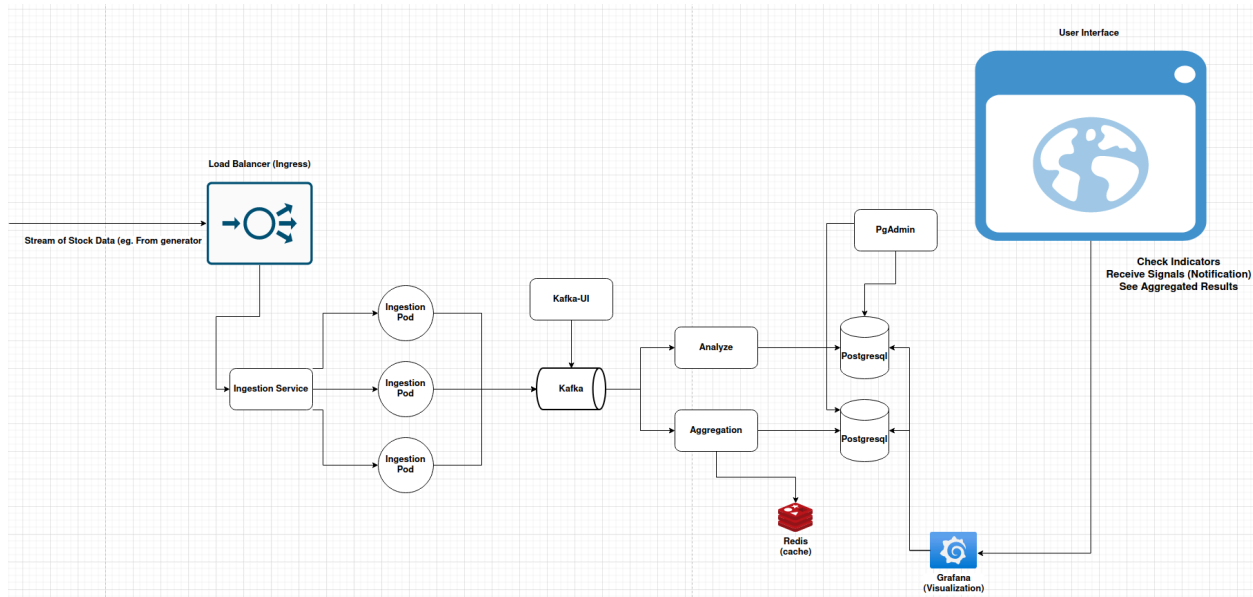


گزارش پروژه‌ی پایانی درس سیستم‌های توزیع‌شده

اعضای گروه: علی نظری 401725173 - امین شهبسوارخانی 401725898

مقدمه

معماری کلی پروژه‌ای که ما انجام دادیم به صورت زیر است. (تمامی تصاویر نیز ضمیمه شده‌اند)

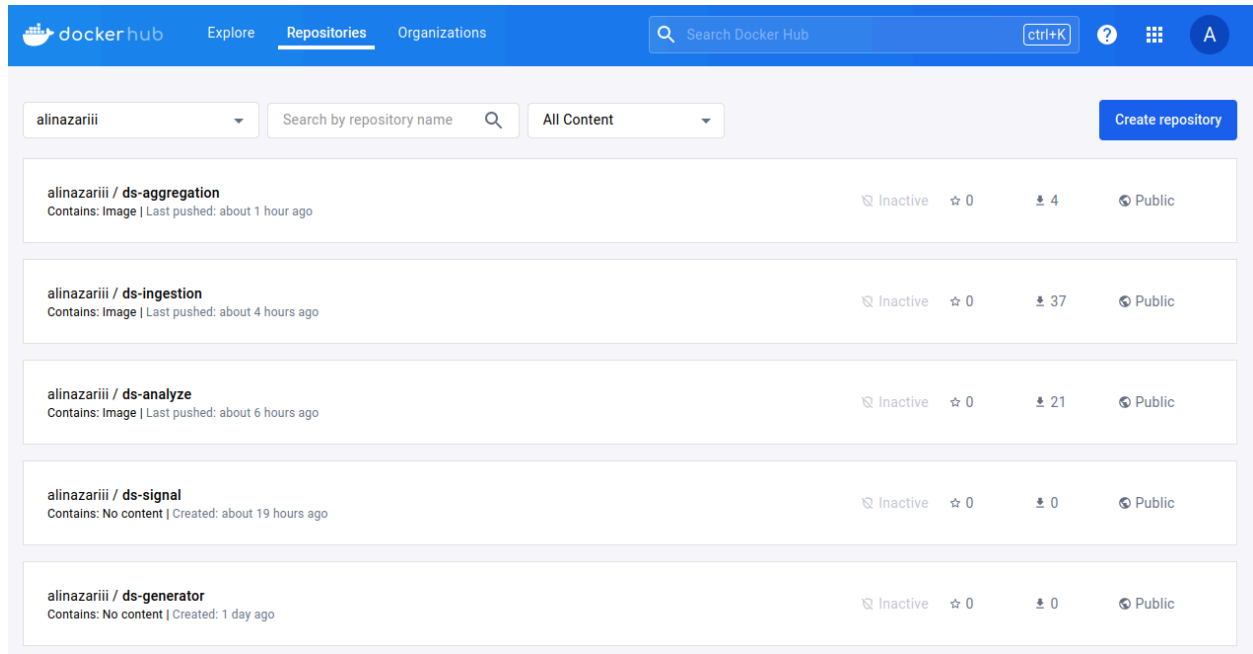


همان‌طور که در این تصویر مشخص است، داده‌ها توسط قطعه کدی که در پروژه قرار داده شده بود به سمت load-balancer می‌روند و داده‌ها توسط آن بین podهای مختلف سرویس ingestion برای پردازش پخش می‌شوند. سرویس ingestion وظیفه دریافت داده‌ها و چک کردن ساختار داده‌ی آن و پس از آن منتقل کردن آن درون سیستم پردازش stream ما را بر عهده دارد. این سرویس پس از انجام وظایف خود، داده‌ها را درون kafka می‌ریزد (produce). یک ابزار kafka-ui هم برای مشاهده‌ی داده‌هایی که درون kafka ریخته می‌شود به صورت جانبی بالا آمده که در دیباگ کردن سرویس به ما کمک می‌کند. این داده‌ها توسط ۲ سرویس مختلف یعنی analyze و aggregation خوانده (consume) می‌شوند. سرویس analyze وظیفه‌ی پردازش داده‌ها و محاسبه‌ی indicatorها را بر عهده دارد که با کمک آن‌ها به کاربر سیگنال‌هایی برای خرید و فروش داده می‌شود. سرویس aggregation هم وظیفه‌ی تجمیع داده‌ها و به دست‌آوردن چند متریک کلی و تجمعی را به عهده دارد. برای دیباگ کردن داده‌ها از pgAdmin استفاده شده و در قسمت aggregation برای دستیابی به سرعت بهتر از redis بهره بردیم. در نهایت برای visualization، داشبوردهایی ایجاد کردیم که قابلیت مشخص کردن بازه‌ی زمانی و مشاهده‌ی داده‌های مختلف را به کاربران می‌دهد که در ادامه بیشتر در مورد آن‌ها توضیح داده شده. سیگنال‌های خرید و فروش سهام هم که با کمک indicatorها به دست آمده در این قسمت داده می‌شود.

گزارش پروژه‌ی پایانی درس سیستم‌های توزیع‌شده

جزئیات سرویس‌ها و پیاده‌سازی

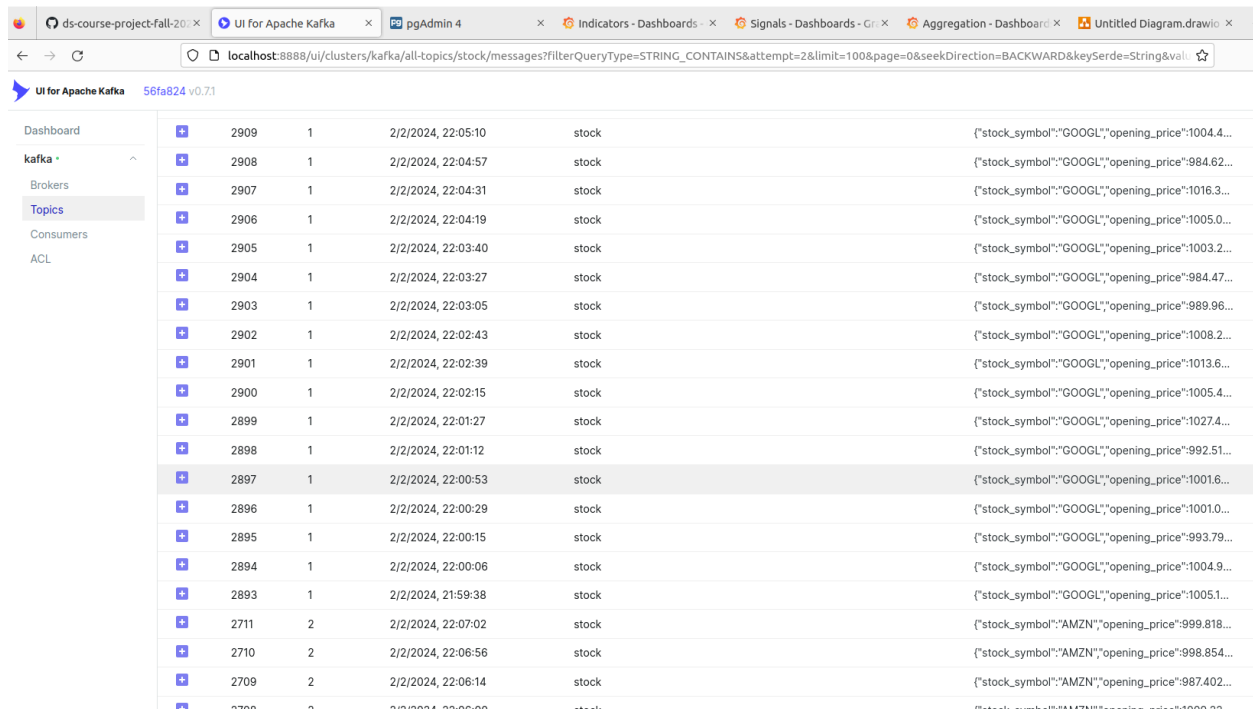
تمامی میکروسرویس‌های توسعه داده شده، به کمک Dockerfile موجود در هر پروژه dockerize شده‌اند و ایمج‌های آن‌ها در docker hub پوش شده.



میکروسرویس‌های مختلف به کمک deployment, service, configMap مجزا به فراخور نیاز در یک کلاستر کوبرنتیز minikube بالا آمده‌اند که امکان scale شدن را هم دارند و با هم در ارتباط اند. همان‌طور که در معماری کلی نیز دیده می‌شود، سرویس ingestion دارای چندین pod است که load-balancer ریکوئست‌ها را بین آن‌ها پخش می‌کند.

همان‌طور که بالاتر هم گفته شد، از kafka به عنوان ابزار real-time stream processing استفاده شده که داده‌های موجود در آن به کمک پنل زیر قابل مشاهده است که در دیباگ مشکلات و دیدن داده‌ها بسیار موثر بود. لازم به ذکر است که کافکا هم به کمک helm-chart که به نوعی package-managerهای کوبرنتیز هستند در همین کلاستر بالا آمده و قابل دسترسی است.

گزارش پروژه‌ی پایانی درس سیستم‌های توزیع‌شده



| | | | | | | |
|-----------|---|------|---|--------------------|-------|---|
| Dashboard | + | 2909 | 1 | 2/2/2024, 22:05:10 | stock | ("stock_symbol":"GOOGL","opening_price":1004.4... |
| kafka | + | 2908 | 1 | 2/2/2024, 22:04:57 | stock | ("stock_symbol":"GOOGL","opening_price":984.62... |
| Brokers | + | 2907 | 1 | 2/2/2024, 22:04:31 | stock | ("stock_symbol":"GOOGL","opening_price":1016.3... |
| Topics | + | 2906 | 1 | 2/2/2024, 22:04:19 | stock | ("stock_symbol":"GOOGL","opening_price":1005.0... |
| Consumers | + | 2905 | 1 | 2/2/2024, 22:03:40 | stock | ("stock_symbol":"GOOGL","opening_price":1003.2... |
| ACL | + | 2904 | 1 | 2/2/2024, 22:03:27 | stock | ("stock_symbol":"GOOGL","opening_price":984.47... |
| | + | 2903 | 1 | 2/2/2024, 22:03:05 | stock | ("stock_symbol":"GOOGL","opening_price":989.96... |
| | + | 2902 | 1 | 2/2/2024, 22:02:43 | stock | ("stock_symbol":"GOOGL","opening_price":1008.2... |
| | + | 2901 | 1 | 2/2/2024, 22:02:39 | stock | ("stock_symbol":"GOOGL","opening_price":1013.6... |
| | + | 2900 | 1 | 2/2/2024, 22:02:15 | stock | ("stock_symbol":"GOOGL","opening_price":1005.4... |
| | + | 2899 | 1 | 2/2/2024, 22:01:27 | stock | ("stock_symbol":"GOOGL","opening_price":1027.4... |
| | + | 2898 | 1 | 2/2/2024, 22:01:12 | stock | ("stock_symbol":"GOOGL","opening_price":992.51... |
| | + | 2897 | 1 | 2/2/2024, 22:00:53 | stock | ("stock_symbol":"GOOGL","opening_price":1001.6... |
| | + | 2896 | 1 | 2/2/2024, 22:00:29 | stock | ("stock_symbol":"GOOGL","opening_price":1001.0... |
| | + | 2895 | 1 | 2/2/2024, 22:00:15 | stock | ("stock_symbol":"GOOGL","opening_price":993.79... |
| | + | 2894 | 1 | 2/2/2024, 22:00:06 | stock | ("stock_symbol":"GOOGL","opening_price":1004.9... |
| | + | 2893 | 1 | 2/2/2024, 21:59:38 | stock | ("stock_symbol":"GOOGL","opening_price":1005.1... |
| | + | 2711 | 2 | 2/2/2024, 22:07:02 | stock | ("stock_symbol":"AMZN","opening_price":999.818... |
| | + | 2710 | 2 | 2/2/2024, 22:06:56 | stock | ("stock_symbol":"AMZN","opening_price":998.854... |
| | + | 2709 | 2 | 2/2/2024, 22:06:14 | stock | ("stock_symbol":"AMZN","opening_price":987.402... |
| | + | 2708 | 2 | 2/2/2024, 22:05:00 | stock | ("stock_symbol":"AMZN","opening_price":987.402... |

روند کار به این صورت است که وقتی داده‌ها از generator به سرویس ingestion می‌رسد، پس از یک بررسی اولیه و چک کردن اسکیمایها، داده‌ها داخل کافکا produce می‌شوند. برای هر کدام از سهام‌ها درون تایپیک stock موجود در کافکا، یک partition موجود است که ترتیب پردازش داده‌های هر کدام از سهام‌ها حفظ شود. همان‌طور که در تصویر بالا هم مشاهده می‌شود داده‌های GOOGL در پارتیشن ۱ و داده‌های AMZN در پارتیشن ۲ ریخته شده‌اند.

```
func main() {
    logger.InitLogger()

    configs := config.LoadConfigs(pkg.IsLocalEnv())
    fmt.Println(configs)

    kafkaProducer := kafka.NewKafkaProducer(configs.Kafka.Hosts, configs.Kafka.Timeout)

    ingestionService := service.NewIngestionService(kafkaProducer)

    ingestionAPI := api.NewIngestionAPI(ingestionService)
    ingestionAPI.Start()
}
```

گزارش پروژه‌ی پایانی درس سیستم‌های توزیع‌شده

داده‌های موجود در این تایپیک کافکا توسط دو سرویس analyze و aggregation به صورت بلادرنگ consume می‌شوند. سرویس analyze هدفش محاسبه‌ی indicatorهای مختلف است که در این پروژه Moving Average، Exponential Moving Average و Relative Strength Index به صورت پیوسته اندازه‌گیری می‌شود. این خروجی‌ها داخل یک جدول دیتابیس postgresql ریخته می‌شود. برای نمایش این داده‌ها، گرافانا به این دیتابیس وصل شده که وضعیت هر کدام از سهام‌های مختلف از طریق آن قابل مشاهده است. این indicatorهایی که سرویس analyze تولید می‌کند، به صورت لحظه به لحظه در اینجا به صورت زیر قابل نمایش است که می‌توان از بالا، سهام‌های مختلف را انتخاب کرد و بازه‌ی زمانی را هم تعیین کرد. این داده‌ها ۵ ثانیه یک بار به صورت خودکار به روز می‌شوند. فایل json تولید این داشبوردها در گرافانا به ضمیمه ارسال شده‌اند.



سیگنال‌های خرید و فروشی بر اساس indicatorهای تولید شده در سرویس analyzer، به کاربران در هر لحظه داده می‌شود. این سیگنال‌ها بر اساس RSI کار می‌کنند که اگر مقدار آن کمتر از ۳۰ شود سیگنال خرید و اگر مقدار آن بیشتر از ۷۰ شود سیگنال فروش را ایجاد می‌کنند. زمان و نام هر کدام از سهام‌ها در نمودار این سیگنال‌ها مشخص شده و به صورت پیوسته آپدیت می‌شوند.

گزارش پروژه‌ی پایانی درس سیستم‌های توزیع‌شده

| Signals | | | |
|---------|---------------------|--------|---------------------|
| Buy | | Sell | |
| symbol | time | symbol | time |
| GOOGL | 2024-02-02 22:05:09 | AMZN | 2024-02-02 22:05:59 |
| TSLA | 2024-02-02 22:03:28 | AAPL | 2024-02-02 22:05:38 |
| AMZN | 2024-02-02 22:02:34 | AAPL | 2024-02-02 22:04:48 |
| TSLA | 2024-02-02 22:02:01 | AMZN | 2024-02-02 22:04:26 |
| AMZN | 2024-02-02 22:01:29 | GOOGL | 2024-02-02 22:04:18 |
| MSFT | 2024-02-02 22:00:43 | MSFT | 2024-02-02 22:04:02 |
| AAPL | 2024-02-02 22:00:17 | AAPL | 2024-02-02 22:03:16 |
| GOOGL | 2024-02-02 22:00:05 | AAPL | 2024-02-02 22:01:55 |
| TSLA | 2024-02-02 21:59:49 | MSFT | 2024-02-02 22:01:50 |
| AAPL | 2024-02-02 21:59:42 | AAPL | 2024-02-02 22:01:32 |

به صورت کلی سرویس analyze به صورت زیر کار می‌کند:

```
func main() {  Ali Nazari
    logger.InitLogger()

    configs := config.LoadConfigs(pkg.IsLocalEnv())
    fmt.Println(configs)

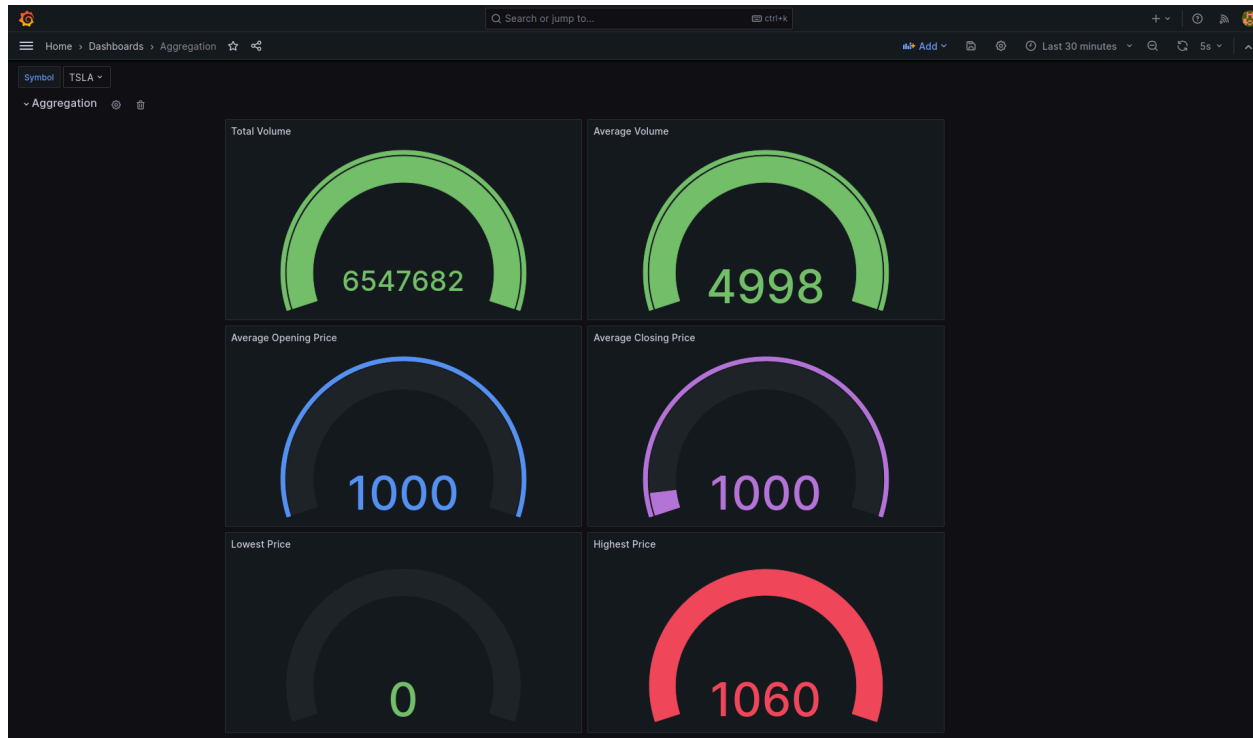
    db := sqlx.NewDB(configs.Postgresql)
    stockAnalysisRepo := repository.NewStockAnalysisRepo(db)

    consumers := make([]broker.Consumer, 0)
    for _, partition := range kafka.StockDataSymbolToPartition {
        kafkaConfigs := configs.Kafka
        kafkaConfigs.Partition = partition
        consumers = append(consumers, kafka.NewKafkaConsumer(kafkaConfigs))
    }

    analyzeService := service.NewAnalyzeService(consumers, stockAnalysisRepo)
    analyzeService.Start()
}
```

گزارش پروژه‌ی پایانی درس سیستم‌های توزیع‌شده

سرویس aggregation هم با کانسیوم داده‌های موجود در کافکا کار می‌کند و اطلاعات مختلفی هم‌چون Total Volume، Average Volume، Average Opening Price، Average Closing Price، Lowest Price و Highest Price را محاسبه می‌کند. این داده‌ها درون دیتابیس postgresql برای نمایش به صورت زیر ذخیره می‌شوند. در این نمودار امتحان مشخص کردن سهام مورد نظر و بازه زمانی فراهم شده است و به صورت خودکار هم هر ۵ ثانیه به روز می‌شود.



این سرویس برای محاسبه سریع‌تر نتایج از ردیس استفاده می‌کند. به این صورت که برخی اطلاعات میانی برای محاسبه‌ی این نتایج را در ردیس نگهداری می‌کند و در زمان بالا آمدن سرویس آن‌ها را از ردیس می‌خواند تا نیازی نباشد تا کل داده‌ها را از پستگرس بخواند و از اول آن‌ها محاسبه کند.

تصویر زیر یک نمای کلی از نحوه‌ی عملکرد این سرویس را نشان می‌دهد:

گزارش پروژه‌ی پایانی درس سیستم‌های توزیع‌شده

```
func main() {  Ali Nazari
    logger.InitLogger()

    configs := config.LoadConfigs(pkg.IsLocalEnv())
    fmt.Println(configs)

    db := sqlx.NewDB(configs.Postgresql)
    aggregatedMetricsRepo := repository.NewAggregatedMetricsRepo(db)

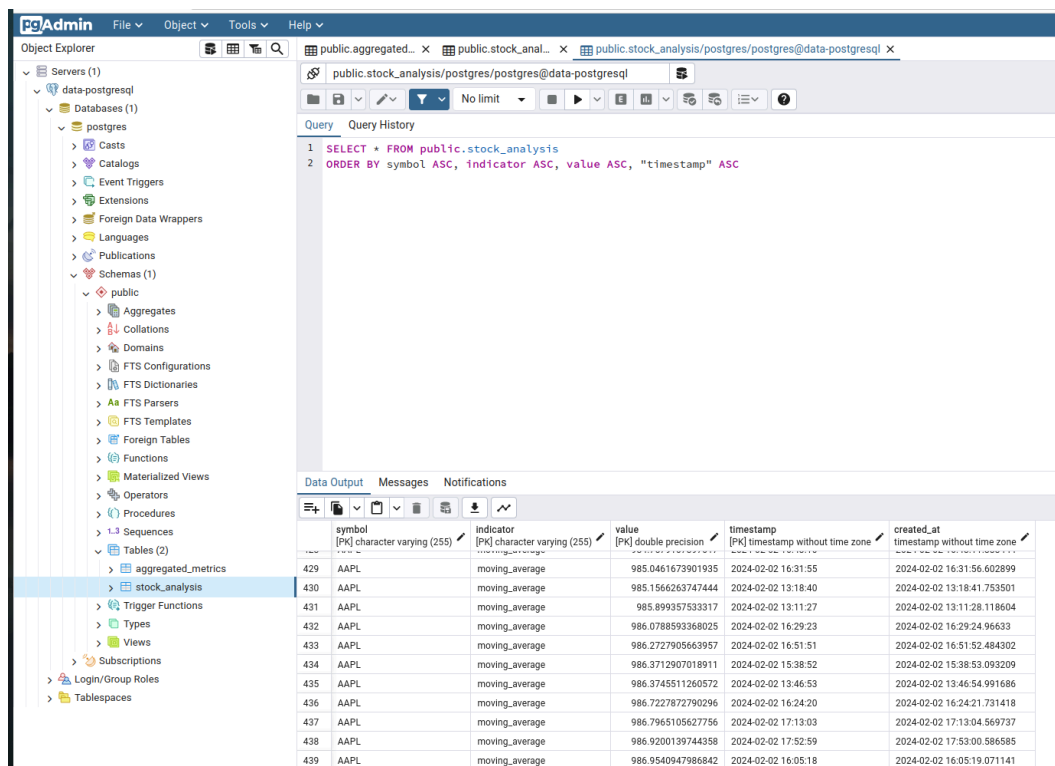
    consumers := make([]broker.Consumer, 0)
    for _, partition := range kafka.StockDataSymbolToPartition {
        kafkaConfigs := configs.Kafka
        kafkaConfigs.Partition = partition
        consumers = append(consumers, kafka.NewKafkaConsumer(kafkaConfigs))
    }

    redisClient := redis.NewRedisClient(configs.Redis)

    metricsCache := cache.NewMetricsCache(redisClient)

    aggregationService := service.NewAggregationService(consumers, aggregatedMetricsRepo, metricsCache)
    aggregationService.Start()
}
```

همان‌طور که پیش‌تر هم گفته شد، برای دیدن و دیباگ داده‌های موجود در پستگرس هم از ابزار pgAdmin استفاده کردیم که در تصویر زیر مشخص است.



The screenshot shows the pgAdmin interface with a query executed in the 'public.stock_analysis/postgres/postgres@data-postgresql' database. The query is:

```
1 SELECT * FROM public.stock_analysis
2 ORDER BY symbol ASC, indicator ASC, value ASC, "timestamp" ASC
```

The result is displayed in a table with the following columns: symbol, indicator, value, timestamp, and created_at. The data shows multiple rows for the 'AAPL' symbol, all with the 'moving_average' indicator, sorted by value and timestamp.

| symbol | indicator | value | timestamp | created_at |
|--------|----------------|-------------------|---------------------|----------------------------|
| AAPL | moving_average | 985.0461673901935 | 2024-02-02 16:31:55 | 2024-02-02 16:31:56.602899 |
| AAPL | moving_average | 985.1566263747444 | 2024-02-02 13:18:40 | 2024-02-02 13:18:41.753501 |
| AAPL | moving_average | 985.899357533317 | 2024-02-02 13:11:27 | 2024-02-02 13:11:28.118604 |
| AAPL | moving_average | 986.0788593368025 | 2024-02-02 16:29:23 | 2024-02-02 16:29:24.96633 |
| AAPL | moving_average | 986.2727905663957 | 2024-02-02 16:51:51 | 2024-02-02 16:51:52.484302 |
| AAPL | moving_average | 986.3712907018911 | 2024-02-02 15:38:52 | 2024-02-02 15:38:53.093209 |
| AAPL | moving_average | 986.3745511260572 | 2024-02-02 13:46:53 | 2024-02-02 13:46:54.991686 |
| AAPL | moving_average | 986.7227872790296 | 2024-02-02 16:24:20 | 2024-02-02 16:24:21.731418 |
| AAPL | moving_average | 986.7965105627756 | 2024-02-02 17:13:03 | 2024-02-02 17:13:04.569737 |
| AAPL | moving_average | 986.9200139744358 | 2024-02-02 17:52:59 | 2024-02-02 17:53:00.586585 |
| AAPL | moving_average | 986.9540947986842 | 2024-02-02 16:05:18 | 2024-02-02 16:05:19.071141 |