FACULTAD DE CIENCIAS EXACTAS, FÍSICAS y NATURALES

INFORME DE PROYECTO INTEGRADOR

Modelo de seguridad multinivel para servidores virtuales con infraestructura de clave pública y software libre

Realizado por

Germán A. Lamberti Pecile
german.lamberti@unc.edu.ar / 3564651828

Saúl Muñoz
saul.munoz@mi.unc.edu.ar / 3512474305

Para la obtención del título de
Grado en Ingeniería en Computación

Nombre del Director
Mgter. Ing. Miguel Solinas

Nombre del Codirector
Esp. Ing. Javier Alejandro Jorge

Agradecimientos

Queremos expresar nuestra infinita gratitud a nuestros padres, quienes, con su amor, su apoyo y su sacrificio, han sido el pilar fundamental de nuestra formación.

Asimismo, agradecemos a nuestros hermanos, abuelos, pareja, familiares y amigos por su compañía y su incansable apoyo en esta travesía, así como por ser parte esencial de nuestra vida.

Extendemos nuestro agradecimiento al director del proyecto integrador, Mgter. Ing. Miguel Solinas, y al codirector, Esp. Ing. Javier Jorge, por su guía, sus conocimientos y su permanente disposición para ayudar.

Del mismo modo, expresamos nuestro agradecimiento a la Universidad Nacional de Córdoba, que no solo nos brindó conocimientos técnicos de alto nivel, sino también valores fundamentales como el compromiso, la responsabilidad y el pensamiento crítico.

Finalmente, manifestamos nuestro reconocimiento y gratitud a toda la comunidad universitaria que, directa o indirectamente, formó parte de este camino.

Con gratitud,

Germán Lamberti P.
Saúl Muñoz

Resumen del trabajo efectuado

Este proyecto integrador aborda, como tema principal, la autorización y auditoría, propone un modelo de seguridad de acceso multinivel para un sistema distribuido basado en una infraestructura de clave pública.

La autenticación de usuarios mediante SSH hacia servidores virtuales se integró a través del uso de certificados digitales, emitidos de forma centralizada mediante el framework EJBCA, conforme al estándar X.509.

Cada certificado emitido a un usuario contiene roles específicos, sobre dichos roles se definen reglas que determinan las operaciones permitidas.

Con respecto al registro y a la trazabilidad de las acciones dentro del sistema, se implementó en los servidores virtuales de la infraestructura una configuración que permite identificar las operaciones ejecutadas por cada usuario. Estos eventos se centralizan y se ponen a disposición para su observación y monitoreo, mediante Elasticseach, Kibana y Filebeat.

Palabras clave: Criptografía, Auteticación, Autorización, Auditoria, Monitoreo, PKI, RBAC

Índice general

1. Introducción **1**
- 1.1. Descripción del Problema . . . 1
- 1.2. Antecedentes . . . 1
- 1.3. Objetivos . . . 2
  - 1.3.1. Objetivo General . . . 2
  - 1.3.2. Objetivo Especificos . . . 2
- 1.4. Motivación . . . 2

2. Marco Teórico **4**
- 2.1. Criptografía . . . 4
- 2.2. Autenticación . . . 5
  - 2.2.1. Métodos de autenticación en servidores . . . 5
- 2.3. Autorización . . . 6
  - 2.3.1. Política de control de acceso . . . 7
- 2.4. Auditoría . . . 7
  - 2.4.1. Monitoreo . . . 7
- 2.5. Clave pública-privada . . . 7
- 2.6. Certificados digitales X.509 . . . 8
- 2.7. Infraestructura de Clave Pública (PKI) . . . 8
  - 2.7.1. Componentes . . . 9
  - 2.7.2. Entidad final . . . 11
  - 2.7.3. Perfil de certificado . . . 11
  - 2.7.4. Descripción general de los protocolos remotos . . . 12
- 2.8. PKCS11 . . . 13
  - 2.8.1. Opciones de implementación . . . 13
- 2.9. Protocolo SSH . . . 15
  - 2.9.1. OpenSSH . . . 16
- 2.10. PAM . . . 16
- 2.11. Ansible . . . 17
- 2.12. Elasticsearch . . . 17
- 2.13. Kibana . . . 18
- 2.14. Beats . . . 19
- 2.15. Virtualización . . . 20
- 2.16. Docker . . . 20
  - 2.16.1. Docker Compose . . . 21
- 2.17. API . . . 21
  - 2.17.1. API REST . . . 22
- 2.18. EJBCA . . . 23
  - 2.18.1. Tokens criptográficos . . . 23

3. Descripción de requerimientos **25**
  - 3.0.1. Stakeholders . . . 25

3.1. Características y Requerimientos del Sistema . . . 25
3.1.1. Requerimientos Funcionales . . . 25
3.1.2. Requerimientos No Funcionales . . . 26
3.2. Análisis de Riesgos . . . 27
3.3. Plan de Trabajo . . . 27

4. Diseño **29**
4.1. Diseño a nivel de sistemas . . . 29
4.1.1. Modelos Estáticos . . . 30
4.1.2. Modelos Dinámicos . . . 33
4.2. Topología de Red . . . 43
4.3. Política de roles del sistema . . . 45

5. Selección de Componentes **48**
5.1. Hardware . . . 48
5.1.1. Servidor físico principal . . . 48
5.2. Software . . . 49
5.2.1. Plataforma de virtualización . . . 49
5.2.2. Sistema operativo base . . . 49
5.2.3. Infraestructura PKI: EJBCA . . . 49
5.2.4. Autenticación para el acceso a servidores . . . 50
5.2.5. Autorización multinivel y su propagación: Ansible . . . 50
5.2.6. Auditoria y monitoreo: Elastic Stack . . . 50
5.2.7. Otras herramientas complementarias . . . 51

6. Actividades Desarrolladas **52**
6.1. Iteración I: Configuración de máquinas virtuales en el servidor . . . 52
6.1.1. Instalación de VMware ESXi . . . 52
6.1.2. Creación de máquinas virtuales . . . 54
6.1.3. Configuración de VLAN . . . 56
6.1.4. Configuraciones no implementadas . . . 57
6.2. Iteración II: Despliegue y configuración de la Infraestructura de Clave Pública . . . 57
6.3. Iteración III: Despliegue y configuración del sistema de Autenticación . . . 70
6.3.1. Despliegue del servicio de autenticación . . . 70
6.3.2. Configuraciones esenciales de PAM en Gateway y Hosts . . . 72
6.3.3. Configuraciones no implementadas . . . 77
6.4. Iteración IV: Implementación del sistema de Autorización . . . 77
6.4.1. Instalación de Ansible . . . 78
6.4.2. Gestión de roles en gateway y hosts . . . 79
6.4.3. Despliegue de reglas vía Ansible . . . 89
6.4.4. Desarrollo de la capa de Autorización en auth-server . . . 92
6.4.5. Prueba de integración del sistema RBAC . . . 94
6.5. Iteración V: Implementación del sistema de auditoria y monitoreo . . . 98
6.5.1. Captura de comandos . . . 98
6.5.2. Configuración y puesta en marcha . . . 100

7. Conclusiones **107**

7.1. Evaluación de resultados 107
7.2. Desafíos y Soluciones Específicas 107
7.3. Reflexión Final 108

8. Perspectivas Futuras **109**

9. Glosario **111**

10. Bibliografía **115**

Índice de figuras

4.1. Diagrama de Definición de Bloques del sistema 30
4.2. Diagrama Interno de Bloques del sistema 32
4.3. Diagrama de caso de uso del sistema de acceso multinivel 34
4.4. Diagrama de Secuencia de Configuración de la PKI en EJBCA 36
4.5. Diagrama de secuencia de alta de usuario técnico 37
4.6. Diagrama de secuencia de acceso de usuario técnico 39
4.7. Diagrama de secuencia para la configuración de nuevos usuarios y roles 41
4.8. Diagrama de secuencia del almacenamiento y visualización de logs . . . 42
4.9. Topología lógica de red del sistema. 44

5.1. Características del servidor Dell PowerEdge R210 II 49

6.1. Panel de configuración del sistema del servidor 53
6.2. Opciones de configuración en VMware 53
6.3. Inicio de sesión de VMware 54
6.4. Almacenamiento de la ISO de un sistema operativo 54
6.5. Pantalla de configuración de recursos de un servidor virtual 55
6.6. Configuración del nombre del servidor y del usuario de sistema 55
6.7. Creación de las máquinas virtuales 56
6.8. Port Group configurado con VLAN ID 10 56
6.9. Repositorio del proyecto en GitHub 58
6.10. Contenedores desplegados por Docker Compose 59
6.11. Protocolos habilitados en EJBCA 60
6.12. Configuración del token criptográfico en EJBCA 61
6.13. Valores iniciales de la Autoridad de Certificación PSI-CA 62
6.14. Certificado de la Autoridad de Certificación PSI-CA 64
6.15. Parámetros de configuración de la CRL 64
6.16. Configuración de puntos de validación de certificados 65
6.17. Configuración del perfil de certificado ROOTCA 66
6.18. Perfil de certificado ENDUSER 67
6.19. Perfil de certificado SERVER_MTLS 68
6.20. Listado de perfiles de entidades finales en EJBCA 68
6.21. Perfil de entidad final Personal PSI 68
6.22. Formulario de solicitud de emisión de certificado 69
6.23. Opciones de descarga del certificado emitido 70
6.24. Repositorio del RBAC Module 71
6.25. Contenedores desplegados por Docker Compose 71
6.26. Clave pública de ARCenter en Destino 2 79
6.27. Estructura de carpetas y archivos para creación de usuarios 81
6.28. Estructura de carpetas y archivos para modificación de roles 86
6.29. Estructura de carpetas y archivos para eliminación de roles 88
6.30. Resultado de la ejecución del playbook de creación de usuarios 90
6.31. Resultado de la ejecución del playbook de modificación de reglas 91

6.32. Resultado de la ejecución del playbook de eliminación de usuarios . . . 91
6.33. Certificado digital con roles 92
6.34. Conexión SSH exitosa al servidor Gateway como `sysadmin` 96
6.35. Conexión SSH exitosa al servidor destino1 como `databases` vía Gateway 97
6.36. Conexión SSH exitosa al servidor destino2 como `devops` vía Gateway . 98
6.37. dashboad de monitoreo de accesos 105
6.38. dashboad de comandos ejecutados 106

Índice de extractos de código

6.1. Ruta destino de certificados cliente . . . 73
6.2. Extracción del certificado desde un archivo .p12 . . . 73
6.3. Extracción de la clave privada desde un archivo .p12 . . . 74
6.4. Permisos mínimos recomendados . . . 74
6.5. Modulo personalizado PAM Python con escritura de clave pública . . . 74
6.6. Instalación de Ansible en ARCenter . . . 78
6.7. Generación de clave SSH en ARCenter . . . 78
6.8. Creación del usuario ansibleadmin en los servidores destino . . . 78
6.9. Regla sudoers para ansibleadmin . . . 79
6.10. Exclusión de ansibleadmin del control de PAM . . . 79
6.11. Definición de regla en el archivo sudoers . . . 80
6.12. Regla para el rol databases . . . 80
6.13. Regla para el rol sysadmin . . . 80
6.14. Regla para el rol devops . . . 81
6.15. Playbook para crear usuarios . . . 82
6.16. Playbook que crea un usuario . . . 82
6.17. Archivo de inventario Ansible . . . 84
6.18. Archivo `bastion.yml` . . . 84
6.19. Archivo `destino1.yml` . . . 84
6.20. Archivo `destino2.yml` . . . 84
6.21. Archivo de configuración de reglas de permisos . . . 85
6.22. Playbook principal de actualizacion de reglas sudo . . . 86
6.23. Tareas individuales para cada usuario . . . 86
6.24. Ejemplo de reglas sudo definidas por rol . . . 87
6.25. Archivo `bastion.yml` . . . 87
6.26. Archivo `destino1.yml` . . . 87
6.27. Archivo `destino2.yml` . . . 87
6.28. Playbook principal de eliminación de usuarios . . . 88
6.29. Tareas individuales para eliminación de usuario . . . 88
6.30. Archivo de bastion.yml . . . 89
6.31. Archivo de destino1.yml . . . 89
6.32. Archivo de destino2.yml . . . 89
6.33. Ejecución del playbook de creación de usuarios . . . 90
6.34. Ejecución del playbook de actualización de sudoers . . . 90
6.35. Ejecución del playbook de eliminación de usuarios . . . 91
6.36. Implementación completa de la capa de autorización en auth-server . . . 93
6.37. Extracción de clave privada . . . 95
6.38. Extracción de certificado digital . . . 95
6.39. Obtención del serial del certificado . . . 95
6.40. Acceso directo al servidor Gateway . . . 95
6.41. Acceso SSH a destino1 a través del Gateway . . . 96
6.42. Acceso SSH a destino2 a través del Gateway . . . 97
6.43. Captura de comando . . . 99

6.44. Achivo de configuración de filebeat 103

1. Introducción

1.1. Descripción del Problema

La creciente adopción de infraestructuras virtualizadas y servicios en la nube en entornos institucionales ha generado nuevos desafíos en materia de seguridad informática, especialmente en lo que respecta al control de acceso y la gestión de identidades digitales. En particular, la virtualización de servidores permite la coexistencia de múltiples máquinas virtuales (VM) con diferentes niveles de acceso y requerimientos de seguridad, lo que plantea la necesidad de implementar modelos de control de acceso más rigurosos.

En la actualidad, instituciones como la Prosecretaría de Informática de la Universidad Nacional de Córdoba utiliza mecanismos basados en criptografía de clave pública para controlar el acceso a sus servidores físicos y virtuales. Sin embargo, este sistema carece de una infraestructura centralizada para la emisión, verificación y revocación de certificados digitales, y no contempla un control formal de roles ni de responsabilidades en la gestión de identidades. Esta situación limita la escalabilidad, la trazabilidad y la solidez de la seguridad del entorno, especialmente ante escenarios de crecimiento o integración con servicios externos.

La falta de un modelo centralizado de infraestructura de clave pública (PKI), acompañado de un sistema de control de acceso multinivel, puede representar un punto débil en la arquitectura de seguridad de la institución. Frente a este escenario, se plantea la necesidad de implementar un modelo que integre control de acceso obligatorio con mecanismos seguros de autenticación, autorización y registro, y que contemple la incorporación de componentes confiables como Autoridades de Certificación (AC) y Autoridades de Registro (AR), dentro de un marco normativo estandarizado (X.509).

1.2. Antecedentes

El framework de código abierto Enterprise JavaBeans Certificate Authority (EJBCA) se ha consolidado durante más de quince años como una solución robusta, escalable y confiable para implementar PKI conforme al estándar X.509. Además, en los últimos años ha incorporado herramientas como Soft HSM y una API RESTful, que facilitan su integración con otros servicios y permiten automatizar procesos críticos.

1.3. Objetivos

1.3.1. Objetivo General

Implementar un modelo de seguridad para control de acceso a servidores virtuales que incluya autorización y registro con una infraestructura de clave pública y herramientas open source.

1.3.2. Objetivo Especificos

- Investigar los distintos modelos de seguridad aplicables a servidores, con especial énfasis en aquellos utilizados en la Prosecretaría de Informática de la Universidad Nacional de Córdoba.
- Diseñar una propuesta de arquitectura que contemple mecanismos de autenticación, control de acceso y registro de eventos.
- Incorporar los requerimientos de autorización de servidores a las políticas de seguridad existentes para la emisión de certificados digitales a entidades finales y autoridades de registro.
- Implementar las políticas de control de acceso definidas en un modelo a escala.
- Verificar el comportamiento del sistema implementado y realizar los ajustes necesarios.
- Escalar la solución al modelo de pre-producción.
- Publicar resultados.
- Redactar informe final y documentación para usuarios.

1.4. Motivación

La transformación digital de los entornos universitarios y la creciente dependencia de servicios informáticos virtualizados hacen indispensable contar con soluciones de seguridad que garanticen la confidencialidad, integridad y disponibilidad de los sistemas. En este contexto, la posibilidad de implementar una infraestructura de clave pública centralizada y moderna representa una mejora sustancial respecto a los mecanismos actualmente en uso.

Además, este proyecto tiene un impacto directo y tangible en la seguridad de los servicios administrados por la Prosecretaría de Informática, permitiendo establecer mecanismos más sólidos y transparentes para la autenticación de usuarios y el control de acceso a servidores. La incorporación de roles claros como la Autoridad de Certificación (AC) y la Autoridad de Registro (AR), junto con una política formal de gestión de certificados, permitirá fortalecer los procesos institucionales y mejorar la trazabilidad y gobernanza de la infraestructura informática.

Finalmente, la experiencia adquirida en este trabajo podrá ser extrapolada a otros entornos similares, y constituirá un valioso aporte para futuras iniciativas de seguridad en instituciones académicas y gubernamentales.

2. Marco Teórico

La creciente necesidad de controlar de forma segura el acceso a sistemas críticos ha impulsado el desarrollo y la adopción de modelos de autenticación, autorización y registro robustos, especialmente en entornos donde la virtualización y el acceso remoto son predominantes. Este trabajo se inscribe dentro de esa necesidad, abordando la implementación de un modelo de seguridad multinivel basado en Infraestructura de Clave Pública (PKI), orientado al control centralizado y seguro de accesos a servidores físicos y virtuales.

2.1. Criptografía

La criptografía es la ciencia y práctica de proteger información mediante el uso de algoritmos matemáticos que garantizan la seguridad de los datos en distintas situaciones: en reposo (almacenados), en tránsito (durante la transmisión) o en uso (durante el procesamiento) [1].

Los objetivos principales de la criptografía son:

Confidencialidad: garantizar que solo los usuarios autorizados accedan a la información.

Integridad: asegurar que los datos no han sido alterados.

Autenticación: verificar la identidad de los usuarios o la validez de los mensajes.

No repudio: impedir que alguien niegue haber realizado una acción o enviado un mensaje [1].

Para cumplir estos objetivos, se utilizan algoritmos criptográficos de bajo nivel, como:

- Algoritmos de cifrado (simétricos y asimétricos)
- Algoritmos de firma digital
- Funciones hash

Existen dos tipos principales de criptografía:

- Criptografía simétrica: usa una sola clave para cifrar y descifrar la información.
- Criptografía asimétrica (o de clave pública): utiliza un par de claves (pública y privada) distintas para cifrado y descifrado [1].

Una Infraestructura de Clave Pública (PKI) permite gestionar certificados digitales mediante criptografía asimétrica, proporcionando servicios de seguridad a gran escala, como en internet [2].

Hoy en día, la criptografía es fundamental para la seguridad digital: permite transacciones electrónicas seguras, protege la comunicación en línea (como HTTPS). Su evolución ha llevado a soluciones basadas en análisis matemáticos rigurosos, muy superiores a los métodos clásicos de encriptación [1].

2.2. Autenticación

La autenticación es el proceso mediante el cual un sistema verifica la identidad de un usuario, dispositivo o entidad que intenta acceder a un recurso. Este proceso busca asegurar que quien solicita acceso es realmente quien dice ser [3].

En lo que se refiere al proceso de identificación, los elementos utilizados para identificar a un usuario pueden basarse en:

- lo que sabe, por ejemplo contraseñas [4]
- lo que posee, por ejemplo una tarjeta inteligente, una llave usb
- lo que es, características biométricas del individuo, por ejemplo características faciales para su reconocimiento
- lo que sabe hacer, por ejemplo firma manuscrita

Tradicionalmente, la autenticación se ha basado en el uso de credenciales como nombres de usuario y contraseñas, sin embargo, en contextos más exigentes en términos de seguridad, se recurre a métodos como el uso de clave pública, certificados digitales, tokens criptográficos o autenticación multifactor [3].

2.2.1. Métodos de autenticación en servidores

Existen diversos métodos de autenticación para servidores, cada uno con características propias, que pueden ser seleccionados según los requerimientos de seguridad de la organización.

La implementación de un mecanismo de autenticación basado en contraseña en servidores requiere que el cliente proporcione la contraseña correspondiente al usuario del sistema al que intenta acceder. A pesar de su simplicidad, este enfoque requiere el cumplimiento de buenas prácticas como el uso de contraseñas complejas y políticas de renovación periódica para mantener su efectividad [4].

Otro método ampliamente utilizado es la autenticación mediante claves públicas y privadas, especialmente en entornos que utilizan el protocolo SSH. En este caso, el usuario posee una clave privada y el servidor contiene la correspondiente clave pública, permitiendo establecer una conexión segura sin necesidad de ingresar una contraseña en cada acceso. Este método no solo mejora la seguridad, sino que también facilita la automatización de tareas en sistemas distribuidos [5].

La autenticación mediante certificados digitales se basa en una infraestructura de clave pública (PKI) y en el uso de certificados X.509. Este mecanismo permite validar

la identidad de un usuario o sistema mediante la verificación de un certificado firmado por una autoridad de certificación confiable. Al integrarse con políticas de control de acceso centralizadas, los certificados digitales brindan una solución escalable y segura, adecuada para entornos donde es necesario auditar y gestionar identidades de forma rigurosa [2].

Existen también mecanismos que utilizan contraseñas de un solo uso (OTP), generadas por dispositivos físicos o aplicaciones móviles. Estos métodos se emplean en esquemas de autenticación de doble factor, proporcionando una capa adicional de protección al requerir un código temporal que cambia con cada intento de acceso [3].

En entornos corporativos es común emplear mecanismos de autenticación centralizada como LDAP, RADIUS o Kerberos. Estos sistemas permiten unificar la gestión de credenciales y políticas de acceso, facilitando el mantenimiento y la trazabilidad de las sesiones de los usuarios. LDAP se utiliza habitualmente para autenticar usuarios a través de directorios jerárquicos, mientras que RADIUS es ampliamente implementado en redes y servicios VPN. Kerberos, por su parte, proporciona un sistema robusto de autenticación mutua mediante el uso de tickets temporales.

Por último, la autenticación multifactor combina distintos elementos para confirmar la identidad de un usuario, tales como una contraseña, un dispositivo físico o una característica biométrica. Este enfoque refuerza la seguridad general del sistema, ya que requiere múltiples formas de validación para conceder el acceso [3].

Cada par de claves incluye dos claves, una clave pública y una clave privada.

- La clave pública se copia en el servidor de destino, en el archivo de claves autorizadas $authorized_keys$. La información cifrada con esta clave solo podrá ser descifrada por quien posea la clave privada correspondiente.
- La clave privada permanece únicamente en poder del usuario. Su posesión constituye una prueba de identidad. Solo un usuario cuya clave privada coincida con la clave pública registrada en el servidor podrá autenticarse correctamente [6].

2.3. Autorización

Una organización debe implementar determinados mecanismos para evitar que un usuario, equipo, servicio o aplicación informática pueda acceder a datos o recursos con derechos distintos de los autorizados [7].

Mediante el control de acceso a los distintos recursos del sistema es posible implementar las medidas definidas por la organización, teniendo en cuenta las restricciones de acceso a las aplicaciones, a los datos guardados en el sistema informático, a los servicios ofrecidos (tanto internos como externos) y a otros recursos de tipo lógico del sistema [7].

La implementación del control de acceso en un sistema informático depende fundamentalmente de la gestión de cuentas de usuarios y de la gestión de permisos y privilegios. Para facilitar el control de acceso a los datos y aplicaciones se pueden definir distintos grupos de usuarios dentro del sistema [7].

2.3.1. Política de control de acceso

La política de control de acceso permite definir un conjunto de restricciones que determinan quién puede acceder a qué recursos, y bajo qué condiciones. Estas restricciones se establecen en función de la identidad del sujeto y el rol asignado al mismo [7].

Al aplicar el principio de mínimos privilegios, se asigna a cada usuario únicamente los permisos estrictamente necesarios para desempeñar sus funciones. Esta medida reduce significativamente la superficie de ataque y los riesgos asociados a errores o accesos indebidos [7]. Además, al realizar revisiones periódicas de los privilegios otorgados, se garantiza que los accesos continúen siendo pertinentes en función de las responsabilidades actuales del usuario [8].

Otro aspecto clave de las políticas de seguridad es la gestión de los privilegios administrativos. Al restringir estos derechos únicamente al personal autorizado, solo los administradores del sistema pueden conceder, modificar o revocar accesos a los recursos [9].

2.4. Auditoría

2.4.1. Monitoreo

El monitoreo constante del estado de los servidores y de los distintos dispositivos que conforman la infraestructura tecnológica de una organización es una práctica esencial dentro de cualquier política de seguridad bien estructurada. Esta acción preventiva permite identificar posibles accesos no autorizados, comportamientos inusuales o señales tempranas de intentos de ataque dirigidos a los recursos del sistema [10].

Para ello, es necesario habilitar y configurar los mecanismos de registro de actividad, conocidos como logs, en todos los equipos críticos. Estos registros actúan como fuentes de información clave, proporcionando datos valiosos sobre distintos eventos, entre ellos:

Inicios de sesión realizados por los usuarios en los servidores [11].

2.5. Clave pública-privada

Como se mencionó anteriormente, la autenticación con clave pública-privada es un método ampliamente utilizado para acceder a servidores, especialmente en entornos tipo UNIX/Linux, a través de protocolos como SSH (Secure Shell) [5].

Es bastante habitual que los usuarios generen el par de claves ellos mismos. Las implementaciones de SSH incluyen utilidades sencillas de utilizar para esto, como los comandos `ssh-keygen` para generar el par de claves [12].

2.6. Certificados digitales X.509

Un certificado digital es un documento público emitido por una autoridad certificante hacia un individuo, servidor u otra entidad (sujeto), con determinados privilegios de acceso. El certificado está asociado a una clave pública que le pertenece al sujeto. Además, el certificado contiene una serie de datos sobre la Autoridad de Certificación que emitió el certificado, el período de validez del mismo, el rol con que se emitió, entre las principales características [13].

2.7. Infraestructura de Clave Pública (PKI)

Una Infraestructura de Clave Pública (PKI) es el conjunto de herramientas, procedimientos y políticas que permiten cubrir el ciclo de vida de los certificados digitales, así como la gestión administrativa asociada. Su objetivo principal es proporcionar un entorno seguro de autenticación, cifrado y validación de identidades en entornos digitales [2].

El ciclo de vida de un certificado comienza con la solicitud del mismo, sigue con su emisión y gestión conforme a los pasos definidos en los procedimientos de la PKI, y finaliza con la expiración de su periodo de validez. Durante su vigencia, un certificado puede encontrarse en estado válido o revocado. Todas las transiciones están debidamente especificadas mediante procedimientos dentro de la infraestructura [13].

Los componentes esenciales de una PKI son:

- **Autoridad de Certificación (AC)**: Encargada de emitir, renovar y revocar los certificados digitales [13].
- **Autoridad de Registro (AR)**: Actúa como intermediaria entre el solicitante y la AC, validando la identidad del solicitante antes de la emisión del certificado [2].
- **Repositorio de certificados**: Espacio donde se almacenan los certificados públicos y las Listas de Revocación de Certificados (CRL) [13].
- **Lista de Revocación de Certificados (CRL)**: Documento que enumera los certificados que han sido revocados antes de su expiración natural [14].
- **Módulo de Seguridad Hardware (HSM)**: Dispositivo físico utilizado para generar, almacenar y proteger claves privadas, asegurando un nivel elevado de protección contra accesos no autorizados [15].

La PKI define un entorno de confianza basado en la certificación de las claves públicas, y se apoya en un conjunto de políticas de certificación y prácticas de certificación que establecen los criterios bajo los cuales se otorgan, utilizan y revocan los certificados. Estas políticas aseguran la homogeneidad y la confianza en la gestión de identidades digitales [2].

2.7.1. Componentes

Autoridad de Certificación

Dentro de una infraestructura de clave pública (PKI), la Autoridad de Certificación (AC) desempeña un rol central y esencial, ya que es responsable de emitir, firmar, renovar y revocar los certificados digitales utilizados por los distintos sujetos dentro del sistema [13]. Un certificado adquiere validez a partir del momento en que la AC aplica su firma digital, lo que garantiza su autenticidad e integridad.

La clave privada de la AC es el componente más crítico del sistema, ya que con ella se realiza la firma de los certificados. Por lo tanto, la seguridad de toda la infraestructura depende de la protección adecuada de esta clave. Para mitigar los riesgos, su almacenamiento y uso suelen delegarse a dispositivos especializados como los Módulos de Seguridad de Hardware (HSM), que están diseñados específicamente para proteger claves criptográficas y evitar su extracción [16].

En cuanto a la jerarquía, si la AC actúa como autoridad raíz, su certificado es autofirmado. En cambio, si forma parte de una jerarquía mayor, como una autoridad intermedia, su certificado es firmado por una AC superior, lo que refuerza la cadena de confianza que sustenta toda la PKI [17].

La operación de la AC está regida por políticas y procedimientos claramente definidos, formalizados en un documento denominado Prácticas de Certificación (Certification Practice Statement, CPS). Este documento detalla los criterios y métodos utilizados para la emisión, revocación, renovación y gestión de certificados, así como los controles de seguridad y auditoría implementados [18].

Además, la AC puede delegar tareas operativas, como la validación de identidad, en una Autoridad de Registro (AR), que actúa como punto de contacto entre los usuarios finales y la AC [2].

Autoridad de Registro

La Autoridad de Registro (AR) es un componente fundamental dentro de una PKI, encargada de llevar a cabo los procesos de validación y registro de los solicitantes de certificados digitales. Su principal función es actuar como intermediaria entre los usuarios finales (entidades finales) y la Autoridad de Certificación (AC) [2].

Durante el proceso de solicitud de un certificado, la AR verifica la identidad del solicitante mediante distintos mecanismos que pueden ser presenciales o remotos, de acuerdo con las políticas de la organización. Una vez realizada la verificación, la AR envía una solicitud formal a la AC para que emita y firme el certificado correspondiente [13].

Además de gestionar las solicitudes iniciales, la AR también se encarga de otros procesos relacionados con el ciclo de vida del certificado, entre ellos:

- **Revocación de certificados**: La AR recibe y valida las solicitudes de revocación, y una vez confirmadas, las envía a la AC para que el certificado afectado

sea incluido en la Lista de Certificados Revocados (CRL).

- **Actualización de información**: Administra solicitudes para actualizar los datos de los certificados, como cambios en el nombre, correo electrónico o datos organizativos del titular.
- **Reemisión de certificados**: Gestiona solicitudes de reemisión de certificados cuando no ha habido compromiso de la clave privada, pero se requiere modificar algún atributo del certificado.
- **Auditoría y registros**: Mantiene un registro detallado de todas las operaciones realizadas, lo cual es esencial para fines de auditoría, trazabilidad y cumplimiento normativo [18].
- **Comunicación de políticas**: Informa a los usuarios sobre los procedimientos, requisitos y políticas que regulan la emisión, renovación y revocación de certificados dentro de la PKI.

La AR cumple así una función de confianza crítica dentro del ecosistema PKI, garantizando que solo usuarios autorizados y correctamente identificados obtengan certificados válidos.

Autoridad de Validación

La Autoridad de Validación (AV) es un componente clave en una infraestructura de clave pública (PKI), encargada de verificar en tiempo real el estado de validez de los certificados digitales. Su principal responsabilidad es asegurar que un certificado no haya sido revocado o expirado en el momento en que se utiliza para establecer una comunicación segura o realizar una transacción electrónica [14].

La AV opera principalmente mediante dos mecanismos de validación:

- **CRL (Certificate Revocation List)**: La Autoridad de Validación puede consultar las listas de certificados revocados publicadas periódicamente por la AC. Este método implica descargar una lista completa y verificar si el certificado en cuestión se encuentra incluido en ella. Aunque es un método tradicional, puede no ser el más eficiente en entornos que requieren validación inmediata [17].
- **OCSP (Online Certificate Status Protocol)**: La AV puede utilizar este protocolo para consultar en línea el estado de un certificado específico. A diferencia de las CRL, OCSP proporciona una respuesta puntual sobre un único certificado, lo que reduce el uso de ancho de banda y mejora la velocidad de respuesta. En este esquema, la AV puede actuar como un servidor OCSP, devolviendo respuestas firmadas digitalmente a las solicitudes de validación [14].

Además de estos mecanismos, en entornos más complejos o distribuidos, la AV también puede utilizar protocolos avanzados o personalizados (como SCVP o validadores embebidos en aplicaciones) para responder de forma automatizada a las consultas sobre el estado de los certificados.

Las funciones de la Autoridad de Validación no solo mejoran la seguridad en las comunicaciones, sino que también optimizan el rendimiento y la experiencia de usuario

al permitir respuestas más rápidas que las obtenidas mediante el uso exclusivo de CRL. Asimismo, la AV puede integrarse con otros servicios de seguridad, como pasarelas de confianza o proxies TLS, para realizar validaciones de manera centralizada y eficiente.

Por último, las AV deben contar con mecanismos de alta disponibilidad, firma digital de respuestas (en el caso de OCSP), y políticas claras sobre el tiempo de retención de la información de estado, para garantizar un servicio confiable y conforme a los estándares de la infraestructura de clave pública.

2.7.2. Entidad final

Una entidad final es un usuario de la PKI, que puede ser un dispositivo, una persona o un servidor. Se la denomina End Entity porque, dentro de la jerarquía de certificados de la PKI, siempre representa el último eslabón: no está autorizada a emitir certificados por sí misma. La entidad final solicita su certificado a través de una Autoridad de Registro (AR), que actúa como intermediaria frente a la Autoridad Certificadora (AC) [2].

Una entidad final puede poseer múltiples certificados, los cuales comparten los mismos valores de identificación. Es importante destacar que una entidad final no debe entenderse exclusivamente como una persona física. Desde la perspectiva de una AC, puede tratarse de un individuo, pero en términos generales es cualquier entidad que posea un certificado [13].

Perfil de entidad final

Los perfiles de entidad final definen plantillas con configuraciones específicas para estas entidades. Aunque las PKI proporcionan un perfil predeterminado denominado Vacío (sin restricciones), en la mayoría de las PKI es útil, e incluso necesario, establecer ciertas limitaciones sobre los valores que los usuarios pueden utilizar al registrarse como entidad final.

Los valores definidos en un perfil de entidad final corresponden directamente a los atributos de dicha entidad, y por extensión, a los campos de identificación del certificado, con excepción de los datos relacionados con claves y firmas, que se especifican en los perfiles de certificado [13].

Entre los valores típicos se incluyen los Distinguished Name (DN) del sujeto (como país, organización, nombre común, etc.), nombres alternativos del sujeto, los perfiles de certificado disponibles y las AC autorizadas.

2.7.3. Perfil de certificado

Un perfil de certificado se utiliza para definir el contenido y las restricciones aplicables a los certificados emitidos. Esto incluye configuraciones como extensiones de certificado, algoritmos permitidos, tamaños de clave, entre otros. En esencia, el perfil

de certificado determina las características y limitaciones que tendrá un certificado al ser emitido [18].

2.7.4. Descripción general de los protocolos remotos

La PKI admite una variedad de protocolos remotos diseñados para cubrir distintos casos de uso, tanto en la emisión como en la gestión del ciclo de vida de los certificados digitales. Estos protocolos permiten a las entidades finales, sistemas automatizados y aplicaciones externas interactuar de forma segura con los servicios de certificación [14].

A continuación, se describen los principales protocolos admitidos:

OCSP (Online Certificate Status Protocol): Es el protocolo más utilizado para consultar en tiempo real el estado de un certificado digital (vigente, revocado o desconocido). OCSP es más eficiente que las listas de revocación (CRL), ya que permite una verificación puntual del estado sin necesidad de descargar listas completas [14].

SCEP (Simple Certificate Enrollment Protocol): Diseñado para dispositivos con recursos limitados, como routers y switches, permite solicitudes automatizadas de certificados. SCEP es ampliamente utilizado en entornos de red, aunque hoy en día se considera menos seguro en comparación con alternativas modernas [14].

EST (Enrollment over Secure Transport): Es una evolución más segura y moderna de SCEP, basada en HTTPS. Proporciona una inscripción más robusta de certificados e incluye funcionalidades como renovación y recuperación, además de autenticar clientes y servidores mutuamente [14].

ACME (Automatic Certificate Management Environment): Es un protocolo ampliamente adoptado para la emisión automática de certificados, especialmente popularizado por servicios como Let's Encrypt. Permite automatizar la solicitud, renovación y revocación de certificados para servidores web y otros servicios en línea [14].

CMP (Certificate Management Protocol): Es un protocolo robusto y flexible definido por la norma RFC 4210. CMP permite operaciones avanzadas como inscripción, renovación, revocación y recuperación de certificados, además de soportar flujos de trabajo con múltiples pasos y requisitos de aprobación [14].

API REST de administración de certificados: Proporciona una interfaz moderna y sencilla para integrar funcionalidades de gestión de certificados en aplicaciones externas. A través de esta API, es posible realizar operaciones como emisión, renovación, revocación y consulta de certificados, así como gestión de entidades finales [19].

Servicios web de la PKI: Estos servicios permiten realizar tareas administrativas avanzadas a través de interfaces basadas en estándares como SOAP. Incluyen funcionalidades como flujos de aprobación, administración de autoridades certificadoras (AC) y configuración de perfiles de certificado y entidad final [13].

Además, todos estos protocolos pueden ser reenviados por proxy mediante otra instancia de la PKI que actúe como Autoridad de Registro (AR), lo que permite cons-

truir arquitecturas distribuidas y jerárquicas. Esta capacidad es especialmente útil en organizaciones grandes, donde múltiples nodos de inscripción están conectados a una infraestructura centralizada de certificación [2].

2.8. PKCS11

El estándar PKCS11 (Public-Key Cryptography Standards 11), también conocido como Cryptoki, define una interfaz estándar para el acceso a dispositivos criptográficos, como tokens de hardware, tarjetas inteligentes (smart cards), y módulos de seguridad de hardware (HSM, Hardware Security Modules). Esta interfaz permite a las aplicaciones realizar operaciones criptográficas (como generación de claves, firma digital, cifrado/descifrado y gestión de certificados) sin preocuparse por los detalles internos del dispositivo subyacente [15].

PKCS11 proporciona un conjunto de funciones en forma de biblioteca compartida (por ejemplo, .so en Linux o .dll en Windows) que las aplicaciones pueden utilizar para interactuar con dispositivos seguros de forma uniforme [15].

2.8.1. Opciones de implementación

Existen varias opciones para implementar PKCS11, tanto en entornos de hardware como en software. La elección depende del nivel de seguridad, compatibilidad y flexibilidad que requiera la organización o sistema. A continuación se describen las principales alternativas:

Módulos PKCS11 basados en HSM

Los Hardware Security Modules (HSM) son dispositivos físicos diseñados para proteger claves criptográficas y realizar operaciones criptográficas en un entorno aislado y seguro. Suelen cumplir con estándares de seguridad como FIPS 140-2 o Common Criteria [15]. Fabricantes populares incluyen:

- Utimaco
- Thales (anteriormente SafeNet)
- AWS CloudHSM
- YubiHSM de Yubico

Estos dispositivos proporcionan bibliotecas PKCS11 compatibles que se integran con servidores de certificados, software de firma y sistemas de gestión de claves [16].

Tokens USB y Smart Cards

Las tarjetas inteligentes y los tokens USB (como los de Yubico, SafeNet o Feitian) son soluciones portátiles que también implementan PKCS11. Suelen utilizarse para autenticación multifactor, firma digital y almacenamiento seguro de certificados. Son ideales para casos en los que se requiere movilidad o autenticación individual por usuario [15].

Implementaciones de software

Existen implementaciones de PKCS11 completamente en software, utilizadas principalmente para entornos de prueba, automatización o cuando no se necesita el mismo nivel de seguridad física. Algunas opciones incluyen:

SoftHSM: Una implementación de software que simula un HSM, desarrollada por OpenDNSSEC. Muy útil para desarrollo y pruebas [20, 21, 22].

OpenSC: Biblioteca que soporta una variedad de tarjetas inteligentes y tokens, también provee una interfaz PKCS11.

Virtual HSMs: Algunos entornos en la nube (como Azure Key Vault o Google Cloud KMS) ofrecen compatibilidad con PKCS11 a través de módulos que emulan dispositivos HSM [16, 23, 24].

Entornos de nube y contenedores

Varios proveedores de servicios en la nube ofrecen servicios HSM administrados compatibles con PKCS11. Estas soluciones permiten a las organizaciones aprovechar capacidades criptográficas robustas sin la necesidad de mantener hardware físico. Ejemplos incluyen:

- AWS CloudHSM
- Azure Dedicated HSM
- Google Cloud HSM

Además, algunos entornos permiten montar módulos PKCS11 en contenedores Docker o en aplicaciones dentro de Kubernetes, lo que permite escalar servicios criptográficos de forma flexible [16, 25].

Compatibilidad con aplicaciones

Muchas aplicaciones y sistemas que requieren operaciones criptográficas (como EJBCA/PKI, navegadores, clientes VPN, sistemas de firma digital y bibliotecas de seguridad como OpenSSL) ofrecen soporte para módulos PKCS11. La correcta selección e integración del módulo garantiza compatibilidad, cumplimiento normativo y seguridad operativa [26, 27, 28, 29].

2.9. Protocolo SSH

El protocolo SSH (Secure Shell) es un protocolo criptográfico diseñado para proporcionar acceso remoto seguro en sistemas informáticos. Surgió como una alternativa robusta frente a protocolos antiguos como Telnet, rlogin y FTP, que transmiten información sensible en texto plano. SSH garantiza la confidencialidad, integridad y autenticidad de los datos mediante el uso de técnicas criptográficas avanzadas, funciones de integridad y mecanismos de autenticación [5].

La autenticación mutua entre cliente y servidor es un aspecto esencial del protocolo SSH. El cliente verifica la identidad del servidor mediante su clave de host, mientras que el servidor puede autenticar al cliente usando contraseñas, claves públicas o certificados. Esta doble autenticación refuerza significativamente la seguridad de la conexión [6].

SSH permite además la creación de túneles cifrados, también conocidos como port forwarding, que habilitan el acceso seguro a servicios internos de una red sin necesidad de exponerlos públicamente. Esta funcionalidad es ampliamente utilizada para encapsular protocolos no seguros o para establecer canales privados dentro de entornos corporativos [30].

En términos de resistencia a ataques, SSH incluye medidas contra ataques del tipo man-in-the-middle mediante la verificación de claves, y protege contra ataques de repetición o modificación de datos gracias al uso de mecanismos de integridad [4].

SSH es altamente portable y compatible con múltiples plataformas, incluyendo Linux, BSD, macOS y Windows, y puede ser fácilmente integrado en flujos de trabajo automatizados [12].

Es posible multiplexar múltiples canales virtuales sobre una única conexión SSH, lo que permite realizar varias operaciones simultáneamente sin necesidad de abrir conexiones adicionales [30].

El funcionamiento básico del protocolo SSH comienza cuando el cliente inicia una conexión TCP al puerto 22 del servidor, que es el puerto predeterminado para este servicio. Una vez establecida la conexión, se realiza un proceso de intercambio de claves y negociación de algoritmos criptográficos. Durante esta etapa, cliente y servidor acuerdan los algoritmos que se utilizarán para el cifrado, la integridad de los datos y la autenticación, asegurando así que la comunicación posterior sea segura y confidencial [6].

A continuación, se procede con la autenticación del usuario. El cliente intenta autenticarse utilizando alguno de los métodos permitidos por el servidor, como contraseñas, claves públicas, certificados u otros mecanismos. Una vez que la autenticación es exitosa, se establece una sesión segura en la que el usuario puede iniciar una shell interactiva, ejecutar comandos remotos, establecer túneles cifrados o realizar transferencias de archivos.

Esta arquitectura eficiente y segura ha convertido a SSH en un estándar para la administración remota de sistemas y la comunicación protegida en redes inseguras [30].

2.9.1. OpenSSH

OpenSSH es una herramienta opensource fundamental para la conectividad remota segura, basada en el protocolo SSH (Secure Shell). Su principal función es cifrar todo el tráfico de red entre el cliente y el servidor, garantizando la confidencialidad e integridad de la información transmitida. De esta forma, protege contra escuchas no autorizadas, secuestros de sesión y otros tipos de ataques asociados a redes inseguras [12].

Además de establecer conexiones seguras, OpenSSH proporciona un conjunto completo de funcionalidades para tunelización cifrada, múltiples métodos de autenticación y una amplia gama de opciones de configuración que permiten adaptarse a distintos entornos y necesidades de seguridad [12].

La suite de OpenSSH incluye varias herramientas orientadas a la administración remota y la transferencia segura de archivos [6]:

- **ssh (Secure Shell)**: Comando principal para establecer sesiones de terminal remotas cifradas. Se utiliza para acceder y administrar servidores de manera segura a través de una red.
- **scp (Secure Copy)**: Permite copiar archivos de forma segura entre un sistema local y uno remoto (o entre dos sistemas remotos), utilizando el protocolo SSH para garantizar la confidencialidad e integridad de los datos.
- **sftp (SSH File Transfer Protocol)**: Proporciona una interfaz interactiva para la transferencia y gestión de archivos a través de SSH. Permite navegar por directorios, subir, descargar y modificar archivos con mayor flexibilidad que scp.

2.10. PAM

PAM, sigla de Pluggable Authentication Modules, es una infraestructura modular integrada en sistemas UNIX y Linux que permite gestionar la autenticación de usuarios de forma flexible y centralizada. Esta arquitectura es utilizada tanto por los componentes de entrada del sistema como por aplicaciones que están diseñadas para ser compatibles con PAM, facilitando un control uniforme sobre quién accede al sistema y bajo qué condiciones [31].

Cada método de autenticación, como contraseñas, se implementa mediante un módulo PAM específico. Esta modularidad permite adaptar y actualizar fácilmente los métodos de autenticación sin necesidad de modificar las aplicaciones. Los administradores pueden definir políticas de autenticación estrictas y personalizadas para cada servicio del sistema, optimizando la seguridad y simplificando la administración de usuarios [32].

Cuando una aplicación necesita autenticar a un usuario, realiza una llamada a la biblioteca PAM. Esta, a su vez, carga los módulos de autenticación configurados para esa aplicación específica. Los módulos se encargan de ejecutar las comprobaciones necesarias, como verificar contraseñas, autenticar con claves criptográficas o validar el acceso

mediante hardware externo. Finalmente, PAM devuelve a la aplicación el resultado del proceso de autenticación, permitiendo o denegando el acceso según corresponda [31].

PAM proporciona un marco versátil y potente para implementar políticas de autenticación coherentes y seguras en entornos UNIX y Linux [32].

- Autenticación fallida de usuarios.
- Comandos ejecutados durante las sesiones en cada servidor.
- Intentos de accesos no autorizados a recursos protegidos, como archivos o directorios.
- Acciones que impliquen una posible infracción de las políticas de seguridad establecidas.

El análisis regular de estos logs permite no solo detectar incidentes en tiempo real, sino también generar alertas, establecer patrones de comportamiento y mejorar la capacidad de respuesta ante eventuales amenazas [11].

2.11. Ansible

Ansible es una herramienta de automatización de código abierto diseñada para facilitar la gestión de configuraciones, el aprovisionamiento de servidores, la implementación de aplicaciones y la orquestación de tareas repetitivas [33]. Su arquitectura sin agentes (agentless) permite ejecutar tareas de automatización directamente sobre los nodos gestionados a través del protocolo SSH, sin necesidad de instalar software adicional en los servidores [33].

Ansible utiliza archivos YAML, denominados *playbooks*, para definir de manera declarativa las tareas que deben ejecutarse en los sistemas gestionados. Cada tarea, como instalar software, configurar servicios o administrar usuarios, se lleva a cabo mediante módulos, que son pequeños componentes encargados de ejecutar acciones concretas [34].

En entornos donde se requieren despliegues eficientes, Ansible resulta especialmente útil. Su uso permite reducir errores humanos, agilizar procesos de configuración y asegurar que todos los destinos gestionados cumplan con la misma configuración [35].

2.12. Elasticsearch

Elasticsearch es un motor de búsqueda y análisis distribuido, de código abierto, basado en Apache Lucene. Permite almacenar y buscar datos de forma eficiente, tanto estructurados como no estructurados, y ofrece capacidades analíticas para explorar grandes cantidades de información. Puede ser distribuido en múltiples servidores para manejar grandes volúmenes de datos y tráfico [10].

Elasticsearch es el núcleo de Elastic Stack. En combinación con Kibana, ofrece un conjunto de funcionalidades clave en los ámbitos de la observabilidad, la búsqueda y la seguridad [9].

En el área de observabilidad, permite recopilar, almacenar y analizar registros, métricas y rastreos provenientes de aplicaciones, servicios y sistemas. Esto incluye el monitoreo del rendimiento de aplicaciones (APM), la supervisión de la interacción de usuarios reales (RUM) con aplicaciones web, y la compatibilidad con el estándar OpenTelemetry para reutilizar la instrumentación existente en la recopilación de telemetría [11].

En cuanto a sus capacidades de búsqueda, Elasticsearch ofrece mecanismos de búsqueda de texto completo mediante índices invertidos, análisis de texto y tokenización, así como búsqueda semántica y vectorial, que permiten interpretar el contexto e intención de las consultas. Estas funciones se complementan con técnicas de búsqueda híbrida, combinando texto y vectores, y soporte para consultas geoespaciales. También es posible desarrollar motores de búsqueda personalizados o integrarlos en aplicaciones web mediante sus APIs, incluso aplicando estrategias de recuperación aumentada (RAG) para complementar modelos de inteligencia artificial generativa con información actualizada y contextual.

En el ámbito de la seguridad, Elasticsearch proporciona una solución robusta para la gestión de información y eventos de seguridad (SIEM), que permite recolectar, correlacionar y analizar eventos provenientes de múltiples fuentes. Además, facilita el monitoreo de endpoints y ofrece herramientas para la búsqueda activa de amenazas (threat hunting), proporcionando una base sólida para la detección y respuesta ante incidentes de seguridad en tiempo real [9].

2.13. Kibana

Kibana es la interfaz gráfica de usuario de Elasticsearch y constituye un componente fundamental dentro del stack. Kibana permite a los usuarios explorar sus datos en tiempo real, identificar patrones, descubrir tendencias y generar reportes significativos para la toma de decisiones. [36]

Una de las funcionalidades clave de Kibana es su capacidad de consulta y filtrado. Utilizando el lenguaje específico del dominio (DSL) de Elasticsearch, se pueden construir consultas potentes que admiten búsquedas de texto completo, lógica booleana, coincidencias difusas, búsquedas de proximidad y semánticas, entre otras.

Además, Kibana permite aplicar agregaciones, que son herramientas de análisis avanzado para transformar datos sin procesar en estructuras organizadas y comprensibles. Las agregaciones permiten calcular métricas estadísticas como promedios, sumas y medianas, así como agrupar datos en histogramas, categorías o realizar análisis jerárquicos de múltiples niveles, facilitando así una comprensión profunda de la información.

El componente de aprendizaje automático integrado en Elasticsearch se potencia a través de Kibana, permitiendo la detección de anomalías en series temporales, la

predicción de tendencias futuras, el análisis de patrones estacionales y la ejecución de técnicas avanzadas de procesamiento del lenguaje natural, como la búsqueda semántica.

Otra característica destacada de Kibana es el soporte para scripting, que permite a los usuarios definir transformaciones y manipulaciones personalizadas de los datos durante los procesos de consulta y agregación. [37]

La sección Discover de Kibana proporciona un entorno interactivo para la exploración directa de datos sin procesar. Desde allí, es posible examinar documentos en los índices, aplicar filtros y consultas personalizadas, así como visualizar resultados en tiempo real.

Para una representación visual efectiva, Kibana permite crear múltiples tipos de visualizaciones, que pueden ser agrupadas en paneles de control interactivos. Estos paneles permiten combinar diversas visualizaciones en una vista unificada, mostrar datos de múltiples fuentes, y personalizar la disposición según las preferencias del usuario o las necesidades del flujo de trabajo. [38]

Asimismo, Kibana permite compartir hallazgos mediante informes generados de forma automática o bajo demanda. Estos informes pueden exportarse en formatos como PDF o CSV, y distribuidos entre equipos o partes interesadas. Complementando esta funcionalidad, Kibana ofrece la capacidad de definir alertas, que monitorean continuamente los datos y notifiquen al usuario cuando se cumplen ciertas condiciones predefinidas, lo cual es vital para una respuesta rápida ante eventos importantes o incidentes. [39]

2.14. Beats

Beats es un conjunto de agentes de datos livianos y de código abierto desarrollados por Elastic, diseñados para recopilar y enviar información operativa directamente a Elasticsearch [11]. Cada Beat está especializado en un tipo de dato específico y se instala como agente en los servidores o sistemas donde se produce la información.

Filebeat monitoriza los archivos de registro o las ubicaciones que se especifiquen, recopila eventos de registro y los reenvía a Elasticsearch para su indexación [11].

El funcionamiento interno de Filebeat se basa en dos elementos clave: entradas y recolectores. Las entradas son responsables de identificar y gestionar las ubicaciones o fuentes donde se deben leer los registros. Una vez detectadas las fuentes, Filebeat crea recolectores individuales, uno por cada archivo encontrado. Estos recolectores son los encargados de leer el contenido en los archivos a medida que se genera, y enviar esa información a la salida configurada, generalmente Elasticsearch [11].

Para asegurar la fiabilidad en la lectura de datos, Filebeat guarda el estado de cada archivo que monitorea, utilizando identificadores únicos. Esto permite manejar situaciones comunes como renombrado o movimiento de archivos sin perder seguimiento, garantizando que no se dupliquen ni se omitan eventos ya procesados [11].

En su funcionamiento típico, cuando se inicia Filebeat, se activan una o varias entradas que buscan activamente en las rutas configuradas en busca de archivos. Cada

vez que una entrada detecta un nuevo archivo o una modificación, se inicia un recolector asociado que comienza a leer los registros. La información recolectada se transfiere a través del framework `libbeat`, una capa intermedia encargada de agregar los eventos y preparar los datos para su envío. Finalmente, los datos son enviados a la salida configurada, donde pueden ser indexados, visualizados y analizados dentro de Elastic Stack [11].

2.15. Virtualización

La virtualización es una tecnología que permite ejecutar múltiples sistemas operativos y aplicaciones en un único computador físico, mediante la creación de entornos virtuales aislados conocidos como máquinas virtuales (VMs). Cada VM actúa como un sistema independiente, con su propio sistema operativo, recursos asignados y configuración, aunque compartan la infraestructura física subyacente [40].

Esta tecnología se apoya en un software llamado *hipervisor*, que se encarga de abstraer y asignar los recursos del hardware físico, como CPU, memoria, almacenamiento y red, a las diferentes máquinas virtuales [41]. Existen dos tipos principales de hipervisores: los de tipo 1 o *bare-metal*, que se ejecutan directamente sobre el hardware, y los de tipo 2, que funcionan sobre un sistema operativo anfitrión [42].

La virtualización ofrece múltiples beneficios, entre ellos la consolidación de cargas de trabajo, una mejor utilización de los recursos, facilidad de escalado, aislamiento entre entornos, y mayor flexibilidad para la administración y recuperación ante fallos [43]. Esto se traduce en una reducción de costos operativos, mejora en la eficiencia energética y simplificación de las tareas de mantenimiento [40].

2.16. Docker

Docker es una plataforma de código abierto que permite automatizar la creación, despliegue y ejecución de aplicaciones mediante el uso de contenedores. Un contenedor es una unidad ligera y portátil que incluye todo lo necesario para ejecutar una aplicación: el código, las bibliotecas, las dependencias y la configuración del sistema, lo que garantiza que la aplicación se ejecute de forma uniforme en cualquier entorno, ya sea en desarrollo, pruebas o producción [44].

A diferencia de las máquinas virtuales tradicionales, que emulan un sistema operativo completo, los contenedores de Docker comparten el mismo kernel del sistema operativo anfitrión. Esto permite una mayor eficiencia en el uso de recursos, menor sobrecarga y tiempos de arranque mucho más rápidos. Cada contenedor es aislado, lo que proporciona seguridad y estabilidad entre aplicaciones que se ejecutan simultáneamente en el mismo host [45].

Docker se basa en el uso de imágenes, que son plantillas a partir de las cuales se crean los contenedores. Las imágenes pueden construirse a partir de archivos llamados Dockerfile, en los que se definen de manera declarativa las instrucciones necesarias para

preparar el entorno de ejecución. Estas imágenes pueden almacenarse y compartirse en registros como Docker Hub o repositorios privados, lo que facilita la distribución y el versionado del software.

Una de las principales ventajas de Docker es su capacidad para integrarse con herramientas de automatización, orquestadores como Kubernetes y flujos de integración y entrega continua (CI/CD). Esto convierte a Docker en una solución ideal para entornos de desarrollo ágiles, DevOps y despliegues en la nube [46].

Docker ofrece una forma eficiente, reproducible y escalable de empacar, distribuir y ejecutar aplicaciones, promoviendo la portabilidad entre entornos y reduciendo problemas de compatibilidad y configuración.

2.16.1. Docker Compose

Docker Compose es una herramienta que permite definir y administrar aplicaciones multi-contenedor de forma sencilla. Utiliza un archivo de configuración en formato YAML (por defecto llamado `docker-compose.yml`), donde se describen los servicios que componen una aplicación, así como sus volúmenes, redes, variables de entorno y otras configuraciones necesarias para su ejecución [47].

Con Docker Compose, en lugar de iniciar manualmente múltiples contenedores con configuraciones específicas, se puede levantar toda la infraestructura con una sola instrucción. Esto simplifica enormemente la orquestación de servicios que dependen unos de otros, como bases de datos, servidores web, backends y frontends.

Cada servicio definido en el archivo Compose se basa en una imagen Docker, que puede ser descargada desde un registro o construida a partir de un Dockerfile. Además, Docker Compose permite vincular servicios a través de redes internas, definir políticas de reinicio, montar volúmenes persistentes y manejar la configuración de forma declarativa y reutilizable.

Esta herramienta es especialmente útil en entornos de desarrollo, pruebas e integración continua, ya que permite replicar de forma idéntica el entorno de producción en cualquier máquina. Asimismo, facilita la colaboración entre equipos, ya que el archivo Compose actúa como documentación viva de la infraestructura necesaria para ejecutar la aplicación.

Docker Compose extiende las capacidades de Docker al facilitar el manejo de múltiples contenedores como una única unidad lógica, promoviendo la automatización, la reproducibilidad y la eficiencia en la gestión de entornos complejos.

2.17. API

Una API (*Application Programming Interface*, por sus siglas en inglés) es una interfaz de programación de aplicaciones que actúa como un conjunto de definiciones y protocolos utilizados para desarrollar e integrar el software de distintas aplicaciones. Su función principal es permitir que distintos sistemas interactúen entre sí de forma

estandarizada, posibilitando el acceso a funciones o datos sin necesidad de conocer los detalles internos de implementación [48].

Las API funcionan como intermediarios entre los usuarios o clientes y los recursos o servicios que desean utilizar. Cuando una aplicación realiza una solicitud a través de una API, esta se encarga de traducirla de forma que el sistema receptor la entienda y pueda responder de manera adecuada. Este proceso garantiza que la comunicación entre componentes sea clara, segura y controlada [49].

2.17.1. API REST

REST se basa en una estructura de cliente-servidor sin estado, en la que las operaciones se realizan utilizando métodos estándar como GET, POST, PUT, PATCH y DELETE. Este enfoque permite una forma flexible, ligera y escalable de integrar aplicaciones y conectar componentes, siendo ampliamente adoptado en entornos basados en microservicios [50]. Por estas razones, las API REST han surgido como un método común para conectar componentes y aplicaciones en una arquitectura de microservicios.

Los desarrolladores pueden elaborar API REST utilizando prácticamente cualquier lenguaje de programación y admitir diversos formatos de datos. El único requisito es que se alineen con una serie de principios de diseño REST, también conocidos como restricciones arquitectónicas:

- **Interfaz uniforme**: todas las solicitudes de API para un recurso específico deben tener un formato consistente y estar identificadas mediante un único URI (Identificador Uniforme de Recursos).
- **Desacoplamiento cliente-servidor**: las aplicaciones de cliente y de servidor deben ser completamente independientes entre sí. El cliente solo necesita conocer el URI del recurso, mientras que el servidor no debe tener conocimiento ni control sobre la implementación del cliente, limitándose a enviar los datos solicitados.
- **Sin estado**: las API REST no tienen estado, lo que implica que cada solicitud del cliente debe contener toda la información necesaria para que el servidor la procese, sin conservar contexto o información de solicitudes anteriores.
- **Capacidad de almacenamiento en caché**: los recursos pueden almacenarse en caché. Las respuestas del servidor también deben contener información sobre si se permite el almacenamiento en caché para el recurso entregado. El objetivo es mejorar el rendimiento del cliente, al tiempo que reduce la carga sobre el servidor.
- **Arquitectura del sistema en capas**: las llamadas y respuestas pasan por diferentes capas, puede haber varios intermediarios diferentes en el bucle de comunicación. Las API REST deben diseñarse de modo que ni el cliente ni el servidor puedan saber si se comunica con la aplicación final o con un intermediario.

En conjunto, estas características hacen de REST una opción poderosa y eficiente para el diseño de interfaces que requieren interoperabilidad, simplicidad y rendimiento en arquitecturas modernas distribuidas.

2.18. EJBCA

EJBCA (Enterprise Java Beans Certificate Authority) es una solución de código abierto para la implementación de una Autoridad Certificadora (CA) dentro de una infraestructura de clave pública (PKI). Desarrollada en lenguaje Java y basada en la especificación EJB (Enterprise JavaBeans), esta herramienta ofrece una arquitectura escalable, modular y altamente configurable para la gestión del ciclo de vida de certificados digitales [51].

EJBCA es mantenida por PrimeKey, actualmente parte de la empresa Keyfactor, y se destaca por su robustez, estabilidad y cumplimiento con estándares internacionales como X.509 para certificados digitales, PKCS#10 para solicitudes de firma, OCSP para validación en línea, CRL para listas de revocación, y protocolos como CMP, SCEP y EST [52]. Gracias a esta amplia compatibilidad, es adoptada en sectores críticos como gobiernos, telecomunicaciones, banca, sistemas industriales e Internet de las cosas (IoT).

Entre sus principales características se encuentra el soporte de certificados X.509, la emisión y gestión de listas CRL, y validación OCSP, así como la capacidad de integrarse con Hardware Security Modules (HSM) para proteger claves privadas de manera segura. Además, EJBCA proporciona interfaces web de administración, así como APIs REST y SOAP, que facilitan su integración con otros servicios y sistemas externos. También permite la automatización del ciclo de vida de los certificados, incluyendo su emisión, renovación y revocación [53].

En un escenario donde la protección de la identidad digital y la integridad de las comunicaciones es fundamental, EJBCA se presenta como una solución confiable para establecer y mantener una infraestructura de clave pública sólida. Su modelo de desarrollo open source y su capacidad de adaptación a distintos entornos hacen de EJBCA una opción flexible, accesible y segura frente a soluciones propietarias.

2.18.1. Tokens criptográficos

En EJBCA, el término token criptográfico se refiere a un área de memoria que contiene claves, ya sea una ranura del HSM o un almacén de claves de software creado localmente.

Lo que todos los tokens criptográficos EJBCA tienen en común es que:

- Un token criptográfico puede existir en una sola ranura.
- Un token criptográfico puede contener cualquier número de pares de claves y alias.

Los tokens criptográficos tienen varias aplicaciones diferentes en EJBCA:

- Se incluye en las CA para la firma de certificados y CRL.
- Los respondedores de OCSP lo utilizan para firmar las respuestas de OCSP.

- Se utiliza para autenticar la comunicación TLS entre nodos EJBCA.

3. Descripción de requerimientos

3.0.1. Stakeholders

El sistema propuesto involucra varios actores que requieren y utilizan el servicio de autenticación y control de acceso. A continuación, se describen los principales stakeholders y su relación con los requerimientos del sistema.

Usuarios técnicos

Son los usuarios que acceden a los servidores virtuales para realizar tareas de mantenimiento, configuración, instalación o monitoreo. Su acceso debe estar autenticado mediante certificados digitales y restringido según su nivel de autorización.

Administrador del sistema

Es responsable de la infraestructura virtual y de garantizar que los servicios estén protegidos. Administra las políticas de control de acceso, monitorea los logs de auditoría y verifica el cumplimiento de las normas de seguridad. También puede integrar y configurar herramientas de gestión de certificados en los servidores.

Oficial de Registro (OR)

Es la persona designada para validar la identidad de los usuarios técnicos antes de emitir un certificado digital. Su rol es crítico dentro de la PKI, ya que actúa como puente entre el usuario final y la Autoridad de Certificación. También gestiona solicitudes de revocación y reemisión.

Administrador de la PKI

Aunque puede coincidir con otros roles, este actor es responsable del mantenimiento general de la infraestructura PKI, incluyendo la operación de la Autoridad de Certificación, el sistema de validación (CRL/OCSP), y la publicación segura de los certificados.

3.1. Características y Requerimientos del Sistema

3.1.1. Requerimientos Funcionales

El sistema debe cumplir con las siguientes consideraciones operativas:

- **RF1**: Emitir, renovar y revocar certificados digitales X.509 a través de un sistema centralizado implementada con EJBCA.
- **RF2**: Los usuarios deben conectarse a los servidores de manera remota utilizando su certificado digital X.509 emitido por la autoridad certificadora implementada con EJBCA.
- **RF3**: El sistema debe verificar el rol del usuario mediante la información contenida en su certificado digital antes de permitir el acceso a los servidores.
- **RF4**: El administrador de sistema debe configurar los roles de los usuarios access, sysadmin, databases y devops de los servidores de una forma automatizada y centralizada.
- **RF5**: El sistema debe recolectar, centralizar y almacenar los registros de autenticación y actividades de los usuarios en los servidores, permitiendo su análisis posterior. Este requerimiento sirve como base para funcionalidades avanzadas de monitoreo, auditoría, correlación y reporte descritas en el RNF8.
- **RF6**: El sistema debe registrar los comandos ejecutados por cada usuario con su respectiva marca de tiempo, permitiendo su análisis gráfico y temporal.

3.1.2. Requerimientos No Funcionales

El sistema debe cumplir con los siguientes requerimientos no funcionales:

- **RNF1**: El sistema debe ser escalable, un administrador debe configurar un nuevo host o un rol sin perjucio de los anteriores.
- **RNF2**: Las herramientas utilizadas deben ser de código abierto y contar con documentación oficial y soporte activo de la comunidad.
- **RNF3**: El sistema debe registrar toda actividad relevante para fines de auditoría.
- **RNF4**: Las operaciones de emisión de certificados deberían completarse en menos de 10 minutos bajo condiciones normales.
- **RNF5**: Todas las comunicaciones entre los componentes del sistema debe realizarse utilizando protocolos cifrados, garantizando la confidencialidad e integridad de los datos.
- **RNF6**: La infraestructura debe ser escalable, permitiendo incorporar nuevos nodos sin necesidad de rediseñar la arquitectura base.
- **RNF7**: Las tareas de configuración y actualización del sistema debe ser automatizada, con el fin de reducir errores manuales y facilitar el mantenimiento.
- **RNF8**: Toda operación del sistema debe quedar registrada con su correspondiente marca de tiempo, usuario, y acción realizada, permitiendo su posterior auditoría.
- **RNF9**: Para mejorar la seguridad, las claves privadas asociadas a la Autoridad Certificadora (CA) deben ser almacenadas de forma segura.

3.2. Análisis de Riesgos

El sistema de seguridad propuesto está expuesto a diferentes tipos de amenazas que deben ser consideradas para minimizar su impacto. A continuación, se describen algunos de los riesgos identificados y las medidas de mitigación correspondientes:

Riesgo	**Impacto**	**Probabilidad**	**Mitigación**
Compromiso de la autoridad certificadora	Crítico	Bajo	Aislamiento de la CA, uso de HSM, backup regular y monitoreo
Exposición de claves privadas de servidores	Alto	Bajo	Almacenamiento seguro, uso de permisos restrictivos y rotación de claves
Fallos en la configuración de servicios	Medio	Medio	Uso de herramientas de automatización (Ansible) y validación continua
Acceso no autorizado a logs	Medio	Bajo	Control de acceso y cifrado en tránsito
Fallas de hardware en el servidor físico	Crítico	Bajo	Copias de seguridad, redundancia y snapshots periódicos
Saturación del sistema de monitoreo (Elastic)	Medio	Bajo	Configuración adecuada de índices, almacenamiento escalable

Cuadro 3.1: Análisis de riesgos y medidas de mitigación

3.3. Plan de Trabajo

El desarrollo del proyecto se dividió en varias fases, permitiendo una implementación organizada, a continuación se listan las fases:

1. **Análisis y diseño**
    - Relevamiento de necesidades y definición de arquitectura.
    - Elección de herramientas de software y hardware apropiadas.comunidad.
2. **Configuración de la infraestructura base**
    - Instalación y configuración de VMware ESXi.
    - Creación de las máquinas virtuales necesarias.
3. **Despliegue y configuración de Infraestructura de Clave Pública**

- Instalación y configuración del framework.
- Creación del crytotoken y configuración de la AC.
- Establecer del perfil de certificado.
- Definir de perfil de entidad final.

4. **Despliegue y configuración del sistema de autencación**
    - Instalación de PAM en servidores.
    - Despliegue del servicio de autenticación.
5. **Implementación del sistema de autorización**
    - Configuración del filtro de rol en el sistema de acceso.
    - Definición de políticas de operaciones permitidas.
    - Creación usuarios de sistemas para roles específicos.
    - Desarrollo de playbooks con Ansible para automatizar configuraciones.
6. **Implementación del sistema de monitoreo**
    - Instalación de Filebeat, Elasticsearch y Kibana.
    - Configuración de paneles y reglas de auditoría.
7. **Pruebas, ajuste fino y cierre**
    - Validación de requerimientos funcionales y no funcionales.
    - Ajustes finales, presentación y entrega de la tesis.
8. **Redacción de informe**

4. Diseño

En este capítulo se presenta el diseño del sistema propuesto como solución técnica, abarcando tanto su arquitectura a nivel de sistemas como los componentes de software que lo conforman. Se detallan los modelos estáticos y dinámicos que estructuran el comportamiento del sistema, así como la topología de red que permite su despliegue seguro y operativo.

4.1. Diseño a nivel de sistemas

El diseño a nivel de sistemas contempla la arquitectura general de los componentes que integran la solución: servidores virtuales, servicios de infraestructura de clave pública (PKI), módulos de control de acceso y herramientas de auditoría y configuración.

La arquitectura se basa en un enfoque distribuido y modular, con separación clara de responsabilidades. Cada subsistema cumple una función específica dentro del flujo de autenticación, autorización y monitoreo. A continuación, se describen los componentes principales:

- **Gateway (Bastion)**: Es el servidor de frontera que actúa como único punto de entrada a la red interna desde la red pública. Los usuarios técnicos se conectan inicialmente a este nodo, donde se realiza una primera validación de credenciales mediante módulos PAM basados en certificados.
- **Control de acceso basado en roles (RBAC)**: Corresponde al conjunto de servicios que autentican y autorizan a los usuarios técnicos. Está compuesto por una Autoridad Certificadora (PKI - EJBCA), una Autoridad de Registro y un servicio de validación en línea (OCSP). También verifica que el rol declarado en el certificado coincida con los permisos configurados.
- **Hosts (Destino N)**: Son los servidores internos en los que los usuarios técnicos ejecutan sus tareas operativas. El acceso a estos servidores también está mediado por un módulo PAM que revalida los permisos del usuario antes de conceder el acceso.
- **Servidor de configuración y monitoreo (ARCenter)**: Es el componente encargado de aplicar las políticas de control de acceso (usando Ansible), recolectar logs de auditoría (con Filebeat y Elasticsearch) y visualizarlos (mediante Kibana). Este subsistema centraliza la configuración de roles, la trazabilidad de accesos y la supervisión del sistema.

4.1.1. Modelos Estáticos

Los modelos estáticos representan la estructura del sistema y sus relaciones. Por ello, optamos por dos tipos de diagramas distintos para definir las interacciones del sistema.

Diagrama de Definición de Bloques

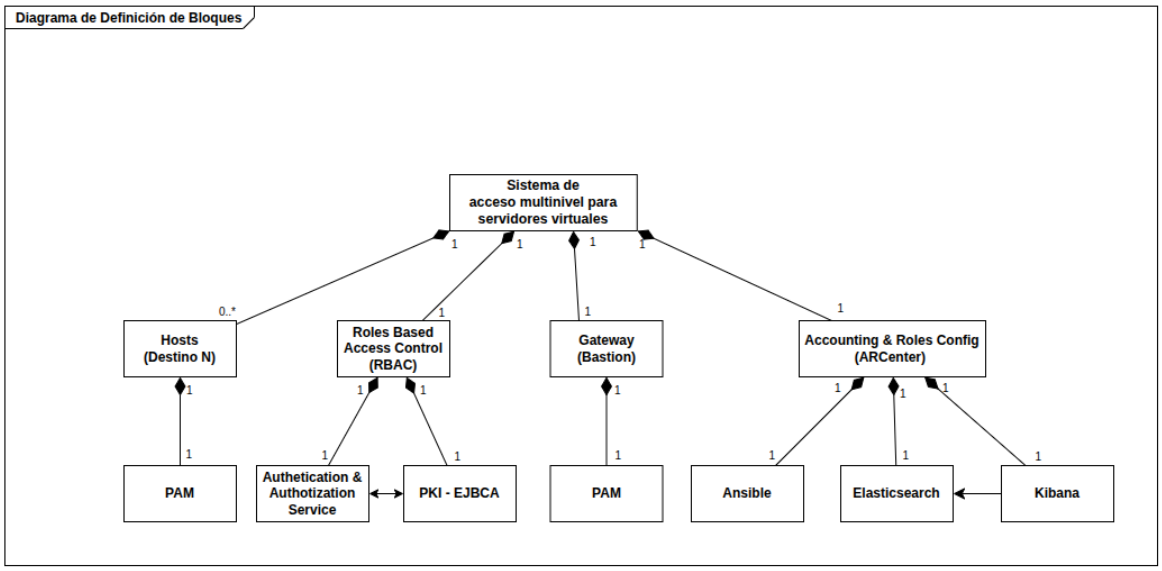

Figura 4.1: Diagrama de Definición de Bloques del sistema

La Figura 4.1 representa la arquitectura lógica del sistema propuesto para gestionar el acceso seguro y multicliente a servidores virtuales en un entorno controlado. Se identifican cuatro bloques funcionales principales que conforman el sistema, cada uno de ellos compuesto por subcomponentes especializados:

- **Hosts (Destino N)**: Representan los servidores destino sobre los cuales los usuarios técnicos ejecutan sus tareas operativas. Cada host cuenta con la integración de un módulo **PAM (Pluggable Authentication Module)**, que permite aplicar políticas de autenticación local basadas en certificados digitales y control de acceso por roles.
- **Roles Based Access Control (RBAC)**: Es el servidor encargado de aplicar los controles de acceso en función de la identidad y el rol del usuario. Está compuesto por:
  - **Authentication & Authorization Service**: Subsistema que valida en tiempo real que el certificado utilizado por el usuario sea legítimo y que su rol tenga autorización para acceder al recurso solicitado.
  - **PKI - EJBCA**: Infraestructura de clave pública que provee servicios de certificación digital. EJBCA actúa como Autoridad Certificadora (CA), Au-

toridad de Registro (RA) y Autoridad de Validación (VA), gestionando el ciclo de vida de los certificados.

- **Gateway (Bastion)**: Constituye el punto único de acceso al entorno interno. Todos los intentos de conexión a los servidores destino deben pasar por este servidor, el cual también utiliza un módulo **PAM** para realizar la autenticación inicial y aplicar filtros de acceso.
- **Accounting & Roles Config (ARCenter)**: Responsable de la configuración de roles, distribución de credenciales y auditoría de accesos. Se compone de:
  - **Ansible**: Herramienta de automatización utilizada para distribuir políticas de acceso, crear usuarios y aplicar configuraciones en los distintos servidores.
  - **Elasticsearch**: Almacena los registros generados por los eventos de autenticación y acceso, en formato estructurado.
  - **Kibana**: Provee una interfaz visual para la exploración, análisis y visualización de los registros almacenados en Elasticsearch.

Las relaciones entre los bloques están modeladas mediante composiciones fuertes, indicando que los componentes forman parte integral del sistema. La cardinalidad `0..*` en el bloque "Hosts (Destino N)" refleja la posibilidad de gestionar múltiples servidores destino de forma escalable. Este modelo de definición de bloques permite representar con claridad los roles funcionales y su interacción dentro del sistema de acceso multicliente.

Diagrama Interno de Bloques

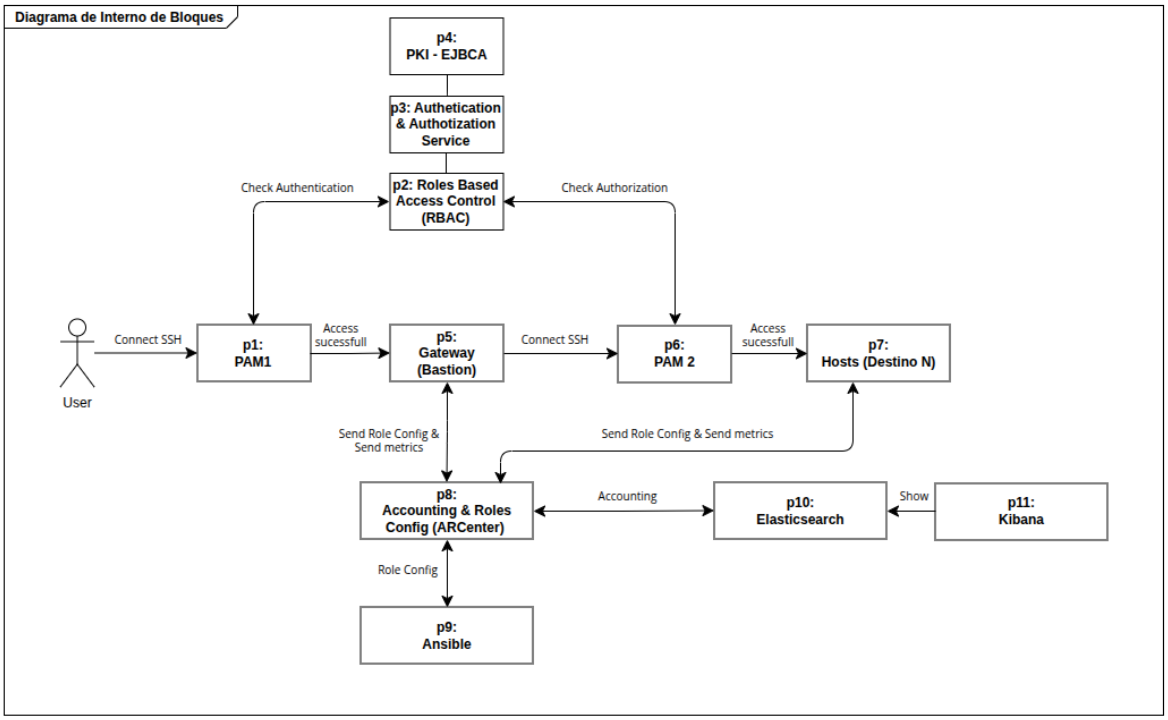

Figura 4.2: Diagrama Interno de Bloques del sistema

La Figura 4.2 representa el flujo funcional del sistema propuesto, modelando la interacción entre los principales módulos encargados de la autenticación, autorización, control de acceso y auditoría sobre servidores virtuales. Este diagrama permite entender de forma clara cómo se encadenan los procesos de seguridad desde el intento de acceso inicial hasta la ejecución de tareas en el host destino.

- **p1: PAM1** — Primer punto de validación. Cuando un usuario técnico inicia una conexión SSH hacia el sistema, esta es interceptada por el módulo PAM del gateway, que lanza el proceso de autenticación basado en el serial id propuesto.
- **p2: Roles Based Access Control (RBAC)** — Orquesta la lógica de seguridad. Una vez recibido el intento de acceso, este módulo se comunica con los servicios encargados de verificar la identidad del usuario y validar sus permisos.
- **p3: Authentication & Authorization Service** — Busca el certificado correspondiente, evalúa la validez y consulta si el rol asignado al usuario permite acceder al recurso solicitado.
- **p4: PKI - EJBCA** — Provee los servicios de infraestructura de clave pública (PKI), como la emisión, revocación y validación de certificados digitales. Es una autoridad confiable utilizada tanto para autenticación como autorización.
- **p5: Gateway (Bastion)** — Nodo central que media todo el acceso hacia la red interna. Solo si el proceso de autenticación y autorización tiene éxito, permite al

usuario establecer una nueva conexión SSH hacia el host destino.

- **p6: PAM2** — Segundo módulo PAM ubicado en los servidores destino. Realiza una segunda validación, esta vez reforzando la política de acceso aplicada en el Bastion y verificando condiciones particulares del host.
- **p7: Hosts (Destino N)** — Son los servidores virtuales sobre los que el usuario técnico desea operar. El acceso efectivo solo ocurre si ambas etapas de validación fueron satisfactorias.
- **p8: Accounting & Roles Config (ARCenter)** — Consolida dos funciones críticas: i) configurar y distribuir roles de acceso, y ii) registrar los eventos generados por el sistema, incluyendo accesos exitosos y fallidos.
- **p9: Ansible** — Plataforma de automatización que aplica las configuraciones generadas por ARCenter en los diferentes componentes del sistema, asegurando consistencia y trazabilidad.
- **p10: Elasticsearch** — Almacén central de registros y métricas. Recibe información desde ARCenter y desde los módulos PAM distribuidos en el sistema.
- **p11: Kibana** — Interfaz de visualización que permite consultar y analizar los logs y métricas recolectados, brindando soporte a la auditoría del sistema.

Este modelo interno permite visualizar claramente el flujo completo de acceso, desde la autenticación inicial hasta la ejecución de tareas, y proporciona un marco estructurado para la auditoría y trazabilidad de los eventos de seguridad.

4.1.2. Modelos Dinámicos

Los modelos dinámicos explican el comportamiento del sistema en tiempo de ejecución. Se incluyen:

Diagrama de Casos de Uso

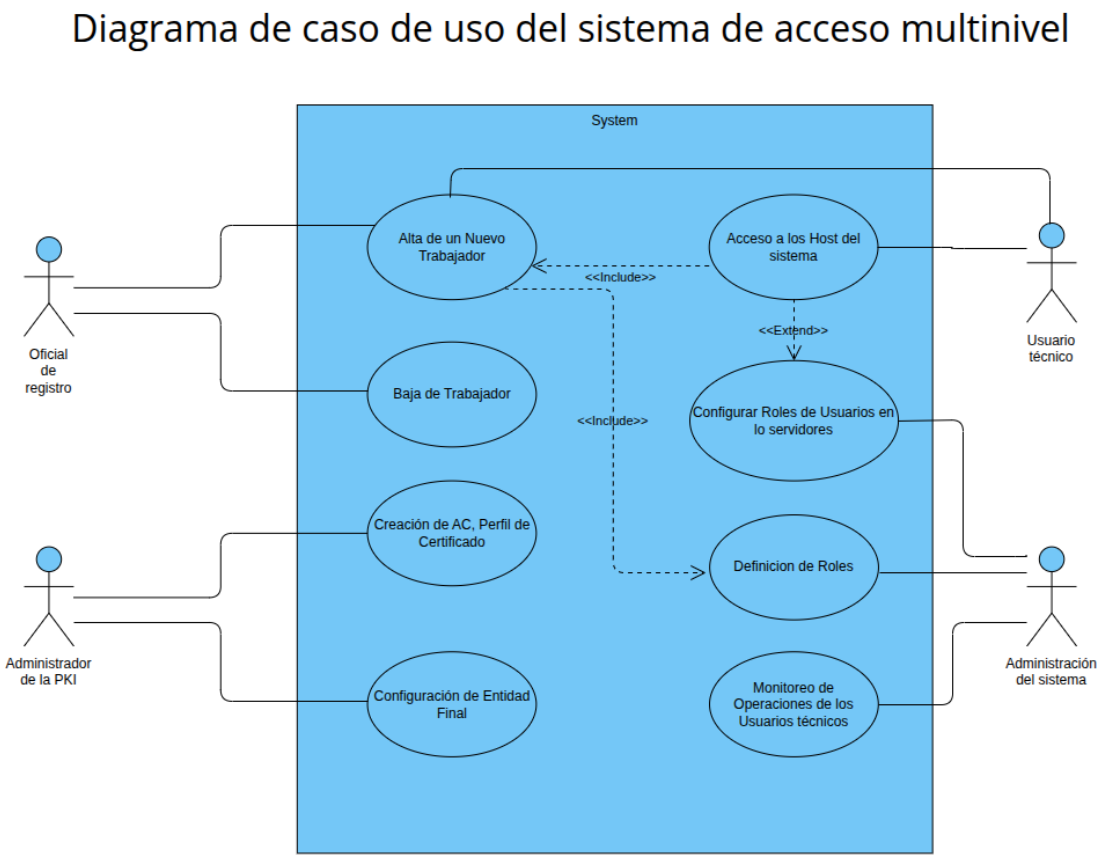

Figura 4.3: Diagrama de caso de uso del sistema de acceso multinivel

La Figura 4.3 presenta el diagrama de casos de uso del sistema de acceso multinivel, modelando las interacciones principales entre los actores y las funcionalidades previstas.

Actores del sistema:

- **Oficial de Registro (OR)**: Responsable de la gestión del ciclo de vida de los trabajadores, incluyendo su alta y baja. Tiene un rol clave en la incorporación inicial de los usuarios técnicos.
- **Administrador de la PKI**: Encargado de configurar la infraestructura de clave pública (PKI), incluyendo la creación de autoridades certificadoras (CA), perfiles de certificados y parámetros de las entidades finales.
- **Administración del Sistema**: Actor que define y configura los roles de acceso y políticas de autorización asociadas a los usuarios técnicos dentro de los distintos servidores destino.

- **Usuario Técnico**: Actor final del sistema, que accede a los servidores destino para llevar a cabo sus tareas operativas, autenticándose mediante certificados digitales.

Casos de uso del sistema:

- **Alta de un Nuevo Trabajador**: Proceso que permite registrar e incorporar un nuevo usuario técnico. Incluye la generación de su certificado digital y la posterior configuración de roles.
  - Incluye el caso **Acceso a los Host del sistema**, el cual modela el objetivo final del usuario técnico.
  - Extiende al caso **Configurar Roles de Usuarios en los servidores**, que define los permisos asignados al nuevo usuario.
- **Baja de Trabajador**: Proceso para revocar el acceso de un trabajador, deshabilitando su certificado y eliminando su rol del sistema.
- **Creación de AC, Perfil de Certificado**: Actividad técnica que establece los componentes fundamentales de la PKI necesarios para emitir certificados válidos dentro del sistema.
- **Configuración de Entidad Final**: Define los atributos y políticas asociadas a los certificados emitidos para usuarios finales, como nombre común, rol y dirección de correo electrónico.
- **Definición de Roles**: Caso de uso transversal que permite establecer los niveles de autorización dentro del sistema en función del tipo de tarea que el usuario debe realizar.
- **Monitoreo de Operaciones de los Usuarios Técnicos**: Proceso que permite visualizar y auditar las acciones realizadas por los usuarios técnicos en los hosts destino, con fines de control y trazabilidad.

Este diagrama sintetiza las funcionalidades clave del sistema de acceso multinivel, evidenciando la interacción entre los distintos perfiles de usuarios y la infraestructura de control de acceso basada en certificados digitales y políticas de autorización.

Diagrama de secuencia de configuración de la PKI-EJBCA

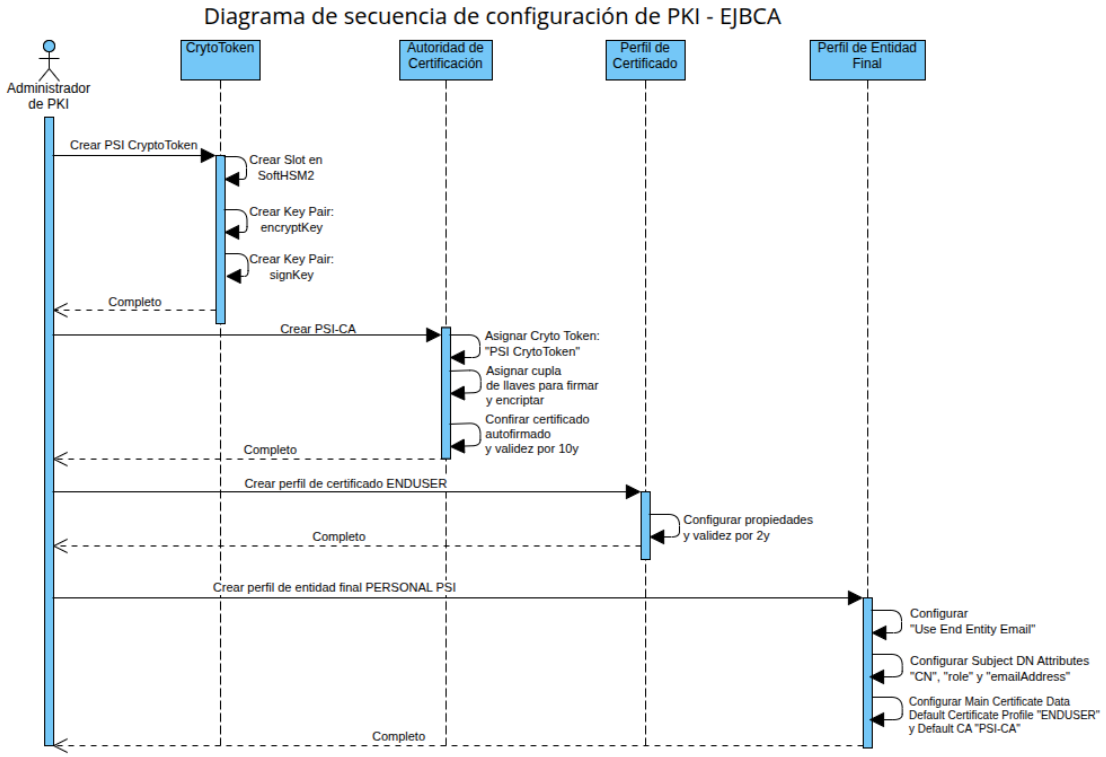

Figura 4.4: Diagrama de Secuencia de Configuración de la PKI en EJBCA

La Figura 4.4 representa el proceso de configuración inicial de la infraestructura de clave pública (PKI) utilizando la plataforma **EJBCA**. Este procedimiento es realizado por el administrador de la PKI y constituye la base para la emisión y gestión de certificados digitales dentro del sistema.

El flujo comprende las siguientes etapas principales:

1. El administrador crea un **CryptoToken** en el módulo `SoftHSM2`, que funciona como un dispositivo criptográfico lógico. Este proceso incluye:
   - Creación de un *slot* seguro.
   - Generación de un par de claves para cifrado (`encryptKey`).
   - Generación de un par de claves para firma digital (`signKey`).
2. Una vez creado el CryptoToken, se procede a la creación de una **Autoridad de Certificación** (CA), en este caso denominada `PSI-CA`, a la cual se le asigna:
   - El CryptoToken previamente creado.
   - Las claves criptográficas para firma y cifrado.
   - Un certificado autofirmado con validez de 10 años.

3. Luego se define un **Perfil de Certificado** para los usuarios finales, denominado `ENDUSER`, configurando sus propiedades y estableciendo una validez de 2 años para los certificados emitidos con dicho perfil.
4. Finalmente, se crea un **Perfil de Entidad Final** (`PERSONAL PSI`) que servirá para identificar a los usuarios técnicos. Este perfil se configura para:
   - Utilizar el perfil de certificado `ENDUSER`.
   - Asociarse por defecto a la CA `PSI-CA`.
   - Incluir atributos del `Distinguished Name` como `CN`, `role` y `emailAddress`.

Este proceso establece la jerarquía de confianza y los parámetros necesarios para la emisión controlada de certificados digitales, asegurando tanto la integridad criptográfica como la correcta asignación de identidades dentro del sistema.

Diagrama de secuencia de alta de usuario técnico

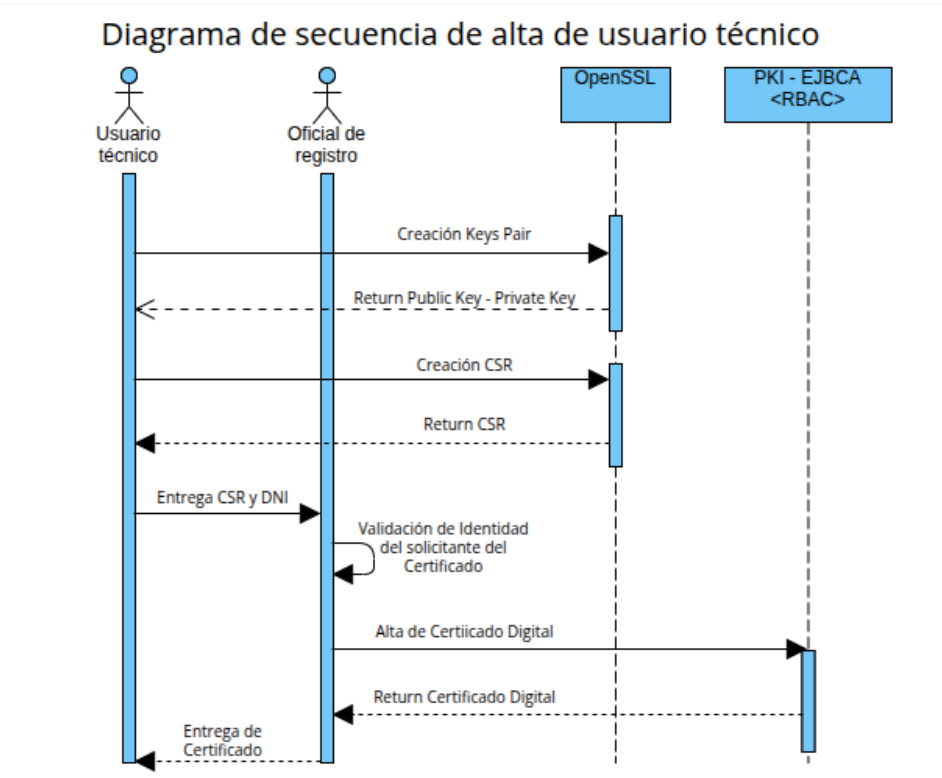

Figura 4.5: Diagrama de secuencia de alta de usuario técnico

La Figura 4.5 representa el proceso de incorporación de un nuevo usuario técnico al sistema, desde la generación de sus credenciales digitales hasta la emisión final del certificado. El procedimiento está guiado por principios de seguridad informática y

gobernado por una infraestructura de clave pública (PKI) que garantiza la autenticidad e integridad de la identidad digital del usuario.

Las entidades participantes son:

- **Usuario Técnico**: Inicia el proceso de generación de credenciales.
- **Oficial de Registro**: Responsable de validar la identidad del usuario y registrar la solicitud en el sistema PKI.
- **OpenSSL**: Herramienta utilizada para la creación de pares de claves y la generación de la CSR (Certificate Signing Request).
- **PKI - EJBCA <RBAC>**: Autoridad Certificadora que emite y firma el certificado digital, integrando la política de roles de acceso.

El flujo de acciones es el siguiente:

1. El usuario técnico genera un par de claves asimétricas (clave pública y privada) utilizando la herramienta OpenSSL.
2. OpenSSL devuelve las claves generadas, que quedan almacenadas localmente por el usuario.
3. Con base en esas claves, se genera una solicitud de firma de certificado (CSR), que contiene la clave pública y datos de identidad.
4. El usuario entrega la CSR junto a su Documento Nacional de Identidad (DNI) al Oficial de Registro.
5. El Oficial de Registro realiza la validación manual de identidad, asegurándose de que la solicitud proviene de un individuo legítimo.
6. Una vez verificada la identidad, el Oficial registra la CSR en el sistema PKI (EJBCA), dando inicio al proceso de emisión del certificado digital.
7. La infraestructura PKI genera el certificado firmado digitalmente, que es devuelto al Oficial de Registro y luego entregado al usuario técnico.

Este proceso integra componentes criptográficos y administrativos que garantizan una gestión segura de identidades. La emisión del certificado digital vincula inequívocamente al usuario con su par de claves, habilitándolo para autenticarse ante los distintos componentes del sistema mediante mecanismos de autenticación robusta basados en certificados X.509.

Diagrama de secuencia de acceso de usuario técnico

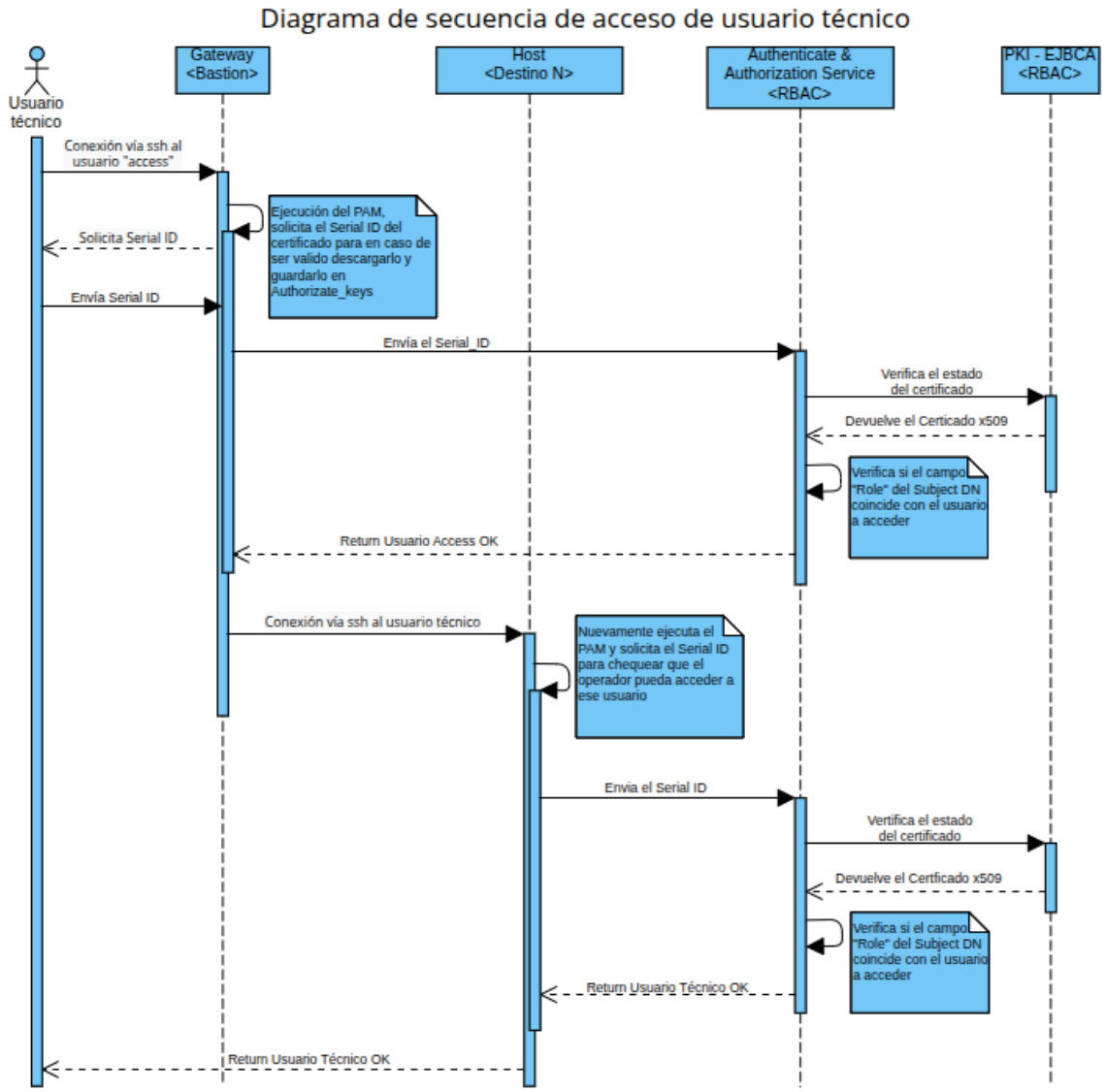

Figura 4.6: Diagrama de secuencia de acceso de usuario técnico

La Figura 4.6 representa el flujo completo de autenticación y autorización de un usuario técnico que accede a un servidor destino a través de una arquitectura de acceso seguro basada en certificados digitales X.509. El proceso está dividido en dos etapas principales: acceso inicial al Gateway y validación posterior en el Host destino.

Las entidades involucradas son:

- **Usuario Técnico**: Operador que inicia la conexión SSH hacia los servidores del entorno protegido.
- **Gateway (Bastion)**: Primer punto de entrada a la red interna, encargado de validar el certificado del usuario antes de permitir su acceso.

- **Host (Destino N)**: Servidor final al cual el usuario desea acceder para ejecutar sus tareas técnicas.
- **Authentication & Authorization Service**: Servicio encargado de validar el certificado digital, verificar el estado del mismo y comprobar los permisos asociados al rol.
- **PKI - EJBCA**: Autoridad Certificadora que emite y mantiene los certificados digitales utilizados por el sistema.

Etapa 1 — Acceso al Gateway:

1. El usuario se conecta por SSH al Gateway utilizando un usuario técnico compartido (por ejemplo, `access`).
2. Se ejecuta un módulo **PAM** que solicita el *Serial ID* del certificado digital presentado.
3. El Gateway consulta al servicio de autenticación para:
   - Verificar el estado del certificado (revocación, expiración, validez).
   - Consultar la PKI para obtener la clave pública.
   - Validar que el `Subject DN` y el atributo `Role` del certificado coincidan con el usuario técnico que se intenta acceder.
4. Si la verificación es exitosa, el certificado es registrado en el archivo `authorized_keys` y el acceso al Gateway es concedido.

Etapa 2 — Acceso al Host destino:

1. Desde el Gateway, el usuario establece una nueva conexión SSH hacia el Host destino.
2. Se ejecuta nuevamente el módulo **PAM**, repitiendo el proceso de validación del certificado y verificación del atributo `Role`.
3. El servicio de autenticación consulta nuevamente el estado del certificado a través de la PKI.
4. Si las condiciones son válidas, el acceso al servidor es concedido y se completa el proceso.

Este flujo garantiza una autenticación sólida en múltiples niveles y refuerza el principio de menor privilegio, al exigir validaciones independientes en cada punto del sistema. El uso del mismo certificado X.509 a lo largo del proceso permite una trazabilidad total, mientras que la integración con los módulos PAM asegura compatibilidad nativa con sistemas UNIX/Linux.

Diagrama de secuencia para la configuración de nuevos usuarios y roles del sistema

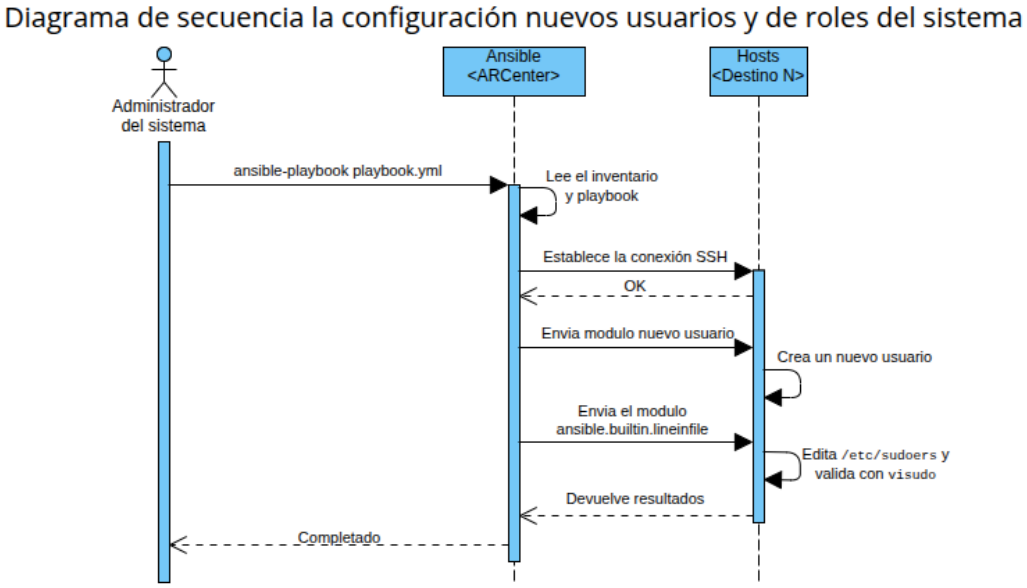

Figura 4.7: Diagrama de secuencia para la configuración de nuevos usuarios y roles

La Figura 4.7 representa el proceso automatizado de aprovisionamiento de usuarios técnicos y la asignación de privilegios en los servidores destino. Este flujo es coordinado por el administrador del sistema mediante la herramienta de automatización **Ansible**, integrada como parte del subsistema **ARCenter**.

El procedimiento comprende las siguientes etapas:

1. El administrador ejecuta un *playbook* de Ansible que contiene las tareas necesarias para la creación de usuarios y la asignación de privilegios administrativos.
2. Ansible interpreta tanto el inventario de hosts como las instrucciones del *playbook*, e inicia una conexión SSH hacia cada servidor destino.
3. Una vez establecida la conexión, Ansible ejecuta un módulo específico para la creación de un nuevo usuario en el sistema operativo.
4. Luego, se aplica el módulo `ansible.builtin.lineinfile`, el cual permite modificar el archivo `/etc/sudoers` de forma segura, insertando las reglas necesarias para otorgar privilegios al nuevo usuario. Esta operación se valida utilizando la herramienta `visudo`, lo cual previene errores de sintaxis que podrían comprometer la administración del sistema.
5. Ansible devuelve un informe detallado sobre el estado de cada operación ejecutada, permitiendo al administrador confirmar que el procedimiento se completó correctamente o intervenir en caso de fallos.

6. Finalmente, el sistema notifica la finalización del proceso, asegurando la trazabilidad de los cambios realizados.

Este diagrama refleja la importancia de la automatización en entornos seguros y escalables, ya que permite la administración centralizada, reproducible y confiable de los usuarios y sus permisos, reduciendo al mínimo el riesgo de errores manuales y garantizando el cumplimiento de políticas de seguridad.

Diagrama de secuencia del almacenamiento y visualización de logs

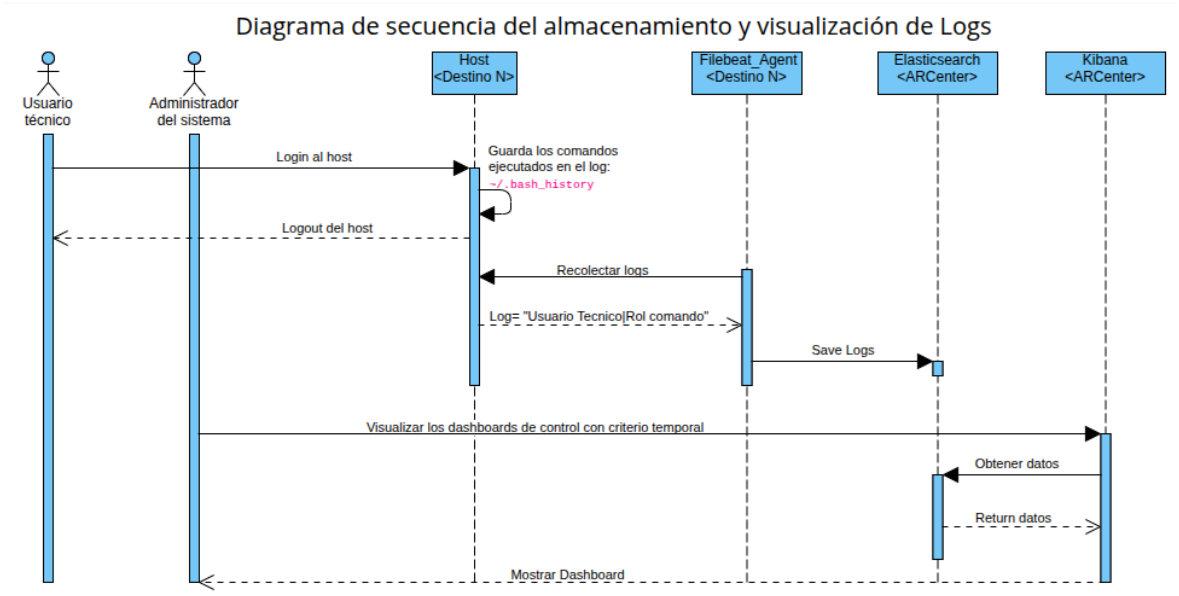

Figura 4.8: Diagrama de secuencia del almacenamiento y visualización de logs

La Figura 4.8 representa el proceso completo de auditoría técnica mediante la recolección, almacenamiento centralizado y visualización de logs generados durante la actividad de los usuarios técnicos en los servidores destino.

Este flujo involucra las siguientes entidades principales: el **Usuario Técnico**, el **Administrador del Sistema**, los **Hosts (Destino N)**, el agente recolector **Filebeat**, y los componentes de visualización y almacenamiento del subsistema **ARCenter** (*Elasticsearch* y *Kibana*).

El procedimiento se desarrolla de la siguiente forma:

1. El usuario técnico inicia sesión (*login*) en un host destino y ejecuta comandos de administración o mantenimiento. Cada instrucción queda registrada automáticamente en el archivo `~/.bash_history` del sistema operativo.
2. Al finalizar la sesión (*logout*), el historial de comandos queda disponible para su análisis posterior. Se espera que los logs incluyan una referencia explícita al usuario y al rol desde el cual se ejecutó el comando.

3. El agente **Filebeat**, previamente instalado y configurado en los servidores destino, recolecta estos logs y los reenvía al motor de búsqueda **Elasticsearch**, que actúa como base de datos centralizada para eventos de auditoría. El software monitorea en tiempo real los archivos de log, usando un harvester que lee las nuevas líneas a medida que se escriben. Apenas detecta una nueva entrada en el log, la envía al spooler interno de Filebeat. Por defecto, Filebeat envía los datos a Elasticsearch en lotes de hasta 2048 eventos o cada 1 segundo, lo que ocurra primero.
4. Una vez almacenados, los datos son indexados y estructurados para permitir su consulta eficiente. Esto facilita la trazabilidad de eventos críticos y la reconstrucción de sesiones.
5. El administrador puede acceder a la herramienta **Kibana**, conectada a Elasticsearch, para generar dashboards dinámicos que permiten visualizar el historial de comandos por usuario, por host o por rangos de tiempo.
6. Finalmente, el administrador puede realizar auditoría forense, identificar comportamientos anómalos o evaluar la adherencia a políticas de uso establecidas.

Este mecanismo integral fortalece la trazabilidad y la rendición de cuentas en el entorno de servidores virtuales, alineándose con las mejores prácticas de seguridad y requisitos normativos en materia de monitoreo y auditoría.

4.2. Topología de Red

Topología lógica del sistema propuesto

La Figura 4.9 ilustra la topología lógica de red correspondiente al sistema multicliente propuesto, desplegado sobre la infraestructura virtualizada de VMware ESXi. El diseño busca lograr una arquitectura segura, segmentada y con rutas de acceso controladas.

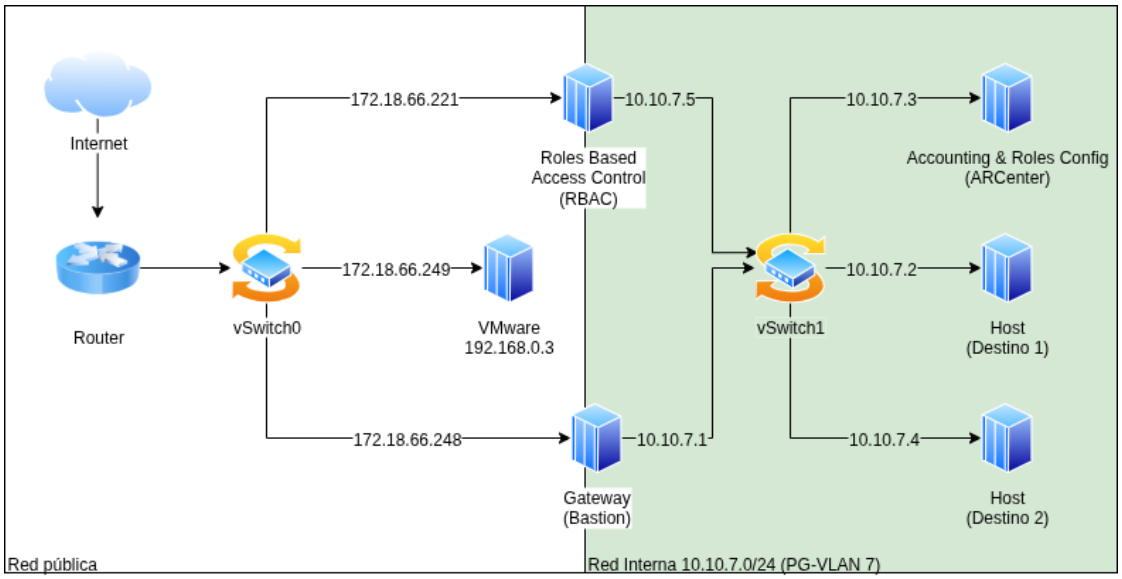

Figura 4.9: Topología lógica de red del sistema.

El esquema contempla el uso de dos **conmutadores virtuales (vSwitch)** diferenciados por propósito funcional y nivel de exposición:

- **vSwitch0 — Red pública**: conecta las máquinas virtuales expuestas hacia el exterior (Internet) mediante el router y provee conectividad con el host físico donde corre el hipervisor ESXi (`192.168.0.3`). Dentro de esta red se ubican dos componentes accesibles desde el exterior:
  - **Roles Based Access Control (RBAC)** — EJBCA, accesible desde la IP pública `172.18.66.221`.
  - **Gateway (Bastion)**, disponible en `172.18.66.248`, quien actúa como punto único de acceso a la red interna.
- **vSwitch1 — Red interna (VLAN 7)**: corresponde a la red privada identificada con el prefijo `10.10.7.0/24`, utilizada exclusivamente para la comunicación interna entre servidores virtuales. Esta red es inaccesible directamente desde el exterior y solo es alcanzable a través del Bastion, que cumple funciones de enrutamiento. Los servidores conectados a esta red son:
  - **Accounting & Roles Config (ARCenter)** — IP: `10.10.7.3`
  - **Host Destino 1** — IP: `10.10.7.2`
  - **Host Destino 2** — IP: `10.10.7.4`
  - **RBAC (EJBCA)**, con doble interfaz de red: pública y privada (`10.10.7.5`)

El servidor Bastion, al estar conectado a ambas redes, actúa como pasarela segura entre los dos dominios de red. Toda comunicación entrante hacia los servidores internos pasa obligatoriamente por este nodo, lo que permite aplicar políticas de seguridad como reglas de firewall, monitoreo, registro de sesiones y control de acceso.

Este diseño está alineado con el principio de *defensa en profundidad*, favoreciendo la segmentación lógica, el aislamiento de servicios sensibles y la supervisión centralizada del tráfico, requisitos clave en entornos que manejan credenciales, certificados y configuraciones críticas.

4.3. Política de roles del sistema

Con el fin de garantizar un control de acceso granular y seguro sobre los servidores administrados, se definió una política de roles basada en los distintos perfiles técnicos que interactúan con el sistema. Esta política permite limitar el conjunto de comandos disponibles para cada usuario, así como sus privilegios dentro del entorno operativo. Los roles definidos son los siguientes:

Rol: Access

Este rol corresponde al nivel mínimo de acceso permitido. Los usuarios con rol **Access**:

- No tienen acceso a una consola interactiva en el servidor bastión.
- Solo pueden utilizar el gateway como punto de acceso hacia otros servidores autorizados.
- No pueden ejecutar comandos en el bastión.

Este rol está presente en todos los certificados de los usuarios técnicos y garantiza que el servidor bastión actúe estrictamente como un canal de conexión seguro, sin exposición innecesaria al sistema base.

Rol: Sysadmin

El rol **Sysadmin** está destinado a administradores del sistema que requieren acceso completo para tareas de mantenimiento, configuración y resolución de problemas. Los usuarios con este rol:

- No cuenta con privilegios de superusuario.
- Son responsables de la administración general de los servidores.
- Puede ejecutar comandos relacionados con servicios del host, incluyendo:
  - `useradd`
  - `userdel`
  - `usermod`
  - `passwd`
  - `firewall-cmd`

  - `ufw`
  - `rsync`
  - `htop`
  - `systemctl`
  - `mkdir`
  - `chown`
  - `chmod`
  - `su`

Rol: Databases

El rol **Databases** corresponde a usuarios responsables de la gestión de bases de datos. Este perfil:

- No cuenta con privilegios de superusuario.
- Puede ejecutar comandos relacionados con servicios de bases de datos, incluyendo:
  - `psql`
  - `mysql`
  - `mysqldump`
  - `mysqladmin`
  - `apt update mysql-servers`
  - `systemctl` (limitado a servicios de base de datos)
  - `postgresql`

Este rol permite tareas de mantenimiento y administración de bases de datos, sin comprometer el resto del sistema.

Rol: Devops

El rol **Devops** está orientado a personal encargado de la infraestructura como código y del despliegue de servicios. Los usuarios con este rol:

- No poseen permisos de superusuario.
- Están autorizados a ejecutar comandos clave para la gestión de servicios y contenedores, incluyendo:
  - `docker`
  - `docker-compose`

- `kubectl`
- `systemctl`
- `journalctl`
- `ansible-playbook`
- `systemctl restart apache2`
- `git`

Este rol permite intervenir sobre el ciclo de vida de servicios, contenedores y redes, sin otorgar control total sobre el sistema operativo.

La asignación de roles se realiza al momento de emitir el certificado digital, lo que permite integrar esta política directamente en el mecanismo de autenticación. Esta integración asegura que cada acceso al sistema se ajuste automáticamente a los permisos definidos por su rol.

5. Selección de Componentes

Este capítulo detalla los componentes de hardware y software seleccionados para llevar a cabo la implementación del sistema de acceso multinivel propuesto. La elección de cada elemento respondió a criterios de compatibilidad, estabilidad, soporte comunitario y alineación con los objetivos de seguridad, trazabilidad y eficiencia definidos en los capítulos anteriores.

5.1. Hardware

5.1.1. Servidor físico principal

Para el desarrollo, despliegue y evaluación del sistema se utilizó un servidor físico proporcionado por el Laboratorio de Redes y Ciberseguridad (LARYC). El equipo correspondió al modelo **Dell PowerEdge R210 II**, fabricado por Dell Inc., el cual se encontró montado en un entorno controlado y dedicado a investigación y desarrollo de soluciones en ciberseguridad.

Las especificaciones técnicas del servidor fueron las siguientes:

- **Procesador:** Intel(R) Xeon(R) E3-1220 a 3.10 GHz (4 núcleos físicos, sin Hyper-Threading).
- **Memoria RAM:** 8 GB DDR3 (7,99 GB útiles).
- **Almacenamiento:** Disco duro SATA de 1 TB.
- **Plataforma de virtualización:** VMware ESXi.

El servidor contó con soporte para virtualización por hardware (Intel VT-x), lo cual permitió ejecutar múltiples máquinas virtuales con independencia y seguridad, condición necesaria para separar lógicamente los componentes del sistema en entornos diferenciados.

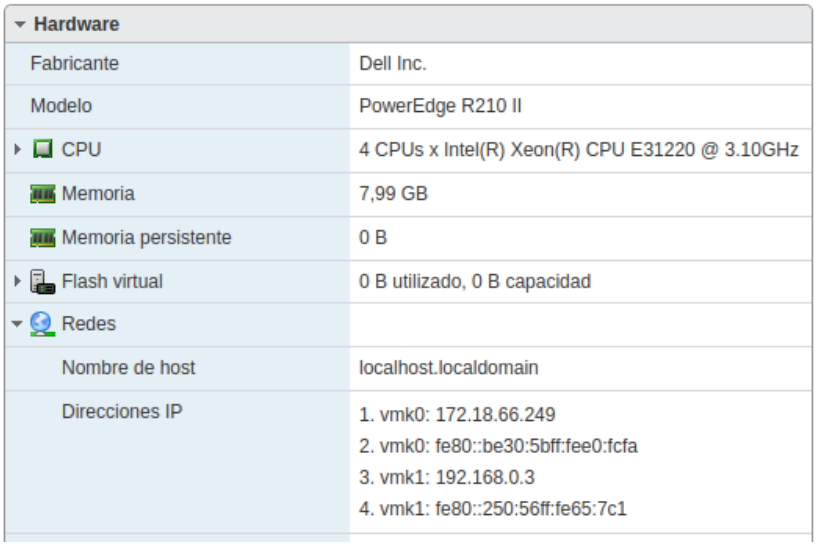

Figura 5.1: Características del servidor Dell PowerEdge R210 II

5.2. Software

Para la implementación del sistema se seleccionaron tecnologías de código abierto y/o licencia gratuita, todas ellas ampliamente utilizadas en la industria y con soporte comunitario activo. A continuación se detallan los principales componentes de software adoptados.

5.2.1. Plataforma de virtualización

Se empleó **VMware ESXi** versión `ESXi-6.7.0-8169922-standard` como hipervisor para la administración de máquinas virtuales. Esta versión permitió gestionar de forma eficiente los recursos físicos del servidor y proporcionar entornos virtuales seguros y aislados para cada subsistema del diseño.

5.2.2. Sistema operativo base

Todas las máquinas virtuales fueron desplegadas con el sistema operativo **Ubuntu 20.04.6 LTS**, debido a su estabilidad, soporte extendido y compatibilidad con herramientas de administración, monitoreo y desarrollo. La elección de una versión LTS garantizó actualizaciones de seguridad por al menos cinco años, lo cual fue crítico en entornos con requerimientos de alta disponibilidad.

5.2.3. Infraestructura PKI: EJBCA

Para implementar la infraestructura de clave pública (PKI), se utilizó **EJBCA Community Edition**, una solución open source robusta y escalable para la gestión de

certificados digitales. La instancia de EJBCA fue desplegada mediante el contenedor **Bitnami EJBCA Docker**, lo cual permitió simplificar su configuración, integración y mantenimiento en un entorno virtualizado.

El contenedor Bitnami incluyó soporte integrado para los siguientes servicios:

- Interfaz de administración web (CA y RA).
- Protocolos de inscripción y validación (CMP, OCSP, CRL).
- API REST para automatización y gestión remota.
- Módulo PKCS11 embebido (via SoftHSM2).

La utilización de esta solución permitió cumplir con los requerimientos funcionales del sistema sin necesidad de implementar múltiples nodos, dado que EJBCA centralizó los roles de Autoridad de Certificación, Autoridad de Registro y Autoridad de Validación.

5.2.4. Autenticación para el acceso a servidores

Se optó por utilizar el software libre desarrollado por Klincovisky-Perez(2025) [54] para gestionar el acceso a los servidores, ya que incorpora certificados digitales X.509 como mecanismo de autenticación y, al estar implementado en Python, ofrece la flexibilidad necesaria para introducir mecanismos de control de acceso basados en roles.

5.2.5. Autorización multinivel y su propagación: Ansible

Se empleó **Ansible** como herramienta de automatización para la gestión de usuarios, la configuración de accesos y la definición de políticas de control de roles. Su arquitectura sin agentes y su compatibilidad nativa con SSH lo hicieron ideal para entornos distribuidos y para la gestión declarativa de estados del sistema.

5.2.6. Auditoria y monitoreo: Elastic Stack

Para el monitoreo, recolección y análisis de eventos de acceso se utilizó **Elastic Stack 7.17.26**, una pila de software especializada en observabilidad, búsqueda y visualización de datos.

- **Elasticsearch 7.17.26**: motor de búsqueda y análisis distribuido, utilizado como base de datos de eventos.
- **Kibana 7.17.26**: interfaz web que permitió visualizar logs, construir dashboards e identificar patrones.
- **Filebeat 7.17.26**: agente ligero de recolección de logs, instalado en los servidores destino.

La elección de esta versión específica respondió a su condición de *Long-Term Support (LTS)*, lo que aseguró parches de seguridad y compatibilidad a largo plazo sin necesidad de actualizaciones disruptivas.

5.2.7. Otras herramientas complementarias

- **Docker y Docker Compose:** utilizados para contenerizar los servicios del sistema (EJBCA, ARCenter, Elastic Stack).
- **OpenSSH:** servidor y cliente de conexiones seguras, base del sistema de acceso técnico a los hosts.
- **SoftHSM2:** HSM virtual basado en software, utilizado para simular el almacenamiento seguro de claves privadas en el entorno de pruebas.

En conjunto, los componentes seleccionados permitieron implementar una solución segura, auditable y flexible, cumpliendo con los objetivos definidos en el diseño general del sistema propuesto.

6. Actividades Desarrolladas

6.1. Iteración I: Configuración de máquinas virtuales en el servidor

Durante esta primera iteración se llevó a cabo la preparación del entorno de virtualización en el servidor físico, instalando el hipervisor VMware ESXi y creando las máquinas virtuales necesarias para la arquitectura propuesta. Esta etapa incluye las siguientes secciones:

- Instalación del hipervisor VMware ESXi.
- Creación y configuración de las máquinas virtuales.
- Configuración de VLANs y adaptadores de red.

Si bien uno de los requerimientos no funcionales establece que las herramientas empleadas deben ser de código abierto, VMware ESXi es una solución con licencia comercial. No obstante, existe una versión gratuita, VMware vSphere Hypervisor, que ofrece todas las funcionalidades necesarias para su uso sin inconvenientes operativos.

En el caso particular de la Prosecretaría de Informática de la Universidad Nacional de Córdoba, esta plataforma ya se encuentra instalada y en producción.

La elección de VMware responde a la necesidad de trabajar sobre una infraestructura de virtualización estable, robusta. Cabe destacar que el alcance del presente trabajo se centra en el diseño e implementación de un modelo de autenticación y control de acceso basado en tecnologías open source , como EJBCA, Filebeat, Ansible y OpenSSH, siendo la capa de virtualización únicamente el soporte de base, sin comprometer los principios de apertura y transparencia que guían el desarrollo propuesto.

6.1.1. Instalación de VMware ESXi

Se empleó VMware ESXi como plataforma de virtualización para replicar un entorno controlado, simple y funcional, con múltiples servidores virtuales. La instalación se realizó sobre un servidor *Dell PowerEdge R210 II* con soporte para virtualización por hardware (Intel VT-x).

El proceso comenzó con la descarga de la imagen ISO de VMware ESXi desde el sitio oficial. Posteriormente, se creó una unidad USB booteable con la herramienta *Rufus*, utilizando un dispositivo de 8GB formateado en FAT32 y esquema de partición MBR.

A continuación, se accedió a la BIOS del servidor (tecla F11 durante el arranque) y se seleccionó la unidad USB desde el menú *One-shot BIOS Boot Menu*. Esto permitió iniciar el instalador del hipervisor, como se muestra en la Figura 6.1.

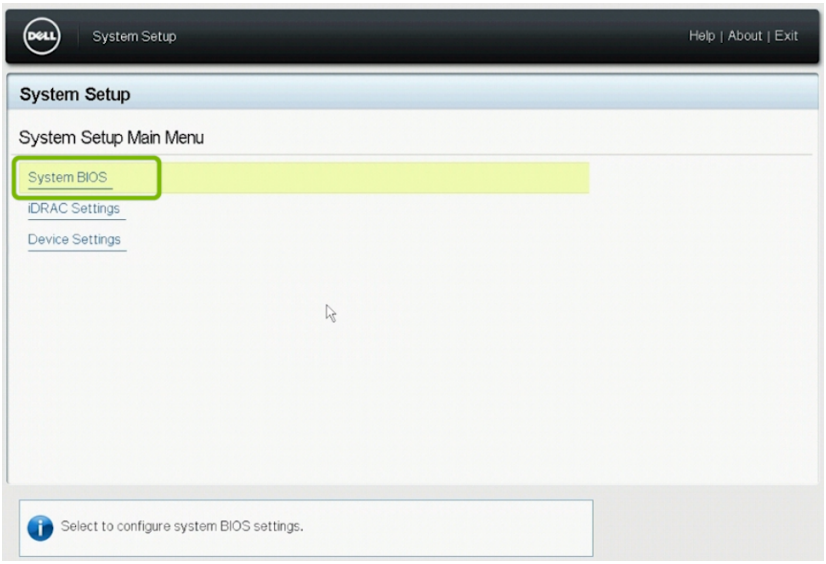

Figura 6.1: Panel de configuración del sistema del servidor

Se utilizó el asistente de instalación para definir las opciones básicas del sistema. Una vez configurados el disco, el idioma y la contraseña del usuario root, se completó la instalación y se reinició el equipo para aplicar los ajustes.

En el primer arranque de ESXi, se accedió a la configuración inicial presionando F2. Allí se definieron los parámetros de red del host desde la opción *Configure Management Network*. Se seleccionó el adaptador de red adecuado y se configuraron manualmente la dirección IPv4, máscara de red y gateway.

Figura 6.2: Opciones de configuración en VMware

Una vez finalizada la configuración de red, se accedió a la interfaz web del hipervisor mediante un navegador, ingresando la dirección IP del host. Desde esta consola se gestionaron todas las máquinas virtuales del sistema (Figura 6.3).

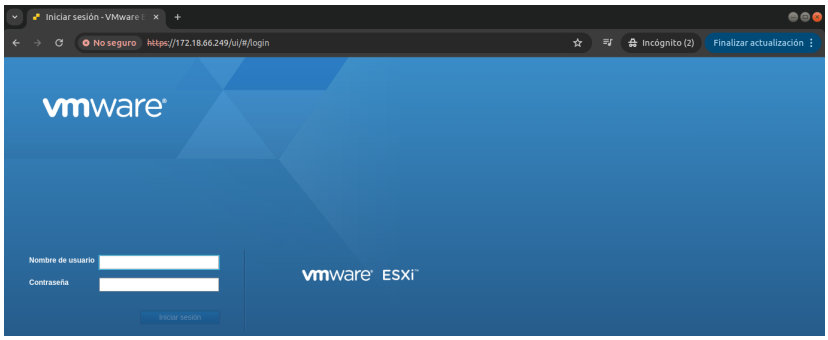

Figura 6.3: Inicio de sesión de VMware

6.1.2. Creación de máquinas virtuales

Cada servidor virtual fue creado a partir de la imagen ISO de Ubuntu Server 20.04, la cual fue instalada y configurada con parámetros básicos.

En primer lugar, se cargó la ISO del sistema operativo al almacenamiento del host, dentro del explorador de almacenamiento de datos (Figura 6.4).

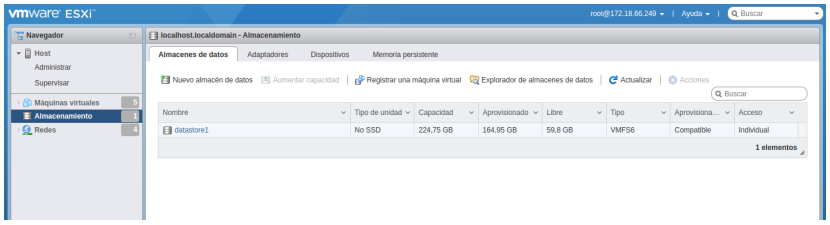

Figura 6.4: Almacenamiento de la ISO de un sistema operativo

Luego se inició el asistente de creación de máquina virtual, eligiendo la opción *Crear una nueva máquina virtual*. Se especificaron parámetros como nombre, compatibilidad con ESXi 6.7, tipo y versión de sistema operativo (Ubuntu 64 bits), almacenamiento y configuración de recursos (CPU, RAM, disco).

Durante la configuración de hardware, se asignó un adaptador de red y se adjuntó la imagen ISO cargada previamente desde la unidad virtual de CD/DVD.

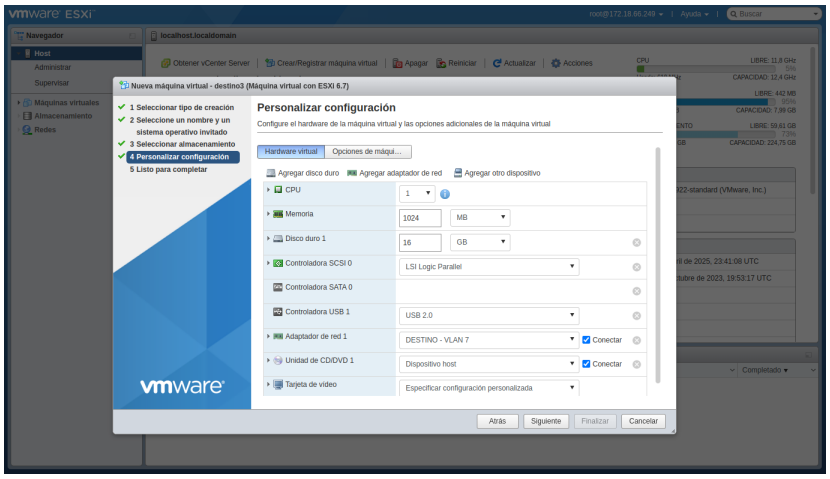

Figura 6.5: Pantalla de configuración de recursos de un servidor virtual

Una vez creada, se encendió la máquina virtual para comenzar la instalación del sistema operativo. Durante el proceso, se seleccionó el idioma, el diseño de teclado y se aplicaron configuraciones por defecto para red, proxy, y almacenamiento (*Use an entire disk*).

En la sección *Profile Setup*, se definió el nombre del host, usuario administrador y contraseña. A los fines de poder desarrollar las pruebas correspondientes, se definió como criterio que cada host tuviera un usuario y una contraseña iguales al nombre del propio host. Se habilitó la instalación de *OpenSSH Server* y se omitieron paquetes adicionales.

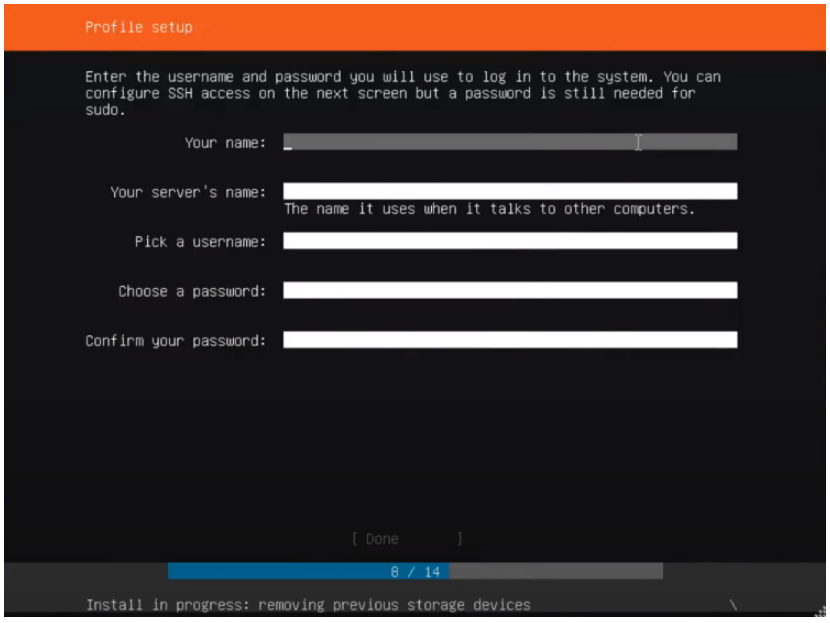

Figura 6.6: Configuración del nombre del servidor y del usuario de sistema

Finalizada la instalación, se reinició el sistema y se accedió con las credenciales configuradas. Este procedimiento fue repetido para cada uno de los servidores requeridos por la arquitectura del sistema.

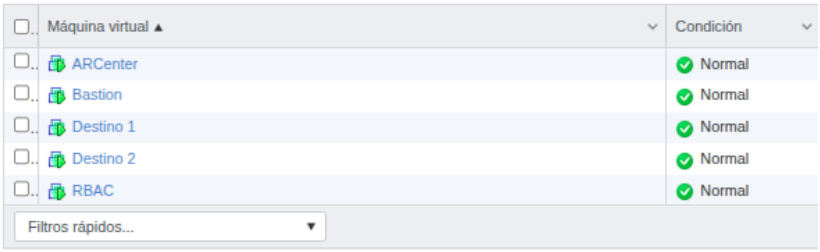

Figura 6.7: Creación de las máquinas virtuales

Los servidores creados en esta iteración fueron:

- **RBAC:** servidor central de autorización basado en roles.
- **Bastión:** punto de acceso intermedio seguro.
- **Destino 1 y Destino 2:** servidores de aplicación simulada.
- **ARCenter:** componente de monitoreo y auditoría centralizada.

6.1.3. Configuración de VLAN

La red del entorno virtualizado fue configurada mediante una única VLAN con identificador **10**, que permite interconectar de forma segura y controlada a todos los servidores virtuales desplegados en VMware ESXi. Esta decisión simplifica la topología de red al mismo tiempo que garantiza una comunicación eficaz entre los distintos componentes del sistema.

La configuración se realizó accediendo a la interfaz de administración de ESXi, en la sección *Networking*. Allí, se definió un único *Port Group*, al cual se le asignó el `VLAN ID 10`. Este grupo fue creado sobre un *vSwitch* virtual asociado a la interfaz de red física del servidor.

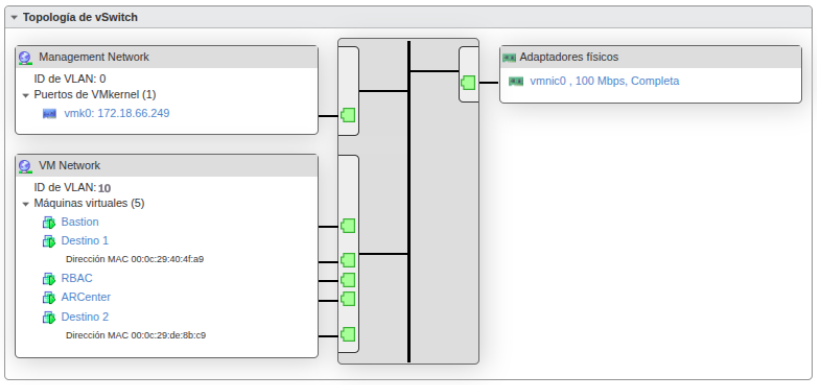

Figura 6.8: Port Group configurado con VLAN ID 10

Durante la configuración de cada máquina virtual, se seleccionó este *Port Group* como interfaz de red, permitiendo que todos los hosts compartan el mismo segmento

de red. Esta configuración resultó suficiente para el entorno de pruebas, ya que no se requería un aislamiento estricto entre componentes, y permitía el monitoreo y control centralizado desde el servidor Bastión.

En futuras iteraciones, si se busca segmentar el tráfico por funciones o niveles de seguridad, será posible expandir esta configuración utilizando múltiples VLANs. Sin embargo, para esta primera versión del sistema, la utilización de una única VLAN permitió validar el modelo propuesto de forma eficiente y con menor complejidad de administración.

6.1.4. Configuraciones no implementadas

Limitaciones del alcance en Hardening y Protección

Es importante señalar que el hardening de los sistemas operativos y servicios involucrados no fue abordado en este trabajo, ya que quedó fuera del alcance del proyecto. Las tareas de aseguramiento del sistema, como la desactivación de servicios innecesarios, el refuerzo de políticas de seguridad, el control detallado de puertos y protocolos, y la aplicación de configuraciones avanzadas de seguridad en el sistema operativo, son fundamentales para la protección integral de los entornos informáticos. Sin embargo, este proyecto se centró específicamente en el diseño e implementación de un modelo de seguridad multinivel basado en autenticación y accounting, apoyado en el uso de certificados digitales X.509, y en la automatización de configuraciones y registros de acceso. La incorporación de prácticas de hardening se considera una etapa complementaria y recomendada para trabajos futuros o para la implementación final del sistema en producción.

6.2. Iteración II: Despliegue y configuración de la Infraestructura de Clave Pública

Esta segunda iteración tuvo como objetivo validar los requerimientos funcionales RF1 y RF2, los cuales están vinculados a la emisión de certificados digitales X.509 y a la autenticación de usuarios basada en roles.

Para lograr una gestión centralizada del acceso a los recursos institucionales, se implementó una Infraestructura de Clave Pública (PKI) utilizando el framework de código abierto EJBCA Community, en su versión 8. Esta herramienta permite establecer una PKI propia, robusta y flexible, capaz de administrar el ciclo de vida de certificados digitales de forma segura y eficiente. A través de su uso, es posible controlar con precisión los accesos y roles asignados, garantizando que cada usuario cuente con un certificado digital válido que respalde su identidad y permisos dentro del sistema.

En cumplimiento del requerimiento no funcional RNF9, relacionado con la seguridad del sistema, se incorporaron medidas adicionales para proteger el par de claves criptográficas que da origen a la PKI. En particular, se integró la herramienta SoftHSM2

dentro de la imagen de Docker que aloja la Autoridad Certificadora. Esta herramienta emula un módulo de seguridad de hardware (HSM), lo cual permite almacenar las claves privadas en un cripto-token cifrado dentro del contenedor. Esta implementación fortalece significativamente la confidencialidad e integridad del material criptográfico, reduciendo el riesgo de accesos no autorizados y aumentando el control sobre los activos críticos de la infraestructura de certificación.

Para el despliegue inicial de la infraestructura, se clonó el repositorio del proyecto disponible en la URL:

https://github.com/gerlamberti/PI-lamberti-munoz.git

Este repositorio contiene todos los archivos necesarios para la orquestación automatizada de servicios mediante contenedores Docker.

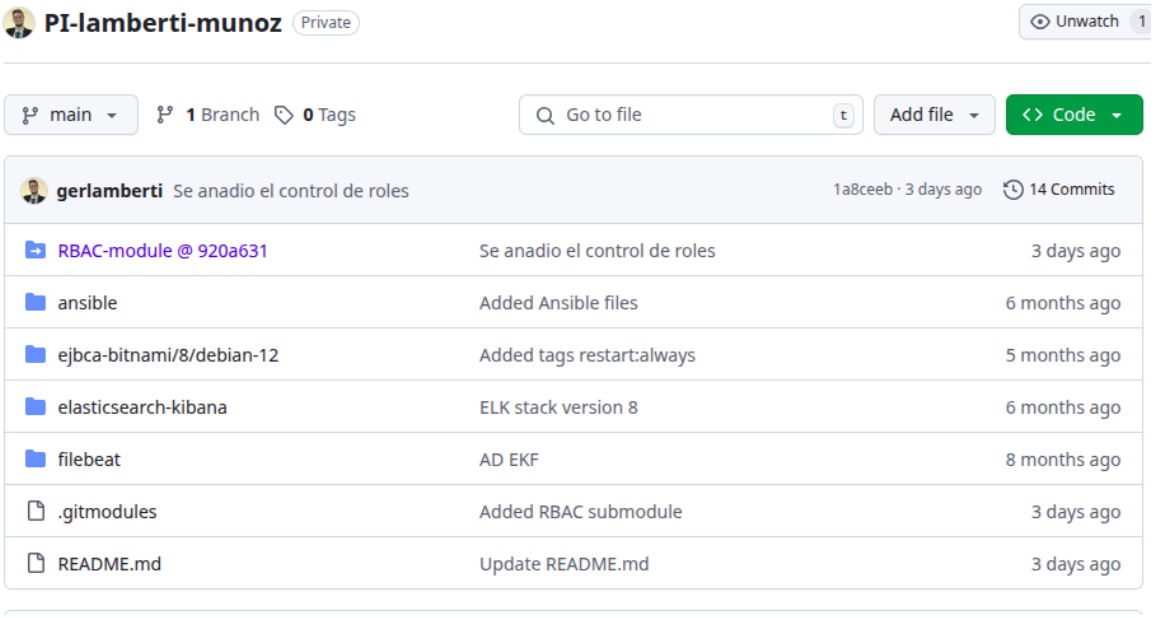

Figura 6.9: Repositorio del proyecto en GitHub

Luego de clonar el repositorio, se ejecutó el siguiente comando para iniciar los servicios definidos en el archivo `docker-compose.yml`:

```
docker compose up -d
```

Este comando puso en funcionamiento los contenedores en segundo plano, creando una red interna entre ellos y exponiendo los puertos requeridos para su interacción. La verificación del estado de los contenedores activos se realizó con:

```
docker ps
```

El resultado evidenció que los servicios se encontraban en ejecución correctamente, como se muestra en la Figura 6.10.

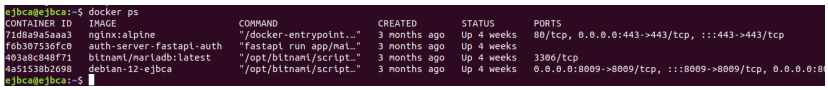

Figura 6.10: Contenedores desplegados por Docker Compose

A continuación, se detalla el propósito de cada uno de los servicios desplegados relacionados a la PKI:

- **auth-server-nginx-1**: Servidor Nginx que actúa como proxy inverso, encaminando las solicitudes HTTPS entrantes hacia el backend de autenticación o al panel de administración de EJBCA, según corresponda.
- **auth-server-fastapi-auth-1**: Servicio desarrollado en FastAPI encargado de gestionar la autenticación y autorización externa, validando certificados y extrayendo atributos relacionados con los roles.
- **debian-12-mariadb-1**: Contenedor que ejecuta MariaDB, base de datos relacional en la que EJBCA almacena la información de certificados, entidades finales y otros metadatos. Se utiliza una imagen oficial de Bitnami.
- **debian-12-ejbca-1**: Contenedor principal donde se ejecuta la aplicación EJBCA. Este servicio expone los puertos 8080 (HTTP), 8443 (HTTPS), 8009 (AJP) y 9990 (administración de JBoss/Wildfly).

Con todos los servicios desplegados correctamente, se accedió a la interfaz de administración web de EJBCA para comenzar con la configuración de la Autoridad Certificadora (CA). Esta interfaz está disponible en la siguiente URL:

`https://172.18.66.221:8443/ejbca/adminweb/`

Configuración de EJBCA

EJBCA proporciona una interfaz de programación de aplicaciones (API REST) para la gestión de certificados digitales, lo que permite interactuar con la plataforma mediante solicitudes HTTP. A través de esta interfaz es posible realizar acciones como emitir certificados, consultar su estado, revocarlos y gestionar entidades finales. Para utilizar esta funcionalidad, es necesario habilitar explícitamente el protocolo de gestión de certificados REST en la configuración del sistema.

La habilitación del protocolo se realizó desde la consola web de administración de EJBCA. Tras iniciar sesión, se accedió a la sección *System Configuration* y dentro de ella a *Protocol Configuration*. En este apartado se verificó que estuvieran activadas las opciones *REST Certificate Management* y *REST V2 Certificate Management*, tal como se ilustra en la Figura 6.11.

Enable or disable protocol access[?]

Protocol	Resource Default URL	Status	Actions
ACME	/ejbca/acme	✗ Unavailable	Enable
Certstore	/ejbca/publicweb/certificates	✓ Enabled	Disable
CMP	/ejbca/publicweb/cmp	✓ Enabled	Disable
CRLstore	/ejbca/publicweb/crls	✓ Enabled	Disable
EST	/.well-known/est	✗ Unavailable	Enable
MSAE	/ejbca/msae	✗ Unavailable	Enable
OCSP	/ejbca/publicweb/status/ocsp	✓ Enabled	Disable
SCEP	/ejbca/publicweb/apply/scep	✓ Enabled	Disable
RA Web	/ejbca/ra	✓ Enabled	Disable
REST CA Management	/ejbca/ejbca-rest-api/v1/ca_management	✗ Unavailable	Enable
REST Certificate Management	/ejbca/ejbca-rest-api/v1/ca /ejbca/ejbca-rest-api/v1/certificate	✓ Enabled	Disable
REST Coap Management	/ejbca/ejbca-rest-api/v1/coap	✗ Unavailable	Enable
REST Crypto Token Management	/ejbca/ejbca-rest-api/v1/cryptotoken	✗ Unavailable	Enable
REST End Entity Management	/ejbca/ejbca-rest-api/v1/endentity	✗ Unavailable	Enable
REST End Entity Management V2	/ejbca/ejbca-rest-api/v2/endentity	✗ Unavailable	Enable
REST Configdump	/ejbca/ejbca-rest-api/v1/configdump	✗ Unavailable	Enable
REST Certificate Management V2	/ejbca/ejbca-rest-api/v2/certificate	✓ Enabled	Disable
REST SSH V1	/ejbca/ejbca-rest-api/v1/ssh	✗ Unavailable	Enable
REST System V1	/ejbca/ejbca-rest-api/v1/system	✗ Unavailable	Enable
Webdist	/ejbca/publicweb/webdist	✓ Enabled	Disable
Web Service	/ejbca/ejbcaws	✓ Enabled	Disable
ITS Certificate Management	/ejbca/its	✓ Enabled	Disable

Figura 6.11: Protocolos habilitados en EJBCA

Además del protocolo REST, se habilitaron otros servicios esenciales para el correcto funcionamiento del sistema de certificación, entre ellos:

- **Certstore**: permite acceder a certificados públicos almacenados en el sistema.
- **CMP (Certificate Management Protocol)**: facilita la gestión automatizada de certificados.
- **CRLstore**: habilita la distribución de listas de certificados revocados (CRL).
- **OCSP (Online Certificate Status Protocol)**: permite validar en línea el estado de un certificado sin necesidad de descargar una CRL.
- **SCEP (Simple Certificate Enrollment Protocol)**: útil para la inscripción de certificados en dispositivos.
- **RA Web**: proporciona una interfaz web para las autoridades de registro.
- **Webdist**: permite la distribución web de certificados y listas CRL.
- **Web Service**: habilita servicios web estándar compatibles con SOAP.

Como parte fundamental de la configuración de la Autoridad Certificadora (CA), se creó un token criptográfico. Este token se utiliza para firmar digitalmente los certificados emitidos, las CRLs generadas por la CA, y las respuestas del servicio OCSP. Su activación se realizó desde la opción *CA Activation* dentro del panel de administración, siendo un paso imprescindible para habilitar las operaciones de firma dentro de la PKI.

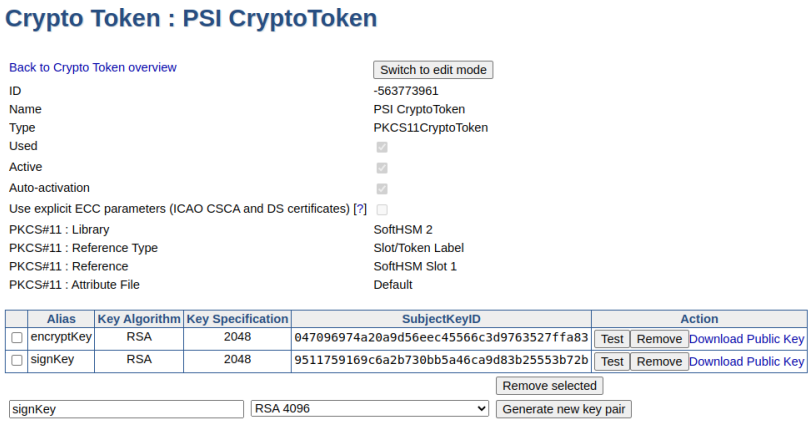

Figura 6.12: Configuración del token criptográfico en EJBCA

A continuación se describen los campos más relevantes que se completaron al momento de configurar el token criptográfico:

- **Name**: Nombre asignado al token criptográfico.
- **Type**: Tipo de token; en este caso, uno asociado a una ranura HSM.
- **Authentication Code**: Código de autenticación (PIN) correspondiente a la ranura PKCS#11 utilizada.
- **Auto-activation**: Si está habilitada, el PIN se almacena cifrado en la base de datos, permitiendo que el token permanezca activo sin intervención manual.
- **Allow export of private keys**: En caso de activarse, permite la exportación del almacén de claves privadas desde el token.

En el caso de que el tipo de token seleccionado sea PKCS#11, EJBCA habilita configuraciones adicionales específicas:

- **PKCS#11 Library**: Ruta absoluta hacia la biblioteca compartida `libsofthsm2.so`.
- **PKCS#11 Reference Type**: Tipo de referencia para seleccionar la ranura del HSM (por número o etiqueta).
- **PKCS#11 Reference**: Valor específico de la ranura, según el tipo de referencia elegido.
- **PKCS#11 Attribute File**: Ruta al archivo de atributos definido para el módulo PKCS#11, generalmente especificado en `conf/web.properties`.

Creación de Autoridades de Certificación (CA)

Tras completar la instalación inicial de EJBCA, se generó automáticamente una Autoridad de Certificación predeterminada denominada **ManagementCA**. Esta entidad actúa como la CA de arranque del sistema y tiene como propósito principal emitir certificados digitales para los administradores que acceden al entorno de gestión.

Creación de una CA raíz

Posteriormente, se procedió a la creación de una CA raíz específica para el sistema propuesto. Este proceso se realizó desde el apartado *Certification Authorities*, dentro de la sección *CA Functions* del panel de administración. Allí se completó el campo *Add CA* con el nombre **PSI-CA**, correspondiente a la nueva Autoridad de Certificación raíz.

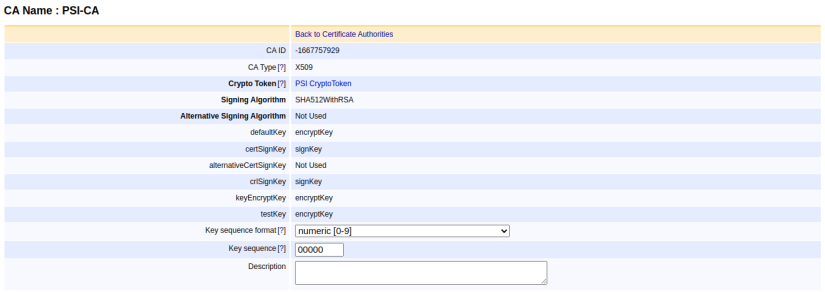

Figura 6.13: Valores iniciales de la Autoridad de Certificación PSI-CA

A continuación, se describen los campos más relevantes que se completaron durante la configuración de la CA:

- **CA Type**: Se seleccionó el tipo `X.509`, conforme a la especificación del estándar RFC 5280, utilizado para emitir certificados digitales interoperables.
- **Crypto Token**: Se eligió el token criptográfico previamente creado, denominado `PSI CryptoToken`, encargado de resguardar las claves privadas utilizadas por la CA.
- **Signing Algorithm**: Se seleccionó el algoritmo de firma `SHA256WithRSA`, compatible con los pares de claves RSA disponibles en el token. Este algoritmo se emplea tanto para la firma de certificados como para listas de revocación (CRL).
- **defaultKey**: Se configuró la clave `encryptKey` como clave por defecto para operaciones de cifrado general.
- **certSignKey**: Se estableció la clave `signKey` como responsable de firmar los certificados emitidos.
- **crlSignKey**: También se asignó la clave `signKey` para la firma de las listas de certificados revocados (CRL).
- **keyEncryptKey**: Se designó la clave `encryptKey` para operaciones de recuperación de claves y desencriptación de información almacenada.
- **testKey**: Finalmente, se utilizó nuevamente la clave `encryptKey` como clave de prueba para validar el correcto funcionamiento del token criptográfico asociado.

Algoritmo de firma

A continuación los motivos que fundamentan de la elección de SHA256WithRSA como algoritmo de firma:

- **Compatibilidad y amplio soporte:** SHA256WithRSA es compatible con herramientas, bibliotecas y sistemas operativos modernos, incluidos OpenSSH, OpenSSL. Esto asegura la operatividad con los servicios de infraestructura existentes sin requerir configuraciones adicionales o software especializado. Otros algoritmos como Ed25519, ECDSA o SHA3 tienen soporte parcial o pueden requerir condiciones específicas de versión o librerías, lo que dificulta su uso en infraestructuras heterogéneas.
- **Seguridad probada y robusta:** SHA-2 (familia a la que pertenece SHA-256) es actualmente considerada segura por la comunidad criptográfica internacional. Combinada con RSA, una de las tecnologías de cifrado asimétrico más evaluadas y maduras, ofrece un excelente equilibrio entre robustez y desempeño.
- **Rendimiento y eficiencia:** A diferencia de algoritmos más modernos (como los basados en curvas elípticas o post-cuánticos), RSA con claves de 2048 o 3072 bits y SHA-256 no representa una carga computacional excesiva, y su desempeño es adecuado para la mayoría de los casos de uso, incluyendo la autenticación de múltiples usuarios en sistemas distribuidos.
- **Estándares y recomendaciones internacionales:** Organismos como NIST recomiendan SHA-2 (y en particular SHA-256) como algoritmo de hash seguro.
- **Disponibilidad en EJBCA:** El framework EJBCA utilizado como Autoridad Certificante en este proyecto permite una configuración sencilla y directa con SHA256WithRSA, sin necesidad de modificar perfiles avanzados o integrar extensiones adicionales.
- **Evitar algoritmos obsoletos o experimentales:** Aunque existen opciones más recientes como Ed25519, algoritmos post-cuánticos (FALCON, KYBER, DILITHIUM) o los basados en curvas elípticas (ECDSA), estos no cuentan aún con un soporte igual de extendido en todas las plataformas utilizadas en la infraestructura (por ejemplo, OpenSSH). Además, algoritmos como SHA1WithRSA han sido descartados por ser considerados vulnerables ante ataques de colisión.

SHA256WithRSA representa una opción madura, segura y ampliamente soportada, lo que lo convierte en la alternativa más conveniente para un entorno que prioriza la estabilidad, la compatibilidad y la facilidad de gestión centralizada de certificados digitales.

Certificado de la CA

El certificado correspondiente a la Autoridad de Certificación raíz (CA) fue configurado como autofirmado, consolidando una infraestructura con una única CA raíz. Esta decisión responde al modelo jerárquico adoptado para la Infraestructura de Clave Pública implementada.

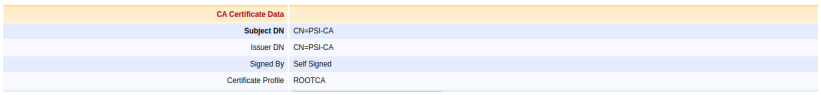

Figura 6.14: Certificado de la Autoridad de Certificación PSI-CA

Los campos más relevantes del certificado generado son los siguientes:

- **Subject DN**: `CN=PSI-CA`. Define el *Distinguished Name* (DN) que identifica de forma única a la CA dentro de la jerarquía de certificados.
- **Issuer DN**: `CN=PSI-CA`. Dado que se trata de una CA raíz, el emisor del certificado es la propia entidad (autofirmado).
- **Signed By**: *Self Signed*. Indica que la firma del certificado fue realizada por la misma entidad emisora.
- **Certificate Profile**: `ROOTCA`. Hace referencia al perfil de certificado utilizado, el cual incluye configuraciones y extensiones específicas para autoridades de certificación raíz. Este perfil será detallado más adelante.

Parámetros de la Lista de Certificados Revocados (CRL)

La configuración de las Listas de Certificados Revocados (CRL) permite establecer políticas sobre la validez, emisión periódica y superposición temporal entre versiones. Estos parámetros son fundamentales para asegurar la disponibilidad y vigencia de la información de revocación.

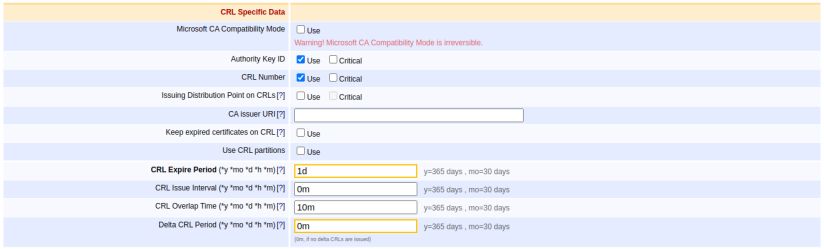

Figura 6.15: Parámetros de configuración de la CRL

- **CRL Expire Period**: `1d`. Define la duración de validez de una CRL. En este caso, cada CRL tiene una vigencia de 24 horas desde su emisión.
- **CRL Issue Interval**: `0m`. Indica que la generación de nuevas CRLs no se realizará de forma automática en intervalos fijos.
- **CRL Overlap Time**: `10m`. Establece un margen de superposición entre la CRL saliente y la entrante, permitiendo una transición segura de 10 minutos sin pérdida de validez.
- **Delta CRL Period**: `0m`. Al estar configurado en cero, no se generarán CRLs diferenciales (delta), sino únicamente listas completas.

Validación del estado de certificados

La validación del estado de los certificados emitidos por la CA se realiza a través de múltiples mecanismos habilitados desde la configuración de la misma. Estos puntos de acceso permiten a los clientes consultar información actualizada sobre la vigencia de los certificados.

Figura 6.16: Configuración de puntos de validación de certificados

- **Default CRL Distribution Point**: URL que se incluye en los certificados para permitir la descarga de la CRL correspondiente.
- **Default CRL Issuer**: `CN=PSI-CA`. Identifica a la entidad emisora de la CRL, que en este caso coincide con la CA raíz.
- **Default Freshest CRL Distribution Point**: URL adicional para la distribución de CRLs delta, en caso de estar habilitadas.
- **OCSP Service Default URI**: Dirección del servicio OCSP que permite verificar en línea el estado de los certificados, sin necesidad de descargar la CRL completa.

Perfil de certificado ROOTCA

El perfil `ROOTCA` fue utilizado para emitir el certificado autofirmado de la Autoridad de Certificación raíz. Este perfil establece una serie de restricciones y extensiones críticas, orientadas a reforzar la seguridad y validez del certificado en el largo plazo.

- **Certificado autofirmado (Self-signed)**: El emisor y el sujeto son la misma entidad, característica esencial de una CA raíz.
- **Uso de clave**: Se habilitan los usos de clave necesarios para firmar otros certificados (keyCertSign) y listas de revocación (cRLSign).
- **Extensiones críticas**: Incluye extensiones como `Basic Constraints` (CA=true) y `Key Usage`, ambas marcadas como críticas para garantizar su validación.
- **Alta validez temporal**: El certificado se configuró con un período de validez extendido, en este caso 25 años, acorde al rol central de una CA raíz dentro de la PKI.

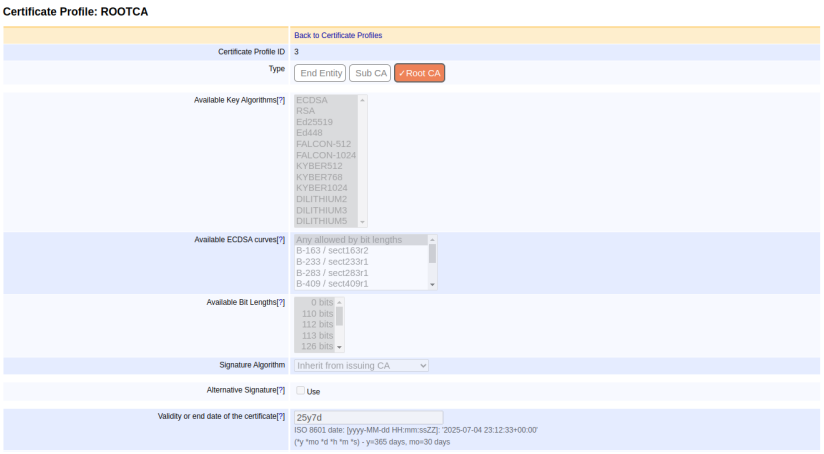

Figura 6.17: Configuración del perfil de certificado ROOTCA

Perfil de Certificado ENDUSER

El perfil `ENDUSER` está orientado a la emisión de certificados digitales para entidades finales, tales como usuarios, dispositivos o servicios. Este perfil se encuentra configurado con parámetros específicos que responden a las necesidades de autenticación, firma y cifrado sin conferir privilegios de emisión propios de una CA.

- **Uso de clave**: Se encuentran habilitados los campos `Digital Signature` y `Key Encipherment`, necesarios para autenticación, firma digital y cifrado de datos.
- **Extensiones del certificado**: Se incluye la extensión `Subject Alternative Name (SAN)`, utilizada para incorporar identificadores adicionales como direcciones de correo electrónico, nombres DNS o URIs.
- **Validez**: El período de validez del certificado se estableció en 2 años, menor al de los certificados de CA, promoviendo su renovación periódica.
- **Puntos de validación**: Se habilitaron campos como `CRL Distribution Point` y `OCSP URI`, permitiendo la verificación del estado de los certificados emitidos.
- **Restricciones de uso**: La opción de actuar como Autoridad de Certificación está deshabilitada (`Basic Constraints: CA=false`), impidiendo que este certificado firme otros certificados.

Este perfil constituye una base segura para la administración del ciclo de vida de los certificados emitidos a usuarios finales dentro del sistema.

Figura 6.18: Perfil de certificado ENDUSER

Perfil de Certificado SERVER_MTLS

El perfil `SERVER_MTLS` está diseñado específicamente para la emisión de certificados utilizados en conexiones TLS mutuas (mTLS), tanto del lado del cliente como del servidor. Estos certificados son requeridos para autenticar hosts y servicios dentro del sistema, permitiendo establecer conexiones seguras y verificadas entre componentes.

- **Uso de clave**: Se habilitan los campos `Digital Signature` y `Key Encipherment`, necesarios para establecer sesiones TLS, firmar mensajes y cifrar claves de sesión durante el handshake.
- **Uso extendido de clave (Extended Key Usage)**: Este perfil habilita los propósitos `Client Authentication` y `Server Authentication`, lo que permite al certificado ser utilizado tanto por clientes (como el módulo PAM) como por servidores (como el servicio FastAPI) en procesos de autenticación mutua.
- **Extensiones del certificado**: Se incorpora la extensión `Subject Alternative Name (SAN)` para asociar al certificado identificadores adicionales como nombres DNS o direcciones IP del host.
- **Validez**: La validez del certificado se estableció en un año, considerando las buenas prácticas de seguridad para la rotación periódica de certificados expuestos a la red.
- **Puntos de validación**: Se configuraron campos como `CRL Distribution Point` y `OCSP URI` para permitir la verificación del estado de revocación del certificado durante las conexiones TLS.
- **Restricciones de uso**: La capacidad de actuar como una Autoridad de Certificación está explícitamente deshabilitada (`Basic Constraints: CA=false`), garantizando que estos certificados no puedan ser utilizados para emitir otros.

Este perfil establece una base segura para la administración de los host dentro del sistema.

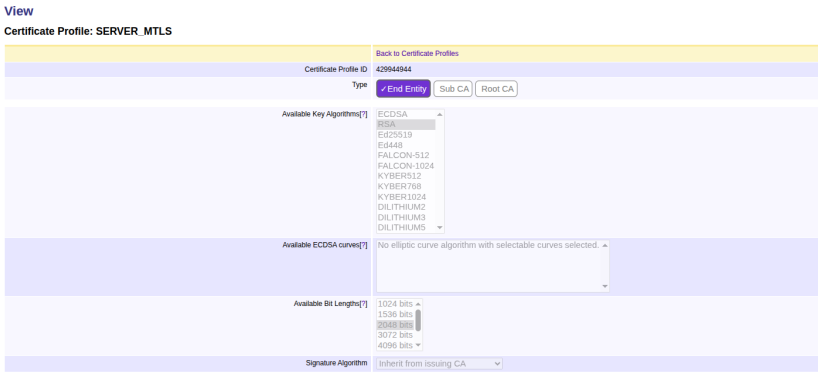

Figura 6.19: Perfil de certificado SERVER_MTLS

Perfil de Entidad Final

En EJBCA, una entidad final representa al titular de un certificado digital. Puede tratarse de una persona, un dispositivo, una sub-CA o un componente específico del sistema. La gestión de estas entidades se realiza a través de los perfiles de entidad final, los cuales permiten definir políticas de inscripción y emisión, tales como campos obligatorios, restricciones y valores por defecto.

Desde la sección `RA Functions >End Entity Profiles`, es posible gestionar y crear nuevos perfiles a través del apartado `Add End Entity`.

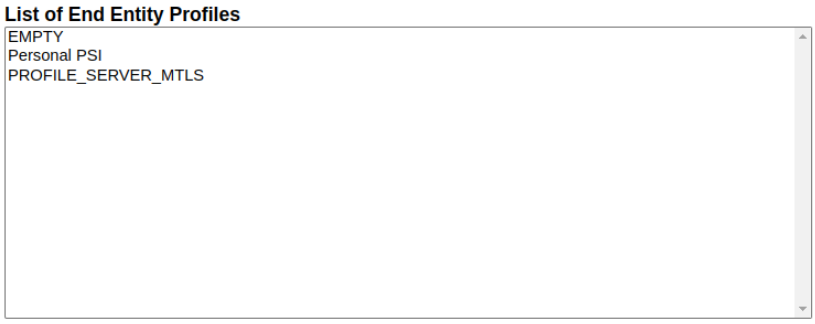

Figura 6.20: Listado de perfiles de entidades finales en EJBCA

En este proyecto, se definió dos perfiles de entidad final denominado `Personal PSI` y `PROFILE_SERVER_MTLS`, el cual utiliza el perfil de certificado `ENDUSER` y `SERVER_MTLS` respectivamente y tiene asignada como CA emisora a `PSI-CA`.

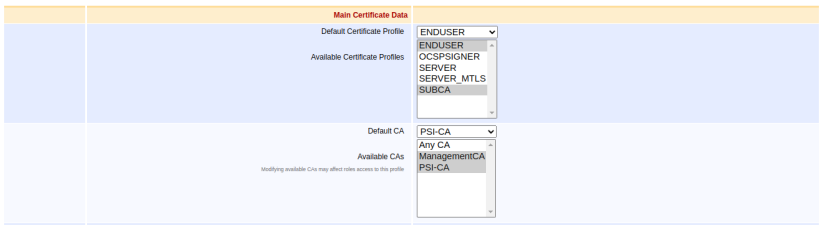

Figura 6.21: Perfil de entidad final Personal PSI

Emisión de Certificados

La emisión de certificados se realizó desde la interfaz RA Web de EJBCA. Para ello, se accedió a la sección `Enroll` y se seleccionó la opción `Make New Request`. En esta sección, se completaron los campos relacionados con el tipo de certificado, el método de generación de claves y los datos del solicitante.

Se eligió el perfil de entidad final `Personal PSI`, asociado al perfil de certificado `ENDUSER`, y se designó como autoridad emisora a la CA `PSI-CA`. Para la generación del par de claves, se seleccionó la opción que permite a la CA crear internamente el par criptográfico.

- **Algoritmo de clave**: RSA de 2048 bits, en concordancia con las políticas establecidas.
- **CN**: Se ingresaron el nombre completo.
- **Email**: Correo electronico del usuario técnico.
- **Role**: Se ingresaron los roles de acceso que se le asignarán al certificado.

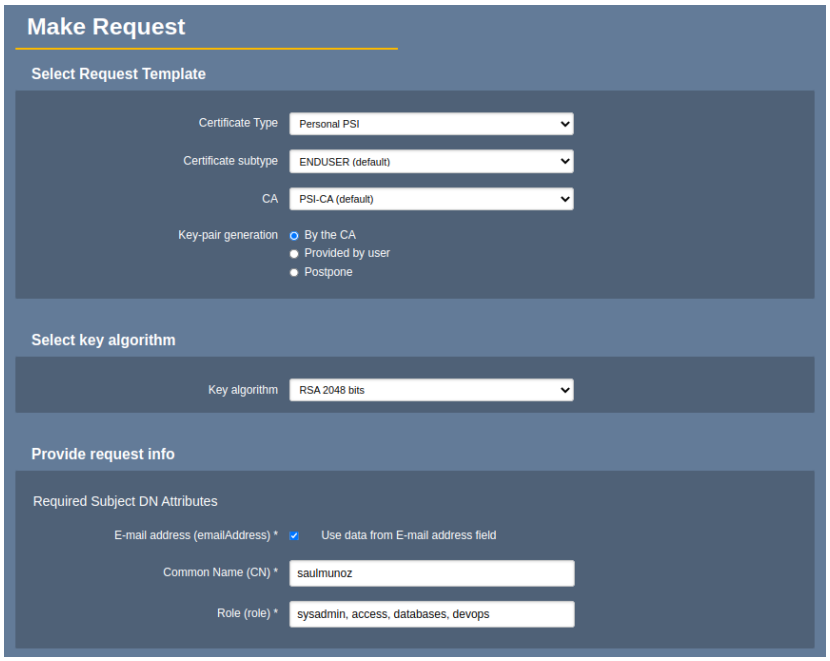

Figura 6.22: Formulario de solicitud de emisión de certificado

Tras completar la solicitud, el sistema generó el certificado correspondiente, habilitando su descarga en distintos formatos:

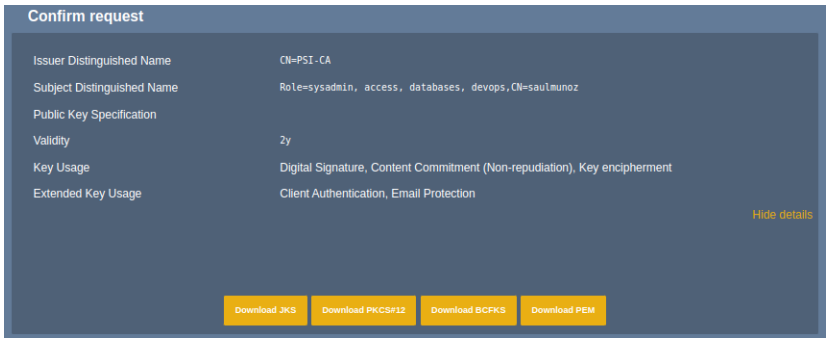

Figura 6.23: Opciones de descarga del certificado emitido

Este proceso es idéntico para el caso de dar de alta nuevos host dentro del sistema. Solamente que en vez de tener perfil de entidad final `Personal PSI`, será `PROFILE_SERVER_MTLS` asociado al perfil de certificado `SERVER_MTLS`.

6.3. Iteración III: Despliegue y configuración del sistema de Autenticación

Esta tercera iteración tuvo como propósito validar los requerimientos funcionales **RF2**, enfocados en garantizar que los usuarios accedan a los servidores únicamente si cuentan con un certificado digital válido y poseen los permisos adecuados según su rol dentro de la organización.

Para lograr este control, se implementó un sistema de software libre de doble autenticación que combina el uso del servidor bastión como punto de entrada (gateway), un nuevo chequeo en los Destinos (hosts) y una lógica personalizada de validación alojada en un servidor RBAC. Este sistema se apoya en el uso del **Pluggable Authentication Module (PAM)** en el bastión, junto a un script en Python que verifica los certificados digitales presentados por los usuarios técnicos al momento de realizar conexiones SSH.

6.3.1. Despliegue del servicio de autenticación

Para el despliegue inicial del sistema, se clonó el repositorio del proyecto disponible en la siguiente URL:

https://github.com/gerlamberti/RBAC-module.git

Este repositorio contiene todos los archivos necesarios para la orquestación del servicio involucrado.

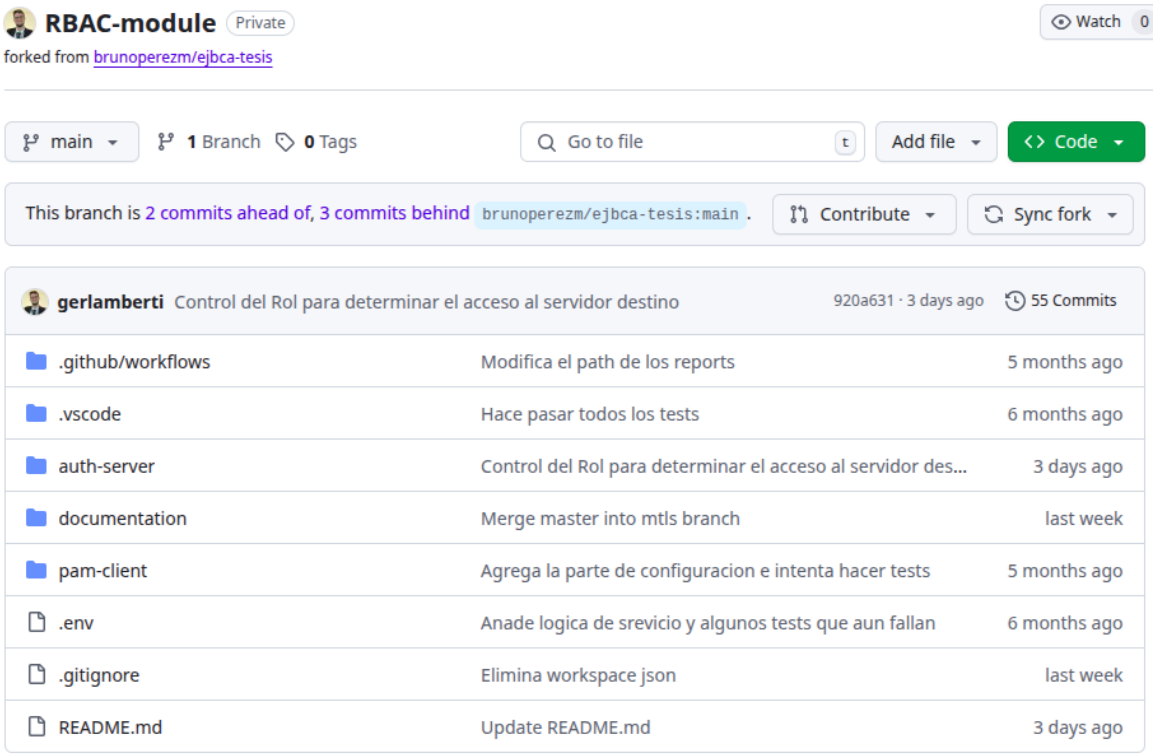

Figura 6.24: Repositorio del RBAC Module

Una vez clonado el repositorio, se procedió al despliegue de los servicios definidos en el archivo `auth-server/docker-compose.yml`, mediante el siguiente comando:

```
docker compose up -d
```

La Figura 6.25 muestra los contenedores activos luego del despliegue inicial.

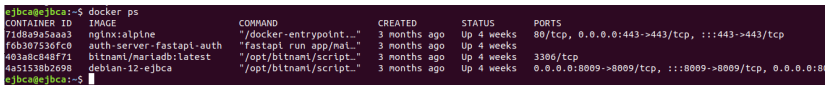

Figura 6.25: Contenedores desplegados por Docker Compose

Los servicios principales desplegados fueron:

- **sauth-server-nginx-1**: Servidor NGINX que actúa como proxy inverso para solicitar el certificado mTLS y enrutar las solicitudes HTTPS entrantes, ya sea hacia el backend de autenticación FastAPI o hacia la interfaz de administración de EJBCA.
- **auth-server-fastapi-auth-1**: Servicio que gestiona la autenticación de los usuarios técnicos. Este servicio expone una API REST que recibe el `SERIAL_ID` del certificado, valida su autenticidad y consulta los atributos asociados en función de la estructura de roles definida.

El servicio de autenticación está desarrollado en el framework **FastAPI** y se desplegó en el servidor RBAC, dentro de un contenedor Docker. Su función principal es validar certificados X.509 emitidos por EJBCA mediante una API REST que interactúa con la infraestructura de autenticación.

Para facilitar el desarrollo y la portabilidad del sistema, se ha empleado una arquitectura basada en capas (presentación, aplicación, dominio e infraestructura), lo que permite una separación clara de responsabilidades, mayor mantenibilidad y facilidad de pruebas unitarias.

El contenedor expone un endpoint principal: `GET /certificate/{serial_id}/validate`, el cual recibe el número de serie del certificado y responde con el estado de validez y, si corresponde, la entrada que debe ser escrita en `authorized_keys` por el módulo PAM del servidor bastión.

Internamente, el servicio realiza las siguientes acciones:

1. Consulta a EJBCA el estado de revocación del certificado.
2. Recupera el certificado completo si no está revocado.
3. Valida su vigencia.
4. Extrae la clave pública y la convierte a formato SSH.
5. Devuelve una respuesta JSON que incluye el campo `authorized_keys_entry`.

6.3.2. Configuraciones esenciales de PAM en Gateway y Hosts

Para implementar el sistema de autenticación, se realizaron las siguientes configuraciones clave:

Configuración del servidor SSH En el archivo `/etc/ssh/sshd_config`, se definieron los siguientes parámetros:

- `PubkeyAuthentication yes`: Habilita la autenticación por clave pública.
- `KbdInteractiveAuthentication yes`: Habilita el método de autenticación por teclado interactivo.
- `AuthenticationMethods keyboard-interactive,publickey`: Requiere primero interacción con el usuario (vía PAM) y luego la autenticación por clave pública.
- `UsePAM yes`: Permite el uso de módulos PAM en el proceso de autenticación.
- `PermitUserEnvironment yes`: Esta opción permite que se cargue un archivo .ssh/environment desde el home del usuario al momento de iniciar sesión vía SSH.

Instalación de PAM Python Se instaló el soporte para módulos PAM en Python con el siguiente comando:

```
sudo apt install libpam-python
```

El módulo `pam_python.so` permite escribir módulos en Python que pueden interactuar con el sistema PAM. Este módulo debe estar ubicado en `/lib/x86_64-linux-gnu/security`.

Configuración de PAM Python El archivo `/etc/pam.d/sshd` fue modificado para incluir el módulo personalizado en Python. Añadiendo:

```
auth    required    pam_python.so /usr/lib/security/pam_certauth.py
session required    pam_python.so /usr/lib/security/pam_certauth.py
```

Estas directivas permiten invocar funciones de autenticación (`pam_sm_authenticate`) y gestión de sesión (`pam_sm_close_session`) desde el script Python.

Instalación del certificado mTLS Para el funcionamiento correcto del sistema de autenticación basado en mTLS (autenticación mutua mediante TLS), es indispensable que cada host involucrado —como el **Gateway**, **Destino1** y **Destino2**— cuente con un certificado digital emitido por la Autoridad Certificadora (AC) del sistema. Este certificado debe ser reconocido y confiable por el servicio de autenticación, el cual verifica que cada cliente esté autenticado criptográficamente antes de responder a cualquier solicitud.

El cliente PAM (implementado en el servidor Gateway) debe presentar este certificado en cada conexión HTTPS hacia el servicio de autenticación, ubicado en el servidor RBAC. Para esto, es necesario extraer tanto el certificado como su clave privada desde un archivo PKCS#12 (formato `.p12` o `.pfx`), y colocarlos en una ruta predefinida accesible por el módulo PAM, típicamente:

```
/usr/lib/security/
```

Extracto de código 6.1: Ruta destino de certificados cliente

Dicho directorio debe contener los siguientes archivos:

- `client.pem`: certificado público del host.
- `client.key`: clave privada correspondiente.

A continuación se detallan los comandos para obtener estos archivos desde un paquete PKCS#12 (`certificado.p12`) proporcionado por la AC:

1. Extraer el certificado público:

```
openssl pkcs12 -in certificado.p12 -clcerts -nokeys -out /usr/lib/security/client.
    pem

```

Extracto de código 6.2: Extracción del certificado desde un archivo .p12

2. Extraer la clave privada:

```
openssl pkcs12 -in certificado.p12 -nocerts -nodes -out /usr/lib/security/client.
    key

```

Extracto de código 6.3: Extracción de la clave privada desde un archivo .p12

3. Asegurar los permisos sobre los archivos extraídos:

```
chmod 600 /usr/lib/security/client.key
chmod 644 /usr/lib/security/client.pem

```

Extracto de código 6.4: Permisos mínimos recomendados

Es importante asegurar que los archivos tengan los permisos mínimos necesarios para evitar fugas de información sensible, especialmente en el caso de la clave privada. Por convención, ambos archivos deben ser accesibles por el servicio PAM y mantenidos en un entorno controlado.

Funcionamiento del pam_certauth.py El módulo Python implementa las siguientes funcionalidades:

- En la función `pam_sm_authenticate`, solicita al usuario su `serial_id`, consulta al servidor RBAC si el certificado es válido, y escribe la clave pública en `authorized_keys`.
- En `pam_sm_close_session`, al finalizar la sesión, se borra el contenido de `authorized_keys`, garantizando que cada conexión requiere una autenticación completa.

El script debe ser ubicado en el directorio `/usr/lib/security` bajo el nombre `pam_certauth.py`, para que pueda ser invocado correctamente por el módulo PAM.

```

#!/usr/bin/env python2
# -*- coding: utf-8 -*-
import json
import urllib2
import ssl

# Funcion para llamar al endpoint del servidor de autenticacion usando un certificado
    cliente.
def authenticate(serial_id, username):
    base_url = "https://10.10.5.2"  # cambiado a HTTPS
    url = base_url + "/api/v1/certificate/" + serial_id + "/validate?username=" +
    username

    try:
        # Configurar el contexto SSL con los certificados
        ctx = ssl.create_default_context(ssl.Purpose.CLIENT_AUTH)

        # Cargar certificado y llave del cliente
        ctx.load_cert_chain(
            certfile='/usr/lib/security/client.pem',  # Ruta al certificado del cliente
            keyfile='/usr/lib/security/client.key'    # Ruta a la llave privada del
    cliente
```

```
        )

        # Verificacion del certificado del servidor (opcional, pero recomendado)
        #ctx.load_verify_locations('/etc/pam_python/ca.crt')  # Ruta al certificado CA

        # Crear solicitud HTTPS
        request = urllib2.Request(url)

        # Abrir la conexion con el contexto SSL
        response = urllib2.urlopen(request, context=ctx)
        code = response.getcode()
        content = response.read()

    except urllib2.URLError as e:
        print("Error llamando al endpoint para serial: %s" % serial_id)
        print(str(e))
        return {"allowed": False, "authorized_keys_entry": None}

    if code == 200:
        return json.loads(content)
    elif code in (400, 403):
        try:
            return json.loads(content)
        except Exception:
            return {"allowed": False, "authorized_keys_entry": None}
    elif code == 500:
        print("Error interno en el servidor de autenticacion para el certificado %s" %
    serial_id)
        return {"allowed": False, "authorized_keys_entry": None}
    else:
        print("Codigo de respuesta inesperado %s para el certificado %s" % (code,
    serial_id))
        return {"allowed": False, "authorized_keys_entry": None}

def pam_sm_authenticate(pamh, flags, argv):
    try:
        user = pamh.get_user(None)
        if user is None:
            print("Usuario es None")
            return pamh.PAM_USER_UNKNOWN

        # Abrir el archivo authorized_keys
        f = open("/home/" + user + "/.ssh/authorized_keys", "w+")
        if f is None:
            print("No se pudo abrir el archivo authorized_keys")
            return pamh.PAM_USER_UNKNOWN

        msg = pamh.Message(
            pamh.PAM_PROMPT_ECHO_ON, "Ingrese el serial_id de su certificado: "
        )
        resp = pamh.conversation(msg)
        serial_id = resp.resp
        if serial_id is None:
            return pamh.PAM_USER_UNKNOWN

        pamh.conversation(
            pamh.Message(
```

```
                pamh.PAM_TEXT_INFO, "Buscando certificado con serial_id: " + serial_id
            )
        )

        # Llamar al servidor de autenticacion
        auth_response = authenticate(serial_id, user)
        if not auth_response.get("allowed"):
            print("El certificado %s no esta autorizado." % serial_id)
            return pamh.PAM_AUTH_ERR

        # Obtener el campo autorizado_keys_entry (nota: en la respuesta es "
    authorized_keys_entry")
        authorized_keys_entry = auth_response.get("authorized_keys_entry")
        if not authorized_keys_entry:
            print("Error en la respuesta del servicio para serial_id %s." % serial_id)
            return pamh.PAM_AUTH_ERR


        pamh.conversation(
            pamh.Message(
                pamh.PAM_TEXT_INFO,
                "Certificado encontrado! Clave autorizada anadida a authorized_keys.",
            )
        )

        f.write(authorized_keys_entry + "\n")
        f.close()

        # Para testeo se escribe informacion en /tmp/enviroment_test
        f2 = open("/tmp/enviroment_test", "w")
        f2.write("\nuser:" + user)
        f2.close()
    except Exception as e:
        f3 = open("/tmp/error", "w")
        f3.write(str(e))
        f3.close()
        return pamh.PAM_USER_UNKNOWN

    return pamh.PAM_SUCCESS


def pam_sm_setcred(pamh, flags, argv):
    return pamh.PAM_SUCCESS


def pam_sm_acct_mgmt(pamh, flags, argv):
    return pamh.PAM_SUCCESS


def pam_sm_open_session(pamh, flags, argv):
    return pamh.PAM_SUCCESS


def pam_sm_close_session(pamh, flags, argv):
    user = pamh.get_user(None)
    if user is None:
        print("Usuario es None")
```

```
132         return pamh.PAM_USER_UNKNOWN
133
134     # Abrir el archivo authorized_keys
135     f = open("/home/" + user + "/.ssh/authorized_keys", "w+")
136     f.write("")
137     f.close()
138     if f is None:
139         print("No se pudo abrir el archivo authorized_keys")
140
141     return pamh.PAM_SUCCESS
142
143
144 def pam_sm_chauthtok(pamh, flags, argv):
145     return pamh.PAM_SUCCESS
146
147
148 if __name__ == "__main__":
149     # Prueba simple de la llamada al servidor de autenticacion
150     result = authenticate("1eb97febf0e01bb7f1891cbd837087af3064740b", "test_user")
151     print("Resultado de autenticacion: %s" % result)
```

Extracto de código 6.5: Modulo personalizado PAM Python con escritura de clave pública

6.3.3. Configuraciones no implementadas

La integración de OpenSSH con herramientas de protección contra ataques por fuerza bruta, como Fail2Ban, no fue agregada en el desarrollo de este proyecto, al quedar fuera del alcance definido. La incorporación de mecanismos de protección dinámica como Fail2Ban podría complementar eficazmente este modelo, fortaleciendo la seguridad en entornos productivos y mitigando riesgos de accesos no autorizados o automatizados. Se recomienda su evaluación e implementación en futuras etapas de mejora continua del sistema.

6.4. Iteración IV: Implementación del sistema de Autorización

En esta iteración se documenta la implementación del sistema de autorización, encargado de controlar las operaciones permitidas para los usuarios autenticados, en función de los roles definidos en sus certificados digitales. Esta funcionalidad respondió a los requerimientos funcionales RF3 y RF4, que establecían la necesidad de aplicar políticas de control de acceso basadas en roles (RBAC) en los servidores gestionados.

6.4.1. Instalación de Ansible

Ansible en ARCenter

Con el objetivo de automatizar la configuración de los servidores, se utilizó Ansible como herramienta de orquestación. Desde el servidor de administración central (ARCenter), se definieron playbooks específicos para crear usuarios, asignar permisos y aplicar reglas de acceso de forma controlada y repetible. Esta estrategia permitió asegurar consistencia y trazabilidad en todos los nodos gestionados.

La instalación de Ansible se llevó a cabo en el servidor ARCenter. Primero, se actualizó la lista de paquetes:

```
sudo apt update
```

Luego, se instaló el paquete correspondiente:

```
sudo apt install ansible
```

Extracto de código 6.6: Instalación de Ansible en ARCenter

Una vez instalado `ansible` en el servidor ARCenter, se procedió a generar una clave SSH que permitiera establecer conexión segura con los servidores gateway y hosts, sin requerir autenticación interactiva.

La generación del par de claves se realizó mediante el siguiente comando:

```
ssh-keygen -t ed25519
```

Extracto de código 6.7: Generación de clave SSH en ARCenter

Dicho comando generó un par de claves —pública y privada— en el directorio `~/.ssh`, que luego se utilizaron para establecer la confianza entre ARCenter y los servidores restantes.

Creación de usuarios receptores

Una vez instalado Ansible en ARCenter y configurado el acceso SSH, se procedió a la creación de los usuarios remotos que funcionarían como receptores de las tareas automatizadas. En este caso, se optó por el usuario `ansibleadmin`, creado en cada servidor gestionado mediante el siguiente comando:

```
sudo useradd -m -s /bin/bash ansibleadmin
echo "ansibleadmin:ansibleadmin" | sudo chpasswd
```

Extracto de código 6.8: Creación del usuario ansibleadmin en los servidores destino

El usuario fue configurado con un intérprete bash y contraseña por defecto igual a su nombre, exclusivamente con fines de pruebas controladas.

Posteriormente, se editó el archivo `/etc/sudoers` en cada servidor para otorgar a `ansibleadmin` privilegios de superusuario sin requerir contraseña. Para ello se añadió la siguiente línea:

```
ansibleadmin ALL=(ALL) NOPASSWD: ALL
```

Extracto de código 6.9: Regla sudoers para ansibleadmin

Este permiso fue necesario para permitir la ejecución de tareas administrativas vía Ansible.

Configuración de PAM para compatibilidad con Ansible

Dado que el mecanismo de autenticación implementado a través de PAM interfería con el acceso automático requerido por Ansible, fue necesario excluir al usuario `ansibleadmin` de dicha interacción. Para lograrlo, se agregó una excepción en el archivo `/etc/pam.d/sshd`, incorporando las siguientes líneas al inicio del archivo:

```
auth    [success=1 default=ignore] pam_succeed_if.so user = ansibleadmin
auth    required      pam_python.so /usr/lib/security/pam_certauth.py

session    [success=1 default=ignore] pam_succeed_if.so user = ansibleadmin
session required      pam_python.so /usr/lib/security/pam_certauth.py
```

Extracto de código 6.10: Exclusión de ansibleadmin del control de PAM

Esta instrucción permitió que el usuario `ansibleadmin` accediera mediante SSH sin ser interceptado por las reglas personalizadas del módulo PAM, reservadas para los usuarios técnicos autenticados por certificado.

Intercambio de claves y verificación de acceso

Finalmente, la clave pública generada en ARCenter fue copiada al archivo `/.ssh/authorized_key` del usuario `ansibleadmin` en cada servidor. Esta operación habilitó el acceso remoto sin interacción, requisito indispensable para el funcionamiento de los playbooks de Ansible.

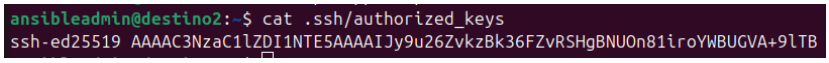

Figura 6.26: Clave pública de ARCenter en Destino 2

El sistema quedó así configurado para permitir la ejecución remota de tareas automatizadas, manteniendo control sobre los permisos y simplificando el proceso de despliegue de reglas de autorización por rol.

6.4.2. Gestión de roles en gateway y hosts

Una vez configurado Ansible, se procedió a la creación de los usuarios de sistema que representarían los roles operativos. Esta tarea se automatizó mediante playbooks

ejecutados desde ARCenter hacia los servidores destino (gateway y hosts), como se muestra en esta subsección.

Definición de reglas

En sistemas Linux, los usuarios que pertenecen al grupo `sudo` pueden ejecutar comandos privilegiados. Para evitar accesos amplios no controlados, se definieron reglas específicas en el archivo `/etc/sudoers`, limitando las operaciones autorizadas para cada usuario.

Un ejemplo simple de regla definida fue el siguiente:

```
databases ALL=(ALL) NOPASSWD: /usr/bin/apt update mysql-servers
```

Extracto de código 6.11: Definición de regla en el archivo sudoers

Análisis de la sintaxis:

- `databases`: usuario al cual se le aplica la regla.
- `ALL=(ALL)`: puede ejecutar como cualquier usuario y grupo.
- `NOPASSWD:`: no se solicita contraseña para la ejecución.
- `/usr/bin/apt update mysql-servers`: comando autorizado.

Entendida la estructura general, se procedió a construir reglas específicas para los tres roles principales: `databases`, `sysadmin` y `devops`, respetando el principio de privilegio mínimo.

Rol: databases

```
databases ALL=(ALL) NOPASSWD: \
    /usr/bin/systemctl start mysql, \
    /usr/bin/systemctl stop mysql, \
    /usr/bin/systemctl restart mysql, \
    /usr/bin/mysql, \
    /usr/bin/mysqldump, \
    /usr/bin/mysqladmin, \
    /usr/bin/apt update mysql-servers
```

Extracto de código 6.12: Regla para el rol databases

- Permite gestionar el servicio MySQL.
- Habilita herramientas administrativas como dumps o reinicios.

Rol: sysadmin

```
sysadmin ALL=(ALL) NOPASSWD: \
    /usr/sbin/useradd, \
    /usr/sbin/userdel, \
    /usr/sbin/usermod, \
    /usr/bin/passwd, \
    /usr/bin/systemctl *, \
    /usr/bin/journalctl, \
    /usr/bin/firewall-cmd, \
```

```
    /usr/bin/ufw, \
    /usr/bin/rsync, \
    /usr/bin/htop, \
    /bin/systemctl, \
    /bin/mkdir,
    /bin/chown, \
    /bin/chmod,
    /bin/su
```

Extracto de código 6.13: Regla para el rol sysadmin

- Permite administrar usuarios, servicios y configuración del sistema.
- Mantiene restricciones en comandos de acceso interactivo (e.g., shells).

Rol: devops

```
devops ALL=(ALL) NOPASSWD: \
    /usr/bin/docker, \
    /usr/bin/docker-compose, \
    /usr/bin/ansible-playbook, \
    /usr/local/bin/kubectl, \
    /usr/bin/systemctl restart nginx, \
    /usr/bin/systemctl restart apache2, \
    /usr/bin/git
```

Extracto de código 6.14: Regla para el rol devops

- Permite ejecutar herramientas de despliegue y automatización.
- No habilita acceso a configuración sensible del sistema base.

Creación de roles

Para automatizar la gestión de cuentas de usuario en los servidores gestionados, se desarrolló un conjunto de playbooks de Ansible organizados en el directorio `ansible/crear_usuarios/`. Esta estructura permite definir qué usuarios deben crearse en cada host y aplicar reglas específicas para cada uno, según su rol funcional en el sistema.

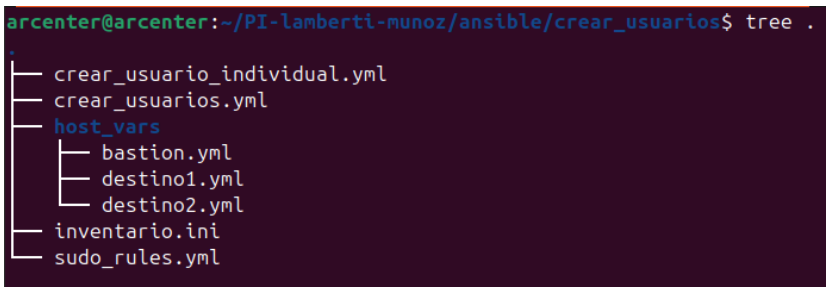

Figura 6.27: Estructura de carpetas y archivos para creación de usuarios

Playbook principal: `crear_usuarios.yml`

```yaml
---
- name: Crear usuarios en servidores segun configuracion
  hosts: all
  become: true

  vars:
    usuarios_definidos:
      access:
        password: "{{ 'access' | password_hash('sha512') }}"
        shell: /usr/sbin/nologin
        sudo: false
      sysadmin:
        password: "{{ 'sysadmin' | password_hash('sha512') }}"
        shell: /bin/bash
        sudo: false
      databases:
        password: "{{ 'databases' | password_hash('sha512') }}"
        shell: /bin/bash
        sudo: false
      devops:
        password: "{{ 'devops' | password_hash('sha512') }}"
        shell: /bin/bash
        sudo: false

  vars_files:
    - sudo_rules.yml

  tasks:
    - name: Incluir tareas para cada usuario
      include_tasks: crear_usuario_individual.yml
      loop: "{{ usuarios_a_crear | default([]) }}"
      loop_control:
        loop_var: usuario
```

Extracto de código 6.15: Playbook para crear usuarios

Este playbook es el punto de entrada principal. Aplica sobre todos los hosts definidos en el archivo de inventario y recorre la lista de usuarios especificados en los archivos `host_vars/`. Por cada usuario listado, incluye y ejecuta el archivo `crear_usuario_individual.yml` con las variables adecuadas.

El archivo también define una variable global llamada `usuarios_definidos`, que establece para cada usuario su contraseña encriptada, shell predeterminado y si debe agregarse o no al grupo `sudo`.

Playbook individual: `crear_usuario_individual.yml`

```yaml
---
- name: Crear usuario {{ usuario }}
  user:
    name: "{{ usuario }}"
    password: "{{ usuarios_definidos[usuario].password }}"
    shell: "{{ usuarios_definidos[usuario].shell }}"
    state: present
    create_home: yes

```

```
- name: Agregar {{ usuario }} al grupo sudo si corresponde
  when: usuarios_definidos[usuario].sudo
  user:
    name: "{{ usuario }}"
    groups: sudo
    append: yes

- name: Configurar reglas de sudo personalizadas para {{ usuario }}
  when: usuario in sudo_rules
  lineinfile:
    path: /etc/sudoers
    state: present
    regexp: "^{{ usuario }}\\s+ALL="
    line: "{{ usuario }} ALL=(ALL) NOPASSWD: {{ sudo_rules[usuario] | join(', ') }}"
    validate: 'visudo -cf %s'
    backup: yes

- name: Configurar ~/.bashrc del usuario {{ usuario }}
  blockinfile:
    path: "/home/{{ usuario }}/.bashrc"
    create: yes
    owner: "{{ usuario }}"
    group: "{{ usuario }}"
    mode: '0644'
    block: |
      export HISTTIMEFORMAT="%F %T "
      if [ -n "$SSH_USER_AUTH" ]; then
          export CERT=$(cat $SSH_USER_AUTH | awk '{print $2 " " $3}' | ssh-keygen -Lf -
    2> /dev/null | awk '/Key ID/ {gsub(/", "", $0); print $3}')
          if [ -n "$CERT" ]; then
              export PS1="($CERT)$PS1"
          else
              export CERT="$REMOTEUSER"
              export PS1="($CERT)$PS1"
          fi
          IP_CONNECTION=$(echo $SSH_CONNECTION | awk '{print $1}')
      fi

      PROMPT_COMMAND="${PROMPT_COMMAND:+$PROMPT_COMMAND ; }"'{
          current_hist_num=$(history 1 | awk "{print \$1}")
          last_hist_num=$(awk "{print \$4}" <(tail -n 1 /var/log/comandos_history))
          if [ "$current_hist_num" != "$last_hist_num" ]; then
              current_history=$(history 1)
              vsecuencia=$(echo "$current_history" | awk "{print \$1}")
              vtimestamp=$(echo "$current_history" | awk "{print \$2, \$3}")
              vcomando=$(echo "$current_history" | awk "{\$1=\"\"; \$2=\"\"; \$3=\"\";
    print \$0}" | sed "s/^ *//")
              echo "$CERT $USER $IP_CONNECTION $vsecuencia $vtimestamp $vcomando" >> /
    var/log/comandos_history
          fi
      }'
```

Extracto de código 6.16: Playbook que crea un usuario

Este archivo define las tareas necesarias para crear cada usuario:

- Crear la cuenta de usuario con su respectiva contraseña, shell y home directory.

- Añadirlo al grupo `sudo` si corresponde.
- Configurar reglas personalizadas en el archivo `/etc/sudoers`, utilizando las definidas en `sudo_rules.yml`.
- Insertar un bloque personalizado en su `.bashrc`, el cual permite registrar comandos ejecutados con información adicional como el rol, IP de origen y marca de tiempo. Esta información se guarda en el archivo `/var/log/comandos_history`.

Inventario: `inventario.ini`

Este archivo define los grupos de hosts y las direcciones IP correspondientes para cada servidor: `bastion`, `destino1` y `destino2`. También especifica el usuario y contraseña utilizados por Ansible para conectarse remotamente:

```
[gateway]
bastion ansible_host=10.10.7.1

[host1]
destino1 ansible_host=10.10.7.2

[host2]
destino2 ansible_host=10.10.7.4

[all:vars]
ansible_user=ansibleadmin
ansible_password=ansibleadmin
```

Extracto de código 6.17: Archivo de inventario Ansible

Asignación de usuarios por host: `host_vars/`

La asignación de usuarios a cada servidor se realizó de forma declarativa, utilizando archivos YAML ubicados en el directorio `host_vars/`. Cada archivo corresponde a un host específico y define la lista de usuarios que deben ser creados en dicho nodo, a través de la variable `usuarios_a_crear`. Esta estrategia permitió parametrizar el despliegue y evitar redundancias en los playbooks.

A continuación se muestran los archivos definidos para cada servidor:

```
usuarios_a_crear:
  - access
  - sysadmin
```

Extracto de código 6.18: Archivo `bastion.yml`

```
usuarios_a_crear:
  - databases
  - sysadmin
```

Extracto de código 6.19: Archivo `destino1.yml`

```
usuarios_a_crear:
  - devops
  - sysadmin
```

Extracto de código 6.20: Archivo `destino2.yml`

En resumen:

- En `bastion` se desplegaron los usuarios `access` y `sysadmin`.
- En `destino1`, los usuarios `databases` y `sysadmin`.
- En `destino2`, los usuarios `devops` y `sysadmin`.

Reglas sudo personalizadas: `sudo_rules.yml`

Este archivo define las operaciones autorizadas por rol en el archivo `/etc/sudoers`. Cada conjunto de reglas se asigna dinámicamente si el usuario corresponde a uno de los definidos. Por ejemplo:

```
sudo_rules:
  databases:
    - /usr/bin/systemctl start mysql
    - /usr/bin/systemctl stop mysql
    - /usr/bin/systemctl restart mysql
    - /usr/bin/mysql
    - /usr/bin/mysqldump
    - /usr/bin/mysqladmin
    - /usr/bin/apt update mysql-servers

  sysadmin:
    - /usr/sbin/useradd
    - /usr/sbin/userdel
    - /usr/sbin/usermod
    - /usr/bin/passwd
    - /usr/bin/systemctl *
    - /usr/bin/journalctl
    - /usr/bin/firewall-cmd
    - /usr/bin/ufw
    - /usr/bin/rsync
    - /usr/bin/htop
    - /bin/systemctl
    - /bin/mkdir
    - /bin/chown
    - /bin/chmod
    - /bin/su

  devops:
    - /usr/bin/docker
    - /usr/bin/docker-compose
    - /usr/bin/ansible-playbook
    - /usr/local/bin/kubectl
    - /usr/bin/systemctl restart nginx
    - /usr/bin/systemctl restart apache2
    - /usr/bin/git
```

Extracto de código 6.21: Archivo de configuración de reglas de permisos

- El usuario `databases` puede reiniciar servicios y ejecutar herramientas de MySQL.
- El usuario `sysadmin` tiene acceso a tareas administrativas amplias.
- El usuario `devops` puede ejecutar herramientas de automatización y despliegue como Docker, Ansible y Git.

Modificación de permisos de roles

Una vez creados los usuarios, se implementó un mecanismo automatizado para actualizar sus privilegios mediante la herramienta Ansible. Esta funcionalidad resultó fundamental para aplicar cambios en las reglas de `sudo` de forma segura y centralizada, asegurando coherencia entre servidores.

La estructura del directorio `modificar_permisos/` contiene los archivos necesarios para definir qué usuarios deben actualizar sus privilegios y con qué comandos específicos:

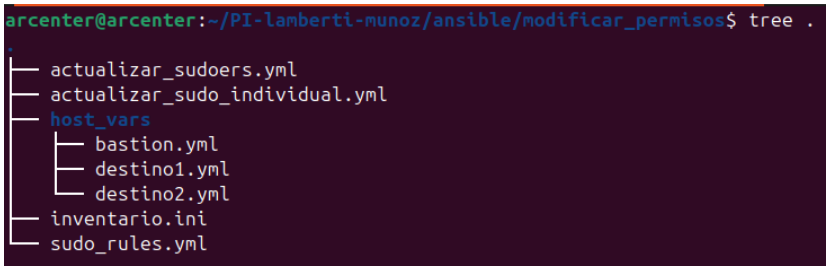

Figura 6.28: Estructura de carpetas y archivos para modificación de roles

El archivo principal `actualizar_sudoers.yml` define el playbook encargado de iterar sobre los usuarios que deben actualizar sus reglas de `sudo`, utilizando como fuente los archivos del directorio `host_vars/`:

```yaml
---
- name: Actualizar reglas de sudo de usuarios
  hosts: all
  become: true

  vars_files:
    - sudo_rules.yml

  tasks:
    - name: Incluir tareas para actualizar sudo de cada usuario
      include_tasks: actualizar_sudo_individual.yml
      loop: "{{ usuarios_a_actualizar_sudo | default([]) }}"
      loop_control:
        loop_var: usuario_item
      vars:
        usuario: "{{ usuario_item }}"
```

Extracto de código 6.22: Playbook principal de actualizacion de reglas sudo

Cada tarea individual se encuentra en el archivo `actualizar_sudo_individual.yml`, que edita el archivo `/etc/sudoers` de forma validada:

```yaml
---
- name: Asegurar entrada sudo actualizada para {{ usuario }}
  when: usuario in sudo_rules
  lineinfile:
    path: /etc/sudoers
    regexp: "^{{ usuario }}\\s+ALL="
```

```
    line: "{{ usuario }} ALL=(ALL) NOPASSWD: {{ sudo_rules[usuario] | join(', ') }}"
    validate: 'visudo -cf %s'
    backup: yes
```

Extracto de código 6.23: Tareas individuales para cada usuario

Las reglas correspondientes a cada rol se definen en el archivo `sudo_rules.yml`. A continuación se muestra un extracto representativo:

```
sudo_rules:
  databases:
    - /usr/bin/systemctl start mysql
    - /usr/bin/systemctl stop mysql
    ...
  sysadmin:
    - /usr/sbin/useradd
    - /usr/sbin/userdel
    ...
  devops:
    - /usr/bin/docker
    - /usr/bin/ansible-playbook
    ...
```

Extracto de código 6.24: Ejemplo de reglas sudo definidas por rol

Asignación de reglas por host: `host_vars/`

Cada servidor define explícitamente qué usuarios deben recibir actualizaciones en sus permisos, mediante archivos YAML localizados en el directorio `host_vars/`. Por ejemplo:

```
usuarios_a_actualizar_sudo:
 - sysadmin
```

Extracto de código 6.25: Archivo `bastion.yml`

```
usuarios_a_actualizar_sudo:
  - sysadmin
  - databases
```

Extracto de código 6.26: Archivo `destino1.yml`

```
usuarios_a_actualizar_sudo:
  - sysadmin
  - devops
```

Extracto de código 6.27: Archivo `destino2.yml`

El archivo `inventario.ini` cumple la misma función que en el proceso de creación de usuarios: define los grupos de servidores (`bastion`, `destino1`, `destino2`), sus direcciones IP y las credenciales necesarias para establecer conexión remota con cada uno. Este archivo es reutilizado para asegurar consistencia entre los distintos playbooks ejecutados desde el nodo de administración central (`ARCenter`).

Eliminación de roles

Para asegurar la correcta remoción de cuentas de usuarios del sistema, se desarrolló un conjunto de playbooks que automatizan el proceso de eliminación. Esta tarea incluye tanto la revocación de permisos administrativos en el archivo `/etc/sudoers`, como la eliminación completa del usuario y su directorio personal del sistema.

La estructura de directorios correspondiente se detalla a continuación:

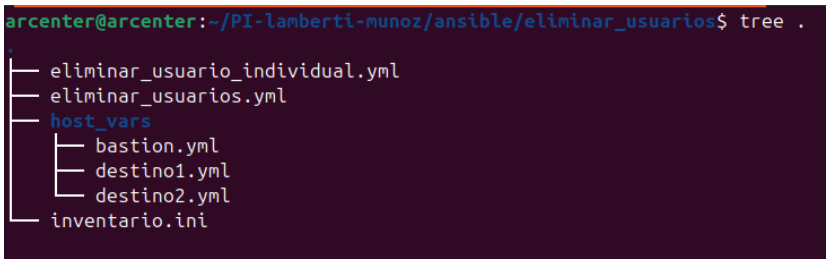

Figura 6.29: Estructura de carpetas y archivos para eliminación de roles

El playbook principal, `eliminar_usuarios.yml`, ejecuta un conjunto de tareas por cada host, basándose en las variables definidas en los archivos `host_vars`.

```
- name: Eliminar usuarios de los servidores segun configuracin
  hosts: all
  become: true

  vars:
    sudoers_path: /etc/sudoers

  tasks:
    - name: Incluir tareas para eliminar cada usuario
      include_tasks: eliminar_usuario_individual.yml
      loop: "{{ usuarios_a_eliminar | default([]) }}"
      loop_control:
        loop_var: usuario_item
      vars:
        usuario: "{{ usuario_item }}"
```

Extracto de código 6.28: Playbook principal de eliminación de usuarios

El archivo `eliminar_usuario_individual.yml` contiene las instrucciones para:

- Eliminar cualquier entrada existente del usuario en el archivo `/etc/sudoers`, si corresponde.
- Eliminar completamente el usuario del sistema, incluyendo su directorio personal y archivo de correo.

```
- name: Eliminar entrada de {{ usuario }} de /etc/sudoers si existe
  lineinfile:
    path: "{{ sudoers_path }}"
    regexp: "^{{ usuario }}\\s+ALL="
    state: absent
    validate: 'visudo -cf %s'
```

```
  ignore_errors: true

- name: Eliminar usuario {{ usuario }}
  user:
    name: "{{ usuario }}"
    state: absent
    remove: yes
```

Extracto de código 6.29: Tareas individuales para eliminación de usuario

Asignación de usuarios a eliminar por host: `host_vars/`

Al igual que en los playbooks de creación de usuarios, para cada servidor se define una lista de usuarios a eliminar en archivos YAML dentro del directorio `host_vars/`. A continuación se muestran ejemplos representativos:

```
usuarios_a_eliminar:
  - access
  - sysadmin
```

Extracto de código 6.30: Archivo de bastion.yml

```
usuarios_a_eliminar:
  - databases
  - sysadmin
```

Extracto de código 6.31: Archivo de destino1.yml

```
usuarios_a_eliminar:
  - devops
  - sysadmin
```

Extracto de código 6.32: Archivo de destino2.yml

- En `bastion.yml` se eliminaron los usuarios `access`, `sysadmin` y `databases`.
- En `destino1.yml` se eliminaron `databases` y `sysadmin`.
- En `destino2.yml` se eliminaron `devops` y `sysadmin`.

El archivo `inventario.ini` utilizado en esta tarea es el mismo que en las etapas anteriores, y cumple la función de definir la infraestructura objetivo para los playbooks de Ansible. Contiene los grupos de servidores y las credenciales necesarias para establecer la conexión desde el nodo de control central `ARCenter`.

6.4.3. Despliegue de reglas vía Ansible

Una vez desarrollados los playbooks correspondientes a la creación, modificación y eliminación de usuarios y sus respectivas reglas de acceso, se procedió a su ejecución desde el servidor central de administración `ARCenter`. Las tareas se ejecutan utilizando el comando `ansible-playbook`, que permite aplicar de forma automática y reproducible las configuraciones sobre los servidores de destino especificados en el archivo `inventario.ini`. A los fines demostrativos se presenta la creación de los usuarios `access2, sysadmin2, databases2 y devops2`.

Creación de usuarios y reglas iniciales El siguiente comando ejecuta el playbook de creación de usuarios, definiendo tanto los usuarios del sistema como las reglas personalizadas en `/etc/sudoers` para cada rol.

```
ansible-playbook -i inventario.ini crear_usuarios.yml
```

Extracto de código 6.33: Ejecución del playbook de creación de usuarios

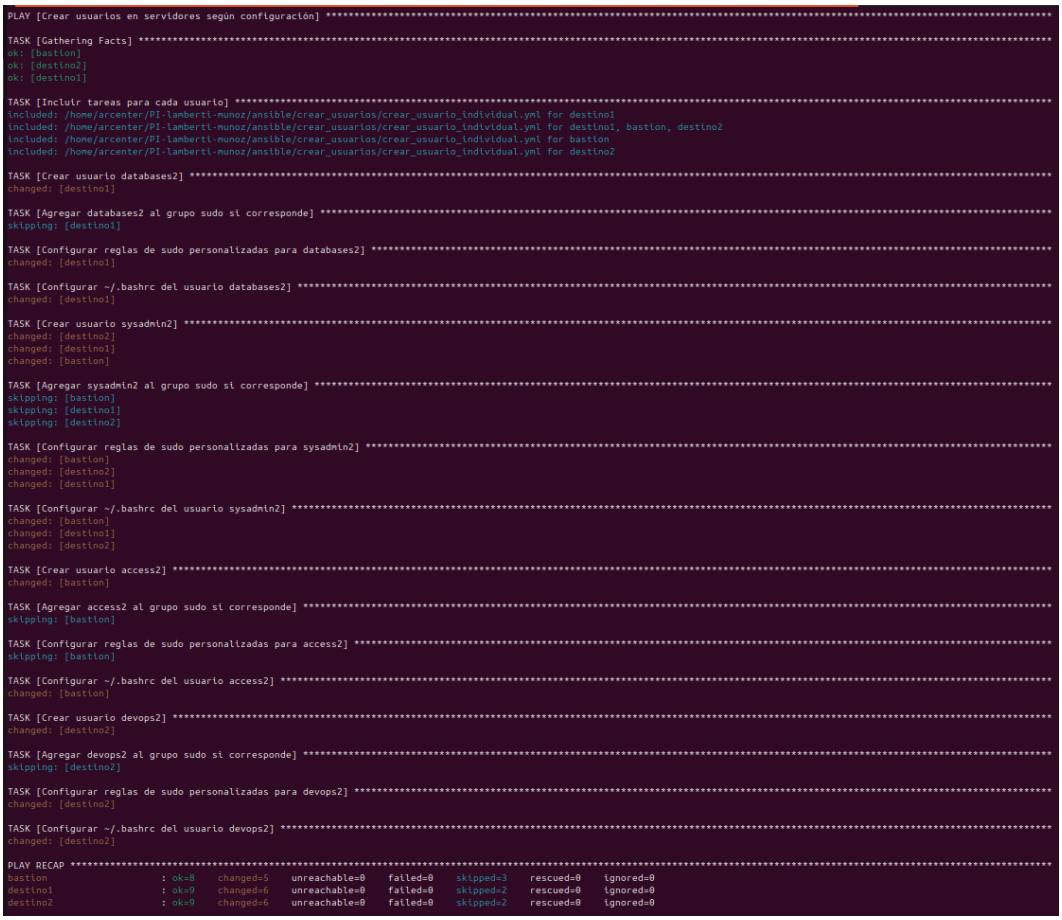

Figura 6.30: Resultado de la ejecución del playbook de creación de usuarios

Modificación de reglas de sudo Para actualizar las reglas de `sudo` asociadas a los usuarios existentes en el sistema, se ejecutó el siguiente playbook. Este proceso permite agregar o actualizar los comandos permitidos para cada usuario según los cambios en el archivo `sudo_rules.yml`.

```
ansible-playbook -i inventario.ini actualizar_sudoers.yml
```

Extracto de código 6.34: Ejecución del playbook de actualización de sudoers

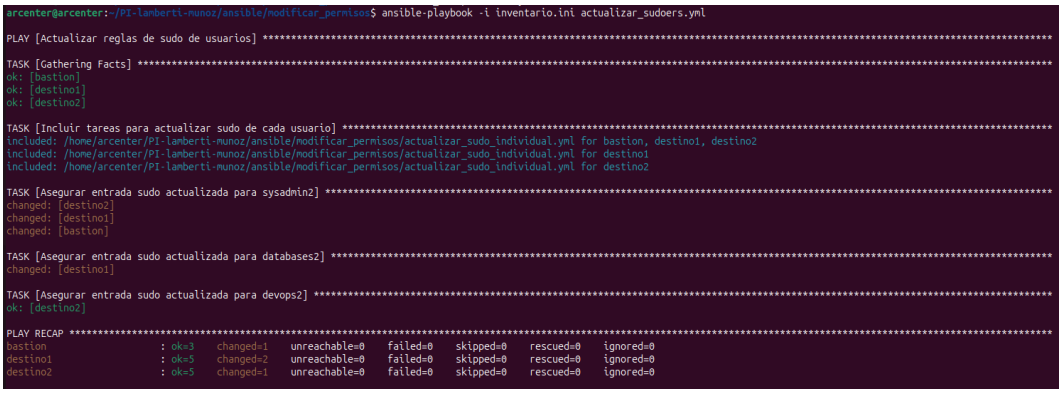

Figura 6.31: Resultado de la ejecución del playbook de modificación de reglas

Eliminación de usuarios y reglas asociadas Finalmente, para eliminar cuentas de usuarios y remover sus permisos de `sudo`, se utilizó el siguiente comando, el cual ejecuta las tareas del playbook correspondiente a la eliminación:

```
ansible-playbook -i inventario.ini eliminar_usuarios.yml
```

Extracto de código 6.35: Ejecución del playbook de eliminación de usuarios

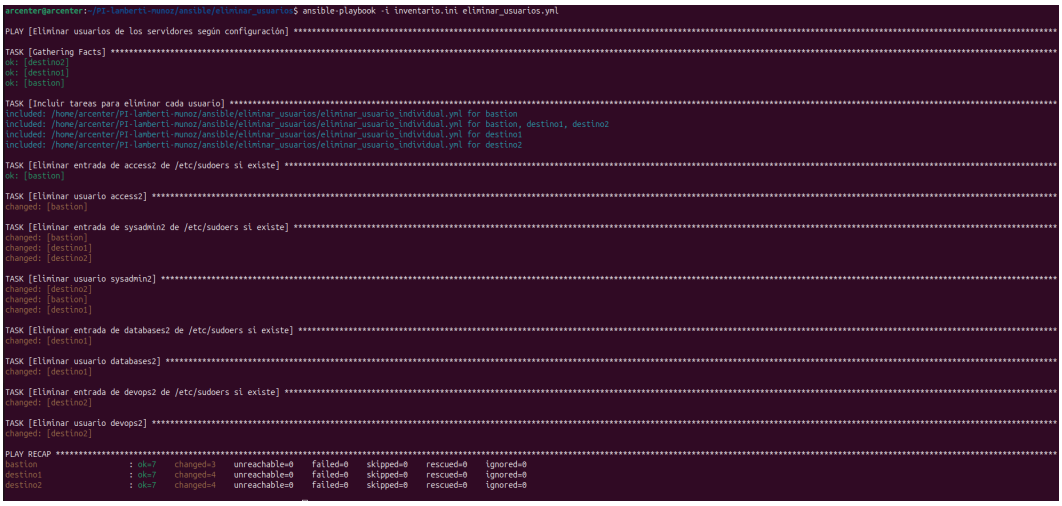

Figura 6.32: Resultado de la ejecución del playbook de eliminación de usuarios

A los fines de continuar con la implementación del sistema de autorización, se dejó preconfigurada la estructura básica de usuarios en cada servidor. En el nodo `bastion` se crearon los usuarios `access` y `sysadmin`; en `destino1`, los usuarios `databases` y `sysadmin`; y en `destino2`, los usuarios `devops` y `sysadmin`. Esta distribución responde a la segmentación de responsabilidades definida por roles y permite avanzar con las siguientes fases del control de acceso basado en certificados X.509.

6.4.4. Desarrollo de la capa de Autorización en auth-server

Una vez creados los roles en el gateway y los respectivos hosts, continuamos con el desarrollo de la capa de autorización en el servicio de autenticación desplegado en la iteración III.

El certificado con el que se autenticaba un cliente contenía como atributo `Role` donde se incorporaban dos o más roles, siendo `access` el rol obligatorio e indispensable en todos los casos.

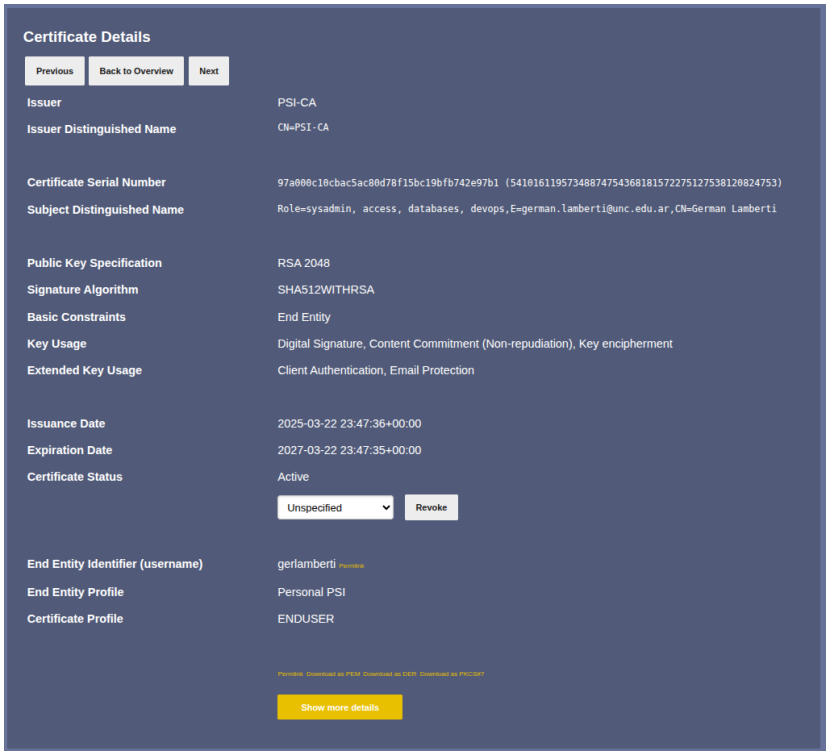

Figura 6.33: Certificado digital con roles

La lógica de autorización fue implementada en el microservicio `auth-server`, desarrollado con FastAPI. Este servicio expone un endpoint que recibe como parámetros el `SERIAL_ID` del certificado utilizado durante la conexión SSH, así como el nombre de usuario del sistema operativo que intenta acceder. A partir de estos datos, se realiza una serie de validaciones:

- Se verifica que el certificado no haya sido revocado mediante la función `is_revoked(serial_id)`.
- Se recupera el certificado con `get_certificate(serial_id)` y se comprueba que no esté expirado.
- Se extrae el atributo `role` del campo `Subject` del certificado, que puede contener múltiples roles separados por comas.
- Se valida que el nombre de usuario del sistema operativo (`username`) coincida con alguno de los roles autorizados en el certificado.

A continuación se presenta el código completo del archivo `authenticate_service.py`, que implementa esta lógica:

```python
import logging
from typing import Optional, Tuple

from app.domain.entities.authorized_keys import AuthorizedKeysBuilder
from app.domain.repositories.certificate_repository import \
    CertificateRepository
from pydantic import BaseModel


class AuthResponse(BaseModel):
    allowed: bool
    authorized_keys_entry: Optional[str] = None


class AuthenticateService:
    def __init__(self,
                 certificate_repository: CertificateRepository,
                 authorized_keys_builder: AuthorizedKeysBuilder,
                 logger: logging.Logger = logging.getLogger(__name__)):
        self.certificate_repository = certificate_repository
        self.authorized_keys_builder = authorized_keys_builder
        self.logger = logger

    def authenticate(self, serial_id: str, username: str) -> Tuple[AuthResponse, dict]:
        isRevoked, err = self.certificate_repository.is_revoked(serial_id)
        if err:
            return None, {"error": "is_revoked call failed", "detail": err}
        if isRevoked:
            return AuthResponse(allowed=False), None

        certificate, err = self.certificate_repository.get_certificate(
            serial_id)

        if err:
            return None, {"error": "get_certificate failed", "detail": err}

        if certificate.is_expired():
            self.logger.info("Certificate is expired")
            return AuthResponse(allowed=False), None

        self.logger.debug("Certificate role: %s",
                          certificate.subject_components["role"])
        self.logger.debug("Username: %s", username)

        # Control de roles
        roles_str = certificate.subject_components["role"]
        roles = [role.strip() for role in roles_str.split(",")]
        if username not in roles:
            return AuthResponse(allowed=False), None
        user_role = username

        try:
            authorized_keys_entry = self.authorized_keys_builder.build(
                certificate.subject_components["emailAddress"],
```

```
                certificate.subject_components["CN"],
                user_role,
                user_role,
                certificate.public_key
            )
        except Exception as e:
            return None, {"error": "authorized_keys_builder failed", "detail": str(e)}

        return AuthResponse(allowed=True, authorized_keys_entry=authorized_keys_entry),
    None
```

Extracto de código 6.36: Implementación completa de la capa de autorización en auth-server

A continuación se destacan los cambios clave introducidos en este módulo:

- **Validación de roles múltiples**: El atributo `role` del certificado puede contener más de un rol, separados por comas. Esta cadena es separada y convertida en una lista mediante:

```
roles = [role.strip() for role in roles_str.split(",")]

```

- **Comparación con el nombre de usuario**: Se exige que el usuario del sistema (`username`) coincida exactamente con uno de los roles definidos en el certificado. En caso contrario, el acceso es denegado:

```
if username not in roles:
    return AuthResponse(allowed=False), None

```

- **Generación de clave autorizada personalizada**: El rol validado es utilizado como parámetro en la construcción del campo `authorized_keys_entry`, facilitando el registro y trazabilidad del acceso autorizado.

Gracias a estas validaciones, el servicio de autenticación se transformó en un sistema completo de control de acceso basado en roles (RBAC), cumpliendo con los requerimientos funcionales del sistema y mejorando significativamente la seguridad en el entorno distribuido.

6.4.5. Prueba de integración del sistema RBAC

Extracción de clave privada y certificado digital

Para realizar una conexión SSH autenticada con un certificado digital, es necesario extraer desde el archivo `.p12` tanto la clave privada como el certificado, y ubicarlos adecuadamente en el directorio `~/.ssh/` del usuario. A su vez, se requiere el número de serie del certificado (`serial_id`) para su posterior verificación por parte del microservicio de autorización.

Extraer la clave privada del archivo `.p12`

```
openssl pkcs12 -in usuario.p12 -nocerts -nodes -out ~/.ssh/id_rsa_tesis
chmod 600 ~/.ssh/id_rsa_tesis
```

Extracto de código 6.37: Extracción de clave privada

Este comando genera la clave privada en formato PEM sin cifrado y establece los permisos adecuados para su uso por SSH.

Extraer el certificado digital

```
openssl pkcs12 -in usuario.p12 -clcerts -nokeys -out ~/.ssh/certificado.pem
```

Extracto de código 6.38: Extracción de certificado digital

Este certificado se utilizará para validar el rol del usuario y obtener información como el nombre común (CN) o la dirección de correo electrónico.

Obtener el número de serie del certificado (`serial_id`)

```
openssl x509 -in ~/.ssh/certificado.pem -serial -noout
```

Extracto de código 6.39: Obtención del serial del certificado

La salida de este comando será similar a:

```
serial=97a000c10cbac5ac80d78f15bc19bfb742e97b1
```

Este número de serie será utilizado por el servidor de autorización para verificar el estado del certificado y validar los roles asociados.

Acceso a los servidores mediante autenticación con certificado digital Una vez preparada la clave privada y extraído el número de serie del certificado digital, se procedió a realizar conexiones reales a los servidores, utilizando el sistema de autorización desarrollado. En todos los casos se empleó como mecanismo de autenticación la clave privada asociada al certificado X.509 emitido, y como filtro de autorización, el servicio `auth-server` que valida el rol del usuario técnico según su certificado.

Acceso al servidor Gateway (Bastion) Para establecer una sesión SSH directamente al servidor bastión como el usuario `sysadmin`, se ejecutó el siguiente comando:

```
ssh -i ~/.ssh/id_rsa_tesis sysadmin@172.18.66.248
```

Extracto de código 6.40: Acceso directo al servidor Gateway

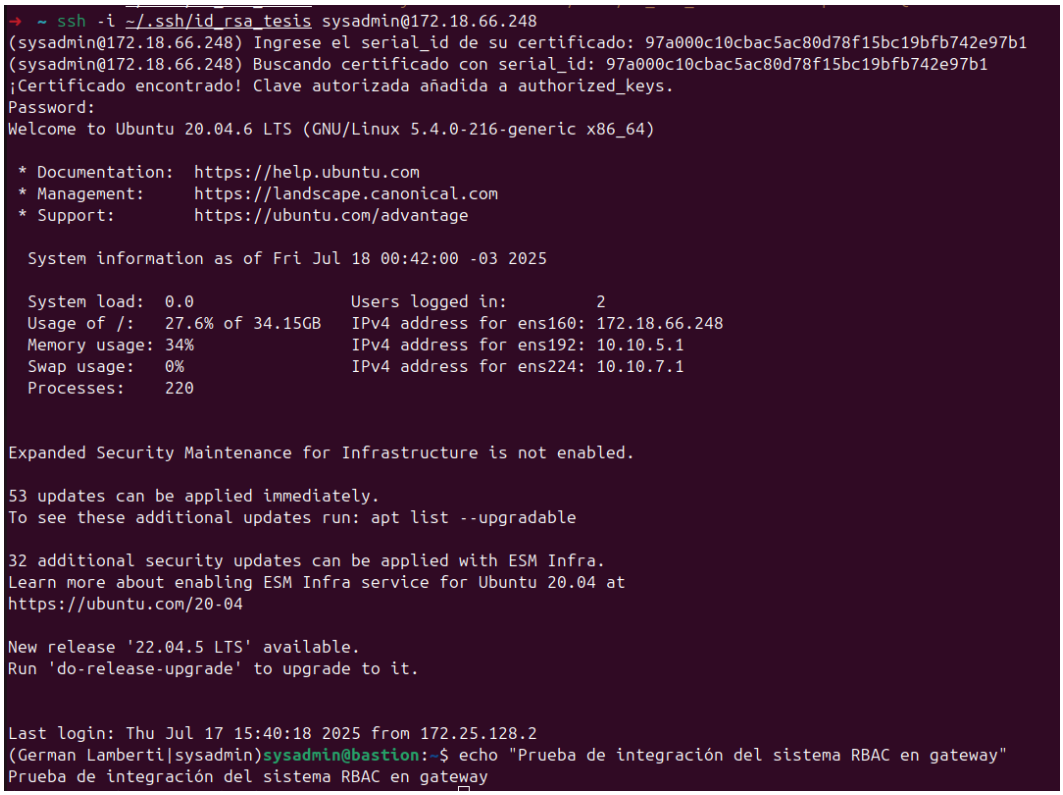

Figura 6.34: Conexión SSH exitosa al servidor Gateway como `sysadmin`

Como puede observarse en la Figura 6.34, el acceso fue concedido, permitiendo operar en el servidor bastión con los privilegios definidos para el rol `sysadmin`.

Acceso al servidor Destino 1 vía Gateway Para acceder al servidor `destino1` como el usuario `databases`, se utilizó la funcionalidad de `ProxyCommand` de SSH, la cual permite establecer un túnel a través del servidor bastión:

```
ssh -i ~/.ssh/id_rsa_tesis -o "ProxyCommand ssh -i ~/.ssh/id_rsa_tesis -W %h:%p
    access@172.18.66.248" databases@10.10.7.2
```

Extracto de código 6.41: Acceso SSH a destino1 a través del Gateway

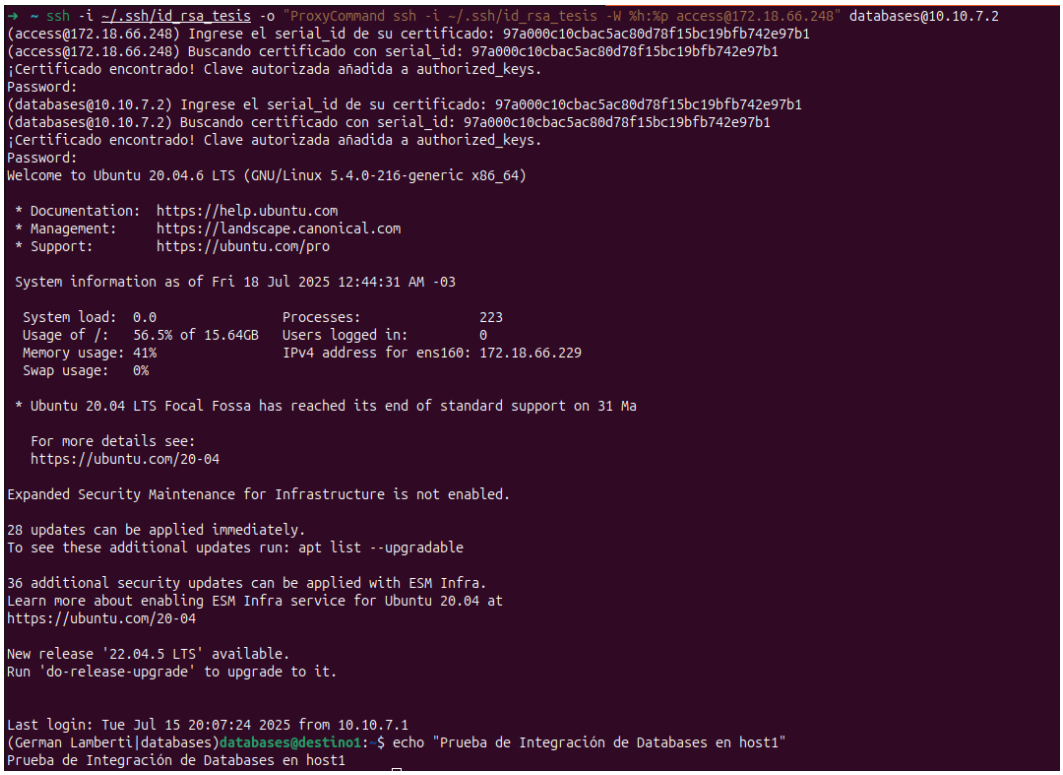

Figura 6.35: Conexión SSH exitosa al servidor destino1 como `databases` vía Gateway

En la Figura 6.35 se observa cómo, luego de ser autenticado y autorizado como `access` en el bastión, se establece una segunda conexión al servidor destino utilizando el usuario `databases`, autorizado por su certificado.

Acceso al servidor Destino 2 vía Gateway Del mismo modo, para acceder al servidor `destino2` como usuario `devops`, se aplicó el siguiente comando:

```
ssh -i ~/.ssh/id_rsa_tesis -o "ProxyCommand ssh -i ~/.ssh/id_rsa_tesis -W %h:%p
    access@172.18.66.248" devops@10.10.7.4
```

Extracto de código 6.42: Acceso SSH a destino2 a través del Gateway

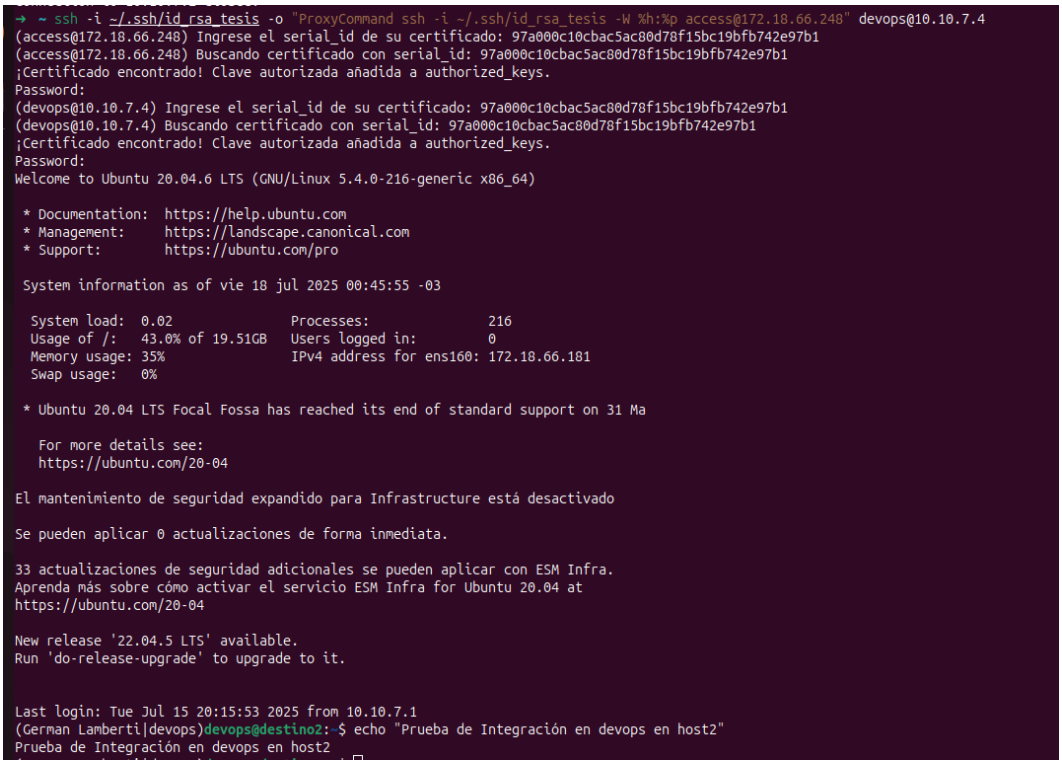

Figura 6.36: Conexión SSH exitosa al servidor destino2 como `devops` vía Gateway

La Figura 6.36 muestra el resultado exitoso de este proceso evidencia el funcionamiento completo del sistema de autenticación y autorización propuesto, donde el acceso a los distintos nodos se encuentra condicionado por el certificado digital presentado y el rol correspondiente validado por el `auth-server`.

6.5. Iteración V: Implementación del sistema de auditoria y monitoreo

6.5.1. Captura de comandos

Para cumplir el requerimiento funcional RF5 y RF6, se implementó un mecanismo de registro de operaciones. Para lograr esto, se configuró el archivo /.bashrc del directorio home para cada usuario de sistema al momento de ser creado, de modo que cada comando ejecutado por un cliente quede registrado junto con su marca temporal e información divisional útil.

La configuración establece el formato de tiempo para los comandos del historial mediante la variable `HISTTIMEFORMAT`, e incorpora una lógica dentro de `PROMPT_COMMAND` para verificar si el comando actual ya ha sido registrado. En caso de ser nuevo, extrae el número de secuencia, la fecha y hora, y el contenido del comando. Esta información se concatena junto al identificador del usuario (obtenido del certificado o del nombre remoto agregado junto a la clave pública como variable) y la IP de conexión.

Los registros generados por esta configuración se almacenan en el archivo `/var/log/comandos_history`, disponible para su posterior análisis o correlación de eventos mediante herramientas como Filebeat y Elasticsearch. Para permitir el uso de las variables de entorno en sesiones SSH, fue necesario habilitar las opciones PermitUserEnvironment y ExposeAuthInfo en el archivo de configuración del servicio SSH (`/etc/ssh/sshd_config`).

```
1
2  export HISTTIMEFORMAT="%F %T "
3
4  if [ -n "$SSH_USER_AUTH" ]; then
5      export CERT=$(cat $SSH_USER_AUTH | awk '{print $2 " " $3}' | ssh-keygen -Lf - 2> /
       dev/null | awk '/Key ID/ {gsub(/", "", $0); print $3}')
6      if [ -n "$CERT" ]; then
7          export PS1="($CERT)$PS1"
8      else
9          export CERT="$REMOTEUSER"
10         export PS1="($CERT)$PS1"
11     fi
12     IP_CONNECTION=$(echo $SSH_CONNECTION | awk '{print $1}')
13 fi
14
15 PROMPT_COMMAND="${PROMPT_COMMAND:+$PROMPT_COMMAND ; }"'{
16     current_hist_num=$(history 1 | awk "{print \$1}")
17     last_hist_num=$(awk "{print \$4}" <(tail -n 1 /var/log/comandos_history))
18     if [ "$current_hist_num" != "$last_hist_num" ]; then
19         current_history=$(history 1)
20         vsecuencia=$(echo "$current_history" | awk "{print \$1}")
21         vtimestamp=$(echo "$current_history" | awk "{print \$2, \$3}")
22         vcomando=$(echo "$current_history" | awk "{\$1=\"\"; \$2=\"\"; \$3=\"\"; print \
       $0}" | sed "s/^ *//")
23         echo "$CERT $USER $IP_CONNECTION $vsecuencia $vtimestamp $vcomando" >> /var/log/
       comandos_history
24     fi
25 }'
```

Extracto de código 6.43: Captura de comando

Estas opciones permiten exponer información del entorno de autenticación del usuario a la sesión interactiva. Al finalizar la configuración se reinició el servicio SSH para aplicar los cambios.

Esta solución permite asociar de forma precisa cada operación del sistema a un cliente autenticado y a su correspondiente certificado digital, particularmente útil en escenarios donde múltiples clientes acceden al servidor utilizando el mismo usuario de sistema, contribuyendo de este modo a garantizar la trazabilidad individual de las acciones realizadas.

Elasticsearch

A continuación se describe el proceso de instalación e implementación de Elasticsearch, uno de los componentes centrales del sistema de monitoreo y análisis de

eventos. Elasticsearch es un motor de búsqueda y análisis distribuido que permite almacenar, indexar y consultar grandes volúmenes de datos en tiempo real. Elasticsearch se desplegó en el servidor de administración general y constituye la base para el procesamiento centralizado de los registros enviados por Filebeat.

Instalación e implementación de Elasticsearch

El procedimiento se inició importando la clave de firma pública de Elasticsearch, necesaria para verificar la autenticidad de los paquetes descargados desde el repositorio:

```
wget -qO - https://artifacts.elastic.co/GPG-KEY-elasticsearch | sudo gpg --dearmor -o /
    usr/share/keyrings/elasticsearch-keyring.gpg
```

Posteriormente, se instaló el paquete `apt-transport-https`, requerido para acceder a repositorios HTTPS de forma segura:

```
sudo apt-get install apt-transport-https
```

A continuación se agregó el repositorio oficial de Elastic para la versión 7.x en el archivo correspondiente:

```
echo "deb [signed-by=/usr/share/keyrings/elasticsearch-keyring.gpg] https://artifacts.
    elastic.co/packages/7.x/apt stable main" | sudo tee /etc/apt/sources.list.d/elastic
    -7.x.list
```

Una vez definido el repositorio, se procedió a actualizar la lista de paquetes e instalar Elasticsearch mediante APT:

```
sudo apt-get update && sudo apt-get install elasticsearch
```

6.5.2. Configuración y puesta en marcha

El servicio de Elasticsearch fue configurado para que se inicie automáticamente junto con el sistema operativo. Para ello, se ejecutaron los siguientes comandos:

```
sudo /bin/systemctl daemon-reload
sudo /bin/systemctl enable elasticsearch.service
```

La gestión del servicio puede realizarse con los siguientes comandos:

```
sudo systemctl start elasticsearch.service
sudo systemctl stop elasticsearch.service
```

La configuración principal de Elasticsearch se encuentra en el archivo `/etc/elasticsearch/elasticsearch.yml`, este archivo está escrito en formato YAML.

Con Elasticsearch correctamente instalado y en ejecución, el sistema quedó preparado para recibir, almacenar e indexar los registros operativos enviados desde los agentes Filebeat, facilitando así las consultas y visualizaciones posteriores mediante Kibana.

Instalación e implementación de Kibana

Kibana es una herramienta que proporciona una interfaz de usuario para visualizar, explorar y analizar los datos almacenados en Elasticsearch.

Su instalación y configuración es similar a la de Elasticsearch. Una práctica recomendada es utilizar la misma versión que la utilizada en Elasticsearch, a fin de evitar incompatibilidades.

Importación de la clave PGP de Elastic

Como primer paso, se importó la clave de firma pública de Elastic para garantizar la autenticidad de los paquetes instalados. Esto se realizó con el siguiente comando:

```
wget -qO - https://artifacts.elastic.co/GPG-KEY-elasticsearch | sudo gpg --dearmor -o /
    usr/share/keyrings/elasticsearch-keyring.gpg
```

Configuración del repositorio APT

Se instaló el paquete `apt-transport-https`

```
sudo apt-get install apt-transport-https
```

Luego, se agregó la definición del repositorio oficial de Elastic en el archivo correspondiente:

```
echo "deb [signed-by=/usr/share/keyrings/elasticsearch-keyring.gpg] https://artifacts.
    elastic.co/packages/9.x/apt stable main" | sudo tee /etc/apt/sources.list.d/elastic
    -9.x.list
```

Instalación de Kibana

Con el repositorio configurado, se actualizó el índice local y se procedió a instalar Kibana mediante:

```
sudo apt-get update && sudo apt-get install kibana
```

Configuración para acceso externo

Para permitir el acceso remoto a Kibana desde otros dispositivos, fue necesario modificar su archivo de configuración ubicado en `/etc/kibana/kibana.yml`. Se descomentó y modificó la línea correspondiente al parámetro `server.host`, estableciendo la dirección `0.0.0.0`, lo que permite que Kibana escuche conexiones entrantes en todas las interfaces de red disponibles:

```
server.host: 0.0.0.0
```

Habilitación y ejecución del servicio Kibana

Finalmente, se configuró el servicio Kibana para que se inicie automáticamente al arrancar el sistema, ejecutando los siguientes comandos:

```
sudo /bin/systemctl daemon-reload
sudo /bin/systemctl enable kibana.service
```

El servicio puede iniciarse y detenerse al ejecutar:

```
sudo systemctl start kibana.service
sudo systemctl stop kibana.service
```

Para verificar el estado del servicio y confirmar que se ejecuta correctamente, se utiliza el siguiente comando:

```
sudo systemctl status kibana
```

En la salida de este comando se muestra la URL con la dirección de host, desde donde se puede acceder a la interfaz web de Kibana.

Instalación e implementación de Filebeat

Filebeat es un agente ligero que forma parte del Elastic Stack y se encarga de recolectar y enviar registros de eventos (logs) desde los servidores hacia Elasticsearch.

Se registran de forma persistente todos los comandos ingresados por el usuario durante las sesiones. Estos eventos se almacenan en un archivo de registro ubicado en el directorio de logs del sistema, específicamente en:

```
/var/logs/comandos_history
```

En cuanto a su protección, el archivo de registro posee permisos restringidos de modificación únicamente para el usuario root, lo que previene alteraciones no autorizadas y asegura la integridad de la información registrada.

En este proyecto, Filebeat fue instalado en todos los servidores para garantizar una cobertura completa del sistema.

Instalación

La instalación de Filebeat comienza con la descarga e instalación de la clave de firma pública de Elasticsearch que garantiza la autenticidad de los paquetes:

```
wget -qO - https://artifacts.elastic.co/GPG-KEY-elasticsearch | sudo apt-key add -
```

Luego, se instalaron los paquetes necesarios para permitir el uso del protocolo HTTPS con APT:

```
sudo apt-get install apt-transport-https
```

Posteriormente, se guardó la definición del repositorio Beats en el archivo correspondiente:

```
echo "deb https://artifacts.elastic.co/packages/7.x/apt stable main" | sudo tee -a /etc/
    apt/sources.list.d/elastic-7.x.list
```

Con el repositorio correctamente configurado, se ejecutó una actualización del índice de paquetes y se procedió a instalar Filebeat:

```
sudo apt-get update && sudo apt-get install filebeat
```

Configuración del servicio

Para que Filebeat se ejecute automáticamente en cada arranque del sistema, se configuró el servicio de la siguiente manera:

```
sudo systemctl daemon-reload
sudo systemctl enable filebeat.service
```

Una vez instalado y activo, Filebeat puede ser configurado para enviar registros específicos a Elasticsearch.

Parámetros de configuración del archivo filebeat.yml

```
# ================================== General ===================================

# Filebeat Name
name: destino1

# ================================== Outputs ===================================

output.elasticsearch:
  hosts: ["172.18.66.138:9200"]

# ============================== Filebeat inputs ===============================

#filebeat.inputs:
#- type: log
#  paths:
#    - /var/log/comandos_history

filebeat.inputs:
  - type: log
    enabled: true
    paths:
      - /var/log/comandos_history
    fields:
      log_type: "comandos_history"
    fields_under_root: true
    processors:
      - dissect:
          tokenizer: "%{cliente} %{usuario_sistema} %{ip_origen} %{id} %{+timestamp}
    %{+timestamp} %{comando}"
```

```
29          field: "message"
30          target_prefix: "parsed"
```

Extracto de código 6.44: Achivo de configuración de filebeat

Este archivo `filebeat.yml` permite:

- Identificar el origen del log.
- Capturar los comandos ejecutados por los usuarios.
- Analizar cada línea de log de forma estructurada.
- Enviar los datos a Elasticsearch para su posterior visualización en Kibana.

La configuración de filebeat es común en los servidores destino. A continuación una descripción de los parámetros

`name`: Define un nombre único para el agente Filebeat. En este caso, el nombre es `destino1`, lo cual es útil para identificar desde qué servidor provienen los logs cuando son enviados a Elasticsearch. Esto es especialmente importante en entornos con múltiples servidores.

`output.elasticsearch`: Configura la salida de Filebeat. Se enviará los logs directamente al host de administración general ubicado en la IP `172.18.66.138` y el puerto `9200`. Se centralizan los logs para visualizarlos en Kibana.

`filebeat.inputs`: Aquí se indica que Filebeat debe monitorear archivos de tipo log, y se especifica la ruta del archivo a recolectar: `/var/log/comandos_history`, que es el archivo donde se registran los comandos ejecutados por los usuarios, como se configuró previamente en el `.bashrc`.

`fields`: Agrega metadatos personalizados al log, en este caso un campo `log_type` con valor çomandos_history". Esto ayuda a distinguir este tipo de log en Elasticsearch y en Kibana.

`fields_under_root: true`: Indica que estos campos no deben ir anidados en una subestructura, sino directamente bajo la raíz del evento.

Se utiliza un procesador `dissect` para extraer campos específicos del contenido del log. Se divide cada línea del log en varias partes estructuradas:

- **cliente**: identificador del usuario.
- **usuario_sistema**: nombre del usuario del sistema.
- **ip_origen**: IP desde la que se realizó la conexión SSH.
- **id**: número de secuencia del comando.
- **timestamp**: fecha y hora.

La salida de esta operación se almacena bajo el prefijo parsed, lo que permite que los campos estén organizados y listos para consultas y visualizaciones en Kibana.

Visualización de Actividad mediante Dashboards en Kibana

Como parte de la implementación del sistema de monitoreo, se desarrollaron dos dashboards personalizados en Kibana con el objetivo de visualizar y analizar los eventos recolectados por Filebeat y almacenados en Elasticsearch.

Dashboard 1: Accesos exitosos y fallidos por usuario

Este primer panel fue diseñado para identificar intentos de acceso exitosos y fallidos hacia los servidores destino1 y destino2. El panel permite:

- Correlacionar usuarios con altas tasas de intentos fallidos de autenticación.
- Detectar posibles ataques de fuerza bruta o errores reiterados de autenticación.
- Analizar tendencias de accesos fallidos dirigidos a un servidor específico.

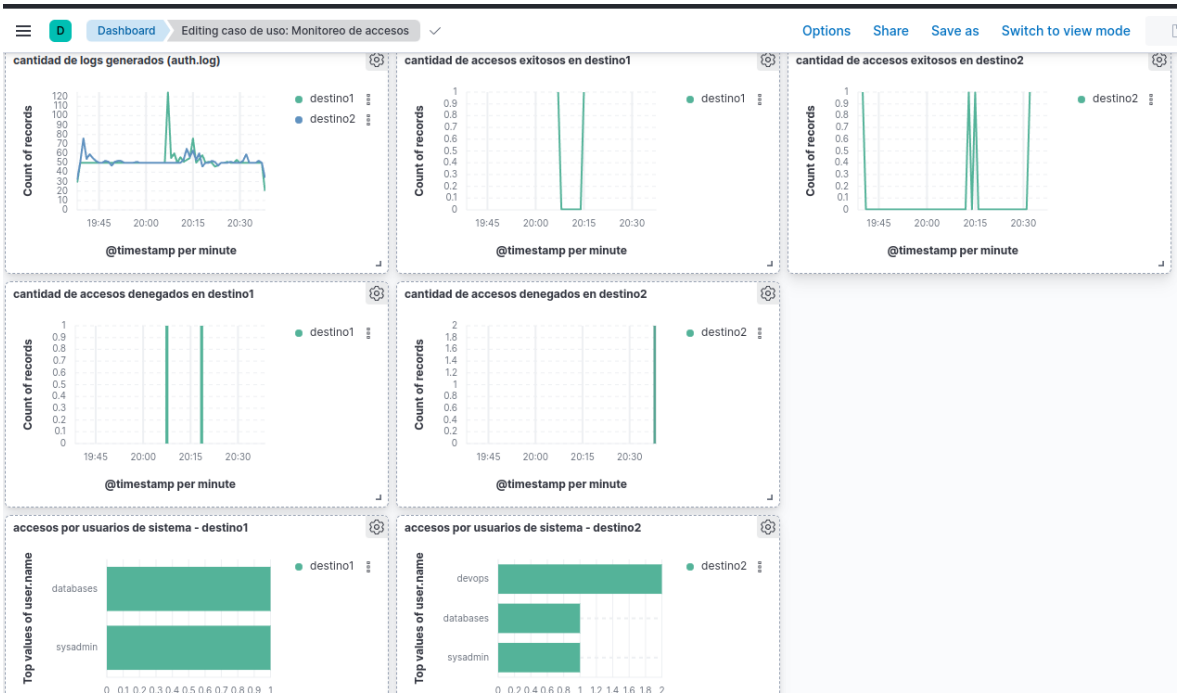

Figura 6.37: dashboad de monitoreo de accesos

Esta visualización es útil tanto para tareas de auditoría como para el fortalecimiento de políticas de acceso y autenticación.

Dashboard 2: Comandos ejecutados por usuario

El segundo panel se enfoca en el monitoreo de comandos ejecutados por los usuarios en ambos servidores. Se construyó con el objetivo de:

- Identificar qué comandos ejecuta cada usuario.
- Comparar la actividad entre destino1 y destino2 para detectar inconsistencias o desvíos en la operación normal.

- Este dashboard permite identificar a usuarios que estén ejecutando comandos no autorizados o fuera del alcance de su rol, contribuyendo directamente al cumplimiento del requerimiento de auditoría y control operativo.

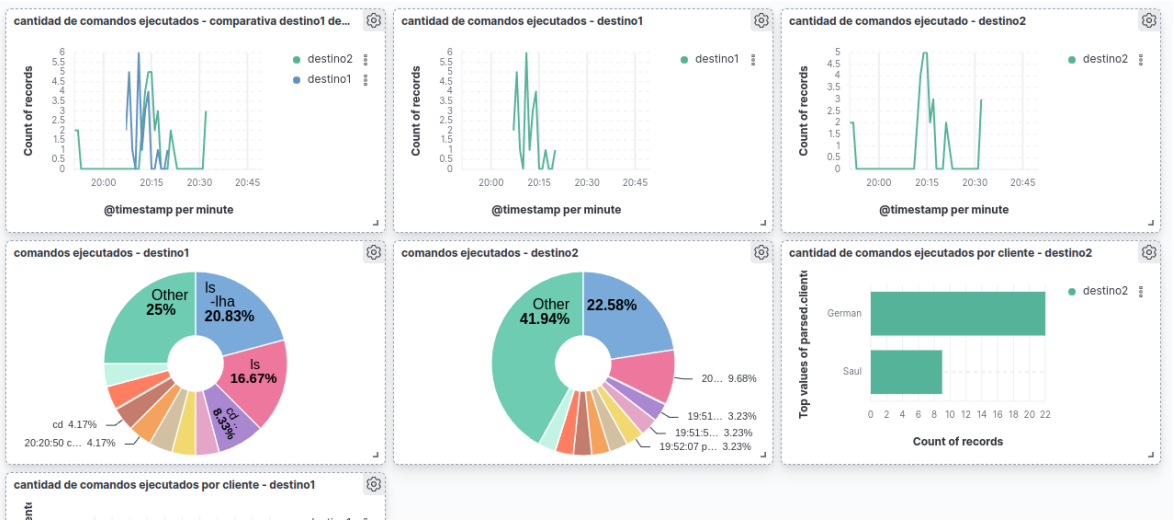

Figura 6.38: dashboad de comandos ejecutados

7. Conclusiones

7.1. Evaluación de resultados

A lo largo del desarrollo se contextualizó el problema de control de acceso a servidores y se establecieron los objetivos generales y específicos que guiaron el desarrollo del proyecto. Se justificó el uso de certificados digitales como mecanismo principal de autenticación y se planteó la hipótesis de solución.

Se desarrolló un marco teórico que sustentó técnicamente la solución, abordando conceptos clave como PKI, certificados X.509, autenticación SSH, control de acceso, módulos PAM, y tecnologías como Docker, Elasticsearch, Kibana y Ansible.

Se detallaron los stakeholders del sistema, los requerimientos funcionales y no funcionales, el análisis de riesgos y la planificación de tareas, estableciendo así una hoja de ruta clara para el diseño e implementación del sistema.

Se presentó la arquitectura lógica del sistema, modelos estáticos y dinámicos, y la topología de red. Se propuso un diseño modular, con separación de funciones y un flujo seguro de autenticación distribuida mediante doble validación.

Se justificó la selección de hardware y software, priorizando tecnologías libres, estables y con soporte comunitario. Se eligió una infraestructura virtualizada con VMware ESXi y herramientas como EJBCA, SoftHSM2, Filebeat, Elasticsearch, Kibana y Ansible.

Se documentaron detalladamente las acciones realizadas en cada fase del proyecto, desde el diseño hasta las pruebas finales, reflejando una ejecución metódica y alineada con la planificación inicial.

7.2. Desafíos y Soluciones Específicas

El desarrollo de este proyecto no siguió un camino lineal; a lo largo del proceso surgieron desafíos técnicos de distinta naturaleza que requirieron análisis y soluciones específicas. Entre ellos, resulta pertinente destacar aquellos que tuvieron mayor impacto en la implementación final.

En primer lugar, durante la configuración de las máquinas virtuales (VMs) destinadas al despliegue del laboratorio de operaciones, se identificó una limitación en la capacidad del hardware disponible. Esta restricción llevó a reducir a dos la cantidad de hosts empleados como servidores de destino, optimizando así los recursos sin comprometer la funcionalidad del entorno.

Asimismo, en el análisis de la conexión mediante OpenSSH se constató que el sistema no admitía, de forma nativa, el uso de certificados X.509, dado que emplea su propio formato de credenciales. Para superar esta limitación, se desarrolló una solución

basada en la integración de PAM, la cual permite validar certificados X.509 y asegurar la autenticidad de los accesos.

Otro desafío significativo fue la comunicación entre Filebeat y Elasticsearch. Inicialmente, Elasticsearch se encontraba desplegado en un contenedor Docker, lo que ocasionó conflictos en la asignación de direcciones IP dentro de la red de Docker y un consumo de recursos superior al previsto. Se resolvió mediante la instalación nativa de Elasticsearch en el sistema, eliminando los problemas de red y mejorando el rendimiento general.

Por último, la adopción de Ansible como motor de configuración de los hosts evidenció su gran potencial para la gestión de usuarios, tanto en su creación, modificación como eliminación. No obstante, el verdadero reto consistió en diseñar una estructura modular y recursiva de playbooks que permitiera aplicar configuraciones de manera consistente y eficiente en cada host.

Estos casos representan solo una parte de los retos técnicos enfrentados y resueltos durante el desarrollo, reflejando la capacidad de adaptación y la solidez de las soluciones implementadas en el marco del proyecto.

7.3. Reflexión Final

El sistema propuesto alcanzó plenamente los requerimientos funcionales y no funcionales establecidos en las fases iniciales del proyecto, consolidándose como una solución técnicamente viable, escalable y segura. La incorporación de módulos PAM personalizados, certificados X.509 emitidos por una infraestructura de clave pública basada en EJBCA y la automatización de tareas administrativas mediante Ansible garantizó niveles elevados de trazabilidad, seguridad y mantenibilidad. La modularidad de la arquitectura, junto con el empleo de estándares abiertos y herramientas de software libre, refuerza su potencial de adaptación a entornos organizacionales diversos, demostrando que es posible alcanzar altos niveles de control y visibilidad en la gestión de accesos remotos sin depender de soluciones propietarias.

El desarrollo implementado materializa un modelo de seguridad multinivel en el que convergen autenticación, autorización y registro robusto sustentados en certificados digitales; canales de comunicación seguros y cifrados a través de SSH; mecanismos de aislamiento y contención a nivel de hipervisor mediante VMware ESXi; y una separación arquitectónica de las funciones de seguridad en componentes distribuidos. Esta integración de capas defensivas constituye un enfoque de defensa en profundidad que reduce significativamente la superficie de ataque y aumenta la resiliencia global del sistema. En conjunto, los resultados obtenidos validan un marco integral para la protección de servidores virtualizados y sus accesos, ofreciendo una base sólida sobre la cual construir futuras mejoras y adaptaciones en entornos donde la seguridad y la trazabilidad resulten críticas.

8. Perspectivas Futuras

Si bien el sistema desarrollado logró cumplir los objetivos planteados, se identificaron diversas áreas susceptibles de mejora y expansión. A continuación, se detallan las oportunidades detectadas durante el diseño, implementación y evaluación del proyecto.

- **Seguridad del HSM virtual:** Actualmente se utiliza SoftHSM2 para generar el crytoToken de EJBCA, una solución basada en software. En un entorno productivo, se recomienda sustituirlo por un HSM físico certificado (FIPS 140-2 nivel 3) para mejorar la protección de las claves privadas.
- **Gestión del ciclo de vida de certificados:** Podría implementarse una herramienta complementaria para la gestión automatizada de expiraciones, renovaciones y revocaciones, incluyendo notificaciones automáticas por correo electrónico.
- **Mejora de la interfaz de administración:** La configuración de usuarios y roles se realiza actualmente mediante archivos YAML y ejecución de playbooks. Una interfaz gráfica de administración podría facilitar la adopción por parte de administradores con menor experiencia en línea de comandos.
- **Integración con doble factor de autenticación (2FA):** Para entornos con mayores requerimientos de seguridad, se sugiere integrar mecanismos de 2FA que complementen la autenticación basada en certificados.
- **Alta disponibilidad de servicios críticos:** Actualmente los servicios como EJBCA, Elasticsearch y Kibana están desplegados en instancias únicas. Para entornos de producción, se recomienda implementar redundancia y balanceo de carga.
- **Auditoría más granular:** Ampliar la recolección de logs para incluir información adicional sobre comandos ejecutados, duración de sesión y correlación entre eventos podría mejorar la capacidad de auditoría y detección de anomalías.
- **Soporte para usuarios móviles o externos:** Actualmente el acceso está limitado a usuarios técnicos conectados vía SSH. Una posible expansión del sistema podría incluir portales de acceso web autenticados con certificados.
- **Migración a una plataforma de virtualización de código abierto:** Una oportunidad de mejora consiste en evaluar, a mediano plazo, la migración hacia una solución de virtualización open source, como Proxmox VE o XCP-ng, que permita prescindir de software con licenciamiento comercial. Esta transición favorecería la alineación total con los principios de software libre establecidos en los requerimientos no funcionales, manteniendo al mismo tiempo la robustez y flexibilidad necesarias para la operación de los servicios virtualizados.
- **Implementación de prácticas de hardening:** Una posible mejora futura es la incorporación de prácticas de hardening en los servidores y sistemas, orientadas a reducir la superficie de ataque y fortalecer la seguridad del entorno. Esto incluiría la desactivación de servicios innecesarios, la aplicación de configura-

ciones seguras en SSH, el endurecimiento de permisos en archivos críticos y la implementación de mecanismos de detección temprana de intrusiones.

- **Protección contra ataques en SSH:** Se recomienda la incorporación de una herramienta de código abierto que permita detectar y mitigar intentos de acceso no autorizados al servicio SSH. Una posible opción es File2ban, que analiza los registros del sistema en busca de intentos fallidos de autenticación y bloquea temporalmente las direcciones IP ofensivas. La implementación de este tipo de soluciones contribuiría a reforzar la seguridad del servicio y reducir la exposición a ataques automatizados.

Estas oportunidades de mejora pueden guiar futuras iteraciones del sistema, adaptándolo a nuevos escenarios y elevando su nivel de robustez y funcionalidad.

9. Glosario

AC (Autoridad de Certificación): Entidad responsable de emitir, renovar y revocar certificados digitales. Verifica la identidad de los solicitantes y firma los certificados para garantizar su validez.

Alta de usuario técnico: Proceso de incorporación de un nuevo operador técnico al sistema, que incluye la generación de credenciales digitales, validación de identidad y configuración de roles de acceso.

Algoritmos de curvas elípticas (ECC): Son algoritmos criptográficos basados en las propiedades matemáticas de las curvas elípticas sobre campos finitos. Se utilizan para generar claves más pequeñas que otros sistemas (como RSA) con un nivel equivalente de seguridad, lo que los hace más eficientes en términos de procesamiento y uso de recursos. Se emplean en cifrado, firma digital y establecimiento de claves seguras.

Algoritmos post-cuánticos: Son algoritmos criptográficos diseñados para resistir ataques de computadoras cuánticas, que podrían romper gran parte de la criptografía clásica actual (como RSA o ECC). Estos algoritmos se basan en problemas matemáticos diferentes a los vulnerables a la computación cuántica, como retículas (lattices), códigos de corrección de errores o funciones hash, y buscan garantizar la seguridad a largo plazo en la era poscuántica.

Ansible: Herramienta de automatización de configuración y orquestación de tareas en sistemas. Permite gestionar servidores de manera centralizada y sin necesidad de instalar agentes en los nodos administrados.

AR (Autoridad de Registro): Entidad intermedia dentro de una infraestructura PKI encargada de validar la identidad de los usuarios antes de que sus certificados sean emitidos por la AC.

ARCenter: Subsistema del diseño propuesto encargado de aplicar políticas de control de acceso, centralizar la recolección de logs y visualizarlos mediante herramientas como Kibana.

Auditoría: Proceso de supervisión y registro de las actividades de los usuarios y del sistema para garantizar la trazabilidad y cumplimiento de políticas de seguridad.

Bash History (`.bash_history`): Archivo de historial de comandos ejecutados por un usuario en una terminal de sistemas basados en UNIX/Linux.

Bastion (Gateway): Servidor intermediario entre la red externa e interna. Funciona como único punto de acceso hacia los servidores destino, aplicando filtros de autenticación y autorización.

CA autofirmada: Certificado raíz que una Autoridad de Certificación genera y firma por sí misma. Se utiliza como punto de confianza inicial en una PKI.

Certificado digital (X.509): Documento electrónico que asocia una clave pública

con la identidad de su titular, y que ha sido firmado por una AC. Permite la autenticación, el cifrado y la firma digital.

CRL (Certificate Revocation List): Lista publicada por una AC que contiene los números de serie de los certificados revocados y que ya no deben considerarse válidos.

CryptoToken: Objeto lógico que encapsula y protege el acceso a claves privadas dentro de un módulo de seguridad (como SoftHSM2).

Docker / Docker Compose: Herramientas que permiten contenerizar aplicaciones y servicios, asegurando su despliegue reproducible, portátil y escalable en diferentes entornos.

Elasticsearch: Motor de búsqueda y análisis distribuido utilizado para almacenar y consultar grandes volúmenes de datos, como logs de eventos del sistema.

Elastic Stack: Conjunto de herramientas de software compuesto por Elasticsearch, Logstash, Kibana y Beats (en este caso Filebeat), orientado a la observabilidad, búsqueda y visualización de datos en tiempo real.

EJBCA (Enterprise JavaBeans Certificate Authority): Framework de código abierto para la implementación de una infraestructura PKI. Ofrece funcionalidades de emisión y gestión de certificados, OCSP, CRL y API REST.

Entidad Final: Usuario o sistema que recibe un certificado digital emitido por la PKI y lo utiliza para autenticarse ante otros sistemas.

Filebeat: Agente ligero de recolección de logs que envía datos desde los servidores hacia Elasticsearch u otros destinos para su posterior análisis.

Firewall: Componente de seguridad que controla el tráfico de red entrante y saliente en función de reglas definidas.

Glosario: Sección de una tesis que explica los términos técnicos más relevantes utilizados a lo largo del documento.

Hardening: También llamado endurecimiento o robustecimiento, se refiere al proceso de fortalecer un sistema informático para hacerlo más resistente a ataques y vulnerabilidades. Implica la implementación de medidas de seguridad para reducir la superficie de ataque.

Harvester: En Filebeat, un harvester es el proceso interno encargado de leer un archivo de log línea por línea y enviarlo al siguiente componente de la cadena de procesamiento.

HSM (Hardware Security Module): Dispositivo físico o software especializado que permite almacenar claves criptográficas de forma segura y realizar operaciones criptográficas sin exponer las claves privadas.

Host destino: Servidor interno al que acceden los usuarios técnicos para realizar tareas operativas. Su acceso está protegido por políticas de autenticación y autorización.

Hypervisor: Software que permite crear y administrar máquinas virtuales sobre hardware físico. En este proyecto se utilizó VMware ESXi como hipervisor de tipo 1.

Interfaz de red: Adaptador físico o lógico que permite a una máquina conectarse a una red. En entornos virtualizados, puede haber múltiples interfaces asociadas a diferentes redes virtuales.

Kibana: Herramienta de visualización de datos que se integra con Elasticsearch para crear paneles (dashboards) e informes sobre logs y métricas del sistema.

OCSP (Online Certificate Status Protocol): Protocolo que permite verificar en línea el estado de revocación de un certificado digital, como alternativa a las listas CRL.

OpenSSL: Conjunto de herramientas de línea de comandos y librerías que implementan los protocolos SSL/TLS, utilizados para generar certificados, claves y solicitudes de firma (CSR).

OpenSSH: Conjunto de herramientas que implementan el protocolo SSH para establecer conexiones seguras. Incluye tanto servidor como cliente SSH.

PAM (Pluggable Authentication Module): Marco modular utilizado en sistemas UNIX/Linux para implementar mecanismos de autenticación personalizados, como autenticación mediante certificados digitales.

PKCS#11: Estándar que define una API para el acceso a dispositivos criptográficos como HSM. Utilizado por EJBCA para interactuar con SoftHSM2.

PKI (Infraestructura de Clave Pública): Conjunto de tecnologías, políticas y procedimientos que permiten la creación, gestión y validación de certificados digitales para asegurar comunicaciones electrónicas.

RBAC (Role-Based Access Control): Modelo de control de acceso basado en roles, en el cual los permisos se asignan a roles específicos, y los usuarios heredan esos permisos al estar asociados a un rol.

Red Interna (VLAN): Segmento lógico de red utilizado para aislar el tráfico de comunicación entre servidores, inaccesible desde el exterior por diseño.

Red Pública: Segmento de red expuesto a Internet u otras redes externas, donde se ubican servicios como el Gateway y la interfaz de administración de EJBCA.

Revocación de certificado: Proceso mediante el cual se invalida un certificado antes de su fecha de expiración, por motivos como pérdida de clave privada o cambio de rol.

Rol: Atributo asignado a un usuario que define los permisos o tareas que puede realizar dentro de un sistema. Es clave para implementar políticas de autorización.

Secuencia de autenticación: Conjunto de pasos ejecutados para validar la identidad de un usuario mediante su certificado digital.

Serial ID: Identificador único asociado a un certificado digital, utilizado para verificar su estado, validez y trazabilidad.

Sesión SSH: Conexión segura entre un cliente y un servidor, establecida mediante el protocolo Secure Shell (SSH), que permite ejecutar comandos de forma remota.

Slot (en HSM): Contenedor lógico dentro de un HSM donde se almacenan claves criptográficas asociadas a un token.

SoftHSM2: Implementación de software de un módulo de seguridad de hardware (HSM) que emula el almacenamiento seguro de claves criptográficas para entornos de prueba o desarrollo.

SSH (Secure Shell): Protocolo de red que permite establecer sesiones remotas seguras, usualmente sobre el puerto 22. Utilizado para acceso técnico a servidores.

Subject DN (Distinguished Name): Conjunto de atributos que identifican de forma única a una entidad dentro de un certificado X.509 (por ejemplo: CN, email, rol).

Topología de red: Representación gráfica o lógica de la estructura de interconexión entre los dispositivos de una red.

Traza de auditoría: Registro detallado de las actividades realizadas por los usuarios en el sistema, incluyendo marcas de tiempo, acciones y resultados.

VMware ESXi: Hipervisor bare-metal desarrollado por VMware, que permite desplegar múltiples máquinas virtuales sobre un mismo hardware físico.

vSwitch: Conmutador virtual utilizado por hipervisores como VMware ESXi para conectar máquinas virtuales entre sí o con redes físicas.

10. Bibliografía

[1] B. Schneier, *Applied Cryptography*. New York, NY, USA: John Wiley & Sons, 1996.

[2] M. Á. Solinas, R. J. Castello, L. Tula, C. Gallo, J. Jorge, and D. Bollo, "Implementación de una infraestructura de clave pública con herramientas de software libre," in *Proc. X Jornadas Argentinas de Software Libre (JSL) – JAIIO 42*, 2013, pp. 107–117, doi: 1850–2857.

[3] M. Tolbert, "Vulnerabilities of multi-factor authentication in modern computer networks," *Worcester Polytechnic Institute (WPI) Report*, 2021.

[4] J. Owens and J. Matthews, "A study of passwords and methods used in brute-force SSH attacks," in *USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)*, 2008, p. 8.

[5] T. Ylonen, "SSH – secure login connections over the internet," in *Proc. 6th USENIX Security Symposium*, San Jose, CA, USA, Jul. 1996, pp. 37–42, [Online]. Available: https://www.usenix.org/conference/6th-usenix-security-symposium/ssh-secure-login-connections-over-internet [Accessed: Aug. 14, 2025].

[6] J. Ellingwood. (2022) Understanding the SSH encryption and connection process. [Online]. Available: https://www.digitalocean.com/community/tutorials/understanding-the-ssh-encryption-and-connection-process [Accessed: Aug. 14, 2025].

[7] D. F. Ferraiolo and D. R. Kuhn, "Role–based access controls," *arXiv preprint arXiv:0903.2171*, 2009, [Online]. Available: https://arxiv.org/abs/0903.2171 [Accessed: Aug. 14, 2025].

[8] GeeksforGeeks, "Setting up rbac in elasticsearch with kibana," *GeeksforGeeks*, 2024, [Online]. Available: https://www.geeksforgeeks.org/ [Accessed: Aug. 14, 2025].

[9] Elastic. (2024) User roles | elastic docs. [Online]. Available: https://www.elastic.co/guide/en/elasticsearch/reference/current/security-privileges.html [Accessed: Aug. 14, 2025].

[10] W. Takase, T. Nakamura, Y. Watase, and T. Sasaki, "A solution for secure use of kibana and elasticsearch in multi-user environment," *arXiv preprint arXiv:1706.10040*, 2017, [Online]. Available: https://arxiv.org/abs/1706.10040 [Accessed: Aug. 14, 2025].

[11] TigerGraph Documentation. (2024) Set up log viewing with elasticsearch, kibana and filebeat. [Online]. Available: https://docs.tigergraph.com/ [Accessed: Aug. 14, 2025].

[12] OpenBSD Project. (2024) Openssh. [Online]. Available: https://www.openssh.com/ [Accessed: Aug. 14, 2025].

[13] Keyfactor. (2024) Ejbca. [Online]. Available: https://www.ejbca.org/ [Accessed: Aug. 14, 2025].

[14] ——. (2024) Protocols — ejbca documentation. [Online]. Available: https://doc.primekey.com/ejbca/ejbca-operations/ejbca-ca-concept-guide/protocols [Accessed: Aug. 14, 2025].

[15] Thales. (2024) An introduction to PKCS#11. [Online]. Available: https://thalesdocs.com/gphsm/ptk/5.9/docs/Content/PTK-C_Program/intro_PKCS11.htm [Accessed: Aug. 14, 2025].

[16] Keyfactor. (2024) Hardware security modules (hsm) — ejbca ce. [Online]. Available: https://doc.primekey.com/ejbca/ejbca-integration/hardware-security-modules-hsm [Accessed: Aug. 14, 2025].

[17] ——. (2024) Crypto tokens overview. [Online]. Available: https://doc.primekey.com/ejbca/ejbca-operations/ejbca-ca-concept-guide/crypto-tokens-overview [Accessed: Aug. 14, 2025].

[18] ——. (2024) Managing crypto tokens. [Online]. Available: https://doc.primekey.com/ejbca/ejbca-operations/ejbca-operations-guide/ca-operations-guide/managing-crypto-tokens [Accessed: Aug. 14, 2025].

[19] ——. (2024) Ejbca easy rest client. [Online]. Available: https://github.com/Keyfactor/ejbca-easy-rest-client [Accessed: Aug. 14, 2025].

[20] OpenDNSSEC. (2022) Softhsm version 2. [Online]. Available: https://github.com/opendnssec/SoftHSMv2 [Accessed: Aug. 14, 2025].

[21] C. D'Cruz. (2020) A dive into softhsm. [Online]. Available: https://clydedcruz.medium.com/a-dive-into-softhsm-e4be3e70c7bc [Accessed: Aug. 14, 2025].

[22] Adrien. (2023) Install softhsmv2 and use it via openssl and PKCS#11. [Online]. Available: https://illuad.fr/2022/01/30/install-softhsmv2-and-use-it-via-openssl-and-pkcs11-11.html [Accessed: Aug. 14, 2025].

[23] Wikipedia. (2024) Hardware security module. [Online]. Available: https://en.wikipedia.org/wiki/Hardware_security_module [Accessed: Aug. 14, 2025].

[24] Thales. (2024) Hardware security modules (hsms). [Online]. Available: https://cpl.thalesgroup.com/encryption/hardware-security-modules [Accessed: Aug. 14, 2025].

[25] Keyfactor. (2022) Drivers for hardware security module (hsm) integration. [Online]. Available: https://github.com/Keyfactor/ejbca-containers/tree/master/hsm-integration [Accessed: Aug. 14, 2025].

[26] Intel. (2021) Creating authentication key pairs in softhsm. [Online]. Available: https://www.intel.com/content/www/us/en/docs/programmable/683642/21-4/creating-authentication-key-pairs-in.html [Accessed: Aug. 14, 2025].

[27] KeyFactor. (2017) Managing crypto tokens documentation. [Online]. Available: https://certs.advania.is/doc/userguide.html#Managing%20Crypto%20Tokens [Accessed: Aug. 14, 2025].

[28] Keyfactor. (2017) Hardware security modules (hsm) — ejbca documentation. [Online]. Available: https://certs.advania.is/doc/adminguide.html#Hardware%20Security%20Modules%20(HSM) [Accessed: Aug. 14, 2025].

[29] ——. (2024) Softhsm — ejbca ce. [Online]. Available: https://doc.primekey.com/ejbca/ejbca-integration/hardware-security-modules-hsm/softhsm [Accessed: Aug. 14, 2025].

[30] SSH Communications Security. (2024) Ssh. [Online]. Available: https://www.ssh.com/academy/ssh [Accessed: Aug. 14, 2025].

[31] Canonical Ltd. (2019) Pam (pluggable authentication modules). [Online]. Available: https://manpages.ubuntu.com/manpages/jammy/es/man7/PAM.7.html [Accessed: Aug. 14, 2025].

[32] SSH Communications Security. (2024) Privileged access management (pam). [Online]. Available: https://www.ssh.com/academy/pam/what-is-privileged-access-management [Accessed: Aug. 14, 2025].

[33] IJCT Journal, "Automating infrastructure management: The power of ansible for devops," *IJCT Journal*, 2025, [Online]. Available: https://ijctjournal.org/volume-7-issue-2-ansible-devops-efficiency/ [Accessed: Aug. 14, 2025].

[34] P. Sahoo, S. Pujar, G. Nalawade, R. Gebhardt, L. Mandel, and L. Buratti, "Insights from the usage of the ansible lightspeed code completion service," *arXiv preprint arXiv:2402.17442*, 2024, [Online]. Available: https://arxiv.org/abs/2402.17442 [Accessed: Aug. 14, 2025].

[35] C. Carreira, N. Saavedra, A. Mendes, and J. F. Ferreira, "From "worse is better" to better: Lessons from a mixed methods study of ansible's challenges," *arXiv preprint arXiv:2504.08678*, 2025, [Online]. Available: https://arxiv.org/abs/2504.08678 [Accessed: Aug. 14, 2025].

[36] Elastic. (2025) Kibana guide. Documentation. [Online]. Available: https://www.elastic.co/guide/en/kibana/index.html

[37] ——. (2025) Aggregations. Documentation. [Online]. Available: https://www.elastic.co/docs/explore-analyze/query-filter/aggregations

[38] ——. (2025) Dashboards. Documentation. [Online]. Available: https://www.elastic.co/docs/explore-analyze/dashboards

[39] ——. (2025) Reporting and sharing. Documentation. [Online]. Available: https://www.elastic.co/docs/explore-analyze/report-and-share

[40] D. Ruest and N. Ruest, *Virtualization: A Beginner's Guide*. New York, NY, USA: McGraw–Hill Education, 2009.

[41] VMware, Inc. (2020) What is a hypervisor? [Online]. Available: https://www.vmware.com/topics/glossary/content/hypervisor.html [Accessed: Aug. 14, 2025].

[42] G. J. Popek and R. P. Goldberg, "Formal requirements for virtualizable third generation architectures," *Communications of the ACM*, vol. 17, no. 7, pp. 412–421, 1974.

[43] R. Buyya, J. Broberg, and A. Goscinski, *Cloud Computing: Principles and Paradigms*. Hoboken, NJ, USA: Wiley, 2011.

[44] D. Merkel, "Docker: Lightweight linux containers for consistent development and deployment," *Linux Journal*, no. 239, 2014.

[45] J. Turnbull, *The Docker Book: Containerization is the New Virtualization*. James Turnbull, 2014.

[46] C. Boettiger, "An introduction to docker for reproducible research," *ACM SIGOPS Operating Systems Review*, vol. 49, no. 1, pp. 71–79, 2015.

[47] Docker, Inc. (2023) Overview of docker compose. [Online]. Available: https://docs.docker.com/compose/ [Accessed: Aug. 14, 2025].

[48] D. Jacobson, G. Brail, and D. Woods, *APIs: A Strategy Guide*. Sebastopol, CA, USA: O'Reilly Media, 2011.

[49] D. Zhang, W. Yao, and Z. Jin, "A survey on restful web services," *ACM Computing Surveys (CSUR)*, vol. 53, no. 6, pp. 1–37, 2020.

[50] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, University of California, Irvine, 2000, [Online]. Available: https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm [Accessed: Aug. 14, 2025].

[51] PrimeKey Solutions AB, *EJBCA Documentation*, 2022, available online: https://docs.ejbca.org/ [Accessed: Jul. 18, 2025].

[52] Keyfactor. (2023) Ejbca by keyfactor — technical overview. [Online]. Available: https://www.keyfactor.com/platform/ejbca/ [Accessed: Jul. 18, 2025].

[53] M. Gorecki and T. Walkowiak, "Open source pki implementation based on ejbca," in *Proc. Federated Conf. on Computer Science and Information Systems (FedCSIS)*, 2017, pp. 897–900.

[54] S. Klincovisky and B. S. Perez, "Autenticación a servidores virtuales con infraestructura de clave pública y software libre," in *Trabajo de Proyecto Integrador, Facultad de Ciencias Exactas, Físicas y Naturales*, 2025.