TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

UNIVERSIDAD DE ALMERIA

ESCUELA POLITÉCNICA SUPERIOR

"Desarrollo de un hypervisor centralizado que permita la administración remota de cualquier máquina virtual"

Curso 2016/2017

Alumno/a:

Antonio Ángel Ureña Gálvez

Director/es:

Juan Francisco Sanjuan Estrada

Contenido

Lista de figuras

1. Introducción

Cada vez hay menos cantidad de empresas que ofrecen sus servicios de manera centralizada, en un único equipo y con un solo sistema operativo, para la gestión de correo electrónico corporativo, página web de la empresa, servidor ftp para la gestión de archivos, etc. Al utilizar un único equipo con todos estos servicios, seguramente se está poniendo en riesgo la información de la organización dado que un equipo, por ejemplo, con un servicio web está muy expuesto a:

- Ataques informáticos.
- Saturación de la red.
- Posibles problemas de incompatibilidad con algún software.
- Si se llega a caer el equipo se pararían todos los servicios.
- No es portable a otro hardware distinto.

Por estos y otros motivos se utilizan las máquinas virtuales (MVs):

- Disponibilidad: los servicios están separados en MVs, unas aisladas de otras, si se cae una MV solo se cae un servicio.
- Aislamiento: si un atacante consigue vulnerar el servicio web como el sistema está aislado solo conseguirá comprometer el servicio web pero los demás servicios seguirán intactos.
- Se pueden migrar los datos con relativa facilidad si hubiera un fallo de hardware.
- Compatibilidad con cualquier software aunque requerirá del sistema operativo compatible con ese software.

Por ello, hoy en día, hay muchas empresas y organismos públicos que utilizan distintas MVs ofreciendo diferentes servicios que se ejecutan sobre el mismo sistema, compartiendo los mismos recursos hardware entre las distintas MVs, pero sin solaparse y aislando las aplicaciones entre las distintas MVs, lo que permite una mejor administración y control del sistema.

Actualmente, están apareciendo en el mercado nuevos hypervisores para la gestión de MVs, con lo que la variedad de hypervisores, tanto de pago como gratuitos, está en aumento. Por ejemplo, VmWare, VirtualBox, KVM, OpenStack, XEN…

En este tipo de sistemas, el administrador debe revisar y mantener las diferentes MVs, así como realizar copias de seguridad de las mismas, hacer restauraciones si algún servicio no funciona correctamente, denegar servicios o habilitarlos en diferentes momentos. Estas tareas se hacen tediosas cuando el abanico de MVs crece, y aún más si se trabaja simultáneamente con distintos tipos de hypervisores.

Actualmente los hypervisores permiten migrar MVs de un equipo a otro, pero no permiten mover una MV de un tipo de hypervisor a otro hypervisor diferente, por ejemplo no se permite pasar una MV de VirtualBox a VmWare. Esta situación, fuerza a los administradores de sistemas a trabajar con un único hypervisor, ya que les disuade la idea de mover las MVs creadas con el hypervisor A a un nuevo hypervisor B. Sin embargo, el hypervisor centralizado diseñado en el presente TFG permitirá al administrador abstraerse respecto del hypervisor utilizado, permitiendo al administrador de sistemas de una empresa, utilizar diferentes tipos de hypervisores. La Figura 1 muestra un esquema general de como el

hypervisor centralizado propuesto en el presente TFG puede gestionar distintos tipos de hypervisores distribuidos en diferentes equipos de la empresa. Básicamente, el hypervisor centralizado está compuesta por una aplicación web y distintas aplicaciones clientes que sirven de interfaz entre el administrador de sistemas y los distintos hypervisores distribuidos entre los equipos de la empresa, independientemente de los distintos tipos de hypervisores utilizados.

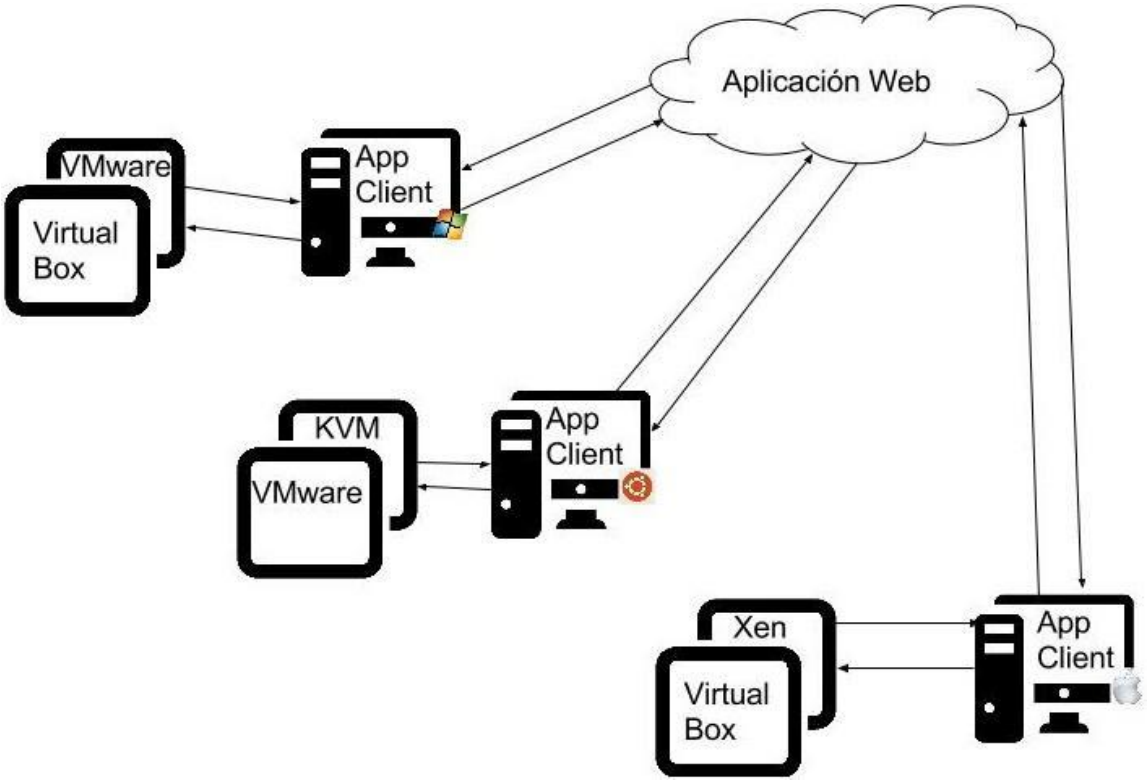

Fig.1: Hypervisor Centralizado.

1.1. Objetivos del TFG

El administrador de sistemas gestiona las MVs de la empresa con la consiguiente pérdida de tiempo que supone acceder remotamente a los diferentes servidores donde se alojan las distintas MVs. En este sentido, nace la idea de desarrollar un marco común en forma de aplicación web para estas tareas, para las cuales el administrador ya no tendrá que entrar al sistema y programar dichas tareas o gestionarlas por separado, teniendo cada MV, independientemente del tipo de hypervisor utilizado, al alcance de un solo click desde cualquier dispositivo que tenga acceso a Internet (ordenadores, portátiles, tablets, smartphones, …).

El presente Trabajo Fin de Grado (TFG) pretende realizar un hypervisor centralizado en forma de aplicación web, que permita realizar un mantenimiento básico de las máquinas virtuales (guardar el estado actual de la máquina, restaurar una máquina a un estado anterior, apagarla, reiniciarla, encenderla,…) creadas en cualquier equipo, de forma remota y sin importar el hypervisor utilizado para las diferentes máquinas virtuales, ya sea VmWare, VirtualBox, KVM, XEN…

En otro aspecto, cada vez se están desarrollando más aplicaciones web y menos de escritorio debido a las ventajas que conllevan:

- **Multiplataforma**: las aplicaciones web no dependen del sistema operativo que usa el usuario, las sirve el servidor a través del navegador, que normalmente si se encuentra actualizado no tendrá problemas con la visualización de la web.
- **Actualización**: el usuario no tiene que preocuparse por actualizar el software dado que el software ya es actualizado por quien lo ofrece.
- **Acceso instantáneo**: las aplicaciones web no necesitan que el usuario las descargue, instale y configure en su equipo, tan solo necesita un dispositivo con conexión a internet para hacer uso de la aplicación web.
- **Menor uso de memoria**: si un usuario hace una acción sobre una aplicación web, normalmente el servidor es el encargado de realizar la operación, por lo que el usuario consume menos recursos que con la misma aplicación en modo escritorio.
- **Múltiples usuarios**: las aplicaciones web pueden ser usadas por muchos usuarios al mismo tiempo.

Existen algunas aplicaciones web con un propósito similar al de este TFG, como la aplicación para empresas de VmWare (**vSphere**), sin embargo, esta aplicación, y otras semejantes, solo trabajan con su propio hypervisor [13]. Esta situación, fuerza a los administradores de sistemas a trabajar con un único hypervisor, pues sería tedioso lidiar con varios y de distintos fabricantes, tal y como se analizó en la asignatura Administración de Redes y Sistemas Operativos [3].

Este proyecto pretende solucionar esta situación, dejando que el administrador del sistema utilice los hypervisores que estime oportunos, ya sean gratuitos o de pago.

Así pues, los aspectos principales del TFG se centran en:

1. Desarrollo de una aplicación web con autentificación de usuarios, donde el usuario podrá visualizar el estado de cada una de las MVs y gestionarlas desde la propia web. También, se ofrece la posibilidad de añadir nuevos comandos parametrizados que permitan ofrecer otras acciones a los hypervisores, acciones que no se han definido previamente en este TFG. Además, se permite añadir funcionalidades para nuevos hypervisores que aparezcan en el mercado. Un aspecto importante en este TFG es definir correctamente las acciones a través de comandos parametrizados, donde estos comandos deben tener en cuenta el sistema operativo del equipo anfitrión, el tipo de hypervisor y parámetros de configuración de la MV.

2. Desarrollo de una aplicación cliente, la cual debe en primera instancia autentificar al usuario con una secret key (*AccessToken*). El siguiente paso a realizar será reconocer el sistema operativo del equipo anfitrión para determinar así que comandos utilizar para detectar los hypervisores y MVs que posee el sistema. Posteriormente, esta información se enviará a la web donde el usuario podrá gestionar sus MVs. Una vez ocurre la gestión, el programa cliente comprobará en la aplicación web si hay alguna orden relativa a sus MVs y ejecutará el comando adecuado, según el hypervisor utilizado. Finalmente, se enviará el éxito o fracaso de esta operación a la web actualizando el estado de las MVs.

1.2. Planificación temporal

A continuación, se explica el desarrollo del TFG según las siguientes fases:

- Fase de instalación: se prepara el entorno donde se van a automatizar las pruebas realizadas y el *deploy* de la aplicación web.
- Fase de diseño: se crea la base de datos, diagramas de clases y esbozos de la aplicación web y cliente.
- Fase de desarrollo: codificación de la aplicación web y cliente.
- Fase de ensayo: se preparan pruebas unitarias y se arreglan fallos en la codificación.

Al principio, se tuvo que preparar una plataforma donde realizar las pruebas y el *deploy* de forma automática. A continuación, se comentan los pasos realizados:

1. Primero se creó un proyecto en Gitlab [4], WebHypervisorCentralizado.
2. Se contrató el servicio de Windows Azure [1] y configuró un usuario creando pares de claves para poder acceder por ssh.
3. Se instaló en la máquina de Windows Azure el servidor web Nginx [8], la configuración llevo más tiempo del esperado pues al usar ruby como lenguaje de programación hubo que probar varias configuraciones.
4. Se instaló y configuró Jenkins [7] para automatizar las pruebas y el deploy.

Se han seleccionado estas herramientas por su manejo previo en la asignatura *Herramientas y Métodos de Ingeniería del Software [2].* La fase de instalación se completó en un total de 30 horas.

Después de probar la integración de un proyecto web simple con Jenkins, Nginx y Gitlab para ver que todo estaba dispuesto, empezó la fase de diseño. Se realizó el diseño de la base de datos con un diagrama de entidad-relación en mysql workbench, también se llevó a cabo el diseño de la aplicación web y la aplicación cliente con diferentes diagramas y bocetos. La fase de diseño se completó en un total de 20 horas.

Comienza entonces la fase de desarrollo, desarrollando una web muy simple solo con la funcionalidad CRUD (Crear, Obtener, Actualizar y Borrar) de la base de datos implementada y la funcionalidad de registrar y autenticar un usuario.

Posteriormente, se seleccionó un *template* adecuado para la página web que encaje con las características del esbozo de la página principal mostrado en la Figura 14. Se empieza a realizar simultáneamente la aplicación cliente y la aplicación web pues hay que ir probando si funcionan las peticiones a la web y si se envían peticiones desde la aplicación cliente. La fase de desarrollo se completó en un total de 252 horas.

Llega entonces la fase de ensayo, como se han estado haciendo pruebas durante todo el proceso solo hace falta terminar de pulir algunas cosas, generar pruebas para algunos aspectos y empezar a estudiar otros sistemas para insertar comandos parametrizados en la base de datos. Este proceso es indefinido, pues habría que insertar comandos parametrizados para cada sistema operativo y para cada hypervisor. En este caso el tiempo que llevó para el sistema operativo Windows 8.1 y los hypervisores de VmWare y VirtualBox fue de 28 horas.

Ha habido algunos cambios respecto a la planificación estimada en el anteproyecto inicial, como se puede observar:

Fases:

1. Fase de instalación: Tiempo estimado 26h / Real 30h.

1. Habilitar dos proyectos en Gitlab (Web y Cliente). 1h
2. Configurar el servidor de Windows Azure. 15h
3. Instalar y configurar Nginx. 4h
4. Instalar y configurar Jenkins para automatizar las pruebas y el deploy. 6h

En esta fase hubo dificultades a la hora de automatizar con Jenkins las pruebas y el deploy de la aplicación web, pues el lenguaje usado *Ruby* requiere de una configuración más exhaustiva en Jenkins.

2. Fase de diseño: Tiempo estimado 21h / Real 20h.

5. Diseño de la base de datos. 4h
6. Diseño de la Web. 10h
7. Diseño de la app cliente. 7h

3. Fase de desarrollo: Tiempo estimado 185h / Real 252h.

8. Desarrollo de la base de datos. 3h
9. Desarrollo de la Web.
130h
10. Desarrollo de la app cliente. 50h
11. Integración con Jenkins. 2h

En esta fase hay un desajuste de horas porque se estaba desarrollando simultáneamente la aplicación web y cliente mientras se hacían las pruebas y se arreglaban los fallos, aquellos más representativos eran los relacionados con la interacción de la aplicación web con la aplicación cliente y viceversa, lo que llevó más tiempo de lo estimado.

4. Fase de ensayo: Tiempo estimado 68h / 28h.

12. Pruebas de la base de datos. 8h
13. Pruebas de la Web. 24h
14. Pruebas de la app cliente. 12h
15. Prueba de la comunicación entre la Web y la app cliente. 20h
16. Puesta en marcha. 4h

Parte de esta fase se encuentra realizada en la fase anterior, las 28h corresponden a la realización de comandos parametrizados para los hypervisores VmWare y VirtualBox en la plataforma Windows 8.1.

5. Fase de difusión: Tiempo estimado 80h / 68h.

17. Elaboración de la memoria del TFG. 50h
18. Preparación defensa del TFG. 30h

Trabajo Total: Estimado 300h+80h/ Real 330h+68h.

2. Antecedentes

2.1. Concepto de máquina virtual

El concepto de máquina virtual fue introducido por el sistema VM/370 IBM desarrollado en base al sistema de la IBM/360 en 1972. Esto hizo realidad el poder ejecutar sistemas operativos simultáneamente sin cambiar el hardware. Para lograrlo, hay que separar dos funciones que realiza un sistema de tiempo compartido (CTSS): abstracción del hardware y multiprogramación.

El núcleo del sistema es conocido como hypervisor, y se ejecuta sobre el hardware, abstrayéndolo del sistema operativo y proporcionando un software entre el hardware y el sistema operativo anfitrión, lo que permite ejecutar diferentes sistemas operativos. Hay tres tipos de virtualización: emulación (Emulation), virtualización completa (Full Virtualization) y paravirtualización (Paravirtualization).

2.1.1. Emulación

La emulación se basa en simular el comportamiento de un hardware distinto. Este tipo de virtualización es la más costosa y la menos eficiente, ya que simular el hardware implica que cada instrucción que se ejecute debe ser traducida al hardware real. La Figura 2 muestra las distintas capas que permiten la emulación.

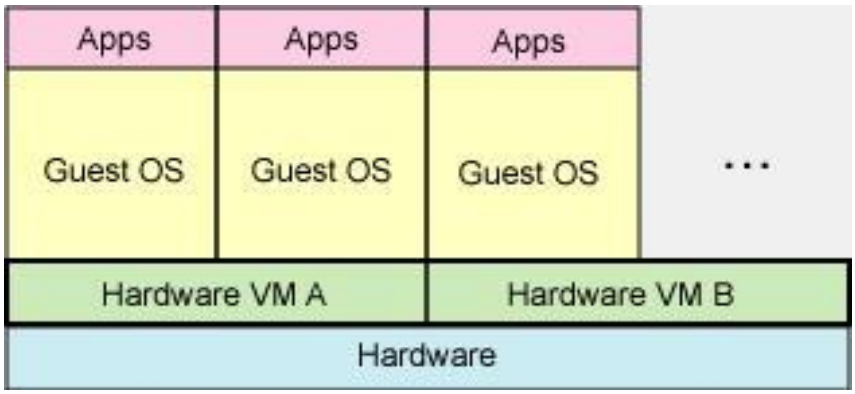

Fig.2: Emulación.
Fuente: http://arcos.inf.uc3m.es/~folcina/pfc-html/node16.html

Sin embargo la emulación es útil en el desarrollo de firmware dado que se puede empezar a desarrollar sin tener el hardware real.

2.1.2. Virtualización completa

Este tipo de virtualización permite ejecutar distintos sistemas operativos invitados, sobre un sistema operativo anfitrión, para ello se utiliza en medio un hypervisor que permite compartir el hardware real entre los distintos sistemas operativos en ejecución. Esta capa intermedia,

denominada hypervisor, gestiona las instrucciones protegidas de acceso al hardware. La Figura 3 muestra la interrelación entre un hypervisor y el resto de capas que permiten la virtualización completa.

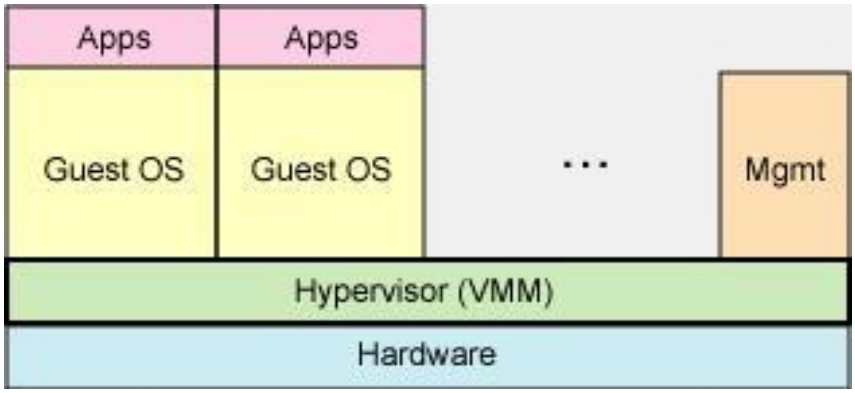

Fig.3: Virtualización completa.
Fuente: http://arcos.inf.uc3m.es/~folcina/pfc-html/node16.html

En esta virtualización el rendimiento está muy por encima de la emulación pero siempre va a ser menor que de forma nativa dado que el hypervisor gestiona el hardware. Sin embargo, las plataformas x86 fabricadas por Intel y AMD, con Intel VT y AMD-V, fueron diseñadas para optimizar la virtualización y permitir un gran acercamiento al rendimiento nativo.

2.1.3. Paravirtualización

La paravirtualización mejoró la virtualización completa dejando que los sistemas virtualizados, adaptados al hypervisor previamente, pudieran interactuar con los componentes hardware esenciales (procesador, RAM, memoria, etc…) de forma nativa y sin la gestión directa del hypervisor, esto acerca a los sistemas operativos al rendimiento nativo. La Figura 4 muestra las modificaciones realizadas sobre los sistemas operativos invitados, respecto de la Figura 3 para realizar una paravirtualización.

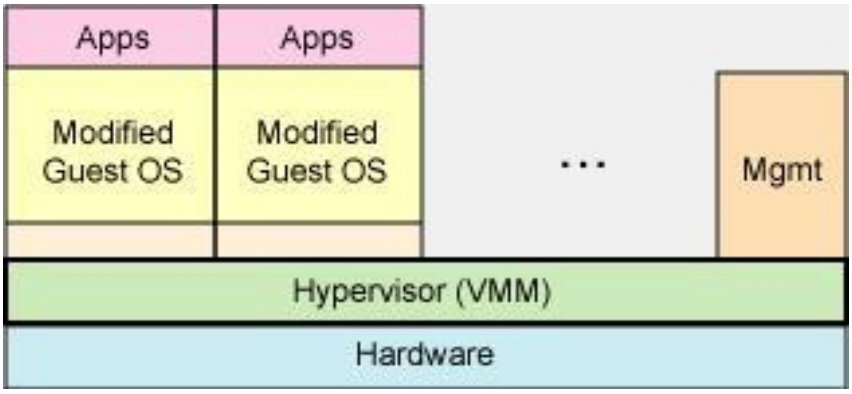

Fig.4: Paravirtualización.
Fuente: http://arcos.inf.uc3m.es/~folcina/pfc-html/node16.html

Xen es uno de los hypervisores más destacados en paravirtualización usando sistemas operativos invitados modificados. Por otro lado, VmWare, Virtualbox y KVM son unos de los

hypervisores más destacados en virtualización completa. Finalmente, en el apartado de emulación el más destacado es QEMU.

2.1.4. Usos de la virtualización

En la actualidad, se pueden dar distintos usos a la virtualización de sistemas. A continuación, se detallan algunos aspectos que permite la virtualización:

- **Recursos de los servidores**: normalmente, los servidores están desaprovechados durante la ejecución de servicios, con la virtualización se pueden montar varios sistemas o servicios aislados entre sí sobre el mismo servidor y así aprovechar mejor los servidores, reduciendo su espacio físico y el consumo de energía.
- **Desarrollo multiplataforma**: se puede desarrollar una aplicación para diferentes sistemas operativos.
- **Plataformas obsoletas**: si se migran antiguas plataformas a sistemas virtualizados, se podrá actualizar el hardware y seguir manteniendo las plataformas antiguas.
- **Seguridad**: es frecuente separar los servicios ofrecidos en una red en distintas MVs por razones de seguridad.
- **Variedad de sistemas operativos**: permite instalar diferentes sistemas operativos en un equipo, ya sea para probarlos, utilizarlos como *sandbox*, instalar diferentes softwares, etc.

2.2. Enfoque práctico de las máquinas virtuales

Uno de los primeros en introducir las MVs fue VmWare Workstation, esta tecnología fue tomada a la ligera y en un principio fue solo utilizada para ejecutar varios sistemas operativos en un mismo entorno sin la necesidad de arranques duales. La tecnología quedó estancada por los departamentos de Tecnología de la Información que se guiaban por el dogma de que cada aplicación o servicio debía tener un único sistema operativo. Esto llevó al aumento de servidores en los centros de datos, hasta el punto de que las organizaciones estaban literalmente acabando el espacio, además de un elevado consumo energético, por lo que la construcción de un centro de datos era demasiado costoso.

Por ello, se ha visto un incremento de la proporción de MVs en relación a equipos físicos a medida que cada año se encuentran servidores más potentes y mucho más rentables que un centro de datos. Últimamente se están trasladando los centros de datos a la nube, en lo que respecta a software, se pueden encontrar algunos como OpenStack [14] y vSphere [13] que trabajan en la nube y gestionan su propio hypervisor. Entre sus funcionalidades se pueden destacar:

- Crea, elimina y monitoriza las MVs de su entorno. Por ejemplo, se puede observar en la Figura 5 la creación de una máquina virtual con el hypervisor OpenStack.
- Gestión de los recursos como son la memoria, núcleos de procesamiento, espacio en disco e interfaces de red. Se puede observar en la Figura 5 en “Límites del proyecto”.
- Conexión a máquinas virtuales mediante ssh y vnc. Se puede observar en la Figura 6 como se asocia un par de claves.

- Crea, restaura y elimina snapshots.
- Capacidad de apagar, encender, reiniciar y suspender las máquinas virtuales.
- Creación de redes locales donde se pueden interconectar las máquinas virtuales. Se puede observar en la Figura 7 y su topología en la Figura 8.

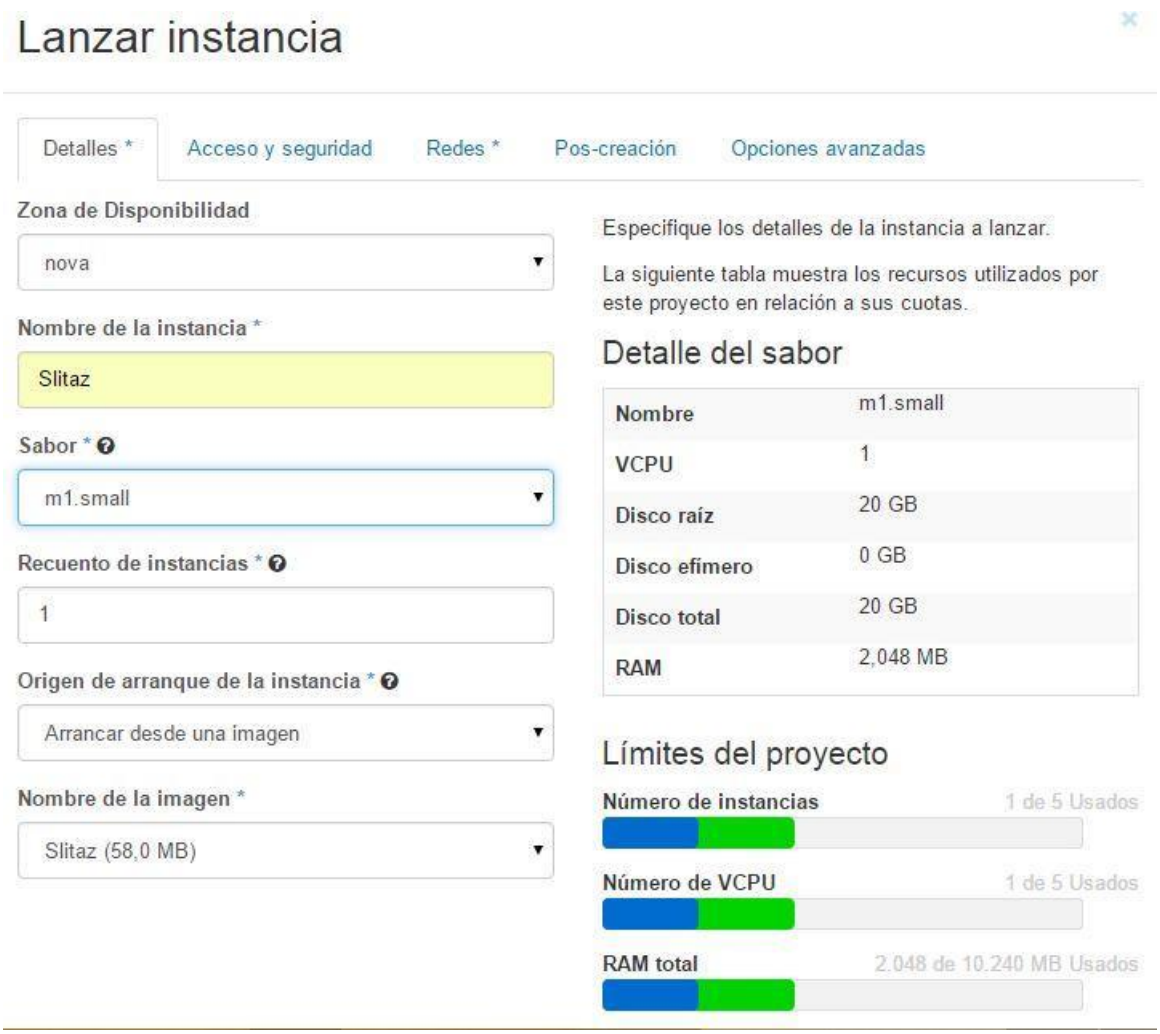

Fig.5: Creación de una MV en OpenStack.

Fig.6: Asignar pares de claves a una MV en OpenStack.

Fig.7: Selección de las redes a interconectar de una MV en OpenStack.

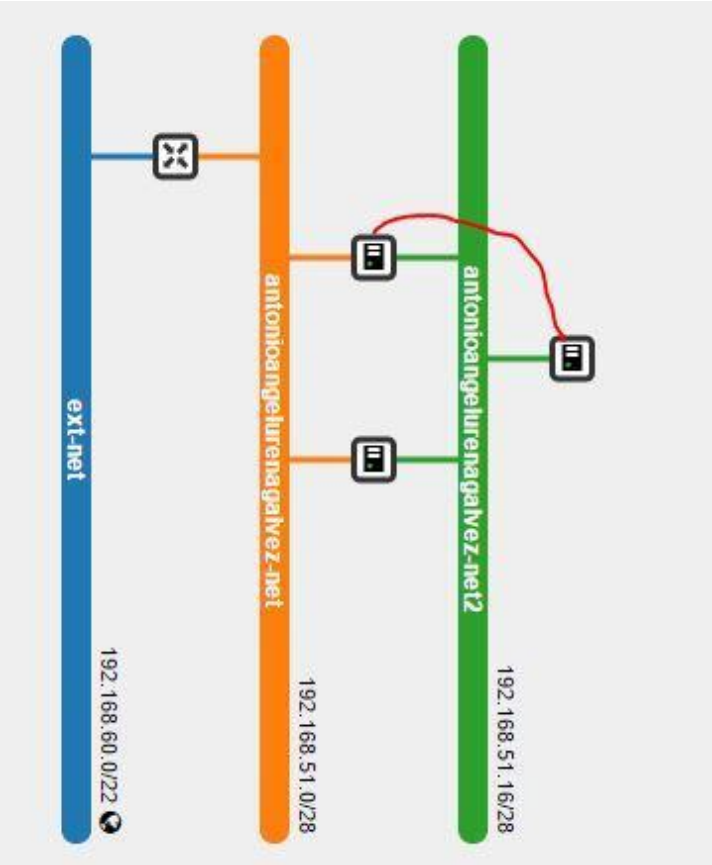

Fig.8: Topología de red creada en OpenStack.

2.3. Hypervisores

Existen multitud de hypervisores (vSphere, VmWare, VirtualBox, KVM, Xen, QEMU, Citrix, Oracle VM, Microsoft Hyper-V, etc) que permiten la gestión de MVs y de entre todas las acciones posibles, las más comunes son:

- Encenderla
- Apagarla
- Suspenderla
- Guardar estados de la MV
- Restaurar un estado anterior
- Eliminar un estado anterior
- Modificar memoria RAM
- Modificar memoria del disco
- Crear una MV (instalando un sistema operativo)
- Destruir una MV
- Crear/eliminar/gestionar interfaces de red
- Crear discos duros virtuales
- Eliminar discos duros virtuales

Normalmente, el administrador ejecuta estas acciones a través del interfaz gráfico del hypervisor correspondiente. Sin embargo, también se puede realizar vía comandos en una consola o terminal. Este último escenario es el que utilizará el hypervisor centralizado para comunicarse con el resto de hypervisores. Por lo tanto, cada una de estas acciones requiere de la ejecución de scripts y comandos parametrizados que permitan interactuar con el hypervisor correspondiente. Una de las partes críticas de este TFG ha sido la confección de los scripts y comandos parametrizados. Estos scripts y comandos parametrizados se encuentran definidos en la base de datos de la aplicación web.

A continuación, se muestran los scripts y comandos parametrizados para interactuar con el hypervisor de VirtualBox [11].

- Script para Windows de búsqueda de MVs con VirtualBox y formateo de la información para su envío al servidor:

```
//Se deshabilita echo por lo que se mostrará solo la información relevante
echo off
//se obtiene una lista de las MVs de VirtualBox y pasan por un bucle para obtener de cada equipo su información.
FOR /F "tokens=1 delims={" %r IN ('VBoxManage list vms')
DO (
//se escribe ---mv--- en consola para diferenciar las diferentes máquinas virtuales
        echo "---mv---"
        //se guarda en la variable namevm el nombre de la MV
        set "namevm=%r"
        //se escribe el nombre de la MV formateado
        call echo %^namevm:~0,-1%
        //se escribe rutaMV porque VirtualBox no necesita conocer su archivo de
        //configuración para ejecutar algún comando pero otros hypervisores como
        //VmWare si lo necesitan
        echo rutaMV
        //Se formatea con un bucle el valor total de memoria en el equipo
        FOR /F "tokens=2 delims==" %a IN ('wmic memoryChip get Capacity /value')
        //se escribe el valor total de la memoria del equipo
        DO (echo %a)
        //se filtra la información general de la MV por Memory para obtener el valor de la memoria asignada
        FOR /F "tokens=*" %a IN ('VBoxManage showvminfo %r ^| findstr "Memory"')
        DO (
                //se formatea el valor de la memoria de la MV con un bucle
                FOR /F "tokens=3 delims= " %b IN ("%a")
                DO (
                        //se formatea y escribe el valor de la memoria de la MV en MB
                        FOR /F "tokens=1 delims=M" %c IN ("%b")
                        DO (echo %c MB)))
        // se filtra la información general de la MV por State para obtener el estado
        FOR /F "tokens=*" %a IN ('VBoxManage showvminfo %r ^| findstr "State"') DO (
                //se formatea en un bucle el valor del estado
                FOR /F "tokens=2 delims=:" %b IN ("%a")
                DO (
                        //se formatea en un bucle y se escribe el valor del estado
                        FOR /F "tokens=1 delims=(" %c IN ("%b")
                        DO (set "estado=%c" & call echo %^estado: =%)
```

```
		)
	)
	//se obtiene la ruta del disco duro virtual
	FOR /F "tokens=*" %a IN ('VBoxManage showvminfo %r ^| findstr ".vdi"')
	DO (
		//se formatea con un bucle
		FOR /F "tokens=2 delims=(" %b IN ("%a")
		DO (
		//se formatea y escribe la ruta del HDD
			FOR /F "tokens=2 delims=)" %c IN ("%b")
			DO (
				set "hdd=%c"
				call echo %^hdd:~2,-1%
				//filtra la información del HDD virtual por Capacity y Size
				FOR /F "tokens=2 delims=:" %e IN ('call VBoxManage showhdinfo
				"%hdd:~2,-1%" ^| findstr "Capacity Size"')
				 DO (
					//se formatea y escribe la capacidad y ocupación
					del hdd virtual en MB
					FOR /F "tokens=1 delims= " %f IN ("%e")
					DO (echo %f MB)
				)
			)
		)
	)
	//se formatea y escribe el espacio libre del disco duro físico en Bytes
	FOR /F "tokens=3 delims= " %a IN ('dir ^| findstr "dirs" ^| findstr "bytes"')
	DO (set "libre=%a" 1>nul & call echo %^libre:.=% Bytes)
	//se escribe galería de snapshots para posteriormente filtrar los snapshots
	echo galería de snapshots
	//se formatea y se escribe una lista de los snapshots
	FOR /F "tokens=2 delims=:" %a IN ('VBoxManage snapshot %r list ^| findstr -v "not any"')
	DO (set "snap=%a" & call echo %^snap:~1,-6%)
)
```

La Figura 9 muestra la salida de este script cuando se ejecuta y si se encuentra alguna MV de VirtualBox. Como se puede observar la información de salida del script tiene el siguiente significado:

- ✓ Nombre MV
- ✓ Ruta configuración MV
- ✓ Cantidad total de RAM física en Bytes
- ✓ Memoria RAM asignada en MB
- ✓ Estado de la MV
- ✓ Ruta del disco duro virtual
- ✓ Capacidad del disco duro virtual
- ✓ Espacio ocupado del disco duro virtual
- ✓ Espacio libre del disco duro físico
- ✓ Galería de snapshots

Fig.9: Salida datos MVs VirtualBox.

- Comando Windows para encender la MV con VirtualBox:

  VBoxManage startvm ##nombre##

  Las almohadillas son para parametrizar el comando y buscar el nombre de la máquina virtual en la que se ha realizado la acción.

- Comando Windows para apagar la MV con VirtualBox:

  VBoxManage controlvm ##nombre## poweroff

- Comando Windows para reiniciar la MV con VirtualBox:

  VBoxManage controlvm ##nombre## reset

- Comando Windows para hacer una snapshot de la MV con VirtualBox:

  VBoxManage snapshot ##nombre## take ##valor##

  Valor es el parámetro por defecto si el usuario tiene que escribir un *valor*, en este caso *valor* equivale al nombre que se le quiere poner a la snapshot.

- Comando Windows para restaurar una snapshot de la MV con VirtualBox:

  VBoxManage snapshot ##nombre## restore ##valor##

- Comando Windows para eliminar una snapshot de la MV con VirtualBox:

  VBoxManage snapshot ##nombre## delete ##valor##

- Comando Windows para modificar el tamaño de la memoria de la MV con VirtualBox:

  VBoxManage controlvm ##nombre## poweroff & VBoxManage modifyvm ##nombre## --memory ##valor##

  En este caso para modificar el tamaño de la memoria se necesita que la MV esté apagada y el *valor* equivale al número de MB de la RAM que quiere asignar.

- Comando Windows para aumentar la capacidad del disco duro de la MV con VirtualBox:

```
VBoxManage controlvm ##nombre## poweroff & VBoxManage modifyhd "##rutaHDD##" --resize ##valor##
```

  Como se puede observar su correcto funcionamiento depende de que la MV esté apagada, aquí se observa otro de los parámetros (*rutaHDD*), el cual se sustituirá por la ruta del disco duro virtual de la MV que ha ejecutado la acción.

En este apartado se han mostrado distintos scripts y comandos parametrizados para interactuar con el hypervisor VirtualBox. Sin embargo, el lector puede profundizar en el diseño de scripts y comandos parametrizados para interactuar con otros hypervisores (ver Anexo del TFG).

3. Herramientas y materiales

Para el desarrollo del TFG, se necesita hacer uso de herramientas que nos faciliten el progreso. A continuación se explicarán las más relevantes:

1. Sistema operativo Windows 8.1: se dispone de un equipo portátil con este sistema operativo.

2. Editor de texto Atom: se utilizará este editor de textos pues con el *plugin* adecuado, facilita mucho la tarea de escribir código en *Ruby*.

3. Editor de texto Aptana Studio 3: es el editor de textos estrella para aplicaciones web, ya que tiene ayudas para *HTML*, *CSS*, *JavaScript*, *PHP* y *RubyOnRails*.

4. Google Chrome y Firefox: para visualizar el progreso de la aplicación web.

5. Gitlab como control de versiones: se ha preferido *gitlab* porque es un sistema de control de versiones distribuido y porque no requiere de pago para hacer proyectos privados.

6. Visual Paradigm para modelado UML: herramienta usada en clase para modelado UML.

7. Balsamiq para esbozo de interfaces: herramienta usada en clase para esbozar la interfaz de usuario.

8. Servidor Ubuntu en Windows Azure: en el que hacer pruebas y *deploy*: se ha preferido porque se ha usado en distintas asignaturas de la titulación y se utilizará para el *deploy*.

9. Jenkins: para automatización de las pruebas y el deploy: se podría haber usado *Travis*, pero se ha elegido Jenkins porque se ha usado en distintas asignaturas de la titulación.

10. Nginx como servidor web: se podría haber usado *Apache*, pero se ha elegido *Nginx* porque se ha usado en distintas asignaturas de la titulación.

3.1. Lenguajes de programación y frameworks

3.1.1. Servidor

Últimamente se está popularizando el uso de Node.js y PHP para el desarrollo de la parte servidor en aplicaciones web, por una serie de motivos:

- Node.js es más eficiente que otras alternativas.
- Node.js es asíncrono, por lo que soporta mayor número de conexiones y las llamadas no bloquean el servidor.
- JavaScript es un lenguaje conocido por desarrolladores de aplicaciones web.
- PHP es un lenguaje que permite mezclar código PHP con HTML de forma sencilla.
- PHP está hecho para coexistir con MySQL y sus variantes, lo que permite una rápida gestión de una base de datos.
- PHP es el lenguaje más conocido por desarrolladores de aplicaciones web y tiene una gran comunidad.

Sin embargo, el lenguaje de programación escogido para el desarrollo del servidor en este trabajo es Ruby, por las siguientes razones:

- Experiencia previa con el lenguaje.
- Su código abierto está enfocado a la simplicidad y a la productividad.
- Es un lenguaje interpretado.
- Multiplataforma.
- Orientado a objetos.
- Su sintaxis se siente natural al leerla y fácil al escribirla.
- El framework utilizado es asíncrono.
- Se puede mezclar código Ruby con HTML.
- Coexiste y separa en funciones las consultas a cualquier base de datos (independientemente de la base de datos se usarán las mismas funciones).
- Mayor facilidad a la hora de operar con los datos, ya que es más legible que JavaScript y PHP.

Entrando en detalle en los frameworks y utilidades de Ruby, se utilizarán los siguientes:

- Ruby On Rails como servidor web asíncrono, donde se gestionaran las peticiones de la aplicación cliente, gestión de la información de las MVs y usuarios, así como de los comandos parametrizados y asignación de estos a los equipos.
- Rake (gema de Ruby) como gestor de tareas y automatización de procesos.
- Bundler (gema de Ruby) como gestor de dependencias de Ruby.

Finalmente, la parte servidora es la que se encargará de controlar las peticiones realizadas por los clientes, aceptándolas o rechazándolas en caso de cualquier tipo de error. En este tipo de peticiones se incluyen las consultas a la base de datos y la interacción con los diferentes controladores que se explicará en los apartados 4.1.3. Diseño del servidor (BackEnd) y 4.1.4. Interacción cliente (FrontEnd)-servidor (BackEnd).

3.1.2. Clientes

Para la parte cliente (FrontEnd) de la aplicación web, será necesario el uso de JavaScript, además de HTML5 y CSS3 para la aplicación web y las siguientes utilidades:

- jQuery: librería de JavaScript que se utilizará para dibujar gráficas, hacer texto y código HTML flotante.
- Bootstrap: framework para desarrollo *responsive*, esto permitirá que la aplicación se adapte al dispositivo según la resolución y con un estilo adecuado.

El diseño de la parte cliente (FrontEnd) puede verse esbozado y completado en los apartados 4.1.5. Diseño de la interfaz y 4.1.7. Resultado, así como su interacción con la parte servidor (BackEnd) en el apartado 4.1.4. Interacción cliente (FrontEnd)-servidor (BackEnd).

Para la aplicación cliente que interacciona con el servidor como una aplicación cliente/servidor, se utilizará Ruby y las siguientes utilidades:

- net/http: para realizar peticiones al servidor.
- gtbindings: para la interfaz gráfica.

La aplicación cliente será la encargada de monitorizar el estado de las MVs del sistema operativo anfitrión y realizar peticiones sobre la aplicación web. Los métodos que interaccionan con el servidor pueden verse en 4.2.1. Requisitos del sistema y 4.2.2. Diseño de la aplicación.

4. Hypervisor centralizado

En este TFG se va a desarrollar una aplicación cliente/servidor, para solucionar el problema descrito en 1.1. Objetivos, para lo cual se separa el desarrollo de la aplicación en Aplicación web y en Aplicación cliente.

4.1. Aplicación web

4.1.1. Requisitos del sistema

El análisis de los requisitos es un proceso de formulación de los conceptos del sistema a desarrollar. Es esencial realizar este paso antes de proseguir, pues la mayoría de los defectos se pueden detectar en esta fase. A continuación, se procede a detallar cada uno de los requisitos que se debe tener en cuenta:

Se requiere el desarrollo de una parte cliente (FrontEnd) y otra servidora (BackEnd), en JavaScript, HTML 5, CSS3 y Ruby, respectivamente.

La interconexión entre dichas partes se realizará mediante el uso del framework Ruby On Rails que permitirá la interacción del cliente con el servidor y este con las peticiones a la base de datos.

El *FrontEnd* se encargará de mostrar y realizar las solicitudes sobre la gestión de las MVs. Esta interfaz permitirá al usuario realizar los cambios que se consideren oportunos sobre las MVs, como encenderla, apagarla, reiniciarla, hacer un snapshot, restaurar un snapshot, eliminar un snapshot, modificar la RAM o la capacidad del disco duro de la MV y conectarse por VNC, solo aquellos que tengan su sistema configurado para ello.

El usuario podrá también entrar en su perfil y generar una secret key nueva si cree que su cuenta ha sido comprometida, así como cambiar la contraseña de su cuenta o eliminar por completo la cuenta.

La secret key es una clave aleatoria asignada al usuario tras su registro, esta secret key será utilizada para autenticar al usuario desde fuera de la aplicación web, o sea para autentificarlo en la aplicación cliente, se utiliza la secret key en vez de usuario y contraseña desde fuera de la aplicación web para no enviar el usuario y la contraseña por la web, pues un sniffer podría obtener sus credenciales. Por tanto, con la implementación de la *secret key* se pretende dar cierto grado de seguridad al usuario.

No se permite modificar ni el nombre del usuario, ni el email una vez registrado, por ello si se ha equivocado en alguno de estos datos deberá eliminar la cuenta y hacerse otra, pues no puede haber ni nombres de usuario ni email repetidos.

También podrá observar los nombres de los equipos que contienen las MVs para una mejor organización de las MVs y si se selecciona alguno de estos equipos tendrá acceso a cada una de las MVs disponibles en dicho equipo.

Se mostrará una lista con las MVs que posee dicho equipo en las que como resumen se mostrará el hypervisor de la MV, nombre de la MV, RAM de la MV, espacio ocupado del disco duro virtual / espacio total del disco duro virtual, estado de la MV y gestión de la MV y botón de actualizar MV en caso de que se haya modificado la MV directamente y no se reflejen los cambios en la web.

Además, es necesario el botón actualizar para no saturar la aplicación web por las peticiones que envía la aplicación cliente y para no saturar el equipo donde se ejecute la aplicación cliente, pues los comandos que se ejecutan para buscar las MVs tardan entre 20 y 90 segundos.

Una vez estamos gestionando una MV, se tienen que poder realizar las siguientes acciones:

- Apagar MV seleccionada.
- Encender MV seleccionada.
- Reiniciar MV seleccionada.
- Hacer una snapshot de la MV seleccionada.
- Restaurar una snapshot de la MV seleccionada.
- Eliminar una snapshot de la MV seleccionada.
- Conectarse por VNC a la MV seleccionada si está configurada para ello.

Dichas acciones requieren una orden enviada al hypervisor correspondiente en función de un comando parametrizado.

La parte servidora es la que se encargará de controlar las solicitudes de cambios realizados por los clientes, aceptándolos o rechazándolos en caso de cualquier tipo de error.

4.1.2. Diseño de la base de datos

Tras los requisitos del sistema descritos anteriormente, se ha creado un diagrama entidad-relación para la base de datos de esta aplicación web, teniendo en cuenta las siguientes tablas:

Usuario

- ID
- Username
- Password
- Email
- Auth/secretKey
- Resetpassword

Maquina (Anfitrion)

- ID
- Nombre/hostname
- Sistema operativo

Orden

- ID
- Comando

MáquinaVirtual (host)

- ID
- Nombre
- Estado
- Sistema operativo
- Opciones (aquí se pondrán todos los atributos extras de la MV)
- Hypervisor
- VersionHypervisor

Snapshot

- ID
- Nombre
- DirecciónArchivo

Comando

- ID
- Nombre
- ComandoParam
- Hypervisor
- Sistema operativo

Con las asociaciones quedaría como se muestra en la Figura 10, donde se puede observar que:

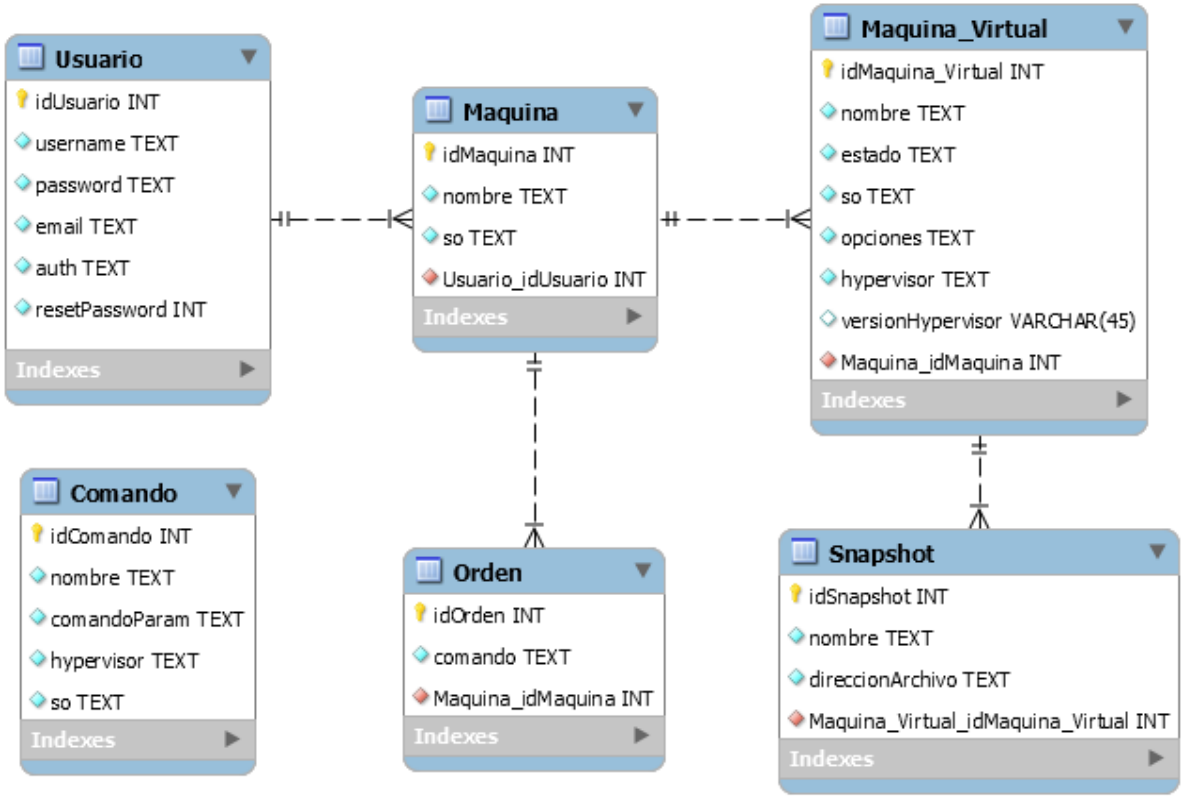

Fig.10: Diagrama Entidad-Relación.

- Un usuario puede tener muchas máquinas.
- Una máquina o equipo puede tener distintas MVs.
- Una máquina o equipo puede tener ordenes (acciones).
- Una MV puede tener snapshots.
- Si un comando no tiene asociación, se buscará en su tabla el comando parametrizado adecuado.

4.1.3. Diseño del servidor (BackEnd)

Una vez obtenida una especificación del sistema, el siguiente paso es el diseño del sistema, que se apoyará con la creación de diagramas UML. UML es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema.

Es importante un diseño exhaustivo en el que se contemplen todos los aspectos y comportamientos, ya que es más fácil de arreglar cualquier problema en esta fase que durante la implementación, ahorrando posteriormente más tiempo, para ello se utilizarán diagramas de clases.

Diagramas de clases: se desarrollarán en función del lenguaje de programación elegido, en el que se definen las clases que habrá que implementar y los atributos y métodos que deben contener. Se tendrá que tener en cuenta todo lo que debe hacer el sistema para que sea lo más específico posible.

Se ha creado un diagrama de clases para la parte servidora (BackEnd) que se puede observar en la Figura 11.

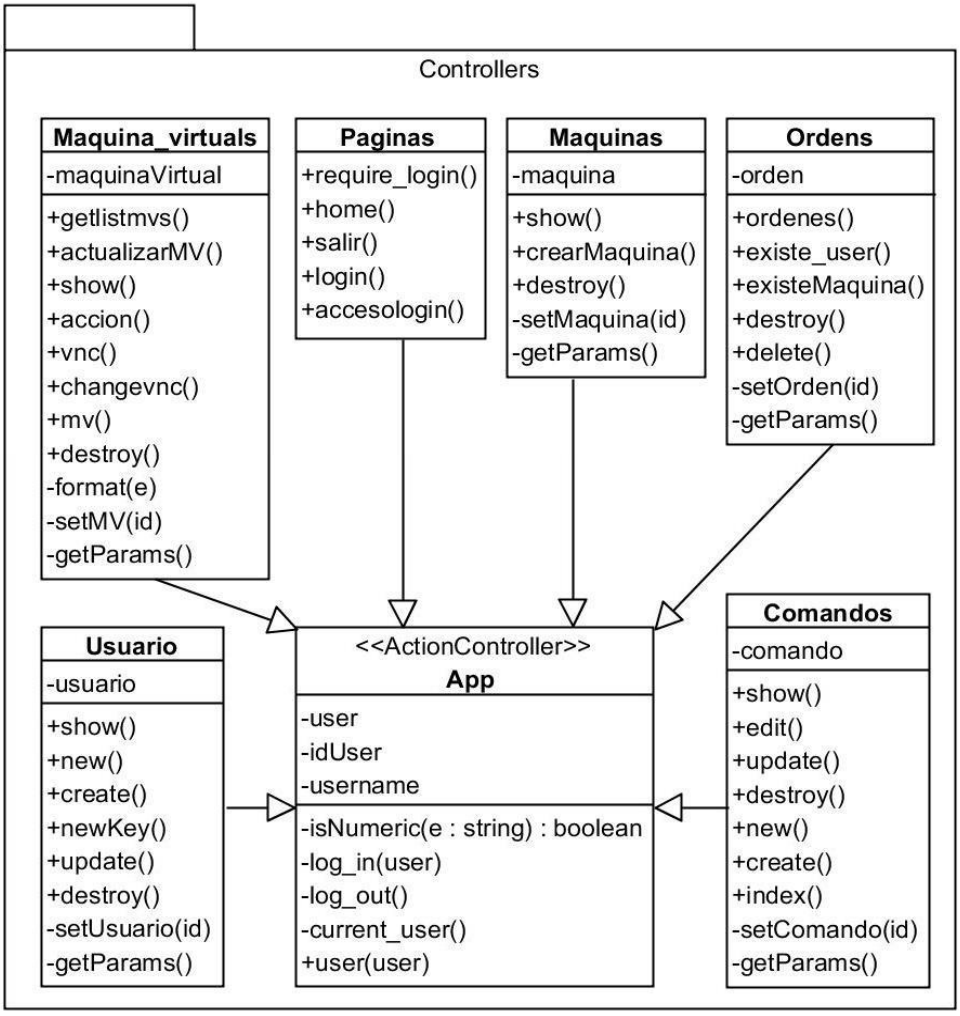

Fig.11: Diagrama de clases controladores del servidor.

La clase "*App*" es un actionController de Rails, que es una clase de Rails utilizada para implementar controladores en una arquitectura Modelo Vista-Controlador (MVC). En ella, se implementan los métodos de las acciones a realizar cuando un usuario se conecta, se mantiene la sesión o se desconecta.

Cuando un usuario realiza una conexión, se crea un objeto "*usuario*", donde se almacena la información relativa al usuario en la base de datos y puede hacer las siguientes operaciones. Si el usuario todavía no existe y hay que registrarlo tiene permitido usar los métodos *new* y *create*, una vez ya está registrado si inicia sesión utilizará el método de

"paginas accesologin" y si el email y la contraseña son correctos se utilizará la operación log_in(user). Lo cual le permitirá ver su perfil con la operación *show* que llamará a la vista correspondiente, dentro de su perfil tendrá la opción de generar una *secret key* aleatoria que llamará a la operación *newKey*, también tendrá permitido cambiar la contraseña para lo cual llamará al método *update* y por último si desea eliminar la cuenta llamará al método *destroy*.

Si el usuario registrado utiliza su *secret key* en la aplicación cliente, esta empezará a enviar información relevante como el nombre de su equipo y el reconocimiento de las MVs. Entonces ya estará el usuario ligado con los equipos, los cuales podrán crearse *(crearMáquina)* automáticamente desde la aplicación cliente o destruirse *(destroy)* por orden del usuario.

En el caso de que el usuario tenga "MVs" estas se asignaran a su "equipo" correspondiente pudiendo automáticamente desde la aplicación cliente realizar las siguientes operaciones:

- *getlistmvs*: devuelve una lista con las MVs asignadas a un equipo.
- *mv*: la aplicación cliente envía la información recolectada a este método para el tratamiento de los datos de cada MV, de tal forma que si existe, la actualiza, y si no existe, la crea en la base de datos para su gestión.
- *destroy*: la aplicación cliente elimina automáticamente las MVs que se vean reflejadas en el servidor pero no se encuentran en la fase de reconocimiento de las MVs.

Por otro lado el usuario podrá realizar las siguientes operaciones:

- *actualizarMV*: contacta con la aplicación cliente para que esta realice un reconocimiento del sistema y envía la información recolectada al servidor.
- *show*: muestra información de la MV seleccionada para poder hacer acciones con ella.
- *acción*: envía una acción que busca el comando parametrizado correspondiente y este generará una orden adecuada para la MV correspondiente, la cual será ejecutada por la aplicación cliente.
- *vnc*: establece una conexión *vnc* con la MV seleccionada, siempre y cuando dicha MV esté configurada para ello.
- *changevnc*: cambia los parámetros de *ipvnc* y *portvnc* para conectar con la MV seleccionada.

Si el usuario hubiera realizado una acción sobre una MV esta llamaría a la clase o controlador *"Comandos"* y posteriormente a la clase *"Ordens"*

De la clase *"Comandos"* (solo accesible para usuarios con privilegios de administrador) obtenemos los métodos:

- *index*: el cual llamará a la vista *comandos/index.html.erb* y será utilizada por la aplicación cliente para captar aquellos comandos de reconocimiento de MVs.
- *show*: muestra un comando en específico, solo disponible para el administrador de la página.
- *edit*: llama a la vista *comandos/edit.html.erb*, la cual solo está disponible para el administrador y se muestra un formulario para pasar los datos cambiados al método *update*.
- *update*: actualizará la información de la base de datos.

- *destroy*: elimina un comando específico, solo está disponible para el administrador.
- *new*: llama a la vista *commandos/new.html.erb*, la cual solo está disponible para el administrador y dentro se encuentra un formulario para crear un nuevo comando llamando al método *create*.
- *create*: creará un nuevo comando en la base de datos.

De la clase *"Ordens"* obtenemos que los métodos utilizados por la aplicación cliente serán los siguientes:

- *ordenes*: la aplicación cliente obtiene las órdenes correspondientes a la máquina del usuario y ejecuta dichas órdenes.
- *existe_user*: la aplicación cliente hace una petición al servidor y determina si existe el usuario según la *secret key* del usuario.
- *existeMáquina*: comprueba que el equipo en el que se está ejecutando la aplicación cliente se encuentra en la base de datos, si no es así la crea para empezar el reconocimiento de las MVs.
- *delete*: una vez ejecutada la orden, la aplicación cliente hace una petición para eliminarla de la base de datos.

Se debe resaltar que por parte del usuario tan solo podrá eliminar un comando que haya ejecutado con el método *"destroy"*.

Finalmente el usuario podrá salir de su sesión usando el método de *"páginas salir",* al hacer esto será redirigido a *"páginas home"* y mostrará la página principal.

4.1.4. Interacción cliente (FrontEnd)-servidor (BackEnd)

Para diseñar la lógica del cliente primero se debe conocer el funcionamiento de la arquitectura MVC del framework de Ruby *"RubyOnRails"* [5], en dicho framework siempre que se crea un proyecto se genera por defecto la clase *"Application"* como controlador, dicha clase ya está vinculada en la vista *(FrontEnd)* como layout, dado que los demás controladores que se vayan creando heredan de *"Application"* esta vista es adecuada para crear la cabecera y el pie de página que serán comunes en todas las páginas.

Para que la parte servidor (los controladores) y la parte cliente (las vistas) se comuniquen entre sí, hay varias opciones:

- Interacción *content_for*, *yield*: usados en la aplicación web para gestionar la cabecera y el menú de la izquierda (*layout*), pues van adaptándose dinámicamente, y al ser comunes en todas las páginas se consigue un renderizado del contenido eficiente.
- Interacción métodos POST y GET: utilizados para el contenido dinámico de las diferentes páginas excepto la parte común (cabecera y menú de la izquierda).

Se va a empezar por la interacción *content_for, yield*, el objetivo es manipular la vista de *"Application"*, para ello es necesario crear unos métodos en unos archivos llamados *helpers*.

Estos archivos *helpers* son accesibles tanto desde los controladores como desde las vistas. Esto hace que sea útil por ejemplo, crear métodos que guarden la sesión del usuario conectado.

Una vez dentro de un archivo *helper*, se crea un método y para crear variables globales y accesibles desde cualquier vista, se asignan con *"content_for :nombreVariable, valorVariable"* para así poder llamarlas desde la vista con *"yield(:nombreVariable)"*.

En la lógica de *Rails,* cuando se asignan variables de esta forma, se almacenan en un diccionario, donde "*:nombreVariable*" es la key.

Este diccionario es accesible desde cualquier vista. Aquí se muestra un ejemplo y su salida.

En */helpers/hola_helper.rb* se escribe lo siguiente:

```
module HolaMundoHelper
        def hola (nombre)
        content_for :hola, "Hola "+nombre
        end
end
```

Ahora se incluye a hola_helper en un controlador, por ejemplo en */controllers/hola_controller.rb*

```
class HolaController < ApplicationController
          include HolaMundoHelper
          def home
          end
end
```

Aquí se puede observar cómo se incluye dicho método del helper el cual podrá ser llamado por la vista, como se observa hay un método llamado *home* que está vacío, al estar vacío llamará directamente a la vista *home* dentro de la carpeta *hola* en *views*, si no estuviera vacío haría las operaciones correspondientes antes de llamar a la vista y si se quiere utilizar una variable en la vista home habría que ponerla en dicho método.

En la vista */views/hola/home.html.erb* se observa lo siguiente:

```
<head>
 <title>Hola</title>
 <% hola "Antonio" %>
</head>
<body>
 <p><%=yield(:hola)%></p>
</body>
```

Una vez guardado todo se observa el resultado y como lo muestra Chrome en la Figura 12.

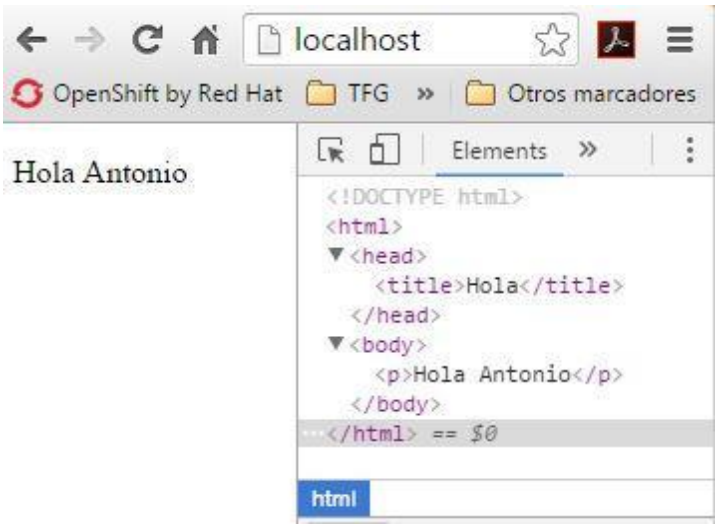

Fig.12: Resultado interacción *content_for*, *yield.*

Ahora se conoce cómo interactúa *Rails* con la arquitectura MVC, pero también se puede utilizar la interacción métodos POST y GET, para ello se pasan los datos necesarios por POST o por GET y se recogen al llamar a un controlador concreto.

Por ejemplo, si se llama a la url "localhost/Antonio" y se quiere obtener el mismo resultado que con el ejemplo anterior primero habrá que configurar *routes.rb* añadiendo esta línea:

```
get ':nombre'=> 'hola#home'
```

Lo que hace *routes.rb* es que con *get* hace una petición a *localhost* y con *:nombre* se le asigna cualquier valor pues *:nombre* actúa como un *key* de un diccionario, con "=>" se le dice que cuando se haga esta petición vaya al método *home* del controlador *hola*.

El controlador *"/controllers/hola_controller.rb"* queda así:

```
class HolaController < ApplicationController
        def home
        @hola="Hola "+params[:nombre]
         end
end
```

Con *"params"* se recoge el valor de *":nombre"* y con *"@"* antes de la variable hace que esta sea pública y se pueda llamar desde la vista.

Por tanto ahora la vista */views/hola/home.html.erb* tendrá la siguiente estructura:

```
<head>
 <title>Hola</title>
</head>
<body>
 <p><%=@hola %></p>
</body>
```

Y ahora se observa el resultado en la Figura 13:

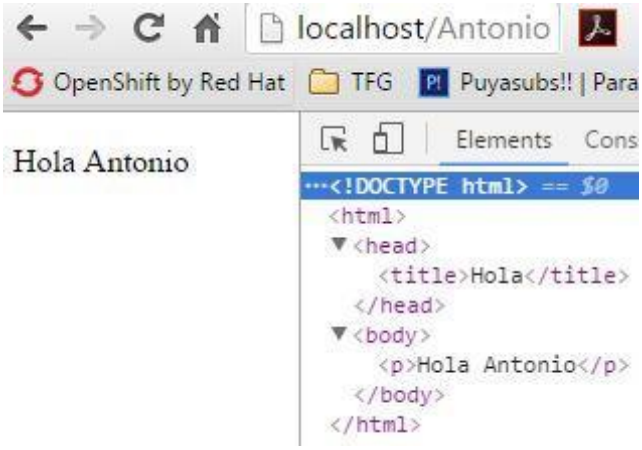

Fig.13: Resultado interacción métodos POST y GET.

Una vez aprendido como interaccionan el *FrontEnd* y el *BackEnd* de la aplicación web es momento de diseñar la interfaz (*FrontEnd*) para luego realizarla con HTML 5, JavaScript, CSS3 y Ruby para que interaccione con nuestro *BackEnd*.

4.1.5. Diseño de la interfaz

4.1.5.1. Esbozos de la interfaz

En primer lugar se ha creado un esbozo de la pantalla principal con la herramienta "Balsamiq", donde se puede observar en la Figura 14 una barra lateral a la izquierda que mostrará los equipos del usuario así como la página de inicio.

En la barra superior se encuentra a la izquierda un icono que al seleccionarlo ocultará o hará visible la barra lateral izquierda y a su lado el logo de la página. En la parte superior derecha se encontrará el nombre del usuario activo, al pulsar encima del nombre se desplegará un menú que contendrá el perfil del usuario y la opción de cerrar sesión (Salir). Justo al lado izquierdo del nombre de usuario, hay un icono que permite visualizar el estado de las acciones que se están ejecutando sobre las MVs. Todo lo anterior es lo que se llama cabecera y es común en toda la web.

En el centro de esta página principal se muestra información de cómo usar esta página, junto con la aplicación cliente.

En la Figura 15 se puede apreciar cómo será la página del perfil de usuario:

En la zona central se puede apreciar que el usuario en su perfil deberá tener al menos la información que ha proporcionado, como la de *nombre de usuario*, la cual se encuentra en Perfil *"nombre de usuario"*, la de *email*, así como la posibilidad de *cambiar de contraseña* (la cual requiere un mínimo de 8 caracteres con al menos 2 números y 2 letras) y *eliminar la cuenta* (se pedirá confirmación).

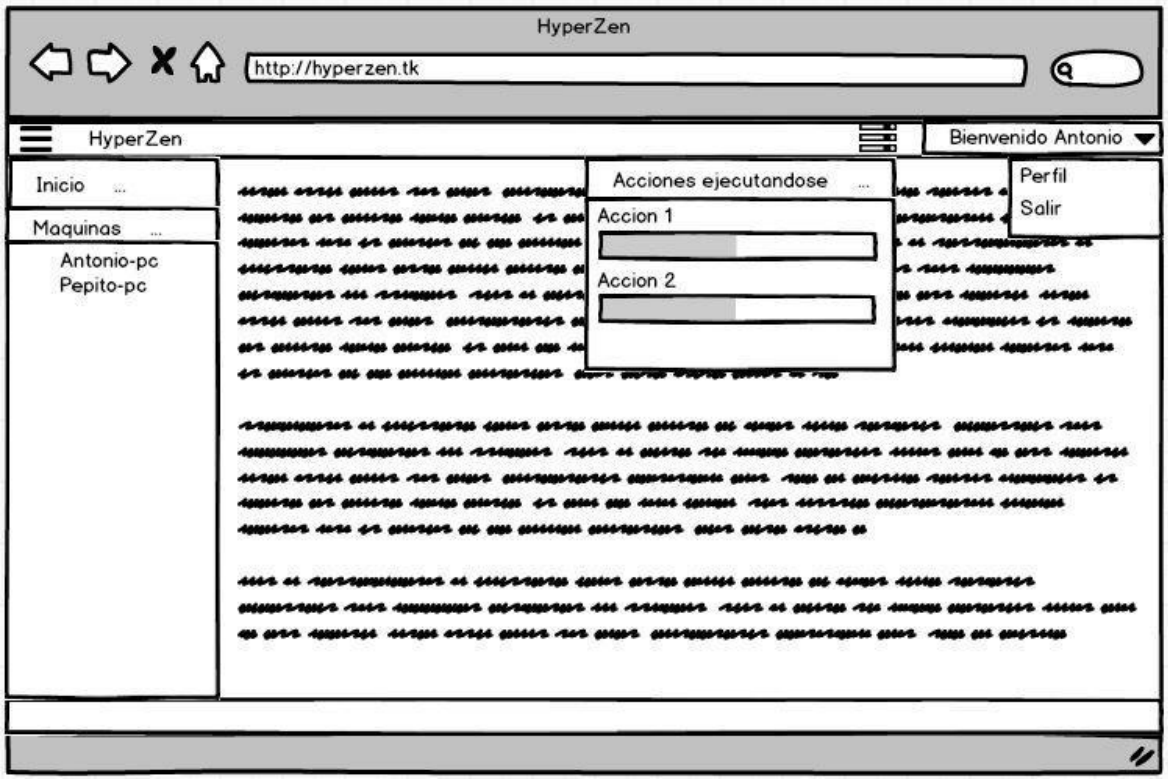

Fig.14: Esbozo de la pantalla principal.

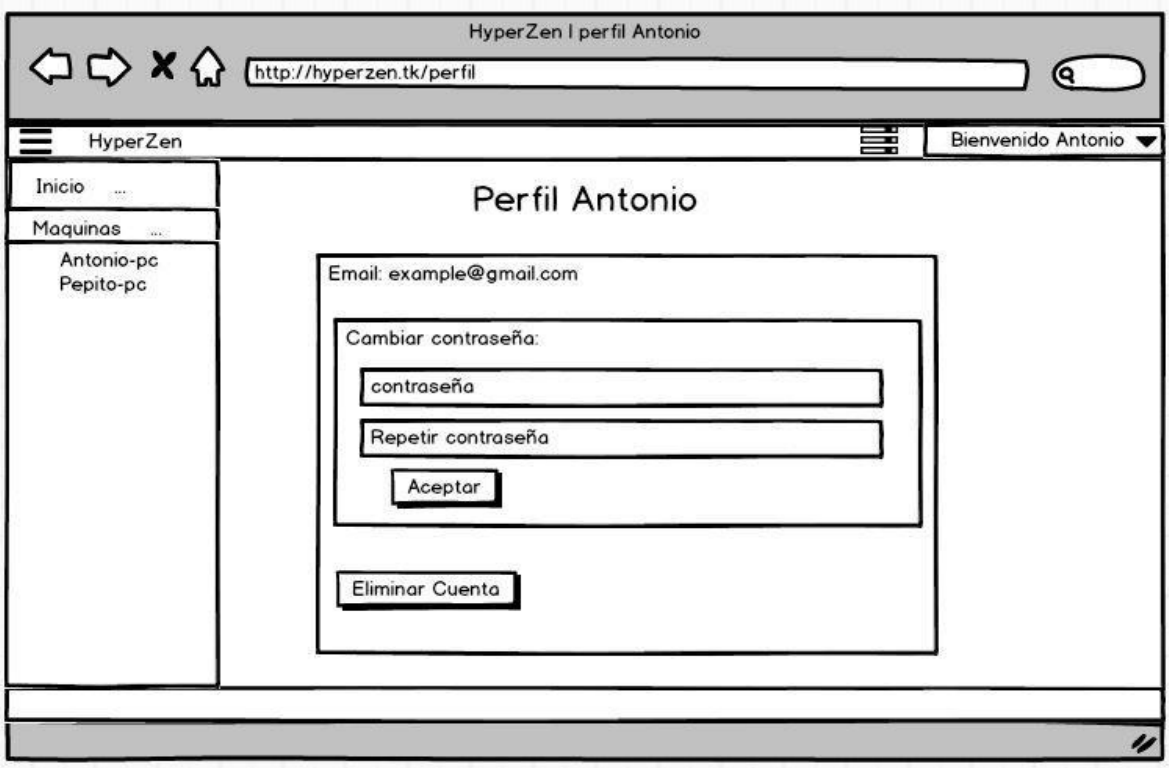

Fig.15: Esbozo de la pantalla perfil.

En la Figura 16 se puede apreciar cómo será la página al seleccionar un equipo. En la zona central se puede apreciar una tabla de las MVs en el equipo seleccionado con la siguiente información recolectada de la aplicación cliente:

- Hypervisor de la MV.
- Nombre de la MV.
- RAM de la MV.
- HDD (Espacio ocupado/Espacio total de la MV).
- Estado (si está encendida, apagada o suspendida).
- Ver (se podrá entrar en la MV seleccionada para más detalles).

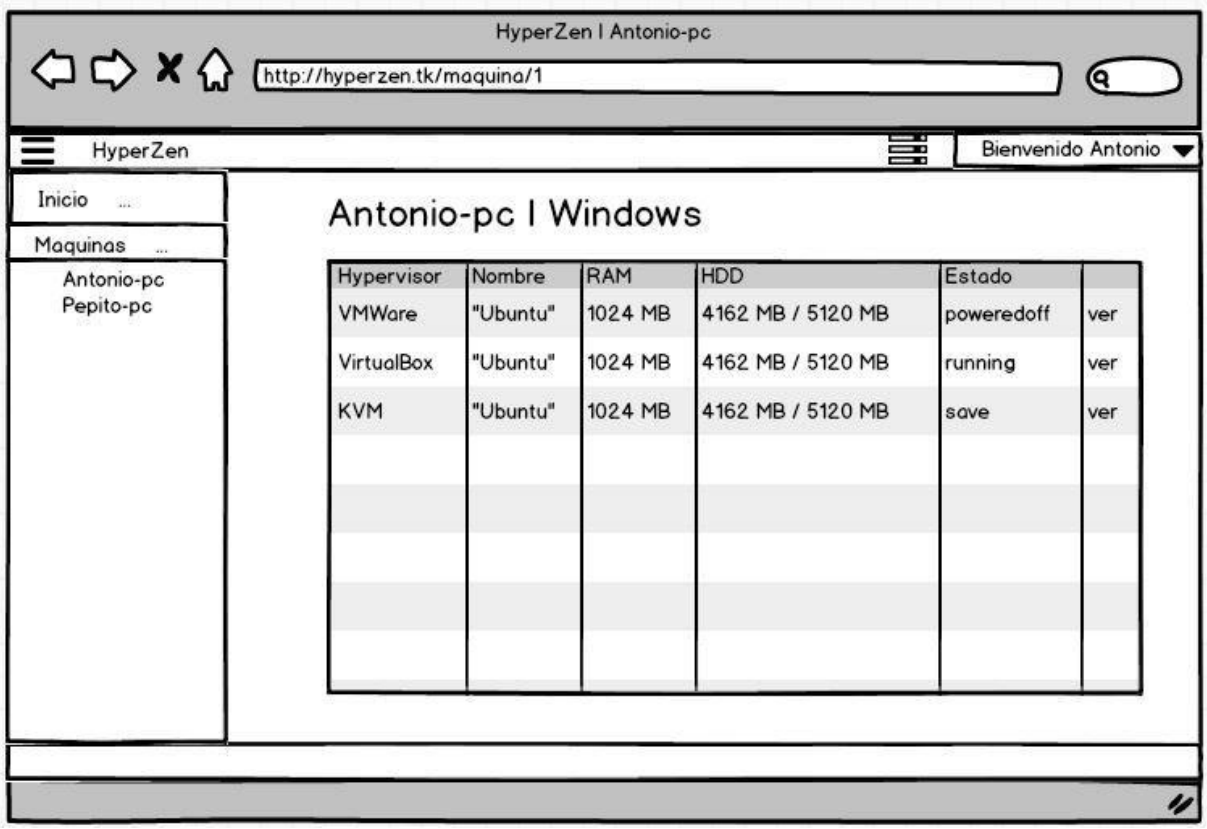

Fig.16: Esbozo de la pantalla al seleccionar un equipo.

En las Figuras 17 y 18 se puede apreciar el aspecto de la página al seleccionar una MV. En la zona central se puede apreciar:

- Título (compuesto de Hypervisor | Nombre).
- HDD: estará compuesto por un gráfico circular, que mostrará tres secciones para ver de manera visual el estado del disco duro.
    - Espacio ocupado del disco duro de la MV.
    - Espacio libre del disco duro de la MV.
    - Espacio libre del disco duro del equipo.
- Icono editar HDD: abrirá un cuadro de HTML flotante al seleccionarlo, dando la posibilidad de aumentar el tamaño del disco duro. El tamaño máximo será el espacio libre del disco duro del equipo.

- RAM: estará compuesto por un gráfico circular, que mostrará dos secciones para ver de manera visual el estado de la memoria.
    - Memoria asignada a la MV.
    - Memoria total – memoria asignada.
- Icono editar RAM: abrirá un cuadro de HTML flotante al seleccionarlo, y dará la posibilidad de cambiar el tamaño de la memoria.
- Estado: estará compuesto por un gráfico circular, que mostrará el estado de la MV de forma visual (Rojo → Apagada, Amarillo → Suspendida, Verde → Encendida).
- Icono opciones Estado: abrirá un cuadro de HTML flotante con opciones. El listado de opciones que aparece puede variar en función del estado en que se encuentre la MV. De tal forma, que aquellas opciones que no se pueden ejecutar, no aparecerán en el listado.
    - Apagar: apagará la MV si está encendida o suspendida.
    - Encender: encenderá la MV si está apagada o suspendida.
    - Reset: reiniciará la MV.
    - Conexión vnc: podrá conectarse a la MV configurando *ipvnc* y *portvnc* siempre y cuando tenga su máquina configurada para ello.
    - Hacer snapshot: se pedirá el nombre de la snapshot y creará una snapshot.
    - Restaurar snapshot: podrá seleccionar de un *select* la snapshot a restaurar y esta se restaurará.
    - Eliminar snapshot: podrá seleccionar de un *select* la snapshot a eliminar y esta será eliminada.

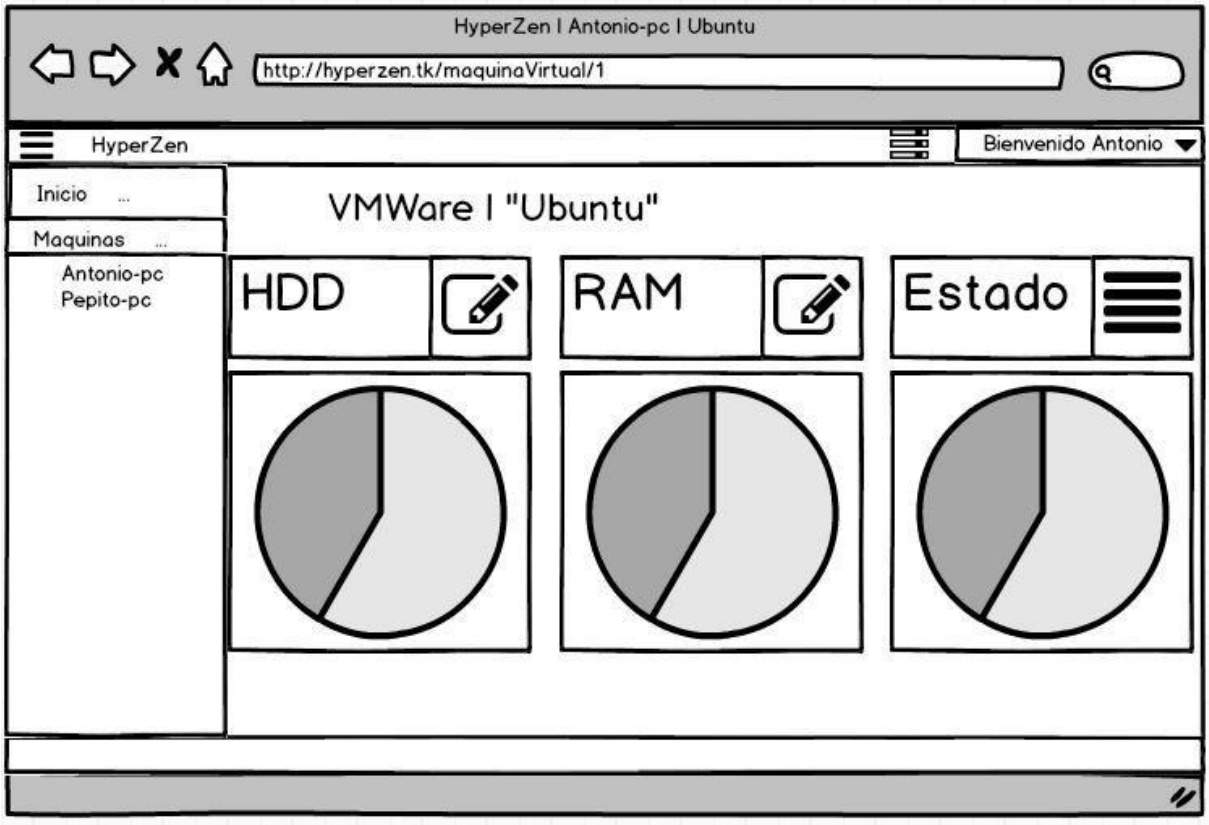

Fig.17: Esbozo de la pantalla al seleccionar una MV.

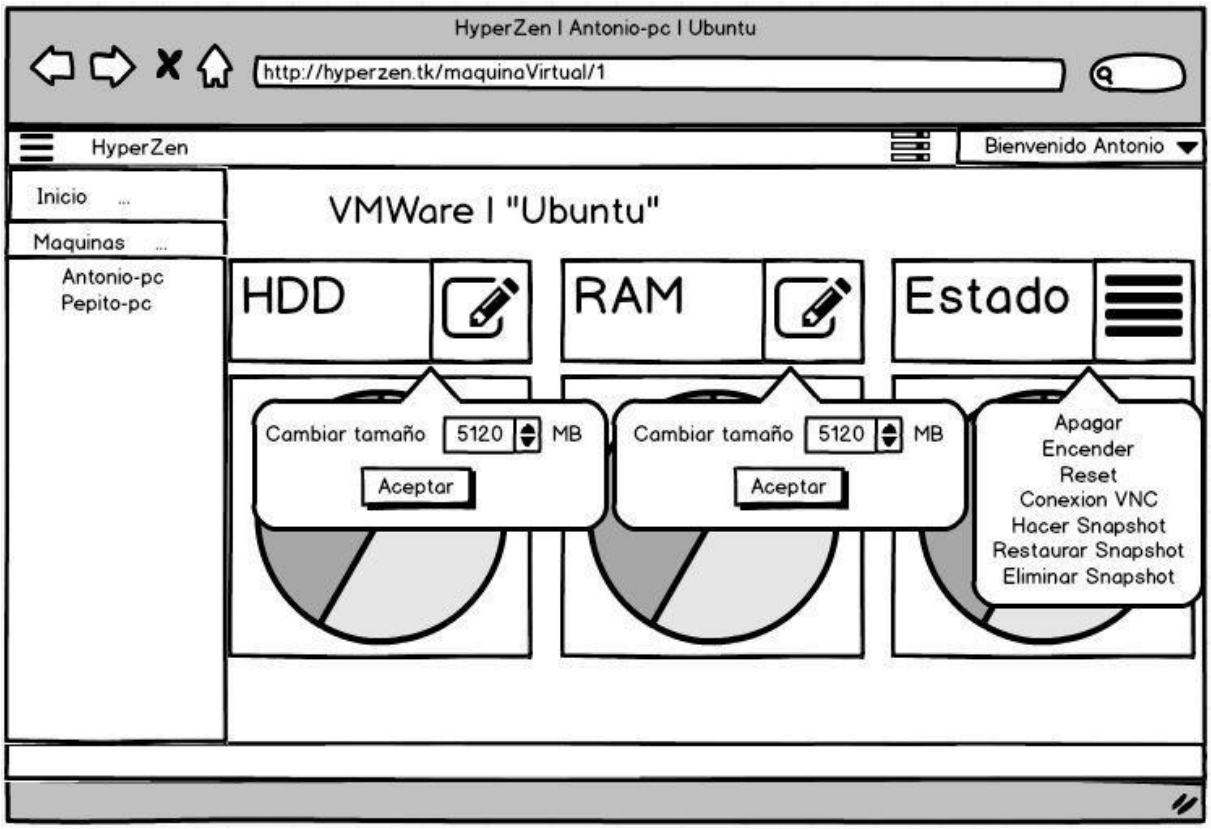

Fig.18: Esbozo de la pantalla que muestra las acciones que se pueden realizar.

Las opciones descritas anteriormente vienen por defecto en la aplicación pero se pueden crear nuevas opciones en función de con que parámetros se crea un comando parametrizado.

4.1.6. Desarrollo de la interfaz

Para no empezar la interfaz desde cero, se hará uso de plantillas HTML5 con Bootstrap de uso libre que faciliten el trabajo. La que se va a utilizar será "Novus Admin Panel".

Para desarrollar la interfaz, es necesario conocer la "interacción controlador-vista de Rails" explicado anteriormente y la "inyección de código Ruby" en consonancia con el diseño de tal forma que se puedan hacer páginas dinámicas.

4.1.6.1. Inyección de código Ruby

La metodología utilizada para inyectar código Ruby en el diseño web de páginas dinámicas se explica a continuación, a través de sencillos ejemplos sobre instrucciones de salto condicionales (if else), bucles, enlaces y formularios.

If else

Para poner un ejemplo se da por hecho que en el controlador se encuentra una variable "@usuarios" donde se almacena un Array con usuarios.

```
<div >
<%if @usuarios.count>0%>
<%else%>
<p >No hay Usuarios disponibles. </p>
<%end%>
</div>
```

Como se puede apreciar para inyectar código Ruby hay que ponerlo entre "<%%>", de esta forma se pueden hacer cambios antes de que el HTML se muestre.

Ahora como se puede observar con la sentencia **<%if @usuarios.count>0%>**", se comprueba el número de usuarios que hay en el Array, si este es mayor que 0 no se hace nada por el momento y si no se escribirá " No hay Usuarios disponibles. " en HTML, como se ve esta sentencia debe acabar con "**<%end%>**".

Resultado HTML5:

```
<div >
<p >No hay Usuarios disponibles. </p>
</div>
```

Bucles

Siguiendo con el ejemplo anterior, donde ahora "@usuarios=[{id=>1,nombre=>"Antonio"} ,{id=>2,nombre=>"Pepito"}]" se añadirá un bucle si se encuentra algún usuario:

```
<div >
<%if @usuarios.count>0%>
<select>
<%@usuarios.each do |user|%>
 <option value='<%=user.id%>'><%=user.nombre%></option>
 <%end%>
</select>
<%else%>
<p >No hay Usuarios disponibles. </p>
<%end%>
</div>
```

Ahora como se puede observar con la sentencia "**<%@usuarios.each do |user|%>**", se recorre el Array de usuarios y se va a utilizar la propiedad *"id"* y la propiedad *"nombre"* de usuario, para establecer el valor de la etiqueta *"option"* y como se visualizará en el "*select*". Como se puede observar, esta sentencia debe acabar con un "**<%end%>**".

Resultado HTML5:

```
<div >
<select>
 <option value='1'>Antonio</option>
        <option value='2'>Pepito</option>
</select>
</div>
```

Link_to

Link_to es una directiva de *RubyOnRails* la cual resuelve la ruta especificada en *routes.rb*, para tener acceso a dicha url. Siguiendo con el ejemplo anterior, si se selecciona un usuario se redirecciona a la url del usuario.

```
<div >
<%if @usuarios.count>0%>
        <%@usuarios.each do |user|%>
<%=link_to user_path(user.id), user.nombre do%><i class="fa fa-user"></i><%end%>
 <%end%>
<%else%>
<p >No hay Usuarios disponibles. </p>
<%end%>
</div>
```

Resultado HTML5:

```
<div >
<a  href="/user/1"><i class="fa fa-user">”Antonio”</i></a>
<a  href="/user/2"><i class="fa fa-user">”Pepito”</i></a>
</div>
```

Como se puede observar *link_to* se transforma en código HTML5 para que el navegador pueda interpretarlo sin problema. En este caso convirtiéndose en la etiqueta “<a>”, pasándole como parámetros la dirección url (*href*) y como debe visualizarse en el navegador.

Form

Form es una directiva de *RubyOnRails* muy útil pues puede generar fácilmente un formulario en una vista que sirva tanto para crear un objeto como para actualizarlo. Para ver cómo funciona se usarán dos ejemplos, uno en el que el controlador posea la variable *“@usuario=Usuario.new”* (se crea un nuevo usuario que no se encuentra en la base de datos) y otro en el que posea la variable *“@usuario={id=>1,nombre=>”Antonio”}”*.

```
<%= form_for(@usuario) do |f| %>
  <div class="field">
    <%= f.label :nombre %><br>
    <%= f.text_field :nombre %>
  </div>
  <div class="actions">
    <%= f.submit %>
  </div>
<% end %>
```

Resultado HTML5 *"@usuario=Usuario.new"*:

```
< form class="new_usuario" id="new_ usuario" action="/usuarios" accept-charset="UTF-8"
method="post">
<div class="field">
   <label for=" usuario_nombre">Nombre</label><br>
   <input type="text" name=" usuario[nombre]" id=" usuario_nombre">
 </div>
<div class="actions">
   <input type="submit" name="commit" value="Create Usuario">
 </div>
</form>
```

Resultado HTML5 *"@usuario={id=>1,nombre=>"Antonio"}"*:

```
< form class="edit_usuario" id="edit_usuario_1" action="/usuarios/1" accept-charset="UTF-8"
method="post">
<div class="field">
   <label for="usuario_nombre">Nombre</label><br>
   <input type="text" name="usuario[nombre]" id="usuario_nombre" value="Antonio">
 </div>
<div class="actions">
   <input type="submit" name="commit" value="Create Usuario">
 </div>
</form>
```

Como se puede apreciar en función de si el usuario ya existe o no, generará un formulario para crear un nuevo usuario, o generará un formulario para modificar un usuario existente en la base de datos.

Esta explicación de la metodología para inyectar código Ruby ha servido de base para diseñar la interfaz de la aplicación web.

4.1.7. Resultado

En las siguientes figuras, se puede ver el resultado final de la interfaz de la aplicación web, diseñada en el presente TFG, y disponible en la dirección web http://www.hyperzen.tk/. La Figura 19 muestra el aspecto de la página inicial donde se muestra una breve guía de ayuda para su correcta utilización. Mientras, que la Figura 20 muestra el perfil de un usuario registrado previamente.

Si se selecciona alguno de los equipos del usuario, aparecerá la página de la Figura 21, donde se muestra el listado de MVs disponibles en ese equipo. Mientras que la Figura 22 muestra toda la información relativa a la MV seleccionada, y la Figura 23 muestra las acciones que puede realizar el usuario en la MV seleccionada.

La Figura 24 muestra en una tabla la información relativa a las acciones definidas en el hypervisor centralizado. En cada fila de la tabla se muestra cada uno de los comandos necesarios para realizar la acción correspondiente con la MV del hypervisor. La Información más destacada es el tipo de sistema operativo, hypervisor de la MV y el nombre de la acción a realizar. A la izquierda del nombre de la tabla hay un icono (+) que al pulsarlo habilitará un

formulario para crear un nuevo comando, al lado de cada fila se ha añadido la opción de editar y eliminar el comando.

La Figura 25 muestra un formulario donde crear o modificar un comando, mientras que la Figura 26 muestra un ejemplo para editar un comando que permite modificar el tamaño del disco de una MV con sistema operativo Windows en VirtualBox. Los parámetros que se deben rellenar para crear/editar un nuevo comando son:

- Nombre del comando
- Sistema operativo
- Hypervisor
- Comando parametrizado

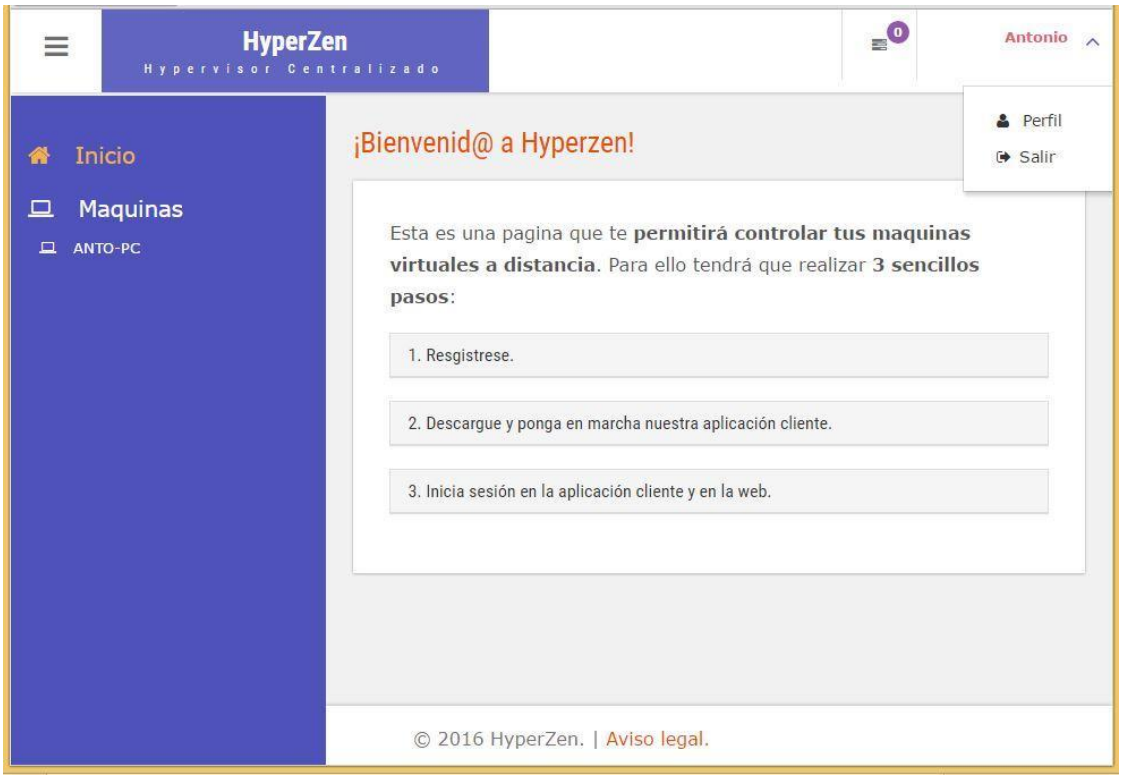

Fig.19: Captura de pantalla de la página principal.

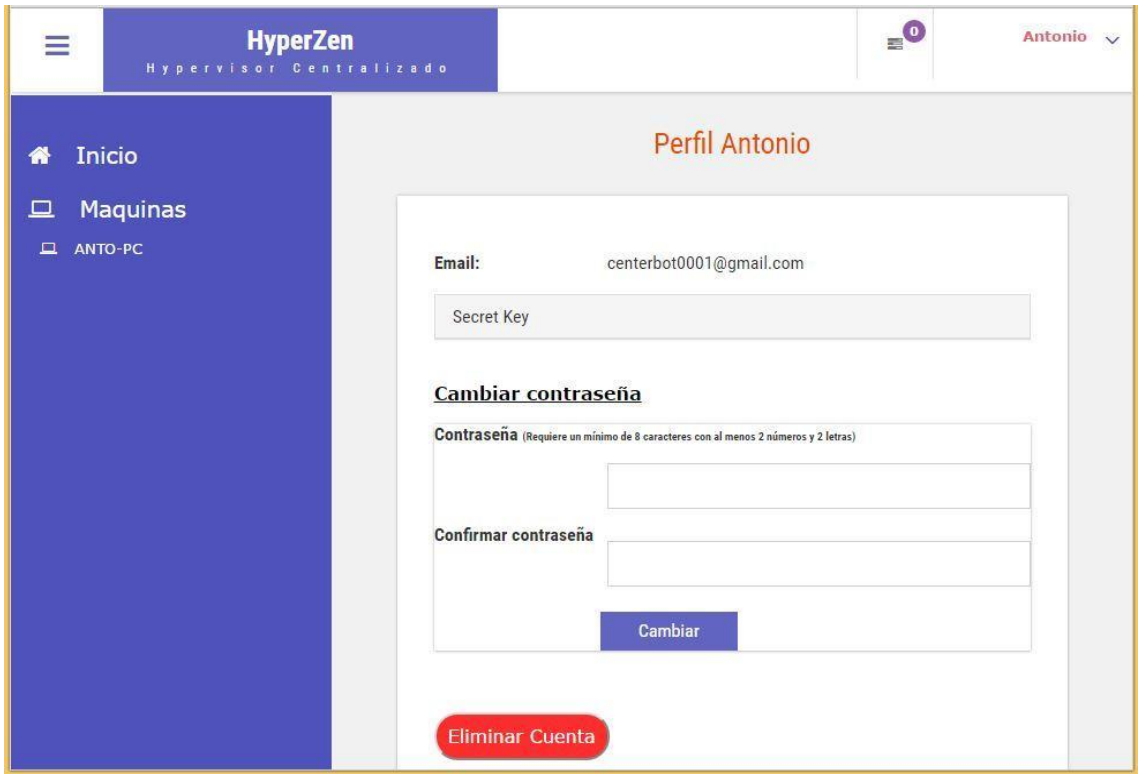

Fig.20: Captura de pantalla de la página perfil.

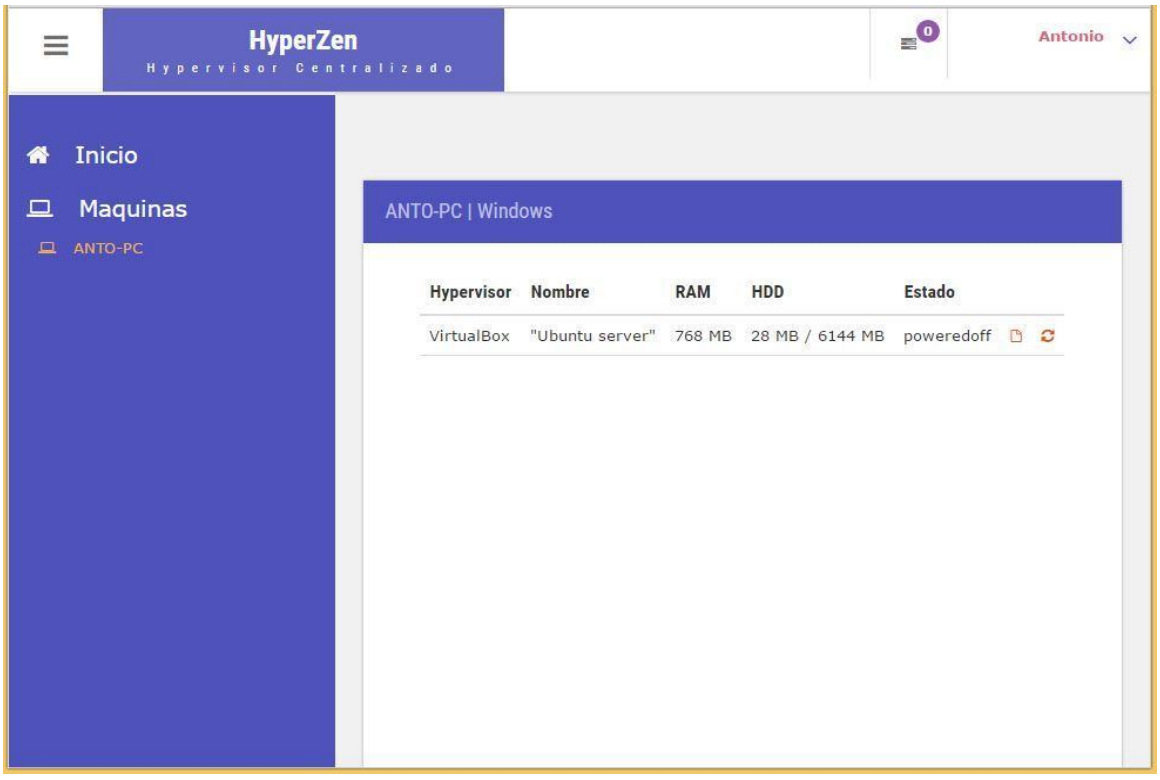

Fig.21: Captura de pantalla de la página al seleccionar un equipo.

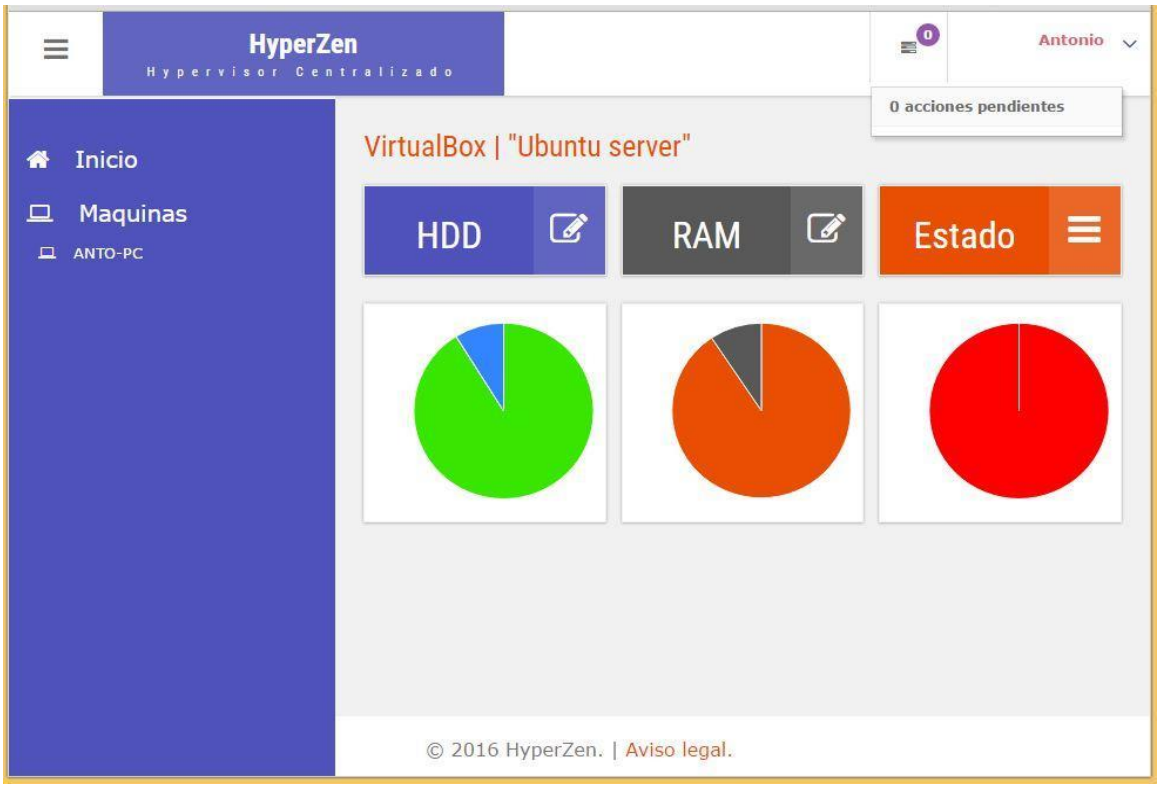

Fig.22: Captura de pantalla de la página al seleccionar una MV.

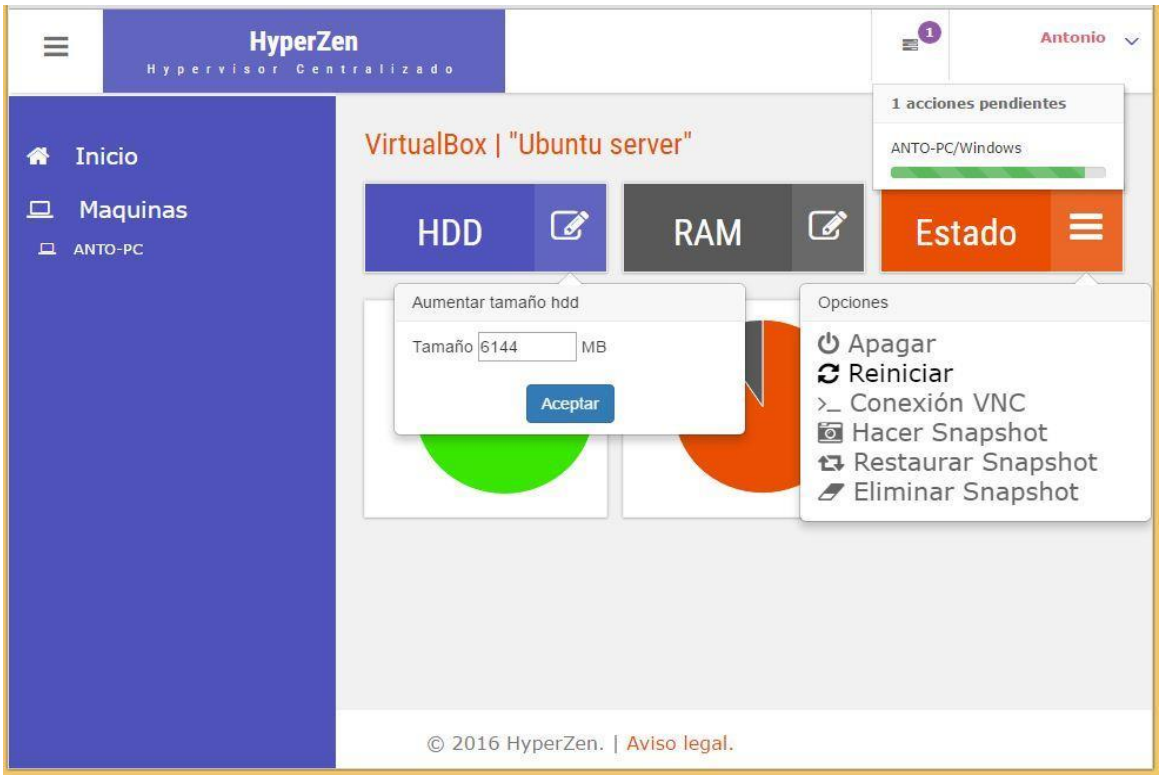

Fig.23: Captura de pantalla de la página mostrando las acciones que puede realizar.

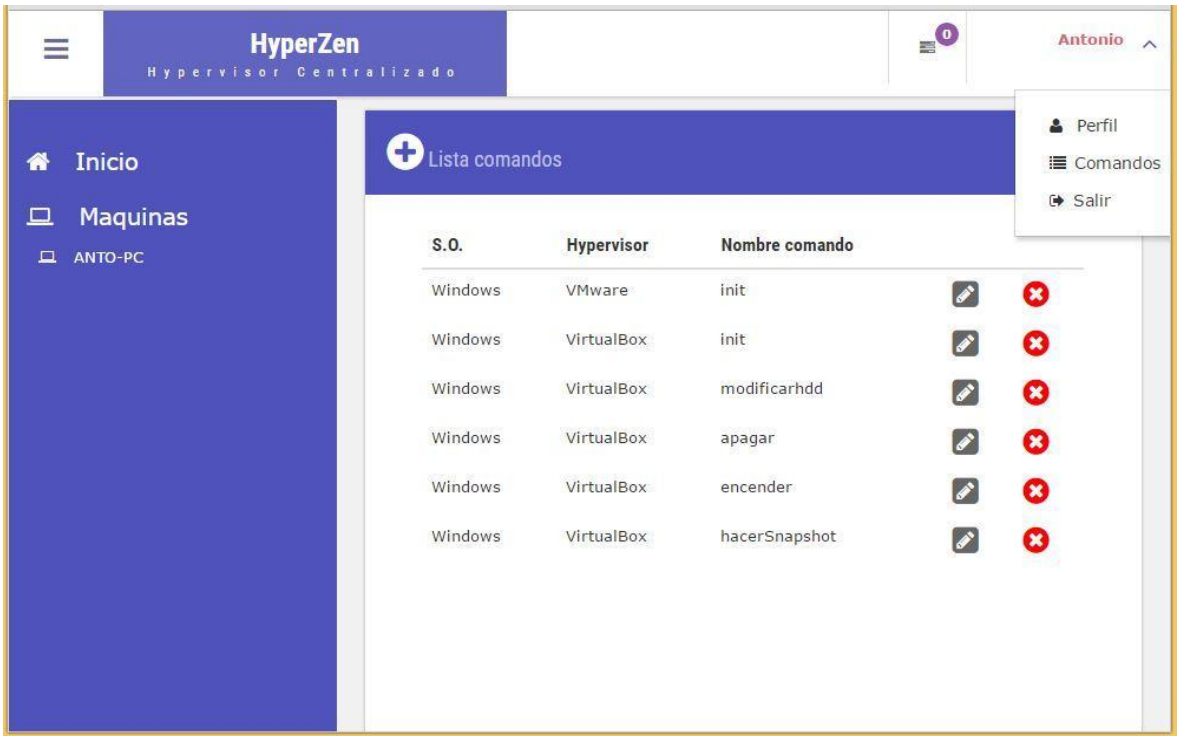

Fig.24: Captura de pantalla de la página lista de comandos.

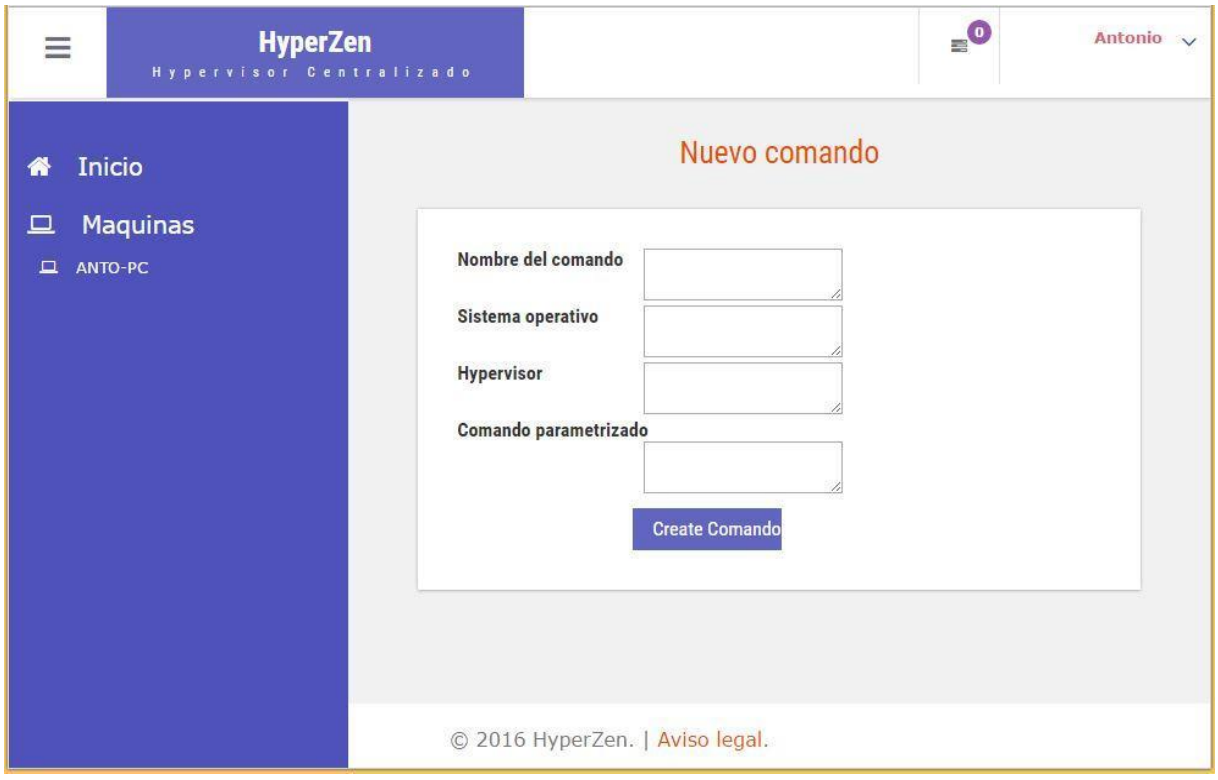

Fig.25: Captura de pantalla de la página nuevo comando.

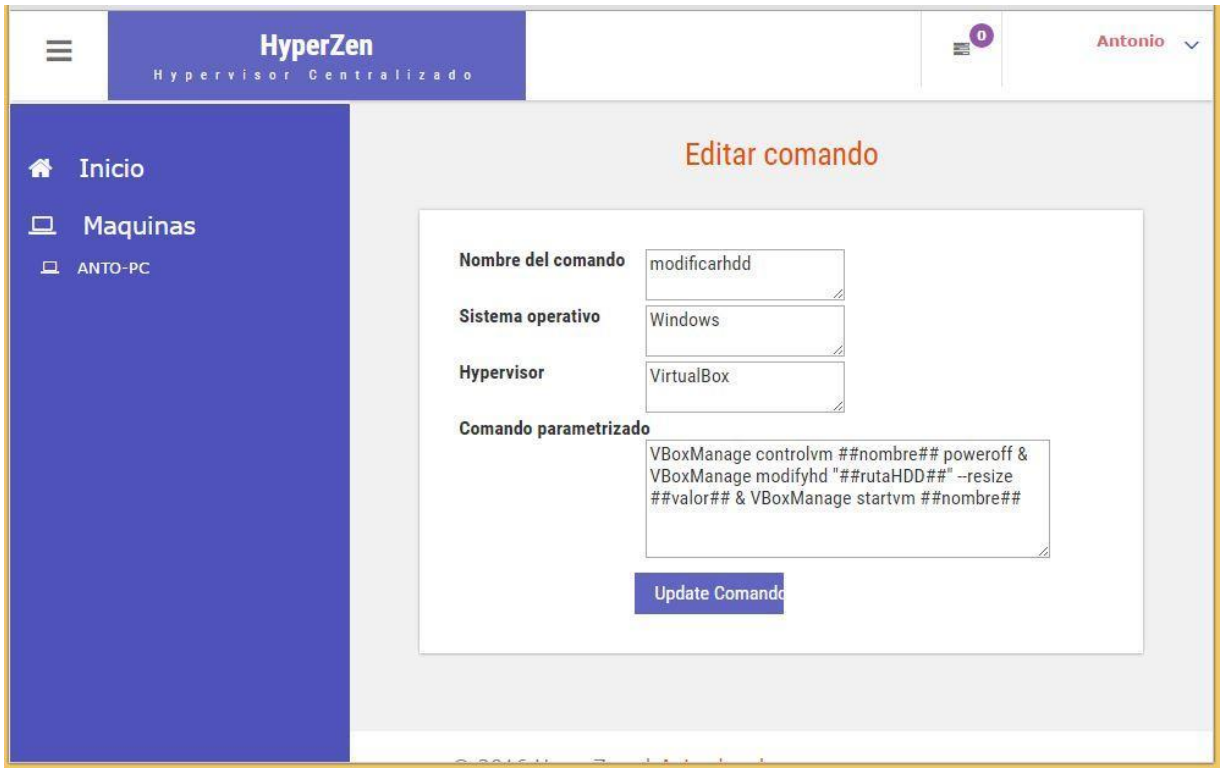

Fig.26: Captura de pantalla de la página editar comando.

4.1.8. Pruebas unitarias

Las pruebas del sistema son muy importantes si se planea añadir en un futuro nuevas funcionalidades o rehacer parte de él. Gracias a las pruebas unitarias, se evita probar manualmente la aplicación para detectar un correcto funcionamiento de la aplicación, con el consiguiente ahorro de tiempo.

Que todas las pruebas unitarias se ejecuten satisfactoriamente no quiere decir que el sistema funcione como se espera. Las pruebas también han de ser creadas, y es posible que no ejecuten todo el código, o no prueben todos los casos posibles, además de que también puede haber algún error en la codificación de las pruebas.

Para solucionar el problema de ejecutar todo el código del sistema se suele hacer uso de pruebas de cobertura, que muestran los fragmentos de código que se ejecutan y los que no. Una cobertura del 100% indicaría que las pruebas unitarias prueban todas las líneas del código fuente. Si bien no es necesario llegar a tal cifra, es conveniente maximizarla todo lo posible.

Aprovechando el disponer de pruebas unitarias, es conveniente hacer uso de herramientas de integración continua que hagan las tareas repetitivas como pueden ser el obtener la última versión de la aplicación, construirla, configurarla y posteriormente ejecutar las pruebas. De esta forma se consigue dedicar más tiempo a la implementación y arreglo de

errores que a probar el correcto funcionamiento, aunque se debe invertir al principio un tiempo en preparar y configurar la herramienta de integración continua.

Para garantizar que los usuarios, equipos, MVs, comandos y órdenes se están creando, actualizando o eliminando correctamente, se ha creado una batería de tests unitarios, además de conseguir así una disminución del mantenimiento al lanzarlos automáticamente en Jenkins.

4.1.8.1. Pruebas con Rake

Se debe saber que todas las aplicaciones que se desarrollan en *Rails* tienen por defecto tres entornos: Entorno de desarrollo, entorno de producción y entorno para pruebas. Esto significa que cada uno de ellos está completamente aislado del resto, lo que da la confianza de realizar todas las pruebas que se deseen sin alterar el entorno de producción o de desarrollo.

La estructura de las pruebas está alojada en una carpeta definida por *Rails* llamada test y básicamente la estructura es la siguiente:

- *controllers/*
- *helpers/*
- *test_helper.rb*
- *fixtures/*
- *integration/*
- *models/*

Una estructura bastante sencilla: en la carpeta *models* se diseñan las pruebas de los modelos de la aplicación; en *controllers* las pruebas de los controladores; *fixtures* es una forma de organizar nuestros datos de prueba y por último *test_helper.rb* es un archivo que contiene la configuración por defecto para sus análisis.

Se generan datos que se usarán para las pruebas de forma aislada.

```
# /fixtures/usuarios.yml
<% 100.times do |n| %>
user_<%= n %>:
  username: <%= "user#{n}" %>
  password: <%= "userpassword#{n}" %>
  email: <%= "user#{n}@example.com" %>
  auth: <%= "user#{n}"%>
  resetPassword: 0
<% end %>
```

Un ejemplo de prueba unitaria sencilla sería testear el comportamiento de la aplicación cuando un usuario inicia una sesión. A continuación, se muestra el código de la prueba unitaria:

```
test "iniciar sesion" do
 user= usuarios(:user_1)
 log_in(user)
 current_user
 assert_equal user, @current_user
```

```
log_out
assert_equal nil, @current_user
  end
```

Donde:

- “iniciar sesion” se utiliza como nombre del test que se va a ejecutar.
- “user= usuarios**(:user_1)**” llama a un usuario que hemos creado en *fixtures*.
- “log_in**(**user**)**” inicia la sesión del usuario que se le da como parámetro.
- “current_user” genera una variable *@current_user* con la información del usuario identificado.
- “assert_equal” se utiliza para las aserciones. Se le pasa como primer parámetro el valor esperado y como segundo parámetro la variable que debe comprobar. Como resultado se comprueba si pasa o falla parte del test.
- “log_out” cierra la sesión del usuario, y pone a *nil* la variable *@current_user*.

4.1.8.2. Ejecución de las pruebas

Ejecutar pruebas en *Rails* es bastante sencillo sólo se deben ejecutar unas líneas de comando y ver qué resultados arroja. Primero hay que asegurarse de que la base de datos de prueba esté creada con éxito, si no se deben ejecutar los siguientes comandos para generarla:

bundle exec rake db:migrate

bundle exec rake db:test:load

Una vez comprobado y preparado el ambiente de pruebas sólo queda saber cómo ejecutar pruebas. Para ello sólo hay que ejecutar la siguiente línea de comando para llamar a todas las pruebas del sistema.

bundle exec rake test

La Figura 27 muestra la salida obtenida tras ejecutar 14 test con 24 aserciones y verificando que se han pasado todos los test:

```
# Running:

..............

Finished in 0.804254s, 17.4074 runs/s, 29.8413 assertions/s.

14 runs, 24 assertions, 0 failures, 0 errors, 0 skips
```

Fig.27: Resultado pruebas unitarias.

4.1.8.3. Cobertura de las pruebas

En esta aplicación lo más importante es garantizar la integridad de los datos, porque hay muchos cambios en la base de datos debido a la sincronización continua de los equipos y MVs de un usuario, y si no se garantiza la integridad de estos datos, por ejemplo, podríamos encontrar problemas al ejecutar acciones para una MV que ya no existe.

Por tanto, es necesario tener una cobertura amplia de los controladores *"máquina_virtual", "máquina", "usuario", "comando"* y *"orden"* en los que se contemplen todos los casos posibles.

En la Figura 28 se muestran las coberturas de todos los controladores generadas con la gema "simplecov".

	% covered	Lines	Relevant Lines	Lines covered
/usuarios_controller.rb	84.85 %	117	66	56
/maquina_virtuals_controller.rb	87.8 %	181	123	108
/comandos_controller.rb	91.67 %	98	48	44
/maquinas_controller.rb	96.88 %	55	32	31
/ordens_controller.rb	97.78 %	73	45	44
/application_controller.rb	100.0 %	9	5	5
/pages_controller.rb	100.0 %	49	27	27

Fig.28: Cobertura de los servicios. Generada con la gema "simplecov".

Como se puede observar en la Figura 28, hay un alto porcentaje de líneas cubiertas, la mayoría de ellos no llegan al 100%, pero se acercan mucho y eso es debido a que hay partes de métodos bastantes complejos que no se llegan a ejecutar.

4.2. Aplicación cliente

4.2.1. Requisitos del sistema

La aplicación cliente, desarrollada en Ruby [10], será la encargada de monitorizar el estado de las MVs del equipo y realizar peticiones sobre la aplicación web.

La aplicación cliente será desarrollada tanto con interfaz de usuario (GUI) como sin ella (uso de la consola de comandos) para permitir que cualquier sistema operativo pueda usar la aplicación, tanto en la aplicación con GUI como sin ella se pedirá la *secret key* del usuario previamente registrado en la aplicación web.

Haciendo una petición a la aplicación web con la *secret key*, se determinará si el usuario está registrado en la base de datos. Si el usuario existe, se procederá a identificar el nombre del equipo (hostname) y el sistema operativo, realizando una petición a la aplicación web con la *secret key*, el *hostname* y el *sistema operativo*, determinándose si el equipo está vinculado con la *secret key*. En caso afirmativo, se procede a la búsqueda de las MVs en el equipo. Sin embargo, si se determina que el equipo no está vinculado, se envía una petición para vincular el equipo y posteriormente se buscan las MVs.

Para la identificación de las MVs primero se obtiene una lista de las MVs que se encuentran vinculadas al equipo correspondiente y se añade un flag con un “0”, este flag se utilizará para eliminar de la aplicación web aquellas MVs que ya no se encuentran en el equipo.

Posteriormente, se procede a la identificación de las MVs en el equipo. Para ello, primero se realiza una llamada a la aplicación web donde se recogerán aquellos comandos compatibles con el sistema operativo anfitrión (un comando por cada hypervisor a identificar, por ejemplo, para VirtualBox se utilizará el script de búsqueda de MVs descrito en el apartado 2.3. Hypervisores), que servirán para obtener la siguiente información de cada MV:

- “Hypervisor”
- ”Nombre”
- ”Estado”
- “Versión del hypervisor (si influyera)”
- “Estado (apagada, encendida, suspendida)”
- “RAM usada por la MV”
- “RAM total del sistema anfitrión”
- “Ruta de la MV”
- “Ruta del HDD principal de la MV”
- “capacidad total del HDD de la MV”
- ”Espacio ocupado del HDD de la MV”
- “Espacio libre del sistema anfitrión”
- “Snapshots de la MV”

Una vez identificada la MV, se comprueba si existe en la lista de MVs, en caso afirmativo se cambia el flag de “0” a “1”. Posteriormente, se realiza una petición a la aplicación web con los datos de la MV y si esta existe actualiza los datos, sino se crea una nueva MV con todos los datos. Este procedimiento se repite hasta identificar todas las MVs. Se procede a recorrer la lista de las MVs y aquellas que tengan flag “0” serán eliminadas de la aplicación web.

Finalmente, una vez identificadas todas las MVs, la aplicación cliente realiza llamadas a la aplicación web cada 5 segundos con el fin de no saturar el servidor. Este chequeo periódico permite a la aplicación cliente detectar alguna acción pendiente a ejecutar sobre las MVs (apagar, encender, reiniciar, cambiar la capacidad de la RAM, aumentar la capacidad del HDD, hacer, restaurar o eliminar una snapshot). Si se detecta una acción pendiente, se detienen las llamadas periódicas al servidor, volviéndose a identificar el estado de las MVs (tardará entre 20-90 segundos) y habilitándose de nuevo las llamadas al servidor.

4.2.2. Diseño de la aplicación

Se ha creado un diagrama de clases para la aplicación cliente que se puede observar en la Figura 29. La clase *"AppCliente"* es una clase de Ruby. En ella, se implementan los métodos necesarios para la comprobación del usuario, monitorización y gestión de sus MVs.

Para poner en marcha esta aplicación es necesario que el usuario escriba su *secretKey* (proporcionada automáticamente cuando se registró en la aplicación web), bien cuando se la pida por consola, o si tiene GUI deberá escribirla en el recuadro y pulsar en el botón.

Una vez escrita la *secretKey* se llama al método *'app'*, el cual establecerá el nombre de la url base en la variable *"baseURL"*, escribirá el valor de la *secretKey* en la variable *"secretKey"* y si está utilizando GUI ocultará el botón para que el usuario no pueda volver a darle de nuevo.

Lo siguiente es verificar si la *secretKey* pertenece a algún usuario, para ello se hace una petición a la aplicación web (concretamente a *"baseURL"+key/+"secretKey"*), y se verifica si el usuario existe o no, en caso de existir se llamará al método *'updateInfo'*.

El método *'updateInfo'* hace una llamada al método *'recuperarOS'* que modifica la variable *"máquina"* con el sistema operativo anfitrión y el *hostname* del sistema anfitrión. Se procede entonces a comprobar si está el equipo vinculado al usuario haciendo una petición a la aplicación web (concretamente a *"baseURL"+existeMáquina/+"secretKey"+'/'+"máquina"*). Si no se encuentra vinculada entonces se hace una petición a la aplicación web para crear y vincular dicho equipo al usuario, en caso contrario hacemos una llamada al método *'initInfo'* y posteriormente al método *'comandos'*.

El método *'initInfo'* (tardará entre 20-90 segundos) llamará al método *'getlistmvs'*, el cual haciendo una petición a la aplicación web obtendrá un Array con las MVs vinculadas al equipo, dicho Array se almacenará en la variable *"listaMVS"*. Después, se hace una llamada a la aplicación web para obtener que comandos deben ejecutarse en el sistema para identificar las posibles MVs, por cada comando (cada comando equivale a obtener todas las MVs de un hypervisor concreto) se llama al método *'initvms(hypervisor, comando)'*.

El método *'initvms(hypervisor,comando)'*, ejecutará un comando para obtener todas las MVs de un hypervisor concreto, obteniéndose la siguiente información de cada una de las MVs:

- "Nombre"
- "Estado"
- "Versión del hypervisor(si influyera)"
- "Estado (apagada, encendida, suspendida)"
- "RAM usada por la MV"
- "RAM total del sistema anfitrión"
- "Ruta de la MV"
- "Ruta del HDD principal de la MV"
- "capacidad total del HDD de la MV"
- "Espacio ocupado del HDD de la MV"
- "Espacio libre del sistema anfitrión"
- "Snapshots de la MV"

Si alguna MV se encuentra en el Array *"listaMVS"*, se pondrá un flag que inicialmente se encuentra a "0" en "1", después los datos numéricos se transforman a MB con el método *'toMB*(cadena)' y toda esta información se almacenará como una cadena en una variable local. Esta variable local debe formatearse con el método '*format*(cadena)' para poder pasar la información por una url. Finalmente se hace una petición a la aplicación cliente con la información de cada MV (concretamente a "baseURL" +mv/+ "secretKey" +'/'+ "máquina" +'/'+ "informacionMV"), donde se actualizará la MV si está vinculada al equipo o se creará si no lo está y se llama al método *'delmvs'* donde se eliminarán las MVs que no tengan el flag a "1", dejando actualizada la base de datos de la aplicación web.

El método *'comandos'*, comprobará cada 5 segundos (para no saturar el servidor buscando acciones a realizar y es un tiempo de respuesta para ejecutar las acciones aceptable) si hay que realizar alguna acción sobre alguna MV, se ejecutará dicha acción y si es ejecutada con éxito se llamará al método *'initInfo'* para volver a comprobar el estado de las MVs y actualizarlo.

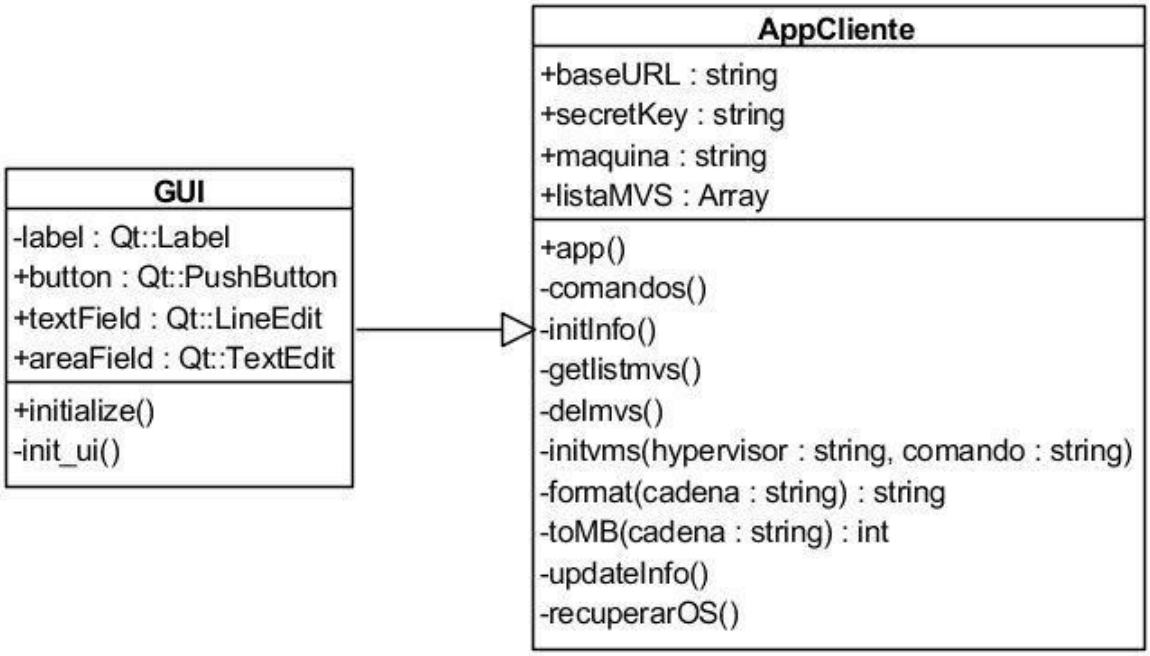

Fig.29: Diagrama de clases de la aplicación cliente.

4.2.3. Diseño de la interfaz

Si bien la aplicación cliente puede o no tener interfaz gráfica de usuario (GUI), se ha realizado un esbozo de cómo sería.

4.2.3.1. Esbozo

La Figura 30 muestra la apariencia que debería tener la aplicación cliente, y como se puede apreciar en la parte superior se encuentra un campo de texto donde se insertará la *secret Key* del usuario, y para poner la aplicación en marcha habrá que pulsar en el botón "Enviar".

Una vez la aplicación se ponga a funcionar irá notificando las tareas que va realizando en el área de texto que se encuentra en el centro.

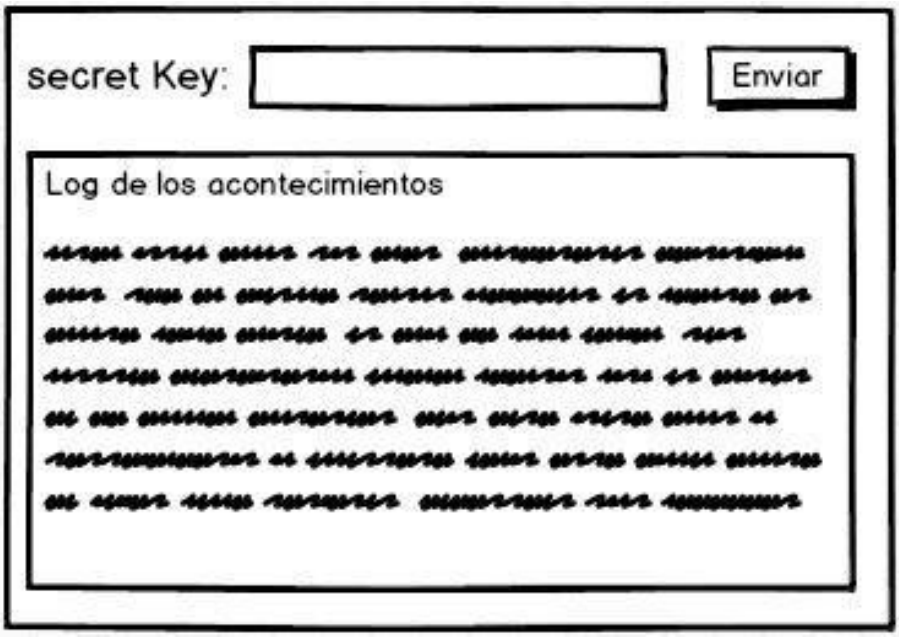

Fig.30: Esbozo de la aplicación cliente.

4.2.4. Desarrollo de la interfaz

Para el desarrollo de la interfaz se utilizará una gema de ruby llamada *Qtbindings*, se puede descargar de github o bien se puede utilizar el comando *"gem install Qtbindings"* y se instalará.

Si se analiza el código de la Figura 31, se aprecia que para usar la gema *Qtbindings* tan solo se necesita escribir *" require 'Qt4' "*, si se quieren utilizar los métodos y objetos de *Qtbindings* se tiene que hacer una herencia de *Qt::Widget*.

```
require 'Qt4'

class AppCliente < Qt::Widget
slots 'app()'

  def initialize
      super
      setWindowTitle "HyperZen Client"
      init_ui
      resize 500, 500
      show
  end

  def init_ui
    letra=14
    gridLayout = Qt::GridLayout.new(self)
    text="Escriba su clave secreta:"
    label = Qt::Label.new text, self
    label.setFont(Qt::Font.new('Times', letra, Qt::Font::Bold))
    @button = Qt::PushButton.new('Enviar')
    @button.setFont(Qt::Font.new('Times', letra, Qt::Font::Bold))
    @textField = Qt::LineEdit.new self
    @textField.setFont(Qt::Font.new('Times', 12, Qt::Font::Bold))
    @areaField = Qt::TextEdit.new "=========================================
    \n==============Registro de acontecimientos==============
    \n================================================\n", self
    @areaField.setFont(Qt::Font.new('Times', 12, Qt::Font::Bold))
    connect @button, SIGNAL("clicked()"),
            self, SLOT("app()")

    gridLayout.addWidget(label, 0, 0)
    gridLayout.addWidget(@textField, 0, 1)
    gridLayout.addWidget(@button, 0, 3)
    gridLayout.addWidget(@areaField, 1, 0,1,4)
   end
```

Fig.31: Código de la interfaz.

Los slots se utilizan en *Qtbindings* para tener visibilidad del método que se quiere llamar a través de una señal (SIGNAL), en este caso se hace visible para *Qtbindings* el método *'app()'*.

El método *'initialize'* es el encargado de generar una ventana con los objetos definidos en *'init_ui'*, en este caso con título "Hyperzen Client" y una resolución (500,500), para luego mostrarlos con el método *'show'* de la herencia de *Qt::Widget*.

En el método *'init*_ui' se puede observar lo siguiente:

- Para crear un layout, donde se van a "pintar" los objetos, se crea con *Qt:GridLayout.new*, esto creará un layout con forma de matriz.
- Para crear un label se escribe, *Qt:Label.new*.
- Para crear un botón se escribe, *Qt:PushButton.new*.
- Para crear un campo de texto se escribe, *Qt:LineEdit.new*.
- Para crear un área de texto se escribe, *Qt:TextEdit.new*.
- Para crear un evento al pulsar sobre el botón "Enviar", *Qtbindings* necesita de *connect* (requiere del objeto que debe escuchar, el evento que debe escuchar y la respuesta que debe ofrecer), como objeto de escucha se le pasa como parámetro el botón *@button*, se establece el evento que debe escuchar *('clicked()')*, ahora se requiere del objeto que dará una respuesta (*self*, pues es donde se encuentra el método *app*) y la respuesta que debe dar *('app()')*.
- Finalmente se añaden todos los objetos al layout y se le asignan los espacios de layout que va a ocupar cada uno, de esta forma si se hace la ventana más grande o pequeña se ajustarán los objetos a la ventana, la matriz que genera el layout vendrá definida por los objetos y posiciones que se inserten (así si se insertan dos objetos (0,0 y 1,1) genera una matriz de 2x2).

4.2.5. Resultado

Se puede ver el resultado final de la interfaz gráfica de la aplicación cliente en la Figura 32.

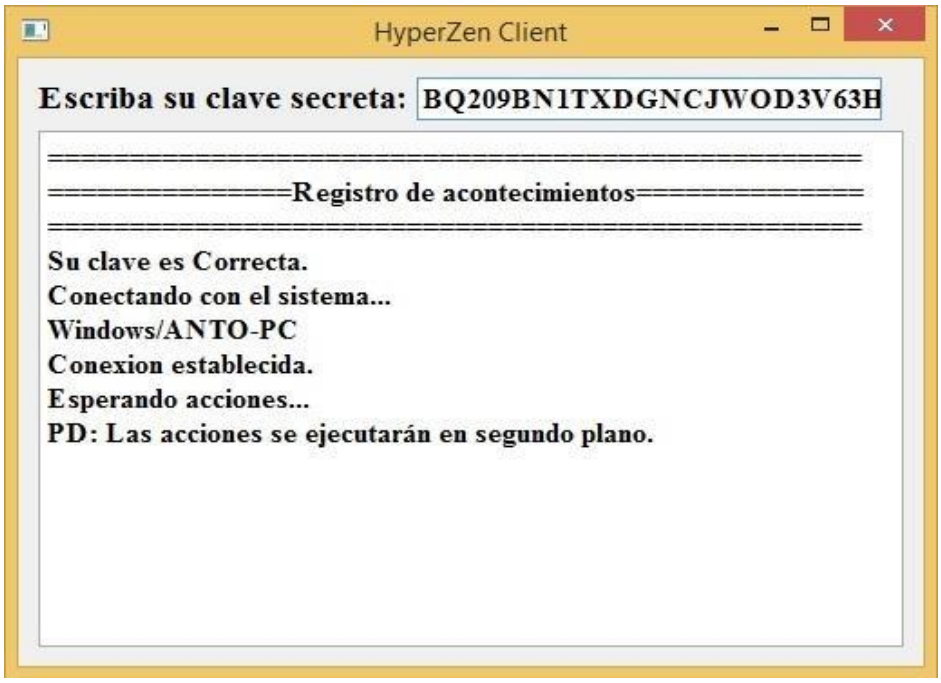

Fig.32: Interfaz gráfica de aplicación cliente.

5. Instalación y configuración

A continuación, se explica cómo hacer uso de la aplicación disponible en la dirección web http://hyperzen.tk/. Si se desea un despliegue automático se utilizará Jenkins y si no también se explicará cómo desplegarlo en cualquier entorno.

5.1. Despliegue con Jenkins

Requisitos:

- Tener instalado Jenkins [7]
- Tener instalado Nginx [8] o Apache para usarlos como servidor web.

Preparación del entorno:

- Instalar plugin “HTML Publisher plugin” en Jenkins
- Instalar plugin “rbenv plugin” en Jenkins

Una vez instalados los plugins hay que crear dos tareas en Jenkins, una para automatizar los test, como se muestra en las figuras 33,34 y 35, y otra para el despliegue de la aplicación web, como se muestra en las figuras 36 y 37.

Primera tarea: TestingWebHypervisorCentralizado

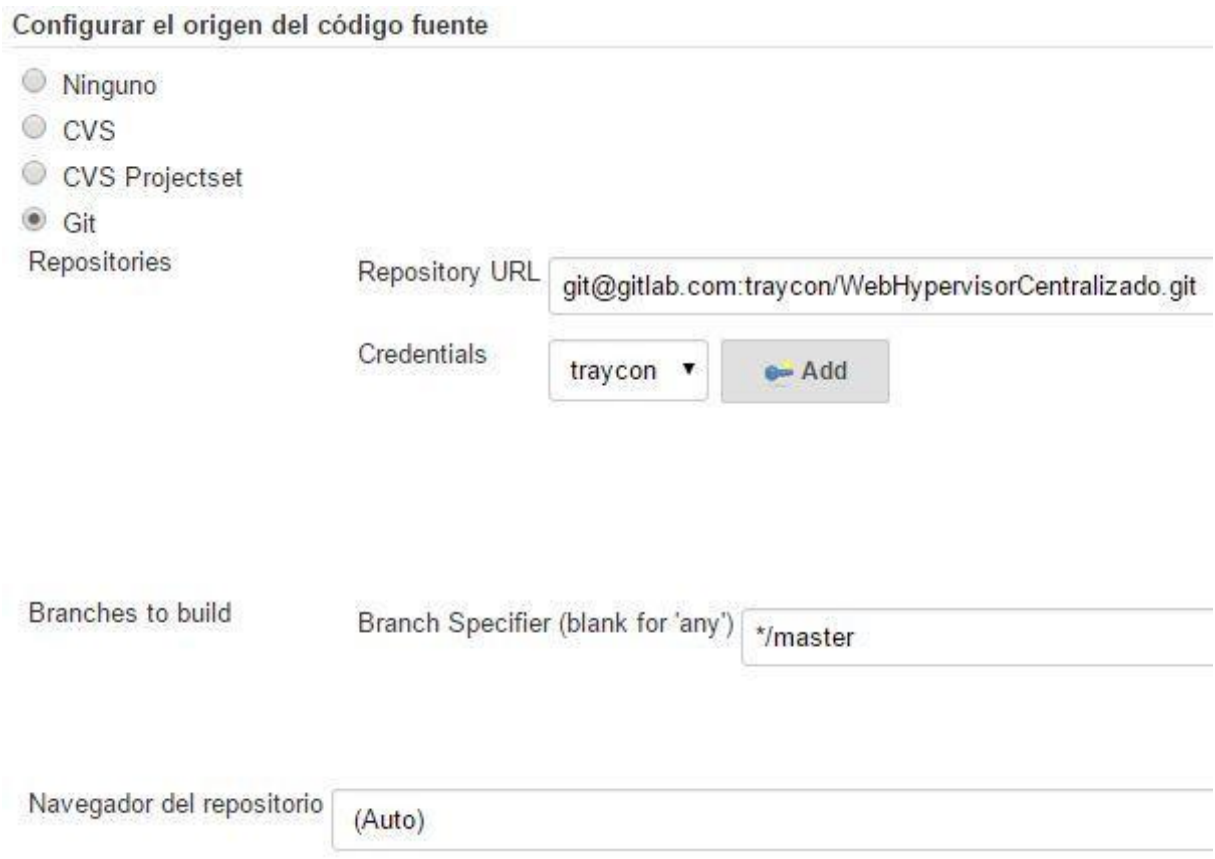

Fig.33: Configuración Jenkins tarea TestingWebHypervisorCentralizado – parte 1.

Fig.34: Configuración Jenkins tarea TestingWebHypervisorCentralizado – parte 2.

Fig.35: Configuración Jenkins tarea TestingWebHypervisorCentralizado – parte 3.

Como se observa en la Figura 34, el entorno de ejecución lo va a proporcionar el plugin *rbenv*, el cual se encarga de instalar la versión de Ruby que interese y de instalar las gemas que se necesiten. Ejecutar en la línea de comandos lo siguiente:

- *bundle install*: instala las dependencias de la aplicación web.
- *rake db:create*: crea la base de datos si no está creada.
- *rake db:schema:load*: carga el schema de la base de datos.
- *rake db:test:prepare*: prepara la base de datos aislada para los test.
- *rake minitest test*: ejecuta los test.

Como se muestra en la Figura 35, después de ejecutar los comandos y haber pasado los test se muestra el resultado de los mismos, con el plugin *"HTML reports"* se muestra la cobertura de los test, y si ha pasado los test correctamente, se ejecutará la tarea que despliega la aplicación web.

Segunda tarea: WebHypervisorCentralizado

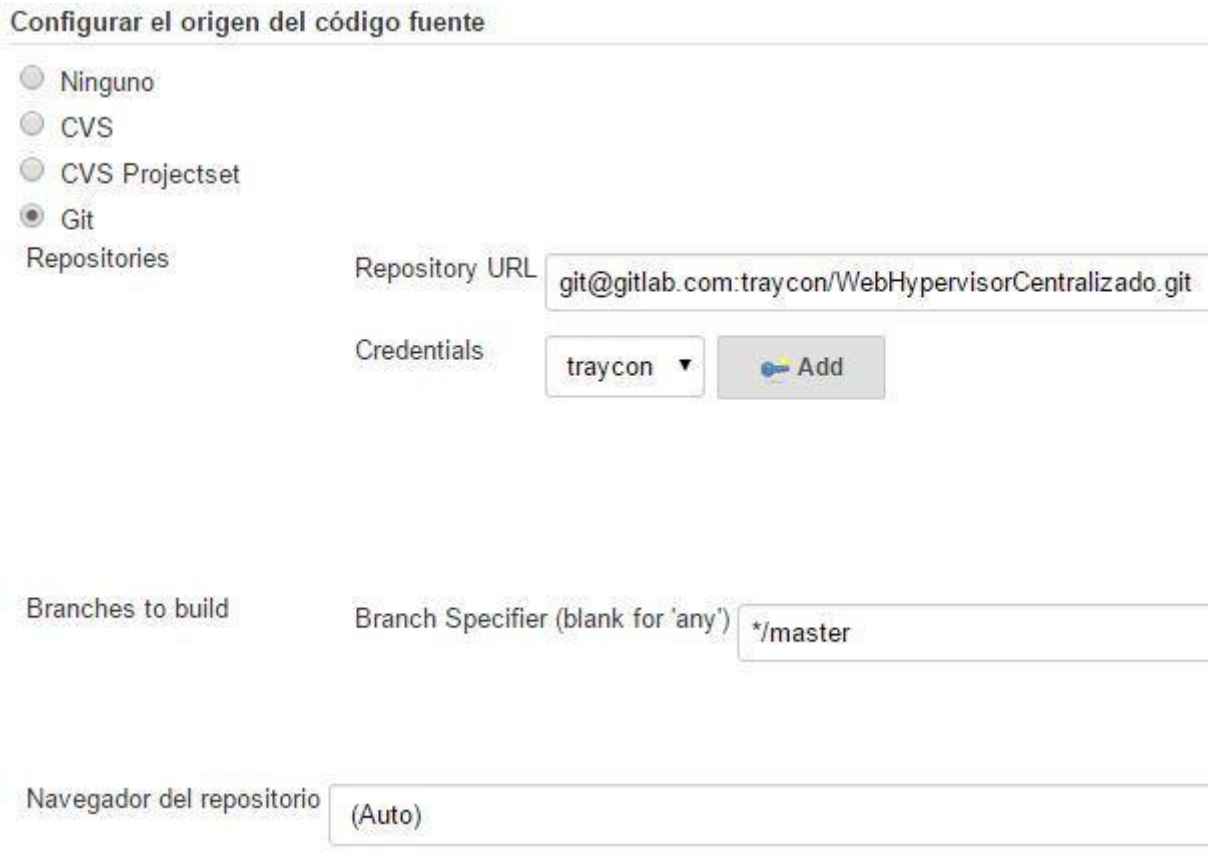

Fig.36: Configuración Jenkins tarea WebHypervisorCentralizado – parte 1.

Como se muestra en la Figura 37, se ejecuta el comando *rails s –p 3001* que habilita el servidor web de *Rails* en el puerto 3001, si el despliegue fuera solo en local únicamente habría que acceder a la url http://localhost:3001. En este caso, se va a habilitar la aplicación web para acceder desde cualquier lugar, para ello hay que configurar en el servidor web Nginx un proxypass que vaya al puerto 3001, de tal manera que se pueda acceder desde el navegador a la aplicación web.

Descarga de la aplicación cliente:

- Una vez desplegada la aplicación web, accedemos a ella a través del navegador web, como se muestra en la Figura 38.
- En la página de inicio habrá un apartado como en la Figura 38, que ponga *"2. Descargue y ponga en marcha nuestra aplicación cliente.",* tan solo habrá que seguir los pasos que allí se describen para descargar la aplicación cliente según el sistema operativo que vaya a monitorizar.
- Una vez descargada la aplicación cliente adecuada, hay que registrarse en la web, iniciar sesión con la cuenta y entrar en perfil, allí se encuentra la *secret key* que se debe copiar en la aplicación cliente cuando se ejecute.

Fig.37: Configuración Jenkins tarea WebHypervisorCentralizado – parte 2.

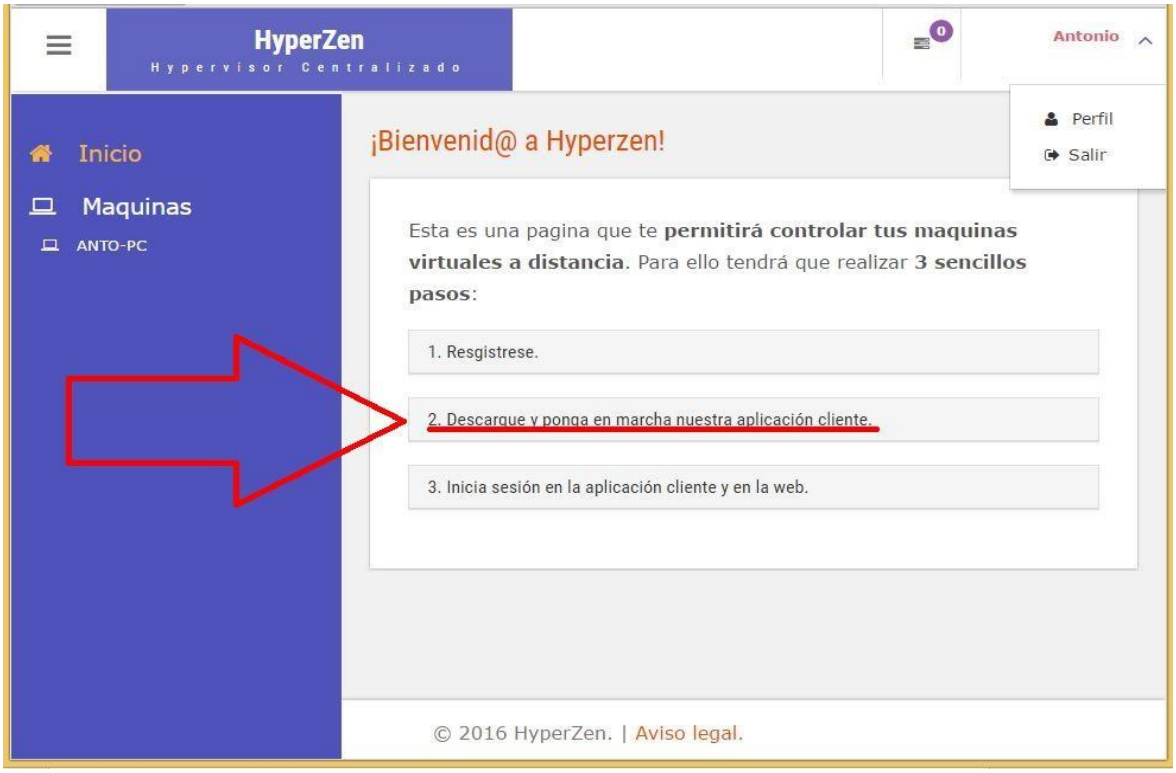

Fig.38: Detalles de la descarga de la aplicación cliente.

5.2. Despliegue en cualquier entorno

1.- Requisitos:

- Tener instalado Ruby 2.1.6 o superior.
- Tener instalado Git.

2.- Preparación del entorno:

- Instalar RubyOnRails, para ello solo habrá que ejecutar *"gem install rails"*
- Instalar Bundler, *"gem install bundler"*
- Instalar Rake, *"gem install rake"*
- Instalar Qtbindings, *"gem install Qtbindings"*

3.- Clonar el repositorio en una carpeta cualquiera y, después entrar en ella:

- *git clone git@gitlab.com:traycon/WebHypervisorCentralizado.git*
- *cd WebHypervisorCentralizado*

4.- Preparación de la aplicación web, ejecutar los siguientes comandos:

- *bundle install*: instala las dependencias de la aplicación web.
- *rake db:create*: crea la base de datos si no está creada.
- *rake db:schema:load*: carga el schema de la base de datos.
- *rake db:test:prepare*: prepara la base de datos aislada para los test.
- *rake minitest test*: ejecuta los test y si todo ha salido bien ya podemos iniciar la aplicación web.

5.- Para iniciar la aplicación web solo resta ejecutar el siguiente comando:

rails s –p <puerto>

Normalmente se utiliza el puerto 80 así que queda de la siguiente manera:

rails s –p 80

6.- Ya se puede acceder a la aplicación web entrando en http://localhost:<puerto>.

7.- Descarga de la aplicación cliente tal y como se ha explicado en el apartado anterior.

6. Conclusiones y trabajo futuro

En este trabajo se ha diseñado e implementado un servicio web que permite la gestión de MVs, lo he finalizado con una grata satisfacción personal debido al aprendizaje y el producto obtenidos. El código es libre y puede descargarse el código completo en https://gitlab.com/traycon/WebHypervisorCentralizado.git, para desplegarlo y usarlo en local o bien se puede acceder a http://www.hyperzen.tk/ y descargando la aplicación cliente empezar a usar la aplicación.

No ha sido tarea sencilla aprender a utilizar el framework que me ha ayudado a desarrollar el trabajo, teniendo que haberme leído libros enteros sobre RubyOnRails [6][9] antes de

empezar, porque incluso los diagramas que aunque intentan no ser dependientes del lenguaje, están muy condicionados por él.

Uno de los grandes problemas que me ocurren durante la programación, es la incertidumbre de saber si lo que acabo de programar es de la forma más correcta y lógica posible. Hay que diferenciar el que una aplicación funcione de que esté realmente bien hecha. En mi caso, suelo invertir mucho tiempo en la segunda opción, casi más que en la primera, motivo por el que siempre estoy buscando todo lo que hago o quiero hacer en sitios como StackOverflow, donde alguien plantea un problema y otros responden la solución que darían a dicho problema.

He intentado modularizar la aplicación para que sea más mantenible en un futuro y más legible, aspectos que facilita RubyOnRails, aunque creo que todavía se podía haber desglosado más, pero con mis conocimientos actuales quedaba quizás algo lejos.

Creo que el tipo de aplicación que he desarrollado en un futuro será lo que se encuentre en todos lados. Cada vez se desarrollan más aplicaciones web y se dejan de lado las de escritorio por el simple motivo de que las aplicaciones sean multiplataforma, además de otras ventajas como el que no haya que instalar la aplicación en el sistema, que sea accesible desde un navegador, etc...

Personalmente estoy satisfecho con el sistema que he conseguido, aunque se podrían mejorar algunos aspectos, como el de reconocer más de un disco duro virtual si los hubiera.

Mi objetivo es continuar desarrollando el trabajo incluso tras haber terminado el grado de Ingeniería Informática, añadiéndole nuevas funcionalidades, arreglando los posibles errores que encuentre y convirtiéndola en una aplicación que algún día le sea de utilidad a alguien. Si alguien que ha usado la aplicación me dijera que le es de utilidad, no existiría para mí mayor satisfacción y motivación para continuar.

Las nuevas funcionalidades que podría añadir en un futuro son:

- Ampliar las opciones de gestión de las MVs.

  Actualmente solo se pueden gestionar la capacidad de la RAM, el aumento de la capacidad del disco duro principal (si se tienen más discos duros no se tienen en cuenta), apagado, encendido, etc… y se podrían añadir nuevos aspectos de la configuración de las MVs como las interfaces de red y sus modos, poder crear nuevas interfaces, etc.

- Añadir la capacidad de crear una MV.

  Actualmente no se pueden crear MVs desde la aplicación web.

- Añadir la capacidad de eliminar una MV.

  Actualmente no se pueden eliminar las MVs.

- Posibilidad de crear grupos de usuarios.

  Si es un sistema en el que gestionan las MVs varios usuarios, poder hacer grupos de usuarios es una buena idea para que todos los usuarios del mismo grupo tengan accesibles las mismas MVs. Esta mejora implica llevar un registro de las acciones realizadas por cada usuario del grupo, de tal forma que se pueda averiguar lo sucedido analizando el histórico de acciones ejecutadas.

- Mejora del vnc.

  Actualmente es un applet de tightvnc, pero esto condiciona a que el navegador pueda ejecutar el applet y que el usuario tenga java instalado en su sistema. Se solucionaría por ejemplo con servicio vnc basado en websocket como el que utiliza actualmente OpenStack pero este habría que modificarlo para que no dejará acceder solo a conexiones locales y habiendo estudiado un poco el tema. He de decir que es complejo para el tiempo estimado de este TFG.

- Soporte para todos lo hypervisores.

  Actualmente solo he creado soporte para VmWare y VirtualBox, añadir más hypervisores no sería problema por cómo está estructurada la aplicación web, en la que para dar soporte a otro hypervisor tan solo habría que añadir una serie de comandos de forma parametrizada. Pero esto requiere de conocer el hypervisor, estudiar sus características y configuraciones. Se recomienda hacer pruebas en local antes de añadirlo a la aplicación web, lo que requiere de bastante tiempo para cada hypervisor.

7. Bibliografía

[1] Azure.microsoft.com. (2016). *Microsoft Azure: plataforma y servicios de informática en la nube*. [online] Available at: https://azure.microsoft.com/es-es/

[2] Cms.ual.es. (2016). *Asignatura: Herramientas y Métodos de Ingeniería del Software - Grado en Ingeniería Informática (Plan 2015) - Universidad de Almería*. [online] Available at: http://cms.ual.es/UAL/estudios/grados/plandeestudios/asignaturas/asignatura/GRADO4015?idAss=40153306

[3] Cms.ual.es. (2016). *Asignatura: Administración de Redes y Sistemas Operativos - Grado en Ingeniería Informática (Plan 2015) - Universidad de Almería*. [online] Available at: http://cms.ual.es/UAL/estudios/grados/plandeestudios/asignaturas/asignatura/GRADO4015?idAss=40154324

[4] GitLab. (2016). *Code, test, and deploy together with GitLab open source git repo management software*. [online] Available at: https://gitlab.com/

[5] Guides.rubyonrails.org. (2016). *Ruby on Rails Guides*. [online] Available at: http://guides.rubyonrails.org/

[6] Hartl, M. (n.d.). *Ruby on rails tutorial*.

[7] Jenkins.io. (2016). *Jenkins*. [online] Available at: https://jenkins.io/

[8] Nginx.org. (2016). *nginx documentation*. [online] Available at: https://nginx.org/en/docs/

[9] Ruby, S., Thomas, D. and Hansson, D. (n.d.). *Agile web development with Rails 4*.

[10] Ruby-lang.org. (2016). *Documentación*. [online] Available at: https://www.ruby-lang.org/es/documentation/

[11] Virtualbox.org. (2016). *Chapter 8. VBoxManage*. [online] Available at: https://www.virtualbox.org/manual/ch08.html

[12] VmWare.com. (2016). *VmWare Workstation Documentation*. [online] Available at: http://www.VmWare.com/es.html

[13] VmWare.com. (2016). VmWare vSphere y vSphere with Operations Management. [online] Available at: http://www.VmWare.com/es/products/vsphere.html

[14] Openstack.org. (2016). OpenStack Docs: Current. [online] Available at: http://docs.openstack.org/

8. Anexo

A continuación, se muestran los scripts para diferentes hypervisores:

VmWare en el sistema operativo Windows [12]

- Script de búsqueda de MVs y formateo de la información para su envío al servidor:

```
echo off & FOR /F "tokens=*" %r IN ('cd / ^& dir /s /b ^"*.vmx^" ^2^>nul ^| findstr /v ^".vmxf^"') DO (echo "---mv---"&set totalhdd=0 & FOR /F "tokens=2 delims==" %a IN ('type ^"%r^" ^| findstr "displayName"') DO (set "name=%a" & call echo %^name:~1%) & echo ^"%r^"& FOR /F "tokens=2 delims==" %a IN ('wmic memoryChip get Capacity /value') DO (echo %a) & FOR /F "tokens=2 delims==" %a IN ('type ^"%r^" ^| findstr "memsize"') DO (set "mem=%a" & call echo %^mem:~2,-1% MB) & FOR /F "tokens=2 delims==" %a IN ('type ^"%r^" ^| findstr "cleanShutdown"') DO (set "off=%a" & FOR /F "tokens=2 delims==" %b IN ('type ^"%r^" ^| findstr "checkpoint.vmState"') DO (set "suspendido=%b" & FOR /F "tokens=*" %c IN ('call echo %^off:~1%') DO (FOR /F "tokens=*" %d IN ('call echo %^suspendido:~1%') DO (if /i %c=="TRUE" (echo poweredoff) else (if /i %d=="" (echo running) else (echo saved)))))) & FOR /F "tokens=2 delims==" %a IN ('type ^"%r^" ^| findstr ".vmdk"') DO (set "phdd=%a" & call echo %^phdd:~2,-1%& cd "%~dpr" & FOR /F "tokens=2 delims= " %b IN ('call type %^phdd:~1% ^| findstr "SPARSE" ^| findstr "RW"') DO (call set /a totalhdd+=%b/2 1>nul))& call echo %^totalhdd% KB& FOR /F "tokens=3 delims= " %a IN ('dir ^| findstr "bytes"') DO (set "libre=%a" & call echo %^libre:.=% Bytes) )& call echo galeria de snapshots & call dir /a:d /b Snapshots 2>nul
```

- Comando para encender la MV:

```
start ##ruta##
```

- Comando para apagar la MV:

```
FOR /F "tokens=*" %r IN ("##nombre##") DO (set "nombre=%r" & FOR /F "tokens=*" %b IN ('call echo %^nombre:~1^,-1%.vmx') DO ( FOR /F "tokens=3 delims=," %a IN ('wmic process get commandline^,processid^,handle /format^:csv ^| call findstr /C:"%b" ^| findstr /v "findstr"') DO (taskkill /pid %a /f /t))) & FOR /F "tokens=*" %b IN ("##ruta##") DO (cd "%~dpb") & FOR /F "tokens=*" %b IN ('call dir /a:d /b 2^>nul ^| findstr ".lck"') DO (rd /q /s "%b") & FOR /F "tokens=*" %b IN ('call dir /b 2^>nul ^| findstr ".vmem"') DO (del /f /q /s "%b") & type "##ruta##" | findstr /v "cleanShutdown" > "##ruta##1" & echo cleanShutdown = "TRUE" >> "##ruta##1" & type "##ruta##1" > "##ruta##" & del "##ruta##1"
```

- Comando para reiniciar la MV:

```
FOR /F "tokens=*" %r IN ("##nombre##") DO (set "nombre=%r" & FOR /F "tokens=*" %b IN ('call echo %^nombre:~1^,-1%.vmx') DO ( FOR /F "tokens=3 delims=," %a IN ('wmic process get commandline^,processid^,handle /format^:csv ^| call findstr /C:"%b" ^| findstr /v "findstr"') DO (taskkill /pid %a /f /t))) & FOR /F "tokens=*" %b IN ("##ruta##") DO (cd "%~dpb") & FOR /F "tokens=*" %b IN ('call dir /a:d /b 2^>nul ^| findstr ".lck"') DO (rd /q /s "%b") & FOR /F "tokens=*" %b IN ('call dir /b 2^>nul ^| findstr ".vmem"') DO (del /f /q /s "%b") & type "##ruta##" | findstr /v "cleanShutdown" > "##ruta##1" & echo cleanShutdown = "TRUE" >> "##ruta##1" & type "##ruta##1" > "##ruta##" & del "##ruta##1" & start ##ruta##
```

- Comando para hacer una snapshot de la MV:

```
FOR /F "tokens=*" %b IN ("##ruta##") DO (cd "%~dpb") & mkdir Snapshots & cd Snapshots & mkdir "##valor##" & cd .. & FOR /F "tokens=*" %b IN ('call dir /a:a /b 2^>nul ^| findstr ".vmdk"') DO (copy "%b" "Snapshots\##valor##\%b")
```

- Comando para restaurar una snapshot de la MV:
  FOR /F "tokens=*" %b IN ("##ruta##") DO (cd "%~dpb") & FOR /F "tokens=*" %b IN ('dir /b "Snapshots\##valor##"') DO (copy /v /y "Snapshots\##valor##\%b" "%b")

- Comando para eliminar una snapshot de la MV:
  FOR /F "tokens=*" %b IN ("##ruta##") DO (cd "%~dpb") & rd /q /s "Snapshots\##valor##"

- Comando para modificar el tamaño de la memoria de la MV:

  FOR /F "tokens=*" %r IN ("##nombre##") DO (set "nombre=%r" & FOR /F "tokens=*" %b IN ('call echo %^nombre:~1^,-1%.vmx') DO (FOR /F "tokens=3 delims=," %a IN ('wmic process get commandline^,processid^,handle /format^:csv ^| call findstr /C:"%b" ^| findstr /v "findstr"') DO (taskkill /pid %a /f /t))) & FOR /F "tokens=*" %b IN ("##ruta##") DO (cd "%~dpb") & FOR /F "tokens=*" %b IN ('call dir /a:d /b 2^>nul ^| findstr ".lck"') DO (rd /q /s "%b") & FOR /F "tokens=*" %b IN ('call dir /b 2^>nul ^| findstr ".vmem"') DO (del /f /q /s "%b") & type "##ruta##" | findstr /v "cleanShutdown" > "##ruta##1" & echo cleanShutdown = "TRUE" >> "##ruta##1" & type "##ruta##1" > "##ruta##" & del "##ruta##1" & type "##ruta##" | findstr /v "memsize" > "##ruta##1" & echo memsize = "##valor##" >> "##ruta##1" & type "##ruta##1" > "##ruta##" & del "##ruta##1"

- Comando para aumentar la capacidad del disco duro de la MV:

  FOR /F "tokens=*" %r IN ("##nombre##") DO (set "nombre=%r" & FOR /F "tokens=*" %b IN ('call echo %^nombre:~1^,-1%.vmx') DO (FOR /F "tokens=3 delims=," %a IN ('wmic process get commandline^,processid^,handle /format^:csv ^| call findstr /C:"%b" ^| findstr /v "findstr"') DO (taskkill /pid %a /f /t))) & FOR /F "tokens=*" %b IN ("##ruta##") DO (cd "%~dpb") & FOR /F "tokens=*" %b IN ('call dir /a:d /b 2^>nul ^| findstr ".lck"') DO (rd /q /s "%b") & FOR /F "tokens=*" %b IN ('call dir /b 2^>nul ^| findstr ".vmem"') DO (del /f /q /s "%b") & type "##ruta##" | findstr /v "cleanShutdown" > "##ruta##1" & echo cleanShutdown = "TRUE" >> "##ruta##1" & type "##ruta##1" > "##ruta##" & del "##ruta##1" & FOR /F "tokens=*" %r IN ("##ruta##") DO (set aumentar=##valor## & call set /a aumentar*=2048 & cd "%~dpr" & call set first=1 & FOR /F "tokens=2 delims= " %b IN ('call type "##rutaHDD##" ^| findstr "SPARSE" ^| findstr "RW"') DO (for /f %a in ('call echo %^first%') do (if %a==1 call set /a aumentar+=%b & call set first=0)) & call set first=1 & FOR /F "tokens=* delims= " %b IN ('call type "##rutaHDD##" ^| findstr "SPARSE" ^| findstr "RW"') DO (for /f %a in ('call echo %^first%') do (if %a==1 (type "##rutaHDD##" | findstr /v /C:"%b" > "##rutaHDD##1" & call set first=0 & for /f delims^=^"^ tokens^=2 %d in ("%b") do (set "vmdk=%d")))) & call echo RW %^aumentar% SPARSE "%^vmdk%">> "##rutaHDD##1" & type "##rutaHDD##1" > "##rutaHDD##" & del "##rutaHDD##1")

KVM en sistema operativo Linux (Debian) [3]

- Comando para encender la MV:
  virsh start ##nombre##

- Comando para apagar la MV:
  virsh shutdown ##nombre##

- Comando para reiniciar la MV:
  virsh reboot ##nombre##

- Comando para hacer una snapshot de la MV:

  virsh snapshot-create-as ##nombre## ##valor## --disk-only --atomic

- Comando para restaurar una snapshot de la MV:

  virsh shutdown ##nombre## && virsh snapshot-revert ##nombre## ##valor##

- Comando para eliminar una snapshot de la MV:

  virsh shutdown ##nombre## && virsh snapshot-delete ##nombre## ##valor##

- Comando para modificar el tamaño de la memoria de la MV:

  virsh shutdown ##nombre## && virsh setmem ##nombre## ##valor##M

- Comando para aumentar la capacidad del disco duro de la MV:

  virsh shutdown ##nombre## && qemu-img resize ##rutaHDD## ##valor##M

UNIVERSIDAD DE ALMERÍA

Resumen/Abstract

El siguiente proyecto tratara sobre la realización de un hypervisor centralizado, que permita realizar un mantenimiento básico de las máquinas virtuales (guardar el estado actual de la máquina, restaurar una maquina a un estado anterior, apagarla, reiniciarla, encenderla,...) creadas en cualquier equipo, de forma remota y sin importar el hypervisor utilizado para las diferentes máquinas virtuales, ya sea VMware, VirtualBox, KVM, XEN...

Así pues, los aspectos principales del trabajo son:

1. Desarrollo de una aplicación web con autentificación de usuarios, donde el usuario podrá visualizar el estado de cada una de las máquinas virtuales y gestionarlas desde la propia web.

2. Desarrollo de una aplicación cliente, la cual debe en primera instancia autentificar al usuario con una secret key (AccessToken). El siguiente paso a realizar será reconocer el S.O. para determinar así que comandos utilizar para detectar los hypervisores y MVs que posee el sistema. Posteriormente, esta información se enviará a la web donde el usuario podrá gestionar sus MVs. Una vez ocurre la gestión, el programa cliente comprobará en la web si hay alguna orden relativa a sus MVs y ejecutará el comando adecuado, según el hypervisor utilizado. Finalmente, se enviará el éxito o fracaso de esta operación a la web actualizando el estado de las MVs.