LoRaWAN network setup guide

Contents

[Introduction 2](#_Toc97649092)

[Guide 2](#_Toc97649093)

[Prerequisites 2](#_Toc97649094)

[Configuration 2](#_Toc97649095)

[File tree 3](#_Toc97649096)

[docker-compose.yml 3](#_Toc97649097)

[PostgreSQL 4](#_Toc97649098)

[Chirpstack application server 4](#_Toc97649099)

[Chirpstack gateway bridge 5](#_Toc97649100)

[Chirpstack network server 6](#_Toc97649101)

[Nginx 6](#_Toc97649102)

[Running the stack 6](#_Toc97649103)

[Download and deploy 6](#_Toc97649104)

[Add Network server 6](#_Toc97649105)

[Add service profile 7](#_Toc97649106)

[Add a gateway 8](#_Toc97649107)

[Add device profile 9](#_Toc97649108)

[Appendix 10](#_Toc97649109)

[A. Install Docker, Docker Compose, Git, Curl 10](#_Toc97649110)

[Git & curl 10](#_Toc97649111)

[Docker 10](#_Toc97649112)

[Docker Compose 12](#_Toc97649113)

[B. Example application 12](#_Toc97649114)

# Introduction

This guide will help you in deploying a ChirpStack server stack using Docker and Docker Compose.

It will take you step-by-step through everything needed in order for getting a ChirpStack instance configured and running. It is intended for use with a debian based linux distro.

Ubuntu 20.04 have been used when writing this guide.

# Guide

For installation of Docker, Docker Compose, Git, and Curl see [Appendix A](#_Install_Docker,_Docker)

## Prerequisites

A linux-based VM/computer/server with Docker and Docker Compose installed (should work on all debian based systems, tested on ubuntu 20.04).

## Configuration

In a Terminal emulator, cd into the directory where you want the docker-compose solution to be stored (recommended: /home/<user\_name>) then write:

git clone <http://thtlab.apator.com/andras/chirpstack-docker>

### File tree

In the following sections, each component of the chirpstack-docker solution will be explained. The solution consists of a file tree:

chirpstack-docker/

├─ configuration/

│ ├─ chirpstack-application-server/

│ │ ├─ **chirpstack-application-server.toml**

│ ├─ chirpstack-gateway-bridge/

│ │ ├─ **chirpstack-gateway-bridge.toml**

│ ├─ chirpstack-network-server/

│ │ ├─ **chirpstack-network-server.toml**

│ ├─ eclipse-mosquitto/

│ │ ├─ **mosquitto.conf**

│ ├─ nginx/

│ │ ├─ certs/

│ │ │ ├─ **chirp.crt**

│ │ │ ├─ **chirp.key**

│ │ ├─ conf/

│ │ │ ├─ **default.conf**

│ │ │ ├─ **proxy\_params**

│ │ ├─ Dockerfile

│ ├─ php-apache/

│ │ ├─ sites-available/

│ │ │ ├─ **000-default.conf**

│ ├─ postgresql/

│ │ ├─ initdb/

│ │ │ ├─ **001-init-chirpstack\_ns.sh**

│ │ │ ├─ **002-init-chirpstack\_as.sh**

│ │ │ ├─ **003-init-chirpstack\_as\_events.sh**

│ │ │ ├─ **004-chirpstack\_as\_extensions.sh**

│ ├─ scripts/

│ │ ├─ post\_deployment\_script.py

### docker-compose.yml

The docker-compose file is where every container is declared, environment variables are set and ports are exposed. **Note:** When first cloning the repository, the ports have been mapped to not conflict with eachother and should not be altered except if the mapping is conflicting with other services running on the host.

In Docker Compose port mappings are declared as:

HOST\_PORT:CONTAINER\_PORT

The container port is used internally on the docker virtual network and the host port is used if the service should be accessible from the outside.

In the table below, the default port mapping used by the project is shown:

|  |  |  |  |
| --- | --- | --- | --- |
| Container name | HOST\_PORT | CONTAINER\_PORT | Comment |
| Chirpstack-network-server | Non-accessible | 80 |  |
| Chirpstack-application-server | Non-accessible | 8081 | The access to the application server happens through the nginx proxy. |
| Chirpstack-gateway-bridge | 1700 | 1700 |  |
| Postgresql | 5432 | 5432 |  |
| Nginx | 80, 443 | 80, 443 |  |
| Adminer | Non-accessible | 8080 |  |
| Mosquitto | 1883 | 1883 |  |

Table Default port mapping

### PostgreSQL

As can be seen in the file tree, configuration/postgresql/initdb contains 4 scripts. These scripts are being run the first time the solution is deployed. They serve to create the 3 databases that are needed for chirpstack.

In 001-init-chirpstack\_ns.sh, the password ‘VERYSECUREPASSWORD123’ should be changed to something secure (it is the same as the database name and role per default):

psql -v ON\_ERROR\_STOP=1 --username "$POSTGRES\_USER" <<-EOSQL

    create role chirpstack\_ns with login password 'VERYSECUREPASSWORD123';

    create database chirpstack\_ns with owner chirpstack\_ns;

EOSQL

This should be repeated in 002-init-chirpstack\_as.sh and 003-init-chirpstack\_as\_events.sh, i.e.:

psql -v ON\_ERROR\_STOP=1 --username "$POSTGRES\_USER" <<-EOSQL

    create role chirpstack\_as with login password 'VERYSECUREPASSWORD123';

    create database chirpstack\_as with owner chirpstack\_as;

EOSQL

And

psql -v ON\_ERROR\_STOP=1 --username "$POSTGRES\_USER" <<-EOSQL

    create role chirpstack\_as\_events with login password 'VERYSECUREPASSWORD123';

    create database chirpstack\_as\_events with owner chirpstack\_as\_events;

EOSQL

These role names, database names and passwords are used in the configuration files in the next three sections.

### Chirpstack application server

In configuration/chirpstack-application-server/chirpstack-application-server.toml, a few changes needs to be made.

#### Database logins

For connecting – and logging in to the databases created in the init scripts, the following Data Source Name (DSN) strings should be modified to reflect the passwords made in the init scripts:

Under the [postgresql] tag:

dsn="postgres://chirpstack\_as:VERYSECUREPASSWORD123@postgresql/

chirpstack\_as?sslmode=disable"

And under the [application\_server.integration.postgresql] tag:

dsn="postgres://chirpstack\_as\_events:VERYSECUREPASSWORD123@postgresql/

chirpstack\_as\_events?sslmode=disable"

#### Port mapping (Optional)

As stated in the [docker-compose.yml section](#_docker-compose.yml), the ports should only be remapped if the ports used in this project are conflicting with existing ports.

### Chirpstack gateway bridge

In configuration/chirpstack-gateway-bridge.toml, the gateway backend is defined.

#### Port mapping

Taken from the comments in the file:

*# Gateway backend configuration.*

[*backend*]

*# Backend type.*

type="semtech\_udp"

*# Semtech UDP packet-forwarder backend.*

  [*backend*.*semtech\_udp*]

*# ip:port to bind the UDP listener to*

*#*

*# Example: 0.0.0.0:1700 to listen on port 1700 for all network interfaces.*

*# This is the listener to which the packet-forwarder forwards its data*

*# so make sure the 'serv\_port\_up' and 'serv\_port\_down' from your*

*# packet-forwarder matches this port.*

  udp\_bind = "0.0.0.0:1700"

If port 1700, which is also exposed to the host computer and therefore the network, is conflicting with other services running on the host, change the HOST\_PORT in docker-compose.yml, i.e., here:

  chirpstack-gateway-bridge:

    image: chirpstack/chirpstack-gateway-bridge:3.13.2

    ports:

      - **<HOST\_PORT>**:1700/udp

This port is used to configure the gateway backend as the destination port. For more information about how to set up a gateway using a Raspberry Pi and a Semtech Packet Forwarder. See the Raspberry Pi + Picocell Gateway guide.

### Chirpstack network server

In the configuration/chirpstack-network-server.toml file, one change needs to be made.

#### Database login

In the configuration/chirpstack-network-server.toml file, change this line:

dsn="postgres://chirpstack\_ns:VERYSECUREPASSWORD123@postgresql/chirpstack\_ns?sslmode=disable"

to match the password set in 001-init-chirpstack\_ns.sh.

Everything else should be left as is.

### Nginx

In the configuration/nginx/certs/ folder, a .crt and .key file should be placed. These can just be self-signed SSL certificates or proper CA certificates.

Everything else should be left as is.

## Running the stack

At this point, the required software has been installed and the changes needed to the configuration has been made, which means we can now install and deploy the ChirpStack.

### Download and deploy

Now, everything is ready to be deployed.

Write the following:

cd chirpstack-docker

sudo docker-compose up

This deploys the whole stack and you can verify that it is working by going to the hostname of the application server in a web browser.

### Add Network server

In the web UI add the chirpstack-network-server as a Network Server, like so:

Graphical user interface, text, application, email

Description automatically generated

Figure Adding a network server.

The network server in the stack is located at chirpstack-network-server:8000 unless this port has been altered during configuration.

### Add service profile

A service profile is needed for ChirpStack to work. Add one like so:

Graphical user interface, text, application, email

Description automatically generated

Figure Adding a service profile.

### Add a gateway

Add atleast one gateway to ChirpStack. The ID is the unique ID used by the gateway. If using a Picocell gateway, this ID is stored in global\_conf.json in gateway\_conf.gateway\_ID.

Graphical user interface, text, email

Description automatically generated

Figure Adding a gateway.

### Add device profile

A device profile is needed when adding new devices to applications.

Default configuration:

* Name: default-otaa
* Network server: default
* LoRaWAN MAC version: 1.0.4
* ADR algorithm: Default ADR algorithm
* Max EIRP: 14
* Uplink interval: 3600

Graphical user interface, text, application, email

Description automatically generated

Figure Adding a device profile.

Make sure to enable Over-The-Air-Activation (OTAA) in the JOIN(OTAA / ABP) tab.

These steps could be performed using the API aswell. The documentation of this can be found at <FIX\_THIS\_URL\_WHEN\_MIILIB\_IS\_PROPERLY\_DOXED>. An example of how to use the API can be found in [Appendix B](#_Example_application).

# Appendix

## Install Docker, Docker Compose, Git, Curl

### Git & curl

Run the following:

sudo apt-get update

sudo apt-get install git curl

### Docker

Run the following:

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

This is getting a “convenience script” from Docker’s website, that installs Docker Engine and Docker Engine CLI on the Pi.

Then run the script:

sudo sh get-docker.sh

Verify that Docker has been installed correctly by running the hello-world container:

sudo docker run hello-world

It should look like this:

Text

Description automatically generated

### Docker Compose

Install Docker Compose using Python’s package manager Pip.

Install python3 (python3 is probably already installed with the linux distro) and pip:

sudo apt-get install python3 python3-pip

Install Docker Compose:

sudo pip install docker-compose

## Example application