Securing a dockerized plumber API with SSL and Basic Authentication

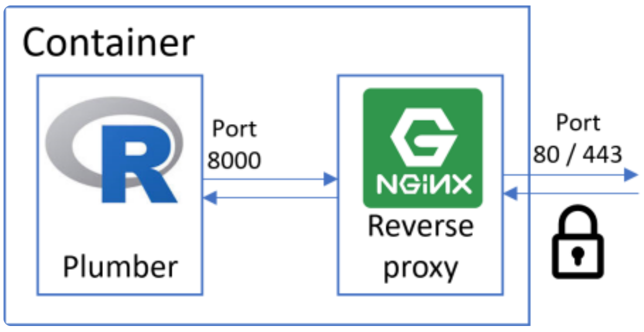
Posted on March 28, 2019 by **Martin Hanewald** in **R bloggers** | 0 Comments

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 (github.com/qunis/ahub).

The sample files for this tutorial can be found on github.com/MartinHanewald/rbloggerstutorial1.

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 Colin Fay (www.r-bloggers.com/an-introduction-to-docker-for-r-users) and Oliver Guggenbühl (www.r-bloggers.com/running-your-r-script-in-docker).

The following diagram shows the setup we want to achieve:



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.

Basic setup

As a basis for a plumber application we use the simple example from the plumber main page (https://www.rplumber.io/).

start_api.R

```
library(plumber)
r <- plumb("api_functions.R")
r$run(port=8000, host='0.0.0.0')
```

api_functions.R

```
# plumber.R

#* Echo back the input
#* @param msg The message to echo
#* @get /echo
function(msg=""){
    list(msg = paste0("The message is: '", msg, "'"))
}

#* Plot a histogram
#* @png
#* @get /plot
function(){
    rand <- rnorm(100)
    hist(rand)
}

#* Return the sum of two numbers
#* @param a The first number to add
#* @param b The second number to add
#* @post /sum
function(a, b){
    as.numeric(a) + as.numeric(b)
}
```

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

Dockerfile

```
FROM rocker/r-base

# install R packages
RUN install2.r \
plumber

EXPOSE 8000

ADD . /app
WORKDIR /app

CMD R -e "source('start_api.R')"
```

Building and running it with the following commands in *bash* or *powershell*:

```
docker build -t plumber_insecure .
docker run --rm -p 8000:8000 plumber_insecure
```

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

Adding authentication

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.

Dockerfile

```
FROM rocker/r-base

# install R packages
RUN install2.r \
plumber

# setup nginx
RUN apt-get update && \
apt-get install -y nginx apache2-utils && \
htpasswd -bc /etc/nginx/.htpasswd test test

ADD ./nginx.conf /etc/nginx/nginx.conf

EXPOSE 80

ADD . /app
WORKDIR /app

CMD service nginx start && R -e "source('start_api.R')"
```

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.

nginx.conf

```
user www-data;
worker_processes auto;

events {
  worker_connections 1024;
}

http {
  server {
    listen 80;
    auth_basic "Username and Password are required";
    auth_basic_user_file /etc/nginx/.htpasswd;

    location / {
      proxy_pass http://127.0.0.1:8000/;
    }
  }
}
```

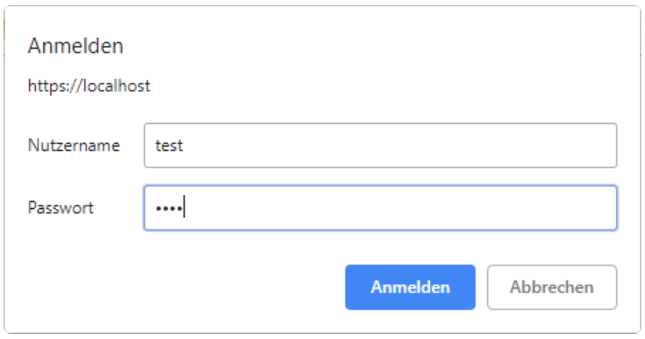
Save this file as nginx.conf in the main folder. Your directory should now contain the following files:

```
start_api.R

api_functions.R

Dockerfile

nginx.conf
```

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

```
docker build plumber_auth .
docker run --rm -p 80:80 plumber_auth
```

The authentication popup should appear when navigating to http://localhost/plot (depending on your browser, this might look different).


Anmelden

https://localhost

Nutzername: test

Passwort: ••••

After entering the credentials you are given access to the resource.

Adding SSL encryption

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

```
FROM rocker/r-base

# install R packages
RUN install2.r \
   plumber

# setup nginx
RUN apt-get update && \
    apt-get install -y nginx apache2-utils && \
    htpasswd -bc /etc/nginx/.htpasswd test test

RUN openssl req -batch -x509 -nodes -days 365 -newkey rsa:2048 \
        -keyout /etc/ssl/private/server.key \
        -out /etc/ssl/private/server.crt

ADD ./nginx.conf /etc/nginx/nginx.conf

EXPOSE 80 443

ADD . /app
WORKDIR /app

CMD service nginx start && R -e "source('start_api.R')"
```

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:

nginx.conf

```
user www-data;
worker_processes auto;

events {
  worker_connections 1024;
}

http {
    server {
        listen 80;
        return 301 https://$host$request_uri;
    }

    server {
        listen 443 ssl;
        ssl_certificate     /etc/ssl/private/server.crt;
        ssl_certificate_key /etc/ssl/private/server.key;

        auth_basic "Username and Password are required";
        auth_basic_user_file /etc/nginx/.htpasswd;

        location / {
            proxy_pass http://127.0.0.1:8000/;
        }
  }
}
```

Here we have created two servers:

- The first listens on port 80 and redirects all traffic to https://... on port 443
- The second listens on the SSL port 443 and points to the key and certificate files we have created in the Dockerfile.

Now we build and run the container again. This time we need to publish the SSL port as well:

```
docker build -t plumber_auth_ssl .
docker run --rm -p 80:80 -p 443:443 plumber_auth_ssl
```

When navigating to http://localhost/plot we directly get transferred to https://localhost/plot 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.