StockInfo architecture

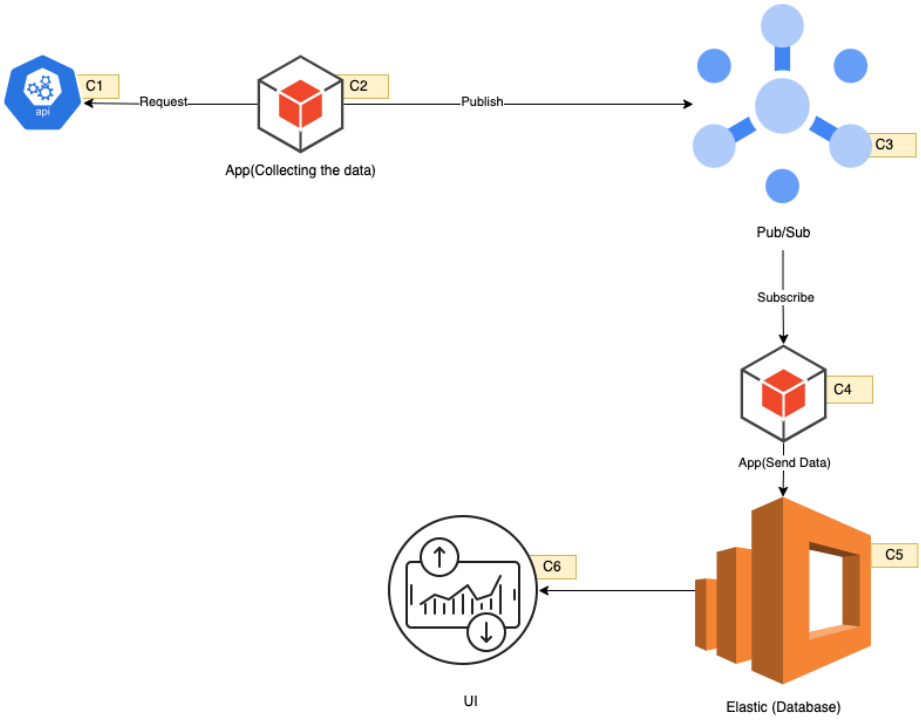

Figure 1: StockInfoArch

StockInfo application is composed of six components, that are independent between them following the microservice architecture. The resasons of choosing the microservice arhitecture and not a monolit was:

1. Improved scalability: Because each microservice is a self-contained unit that performs a specific function, it is easier to scale specific microservices as needed, rather than scaling the entire application as a monolithic unit
2. Enhanced flexibility: Microservices allow organizations to build and deploy applications using a variety of programming languages and tools, rather than being tied to a single technology stack.
3. Improved resilience and fault tolerance: Because microservices are independently deployable units, if one microservice fails, it can be restarted or replaced without affecting the overall operation of the application.

Components overview

1. API Component (C1)

2. App - Data Collector Component (C2)
3. PubSub Component (C3)
4. App - Data Sender Component (C4)
5. Elastic Database Component (C5)
6. Kibana Component (C6)

Components description

API Component C1

The purpose of this component is to provide information about the price for the stocks/materials/energy. This component is not developed by us but is necessary in order to obtain the data. C1 should be able to provide the data through a REST call or by providing an easy parsing HTML. Example of usage of API Component:

- Yahoo Finance API

  url: `https://query1.finance.yahoo.com/v7/finance/options/{company_symbol}`

  In order to get the information about the stock price from Microsoft we will need to replace the **{company_symbol}** with **MSFT** in the yahoo finance api url. The response will be a json, which is easy to parse and get the useful information.

```
{
    ...
    "hasMiniOptions": false,
        "quote": {
            "region": "US",
            "currency": "USD",
            "shortName": "Microsoft Corporation",
            "longName": "Microsoft Corporation",
            "exchangeTimezoneName": "America/New_York",
            "regularMarketPrice": 241.22,
            "regularMarketDayHigh": 243.74,
            "regularMarketDayRange": "239.03 - 243.74",
            "regularMarketDayLow": 239.03,
            "regularMarketVolume": 27613523,
    ...
}
```

- Opcom HTML page

  For this data resource, we will rely on creating a parser for the HTML page. The data obtained from this website are in a table and a chart.

  url: Opcom price energy

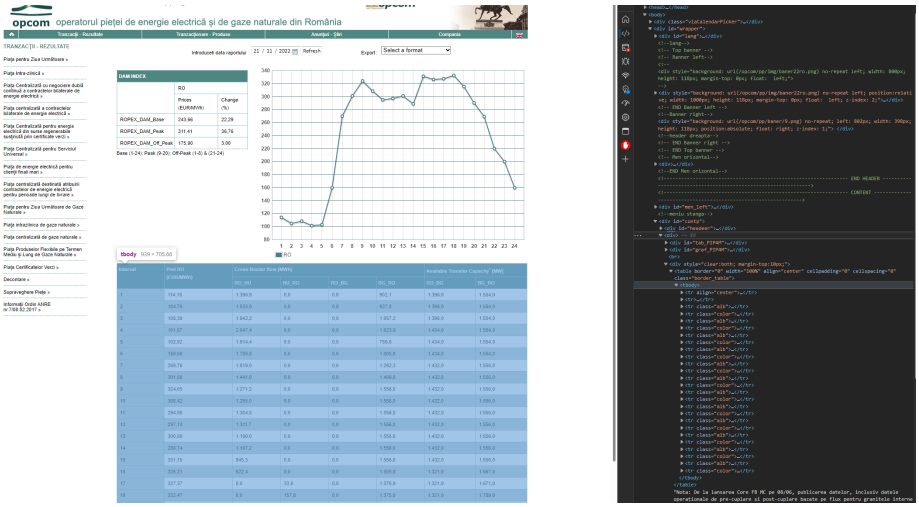

Figure 2: Opcom

App Data Collector Component C2

App Data Collector Component will be a microservice written in any programming language with the scope of collecting data and submitting this data to C3. This component should be able to do REST requests or parse the data from C1. After the data is obtained from C1, App Data Collector Component will send the data to PubSub service which is the third component. We can have any number of C2 microservices with different purposes for collecting the data, for example, one microservice can collect the data by doing a rest call to an API and another microservice can parse an HTML page. Both microservices will be able to send the data in the time and can scale to any number.

PubSub Component C3

PubSub Component will be a service in the cloud to assure decoupling between App Data Collector and App Data Sender. This service will provide asynchronous communication between C2 and C4. For the actual implementation of PubSub we will use the Azure Web PubSub service from Microsoft. A key benefit of using queues is to achieve temporal decoupling of application components. In other words, the producers (senders) and consumers (receivers) don't have to send and receive messages at the same time. That's because messages are stored durably in the queue. Furthermore, the producer doesn't have to wait for a reply from the consumer to continue to process and send messages.

Subscriber: The subscriber is the *Receiver* from the above picture. The role of subscriber is to read messages about stock info from the *Message Queue*.

Publisher: The publisher is the Sender from the above picture. The role of the

Figure 3: PubSub

publisher is to send messages related to the stock info into *Message Queue*.

Actual implementation: - The client will connect to the Azure Web PubSub service through the standard WebSocket protocol using JSON Web Token (JWT) authentication. We need to create a WebSocket connection to connect to a hub in Azure Web PubSub. Hub is a logical unit in Azure Web PubSub where you can publish messages to a group of clients.

Message format:

```
{
    "category": "string-value",
    "name": "string-value",
    "date": "date-value",
    "price": "double-value"
}
```

App Data Sender Component C4

The App Data Sender Component is a microservice which listen for messages from PubSub Component. Because we rely on PubSub we can have multiple C4 applications in cloud to listen for messages. The purpose of this microservice is to get he messages related to the stock information from *Message Queue* and send them to the Elastic Database through a REST call.

Elastic Database Component C5

The reason of using an elastic database over a traditional database like MongoDB is the ability to easily scale up or down the amount of resources allocated to the database in response to changing workload demands. This can help to optimize performance and minimize cost by allocating only the necessary resources when they are needed.

The Elastic Database will be our No Sql database for storing, searching and analyzing our large volumes of data in near-real-time.

Elasticsearch is compose by node and cluster.

Node A node is a server and a part of the cluster that stores the data. It can be either virtual or physical. A node refers to an instance of Elasticsearch,

not a machine. Therefore, any number of nodes can run on the same machine. Whenever an elasticsearch instance starts, a node starts running.

Each and every node be a part of the cluster. It participates in searching and indexing of clusters, which means that a node participates in search query by searching the data stored by it. A node stores the data, which is searched by the search query.

Cluster An Elasticsearch cluster is a group of Elasticsearch nodes, which are connected to each other and together stores all of your data. Each node contains a part of the cluster's data that you add to the cluster. You can use any number of clusters, but one node is usually sufficient. A cluster is automatically created when a node starts up.

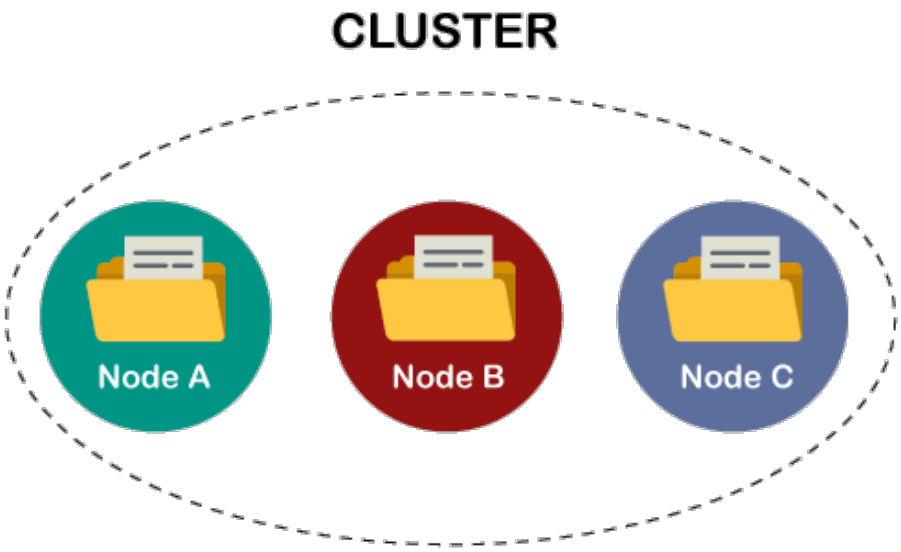

Figure 4: NodeCluster

UI Component C6

The Kibana Component is our solution for the data visialization. The data comes from Elastic Database. Kibana lets users visualize data with charts and graphs in Elasticsearch.

Conclusioin

In conclusion, implementing a microservice architecture for a stock information visualization application can offer a number of benefits, including improved scalability, flexibility, and resilience. By breaking down the application into

independently deployable units, it is possible to scale specific microservices as needed to handle increased traffic or demand, and to make changes to individual microservices without affecting the entire application.

References

[1] https://www.elastic.co/guide/index.html#viewall

[2] https://learn.microsoft.com/en-us/azure/service-bus-messaging/service-bus-messaging-overview

[3] https://learn.microsoft.com/en-us/azure/?product=popular

[4] https://learn.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices

[5] https://www.opcom.ro/acasa/ro