
Hey! Here's a guide to get your own version of the Telegram tracker bot up and running. Just follow these steps.

Step 1: Fork the Code on GitHub

What you're doing: Creating a personal copy of the bot's source code in your own GitHub account. This allows you to connect it to Render.

Where to do it: In your web browser, on the GitHub website.

1. Log in to your own GitHub account.
2. Go to the project's GitHub page: <https://github.com/markhendriksen1990/Nodify.git>
3. In the top-right corner of the page, click the **Fork** button. This will create an exact copy of the project under your own GitHub account.

Step 2: Create Your Telegram Bot

What you're doing: Registering your bot with Telegram to get an API token, which is like a password for your code to control the bot.

Where to do it: In the Telegram app, by talking to a bot called **BotFather**.

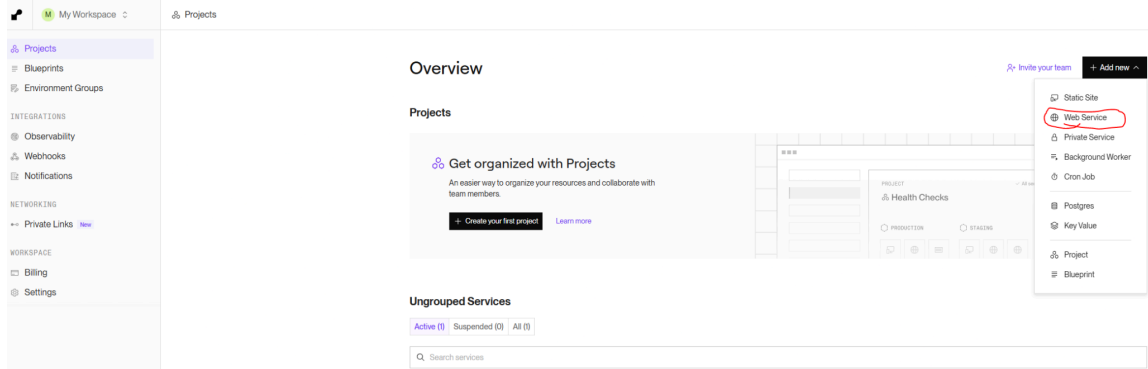
1. In Telegram, search for **BotFather** (it has a blue checkmark) and start a chat with it.
2. Send the command `/newbot`.
3. BotFather will ask for a name for your bot (e.g., "My DeFi Tracker") and then a username, which must be unique and end in **bot** (e.g., **MyDeFiTrackerBot**).
4. Once you're done, it will give you a message containing your **bot token**. It looks something like `7572395356:AAHvp1XBZzqi7Eoz3TzUDjDKui0beQAQyPE`.
5. **Copy this token and save it somewhere safe.** You'll need it in a later step. This is your **TELEGRAM_BOT_TOKEN**.
6. When a new chat c.q. bot is created you have to find it's chat ID
To get it, send `/start` to the **@userinfobot** on Telegram and copy the ID it gives you.
Copy this chat ID and save it somewhere safe. It looks something like this:
`540263811`
You'll need it in a later step. This is your **CHAT_ID**.

Step 3: Deploy the Bot on Render

What you're doing: Setting up a free web server on Render that will run your bot's code 24/7.

Where to do it: In your web browser, on the [Render dashboard](#).

1. Create a Render account. It's easiest to sign up using your GitHub account.
2. On the dashboard, click **New +** and select **Web Service**.



3. Connect the GitHub repository that you forked in Step 1.
4. Render will ask you to configure the service. Use these settings:
 - **Name:** Give your service a unique name (e.g., `nodify-yourname`).
 - **Runtime:** Render should automatically detect `Node`.
 - **Build Command:** `npm install`.
 - **Start Command:** `node main.js`.

- **Instance Type:** Select the **Free** tier.

Settings

General

Name

A unique name for your Web Service.

Nodify

 Edit

Region


Your services in the same [region](#) can communicate over a [private network](#).


Frankfurt (EU Central)

Instance Type

Free 0.1 CPU 512 MB

 Update

 Please [enter your payment information](#) to select an instance type with higher limits.

 See [remaining free usage](#), or learn about [free service limits](#).

Build & Deploy

Repository

The repository used for your Web Service.

https://github.com/markhendriksen1990/Nodify

 Edit

Branch

The Git branch to build and deploy.

main

 Edit

Git Credentials

User providing the credentials to pull the repository.

@gmail.com (you)

 Use My Credentials

Root Directory Optional

If set, Render runs commands from this directory instead of the repository root. Additionally, code changes outside of this directory do not trigger an auto-deploy. Most commonly used with a [monorepo](#).

 Edit


Build Filters

Include or ignore specific paths in your repo when determining whether to trigger an auto-deploy. Paths are relative to your repo's root directory. [Learn more](#).

 Edit


Included Paths

Changes that match these paths will trigger a new build.

 Add Included Path

Ignored Paths

Changes that match these paths will not trigger a new build.

 Add Ignored Path

Build Command

Render runs this command to build your app before each deploy.

\$ npm install

 Edit

Pre-Deploy Command Optional

Render runs this command before the start command. Useful for database migrations and static asset uploads.

\$

 Edit

Start Command

Render runs this command to start your app with each deploy.

\$ node main.js

 Edit

Auto-Deploy

By default, Render automatically deploys your service whenever you update its code or configuration. Disable to handle deploys manually. [Learn more](#).

On Commit

 Edit

Deploy Hook

Your private URL to trigger a deploy for this server. Remember to keep this a secret.

.....

 Regenerate hook

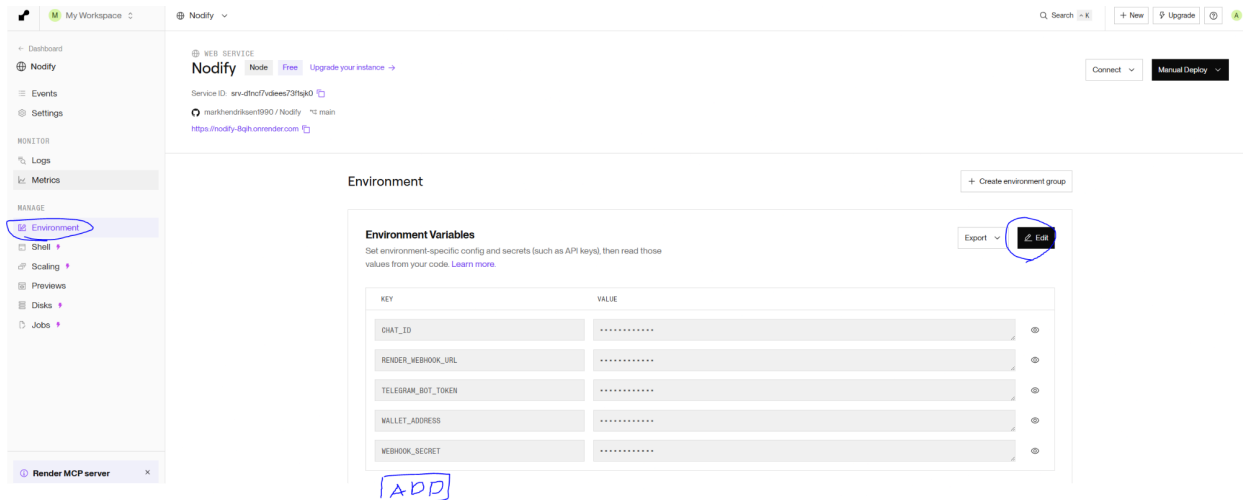
5. Before creating the service, scroll down to the **Environment Variables** section. This is where you'll put your secrets.
-

Step 4: Add Your Environment Variables

What you're doing: Giving your Render service the secret keys and configuration it needs to run.

Where to do it: In the "Environment" section during the Render setup process.

1. Click **Add Environment Variable** and add the following key-value pairs one by one:
 - **Key:** `TELEGRAM_BOT_TOKEN`
 - **Value:** The token you got from BotFather in Step 2.
 - **Key:** `CHAT_ID`
 - **Value:** See Step 2 - create telegram bot.
 - **Key:** `RENDER_WEBHOOK_URL`
 - **Value:** Your service's public URL. Render gives you this once it's created.
It will look like
`https://your-service-name.onrender.com`.
 - **Key:** `WALLET_ADDRESS`
 - **Value:** The public Ethereum wallet address you want the bot to monitor (e.g., `0xAb1234a7D312341b8bE11C439e05C5A3259aeC9A`).
You can add multiple addresses using “,”.
For instance
`0xAb...1111aeC9B, 0xAb...eC9B, 0xAb...eC9C`
 - **Key:** `WEBHOOK_SECRET`
 - **Value:** A random, secure password you create yourself. This is used to make sure the requests are coming from Telegram. Example:
`MySuperSecretPassword123!`.



2. Once all variables are added, click **Create Web Service**. Render will now build and deploy your bot. This might take a few minutes.

Step 5: Connect Telegram to Your Bot on Render

What you're doing: Telling Telegram to send all messages and commands for your bot to your new app running on Render. This is called setting the "webhook."

Where to do it: In your web browser's address bar.

Construct the following URL by replacing the parts in

<...> with your actual values from the previous steps.

`https://api.telegram.org/bot<YOUR_TELEGRAM_BOT_TOKEN>/setWebhook?url=<YOUR_RE
NDER_WEBHOOK_URL>/bot<YOUR_TELEGRAM_BOT_TOKEN>/webhook&secret_token=<
YOUR_WEBHOOK_SECRET>`

1. **Example:**

`https://api.telegram.org/bot1234123456:ABHVpiXBZsqi7Eoz5TzUDjDKwi
3beQAQyPE/setWebhook?url=https://nodify-1q1v.onrender.com/bot1234
123456:ABHVpiXBZsqi7Eoz5TzUDjDKwi3beQAQyPE/webhook&secret_token=M
ySuperSecretPassword123!`

2. Copy your fully constructed URL, paste it into your browser's address bar, and press **Enter**.
3. If it's successful, you will see a simple message in your browser like:
`{"ok":true,"result":true,"description":"Webhook was set"}`

4. In case needed:

`https://api.telegram.org/bot1234123456:ABHVpiXBZsqi7Eoz5TzUDjDKwi3beQAQyPE/getWebhookInfo`

This gives you it's status

5. `https://api.telegram.org/bot1234123456:ABHVpiXBZsqi7Eoz5TzUDjDKwi3beQAQyPE/deleteWebhook`

This deletes the telegram webhook link.

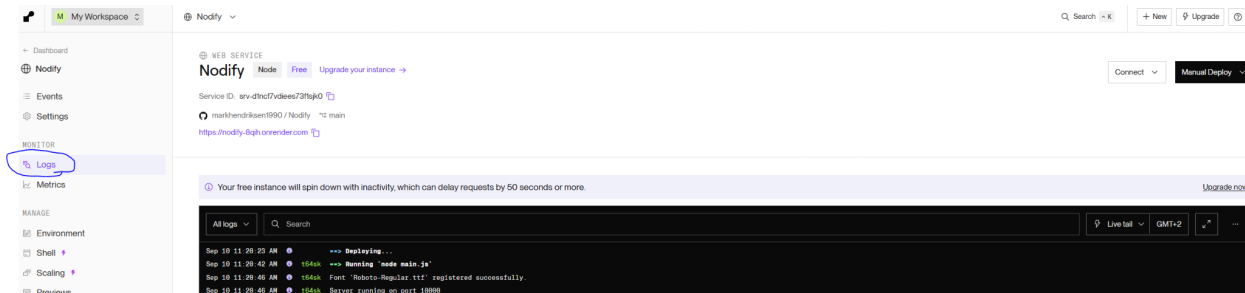
Sometimes you have to delete it and connect it after running faulty code to often.

Final Check

You're all set! 🚀 Go to your bot in Telegram and send the

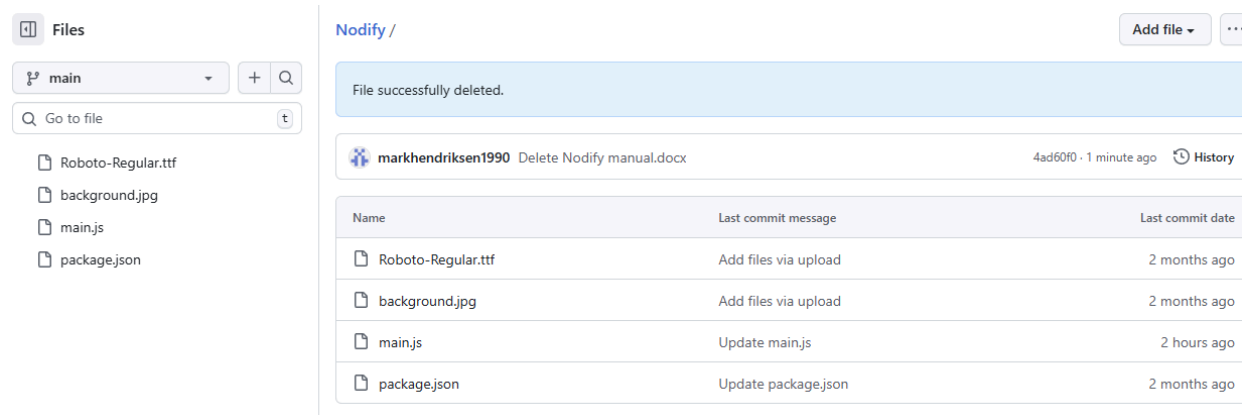
`/positions` or `/snapshot` command. It might take a moment to respond the first time as it gathers all the data.

If it doesn't work, the first place to check is the **Logs** tab for your service in the Render dashboard. It will usually tell you what went wrong.



Good luck!

General info, What is what of the github files:



The screenshot shows the GitHub interface for the 'Nodify' repository. On the left, the 'Files' sidebar lists the repository structure: 'main' (selected), 'Roboto-Regular.ttf', 'background.jpg', 'main.js', and 'package.json'. The main content area shows a commit history table for the 'main' branch. A notification at the top states 'File successfully deleted.' and a commit by 'markhendriksen1990' is shown, deleting 'Nodify manual.docx'. The commit history table lists the following files and their commit messages:

Name	Last commit message	Last commit date
Roboto-Regular.ttf	Add files via upload	2 months ago
background.jpg	Add files via upload	2 months ago
main.js	Update main.js	2 hours ago
package.json	Update package.json	2 months ago

main.js

Main code

package.json

Contains all dependencies and external libraries (these dependencies are automatically installed in Render). These are general coding rules and libraries and are not programmed by me, but used by many developers. "ethers" being the main one, enabling any connection with blockchain data.

background.jpg

Used as background for the /snapshot command in telegram. You may replace the image to your preference, as long as the name is identical.

Roboto-Regular

Lettering for /snapshot function

General info and resources:

- My personal Coingecko API is used in main.js. That's OK, but you may setup your own coingecko API-url if you wish.
It is used for finding current price of token0 and token1 of your pool. As well as the historic price of the day the position was started/minted. It uses the the day's price. This is not exact for the minute or hour.
- <https://docs.uniswap.org/contracts/v3/reference/deployments>
Contract addresses for each chain of Uniswap V3. As used in the code:
managerAddress
factoryAddress
- Crucial info for adding additional chains:
rpcUrl
managerAddress
factoryAddress

- According to AI it's possible to add uniswap v2 or v4 quite easily. However, i expect a lot of bugfixing and manual searching for correct addresses (rpcUrl, managerAddress, factoryAddress)