Master of Applied Computer Science

CSCI 5409

Advanced Topic in Cloud Computing

Term Assignment – Final Report

Project Title – Job Application Portal

Prepared By

Jinal Dave (B00936848)

Table of Contents

Link to Gitlab Repo

https://git.cs.dal.ca/courses/2023-fall/csci5409/jinal.git

Introduction

What is built and what it is supposed to do:

The Job Application Portal is a web-based application platform designed to make it easier for a company to receive employment applications. This application, which is built on AWS infrastructure, makes use of a few AWS services that ensure scalability, security, and effective data processing. The program processes incoming job applications using AWS Lambda on the backend and stores data securely in Amazon S3 and DynamoDB. Provisioning and deploying the entire infrastructure is done automatically with the CloudFormation script.

Overview of modules:

1. Application Interface and Data Storage:
Applicant data is stored in DynamoDB and documents are safely uploaded to Amazon S3 by means of APIs that users submit through the interface.

2. Backend Processing with Lambda:
Data is routed to DynamoDB and S3 endpoints for storage after being processed by Lambda functions.

3. AWS EC2 Hosting and Deployment:
The application is hosted on an AWS EC2 instance, utilizing CloudFormation template for automated and scalable deployment across environments.

4. Automated Notifications with SNS:
An automated email confirmation system that uses Amazon SNS notifies registered email address after a successful submission.

Services selection and Comparison with other Services

How I met the menu item requirements:

I have selected below services from each category:

Compute:

(1) AWS EC2
The lightweight HTML frontend is hosted on EC2, which I chose because it works well with Apache servers. The requirement for a dependable, adaptable, and scalable

computing environment led me to this decision. Smooth control over server configurations is made possible by EC2 instances, which makes it possible to host databases, web applications, and APIs that are specific to the needs of the company.
Additional choices: Although more sophisticated functionalities are available through services like Google Compute Engine, Beanstalk, ECS, EKS, and Azure Virtual Machines, they might be too much for my simple application. I was able to eliminate the needless complexity and resource overhead that come with these services by using EC2, which resulted in a more efficient and economical solution for our particular requirements.
Factor of cost: In the US East region, it begins at about $0.0058 per hour for a t3.micro instance.

(2) AWS Lambda
The foundation for processing incoming job applications in response to different triggers is AWS Lambda. Lambda's serverless architecture and event-driven nature make it ideal for applications where server setup and management are not required. This guarantees effective management of fluctuating incoming application loads without worrying about server capacity.
Other choices are Google Cloud Functions and Azure Functions.
Charged at $0.20 for every million requests and $0.00001667 for every gigabyte-second, the cost factor is based on 100 milliseconds of execution time.

Storage:

(1) AWS Simple Storage Service (S3)
Resumes and other related files from job seekers will be safely stored on Amazon S3. The service meets the needs of the application for document storage while guaranteeing high availability and accessibility by offering extremely robust and scalable object storage. S3 was chosen because it would ensure dependable storage for files uploaded by users.
Other choices are Azure Blob Storage and Google Cloud Storage.
Cost factor: Standard storage in the US East area starts at $0.023 per GB per month..

(2) AWS DynamoDB
The main database used to hold the information of job applicants is AWS DynamoDB. DynamoDB was chosen because it can provide single-digit millisecond performance at any scale. The application's requirements for quick and reliable performance are met by this NoSQL database, which guarantees easy and effective data handling for job submissions and associated documents.
Other possibilities are Google Cloud Firestore and Azure Cosmos DB.

Factor of cost: begins in the US East region at $1.25 per WCU and $0.25 for each RCU per month.

Network:

(1) AWS API Gateway

To manage incoming HTTP requests from the frontend of the application to the backend services, API Gateway was chosen. Its goal is to develop APIs that safely interface with AWS Lambda and additional AWS services, providing application endpoints with a scalable and controllable gateway.

<u>Alternatives include</u> Azure API Management and Google Cloud Endpoints.

Cost factor: Cost is dependent on data transported and API calls received, with the US East region charging $3.50 per million API calls.

General:

(1) AWS Backup

By automatically producing backups of DynamoDB tables, S3 buckets, and other crucial data stores utilized by the job application portal, AWS Backup was put in place to guarantee data resilience and compliance.

<u>Reason for choosing:</u> With AWS Backup, managing and restoring data across various AWS services is made simpler and more uniform. This lowers the possibility of data loss and streamlines data protection.

(2) AWS Simple Notification Service (SNS)

After a job application is successfully submitted, a subscriber receives automatic email confirmations from Amazon SNS. It guarantees scalable and dependable email confirmation alerts for users.

<u>Reason for choosing:</u> AWS SNS offers a dependable and expandable messaging service that streamlines email notification sending, improving user experience and expediting the application confirmation process.

Cost factor: The cost of this service varies according to how many messages are sent and published. In the US East, prices begin at $0.50 for a million publications and $0.06 for every 100,000 delivery.

Deployment Model: Selection and Reason

A description and reasoning of the deployment model I chose for the system:

My job application portal uses the Public Cloud deployment architecture, which makes use of Amazon Web Services (AWS) and its extensive cloud service range. Under this architecture, AWS hosts and manages all of the application's cloud infrastructure resources, including processing power, data storage, and networking.

The reason for this choice is broken down as follows:

Elasticity and Scalability: By utilizing AWS's cloud architecture, scalability is made simple, allowing the application to adjust resources to meet demand and handle variable loads.

Cost-Efficiency: By charging only for the resources used, AWS's pay-as-you-go pricing model maximizes cost efficiency by removing the need for significant upfront investments.

Global Accessibility and Reliability: Low latency access to resources from a variety of geographic areas is made possible by AWS data centres' worldwide coverage, which improves user reliability and accessibility.

Security and Compliance: To guarantee data security, encryption, and regulatory compliance, AWS provides strong security measures and compliance certifications.

Innovative Services and Ecosystem: Having access to a large range of AWS services encourages creativity and makes it possible to incorporate state-of-the-art features and technology into applications.

Delivery Model: Selection and Reason

A description and reasoning of the delivery model I chose for the system:

For my system, a hybrid cloud delivery technique that combines Infrastructure as a Service (IaaS) and Function as a Service (FaaS) has been selected. I use AWS CloudFormation for infrastructure provisioning and management (IaaS) and AWS Lambda for serverless function execution (FaaS) by utilizing Amazon Web Services (AWS).

Reasoning for Choosing FaaS + IaaS Cloud Delivery Model:

Scalability and Flexibility: Without requiring me to maintain servers, AWS Lambda, a FaaS product, gives me scalable and event-driven compute capabilities. When combined with AWS CloudFormation for IaaS, it made it simple for me to provide and configure networking, storage (S3), databases (DynamoDB), and other supporting infrastructure components, all of which were able to seamlessly adjust to the demands of my application.

Cost-Efficiency: I was able to optimize expenses by charging according to actual function executions and resource use by combining the FaaS and IaaS models. With AWS Lambda's pay-per-use pricing mechanism and CloudFormation's IaaS framework, I could allocate resources effectively and match charges to my application's actual consumption.

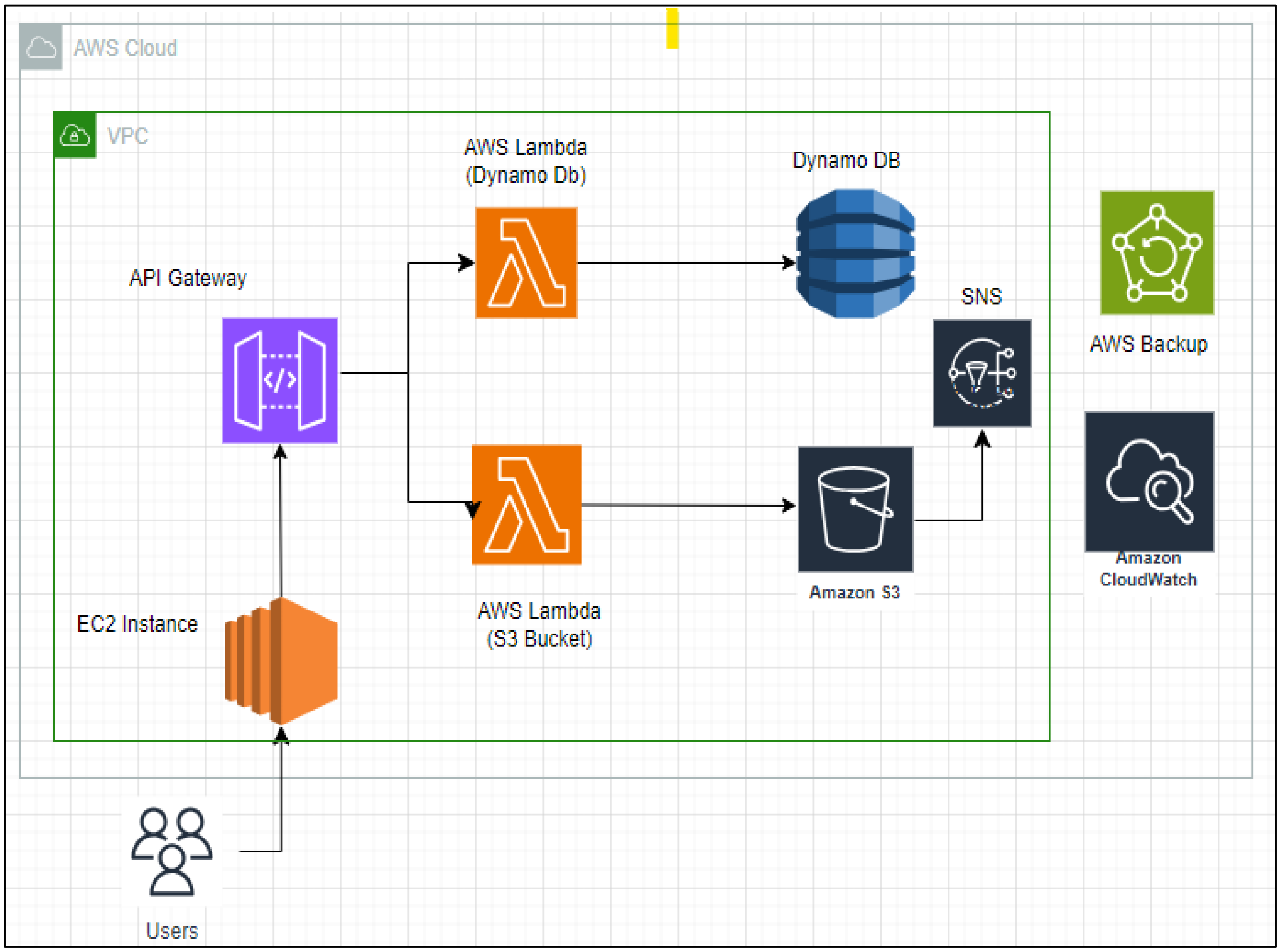
Automation and Efficiency: I was able to avoid manual intervention by automating the provisioning and administration of infrastructure resources by utilizing AWS CloudFormation's infrastructure-as-code approach. Furthermore, the serverless architecture of AWS Lambda allowed for automated resource provisioning and scaling, adapting quickly to the changing needs of my application.

Security and Compliance: The highest level of security for my built portal was ensured by AWS's strong security features throughout its FaaS and IaaS services. These features included data encryption, stringent access controls, and compliance with industry norms and regulations.

Integrated Serverless Architecture: By combining AWS Lambda with CloudFormation, I was able to create a unified environment that made it possible for me to efficiently orchestrate serverless functions and infrastructure resources for my job application platform.

Architecture

The architecture of the final system:



How do all of the cloud mechanisms fit together to deliver the application?

I have incorporated a number of essential AWS services into the development of my job application platform to ensure a productive user experience. The core of my application is

AWS EC2, which offers hosting capabilities. The backend operations of AWS Lambda take care of processing incoming applicant data and handling its storage.

I have used AWS S3 for safe document storage, and AWS DynamoDB is thought to be a dependable storage option for structured applicant data. I ensure effective data processing by forwarding incoming requests to the relevant backend Lambda functions using AWS API Gateway.

AWS Backup regularly backs up important data to protect my system, and AWS SNS manages automated email notifications to give me a dependable line of contact.

Lastly, the deployment procedure across several environments is simplified and automated via the AWS CloudFormation script.

Because of these AWS services working together, I've been able to integrate computing, storage, network, and automated processes to develop a job application platform that is safe and scalable**.**

Where is data stored?

Documents are safely saved in Amazon S3, making uploaded files easily accessible, while data is stored in DynamoDB for structured applicant details, offering scalability and reliability.

What programming languages did I use and why, and what parts of the application required code?

The front end of the program is developed with HTML, CSS, and JavaScript to build a user interface.

For backend processing, I've been using Python 3.8 in my Lambda functions. This programming language was selected because of how well it handles data and integrates with AWS services like AWS SNS.

The project's front-end form submission handling, back-end Lambda function processing, and API integrations are the areas where code is mostly utilized.

How is the system deployed to the cloud?

The deployment to the AWS cloud is carried out using the CloudFormation script to set up the infrastructure of the application as code, according to the project requirements. Frontend files are stored in an Amazon S3 bucket that is set up to host static websites. The application is hosted on an AWS EC2 instance, and backend processing is handled using AWS Lambda functions. To direct incoming requests to the relevant Lambda functions, an API Gateway is set up. Amazon DynamoDB is used for structured data storage, and AWS S3 is used for document storage that is secure. Critical data is regularly backed up using AWS Backup. Upon submission success, automated notifications are managed using AWS SNS.

Whether the system matches the architecture taught in the course or not.
My system's architecture, which makes use of cloud-native technologies like AWS Lambda, DynamoDB, S3, EC2, and API Gateway, is in line with the concepts covered in the course.

Data Security

How does the application architecture keep data secure at all layers? If it does not, if there are vulnerabilities, please explain where the data is vulnerable and how you could address these vulnerabilities with further work.

User Interaction Layer:
I've created HTTPS for safe communication at the front-end using HTML, CSS, and JavaScript. However, there could be weaknesses in the validation of user input. Although HTML input types offer simple safety controls, improving client-side validation can strengthen security.

Backend Infrastructure:
My backend configuration employs AWS IAM for access control, configured with the principle of least privilege, and involves AWS S3, AWS Lambda, and AWS SNS. Enhancing security at this layer could be further achieved by encrypting sensitive data before sending it via SNS.

Network Security:
Ensuring secure data transmission is essential for the security of the application. To handle this, the network layer requires all data to travel over HTTPS, a secure channel. It simulates a data safety tunnel. AWS API Gateway controls the application's communication with the server. Continuous monitoring ensures that just the appropriate amount of data passes through, protecting it from potential attacks and enhancing the app's security.

Data Handling and Processing:
I handle and work with user data at this layer using AWS Lambda and SNS services. It is essential to ensure that this data is trustworthy and protected from unauthorized alterations. My focus is on enhancing Lambda's security checks to ensure that all received data is safe from potential threats.

Which security mechanisms are used to achieve the data security described in the previous question? List them, and explain any choices you made for each mechanism (technology you used, algorithm, cloud provider service, etc.)

Below is a concise breakdown of the security mechanisms used in my application:

AWS Lambda (IAM Roles): Using the least privilege principle, I assign IAM roles to Lambda functions, limiting their access to particular AWS resources. Functions will only be able to access what they require due to this provision.

AWS S3 (Bucket Policies): By specifying who can interact with the content and controlling access to it, S3 bucket policies allow me to maintain data security.

AWS API Gateway (HTTPS): I configure API Gateway to use REST APIs, encrypting data during transit between clients and APIs, and safeguarding against eavesdropping or data interception.

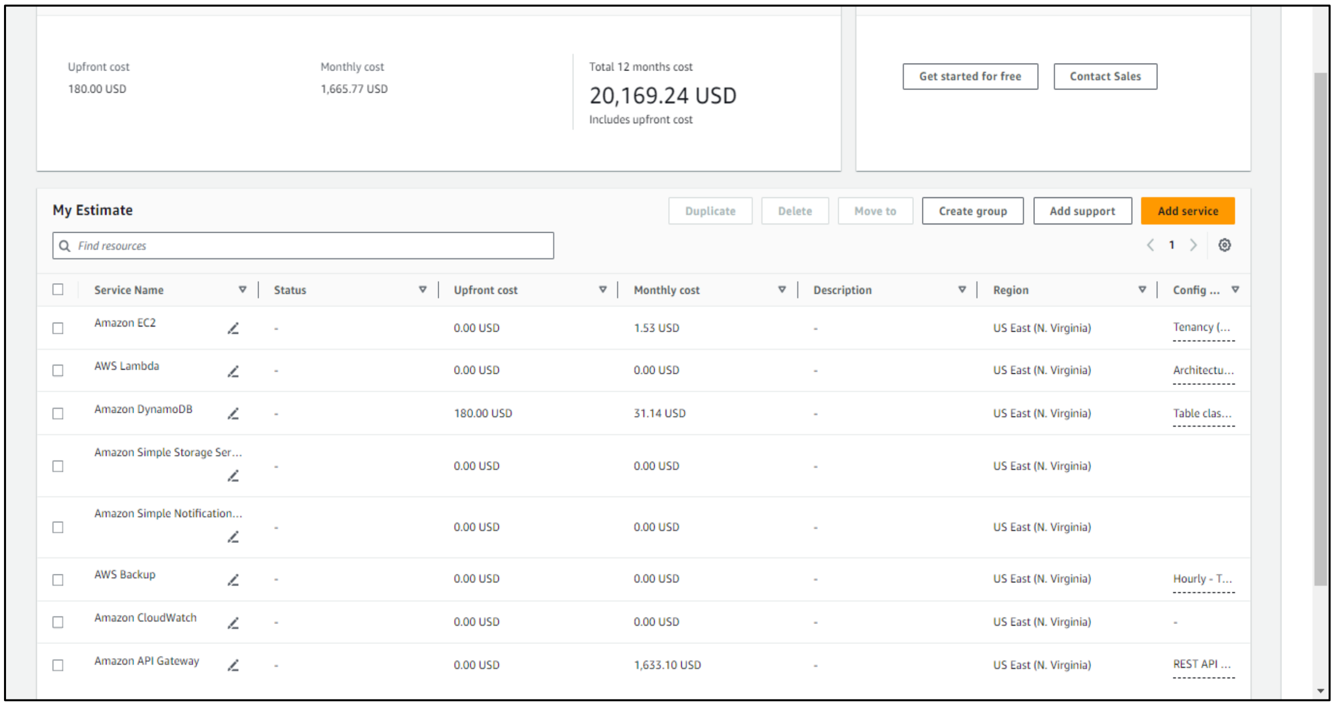
AWS Backup (Automated Backups): This ensures data redundancy and acts as a failsafe against unintentional loss or corruption.

AWS SNS (Access Control): Securing communication channels, limiting access to SNS topics, and ensuring that only authorized entities can publish or subscribe to topics are all accomplished by using IAM policies.

Cost Metrics

An analysis of the cost metrics for operating the system:

Below is the approximate cost analysis of operating the system based on the services I have used:



Upfront cost 180.00 USD

Monthly cost 1,665.77 USD

Total 12 months cost 20,169.24 USD
Includes upfront cost

My Estimate

Service Name	Status	Upfront cost	Monthly cost	Description	Region	Config ...
Amazon EC2	-	0.00 USD	1.53 USD	-	US East (N. Virginia)	Tenancy (...
AWS Lambda	-	0.00 USD	0.00 USD	-	US East (N. Virginia)	Architectu...
Amazon DynamoDB	-	180.00 USD	31.14 USD	-	US East (N. Virginia)	Table clas...
Amazon Simple Storage Ser...	-	0.00 USD	0.00 USD	-	US East (N. Virginia)	
Amazon Simple Notification...	-	0.00 USD	0.00 USD	-	US East (N. Virginia)	
AWS Backup	-	0.00 USD	0.00 USD	-	US East (N. Virginia)	Hourly - T...
Amazon CloudWatch	-	0.00 USD	0.00 USD	-	US East (N. Virginia)	-
Amazon API Gateway	-	0.00 USD	1,633.10 USD	-	US East (N. Virginia)	REST API ...

Reproducing Cloud Architecture On-Premise

What would your organization have to purchase to reproduce your architecture in a private cloud while providing relatively the same level of availability as your cloud implementation? Try to give a rough estimate of what it would cost, don't worry if you are far off. These systems are complicated and you don't know all the exact equipment and software you would need to purchase. Just explore and try your best to figure out the combination of software and hardware you would need to buy to reproduce your app on-premise.

Hardware:

Servers: Servers are essential for hosting application, web, and database services. Dell or HP servers could be considered as options for medium-scale apps, with prices ranging from $5,000 - $10,000 per server.

Storage Devices: Network Attached Storage (NAS) for data storage and retrieval can be considered with Synology or QNAP which could be viable options, ranging from $5,000 - $10,000.

Options: Synology DiskStation, QNAP TS-x53D. The price is estimated between $5,000 - $10,000.

Software:

Operating Systems: For server functionality, consideration can be Linux distributions or Windows Server Datacenter. The cost per server may vary from $1,000 to $3,000.

Options: Windows Server Datacenter Edition, Ubuntu Server, CentOS.

Database Software: Databases for managing data that come in free versions, such as PostgreSQL or MySQL.

Options: MySQL Enterprise Edition, PostgreSQL. Prices is estimated between $3,000 - $7,000 per database for paid versions.

Web Server Software: Microsoft IIS or Apache HTTP Server are used to provide web content. The price may vary from $0 to $2,000 based on the platform that is selected.

Serverless Framework: An open-source framework that supports several cloud providers and performs a similar role to AWS Lambda may be of interest to the enterprise. Among the choices are IronFunctions, OpenFaaS, and Kubeless.

Security:

Firewall and Security Software: For network security, use Cisco ASA or Fortinet FortiGate can be used. The price range is between $3,000 and $8,000.

Encryption Tools: Symantec Endpoint Encryption and McAfee Total Protection are two examples of endpoint encryption tools that are included here. Expenses may be between $500 and $2,000.

Networking:

Redundant Network Infrastructure: Some solutions with prices ranging from $5,000 to $15,000 or more are the Cisco Catalyst and Juniper EX Series for redundancy.

Internet Connectivity: The price of internet connectivity varies according to location and required bandwidth. The estimated monthly range is between $500 and $2,000. Options include fibre optic networks and local ISPs.

Monitoring and Logging:

Monitoring Tools: The company may consider using free and open-source programs like Zabbix or Nagios, which may not require a license but do require some setup and configuration work.

Backup and Disaster Recovery:

Backup Solutions: Depending on the functionality and volume of data, Veeam Backup or Commvault may cost $5,000 to $10,000 or more for data protection.

High Availability Setup: he anticipated cost for medium installations ranges from $50,000 to $100,000+, depending on hardware redundancy and complexity.

In summary, the initial setup costs are estimated to be between $70,000 and $120,000. Annual operating expenses, on the other hand, may be as high as $90,000, depending on the size and specific needs of the application.

Monitoring for Cost Control

Which cloud mechanism would be most important for me to add monitoring to in order to make sure costs do not escalate out of budget unexpectedly? In other words, which of your cloud mechanisms has the most potential to cost the most money?

AWS EC2 and AWS Lambda are the cloud methods that could cost me the most in my configuration, based on the services I have used. It is essential to keep an eye on these services for cost management reasons to avoid unforeseen budget overruns.

Costs for AWS EC2 instances can build up depending on the data sent, operating hours, and instance types selected. Using Auto Scaling to control instance counts dynamically, optimizing instance sizes based on workload requirements, and putting monitoring tools in place to track instance utilization can all help save needless costs.

In the same way, compute time and the number of invocations are used to determine AWS Lambda costs. Therefore, keeping an eye on Lambda invocations, streamlining code to run faster, and analysing how often code is executed can all help with cost management.

Thus, I can proactively control consumption patterns, optimize resource utilization, and guarantee that expenses stay within budgetary restrictions by putting monitoring and tracking methods for AWS EC2 and Lambda services in place.

Future Development and Feature Advancements

How would my application evolve if I were to continue development? What features might I add next and which cloud mechanisms would I use to implement those features?

The application's evolution will be focused on improving scalability, user experience, and backend capabilities as it acquires growth and user involvement increases. Here is my refined approach:

Personalized Notifications:
What I'm planning: Customizing emails according to applicants' actions or progress will make them feel more personalized.

How it works: Throughout the application process, I will use intelligent technologies like Lambda functions to modify email content to keep applicants informed and interested.

Better User Interface (UI):
What I aim for: My goal is to make the application look and feel better while ensuring it is responsive and easy to use.

What I'll do: To jazz up the user interface and give users a more tailored experience and faster loading times, I'm thinking about incorporating dynamic features.

Live Chat Support:
What I'm exploring: Providing prompt assistance throughout the application for any queries or needs that candidates may have.

How it helps: In order to ensure that candidates receive assistance whenever they need it during the application process, I'm considering adding a live chat facility.

Updates on Application Status:
Keeping applicants informed about their position in the application process can improve user satisfaction. Applicants can check their application status anytime. As I strive for openness

and ongoing communication with them during their application process, my apps will keep them informed of any updates.

References

[1] “Flowchart maker & online diagram software,” Diagrams.net. [Online]. Available: https://app.diagrams.net/. [Accessed: 30-Nov-2023].

[2] “AWS Pricing Calculator,” Calculator.aws. [Online]. Available: https://calculator.aws/#/estimate. [Accessed: 30-Nov-2023].

[3] Amazon.com. [Online]. Available: https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/intrinsic-function-reference-ref.html. [Accessed: 30-Nov-2023].

[4] Amazon.com. [Online]. Available: https://docs.aws.amazon.com/whitepapers/latest/optimizing-enterprise-economics-with-serverless/understanding-serverless-architectures.html. [Accessed: 30-Nov-2023].

[5] K. Jones, “Four reasons you should consider going serverless,” Forbes, 04-Nov-2021. [Online]. Available: https://www.forbes.com/sites/forbestechcouncil/2021/11/04/four-reasons-you-should-consider-going-serverless/?sh=620280f869cf. [Accessed: 30-Nov-2023].

[6] “Week 4 (2) - Cloud Delivery Models - Original,” Brightspace.com. [Online]. Available: https://dal.brightspace.com/d2l/le/content/287433/viewContent/3911606/View. [Accessed: 30-Nov-2023].

[7] “QNAP,” Qnap.com. [Online]. Available: https://www.qnap.com/static/landing/2020/en-diwali-ts-x53D/index.html. [Accessed: 30-Nov-2023].

[8] Wikipedia contributors, “Juniper EX-Series,” Wikipedia, The Free Encyclopedia, 04-Sep-2023. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Juniper_EX-Series&oldid=1173736649.

[9] “Serverless Open-Source Frameworks: OpenFaaS, Knative, & more,” CNCF, 13-Apr-2020. [Online]. Available: https://www.cncf.io/blog/2020/04/13/serverless-open-source-frameworks-openfaas-knative-more/. [Accessed: 30-Nov-2023].

[10] JasonGerend, “Comparison of Standard and Datacenter editions Windows Server 2019,” Microsoft.com. [Online]. Available: https://learn.microsoft.com/en-us/windows-server/get-started/editions-comparison-windows-server-2019?tabs=full-comparison. [Accessed: 30-Nov-2023].