Data Engineering 3 – Final Assignment

This document outlines my solution for the final assignment for the Data Engineering 3 class at Central European University. Please find my credentials, including the EC2 instance ID in the header of each page. The document goes through the following sections:

- API construction and presentation of R script;
- Dockerization of the API;
- Launching the API as an ECS service.

1. API construction & R script

To construct my API, I used the *binancer* package as we did in class. I implemented 4 distinct endpoints for the API. As a brief overview, these are:

1. */narrowest_spreads*: an endpoint to get back the most liquid N symbols (N is user-defined), based on bid-ask spread. The narrower the spread (in relative terms), the more liquid the symbol is considered.
2. */compare_summary*: an endpoint to generate a short HTML summary page for selected two symbols, with parameterized interval and lookback period. The summary page shows some simple descriptive statistics and a chart comparing the relative price changes of the two symbols for the given period.
3. */candlestick_chart:* this endpoint generates a candlestick chart for a given symbol with parameterized interval and lookback period. In addition, the API can take some technical indicators (like SMA or EMA, with specific period parameters) and also include these on the plot.
4. */forecast*: an endpoint that creates an auto-ARIMA model based on the last 1000 observations for a given symbol, and makes a prediction for the next N periods, with a given interval.

Now, let's dive into the R implementation of these endpoints. The API uses the following packages: *binancer, deplyr, scales, ggplot2, quantmod, forecast, rmarkdown, plumber*.
Some basic description is set-up for the API is achieved with the following code:

API title and description set-up

```
#* @apiTitle Marton Nagy's Crypto Analytics API
#* @apiDescription Real-time Binance crypto analytics using binancer -
Developed for DE3 class at CEU
```

1.1 Narrowest spreads endpoint

The logic of this endpoint is to query the most recent bid and ask prices for all symbols from Binance using the *binance_ticker_all_books()* function. Then I calculate the absolute spread values and the relative spreads. I then sort by the relative spreads and return the first N elements, where N is an API parameter. The result is serialized as a JSON (by default). The code that does this is the following.

Narrowest spreads endpoint R code

```
#--------------------
# 1. Narrowest spreads
#--------------------
#* Get N coins with the narrowest spread (proxy for liquidity of coins)
#* @param n Number of symbols to return
#* @get /narrowest_spreads
function(n = 10) {
  n <- as.integer(n)
  books <- binance_ticker_all_books()
  books <- books %>%
    filter(bid_price > 0 & ask_price > 0) %>%
```

```
    mutate(
      spread_abs = ask_price - bid_price,
      spread_rel = spread_abs / ((ask_price + bid_price) / 2)
    ) %>%
    arrange(spread_rel) %>%
    select(symbol, spread_abs, spread_rel) %>%
    head(n)
  return(books)
}
```

1.2 Compare summary endpoint

The second endpoint is useful to get a basic comparison page about two selected symbols. Additionally, the API has parameters for the interval and the lookback period to use. The result is a rendered Rmarkdown document, serialized as HTML. The code achieving this is the following.

Compare summary endpoint R code

```
#--------------------
# 2. Comparison summary
#--------------------
#* Get comparison summary page for selected two symbols
#* @param symbol1 First symbol (e.g., BTCUSDT)
#* @param symbol2 Second symbol (e.g., ETHUSDT)
#* @param interval Binance interval (e.g., 1h)
#* @param lookback Lookback period (in number of interval units)
#* @serializer html
#* @get /compare_summary
function(res, symbol1 = "BTCUSDT", symbol2 = "ETHUSDT", interval = "1m",
lookback = 360) {
  filename <- tempfile(fileext = '.html')
  on.exit(unlink(filename))
  params = list(symbol1 = symbol1, symbol2 = symbol2, interval = interval,
lookback = lookback)
  render('price_comparison_report.Rmd', output_file = filename, params =
params)
  include_file(filename, res)
}
```

The actual Rmarkdown, however, is masked from this code as it is located in a separate .Rmd file. The parameters of the endpoint are passed to this file by the *render* function. The code basically queries the candlestick data for the given symbols with the given parameters. Then, it first calculates the summary statistics (both in absolute and relative terms). Relative price means in this context the price in percentage terms relative to the price at the start date. Next, it also plots the relative prices for the two symbols. For a nicer look, the code is not printed in the rendered document. The code in this document is the following.

Compare summary endpoint Rmd code

````
---
title: "Price comparison of selected symbols"
output: html_document
date: "Generated at: `r Sys.Date()`"
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = FALSE, warning=FALSE)
library(binancer)
library(dplyr)
````
````

```
library(scales)
library(ggplot2)
library(knitr)

lookback <- as.integer(params$lookback)

df1 <- binance_klines(params$symbol1, interval = params$interval, limit =
lookback)
df2 <- binance_klines(params$symbol2, interval = params$interval, limit =
lookback)

df1 <- df1 %>% mutate(rel_price = 100 * close / first(close))
df2 <- df2 %>% mutate(rel_price = 100 * close / first(close))
```

Welcome to the price comparison site!

You are now comparing the price evolution of `r params$symbol1` and `r
params$symbol2`.

Descriptive statistics

Statistics about `r params$symbol1`

In original units:

```{r echo=FALSE}
pander::pander(summary(df1$close))
```

In relative terms (100=starting period):

```{r echo=FALSE}
pander::pander(summary(df1$rel_price))
```

Statistics about `r params$symbol2`:

In original units:

```{r echo=FALSE}
pander::pander(summary(df2$close))
```

In relative terms (100=starting period):

```{r echo=FALSE}
pander::pander(summary(df2$rel_price))
```

Price comparison plot

The below plot shows the relative price evolution of the two symbols, with
100% being the starting period.

```{r echo=FALSE}
ggplot() +
  geom_line(data = df1, aes(x = close_time, y = rel_price, color =
params$symbol1)) +
```

```
  geom_line(data = df2, aes(x = close_time, y = rel_price, color =
params$symbol2)) +
  labs(title = paste0("Relative Price Comparison of ", params$symbol1, " and
", params$symbol2),
       subtitle = paste0("Frequency: ", params$interval, ", No. of periods: ",
lookback),
       x = "Time",
       y = "Relative Price (Start = 100%)",
       color = "Symbol") +
  scale_color_manual(values = c("#5D576B", "#EB6534")) +
  theme_minimal() +
  theme(legend.position = 'top',
        plot.title = element_text(hjust = 0.5),
        plot.subtitle = element_text(hjust = 0.5))
```

1.3 Candlestick chart with technical indicators endpoint

This endpoint takes a symbol and an interval and lookback period parameter. In addition, it also takes a list of technical indicators (simple or exponential moving average, with numbers after colons denoting the number of periods to use for averaging). First, the simple candlestick chart is plotted. Then, the list of technical indicators is parsed and looped through, with the indicator values calculated and added to the plot. Finally, the plot is printed and serialized as a PNG. The code for this endpoint is the following.

Candlestick chart endpoint R code

```
#--------------------
# 3. Candlestick chart with indicators
#--------------------
#* Get candlestick chart of a given symbol with optional technical indicators
#* @param symbol Trading symbol (e.g., BTCUSDT)
#* @param interval Frequency (e.g., 1h)
#* @param lookback Lookback in number of frequency units
#* @param indicators Comma-separated selection of SMA or EMA, with numbers
after the indicators used to indicate number of periods to use.
#* @serializer png
#* @get /candlestick_chart
function(symbol = "BTCUSDT", interval = "1h", lookback = 360, indicators =
"SMA:24,EMA:24,SMA72,EMA:72") {
  lookback <- as.integer(lookback)
  ind_list <- strsplit(trimws(indicators), ",")[[1]]
  df <- binance_klines(symbol, interval = interval, limit = lookback)
  ohlc <- xts::xts(df[, c("open", "high", "low", "close")], order.by =
df$open_time)
  color_palette <- scales::hue_pal()(length(ind_list))

  img <- ggplot(df, aes(open_time)) +
    geom_linerange(aes(ymin = open, ymax = close, color = close < open), size
= 2, show.legend = FALSE) +
    geom_errorbar(aes(ymin = low, ymax = high), size = 0.25) +
    scale_y_continuous(labels = dollar) +
    scale_color_manual(values = c('#1a9850', '#d73027'))

  i <- 1
  for (ind in ind_list) {
    ind <- trimws(ind)
    indicator_name <- toupper(gsub("[^A-Za-z]", "", ind))
    period <- as.numeric(gsub("[^0-9]", "", ind))
```

```
    switch(indicator_name,
           "SMA" = {
             df$SMA <- SMA(Cl(ohlc), n = period)
             img <- img + geom_line(data = df, aes(x = open_time, y = SMA),
color = color_palette[i], show.legend = FALSE)
           },
           "EMA" = {
             df$EMA <- EMA(Cl(ohlc), n = period)
             img <- img + geom_line(data = df, aes(x = open_time, y = EMA),
color = color_palette[i], show.legend = FALSE)
           }
    )
    i <- i + 1
  }

  img <- (img +
            theme_bw() +
            ylab('Price') +
            xlab('Time') +
            ggtitle(paste(symbol, '- last updated: UTC', format(Sys.time(),
"%Y-%m-%d %H:%M:%S")))
  )
  y_pos_start <- max(df$high, na.rm = TRUE)
  for (j in seq_along(ind_list)) {
    img <- img + annotate("text",
                          x = min(df$open_time),
                          y = y_pos_start - (j - 1) * (0.01 * y_pos_start),
                          label = ind_list[j],
                          hjust = 0,
                          vjust = 1,
                          color = color_palette[j],
                          size = 3)
  }

  print(img)
}
```

1.4 Forecast endpoint

The last endpoint implemented takes a symbol, an interval and the number of forecast periods. It then queries the last 1000 observations using *binance_klines*. From these, a time series of the closing prices is constructed, with frequency inferred from the given interval. An auto-ARIMA model is then fitted to this time series. Lastly, a forecast is made for the given number of periods. This result is serialized as a CSV (just so that I have four different serializers – this easily could have been another JSON endpoint). The code for this endpoint is the following.

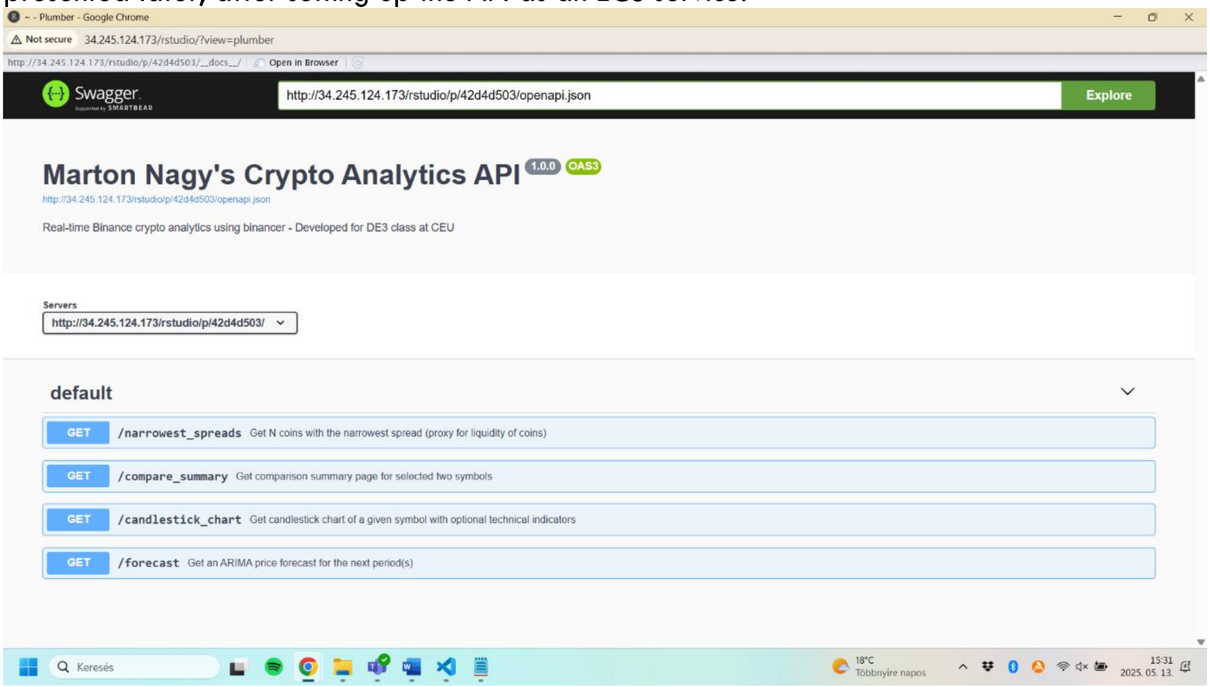
Forecast endpoint R code

```
#--------------------
# 4. Forecast with ARIMA
#--------------------
#* Get an ARIMA price forecast for the next period(s)
#* @param symbol Symbol (e.g., BTCUSDT)
#* @param interval Interval (e.g., 1h)
#* @param forecast_period Number of periods to forecast for
#* @serializer csv
#* @get /forecast
function(symbol = "BTCUSDT", interval = "1h", forecast_period = 1) {
```

```
  freq = switch(interval,
                "1m" = 1440,
                "3m" = 480,
                "5m" = 288,
                "15m" = 96,
                "30m" = 48,
                "1h" = 24,
                "2h" = 12,
                "4h" = 6,
                "6h" = 4,
                "8h" = 3,
                "12h" = 14,
                "1d" = 7,
                "3d" = 7/3,
                "1w" = 52,
                "1M" = 12
  )
  df <- binance_klines(symbol, interval = interval, limit = 1000)
  ts_data <- ts(df$close, frequency = freq)
  fit <- auto.arima(ts_data)
  fcast <- forecast(fit, h = as.integer(forecast_period))$mean
  return(as.data.frame(fcast))
}
```

1.5 API check

To confirm that the API is functional, I ran it from Rstudio Server. Below is the screenshot of the API home page. I also tested each endpoint manually. Example outputs of the endpoints will be presented later, after setting up the API as an ECS service.



2. Dockerization of the API

In order to deploy the API as an ECS service, I first needed to dockerize the API I have built. For this, I first built a Dockerfile. Then, I created a docker image of the API. To test that everything works, I also created a container of the image on my EC2 instance and visited the API on port 8000. The contents of the Dockerfile, the shell commands to build the image and run the container, and the screenshot of the API running in the container are included in the next sections.

Dockerfile

```
FROM rstudio/plumber

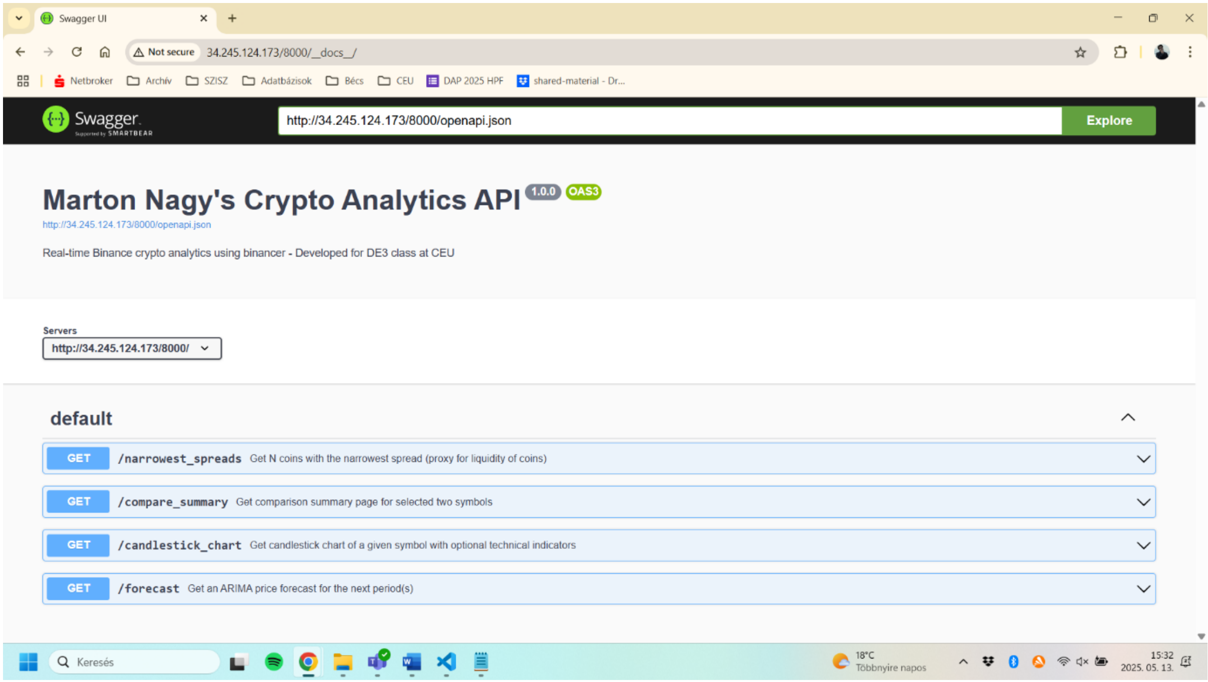
RUN apt-get update && apt-get install -y pandoc && apt-get clean && rm -rf
/var/lib/apt/lists/
RUN install2.r ggplot2 quantmod forecast rmarkdown dplyr pander readr
RUN installGithub.r daroczig/binancer
ADD price_comparison_report.Rmd /app/price_comparison_report.Rmd
ADD mnagy_de3_ha_api.R /app/mnagy_de3_ha_api.R
EXPOSE 8000
WORKDIR /app
```

Build docker image command

```
sudo docker build -t mnagy_de3_ha_api .
```

Run docker container command

```
sudo docker run --rm -p 8000:8000 -ti mnagy_de3_ha_api mnagy_de3_ha_api.R
```



As the dockerization seems to be working properly, we can now push the image to ECR. For this, we first have to create a private ECR repository. The below screenshot shows how this has been set-up. Next, from the command line, we can push the image to ECR – with the shell commands shown below.

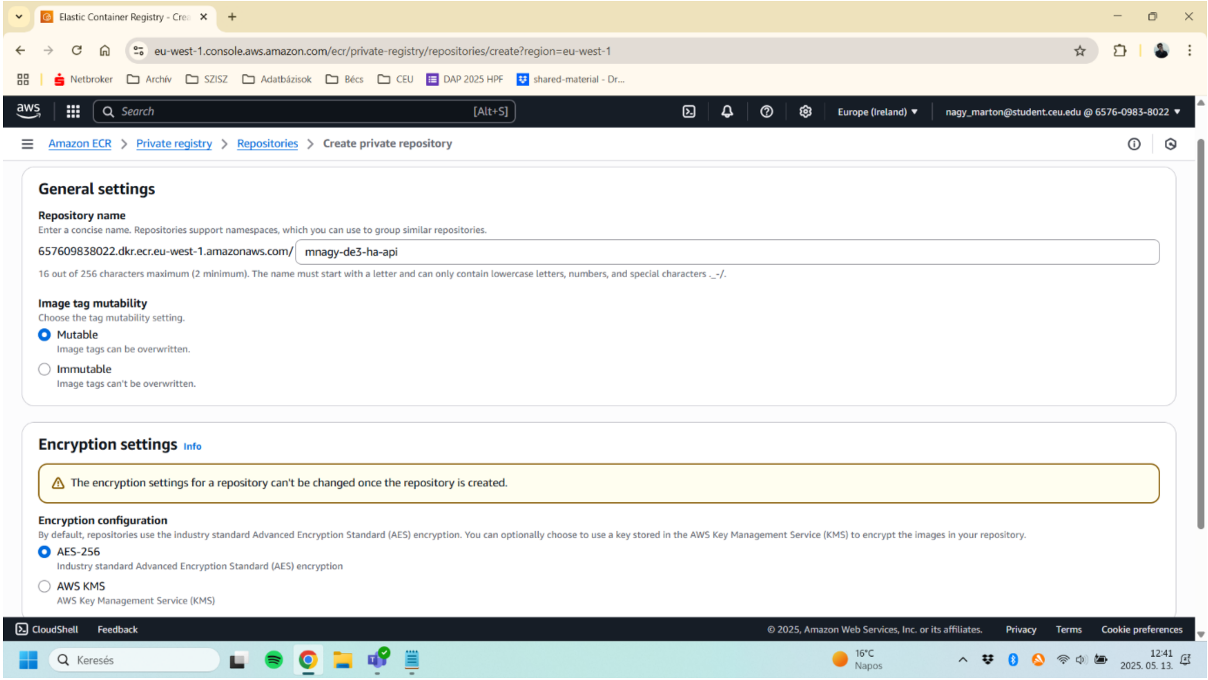
Shell commands to push image to ECR

```
aws ecr get-login-password --region eu-west-1 | sudo docker login --username
AWS --password-stdin 657609838022.dkr.ecr.eu-west-1.amazonaws.com

sudo docker tag mnagy_de3_ha_api:latest 657609838022.dkr.ecr.eu-west-
1.amazonaws.com/mnagy-de3-ha-api:latest

sudo docker push 657609838022.dkr.ecr.eu-west-1.amazonaws.com/mnagy-de3-ha-
api:latest
```

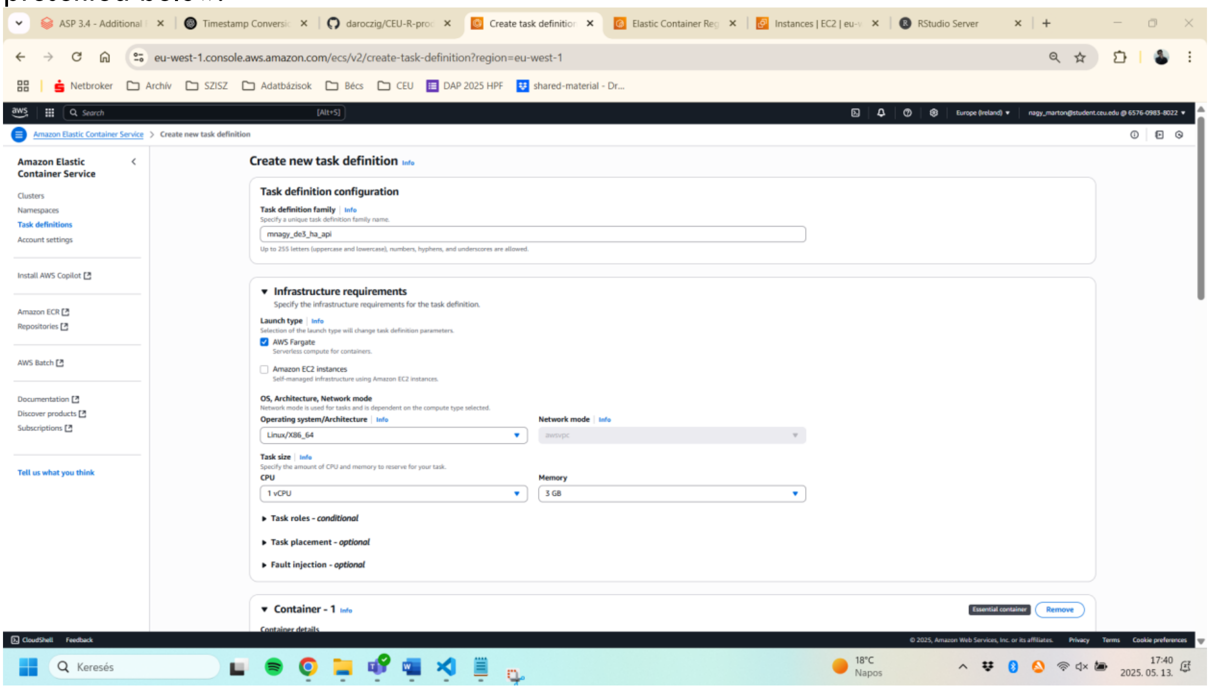
Author: Márton Nagy
Student ID: 2402779
Instance ID: i-0657dc34b986263aa (mnagy_de3_home_assignment)



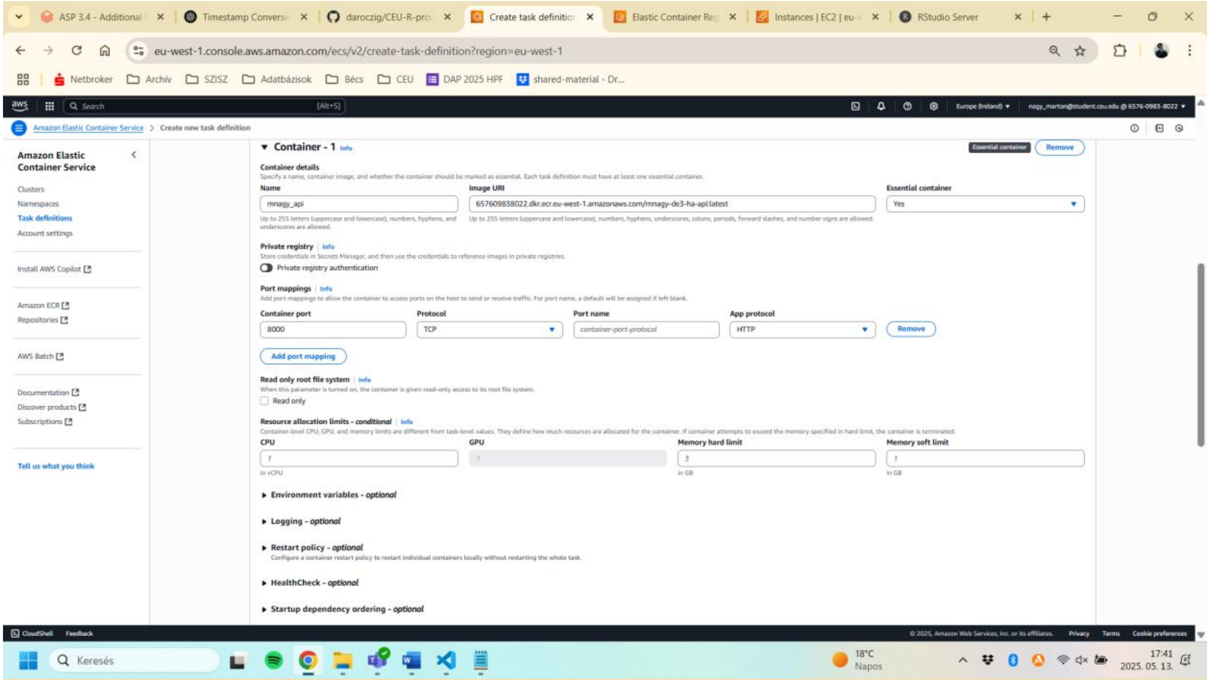
3. Launching the API as an ECS service

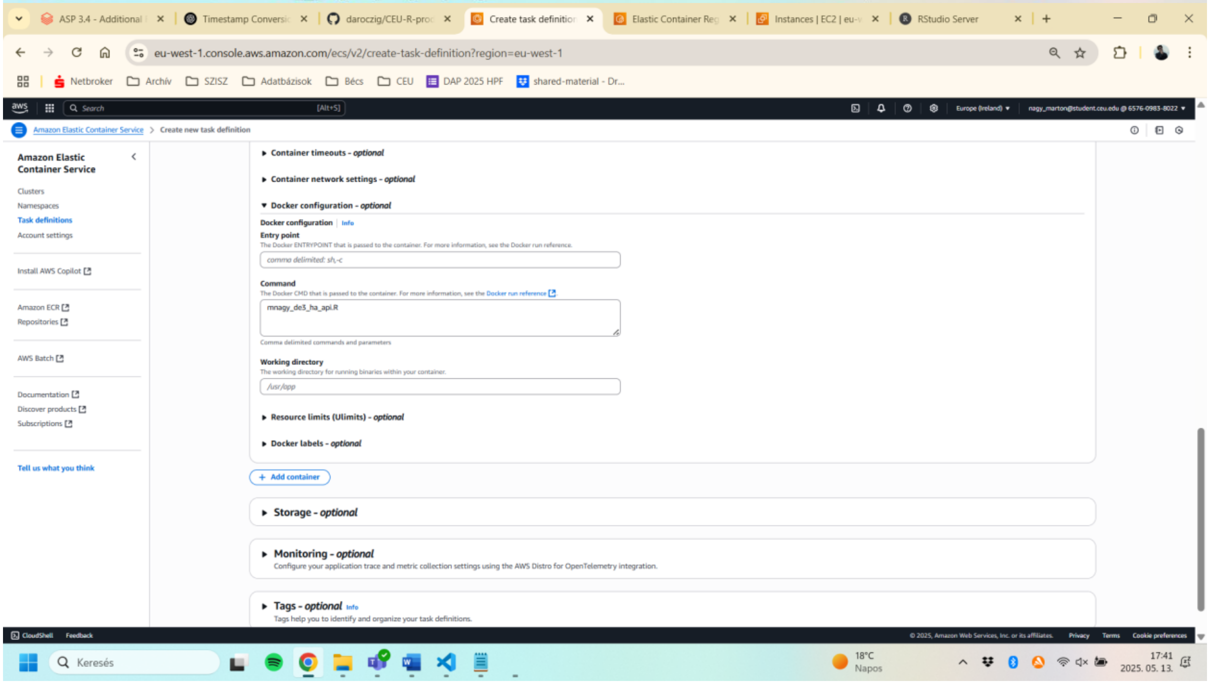
The last thing to do was to set-up the API as an ECS service. As this has been done on the GUI of AWS, the below screenshots will illustrate the steps needed to achieve this. In the end, the example response of the four API endpoints will also be presented, as accessed through the public ECS URL.

The first step was to set-up a new task definition. The inputs that I gave on the AWS GUI are presented below.



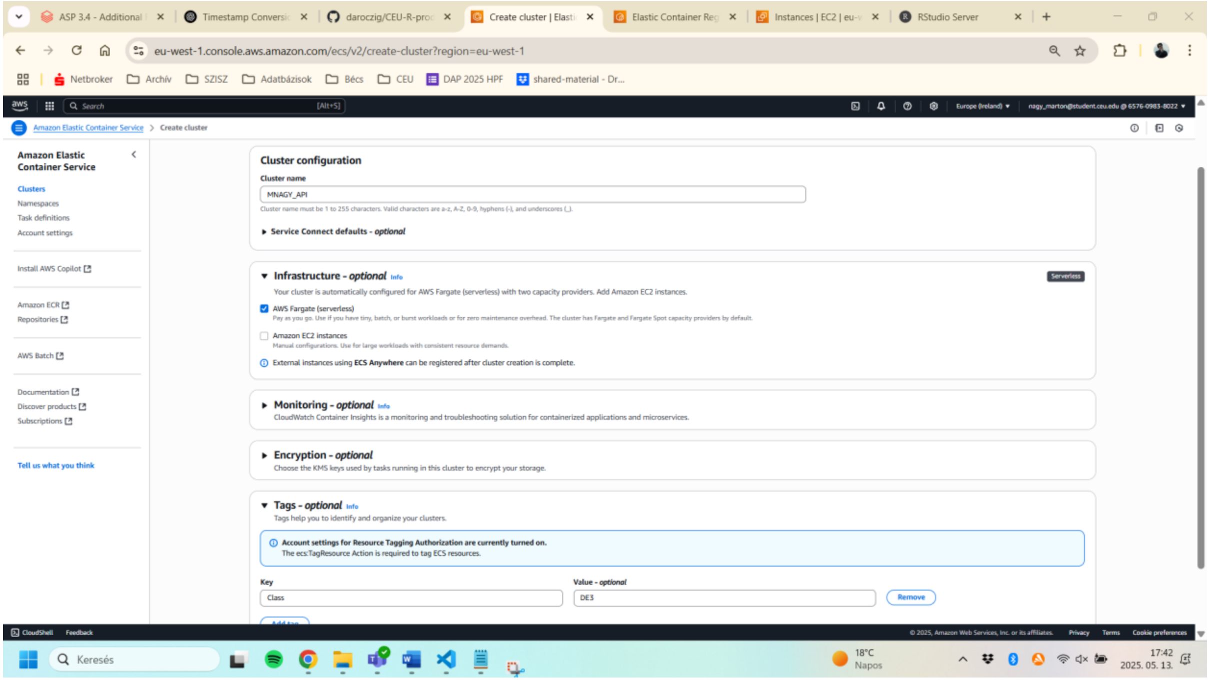
Author: Márton Nagy
Student ID: 2402779
Instance ID: i-0657dc34b986263aa (mnagy_de3_home_assignment)



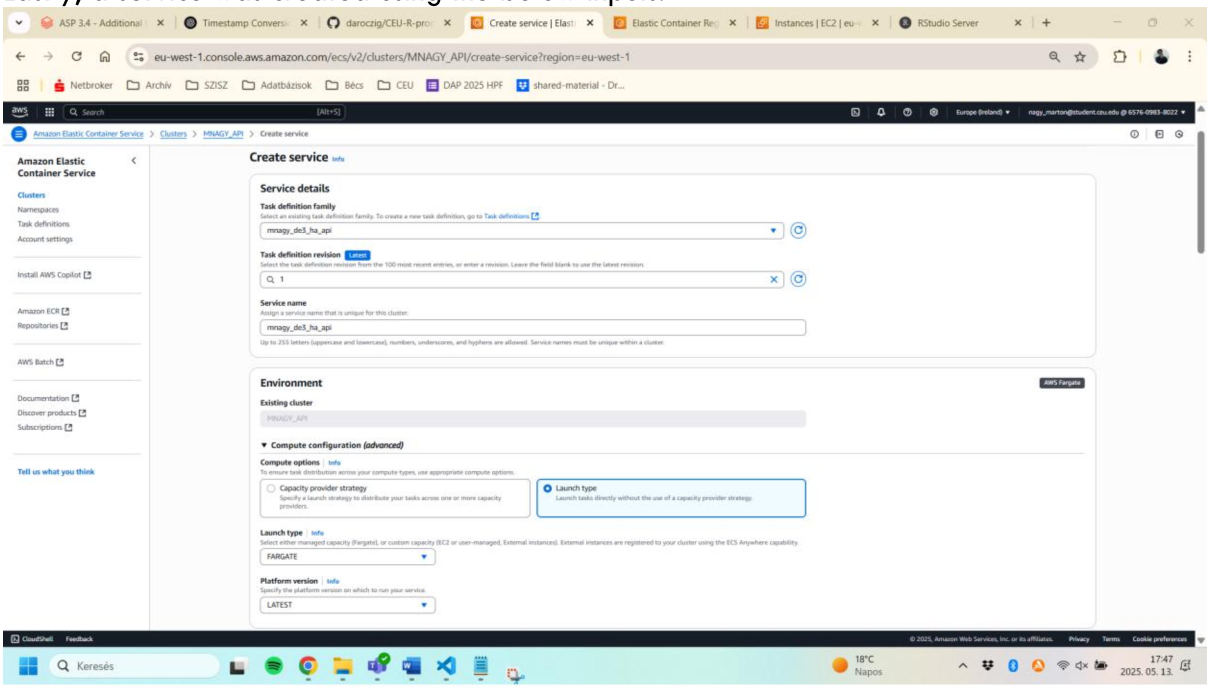


The next piece was to create a new cluster. Again, the inputs given are shown below.

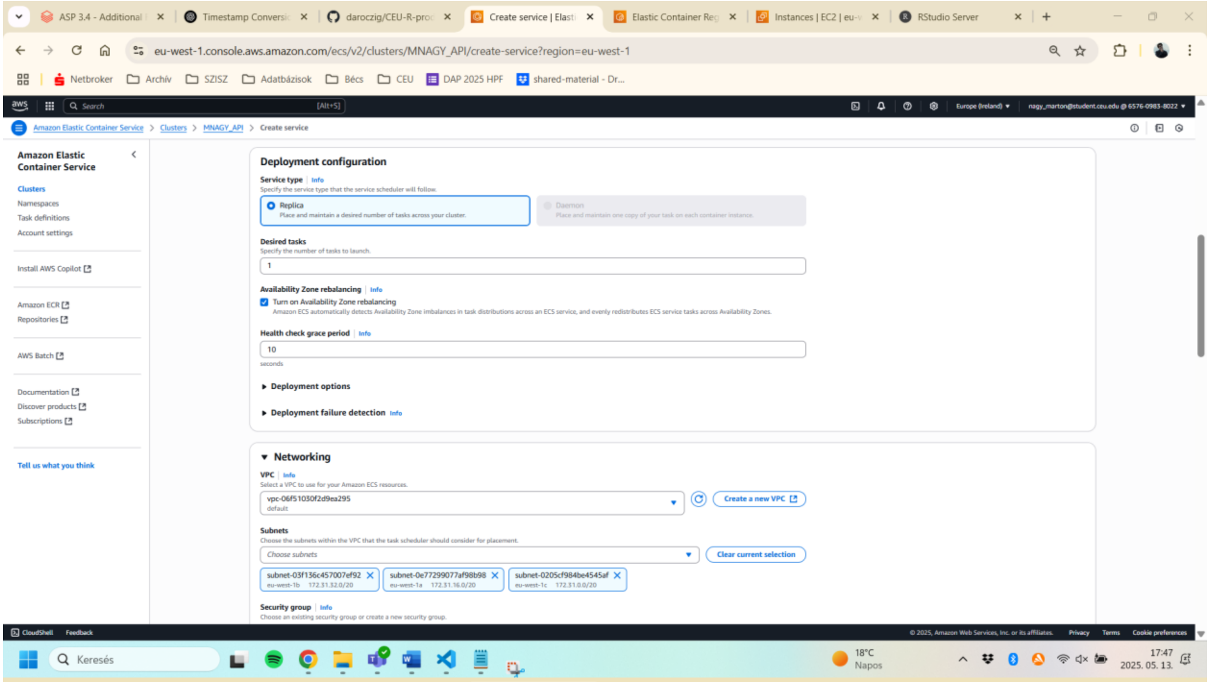
Author: Márton Nagy
Student ID: 2402779
Instance ID: i-0657dc34b986263aa (mnagy_de3_home_assignment)

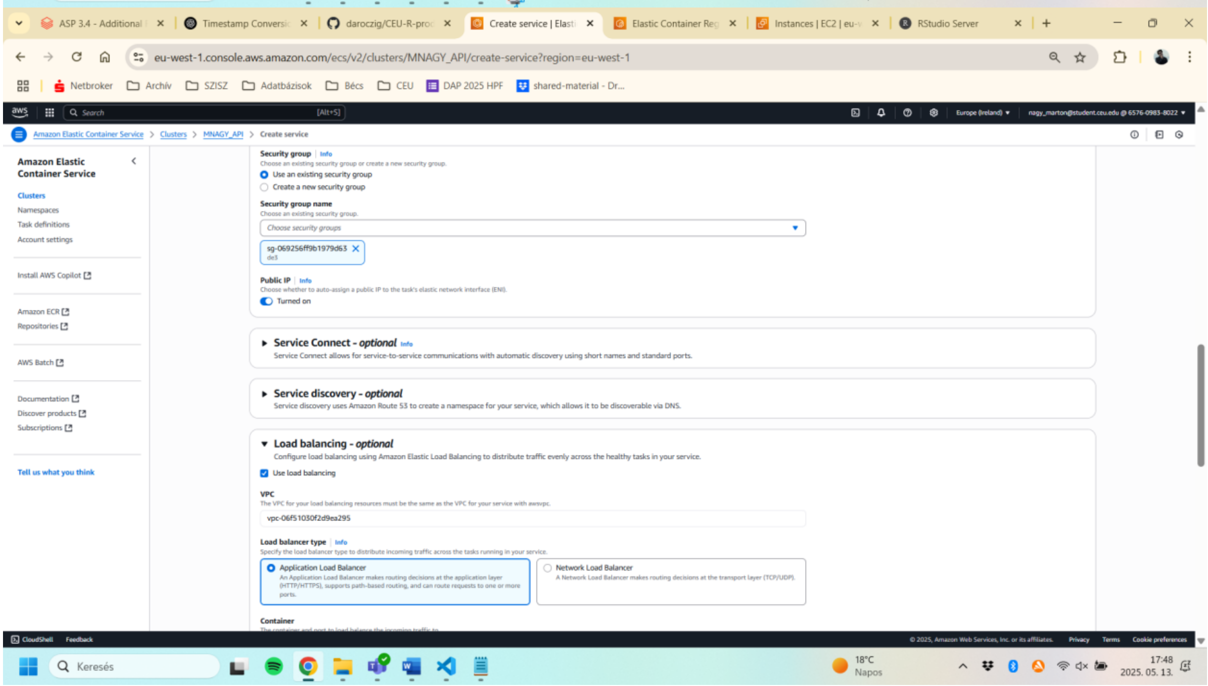


Lastly, a service was created using the below inputs.

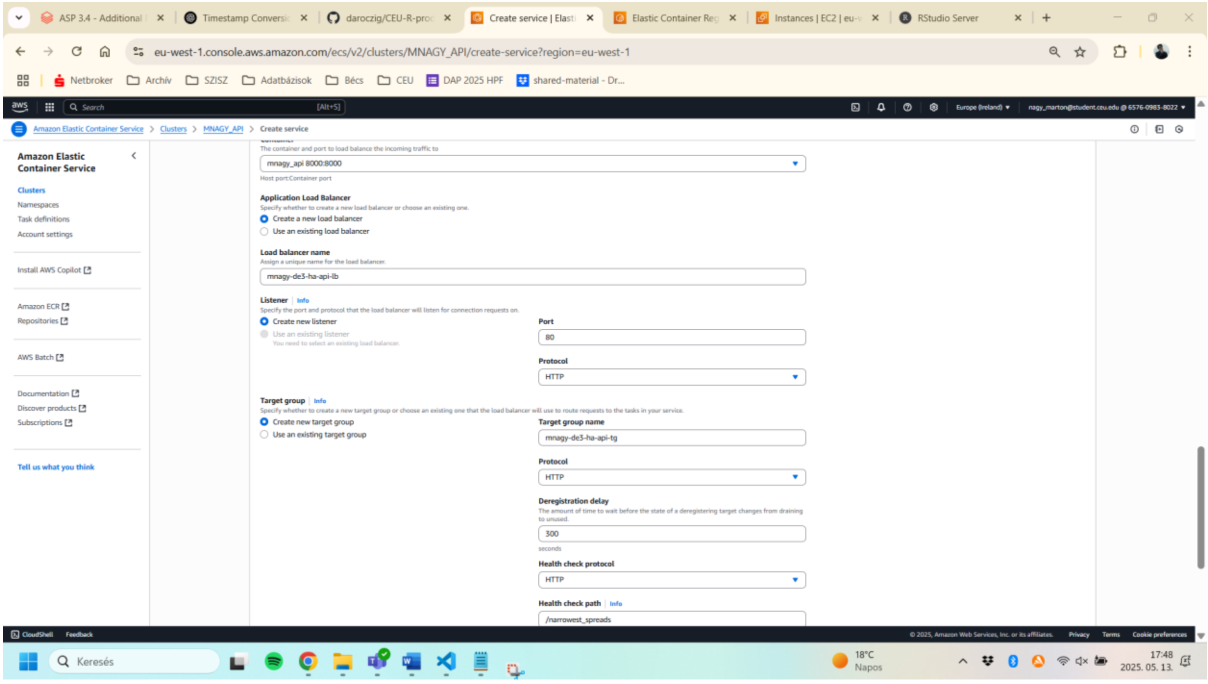


Author: Márton Nagy
Student ID: 2402779
Instance ID: i-0657dc34b986263aa (mnagy_de3_home_assignment)

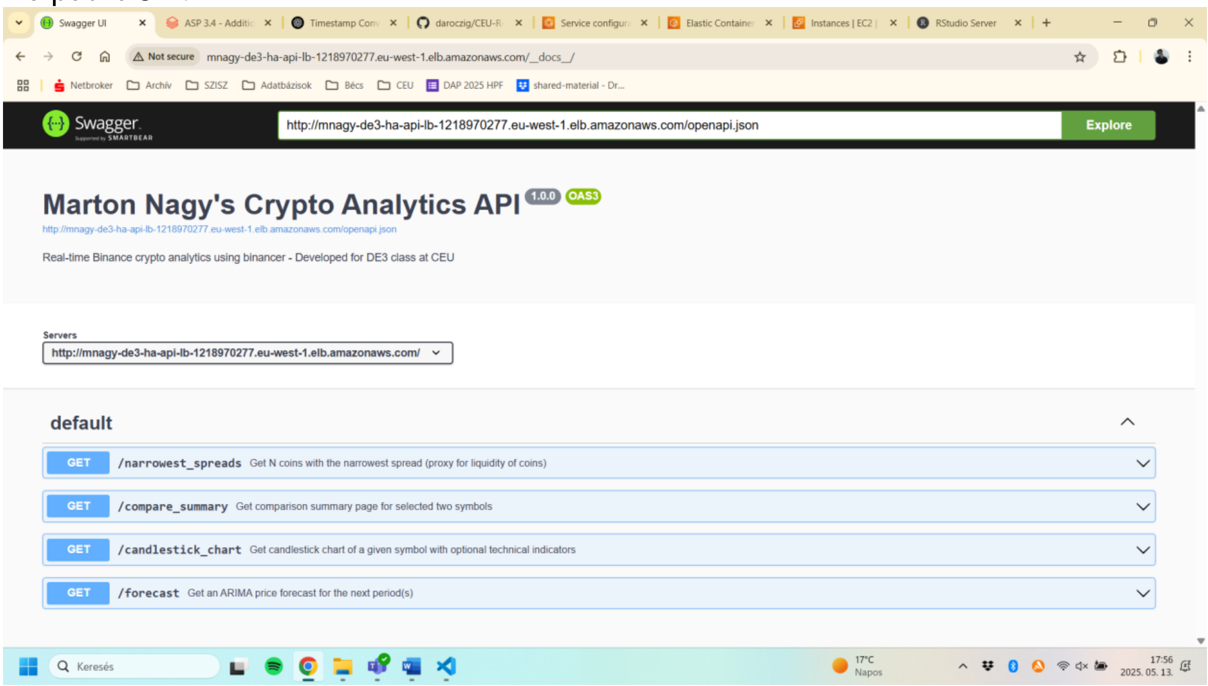




Author: Márton Nagy
Student ID: 2402779
Instance ID: i-0657dc34b986263aa (mnagy_de3_home_assignment)



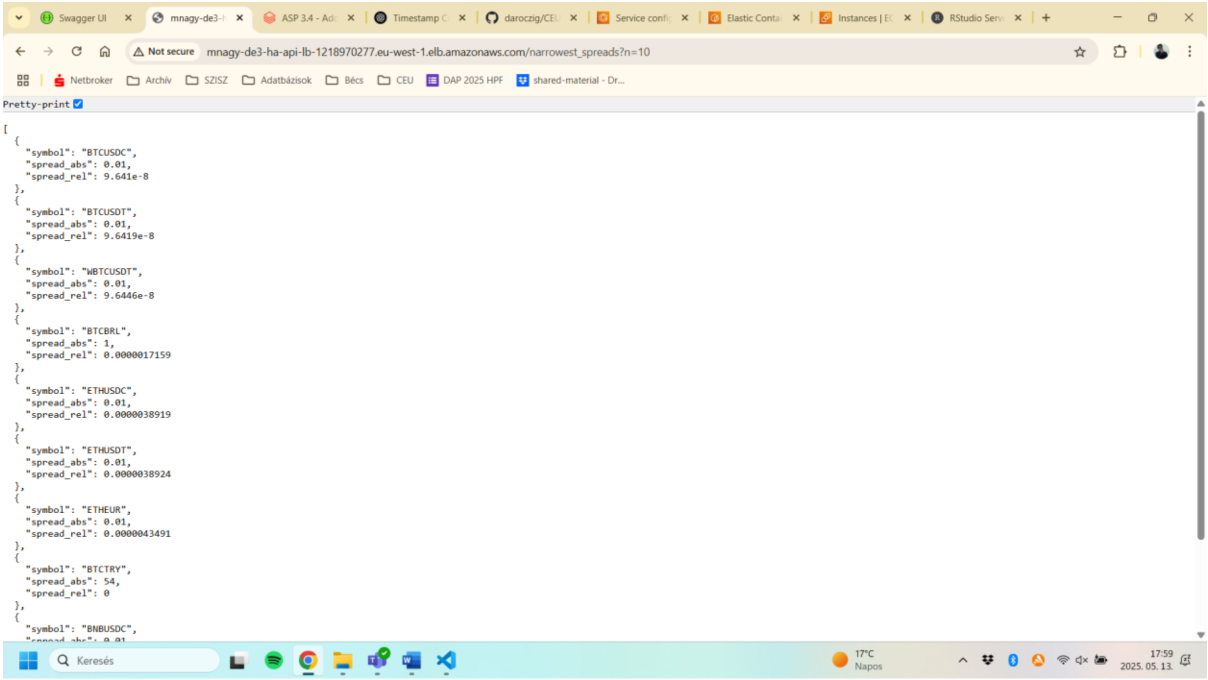
Now let's check that the API is working. For this, the below screenshot shows the API running on the public URL.



After the ECS service was up and running, I visited the API at the public URL. Below are screenshots of some example API responses.

Narrowest spreads endpoint

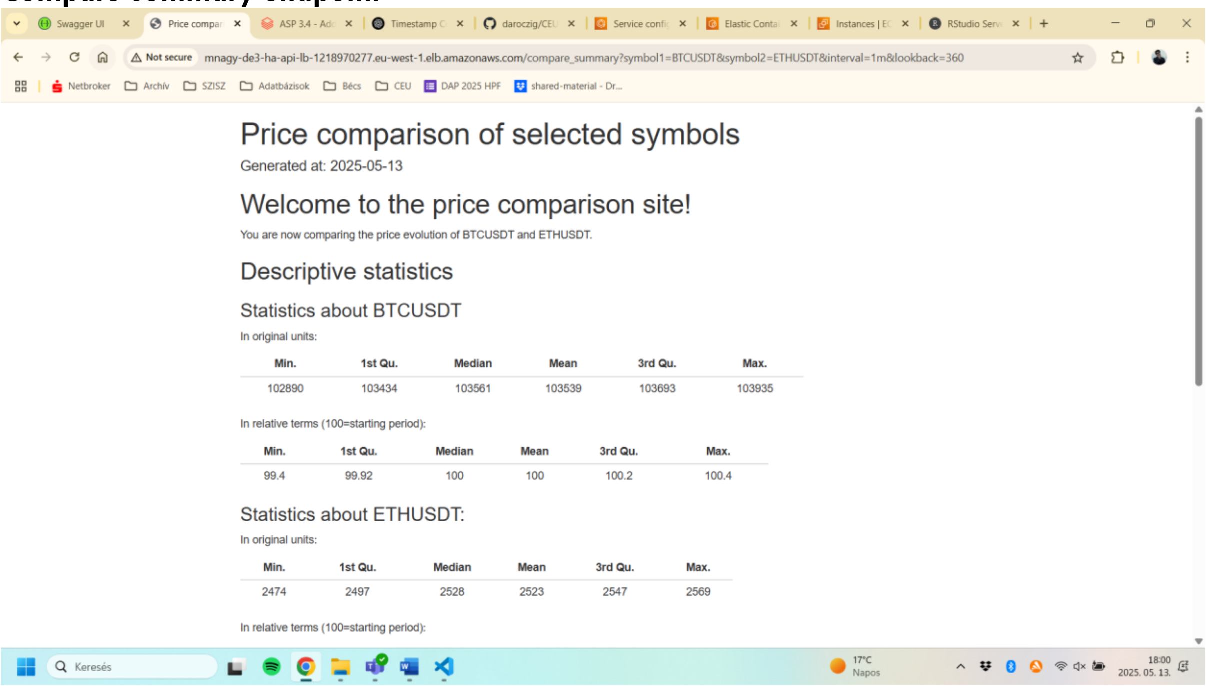
Author: Márton Nagy
Student ID: 2402779
Instance ID: i-0657dc34b986263aa (mnagy_de3_home_assignment)


mnagy-de3-ha-api-lb-1218970277.eu-west-1.elb.amazonaws.com/narrowest_spreads?n=10

Pretty-print


```
[
  {
    "symbol": "BTCUSDC",
    "spread_abs": 0.01,
    "spread_rel": 9.641e-8
  },
  {
    "symbol": "BTCUSDT",
    "spread_abs": 0.01,
    "spread_rel": 9.6419e-8
  },
  {
    "symbol": "WBTCUSDT",
    "spread_abs": 0.01,
    "spread_rel": 9.6446e-8
  },
  {
    "symbol": "BTCBRL",
    "spread_abs": 1,
    "spread_rel": 0.0000017159
  },
  {
    "symbol": "ETHUSDC",
    "spread_abs": 0.01,
    "spread_rel": 0.0000038919
  },
  {
    "symbol": "ETHUSDT",
    "spread_abs": 0.01,
    "spread_rel": 0.0000038924
  },
  {
    "symbol": "ETHEUR",
    "spread_abs": 0.01,
    "spread_rel": 0.0000043491
  },
  {
    "symbol": "BTCTRY",
    "spread_abs": 54,
    "spread_rel": 0
  },
  {
    "symbol": "BNBUSDC",
```


Compare summary endpoint


mnagy-de3-ha-api-lb-1218970277.eu-west-1.elb.amazonaws.com/compare_summary?symbol1=BTCUSDT&symbol2=ETHUSDT&interval=1m&lookback=360

Price comparison of selected symbols

Generated at: 2025-05-13

Welcome to the price comparison site!

You are now comparing the price evolution of BTCUSDT and ETHUSDT.

Descriptive statistics

Statistics about BTCUSDT

In original units:

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
102890	103434	103561	103539	103693	103935

In relative terms (100=starting period):

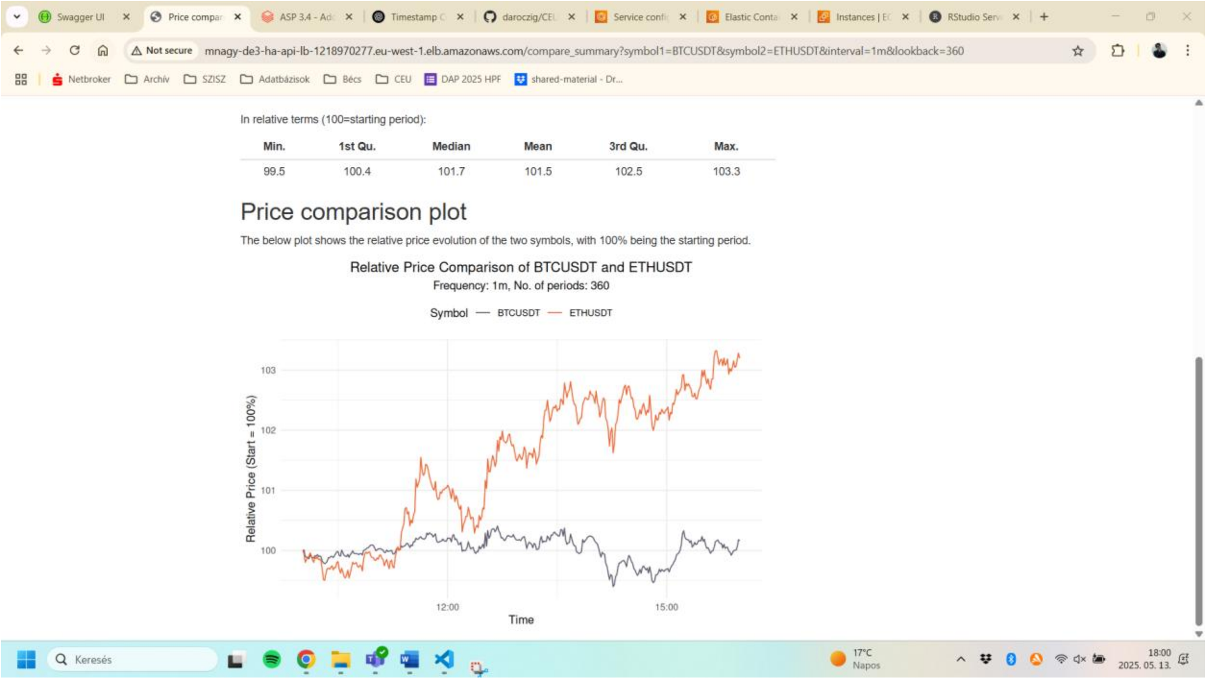
Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
99.4	99.92	100	100	100.2	100.4

Statistics about ETHUSDT:

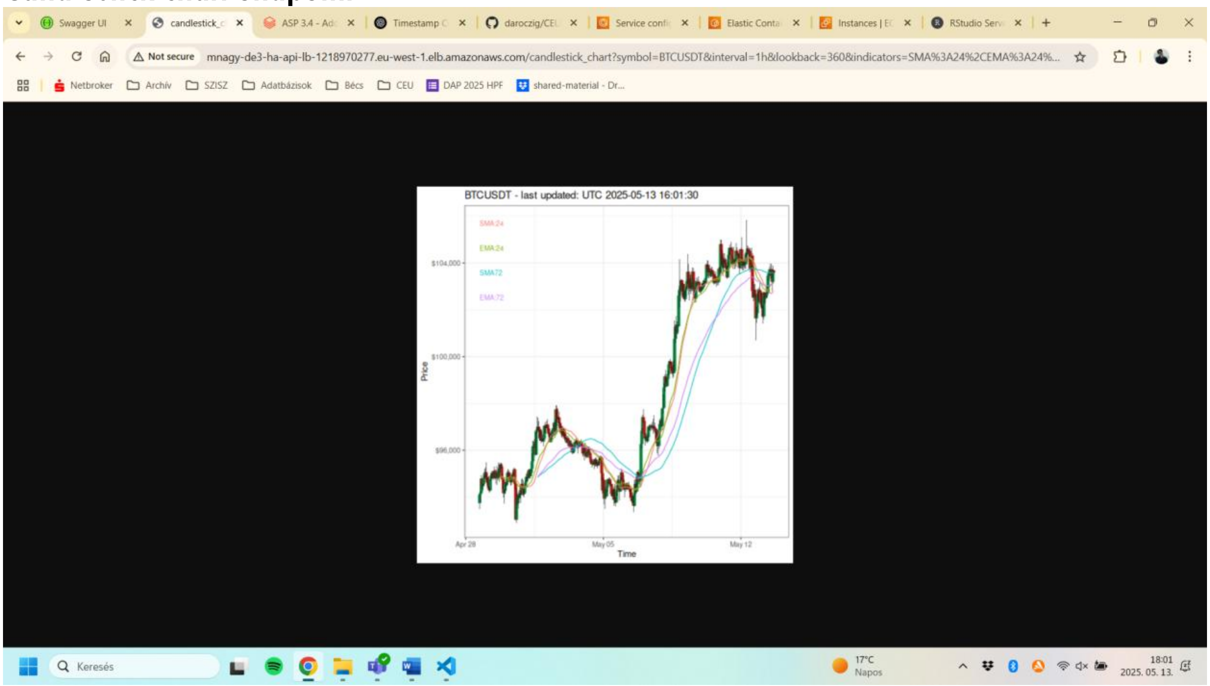
In original units:

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
2474	2497	2528	2523	2547	2569

In relative terms (100=starting period):

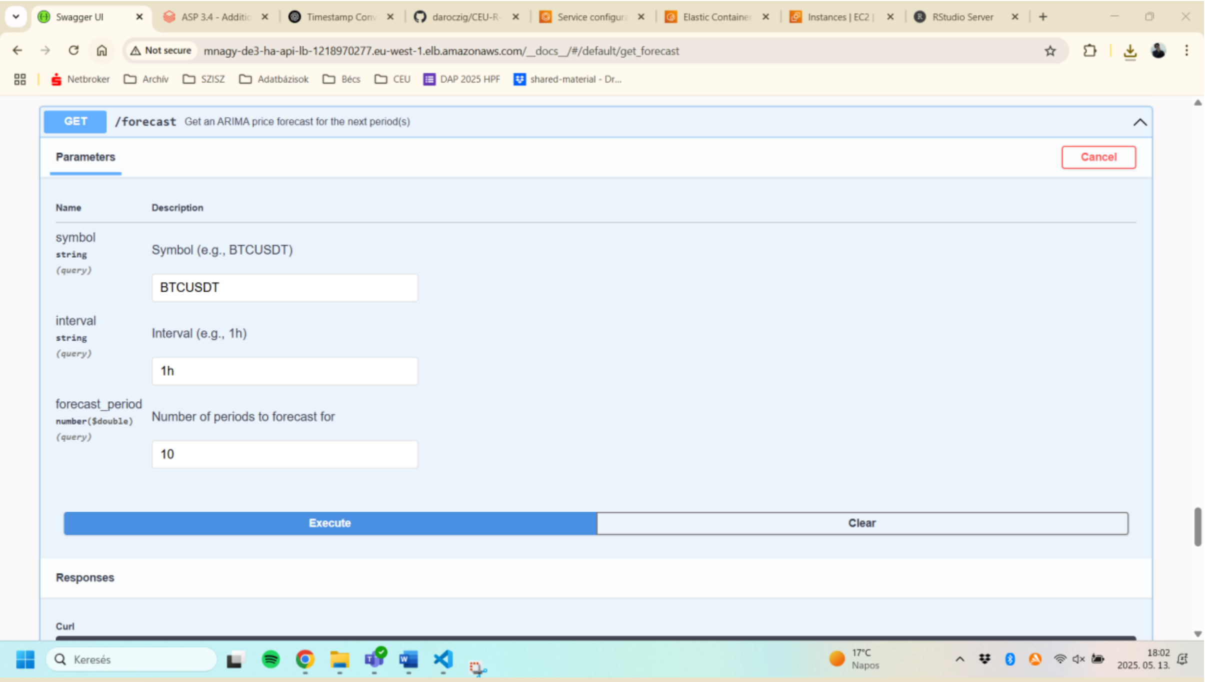


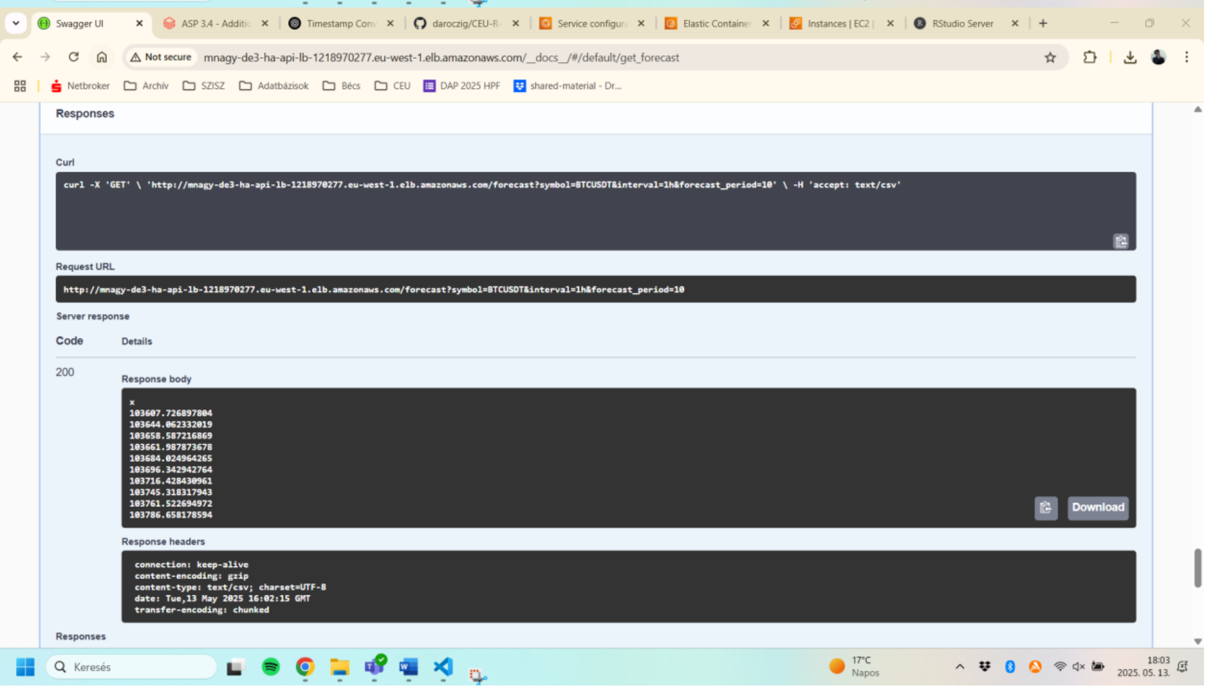
Candlestick chart endpoint



Forecast endpoint (as this returns a CSV which is not rendered in the browser, but downloaded, I show the result on the Swagger site)

Author: Márton Nagy
Student ID: 2402779
Instance ID: i-0657dc34b986263aa (mnagy_de3_home_assignment)

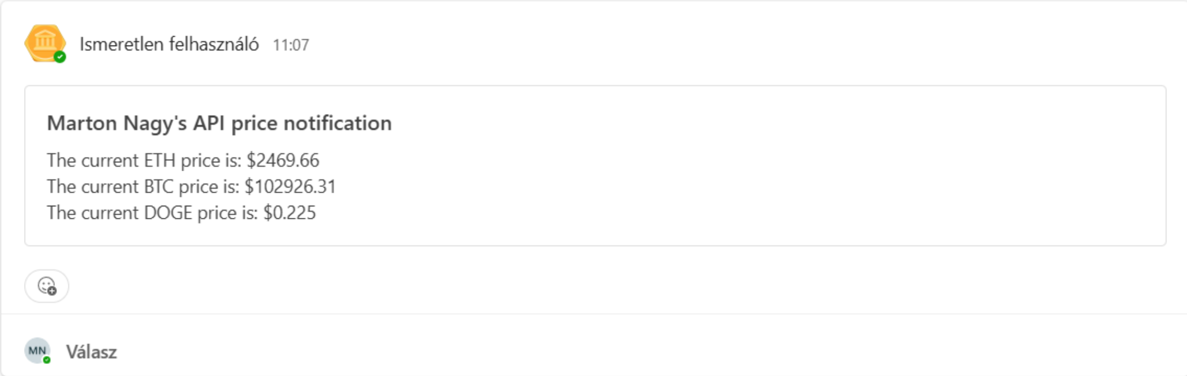




After confirming that the API could work, I deleted everything on ECS and ECR and stopped my EC2 instance (but not terminated it). I also cleaned up artifacts created by me during the class. Note that my keypair has not been deleted as it is used by my EC2 instance. This concludes my final assignment for the Data Engineering 3 class.

A bit of extra that did not work out…

Although the above presented solution should satisfy everything required by this assignment, I tried to implement a fifth endpoint. The goal of this would have been to have an endpoint that takes a list of symbols and sends a Teams message with the current USD prices. I could successfully implement this, and it worked fine when I ran the API from my EC2 instance (as you may see from the respective messages on Teams).



However, after dockerization, the API was either unable to get my webhook URL from the parameter store, or unable to send the message to Teams. I believe this may have been issue with access policies or ports, which I could not resolve. So, I would greatly appreciate if you could provide some inputs on how this endpoint could have been successfully dockerized and deployed. For reference, the code for this is below. Nonetheless, this is not an integral part of my submission and is only included out of curiosity.

Teams message endpoint R code

```
library(botor)
Library(teamr)

#--------------------
# 5. Send Teams report
#--------------------
#* Send a Teams notification with the current USD price of the given symbols
#* @param symbols Comma separated list of symbols (e.g., BTC,ETH)
#* @get /send_teams_report
function(symbols = "BTC,ETH") {
  symbols_list <- strsplit(trimws(symbols), ",")[[1]]
  msg <- ""
  prices <- binance_coins_prices()
  for (symbol in symbols_list) {
    sym <- trimws(symbol)
    msg <- paste0(msg, 'The current ', sym, ' price is: $', prices[symbol == 
sym, usd], '<br>')
  }

  botor(region = 'eu-west-1')
  webhook <- ssm_get_parameter('/teams/mnagy')

  cc <- connector_card$new(hookurl = webhook)
  cc$title("Marton Nagy's API price notification")
  cc$text(msg)
  cc$send()
```