**N8N Self-Host Setup**

**Part 1: AWS EC2 Setup**

We'll launch a virtual server on AWS that is eligible for the Free Tier.

1. **Log in to your AWS Management Console** and navigate to the EC2 service.
2. **Launch a new instance:**
   * Click the "Launch instances" button.
   * **Name:** Give it a recognizable name, like n8n-server.
   * **Application and OS Images (AMI):** Search for and select **Ubuntu**. Choose the latest LTS version (e.g., Ubuntu Server 22.04 LTS) and ensure it has the "Free tier eligible" label.
   * **Instance type:** Select **t2.micro** (or t3.micro depending on your region's free tier). This is crucial for staying within the free tier.
   * **Key pair (login):** This is essential for SSH access.
     + If you have a key pair, select it.
     + If not, click "Create new key pair", give it a name (e.g., n8n-key), and download the .pem file. **Store this file securely; you cannot download it again.**
   * **Network settings:**
     + Click "Edit".
     + In the "Security group" section, create a new security group.
     + Add the following **inbound rules**:
       - **Type:** SSH, **Source:** My IP (This is more secure, allowing only you to connect).
       - **Type:** HTTP, **Source:** Anywhere (0.0.0.0/0).
       - **Type:** HTTPS, **Source:** Anywhere (0.0.0.0/0).
   * **Configure storage:** The default 8 GB is fine, but you can go up to 30 GB under the Free Tier.
   * **Launch Instance:** Review the summary and click "Launch instance".
3. **Get your Public IP Address:**
   * Go back to the EC2 Instances list.
   * Select your n8n-server instance.
   * In the details pane, copy the **Public IPv4 address**. We'll need this for Cloudflare.

**Part 2: Cloudflare DNS Configuration**

Now, we'll point a subdomain (e.g., n8n.yourdomain.com) to your new EC2 instance.

1. **Log in to your Cloudflare dashboard** and select your domain.
2. Navigate to the **DNS** section.
3. **Add a new DNS record:**
   * Click "Add record".
   * **Type:** A
   * **Name:** n8n (or whatever subdomain you prefer).
   * **IPv4 address:** Paste the **Public IPv4 address** of your EC2 instance that you copied earlier.
   * **Proxy status:** Make sure the cloud icon is **orange** (Proxied). This enables Cloudflare's security and SSL features.
   * Click **Save**.
4. **Configure SSL/TLS:**
   * Navigate to the **SSL/TLS** section in the Cloudflare sidebar.
   * Set your encryption mode to **Full (Strict)**. This is the most secure option and ensures end-to-end encryption.

**Part 3: Server Preparation & Software Installation**

Let's connect to the server and install the necessary software.

1. **Connect to your EC2 instance via SSH:**
   * Open your terminal.
   * If you downloaded a .pem key, you might need to set its permissions first:

Generated bash

chmod 400 /path/to/your/n8n-key.pem

content\_copydownload

Use code [with caution](https://support.google.com/legal/answer/13505487).Bash

* + Connect using the following command, replacing the placeholders:

Generated bash

ssh -i /path/to/your/n8n-key.pem ubuntu@YOUR\_EC2\_PUBLIC\_IP

content\_copydownload

Use code [with caution](https://support.google.com/legal/answer/13505487).Bash

1. **Update System Packages:**  
   Once connected, update your server's package list and upgrade existing packages.

Generated bash

sudo apt update && sudo apt upgrade -y

content\_copydownload

Use code [with caution](https://support.google.com/legal/answer/13505487).Bash

1. **Install Docker and Docker Compose:**  
   We'll use the official Docker script for an easy and up-to-date installation.

Generated bash

# Install Docker

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

sudo sh get-docker.sh

# Add your user to the 'docker' group to run docker commands without sudo

sudo usermod -aG docker $USER

# Install Docker Compose

sudo apt install docker-compose-v2 -y

content\_copydownload

Use code [with caution](https://support.google.com/legal/answer/13505487).Bash

**IMPORTANT:** You need to log out and log back in for the group changes to take effect.

Generated bash

exit

content\_copydownload

Use code [with caution](https://support.google.com/legal/answer/13505487).Bash

Now, reconnect with the same SSH command as before.

1. **Install Nginx:**  
   This will act as our reverse proxy.

Generated bash

sudo apt install nginx -y

**Part 4: Running n8n with Docker Compose**

Using Docker Compose makes managing n8n's configuration and data simple.

1. **Create a Directory for n8n Data:**  
   This directory will store your n8n database and configuration, ensuring it persists even if the container is removed.

Generated bash

mkdir ~/n8n-data

content\_copydownload

Use code [with caution](https://support.google.com/legal/answer/13505487).Bash

1. **Create the docker-compose.yml file:**

Generated bash

nano docker-compose.yml

content\_copydownload

Use code [with caution](https://support.google.com/legal/answer/13505487).Bash

Paste the following content into the file. **Remember to replace n8n.yourdomain.com with your actual subdomain.**

Generated yaml

version: '3.7'

services:

n8n:

image: n8nio/n8n

restart: always

ports:

# Binds port 5678 on the host's localhost interface to port 5678 in the container

# This is secure because it's not exposed to the public internet directly

- "127.0.0.1:5678:5678"

environment:

# Replace with your actual subdomain

- N8N\_HOST=n8n.yourdomain.com

- N8N\_PORT=5678

- N8N\_PROTOCOL=https

- NODE\_ENV=production

# This is the URL n8n will use for webhooks

- WEBHOOK\_URL=https://n8n.yourdomain.com/

# Timezone for cron nodes

- GENERIC\_TIMEZONE=America/New\_York # Optional: Change to your timezone

volumes:

# Maps the local directory to the container's data directory for persistence

- ~/n8n-data:/home/node/.n8n

content\_copydownload

Use code [with caution](https://support.google.com/legal/answer/13505487).Yaml

Save the file and exit the editor (Ctrl+X, then Y, then Enter).

1. **Start n8n:**  
   From the same directory where your docker-compose.yml file is located, run:

Generated bash

docker compose up -d

content\_copydownload

Use code [with caution](https://support.google.com/legal/answer/13505487).Bash

The -d flag runs the container in detached mode (in the background). You can check if it's running with docker ps.

**Part 5: Configuring Nginx as a Reverse Proxy**

Nginx will handle incoming traffic from your domain and forward it securely to the n8n Docker container.

1. **Create an Nginx Configuration File for n8n:**

Generated bash

sudo nano /etc/nginx/sites-available/n8n.yourdomain.com

content\_copydownload

Use code [with caution](https://support.google.com/legal/answer/13505487).Bash

1. **Paste the following configuration.** Again, replace n8n.yourdomain.com with your subdomain.

Generated nginx

server {

listen 80;

listen [::]:80;

server\_name n8n.yourdomain.com;

location / {

proxy\_pass http://127.0.0.1:5678; # Forward traffic to the n8n container

proxy\_set\_header Connection '';

proxy\_http\_version 1.1;

proxy\_set\_header Host $host;

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

proxy\_set\_header X-Forwarded-Proto https; # Let n8n know it's behind an HTTPS proxy

proxy\_buffering off;

proxy\_cache off;

# Required for the n8n UI to work correctly

chunked\_transfer\_encoding off;

# WebSocket support for live UI updates

proxy\_set\_header Upgrade $http\_upgrade;

proxy\_set\_header Connection "Upgrade";

}

}

content\_copydownload

Use code [with caution](https://support.google.com/legal/answer/13505487).Nginx

Save and exit (Ctrl+X, Y, Enter).

1. **Enable the Nginx Site:**  
   Create a symbolic link from sites-available to sites-enabled.

Generated bash

sudo ln -s /etc/nginx/sites-available/n8n.yourdomain.com /etc/nginx/sites-enabled/

content\_copydownload

Use code [with caution](https://support.google.com/legal/answer/13505487).Bash

1. **Test and Restart Nginx:**

Generated bash

# Test for syntax errors

sudo nginx -t

# If the test is successful, restart Nginx to apply the changes

sudo systemctl restart nginx