Universidad Politécnica
de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

Máster Universitario en Máster Universitario en Ingeniería Informática

Trabajo Fin de Máster

Diseño de una Solución Integrada de Monitorización para Entornos Distribuidos

Autor(a): Lucía Fernández Molleda
Tutor(a): Fernando Pérez Costoya

Madrid, Julio 2024

Este Trabajo Fin de Máster se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Máster
Máster Universitario en Máster Universitario en Ingeniería Informática

Título: Diseño de una Solución Integrada de Monitorización para Entornos Distribuidos

Julio 2024

Autor(a): Lucía Fernández Molleda
Tutor(a): Fernando Pérez Costoya
Dept. de Arquitectura y Tecnología de Sistemas Informáticos
ETSI Informáticos
Universidad Politécnica de Madrid

Resumen

Este Trabajo de Fin de Máster se centra en la relevancia de la computación distribuida tanto en la vida cotidiana como en el entorno empresarial. Analiza cómo esta tecnología, empleada en actividades diarias como búsquedas en internet o videoconferencias, contribuye a la mejora de la eficiencia y a la optimización de recursos.

El proyecto principal ha consistido en el desarrollo de un sistema integral de monitorización para sistemas distribuidos, abordando aspectos fundamentales como el rendimiento y la seguridad. Se ha diseñado una solución que permite a los administradores de sistemas supervisar eficientemente toda la infraestructura TI de una empresa, facilitando así su labor y reduciendo los costos operativos.

Para ello, se ha llevado a cabo un análisis detallado de los sistemas distribuidos y sus aplicaciones, evaluando también una variedad de herramientas de monitorización. El diseño de la solución integrada de monitorización está enfocado en sistemas distribuidos e incluye la automatización en la configuración y el despliegue.

Adicionalmente, se ha desarrollado un prototipo funcional como demostración de concepto. Este prototipo se ha diseñado simulando la infraestructura de una pequeña empresa, lo que facilita una evaluación realista de su funcionalidad y eficacia en un contexto empresarial concreto.

Abstract

This Master's Thesis focuses on the significance of distributed computing in everyday life and in the business environment. It analyzes how this technology, used in daily activities such as internet searches or video conferences, contributes to improving efficiency and optimizing resources.

The main project has involved the development of an integrated monitoring system for distributed systems, addressing fundamental aspects such as performance and security. A solution has been designed that allows system administrators to efficiently supervise the entire IT infrastructure of a company, thereby facilitating their work and reducing operational costs.

To achieve this, a detailed analysis of distributed systems and their applications was conducted, along with an evaluation of a variety of monitoring tools. The design of the integrated monitoring solution is focused on distributed systems and includes automation in its configuration and deployment.

Additionally, a functional prototype has been developed as a proof of concept. This prototype was designed to simulate the infrastructure of a small business, which facilitates a realistic evaluation of its functionality and effectiveness in a specific business context.

Tabla de contenidos

Capítulo 1

Introducción

1.1. Motivación

La computación distribuida se ha establecido como un elemento esencial en la vida cotidiana de las personas desde hace varias décadas. Ejemplos claros de su uso incluyen actividades como realizar búsquedas en internet, enviar correos electrónicos, participar en videoconferencias o hacer reservas de hotel en línea.

Las arquitecturas de sistemas distribuidos han revolucionado la manera de abordar problemas complejos en el ámbito empresarial. Esto se debe a su capacidad para proporcionar una mayor cantidad de recursos computacionales, optimizando costes y tiempo simultáneamente.

El rendimiento de un computador se ve limitado por su hardware y software, y puede medirse en función del tiempo que tarda en ejecutar una tarea. Cuanto menor sea este tiempo, mayor será el rendimiento.

La computación distribuida implica que varios computadores trabajen conjuntamente para resolver un mismo problema de manera más eficiente. De este modo, una red de computadores actúa como si fuera un único equipo más potente, capaz de aportar recursos a gran escala. Este enfoque permite lograr un rendimiento superior y una utilización más óptima de los recursos de hardware, manejando cargas de trabajo más elevadas.

Sin embargo, este modelo de computación requiere infraestructuras más complejas, generalmente compuestas por numerosos servidores que colaboran entre ellos. Esto representa un desafío adicional para los administradores de sistemas, quienes deben mantener y asegurar el correcto funcionamiento de la infraestructura.

En el entorno empresarial, es fundamental controlar todos los sistemas informáticos mediante herramientas de monitorización. Estas permiten detectar instantáneamente el origen de cualquier incidencia y realizar un seguimiento en tiempo real del estado del sistema, tanto de la infraestructura como de los subsistemas. El objetivo es garantizar que el sistema sea fiable, estable y capaz de proporcionar los servicios para los cuales fue diseñado. Generalmente, se recomienda una supervisión periódica de los servidores para identificar y mitigar problemas de manera temprana, reduciendo así su impacto.

En este trabajo, se presenta una propuesta de monitorización diseñada para satisfacer las necesidades de un sistema distribuido. Esta propuesta incluye el estudio de diversas herramientas, su coste y cómo se integran con el resto del sistema.

1.2. Objetivo de proyecto

El objetivo de este trabajo es desarrollar una solución que permita al administrador de sistemas de una empresa monitorizar de manera completa y eficiente toda la infraestructura, facilitando así su labor y generando ahorro de costos.

Para lograr este objetivo, se ha diseñado un programa con dos funcionalidades principales claramente definidas. La primera funcionalidad permite la instalación y configuración de las herramientas necesarias para una monitorización integral, minimizando la intervención del administrador. La segunda funcionalidad habilita la modificación de las herramientas ya instaladas y la automatización de tareas comunes, lo cual es beneficioso para futuras modificaciones o mantenimientos de los servidores.

Además, la plataforma está diseñada para centralizar la gestión de alertas generadas por diversas herramientas de monitorización y notificar al administrador. Esto permitiría obtener una visión global del estado del sistema sin tener que acceder a cada herramienta por separado, facilitando y acelerando la toma de decisiones.

En resumen, este proyecto busca crear una solución integral de monitorización que optimice la gestión y el mantenimiento de la infraestructura. Para lograrlo, se requiere:

- Realizar un estudio exhaustivo sobre sistemas distribuidos y las posibilidades que ofrecen.
- Llevar a cabo un análisis comparativo de las herramientas de monitorización disponibles en el mercado.
- Diseñar una solución de monitorización integrada específica para entornos distribuidos.
- Investigar en detalle un problema específico y sus requisitos para formular una solución adecuada.
- Automatizar la configuración y el despliegue de la solución de monitorización integral desarrollada.
- Diseñar y desarrollar un prototipo funcional que sirva como demostración de concepto.
- Evaluar la viabilidad y realizar un análisis crítico del prototipo.

Esta propuesta busca no solo implementar una solución técnica, sino también proporcionar un marco que facilite el entendimiento, la gestión y la mejora continua de la infraestructura monitorizada. El estudio y análisis previos garantizan que la solución sea adecuada y efectiva para los entornos distribuidos, mientras que la automatización y el prototipo funcional aseguran su operatividad y eficiencia. La evaluación final permite identificar posibles mejoras y ajustar la solución según las necesidades reales.

1.3. Estructura del Trabajo

Este documento está organizado en varios capítulos que abordan los aspectos clave de cada etapa del desarrollo del proyecto, incluyendo las decisiones tomadas y sus justificaciones.

El primer capítulo, Estado del Arte, presenta un análisis profundo de los sistemas distribuidos. Aquí se proporciona una introducción general al tema, definiendo qué son los sistemas distribuidos, su origen, los diferentes tipos existentes y discutiendo las ventajas y desventajas asociadas a su uso.

En el siguiente capítulo, se examinan diversas herramientas de monitorización disponibles, evaluando sus pros y contras. Se presentan alternativas para la monitorización completa del sistema, describiendo brevemente cada herramienta considerada.

A continuación, se dedica una capítulo completo al diseño del prototipo. En esta parte, se establecen los requisitos, el nivel de desarrollo previsto y la estructura del prototipo. Se realiza una comparativa para determinar qué tecnologías de monitorización y automatización son más adecuadas para el proyecto, justificando la elección de las tecnologías implementadas en la fase de desarrollo.

El siguiente capítulo aborda la implementación del prototipo funcional. Aquí se explican los razonamientos detrás del desarrollo y la estructura seguida. Se detalla la estructura del sistema, el proceso de instalación y se analiza la funcionalidad de las herramientas seleccionadas en la fase de diseño y su contribución al proyecto. Se demuestra el funcionamiento del prototipo y se realiza un estudio de viabilidad para su implementación en un escenario real.

Finalmente, se analiza si la solución propuesta realmente facilita la labor del administrador de sistemas y si esta mejora es significativa. Se elaboran las conclusiones del documento, que incluyen posibles líneas de trabajo futuro y una valoración general de todo el proyecto realizado para este Trabajo de Fin de Máster.

Capítulo 2

Estado del Arte

Este Trabajo de Fin de Máster comienza con una fase inicial que presenta el estado del arte de los tipos de computación, proporcionando una introducción detallada. En esta sección, se profundiza en los sistemas distribuidos, explicando su funcionamiento, características y usos más comunes.

2.1. Computación Centralizada

La computación centralizada[1] es un modelo de procesamiento de datos en el cual un sistema central concentra todos los recursos computacionales. Este sistema es responsable de procesar todas las solicitudes, manejar la información y distribuirla a los clientes que la requieran.

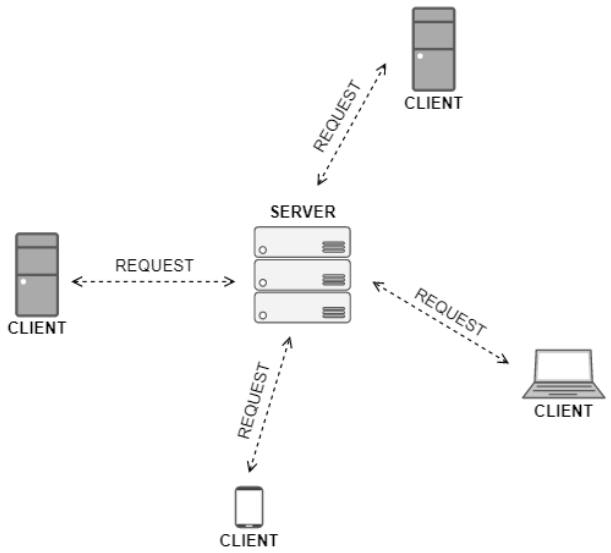


Figura 2.1: Sistema centralizado modelo cliente-servidor.

Este tipo de arquitectura se caracteriza por seguir un modelo cliente-servidor, donde el cliente es un sistema informático que solicita servicios, recursos o información a otro sistema llamado servidor.

Los ejemplos más característicos de computación centralizada son:

- **Mainframe:** [2] Son computadoras de alto rendimiento con una amplia gama de recursos, diseñadas para atender simultáneamente solicitudes de múltiples usuarios. La capacidad de procesamiento se distribuye temporalmente entre todos los usuarios, asegurando una utilización eficiente y equitativa de los recursos computacionales.

  Históricamente, los mainframes han sido utilizados en entornos empresariales para ejecutar aplicaciones críticas y manejar grandes volúmenes de datos. Tradicionalmente, la conexión de los clientes al servidor se hacía mediante terminales básicas, que actuaban como intermediarios para realizar peticiones al mainframe sin capacidad propia de procesamiento de datos.

  El software de estos sistemas se distingue por su estructura monolítica, integrando en una sola aplicación la interfaz de usuario, la lógica de negocios y el acceso a bases de datos. Esta aplicación se ejecutaba directamente en el mainframe, ya que las terminales conectadas al ordenador central carecían de capacidad de procesamiento. [3]

- **Supercomputadora:** [4] Es un sistema informático avanzado, diseñado para realizar tareas de procesamiento de datos a una velocidad y eficiencia muy superiores a las de las computadoras convencionales. Está optimizada para ejecutar cálculos complejos y manejar grandes cantidades de datos. Se utilizan principalmente en aplicaciones científicas, de ingeniería y de investigación que requieren un alto rendimiento computacional debido a su especialización y los recursos significativos necesarios para su operación y mantenimiento.

En resumen, un mainframe está diseñado para manejar grandes cantidades de transacciones empresariales, mientras que una supercomputadora está orientada a tareas que requieren un procesamiento intensivo y un alto rendimiento computacional.

Mientras que los mainframes se destinan para computación de volumen confiable en dominios que requieren operaciones enteras, por ejemplo en finanzas, indexación, comparaciones, etc. Las supercomputadoras están diseñadas para realizar operaciones de coma flotante como son la suma, resta y multiplicación con suficientes dígitos de precisión para modelar fenómenos continuos como por ejemplo el clima.

2.1.1. Ventajas y Desventajas de la Computación Centralizada

La computación centralizada ofrece varias ventajas:

- Facilita el control y la administración de recursos, lo que puede resultar en una mayor eficiencia operativa y una mejor utilización de los mismos.
- Es más fácil mantener la seguridad y la integridad de los datos, ya que se pueden implementar y gestionar medidas de seguridad centralizadas, como firewalls o sistemas de detección de accesos no autorizados.
- Las políticas y configuraciones pueden ser aplicadas de manera más uniforme en un entorno centralizado, lo que facilita la implementación y el cumplimiento de políticas de seguridad y acceso.

Sin embargo, los sistemas centralizados también presentan grandes inconvenientes:

- La escalabilidad es un desafío, ya que la adición de recursos o la adaptación a cambios puede requerir modificaciones significativas en la arquitectura.
- La dependencia de un único punto central puede ser una vulnerabilidad potencial, ya que un fallo en el sistema central tiene un impacto significativo en todo el sistema.
- Puede haber problemas de latencia en la comunicación, dependiendo de la cantidad de clientes y la distancia a la ubicación centralizada. [5]

2.2. Computación Distribuida

En el libro *"Distributed Systems: Principles and Paradigms"*, Tanenbaum define un sistema distribuido como una colección de computadoras independientes y aplicaciones que aparecen ante los usuarios como un sistema unificado y coherente. Estos sistemas están interconectados y colaboran para alcanzar un objetivo común.

En un sistema distribuido, una red de ordenadores conectados entre sí colabora dividiendo tareas para atender las solicitudes entrantes. Esta configuración permite que la red funcione como si fuera un único computador más potente, pero aprovechando las ventajas de contar con diversos servidores. Por ejemplo, se puede asignar una tarea específica al nodo o nodos más adecuados para ejecutarla con mayor eficiencia, o distribuir la carga entre varios nodos para optimizar el rendimiento y la utilización de recursos. [6]

Esta metodología permite un rendimiento mejorado, ya que hace un uso eficiente de los recursos de hardware disponibles. Como resultado, los sistemas distribuidos tienen la capacidad de manejar cargas de trabajo más elevadas que los sistemas centralizados, minimizando el impacto de los picos de demanda.

La principal diferencia entre un sistema centralizado y uno distribuido es el patrón de comunicación entre los nodos del sistema. En un sistema centralizado, todos los nodos se comunican con un nodo central, lo que puede llevar a la sobrecarga y ralentización de la red. En cambio, en los sistemas distribuidos, esta concentración de comunicaciones y procesamiento no se presenta, permitiendo una distribución más equilibrada de la carga y una mayor eficiencia en el manejo de las tareas. [7]

En un entorno de computación distribuida, un problema se descompone en múltiples tareas, asignando cada una de ellas a un nodo diferente. Dado que los nodos están físicamente separados y no tienen la capacidad de utilizar una memoria compartida, por lo que es necesario transmitir la información entre ellos a través de la red. Esto se logra mediante el uso de protocolos de comunicación estándar, como TCP/IP y UDP, que facilitan el intercambio eficiente de datos entre los nodos. [6]

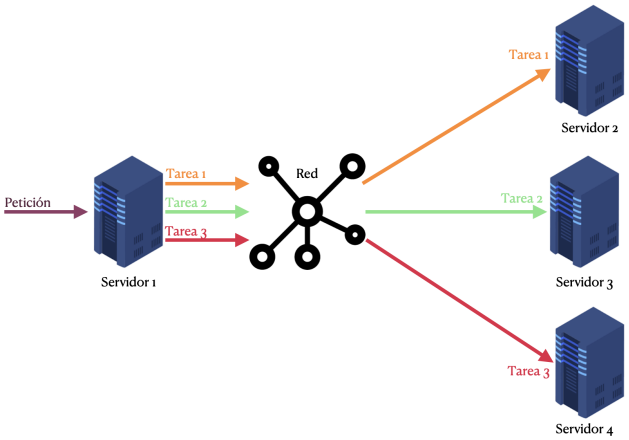


Figura 2.2: Reparto de tareas en un sistema distribuido.

2.2.1. Origen de los Sistemas Distribuidos

Los sistemas distribuidos han experimentado una notable evolución a lo largo del tiempo, impulsados por la creciente necesidad de aplicaciones más robustas, escalables y resistentes a fallos. Su desarrollo ha sido el resultado de una combinación de avances tanto teóricos como prácticos.

El concepto de sistemas distribuidos comenzó a usarse en la década de 1960. Fue entonces cuando surgió la idea de estructurar un sistema operativo como una colección de procesos de comunicación, donde cada proceso tiene una función específica y no pueden interferir otros (variables no compartidas). Este enfoque sentó las bases para el concepto de transmisión de mensajes.

Proyectos como el *Berkeley Timesharing System*, introducido en 1961, y el *Compatible Time-Sharing System* (CTSS), fueron pioneros en permitir a múltiples usuarios el acceso y la compartición de recursos de una computadora central. Estos desarrollos iniciales fueron fundamentales en la evolución de los sistemas distribuidos hacia las complejas estructuras que conocemos hoy. [8]

ARPANET, el precursor de Internet, introdujo la idea de interconectar computadoras distribuidas geográficamente. En 1969, se estableció la primera red de ARPANET, conectando cuatro ordenadores situados en diferentes universidades de Estados Unidos. Esta interconexión es considerada un momento clave en el desarrollo de lo que hoy conocemos como Internet, convirtiéndose en una de las primeras redes de computadoras en el mundo.

A principios de la década de 1970, surgió el correo electrónico de ARPANET. Este sistema de correo electrónico es reconocido como la primera aplicación distribuida a gran escala, abriendo el camino para el desarrollo de una multitud de aplicaciones y servicios distribuidos que hoy forman la base de nuestra interacción y comunicación digital. [9]

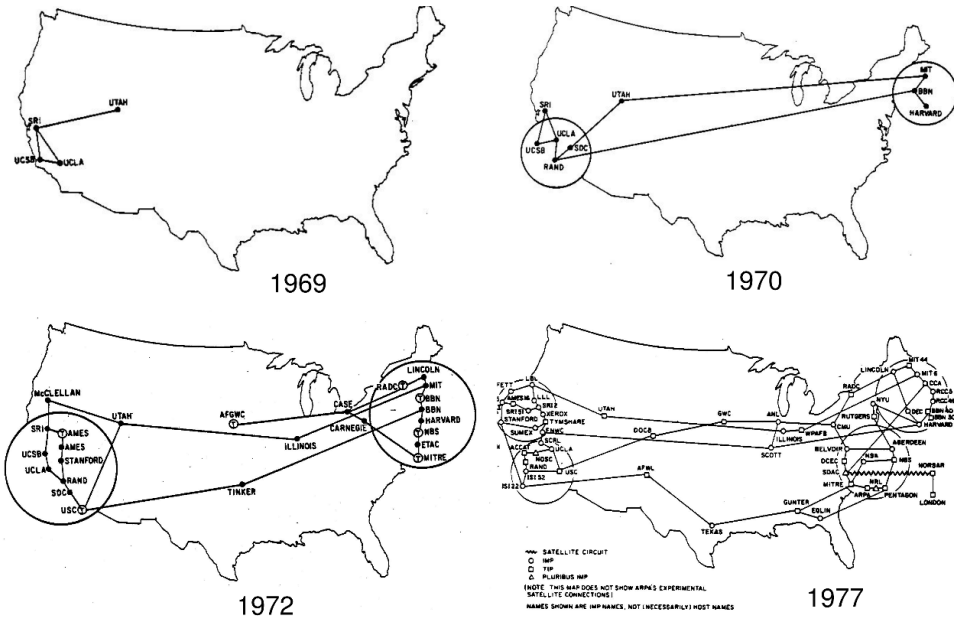


Figura 2.3: Desarrollo de ARPANET a lo largo de los años.

En 1972, se llevó a cabo la implementación del sistema de ficheros distribuido de Berkeley (DFS). DFS facilita el acceso a archivos remotos como si estuvieran almacenados localmente, permitiendo a los usuarios interactuar con ellos desde cualquier dispositivo y ubicación, brindando una mayor flexibilidad y eficiencia en la gestión de datos. [10]

A lo largo de la década de 1980, se introdujeron conceptos clave como la transparencia, la comunicación entre procesos y la replicación de datos, todos ellos esenciales para el funcionamiento eficiente de estos sistemas.

El desarrollo de protocolos de comunicación como TCP/IP jugó un papel fundamental en facilitar la comunicación entre sistemas distribuidos. Además, se empezó a adoptar el uso de redes de área local (LAN) de alta velocidad, como Ethernet, proporcionando una infraestructura sólida para la conectividad y el intercambio de datos en entornos distribuidos. [11]

2.2.2. Características de un Sistema Distribuido

Un sistema distribuido debe abordar varios aspectos clave:

- **Escalabilidad:** Los sistemas distribuidos pueden expandirse para satisfacer las crecientes necesidades y cargas de trabajo mediante la adición de nuevos nodos a la red.
- **Disponibilidad:** En caso de fallo de un nodo, otro puede asumir sus tareas asignadas, causando un mínimo impacto en el funcionamiento general del sistema.
- **Consistencia:** Los sistemas distribuidos aseguran que los datos sean coherentes entre todos los nodos, incluso cuando se producen fallos. Esto se logra mediante la replicación y sincronización de la información. Al duplicar los datos y compartir la información entre los nodos, el sistema garantiza que todos los nodos tengan una copia actualizada y consistente de los datos, permitiendo así la tolerancia a fallos sin comprometer la integridad de la información.
- **Transparencia:** Permiten interactuar con el sistema como si fuera una sola computadora, a pesar de la diversidad de hardware, middleware, software y sistemas operativos. El usuario no es consciente de la complejidad del sistema y tiene la sensación de estar trabajando en una sola máquina local.
- **Eficiencia:** Ofrecen un rendimiento más rápido y un uso más óptimo de los recursos disponibles, gestionando grandes cargas de trabajo y minimizando el impacto de los picos de carga.
- **Costo:** Permiten construir infraestructuras con componentes hardware menos costosos que, en conjunto, pueden contar con la capacidad de un superordenador, ofreciendo un rendimiento superior a un menor costo.
- **Flexibilidad:** Pueden adaptarse a diferentes necesidades y configuraciones, analizando y aprovechando los recursos disponibles en tiempo real. El sistema distribuye la carga de manera equitativa para evitar la saturación de los nodos.
- **Seguridad:** Se implementan mecanismos de seguridad para proteger los datos y recursos del sistema, tales como autenticación, autorización y cifrado.
- **Desempeño:** Se asegura un tiempo máximo de respuesta para las aplicaciones.

Pero también presenta algunos inconvenientes:

- **Sincronización:** Pueden surgir problemas de tiempo y sincronización entre las instancias distribuidas, ya que las tareas no se realizan en el mismo nodo, lo que puede complicar la coordinación y la coherencia de los datos.
- **Fiabilidad:** Aunque la descentralización ofrece ventajas, también puede generar vulnerabilidades de seguridad, como la transmisión de datos a través de redes públicas y la exposición a sabotajes y ataques informáticos.
- **Identificación de errores:** La complejidad de la infraestructura distribuida hace que diagnosticar y corregir fallos sea más complicado y costoso.

En general, las infraestructuras distribuidas son más propensas a errores debido a la mayor cantidad de componentes y posibles causas de problemas tanto a nivel de hardware como de software. Sin embargo, también suelen ser más tolerantes a estos errores, lo que les permite continuar operando incluso cuando se producen fallos.

2.2.3. Arquitecturas Distribuidas

La arquitectura de un sistema se refiere al diseño fundamental y la estructura que define cómo interactúan y cooperan sus distintos componentes. Esta arquitectura puede adoptar diversas formas, dependiendo de la naturaleza de las interacciones entre los elementos y de los requisitos específicos del sistema. Por lo tanto, existen múltiples tipos de arquitecturas de sistemas, cada una adecuada para diferentes necesidades y contextos.

En el ámbito de los sistemas distribuidos, hay varios tipos de computación que se diferencian según la arquitectura del sistema y los modelos de interacción empleados en la infraestructura. Estas variaciones reflejan las distintas maneras en que se pueden organizar y coordinar los elementos dentro de un sistema distribuido [12]

2.2.3.1. Arquitectura Cliente-servidor:

La arquitectura cliente-servidor es una de las más comunes en sistemas distribuidos y se basa en la clara división de roles y responsabilidades entre dos componentes principales: el cliente y el servidor. [13]

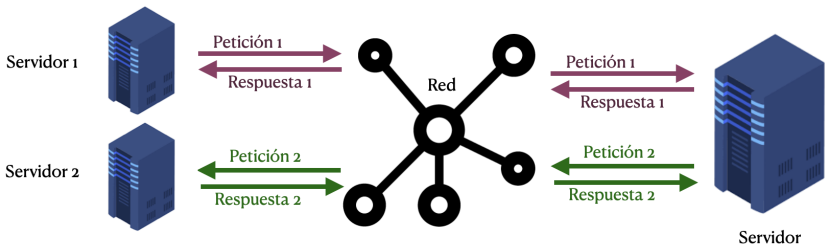


Figura 2.4: Arquitectura cliente-servidor.

- **Cliente**: Es la entidad que solicita servicios o recursos de un servidor. Puede ser una aplicación, un dispositivo o incluso otro servidor que necesita realizar

operaciones o acceder a datos almacenados en otra ubicación. Generalmente, el cliente tiene capacidad de procesamiento y almacenamiento limitadas.

- **Servidor:** Es la entidad que responde a las solicitudes de los clientes, proporcionándoles servicios o recursos. Cuando un servidor recibe una solicitud de un cliente, realiza el procesamiento necesario y envía una respuesta al cliente. Tanto la lógica de la aplicación como los datos suelen estar alojados y gestionados en el servidor.

El cliente y el servidor pueden estar ubicados en diferentes máquinas o dispositivos interconectados a través de una red. La comunicación entre ellos se realiza mediante protocolos de red y puede ser síncrona o asíncrona, dependiendo de las necesidades de la aplicación y del sistema.

La comunicación síncrona requiere que tanto el emisor como el receptor estén activos al mismo tiempo, como en una llamada telefónica, mientras que la comunicación asíncrona permite enviar y recibir mensajes en momentos distintos, similar al correo electrónico.

En los sistemas cliente-servidor tradicionales, generalmente hay dos nodos con roles y responsabilidades bien definidos. Sin embargo, en los sistemas distribuidos modernos, la configuración puede incluir más de dos nodos con roles dinámicos. Un nodo que actúa como cliente en una interacción puede ser el servidor en otra. Aunque en estos sistemas distribuidos no siempre hay clientes y servidores fijos, siempre existe una entidad que inicia la comunicación (cliente) y otra que responde (servidor).

2.2.3.2. Arquitectura de N Capas

Una capa es un conjunto de componentes o módulos de software que realizan funciones específicas dentro de un sistema. Cada capa se comunica con las capas adyacentes mediante solicitudes de cliente-servidor a través de una interfaz bien definida, facilitando la conexión y la cooperación eficiente entre diferentes elementos del sistema. Esta estructura permite la separación de responsabilidades y la modularidad del sistema.

La arquitectura de tres capas organiza la funcionalidad de una aplicación en tres niveles distintos: [13]

- **Capa de presentación:** Se encarga de mostrar los datos y respuestas procesadas por el sistema, proporcionando una interfaz de usuario. Aquí es donde el usuario introduce comandos o realiza acciones que son recogidas y transmitidas a las capas subsiguientes para su procesamiento.
- **Capa de aplicación o negocio:** Actúa como un intermediario entre las capas de presentación y de almacenamiento. Contiene la lógica de la aplicación y las funciones centrales para las que se diseñó el sistema distribuido, como cálculos, validaciones y operaciones específicas de la aplicación. Solicita datos a la capa de almacenamiento cuando es necesario.
- **Capa de almacenamiento o base de datos:** Gestiona el acceso y la manipulación de los datos almacenados en bases de datos u otras fuentes, realizando operaciones de lectura, escritura, consultas y actualizaciones. Esta capa asegura la persistencia y el acceso eficiente a los datos.

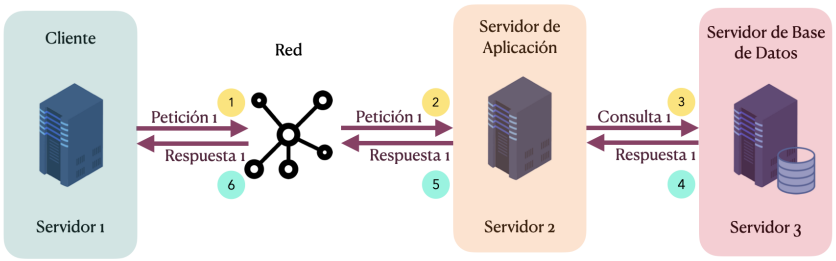


Figura 2.5: Arquitectura de 3 capas.

La principal diferencia entre la arquitectura cliente-servidor y la arquitectura de tres capas radica en su enfoque de división de tareas. La arquitectura cliente-servidor separa las responsabilidades entre dos entidades: el cliente y el servidor. En cambio, la arquitectura de tres capas segmenta la aplicación en tres capas lógicas diferenciadas: presentación, aplicación y almacenamiento.

La arquitectura de N capas es una extensión de la arquitectura de tres capas, introduciendo mayor complejidad al incluir múltiples sistemas cliente-servidor con funcionalidades específicas que interactúan para resolver un problema común. Este modelo ofrece la flexibilidad necesaria para adaptarse a los requisitos particulares de cada sistema, incorporando capas adicionales según sea necesario. Estas capas adicionales pueden incluir:

- **Capa de Servicios:** Proporciona servicios esenciales como autenticación de usuarios, autorización, gestión de sesiones, monitorización y balanceo de carga. Estos servicios aseguran un funcionamiento eficiente y seguro del sistema, distribuyendo las solicitudes equitativamente entre los recursos disponibles para optimizar el rendimiento y evitar sobrecargas.
- **Capa de Integración:** Facilita la comunicación entre distintos componentes y aplicaciones del sistema, que pueden estar construidos en diferentes plataformas o tecnologías. Esta capa maneja la traducción de datos entre formatos diversos, distintos protocolos de comunicación y la coordinación de procesos y transacciones entre servicios, asegurando una colaboración efectiva entre los sistemas.
- **Capa de Infraestructura o Plataforma:** Abarca los recursos físicos y virtuales necesarios, como servidores, almacenamiento, redes y plataformas de virtualización. Es responsable de proporcionar y gestionar estos recursos críticos, asegurando que estén disponibles y optimizados para el rendimiento y la disponibilidad del sistema. Incluye también la implementación de medidas de seguridad robustas para proteger los datos y las operaciones del sistema, así como el cumplimiento de normativas y estándares relevantes. En resumen, esta capa constituye la base técnica esencial sobre la que se apoyan las aplicaciones y servicios del sistema, garantizando su funcionamiento eficiente y seguro.

La mayoría de los sistemas distribuidos modernos utilizan una arquitectura de N capas con diversas aplicaciones empresariales operando conjuntamente en un sistema unificado. Las ventajas de este tipo de arquitectura son:

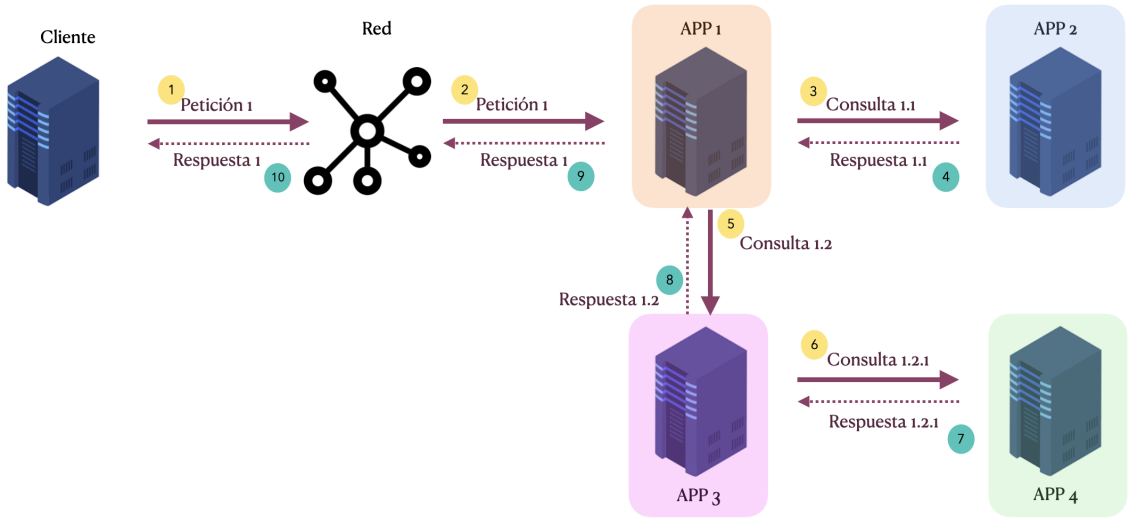


Figura 2.6: Arquitectura de N capas.

- **Modularidad:** Cada capa es autónoma y tiene responsabilidades bien definidas, lo que simplifica la modificación, actualización y mantenimiento de cada componente individual sin impactar los demás.
- **Reusabilidad:** La modularidad facilita la reutilización de las capas en diferentes contextos y proyectos, acelerando el desarrollo y reduciendo la duplicación de esfuerzos.
- **Escalabilidad:** Las capas pueden escalarse de manera independiente, permitiendo la adaptación a cambios en la carga de trabajo sin afectar al sistema en su totalidad, optimizando la asignación de recursos.

Finalmente, es importante diferenciar entre el concepto de ”capa”, que se refiere al diseño y división lógica del código del sistema, y el concepto de ”nivel”, que alude a la organización y división física de los componentes del sistema. Los niveles implican una separación física de los componentes, mientras que las capas se refieren a una organización lógica. Aunque a menudo se confunden, una aplicación de tres capas no necesariamente implica tres niveles físicos. [14]

En el proyecto se implementará un sistema de 5 capas y 11 niveles, el cual se explica en profundidad en el capítulo 4.

2.2.3.3. Arquitectura entre Pares o Peer-to-Peer (P2P)

La arquitectura Peer-to-Peer (P2P) en sistemas distribuidos es un modelo en el que los nodos, también conocidos como ”pares” o ”peers”, distribuyen entre sí tareas o cargas de trabajo. En este tipo de arquitectura, no existe una distinción clara entre clientes y servidores fijos; en su lugar, todos los nodos tienen igual capacidad y autoridad para ejecutar tareas, funcionando sin necesidad de una autoridad central. [15]

Un peer es un nodo en una red P2P que constituye la unidad fundamental de procesamiento. Cada nodo posee una copia de los archivos y actúa tanto como cliente como servidor para otros nodos, permitiendo tanto descargar archivos de otros nodos

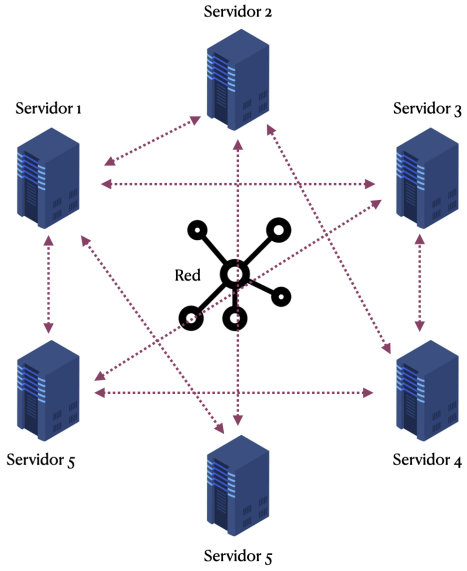


Figura 2.7: Arquitectura Peer to Peer.

como subir archivos a ellos. Esto distingue a las redes P2P de los sistemas cliente-servidor tradicionales, donde los dispositivos cliente únicamente pueden descargar archivos de un servidor.[16]

En cada nodo de una red P2P, se instala un software en la capa de aplicación que opera por encima de la topología física de la red y gestiona la comunicación entre pares. Este software utiliza su propio sistema de direccionamiento para almacenar y recuperar datos del sistema y puede encaminar las peticiones hacia ubicaciones que quizás no sean conocidas por el cliente.[17]

Debido a que cada nodo almacena, transmite y recibe archivos, las redes P2P se vuelven más rápidas y eficientes a medida que crece su base de usuarios. Su arquitectura distribuida también las hace muy resistentes a ciberataques, al no presentar un único punto de fallo.

Las características de una red P2P incluyen:

- **Descentralización:** En lugar de depender de un servidor centralizado, cada nodo en una red P2P es autónomo y capaz de realizar funciones tanto de cliente como de servidor
- **Igualdad:** Todos los nodos en una red P2P tienen roles similares y comparten la carga de trabajo sin una jerarquía fija.
- **Recursos Compartidos:** Los recursos como almacenamiento, potencia de procesamiento e información se distribuyen entre los nodos de la red, con cada nodo contribuyendo y accediendo a los recursos de otros.
- **Escalabilidad:** La arquitectura P2P puede crecer fácilmente, permitiendo la incorporación de nuevos nodos sin necesidad de realizar modificaciones importantes en la infraestructura ya establecida.
- **Resistencia a Fallos:** La distribución de funcionalidades e información entre

múltiples nodos proporciona una mayor resistencia a fallos.

- **Diversidad de Conexiones:** Cada nodo puede tener conexiones directas con otros nodos, permitiendo la comunicación directa entre pares sin necesidad de un servidor central.
- **Autonomía de Nodos:** Cada nodo puede operar de manera independiente, unirse y dejar la red en cualquier momento, operar con recursos propios y decidir sobre el uso y compartición de recursos.
- **Dinamismo:** Las redes P2P son dinámicas, con nodos que constantemente entran y salen, requiriendo mecanismos robustos para manejar cambios en la topología de la red.

Aunque la arquitectura Peer-to-Peer (P2P) es por naturaleza distribuida, es importante destacar que no todas las redes P2P son completamente descentralizadas. Algunas redes P2P todavía dependen de una autoridad central para coordinar las actividades, lo que introduce cierto nivel de centralización. Por lo tanto, las redes P2P pueden clasificarse en función de su grado de centralización. [15]

- **Centralizadas:** Un único servidor actúa como intermediario en todas las transacciones, limitando la privacidad y escalabilidad del sistema. Presentan puntos únicos de fallo y altos costos de mantenimiento.
- **P2P Descentralizadas o Puras:** No requieren gestión centralizada. Los nodos manejan peticiones y almacenan información, actuando como clientes y servidores. Ofrecen mayor robustez y economía, pero pueden tener sobrecarga de ancho de banda en las búsquedas.
- **Híbridas, Semicentralizadas o Mixtas:** Combinan características de redes centralizadas y descentralizadas. Varios servidores gestionan recursos compartidos, pero los nodos pueden operar independientemente si los servidores fallan.

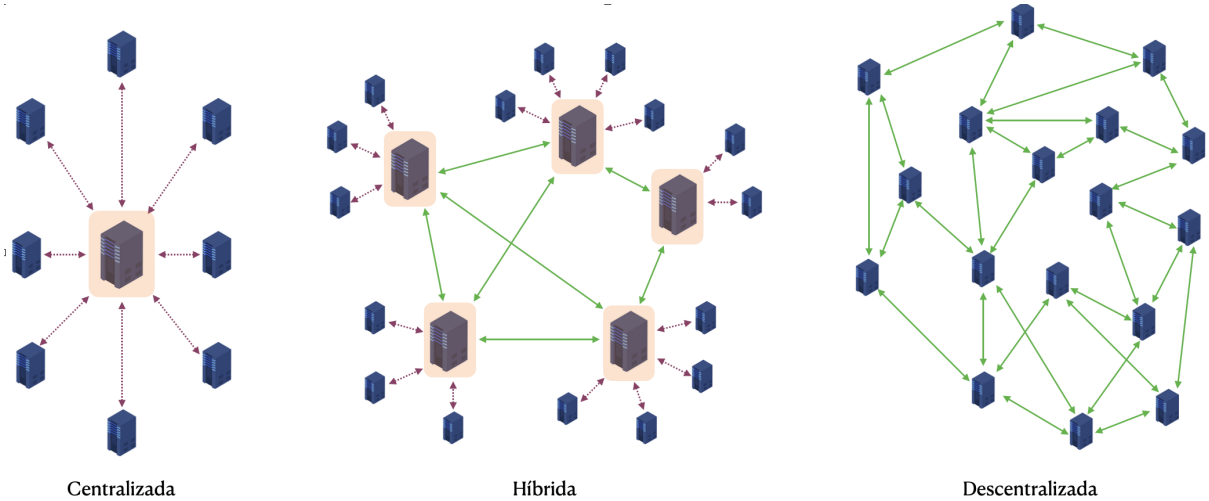


Figura 2.8: Clasificación por grado de centralización.

La arquitectura Peer-to-Peer (P2P) es versátil y popular, especialmente desde los años 90 con los programas de compartición de archivos. Actualmente, sustenta criptomonedas, blockchain, motores de búsqueda, plataformas de streaming y marketplaces en línea. [16]

2.2.3.4. Arquitectura Basada en Microservicios

Un microservicio en sistemas distribuidos es una técnica de diseño que descompone una aplicación en servicios más pequeños y autónomos, cada uno enfocado en una tarea específica. Estos microservicios se gestionan individualmente, facilitando un desarrollo ágil y eficiente. Al operar como procesos ligeros, pueden desarrollarse, desplegarse, operarse y escalarse de forma independiente.

Se comunican entre sí a través de APIs estandarizadas, permitiendo el uso de distintos lenguajes y tecnologías, a diferencia de los sistemas monolíticos tradicionales que deben escalarse conjuntamente. La modularidad de los microservicios los hace más pequeños y menos complejos.

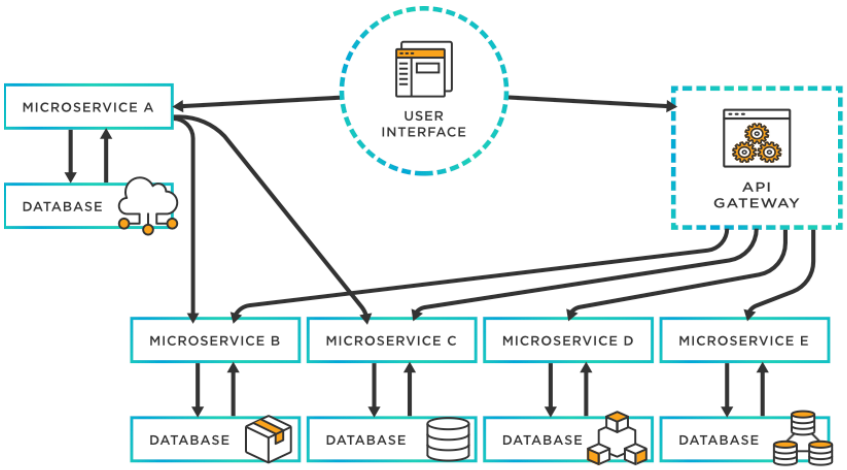


Figura 2.9: Microservicios.

Esta arquitectura promueve la descentralización, mejorando el mantenimiento y la tolerancia a fallos. Sus características principales incluyen:

- **Modularidad:** Cada microservicio se centra en una responsabilidad específica. La división de la aplicación en microservicios debe ser lógica y coherente en términos de funcionalidad y dominio. Agrupar funcionalidades relacionadas dentro de un mismo microservicio facilita tanto la comprensión como el mantenimiento de la aplicación.
- **Independencia:** Operan como procesos independientes. Deben ser lo suficientemente pequeños para concentrarse en tareas concretas, pero también lo suficientemente completos para ser autosuficientes.
- **Escalabilidad horizontal:** Permite escalar individualmente cada microservicio según la demanda, conocido como escalabilidad horizontal. Esto implica que los recursos pueden ser asignados específicamente donde exista una mayor demanda, en lugar de escalar toda la aplicación uniformemente.
- **Tolerancia a fallos:** El aislamiento asegura que un fallo en un microservicio no afecte a los demás, ya que se suele tener múltiples instancias de un mismo microservicio en diferentes nodos. Si un microservicio falla, otros pueden seguir gestionando las solicitudes, incrementando así la tolerancia a fallos.
- **Equilibrio de carga:** Distribuye las solicitudes de manera uniforme

- **Mantenimiento y actualización:** Facilita el desarrollo y despliegue ágil de aplicaciones, gracias a la división de sus componentes en servicios independientes y especializados.

Para llevar a cabo tareas más complejas, los microservicios necesitan comunicarse entre sí, utilizando para ello diferentes mecanismos de comunicación:

- **API REST:** Utilizan APIs basadas en REST (Representational State Transfer) con estándares web y protocolos como HTTP. Esto permite una integración flexible entre servicios mediante operaciones bien definidas (GET, POST, PUT, DELETE) y representaciones de recursos (JSON, XML), facilitando la comunicación uniforme y eficiente entre los microservicios.
- **Eventos:** Son capaces de responder dinámicamente a eventos dentro del sistema o de otros servicios. Esta capacidad crea un flujo de trabajo asíncrono donde los servicios pueden reaccionar y adaptarse a cambios en tiempo real, mejorando la flexibilidad y la capacidad de escalar según la demanda del sistema.
- **Mensajería:** Para una comunicación fiable se utilizan sistemas de mensajería como RabbitMQ o Kafka, que emplean colas de mensajes para transmitir y almacenar información de manera segura. Esta técnica permite una comunicación asíncrona, desacoplada y robusta entre los servicios, asegurando la entrega de mensajes incluso en condiciones adversas o picos de carga.

En conclusión, la arquitectura de microservicios mejora la escalabilidad y disponibilidad de aplicaciones empresariales, aunque plantea desafíos significativos como:

- **Complejidad de Gestión:** Coordinar y monitorizar múltiples microservicios independientes requiere herramientas especializadas para asegurar su funcionamiento armonioso y eficiente.
- **Rendimiento y Latencia:** Es necesario optimizar las interacciones entre microservicios para minimizar la latencia y mantener un alto rendimiento del sistema.
- **Consistencia de Datos:** Mantener la consistencia de los datos entre múltiples microservicios en entornos distribuidos es complejo y requiere mecanismos para garantizar la coherencia de la información.
- **Seguridad:** Cada punto de acceso y comunicación entre microservicios debe ser protegido adecuadamente para prevenir vulnerabilidades.
- **Gestión de la Carga de Trabajo y Escalabilidad:** Planificar y gestionar la carga de trabajo de manera óptima para cada microservicio implica implementar herramientas de escalado dinámico para adaptar los recursos según las necesidades del sistema.

2.2.3.5. Arquitectura Orientada a Servicios

La Arquitectura Orientada a Servicios (SOA) es un enfoque de diseño de software que permite crear aplicaciones mediante la combinación de servicios independientes, cada uno con una función específica. Estos servicios encapsulan funciones de software y pueden reutilizarse en distintos sistemas, facilitando el mantenimiento y la actualización. [18]

En SOA, un consumidor solicita información o funcionalidades a un servicio, que procesa los datos y devuelve una respuesta, haciendo de SOA una arquitectura flexible y eficiente para aplicaciones empresariales modernas. [19].

Los componentes clave de SOA incluyen:

- **Servicios:** Son los componentes fundamentales de SOA, pudiendo ser privados (para usuarios internos) o públicos (accesibles a cualquier usuario en Internet). Un servicio tiene:
  - **Implementación:** El código que ejecuta la lógica de una función específica. Ejemplos de estas funciones incluyen la autenticación de usuarios o el cálculo de facturas.
  - **Contrato:** Define las condiciones y términos para obtener el servicio, especificando requisitos, costo y calidad.
  - **Interfaz:** Permite la comunicación con el servicio para realizar operaciones o intercambiar información, reduciendo dependencias entre servicios.
- **Proveedor de servicios:** Crea, mantiene y ofrece uno o más servicios para ser utilizados por los consumidores. Las organizaciones pueden optar por desarrollar sus propios servicios internamente o pueden adquirir servicios de proveedores externos.
- **Agente o registro de servicios:** Directorio centralizado que almacena documentos descriptivos sobre los servicios proporcionados por diferentes proveedores.
- **Consumidor de servicios:** Entidad que solicita y utiliza los servicios ofrecidos por los proveedores. Los términos de interacción entre el proveedor y el consumidor están delineados en el contrato de servicio, que especifica las reglas y condiciones a seguir.

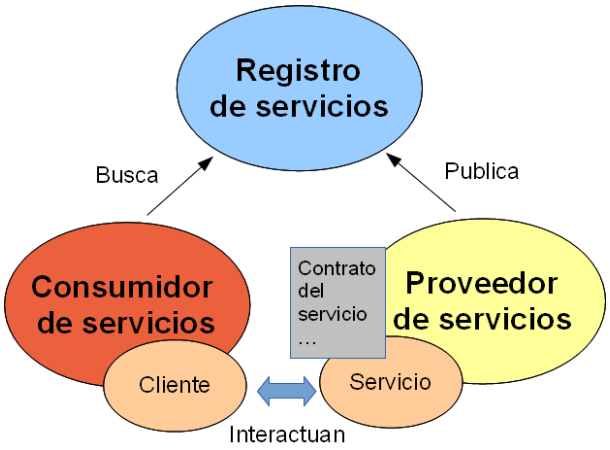


Figura 2.10: Roles SOA

Un ejemplo de interacción en SOA es el uso de un servicio de autorización en una aplicación. La aplicación envía las credenciales del usuario al servicio, que verifica la información y devuelve una respuesta de éxito o error, según corresponda. Este ejemplo ilustra cómo los servicios en una arquitectura SOA interactúan con sus consumidores, procesando las solicitudes y proporcionando las respuestas y funcionalidades necesarias.

Aunque no hay directrices estrictas para implementar SOA, existen principios básicos comúnmente aceptados:

- **Interoperabilidad:** Cada servicio proporciona documentos descriptivos que detallan su funcionalidad y los términos y condiciones relacionados. Permiten a cualquier sistema cliente utilizar el servicio, independientemente de su plataforma o lenguaje de programación. Por ejemplo, procesos empresariales específicos podrían aprovechar servicios escritos tanto en C# como en Python. Debido a la ausencia de interacciones directas, cualquier modificación en un servicio no afecta a otros componentes que lo utilizan.
- **Acoplamiento flexible:** Los servicios deben diseñarse con la menor dependencia posible de recursos externos y sin almacenar información de sesiones o transacciones anteriores.
- **Abstracción:** Los usuarios no necesitan comprender la lógica del código, accediendo a la funcionalidad del servicio a través de los contratos.
- **Granularidad:** Los servicios deben centrarse en una función empresarial específica, permitiendo combinar varios servicios para crear operaciones más complejas.

SOA introdujo el concepto de servicio, base para el desarrollo de la virtualización y el cloud computing modernos. SOA y la arquitectura de microservicios, aunque comparten ciertas similitudes, presentan diferencias clave. SOA es un enfoque de arquitectura a nivel empresarial que busca integrar una variedad de aplicaciones y servicios a lo largo de una organización. Se centra en la interoperabilidad y la eficiencia en la comunicación entre servicios dispares dentro de una empresa. Por otro lado, los microservicios representan una estrategia de implementación más específica y se utilizan principalmente por equipos de desarrollo para construir aplicaciones de forma modular.

2.2.3.6. Arquitectura Basada en Cloud Computing

La computación en la nube, o Cloud Computing, implica el uso de una red de servidores remotos a través de Internet para almacenar, gestionar y procesar datos. A diferencia de los sistemas locales, en la nube, tanto el software como el hardware están virtualizados y accesibles desde cualquier lugar. Esto permite a los usuarios acceder a recursos informáticos sin necesidad de gestionarlos directamente [20].

Este modelo ha transformado el mundo empresarial mediante infraestructuras flexibles, virtualización, automatización y capacidad de adaptación a demandas cambiantes. Los servicios de computación en la nube se dividen en tres categorías principales:

- **Software como Servicio (SaaS):** Proporciona aplicaciones completas bajo demanda, accesibles a través de un navegador web o aplicaciones específicas. Los usuarios no gestionan la infraestructura y pueden personalizar limitadamente la aplicación, todo esto lo hace el proveedor. SaaS elimina la necesidad de instalar, mantener y actualizar software en sistemas propios, reduciendo costes de soporte técnico.
- **Plataforma como Servicio:** Ofrece a los desarrolladores una plataforma en línea que incluye un sistema operativo, entorno de programación, base de datos y

servidor web. Permite a los usuarios crear, desarrollar y desplegar aplicaciones sin gestionar la infraestructura subyacente. Esto facilita a los desarrolladores enfocarse en la innovación sin preocuparse por los aspectos técnicos del entorno de desarrollo.

- **Infraestructura como Servicio:** Proporciona capacidad de almacenamiento y cómputo a través de la red como un servicio estandarizado. Incluye servidores, almacenamiento, conexiones de red y otros componentes críticos. Ejemplos comerciales incluyen Amazon Web Services (AWS) con servicios como EC2 y S3.

  IaaS es útil para empresas que necesitan infraestructura de TI escalable y personalizable sin la inversión en hardware físico, permitiendo a las organizaciones centrarse en sus actividades principales mientras un proveedor externo gestiona la infraestructura.

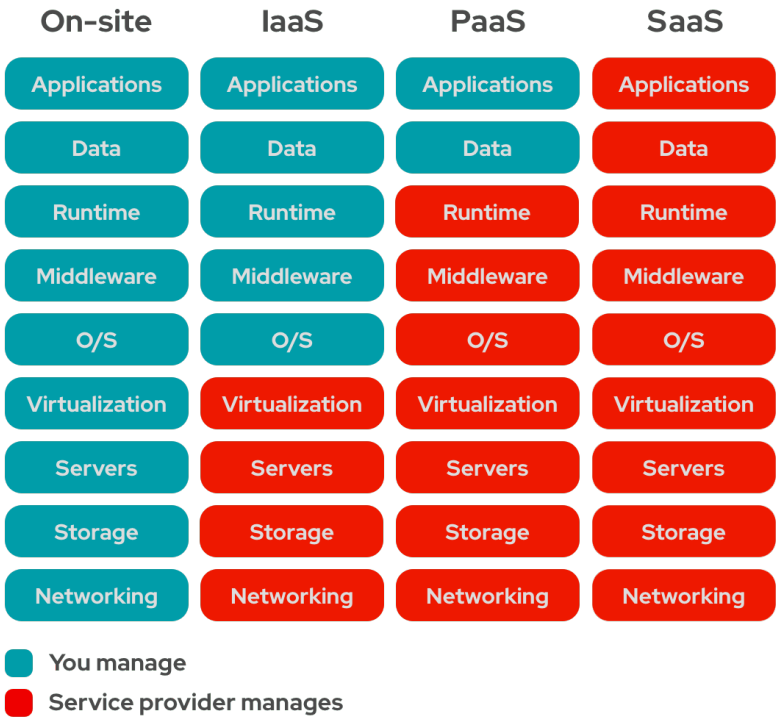


Figura 2.11: Tabla comparativa Cloud Computing.

En la computación en la nube, los servicios se pueden ofrecer mediante varias arquitecturas, cada una con características y beneficios específicos. Los principales tipos de arquitectura cloud son:

- **Nube Pública:** Ofrece recursos como máquinas virtuales, aplicaciones SaaS y hardware accesibles a través de Internet, gestionados por proveedores externos. Destaca por su elasticidad y escalabilidad, permitiendo ajustar recursos rápidamente según la demanda. Es eficiente en coste, ya que los clientes solo pagan por lo que consumen. Es ideal para alojar aplicaciones no críticas, almacenamiento de datos y proyectos de desarrollo y pruebas.
- **Nube Privada:** Proporciona una infraestructura de nube dedicada a un solo cliente, ofreciendo mayor seguridad y privacidad. Puede estar alojada en el cen-

tro de datos del cliente o en un proveedor externo. Es adecuada para organizaciones que necesitan control total sobre sus datos y aplicaciones, como entidades gubernamentales, instituciones financieras y empresas de salud.

- **Nube Híbrida:** Integra elementos de nubes públicas y privadas, permitiendo mover cargas de trabajo entre ambas según sea necesario. Ofrece la escalabilidad y eficiencia de costes de la nube pública para cargas menos sensibles, mientras mantiene las cargas críticas en una nube privada segura. Es ideal para empresas que buscan equilibrar flexibilidad, costos y seguridad, siendo útil para gestionar picos de demanda o aplicaciones que requieren cambios frecuentes entre ambas nubes.
- **Multicloud**: Implica el uso de múltiples servicios de nube, públicos o privados, de diferentes proveedores. Puede combinar IaaS, PaaS y SaaS, reduciendo la dependencia de un solo proveedor y ofreciendo una amplia gama de servicios y tecnologías. Permite a las empresas seleccionar la mejor solución para cada necesidad específica, optimizando su infraestructura de TI. Sin embargo, este enfoque requiere una gestión más compleja para integrar y operar eficientemente los servicios de distintos proveedores.

Cada arquitectura de computación en la nube (pública, privada, híbrida y multicloud) tiene sus propias fortalezas y se adapta a diferentes necesidades empresariales. La elección entre ellas depende de factores como la sensibilidad de los datos, los requisitos reglamentarios, la necesidad de escalabilidad y la estrategia de TI de la organización.

La computación en la nube ofrece ventajas significativas en comparación con las soluciones de TI tradicionales: [21]

- **Reducción de Costos**: Disminuye los gastos relacionados con la adquisición y mantenimiento de infraestructura de TI, permitiendo pagar solo por los recursos utilizados.
- **Mejora en Agilidad**: Facilita un despliegue rápido de aplicaciones empresariales, permitiendo a las empresas responder rápidamente a las necesidades del mercado.
- **Escalabilidad:** Permite ajustar la capacidad de TI según la demanda sin grandes inversiones previas.
- **Virtualización y Optimización de Recursos:** Optimiza el uso de recursos dividiendo servidores físicos en múltiples servidores virtuales, mejorando la eficiencia y gestión de los recursos informáticos disponibles.

2.2.4. Redes Distribuidas

Se considera que dos computadoras están interconectadas cuando tienen la capacidad de intercambiar información entre sí, lo que implica que ambas estén conectadas a una red. Esta conexión no se limita simplemente a un cable físico, sino que abarca todo un conjunto de elementos y configuraciones que permiten la transferencia de datos de manera eficiente y segura.

La estructura y el funcionamiento de una red están determinados por diversos factores que definen cómo las computadoras se conectan y comunican:

2.2.4.1. Extensión

El alcance geográfico de una red define su estructura, distinguiéndose tradicionalmente dos tipos principales:

- **Redes de área local (LAN) (Local Area Networks)**: Estas redes conectan computadoras en un área geográfica restringida, como un edificio o un grupo de edificios cercanos, con un alcance de hasta unos pocos kilómetros. Son comunes en oficinas y fábricas para compartir e intercambiar datos y aplicaciones. Su limitación espacial permite conocer el tiempo máximo de transmisión, simplificando el diseño, alcanzando velocidades de transmisión elevadas (10 Mbps a 10 Gbps) y presentando bajos retardos.
- **Redes de área amplia (WAN) (Wide Area Networks)**: Estas redes conectan LANs o servidores que se encuentran geográficamente separados, generalmente cuando la extensión supera los 10 kilómetros. En una WAN, las estaciones se interconectan a través de una subred, cuya función principal es transportar mensajes de un host a otro.

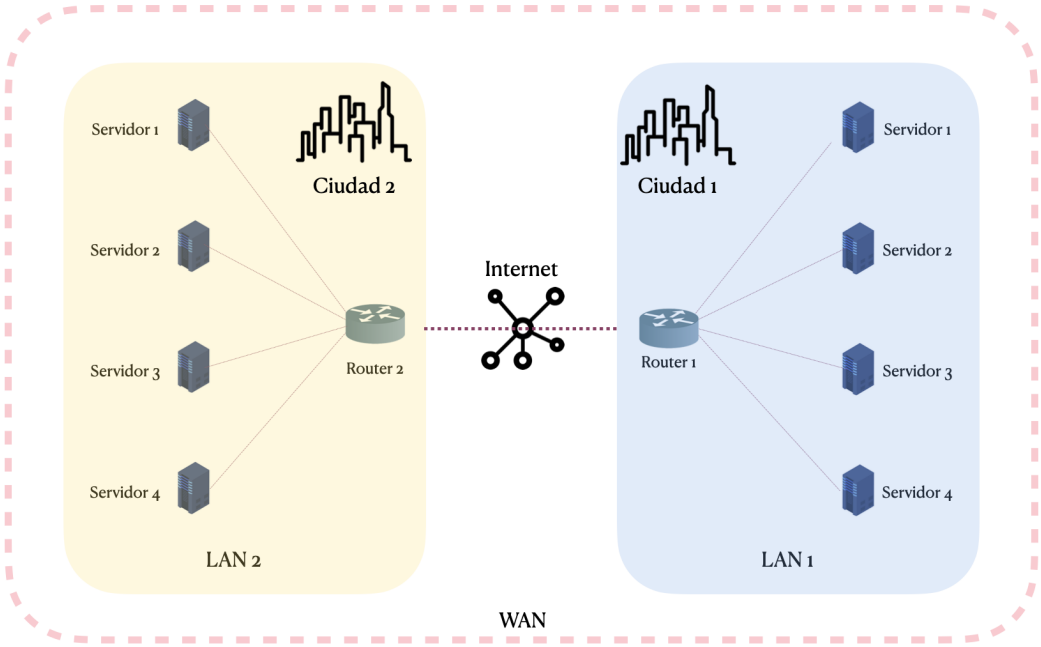


Figura 2.12: Extensión de la red

2.2.4.2. Topología

La topología de una red se refiere a cómo están interconectados los nodos en ella. Esta estructura determina aspectos importantes de la red como la facilidad de configuración, la fiabilidad y el rendimiento. Las dos principales topologías son:

- **Redes de difusión:** En estas redes, todos los nodos comparten un único medio de comunicación. Los nodos envían mensajes en forma de tramas o paquetes que ocupan el medio durante un cierto tiempo, y estos mensajes son recibidos por todos los demás nodos de la red. Lo que permite la transmisión en broadcast

(envío de mensajes a todos los usuarios simultáneamente) y multicast (envío a un subconjunto específico de máquinas). Aunque simplifican la red, requieren líneas de alta capacidad y un fallo en el medio puede afectar a toda la red.

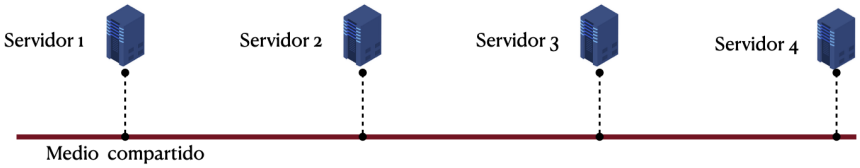


Figura 2.13: Topología red de difusión

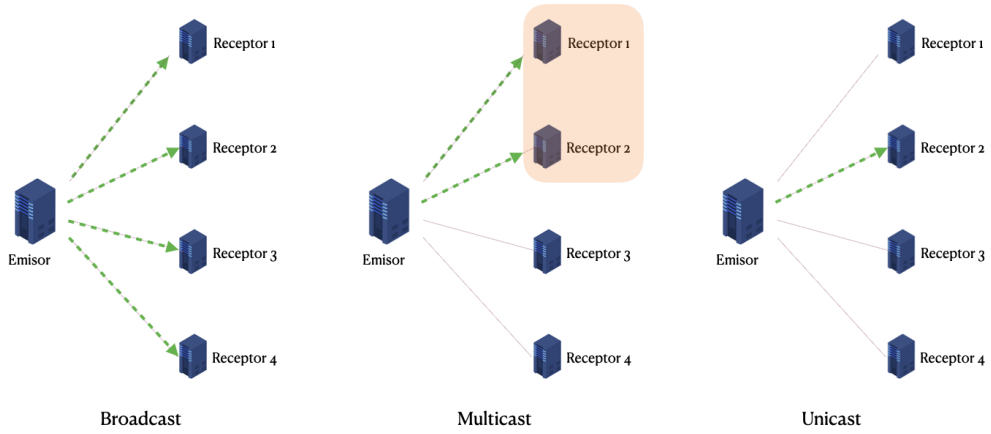


Figura 2.14: Envió de mensajes en la red

- **Redes punto a punto:** Estas redes consisten en múltiples conexiones entre pares de nodos, lo que aumenta la robustez ante fallos en los enlaces. Los mensajes pueden pasar por varios nodos intermedios antes de llegar a su destino, lo que puede generar retardo. La redundancia y robustez de la red dependen del número de conexiones por nodo, es decir, de su capacidad para encontrar rutas alternativas cuando la principal no está disponible.

Sobre estos dos tipos de estructura básicos de tecnologías de transmisión existen diferentes topologías usadas diferentes redes.

- **Total:** Cada par de estaciones está conectado por un canal punto a punto dedicado. Un canal es un medio o vía a través del cual se transmiten los datos entre dos o más puntos de la red. Los canales pueden operar simultáneamente, ofreciendo caminos alternativos en caso de fallos. Requiere muchos enlaces y tarjetas de interfaz, lo que limita su uso a redes pequeñas.
- **Parcial:** Algunos pares de estaciones tienen enlaces punto a punto, mientras que otros no, por lo que algunas transmisiones pueden requerir pasar por nodos intermedios. Esta configuración mantiene bajo el número de saltos intermedios y permite redundancia con múltiples caminos alternativos. Es ampliamente

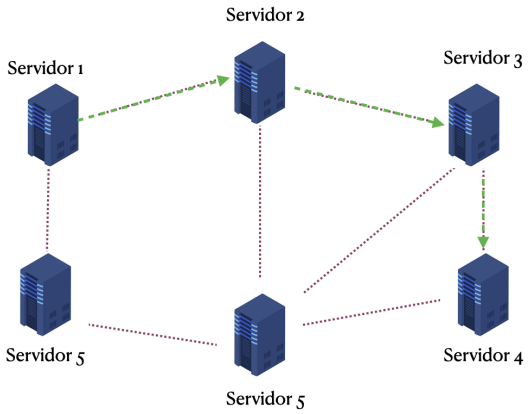


Figura 2.15: Comunicación entre servidor 1 y 4 en un red punto a punto

usada en redes de área amplia, permitiendo ajustar el tamaño de la red al tráfico existente. El coste depende del grado de interconexión requerido.

- **Estrella:** Todas las estaciones se conectan a un nodo central. Es de bajo costo y facilita la expansión, pero un fallo en el nodo central afecta a toda la red. Este método es común en la computación centralizada.
- **Árbol:** Se compone de topologías estrella donde los nodos centrales están interconectados. Esta topología se adapta bien a redes jerárquicas de múltiples niveles, aislando el tráfico y los problemas de una rama del resto de la red.
- **Bus:** Todas las estaciones comparten un medio de transmisión donde se difunde la información. Requiere mecanismos de acceso y detección de colisiones para evitar el acceso simultáneo al canal. Ofrece bajo coste de instalación y facilidad de expansión, y el software es simple al no requerir encaminamiento. Un problema en el bus afecta a toda la red, por lo que a menudo se duplica para añadir redundancia. El bus debe tener un ancho de banda elevado, ya que todo el flujo de datos pasa a través de él.
- **Anillo:** Cada estación se conecta con su vecina mediante un enlace unidireccional, y la comunicación sigue este camino hasta completar el lazo. Compartiendo el medio, es necesario regular el acceso. Es fácil añadir nuevas estaciones, pero esto añade retardos. Aunque el software es sencillo, un fallo en un enlace afecta a todo el anillo.

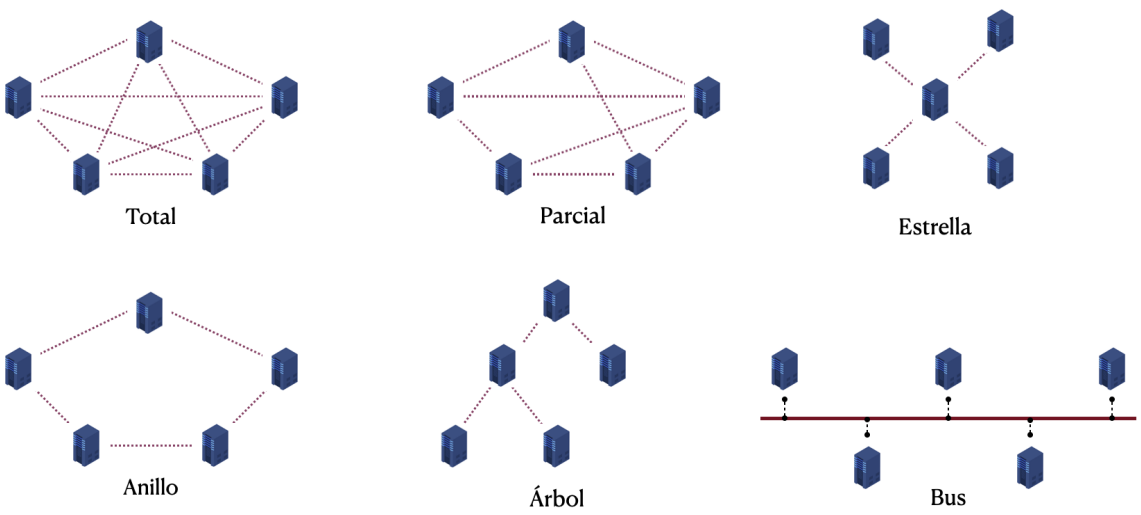


Figura 2.16: Topologías de red

Capítulo 3

Plan de Monitorización

En este capítulo se aborda la importancia de la monitorización de sistemas en el ámbito de las Tecnologías de la Información (TI) y su papel dentro de las empresas. Se realiza un análisis detallado de las alternativas disponibles, evaluando sus ventajas y limitaciones, para luego proponer una solución específica de monitorización para un prototipo a desarrollar en el siguiente capítulo.

3.1. Monitorización

La monitorización de sistemas de TI implica un proceso continuo de seguimiento y análisis del rendimiento de la infraestructura tecnológica. Este proceso es necesario para detectar y prevenir fallos tanto en sistemas como en aplicaciones. Para llevar a cabo esta tarea, se emplean una variedad de herramientas y software específicamente diseñados para supervisar y evaluar el funcionamiento, rendimiento y estado general de las infraestructuras. Estas herramientas abarcan la monitorización de servidores, redes, aplicaciones y otros componentes del sistema, permitiendo identificar posibles fallos, optimizar el rendimiento y mantener la integridad de los sistemas informáticos.

Para comprender la monitorización de sistemas, es importante conocer algunos términos clave: [22]

- **Métrica:** Medida cuantitativa para evaluar el rendimiento o eficiencia de un proceso o sistema. Las métricas pueden incluir elementos como tiempo de respuesta, utilización de recursos, cantidad de transacciones procesadas y disponibilidad del sistema. Son útiles para comprender el estado y comportamiento de los sistemas y tomar decisiones informadas sobre su gestión y mejora.
- **Frecuencia de medición:** Regularidad con la que se recopilan datos de un sistema o proceso que determina la actualidad y relevancia de la información monitorizada. Una frecuencia adecuada asegura una recolección de datos suficiente para análisis efectivos, sin sobrecargar el sistema.
- **Perspectiva de monitorización:** Define los aspectos específicos del sistema que se monitorizan y cómo se interpretan los datos recogidos. Por ejemplo, puede enfocarse en el rendimiento del sistema desde la perspectiva del usuario final o en aspectos técnicos internos, como el uso de recursos del servidor. La elección depende de los objetivos y necesidades de la organización.

- **Alerta:** Notificación automática cuando se detectan condiciones anómalas o umbrales predefinidos.
- **Rendimiento:** Evaluación del funcionamiento de un sistema. Implica medir y analizar parámetros como velocidad de procesamiento, capacidad de respuesta, utilización de recursos y eficiencia general para identificar áreas que requieran mejoras o ajustes.
- **Uso de Recursos:** Medición y análisis del consumo de recursos como CPU, memoria o ancho de banda. Ayuda a entender la carga de trabajo del sistema, identificar cuellos de botella y optimizar el rendimiento.
- **Disponibilidad:** Tiempo durante el cual un sistema está operativo y accesible.
- **Salud del Sistema:** Evaluación integral del estado y funcionamiento de un sistema.
- **Tendencia:** Identificación de patrones o cambios en el rendimiento uso o comportamiento de los sistemas a lo largo del tiempo.

3.2. Análisis de Herramientas de Monitorización

Una herramienta de monitorización es un software diseñado para supervisar y analizar el rendimiento y la salud de componentes de un sistema, como redes, servidores, aplicaciones y dispositivos. Proporciona datos importantes para garantizar un funcionamiento eficiente y seguro de los recursos de TI.

Una herramienta de monitorización puede abarcar varios aspectos del sistema o enfocarse solo en tareas específicas:

- **Monitorización del Rendimiento**: Permite medir y evaluar los niveles de rendimiento de los sistemas de TI. Esto incluye aspectos como el tiempo de actividad del servidor, los tiempos de respuesta de las aplicaciones y las tasas de utilización de recursos como la CPU, la memoria y el uso de disco. Permite comprender cómo están funcionando los recursos y si están siendo utilizados eficientemente.
- **Monitorización de la Red**: Revisa la salud y el rendimiento de los componentes de la red, como routers, switches, firewalls y redes inalámbricas. Realiza un seguimiento del tráfico de red, el uso del ancho de banda y las tasas de error, proporcionando información vital sobre la integridad y el funcionamiento de la red.
- **Monitorización de la Seguridad**: Escanea la infraestructura con el objetivo de mantener la integridad y seguridad de los sistemas de TI. Monitoriza actividades sospechosas, brechas de seguridad y vulnerabilidades potenciales para proteger los sistemas contra amenazas internas y externas.
- **Alertas y Notificaciones**: Informa al personal de TI sobre posibles problemas de forma proactiva. Permite detectar y responder rápidamente a incidentes, minimizando su impacto en el sistema.
- **Análisis de Datos e Informes**: Permite analizar los datos recopilados y generar informes detallados. Estos informes son útiles para identificar tendencias a lo

largo del tiempo, predecir problemas futuros y respaldar la toma de decisiones informadas en la gestión de TI.

A la hora de elegir una herramienta de monitorización es necesario asegurarse de que la herramienta seleccionada no solo cumple las necesidades actuales de la organización, sino que también sea lo suficientemente flexible y escalable para adaptarse a los cambios futuros y al crecimiento de la infraestructura de TI.

La elección de la herramienta de monitorización adecuada para una organización es un proceso complejo y que requiere tiempo debido a la variedad de factores a considerar: [23]

- **Elementos a Monitorizar:** Identificar los componentes de la infraestructura de TI y procesos de negocio que necesitan ser supervisados.
- **Compatibilidad con Aplicaciones:** Asegurar que la herramienta sea compatible con las aplicaciones y plataformas utilizadas en la organización.
- **Sistema de Licencias:** Evaluar las opciones de licencia según la cantidad y tipo de elementos a monitorizar.
- **Facilidad de Configuración y Gestión:** Analizar la complejidad de la configuración y el manejo diario de la herramienta.
- **Escalabilidad:** Verificar si la herramienta puede adaptarse al crecimiento y cambios en la infraestructura.
- **Integración mediante API:** Comprobar la disponibilidad de APIs para integrar elementos externos al sistema de monitorización.
- **Gestión de Alertas:** Asegurar un sistema de gestión de alertas flexible que permita personalización de canales y políticas de envío.
- **Monitorización de Virtualización:** Capacidad para supervisar máquinas virtuales e infraestructuras de virtualización.
- **Inventariado y Visualización:** Creación de un inventario y visualización clara de los componentes del sistema.
- **Informes de Estado del Sistema:** Generar y distribuir informes detallados sobre el estado y rendimiento del sistema.
- **Monitorización con/sin Agentes:** Ofrecer opciones de monitorización con o sin agentes instalados en los dispositivos.
- **Monitorización Híbrida para la Nube:** Soporte para la supervisión de infraestructuras tanto en la nube como locales.
- **Análisis Histórico:** Capacidad de almacenar y analizar datos históricos para identificar tendencias.
- **Panel Personalizable:** Disponibilidad de interfaces y paneles adaptables a las necesidades del usuario.

En el contexto actual, el sector de monitorización de TI se caracteriza por una notable fragmentación. La mayoría de las empresas utilizan una combinación de dos o tres herramientas diferentes para supervisar y mantener sus sistemas de TI. Esta práctica

refleja la diversidad de necesidades y la complejidad de las infraestructuras tecnológicas modernas, lo que significa que una sola herramienta rara vez es suficiente para cubrir todas las áreas de monitorización necesarias.

3.2.1. Herramientas de Monitorización

3.2.1.1. Nagios

Nagios [24] es una herramienta de monitorización de sistemas, redes e infraestructuras de TI, que permite detectar problemas con antelación y evitar interrupciones en los procesos empresariales. Su popularidad se debe a varias características clave:

- **Flexibilidad:** Su arquitectura es extensible, permitiendo añadir plugins y adaptar la herramienta a necesidades específicas. Los plugins son programas o scripts que realizan comprobaciones específicas sobre el estado de un elemento del sistema y devuelven códigos de estado (OK, WARNING, CRITICAL, UNKNOWN). Los usuarios pueden crear plugins personalizados o acceder a una amplia biblioteca de soluciones.
- **Alertas:** Permite la configuración de alertas a través de correos electrónicos, SMS y scripts personalizados para informar en tiempo real de anomalías.
- **Plantillas:** Facilita la configuración y gestión mediante plantillas preconfiguradas, que aplican un conjunto estándar de propiedades a diferentes objetos.
- **Agentes:** Ofrece monitorización con y sin agentes. Un agente es un software instalado en el dispositivo a monitorizar que realiza tareas específicas.
  - **Monitorización Basada en Agentes:** Requiere la instalación de software en el host remoto que permite recopilar datos del sistema y enviarlos de vuelta al servidor de Nagios.
    - **NRPE (Nagios Remote Plugin Executor):** Utilizado en Linux/Unix para ejecutar plugins localmente y enviar resultados al servidor Nagios.
    - **NSClient++:** Agente para sistemas Windows.
  - **Monitorización Sin Agentes:** Utiliza protocolos estándar de red y servicios disponibles en el host para realizar las comprobaciones.
  - **SNMP (Simple Network Management Protocol):** [25] Es un protocolo estándar para la gestión de dispositivos en redes IP. Permite monitorizar una amplia gama de dispositivos, recopilando información y recibiendo alertas sin acceder a ellos.
  - **SSH (Secure Shell):** Nagios puede conectarse a servidores cliente mediante SSH para ejecutar plugins localmente y verificar funciones internas como la carga de CPU, memoria, procesos, etc. La seguridad es una ventaja significativa de este método, aunque puede ser más demandante en términos de recursos que otros tipos de comprobaciones.

  La combinación de estos enfoques permite ofrecer una cobertura de monitorización amplia y adaptable para una gran variedad de dispositivos y servicios, desde servidores y estaciones de trabajo hasta dispositivos de red y aplicaciones.
- **Chequeos:** Se realiza la monitorización a través de chequeos activos y pasivos.

- **Chequeos Activos:** Nagios inicia la verificación del estado de servidores y servicios en intervalos de tiempo regulares definidos en la configuración y utilizando scripts o ejecutables específicos.

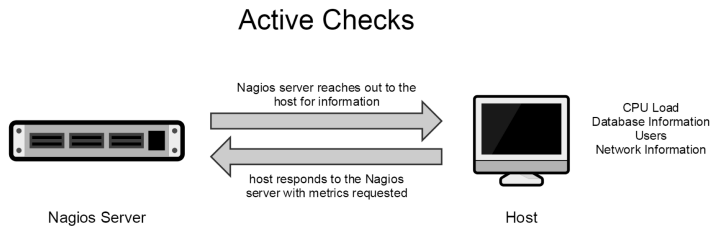


Figura 3.1: Chequeos activos

- **Chequeos Pasivos:** Los resultados de la monitorización son enviados a Nagios sin una solicitud previa activa.

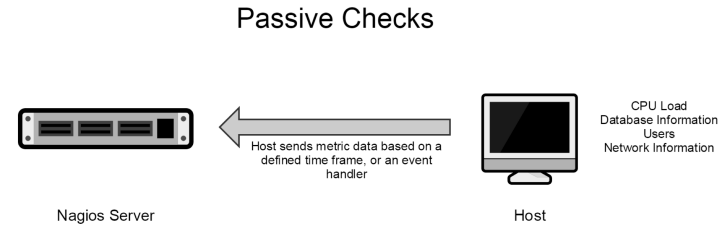


Figura 3.2: Chequeos pasivos

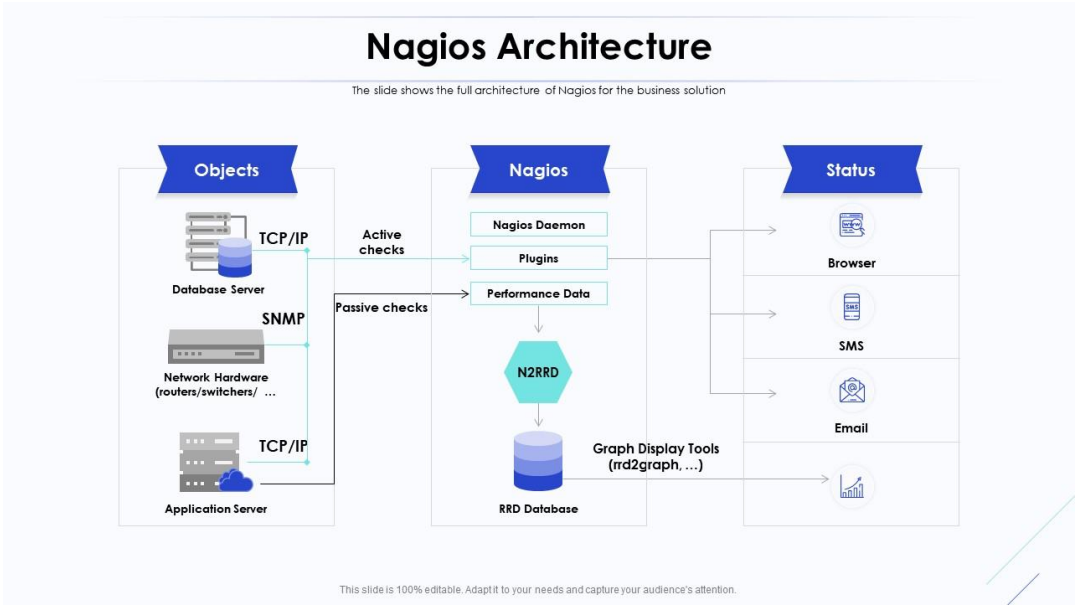


Figura 3.3: Arquitectura Nagios

- **Licencia:** [26] Licencia Pública General (GPL) de GNU. Presenta dos versiones que ofrecen funcionalidad básica de monitorización, pero que difieren en características, usabilidad y precio.

  - **Nagios Core:** [27] Es la versión original y gratuita de Nagios, enfocada en proveer un sistema de monitorización básico y robusto. Permite a los usuarios desarrollar y modificar plugins para personalizar el sistema según sus necesidades. Sin embargo, la configuración y el mantenimiento pueden ser más complejos, y su interfaz de usuario es básica y menos intuitiva, lo que puede representar un desafío para usuarios menos experimentados.

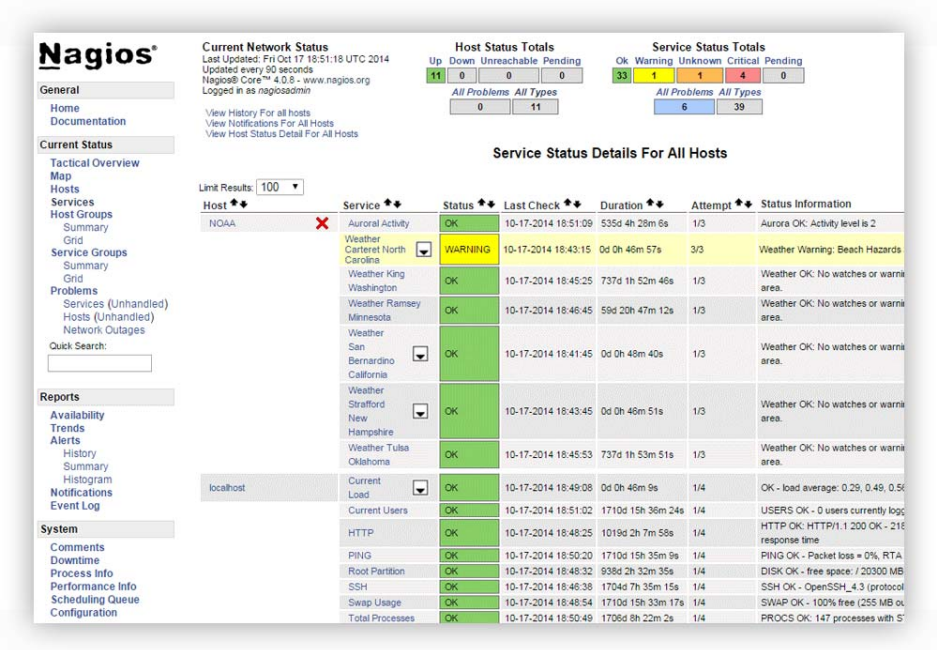


Figura 3.4: Interfaz Nagios Core

  - **Nagios XI:** Es la versión comercial de Nagios, que amplía las capacidades de Nagios Core con una interfaz de usuario mejorada y herramientas de configuración web. Nagios XI está dirigido a empresas que buscan una solución más accesible y menos técnica, ofreciendo soporte y actualizaciones regulares.

En resumen, Nagios se ha consolidado como una herramienta de monitorización altamente valorada y extensamente utilizada en el sector de TI, reconocida por su versatilidad y código abierto. Como es una de las soluciones pioneras en este campo, Nagios ha establecido un estándar en la industria de desde su lanzamiento. Sus dos versiones proporcionan un amplio rango de funcionalidades, adaptándose tanto a las necesidades de pequeñas empresas como a las de grandes corporaciones.

Su popularidad además se atribuye a su amplia funcionalidad, que permite a los usuarios con conocimientos avanzados configurar y monitorizar escenarios específicos y complejos de manera efectiva. La existencia de una comunidad activa y experimentada es un factor clave que facilita su implementación y soporte. Nagios ofrece una gran variedad de plugins, permitiendo adaptar la herramienta

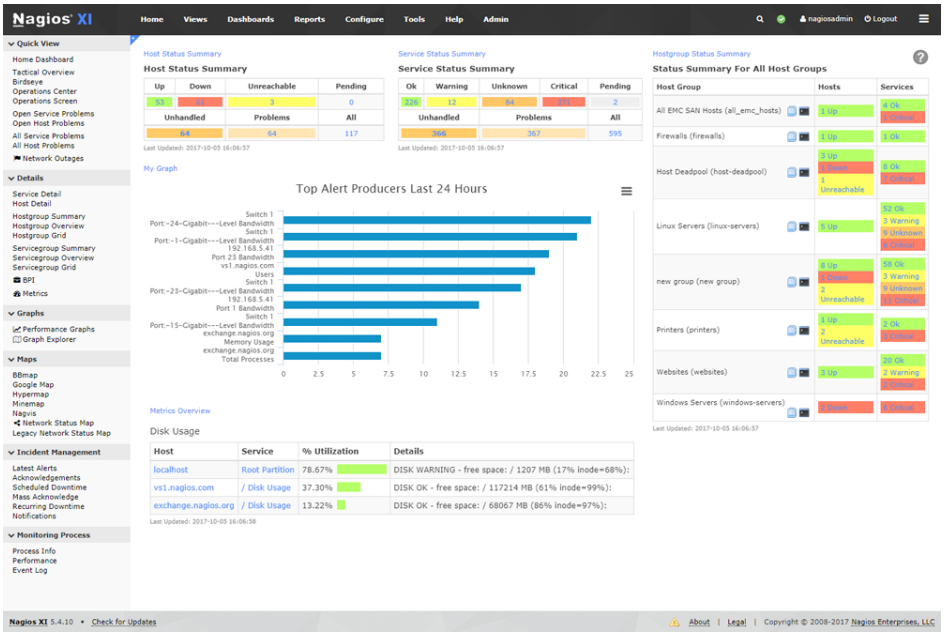


Figura 3.5: Interfaz Nagios XI

a diversas necesidades y preferencias. A pesar de su potencial avanzado, la configuración básica de Nagios es relativamente sencilla, lo que la hace accesible incluso para usuarios menos experimentados.

No obstante, Nagios enfrenta ciertos desafíos. La configuración y edición avanzadas pueden ser complejas, requiriendo múltiples modificaciones manuales, lo que a menudo conduce a una implementación personalizada que puede ser difícil de mantener y evolucionar. La interfaz gráfica de usuario de Nagios es poco intuitiva y su curva de aprendizaje es elevada. Además, los informes generados son básicos y su capacidad para manejar el Protocolo Simple de Administración de Red (SNMP) es limitada.

Estos aspectos pueden representar desafíos significativos, especialmente debido a la falta de instrucciones en ciertos idiomas, como el español.

3.2.1.2. Zabbix

Zabbix [28] es una solución integral de monitorización de código abierto, adecuada para empresas de cualquier tamaño. Esta herramienta ha sido diseñada para controlar diversos parámetros de red y evaluar la salud de servidores, máquinas virtuales, aplicaciones y más. Se destaca por su eficiencia en la generación de informes y visualización de datos, lo que resulta extremadamente útil para la planificación de capacidad y la gestión de recursos de TI.

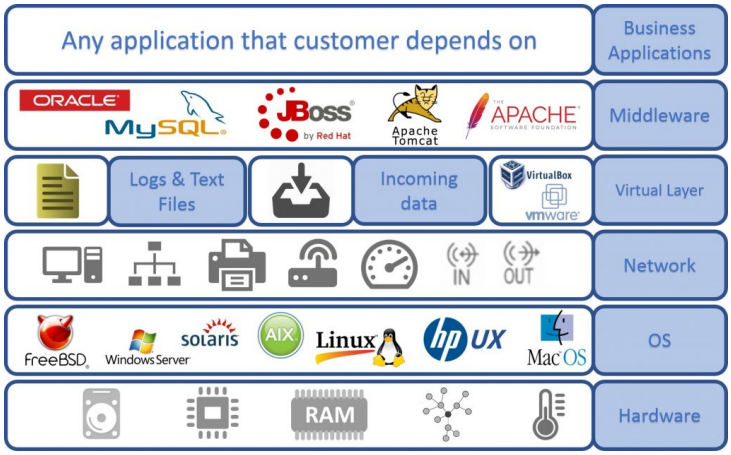


Figura 3.6: Sistemas compatibles

La popularidad y eficacia de Zabbix se deben a sus características y funcionalidades clave, que la convierten en una herramienta potente y versátil para la monitorización.

- **Componentes:**
  - **Servidor:** Actúa como el núcleo central, donde los agentes reportan información sobre disponibilidad, integridad y estadísticas. Es el depósito principal para todos los datos de configuración y estadísticas.
  - **Agentes:** Monitorizan recursos y aplicaciones locales, enviando datos al servidor.
    - **Agente Zabbix:** Escrito en C, este agente es eficiente y compatible con muchos sistemas operativos. Realiza controles pasivos y activos, iniciando o respondiendo a solicitudes del servidor.
    - **Agente Zabbix 2:** Desarrollado en Go, maneja múltiples tareas simultáneamente, mejorando el rendimiento en entornos de alta carga. Es extensible con complementos y optimiza las conexiones TCP, ideal para grandes implementaciones.

  Al igual que Nagios, Zabbix ofrece amplia compatibilidad en la monitorización de sistemas, incluyendo el sondeo y la captura de datos. Puede realizar monitorización sin agentes o utilizar agentes nativos para recopilar datos de sistemas operativos comunes, utilizando SNMP, IPMI, JMX o monitorización de VMware.

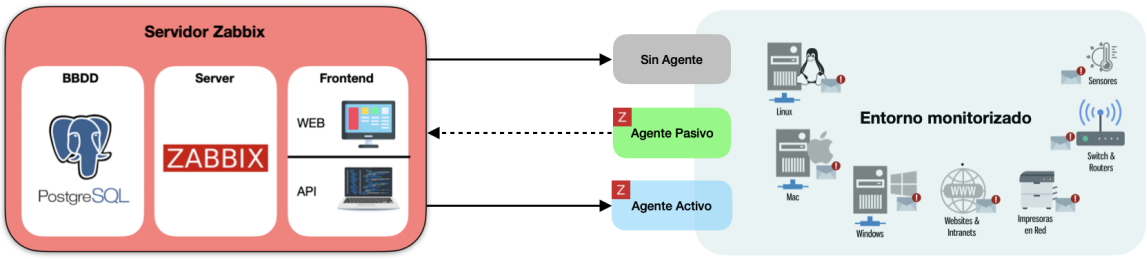


Figura 3.7: Arquitectura de Zabbix

- **Chequeos:** Los agentes de Zabbix realizan controles pasivos (responden a solicitudes del servidor) y activos (recuperan y envían datos periódicamente al servidor). La elección depende del tipo de elemento de seguimiento.
- **Flujo de Monitorización:** Para recopilar datos el proceso comienza creando un host, que representa el dispositivo o sistema que se va a monitorizar, como un servidor específico (por ejemplo, servidor X). Dentro de este host, se define un elemento, que es la métrica específica que se desea monitorizar, como la carga de la CPU.

  Una vez configurado el elemento, se crea un activador o trigger que define las condiciones bajo las cuales se activará una acción específica, como enviar una alerta por correo electrónico. Por ejemplo, se puede configurar un activador para que dispare una alerta cuando la carga de la CPU del servidor X supere un umbral predefinido.

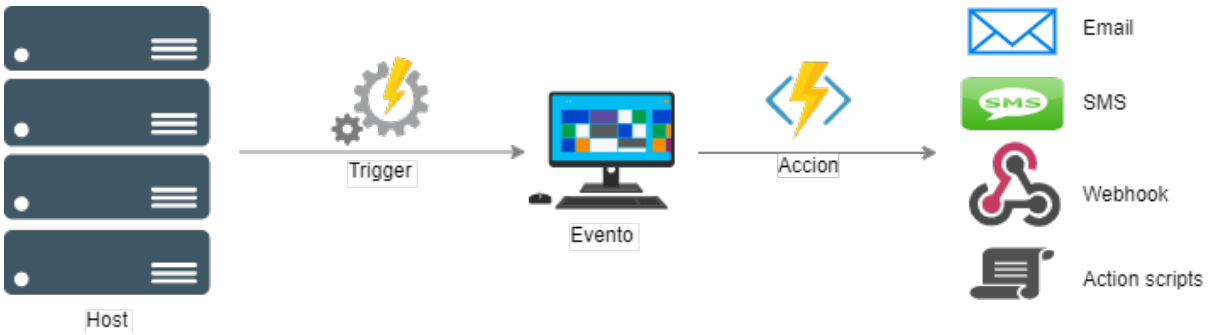


Figura 3.8: Fujo Zabbix

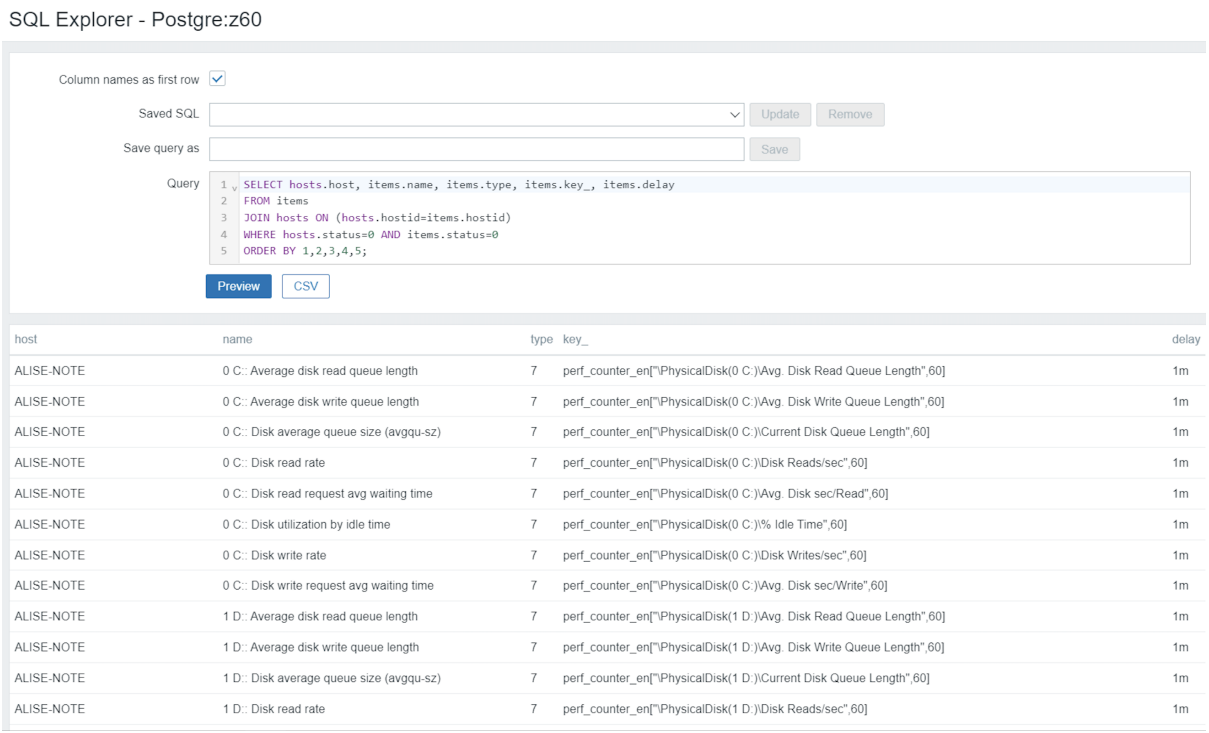
- **Plantillas:** Zabbix utiliza un conjunto predeterminado de reglas que abarcan dispositivos, disparadores (triggers), gráficos, aplicaciones y, en algunos casos y reglas de autocreación. Estas plantillas se aplican a múltiples hosts o dispositivos para estandarizar y simplificar la configuración, evitando configuraciones repetitivas individuales.
- **Descubrimiento Automático:** Cuenta con una función de autodescubrimiento que facilita la detección de nuevos dispositivos o cambios en la red. Esta capacidad optimiza la administración y actualización de la infraestructura al supervisar automáticamente dispositivos de red y detectar cambios de configuración para notificar actualizaciones.

- **Concurrencia:** Permite la ejecución simultánea de comprobaciones de diferentes complementos.
- **Alertas:** Ofrece un sistema flexible de notificaciones que permite configurar alertas mediante correo electrónico, SMS o Jabber, adaptándose a diversas necesidades y preferencias de comunicación.
- **Automatización:** Facilita la automatización de tareas y respuestas a eventos mediante la ejecución de scripts personalizados.

Figura 3.9: Interfaz Zabbix

- **Sistema de Permisos:** Implementa un sistema seguro de autenticación de usuarios que permite establecer restricciones de acceso a vistas específicas según el perfil del usuario. Esta característica garantiza que los usuarios solo accedan a la información y funciones relevantes para sus roles.
- **API:** Proporciona una interfaz de programación de aplicaciones (API) que facilita la manipulación masiva de datos, la integración con software de terceros y otros propósitos. Un ejemplo de su utilidad es la capacidad de simular clics de ratón en sitios web para verificar funcionalidades y tiempos de respuesta, ampliando significativamente las capacidades de monitorización y automatización.
- **Licencia:** Es un software gratuito desarrollado y distribuido bajo la Licencia Pública General GNU versión 2 (GPLv2). Aunque el software en sí no tiene coste, se ofrecen suscripciones de servicio técnico que permiten a los usuarios acceder a soporte profesional para sus implementaciones.

Zabbix es altamente flexible y escalable, adecuado para diversos entornos, desde pequeñas empresas hasta grandes corporaciones con hasta 10,000 servidores,



host	name	type	key_	delay
ALISE-NOTE	0 C:: Average disk read queue length	7	perf_counter_en["\PhysicalDisk(0 C:)\Avg. Disk Read Queue Length",60]	1m
ALISE-NOTE	0 C:: Average disk write queue length	7	perf_counter_en["\PhysicalDisk(0 C:)\Avg. Disk Write Queue Length",60]	1m
ALISE-NOTE	0 C:: Disk average queue size (avgqu-sz)	7	perf_counter_en["\PhysicalDisk(0 C:)\Current Disk Queue Length",60]	1m
ALISE-NOTE	0 C:: Disk read rate	7	perf_counter_en["\PhysicalDisk(0 C:)\Disk Reads/sec",60]	1m
ALISE-NOTE	0 C:: Disk read request avg waiting time	7	perf_counter_en["\PhysicalDisk(0 C:)\Avg. Disk sec/Read",60]	1m
ALISE-NOTE	0 C:: Disk utilization by idle time	7	perf_counter_en["\PhysicalDisk(0 C:)\% Idle Time",60]	1m
ALISE-NOTE	0 C:: Disk write rate	7	perf_counter_en["\PhysicalDisk(0 C:)\Disk Writes/sec",60]	1m
ALISE-NOTE	0 C:: Disk write request avg waiting time	7	perf_counter_en["\PhysicalDisk(0 C:)\Avg. Disk sec/Write",60]	1m
ALISE-NOTE	1 D:: Average disk read queue length	7	perf_counter_en["\PhysicalDisk(1 D:)\Avg. Disk Read Queue Length",60]	1m
ALISE-NOTE	1 D:: Average disk write queue length	7	perf_counter_en["\PhysicalDisk(1 D:)\Avg. Disk Write Queue Length",60]	1m
ALISE-NOTE	1 D:: Disk average queue size (avgqu-sz)	7	perf_counter_en["\PhysicalDisk(1 D:)\Current Disk Queue Length",60]	1m
ALISE-NOTE	1 D:: Disk read rate	7	perf_counter_en["\PhysicalDisk(1 D:)\Disk Reads/sec",60]	1m

Figura 3.10: Informe Zabbix

aunque puede enfrentar desafíos de rendimiento más allá de este límite. Su capacidad para manejar múltiples protocolos de monitorización y su interfaz intuitiva simplifican el análisis de datos complejos, mejorando significativamente la gestión de sistemas y redes. Las avanzadas funciones de alertas y notificaciones aseguran respuestas rápidas a incidentes, garantizando la continuidad y estabilidad de los servicios de TI, lo que convierte a Zabbix en una solución integral y confiable para la monitorización en diversos contextos.

Zabbix destaca como una opción robusta para la monitorización de TI, gracias a su notable flexibilidad y escalabilidad. Esta herramienta es ideal para una amplia variedad de entornos, siendo efectiva en pequeñas empresas y también en grandes corporaciones con hasta 10.000 servidores, aunque más allá de este límite puede empezar a experimentar problemas de rendimiento.

3.2.1.3. Centreon

Centreon es una solución avanzada de monitorización de TI conocida por su flexibilidad, potencia y facilidad de uso, adaptándose eficazmente a una amplia gama de entornos empresariales. Su diseño esta basado en Nagios, pero ofrece con numerosas mejoras que lo hacen ideal tanto para pequeñas empresas como para grandes corporaciones.

- **Modo de instalación:** Centreon ofrece dos opciones de instalación:
  - **Modo SaaS (Software as a Service):** Permite a los usuarios acceder al software a través de internet sin necesidad de instalación local. Es utilizado mediante navegadores web, con acceso mediante suscripción, lo

que elimina la necesidad de gestionar mantenimientos y actualizaciones. Este enfoque es flexible, escalable y tiene menor coste, esta pensado para empresas que buscan acceso global y concentrarse en sus operaciones principales sin preocuparse por la infraestructura de TI.

- **Modo autoalojado:** Permite a los usuarios instalar y gestionar el software en su propia infraestructura. Esto aporta control total sobre la implementación y configuración del sistema, adecuándose a necesidades específicas de seguridad y rendimiento. Esta pensado para organizaciones que prefieren mantener datos sensibles dentro de su propia red y requieren una configuración más personalizada.

- **Componentes:**[29] La arquitectura de Centreon varía según la escala de los recursos a monitorizar. Un servidor central es adecuado para cargas pequeñas, pero para entornos más grandes se recomienda una estructura distribuida. Esto incluye un servidor central para configuración y supervisión, servidores remotos para grandes extensiones geográficas y pollers para distribuir eficientemente la carga.

  - **Servidor Central:** Funciona como la consola principal para la supervisión de la infraestructura de TI. Es el núcleo de la monitorización, permitiendo configurar, supervisar y gestionar todos los recursos. Está compuesto por:

    - **Centreon Web:** Interfaz de usuario que proporciona un panel visual para que administradores y usuarios visualicen el estado y rendimiento de la infraestructura TI. Facilita la visualización en tiempo real y la generación de gráficos e informes detallados.

    - **Centreon Broker:** Gestiona datos y eventos en la arquitectura de Centreon. Procesa y transmite datos recolectados por el Centreon Engine hacia la base de datos y otras áreas del sistema, asegurando un flujo eficiente de información.

    - **Centreon Engine:** Motor de recolección de datos basado en Nagios. Realiza comprobaciones de estado en recursos y servicios de red, recopilando datos de rendimiento y disponibilidad para evaluar la salud y eficiencia operativa.

  - **Servidor Remoto:** Amplía la capacidad de monitorización a ubicaciones geográficas diversas y segmentos de red, aliviando la carga del servidor central. Facilita la monitorización en áreas con conectividad limitada, manteniendo la seguridad y aislamiento de la red.

    Cuenta con su propio Centreon Engine, lo que asegura una supervisión constante y efectiva de los recursos. Aunque posee una interfaz gráfica, esta carece de menús de configuración, enfocándose más en la presentación de la información.

    Un aspecto destacado de este sistema es su capacidad para visualizar los datos. Los recursos supervisados se muestran tanto en su propia interfaz como en la del servidor central, proporcionando una visión integral y accesible del estado de los recursos.

- **Poller:** Sistema conectado al servidor central o remoto, se enfoca en la supervisión activa de recursos. No tiene interfaz gráfica propia; muestra datos en la interfaz del servidor al que esta conectado. Distribuye eficientemente la carga de monitorización en redes grandes o complejas para una supervisión escalable y eficiente.

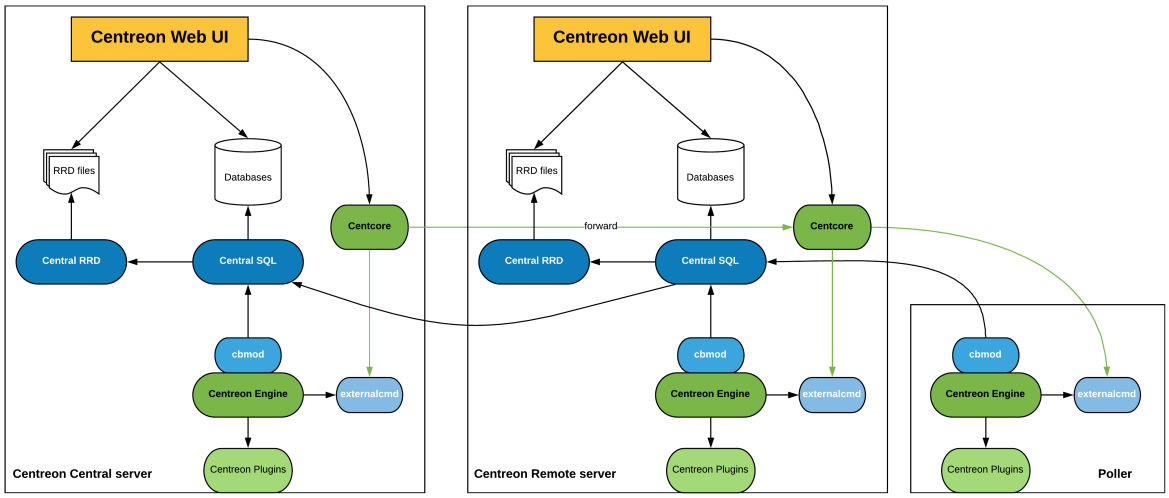


Figura 3.11: Arquitectura Centeon

- **Agentes:** los chequeos pueden realizarse con o sin agentes, dependiendo de las necesidades y configuración de la infraestructura:
  - **Chequeos Sin Agente**: También conocidos como chequeos pasivos o basados en SNMP (Simple Network Management Protocol). Son comunes para dispositivos de red como switches, routers, impresoras, y otros equipos que soportan SNMP.
  - **Chequeos con Agente**: Los agentes permiten recolectar datos más detallados del sistema, como el uso de CPU, memoria, disco y procesos en ejecución, y luego envían esta información a Centreon. Un ejemplo común de agente utilizado en sistemas de monitorización basados en Nagios es NRPE (Nagios Remote Plugin Executor).

  La elección entre chequeos con o sin agente depende de varios factores como la profundidad y tipo de datos requeridos, la infraestructura de la red, y las preferencias de seguridad y administración de la organización. Los chequeos sin agente son más simples de configurar y mantener, mientras que los chequeos con agente pueden proporcionar información más específica y detallada del sistema. En muchos entornos, una combinación de ambos métodos ofrece un equilibrio entre cobertura, precisión de los datos y complejidad de la configuración.
- **Escalabilidad:** Centreon es altamente escalable, gestionando eficientemente grandes volúmenes de datos de dispositivos y servicios, adaptándose fácilmente al crecimiento empresarial para asegurar una monitorización efectiva en cualquier escala.
- **Visualización de datos y la generación de reportes:** ofrece avanzadas

capacidades de visualización con mapas geográficos y de topología de red, facilitando representaciones visuales detalladas de la infraestructura monitorizada. Además, permite la personalización y exportación de informes en múltiples formatos para una gestión y análisis flexibles de los datos recolectados.

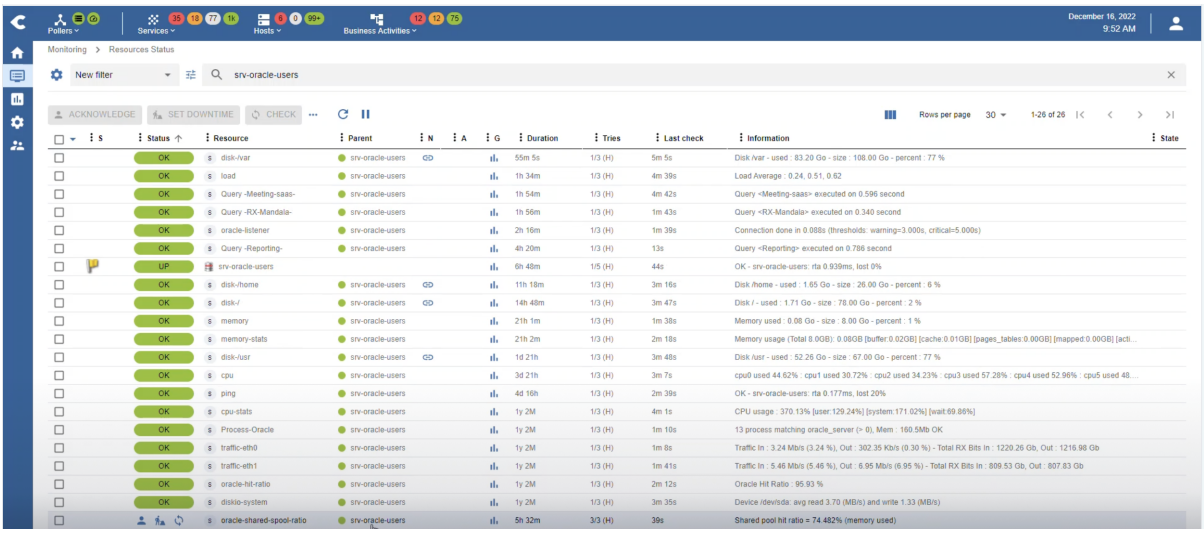


Figura 3.12: Interfaz gráfica Centreon

Figura 3.13: Monitorización aplicaciones Centreon

- **Plugins**: Son compatibles con Nagios. Permiten realizar chequeos específicos en dispositivos y servicios mediante una amplia biblioteca desarrollada por la comunidad de Nagios.
- **Plantillas**: Simplifican la gestión al permitir predefinir configuraciones para hosts y servicios, asegurando consistencia y facilitando actualizaciones en toda la red.

- **Automatización de Tareas**: Permite la automatización de respuestas a eventos comunes, optimizando el uso de recursos.
- **Descubrimiento Automático:**[30] Utilizando el módulo Auto Discovery, Centreon puede identificar automáticamente nuevos hosts y servicios en la red, agregándolos de manera automática a la plataforma.
- **Alertas:** Ofrecen una alta personalización, permitiendo a los usuarios definir tanto el momento como el modo en que se reciben estas alertas, incluyendo opciones como correo electrónico, SMS y aplicaciones móviles.
- **Soporte para Virtualización y Cloud**: Ofrece compatibilidad con entornos de virtualización y soluciones en la nube, facilitando la monitorización de infraestructuras modernas y dinámicas.
- **Licencia:**[31] Centreon está bajo licencia GPL (General Public License) versión 2, lo que significa que es software de código abierto, permitiendo su uso, modificación y distribución libremente. Además de la versión de código abierto, Centreon ofrece ediciones empresariales con funcionalidades adicionales y soporte dedicado, sujetas a licencias comerciales.

Centreon se presenta como una solución integral para la monitorización de TI, que abarca desde la supervisión básica hasta la gestión avanzada de eventos y rendimiento. Su combinación de escalabilidad, personalización y potentes capacidades de análisis lo convierten en la opción ideal para cualquier organización que desee optimizar y proteger su infraestructura de TI, asegurando eficiencia y estabilidad en sus operaciones.

Centreon proporciona una interfaz simplificada que facilita la visualización del estado del sistema para una variedad amplia de usuarios, desde no técnicos hasta gráficos de rendimiento explícitos. Sin embargo, los técnicos siempre tienen acceso a la información avanzada del planificador.

Capítulo 4

Diseño del Proyecto

En este capítulo, se presenta de manera detallada el proceso de diseño de una solución integral para la monitorización de sistemas, acompañado de una propuesta de automatización eficiente. Se incluye un análisis exhaustivo de los requisitos, las técnicas empleadas, la metodología de desarrollo y la planificación ejecutada. Además, se introducen las herramientas seleccionadas para el desarrollo del proyecto y se propone una solución automatizada.

El proyecto se desarrolla en un escenario inicial que simula la infraestructura de una pequeña empresa con múltiples servidores y aplicaciones distribuidas. Además, se ha realizado previamente una labor de documentación y se ha propuesto diversas herramientas de monitorización, evaluando sus ventajas y desventajas para el desarrollo del prototipo.

4.1. Metodología de Desarrollo

Una metodología de desarrollo de software[32] es un conjunto estructurado de prácticas, técnicas, procedimientos y roles organizativos que guían el proceso de creación de software desde la idea hasta la entrega final y mantenimiento.

En este caso, el objetivo no es la creación de un sistema final, sino la implementación de un prototipo funcional para estudiar su viabilidad. Para este propósito, se empleará la metodología de desarrollo de prototipos [33], la cual forma parte de los modelos de desarrollo evolutivo.

Esta metodología se basa en ajustar continuamente el prototipo para cumplir con las necesidades emergentes. Esto ayuda al desarrollador a entender mejor los requisitos y permite al cliente ver resultados rápidos. Es útil cuando los objetivos generales del software están claros, pero los detalles específicos aún no están definidos por completo.

Se pueden dividir las actividades en 4 etapas bien definidas:

1. **Análisis del sistema**: Se identifican los requisitos necesarios para el desarrollo del prototipo funcional. Debe incluir funcionalidades que aborden los problemas y los solucionen total o parcialmente.

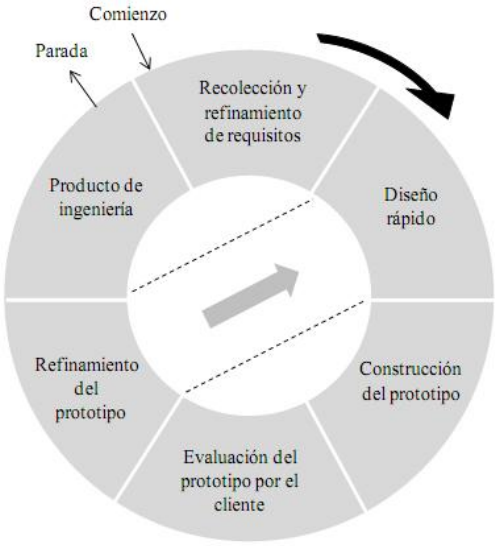


Figura 4.1: Metodología de desarrollo de prototipos.

2. **Diseño rápido del prototipo**: Se define la implementación o las modificaciones a realizar, junto con las tecnologías y el flujo a seguir.
3. **Construcción de un prototipo**: Se realiza un prototipo funcional basado en el diseño de la fase anterior.
4. **Pruebas y mejoras del prototipo**: Se realizan pruebas para verificar el funcionamiento correcto y el cumplimiento de los requisitos identificados. Se presenta el prototipo al cliente para su evaluación. Tras las pruebas, se recibe retroalimentación y se proponen cambios necesarios para futuras implementaciones, volviendo así al inicio del ciclo.

4.2. Análisis del Sistema

Los requisitos tienen como objetivo resolver problemas del mundo real o automatizar procesos y tareas para el usuario. También se utilizan para agregar nuevas funcionalidades al software o corregir el mal funcionamiento de uno existente. De esta manera, esta sección facilitará la comprensión del diseño, reduciendo el tiempo necesario para poner en funcionamiento todos los componentes involucrados.

Los requisitos se dividen principalmente en funcionales y no funcionales [34]:

- **Requisitos funcionales:** Describen las acciones que el sistema debe hacer. Dependen del software a desarrollar y en su mayor parte se redactan de una forma abstracta. Solo se define su función, entradas y salidas.
  1. Monitorización del estado de los servidores.
  2. Generación de informes e históricos sobre la salud de los servidores.
  3. Visualización gráfica del estado del sistema.
  4. Generación de alertas ante situaciones anómalas.

5. Envío de alertas a través de varios canales.
6. Automatización de la configuración de la herramienta de monitorización.
7. Automatización de tareas básicas al recibir una alerta.

- **Requisitos no funcionales:** Describen otras características, prestaciones y limitaciones que debe cumplir el sistema.
  1. Escalabilidad.
  2. Facilidad de uso.
  3. Infraestructura bien definida y flexible que permita modificaciones.
  4. Tolerancia del sistema ante fallos.
  5. Capacidad de recuperación ante fallos.
  6. Presupuesto bajo.

4.3. Análisis del Sistema

Implementar un plan de monitorización en un entorno de TI requiere seguir varios pasos clave para asegurar que se cubran todos los aspectos del sistema [35].

La evaluación de necesidades y objetivos para un proyecto que pretende implementar un sistema de monitorización en una pequeña empresa implica definir:

4.3.1. Escenario

Se presenta un sistema distribuido básico diseñado para simular un entorno típico de una pequeña empresa, pero con alta escalabilidad. En este contexto, se realiza una selección de tecnologías, cada una elegida por su relevancia y aplicabilidad en entornos empresariales, así como por su capacidad para demostrar conceptos previamente explorados.

Este sistema se caracteriza por tener una Arquitectura Orientada a Servicios (SOA), integrando múltiples aplicaciones que se prestan servicios entre sí. Además, las aplicaciones implementan diversas arquitecturas distribuidas internamente.

El sistema presentado a modo de ejemplo va a contar con 9 servidores:

- **2 servidores de RedHat Identity Management[36]**: IdM es una solución para la gestión centralizada de identidades y políticas de acceso en sistemas Linux y Unix. Permite administrar usuarios desde un único punto, se trata de una arquitectura P2P donde dos servidores con el mismo rol aseguran alta disponibilidad.
- **2 balanceadores de carga**: HAProxy es un balanceador de carga diseñado para distribuir el tráfico entre servidores, mejorando la disponibilidad y rendimiento de sitios web y aplicaciones. En arquitecturas de N capas,

HAProxy se sitúa en la capa de servicios para equilibrar el acceso a los servidores web.

- **3 Servidores web Apache**: Apache es un servidor web de código abierto ampliamente utilizado para alojar páginas web, correspondiente a la capa de presentación en arquitecturas de N capas.
- **1 servidor de Jira[37]**: Jira es una herramienta para la gestión de proyectos que permite la creación, asignación y seguimiento de tareas dentro de un equipo. Las herramientas de monitorización se integrarán con Jira para crear tareas automáticamente ante cualquier alerta, notificando al administrador de sistemas sobre la necesidad de intervención en algún servidor.
- **1 servidor con Red Hat Ansible Automation Platform[38]**: AAP es una solución integral para la automatización de TI, mejorando la eficiencia y agilidad en la gestión de sistemas y aplicaciones. Se utilizará para automatizar tareas en respuesta a alertas.

En el siguiente paso es necesario entender las operaciones diarias de la empresa, los flujos de trabajo, los sistemas críticos para el negocio y los desafíos específicos relacionados con la tecnología y la seguridad. Esto implica un conocimiento detallado del hardware y software en uso, así como de las necesidades de rendimiento y los requisitos de seguridad.

En el entorno empresarial, los servidores suelen clasificarse en tres tipos según sus roles específicos:

- **Servidores de Pruebas:** Utilizados en el desarrollo de software para validar y probar nuevos códigos o aplicaciones en un entorno similar al de producción, pero separado. Esto permite identificar y corregir errores sin afectar las operaciones normales del negocio.
- **Servidores de Formación:** Destinados a la capacitación de usuarios, permiten practicar y aprender a utilizar nuevas aplicaciones o sistemas sin riesgo de afectar datos críticos o procesos en el servidor de producción. Proporcionan un entorno seguro y controlado, ideal para entrenamientos y familiarización con los sistemas.
- **Servidores de Producción:** Ejecutan las aplicaciones y sistemas vitales para el funcionamiento diario de la empresa. Manejan datos reales y son los que realizan las operaciones continuas del negocio. La estabilidad y seguridad de estos servidores son prioritarias, ya que cualquier problema puede impactar directamente en la eficiencia y rentabilidad de la empresa.

El contrato que garantiza el tiempo de disponibilidad de los servidores en una empresa se conoce como acuerdo de nivel de servicio, o SLA (Service Level Agreement). Este acuerdo especifica los estándares de servicio que la empresa se compromete a cumplir con el cliente. Incluye aspectos como la disponibilidad del servidor, el tiempo de respuesta para el soporte técnico y los procedimientos y tiempos de respuesta en caso de interrupciones del servicio. En el caso de la empresa propuesta, se utilizarán únicamente servidores de formación y producción para simplificar el escenario.

- **Formación:** Debe estar disponible en días laborables de 8 a 17 horas, con

soporte en menos de 4 horas.

- *balancerform1:* Balanceador de carga para la web de formación que simula la infraestructura de producción con un solo nodo para la web.
- *webform1:* Servidor que aloja la página web de formación.

- **Producción:** Requiere disponibilidad las 24 horas del día, 5 días a la semana, con soporte en menos de 2 horas.
  - *ipa1 e ipa2:* Servidores de Identity Management (IdM) que gestionan los usuarios con acceso a los servidores de la empresa.
  - *balancerprod1 y balancerprod2:* Balanceadores de carga para la web de producción, distribuyen el tráfico entre *webprod1* y *webprod2*.
  - *webprod1 y webprod2:* Servidores web que hospedan cada uno una instancia de la web de producción.
  - *jira1:* Aplicación Jira utilizada por los administradores de sistemas para organizar tareas y gestionar alertas.
  - *ansible1:* Plataforma de Automatización Ansible, utilizada para automatizar tareas diarias de la empresa.

El inventario organizado de todos los elementos de la infraestructura, clasificados por prioridad, se presentaría de la siguiente manera:

Nombre	**Ip**	**Tipo**	**Hardware**	**Entorno**	**App**	**Prioridad**
ipa1	192.168.1.12	Servidor	VMWare	Prod	IdM	Media
ipa2	192.168.1.20	Servidor	VMWare	Prod	IdM	Media
jira1	192.168.1.13	Servidor	VMWare	Prod	Jira	Alta
ansible1	172.20.10.4	Servidor	VMWare	Prod	Ansible AP	Alta
balancerprod1	192.168.1.14	Servidor	VMWare	Prod	App Prod	Alta
balancerprod2	192.168.1.15	Servidor	VMWare	Prod	App Prod	Alta
balancerform1	192.168.1.19	Servidor	VMWare	Form	App Form	Baja
webprod1	192.168.1.16	Servidor	VMWare	Prod	App Prod	Alta
webprod2	192.168.1.17	Servidor	VMWare	Prod	App Prod	Alta
webform1	192.168.1.18	Servidor	VMWare	Form	App Form	Baja

4.3.2. Alertas

Es necesario establecer alertas específicas para cada elemento y marca. Estas pueden incluir situaciones como procesadores saturados, ancho de banda limitado o discos llenos. Para cada alarma, se deben definir umbrales y niveles específicos que determinen su activación.

Cada alerta puede tener cualquiera de los siguientes estados:

- **Warning:** Indica que se ha detectado algo inusual o potencialmente problemático en el sistema o servicio monitorizado. No es tan grave como un error, pero sugiere que algo requiere atención antes de que empeore. Por ejemplo, podría ser un uso de recursos más alto de lo normal, pero todavía dentro de un rango seguro.
- **Error (Critical):** Señala una condición crítica que necesita atención inmediata. Significa que algún aspecto del sistema o servicio ha fallado o está funcionando de manera inadecuada. Ejemplos podrían incluir un servicio caído, un disco duro lleno, o un proceso crítico que no funciona.
- **Unknown:** Se utiliza cuando el sistema de monitorización no puede determinar el estado del servicio o sistema. Puede deberse a un error en la herramienta de monitorización, una respuesta inesperada del sistema o una configuración incorrecta. Este estado suele indicar que es necesario revisar la monitorización o el sistema para entender la causa.
- **Ok:** Indica que el sistema funciona correctamente y no se requieren acciones.

Las alertas para los servidores del inventario se pueden dividir en dos tipos:

- **Chequeos genéricos**, aplicables a todos los servidores sin excepción:
  - **Disponibilidad:** Estado de ”Error” cuando el servidor no esté disponible en la red.
  - **Espacio en discos:** Estado de ”Warning” cuando el espacio ocupado supere el 70 % y de ”Error” cuando supere el 90 %.
  - **Carga de CPU:** Estado de ”Warning” cuando la carga supere el 60 % y de ”Error” cuando supere el 85 %.
  - **Servicios activos:** sssd (necesario para la autenticación de usuarios de IdM), snmp y ssh. Estado de ”Error” cuando un servicio no esté activo.
- **Chequeos específicos de cada servidor:**
  - *ipa1 e ipa2*:
    - **Aplicación web:** Estado de ”Error” cuando no esté disponible.
    - **Servicios activos:** ipa, sssd, snmp, dirsv y ldap. Estado de ”Error” cuando un servicio no esté activo.
  - *jira1:*
    - **Aplicación web:** Estado de ”Error” cuando no esté disponible.

- ◇ **Servicios activos:** jira y mariadb. Estado de "Error" cuando un servicio no esté activo.

- ○ *ansible1:*
  - ◇ **Aplicación web:** Estado de "Error" cuando no esté disponible.
  - ◇ **Servicios activos:** ansible-automation-platform. Estado de "Error" cuando el servicio no esté activo.
- ○ *balancerprod1, balancerprod2, balancerform1:*
  - ◇ **Servicio activo:** haproxy. Estado de "Error" cuando no esté activo.
- ○ *webprod1, webprod2, webform1:*
  - ◇ **Servicio activo:** httpd. Estado de "Error" cuando no esté activo.

4.3.3. Canales de Comunicación

Es necesario establecer métodos de comunicación claramente definidos para la gestión eficaz de las alertas. Estos métodos incluyen SMS, correo electrónico, WhatsApp o notificaciones push. Además, es necesario implementar un proceso detallado para la atención y manejo adecuado de estas alarmas.

En cuanto a la gestión de las alertas, se distinguen dos categorías principales. Durante el horario de oficina, cuando se detecte una alerta, se generará automáticamente un ticket en Jira y se enviará un correo electrónico de carácter informativo. Para las alertas que ocurran fuera del horario de oficina solo serán notificadas al personal de guardia las de producción. Estas alertas se enviarán a su teléfono móvil usando Jira, para asegurar una respuesta rápida y efectiva ante cualquier incidencia.

4.4. Diseño de un Prototipo

4.4.1. Análisis de Herramientas Disponibles:

En esta fase, se ha realizado una evaluación exhaustiva de las herramientas de monitorización disponibles en el mercado para seleccionar la qué más se ajuste al presupuesto y que cumpla con los requisitos identificados en las etapas previas.

Entre las opciones consideradas en el capítulo 3, se ha elegido Centreon debido a su capacidad para cumplir con todos los requisitos del proyecto. Centreon es una herramienta altamente eficiente y versátil para la monitorización de sistemas, reconocida por su interfaz intuitiva y personalizable. Destaca por su capacidad de adaptación a una amplia gama de dispositivos y protocolos, facilitando la monitorización centralizada de infraestructuras de TI de diversos tamaños y complejidades.

Esta herramienta ofrece funcionalidades avanzadas como alertas proactivas, generación de informes detallados y dashboards personalizables, lo que permite identificar rápidamente problemas y mejorar la toma de decisiones.

La escalabilidad de Centreon y su capacidad de integración con otras herramientas de gestión de TI, como Jira, junto con su soporte para la automatización de tareas, la convierten en una opción robusta para entornos dinámicos y en constante evolución.

La activa comunidad de usuarios de Centreon y las opciones de soporte técnico refuerzan su fiabilidad, haciéndola una herramienta valiosa para garantizar la seguridad y el cumplimiento en la gestión de sistemas de TI. Además, su modelo de precios, que ofrece una versión gratuita con la opción de contratar soporte adicional según las necesidades, la hace aún más atractiva.

4.4.2. Protocolo de Resolución de Incidencias

Una vez seleccionada la herramienta de monitorización, se debe definir un procedimiento detallado para la resolución de incidentes. Integrar un sistema de monitorización junto con un software de gestión de incidencias es necesario para gestionar los problemas de forma eficaz. [39]

4.4.2.1. Grupos Responsables

La estructura de soporte se organizará en dos equipos principales.

- **Equipo de Sistemas:** Responsable de problemas de hardware, sistema operativo, y aplicaciones internas críticas como Jira, Haproxy e IdM (Identity Management). Su función es garantizar la estabilidad y eficiencia de los sistemas.
- **Equipo de Desarrollo**: Enfocado en problemas de la aplicación web. Su tarea es resolver problemas de la aplicación, optimizar su rendimiento y asegurar su integración con la infraestructura.

4.4.2.2. Sistema de Comunicación

Se implementará un sistema de comunicación utilizando Jira. Cuando se genere una alerta, se creará automáticamente un ticket correspondiente en Jira para su revisión. Algunas alertas también desencadenarán una tarea para intentar resolver el incidente de forma autónoma. Por ejemplo, si se detecta que un servicio web en un servidor ha estado inactivo durante más de 5 minutos, se intentará un reinicio automático. Si el reinicio tiene éxito, el ticket no se creará.

En caso de alertas en producción fuera del horario laboral, se contactará al responsable a través de su teléfono móvil a través de Jira. Cada ticket se crea con un título significativo y una descripción detallada. Posteriormente, se asigna al individuo o equipo correspondiente y se etiqueta para su seguimiento. Durante el proceso, el responsable actualiza regularmente el estado y añade comentarios para reflejar el progreso realizado. Antes de cerrar el ticket, se documenta toda la información relevante, concluyendo con un resumen o comentarios finales sobre la solución adoptada.

Capítulo 5

Escenario Propuesto

Este capítulo ofrece una guía detallada sobre la configuración, despliegue y mantenimiento de sistemas distribuidos en entornos empresariales. Incluyendo su instalación, monitorización, automatización y gestión de vulnerabilidades.

Se presenta un sistema distribuido básico, diseñado para simular un entorno de una pequeña empresa, pero con alta escalabilidad.

5.1. Distribución de Linux

La elección de una distribución de Linux depende de los requisitos específicos, las preferencias y el presupuesto de la empresa: [40]

- **Red Hat Enterprise Linux (RHEL)**: Está pensado para empresas que buscan la máxima compatibilidad con software empresarial y que están dispuestas a pagar una tarifa de licencia adicional. Es conocido por su estabilidad y capacidad para manejar cargas pesadas, siendo una buena opción para empresas que ejecutan software que requiere RHEL específicamente. Si bien puede ser costoso, ofrece estabilidad y soporte de nivel empresarial, lo que lo hace adecuado para empresas más grandes y entornos críticos.
- **CentOS**: Es una versión altamente estable de Linux que ofrece fiabilidad a nivel empresarial, pero sin el costo asociado con RHEL. Viene con la misma base de software estable que RHEL y es compatible con actualizaciones de seguridad durante un mínimo de 5 años. Sin embargo, no ofrece el mismo nivel de soporte.
- **Fedora**: Se centra en incluir software de vanguardia y se actualiza continuamente. Es recomendado para administradores de Linux avanzados, pero puede no ser adecuado para entornos de servidores estables o para usuarios menos experimentados.
- **Debian**: Es una distribución que se actualiza de manera continua y está recomendada para usuarios con experiencia en Linux. No es la opción ideal para principiantes debido a que su naturaleza en constante cambio puede introducir problemas con las nuevas actualizaciones.

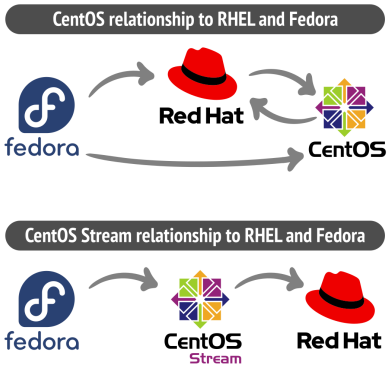


Figura 5.1: Desarrollo diferentes distribuciones.

Para este prototipo, se ha seleccionado CentOS debido a que su licencia es gratuita. No obstante, en un entorno empresarial real, se recomienda preferiblemente utilizar Red Hat Enterprise Linux (RHEL) debido a su fiabilidad y soporte robusto [41].

5.2. Preparación del Escenario

El sistema presentado a modo de ejemplo va a contar con 9 servidores:

- 2 nodos de RedHat Identity Management (IdM).
- 2 balanceadores de carga HAProxy.
- 3 servidores web Apache.
- 1 servidor de Jira.
- 1 servidor con Red Hat Ansible Automation Platform

5.2.1. Red Hat Identity Management (IdM)

Red Hat Identity Management (IdM)[42] es una solución integral de Red Hat diseñada para la gestión centralizada de identidades, autenticación y políticas de acceso en entornos empresariales de TI, asegurando una administración segura y eficiente de usuarios, grupos y accesos en infraestructuras de TI.

Sin IdM, cada servidor se gestiona de forma independiente, lo que implica almacenar contraseñas en máquinas locales. Los administradores deben manejar usuarios y políticas de autenticación por separado en cada máquina, y los usuarios deben recordar múltiples credenciales para acceder a diferentes servicios.

El dominio de Gestión de Identidad (IdM) agrupa máquinas con configuraciones, políticas y almacenamiento de identidades compartidos, permitiendo la comunicación y operación coordinada entre ellas. Este dominio abarca distintos tipos de máquinas, cada una con roles específicos dentro del sistema de gestión de identidad:

- **Servidores IdM**: Actúan como controladores de dominio y sirven como repositorios centrales para la información de identidad y políticas. Estos ser-

vidores proporcionan servicios utilizados por los miembros del dominio, incluyendo una interfaz web y utilidades de línea de comandos de IdM. Para garantizar redundancia y equilibrio de carga, los datos y la configuración pueden replicarse entre diferentes servidores IdM.

- **Clientes de IdM**: Es un software que administra las identidades digitales y regula el acceso de los usuarios a los recursos de un host. Los hosts, en los que se instalan los Clientes de IdM, están configurados para operar dentro del dominio de IdM. Esto implica interactuar con los servidores de IdM para acceder a recursos del dominio, por lo que actúan como intermediarios que facilitan la conexión entre los usuarios y los recursos dentro del entorno de IdM.

Los servidores IdM ofrecen varias funcionalidades:

- **Gestión Unificada de Identidades**: IdM simplifica la gestión de identidades al ofrecer una plataforma única para gestionar cuentas de usuario y permisos. Esto reduce la complejidad y los costes asociados con la administración de múltiples sistemas.
- **Autenticación y Autorización**: Utilizan Kerberos para una autenticación segura, control de acceso basado en roles y políticas de acceso detalladas.
- **Gestión de Certificados (PKI)**: Proporcionan servicios de infraestructura de clave pública (PKI) para la gestión de certificados digitales.
- **Interfaz de Usuario y Herramientas de Administración**: Incluyen una interfaz web intuitiva y herramientas de línea de comandos para la automatización y gestión eficiente del entorno IdM.
- **Autenticación Multifactor (MFA)**: Soportan la autenticación multifactor, mejorando la seguridad mediante la verificación de múltiples formas de identidad más allá de las contraseñas tradicionales.

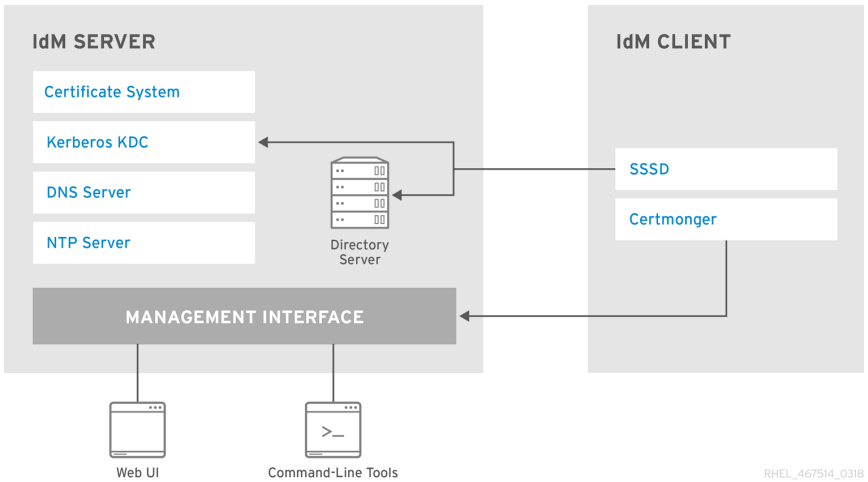


Figura 5.2: Arquitectura servidor y cliente IdM.

5.2.2. Jira

Jira[37] es una herramienta de software utilizada para la gestión de proyectos, seguimiento de errores y organización de tareas. Funciona como una plataforma integral de planificación y ejecución que optimiza el flujo de trabajo en equipos.

Diseñado tanto para administradores como para usuarios finales, Jira facilita la distribución de tareas, el seguimiento del progreso y la resolución ágil de problemas mediante la creación y asignación de tickets individuales.

También, Jira ofrece soluciones de documentación de incidencias, facilitando la integración rápida de nuevos miembros al equipo.

Además del producto principal, Jira se puede ampliar mediante una gran variedad de plugins y soluciones adicionales que ofrece Atlassian, permitiendo una personalización flexible y una optimización eficaz del flujo de trabajo para uno o varios proyectos.

5.2.3. Ansible Automation Platform

Ansible es una herramienta de código abierto proporcionada por RedHat Enterprise Linux (RHEL), que facilita la administración de configuraciones, la orquestación de servicios, la automatización de tareas y la implementación de aplicaciones en múltiples máquinas. [43]

Esta herramienta es de ayuda para los administradores de sistemas que buscan minimizar tareas recurrentes, simplificar la implementación y lograr una automatización eficiente.

Una de las ventajas destacadas de Ansible es su arquitectura sin agentes; no requiere instalación de ninun software en todas las máquinas. Solo necesita estar instalado en la máquina principal, desde donde se conecta a las demás mediante SSH.

A continuación, se detallan los elementos más importantes de Ansible:[44]

- **Nodo de control**: Es la máquina principal donde está instalada Ansible. Esta máquina se encarga de conectarse a los demás nodos mediante SSH para ejecutar tareas. Desde aquí, se pueden ejecutar comandos y playbooks.
- **Nodo gestionado**: Cualquier máquina que se administra con Ansible. Estos nodos también se conocen como hosts y no necesitan tener Ansible instalado.
- **Inventario**: Un fichero que recoge todas las máquinas a gestionar. Permite crear grupos de máquinas y definir variables globales, facilitando la administración. Contiene información específica sobre cada host, como su dirección IP.
- **Módulos**: Librerías que contienen funcionalidades para realizar diversas tareas, como la gestión de paquetes o la ejecución de comandos específicos. Ansible viene con módulos incorporados, pero también se pueden escribir módulos personalizados.

- **Playbook**: Define el estado deseado de un sistema a través de una lista ordenada de tareas a realizar secuencialmente. Se almacena en un archivo con formato YAML.

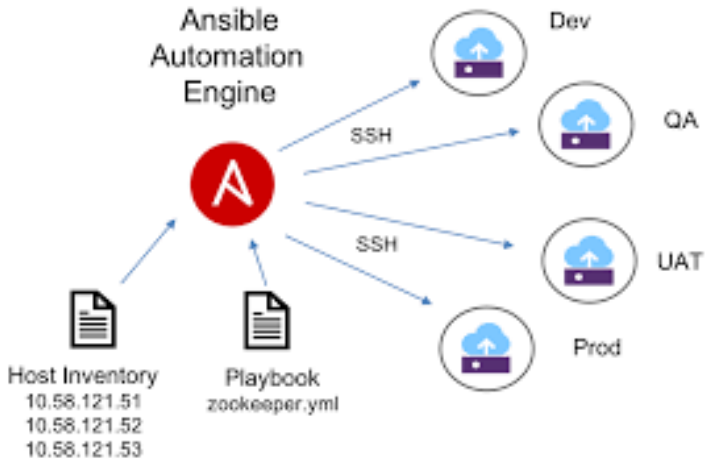


Figura 5.3: Esquema del funcionamiento de Ansible

La principal ventaja de Ansible reside en su simplicidad y facilidad de configuración y uso. No exige una gran curva de aprendizaje para los desarrolladores.

Ansible Automation Platform[45] es una solución empresarial de automatización de TI desarrollada por Red Hat, basada en Ansible. A diferencia de Ansible, que en su forma básica es una herramienta de línea de comandos, AAP la amplía a una solución con una interfaz gráfica y mejores capacidades de gestión, control, seguridad y soporte. Esto la hace más adecuada para entornos empresariales donde factores como escalabilidad, control y cumplimiento son esenciales.

Empresas y organizaciones utilizan AAP para automatizar procesos de TI repetitivos, reducir errores, aumentar la eficiencia y acelerar la entrega de aplicaciones y servicios. Es especialmente útil en entornos de TI híbridos y complejos, donde la coherencia y la gestión eficiente son críticas.

5.2.4. Servidor Web Apache

Apache[46] es un servidor web de código abierto que se encarga de almacenar, procesar y entregar páginas web a los usuarios. Su distribución bajo una licencia de código abierto significa que es gratuito, adaptable, personalizable y reutilizable.

Como servidor web, Apache gestiona las peticiones de los clientes, respondiendo con el contenido correspondiente de las páginas web y convierte datos escritos en diversos lenguajes de programación web a archivos HTML estáticos, que luego se transmiten a los navegadores de los usuarios.

Aunque comúnmente se le llama "servidor Apache", es importante destacar que Apache es en realidad un software que opera en conjunto con un servidor físico. Cuando un usuario accede a una página web, su navegador envía una petición al servidor mediante el protocolo HTTP. Apache entonces genera una respuesta

que incluye todos los datos de la página solicitada, como texto, imágenes, vídeos, entre otros.

Además de asegurar una entrega completa y rápida de estos datos, Apache también se enfoca en la seguridad de la transmisión.

5.2.5. Balanceador de Carga Haproxy

Un balanceador de carga es un dispositivo o software diseñado para distribuir el tráfico de red o las solicitudes entre varios servidores. Su objetivo es aumentar la capacidad, velocidad y disponibilidad del sistema, optimizando la utilización de recursos y permitiendo el mantenimiento sin interrupciones. Este tipo de solución es ampliamente utilizado en sitios web con alto tráfico, aplicaciones de Internet y entornos de computación en la nube.

HAProxy[47], un software de código abierto, se emplea para el balanceo de carga y actúa como proxy para aplicaciones TCP y HTTP. Distribuye el tráfico de red entre varios servidores para mejorar la disponibilidad y el rendimiento de sitios web y aplicaciones. Además, ofrece funciones como el control de salud de los servidores y seguridad con terminación SSL. Es particularmente popular en entornos que requieren alta disponibilidad y rendimiento.

Una característica distintiva de HAProxy es su habilidad para realizar chequeos regulares del estado de los servidores backend, garantizando su disponibilidad y correcto funcionamiento. Esto es esencial para mantener la continuidad y la fiabilidad del servicio, especialmente en entornos donde el tiempo de inactividad puede resultar crítico.

HAProxy no solo optimiza el tráfico mediante técnicas como la compresión y la caché, sino que también proporciona herramientas para reescribir y modificar solicitudes, mejorando la interoperabilidad y la eficiencia. En términos de seguridad, proporciona protecciones robustas contra una variedad de ataques de red, incluidos los ataques de denegación de servicio, lo cual es crucial para proteger infraestructuras críticas.

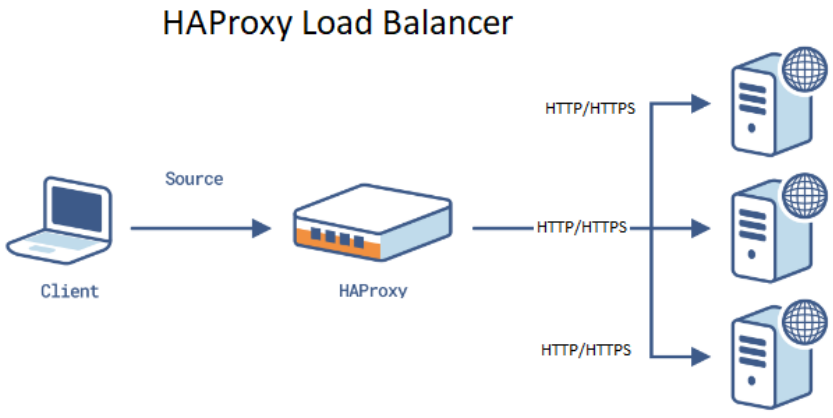


Figura 5.4: Funcionamiento haproxy

5.3. Centreon

5.3.1. Arquitectura

Una vez decidida la herramienta de monitorización, el siguiente paso es planificar su despliegue. Según las necesidades de la infraestructura, la plataforma puede constar de varios componentes: un servidor central, uno o varios servidores remotos y uno o varios pollers. Si se monitoriza un número reducido de hosts y servicios, un servidor central puede ser adecuado. Sin embargo, para gestionar un gran volumen de hosts y servicios, es necesario distribuir la carga entre varios servidores.

Además, hay que tener en cuenta las siguientes consideraciones: [48]

- El número de hosts y servicios a monitorizar, así como la cantidad de métricas por servicio.
- La necesidad de utilizar pollers o servidores remotos para separar los recursos monito rizados según criterios geográficos o lógicos. Por ejemplo, es más práctico (y seguro) ubicar un servidor remoto en la red DMZ si se debe monitorizar esa área.
- Un servidor central debe supervisar solo un pequeño número de hosts y servicios, ya que su CPU principal debe manejar los datos enviados por servidores remotos y pollers (lo mismo aplica para los servidores remotos). Cuantos más hosts y servicios se supervisen en el servidor central, mayor será el riesgo de ralentización de la interfaz.
- El servidor central debe monitorizar todos los servidores remotos y pollers.
- Se debe usar un servidor remoto en lugar de un poller si se necesita ver los datos en un sitio diferente al del servidor central.

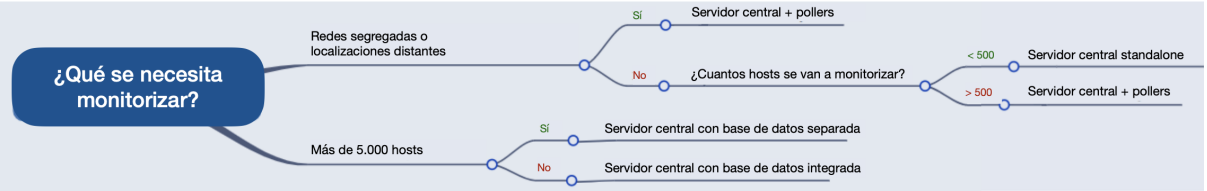


Figura 5.5: Diseño arquitectura Centreon.

Comprender los requisitos específicos, como el número de servicios por anfitrión, la frecuencia de recolección de datos y el período de retención de datos, es necesario para diseñar una solución que satisfaga las necesidades de monitorización de la empresa. Para este ejemplo en particular, los requisitos son los siguientes:

- Un promedio de 8 servicios por anfitrión.
- Recolección de datos cada 1 o 2 minutos.
- Período de retención de datos de 12 meses.
- Cada servicio tiene un promedio de 8 métricas.

Para este caso, un único servidor central autónomo con 4 CPUs y 4GB de RAM es adecuado. Sin embargo, esta arquitectura es escalable, lo que significa que se pueden añadir pollers o servidores remotos según crezca la infraestructura.

5.3.2. Componentes

- **Servidor web Apache** para la interfaz gráfica de Centreon.
- **Base de datos** de MariaDB para almacenar los parámetros de configuración de Centreon, así como datos de monitorización y rendimiento.
- **Centreon Engine** recolecta datos de monitorizción y mantiene registros del estado de los hosts y servicios.
- **Centreon Broker SQL** recibe los datos recolectados por el Centreon Engine a través del módulo cbmod. Este último actúa como intermediario en la transmisión de datos entre el motor de Centreon y la base de datos SQL.

  CBMod (Centreon Broker Module) gestiona la transferencia de datos entre Centreon Engine y las bases de datos de MariaDB. Luego, reenvía esta información a Centreon Broker RRD para su procesamiento y análisis.
- **Centreon Broker RRD** crea y actualiza archivos RRD para visualizar gráficos de rendimiento. Estos archivos, conocidos como RRD (Round Robin Database), almacenan series temporales de datos, como métricas de red, tráfico de Internet y uso de CPU, permitiendo su análisis.

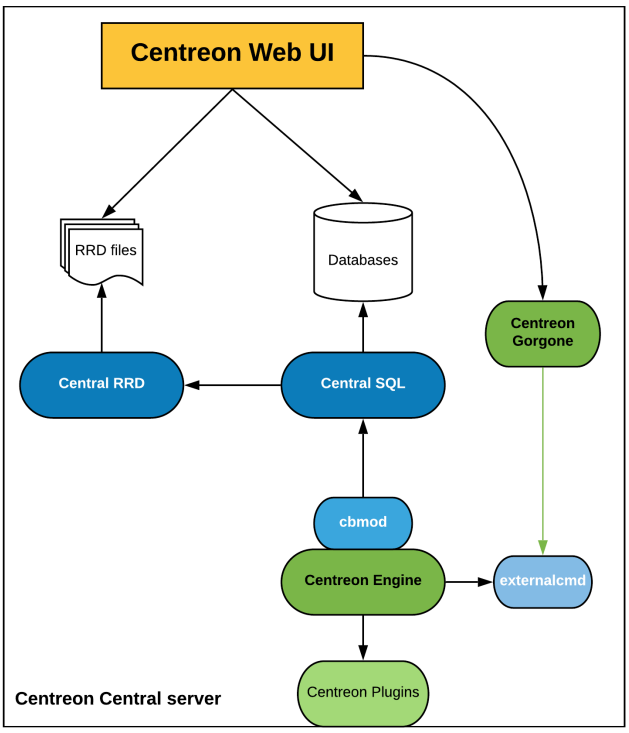


Figura 5.6: Arquitectura del servidor standalone.

5.4. Integración de Herramientas

La integración es el proceso de conectar diferentes aplicaciones y sistemas para que funcionen juntos de manera eficiente. Esto permite automatizar procesos, mejorar la eficiencia operativa, facilitar la toma de decisiones y aumentar la productividad, asegurando un flujo continuo de datos entre las herramientas utilizadas en una organización.

Para esta etapa, se integrarán las alertas de Centreon, los tickets de Jira y los trabajos de Ansible, combinando estas tres herramientas para crear un sistema de mantenimiento optimizado y fácil de gestionar. El flujo propuesto es el siguiente:

1. **Registro de un host en IdM:** Se registra un host en IdM y se asigna al grupo correspondiente (formación o producción). Este proceso requiere intervención manual ya que implica seleccionar el grupo apropiado y acceder al host con un usuario local que tenga permisos de root.
2. **Automatización del Inventario en AAP:** Se ejecuta diariamente una tarea que verifica si hay nuevos hosts dados de alta en IdM. Si se encuentra un nuevo host, se añade automáticamente al inventario de AAP marcando si es para formación o producción.
3. **Configuración automática de host:** Una vez que un host es añadido a AAP, se ejecuta un playbook para realizar una serie de configuraciones básicas y así integrarlo completamente en la infraestructura.
4. **Registro de hosts en Centreon:** Gracias a la configuración realizada en el paso anterior, Centreon puede descubrir el nuevo host en la red, asignarlo a un grupo de monitorización y supervisar sus servicios.
5. **Automatización de gestión de incidentes:** En algunos casos, cuando Centreon detecte una alarma durante la monitorización, podrá tomar medidas para solucionar el problema de forma autónoma. Si esto no fuese posible, solo se notificará del incidente mediante un Jira.
6. **Creación de tickets en Jira:** Cuando se detecta una alarma en la monitorización de Centreon, se genera automáticamente un ticket en Jira en el proyecto correspondiente para su revisión.

5.4.1. Automatización del Inventario en Ansible Automation Platform

Un inventario de hosts es una lista detallada de todos los dispositivos conectados a una red, incluyendo su nombre, dirección IP, tipo, sistema operativo y estado. Este inventario es necesario para gestionar y monitorizar una infraestructura.

En este escenario el inventario de Ansible Automation Platform (AAP) se actualizará automáticamente y será utilizado por cada una de las tres herramientas mencionadas, asegurando que todas trabajen con la información más reciente y precisa sobre los recursos.

Los pasos para automatizar la generación del inventario son los siguientes:

1. Registrar todos los hosts en IdM y añadirlos al grupo correspondiente.

```
[root@jira1 ~]# ipa-client-install --domain=mydomain.local --server=
    ipa1.mydomain.local --mkhomedir
```

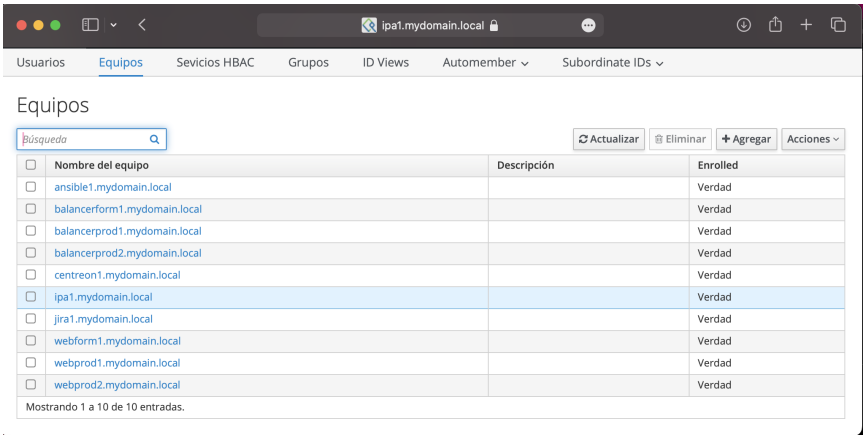


Figura 5.7: Hosts registrados IdM.

2. Crear el usuario *ansible* en IdM. Este usuario tendrá permisos de root en todos los servidores y su acceso estará habilitado desde *ansible1* mediante una clave pública. Para añadir la clave pública, ejecutar el siguiente comando en el servidor IPA (*ipa1*):

```
[root@ipa1 ~]# ipa user-mod ansible --sshpubkey="CLAVEPUBLICA"
```

3. Registrar el usuario *ansible* en AAP para que pueda ejecutar los playbooks. Para ello, acceder a "Credenciales → Nueva Credencial" y añadir la clave pública y privada del usuario *ansible1*.

4. Crear un proyecto en AAP donde se almacenarán los playbooks necesarios para automatizar la generación del inventario. Para ello, hay que acceder a "Proyectos → Crear nuevo" y elegir entre un proyecto local o de Git.

5. Automatizar la gestión del inventario en AAP, usando un playbook que se autentica en el servidor de IdM (*ipa1*), consulta la lista actual de servidores registrados y la compara con el inventario de AAP. Si encuentra un servidor que no esté registrado en AAP, lo añadirá automáticamente y, dependiendo del grupo de host al que pertenece, lo marcará como formación o producción. El playbook necesario se puede encontrar en el Anexo 2: *check_inventory.yml*.

6. Antes de ejecutar un playbook, es necesario crear una plantilla de trabajo en AAP. Una plantilla de trabajo es una configuración predefinida que automatiza tareas específicas, como la ejecución de secuencias de playbooks con variables personalizadas, credenciales específicas y el inventario de hosts donde se ejecutará. Para crear una plantilla, hay que acceder al menú "Plantillas → Agregar Plantilla de Trabajo" y seleccionar el playbook deseado, especificando los servidores y las credenciales a utilizar, en este caso, las del usuario *ansible1*.

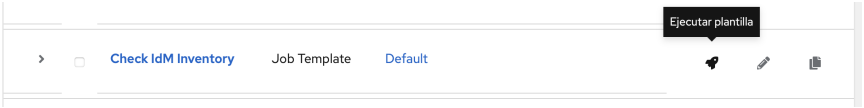


Figura 5.8: Plantilla de trabajo.

7. Para ejecutar la plantilla de trabajo diariamente, se debe crear una programación en AAP. Una programación es una configuración que permite ejecutar tareas automatizadas en momentos específicos o a intervalos regulares, asegurando su ejecución sin intervención manual. Para programar la ejecución, acceder a la plantilla de trabajo deseada y seleccionar "Programaciones → Agregar Programación", indicando la frecuencia de ejecución deseada.

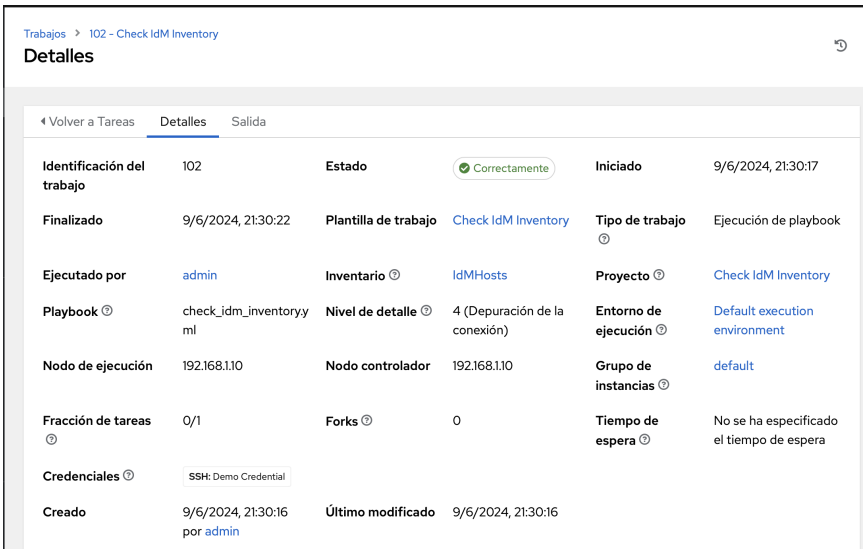


Figura 5.9: Ejecución de la programación.

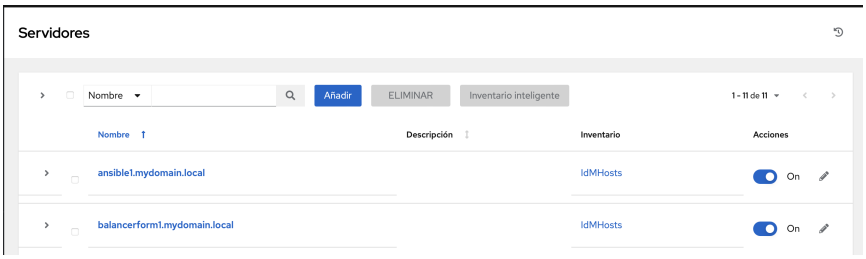


Figura 5.10: Servidores añadidos al inventario de AAP.

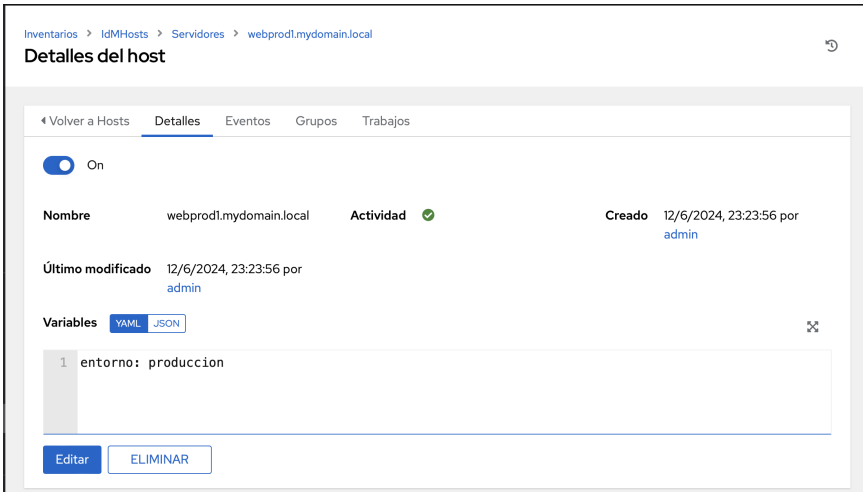


Figura 5.11: Variable con el entorno declarada.

5.4.2. Configuración automática de host:

Una vez que se añade un host a Ansible Automation Platform (AAP), se ejecuta por primera vez el playbook *manage_host.yml*, el cual realiza configuraciones básicas en el servidor. Este playbook tiene diversas aplicaciones, como securizar el sistema, instalar agentes o realizar configuraciones necesarias para la homogeneización de la infraestructura. En este escenario, este paso es necesario para configurar el agente de SNMP, y así permitir la monitorización automática de Centreon.

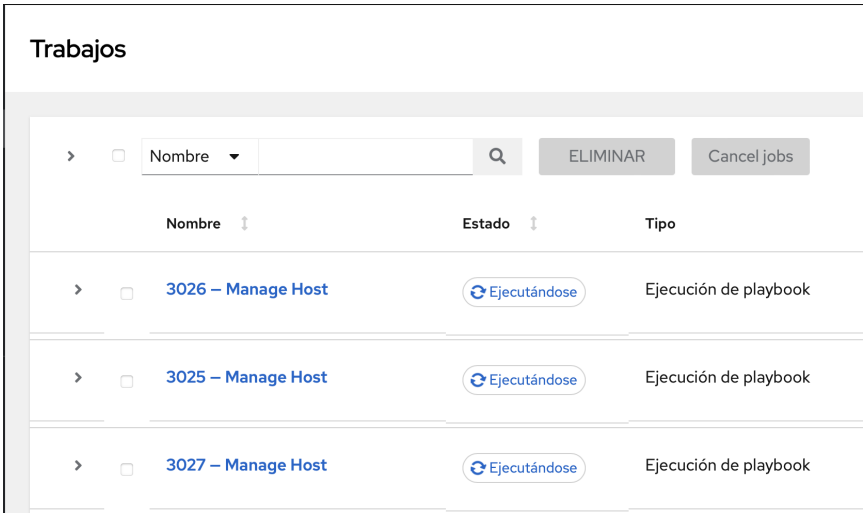


Figura 5.12: Tareas de configurar nuevos servidores lanzadas por *check_inventory.yml*.

SNMP (Simple Network Management Protocol) es un estándar de Internet utilizado para la gestión y supervisión de dispositivos de red. Permite a los administradores recopilar información y monitorizar el rendimiento de dispositivos como routers, switches, impresoras y servidores. SNMP opera a través de agentes instalados en los dispositivos que recopilan datos y responden a solicitudes

de gestión, facilitando la administración centralizada y la detección de problemas en la red.

En el playbook *manage_host.yml*, se realiza lo siguiente:

1. Se descarga el paquete SNMP.
2. Se configura el agente SNMP.
3. Se declara la variable *sysDescr* para indicar si es formación o producción.
4. Se establece una clave para que Centreon pueda monitorizar el servidor.
5. Se habilita el servicio *snmpd* para que inicie automáticamente con el servidor.

Esta configuración asegura que el sistema esté preparado para ser monitorizado automáticamente desde Centreon. Pero además, este PlayBook se podría utilizar para ejecutar diversas tareas de configuración en el servidor. Como por ejemplo:

- Implementar medidas de seguridad como deshabilitar servicios innecesarios, configurar Selinux y ajustar permisos.
- Instalar y configurar software nuevo.
- Reiniciar el servidor.

Implementar estas tareas mediante un playbook asegura configuraciones consistentes y repetibles, mejora la eficiencia operativa y minimiza el riesgo de errores humanos.

5.4.3. Registro de Hosts en Centreon

El registro de un host en Centreon puede realizarse de dos formas: cada servidpr manualmente, o mediante el módulo de Autodescubrimiento[49]. Esta extensión automatiza la detección de dispositivos y servicios en la red, integrándolos automáticamente en el sistema de monitorización. Además, el módulo identifica recursos no disponibles y los deshabilita automáticamente, mejorando la eficiencia, ahorrando tiempo y reduciendo posibles errores manuales. Para llevar a cabo esta tarea, el módulo utiliza Nmap o SNMP.

Nmap es un escáner de puertos que detecta puertos abiertos, identifica servicios y obtiene información sobre sistemas operativos. Por otro lado, SNMP funciona mediante un agente instalado en cada dispositivo, que recopila datos sobre su estado y los pone a disposición de un gestor central. Centreon utiliza Nmap para verificar la disponibilidad de hosts y servicios públicos sin autenticación, como una página web, y SNMP para la monitorización y control del estado del servidor.

Los pasos para poder registrar los host en Centreon automáticamente son:

1. **Crear una plantilla:** Una plantilla de host es una configuración predefinida que agrupa parámetros y servicios comunes para simplificar y estandarizar la monitorización de dispositivos en una red. Al usar plantillas, se reduce el esfuerzo manual y se asegura la consistencia en la configuración de hosts similares. Para crear una plantilla, hay que ir al menú "Configuración → Host → Plantilla" y rellenar los siguientes campos:

- **Parámetros Básicos del Host**: Incluyen el nombre de la plantilla, la dirección de red o la comunidad SNMP.
- **Comandos de Check**: Especifica los scripts o programas que se utilizan para comprobar el estado de los dispositivos. Estos pueden ser comandos personalizados o predefinidos en Centreon.
- **Intervalos de Monitorización**: Define la frecuencia con la que se realiza la monitorización del dispositivo (por ejemplo, cada 2 minutos) y establece la política de reintentos en caso de fallo.
- **Parámetros de Notificación**: Define quién debe ser notificado en caso de fallos o advertencias, y cómo se deben enviar las notificaciones (por ejemplo, correo electrónico, SMS).
- **Servicios Asociados**: Incluye los servicios específicos que deben ser monitorizados en el host. Por ejemplo, en un servidor web se podrían monitorizar servicios como ssh, http, etc.

Para este escenario, se han creado dos plantillas diferentes: una para entornos de formación *Linux_FORM* y otra para entornos de producción *Linux_PROD*. Se asignará un servidor a una de estas plantillas en función del parámetro SNMP *sysDescr* configurado en el apartado anterior.

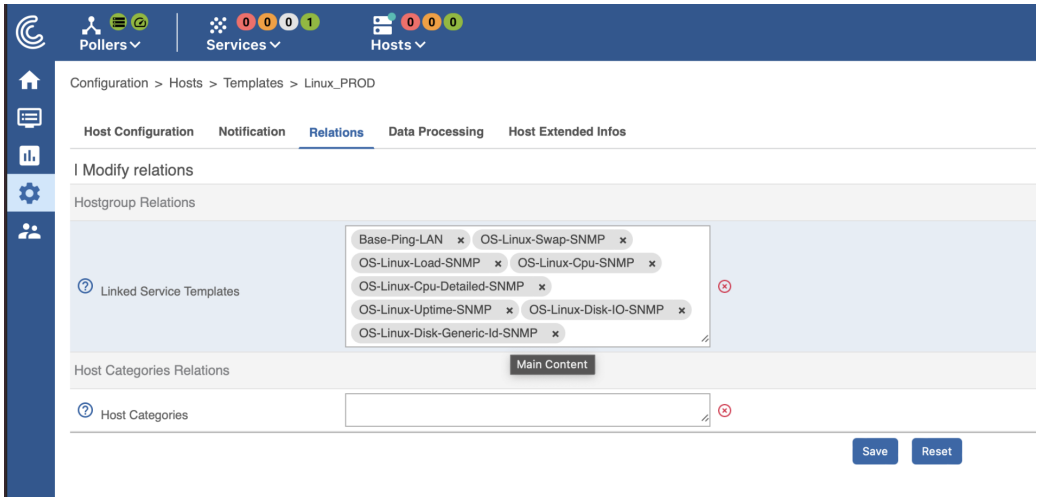


Figura 5.13: Definición plantilla *Linux_PROD*.

La principal diferencia entre ambas es que la plantilla de producción tiene un tiempo de notificación 24/7, mientras que la de formación solo notifica durante el horario laboral. Además, los chequeos en producción se realizarán cada minuto, mientras que en formación se harán cada cinco minutos.

2. **Configurar el módulo de Autodiscovery:** Automatiza la detección y configuración de nuevos dispositivos dentro de una red. Utilizando métodos como el escaneo de red o la importación de listas de dispositivos, Autodiscovery identifica automáticamente dispositivos activos y disponibles para monitorización. Una vez detectados, el módulo puede aplicar plantillas predefinidas para establecer parámetros estándar de monitorización, como direcciones IP, comunidades SNMP y servicios específicos a monitorizar. Es-

te módulo es uno de los pocos que requiere licencia para ser utilizado. La licencia es gratuita para sistemas con hasta 100 servidores [50].

Para crear una tarea de Autodiscovery, primero hay que ir al menú "Configuración → Host → Discovery" y crear una nueva tarea basada en el proveedor de SNMP:

- **Método de Detección**: Especifica cómo se detectarán los dispositivos (por ejemplo, NMAP, SNMP, importación de archivos CSV). En este caso, dado que el agente de SNMP está configurado en todos los servidores, se utilizará este método.
- **Filtros de Detección**: Define criterios para filtrar qué dispositivos serán detectados (por ejemplo, por dirección IP, nombre de host, tipo de dispositivo). Aquí se incluye la subred de destino y el tiempo máximo de ejecución del comando (generalmente 120 segundos para una red con una máscara /24).
- **Plantilla de Host**: Asigna una plantilla predefinida que se aplicará automáticamente a los dispositivos detectados para configurar parámetros estándar de monitorización. Se revisará el parámetro SNMP *sysDescr* para determinar si el dispositivo pertenece al entorno de formación o producción y se asignará la plantilla correspondiente.
- **Parámetros de Conexión**: Incluye configuraciones como la comunidad SNMP, si es aplicable, para acceder a los dispositivos y realizar la monitorización.
- **Intervalos de Chequeo**: Especifica con qué frecuencia se realizarán los chequeos de los nuevos dispositivos una vez que sean detectados.

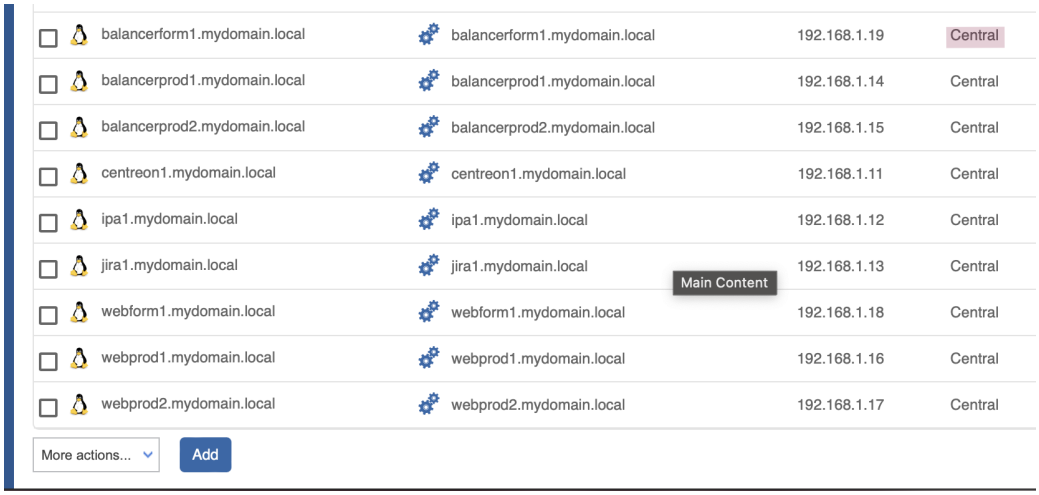


Figura 5.14: Host encontrados.

5.4.4. Automatización de Gestión de Incidentes

Cuando se detecta una alarma en la monitorización, Centreon ofrece la capacidad de ejecutar comandos como parte de la respuesta a una alarma. Esto

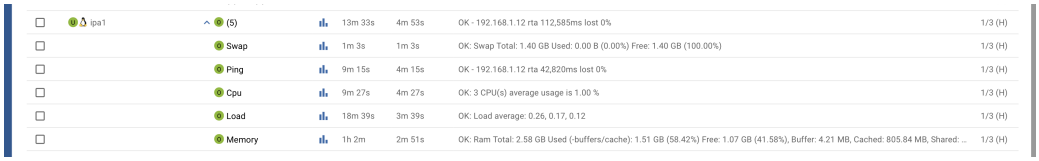


Figura 5.15: Host monitorizado.

significa que, en caso de que un dispositivo o servicio supervisado presente algún problema o estado no deseado, Centreon puede tomar medidas automáticas o permitir que los usuarios realicen acciones específicas para abordar la situación.

Por ejemplo, cuando se produce una alarma debido a un servicio caído, Centreon puede estar configurado para ejecutar automáticamente un script que lo reinicie. O bien, puede permitir a los administradores ejecutar manualmente comandos para reiniciar servicios, limpiar archivos temporales, o cualquier otra acción necesaria para resolver el problema.

Para ejecutar comandos de manera rápida dentro del servidor desde Centreon, es posible configurar un usuario con permisos SSH y capacidad para convertirse en root en los hosts. Esto permite realizar tareas sencillas y administrar comandos de shell directamente desde Centreon, lo cual es conveniente para la gestión operativa de los servidores.

Sin embargo, esta configuración conlleva riesgos importantes en términos de seguridad. Otorgar acceso root a través de SSH desde Centreon implica que cualquier compromiso de la cuenta Centreon podría poner en peligro la seguridad de todos los servidores, ya que un atacante podría potencialmente ejecutar comandos maliciosos con privilegios de root.

Además, la auditoría y la trazabilidad de las acciones ejecutadas son difíciles de gestionar. Es complicado mantener un registro detallado de quién realizó qué acciones y cuándo, lo cual es necesario para la monitorización y la seguridad del sistema.

En este escenario, Centreon no se conectará directamente a los servidores, sino que lanzará un playbook en AAP con destino al servidor afectado. Esta decisión se tomó debido a que es más cómodo realizar automatizaciones con Ansible y también permite consultar los logs de las ejecuciones. Además, evita la necesidad de otorgar acceso como root a más usuarios dentro de los servidores, mitigando así riesgos potenciales para la seguridad.

Los pasos a seguir para poder ejecutar un playbook desde Centreon son:

1. **Autorizar las conexiones de Centreon a la API de AAP**: Para ello, en AAP se debe crear una aplicación en "Administrar → Aplicaciones". Aquí se especifica el nombre de la aplicación desde la cual se realizará la conexión, en este caso Centreon, su URI y el método de autenticación, que en este caso es mediante código de autenticación. Esto generará un PAT (token de acceso personal) que permite realizar llamadas remotas a la API de AAP. Esta API facilita el lanzamiento de plantillas de trabajo para ejecutar tareas de configuración y administración del sistema de forma remota.

2. **Crear la reacción en Centreon** utilizando controladores de eventos. Para ello, se accede a "Configuración → Comandos" para crear un comando. Para conectarse con la API de APP, se utilizará el comando curl, especificando la plantilla de AAP a ejecutar, el PAT generado en el paso anterior, el servidor afectado y el servicio afectado.

```
curl -L -k -X POST   -H "Authorization: Bearer *****""
       -H "Content-Type: application/json"
       -d '{
           "extra_vars": {
               "centreon_host": "$HOSTNAME$",
               "centreon_service": "$TOTALSERVICEPROBLEMS$",
               }
           }'   https://ansible1.mydomain.local/api/v2/job_templates/14/launch
```

3. **Asignación del comando a una plantilla**: Finalmente, es necesario asignar este comando a la plantilla de monitorización de un servicio específico y especificar el tiempo de espera desde que se genera la alarma hasta que se ejecuta el comando.

   En este caso, se ha asignado a la plantilla de monitorización de los servicios http, sssd, ssh, haproxy e ipa un minuto de espera desde que se detecta la alarma hasta que se ejecuta el comando.

5.4.5. Creación de Tickets en Jira

Cuando Centreon detecta una alarma durante la monitorización, se genera automáticamente un ticket en Jira en el proyecto correspondiente para su revisión.

1. **Autorizar las conexiones de AAP a la API de Jira**: El primer paso es crear un usuario dentro de Jira para que AAP pueda autenticarse con él. En Jira, ir a "Configuración → Gestión de Usuarios" y añada un nuevo usuario. Una vez creado, asignar los permisos adecuados para la creación de tickets y generar un token de API que se utilizará en los playbooks para la autenticación.

2. **Plantilla de trabajo de AAP**: Crear una plantilla de trabajo en AAP que permita la creación de tickets en Jira a través de la API, utilizando las variables proporcionadas por Centreon. Este playbook también puede configurarse para otras acciones, como enviar correos electrónicos o notificar problemas por otros medios. El playbook específico se encuentra en el Anexo 2.

3. **Crear la reacción en Centreon**: Configurar un nuevo comando en Centreon que envíe una solicitud a AAP para generar el ticket en Jira, pasando como parámetros la información relevante sobre la alerta detectada.

4. **Programación y retraso en Centreon**: Asignar este comando a la plantilla de monitorización de servicios de producción y formación. Programar su ejecución con un retraso de tres minutos para evitar notificaciones en caso de que Centreon haya podido resolver el problema de forma autónoma. Si no puede resolverlo, se generará automáticamente un ticket en Jira para su revisión por un técnico.

```
curl -L -k -X POST \
```

```
                -H "Authorization: Bearer *****" \
                -H "Content-Type: application/json" \
                -d '{
                  "extra_vars": {
                    "centreon_host": "$HOSTNAME$",
                    "centreon_service": "$TOTALSERVICEPROBLEMS$",
                    "service_state": "$SERVICESTATE",
                    "centreon_date": "$DATE$",
                    "jira_project": "AL"
                  }
                }' https://ansible1.mydomain.local/api/v2/job_templates/15/launch
```

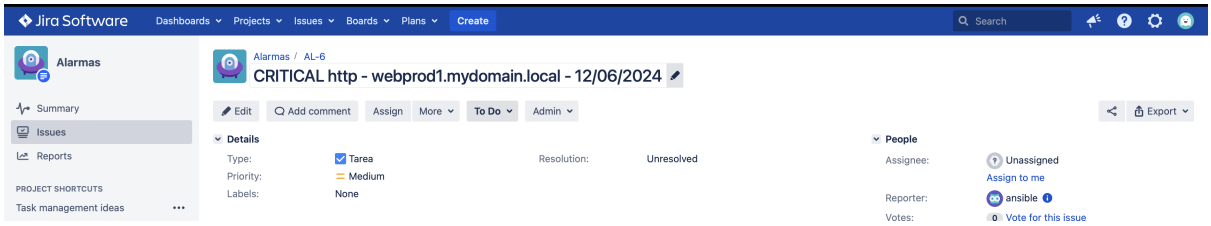


Figura 5.16: Jira creado.

Capítulo 6

Valoración final

Este proyecto ha permitido explorar diversas tecnologías y metodologías en la monitorización de sistemas distribuidos y destacar la importancia de tener una visión global y centralizada del estado del sistema, lo que facilita la detección y resolución proactiva de problemas.

La solución propuesta ha demostrado ser efectiva y cumplir con los objetivos planteados al inicio del proyecto. La automatización en la configuración y despliegue de herramientas de monitorización ha reducido significativamente la necesidad de intervención manual, simplificando el trabajo del administrador de sistemas y permitiendo un ahorro considerable de tiempo y costos.

Aunque se trata de una prueba de concepto, el desarrollo del prototipo ha superado diversas complicaciones y ha cumplido con las funcionalidades propuestas inicialmente. El prototipo funcional ha demostrado claramente los beneficios potenciales de una solución integrada de monitorización, validando su viabilidad y eficacia en un entorno real dentro de una empresa. La capacidad de integrar múltiples herramientas y automatizar procesos ha mostrado ser útil para optimizar la gestión y el mantenimiento de infraestructuras complejas.

Para seguir mejorando y expandiendo la solución de monitorización de sistemas distribuidos, se proponen los siguientes desarrollos futuros:

- **Desarrollo de nuevas funcionalidades:** Implementar características adicionales que permitan una mayor personalización de sistema y adaptación de la plataforma a diferentes entornos empresariales. Esto incluiría la posibilidad de configurar alertas específicas o personalizar los paneles de control según las necesidades particulares de cada empresa.
- **Integración con más herramientas de monitorización:** Ampliar la compatibilidad de la plataforma con otras herramientas y tecnologías de monitorización, mejorando así su versatilidad y eficacia. Por ejemplo, se podría integrar una herramienta de monitorización de seguridad o un escáner de vulnerabilidades, lo cual permitiría detectar y gestionar riesgos de seguridad de manera eficiente.
- **Seguridad y privacidad:** Implementar mecanismos avanzados de seguridad para proteger los datos monitorizados y garantizar la privacidad de la información. Esto podría incluir políticas de contraseñas robustas, cifrado

de datos sensibles almacenados en AAP, o la implementación de autenticación multifactor para los usuarios.

- **Análisis predictivo:** Incorporar algoritmos de análisis predictivo y machine learning para anticipar posibles fallos o incidencias antes de que ocurran. Estos algoritmos podrían analizar patrones en los datos monitorizados para predecir problemas y permitir a los administradores tomar medidas preventivas.
- **Interfaz de usuario mejorada:** Desarrollar una interfaz de usuario más intuitiva y amigable, añadiendo plantillas gráficas a herramientas como Centreon para mostrar un esquema visual del estado actual del sistema. Esto ayudaría a los administradores a comprender rápidamente la situación y a tomar decisiones informadas.
- **Documentación y formación:** Integrar una herramienta de documentación que registre los problemas más comunes y sus soluciones. Se podría utilizar la herramienta Confluence, que se integra estrechamente con Jira, para proporcionar a los administradores de sistemas una base de conocimientos completa y accesible, facilitando la resolución de incidencias recurrentes.
- **Recuperación ante desastres:** Implementar una política de respaldo para los servidores, que permita restaurar el estado del sistema a una fecha específica en caso de ser necesario. Esto garantizará la continuidad del servicio y la integridad de los datos ante posibles fallos o incidentes críticos.

Estos desarrollos futuros no solo mejorarían la funcionalidad y la eficacia de la plataforma de monitorización, sino que también incrementarían su valor para las empresas al ofrecer una solución más robusta, segura y fácil de usar.

En resumen, este proyecto ha demostrado que una solución centralizada y automatizada para la monitorización de sistemas distribuidos es viable y beneficiosa para la gestión eficiente de infraestructuras tecnológicas en entornos empresariales. Además, ha cumplido con las expectativas y ha abierto la puerta a futuras mejoras que aumentarán la utilidad y eficacia de la plataforma, ofreciendo un valor añadido significativo a las empresas que la adopten.

Apéndice A

Instalación del Entorno

A.1. Instalación CentOS Stream 9

Los pasos para instalar CentOS Stream 9 en un servidor virtualizado en VMWare son:

1. **Descargar la Imagen ISO**: Dado que es software de código libre, la imagen ISO de CentOS puede descargarse gratuitamente desde su página web oficial[41].
2. **Cargar la ISO en una máquina virtual de VMWare**: Para ello, se crea una nueva máquina virtual. En la etapa de selección del disco de instalación, se elige la ISO descargada en el paso anterior. Posteriormente, se detallará cómo se puede automatizar esta tarea.
3. **Seleccionar la opción de instalar Centos Stream 9**: Durante el proceso de instalación, se elige específicamente la versión de CentOS Stream 9 para la configuración de la máquina virtual.

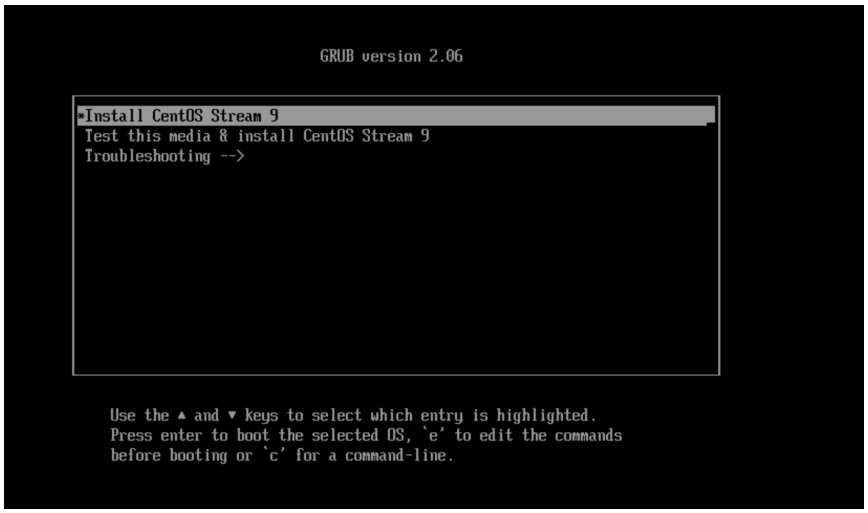


Figura A.1: Consola de instalación de Centos Stream 9.

4. **Configurar el sistema:** Durante el proceso de instalación, se debe elegir un disco de destino donde se instalará CentOS. Además, es necesario activar

la cuenta de root, lo cual es necesario para la gestión del sistema, y seleccionar la opción de instalar CentOS como un servidor sin interfaz gráfica de usuario, lo que es preferible en muchos entornos de servidor por razones de eficiencia y seguridad.

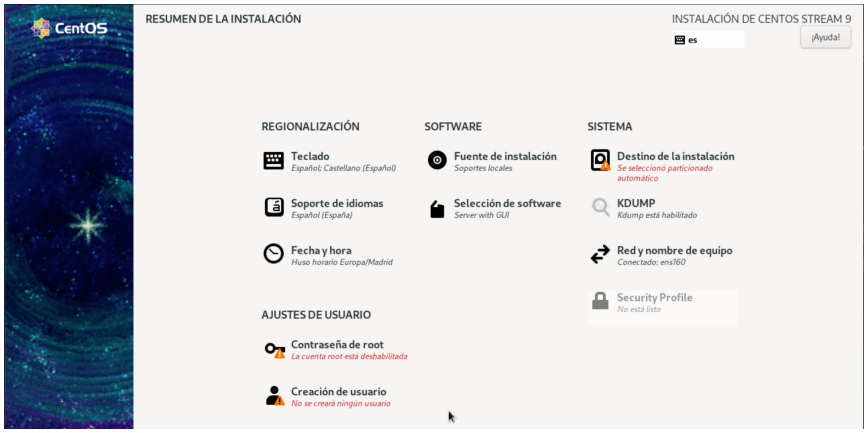


Figura A.2: Configuración de Centos Stream 9.

5. **Acceder al servidor:** Tras completar la instalación, se debe acceder al servidor a través de la consola utilizando las credenciales de root.

A.2. Instalación Servidor IdM

Pasos seguidos para configurar un servidor de IdM, que deben realizarse como root:

1. **Fijar el hostname del servidor**: El hostname es el nombre único asignado a un dispositivo (como una computadora, un servidor o una impresora) en una red, utilizado para identificarlo y permitir la comunicación entre diferentes dispositivos.

```
[root@ipa1 ~]# hostnamectl set-hostname ipa1
```

2. **Instalar los paquetes necesarios**:

```
[root@ipa1 ~]# yum install -y ipa-server ipa-server-dns
```

3. **Instalar el servidor más básico de IdM**: El comando *ipa-server-install* instala y configura un servidor IPA (Identity, Policy, and Audit) en sistemas Linux, estableciendo servicios de gestión de identidades, autenticación (Kerberos), y opcionalmente DNS y otros componentes relacionados.

```
[root@ipa1 ~]# ipa-server-install
```

4. **Configurar Fully Qualified Domain Name (FQDN)**: Es una forma de nombrar un dispositivo dentro de una red de forma única e identificar su posición exacta en la jerarquía del DNS. Es la combinación del nombre del host

(hostname) y el dominio. Dominio es el nombre de la red al que pertenece el dispositivo. Siguiendo con el ejemplo anterior, el hostname *ipa1* en el dominio *mydomain.local*.

```
Enter the fully qualified domain name of the computer
on which you're setting up server software. Using the form
<hostname>.<domainname>
Example: master.example.com


Server host name [ipa1]: ipa1.mydomain.local
```

Figura A.3: Configuración FQDN

5. **Configurar cuentas de administración**: En los sistemas IdM hay algunas diferencias clave entre las cuentas de root, admin y Directory Manager. Root tiene acceso total al sistema, admin se enfoca en tareas administrativas dentro de IDM, y Directory Manager está especializado en la gestión del directorio LDAP.

```
Certain directory server operations require an administrative user.
This user is referred to as the Directory Manager and has full access
to the Directory for system management tasks and will be added to the
instance of directory server created for IPA.
The password must be at least 8 characters long.

Directory Manager password:
Password (confirm):

The IPA server requires an administrative user, named 'admin'.
This user is a regular system account used for IPA server administration.

IPA admin password:
Password (confirm):
```

Figura A.4: Configuración cuentas

6. **Revisar parámetros y continuar**:

```
The IPA Master Server will be configured with:
Hostname:       ipa1.mydomain.local
IP address(es): 192.168.164.133
Domain name:    mydomain.local
Realm name:     MYDOMAIN.LOCAL

The CA will be configured with:
Subject DN:   CN=Certificate Authority,O=MYDOMAIN.LOCAL
Subject base: O=MYDOMAIN.LOCAL
Chaining:     self-signed

Continue to configure the system with these values? [no]: yes

The following operations may take some minutes to complete.
Please wait until the prompt is returned.
```

Figura A.5: Revisión de configuración

7. **Comprobar si todos los servicios están corriendo**:

```
[root@ipa1 ~]# ipactl status
Directory Service: RUNNING
krb5kdc Service: RUNNING
kadmin Service: RUNNING
httpd Service: RUNNING
ipa-custodia Service: RUNNING
pki-tomcatd Service: RUNNING
ipa-otpd Service: RUNNING
ipa: INFO: The ipactl command was successful
```

Figura A.6: Comprobación de servicios

8. **Comprobar el acceso con admin**:

```
[root@ipa1 ~]# kinit admin
Password for admin@MYDOMAIN.LOCAL:
```

Figura A.7: Comprobación de cuenta admin

9. **Comprobar el acceso a la interfaz gráfica**.

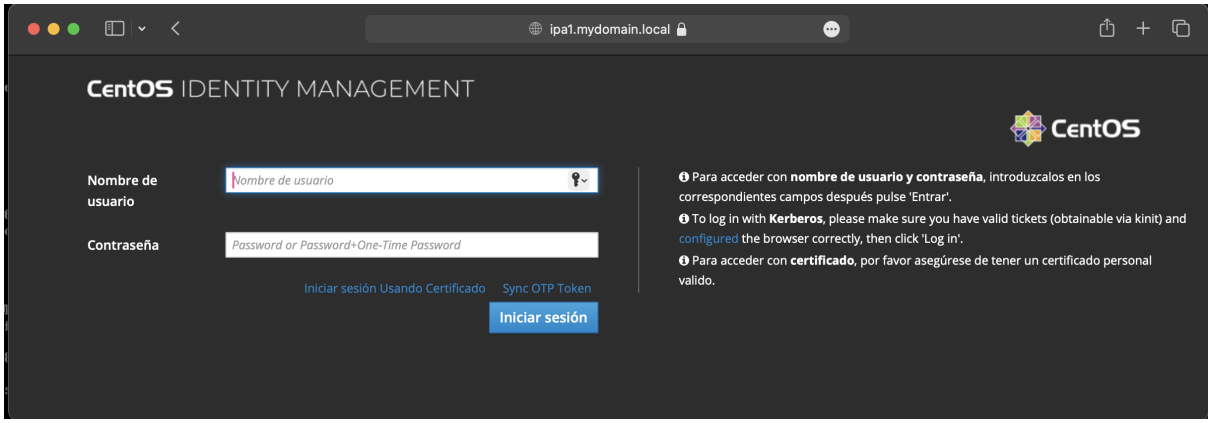


Figura A.8: Inicio de sesión desde la interfaz gráfica

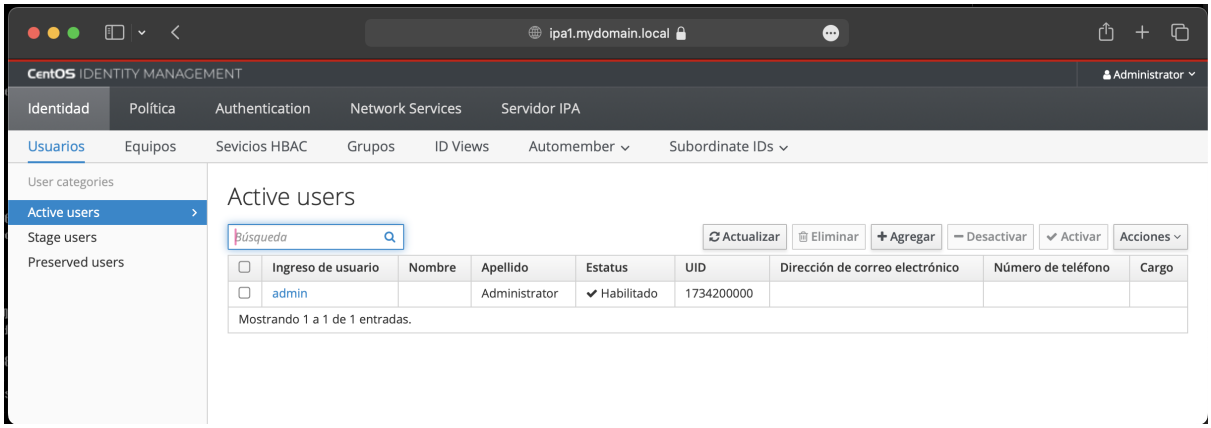


Figura A.9: Interfaz gráfica IdM

10. **Crear usuario y gestionar sus accesos**: La creación de usuarios es más sencilla a través de la interfaz gráfica. Para crear el usuario *user1*, se accede a "Identidad → Usuarios → Agregar". A continuación le daremos permisos para convertirse en root en todos los clientes.

- **Reglas HBAC**: Las reglas de control de acceso basado en host (HBAC) definen qué usuarios o grupos pueden acceder a qué hosts o grupos de hosts utilizando qué servicios. Por defecto, IdM tiene una regla HBAC llamada *allow_all* que permite acceso universal a todos los usuarios a través de cada servicio en cada host. Este acceso puede ser ajustado sustituyendo la regla *allow_all* por reglas personalizadas para restringir el acceso a diferentes hosts.
- **Reglas Sudo**: Estas reglas determinan qué comandos pueden ejecutar los usuarios y bajo qué condiciones en los clientes IdM. Para otorgar permisos sudo al nuevo usuario, se debe ir a "Política → Sudo → Reglas Sudo" y crear una nueva regla usando el botón agregar. Se edita la regla añadiendo la opción *!authenticate* para que no solicite contraseña cada vez que se ejecute sudo, permitiendo al usuario *user1* ejecutar todos los comandos en todos los hosts.

A.3. Instalación Cliente IdM

Para instalar un cliente de Red Hat Identity Management (IdM), es necesario configurar el host para que se integre con un servidor IdM. Esta configuración permite que el cliente acceda a los usuarios creados en el servidor IdM.

1. **Instalar el paquete del cliente.**

```
[root@jira1 ~]# yum install ipa-client
```

2. **Añadir al archivo */etc/hosts*** el FQDN (Fully Qualified Domain Name) del cliente y del servidor, utilizando el dominio *mydomain.local*. Se debe especificar primero el FQDN y luego el nombre de host.

```
[root@jira1 ~]# cat /etc/hosts
127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
::1         localhost localhost.localdomain localhost6 localhost6.localdomain6

192.168.164.134 jira1.mydomain.local jira1
192.168.164.133 ipa1.mydomain.local ipa1
```

Figura A.10: Fichero `/etc/hosts` del cliente.

3. **Instalar el cliente de IdM:** El comando *ipa-client-install* se usa para registrar un sistema cliente con un servidor IPA, configurando así el cliente para autenticación y gestión de identidades dentro de la red de una organización. Este proceso incluye configurar Kerberos para autenticación segura y actualizar los archivos de configuración necesarios para integrar el cliente en el entorno IPA. Ejecutar el comando de instalación del cliente IdM, especificando el dominio y el servidor IdM. La opción *–mkhomedir* crea un directorio home para los usuarios que se conecten.

```
[root@jira1 ~]# ipa-client-install --domain=mydomain.local --server=
    ipa1.mydomain.local --mkhomedir
```

4. **Comprobar la existencia del usuario** *user1*.

```
[root@jira1 ~]# id user1
```

A.4. Instalación de Jira

1. **Instalar dependencias**.

```
[root@jira1 opt]# yum install jre
```

2. **Descargar el paquete de Jira** desde su sitio web[37]. Seleccionar la versión *standalone*, que permite operar el servidor independientemente de otros sistemas.
3. **Mover el fichero descargado al directorio */opt* y descomprimirlo**. El directorio */opt* se utiliza comúnmente para instalar y almacenar aplicaciones de software que no forman parte del sistema operativo central o que no vienen con la distribución por defecto, incluyendo software comercial o de terceros como Jira.

```
[root@jira1 opt]# tar -xfv atlassian-jira-software-9.12.1.tar
```

4. **Configurar el directorio de inicio de Jira[51]**. Este directorio almacena datos clave que son esenciales para el funcionamiento de Jira.

```
[root@jira1 opt]# mkdir /opt/atlassian-jira/home
```

5. **Añadir la localización del directorio */home*** a los ficheros:
   - `atlassian-jira/WEB-INF/classes/jira-application.properties`
   - `bin/start-jira.sh`.

```
[root@jira1 atlassian-jira]# grep jira /opt/atlassian-jira/atlassian-jira/WEB-INF/classes/jira-application.properties
jira.home = /opt/atlassian-jira/home
[root@jira1 atlassian-jira]# grep JIRA_HOME /opt/atlassian-jira/bin/start-jira.sh
export JIRA_HOME=/opt/atlassian-jira/home
[root@jira1 atlassian-jira]#
```

Figura A.11: Ficheros que hacen referencia al home de Jira.

6. **Ejecutar el script de arranque de Jira**: El script */opt/atlassian-jira/bin/start-jira.sh* prepara el entorno del servidor ejecutando una serie de comandos y verificaciones. Configura las variables de entorno necesarias y comprueba que todos los componentes y pre-requisitos, como la versión adecuada de Java, estén presentes y configurados correctamente.

```
[root@jira1 opt]#/opt/atlassian-jira/bin/start-jira.sh
```

7. **Acceder al servidor web** iniciado por el script de arranque mediante HTTP. Para ello, usar la dirección IP y el puerto 8080.
8. **Seleccionar "Configúrenlo por mi" y generar la licencia de Jira.**

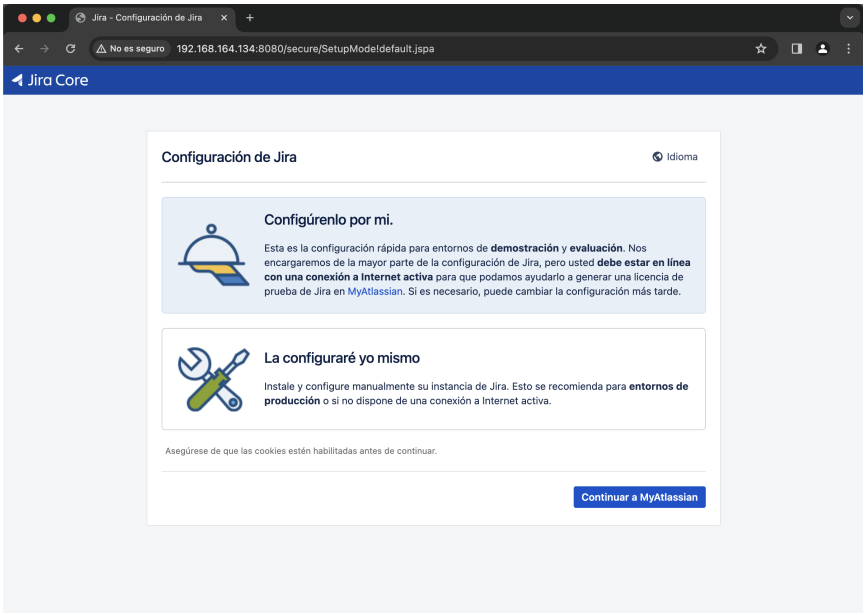


Figura A.12: Pantalla inicial de configuración de Jira.

9. **Crear una cuenta de administrador**.

Figura A.13: Fin del instalador de Jira.

10. **Crear un proyecto:** Una vez terminada la instalación, acceder a la web indicada por el instalador para crear un proyecto del tipo "Administración de tareas".

    Se nombrará este proyecto *Alarmas* y se le asignará la clave *AL*, la cual se utiliza para identificar el proyecto al que pertenece una tarea.

11. **Integrar Jira con IdM:** Para integrar Jira con los usuarios del servidor IdM, acceder al menú "Administración → Administración de Usuarios → Directorio de Usuario → Añadir Directorio LDAP". Añadir el nombre del servidor IdM, seleccionar "Generic Directory Server" y probar la conexión.

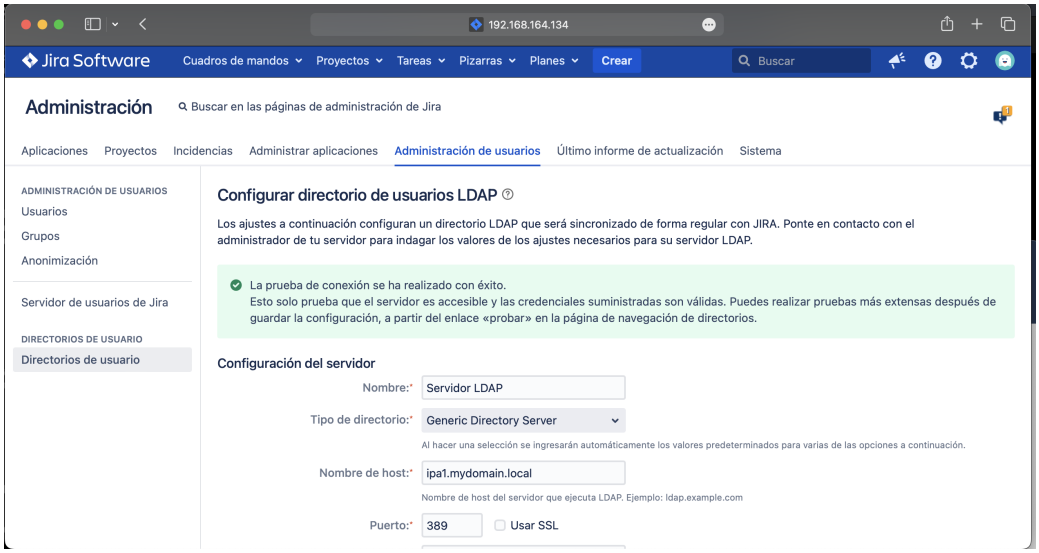


Figura A.14: Integración del servidor IdM en Jira.

Una vez que el servidor verifique la conexión, introducir un usuario con permisos de administrador de IdM para sincronizar los datos.

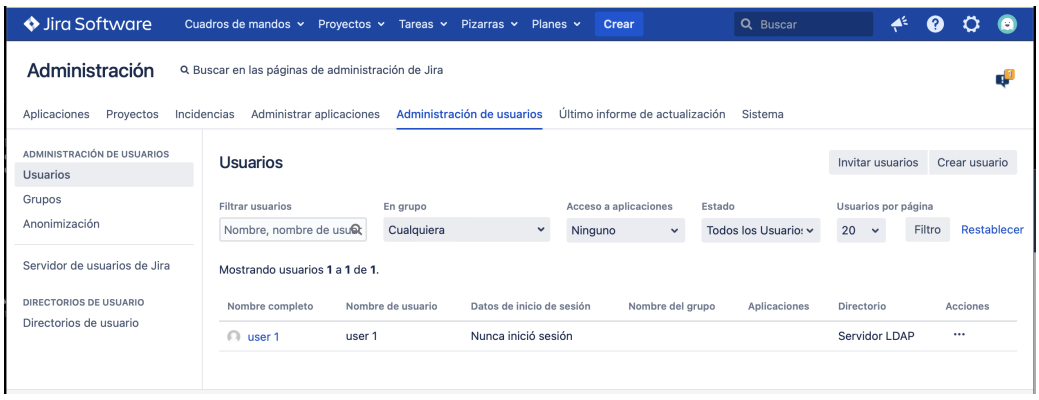


Figura A.15: Usuario del servidor IdM en Jira.

12. **Crear un servicio systemd** de Linux para que Jira se inicie automáticamente al reiniciar el servidor. Systemd es un conjunto de demonios, herramientas, librerías y servicios para administrar y configurar servicios en sistemas operativos Linux.

Crear el archivo */etc/systemd/system/jira.service* con el contenido:

```
[[Unit]
Description=Atlassian Jira
After=network.target

[Service]
Type=forking
User=root
LimitNOFILE=20000
PIDFile=/opt/atlassian-jira/work/catalina.pid
ExecStart=/opt/atlassian-jira/bin/start-jira.sh
ExecStop=/opt/atlassian-jira/bin/stop-jira.sh

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

13. **Activar el servicio** para que se levante cuando se arranque el servidor

```
[root@jira1 opt]# systemctl daemon-reload
[root@jira1 opt]# systemctl enable jira
[root@jira1 opt]# systemctl start jira
```

A.5. Instalación de AAP

La herramienta Red Hat Ansible Automation Platform (AAP) actualmente no está disponible para RedHat 9 o CentOS 9, por lo que se realizará la instalación en un servidor con RedHat 8.9. [52]

1. **Obtener una suscripción activa** a Red Hat Ansible Automation Platform a través del portal de clientes de Red Hat[53], lo que permite descargar un archivo tar con todo lo necesario para la instalación.
2. **Descargar el archivo tar**: en el servidor deseado y descomprimirlo en el directorio */opt*.

```
tar -xvf nombre_del_archivo.tar
```

3. **Instalar las dependencias**.

```
[root@ansible1 ~]# yum install ansible
```

4. **Editar el archivo *inventory***: añadiendo los parámetros necesarios. Para una instalación en un solo servidor, se utiliza un archivo de inventario básico.

```
[automationcontroller]
172.20.10.4 ansible_connection=local
[database]
[all:vars]
admin_password='admin1234'
pg_host=''
pg_port=''
pg_database='awx'
pg_username='awx'
pg_password='pg1234'
```

- **[automationcontroller]**: Define un grupo de hosts llamado *automation-controller*
  - **172.20.10.4 ansible_connection=local**: Indica que Ansible debe conectarse a este host utilizando una conexión local, es decir, se asume que los comandos de Ansible se ejecutan en la misma máquina y no a través de SSH o otro protocolo remoto.
- **[database]**: Se usaría para listar los hosts que actúan como servidores de base de datos.
- **[all:vars]**: Define variables que se aplican a todos los hosts del inventario.
  - **admin_password='admin1234'**: Contraseña de administrador.
  - **pg_host=''**: Host de PostgreSQL, actualmente no especificado.
  - **pg_port=''**: Puerto de PostgreSQL, actualmente no especificado.
  - **pg_database='awx'**: Nombre de la base de datos PostgreSQL.
  - **pg_username='awx'**: Nombre de usuario para PostgreSQL.
  - **pg_password='pg1234'**: Contraseña para PostgreSQL.

5. **Ejecutar el script *setup.sh***. Este script instala y configura los componentes necesarios de la plataforma, verifica los requisitos del sistema, establece configuraciones y prepara la plataforma para su uso. Su ejecución llevara varios minutos.

```
[root@ansible1 ansible-automation]# sh setup.sh
```

6. **Acceder a la interfaz web** con el usuario y contraseña configurados en el archivo de inventario.
7. **Seleccionar la licencia** obtenida en el primer paso.

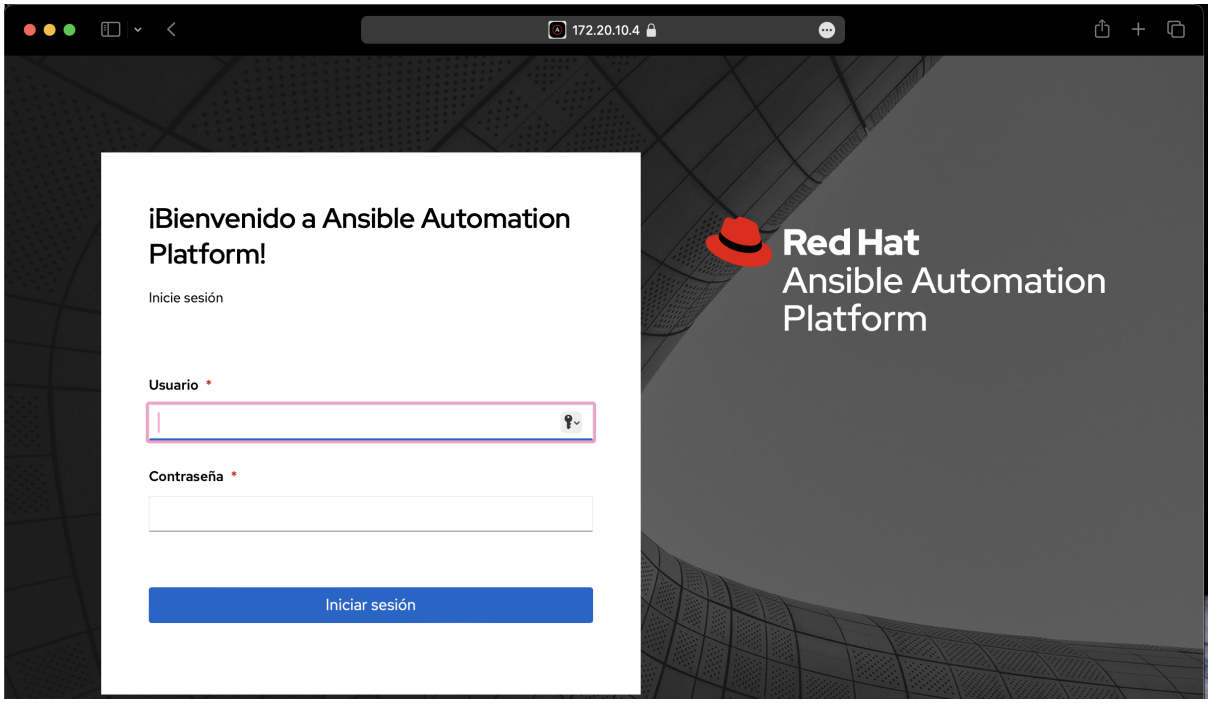


Figura A.16: Página principal de AAP

8. **Acceder al panel del control.**

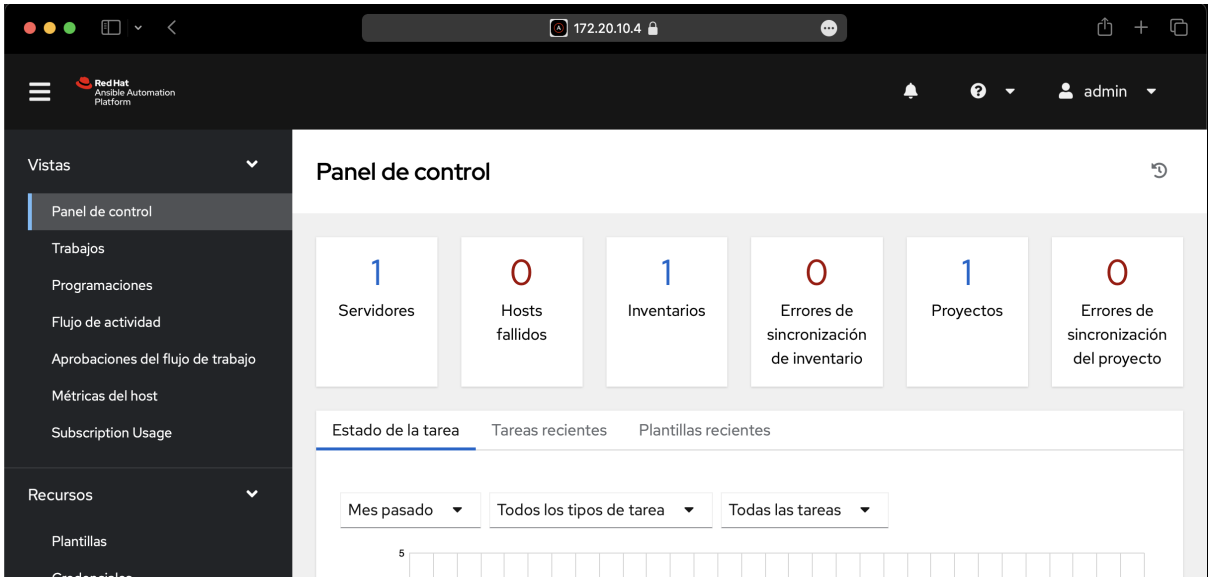


Figura A.17: Panel de control AAP

A.6. Instalación Apache

1. **Instalar el paquete httpd.**

```
[root@webprod1 ~]# yum install httpd
```

2. **Arrancar el servicio httpd** y configurarlo para que se inicie automáticamente al encender el servidor.

```
[root@webprod1 ~]# systemctl start httpd
[root@webprod1 ~]# systemctl enable httpd
```

3. **Comprobar la instalación** accediendo a la página web de prueba mediante la URL *http://webprod1.mydomain.local*.

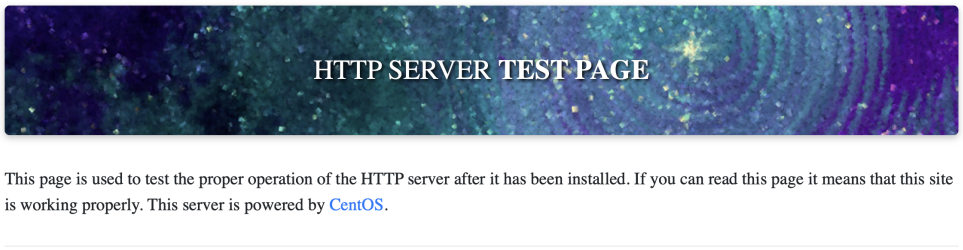


Figura A.18: Página web de prueba.

4. **Configurar el contenido de la página web**. Para ello, crear o modificar el fichero */var/www/html/index.html* con el siguiente contenido:

```
<html>
  <head>
    <title>Web de Produccion</title>
  </head>
  <body>
    <h1>Web de Produccion</h1>
    <h2>Se esta accediendo al servidor webprod1</h2>
  </body>
</html>
```

El archivo *index.html* es la página de inicio por defecto en la mayoría de los sitios web. Este archivo, que contiene código HTML, es lo que primero que ven los visitantes al acceder a un sitio web.

Una vez configurado, al acceder a la web de *webprod1* se mostrará:

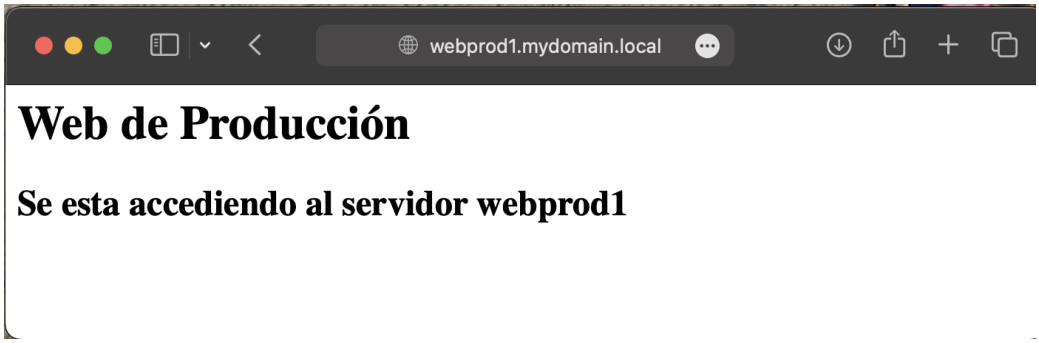


Figura A.19: Página web configurada de webprod1.

5. Repetir los mismos pasos para los servidores *webprod2* y *webform1*, modificando el archivo *index.html* para indicar el servidor específico al que se está accediendo.

A.7. Instalación Haproxy

1. **Instalar el paquete haproxy**

```
[root@balancerprod1 ~]# yum install haproxy
```

2. **Modificar el fichero de configuración *letc/haproxy/haproxy.cfg*** para indicar que balancee las visitas a la pagina web de producción entre los servidores *webprod1* y *webprod2*.

```
listen web_server
    bind    *:80
    mode    http
    stats   enable
    stats   auth    admin:admin
    balance roundrobin
    server  webprod1      192.168.164.140
    server  webprod2      192.168.164.141
```

   - **listen web_server**: Inicia la configuración del balanceador de carga con el nombre *web_server*.
   - **bind *:80**: Configura el balanceador para escuchar en el puerto 80, el puerto estándar para HTTP, en todas las interfaces de red.
   - **mode http**: Establece el balanceador para trabajar en modo HTTP.
   - **stats enable**: Activa la generación de estadísticas para el balanceador.
   - **stats auth admin:admin**: Establece *admin* como el usuario y contraseña para acceder a las estadísticas.
   - **balance roundrobin**: Utiliza el algoritmo de balanceo de carga round-robin. El algoritmo round-robin distribuye las solicitudes entrantes de manera secuencial y equitativa entre todos los servidores disponibles.
   - **server webprod1 192.168.164.140** y **server webprod2 192.168.164.141**: Define los servidores web a los que se debe acceder.

   Esta configuración permite el acceso a uno de los dos servidores web a través de la dirección del balanceador.

3. **Arrancar y habilitar el servicio haproxy.**

```
[root@balancerprod1 ~]# systemctl start haproxy
[root@balancerprod1 ~]# systemctl enable haproxy
```

4. **Acceder a la web** de producción usando la ip de *balancerprod1*.

Figura A.20: Redireccionamiento a la web1

En este caso, el balanceador ha redirigido a *webprod1*. Para probar su funcionamiento, detener el servicio httpd de *webprod1* y acceder nuevamente para verificar si redirige a *webprod2*.

Figura A.21: Redireccionamiento a la web2

Se debe repetir los mismos pasos para el entorno de formación con *balancerform1*. Aunque solo hay un servidor web y no es necesario un balanceador, se instala para replicar lo más posible el escenario de producción.

A.8. Instalación de Centreon

Los pasos para instalar centreon son: [54]

1. **Deshabilitar SELinux**. SELinux (Security-Enhanced Linux) es un módulo de seguridad integrado en el kernel de Linux que proporciona mecanismos avanzados de control de acceso para mejorar la seguridad del sistema operativo. Se basa en políticas que definen cómo los procesos y usuarios pueden interactuar con diversos recursos del sistema, limitando así las acciones que pueden realizar y aumentando la seguridad contra accesos no autorizados o maliciosos.

```
1                    [root@centreon1 ~]# sed -i s/^SELINUX=.*$/SELINUX=disabled/ /
                         etc/selinux/config
```

2. **Parar y deshabilitar el firewall**.

```
[root@centreon1 ~]# systemctl stop firewalld
[root@centreon1 ~]# systemctl disable firewalld
```

3. **Reiniciar** el servidor para aplicar cambios.

```
[root@centreon1 ~]# shutdown -r now
```

```
[root@centreon1 ~]# getenforce
Disabled
[root@centreon1 ~]#
```

Figura A.22: Configuración de SELinux

4. **Instalar el repositorio CodeReady Builder**.

```
[root@centreon1 ~]# dnf install epel-release epel-next-release
NO [root@centreon1 ~]# dnf config-manager --set-enabled
    codeready-builder-for-rhel-9-rhui-rpms
```

5. **Habilitar PHP**.

```
[root@centreon1 ~]# dnf module reset php
[root@centreon1 ~]# dnf module install php:8.1
```

6. **Instalar repositorio de Centreon**.

```
[root@centreon1 ~]# dnf install -y dnf-plugins-core
[root@centreon1 ~]# dnf config-manager --add-repo https://
    packages.centreon.com/rpm-standard/23.10/el9/centreon
    -23.10.repo
```

7. **Instalar repositorio MariaDB**.

```
[root@centreon1 ~]# curl -LsS https://r.mariadb.com/downloads/
    mariadb_repo_setup | sudo bash -s -- --os-type=rhel --os-
    version=9 --mariadb-server-version="mariadb-10.5"
```

```
[root@centreon1 centreon]# curl -LsS https://r.mariadb.com/downloads/mariadb_repo_setup | sudo bash -s -- --os-type=rhe
l --os-version=9 --mariadb-server-version="mariadb-10.5"
# [info] Skipping OS detection and using OS type 'rhel' and version '9' as given on the command line
# [info] Checking for script prerequisites.
# [info] MariaDB Server version 10.5 is valid
# [info] Repository file successfully written to /etc/yum.repos.d/mariadb.repo
# [info] Adding trusted package signing keys...
/etc/pki/rpm-gpg /opt/centreon
/opt/centreon
# [info] Successfully added trusted package signing keys
# [info] Cleaning package cache...
58 archivos eliminados
```

Figura A.23: Instalación MariaDB

8. **Instalar el servidor central de Centreon.** Se puede instalar con una base de datos local en el servidor o una base de datos remota en un servidor dedicado, elegimos la primera opción.

```
[root@centreon1 ~]# dnf install -y centreon --enablerepo=crb
[root@centreon1 ~]# systemctl daemon-reload
```

```
3                    [root@centreon1 ~]# systemctl restart mariadb
```

```
[root@centreon1 centreon]# curl -LsS https://r.mariadb.com/downloads/mariadb_repo_setup | sudo bash -s -- --os-type=rhe
l --os-version=9 --mariadb-server-version="mariadb-10.5"
# [info] Skipping OS detection and using OS type 'rhel' and version '9' as given on the command line
# [info] Checking for script prerequisites.
# [info] MariaDB Server version 10.5 is valid
# [info] Repository file successfully written to /etc/yum.repos.d/mariadb.repo
# [info] Adding trusted package signing keys...
/etc/pki/rpm-gpg /opt/centreon
/opt/centreon
# [info] Successfully added trusted package signing keys
# [info] Cleaning package cache...
58 archivos eliminados
```

Figura A.24: Instalación MariaDB

9. **Añadir la zona horaria** a los ficheros de configuración de php y reinicia el servicio.

```
1                    [root@centreon1 ~]# echo "date.timezone = Europe/Madrid" >> /
                         etc/php.d/50-centreon.ini
2                    [root@centreon1 ~]# systemctl restart php-fpm
```

10. **Añadir los servicios** necesarios por Centreon para que se levanten automáticamente tras el arranque.

```
1                  [root@centreon1 ~]#systemctl enable php-fpm httpd centreon cbd
                       centengine gorgoned snmptrapd centreontrapd snmpd crond mariadb
```

11. **Securizar la base de datos**: Desde MariaDB 10.5, es obligatorio asegurar el acceso root a la base de datos antes de instalar Centreon. Para ello hay que ejecutar el siguiente comando:

```
1                 [root@centreon1 ~]# mysql_secure_installation
```

    El comando realiza una seria de preguntas a las que hay que decir que sí a todas menos a la de "¿No permitir el inicio de sesión raíz de forma remota?". Además pide establecer una contraseña para el usuario root de la base de datos. Esta contraseña es necesaria durante la instalación web.

12. **Arrancar servidor web**

```
1                  [root@centreon1 ~]# systemctl start httpd
```

13. **Acceder a la interfaz web** de Centreon usando la URL *http://<IP>/centreon* para comenzar la configuración. Asegurarse que marca todas las dependencias como "Loaded".

14. **Configurar en "Database Information" el acceso a root** de la base de datos con las credenciales creadas en el paso 11. Además, configurar un usuario para el acceso a la base de datos y su contraseña.

15. **Acceder a Centreon** con el usuario y contraseña de admin.

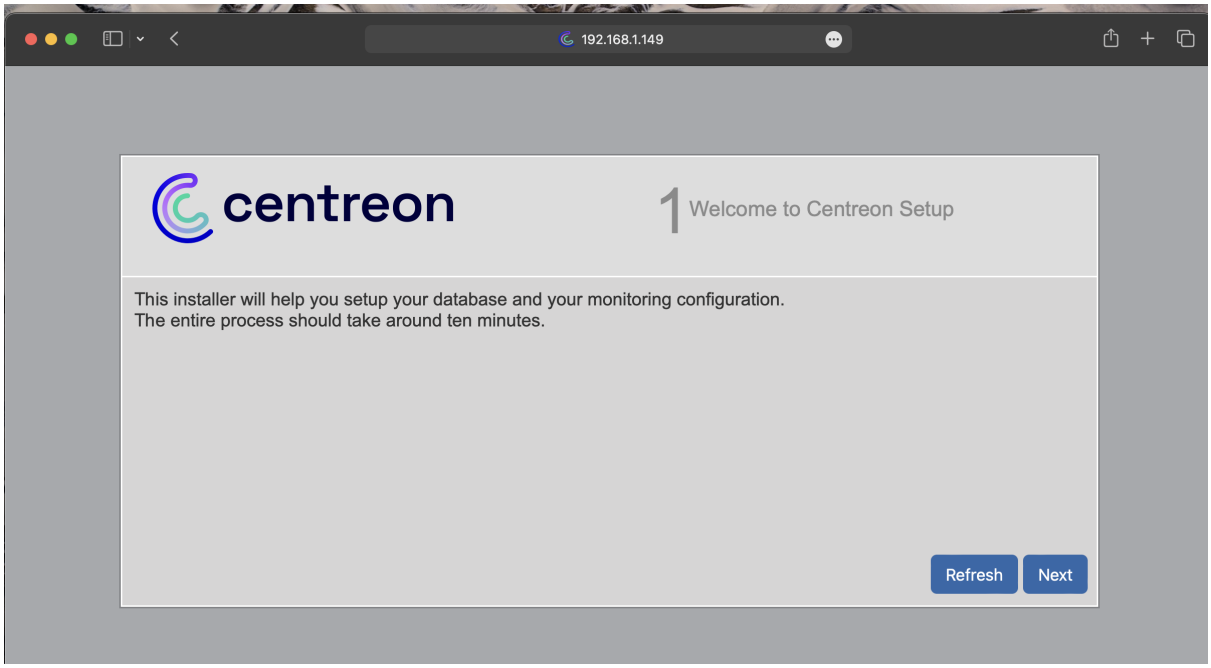


Figura A.25: Configurador de Centreon

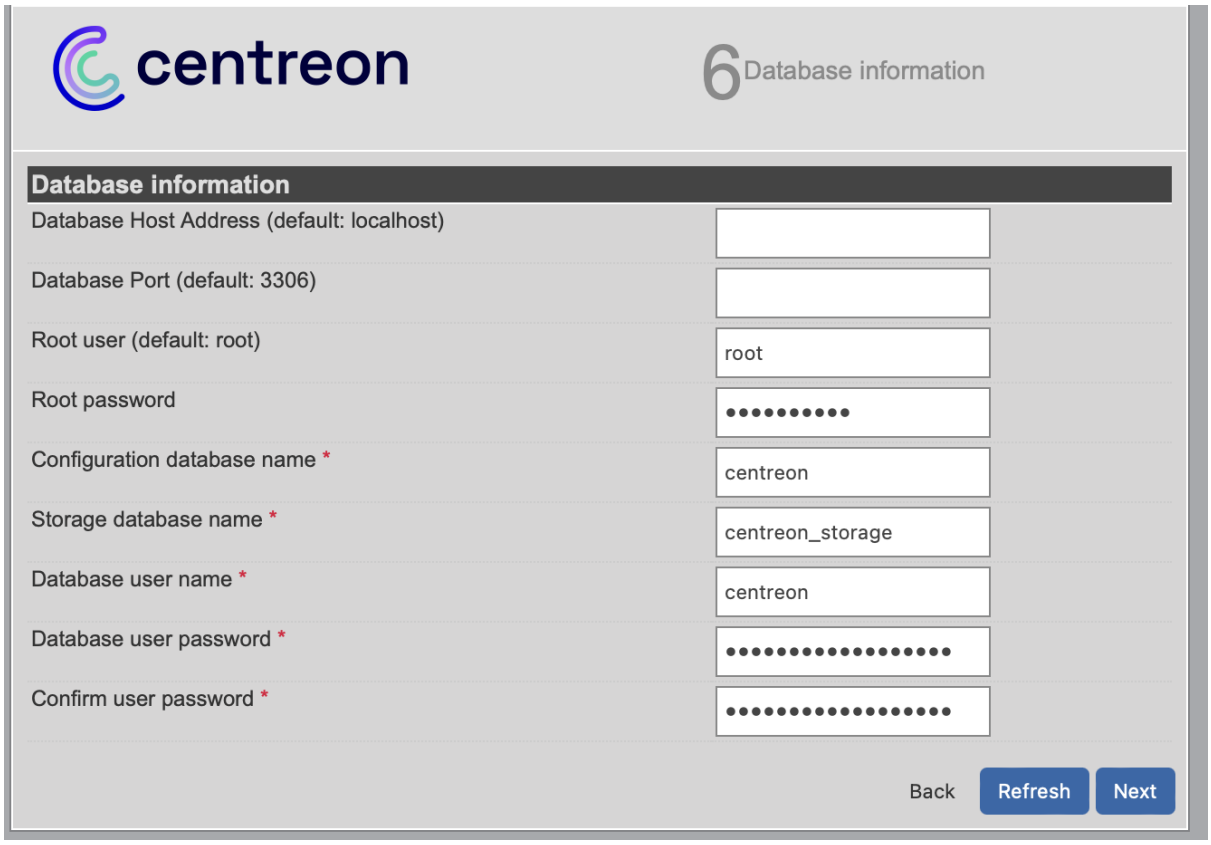


Figura A.26: Configurar la base de datos de Centreon

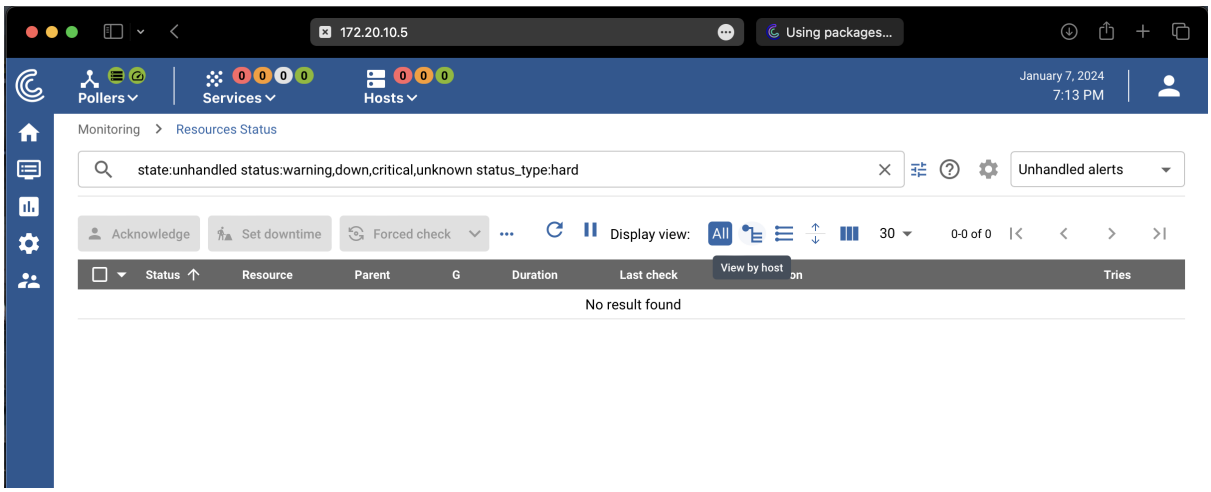


Figura A.27: Interfaz de Centreon

Apéndice B

Automatización

B.1. Automatización del Inventario

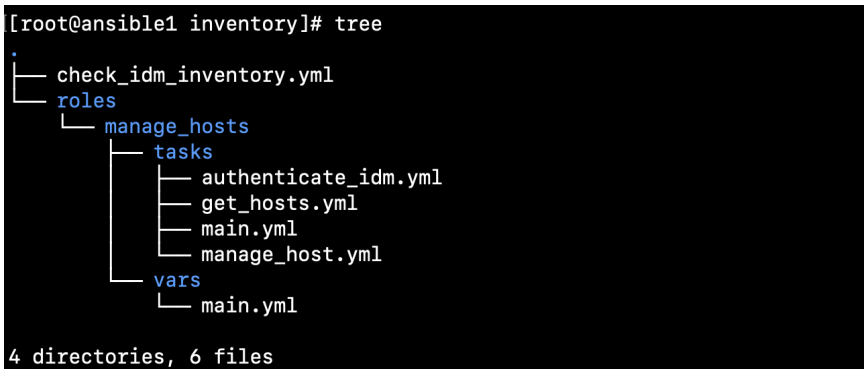


Figura B.1: Estructura check_idm_inventory.yml.

El playbook `check_idm_inventory.yml` está diseñado para gestionar y verificar la presencia de hosts registrados tanto en IdM como en AAP. Además, en caso de detectar un nuevo host, también lanza otro playbook para configurarlo automáticamente.

- **Autenticación con IdM (`authenticate_idm.yml`):** Esta tarea obtiene un ticket de Kerberos de IdM para poder realizar consultas al servidor..
- **Obtención de información de hosts en IdM (`get_hosts.yml`):** Se realiza una consulta al servidor IdM para obtener información detallada sobre los hosts registrados, incluyendo sus nombres de dominio completos (FQDN).
- **Gestión de hosts en AAP (`manage_host.yml`):** Esta tarea se encarga de gestionar los hosts obtenidos de IdM en la aplicación de automatización de procesos (AAP).
  - Verifica si cada host registrado en IdM está también registrado en AAP mediante una solicitud GET al API de AAP.
  - Si el host no está registrado en AAP, se procede a registrarlo utilizando una solicitud POST al endpoint correspondiente.

- Después de registrar correctamente un nuevo host en AAP, Ansible lanza un template de trabajo específico (`configure_host.yml`) en AAP para configurar automáticamente el host según las especificaciones del entorno.

B.1.1. check_idm_inventory.yml

```
- name: Gestionar hosts en IdM y AAP
  hosts: localhost
  roles:
  - manage_hosts
```

B.1.2. roles/manage_hosts/

B.1.2.1. tasks/

B.1.2.1.1 main.yml

```
- include_tasks: authenticate_idm.yml
- include_tasks: get_hosts.yml

- name: Verificar y registrar cada host en AAP y en el inventario formacion
  loop: "{{ hosts_idm_formacion.stdout_lines }}"
  include_tasks: manage_host.yml
    vars:
      ipa_group: "formacion"

- name: Verificar y registrar cada host en AAP y en el inventario produccion
  loop: "{{ hosts_idm_produccion.stdout_lines }}"
  include_tasks: manage_host.yml
    vars:
      ipa_group: "produccion"
```

B.1.2.1.2 authenticate_idm.yml

```
 - name: Autenticacion usando kinit del usuario IdM
   ansible.builtin.expect:
     command: kinit {{ idm_user }}@MYDOMAIN.LOCAL
   responses:
     Password for {{ idm_user }}@MYDOMAIN.LOCAL: "{{ idm_password }}"
```

B.1.2.1.3 get_hosts.yml

```

- name: Obtener la informacion de los hosts de produccion
  ansible.builtin.shell:
    cmd: ipa hostgroup-show produccion | grep miembro | cut -d ":" -f 2 | tr ',' '\n' | sed 's/^ //'
  register: hosts_idm_produccion

- name: Obtener la informacion de los hosts de formacion
  ansible.builtin.shell:
    cmd: ipa hostgroup-show formacion | grep miembro | cut -d ":" -f 2 | tr ',' '\n' | sed 's/^ //'
  register: hosts_idm_formacion

- name: Mostrar FQDNs procesados para produccion
  debug:
    var: hosts_idm_produccion.stdout_lines

```

```
16          - name: Mostrar FQDNs procesados para formacion
17            debug:
18              var: hosts_idm_formacion.stdout_lines
```

B.1.2.1.4 manage_host.yml

```
1
2           - name: Verificar si el host ya esta registrado en AAP
3             ansible.builtin.uri:
4               url: "https://{{ app_host }}/api/v2/hosts/?name={{ item }}"
5               method: GET
6               user: "{{ app_username }}"
7               password: "{{ app_password }}"
8               force_basic_auth: yes
9               validate_certs: no
10            register: host_exists
11            ignore_errors: yes
12
13          - name: Registrar host en AAP si no existe
14            ansible.builtin.uri:
15              url: "https://{{ app_host }}/api/v2/hosts/"
16              method: POST
17              user: "{{ app_username }}"
18              password: "{{ app_password }}"
19              force_basic_auth: yes
20              body_format: json
21              body:
22                name: "{{ item }}"
23                inventory: "{{ app_inventory_id }}"
24              status_code: 201
25              validate_certs: no
26            when: "host_exists is failed or ('results' in host_exists.json and host_exists
                  .json.results | length == 0)"
27
28          - name: Lanzar template de AAP que configura el host (configure_host.yml)
29            ansible.builtin.uri:
30              url: "https://{{ app_host }}/api/v2/job_templates/17/launch"
31              method: POST
32              user: "{{ app_username }}"
33              password: "{{ app_password }}"
34              body_format: json
35              validate_certs: no
36              body:
37                inventory: "{{ app_inventory_id }}"
38                limit_hosts: "{{ item }}"
39            when: "host_exists is failed or ('results' in host_exists.json and host_exists
                  .json.results | length == 0)"
```

B.1.3. vars/

B.1.2.2.1 main.yml

Este archivo almacena las variables necesarias y la contraseña para acceder a los servidores de IdM y AAP. Dado que estos datos son sensibles, se recomienda utilizar alguna herramienta de gestión de claves como Vault para garantizar su seguridad.

```
1           idm_user: "rw_host_idm"  # Usuario IdM con rol de administrador de hosts.
2           idm_password: "***"  # Contrasena del usuario rw_host_idm
3           app_host: "ansible1.mydomain.local"
4           app_username: "rw_inv_app" # Usuario con permiso para editar el inventario en
                ansible1.
5           app_password: "***"  # Contrasena de rw_inv_app.
6           app_inventory_id: "2" # Id del inventario a editar.
```

B.2. Creación Tarea de Jira

B.2.1. jira_task.yml

- **Creación de una tarea en JIRA (`jira_task.yml`):** Se realiza la creación de una nueva tarea utilizando los siguientes parámetros:
  - `uri`: URL del servidor Jira donde se realiza la operación.
  - `token`: Token de autenticación requerido para acceder a JIRA de forma segura.
  - `operation`: Operación a realizar, en este caso, crear una nueva tarea.
  - `project`: Proyecto en el que se crea la tarea, especificado mediante una variable de Ansible (`jira_project`).
  - `summary`: Resumen de la tarea, que incluye información relevante como el nombre del host (`centreon_host`), servicio (`centreon_service`), y fecha (`centreon_date`).
  - `description`: Descripción detallada de la tarea, utilizando la variable `centreon_description` para proporcionar contexto adicional.
  - `issuetype`: Tipo de incidencia o tarea en Jira, en este caso, definido como `Tarea`.
  - `fields`: Campos adicionales como etiquetas (`labels: centreon`) que ayudan a categorizar y organizar la tarea dentro de Jira.

```
1          - name: Crear una tarea en Jira
2            jira:
3              uri: "http://jira1.mydomain.local"
4              token: "{{ jira_token}}"
5              operation: create
6              project: "{{ jira_project }}"
7              summary: "{{ centreon_host }} - {{ centreon_service }} - {{ centreon_date
                   }}"
8              description: "{{ centreon_description }}"
9              issuetype: Tarea
10             fields:
11               labels:
12                 - centreon
13           register: jira_issue
14           ignore_errors: yes
15
16         - name: Mostrar detalles de la tarea creada
17           debug:
18             var: jira_issue
```

```
[root@ansible1 configure_host]# tree
.
├── configure_host.yml
└── roles
    └── configure_host
        ├── handlers
        │   └── main.yml
        ├── tasks
        │   ├── configure_snmp.yml
        │   ├── install_packages.yml
        │   ├── main.yml
        │   ├── reboot_server.yml
        │   └── secure_system.yml
        └── vars
            └── main.yml

5 directories, 8 files
```

Figura B.2: Estructura configure_host.yml

B.3. Configure Host

El playbook *configure_host.yml* automatiza la configuración inicial del servidor mediante tareas modulares:

- **Instalación de paquetes necesarios (`install_packages.yml`):** Esta tarea instala los paquetes especificados en la variable `packages`.
- **Configuración del servicio SNMP (`configure_snmp.yml`):**
  - **Añadir archivo de configuración SNMP:** Crea el archivo `/etc/snmp/snmpd.conf` con configuraciones como la comunidad SNMP y la descripción del sistema basada en la variable `entorno`.
  - **Habilitar servicio SNMP:** Asegura que el servicio SNMP esté activo para permitir la gestión y monitorización remota.
- **Aplicación de medidas de seguridad (`secure_system.yml`):** Este módulo aplica configuraciones como SELinux en modo enforcing y deshabilita el acceso root por SSH para reforzar la seguridad del sistema. Se usa a modo de ejemplo de tareas posibles a ejecutar en este rol.
- **Reinicio del servidor (`reboot_server.yml`):** Reinicia el servidor con un tiempo máximo de 300 segundos (`reboot_timeout: 300`) para aplicar las configuraciones realizadas de manera efectiva.

B.3.1. configure_host.yml

```
1
2             - name: Configuracion inicial de un servidor
3               hosts: all
4               become: yes
5               roles:
6                 - configure_host
```

B.3.2. roles/configure_host/

B.3.2.1. handlers/

B.3.2.1.1 main.yml

```
            - name: restart snmpd
              ansible.builtin.service:
                name: snmpd
                state: restarted

            - name: restart sshd
              ansible.builtin.service:
                name: sshd
                state: restarted
```

B.3.2.2. tasks/

B.3.2.2.1 main.yml

```
        - include_tasks: install_packages.yml
        - include_tasks: configure_snmp.yml
        - include_tasks: secure_system.yml
        - include_tasks: restart_server.yml
```

B.3.2.2.2 install_packages.yml

```
        - name: Instalar los paquetes necesarios
          ansible.builtin.package:
            name: "{{ packages }}"
            state: present
          become: yes
```

B.3.2.2.3 configure_snmp.yml

```
        - name: Anadir archivo de configuracion SNMP
          ansible.builtin.copy:
            content: |
              # Archivo basico de configuracion de snmpd

              # Escuchar en todas las interfaces de red en el puerto 161
              agentAddress udp:161

              # Configuracion de la comunidad SNMP v2c "centreon" con acceso de solo
                  lectura
              rwcommunity centreon

              sysDescr {{ entorno }}
            dest: /etc/snmp/snmpd.conf
          notify: restart snmpd

        - name: Habilitar servicio snmpd
          ansible.builtin.service:
            name: snmpd
            enabled: yes
```

B.3.2.2.4 secure_system.yml:

```
            - name: Configurar SELinux en modo enforcing
              ansible.posix.selinux:
```

```
3                 policy: targeted
4                 state: enforcing
5
6             - name: Deshabilitar acceso root por SSH
7               ansible.builtin.lineinfile:
8                 path: /etc/ssh/sshd_config
9                 regexp: '^PermitRootLogin'
10                line: 'PermitRootLogin no'
11                backup: yes
12              notify:
13                - restart sshd
```

B.3.2.2.5 reboot_server.yml

```
1             - name: Reiniciar el servidor
2               ansible.builtin.reboot:
3                 reboot_timeout: 300
4                 msg: "Reiniciando el servidor"
```

B.3.2.3. vars/

B.3.2.3.1 main.yml

```
1             packages:
2               - net-snmp
3               - net-snmp-utils
```

Bibliografía

[1] Wikipedia. Computación centralizada. Online. [Online]. Available: https://es.wikipedia.org/wiki/Computaci%C3%B3n_centralizada

[2] T. E.Beach. Computer concepts and terminology. Online. [Online]. Available: http://www.unm.edu/~tbeach/terms/types.html

[3] U. de Oviedo. Cliente-servidor. Online. [Online]. Available: http://isa.uniovi.es/domotica/Temas/T6/T6-ClienteServidor.htm

[4] Wikipedia. Supercomputadora. Online. [Online]. Available: https://es.wikipedia.org/wiki/Supercomputadora

[5] V. Fulber-Garcia. Centralized computing vs. distributed computing. Online. [Online]. Available: https://www.baeldung.com/cs/centralized-vs-distributed-computing.

[6] Ionos. ¿qué es la computación distribuida? Online. [Online]. Available: https://www.ionos.es/digitalguide/servidores/know-how/que-es-la-computacion-distribuida/

[7] Atlasian. ¿qué es un sistema distribuido? Online. [Online]. Available: https://www.atlassian.com/es/microservices/microservices-architecture/distributed-architecture#:~:text=Un%20sistema%20de%20computaci%C3%B3n%20centralizado,entre%20los%20nodos%20del%20sistema.

[8] dokumen. Linea del tiempo 1960-2013. Online. [Online]. Available: https://dokumen.tips/documents/linea-del-tiempo-1960-2013.html?page=6

[9] UPM. El origen de internet. el camino hacia la red de redes. Online. [Online]. Available: https://oa.upm.es/22577/1/PFC_IVAN_NEBREDA_RODRIGO.pdf

[10] Wikipedia. Dfs. Online. [Online]. Available: https://www-cohesity-com/glossary/distributed-file-system

[11] ——. Computación distribuida. Online. [Online]. Available: https://es.wikipedia.org/wiki/ComputaciÃşn_distribuida

[12] Amazon. Computación distribuida. Online. [Online]. Available: https://aws.amazon.com/es/what-is/distributed-computing/

[13] pispos. Arquitectura cliente-servidor. Online. [Online]. Available: https://www.pispos.co/arquitectura-cliente-servidor

[14] ——. Capas y niveles. Online. [Online]. Available: https://www.disrupciontecnologica.com/capas-y-niveles-diseno-y-confusion

[15] Wikipediia. Peer to peer. Online. [Online]. Available: https://es.wikipedia.org/wiki/Peer-to-peer

[16] Binance. Peer to peer. Online. [Online]. Available: https://academy.binance.com/es/articles/peer-to-peer-networks-explained

[17] M. M. Digital. Redes p2p. Online. [Online]. Available: https://mastermarketingdigital.es/p2p-arquitectura-red-peer-to-peer/

[18] R. Hat. Soa. Online. [Online]. Available: https://www.redhat.com/es/topics/cloud-native-apps/what-is-service-oriented-architecture

[19] Amazon. Soa. Online. [Online]. Available: https://aws.amazon.com/es/what-is/service-oriented-architecture/#:~:text=En%20la%20arquitectura%20orientada%20a%20servicios%20(SOA)%2C%20los%20servicios,tarea%20y%20edevuelve%20una%20respuesta.

[20] Wikipedia. Computación en la nube. Online. [Online]. Available: https://es.wikipedia.org/wiki/ComputaciÃşn_en_la_nube

[21] R. Hat. Computación en la nube. Online. [Online]. Available: https://www.redhat.com/es/topics/cloud

[22] Keepcoding. Monitorizacion. Online. [Online]. Available: https://keepcoding.io/blog/que-es-la-monitorizacion-de-sistemas/

[23] Pandora. Herramientas monitorización. Online. [Online]. Available: https://pandorafms.com/blog/es/herramientas-de-monitoreo-de-redes/

[24] Nagios. Nagios. Online. [Online]. Available: https://assets.nagios.com/downloads/nagioscore/docs/nagioscore/4/en/about.html#whatis?

[25] Wikipedia. Snmp. Online. [Online]. Available: https://es.wikipedia.org/wiki/Protocolo_simple_de_administraciÃşn_de_red

[26] Nagios. Guia nagios. Online. [Online]. Available: https://assets.nagios.com/training/selfpaced/materials/Nagios_StartUp_Guide.pdf

[27] ——. Nagios core. Online. [Online]. Available: https://confluence.atlassian.com/adminjiraserver/setting-your-jira-application-home-directory-938847747.html

[28] Zabbix. Zabbix. Online. [Online]. Available: https://www.zabbix.com/documentation/current/es/manual/introduction/about

[29] Centron. Centreon. Online. [Online]. Available: https://docs.centreon.com/docs/getting-started/platform/

[30] ——. Autodiscovery. Online. [Online]. Available: https://docs.centreon.com/docs/monitoring/discovery/introduction/

[31] Wikipedia. Centreon. Online. [Online]. Available: https://es.wikipedia.org/wiki/Centreon

[32] Santander. Metodología de trabajo. Online. [Online]. Available: https://blog.becas-santander.com/es/metodologias-desarrollo-software.html

[33] Wikipedia. Modelo de prototipos. Online. [Online]. Available: https://es.wikipedia.org/wiki/Modelo_de_prototipos

[34] J. de Andalucía. Requisitos. Online. [Online]. Available: http://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/407

[35] Pandora. Plan de monitorizacion. Online. [Online]. Available: https://pandorafms.com/blog/es/monitorizacion-de-sistemas/

[36] R. Hat. Idm. Online. [Online]. Available: https://access.redhat.com/products/identity-management/

[37] Atlassian. Jira. Online. [Online]. Available: https://www.atlassian.com/es/software/jira/download-journey

[38] RedHat. Ansible automation platform. Online. [Online]. Available: https://www.redhat.com/en/technologies/management/ansible

[39] Pandora. Monitorización. Online. [Online]. Available: https://pandorafms.com/blo:g/es/monitorizacion-de-sistemas/

[40] Comparison of top 7 linux distributions. Online. [Online]. Available: https://www.threehosts.com/ratings/comparison-software/linux-vs-ubuntu-vs-centos-fedora-vs-debian-vs-red-hat-vs-open-suse-vs-mint.html

[41] Centos. Centos stream. Online. [Online]. Available: https://www.centos.org/centos-stream/#tab-3

[42] R. Hat. Documentación idm. Online. [Online]. Available: https://access.redhat.com/documentation/es-es/red_hat_enterprise_linux/7/html/linux_domain_identity_authentication_and_policy_guide/introduction

[43] RedHat. Ansible. Online. [Online]. Available: https://www.redhat.com/es/topics/automation/learning-ansible-tutorial

[44] atareado. Inventario ansible. Online. [Online]. Available: https://atareao.es/tutorial/ansible/el-inventario-de-ansible/

[45] R. Hat. Ansible automation platform. Online. [Online]. Available: https://www.redhat.com/es/topics/

[46] Ionos. Apache. Online. [Online]. Available: https://www.ionos.es/digitalguide/servidores/know-how/que-es-apache/

[47] Haproxy. Haproxy. Online. [Online]. Available: http://docs.haproxy.org/2.9/intro.html

[48] Centreon. Arquitectura centreon. Online. [Online]. Available: https://docs.centreon.com/docs/installation/architectures/#what-kind-of-architecture-do-you-need

[49] ——. Modulo de autodescubrimiento. Online. [Online]. Available: https://docs.centreon.com/docs/monitoring/pluginpacks/

[50] ——. Prueba gratuita centreon. Online. [Online]. Available: https://www.centreon.com/free-trial/

[51] Atlassian. Configuración jira. Online. [Online]. Available: https://confluence.atlassian.com/adminjiraserver/setting-your-jira-application-home-directory-938847747.html

[52] R. Hat. Instalación aap. Online. [Online]. Available: https://access.redhat.com/documentation/es-es/red_hat_ansible_automation_platform/2.0-ea/html/red_hat_ansible_automation_platform_installation_guide/single-machine-scenario

[53] RedHat. Portal rh. Online. [Online]. Available: https://www.redhat.com/en/technologies/management/ansible

[54] Centreon. Instalación centreon. Online. [Online]. Available: https://download.centreon.com

[55] R. P. y. L. P. Luis Miguel Jiménez, *Sistemas distribuidos: Arquitectura y aplicaciones*. Luis Miguel Jiménez, Rafael Puerto y Luis Payá, 2017. [Online]. Available: https://books.google.es/books?hl=es&lr=&id=2W41DwAAQBAJ&oi=fnd&pg=PR9&dq=sistemas+distribuidos&ots=9kHulv2mXc&sig=QXhw8XHRUJGGy7_TFhRoB9sVhIM#v=onepage&q=sistemas%20distribuidos&f=false

[56] Microsoft. El modelo para sistemas distribuidos. Online. [Online]. Available: https://learn.microsoft.com/es-es/windows/win32/rpc/the-model-for-distributed-systems

[57] P. Ponsico Martin, "Tecnología de contenedores docker," B.S. thesis, Universitat Politècnica de Catalunya, 2017.

[58] itdo. Capas y niveles. Online. [Online]. Available: https://www.itdo.com/blog/como-es-una-arquitectura-de-microservicios-y-sistemas-distribuidos/

[59] R. Hat. Rhel9. Online. [Online]. Available: https://www.redhat.com/en/technologies/linux-platforms/enterprise-linux?sc_cid=7013a000002gqFrAAI&gad_source=1&gclid=Cj0KCQiAsvWrBhC0ARIsAO4E6f8gFdhrRSLWnJGzr3AMkuO4qXrEjBm2OXCaUc79LcmXwcB&gclsrc=aw.ds

[60] ——. Portal rh. Online. [Online]. Available: https://sso.redhat.com/auth/realms/redhat-external/login-actions/registration?client_id=rhcom&tab_id=I5eX87BA2zg

[61] ——. Rhel for arm64. Online. [Online]. Available: https://www.redhat.com/en/technologies/linux-platforms/enterprise-linux/server/trial

[62] RedHat. Ha cluster. Online. [Online]. Available: https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html/configuring_and_managing_high_availability_clusters/index