

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

```

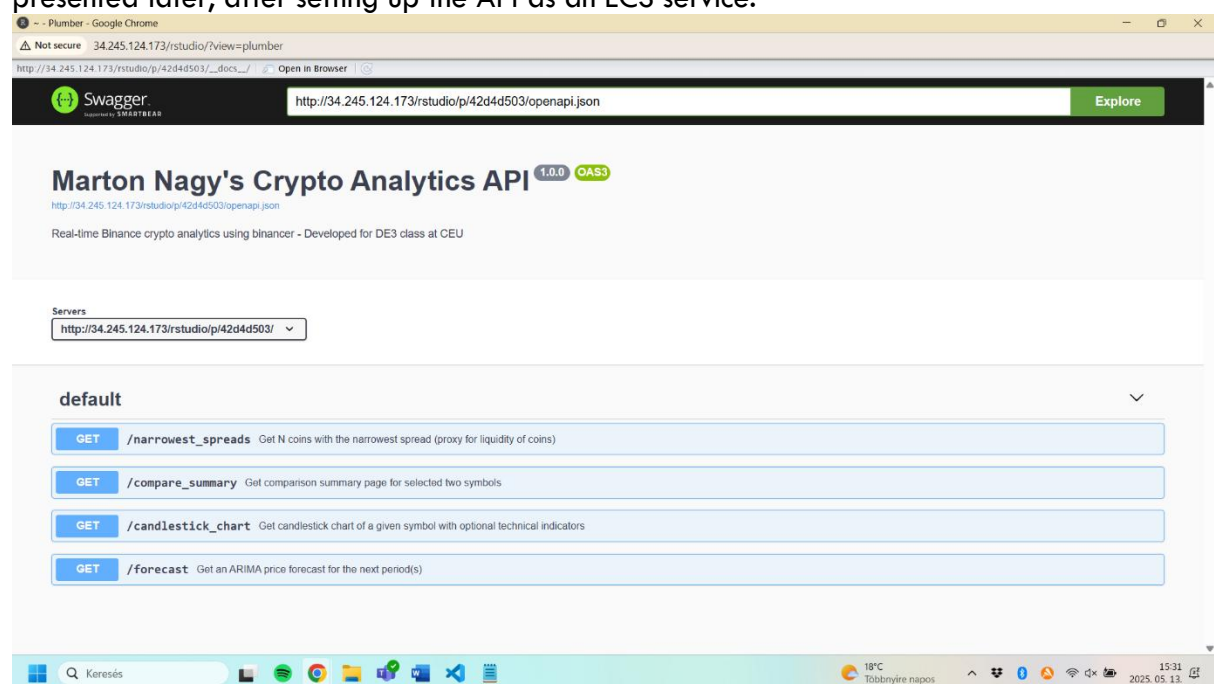
#-----
4. Forecast with ARIMA
#-----
#* Get an ARIMA price forecast for the next period(s)
#* @param symbol Symbol (e.g., BTCUSDT)
#* @param interval Interval (e.g., 1h)
#* @param forecast_period Number of periods to forecast for
#* @serializer csv
#* @get /forecast
function(symbol = "BTCUSDT", interval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

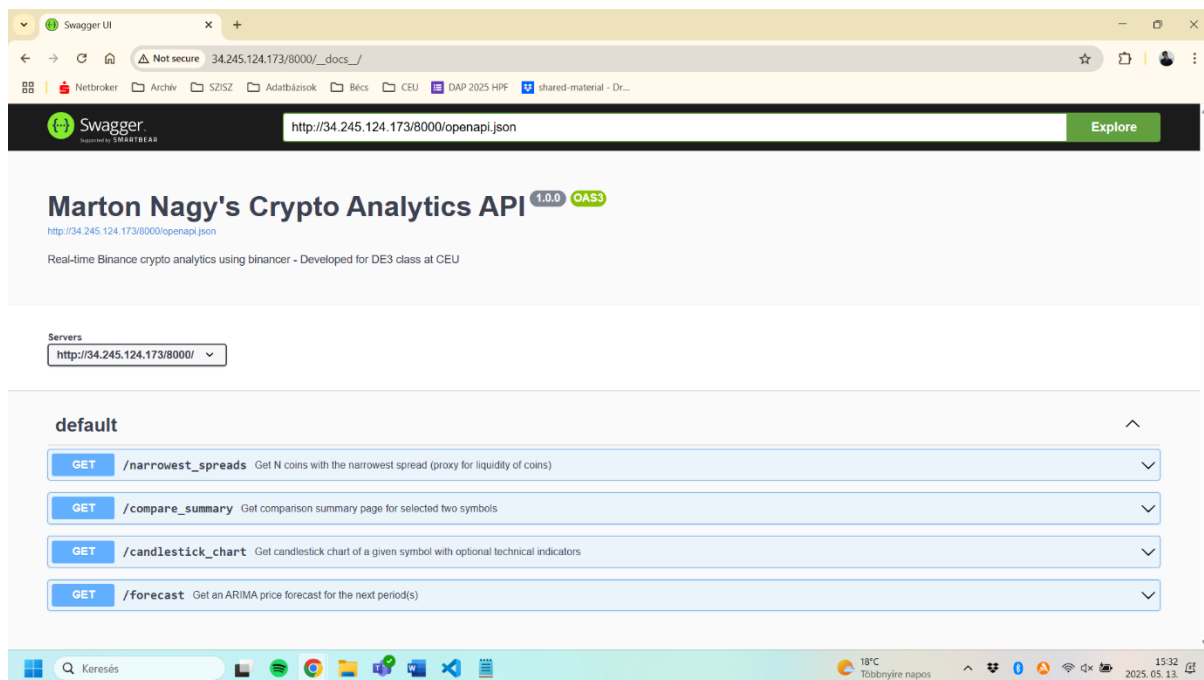
```
RUN apt-get update && apt-get install -y pandoc && apt-get clean && rm -rf /var/lib/apt/lists/
RUN install2.r ggplot2 quantmod forecast rmarkdown dplyr pander readr
RUN installGithub.r daroczig/binancer
ADD price_comparison_report.Rmd /app/price_comparison_report.Rmd
ADD mnagy_de3_ha_api.R /app/mnag_de3_ha_api.R
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/mnag_de3_ha_api:latest
```

```
sudo docker push 657609838022.dkr.ecr.eu-west-1.amazonaws.com/mnag_de3_ha_api:latest
```

**General settings**

**Repository name**  
Enter a concise name. Repositories support namespaces, which you can use to group similar repositories.  
657609838022.dkr.ecr.eu-west-1.amazonaws.com/**mnagy-de3-ha-api**  
16 out of 256 characters maximum (2 minimum). The name must start with a letter and can only contain lowercase letters, numbers, and special characters \_-./.

**Image tag mutability**  
Choose the tag mutability setting.  
☒ **Mutable**  
Image tags can be overwritten.  
☐ **Immutable**  
Image tags can't be overwritten.

**Encryption settings** [Info](#)

The encryption settings for a repository can't be changed once the repository is created.

**Encryption configuration**  
By default, repositories use the industry standard Advanced Encryption Standard (AES) encryption. You can optionally choose to use a key stored in the AWS Key Management Service (KMS) to encrypt the images in your repository.  
☒ **AES-256**  
Industry standard Advanced Encryption Standard (AES) encryption  
☐ **AWS KMS**  
AWS Key Management Service (KMS)

### 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.

**Create new task definition** [Info](#)

**Task definition configuration**

**Task definition family** [Info](#)  
Specify a unique task definition family name.  
**mnagy-de3-ha-api**  
Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed.

**Infrastructure requirements**  
Specify the infrastructure requirements for the task definition.

**Launch type** [Info](#)  
Selection of the launch type will change task definition parameters.  
☒ **AWS Fargate**  
Serverless compute for containers.  
☐ **Amazon EC2 instances**  
Self-managed infrastructure using Amazon EC2 instances.

**OS, Architecture, Network mode**  
Network mode is used for tasks and is dependent on the compute type selected.  
**Operating system/Architecture** [Info](#)  
Linux/x86\_64  
**Network mode** [Info](#)  
awsvpc

**Task size** [Info](#)  
Specify the amount of CPU and memory to reserve for your task.  
**CPU**  
1 vCPU  
**Memory**  
3 GB

**Task roles - conditional**  
**Task placement - optional**  
**Fault injection - optional**

**Container - 1** [Info](#)  
Essential container [Remove](#)  
Container details



Author: Márton Nagy  
Student ID: 2402779  
Instance ID: i-0657dc34b986263aa (mnagy\_de3\_home\_assignment)

The screenshot displays the AWS Management Console interface for creating a new task definition. The top navigation bar shows the user is logged in as 'magn\_marton@student.cau.edu' with a user ID of '6576-0983-8022'. The left sidebar contains links to various AWS services, including Amazon Elastic Container Service, Amazon ECR, and AWS Batch. The main content area is titled 'Create new task definition' and is divided into two main sections: 'Container - 1' and 'Docker configuration - optional'.

**Container - 1**

- Container details:** This section includes fields for 'Name' (mnagy\_api), 'Image URI' (657609838022.dkr.ecr.eu-west-1.amazonaws.com/mnagy-de3-ha-apl:latest), and 'Essential container' (Yes).
- Private registry:** This section includes a checkbox for 'Private registry authentication' and a field for 'Private registry authentication'.
- Port mappings:** This section includes a table for 'Container port', 'Protocol', 'Port name', and 'App protocol'. The table has one row with '8000' for container port, 'TCP' for protocol, 'container-port-protocol' for port name, and 'HTTP' for app protocol.
- Read only root file system:** This section includes a checkbox for 'Read only'.
- Resource allocation limits - conditional:** This section includes fields for 'CPU' (1), 'GPU' (1), 'Memory hard limit' (3), and 'Memory soft limit' (1).
- Environment variables - optional:** This section includes a field for 'Environment variables'.
- Logging - optional:** This section includes a field for 'Logging'.
- Restart policy - optional:** This section includes a field for 'Restart policy'.
- HealthCheck - optional:** This section includes a field for 'HealthCheck'.
- Startup dependency ordering - optional:** This section includes a field for 'Startup dependency ordering'.

**Docker configuration - optional**

- Container timeouts - optional:** This section includes a field for 'Container timeouts'.
- Container network settings - optional:** This section includes a field for 'Container network settings'.
- Docker configuration - optional:** This section includes fields for 'Entry point' (command defined: sh -c), 'Command' (mnagy\_de3\_ha\_api.R), and 'Working directory' (/usr/app).
- Resource limits (limits) - optional:** This section includes a field for 'Resource limits (limits)'.
- Docker labels - optional:** This section includes a field for 'Docker labels'.
- Storage - optional:** This section includes a field for 'Storage'.
- Monitoring - optional:** This section includes a field for 'Monitoring'.
- Tags - optional:** This section includes a field for 'Tags'.

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

eu-west-1.console.aws.amazon.com/ecs/v2/create-cluster?region=eu-west-1

Amazon Elastic Container Service > Create cluster

Cluster configuration

Cluster name  
MNAGY\_API  
Cluster name must be 1 to 255 characters. Valid characters are a-z, A-Z, 0-9, hyphens (-), and underscores (\_).

Service Connect defaults - optional

Infrastructure - optional [info](#)

Your cluster is automatically configured for AWS Fargate (serverless) with two capacity providers. Add Amazon EC2 instances.

☒ AWS Fargate (serverless)  
Pay as you go. Use if you have tiny, batch, or burst workloads or for zero maintenance overhead. The cluster has Fargate and Fargate Spot capacity providers by default.

☐ Amazon EC2 instances  
Manual configurations. Use for large workloads with consistent resource demands.

☐ External instances using ECS Anywhere can be registered after cluster creation is complete.

Monitoring - optional [info](#)

CloudWatch Container Insights is a monitoring and troubleshooting solution for containerized applications and microservices.

Encryption - optional

Choose the KMS keys used by tasks running in this cluster to encrypt your storage.

Tags - optional [info](#)

Tags help you to identify and organize your clusters.

☒ Account settings for Resource Tagging Authorization are currently turned on.  
The ecs:TagResource Action is required to tag ECS resources.

Key  
Class

Value - optional  
DE3

Remove

Lastly, a service was created using the below inputs.

eu-west-1.console.aws.amazon.com/ecs/v2/clusters/MNAGY\_API/create-service?region=eu-west-1

Amazon Elastic Container Service > Clusters > MNAGY\_API > Create service

Create service [info](#)

Service details

Task definition family  
Select an existing task definition family. To create a new task definition, go to [Task definitions](#).

mnagy\_de3\_ha\_api

Task definition revision [LATEST](#)  
Select the task definition revision from the 100 most recent entries, or enter a revision. Leave the field blank to use the latest revision.

1

Service name  
Assign a service name that is unique for this cluster.  
Up to 255 letters (uppercase and lowercase), numbers, underscores, and hyphens are allowed. Service names must be unique within a cluster.

mnagy\_de3\_ha\_api

Environment

Existing cluster  
MNAGY\_API

Compute configuration (advanced)

Compute options [info](#)

To ensure task distribution across your compute types, use appropriate compute options.

☐ Capacity provider strategy  
Specify a launch strategy to distribute your tasks across one or more capacity providers.

☒ Launch type  
Launch tasks directly without the use of a capacity provider strategy.

Launch type [info](#)

Select either managed capacity (Fargate), or custom capacity (EC2 or user-managed). External instances are registered to your cluster using the ECS Anywhere capability.

FARGATE

Platform version [info](#)

Select the platform version on which to run your service.

LATEST

ASP 3.4 - Additional x Timestamp Convers x daroczig/CEU-R-pro x Create service | Elastic Container Re x Instances | EC2 | eu- x RStudio Server x + -

eu-west-1.console.aws.amazon.com/ecs/v2/clusters/MNAGY\_API/create-service?region=eu-west-1

Netbroker Archiv SZISZ Adatbázisok Bács CEU DAP 2025 HPF shared-material - Dr...

Amazon Elastic Container Service

Clusters  
Namespaces  
Task definitions  
Account settings

Install AWS Copilot  
Amazon ECR  
Repositories  
AWS Batch  
Documentation  
Discover products  
Subscriptions

Tell us what you think

Deployment configuration

Service type | info  
Specify the service type that the service scheduler will follow.

☒ Replica  
Place and maintain a desired number of tasks across your cluster.

☐ Daemon  
Place and maintain one copy of your task on each container instance.

Desired tasks  
Specify the number of tasks to launch.

1

Availability Zone rebalancing | info  
☒ Turn on Availability Zone rebalancing  
Amazon ECS automatically detects Availability Zone imbalances in task distributions across an ECS service, and evenly redistributes ECS service tasks across Availability Zones.

Health check grace period | info  
10  
seconds

Deployment options  
Deployment failure detection | info

Networking

VPC | info  
Select a VPC to use for your Amazon ECS resources.

vpc-06f51030f2b9ca295  
default

Create a new VPC

Subnets  
Choose the subnets within the VPC that the task scheduler should consider for placement.

Choose subnets

subnet-03f136e4570074f92 X subnet-0c7729077af98b08 X subnet-0205cf984be4545af X  
eu-west-1a 172.31.32.0/20 eu-west-1a 172.31.16.0/20 eu-west-1c 172.31.0.0/20

Clear current selection

Security group | info  
Choose an existing security group or create a new security group.

Use an existing security group  
Create a new security group

Security group name  
Choose an existing security group.

sg-06a256ff0b1979d63  
default

Public IP | info  
Choose whether to auto-assign a public IP to the task's elastic network interface (ENI).

☒ Turned on

Service Connect - optional | info  
Service Connect allows for service-to-service communications with automatic discovery using short names and standard ports.

Service discovery - optional | info  
Service discovery uses Amazon Route 53 to create a namespace for your service, which allows it to be discoverable via DNS.

Load balancing - optional | info  
Configure load balancing using Amazon Elastic Load Balancing to distribute traffic evenly across the healthy tasks in your service.

☒ Use load balancing

VPC  
The VPC for your load balancing resources must be the same as the VPC for your service with awscli.

vpc-06f51030f2b9ca295

Load balancer type | info  
Specify the load balancer type to distribute incoming traffic across the tasks running in your service.

☒ Application Load Balancer  
An Application Load Balancer makes routing decisions at the application layer (HTTP/HTTPS), supports path-based routing, and can route requests to one or more ports.

☐ Network Load Balancer  
A Network Load Balancer makes routing decisions at the transport layer (TCP/UDP).

Container  
This section is not visible in the screenshot.

CloudShell Feedback

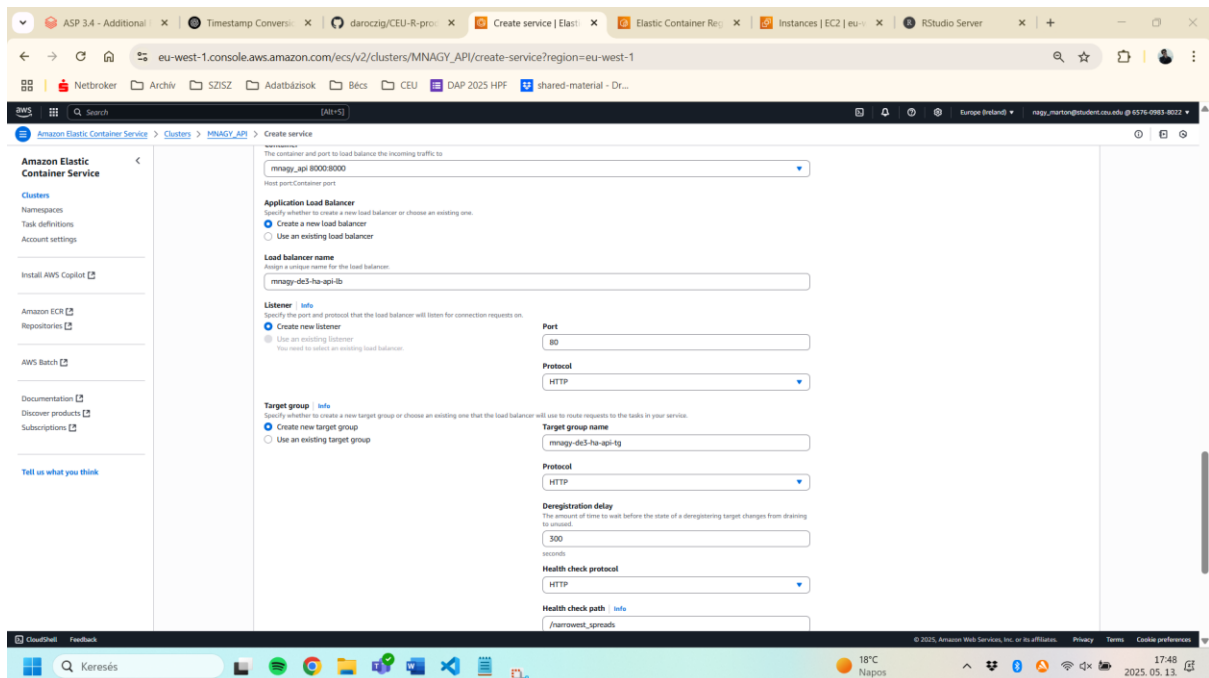
Keresés

18°C  
Napos

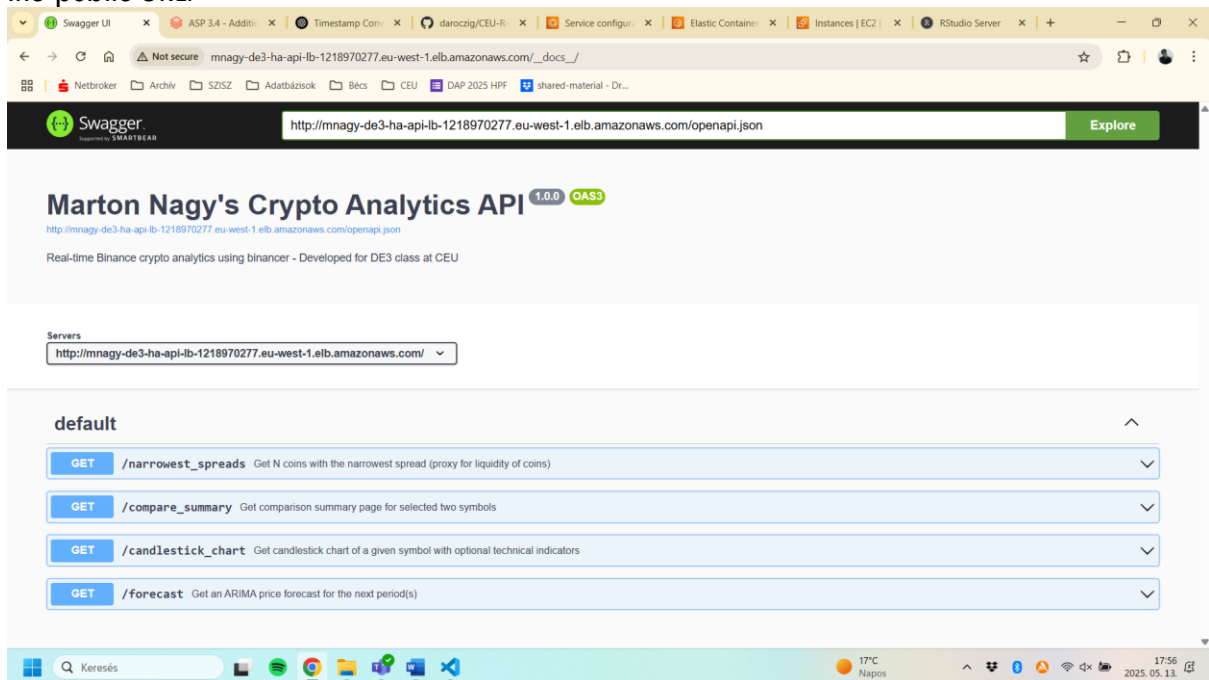
17:47  
2025. 05. 13.

© 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Console preferences

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

Swagger UI | mnagy-de3- | ASP 3.4 - Ad- | Timestamp | daroczig/CE- | Service confi- | Elastic Cont- | Instances | E- | RStudio Serv- | + | - | X

Not secure mnagy-de3-ha-api-lb-1218970277.eu-west-1.elb.amazonaws.com/narrowest\_spreads?n=10

Netbroker | Archiv | SZISZ | Adatbázisok | Bécs | CEU | DAP 2025 HPF | shared-material - Dr...

Pretty-print

```
[
 {
 "symbol": "BTCUSDC",
 "spread_abs": 0.01,
 "spread_rel": 9.641e-8
 },
 {
 "symbol": "BTCUSD",
 "spread_abs": 0.01,
 "spread_rel": 9.6419e-8
 },
 {
 "symbol": "WBTCUSD",
 "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": "ETHUSD",
 "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",
 "spread_abs": 0.01,
 "spread_rel": 0.01
 }
]
```

Keresés 17°C Napos 17:59 2025. 05. 13.

Swagger UI | Price compa- | ASP 3.4 - Ad- | Timestamp | daroczig/CE- | Service confi- | Elastic Cont- | Instances | E- | RStudio Serv- | + | - | X

Not secure mnagy-de3-ha-api-lb-1218970277.eu-west-1.elb.amazonaws.com/compare\_summary?symbol1=BTCUSDT&symbol2=ETHUSD&interval=1m&lookback=360

Netbroker | Archiv | SZISZ | Adatbázisok | Bécs | CEU | DAP 2025 HPF | shared-material - Dr...

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 ETHUSD.

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

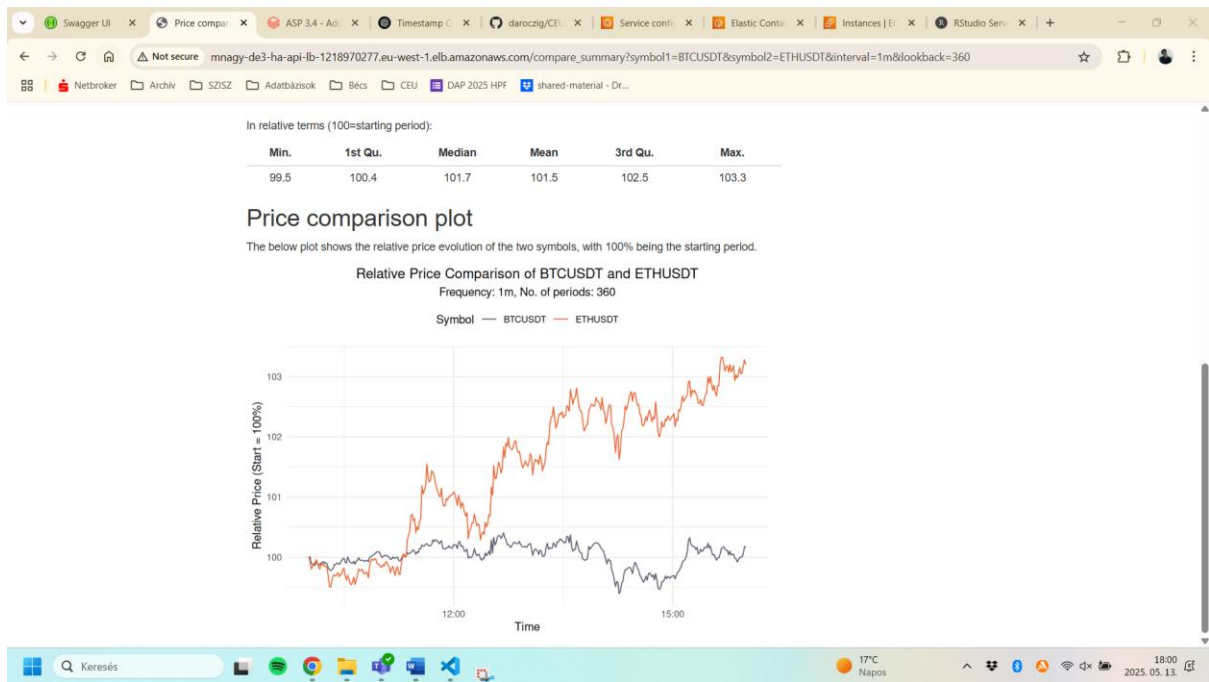
Statistics about ETHUSD:

In original units:

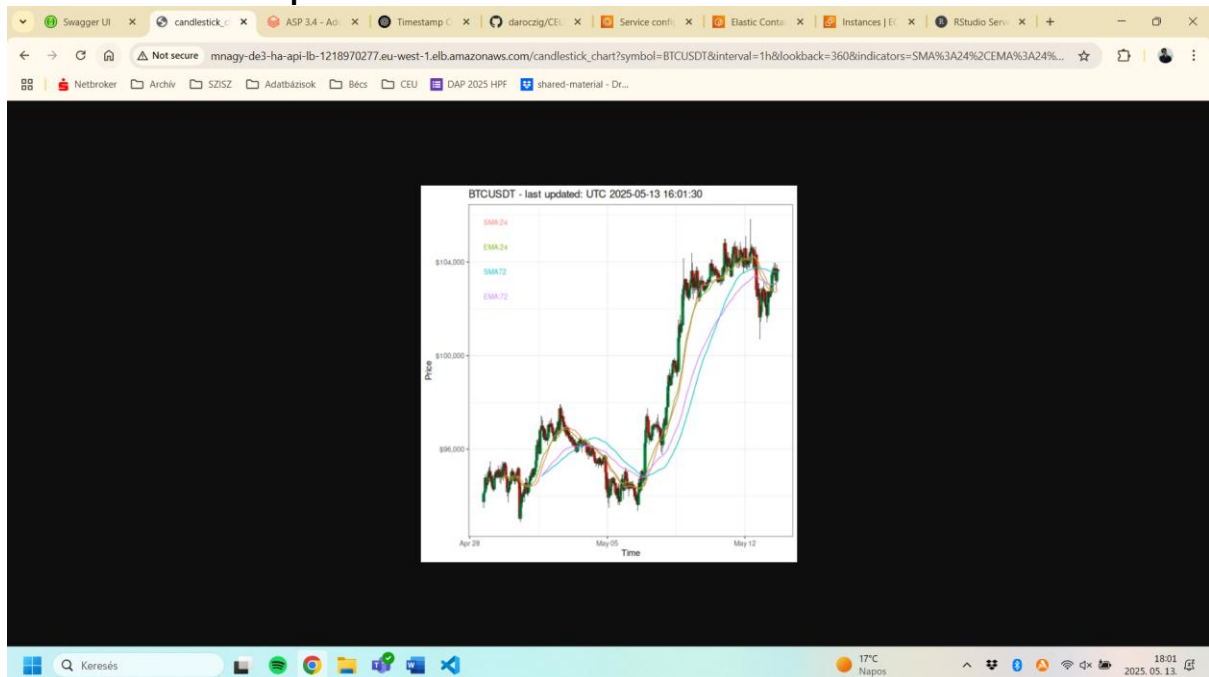
Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
2474	2497	2528	2523	2547	2569

In relative terms (100=starting period):

Keresés 17°C Napos 18:00 2025. 05. 13.



## 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)

The top screenshot shows the Swagger UI for the `/forecast` endpoint. The parameters are:

- `symbol` (string, query): Symbol (e.g., BTCUSDT) - Value: `BTCUSDT`
- `interval` (string, query): Interval (e.g., 1h) - Value: `1h`
- `forecast_period` (number, query): Number of periods to forecast for - Value: `10`

The bottom screenshot shows the response for the `/forecast` endpoint. The response is a CSV file with 10 data points.

Request URL: `http://mnagy-de3-ha-api-lb-1218970277.eu-west-1.elb.amazonaws.com/forecast?symbol=BTCUSDT&interval=1h&forecast_period=10`

Server response

Code: 200

Response body

```
x
103607.726897804
103604.062312019
103555.507216809
103661.987873678
103684.024964265
103696.343942764
103716.428430961
103745.318317943
103761.522094972
103781.658178594
```

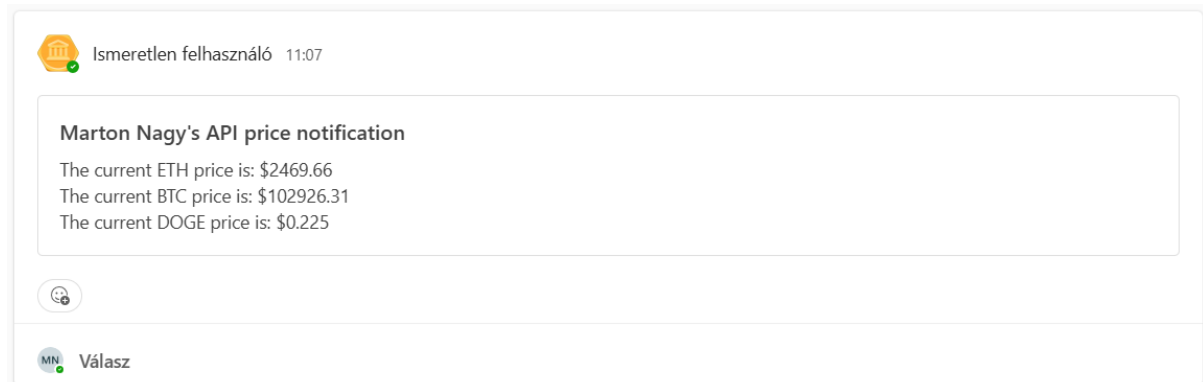
Response headers

```
connection: keep-alive
content-encoding: gzip
content-type: text/csv; charset=UTF-8
date: Tue, 13 May 2025 16:02:13 GMT
transfer-encoding: chunked
```

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], '
')
 }

 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()
```