Migrating CryptoCommunity to a Cloud-Based Serverless Architecture

By Richard Castaneda, Ernest Gurish, Zhiyuan Luo, and Akhil Maddipatla

Introduction & Use Case:

Cryptocurrency and similar blockchain technologies have entered a realm of unexpected popularity. What began as an obscure piece of technology has evolved to something many are considering useful. Cryptocurrencies like Bitcoin are now widely reported on and, in some places like Argentina, are being considered a more valuable form of payment than plain money.[1] Cryptocurrencies have multiple layers of complexity. It is used as a form of currency though the value of any individual cryptocurrency is quite variable. Not only are there economic and technological complexities, but also concerns around the environmental impact of proof-of-work systems.[2] As a result of this rapidly growing multifaceted technology, people will no doubt look for resources dedicated to understanding it as well as sharing their opinions on it. This is where our motivation for ***CryptoCommunity***, our cloud-based cryptocurrency community and forum, begins.

The Problem:

The problem we are looking to solve is creating a lightweight, user friendly solution to gaining valuable information about Cryptocurrency while minimizing long term and upfront startup cost. ***CryptoCommunity*** is currently hosted locally, and while this allows for greater control of our product, we are looking to implement a solution that reduces costs and maintenance time on our end. Our project migrates an existing React web application and the associated database management to Amazon Web Services. In this project we took a web application ***CryptoCommunity*** created by one of our project members, Richard Castaneda, and migrated it over to be hosted and maintained on AWS using all free-tier eligible products. This means that CryptoCommunity can enjoy the added security and reliability benefits of cloud-computing while significantly reducing cost using free-tier eligible products.

This work is important in that it creates an environment in which CryptoCommunity can utilize some of the basic characteristics of cloud computing while providing a platform for easy growth in the future as well as low upfront startup cost. In addition, this project displays the convenience of being able to provision resources from a wide array of available services on AWS without needing the help of onsite AWS staff to implement our design. The work we did was able to be done with the assistance of any at AWS while still producing a full implementation of our product and having it go live that very same day.

---

[1] https://www.bbc.com/news/business-60912789

[2] https://www.washingtonpost.com/business/energy/why-bitcoins-environmental-problems-are-so-hard-to-fix/2022/03/16/b71e1d22-a4df-11ec-8628-3da4fa8f8714_story.html

A cloud-based solution offers a number of key advantages. The ease at which the front-end can be hosted and accessible in multiple different regions and countries is one advantage. Rather than just be localized to the U.S. on a personal server, in a few simple steps it can be deployed in Europe and Asia as well, giving the application a far more global reach. Moving the implementation to the cloud also gives us all the tools and services needed to modularize and improve our current deployment model. Perhaps the front-end offerings would expand, necessitating a move to a dynamic website hosted on EC2. Likely the API would need to be extended and expanded upon as well which would be better served by the AWS API Gateway. Lastly, the cloud-based solution creates a far more easily scalable product. If the back-end database cannot keep up with requests, it can be upscaled to a more powerful RDS instance with a few clicks. Overall, a cloud-based solution gives the application a global reach with the flexibility to continuously improve and scale our current implementation with the wide array of AWS services.

Cloud Computing Characteristics:

CryptoCommunity's unique problem and use case calls for three primary characteristics of cloud computing. First and foremost, our objective is to keep both initial startup costs and monthly maintenance costs low. Second, this website was created, and maintained by us as there are no intermediary services required to help upkeep the website and its content. Finally, we want our website to be available to a multitude of clientele and minimize downtime for our customers. With these goals in mind, we chose a cloud computing model for its **measured service, broad network access, and on-demand self service.** Measured service allows us to provision only what is absolutely necessary to host our site at dynamic periods of volume. This means that as we grow, we can provision more resources but only as we need them rather than investing in large startup costs to get our product off the ground. Broad network access helps us ensure that our product is deliverable to many points around the globe without having to worry about content delivery or high propagation and transmission delays. Finally, on-demand self service allows us full control over our provisioned services and website front and backend allowing us to have the engineers in control and access to our website 24/7 without requiring the help from someone at AWS.

Early Iterations:

Our implementation went through two major shifts in strategy before arriving at our final architecture. Our initial project proposal was to decompose the original web application into a small set of microservices. As a group, we wanted to gain experience with planning and implementing a microservice architecture as well as learn more about Docker. Since the original CryptoCommunity web application was relatively compact in terms of codebase size and scope, we thought it would be a good candidate for getting this experience with microservices. The microservice architecture would also serve as an improvement on the application itself. By dividing it into smaller services, it would be easier to scale smaller portions of the app as well as make feature extension an easier process. After reviewing our available resources in AWS and the timeframe for which we had to implement this project alongside guidance from faculty members, we shifted our focus from migrating ***CryptoCommunity*** to a microservice architecture to a

monolithic architecture hosted on AWS's S3 platform. This lightweight hosting alongside easy database retrieval powered by AWS RDS would allow for a stable and deployment ready implementation of ***CryptoCommunity*** in the time frame allotted while retaining a database of users and user information that could be migrated to a microservices architecture later on.

During our initial runs we experimented with a multitude of AWS services such as AWS Lambda, AWS API Gateway, AWS Amplify, AWS CloudFront, AWS CloudFormation, AWS S3, and the classic AWS EC2. Our initial research led us to try implementing our API in AWS API Gateway and have that route back to AWS RDS to manipulate data in our database.(Figure 2.0)This process proved to be extensive as we had to essentially rewrite all of our API calls and REST controllers to adapt to this infrastructure. Given the amount of resources we were using and how long the implementation was taking we decided it didn't fit into the original ideology for the solution we would craft. We wanted to focus on low cost, quick deployment, and a stable product.

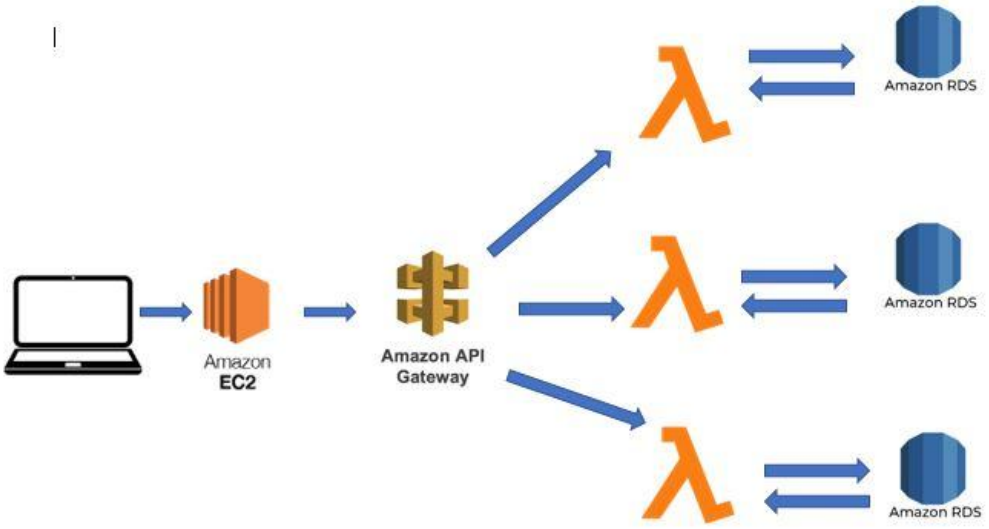

Figure 2.0 - Original Implementation Microservices in EC2, API Gateway, RDS

Our next path led us to where we are now. Having a SpringBoot program that could handle the API requests was the easiest way to implement our backend and ensure it was not only stable but was easy to deploy and update if changes needed to be made. Very little source code needed to be updated and the existing codebase was reused rather than rewritten to fit the mold of an AWS product. We felt this approach was closer to our original goals and created a product that was most similar to the original implementation of ***CryptoCommunity***.

Our final implementation hurdle was choosing a front end implementation. We had experimented with Amplify, CloudFront, and S3 in these respects and found S3 was the cheapest way to hose ***CryptoCommunity*** while still having a lot of control of the permissions related to the website. AWS Amplify, while a great tool, took our front end package, built it, and deployed it all without additional provisioning. This approach is geat for quick deployment and ease but we wanted to have more control over our

content permissions and very little settings were available in the Amplify console to do that. AWS CloudFront was also a great resource however we found that it ultimately served our site from an S3 bucket which added another layer of services when all we really needed was one. We decided to take the simplest approach to ensure billing simplicity and ease of deployment when updated to the site are needed.

Our second strategy was born from our conversation with our TA Mihir who recommended looking into implementing a serverless architecture.

Final Architecture & Overview:

Our project frontend is based on a React application that is stored in an S3 bucket and hosted in a serverless implementation. The front end is hosted and served by S3 as a static website. Cookies allow us to maintain login state persistent throughout the application while logged in. API calls are made from the S3 hosted front end in order to GET, DELETE, PUT, and POST information to both display and store on the MySQL database. These API requests are called to both our own AWS RDS instance and to CoinAPI which we use to collect up to date coin data. CoinAPI is a third party service for which we obtained an API key for testing purposes.

Our project back end is built on AWS Elastic Beanstalk (which is a free service run on EC2 and S3) in conjunction with AWS RDS running MySQL. Elastic Beanstalk hosts our Java Springboot backend which handles REST API calls made from the front end and delivers them to the AWS RDS SQL database. This is also how content is served from the database to the users. Diagram below for reference. (Figure 3.0)

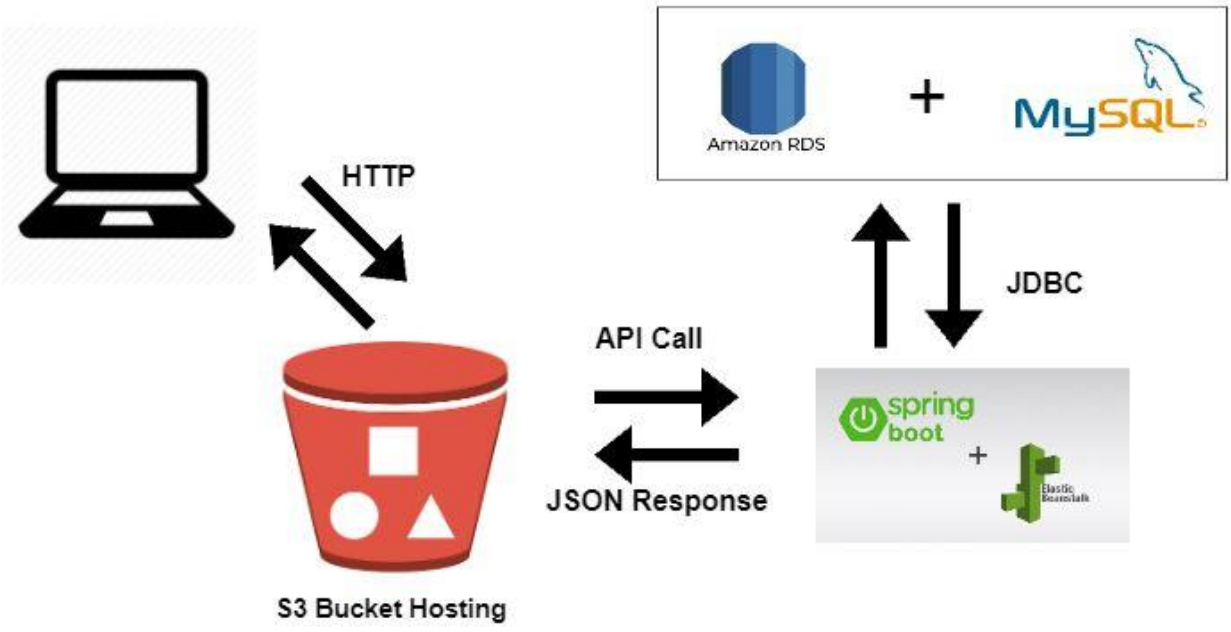

Figure 3.0 - Final Implementation

Data Flow:

User inputted data is received by the react app in an S3 bucket and sent out to a Java spring boot application via RestAPI calls that uses a Rest controller to save and retrieve data from MySQL. Our Springboot application has a predefined list of API calls to handle when handling user data. When an API call is made, it is handled by our Java program and data is either retrieved or saved to the database. In our implementation of CryptoCommunity we have many types of data being stored including user account data, forum posts, comments, and cryptocurrency (coin) data. Our MySQL database is hosted by AWS RDS. Users do not have direct access to the database as all requests are made through API calls that are handled by SpringBoot and JDBC (Java Database Connector). Since there is no user to database connection and only approved API calls are sent to the database, our data is protected and secured from outside sources. Administrative users are able to access our database as needed for maintenance and other general technology issues. A modification that we can make in the future is moving our hosting platform to a more robust service like AWS CloudFront in order to add HTTPS abilities.

Implementation Details and Codebase:

CryptoCommunity is based on React on the front end and SpringBoot on the backend. Our code base is online at https://github.com/NEU-Cloud-Computing/CryptoCommunity. This repository holds the front end and the backend. In the front end, the React website is built with Node Package Manager (NPM) and the associated build folder is hosted on S3 and deployed there as well. Our backend is packaged into a Jar file with Maven and uploaded and deployed using AWS Elastic Beanstalk which is run on EC2 instances and S3 buckets. All of our AWS services are free tier eligible making our current deployment of CrypoCommunity free up to the AWS free tier monthly limits. The current deployment of ***CryptoCommunity*** is available live at http://cryptocommunity-preflight-stage.s3-website-us-west-2.amazonaws.com/

Challenges & Looking Ahead

The largest challenges of this project were modifying the current codebase to work correctly with the services provided by AWS and properly provisioning AWS resources so they could communicate with each other. In the original implementation, many of the database URL and API structures were hard-coded into the program, all of which had to be changed to fit the structure and naming conventions of AWS. In addition, issues like CORS and API mis-calls made for a strenuous testing and redeployment cycle that led to much of our free-tier resources being used just in testing and deploying our project. In addition to this work, we also had to provision all of our resources to be able to communicate with each other with ease within the AWS network of services. In the end, we successfully migrated CryptoCommunity over to AWS completely and have found a reliable, low-cost, and highly available hosting platform that will help server CryptoCommunity and it's growing base of users for years to come.In the future, S3 bucket hosting will not be adequate to deliver out content out to a variety of users, utilizing technologies like AWS Cloudfront and HTTPS we can provide

an even more secure and seamless user experience as we continue to serve more countries around the globe.