Inspiron DT 3668	i5 8GB Ram
	Monitor	180,00 €	217,80 €	U2414H	

Tabla 14 Precio desglosado de los elementos posibles del sistema

7.4 Plan de amortización

Para cada uno de los nodos establecemos los siguientes parámetros:

- **Número de usuarios:** 10
- **Período de funcionamiento:** 5 años

Esto nos arroja un coste total por nodo y gasto por usuario a lo largo del tiempo para cada una de las opciones consideradas en la siguiente tabla:

Combinaciones	Precio	Precio/usuario	Precio/usuario/mes
Rack + portátil	16.875,00 €	1.687,50 €	28,13 €
Rack + sobremesa	16.275,00 €	1.627,50 €	27,13 €
Estación de trabajo + portátil	12.447,00 €	1.244,70 €	20,75 €
Estación de trabajo + sobremesa	11.847,00 €	1.184,70 €	19,75 €

Tabla 15 Plan de amortización a 5 años de un nodo

El precio final por usuario en el período de amortización previsto resulta interesantemente bajo si se tiene en cuenta las decenas de miles de euros que costaría equipar a cada usuario con su propia estación de trabajo. En este caso, la contención del gasto es bastante notable.

Capítulo 8. Resultados del trabajo

Para poder hacer una aproximación sobre el número de usuarios simultáneos que podría funcionar sin problema en cada uno de los nodos (equipos anfitriones) se han de realizar una serie de pruebas acerca de la carga sobre el sistema de cada uno de los contenedores, el ancho de banda que consumen en cada codificación y la experiencia de usuario basándonos en un sistema de Mean Opinion Score.

8.1 Carga del sistema

Al realizar varias pruebas de rendimiento y usabilidad, para cada uno de los servidores se obtiene los siguientes valores de consumo de recursos del sistema:

RAM del sistema	~ 1 GB
RAM de la GPU	~ 100-200 MB
Uso de la CPU	Varía según usuario

Tabla 16 Consumo de recursos del sistema por el servidor VNC

Estos resultados muestran un consumo insignificante para las especificaciones de nuestro equipo anfitrión. Mientras que el consumo de RAM del sistema se mantiene más o menos estable en torno a 1 GB, el consumo de memoria RAM de la tarjeta gráfica depende de la complejidad del modelo representado (Fig. 32). Aun así, no suponen ningún inconveniente dado que el sistema posee 128 GB de RAM y la tarjeta gráfica 8 GB de RAM.

Estas prueba se han realizado ejecutando el software CAE en varios contenedores independientes y durante el trascurso de la misma no se han observado problemas de funcionamiento con distintas cantidades de usuarios.

```
nvidia-smi


+-----------------------------------------------------------------------------+
| NVIDIA-SMI 390.48                 Driver Version: 390.48                    |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Quadro P4000        Off  | 00000000:21:00.0  On |                  N/A |
| 48%   45C    P0    28W / 105W |    394MiB /  8111MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|    0      2945      G   /usr/libexec/Xorg                             99MiB |
|    0     28651      G   ...ANSA/ansa_v17.1.0/bin/ansa_linux_x86_64    74MiB |
|    0     29145      G   ...ANSA/ansa_v17.1.0/bin/ansa_linux_x86_64   141MiB |
|    0     29605      G   ...ANSA/ansa_v17.1.0/bin/ansa_linux_x86_64    42MiB |
+-----------------------------------------------------------------------------+
```

Figura 32 Consumo de memoria RAM de la gráfica por distintos contenedores

8.2 Ancho de banda

En este caso se han estudiado los distintos consumos de ancho de banda que se han dado con las distintas codificaciones establecidas entre el cliente y el servidor VNC. En un equipo ocurren simultáneamente muchos procesos que hacen uso de ancho de banda de red. A poder aislar los valores que corresponden sólo al tráfico VNC haremos uso de una herramienta llamada **nethogs** (Fig. 33) que hace un análisis del ancho de banda consumido dividido en procesos.

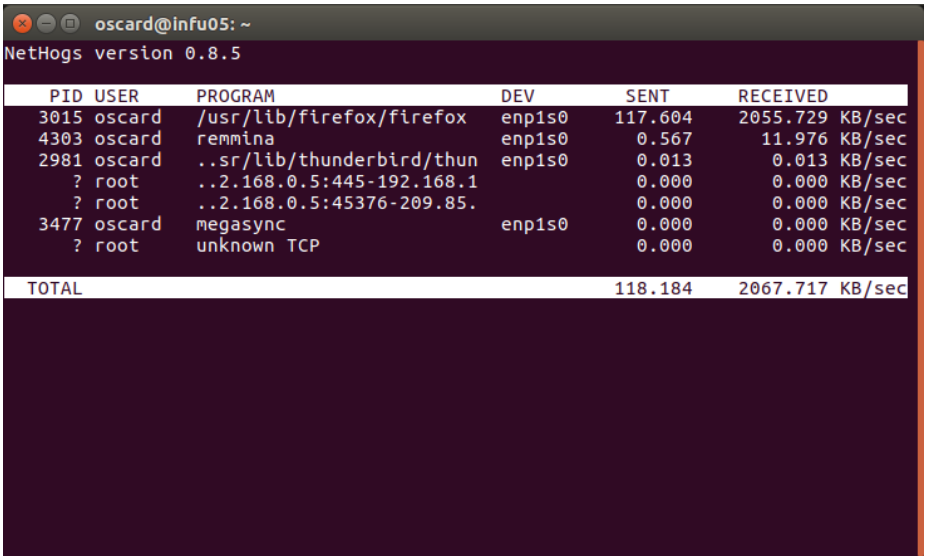

Figura 33 Nethogs

Una vez identificado el proceso correspondiente al servidor VNC, tomamos muestras a lo largo de un período que oscila entre los 5 y 10 minutos, suficiente para obtener de forma aproximada una media y un valor máximo o de pico.

8.2.1 Versiones VNC

Para obtener estos datos se han testeado todas las posibles codificaciones soportadas por TigerVNC que se consideraban usables.

Por lo tanto sólo se han testeado las codificaciones ZRLE en sus niveles de compresión de 1 a 3 y la codificación Tight con sus valores por defecto dado que las demás codificaciones no arrojaban una experiencia de usuario aceptable para los testeadores.

En la siguiente tabla se pueden observar los valores medios y de pico obtenidos en las pruebas, han sido realizados por personas con un manejo avanzado del software CAE para obtener la mayor fidelidad a un caso real de uso y redondeadas a la centena más cercana.

Codificación	Media (kBps)	Valor máximo (kBps)
Raw	2200	9400
Hextile	2000	9100
ZRLE (Level 1)	1800	8600
ZRLE (Level 2)	1400	7800
ZRLE (Level 3)	1300	7200
Tight	900	5400

Tabla 17 Uso de ancho de banda según codificaciones en TigerVNC

Estos datos son bastante aproximados a los determinados por el propio cliente ya que por ejemplo Hextile nos da una media de 2000kBps (equivalente a 16Mbps de media), que está ajustado a los 20000kbps (20Mbps) que muestra como estimación la información referente a la conexión en el cliente TigerVNC (Fig. 34).

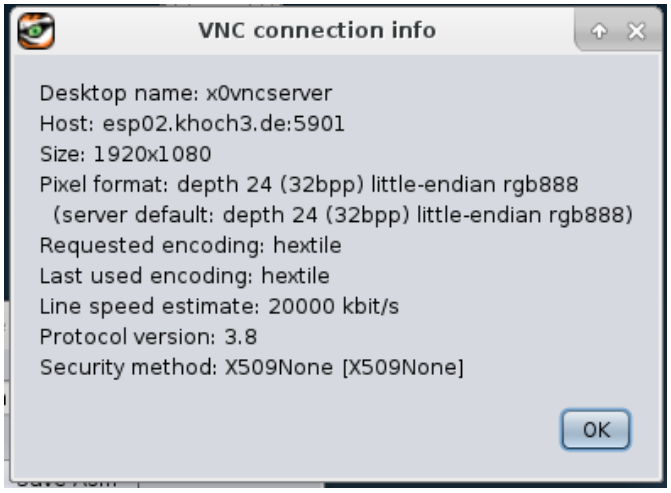

Figura 34 Información de la conexión (Hextile)

Teniendo en cuenta que los valores más elevados son 2200 kBps (17,6 Mbps) de media resulta factible el número de usuarios por nodo que hemos planteado, ya que este está equipado con una interfaz de red Gigabit Ethernet (1000Mbps), e, incluso en el peor de los casos en que todos los usuarios hicieran un uso máximo de ancho de banda este lo soportaría sin problema. Esta posibilidad es bastante remota ya que como puede observarse en la representación gráfica del tráfico en codificación Tight que podemos observar a continuación (Fig. 35), el ancho de banda consumido es muy irregular y los valores máximos se alcanzan en momentos muy puntuales.

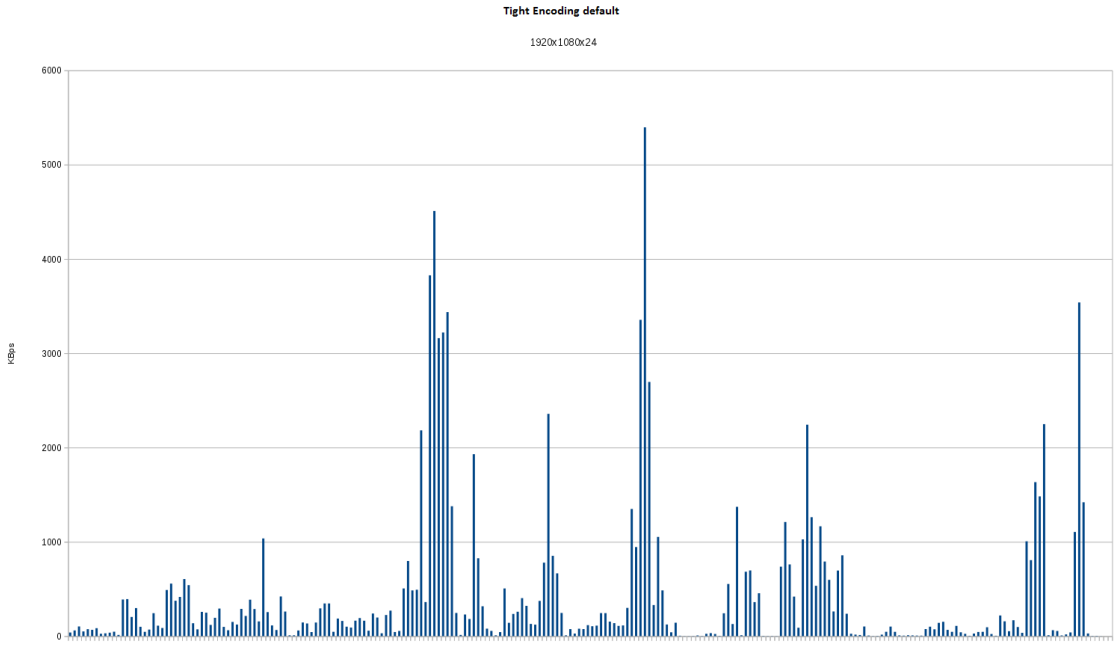

Figura 35 Representacion gráfica del tráfico generado durante 5 minutos (Tight)

Por lo tanto, 10 usuarios podrían trabajar al unísono sin ningún problema en ninguno de los casos.

8.2.2 Versiones noVNC

En este caso estamos limitados a las codificaciones Raw y Hextile dado que el motor Javascript de los navegadores web es incapaz de momento de tratar en tiempo real las codificaciones basadas en compresión zlib.

Codificación	Media (kBps)	Valor máximo (kBps)
Raw	500	2400
Hextile	400	1900

Tabla 18 Uso de ancho de banda según codificaciones en noVNC

El ancho de banda en este caso es más reducido debido a que la conexión entre cliente y servidor no usa protocolo VNC sino que aprovecha la tecnología HTML para realizar el dibujo en el navegador web del cliente.

8.3 Experiencia de usuario

8.3.1 Mean Opinion Score (MOS)

Para evaluar la experiencia de usuario se ha establecido un sistema de Mean Opinion Score o MOS[15].

MOS es una medida utilizada en el ámbito de la calidad de experiencia y la ingeniería de telecomunicaciones, que representa la calidad general de un estímulo o sistema. Es la media aritmética sobre todos los "valores en una escala predefinida que un sujeto asigna a su opinión sobre el rendimiento de una calidad de sistema". Estas clasificaciones se suelen recoger en una prueba subjetiva de evaluación de la calidad.

El MOS es una medida comúnmente utilizada para la evaluación de la calidad de video, audio y audiovisuales, pero no se limita a esas modalidades, dado que nos sirve para evaluar en este caso la experiencia de usuario en el escritorio remoto.

Por lo tanto establecemos un sistema MOS con las siguientes características:

Valoración	Puntuación
Excelente	5
Buena	4
Suficiente	3
Pobre	2
Mala	1

Tabla 19 Valoración y su equivalencia numérica

8.3.2 Versiones VNC

Con 4 usuarios con dominio en el software CAE nos disponemos a rellenar una tabla con los valores descritos en el apartado anterior, con esto se pretende conseguir unos valores bastante significativos, ya que la valoración máxima correspondería al uso del software en un equipo físico. Con ello se pretende evaluar de un modo sencillo la viabilidad de introducir este sistema en un entorno de trabajo convencional.

Para estas valoraciones, se hizo un uso del escritorio remoto en red local durante horas, por lo que son lo más ajustadas a la evaluación que pudiera hacer un tercero.

Codificación	Usuario 1	Usuario 2	Usuario 3	Usuario 4	Media
Raw	5	4	5	4	4.5
Hextile	4	4	5	4	4.25

ZRLE (Level 1)	4	4	5	4	4.25
ZRLE (Level 2)	4	4	4	4	4
ZRLE (Level 3)	3	4	4	4	3.75
Tight	3	3	3	3	3

Tabla 20 Puntuaciones otorgadas por los usuarios de versiones VNC

Como se puede comprobar, todas las codificaciones han obtenido buen resultado excepto la Tight, más pensada para funcionar en entornos con ancho de banda limitado y no basada en el rendimiento o la correcta muestra de las imágenes.

Como las demás codificaciones son bastante parejas en cuanto a valoración y el ancho de banda y uso de CPU no resultan un problema, determinamos que la mejor opción es utilizar la codificación **ZRLE** con **nivel de compresión 1**.

8.3.3 Versiones noVNC

En este caso se repiten las pruebas anteriores pero con la versión noVNC en el navegador web hecho por los mismos usuarios.

Codificación	Usuario 1	Usuario 2	Usuario 3	Usuario 4	Media
Raw	2	2	3	3	2.5
Hextile	2	2	3	2	2.25

Tabla 21 Puntuaciones otorgadas por los usuarios de versiones noVNC

En este caso los resultados no han sido satisfactorios, llegando incluso los usuarios a parar las pruebas a los pocos minutos dada la falta de usabilidad. Las imágenes no se muestran de forma fluida y el software no responde en la latencia necesaria para un correcto uso.

Por lo tanto estas versiones se **desaconsejan** para su uso en software CAE.

Capítulo 9. Conclusiones

Aunque resulta bastante obvio que el mejor rendimiento a la hora de ejecutar un software con altos requisitos es ejecutarlo directamente sobre un equipo, la experiencia de usuario al ejecutarse el software CAE desde un cliente VNC ha resultado realmente satisfactoria. Se han realizado diversas pruebas en las cuáles los usuarios han podido realizar su trabajo sin ningún problema, tal y como lo harían desarrollando su labor sobre su propio equipo físico. El rendimiento resulta totalmente aceptable para las versiones VNC que han superado todas las expectativas que había al principio del desarrollo.

Sin embargo, las versiones noVNC no han alcanzado el rendimiento requerido para el tipo de trabajo que se esperaba y suponen una brecha de seguridad ya que el tráfico entre proxy y servidor no se puede cifrar, de todos modos, su desarrollo puede aprovecharse para desplegar aplicaciones que no requieran de un rendimiento tan alto y poder acceder a ellas remotamente sólo con un navegador web.

El consumo de recursos del equipo anfitrión por parte de los servidores VNC es realmente bajo para sus especificaciones técnicas y seguramente nos veamos antes limitados por el ancho de banda de la interfaz de red del equipo que por número de servidores VNC ejecutándose en paralelo. Aunque con un interfaz de red Gigabit Ethernet como el que dispone es muy difícil determinar donde encontraríamos el *cuello de botella* del sistema.

No obstante, nos hemos decantado por una posición conservadora y determinado un número de usuarios por nodo moderado para los resultados obtenidos, buscando en todo momento un excelente rendimiento y primar la experiencia de usuario.

En cuanto al aspecto económico, el plan de amortización a largo plazo nos da un coste por usuario realmente bajo para las prestaciones que ofrece el sistema, resultando una opción muy atractiva a nivel empresarial.

9.1 Propuesta de trabajos futuros

Aun habiendo cumplido con todos los objetivos, el sistema se puede mejorar en los siguientes aspectos:

Los usuarios siguen necesitando saber la dirección actual del proxy y los archivos SSL válidos, como una expansión futura del trabajo se ha pensado en un gestor de configuración automática[16] de los equipos cliente. Esta se haría remotamente a través de un servidor dedicado a ello y de forma segura a través de SSH.

Este sistema también serviría para monitorizar y corregir el mal funcionamiento de los contenedores.

También, a raíz de este sistema de configuración se podrían crear contenedores específicos para cada usuario bajo demanda, dependiendo de las herramientas que necesitara que estuvieran instaladas para una labor determinada.

Además, se podría separar el resto de tareas que se realizan en las estaciones de trabajo tales como cálculo a servicios en la "nube" del mismo modo que hemos hecho con el escritorio, introducirlos en el entorno de trabajo e interactuar con nuestro proyecto.

Por último, se podría sustituir el despliegue a base de comandos utilizando algún orquestador Docker tal como Docker Swarm o Kubernetes, cuya configuración inicial es más compleja pero aporta muchas funciones para desplegar y configurar servidores y proxys automáticamente.

Bibliografía

[1] Mui, L.; Pearce, E. X Window system administrator's guide : for X version 11, pp. 3-4, February 1993.

[2] The VirtualGL Project, VirtualGL Background, August 2017. [En línea]. Available: https://virtualgl.org/About/Background/. [Último acceso: 02 07 2018].

[3] Docker Inc., Docker documentation, 2018. [En línea]. Available: https://docs.docker.com/. [Último acceso: 28 06 2018].

[4] Alonso A. Docker: definición y ventajas principales, November 2017. [En línea]. Available: https://labs.beeva.com/docker-definici%C3%B3n-y-ventajas-principales-fe49f1a6481b. [Último acceso: 28 06 2018].

[5] RealVNC Limited, RealVNC - Frequently asked questions, 2018. [En línea]. Available: https://www.realvnc.com/en/connect/docs/faq/. [Último acceso: 29 06 2018].

[6]Richardson, T.; Levine, J. Internet Engineering Task Force (IETF) RFC: 6143 The Remote Framebuffer Protocol, pp. 2-3, March 2011.

[7] Ossman, P. TigerVNC, 2018. [En línea]. Available: http://tigervnc.org/. [Último acceso: 29 06 2018]

[8] Ossman, P. noVNC Wiki, 2018. [En línea]. Available: https://github.com/novnc/noVNC/wiki [Último acceso: 29 06 2018]

[9] Richardson, T.; Levine, J. Internet Engineering Task Force (IETF) RFC: 6143 The Remote Framebuffer Protocol, pp. 23-31, March 2011.

[10] Tarreau, W. HAProxy. The Reliable, High Performance TCP/HTTP Load Balancer, 2018. [En línea]. Available: http://www.haproxy.org/. [Último acceso: 05 04 2018].

[11] McCormick, M. Docker image for running X GUI apps with OpenGL hardware acceleration via the nvidia driver, December 2015. [En línea]. Available: https://github.com/thewtex/docker-opengl-nvidia. [Último acceso: 27 03 2018].

[12] McCormick, M. A docker image that supports rendering graphical applications, 2018. [En línea]. Available: https://github.com/thewtex/docker-opengl. [Último acceso: 19 02 2018].

[13] Ploria, M. Pass-through SSL with HAProxy, February 2015. [En línea]. Available: https://scriptthe.net/2015/02/08/pass-through-ssl-with-haproxy/. [Último acceso: 05 04 2018].

[14] Nginx Inc., NGINX Reverse Proxy, 2018. [En línea]. Available: https://docs.nginx.com/nginx/admin-guide/web-server/reverse-proxy/. [Último acceso: 28 06 2018].

[15] International Telecommunication Union, ITU-T Rec. P.10 G.100 (11/2017) Vocabulary for performance and quality of service, pp. 8-9, November 2017.

[16] Dreyfuss, J. Deployment Management Tools: Chef vs. Puppet vs. Ansible vs. SaltStack vs. Fabric, August 2015. [En línea]. Available: https://blog.takipi.com/deployment-management-tools-chef-vs-puppet-vs-ansible-vs-saltstack-vs-fabric/. [Último acceso: 29 06 2018].