Proyecto Fin de Carrera
Grado en Ingeniería de las Tecnologías de Telecomunicación

Servidor de teletrabajo para PYMES de muy bajo coste

Autor: Enrique Sánchez Cardoso
Tutor: Juan Manuel Vozmediano Torres

Dpto. de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2022

Proyecto Fin de Carrera
Ingeniería de las Tecnologías de Telecomunicación

Servidor de teletrabajo para PYMES de muy bajo coste

Autor:
Enrique Sánchez Cardoso

Tutor:
Juan Manuel Vozmediano Torres
Profesor titular

Dpto. de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2022

Proyecto Fin de Carrera: Servidor de teletrabajo para PYMES de muy bajo coste

Autor: Enrique Sánchez Cardoso

Tutor: Juan Manuel Vozmediano Torres

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2022

El Secretario del Tribunal

A mi familia

A mis maestros

Agradecimientos

En primer lugar, agradecer al Profesor Dr. Juan Manuel Vozmediano Torres por haber aceptado ser el tutor del TFG que presento y, no menos, por el afán que durante el Grado nos transmite a los alumnos para que aprendamos.

Agradecer a Guille, Martín y a mis amigos y compañeros, *los presos*, por el apoyo y los momentos vividos durante estos 4 años, y a Javi y Pablo porque siempre puedo contar con ellos.

Por último, gracias a mi familia, a mis padres y a mi hermana Marta, porque nada de esto habría sido posible sin ellos.

Enrique Sánchez Cardoso

Sevilla, 2022

Resumen

Este proyecto trata el diseño y desarrollo de una solución para el problema que las PYMES encuentran a la hora de teletrabajar, ya que, por coste o seguridad, no siempre tienen la posibilidad de replicar el software licenciado en los equipos de los empleados que teletrabajan.

Además, con las restricciones derivadas de la pandemia, ha aumentado el número de empresas que optan por trabajar desde casa, por lo que es conveniente que pequeñas y medianas empresas cuenten con un entorno de teletrabajo barato y seguro.

Este sistema, ubicado en la empresa, se autoconfigura y proporciona un acceso remoto a los dispositivos del trabajo. Para ello, hará uso de Apache Guacamole para el escritorio remoto, del servidor HTTP de Apache, OpenSSL y Let's Encrypt para la gestión de certificados, Docker, DNS dinámico, y Terraform con AWS como proveedor para el acceso desde el exterior.

Abstract

This proyect deals with the design and development of a solution to the problem that PYMES encounter when teleworking, as, due to cost or security, they do not always have the possibility to replicate licensed software on the computers of teleworking employees.

In addition, with pandemic restrictions, more companies are choosing to work from home, so it makes sense for small and medium-sized businesses to have an inexpensive and secure teleworking environment.

This system, located in the company, is self-configuring and provides remote access to work devices. To do this, it will make use of Apache Guacamole for remote desktop, Apache HTTP Server, OpenSSL and Let's Encrypt for certificate management, Docker, dynamic DNS, and Terraform with AWS provider for external access.

Índice

Agradecimientos ..... **ix**

Resumen ..... **xi**

Abstract ..... **xiii**

Índice ..... **xv**

Índice de Figuras ..... **xvii**

Índice de Códigos ..... **xix**

Notación ..... **xxi**

1 Introducción **11**

1.1 Motivación *11*

1.2 Objetivos *11*

1.3 Estado del arte *12*

2 Diseño **13**

2.1 Tecnologías y componentes *13*

2.1.1 Raspberry Pi 13

2.1.2 Ubuntu Server 13

2.1.3 Apache HTTP Server 14

2.1.4 Apache Tomcat 14

2.1.5 Apache Guacamole 14

2.1.6 Docker 15

2.1.7 Terraform 15

2.1.8 CGI 15

2.1.9 Nmap 15

2.1.10 DuckDNS 15

2.1.11 Let's Encrypt 17

2.1.12 Certbot 17

2.1.13 OpenSSL 17

2.1.14 Cron 17

2.1.15 Systemd timers 18

2.1.16 SSH 18

2.1.17 Túnel SSH 18

2.1.18 RDP 19

2.1.19 Wake on Lan 19

2.1.20 Amazon EC2 20

2.1.21 Iptables 22

2.1.22 WebSockets 23

2.1.23 Javascript AJAX 23

2.1.24 Rentry.co 23

2.2 Esquema general *24*

3 Implementación **27**

3.1 Configuración de instalación *27*

3.2 Instancia AWS EC2 ..... *28*
3.2.1 Terraform 28
3.2.2 Tareas Cron 33
3.3 Servidor ubicado en la empresa *35*
3.3.1 Firewall 35
3.3.2 Contenedor Docker de oznu/guacamole 36
3.3.3 Apache HTTP Server 38
3.3.3.1 Instalación 38
3.3.3.2 Certificado para HTTPS 39
3.3.3.3 Configuración de proxy inverso 39
3.3.3.4 Panel de administración 41
3.3.3.5 Redirección de HTTP a HTTPS 43
3.3.4 Detección del cambio de IP local 43
3.3.5 CGI para el panel de administración 45
3.3.6 Contenido estático 54
3.3.6.1 Panel de administración 54
3.3.6.2 Instrucciones 59
3.3.7 Cambio de contraseña por defecto 61
3.4 Desinstalación *62*

4 Uso 63
4.1 Administrador *63*
4.1.1 Panel de administración 63
4.1.1.1 Inscripción de usuarios 64
4.1.1.2 Descubrimiento de dispositivos 65
4.1.1.3 Creación de usuarios con todas las conexiones 66
4.1.2 Plataforma Apache Guacamole 66
4.1.2.1 Cambio de contraseña 66
4.1.2.2 Creación de conexiones 67
4.1.2.3 Creación de usuarios 70
4.2 Usuario *72*
4.2.1 Instrucciones 72
4.2.2 Plataforma de Apache Guacamole 73
4.2.2.1 Inicio de sesión 73
4.2.2.2 Conexión remota 74
4.3 Compatibilidad *76*

5 Conclusión 77

6 Líneas de mejora 79

Glosario 81

Referencias 83

Anexo A 87

ÍNDICE DE FIGURAS

Figura 2-1. Apache Guacamole 14
Figura 2-2. Duck DNS – Inicio de sesión 16
Figura 2-3. Duck DNS – Token y subdominio 16
Figura 2-4. Túnel SSH – Local Port Forwarding 18
Figura 2-5. Túnel SSH – Remote Port Forwarding 19
Figura 2-6. Wireshark – Paquete Wake on Lan 19
Figura 2-7. AWS – Registro paso 1 20
Figura 2-8. AWS – Registro paso 2 21
Figura 2-9. AWS – Registro paso 3 21
Figura 2-10. AWS – Generación de Access Keys 22
Figura 2-11. AWS – Reglas de entrada Firewall 22
Figura 2-12. Esquema general 24
Figura 2-13. Esquema general – Ampliado empresa 24
Figura 2-14. Esquema general – Ampliado AWS 25
Figura 3-1. Instancia AWS EC2 – Redirección de puertos 31
Figura 3-2. Advertencia SSH identificación host remoto 33
Figura 3-3. Instancia AWS EC2 – Redirección de puertos y túnel SSH 33
Figura 3-4. Servidor ubicado en la empresa 35
Figura 3-5. Contenedor de oznu/guacamole 36
Figura 3-6. Apache Guacamole 37
Figura 3-7. Wireshark – Paquete Wake on Lan 37
Figura 3-8. Certificado digital FNMT de profundidad 2 40
Figura 3-9. Wireshark – Credenciales en texto plano al usar HTTP 41
Figura 3-10. RDPW_Installer.exe 60
Figura 4-1. Inicio de sesión para el panel de administración 64
Figura 4-2. Panel de administración 64
Figura 4-3. Panel de administración con nmap cargando 65
Figura 4-4. Panel de configuración con nmap cargado 65
Figura 4-5. Creación de usuarios con todas las conexiones 66
Figura 4-6. Inicio de sesión en el cliente web 66
Figura 4-7. Menú desplegable para configuración 67
Figura 4-8. Cambiar contraseña 67
Figura 4-9. Menú conexiones 68
Figura 4-10. Nueva conexión 68

Figura 4-11. Formulario de nueva conexión 69
Figura 4-12. Rellenar MAC para WoL 69
Figura 4-13. Crear nuevo usuario 70
Figura 4-14. Formulario de nuevo usuario 71
Figura 4-15. Selección de conexiones permitidas a un usuario 71
Figura 4-16. Instrucciones para Windows 72
Figura 4-17. Instrucciones para Ubuntu 72
Figura 4-18. Instrucciones para CentOS 73
Figura 4-19. Solicitud de certificado digital del cliente 73
Figura 4-20. Inicio de sesión del usuario 74
Figura 4-21. Conexiones de usuario 74
Figura 4-22. Dos conexiones abiertas al mismo tiempo 75
Figura 4-23. Inicio de sesión en dispositivo sin usuario/contraseña preconfigurado 75

ÍNDICE DE CÓDIGOS

Código 3.1. install.sh – Instalación de Terraform 28
Código 3.2. install.sh – Token AWS 28
Código 3.3. install.sh – Copiar código de la infraestructura para Terraform 28
Código 3.4. variables.tf 29
Código 3.5. main.tf – Versiones Terraform y AWS provider 29
Código 3.6. main.tf – Creación de clave RSA 29
Código 3.7. main.tf – Ubicación de la instancia 30
Código 3.8. main.tf – Parámetros de la instancia AWS EC2 30
Código 3.9. main.tf – Conexión SSH 30
Código 3.10. main.tf – Ejecución remota de comandos 31
Código 3.11. main.tf – Ejecución de comandos en local, cron para el túnel SSH 31
Código 3.12. tunnel.sh – Túnel SSH reiniciable automáticamente 32
Código 3.13. main.tf – Ejecución de comandos en local, destrucción de la instancia 32
Código 3.14. install.sh – Tareas cron encendido y apagado 33
Código 3.15. install.sh – Servicio systemd para encendido del sistema en cada @reboot 34
Código 3.16. install.sh – Firewall con Iptables 36
Código 3.17. install.sh – Persistir las reglas de Iptables 36
Código 3.18. install.sh – Descargar, instalar y poner en marcha el contenedor oznu/guacamole 37
Código 3.19. install.sh – Branding personalizado 38
Código 3.20. install.sh – Instalar Apache HTTP Server y módulos 38
Código 3.21. install.sh – Instalar Certbot y pedir certificado 39
Código 3.22. install.sh – Configuración del *VirtualHost* que gestiona el puerto 443 39
Código 3.23. 000-default-le-ssl.conf – Activar verificación del cliente con certificado digital 40
Código 3.24. 000-default-le-ssl.conf – Aceptar solo los certificados de la lista 40
Código 3.25. 000-default-le-ssl.conf – Bloques *Location* 40
Código 3.26. install.sh – Copiar el contenido estático de /instrucciones 40
Código 3.27. 000-default-le-ssl.conf – Desactivar *logging* de peticiones del proxy 41
Código 3.28. install.sh – Generar certificado autofirmado para el panel 41
Código 3.29. install.sh – Activar *login* y crear usuario 42
Código 3.30. Obtener la IP local y modificar la configuración del panel 42
Código 3.31. panel.conf – Certificado y condiciones para acceder al panel 42
Código 3.32. install.sh – Redirección de HTTP a HTTPS para el panel 43
Código 3.33. 000-default.conf – Redirección de HTTP a HTTPS 43
Código 3.34. install.sh – Reinicio de Apache HTTP Server 43
Código 3.35. postIp.sh – Detección del cambio de IP local 44

Código 3.36. install.sh – Instalar detección del cambio de IP local 44
Código 3.37. panel.conf – Configuración de cgi 45
Código 3.38. install.sh – Instalar nmap, xmllint, python3 y activar el mod_cgi 45
Código 3.39. install.sh – Localización y permisos de archivos cgi 46
Código 3.40. sincronizar_usuarios.sh 46
Código 3.41. Permisos y sudoers para sincronizar_usuarios.sh 47
Código 3.42. recibir.cgi 47
Código 3.43. leer.cgi 48
Código 3.44. nmap.cgi 48
Código 3.45. crear.cgi 49
Código 3.46. crear_usuario.py 50
Código 3.47. crear_conexion.py 51
Código 3.48. crear_todas.cgi 53
Código 3.49. crear_conexion_todas.py 53
Código 3.50. index.html – Página de inicio del panel de administración 54
Código 3.51. panel.css – Estilos para el panel de administración 55
Código 3.52. panel.js – Peticiones al CGI de control de usuarios usando AJAX 56
Código 3.53. panel.js – Peticiones al CGI de control de dispositivos usando AJAX 57
Código 3.54. panel.js – Petición a crear usuario con todas las conexiones asociadas 58
Código 3.55. index.html – Página de inicio para las instrucciones de los usuarios 59
Código 3.56. centos.sh – Instalar servidor RDP CentOS 60
Código 3.57. ubuntu.sh – Instalar servidor RDP Ubuntu 60
Código 3.58. windowspro.bat – Activar servidor RDP Windows PRO 60
Código 3.59. install.sh – Cambio de contraseña del usuario guacadmin 61
Código 3.60. cambiar_guacadmin.py – Cambio de contraseña del usuario guacadmin 61

Notación

RAM	Random Access Memory
LTS	Long Term Support
RDP	Remote Desktop Protocol
AWS	Amazon Web Services
EC2	Elastic Compute Cloud
CGI	Common Gateway Interface
DNS	Domain Name Service
AC	Autoridad Certificadora
ISRG	Internet Security Research Group
ACME	Automatic Certificate Management Environment
EFF	Electronic Frontier Foundation
SSH	Secure Shell
HTTP	Hypertext Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
IAM	Identity and Access Management
FNMT	Fábrica Nacional de Moneda y Timbre
CN	Common Name

1 INTRODUCCIÓN

La simplicidad es la máxima sofisticación.

- Leonardo da Vinci -

1.1 Motivación

El teletrabajo y las jornadas mixtas oficina-casa cada vez son más populares. Se comenzó debido a las restricciones de movilidad a causa de la pandemia, y tras probarlo, muchas empresas decidieron apostar por estos modelos.

En este momento surge el problema de que la mayoría del software que usa un trabajador en la oficina tiene licencia y, por coste o seguridad, no puede ser instalado en el ordenador personal. Por ello, es conveniente que las empresas tengan un sistema seguro para permitir el acceso a los ordenadores del trabajo, desde casa.

Existen soluciones para esto como pueden ser Team Viewer, Any Desk, etc. pero debido al elevado coste de los planes empresariales, sobre todo para pequeñas y medianas empresas, y que usar estos servicios supone que todos los datos pasen por empresas externas, surge este proyecto.

1.2 Objetivos

Se busca ofrecer un sistema seguro y barato para PYMES que, ubicado en la empresa, permita el teletrabajo. Si desarrollamos los objetivos, se pide que:

- Los trabajadores puedan acceder desde sus dispositivos personales a los dispositivos de la empresa.
- El sistema esté ubicado en la empresa.
- Sea Plug-and-Play, es decir, que una vez se instale en el servidor, se ubique en la empresa y se autoconfigure e inicie.
- La comunicación se haga sobre un canal seguro, con encriptación tanto en el trayecto dispositivo empresa-servidor web (red privada de la empresa), como servidor web-dispositivo personal (red pública Internet).
- No se necesite modificar la configuración del router de la empresa, por ejemplo, para usar *Port Forwarding*.
- Se acceda al sistema usando un nombre de dominio personalizado.
- La web de inicio de sesión esté personalizada con el logo y nombre de la empresa.
- Para permitir el acceso a la plataforma sea necesario el uso de un certificado digital personal expedido por la Fábrica Nacional de Moneda y Timbre, que autorizará el administrador del sistema.
- Una vez se accede a la plataforma con el certificado digital, cada miembro de la empresa tenga su propio usuario.

- Cada usuario pueda acceder solo a los dispositivos que se le permita por el administrador del sistema.
- Exista un panel de administración donde el administrador configurará los usuarios y las conexiones que tienen permitidas.
- Exista un panel de administración desde el cual el administrador del sistema pueda modificar los certificados digitales que permitan el acceso al sistema.
- El acceso al panel esté protegido con usuario y contraseña de administrador y solo se pueda acceder a este desde la red local de la empresa.
- Se renueven automáticamente los certificados para https.
- Soporte el Wake on Lan (WoL) para que no sea necesario que un dispositivo esté encendido para acceder a él desde fuera de la empresa.
- No se necesite la instalación de software en el dispositivo personal, sino que se acceda vía web, de manera que será posible el uso del sistema tanto desde ordenador como dispositivo móvil.
- El software necesario para el funcionamiento en el dispositivo de la empresa sea mínimo.
- Exista una sección de instrucciones que detecte el sistema operativo desde el que accedes para que así te ayude a configurar el dispositivo de la empresa.

Para ello, haremos uso de diferentes tecnologías, entre ellas, Apache Guacamole, Docker, Terraform, AWS, DNS dinámico y Let's Encrypt.

1.3 Estado del arte

Existen varias plataformas que ofrecen una solución para el acceso remoto a dispositivos, como pueden ser TeamViewer, AnyDesk, RemotePC, etc. Todas estas ofrecen un plan para empresas con una suscripción anual. Generalmente cobran por licencia de usuario y ponen un límite tanto a los equipos a los que se puede acceder como a desde cuántos equipos se puede usar la licencia.

Si nos centramos en el ámbito económico, estas soluciones tienen un precio mínimo mensual de 15€ por usuario, aunque el pago real se haga de un año entero. Suponiendo una pequeña empresa (<50 empleados) [1], si hacemos los cálculos, pagar uno de estos servicios puede costar hasta 9.000€ anuales. Además, debemos tener en cuenta las limitaciones específicas como pueden ser un número máximo de estos usuarios en línea simultáneamente.

Por otro lado, el uso de estas plataformas conlleva que los datos de uso de los dispositivos pasen por sus servidores, algo que a nivel de privacidad puede no gustar a todas las empresas.

2 DISEÑO

Reconocer la necesidad es la primera condición para diseñar.

- Charles Eames -

Considerando los objetivos a cumplir para solucionar el problema planteado se decide diseñar un sistema a partir de servicios y componentes de código libre ya existentes, de manera que, con todos estos trabajando como grupo se logre el fin expuesto anteriormente.

2.1 Tecnologías y componentes

Este apartado describe las tecnologías y componentes usados. Si estos necesitan registro, se explica cómo hacerlo.

2.1.1 **Raspberry Pi**

Una Raspberry Pi es un miniordenador de bajo coste desarrollado en Reino Unido por la Raspberry Pi Foundation. Inicialmente fue creada para fomentar la informática en el mundo educativo [2], pero ha terminado siendo popular en otros muchos ámbitos.

Se ha escogido una Raspberry Pi 4 Model B con 2GB de RAM como servidor para este sistema, debido a que ofrece comodidad a la hora de desarrollo, ya que es una máquina aislada solo para este proyecto, tiene un bajo coste tanto de inversión inicial como de gasto de electricidad, es portable y, además, ya disponía de ella anteriormente.

Estará conectada a un router con acceso a Internet mediante un cable ethernet y la alimentación a través de un cable USB-C y una fuente de alimentación de 5V 3A.

2.1.2 **Ubuntu Server**

Ubuntu Server es una distribución open source de Linux basada en Debian GNU/Linux. Es uno de los sistemas operativos para servidores más usados a nivel mundial con hasta 11,902,679 sitios web [3].

Este sistema operativo tiene varias versiones, entre ellas las LTS (Long Term Support) que reciben actualizaciones durante 5 años.

Por todo esto, para el desarrollo de este sistema se ha elegido Ubuntu Server 20.04 LTS, ya que así se cubre una mayor cuota de mercado con respecto a otros sistemas operativos como Raspbian, diseñados específicamente para Raspberry Pi, la cual utilizamos como servidor.

2.1.3 Apache HTTP Server

Se trata de un servidor HTTP de código libre. Fue creado en 1995 y desde 1996 es el servidor web más popular según su web oficial [4]. Es un proyecto desarrollado y mantenido por la comunidad bajo la supervisión de Apache Software Foundation, con la licencia Apache License 2.0.

Destaca que es un servidor muy configurable lo que nos permitirá usar cgi, conexión con Tomcat, verificación de clientes mediante certificado digital, etc.

2.1.4 Apache Tomcat

Se trata de una implementación de software libre de la plataforma Jakarta Enterprise Edition (la evolución de Java EE). Fue creado en el 1999 y Está desarrollado y mantenido por la comunidad bajo la supervisión de Apache Software Foundation., con la licencia Apache License Version 2.0 [5].

El 0.23% de los servidores web usan Tomcat siendo así el servidor de Jakarta EE más popular [6].

Sobre este, ejecutaremos Apache Guacamole que nos permitirá acceder remotamente a los ordenadores usando RDP.

2.1.5 Apache Guacamole

Es una puerta de enlace para escritorio remoto que no necesita la instalación de un cliente, es decir, permite controlar otros ordenadores mediante los protocolos RDP, VNC y SSH, entre otros, con el cliente HTML5 de su servidor web [7]. Gracias a este podremos usar remotamente los dispositivos de la empresa.

Está desarrollado y mantenido por la comunidad bajo la supervisión de Apache Software Foundation, con la licencia Apache License Version 2.0.

Proporciona encriptación para la conexión entre el servidor de Apache Guacamole y el ordenador que queramos controlar remotamente.

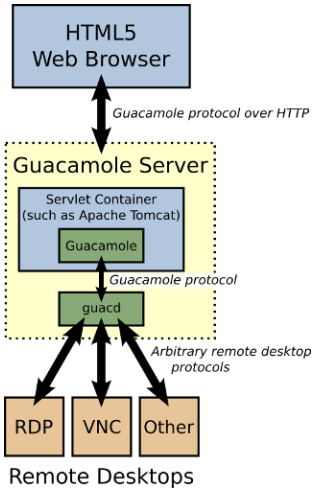

Figura 2-1. Apache Guacamole

2.1.6 Docker

Docker es una plataforma de software libre creada en 2013, que sigue la licencia Apache License 2.0 [8]. Permite crear, probar e implementar aplicaciones rápidamente. Para ello, empaqueta software en unidades estandarizadas llamadas contenedores [9] que incluye todo lo necesario para que ese software en concreto funcione.

Es muy útil ya que nos permite tener rápidamente un servidor de Apache Guacamole funcionando.

2.1.7 Terraform

Terraform es una herramienta de software libre que permite crear infraestructura a partir de código [10], es decir, posee un lenguaje de alto nivel para desplegar infraestructura como instancias de máquinas virtuales, bases de datos, servicios de almacenamiento... Soporta varios proveedores como AWS, Azure, Google Cloud, etc. Fue lanzado en 2014 bajo la licencia Mozilla Public License 2.0 [11].

Esto nos servirá para desplegar automáticamente instancias EC2 de AWS, con *terraform apply*, y para destruirlas, con *terraform destroy*.

2.1.8 CGI

CGI (Common Gateway Interface) es un método por el cual un servidor web puede interactuar con programas externos de generación de contenido, a ellos nos referimos comúnmente como programas CGI o scripts CGI. Es un método muy común y sencillo de mostrar contenido dinámico en un sitio web [12].

Fue desarrollado en la década de 1990. Fue uno de los primeros métodos que permitían crear una web interactiva [13].

Esto nos servirá para crear un panel web de administración del sistema.

2.1.9 Nmap

Network Mapper es una herramienta de software libre creada para la evaluación de seguridad. Esta utilidad nos permite el descubrimiento de dispositivos en una red, escaneo de puertos, etc. [14]

Gracias a ello, podremos ver desde el panel de administración qué ordenadores tienen RDP activado y mostrar su IP y MAC. Esto será útil para el administrador del sistema ya que necesitará estos datos para crear las conexiones en Apache Guacamole.

2.1.10 DuckDNS

DuckDNS es un servicio de DNS dinámico gratuito hospedado en AWS. Nos permite crear una cuenta y un subdominio del tipo subdominio.duckdns.org. Mediante una petición http podremos actualizar la IP a la que apunta este dominio [15].

Esto nos servirá para tener un nombre de dominio con el que acceder siempre a nuestra instancia AWS EC2.

Para el uso de este servicio tendremos que registrarnos usando una cuenta de Google, Twitter o Github. El login con Persona y Reddit está desactivado.

Figura 2-2. Duck DNS – Inicio de sesión

Una vez dentro, nos aparecerá el token de la cuenta y un botón para añadir un subdominio. Añadimos el subdominio que queramos usar para nuestra web.

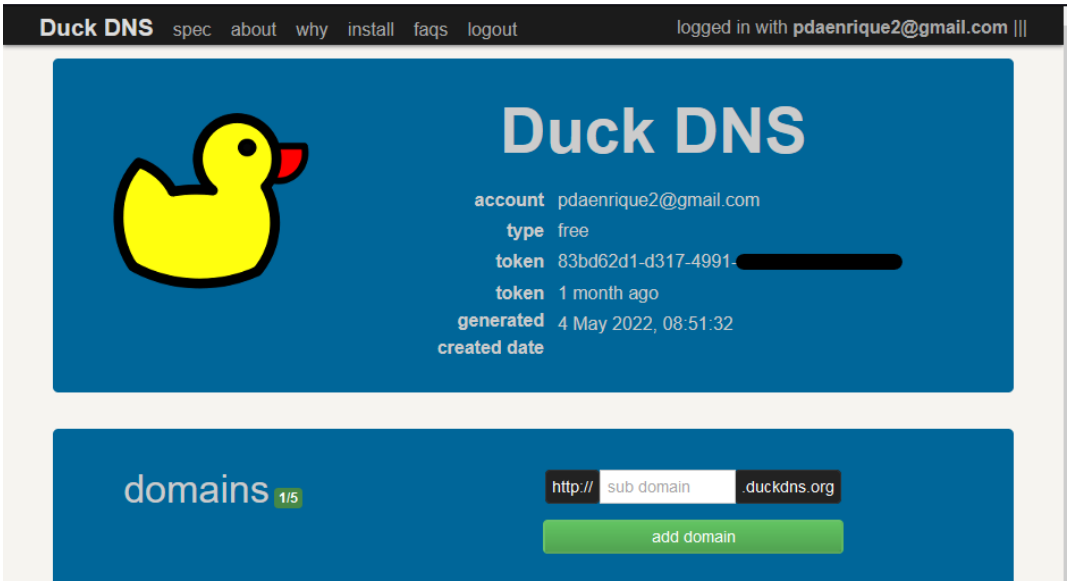

Figura 2-3. Duck DNS – Token y subdominio

Añadiremos tanto el token como el subdominio al archivo config del proyecto.

2.1.11 **Let's Encrypt**

Let's Encrypt es una autoridad de certificación (AC, o CA por sus siglas en inglés) que se puso en marcha en 2016. Está provisto por el Internet Security Research Group [16].

Es una manera gratuita, automática y transparente de obtener un certificado para un servicio web. Actualmente (06/06/2022) hay más de 217 millones de certificados activos expedidos por Let's Encrypt [17].

Para ello, utiliza el protocolo ACME (Automatic Certificate Management Environment) [18], un protocolo que automatiza el proceso de verificación de un cliente que solicita un certificado. Hay muchos clientes ACME para esto, pero recomiendan el uso de Certbot [19].

2.1.12 **Certbot**

Es un cliente ACME gratuito de software libre para la automatización del uso de certificados Let's Encrypt. Fue creado por la EFF (Electronic Frontier Foundation), una organización sin ánimo de lucro [20].

Lo usaremos para obtener y renovar un certificado Let's Encrypt para nuestro dominio.

Para usar Certbot tendremos que introducir una dirección de correo electrónico, pero no hará falta registrarse. Este se usará para que nos avisen de cuando haya que renovar el certificado.

2.1.13 **OpenSSL**

OpenSSL es un proyecto de software libre que agrupa herramientas de administración y bibliotecas relacionadas con la criptografía. Este paquete suministra funciones criptográficas a otros paquetes como OpenSSH y navegadores web. Estas herramientas ayudan al sistema a implementar el Secure Sockets Layer (SSL), así como otros protocolos relacionados con la seguridad, como el Transport Layer Security (TLS). [21]

Lo usaremos para generar un certificado autofirmado para el panel de administración. Como el nombre usado para acceder a este es una dirección IP privada, ninguna autoridad certificadora nos ofrecería uno.

2.1.14 **Cron**

Cron es un administrador de procesos en segundo plano (demonio) que ejecuta procesos en intervalos regulares, como puede ser cada cierto tiempo en minutos, horas, días, cuando se reinicie el dispositivo, etc. [22]

Lo usaremos para:

- Ejecutar un script cada minuto que comprueba que el túnel SSH funciona y si no, lo reinicia.
- Crear y destruir la infraestructura con Terraform en los horarios configurados. Por ejemplo, hacer *terraform apply* los lunes a las 07:00 y *destroy* los domingos a las 00:00 para ahorrar costes en AWS EC2.
- Comprobar los cambios de la IP local, modificar en consecuencia la configuración y publicar este cambio en una nota de rentry.co. Esto será necesario para que el administrador pueda acceder a su panel.

2.1.15 Systemd timers

Los temporizadores son archivos de systemd con extensión .timer que controlan archivos .service o eventos. Se pueden usar como una alternativa a cron. Tienen soporte incorporado para ejecutar eventos basados en el calendario, eventos de tiempo monotónicos y se pueden ejecutar de forma asíncrona [23].

En este caso:

- Certbot lo usa para establecer un temporizador que renueve automáticamente el certificado que ha expedido.
- Creamos la infraestructura con Terraform cuando se arranca el sistema operativo con un servicio de systemd que se ejecuta al iniciar el servidor. Esto nos sirve para que automáticamente cuando el servidor se ubique en la empresa, empiece todo a funcionar.

2.1.16 SSH

SSH (Secure Shell) es un protocolo que facilita las comunicaciones seguras entre dos sistemas usando una arquitectura cliente/servidor y que permite a los usuarios conectarse a un host remotamente. A diferencia de otros protocolos de comunicación remota tales como FTP o Telnet, SSH encripta la sesión de conexión [24].

Ya que el protocolo SSH encripta todo lo que envía y recibe, se puede usar para asegurar protocolos inseguros.

Una conexión SSH puede ofrecer un conducto para enviar mensajes de otros protocolos, este mecanismo se conoce como túnel ssh [25].

2.1.17 Túnel SSH

Un túnel SSH es un mecanismo que permite el reenvío de puertos desde la máquina del cliente al servidor (Local Port Forwarding) o viceversa (Remote Port Forwarding) [26].

Usaremos tanto el reenvío de puertos remoto, para que las conexiones HTTP y HTTPS entrantes a la instancia EC2 se redirijan a nuestra Raspberry Pi, como el reenvío de puertos local, para poder monitorizar el estado del túnel ssh.

A veces, un túnel SSH puede caerse y dejar de funcionar, por lo que tendremos que asegurarnos de que cuando esto pase, se reinicie. Ante esto hay dos opciones, usar dicho túnel para ejecutar un comando, por ejemplo, *ls* en la máquina remota y si este da error, reiniciarlo; o usar la utilidad autossh. En este proyecto hemos escogido la primera opción, ahorrándonos una dependencia.

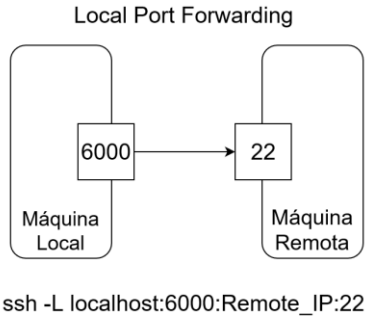

Figura 2-4. Túnel SSH – Local Port Forwarding

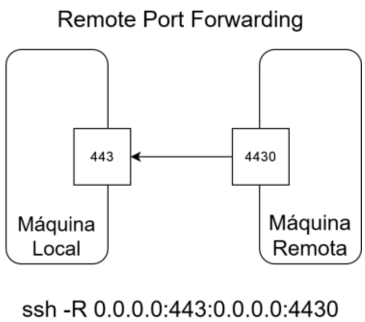

Figura 2-5. Túnel SSH – Remote Port Forwarding

La diferencia entre estos dos modos es que en el *Local Port Forwarding* lo que le llegue al puerto local se reenviará hacia el puerto remoto, y en el *Remote Port Forwarding*, lo que le llegue al puerto remoto, se reenviará hacia el puerto local. Es decir, funcionan en una dirección.

2.1.18 **RDP**

Remote Desktop Protocol (RDP) es un protocolo propietario desarrollado por Microsoft que se utiliza para acceder a los programas, archivos y recursos de los equipos destino como si se estuviera sentado delante de él [27].

El servidor convierte la información gráfica a un formato propio y la encapsula en paquetes RDP, para así, enviarla al cliente, el cual se encarga de interpretar y reconstruir la imagen. Por otro lado, tanto la información del teclado como ratón del cliente es enviada al servidor. Todos estos datos pueden ser comprimidos para mejorar el rendimiento [28]. Por defecto, el servidor usa el puerto TCP 3389.

2.1.19 **Wake on Lan**

Wake on Lan es un estándar creado por Intel e IBM en abril de 1997 que permite encender remotamente un dispositivo compatible mediante el envío de un paquete especial llamado Magic Packet. Este paquete contiene 16 repeticiones de la dirección MAC del dispositivo a encender y se envía a la dirección MAC de difusión.

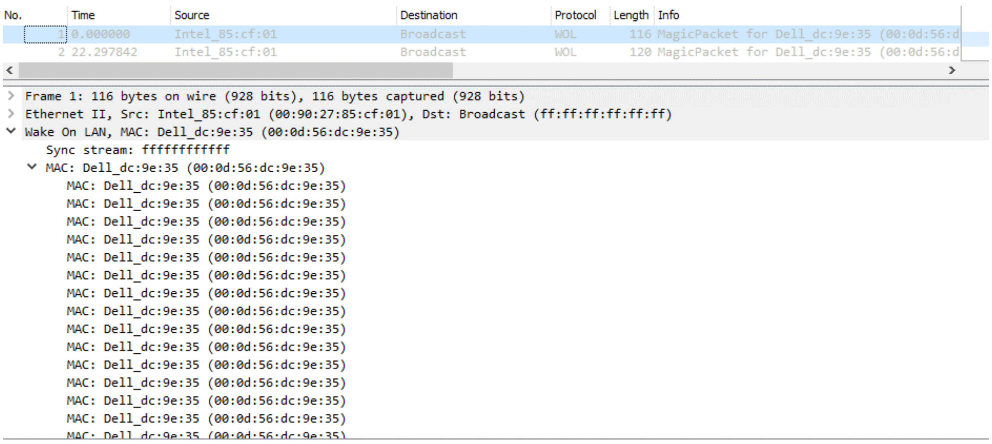

Figura 2-6. Wireshark – Paquete Wake on Lan

Si los dispositivos de la empresa soportan WoL podremos hacer que Apache Guacamole envíe un magic packet y así usar remotamente los dispositivos sin necesidad de que estén encendidos.

2.1.20 **Amazon EC2**

Amazon Elastic Compute Cloud (Amazon EC2) proporciona capacidad de computación escalable en la nube de Amazon Web Services (AWS), es decir, nos permite alquilar instancias virtuales con una configuración personalizada de CPU, memoria, almacenamiento, capacidad de red, ubicación, etc. [29]

Esto nos permite ejecutar aplicaciones de todo tipo, servicios web, machine learning, apps distribuidas, etc.

En nuestro caso lo usaremos para establecer un túnel SSH desde el servidor ubicado en la empresa y así exponer sus puertos a Internet. De esta manera no tendríamos la necesidad de cambiar la configuración del router de la oficina ya que no necesitaríamos *Port Forwarding*.

Para usar este servicio tendremos que registrarnos en los servicios web de Amazon. Introducimos correo electrónico y nombre de usuario.

Figura 2-7. AWS – Registro paso 1

Una vez verifiquemos el email introducido nos pedirá una contraseña y datos de contacto, nombre, teléfono, ciudad, etc.

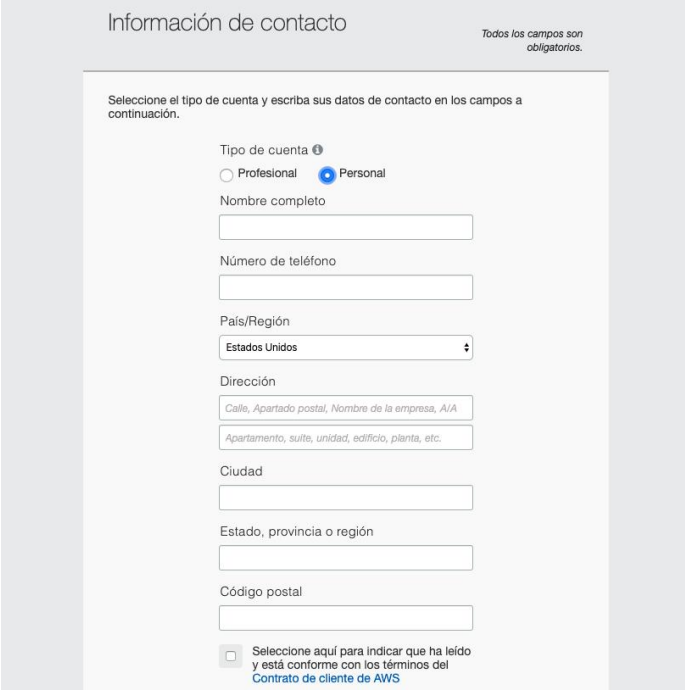

Figura 2-8. AWS – Registro paso 2

Y luego información de pago. Se nos hará un cargo de 1€ para verificar que la cuenta existe y posteriormente se reembolsará.

Figura 2-9. AWS – Registro paso 3

Ya tendremos acceso a la consola de AWS donde debemos generar un par de claves para el acceso a los servicios de Amazon desde Terraform. Accedemos a la sección IAM y pulsamos el botón de Create New Acces Key.

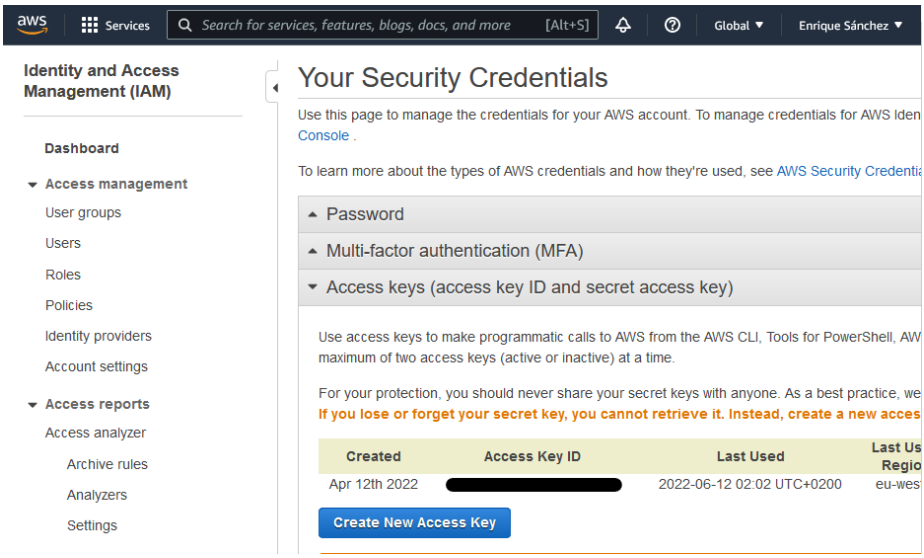

Figura 2-10. AWS – Generación de Access Keys

Guardamos estas claves para incluirlas en el archivo de configuración de instalación. El siguiente paso es crear un Security Group para establecer qué puertos vamos a abrir de cara a la IP pública. Estos serán el 22 para usar SSH, el 80 para HTTP y el 443 para HTTPS.

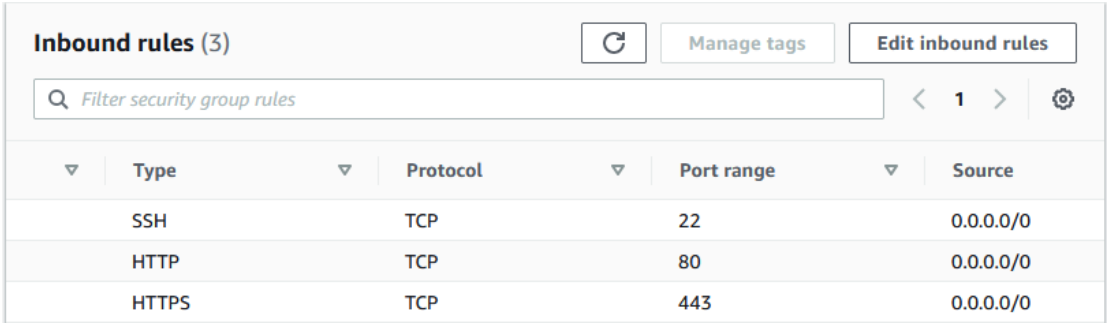

Figura 2-11. AWS – Reglas de entrada Firewall

Ya tendríamos completado el registro y configurado AWS. Con estas opciones, desplegar una instancia EC2 no supondrá ningún coste el primer año ya que están incluidas en el plan gratuito que consta de 750 horas al mes (un mes con 31 días tiene 744 horas).

2.1.21 Iptables

Iptables es un programa que modifica el comportamiento del módulo Netfilter del kernel de Linux. De esta manera podremos filtrar los paquetes según reglas configurables como pueden ser ip origen y destino, puerto, protocolo, etc. Para la persistencia de las reglas, instalamos iptables-persistent.

Esto lo usaremos para que el servidor solo acepte los paquetes de los servicios que necesitamos y los demás los tire.

2.1.22 **WebSockets**

WebSockets es un protocolo que permite abrir una sesión de comunicación *full-duplex* entre el navegador web y el servidor, aunque puede utilizarse por cualquier aplicación cliente/servidor. Está descrito en la RFC 6455, basado en TCP y es fiable, eficiente e implementado por la mayoría de los clientes web. [30]

En este proyecto usaremos este protocolo para el proxy entre Apache HTTP Server y Apache Guacamole (que funciona sobre Apache Tomcat).

2.1.23 **Javascript AJAX**

JavaScript Asíncrono + XML (AJAX) no es una tecnología por sí misma, es un término que describe un modo de utilizar conjuntamente varias tecnologías existentes. Esto incluye HTML, CSS, JavaScript, DOM, XML… y lo más importante, el objeto XMLHttpRequest. Se usa para crear aplicaciones web que pueden actualizarse sin recargar la página completa, lo cual hace que la experiencia del usuario mejore al recibir respuestas más rápidas. [31]

Mediante AJAX se harán las peticiones del panel de administración a los endpoints gestionados por CGI que realizan las labores administrativas.

2.1.24 **Rentry.co**

Se trata de un servicio de creación de notas que permite usar una url personalizada y modificar su contenido mediante una contraseña. Gracias a un script en *python3* de su creador, el usuario de github “radude”, podremos crear, borrar y editar las notas desde la línea de comandos.

Lo usaremos con el fin de que el administrador conozca la dirección con la que acceder en todo momento al panel de administración. Así pues, en la nota que configuremos se encontrará la IP local del servidor actualizada al minuto.

Se ha elegido este servicio para notificar del cambio de IP frente a otros como el correo electrónico debido a la sencillez que aporta, no necesita registro, no necesita usar un servidor de emails y siempre mantiene la misma url donde consultar la dirección IP.

2.2 Esquema general

El esquema general que sigue nuestro diseño es el siguiente:

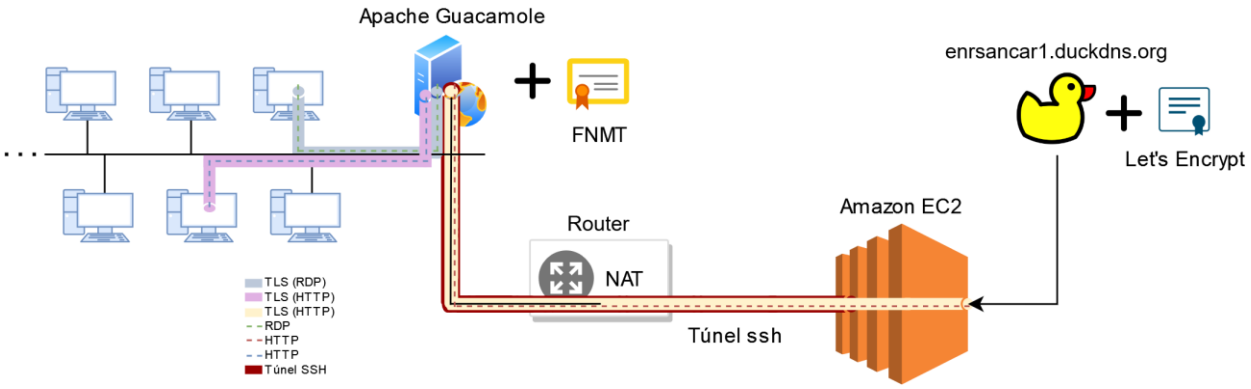

Figura 2-12. Esquema general

A la izquierda podemos ver la red LAN de la empresa donde están los dispositivos a usar remotamente y el servidor (en nuestro caso Raspberry Pi) donde se ejecuta Apache Guacamole. El servidor se comunicará con los dispositivos de la empresa mediante RDP usando un protocolo de cifrado a elegir por el administrador en el momento de crear las conexiones, por defecto, TLS.

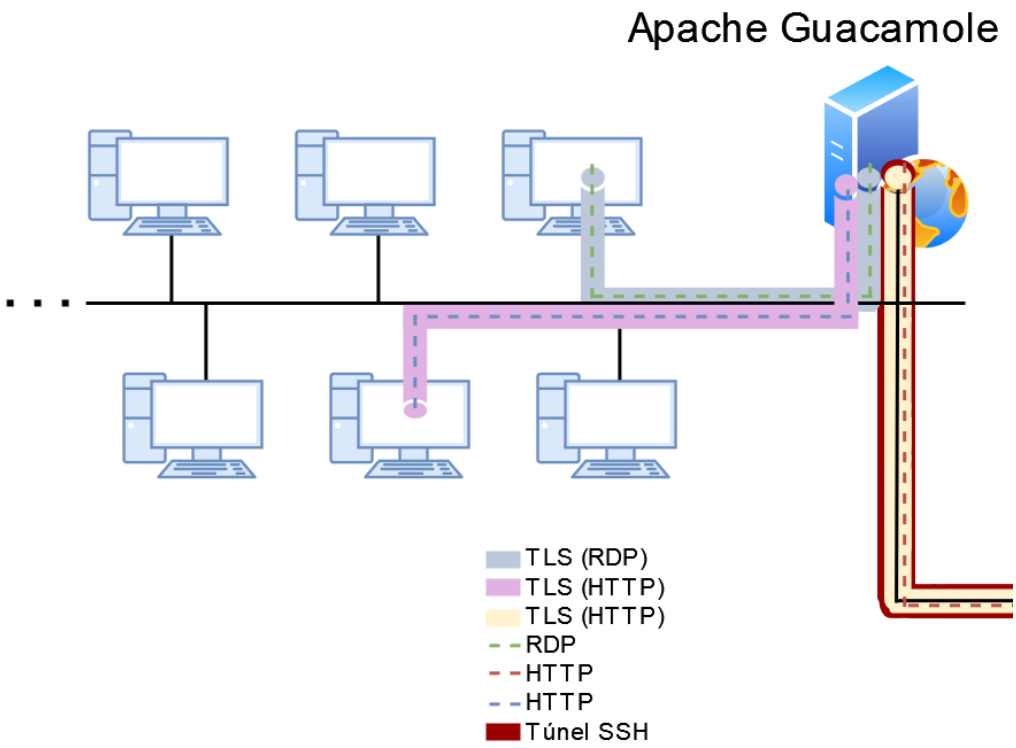

Figura 2-13. Esquema general – Ampliado empresa

El servidor de Apache Guacamole tendrá un túnel SSH con la instancia de Amazon AWS EC2, de manera que sobre éste viajarán peticiones y respuestas HTTP y HTTPS. Es así como expondrá sus puertos 80 (HTTP) y 443

(HTTPS) al exterior. De esta manera no habrá que tocar la configuración del router de la empresa para hacer *Port Forwarding*.

Por otro lado, el administrador se comunicará con Apache HTTP Server solo desde el interior de la empresa mediante HTTPS (tubo violeta) para realizar sus labores en el panel de administración.

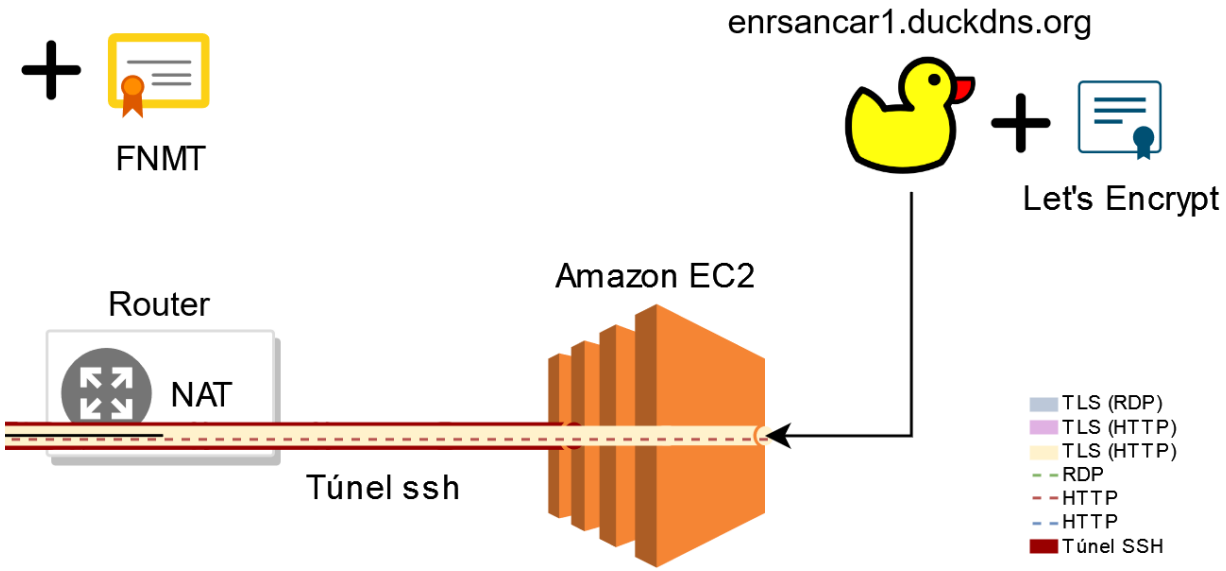

Figura 2-14. Esquema general – Ampliado AWS

Finalmente, utilizaremos DNS dinámico para tener siempre el mismo dominio apuntando a la instancia de Amazon AWS EC2 que despleguemos en cada momento, usando Terraform. Para ello usamos el servicio Duck DNS que ofrece una API de manera que con peticiones HTTP podemos modificar la IP a la que apunta el registro. Activaremos SSL en este domino con un certificado Let's Encrypt.

Cabe mencionar que el servidor de Apache Guacamole pedirá un certificado digital de la Fábrica de Moneda y Timbre (FNMT) para poder acceder a su uso. El administrador del sistema habrá configurado previamente en un panel de administración qué certificados aceptará.

3 IMPLEMENTACIÓN

Las ideas no duran mucho. Hay que hacer algo con ellas.

- Santiago Ramón y Cajal -

Dividimos la implementación en 4 bloques, el archivo de configuración de instalación, la instancia de Amazon AWS EC2, el servidor ubicado en la empresa y la desinstalación.

3.1 Configuración de instalación

Para personalizar la instalación tenemos un archivo llamado config en el que podremos modificar los siguientes valores:

- $USERNAME: es el usuario usado para el login en el panel de administración.
- $PASSWORD: es la contraeña del usuario usado para el login en el panel de administración.
- $PASSWORD_GUAC: es la contraseña que tendrá el usuario guacadmin de Apache Guacamole, administrador de la plataforma.
- $DOMAIN: dominio de Duck DNS a usar.
- $DUCKDNS_TOKEN: token para usar la API de Duck DNS.
- $AWS_SECRET_ACCESS_KEY: token para usar la API de AWS.
- $AWS_ACCESS_KEY_ID: token para usar la API de AWS.
- $ENCENDIDO: horario para hacer apply de terraform, en formato compatible con cron. Por ejemplo "0 7 * * mon" hace un apply todos los lunes a las 07:00.
- $APAGADO: horario para hacer apply de terraform, en formato compatible con cron. Por ejemplo "0 0 * * sun" hace un destroy todos los domingos a las 00:00.
- $EMPRESA: Nombre de la empresa, para la personalización del login de la web de inicio de sesión.
- $EMAIL: Email usado para solicitar el certificado usando Certbot y Let’s Encrypt.
- $ENDPOINT: Parte final de la url que extiende a https://rentry.co/. Por ejemplo, con tfgenrique quedaría https://rentry.co/tfgenrique.
- $CODE: Contraseña que permite modificar la nota de rentry.co.
- $NTPSERVER: Servidor para obtener la hora mediante el protocolo NTP.

Al script de instalación se le pasarán estas variables. El código completo se encuentra en el Anexo A.

3.2 Instancia AWS EC2

La instancia EC2 la usaremos para establecer un túnel SSH con el servidor de la empresa. De esta manera no habrá que configurar *Port Forwarding*.

3.2.1 Terraform

Para su creación instalamos Terraform en la Raspberry usando los siguientes bloques de código, los cuales pueden contener variables como por ejemplo ${AWS_ACCESS_KEY_ID}.

Código 3.1. install.sh – Instalación de Terraform

```
  #Instalo terraform usando el binario v1.1.9
  wget
https://releases.hashicorp.com/terraform/1.1.9/terraform_1.1.9_linux_arm64.zi
p
  unzip terraform_1.1.9_linux_arm64.zip
  cp terraform /usr/local/bin/
```

Guardamos las credenciales AWS_SECRET_ACCESS_KEY y AWS_ACCESS_KEY_ID, que obtenemos en el registro en Amazon AWS, en un fichero credentials el cual solo será legible por el usuario root. Además, hacemos un *source* de este archivo para poner estas credenciales como variables de entorno actuales.

Código 3.2. install.sh – Token AWS

```
  #Añadimos keys amazon ec2 al fichero credentials
  mkdir /root/.aws
  echo "export AWS_SECRET_ACCESS_KEY=${AWS_SECRET_ACCESS_KEY}" >>
/root/.aws/credentials
  echo "export AWS_ACCESS_KEY_ID=${AWS_ACCESS_KEY_ID}" >>
/root/.aws/credentials
  chmod 400 /root/.aws/credentials
  source /root/.aws/credentials
```

Ahora copiamos los archivos plantilla de Terraform y los modificamos según las variables de configuración del script.

Código 3.3. install.sh – Copiar código de la infraestructura para Terraform

```
  cp -r cron-ssh-tunnel /root/
  chmod 704 /root/cron-ssh-tunnel/scripts/tunnel.sh
  cd /root/cron-ssh-tunnel
  sed -i "s/DOMAIN/${DOMAIN}/" variables.tf
  sed -i "s/DUCKDNS_TOKEN/${DUCKDNS_TOKEN}/" variables.tf
```

Si vemos en detalle los archivos que usará Terraform para crear la instancia tendremos a variables.tf donde se encuentran valores que podemos cambiar para la instalación como el directorio de instalación, el nombre de la clave a generar para la conexión SSH, el dominio de Duck DNS a usar y el token de DuckDNS.

Código 3.4. variables.tf

```
variable "dir" {
  description = "Path to the working dir"
  type        = string
  default     = "/root/cron-ssh-tunnel"
}

variable "key_name" {
  description = "Name of the generated key"
  type        = string
  default     = "mi_clave"
}

variable "duckdns_domain" {
  description = "Duckdns domain, used to dynamic dns"
  type        = string
  default     = "DOMAIN"
}

variable "duckdns_token" {
  description = "Duckdns token, used to update dynamic dns"
  type        = string
  default     = "DUCKDNS_TOKEN"
}
```

Por otro lado, en el archivo main.tf se define la infraestructura a crear mediante código. Primero declaramos qué versión de Terraform y de proveedor AWS necesitamos.

Código 3.5. main.tf – Versiones Terraform y AWS provider

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 3.27"
    }
  }

  required_version = ">= 0.14.9"
}
```

Luego creamos un recurso para generar una clave RSA de 4096 bits y usarla en la conexión SSH que necesitaremos con la instancia. Mediante un *local-exec* guardaremos esta en el directorio ${self.tags.dir}/priv_key/ con el nombre ${self.key_name}.pem y le daremos los permisos 400 para que solo el usuario root pueda leerla.

Código 3.6. main.tf – Creación de clave RSA

```
#Clave RSA de 4096 bits
resource "tls_private_key" "rsa-4096" {
  algorithm = "RSA"
  rsa_bits  = 4096
}

resource "aws_key_pair" "generated_key" {
  key_name   = var.key_name
```

```
    public_key = tls_private_key.rsa-4096.public_key_openssh

    tags = {
      dir = var.dir
    }

    provisioner "local-exec" { #Guarda la clave privada generada
      command = <<EOT
      echo '${tls_private_key.rsa-4096.private_key_pem}' >
${self.tags.dir}/priv_key/${self.key_name}.pem
      chmod 400 ${self.tags.dir}/priv_key/${self.key_name}.pem
      EOT
    }
  }
```

Seguidamente declaramos un bloque *provider* llamado "aws" donde establecemos la región en la que quermos que la instancia esté ubicada.

Código 3.7. main.tf – Ubicación de la instancia

```
 provider "aws" {
   profile = "default"
   region  = "eu-west-3"
 }
```

Ahora creamos un *resource* llamado "aws_instance" "app_server" donde establecemos el *ami* a usar, es decir, la Imagen de Máquina de Amazon, que será Amazon Linux. El tipo de instancia será t2.micro ya que supone un bajo coste y entra en el plan gratuito.

Con associate_public_ip_address=true establecemos que se le asocie la IP pública de la máquina a este recurso. Luego con key_name asignamos la clave RSA anteriormente creada y con vpc_security_group_ids=["id-grupo"] hacemos uso del grupo que hemos configurado para que tenga los puertos 22, 80 y 443 abiertos.

Código 3.8. main.tf – Parámetros de la instancia AWS EC2

```
 resource "aws_instance" "app_server" {
   ami           = "ami-0960de83329d12f2f"
   instance_type = "t2.micro"
   associate_public_ip_address = true
   key_name = aws_key_pair.generated_key.key_name
   vpc_security_group_ids = ["sg-049fe496ee9eefa30"]
```

Si seguimos, nos encontramos con el bloque connection, el cual declara que realizaremos una conexión con esta instancia a través de SSH con el usuario "ec2-user" creado por AWS ya que está desaconsejado el inicio de sesión como root en las buenas prácticas de AWS [32]. Establecemos el uso de la clave creada y la IP pública asociada.

Código 3.9. main.tf – Conexión SSH

```
 connection {
     type     = "ssh"
     user     = "ec2-user"
     private_key = file("${self.tags.dir}/priv_key/${self.key_name}.pem")
     host     = self.public_dns
 }
```

Esta conexión la usaremos para ejecutar comandos remotos con el *provisioner "remote-exec"*. Primero modificamos la configuración del servicio *sshd* para que desde el lado del cliente se pueda especificar que los puertos que usemos para el túnel escuchen desde 0.0.0.0, ya que por defecto el túnel solo sería accesible desde *localhost*.

Luego mediante iptables establecemos que lo que le llegue a la instancia por el puerto 80 lo redirija al puerto 8080. Lo mismo para el puerto 443 y 4430. Esto se debe a que como el usuario que usamos para la conexión SSH no es superusuario, no podremos usar puertos <1024, por lo que los puertos de los túneles serán 8080 (para http) y 4430 (para https).

Por último, hacemos una petición a Duck DNS para que el registro DNS apunte a la IP de la instancia.

Código 3.10. main.tf – Ejecución remota de comandos

```
  provisioner "remote-exec" {
      inline = [
        "sudo chmod 606 /etc/ssh/sshd_config",
        "echo 'GatewayPorts clientspecified' >> /etc/ssh/sshd_config",
        "sudo service sshd restart",
        "sudo iptables -t nat -A PREROUTING -p tcp --dport 80 -j REDIRECT --
to-ports 8080", #redirijo el puerto 80 al 8080 (donde está el tunel ssh)
        "sudo iptables -t nat -A PREROUTING -p tcp --dport 443 -j REDIRECT --
to-ports 4430", #redirijo el puerto 443 al 4430 (donde está el tunel ssh)
        "echo
url='https://www.duckdns.org/update?domains=${var.duckdns_domain}&token=${var
.duckdns_token}&ip=' | curl -k -K -", #actualizo el dns dinamico
      ]
  }
```

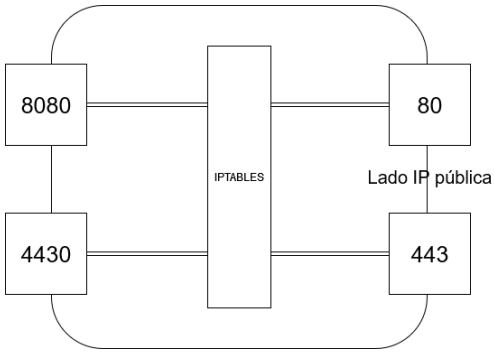

Figura 3-1. Instancia AWS EC2 – Redirección de puertos

Lo siguiente es un *provisioner "local-exec"* que se encarga de crear un archivo llamado tunnel-cron-job. Este configura una tarea para que cron ejecute cada minuto un script que se encarga de comprobar si el túnel SSH a la instancia está creado y funcionando, si no, lo reinicia.

Código 3.11. main.tf – Ejecución de comandos en local, cron para el túnel SSH

```
  provisioner "local-exec" {
      command = <<EOT
        sudo echo -e
"SHELL=/bin/bash\nPATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/loca
l/sbin\n\n* * * * * root export HOST=${self.public_dns} ; export
KEY=${self.tags.dir}/priv_key/${self.key_name}.pem ; export
DIR=${self.tags.dir} ; ${self.tags.dir}/scripts/tunnel.sh > /dev/null 2>
${self.tags.dir}/log/error.log" > /etc/cron.d/tunnel-cron-job
```

```
        export HOST=${self.public_dns} ; export
KEY=${self.tags.dir}/priv_key/${self.key_name}.pem ; export
DIR=${self.tags.dir} ;  sudo ${self.tags.dir}/scripts/tunnel.sh > /dev/null
2> ${self.tags.dir}/log/error.log
        EOT
 }
```

Si vemos dicho script en detalle, contiene una función tunnel() que establece un túnel SSH con la instancia remota para que el puerto remoto 8080 sea el 80 de nuestro servidor, y lo mismo con el puerto remoto 4430 y el 443. Además, establece que el puerto local 6000 sea el 22 remoto.

En la siguiente línea vemos que intentamos ejecutar el comando *ls* usando el túnel y si este da error, el túnel se ha caído, por lo que matamos dicho proceso y lo volvemos a crear. Tendremos así un túnel que se reinicia cuando falla.

Código 3.12. tunnel.sh – Túnel SSH reiniciable automáticamente

```
  tunnel() {
  #Crea tunel entre 4430(remoto):443(local) y 8080(remoto):80(local)
  #Además de un tunel entre 6000(local):22(remoto) para comprobar si el tunel
sigue activo

  ssh -i $KEY ec2-user@$HOST -R 0.0.0.0:8080:0.0.0.0:80 -R
0.0.0.0:4430:0.0.0.0:443 -L localhost:6000:$HOST:22 -N -o
StrictHostKeychecking=no &

  echo $! > $DIR/tunnel.pid
  return 0
 }


  ssh -i $KEY ec2-user@localhost -p 6000 -o StrictHostKeychecking=no ls

  #Si la ejecucion remota del comando ls falla, vuelve a crear el tunel
  if [[ $? -ne 0 ]]; then
      echo creando tunel
      kill -9 `cat "$DIR/tunnel.pid"`
      rm $DIR/tunnel.pid
      tunnel
  fi
```

Finalmente, creamos otro *provisioner* "local-exec" que actúa solo en la destrucción de la infraestructura. Este se encarga de eliminar el archivo tunnel-cron-job creado por el anterior *provisioner*, de esta manera solo se ejecutará la tarea si la instancia está creada.

Código 3.13. main.tf – Ejecución de comandos en local, destrucción de la instancia

```
  provisioner "local-exec" {
      when = destroy

      command = <<EOT
        sudo ssh-keygen -f "/root/.ssh/known_hosts" -R "[localhost]:6000"
        sudo rm /etc/cron.d/tunnel-cron-job
        EOT

      on_failure = continue  #destruye la instancia ec2 aunque el comando
anterior devuelva un error
 }
```

También borra el localhost:6000 de los *known_host* ya que como en el script.sh ejecutamos *ls* a través del puerto 6000 del localhost, si la instancia cambia, cambiará la firma manteniendo el mismo nombre. Esto hace que salte una advertencia de que el host ha cambiado.

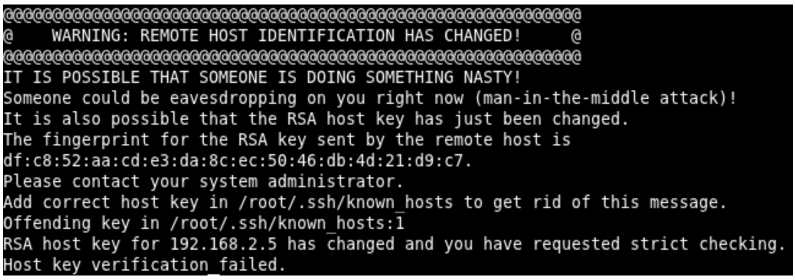

Figura 3-2. Advertencia SSH identificación host remoto

Siguiendo con la instalación, para descargar las dependencias necesarias para la ejecución de Terraform, hacemos *terraform init*. Ahora ya podremos hacer *terraform apply* y *terraform destroy* en el servidor Ubuntu, y así crear y destruir la infraestructura descrita.

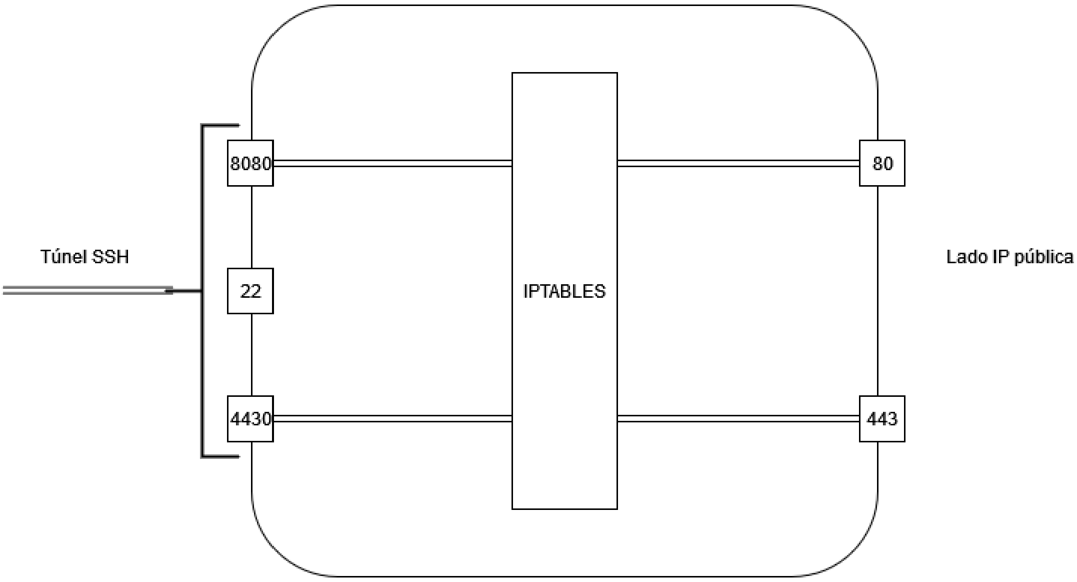

Figura 3-3. Instancia AWS EC2 – Redirección de puertos y túnel SSH

3.2.2 **Tareas Cron**

Adicionalmente creamos una tarea para que cron haga *apply* y *destroy* automáticamente, según el día de la semana que se establezca en el archivo de configuración (${ENCENDIDO} y ${APAGADO}). Así ahorraremos coste ya que usaremos menos tiempo la instancia EC2. Esto es muy útil puesto que muchas empresas no necesitan este servicio por ejemplo los domingos.

Código 3.14. install.sh – Tareas cron encendido y apagado

```
  sudo echo -e
"SHELL=/bin/bash\nPATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/loca
l/sbin\n\n${ENCENDIDO} root source /root/.aws/credentials; cd /root/cron-ssh-
tunnel; terraform apply --auto-approve > /dev/null 2> /root/cron-ssh-
tunnel/log/apply-destroy-error.log" > /etc/cron.d/terraform-apply

  sudo echo -e
"SHELL=/bin/bash\nPATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/loca
```

```
l/sbin\n\n${APAGADO} root source /root/.aws/credentials; cd /root/cron-ssh-
tunnel; terraform destroy --auto-approve > /dev/null 2> /root/cron-ssh-
tunnel/log/apply-destroy-error.log" > /etc/cron.d/terraform-destroy
```

Y por útilmo, añadimos un servicio de systemd para que cada vez que se inicie la raspberry, tanto en reinicio como en inicio desde apagado, se haga un *terraform apply*. Conseguimos así que, con tan solo encender el servidor en la empresa con una conexión a internet, se monte todo el sistema. Se hace así, ya que systemd permite tener como requisito que el servicio de red esté funcionando antes de ejecutar estos comandos (*After=network-online.target*), de otra manera, podríamos tener un error de red.

Adicionalmente, anteponemos el comando *ntpdate -b ${NTPSERVER}* para que la hora sea la correcta en cada inicio de la raspberry, y así funcionen correctamente tanto las marcas de tiempo de Terraform como las tareas de cron.

Código 3.15. install.sh – Servicio systemd para encendido del sistema en cada @reboot

```
  sudo echo -e "ntpdate -b ${NTPSERVER}; source /root/.aws/credentials; cd
/root/cron-ssh-tunnel; terraform apply --auto-approve > /dev/null 2>
/root/cron-ssh-tunnel/log/apply-startup-error.log" > /usr/bin/terraform-
apply-startup.sh

  chmod +x /usr/bin/terraform-apply-startup.sh
  cp terraform-apply-startup.service /lib/systemd/system/terraform-apply-
startup.service
  cp terraform-apply-startup.service /etc/systemd/system/terraform-apply-
startup.service
  chmod 644 /etc/systemd/system/terraform-apply-startup.service
  systemctl enable terraform-apply-startup
```

Que pueda haber varios *terraform apply* seguidos no supone ningún problema puesto que si la infraestructura está creada no la vuelve a crear.

3.3 Servidor ubicado en la empresa

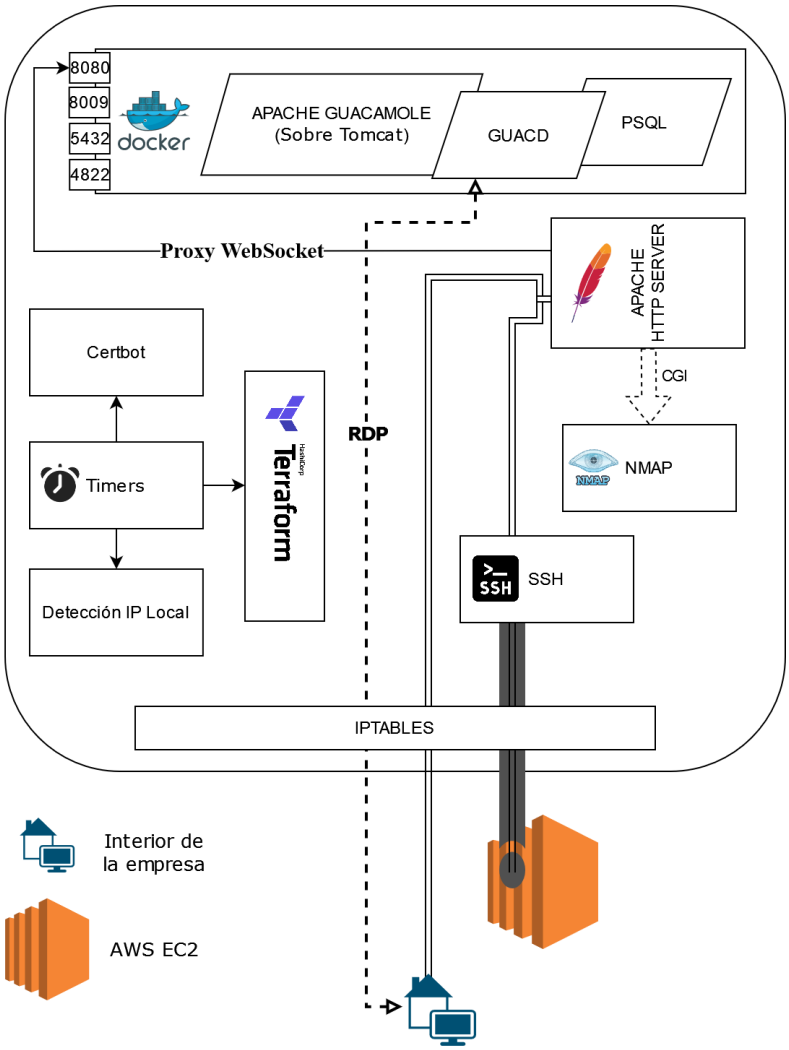

Figura 3-4. Servidor ubicado en la empresa

Dentro de este dispondremos de varios componentes:

3.3.1 Firewall

El firewall lo configuramos usando el comando iptables para la cadena INPUT, es decir, afectará a los paquetes para los que somos el destino. Hacemos DROP de los paquetes dirigidos a los puertos del contenedor Docker (8080 – HTTP Proxy, 8009 – AJP13, 5432 – PSQL, 4822 – Guacd) que no se generen en la interfaz local y aceptaremos todo lo que se genere en esta.

Código 3.16. install.sh – Firewall con Iptables

```
 iptables -A INPUT -p tcp ! -i lo --dport 8080 -j DROP
 iptables -A INPUT -p tcp ! -i lo --dport 8009 -j DROP
 iptables -A INPUT -p tcp ! -i lo --dport 5432 -j DROP
 iptables -A INPUT -p tcp ! -i lo --dport 4822 -j DROP
 iptables -A INPUT -p all -i lo -j ACCEPT
 iptables -A INPUT -p tcp --dport 22 -j ACCEPT
 iptables -A INPUT -p tcp --sport 22 -j ACCEPT
 iptables -A INPUT -p tcp --dport 80 -j ACCEPT
 iptables -A INPUT -p tcp --dport 443 -j ACCEPT
 iptables -A INPUT -p tcp --sport 3389 -j ACCEPT
 iptables -A INPUT -p tcp --sport 80 -j ACCEPT
 iptables -A INPUT -p tcp --sport 443 -j ACCEPT
 iptables -A INPUT -p udp --sport 53 -j ACCEPT
 iptables -A INPUT -p udp --sport 123 -j ACCEPT
 iptables -P INPUT DROP
```

Seguidamente aceptamos paquetes TCP con destino puerto 22 (SSH que usamos para acceder al servidor ubicado en la empresa), paquetes con puerto origen 22 (los que nos envía la instancia AWS EC2 mediante SSH), paquetes con destino puerto 80 y 443 para HTTP y HTTPS, paquetes con origen puerto 3389 (los generados por RDP en los dispositivos de la empresa). Aceptaremos paquetes TCP con puerto origen 80 y 443 ya que Terraform los necesita [33] y los paquetes UDP con puerto 53 (DNS) y puerto 123 (Network Time Protocol).

Finalmente, descartamos los paquetes que no cumplan ninguna de estas reglas con la política DROP.

Para que estas reglas sean persistentes, instalamos iptables-persistent y hacemos una copia de ellas a la ubicación indicada.

Código 3.17. install.sh – Persistir las reglas de Iptables

```
 #Instalo iptables-persistent para que cargue en el proximo inicio las reglas
iptables automaticamente
 apt-get install -y iptables-persistent
 iptables-save > /etc/iptables/rules.v4
```

3.3.2 Contenedor Docker de oznu/guacamole

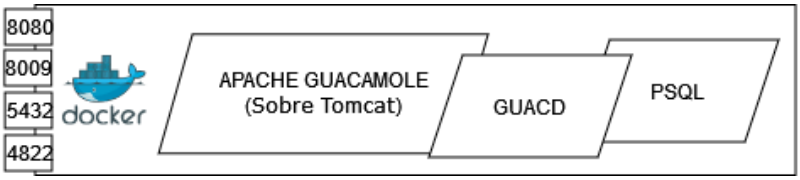

Figura 3-5. Contenedor de oznu/guacamole

El contenedor Docker de oznu/guacamole [34], se trata de un contenedor con Apache Guacamole listo para funcionar, es decir, dispone de Apache Guacamole sobre un servidor Tomcat conectado con el proxy Guacd y con una base de datos PostgreSQL.

El servidor Tomcat escucha peticiones en los puertos 8080 (http-proxy) y 8009 (ajp13), y se conecta a Guacd por el puerto 4822 y a la base de datos por el 5432 [35].

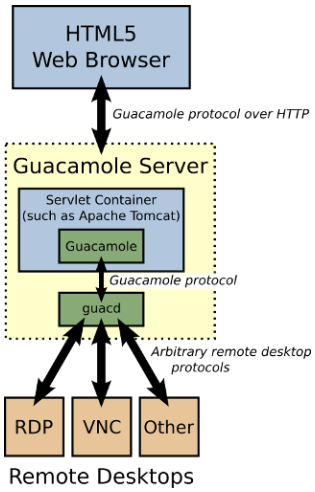

Figura 3-6. Apache Guacamole

Para crear este contenedor descargamos Docker y ejecutamos el siguiente bloque de código. Añadimos *--restart always* para que el contenedor se inicie automáticamente.

Código 3.18. install.sh – Descargar, instalar y poner en marcha el contenedor oznu/guacamole

```
  #Instalo docker
  curl -fsSL https://get.docker.com -o get-docker.sh
  sh ./get-docker.sh

  #oznu copyright - GNU License
  docker run -d -v /home/guacamole/oznu_config:/config --network host --
restart always --name guacamole oznu/guacamole:armhf
```

Cabe destacar que usamos el parámetro *--network host* debido a que, si no lo hacemos, Docker aísla el contenedor en otra subred y esto haría que los mensajes Wake on Lan enviados por Apache Guacamole no llegasen a su destino. Esto se debe a que se envían como difusión a nivel de subred [36]. Así el contenedor Docker compartirá tanto IP como puertos con la máquina host.

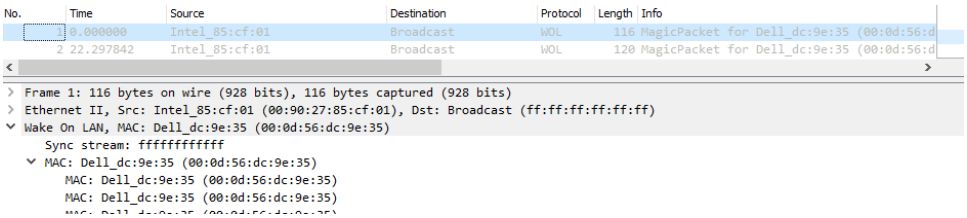

Figura 3-7. Wireshark – Paquete Wake on Lan

Para la configuración disponemos del directorio /home/guacamole/oznu_config. Los cambios realizados en este se verán reflejados en el directorio /config del contenedor.

Finalmente, le damos un toque de personalización a Apache Guacamole usando una extensión que cambia el nombre y logo de la web. El nombre será uno de los valores del archivo de configuración y el logo habrá que ubicarlo en la carpeta guacamole-branding-example/resources/images/logo/logo.png. Este código empaqueta la carpeta en un archivo zip con la extensión .jar y el contenedor de Apache Guacamole lo cargará en el próximo inicio.

Código 3.19. install.sh – Branding personalizado

```
  #Añado el branding personalizado
  apt install zip
  docker stop guacamole
  cd guacamole-branding-example
  sed -i "s/EMPRESA/${EMPRESA}/" translations/en.json
  zip -r guacamole-branding-example.jar *
  cd ..
  mv guacamole-branding-example/guacamole-branding-example.jar
/home/guacamole/oznu_config/guacamole/extensions/
  docker start guacamole
```

3.3.3 Apache HTTP Server

El servidor Apache HTTP tendrá configurado:

- en la raíz / reenvía las peticiones al servidor Tomcat.
- /instrucciones sirve contenido estático de los pasos a seguir en un dispositivo de la empresa para que funcione remotamente.
- usando la IP local del servidor se accede al login para del panel de administración.

3.3.3.1 Instalación

Instalamos el servidor Apache HTTP usando *apt* y activamos los módulos para hacer funcionar Apache como un proxy inverso, de manera que las peticiones que le lleguen a la raíz, las reenvíe al contenedor Docker por el puerto 8080 donde estará escuchando Tomcat.

Código 3.20. install.sh – Instalar Apache HTTP Server y módulos

```
  #Instalo apache2
  apt-get install apache2 -y

  #activo mod_proxy y mod_proxy_wstunnel
  a2enmod proxy
  a2enmod proxy_http
  a2enmod proxy_wstunnel
```

Como podemos observar en el bloque de código, para el reenvío hacia el servidor tomcat usamos *mod_proxy* en su versión WebSocket (*proxy_wstunnel*), y si el navegador que use el cliente no lo soporta, *mod_proxy* en su versión HTTP (*proxy_http*).

Habitualmente para interconectar Apache HTTP Server y Apache Tomcat se elige *mod_jk* o *mod_proxy_ajp* por encima de *mod_proxy_http*, ya que usan el protocolo AJP, un protocolo orientado a bytes más eficiente que las peticiones HTTP para estos casos. Sin embargo, según la documentación de Apache Guacamole, éste no es compatible con AJP, por lo que, si nuestro navegador no soporta WebSockets, habrá que usar el proxy HTTP [37].

De todas formas, los WebSockets se introdujeron en el año 2010 así que la inmensa mayoría de los navegadores web son compatibles.

3.3.3.2 Certificado para HTTPS

El siguiente paso es obtener el certificado Let's Encrypt mediante Certbot. Para ello instalamos certbot usando snap y luego solicitamos el certificado para nuestro domino, con el email de la configuración y la opción --apache. De manera automática, certbot guarda el certificado en /etc/letsencrypt/live y pone una tarea para la renovación del mismo.

Código 3.21. install.sh – Instalar Certbot y pedir certificado

```
snap install core
snap refresh core
snap install --classic certbot
ln -s /snap/bin/certbot /usr/bin/certbot
certbot --apache -d $DOMAIN --non-interactive --agree-tos --email ${EMAIL}
```

Es importante haber realizado primero los pasos del apartado anterior Instancia de Amazon AWS EC2, puesto que, si no tenemos el túnel funcionando y el DNS apuntando a este, no podremos solicitar el certificado ya que Certbot hace una petición a nuestro servidor para comprobar la autenticidad.

La opción de --apache también nos generará dos archivos de configuración de Apache en el directorio /etc/apache2/sites-enabled, ambos con un *VirtualHost* dentro. El primero, 000-default.conf será un *VirtualHost* para el puerto 80 que reenviará mediante un 301 las peticiones de http a https. El segundo, 000-default-le-ssl.conf será un *VirtualHost* para el puerto 443 con el dominio solicitado como *ServerName*.

3.3.3.3 Configuración de proxy inverso

Al archivo de configuración 000-defualt-le-ssl.conf le añadiremos nuestra configuración.

Código 3.22. install.sh – Configuración del *VirtualHost* que gestiona el puerto 443

```
echo -e "
SSLVerifyClient require
SSLVerifyDepth 2
SSLCACertificateFile /etc/apache2/ca/fnmt.pem
<Location />
    ProxyPass http://127.0.0.1:8080/ flushpackets=on
    ProxyPassReverse http://127.0.0.1:8080/
    Require expr \"(%{SSL_CLIENT_S_DN_CN} in {''})\"
    Options -Indexes
</Location>
<Location /websocket-tunnel>
    ProxyPass ws://127.0.0.1:8080/websocket-tunnel
    ProxyPassReverse ws://127.0.0.1:8080/websocket-tunnel
    Require expr \"(%{SSL_CLIENT_S_DN_CN} in {''})\"
    Options -Indexes
</Location>
<Location /instrucciones>
    ProxyPass !
    Require expr \"(%{SSL_CLIENT_S_DN_CN} in {''})\"
    Options -Indexes
</Location>

SetEnvIf Request_URI \"^/tunnel\" dontlog
CustomLog  /var/log/apache2/guac.log common env=!dontlog

</VirtualHost>
</IfModule>" >> /etc/apache2/sites-enabled/000-default-le-ssl.conf
```

Si vamos por partes, estas 3 directivas junto con la siguiente activan la autenticación del cliente mediante certificado digital, en este caso se pide un certificado digital expedido por la FNMT de profundidad 2, como por ejemplo este:

Código 3.23. 000-default-le-ssl.conf – Activar verificación del cliente con certificado digital

```
SSLVerifyClient require
SSLVerifyDepth 2
SSLCACertificateFile /etc/apache2/ca/fnmt.pem
```

Código 3.24. 000-default-le-ssl.conf – Aceptar solo los certificados de la lista

```
Require expr \"(%{SSL_CLIENT_S_DN_CN} in {''})\"
```

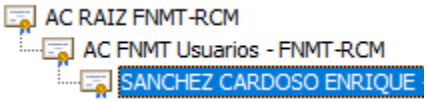

Figura 3-8. Certificado digital FNMT de profundidad 2

Luego tenemos 3 *Location* que definen cuándo reenviar las peticiones a Apache Tomcat. Los dos primeros reenvían las peticiones a Tomcat (127.0.0.1:8080), usando HTTP como proxy para el primer caso y WebSockets para el segundo. Apache Guacamole se encarga de comprobar si el navegador es compatible con WebSockets y usarlos, si no, escoge la otra opción.

El último *Location* establece que no se reenvíen las peticiones a la url /instrucciones puesto será contenido estático servido por Apache HTTP Server.

Código 3.25. 000-default-le-ssl.conf – Bloques *Location*

```
<Location />
    ProxyPass http://127.0.0.1:8080/ flushpackets=on
    ProxyPassReverse http://127.0.0.1:8080/
    Require expr \"(%{SSL_CLIENT_S_DN_CN} in {''})\"
    Options -Indexes
</Location>
<Location /websocket-tunnel>
    ProxyPass ws://127.0.0.1:8080/websocket-tunnel
    ProxyPassReverse ws://127.0.0.1:8080/websocket-tunnel
    Require expr \"(%{SSL_CLIENT_S_DN_CN} in {''})\"
    Options -Indexes
</Location>
<Location /instrucciones>
    ProxyPass !
    Require expr \"(%{SSL_CLIENT_S_DN_CN} in {''})\"
    Options -Indexes
</Location>
```

Copiamos el contenido estático de /instrucciones al directorio /var/www/html.

Código 3.26. install.sh – Copiar el contenido estático de /instrucciones

```
cp -r html/instrucciones/ /var/www/html/instrucciones/
```

Finalmente, desactivamos el *logging* de peticiones que usen el proxy HTTP. Esto es recomendable ya que serían muchas líneas de log lo que podría ralentizar el servicio según la documentación oficial [37].

Código 3.27. 000-default-le-ssl.conf – Desactivar *logging* de peticiones del proxy

```
  SetEnvIf Request_URI \"^/tunnel\" dontlog
  CustomLog  /var/log/apache2/guac.log common env=!dontlog
```

3.3.3.4 Panel de administración

Lo siguiente que haremos es crear el panel de administración. Este será únicamente accesible desde la red de la empresa, es decir, se accederá a través de la IP privada del servidor. Esto le da un extra de seguridad ya que para que alguien externo intente usarlo, primero necesitará colarse en la red local. Además, estará protegido por un *login*.

Para que las credenciales de este *login* no sean visibles para el resto de usuarios de la red local, usaremos HTTPS con un certificado autofirmado porque no se le puede solicitar a una AC un certificado cuyo dominio es una dirección IP privada.

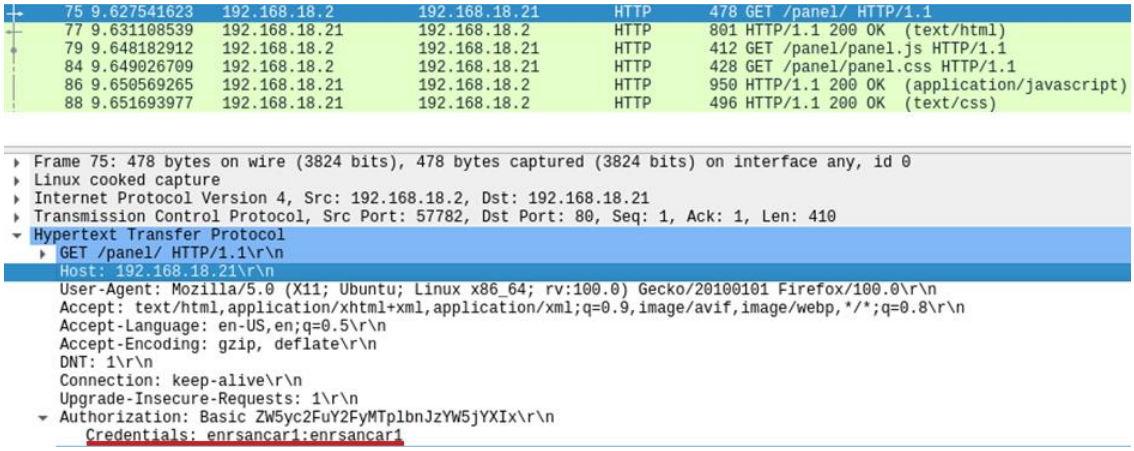

Figura 3-9. Wireshark – Credenciales en texto plano al usar HTTP

Ponemos que este certificado expire en 9999 días ya que es autofirmado y no queremos tenerlo que renovar.

Código 3.28. install.sh – Generar certificado autofirmado para el panel

```
  openssl req -x509 -nodes -days 9999 -newkey rsa:2048 -subj
"/CN=${IP_LOCAL}/C=ES" -keyout /etc/ssl/private/apache-selfsigned.key -out
/etc/ssl/certs/apache-selfsigned.crt
```

Ahora activamos el *mod_auth_basic* y creamos el archivo donde se guardará usuario y contraseña del sistema de *login*.

Código 3.29. install.sh – Activar *login* y crear usuario

```
#Activamos mod_auth_basic
a2enmod auth_basic

#Creo el archivo que almacena la contraseña del panel y cgi
mkdir -p /usr/local/apache/passwd
htpasswd -bc /usr/local/apache/passwd/passwords $USERNAME $PASSWORD
```

Copiamos el archivo de configuración para este panel a /etc/apache2/sites-enabled y colocamos la IP_LOCAL donde corresponde usando *sed*. Esta IP_LOCAL se obtiene con el siguiente comando.

Código 3.30. Obtener la IP local y modificar la configuración del panel

```
IP_LOCAL=$(ip route get 8.8.8.8 | head -1 | cut -d' ' -f7)

cp apache_conf/panel.conf /etc/apache2/sites-enabled/

#Sustituimos IP_LOCAL de la plantilla por la ip local de la maquina
sed -i "s/IP_LOCAL/${IP_LOCAL}/" /etc/apache2/sites-enabled/panel.conf
```

El archivo panel.conf establece la IP:puerto con el que se accede al *VirtualHost* y con *DocumentRoot* /var/www/panel el directorio donde se encuentran los archivos que se servirán. Las directivas SSL hacen que se use el certificado autofirmado anteriormente para HTTPS.

La configuración del directorio a servir establece que solo se acepten conexiones entrantes cuya IP sea del rango privado y que además completen correctamente el *login* creado.

Código 3.31. panel.conf – Certificado y condiciones para acceder al panel

```
<IfModule mod_ssl.c>
<VirtualHost IP_LOCAL:443>
    ServerAdmin webmaster@localhost
    DocumentRoot /var/www/panel
    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined

    ServerName IP_LOCAL

    SSLEngine on
    SSLCertificateFile /etc/ssl/certs/apache-selfsigned.crt
    SSLCertificateKeyFile /etc/ssl/private/apache-selfsigned.key

    <Directory /var/www/panel>
        <RequireAll>
            <RequireAny>
              Require ip 192.168
              Require ip 172.16.0.0/12
              Require ip 10
            </RequireAny>
            AuthType Basic
            AuthName "Panel de administración"
            AuthUserFile "/usr/local/apache/passwd/passwords"
            Require valid-user
        </RequireAll>
    </Directory>
```

3.3.3.5 Redirección de HTTP a HTTPS

Como estamos creando *VirtualHosts* a los que se accede por el puerto 443 (HTTPS), establecemos que si se intenta entrar por el puerto 80 (HTTP) se le redireccione a HTTPS. Para ello añadimos la siguiente configuración en 000-default.conf.

Código 3.32. install.sh – Redirección de HTTP a HTTPS para el panel

```
  #Activamos el Rewrite para IP LOCAL
  sed -i "s/<\/VirtualHost>/ RewriteEngine on \n\
  RewriteCond %{SERVER_NAME} =${IP_LOCAL} \n\
  RewriteRule ^ https:\/\/%{SERVER_NAME}%{REQUEST_URI} [END,NE,R=permanent]
\n\
  <\/VirtualHost>/" /etc/apache2/sites-enabled/000-default.conf
```

Quedando el achivo de configuración de la siguiente manera. El primer *Rewrite* para la redirección fue añadido automáticamente por Cerbot cuando se instaló el certificado con la opción --apache.

Código 3.33. 000-default.conf – Redirección de HTTP a HTTPS

```
  <VirtualHost *:80>
    ServerAdmin webmaster@localhost
    DocumentRoot /var/www/html

    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined

    RewriteEngine on
    RewriteCond %{SERVER_NAME} =enrsancar1.duckdns.org
    RewriteRule ^ https://%{SERVER_NAME}%{REQUEST_URI} [END,NE,R=permanent]
    RewriteEngine on
    RewriteCond %{SERVER_NAME} =192.168.18.21
    RewriteRule ^ https://%{SERVER_NAME}%{REQUEST_URI} [END,NE,R=permanent]
  </VirtualHost>
```

Código 3.34. install.sh – Reinicio de Apache HTTP Server

```
  #Restart para que apache use la nueva configuracion y se conecte a tomcat
  service apache2 restart
```

Finalmente reiniciamos Apache HTTP Server para que cargue toda la configuración.

En el Anexo A se encuentran todos los archivos usados en este proceso, junto con los scripts de instalación y desinstalación.

3.3.4 Detección del cambio de IP local

Durante el tiempo de funcionamiento de este sistema, será probable que la IP local que tenga asociada dentro de la Red LAN cambie. Esto puede deberse a varios motivos, el uso de DHCP, el cambio del router de la oficina, etc. Incluso puede que se instale y se prepare en un entorno, y luego se ubique en la oficina para hacerlo funcionar. Por todo esto, y debido a que la configuración depende de la IP local, se crea este script de detención del cambio.

Usaremos una tarea cron que se ejecute cada minuto para comprobar si la IP local ha cambiado. Si lo ha hecho, se cambia la configuración del panel ya que usa esta IP para el *VirtualHost* y la configuración del *VirtualHost* que gestiona el puerto 80 (000-default.conf) para cambiar la redirección a HTTPS. Además, generará un nuevo certificado autofirmado para el SSL del panel, puesto que ha cambiado la url con la que se accede.

Finalmente, usando la API de rentry.co publicará esta IP en la nota configurada según el archivo config. Esto servirá para que el administrador conozca en todo momento dicha IP, por ejemplo, accediendo a https://rentry.co/tfgenrique.

Código 3.35. postIp.sh – Detección del cambio de IP local

```
  #/bin/bash

  OLD=$(cat /usr/local/bin/rentryco/ip)
  IP=$(ip route get 8.8.8.8 | head -1 | cut -d' ' -f7)

  #Si la ip ha cambiado y tiene más de 7 caracteres
  if [[ $OLD != $IP  ]]  && (( ${#IP} > 7)); then
  echo -n "Antigua IP local:"
  echo $OLD
  echo -n "Nueva IP local:"
  echo $IP

  sed -i "s/${OLD}/${IP}/" /etc/apache2/sites-enabled/panel.conf
  sed -i "s/${OLD}/${IP}/" /etc/apache2/sites-enabled/000-default.conf
  openssl req -x509 -nodes -days 9999 -newkey rsa:2048 -subj "/CN=${IP}/C=ES"
-keyout /etc/ssl/private/apache-selfsigned.key -out /etc/ssl/certs/apache-
selfsigned.crt

  service apache2 reload

  /usr/local/bin/rentryco/rentry edit -u tfgenrique -p enrsancar1
"https://${IP}/"

  echo $IP > /usr/local/bin/rentryco/ip

  else
  echo "La ip local no ha cambiado"

  fi
```

Para la instalación, *python3* será una dependencia de rentry.co. Copiamos tanto el script python3 de rentry.co, como postIp.sh al directorio /usr/local/bin, modificando los valores configurables del ENDPOINT y CODE. Finalmente, otorgamos permiso de ejecución solo a root y creamos el archivo iplocal en el directorio /etc/cron.d. Este archivo define una tarea para que cron ejecute el script post.sh cada minuto.

Código 3.36. install.sh – Instalar detección del cambio de IP local

```
  #Instalar python3
  apt install python3
  #Rentry.co para saber la ip local
  cp -r rentryco/ /usr/local/bin/
  sed -i "s/ENDPOINT/${ENDPOINT}/" /usr/local/bin/rentryco/postIp.sh
  sed -i "s/CODE/${CODE}/" /usr/local/bin/rentryco/postIp.sh
  chmod 744 /usr/local/bin/rentryco/rentry
  chmod 744 /usr/local/bin/rentryco/postIp.sh
  bash /usr/local/bin/rentryco/postIp.sh
```

```
  echo -e
"SHELL=/bin/bash\nPATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/loca
l/sbin\n\n* * * * * root bash /usr/local/bin/rentryco/postIp.sh" >>
/etc/cron.d/iplocal
```

3.3.5 CGI para el panel de administración

Activamos el CGI que sirve el contenido dinámico para la administración del sistema, como puede ser el modificar los certificados que son aceptados para acceder a la plataforma o usar nmap para identificar los equipos con un servidor RDP en marcha. Este CGI estará también protegido por las mismas condiciones que el panel de administración, ya que solo lo podrá usar el usuario administrador. Añadimos el siguiente bloque a panel.conf.

Código 3.37. panel.conf – Configuración de cgi

```
  <IfDefine ENABLE_USR_LIB_CGI_BIN>
      ScriptAlias /cgi-bin/ /usr/lib/cgi-bin/
      <Directory "/usr/lib/cgi-bin">
          AllowOverride None
          Options +ExecCGI -MultiViews
          <RequireAll>
              <RequireAny>
                  Require ip 192.168
                  Require ip 172.16.0.0/12
                  Require ip 10
              </RequireAny>
              AuthType Basic
              AuthName "Panel de administración"
              AuthUserFile "/usr/local/apache/passwd/passwords"
              Require valid-user
          </RequireAll>
      </Directory>
  </IfDefine>

  </VirtualHost>
  </IfModule>
```

Para poder usar el CGI que hemos establecido en la configuración panel.conf instalamos *nmap, xmllint* (filtrará la salida de nmap) y *python3*, activamos el *mod_cgi* y modificamos la configuración por defecto ya que esta activa CGI para todos los *VirtualHost* y solo nos interesa para el panel de administración.

Código 3.38. install.sh – Instalar nmap, xmllint, python3 y activar el mod_cgi

```
  #Instalamos nmap y xmllint
  apt-get install nmap -y
  apt install libxml2-utils -y

  #Instalamos las dependencias que usan los scripts de python3
  apt install python3 python3-pip
  pip3 install psycopg2

  #Activamos cgi
  a2enmod cgi
  cp apache_conf/serve-cgi-bin.conf /etc/apache2/conf-enabled/
```

Ahora copiamos los scripts CGI que se encargan de las funciones del panel mencionadas y le damos permisos de ejecución para el usuario www-data que es el que ejecuta Apache HTTP Server. Estos scripts están diseñados para que no hagan falta permisos root para ejecutarlos.

Código 3.39. install.sh – Localización y permisos de archivos cgi

```
  cp cgi-bin/leer.cgi /usr/lib/cgi-bin/
  cp cgi-bin/recibir.cgi /usr/lib/cgi-bin/
  cp cgi-bin/nmap.cgi /usr/lib/cgi-bin/
  cp cgi-bin/crear.cgi /usr/lib/cgi-bin/
  cp cgi-bin/crear_todas.cgi /usr/lib/cgi-bin/
  cp cgi-bin/sincronizar_usuarios.sh /etc/apache2/
  cp cgi-bin/crear_usuario.py /etc/apache2/
  cp cgi-bin/crear_conexion.py /etc/apache2/
  cp cgi-bin/crear_conexion_todas.py /etc/apache2/

  chown root:www-data /usr/lib/cgi-bin/leer.cgi
  chown root:www-data /usr/lib/cgi-bin/recibir.cgi
  chown root:www-data /usr/lib/cgi-bin/nmap.cgi
  chown root:www-data /usr/lib/cgi-bin/crear.cgi
  chown root:www-data /usr/lib/cgi-bin/crear_todas.cgi

  chmod 750 /usr/lib/cgi-bin/leer.cgi
  chmod 750 /usr/lib/cgi-bin/recibir.cgi
  chmod 750 /usr/lib/cgi-bin/nmap.cgi
  chmod 750 /usr/lib/cgi-bin/crear.cgi
  chmod 750 /usr/lib/cgi-bin/crear_todas.cgi
```

Un caso especial de estos scripts es sincronizar_usuarios.sh, el cual se encarga de modificar los certificados que acepta Apache para la autenticación del cliente. Para esto, tendrá que modificar los archivos de configuración y recargar el servicio, algo solo disponible para permisos de superusuario.

Inicialmente se pensó que para modificar la lista de certificados no haría falta permisos de super usuario y se podría usar la directiva file() de Apache, pero esta directiva solo sirve para cargar strings y no elementos de lista, por lo que habrá que modificar la configuración con un script para este cometido. No nos supone ningún problema puesto que hacer que www-data ejecute un script como super usuario iba a ser igualmente necesario debido al *service reload* para cargar la configuración actualizada.

Código 3.40. sincronizar_usuarios.sh

```
  #!/bin/bash

  string=`cat /home/www-data/usuarios.txt`

  if [[ $string == *"\\"* || $string == *"}"* || $string == *")"* || $string
== *"\""* || $string == *"#"* ]]; then
  echo "denegado"
  exit 1
  fi

  sed -i "s/in {.*}/in {${string}}/g" /etc/apache2/sites-enabled/000-default-
le-ssl.conf
  sed -i "s/in {.*}/in {${string}}/g" /etc/apache2/sites-enabled/panel.conf

  if [[ $string = "" ]]
  then
    sed -i "s/in {.*}/in {''}/g" /etc/apache2/sites-enabled/000-default-le-
ssl.conf
```

```
    sed -i "s/in {.*}/in {''}/g" /etc/apache2/sites-enabled/panel.conf
  fi

  service apache2 reload
```

Ante esto había varias opciones, ejecutar Apache por el usuario root, darle permisos a www-data para que pueda modificar los archivos de configuración… ambas opciones desaconsejadas por motivos de seguridad. Finalmente, nos decantamos por añadir este script al archivo /etc/sudoers para que www-data pueda ejecutarlo como superusuario, ya que es la manera que mas restringe los permisos otorgados. Debemos asegurarnos de que www-data no tiene permisos para modificar este script ya que entonces podría escalar privilegios invocando una *Shell*.

Para que esto sea seguro, el script se encarga de sanear el string (permite solo caracteres alfanuméricos y -) que se le pasa, debido a que si se permitieran caracteres como }) # “ ‘, se podría diseñar un string que fuese capaz de alterar la configuración de Apache.

Un ejemplo sería: *\'})\" \nRequire all granted \n \#*

Este string cerraría el bloque donde hay que escribir los certificados, con Require all granted permite el acceso a todos los usuarios y luego con # comenta el resto de la línea para que no haya errores de configuración. Estaríamos así inyectando configuración de Apache.

Código 3.41. Permisos y sudoers para sincronizar_usuarios.sh

```
  chmod 700 /etc/apache2/sincronizar_usuarios.sh

  #Permisos necesarios para modificar certificados aceptados
  chown root:www-data /etc/apache2/sites-enabled/000-default-le-ssl.conf
  chmod 660 /etc/apache2/sites-enabled/000-default-le-ssl.conf
  mkdir /home/www-data
  chown www-data:www-data /home/www-data

  #Modificamos el archivo sudoers para poder ejecutar sincronizar_usuarios
como root ya que necesita hacer reload de apache2
  echo "www-data                ALL=(ALL) NOPASSWD:
/etc/apache2/sincronizar_usuarios.sh" >> /etc/sudoers

  #Copiamos el panel de administración a su destino
  cp -r html/panel/ /var/www/panel
```

Si vemos en detalle los archivos .cgi, recibir.cgi recibe mediante un POST desde el panel de administracion una lista de nombres (CN) a los que permitir el acceso a la plataforma. Para cumplir este propósito, primero sanea el string eliminando caracteres no deseados al igual que sincronizar_usuarios.sh. Luego guarda este string en /home/www-data/usuarios.txt.

Código 3.42. recibir.cgi

```
  #!/bin/bash

  echo "Content-type: text/html"
  echo ""

  echo '<html>'
  echo '<head>'
  echo '<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">'
  echo '<title>Foo</title>'
  echo '</head>'
```

```
  echo '<body>'

  if [ "$REQUEST_METHOD" = "POST" ]; then
      if [ "$CONTENT_LENGTH" -gt 0 ]; then

          string=$(cat | sed "s/\"//g" | sed "s/\`//g" | sed "s/\\\//g" | sed
"s/{//g" | sed "s/}//g" | sed "s/(//g" | sed "s/)//g" | sed "s/#//g")

          echo $string > /home/www-data/usuarios.txt
          cat /home/www-data/usuarios.txt

          sudo /etc/apache2/sincronizar_usuarios.sh
          echo $?
      fi
  fi


  echo '</body>'
  echo '</html>'

  exit 0
```

De este archivo de texto, leer.cgi lee los usuarios actuales para mostrarlos en el panel de administración. Así podremos ver en todo momento los certificados permitidos.

Código 3.43. leer.cgi

```
  #!/bin/bash

  echo "Content-type: text/plain"
  echo ""

  string=$(cat /home/www-data/usuarios.txt)

  #elimino primer y ultimo caracter "'"
  #cambio ',' por saltos de linea
  echo ${string:1:-1} | sed 's|'\'','\'''|\n|g'

  exit 0
```

Por otro lado, nmap.cgi se encarga de escanear la red local en busca de equipos con un servidor RDP en marcha (puerto 3389). Filtra los datos usando un *parser* XML y muestra la IP, que la obtiene de la salida de nmap, y MAC de dichos equipos, obtenida de la tabla ARP.

Por cada dispositivo que detecta, aparecen unos campos para rellenar y crear un usuario que tenga conexión a dicho dispositivo.

Código 3.44. nmap.cgi

```
  #!/bin/bash

  LOCAL=$(ip route get 8.8.8.8 | head -1 | cut -d' ' -f7)
  MASK=$(ip a | grep ${LOCAL} | grep -oE "/[0-9]{1,2}")
  DEVICE=$(ip route get 8.8.8.8 | head -1 | cut -d' ' -f5)
```

```
  DATA=`nmap -p 3389 -Pn -n ${LOCAL}${MASK} -oX - | xmllint --xpath
'/nmaprun/host/ports/port[@portid="3389"]/state[@state="open"] |
/nmaprun/host/address[@addrtype="ipv4"]' -`

  echo "content-type: text/plain"
  echo
  echo '<ul>'

  readarray -t DATA_PARSED <<< "$DATA"

  for (( i=0; i<"${#DATA_PARSED[@]}"; i++ ))
  do
      j=i+1
      if [[ i -lt "${#DATA_PARSED[@]}" && "${DATA_PARSED[$i]}" == *"addr="* &&
"${DATA_PARSED[$j]}" == *"state="* ]]
      then
          printf "<li>"
          IP=$(echo "${DATA_PARSED[$i]}}" | sed 's/[^"]*"\([^"]*\).*/\1/')
          printf "IP=<b>${IP}</b> "
          MAC=$(ip n get $IP dev $DEVICE | head -1 | cut -d' ' -f5)
          printf "MAC=<b>${MAC}</b> "
          printf "<input type='text' name='connection_name'
placeholder='nombre de la conexión'>"
          printf "<input type='text' name='user_create'
placeholder='usuario'>"
          printf "<input type='password' name='password_create'
placeholder='contraseña'>"
          printf "<input type='submit' value='Crear' class='enviar_conexion'>"
          printf "</li>"
      fi
  done

  echo "</ul>"

  exit 0
```

El script crear.cgi se encarga de recibir una petición POST con un JSON y ejecutar los scripts crear_usuario.py y crear_conexion.py, para a partir de un usuario, contraseña, ip, mac y nombre de conexión, crear un usuario y asociarlo a la conexión RDP proporcionada.

Código 3.45. crear.cgi

```
  #!/bin/bash

  json=$(cat)

  echo "Content-type: text/html"

  #Le paso el json como primer parámetro a los scripts de creacion de usuario
y conexion asociada a este
  /usr/bin/python3 /etc/apache2/crear_usuario.py $json

  if [[ $? -eq 0 ]]; then
    echo ""
    /usr/bin/python3 /etc/apache2/crear_conexion.py $json

  else
    echo "Status: 400 Bad Request"
    echo ""

    echo "El usuario ya existe y has introducido mal la contraseña"
```

```
    exit 1

  fi

  exit 0
```

crear_usuario.py recibe un JSON con un usuario y contraseña. Si el usuario recibido no existe, lo crea. Para ello inserta en la base de datos que usa Apache Guacamole tanto en la tabla guacamole_entity como en guacamole_user. En estas guarda el nombre de usuario, una sal y el *hash* de la contraseña+sal.

La sal es una combinación de 32 bytes aleatorios que se añade a la contraseña previo a la operación de hash para proteger las contraseñas de ataques de fuerza bruta, ralentizándolos.

En el caso de que el usuario creado exista, comprueba que la contraseña introducida es la correcta, de otra manera, devuelve un error de contraseña incorrecta. Esto se usa para si el usuario introducido existe y las credenciales son correctas, no volverlo a crear y tan solo añadirle la conexión correspondiente con el otro script de Python.

Código 3.46. crear_usuario.py

```
  #!/usr/bin/python3

  import sys
  import json
  import hashlib
  import secrets
  import psycopg2

  json = json.loads(sys.argv[1])

  #Compruebo que todos los parámetros necesarios han sido introducidos
  if("password_create" in json and "user_create" in json):

    connection = psycopg2.connect(host="localhost", database="guacamole_db",
user="guacamole")
    cursor = connection.cursor()

    cursor.execute("select name, password_hash, password_salt from
guacamole_user inner join guacamole_entity on
guacamole_user.entity_id=guacamole_entity.entity_id where name=%s;",
(json["user_create"],))
    existente = cursor.fetchall()

    #Si el usuario no existe, lo creo
    if(len(existente) == 0):
      #generamos un salt como Apache Guacamole
      password_salt = hex(secrets.randbits(32*8)).lstrip('0x').upper()

      #concatenamos contraseña y salt y hacemos hash sha256
      password_hash = hashlib.sha256((json["password_create"] +
password_salt).encode())

      #Insertamos los datos en PostgreSQL
      cursor.execute("insert into guacamole_entity (name, type) values (%s,
'USER');", (json["user_create"],))
      cursor.execute("insert into guacamole_user (entity_id, password_hash,
password_salt, password_date) \
      select entity_id, decode(%s, 'hex'), decode(%s, 'hex'),
CURRENT_TIMESTAMP as group \
```

```
      from guacamole_entity where name=%s and type='USER';",
(password_hash.hexdigest(), password_salt, json["user_create"]))

      connection.commit()
      cursor.close()
      connection.close()

    else:

      #Si el usuario no existe, compruebo que la contraseña introducida es la
correcta
      password_salt = bytes.hex(bytes(existente[0][2])).lstrip('0x').upper()
      password_hash_correcto = bytes.hex(bytes(existente[0][1]))

      #concatenamos contraseña y salt y hacemos hash sha256
      password_hash = hashlib.sha256((json["password_create"] +
password_salt).encode()).hexdigest()

      if(password_hash == password_hash_correcto):
        #Si la contraseña es correcta, no tenemos que hacer nada, el otro
script registrará la conexión
        cursor.close()
        connection.close()

      else:
        #Si la contraseña es incorrecta, mostramos error
        cursor.close()
        connection.close()
        sys.exit("Contraseña incorrecta")

  else:
    sys.exit("Faltan campos por rellenar para la creación del usuario")
```

crear_conexion.py recibe un json con el nombre de la conexión, la ip, la mac y el usuario al que se le debe asociar dicha conexión. Para ello inserta las filas correspondientes en las tablas guacamole_connection, guacamole_connection_parameter y guacamole_connection_permission tal y como se indica en la documentación [39].

Código 3.47. crear_conexion.py

```
  #!/usr/bin/python3

  ##
  ## Script para crear una conexion de Apache Guacamole y asignarle un usuario
  ##

  import sys
  import json
  import psycopg2

  json = json.loads(sys.argv[1])

  #Compruebo que todos los parámetros necesarios han sido introducidos
  if("connection_name" in json and "connection_ip" in json and "mac" in json
and "user_create" in json):
    connection = psycopg2.connect(host="localhost", database="guacamole_db",
user="guacamole")
    cursor = connection.cursor()
```

```
    cursor.execute("select * from guacamole_connection_parameter where
parameter_name='hostname' and parameter_value=%s;", (json["connection_ip"],))
    conexiones = cursor.fetchall()

    #Si la conexión a esa ip no existe, la creo
    if(len(conexiones) == 0):
      cursor.execute("insert into guacamole_connection (connection_name,
protocol) values (%s, 'rdp');", (json["connection_name"],))

      cursor.execute("insert into guacamole_connection_parameter
(connection_id, parameter_name, parameter_value) select connection_id,
'hostname', %s from guacamole_connection where connection_name=%s;",
(json["connection_ip"], json["connection_name"]))
      cursor.execute("insert into guacamole_connection_parameter
(connection_id, parameter_name, parameter_value) select connection_id,
'password', '${GUAC_PASSWORD}' from guacamole_connection where
connection_name=%s;", (json["connection_name"],))

      cursor.execute("insert into guacamole_connection_parameter
(connection_id, parameter_name, parameter_value) select connection_id, 'wol-
mac-addr', %s from guacamole_connection where connection_name=%s;",
(json["mac"], json["connection_name"]))

      cursor.execute("insert into guacamole_connection_parameter
(connection_id, parameter_name, parameter_value) select connection_id,
'ignore-cert', 'true' from guacamole_connection where connection_name=%s;",
(json["connection_name"],))

      cursor.execute("insert into guacamole_connection_parameter
(connection_id, parameter_name, parameter_value) select connection_id, 'wol-
send-packet', 'true' from guacamole_connection where connection_name=%s;",
(json["connection_name"],))

      cursor.execute("insert into guacamole_connection_parameter
(connection_id, parameter_name, parameter_value) select connection_id,
'username', '${GUAC_USERNAME}' from guacamole_connection where
connection_name=%s;", (json["connection_name"],))

    #Asocio la conexión con el usuario
    cursor.execute("insert into guacamole_connection_permission (entity_id,
connection_id, permission) values ((select entity_id from guacamole_entity
where name=%s), (select distinct connection_id from
guacamole_connection_parameter where parameter_name='hostname' and
parameter_value=%s), 'READ');", (json["user_create"], json["connection_ip"]))

    connection.commit()
    cursor.close()
    connection.close()

  else:
    sys.exit("Faltan campos por rellenar para la creación de la conexión")
```

Finalmente, tenemos crear_todas.cgi que realiza el mismo proceso que crear.cgi solo que al usuario se le asignan todas las conexiones existentes.

Código 3.48. crear_todas.cgi

```
  #!/bin/bash

  json=$(cat)

  echo "Content-type: text/html"

  #Le paso el json como primer parámetro a los scripts de creacion de usuario
y conexion asociada a este
  /usr/bin/python3 /etc/apache2/crear_usuario.py $json

  if [[ $? -eq 0 ]]; then
    echo ""
    /usr/bin/python3 /etc/apache2/crear_conexion_todas.py $json

  else
    echo "Status: 400 Bad Request"
    echo ""

    echo "El usuario ya existe y has introducido mal la contraseña"
    exit 1

  fi

  exit 0
```

Para ello, en lugar de crear_conexion.py usa crear_conexion_todas.py el cual hace una petición a la base de datos para obtener todas las conexiones existentes y asociarlas.

Código 3.49. crear_conexion_todas.py

```
  #!/usr/bin/python3

  ##
  ## Script para crear una conexion de Apache Guacamole y asignarle un usuario
  ##

  import sys
  import json
  import psycopg2

  json = json.loads(sys.argv[1])

  #Compruebo que todos los parámetros necesarios han sido introducidos
  if("user_create" in json):
    connection = psycopg2.connect(host="localhost", database="guacamole_db",
user="guacamole")
    cursor = connection.cursor()

    cursor.execute("select distinct connection_id from guacamole_connection;")
    conexiones = cursor.fetchall()

    #Si hay conexiones creadas, le asigno todas
    if(len(conexiones) != 0):
      for conexion in conexiones:

        cursor.execute("select 1 from guacamole_connection_permission where
entity_id=(select entity_id from guacamole_entity where name=%s) and
```

```
guacamole_connection_permission.connection_id=%s", (json["user_create"],
conexion[0]))
        existente = cursor.fetchall()

        if(len(existente) == 0):
        #Asocio las conexiones con el usuario si no está
          cursor.execute("insert into guacamole_connection_permission
(entity_id, connection_id, permission) values ((select entity_id from
guacamole_entity where name=%s), %s, 'READ');", (json["user_create"],
conexion[0]))

      connection.commit()

    cursor.close()
    connection.close()

  else:
    sys.exit("Faltan campos por rellenar para la creación de la conexión")
```

3.3.6 Contenido estático

En este apartado detallaremos el contenido estático que hemos copiado a la carpeta /var/www para mostar el panel de administración y la web de instrucciones.

3.3.6.1 Panel de administración

Para el panel contamos con un archivo index.html localizado en /var/www/panel/index.html, el cual consta de tres secciones indicadas por la etiqueta <fieldset></fieldset>. La primera contiene un campo de texto para rellenar con los nombres de los usuarios cuyo certificado digital será aceptado y un botón para enviarlos. La segunda sección se mostrarán los equipos con un servidor RDP activo una vez se use el botón de cargar. Junto con cada dispositivo aparecerán 3 campos de texto para rellenarlos con el nombre de la conexión, nombre de usuario y contrasñea. Esto creará automáticamente un usuario para Apache Guacamole, si no está ya creado, y le asociará la conexión con el dispositivo elegido. La tercera es una sección para crear usuarios que tengan asociadas todas las conexiones existentes.

Código 3.50. index.html – Página de inicio del panel de administración

```
<!doctype html>
<html>
  <head>
    <meta http-equiv="Content-Type" charset="UTF-8" />
    <title>Panel de administración</title>
    <script type="text/javascript" src="panel.js"></script>
    <link rel="stylesheet" href="panel.css">
  </head>

  <body>
    <h1>Panel de administración</h1>

    <fieldset>
      <legend>Registrar usuarios</legend>
      <p>
        Usuarios:
```

```
        </p>
        <p>
          <textarea type="text" name="usuarios" rows="10"
cols="40"></textarea><br/>
        </p>
        <p>
          <input id="enviar_usuarios" type="submit" value="Enviar"/>
        </p>
      </fieldset>

    <fieldset>
        <legend>Descubrir dispositivos</legend>
        <p>
          Aquí se muestran los dispositivos con un servidor RDP activo. Para
crear un usuario/conexión para acceder a cada uno, rellene los siguientes
campos:
        </p>
        <br/>
        <p id="nmap">
        </p>
        <br/>
        <p>
          <input id="enviar_dispositivos" type="submit" value="Cargar"/>
        </p>
    </fieldset>

    <fieldset>
        <legend>Crear usuario con todas las conexiones</legend>
        <p>
        <input type="text" placeholder="usuario" id="usuario_todas">
        <input type="password" placeholder="contraseña" id="password_todas">
        <input type="submit" value="Crear" id="enviar_conexion_todas">
        </p>
    </fieldset>

    </body>
  </html>
```

También mediante una etiqueta <link> cargamos los estilos del archivo panel.css y con <script> se carga panel.js.

Código 3.51. panel.css – Estilos para el panel de administración

```
  body {
      margin-left: 10%;
      margin-right: 10%;
      background: beige;
  }


  h1 {
      margin-top: 1em;
      margin-bottom: 5%;
  }

  fieldset {
      margin-bottom: 5%;
      background: white;
  }
```

En este archivo .js se establece que una vez cargue el panel, se haga una petición mediante la función carga_usuarios() y el objeto XMLHttpRequest al endpoint /cgi-bin/leer.cgi para obtener los nombres que en este momento tienen permitido el acceso. Además, cuando se pulse el botón enviar, de nuevo usando Javascript AJAX, se lee el campo de texto existente, se valida con validateUsers(str) y si contiene solo caracteres permitidos (alfanuméricos, espacios y saltos de línea), se envían estos datos a la url /cgi-bin/recibir.cgi que se encargará de hacer los cambios pertinentes para actualizar los nombres. Esta validación de los caracteres permitidos se realiza tanto en .js como en nmap.cgi

Código 3.52. panel.js – Peticiones al CGI de control de usuarios usando AJAX

```
  "use strict";
  window.addEventListener('load', anadirEventos, false);

  function anadirEventos() {
      carga_usuarios();
      document.getElementById("enviar_dispositivos").addEventListener('click',
carga_dispositivos, false);
      document.getElementById("enviar_usuarios").addEventListener('click',
enviar_usuarios, false);
  }

  function carga_usuarios() {
      var request = new XMLHttpRequest();
      const url = "/cgi-bin/leer.cgi"
      request.open("get", url);
      request.send();

      request.onload = function () {
        document.getElementsByName("usuarios")[0].value = this.responseText;
      }
  }


  function enviar_usuarios (formulario) {
      if(validateUsers(document.getElementsByName("usuarios")[0].value) ==
true) {

        var request = new XMLHttpRequest();
        const url = "/cgi-bin/recibir.cgi"
        request.open("post", url);

        var string_split =
document.getElementsByName("usuarios")[0].value.split('\n');
        var string = "'" + string_split.join("','") + "'";

        request.send(string);
        request.onload = function () {
          if (this.status == 200) {
            alert("Guardado correctamente");
          } else {
            alert("Error");
          }
        }
      } else {
        alert("Has introducido algún carácter no permitido\nPermitidos
alfanuméricos y -");
      }
  }
```

```
  function validateUsers(str) {
    let code, i, len;

    for (i = 0, len = str.length; i < len; i++) {
      code = str.charCodeAt(i);
      if (!(code > 47 && code < 58) && // (0-9)
          !(code > 64 && code < 91) && // (A-Z)
          !(code > 96 && code < 123) && // (a-z)
          !(code == 32) && // " " espacio
          !(code == 10) && // "\n" salto de linea
          !(code == 45) && // "-" guion
          !(code == 209) && // Ñ
          !(code == 241)) { // ñ
        return false;
      }
    }
    return true;
  };
```

Por otro lado, en el segundo <fieldset>, una vez se pulsa el botón enviar, se ejecuta la función JavaScript llamada carga_dispositivos() que hace una petición a /cgi-bin/nmap-cgi para obtener los dispositivos de la red. Esta función pondrá un GIF de cargando en la pantalla ya que la petición puede llegar a tardar unos segundos en responderse.

Si en esta sección se rellenan los campos de texto y utiliza el botón crear, se envia una petición al endpoint /cgi-bin/crear.cgi, que se encarga de crear un usuario con la conexión al dispositivo elegido.

Código 3.53. panel.js – Peticiones al CGI de control de dispositivos usando AJAX

```
  function carga_dispositivos() {
      let request = new XMLHttpRequest();
      const url = "/cgi-bin/nmap.cgi"

      let img = document.createElement("img");
      img.src="cargando.gif";
      img.alt="Cargando...";

      document.getElementById("nmap").appendChild(img);

      request.open("get", url);
      request.send();

      request.onload = function () {
        document.getElementById("nmap").innerHTML = this.responseText;
        let botones = document.getElementsByClassName("enviar_conexion");

        for(let i=0; i<botones.length; i++) {
          botones[i].addEventListener('click', enviar_conexion, false);
        }

      }
  }

  function enviar_conexion(boton) {
    let connection_ip =
boton.srcElement.parentElement.childNodes[1].textContent;
    let mac = boton.srcElement.parentElement.childNodes[3].textContent;
```

```
    let connection_name = boton.srcElement.parentElement.childNodes[5].value;
    let user_create = boton.srcElement.parentElement.childNodes[6].value;
    let password_create = boton.srcElement.parentElement.childNodes[7].value;

    let json = {"user_create": user_create, "password_create":
password_create, "connection_name": connection_name, "connection_ip":
connection_ip, "mac": mac};
    console.log(connection_name.length);
    if(connection_name.length > 4 && user_create.length > 4 &&
password_create.length > 4) {
      let request = new XMLHttpRequest();
      const url = "/cgi-bin/crear.cgi"
      request.open("post", url);

      request.setRequestHeader("Content-Type", "application/json;charset=UTF-
8");
      request.send(JSON.stringify(json));
      request.onload = function () {
        if (this.status == 200) {
          alert("Creado correctamente");
        } else {
          alert(this.responseText);
        }
      }
    } else {
      alert("-Debe rellenar todos los campos\n-Todos los campos deben tener al
menos 4 caracteres");
    }
  }
```

Por último, en la tercera sección habrá 2 campos, usuario y contraseña, esta se encarga de crear un usuario que tenga todas las conexiones existentes asociadas. La petición se hará de nuevo usando AJAX como muestra el siguiente código.

Código 3.54. panel.js – Petición a crear usuario con todas las conexiones asociadas

```
  function enviar_conexion_todas(boton) {
    let user_create = document.getElementById("usuario_todas").value;
    let password_create = document.getElementById("password_todas").value;

    let json = {"user_create": user_create, "password_create":
password_create}
    if(user_create.length > 4 && password_create.length > 4) {
      let request = new XMLHttpRequest();
      const url = "/cgi-bin/crear_todas.cgi"
      request.open("post", url);

      request.setRequestHeader("Content-Type", "application/json;charset=UTF-
8");
      request.send(JSON.stringify(json));
      request.onload = function () {
        if (this.status == 200) {
          alert("Creado correctamente");
        } else {
          alert(this.responseText);
        }
      }
    } else {
```

```
      alert("-Debe rellenar todos los campos\n-Todos los campos deben tener al
menos 4 caracteres");
    }
  }
```

3.3.6.2 Instrucciones

En /var/www/html/instrucciones estarán localizados los archivos index.html, centos.sh, ubuntu.sh, windowspro.bat y RDPW_installer.exe.

El primero define una página formada por etiquetas html que varían según el código JavaScript de la etiqueta <script>. Dicho código se encarga de comprobar la cabecera userAgent y variar el contenido de la etiqueta <p> con id="so".

Si el userAgent contiene CentOS, mostrará un link al archivo centos.sh, si contiene Ubuntu, el link será a ubuntu.sh y si contiene Windows, mostrará dos links, un archivo .bat para los Windows versión PRO y un .exe para los HOME. Esto se debe a que la versión PRO de Windows tiene por defecto el servidor RDP habilitado y solo hay que activarlo y la versión HOME necesita un binario que lo habilite.

Código 3.55. index.html – Página de inicio para las instrucciones de los usuarios

```
  <!DOCTYPE html>
  <html>
    <head>
      <meta charset="UTF-8">
  <link rel="stylesheet" href="estilos.css">
    </head>
    <body>
      <p id="inicio">
        <a href="/">Ir a la web de inicio de sesión</a>
      </p>
      <br>
      <fieldset>
      <legend>Cómo activar el servicio</legend>
      <p id="so"></p>
      </fieldset>
    </body>
      <script>
        var parrafo = document.getElementById("so");
        if(navigator.userAgent.includes("Ubuntu")) {
              parrafo.innerHTML = "<b>Ubuntu:</b><br>descargar y ejecutar el
.sh del siguiente link <a href=\"ubuntu.sh\">Descargar</a>";
        } else if(navigator.userAgent.includes("CentOS")) {
              parrafo.innerHTML = "<b>CentOS:</b><br>sudo yum install xrdp, o
descargar y ejecutar el .sh del siguiente link <a
href=\"centos.sh\">Descargar</a>";
        } else if(navigator.userAgent.includes("Windows")) {
              parrafo.innerHTML = "<b>Windows:</b> <br>-Versión PRO: ejecutar
el siguiente archivo como administrador <a
href=\"windowspro.bat\">Descargar</a><br>-Versión HOME: ejecutar el siguiente
archivo como administrador <a href=\"RDPW_installer.exe\">Descargar</a>";
        }
      </script>
  </html>
```

El archivo centos.sh contiene 3 líneas. En la primera instalamos el servidor RDP necesario usando el gestor de

paquetes yum, luego añadimos una excepción al firewall (puerto 3389) y lo recargamos

Código 3.56. centos.sh – Instalar servidor RDP CentOS

```
  sudo yum install xrdp -y
  sudo firewall-cmd --permanent --add-port=3389/tcp
  sudo firewall-cmd --reload
```

En ubuntu.sh, mediante apt instalamos el servidor RDP y luego permitimos el puerto de RDP en el firewall.

Código 3.57. ubuntu.sh – Instalar servidor RDP Ubuntu

```
  sudo apt install xrdp -y
  sudo ufw allow 3389
```

En windows.bat, primero establecemos que el servicio RemoteRegistry (escritorio remoto con RDP) se inicie automáticamente y además lo encendemos manualmente. Luego añadimos la clave fDenyTSConnections igualada a 0 para que permita conexiones entrantes a este servidor. Finalmente permitimos el escritorio remoto en el firewall de Windows.

Código 3.58. windowspro.bat – Activar servidor RDP Windows PRO

```
  sc config RemoteRegistry start= auto
  net start remoteregistry
  reg add "HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Terminal
Server" /v fDenyTSConnections /t REG_DWORD /d 0 /f
  netsh firewall set service type = remotedesktop mode = enable
```

El archivo RDPW_installer.exe es un archivo binario que habilita el servicio de escritorio remoto en las versiones de Windows que no lo traen por defecto, es decir, en las versiones HOME. Una vez lo ejecutemos tendremos el servidor RDP funcionando y programado para que se encienda en cada inicio. También posee una GUI para la configuración de algunos de los parámetros del escritorio remoto de Windows.

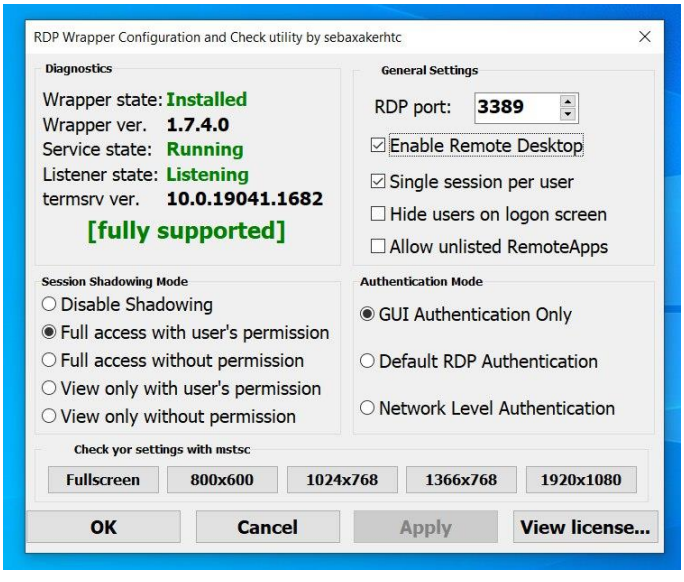

Figura 3-10. RDPW_Installer.exe

3.3.7 **Cambio de contraseña por defecto**

Apache Guacamole tiene un usuario administrador por defecto cuyas credenciales son *guacadmin*, tanto usuario como contraseña. Como esto supone un riesgo de seguridad hasta que lo cambiemos, el script de instalación *install.sh* se encargará de ello, ejecutando un script en Python que hace los cambios pertinentes en la base de datos del contenedor Docker.

Código 3.59. install.sh – Cambio de contraseña del usuario guacadmin

```
  #Cambio la contraseña al usuario guacadmin
  /usr/bin/python3 cambiar_guacadmin.py $PASSWORD_GUAC
```

Apache Guacamole guarda sus contraseñas en la base de datos PostgreSQL usando un *salt* de 32 bytes, hexadecimal y en mayúsculas, el cual añade a la contraseña, para después aplicar el algoritmo SHA-256 [39]. El script de Python *cambiar_guacadmin.py* replicará esto con la contraseña configurada en el archivo config.

Código 3.60. cambiar_guacadmin.py – Cambio de contraseña del usuario guacadmin

```
  #!/usr/bin/python3

  import sys
  import hashlib
  import secrets
  import psycopg2

  password = sys.argv[1]

  #Compruebo que la contraseña ha sido introducida
  if(len(password) > 0):

    connection = psycopg2.connect(host="localhost", database="guacamole_db",
user="guacamole")
    cursor = connection.cursor()

    #generamos un salt como Apache Guacamole
    password_salt = hex(secrets.randbits(32*8)).lstrip('0x').upper()

    #concatenamos contraseña y salt y hacemos hash sha256
    password_hash = hashlib.sha256((password + password_salt).encode())

    print(password_hash.hexdigest(), password_salt)

    #Insertamos los datos en PostgreSQL
    cursor.execute("update guacamole_user set password_hash=decode(%s, 'hex'),
password_salt=decode(%s, 'hex'), password_date=CURRENT_TIMESTAMP \
    where entity_id=1;", (password_hash.hexdigest(), password_salt))

    connection.commit()
    cursor.close()
    connection.close()
```

Una vez hecho, podremos acceder al usuario administrador guacadmin con la contraseña $PASSWORD_GUAC y si en algún momento deseamos cambiar de nuevo las credenciales, podremos hacerlo desde el menú de preferencias de Apache Guacamole.

3.4 Desinstalación

En el Anexo A se adjunta un script que desinstala y borra todos los componentes usados en este proyecto, dejando la máquina en su estado anterior. Las cuentas de AWS y DuckDNS habrá que borrarlas manualmente desde sus páginas web oficiales.

4 Uso

El valor de una idea radica en el uso de la misma.

- Thomas Edison -

Este sistema está diseñado para ser Plug-and-Play, es decir, enchufar y usar. Para ello cuenta con mecanismos de autoconfiguración como pueden ser el despliegue de infraestructura en la nube de AWS para evitar la necesidad del Port Forwarding y configuración del router, o la detección del cambio de IP local.

Una vez configuradas las variables del archivo config e instalado con el archivo install.sh, el dispositivo que usemos como servidor, en nuestro caso, la Raspberry, estará listo para ubicarse en la empresa y empezar a funcionar. Solo necesita un cable ethernet para la conexión a internet y corriente a través de su USB-C.

Cuando se encienda, iniciará todos los servicios necesarios como Docker, Apache, etc. creará la infraestructura con Terraform y detectando la IP local que posea, hará cambios en la configuración para funcionar correctamente. Una vez hecho esto, nuestro sistema de teletrabajo estará disponible en el horario que establecimos en la configuración.

A partir de ahora se describirá el funcionamiento que debe conocer un usuario administrador y un usuario que desee teletrabajar.

4.1 Administrador

El usuario administrador debe tener conocimientos para usar el panel de administración y los ajustes dentro de la plataforma de Apache guacamole para crear usuarios, conexiones, grupos, etc. A este se le proporcionará una nota de rentry.co donde aparece la IP local del servidor actualizada al minuto, y el usuario y contraseña del panel y del usuario administrador.

4.1.1 Panel de administración

En primer lugar, para acceder al panel introducirá en un navegador web o hará click en la IP de la nota proporcionada. Tendrá que introducir el usuario y contraseña de administrador.

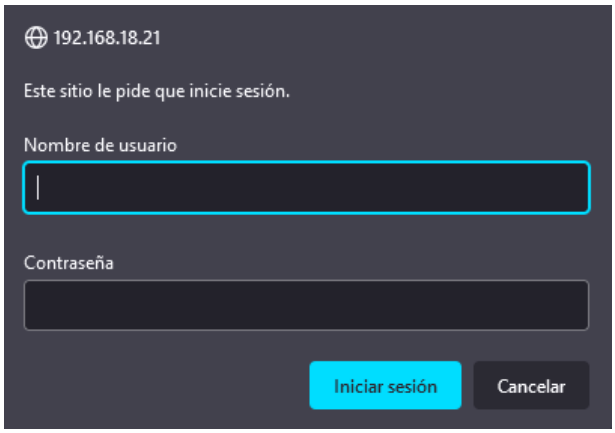

Figura 4-1. Inicio de sesión para el panel de administración

Cuando rellene este desplegable, verá tres secciones, la primera para registrar certificados digitales en la plataforma, permitiéndoles así el acceso, la segunda para realizar un escaneo de la red y crear conexiones a los dispositivos existentes y la última para crear usuarios con acceso a todas las conexiones.

4.1.1.1 Inscripción de usuarios

Para inscribir a los usuarios, introducirá una lista de Nombres Comunes (campo CN del certificado digital) que estará formada por *PRIMER_APELLIDO SEGUNDO_APELLIDO NOMBRE – DNI*. Cada entrada de la lista estará separada por un salto de línea. En la imagen podemos ver un ejemplo.

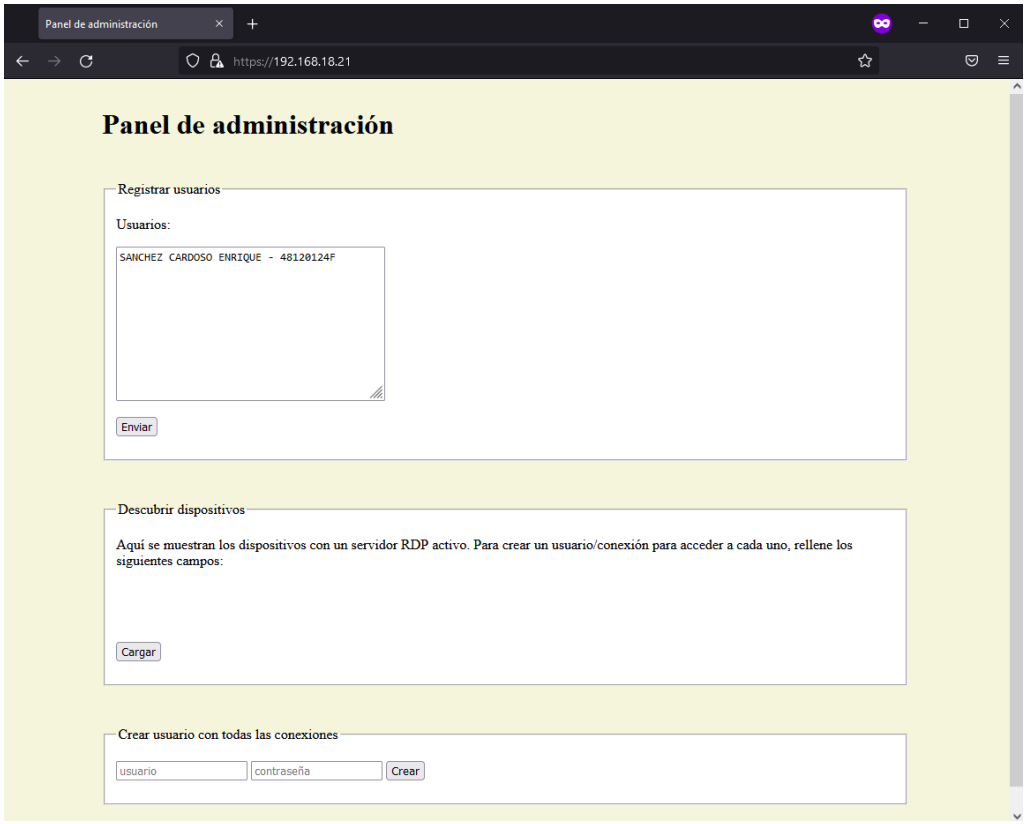

Figura 4-2. Panel de administración

4.1.1.2 Descubrimiento de dispositivos

Una vez los usuarios sigan los pasos de sus instrucciones, el administrador puede pulsar el botón de la segunda sección y se le mostrará dirección IP y MAC de los dispositivos con un servidor RDP activo. Junto a cada dispositivo habrá 3 campos para rellenar con el nombre de conexión, usuario y contraseña para crear un usuario que pueda conectarse a dicho dispositivo. Si el usuario ya existiese, tan solo se crea la conexión al equipo.

La contraseña y el usuario introducido serán usados para el inicio de sesión en el dispositivo mediante RDP (esto es debido a que empleamos las variables de Apache Guacamole $GUAC_USERNAME y $GUAC_PASSWORD [35]).

Figura 4-3. Panel de administración con nmap cargando

Figura 4-4. Panel de configuración con nmap cargado

4.1.1.3 Creación de usuarios con todas las conexiones

La sección anterior nos permite crear usuarios y darle acesso a las conexiones de una en una. Es frecuente que haya usuarios con acceso a todas las conexiones por lo que la sección 3 tendrá dos campos de texto, uno para usuario y otro para contraseña que cuando se rellenen y envíen crearán un usuario en Apache Guacamole con estas características.

Figura 4-5. Creación de usuarios con todas las conexiones

4.1.2 Plataforma Apache Guacamole

Las tareas a realizar por el administrador en la plataforma de Apache Guacamole serán las siguientes:

4.1.2.1 Cambio de contraseña

Si en algún momento el administrador desea cambiar la contraseña que se le estableció en el proceso de instalación, inicia sesión en la plataforma, se dirige al menú de opciones y procede a cambiar la contraseña.

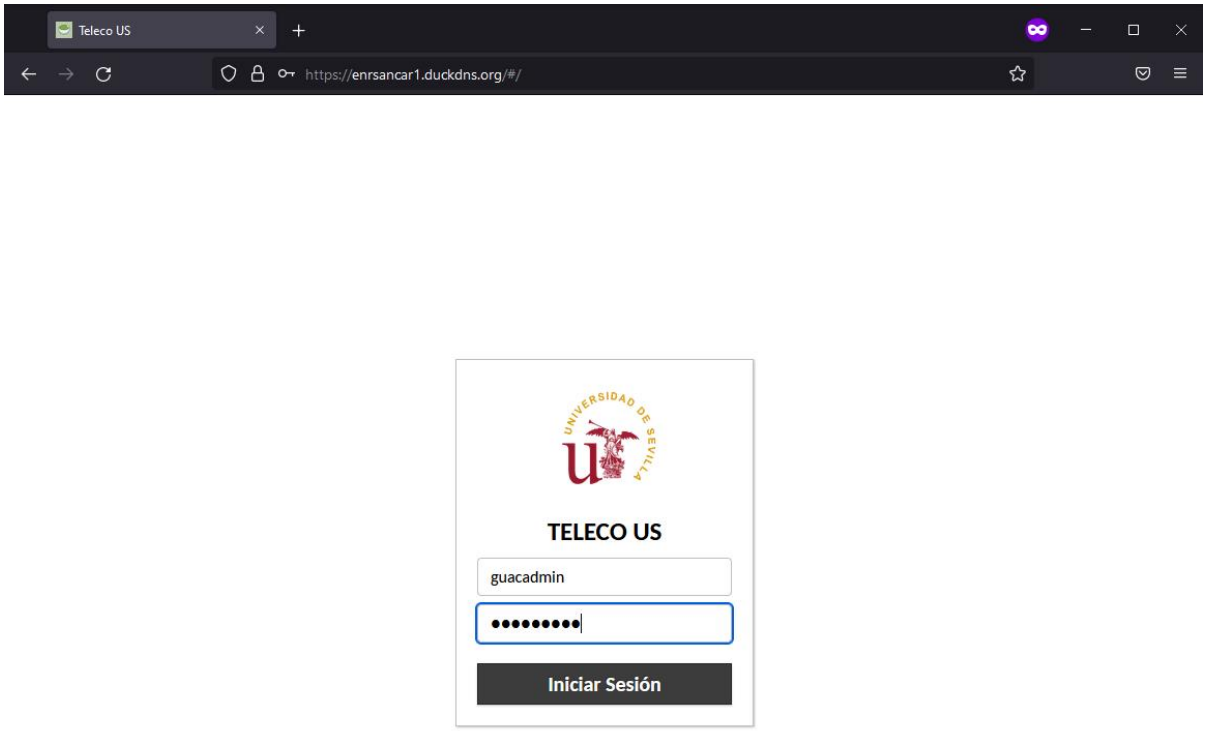

Figura 4-6. Inicio de sesión en el cliente web

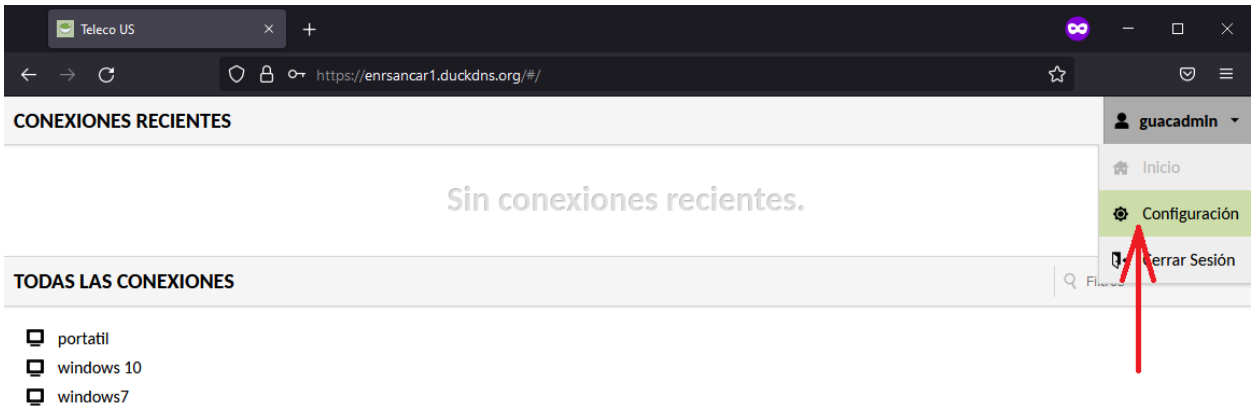

Figura 4-7. Menú desplegable para configuración

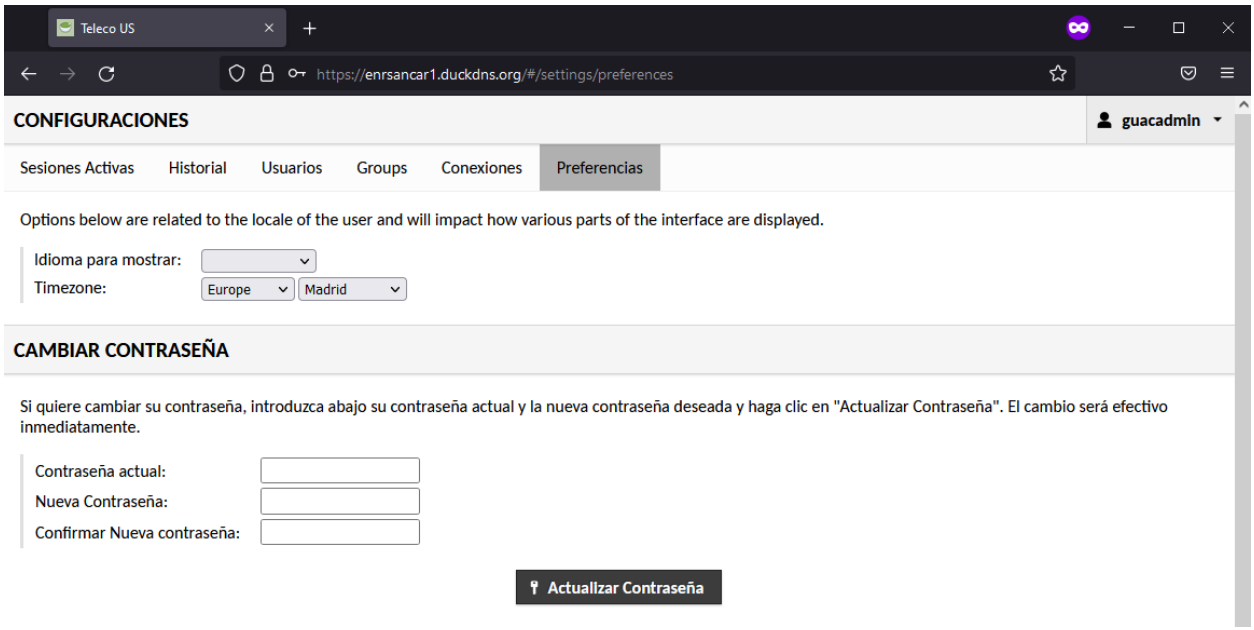

Figura 4-8. Cambiar contraseña

4.1.2.2 Creación de conexiones

Alternativamente a la forma propuesta por el panel de administración para la creación de conexiones, podemos usar las opciones de Apache Guacamole para crear conexiones. Asignándole un nombre, con su IP, el usuario y contraseña (solo si se quiere inicio de sesión automático, si no, saldrá un desplegable para introducirlos) y la MAC (solo si se quiere usar el Wake on Lan) se crearán las conexiones a los diferentes dispositivos.

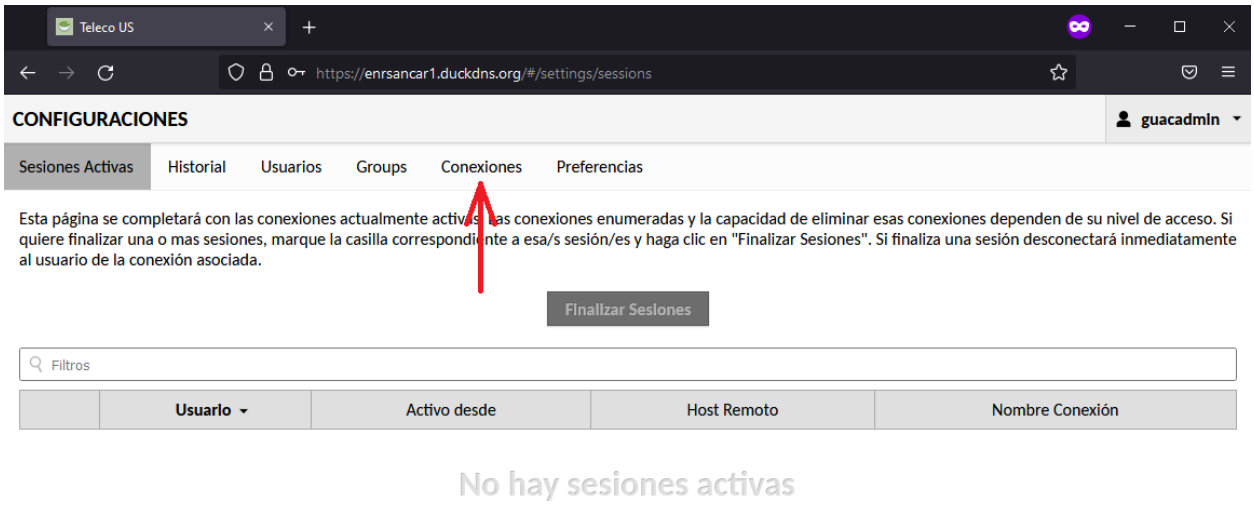

Figura 4-9. Menú conexiones

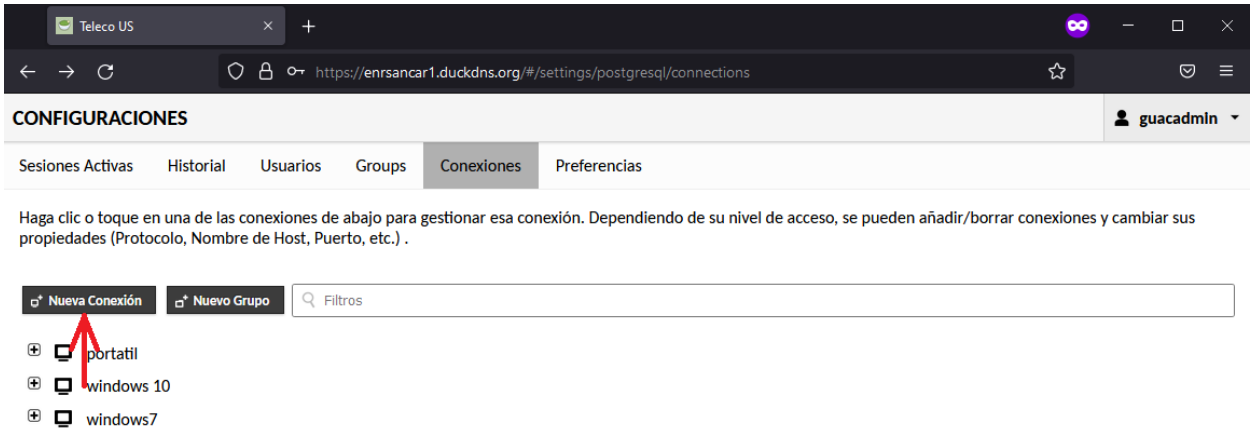

Figura 4-10. Nueva conexión

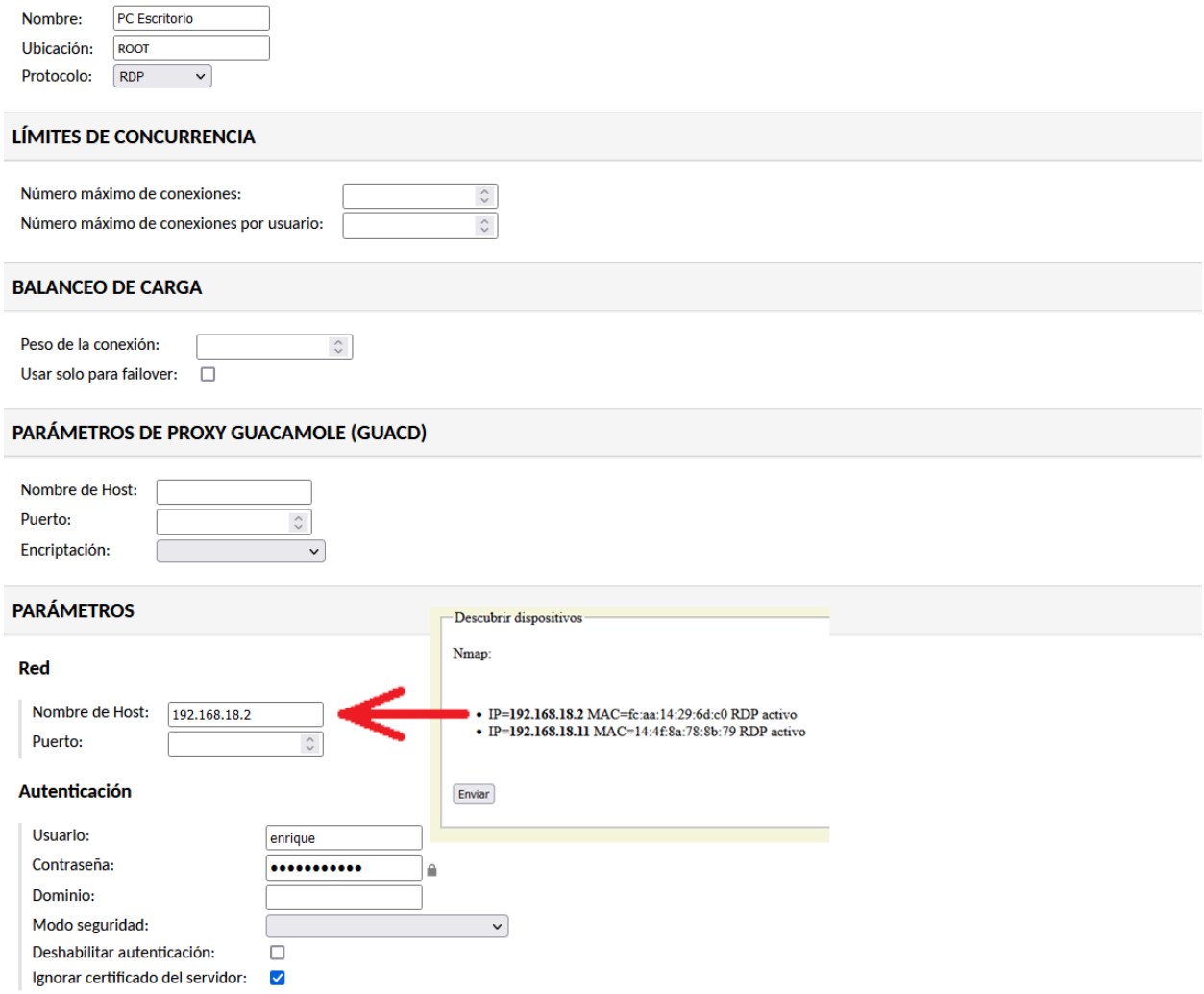

Figura 4-11. Formulario de nueva conexión

Para que se permita la conexión, debemos indicarle a Apache Guacamole que ignore el certificado del servidor (dispositivo a usar remotamente).

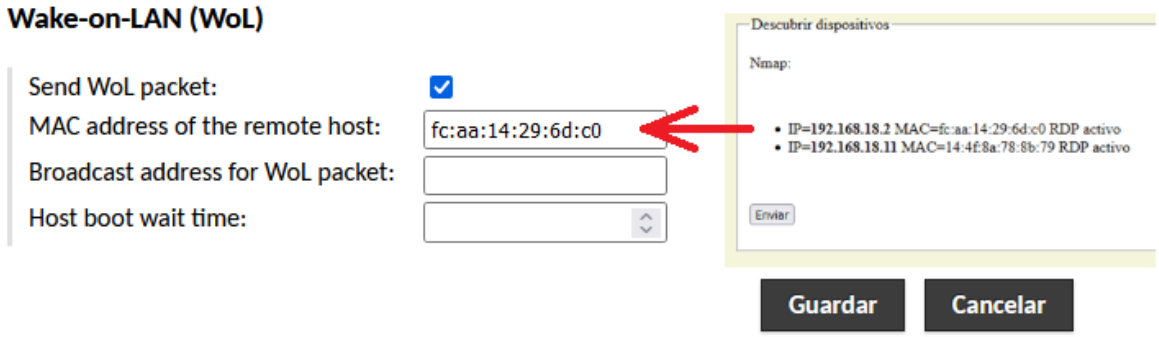

Figura 4-12. Rellenar MAC para WoL

4.1.2.3 Creación de usuarios

Alternativamente a la forma propuesta por el panel de administración, podremos crear usuarios desde las opciones de configuración de Apache Guacamole. Estos podrán tener diferentes permisos como el rol administrador o poder cambiar su contraseña.

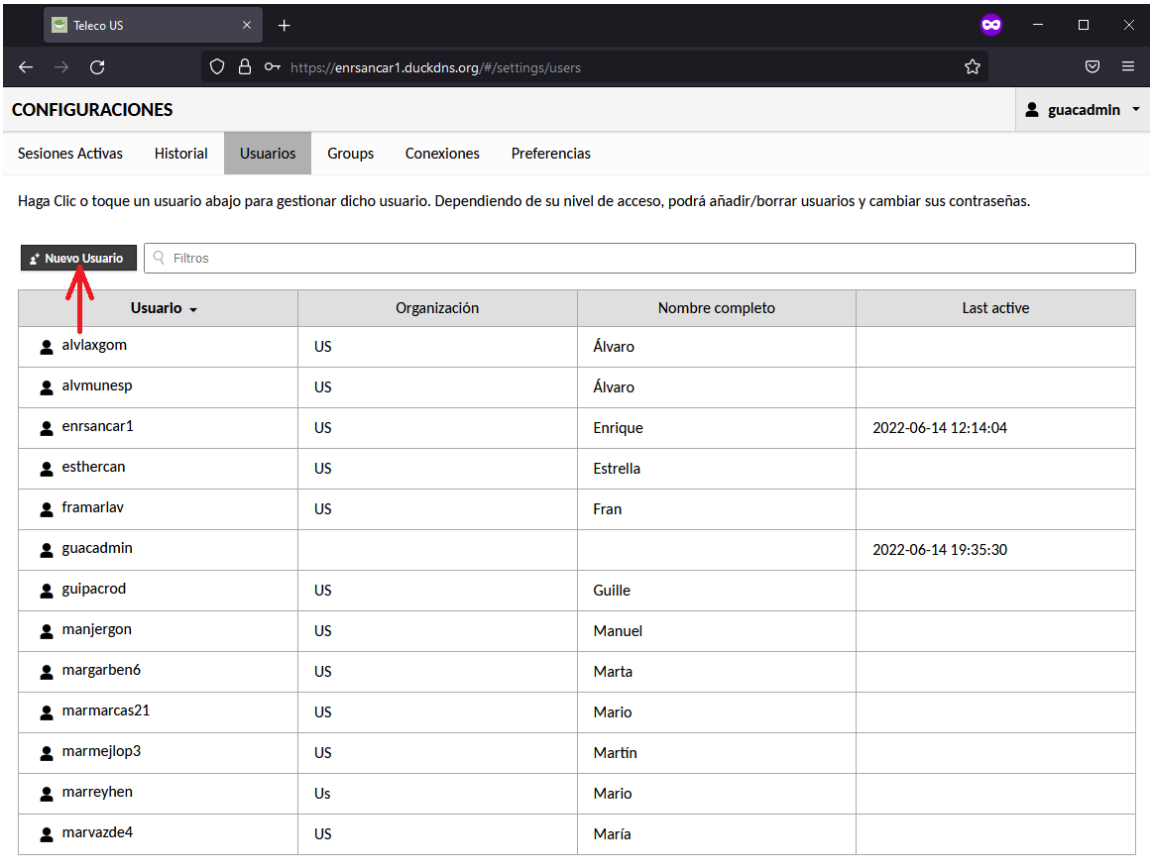

Figura 4-13. Crear nuevo usuario

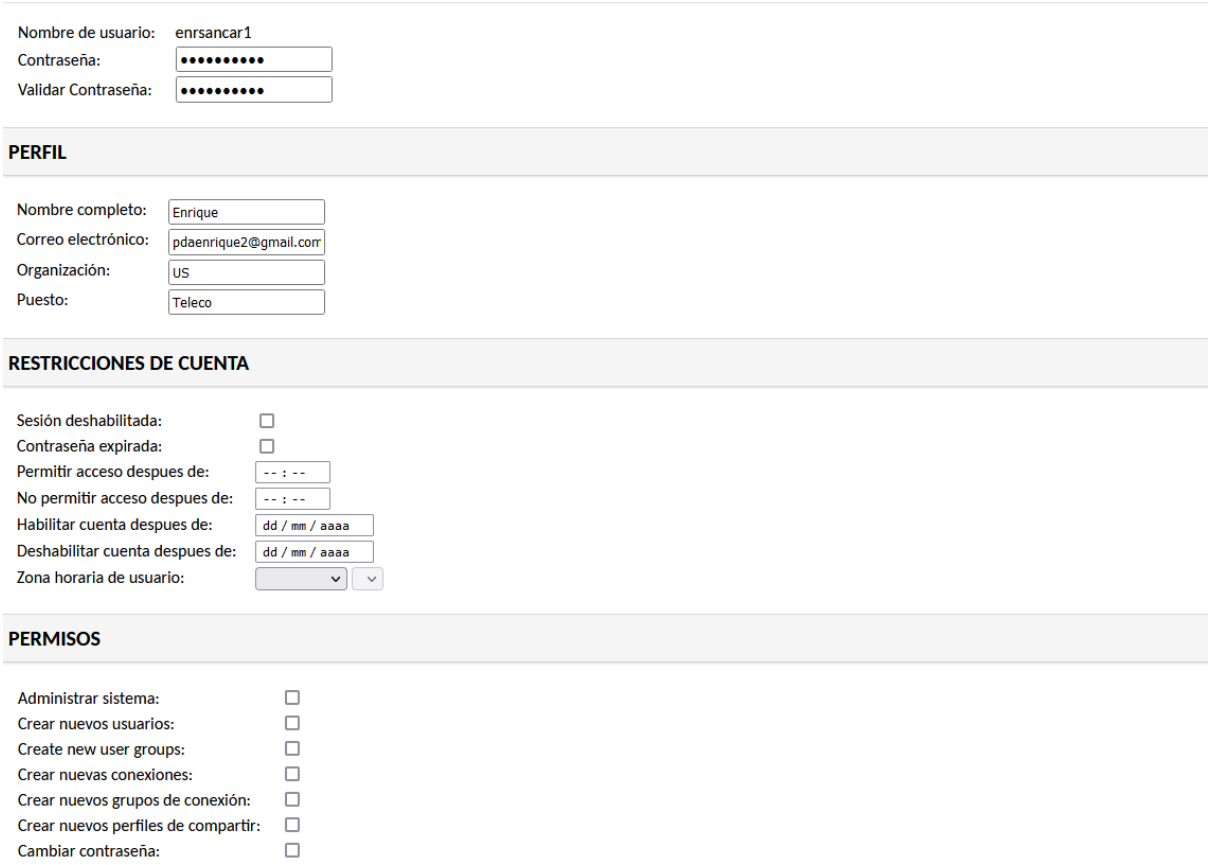

Figura 4-14. Formulario de nuevo usuario

Cada uno tendrá asignados una cantidad ilimitada de dispositivos.

Figura 4-15. Selección de conexiones permitidas a un usuario

Para obtener más información puede consultar la documentación de administración de Apache Guacamole [40].

4.2 Usuario

Por su parte, el primer paso del usuario será entrar a la web de instrucciones desde el dispositivo de la empresa que quiera usar remotamente.

4.2.1 Instrucciones

Aquí, dependiendo del sistema operativo que use, le aparecerá un ejecutable u otro para activar RDP.

En Windows se ven dos opciones, una para los Windows Pro y otra para los Windows Home. Ubuntu y CentOS tienen una sola opción disponible.

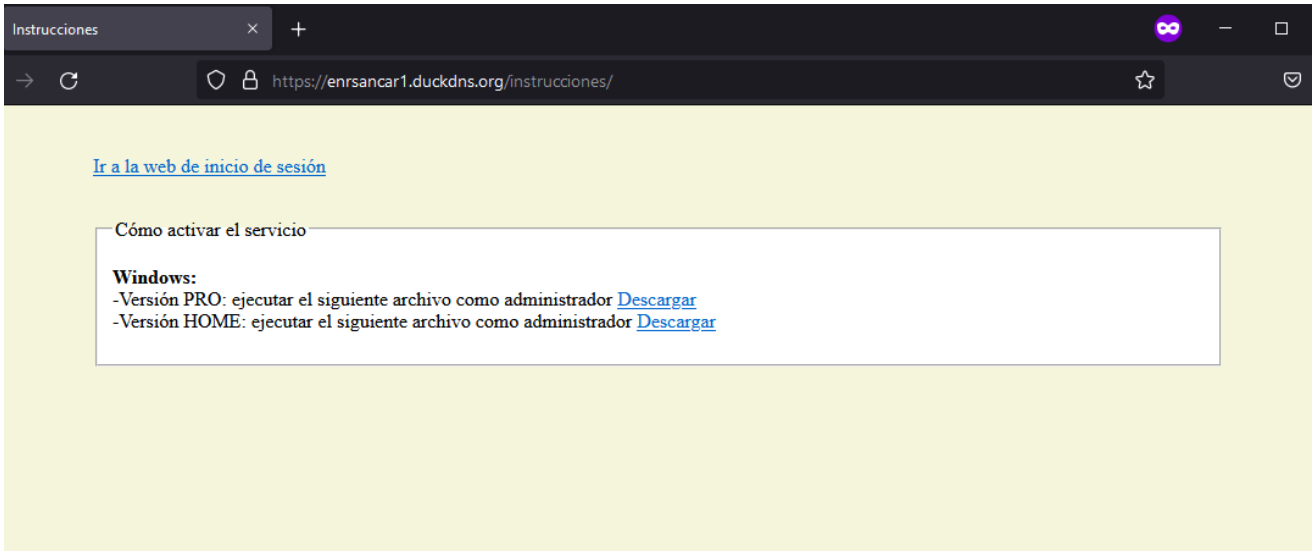

Figura 4-16. Instrucciones para Windows

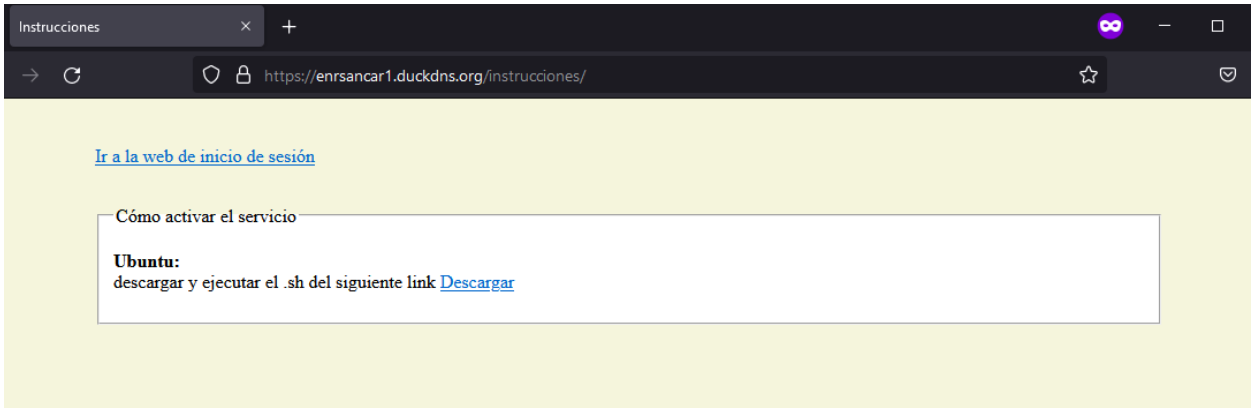

Figura 4-17. Instrucciones para Ubuntu

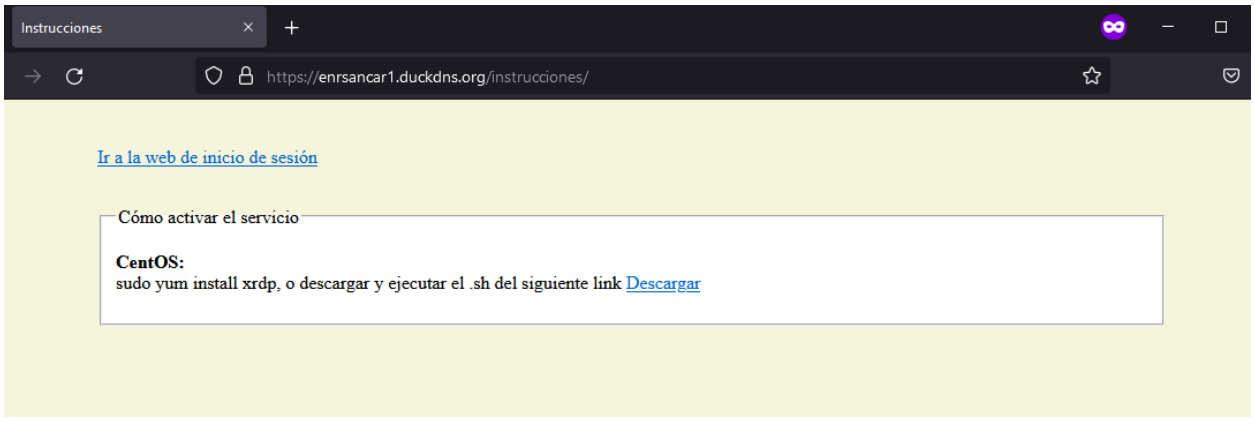

Figura 4-18. Instrucciones para CentOS

El siguiente paso es avisar al usuario administrador para que le registre en la plataforma y cree una conexión a dicho dispositivo para su usuario.

4.2.2 Plataforma de Apache Guacamole

Para utilizar la plataforma, el usuario tendrá que hacer lo siguiente:

4.2.2.1 Inicio de sesión

Una vez hecho esto, se podrá acceder a la plataforma mostrando el certificado digital y usando el usuario y contraseña provisto.

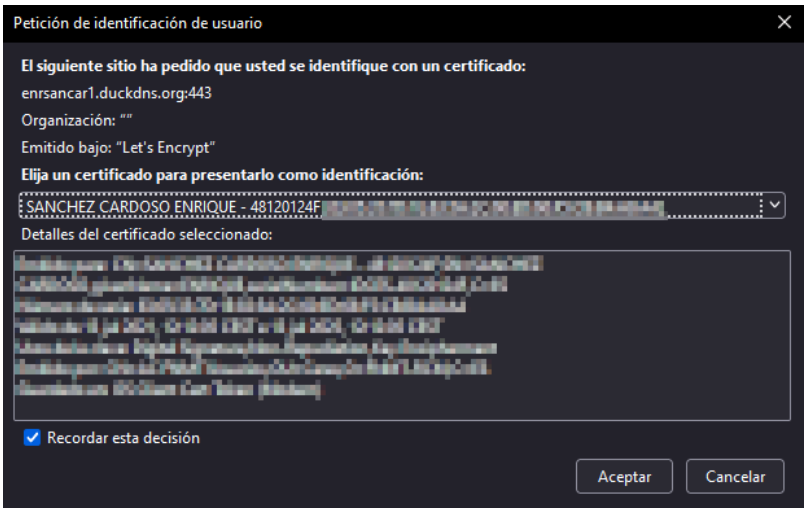

Figura 4-19. Solicitud de certificado digital del cliente

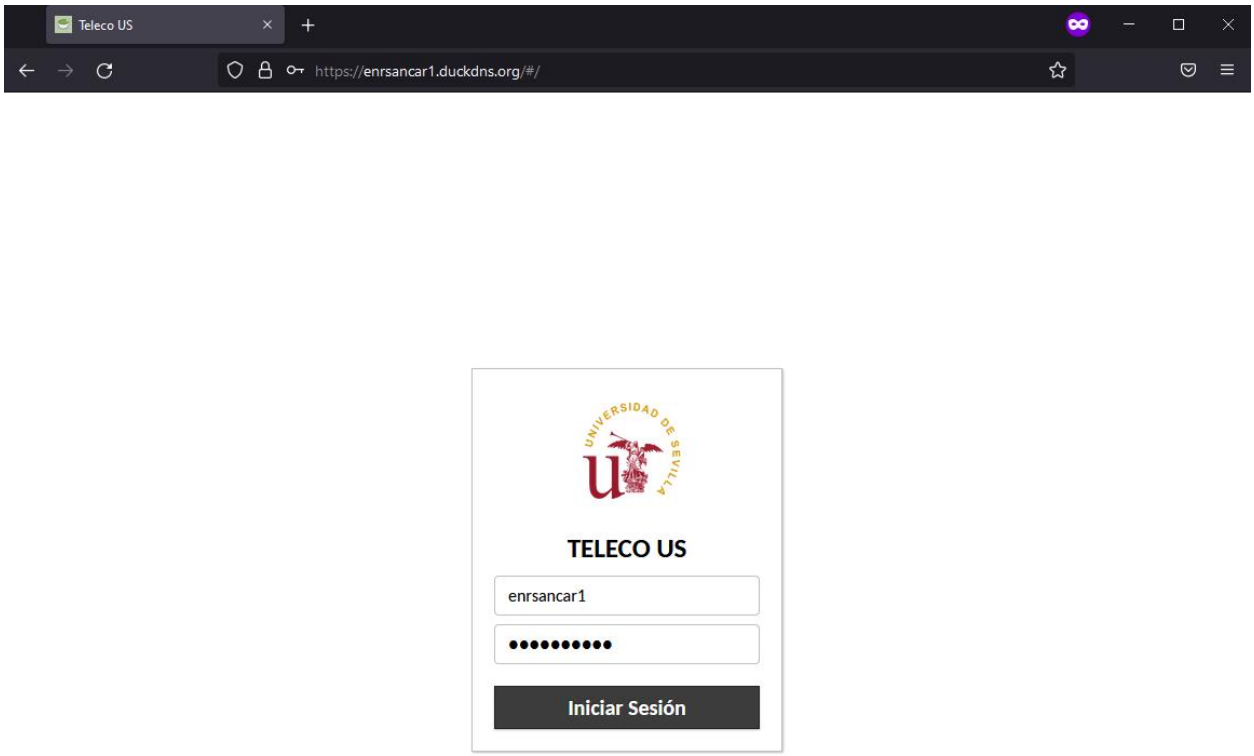

Figura 4-20. Inicio de sesión del usuario

4.2.2.2 Conexión remota

Para utilizar un dispositivo de los que tiene permitido, solo tendrá que pinchar en la fila correspondiente de la tabla “Todas las conexiones”.

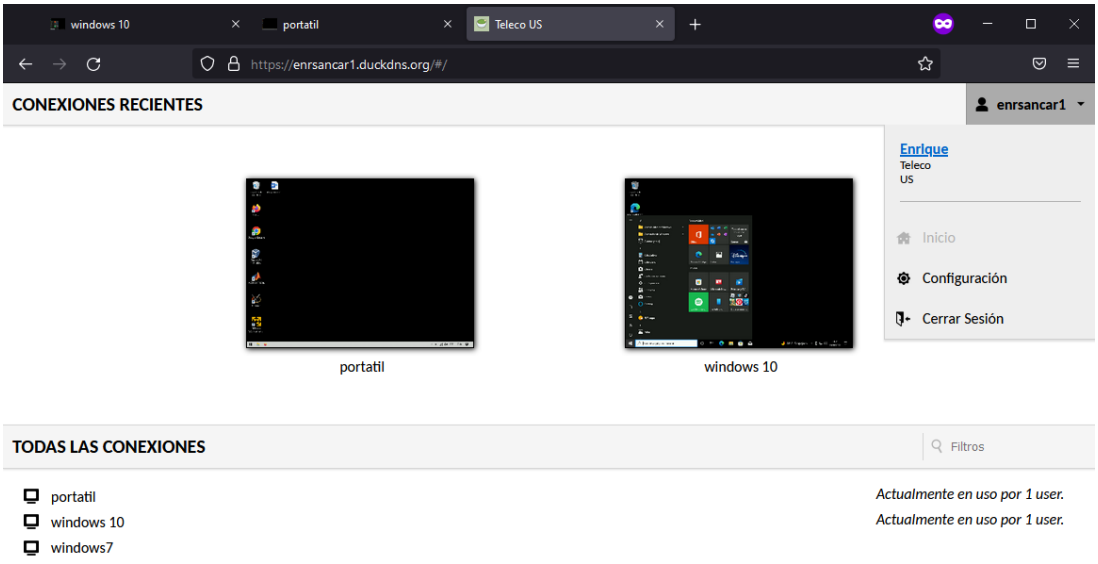

Figura 4-21. Conexiones de usuario

Podrá tener incluso varias conexiones remotas abiertas al mismo tiempo. Como aparece en la imagen, el usuario está conectado a “windows 10” y “portátil” a la vez.

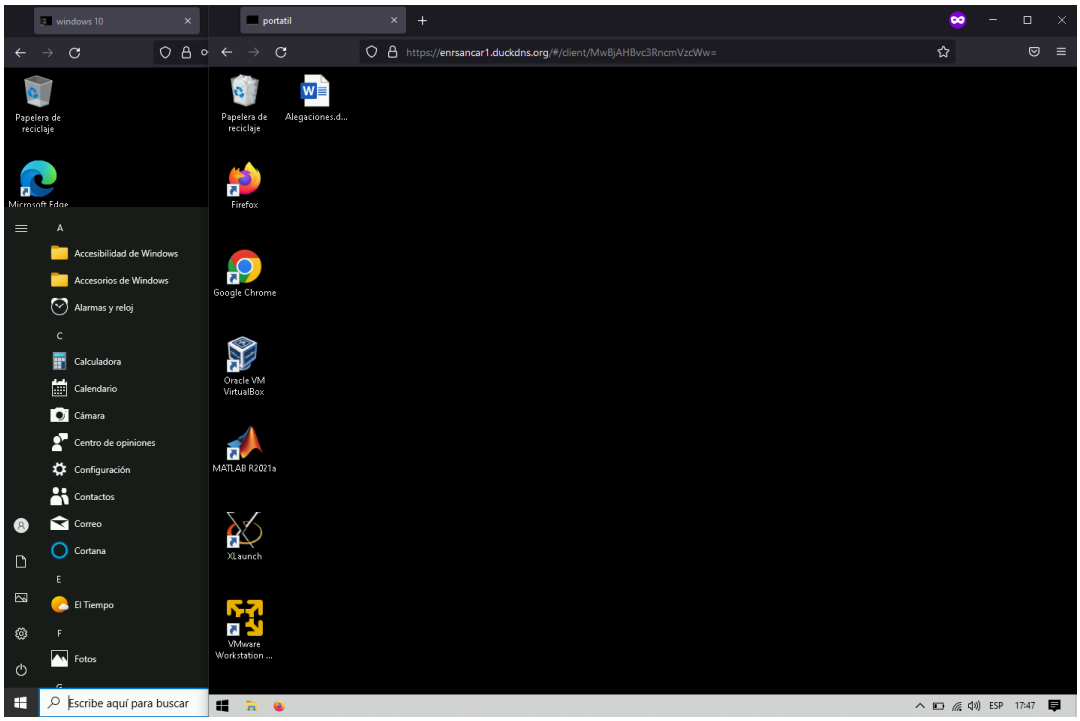

Figura 4-22. Dos conexiones abiertas al mismo tiempo

Si el administrador no ha introducido usuario y contraseña para el dispositivo en la creación de la conexión se mostrará un menú para introducirlos.

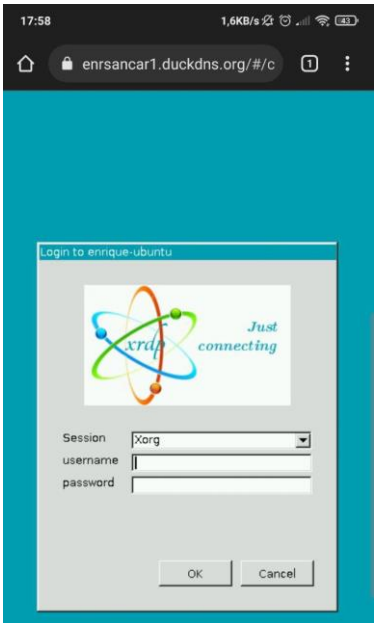

Figura 4-23. Inicio de sesión en dispositivo sin usuario/contraseña preconfigurado

Para obtener más información puede consultar el apartado uso ded la documentación de Apache Guacamole [41].

4.3 Compatibilidad

Se han realizado pruebas de compatibilidad con éxito accediendo remotamente a los siguientes sistemas operativos:

- Windows:
    - Windows XP
    - Windows 7
    - Windows 10
    - Windows 11

- Distribuciones basadas en Linux:
    - Ubuntu
    - CentOS

Para estas pruebas se han creado dos máquinas virtuales para cada SO Windows, una con la versión Home y otra con la Pro, y una para cada distribución basada en Linux.

En cuanto al sistema operativo desde el que se use el cliente de Apache Guacamole el único requisito es tener un navegador web que soporte HTML5, ya puede ser una tablet, móvil, ordenador, etc.

5 CONCLUSIÓN

Con este trabajo se llega a la conclusión de que uniendo diferentes piezas de software libre se pueden llegar a hacer servicios muy útiles, elaborados y sofisticados.

Un sistema de este tipo, a parte de hacer más fácil la vida de los empleados y jefes, permite aumentar la calidad y satisfacción del trabajo de estos, proporcionando un método sencillo de acceder a los dispositivos de la empresa desde cualquier lugar.

El punto más importante ha sido ponerse en la piel del cliente o usuario, entender sus necesidades y buscarles una solución. Esto ha sido vital para comprender que lo que todo cliente desea es que el dispositivo sea seguro, barato y funcione sin que necesite excesiva configuración, de ahí que se haya diseñado para ser Plug-and-Play y que reutilice el servidor RDP de Windows, el sistema operativo más popular, instalado por defecto.

Personalmente, este proyecto ha supuesto no solo implementar una manera de teletrabajar, sino aprender a conocer muchas tecnologías y a crear puentes entre ellas para usarlas como conjunto, valorar el trabajo de las comunidades de software libre como Apache, investigar entre los cientos de páginas de documentación y persistir.

Finalmente, es conveniente destacar los siguientes puntos sobre lecciones aprendidas que ayudarán en proyectos futuros:

- La organización es la base de los proyectos.
- Hay que tener siempre en mente las necesidades para no desviar la atención de los objetivos.
- Es necesario utilizar un control de versiones y copias de seguridad para proyectos que se alarguen en el tiempo.
- Documentar las decisiones tomadas ayuda a no tener que volver a preguntarse lo mismo varias veces.
- La bibliografía y documentación oficial de los diferentes proyectos siempre será la más completa y actualizada.
- El inglés es esencial puesto que mucha de la documentación y bibliografía, donde estará la solución de la mayor parte de errores y problemas, usará este idioma.

6 LÍNEAS DE MEJORA

En la carrera por la calidad no hay línea de meta.
- David T. Kearns -

El marco de este trabajo es muy amplio, por lo que admite varias líneas de mejora:

- **Autodestrucción de la instancia EC2:** Durante el tiempo configurado para que la instancia EC2 esté creada, puede que haya una avería, corte de luz, corte de Internet, etc. en la empresa. Esto provocará que nuestro sistema no esté funcionando, pero la instancia de AWS seguirá creada y por lo tanto suponiendo un coste. Para solucionar esto podríamos crear un mecanismo que destruyese la instancia si lleva un periodo de tiempo sin conectarse al servidor mediante el túnel SSH.
- **Registro de conexiones:** Una de las tareas del Administrador es crear las conexiones a cada dispositivo de la empresa para que se pueda usar remotamente, una vez el empleado que lo quiere usar haya seguido las instrucciones. Este proceso podría automatizarse si el ejecutable que descarga el usuario hiciese una petición a un servicio alternativo que registre la conexión a ese dispositivo en la base de datos que usa Apache Guacamole.
- **Cambio de horario:** Cabe la posibilidad de que una empresa cambie el horario de la jornada laboral por lo que puede interesarle cambiar el horario en el que funciona el sistema de teletrabajo. Podría hacerse una tercera sección en el panel de administrador para esta tarea.
- **Gestión remota del servidor:** Una aplicación para gestionar de manera remota el servidor sin necesitar estar en la misma LAN sería útil, por si surgen inconvenientes.

GLOSARIO

- VirtualHost: se refiere a la práctica de tener más de un sitio web en una sola máquina. Estos pueden estar basados en la dirección IP, es decir, variando esta IP se accede a las diferentes webs, o basados en nombre que, cambiando el nombre de dominio sin cambiar la IP, se accede a diferentes sitios.

- DocumentRoot: define la ruta a la que apunta la raíz del sitio web.

- Directory: Agrupa un número de directivas de Apache que afectan a dicho directorio, sus subdirectorios y su contenido.

- Location: Agrupa un número de directivas que aplican a la url especificada.

- Resource (Terraform): Es un bloque de Terraform que describe uno o más objetos de infraestructura como redes virtuales, instancias de computación o componentes de nivel superior como registros DNS.

- Provider (Terraform): Son plugins para interactuar con proveedores de la nube, proveedores de SaaS y otras APIs.

- Provisioner (Terraform): Es un bloque de Terraform que provee al usuario de funciones que pueden ejecutarse en la máquina local o remota con el fin de preparar servidores u otros objetos de infraestructura.

- Logging: consiste en la grabación secuencial de todos los acontecimientos que afectan a un proceso en particular para así comprender su funcionamiento, detectar errores, etc.

- Hash: una función resumen o *hash* es una función matemática cuya entrada es uno o varios elementos y los convierte en una salida, normalmente cadena de longitud fija. A partir de la salida no podremos obtener la entrada.

- Sal: se trata de una cadena aleatoria que se añade a las contraseñas antes de realizar la función hash. Esto ayuda a evitar el uso de bases de datos pre-generadas con hashes de contraseñas.

REFERENCIAS

[1] Ministerio de Indistria, Comercio y Turismo, «Definición de PYME en la UE,» 26 Junio 2014. [En línea]. Available: http://www.ipyme.org/es-ES/UnionEuropea/UnionEuropea/PoliticaEuropea/Marco/Paginas/NuevaDefinicionPYME.aspx. [Último acceso: 7 Junio 2022].

[2] BBC, «A 15 pound computer to inspire young programmers,» 5 Mayo 2011. [En línea]. Available: https://www.bbc.co.uk/blogs/thereporters/rorycellanjones/2011/05/a_15_computer_to_inspire_young.html. [Último acceso: 6 Junio 2022].

[3] BuiltWith, «Ubuntu Usage Statistics,» [En línea]. Available: https://trends.builtwith.com/Server/Ubuntu. [Último acceso: 6 Junio 2022].

[4] Apache Software Foundation, «Apache HTTP Server Project,» [En línea]. Available: https://httpd.apache.org/. [Último acceso: 7 Junio 2022].

[5] Apache Software Foundation, «Apache Tomcat,» [En línea]. Available: https://tomcat.apache.org/. [Último acceso: 6 Junio 2022].

[6] WebTechSurvey, «Apache Tomcat market share and usage statistics,» [En línea]. Available: https://webtechsurvey.com/technology/apache-tomcat. [Último acceso: 7 Junio 2022].

[7] Apache Software Foundation, «Apache Guacamole,» [En línea]. Available: https://guacamole.apache.org/. [Último acceso: 7 Junio 2022].

[8] Amazon AWS, «Contenedores de Docker,» [En línea]. Available: https://aws.amazon.com/es/docker/. [Último acceso: 5 Junio 2022].

[9] Docker, «What is a Container?,» [En línea]. Available: https://www.docker.com/resources/what-container/. [Último acceso: 6 Junio 2022].

[10] HashiCorp, «Terraform,» [En línea]. Available: https://www.terraform.io/. [Último acceso: 7 Junio 2022].

[11] HashiCorp, «Licencia de Terraform,» [En línea]. Available: https://github.com/hashicorp/terraform/blob/main/LICENSE. [Último acceso: 8 Junio 2022].

[12] Apache Software Foundation, «Tutorial de Apache: Contenido dinámico con CGI,» [En línea]. Available: https://httpd.apache.org/docs/2.4/howto/cgi.html. [Último acceso: 8 Junio 2022].

[13] IETF, «RFC 3875,» [En línea]. Available: https://datatracker.ietf.org/doc/html/rfc3875. [Último acceso: 8 Junio 2022].

[14] G. Lyon, «Nmap,» [En línea]. Available: https://nmap.org/. [Último acceso: 7 Junio 2022].

[15] Duck DNS, «Duck DNS,» [En línea]. Available: https://www.duckdns.org/about.jsp. [Último acceso: 6

Junio 2022].

[16] Internet Security Research Group, «Acerda de Let's Encrypt,» [En línea]. Available: https://letsencrypt.org/es/about/. [Último acceso: 9 Junio 2022].

[17] Internet Security Research Group, «Estadísticas de Let's Encrypt,» [En línea]. Available: https://letsencrypt.org/es/stats/. [Último acceso: 9 Junio 2022].

[18] IETF, «RFC 8555,» [En línea]. Available: https://datatracker.ietf.org/doc/html/rfc8555. [Último acceso: 9 Junio 2022].

[19] Electronic Frontier Foundation, «Certbot,» [En línea]. Available: https://certbot.eff.org/. [Último acceso: 9 Junio 2022].

[20] Electronic Frontier Foundation, «About Certbot,» [En línea]. Available: https://certbot.eff.org/pages/about. [Último acceso: 9 Junio 2022].

[21] Wikipedia, [En línea]. Available: https://es.wikipedia.org/wiki/OpenSSL. [Último acceso: 12 Junio 2022].

[22] Wikipedia, «cron,» [En línea]. Available: https://en.wikipedia.org/wiki/Cron. [Último acceso: 8 Junio 2022].

[23] Wiki ArchLinux, «Timers de systemd,» [En línea]. Available: https://wiki.archlinux.org/title/Systemd_(Espa%C3%B1ol)/Timers_(Espa%C3%B1ol). [Último acceso: 12 Junio 2022].

[24] Red Hat , «Protocolo SSH,» [En línea]. Available: https://web.mit.edu/rhel-doc/4/RH-DOCS/rhel-rg-es-4/ch-ssh.html. [Último acceso: 7 Junio 2022].

[25] IETF, «RFC 4253 - SSH,» [En línea]. Available: https://datatracker.ietf.org/doc/html/rfc4253. [Último acceso: 7 Junio 2022].

[26] SSH Academy, «SSH port forwarding/tunneling,» [En línea]. Available: https://www.ssh.com/academy/ssh/tunneling/example. [Último acceso: 7 Junio 2022].

[27] Ivanti, «¿Qué es RDP?,» [En línea]. Available: https://help.ivanti.com/iv/help/es_ES/isec/94/Topics/Understanding_RDP.htm. [Último acceso: 6 Junio 2022].

[28] Wikipedia, «Remote Desktop Protocol - Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Remote_Desktop_Protocol. [Último acceso: 7 Junio 2022].

[29] Amazon AWS, «¿Qué es Amazon EC2?,» [En línea]. Available: https://docs.aws.amazon.com/es_es/AWSEC2/latest/UserGuide/concepts.html. [Último acceso: 7 Junio 2022].

[30] Ionos, «WebSocket | Un canal de comunicación para la web en tiempo real,» [En línea]. Available: https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/que-es-websocket/. [Último acceso: 12 Junio 2022].

[31] Fundación Mozilla, «Ajax - Guía de desarrollo web,» [En línea]. Available:

https://developer.mozilla.org/es/docs/Web/Guide/AJAX. [Último acceso: 10 Junio 2022].

[32] Amazon AWS, «Buenas prácticas para mantener EC2 seguro,» [En línea]. Available: https://aws.amazon.com/es/premiumsupport/knowledge-center/ec2-ssh-best-practices/. [Último acceso: 4 Junio 2022].

[33] HashiCorp, «Network Requirements - Terraform,» [En línea]. Available: https://www.terraform.io/enterprise/requirements/network. [Último acceso: 1 Junio 2022].

[34] oznu, «oznu/guacamole - Docker Image,» [En línea]. Available: https://hub.docker.com/r/oznu/guacamole/. [Último acceso: 20 Marzo 2022].

[35] Apache Software Foundation, «Configuring Guacamole,» [En línea]. Available: https://guacamole.apache.org/doc/gug/configuring-guacamole.html. [Último acceso: 13 Junio 2022].

[36] Wireshark, «Wake on Lan,» [En línea]. Available: https://wiki.wireshark.org/WakeOnLAN. [Último acceso: 12 Junio 2022].

[37] Apache Software Foundation, «Proxying Guacamole - Apache Guacamole Manual,» [En línea]. Available: https://guacamole.apache.org/doc/gug/reverse-proxy.html. [Último acceso: 12 Junio 2022].

[38] M. Jumper, «Performance testing of Guacamole,» [En línea]. Available: https://lists.apache.org/thread/hdxg4f17qgqhbspxqy6qp4go1g5qvfvq. [Último acceso: 17 Junio 2022].

[39] Apache Software Foundation, «Database authentication,» [En línea]. Available: https://guacamole.apache.org/doc/gug/jdbc-auth.html. [Último acceso: 24 Junio 2022].

[40] Apache Software Foundation, «Administration - Apache Guacamole,» [En línea]. Available: https://guacamole.apache.org/doc/gug/administration.html. [Último acceso: 1 Junio 2022].

[41] Apache Software Foundation, «Using Guacamole - Apache Guacamole,» [En línea]. Available: https://guacamole.apache.org/doc/gug/using-guacamole.html. [Último acceso: 1 Junio 2022].

[42] WRK, «wrk-guacamole-client,» [En línea]. Available: https://github.com/wrktech/wrk-guacamole-client/tree/72c6279579e40eb452be923b585ccdf8ea016a14/doc/guacamole-branding-example. [Último acceso: 6 Junio 2022].

[43] sebaxakerhtc. [En línea]. Available: https://github.com/sebaxakerhtc/rdpwrap. [Último acceso: 3 Junio 2022].

[44] Paul Vixie, «cron - Linux man page,» [En línea]. Available: https://linux.die.net/man/8/cron. [Último acceso: 6 Junio 2022].

[45] J. M. Calle, «Apuntes de la asignatura Servicios Telemáticos Avanzados,» 2022.

ANEXO A

apache_conf/panel.conf

```
<IfModule mod_ssl.c>
<VirtualHost IP_LOCAL:443>
ServerAdmin webmaster@localhost
DocumentRoot /var/www/panel
ErrorLog ${APACHE_LOG_DIR}/error.log
CustomLog ${APACHE_LOG_DIR}/access.log combined

        ServerName IP_LOCAL

        SSLEngine on
        SSLCertificateFile /etc/ssl/certs/apache-selfsigned.crt
        SSLCertificateKeyFile /etc/ssl/private/apache-selfsigned.key

        <Directory /var/www/panel>
          <RequireAll>
            <RequireAny>
              Require ip 192.168
              Require ip 172.16.0.0/12
              Require ip 10
            </RequireAny>
            AuthType Basic
            AuthName "Panel de administración"
            AuthUserFile "/usr/local/apache/passwd/passwords"
            Require valid-user
          </RequireAll>
        </Directory>


<IfDefine ENABLE_USR_LIB_CGI_BIN>
          ScriptAlias /cgi-bin/ /usr/lib/cgi-bin/
          <Directory "/usr/lib/cgi-bin">
            AllowOverride None
            Options +ExecCGI -MultiViews
            <RequireAll>
              <RequireAny>
                Require ip 192.168
                Require ip 172.16.0.0/12
                Require ip 10
              </RequireAny>
              AuthType Basic
              AuthName "Panel de administración"
              AuthUserFile "/usr/local/apache/passwd/passwords"
              Require valid-user
            </RequireAll>
          </Directory>
</IfDefine>
</VirtualHost>
</IfModule>
```

apache_conf/serve-cgi-bin.conf

```
  <IfModule mod_alias.c>

  <IfModule mod_cgi.c>
    Define ENABLE_USR_LIB_CGI_BIN
  </IfModule>

  <IfModule mod_cgid.c>
    Define ENABLE_USR_LIB_CGI_BIN
  </IfModule>

  <IfModule mod_alias.c>
  <IfModule mod_cgi.c>
    Define ENABLE_USR_LIB_CGI_BIN
  </IfModule>
  <IfModule mod_cgid.c>
    Define ENABLE_USR_LIB_CGI_BIN
  </IfModule>

  #<IfDefine ENABLE_USR_LIB_CGI_BIN>
    #ScriptAlias /cgi-bin/ /usr/lib/cgi-bin/
    #<Directory "/usr/lib/cgi-bin">
      #AllowOverride None
      #Options +ExecCGI -MultiViews +SymLinksIfOwnerMatch
      #Require all granted
    #</Directory>
  #</IfDefine>

  </IfModule>
  </IfModule>
```

cgi-bin/leer.cgi

```
  #!/bin/bash

  echo "Content-type: text/plain"
  echo ""

  string=$(cat /home/www-data/usuarios.txt)

  #elimino primer y ultimo caracter "'"
  #cambio ',' por saltos de linea
  echo ${string:1:-1} | sed 's|''\'','\'''|\n|g'

  exit 0
```

cgi-bin/nmap.cgi

```
  #!/bin/bash

  LOCAL=$(ip route get 8.8.8.8 | head -1 | cut -d' ' -f7)
  MASK=$(ip a | grep ${LOCAL} | grep -oE "/[0-9]{1,2}")
  DEVICE=$(ip route get 8.8.8.8 | head -1 | cut -d' ' -f5)

  DATA=`nmap -p 3389 -Pn -n ${LOCAL}${MASK} -oX - | xmllint --xpath
'/nmaprun/host/ports/port[@portid="3389"]/state[@state="open"] |
/nmaprun/host/address[@addrtype="ipv4"]' -`

  echo "content-type: text/plain"
```

```
  echo
  echo '<ul>'

  readarray -t DATA_PARSED <<< "$DATA"

  for (( i=0; i<"${#DATA_PARSED[@]}"; i++ ))
  do
      j=i+1
      if [[ i -lt "${#DATA_PARSED[@]}" && "${DATA_PARSED[$i]}" == *"addr="* &&
"${DATA_PARSED[$j]}" == *"state="* ]]
      then
          printf "<li>"
          IP=$(echo "${DATA_PARSED[$i]}}" | sed 's/[^"]*"\([^"]*\).*/\1/')
          printf "IP=<b>${IP}</b> "
          MAC=$(ip n get $IP dev $DEVICE | head -1 | cut -d' ' -f5)
          printf "MAC=<b>${MAC}</b> "
          printf "<input type='text' name='connection_name'
placeholder='nombre de la conexión'>"
          printf "<input type='text' name='user_create'
placeholder='usuario'>"
          printf "<input type='password' name='password_create'
placeholder='contraseña'>"
          printf "<input type='submit' value='Crear' class='enviar_conexion'>"
          printf "</li>"
      fi
  done

  echo "</ul>"

  exit 0
```

cgi-bin/recibir.cgi

```
  #!/bin/bash

  echo "Content-type: text/html"
  echo ""

  echo '<html>'
  echo '<head>'
  echo '<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">'
  echo '<title>Foo</title>'
  echo '</head>'
  echo '<body>'

  if [ "$REQUEST_METHOD" = "POST" ]; then
      if [ "$CONTENT_LENGTH" -gt 0 ]; then

          string=$(cat | sed "s/\"//g" | sed "s/\`//g" | sed "s/\\\//g" | sed
"s/{//g" | sed "s/}//g" | sed "s/(//g" | sed "s/)//g" | sed "s/#//g")

          echo $string > /home/www-data/usuarios.txt
          cat /home/www-data/usuarios.txt

          sudo /etc/apache2/sincronizar_usuarios.sh
          echo $?
      fi
  fi


  echo '</body>'
```

```
  echo '</html>'

  exit 0
```

cgi-bin/crear.cgi

```
  #!/bin/bash

  json=$(cat)

  echo "Content-type: text/html"

  #Le paso el json como primer parámetro a los scripts de creacion de usuario
y conexion asociada a este
  /usr/bin/python3 /etc/apache2/crear_usuario.py $json

  if [[ $? -eq 0 ]]; then
    echo ""
    /usr/bin/python3 /etc/apache2/crear_conexion.py $json

  else
    echo "Status: 400 Bad Request"
    echo ""

    echo "El usuario ya existe y has introducido mal la contraseña"
    exit 1

  fi

  exit 0
```

cgi-bin/crear_usuario.py

```
  #!/usr/bin/python3

  ##
  ## Script para crear usuario de Apache Guacamole
  ##

  #La documentación oficial de Apache Guacamole lo hace así
  #SET @salt =
UNHEX("fe24adc5e11e2b25288d1704abe67a79e342ecc26064ce69c5b3177795a82264");
  #UNHEX(SHA2(CONCAT('guacadmin', HEX(@salt)), 256))

  #apache guacamole aplica el siguiente salt para sus hashes
  #primero obtiene un salt aleatorio de 32 bytes en hexadecimal
  #esta cadena hexadecimal la convierte a un string en mayusculas y la
concatena
  #con la contraseña en texto plano del usuario añadiendo el salt al final

  import sys
  import json
  import hashlib
  import secrets
  import psycopg2

  json = json.loads(sys.argv[1])

  #Compruebo que todos los parámetros necesarios han sido introducidos
  if("password_create" in json and "user_create" in json):
```

```
    connection = psycopg2.connect(host="localhost", database="guacamole_db",
user="guacamole")
    cursor = connection.cursor()

    cursor.execute("select name, password_hash, password_salt from
guacamole_user inner join guacamole_entity on
guacamole_user.entity_id=guacamole_entity.entity_id where name=%s;",
(json["user_create"],))
    existente = cursor.fetchall()

    #Si el usuario no existe, lo creo
    if(len(existente) == 0):
      #generamos un salt como Apache Guacamole
      password_salt = hex(secrets.randbits(32*8)).lstrip('0x').upper()

      #concatenamos contraseña y salt y hacemos hash sha256
      password_hash = hashlib.sha256((json["password_create"] +
password_salt).encode())

      #Insertamos los datos en PostgreSQL
      cursor.execute("insert into guacamole_entity (name, type) values (%s,
'USER');", (json["user_create"],))
      cursor.execute("insert into guacamole_user (entity_id, password_hash,
password_salt, password_date) \
      select entity_id, decode(%s, 'hex'), decode(%s, 'hex'),
CURRENT_TIMESTAMP as group \
      from guacamole_entity where name=%s and type='USER';",
(password_hash.hexdigest(), password_salt, json["user_create"]))

      connection.commit()
      cursor.close()
      connection.close()

    else:

      #Si el usuario no existe, compruebo que la contraseña introducida es la
correcta
      password_salt = bytes.hex(bytes(existente[0][2])).lstrip('0x').upper()
      password_hash_correcto = bytes.hex(bytes(existente[0][1]))

      #concatenamos contraseña y salt y hacemos hash sha256
      password_hash = hashlib.sha256((json["password_create"] +
password_salt).encode()).hexdigest()

      if(password_hash == password_hash_correcto):
        #Si la contraseña es correcta, no tenemos que hacer nada, el otro
script registrará la conexión
        cursor.close()
        connection.close()

      else:
        #Si la contraseña es incorrecta, mostramos error
        cursor.close()
        connection.close()
        sys.exit("Contraseña incorrecta")

  else:
    sys.exit("Faltan campos por rellenar para la creación del usuario")
```

cgi-bin/crear_conexion.py

```
  #!/usr/bin/python3

  ##
  ## Script para crear una conexion de Apache Guacamole y asignarle un usuario
  ##

  import sys
  import json
  import psycopg2

  json = json.loads(sys.argv[1])

  #Compruebo que todos los parámetros necesarios han sido introducidos
  if("connection_name" in json and "connection_ip" in json and "mac" in json
and "user_create" in json):
    connection = psycopg2.connect(host="localhost", database="guacamole_db",
user="guacamole")
    cursor = connection.cursor()

    cursor.execute("select * from guacamole_connection_parameter where
parameter_name='hostname' and parameter_value=%s;", (json["connection_ip"],))
    conexiones = cursor.fetchall()

    #Si la conexión a esa ip no existe, la creo
    if(len(conexiones) == 0):
      cursor.execute("insert into guacamole_connection (connection_name,
protocol) values (%s, 'rdp');", (json["connection_name"],))

      cursor.execute("insert into guacamole_connection_parameter
(connection_id, parameter_name, parameter_value) select connection_id,
'hostname', %s from guacamole_connection where connection_name=%s;",
(json["connection_ip"], json["connection_name"]))

      cursor.execute("insert into guacamole_connection_parameter
(connection_id, parameter_name, parameter_value) select connection_id,
'password', '${GUAC_PASSWORD}' from guacamole_connection where
connection_name=%s;", (json["connection_name"],))

      cursor.execute("insert into guacamole_connection_parameter
(connection_id, parameter_name, parameter_value) select connection_id, 'wol-
mac-addr', %s from guacamole_connection where connection_name=%s;",
(json["mac"], json["connection_name"]))

      cursor.execute("insert into guacamole_connection_parameter
(connection_id, parameter_name, parameter_value) select connection_id,
'ignore-cert', 'true' from guacamole_connection where connection_name=%s;",
(json["connection_name"],))

      cursor.execute("insert into guacamole_connection_parameter
(connection_id, parameter_name, parameter_value) select connection_id, 'wol-
send-packet', 'true' from guacamole_connection where connection_name=%s;",
(json["connection_name"],))

      cursor.execute("insert into guacamole_connection_parameter
(connection_id, parameter_name, parameter_value) select connection_id,
'username', '${GUAC_USERNAME}' from guacamole_connection where
connection_name=%s;", (json["connection_name"],))

    #Asocio la conexión con el usuario
```

```
    cursor.execute("insert into guacamole_connection_permission (entity_id,
connection_id, permission) values ((select entity_id from guacamole_entity
where name=%s), (select distinct connection_id from
guacamole_connection_parameter where parameter_name='hostname' and
parameter_value=%s), 'READ');", (json["user_create"], json["connection_ip"]))

    connection.commit()
    cursor.close()
    connection.close()

  else:
    sys.exit("Faltan campos por rellenar para la creación de la conexión")
```

cgi-bin/crear_todas.cgi

```
  #!/bin/bash

  json=$(cat)

  echo "Content-type: text/html"

  #Le paso el json como primer parámetro a los scripts de creacion de usuario
y conexion asociada a este
  /usr/bin/python3 /etc/apache2/crear_usuario.py $json

  if [[ $? -eq 0 ]]; then
    echo ""
    /usr/bin/python3 /etc/apache2/crear_conexion_todas.py $json

  else
    echo "Status: 400 Bad Request"
    echo ""

    echo "El usuario ya existe y has introducido mal la contraseña"
    exit 1

  fi

  exit 0
```

cgi-bin/crear_conexion_todas.py

```
  #!/usr/bin/python3

  ##
  ## Script para crear una conexion de Apache Guacamole y asignarle un usuario
  ##

  import sys
  import json
  import psycopg2

  json = json.loads(sys.argv[1])

  #Compruebo que todos los parámetros necesarios han sido introducidos
  if("user_create" in json):
    connection = psycopg2.connect(host="localhost", database="guacamole_db",
user="guacamole")
    cursor = connection.cursor()
```

```
    cursor.execute("select distinct connection_id from guacamole_connection;")
    conexiones = cursor.fetchall()

    #Si hay conexiones creadas, le asigno todas
    if(len(conexiones) != 0):
      for conexion in conexiones:

        cursor.execute("select 1 from guacamole_connection_permission where
entity_id=(select entity_id from guacamole_entity where name=%s) and
guacamole_connection_permission.connection_id=%s", (json["user_create"],
conexion[0]))
        existente = cursor.fetchall()

        if(len(existente) == 0):
        #Asocio las conexiones con el usuario si no está
          cursor.execute("insert into guacamole_connection_permission
(entity_id, connection_id, permission) values ((select entity_id from
guacamole_entity where name=%s), %s, 'READ');", (json["user_create"],
conexion[0]))

      connection.commit()

    cursor.close()
    connection.close()

  else:
    sys.exit("Faltan campos por rellenar para la creación de la conexión")
```

cambiar_guacadmin.py

```
  #!/usr/bin/python3

  ##
  ## Script para cambiar la contraseña del usuario por defecto, guacadmin
  ##

  #apache guacamole aplica el siguiente salt para sus hashes
  #primero obtiene un salt aleatorio de 32 bytes en hexadecimal
  #esta cadena hexadecimal la convierte a un string en mayusculas y la
concatena
  #con la contraseña en texto plano del usuario añadiendo el salt al final

  import sys
  import hashlib
  import secrets
  import psycopg2

  password = sys.argv[1]

  #Compruebo que la contraseña ha sido introducida
  if(len(password) > 0):

    connection = psycopg2.connect(host="localhost", database="guacamole_db",
user="guacamole")
    cursor = connection.cursor()

    #generamos un salt como Apache Guacamole
    password_salt = hex(secrets.randbits(32*8)).lstrip('0x').upper()

    #concatenamos contraseña y salt y hacemos hash sha256
    password_hash = hashlib.sha256((password + password_salt).encode())
```

```
    print(password_hash.hexdigest(), password_salt)

    #Insertamos los datos en PostgreSQL
    cursor.execute("update guacamole_user set password_hash=decode(%s, 'hex'),
password_salt=decode(%s, 'hex'), password_date=CURRENT_TIMESTAMP \
    where entity_id=1;", (password_hash.hexdigest(), password_salt))

    connection.commit()
    cursor.close()
    connection.close()
```

cgi-bin/sincronizar_usuarios.sh

```
  #!/bin/bash

  string=`cat /home/www-data/usuarios.txt`

  if [[ $string == *"\\"* || $string == *"}"* || $string == *")"* || $string
== *"\""* || $string == *"#"* ]]; then
  echo "denegado"
  exit 1
  fi

  sed -i "s/in {.*}/in {${string}}/g" /etc/apache2/sites-enabled/000-default-
le-ssl.conf
  sed -i "s/in {.*}/in {${string}}/g" /etc/apache2/sites-enabled/panel.conf

  if [[ $string = "" ]]
  then
    sed -i "s/in {.*}/in {''}/g" /etc/apache2/sites-enabled/000-default-le-
ssl.conf
    sed -i "s/in {.*}/in {''}/g" /etc/apache2/sites-enabled/panel.conf
  fi

  service apache2 reload
```

cron-ssh-tunnel/scripts/tunnel.sh

```
  tunnel() {
  #Crea tunel entre 4430(remoto):443(local) y 8080(remoto):80(local)
  #Además de un tunel entre 6000(local):22(remoto) para comprobar si el tunel
sigue activo

  ssh -i $KEY ec2-user@$HOST -R 0.0.0.0:8080:0.0.0.0:80 -R
0.0.0.0:4430:0.0.0.0:443 -L localhost:6000:$HOST:22 -N -o
StrictHostKeychecking=no &

  echo $! > $DIR/tunnel.pid
  return 0
  }
  ssh -i $KEY ec2-user@localhost -p 6000 -o StrictHostKeychecking=no ls

  #Si la ejecucion remota del comando ls falla, vuelve a crear el tunel
  if [[ $? -ne 0 ]]; then
      echo creando tunel
      kill -9 `cat "$DIR/tunnel.pid"`
      rm $DIR/tunnel.pid
      tunnel
  fi
```

cron-ssh-tunnel/main.tf

```
  terraform {
    required_providers {
      aws = {
        source  = "hashicorp/aws"
        version = "~> 3.27"
      }
    }

    required_version = ">= 0.14.9"
  }

  #Clave RSA de 4096 bits
  resource "tls_private_key" "rsa-4096" {
    algorithm = "RSA"
    rsa_bits  = 4096
  }

  resource "aws_key_pair" "generated_key" {
    key_name   = var.key_name
    public_key = tls_private_key.rsa-4096.public_key_openssh

    tags = {
      dir = var.dir
    }

    provisioner "local-exec" { #Guarda la clave privada generada
      command = <<EOT
      echo '${tls_private_key.rsa-4096.private_key_pem}' >
${self.tags.dir}/priv_key/${self.key_name}.pem
      chmod 400 ${self.tags.dir}/priv_key/${self.key_name}.pem
      EOT
    }
  }


  provider "aws" {
    profile = "default"
    region  = "eu-west-3"
  }

  resource "aws_instance" "app_server" {
    ami           = "ami-0960de83329d12f2f"
    instance_type = "t2.micro"
    associate_public_ip_address = true
    key_name = aws_key_pair.generated_key.key_name
    vpc_security_group_ids = ["sg-049fe496ee9eefa30"]

    tags = {
      Name = "ssh tunnel"
      dir = var.dir
    }

    connection {
      type     = "ssh"
      user     = "ec2-user"
      private_key = file("${self.tags.dir}/priv_key/${self.key_name}.pem")
      host     = self.public_dns
    }

    provisioner "remote-exec" {
```

```
      inline = [
        "sudo chmod 606 /etc/ssh/sshd_config",
        "echo 'GatewayPorts clientspecified' >> /etc/ssh/sshd_config",
        "sudo service sshd restart",
        "sudo iptables -t nat -A PREROUTING -p tcp --dport 80 -j REDIRECT --
to-ports 8080", #redirijo el puerto 80 al 8080 (donde está el tunel ssh)
        "sudo iptables -t nat -A PREROUTING -p tcp --dport 443 -j REDIRECT --
to-ports 4430", #redirijo el puerto 443 al 4430 (donde está el tunel ssh)
        "echo
url='https://www.duckdns.org/update?domains=${var.duckdns_domain}&token=${var
.duckdns_token}&ip=' | curl -k -K -", #actualizo el dns dinamico
      ]
    }

    provisioner "local-exec" {
      command = <<EOT
        sudo echo -e
"SHELL=/bin/bash\nPATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/loca
l/sbin\n\n* * * * * root export HOST=${self.public_dns} ; export
KEY=${self.tags.dir}/priv_key/${self.key_name}.pem ; export
DIR=${self.tags.dir} ; ${self.tags.dir}/scripts/tunnel.sh > /dev/null 2>
${self.tags.dir}/log/error.log" > /etc/cron.d/tunnel-cron-job
        export HOST=${self.public_dns} ; export
KEY=${self.tags.dir}/priv_key/${self.key_name}.pem ; export
DIR=${self.tags.dir} ;  sudo ${self.tags.dir}/scripts/tunnel.sh > /dev/null
2> ${self.tags.dir}/log/error.log
        EOT
    }

    provisioner "local-exec" {
      when = destroy

      command = <<EOT
        sudo ssh-keygen -f "/root/.ssh/known_hosts" -R "[localhost]:6000"
        sudo rm /etc/cron.d/tunnel-cron-job
        EOT

      on_failure = continue  #destruye la instancia ec2 aunque el comando
anterior devuelva un error
    }
  }
```

cron-ssh-tunnel/variables.tf

```
  variable "dir" {
    description = "Path to the working dir"
    type        = string
    default     = "/root/cron-ssh-tunnel"
  }

  variable "key_name" {
    description = "Name of the generated key"
    type        = string
    default     = "mi_clave"
  }

  variable "duckdns_domain" {
    description = "Duckdns domain, used to dynamic dns"
    type        = string
    default     = "DOMAIN"
  }
```

```
  variable "duckdns_token" {
    description = "Duckdns token, used to update dynamic dns"
    type        = string
    default     = "DUCKDNS_TOKEN"
  }
```

guacamole-branding-example

Código descargado del repositorio de github wrk-guacamole-client [42], modificando los siguientes archivos.

guacamole-branding-example/css/example.css

```
  .login-ui .login-dialog .logo {
      width: 5em;
      height: 5em;
      -webkit-background-size: 5em 5em;
      background-image: url('app/ext/guacamole-branding-
example/resources/images/logo/logo.png');
  }
```

guacamole-branding-example/translations/en.json

```
  {
      "APP": {
          "NAME" : "EMPRESA"
      }
  }
```

guacamole-branding-example/guac-manifest.json

```
  {
      "guacamoleVersion" : "*",
      "name"             : "Guacamole Branding Example",
      "namespace"        : "guacamole-branding-example",

      "css" : [
          "css/example.css"
      ],

      "resources" : {
          "resources/images/logo/logo.png" : "image/png"
      },

      "translations" : [
          "translations/en.json"
      ]
  }
```

html/instrucciones/index.html

```
  <!DOCTYPE html>
  <html>
    <head>
      <meta charset="UTF-8">
```

```
      <title>Instrucciones</title>
      <link rel="stylesheet" href="estilos.css">
    </head>
    <body>
      <p id="inicio">
        <a href="/">Ir a la web de inicio de sesión</a>
      </p>
      <br>
      <fieldset>
      <legend>Cómo activar el servicio</legend>
      <p id="so"></p>
      </fieldset>
    </body>
      <script>
        var parrafo = document.getElementById("so");
        if(navigator.userAgent.includes("Ubuntu")) {
              parrafo.innerHTML = "<b>Ubuntu:</b><br>descargar y ejecutar el
.sh del siguiente link <a href=\"ubuntu.sh\">Descargar</a>";
        } else if(navigator.userAgent.includes("CentOS")) {
              parrafo.innerHTML = "<b>CentOS:</b><br>sudo yum install xrdp, o
descargar y ejecutar el .sh del siguiente link <a
href=\"centos.sh\">Descargar</a>";
        } else if(navigator.userAgent.includes("Windows")) {
              parrafo.innerHTML = "<b>Windows:</b> <br>-Versión PRO: ejecutar
el siguiente archivo como administrador <a
href=\"windowspro.bat\">Descargar</a><br>-Versión HOME: ejecutar el siguiente
archivo como administrador <a href=\"RDPW_installer.exe\">Descargar</a>";
        }
      </script>
  </html>
```

html/instrucciones/estilos.css

```
  body {
      margin-left: 10%;
      margin-right: 10%;
      background: beige;
  }

  p#inicio {
      padding-top: 1.5em;
  }

  fieldset {
      margin-bottom: 5%;
      background: white;
  }
```

html/instrucciones/centos.sh

```
  sudo yum install xrdp -y
  sudo firewall-cmd --permanent --add-port=3389/tcp
  sudo firewall-cmd -reload
```

html/instrucciones/ubuntu.sh

```
  sudo apt install xrdp -y
  sudo ufw allow 3389
```

html/instrucciones/windowspro.bat

```
  sc config RemoteRegistry start= auto
  net start remoteregistry
  reg add "HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Terminal
Server" /v fDenyTSConnections /t REG_DWORD /d 0 /f
  netsh firewall set service type = remotedesktop mode = enable
```

html/instrucciones/RDPW_installer.exe

Binario descargado del repositorio de github rdpwrap [43].

html/panel/index.html

```
  <!doctype html>
  <html>
    <head>
      <meta http-equiv="Content-Type" charset="UTF-8" />
      <title>Panel de administración</title>
      <script type="text/javascript" src="panel.js"></script>
      <link rel="stylesheet" href="panel.css">
    </head>

    <body>
      <h1>Panel de administración</h1>

      <fieldset>
        <legend>Registrar usuarios</legend>
        <p>
          Usuarios:
        </p>
        <p>
          <textarea type="text" name="usuarios" rows="10"
cols="40"></textarea><br/>
        </p>
        <p>
          <input id="enviar_usuarios" type="submit" value="Enviar"/>
        </p>
      </fieldset>

    <fieldset>
        <legend>Descubrir dispositivos</legend>
        <p>
          Nmap:
        </p>
        <br/>
        <p id="nmap">
        </p>
        <br/>
        <p>
          <input id="enviar_dispositivos" type="submit" value="Enviar"/>
        </p>
    </fieldset>

    </body>
  </html>
```

html/panel/panel.css

```
body {
    margin-left: 10%;
    margin-right: 10%;
    background: beige;
}


h1 {
    margin-top: 1em;
    margin-bottom: 5%;
}

fieldset {
    margin-bottom: 5%;
    background: white;
}
```

html/panel/panel.js

```
"use strict";

window.addEventListener('load', anadirEventos, false);

function anadirEventos() {
    carga_usuarios();
    document.getElementById("enviar_dispositivos").addEventListener('click',
carga_dispositivos, false);
    document.getElementById("enviar_usuarios").addEventListener('click',
enviar_usuarios, false);

document.getElementById("enviar_conexion_todas").addEventListener('click',
enviar_conexion_todas, false);
}

function carga_dispositivos() {
    let request = new XMLHttpRequest();
    const url = "/cgi-bin/nmap.cgi"

    let img = document.createElement("img");
    img.src="cargando.gif";
    img.alt="Cargando...";

    document.getElementById("nmap").appendChild(img);

    request.open("get", url);
    request.send();

    request.onload = function () {
      document.getElementById("nmap").innerHTML = this.responseText;
      let botones = document.getElementsByClassName("enviar_conexion");

      for(let i=0; i<botones.length; i++) {
        botones[i].addEventListener('click', enviar_conexion, false);
      }

    }
}

function carga_usuarios() {
```

```
      let request = new XMLHttpRequest();
      const url = "/cgi-bin/leer.cgi"
      request.open("get", url);
      request.send();

      request.onload = function () {
        document.getElementsByName("usuarios")[0].value = this.responseText;
      }
  }


  function enviar_usuarios (formulario) {
      if(validateUsers(document.getElementsByName("usuarios")[0].value) ==
true) {

        let request = new XMLHttpRequest();
        const url = "/cgi-bin/recibir.cgi"
        request.open("post", url);

        let string_split =
document.getElementsByName("usuarios")[0].value.split('\n');
        let string = "'" + string_split.join("','") + "'";

        request.send(string);
        request.onload = function () {
          if (this.status == 200) {
            alert("Guardado correctamente");
          } else {
            alert("Error");
          }
        }
      } else {
        alert("Has introducido algún carácter no permitido\nPermitidos
alfanuméricos y -");
      }
  }


  function validateUsers(str) {
    let code, i, len;

    for (i = 0, len = str.length; i < len; i++) {
      code = str.charCodeAt(i);
      if (!(code > 47 && code < 58) && // (0-9)
          !(code > 64 && code < 91) && // (A-Z)
          !(code > 96 && code < 123) && // (a-z)
          !(code == 32) && // " " espacio
          !(code == 10) && // "\n" salto de linea
          !(code == 45) && // "-" guion
          !(code == 209) && // Ñ
          !(code == 241)) { // ñ
        return false;
      }
    }
    return true;
  };


  function enviar_conexion(boton) {
    let connection_ip =
boton.srcElement.parentElement.childNodes[1].textContent;
    let mac = boton.srcElement.parentElement.childNodes[3].textContent;
```

```
    let connection_name = boton.srcElement.parentElement.childNodes[5].value;
    let user_create = boton.srcElement.parentElement.childNodes[6].value;
    let password_create = boton.srcElement.parentElement.childNodes[7].value;

    let json = {"user_create": user_create, "password_create":
password_create, "connection_name": connection_name, "connection_ip":
connection_ip, "mac": mac};
    console.log(connection_name.length);
    if(connection_name.length > 4 && user_create.length > 4 &&
password_create.length > 4) {
      let request = new XMLHttpRequest();
      const url = "/cgi-bin/crear.cgi"
      request.open("post", url);

      request.setRequestHeader("Content-Type", "application/json;charset=UTF-
8");
      request.send(JSON.stringify(json));
      request.onload = function () {
        if (this.status == 200) {
          alert("Creado correctamente");
        } else {
          alert(this.responseText);
        }
      }
    } else {
      alert("-Debe rellenar todos los campos\n-Todos los campos deben tener al
menos 4 caracteres");
    }
  }

  function enviar_conexion_todas(boton) {
    let user_create = document.getElementById("usuario_todas").value;
    let password_create = document.getElementById("password_todas").value;

    let json = {"user_create": user_create, "password_create":
password_create}
    if(user_create.length > 4 && password_create.length > 4) {
      let request = new XMLHttpRequest();
      const url = "/cgi-bin/crear_todas.cgi"
      request.open("post", url);

      request.setRequestHeader("Content-Type", "application/json;charset=UTF-
8");
      request.send(JSON.stringify(json));
      request.onload = function () {
        if (this.status == 200) {
          alert("Creado correctamente");
        } else {
          alert(this.responseText);
        }
      }
    } else {
      alert("-Debe rellenar todos los campos\n-Todos los campos deben tener al
menos 4 caracteres");
    }
  }
```

rentryco/postIp.sh

```
  #/bin/bash

  OLD=$(cat /usr/local/bin/rentryco/ip)
  IP=$(ip route get 8.8.8.8 | head -1 | cut -d' ' -f7)

  #Si la ip ha cambiado y tiene más de 7 caracteres
  if [[ $OLD != $IP  ]]  && (( ${#IP} > 7)); then
  echo -n "Antigua IP local:"
  echo $OLD
  echo -n "Nueva IP local:"
  echo $IP

  sed -i "s/${OLD}/${IP}/" /etc/apache2/sites-enabled/panel.conf
  sed -i "s/${OLD}/${IP}/" /etc/apache2/sites-enabled/000-default.conf
  openssl req -x509 -nodes -days 9999 -newkey rsa:2048 -subj "/CN=${IP}/C=ES"
-keyout /etc/ssl/private/apache-selfsigned.key -out /etc/ssl/certs/apache-
selfsigned.crt

  service apache2 reload

  /usr/local/bin/rentryco/rentry edit -u tfgenrique -p enrsancar1
"https://${IP}/"

  echo $IP > /usr/local/bin/rentryco/ip

  else
  echo "La ip local no ha cambiado"

  fi
```

terraform-apply-startup.service

```
  [Unit]
  Description=Terraform apply on startup
  Wants=network-online.target
  After=network-online.target

  [Service]
  Type=simple
  ExecStart=/bin/bash /usr/bin/terraform-apply-startup.sh

  [Install]
  WantedBy=multi-user.target
```

config

```
  #!/bin/bash
  ####################################################################
  #Configuracion

  #Panel de administracion
  USERNAME="enrsancar1"
  PASSWORD="enrsancar1"

  #Contraseña usuario guacadmin (administrador)
  PASSWORD_GUAC="enrsancar1"

  #DuckDNS
```

```
  DOMAIN="enrsancar1.duckdns.org"
  DUCKDNS_TOKEN="83bd62d1-d317-4991-a8dc-689cf8d5897f"

  #AWS keys para Terraform
  AWS_SECRET_ACCESS_KEY="U2RMzehDPk8K9r2DdRRnSK9KDbiQenhws0duvp1n"
  AWS_ACCESS_KEY_ID="AKIAUSF6T5HUFZHNU3PM"

  #Horario para apply y destroy de Terraform
  #ejemplo sintaxis todos los lunes a las 07:00 -> "0 7 * * mon"
  #ejemplo sintaxis todos los domingos a las 00:00 -> "0 0 * * sun"
  ENCENDIDO="0 7 * * mon"
  APAGADO="0 0 * * sun"

  #Personalizacion web de inicio de sesión
  #Para añadir logo personalizado reemplazar
  #png en guacamole-branding-example/resources/images/logo/logo.png
  EMPRESA="Teleco US"

  #Email para certbot
  EMAIL="pdaenrique2@gmail.com"

  #Rentry.co para saber la iplocal
  ENDPOINT="tfgenrique"
  CODE="enrsancar1"

  #Servidor para obtener la hora
  NTPSERVER="hora.roa.es"

  ########################################################################
```

install.sh

```
  #!/bin/bash
  ########################################################################
  #Configuracion
  source ./config
  ########################################################################
  IP_LOCAL=$(ip route get 8.8.8.8 | head -1 | cut -d' ' -f7)

  #Instalo terraform usando el binario v1.1.9
  wget
https://releases.hashicorp.com/terraform/1.1.9/terraform_1.1.9_linux_arm64.zi
p
  unzip terraform_1.1.9_linux_arm64.zip
  cp terraform /usr/local/bin/

  #Añadimos keys amazon ec2 al fichero credentials
  mkdir /root/.aws
  echo "export AWS_SECRET_ACCESS_KEY=${AWS_SECRET_ACCESS_KEY}" >>
/root/.aws/credentials
  echo "export AWS_ACCESS_KEY_ID=${AWS_ACCESS_KEY_ID}" >>
/root/.aws/credentials
  chmod 400 /root/.aws/credentials
  source /root/.aws/credentials

  #Hacemos el terraform apply para activar el tunel y ya luego poder pedir el
certificado
  cp -r cron-ssh-tunnel /root/
  chmod 704 /root/cron-ssh-tunnel/scripts/tunnel.sh
  cd /root/cron-ssh-tunnel
  sed -i "s/DOMAIN/${DOMAIN}/" variables.tf
```

```
  sed -i "s/DUCKDNS_TOKEN/${DUCKDNS_TOKEN}/" variables.tf
  terraform init
  terraform apply --auto-approve
  cd /root/tfg

  #Ponemos tarea en cron para que haga terraform apply todos los lunes a las
07:00 (ejemplo)
  #Ponemos tarea en cron para que haga terrafrom destroy todos los domingos a
las 00:00 (ejemplo)
  sudo echo -e
"SHELL=/bin/bash\nPATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/loca
l/sbin\n\n${ENCENDIDO} root source /root/.aws/credentials; cd /root/cron-ssh-
tunnel; terraform apply --auto-approve > /dev/null 2> /root/cron-ssh-
tunnel/log/apply-destroy-error.log" > /etc/cron.d/terraform-apply
  sudo echo -e
"SHELL=/bin/bash\nPATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/loca
l/sbin\n\n${APAGADO} root source /root/.aws/credentials; cd /root/cron-ssh-
tunnel; terraform destroy --auto-approve > /dev/null 2> /root/cron-ssh-
tunnel/log/apply-destroy-error.log" > /etc/cron.d/terraform-destroy

  #Ponemos servicio en systemd para que en cada inicio, despues de cargar el
servicio de red, se haga ntpdate (sincronizar hora) y terraform apply
  sudo echo -e "ntpdate -b ${NTPSERVER}; source /root/.aws/credentials; cd
/root/cron-ssh-tunnel; terraform apply --auto-approve > /dev/null 2>
/root/cron-ssh-tunnel/log/apply-startup-error.log" > /usr/bin/terraform-
apply-startup.sh
  chmod +x /usr/bin/terraform-apply-startup.sh
  cp terraform-apply-startup.service /lib/systemd/system/terraform-apply-
startup.service
  cp terraform-apply-startup.service /etc/systemd/system/terraform-apply-
startup.service
  chmod 644 /etc/systemd/system/terraform-apply-startup.service
  systemctl enable terraform-apply-startup

  #Instalo docker
  curl -fsSL https://get.docker.com -o get-docker.sh
  sh ./get-docker.sh

  #oznu copyright - GNU License
  docker run -d -v /home/guacamole/oznu_config:/config --network host --
restart always --name guacamole oznu/guacamole:armhf #armhf arquitectura arm

  #Instalo apache2
  apt-get install apache2 -y

  #activo mod_proxy y mod_proxy_wstunnel
  a2enmod proxy
  a2enmod proxy_http
  a2enmod proxy_wstunnel

  #Let's Encrypt
  snap install core
  snap refresh core
  snap install --classic certbot
  ln -s /snap/bin/certbot /usr/bin/certbot
  certbot --apache -d $DOMAIN --non-interactive --agree-tos --email ${EMAIL}

  #Esto tambien nos configura un timer para renovar automaticamente el
certificado
  #/etc/systemd/system/snap.certbot.renew.timer

  #Copio el certificado de la fnmt a la carpeta ca
```

```
  #fnmt obtenido a partir de la union de AC_FNMT_Usuarios.pem y AC_Raiz_FNMT-
RCM_SHA256.pem (descargados desde la pag oficial de la FNMT)
  mkdir /etc/apache2/ca
  cp ca/fnmt/fnmt.pem /etc/apache2/ca/fnmt.pem

  #Activo  y la autenticacion por certificado del cliente en la configuracion
de vHost:443
  #Linea donde se encuentra el cierre </VirtualHost>
  L_VHOSTSSL=$(cat /etc/apache2/sites-enabled/000-default-le-ssl.conf | grep -
n "</VirtualHost>" | cut -d ":" -f 1)
  #Ultima linea del fichero
  L_FINSSL=$(cat /etc/apache2/sites-enabled/000-default-le-ssl.conf | wc -l)

  sed "${L_VHOSTSSL},${L_FINSSL}d" -i /etc/apache2/sites-enabled/000-default-
le-ssl.conf
  echo -e "
  SSLVerifyClient require
  SSLVerifyDepth 2
  SSLCACertificateFile /etc/apache2/ca/fnmt.pem
  <Location />
      ProxyPass http://127.0.0.1:8080/ flushpackets=on
      ProxyPassReverse http://127.0.0.1:8080/
      Require expr \"(%{SSL_CLIENT_S_DN_CN} in {''})\"
      Options -Indexes
  </Location>
  <Location /websocket-tunnel>
      ProxyPass ws://127.0.0.1:8080/websocket-tunnel
      ProxyPassReverse ws://127.0.0.1:8080/websocket-tunnel
      Require expr \"(%{SSL_CLIENT_S_DN_CN} in {''})\"
      Options -Indexes
  </Location>
  <Location /instrucciones>
      ProxyPass !
      Require expr \"(%{SSL_CLIENT_S_DN_CN} in {''})\"
      Options -Indexes
  </Location>

  SetEnvIf Request_URI \"^/tunnel\" dontlog
  CustomLog  /var/log/apache2/guac.log common env=!dontlog

  </VirtualHost>
  </IfModule>" >> /etc/apache2/sites-enabled/000-default-le-ssl.conf

  #Creamos las instrucciones de instalacion
  cp -r html/instrucciones/ /var/www/html/instrucciones/

  #Activamos mod_auth_basic
  a2enmod auth_basic

  #Creo el archivo que almacena la contraseña del panel y cgi
  mkdir -p /usr/local/apache/passwd
  htpasswd -bc /usr/local/apache/passwd/passwords $USERNAME $PASSWORD

  #Creamos config para el panel de administracion
  #Creamos certificado autofirmado para ssl en IP_LOCAL
  openssl req -x509 -nodes -days 9999 -newkey rsa:2048 -subj
"/CN=${IP_LOCAL}/C=ES" -keyout /etc/ssl/private/apache-selfsigned.key -out
/etc/ssl/certs/apache-selfsigned.crt

  #Copiamos la plantilla
  cp apache_conf/panel.conf /etc/apache2/sites-enabled/
```

```
  sed -i "s/IP_LOCAL/${IP_LOCAL}/" /etc/apache2/sites-enabled/panel.conf
#Sustituimos IP_LOCAL de la plantilla por la ip local de la maquina

  #Activamos el Rewrite para IP LOCAL
  sed -i "s/<\/VirtualHost>/ RewriteEngine on \n\
  RewriteCond %{SERVER_NAME} =${IP_LOCAL} \n\
  RewriteRule ^ https:\/\/%{SERVER_NAME}%{REQUEST_URI} [END,NE,R=permanent]
\n\
  <\/VirtualHost>/" /etc/apache2/sites-enabled/000-default.conf

  #Restart a apache2 para que se conecte a tomcat
  #service apache2 restart

  #Instalamos nmap y xmllint
  apt-get install nmap -y
  apt install libxml2-utils -y

  #Instalamos las dependencias que usan los scripts de python3
  apt install python3 python3-pip
  pip3 install psycopg2

  #Activamos cgi
  a2enmod cgi
  cp apache_conf/serve-cgi-bin.conf /etc/apache2/conf-enabled/

  cp cgi-bin/leer.cgi /usr/lib/cgi-bin/
  cp cgi-bin/recibir.cgi /usr/lib/cgi-bin/
  cp cgi-bin/nmap.cgi /usr/lib/cgi-bin/
  cp cgi-bin/sincronizar_usuarios.sh /etc/apache2/

  chmod 705 /usr/lib/cgi-bin/leer.cgi
  chmod 705 /usr/lib/cgi-bin/recibir.cgi
  chmod 705 /usr/lib/cgi-bin/nmap.cgi
  chmod 700 /etc/apache2/sincronizar_usuarios.sh

  #Permisos necesarios para modificar certificados aceptados
  chown root:www-data /etc/apache2/sites-enabled/000-default-le-ssl.conf
  chmod 660 /etc/apache2/sites-enabled/000-default-le-ssl.conf
  mkdir /home/www-data
  chown www-data:www-data /home/www-data

  #Modificamos el archivo sudoers para poder ejecutar sincronizar_usuarios
como root ya que necesita hacer reload de apache2
  echo "www-data                ALL=(ALL) NOPASSWD:
/etc/apache2/sincronizar_usuarios.sh" >> /etc/sudoers

  #Creamos el panel de administracion
  cp -r html/panel/ /var/www/panel

  #Añado el branding personalizado
  apt install zip
  docker stop guacamole
  cd guacamole-branding-example
  sed -i "s/EMPRESA/${EMPRESA}/" translations/en.json
  zip -r guacamole-branding-example.jar *
  cd ..
  mv guacamole-branding-example/guacamole-branding-example.jar
/home/guacamole/oznu_config/guacamole/extensions/
  docker start guacamole

  #Restart para que apache use la nueva configuracion y se conecte a tomcat
  service apache2 restart
```

```
  #Instalar python3
  apt install python3
  #Rentry.co para saber la ip local
  cp -r rentryco/ /usr/local/bin/
  sed -i "s/ENDPOINT/${ENDPOINT}/" /usr/local/bin/rentryco/postIp.sh
  sed -i "s/CODE/${CODE}/" /usr/local/bin/rentryco/postIp.sh
  chmod 744 /usr/local/bin/rentryco/rentry
  chmod 744 /usr/local/bin/rentryco/postIp.sh
  bash /usr/local/bin/rentryco/postIp.sh
  echo -e
"SHELL=/bin/bash\nPATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/loca
l/sbin\n\n* * * * * root bash /usr/local/bin/rentryco/postIp.sh" >>
/etc/cron.d/iplocal

  #IPTABLES permite los servicios que necesitamos unicamente
  iptables -A INPUT -p tcp ! -i lo --dport 8080 -j DROP
  iptables -A INPUT -p tcp ! -i lo --dport 8009 -j DROP
  iptables -A INPUT -p tcp ! -i lo --dport 5432 -j DROP
  iptables -A INPUT -p tcp ! -i lo --dport 4822 -j DROP
  iptables -A INPUT -p all -i lo -j ACCEPT
  iptables -A INPUT -p tcp --dport 22 -j ACCEPT
  iptables -A INPUT -p tcp --sport 22 -j ACCEPT
  iptables -A INPUT -p tcp --dport 80 -j ACCEPT
  iptables -A INPUT -p tcp --dport 443 -j ACCEPT
  iptables -A INPUT -p tcp --sport 3389 -j ACCEPT
  iptables -A INPUT -p tcp --sport 80 -j ACCEPT
  iptables -A INPUT -p tcp --sport 443 -j ACCEPT
  iptables -A INPUT -p udp --sport 53 -j ACCEPT
  iptables -A INPUT -p udp --sport 123 -j ACCEPT
  iptables -P INPUT DROP

  #Instalo iptables-persistent para que cargue en el proximo inicio las reglas
iptables automaticamente
  apt-get install -y iptables-persistent
  iptables-save > /etc/iptables/rules.v4

  #Cambio la contraseña al usuario guacadmin
  /usr/bin/python3 cambiar_guacadmin.py $PASSWORD_GUAC
```

uninstall.sh

```
  #!/bin/bash
  #Elimino las reglas iptables
  iptables -P INPUT ACCEPT
  iptables -F INPUT

  #Elimino credenciales AWS
  source /root/.aws/credentials
  rm /root/.aws/credentials

  #Paro los servicios
  service apache2 stop
  docker stop guacamole
  docker rm guacamole
  cd /root/cron-ssh-tunnel
  terraform destroy --auto-approve

  #Desinstalar apache2
  apt-get purge apache2 apache2-bin apache2-data apache2-utils -y
  rm -rf /etc/apache2
```

```
  rm -rf /var/lib/apache2
  rm -rf /usr/lib/apache2

  #Eliminar archivo contraseñas
  rm /usr/local/apache/passwd/passwords

  #Eliminar cert autofirmado para panel
  rm /etc/ssl/certs/apache-selfsigned.crt
  rm /etc/ssl/private/apache-selfsigned.key

  #Desinstalar nmap y xmllint
  apt purge nmap -y
  apt purge libxml2-utils -y

  #Desinstalar pip3 y psycopg2
  pip3 uninstall psycopg2 -y
  apt purge python3-pip -y

  #Borrar scripts cgi
  rm /usr/lib/cgi-bin/leer.cgi
  rm /usr/lib/cgi-bin/recibir.cgi
  rm /usr/lib/cgi-bin/nmap.cgi
  rm /usr/lib/cgi-bin/crear.cgi
  rm /usr/lib/cgi-bin/crear_todas.cgi

  #Borro la linea del /etc/sudoers
  sed -i "s/.*sincronizar_usuarios\.sh//" /etc/sudoers

  #Borrar html
  rm -rf /var/www/html/instrucciones
  rm -rf /var/www/panel

  #Desinstalar docker
  apt-get purge docker-ce docker-ce-cli containerd.io docker-ce-rootless-
extras -y
  rm -rf /var/lib/docker
  rm -rf /var/lib/containerd
  rm -rf /etc/docker
  rm -rf /var/run/docker.sock
  groupdel docker

  #Borro la configuracion del contenedor
  rm -rf /home/guacamole

  #Desinstalar certbot
  certbot delete
  snap remove certbot
  apt purge python-certbot-apache
  apt-get purge certbot
  rm /usr/bin/certbot
  rm -rf /etc/letsencrypt/
  rm -rf /var/lib/letsencrypt/
  rm -rf /var/log/letsencrypt/
  systemctl disable snap.certbot.renew.timer
  systemctl stop snap.certbot.renew.timer
  rm /etc/systemd/system/snap.certbot.renew.timer
  rm /etc/systemd/system/snap.certbot.renew.service

  #Desinstalar terraform
  rm /usr/local/bin/terraform
  rm /etc/cron.d/terraform-apply
  rm /etc/cron.d/terraform-destroy
```

```
systemctl stop terraform-apply-startup
systemctl disable terraform-apply-startup
rm /usr/bin/terraform-apply-startup
rm /lib/systemd/system/terraform-apply-startup.service
rm /etc/systemd/system/terraform-apply-startup.service

#Elimino rentryco
rm -rf /usr/local/bin/rentryco
rm /etc/cron.d/iplocal

apt autoremove -y
apt autoclean
```