AWS Microservices Architecture for Vienna Stock Exchange's Direct Market Plus Application

Sandesh Muralidhar
StudentId: 20195737
Cloud Platform Programming, MSc in Cloud Computing
National College of Ireland Dublin, IRELAND
Email: x201957374@student.ncirl.ie. URL: www.ncirl.ie

***Abstract*—Stock market will always be one of the critical component of the country economy. The stock market industry is increasingly adopting cloud platform as It has many benefits such as Scalability, Reliability and Accessibility. They are also looking into reducing the cost of operations by moving into the cloud. However many companies face significant challenges in migrating their complete operations to the cloud due to their complex legacy systems and security/regulation concerns. Vienna Stock Exchange is a platform stock exchange of Austria that currently rely on using Xetra system by Deutsche Börse which is a German-based financial company that provides many services for the functioning of the complete stock exchange. In this case, study I will be proposing to them an AWS Microservice architecture for their application direct market plus which will serve all the functionality of the application including their own matching server logic and stock value re-calculation using AWS services so that they don't have to depend on Xetra systems for their operations. I am hosting the application on Elastic Beanstalk and for the backend I am using Microservices for functionality. I am using AWS Elastic Map reduce for stock value re-calculations following all the compliance policies.**

***Index Terms*—Amazon Web Services (AWS), Elastic Beanstalk(EBS), Elastic Map reduce(EMR), Financial Market Authority(FMA), simple queue service(SQS)**

I. Introduction

Stock market is considered to be the most important contributing factor to the national economy. Exchange of stocks, trades will directly have an impact on the economy of the country. The relation between stock trading volume and the return an investor gets has been an active area for research [1]. There are many contributing factors affecting a stock price but It has a direct relationship to the demand of the stock(buy/sell volume of the stock) .

In this paper, we will be focusing on the usage of cloud platform programming using relevant cloud services for the stock exchange platform. Stock exchange platform is a marketplace where stocks, bonds and other securities are traded between investors of the stock market. Stock trading platform needs to be highly transparent, secure and also needs to be compliant with its government finance authority. The demand for stock trading is continuously increasing and therefore stock exchange platforms are facing problems related to scalability, performance and cost of operations. Stock exchange platform needs to serve the brokering applications and it's demand to buy/sell stocks by matching the sell order and buy order for the required stock and also re-calculating the stock price based on orders. It also needs to have the functionality to display the graph of the stock based on its recalculated value in real time.

For my case study I am using Vienna Stock Exchange (Wiener Börse) from austria. It has been in partnership with Deutsche Börse German company for it's Xetra® system which enables then for matching orders, floor trading and many other functionalities on which Vienna Stock Exchange completely relies on. They also signed a contract with German company on 2020 to extend their contract for 5 more years so that they can use modern technology exchange platform [2]. I will be proposing a case study of Vienna Stock Exchange application completely operating in AWS Cloud without Xetra system for their operations.

II. Company Description

Vienna Stock Exchange is a stock exchange situated in Vienna, Austria. It is the only security exchange for Austria and is under supervision of the Austrian Financial Market Authority(FMA). It lists both official market stocks as well as stocks only available in Vienna stock indices. They use Xetra of Deutsche Börse company based in Germany for floor trading and other functionalities [3]. They have an application called Direct Market plus which offers Austrian small and medium-sized enterprises to list themselves in the stock market and startups in Austria and they also handle other regulated stocks of Austria for trading. So it can serve both customers and brokerage applications [4]

A. Functional Requirements

Functional requirements are the functions that define what an application should do. These requirements need to be met **Buy/Sell Stocks**: The application must provide the functionality for brokers to place their orders to buy or sell listed stocks in the platform. Brokers/customer with this functionality will be able to specify the price at which they want to buy or sell a particular listed stock.

Validate Order: Application should be able to validate an order placed to see If the order is valid for instance If an order is placed for a particular stock then application needs to validate If the customer has sufficient balance in their DMAT

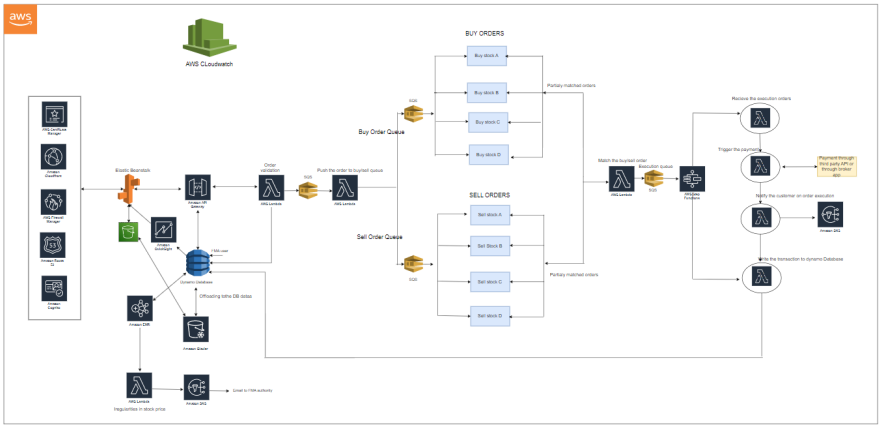

Fig 1. Diagram showing Implementation of market plus platform application using AWS services

account to buy the stock and If there is an order to sell the stock then customer's dmat account needs to be validated to see If that DMAT account has the quantity of stocks in his DMAT account to sell.

Matching: The application with the matching feature needs to match the order that is it needs to check the buy order and sell order for a particular stock and then match it based on the quantity available and price of the stock. For example, If stock A of quantity 5 is being bought by a customer at a certain price then same stock needs to have sell order as well of 5 quantities or more with the same or lesser price for successful matching.

Partial Matching: If an order cannot be fully matched, Then application should also support partial order execution where only part of order can be executed. For instance, If there is a buy order of a stock A quantity 10 but only 5 stock A quantity are available to be sold. Then It is a partial match where only 5 quantity of Stock A will be executed and the remaining buy order of 5 quantity will remain to be executed.

Execution: The application should be able to execute the orders as soon as they are matched. This includes that application should be able to send payment notification to third party application and wait for the confirmation to proceed further and keep an record.

Re-calculation of stock price: Stock price needs to be re-calculated every time based on demand that is based on buy/sell order for that particular stock.

Real-time Stock Updates: The application must display real-time updates of the stock prices and display those changes in terms of a graph for user readability.

B. Non -Functional Requirements

These are the functionalities required for efficient operation of an application.

Report unfair practice Application should also be able to report any irregularities on stock fluctuation. It should be able to detect and report it.

Scalability: Application should be able to scale up and scale down according to the application usage demand without impacting performance

Security: Application should be highly secure as it will contain very sensitive stock market-related data and complete transactions

Reliability: Application should be highly available all the time without any issue and downtime or maintenance window should be minimal.

Regulatory compliance: Application should follow all the rules and regulations set by the financial authority in this case Austrian Financial market authority(FMA)

III. Proposed Architecture Explanation

I have proposed a microservice-level architecture where the application is built on independent component and each service performs a single function. I have also followed AWS Cloud Adoption Framework (AWS CAF) while designing the architecture fig1 shows the proposed architecture proposal to use AWS service for the complete operation of Vienna Stock Exchange Direct market plus application without using Xetra system.

1) Clients will first hit the CDN, which will route the request to the elastic beanstalk **2)** Front end also consists of AWS cognito for user authentication, AWS secret manager for securely storing secrets like API keys, Route 53 for DNS management, AWS certificate manager for SSL/TLS certificate management and AWS WAF for firewall protection. Requests will be directed to Elastic Beanstalk **3)** Elastic Beanstalk on which frontend is hosted is connected to S3 storage bucket to store static files and other data of the application **4)** Elastic Beanstalk will use an API gateway to handle API requests and direct them to the microservices as I have used lambda functions **5)** So when Elastic Beanstalk receives an order It is sent to API gateway and then to a lambda function for order validation. In this stage It will perform all the necessary checks, such as verifying If the user has a proper balance and If It is a sell order then It will check If the DMAT account has the corresponding quantity of shares to sell. Once verified It will write the order to the dynamo DB and then with the order ID places it on the queue. I am using the simple queue service

of AWS **6)** Then the orders go to next lambda function which sorts the orders according to buy or sell order and places it in its respective queue. For each stock, there is buy queue and sell queue. So depending on the stock and it's order type. It places it in buy/sell order queue **7)** Once an order is placed in buy or sell queue then It triggers its respective lambda function which performs matching functionality. If a buy order placed in the queue triggers lambda then It will check the respective stock selling queue to check for any match. If it finds a match then It forwards It into the execution queue. If no match is found then the function simply terminates and It will check for matching functionality again when there is any order for the queue. It performs matching related to the Quantity and price of the stock For instance, If a buy order is placed for stock A of quantity 10 at price 1000 euro then It will be placed in buy order queue of stock A then matching functionality lambda will look at sell queue of stock A if it finds a) sell stock A of quantity 10 at price 1000 or lower then It will execute it and forward it to execution queue b) sell stock A of quantity 10 at price 1050 euro then It won't execute it because It is not a match C) Sell stock A of quantity 8 at 1000 then It will only match 8 orders then put remaining 2 quantity buy orders back to the buy queue **8)** Matched orders are forwarded to the Execution queue there lambda step function will be executed where in the first step It will receive the orders and then in the next step It will trigger the payment option through third party api where If the order is through broker app It will connect to them If the order is for same exchange stocks then It will initiate the payment. It will wait for the payment success message. In the next step, It will trigger a notification of the payment status to the customer who has purchased stocks in it's exchange then in the final step It writes the status to the dynamo DB **9)** Amazon Elastic Map reduce will be watching the execution table in dynamo Db and as and when there is changes It will recalculate the stock price of the stock for which order was executed. Stock price varies according to it's demand that is order execution. EMR uses Apache spark framework to perform this functionality. It will use a script written in python language to perform the recalculation. stock price also depends on various other factors like Quater results and other indices. So in order to feed the same I have created IAM access role to FMA authority of Austria who will feed the relevant Information so that they can get accurate values. **10)** If there is a sudden surge of the price of a stock then EMR will trigger a lambda function which triggers a notification to FMA authority asking them to check on the corresponding stock. This will help to maintain fairness and find out any irregularities. **11)** EMR will then update the dynamo DB with the recalculated value of the stock price and then AWS quick sight will plot a graph or insert changes to its graph in real-time based on the value inserted in the dynamo DB **12)** Dynamo DB will only keep latest 3 days of data and then offloads 4th day data to AWS glacier which is for data archiving and backup service. S3 bucket of the application is also connected to AWS Glacier. **13)** AWS Cloudwatch is used for threat detection, Monitoring purpose and to trigger alarms.

IV. Justification of Selected AWS services including their Advantages and Disadvantages

AWS cloudfront: It is a service to securely deliver content with high speed and low latency. It can serve any user accessing the website in mili seconds. It caches and delivers static and dynamic content for users as it caches the content closer to the user. In my proposal I am using CloudFront for faster content delivery and to route the requests to Elastic Beanstalk thereby reducing the load directly on the application. It also helps in improving performance as it caches content near end user for faster access. Its disadvantage is that It adds for additional cost as it is charged for each data transfer along with HTTPS request. Since I am using only HTTPS connection managing the certificate will be complex this can be overcome by using AWS certificate manager [5]

AWS certificate manager: It is a service that removes the overhead of purchasing, renewing and handling SSL/TLS certificate. This service adds to our application security and secure traffic handling. It is used along with AWS Cloudfront as It can provide SSL certificate and maintain it for Cloudfront. Usage of this service causes vendor lock in but since in our architecture, we are only using AWS services. A certificate manager is a wise choice. [6]

AWS Cognito : It easily enables adding user authentication to the web application. It provides functionality to store user session and also provide functionality to assist during forgotten passwords. It adds up to the application security level due to its various functionalities. It is difficult to integrate this service with external provider [7]

Amazon Route 53: It provides Scalable DNS service. It is a very cost-effective way to channel or route the users to our web application. It can be easily integrated with other AWS services and in our proposal, I am using route 53 for directing requests to Elastic beanstalk application. [8]

AWS WAF: It is a service that allows us to create security rules and prevent bots, sql injection and other various attacks. I am using AWS WAF along with other AWS service in frontend to add to security functionality of the application. Using this server we will also be able monitor user access accounts that are compromised [9].

AWS Elastic beanstalk :It is a service that allows to deploy an website on it and then automatically creates the resources required for it. It is a fully managed service. It is auto scalable and all by itself create a load balancer. Another advantage of Elastic beanstalk is we can have multiple environment. To leverage all these features I am using this service to deploy my frontend application. Disadvantages of this service is limited control of underlying infrastructure as it is fully managed and Troubleshooting can also be challenging [10].

AWS lambda and AWS step function : AWS lambda is a serveless computing service that runs the code when there is change of events. It is also fully managed and scale on demand. It is also very cost-effective solution. In my Proposal I am using lambda for various functionalities like Matching orders, Validating orders and triggering notifications.

It's disadvantages are that a lambda function can only last for 15 minutes also debugging of AWS lambda is difficult as the functionalities are decoupled. AWS step function in an orchestration layer which is also serverless. AWS step function handles input, output and errors and retires the functions. It works on a concept of state machines and tasks. Where state machine is the workflow and task is a single function. I am using AWS step function in a proposal to perform the execution of the orders [11].

AWS SQS: It is a messaging queue for microservices. It allows for decoupling of service hence suitable for our microservices backend. All the lambda function can publish their output to the queue and other services can receive from same queue. It acts as a buffer between the services. It works on First in first out(FIFO) which helps to maintain order. I have used SQS in many aspects of application as I have proposed a microservices architecture. The only disadvantage of this service is It can handle 20,000 messages at a time and won't be able to accommodate new messages until the old message exits however in our application we don't project the queue to have more than 20,000 messages to processed at a time. [12]

AWS dynamo Database: It is a Nosql database service that is fast and flexible. It can serve the request in a very few millisecond. It is very secure and has SLA of 99.999 percent. It can be easily integrated. It is a fully managed serverless database which supports in-memory caching. It also automatically scales up and scales down based on the demand hence making this database an ideal choice for the proposed plan. [13]

AMAZON EMR : It is a cluster platform that runs big data framework such as Apache Spark It has 3 nodes Primary Node, Core Node and Task node. It is a managed cluster that processes the data in parallel making it an ideal choice for our proposed case since we need real-time calculation and precise values. In my architecture, I am using Amzon EMR to recalculate the stock price every time there is write performed on the execution table of dynamo DB and also have set a threshold on values so that It triggers lambda If there is a huge difference of price between previously calculated value and present value [14]

AWS quicksight :It is cloud powered bussiness intelligence service which is used to deliver insights.It is used in my proposal to generate and deliver the graph based on the recalculated value of the stock. It would be able to publish the graph in real-time and fully managed. [15]

AWS Glacier : It is storage for data archiving, It provides high performance and fast retrieval of data. It is low cost archival storage of cloud. It has 3 different archival storage class as options. I am using AWS Glacier in my proposal to offload the data from dynamo DB and S3 bucket [16]

AWS cloudwatch: It is a monitoring tool that collects the logs, and monitors the health of the services. It can be used to set alarms If a resource crosses a threshold and also automated actions can be set to perform when crossed.I am using AWS cloud watch in my proposal for monitoring purposes, log collection for RCA and for alarm generation [17]

V. Conclusions

In this paper I have proposed a micro-service architecture using AWS services for direct market plus application used by Vienna Stock Exchange (Wiener Börse). As a stock market enthusiast, I was absorbed in the research and I was able to learn while researching on how stock market regulations and indices work in europe and the dominance of euronext. Various technologies and cloud services used by Stock trading companies. How important It is to follow the regulations and maintain compliance for an application. I also learned the importance of microservices architecture and decoupling functionalities which enables to independently scale different services based on demand and usage. I was able to understand how important is scalability and reliability for an application and the importance of cloud technologies. For future scope, I would like to add Machine learning to predict the stock value in the same architecture. I also learned from the proposal that the architecture proposed should be cost-effective and should follow best practises of Cloud Adoption Framework. Overall this proposal has given me a great learning experience and has also provided me with learning on the stock market and implementation of cloud services.

References

[1] D.-H. Shih, H.-L. Hsu, and P.-Y. Shih, "A study of early warning system in volume burst risk assessment of stock with big data platform," in *2019 IEEE 4th International Conference on Cloud Computing and Big Data Analysis (ICCCBDA)*, 2019, pp. 244–248.

[2] XETRA, "Wiener börse and deutsche börse to extend cooperation." [Online]. Available: https://www.xetra.com/xetra-en/newsroom/press-releases/list-press-releases/Wiener-B-rse-and-Deutsche-B-rse-to-extend-cooperation-ahead-of-schedule-1936402

[3] B. MCKENZIE, "Overview of exchange." [Online]. Available: https://resourcehub.bakermckenzie.com/en/resources/cross-border-listings-handbook/europe-middle-east--africa/vienna-stock-exchange/topics/overview-of-exchange

[4] W. Borse, "Easy access to the capital market for smes – direct market plus." [Online]. Available: https://www.wienerborse.at/en/listing/going-public-ipo/directmarketplus

[5] AWS, "Amazon cloudfront." [Online]. Available: https://aws.amazon.com/cloudfront/

[6] ——, "Requirements for using ssl/tls certificates with cloudfront." [Online]. Available: https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/cnames-and-https-requirements.html/

[7] A. Distasio, "Pros and cons of using amazon cognito for user authentication." [Online]. Available: https://www.gavant.com/library/pros-and-cons-of-using-amazon-cognito-for-user-authentication/

[8] AWS, "Amazon route53." [Online]. Available: https://aws.amazon.com/route53/

[9] ——, "Amazon waf." [Online]. Available: https://aws.amazon.com/waf/

[10] ——, "Amazon elastic beanstalk." [Online]. Available: https://leandromerli.com/elastic-beanstalk/

[11] ClearScale, "Aws step functions vs aws lambda." [Online]. Available: https://blog.clearscale.com/aws-step-functions-vs-aws-lambda/

[12] AWS, "Amazon sqs." [Online]. Available: https://aws.amazon.com/sqs/faqs/

[13] ——, "Amazon dynamo db." [Online]. Available: https://aws.amazon.com/dynamodb/

[14] ——, "Overview of amazon emr." [Online]. Available: https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-overview.html

[15] ——, "Aws quicksight." [Online]. Available: https://docs.aws.amazon.com/managedservices/latest/userguide/quicksight.html

[16] ——, "Aws glacier." [Online]. Available: https://aws.amazon.com/s3/storage-classes/glacier/

[17] ——, "Aws cloudwatch." [Online]. Available: https://aws.amazon.com/cloudwatch/