CSCI4145 Cloud Computing Term Assignment

The Grapevine Chronicles: A web app hosted on AWS

Jenna Cogswell
B00829098

2024-04-09

Table of Contents

Introduction

This project involves leveraging AWS cloud computing resources and architecture, in order to develop, deploy, and deliver a Next.js web application. Titled “The Grapevine Chronicles”, the aim of this web application is to provide users a platform to browse, create, and interact with posts by other users, in ways such as listening to the post content using Amazon Polly’s text-to-speech. These posts can encompass a wide range of content, including blog posts, short stories, poems, essays, reviews, articles, and more. Each post takes the form of a title, description, and the text content. Users are able to sign up, log in, log out, browse, and create posts.

The goal is to provide users with a space to share and collaborate with fellow creative writing and blogging enthusiasts. This app will curate a space that supports open-sourced information sharing and story telling. While the current version of this application does not meet industry standards for public release (as it does not have all CRUD operations working, nor does it have many features), the ideal finished version would compete with other web applications such as, Wattpad, WordPress.com, and Medium. To achieve this level of competition, the application needs to be highly available, scalable, and secure.

Throughout the planning and design phases, I considered various factors such as AWS Academy limitations, timeline constraints, and the realistic expectations of the web applications user base. The final cloud infrastructure was designed to deploy and deliver the app efficiently.

Menu Item Requirements

List of selected services.

- AWS EC2
- AWS Elastic Container Registry & AWS Elastic Container Service (mostly working)
- AWS VPC
- AWS CloudFront
- AWS RDS (MySQL)
- AWS S3
- AWS Secrets Manager
- Amazon Polly

Comparison of alternative services and why chosen services were selected over others.

Compute

AWS EC2 vs Elastic Beanstalk + AWS Elastic Load Balancer – Elastic Beanstalk with a load balancer can host a web app environment on EC2’s by uploading new versions of the app in zip files, and horizontally scale the application by automatically adding or removing EC2’s as needed. Elastic Beanstalk can also ensure high availability by storing the EC2’s in multiple availability zones for fast disaster recover and little to no downtime. An EC2 without Elastic

Beanstalk or load balancing can host a web application directly on the EC2, with the ability to SSH into the virtual machine for directly configuring the web app environment and files as needed. Using an EC2, PM2 can be incorporated with the Next.js app to keep the server running even when not connected to the EC2.

Initially, I opted to use AWS Elastic Beanstalk for hosting the web application, as it provides a highly scalable and accessible solution. However, during the development process, I encountered challenges when updating the application environment that became increasingly difficult to resolve. Following a suggestion from the professor, I transitioned to hosting directly on an EC2 instance, which I connected to my RDS database, Secrets Manager, and S3 bucket, all within a VPC. Once I was able to get a working version of my application directly on the EC2, by testing the database connection, NextAuth for user authentication, post creation, and Amazon Polly and S3 connection, my project still only met one of the compute requirements (EC2). I considered using AWS Lambda since I had some experience with it from the course, I went as far as creating and testing a Lambda function that I could've used within my application. However, after some more research and reflection, I did not end up incorporating AWS Lambda, as I realized since I am using Next.js with a backend, my application requires a server, which eliminates most of the benefits of AWS Lambda (as well as AWS Step Functions).

I opted to use AWS Elastic Container Service to create an EC2 cluster for hosting a Dockerized version of my web app through Elastic Container Registry. Upon following along with various tutorials on ECS and ECR, I was able to setup my cluster and run my Docker container as a task on the cluster.

AWS IoT 1-click was ruled out since I would not be using any IoT devices for this project.

Storage

For the storage resources, I chose to use both AWS RDS and AWS S3. Despite only requiring one storage option, this web app requires both a place to easily store and retrieve user and post data, as well as a place to store the audio files. In comparison to the other storage options, I found that S3 is the best option for storing files, with 99.99999999% durability, 99.99% availability, and pay as you go pricing [1]. For storing user and post data, I quickly decided to use AWS RDS because of its easy connection to EC2's, my experience with using RDS in a past assignment, and my existing knowledge of MySQL.

Since the tables and data involved in this application do not currently require complex relationships, I felt that the graph database options such as AppSync and Neptune were not necessary. I opted not to use DynamoDB since my data is structured, consistent, and doesn't currently have a need for high scalability []. AWS Aurora was warned to be quite costly, and IoT analytics again was not chosen since I am not incorporating IoT devices.

Networks

For networking I was set on using AWS VPC since completing the first assignment in this course, I feel that AWS VPC is highly secure, useful, and easy to setup. The ability to have both private and public subnets in multiple availability zones was very appealing. I also incorporated Amazon CloudFront for assigning a domain name to my S3 bucket for high-speed content delivery of my MP3 files. AWS API Gateway and Amazon event bridge were not used, as they are most useful to support a serverless and/or microservices architecture.

General

Finally, for the two general service requirements, I opted to use AWS Secrets Manager and Amazon Polly. Initially, with my plan to use Elastic Beanstalk, I would use AWS Elastic Load Balancing, but upon opting out of these options I replaced the Load Balancer with the use of Amazon Polly. I did contemplate using infrastructure as code through CloudFormation, I considered this because I had experience using Terraform to create and automate AWS resources in a past internship. I ended up not incorporating IaC since it would be quite the learning curve and possibly take up more of this projects timeline than I would like.

With two services being required here, I felt it best to choose one option that would add to the security of my application, and one option that might be fun to try out and make my application stand out compared to others. Keeping in mind my priorities of low complexity, low cost, utilizing skills I already know, and incorporating options best suited for my application idea, I contemplated various machine learning / AI related services. However, I was warned that these options through AWS, or AWS Academy in particular, often raise some difficulties and unreliability. Once I finalized my app idea, Amazon Polly stood out to me as a good option to listen to the posts instead of reading lengthy text, this would mimic an audiobook feature.

As for the service that would add to the security of my application, I went with AWS Secrets Manager. This decision became clear to me quickly as during the process of creating an RDS database, there was an option to immediately save the credentials in Secrets Manager. This allows me access the credentials directly from my web app, without storing sensitive data in environment variables.

Deployment Model

Description of chosen deployment model.

The chosen deployment model for "The Grapevine Chronicles" is a public cloud deployment. This involves hosting the application on publicly accessible cloud infrastructure owned and operated by a third-party provider, in this case, AWS (Amazon Web Services).

Reasoning behind selection of this deployment model.

Considering that this course and project requires the use of AWS, I could rule out the option of private only cloud. This leaves the options of public only cloud, hybrid cloud, and multi-cloud. The hybrid cloud option would come with the choice to “Brave the wilds of the FCS OpenStack infrastructure to build a hybrid system that combines public cloud and private cloud”, I opted not to go with this option. The multi-cloud option would come with choosing to use Heroku, or another cloud platform to build a multi-cloud system. I wanted to keep my complexity and learning curve to a minimum, ruling out the multi-cloud option.

Delivery Model

Description of chosen delivery model.

As the cloud consumer and application developer, I have mostly used the Platform as a service delivery model. I am using platform as a service because I do not need to manage the underlying infrastructure, I can simply push and pull containers and AWS will handle any monitoring and scaling that may be needed. My RDS might be more of a IaaS model, since I do not have it in any kind of cluster or load balancer and I must maintain it myself.

Reasoning behind the selection of this delivery model.

I opted for PaaS because it abstracts away most of the management so I do not have to worry about provisioning, configuring, or maintaining my servers. With ECS and ECR I do not have to think about if or when I have to scale my EC2's. PaaS is also cost effective, with the pay-as-you-go model I only pay for what is in use. I can also depend on my application being highly available since any spike in traffic would scale accordingly with no downtime.

System Architecture

Description of the architecture of the final system.

"The Grapevine Chronicles" is a web application built using React, a component-based JavaScript library, along with Next.js, a React Framework, built on top of Node.js to create full stack web applications with server-side components, and leverage NextAuth for efficient user authentication and sessions. The application allows users to register, log in, and log out, storing user data with hashed passwords in a MySQL database. Users can then browse and view the available posts on the main page, or use the navigation bar to create a new post or log out. Once viewing the contents of a post, users can then choose to listen to the post, similar to an audiobook, by using the Amazon Polly text-to-speech feature. Using Next.js server-side components I created REST API's for querying data to/from the MySQL database as well as using Amazon Polly to get the MP3 files and store them in a S3 bucket.

My AWS architecture involves a Virtual Private Cloud for having private and public subnets according to each services security requirements. My EC2 cluster is publicly accessible. My RDS is kept private for securing sensitive user and post data. The S3 bucket uses Amazon CloudFront to create a publicly accessible domain name for fast and efficient content delivery to the users. Amazon Polly and Secrets Manager are also utilized by my application.

For deploying my application initially, before I implemented Docker, I used Git in order to locally make changes to my application, push these changes to my repository, then when I needed to test a new feature or AWS service connection, I would SSH into my EC2, and pull the changes from Git, running the application manually.

For deploying my application on ECR and ECS, I created a Dockerfile for my Next.js application, built the image locally, then used the steps provided by ECR to push my container to AWS. Once the container was pushed to ECR, I was then able to use ECS to create an EC2 cluster, where I then ran my container as a task on the EC2s.

Explanation if the system deviates from taught architectures, and analysis of the choices made.

The architectures taught in class that I believe are similar to my architecture are, workload distribution, my ECS cluster will distribute workloads to the available EC2 instances as needed, and dynamic scalability, this adds onto the last one, again my cluster automatically horizontally scales my EC2's.

Security Analysis

Approach to securing data through all stages of the system (in transit, at rest).

For accessing my AWS services, I am using IAM roles with the principle of least privilege, with specific permissions so that only the services that need to access something can. For example, my EC2 cluster has permissions to access my secret manager for getting the RDS credentials, but no other services can access those. For the security of my application, I am using a Virtual private cloud to store my data privately, while ensuring my application servers can be publicly accessed. VPC can be used to help prevent things such as traffic eavesdropping and malicious intermediary. My EC2 instances are protected as they do not allow SSH or public connection. They can only be accessed through the public IP.
For protecting my data, I made sure to use Bcrypt hashing for storing my passwords in the database, this is a one-way non-reversable way of securing password data. I chose Bcrypt as I know it is popularly trusted and easy to implement with React. My use of query prepared statements helps protect against attacks such as SQL injection.
My use of Docker validates code integrity of my web app.

I do believe there are vulnerabilities in my security of my application. For example, currently my app only runs on HTTP, not HTTPS, I would fix this to ensure secure http connections. I am currently not implementing much monitoring of my services. For example, when I was creating connections to my RDS database and forgetting to close those connections, my database shut down due to too many connections. I would monitor these connections and close any long running connections.
I also currently am using CloudFront to access my S3 files, however, this could be risky given that users can enter any content they would like in their posts, meaning that if there were any sensitive data in a post content, then that can be accessed through the public MP3 file. I would possibly figure out a way to double check the contents of posts before storing them in my bucket, or revert to another way of accessing those files.

Cost Metrics

Calculation of upfront, ongoing, and additional costs.

Explanation of alternative approaches for cost-saving or justification for a more expensive solution.

To build this architecture in a private cloud, with the same level of availability, the costs would be significantly higher. This would require purchasing the hardware, such as servers, storage devices, and networking equipment. As well as upkeep costs such as maintenance and management of the hardware, rent, electricity, cooling, and physical security. Extra costs would also come from needing backup plans, this might include running servers or data centers that are not currently in use. Costs can also come from training personnel to configure and upkeep these services, but costs also come from training cloud architect engineers.

According to [3] a small business would likely pay ~$1476 per month for an on-premise private cloud, vs $313 a month for a private cloud.

My cloud service that has the potential to cost the most money would be my EC2's that are my ECS cluster is running on. I would make sure to add monitoring to these services so that I can be aware of the costs and make changes accordingly. Virtual servers can often be quite costly despite how small the application running on it may seem. I would like to explore ensuring my servers can scale both horizontally and vertically, so that I may start with the minimum size and cost for a server and allow it to scale only as needed.

In addition to the EC2's building in cost, my RDS also has the potential to build in costs very quickly. This is one reason I may want to switch to DynamoDB, as this can have much lower initial costs and easier vertical scaling.

Future Additions and Reflections

Some of the intended features that have not yet been fully implemented in my application are, being able to save unfinished posts as drafts then publish to the public later, sorting the posts by category (short story, blog, etc.), liking, and commenting on posts, following other users, being able to add images as a post cover art, adding as many images throughout the post content as users like, and allowing users to customize their profile pages.

AWS services that I could use to implement these would be, a more robust and modifiable database solution such as a graph database option for a more connected and modifiable option. This would be particularly useful when implementing following, and post liking and commenting. I would use also another S3 bucket for storing images. I would implement database clusters and load balancing to ensure no downtime on my databases, so in the event of another connection overload, I can automatically recover.

If I could restart this project from scratch knowing what I know now, I would potentially redo the web app as a front end only application using React and Node.js and create a totally serverless web app environment using AWS Lambda, Step Functions, AWS API Gateway, AWS DynamoDB, AWS S3, storing the front end in the s3 with an API gateway and using lambdas for backend functions. A totally serverless application option would greatly improve my possible costs and scaling abilities.

References

[1]
“AWS | Media Sharing - Cloud File Storage for Photos & Video,” *Amazon Web Services, Inc.* https://aws.amazon.com/media-sharing/#:~:text=AWS (accessed Apr. 10, 2024).
[2]
M. Toiba, “NoSQL vs. relational: Which database should you use for your app?,” *Azure Cosmos DB Blog*, Jan. 17, 2023. https://devblogs.microsoft.com/cosmosdb/nosql-vs-relational-which-database-should-you-use-for-your-app/
[3]
T. Robinson, “A Cost Tipping Point Guide for IT Professionals: Public vs Private Cloud,” *OpenMetal IaaS*, Mar. 15, 2022. https://openmetal.io/resources/blog/public-cloud-vs-private-cloud-cost-tipping-points/