CESUR
Tu Centro Oficial de FP

Proyecto final ASIR

BitCLD

Nombre del alumno/a: Ittai Rivero Mederos

Curso académico: 2 CFGS ASIR

Centro educativo: Cesur Plaza Elíptica

Nombre del tutor del proyecto: Gonzalo Pérez Crespo

Índice

1. Resumen

1.1 Breve descripción del proyecto

Nuestro proyecto consiste en una infraestructura de servicios auto alojados ofrecidos como servicios en la nube. BitCLD, mediante contenedores docker ofrece servicios de productividad (almacenamiento, gestor de contraseñas, servidor DNS, gestor de contenedores), inteligencia artificial (mediante API y modelos locales), monitorización (tanto de servicios como del propio servidor) y seguridad perimetral (gracias al reverse proxy). También hemos añadido una VPN para un acceso remoto seguro a la terminal de sistema y como backup para acceder a los servicios y la gestión de un correo profesional con nuestro dominio personalizado. Con este proyecto no necesitamos abrir puertos del servidor a internet, ofreciéndonos aspectos de ciberseguridad y buenas prácticas.

1.2 Tecnologías utilizadas

Las tecnologías que hemos usado para poder llevar a cabo el proyecto son las siguientes:

Tecnología	Descripción
 Ubuntu Server	En su versión 24.04.2 como sistema operativo de nuestro servidor
 Docker y docker compose	Para ofrecer nuestros servicios mediante contenedores
 Tailscale	Como solución VPN

 Cloudflare	Como gestor de DNS, Cloudflared Tunnel, Cloudflare Access, gestor de dominio y subdominios, Cloudflare Pages
 Caddy server	Como reverse proxy
 Dodge	Como gestor de contenedores
 Adguard Home	Como solución de filtrado DNS para mejorar la privacidad y seguridad.
 Nextcloud	Como un sistema de almacenamiento seguro y privado en la nube
 Inteligencia artificial	Implementar inteligencia artificial tanto en local como mediante APIs externas
 Vaultwarden	Como gestor de contraseñas seguro y accesible

 Uptime Kuma	Para la monitorización y alertas de servicios mediante Telegram
 Netdata	Para monitorizar en tiempo real nuestro mini servidor
 Zoho mail	Para la creación de un correo profesional usando nuestro dominio personalizado
 GitHub y Github pages	Para tener una copia de seguridad del proyecto y para desplegar la página web de bitCLD. https://bitcld.com/

1.3 Resultados obtenidos

Tras finalizar el proyecto disponemos de una infraestructura completa y un gran aprendizaje sobre redes, sistemas y ciberseguridad. Hemos podido aprender acerca del despliegue de aplicaciones en contenedores y su gestión. El aprendizaje y uso de una plataforma tan avanzada como lo es Cloudflare, donde a parte de gestionar el dominio, también hemos trabajados con redes mediantes los túneles y ciberseguridad con la autorización de usuarios mediante Access.

También hemos podido experimentar con la inteligencia artificial tanto en modelos locales (en los cuáles no tuvimos muy buenos resultados debido a las limitaciones de hardware de nuestro servidor) y mediante una API gratuita de Groq, con la que obtuvimos mejores resultados ya que no dependemos del procesamiento de nuestra máquina.

Hemos tenido la oportunidad de trabajar también, relacionado con la seguridad, con una VPN principalmente para acceso remoto a la consola de comandos de nuestro sistema. Y finalmente también gracias al desarrollo de nuestra web principal hemos tenido la oportunidad de trabajar con tecnologías como GitHub y GitHub Pages. Adicionalmente, para poder tocar todas las opciones posibles, la documentación versión web del proyecto usamos Markdown, Material y Cloudflare Pages.

En definitiva, hemos trabajado con diversas tecnologías y aprendido enormemente gracias al proyecto y como resultado hemos obtenido un ecosistema de herramientas que nos ofrecen un valor real con posibilidad de usarlo en nuestro día a día.

2. Introducción

2.1 Contexto que motiva el proyecto

BitCLD nació como parte del proyecto final del Ciclo Superior de Administración de Sistemas Informáticos en Red con especialización en Ciberseguridad que estamos terminando y no sólo como un proyecto sino como un ecosistema que pudiera sernos de utilidad más allá de aprender y presentarlo.

Es por ello que con esa filosofía se unió con la idea principal de demostrar que podemos recuperar el control de datos y servicios sin depender de otras empresas. En un mundo cada vez más dependiente de grandes tecnológicas estas nos ofrecen comodidad, alta disponibilidad y seguridad a un bajo precio o incluso gratis, sin embargo, salvo en ciertas excepciones, esto suele implicar un enorme sacrificio en nuestra privacidad y sin privacidad no existe libertad.

Cada componente de bitCLD fue escogido cuidadosamente para ofrecer la mejor y más completa experiencia posible. Los pilares en los que basamos el proyecto son:

Simple: el ecosistema debía ser lo más sencillo posible, fácil de mantener y escalar.

Seguro: gracias a no abrir ningún puerto al exterior, al control de usuarios y a la comunicación cifrada.

Autoalojado: salvo una excepción que hicimos para probar otra tecnología extra, todos los servicios se encuentran en nuestro servidor, lo que nos garantiza privacidad y control sobre nuestros datos.

Código abierto: priorizando siempre que sea posible pero no limitado a soluciones open source las cuales son libres y auditables. Evitando lo más posible el software propietario el cuál no es auditable por la comunidad.

2.2 Breve historia del proyecto

Cuando estuvimos en la búsqueda del nombre del proyecto se plantearon varias opciones e ideas. Sin embargo, debido a las características del proyecto se optó por *bitCLD*. El término *bit* hace referencia a la unidad mínima de información en los sistemas informáticos y *CLD* es la abreviatura de cloud, nube en inglés. Por lo que el nombre nos transmite la idea de crear una nube con recursos muy limitados, el cual es nuestro caso.

BitCLD nos demuestra que no es necesario desembolsar una gran fortuna para crear una nube propia gestionada por nosotros y que no dependa de terceros.

2.3 Objetivos generales y específicos

2.3.1. Objetivos generales

Estos son los objetivos generales que nos hemos planteado para nuestro proyecto final.

Servir como proyecto final de ASIR

Principalmente debe servirnos como proyecto final de ASIR por lo que debemos tocar todos los módulos posibles para demostrar nuestro conocimiento en todas las áreas que podamos como redes, seguridad, virtualización, gestión de dominios… Además, también debe lucir lo más profesional posible para poder añadirlo a nuestro CV.

Crear un ecosistema lo más completo y seguro posible

Para ello creamos deberemos una infraestructura, basada principalmente en contenedores Docker, que nos ofrezcan servicios de distintos tipos pero que a su vez la hagamos de la manera más segura posible.

Crear un entorno seguro

Buscamos también que bitCLD pueda ser un entorno seguro para tener un lugar donde poder trabajar cómodamente para hacer todas las pruebas que queramos al estilo un laboratorio personal para desplegar servicios.

Crear una arquitectura escalable

Nuestro ecosistema debe tener la oportunidad de poder ser fácilmente ampliado para poder seguir aprendiendo y creciendo profesionalmente.

2.3.2 Objetivos específicos

Para lograr los objetivos generales hemos establecido estos objetivos específicos.

Configuración de servidores

Implementaremos un servidor físico con Ubuntu Server haciendo todas las configuraciones necesarias para que esté lo más optimizado y seguro posible.

Gestión de contenedores

Para poder hacerlo lo más escalable posible desplegaremos nuestros servicios, siempre que sea posible, desde contenedores docker compose gracias a la escalabilidad que nos ofrece y a la gestión tan fácil y cómoda que tenemos con Dockge.

Seguridad con Cloudflare

Para que podamos acceder de manera segura a nuestros servicios de manera segura sin abrir puertos gracias a Cloudflare Tunnel. Además Cloudflare también se encarga de

gestionar nuestro dominio y subdominios, de autenticar usuarios mediante Access y proteger nuestras aplicaciones y página web.

Proxy inverso

Para una mayor seguridad usaremos Caddy server como proxy inverso en nuestra red local con certificados SSL y para que nuestros servicios sean más cómodos de buscar en nuestra red LAN.

Acceso a una VPN

Usaremos Tailscale para poder trabajar con una red privada virtual y para usarla como un acceso externo secundario a nuestro sistema en caso de necesitar acceder a la terminal en algún momento.

Monitorización de servicios

Implementaremos Uptime kuma para monitorizar nuestros servicios y para que podamos recibir alertas en caso de algún imprevisto.

Almacenamiento en la nube

Añadiremos Nextcloud para usarlo como sistema de almacenamiento privado y seguro en la nube, además de notas y calendario. Debido a las características de nuestro proyecto una herramienta como esta era necesaria.

Gestión de contraseñas

Como gestor de contraseñas seguras usaremos Vaultwarden para poder tener todas las contraseñas que necesitemos a buen recaudo y también para tener buenas prácticas ya que lo más seguro es nunca repetir contraseña.

Filtrado de DNS

Usaremos Adguard Home como nuestro servidor de DNS filtrado para nuestra red. Además, lo personalizaremos mediante listas de bloqueo previamente seleccionadas que nos bloquean tanto los anuncios y el malware.

Inteligencia artificial

Debido a la importancia de esta, trabajaremos con la inteligencia artificial tanto de manera local en nuestro servidor como mediante API externa.

Monitorización de rendimiento

Usarmos Netdata para monitorizar en tiempo real nuestro sistema, y en caso de ser necesario recibir notificaciones de cualquier problema que pueda haber en nuestro equipo

Gestión de código

Usaremos GitHub para poder gestionar la página web y nuestra documentación profesional con Mkdocs y material y desplegado mediante Cloudflare Pages. Además de ser públicos para que cualquiera pueda verlo y usarlo.

Crear una imagen profesional

Intentamos crear una identidad propia y profesional al proyecto creando un logo y usando una paleta de colores coherente en todo momento. Además, de gestionar nuestro propio dominio bitcld.com para que sea el eje principal del proyecto, alojando tanto la web como las aplicaciones y documentación en él.

Cumplir objetivos académicos

Buscamos poder completar mi ciclo formativo de grado superior y demostrar nuestros conocimientos en todos los módulos posibles de ASIR como Implantación de Sistemas Operativos, Administración de Sistemas Operativos, Planificación y Administración de Redes, Seguridad y Alta Disponibilidad, Servicios de Red e Internet… Además de presentar una documentación completa con la que pueda demostrar que tengo esos conocimientos.

Cumplir objetivos profesionales

A nivel profesional buscábamos crear un proyecto lo más completo posible para demostrar nuestras capacidades a posibles empleadores ya que un proyecto profesional y completo puede abrir puertas.

2.4 Relación de los módulos cursados con el proyecto

BitCLD fue pensado para que pudiéramos aplicar de la forma más práctica posible todos los conocimientos adquiridos en el ciclo tocando y profundizando en el mayor número de módulos posibles. Además también buscábamos que fuera un proyecto que pudiera tener una utilidad real una vez finalizado el curso.

A continuación mostramos cada tecnología con su relación correspondiente a los principales módulos que se ha trabajado en cada una bajo nuestra opinión:

Servidor Ubuntu

- ➔ Implantación de Sistemas Operativos (ISO) → Debido a la instalación, la configuración y la administración del sistema operativo de nuestro equipo.
- ➔ Planificación y Administración de Redes (PAR) → Ya que debemos asignar una IP fija y gestionar interfaces y rutas.
- ➔ Administración de Sistemas Operativos (ASO) → Por el tema de gestionar los usuarios y permisos.
- ➔ Fundamentos de Hardware (FH) → Debido a que nuestro servidor está en físico en nuestra casa hemos necesitado configurarlo y hacerle mantenimiento al minipc.

Docker / Dockge

- ➔ Implantación de Aplicaciones Web (IAW) → Debido a que docker despliega nuestros servicios mediante contenedores.

- ➔ Administración de Sistemas Operativos (ASO) → Gracias a dockge tenemos automatizado el despliegue y administración de los entornos virtualizados.
- ➔ Planificación y Administración de Redes (PAR) → Ya que debimos crear redes internas para que los contenedores se pudieran comunicar entre ellos.

Cloudflare, Cloudflared tunnel, Access y Pages

- ➔ Servicios de Red e Internet (SRI) → Gracias a Cloudflared tunnel podemos publicar nuestros servicios internos sin exponer puertos. Además de que Cloudflare también resuelve nombres y configura los dominios
- ➔ Seguridad y Alta Disponibilidad (SAD) → Nuestros túneles están cifrados a través de Cloudflare tunnel y se realiza la autenticación de usuarios mediante Zero Trust Access.
- ➔ Planificación y Administración de Redes (PAR) → No ofrece también enrutamiento seguro entre nuestra red local y la externa.
- ➔ Ciberseguridad (CIBER) → Adicionalmente obtenemos protección frente a ataques DDoS y spoofing DNS

Caddy server

- ➔ Servicios de Red e Internet (SRI) → Debido al uso del proxy inverso.
- ➔ Planificación y Administración de Redes (PAR) → Ya que gracias a caddy gestionamos el tráfico interno con nuestro dominio local `.lan` y por la creación de certificados.
- ➔ Seguridad y Alta Disponibilidad (SAD) → Nos permite tener una encriptación TLS, aislar nuestros servicios y gestionar el acceso local de forma segura.

Tailscale

- ➔ Planificación y Administración de Redes (PAR) → Ya que creamos una red privada virtual entre nuestro servidor y nuestros dispositivos.
- ➔ Seguridad y Alta Disponibilidad (SAD) → La VPN nos permite tener un cifrado de extremo a extremo y controlar el acceso de forma remota a nuestro servidor.
- ➔ Administración de Sistemas Operativos (ASO) → Ya que Tailcale la integramos con nuestro servidor y podemos hacer una conexión remota por SSH.

Uptime Kuma

- ➔ Seguridad y Alta Disponibilidad (SAD) → Ya que se encarga de monitorizar la salud de nuestros servicios y su disponibilidad. Además de mandarnos notificaciones.

➔ Administración de Sistemas Operativos (ASO) → Otra función que tiene es que nos permite tener una respuesta ante caídas o errores como reiniciar o parar los servicios o el servidor.

Nextcloud

➔ Ciberseguridad (CIBER) → Creamos una nube privada donde tenemos control total de nuestros datos
➔ Seguridad y Alta Disponibilidad (SAD) → Debido a que controlamos el acceso, y tenemos la opción de gestionar usuarios y permisos.

Vaultwarden

➔ Seguridad y Alta Disponibilidad (SAD) → Debido a que no dejamos un puerto expuesto todo el tráfico está cifrado extremo a extremo y podemos controlar accesos y autenticar a usuarios.
➔ Ciberseguridad (CIBER) → Es esencial como buenas prácticas en ciberseguridad no utilizar la misma contraseña en todas partes.

Adguard Home

➔ Planificación y Administración de Redes (PAR) → Ya que configuramos un servidor DNS interno.
➔ Seguridad y Alta Disponibilidad (SAD) y ciberseguridad (CIBER) → Gracias a nuestras listas personalizadas podemos filtrar el tráfico, bloquear la publicidad y protegernos de rastreos de cookies.

/ 9 Inteligencia artificial (Local / API)

➔ Implantación de Aplicaciones Web (IAW) → Debido a que integramos servicios web con nuestra interfaz y con las APIs.
➔ Administración de Sistemas Operativos (ASO) → Para evitar un exceso de consumo de recursos debimos optimizar nuestros recursos.

Zoho mail

➔ Servicios de Red e Internet (SRI) → Gracias a que configuramos un servicio de correo con nuestro dominio propio.
➔ Seguridad y Alta Disponibilidad (SAD) → Al utilizar un servidor externo nos protege frente al spam y nos permite una autenticación segura.

Netdata

- ➔ Seguridad y Alta Disponibilidad (SAD) → No permite monitorizar en tiempo real el rendimiento de nuestro servidor. Además de mandarnos notificaciones.
- ➔ Administración de Sistemas Operativos (ASO) → Gracias a Netdata podemos detectar cuellos de botella y gestionar mejor los procesos.

GitHub y Github pages

- ➔ Implantación de Aplicaciones Web (IAW) → Ya que nos permite tener un control de las versiones de nuestro proyecto, alojar nuestra documentación y hospedar la web del proyecto.
- ➔ Empresa e Iniciativa Emprendedora (EIE) → Gracias a GitHub nos permite tener una presentación mucho más profesional del proyecto y nos ayuda a mejorar nuestra imagen profesional.

3. Diseño del ecosistema

Como nos muestra este diagrama podemos apreciar el funcionamiento de bitCLD. Tendríamos los servicios, que ocupan la columna central y según cómo accedamos a cada servicio el flujo de trabajo cambia.

Si usamos nuestros servicios en nuestra red local accederemos mediante nuestros dominios `.lan`, haciendo uso del reverse proxy (caddy), que nos genera los certificados y contiene las rutas a los dominios internos en el CaddyFile y nuestro servidor DNS (Adguard Home) que se encargará de indicar a cualquier dispositivo que lo como DNS redirigirá las rutas de nuestros dominios .lan a cada servicio gracias a los rewrites.

Sin embargo, si accedemos mediante nuestro dominio personalizado `subdominio.bitcld.com` se usará el túnel que hemos creado para exponerlo a internet sin abrir puertos y Access se encargará de autenticar a los usuarios.

Este diseño nos garantiza seguridad en todo momento, ya sea en local como en internet ya que accederemos mediante `https://` en ambos casos cifrando nuestra conexión.

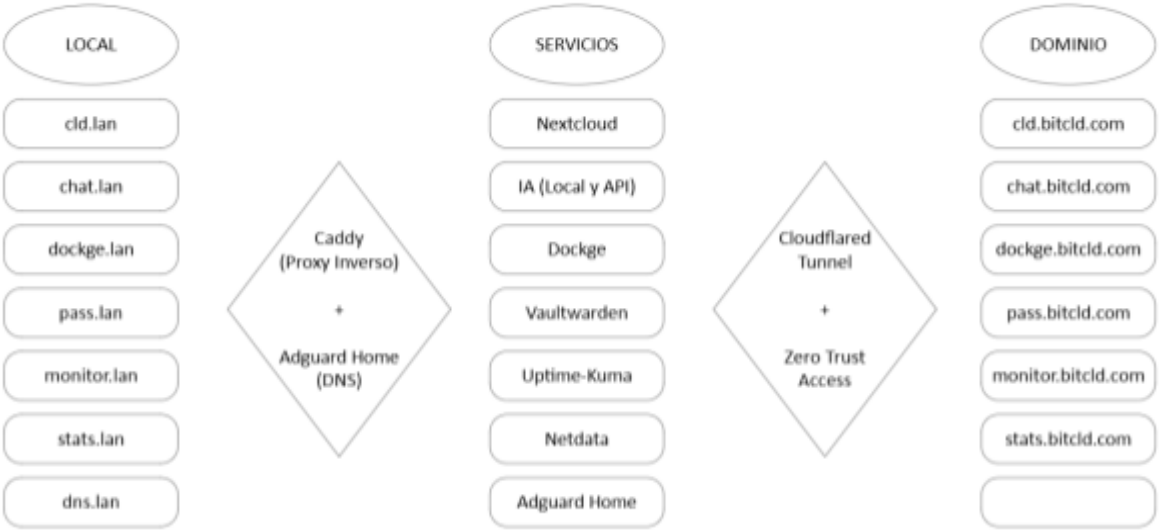


Figura 1: Diseño del ecosistema

4. Implementación

En este apartado hablaremos en más profundidad cada tecnología con la que hemos trabajado para poder hacer posible que bitCLD exista. En cada una pondremos una introducción del servicio o tecnología, una explicación del motivo de la elección y la función que cumple dentro del proyecto. Además, explicaremos con detalle la instalación y configuración de todo lo que hicimos, cómo sería el uso diario y para finalizar haremos una pequeña conclusión con los aspectos claves.

Comenzaremos con la preparación del servidor, lo cual incluye la instalación de Ubuntu Server y docker compose. Seguiremos con la configuración de red y seguridad perimetral donde se encuentran tecnologías las cuales son esenciales para poder acceder a nuestro ecosistema desde el exterior y en local de manera segura, luego continuaremos con cada uno de los distintos servicios que ofrece bitCLD. Para finalizar, añadiremos una tabla resumen explicativa de los distintos puertos, dominios y redes docker.

4.1. Preparación del servidor

4.1.1 Hardware y especificaciones

Introducción

Como servidor para nuestro proyecto hemos elegido el GMKtec NucBox G2. Este equipo es un minipc bastante pequeño con componentes modernos y sobretodo eficientes. Aunque según el fabricante está pensado para un uso de escritorio nosotros lo usaremos como nuestro servidor. Dispone de un procesador Intel N100 de 4 núcleos, memoria de 12 GB DDR5 y 512 GB de almacenamiento SSD.

Motivos de elección

Elegimos este minipc por delante de otras opciones por tres razones principales:

1. La primera y principal razón por la que lo escogimos fue su precio ya que tras una larga investigación para elegir el equipo este era la opción más potente disponible por menos de 100 €.
2. El segundo motivo fue por su bajo consumo energético, ya que al tratarse de un servidor, la idea es que lo tengamos operativo a diario o al menos durante un largo tiempo por lo que debía consumir muy poco para evitar gastos en electricidad y en sobrecalentamiento del dispositivo.
3. El último factor por el que nos decidimos por esta opción fue por su memoria RAM, ya que por esos precios sólo encontrábamos equipos con 8 GB de RAM y teníamos miedo de que no fueran suficientes.

Función dentro del proyecto

Este equipo actúa como el servidor principal de nuestro proyecto actualmente. Se encarga de desplegar y mantener iniciados todos nuestros servicios organizados en contenedores. Además, también funciona como nuestro enlace seguro al exterior gracias al túnel de Cloudflare quitándonos la necesidad de abrir puertos. Y finalmente, como gestor de inteligencia artificial ya que como no tenemos la potencia suficiente lo usamos como intermediario a través de nuestra API de Groq liberándonos carga de procesamiento.

Componentes

Nuestro servidor consiste en un mini pc GMKtec G2 que tiene las siguientes características:

Procesador	Intel de 12ª generación Alder Lake N100
RAM	12GB DDR5
Almacenamiento	512GB M.2
Puertos	x3 USB A 3.2, x2 HDMI, x1 Display Port, x2 RJ45
Almacenamiento extra	1 Tb HDD

Procesador Intel de 12ª generación Alder Lake N100

Es la mayor limitación que tenemos en nuestro servidor ya que para la mayoría de los servicios de los que disponemos vamos con bastante margen como lo es Caddy, Tailscale,

AdGuard Home, Vaultwarden… Pero en servicios como Nextcloud si que notamos un mayor uso del procesador pero es perfectamente usable. Sin embargo, sin lugar a dudas, donde más limitados estamos es con el tema de la inteligencia artificial cuando la corremos en local. De hecho inicialmente se tenía pensado usar IA sólo en local, pero debido a la imposibilidad de uso, ya que es extremadamente lento y consume demasiados recursos, optamos por añadir IA a través de una API para poder tener algo medianamente funcional.

Además, también se nos hace imposible virtualizar entornos como Proxmox, es por eso que optamos por trabajar directamente en Ubuntu Server con contenedores Docker.

Memoria RAM 12GB DDR5

En cuanto a RAM vamos bastante bien para cubrir las necesidades de nuestro proyecto actual. Al ser DDR5 nos permite una mayor fluidez entre el CPU y la memoria, por lo que la experiencia de uso nos es bastante positiva.

Sin embargo, el mayor inconveniente es que la RAM está soldada, lo que nos reduce enormemente la posibilidad de escalabilidad en el futuro. Aunque teniendo en cuenta el procesador del que disponemos es más probable que en caso de necesitar cambiar a otro equipo fuera por las limitaciones de la CPU y no de la RAM.

Almacenamiento 512GB M.2

En cuanto al almacenamiento principal, el que sea un M.2 es una gran ventaja para nosotros ya que nos permite tener una muy buena velocidad de arranque y que nos cargue los contenedores también bastante rápido.

La mayor desventaja después de hacer algunas investigaciones es la disipación térmica, es por ello que Netdata es tan importante para poder controlar la temperatura de nuestro disco y que nos envíe alertas si fuera necesario.

Conectividad y puertos

En cuanto a la conectividad dispone de WIFI 6, sin embargo, al usarlo como servidor, por estabilidad estará siempre conectado por cable. En cuanto a los puertos tenemos todo lo que necesitamos ya que disponemos de 3 puertos USB 3.0 para poder añadir hasta 3 discos duros externos.

Además, también disponemos de 2 puertos RJ45 lo cuál nos permitiría segmentar el tráfico de nuestro equipo, por ejemplo, un puerto para el almacenamiento y otro para los servicios. Sin embargo, en nuestro proyecto sólo usaremos un puerto para todo, pero más adelante podríamos mirarlo.

Los puertos de HDMI y Display, nos vinieron bien cuando tuvimos que hacer las primeras configuraciones ya que de fábrica nos vino con Windows 11 pro, pero una vez que hicimos la instalación de Ubuntu ya no los hemos vuelto a utilizar.

Disco duro externo 1 Tb HDD

Optamos por añadir un disco duro externo, para aumentar nuestra capacidad de almacenamiento principalmente para nuestro servicio de Nextcloud. El que dispongamos de 3 puertos USB 3.0 es una gran ventaja ya que podemos añadir más almacenamiento.

La principal desventaja es que es un disco duro por lo que la velocidad de lectura y escritura es lenta, y también no tenemos tanta fiabilidad comparado con un disco de estado sólido. Además, es un disco duro con algunos años, por lo que para nuestro proyecto es suficiente para demostrar de que puede llegar a ser capaz bitCLD y para aprender, pero es necesaria una mejora en el almacenamiento, para incluso añadir más adelante un sistema de Raids para asegurar nuestra información.

Conclusión

Hemos podido comprobar que elegir GMKtec NucBox G2 fue una buena decisión ya que ha cumplido con creces perfectamente para nuestro proyecto actual.

Gracias a sus componentes hemos tenido un buen equilibrio entre rendimiento y eficiencia energética, pudiendo correr sin problemas todos nuestros servicios, exceptuando la IA local. Además, el añadir un disco duro externo para el almacenamiento fue una buena decisión y también cumple muy bien como intermediario entre nuestros servicios y el exterior mediante cloudflare.

4.1.2 Ubuntu server

Introducción

Ubuntu es una distribución de GNU/Linux basada en Debian y es una de las distribuciones más estables, seguras y fáciles de administrar de linux y la versión server es la opción optimizada para entornos de servidor. Además, es de código abierto lo que proporciona un entorno fiable tanto a nivel doméstico como profesional.

Motivos de elección

Optamos por Ubuntu Server debido a que en su versión sin interfaz gráfica es muy ligera, lo que nos ayuda bastante debido a nuestros limitados recursos. Esta distro es una de las más usadas a nivel empresarial lo que nos permite disponer con una gran documentación y compatibilidad asegurada con nuestras herramientas y servicios. Además elegimos su versión LTS debido a que nos garantiza actualizaciones a largo plazo, ideal para el día a día.

Función dentro del proyecto

En bitCLD, Ubuntu Server es nuestro sistema operativo principal, por lo que se trata de la base en la que corren todos nuestros servicios. El papel de Ubuntu Server en nuestro proyecto, al igual que cualquier sistema operativo, es ofrecernos un entorno estable, seguro y eficiente para que podamos despreocuparnos del SO y enfocarnos en los servicios, contenedores y conexiones.

Instalación

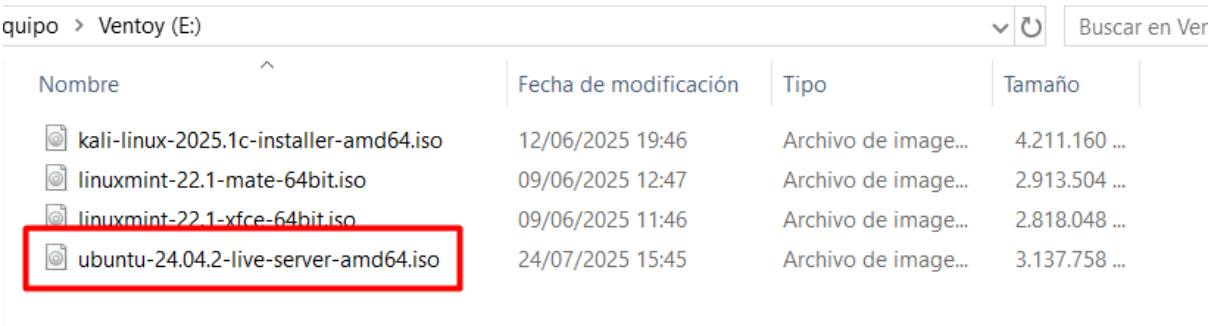
1. Descarga de la imagen ISO

Descargamos la versión Ubuntu Server LTS (por ejemplo, 24.04 LTS) desde la página oficial de Ubuntu:

https://ubuntu.com/download/server

2. Creación del medio de instalación

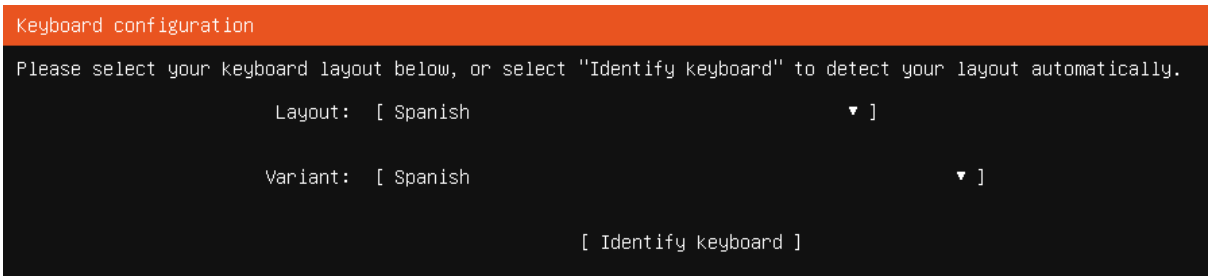
Quemamos la imagen en una memoria USB mediante herramientas como Rufus, Ventoy o BalenaEtcher, en nuestro caso usamos Ventoy.



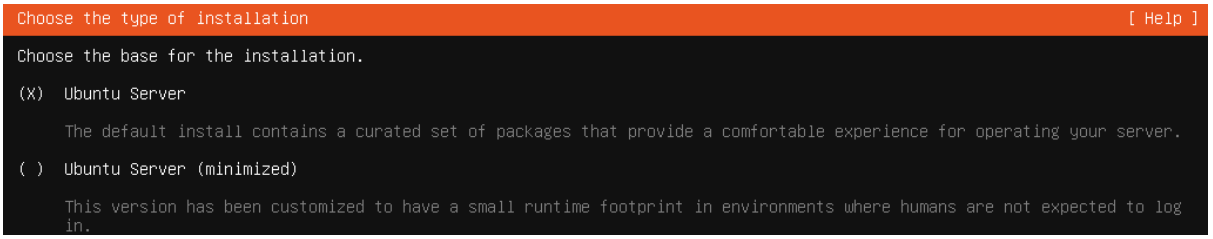
3. Instalación en el equipo

Iniciamos desde el USB e instalamos Ubuntu Server siguiendo los pasos del asistente:

➔ Seleccionamos idioma y disposición del teclado

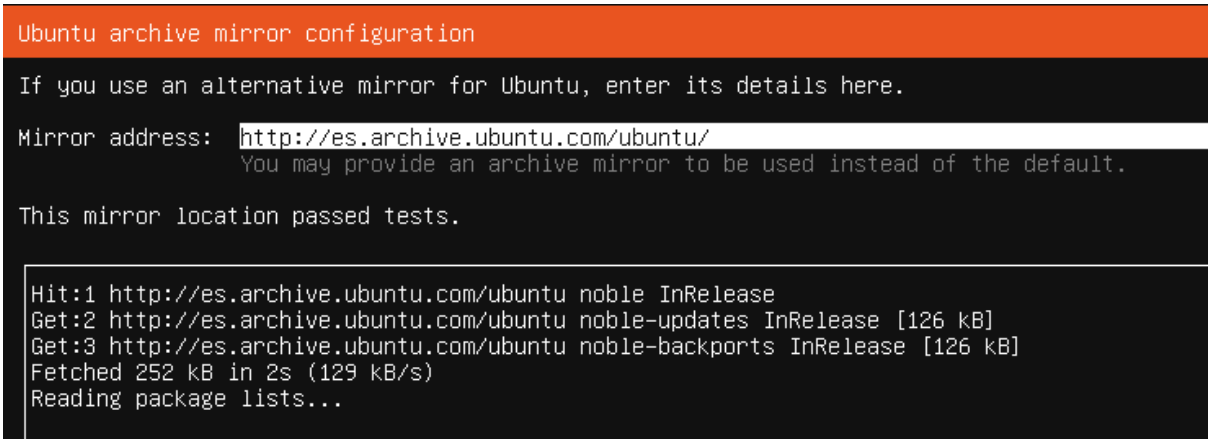


➔ Elegimos el tipo de instalación



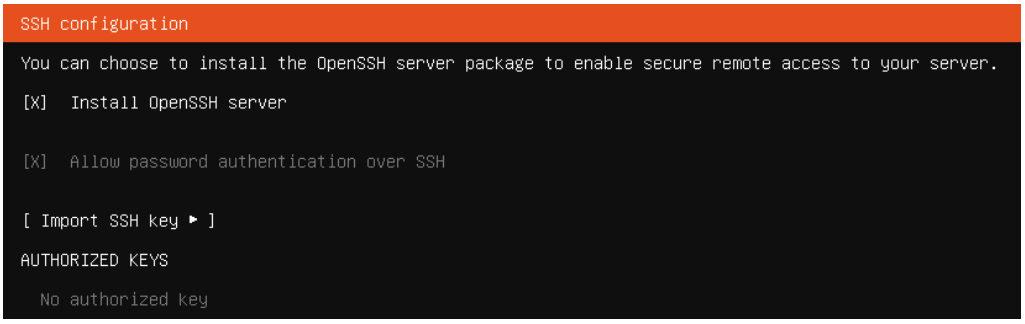
➔ Al estar por cable se conectará a nuestra red y nos saltamos el añadir un proxy

➔ Seleccionamos de donde queremos descargar los paquetes



➔ Creamos nuestro usuario root y le damos nombre al servidor

➔ Activamos la instalación de OpenSSH Server para acceso remoto.

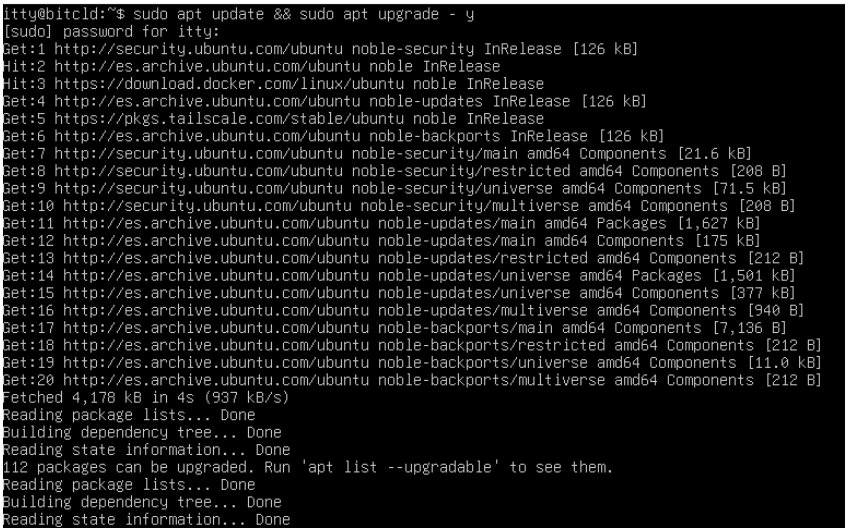


➔ Dejamos las demás opciones por defecto y finalizamos la instalación.

Primer inicio y actualización del sistema

Una vez completada la instalación, accedemos al servidor localmente o por SSH, en nuestro caso por comodidad siempre usaremos SSH y actualizamos el sistema:

```
sudo apt update && sudo apt upgrade -y
```

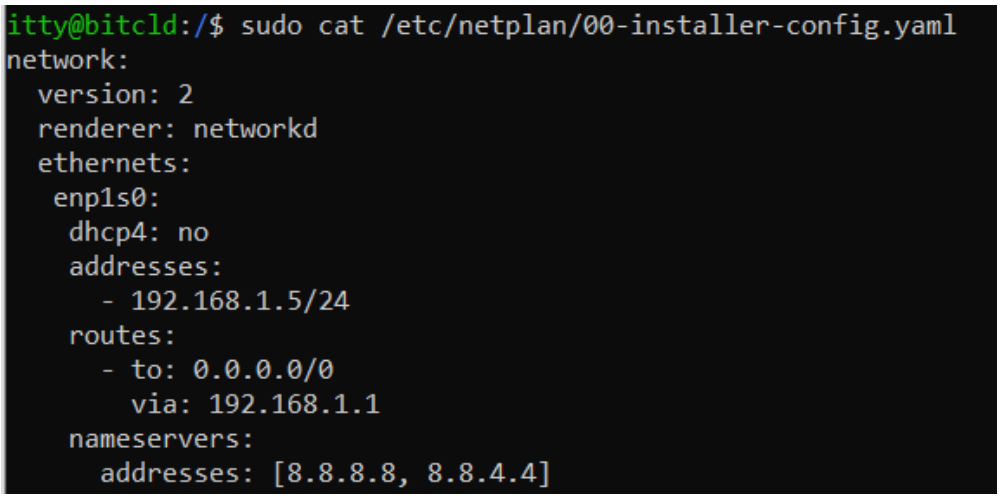


Configuración básica del entorno

1. Red local

Para establecer una IP estática, algo esencial para que todo pueda funcionar sin complicaciones, editamos el archivo de configuración de red y establecemos la ip fija en nuestro servidor:

```
sudo nano /etc/netplan/00-installer-config.yaml
```



	Descripción
renderer: networkd	Usamos Netplan con el renderer networkd, ideal para servidores sin entorno gráfico
ethernets: **enp1s0:**	Definimos la interfaz
dhcp4: no	Imprescindible para una IP estática.

Luego aplicamos los cambios:

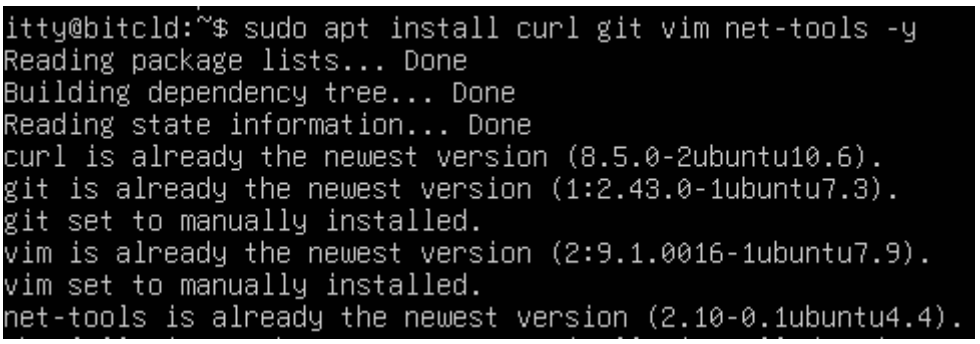
```
sudo netplan apply
```

Mantener una dirección estática es esencial para que todos los servicios del ecosistema puedan comunicarse de forma predecible. Además, simplifica la administración, las reglas de firewall y la resolución de nombres locales como los dominios .lan en el archivo de Caddyfile.

2.Instalar utilidades

Instalamos algunas utilidades base con:

```
sudo apt install curl git vim net-tools -y
```



Montaje de disco externo

Debido a que nuestro mini-pc tiene 415gb optamos por añadir un disco duro externo para ampliar el almacenamiento del sistema, identificado como dexterno, utilizado principalmente para el almacenamiento de Nextcloud. Para montar un disco deberemos de seguir los siguientes pasos:

1. Verificamos el dispositivo:

```
sudo fdisk -l
```

```
Disk /dev/sdb: 931.48 GiB, 1000170586112 bytes, 1953458176 sectors
Disk model: Elements 25A2
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: gpt
Disk identifier: 77132F63-FD98-11EA-BF72-40167E996018
```

y localizamos el disco que queremos montar.

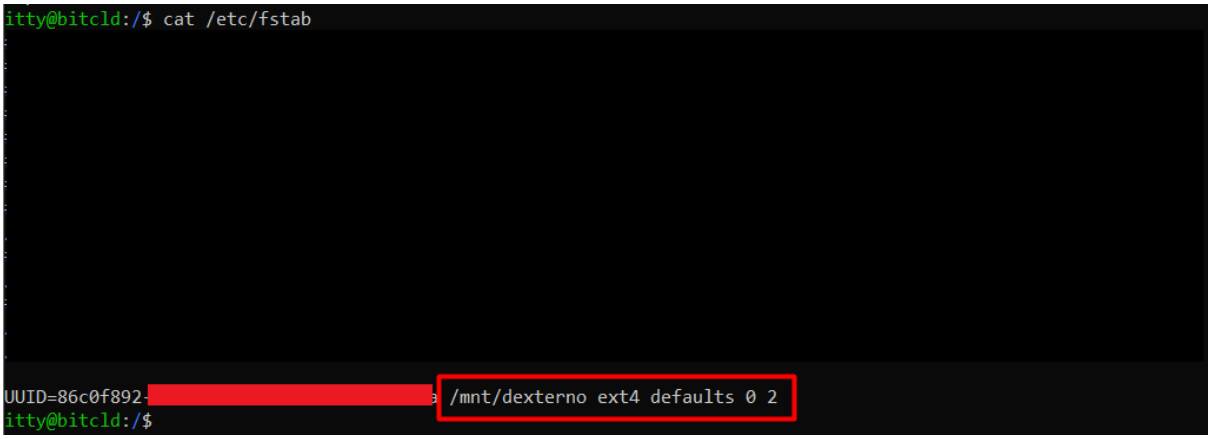
2. Creamos el punto de montaje:

```
sudo mount /dev/sda1 /mnt/dexterno
```

3. Para que el montaje sea automático al iniciar el sistema:

Editamos /etc/fstab y añadimos la siguiente línea:

```
/dev/sda1   /mnt/dexterno   ext4   defaults   0   2
```



4. Comprobación final:

Ejecutamos el siguiente comando para verificar que el disco dexterno esté correctamente montado y accesible.

```
df -h
```

```
itty@bitcld:/$ df -h
Filesystem                         Size  Used Avail Use% Mounted on
tmpfs                              1.2G  1.9M  1.2G   1% /run
efivarfs                           192K  104K   84K  56% /sys/firmware/efi/efivars
/dev/mapper/ubuntu--vg-ubuntu--lv   98G   34G   60G  37% /
tmpfs                              5.8G  260K  5.8G   1% /dev/shm
tmpfs                              5.0M     0  5.0M   0% /run/lock
/dev/sda2                          2.0G  193M  1.6G  11% /boot
/dev/sda1                          1.1G  6.2M  1.1G   1% /boot/efi
/dev/sdb1                          916G   45M  870G   1% /mnt/dexterno
tmpfs                              1.2G   12K  1.2G   1% /run/user/1000
```

Con este comando verificamos que el disco dexterno esté correctamente montado y accesible.

Estructura de carpetas del proyecto

Para dejar preparadas carpetas que gestionaremos a través de Dockge ejecutaremos:

```
sudo mkdir -p /srv/docker/stacks
sudo chown -R $USER:$USER /srv/docker
```

```
itty@bitcld:/srv/docker$ ls
dockge  stacks
itty@bitcld:/srv/docker$ cd stacks/
itty@bitcld:/srv/docker/stacks$ ls
adguard  caddy  ia  nextcloud  uptime-kuma  vaultwarden
itty@bitcld:/srv/docker/stacks$
```

Una vez creada la carpeta donde irán los servicios, Dockge se encargará de la gestión de las carpetas y archivos de cada servicio que necesitemos.

Ahora creamos la carpeta en el disco duro dexterno para nuestro almacenamiento adicional para Nextcloud:

```
sudo mkdir -p /mnt/dexterno/nextcloud/data
```

```
itty@bitcld:/mnt/dexterno/nextcloud$ ls
data
itty@bitcld:/mnt/dexterno/nextcloud$
```

Verificación final

Ahora para comprobar que todo lo que hemos realizado se ha configurado correctamente haremos las siguientes comprobaciones:

Para mostrar la información de nuestro sistema operativo usaremos:

lsb_release -a

```
itty@bitcld:/$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 24.04.3 LTS
Release:        24.04
Codename:       noble
```

Para comprobar la versión de docker descargada:

docker --version

```
itty@bitcld:/$ docker --version
Docker version 27.5.1, build 27.5.1-0ubuntu3~24.04.2
```

Con el siguiente comando nos aseguramos que la conexión remota funciona correctamente:

sudo systemctl status ssh

```
itty@bitcld:/$ sudo systemctl status ssh
[sudo] password for itty:
● ssh.service - OpenBSD Secure Shell server
     Loaded: loaded (/usr/lib/systemd/system/ssh.service; disabled; preset: enabled)
     Active: active (running) since Thu 2025-10-16 13:28:56 UTC; 22h ago
TriggeredBy: ● ssh.socket
       Docs: man:sshd(8)
             man:sshd_config(5)
   Main PID: 4608 (sshd)
      Tasks: 1 (limit: 13953)
     Memory: 4.9M (peak: 7.7M)
        CPU: 271ms
     CGroup: /system.slice/ssh.service
             └─4608 "sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startups"

Oct 16 13:28:56 bitcld sshd[4610]: Invalid user ittai from 192.168.1.39 port 55231
Oct 16 13:29:07 bitcld sshd[4610]: Connection reset by invalid user ittai 192.168.1.39 port 55231 [preauth]
Oct 16 13:29:25 bitcld sshd[5084]: Accepted password for itty from 192.168.1.39 port 55234 ssh2
Oct 16 13:29:25 bitcld sshd[5084]: pam_unix(sshd:session): session opened for user itty(uid=1000) by itty(uid=0)
Oct 16 14:48:45 bitcld sshd[45397]: Accepted password for itty from 100.107.78.122 port 56192 ssh2
Oct 16 14:48:45 bitcld sshd[45397]: pam_unix(sshd:session): session opened for user itty(uid=1000) by itty(uid=0)
Oct 17 09:45:38 bitcld sshd[578365]: Accepted password for itty from 192.168.1.39 port 65523 ssh2
Oct 17 09:45:38 bitcld sshd[578365]: pam_unix(sshd:session): session opened for user itty(uid=1000) by itty(uid=0)
Oct 17 09:53:36 bitcld sshd[582289]: Accepted password for itty from 192.168.1.39 port 49230 ssh2
Oct 17 09:53:36 bitcld sshd[582289]: pam_unix(sshd:session): session opened for user itty(uid=1000) by itty(uid=0)
```

Nos aseguramos que la IP de nuestro servidor está bien puesta:

ip -4 addr show enp1s0 | grep inet

```
itty@bitcld:/$ ip -4 addr show enp1s0 | grep inet
    inet 192.168.1.5/24 brd 192.168.1.255 scope global enp1s0
```

Y finalmente nos aseguramos que nuestro disco duro externo está bien montado:

df -h | grep dexterno

```
itty@bitcld:/$ df -h | grep dexterno
/dev/sdb1                       916G   45M  870G   1% /mnt/dexterno
```

Conclusión

Ubuntu Server nos proporciona una muy buena base sobre la que montar BitCLD. Es estable, seguro y compatible con una gran cantidad de herramientas lo que lo convierte en un muy buen sistema operativo para nosotros ya que suple todas nuestras necesidades en cuanto a SO.

El configurar una IP fija, en nuestro caso `192.168.1.5` simplifica enormemente la administración y el añadido del disco duro externo nos ofrece una gran capacidad y la posibilidad de despreocuparnos del almacenamiento.

4.1.3 Docker y docker Compose

Introducción

Docker es una plataforma de virtualización ligera que está basada en contenedores lo cuál es una gran ventaja ya que nos permite desplegar nuestros servicios de forma simple, aislada y escalable. Debido a que se basa en imágenes cada servicio se encarga de sus dependencias sin interferir en nuestro sistema base ni en otros contenedores. Docker compose, por otro lado, necesita de docker para funcionar ya que docker se encarga de ejecutar los contenedores y docker compose facilita la configuración y el arranque conjunto de varios contenedores relacionados y también se encarga de los puertos, volúmenes y redes.

Motivos de elección

Optamos por usar docker y docker compose ya que es bastante estable y muy adoptado con una gran comunidad detrás. Necesitábamos un entorno para desplegar y ejecutar nuestras aplicaciones y estas dos tecnologías satisfacen por completo nuestras necesidades. Nos ofrecen también soporte nativo en Linux, lo cuál era esencial para nosotros. Además, estas tecnologías están perfectamente alineadas con los pilares de nuestro proyecto ya que es algo simple, es seguro y lo podemos auto hospedar.

Función dentro del proyecto

Dentro de nuestro proyecto bitCLD, actúan como el entorno de ejecución principal de todos los servicios. Ya que cada aplicación se ejecuta en su propio contenedor, como iremos viendo más adelante en los distintos servicios que usamos y también permite mantener un sistema limpio, ordenado y fácil de administrar gracias a Dockge, el cual es uno de los servicios que tenemos en el proyecto y nos permite administrar todo aún más sencillo.

Instalación de Docker y Docker Compose

En bitCLD hemo optado por una instalación directa desde los repositorios oficiales de Ubuntu, buscando que sea de la manera más simple y estable. Con este método, tanto Docker como Docker Compose se instalan con un solo comando, sin necesidad de añadir repositorios externos.

1. Instalación de Docker y Docker Compose

Ejecutamos el siguiente script oficial para instalar tanto Docker como docker compose en su versión V2:

```
curl -fsSL https://get.docker.com | sudo sh
```

Antes de que ejecutemos este script debemos asegurarnos que no tengamos instalado Docker anteriormente porque podría rompernos configuraciones que ya tengamos. En nuestro caso al ser la primera vez que instalamos no hay problema.

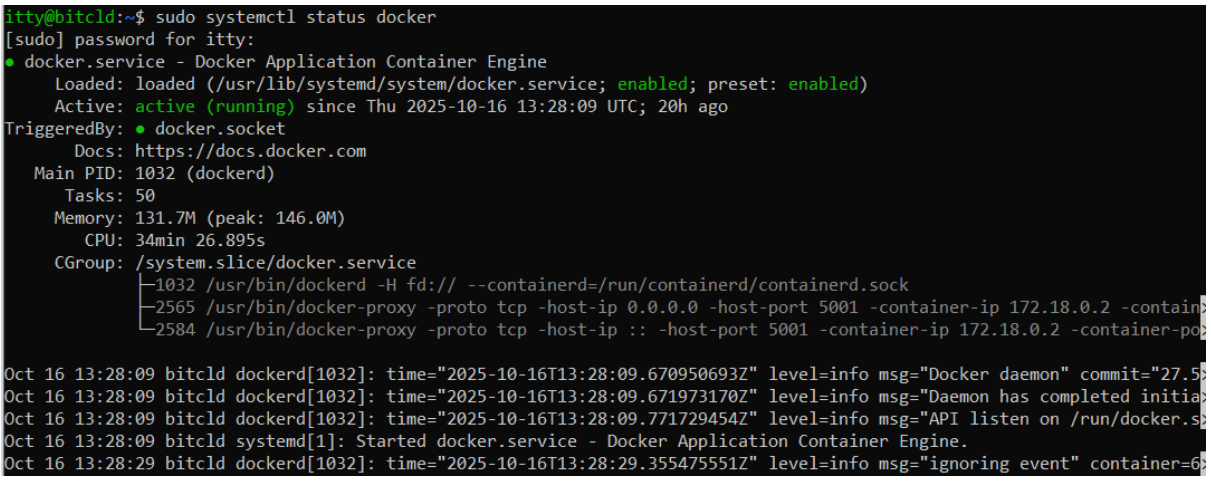
2. Habilitar y arrancar el servicio

A continuación, activamos el servicio de Docker para que se inicie automáticamente al arrancar el sistema y lo iniciamos manualmente por primera vez:

```
sudo systemctl enable docker
sudo systemctl start docker
```

Podemos verificar su estado con:

```
systemctl status docker
```

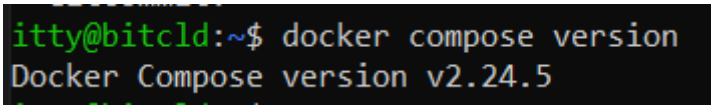


Verificación del funcionamiento de Docker

1. Ver versión y servicios activos

Comprobamos que tanto Docker Engine como Compose están disponibles:

```
docker version
docker compose version
```



```
itty@bitcld:~$ docker version
Client:
 Version:           27.5.1
 API version:       1.47
 Go version:        go1.22.2
 Git commit:        27.5.1-0ubuntu3~24.04.2
 Built:             Mon Jun  2 11:51:53 2025
 OS/Arch:           linux/amd64
 Context:           default

Server:
 Engine:
  Version:          27.5.1
  API version:      1.47 (minimum version 1.24)
  Go version:       go1.22.2
  Git commit:       27.5.1-0ubuntu3~24.04.2
  Built:            Mon Jun  2 11:51:53 2025
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          1.7.27
  GitCommit:
 runc:
  Version:          1.2.5-0ubuntu1~24.04.1
  GitCommit:
 docker-init:
  Version:          0.19.0
  GitCommit:
```

2. Ver lista de contenedores activos y detenidos

Con el primer comando podemos ver los contenedores que están activos en ese momento y con el segundo todos los que tenemos.

```
docker ps
docker ps -a
```

```
itty@bitcld:/$ sudo docker ps
[sudo] password for itty:
CONTAINER ID   IMAGE                   COMMAND                  CREATED       STATUS                   PORTS                                       NAMES
284918a29e7e   louislam/dockge:latest   "/usr/bin/dumb-init …"   6 weeks ago   Up 21 hours (healthy)   0.0.0.0:5001->5001/tcp, :::5001->5001/tcp   dockge
```

```
itty@bitcld:~$ docker ps -a
CONTAINER ID   IMAGE                                COMMAND                  CREATED       STATUS                   PORTS                                       NAMES
f0c50bb81f7a   caddy:latest                         "caddy run --config …"   2 weeks ago   Exited (0) 20 hours ago                                              caddy
83335f687bff   louislam/uptime-kuma:latest          "/usr/bin/dumb-init …"   2 weeks ago   Exited (0) 20 hours ago                                              uptime-kuma
34d10caf4af3   vaultwarden/server:latest            "/start.sh"              2 weeks ago   Exited (0) 2 weeks ago                                               vaultwarden
bac8e75e34c5   ghcr.io/open-webui/open-webui:main   "bash start.sh"          2 weeks ago   Exited (0) 2 weeks ago                                               open-webui
fdc5a9404555   ollama/ollama:latest                 "/bin/ollama serve"      2 weeks ago   Exited (0) 2 weeks ago                                               ollama
6584a176986d   adguard/adguardhome:latest           "/opt/adguardhome/Ad…"   4 weeks ago   Exited (0) 20 hours ago                                              adguardhome
b1a286bc4d6d   nextcloud:latest                     "/entrypoint.sh apac…"   6 weeks ago   Exited (0) 2 weeks ago                                               nextcloud-app
159668e877fe   mariadb:10.8                         "docker-entrypoint.s…"   6 weeks ago   Exited (0) 2 weeks ago                                               nextcloud-db
284918a29e7e   louislam/dockge:latest               "/usr/bin/dumb-init …"   6 weeks ago   Up 20 hours (healthy)    0.0.0.0:5001->5001/tcp, :::5001->5001/tcp   dockge
```

4. Listar imágenes descargadas

Para comprobar las imágenes que tenemos descargadas usaremos este comando

```
docker images
```

```
itty@bitcld:~$ docker images
REPOSITORY                      TAG       IMAGE ID       CREATED         SIZE
ollama/ollama                   latest    bd935981a575   3 weeks ago     3.11GB
ghcr.io/open-webui/open-webui   main      088d66765f5c   3 weeks ago     4.83GB
caddy                           latest    572a72cdc384   7 weeks ago     53.4MB
netdata/netdata                 stable    005ab8bb49d3   7 weeks ago     787MB
squidfunk/mkdocs-material       latest    fed34b78210f   8 weeks ago     163MB
cloudflare/cloudflared          latest    f2cd7be5f497   8 weeks ago     61.4MB
adguard/adguardhome             latest    adba5520580b   8 weeks ago     74.2MB
vaultwarden/server              latest    36fd2ebd3761   2 months ago    256MB
nextcloud                       latest    93bfd883730b   3 months ago    1.42GB
louislam/dockge                 latest    e85d511b2a8b   6 months ago    755MB
louislam/uptime-kuma            latest    542ef8cfcae2   10 months ago   440MB
mariadb                         10.8      d198cd11e8fc   2 years ago     370MB
```

Conclusión

En conclusión Docker y docker compose son una base fundamental de bitCLD, ya que sin estas el despliegue, ejecución y mantenimiento de aplicaciones se haría mucho más complejo y tedioso. Gracias al sistema de contenedores mantenemos nuestro sistema limpio, estable y nos aseguramos que no hayan conflictos entre las aplicaciones.

Combinado con Dockge, que nos ofrece una gestión mucho más visual y automatizada, es una gran solución para bitCLD. Con estas tecnologías cumplimos nuestros pilares: simplicidad, seguridad, autogestión y usar herramientas de código abierto siempre que sea posible.

4.2 Red y seguridad perimetral

4.2.1 Tailscale

Introducción

Tailscale es una red privada virtual (VPN) que usa el protocolo WireGuard. Está diseñada para crear redes privadas seguras de una manera muy simple entre nuestros dispositivos y nuestro servidor siempre que usen la misma cuenta de Tailscale.

Motivos de elección

Nos decantamos por Tailscale ya que necesitábamos una solución de VPN y esta es la que mejor cumplía nuestras necesidades ya que es simple, segura y muy accesible. Además de ser una de las opciones más ligeras, lo cuál es esencial en nuestro caso. Otro de los principales motivos que nos llevaron a decantarnos por Tailscale es que ofrece un plan gratuito muy completo y generoso, algo vital a la hora de reducir costos.

Función dentro del proyecto

En nuestro proyecto, Tailscale funciona como una puerta de acceso secundaria y un canal seguro de administración. Decidimos por compatibilidad y simplicidad que se ejecutara directamente en nuestro servidor y no dentro de un contenedor Docker, dándonos la oportunidad de acceder por SSH en caso de ser necesario desde cualquier ubicación, de forma cifrada y autenticada.

El acceso a nuestros servicios lo hacemos de forma local mediante nuestros dominios .lan y de manera externa mediante los subdominios de bitcld.com. Por lo que Tailscale es más una alternativa de acceso seguro adicional ante cualquier imprevisto y para acceder a la consola de comandos de forma remota.

Instalación

La instalación de Tailscale en nuestro servidor es muy sencilla y no requiere contenedores Docker, ya que se ejecuta directamente sobre el sistema operativo. Para la instalación deberemos:

1 Creamos una cuenta en Tailscale

Debemos crear una cuenta en la página web oficial: https://tailscale.com/

Y creamos una cuenta, nos deja registrarnos con una cuenta de google, microsoft, github o apple. En nuestro caso, por comodidad, usaremos la cuenta de google del proyecto.

2 Instalación del paquete Tailscale

Primero, añadimos el repositorio oficial de Tailscale e instalamos el paquete con el siguiente comando:

```
curl -fsSL https://tailscale.com/install.sh | sh
```

Este script detecta automáticamente la distribución y configura los repositorios necesarios, por lo que nos instala la versión más reciente del cliente Tailscale.

Configuración

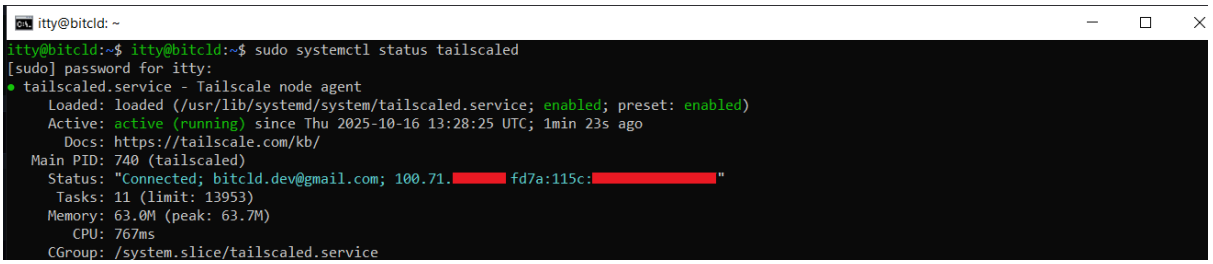
1. Inicio y activación del servicio

Una vez instalado, habilitamos e iniciamos el servicio para que se ejecute automáticamente en cada arranque en caso de que apaguemos o reiniciemos el servidor:

```
sudo systemctl enable tailscaled
sudo systemctl start tailscaled
```

Podemos comprobar su estado con:

```
sudo systemctl status tailscaled
```



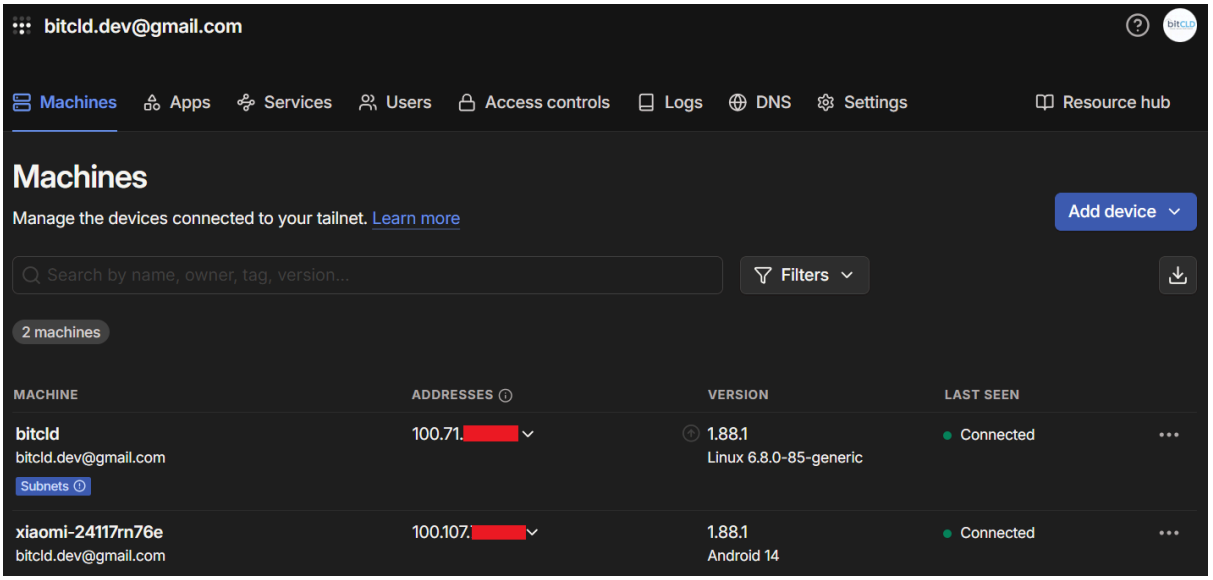
2 Autenticación del dispositivo

El siguiente paso consiste en autenticar el servidor dentro de nuestra red Tailscale. Para ello ejecutamos el siguiente comando y seguimos las instrucciones que aparecen en pantalla:

```
sudo tailscale up
```

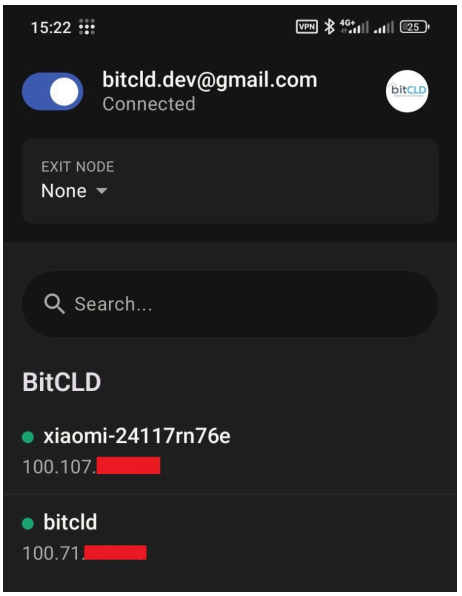
```
itty@bitcld:~$ sudo tailscale up

To authenticate, visit:

        https://login.tailscale.com/a/1b17b67f017fd2
```

El sistema mostrará una URL de autenticación, que abriremos desde cualquier navegador donde hayamos iniciado sesión con nuestra cuenta de Tailscale Tras aprobar el dispositivo, quedará automáticamente vinculado a nuestra red privada.



3 Descarga de la aplicación móvil

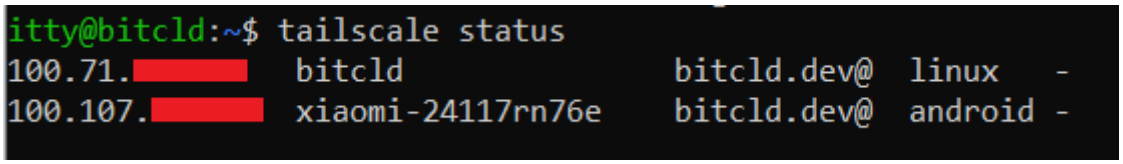
Para poder conectarnos desde nuestro dispositivo móvil descargaremos la aplicación de Tailscale oficial desde la tienda de aplicaciones . E iniciamos sesión con nuestra cuenta ya creada anteriormente.



4 Verificación de conexión

Una vez autenticado, verificamos el estado de la conexión con:

```
tailscale status
```

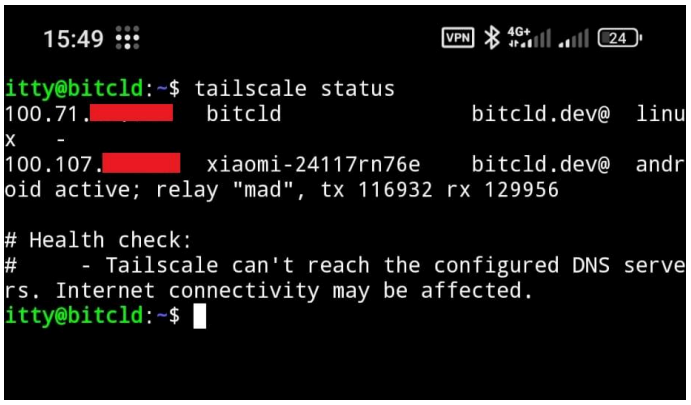


Esto nos mostrará todos los dispositivos conectados a la red, sus direcciones IP privadas de Tailscale y el estado de la conexión. A partir de ese momento, podremos conectarnos al servidor mediante su IP Tailscale o nombre de host interno desde cualquier otro equipo autorizado.

5 Acceso remoto por SSH

Una vez finalizada la autenticación de los dispositivos ya podremos hacer una conexión a nuestro servidor de forma remota desde cualquier dispositivo conectado a la red privada de Tailscale, podemos acceder al servidor mediante SSH de forma segura:

```
ssh usuario@nombre-equipo-tailscale
```



Para poder acceder a la terminal desde el móvil existen diversas aplicaciones, en nuestro caso hemos optado por la aplicación de terminux , por tratarse de una de las más descargadas y su simpleza.

Conclusión

Tailscale VPN nos ofrece un acceso remoto seguro y cifrado al servidor, sin necesidad de abrir puertos ni exponer servicios a Internet. Como la ejecutamos directamente en nuestro servidor nos garantiza una conexión estable y no depender de docker y docker compose.

Nuestra VPN os permite realizar mantenimiento, reinicios o ajustes de emergencia de manera totalmente cifrada y sin depender de puertos abiertos o túneles externos lo cuál es

excelente y justo lo que necesitábamos. Y se alinéa con nuestros pilares: simple, seguro, autogestionado y de código abierto.

4.2.2 Cloudflare

Introducción

Cloudflare es una plataforma que ofrece seguridad y rendimiento para aplicaciones web. Un gran número de páginas webs y empresas confían en sus servicios y se diferencian por su gran estabilidad por norma general, salvo durante partidos de la liga donde ahí sí que se ve afectada por los bloqueos de IP masivos de La Liga.

Motivos de elección

Hemos escogido Cloudflare debido a cuatro motivos principales, los cuáles son esenciales en nuestro proyecto.

Seguridad: La solución de cloudflared tunnel elimina la necesidad de que tengamos que abrir puertos en nuestro servidor, lo cual reduce enormemente las posibilidades de ataque. Además, también nos ofrece sistemas de autenticación mediante Cloudflare Access, que permite restringir el acceso a las aplicaciones mediante identidades verificadas, añadiendo una capa Zero Trust sin necesidad de configurar VPNs o firewalls complejos.

Simplicidad: Cloudflared Tunnel nos facilita la vida con la exposición de servicios con un solo comando y su posterior gestión desde la página web. Además, los certificados HTTPS y la gestión DNS los automatiza como veremos más adelante, reduciendo casi al mínimo la configuración manual y posibles errores.

Rendimiento: Debido a que usamos su red CDN global, las solicitudes se enrutan de manera más eficiente y se mejora la velocidad y estabilidad del acceso, incluso desde ubicaciones remotas.

Precio: Aparte de todo eso también nos proporciona una experiencia profesional sin ningún coste adicional en su plan gratuito el cual es bastante generoso.

Función dentro del proyecto

Cloudflare es vital para la conectividad externa del proyecto. Ya que por una parte es la plataforma que gestiona nuestro dominio.

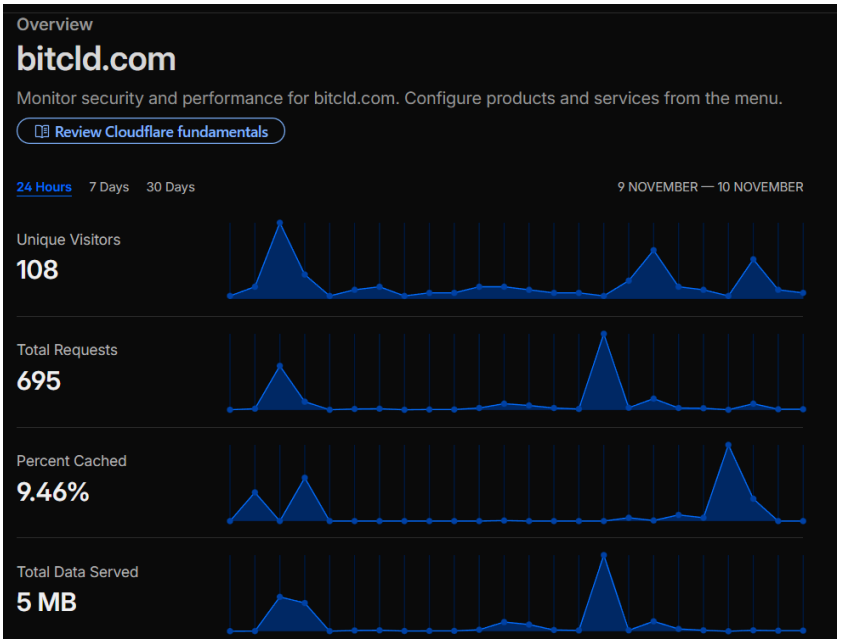
A través de Cloudflared Tunnel, cada servicio que desplegamos en Docker se conecta directamente a la red de Cloudflare, evitando el uso de redirecciones o puertos abiertos y nos permite acceder mediante subdominios personalizados.

Con Cloudflare Access controlamos quién puede entrar a cada aplicación mediante un código OTP que se recibirá en el correo.

Y adicionalmente, Con Cloudflare Pages, Nos da la oportunidad de desplegar páginas web, en nuestro caso la de nuestra documentación. Sin embargo, para no mezclarlo con el proyecto en sí, lo hemos añadido en el Anexo correspondiente A.1.2 Cloudflare Pages.

Gestor de dominos

Para una gestión mucho más simple y eficaz optamos por Cloudflare para adquirir nuestro dominio y así facilitar mejor la integración con todo lo que deberemos configurar para que funcione nuestro ecosistema.



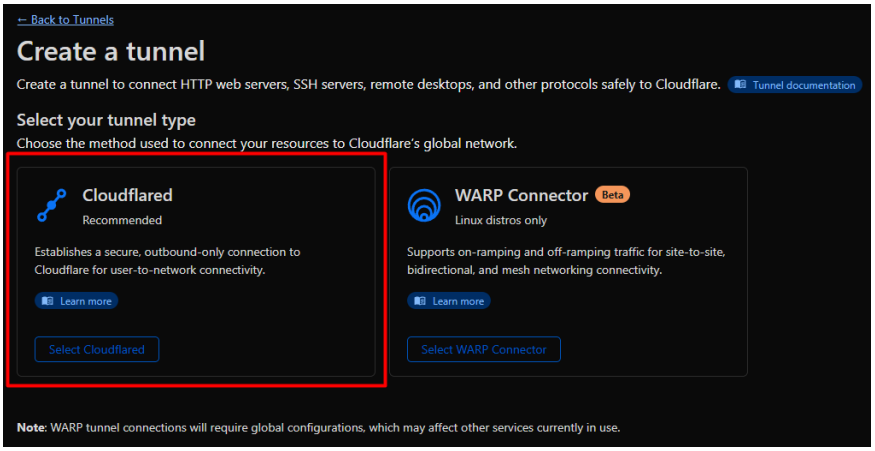
Cloudflare Tunnel

Cloudflared permite conectar el servidor local con la red segura de Cloudflare sin necesidad de abrir puertos. Ya que actúa como un “túnel inverso". Gracias a eso, nuestros servicios son accesibles mediante nuestro dominio al público (pass.bitcld.com, cld.bitcld.com...) evitándonos tener que exponer puertos.

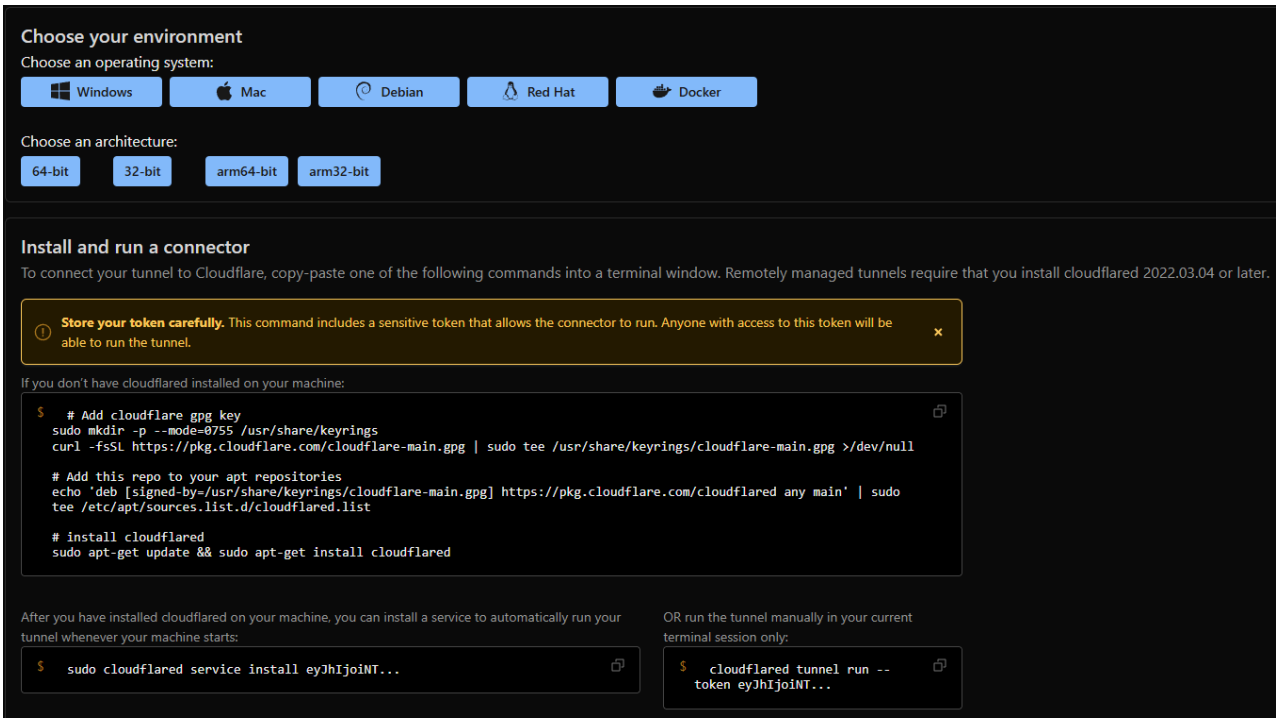
Creación de túnel

Para la creación del tunel usaremos el panel de Cloudflare ya que es mucho más intuitivo. Una vez tengamos nuestra cuenta creada y abierta nos iremos a la opción de Zero Trust.

Una vez en la sección de Zero Trust deberemos ir a la sección de Networks > Tunnels > + Create a tunnel y una vez dentro seleccionaremos Cloudflared.



Asignaremos un nombre al túnel y seleccionamos el sistema operativo donde lo vamos a instalar, en nuestro caso Debian, y Cloudflare nos generará un archivo y nos mostrará los pasos que deberemos seguir:



Los primeros comandos son para instalar cloduflared en el dispositivo y el siguiente comando es para instalar y registrar el túnel en nuestro servidor. Así toda la gestión la podremos hacer desde el panel y no manualmente.

```
sudo cloudflared service install eyJhIjoiNTE3MmE5OTkwN...
```

Para verificar que está en funcionamiento podemos comprobarlo con el comando:

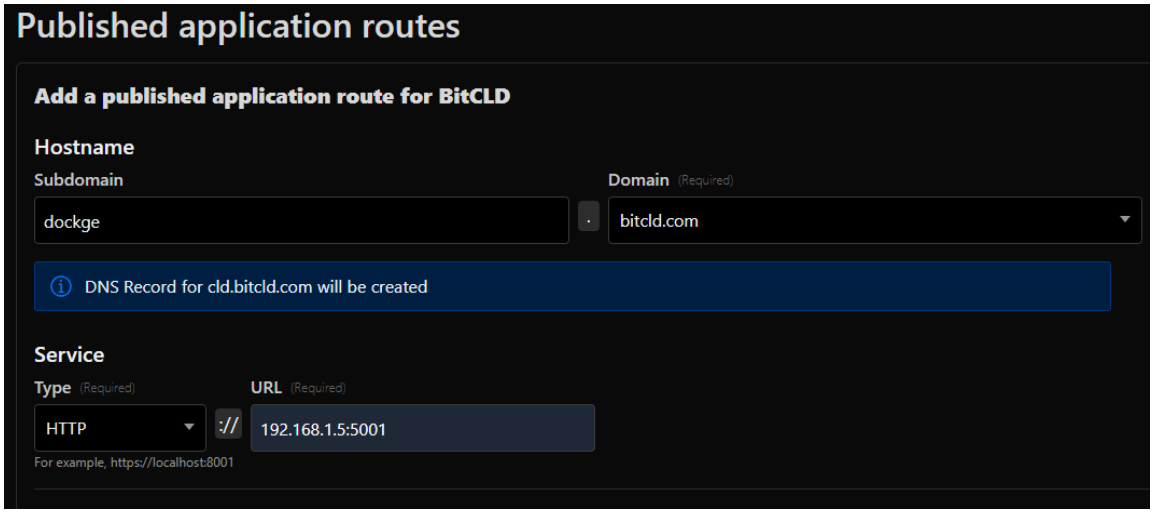
```
sudo systemctl status cloudflared
```

```
itty@bitcld:~$ sudo systemctl status cloudflared
[sudo] password for itty:
● cloudflared.service - cloudflared
     Loaded: loaded (/etc/systemd/system/cloudflared.service; enabled; preset: enabled)
     Active: active (running) since Mon 2025-11-03 10:07:35 UTC; 5h 49min ago
   Main PID: 1026 (cloudflared)
      Tasks: 9 (limit: 13953)
     Memory: 55.1M (peak: 57.9M)
        CPU: 1min 46.953s
     CGroup: /system.slice/cloudflared.service
             └─1026 /usr/bin/cloudflared --no-autoupdate tunnel run --token [redacted]
```

Publicación de servicios internos

Una vez que tengamos nuestro túnel ya podremos añadir aplicaciones, para ello en la configuración del túnel iremos a la opción de Published application Routes y seleccionaremos + Add a published application route.

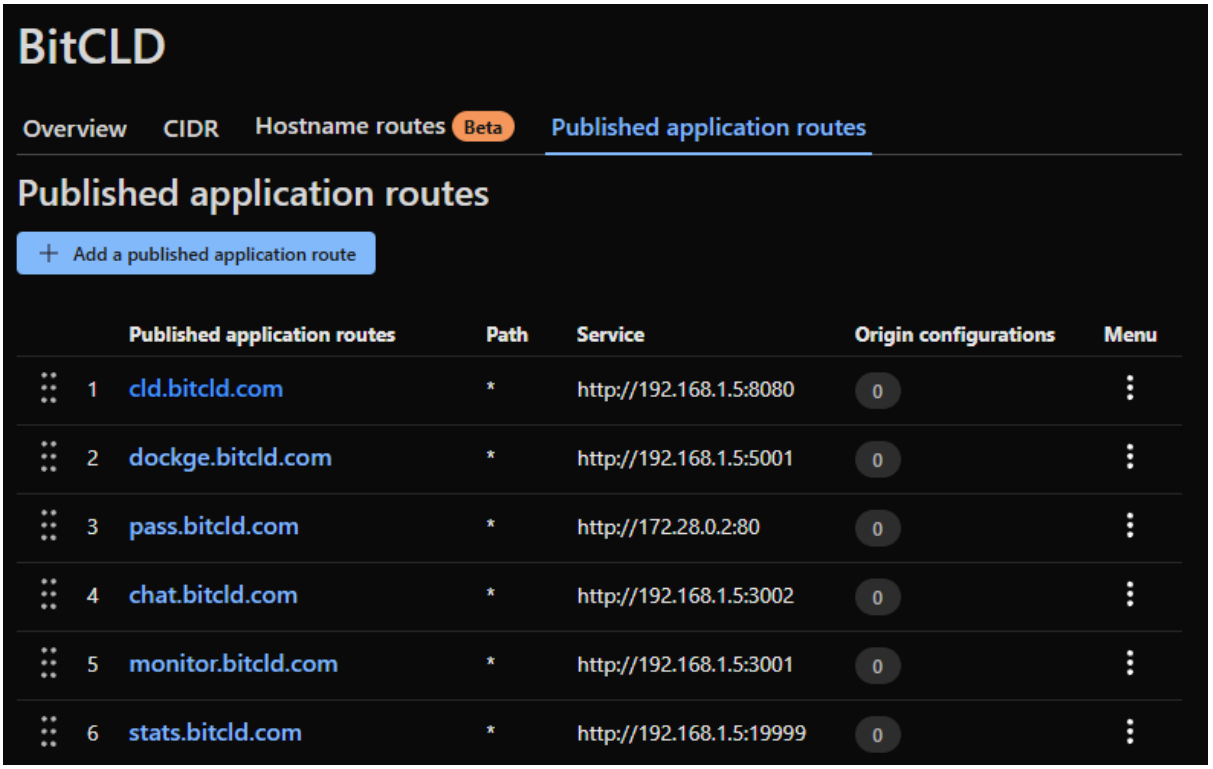
E indicaremos el subdominio que queramos para la aplicación, el dominio, el tipo de servicio y la dirección IP y el puerto:



Una vez creado Cloudflare se encargará de:

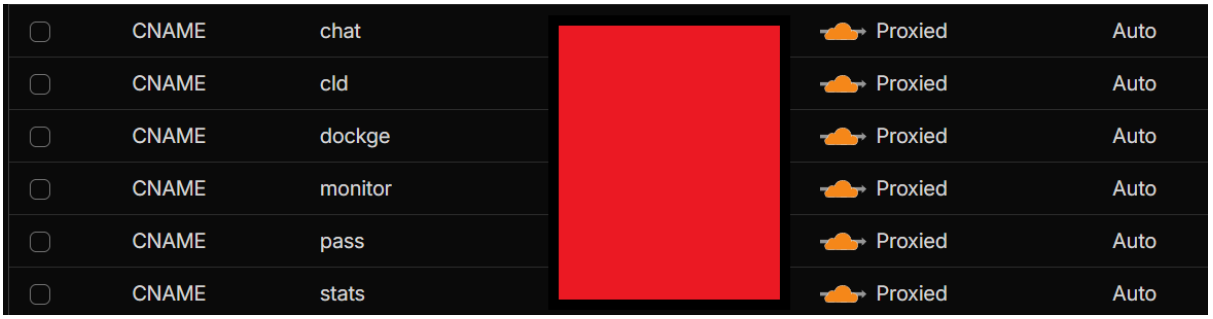
- ➔ Crear el registro DNS correspondiente (tipo CNAME).
- ➔ Asignar el certificado SSL/TLS.
- ➔ Rutar el tráfico por el túnel de forma cifrada.

Una vez hecho con cada servicios que queramos exponer ya los tendríamos públicos:



De esta manera el acceso a cada servicio se realiza directamente a través de su dominio público, sin necesidad de abrir ningún puerto en el servidor o en el router.

Cómo podemos observar en los registros DNS de nuestro dominio. Cloudflare ha creado los correspondientes registros CNAME para cada servicio:



Cloudflare Access

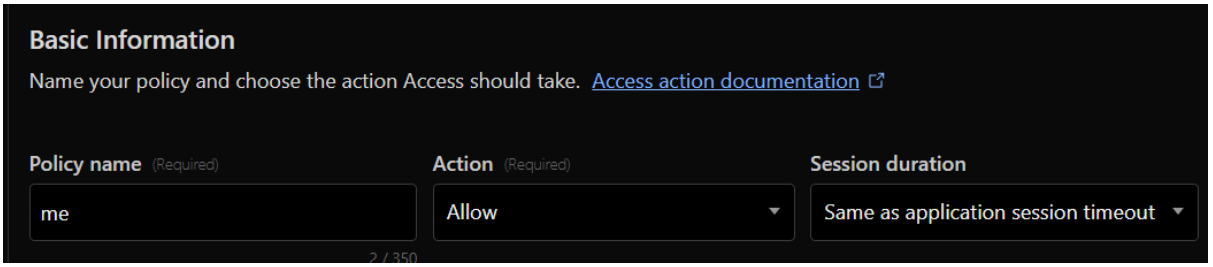
Ahora nuestros servicios están públicos pero cualquiera con el subdominio podría acceder es por ello que necesitamos de Cloudflare Access para poder autentificar usuarios. En nuestro caso se hará mediante un código OTP y con limitación geográfica.

Crear políticas

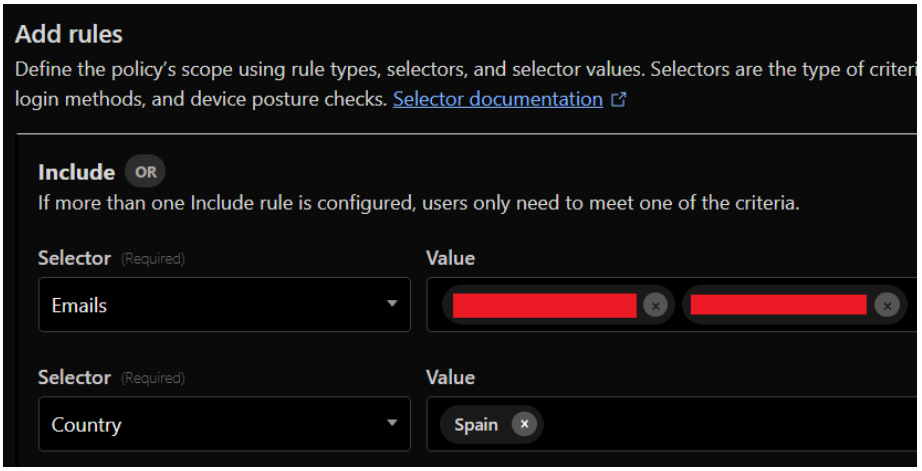
Para poder proteger nuestros servicios con Access necesitaremos primeramente crear una política:

Para ello en la sección de Zero Trust seleccionamos la opción de policies y +Add a policy
Política de autorización de usuarios

Para la política de autorización de usuarios asignaremos un nombre en Action seleccionaremos la opción Allow e indicamos un tiempo de duración de la sesión, en nuestro caso seleccionaremos la opción Same as application session timeout, para que dure lo mismo que la aplicación:



En Add rules:
En la opción Selector indicamos Emails y añadimos los correos que deseemos y por seguridad añadimos también la opción de país seleccionando en Country y el país que deseamos, en nuestro caso España.

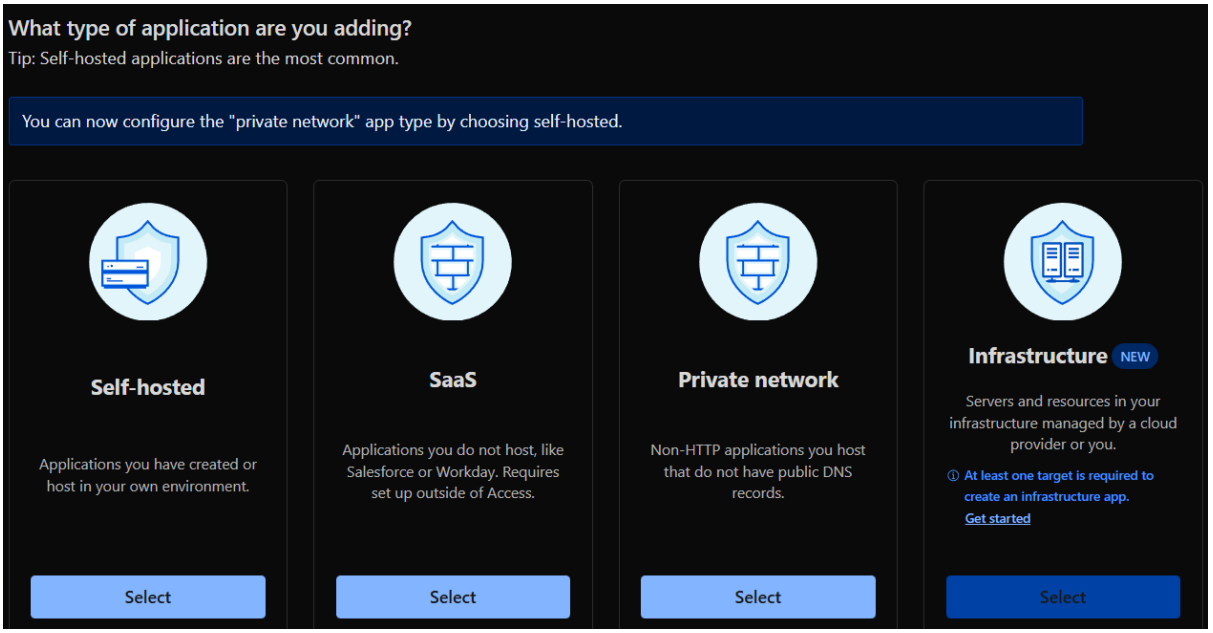


De esta manera ya sólo se podrá a las aplicaciones de España y cada vez que queramos entrar deberemos poner el código OTP que nos habrá llegado al correo.

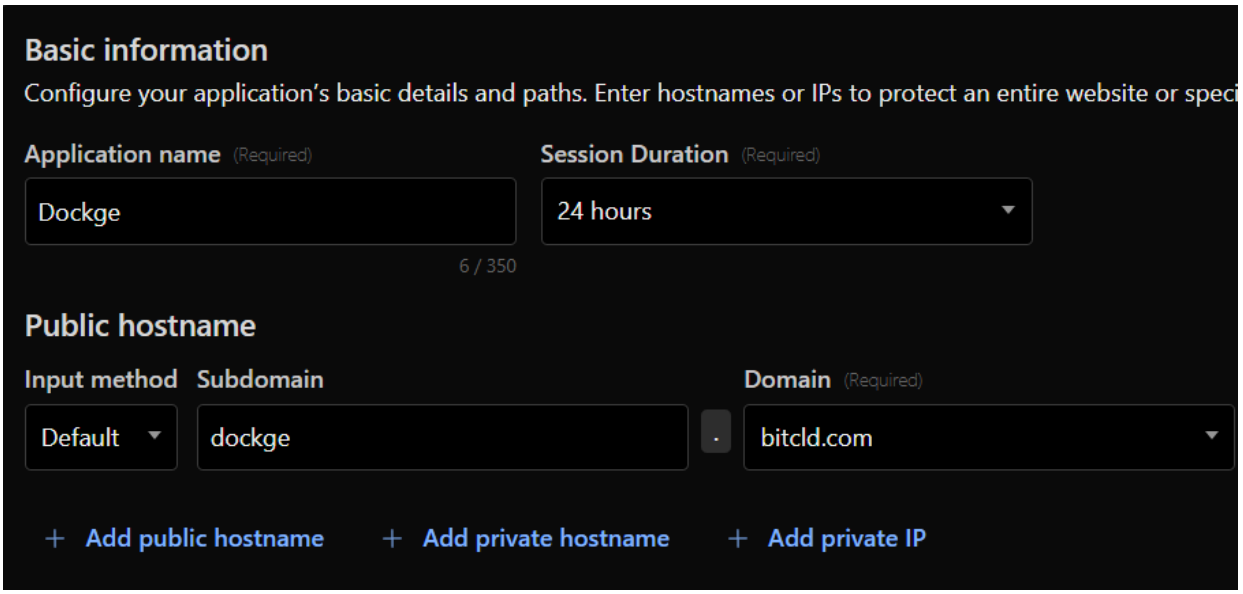
Crear aplicaciones

Una vez creada la política crearemos una aplicación para cada servicio que ya tenemos expuesto:

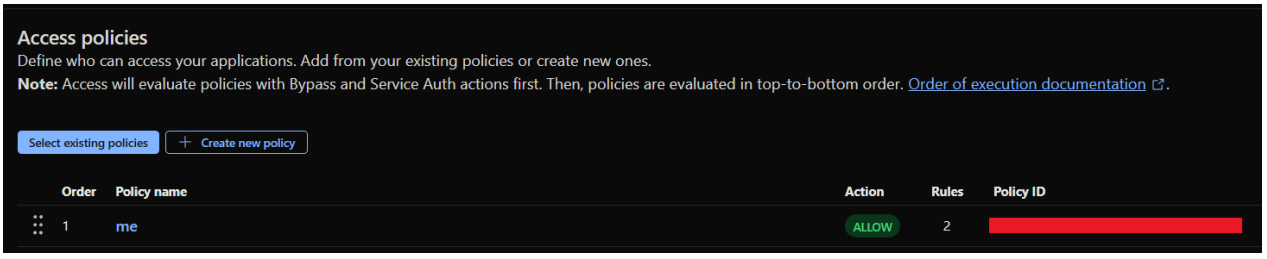
Vamos a la sección de Zero Trust en nuestro panel de administración de cloudflare y en Access seleccionamos la opción de Applications > +Add an application y seleccionamos la opción de Self-hosted.



En el siguiente paso asignamos un nombre a la aplicación y seleccionamos +Add public hostname y ponemos el subdominio y dominio correspondiente al servicio.

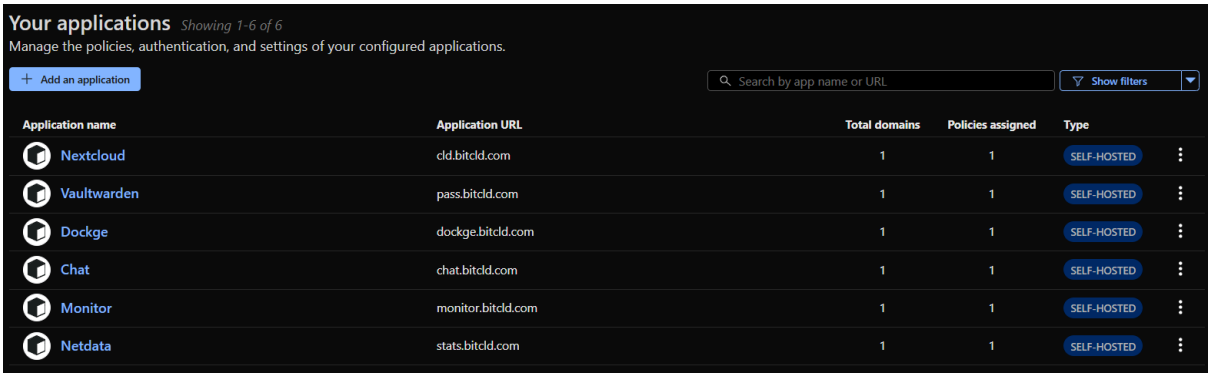


En Access policies seleccionamos la política que hemos creado anteriormente:



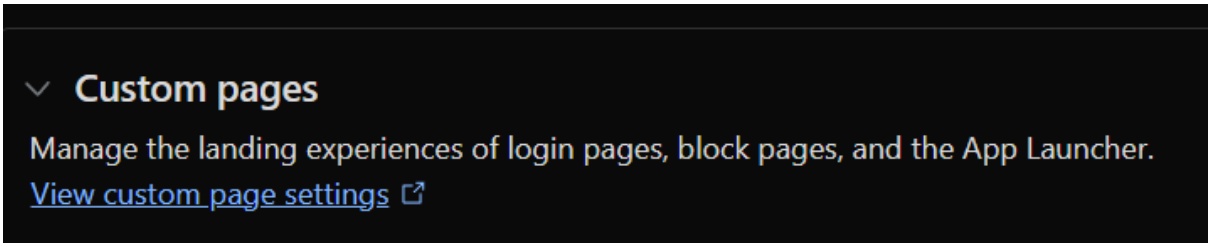
Es muy importante añadir la política ya que si no se pone, no se podrá acceder al servicio ya que no podremos recibir el código al email.

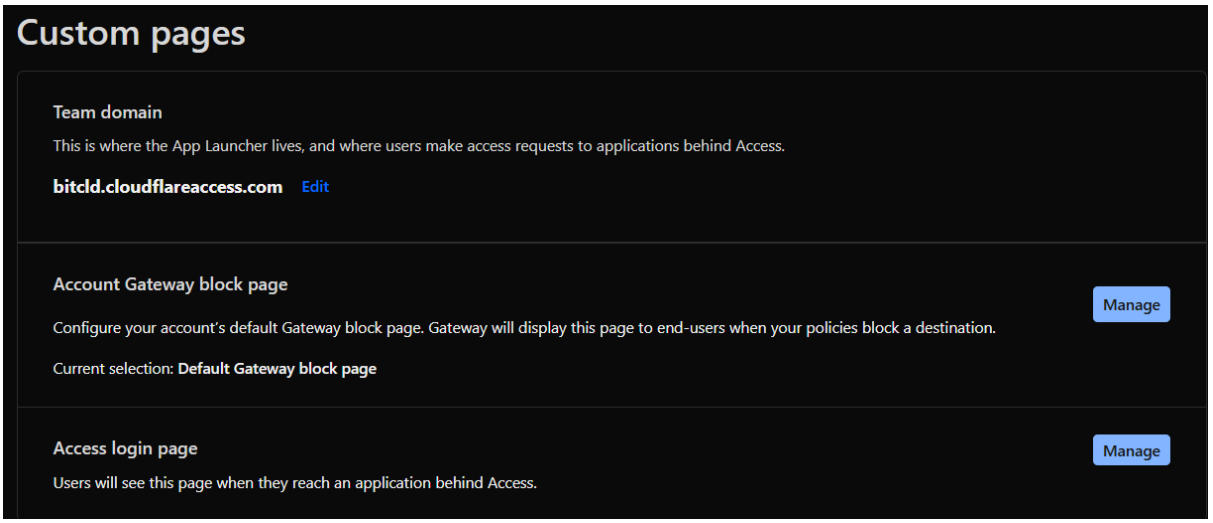
Una vez creadas todas nuestras aplicaciones deberemos asegurarnos de que en todas está con la política asignada:



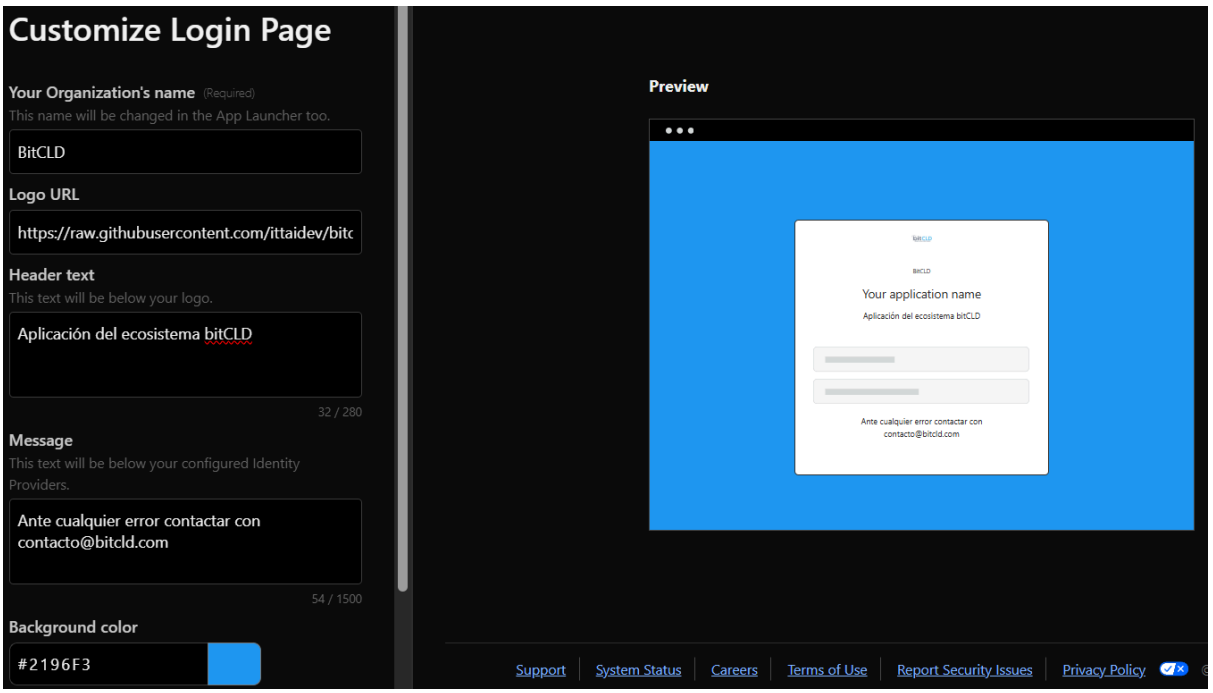
Personalizar la página de Access

Para personalizar la página de Access y darle un toque más profesional es muy sencillo. Seleccionamos cualquier aplicación que hemos creado y en Configure > Experience settings accedemos al enlace del apartado Custom Pages y una vez dentro seleccionamos Mange en Access login page





En este apartado podemos poner un nombre a nuestra organización, ponerle un logo, añadir algún mensaje y un color para que se vea mucho más profesional:



Para poder poner el logo necesitamos que la imagen esté subida a internet, la opción más sencilla y sin coste alguno es subirla a Github. Para ello la imagen deseada la subimos a un repositorio público y copiamos el enlace de la imagen:

```
https://github.com/ittaidev/bitcld-homepage/blob/main/images/logorecortenb.png
```

Y una vez copiado lo modificamos cambiando github.com por raw.githubusercontent.com y quitamos el /blob

```
https://raw.githubusercontent.com/ittaidev/bitcld-homepage/main/images/logorecortenb.png
```

De esta manera ya tendríamos el logo para poder ser usado por Cloudflare Access.

Conclusión

Cloudflare nos permite desplegar un túnel, conectar con el servidor y publicar servicios en cuestión de minutos, sin necesidad de tener que estar poniendo comandos complejos. Además, la unión con Cloudflare Access refuerza la seguridad de nuestro ecosistema y también nos permite publicar páginas web mediante Pages. Es por ello que nos decantamos por usar esta tecnología.

4.2.3 Caddy Server

Introducción

Caddy server como su nombre indica es un servidor web y proxy inverso de código abierto, el cual es muy conocido por su simplicidad, ligereza y facilidad de configuración.

Motivos de elección

Elegimos Caddy ya que necesitábamos una herramienta ligera, segura y fácil de mantener, en línea con los pilares de bitCLD: simplicidad, seguridad, autohospedaje y código abierto.

Caddy server nos permite gestionar varios servicios mediante un único archivo de configuración (Caddyfile), sin depender de configuraciones complejas como las de Nginx o Apache.

Función dentro del proyecto

En bitCLD, Caddy actúa únicamente como reverse proxy interno dentro de la red local.
Nos gestiona las conexiones entre los distintos servicios y nos ofrece acceso mediante dominios internos (*.lan) y certificados TLS locales generados automáticamente.

En cambio el tráfico externo no pasa por Caddy, sino que es gestionado directamente por Cloudflared. De esta forma, Caddy gestiona el entorno LAN mientras que Cloudflare se encarga de controlar el acceso remoto seguro.

Despliegue con Docker

Caddy se ejecuta en un contenedor Docker, con su propia red interna caddy_net:

```
services:
caddy:
    image: caddy:latest
    container_name: caddy
    restart: unless-stopped
    ports:
    - 80:80
    - 443:443
    volumes:
    - ./Caddyfile:/etc/caddy/Caddyfile:ro
    - ./data:/data
    - ./config:/config
    networks:
    - caddy_net

networks:
caddy_net:
    external: true
```

Explicación del docker compose

Parámetro	Descripción
image: caddy:latest	Usa la última versión oficial de la imagen de Caddy desde Docker Hub.
container_name: caddy	Asigna un nombre fijo al contenedor (útil para comandos y depuración).
restart: unless-stopped	Hace que el contenedor se reinicie automáticamente si se detiene de forma inesperada, salvo que se haya detenido manualmente.
80:80	Expone el puerto 80 para tráfico HTTP.
443:443	Expone el puerto 443 para tráfico HTTPS.
./Caddyfile:/etc/caddy/Caddyfile:ro	Archivo principal de configuración de Caddy. Se monta como solo lectura (ro) para mayor seguridad.
./data:/data	Almacena certificados SSL y datos generados por Caddy.
./config:/config	Guarda configuraciones internas de Caddy (por ejemplo, caché o estados).
caddy_net	El servicio se conecta a la red caddy_net, lo que permite que Caddy se comunique con otros contenedores
external: true	Indica que la red caddy_net ya existe y no será creada por este docker-compose. Esto permite compartir la misma red entre distintos docker-compose.yml, haciendo posible que varios contenedores se comuniquen entre sí.

Configuración del Caddyfile

El archivo Caddyile contiene los distintos subdominios internos y sus destinos locales. Cada dominio usa tls internal para generar certificados automáticos válidos dentro de la red. Para ver el CaddyFile completo puede hacerlo en el Anexo Correspondiente A.4 Caddy File.

Integración con AdGuard Home (DNS interno)

AdGuard Home es nuestro servidor DNS en bitCLD y lo hemos configurado para que los dispositivos de la red puedan resolver los dominios locales `*.lan` hacia la IP del servidor principal `192.168.1.5`.

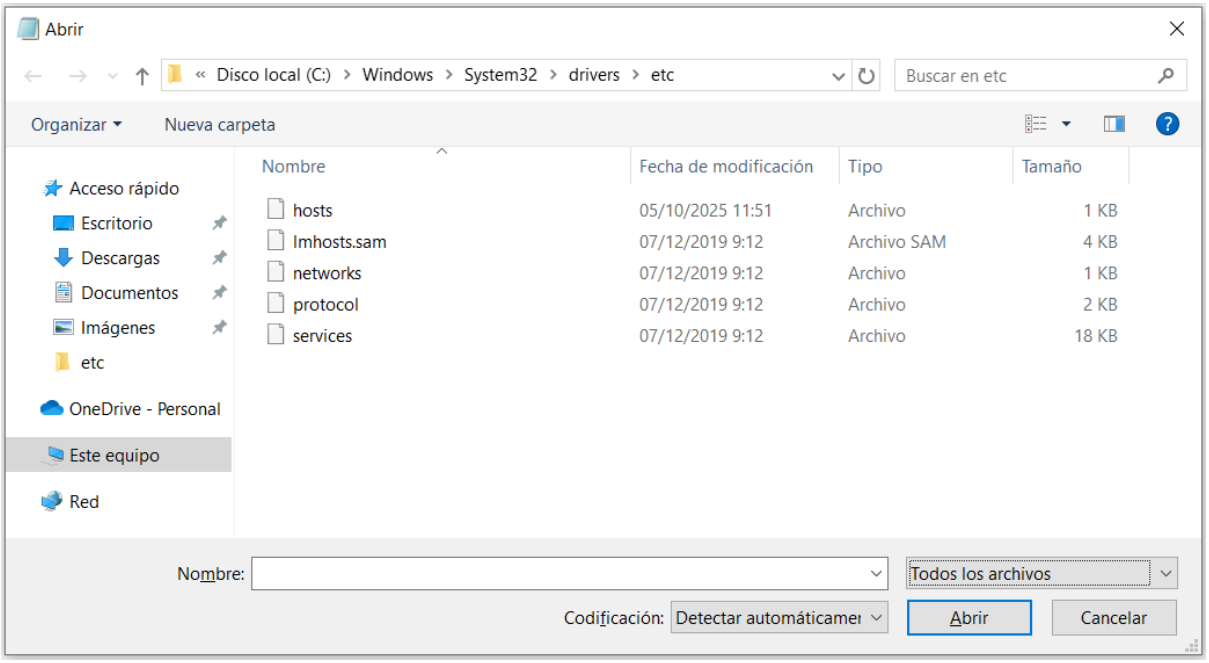
Para ello, sólo deberemos definir unos rewrites personalizados, de forma que cuando un dispositivo de la red haga una consulta a, por ejemplo, `dockge.lan`, el DNS le redirige correctamente al servicio correspondiente gestionado por Caddy. Para poder usar esta opción es necesario que usemos AdGuard Home como servidor DNS en nuestro dispositivo.

Para ver en más profundidad el funcionamiento de nuestro servidor DNS puede hacerlo en el apartado de 4.3.2 Adguard Home.

Integración sin necesidad de cambiar DNS

En Windows y Linux podemos evitar la necesidad de usar Adguard Home como DNS modificando un archivo, sin embargo, en móvil si que es obligatorio usar bitCLD como servidor DNS para que podamos usar los dominios .lan.

En Windows se debe modificar el archivo hosts. Para ello abrimos el bloc de notas como administrador > Abrir... > C:\Windows\System32\drivers\etc\hosts y seleccionamos "Todos los archivos" para que podamos verlo.



Dentro del archivo incluimos las rutas correspondientes las cuales son:

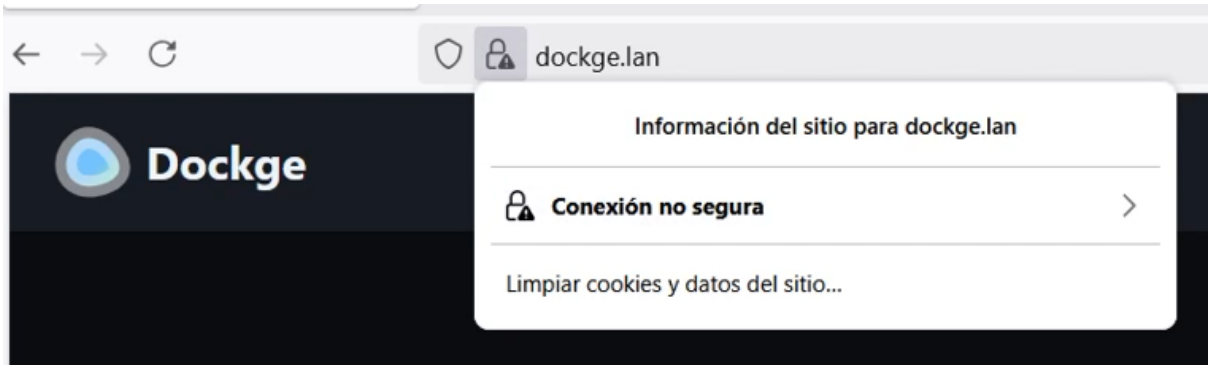
```
192.168.1.5 dockge.lan
192.168.1.5 cld.lan
192.168.1.5 dns.lan
192.168.1.5 monitor.lan
192.168.1.5 stats.lan
192.168.1.5 pass.lan
192.168.1.5 chat.lan
```

Después de guardar los cambios en el archivo hosts, debemos vaciar la caché del DNS para que los cambios surtan efecto. Para ello abrimos el símbolo del sistema como administrador y ejecutamos el comando `ipconfig /flushdns`.

En linux se trata de un proceso similar, el archivo que deberemos modificar se encuentra en /etc/hosts y deberemos copiar las mismas rutas que en windows. Una vez copiadas vaciaremos la caché de DNS con sudo `systemctl restart systemd-resolved` Y ya tendríamos las rutas en nuestro sistema linux.

Certificados y acceso HTTPS interno

Para que los dispositivos de la red puedan acceder correctamente a los dominios locales con HTTPS, es necesario instalar el certificado raíz de Caddy en cada dispositivo que los utilice. Ya que sin este paso, los navegadores marcarán los dominios .lan como “no seguros”, aunque la conexión sea realmente cifrada.



Ubicación del certificado

El certificado raíz se encuentra en el contenedor de Caddy en:

```
/data/caddy/pki/authorities/local/root.crt
```

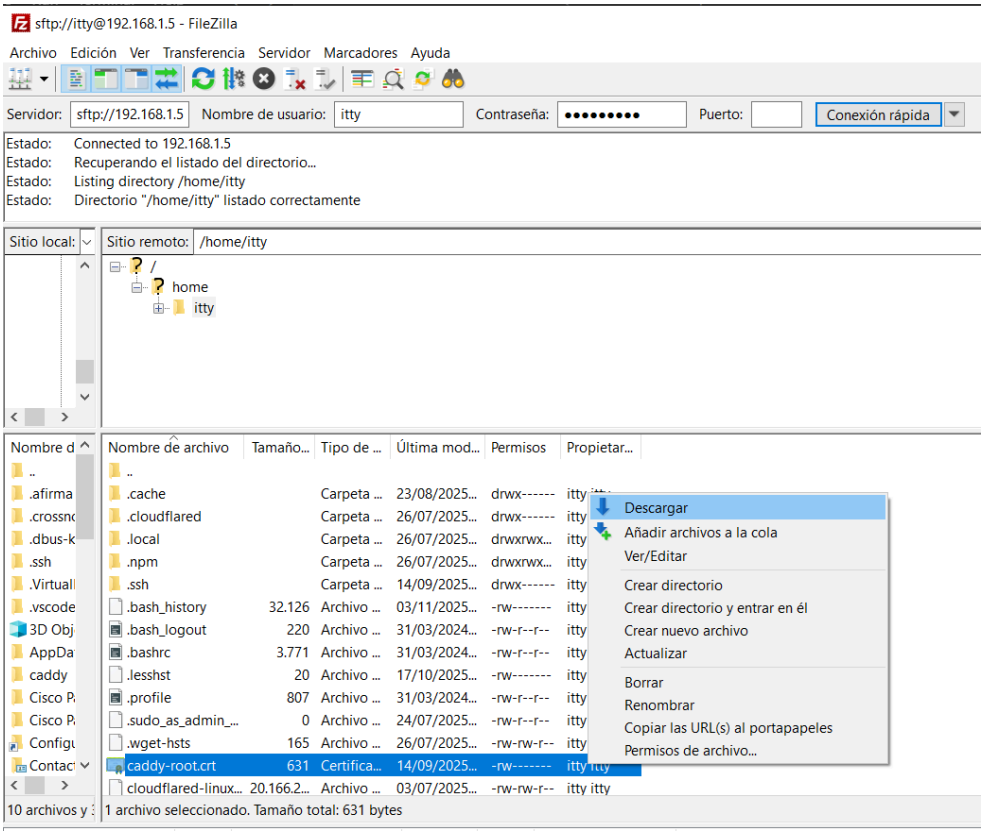
Debido a que nuestro usuario no es root, para copiarlo a nuestro equipo primero debemos abrir una conexión ssh a nuestro servidor y usar el comando:

```
sudo cp
/srv/docker/stacks/caddy/data/caddy/pki/authorities/local/root.crt
/home/itty/caddy-root.crt
```

Una vez copiado a nuestra carpeta personal, cambiamos el propietario para que podamos leerlo con `sudo chown itty:itty /home/itty/caddy-root.crt`
Y ya lo tendríamos en nuestra carpeta personal:

```
itty@bitcld:~$ ls
caddy-root.crt  cloudflared-linux-amd64.deb
```

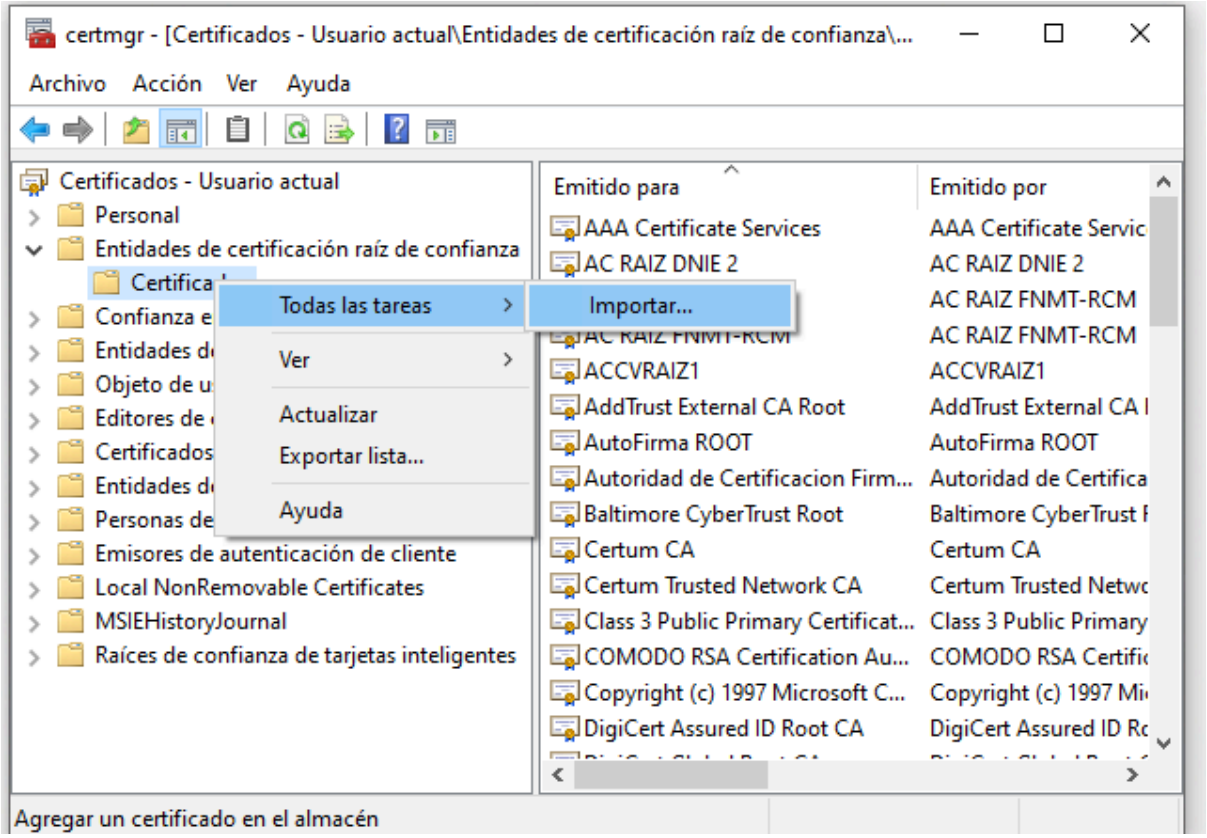
Una vez en la carpeta personal, ya podremos copiarlo con el siguiente comando en nuestro ordenador personal scp `itty@192.168.1.5:/home/itty/root.crt ./` o descargarlo mediante FileZilla.



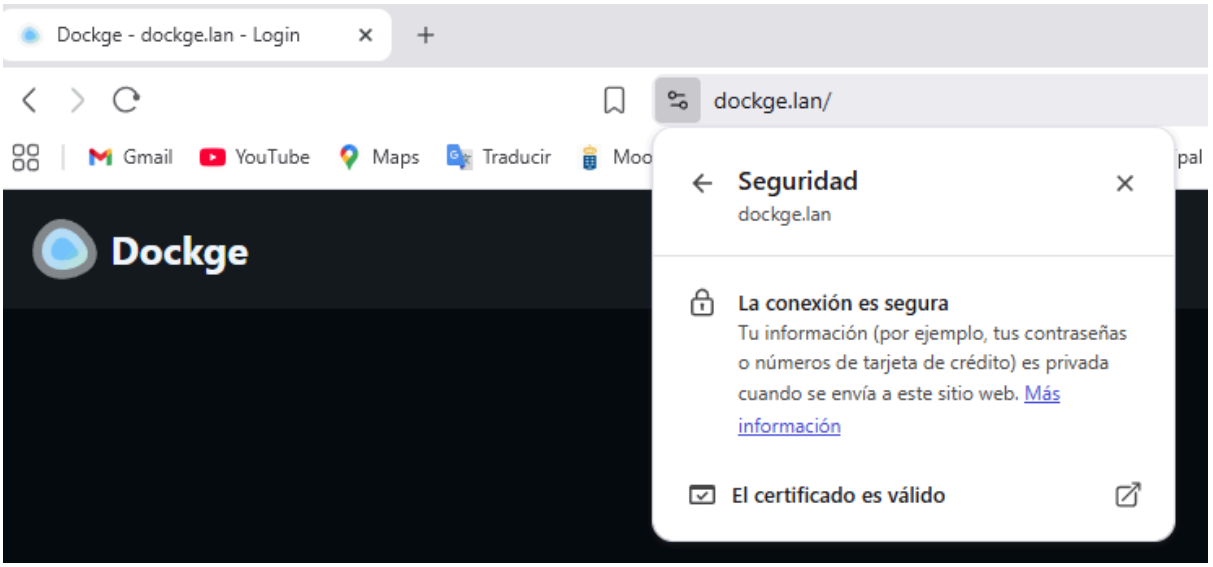
Proceso de instalación del certificado

Windows

En Windows debemos ir a certmgr.msc > Entidades de certificación raíz de confianza > Todas las tareas > Importar



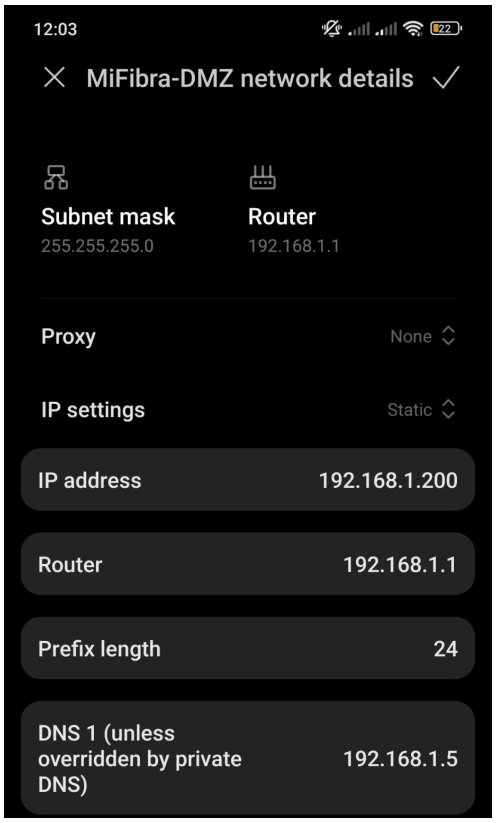
Una vez descargado veremos que ya reconoce el certificado y que se establece una conexión segura.



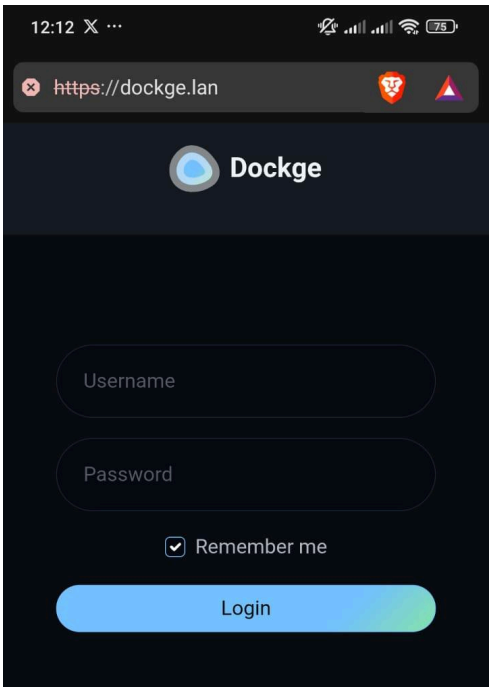
Android

En Android, a diferencia de Windows o Linux, es obligatorio usar Adguard Home como servidor por lo que primero debemos asegurarnos que está como nuestro servidor DNS. Por

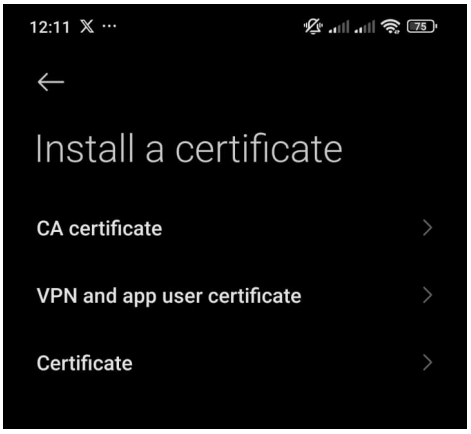
lo que primero deberemos asegurarnos que estamos usando Adguard Home como servidor DNS.



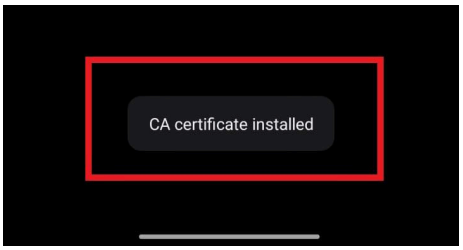
Una vez estamos usando nuestro servidor como servidor DNS podemos observar que tenemos acceso a `dockge.lan`, pero que nuestro dispositivo no reconoce el certificado de caddy y nos muestra como conexión no segura.



Para instalar certificado debemos irnos a Privacidad > Más ajustes > Encriptación y credenciales > Instalar certificado > CA certificate



Seleccionamos la opción de Instalar de todos modos y ponemos nuestro código de desbloqueo del móvil, seleccionamos donde tengamos guardado el archivo .cer



En Linux: Debemos añadir el .crt en /usr/local/share/ca-certificates/ y ejecutar el comando update-ca-certificates.

Conclusión

Aunque la función es interna, Caddy es un componente muy importante dentro de bitCLD. Ya que junto con Adguard Home, nos permite un acceso seguro y organizado a todos los servicios mediante dominios locales y certificados válidos.

Al combinarlo con Cloudflare Tunnel para el acceso remoto nos permite crear una infraestructura completa, donde cada herramienta cumple una función clara dentro de los pilares del proyecto: simplicidad, seguridad, autohospedaje y uso de código abierto.

4.3 Despliegue de servicios

4.3.1 Dockge

Introducción

Dockge es un gestor de contenedores moderno, muy ligero y de código abierto. Nos da la oportunidad de visualizar, administrar y gestionar varios stacks de una manera centralizada y muy simple, lo que nos ahorra bastante trabajo.

Motivos de elección

Elegimos dockge ya que necesitábamos una solución para administrar nuestros contenedores docker compose. Una de las principales razones por las que nos decidimos por este y no por otro gestor es porque nos permite editar directamente los archivos compose.yml desde su propia interfaz, a parte de crearnos las carpetas que necesitemos con sólo añadirlas al docker compose. Por todo esto lo elegimos para nuestro proyecto ya que cumple con los pilares del mismo: simple, seguro, autogestionado y de código abierto.

Función dentro del proyecto

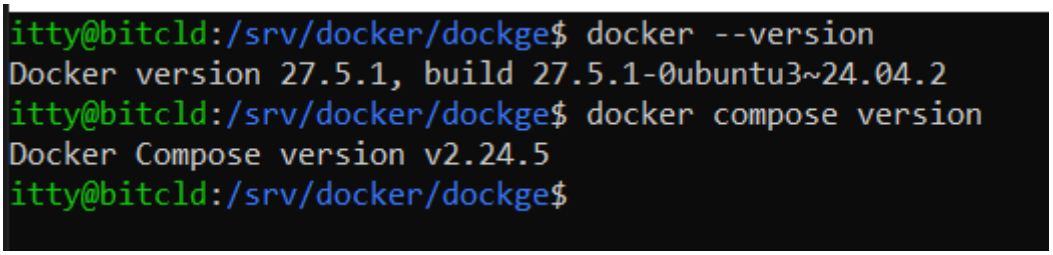
Dentro de bitCLD la función de Dockge es de las más importantes, ya que es el gestor de todos nuestros servicios. Desde su interfaz podemos iniciar, detener, reiniciar o editar cualquier contenedor que gestione, ahorrándonos el tener que hacerlo mediante la consola.

Instalación

0.1 Requisitos previos

Debemos asegurarnos de tener docker y docker compose instalado en nuestro servidor:

```
docker --version
docker compose version
```



Puede ver la documentación de Docker y docker compose en este enlace 4.1.3 Docker y docker Compose.

Y también nos aseguramos de crear las carpetas necesarias con:

```
sudo mkdir -p /srv/docker/dockge/data
sudo mkdir -p /srv/docker/stacks
```

Creamos el docker-compose.yml de Dockge

Creamos el archivo en `/srv/docker/dockge/docker-compose.yml` con este contenido (ajustado a nuestras necesidades):

```
services:
dockge:
    image: louislam/dockge:latest
    container_name: dockge
    restart: unless-stopped
    ports:
    - "5001:5001"
    volumes:
    - /var/run/docker.sock:/var/run/docker.sock
    - ./data:/app/data
    - /srv/docker/stacks:/srv/docker/stacks
    environment:
    - DOCKGE_STACKS_DIR=/srv/docker/stacks
```

Explicación del docker compose

Parámetro	**Descripción**
image: louislam/dockge:latest	Usa la última versión oficial de la imagen de Dockge disponible desde Docker Hub.
container_name: dockge	Asigna un nombre fijo al contenedor (útil para comandos y depuración).
restart: unless-stopped	Hace que el contenedor se reinicie automáticamente si se detiene de forma inesperada, salvo que se haya detenido manualmente.
- /var/run/docker.sock:/var/run/docker.sock	Es lo que permite a Dockge comunicarse directamente con Docker, listar contenedores, crear, detener, y modificar stacks. (Sin esto, Dockge no podría controlar Docker.)
- ./data:/app/data	Guarda la configuración interna y datos persistentes de Dockge (así no se pierden lass configuraciones).
- /srv/docker/stacks:/srv/docker/stacks	Es el directorio donde Dockge buscará, creará o modificará los docker-compose de los servicios (Nextcloud, Caddy...) Dockge creará automáticamente las carpetas necesarias al crear nuevos stacks desde su interfaz.
- DOCKGE_STACKS_DIR=/srv/docker/stack	Indica a Dockge cuál es la ruta principal donde debe gestionar los stacks. (Aunque

s	ya está montado como volumen, esta variable es necesaria para que la interfaz sepa dónde leer y guardar los archivos docker-compose.yml)

Despliegue de Dockge

Una vez tengamos hecho el compose.yml en `/srv/docker/dockge` procederemos a instalarlo:

Desde /srv/docker/dockge ejecutamos los siguientes comandos:

```
docker compose pull
docker compose up -d
```

Comprobamos que está en marcha:

```
docker ps --filter name=dockge
```

```
itty@bitcld:/$ docker ps --filter name=dockge
CONTAINER ID   IMAGE                  COMMAND                  CREATED       STATUS                    PORTS                                         NAMES
284918a29e7e   louislam/dockge:latest   "/usr/bin/dumb-init …"   2 months ago   Up 59 minutes (healthy)   0.0.0.0:5001->5001/tcp, :::5001->5001/tcp   dockge
```

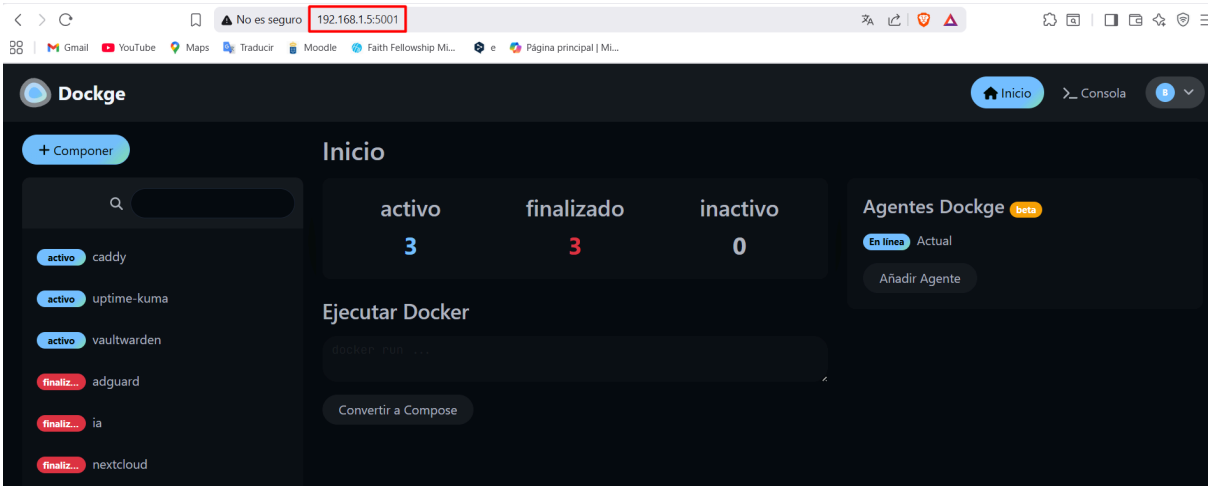
Configuración

Acceso a la interfaz y creación de servicios

En nuestra red local accedemos a:

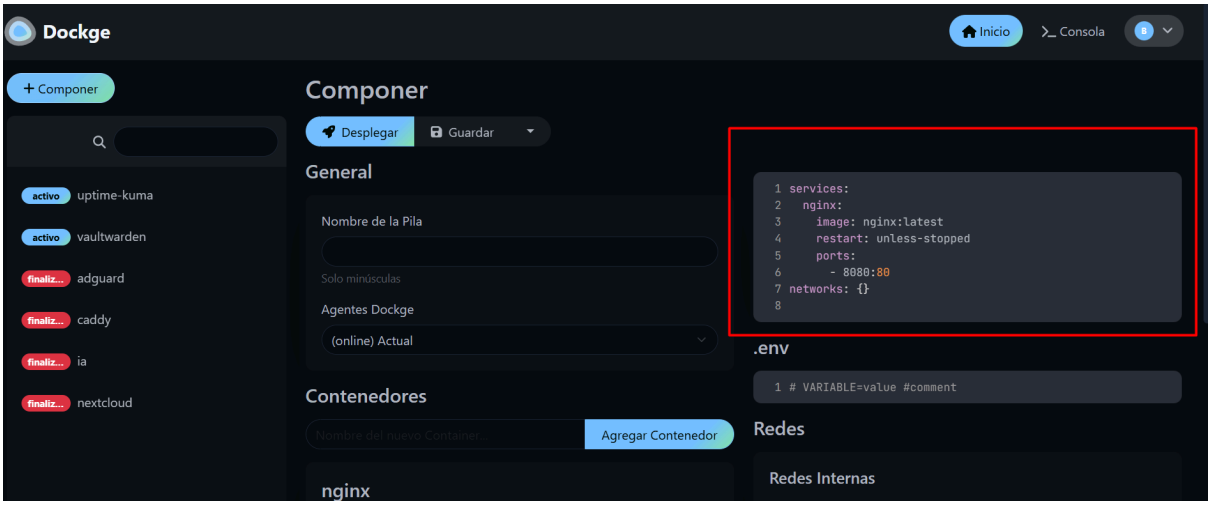
```
http://192.168.1.5:5001
```

Y una vez dentro la primera vez creamos un usuario y una contraseña y de esta manera ya tendremos acceso al panel de administración de Dockge.

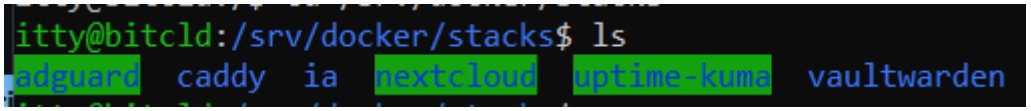


Crear stack nuevo desde la UI

Cuando vayamos a crear un servicios mediante la UI deberemos seleccionar: + componer, introducir el docker compose y darle a desplegar. Una vez desplegado el servicio. Dockge genera la carpeta del servicio en /srv/docker/stacks, el compose.yml y los volúmenes automáticamente.



Una vez desplegado el servicio. Dockge genera la carpeta del servicio en `/srv/docker/stacks`, el compose.yml y los volúmenes automáticamente.



Importar uno existente

Para importar uno existente en caso de ser necesario.

Deberemos crear una carpeta dentro de `/srv/docker/stacks/<nombre-del-servicio>`, colocar su compose.yml y pegar los volúmenes si queremos conservar la información. Una vez creada Dockge lo detectará y podrá gestionarlo. Para una mejor organización y gestión lo mejor es asegurarnos que cada servicio tiene su propia carpeta.

Flujo de trabajo

Para nuestro uso, el flujo de trabajo si deseamos añadir más servicios sería:

1. Creemos u organicemos las carpetas de cada servicio en `/srv/docker/stacks`.

2. Desde Dockge, crearemos o importaremos el stack.

3. Editaremos el compose.yml desde la UI de Dockge ya que es mucho más cómodo y escalable.

4. Desplegaremos nuestro servicio, y si fuera necesario podríamos iniciar o pausarlos directamente desde el panel.

5. Repetiremos por cada servicio que queramos añadir a nuestro ecosistema.

Conclusión

Dockge es uno de los servicios más esenciales de los que disponemos, ya que podríamos considerarlo como el centro de mando de bitCLD desde donde podemos arrancar, reiniciar y todas las demás funciones que necesitemos. Además, el poder gestionar los compose.yml directamente desde su interfaz siendo muy intuitivo.

4.3.2 Adguard Home

Introducción

Adguard Home es un servidor DNS local, de código abierto que se encarga de bloquear publicidad, rastreadores y sitios potencialmente peligrosos. A diferencia de los bloqueadores convencionales, Adguard Home los bloquea antes de que lleguen al ordenador.

Motivos de elección

Elegimos Adguard Home debido a que nos permite personalizar qué deseamos bloquear mediante listas personalizadas y sobretodo por ser de código abierto ya que nos garantiza transparencia.

Función dentro del proyecto

En bitCLD, Adguard Home funciona como nuestro servidor DNS interno, se encuentra desplegado como un contenedor y es gestionado por Dockge. En nuestro caso no lo usamos a nivel de red, sino que conectamos nuestros dispositivos a él.

Instalación

En Dockge añadimos el compose.yml de Adguard home y le damos a desplegar:

```
services:
adguardhome:
    image: adguard/adguardhome:latest
    container_name: adguardhome
    restart: unless-stopped
    ports:
    - 192.168.1.5:53:53/tcp
    - 192.168.1.5:53:53/udp
    - 100.71.42.126:53:53/tcp
    - 100.71.42.126:53:53/udp
    - 192.168.1.5:3000:3000/tcp
    volumes:
    - ./workdir:/opt/adguardhome/work
    - ./confdir:/opt/adguardhome/conf
    environment:
    - TZ=Atlantic/Canary
    networks:
    - caddy_net
networks:
caddy_net:
    external: true
```

Explicación del docker compose

Parámetro	**Descripción**
image: adguard/adguardhome:latest	Usa la última versión oficial de la imagen de Adguard Home disponible desde Docker Hub.
container_name: adguardhome	Asigna un nombre fijo al contenedor (útil para comandos y depuración).
restart: unless-stopped	Hace que el contenedor se reinicie automáticamente si se detiene de forma inesperada, salvo que se haya detenido manualmente.
- 192.168.1.5:53:53/tcp - 192.168.1.5:53:53/udp	DNS local en LAN, publica DNS sobre TCP y UDP en la IP LAN del host.
- 100.71.42.126:53:53/tcp - 100.71.42.126:53:53/udp	DNS accesible por Tailscale, Publica DNS sobre TCP y UDP, en la IP Tailscale del host
- 192.168.1.5:3000:3000/tcp	Panel web para administración, Expone la UI del asistente en :3000 por la IP LAN.
- TZ=Atlantic/Canary	Zona horaria del contenedor afecta a logs y estadísticas.
caddy_net	El servicio se conecta a la red caddy_net, lo que permite que Caddy se comunique con otros contenedores
external: true	Indica que la red caddy_net ya existe y no

	será creada por este docker-compose. Esto permite compartir la misma red entre distintos docker-compose.yml, haciendo posible que varios contenedores se comuniquen entre sí.

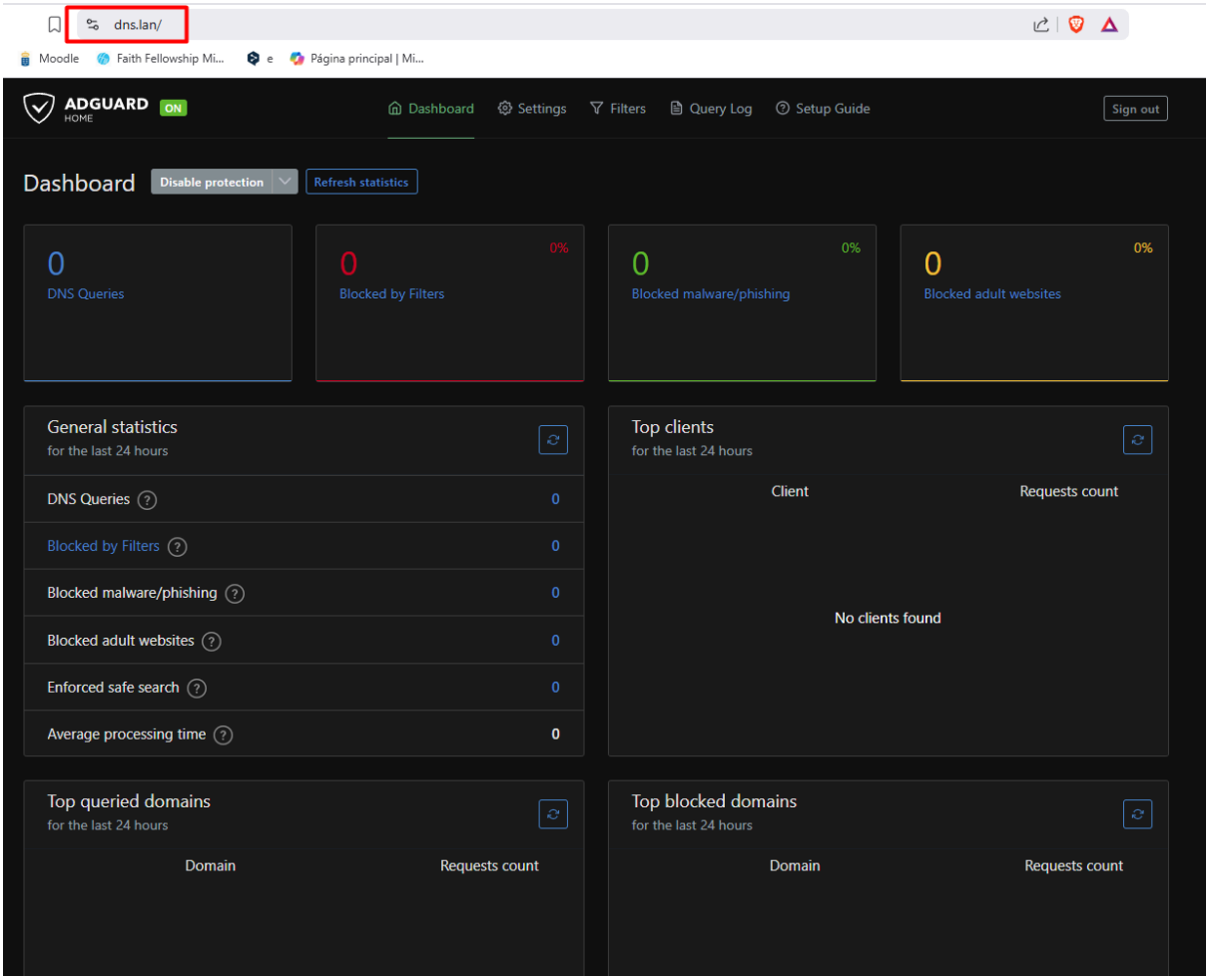
Configuración

1. Acceso a la interfaz

En nuestra red local accedemos a `http://192.168.1.5:3000`, Creamos un usuario y una contraseña y ya tendremos acceso al panel de administración de Adguard Home.

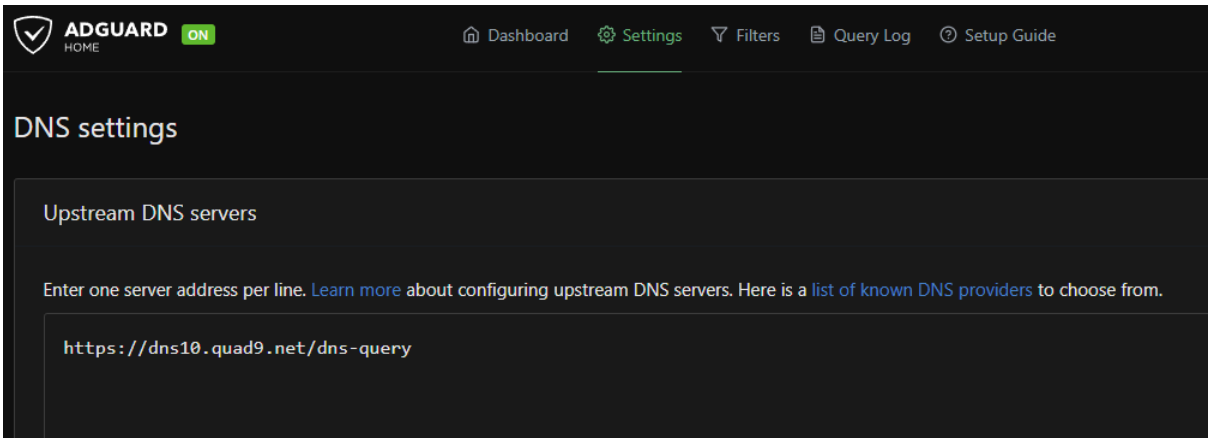
Una vez configurado Caddy server, podremos acceder a la interfaz desde:

`https://dns.lan`



Upstreams DNS

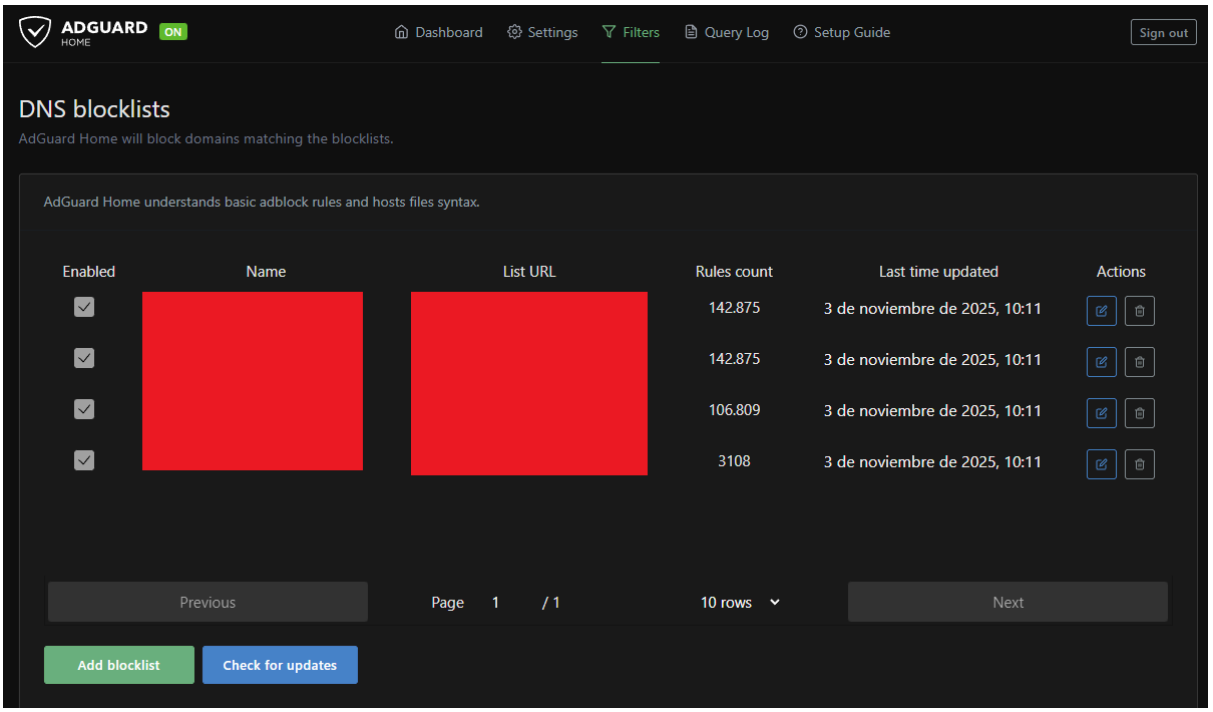
Los upstreams son los servidores externos a los que AdGuard consulta cuando no tiene la respuesta en caché. Por privacidad, seguridad y rendimiento se ha seleccionado el DNS de Quad 9.



Listas de bloqueo

Las listas de bloqueo son un conjunto de dominios, si un dominio aparece en una lista de bloqueo el servidor DNS impide que la conexión se realice. De esta manera evitamos que los anuncios, rastreadores, mineros o webs maliciosas se carguen siquiera, antes de que lleguen al navegador.

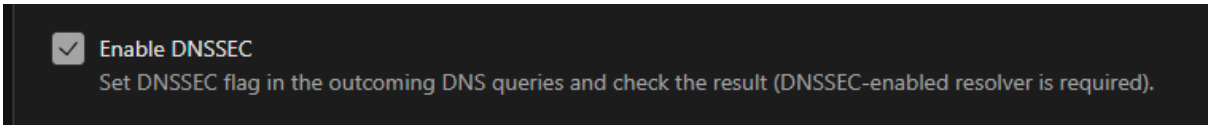
Para establecer listas de bloqueo deberemos irnos a Filters > DNS blocklist > Add > Add a custom list. Y añadimos las listas que consideremos.



Existen una gran cantidad de ellas, lo ideal es ir probando y añadir o quitar las que no nos interesan. Debemos evitar combinar listas enormes redundantes si no necesitamos agresividad extrema ya que podría romper ciertos sitios.

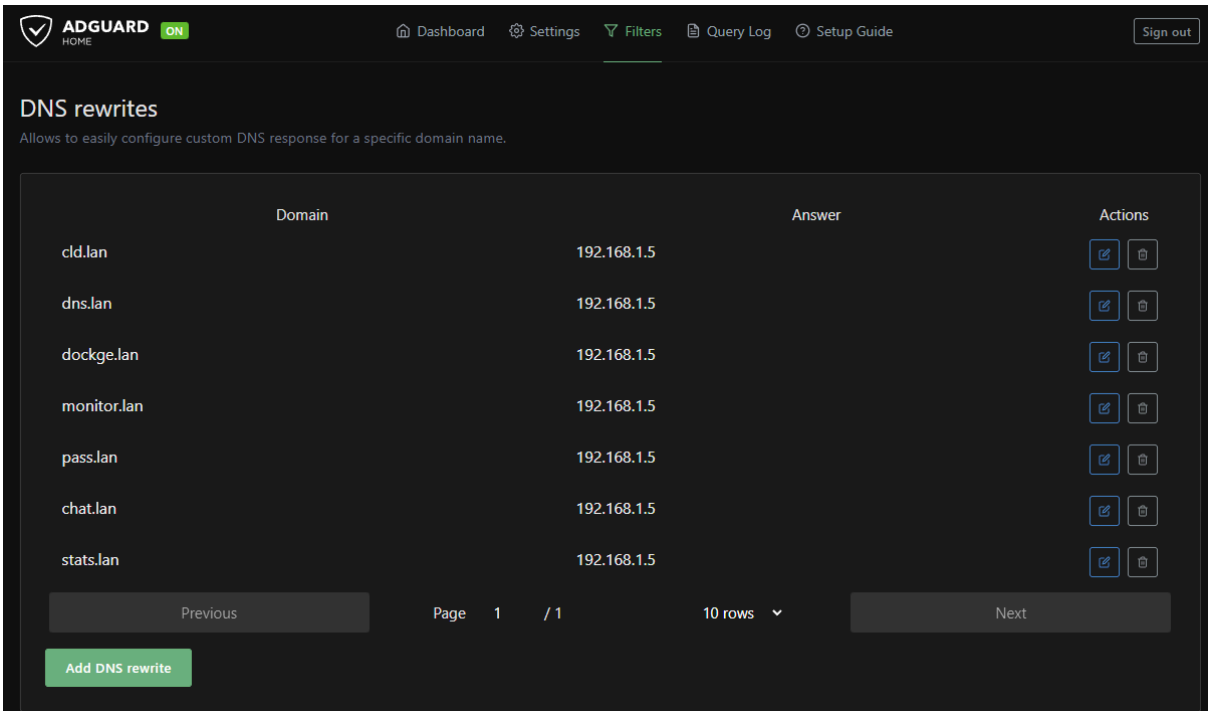
Activar DNSSEC

El DNSSEC (Domain Name System Security Extensions) nos añade una capa de verificación criptográfica a las respuestas DNS.
Para activarlo deberemos ir a Settings → DNS Settings → Enable DNSSEC



DNS Rewrites: para dominios internos .lan

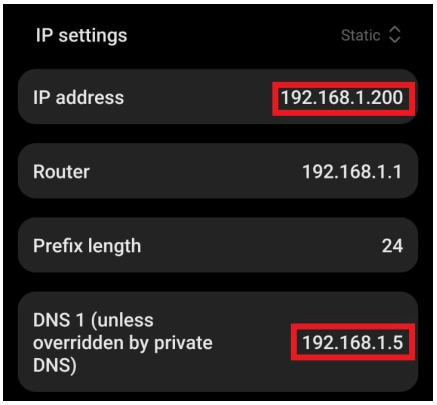
Un DNS rewrite sustituye una dirección de dominio por una IP fija. En nuestro caso, los usamos para que nuestros servicios internos apunten a nuestro servidor local 192.168.1.5:



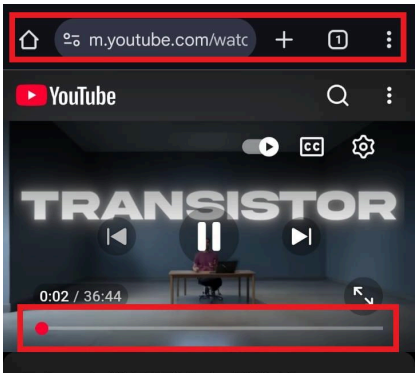
Esto facilita la integración con Caddy, que recibe el tráfico local y lo distribuye al contenedor correspondiente por la red interna Docker (caddy_net).

Funcionamiento de Adguard Home

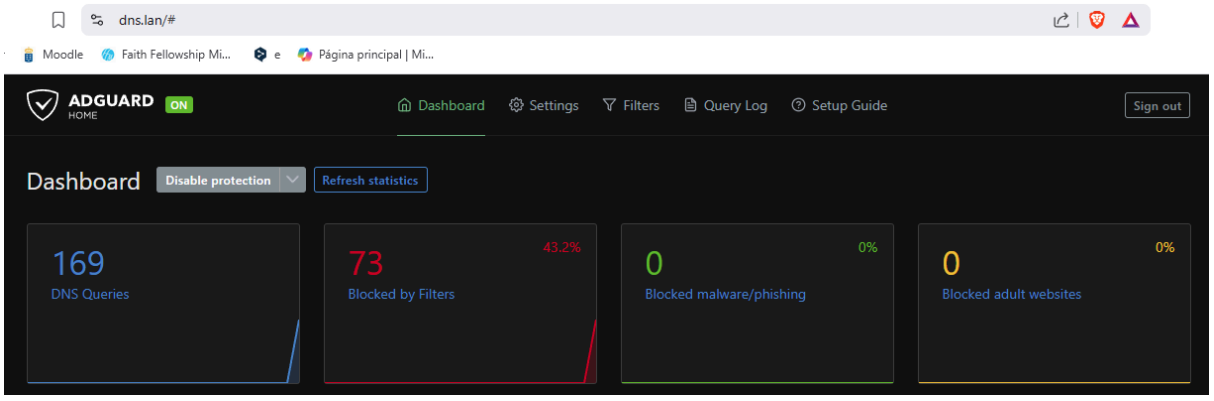
Una vez tengamos todo configurado ya podremos usar nuestro servidor DNS, para ello deberemos establecerlo como nuestro DNS principal y a partir de ese momento ya podremos tener todas las ventajas que nos ofrece Adguard Home.

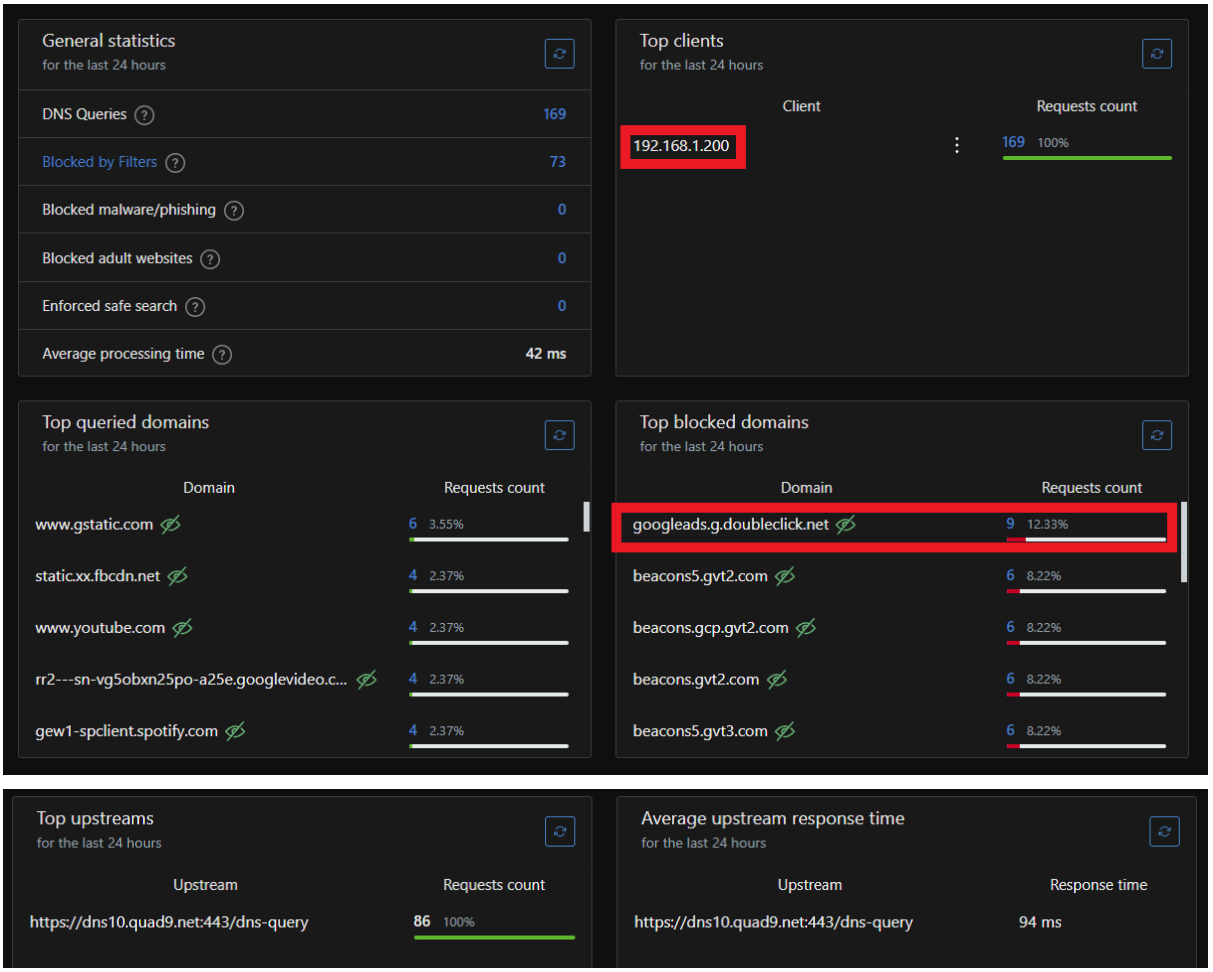


Una vez establecido como servidor DNS, Adguard Home bloqueará anuncios y rastreadores como por ejemplo los anuncios en los vídeos de YouTube, siempre y cuando se abra en el navegador y no en la aplicación:



Y desde el propio panel podemos observar qué dominios ha permitido y cuales ha bloqueado:





Conclusión

Adguard Home es un servicio esencial para añadir a nuestro ecosistema ya que nos ofrece privacidad y control sobre nuestras consultas de una manera muy simple. Además al ser autoalojado tenemos la oportunidad de iniciarlo o detenerlo fácilmente desde Dockge. Es por ello que es una gran incorporación a bitCLD ya que respeta los cuatro pilares del proyecto.

4.3.3 Nextcloud

Introducción

Nextcloud es una plataforma de código abierto tanto de almacenamiento como de sincronización y colaboración en la nube, está diseñada para darnos la oportunidad de tener una alternativa privada y auto alojada a servicios como Google Drive o Dropbox. Nos permite alojar, gestionar y compartir archivos, calendarios, contactos, y añadir más aplicaciones como mapas, multimedia.. Todo bajo control total del propio servidor.

Motivo de elección

Elegimos Nextcloud como solución de almacenamiento debido sobretodo a que es de código abierto, su extensa documentación y a la posibilidad de desplegarlo mediante docker compose de forma nativa, lo cuál nos facilita enormemente la gestión gracias a Dockge y se mantiene en sincronía con todo el ecosistema. Además, Nextcloud es un ecosistema en sí mismo ya que nos ofrece la posibilidad de añadir muchas más funcionalidades como calendarios, notas, edición colaborativa, mapas... Lo cuál es un añadido de valor a bitCLD. Otro aspecto por el cuál nos decidimos fue por su compatibilidad multiplataforma, ya que existe tanto una aplicación oficial para móvil, como para escritorio con la opción de sincronización automática, esto nos facilita enormemente su uso.

Es por todo ello que cumple con los pilares de nuestro proyecto: simplicidad, seguridad, autohospedaje y código abierto.

Función dentro del proyecto

Dentro de bitcld, Nextcloud cumple principalmente la función de nube personal para almacenar archivos y datos sensibles o vitales. Desgraciadamente debido a nuestras limitaciones técnicas actuales no podemos hacer uso de otras funciones como los mapas o el trabajos colaborativos. Sin embargo, nos sirve para poder ver las enormes posibilidades que nos permite y familiarizarnos con la tecnología. Además, en estos momentos para este proyecto sólo necesitamos hacer uso del almacenamiento y gracias a Cloudflare tunnel y Access podemos tener acceso de forma remota y segura, al igual que con la aplicación móvil usando Tailscale.

Gracias a esto Nextcloud se convierte en nuestra nube personal privada, segura y autohosteada donde mantenemos el control de todos nuestros datos.

Instalación

Nextcloud lo despliegamos fácilmente como contenedor Docker y lo gestionamos desde Dockge, manteniendo la coherencia y modularidad del ecosistema bitCLD.

Desde la interfaz de Dockge, creamos una nueva pila (stack) llamada nextcloud y añadimos el siguiente contenido en su archivo docker-compose.yml:

```
services:
app:
    image: nextcloud:latest
    container_name: nextcloud-app
    restart: unless-stopped
    ports:
    - 8080:80
    volumes:
    - /srv/docker/stacks/nextcloud/config:/var/www/html/config
    - /mnt/dexterno/nextcloud/data:/var/www/html/data # data en disco externo
    depends_on:
    - db

db:
    image: mariadb:10.8
    container_name: nextcloud-db
    restart: unless-stopped
    command: --transaction-isolation=READ-COMMITTED --binlog-format=ROW
    environment:
    - MYSQL_ROOT_PASSWORD=<contraseña segura>
    - MYSQL_DATABASE=<nombre de la base de datos>
    - MYSQL_USER=<usario de la base de datos>
    - MYSQL_PASSWORD=<contraseña segura>
    volumes:
    - /srv/docker/stacks/nextcloud/db:/var/lib/mysql
```

Explicación del docker compose

Parámetro	Descripción
image: nextcloud:latest	Usa la última versión oficial de la imagen de Nextcloud desde Docker Hub.
container_name: nextcloud-app	Asigna un nombre fijo al contenedor (útil para comandos y depuración).
restart: unless-stopped	Hace que el contenedor se reinicie automáticamente si se detiene de forma inesperada, salvo que se haya detenido manualmente.
ports: - 8080:80	Expone el puerto 80 del contenedor y accederemos a su interfaz desde el puerto 8080.
-/srv/docker/stacks/nextcloud/config:/var/www/html/config	Carpeta de configuración del contenedor localizada en el almacenamiento interno.
-/mnt/dexterno/nextcloud/data:/var/www/html/data	Carpeta de almacenamiento de nuestras fotos y archivos en el disco duro externo.

depends_on: - db	Indica que no inicie el servicio de Nextcloud sin antes que antes se haya iniciado la base de datos para evitar errores.
image: mariadb:10.8	Usa la versión 10.8 de la imagen de MariaDB desde Docker Hub. No usamos la última versión ya que una actualización nos podría romper algo.
container_name: nextcloud-db	Asigna un nombre fijo al contenedor (útil para comandos y depuración).
restart: unless-stopped	Hace que el contenedor se reinicie automáticamente si se detiene de forma inesperada, salvo que se haya detenido manualmente.
--transaction-isolation=READ-COMMITTED	Gestiona cómo se bloquean los datos cuando se leen. Evita que la base de datos se bloquee cuando hay muchas peticiones a la vez.
--binlog-format=ROW	Formato de registro binario. Necesario para funciones avanzadas y estabilidad de Nextcloud. Sin esta línea, Nextcloud te daría advertencias o errores graves.
- MYSQL_ROOT_PASSWORD=superseguro	En este campo indicamos la contraseña root de nuestra base de datos
- MYSQL_DATABASE=nextcloud	En este campo indicamos el nombre de nuestra base de datos
- MYSQL_USER=nextcloud	En este campo indicamos el nombre de usuario en nuestra base de datos
- MYSQL_PASSWORD=contraseña	En este campo indicamos la contraseña de nuestra base de datos
-/srv/docker/stacks/nextcloud/db:/var/lib/mysql	Persistencia de la base de datos.

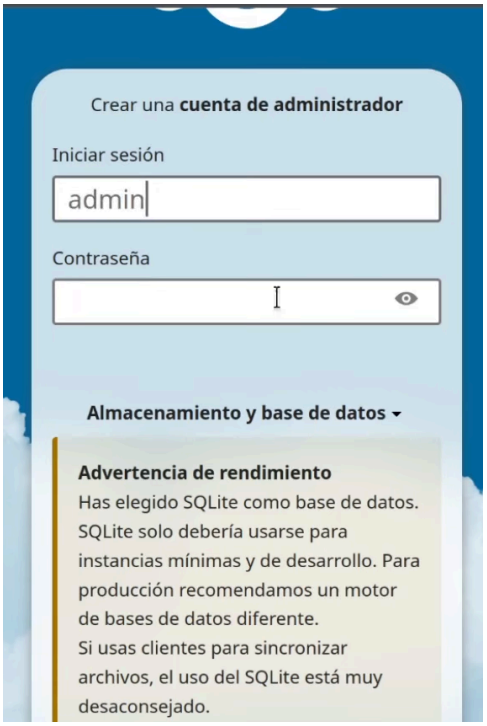
Es necesario que este último volumen esté en el SSD interno y no en el disco duro externo ya que Las bases de datos necesitan hacer muchas operaciones pequeñas y rápidas. Por lo que si lo ponemos en el disco externo, Nextcloud nos irá lentísimo.

Configuración

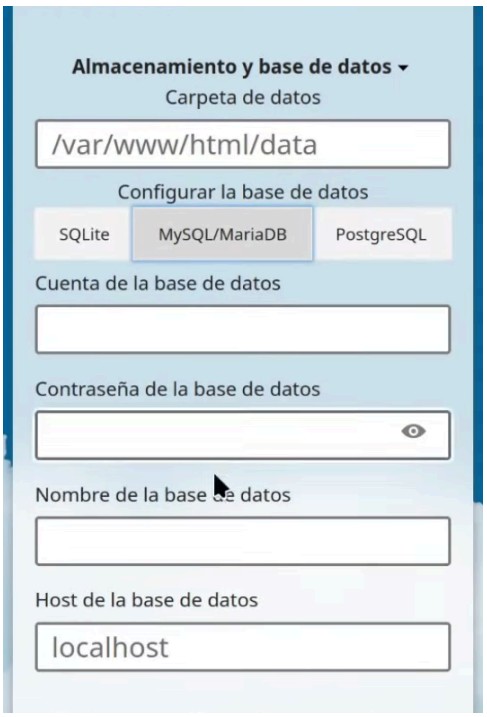
Una vez lo hayamos desplegado e iniciado desde Dockge podremos acceder a `http://192.168.1.5:8080` para configurarlo.

1. Creamos cuenta de administrador

Para empezar deberemos crear una cuenta de administrador de Nextcloud en la que tendremos que poner un nombre de usuario y una contraseña:

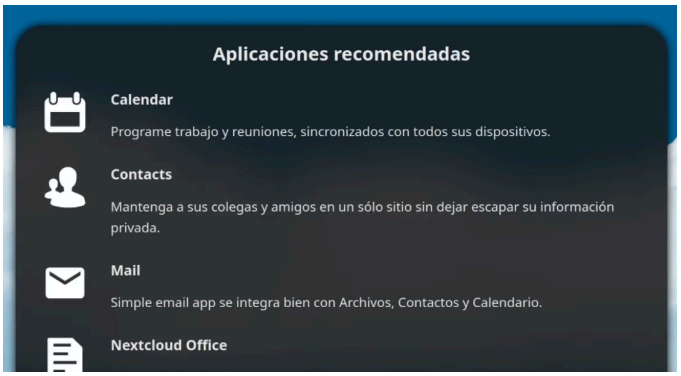


Por defecto tendremos como base de datos a SQLite, la cuál incluso Nextcloud nos la desaconseja ya que es muy limitada. Por lo que le daremos a la opción de Storage & database y seleccionaremos la opción MySQL/MariaDB y añadiremos los datos que hemos puesto anteriormente en el docker compose en db:

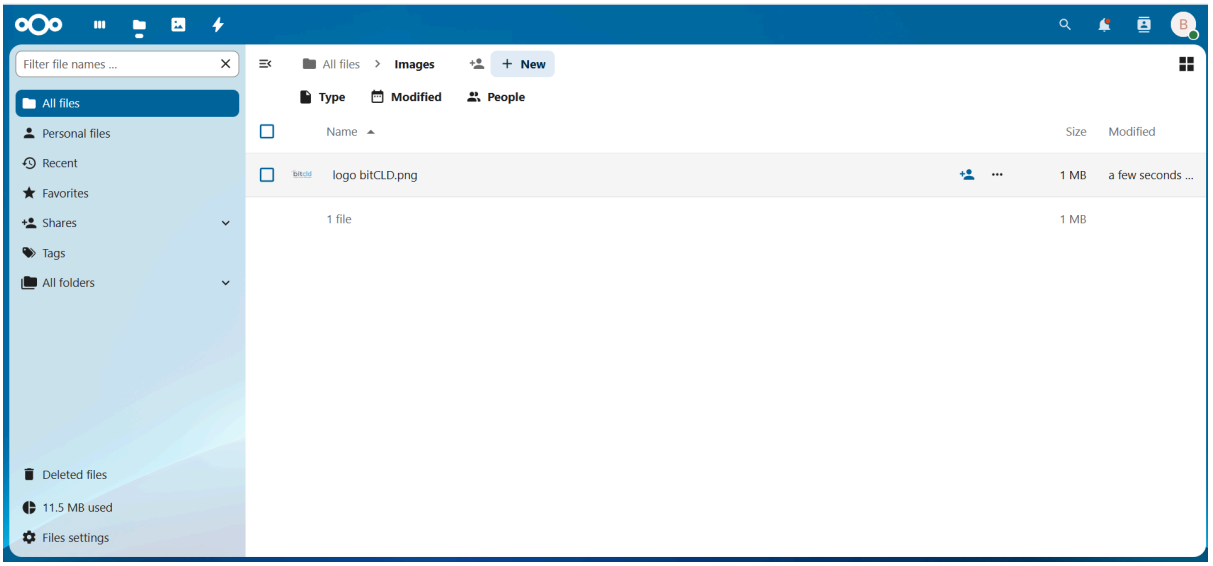


En la opción de LocalHost deberemos añadir db ya que es el nombre que le hemos puesto a nuestro servicio.

En la opción de añadir aplicaciones en nuestro caso, al ser usada sólo como almacenamiento, seleccionaremos contacs y calendar.

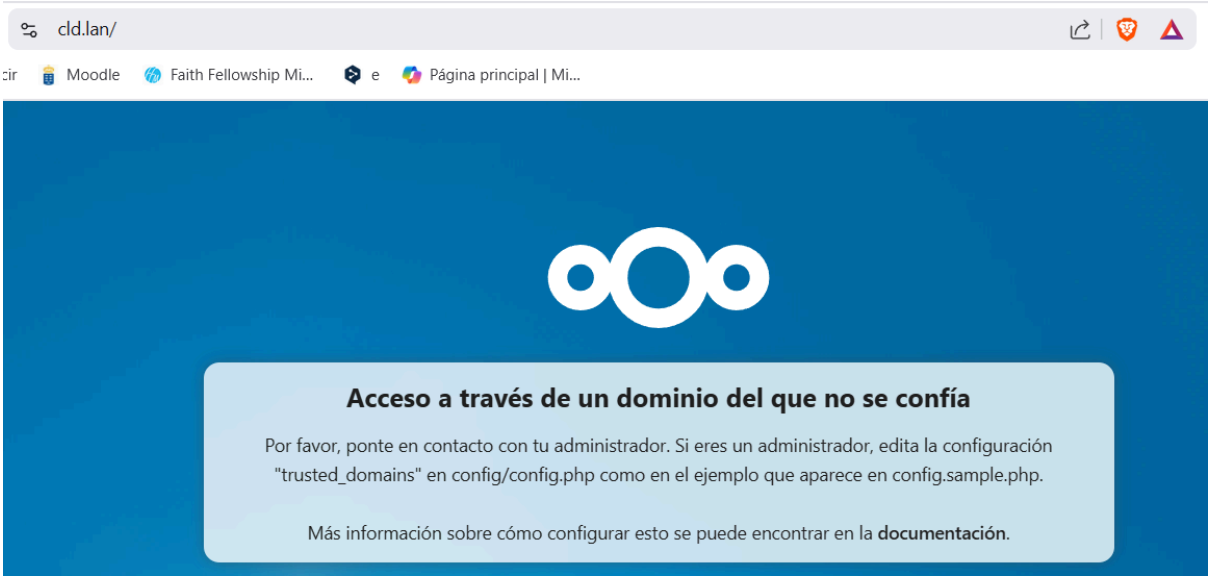


Y ya podremos acceder a nuestro Nextcloud y guardar nuestros archivos



2. Ajustar config.php

Sin embargo, aunque tengamos configurado caddy o cloudflare no podremos acceder a Nextcloud mediante `cld.lan` o cld.bitcld.com :

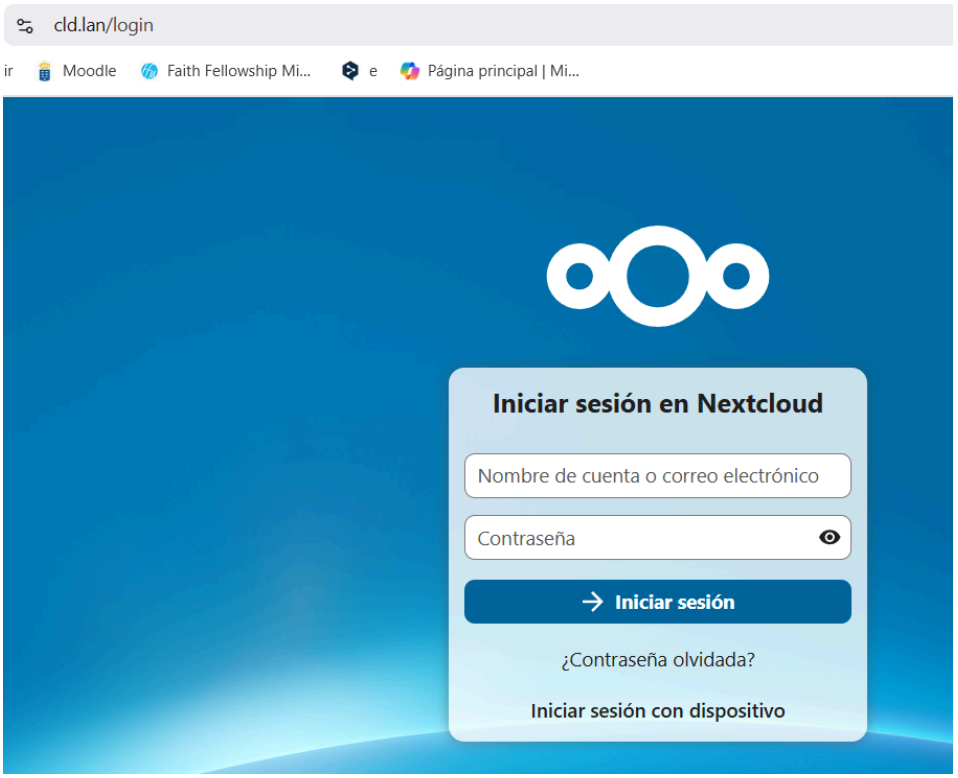


Para solucionar esto deberemos editar el archivo config.php con:

```
sudo nano /srv/docker/stacks/nextcloud/config/config.php
```

Y dentro del archivo buscaremos `'trusted_domains' array` y añadiremos los dominios desde donde accederemos a Nextcloud:

De esta manera ya podremos acceder desde `cld.lan` o cld.bitcld.com :



Aplicación para móviles

Para que podamos usar la aplicación móvil es muy simple pero deberemos tener cuenta factores de seguridad:

Existen dos maneras de hacerlo añadiendo la ruta a nuestro dominio `cld.bitcld.com` o a la ip que nos ofrece Tailscale. Ambos tienen sus ventajas y desventajas.

cld.bitcld.com

Si queremos usar nuestro dominio personalizado como ruta nos ofrece la ventaja que no deberemos tener que estar conectados a Tailscale cada vez que queramos acceder o subir algún archivo. Sin embargo, esta opción nos obligaría a quitar Access para la aplicación de nextcloud o añadir una regla de Bypass para la aplicación lo cuál es más complejo.

Es necesario hacer una de las dos opciones ya que con Access activado la aplicación no sabe como gestionar la pantalla extra de verificación de email y código. Por lo que nos da error.

Tailscale

La otra opción es usar la ip que nos ofrece Tailscale. Al usar esta opción no tendremos que modificar nada en el Cloudflare Access y es más simple. La principal desventaja como hemos comentado, es que tendremos que estar siempre conectados a la VPN para acceder a la app.

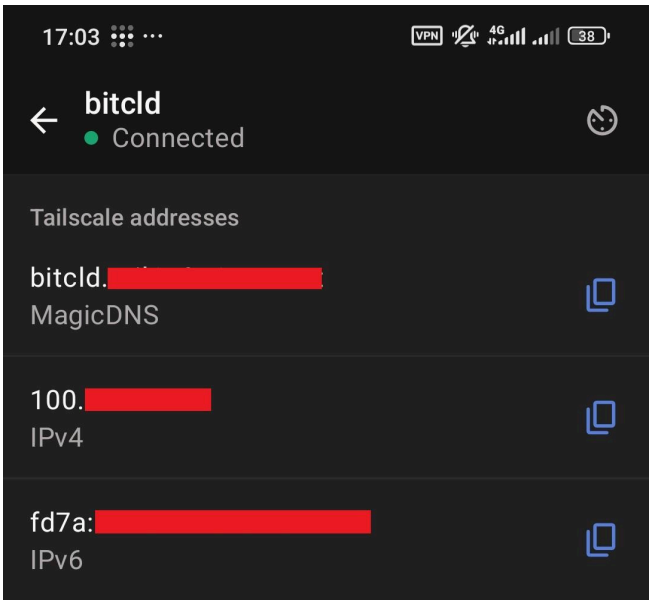
Decisión

Nosotros por comodidad y simplicidad hemos decidido usar la opción de Tailscale, para así poder seguir teniendo la aplicación de nextcloud protegida con Access o no tener que estar modificando las reglas. Ya que para nuestro uso no es un inconveniente activar la VPN cuando vayamos a usar la aplicación.

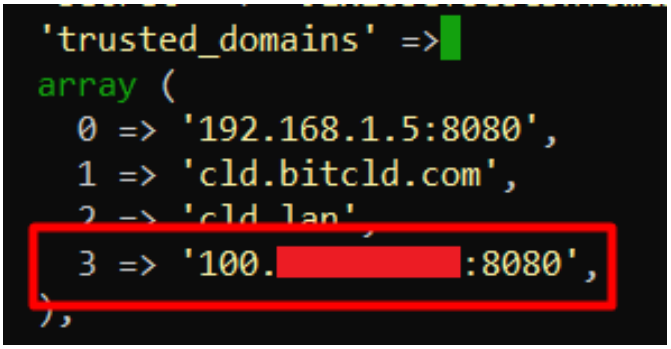
Configuración de la APP

Para configurar la aplicación mediante Tailscale:

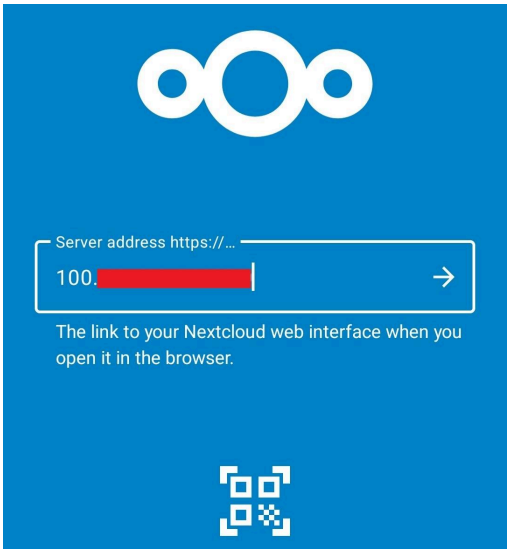
1. Buscaremos la ip de nuestro servidor que nos da Tailscale:



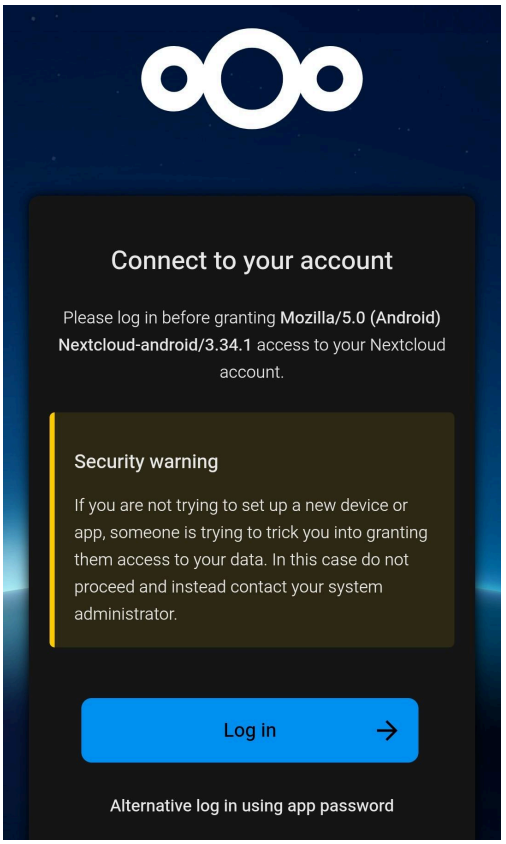
2. Añadiremos la ruta de la ip con el puerto 8080 de Tailscale en el archivo `/srv/docker/stacks/nextcloud/config/config.php`



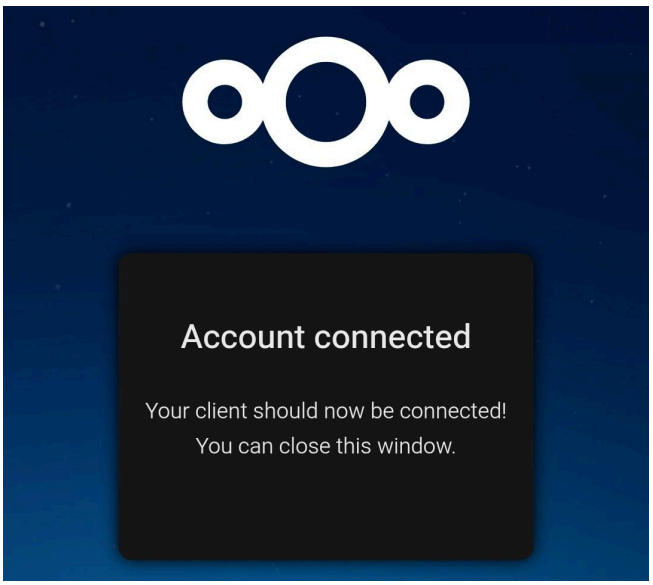
3. Conectados a la VPN pondremos la ruta en la aplicación



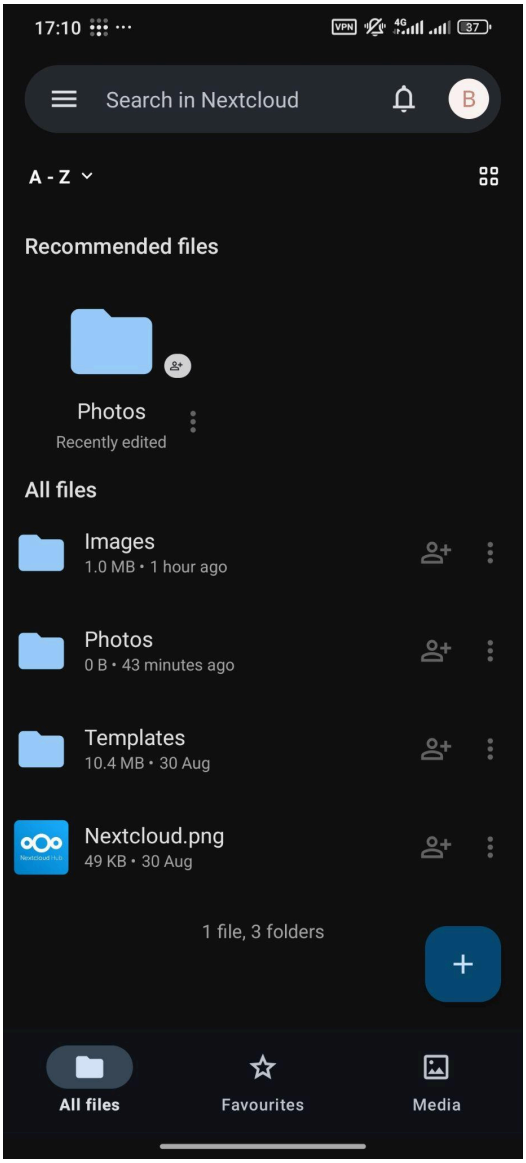
4. Para poder sincronizarla se abrirá una pestaña en nuestro navegador y deberemos iniciar sesión



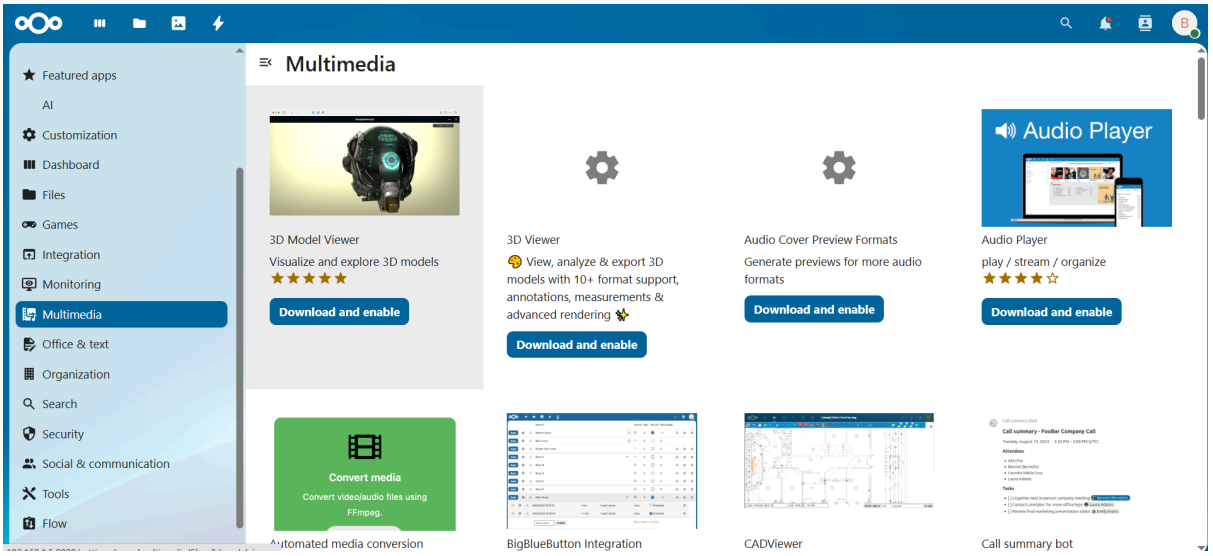
5. Una vez hecho el inicio de sesión la cuenta se conectará



Y ya podremos acceder a Nextcloud desde nuestra app



Además también podemos añadir muchas más aplicaciones para completar nuestra suit con Nextcloud, existen aplicaciones de mapas, de multimedia, para colaboración de documentos... Desgraciadamente con nuestro servidor actual estamos muy limitados y no podemos acceder a más funciones que nos ofrece Nextcloud.



Conclusión

Nextcloud, en bitCLD, es nuestra solución para poder tener un almacenamiento personal, privado y seguro. Podemos acceder a través del subdominio cld.bitcld.com y cld.lan y es gestionada mediante Dockge en su propio contenedor docker-compose. Gracias a Nextcloud podemos disponer de una nube propia sin depender de otros servicios externos a nuestro control, lo que nos garantiza un control total de nuestros datos en nuestro disco duro externo que puede ser ampliable.

Gracias a nuestra solución de VPN (Tailscale), podemos usar la aplicación oficial de Nextcloud en el móvil para sincronizar y acceder a todos nuestros archivos de forma inmediata y cifrada, con una experiencia idéntica a otras nubes como Google Drive o Outlook pero sin necesidad de abrir puertos en nuestro servidor. Con nuestra configuración, el servicio nos ofrece un almacenamiento seguro con alta disponibilidad sin añadir más complejidad innecesaria haciéndolo lo más simple posible.

4.3.4 Inteligencia Artificial

Introducción

Desde hace unos años, la inteligencia artificial se ha ido convirtiendo en una tecnología cada vez más importante en nuestras vidas. Se puede usar para muchas cosas como pedir información, programar, consultar dudas y en definitiva puede ayudarnos en nuestro día a día. Es por ello que hemos decidido implementar inteligencia en nuestro ecosistema, no sólo como un extra sino como un servicio más que nos pueda aportar valor. Para ello montamos un stack completo con modelos locales mediante Ollama y mediante API una API externa de Groq, usando Open WebUI como interfaz para ambos tipos.

Motivos de elección

Originalmente nuestra idea inicial era sólo usar modelos locales para así respetar los pilares de nuestro proyecto: simple, seguro y autoalojado. Es por ello que elegimos Ollama usando Open WebUI y descargamos modelos bastante pequeños como Phi 2.7B y Gemma 2B ya que pensamos que al ser modelos pequeños podríamos correrlo en nuestro servidor y poder mantenerlo todo autoalojado, manteniendo control de nuestros datos y sin depender de terceros. Aprendiendo sobre los modelos mientras los usarmos.

Sin embargo, el rendimiento no fue el esperado con tiempos de respuesta bastante considerables y un gran consumo de CPU. Es por ello, que para no dejar la inteligencia artificial de lado y poder implementarla de manera exitosa en nuestro proyecto decidimos mantener Ollama con los modelos locales para experimentar y para un uso más diario usar la API gratuita de Groq, que nos permite tener modelos más potentes y rápidos de manera gratuita y principalmente liberar a nuestro servidor del procesamiento.

Elegimos Open WebUI como interfaz para nuestra inteligencia artificial y que nos permite cambiar fácilmente entre modelos locales y externos, nos ofrece una interfaz muy limpia y similar a otras inteligencias artificiales y porque es de código abierto la parte que nosotros usamos.

Función dentro del proyecto

Dentro de bitCLD, la inteligencia artificial actúa como nuestro asistente personal para resolver nuestras consultas al estilo ChatGPT o Deepseek pero sin gastar dinero en suscripciones y teniendo la oportunidad de tenerlo en nuestro ecosistema. Gracias a Open WebUI podemos tener una interfaz muy amigable y podemos cambiar muy fácilmente entre nuestros modelos locales y los modelos que nos ofrece Groq.

Mantenemos los modelos locales para así poder demostrar que es posible autoalojar la inteligencia artificial gracias a Docker y para hacer pruebas de cómo funciona todo para que cuando tengamos la oportunidad de tener un servidor más potente ya sepamos como funciona todo.

Por otra parte, usamos Groq para conseguir respuestas más completas y rápidas sin sobrecargar nuestro minipc y también para poder aprender sobre el uso de las APIs y su integración en proyectos como bitCLD.

Instalación

Tanto WebUI y como Ollama los desplegamos fácilmente como contenedor Docker y lo gestionamos desde Dockge, manteniendo la coherencia y modularidad del ecosistema bitCLD.

Desde la interfaz de Dockge, creamos una nueva pila (stack) llamada ia y añadimos el contenido en su archivo docker-compose.yml:

```
services:
ollama:
    image: ollama/ollama:latest
    container_name: ollama
    restart: unless-stopped
    ports:
    - 11434:11434
    volumes:
    - ollama:/root/.ollama
    deploy:
    resources:
        limits:
        cpus: "2.0"
        memory: 8G
        reservations:
        cpus: "1.0"
        memory: 6G
open-webui:
    image: ghcr.io/open-webui/open-webui:main
    container_name: open-webui
    restart: unless-stopped
    depends_on:
    - ollama
    environment:
    - OLLAMA_API_BASE=http://ollama:11434
    ports:
    - 3002:8080
    deploy:
    resources:
        limits:
        cpus: "1.0"
        memory: 2G
        reservations:
        cpus: "0.5"
        memory: 1G
volumes:
ollama: null
```

Explicación del docker compose

Ollama

Parámetro	Descripción
image: ollama/ollama:latest	Usa la última versión oficial de la imagen de Ollama desde Docker Hub.
container_name: ollama	Asigna un nombre fijo al contenedor (útil para comandos y depuración).
restart: unless-stopped	Hace que el contenedor se reinicie automáticamente si se detiene de forma inesperada, salvo que se haya detenido manualmente.
ports: - 11434:11434	Puerto interno de Ollama para comunicación con Open WebUI
volumes: - ollama:/root/.ollama	Volumen donde Ollama guarda: Modelos descargados, configuración y datos persistentes
limits: cpus: "2.0"	Indica que este contenedor no debería usar más de 2 CPU lógicas
memory: 8G	Indica que un límite máximo de 8 GB de RAM para este contenedor
reservations:cpus: "1.0"	Reserva 1 CPU lógica “prioritaria” para este contenedor
memory: 6G	reserva 6 GB de RAM para él

Open WebUI

Parámetro	Descripción
image: ghcr.io/open-webui/open-webui:main	Se usa la imagen de Open WebUi y se descarga desde el registry de GitHub Container Registry de la rama principal del proyecto
container_name: open-webui	Asigna un nombre fijo al contenedor (útil para comandos y depuración).
restart: unless-stopped	Hace que el contenedor se reinicie automáticamente si se detiene de forma inesperada, salvo que se haya detenido manualmente.
depends_on: - ollama	Indica que este servicio depende de ollama por lo que docker arrancará primero ollama luego levantará open-webui

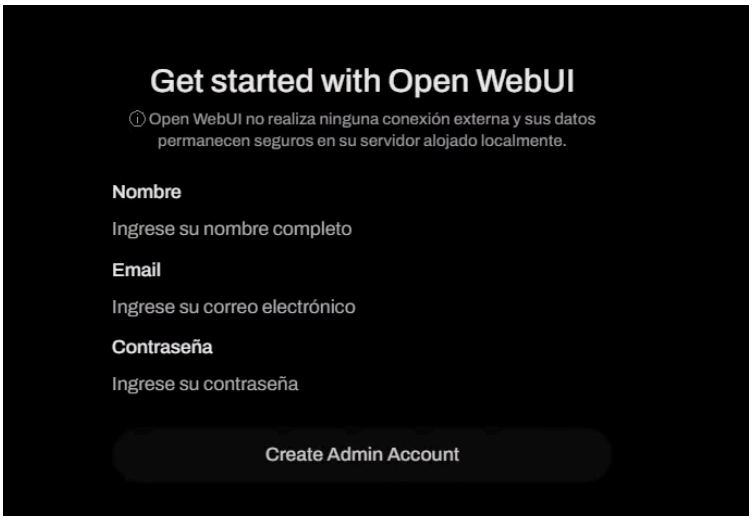
environment: - OLLAMA_API_BASE=http://ollama:11434	Indica URL base de la API de Ollama con el nombre del contenedor y el puerto interno de la API
ports: - 3002:8080	Puerto del host que se usará en el navegador, Cloudflare y Caddy
volumes: ollama: null	Docker creará un volumen persistente llamado ollama con datos persistentes

Configuración

Una vez lo hayamos desplegado e iniciado desde Dockge podremos acceder a `http://192.168.1.5:3002` para configurarlo.

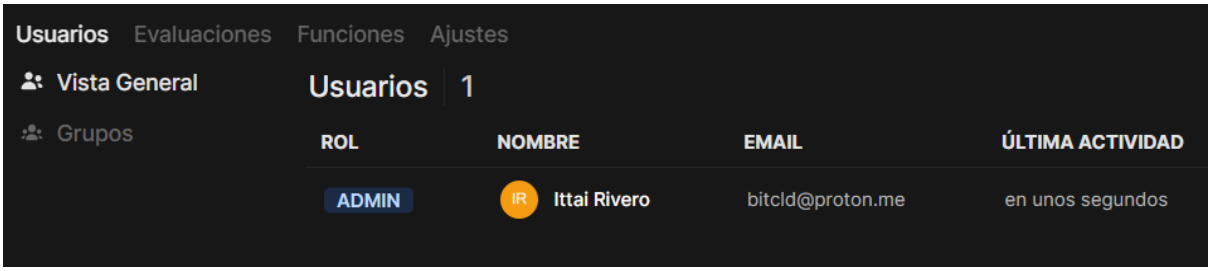
Crear cuenta de administrador

Una vez iniciemos el servicio por primera vez crearemos una cuenta, la cuál se convertirá en la de administrador. Después podremos bloquear que se creen nuevas cuentas:



Los datos que pongamos se quedarán en local, por lo que no será necesario verificar el correo.

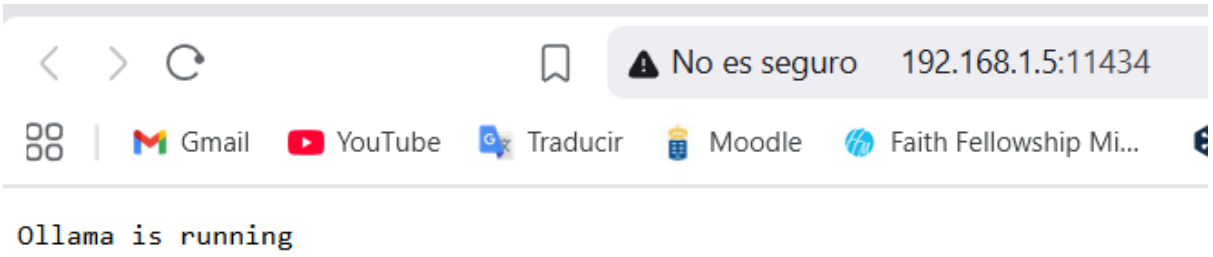
Una vez creado ya tendremos nuestra cuenta de administrador



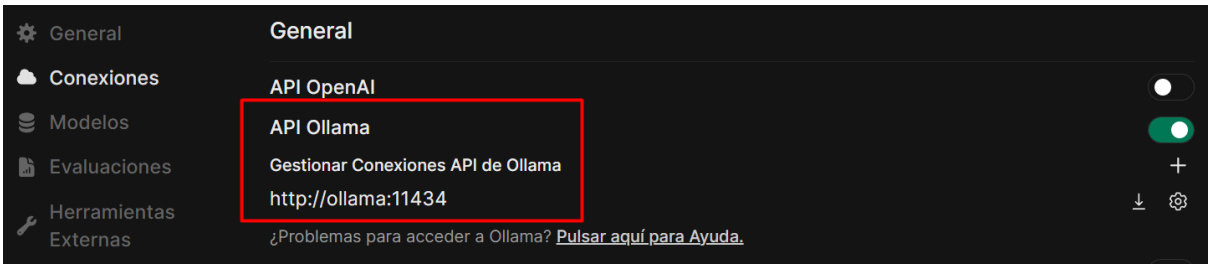
Verificar la conexión de Open WebUI con Ollama

Al añadir OLLAMA_API_BASE=http://ollama:11434 Ollama ya debería estar conectada con Open WebUI pero debemos asegurarnos para poder proseguir.

1. Nos aseguramos que ollama está corriendo accediendo a `192.168.1.5:11434`



2. Ahora dentro de Open WebUI como administrador accedemos a Ajustes > Ajustes de admin > Conexiones Podremos ver una conexión en API Ollama apuntando a `http://ollama:11434`



Descargar modelos locales

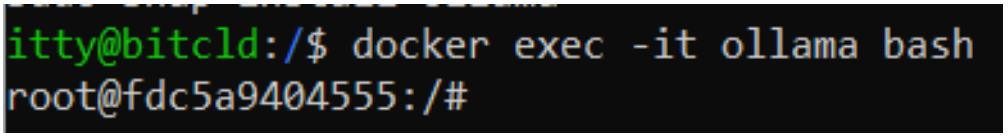
Una vez nos hayamos asegurado que existe conexión entre Ollama y Open WebUI procederemos a descargar modelos locales para correrlos en nuestro servidor.

Debido a las limitaciones de nuestro equipo descargaremos modelos pequeños como Phi ~2.7B o Gemma ~2B

Para la descarga de los modelos es muy sencilla:

1. Deberemos entrar dentro del contenedor de Ollama mediante ssh y poniendo el siguiente comando:

```
docker exec -it ollama bash
```



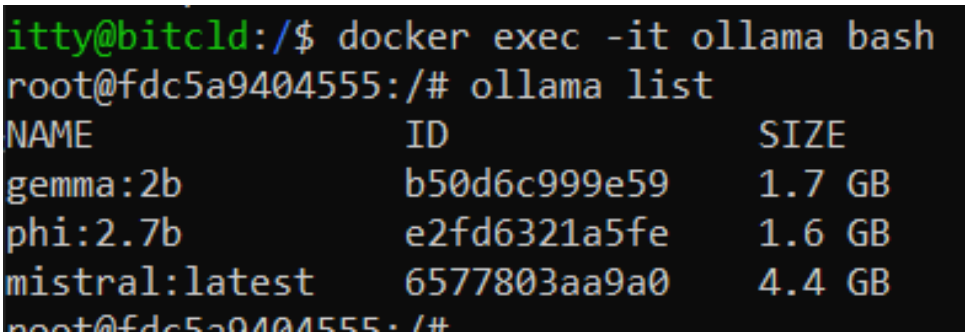
2. Descargaremos los modelos que consideremos mediante el comando

```
ollama pull nombre-del-modelo
```

Para encontrar modelos podemos usar la web oficial de ollama https://ollama.com/search

3. Para poder ver los modelos que tenemos descargados usaremos el comando

```
ollama list
```



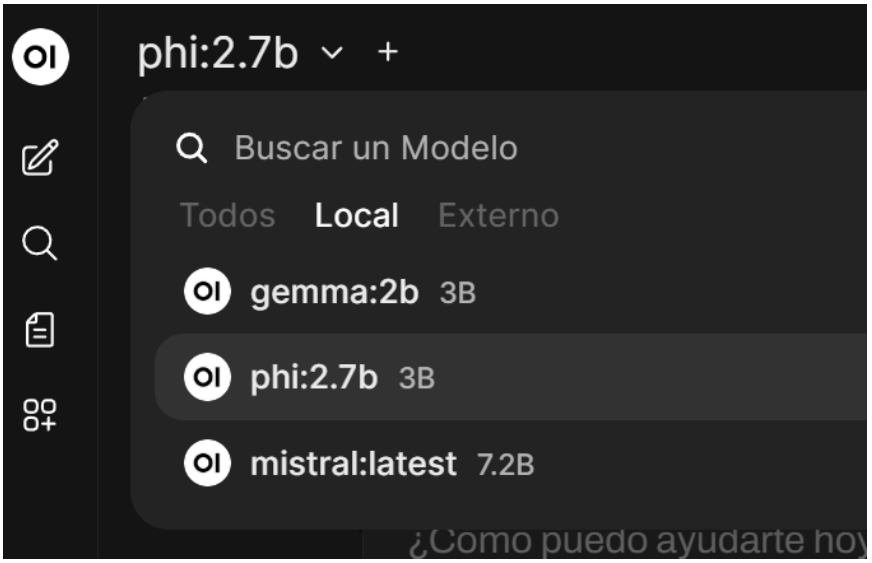
En nuestro caso, nuestro primer intento fue con mistral pero al ver que era exageradamente grande decidimos no usarlo.

4. Y salimos del contenedor con:

```
exit
```

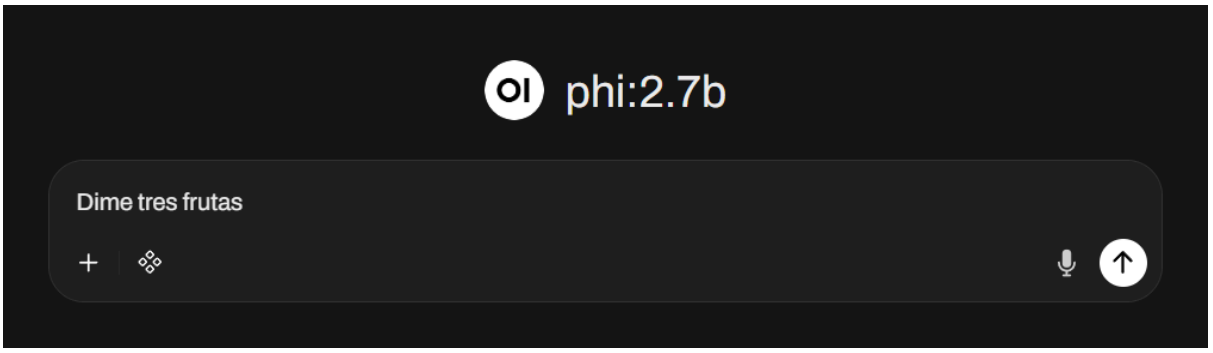
Probar los modelos

Una vez descargados los modelos, ya podremos acceder uso de ellos desde la interfaz de Open WebUI

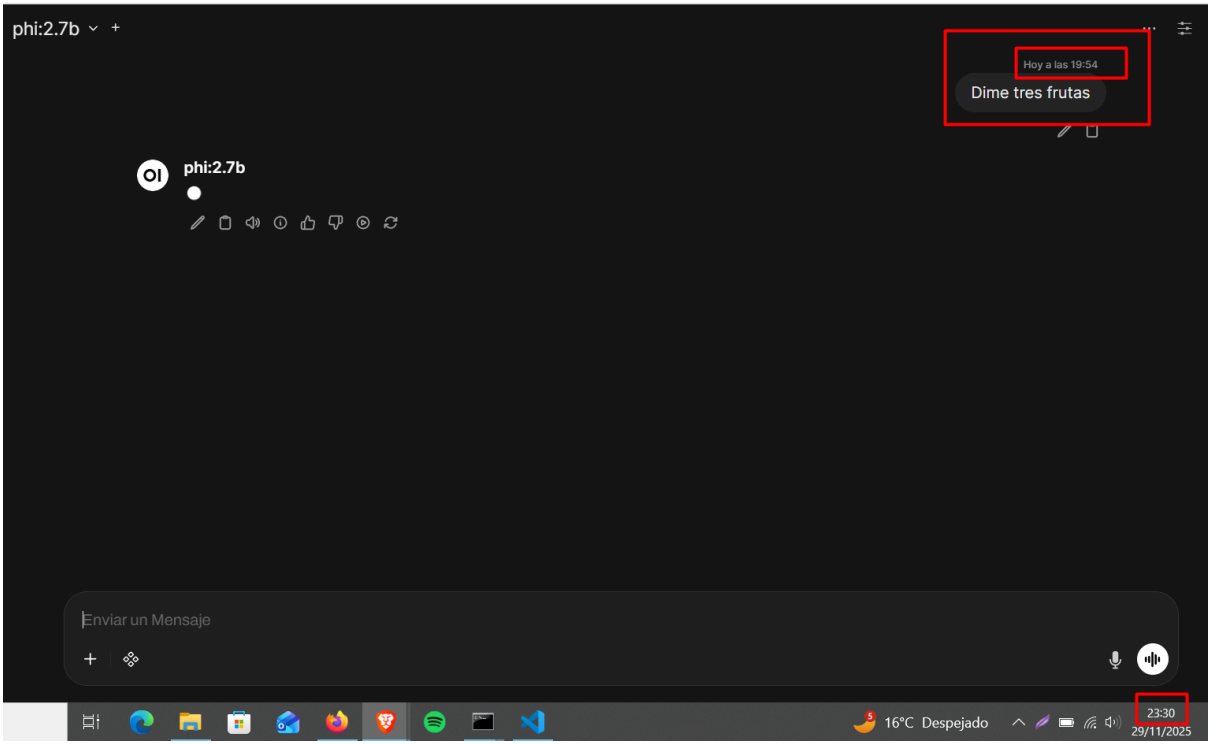


1. Primer intento

Para nuestro primer intento usaremos Phi 2.7B, por lo que hacemos una consulta simple:



Con este modelo no pudimos obtener ninguna respuesta, incluso esperando por varias horas. Habiendo hecho la consulta a las 19:54 y no obteniendo respuesta a las 23:30, por lo que definitivamente no es un modelo viable para nuestro servidor.



2. Segundo intento

Para nuestra segunda consulta usamos el modelo de Gemma esperando tener un mejor resultado.

Con este modelo obtuvimos una respuesta con unos 35 segundos de espera, por lo que será el modelo que usemos de manera local. Sin embargo, para un uso normal y cotidiano no es viable, además de que es una gran carga para nuestro servidor:

Esta es una captura de Netdata segundos antes de las consultas:

Y esta durante la consulta:

API

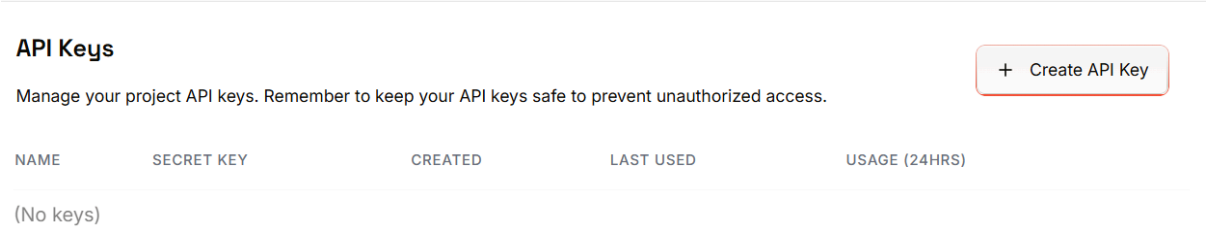
Para poder tener una experiencia más fluida y más parecida a un ChatGPT o Deepseek, hemos decidido implementar una API para que nuestro servidor sea simplemente el cliente y todo el procesamiento lo haga otro servidor. De esta manera obtendremos las ventajas de un chat mucho más ágil y sobretodo la oportunidad de poder seguir trabajando con Inteligencia Artificial y poder aprender más sobre el funcionamiento de las APIs. Sin embargo, al utilizar otros servidores para el procesamiento ya no será completamente autoalojado y perderemos el control completo de nuestros datos ya que se encargará otro servidor de realizar la consulta. Algo bastante relevante que tenemos que tener en cuenta.

Elección de API

La elección de qué API escoger fue bastante secilla, ya que al contrario que otras como ChatGPT o Deepseek por las que tienes que pagar, Groq nos da la oportunidad de tener todas las ventajas de usar una API profesional, la cual es compatible con OpenAI, pero sin tener que desembolsar nada siempre que no superemos un límite. Es por ello que elegimos la API de Groq para trabajar en nuestro proyecto.

1. Conseguir API de Groq

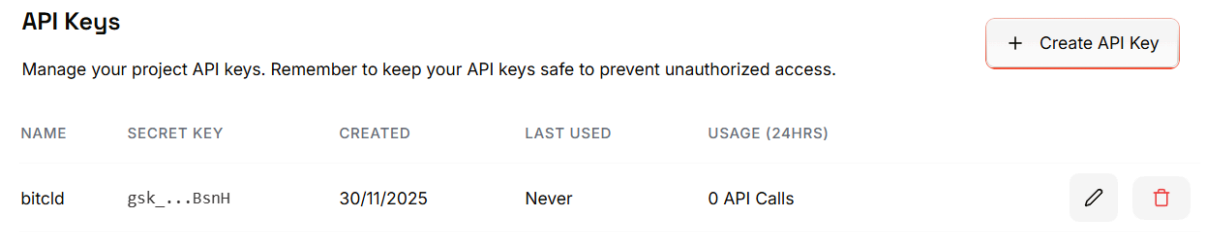
Para conseguir la API de Groq deberemos crearnos una cuenta y acceder a `https://console.groq.com/keys`



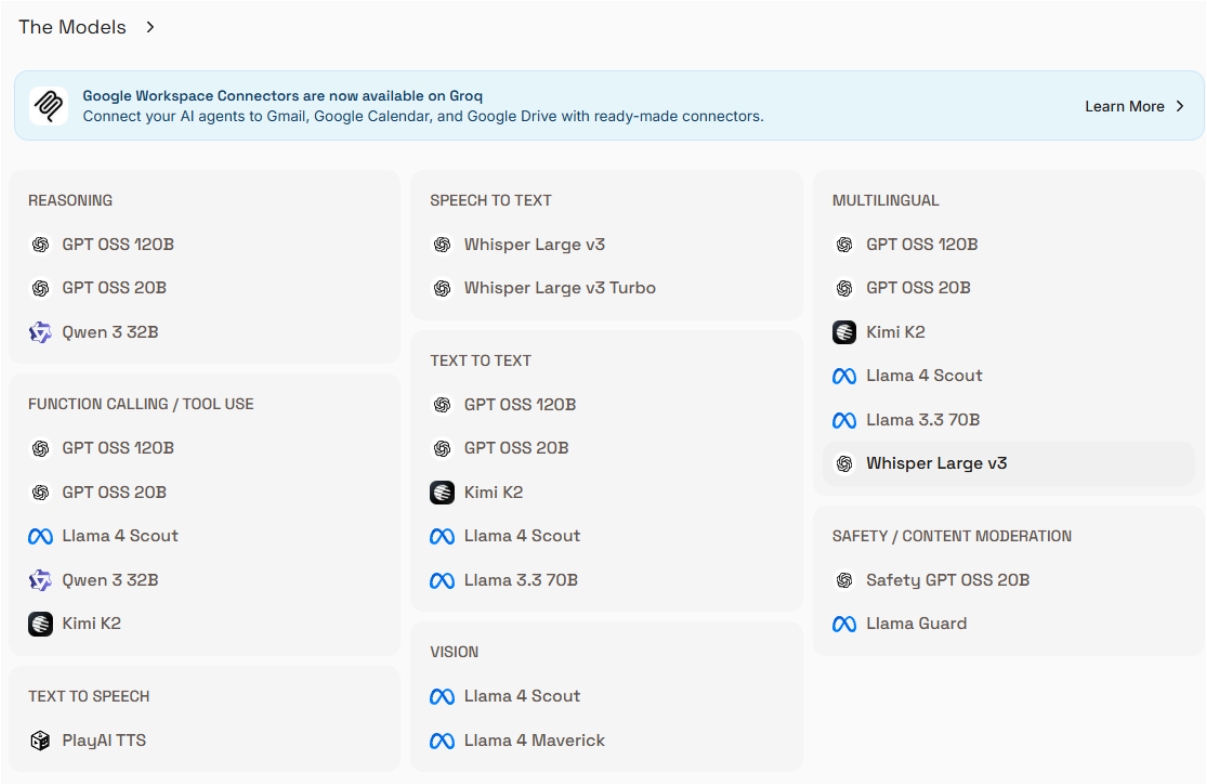
Una vez dentro haremos clic en + Create API Key y seleccionaremos un nombre para identificarla y la copiamos.

Es muy importante que la copiemos en este paso, ya que una vez creada no podremos acceder a ella de nuevo.

Y ya tendremos nuestra API



Con esta API gratuita podremos tener acceso a los siguientes modelos:



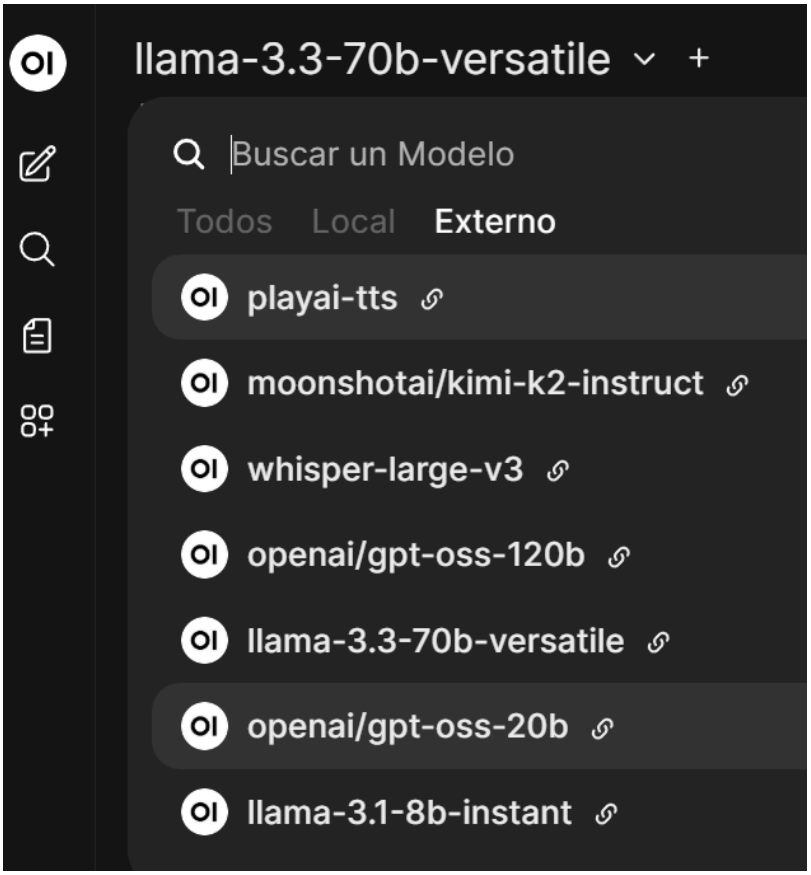
2. Añadir la API a Open WebUI

Dentro de Open WebUI volvemos a acceder a Ajustes > Ajustes de admin > Conexiones y en API OpenAI le damos a Añadir Conexión (ya que la API de Groq es compatible con OpenAI). Una vez dentro rellenamos los siguientes campos:

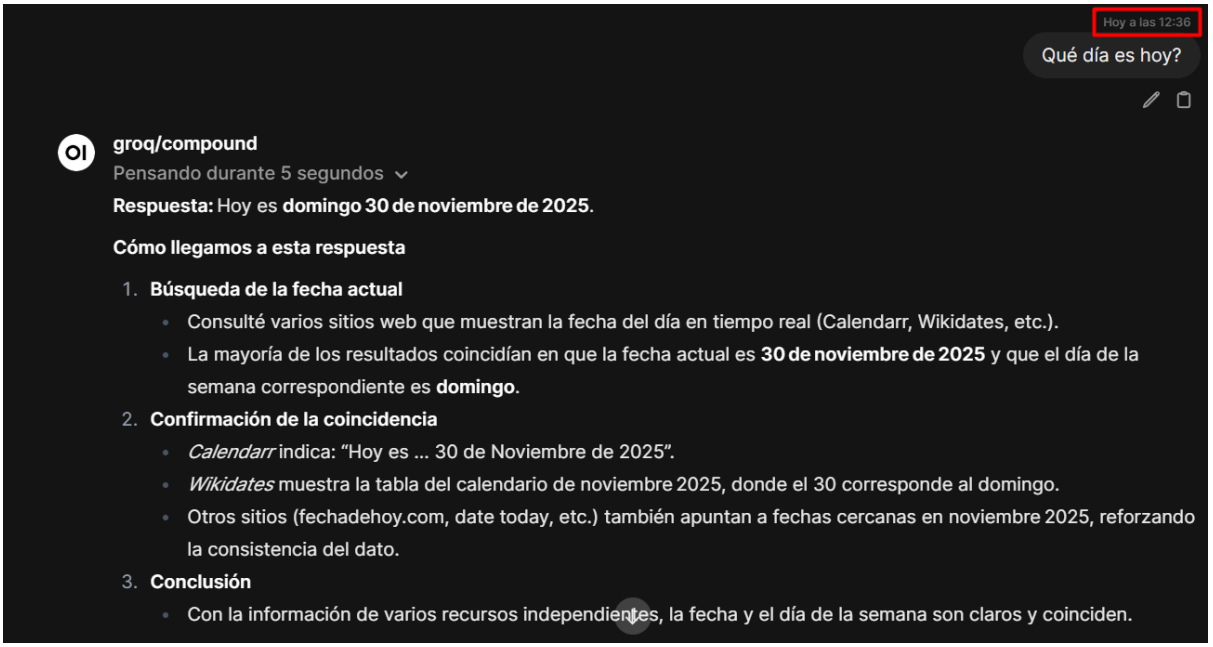
En la documentación oficial encontraremos la URL que deberemos poner `https://api.groq.com/openai/v1` y en Autorización añadiremos una API Key.

3. Uso de la API de Groq

Una vez añadida la conexión, podremos acceder a todos los modelos que nos ofrecen (en la captura no aparecen todos).

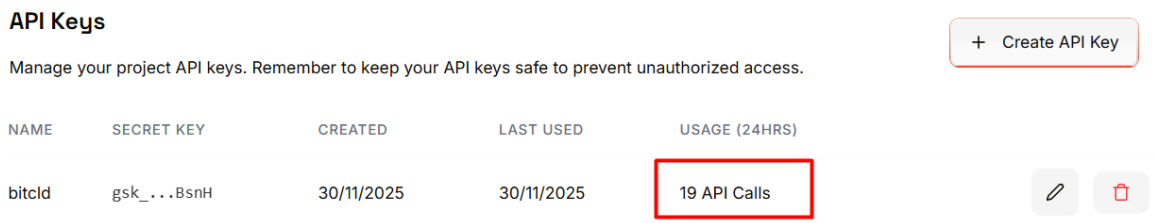


Y al igual que con los modelos locales, seleccionaremos un modelo y le haremos una consulta:



Como resultado obtenemos una respuesta elaborada, precisa y rápida sin que nuestro servidor haga el procesamiento.

Podemos ver en el panel cómo se ve reflejado el uso que hemos hecho.



Conclusión

El integrar la inteligencia artificial en bitCLD nos ha servido para adaptar nuestro proyectos a tecnologías más modernas y recientes de una manera realista y justificada. Gracias a Ollama y Open WebUI podemos ver cómo se despliegaría la IA de manera completamente local y autoalojada con modelos como Phi y Gemma pero debido a las limitaciones de hardware no pudimos tener una experiencia completamente satisfactoria.

Es por ello que podemos justificar el uso de la API de Groq ya que nos ofrece una mejor experiencia con modelos más potentes y rápidos de una manera gratuita y gracias a Open WebUI podemos usar la misma interfaz tanto para los modelos locales como para los que nos ofrece Groq. Esto nos permite que aunque no hayamos podido cumplir con los tres pilares de manera íntegra. Hemos conseguido que sea simple (al usar la misma interfaz para ambos), es seguro gracias a Caddy Server y Cloudflare + Access, parcialmente autolajoado (ya que sí alojamos Ollama, los modelos y Open WebUI en nuestro servidor) y también parcialmente de código abierto (gracias a Ollama y WebUI).

4.3.5 Vaultwarden

Introducción

Vaultwarden es una implementación ligera y autohosteada del gestor de contraseñas de Bitwarden que fue desarrollada por Rust y que ofrece las mismas funcionalidades sin depender de ningún servidor externo. Permite almacenar y compartir contraseñas de forma segura con cifrado de extremo a extremo. Y debido a que tiene compatibilidad nos proporciona una experiencia igual.

Motivos de elección

Elegimos vaultwarden debido a la necesidad de almacenar nuestras credenciales de manera cifrada con AES-256 y al ser una solución ligera, eficiente y privada al ser autogestionada. Además, algo esencial para nuestro proyecto es que podamos desplegarlo en un contenedor. Todo esto sumado a que es un software libre nos ofrece control total y está por lo que se alinea a los pilares de nuestro proyecto.

Función dentro del proyecto

Dentro de bitCLD, como es de esperar, actúa como nuestro gestor de contraseñas seguras para nuestras aplicaciones. Como nuestros otros servicios el acceso lo hacemos en local mediante `pass.lan` y a través de nuestro subdominio pass.bitcld.com. Debido a la importancia de lo que se almacena, no existe una ruta con puerto en nuestra red local. Esta aplicación ofrece seguridad al proyecto y nos muestra la importancia de usar gestores de contraseñas en nuestro día a día.

Instalación

Vaultwarden se despliega fácilmente como contenedor Docker y se gestiona desde Dockge, manteniendo la coherencia y modularidad del ecosistema bitCLD.

Desde la interfaz de Dockge, creamos una nueva pila (stack) llamada vaultwarden y añadimos el siguiente contenido en su archivo docker-compose.yml:

```
services:
vaultwarden:
    image: vaultwarden/server:latest
    container_name: vaultwarden
    restart: unless-stopped
    environment:
    SIGNUPS_ALLOWED: "false"
    INVITATIONS_ALLOWED: "false"
    ADMIN_TOKEN: <MiToken>
    ROCKET_PORT: "80"
    volumes:
    - ./data:/data
    networks:
    caddy_net:
        ipv4_address: 172.28.0.2
    expose:
    - "80"
networks:
caddy_net:
    external: true
```

Explicación del docker compose

Parámetro	Descripción
SIGNUPS_ALLOWED: "no"	Usar "yes" solo al principio para crear el primer usuario; después debe cambiarse a "no" por seguridad.
INVITATIONS_ALLOWED: "false"	Controla si se pueden enviar invitaciones a otros usuarios desde el panel de administración. Al estar en "false", solo los administradores pueden crear nuevas cuentas manualmente.
ADMIN_TOKEN: <MiToken>	Token secreto que permite acceder al panel de administración en /admin. Debe ser una cadena larga y única (por ejemplo, generada con openssl rand -hex 32).
ROCKET_PORT: "80"	Indica el puerto interno en el que Vaultwarden escucha dentro del contenedor. Caddy y Cloudflare Tunnel se conectarán a este puerto.
- ./data:/data	Crea un volumen persistente para los datos de Vaultwarden. Todo lo que guarda (base de datos SQLite, archivos cifrados, configuración) queda dentro de ./data en mi

	servidor, asegurando que la información no se pierda aunque el contenedor se borre o actualice.
caddy_net	El servicio se conecta a la red caddy_net, lo que permite que Caddy se comunique con este contenedor
ipv4_address: 172.28.0.2	Al contenedor se le asigna una IP fija
expose: - "80"	“Expone” el puerto 80 dentro de la red interna de Docker, para que Caddy, Cloudflare Tunnel o Uptime Kuma puedan comunicarse con Vaultwarden. No lo publica hacia el exterior, solo dentro de la red.

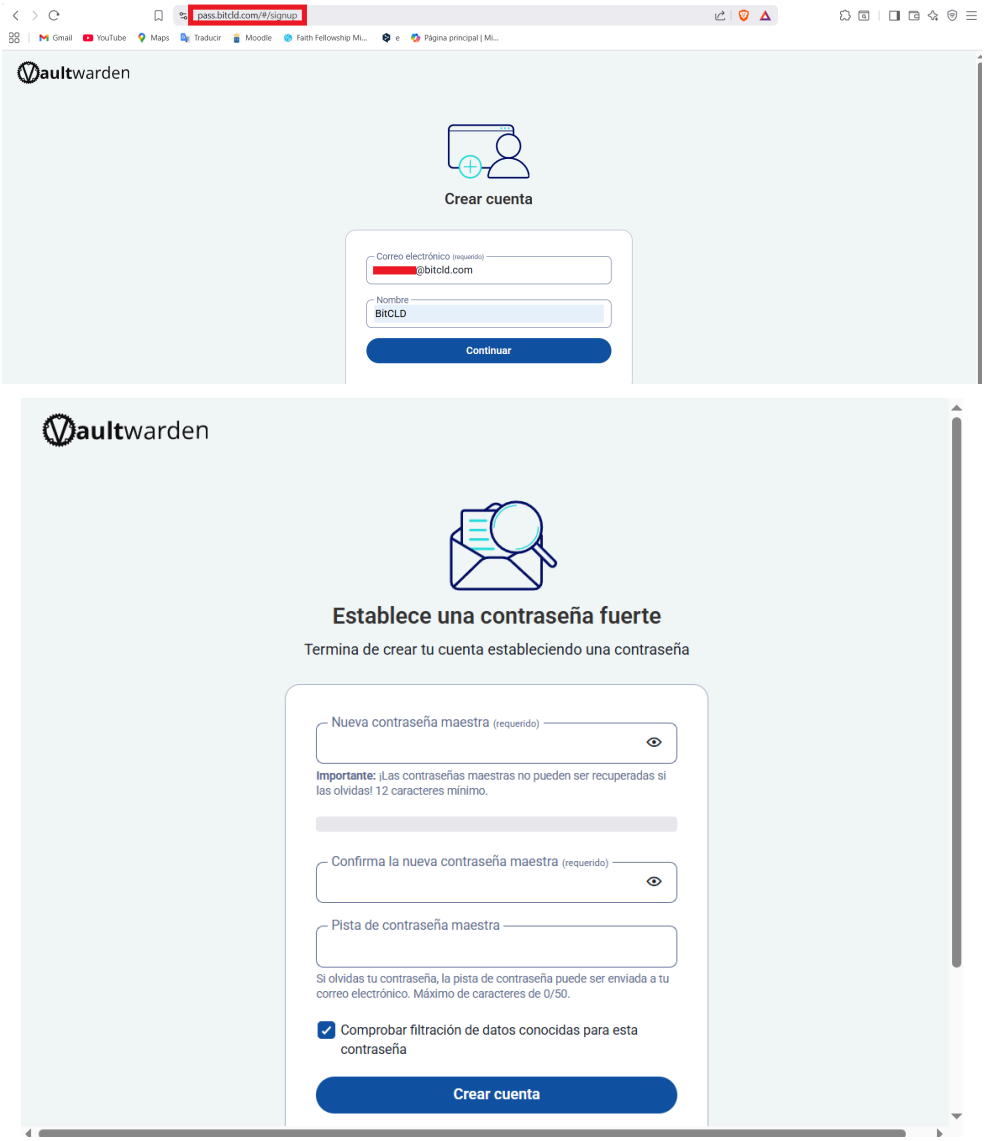
La IP fija nos garantiza que el servicio sea accesible desde Caddy o Cloudflare Tunnel siempre en la misma dirección. También facilita su monitoreo con Uptime Kuma, ya que el destino no cambia.

Configuración

Para poder crear una cuenta y usar Vaultwarden debemos asegurarnos que arrancamos el docker compose con `SIGNUPS_ALLOWED: "true"`. Para poder crear nuestra cuenta, después lo dejaremos en “`false`”.

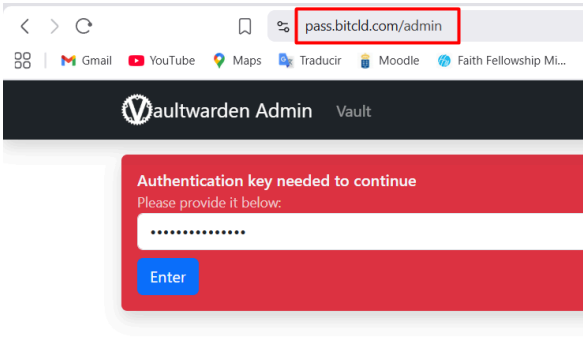
```
environment:
SIGNUPS_ALLOWED: "false"
INVITATIONS_ALLOWED: "false"
ADMIN_TOKEN: <MiToken>
ROCKET_PORT: "80"
```

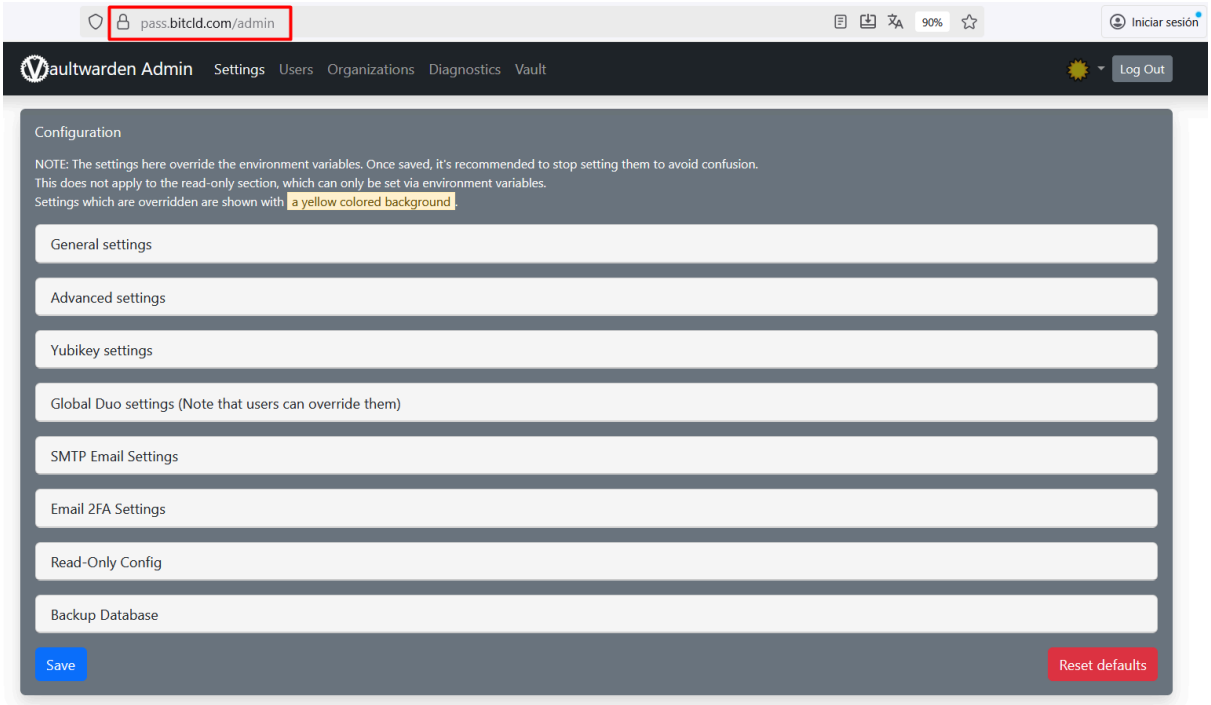
1. La primera vez que lo arrancamos, una vez configurado Caddy y Cloudflare Access, tendremos que dirigirnos a pass.bitcld.com o a pass.lan y dar a la opción de crear cuenta. Dónde pondremos un e-mail, un nombre de usuario y una contraseña maestra (la cuál debe ser lo más segura posible):



2. Una vez creada la cuenta lo siguiente sería detener el contenedor y cambiar la opción de `SIGNUPS_ALLOWED`: a `false`.

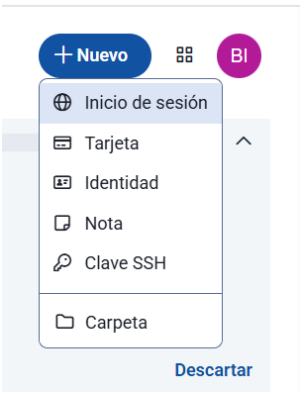
3. En caso de necesitar eliminar alguna cuenta, o ir al panel de administración deberemos abrir https://pass.bitcld.com/admin e introducir el ADMIN_TOKEN para entrar al panel de administración.



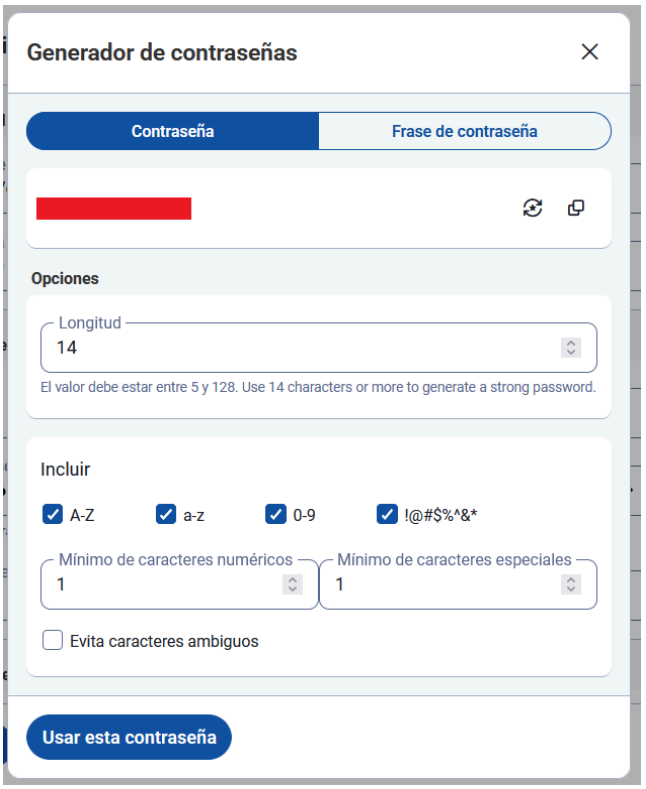


Creación de inicio de sesión

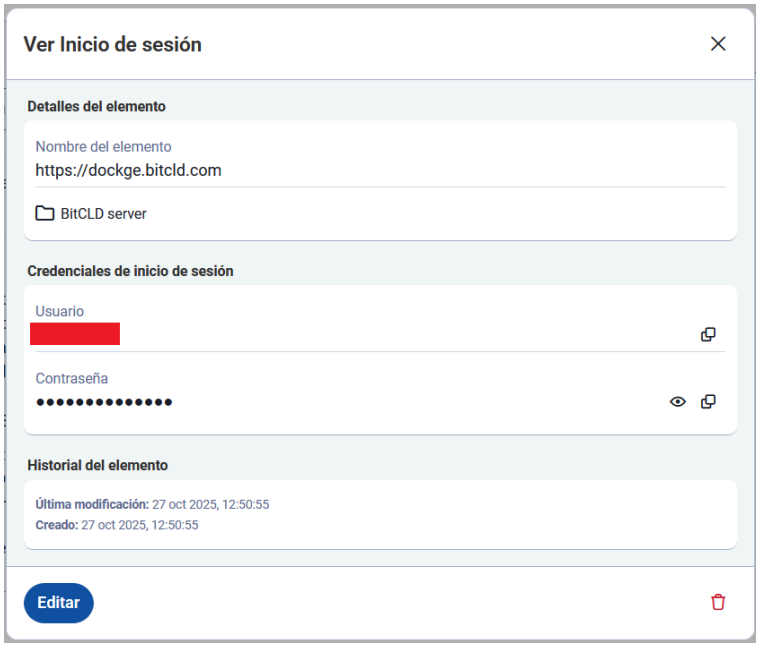
Para la creación de inicio de sesión es muy sencillo:
1. Seleccionamos Nuevo > Inicio de sesión.



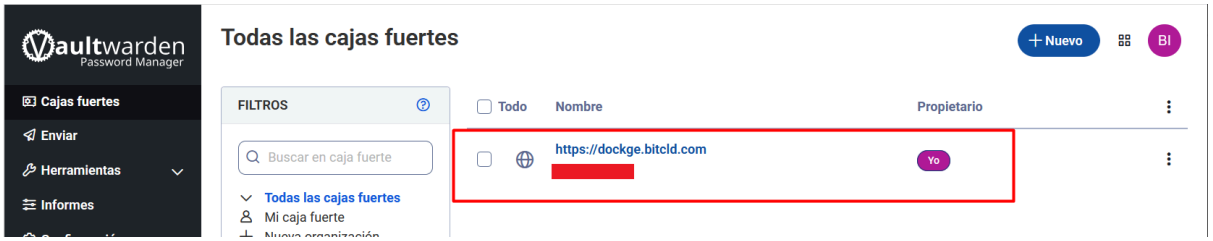
2. Elegimos los parámetros que queremos que se incluyan en nuestra nueva contraseña.



3. Introducimos y los datos que consideremos necesarios (para una mejor organización recomendamos el uso de carpetas).



4. Y ya tendríamos creado inicio de sesión



Conclusión

Vaultwarden es una aplicación fundamental en bitCLD ya que nos permite gestionar todas nuestras contraseñas de una manera muy simple y segura. El que sólo exista la opción de acceder de manera cifrada `https://` nos garantiza mucha más seguridad evitando posibles accesos no autorizados. Con todo esto obtenemos más fiabilidad, seguridad y privacidad sobre nuestras credenciales.

4.3.6 Uptime Kuma

Introducción

Uptime Kuma es una herramienta autoalojada que monitoriza servicios para comprobar su disponibilidad y su rendimiento. Dispone de una interfaz muy intuitiva y pertenece al mismo equipo que desarrolla Dockge. Y también se trata de una solución de código abierto.

Motivos de elección

Elegimos Uptime Kuma por su simplicidad, su facilidad de administración y gestión y sobretodo porque es una solución de código abierto y se puede desplegar mediante Docker, lo cuál lo hace ideal para nosotros. Además, de ser muy ligero también nos permite recibir notificaciones en Telegram, correo electrónico, discord... Debido a todo esto fue la aplicación elegida para la monitorización ya que cumple los pilares de nuestro proyecto: simple, seguro, autogestionado y de código abierto.

Función dentro del proyecto

Dentro de bitCLD, Uptime Kuma se encarga de monitorizar cada 60 segundos la disponibilidad de cada uno de nuestros servicios y también nos envía alertas en nuestro canal de Telegram dedicado para el proyecto indicándonos si está desplegado o no el servicio.

Instalación

Desde la interfaz de Dockge, creamos una nueva pila (stack) llamada uptime-kuma y añadimos el siguiente contenido en su archivo docker-compose.yml:

```
services:
uptime-kuma:
    image: louislam/uptime-kuma:latest
    container_name: uptime-kuma
    restart: unless-stopped
    volumes:
    - /srv/docker/stacks/uptime-kuma:/app/data
    ports:
    - 3001:3001
    networks:
    - caddy_net
networks:
caddy_net:
    external: true
```

Explicación del docker compose:

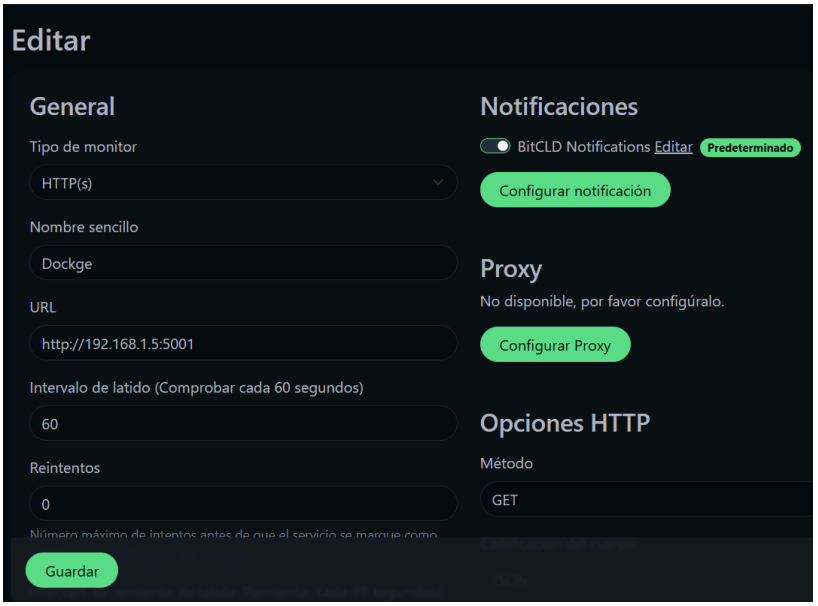
Parámetro	Descripción
image: louislam/uptime-kuma:latest	Usa la última versión oficial de la imagen de Uptime Kuma disponible desde Docker Hub.
container_name: uptime-kuma	Asigna un nombre fijo al contenedor (útil para comandos y depuración).
restart: unless-stopped	Hace que el contenedor se reinicie automáticamente si se detiene de forma inesperada, salvo que se haya detenido manualmente.
volumes: - /srv/docker/stacks/uptime-kuma:/app/data	Guarda la configuración interna y datos persistentes de Uptime Kuma (así no se pierden lass configuraciones).
ports: - 3001:3001	Puerto del host que se usará en el navegador, Cloudflare y Caddy
caddy_net	El servicio se conecta a la red caddy_net, lo que permite que Caddy se comunique con otros contenedores
external: true	Indica que la red caddy_net ya existe y no será creada por este docker-compose. Esto permite compartir la misma red entre distintos docker-compose.yml, haciendo posible que varios contenedores se comuniquen entre sí.

Configuración

En el primer acceso se solicitará crear un usuario administrador. Creamos un usuario y una contraseña y ya podremos acceder al menú de configuración. Mediante la siguiente dirección: `http://192.168.1.5:3001`

Creación de un nuevo monitor

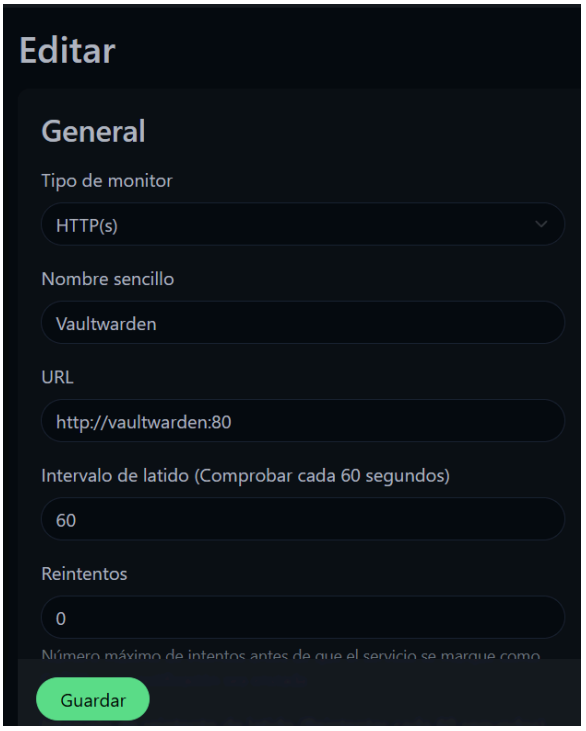
Para la creación de nuevo monitor para un servicio es muy sencillo, simplemente debemos establecer el tipo, en nuestro caso HTTP(s), un nombre, la URL `http://192.168.1.5:<puerto correspondiente>` (en nuestro caso utilizamos el puerto y no el enlace .lan para poder seguir comprobando nuestros servicios sin necesidad de que Caddy esté desplegado):



Una vez guardado ya podremos ver el estado de cada servicio en todo momento tanto si está activo o si ha caído.

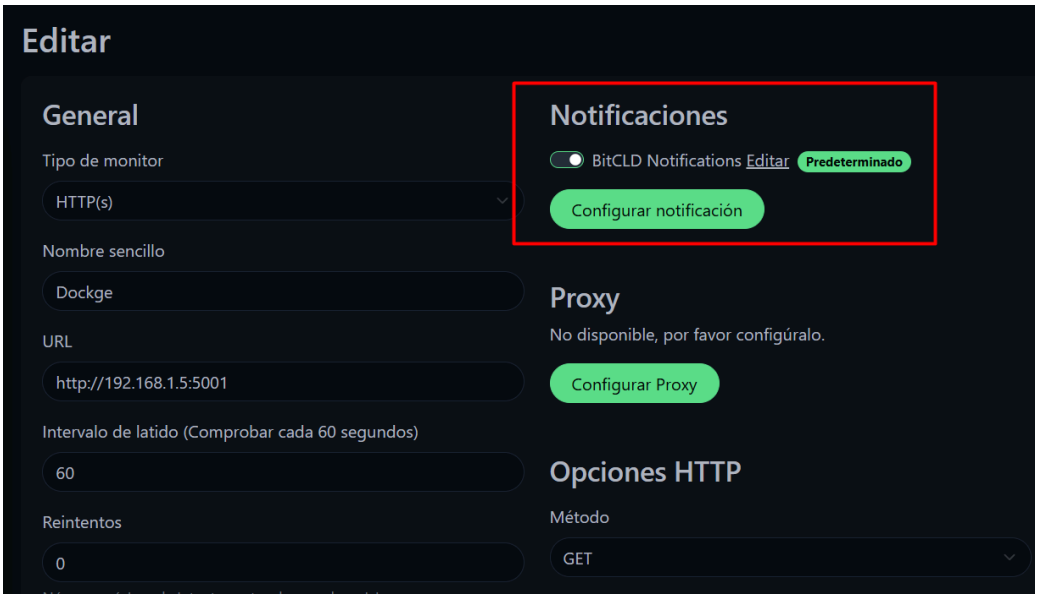
Configuración sin puerto establecido en el docker compose

En caso de no habilitar ningún puerto, como es nuestro caso con Vaultwarden, deberemos asegurarnos que Uptime Kuma esté en la misma red que Vaultwarden y Caddy y en la sección de URL ponemos el nombre del docker compose correspondiente y el que hayamos establecido en el Rocket Expose, esta variable se utiliza para configurar el puerto interno del contenedor donde se ejecuta la aplicación, en nuestro caso, VaultWarden.



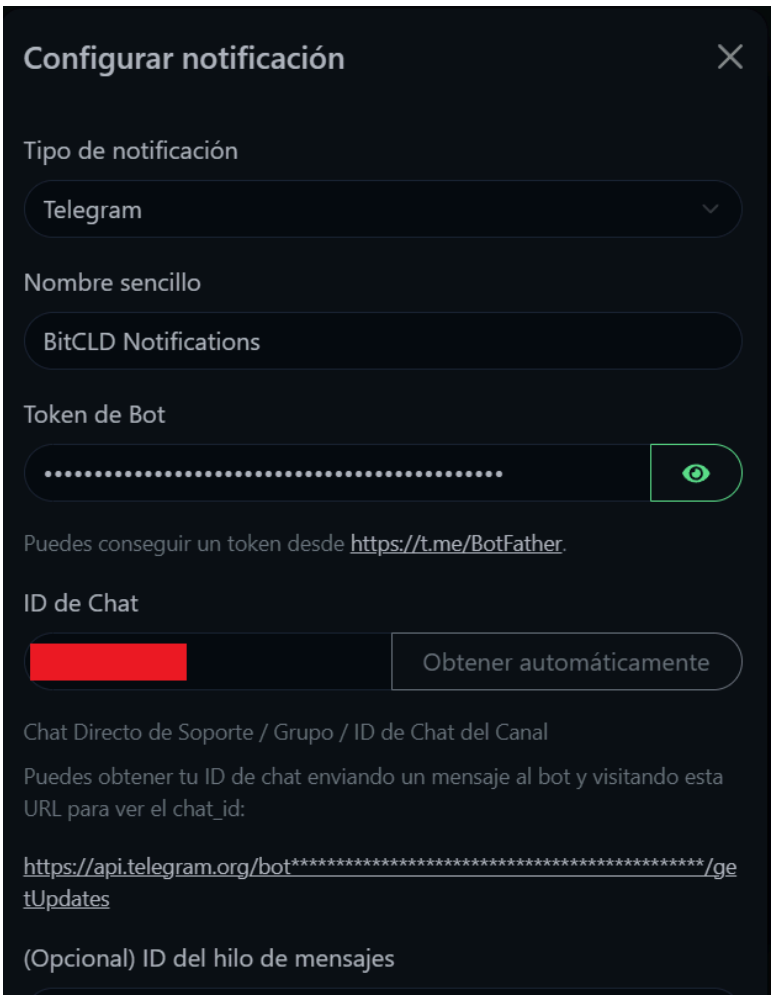
Creación de notificación

Para poder recibir las notificaciones por Telegram, Discord, Email o la plataforma de nuestra elección. Debemos irnos a los monitores ya creados y en la sección de notificaciones seleccionamos Configurar notificación.



[Para ver con detalle cómo creamos el bot, miramos nuestro chat ID del grupo y configuramos el grupo puede verlo en el anexo A.2 Bots en Telegram o en docs.bitcld.com].

Una vez tengamos el bot creado y el ID del chat de nuestro grupo ya podremos crear la notificación en Uptime Kuma.



Una vez hecho ya recibiremos las notificaciones de cada servicio en nuestro canal de Telegram junto con las del bot de Netdata.

Conclusión

Uptime Kuma es una buena aportación al ecosistema bitCLD ya que nos permite monitorizar nuestros servicios de una manera clara, ligera y mediante docker compose y gestionado por Dockge. El hecho de que podamos recibir notificaciones en Telegram nos ofrece tener una supervisión en tiempo real.

4.3.7 Netdata

Introducción

Netdata es una plataforma de monitorización a tiempo real y de salud de nuestro servidor, contenedores y aplicaciones. Nos permite poder ver métricas bastante detalladas de la CPU, memoria, red, discos, contenedores y aplicaciones gracias a su interfaz interactiva y que nos lo muestra en tiempo real. Además es una solución ligera.

Motivos de elección

Optamos por Netdata debido a que necesitábamos una solución que nos permita poder monitorizar nuestro monitor en todo momento y que nos envíe notificaciones si existe alguna incidencia. Netdata también fue escogida ya que, a diferencia de otras es ligera y fácil de configurar. Además se aloja directamente en nuestro servidor por lo que cumple con los principales de bitCLD: simple, seguro, autohospedado.

Función dentro del proyecto

Dentro de nuestro proyecto, principalmente se encarga de monitorizar nuestro servidor y de enviar notificaciones de alerta críticas. Adicionalmente, también analiza los contenedores pero esa no es su misión principal ya que de eso se encarga Uptime Kuma. Además las notificaciones serán mandadas al grupo de notificaciones donde también se encuentra nuestro bot de Uptime Kuma para centralizar las notificaciones.

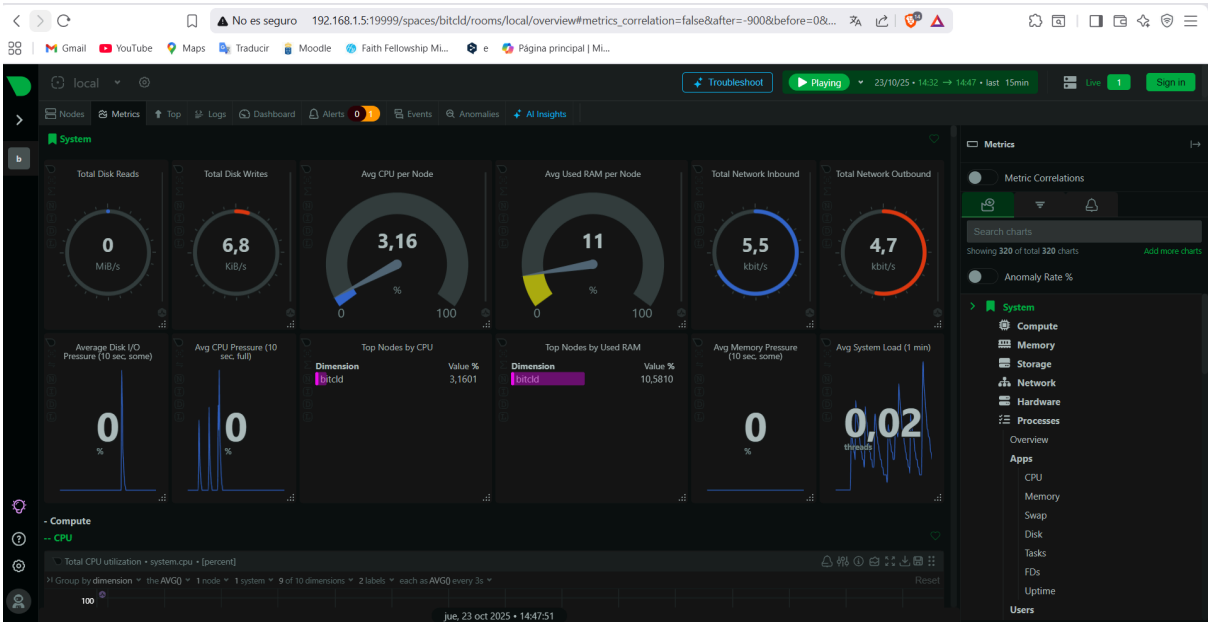
Instalación

Debido a las características de Netdata la ejecutamos directamente en el servidor para que pueda acceder a todas las métricas de nuestro sistema. La instalación es bastante sencilla.

Con estos comandos instalamos la versión más estable de Netdata.

```
sudo apt install curl -y
bash <(curl -Ss https://my-netdata.io/kickstart.sh)
```

El panel web lo tendremos en el puerto 19999.



Configuración

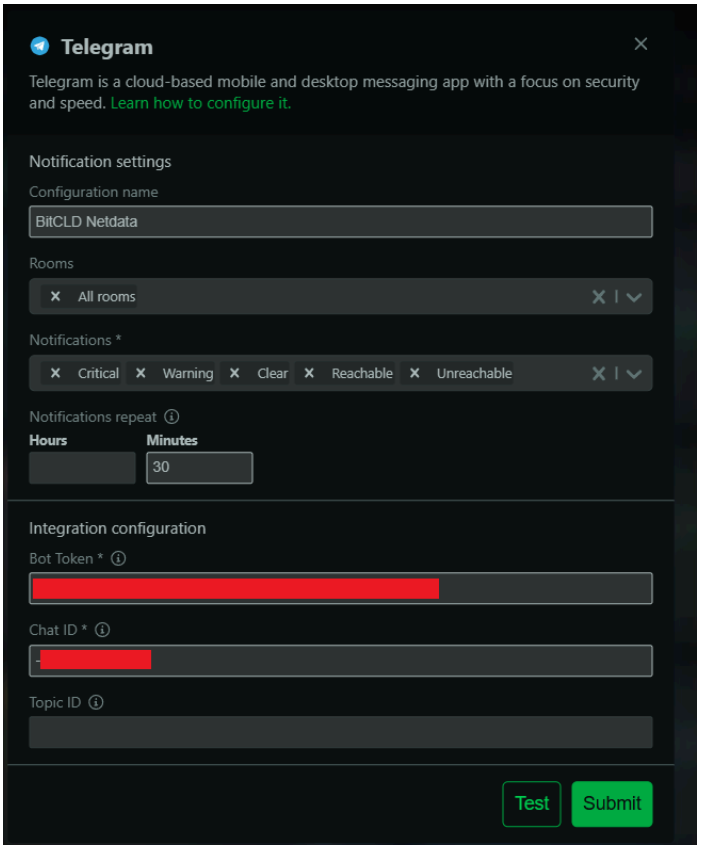
De manera automática Netdata ya se encargará de recopilar métricas de manera automática. En caso de querer personalizar la frecuencia de muestreo deberemos añadirlo en `/etc/netdata/netdata.conf`.

Como recomendación, es ideal crearnos una cuenta en Netdata ya que de esta manera podemos acceder de manera remota y segura a las estadísticas, tendremos las alertas unificadas, un historial extendido y la posibilidad de otorgar permisos a otras personas.

Crear notificaciones

Para la creación de notificaciones deberemos:

1. Irnos a space settings
2. Después en Alert y notifications seleccionamos +Add configuration
3. Seleccionar qué integración usaremos, en nuestro caso Telegram, pero existen muchas más (por email, aplicación de móvil, slack, teams...)
4. Introduciremos un nombre y el API de nuestro bot junto con el Chat ID de nuestro grupo. Además de indicar cada cuánto tiempo las queremos y en que tipo de notificaciones estamos interesados. En nuestro caso estamos interesados en todas las notificaciones posibles sobretodo en las de Critical y Warning.



[Para ver con detalle cómo creamos el bot, miramos nuestro chat ID del grupo y configuramos el grupo puede verlo en el anexo A.2 Bots en Telegram o en docs.bitcld.com].

Una vez hecho ya recibiremos las notificaciones pertinentes en nuestro canal de Telegram junto con las del bot de Uptime Kuma.

Conclusión

Netdata nos aporta una visión a tiempo real de la salud de nuestro sistema, pudiendo detectar anomalías antes de que impida el correcto funcionamiento de nuestro servidor.
Su bajo consumo en recursos, su facilidad de uso y la posibilidad de recibir notificaciones por Telegram lo hacen ideal para nuestro proyecto.

4.3.8 Zoho Mail

Introducción

Zoho Mail es una plataforma de correo electrónico profesional que forma parte del ecosistema de aplicaciones en la nube de Zoho. A diferencia de otros servicios de correo como Google, Outlook… Zoho nos ofrece un correo electrónico sin anuncios, con una interfaz moderna y una buena integración con dominios personalizados.

Motivos de elección

A pesar de no ser un servicio autoalojado, y por tanto incumplir uno de los pilares fundamentales de nuestro proyecto. Escogimos Zoho debido a que no disponíamos de mucho más tiempo para preparar un servidor de correo autoalojado pero aún así, al disponer de un dominio personalizado, queríamos aprovechar la oportunidad y crear un correo profesional ya que nunca habíamos tenido la oportunidad de hacerlo.

Además, de entre las opciones que existen escogimos Zoho mail debido principalmente a la oportunidad de crear hasta 5 cuentas con un dominio personalizado sin coste alguno y a su interfaz limpia, moderna y sin anuncios. Algo que de lo que también salimos beneficiados es que de paso aprovechamos la alta disponibilidad, seguridad y reputación que nos ofrece y que no tendríamos si lo autoalojaramos en un minipc.

Función dentro del proyecto

Dentro de bitCLD, Zoho cumple dos misiones fundamentales. La primera y más evidente es la de correo de contacto principal del proyecto mediante contacto@bitcld.com para cualquier comunicación formal si fuera necesaria y sobretodo para dar una imagen mucho más profesional al proyecto. Y también para probar como funciona el funcionamiento de un correo profesional aprendiendo cómo funciona.

Configuración

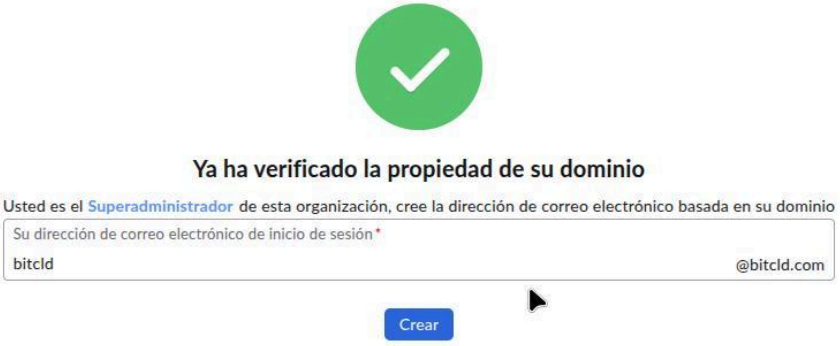
Debido a que no es un servicio autoalojado, no habrá instalación en nuestro servidor y sólo la configuración del dominio bitcld.com con Zoho Mail:

1. Nos debemos registrar en Zoho Mail de la siguiente manera:

1. Accedemos a https://www.zoho.com/es-xl/mail/
2. Seleccionamos la opción de Business Email
3. Indicamos que queremos el plan "Mail Lite – Free for custom domains" (gratuito hasta 5 usuarios)
4. Creamos una cuenta de administrador

2. Verificamos el dominio

Zoho necesita que demostremos que somos propietarios del dominio. Esto lo hacemos añadiendo un registro TXT en los registros de DNS de nuestro proveedor, en nuestro caso Cloudflare. Con el código que nos indique Zoho y una vez añadido, después de replicar que tarda un poco, ya podremos verificarlo desde el panel de Zoho.



3. Configuración de registros MX

Para recibir correos correctamente, es necesario añadir los registros MX que Zoho indica:

Tipo	Nombre	Prioridad
MX	@	10
MX	@	20
MX	@	50

4. Autenticación (SPF, DKIM)

Para mejorar la entrega y seguridad del correo, se configuraron los siguientes registros de SPF y DKIM:

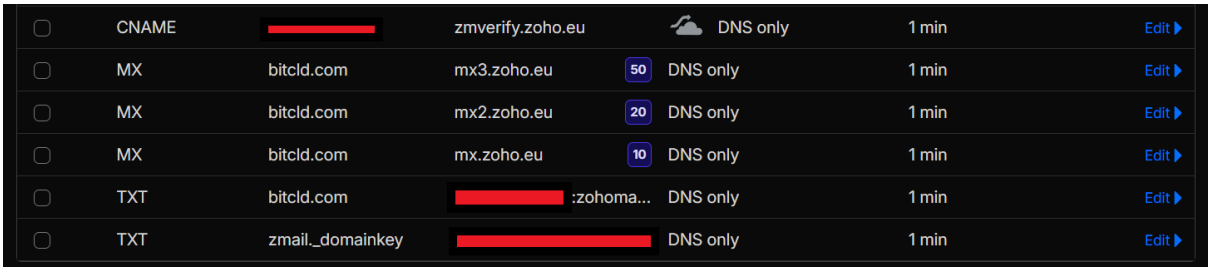
SPF

Añadiendo: v=spf1 include:zoho.eu ~all

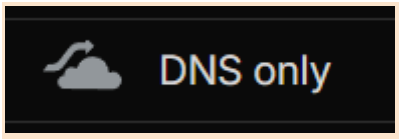
DKIM

Zoho genera un registro tipo TXT con nombre zoho._domainkey y un valor único, que debe añadirse al DNS.

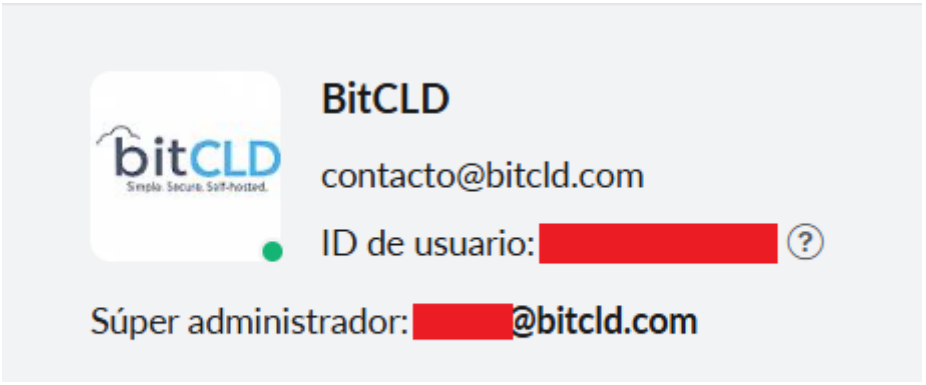
Una vez hayamos finalizado, los registros DNS de nuestro servidor deben aparecer así:



En caso de usar Cloudflare como gestor de dominio. Para que funcione correctamente debemos desactivar el proxy nube naranja en estos registros ya que los servicios de correo requieren resolución directa hacia los servidores de Zoho y no a través de Cloudflare.



Una vez completada la configuración de los registros DNS y la verificación del dominio, creamos una cuenta de contacto y ya tendremos una cuenta de correo funcional.



Conclusión

Aunque no es un servicio autoalojado, Zoho Mail nos ofrece estabilidad, alta disponibilidad y una mejor imagen de marca para bitCLD y todo esto sin ningún coste adicional. Además conseguimos también una suite completa y funcional para el trabajo colaborativo, complementaria a Nextcloud. Sabemos que no es lo ideal pero como un extra para poder trabajar con las tecnologías de correo nos pareció una buena adquisición al ecosistema.

4.4 Tabla explicativa

En este apartado incluiremos una tabla de como han quedado todos nuestros servicios una vez implementados incluyendo puertos internos, dominio local, subdominio y redes docker:

Servicio	Puerto	Local	Subdominio	Red docker
Caddy	-	-	-	Caddy_net
Dockge	5001	dockge.lan	dockge.bitcld.com	-
Adguard Home	53, 3000	dns.lan	-	Caddy_net
Nextcloud	8080	cld.lan	cld.bitcld.com	-
IA	3003	chat.lan	chat.bitcld.com	-
Vaultwarden	3002	pass.lan	pass.bitcld.com	Caddy_net
UptimeKuma	3001	monitor.lan	monitor.bitcld.com	Caddy_net
Netdata	19999	stats.lan	stats.bitcld.com	-

El único servicio en el que actualmente tenemos una IP estática es Vaultwarden, la cuál es `172.28.0.2` . Sin embargo, es algo interesante de añadir en todos los servicios para que todo el tráfico pase cifrado mediante `https://`.

5. Problemas detectados y soluciones aplicadas

Debido a la naturaleza de nuestro proyecto han existido diversos problemas de los que vamos comentar qué fueron y cómo lo solucionamos.

5.1 Cloudflare

Con la plataforma Cloudflare tuvimos un problema de comprensión inicial, debido a que:

✗ Pensábamos que deberíamos tener un túnel para cada servicio, lo cuál es completamente innecesario y desorganizado.

✓ Lo solucionamos viendo vídeos del proceso de generación de túneles y viendo la documentación oficial.

✗ No sabíamos cómo modificar la pantalla de login de Access para darle un toque más profesional ni tampoco añadir el logo una vez que encontramos la opción.

✓ Para solucionarlo estuvimos investigando dentro de las propias aplicaciones hasta que lo encontramos, pero nos costó bastante ya que no encontramos vídeos y no lo llegamos a ver en la documentación. Para el logo fue más simple cuando empezamos a utilizar GitHub.

5.2 Dockge

✗ El principal problema que tuvimos fue que cuando creamos los primeros docker compose de nuestros servicios Dockge no era capaz de gestionarlos por lo que se volvía inútil.

✓ La solución la encontramos en un vídeo en el que descubrimos que no habíamos ajustado bien el docker.sock ni el environment. Una vez ajustados ya podíamos gestionarlos desde Dockge sin problemas.

✗ Otro problema fue que no podíamos iniciar un servicio ya que una dirección IP ya está siendo utilizada, en el caso del servicio que usa una IP estática, Vaultwarden.

✓ La solución que encontramos es iniciarlo primero o en su defecto más adelante poner a cada servicio una IP estática y así no se nos pisan.

✗ El último problema que encontramos fue que estábamos repitiendo el mismo puerto en dos servicios distintos y por tanto nos saltaba error en la terminal.

✓ Lo solucionamos fijándonos bien en los puertos y teniendo un documento donde empezamos a apuntar los puertos.

5.3 Adguard Home / Caddy

✗El problema que tuvimos con Adguard Home fue que no filtraba anuncios ni rastreadores en las páginas en la que entrábamos

✓La solución fue añadir las listas personalizadas de filtrado, a partir de ese momento ya bloqueaba anuncios y rastreadores en páginas como m.youtube.com desde el móvil.

✗El otro problema que detectamos fue que cuando poníamos nuestros dominios locales, después de añadir los rewrittes personalizados, nos seguían sin aparecer por lo que no podíamos acceder desde nuestro dispositivo móvil.

✓La solución que encontramos fue añadir a Adguard Home en la red de Caddy en el docker compose, una vez agregado ya se podían comunicar y por tanto ya los dominios locales funcionaban sin problemas.

5.4 Nextcloud

✗El único inconveniente que tuvimos con Nextcloud fue que desconocíamos que debíamos modificar el archivo `config.php` para poder acceder a nuestros enlaces de cld.lan y cld.bitcld.com.

✓Una vez añadidos los dominios a `trusted_domains` ya no tuvimos ningún problema para poder acceder a ellos.

5.5 Inteligencia Artificial

✗El principal problema que tuvimos con la inteligencia artificial en nuestro proyecto fueron nuestras limitaciones de hardware para poder correr de forma óptima y rápida modelos locales, encontrando uno que tarda pero que al menos nos responde.

✓La solución más sencilla y completa que encontramos para no descartar la IA en bitCLD fue el uso de una API gratuita de Groq. De esta manera tenemos acceso a inteligencia artificial rápida y con modelos más completos sin gastar dinero.

5.6 Vaultwarden

✗Con este servicio el principal problemas que teníamos fue que al tener Caddy y Vaultwarden en dos dockers diferentes no se podían comunicar y por tanto usarlo ya que no disponíamos de un puerto como tal.

✓ La solución fue crear nuestra red `Caddy_net` gracias a la documentación oficial de la web.

✗ Otro problema con el que nos encontramos fue el monitorizar y saber cómo implementarlo con Cloudflare Tunnel y con Caddy en su CaddyFile.

✓ Para solucionarlo, le pusimos una IP estática al contendor para poder tenerlo siempre localizado y poner el puerto que está “expuesto”.

5.7 Uptime Kuma

✗ A pesar de ser bastante simple, no comprendimos bien como usar las notificaciones de Uptime Kuma ya que nunca habíamos trabajado con Telegram ni bots.

✓ Gracias a un vídeo en youtube que nos explicó paso a paso como hacerlo pudimos tenerlas configuradas sin problemas.

5.8 Netdata

✗ Con Netdata, al igual que con Uptime Kuma, no fuimos capaces de configurar las notificaciones al comienzo.

✓ Lo terminamos arreglando cuando creamos la cuenta de Netdata, ya que al principio no la habíamos creado, y buscando en las opciones, una vez creada, la encontramos y pudimos configurarlas con un bot de Telegram.

5.9 GitHub, GitHub Pages y Cloudflare Pages

✗ Al principio nunca habíamos desplegado una página web en internet y mucho menos usado GitHub ni los distintos Pages por lo que todo fue un lío y no nos aclarábamos bien.

✓ Gracias a vídeos explicativos y a mirar en las documentaciones respectivas de cada uno pudimos desplegar tanto nuestra página web, que sirve como presentación del proyecto, y nuestra documentación para darle un toque más profesional.

6. Guía de uso

Una vez realizada toda la implementación, el uso en el día a día de bitCLD es bastante sencillo.

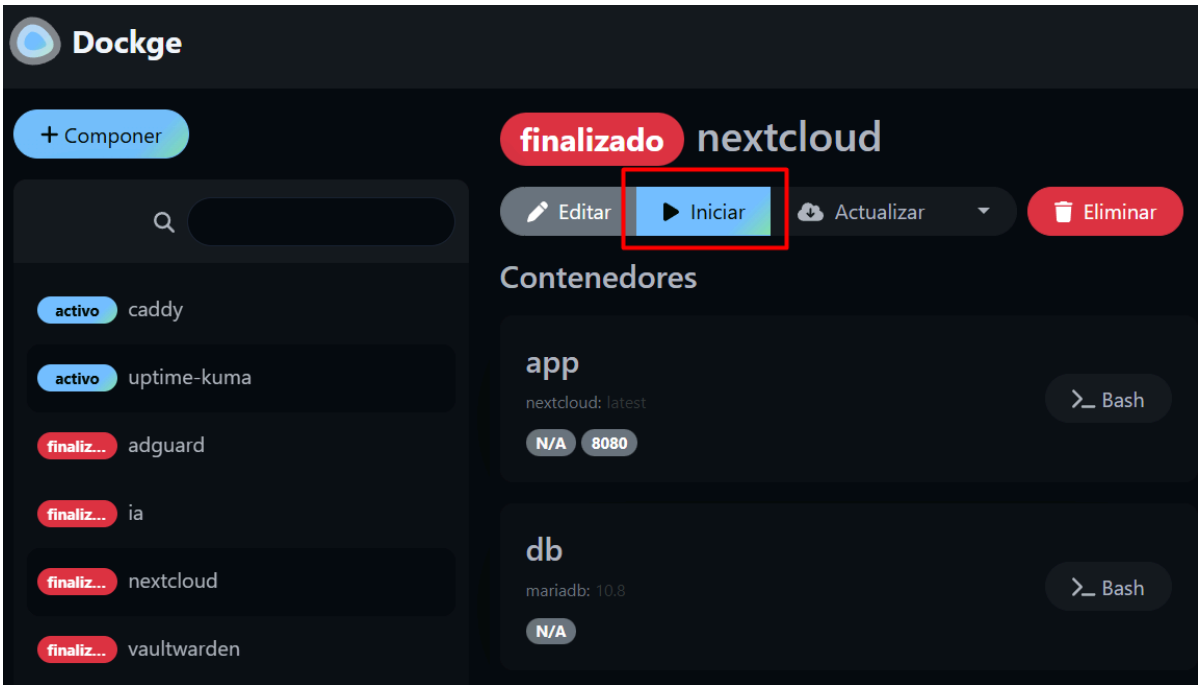
6.1 Iniciar servicio [Dockge]

Una vez encendido nuestro servidor, en el caso de que esté apagado

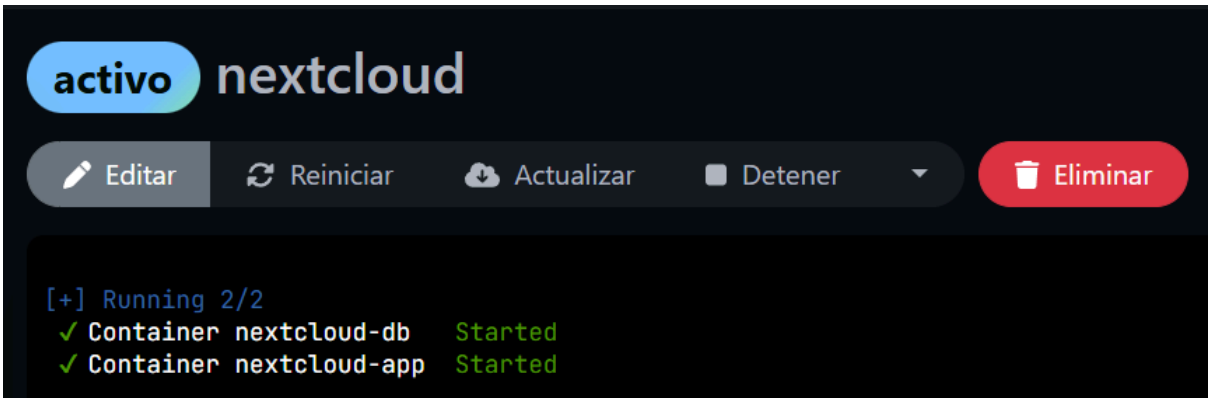
1. Accederemos a **Dockge**, ya sea en local (`dockge.lan`) o mediante dockge.bitcld.com
2. Desde su interfaz miraremos si el servicio que necesitemos está o no iniciado:

Por lo general intentaremos tener siempre iniciados Caddy y Uptime Kuma y los demás en función de su uso los dejaremos apagados para no cargar el servidor de más sin necesidad.

3. Seleccionaremos el servicio que deseemos y le damos Iniciar:



4. Una vez que esté iniciado podremos acceder al servicio:

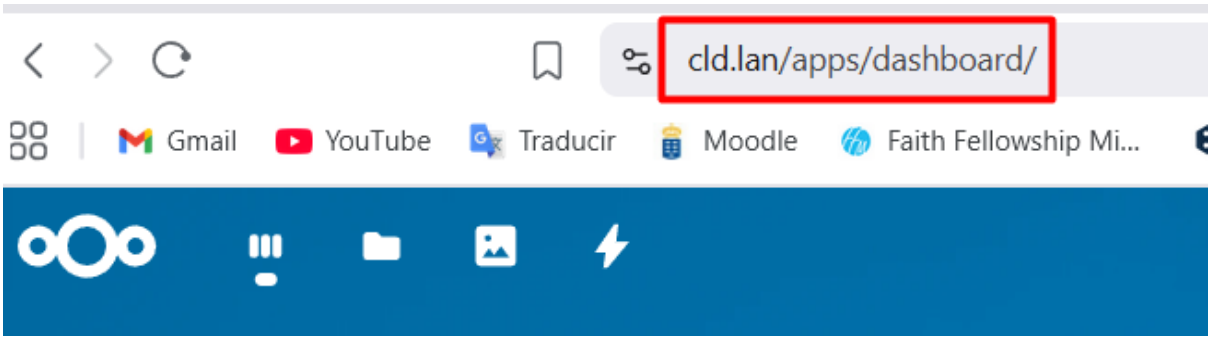


5. Para apagar el contenedor es tan sencillo como darle a la opción Detener en la pila correspondiente.

6.2 Acceder en local

Para acceder en local a nuestros servicios es muy simple.

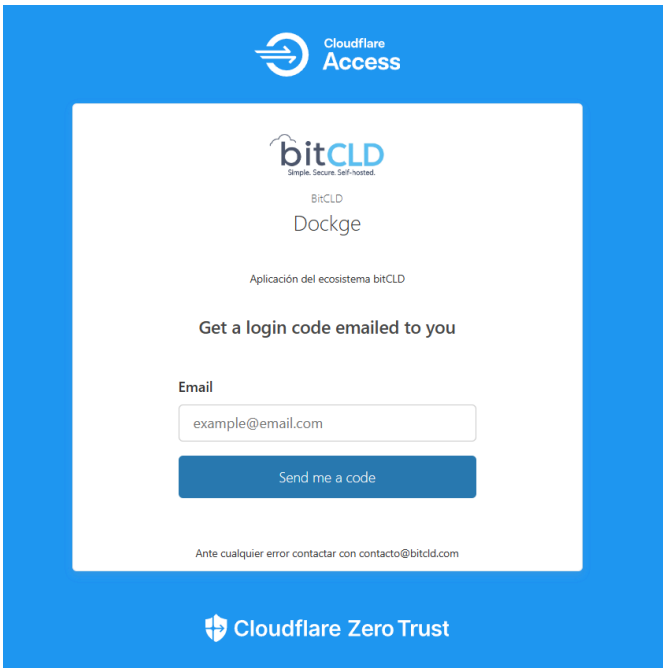
1. Deberemos asegurarnos que el servicio de Caddy esté iniciado (en caso de estar usando nuestro dispositivo móvil también deberemos asegurarnos que Adguard Home esté activado y que lo estemos usando como servidor DNS).
2. Accedemos al servicio deseado mediante su dirección `.lan`, por ejemplo, `cld.lan`
3. Y ya tendríamos acceso a nuestro servicio



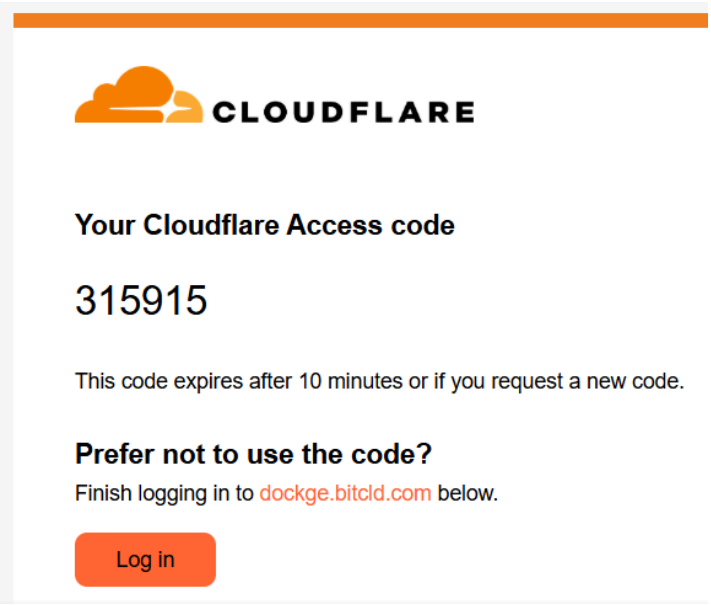
6.3 Acceder mediante subdominio

Para acceder a cualquier servicio mediante el subdominio.

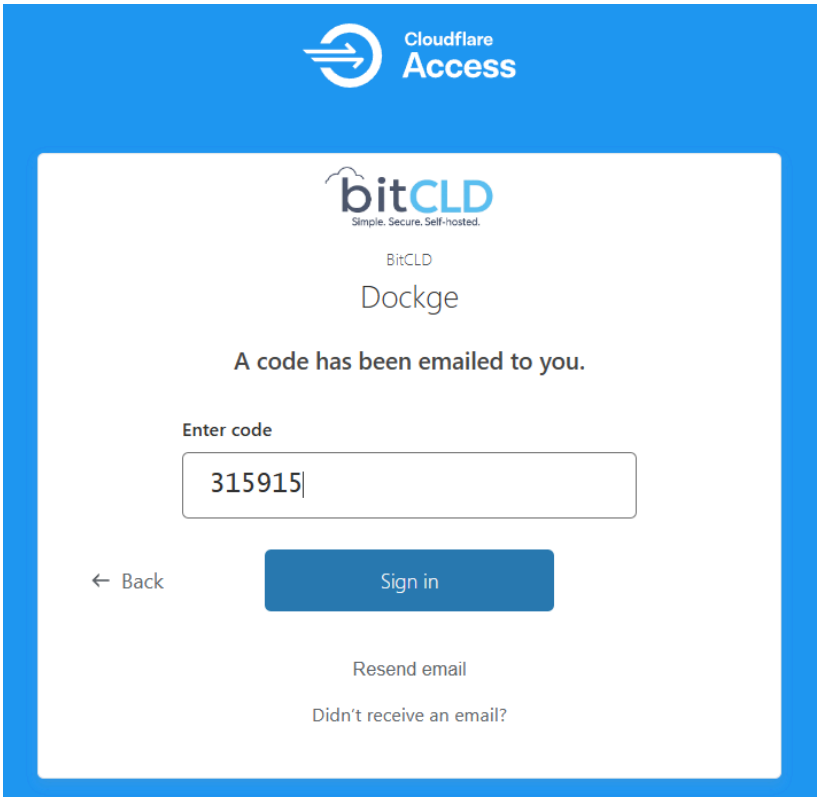
1. Accedemos al subdominio de la aplicación a la que queramos acceder, por ejemplo, dockge.bitcld.com



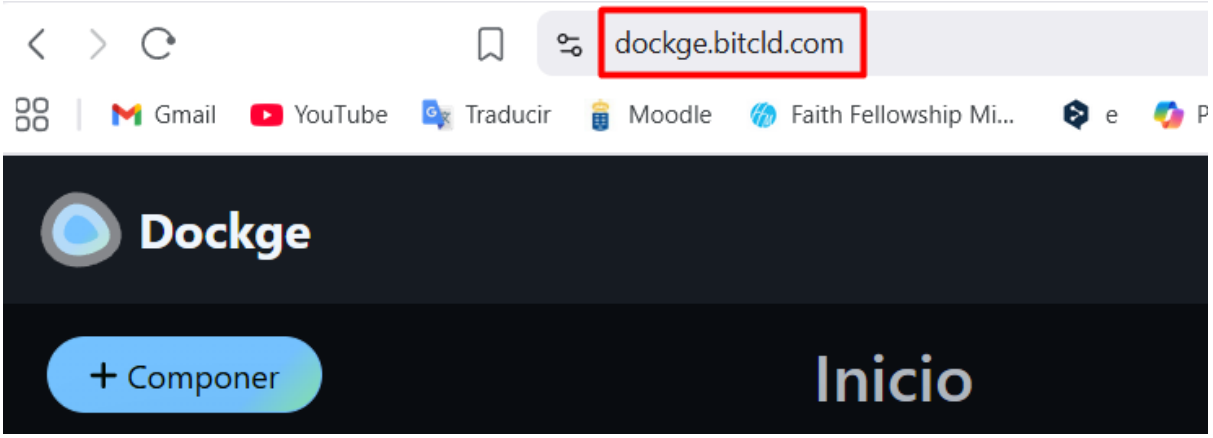
2. Introducimos uno de los emails que hemos establecido en nuestra política y recibiremos un código OTP a nuestro correo.



3. Escribimos el código recibido.



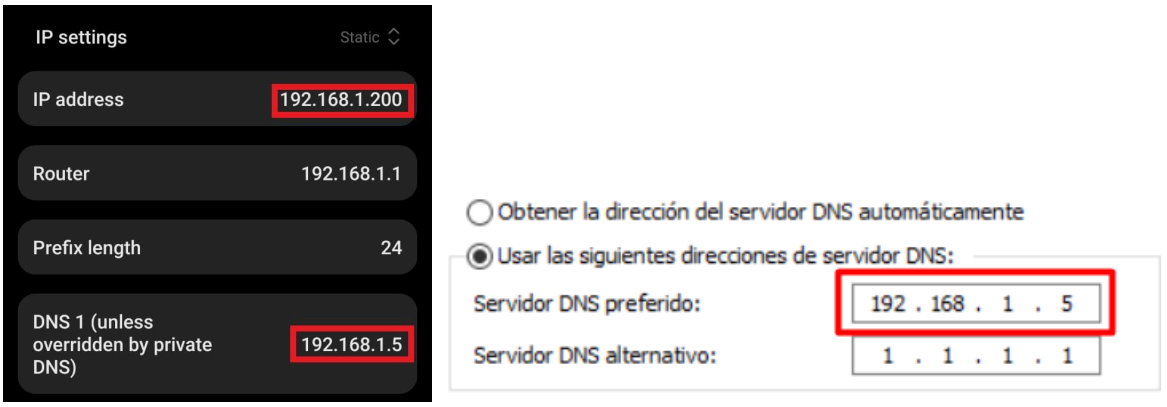
4. Y ya tendríamos acceso a nuestro servicio



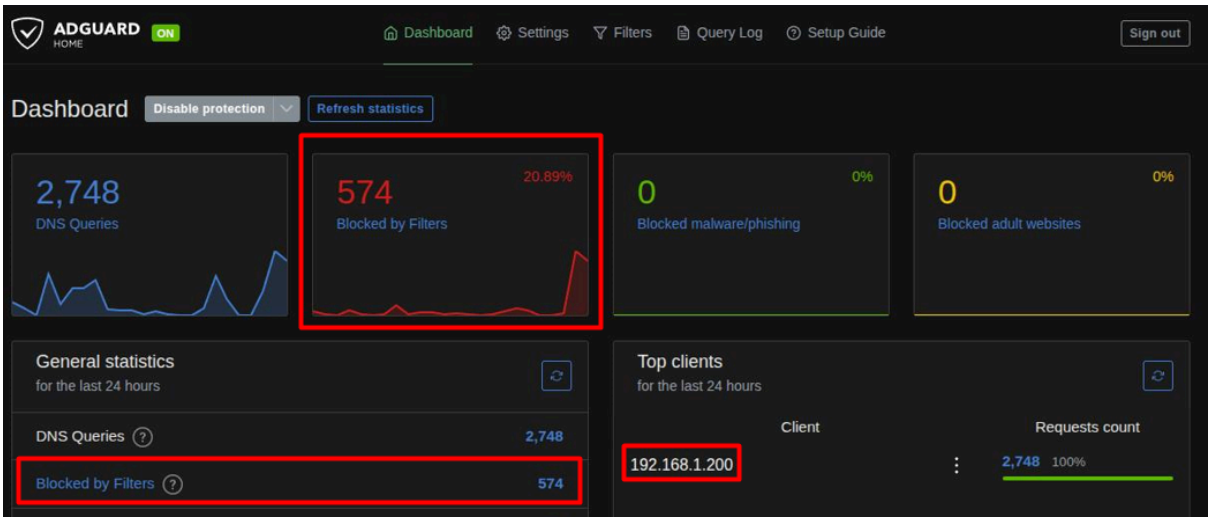
6.4 Adguard Home

Para usar Adguard Home en el día a día debemos:

1. Asegurarnos en Dockge que el stack de adguard esté iniciado.
2. Deberemos establecer a Adguard Home como nuestro servidor DNS, poniendo la la IP de nuestro servidor como Servidor DNS predefinido.



3. Una vez lo hayamos establecido como servidor DNS, Adguard Home bloqueará anuncios y rastreadores.



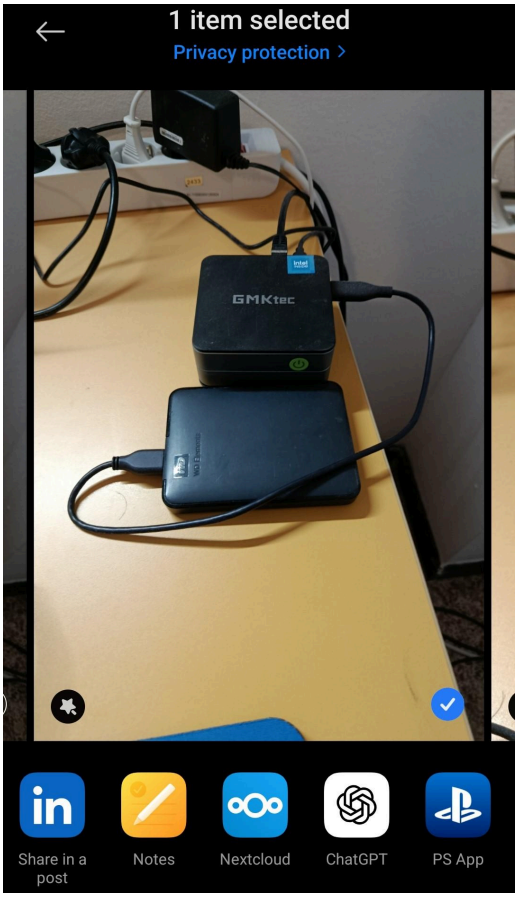
6.5 Nextcloud

Para el uso en el día a día de Nextcloud es muy sencillo ya que gracias a su interfaz tan cómoda es igual que cualquier otra herramienta de servicio en la nube como Google Drive, Outlook…

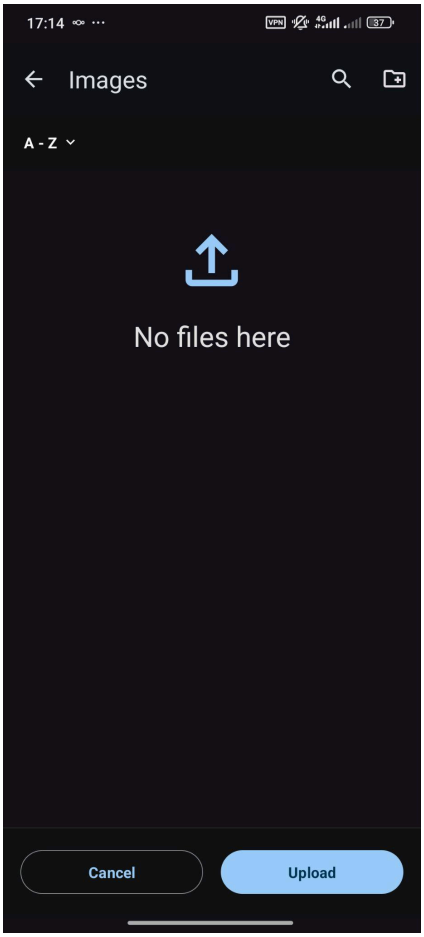
Desde la aplicación móvil

Para demostrar cómo se usa vamos a subir una imagen desde nuestro móvil, como por ejemplo la de nuestro servidor para añadirla a la documentación:

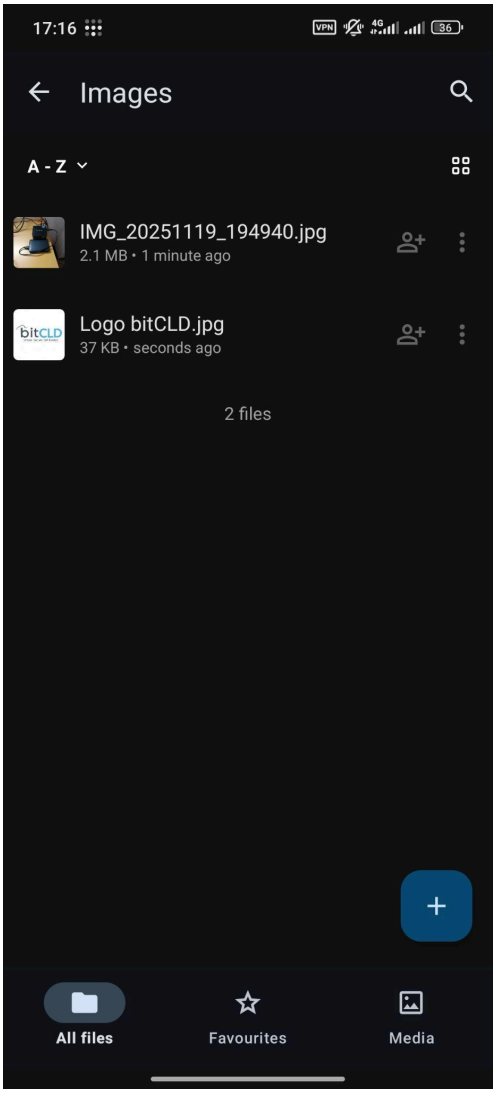
1. Asegurarnos en Dockge que el stack de nextcloud esté iniciado.
2. Conectarnos a Tailscale
3. Seleccionamos la imagen que queremos subir de la galería, le damos a compartir y seleccionamos Nextcloud:



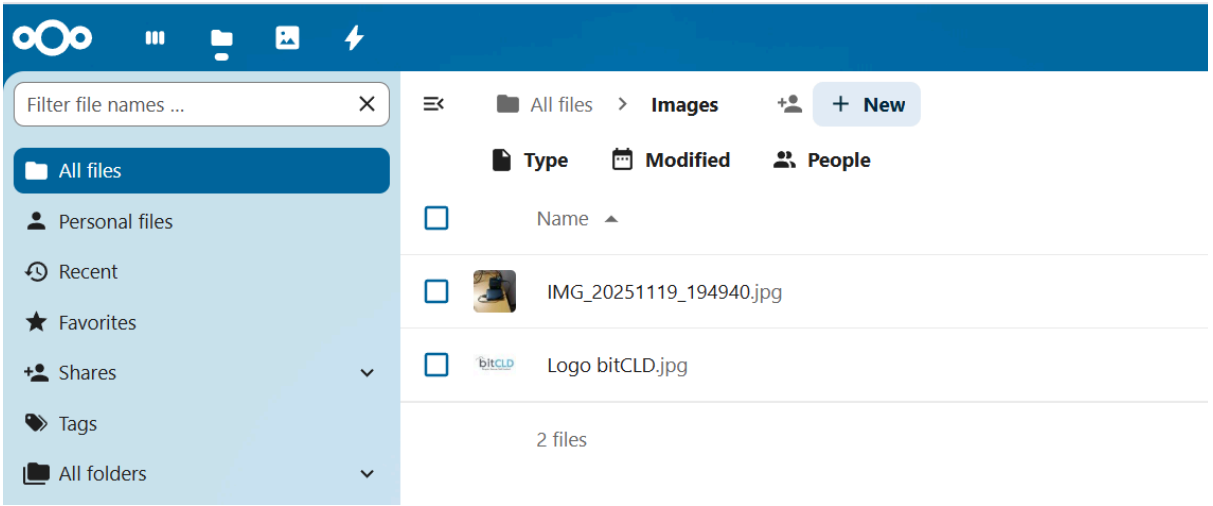
4. Elegimos donde la queremos guardar, en nuestro caso en Images:



5. Y ya la tendríamos subida a nuestro disco duro en nuestro servidor

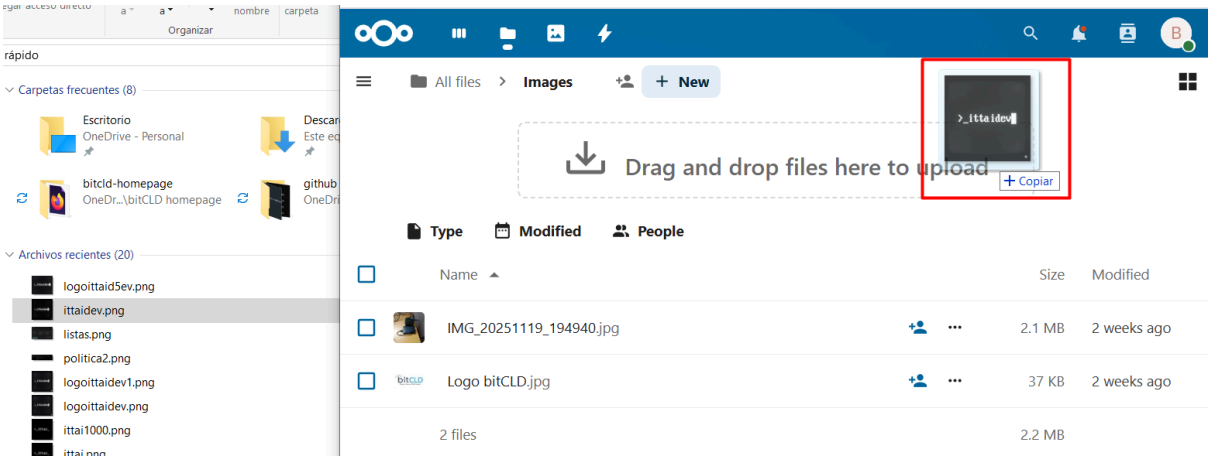


4. Y por supuesto también podemos acceder desde nuestro ordenador

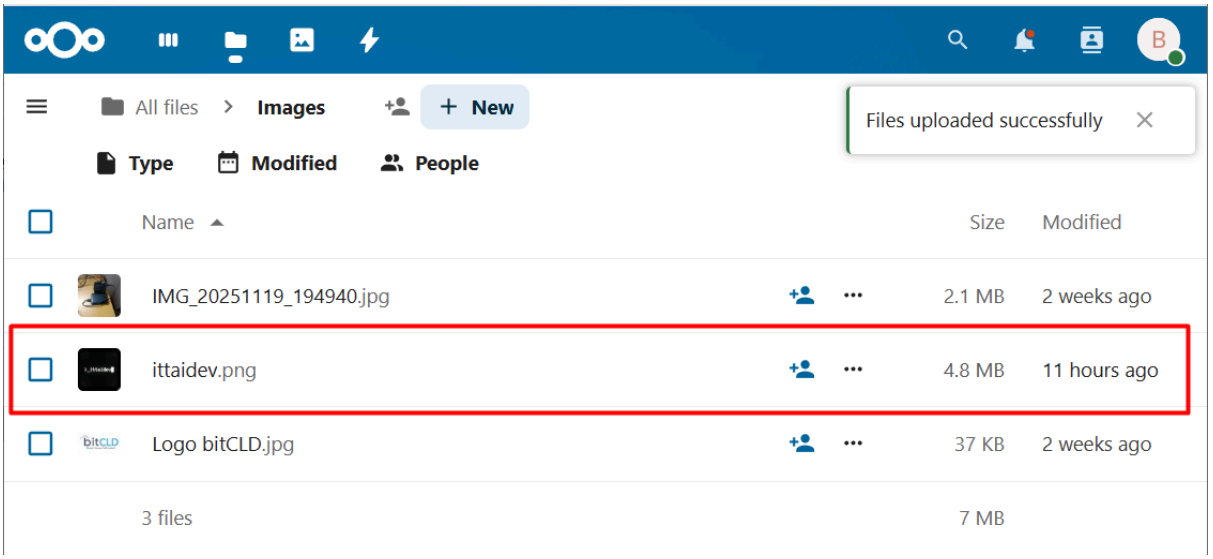


Desde la interfaz web:

1. Para subir imágenes o archivos desde la interfaz web es tan simple como acceder a cld.bitcld.com o a `cld.lan` y arrastrar y soltar lo que deseemos subir:



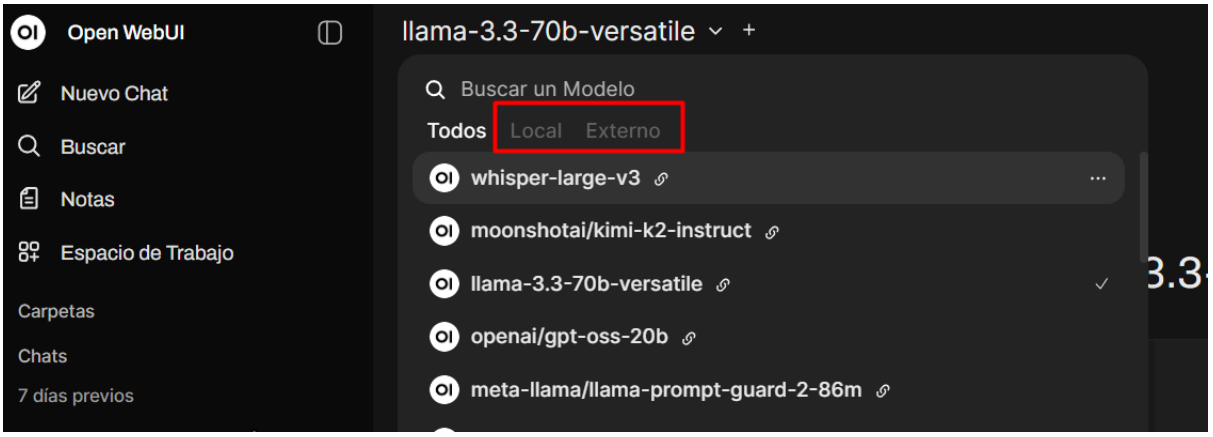
Y ya estaría subido a nuestra nube personal.



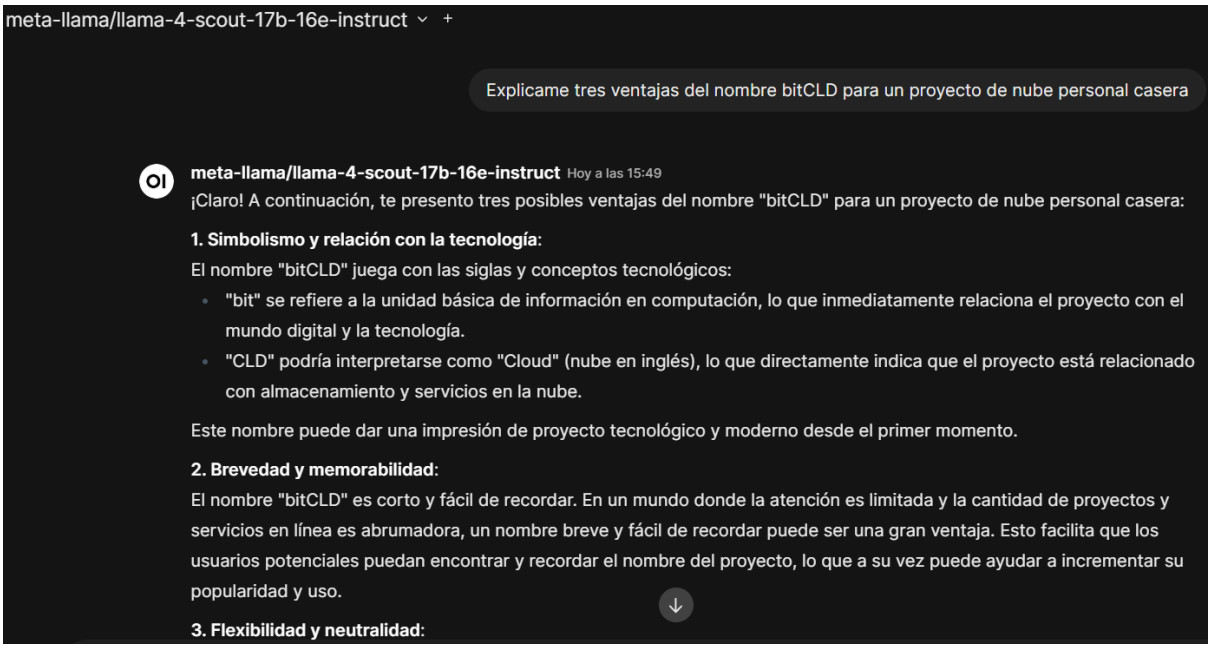
6.6 Inteligencia Artificial

Para el uso de la inteligencia artificial es tan fácil como:

1. Asegurarnos en Dockge que el stack de ia esté iniciado.
2. Acceder a chat.lan o chat.bitcld.com
3. Poner nuestras credenciales
4. Elegir un modelo Local o Externo (al estar en el mismo contenedor Open WebUI y Ollama siempre tendremos acceso a nuestros modelos locales desde la interfaz).



5. Y realizar la consulta que necesitemos.



6.7 Vaultwarden

Para acceder a su interfaz y ver nuestros inicios de sesión deberemos:

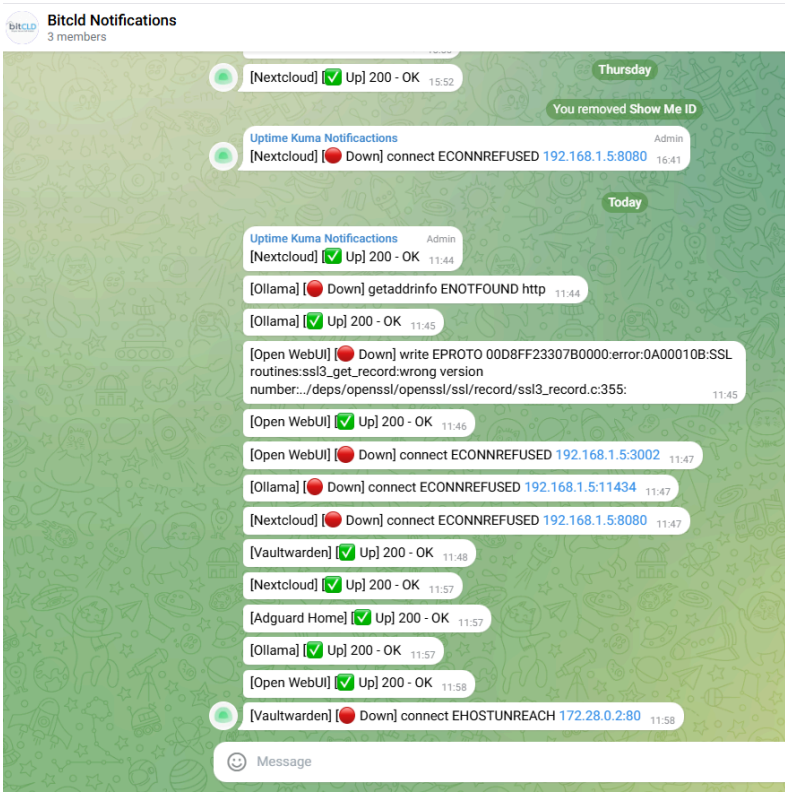
1. Asegurarnos en Dockge que el stack de vaultwarden esté iniciado.
2. Acceder a `pass.lan` en nuestra red local o a pass.bitcld.com.
3. Y ya podríamos ver nuestros inicios de sesión y copiar y pegar tanto el usuario como la contraseña:

6.8 Uptime Kuma

Para poder monitorear nuestros servicios debemos:

1. Asegurarnos en Dockge que el stack de uptime-kuma esté iniciado.
2. Acceder a `monitor.lan` en nuestra red local o a monitor.bitcld.com.
3. Desde la interfaz web donde podemos observar en todo momento la salud de nuestros servicios:

4. De manera pasiva también recibiremos las notificaciones de los servicios que estamos monitorizando indicándonos si está o no activo de manera automática en el grupo de telegram que configuramos:

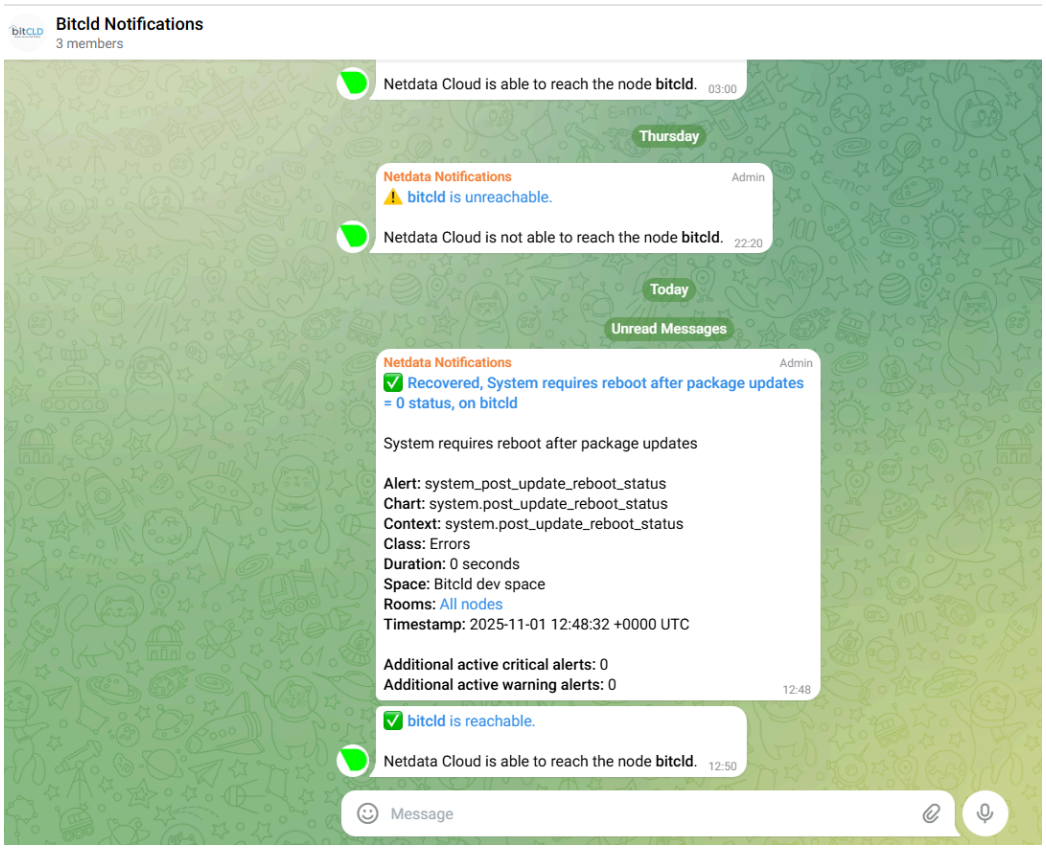


6.9 Netdata

Netdata es muy similar a Uptime kuma, en el sentido que también también tiene una interfaz para monitorear sólo que en este caso es al servidor. Además, no corre en un docker sino en nuestro servidor. Para poder ver el estado de nuestro equipo en tiempo real deberemos:

1. Acceder a `stats.lan` en local o a stats.bitcld.com.
2. Desde la interfaz tendremos acceso a estado en tiempo real de cualquier componente de nuestro equipo (CPU, RAM, Almacenamiento…):

3. Además, al igual que Uptime Kuma, también tiene la función de mandarnos notificaciones de forma pasiva al mismo grupo de Telegram por si existiera algún problema con el servidor.



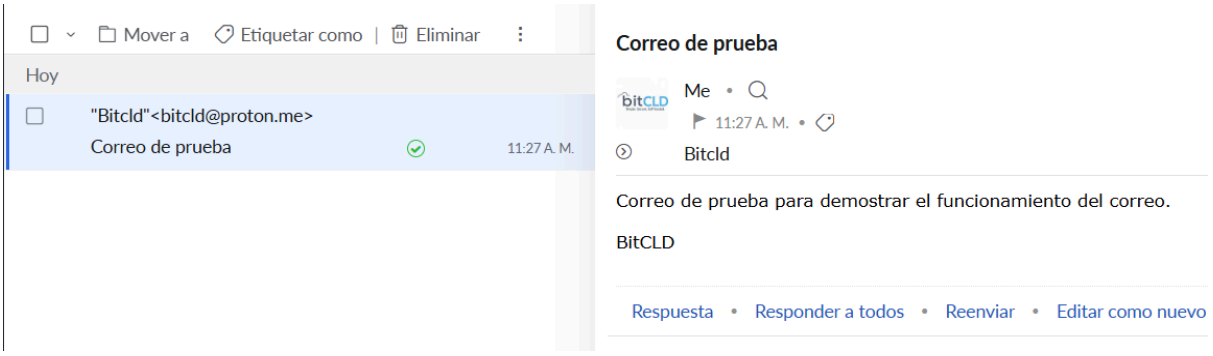
6.10 Zoho Mail

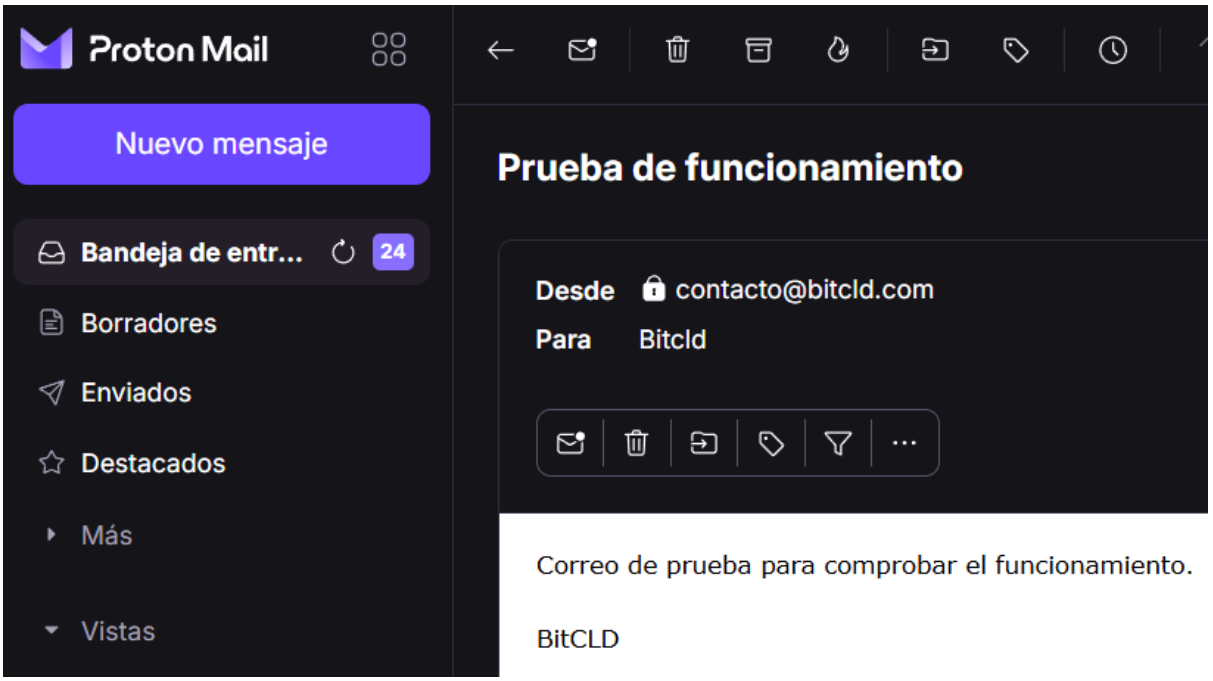
Como no es un servicio que esté autoalojado en nuestro servidor, no necesitaremos iniciarlo ya que siempre estará disponible (siempre y cuando no esté caída la red de Zoho). En cuanto a su uso es como la de cualquier otro correo electrónico.

Ahora vamos a realizar pruebas básicas de funcionamiento para comprobar la correcta entrega y recepción de correos bajo el dominio `contacto@bitcld.com`.

Envío de correo de prueba

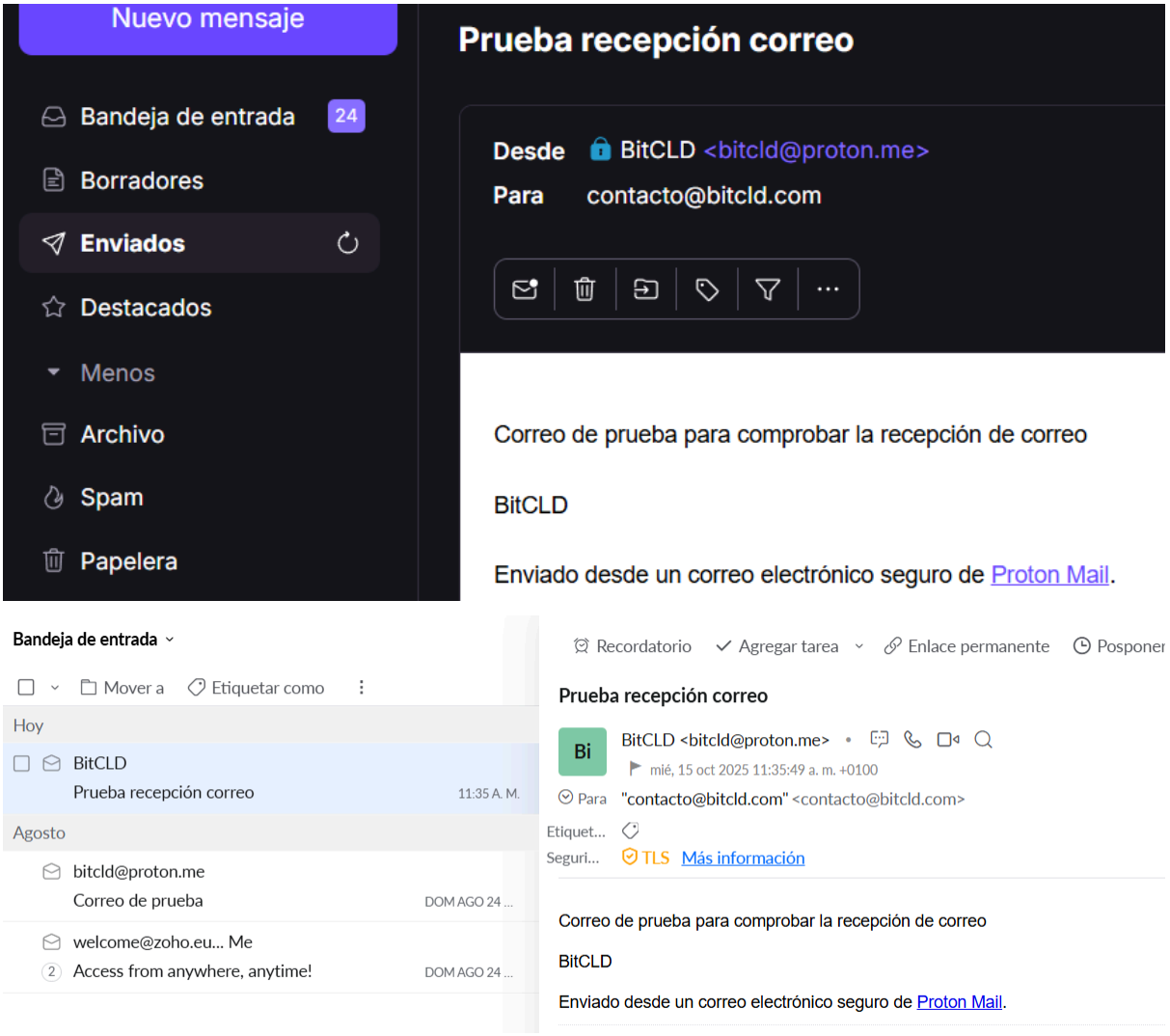
Se envió un mensaje desde una de las cuentas configuradas (`contacto@bitcld.com`) hacia una cuenta externa, en mi caso la cuenta protonmail del proyecto pero podría ser cualquier otro (Gmail, Outlook...).





Recepción de correo

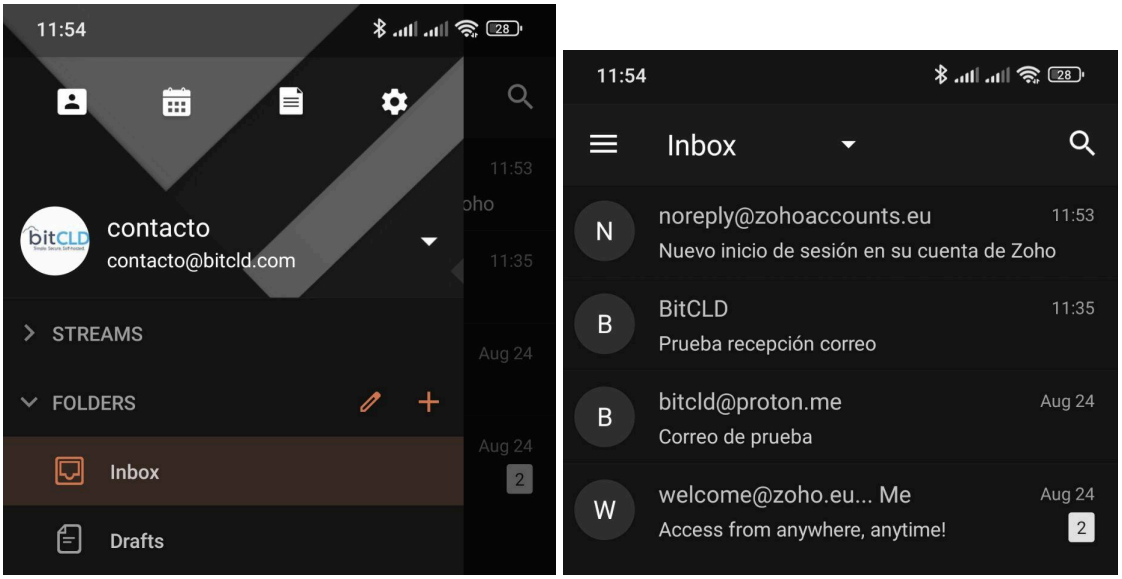
Realizamos una prueba inversa, enviando un correo desde bitcld@proton.me hacia la dirección contact@bitcld.com, comprobando que el mensaje llega al buzón en Zoho Mail sin retrasos ni errores y que la interfaz web de Zoho muestra correctamente los mensajes y notificaciones.



Como podemos observar, una vez configurado su uso es como el de cualquier otro correo electrónico.

Acceso vía aplicación móvil

Zoho Mail también nos ofrece acceso desde sus apps móviles oficiales. Vamos a verificar la sincronización en tiempo real desde la aplicación.

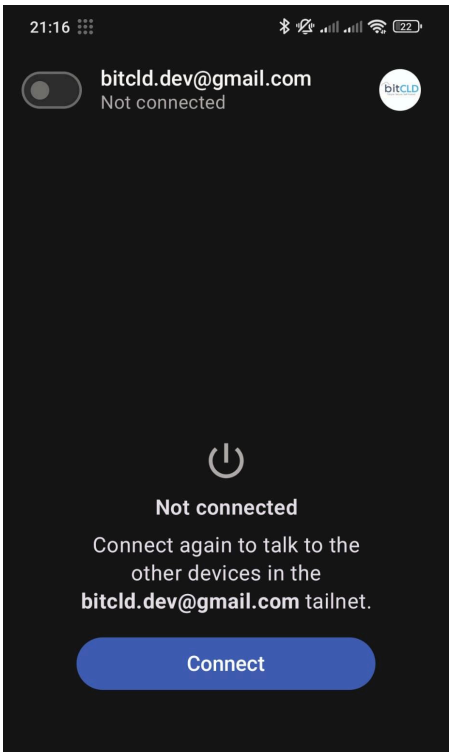


Y podemos comprobar que recibimos los correos sin ningún problema desde la aplicación móvil también.

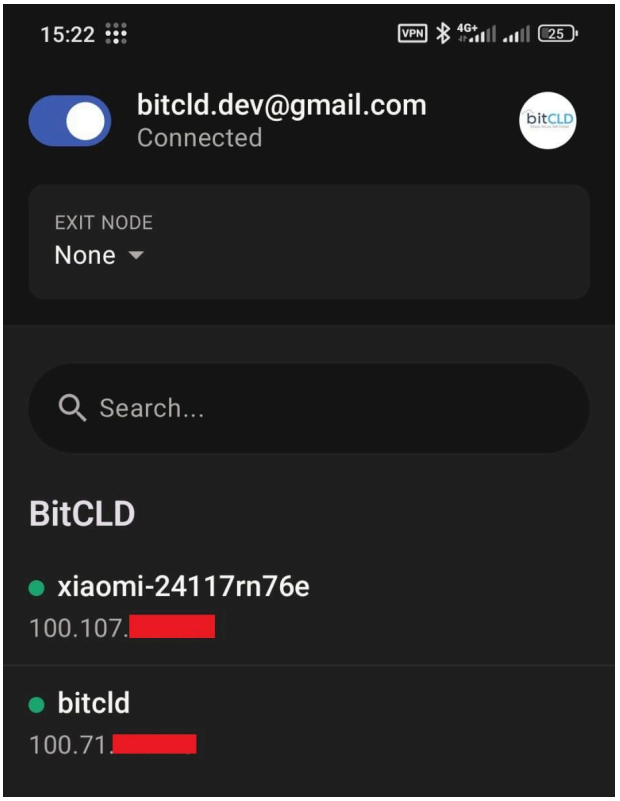
6.11 Acceso remoto al servidor [Tailscale]

Para acceder de manera remota a nuestro servidor usaremos Tailscale VPN. Para ello deberemos:

1. Abrir la aplicación de Tailscale:



2. Darle a conectar:



3. Y ya podríamos acceder de forma remota a nuestro servidor:

Además, para poder usar la aplicación de Nextcloud, es necesario que estemos conectados a nuestra VPN.

7. Riesgos y Buenas Prácticas en un Entorno Autoalojado

En este apartado hablaremos sobre los distintos riesgos que consideramos que podemos tener al estar en un entorno mayormente autoalojado y también de buenas prácticas que podemos hacer para paliar al máximo posible, los riesgos. Siendo conscientes de que no existe un sistema 100% seguro y que a pesar de usar tecnologías como Cloudflare siempre pueden haber brechas.

7.1 Riesgos de un entorno autoalojado

En cuanto a los riesgos a los que nos enfrentamos teniendo un entorno autoalojado expuesto a internet consideramos que se pueden dividir en tres principales: seguridad, operativos y de privacidad.

7.1.1 Riesgos de Seguridad

Accesos no autorizados:

Aunque estemos usando Cloudflare tunnel para exponer nuestros servicios, podemos ser víctimas de ciberdelincuentes que intenten acceder de manera ilegítima a nuestro servidor, si no gestionamos bien las políticas en Access de nuestras aplicaciones.

Configuraciones inseguras:

Un error en la configuración de cualquier servicio puede dejarnos expuestos a un ataque sin que nosotros seamos conscientes.

Falta de actualizaciones:

Otro gran error que podemos cometer es no tener actualizado en todo momento nuestro servidor y nuestras aplicaciones.

Uso de APIs y accesos:

Además, también debemos tener cuidado con el uso que le damos a nuestras APIs y otros accesos como la cuenta de Cloudflare o GitHub.

7.1.2 Operativos

Pérdida de datos:

Dependemos de nuestro disco duro y ya sea por fallos nuestros o por fallos del servidor, al no disponer de un sistema de RAID o de sincronización automática, podemos perder información y configuraciones importantes.

Caída de servicios:

Como solo disponemos un servidor para BitCLD, si el servidor fallara, se reiniciara o le ocurriera cualquier otro problema ya no podríamos acceder a ninguno de nuestros servicios.

Sobrecarga del sistema:

Al ser un equipo muy limitado debemos de tener cuidado con no sobrecargar nuestra CPU o RAM y que por ello podamos tener algún error.

Dependencia de otras empresas para acceder en remoto:

Cloudflare es muy seguro y nos ofrece un plan gratuito muy generoso, pero dependemos por completo de su red o de la de Tailscale (en casos excepcionales) para poder hacer accesible nuestros servicios al exterior.

7.1.3 De privacidad

Exponer información sensible sin querer:

Debemos de tener cuidado con lo que subimos a GitHub, en en nuestra documentación o en nuestra documentación web ya que podemos estar dando información de más o sensible

Sin control de terceros:

No podemos olvidarnos de que empresas como Cloudflare, Tailscale o Groq pueden sufrir ataques y afectarnos indirectamente.

7.2 Buenas Prácticas en un entorno autoalojado

Para intentar paliar todos los riesgos que hemos visto que podemos tener (que no son pocos), debemos poner de nuestra parte y hacer todo lo posible para minimizarlos respecto a los tres principales riesgos anteriores.

7.2.1 Seguridad

Usar siempre Cloudflare Access:

Debemos usar, en todas nuestras aplicaciones y en todo momento, Cloudflare Access con una política lo más restrictiva posible. En nuestro caso mediante sólo tres emails posibles y con restricción de país.

Autenticación 2FA:

Para proteger nuestras cuentas es casi imprescindible que establezcamos el doble factor de autenticación en nuestras cuentas de Cloudflare, GitHub, Tailscale y Groq.

Cambio periódico de APIs y tokens:

Por seguridad también es recomendable que de vez en cuando cambiemos nuestras credenciales como con Tailscale, Groq, o incluso de manera esporádica nuestro propio túnel de cloudflared.

Usar HTTPS:

Siempre que sea posible usar nuestros servicios mediante su subdominio de bitcld.com o mediante los dominios .lan para que así nuestro tráfico siempre esté cifrado gracias a Cloudflare Tunnel o a Caddy.

Mantener el servidor y servicios actualizados:

Debemos asegurarnos de que en todo momento tenemos nuestro sistema operativo y nuestros servicios actualizados para evitar vulnerabilidades.

7.2.2 Operativas

Hacer copias de seguridad periódicamente:

Para evitar al máximo posible la pérdidas de datos debemos de hacer copias de seguridad de nuestros datos, idealmente que no estén en nuestro servidor como por ejemplo en nuestro ordenador o en otra nube como Mega o Proton drive. Además de guardar todas nuestras configuraciones más importantes de cada servicio.

Monitorización del servidor:

Para evitar que el equipo se sobrecargue es importante que monitoreemos nuestro servidor para asegurarnos que no estamos haciendo un uso excesivo de la CPU, RAM u otro componente. Para ello tenemos a Netdata y el envío de notificaciones en nuestro grupo de Telegram.

Documentar todos los cambios:

Algo interesante también es tener un documento, una vez entreguemos la documentación, en el que añadamos todos los cambios que hagamos posteriormente a la entrega. Para así poder tener un registro de bitCLD.

7.2.3 De Privacidad

Exponer información sensible:

Debemos siempre evitar almacenar configuraciones sensibles en GitHub, sobretodo en repositorios públicos, o incluso en nubes externas. En caso de ser necesario deberemos cifrarlas antes de subirla.

8. Conclusión general final

La realización de nuestro proyecto, bitCLD, nos ha permitido crear una infraestructura completa, moderna y coherente orientada a la privacidad y al control de nuestros datos. Durante el desarrollo del proyecto hemos podido trabajar con distintos servicios y tecnologías, los cuales seleccionamos y configuramos para lograr tener un ecosistema completo, moderno y realista.

Con la creación de nuestro ecosistema hemos podido trabajar con sistemas operativos linux (Ubuntu Server), contenedores (docker, docker compose y dockge), redes, seguridad y acceso remoto (Cloudflare, Tailscale y Caddy), servidores DNS (Adguard Home), almacenamiento en la nube (Nextcloud), inteligencia artificial tanto en local como con API (Open WebUI, Ollama y Groq), gestores de contraseña (Vaultwarden), monitoreo (Uptime Kuma y Netdata), correo electrónico (Zoho Mail) y publicación de de páginas webs (GitHub, GitHub y Cloudflare Pages y Mkdocs material).

Esto nos ha permitido obtener un enorme aprendizaje, dándonos la oportunidad de tener una visión más global de cómo se interrelacionan las distintas tecnologías en un entorno autoalojado gestionado por nosotros donde nos vimos obligados a tomar decisiones técnicas, resolver las incidencias que nos encontrábamos y optimizar al máximo para no sobrecargar nuestro servidor.

Para concluir la valoración final de nuestro proyecto es muy positiva. BitCLD cumplió eficazmente con los objetivos que nos planteamos terminando con un ecosistema robusto, escalable y bien documentado, construido con decisiones bien justificadas y una clara orientación a la seguridad y privacidad. Además, gracias a los conocimientos que hemos adquirido a lo largo del proyecto tenemos una buena base técnica para mejorarlo y ampliarlo en el futuro permitiéndonos también, demostrar esos conocimientos de manera práctica a futuros empleadores.

9. Bibliografía

[Vídeo Youtube] Crea tu propia nube con DOCKER y CasaOS | Odisea Homelab #1 Hector Pulido https://www.youtube.com/watch?v=27IiYo37Iww

[Vídeo Youtube] CLOUDFLARE: La forma MÁS FÁCIL de acceder a tu NUBE CASERA | Odisea Homelab #2 Hector Pulido
https://www.youtube.com/watch?v=I3ot7jD7XOY&t=759s

Página web oficial de Cloudflare
https://www.cloudflare.com/es-es/

Documentación oficial de Cloudflare
https://developers.cloudflare.com/cloudflare-one/networks/connectors/cloudflare-tunnel/

Página web oficial de docker
https://www.docker.com/

Página web oficial de documentación de docker
https://docs.docker.com/

[Vídeo Youtube] DOCKER De ZERO a HERO! (ACTUALIZADO 2025 - CURSO COMPLETO EN ESPAÑOL) Hector Pulido
https://www.youtube.com/watch?v=m0N80toLnlE

Búsqueda de imágenes de servicios
https://hub.docker.com/

[Vídeo Youtube] This Docker Compose UI is amazing! // Dockge
https://www.youtube.com/watch?v=HEklvsr7q54&t=34s

[Vídeo Youtube] How To Setup Notifications On Uptime Kuma a Self Hosted Monitoring Application
https://www.youtube.com/watch?v=Cp-rAatifMk

[Vídeo Youtube] Truco fácil para obtener API Key de OpenAI sin pagar en 2025 Hector Pulido
https://www.youtube.com/watch?v=AhJ-EDuDMC4

Página web oficial de Ollama
https://ollama.com/

Ruta de Ollama para buscar modelos locales
https://ollama.com/search

Documentación oficial de Open Web UI
https://docs.openwebui.com/getting-started/quick-start/

Página web oficial de Groq
https://groq.com/

Documentación oficial de Groq
https://console.groq.com/docs/overview

Página web oficial de Ubuntu
https://ubuntu.com/download/server

Página web oficial de Dockge
https://dockge.kuma.pet/

Página web oficial de Caddy Server (reverse proxy)
https://caddyserver.com/

Página web oficial de Tailscale VPN
https://tailscale.com/

Página web oficial de Uptime Kuma
https://uptime.kuma.pet/

Página web oficial de Nextcloud
https://nextcloud.com.es/

Página web oficial de Vaultwarden
https://www.vaultwarden.ca/

Página web oficial de AdguardHome
https://adguard.com/es/adguard-home/overview.html

Página web oficial de Zoho
https://www.zoho.com/

Netdata
https://www.netdata.cloud/

GitHub
https://github.com/

[Vídeo Youtube] Crear documentacion con MkDocs OpenWebinars
https://www.youtube.com/watch?v=7H3DpzqCfME

Página web para Mkdocs en español
https://markdown.es/

A. Anexos

En este apartado añadiremos otras tecnologías que hemos usado en el proyecto, pero que no pertenecen en sí al ecosistema sino que nos sirven para darle un toque más profesional. Además también añadiremos la creación y configuración de bots en Telegram y todas las configuraciones sensibles de nuestro proyecto las cuáles incluyen los docker-compose de cada servicio, el Caddyfile completo, reglas DNS y reglas de Access.

A.1 Otras tecnologías usadas en el proyecto

A.1.1 GitHub

Introducción

GitHub es una plataforma de alojamiento de código y proyectos basada en Git, un sistema de control de versiones que permite guardar, organizar y seguir la evolución de cualquier archivo, especialmente de desarrollo de software y documentación. A través de repositorios, ramas y commits, GitHub facilita trabajar de forma ordenada, colaborar con otras personas, revisar cambios y mantener un historial completo de todo lo que se hace en un proyecto. Además, incluye funciones adicionales como seguimiento de issues, gestión de versiones y publicación de sitios web estáticos mediante GitHub Pages, lo que lo convierte en una herramienta muy completa para centralizar y proteger el trabajo de desarrollo y documentación.

Motivo de elección

Hemos elegido GitHub ya que es una solución que se usa mucho en entornos profesionales y me quería tener contacto con ella para poder acostumbrarme a su uso. Además gracias a GitHub Pages nos permite publicar nuestra web sin ningún coste adicional y nos ofrece una alta disponibilidad para tener una copia de seguridad de nuestra web y nuestra documentación web.

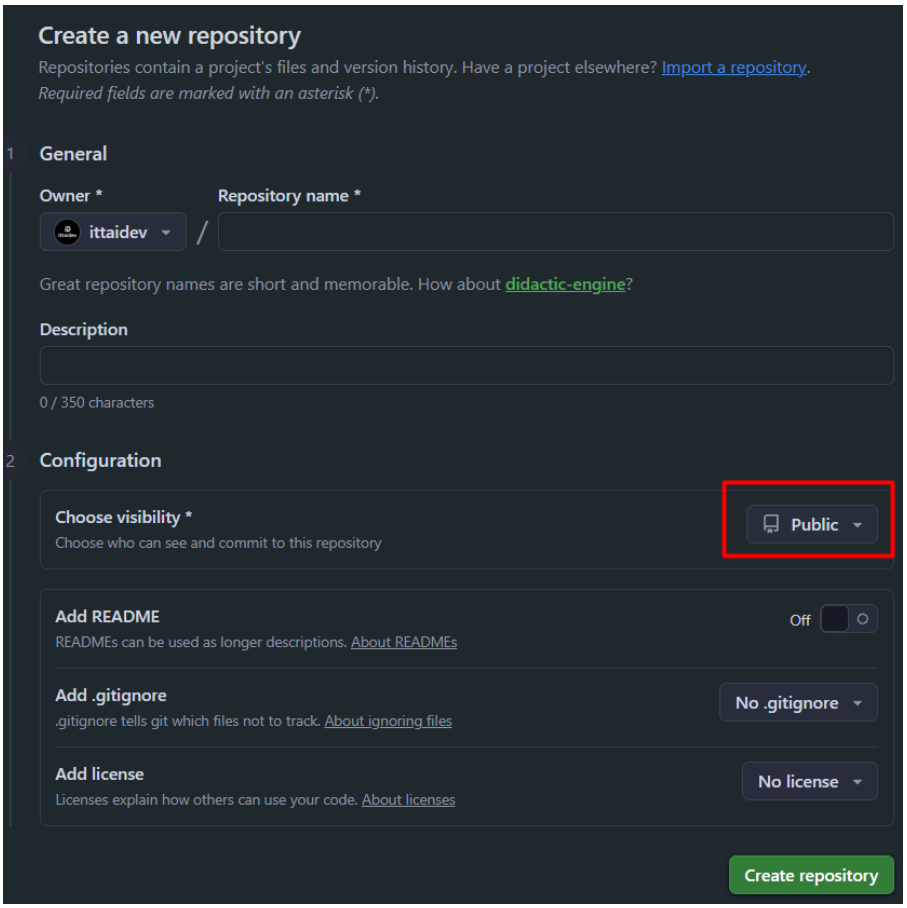
Función dentro del proyecto

Dentro de de bitCLD, GitHub cumple tres funciones principales ya que actúa como repositorio para tener un backup de nuestra documentación dándonos también la oportunidad de conectar Cloudflare Pages con el respositorio y así cualquier cambio que hagamos en GitHub se vea reflejado. En segundo lugar, nos permite almacenar el código de la página web del proyecto. Y, en tercer lugar, GitHub Pages la usamos para publicar la web del proyecto de forma estática, sirviéndonos para darle una imagen más profesional al proyecto, sin necesidad de tener que gastar dinero en hostings.

Creación de repositorio

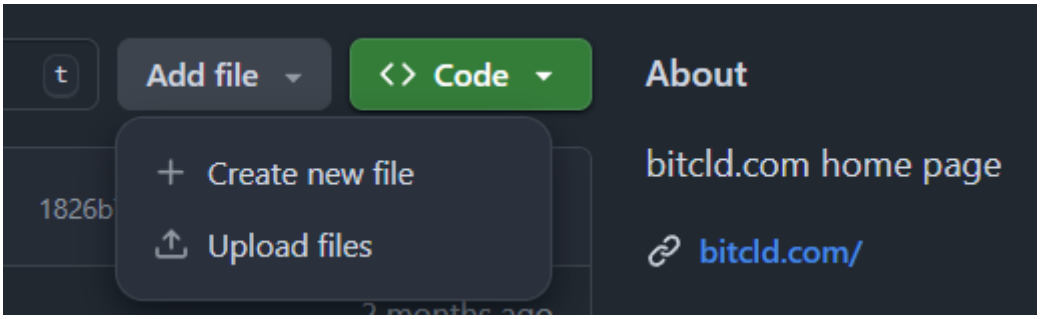
Para la creación de un repositorio es muy sencillo, simplemente debemos crearnos una cuenta en GitHub y seleccionar New Repository.

Deberemos añadir un nombre, y escoger la visibilidad. En este caso seleccionaremos público ya que es para la página web de nuestro proyecto, sin embargo, cuando lo usamos como copia de seguridad lo ponemos en privado.



Si vamos a usar GitHub Pages el repositorio deberá tener la visibilidad en Público

Una vez creado para añadir archivos deberemos seleccionar Add file > Upload files

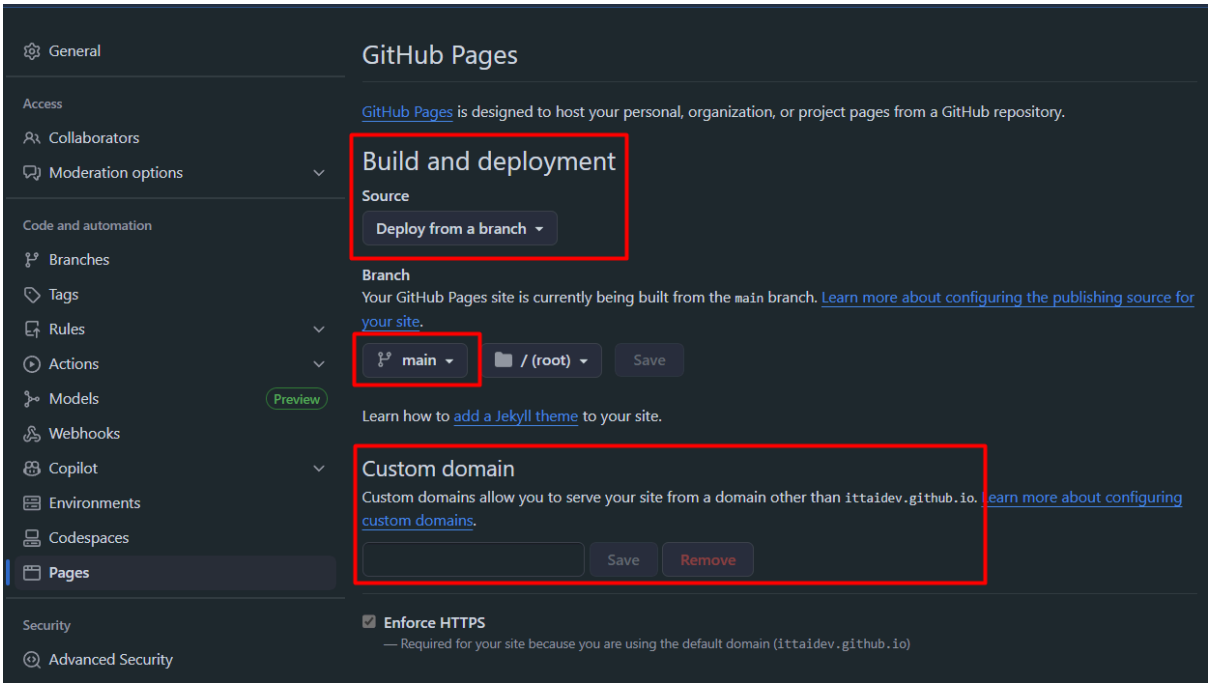


Una vez subidos los archivos ya tendremos nuestra copia de seguridad de nuestra página web. También podemos añadir un Readme para que quede más visual para nosotros mismos y para que cualquier persona pueda tener una idea de qué hay en el repositorio a simple vista.

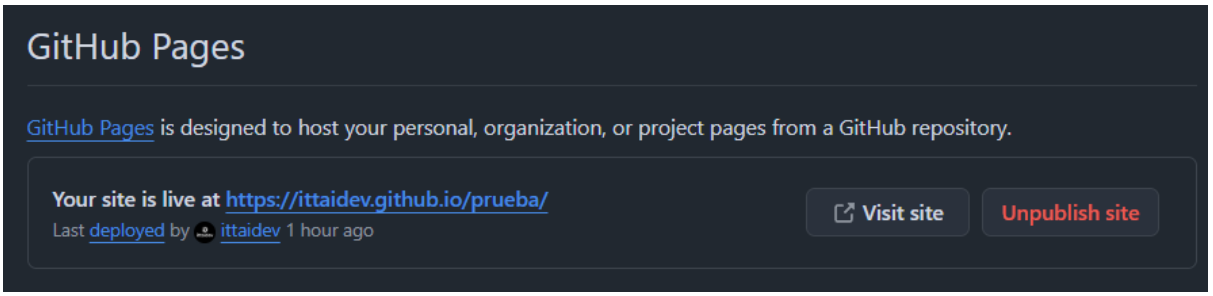
Publicación de la web

Para publicar nuestra web mediante GitHub, en el repositorio público que deseemos, seleccionamos Settings > Pages

En Build and deployment dejaremos la opción de Deploy from a brach, escogeremos la branch main y dejaremos marcado el Enforce HTTPS.



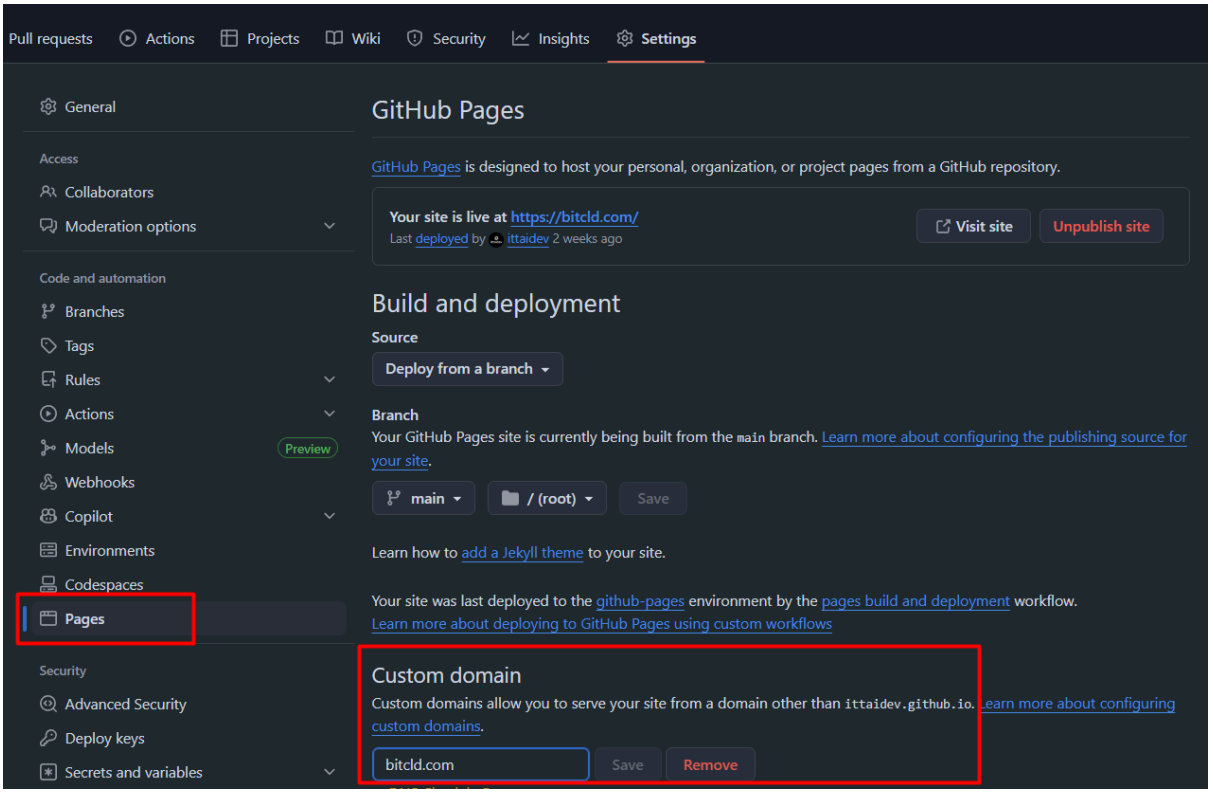
Una vez hecho después de unos momentos ya tendremos nuestro sitio web publicado en internet y podremos acceder a él mediante el enlace que nos ofrece GitHub.



En nuestro caso, al disponer del dominio `bitcld.com`. Lo añadiremos en Custom domain, para ello primero deberemos añadir los registros DNS correspondientes en Cloudflare.

Type	Name	Content	Proxy status	TTL
A	bitcld.com	185.199.108.153	Proxied	Auto
A	bitcld.com	185.199.109.153	Proxied	Auto
A	bitcld.com	185.199.110.153	Proxied	Auto
A	bitcld.com	185.199.111.153	Proxied	Auto

Añadimos el dominio y una vez haya replicado, ya tendremos nuestra página web desplegada a través de nuestro dominio personalizado. En https://bitcld.com/ .



Conclusión

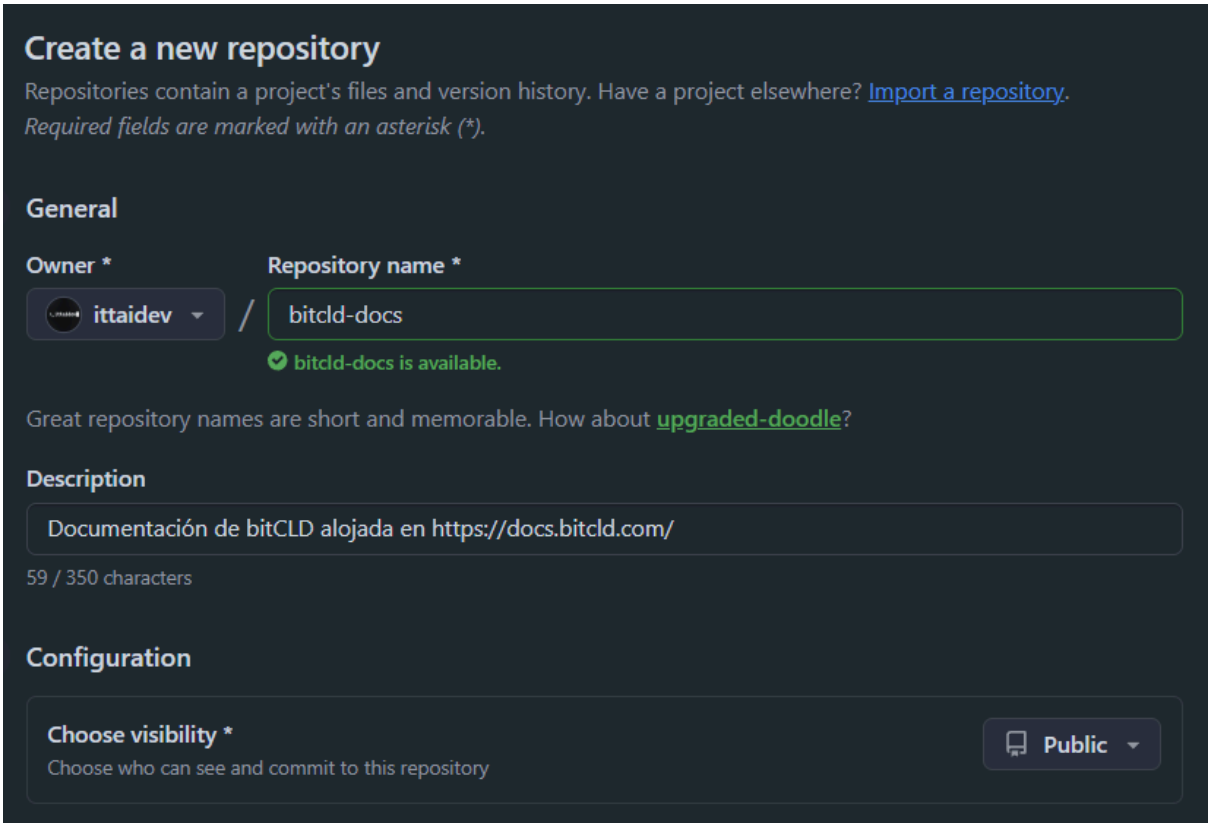
GitHub lo usamos para poder tener protegido todo el trabajo nuestro proyecto ya que podemos guardar la documentación y nuestra página web. Además, nos permite publicar nuestra web del proyecto mediante GitHub Pages y para conectar a Cloudflare la documentación. Gracias a esto, disponemos de una copia de seguridad en la nube de todos nuestros archivos más importantes, y también tenemos la oportunidad de trabajar con una plataforma profesional que nos permite darnos a conocer como profesionales y dar a conocer el proyecto bitCLD.

A.1.2 Cloudflare Pages

Ya hemos hablado de todas las ventajas que nos ofrece la Plataforma de Cloudflare. Es por ello que sólo añadiremos el proceso de publicación de una página web mediante Cloudflare Pages, en nuestro caso será la documentación del proyecto. De esta manera, así podemos trabajar con GitHub y Cloudflare Pages y así decidir el que mejor nos convenga.

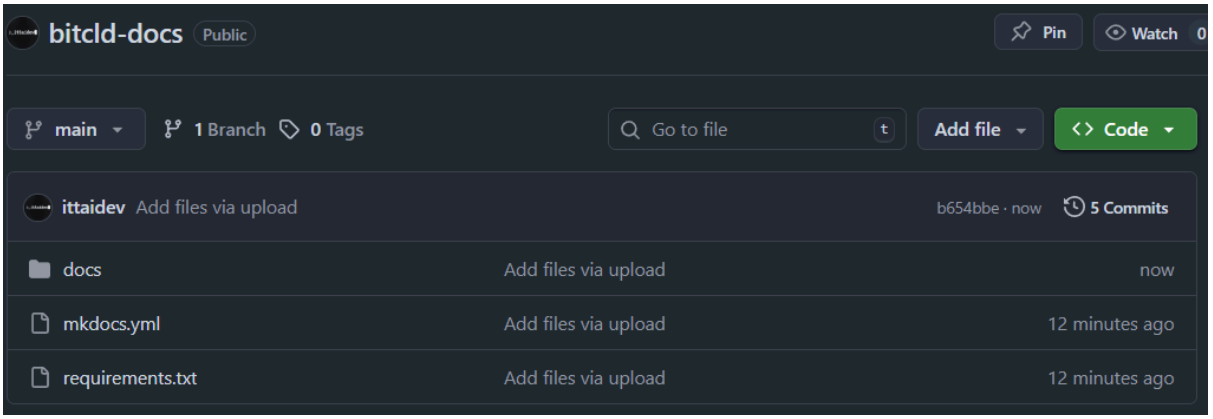
Creación del repositorio en GitHub

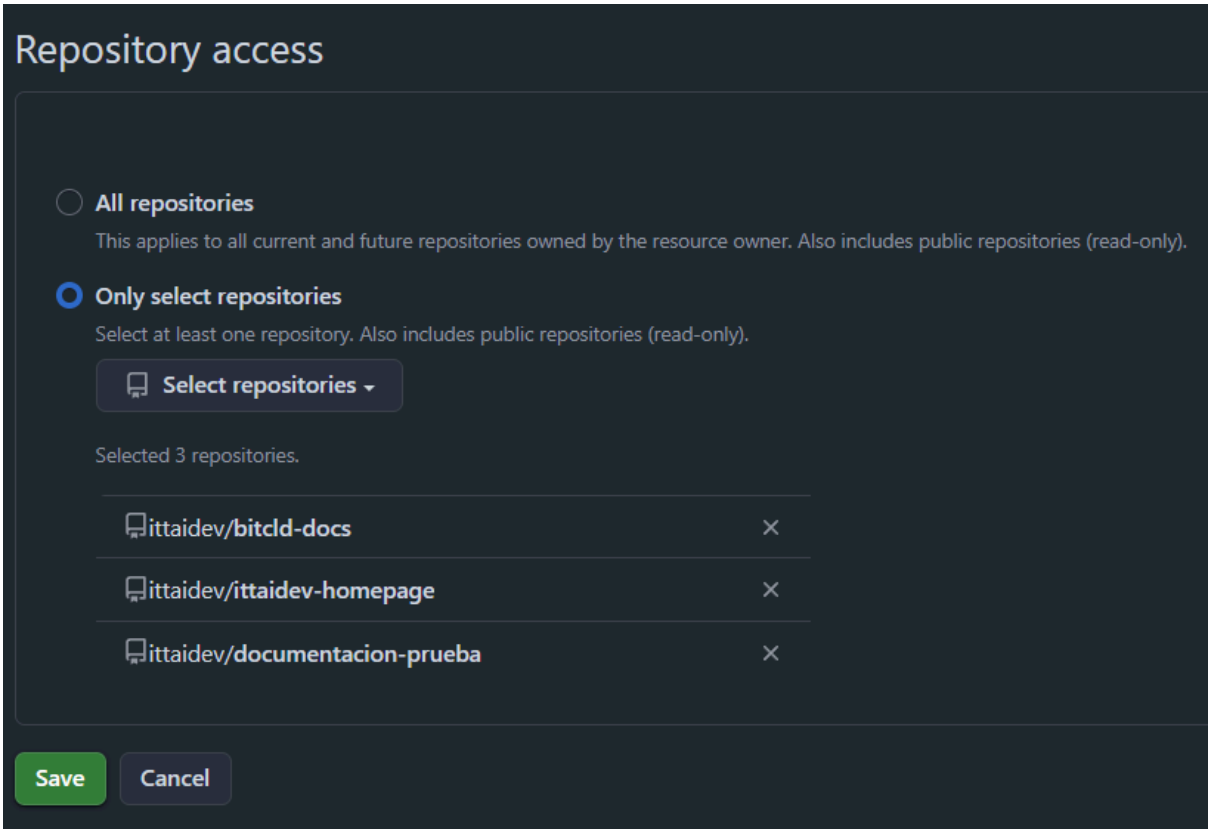
Para ello primero deberemos crear un repositorio en GitHub, no importa si es público o privado ya que le daremos acceso a Cloudflare Pages directamente al repositorio.



Una vez creado subiremos nuestros archivos, para Cloudflare instale MkDocs necesitaremos añadir un documento en la raíz del repositorio que se llame requeriments.txt con el siguiente contenido:

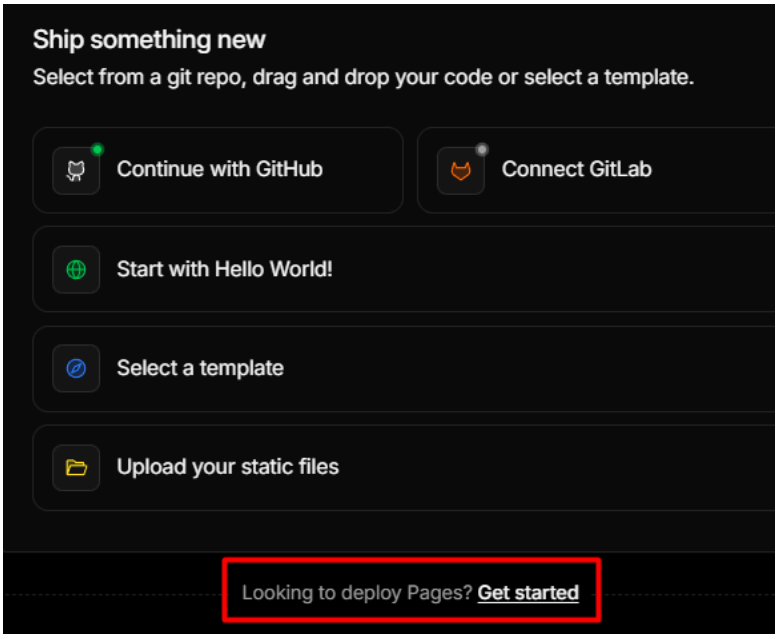
```
Mkdocs
mkdocs-material
```


bitcld-docs
Public
Pin
Watch 0
main
1 Branch
0 Tags
Go to file
t
Add file
Code
ittaidev Add files via upload
b654bbe · now
5 Commits
docs
Add files via upload
now
mkdocs.yml
Add files via upload
12 minutes ago
requirements.txt
Add files via upload
12 minutes ago

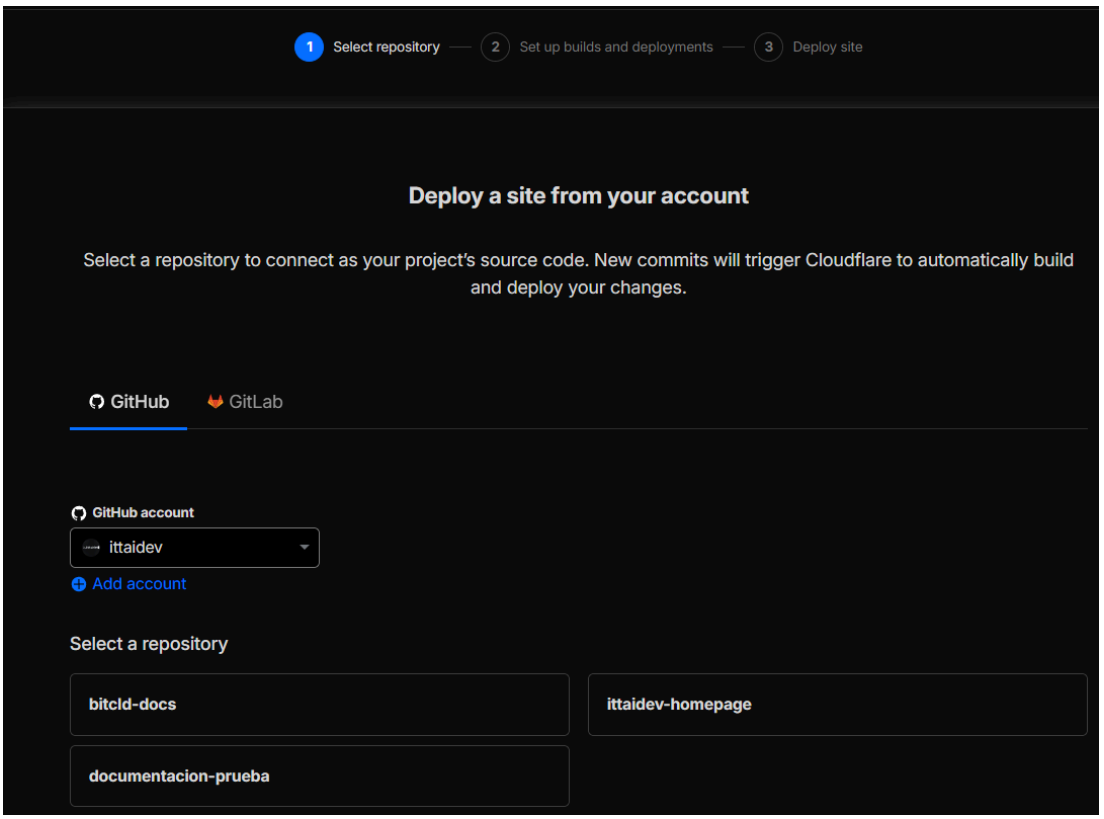

Repository access
All repositories
This applies to all current and future repositories owned by the resource owner. Also includes public repositories (read-only).
Only select repositories
Select at least one repository. Also includes public repositories (read-only).
Select repositories
Selected 3 repositories.
ittaidev/bitcld-docs
ittaidev/ittaidev-homepage
ittaidev/documentacion-prueba
Save
Cancel

Despliegue de la web

Una vez hayamos creado nuestro repositorio, en Cloudflare, abriremos Workers & Pages > Create Application. Y seleccionamos la opción de Pages que encontramos abajo:



1. Una vez en Pages, conectaremos nuestra cuenta de GitHub con Cloudflare y seleccionamos el repositorio que queremos publicar. En nuestro caso le dimos acceso restringido a los repositorios por eso sólo se ven el de la documentación, nuestra web principal y el de pruebas.

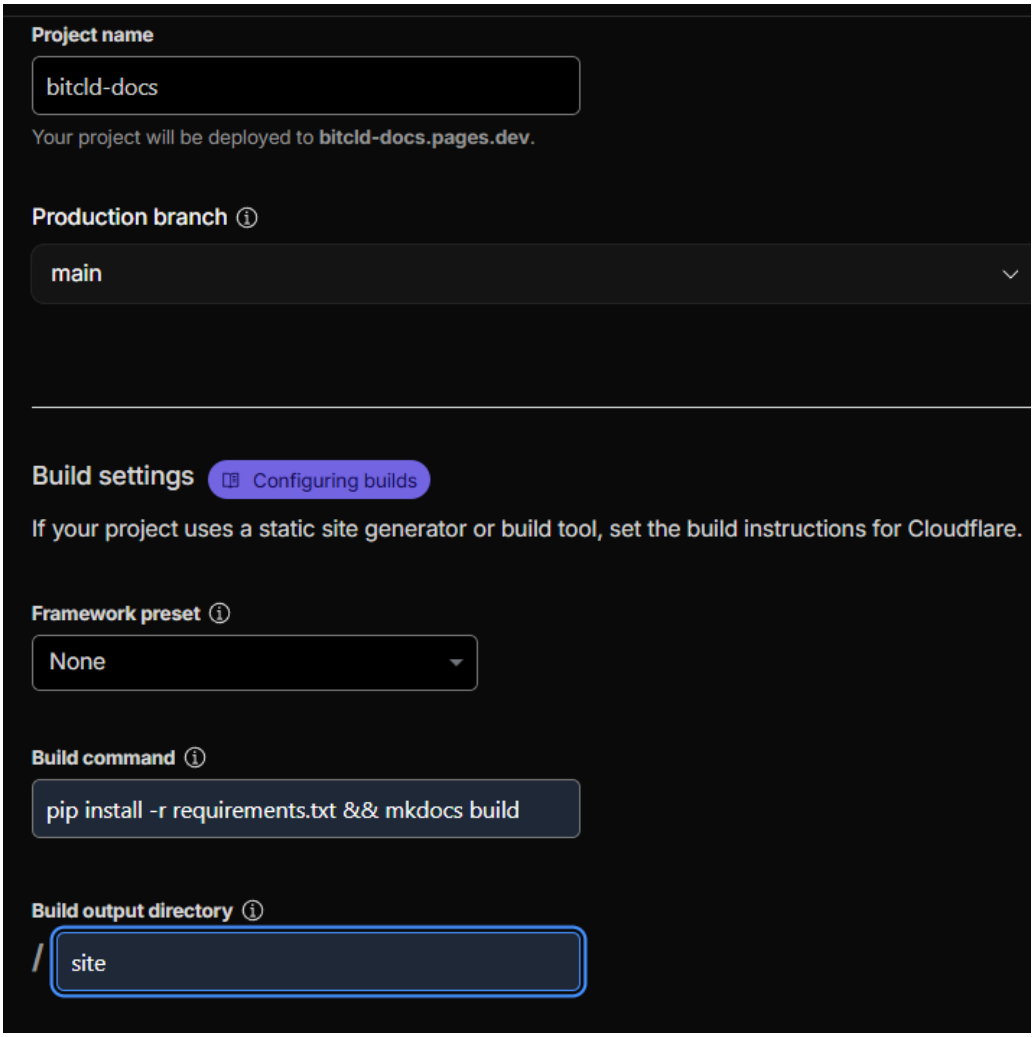


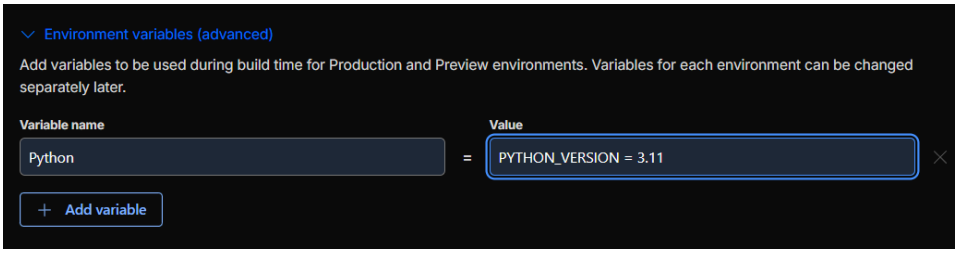
2. Elegimos un nombre para el proyecto (en nuestro caso el mismo que el del repositorio) y en la configuración añadimos:

- ➔ Production branch: main (ya que es la única rama tenemos)
- ➔ Framework preset: none
- ➔ Build command: pip install -r requirements.txt && mkdocs build
- ➔ Build output directory: site

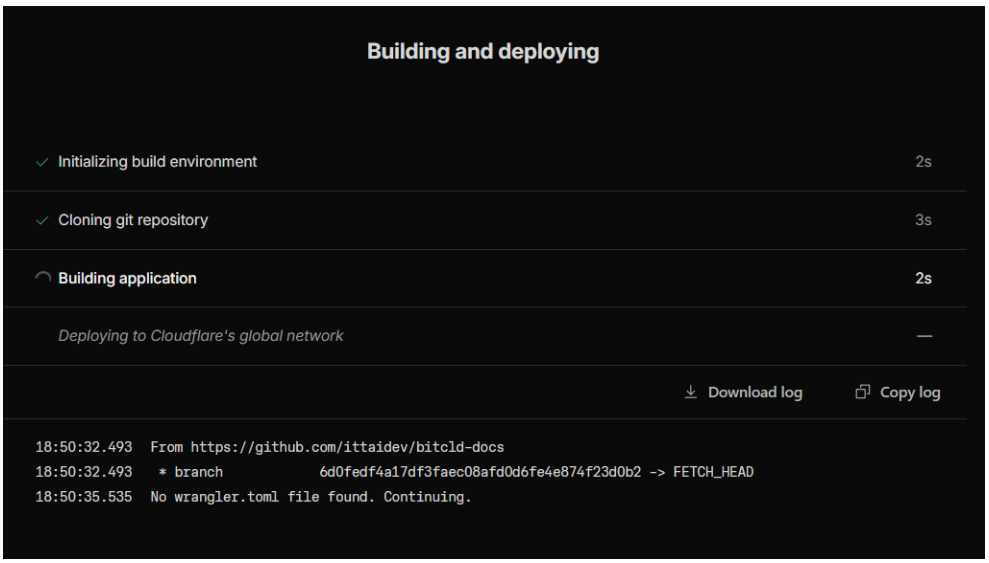
Y en environment variables (advanced) añadimos:

- ➔ PYTHON_VERSION = 3.11

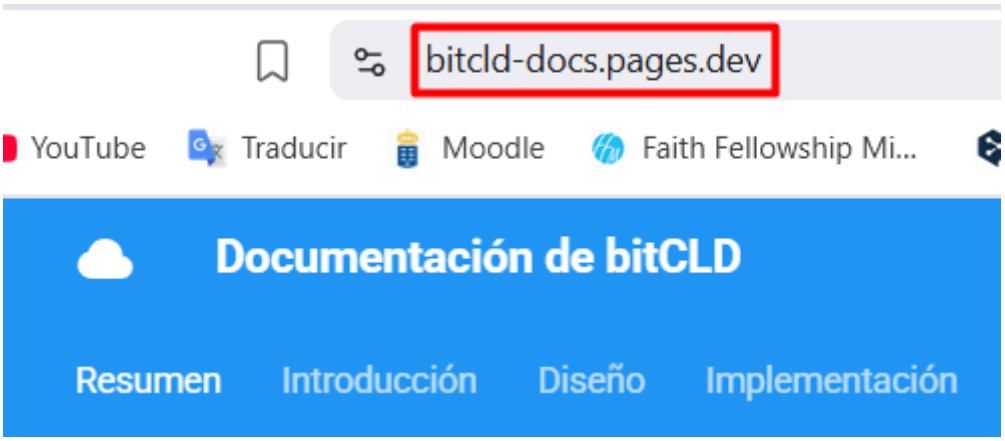




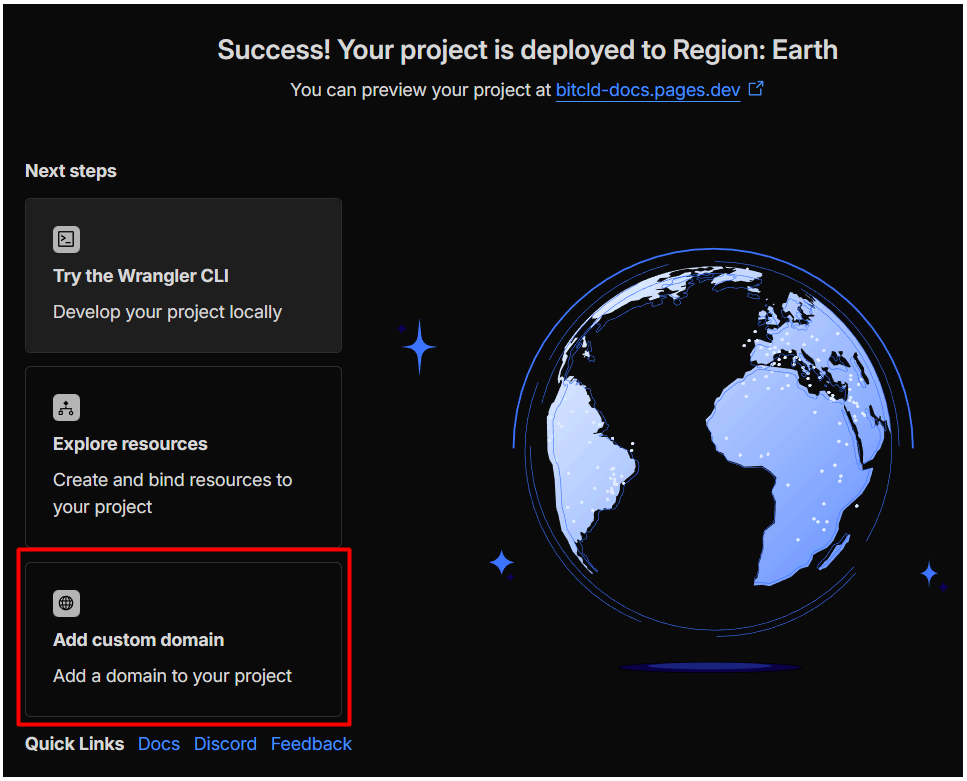
3. Una vez añadida las configuraciones Cloudflare Pages comenzará a construir y desplegar nuestra web:



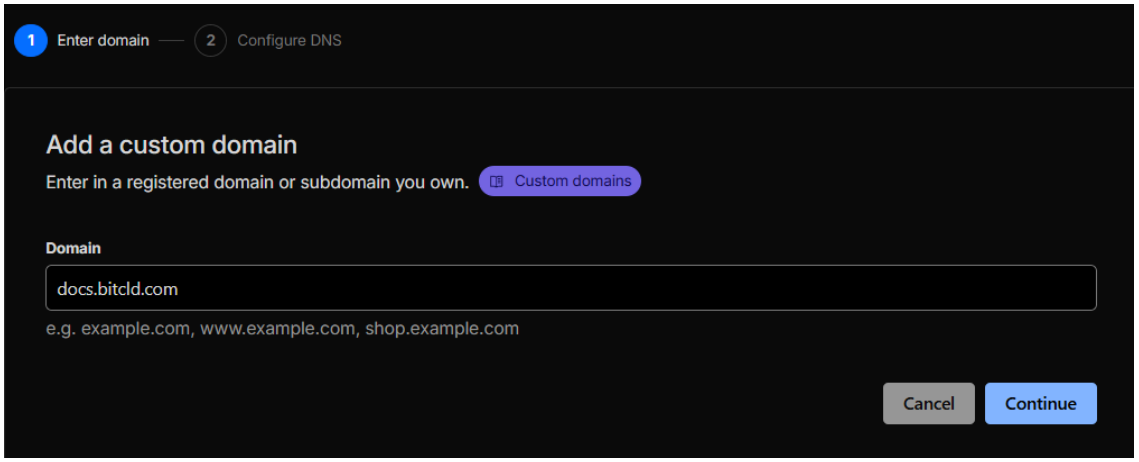
4. Cuando finalice ya podremos acceder a nuestra web desde el enlace que nos proporciona la plataforma: https://bitcld-docs.pages.dev/ :



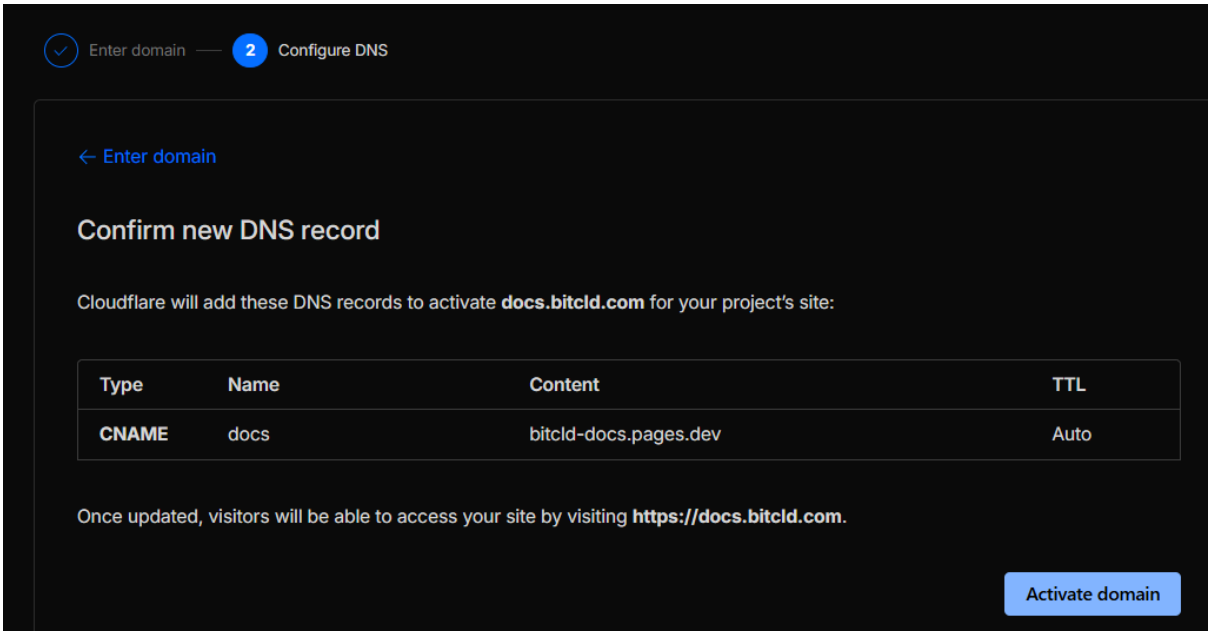
5. Sin embargo, lo que nosotros buscamos es usar nuestro dominio del proyecto para que quede más profesional y coherente. Elegimos docs.bitcld.com . Para ello seleccionamos la opción de Add custom domain:



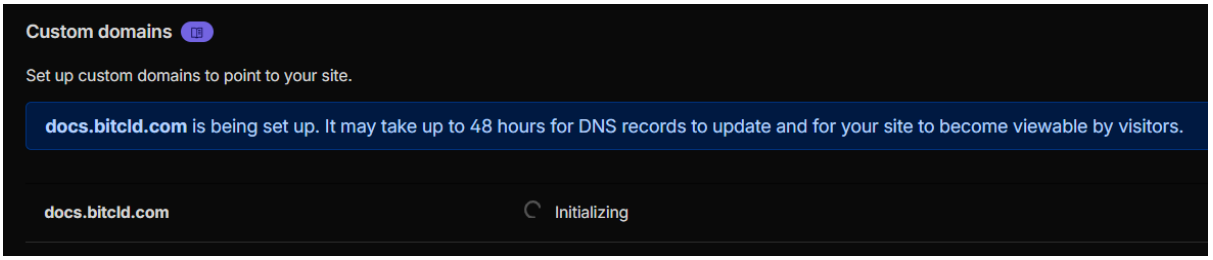
6. Añadimos el subdominio que queremos:



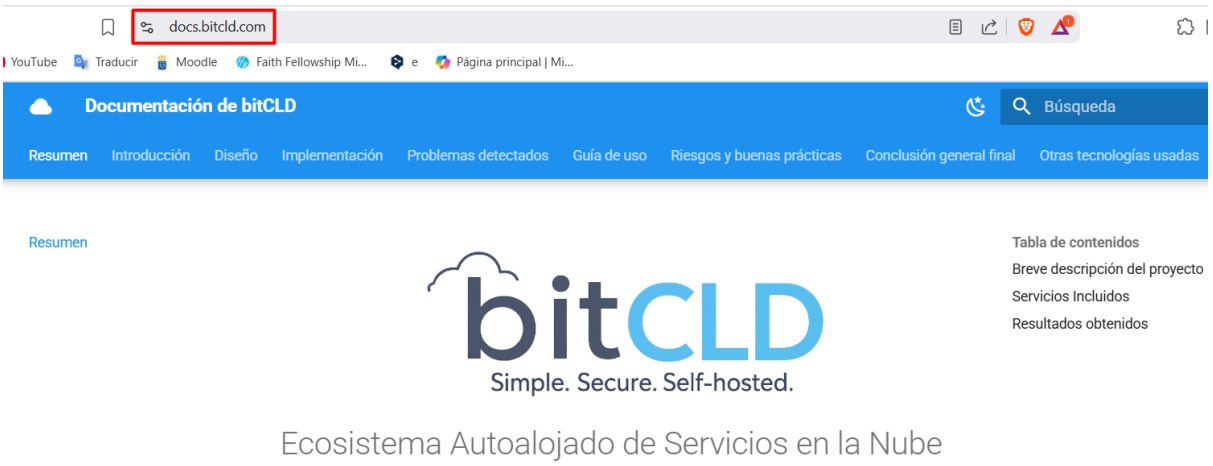
7. Al tener nuestro dominio en Cloudflare, la plataforma se encargará de generar el registro DNS automáticamente:



8. Después ya sólo nos queda esperar a que replique. En nuestro caso fue bastante rápido.



Una vez haya replicado ya podremos acceder a nuestra documentación web hecha en Mkdocs con el tema de Material desde nuestro subdominio:



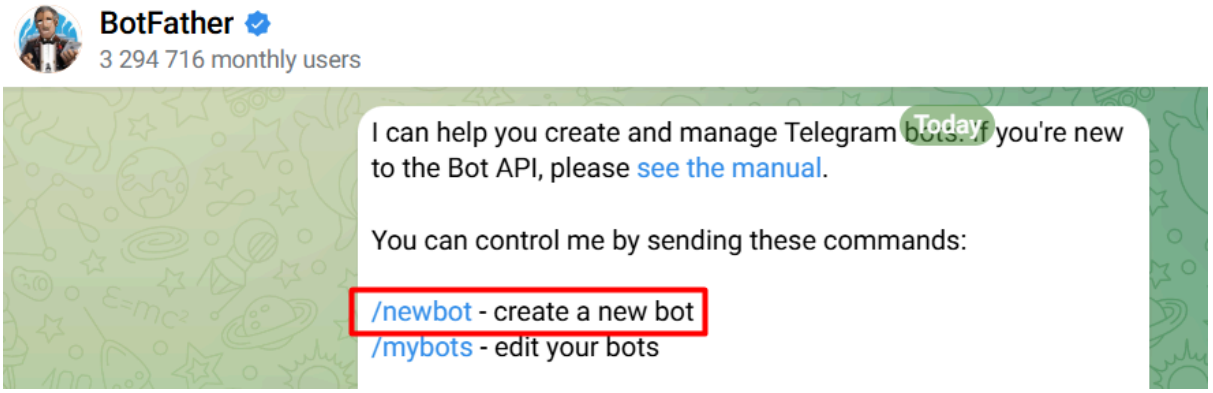
A.2 Bots en Telegram

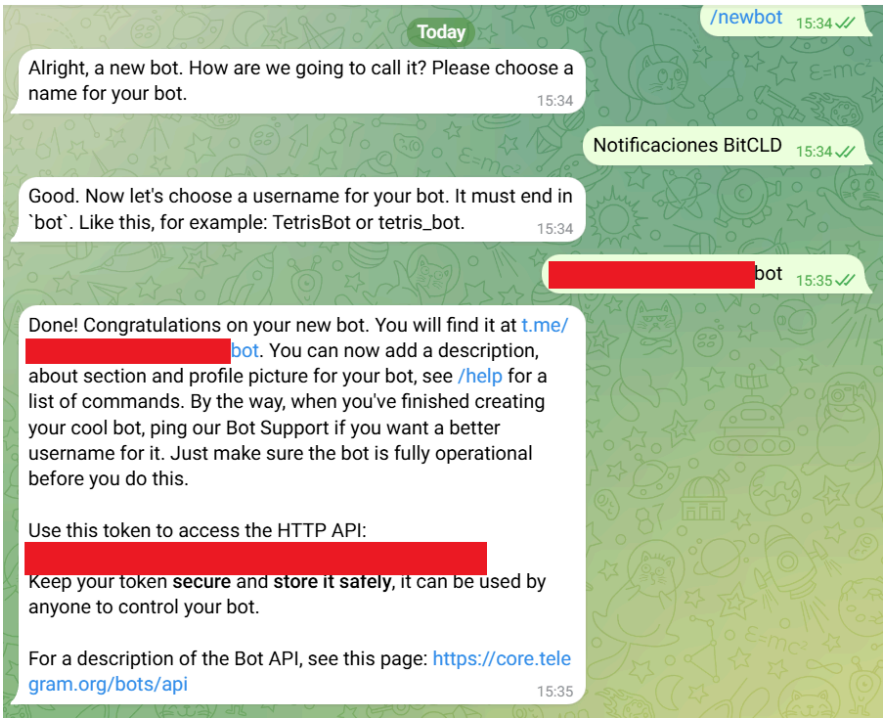
En este anexo explicaremos la creación y configuración de bots en Telegram.

Creación de un bot de telegram

Para crear un bot seguiremos los siguientes pasos:

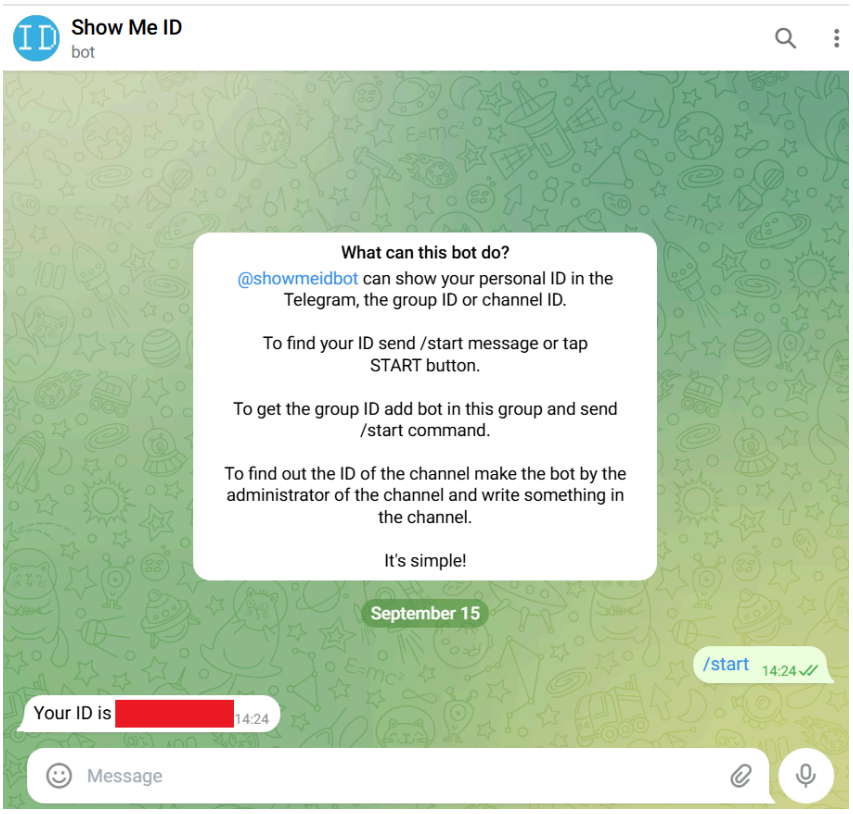
1. Iniciamos un chat con @BotFather
2. Seleccionamos la opción de /newbot
3. Indicamos un nombre
4. Indicamos un nombre de usuario para el bot
5. Ya tendríamos el bot creado





Uso del bot en conversación directa

En caso de quererlo como una conversación con el bot. Una vez tengamos el bot creado necesitaremos conocer nuestro ID de chat, para ello existen diversos bots que ofrecen eso. En nuestro caso elegimos el bot de @showmeidbot. Simplemente debemos iniciar una conversación con él:



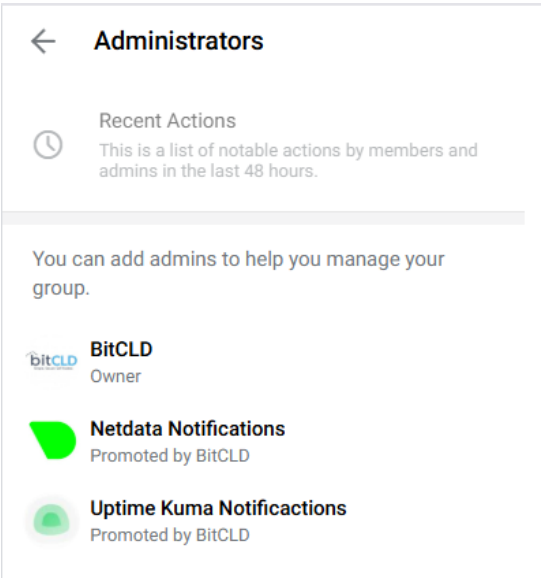
Con el bot creado y el ID del chat de nuestra conversación ya podremos crear la notificación en Uptime Kuma o en Netdata.

Uso del bot en un grupo

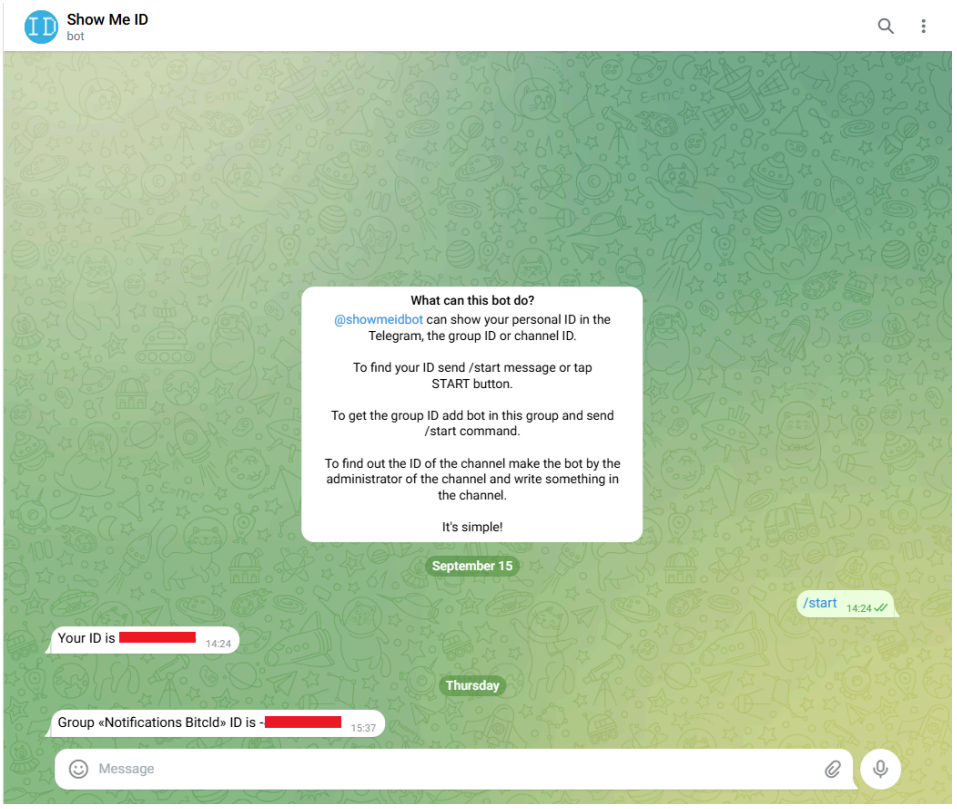
En caso de querer el bot en un grupo de Telegram, como será nuestro caso, ya que al tener un bot para las notificaciones de Uptime Kuma y otro para las de Netdata por comodidad es mejor tener los dos bots en un grupo de Telegram.

En este caso la creación del bot sería la misma.

1. Debemos crear un grupo de telegram
2. Añadimos ambos bots en nuestro grupo como administradores



3. Para que Uptime Kuma, Netdata y cualquier otro servicio nos envíe notificaciones necesitaremos el ID chat del grupo, por lo que añadiremos el bot al grupo para conocerlo. Una vez añadido al grupo, en la conversación con el bot nos pondrá el ID chat del grupo.



A.3 Docker compose

En este anexo añadiremos los docker compose de cada servicio para que sólo sea necesario copiar y pegar.

Dockge

```
services:
dockge:
    image: louislam/dockge:latest
    container_name: dockge
    restart: unless-stopped
    ports:
    - "5001:5001"
    volumes:
    - /var/run/docker.sock:/var/run/docker.sock
    - ./data:/app/data
    - /srv/docker/stacks:/srv/docker/stacks
    environment:
    - DOCKGE_STACKS_DIR=/srv/docker/stacks
```

Caddy

```
services:
caddy:
    image: caddy:latest
    container_name: caddy
    restart: unless-stopped
    ports:
    - 80:80
    - 443:443
    volumes:
    - ./Caddyfile:/etc/caddy/Caddyfile:ro
    - ./data:/data
    - ./config:/config
    networks:
    - caddy_net

networks:
caddy_net:
    external: true
```

Adguard Home

```
services:
adguardhome:
    image: adguard/adguardhome:latest
    container_name: adguardhome
    restart: unless-stopped
    ports:
    - 192.168.1.5:53:53/tcp
    - 192.168.1.5:53:53/udp
    - 100.71.42.126:53:53/tcp
    - 100.71.42.126:53:53/udp
    - 192.168.1.5:3000:3000/tcp
    volumes:
    - ./workdir:/opt/adguardhome/work
    - ./confdir:/opt/adguardhome/conf
    environment:
    - TZ=Atlantic/Canary
    networks:
    - caddy_net
networks:
caddy_net:
    external: true
```

Nextcloud

```
services:
app:
    image: nextcloud:latest
    container_name: nextcloud-app
    restart: unless-stopped
```

Inteligencia Artificial: Ollama + Open WebUI

```
services:
ollama:
    image: ollama/ollama:latest
    container_name: ollama
    restart: unless-stopped
```

```
    ports:
    - 8080:80
    volumes:
    -
/srv/docker/stacks/nextcloud/config:/var/ww
w/html/config
    -
/mnt/dexterno/nextcloud/data:/var/www/html
/data # data en disco externo
    depends_on:
    - db

db:
    image: mariadb:10.8
    container_name: nextcloud-db
    restart: unless-stopped
    command:
--transaction-isolation=READ-COMMITTED
--binlog-format=ROW
    environment:
    -
MYSQL_ROOT_PASSWORD=<contraseña
segura>
    - MYSQL_DATABASE=<nombre de la
base de datos>
    - MYSQL_USER=<usario de la base de
datos>
    - MYSQL_PASSWORD=<contraseña
segura>
    volumes:
    -
/srv/docker/stacks/nextcloud/db:/var/lib/mys
ql
```

```
    ports:
    - 11434:11434
    volumes:
    - ollama:/root/.ollama
    deploy:
    resources:
        limits:
        cpus: "2.0"
        memory: 8G
        reservations:
        cpus: "1.0"
        memory: 6G
open-webui:
    image:
ghcr.io/open-webui/open-webui:main
    container_name: open-webui
    restart: unless-stopped
    depends_on:
    - ollama
    environment:
    -
OLLAMA_API_BASE=http://ollama:11434
    ports:
    - 3002:8080
    deploy:
    resources:
        limits:
        cpus: "1.0"
        memory: 2G
        reservations:
        cpus: "0.5"
        memory: 1G
volumes:
ollama: null
```

Vaultwarden

```
services:
vaultwarden:
    image: vaultwarden/server:latest
    container_name: vaultwarden
    restart: unless-stopped
    environment:
    SIGNUPS_ALLOWED: "no"
    INVITATIONS_ALLOWED: "false"
    ADMIN_TOKEN: <MiToken>
    ROCKET_PORT: "80"
    volumes:
    - ./data:/data
    networks:
    caddy_net:
        ipv4_address: 172.28.0.2
    expose:
    - "80"
```

Uptime Kuma

```
services:
uptime-kuma:
    image: louislam/uptime-kuma:latest
    container_name: uptime-kuma
    restart: always
    volumes:
    -
/srv/docker/stacks/uptime-kuma:/app/data
    ports:
    - 3001:3001
    networks:
    - caddy_net
networks:
caddy_net:
    external: true
```

networks: caddy_net: external: true	

A.4 Caddy File

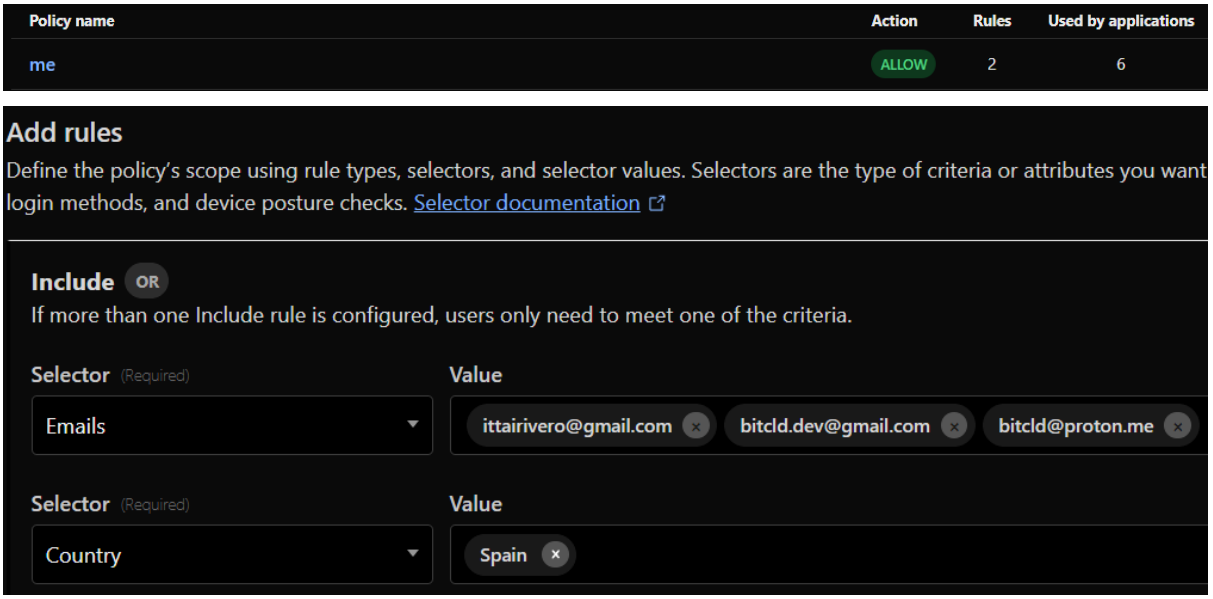
Este es nuestro CaddyFile completo.

```
# Nextcloud
cld.lan {
    tls internal
    reverse_proxy 192.168.1.5:8080
}

# AdGuard Home
dns.lan {
    tls internal
    reverse_proxy 192.168.1.5:3000
}

# Dockge
dockge.lan {
    tls internal
    reverse_proxy 192.168.1.5:5001
}

# Uptime Kuma
monitor.lan {
    tls internal
    reverse_proxy 192.168.1.5:3001
}

# Vaultwarden
pass.lan {
    reverse_proxy vaultwarden:80
    tls internal
}

# Netdata
stats.lan {
    tls internal
    reverse_proxy
192.168.1.5:19999
}

# IA Local y API
chat.lan {
    tls internal
    reverse_proxy 192.168.1.5:3003
}
```

A.5 Política de Zero Trust

En el anexo añadiremos la política que usamos en nuestras aplicaciones la cuál tiene dos reglas.

La primera es para seleccionar los emails que pueden recibir el código OTP, con esto podemos tener la certeza de que sólo nosotros podremos acceder. Y la segunda, es para una seguridad extra, ya que es una limitación regional, para que sólo se pueda acceder desde España.



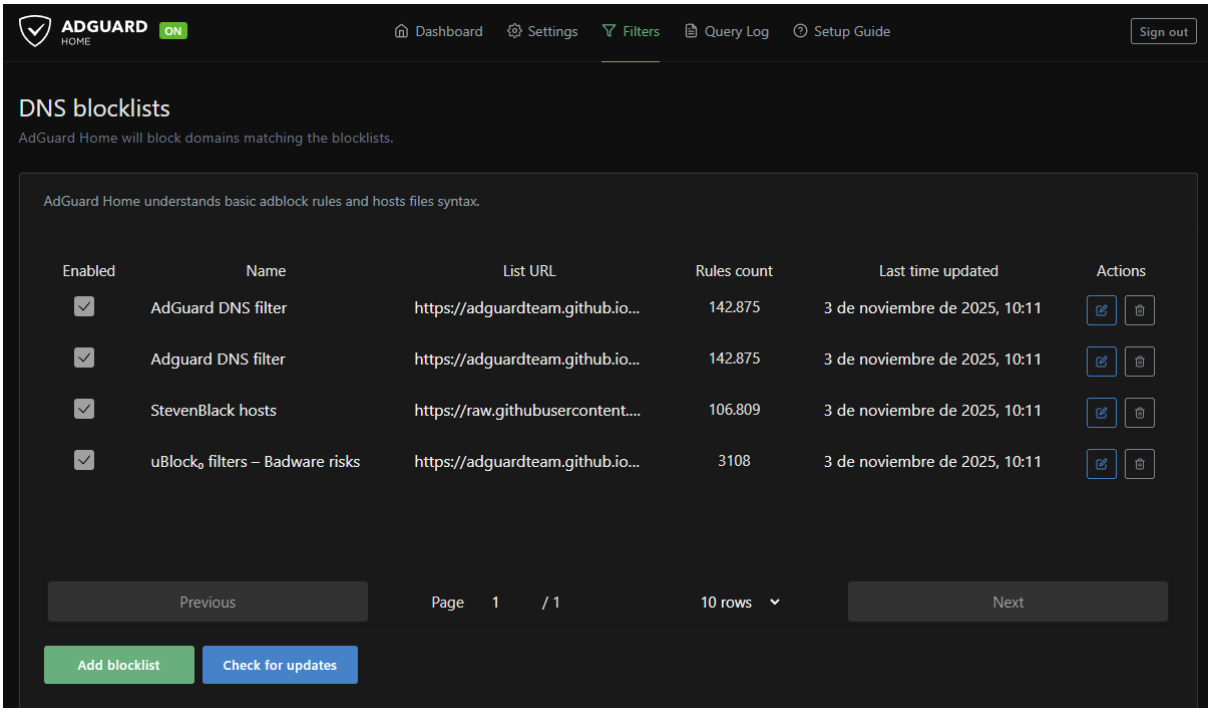
A.6 Listas de bloqueo

En nuestro servidor DNS establecimos las siguientes listas de bloqueo, las cuáles se encargan de impedir conexión con los dominios que estas contengan para evitar anuncios y rastreadores. Esta manera es mucho más efectiva ya que a diferencia de un bloqueador de anuncios como Ublock Origin (que actúa en el navegador después de recibir el contenido), nuestro servidor no llega a recibirlo por que nos ayuda a proteger toda la red, a reducir el tráfico (porque las conexiones no deseadas nunca salen) y nos aumenta la privacidad.

Las listas que seleccionamos son las siguientes:

https://adguardteam.github.io/HostlistsRegistry/assets/filter_1.txt
https://adguardteam.github.io/AdGuardSDNSFilter/Filters/filter.txt
https://raw.githubusercontent.com/StevenBlack/hosts/master/hosts
https://adguardteam.github.io/HostlistsRegistry/assets/filter_50.txt

Pero existen una gran cantidad de ellas, lo ideal es que vayamos probando y añadamos o quitemos las que no nos interesan. Debemos evitar combinar listas enormes redundantes si no necesitamos agresividad extrema ya que podría romper ciertos sitios.


ADGUARD HOME
ON
Dashboard
Settings
Filters
Query Log
Setup Guide
Sign out
DNS blocklists
AdGuard Home will block domains matching the blocklists.
AdGuard Home understands basic adblock rules and hosts files syntax.
Enabled
Name
List URL
Rules count
Last time updated
Actions
AdGuard DNS filter
https://adguardteam.github.io...
142.875
3 de noviembre de 2025, 10:11
Adguard DNS filter
https://adguardteam.github.io...
142.875
3 de noviembre de 2025, 10:11
StevenBlack hosts
https://raw.githubusercontent....
106.809
3 de noviembre de 2025, 10:11
uBlock₀ filters – Badware risks
https://adguardteam.github.io...
3108
3 de noviembre de 2025, 10:11
Previous
Page 1 / 1
10 rows
Next
Add blocklist
Check for updates