

FINAL REPORT

CSCI 5409: CLOUD COMPUTING Summer 2022

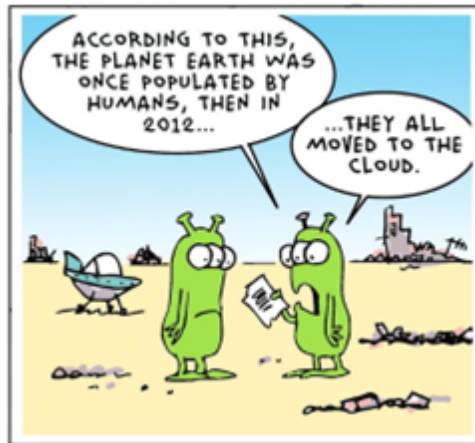


Image source: [AT&T cybersecurity blogs](#)

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

INDEX

1. Describe your final architecture.....	3-15
2. If your final architecture differs from your original project proposal, explain why that is. What did you learn that made you change technologies or approaches?.....	16
3. How would your application evolve if you were to continue development? What features might you add next and which cloud mechanisms would you use to implement those features?.....	17
4. How does your application architecture keep data secure at all layers? If it does not, if there are vulnerabilities, please explain where your data is vulnerable and how you could address these vulnerabilities with further work.....	18
5. 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.).....	19
6. 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.....	20-22
7. Which cloud mechanism would be most important for you 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?.....	22
8. References.....	24-25

1. Describe your final architecture:

Overview:

The motive of our app was to provide a SaaS to universities and colleges. While developing the application, we quickly realized how cumbersome is managing lambdas. So, in order to help dev tooling support and easier deployments, we have used the industry standard in serverless computing which is the “Serverless framework” [1]. The backend of our app is scaffolded using this serverless framework. It creates an API gateway for all our lambdas and automatically deploys the API gateway along with the lambdas with just a single command. Apart from this we have been using AWS Cognito, DynamoDB, S3, ECR, ECS, SNS, Lex, and CloudFormation.

Final Architecture:

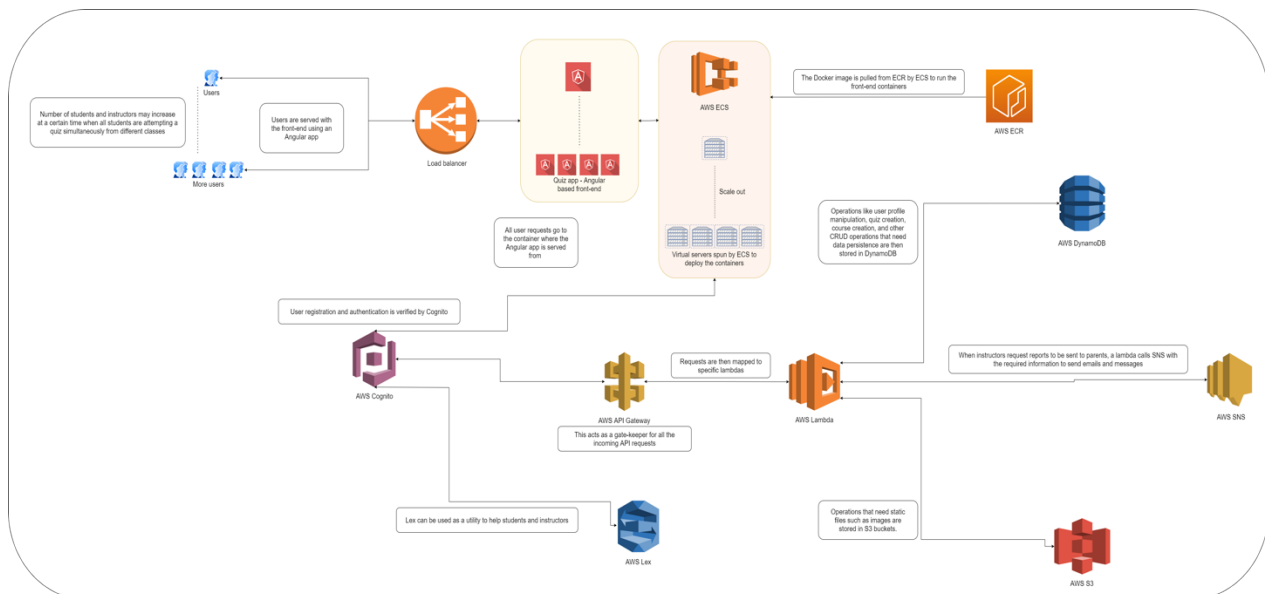


Figure 1. Final architecture of the application.

DATA FLOW IN THE SYSTEM:

- 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.

- 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.

- 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.

- 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.

- 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 is 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.

- 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.

- 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.

- 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.

KEY CLOUD COMPONENTS:

- **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.

- **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.

- **Lambda:**

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

Cloud mechanisms' connection to deliver our app:

To have our app up and running here is how each cloud mechanism works internally and is connected to the other components to deliver our required functionalities:

AWS API Gateway [2] – It acts as a gatekeeper for all our backend API requests. The backend of our app is a NodeJS-Express app that is wrapped around with serverless framework. We have a base URL that's the invocation URL of all our API endpoints. Each controller in the express app is scaffolded as a lambda on deployment. With this in place, whenever we call an API from the frontend, API gateway routes the request to the appropriate lambda that is responsible for handling the request and serving the desired functionality. All CRUD operations and functionalities required such as course creation, quiz attempts, quiz making, etc are API requests routed through API gateway.

AWS Lambda [3] – With API gateway as the route handler of all API requests, each individual request is then put over to the lambda that is responsible for processing the backend logic of database calls or some operations and returning a JSON response.

AWS Cognito [4] – Cognito offers a fully-managed authentication system wherein we store all our user authentication details. From the frontend, we just make calls to the

functions of Cognito using the official Cognito JavaScript API [5]. Cognito performs the heavy lifting of checking if the password matches during authentication or user already exists during registration and even logging out. To maintain sessions Cognito also gives the JWT token on a successful user login. To handle all these registration and login functionalities, we call methods from the official library passing in the pool id and client id of Cognito which then performs the underlying functionalities. After successfully registering with the app, to store more user details, we call up a “post signup confirmation” lambda trigger on Cognito that stores the data in DynamoDB.

AWS DynamoDB [5] – To persists the data of instructors, students, courses, quizzes, attempted quizzes we are using DynamoDB. From the frontend we make API calls that trigger specific lambdas. These lambdas then perform the required operations with DynamoDB.

AWS ECR [6] – Elastic Container Registry acts like Docker Hub wherein it stores our docker image for the containerized frontend application.

AWS ECS [7] – Elastic Container Service is used to deploy our containerized frontend application. An ECS Cluster is created inside which an ECS Service is created for storing ECS Task Definitions. This task definition will execute the docker container to serve our frontend application on a public IP address.

AWS SNS [8]– Simple Notification Service is used to send Student Reports via email. The students will receive an email asking to subscribe to this email service upon registration. A Filter Policy is set so that only the specific student will receive their evaluation report.

AWS Lex [9]– Lex is acting as an online support for user – teacher or student- who are new to the system. Lex interacts with users through intents and the intents are fulfilled by Lambda functions. It has been deployed to the website using CloudFormation deployment template stack available on official website [11].

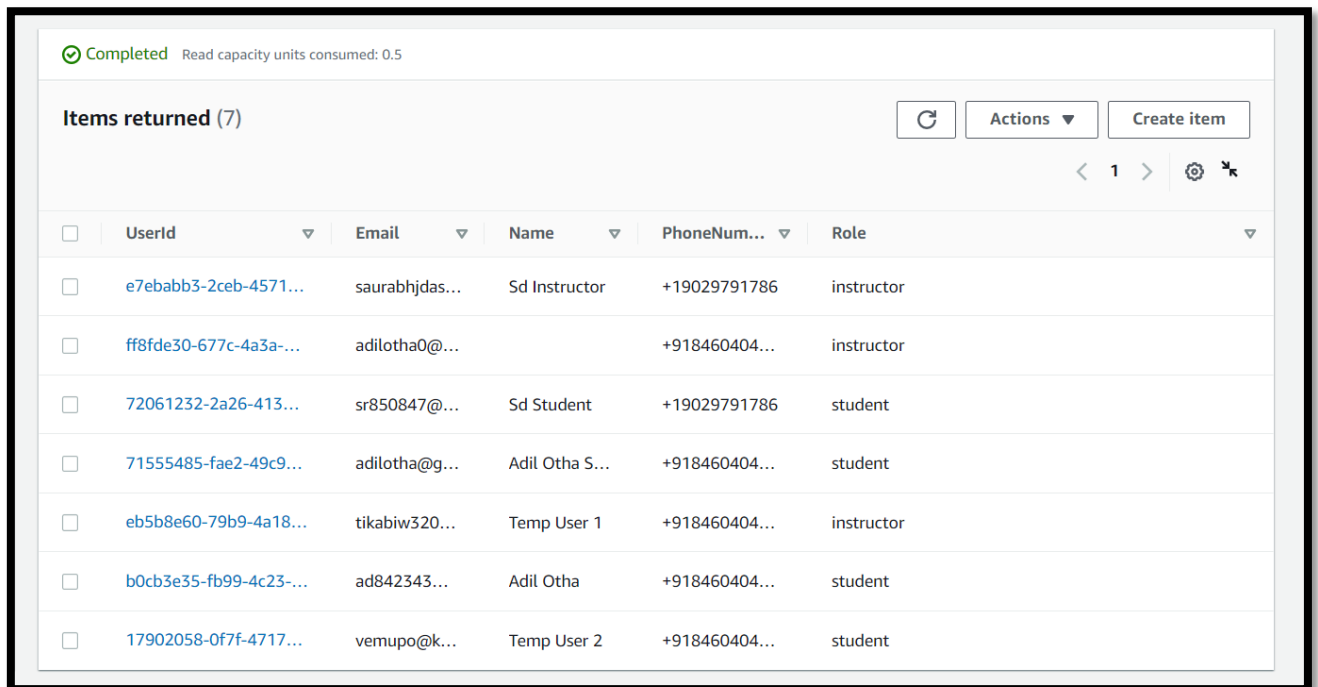
AWS CloudFormation [10] – CSCI5409’s project requirement enlists mandatory (Infrastructure as Code) IaC using AWS CloudFormation. As we are using the serverless framework for BaaS (Backend-as-a-Service), when we use the CLI command “serverless deploy”, the API gateway gets deployed with all the lambdas as well. For rest of our services, we are using another JSON file cloud formation file that scaffolds the infrastructure.

Data storage

To store data for all the students, instructors, courses, quizzes, quiz attempts our app utilizes the NoSQL database provided by AWS which is DynamoDB. Here is a description of all the tables in DynamoDB:

- Users – On successful registration, the “Post signup confirmation” lambda puts a new item in this table that stores the following attributes of the newly registered user:

Field	Description
UserId	A unique UUID generated while registration provided by Cognito
Email	The email id of the user
Name	Name of the user
PhoneNumber	Phone number of the user
Role	Instructor or Student



Completed Read capacity units consumed: 0.5

Items returned (7)

Actions Create item

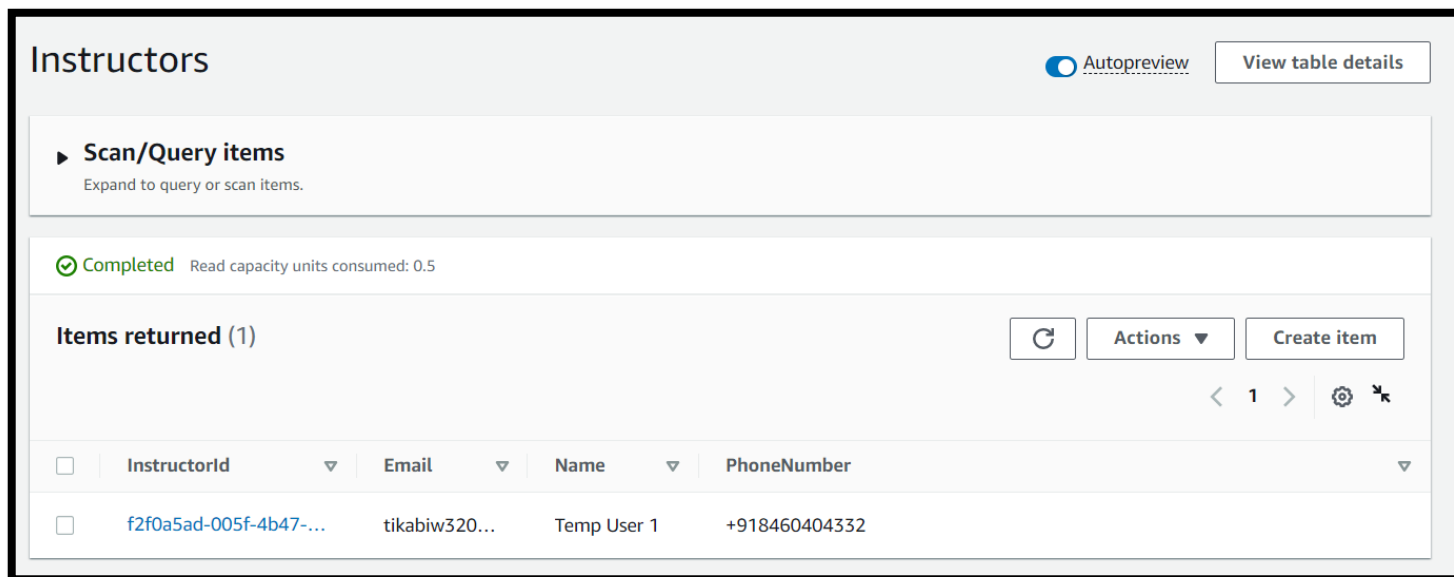
< 1 > ⚙️ 🔍

<input type="checkbox"/>	UserId	Email	Name	PhoneNum...	Role
<input type="checkbox"/>	e7ebabb3-2ceb-4571...	saarabhjd...	Sd Instructor	+19029791786	instructor
<input type="checkbox"/>	ff8fde30-677c-4a3a-...	adilotha0@...		+918460404...	instructor
<input type="checkbox"/>	72061232-2a26-413...	sr850847@...	Sd Student	+19029791786	student
<input type="checkbox"/>	71555485-fae2-49c9...	adilotha@g...	Adil Otha S...	+918460404...	student
<input type="checkbox"/>	eb5b8e60-79b9-4a18...	tikabiw320...	Temp User 1	+918460404...	instructor
<input type="checkbox"/>	b0cb3e35-fb99-4c23-...	ad842343...	Adil Otha	+918460404...	student
<input type="checkbox"/>	17902058-0f7f-4717...	vemupo@k...	Temp User 2	+918460404...	student

Figure 2. Attributes of a user in the "Users" table

- Instructors – As the users table is generic for all the registered users, we needed to query just the instructors for our app with a unique id for each of them. For this reason, we have the “Instructors” table that stores the following fields:

Field	Description
InstructorId	A unique UUID generated while registration provided by Cognito
Email	The email id of the instructor
Name	Name of the instructor
PhoneNumber	Phone number of the instructor



Instructors Autopreview [View table details](#)

► **Scan/Query items**
Expand to query or scan items.

✔ **Completed** Read capacity units consumed: 0.5

Items returned (1) Refresh Actions ▼ Create item

< 1 > ⚙️ 🔍

<input type="checkbox"/>	InstructorId ▼	Email ▼	Name ▼	PhoneNumber ▼
<input type="checkbox"/>	f2f0a5ad-005f-4b47-...	tikabiw320...	Temp User 1	+918460404332

Figure 3. Fields for each instructor in the "Instructors" table

- Students - As the users table is generic for all the registered users, we needed much more fields for the students stored in our database. For this reason, we have a separate "Students" table that has the following fields in it:

Field	Description
StudentId	A unique UUID generated while registration provided by Cognito
AttemptedQuizzes	An array of all the quizzes attempted by the student. When student attempts a quiz, a unique UUID is created for that instance of the attempted quiz.
CoursesEnrolled	An array of all the courses enrolled by the student. This array contains the CourseIds
Email	Email id of the student
Name	Name of the student
PhoneNumber	Phone number of the student



<input type="checkbox"/>	StudentId ▾	AttemptedQuizzes ▾	CoursesEnrolled ▾	Email ▾	Name ▾
<input type="checkbox"/>	72061232-2a26-413...	[]	[{"S": "35516161...	sr850847@...	Sd Student
<input type="checkbox"/>	71555485-fae2-49c9...			adilotha@g...	Adil Otha S...
<input type="checkbox"/>	b0cb3e35-fb99-4c23-...	[{"S": "d246e16c-9...	[{"S": "1ffd640f-6...	ad842343...	Adil Otha D...
<input type="checkbox"/>	ca119bd9-1fa...  	[{"S": "b1312c92-9...	[{"S": "405ad54c-...	lkbe48g3m...	Student 4
<input type="checkbox"/>	17902058-0f7f-4717...	[{"S": "da36684d-7...	[{"S": "1ffd640f-6...	vemupo@k...	Temp User ...
<input type="checkbox"/>	540b82f5-34bb-42db...	[{"S": "b4d1d2f0-e...	[{"S": "983339b7...	walamad63...	Student 1

Figure 4. Field for the "Students" table

- Quizzes – Each quiz generated by a particular course has a unique quiz id associated with it. Each quiz is mapped with the course id. It has the following fields:

Field	Description
quizId	A unique UUID generated for each newly created quiz.
courseId	The course id the quiz is linked to.
createdAt	The time at which the quiz was created.
quizData	A custom structure that we have stringified as JSON and stored that has all the questions, answers, and mark weightage.
quizName	Name of the quiz
quizStatus	Whether the quiz is active or not
timeLimit	Time limit of the quiz
totalMarks	Total marks the quiz carries

Completed Read capacity units consumed: 1

Items returned (8)

Actions Create item

< 1 > ⚙️ 🔍

<input type="checkbox"/>	quizId	courseId	createdAt	quizData	quizName	quizStatus	timeLimit	totalMarks
<input type="checkbox"/>	79ea8b9b-38d3-494c...	1ffd640f-6...	2022-07-2...	[{"M": {"o...	quiz9	ACTIVE	5	15
<input type="checkbox"/>	0e93d692-b141-4a7a...	1ffd640f-6...	2022-07-2...	[{"M": {"o...	quiz3	ACTIVE	5	10
<input type="checkbox"/>	2c8a913a-f753-4ce3-...	1ffd640f-6...	2022-07-2...	[{"M": {"o...	quiz5	ACTIVE	5	14
<input type="checkbox"/>	50a39e3d-6794-492a...	1ffd640f-6...	2022-07-2...	[{"M": {"o...	quiz1	ACTIVE	5	1
<input type="checkbox"/>	6443cf79-4f1b-45a3-...	983339b7-...	2022-07-2...	[{"M": {"o...	quiz201-2	COMPLETED	5	1
<input type="checkbox"/>	61aac5d5-ecef-449c-...	983339b7-...	2022-07-2...	[{"M": {"o...	quiz201-1	ACTIVE	5	10
<input type="checkbox"/>	fc6437f2-ab36-4cc1-...	1ffd640f-6...	2022-07-2...	[{"M": {"o...	quiz7	ACTIVE	15	10
<input type="checkbox"/>	b7778797-5041-439...	1ffd640f-6...	2022-07-2...	[{"M": {"o...	quiz6	ACTIVE	10	10

Figure 5. Fields in the quizzes table

- Courses – Each course is offered by an instructor and has a list of enrolled students in it. Here are the fields in the courses table:

Field	Description
courseId	The course id that is unique for each course.
courseNo	A course number that the instructor puts in while creating the course.
createdAt	The time at which the course was created.
enrolledStudents	An array of student Id (from the students table) that tells us the enrolled students for this course.
instructorId	The id of the instructor who has created this course

✓ Completed Read capacity units consumed: 0.5						
Items returned (12)						↺ ↻ ⚙️ 🔍
	courseId	courseNa...	createdAt	description	enrolledStudents	instructorId
<input type="checkbox"/>	25dda37a-843c-4748...	course555	2022-07-2...	asdasdasd	[]	ff8fde30-677c-4a3a-9537-45be61846445
<input type="checkbox"/>	1ffd640f-64d5-4312-...	course166	2022-07-1...	xcxvcxvcvwe...	[{"S": "b0cb3e35-f...	ff8fde30-677c-4a3a-9537-45be61846445
<input type="checkbox"/>	c499f921-c160-4031...	Cloud comp...	2022-07-1...	Learn cloud a...	[{"S": "b0cb3e35-f...	e7ebabb3-2ceb-4571-8b2d-435560aeb6f7
<input type="checkbox"/>	5b9ce979-94df-4894...	course133	2022-07-1...	asdasdsadsad	[{"S": "b0cb3e35-f...	ff8fde30-677c-4a3a-9537-45be61846445
<input type="checkbox"/>	7dc0afd8-4014-4a10...	course155	2022-07-1...	wewqewqewqe	[{"S": "b0cb3e35-f...	ff8fde30-677c-4a3a-9537-45be61846445

Figure 6. Fields in the courses table.

- AttemptedQuizzes – For each quiz that the student attempts, we need to store the result of that quiz. For each attempted quiz, an entry to this table is made with a unique id corresponding to the attempt of that particular student. Each students has a property that stores the array of his / her attempted quiz's id. The fields in this table are:

Field	Description
attemptedQuizid	The id that is unique for each attempted quiz.
attemptedQuestionList	A list of the questions
courseId	The id of the course the quiz is for
obtainedMarks	Marks obtained by the student in that quiz attempt.
quizId	The id of the quiz this attempt is linked to
quizName	The name of the quiz
totalMarks	Total marks of the quiz

Completed Read capacity units consumed: 1

Items returned (11)

	attemptedQuizId	attemptedQuestionList	courseId	obtainedMarks	quizId	quizName	totalMarks
<input type="checkbox"/>	c3d43fa2-daae-4de8-...	[{"M": {"questionId": {"S...}}	1ffd640f-6...	10	fc6437f2-a...	quiz7	10
<input type="checkbox"/>	13703a6d-1fa9-4b94-...	[{"M": {"questionId": {"S...}}	5b9ce979-...	0	6e8dbc60-c...	quiz1	10
<input type="checkbox"/>	c044d7bf-092e-4175-...	[{"M": {"questionId": {"S...}}	1ffd640f-6...	2	b7778797-...	quiz6	10
<input type="checkbox"/>	d246e16c-982a-43a4-...	[{"M": {"questionId": {"S...}}	5b9ce979-...	4	6e8dbc60-c...	quiz1	10
<input type="checkbox"/>	3a128b0e-11f7-48dc-...	[{"M": {"questionId": {"S...}}	1ffd640f-6...	10	0e93d692-...	quiz3	10
<input type="checkbox"/>	f4a73915-7f0a-4dc9-...	[{"M": {"questionId": {"S...}}	1ffd640f-6...	0	50a39e3d-...	quiz1	1
<input type="checkbox"/>	da36684d-7751-46a0-...	[{"M": {"questionId": {"S...}}	1ffd640f-6...	7	0e93d692-...	quiz3	10
<input type="checkbox"/>	b4d1d2f0-e48f-4884-...	[{"M": {"questionId": {"S...}}	983339b7-...	8	61aac5d5-e...	quiz201-1	10
<input type="checkbox"/>	dbdfd930-3818-4687-...	[{"M": {"questionId": {"S...}}	1ffd640f-6...	0	79ea8b9b-...	quiz9	15

Figure 7. Fields in the quizzes attempted table

Programming languages used:

For the frontend, we have used Angular framework. This enabled us to quickly build the app using its component-based architecture. As our backend is serverless, we have made a NodeJS-Express app that is then wrapped around with serverless framework.

Languages used – TypeScript for frontend, JavaScript for backend.

How is app deployed:

The backend is deployed as API gateway with the corresponding lambdas. This infrastructure is deployed using the “serverless deploy” command.

The frontend is deployed using a containerized docker image on AWS Elastic Container Service. The docker image is stored on AWS Elastic Container Registry.

To automate the docker build process for continuous development on changes to frontend code, we have utilized AWS CodePipeline. Our repository on gitlab is cloned on AWS CodeCommit. CodePipeline has 3 stages: Source, Build and Deploy. It will monitor the CodeCommit repository for changes and will trigger on “git push” command when pushing to the specific CodeCommit remote URL.

The build stage (using AWS CodeBuild) will create a new docker image from the newly committed code and update the existing image on AWS Elastic Container Registry (ECR). The deploy stage (using AWS CodeDeploy) has been configured to create a new task definition using the newly pushed ECR Image. This task definition will be created in the configured ECS Cluster and Service and will be assigned a public IP address.

Since, every time a new change is committed, a new public IP is assigned to the ECS Task, we have set up a load balancer with a static IP and DNS to make sure our entry point for the application remains constant.

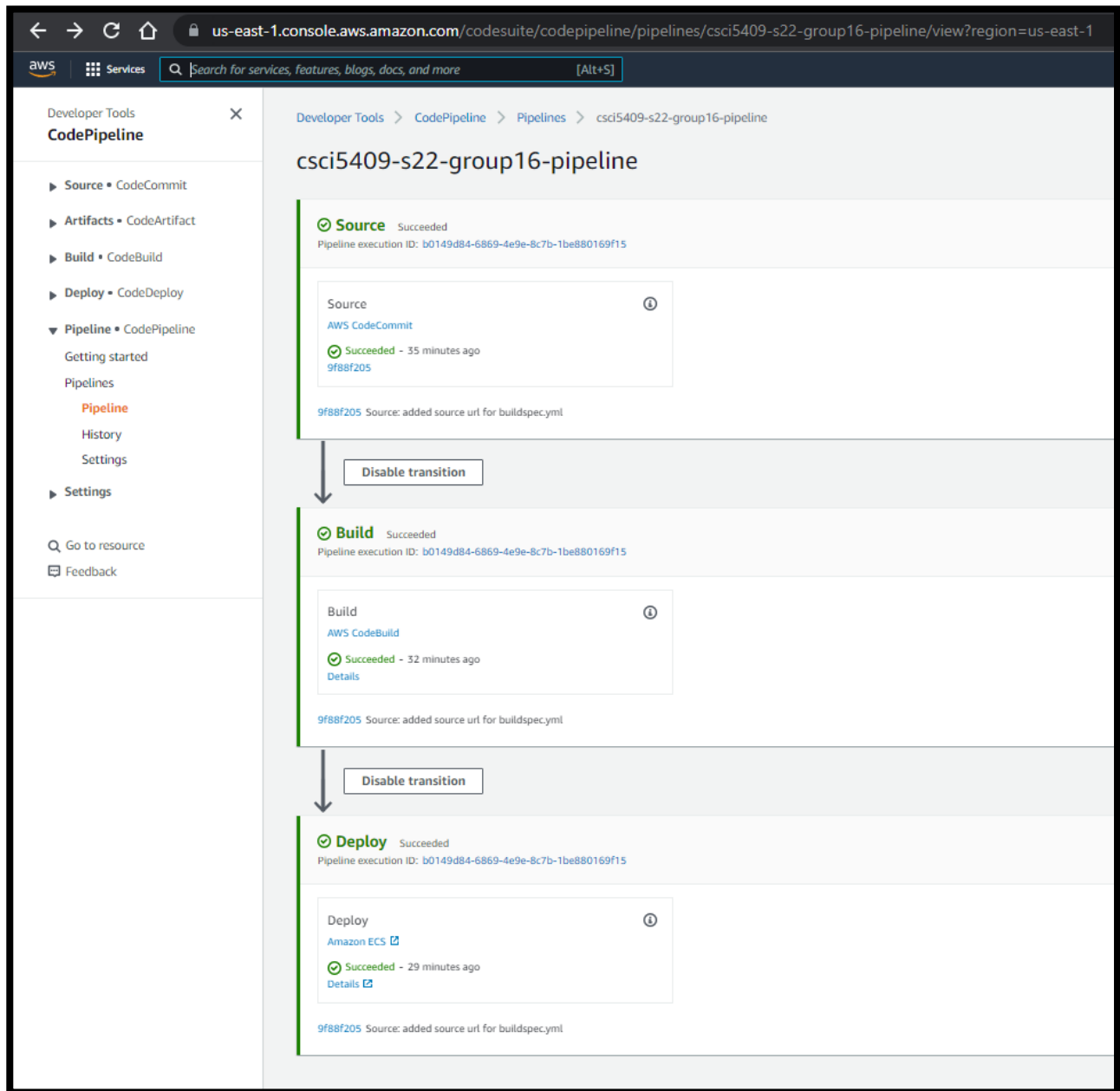


Figure 8.8 CodePipeline for automating frontend container build, push and deploy process

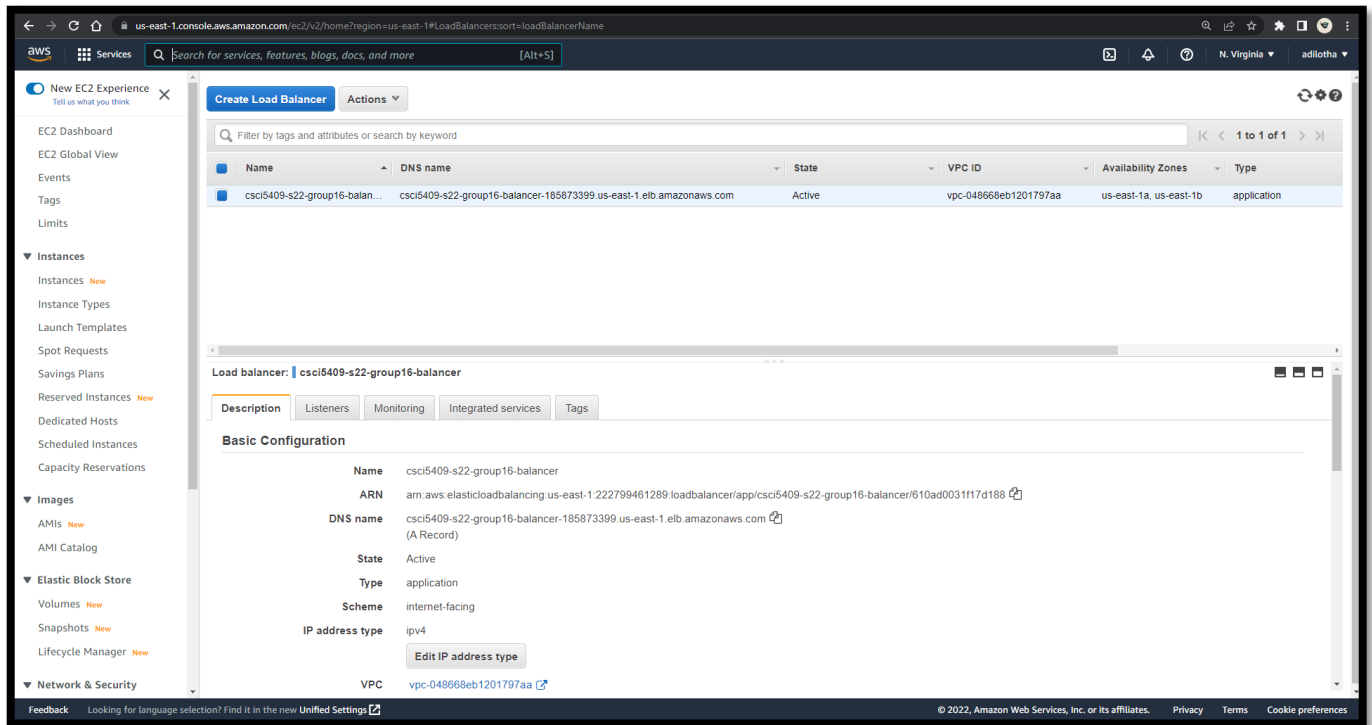


Figure 9.9 Load Balancer

2. If your final architecture differs from your original project proposal, explain why that is. What did you learn that made you change technologies or approaches?

From project proposal the only difference in the final architecture in terms of services is the use of the Secrets Manager [13]. Initially, the secrets manager was to save database credentials like passwords of the users, but as we proceeded with the course and assignments, we figured a better and efficient way. Amazon Web Services (AWS) has keys like secret, access, and token, which helps in programmatically accessing the services like DynamoDB.

The Simple Notification Service (SNS) [9] application has been slightly changed from the project proposal. Users (students) would initially type their parent's email at the completion of the quiz, and SNS would—as we anticipated—send the emails to the parents. But, in our final prototype, the email address that students register with will be the email address that is automatically subscribed to receive grades. This has been implemented by applying a filter policy such that only the supposed student gets the email.

**3. How would your application evolve if you were to continue development?
What features might you add next and which cloud mechanisms would you
use to implement those features?**

A few of the great additions that we can do to the existing application can be as follows:

- *Function:* Document submission.
Just like other online education applications provision to submit the documents from the user, we can add functionality of submitting PDF or Ms Doc documents submission to the portal.
Service: This will mainly use S3 Buckets service for uploading and accessing the documents on both teacher and student ends.
- *Function:* Discussion threads and announcement postings.
Students can subscribe to these discussion threads and whenever anything new is posted, they will get notification. Th students will be however, automatically subscribed to announcement postings.
Service: This service will employ SNS service to send push notifications to the subscriber list.
- *Function:* Predicting student final grade.
Teachers can use a score prediction functionality to know or track the performance of their students beforehand.
Service: This service will employ Amazon ML (Machine Learning) [14] service to do batch predictions of the scores stored in the S3 buckets, generate a report, and store it in S3 buckets for the users to access.

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

For the AWS Services utilized to create our application, we have tried to follow appropriate practices to keep our data secure. However, there is still room for improvement to make our application more secure.

We could utilize AWS Cognito's Multi-factor Authentication for better security during login. Besides this, we could deploy the Backend APIs on a Virtual Private Cloud to provide an additional layer of security to the clients. API Gateway and ECS can be deployed in a Public Subnet of a VPC (Virtual Private Cloud) and the Lambdas, Cognito, DynamoDB, SNS, S3 Services can be kept inside the same VPC but in a private subnet. This way only the services in the public subnet can call services in the private subnet to provide an extra layer of security.

5. 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.)

The backend APIs, even though publicly available, are protected using **JWT Authentication** supported by **AWS Cognito**. Only registered users will be able to access the application data. Even though the Access Token can be accessed from the browser's local storage, it will expire after a limited amount of time. In addition to this, users are required to verify their email on registration.

Besides this, we have used S3 Bucket for storing images. Whenever a student access their profile, a **signed S3 URL** for the image is generated from the backend which will be used by the browser to render it. However, this signed URL will be set to expire after a limited amount of time to prevent malicious access to our S3 Bucket.

- 6. 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.

What our organization may need to purchase if they want to replicate the architecture on private cloud?

The reason our organization may want to have the infrastructure set on-premise as a private cloud can be the fine-granular control and the security of data not leaving the organizational boundaries.

Our architecture uses lambdas for all the backend functionalities. Lambdas scale gracefully on peak load times. For achieving such levels of elasticity and availability, the organization will need to buy high performance servers and then put-up virtualization on top of it to make effective use of the server resources. There will be a need of a virtualization software too that can manage virtualization.

DynamoDB scales automatically on load. It scales twice the size of the read-write capacity to If the organization wants such a rapidly scaling database, then they need to scale horizontally wherein database clusters need to be added when read/write operations increase. To have such a rapid provisioning of database nodes, there needs to be a database cluster manager that can actively listen to the current load on the existing cluster and then scale accordingly.

Also, for the frontend, we have used ECS to serve the Angular frontend app. ECS is a fully-managed service that orchestrates containers as per the load. If the organization wants a similar behaviour then surely they need to use containerization workflow for their apps using Docker. But along with that they also need to manage the orchestrate this configuration using container orchestration wizards like Docker swarm or Kubernetes.

To help the organization a bit in setting this up, we can use the Openstack framework to deploy and manage all the infrastructure. Openstack provides APIs and dashboard for easy management of compute, storage, database, and network resources.

Along with all these, there must be self-service available so that resources can be continuously monitored for usage and scaled accordingly. If we choose up disaster

management for better resiliency, then it would be too much to handle for a new organization give the cost and resources.

An estimate of all of this would be:

Component	Usage rationale	Cost estimate	Recurrence
Servers to replicate lambdas	To offer compute power to the infrastructure	\$5,000 (basic server) in case more throughput is required with more storage and RAM then it may go around \$10,000	Only once
Database clusters	To persist data	\$ 2,000 MongoDB NoSQL database	Only once
Container orchestration	To manage frontend web app hosting	Docker and kubernetes is open-source so \$0	-
Openstack	To manage the infrastructure	It's free as well \$0	-
Service costs	To provide servicing for broken components	Technical experts are required for this.	\$20,000 monthly as salaries
Disaster recovery	To ensure high resiliency	If this is opted-in then the cost may sore anywhere from buying a whole new place for replication to buying the exact replica infrastructure	-

Hardware and software components required to replicate the infrastructure on-premise:

Sr.no	Component	Usage	Cost
1	Server CPU- Intel Xeon	For computational power	\$500 * 4 = \$2000
2	RAM	For memory of the server 512 GB RAM stick racks	\$3000
3	SSD	For storing data10 TB	\$2000
4	Motherboard		\$1000
5	Power supply units	For spinning the server up	\$1000
6	Operating Software - Ubuntu	Free	\$0
7	Docker and Kubernetes	Free	\$0
8	Servicing	For repairs and servicing of components	\$20,000 per month
9	Operational costs	To keep the systems up and running 24x7	\$5,000 per month

- 7. Which cloud mechanism would be most important for you 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?**

We have researched on AWS lambdas and cloud functions that they scale nicely but are also very risky to work with. A piece of bad code can make these functions run infinitely costing thousands in just seconds. We think that we must add CloudWatch alarms for AWS lambdas that we have for our app. In case there's a code that is spinning up huge lambda invocations then we can easily monitor this and notify to us when these service goes out of hands. With these notifications in place, we can set up alarms that can even terminate such lambdas in case they go out of hands.

There's a case study of a start-up that cost them a bill of \$75,000 in just 2 hours of going live [15]. The source of this disaster was a recursive function without a base case. Scenarios like these tell us how important it is to code with care in the cloud.

8. REFERENCES:

- [1] "Serverless Cloud – Documentation", *serverless.com*, 2022. [Online]. Available: <https://www.serverless.com/cloud/docs>. [Accessed: 23- Jul- 2022]
- [2] "What is Amazon Api Gateway? - AWS documentation." [Online]. Available: <https://docs.aws.amazon.com/apigateway/latest/developerguide/welcome.html>. [Accessed: 26-Jul-2022.].
- [3] "AWS Lambda," Amazon. [Online]. Available: <https://aws.amazon.com/lambda/pricing/>. [Accessed: 26-Jul-2022].
- [4] "Amazon Cognito - Simple and Secure User Sign Up & Sign In | Amazon Web Services (AWS)," Amazon Web Services, Inc., 2022. [Online]. Available: <https://aws.amazon.com/cognito/>. [Accessed: 26-Jul-2022].
- [5] "amazon-cognito-identity-js", *npm*, 2022. [Online]. Available: <https://www.npmjs.com/package/amazon-cognito-identity-js>. [Accessed: 25- Jul- 2022]
- [6] "Amazon DynamoDB Documentation", *aws.amazon.com*, 2022. [Online]. Available: <https://docs.aws.amazon.com/dynamodb>. [Accessed: 24- Jul- 2022]
- [7] B. Schneider, "ECR," *Amazon*, 1976. [Online]. Available: <https://docs.aws.amazon.com/ecr/>. [Accessed: 26-Jul-2022]
- [8] "What is Amazon Elastic Container Service?" [Online]. Available: <https://docs.aws.amazon.com/AmazonECS/latest/developerguide/Welcome.html>. [Accessed: 26-Jul-2022].
- [9] "Amazon Simple Notification Service (SNS) | Messaging Service | AWS," Amazon Web Services, Inc., 2021. [Online]. Available: <https://aws.amazon.com/sns/>. [Accessed: 26-Jul-2022].
- [10] "Conversational AI and Chatbots - Amazon Lex - Amazon Web Services", *Amazon Web Services, Inc.*, 2022. [Online]. Available: <https://aws.amazon.com/lex>. [Accessed: 26- Jul- 2022]
- [11] "Deploy a Web UI for Your Chatbot | Amazon Web Services", *Amazon Web Services*, 2022. [Online]. Available: <https://aws.amazon.com/blogs/machine-learning/deploy-a-web-ui-for-your-chatbot/>. [Accessed: 26- Jul- 2022]

[12] "Credential password secrets management - AWS Secrets Manager - Amazon Web Services", *Amazon Web Services, Inc.*, 2022. [Online]. Available: <https://aws.amazon.com/secrets-manager/>. [Accessed: 26- Jul- 2022]

[13] "Cloud Object Storage – Amazon S3 – Amazon Web Services," Amazon Web Services, Inc., 2022. [Online]. Available: <https://aws.amazon.com/s3/>. [Accessed: 26-Jul-2022].

[14] "Use the ML Model to Generate Predictions PDF", AWS, 2022. [Online]. Available: <https://docs.aws.amazon.com/machine-learning/>. [Accessed: 26- Jul- 2022]

[15] "How a Start-Up Received a \$75,000 Bill for 2 Hours of Google Cloud Services", *Electropages.com*, 2022. [Online]. Available: <https://www.electropages.com/blog/2021/01/how-start-received-75000-bill-2-hours-google-cloud-services>. [Accessed: 24- Jul- 2022]