07/10/20

This guide was written to provide information about how to setup a ChirpStack gateway and send LoRa data received by the gateway to a server with a database. This is still in the draft stage and will be improved over time. Interest from the community will drive future changes and updates. If you have questions or comments please contact Ned Horning at nedhorning@gmail.com.

The guide is split into three sections:

- Install ChirpStack on a Raspberry Pi and configure the Application Server
- Program a Gnat LoRa node to send GNSS data to the ChirpStack gateway server
- Set up an Internet server and database to store and access data

The intent is to provide a high-level overview of the process as well as sufficient details so someone without much web server or programming experience can set up a LoRa gateway and a server to store data transmitted by LoRa nodes.

This guide is based on my work to develop a GNSS tracker for turtles and to have that location data transmitted wirelessly using a LoRaWAN configuration with LoRa node mounted on turtles and a LoRa server communicating with all of the nodes. I do all of my development using Ubuntu (18.04) and this guide is based on that work. Some of these sections will need to be modified slightly if using another operating system but those changes should not be too difficult.

Install ChirpStack on a Raspberry Pi and set up the Application Server
There are several different ways to configure a RAK7423 Raspberry Pi (RPi) gateway. The following three options are the ones I tried. Details on two of these methods follows below.

1. Download and install the desktop version of the Raspberry Pi OS (https://www.raspberrypi.org/downloads/raspberry-pi-os/) and then follow the instructions to install ChirpStack on a Debian (such as Ubuntu) system: https://www.chirpstack.io/guides/debian-ubuntu/. This option provides a familiar desktop and is helpful for people who prefer to interact via a desktop instead of a command line. With this approach you need to install firmware for the RAK2245 concentrator: https://github.com/RAKWireless/rak_common_for_gateway. There are more steps to accomplish this setup compared to the other two options but the benefit of having a desktop could be advantageous. This is currently my preferred method.
2. Download an RPi image from RAK (https://downloads.rakwireless.com/LoRa/Pilot-Gateway-Pro-RAK7243/Firmware/) that is preconfigured with Raspberry Pi OS lite (formally Raspbian lite) and ChirpStack. This is likely the easiest way to install and operating system for the RPi and ChirpStack. The downside is that Raspberry Pi OS lite does not have a desktop so all configuration and other interactions with the RPi must be done using a command line via SSH or a terminal connected to the RPi. The RAK website has a helpful online guide (https://doc.rakwireless.com/rak2245-pi-hat-edition-lorawan-gateway-concentrator-module/device-firmware-setup) to install the RPi operating system and ChirpStack software. Since the RAK image has ChirpStack preconfigured you can quickly get up and running and have a LoRa node communicating with Gateway and viewing data on the web-based graphical interface. For some reason I was not able to use HTTP PUST to send data to another server using this setup. I'm still investigating how to fix that.
3. Download and install the RPi image provided by ChirpStack: (https://www.chirpstack.io/gateway-os/install/raspberrypi/). This uses Yocto instead of the Raspberry Pi OS and as such is, for the most part, dedicated to ChirpStack and it's not easy to

add other applications. This is compatible with the RAK RPi gateway. The lack of a familiar operating system is a drawback for me but it is well suited if you just want your RPi dedicated to be a ChirpStack gateway and don't need to add other software.

Notes for installing on a desktop version of the Raspberry Pi OS (Option 1 from above):
Download Raspberry Pi OS from here: https://www.raspberrypi.org/downloads/raspberry-pi-os/. Any of the install options should work although I selected the "desktop and recommended software" option.

Use Balena Etcher to create bootable SD card. Boot the RPi with the new SD card and follow the onscreen guide to configure and update software.

When you insert the SD card and boot the RPi for the first time, follow the instructions to configure the settings. Updating software at the end of the configuration can take several minutes. When complete you will need to restart the RPi.

To enable the ability to ssh into the RPi you need to use the RPi configuration tool. Also, to enable the RakWireless LoRa concentrator you need to enable the SPI, I2C and serial interface using that same configuration tool. To enable those features, enter "sudo raspi-config" in a terminal window. Select "Interfacing Options" from the interface. Select "SSH" then select 'Yes", then "Ok". Do the same to turn on "SPI" and "I2C". For "Serial" you need to disable the login shell over serial and enable serial port hardware. Close the configuration window by selecting "Finish". When complete you should restart the RPi.

Follow the Rak GitHub instructions: https://github.com/RAKWireless/rak_common_for_gateway. Those instructions suggest using the Lite version of RPi OS but they installation also works for the desktop version. In summary, the instructions to install the gateway software from the Rak Wireless GitHub repository are:

- git clone https://github.com/RAKWireless/rak_common_for_gateway.git ~/rak_common_for_gateway   [Note that if git isn't installed you will need to install it using: sudo apt update; sudo apt install git -y]
- cd ~/rak_common_for_gateway
- sudo ./install.sh [You will need to select the gateway model. In my case I used "3".]

After installation is done you can configure the LoRa gateway concentrator using "sudo gateway-config". You can follow the RakWireless instructions for Cconfiguring the Gateway": https://doc.rakwireless.com/rak2245-pi-hat-edition-lorawan-gateway-concentrator-module/configuring-the-gateway. When configuring the Gateway under the second option, "Setup RAK Gateway LoRa concentrator" select "2 Server is ChirpStack". For channel plan select "ChirsStack Channel-plan configuration" and if you are in the US the channel plan should be US_902_928. For other countries you'll need to check for the channel plan that is designated (legal) for your region. The "SERVIR_IP" can be set to the default "127.0.0.1" (IP address for localhost which is the address of your own server). Now your RPi should be set up to connect to the ChirpStack Application Server which you can use to configure ChirpStack to communicate with your LoRa nodes. See the "Connect to the ChirpStack Application Gateway and configure it to listen to your LoRa nodes" section below to start running your LoRaWAN.

Once that is set up you can connect to the ChirpStack Application Gateway.

Notes for using the RAK RPi image (Option 2 from above):

The RAK website has a helpful online guide (https://doc.rakwireless.com/rak2245-pi-hat-edition-lorawan-gateway-concentrator-module/device-firmware-setup) to install the RPi operating system and ChirpStack software. Follow that guide but note that when booting the RPi for the first time with the new image you need to access the RPi gateway. The online RAK instructions provide information about how to do that using another computer. If you have your RPi connected to a terminal and keyboard you can skip that “Accessing your Gateway” page of the instructions since you are connecting directly to the RPi.

Once the WiFi settings are completed under “Accessing the Internet” you can restart the RPi and then ssh into the RPi as long as your computer is on the WiFi router with the SSID that you specified in the configuration. To do that you’ll need to find the RPi IP address. You can do that by logging into your router or looking for a Raspberry Pi device on your network using a mobile app such as Network Scanner. If you have a terminal connected to the RPi you can type “ifconfig” in a terminal window and look for the “inet” number under WLAN0 or something similar. For example for me to SSH into my RPi I type ssh pi@10.0.0.50 and then enter my RPi password. Sometimes it can take a while for the wireless IP address to be established so it might take a few minutes before the RPi can be seen on the network.

When configuring the Gateway under the second option, “Setup RAK Gateway LoRa concentrator” select “2 Server is ChirpStack”. For channel plan select ChirsStack Channel-plan configuration” and if you are in the US the channel plan should be US_902_928. For other countries you’ll need to check for the channel plan that is designated (legal) for your region. The “SERVIR_IP” can be set to the default “127.0.0.1” (IP address for localhost which is the address of your own server). Now your RPi should be set up to connect to the ChirpStack Application Server to configure ChirpStack to listen to yur LoRa nodes. See the “Connect to the ChirpStack Application Gateway and configure it to listen to your LoRa nodes” section below to start running your LoRaWAN.

Once that is set up you can connect to the ChirpStack Application Gateway.

Notes for using the ChirpStack Gateway OS image Option 3 from above):
Follow the directions on the ChirpStack Gateway OS web page (https://www.chirpstack.io/gateway-os/overview/) for the raspberry Pi. If you want to install all of the software on the Raspberry Pi you should use the “full” version. This is the file I downloaded: chirpstack-gateway-os-full-raspberrypi3-20200408112223.rootfs.wic.gz. After you uncompress the file you can use balenaEtcher to burn the image to an SD card.

When you boot the RPi for the first time (see the guide for options to log in) you can run “sudo gateway-config” to set up the LoRa concentrator:

Select your shield. I used “RAK2245”
Select channel plan. I used “US915”
Select the channel-block. I used “Channels 8 – 15 + 65”
Respond “Yes” and “OK” to the messages to appear

Once that is set up you can connect to the ChirpStack Application Gateway.

Connect to the ChirpStack Application Gateway and configure it to receive data from your LoRa nodes
Follow the last page of the RAK website guide (https://doc.rakwireless.com/rak2245-pi-hat-edition-lorawan-gateway-concentrator-module/connect-the-lora-gateway-with-chirpstack) to connect to the

gateway ChirpStack Application server. In my case I open an Internet browser (I used Chrome) and then type in the URL: http://10.0.0.50:8080 which opens a login page for the Application Server. The user name is “admin” and the password is also “admin”. This will open the ChirpStack Application Server web interface. You can use this to configure the gateway to receive data from your LoRa nodes. In the settings below I use the settings for my gateway. I noted cases where you need to enter data unique to your setup.

If you used the Rak Wireless image much of the configuration has already been filled in so you can skip down to the *Service-profiles* step.

Network server

General
Network server name = nedRakGateway1 *[Choose a name relevant to your project]*
Network-server server = 127.0.0.1:8000
Gateway Discovery
“Enable gateway discovery” not checked
TLS Certificates
Do not add anything on this page

Gateway-profiles
You can leave this section blank since it’s not used any more

Organizations

Organization name = chirpstack
Display name = ChirpStack
Gateways = check “Organization can have gateways

All users

Username = admin
Permissions check both “Is active” and “Is global admin”

Service-profiles
Even if most of the service profile is filled in you need to make sure you check the “Add gateway meta-data” box.

Service-profile name = rakServiceProfile *[Choose a name relevant to your project]*
Network-server = nedRakGateway1 *[Should match the name you entered above]*
Check “Add gateway meta-data”
Do not check “Enable network geolocation
use “0” for all three fields at the bottom

Add a new device profile that works with OTAA and Cayenne
In my case, for the turtle tracker, I need to add a new profile that uses otaa and Cayenne LPP. To do that, click on “Device-profiles” then click on “+ create” to create a new profile. Enter the following settings. After making the changes click on “CREATE DEVICE PROFILE”.
General

Device-profile name = otaa_cayenne
Network-server = ns
LoRwWAN MAC version = 1.0.2
LoRaWAN Regional Parameters revision = A

Last three fiels are “0”
Join (OTAA/ABP) tab
Device supports OTAA is checked
CLASS-B tab
Device support Class-B not checked
CLASS-C tab
Device support Class-B not checked
CODEC tab
Payload codec = Cayenne LPP

Configure the gateway
The gateway should be configured already if you used the Rak Wireless image. Otherwise you can follow these settings to configure the gateway:

Gateway details
Gateway name = ned_rak1_gateway *[Choose a name relevant to your project]*
Description = Ned’s first RAK gateway *[Choose a description relevant to your project]*
Gateway ID = b827ebfffeade849 *[Choose ID for your gateway]*
Network server = nedRakGateway1 *[Should match the name you entered above]*
“Gateway discovery enabled” not checked

This is optional but you can follow these instructions to set the gateway location.

Click on the Gateways link of the left then click on the “GATEWAY CONFIGURATION” tab near the top of the page. If you want you can enter the “Gateway altitude” in the appropriate text box. Next, use the map to move the icon to your location. To do that you need to first zoom out by clicking on the “-” button several times until you see you location on the map then move the icon to roughly the correct location. To more accurately place the icon zoom in and move the marker again. Repeat this zoom and move step until you are satisfied with the location then click “UPDATE GATEWAY”.

Add a device
To add a device (Gnat in my case) go to “Applications” then click on the name of an application – the default is “app”. Click on “+ CREATE” and enter the following information. Below is what I used for my device.

Configuration
Device name = neds_gnat2 *[Choose a name relevant to your project]*
Device description = Neds second Gnat *[Choose a description relevant to your project]*
Device EUI = 383434306a379301 *[Choose EUI for your LoRa device]*
Device-profile = otaa_cayenne *[Should match the name you entered above]*
Disable frame-counter validation not checked

After clicking Create Device you need to enter the Application key
Keys (OTAA)
Application key = 9C5E64056A46C2872DED008E20A6F95A *[Choose or generate an application key that you use for your application]*
Gen Application key – leave blank
Click on “SET DEVICE-KEYS”

Viewing data from a LoRa node

To see if the Gateway is communicating with the LoRa node you just configured click on "Applications" then clcik on the application that you use to create a device ("app" is the pre-configured application"). Click on the link for the device you just created then click on the "DEVICE DATA" link near the top of the page. Any incoming data packets should be displayed. If you see the data you expect the gateway is working.

Set up the Application Server to PUSH data to an endpoint (another server)
To set up the Application Server to use HTTP PUSH to forward data to another server (see section below "Set up an Internet server and database to store and access data") you need to configure an Integration. To do that click on "Applications" in the ChirpStack web browser interface then click on the "INTEGRATIONS" tab near the top of the interface. Next, clcik on the "HTTP" link to open the form. The only line you need to fill in is "Uplink data URL". This URL specifies where the LoRa packet will be sent. When testing with a server on a local computer you need to specify the IP address of the computer with the MongoDB database and server. Since data is being sent from the RPi it is important that they are on the same network so they can communicate. For example, if you install MongoDB and the server on the computer that you use to SSH into the RPi then you could use the IP address of that computer. You also need to add the port (3000 as defined in the Node.js code) and the "data" directory. For example, I put the following text on the "Uplink data URL" line: "http://10.0.0.61:3000/data". If you are connecting to the Heroku server you would need that URL. For example for Heroku: https://turtle-server.herokuapp.com/data. You can get that address from Heroku by logging into your account and via the terminal as described below in the "Deploy code to Heroku server" section.

Program a Gnat LoRa node to send GPS data to the ChirpStack Server
The Gnat LoRa/GNSS asset tracker is programmed using Arduino. The Arduino IDE can be downloaded from the Arduino web site: https://www.arduino.cc/. On that website you will also find information about using Arduino. Software and information required to configure the Gnat for use within the Arduino environment can be found here: https://github.com/GrumpyOldPizza/ArduinoCore-stm32l0. Note that the Gnat requires the purchase and installation of a LoRa antenna, GNSS antenna before powering with a battery or USB.

You can download a sketch from from Gnat GitHub repoository (https://github.com/kriswiner/CMWX1ZZABZ/tree/master/Gnat) or from the TerrapinTracker GitHub repository (https://github.com/nedhorning/TerrapinTracker) in the Sketches directory.

Once the Arduino environment is set for the Gnat by specifying the correct board (Gnat-Lo82CZ) and connecting the board to the computer the sketch can up uploaded to the Gnat. If the sketch has print statements (sketch without "NoPrints" in the filename) then you can open a serial monitor from the Arduino IDE to view messages.

If the ChirpStack RPi gateway is running you can verify the Gnat LoRa node and RPi gateway are communicating.

Set up an Internet server and database to store and access data
This example of setting up an Internet server and database uses no-cost versions of Heroku and MongoDB services. This can be suitable for testing but for long term deployments it might be necessary to upgrade the server to another plan.

The ChirpStack Application Server will make POST requests to the configured endpoints when a node join is complete and when data is received from a node. Although there are other options for sending data using the HTTP POST request is probably the most straightforward. The server side code is written using Node.js.

The following steps will be outlined in detail below:

- Download server software from GitHub and setup local test server
- Set up a Heroku server account
- Set up a GitHub account
- Set up a MongoDB server account and configure MongoDB
- Deploy code to Heroku server
- Configure an endpoint integration in the ChirpStack Application Server

Download server software from GitHub and setup local test platform

I use Visual Studio Code (https://code.visualstudio.com/) as my coding environment for Node.js. There are other options for Node.js and any code editor should work fine although any examples in this guide will be based on Visual Studio Code.

Install Node.js and the Node Package Manager (npm)

You will need to download Node.js and the node package manager (npm). Since npm is distributed with Node.js you only need to install the software for your operating system from this site: https://nodejs.org/en/download/. I used Node.js LTS version 12.2.0. On Ubuntu I used the following commands in the terminal instead of downloading the software from the Node.js website:

```
sudo apt-get update
sudo apt-get install nodejs-dev node-gyp libssl1.0-dev
sudo apt-get install npm
```

Download GitHub repo

The code we will be using can be downloaded or cloned from thie GitHub repository: https://github.com/nedhorning/terrapinServer. Once you have the repository you can open it in your code editor. For Visual Studio Code you go to File => Add Foler to Workspace… then select the directory from the GitHub repository “terrapinServer-master”.

Add config/dev.env file

Before running the code locally you need to create a directory named “config” and then add a file named “dev.env” to hold environmental variables for running the server locally. The dev.env file should have the following lines:

```
PORT=3000
MONGODB_URL=mongodb://127.0.0.1:27017/tracker-data
```

Run “npm install” from the “terrapinServer-master” directory to install all of the required packages. I run all of my commands using the Terminal in Visual Studio Code (View => Terminal).

Install MongoDB locally

Next we will install MongoDB locally for testing. You can download the MongoDB Community Server (the free version) from the MongoDB website: https://www.mongodb.com/. You can download the most current version and select the correct operating system. On my system I expanded the .tgz file and then renamed the directory name from “mongodb-linux-x86_64-ubuntu1804-4.2.7” to “mongodb” and then moved it to my home directory. I also created an empty directory “mongodb-data” also inside my home directory. To start the database server run this command in the terminal:

```
    directoryPath/mongodb/bin/mongod --dbpath=directoryPath/mongodb-data
```

For example, this is the command I use:

```
    /home/ned/mongodb/bin/mongod --dbpath=/home/ned/mongodb-data
```

After running this command you should see a lot of text scroll by and when that stops the command line will not appear since the MongoDB server is continuing to run. Use Ctrl-c to stop the MongoDB server.

Install MongoDB Compass (A GUI for MongoDB) to view MongoDB data
Go to the MongoDB Compass download page: https://docs.mongodb.com/compass/master/install/ and follow the installation instructions.

When you start MongoDB Compass for the first time you will need to configure it. You can select any of the privacy setting that you like the click on “Start Using Compass”. Make sure MongoDB is running (see previous step) then at the “New Connection” window click on the “Fill in connection fields individually” link. Use the defaults and click “Connect”.

The next window allows you to create a database but that’s not necessary since the database will be created the first time the local server is run. Once the database is created you can view data by clicking on the connection you just created from the list of connections on the left side of the MongoDB Compass interface.

Starting the local server to receive LoRa node packets and forward to MongoDB
To start the server to receive LoRa node packets you need to enter this command from the directory that was downloaded from GitHub (“terrapinServer-master”): npm run dev

If you started the MongoDB database server in the Visual Studio Code terminal you will need to open a second terminal using the “+” icon at the top-right portion of the terminal window.

You should see a message “Server is up on port 3000” in the terminal. This means the local server is up and running. If you have the ChirpStack software set up on the RPi and have the Application Server set up properly (see “Set up the Application Server to PUSH data to another server” above) the data coming from the LoRo node should be logged in the MongoDB database. When this first runs the database “tracker-data” and collection “trackerdatas” will be created and it will start to be populated with LoRa packet data.

Set up a Heroku server account
Once the local setup is working as expected you can work on setting up a database and server system that can be accessed by anyone. I am using Heroku but there are many options out there. First you will need to set up an account with Heroku. No credit card information is necessary and the nice thing about Heroku is that if you need more server resources you can upgrade your account.

Go to the Heroku site (https://heroku.com) and click on the “Sign up” button. Enter your personal information then click “CREATE FREE ACCOUNT”. You will receive and email and once you click on the link in the email your account should be active.

Next you should install the Heroku's command line tools (CLI). Go to https://devcenter.heroku.com/articles/heroku-cli and follow the "Download and install" instructions. For Ubuntu I entered "sudo snap install --classic heroku" in a terminal. You can verify that Heroku was installed properly by typing "heroku -v" in a terminal. That should provide version information.

Type "heroku login" in a terminal. Open a browser by typing any key except "q" . Next you can login using the email and password you specified when you created the Heroku account. After you are logged in you can close the tab on the browser. You should see a message in the terminal that you are logged in.

If you don't already have a GitHub account you need to set one up. To set one up go to https://github.com and follow the instructions to create a free account.

Set up a GitHub account and creating a Git repository

GitHub is a repository service that uses Git which is a version control system. If you haven't used Git before it would be good to follow a tutorial to learn about some of the basics of how to use it. We will create a local Git repository and then commit that to GitHub

To start using Git for version control you need to navigate in a terminal to the root project directory which is "terrapinServer-master" in our case. Run the command "git init" to initialize the Git repository. Next enter the following commands (you can look at a tutorial to see what each command does):

```
git add .
git status
git commit -m "Init commit"
```

If you get an error "Please tell me who you are" then run the two git config command noted in the message:

```
git config --global user.email "you@example.com"
git config --global user.name "Your Name"
```

Once the local Git repository is ready we will need to set up a GitHub repository so the code can be accessed by Heroku. If you don't already have a GitHub account you need to set one up. To set one up go to https://github.com and follow the instructions to create a free account.

Once you log into your GitHub account you will need to create a new repository. Log into GitHub and then create a new repository by clicking on the "+" icon in the upper right corner of the GitHub webpage. You need to give is a name and you can decide if you want the repository to be public or private. You can have GitHub create a Readme file which you can edit to provide more information about your project. When the form is filled in click the "Create repository" button.

The next step is to push the local Git repository to GitHub. To do that you need to copy the repository address and you can get that by clocking on the "Clone or download" button. If the window that appears includes "Clone with SSH" then click on "Use HTTPS" to open the "Clone with HTTPS" window. Copy the address in the text box. In my case the address is:
https://github.com/nedhorning/test4Guide.git

Next, use the following two commands inserting the repository address you copied :

```
git remote add origin addressFromAbove
git push -f origin master
```

Set up a MongoDB Atlas server account

Before you can deploy the software to Heroku (or another server) you need to set up a MongoDB database that Heroku can access. For this example I am using MongoDB Atlas which has a free optoin for lot volume use.

To set up a MongoDB Atlas account go to the Atlas website (https://www.mongodb.com/cloud/atlas) and click “Start free” and fill in the form to create an account. For the cloud provider you can select the Cloud Provider. I selected “AWS”. Then, you need to select a region. Make sure you select one that has “Free Tier Available” . I selected ”N. Virginia”. All other settings can be left at their default values. Click the “Create Cluster” button and after some time (a few minutes) the cluster will be set up.

To connect to a cluster you need to click on the “CONNECT” button on the “Clusters” page. That opens the “Connect to Cluster0 page. Click on the “Add a Different IP Address” button and then enter 0.0.0.0/0 into the “IP Address text box to whitelist all IPs. Then, click the “Add IP Address” button.

Next you need to create a MongoDB User. Enter a username and password. The last step is to connect to a database. Make sure MongoDB Compass is running then click on the “Choose a connection method” button and then click on “Connect with MongDB Compass”. Click on “I have MongoDB Compass” and check the version selection is ok. Then copy the host address portion of the connection string from the link displayed in the text box. That includes everything between the “@” sign and “/admin”. In my case that address is: “cluster0-rvcbb.mongodb.net”.

In MongoDB Compass click on “New Connection” and then “Fill in connection fields individually” to open the connection form. Paste the host address you just copied to the “Hostname” text box. Leave the port set to the default value of “27017” and click on the “SRV Record to move the toggle to the right. For “Authentication” select “Username / Password” and then type in the username and password you entered when setting up the cluster (not your Atlas username and credentials. The other field(s) can be left as the default. Click “Connect”. There won’t be any data since none has been written to the database.

Deploy code to Heroku server

to deploy the software to Heroku we will use the CLI tool installed above. Open Visual Studio Code and then in the terminal type: heroku create *uniqueNameForApp*. In my case I used “heroku create horning-test-app”.

We will need to create a MONGODB_URL environmental variable on Heroku for the using the Heroku “config” command using a key:value pair. First we need to get the MongoDB Atlas and create another connection. Log into your MongoDB Atlas account then go to the page to create a new connection using the “CONNECT” button. In the “Connect to Cluster0 window click on “Connect your application”. The “DRIVER” should be set to “Node.js” and the “VERSION” set to “3.6 or later”. Copy the text string in the text box for step “2”. You will need to edit that string to replace “<password>” with the password to the cluster that you created above so it might be easiest to past the text string into a text editor. You also need to replace “<dbname>” with a name for the database. If you want to use the same name as above that would be “tracker-data”.

The syntax is: heroku config:set *key value*. This is the command that needs to be run in the terminal:
heroku config:set MONGODB_URL=*yourMongodbConnectionString*

In my case the command is:
heroku config:set MONGODB_URL='mongodb+srv://nedhorning:mypassword@cluster0-rvsbb.mongodb.net/tracker-data?retryWrites=true&w=majority'

Note, if you have special characters in the connection string URLyou will need to add quotes at the beginning and end of the connection string. For Mac or Linux use single quotes and on Windows use double-quotes.

To verify that the Heroku config worked type "heroku config" and verify the output is correct.

To push the code up to the Heroku server from your GitHub repository type the following in the terminal (it will take a while to run):
git push heroku master

This is the same command you would run if you update your code.

The final step is to set the RPi Gateway so it sends data to the Heroku server. To do that we will edit the Application Server Integration as we did above in the section "Set up the Application Server to PUSH data to an endpoint (another server)". The only change is that you need to set the "Uplink data URL" line to the Heroku URL. The Heroku URL is https://*yourAppName*.herokuapp.com/data. Another way to get the URL is to type "heroku info" in a terminal and you will see the "Web URL" listed there.
In my case the URL is: "https://horning-test-app.herokuapp.com/data"