ARCHITECTURAL CRITICAL ANALYSIS AND RESPONSE

CSCI 5409: CLOUD COMPUTING

Summer 2022

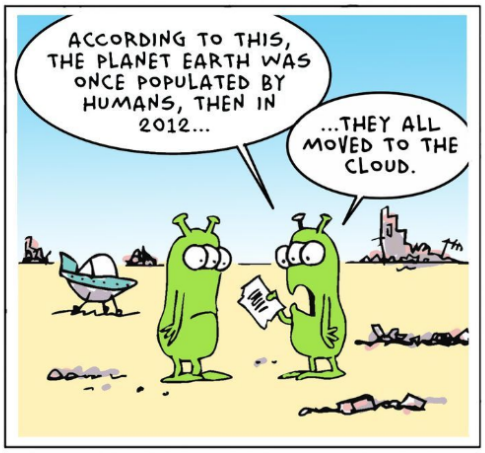

Image source: AT&T cybersecurity blogs

Prof. Robert Hawkey,
Instructor & Faculty at Computer Science

Assigned TA:
Kamran Awaisi

GROUP 16: CLOUDVENGERS

MEMBERS:

Ruhi Rajnish Tyagi
B00872269, rh679297@dal.ca

Adil Dinmahamad Otha
B00900955, ad842343@dal.ca

Saurabh Jayeshbhai Das
B00911733, sr850847@dal.ca

TABLE OF CONTENTS

1. ARCHITECTURAL DIAGRAM:

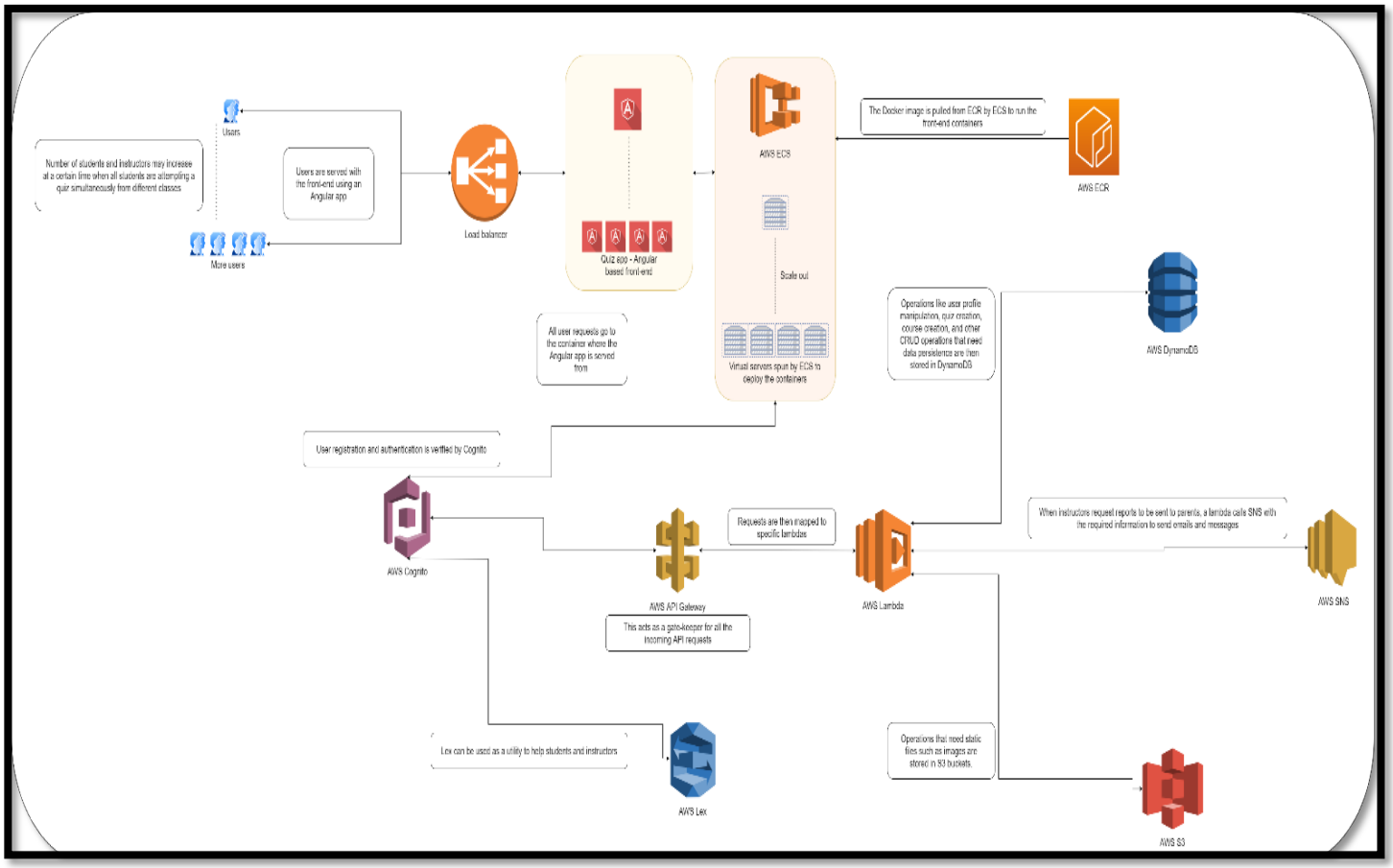

Figure 1. *Architectural diagram of the quiz application.*

Note: For better visibility, we have also submitted a PNG image at Brightspace submission box.

1.1 DATA FLOW IN THE SYSTEM:

1.1.1 Interaction of User with the front-end app:

The User Interface is served through a frontend web-application made using Angular and deployed using AWS Elastic Container Service (ECS) and AWS Elastic Container Registry (ECR). ECS provides auto-scaling functionality to scale out IT Resources to meet the high demand of users during peak times.

1.1.2 Interaction of app with AWS Cognito:

User Authentication and Authorization is done through AWS Cognito. Cognito handles the tasks such as validating and storing user information and maintaining the session by storing the JWT (JSON Web Tokens) Tokens, checking their validity and automatically refreshing Access Tokens through Refresh Tokens.

1.1.3 Interaction of app with API Gateway:

API Gateway will act as the bridge between the frontend and backend (Lambda functions). It will provide an HTTP-based RESTful API [2]. It will utilize JWT Authorizers for authorizing the user requests.

1.1.4 Interaction of AWS API Gateway with AWS Lambda:

The AWS API Gateway acts as a gatekeeper that gets the requests to perform back-end operations. All the back-end requests are then mapped to their respective Lambda functions. An example of the back-end API request can be:

- A POST request on /users route.
- This then routes to a Lambda.
- The lambda then takes care of the operation furthermore.

1.1.5 Interaction of AWS Lambda with AWS DynamoDB:

After getting a trigger from the API gateway for a specific operation, the specific Lambda will take care of the backend implementation. For instance, if the request if for creating a quiz, the API gateway will receive a POST request let's say on the /quiz route. Then the lambda pertaining to this request will get handle the whole operation of properly storing the quiz questions and answers in the DynamoDB.

1.1.6 Interaction of AWS Lambda with AWS SNS:

A feature of our app lets instructors send the marks of the students to their parents / guardians. When the instructor will click on the button on the frontend of our app to send the reports, the progress report of that student will be sent on the parent's email. This will be handled by a Lambda that will trigger the Lambda to carry out the email sending task.

1.1.7 Interaction of AWS Lambda with AWS S3:

Users of our app can store their profile picture. To store such static assets, we have decided to store them on S3 from where they can be easily retrieved.

1.1.8 Interaction with AWS Lex:

New users of our app may like to know how our app functions and offers its features. For this, AWS Lex can guide them so they can understand how to use the app. For instance, a student would like to know how he / she can attempt a quiz that is due in some days. Lex will guide the student. Also, at times some teachers might want to send the marks of a particular student without going through the list of all students, they can do so using the chatbot.

2. KEY CLOUD COMPONENTS:

2.1 Load Balancer:

The load balancer will act as the main entry point for incoming user requests. It will distribute the traffic among different virtual servers to maintain high availability. This will ensure that the application can perform efficiently even when a multitude of users (students and teachers) are accessing it through various departments.

2.2 Elastic Container Service (ECS):

ECS will serve the containerized Frontend Angular Application. The main benefit of ECS is that it internally uses EC2 instances to run the docker containers and can automatically scale them out (horizontally) based on the demand of the users. For example, during exam period, multiple classes belonging to various departments may access the application simultaneously. ECS's auto-scaling will be highly beneficial during such scenarios to ensure a smooth user experience by maintaining high availability.

2.3 Lambda:

Using lambda allows us to write our back-end code without worrying about orchestrating a server. With its fault-tolerance and automatic scaling [7], we are at ease of deploying just the code and giving all the orchestration worries on AWS.

3. ARCHITECTURE SIMILAR TO OUR SYSTEM:

We figured out that the architecture that is most like our system is the "Workload Distribution Architecture". This is because this architecture works on the notion of horizontal scaling. When the load increases, it spins up new instances of the same type. This relates with our app wherein when more users are requesting the front-end app and we need to serve the app as quickly as possible to multiple users without any service disruption, we can use this.

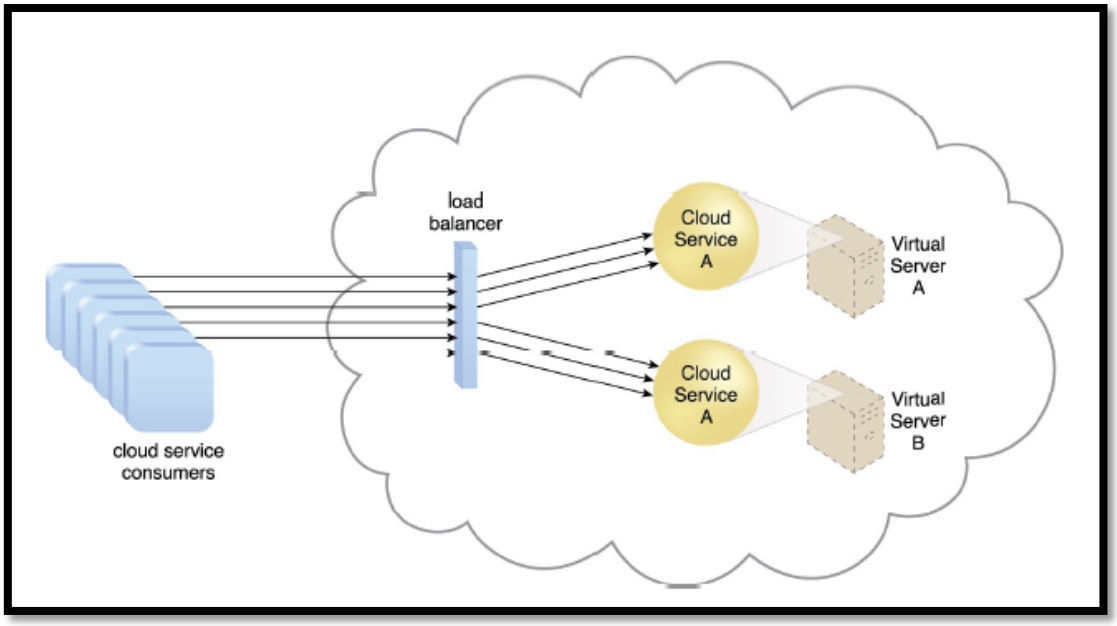

Figure 2. *The Workload Distribution Architecture [4]*

3.1 SIMILARITIES:

- The use-case of this architecture matches with our where it is used in places requiring high-availability.

3.2 DIFFERENCES:

- Instead of virtual servers, our model talks about the docker containers that are being spun up on more requests.
- Under the hood, these docker containers that are deployed by ECS use EC2 too, but we are just abstracted away by the bare usage of EC2 by using ECS [3].

4. ALTERNATIVES TO THE ARCHITECTURE:

4.1 Architecture which provides more availability and better performance:

- Cloud Balancing Architecture:

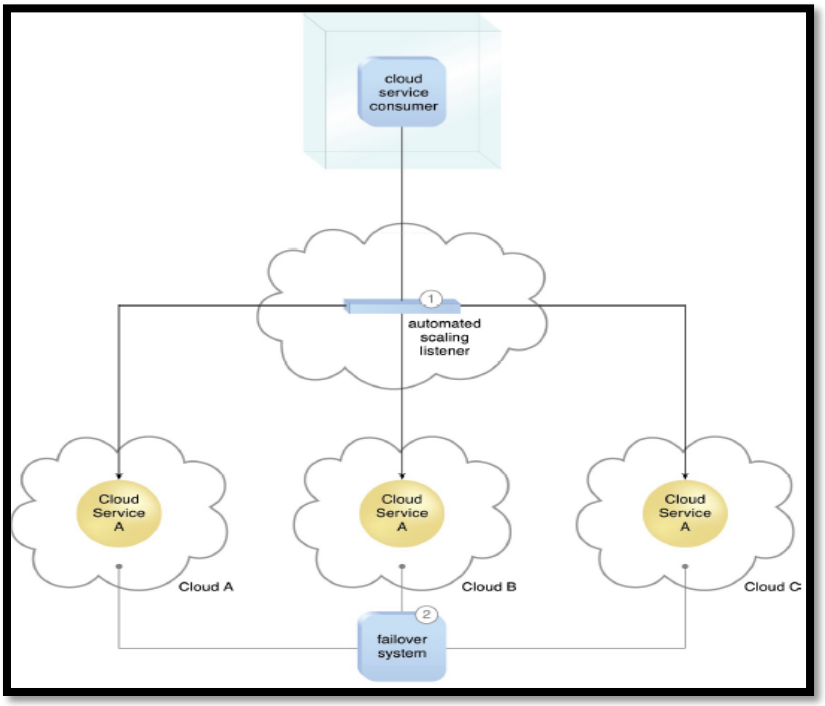

Figure 3. *Cloud Balancing Architecture [4]*

The basic goal of load balancing is to keep many servers from becoming overworked and perhaps malfunctioning. In other words, load balancing raises the availability of services and lessens the likelihood of outages [6] (Erl, Mahmood and Puttini, 2014). We might think of load balancing as the "Traffic Cop" who controls the potentially chaotic atmosphere in the cloud. To guarantee the servers' constant availability and efficient operation, it also runs health checks.

For instance, in AWS, Elastic Load Balancing, distributes the incoming traffic automatically by splitting among numerous targets, including EC2 instances, containers, and IP addresses in one or more Availability Zones.

- Dynamic Failure Detection and Recovery Architecture:

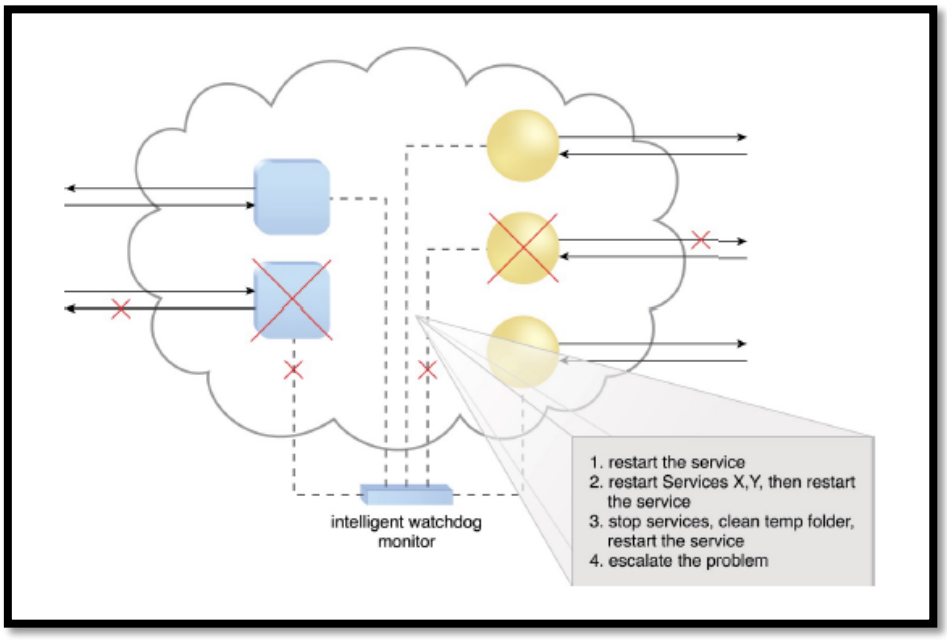

Figure 4. *Dynamic Failure Detection and Recovery Architecture [5]*

If we believe that our application has developed to the point where it now allows students from many universities to take tests, then we are unable to negotiate any performance because exams are far more significant than internal university quizzes. We might need quick recovery from such a situation if thousands of students are taking exams at the same time as a server crashes. The dynamic failure detection and recovery design can be useful in this situation.

4.2 Architecture that provides more scalability:

- Dynamic Scalability Architecture:

Since unused IT resources are successfully reclaimed without the need for manual intervention, dynamic allocation permits varied utilization as determined by consumption demand changes [6] (Erl, Mahmood and Puttini, 2014). When new IT resources must be added to the workload processing, the automated scaling listener is configured with workload thresholds. Based on the provisions of a specific cloud

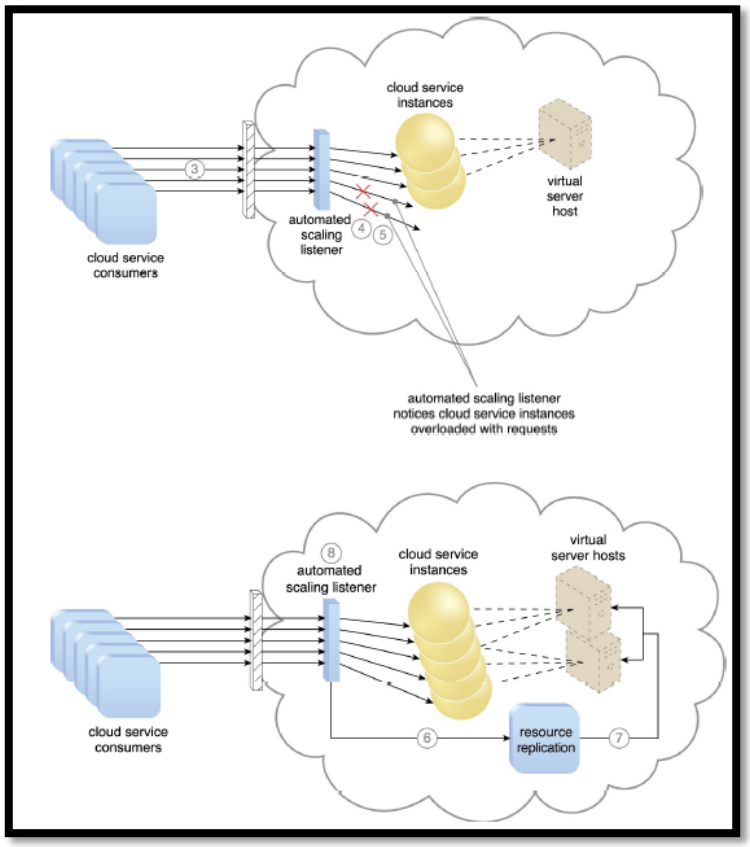

***Figure 5**. The cloud balancing architecture [4]*

consumer's provisioning contract, this method can be equipped with logic that establishes how many more IT resources can be dynamically offered. Dynamic scalability allows vertical, horizontal scaling as well as relocation of resources to higher capacity one.

Possible scenarios where this architecture will benefit our app:

- Building and rendering – maybe three.js and high graphic-rendering sites.
- Heavy bundle size for first successful build of the Angular app.

5. REFERENCES:

[1] Draw.io, "Flowchart Maker & Online Diagram Software," Diagrams.net, 2022. [Online]. Available: https://app.diagrams.net/. [Accessed: Jun. 25, 2022].

[2] "What is Amazon Api Gateway? - AWS documentation." [Online]. Available: https://docs.aws.amazon.com/apigateway/latest/developerguide/welcome.html. [Accessed: Jun. 25, 2022].

[3] T. Salah, M. J. Zemerly, C. Y. Yeun, M. Al-Qutayri and Y. Al-Hammadi, "Performance comparison between container-based and VM-based services," 2017 20th Conference on Innovations in Clouds, Internet and Networks (ICIN), 2017, pp. 185-190, doi: 10.1109/ICIN.2017.7899408 [Accessed: Jun. 25, 2022].

[4] "Fundamental Cloud Computing Architectures - CSCI4145 & CSCI5409 - Cloud Computing (Sec 1) - 2022 Summer," *Brightspace.com*, 2022. [Online]. Available: https://dal.brightspace.com/d2l/le/content/222417/viewContent/3008534/View. [Accessed: Jun. 25, 2022]

[5] "Advanced Cloud Computing Architectures - CSCI4145 & CSCI5409 - Cloud Computing (Sec 1) - 2022 Summer," *Brightspace.com*, 2022. [Online]. Available: https://dal.brightspace.com/d2l/le/content/222417/viewContent/3008535/View. [Accessed: Jun. 25, 2022]

[6] Erl, T., Mahmood, Z. and Puttini, R., 2014. *Cloud computing*. 1st ed. Upper Saddle River: Prentice Hall. [Accessed: Jun. 25, 2022].

[7] "Serverless Computing - AWS Lambda Features - Amazon Web Services," *Amazon Web Services, Inc.*, 2022. [Online]. Available: https://aws.amazon.com/lambda/features/. [Accessed: Jun. 25, 2022]