

FHIR Proxy Installation

03 – Quickstart Install

Version 1 – February 2021

Table of Contents

[1 Introduction 4](#_Toc65486429)

[1.1 Purpose of this Document 4](#_Toc65486430)

[2 The Quickstart Design 5](#_Toc65486431)

[2.1 Overview 5](#_Toc65486432)

[2.2 Database (Postgres) 5](#_Toc65486433)

[2.3 FHIR Proxy 5](#_Toc65486434)

[2.4 Web Proxy (Traefik) 6](#_Toc65486435)

[2.5 Keys and Certificates 6](#_Toc65486436)

[3 Configuration Files 7](#_Toc65486437)

[3.1 Download Configuration 7](#_Toc65486438)

[3.2 Tour of the Configuration 7](#_Toc65486439)

[3.2.1 Docker Compose Configuration 7](#_Toc65486440)

[3.2.2 FHIR Proxy Configuration 8](#_Toc65486441)

[3.2.3 Database initialisation script 9](#_Toc65486442)

[3.2.4 Postgres (database) Configuration 9](#_Toc65486443)

[3.2.5 Traefik (Web Proxy) Configuration 9](#_Toc65486444)

[3.2.6 Example Certificates 9](#_Toc65486445)

[4 Quickstart Installation 11](#_Toc65486446)

[5 Smoke Test 13](#_Toc65486447)

[6 Appendix A – Generating a Self-Signed Certificate 16](#_Toc65486448)

**Version Control**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Release Date** | **Released By** | **Reason for Release** |
| 1 | 18/02/2021 | Tim Davey | Preliminary Draft |
|  |  |  |  |

**Reviewers**

|  |  |  |  |
| --- | --- | --- | --- |
| **Initials** | **Name** | **Role** | **Organisation** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# Introduction

## Purpose of this Document

This document is part of a set which walks through the entire process of installing the FHIR Proxy and connecting to YHCR. It is assumed that the preceding document(s) have already been read, and material already covered will not be repeated.



This document covers setting up a “Quickstart” installation – in other words a full working prototype environment on a single box. This could be used as a developer / demo environment, or as a rehearsal prior to the main install. Important points to note are:

* ***It is assumed that you have already worked through paper “02 – Server Preparation” to install prerequisites on the server***
* ***It is assumed that you will go on to study the remaining papers before deploying to production – ie the steps in this paper do not lead to a production-ready deployment.***
* ***This “quickstart” install is only available for Linux Containers – ie on Linux or with Docker Desktop (Windows 10 or Mac).*** 
  + Specifically this quickstart is not available for Windows Server. This is because there are not Windows Container versions available of all the 3rd party components.
  + Even if you plan to install on Windows Server eventually it is highly recommended, if possible, to get hold of a temporary Linux or Desktop environment and work through the quickstart exercise.
  + Alternatively you could try to adapt the instructions in “05 – Production Topology” and manually install all of the components on a single Windows Server box. (This is certainly possible although more effort).
  + Failing this you should read this document anyway, as it explains many concepts and principles which it will be important to understand.

The good news is that, having completed the Server Preparation steps, then docker makes the actual install very quick and easy. Most of this document therefore focuses on explaining what is being installed and how it works and is configured. The actual installation itself is at the end and very brief.

# The Quickstart Design

## Overview

The Quickstart solution is provided as a Docker Compose file. This spins up the entire infrastructure with a single command. However before looking at the technical configuration, this section explains the design we are trying to build. The diagram below provides an overview:



## Database (Postgres)

Postgres is bundled as an open source database. See <https://www.postgresql.org/>. It is used to store FHIR Resources and the Audit trail.

* The database listens on port 5432 on the “internal” network.
* The pgadmin client is also included as a database administration tool. See <https://www.pgadmin.org/>. It is exposed on the “default” network on port 54321.

## FHIR Proxy

This is the core YHCR software. In this quickstart configuration then two FHIR Proxies are configured and connected to the database:

* One is “open” and is exposed to the “default” network on port 3000. This means you can simply call it with no security hurdles to overcome – thus providing instant gratification for a demo! In a more realistic environment this approach could still be relevant for your own internal services to connect to. For example, your own internal integration engine loading FHIR Resources.
* The other instance is “secured” and is not exposed on the “default” network. It is available only on the “internal” network, on port 3001. It has message signature validation configured - to apply security checking to the message content. It is only exposed on the “default” network via further layers of SSL transport security, as covered under the Web Proxy.

*Note: It is anticipated that a future version of the FHIR Proxy could enable multiple endpoint configurations to be enabled from a single service. This might allow these two instances to be combined.*

## Web Proxy (Traefik)

Traefik is bundled as an open source web proxy. See <https://traefik.io/traefik/>. It works well here as part of a Docker Compose demo, but it is performing only standard web proxy tasks and could easily be replaced by another choice (eg IIS / NGINX) for your actual deployment.

The web proxy provides SSL and TLS Mutual Authentication in front of the “secure” FHIR Proxy. It routes incoming requests arriving on port 443 and path “/fhir” to the backend FHIR server on port 3001.

## Keys and Certificates

The quickstart comes configured with a set of self-signed keys. As such they are in no way secure, but they help to demonstrate the concepts and configuration. It is easy to get confused in the detail of configuring keys, and so important to start out with a clear conception of what each key-pair is for:

* ***Server (SSL) keys (pink)*** – these are BOTH installed on the web proxy. It automatically sends out the public certificate (crt) to any requesting client – so that they can verify the identity of the server and then use this public key it to encrypt their traffic. It retains the private key (key) to decrypt this incoming traffic.
* ***TLS Mutual Authentication keys (yellow)*** – the public cert (crt) is installed on the web proxy and used to verify incoming connections. Approved clients are issued with a private key (key) which they keep secret and use to prove their identity
* ***Signing Keys (green)*** - the public cert (crt) is installed on the FHIR proxy and used to verify incoming messages. Approved clients are issued with a private key (key) which they keep secret and use to sign the message tokens used in their requests.

This is a very basic setup – notably there are no Root Certificates in this example. However it will be enough for us to demonstrate the main concepts for this quickstart.

# Configuration Files

## Download Configuration

***Copy the configuration files onto your chosen server.***

<<TODO – confirm where to get them from. There might also be another repo for developers who want the code?>>

* If you have git installed then use the command: git clone <https://github.com/synanetics/synfhir-store.git>
* Alternatively, open in a browser to <https://github.com/synanetics/synfhir-store>, look for the green “Code” button, select “Download ZIP”, and unzip the files

The location of the install on the server is your choice

Once you have a copy of the files downloaded then look inside the “quickstart” folder

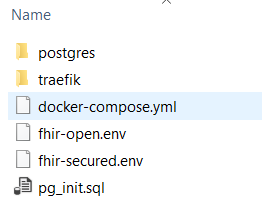
***The first thing to notice is that the installation is very small and consists of only a few text-based configuration files. Docker will automatically download the latest version of the actual container images the first time it is run.***

## Tour of the Configuration

It is worth taking some time to examine each configuration file to understand what it does.

***It is strongly recommended to take some time and understand the configuration.***

***However everything is configured with defaults and ready-to-go – so it is possible to proceed directly to the next section and run up the system if you wish!***



### Docker Compose Configuration

* ***docker-compose.yml*** – this is the main Docker Compose configuration file which describes the whole environment. ***Please take some time to study this file, and make sure you understand each section and what it does***. Even without a deep knowledge of Docker Compose it should be fairly easy to match up the configuration in the file with the diagram in section 2.1. If necessary refer to the Docker Compose documentation at <https://docs.docker.com/compose/compose-file/compose-file-v3/> or other online tutorials.

A couple of points that may need further explanation:

* + You will see some “***labels***” starting with “traefik:” – these define the configuration of the Traefik Web Proxy and are part of the integrated way that Traefik interoperates with docker. It is not hard to guess what these labels do, but unless you are planning to use Traefik yourself (eg for your production deployment) then understanding the details is not vital.
  + You will see some “***volumes***” entries – these are mostly fairly obvious and mount configuration files so that they are visible inside the docker environment. A separate volume is also created to persist the database files. This ensures that they persist even when the docker environment is spun down. (Otherwise the database would be wiped. If you actually want to wipe the database then ***docker-compose down -v*** destroys all volumes to achieve this.).

### FHIR Proxy Configuration

* ***fhir-open.env*** and ***fhir-secured.env*** – configuration files for each of the FHIR Proxy instances. There are many settings in this file, but most of them are defaults which will rarely need adjusting. Important settings to understand include:
  + ***PG\_CONNECTION*** – the Postgres database connection string in format ***postgresql://[USER]:[PASSWORD]@[DB\_HOST]:[DB\_HOST\_PORT]/fhirstore***
    - The pre-created user for this quickstart exercise is “**iamonfhir**”, and the standard port for Postgres is **5432**. When running inside docker then the host name is simply the name of the docker container ie “**localdb**”
    - This is all pre-set ready for the quickstart – however ***if you decide to change the database password in db.env then remember to also change it here***!
  + ***LOGLEVEL*** – either warn, info, or debug. For the quickstart it is set to “info” which seems appropriate for the exercise, however you may wish to experiment and adjust
  + ***TRACE\_CONSOLE*** – this has been set to false for the quickstart, however it is possible to enable additional logging by setting to true.
  + ***FHIR\_STORE\_BASE\_URL*** and ***FHIR\_STORE\_ASYNC\_URL*** – when the FHIR Resources are returned then – as with any RESTful API - they may contain URLs which cross-reference other FHIR Resources. The first part of these URLs needs to contain the externally referenceable location of the server – which is something the FHIR Proxy needs to be told via these settings. (This is not important for a quick demo, but significant for more serious testing where it is desired to follow FHIR links)
  + ***NODE\_ENV*** and ***TAG*** – settings about your environment and organisation, see notes in the config file itself. Not important for the quickstart demo
  + ***JWT validation*** - The one difference between the two FHIR Proxy configurations is the settings for JWT token validation. Only for the “secured” connection then VALIDATE\_JWT set to true and the other settings are populated as documented in the configuration file. Essentially this is where the “Signing Certificate” is pasted (green in the diagram). Note the slightly quirky format, which requires /n linebreaks to be inserted after every line.

### Database initialisation script

* ***pg\_init.sql*** – this contains a SQL script to initialise the database tables. For the quickstart solution then you don’t need to do anything with this - it will be run automatically when the solution is first started.

There are then two subfolders which contain pre-configured supporting components for the quickstart:

### Postgres (database) Configuration

* ***db.env*** – this contains the setting “POSTGRES\_PASSWORD”, which configures the database password for the “iamonfhir” user which the FHIR Proxy uses to connect. A default is set, but feel free to change it.

***If you change the password here, then you must also change the password in the database connection string in the FHIR Proxy configuration file(s)***

* ***dbclient.env*** – this contains the credentials to log on to the postgress pgadmin “front end” client application. Again feel free to change these defaults if you wish.

### Traefik (Web Proxy) Configuration

One of the features of Traefik (which makes it good for this quickstart) is that it can automatically read configuration details from docker. Therefore most of the configuration is actually inside the docker-compose.yml file. However additional settings for TLS do need to be configured:

* + ***dynamic\_conf.yaml*** – this defines default certificates for both SSL and client Mutual Authentication.
  + ***certs*** folder – contains the certificates used by the settings in dynamic\_conf.yaml. This includes:
    - A self-signed key-pair to enable SSL (The “Server (SSL) keys” – pink in the diagram).
    - The TLS mutual authentication certificate (yellow in the diagram)

These keys and certificates are provided ready-to-go, but you may replace them with you own key-pair(s) if you wish. Once you have everything working then it would be a good additional exercise to try this – see Appendix A for details.

Finally it is worth knowing where to find all of the example certificates

### Example Certificates

* ***misc\test-certs*** – this folder is separate from the rest of the “quickstart” files, as it is not directly used of needed for the solution to run. It contains the full set of example (self-signed) key pairs used by the demo:
  + ***ServerSSL.crt*** and ***ServerSSL.key*** are already loaded into the traefik configuration folders to enable SSL
  + ***TLSMA.crt*** is also loaded into the traefik configuration folders to enable TLS Mutual Authentication. However ***TLSMA.key*** is also provided in this examples folder – as you will need it for the client to present when making a connection
  + ***JwtSigning.crt*** is already embedded inside ***fhir-secured.env***. However ***JwtSigning.key*** is also provided in this examples folder, as you will need it later to generate JWT tokens

# Quickstart Installation

Check that you have completed the necessary previous steps:

* Server prepared (see “***YHCR FHIR Proxy Install 02 - Server Preparation***”)
* Configuration files downloaded and inspected (see Section 3 above)

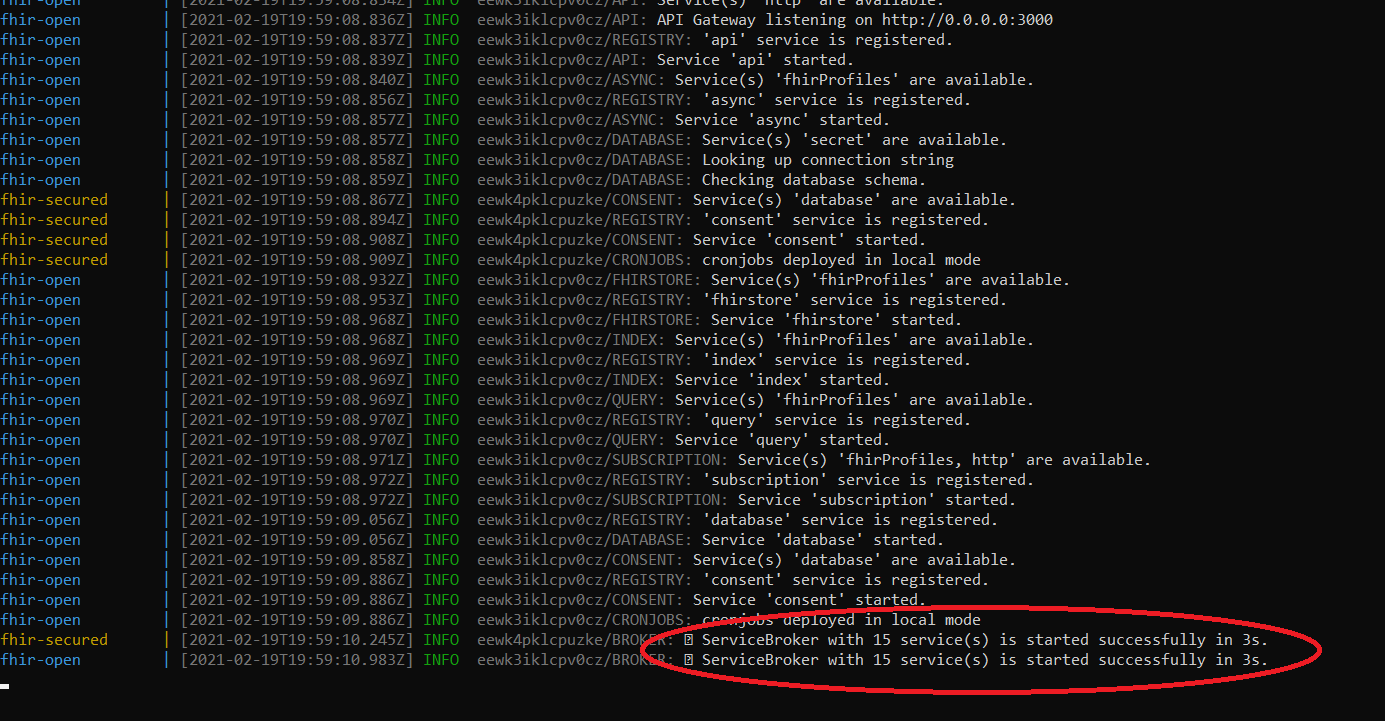
In which case the benefit of docker is that the actual install itself is very simple:

***NB: On Linux you may need to prefix these commands with sudo***

1. Ensure your current directory is the “quickstart” directory where the “docker-compose.yml” file is located
2. Enter the command to spin up the installation: ***docker-compose up***

The first time you run this it will take some time, and you will see it downloading the images. There will also be some delay on startup whilst the database is initialised. (You may see some errors and the other components “spinning around” with retries whist they wait for the database to become available). Subsequent runs will be much faster.

Once successful then you will see logging output similar to the below:



The exact order of events may vary slightly, but near the bottom you should see messages from both “fhir-secured” and “fhir-open” to say that the ServiceBroker has started successfully.

It is worth taking some time to study these logs and get a feel for what they are saying. The coloured labels on the left identify activity from the different components, and it is worth a careful check that there are no errors highlighted. (Notwithstanding the above notes about a delay due to database initialisation on the first run). Any other errors should be investigated before proceeding. One of the more likely errors is a database connection failure due to passwords mismatch – in which case check through the configuration files again.

The system is now running in the command window – which is good for testing as you can easily see the logging output.

1. When done press ***ctrl-C*** to exit.

Alternatively you can kick it off in “detached” mode to run in the background using: ***docker-compose up -d***

* You can then check it is running with: ***docker ps***
* And view the log output with ***docker logs <name>***

When you have finished then you can spin down the infrastructure and tidy up with

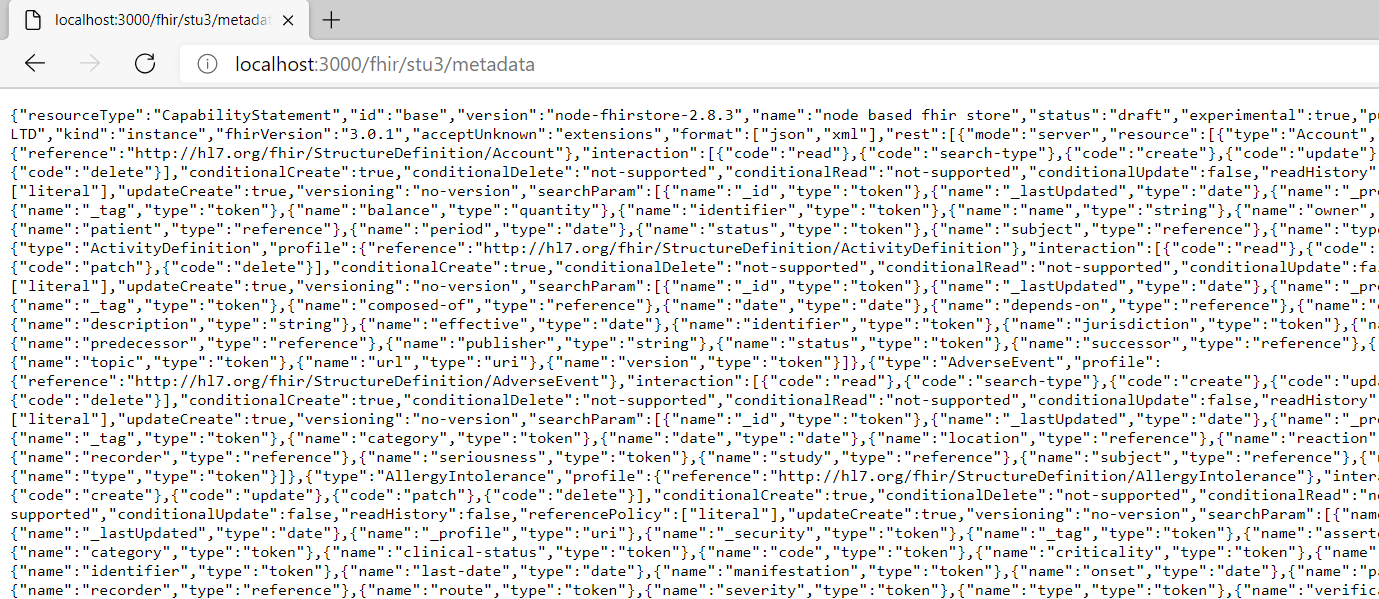
1. ***docker-compose down***

# Smoke Test

If all appears to be working then it is good to do a few simple tests to confirm basic functionality:

1. Spin up the infrastructure again if necessary: ***docker-compose up***
2. ***In a browser navigate to:*** ***<http://localhost:3000/fhir/stu3/metadata>***. (Or alternatively ***curl*** [***http://localhost:3000/fhir/stu3/metadata***](http://localhost:3000/fhir/stu3/metadata))

This is the big test! You should see the server’s metadata statement, similar to the screenshot below:

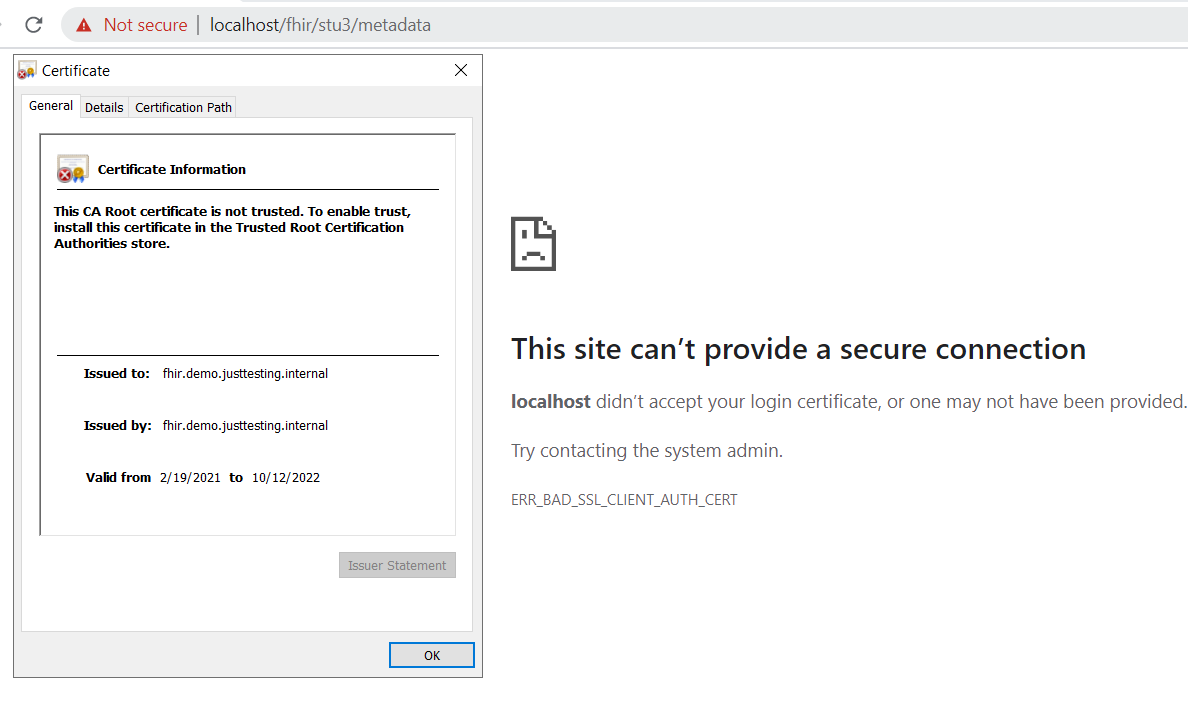


***Troubleshooting tip****: one thing to try if it doesn’t work is replacing “localhost” with the actual server IP address. Historically there have sometimes been problems with docker resolving local loopback routing).*

You can also try just [***http://localhost:300***](http://localhost:300)***0*** which should return simply “Ping”. This can also be useful for healthchecks.

1. ***In a browser navigate to:*** [***https://localhost/fhir/stu3/metadata***](https://localhost/fhir/stu3/metadata).

The previous test was going via the “open” route in. You can also try the “secured” route – at this stage this is obviously not expected to work, and you will see the TLS Mutual Authentication rejecting the connection:

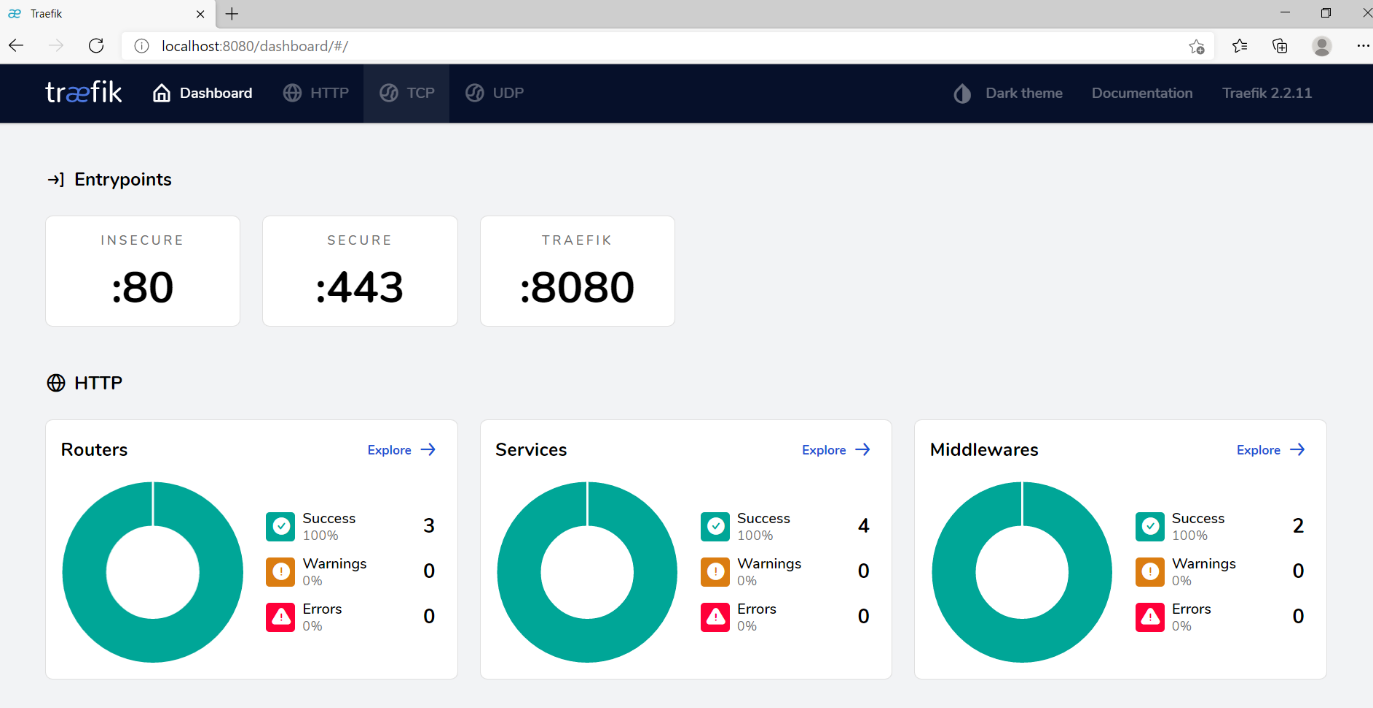


Note that before getting to this screen the browser will warn you that the https certificate is not trusted. We already know that this is because it is a self-signed certificate (see also certificate details in the screenshot above), so it is OK to override this warning. (To silence the warnings then you could install the self-signed cert into your computer’s Trusted Root CA store, plus set up “hosts” and/or DNS routing entries for the FQDN – however this may not be desirable and is not really necessary).

***Overcoming the security layers protecting this “secured” route in is something we will tackle as an exercise in its own right later!***

1. ***In a browser navigate to*** [***http://localhost:8080***](http://localhost:8080)

You should see a console for the Traefic Web Proxy:



It’s not really the purpose of this documentation to teach Traefik, however if you navigate around the menus then it should be fairly easy to see how the routing is configured.

1. ***In a browser navigate to*** [***http://localhost:5432***](http://localhost:5432)***1***

You should see a logon screen for the database client



***If you reach this point, then CONGRATULATIONS – your installation is successful!***

However we have so far only done some very simple smoke-tests. In the next document we will go back for a more detailed tour and to explore the functionality in more depth.

# Appendix A – Generating a Self-Signed Certificate

This is an optional exercise, if you wish to replace the provided self-signed SSL certificates with your own.

***NB: The example here describes how to generate a self-signed certificate-key pair. If you work in a large organisation then there may well be established procedures for generating SSL certificates to use on test servers, in which case you should obviously follow these established practices.***

Otherwise… using OpenSSL issue the command:

***openssl req -x509 -newkey rsa:4096 -sha256 -keyout openssl.key -out openssl.crt -subj "/CN=yourservernamehere" -days 600 -nodes***

Notes:

* On Windows you may have to go into a different command shell to do this – see “***YHCR FHIR Proxy Install 02 - Server Preparation***” for details
* The ***-nodes*** flag means that the key file is not password protected. Generally it is a good practice to use passwords, in which case remove this flag. However traefik cannot work with password protected key files.
* This generates a simple self-signed certificate with a CN, but not a corresponding Server Alternative Name (SAN). Whilst not essential for this exercise, populating a SAN is considered best-practice and is checked by modern browsers such as Chrome. For example, see <https://stackoverflow.com/questions/43665243/invalid-self-signed-ssl-cert-subject-alternative-name-missing>

To achieve this then change the command to:

***openssl req -x509 -newkey rsa:4096 -sha256 -keyout openssl.key -out openssl.crt -days 600 -config san.cnf -nodes***

And provide a configuration file (in this example called san.cnf) containing the following text:

[req]

distinguished\_name = req\_distinguished\_name

x509\_extensions = v3\_req

prompt = no

[req\_distinguished\_name]

CN = yoursevernamehere

[v3\_req]

keyUsage = critical, digitalSignature, keyAgreement

extendedKeyUsage = serverAuth

subjectAltName = @alt\_names

[alt\_names]

DNS.1 = yoursevernamehere

* If you are determined to actually silence the browser’s warnings about certificate mismatches, then you will need to satisfy the browser’s checks that the domain name used in the browser address bar matches the CN in the self-signed certificate. You will probably need to make routing entries to do this – eg either in the “hosts” file on the server where the browser is running or in a local DNS server which you control.