# How to make a dockerized plumbeR API secure with SSL and Basic Authentication

The use of docker containers by now is a well established technique to make the deployment of R scripts to a stable environment incredibly easy and reliable. In cases where you dockerize a shiny app or want to provide a REST API with plumber, often it is mandatory to somehow restrict the access to the resource in a corporate environment. This tutorial aims at showing you the simplest solution by running an NGINX reverse proxy alongside the R application. NGINX handles SSL encryption and a simple username/password authentication (HTTP Basic Authentication). There will be a follow-up tutorial showing a more elaborate approach, by separating these tasks in multiple containers in a Docker swarm, utilizing the AHUB open source framework (Link to AHUB on Github).

This tutorial assumes, that you are already familiar with the concept of Docker and have at least once built an R based container with a Dockerfile. If not, please refer to the excellent posts from XXX and XXX.

The following diagram shows the setup we want to achieve:

<diagram>

The plumber API listens on port 8000, which we not make available to the outside of the container. Instead we install the very lean NGINX http server listening on port 80 and route all traffic through it. Both encryption and password authentication can be enabled for NGINX with minimal configuration effort.

As a basis for a plumber application we use the simple example from the plumber main page (LINK).

<Code snippets>

When preparing an unsecured container we would use the following Dockerfile.

<Dockerfile>

Building and running it with the following commands:

<Powershell>

Now we can access the resource by browsing to localhost:8000/plot

As a first step toward securing our API we want to install NGINX and route all traffic through it. For this we change our Dockerfile as follows.

<Dockerfiler>

Note the following changes:

* We added some apt-get commands for installing nginx and its dependencies
* We created a password file with the username **test** and password **test.** Of course you should change these for your application. You can also add as many user credentials as you like by just repeating this line with different values.
* We copy a configuration file nginx.conf to the appropriate location
* We publish the Port 80 instead of 8000. Plumber therefore is not reachable directly anymore from outside the container.
* We changed the CMD to first start the nginx service before running the R script

Now we need to prepare the configuration file nginx.conf, telling the proxy to listen on port 80 and that it should relay all requests to localhost:8000.

<config file>

Save this file as nginx.conf in the main folder. Your directory should now look like this:

<folder>

Now we build the container again and run it while mapping port 80.

The authentication popup should appear (depending on your browser, this might look different). Atfer entering the credentials you are given access to the resource.

Since sending your credentials over an unencrypted connection is not very secure, we need to follow with the next step: activating SSL.

We change the Dockerfile as follows:

<Dockerfile>

Note the changes:

* We added a command to create a self-signed certificate and key and store both files in the folder /etc/ssl/private
* We additionally expose the port 443, which is the default HTTPS port

Finally an update to the nginx.conf:

<config file>

Here we have created two servers:

* The first listens on port 80 and redirects all traffic to <https://localhost/>...
* The second listens on the SSL port 443 and points to both files we have created in the Dockerfile. The basic authentication directive we leave untouched.

Now we build and run the container again:

<Powershell>

When navigating to <http://localhost> we directly get transferred to <https://localhost> and a browser warning should appear. This is because we signed the certificate ourselves and not issued an official certificate from a certificate authority. But the connection is encrypted eitherway and we can skip this warning, since we can trust ourselves.

Tadaa! Now we have secured our resource with encryption and password protection with minimal additional configuration.

On the next tutorial in this series, I will show how to enable security for multiple containers in a container swarm scenario.