UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

CREACIÓN DE UN LABORATORIO DE ACCESO REMOTO MULTIPLATAFORMA PARA PROCESOS DE QUALITY ASSURANCE (QA) TESTING

Trabajo Fin de Máster

Máster Universitario en Ingeniería Informática

AUTOR/A: Valero Fernández, Pablo

Tutor/a: Acebrón Linuesa, Floreal

CURSO ACADÉMICO: 2021/2022

Resumen

Este proyecto trata sobre la creación, configuración y desarrollo de un laboratorio de pruebas de *software* para procesos de *Quality Assurance*. En otras palabras, asegurar que los requisitos de un producto sean satisfechos por parte de los desarrolladores de este, mediante la realización de pruebas de calidad del código generado.

Creando una subred dentro de la *LAN* de la empresa con el rúter *Mikrotik RB2011IL-RM*, aislamos el laboratorio con el exterior. A través de la gestión de un PC máster, el cual tiene el sistema de aprovisionamiento *Cobbler* instalado, administramos los sistemas operativos, controladores y versiones de estos, necesarios para realizar las pruebas de calidad del *software* a los PCs cliente del laboratorio. Los nodos inicialmente no disponen de sistema operativo propio, lo instalamos desde el máster a través de la red.

Una vez instalados, los nodos son capaces de iniciarse con el S.O. elegido y administrado por el PC máster sin intervención humana, así conseguimos conectarnos a ellos vía *SSH/VPN* para su despliegue remoto. Este proceso permite a los usuarios poder utilizar desde sus casas o puestos de trabajo el laboratorio de pruebas.

Para la instalación e inicio del S.O. administrado por el máster, necesitamos configurar la *BIOS/UEFI* de los nodos con arranque *PXE* por red mediante *DHCP*, esto permite al usuario elegir el nodo que desea utilizar a través de la dirección *MAC* de la tarjeta de red, esta configuración está administrada por *Cobbler* y se hace de manera opaca al usuario del laboratorio mediante una interfaz *web*.

Palabras clave: laboratorio, *Quality Assurance*, rúter, *Cobbler*, despliegue remoto, sistema operativo, interfaz *web*.

Abstract

This project is about the creation, configuration and development of a software testing laboratory for Quality Assurance processes. In other words, ensuring that the requirements of a product are met by the developers of the product, by testing the quality of the generated code.

By creating a subnet within the company's LAN with the Mikrotik RB2011IL-RM router, we isolate the laboratory from the outside world. Through the management of a master PC, which has the Cobbler provisioning system installed, we manage the operating systems, drivers and versions of these necessary to carry out the quality tests of the software to the PC clients of the laboratory. The nodes initially do not have their own operating system, we install it from the master via the network.

Once installed, the nodes are able to boot up with the O.S. chosen and managed by the master PC without human intervention, so we can connect to them via SSH/VPN for remote deployment. This process allows users to use the test lab from their homes or workstations.

For the installation and start of the O.S. managed by the master, we need to configure the BIOS/UEFI of the nodes with PXE boot by network through DHCP, this allows the user to choose the node he wants to use through the MAC address of the network card, this configuration is managed by Cobbler and is done in an opaque way to the lab user through a web interface.

Key words: laboratory, Quality Assurance, router, Cobbler, remote deployment, operating system, web interface.

Resum

Aquest projecte tracta sobre la creació, configuració i desenvolupament d'un laboratori de proves de *software* per a processos de *Quality Assurance*. En altres paraules, assegurar que els requisits d'un producte siguen satisfets per part dels desenvolupadors d'aquest, mitjançant la realització de proves de qualitat del codi generat.

Creant una subxarxa dins de la *LAN* de l'empresa amb el rúter *Mikrotik RB2011IL-RM*, aïllem el laboratori amb l'exterior. A través de la gestió d'un PC màster, el qual té el sistema d'aprovisionament *Cobbler* instal·lat, administrem els sistemes operatius, controladors i versions d'aquests necessaris per a realitzar les proves de qualitat del *software* als PCs clients del laboratori. Els nodes inicialment no disposen de sistema operatiu propi, ho instal·lem des del màster a través de la xarxa.

Una vegada instal·lats, els nodes són capaços d'iniciar-se amb el S.O. triat i administrat pel PC màster sense intervenció humana, així aconseguim connectar-nos a ells via *SSH/VPN* per al seu desplegament remot. Aquest procés permet als usuaris poder utilitzar des de les seues cases o llocs de treball el laboratori de proves.

Per a la instal·lació i inici del S.O. administrat pel màster, necessitem configurar la *BIOS/UEFI* dels nodes amb arrancada *PXE* per xarxa mitjançant *DHCP*, això permet a l'usuari triar el node que desitja utilitzar a través de la direcció *MAC* de la targeta de xarxa, aquesta configuració està administrada per *Cobbler* i es fa de manera opaca a l'usuari del laboratori mitjançant una interfície *web*.

Paraules clau: laboratori, *Quality Assurance*, rúter, *Cobbler*, desplegament remot, sistema operatiu, interfície *web*.

Índice general

Índice de figuras

Glosario de términos

Archivo *ISO*: achivo que se utiliza para almacenar una copia exacta de un sistema de ficheros de una unidad óptica.

Archivo *kikstart*: archivo que contiene una serie de opciones, que serán transferidas al instalador para configurar el sistema operativo automáticamente.

***BIOS*:** *Basic Input-Output System*, medio encargado de indicar al ordenador las funciones básicas de arranque, control, tareas de identificación y configuración del *hardware*.

***Cobbler*:** sistema de aprovisionamiento *Linux* que facilita la instalación de medios basados en red de múltiples S.O.

***Firewall*:** parte de un sistema informático encargado de bloquear los accesos no autorizados a la red, permitiendo únicamente comunicaciones autorizadas.

***GPU*:** *Graphics Processing Unit*, coprocesador dedicado al procesamiento de gráficos y operaciones de coma flotante.

***LAN*:** *Local Area Network*, red de computadoras que permite la comunicación y el intercambio de datos entre diferentes dispositivos a nivel local.

***PXE*:** *Preboot eXecution Environment*, entorno para arrancar e instalar el sistema operativo en computadoras a través de una red.

***Quality Assurance*:** conjunto de actividades planificadas y sistemáticas aplicadas en un sistema de gestión de la calidad para que los requisitos de calidad de un producto o servicio sean satisfechos.

***SELinux*:** módulo de seguridad para el *kernel Linux*.

Servidor *DHCP*: *Dynamic Host Configuration Protocol*, protocolo cliente/servidor que proporciona automáticamente una dirección *IP*, máscara de subred y la puerta de enlace predeterminada, a un PC determinado.

Servidor *DNS*: *Domain Name System*, protocolo encargado de traducir las direcciones *IP* a nombres de dominio.

***SSH*:** *Secure Shell*, protocolo que permite el acceso remoto a otro dispositivo por medio de un canal seguro en el que toda la información se transmite cifrada.

***UEFI*:** *Unified Extensible Firmware Interface*, sucesor de la *BIOS* con funciones y mejoras adicionales, como una interfaz gráfica moderna, un sistema de inicio seguro, mayor velocidad de arranque, etc.

***VPN*:** *Virtual Private Network*, herramienta utilizada para conectarse a *Internet* de forma privada, ocultando la dirección *IP* y enrutando el tráfico mediante un túnel privado y cifrado de forma segura a través de redes públicas.

***WAN*:** *Wide Area Network*, red de computadoras que permite la comunicación y el intercambio de datos entre diferentes redes.

CAPÍTULO 1

Introducción

Las empresas de desarrollo *software* habitualmente se encuentran con la problemática de no poder validar completamente los proyectos desarrollados para sus clientes, en el ciclo de vida del *software*[1], equivale a las fases de Integración y Pruebas. Si esta parte no se realiza con éxito, implica la aparición y detección de errores una vez se ha entregado el producto al cliente. Para conseguir una funcionalidad lo más completa posible, es necesario contar con una tecnología adecuada y potente que permita testar el producto correctamente.

Realizando las prácticas curriculares del Máster Universitario en Ingeniería Informática en Arisnova S.L. Ingeniería de Sistemas[2], una empresa especializada en el desarrollo de aplicaciones *software* a distintos niveles, *Web*, 2D, 3D, etc. Enfocadas a la recogida, tratamiento y análisis de todo tipo de datos mediante una potente plataforma, capaz de gestionar un gran volumen de datos en tiempo real y conectar las diferentes fuentes de información. En dicha empresa se enfrentan a diario a la problemática presentada anteriormente, por lo que nos ofrecieron la posibilidad de crear un laboratorio a pequeña escala, en el cual poder validar el código desarrollado. Para ello era necesario la utilización de un sistema de aprovisionamiento de servicios que permitiese realizar pruebas de *Quality Assurance*[3] en distintos sistemas operativos, versiones, controladores y/o paquetes dependiendo del proyecto a testear.

QA es una forma de prevenir errores, defectos y evitar problemas al entregar productos o servicios a los clientes. La *ISO 9000*[4] lo define como «parte de la gestión de la calidad centrada en proporcionar confianza en que se cumplirán los requisitos de calidad».

1.1 Motivación

La asignatura de «Configuración y Optimización de Sistemas de Cómputo» del primer semestre del Máster Universitario en Ingeniería Informática, está muy bien enfocada al aprendizaje práctico por parte del alumno. Se adquieren los conocimientos básicos para poder llevar a cabo la configuración *hardware* de sistemas de cómputo (servidores, clústers, sistemas híbridos) y sus componentes (nodos, red, sistema de memoria y almacenamiento). La configuración *software* del sistema (instalación del sistema operativo, proceso de clonación, herramientas de gestión y administración), sistemas de archivos, herramientas de distribución de carga, herramientas de monitorización, etc.

Estos son unos conocimientos que hoy en día están muy demandados por empresas tecnológicas, son realmente aplicables y tienen un gran potencial. Surgió la idea de realizar un proyecto que nos permitía ampliar nuestra formación en este campo y a su vez sería de gran utilidad para la empresa.

1.2 Objetivos

El objetivo general del proyecto es desarrollar un laboratorio de pruebas para conectar remotamente un conjunto de PCs (nodos/cliente) sin sistema operativo, pero con disco duro local para almacenar los datos generados durante las pruebas, a un servidor (gestor/máster). El nodo maestro dispondrá de un repositorio de *ISOs* con distintas distribuciones (*Fedora*, *RedHat*, *Ubuntu*, etc). El sistema operativo seleccionado se instalará en el almacenamiento local del nodo cliente elegido. Este proceso se automatizará haciendo uso del sistema de aprovisionamiento *Cobbler*[5]. En función del código a testar se seleccionará una determinada *ISO* de instalación.

El proyecto consta de cuatro bloques:

- Primero, creación y configuración de una red específica y separada de la empresa. Mediante un *firewall* se permitirán las entradas de los usuarios a esta *LAN* desde la red de la empresa, pero se bloqueará cualquier salida al exterior no deseada por los administradores de la red. Dentro de la *LAN* de pruebas, necesitaremos un servidor *DHCP* y un servidor *DNS*, este servicio lo proporcionaremos con el rúter *Mikrotik RB2011IL-RM*[6];

- Segundo, configuración del sistema de aprovisionamiento *Cobbler* en el nodo máster, detección de los nodos a traveś de la red y lanzamiento del sistema operativo seleccionado por el usuario, los PCs cliente dispondrán de un arranque por red *DHCP-PXE*[7];

- Tercero, implementación de un mecanismo para poder almacenar una copia de todo el entorno de ejecución de un nodo, incluyendo, sistema operativo, aplicaciones, controladores, contenido del disco local, etc. Mediante el *software* de recuperación de una imagen creada de un sistema operativo *Clonezilla*[8]. El archivo generado se almacenará en un segundo disco dentro del nodo cliente para su posterior uso. Se generará un fichero conteniendo todo lo que se ha añadido a la distribución base y se guardará dentro de esta;

- Cuarto y último, creación de una interfaz *web* simple, que facilitará el manejo del laboratorio a los usuarios de este, seleccionando el nodo y la distribución a instalar.

1.3 Impacto esperado

Poder realizar pruebas de *QA* sobre el *software* desarrollado dentro de la propia empresa, realizando un laboratorio a medida, con gran adaptabilidad al código testado, realizando pruebas con dos o más nodos en paralelo, ampliar o reducir el laboratorio a demanda y actualizarlo con los S.O. que vayan necesitando los clientes, es un avance significativo dentro de la calidad del producto que vende Arisnova S.L. y que puede marcar la diferencia respecto a su competencia.

El laboratorio en cuestión no sólo sirve para realizar pruebas internas del producto o *software* que vende la empresa, sino que puede ser un producto más que esta ofrezca a sus clientes para mejorar su competitividad, productividad, costes de fabricación y reduciendo los errores de producción.

Actualmente el problema principal para la venta y distribución del producto es la configuración de los distintos nodos y la necesidad de crear una subred a parte para el laboratorio, las pruebas se han realizado con un determinado rúter, y unos determinados PCs, con versiones específicas de la placa base y de la *BIOS* o *UEFI*, crear un producto de este laboratorio, supondría probar otro *hardware* quizá más actual y comprobar su compatibilidad y desempeño para este fin.

1.4 Metodología

Para el cumplimiento de los objetivos se proponen los siguientes pasos:

- Creación y configuración de una *LAN* para el laboratorio;
- Generación de las reglas del *firewall* para introducir seguridad a la red;
- Configuración del rúter como servidor *DHCP* y *DNS*;
- Instalación y configuración del sistema de aprovisionamiento *Cobbler* en el PC máster;
- Descarga de los S.O. y configuración de distribuciones, perfiles y sistemas en *Cobbler*;
- Configuración de ficheros necesarios para el arranque del sistema operativo en el nodo cliente mediante *PXE-DHCP*;
- Configuración de la *BIOS-UEFI* del PC cliente;
- Pruebas de lanzamiento de los S.O. en el nodo cliente;
- Errores encontrados y mejoras propuestas para la creación del laboratorio;
- Validación de las pruebas y correcto funcionamiento;
- Generación de una copia completa del sistema operativo instalado y modificado en el PC cliente, almacenándola en el segundo disco que este contiene;
- Creación de una interfaz *web* simple y funcional que permita a los usuarios utilizar el laboratorio de pruebas de *QA* sin necesitar el emulador de terminal del S.O.

1.5 Estructura de la memoria

Dicho trabajo está compuesto por las siguientes secciones:

- Introducción. Se plantea superficialmente el trabajo a realizar, las motivaciones que lo han hecho posible, los objetivos que hemos ido alcanzando, el impacto esperado del proyecto realizado, la metodología a seguir y por último la estructura de este;

- Estado de la cuestión. Se realiza un estudio de los productos que hay similares en el mercado actualmente y las necesidades concretas de la empresa, al mismo tiempo que presentamos una propuesta de nuestro proyecto;

- Análisis del problema. Se plantean los requisitos especificados por la empresa conjuntamente con un análisis de seguridad, protección de datos y propiedad intelectual de los distintos *software* del proyecto. Sumando la elección de los componentes y programas utilizados;

- Solución propuesta. Se plasma el plan de trabajo, el presupuesto del proyecto, el diseño de la solución, la arquitectura del sistema, se proporciona un diseño detallado del desarrollo del trabajo realizado, la tecnología utilizada y por último, en el desarrollo de la solución propuesta, se explica el procedimiento seguido hasta lograr una solución satisfactoria en el desarrollo del proyecto;

- Implantación. Se lleva el desarrollo del proyecto a explotación, se plantean las diversas pruebas ejecutadas con las modificaciones aplicadas, para mostrar los resultados obtenidos fruto del trabajo realizado;

- Conclusiones. Se exponen las conclusiones que hemos obtenido con el desarrollo del trabajo, a la vez que los conceptos y experiencias adquiridos durante el transcurso del proyecto y la relación de este con los estudios cursados. Además de los posibles trabajos futuros que podrían desarrollarse o ampliarse a raíz de este laboratorio;

- Referencias. Se detallan las distintas fuentes de información utilizadas para complementar o fundamentar el trabajo realizado;

- Apéndice A. Se exponen los distintos archivos de configuración del sistema de aprovisionamiento *Cobbler*;

- Apéndice B. Se muestra el código generado para la creación de la interfaz *web* en sus distintas partes;

- Apéndice C. Se relaciona el trabajo final de máster realizado con los objetivos de desarrollo sostenible;

- Glosario de términos. Al inicio de la memoria se ofrece un pequeño glosario en el cual se explican los distintos términos técnicos mencionados en este documento.

CAPÍTULO 2

Estado de la cuestión

Actualmente las pruebas de *QA* se realizan bien por aplicaciones profesionales que se encargan de detectar los errores mediante tareas repetitivas que buscan los fallos más comunes y tediosos en la creación de código[9], o por grupos de programadores expertos que saben dónde buscar los errores o cómo reproducir casos extremos dentro del código para sacar los fallos. Esto para una empresa dedicada al desarrollo y comercialización de *software*, supone tener en plantilla a profesionales expertos en este campo o tener contratado algún tipo de soporte que facilite esta clase de pruebas[10].

Dichas pruebas se realizarían con herramientas genéricas, no parece fácil encontrar una solución comercial al problema propuesto por la empresa. Se necesitan unos controladores concretos y un entorno no virtualizado para la que la validación del producto sea correcta.

La necesidad de incorporar al laboratorio equipos con alta capacidad de cómputo con *GPUs* potentes, para exprimir al máximo el producto, que principalmente será *software* 3D, hace que los programas comerciales no se adapten a las necesidades que busca la empresa y mantener a un equipo de expertos programadores que dedique gran parte de su jornada laboral a depurar los errores, sea inviable.

Hasta la fecha, hay proyectos *open source* que muestran cómo mediante sistemas de aprovisionamiento, son capaces de inyectar un S.O. a otro PC o máquina virtual a través de un PC máster[11], este tipo de tecnología cuenta con varios *software* capaces de realizar este trabajo (*SpaceWalk*[12], *xCAT*[13], *Cobbler*...). Sin embargo, la creación de un laboratorio de pruebas, que trate con distintos sistemas operativos para mediante la gestión y configuración de un PC máster, ser capaces de tener en paralelo testando varios proyectos a la vez, es un hecho del que no hemos encontrado referencias.

2.1 Propuesta

Visto que no hay un producto similar a lo que proponemos, puesto que el mercado tiende a otras vertientes, productos profesionales con un alto coste económico, que no se adaptarían a lo que se requiere en este momento.

Nuestro trabajo tratará el diseño, creación y configuración de un laboratorio de pruebas de *Quality Assurance*. Para desarrollar el proyecto en cuestión, se creará una *LAN* específica y separada de la red de trabajo de la empresa. Mediante un *firewall* se permitirán entradas esta *LAN* desde la red de la empresa y conexiones *SSH/VPN* de los usuarios, pero se bloqueará cualquier salida al exterior no deseada por los administradores de la red. Los nodos dispondrán de dos discos duros, uno para la instalación del sistema y otro para las copias del mismo, pero carecerán de S.O. propio, pues se les administrara mediante el sistema de aprovisionamiento *Cobbler*, configurado en el PC máster.

Dentro de la *LAN* de pruebas, necesitaremos dos servidores, uno *DHCP* y otro *DNS*, estos servicios pueden ser gestionados por *Cobbler*, pero por motivos de fiabilidad y seguridad del laboratorio los administraremos desde el rúter *Mikrotik RB2011IL-RM*, encargado de gestionar la red de pruebas.

Aunando todos los componentes del laboratorio en cuestión, obtendremos un sistema capaz de servir sistemas operativos a los nodos con las distribuciones, versiones y controladores correctos para cada prueba, dependiendo del código a testar. Todo esto se realizará desde el PC máster, facilitando la instalación y configuración de los mismos mediante el sistema de aprovisionamiento *Cobbler*.

CAPÍTULO 3

Análisis del problema

Cobbler es un servidor de instalación que recopila los distintos métodos para realizar instalaciones de sistema desatendidas, ya sea en servidor, estación de trabajo o sistemas de huéspedes en una instalación completa o virtualizada. Permite a los administradores centralizar la instalación del sistema y la infraestructura de forma automática. Cuenta con varias herramientas para la preinstalación, administración de archivos *kickstart*, administración del entorno de instalación, etc.

Para utilizar *Cobbler* como un servidor de arranque *PXE*, debemos tener configurado en el propio sistema o bien en el rúter de nuestra red, un servidor *DHCP*, indicándole la dirección *IP* de nuestro servidor *Cobbler*. De esta manera, este se encargará de automatizar el arranque remoto por red de los PCs administrados.

Cabe destacar que este proyecto emplea el sistema de aprovisionamiento *Cobbler*, y depende de las versiones y actualizaciones que la comunidad de desarrolladores vaya realizando sobre este sistema (actualmente se utiliza la versión 3.3.2). Esta posibilidad puede ralentizar o alterar el funcionamiento de nuestro laboratorio en un lapso de tiempo indefinido, hasta haber efectuado el mantenimiento y las modificaciones correspondientes.

En versiones anteriores el sistema de aprovisionamiento *Cobbler* contaba con una interfaz *web* para realizar el lanzamiento de las instalaciones. En la versión actual esta opción no está disponible y en las próximas actualizaciones del sistema no se espera que se vuelva a implantar. Por este motivo y por la facilidad de uso que supone de cara a un usuario medio una interfaz gráfica, hemos decidido crear una para nuestro laboratorio de pruebas.

3.1 Especificación de requisitos

Para este proyecto Arisnova S.L. ha trazado un camino a seguir con distintos requisitos mínimos que se deben cumplir, para que el laboratorio cuente con la funcionalidad esperada de cara a la usabilidad interna de sus empleados. Estos son los mencionados a continuación:

- Separar la red del laboratorio de la *LAN* corporativa de la empresa, para evitar trafico adicional generado por el propio laboratorio, que pudieran alterar el flujo de trabajo habitual de la misma;
- Evitar intrusiones en la red del laboratorio ajenas a trabajadores de la empresa;
- Necesidad de implementar un sistema de acceso remoto para facilitar el teletrabajo;
- En la medida de lo posible, separar los distintos componentes y servicios, para no tener un único punto de fallo dentro de un PC, evitar dependencias excesivas de cara a modificar o reemplazar los componentes del laboratorio;
- Posibilidad de generar copias de seguridad del S.O. lanzado a un nodo, para su posterior reutilización.

3.2 Análisis de seguridad

Para que *Cobbler* no dé ningún tipo de problema en su funcionamiento y configuración, debemos indicar al sistema operativo anfitrión, que modifique el módulo de seguridad *Security-Enhanced Linux* (*SELinux*) a un estado permisivo (avisa, pero no actúa bloqueando posibles configuraciones necesarias). Esto facilitará las comunicaciones y configuraciones del propio sistema de aprovisionamiento *Cobbler* hacia sí mismo y los nodos de la red.

También facilita la labor de gestión y manejo, deshabilitar el *firewall* interno del anfitrión, de no ser así. Habría que abrir puertos y habilitar reglas para permitir el funcionamiento óptimo del sistema, este último paso sólo se recomienda si la red de trabajo está bien protegida, y se tiene la certeza de que no hay fallos de seguridad.

Para poder rebajar la seguridad en el nodo máster de esta forma, a priori tan poco aconsejable, nos hemos asegurado de proteger bien la red previamente, mediante la administración del *firewall* del enrutador *Mikrotik RB2011IL-RM*. Administrando la seguridad y los servidores *DHCP* y *DNS* desde el rúter, reducimos la dependencia del laboratorio al correcto funcionamiento del PC máster. Pudiendo reemplazarlo con más facilidad en caso de fallo o avería.

La utilización de los nodos es independiente del máster, donde se encuentran los ficheros de configuración que sólo los administradores del sistema deben tener acceso, pues un cambio mínimo en uno de ellos podría ser fatal para el correcto funcionamiento del laboratorio. A su vez, a los usuarios de los nodos únicamente se les dan los premisos necesarios para poder realizar sus pruebas con éxito, impidiendo que puedan realizar un mal uso del sistema alterando la seguridad de la red o de los propios equipos.

Los accesos remotos desde fuera de la red del laboratorio, únicamente son permitidos si proceden desde una *IP* de la *LAN* o *VPN* corporativa de la empresa, bloqueando así cualquier incursión no deseada por personas ajenas a la empresa. Del mismo modo, se permiten las conexiones desde dentro de la red del laboratorio, en este caso tienen que estar físicamente presentes en él.

3.3 Análisis del marco legal y ético

3.3.1. Análisis de la protección de datos

Los datos almacenados en los archivos *kickstart* (claves *SSH*, contraseñas, usuarios, etc). Respectivos al lanzamiento, instalación y configuración de los sistemas operativos en los nodos del laboratorio, se almacenan en el nodo máster, con acceso únicamente por parte de los administradores del sistema. Las claves de acceso a los nodos cliente, son distintas entre sí, se pueden modificar dependiendo del usuario y distribución a instalar, con intervención del administrador del sistema.

Mencionar, que los sistemas operativos instalados en los equipos del laboratorio, están orientados a pruebas donde no se almacenan datos importantes de usuarios o clientes. Al finalizar el testeo correspondiente al código desarrollado para un proyecto en concreto, ese sistema operativo se elimina completamente, dando lugar a otra instalación de prueba.

Si hay especial interés en mantener una copia de un sistema operativo en concreto, para continuar realizando pruebas, o como base para futuras instalaciones, este se copia a un segundo disco contenido en el nodo cliente, donde se almacena mediante el *software* de copia de imágenes *Clonezilla*. En este caso, para acceder a los datos, habría que estar físicamente conectado al PC, pues hay que modificar el arranque de la *UEFI* del equipo, para poder clonar el sistema copiado al disco de arranque del PC.

Este último apartado relativo a la seguridad, al ser un laboratorio de prueba, orientado al testeo de código de los propios trabajadores de la empresa, no requiere una seguridad física especial, pues el laboratorio se encuentra en el interior de las instalaciones de Arisnova S.L.

3.3.2. Propiedad intelectual

Todas las imágenes relativas al laboratorio de pruebas realizadas en las instalaciones de Arisnova S.L. Ingeniería de Sistemas publicadas en este documento, cuentan con la autorización de los responsables de la misma.

El *software* utilizado en el desarrollo del proyecto cuenta con licencias gratuitas o de código abierto, desarrolladas por la comunidad *open source* o usuarios desarrolladores. Dichos programas son los mencionados a continuación.

- *WinBox*[14]. Es una utilidad creada para la administración de *Mikrotik RouterOS* usando una interfaz gráfica de usuario intuitiva. Se puede descargar gratuitamente desde la *web* del fabricante o desde el propio rúter;
- *Red Hat Enterprise Linux Developer*[15]. Es un programa de desarrollo que ofrece herramientas, *software*, colaboraciones con la comunidad *RHEL*, aprendizaje y formación en sus arquitecturas de forma gratuita, con el fin de formar al usuario en sus productos. A través de esta cuenta hemos descargado y utilizado los sistemas operativos *RHEL 8.5* y *RHEL 7.4 Server*;
- El resto de S.O. almacenados en el repositorio de distribuciones de *Cobbler*, *Alma Linux 8.5*, *Fedora 34 Server*, *Rocky Linux 8.5* y *Ubuntu 20.04*. Son distribuciones *Linux* de código abierto, gratuitas, soportadas por la comunidad *Linux* o por empresas y entidades sin ánimo de lucro;

- *Clonezilla*. Es un *software* libre de recuperación de una imagen creada a partir de un sistema operativo, cuenta con funcionalidades como crear una imagen del sistema;
- *Visual Studio Code*[16]. Es un editor de código fuente *open source* multiplataforma, que permite la instalación de innumerables extensiones.

3.4 Identificación y análisis de soluciones posibles

Tras haber analizado los requisitos proporcionados por la empresa, el proyecto debe desempeñar un papel importante en cuanto a la seguridad y manejo de los componentes, no debe ser bajo ningún concepto un agujero de seguridad dentro de Arisnova S.L. ni tampoco ser excesivamente complejo su funcionamiento, como para que los potenciales usuarios del laboratorio tengan una curva de aprendizaje tal que prefieran descartar su utilización. A su vez, no debe ralentizar ni sobrecargar la red de trabajo habitual, y debería contar con acceso remoto para satisfacer las necesidades del teletrabajo.

3.4.1. Creación de una subred

Para no alterar el funcionamiento habitual de la *LAN* de la empresa, obligatoriamente debemos crear una subred de trabajo dentro de dicha red. Con tal de poder satisfacer este requisito, necesitamos un rúter capaz de otorgar la seguridad suficiente, administración adecuada de los servicios y facilidad de manejo. Para que, en caso de que un empleado de Arisnova S.L. con conocimientos mínimos en arquitectura de sistemas, en un momento dado, sea capaz de gestionar dicha red.

Puesto que ya contábamos con mínimos conocimientos en la administración de *RouterOS* y *WinBox* a causa de experiencias laborales previas. Sumando la casualidad de que la propia empresa también había utilizado puntualmente estos enrutadores en el pasado, no hubo discusión ninguna en la elección del componente destinado al cumplimiento de este requisito. *Mikrotik* cuenta con una amplia gama de dispositivos dedicados a la administración y configuración de redes a nivel profesional[17], este hecho hacía que el componente seleccionado fuese fácil de seleccionar.

En un análisis previo de los productos que actualmente se encuentran en el catálogo de este fabricante, seleccionamos el componente que se adaptaba mejor según sus características y funcionalidades, suponiendo que dicho producto respondiera en un hipotético caso de crecimiento del laboratorio, sin necesidad de sustituirlo por uno más completo.

3.4.2. Elección del sistema de aprovisionamiento

Tras haberlo mencionado anteriormente en el apartado «1.1 Motivación», la elección de *Cobbler* se ha realizado acorde a los conocimientos previos que hemos adquirido al cursar la asignatura de «Configuración y Optimización de Sistemas de Cómputo». Gracias a un análisis de las funcionalidades de este sistema en la actualidad, confirmamos que era totalmente compatible con las expectativas que teníamos acerca de lo que debía realizar dicho sistema de aprovisionamiento.

Otros *software* mencionados en el capítulo «2 Estado de la cuestión» son *SpaceWalk* sin soporte desde hace dos años aproximadamente, quedó descartado al momento. Y *xCAT* muy similar a *Cobbler*, desarrollado por *IBM*. Puesto que *Cobbler* está desarrollado y soportado por la comunidad *Red Hat Enterprise Linux* y nuestro sistema operativo elegido para el PC máster es *RHEL 8.5*, se llegó a la conclusión que el *software* escogido para administrar el sistema de aprovisionamiento de nuestro laboratorio de acceso remoto para procesos de *Quality Assurance Testing* debía ser *Cobbler*.

3.4.3. Elección de los sistemas operativos a instalar

En cuanto a Arisnova S.L. Ingeniería de Sistemas, tiene proyectos en desarrollo con distintos clientes, la empresa se debe ceñir a los sistemas operativos utilizados por estos. De manera que, las distribuciones *software* seleccionadas para este laboratorio estaban bien definidas. La principal distribución en la cual debía basarse nuestro proyecto, era *Red Hat Enterprise Linux* (*RHEL*) con unas versiones concretas en las cuales profundizaremos en el capítulo siguiente.

Asimismo, puesto que la mayoría de empresas con cierta relevancia, suelen trabajar o desarrollar sus aplicaciones en distribuciones *open source*, interesaba añadir S.O. *Linux* al repositorio que íbamos a manejar. Estas son *Ubuntu Linux*, *Fedora Linux*, *Alma Linux* y *Rocky Linux*. Las dos últimas distribuciones son un añadido interesante, pues están actualmente creciendo en el mercado, en un futuro cercano podrían ser de gran relevancia.

Sin olvidarnos de *Microsoft Windows*, un sistema operativo de gran importancia para muchas empresas y particulares. Por motivos internos de la empresa, interesa tenerlo en el repositorio de *Cobbler* pero a medio plazo, ya que el laboratorio se va a empezar a utilizar en proyectos con clientes que poseen sistemas *Linux*, tanto *RHEL* como *Ubuntu*. La configuración para incorporar dicho sistema es parcialmente distinta a los anteriores, y merece una investigación aparte, fuera de los límites y objetivos de este trabajo.

3.4.4. Implementación de un sistema de copia del entorno de ejecución

Tras una ardua investigación y testeo de *software* con aparentemente buenas perspectivas de éxito, no conseguimos por ahora, generar un archivo *ISO* con la estructura adecuada para que al importar dicho archivo a *Cobbler*, este lo reconozca como una distribución válida y funcional la cual añadir al repositorio existente.

Entre los programas que hemos probado sin éxito, se encuentran:

- *Brasero* junto con *Systemback*[18];
- La utilidad de línea de comandos *dd*[19];
- La utilidad de línea de comandos *mkisofs*[20];
- La generación de *ISOs* de *Clonezilla*[21].

Finalmente hemos decidido virar el rumbo establecido, hasta conseguir generar un archivo *ISO* de un sistema operativo en funcionamiento con la estructura adecuada que pueda ser validado por *Cobbler*. El nuevo rumbo pasa por la gestión del sistema de recuperación de imágenes *Clonezilla*, otra utilidad distinta de este programa ya utilizado en la empresa con la certeza de su correcto funcionamiento. Generando una copia exacta del S.O. modificado y almacenándola en el segundo disco, lista para ser clonada al disco principal.

3.4.5. Creación de una interfaz *web* simple y funcional

Para la creación de la interfaz *web*, tras consultar a la empresa las opciones más viables para este cometido, acordamos utilizar *Visual Studio Code* como entorno de programación junto con la extensión *IntelliCode*[22] para el desarrollo *web*, como esta se compone de dos partes claramente diferenciadas:

- Para la parte *front-end* utilizaremos las extensiones del gestor de paquetes *npm*[23] y la biblioteca de código abierto para crear interfaces de usuario *React*[24];
- Para la parte *back-end* utilizaremos el lenguaje de programación *GO*[25].

CAPÍTULO 4

Solución propuesta

Una vez identificados los elementos del laboratorio de acceso remoto multiplataforma para procesos de *Quality Assurance* (*QA*) *Testing*, en este capítulo desarrollaremos la solución escogida en cada componente y su posterior implantación y validación en el conjunto del proyecto.

Empezando por el montaje del laboratorio, que consta de dos PCs, un *HP Compaq dc7800 Convertible Minitower PC*, *GPU NVIDIA GeForce GT 240*, *CPU Intel Core i5*, *16GB* de memoria *RAM* y un disco *SATA* de *1TB*, que actuará como máster donde irá instalado y configurado nuestro sistema de aprovisionamiento.

El otro PC es un *Clon*, contiene una placa base *ASUS CG410PLM Rev:2.0*, *GPU NVIDIA GeForce GTX 1060 6GB*, *CPU Intel Core i5*, *16GB* de memoria *RAM* y dos discos duros *SATA*, con suficiente potencia de cómputo para cumplir su cometido. Actuará de nodo cliente, al cual se le instalarán mediante *DHCP-PXE* los distintos sistemas operativos.

Figura 4.1: Componentes del laboratorio de pruebas. (Elaboración propia, 2022)

Además, el laboratorio se compone de un rúter *Mikrotik RB2011IL-RM* capaz de administrar y gestionar la red que vamos a crear, y un monitor para visualizar el trabajo que vamos realizando.

La parte *software* de este proyecto, constará de la instalación de *RHEL 8.5 Server* en el PC máster, una vez instalado y actualizado el sistema procederemos a la instalación y configuración de *Cobbler*. Tras este paso, importaremos al repositorio de nuestro sistema de aprovisionamiento los distintos sistemas operativos a instalar en el PC cliente:

- *Alma Linux 8.5*;
- *Fedora 34 Server* (sin interfaz de usuario);
- *RHEL 7.4 Server*;
- *RHEL 8.5*;
- *Rocky Linux 8.5*;
- *Ubuntu Linux 20.04*.

Como podemos observar, las versiones *Server* de los S.O. *Fedora*, no cuentan con interfaz gráfica, este se ha escogido por el hecho de tener mayor variedad en el repositorio de cara a posibles pruebas de proyectos futuros.

Cobbler no admite la importación de sistemas operativos lanzados recientemente al mercado, los desarrolladores del proyecto dejan pasar un tiempo antes de permitir importarlo, esto se hace a través de un sistema de firmas. Por este motivo, la última versión válida de *Fedora* es la *34*, de los sistemas basados en *RHEL* es la *8.X* y de *Ubuntu* la *20.X*.

Sumado a todo lo anterior, también necesitaremos el *software Clonezilla* para realizar la copia y el almacenado de los sistemas operativos, permitiendo conservar las modificaciones e instalaciones interesantes de proyectos en curso.

La interfaz *web* la realizaremos con el editor de código fuente *Visual Studio Code* y las extensiones *IntelliCode*, *npm*, *React* y *GO*. Como hemos explicado en el subapartado anterior «3.4.5 Creación de una interfaz *web* simple y funcional».

4.1 Plan de trabajo

El proyecto será llevado a cabo por una sola persona, esto implica la implantación del método cascada o secuencial, donde se adquiere un proceso lineal en la gestión de este, las fases no se inician hasta haber finalizado la inmediatamente anterior a esta. Realizando una división de fases inicial, obtenemos una aproximación del plan de trabajo:

- Primeramente recopilaremos la información necesaria y estableceremos los requisitos para hallar un punto de partida;
- Seguido de un diseño completo del sistema, analizando los componentes necesarios;
- A continuación, implementaremos el laboratorio configurando sus distintos componentes;
- Tras este último paso, realizaremos la fase de pruebas del proyecto, modificando y solucionando los problemas que vayan surgiendo;
- Finalmente, validaremos el proyecto con respecto a los objetivos marcados.

Las limitaciones de las distintas fases del proyecto, impiden gestionar en paralelo dos o más de ellas. Pues, hasta no tener creada la subred del laboratorio y conectados todos los componentes físicamente entre sí, no se puede empezar con la configuración de los componentes. Y hasta no tener configurado el rúter, PC máster, *Cobbler*, etc. Es inviable comenzar con el lanzamiento de los distintos sistemas operativos al nodo cliente.

Una vez completadas dichas fases del proyecto, iniciaremos la fase de copiar y almacenar los sistemas operativos modificados o con instalaciones de *software* genéricas, con el fin de aprovechar el trabajo realizado y no partir de cero en futuras instalaciones.

Tras esta fase, empezaremos la creación de la interfaz *web*, evitando al usuario encontrarse con un entorno de trabajo sin interfaz gráfica, donde el uso del sistema se haría mediante línea de comandos, otorgando al laboratorio una mayor usabilidad.

Desgranadas las distintas partes, procedemos a realizar una estimación aproximada de horas-persona que necesitaremos invertir en cada fase del proyecto, computando las jornadas de trabajo en cinco horas, debido al convenio de prácticas firmado con anterioridad.

Fase del proyecto	Jornadas	Horas-Persona
Recopilación de información y requisitos mínimos	6	30
Diseño del sistema y componentes necesarios	5	25
Configuración de los componentes del laboratorio	20	100
Pruebas y solución de problemas	10	50
Validación del proyecto	3	15
TOTAL	44	220

Como vemos en la tabla anterior, la estimación sugiere una inversión total a 220 horas-persona, el equivalente a 44 jornadas de trabajo en Arisnova S.L. Asumiendo estos datos como orientativos, el proyecto consta de un plazo de entre ocho y nueve semanas desde el comienzo hasta la finalización del mismo.

Somos conscientes que al tratarse de una estimación, el plazo del proyecto puede o no ceñirse a esta toma de tiempos inicial. Teniendo en cuenta esto, se puede asumir un retraso de entre una a dos semanas de trabajo en la entrega de este.

4.2 Presupuesto

Al ser un proyecto conjunto con Arisnova S.L. Ingeniería de Sistemas, se ha desarrollado en sus instalaciones y nos ha cedido gran parte de los componentes del laboratorio. Estos no han tenido coste alguno para nosotros, el único componente que hemos adquirido para llevar a cabo el desarrollo del proyecto ha sido el rúter *Mikrotik RB2011IL-RM* con un coste de 98'85€, del cual se ha hecho cargo la propia empresa.

Para la creación de un laboratorio de semejantes características serían necesarios los siguientes componentes electrónicos, además del rúter anteriormente mencionado.

Componente	Cantidad	Coste
Rúter *Mikrotik RB2011IL-RM*[26]	1	98'85€
PcCom Gold Intel Core i7[27]	1	1834'25€
PcCom Bronze SP Intel Core i5[28]	1	622'35€
Cable de Red *Ethernet Cat.6a-1m*[29]	4	5'25€
Monitor *Samsung LF24T350FHRXEN FullHD 24"*[30]	1	149€
Disco Duro *WD Blue SSD SATA 1TB 2.5"*[31]	1	111'65€
SUBTOTAL		2837'10€

Sumando los costes de todos los componentes sugeridos para la creación del laboratorio de pruebas, obtenemos un total de 2837'10€. A todo lo anterior debemos sumarle las horas-persona que hemos necesitado para el proyecto, computando también las jornadas de trabajo (cinco horas) equivalentes. Hemos realizado una aproximación del tiempo empleado para cada parte del mismo.

Fase del proyecto	Jornadas	Horas-Persona
Bocetado y estimación de componentes necesarios	0.4	2
Investigación, montaje y configuración del laboratorio	25.2	126
Pruebas de funcionamiento y solución de errores	7.2	36
Validación de pruebas y correcto funcionamiento	3.6	18
Instalación del sistema de copias de seguridad	1.2	6
Creación de una interfaz *web* simple	8	40
Incorporación de la interfaz *web* al proyecto	1.6	8
TOTAL	47,2	236

Consultando el convenio de 2020 de un Ingeniero titulado en el BOE[32], nos salen 1792 horas con un total anual de 14 pagas más el plus por convenio, el total anual es de 26323'57€. Dividiendo el total anual entre las horas totales trabajadas al año, obtenemos 14'69€/h que cobraría un ingeniero titulado por convenio en el año 2020. Multiplicando esto por las horas-persona obtenidas anteriormente.

Horas-Persona	Coste
236	14'69€
SUBTOTAL	3466'84€

Finalmente, queremos añadir que la utilización del *software* necesario para el desarrollo del proyecto no ha tenido coste alguno, los sistemas operativos utilizados (*Alma Linux*, *Fedora*, *Red Hat Enterprise Linux*, *Rocky Linux* y *Ubuntu*), *WinBox* (*Mikrotik*), *Cobbler*, *Clonezilla* y el desarrollo de la interfaz *web* realizado en *Visual Studio Code*, son programas de código abierto o con licencias de desarrollador, por los cuales no hay que pagar una licencia por su uso.

Una vez obtenido el importe de cada componente del proyecto, falta obtener el coste total de la creación del laboratorio.

Componente	Cantidad	Coste
Hardware	1	2837'10€
Software	1	0€
Personal	1	3466'84€
TOTAL		6303'94€

El presupuesto total del proyecto asciende a 6303'94€, teniendo en cuenta que únicamente hemos replicado el coste actual del laboratorio, con un sólo PC cliente. Si quisiéramos aumentar el número de nodos disponibles en nuestro laboratorio, idealmente de dos a cuatro PCs, habría que obtener el mismo número de cables de red *Ethernet* y de discos duros *SSD* que PCs quisiéramos adquirir.

4.3 Diseño de la solución

A continuación, vamos a comentar la división en cuatro bloques del proyecto desarrollado, a partir del apartado «1.2 Objetivos» en el cual ya quedaba potencialmente clara una separación lógica, para tratar de finalizar con éxito la creación del laboratorio de acceso remoto multiplataforma para procesos de *Quality Assurance* (*QA*) *Testing*. Tras este subapartado, trataremos más detalladamente los aspectos relevantes de las distintas configuraciones que han hecho posible el correcto funcionamiento del proyecto.

4.3.1. Arquitectura del sistema

El proyecto se divide en cuatro bloques:

- El primer bloque, corresponde a la creación y configuración de una subred separada de la *LAN* corporativa de la empresa. Esta red está manejada por el rúter *Mikrotik RB2011IL-RM*, habilitando el *firewall* generamos las reglas para otorgarle seguridad a la red. Configuramos el rúter como servidor *DHCP* y servidor *DNS*. Cada componente de la red del laboratorio obtiene una *IP* estática, escogida dependiendo del componente del laboratorio que sea, dentro de un rango determinado. Desde el enrutador indicamos a los nodos de la red que el servidor *PXE* es el PC máster, para iniciar un arranque por red a través del archivo *pxelinux.0*.

- El segundo bloque, corresponde a la configuración del sistema de aprovisionamiento *Cobbler* en el PC máster, con detección de los nodos a traveś de la red y lanzamiento del sistema operativo seleccionado por el usuario, instalándolo mediante la *LAN* del laboratorio, importando un conjunto de distribuciones en el almacenamiento local del nodo máster, administradas por el sistema de aprovisionamiento utilizado;

- El tercer bloque, corresponde a la implantación de un mecanismo para generar una copia del entorno de ejecución del nodo en funcionamiento por medio del *software Clonezilla*, incluyendo sistema operativo, aplicaciones, controladores, contenido del disco local, etc. El archivo generado se almacenará en un segundo disco dentro del nodo cliente. A su vez, se creará un documento con los paquetes y controladores añadidos a la distribución base y se guardará en el interior de la copia generada;

- El cuarto y último bloque, corresponde a la realización de una interfaz *web* simple, tras autenticarse como usuario del sistema, se seleccionará el nodo y la distribución *Linux* a instalar, dependiendo del proyecto a testar. Esto facilitará enormemente la interacción de un usuario medio con el laboratorio de pruebas.

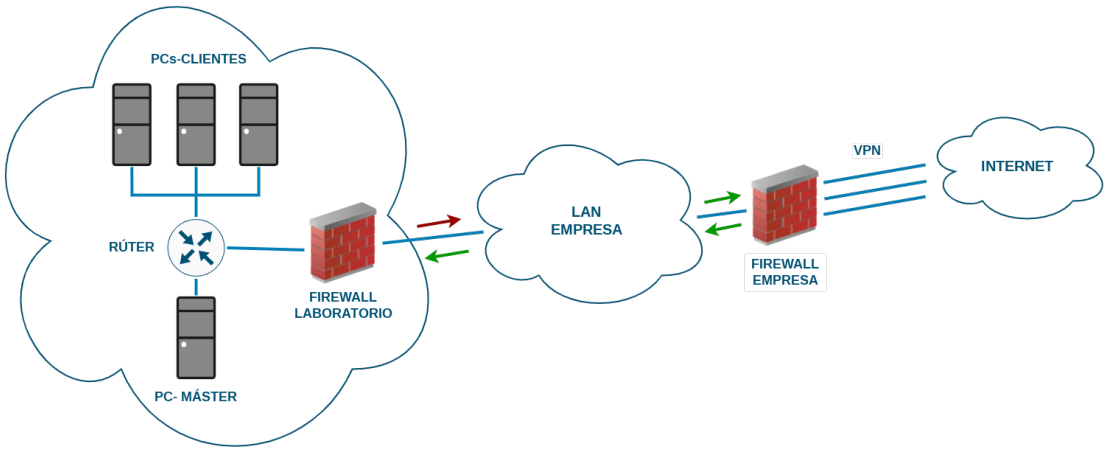

Figura 4.2: Arquitectura del laboratorio. (Elaboración propia, 2022)

Como podemos observar en el diagrama anterior, se muestra la configuración del laboratorio de pruebas, a traveś de la *LAN* de la empresa tendremos acceso a nuestra subred. Si nos encontramos en las instalaciones de Arisnova S.L. no hará falta utilizar la conexión *VPN*. Si por el contrario, nos encontramos realizando teletrabajo, necesitaremos emplear el túnel *VPN* para a través de la red de la empresa poder conectarnos al laboratorio de pruebas.

El boceto se creó inicialmente con tres PCs cliente, el desarrollo del proyecto únicamente se ha realizado con uno, se espera poder ampliar los nodos cliente a un número de entre dos y cuatro PCs próximamente.

4.3.2. Diseño detallado

El único diseño realizado en este laboratorio de pruebas ha sido el desarrollo de la interfaz *web*, el resto de bloques en los que se divide el proyecto han sido instalaciones y configuraciones de dispositivos o *software* existentes.

En esta sección se va a detallar el diseño que presenta la interfaz *web*, los componentes en los que se divide y la interacción que tiene con el sistema de aprovisionamiento, concediendo la comunicación entre el usuario y el sistema de forma visual. Permitiendo elegir la distribución que lanzaremos al nodo cliente y conectarnos remotamente una vez se haya completado dicha instalación.

El menú de *loggeo* de la interfaz *web*, se compone de dos campos a rellenar por el usuario, *Username* y *Password*, se rellenan con las credenciales de acceso del nodo máster. El título hace referencia a *Next Generation Laboratory*, al ser un proyecto desarrollado con la participación de la empresa, es la nomenclatura interna que se ha escogido para este.

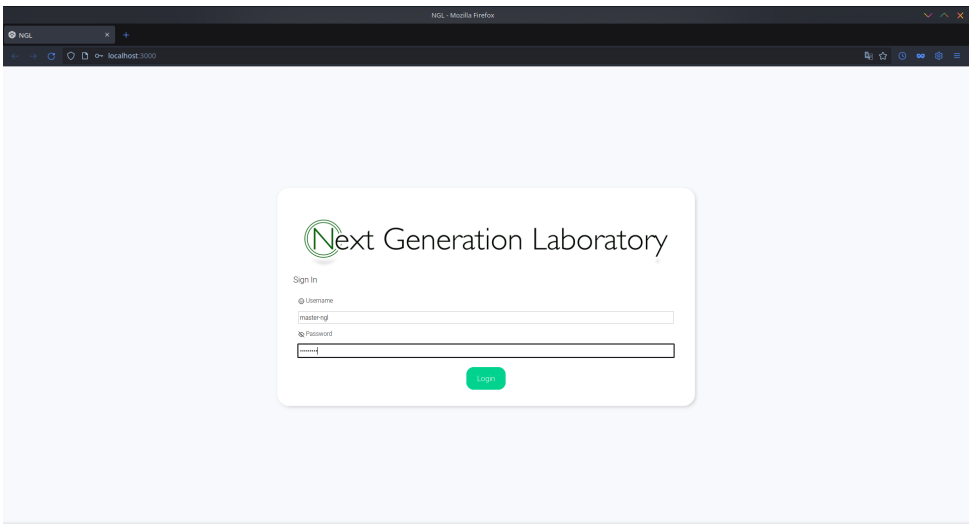

Figura 4.3: Interfaz credenciales *web*. (Elaboración propia, 2022)

Una vez hemos accedido a la página principal de la *web*, observamos una interfaz limpia y simple. Se compone de dos campos principales, la selección de desplegables a la izquierda, y la visualización de nodos a la derecha. De momento sólo hay un nodo visualizado, el único del que disponemos actualmente. Hay espacio para ampliar hasta cuatro tarjetas por fila, con posibilidad de generar más filas.

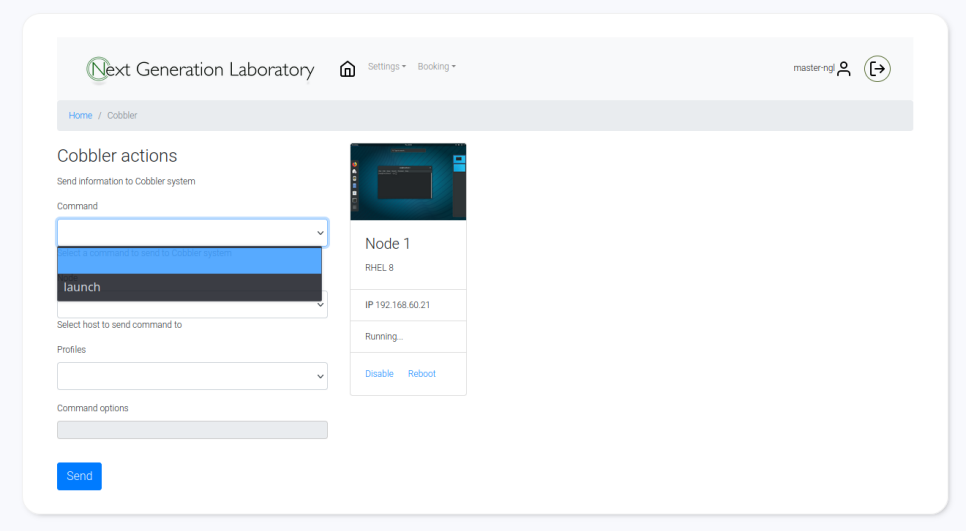

Figura 4.4: Interfaz principal *Command*. (Elaboración propia, 2022)

En la imagen anterior podemos observar en el desplegable *Command* una única opción *launch* relativa a la acción que deseamos realizar. Momentáneamente no hay más opciones disponibles, este campo requiere estar relleno para poder enviar la petición al nodo máster. Está creado con perspectivas de futuro para implementar distintas funciones, por ejemplo, *remove*, que podría borrar el sistema instalado en cierto nodo. Actualmente no tiene interación con *Cobbler*.

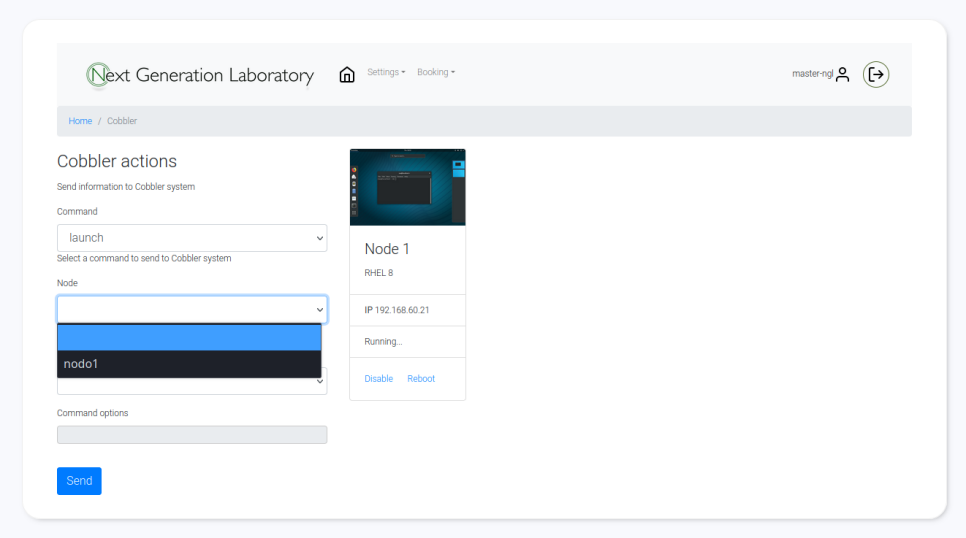

Figura 4.5: Interfaz principal *Node*. (Elaboración propia, 2022)

El siguiente desplegable hace referencia a los PCs clientes disponibles, de momento sólo tenemos el *nodo1*. La elección de este campo renombraría dependiendo del nodo elegido, el nombre del sistema *Cobbler* a crear.

```
cobbler system add --name="nodo1" --profile="ALMA8-Nodo1" --interface="enp4s0"
--mac="E0:D5:5E:F9:30:E3" --netboot-enabled="Y"
```

Reemplazando el valor del *flag –name* por el nodo seleccionado, de momento es fijo el nombre por la sencilla razón que sólo contamos con un nodo cliente.

En la siguiente imagen, visualizamos en el desplegable los distintos perfiles de los que dispone el repositorio *Cobbler*. Una mejora a realizar es el filtrado de perfiles para que únicamente aparezcan los terminados en -Nodo1, pues son los generados por nosotros y enlazados con los archivos *kickstart* de arranque.

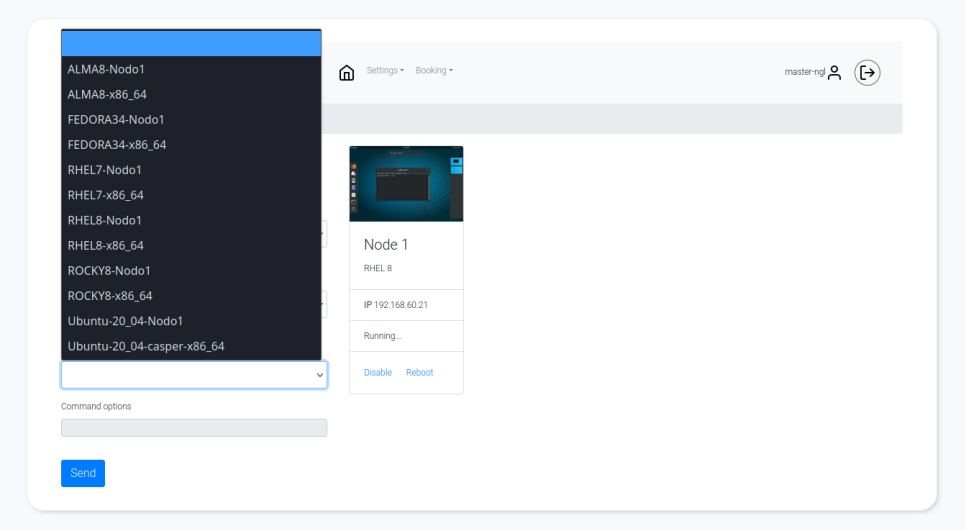

Figura 4.6: Interfaz principal *Profiles*. (Elaboración propia, 2022)

En la visualización de los nodos, observamos la dirección *IP* del PC, el nombre de la distribución que actualmente está en funcionamiento y su estado (*Running...*). Con la opción de apagarlo (*Disable*) y reiniciarlo (*Reboot*). Esta visualización es una simulación, pues no está implementada todavía, se pretende incorporar más adelante.

Una vez seleccionados el comando, el perfil y el nodo, pulsamos el botón de *Send* y automáticamente el máster recibe las instrucciones siguientes para empezar la instalación en el nodo cliente.

```
cobbler system remove --name=nodo1
cobbler system add --name="nodo1" --profile="ROCKY8-Nodo1" --interface="enp4s0"
--mac="E0:D5:5E:F9:30:E3" --netboot-enabled="Y"
cobbler sync
```

El ejemplo anterior se ha realizado con un el perfil de *Rocky Linux 8.5*, este perfil sería reemplazado por el seleccionado en el desplegable correspondiente. Al reiniciar el PC cliente, comenzaría la instalación seleccionada.

En la parte superior central, podemos observar dos desplegables, *Settings* con la única opción de «Actualizar mi información» y *Booking* que se divide en *Booking node* y *Booking laboratory*. No están implementadas actualmente estas opciones, pero se pretende actualizar el estado de las máquinas una vez estén en funcionamiento y tener la posibilidad de generar un sistema de reservas de un nodo concreto o el laboratorio al completo por parte de los usuarios del mismo.

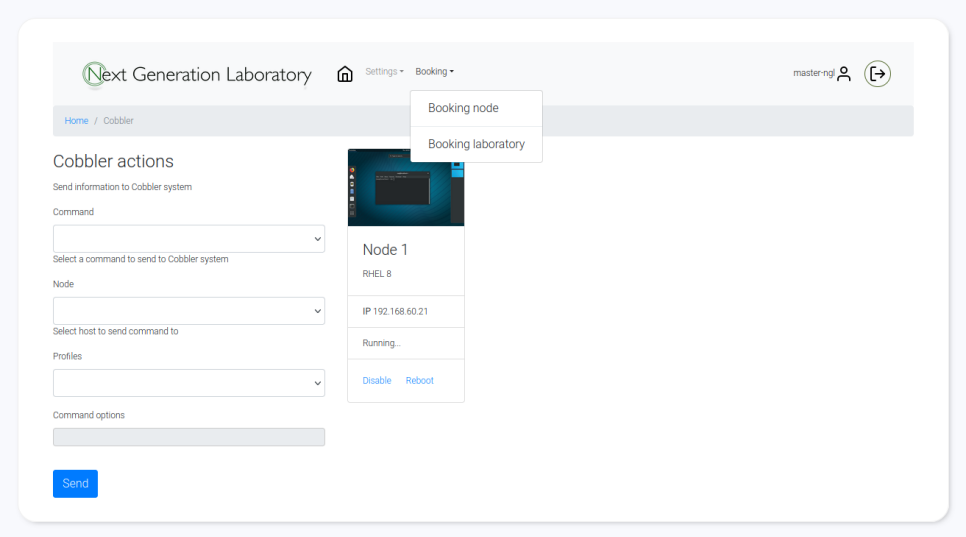

Figura 4.7: Estructura interfaz *web back-end*. (Elaboración propia, 2022)

La interfaz *web* se compone de dos partes fundamentales para su funcionamiento, la parte *back-end* programada con *GO*. En la imagen siguiente, podemos apreciar la estructura de la que se constituye esta sección, conjuntamente con el método *main.go*.

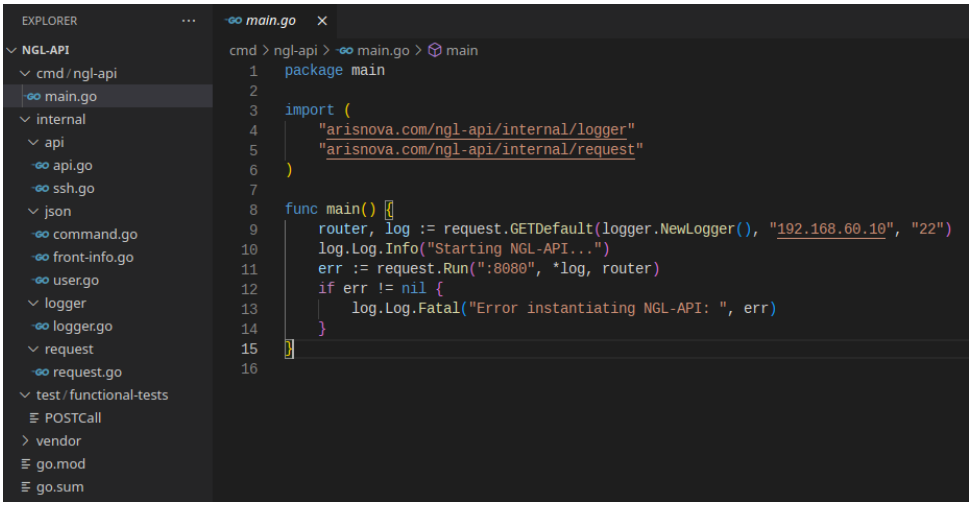

Figura 4.8: Estructura interfaz *web back-end*. (Elaboración propia, 2022)

La parte *front-end* construida mediante *npm* gestor de pauqetes de *Node.js* herramienta de *JavaScript* y la biblioteca para crear interfaces de usuario *React*. En la imagen posterior se visualiza la estructura utilizada en esta sección de la interfaz *web*, con un fragmento del archivo *index.html*.

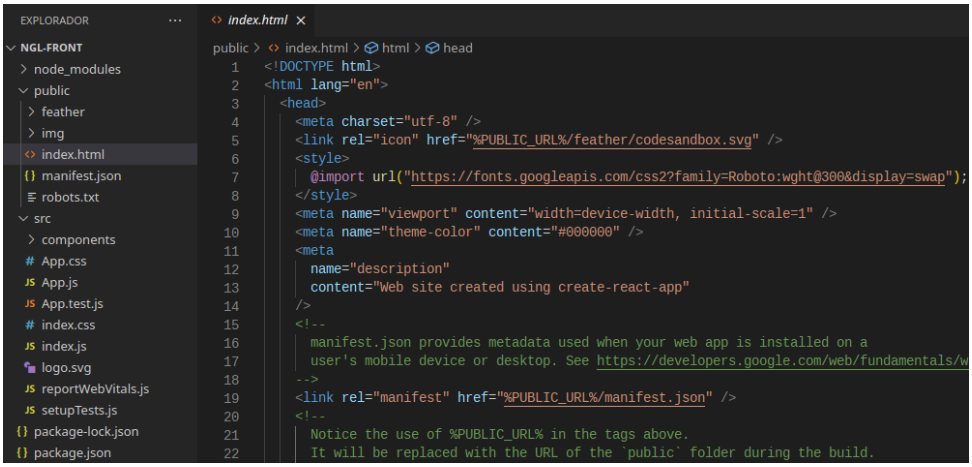

Figura 4.9: Estructura interfaz *web front-end*. (Elaboración propia, 2022)

En el en el Apéndice B «Código de la interfaz *web*», encontraremos los distintos archivos de código generados para dar forma a esta interfaz gráfica.

El siguiente diagrama representa la interacción de la interfaz con sus distintos componentes y el propio laboratorio para dar el servicio deseado a los usuarios de nuestro proyecto.

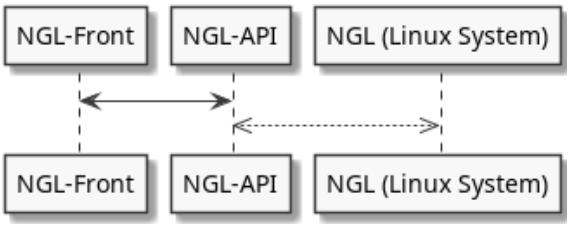

Figura 4.10: Diagrama componentes interacción *web*. (Elaboración propia, 2022)

4.4 Tecnología utilizada

En este apartado vamos a comentar las distintas tecnologías de las que se compone el laboratorio, herramientas, sistemas operativos, sistema de aprovisionamiento, *software* de recuperación de imagen, lenguajes de programación *web*, etc.

Gracias al acoplamiento y configuración de estas tecnologías ha sido posible llevar a cabo el desarrollo del proyecto, permitiendo una utilización sencilla de cara a un usuario final.

4.4.1. *RouterOS WinBox*

MilkroTik RouterOS es un sistema operativo basado en el *kernel* de *Linux* usado en los *Mikrotik RouterBOARD*. Puede instalarse en un PC, convirtiéndolo en un enrutador con todas las características necesarias: *firewall*, *routing*, *Wireless Access Point*, servidor *VPN*, *DHCP*, *DNS*, etc.

RouterOS puede configurarse a través de línea de comandos y es accesible por puerto serie, *telnet*, *SSH* y a través de la interfaz *web WebFig*, también por una aplicación *software* basada en *Microsoft Windows* (*WinBox*) y aplicaciones para *iOS* y *Android*.

WinBox es una herramienta que permite la administración de *MikroTik RouterOS* mediante una interfaz de usuario, las funcionalidades de *WinBox* son muy similares a las de consola. Algunas configuraciones avanzadas o críticas del sistema sólo se pueden acceder desde línea de comandos. Nosotros hemos configurado el rúter del laboratorio completamente desde esta utilidad, una vez configurada hemos dado acceso vía *web* a la administración del laboratorio mediante *WebFig*.

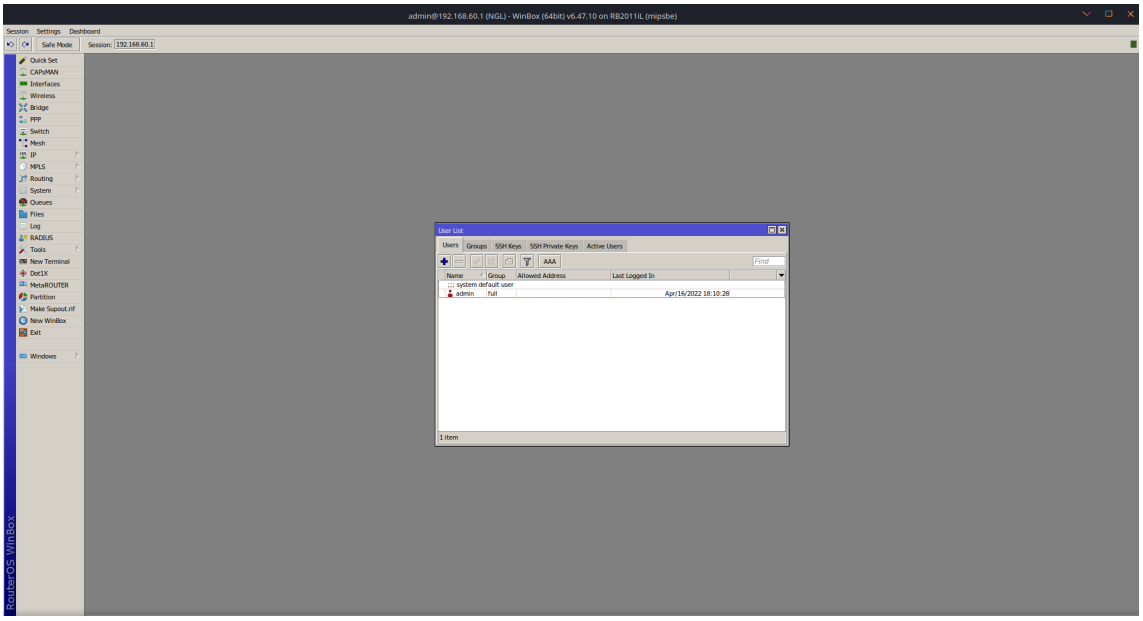

Figura 4.11: Interfaz de usuario *WinBox*. (Elaboración propia, 2022)

4.4.2. Sistemas operativos

Para realizar la instalación de *Cobbler* en el PC máster, este requería previamente la instalación de un sistema operativo *Linux*, puesto que la empresa tiene especial interés en desarrollar sus aplicaciones en *Red Hat Enterprise Linux* por necesidades de sus clientes. Decidimos instalar la versión *Server* de *RHEL 8.5* en el PC máster, sin interfaz de usuario. Como *Cobbler* es un producto gratuito de *RHEL*, nos aseguramos evitar incompatibilidades en la instalación del nodo máster.

La versión *RHEL 8.5* es una versión actual, segura y estable dentro de las distintas que ofrece *Red Hat*, creímos buena opción administrar el laboratorio con esta distribución. Como ya avanzamos en el subapartado «3.3.2 Propiedad intelectual», creamos una cuenta de *Red Hat Enterprise Linux Developer* gratuita que te permite desarrollar y tener soporte de la comunidad.

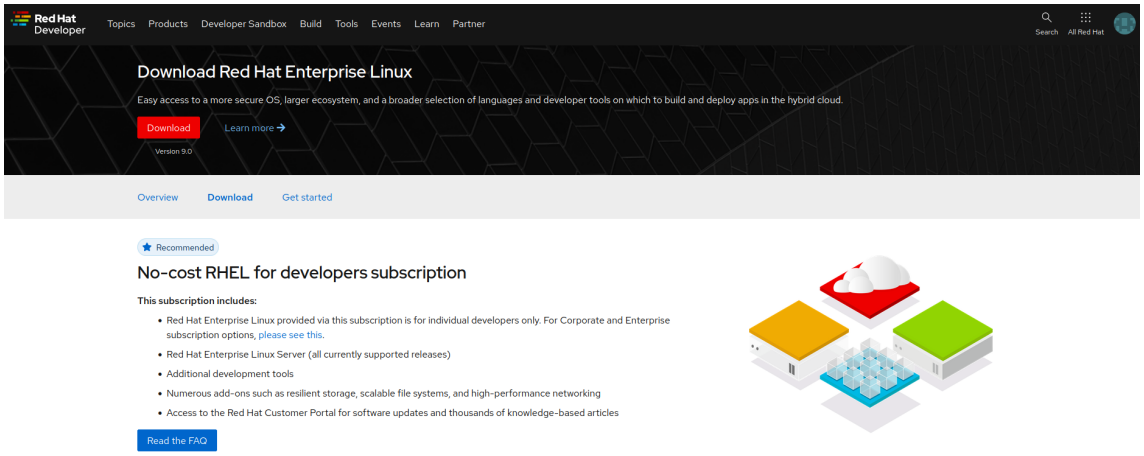

Figura 4.12: Página *web Red Hat Developer*. (developers.redhat.com, 2022)

Utilizando esta cuenta, también descargamos la *ISO RHEL 7.4 Server* para añadir al repositorio de *Cobbler*. Cabe destacar, que la *ISO RHEL 8.5* utilizada en el PC máster es la misma que la importada al repositorio del sistema de aprovisionamiento *Cobbler*, con la diferencia del tipo de instalación elegida durante el proceso.

El resto de distribuciones añadidas al repositorio de nuestro sistema de aprovisionamiento, como hemos mencionado al comienzo de este capítulo son las siguientes: *Alma Linux 8.5*, *Fedora 34 Server* (sin interfaz de usuario), *Rocky Linux 8.5* y *Ubuntu Linux 20.04*.

4.4.3. *Cobbler*

Cobbler es un servidor de aprovisionamiento, instalación y actualización. Admite implementaciones a través de arranque por red *PXE*, virtualización y reinstalaciones de sistemas *Linux* existentes. Las características del servidor de actualización incluyen la duplicación de *yum* y la integración de espejos con archivos de instalación automatizados. *Cobbler* tiene una interfaz de línea de comandos, *WebUI* y *Python* para la integración con *scripts* y aplicaciones externas.

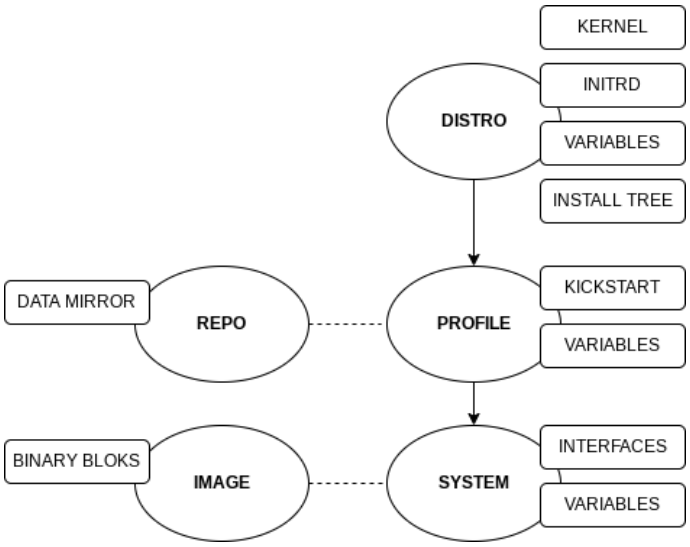

Figura 4.13: Diagrama arquitectura *Cobbler*. (Elaboración propia, 2022)

Cuenta con un repositorio en *GitHub* donde la comunidad ayuda a otros usuarios, sube modificaciones y corrección de errores, comparte discusiones y tienes la posibilidad de abrir incidencias para establecer un seguimiento o ayuda en los distintos problemas que te vayas encontrando con su instalación y configuración. La comunidad está muy activa y es de gran utilidad.

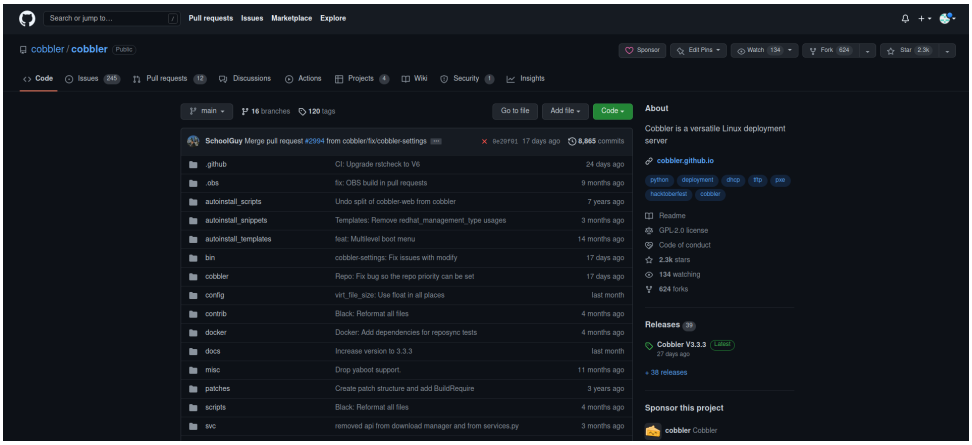

Figura 4.14: Repositorio *Cobbler* en *GitHub*. (github.com/cobbler/cobbler, 2022)

4.4.4. *Clonezilla*

Clonezilla es un programa de creación de imágenes, clonación de particiones y discos similar a otros presentes en el mercado. Sirve de gran ayuda a la hora de realizar implementaciones de sistema, copias de seguridad completas y recuperaciones de imágenes.

Hay tres tipos de *software* disponible, *Clonezilla Live* , *Clonezilla Lite Server* y *Clonezilla Server Edition*. En nuestro caso hemos utilizado *Clonezilla Live*, pues es el adecuado para la copia de seguridad y restauración de una máquina. Guarda y restaura únicamente los bloques utilizados por el disco duro, esto aumenta la eficiencia de la clonación.

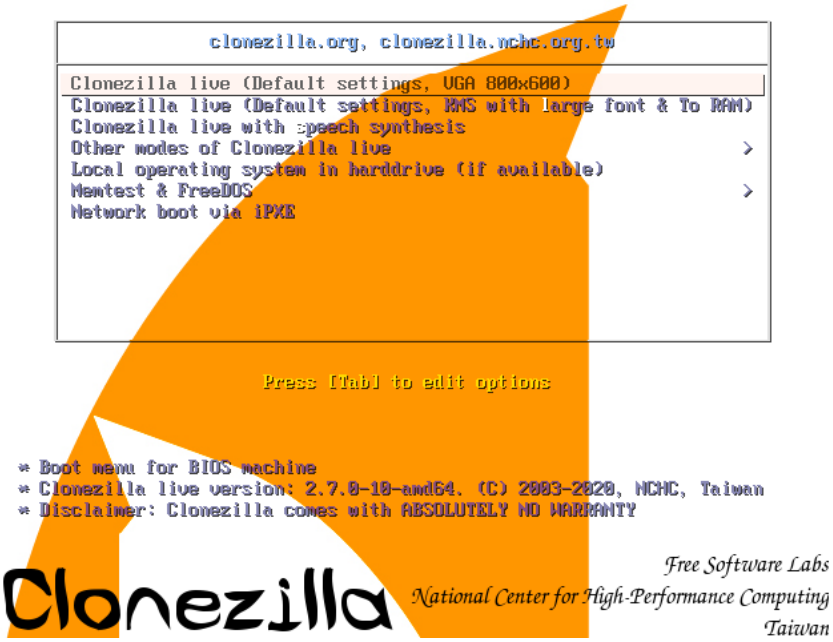

Figura 4.15: Menú de inicio *Clonezilla*. (clonezilla.org, 2022)

4.4.5. *Visual Studio Code*

Visual Studio Code es un editor de código fuente, mencionado en el subapartado «3.4.5 Creación de una interfaz *web* simple y funcional». Lo hemos utilizado para la creación de la interfaz *web* de nuestro laboratorio, conjuntamente con las extensiones *IntelliCode* herramienta de desarrollo *web*. Para la parte *front-end* utilizaremos el gestor de paquetes *npm* y la biblioteca de código abierto para crear interfaces de usuario *React*. Y para la parte *back-end* utilizaremos el lenguaje de programación *GO*.

4.5 Desarrollo de la solución propuesta

En esta sección se va a dar solución a la propuesta que hemos desarrollado para cada parte que conforma el laboratorio de pruebas. Configuraciones que permiten la integración de los componentes del laboratorio, rúter, PC máster, PC cliente, copia del S.O. en ejecución en el nodo cliente e instalación automática de un nodo mediante la interfaz *web*.

4.5.1. Configuración del rúter *Mikrotik*

Necesariamente para la creación de una red de trabajo, es obligatorio el uso de algún dispositivo que ofrezca los servicios de enrutamiento y transferencia de paquetes de red. Este dispositivo perfectamente puede ser un PC previa instalación de *software* que le permita actuar como rúter. En nuestro caso hemos decidido adquirir y configurar el rúter *Mikrotik RB2011IL-RM*.

A continuación, mostraremos paso a paso la configuración que hemos realizado en la utilidad *WinBox* para dotar al laboratorio de una red de trabajo segura y fiable, capaz de interconectar los distintos componentes.

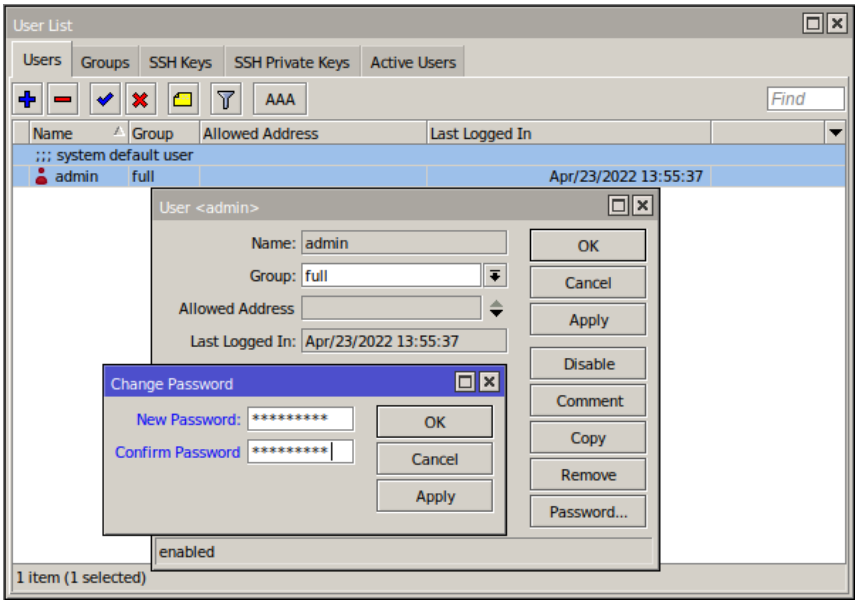

Figura 4.16: Generación de credenciales. (Elaboración propia, 2022)

En la imagen anterior vemos el primer paso a realizar una vez estamos dentro de la utilidad *WinBox*, obligatoriamente debemos generar credenciales de usuario para eliminar las que vienen por defecto, esto es un paso básico de seguridad.

El siguiente paso a realizar, como vemos en la imagen posterior, es actualizar el *software* a la última versión. Este paso es una buena práctica en todos los sistemas, pues evita quedarse obsoleto y reduce los agujeros de seguridad.

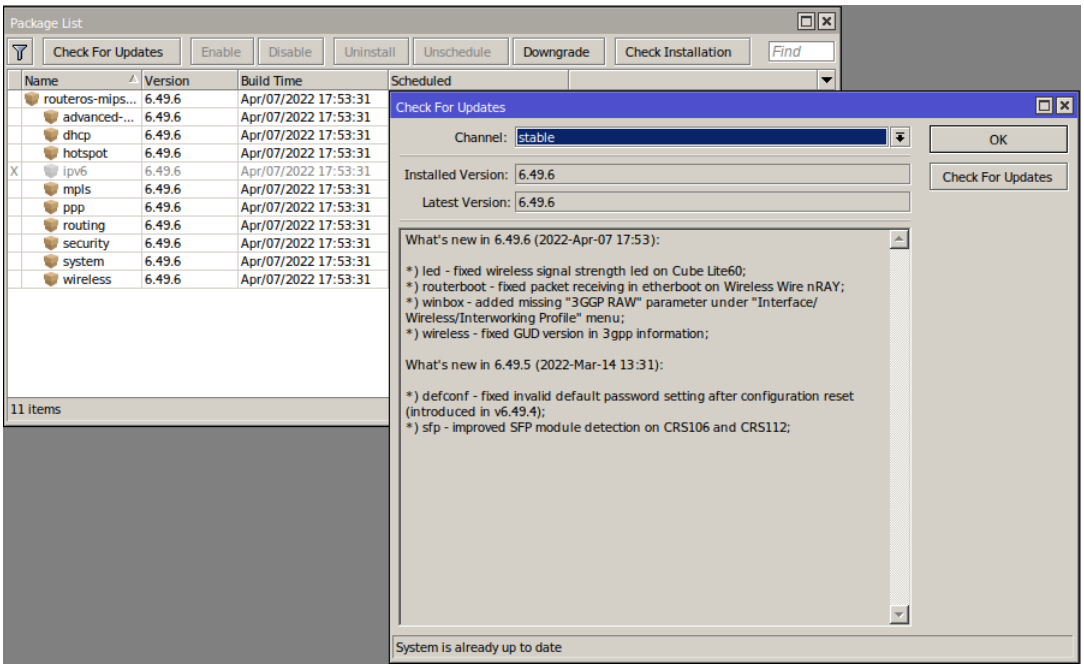

Figura 4.17: Actualización del sistema. (Elaboración propia, 2022)

Una vez realizados los pasos básicos en cualquier software comercial existente. Ya podemos comenzar con la creación de una red de trabajo. Añadimos en *IP->Address* un rango (*/24*) de direcciones *IP* privadas. Nuestra subred del laboratorio es la *IP 192.168.60.0/24*, esto significa que únicamente varía el último octeto del rango. Con esta máscara de subred, tenemos la opción de conectar 254 dispositivos a nuestra red. La *IP 192.168.60.1* pertenece al propio rúter, que actuará como puerta de enlace.

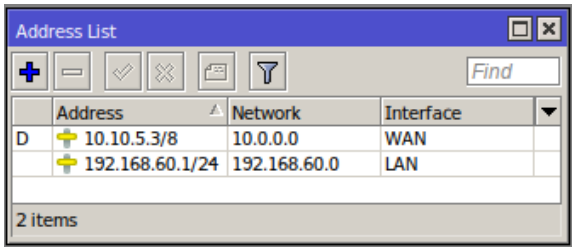

Figura 4.18: Interfaces *IP* del laboratorio. (Elaboración propia, 2022)

Conectamos mediante un cable *Ethernet* el primer puerto del rúter a la red corporativa de la empresa, para que este pueda tener acceso a *Internet* y actuar como un enrutador al uso. Tras este paso, observamos en la imagen anterior como detecta en la *IP* de la empresa como interfaz *WAN*. Ya tenemos conexión a *Internet* y creada nuestra *LAN* de trabajo.

Figura 4.19: Conexiones en el rúter *Mikrotik*. (Elaboración propia, 2022)

Como nos hemos conectado a una red *WAN*, en este caso la propia red de la empresa, automáticamente el rúter crea un cliente *DHCP* para recibir *IP* desde la *WAN*. Este paso es automático, nosotros no hemos intervenido.

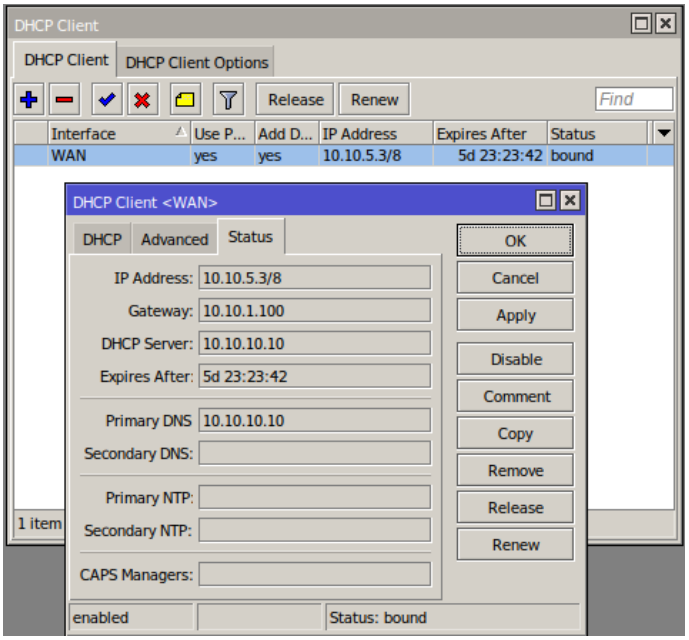

Figura 4.20: Cliente *DHCP* de la *WAN*. (Elaboración propia, 2022)

Acto seguido, creamos nuestro servidor *DHCP*, lo necesitaremos para otorgar a cada dispositivo conectado a la red una dirección *IP* automáticamente. Creamos el servidor en *IP->DHCP->DHCP-Server*.

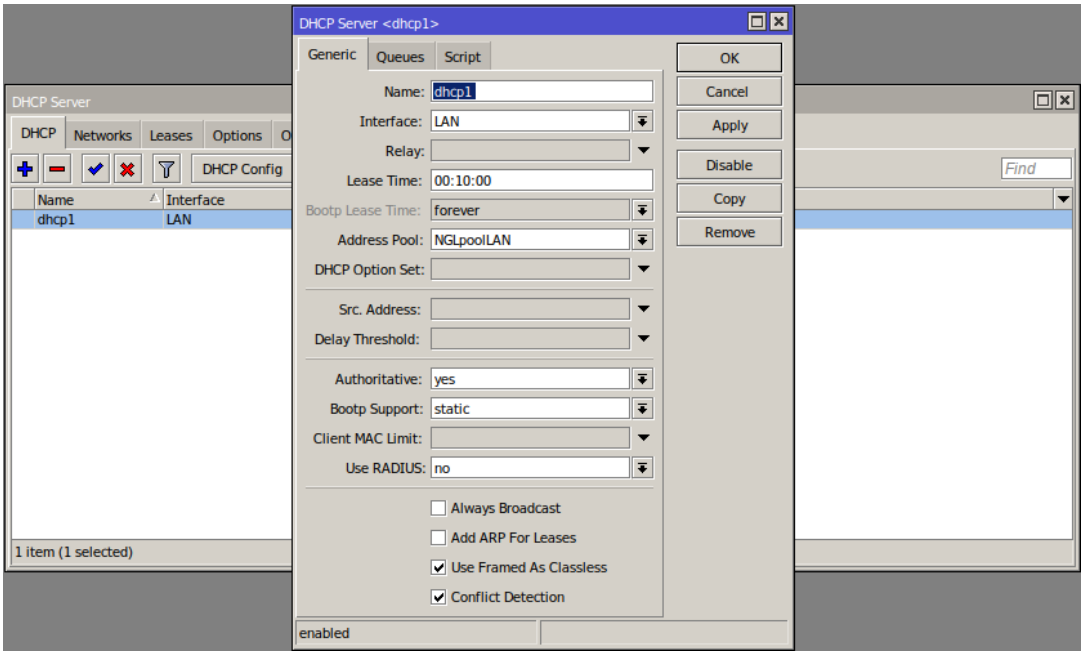

Figura 4.21: Servidor *DHCP* del laboratorio. (Elaboración propia, 2022)

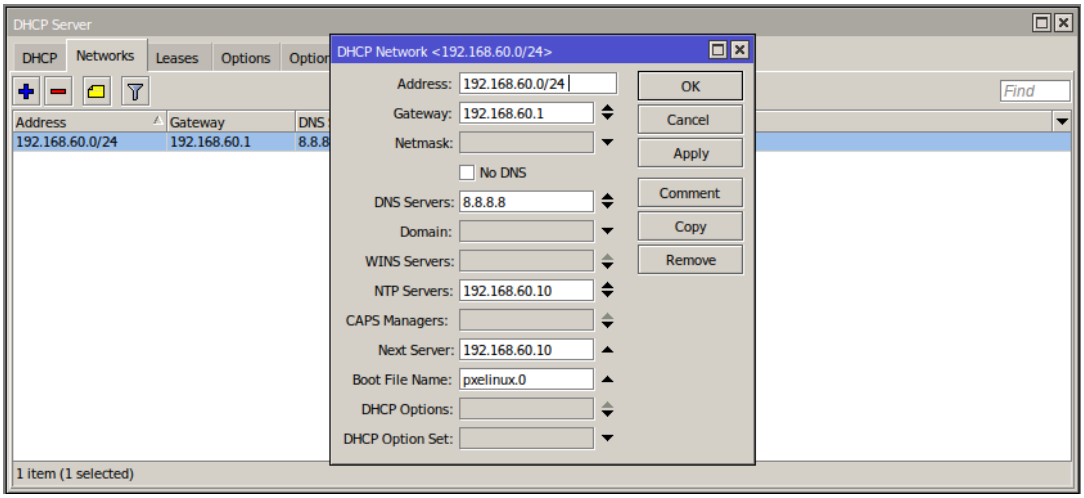

Figura 4.22: Redirección servidor *PXE* al nodo máster. (Elaboración propia, 2022)

Como podemos observar en la imagen anterior, es necesario redireccionar desde el rúter los servicios del servidor *PXE* al PC máster (*Next Server*) mediante la dirección *IP* de este, y el nombre del archivo de arranque (*pxelinux.0*) que los nodos de la red deben pedirle al servidor para iniciar el arranque por red.

Si esta configuración no se realiza correctamente, cuando un nodo cliente entrara en modo arranque por red, finalizaría el plazo de espera predeterminado por el protocolo y cambiaría a la segunda opción de arranque, sin iniciar el proceso.

WinBox nos permite crear unas reglas de arrendamiento o *leases*, en las cuales establecer mediante las direcciones *MAC* de las tarjetas de red de los dispositivos conectados a la *LAN*, una *IP* estática o fija a un determinado dispositivo. Esto interesa de cara a tener identificados los dispositivos que sabemos habituales en una red de trabajo.

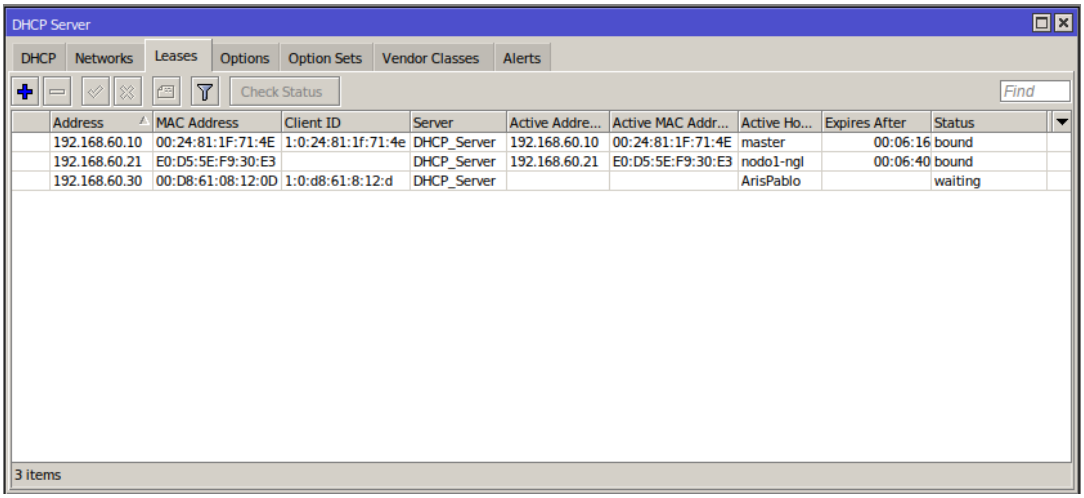

Figura 4.23: Arrendamientos *DHCP* del laboratorio. (Elaboración propia, 2022)

En la imagen anterior, se muestran tres direcciones *IP*, pertenecen a los dispositivos conectados en ese determinado momento a la red. La *IP 192.168.60.10* pertenece al PC máster, la *IP 192.168.60.21* pertenece al PC cliente y la *IP 192.168.60.30* pertenece al PC con el cual hemos realizado la configuración del rúter, es un portátil cedido por la empresa durante el período de prácticas. Hemos creado estos arrendamientos con respecto a unas normas establecidas para detectar dependiendo del rango de la *IP* el tipo de equipo conectado a la red:

- La dirección *IP 192.168.60.1* pertenece al rúter, actúa como puerta de enlace;
- La dirección *IP 192.168.60.10* pertenece al PC máster;
- Las direcciones *IP 192.168.60.21 - 192.168.60.29* pertenecen a los nodos clientes;
- La dirección *IP 192.168.60.30* pertenece al portátil (*Admin*) de la empresa con el cual hemos configurado *WinBox*;
- Las direcciones *IP 192.168.60.50 - 192.168.60.100* son los dispositivos que se conectan puntualmente, externos a la red de trabajo.

Esta división la conseguimos gracias a asignar arrendamientos estáticos a todos los componentes de la red del laboratorio, dejando un *IP Pool* o grupo de direcciones *IP* dinámicas para los equipos o dispositivos con conexiones puntuales.

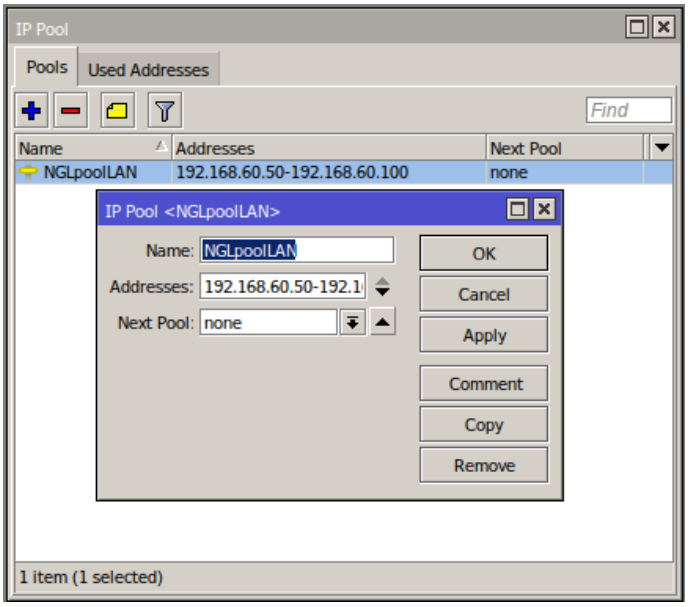

Figura 4.24: *IP Pool* del laboratorio. (Elaboración propia, 2022)

Para completar el proceso de creación de la *LAN* de nuestro laboratorio de acceso remoto multiplataforma para procesos de *Quality Assurance* (*QA*) *Testing*, necesitamos un servidor *DNS*, encargado de traducir las direcciones *IP* de la red en nombres de dominio para salir a *Internet*.

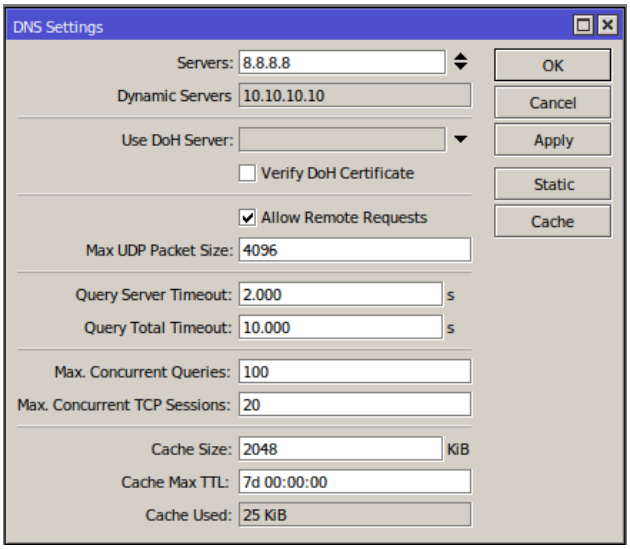

Figura 4.25: Servidor *DNS* del laboratorio. (Elaboración propia, 2022)

Con los pasos anteriores hemos conseguido que un PC conectado a nuestra red de trabajo, tenga salida a *Internet*. A continuación, como se nos exige en el apartado «3.1 Especificación de requisitos», debemos otorgar seguridad a la red y posibilidad de acceso remoto a los usuarios.

La seguridad en una red va de la mano del *firewall*, crear reglas en nuestro cortafuegos es vital para evitar intrusiones no deseadas y posibles ataques del exterior. En nuestro *firewall* creamos tres listas de direcciones *IP* permitidas para realizar las conexiones, de este modo evitamos que se establezca una conexión con el laboratorio externa a nuestra subred o que no provenga de la red de la empresa.

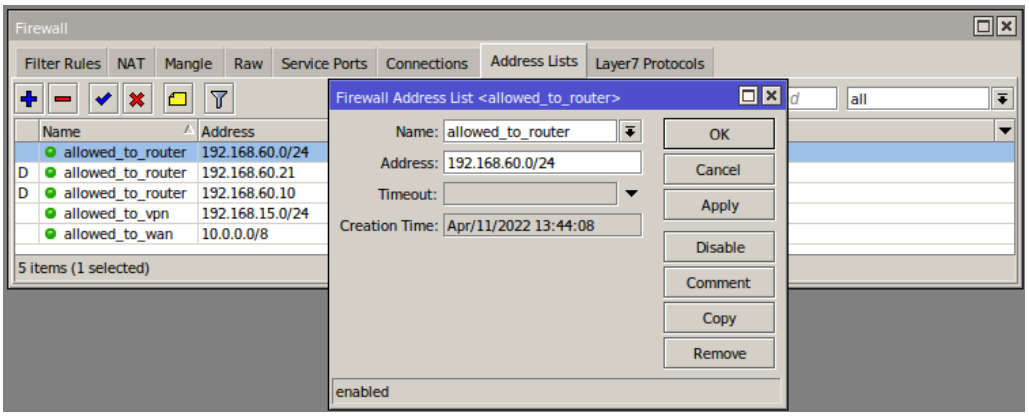

Figura 4.26: Rango de direcciones *IP* permitidas, *LAN*. (Elaboración propia, 2022)

El rango de direcciones permitidas de las cuales se puede acceder al laboratorio son *allowed-to-ruter* (*IP 192.168.60.0/24*), la propia subred del laboratorio. La red de trabajo de Arisnova S.L. *allowed-to-wan* (*IP 10.0.0.0/8*). Y la red VPN de la empresa *allowed-to-vpn* (*IP 192.168.15.0/24*), para permitir accesos remotos desde el exterior.

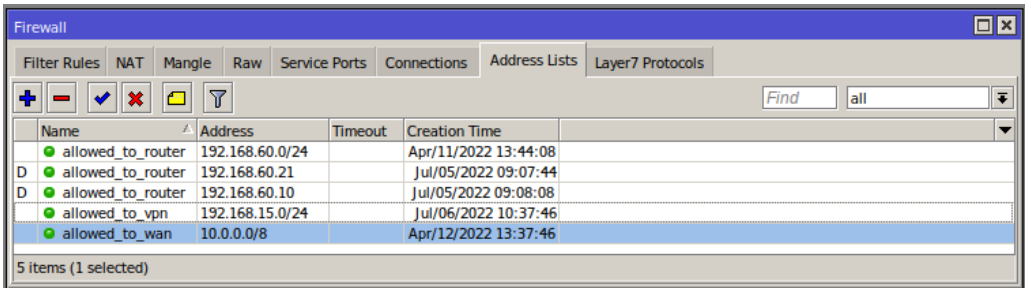

Figura 4.27: Rango de direcciones *IP* permitidas, *WAN*. (Elaboración propia, 2022)

Debemos añadir un listado de reglas a nuestro *firewall*, dónde indicarle las acciones permitidas y bloqueadas dentro de nuestra red de trabajo. Como vemos en la imagen posterior, permitimos explícitamente el uso de los protocolos *SSH* y *L2TP VPN*, para conceder acceso remoto de los usuarios al sistema. A su vez, permitimos el tráfico proveniente de la propia *LAN*, *WAN* y *VPN*, el resto de tráfico queda bloqueado.

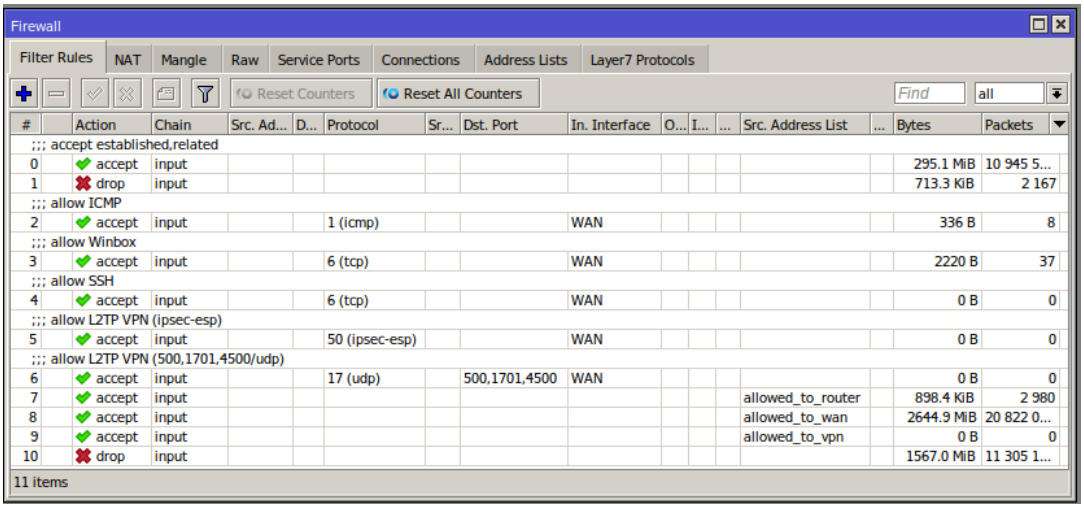

Figura 4.28: Creación de reglas en el *firewall* del laboratorio. (Elaboración propia, 2022)

Para finalizar la seguridad, debemos configurar la traducción de direcciones de red o *NAT*. Este protocolo nos permite utilizar las direcciones *IP* para comunicarnos entre los dispositivos internos de la red y también con los externos, sin este protocolo de traducción, no habría entendimiento entre dispositivos o redes y no se podría establecer una conexión a *Internet*.

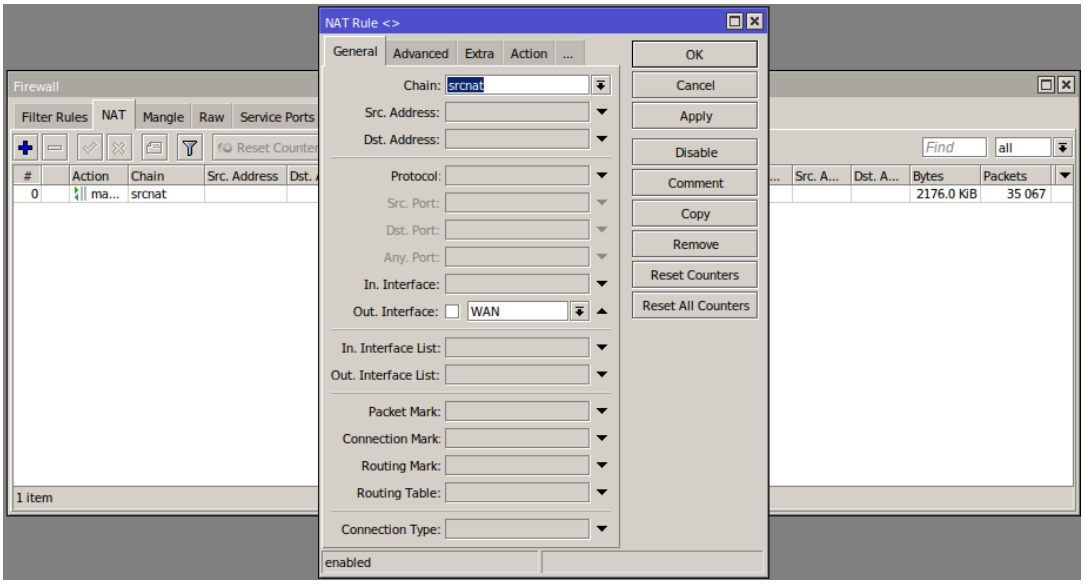

Figura 4.29: Configuración del protocolo *NAT* del laboratorio. (Elaboración propia, 2022)

Esta configuración pasa por la selección de *IP->Firewall->NAT*, creando una regla de *NAT* con la cadena *srcnat* en la interfaz de salida *WAN*. Esto reemplaza la dirección *IP* privada por una nueva dirección *IP* pública para salir a *Internet*. También seleccionamos la acción *masquerade* para soportar *IPs* públicas dinámicas, en caso de que nuestro operador de red cambie nuestra *IP* pública.

Para desbloquear la utilidad *WebFig* y tener acceso a la configuración del enrutador por *web*, tenemos que habilitar en el rúter el protocolo correspondiente en *IP->IP Service*, permitiendo a través del puerto 80 en este caso, aunque podemos modificarlo, cargar la *web* de configuración de nuestro rúter en un buscador.

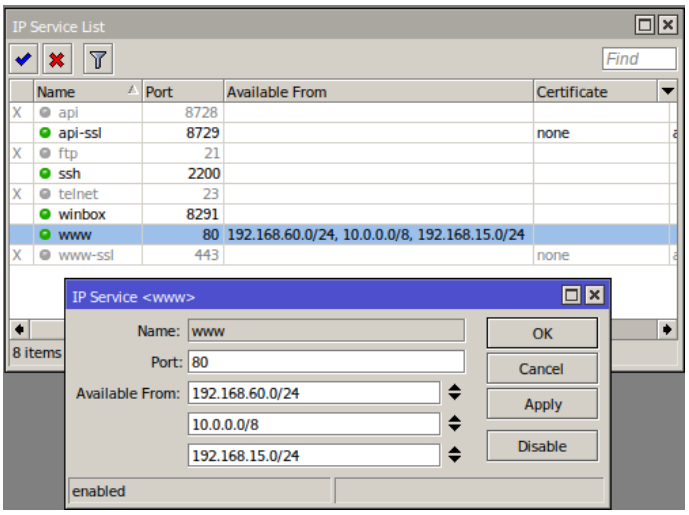

Figura 4.30: Listado de los servicios *IP* del laboratorio. (Elaboración propia, 2022)

Como vemos en la imagen siguiente, la utilidad *WebFig* contiene todos los parámetros y opciones que *WinBox* nos ofrece, con la ventaja de no tener que estar físicamente conectado al rúter para poder acceder a él. Por motivos de seguridad, *WebFig* únicamente es accesible desde la *LAN* del laboratorio o a través de la red de la empresa.

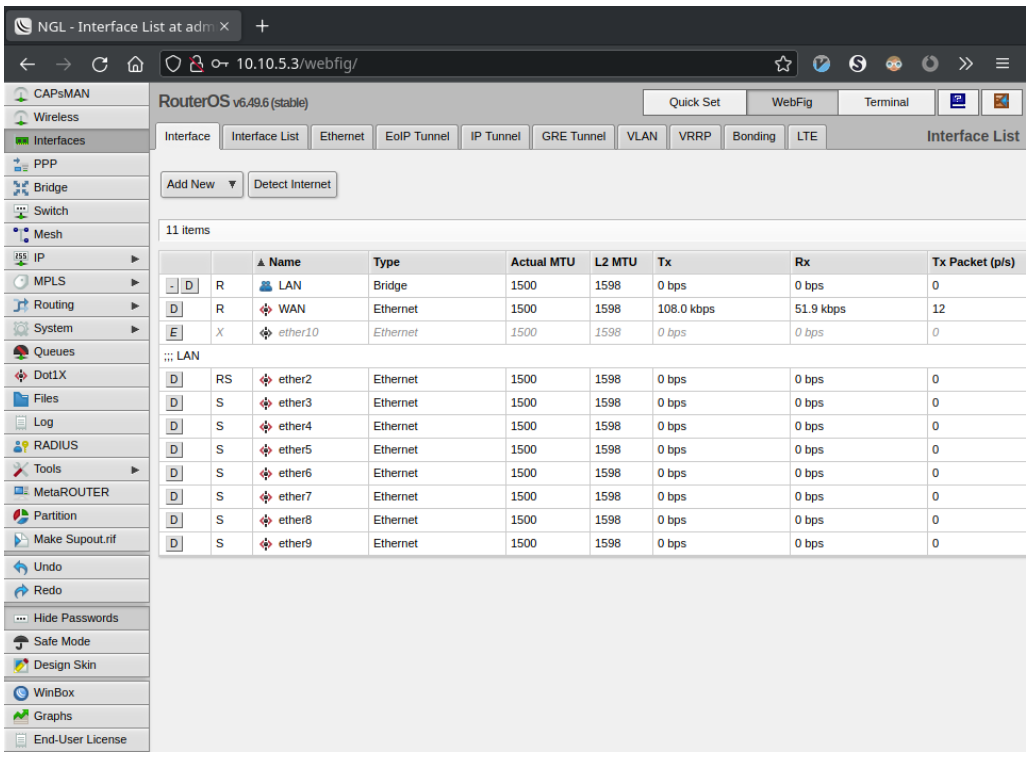

Figura 4.31: Utilidad de configuración *WebFig* del laboratorio. (Elaboración propia, 2022)

Accedemos a través de un buscador introduciendo en la barra de búsqueda la dirección siguiente, siempre y cuando nos encontremos dentro de la subred del laboratorio o en la red de la empresa, ya sea por localización física o por conexión *VPN*.

`http://10.10.5.3/webfig/`

4.5.2. Configuración del PC máster

El sistema de aprovisionamiento *Cobbler* está preparado para administrar los servidores *DHCP* y *DNS* internamente, no es obligatorio que estos servicios los proporcione el mismo *Cobbler*, pero sí opcional. Nosotros en el laboratorio para evitar dependencias excesivas en el PC máster de cara a un posible fallo o avería, hemos decidido administrar estos servicios desde el rúter *Mikrotik RB2011IL-RM*. Reduciendo los archivos de configuración en el nodo máster, pues la labor de gestión de la red, la soporta dicho enrutador.

Los archivos necesarios para que el sistema de aprovisionamiento haga su función son:

- *dhcpd.template*;
- *dnsmasq.template*;
- *dhcpd.conf*;
- *settings.yml*.

Puesto que *Cobbler* no se encarga del soporte de la red del laboratorio, únicamente configuraremos el fichero *settings.yml*, dejando los otros servicios deshabilitados en el PC máster. La descripción de este archivo se encuentra en el Apéndice A «A.2 Archivos de configuración *Cobbler*».

Como hemos mencionado en el apartado «4.4 Tecnología utilizada», previa instalación y configuración del sistema de aprovisionamiento *Cobbler*, hemos instalado el sistema operativo *Red Hat Enterprise Linux 8.5* sin interfaz de usuario en el nodo máster, la instalación de *Cobbler* se realiza mediante línea de comandos, haciendo inútil tener una interfaz gráfica en el máster.

Para instalar el sistema de aprovisionamiento y realizar toda la configuración previa a la utilización de este *software*, podemos visualizar el Apéndice A «A.1 Instalación de *Cobbler* en el PC máster» donde mostramos una pequeña receta paso a paso de cómo instalar este sistema.

Una vez tenemos esta parte completa, aún no hemos finalizado la configuración, pues falta lo más importante. Para que *Cobbler* pueda realizar instalaciones desatendidas, necesita tener importados los sistemas operativos a utilizar, creados los perfiles y el sistema, esto se consigue mediante un conjunto de comandos introducidos por terminal.

Tomamos como referencia la distribución *Alma Linux 8.5* por su simplicidad a la hora de descargarla de un repositorio mediante el comando *wget*. Pues las distribuciones *RHEL*, mediante el comando *wget* no hemos podido descargarlas, hemos utilizado un equipo intermedio para descargarnos el archivo *ISO* previa inserción de credenciales en la *web* de *Red Hat*, y transferido el archivo al PC máster con *scp*. La necesidad de estar *loggeado* en la cuenta *RHEL* para realizar las descargas, nos impedía utilizar los comandos habituales de descarga.

Para importar una *ISO* a *Cobbler*, esta debe estar descargada en el PC en el cual se encuentra este sistema de aprovisionamiento. Mediante el comando *wget* descargamos la *ISO* del repositorio oficial de la distribución, creamos un directorio para almacenarla y por último, montamos el archivo para poder importarlo.

```
wget http://repo.ifca.es/almalinux/8.5/isos/x86_64/AlmaLinux-8.5-x86_64-dvd.iso
mkdir -p /mnt/iso/alma8
mount -t iso9660 -o loop,ro AlmaLinux-8.5-x86_64-dvd.iso /mnt/iso/alma8
```

Acto seguido, ya podemos importar la distribución *Alma Linux 8.5* a *Cobbler*. Con el comando *cobbler import* indicándole la arquitectura del archivo *ISO*, su localización y le asignamos un nombre para poder identificarlo en el sistema.

```
cobbler import --arch="x86_64" --path="/mnt/iso/alma8" --name="ALMA8"
```

Mediante el comando *cobbler distro list*, visualizamos un listado de todas las distribuciones importadas al sistema. Estas distribuciones han sido validadas previamente por un sistema de seguridad de firmas interno de *Cobbler*.

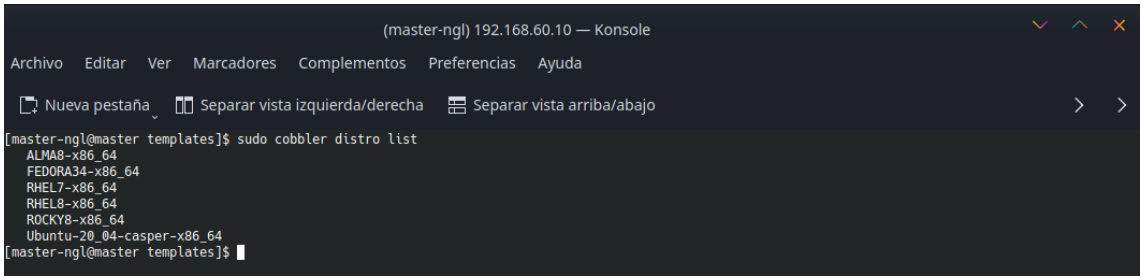

Figura 4.32: Distribuciones importadas en *Cobbler*. (Elaboración propia, 2022)

En el caso de querer importar una distribución que no tenga una estructura interna correcta, que haya salido recientemente al mercado, o que no disponga de una firma en el repositorio de firmas de *Cobbler*, este impedirá la importación, mostrando un mensaje por terminal y cancelando el proceso.

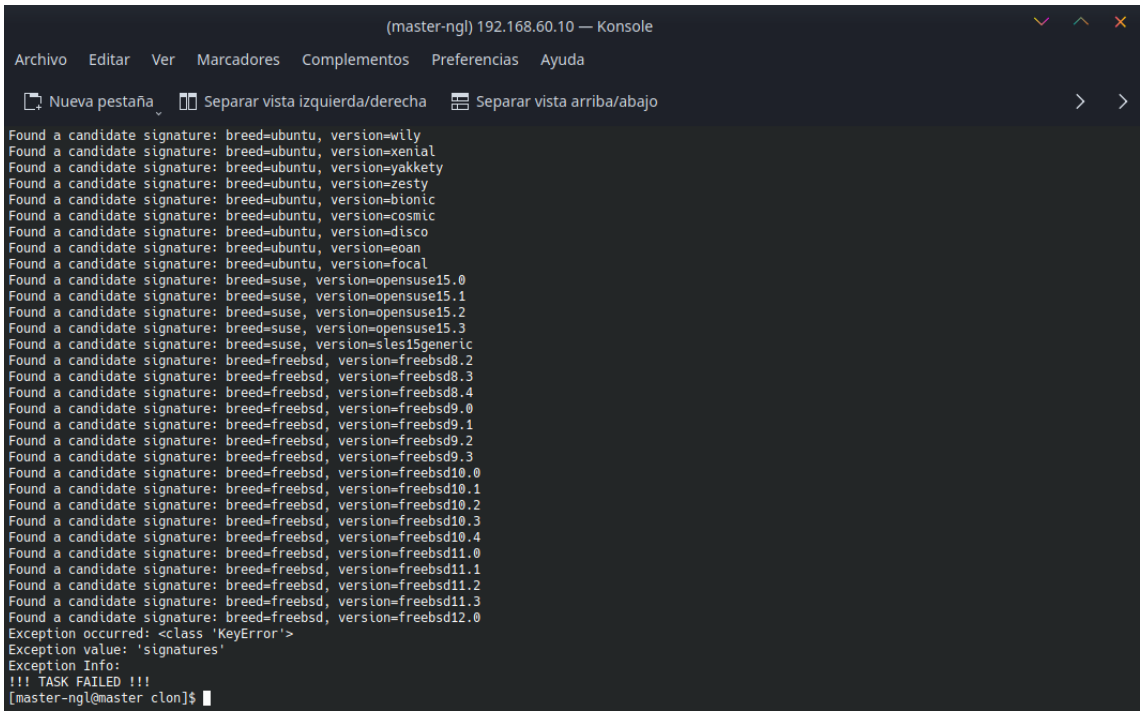

Figura 4.33: Fallo en la importación a *Cobbler*. (Elaboración propia, 2022)

Esto es lo que nos ha sucedido en la generación de *ISOs* modificadas, como mencionamos en el apartado «3.4.4 Implementación de un sistema de copia del entorno de ejecución». Lo que hemos descubierto hasta ahora, es que la estructura de los directorios una vez montas el archivo es distinta en las distribuciones aceptadas por *Cobbler* y en las que impide importar.

La imagen siguiente corresponde a la estructura de directorios de la distribución *RHEL 8.5* validada por *Cobbler*:

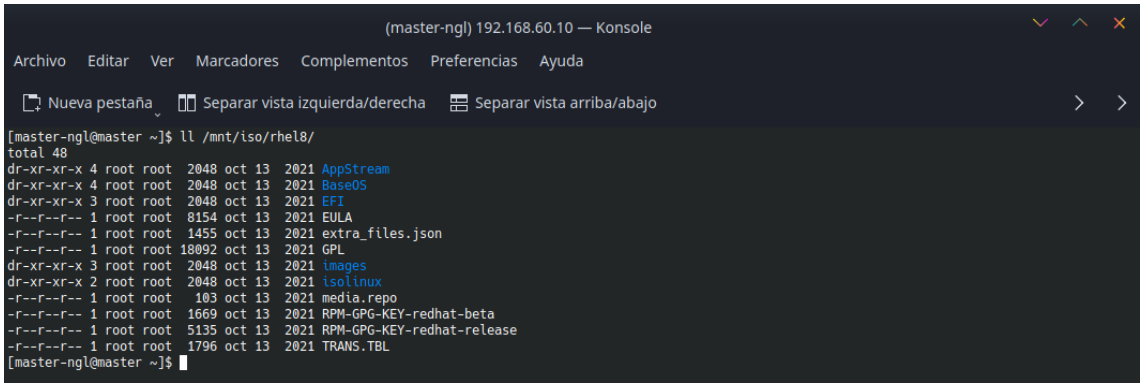

Figura 4.34: Estructura distribución *RHEL 8.5*. (Elaboración propia, 2022)

A continuación, se muestra un intento de generación de *ISO* con el *software Clonezilla*, en la cual se aprecia la clara diferencia en la disposición de los directorios, esto impide que *Cobbler* reconozca el archivo como *ISO* y lo valide para ser añadido al repositorio que hemos generado.

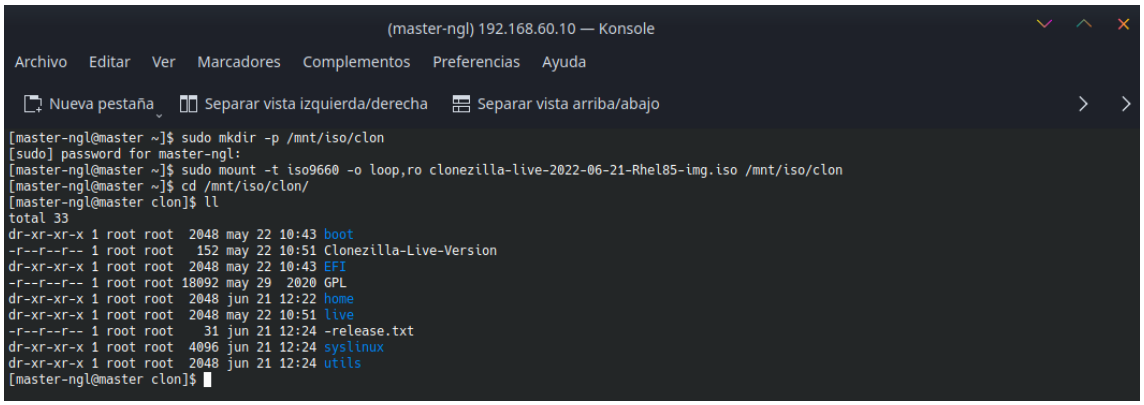

Figura 4.35: Estructura distribución creada con *Clonezilla*. (Elaboración propia, 2022)

Puesto que de momento no hemos encontrado una solución para este problema, hemos decidido utilizar *Clonezilla* mediante una utilidad distinta a la mostrada, como veremos en una sección posterior.

Una vez tenemos la distribución importada a *Cobbler*, necesitamos generar un perfil, este perfil enlaza la distribución importada con el archivo *kickstart* (*flag –autoinstall*) que queremos instalar, el cual contiene la configuración deseada por nuestra parte y lo almacena con un nombre concreto. El archivo *kickstart* se crea y almacena por defecto en el directorio interno *templates*, los archivos utilizados en el desarrollo del proyecto los podemos ver en el Apéndice A «A.3 Archivos *kickstart* utilizados».

```
cobbler profile add --name="ALMA8-Nodo1" --distro="ALMA8-x86_64"
--autoinstall="alma8.ks"
```

Esto es una modificación considerable de la última actualización de este sistema, en las versiones anteriores el *flag* para indicar el archivo *kickstart* era *–kickstart* y se le indicaba la ubicación del archivo en el comando, pues no buscaba por defecto en *templates*. Aquí un ejemplo de cómo era en una versión anterior este comando.

```
cobbler profile add --name="ALMA8-Nodo1" --distro="ALMA8-x86_64"
--kickstart="/var/lib/cobbler/kickstarts/alma8.ks"
```

Mediante el comando *cobbler profile list*, visualizamos un listado de todos los perfiles creados en el sistema.

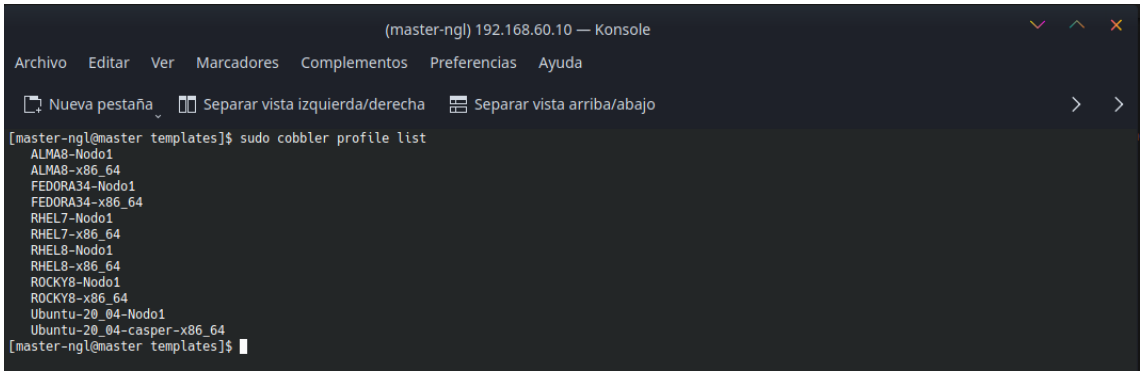

Figura 4.36: Perfiles creados en *Cobbler*. (Elaboración propia, 2022)

Por defecto, *Cobbler* al importar una distribución genera automáticamente un perfil para esta. Estos perfiles nosotros no los utilizamos pues no están enlazados con los *kickstart* que hemos creado. Una opción en vez de crear un perfil nuevo, podría ser editar el perfil generado. Puesto que preferíamos hacer todo el trabajo nosotros, en parte para aprender y en parte para tener la certeza de estar haciéndolo correctamente, hemos dejado estos perfiles sin utilizar. Como dato adicional, los perfiles que hemos creado nosotros, son los acabados en -Nodo1.

El siguiente paso a la creación de un perfil, es la creación de un sistema, este paso es crítico para el buen funcionamiento del laboratorio, pues estamos enlazando el perfil que contiene la instalación previamente configurada con nuestro archivo *kickstart* y con el PC del laboratorio al cual queremos instalar ese sistema operativo. Para esto, en la creación del sistema se le pasan los *flags –inetrface* y *–mac* los cuales son el nombre de la interfaz por la que está escuchando el nodo cliente y su dirección *MAC* de la tarjeta de red. Sin estos campos, es imposible comunicarse con el nodo a instalar.

```
cobbler system add --name="nodo1" --profile="RHEL8-Nodo1" --interface="enp4s0"
--mac="E0:D5:5E:F9:30:E3" --netboot-enabled="Y"
```

También como novedad de la última versión, se ha añadido el *flag –netboot-enabled="Y"*, esto corresponde a habilitar en el arranque *PXE-DHCP* la instalación deseada, sin este parámetro, no se podría automatizar la instalación, pues no inyectaría el S.O. al nodo cliente. Este *flag* sólo esta a *Y* (*yes*) una vez, cuando reinicias el nodo instalado, se inicia con el sistema operativo que tiene recién instalado y no vuelve a cargar un proceso nuevo de instalación, el parámetro automáticamente ha cambiado a *N* (*no*).

Este paso lo podemos apreciar mejor en la siguiente imagen, en el archivo presente en el PC máster */var/lib/tftpboot/01-e0-d5-5e-f9-30-e3* correspondiente al arranque del nodo cliente, se puede apreciar el cambio que hace automáticamente una vez ha instalado el sistema mediante *Cobbler*.

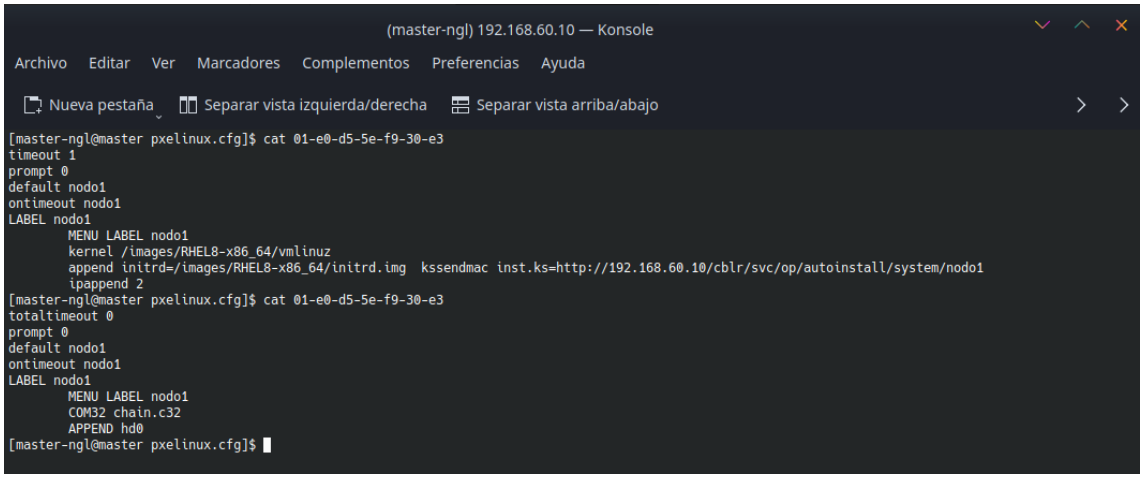

Figura 4.37: Cambio automático arranque *PXE-DHCP*. (Elaboración propia, 2022)

Al generar el sistema, el archivo se crea una referencia a una dirección *web* asociada al sistema *Cobbler*, en la cual se encuentra el archivo *kickstart*, cuando reinicia, automáticamente se reescribe con *hd0*, esto le indica al nodo que inicie desde el disco interno y no por red. Este paso es la consecuencia del cambio de valor del *flag –netboot-enabled="Y"* a *–netboot-enabled="N"*.

Para volver a crear un proceso de instalación, habría que editar el sistema, para volver a activar el parámetro, o borrarlo y volverlo a crear. De esta manera, tienes control total sobre la instalación de un nodo y únicamente se vuelve a instalar cuando modificas estos valores.

```
cobbler system edit --name="nodo1" --netboot-enabled="Y"
cobbler system remove --name="nodo1"
```

Añadir que ningún cambio en *Cobbler* se hace efectivo hasta que realizamos *cobbler sync* previa comprobación de *cobbler check*. Estos parámetros chequean el sistema en busca de errores, si está todo bien configurado podremos sincronizar el sistema para que se efectúen los cambios. Para cada modificación, es necesaria una sincronización para hacerla efectiva.

La única distribución en la que no utilizamos de momento un archivo *kickstart*, es en *Ubuntu Linux 20.04*, pues no hemos conseguido automatizar por completo el proceso, al generar el sistema, sí se crea el enlace *web* al archivo *kickstart*, pero en el proceso de instalación es incapaz de leer ese enlace dando un problema de *URL* no encontrada[33].

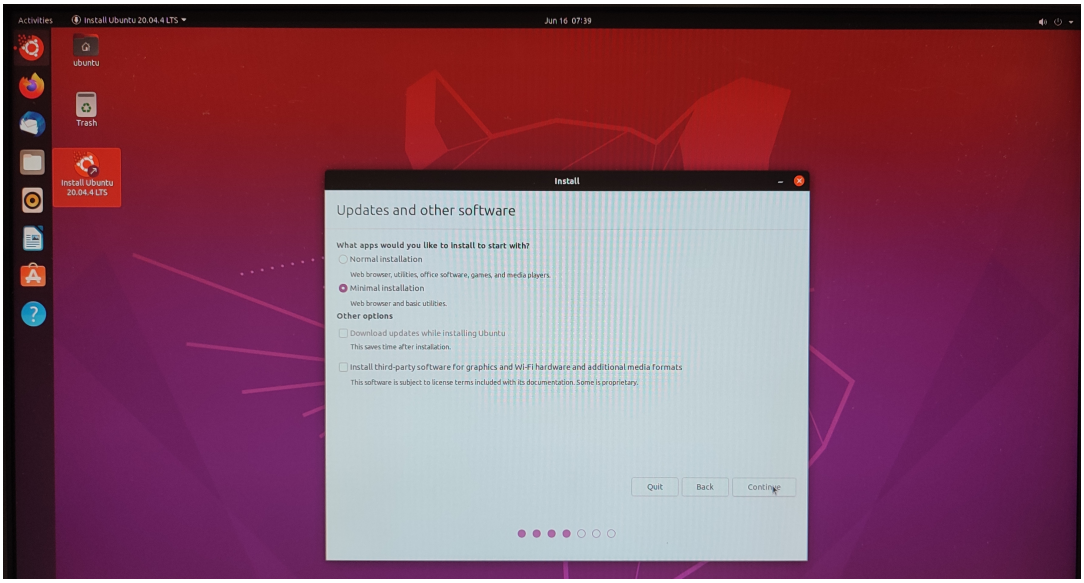

Figura 4.38: Instalación semiautomática *Ubuntu Live*. (Elaboración propia, 2022)

La solución momentánea a este problema, está en reescribir el archivo */var/lib/tftpboot/01-e0-d5-5e-f9-30-e3* con la ubicación del archivo *ISO* almacenado en el PC máster, la instalación se hace modo *Live*, es una instalación que precarga el sistema en la memoria *RAM*, inicia y se genera un icono en el escritorio, el cual una vez lo seleccionamos comienza una instalación manual del S.O. Consiguiendo como resultado una instalación completa, pero no automatizada.

```
/var/lib/tftpboot/pxelinux.cfg/01-e0-d5-5e-f9-30-e3
timeout 1
prompt 0
default nodo1
ontimeout nodo1
LABEL nodo1
     MENU LABEL nodo1
     kernel /images/Ubuntu-20_04-casper-x86_64/vmlinuz
     append initrd=/images/Ubuntu-20_04-casper-x86_64/initrd autoinstall
            url=http://192.168.60.10/ubuntu-20.04.4-desktop-amd64.iso
            s=http://192.168.60.10/ks/ cloud-config-url=/dev/null
            ds=nocloud-net  ip=dhcp fsck.mode=skip ---
ipappend 2
```

Tras finalizar la instalación semiautomática de *Ubuntu Linux 20.04*, reescribimos el archivo */var/lib/tftpboot/01-e0-d5-5e-f9-30-e3* al contenido que *Cobbler* reemplazaría automáticamente si funcionara correctamente, para que inicie el sistema recién instalado.

```
totaltimeout 0
prompt 0
default nodo1
ontimeout nodo1
LABEL nodo1
     MENU LABEL nodo1
     COM32 chain.c32
     APPEND hd0
```

Como dato a tener en cuenta, en el momento creamos un sistema en *Cobbler* con una dirección *MAC* y una interfaz de red, estos datos no se pueden volver a introducir en otro sistema sin previamente borrar el actual, esto quiere decir, que puedes tener tantos sistemas generados como nodos clientes tengas en funcionamiento a la vez, un único sistema por cada PC.

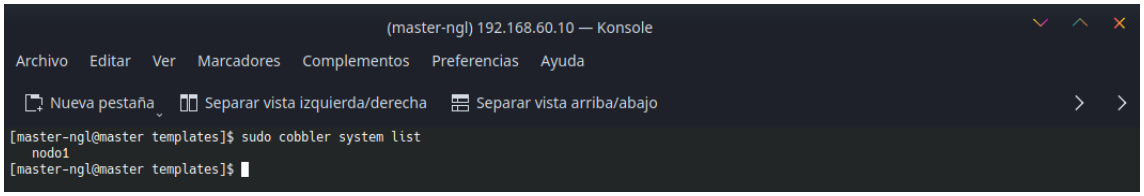

Figura 4.39: Sistemas creados en *Cobbler*. (Elaboración propia, 2022)

En la configuración de *Cobbler* nos hemos encontrado diversos problemas a la hora de conseguir el comportamiento esperado del sistema, a parte de los mencionado anteriormente en esta sección correspondientes al cambio de versión, debemos sumarle dos problemas inesperados, a los cuales hemos podido hacer frente con la ayuda de una intensa investigación y de la comunidad *Cobbler*, a la cual debemos agradecer su corto plazo de respuesta y su interés y seguimiento hacia los usuarios de este sistema de aprovisionamiento.

El primero de ellos, sucedía una vez habíamos generado el sistema de cualquier perfil. En este caso en concreto, sucedía con todas las distribuciones que probamos, lo que nos hacía sospechar que no era un problema nuestro y sí un problema no detectado en el cambio de versiones de *Cobbler*.

Al intentar acceder el nodo cliente a la *URL* generada desde el sistema para obtener los archivos e iniciar la instalación, era inaccesible, pues había unos ficheros importantes internos en *Cobbler* con los permisos mal otorgados, esto hacía que una acción básica como la de descargarse los archivos de la instalación, fuese imposible para el nodo cliente, impidiendo así el proceso de instalación.

Para este problema abrimos una incidencia en el *GitHub* de *Cobbler*[34], tras muchas horas de investigación y revisión de incidencias que pudieran ser similares, encontramos una solución a este problema.

```
chmod g+rw /etc/cobbler /etc/cobbler/settings.yml /etc/cobbler/modules.conf
usermod -aG root apache
systemctl restart httpd cobbler
```

Había que añadir permisos de lectura y escritura a los archivos especificados anteriormente, añadir al usuario *apache*, el encargado de compartir los ficheros de instalación con el nodo cliente al grupo *root* del sistema y por último reiniciar los servicios de *Apache* y *Cobbler*. Con estos pasos conseguimos por primera vez ver un proceso de instalación completo.

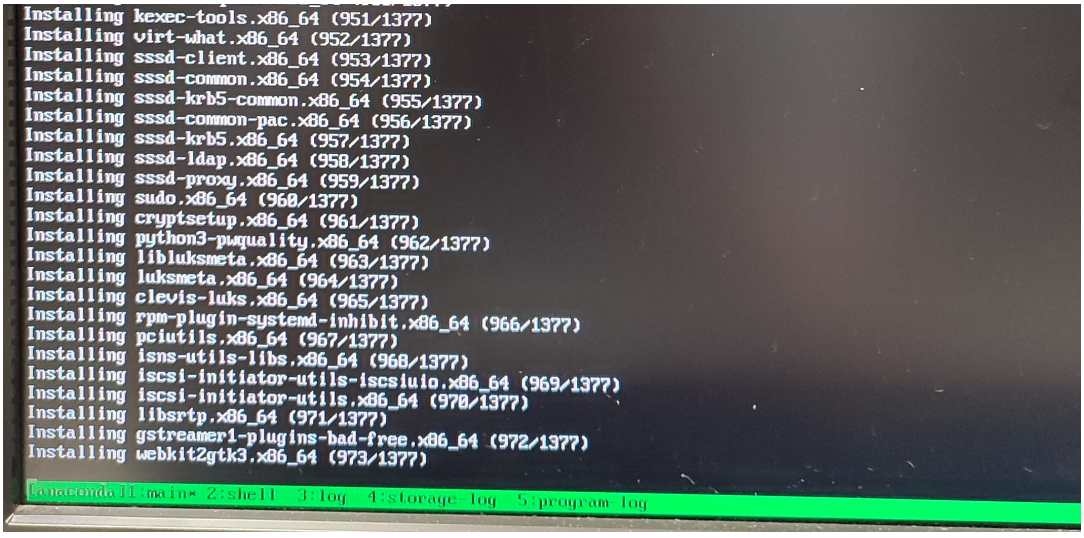

Figura 4.40: Instalación S.O. mediante *Cobbler*. (Elaboración propia, 2022)

4.5.3. Configuración del PC cliente

Para la configuración del nodo cliente únicamente hay que acceder a la *UEFI* del equipo y modificar los parámetros *LAN PXE Boot Option ROM ->Enabled*, permite realizar un arranque por red de un sistema. *Boot Option Priorities* (*Boot Option 1*) *->Realtek PXE B04 D00*, modificamos el orden de arranque para que el sistema arranque en modo *PXE*, para detectar la instalación del S.O. lanzado por *Cobbler*.

La segunda opción de arranque, *Boot Option Priorities* (*Boot Option 2*) *->P3: TOSHIBA-TR200*, debe ser el disco de arranque en el cual *Cobbler* instalará el S.O. En nuestro caso es el disco *SSD TOSHIBA-TR200*, la tercera opción de arranque por mencionarla, es el disco en el cual el sistema de generación de copias de imágenes *Clonezilla*, almacenará sus copias.

Como vemos en la imagen posterior, la opción *CSM Support* está habilitada. Esto forma parte del segundo problema mencionado en el apartado anterior, en los sistemas *UEFI* se tiene que deshabilitar esta opción para que *Cobbler* funcione correctamente, es una herramienta utilizada para mantener la compatibilidad con sistemas de arranque antiguos (*BIOS*).

Figura 4.41: Configuración *UEFI* nodo cliente. (Elaboración propia, 2022)

El segundo problema al cual nos enfrentamos, una vez terminado el proceso de instalación, el sistema era incapaz de reiniciar el sistema con el S.O. instalado, y realizaba un proceso de instalación en bucle, salvo que accediéramos a la *UEFI* del nodo cliente y cambiáramos las opciones de arranque manualmente. Este inconveniente ya no era problema de la versión de *Cobbler*.

Después de largos días de investigación, a base de mucho esfuerzo y dedicación conseguimos obtener la fuente del problema. El PC cliente es un equipo montado a piezas, lo que se denomina vulgarmente como *Clon*, los componentes se seleccionan y se montan teniendo en cuenta la compatibilidad de los mismos.

Al parecer la placa base y la tarjeta gráfica no son compatibles en su totalidad, de hecho si modificamos la opción *CSM Support ->Disabled*, el PC automáticamente pierde la imagen y es incapaz de iniciar correctamente, dejando como única opción resetear manualmente la placa base y volver a configurar la *UEFI*. Este hecho indica que la *GPU* no realiza su trabajo convenientemente, pues debería permitir deshabilitar esta opción. Sumado a todo eso, algunas tarjetas de red son incompatibles con versiones de *BIOS-UEFI* impidiendo el funcionamiento de *localboot -1*[35].

La versión actual de *Cobbler* genera el archivo */var/lib/tftpboot/01-e0-d5-5e-f9-30-e3* como podemos observar a continuación:

```
totaltimeout 0
prompt 0
default nodo1
ontimeout nodo1
LABEL nodo1
     MENU LABEL nodo1
     localboot -1
```

La opción *localboot -1*[36] significa que está dañada o corrupta la primera opción de arranque del sistema y automáticamente la *UEFI* inicia con la segunda opción, en este caso sería el disco de arranque del sistema. Resultando el comportamiento deseado y evitando bucles en el proceso de instalación.

Al no ser compatible la tarjeta de red con la *UEFI*, la opción *localboot -1* es ignorada por el sistema, por lo que hemos investigado es un problema poco común en las tarjetas de red, no es un problema muy conocido e impide un funcionamiento óptimo del *PXE*.

Investigando las opciones que podrían reemplazar a *localboot -1*, encontramos la opción de *chain.c32*[37], módulo *COM32* para *Syslinux* que permite alterar el orden de arranque de los equipos, con *hd0* conseguimos que el equipo arranque con el primer disco detectado. Modificamos parte del archivo de configuración de arranque que genera *Cobbler* automáticamente */etc/cobbler/boot-loader-conf/pxe.template*:

```
#if $system_local
COM32 chain.c32
APPEND hd0
```

Y añadimos los cargadores de arranque *chain.c32*, *libcom.c32* y *libutil.c32* encargados de implementar esta solución al directorio de cargadores de *Cobbler*:

```
cp /usr/share/syslinux/chain.c32 /var/lib/cobbler/loaders/
cp /usr/share/syslinux/libcom.c32 /var/lib/cobbler/loaders/
cp /usr/share/syslinux/libutil.c32 /var/lib/cobbler/loaders/
```

Tras estas modificaciones, conseguimos el cambio de opción de arranque automático y el automatizar el proceso de instalación. Con el inicio del problema, creamos una discusión en la comunidad[38] para solicitar ayuda externa sin mucho éxito, pues desconocían el problema. Los desarrolladores de *Cobbler* han seleccionado nuestra solución como válida para este problema, resaltada en verde.

4.5.4. Configuración de *Clonezilla*

Mencionado anteriormente en el apartado «4.4 Tecnología utilizada», hay tres tipos de *software* disponible, *Clonezilla Live* , *Clonezilla Lite Server* y *Clonezilla Server Edition*. Para el apartado de clonación de una imagen en el segundo disco del PC cliente, con el objetivo de guardar una copia del sistema operativo modificado, para su posterior reutilización, con tal de aprovechar el *software* instalado y no empezar una instalación desde cero, necesitamos la utilidad *Clonezilla Live*[39].

Esta se instala en una unidad *flash USB* y se carga en el sistema desde este *USB*, no necesita ser instalada en el disco de almacenamiento, pero sí debe tener acceso a este disco para poder almacenar las copias que queramos generar.

Entrando en la configuración *UEFI* del nodo cliente, modificamos el orden de arranque del sistema, *Boot Option Priorities* (*Boot Option 1*) ->*KingstonDataTraveler 3.0*, para detectar la instalación de *Clonezilla* contenida en el *USB*. En este caso, es irrelevante seleccionar el disco de almacenamiento como segunda opción, pues lo seleccionaremos en el programa en ejecución. Únicamente hay que cerciorarse de que el disco de almacenamiento esté habilitado como una opción de arranque del sistema.

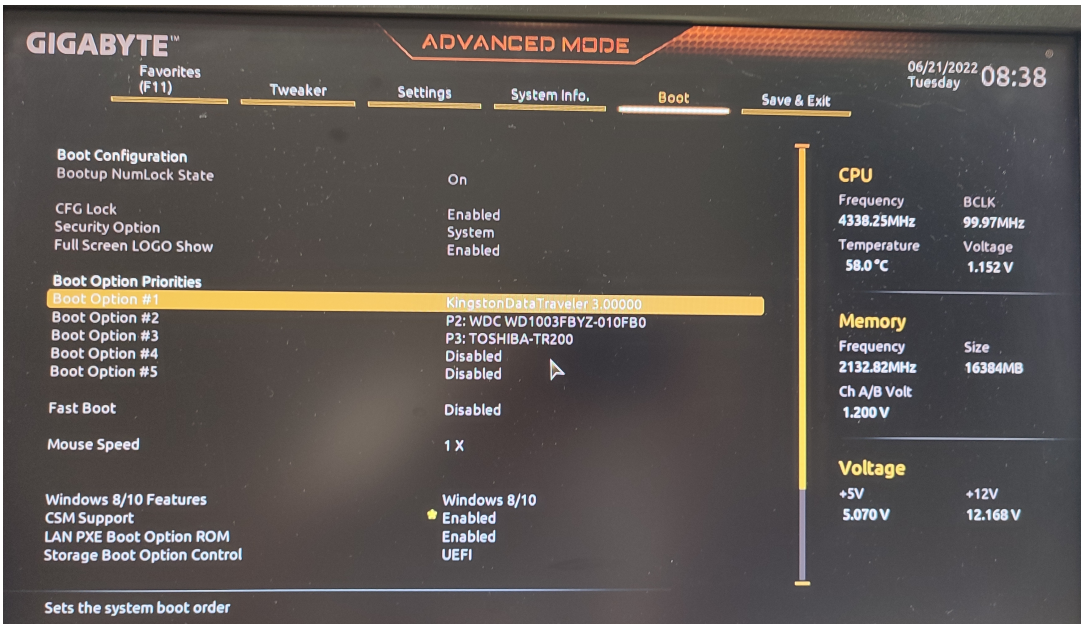

Figura 4.42: Configuración *UEFI* nodo cliente, *Clonezilla*. (Elaboración propia, 2022)

Una vez configurada la *UEFI*, arrancamos el sistema *Clonezilla Live* y comenzamos la generación de una copia de un sistema operativo instalado en el disco de arranque del PC cliente.

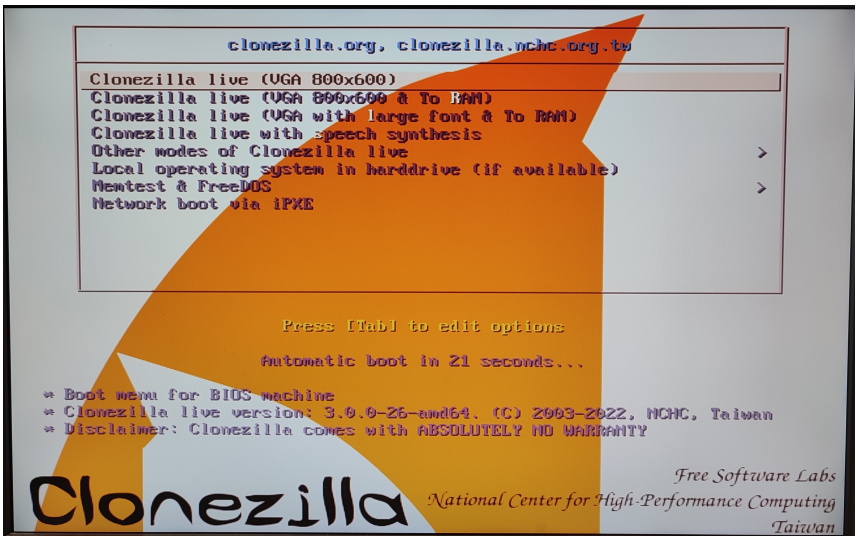

Figura 4.43: Arranque *ISO Clonezilla* nodo cliente. (Elaboración propia, 2022)

Iniciamos con la primera opción que nos aparece en el menú, seleccionando *Clonezilla Live (VGA 800x600)*, es el modo de visualización predeterminado del sistema de generación de imágenes. A continuación, seleccionamos la opción *device-image* como vemos en la imagen siguiente, pues queremos generar una imagen desde un dispositivo.

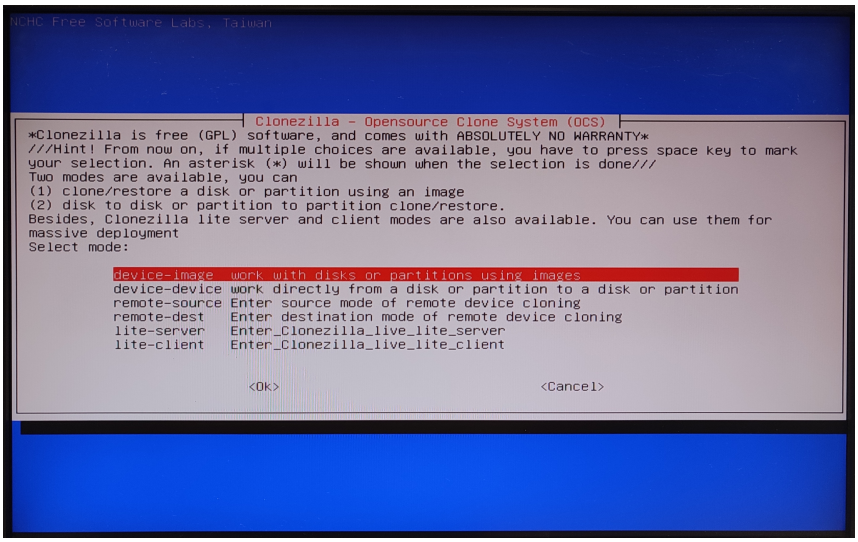

Figura 4.44: Selección *device-image Clonezilla*. (Elaboración propia, 2022)

Continuamos con la selección de parámetros, para que el *software* busque dentro de los dispositivos que se encuentran en local, hay que indicarle la opción *local-dev*. De esta forma permitimos al sistema realizar un escaneo de los discos conectados al PC cliente para identificarlos.

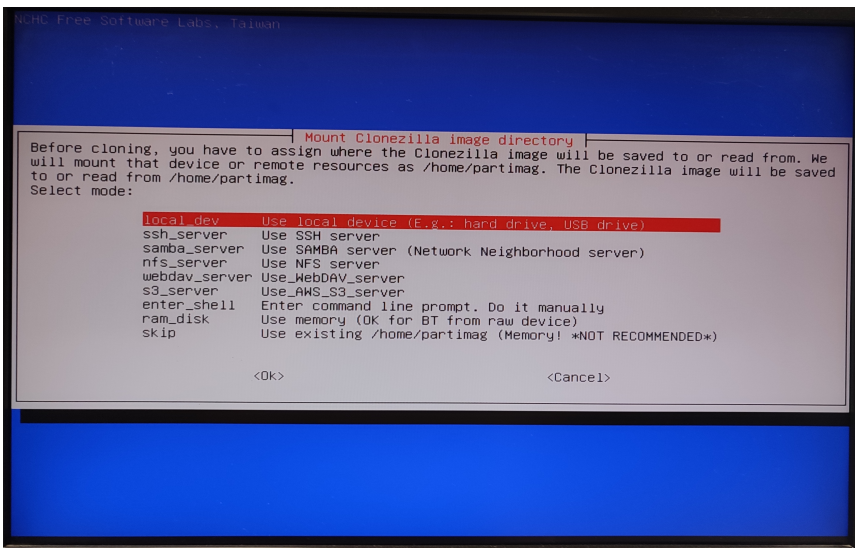

Figura 4.45: Selección dispositivo local *Clonezilla*. (Elaboración propia, 2022)

Tras seleccionar la búsqueda de dispositivos locales en el sistema, debemos elegir el disco en el cual almacenaremos las copias, esta opción es crítica. Pues una mala selección del disco borraría por completo el sistema operativo que queremos clonar, realizando una copia vacía en el disco de arranque, en vez de realizar una copia del sistema operativo que queremos guardar en el disco de almacenamiento.

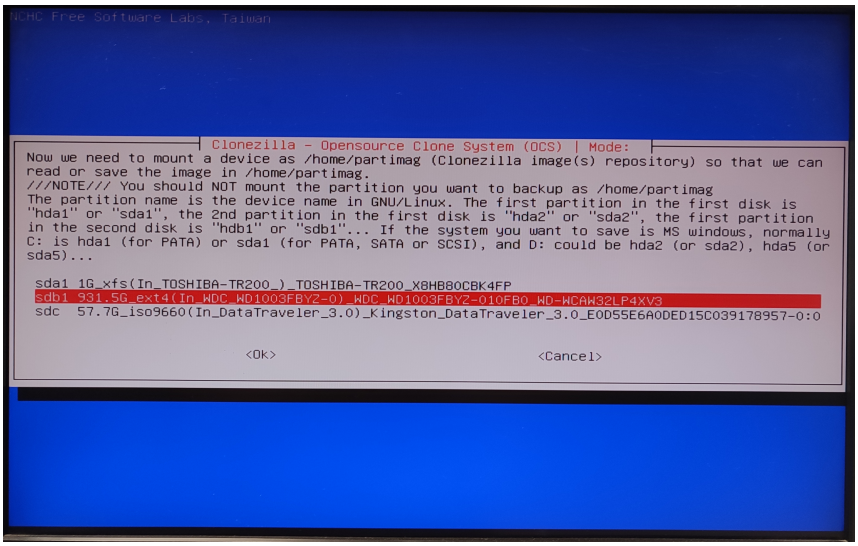

Figura 4.46: Selección disco destino copia *Clonezilla*. (Elaboración propia, 2022)

Como hemos indicado anteriormente, era irrelevante seleccionar el disco de almacenamiento como segunda opción de arranque en la *UEFI*, simplemente con habilitarlo era suficiente pues en este instante se selecciona el disco deseado.

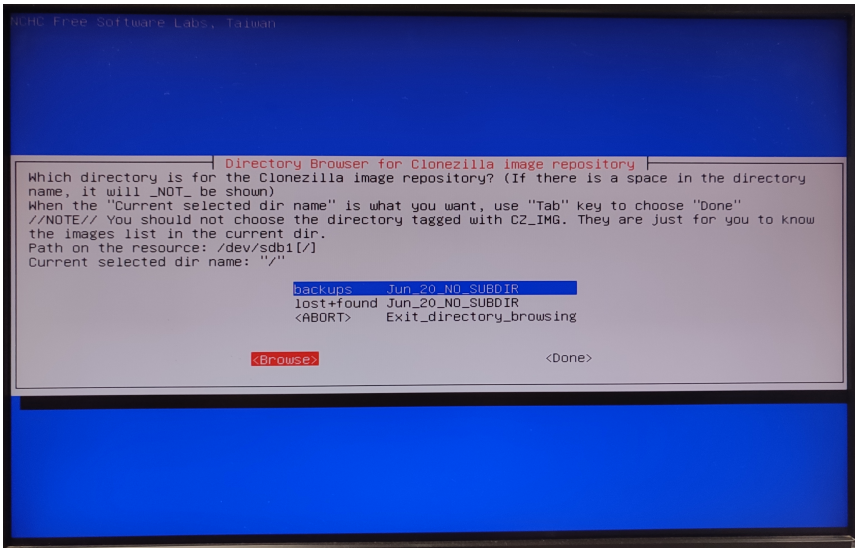

Figura 4.47: Selección directorio destino copia *Clonezilla*. (Elaboración propia, 2022)

Una vez realizada la elección del disco, toca seleccionar el directorio donde almacenar la copia que vamos a crear.

En nuestro caso, teníamos ya un directorio *backups* creado y montado desde el disco principal para poder acceder a las copias mediante el propio sistema operativo. De no tener ningún directorio creado, se puede almacenar la copia en el directorio raíz.

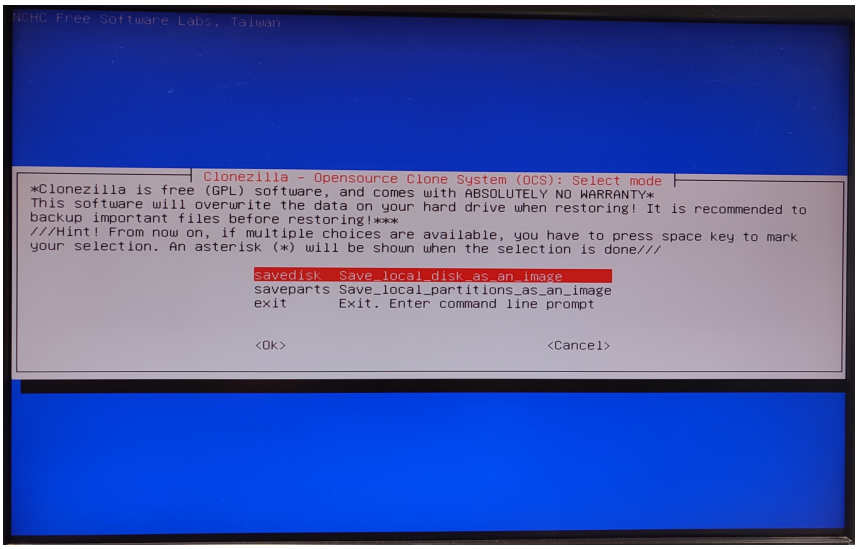

Figura 4.48: Selección guardar disco *Clonezilla*. (Elaboración propia, 2022)

En la imagen anterior, vemos el parámetro que sucede a la elección del directorio de almacenaje de la copia. Nos interesa la opción *savedisk*, pues guarda el contenido del disco como una imagen exacta de él mismo. Extrayendo los sectores del disco vacíos, para reducir el tamaño de la copia y realizar una clonación eficiente. Continuamos indicando el nombre del archivo que vamos a almacenar.

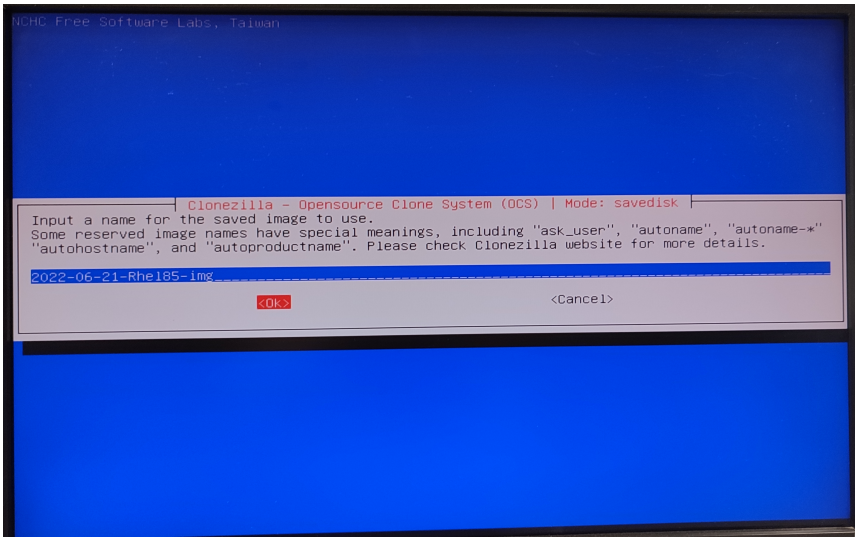

Figura 4.49: Nombre archivo clonado *Clonezilla*. (Elaboración propia, 2022)

Como vemos en la imagen anterior, la copia que queremos generar es de la distribución *RHEL 8.5*. Este procedimiento funciona con todas las distribuciones utilizadas en el laboratorio de pruebas.

En el siguiente parámetro a seleccionar únicamente aparece el disco origen del cual queremos crear la copia del sistema operativo, si en vez de dos discos, el nodo cliente tuviera un tercero, aparecería un segundo disco seleccionable a continuación.

Figura 4.50: Disco fuente copia *Clonezilla*. (Elaboración propia, 2022)

Como no es el caso, seleccionamos el disco origen de la copia y procedemos a generarla. Este proceso varía su duración dependiendo de la velocidad de los discos y el tamaño de la copia, en nuestro caso la duración no ha excedido los cinco minutos de espera.

Figura 4.51: Proceso clonación disco fuente *Clonezilla*. (Elaboración propia, 2022)

Una vez finalizada la copia de la imagen del sistema operativo del disco de arranque del PC cliente, ha quedado almacenada en el segundo disco, a la espera de ser restablecida en algún momento. Con esto finalizamos el proceso de copia de la imagen.

A continuación, invertimos el proceso realizado, la restauración de una imagen partiendo de una copia almacenada en el segundo disco del nodo. Este proceso también requiere la utilización del *softwafre Clonezilla*.

En el contexto actual, se entiende que el PC cliente ha sido reinstalado con otra distribución, previa modificación del orden de arranque de la *UEFI* a su estado funcional dentro del laboratorio de pruebas, *Boot Option Priorities* (*Boot Option 1*) ->*Realtek PXE B04 D00*, (*Boot Option 2*) ->*P3: TOSHIBA-TR200*.

Los primeros cuatro pasos del proceso de copia de una imagen se repiten, seleccionamos la opción de arranque *Boot Option Priorities* (*Boot Option 1*) ->*KingstonDataTraveler 3.0*, previa conexión al nodo cliente de la unidad *flash USB*. Seleccionamos el modo de visualización predeterminado del sistema, accedemos al parámetro *device-image* para realizar una copia entre un dispositivo y una imagen. Seleccionamos la opción *local-dev* para escanear los dispositivos conectados al PC.

Seguidamente, seleccionamos el disco donde se encuentran almacenadas las imágenes de copia de sistemas operativos y escogemos la imagen creada en el proceso anterior.

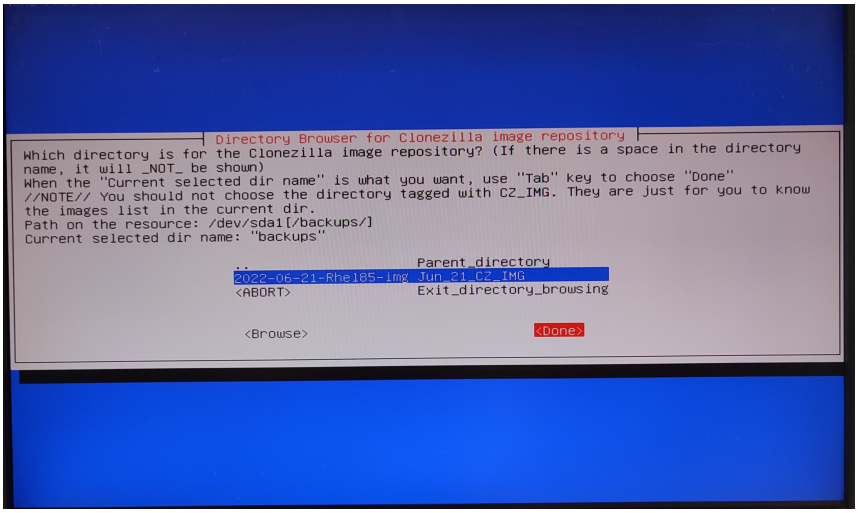

Figura 4.52: Selección imagen copia *Clonezilla*. (Elaboración propia, 2022)

Continuamos eligiendo el tercer parámetro, la opción *restoredisk*, para restaurar la imagen elegida con anterioridad al disco de arranque del sistema. En este punto ya tenemos la imagen que queremos clonar al disco de arranque seleccionada desde sel disco de almacenamiento.

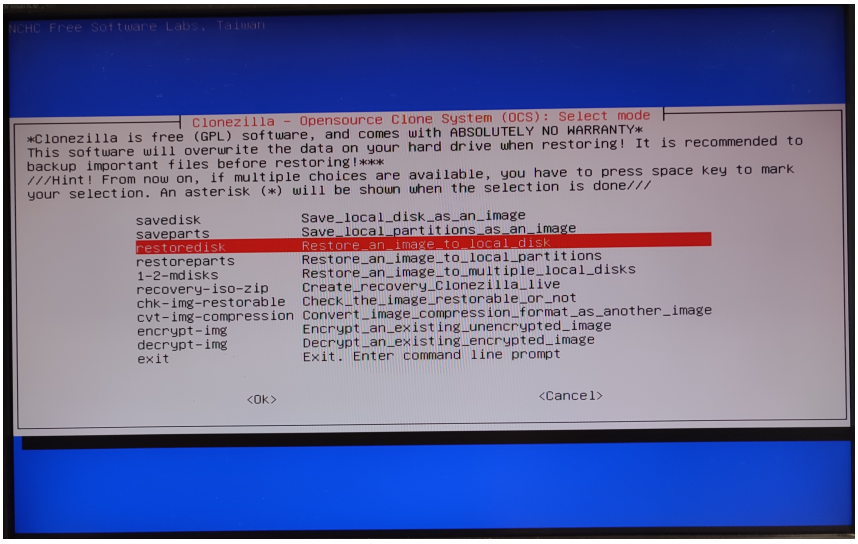

Figura 4.53: Selección restaurar disco *Clonezilla*. (Elaboración propia, 2022)

El siguiente paso es seleccionar el disco al cual queremos inyectar la copia escogida anteriormente, esta elección es crítica de nuevo, pues una mala elección del disco destino, borra por completo este, borrando la información que pudiera contener.

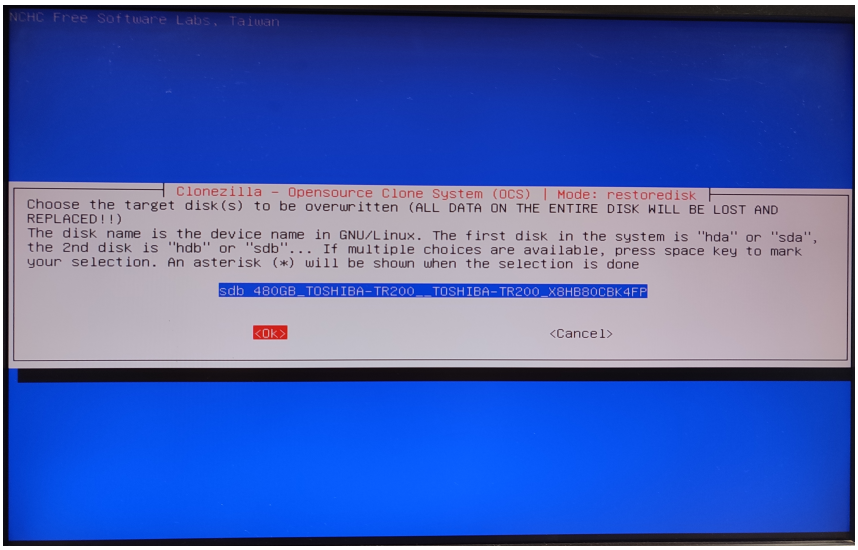

Figura 4.54: Selección disco destino imagen *Clonezilla*. (Elaboración propia, 2022)

Como en nuestro caso, únicamente tenemos un disco al cual copiar la imagen, no hay lugar a dudas y seleccionamos la única posibilidad que nos ofrece *Clonezilla*. Tras este paso, comienza el proceso de restauración de la imagen almacenada en el segundo disco al disco de arranque del sistema.

Al realizar un clonado eficiente y no completo del sistema, la restauración del mismo es inmediata y no requiere un largo tiempo de espera. Como en el anterior proceso, esto depende de la imagen seleccionada y de la velocidad de los discos implicados en dicho desarrollo.

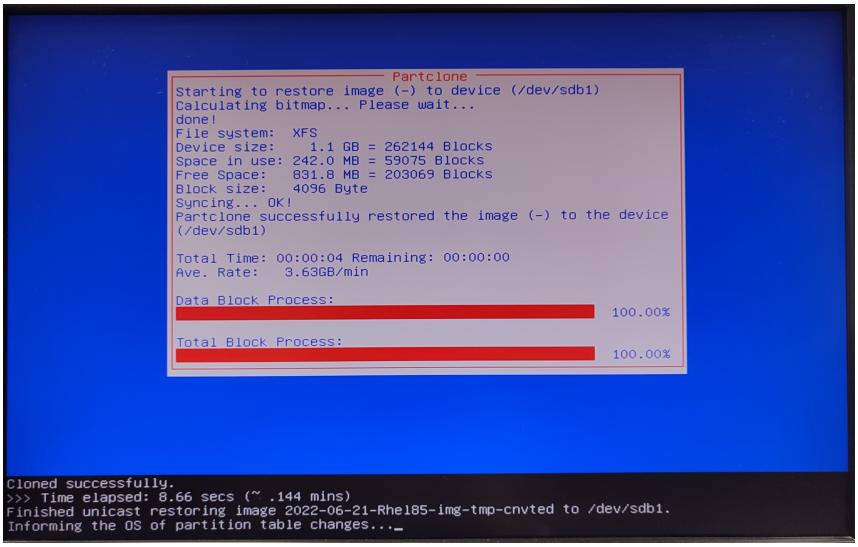

Figura 4.55: Proceso restauración imagen *Clonezilla*. (Elaboración propia, 2022)

Finalmente, tras reiniciar el nodo cliente y como hemos mencionado previamente, modificar el sistema de arranque de la *UEFI* a su estado funcional dentro del laboratorio de pruebas, *Boot Option Priorities* (*Boot Option 1*) ->*Realtek PXE B04 D00*, (*Boot Option 2*) ->*P3: TOSHIBA-TR200*. Observamos en la imagen posterior el sistema operativo *RHEL 8.5* restaurado desde una imagen *Clonezilla*, mediante una unidad *flash USB*.

Figura 4.56: Resultado proceso restauración imagen *Clonezilla*. (Elaboración propia, 2022)

4.5.5. Ejecución de la interfaz *web*

La interfaz web actualmente se lanza desde el PC en el cual se encuentren los dos directorios importantes de la misma, *NGL-API* que es la parte *back-end* de la web y *NGL-FRONT* que es la parte *front-end* de esta. Es una localización puntual pues se pretende incorporar este servicio a un servidor de pruebas interno de la empresa, donde estará en funcionamiento indefinidamente, permitiendo el acceso a los usuarios conectados a la red de Arisnova S.L.

Figura 4.57: Ejecución parte *back-end*. (Elaboración propia, 2022)

Para dar servicio al laboratorio, necesitamos que las dos partes estén en ejecución al mismo tiempo, de lo contrario la interfaz *web* no funcionaría.

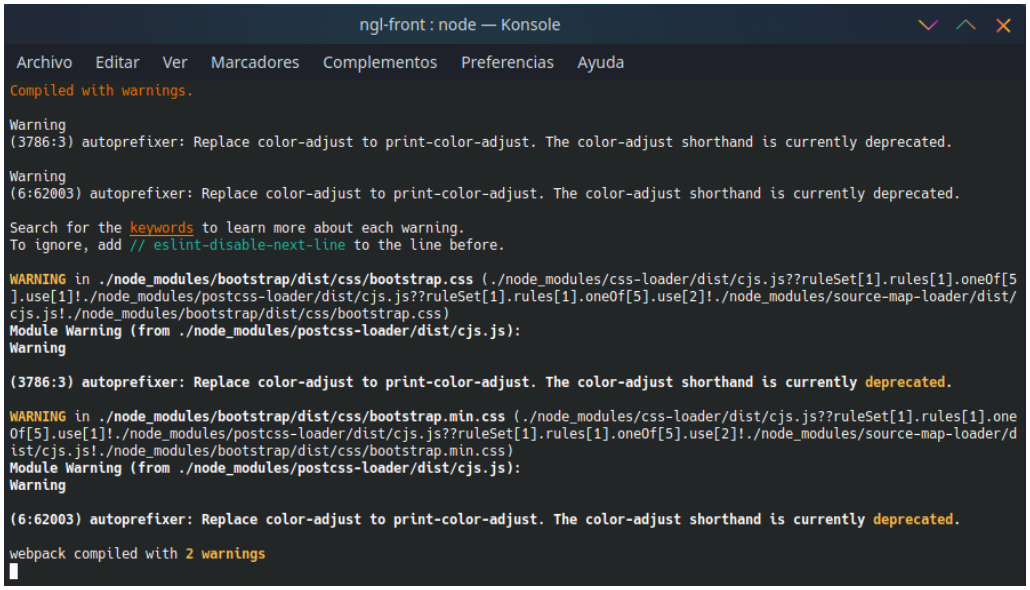

Figura 4.58: Ejecución parte *front-end*. (Elaboración propia, 2022)

CAPÍTULO 5

Implantación

En este apartado evidenciaremos la puesta en marcha de las distintas fases del proyecto, las pruebas realizadas para conseguir y validar el correcto funcionamiento de todas las partes. Mostrando los resultados finales a los que se ha llegado con el desarrollo del laboratorio de acceso remoto multiplataforma para procesos de *Quality Assurance* (*QA*) *Testing*.

5.1 Pruebas

El desarrollo de cada una de las pruebas que hemos llevado a cabo, lo dividimos en secciones para estructurar mejor y poder catalogar las fases que hemos superado para desplegar un laboratorio funcional capaz de cumplir con los objetivos marcados previamente. Se presentarán los ensayos que se han realizado verificando el correcto funcionamiento de nuestro proyecto. Pruebas en la red de laboratorio, en el lanzamiento de sistemas operativos al nodo cliente mediante la terminal previo a la creación de la interfaz *web*.

5.1.1. Validación del correcto funcionamiento de la red del laboratorio

Una vez configurado el laboratorio, debíamos cerciorarnos de si la administración de la red estaba correctamente implementada, pues la gestión del servidor *DHCP* por parte del enrutador *Mikrotik RB2011IL-RM* debía permitir el reparto de direcciones *IP* de forma automática al iniciar un nodo cualquiera en la red. Para llevar a cabo esta prueba, bastaba con iniciar el nodo cliente en modo arranque por red o *PXE* y ver los datos que obtenía del rúter.

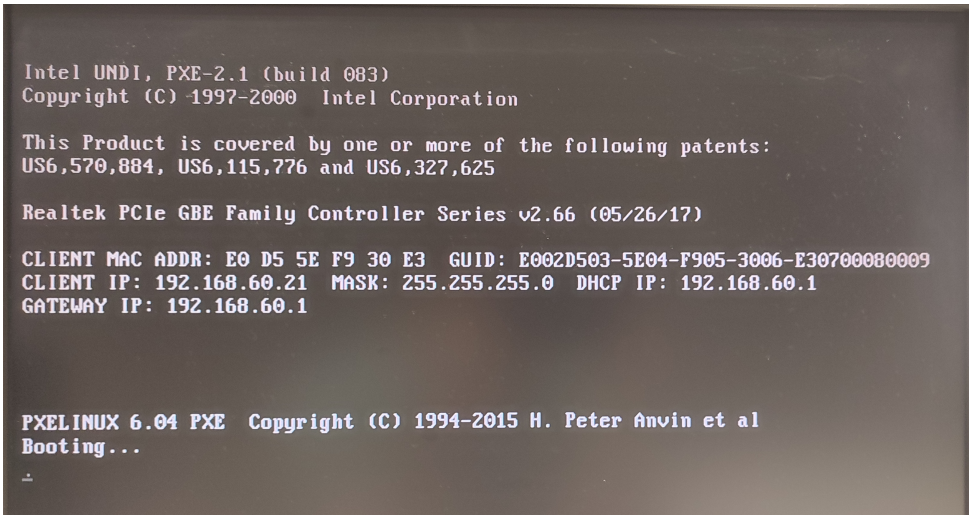

Figura 5.1: Resultado arranque *PXE* del nodo cliente. (Elaboración propia, 2022)

Como vemos en la imagen anterior, el nodo cliente recibe la *IP 192.168.60.21* previamente arrendada a su dirección *MAC*, como vimos en la sección «4.5.1 Configuración del rúter *Mikrotik*», este la recibe del servidor *DHCP* con dirección *IP 192.168.60.1*, el rúter, que además actúa como puerta de enlace. Esta prueba verificaba el correcto funcionamiento del reparto de *IPs* en la red.

Para comprobar que los nodos tenían acceso remoto mediante el protocolo *SSH* desde fuera de la subred del laboratorio, debíamos intentar conectarnos a ellos desde en este caso un equipo externo a la red.

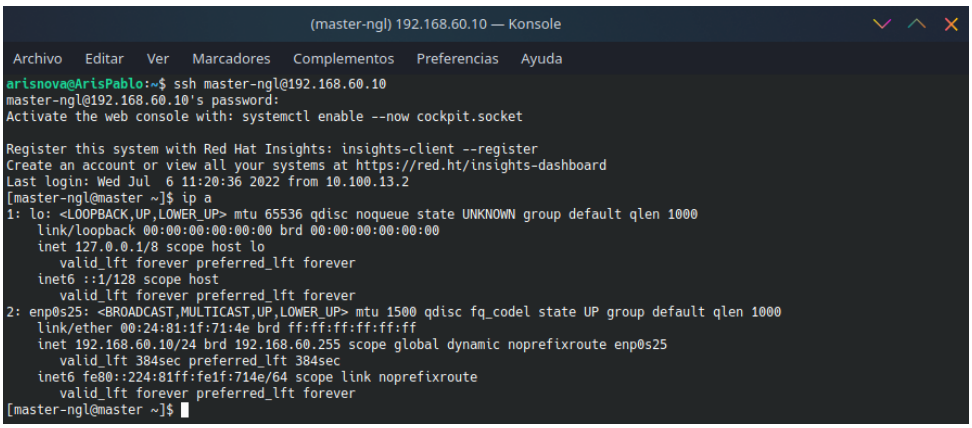

Figura 5.2: Conexión remota *SSH* del nodo máster. (Elaboración propia, 2022)

Mostrado en las imágenes contiguas a este párrafo, observamos como el acceso remoto por *SSH* es correcto y nos permite conectarnos a los nodos máster y cliente, para corroborar su funcionamiento obtenemos las direcciones *IP* de cada uno de ellos. A través de la introducción del comando *ip a* por el terminal, accedemos a todas las interfaces de red de los dispositivos, validando a su vez el reparto de *IPs* por parte del rúter.

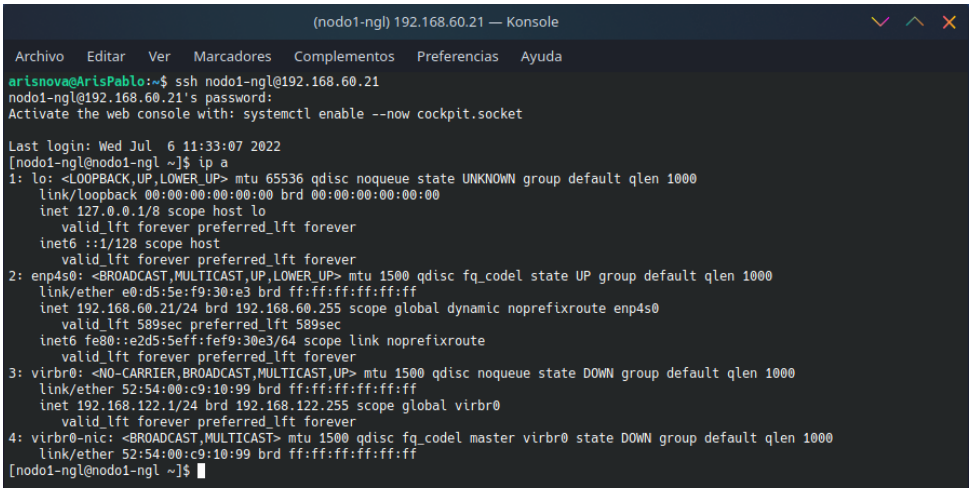

Figura 5.3: Conexión remota *SSH* del nodo cliente. (Elaboración propia, 2022)

Seguidamente debíamos comprobar que los nodos de la red tenían visibilidad entre ellos, esto quiere decir que tienen comunicación directa mediante el enrutador, y verificar su salida a internet, para ello accedimos a cada nodo para teclear los comandos que mostramos a continuación.

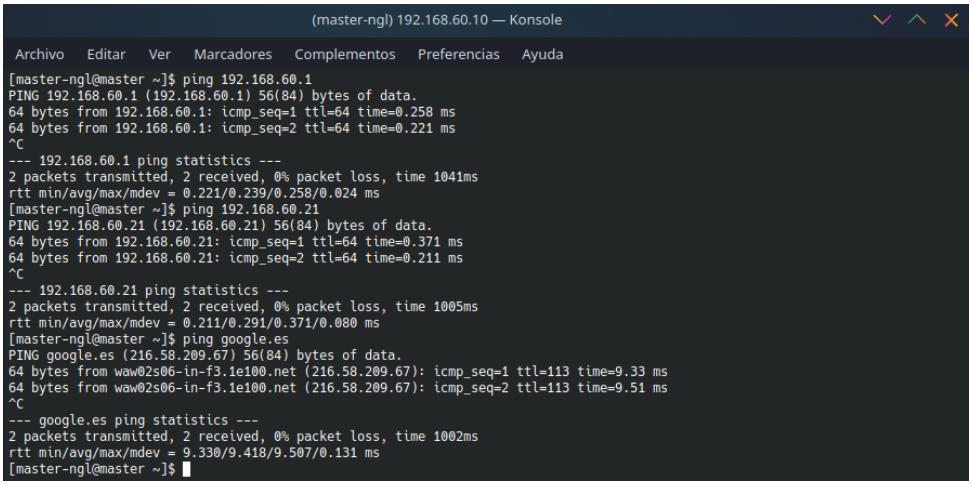

Figura 5.4: Visibilidad dispositivos en red, nodo máster. (Elaboración propia, 2022)

La prueba consiste en a través del comando *ping*, enviar un datagrama por la red y esperar una respuesta, esto comprueba si hay comunicación entre los distintos dispositivos de la red. Realizamos tres comandos *ping* en el nodo máster, *ping 192.168.60.1* comprueba si hay comunicación con la puerta de enlace y el propio rúter, importante para tener salida a *Internet*, *ping 192.168.60.21* verificar comunicación entre nodos, esto valida la transferencia del archivo *pxelinux.0* para la instalación del S.O. por red. Y por último, *ping google.es*, esto confirma la salida a *Internet* pues nos estamos comunicando con los servidores de *Google*, y que el servidor *DNS* del rúter está realizando correctamente la traducción de los nombres de dominio a direcciones *IP*.

Todos los datagramas son recibidos sin errores, esto indica el correcto funcionamiento y visibilidad de los nodos en la red del laboratorio. Para el nodo cliente, realizamos los mismos tres comandos anteriores, pero modificando el segundo *ping* que se lo realizamos al nodo máster, *ping 192.168.60.10*. También validamos la comunicación óptima del PC cliente, como apreciamos en la imagen siguiente.

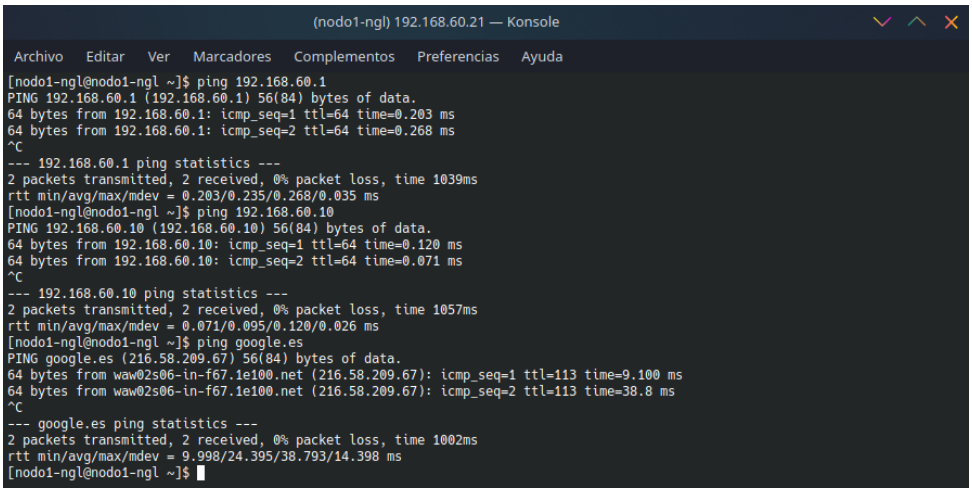

Figura 5.5: Visibilidad dispositivos en red, nodo cliente. (Elaboración propia, 2022)

Por último nos queda corroborar, que el archivo *pxelinux.0* vital para la instalación por red del nodo cliente, se está transfiriendo del PC máster al PC cliente en el momento este inicia un arranque *PXE*. Si este archivo no es transferido desde el máster en el momento un nodo arranca en modo red e implícitamente al protocolo, este le solicita el archivo al rúter que automáticamente redirige la petición al nodo máster, como vimos en la sección «4.5.1 Configuración del rúter *Mikrotik*», sería imposible dicha instalación.

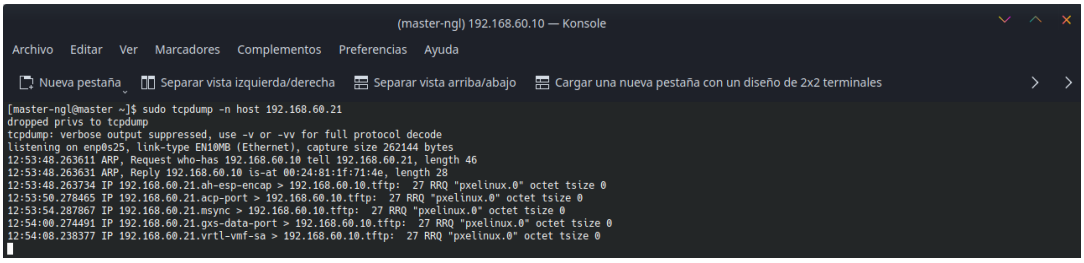

Figura 5.6: Escaneo paquetes *TCP/IP* nodo cliente. (Elaboración propia, 2022)

Utilizando la herramienta *tcpdump*, utilidad de línea de comandos que permite capturar los paquetes *TCP/IP* que pasan a través de una interfaz de red, queríamos comprobar si realmente en el instante que el nodo cliente realizaba un arranque por red, el tráfico *TCP/IP* de dicho PC mostraba la transferencia del archivo *pxelinux.0*. Como se muestra en la imagen previa, el funcionamiento de los componentes de la red es el esperado.

5.1.2. Instalación de un sistema operativo por red

Asegurado el correcto funcionamiento de la red de trabajo del laboratorio, era el momento de realizar pruebas de lanzamiento de los distintos sistemas operativos recopilados en el repositorio de *Cobbler* al nodo cliente. Generados los distintos perfiles que facilitan la configuración durante la instalación del sistema en ejecución mediante los archivos *kickstart*, la siguiente prueba a realizar era la creación del sistema *Cobbler* enlazando los perfiles creados anteriormente con el PC cliente, mediante su dirección *MAC* y nombre de interfaz de la tarjeta de red.

Al no disponer de la interfaz *web* operativa en ese preciso instante, la ejecución de los *tests* se llevó a cabo mediante la introducción de los distintos comandos *Cobbler* por el emulador de terminal. Puesto que las pruebas las hemos realizado nosotros mismos habiendo adquirido una experiencia previa en el manejo del sistema de aprovisionamiento, esto no suponía un problema.

Mencionado en la sección «4.5.2 Configuración del PC máster» únicamente debíamos tratar con la creación y borrado del parámetro sistema en *Cobbler*. Procedimos a la ejecución de los siguientes comandos para las pruebas de instalación por red, en este caso la distribución *Fedora 34 Server* (sin interfaz gráfica). Tras la instalación y reinicio del PC, obtuvimos el resultado esperado.

```
cobbler system add --name="nodo1" --profile="FEDORA34-Nodo1" --interface="enp4s0"
--mac="E0:D5:5E:F9:30:E3" --netboot-enabled="Y"
cobbler sync
```

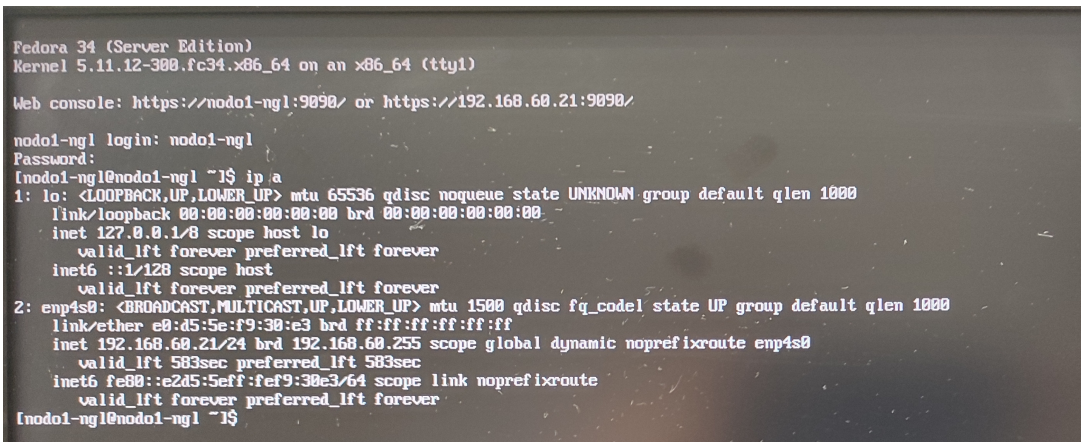

Figura 5.7: Terminal *Fedora Linux 34* en nodo cliente. (Elaboración propia, 2022)

Este mismo procedimiento lo hemos realizado con los distintos sistemas operativos mencionados a lo largo de la memoria. Por poner algún ejemplo más de las pruebas iniciales de instalación, mostramos en las siguientes imagens la distribución *RHEL 7.4* y *Rocky Linux 8.5*, un dato a añadir sobre estas distribuciones es que en los archivos *kickstart* podemos configurar el entorno de lanzamiento, pudiendo instalar la versión *Server*, *Server* (sin interfaz gráfica) o *Workstation*, varía la instalación de paquetes y al tipo de trabajo que van destinadas.

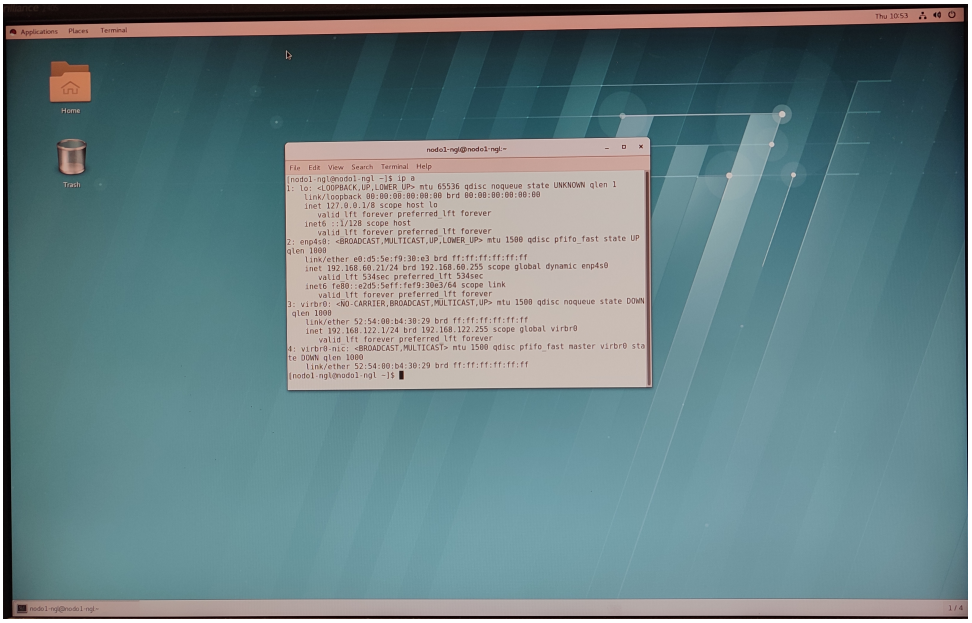

Figura 5.8: Escritorio *RHEL 7.4* en nodo cliente. (Elaboración propia, 2022)

Figura 5.9: Escritorio *Rocky Linux 8.5* en nodo cliente. (Elaboración propia, 2022)

5.2 Resultados

En este apartado demostraremos el resultado final de la instalación de un sistema operativo en el nodo cliente a través de la utilización de la interfaz *web*, finalizando el desarrollo del proyecto dando lugar a una instalación gráfica a través de la red del laboratorio, intuitiva y funcional donde el usuario simplemente tendrá que seleccionar los parámetros que desea en los desplegables de la interfaz *web*.

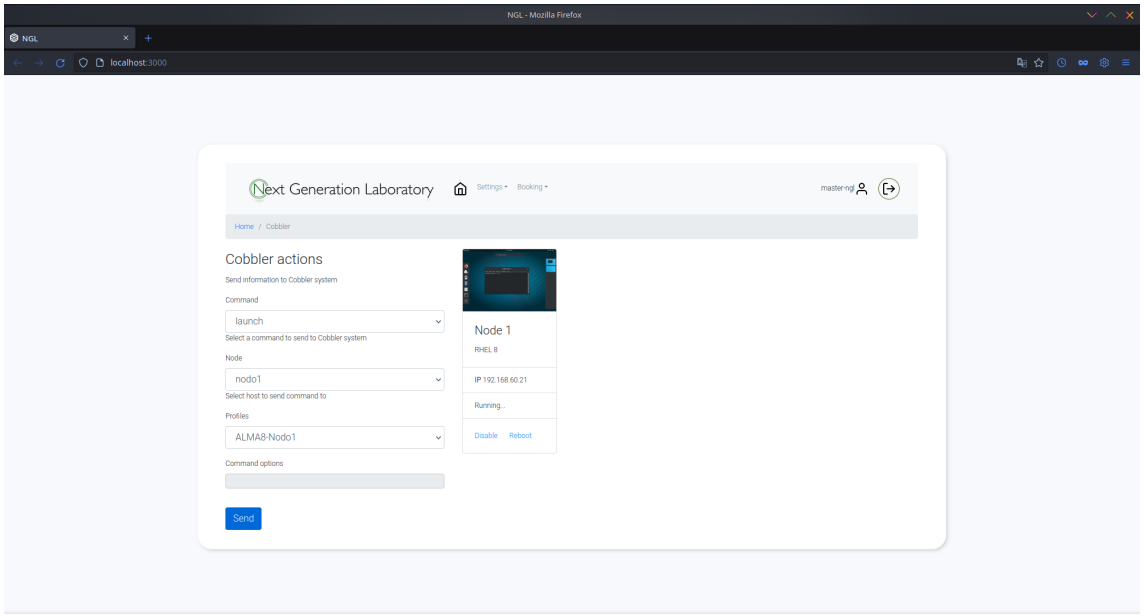

Figura 5.10: Selección de parámetros en la interfaz *web*. (Elaboración propia, 2022)

A través de la interfaz *web* seleccionamos por ejemplo, el perfil *ALMA8-Nodo1*, que contiene el perfil de *Cobbler* enlazado al archivo *kickstart* de *Alma Linux 8.5*, como consecuencia de este paso anterior estamos inyectando en el nodo máster los siguientes comandos *Cobbler* al pulsar el botón *Send*.

```
cobbler system remove --name="nodo1"
cobbler system add --name="nodo1" --profile="ALMA8-Nodo1" --interface="enp4s0"
--mac="E0:D5:5E:F9:30:E3" --netboot-enabled="Y"
cobbler sync
```

Borramos un sistema anterior en caso de haberlo, creamos el sistema enlazado con el perfil seleccionado y sincronizamos para que el sistema de aprovisionamiento automáticamente reescriba el archivo */var/lib/tftpboot/ 01-e0-d5-5e-f9-30-e3*, como mencionamos en la sección «4.5.2 Configuración del PC máster».

Lanzados los comandos al nodo máster, únicamente hay que reiniciar el nodo cliente para comenzar el proceso de instalación por red, actualmente hay dos opciones de realizar el reinicio. Manualmente, acceder al PC y reiniciarlo físicamente o bien conectarse por *SSH* mediante el emulador de terminal y lanzar el comando *reboot*.

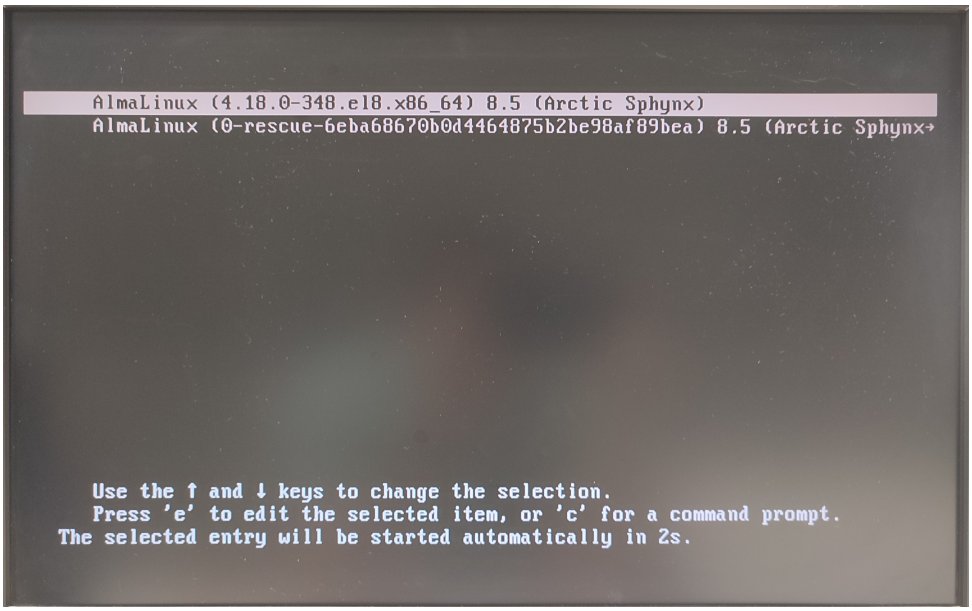

Figura 5.11: Inicio S.O. *Alma Linux 8.5* en nodo cliente. (Elaboración propia, 2022)

Reiniciado el equipo e instalado el S.O. observamos en la imagen anterior cómo detecta el nuevo sistema instalado Alma Linux e inicia desde el disco interno del PC, dando lugar a un sistema operativo completo y funcional en el nodo cliente.

Figura 5.12: Escritorio *Alma Linux 8.5* en nodo cliente. (Elaboración propia, 2022)

CAPÍTULO 6

Conclusiones

Tras la realización del proyecto que hemos llevado a cabo para el trabajo final de máster del Máster Universitario en Ingeniería Informática de la ETSINF, concluimos que el laboratorio desarrollado para este propósito cumple con los objetivos establecidos en el apartado «1.2 Objetivos».

La creación, configuración de la red y del sistema de aprovisionamiento utilizado en el PC máster, para el lanzamiento de sistemas operativos vía *DHCP-PXE* a los PCs cliente y la implementación de un mecanismo de copia del entorno de ejecución en el segundo disco del nodo, añadido a la interfaz *web* que hemos desarrollado para el usuario. Este hecho nos hace estar satisfechos del trabajo realizado y del tiempo empleado para ello. Adquiriendo una experiencia en Arquitectura de Sistemas y en el ámbito laboral, con la que no contábamos previamente.

El resultado del proyecto demuestra que hemos sabido introducir los conocimientos adquiridos durante el máster y grado universitario, gestionando correctamente los problemas que han surgido, administrando los tiempos de cada bloque del proyecto y tratando con tecnologías punteras en el campo de la Arquitectura de Sistemas a nivel profesional, como son la configuración y manejo de un sistema de aprovisionamiento y la creación de una red de trabajo mediante la utilidad *WinBox* de *Mikrotik*.

Adquiriendo experiencia en la gestión de proyectos de un cierto volumen de trabajo ajustándonos casi a la perfección al plan establecido en un principio. Durante el desarrollo del proyecto hemos cometido errores, que afortunadamente se han podido subsanar.

El cambio de versiones de *Cobbler* no fue fácil de gestionar durante un periodo medio de tiempo, en la configuración del rúter realizamos varias modificaciones muy concretas para posibilitar su correcto funcionamiento, tras estos contratiempos queremos mencionar que el laboratorio cuenta con todas las funcionalidades previamente especificadas. Aunque, ocasionaron que nuestra planificación inicial sufriera un retraso de 16 horas o 3'2 jornadas de trabajo, totalmente asumibles.

Sin embargo, se pueden mejorar algunos aspectos de las distribuciones importadas en *Cobbler*, *Ubuntu Linux* actualmente no cuenta con una instalación automatizada, debido a cambios internos en el funcionamiento de las versiones *20.04* en adelante con respecto a los cargadores de arranque. Tampoco hemos incorporado por motivos del proyecto versiones de *Microsoft Windows*. Y por último, debido a la antigüedad de las placas base de los PCs que disponíamos en el laboratorio, no hemos podido implementar el arranque *Wake on LAN* (*WoL*)[40], que hubiera automatizado aún más si cabe el laboratorio.

6.1 Relación del trabajo desarrollado con los estudios cursados

Como previamente hemos mencionado en el apartado «1.1 Motivación», adquirimos los conocimientos básicos relacionados con el proyecto desarrollado, en la asignatura de «Configuración y Optimización de Sistemas de Cómputo», del primer semestre del Máster Universitario en Ingeniería Informática. Hemos logrado ampliar y actualizar los conceptos aprendidos durante los seminarios de la asignatura, los cuales trataban de configurar una máquina virtual que actuara como máster, y generar máquinas virtuales clientes mediante el sistema de aprovisionamiento *Cobbler*.

Los seminarios estaban enfocados a desarrollar conocimientos en *CentOS7*, una distribución que actualmente esta obsoleta, la cual hemos sustituido por la distribución *RHEL 8.5* para el máster. Muy similar a la utilizada en la asignatura mencionada anteriormente, con la ventaja de ser un sistema operativo con soporte y moderno.

Por otro lado, la versión de *Cobbler* utilizada durante la asignatura (2.8), es anterior a la utilizada en este proyecto (3.3.2). Lo cual, a priori no parece un inconveniente, pero las nuevas versiones de *Cobbler*, en concreto desde la 3.2 en adelante, cambian significativamente la configuración de archivos, localización de estos, parámetros de configuración y forma de lanzamiento. Esto, no sólo nos ha hecho investigar para adaptarnos a los cambios, sino que nos ha servido para estar al día de la forma en que trabaja este *software* actualmente.

Para la creación de la red del laboratorio, hemos necesitado los conocimientos adquiridos cursando la asignatura de «Tecnología de Redes» de tercer curso del Grado en Ingeniería Informática. La ampliación de estos conocimientos nos han permitido montar una *LAN* totalmente operativa, sabiendo qué protocolos necesitamos y qué servidores hay que configurar dentro del rúter para otorgar a la red de salida a *Internet*, conectar entre sí todos los componentes y proporcionar la seguridad adecuada a nuestro proyecto.

6.2 Trabajos futuros

Puesto que el proyecto se ha realizado en las instalaciones de Arisnova S.L. Ingeniería de Sistemas, y dicho laboratorio era de interés de la misma, se espera que en un futuro pueda ampliarse el número de PCs cliente, actualmente sólo contamos con uno. Incrementar a su vez las distribuciones de sistemas operativos que puedan instalarse en los respectivos nodos, como por ejemplo, incorporar versiones de *Microsoft Windows* al catálogo y aumentar la automatización de las versiones de *Ubuntu Linux*, generar un archivo *ISO* modificado a partir de un original e incorporarlo al repositorio de *Cobbler*.

Otra faceta a mejorar, sería la funcionalidad y opciones que la interfaz *web* nos ofrece actualmente, como por ejemplo, tener en la tarjeta de cada nodo, información del sistema en ejecución en tiempo real. Todo esto aportaría al proyecto un nivel mayor de funcionalidad en cuanto al fin que este persigue, que no es otro que el testeo multiplataforma para procesos de *QA* del código generado por los desarrolladores de *software* de la empresa.

Del mismo modo, si se consiguiera lo mencionado en los párrafos anteriores, podría plantearse considerar el laboratorio como un producto más de los que la empresa ofrece a sus clientes, este hecho supondría estandarizar los tipos o modelos de PCs y versiones de la placa base, *GPU* y *BIOS/UEFI* a un determinado número de equipos para facilitar la instalación e integración de los PCs entre sí.

Bibliografía

[1] Ciclo de vida del desarrollo de software, emitido el 8 de febrero de 2019. Disponible en: https://ungoti.com/es/soluciones/desarrollo-de-software/sdlc/

[2] Arisnova S.L. Ingeniería de Sistemas, emitido el 30 de mayo de 2017. Disponible en: http://www.arisnova.com/

[3] *Quality assurance*, emitido el 7 de junio de 2022. Disponible en: https://en.wikipedia.org/wiki/Quality_assurance

[4] *ISO 9000:2015 Quality management systems*, emitido en septiembre de 2021. Disponible en: https://www.iso.org/obp/ui/es/#iso:std:iso:9000:ed-4:v1:es

[5] *Welcome to Cobbler's documentation!*, emitido el 3 de junio de 2022. Disponible en: https://cobbler.readthedocs.io/en/v3.3.2/index.html

[6] *Product specifications*, emitido el 27 de enero de 2018. Disponible en: https://mikrotik.com/product/RB2011iL-RM

[7] Entorno de ejecución de prearranque, emitido el 10 de enero de 2022. Disponible en: https://es.wikipedia.org/wiki/Entorno_de_ejecuci%C3%B3n_de_prearranque

[8] *What is Clonezilla?*, emitido el 24 de mayo de 2022. Disponible en: https://clonezilla.org/

[9] *The 10 Best QA Automation Tools For Software Testing In 2022*, emitido el 2 de enero de 2022. Disponible en: https://theqalead.com/tools/qa-automation-tools/

[10] *Types of software testing*, emitido en agosto de 2018. Disponible en: https://www.ibm.com/topics/software-testing

[11] *OS Installation Automation with cobbler and puppet*, emitido el 13 de febrero de 2015. Disponible en: https://www.youtube.com/watch?v=snBSq8OrYEY

[12] *Free and Open Source Systems Management*, emitido el 31 de mayo de 2020. Disponible en: https://spacewalkproject.github.io/

[13] *Build the future of Bare metal infrastructure*, emitido el 20 de junio de 2022. Disponible en: https://www.xcat.org/

[14] *Download the latest RouterOS now!*, emitido el 16 de noviembre de 2020. Disponible en: https://mikrotik.com/software

[15] Programa *Red Hat Enterprise Linux Developer*, emitido el 3 de abril de 2020. Disponible en: https://www.redhat.com/es/technologies/linux-platforms/enterprise-linux/developer-program

[16] *Free. Built on open source. Runs everywhere*, emitido el 3 de noviembre de 2021. Disponible en: https://code.visualstudio.com/

[17] *Mikrotik pruducts*, emitido el 4 de agosto de 2021. Disponible en: https://mikrotik.com/products/group/ethernet-routers

[18] Hazlo *Linux*, emitido el 17 de mayo de 2021. Disponible en: https://acortar.link/7pa5NZ

[19] *'dd' command in Linux*, emitido el 10 de abril de 2022. Disponible en: https://www.geeksforgeeks.org/dd-command-linux/

[20] Comandos de *Linux mkisofs*, emitido en de octubre de 2016. Disponible en: http://www.w3big.com/es/linux/linux-comm-mkisofs.html

[21] *Create Recovery Clonezilla*, emitido el 14 de noviembre de 2020. Disponible en: https://acortar.link/XE1lo9

[22] *Visual Studio IntelliCode*, emitido el 24 de julio de 2018. Disponible en: https://marketplace.visualstudio.com/items?itemName=VisualStudioExptTeam.vscodeintellicode

[23] *Node npm*, emitido el 17 de noviembre de 2015. Disponible en: https://marketplace.visualstudio.com/items?itemName=eg2.vscode-npm-script

[24] *VS Code ES7+ React/Redux/React-Native/JS snippets*, emitido el 7 de junio de 2017. Disponible en: https://marketplace.visualstudio.com/items?itemName=dsznajder.es7-react-js-snippets

[25] *Go for Visual Studio Code*, emitido el 15 de octubre de 2015. Disponible en: https://marketplace.visualstudio.com/items?itemName=golang.Go

[26] *Mikrotik RB2011IL-RM Ethernet* Negro, emitido el 15 de diciembre de 2019. Disponible en: https://acortar.link/njGxXG

[27] *PcCom Gold Intel Core i7-11700K/16GB/1TB SSD/RTX 3070*, emitido el 1 de febrero de 2022. Disponible en: https://www.pccomponentes.com/pccom-gold-intel-core-i7-11700k-16gb-1tb-ssd-rtx-3070

[28] *PcCom Bronze SP Intel Core i5-10400F/8GB/480GB/SSD/GTX1050Ti*, emitido el 14 de diciembre de 2021. Disponible en: https://www.pccomponentes.com/pccom-e-sports

[29] Cable de Red *Ethernet Cat.6a-1m*, emitido el 13 de febrero de 2021. Disponible en: https://acortar.link/397iEB

[30] Monitor *Samsung LF24T350FHRXEN 24"LED IPS FullHD FreeSync*, emitido el 23 de abril de 2022. Disponible en: https://www.pccomponentes.com/samsung-lf24t350fhrxen-24-led-ips-fullhd-freesync

[31] Disco Duro *SSD 2.5"SATA WD Blue 1TB 3D NAND*, emitido el 5 de enero de 2021. Disponible en: https://acortar.link/cn7z8P

[32] BOLETÍN OFICIAL DEL ESTADO. Ministerio de trabajo, migraciones y seguridad social. Sección III. Otras disposiciones, emitido el 18 de octubre de 2018. Disponible en: https://www.boe.es/eli/es/res/2019/10/07/(8)/dof/spa/pdf

[33] *Ubuntu 20.04 Focal does not install automatically*, emitido el 6 de junio de 2022. Disponible en: https://github.com/cobbler/cobbler/discussions/3143

[34] *The requested URL returned error: 500 Internal Server Error*, emitido el 24 de mayo de 2022. Disponible en: https://github.com/cobbler/cobbler/issues/3114

[35] *Hardware Compatibility*, emitido el 18 de abril de 2020. Disponible en: https://wiki.syslinux.org/wiki/index.php?title=Hardware_Compatibility#LOCALBOOT

[36] *SYSLINUX*, emitido el 14 de abril de 2020. Disponible en: https://wiki.syslinux.org/wiki/index.php?title=SYSLINUX#LOCALBOOT_type

[37] *Comboot/chain.c32*, emitido el 10 de junio de 2019. Disponible en: https://wiki.syslinux.org/wiki/index.php?title=Comboot/chain.c32#UEFI

[38] *PXE installation RHEL8.5 does not automatically start the O.S*, emitido el 31 de mayo de 2022. Disponible en: https://github.com/cobbler/cobbler/discussions/3122

[39] *Clonezilla Live - A small bootable GNU/Linux distribution for imaging and cloning*, emitido el 24 de mayo de 2022. Disponible en: https://clonezilla.org/clonezilla-live.php

[40] Qué es el *Wake on LAN*, emitido el 9 de noviembre de 2020. Disponible en: https://www.testdevelocidad.es/redes/wake-on-lan-utilizarlo/

APÉNDICE A

Configuración del sistema

En este apéndice vamos a presentar los distintos pasos a seguir para realizar una correcta instalación y configuración de *Cobbler 3.3.2*. Seguidamente, mostraremos el fichero de configuración *settings.yml*, relevante para la configuración y buen funcionamiento de nuestro laboratorio de pruebas. Finalmente, expondremos todos los archivos *kickstart* utilizados durante el proceso de instalación de las distintas distribuciones *Linux* en el nodo cliente.

A.1 Instalación de *Cobbler* en el PC máster

Pasos a realizar para la instalación y configuración de un servidor de aprovisionamiento *Cobbler 3.3.2* en un PC cualquiera, con una distribución *RHEL 8.5*. En esta configuración, se introducen direcciones *IP* correspondientes a la subred del laboratorio. Se han extraído del código contraseñas e información sensible que pudiese comprometer la seguridad del proyecto.

```
1 ##### PARA REGISTRAR EL S.O. EN RED HAT #####
2 sudo subscription-manager register --username ((username)) --password
      ((password)) --auto-attach --force
3 ##### ACTUALIZAR SISTEMA #####
4 sudo yum update
5 ##### EPEL-RELEASE #####
6 sudo yum install -y https://dl.fedoraproject.org/pub/epel/epel-release-
      latest-8.noarch.rpm
7 ##### INSTALACION COBBLER #####
8 sudo yum install -y syslinux pykickstart fence-agents rsync rsync-daemon
      http* tftp tftp-server
```

```
sudo dnf module enable cobbler:3.3
sudo yum install -y cobbler debmirror python3-hivex
sudo systemctl start cobblerd
sudo systemctl start tftp
sudo systemctl start rsyncd
sudo systemctl start httpd
sudo systemctl enable cobblerd
sudo systemctl enable tftp
sudo systemctl enable rsyncd
sudo systemctl enable httpd
######### ESTE PASO NO HACE FALTA #########
  sudo touch /etc/xinetd.d/tftp
  sudo nano /etc/xinetd.d/tftp
  ######## INTRODUCIMOS EL SIGUIENTE CODIGO ########
  service tftp {
    socket_type     = dgram
    protocol        = udp
    wait            = yes
    user            = root
    server          = /usr/sbin/in.tftpd
    server_args     = -s /var/lib/tftpboot
    disable         = no
    per_source      = 11
    cps             = 100 2
    flags           = IPv4
  }
  ################################################
################################################
sudo cp -v /usr/lib/systemd/system/tftp.service /etc/systemd/system/tftp-
    server.service
sudo cp -v /usr/lib/systemd/system/tftp.socket /etc/systemd/system/tftp-
    server.socket
sudo nano /etc/systemd/system/tftp-server.service
######## MODIFICAMOS EL SIGUIENTE CODIGO ########
  [Unit]
  Description=Tftp Server
  Requires=tftp-server.socket
  Documentation=man:in.tftpd
  [Service]
  ExecStart=/usr/sbin/in.tftpd -c -p -s /var/lib/tftpboot
  StandardInput=socket
  [Install]
  WantedBy=multi-user.target
```

```
  Also=tftp-server.socket
################################################
sudo nano /etc/systemd/system/tftp-server.socket
######## MODIFICAMOS EL SIGUIENTE CODIGO ########
  [Unit]
  Description=Tftp Server Activation Socket
  [Socket]
  ListenDatagram=69
  BindIPv6Only=both
  [Install]
  WantedBy=sockets.target
################################################
sudo systemctl start tftp-server
sudo systemctl enable tftp-server
sudo setsebool -P tftp_anon_write 1 ## ACCESO DE ESCRITURA AL SERVIDOR TFTP
sudo chmod 777 /var/lib/tftpboot ## PERMISOS RWX EN EL DIRECTORIO RAIZ TFTP
##### GENERAR CLAVE PUBLICA, COPIARLA AL ARCHIVO DE CLAVES #####
sudo ssh-keygen ## DARLE ENTER DOS VECES PARA CREAR
sudo cp /root/.ssh/id_rsa.pub /root/.ssh/authorized_keys
openssl passwd -1
##### INTRODUCIMOS LA CLAVE DE USUARIO Y COPIAMOS LA CLAVE SSL#####
sudo nano .bashrc
######## INTRODUCIMOS EL SIGUIENTE PARAMETRO ########
# User specific aliases and functions
export PASSWD_NGL="passwd_user" ## ESCAPAR "'\'simbolos" PARA DETECTARLOS
source .bashrc
echo $PASSWD_NGL ## COMPROBAMOS QUE SE HA INTRODUCIDO BIEN
openssl passwd -1 $PASSWD_NGL ## COPIAMOS LA CLAVE SSL
sudo nano /etc/cobbler/settings.yaml
######## MODIFICAMOS LOS SIGUIENTES PARAMETROS ########
  default_password_crypted: "passwd_ssl" ## PEGAMOS LA CLAVE SSL
  manage_dhcp: false ## QUEREMOS QUE LO HAGA EL ROUTER
  manage_dns: false ## QUEREMOS QUE LO HAGA EL ROUTER
  pxe_just_once: true
  next_server_v4: 192.168.60.10 ## IP DEL MASTER
  next_server_v6: ::ffff:c0a8:3c0a ## IPv6 DEL MASTER
  server: 192.168.60.10 ## IP DEL MASTER
########################################################
######### ESTE PASO NO HACE FALTA, SE ENCARGA EL ROUTER #########
  sudo nano /etc/cobbler/dhcp.template
  ##### MODIFICAMOS LOS SIGUIENTES PARAMETROS #####
  subnet 192.168.60.0 netmask 255.255.255.0 {
    option routers             192.168.60.10;
```

```
93      option domain-name-servers 192.168.60.10;
94      option subnet-mask         255.255.255.0;
95      range dynamic-bootp        192.168.60.21 192.168.60.29;
96      default-lease-time         21600;
97      max-lease-time             43200;
98      next-server                192.168.60.10;
99    ##################################################
100   sudo nano /etc/cobbler/dnsmasq.template
101   ##### MODIFICAMOS LOS SIGUIENTES PARAMETROS #####
102     dhcp-range=192.168.60.21,192.168.60.29
103     dhcp-option=66,192.168.60.10
104   ##################################################
105   sudo nano /etc/dhcp/dhcpd.conf
106   ##### MODIFICAMOS LOS SIGUIENTES PARAMETROS #####
107     subnet 192.168.60.0 netmask 255.255.255.0 {
108       option routers             192.168.60.10;
109       option domain-name-servers 192.168.60.10;
110       option subnet-mask         255.255.255.0;
111       range dynamic-bootp        192.168.60.21 192.168.60.29;
112       default-lease-time         21600;
113       max-lease-time             43200;
114       next-server                192.168.60.10;
115   ##################################################
116 ######################################################################
117 sudo nano /etc/debmirror.conf
118 ##### COMENTAMOS LOS SIGUIENTES PARAMETROS #####
119   # @dists="sid";
120   # @arches="i386";
121 ################################################
122 cd /var/lib/cobbler
123 sudo chmod 664 web.ss
124 ls -l
125 ####### DEBERIA TENER LOS SIGUIENTES PERMISOS #######
126 -rw-rw-r--.  1 apache root  1024 Apr  6 10:53 web.ss
127 ######################################################
128 ######### ESTE PASO NO HACE FALTA, SE ENCARGA EL ROUTER #########
129   ##### EJECUTAMOS LOS SIGUIENTES COMANDOS #####
130   sudo firewall-cmd --add-service=http --permanent
131   sudo firewall-cmd --add-service=https --permanent
132   sudo firewall-cmd --add-service=dhcp --permanent
133   sudo firewall-cmd --add-port=69/tcp --permanent
134   sudo firewall-cmd --add-port=69/udp --permanent
135   sudo firewall-cmd --add-port=4011/udp --permanent
```

```
136   sudo firewall-cmd --reload
137 ###################################################################
138 sudo systemctl stop firewalld
139 sudo systemctl disable firewalld
140 ############### PONERLO POR SI A CASO ###############
141 sudo setsebool -P cobbler_can_network_connect 1
142 sudo setsebool -P httpd_can_network_connect_cobbler 1
143 sudo restorecon -R /var/lib/cobbler
144 sudo chcon -R -t cobbler_var_lib_t /etc/dnsmasq.conf ## NO PONER ESTE
145 sudo touch /etc/ethers
146 sudo chcon -R -t cobbler_var_lib_t /etc/ethers
147 ######################################################
148 sudo cp /usr/share/syslinux/pxelinux.0 /var/lib/cobbler/loaders/
149 sudo cp /usr/share/syslinux/menu.c32 /var/lib/cobbler/loaders/
150 ##### IMPORTANTE PARA UN BUEN FUNCIONAMIENTO #####
151   sudo nano /etc/selinux/config
152   ##### MODIFICAMOS EL SIGUIENTE PARAMETRO #####
153   SELINUX=permissive
154   ################################################
155   sudo getenforce
156   Permissive
157 ####################################################
158 sudo dnf install dnf-plugins-core
159 sudo dnf install yum-utils
160 sudo systemctl restart cobblerd
161 sudo cobbler check
162 sudo cobbler sync ## TASK COMPLETE!!
```

A.2 Archivos de configuración *Cobbler*

El archivo de configuración *settings.yml* es, en el cual, se enlazan los distintos servicios con los dispositivos que los sirven, dejando atada la conexión entre los servidores *DHCP* y *DNS* del rúter con el PC máster. En esta configuración, se introducen direcciones *IP* correspondientes a la subred del laboratorio. Se han extraído del código contraseñas e información sensible que pudiese comprometer la seguridad del proyecto.

```
# Cobbler settings file
# Docs for this file can be found at:
# https://cobbler.readthedocs.io/en/latest/cobbler-conf.html

allow_duplicate_hostnames: false
allow_duplicate_ips: false
allow_duplicate_macs: false
allow_dynamic_settings: false
always_write_dhcp_entries: false
anamon_enabled: false
auth_token_expiration: 3600
authn_pam_service: login
auto_migrate_settings: false
autoinstall: default.ks
autoinstall_snippets_dir: /var/lib/cobbler/snippets
autoinstall_templates_dir: /var/lib/cobbler/templates
bind_chroot_path: ''
bind_manage_ipmi: true
bind_master: 127.0.0.1
bind_zonefile_path: /var/named
boot_loader_conf_template_dir: /etc/cobbler/boot_loader_conf
bootloaders_dir: /var/lib/cobbler/loaders
bootloaders_formats:
IA64: {binary_name: bootia64.efi}
aarch64: {binary_name: grubaa64.efi}
arm: {binary_name: bootarm.efi}
arm64-efi:
binary_name: grubaa64.efi
extra_modules: [efinet]
i386-efi: {binary_name: bootia32.efi}
i386-pc-pxe:
binary_name: grub.0
extra_modules: [chain, pxe, biosdisk]
```

```
mod_dir: i386-pc
i686: {binary_name: bootia32.efi}
powerpc-ieee1275:
binary_name: grub.ppc64le
extra_modules: [net, ofnet]
x86_64-efi:
binary_name: grubx86.efi
extra_modules: [chain, efinet]
bootloaders_ipxe_folder: /usr/share/ipxe/
bootloaders_modules: [btrfs, ext2, xfs, jfs, echo, all_video, boot, cat,
    configfile, normal, fat, font, gfxmenu, gfxterm, gzio, halt, iso9660,
    jpeg, linux, loadenv, minicmd, reiserfs, part_apple, part_gpt,
    part_msdos, password_pbkdf2, png, reboot, search, search_fs_file,
    search_fs_uuid, search_label, sleep, test, 'true', video, mdraid09,
    mdraid1x, lvm, serial, regexp, tr, tftp, http, luks, gcry_rijndael,
    gcry_sha1, gcry_sha256]
bootloaders_shim_file: shim[a-zA-Z0-9]*\.efi
bootloaders_shim_folder: /boot/efi/EFI/*/
build_reporting_email: [root@localhost]
build_reporting_enabled: false
build_reporting_ignorelist: []
build_reporting_sender: ''
build_reporting_smtp_server: localhost
build_reporting_subject: ''
buildisodir: /var/cache/cobbler/buildiso
cheetah_import_whitelist: [random, re, time, netaddr]
client_use_https: false
client_use_localhost: false
cobbler_master: ''
convert_server_to_ip: false
createrepo_flags: -c cache -s sha
default_name_servers: []
default_name_servers_search: []
default_ownership: [admin]
default_password_crypted: "PASSWORD_CRYPTED".
default_template_type: cheetah
default_virt_bridge: xenbr0
default_virt_disk_driver: raw
default_virt_file_size: 5
default_virt_ram: 512
default_virt_type: xenpv
enable_ipxe: false
enable_menu: true
```

```
grub2_mod_dir: /usr/lib/grub
grubconfig_dir: /var/lib/cobbler/grub_config
http_port: 80
include: [/etc/cobbler/settings.d/*.settings]
iso_template_dir: /etc/cobbler/iso
jinja2_includedir: /var/lib/cobbler/jinja2
kernel_options: {}
ldap_anonymous_bind: true
ldap_base_dn: DC=example,DC=com
ldap_port: 389
ldap_search_bind_dn: ''
ldap_search_passwd: ''
ldap_search_prefix: uid=
ldap_server: ldap.example.com
ldap_tls: true
ldap_tls_cacertdir: ''
ldap_tls_cacertfile: ''
ldap_tls_certfile: ''
ldap_tls_cipher_suite: ''
ldap_tls_keyfile: ''
ldap_tls_reqcert: ''
manage_dhcp: false
manage_dhcp_v4: false
manage_dhcp_v6: false
manage_dns: false
manage_forward_zones: []
manage_genders: false
manage_reverse_zones: []
manage_rsync: false
manage_tftpd: true
mgmt_classes: []
mgmt_parameters: {from_cobbler: true}
next_server_v4: 192.168.60.10
next_server_v6: ::ffff:c0a8:3c0a
nopxe_with_triggers: true
nsupdate_enabled: false
nsupdate_log: /var/log/cobbler/nsupdate.log
nsupdate_tsig_algorithm: hmac-sha512
nsupdate_tsig_key: [cobbler_update_key., "KEY"]
power_management_default_type: ipmilanplus
proxy_url_ext: ''
proxy_url_int: ''
puppet_auto_setup: false
```

```
puppet_parameterized_classes: true
puppet_server: ''
puppet_version: 2
puppetca_path: /usr/bin/puppet
pxe_just_once: true
redhat_management_key: ''
redhat_management_permissive: false
redhat_management_server: xmlrpc.rhn.redhat.com
register_new_installs: false
remove_old_puppet_certs_automatically: false
replicate_repo_rsync_options: -avzH
replicate_rsync_options: -avzH
reposync_flags: --newest-only --delete --refresh --remote-time
reposync_rsync_flags: -rltDv --copy-unsafe-links
restart_dhcp: true
restart_dns: true
run_install_triggers: true
samba_distro_share: DISTRO
scm_push_script: /bin/true
scm_track_author: cobbler <cobbler@localhost>
scm_track_enabled: false
scm_track_mode: git
serializer_pretty_json: false
server: 192.168.60.10
sign_puppet_certs_automatically: false
signature_path: /var/lib/cobbler/distro_signatures.json
signature_url: https://cobbler.github.io/signatures/3.0.x/latest.json
syslinux_dir: /usr/share/syslinux
syslinux_memdisk_folder: /usr/share/syslinux
syslinux_pxelinux_folder: /usr/share/syslinux
tftpboot_location: /var/lib/tftpboot
virt_auto_boot: true
webdir: /var/www/cobbler
webdir_whitelist: [misc, web, webui, localmirror, repo_mirror,
    distro_mirror, images, links, pub, repo_profile, repo_system, svc,
    rendered, .link_cache]
windows_enabled: false
windows_template_dir: /etc/cobbler/windows
xmlrpc_port: 25151
yum_distro_priority: 1
yum_post_install_mirror: true
yumdownloader_flags: --resolve
```

A.3 Archivos *kickstart* utilizados

Los archivos *kickstart* son vitales para una configuración automatizada en el proceso de instalación de un nodo mediante arranque *PXE-DHCP*. En esta configuración, se introducen direcciones *IP* correspondientes a la subred del laboratorio. Se han extraído del código contraseñas e información sensible que pudiese comprometer la seguridad del proyecto.

Distribución *Alma Linux 8.5*, archivo *alma8.ks*:

```
#version=ALMA8
# Deshabilitamos el cortafuegos
firewall --disabled
# Medio de arranque por red
url --url="http://192.168.60.10/cblr/links/ALMA8-x86_64/"
# Root password
rootpw --iscrypted "ROOT-PASSWORD"
# Network information
network --bootproto=static --ip=192.168.60.21 --netmask=255.255.255.0 --
    gateway=192.168.60.1 --nameserver=192.168.60.1,8.8.8.8 --device=enp0s25
    --hostname=nodo1-ngl --onboot=on --noipv6 --activate
# Tipo de encriptacion CentOS8/RHEL8
authselect --enableshadow --passalgo=sha512
# Use text install
text
# Keyboard layouts
keyboard --vckeymap=us --xlayouts='us'
# System language
lang en_US.UTF-8
# Configuracion SElinux
selinux --disabled
# System timezone
timezone Europe/Madrid --isUtc --nontp
# Erase disk
zerombr
bootloader --location=mbr --append="biosdevname=0" --boot-drive=sdb
    --driveorder=sdb,sda
# Partition clearing information
clearpart --initlabel --all --drives=sdb
autopart --type=lvm
# Run the Setup Agent on first boot
firstboot --enable
```

```
# Disk configuration section
ignoredisk --only-use=sdb
# Do not configure the X Window System
xconfig  --startxonboot
# Accept the license agreement
eula --agreed
# SSH connection
sshkey --username=root "SSH-KEY"
#Initial user (user with sudo capabilities)
user --groups=wheel --name=NAME --password=PASSWORD
reboot
#Preinstalacion
%pre
$SNIPPET('autoinstall_start')
%end
#Paquetes
%packages
@^workstation-product-environment
kexec-tools
yum-utils
%end
%addon com_redhat_kdump --enable --reserve-mb='auto'
%end
%anaconda
pwpolicy root --minlen=6 --minquality=1 --notstrict --nochanges --notempty
pwpolicy user --minlen=6 --minquality=1 --notstrict --nochanges --emptyok
pwpolicy luks --minlen=6 --minquality=1 --notstrict --nochanges --notempty
%end
#Postinstalacion
%post
$SNIPPET('autoinstall_done')
sed -i -e '/%sudo/s/ALL$/NOPASSWD:ALL/' /target/etc/sudoers
%end
```

Distribución *Fedora Linux 34*, archivo *fedora34.ks*:

```
#version=FEDORA34
# Deshabilitamos el cortafuegos
firewall --disabled
# Medio de arranque por red
url --url="http://192.168.60.10/cblr/links/FEDORA34-x86_64/"
# Root password
```

```
rootpw --iscrypted "ROOT-PASSWORD"
# Network information
network --bootproto=static --ip=192.168.60.21 --netmask=255.255.255.0 --
    gateway=192.168.60.1 --nameserver=192.168.60.1,8.8.8.8 --device=enp0s25
    --hostname=nodo1-ngl --onboot=on --noipv6 --activate
# Tipo de encriptacion CentOS8/RHEL8
authselect --enableshadow --passalgo=sha512
# Use text install
text
# Keyboard layouts
keyboard --vckeymap=us --xlayouts='us'
# System language
lang en_US.UTF-8
# Configuracion SElinux
selinux --disabled
# System timezone
timezone Europe/Madrid
# Erase disk
zerombr
bootloader --location=mbr --append="biosdevname=0" --boot-drive=sdb
    --driveorder=sdb,sda
# Partition clearing information
clearpart --initlabel --all --drives=sdb
autopart --type=lvm
# Run the Setup Agent on first boot
firstboot --enable
# Disk configuration section
ignoredisk --only-use=sdb
# Do not configure the X Window System
xconfig  --startxonboot
# SSH connection
sshkey --username=root "SSH-KEY"
#Initial user (user with sudo capabilities)
user --groups=wheel --name=NAME --password=PASSWORD
reboot
#Preinstalacion
%pre
$SNIPPET('autoinstall_start')
%end
#Paquetes
%packages
@^server-product-environment
%end
```

```
%addon com_redhat_kdump --enable --reserve-mb='auto'
%end
%anaconda
pwpolicy root --minlen=6 --minquality=1 --notstrict --nochanges --notempty
pwpolicy user --minlen=6 --minquality=1 --notstrict --nochanges --emptyok
pwpolicy luks --minlen=6 --minquality=1 --notstrict --nochanges --notempty
%end
#Postinstalacion
%post
$SNIPPET('autoinstall_done')
sed -i -e '/%sudo/s/ALL$/NOPASSWD:ALL/' /target/etc/sudoers
%end
```

Distribución *Red Hat Enterprise Linux 7.4*, archivo *rhel7.ks*:

```
#version=RHEL7
# Deshabilitamos el cortafuegos
firewall --disabled
# Medio de arranque por red
url --url="http://192.168.60.10/cblr/links/RHEL7-x86_64/"
# Root password
rootpw --iscrypted "ROOT-PASSWORD"
# Network information
network --bootproto=static --ip=192.168.60.21 --netmask=255.255.255.0 --
    gateway=192.168.60.1 --nameserver=192.168.60.1,8.8.8.8 --device=enp0s25
    --hostname=nodo1-ngl --onboot=on --noipv6 --activate
# Tipo de encriptacion CentOS8/RHEL8
auth --enableshadow --passalgo=sha512
# Use text install
text
# Keyboard layouts
keyboard --vckeymap=us --xlayouts='us'
# System language
lang en_US.UTF-8
# Configuracion SElinux
selinux --disabled
# System timezone
timezone Europe/Madrid --isUtc --nontp
# Erase disk
zerombr
bootloader --location=mbr --append="biosdevname=0" --boot-drive=sdb
    --driveorder=sdb,sda
```

```
# Partition clearing information
clearpart --initlabel --all --drives=sdb
autopart --type=lvm
# Run the Setup Agent on first boot
firstboot --enable
# Disk configuration section
ignoredisk --only-use=sdb
# Do not configure the X Window System
xconfig  --startxonboot
# Accept the license agreement
eula --agreed
#Initial user (user with sudo capabilities)
user --groups=wheel --name=NAME --password=PASSWORD
reboot
#Preinstalacion
%pre
$SNIPPET('autoinstall_start')
%end
#Paquetes
%packages
@^graphical-server-environment
@X Window System
%end
%addon com_redhat_kdump --enable --reserve-mb='auto'
%end
%anaconda
pwpolicy root --minlen=6 --minquality=1 --notstrict --nochanges --notempty
pwpolicy user --minlen=6 --minquality=1 --notstrict --nochanges --emptyok
pwpolicy luks --minlen=6 --minquality=1 --notstrict --nochanges --notempty
%end
#Postinstalacion
%post --log=/root/registration_results.out
subscription-manager register --auto-attach --force --username=USERNAME
    --password=PASSWORD
$SNIPPET('autoinstall_done')
sed -i -e '/%sudo/s/ALL$/NOPASSWD:ALL/' /target/etc/sudoers
%end
```

Distribución *Red Hat Enterprise Linux 8.5*, archivo *rhel8.ks*:

```
#version=RHEL8
# Deshabilitamos el cortafuegos
firewall --disabled
# Medio de arranque por red
url --url="http://192.168.60.10/cblr/links/RHEL8-x86_64/"
# Root password
rootpw --iscrypted "ROOT-PASSWORD"
# Network information
network --bootproto=static --ip=192.168.60.21 --netmask=255.255.255.0 --
    gateway=192.168.60.1 --nameserver=192.168.60.1,8.8.8.8 --device=enp0s25
    --hostname=nodo1-ngl --onboot=on --noipv6 --activate
# Tipo de encriptacion CentOS8/RHEL8
authselect --enableshadow --passalgo=sha512
# Use text install
text
# Keyboard layouts
keyboard --vckeymap=us --xlayouts='us'
# System language
lang en_US.UTF-8
# Configuracion SElinux
selinux --disabled
# System timezone
timezone Europe/Madrid --isUtc --nontp
# Erase disk
zerombr
bootloader --location=mbr --append="biosdevname=0" --boot-drive=sdb
    --driveorder=sdb,sda
# Partition clearing information
clearpart --initlabel --all --drives=sdb
autopart --type=lvm
# Run the Setup Agent on first boot
firstboot --enable
# Disk configuration section
ignoredisk --only-use=sdb
# Do not configure the X Window System
xconfig  --startxonboot
# Accept the license agreement
eula --agreed
# SSH connection
sshkey --username=root "SSH-KEY"
#Initial user (user with sudo capabilities)
```

```
user --groups=wheel --name=NAME --password=PASSWORD
reboot
#Preinstalacion
%pre
$SNIPPET('autoinstall_start')
%end
#Paquetes
%packages
@^workstation-product-environment
kexec-tools
yum-utils
%end
%addon com_redhat_kdump --enable --reserve-mb='auto'
%end
%anaconda
pwpolicy root --minlen=6 --minquality=1 --notstrict --nochanges --notempty
pwpolicy user --minlen=6 --minquality=1 --notstrict --nochanges --emptyok
pwpolicy luks --minlen=6 --minquality=1 --notstrict --nochanges --notempty
%end
#Postinstalacion
%post --log=/root/registration_results.out
subscription-manager register --auto-attach --force --username=USERNAME
    --password=PASSWORD
$SNIPPET('autoinstall_done')
sed -i -e '/%sudo/s/ALL$/NOPASSWD:ALL/' /target/etc/sudoers
%end
```

Distribución *Rocky Linux 8.5*, archivo *rocky8.ks*:

```
#version=ROCKY8
# Deshabilitamos el cortafuegos
firewall --disabled
# Medio de arranque por red
url --url="http://192.168.60.10/cblr/links/ROCKY8-x86_64/"
# Root password
rootpw --iscrypted "ROOT-PASSWORD"
# Network information
network --bootproto=static --ip=192.168.60.21 --netmask=255.255.255.0 --
    gateway=192.168.60.1 --nameserver=192.168.60.1,8.8.8.8 --device=enp0s25
    --hostname=nodo1-ngl --onboot=on --noipv6 --activate
# Tipo de encriptacion CentOS8/RHEL8
authselect --enableshadow --passalgo=sha512
```

```
# Use text install
text
# Keyboard layouts
keyboard --vckeymap=us --xlayouts='us'
# System language
lang en_US.UTF-8
# Configuracion SElinux
selinux --disabled
# System timezone
timezone Europe/Madrid --isUtc --nontp
# Erase disk
zerombr
bootloader --location=mbr --append="biosdevname=0" --boot-drive=sdb
    --driveorder=sdb,sda
# Partition clearing information
clearpart --initlabel --all --drives=sdb
autopart --type=lvm
# Run the Setup Agent on first boot
firstboot --enable
# Disk configuration section
ignoredisk --only-use=sdb
# Do not configure the X Window System
xconfig  --startxonboot
# Accept the license agreement
eula --agreed
# SSH connection
sshkey --username=root "SSH-KEY"
#Initial user (user with sudo capabilities)
user --groups=wheel --name=NAME --password=PASSWORD
reboot
#Preinstalacion
%pre
$SNIPPET('autoinstall_start')
%end
#Paquetes
%packages
@^workstation-product-environment
kexec-tools
yum-utils
%end
%addon com_redhat_kdump --enable --reserve-mb='auto'
%end
%anaconda
```

```
54 pwpolicy root --minlen=6 --minquality=1 --notstrict --nochanges --notempty
55 pwpolicy user --minlen=6 --minquality=1 --notstrict --nochanges --emptyok
56 pwpolicy luks --minlen=6 --minquality=1 --notstrict --nochanges --notempty
57 %end
58 #Postinstalacion
59 %post
60 $SNIPPET('autoinstall_done')
61 sed -i -e '/%sudo/s/ALL$/NOPASSWD:ALL/' /target/etc/sudoers
62 %end
```

APÉNDICE B

Código de la interfaz *web*

En este apéndice vamos a presentar el código generado mediante *Visual Studio Code* para la realización de la interfaz *web* incorporada al laboratorio de pruebas, mostraremos las dos partes de las que se compone la interfaz, la parte *back-end* y la parte *fornt-end*. Se han extraído del código contraseñas e información sensible que pudiese comprometer la seguridad del proyecto.

B.1 Código de la parte *back-end*

Archivo *cmd/ngl-api/main.go*

```
1  package main
2
3  import (
4  "arisnova.com/ngl-api/internal/logger"
5  "arisnova.com/ngl-api/internal/request"
6  )
7
8  func main() {
9    router, log := request.GETDefault(logger.NewLogger(), "192.168.60.10", "
       22")
10   log.Log.Info("Starting NGL-API...")
11   err := request.Run(":8080", *log, router)
12   if err != nil {
13     log.Log.Fatal("Error instantiating NGL-API: ", err)
14   }
15 }
```

Archivo *internal/api/api.go*

```
package api

import (
arisJSON "arisnova.com/ngl-api/internal/json"
"arisnova.com/ngl-api/internal/logger"
"encoding/json"
"github.com/gin-gonic/gin"
"io/ioutil"
"net/http"
)

func TestUser(router *gin.Engine) {
  router.GET("/testuser", func(context *gin.Context) {
    userTestJSON := arisJSON.GetUserNull()

    context.JSON(http.StatusOK, gin.H{
      "success": true,
      "error":   "",
      "user":    userTestJSON})

  })
}

func (s *ServerSSH) LoginUser(router *gin.Engine, log logger.Logger) {
  router.POST("/login", func(context *gin.Context) {
    jsonData, err := ioutil.ReadAll(context.Request.Body)

    var frontInformationValues arisJSON.FrontInfo

    errJSON := json.Unmarshal(jsonData, &frontInformationValues)
    if errJSON != nil {
      log.Log.Error(errJSON.Error())
      return
    }

    status, err := s.SSHAuth(frontInformationValues.User,
        frontInformationValues.Password, log)

    if err == nil {
      context.JSON(http.StatusOK, gin.H{
        "success": status,
```

```
41          "error":   "",
42          "user":    arisJSON.GetUser(frontInformationValues.User, true)})
43      } else {
44        context.JSON(http.StatusForbidden, gin.H{
45          "success": status,
46          "error":   err.Error(),
47          "user":    arisJSON.GetUser(frontInformationValues.User, false)})
48      }
49
50    })
51  }
52
53  func (s *ServerSSH) LaunchNode(router *gin.Engine, log logger.Logger) {
54    router.POST("/execute/node", func(context *gin.Context) {
55      jsonData, err := ioutil.ReadAll(context.Request.Body)
56
57      var frontInformationValues arisJSON.FrontInfo
58
59      errJSON := json.Unmarshal(jsonData, &frontInformationValues)
60      if errJSON != nil {
61        log.Log.Error(errJSON.Error())
62        return
63      }
64
65      launchNodeCobblerCommand := "sudo cobbler system remove --name=\"" +
          frontInformationValues.Node + "\";" +
66      "sudo cobbler system add " +
67      "--name=\"" + frontInformationValues.Node + "\" " +
68      "--profile=\"" + frontInformationValues.Profile + "\" " +
69      "--interface=\"enp4s0\" " +
70      "--mac=\"E0:D5:5E:F9:30:E3\" " +
71      "--netboot-enabled=\"Y\";" +
72      "sudo cobbler sync;"
73      }
74
75      commandOutput, err :=
76      s.ExecuteCommand(frontInformationValues.User,
77      frontInformationValues.Password,
78      commandDict[frontInformationValues.Command],
79      log)
80
81      if err == nil {
82        context.JSON(http.StatusOK, gin.H{
```

```
83          "success": true,
84          "error":   "",
85          "command": arisJSON.GetCommand(commandDict[frontInformationValues.
                Command], commandOutput)})
86      } else {
87        context.JSON(http.StatusForbidden, gin.H{
88          "success": false,
89          "error":   err.Error(),
90          "command": arisJSON.GetCommand(commandDict[frontInformationValues.
                Command], commandOutput)})
91      }
92    })
93  }
```

Archivo *internal/api/ssh.go*

```
1  package api
2
3  import (
4  "arisnova.com/ngl-api/internal/logger"
5  "bytes"
6  "golang.org/x/crypto/ssh"
7  "golang.org/x/crypto/ssh/knownhosts"
8  "os"
9  )
10
11 type ServerSSH struct {
12   Server string
13   Port   string
14 }
15
16 func (s *ServerSSH) MakeServer() string {
17   return s.Server + ":" + s.Port
18 }
19
20 func getConfig(user, pass string, log logger.Logger) *ssh.ClientConfig {
21   k := os.Getenv("HOME")
22   // Getting known_hosts file
23   hostKeyCallBack, err := knownhosts.New(k + "/.ssh/known_hosts-api")
24   if err != nil {
25     log.Log.Fatal("Error creating Known Hosts file: ", err)
26   }
```

```
27    return &ssh.ClientConfig{
28      User: user,
29      Auth: []ssh.AuthMethod{
30        ssh.Password(pass),
31      },
32      HostKeyCallback: hostKeyCallBack,
33    }
34  }
35
36  func closeConnection(conn *ssh.Client, log logger.Logger) *ssh.Client {
37    if conn != nil {
38      err := conn.Close()
39      if err != nil {
40        log.Log.Fatal("Error closing ssh connection: ", err)
41      }
42    }
43    return conn
44  }
45
46  func (s *ServerSSH) SSHAuth(user, pass string, log logger.Logger) (bool,
      error) {
47    conn, err := ssh.Dial("tcp", s.MakeServer(), getConfig(user, pass, log))
48    defer closeConnection(conn, log)
49    if err == nil {
50      return true, err
51    } else {
52      return false, err
53    }
54  }
55
56  func (s *ServerSSH) ExecuteCommand(user, pass, command string, log logger.
      Logger) (string, error) {
57
58    conn, err := ssh.Dial("tcp", s.MakeServer(), getConfig(user, pass, log))
59    defer closeConnection(conn, log)
60
61    if err == nil {
62      session, errNewSession := conn.NewSession()
63      if errNewSession != nil {
64        return "", err
65      }
66
67      defer func(session *ssh.Session) {
```

```
68        err := session.Close()
69        if err != nil {
70          log.Log.Error("Error closing SSH session: " + err.Error())
71        }
72      }(session)
73
74      var b bytes.Buffer
75      session.Stdout = &b
76      err = session.Run(command)
77
78      return b.String(), err
79    } else {
80      return "", err
81    }
82  }
```

Archivo *internal/json/command.go*

```
1  package json
2
3  type Command struct {
4    Command string `json:"command"`
5    Output  string `json:"output"`
6  }
7
8  func GetCommand(command, output string) Command {
9    return Command{
10     Command: command,
11     Output:  output,
12   }
13 }
```

Archivo *internal/json/front-info.go*

```
1  package json
2
3  type FrontInfo struct {
4    User     string `json:"user"`
5    Password string `json:"password"`
6    Command  string `json:"command"`
7    Node     string `json:"node"`
```

```
  Profile  string `json:"profile"`
  Options  string `json:"options"`
}
```

Archivo *internal/json/user.go*

```
package json

type User struct {
  User   string `json:"user"`
  Access bool   `json:"access"`
}

func GetUser(user string, access bool) User {
  return User{
    User:   user,
    Access: access,
  }
}

func GetUserNull() User {
  return User{
    User:   "fake",
    Access: false,
  }
}
```

Archivo *internal/logger/logger.go*

```
package logger

import (
rotatelogs "github.com/lestrrat-go/file-rotatelogs"
"github.com/rifflock/lfshook"
"github.com/sirupsen/logrus"
"time"
)

type Logger struct {
  Log *logrus.Logger
}
```

```
13
14 const rootPath = "/tmp"
15
16 var Log *logrus.Logger
17
18 func NewLogger() *logrus.Logger {
19   if Log != nil {
20     return Log
21   }
22
23   path := rootPath + "/ngl-api.log"
24   writer, _ := rotatelogs.New(
25   path+".%Y%m%d%H%M",
26   rotatelogs.WithLinkName(path),
27   rotatelogs.WithMaxAge(time.Duration(86400)*time.Second), // Max. age: 1
         day.
28   rotatelogs.WithRotationTime(time.Hour))                  // Rotation:
         every hour.
29
30   Log = logrus.New()
31   Log.Hooks.Add(lfshook.NewHook(
32   lfshook.WriterMap{
33     logrus.InfoLevel:  writer,
34     logrus.ErrorLevel: writer},
35   &logrus.JSONFormatter{}))
36
37   return Log
38 }
```

Archivo *internal/request/request.go*

```
1 package request
2
3 import (
4 "arisnova.com/ngl-api/internal/api"
5 "arisnova.com/ngl-api/internal/logger"
6 "github.com/gin-gonic/gin"
7 "github.com/sirupsen/logrus"
8 "net/http"
9 "net/http/httptest"
10 )
11
```

```
12 func CORSMiddleware() gin.HandlerFunc {
13   return func(c *gin.Context) {
14     c.Header("Access-Control-Allow-Origin", "http://localhost:3000/")
15     c.Header("Access-Control-Allow-Credentials", "true")
16     c.Header("Access-Control-Allow-Headers", "Content-Type, Content-Length,
           Accept-Encoding, X-CSRF-Token, Authorization, accept, origin, Cache
           -Control, X-Requested-With")
17     c.Header("Access-Control-Allow-Methods", "POST,HEAD,PATCH,OPTIONS,GET,
           PUT")
18
19     if c.Request.Method == "OPTIONS" {
20       c.AbortWithStatus(204)
21       return
22     }
23
24     c.Next()
25   }
26 }
27
28 func setStatic(router *gin.Engine) {
29   // Set a lower memory limit for multipart forms (default is 32 MiB)
30   router.MaxMultipartMemory = 8 << 20 // 8 MiB
31
32   // Single files.
33   router.Static("/index.html", "pkg/public/index.html")
34   router.Static("/manifest.json", "pkg/public/manifest.json")
35   router.Static("/robots.txt", "pkg/public/robots.txt")
36   router.Static("/asset-manifest.json", "pkg/public/asset-manifest.json")
37
38   // Directories.
39   router.Static("/feather", "pkg/public/feather")
40   router.Static("/static", "pkg/public/static")
41   router.Static("/img", "pkg/public/img")
42 }
43
44 func GETDefault(log *logrus.Logger, server, port string) (*gin.Engine, *
       logger.Logger) {
45   // NO debug mode (production)
46   gin.SetMode(gin.ReleaseMode)
47   router := gin.Default()
48   setStatic(router)
49
50   // Adding CORS support
```

```
51    router.Use(CORSMiddleware())
52    // -------------------
53
54    l := &logger.Logger{Log: log}
55
56    s := api.ServerSSH{
57      Server: server,
58      Port:   port,
59    }
60
61    // ---------------------------------
62    // API
63    // ---------------------------------
64    api.TestUser(router)
65    s.LoginUser(router, *l)
66    s.LaunchNode(router, *l)
67    // ---------------------------------
68
69    return router, l
70  }
71
72  func Run(p string, log logger.Logger, eng *gin.Engine) error {
73    log.Log.Info("Running server at ", p, " port!")
74    return eng.Run(p)
75  }
76
77  func PerformRequest(method string, url string, log logger.Logger, eng *gin.
        Engine) *httptest.ResponseRecorder {
78    w := httptest.NewRecorder()
79    req, _ := http.NewRequest(method, url, nil)
80    log.Log.Info("Serving method ", method, " with url ", url)
81    eng.ServeHTTP(w, req)
82    return w
83  }
```

Archivo *test/functional-tests/POSTCall*

```
1  curl http://localhost:8080/login \
2  --include \
3  --header "Content-Type: application/json" \
4  --request "POST" \
5  --data '{"User": " ","Password": " ","Command": "","Node": "","Profile":
       "","Options": ""}'
```

B.2 Código de la parte *front-end*

Archivo *public/index.html*

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8" />
<link rel="icon" href="%PUBLIC_URL%/feather/codesandbox.svg" />
<style>
@import url("https://fonts.googleapis.com/css2?family=Roboto:wght@300&
    display=swap");
</style>
<meta name="viewport" content="width=device-width, initial-scale=1" />
<meta name="theme-color" content="#000000" />
<meta
name="description"
content="Web site created using create-react-app"
/>
<!--
manifest.json provides metadata used when your web app is installed on a
user's mobile device or desktop. See https://developers.google.com/web/
    fundamentals/web-app-manifest/
-->
<link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
<!--
Notice the use of %PUBLIC_URL% in the tags above.
It will be replaced with the URL of the `public` folder during the build.
Only files inside the `public` folder can be referenced from the HTML.

Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC_URL%/favicon.ico" will
work correctly both with client-side routing and a non-root public URL.
Learn how to configure a non-root public URL by running `npm run build`.
-->
<title>NGL</title>
</head>
<body>
<noscript>You need to enable JavaScript to run this app.</noscript>
<div id="root"></div>
<!--
This HTML file is a template.
If you open it directly in the browser, you will see an empty page.

```

```
38 You can add webfonts, meta tags, or analytics to this file.
39 The build step will place the bundled scripts into the <body> tag.
40
41 To begin the development, run ‘npm start‘ or ‘yarn start‘.
42 To create a production bundle, use ‘npm run build‘ or ‘yarn build‘.
43 -->
44 </body>
45 </html>
```

Archivo *public/manifest.json*

```
1  {
2    "short_name": "React App",
3    "name": "Create React App Sample",
4    "icons": [
5    {
6      "src": "favicon.ico",
7      "sizes": "64x64 32x32 24x24 16x16",
8      "type": "image/x-icon"
9    },
10   {
11     "src": "logo192.png",
12     "type": "image/png",
13     "sizes": "192x192"
14   },
15   {
16     "src": "logo512.png",
17     "type": "image/png",
18     "sizes": "512x512"
19   }],
20   "start_url": ".",
21   "display": "standalone",
22   "theme_color": "#000000",
23   "background_color": "#ffffff"
24 }
```

Archivo *src/components/cobbler.js*

```
1 import Form from "./cobblerform"
2 import "bootstrap/dist/css/bootstrap.css";
3 import {Breadcrumb, Card, Container, ListGroup, ListGroupItem, Nav, Navbar,
      NavDropdown} from "react-bootstrap";
```

```
import {useState} from "react";

function Cobbler({user,password}){
  const [outputSSH, setOutputSSH] = useState("");
  const [outputSSHTimestamp,setOutputSSHTimestamp] = useState("");
  const setOutputCallBack = (chilData) => {
    setOutputSSH(chilData)
  }
  const setOutputTimeStamp = (chilData) => {
    setOutputSSHTimestamp(chilData)
  }
  const onSubmit = () => {
  }
  return(
  <div className="container-fluid">
  <Navbar bg="light" expand="lg">
  <Container>
  <Navbar.Brand href="#home"><img src="/img/Logo5_T.png" alt="NGL Logo"
      height="60em"/></Navbar.Brand>
  <Navbar.Toggle aria-controls="basic-navbar-nav" />
  <Navbar.Collapse id="basic-navbar-nav">
  <Nav className="me-auto">
  <Nav.Link href="#home"><img alt={"Home Button"} src="/feather/home.svg"
      /></Nav.Link>
  <NavDropdown title="Settings" id="basic-nav-dropdown">
  <NavDropdown.Item href="#action/3.1">Update my information</NavDropdown.
      Item>
  </NavDropdown>
  <NavDropdown title="Booking" id="basic-nav-dropdown">
  <NavDropdown.Item href="#action/3.1">Booking node</NavDropdown.Item>
  <NavDropdown.Divider />
  <NavDropdown.Item href="#action/3.2">Booking laboratory</NavDropdown.Item
      >
  </NavDropdown>
  </Nav>
  </Navbar.Collapse>
  <Navbar.Collapse className="justify-content-end">
  <Navbar.Text>
  <a href="#login">{user}</a><img alt={"User Symbol"} src="/feather/user.
      svg"/>
  </Navbar.Text>
  <Nav.Item>
  <form onSubmit={onSubmit}>
```

```
42  <button className="button-invisible" type={"submit"}><img alt={"Logout
        Symbol"} title={"Logout"} src="/feather/log-out.svg"/></button>
43  </form>
44  </Nav.Item>
45  </Navbar.Collapse>
46  </Container>
47  </Navbar>
48  <Breadcrumb>
49  <Breadcrumb.Item href="#">Home</Breadcrumb.Item>
50  <Breadcrumb.Item active>Cobbler</Breadcrumb.Item>
51  </Breadcrumb>
52  <div className="row">
53  <div className="col-4">
54  <Form
55  nameForm={"Cobbler actions"}
56  descriptionForm={"Send information to Cobbler system"}
57  buttonForm={"Send"}
58  user={user}
59  password={password}
60  parentCallback={setOutputCallBack}
61  parentCallback2={setOutputTimeStamp}/>
62  </div>
63  <div className="col-8">
64  <div className="row">
65  <div className="col-3 col-md-auto">
66  <Card style={{ width: '10.5rem' }}>
67  <Card.Img variant="top" src="/img/rhel_8_gnome_workspace_panel.png" />
68  <Card.Body>
69  <Card.Title>Node 1</Card.Title>
70  <Card.Text>
71  RHEL 8
72  </Card.Text>
73  </Card.Body>
74  <ListGroup className="list-group-flush">
75  <ListGroupItem><b>IP</b> 192.168.60.21</ListGroupItem>
76  <ListGroupItem>Running...</ListGroupItem>
77  </ListGroup>
78  <Card.Body>
79  <Card.Link href="#">Disable</Card.Link>
80  <Card.Link href="#">Reboot</Card.Link>
81  </Card.Body>
82  </Card>
83  </div>
```

```
84    <div className="col-6 col-md-auto">
85    <Card style={{ width: '35rem' }}>
86    <Card.Body>
87    <Card.Title>Node 1. SSH Output</Card.Title>
88    </Card.Body>
89    <ListGroup className="list-group-flush">
90    <ListGroupItem>{outputSSHTimestamp}</ListGroupItem>
91    <ListGroupItem><div style={{whiteSpace: "pre-wrap"}}>{outputSSH}</div></
          ListGroupItem>
92    </ListGroup>
93    </Card>
94    </div>
95    </div>
96    </div>
97    </div>
98    </div>
99    )
100 }
101
102 export default Cobbler;
```

Archivo *src/components/cobblerform.js*

```
1  import React from "react";
2  import JSONSchemaForm from "react-jsonschema-form";
3  import {Button, Spinner} from "react-bootstrap";
4  import {useState} from "react";
5
6  export default function Form({nameForm,descriptionForm,buttonForm,user,
       password,parentCallback,parentCallback2}) {
7    const [renderSpinnerFlag, setRenderSpinnerFlag] = useState(false);
8
9    const JSONSchema = {
10     "definitions": {
11       "commands": {
12         "type": "string",
13         "enum": [
14         "launch"
15         ]
16       },
17       "nodes":{
18         "type": "string",
```

```
        "enum": [
        "nodo1"
        ]
      },
      "profiles":{
        "type": "string",
        "enum": [
        "ALMA8-Nodo1",
        "ALMA8-x86_64",
        "FEDORA34-Nodo1",
        "FEDORA34-x86_64",
        "RHEL7-Nodo1",
        "RHEL7-x86_64",
        "RHEL8-Nodo1",
        "RHEL8-x86_64",
        "ROCKY8-Nodo1",
        "ROCKY8-x86_64",
        "Ubuntu-20_04-Nodo1",
        "Ubuntu-20_04-casper-x86_64"
        ]
      }
    },
    "title": nameForm,
    "description": descriptionForm,
    "type": "object",
    "required": [
    "cobbler",
    "node",
    "profile"
    ],
    "properties": {
      cobbler: {
        title: "Command",
        "$ref": "#/definitions/commands"
      },
      node: {
        title: "Node",
        "$ref": "#/definitions/nodes"
      },
      profile: {
        title: "Profiles",
        "$ref": "#/definitions/profiles"
      },
```

```
62        options: {
63          title: "Command options",
64          type: "string",
65          maxLength: 5,
66        },
67        user: {
68          type: "string",
69          default: user
70        },
71        password: {
72          type: "string",
73          default: password
74        }
75      }
76    }
77
78    const UISchema = {
79      "cobbler": {
80        "ui:widget": "select",
81        "ui:help": "Select a command to send to Cobbler system"
82      },
83      "node": {
84        "ui:widget": "select",
85        "ui:help": "Select host to send command to"
86      },
87      "profile": {
88        "ui:widget": "select"
89      },
90      "options": {
91        "ui:emptyValue": "",
92        "ui:disabled": "true"
93      },
94      "user": {
95        "ui:widget": "hidden"
96      },
97      "password": {
98        "ui:widget": "hidden"
99      }
100   }
101
102   const onSubmit = ({formData}) => {
103     setRenderSpinnerFlag(true);
104
```

```
105     let json = {
106       User : formData.user,
107       Password: formData.password,
108       Command: formData.cobbler,
109       Node: formData.node,
110       Profile: formData.profile,
111       Options: formData.options
112     }
113
114     console.log("FORM JSON:",json)
115
116     fetch("http://localhost:8080/execute/node", {
117       method: 'POST',
118       headers: {
119         'Accept': 'application/json',
120         'Content-Type': 'application/json',
121       },
122       body: JSON.stringify(json)
123     })
124     .then(response => response.json())
125     .then((resJson) => {
126       console.log("Response: ",JSON.stringify(resJson));
127       const timestamp = Date.now();
128       const momment = new Intl.DateTimeFormat('en-US',
129       {
130         year: 'numeric',
131         month: '2-digit',
132         day: '2-digit',
133         hour: '2-digit',
134         minute: '2-digit',
135         second: '2-digit'}).format(timestamp)
136       parentCallback2(momment)
137       parentCallback(resJson.command.output);
138       setRenderSpinnerFlag(false);
139     })
140   }
141
142   const renderSpinner = () =>
143   renderSpinnerFlag === true && (
144   <span><Spinner
145   as="span"
146   variant="light"
147   size="sm"
```

```
148     role="status"
149     aria-hidden="true"
150     animation="border"/> </span>
151     )
152
153     return (
154     <JSONSchemaForm onSubmit={onSubmit} schema={JSONSchema} uiSchema={
            UISchema}>
155     <Button className="btn btn-primary float-left" type={"submit"}>{
            renderSpinner()}{buttonForm}</Button>
156     </JSONSchemaForm>
157     );
158 }
```

APÉNDICE C

Objetivos de desarrollo sostenible

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenible	**Alto**	**Medio**	**Bajo**	**No procede**
ODS 1. **Fin de la pobreza.**				X
ODS 2. **Hambre cero.**				X
ODS 3. **Salud y bienestar.**				X
ODS 4. **Educación de calidad.**				X
ODS 5. **Igualdad de género.**				X
ODS 6. **Agua limpia y saneamiento.**				X
ODS 7. **Energía asequible y no contaminante.**				X
ODS 8. **Trabajo decente y crecimiento económico.**		X		
ODS 9. **Industria, innovación e infraestructuras.**	X			
ODS 10. **Reducción de las desigualdades.**				X
ODS 11. **Ciudades y comunidades sostenibles.**				X
ODS 12. **Producción y consumo responsables.**			X	
ODS 13. **Acción por el clima.**				X
ODS 14. **Vida submarina.**				X
ODS 15. **Vida de ecosistemas terrestres.**				X
ODS 16. **Paz, justicia e instituciones sólidas.**				X
ODS 17. **Alianzas para lograr objetivos.**				X

Reflexión sobre la relación del TFM con los ODS y con el/los ODS más relacionados.

Tener la oportunidad de visionar o relacionar el proyecto desarrollado desde la perspectiva medioambiental, social, laboral y económica, te abre la mente hacia un futuro más sostenible donde estos objetivos serán tenidos en cuenta para la toma de decisiones importantes por parte de trabajadores y empresas. El Trabajo Final de Máster que hemos tratado en el documento anterior, se relaciona en mayor o menor medida con tres ODS de los citados previamente:

- *Trabajo decente y crecimiento económico;*
- *Industria, innovación e infraestructuras;*
- *Producción y consumo responsables.*

La primera relación que establecemos es con el ODS 8: «Trabajo decente y crecimiento económico.» El proyecto se ha llevado a cabo durante las prácticas curriculares y extracurriculares del máster en una empresa, concretamente en Arisnova S.L. Ingeniería de Sistemas. Donde se han garantizado en todo momento unas condiciones laborales dignas, con una remuneración acorde al nivel de experiencia laboral y tipo de contrato, a su vez, dicha empresa está fomentando el empleo joven a través de contrataciones y programas de prácticas. Cumpliendo con las leyes y regulaciones tributarias actuales en España. Además, garantiza contratos indefinidos, oportunidades de promoción y desarrollo profesional a los empleados, tras la finalización de las prácticas extracurriculares, la empresa nos ha propuesto la continuidad mediante un contrato de trabajo, con posibilidades de promoción y crecimiento dentro de la misma. Esto se ha logrado gracias a la formación y desarrollo adquiridos durante el período de prácticas en esta.

La segunda relación que establecemos es con el ODS 9: «Industria, innovación e infraestructuras.» Gran parte del objetivo del TFM no es otro que crear una infraestructura informática que permite inyectar el sistema operativo elegido por el usuario para realizar pruebas de Quality Assurance. Esto ha sido posible gracias a la inversión en I+D+I de la empresa, adquiriendo los componentes necesarios para la creación del laboratorio. Se están incorporando nuevas tecnologías capaces de mejorar y potenciar el trabajo de los empleados. La innovación introducida en este proyecto deja claro el camino a seguir de Arisnova S.L. Ingeniería de Sistemas, que no es otro que la formación y desarrollo en tecnologías punteras en el sector. Automatizar un sistema aprovisionamiento en red con comunicación entre nodos con arranque PXE-DHCP y con previsión de incorporar el encendido Wake on LAN (WoL), permite tener un control total del encendido y apagado remoto de equipos que repercute directamente en la sostenibilidad de la empresa, utilizando únicamente la energía necesaria en cada prueba.

La tercera y última relación que establecemos es con el ODS 12: «Producción y consumo responsables». Con un aspecto tecnológico, el Trabajo Final de Máster ha logrado reutilizar componentes electrónicos de la propia empresa, extendiendo su vida útil y creando una conciencia de sostenibilidad, reduciendo el gasto económico por parte de Arisnova S.L. en la compra de nuevos componentes para la creación del laboratorio. Los elementos reutilizados son los distintos PCs y los discos duros que hemos empleado durante el proyecto. Con esta práctica hemos reducido el impacto sobre el medioambiente y optimizado los materiales y componentes del proyecto.