Ubuntu Server - Home Lab Project

A project by Daniel Felipe Perez Rincon.

Welcome to the Ubuntu Home Lab Project documentation! In this project, I aim to build a self-hosted server that consolidates multiple services, including file storage and media streaming, all running in Docker containers. This setup will allow me to host Nextcloud for cloud storage and Jellyfin for media streaming.

We will configure both local and remote access using Nginx for proxy host management and SSL certificates. A domain is required for this setup—you can obtain a free one from DuckDNS. Additionally, we will need a platform to manage DNS records for the domain.

The server is built on a Dell Vostro laptop, featuring an i5-10210 CPU, 8 GB of RAM, and 1 TB of storage. With these specifications, I can comfortably run multiple Docker containers while ensuring good performance for both local and remote access to my services. By repurposing an old laptop, I am able to make the most of the existing hardware and contribute to reducing e-waste, which is an important step towards sustainability.

This document will walk through the entire process, from setting up a fresh Ubuntu server to configuring the services and ensuring secure access both locally and remotely. Whether you're looking to create your own home server or simply exploring new ways to manage and organize services, this guide will provide you with all the steps and configurations needed.

INSTALLING UBUNTU

In this section, we will cover the process of installing Ubuntu Server on your hardware. Ubuntu Server is a popular, stable, and secure Linux distribution, making it an excellent choice for setting up your home server. This installation will serve as the foundation for running various services, including Nextcloud and Jellyfin, within Docker containers. The installation process is straightforward, and we will walk through the necessary steps, from preparing the installation media to configuring the server post-installation.

Creating Bootable USB

Before we can install Ubuntu Server, we need to create a bootable USB drive with the Ubuntu Server ISO. This USB drive will allow us to install the operating system on the server hardware. In this section, we will use a tool to write the Ubuntu Server image to a USB drive, preparing it for installation.

1. Download the ISO File for Ubuntu server from this [link](https://ubuntu.com/download/server#system-requirements-lts).
2. Download [rufus](https://rufus.ie/en/)
3. Create bootable USB

A screenshot of a computer

AI-generated content may be incorrect.

Installation Process

The installation process is straightforward, but if you need additional guidance, feel free to refer to this helpful [video](https://www.youtube.com/watch?v=K2m52F0S2w8). During the installation, be sure to configure your network settings, as assigning a static IP address is highly recommended for your server. This ensures a stable connection and makes it easier to manage your services later on.

**Remember to change the boot device!**

In the setup process you can select to install OpenSSH to SSH to your machine as soon as the installation is over. This will save time in the configuration.

**This is me doing the installation:**

SECURING MY SERVER

Once Ubuntu Server is installed, the next crucial step is securing it. A properly secured server helps protect your data and services from unauthorized access and potential threats. In this section, we will implement fundamental security measures, including updating the system, configuring a firewall, and disabling root login. These steps will enhance the overall security and stability of your server.

Update The System

**Update the package list:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Upgrade installed packages:**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**Remove unnecessary packages:**

**A screenshot of a computer program

AI-generated content may be incorrect.**

Set Up a Firewall

**Install ufw:**

**A screenshot of a computer program

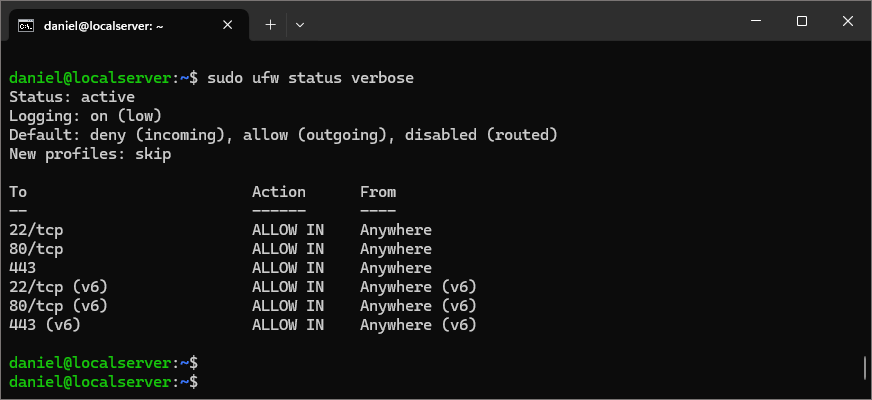
AI-generated content may be incorrect.**

**Allow SSH, HTTP AND HTTPS:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Enable the firewall and check configurations:**

****

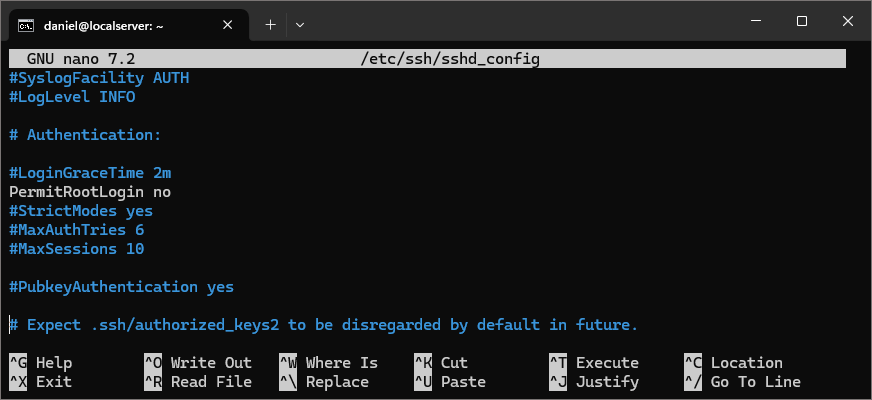
Disable Root Login

**Edit SSH config file:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Uncomment and change “PermitRootLogin” to no:**

****

**Restart SSH service**

**A screenshot of a computer

AI-generated content may be incorrect.**

DOCKER INSTALLATION

Docker is a lightweight containerization platform that allows you to run applications in isolated environments. By using Docker, we can easily deploy and manage services like Nextcloud and Jellyfin without worrying about software dependencies or system conflicts. This approach ensures a more flexible and scalable home server setup.

In this section, we will install Docker on Ubuntu Server and verify that it is running correctly.

I am going to follow the steps given by the docker webpage for the Docker engine installation. Check it out [here!](https://docs.docker.com/engine/install/ubuntu/#install-using-the-repository)

**Steps:**

1. Set up Docker’s **apt** repository:

A screenshot of a computer

AI-generated content may be incorrect.

1. Install Docker packages:

A screenshot of a computer

AI-generated content may be incorrect.

1. Verify the installation:

A screenshot of a computer

AI-generated content may be incorrect.

Docker Compose Installation

Docker Compose is a powerful tool that allows you to define and manage multi-container applications with a simple YAML file. Instead of running multiple docker run commands, Docker Compose makes it easier to deploy and manage services in a structured way. In this step, we will install Docker Compose to simplify the management of our containerized applications.

**Steps:**

1. Update the package index and install Docker compose:

A screenshot of a computer program

AI-generated content may be incorrect.

1. Verify the installation:

A screenshot of a computer

AI-generated content may be incorrect.

NEXTCLOUD CONFIGURATION

Nextcloud is an open-source self-hosted cloud storage solution that allows you to securely store, access, and share your files from anywhere. By hosting Nextcloud on your own server, you gain full control over your data while avoiding third-party cloud providers.

In this section, we will install Nextcloud using Docker Compose, making deployment and management more efficient.

**Steps:**

1. Create a directory for Nextcloud and cd to it:

A screenshot of a computer

AI-generated content may be incorrect.

1. Create a docker-compose.yml file:



1. Add the following configuration to this file:

services:

nextcloud:

image: nextcloud

container\_name: nextcloud

restart: unless-stopped

ports:

- "8080:80"

volumes:

- ./app:/var/www/html

environment:

- MYSQL\_HOST=nextcloud\_db

- MYSQL\_DATABASE=nextcloud

- MYSQL\_USER=nextcloud

- MYSQL\_PASSWORD=yourpassword

depends\_on:

- nextcloud\_db

nextcloud\_db:

image: mariadb

container\_name: nextcloud\_db

restart: unless-stopped

environment:

- MYSQL\_ROOT\_PASSWORD=yourpassword

- MYSQL\_DATABASE=nextcloud

- MYSQL\_USER=nextcloud

- MYSQL\_PASSWORD=yourpassword

volumes:

- ./db:/var/lib/mysql

network:

customnetwork:

external: true

1. Start Nextcloud instance:

A screen shot of a computer

AI-generated content may be incorrect.

1. Go into your browser and type [**http://your-server-ip:8080**](http://your-server-ip:8080)
2. Configure an Admin account and password and hit install.

Now we have installed Nextcloud on our server. We will finish the rest of the installations.

JELLYFIN CONFIGURATION

Jellyfin is a free, open-source media server that allows you to stream your personal media collection to various devices. It provides a self-hosted alternative to commercial streaming services, giving you full control over your media without subscriptions or data tracking.

In this section, we will install Jellyfin using Docker Compose, ensuring a simple and flexible deployment.

**Steps:**

1. Make a new directory and cd to it:



1. Create a docker-compose.yml file:



1. Insert this configuration:

services:

jellyfin:

image: jellyfin/jellyfin

container\_name: jellyfin

ports:

- '8096:8096/tcp' # Jellyfin web interface

- '7359:7359/udp' # Network discovery

- '1900:1900/udp' # DLNA port

volumes:

- /home/marek/Jellyfin/config:/config

- /home/marek/Jellyfin/Movies:/data/Movies

- /home/marek/Jellyfin/Shows:/data/Shows

environment:

- PUID=1000

- PGID=1000

- TZ=America/Toronto

restart: unless-stopped

network:

customnetwork:

external: true

1. Access your Jellyfin server from your browser typing [**http://your-server-ip:8096**](http://your-server-ip:8096)
2. Go through the wizard and configure your Jellyfin account.

NGINX INSTALLATION

Now that we have installed and configured our core services, Nextcloud and Jellyfin, the next step is to set up **Nginx Proxy Manager (NPM)**. This tool will allow us to manage and secure access to our services easily.

Nginx Proxy Manager acts as a **reverse proxy**, meaning it will handle incoming requests and direct them to the appropriate service running on our server

1. Make a directory and cd to it:

A black background with green and blue text

AI-generated content may be incorrect.

1. Create a docker-compose.yml file



1. Insert this configuration:

services:

app:

image: 'jc21/nginx-proxy-manager:latest'

restart: unless-stopped

ports:

# These ports are in format <host-port>:<container-port>

- '80:80' # Public HTTP Port

- '443:443' # Public HTTPS Port

- '81:81' # Admin Web Port

environment:

# Mysql/Maria connection parameters:

DB\_MYSQL\_HOST: "db"

DB\_MYSQL\_PORT: 3306

DB\_MYSQL\_USER: "npm"

DB\_MYSQL\_PASSWORD: "Candycita2005"

DB\_MYSQL\_NAME: "npm"

volumes:

- ./data:/data

- ./letsencrypt:/etc/letsencrypt

depends\_on:

- db

db:

image: 'jc21/mariadb-aria:latest'

restart: unless-stopped

environment:

MYSQL\_ROOT\_PASSWORD: 'Merlin1301'

MYSQL\_DATABASE: 'npm'

MYSQL\_USER: 'npm'

MYSQL\_PASSWORD: 'Candycita2005'

MARIADB\_AUTO\_UPGRADE: '1'

volumes:

- ./mysql:/var/lib/mysql

network:

customnetwork:

external: true

1. Access your Nginx instance from your browser typing [**http://your-server-ip:81**](http://your-server-ip:81)
2. Access using [**admin@example.com**](mailto:admin@example.com) **for email** and **changeme for password.**
3. Change your password

LOCAL ACCESS CONFIGURATION

To enable local access with a valid SSL certificate, we need to configure DNS records and set up Nginx Proxy Manager. First, we create a subdomain that points to our home server’s private IP. Then we configure a Docker network and add all of our containers to it. After this, we generate SSL certificates using a DNS challenge and apply them to our services. Finally, we create a proxy host for each service, allowing secure access through domain names.

This setup allows us to access Nextcloud, Jellyfin, and other services using domain names instead of IP addresses, eliminating security warnings and making access more convenient. The following sections will cover the DNS configuration and Nginx setup in detail.

Cloudflare Configuration (DNS Configuration)

To configure local access using domain names, we need to set up proper **DNS records**. This allows us to access our services using human-readable addresses instead of IP addresses, making management more convenient and secure.

A domain name is required for this setup. You can either purchase a domain from a registrar like Namecheap, Google Domains, or Cloudflare, or get a free dynamic domain from services like DuckDNS or No-IP.

In this section, we will configure:

* An **A record** that points a subdomain to our home server’s private IP.
* A **CNAME record** to enable wildcard subdomains, making it easier to manage multiple services under the same domain.

With this setup, we ensure seamless access to our local services while maintaining flexibility for future configurations.

**Steps:**

1. **Get a Domain**
   1. You can get your own custom domain from websites like: Namecheap, GoDaddy, Google Domains.
   2. Alternatively, get a free domain from services like DuckDNS, FreeDNS (afraid.org), or EU.org.

In my case I own a custom domain and manage my DNS records using Cloudflare.

1. **Add A record** 
   1. Go to your domain’s DNS management panel, (in my case Cloudflare)
   2. Add an A record with the name of your domain. If you are using this domain only for this project, add it to yourdomain.com. In my case, my domain is used for a website, so I will create the A record with a name of local.mydomain.com.
   3. The target should be your server’s private ip address.
   4. Disable Cloudflare Proxy (if applicable)

A screenshot of a computer

AI-generated content may be incorrect.

1. **Add CNAME Record**
   1. Create a CNAME record for the subdomains of yourdomain.com.
   2. Set the target to yourdomain.com.

A screenshot of a computer

AI-generated content may be incorrect.

1. **Create an API Token**
   1. If using Cloudflare for DNS, go to **Cloudflare Dashboard → My Profile → API Tokens**.

A screenshot of a computer

AI-generated content may be incorrect.

* 1. Click **Create Token** and choose the **Create Custom Token**.
  2. Give the token a name.
  3. Grant permissions:
     1. **Zone → DNS → Edit**
     2. **Zone → Zone → Read**

A screenshot of a computer

AI-generated content may be incorrect.

* 1. Click **Continue to Summary**, then **Create Token**.
  2. Copy the token and store it securely (you’ll need it for SSL setup).

Docker Network Configuration

With our DNS records set up, the next step is to configure the Docker network to ensure seamless communication between our services (e.g., Nextcloud, Jellyfin) and Nginx Proxy Manager. Docker networks allow containers to communicate with each other using their service names as hostnames, simplifying internal routing and ensuring isolation from external networks.

* **Create a custom Docker network** to connect our services.
* **Configure containers to use the same network**, enabling them to communicate securely.
* **Verify network connectivity** between containers.

By the end of this process, our services will be able to communicate efficiently within the same Docker network, providing a robust foundation for local access and future scalability.

**Steps:**

* 1. **Create a Custom Docker network:**

docker network create my\_custom\_network

* + - **Verify the network was created:**

docker network ls

* 1. **Connect Existing Containers to the Network**

docker network connect my\_custom\_network nextcloud-app-1

docker network connect my\_custom\_network jellyfin-app-1

docker network connect my\_custom\_network nginx-app-1

Nginx Proxy Manager Configuration

With our DNS records set up, the next step is to configure Nginx Proxy Manager (NPM) to handle local access and SSL certificates. This ensures that our services are accessible using domain names instead of IP addresses and that they have valid SSL encryption, preventing browser security warnings.

In this section, we will:

Obtain SSL Certificates for our local domain using a DNS challenge.

Create Proxy Hosts in Nginx Proxy Manager to route traffic to our services (e.g., Nextcloud, Jellyfin).

By the end of this process, we will have a secure and user-friendly setup, allowing us to access our services with trusted domain names inside our network.

**Steps:**

* 1. **Access Nginx Proxy Manager**
     + Go to <http://yourserveripaddress:81>
     + Log in to the NPM dashboard
  2. **Obtain SSL Certificate**
     + Go to the SSL certificates tab:

A screenshot of a computer

AI-generated content may be incorrect.

* + - Click **Add SSL Certificate → Let's Encrypt**.
    - Enter your domain (e.g., local.mydomain.com and \*.local.mydomain.com).

A screenshot of a computer

AI-generated content may be incorrect.

* + - Select **Use a DNS Challenge**
    - Choose **Cloudflare** as the DNS provider (or another provider based on your setup).
    - Enter your Cloudflare API token (or credentials for your DNS provider). **Example:** dns\_cloudflare\_api\_token=Yourtoken

A screenshot of a computer

AI-generated content may be incorrect.

* + - Click **Save** and wait for the certificate to be issued.
  1. **Create Proxy Hosts for Your Services**
     + Go to the Hosts tab and select Proxy Hosts.
     + We will create the host for Nginx first
     + Click Add Proxy Host and configure the following:
       - Domain Names: Enter the subdomain for your service (local.mydomain.com).
       - Forward Hostname / IP: enter 127.0.0.1 (loopback IP)
       - Forward Port: Enter the port of the service (default for Nginx is 81).

A screenshot of a computer

AI-generated content may be incorrect.

* + - * Under SSL, select the Let's Encrypt certificate you created earlier and enable Force SSL.

A screenshot of a computer

AI-generated content may be incorrect.

* + - Repeat these steps for each service (Nextcloud, Jellyfin, etc.).
      * You can use the docker container name for each service.
      * In this setup Nextcloud uses port **80** and Jellyfin port **8096.**

A screenshot of a computer

AI-generated content may be incorrect.

* 1. **Test Local Access**
     + Open a browser and try accessing your services using the domain names you configured (e.g., <https://nextcloud.local.mydomain.com>).
     + Ensure there are no SSL warnings and that everything works correctly.

REMOTE ACCESS CONFIGURATION

Setting up remote access for your home server allows you to securely connect to your services from anywhere. Instead of exposing ports directly to the internet, Cloudflare Tunnel provides a secure way to route traffic through Cloudflare’s network, reducing security risks and simplifying access management. With this setup, you can access your Nextcloud and Jellyfin servers remotely while keeping your home network safe.

Set Up Cloudflare Tunnel

Cloudflare Tunnel provides a secure and efficient way to expose your home server services to the internet without opening ports on your router. Instead of direct port forwarding, Cloudflare’s network securely routes traffic to your local services. This enhances security by hiding your home IP and mitigating potential threats.

In this section, we will set up a Cloudflare Tunnel using Docker and configure it to route traffic to your Nextcloud and Jellyfin servers.

**Steps:**

1. **Log in or sign up at cloudflare.com**
2. **Add your domain to Cloudflare:**
   1. Go to the Cloudflare Dashboard and select "Add Site". Enter your domain name and follow the instructions to change the name server addresses from the ones of your DNS provider to the addresses of the Cloudflare's name servers. Note that this process may take up to 24 hours.
3. **Deploy Cloudflare tunnel application:**
   1. From the Cloudflare dashboard go to **Zero Trust.** The first time you access you will have to go through the process of selecting a plan and paying for it. Just choose he free plan and finish the payment process.
   2. Go to **Networks -> Tunnels.** Select **create a tunnel.**
   3. Select **Cloudflared** and name your tunnel

**A screenshot of a computer

AI-generated content may be incorrect.**

* 1. Now you can select how to install Cloudflared on your server. For this setup, we will use docker compose (this method is not included in the options shown in the website).
     1. Create a directory for Cloudflare on your machine and create a **docker-compose.yaml** file.

sudo mkdir cloudflare

sudo nano docker-compose.yaml

* + 1. Paste this configuration and save the file.

etworks:

customnetwork:

external: true

services:

cloudflaretunnel:

container\_name: cloudflaretunnel

image: cloudflare/cloudflared:2025.2.1

restart: unless-stopped

environment:

- TUNNEL\_TOKEN=$TUNNEL\_TOKEN

command: tunnel --no-autoupdate run

networks:

- customnetwork

* + 1. Now use this command. export TUNNEL\_TOKEN=xxxxx where xxxx is the token that is given on the Cloudfare docker installation. It is all the text after the ‘—token’ on the command given.

**A screenshot of a computer

AI-generated content may be incorrect.**

* + 1. Start the docker container

docker compose up -d

1. Go to **Networks -> Tunnels.** Your Cloudflare tunnel should be displayed and with ‘healthy’ as its status. 😊

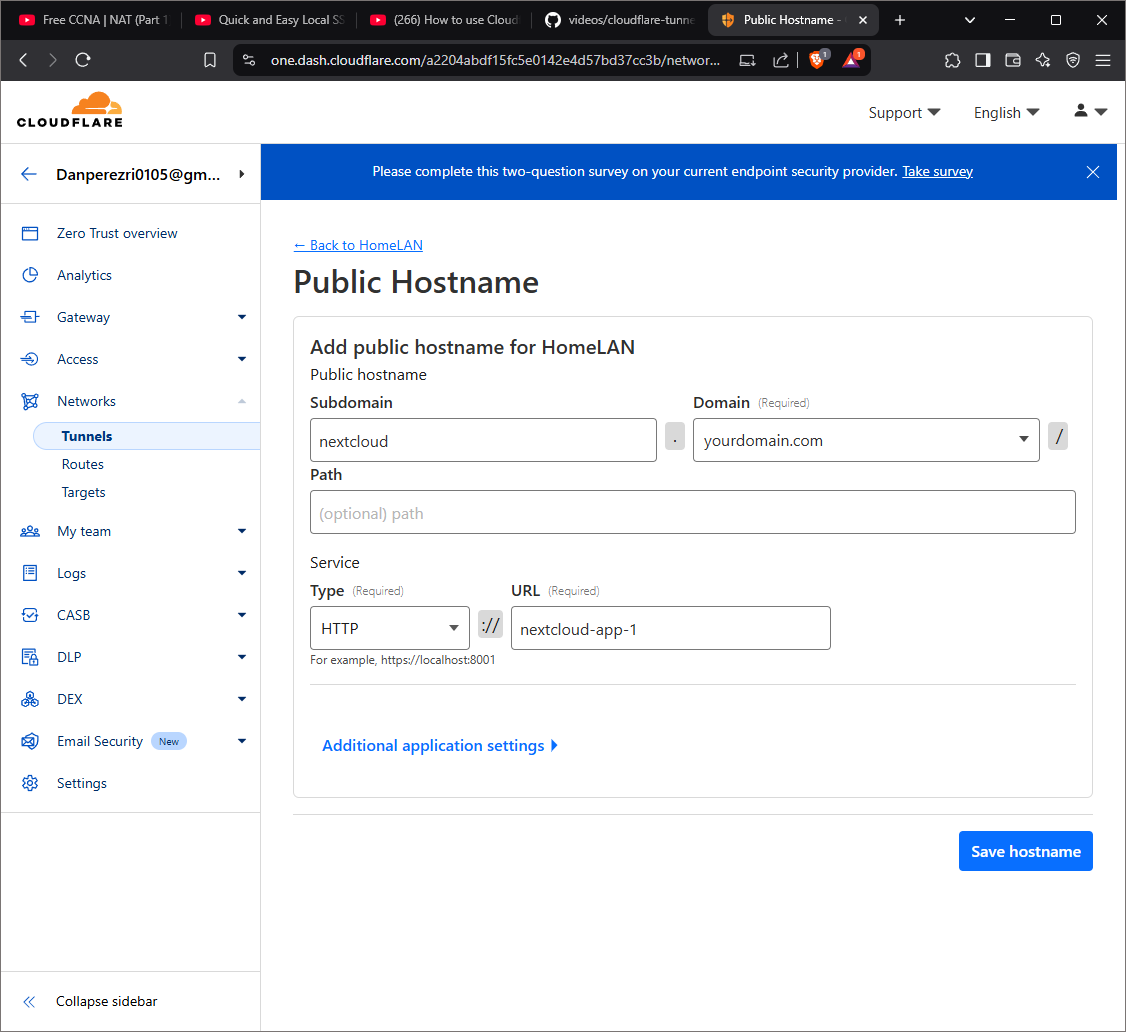
Configure Public Hostnames

After setting up a Cloudflare Tunnel, the next step is to configure public hostnames for your self-hosted services. Public hostnames act as entry points, allowing you to access your services remotely through a domain instead of an IP address. By setting up public hostnames in Cloudflare, you can map your Nextcloud, Jellyfin, or any other services to user-friendly URLs like `cloud.mydomain.com` and `jellyfin.mydomain.com`.

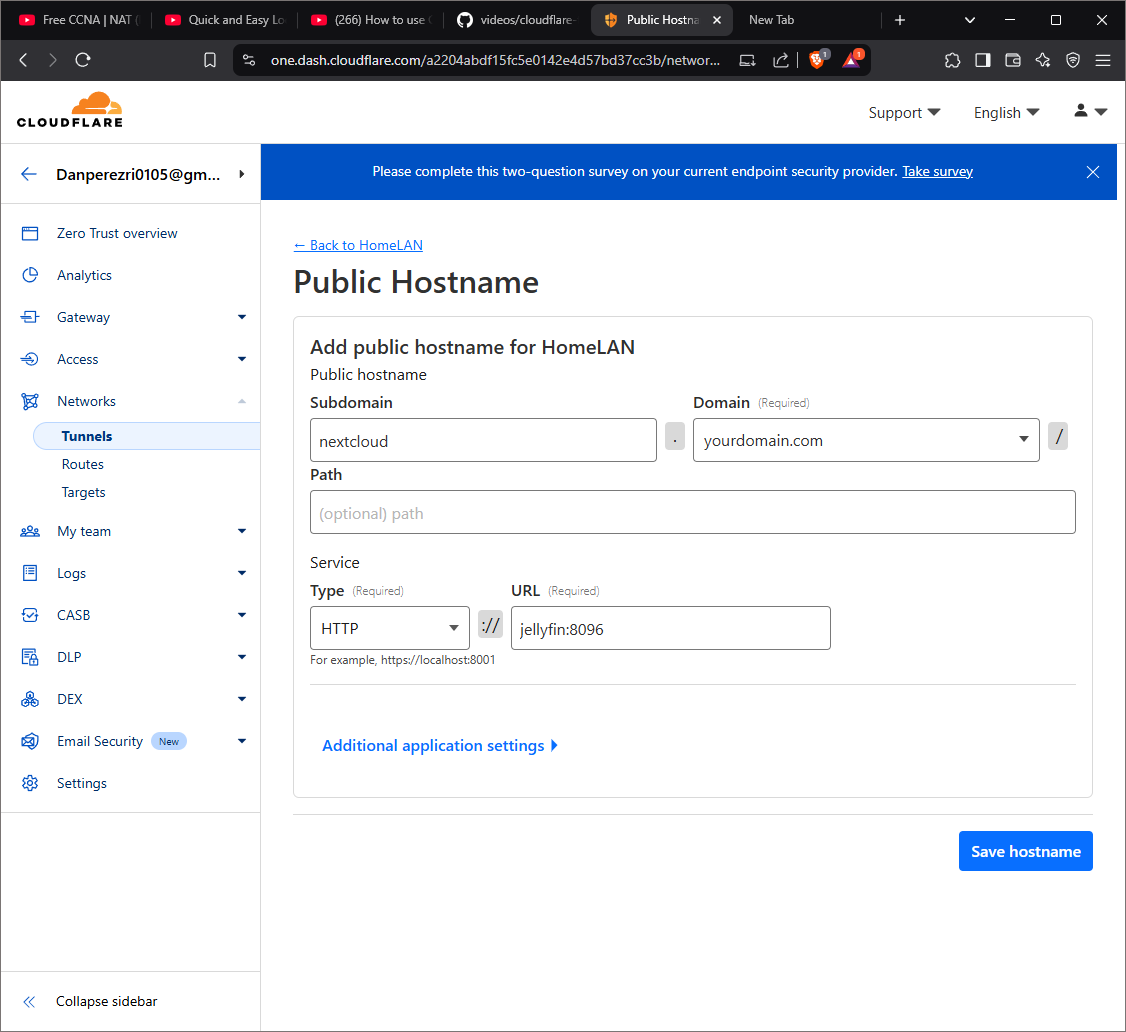
Additionally, Cloudflare provides security features such as authentication and access control, allowing you to protect sensitive services from unauthorized access. In this section, we'll go through the process of adding public hostnames and enabling authentication where needed.

**Steps:**

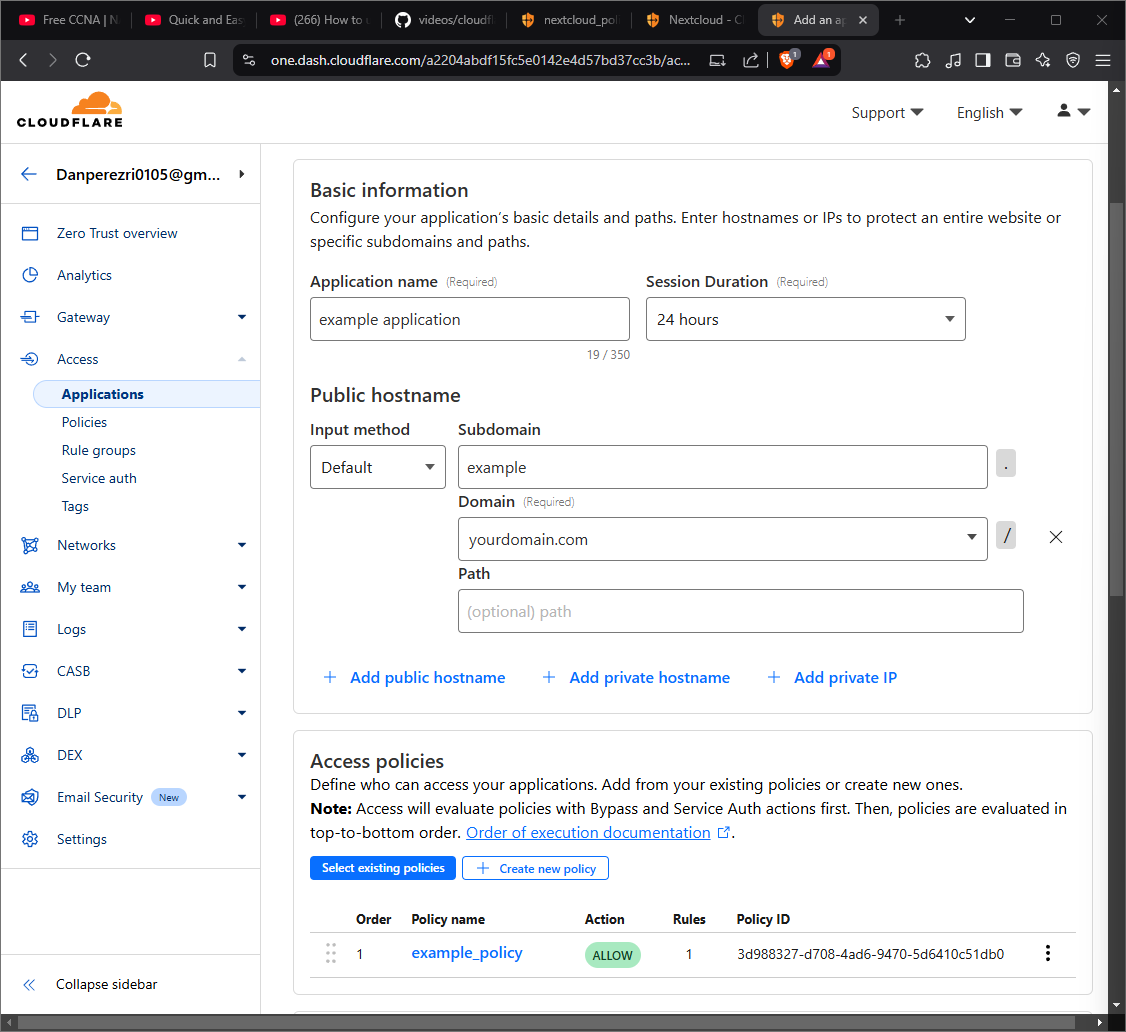
1. Add public hostnames to your tunnel.
   1. Go to **Zero trust -> Network -> Tunnels.** Select your tunnel name and click **edit.**
   2. Go to the **Public Hostname** tab. Select **Add a public hostname**.
   3. Configure the name of the subdomain, add your domain, select what type of service, and type the address for your server running the desired service.
      1. In this setup, you should be able to use the Docker container names as the URL.
      2. The configuration for Nextcloud should look something like this:



* + 1. The configuration for Jellyfin should look something like this:



1. Check that your public hostnames work. Go to your browser and type the URL of one of your services. It should be displayed without any problem.
2. Add a one-time PIN authentication for your services. You can add this if you want to improve the security of your services. This is **optional.** 
   1. Go to **settings -> authentication** and select **add new.**
   2. Select one-time PIN. Here you can select any identity provider that you wish.
   3. Go to **access -> policies** and select **add a policy.**
   4. Name your policy and add the rules you want. You can select only certain emails to be able to access or any rule you want.
   5. Click **save** to save the policy.
   6. Go to **access -> applications** and select **add an application.**
   7. Select **Self-hosted.** Give it a name and a session duration.
   8. Click on **add a public hostname.** Fill the information of the subdomain or domain that you want protected by the one-time PIN.
   9. On the **Access policies** section, click on **select existing policies**  and add the policy you created.



* 1. Click **Next**.
  2. Click **Next** for the Experience settings. You can make changes if you want.
  3. Click **Save** on the Advances settings. You can make changes if you want.
  4. Your application should now be running

1. Go to your subdomain by pasting the URL on your browser. It should prompt you to the one-time PIN page. All done

A screenshot of a computer

AI-generated content may be incorrect.

**This is The End of The Project**

**Thanks For Reading!**