



Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY
(Approved by AICTE New Delhi & Govt. of Maharashtra, Affiliated to University of Mumbai)
(Religious Jain Minority)

Serverless Web Application on AWS

Submitted in Complete Fulfilment of the

Requirement for the APSIT Skills

Internship under the

Course: AWS Academy

Cloud Foundation

Information Technology Engineering

by

Mr. Prathamesh Hambar (19102001)

Ms. Ragini Pandey (18104065)

Ms. Ruta Mhaskar (19104013)

Mr. Tanay Jain (20204004)

Under the Guidance of

Prof. Nahid Kausar Shaikh

Prof. Vishal S. Badgujar

Department of Information Technology Engineering

Academic Year: 2021-2022

APSIT
Skills
We Set Trends



Parshvanath Charitable Trust's
A. P. SHAH INSTITUTE OF TECHNOLOGY
(Approved by AICTE New Delhi & Govt. of Maharashtra, Affiliated to University of Mumbai)
(Religious Jain Minority)

CERTIFICATE

This is to certify that,

Mr. Prathamesh Hambar	(19102001/Computer Engineering)
Ms. Ragini Pandey	(18104065/ Information Technology)
Ms. Ruta Mhaskar	(19104013/ Information Technology)
Mr. Tanay Jain	(20204004/ Information Technology)

have satisfactorily carried out the project work entitled **Serverless Web Application on AWS** in complete fulfillment of the APSIT Skills internship in **Information Technology Engineering** as laid down by the University of Mumbai during the academic year 2021-2022.

Prof. Nahid Kausar Shaikh
Faculty guide

Prof. Kiran Deshpande
Head of Department

CONTENTS

INTRODUCTION	1
SOFTWARE REQUIREMENTS	2
PROPOSED ARCHITECTURE	4
METHODOLOGY	5
Database Creation:	5
Lambda Functions	6
IAM Roles and Policies	6
API Gateway	8
AWS S3	9
AWS CloudFront	10
OUTPUT	11
ADVANTAGES AND DISADVANTAGES	12
Advantages	12
Disadvantages	12
APPLICATIONS	13

INTRODUCTION

Amazon Web Services (AWS) is the world's most comprehensive and broadly adopted cloud platform. AWS has significantly more services than any other cloud provider. It is the most flexible and secure cloud computing application. It helps us leverage the latest technologies to experiment and innovate more quickly.

Serverless computing is a cloud computing execution model in which the cloud provider allocates machine resources on-demand, taking care of the servers on behalf of their customers. Serverless computing does not hold resources in volatile memory; computing is rather done in short bursts with the results persisted to storage. When an app is not in use, there are no computing resources allocated to the app. Pricing is based on the actual amount of resources consumed by an application. It can be a form of utility computing. "Serverless" is a misnomer in the sense that servers are still used by cloud service providers to execute code for developers. However, developers of serverless applications are not concerned with capacity planning, configuration, management, maintenance, fault tolerance, or scaling of containers, VMs, or physical servers.

This project involves the development and deployment of a single-page web application that can be utilized to store what online courses are offered by an author. It is made with React and Redux and boilerplate code sourced from Github. It is designed to perform CRUD (Create, Read, Update & Delete) operations. For the lambda functions, a Node.js runtime environment is used with the Javascript SDK. Everything was made from the AWS Management Console, without any external frameworks, SDK, or command-line interfaces (CLI)

SOFTWARE REQUIREMENTS

Visual Studio Code: It is a lightweight but powerful source code editor which runs on the desktop and is available for Windows, macOS, and Linux. It comes with built-in support for JavaScript, TypeScript, and Node.js and has a rich ecosystem of extensions for other languages and runtimes. It can also be used as an IDE (Integrated Development Environment) for building executables and debugging.

Node.js: It is an asynchronous event-driven JavaScript runtime. It is designed to build scalable network applications.

React.js: It is a JavaScript library for building user interfaces.

AWS Management Console: It provides a web-based user interface that the clients can use to create and manage their AWS resources.

Amazon DynamoDB: It is a fully managed NoSQL database service that provides fast and predictable performance with seamless scalability. It is used to offload the administrative burdens of operating and scaling a distributed database so that the clients don't have to worry about hardware provisioning, setup, and configuration, replication, software patching, or cluster scaling. It also offers encryption at rest, which eliminates the operational burden and complexity involved in protecting sensitive data.

AWS IAM: Identity and Access Management is a web service that helps clients securely control access to AWS resources. IAM is used to control who is authenticated (signed in) and authorized (has permissions) to use resources.

AWS SDK for JavaScript: The AWS Javascript Software Development Kit (SDK) provides access to the web services in either browser scripts or Node.js.

AWS Lambda: It is a compute service that lets clients run code without provisioning or managing servers. It runs the code on a high-availability compute infrastructure and performs all of the administration of the compute resources, including server and operating system maintenance, capacity provisioning and automatic scaling, code monitoring, and logging. It can be used to run code for virtually any type of application or backend service. It supports Javascript among others.

Amazon API Gateway: It is an AWS service for creating, publishing, maintaining, monitoring, and securing REST, HTTP, and WebSocket APIs at any scale. API developers can create APIs that access AWS or other web services. As an API Gateway API developer, one can create APIs for use in their client applications. Or they can make their APIs available to third-party app developers.

Amazon S3: Simple Storage Service (S3) is storage for the Internet. It is designed to make web-scale computing easier. It gives AWS clients access to the same highly scalable, reliable, fast, and inexpensive data storage infrastructure that Amazon uses to run its global network of websites.

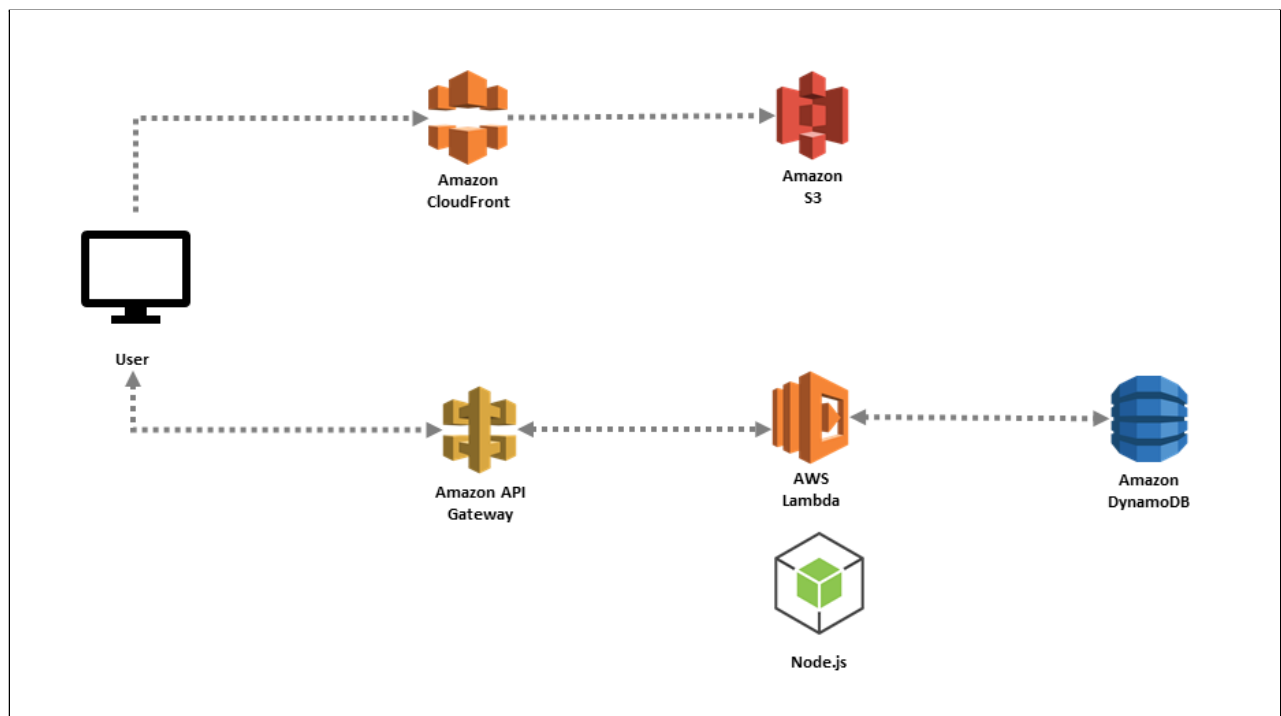
AWS CloudFront: It is a Content Delivery Network (CDN) service from AWS. It is a web service that speeds up the distribution of clients' static and dynamic web content, such as .html, .css, .js, and image files, to their users. CloudFront delivers the content through a worldwide network of data centers called edge locations. When a user requests content that the client is serving with CloudFront, the request is routed to the edge location that provides the lowest latency (time delay), so that content is delivered with the best possible performance.

Amazon CloudWatch: It monitors our AWS resources and the applications clients run on AWS in real-time. One can use CloudWatch to collect and track metrics, which are variables they can measure for our resources and applications. They can gain system-wide visibility into resource utilization, application performance, and operational health.

PROPOSED ARCHITECTURE

In this project, a simple web application was built using the following AWS services:

- AWS DynamoDB as a database.
- AWS Lambda to create functions that will read and write from/to the database.
- AWS API Gateway to create the RESTful API that the web application will use.
- AWS S3 to host the web application.
- AWS CloudFront to deliver the web application from a location near to the user's location.
- AWS CloudWatch for logging and tracking metrics,



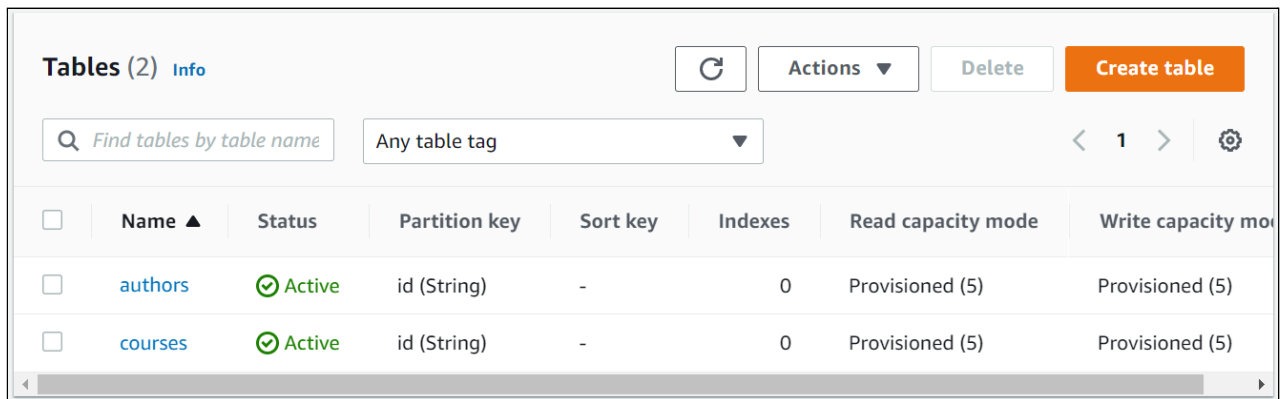
METHODOLOGY

We first create an AWS account with the payment details to activate the free tier. All of the resources used in this project are well under the free tier limits. All the resources except IAM and API Gateway, have their regions set as Mumbai (ap-south-1).

Following are the detailed steps executed via AWS Management Console for the project

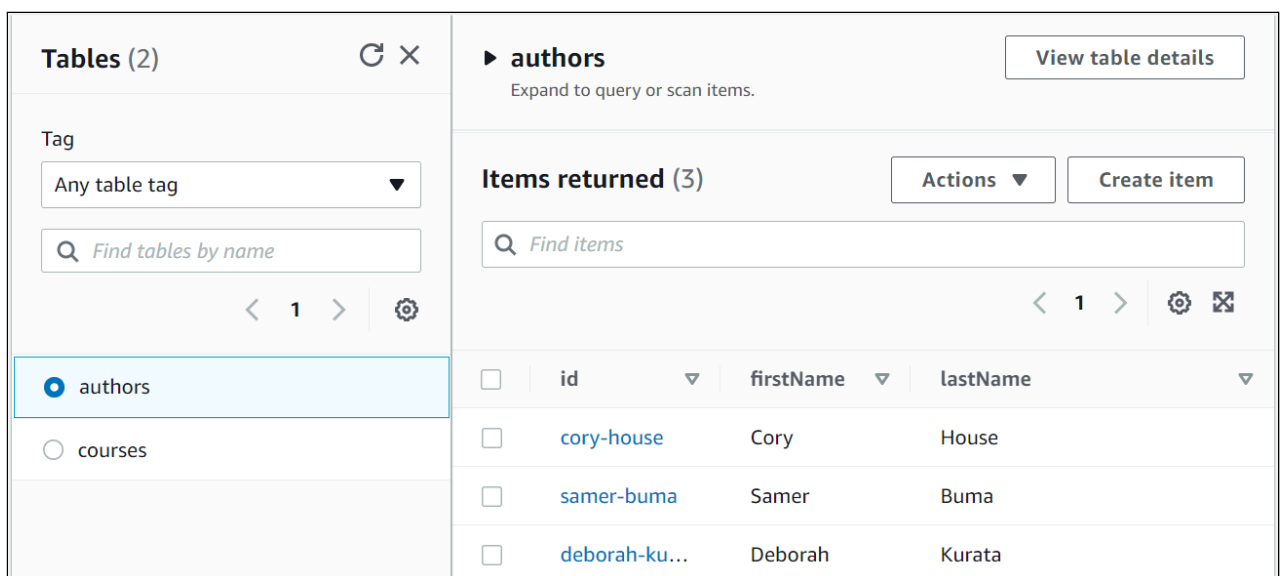
1) Database Creation:

We started by first creating two databases, authors and courses. Both have key-value pair structures in JSON format i.e. a NoSQL database. Some items are added to the “authors” table. Then, the Amazon Resource Number (ARN) is noted for further reference.



<input type="checkbox"/>	Name ▲	Status	Partition key	Sort key	Indexes	Read capacity mode	Write capacity mode
<input type="checkbox"/>	authors	Active	id (String)	-	0	Provisioned (5)	Provisioned (5)
<input type="checkbox"/>	courses	Active	id (String)	-	0	Provisioned (5)	Provisioned (5)

Fig. 1a - DynamoDB Tables



<input type="checkbox"/>	id ▼	firstName ▼	lastName ▼
<input type="checkbox"/>	cory-house	Cory	House
<input type="checkbox"/>	samer-buma	Samer	Buma
<input type="checkbox"/>	deborah-ku...	Deborah	Kurata

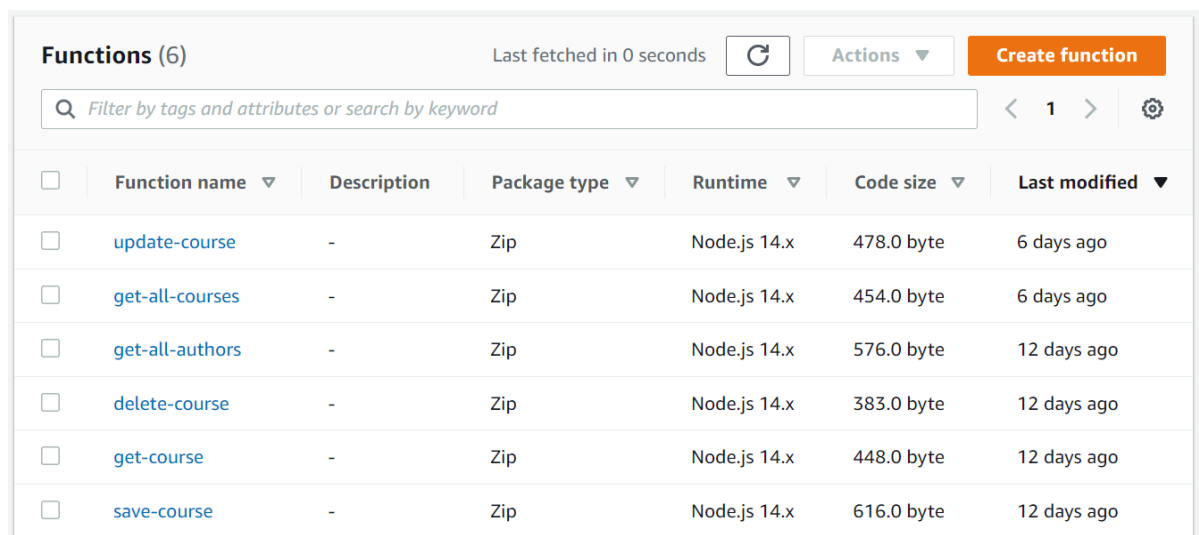
Fig. 1b - DynamoDB Table Items

2) Lambda Functions

A lambda function will be run whenever a request hits one of the API endpoints. There are a total of six lambda functions used in this project written in Javascript with AWS Javascript SDK and a Node.js runtime environment.

The 6 functions are:-

- **get-all-authors:** to return all the users in the database
- **get-all-courses:** to return all the courses in the database
- **get-course:** to return only one course
- **save-course:** to create a new course
- **update-course:** to update a course
- **delete-course:** to delete a course



The screenshot shows the AWS Lambda console interface. At the top, it says 'Functions (6)' and 'Last fetched in 0 seconds'. There is a search bar with the placeholder text 'Filter by tags and attributes or search by keyword'. Below the search bar is a table with 7 columns: 'Function name', 'Description', 'Package type', 'Runtime', 'Code size', and 'Last modified'. The table lists 6 functions: 'update-course', 'get-all-courses', 'get-all-authors', 'delete-course', 'get-course', and 'save-course'. Each function has a checkbox to its left and a dropdown arrow to its right. The 'Description' column contains dashes for all functions. The 'Package type' is 'Zip' for all. The 'Runtime' is 'Node.js 14.x' for all. The 'Code size' varies: 478.0 byte, 454.0 byte, 576.0 byte, 383.0 byte, 448.0 byte, and 616.0 byte respectively. The 'Last modified' times are '6 days ago' for the first two and '12 days ago' for the others.

	Function name	Description	Package type	Runtime	Code size	Last modified
<input type="checkbox"/>	update-course	-	Zip	Node.js 14.x	478.0 byte	6 days ago
<input type="checkbox"/>	get-all-courses	-	Zip	Node.js 14.x	454.0 byte	6 days ago
<input type="checkbox"/>	get-all-authors	-	Zip	Node.js 14.x	576.0 byte	12 days ago
<input type="checkbox"/>	delete-course	-	Zip	Node.js 14.x	383.0 byte	12 days ago
<input type="checkbox"/>	get-course	-	Zip	Node.js 14.x	448.0 byte	12 days ago
<input type="checkbox"/>	save-course	-	Zip	Node.js 14.x	616.0 byte	12 days ago

Fig. 2 - Lambda Functions

The code is stored in index.js. We also create a test event for each of the functions to check if they are returning the expected response. At present, there are no triggers or any destinations added. We won't be utilizing any layers as well.

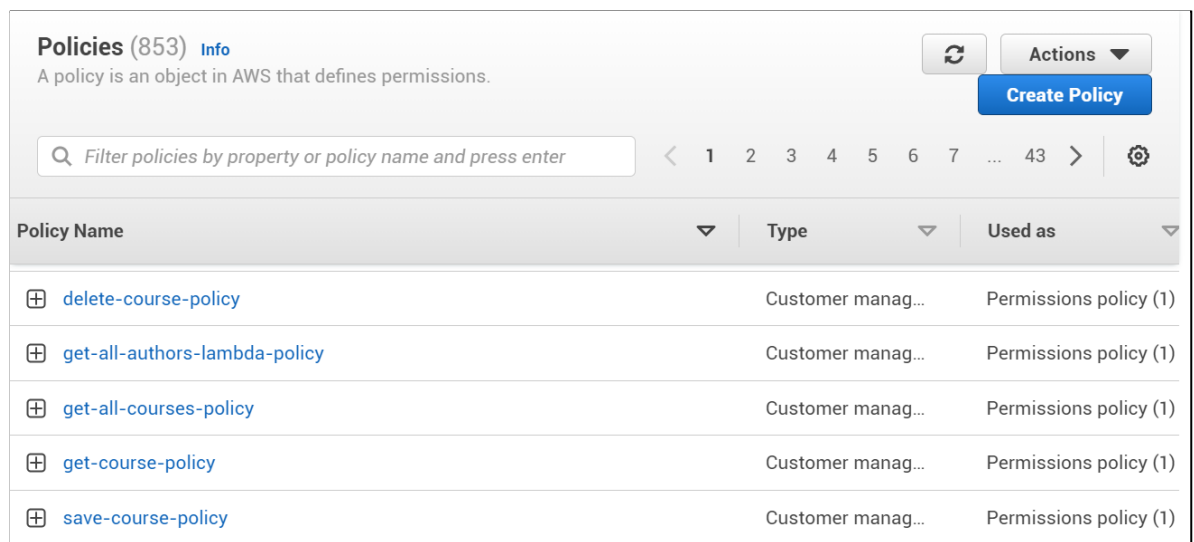
3) IAM Roles and Policies

Before creating the lambda functions that will be used to manipulate the DB, we first created a set of custom policies and their corresponding roles for each lambda function. In general, we added CloudWatch permissions as well to log events i.e. CreateLogGroup, CreateLogStream, PutLogEvents.

While creating IAM policies, we followed the security best practice of granting the least privilege or granting only the permissions required to perform a task. Those are:-

λ -Function	Actions	Resource
get-all-authors	dynamodb:Scan	table/authors
get-all-courses	dynamodb:Scan	table/courses
save-course	dynamodb:PutItem	table/courses
get-course	dynamodb:GetItem	table/courses
update-course	dynamodb:PutItem	table/courses
delete-course	dynamodb>DeleteItem	table/courses

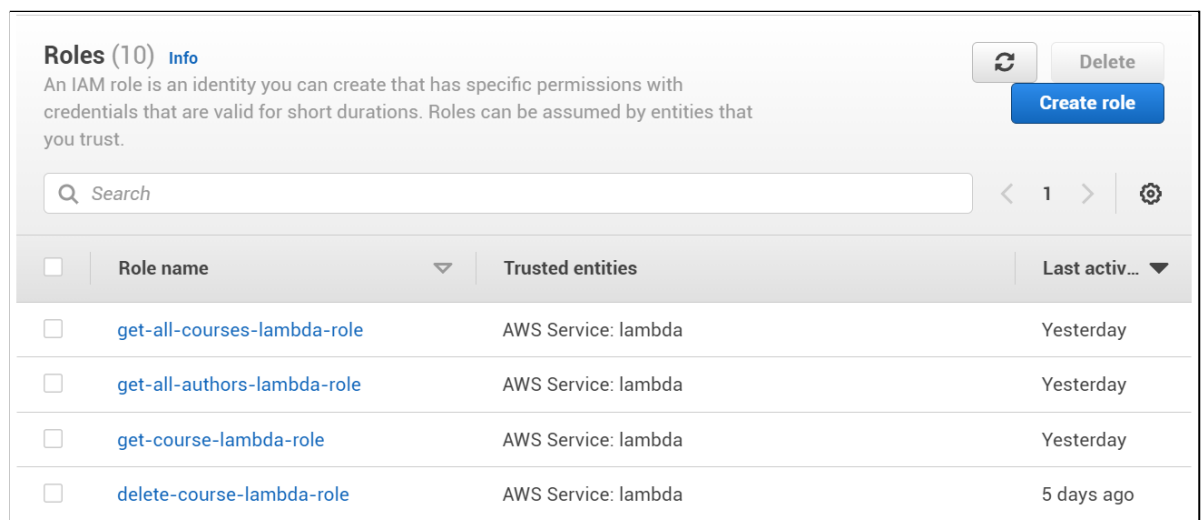
Table 1 - Allowed actions and resources for each of the λ -functions



The screenshot shows the AWS IAM console 'Policies' page. At the top, it says 'Policies (853)' with an 'Info' link. Below this is a description: 'A policy is an object in AWS that defines permissions.' To the right are buttons for 'Refresh', 'Actions', and a prominent blue 'Create Policy' button. A search bar is present with the placeholder text 'Filter policies by property or policy name and press enter'. Below the search bar is a table with columns: 'Policy Name', 'Type', and 'Used as'. The table lists five policies, all of type 'Customer managed' and used as 'Permissions policy (1)'. Each policy name is preceded by a plus icon in a square.

Policy Name	Type	Used as
+ delete-course-policy	Customer managed...	Permissions policy (1)
+ get-all-authors-lambda-policy	Customer managed...	Permissions policy (1)
+ get-all-courses-policy	Customer managed...	Permissions policy (1)
+ get-course-policy	Customer managed...	Permissions policy (1)
+ save-course-policy	Customer managed...	Permissions policy (1)

Fig. 3a - Customer managed policies for λ -functions



The screenshot shows the AWS IAM console 'Roles' page. At the top, it says 'Roles (10)' with an 'Info' link. Below this is a description: 'An IAM role is an identity you can create that has specific permissions with credentials that are valid for short durations. Roles can be assumed by entities that you trust.' To the right are buttons for 'Refresh', 'Delete', and a prominent blue 'Create role' button. A search bar is present with the placeholder text 'Search'. Below the search bar is a table with columns: 'Role name', 'Trusted entities', and 'Last activ...'. The table lists four roles, all with 'AWS Service: lambda' as the trusted entity. Each role name is preceded by an unchecked checkbox.

Role name	Trusted entities	Last activ...
<input type="checkbox"/> get-all-courses-lambda-role	AWS Service: lambda	Yesterday
<input type="checkbox"/> get-all-authors-lambda-role	AWS Service: lambda	Yesterday
<input type="checkbox"/> get-course-lambda-role	AWS Service: lambda	Yesterday
<input type="checkbox"/> delete-course-lambda-role	AWS Service: lambda	5 days ago

Fig. 3b - some of the λ -function roles

4) API Gateway

We used API Gateway to make a RESTful API that acts as a trigger for lambda functions. For this, we create two main endpoints i.e. /courses and /authors. We also make a child resource /{id} which is a path parameter to interact with individual courses.

Endpoint	Method	λ -functions
/authors	GET	get-all-authors
/courses	GET	get-all-courses
/courses	POST	save-course
/ {id}	GET	get-course
/ {id}	PUT	update-course
/ {id}	DELETE	delete-course

Table 2 - API endpoints and methods with their corresponding λ -functions.

- We enable API Gateway CORS (Cross-Origin Resource Sharing) so that we can access our content hosted on another domain.
- We also use models to validate the HTTP request sent from the user's end. A model defines the data structure of a payload. API Gateway models are defined using the JSON schema draft 4.
- In the API requests that involve {id}, the course ID comes from the URL and the rest of the request comes from the request body. Thus, we make use of a body mapping template so that the lambda function receives all of the required info. It is written in the Velocity Template Language which is maintained under the Apache Velocity Project.
- Then we add CORS headers to all of the methods.
- We then deployed our API.

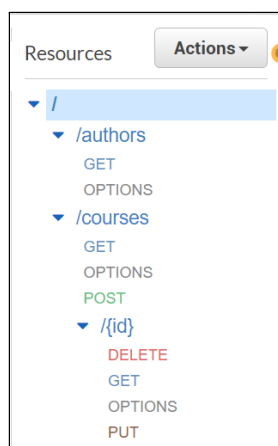
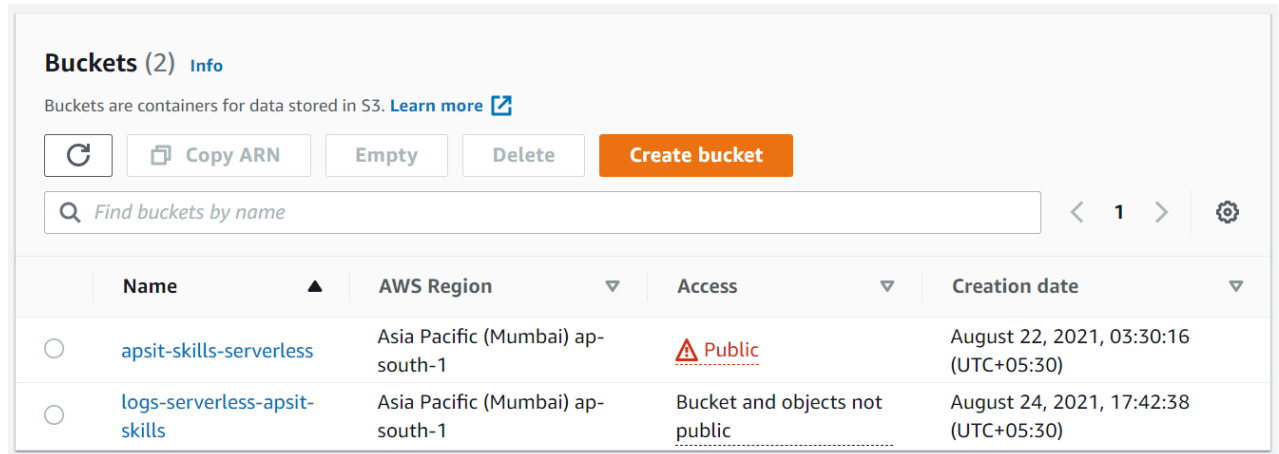


Fig. 4 - API Gateway Endpoints and methods

5) AWS S3

We then upload the build version of our React web app with the URL of the API we created in API Gateway. We first create an S3 bucket. We utilized the Static website hosting for our case with index.html set as the index and error document. We then write a policy with AllowPublicReadAccess and s3:GetObject. We also created another bucket for Server Access Logging.



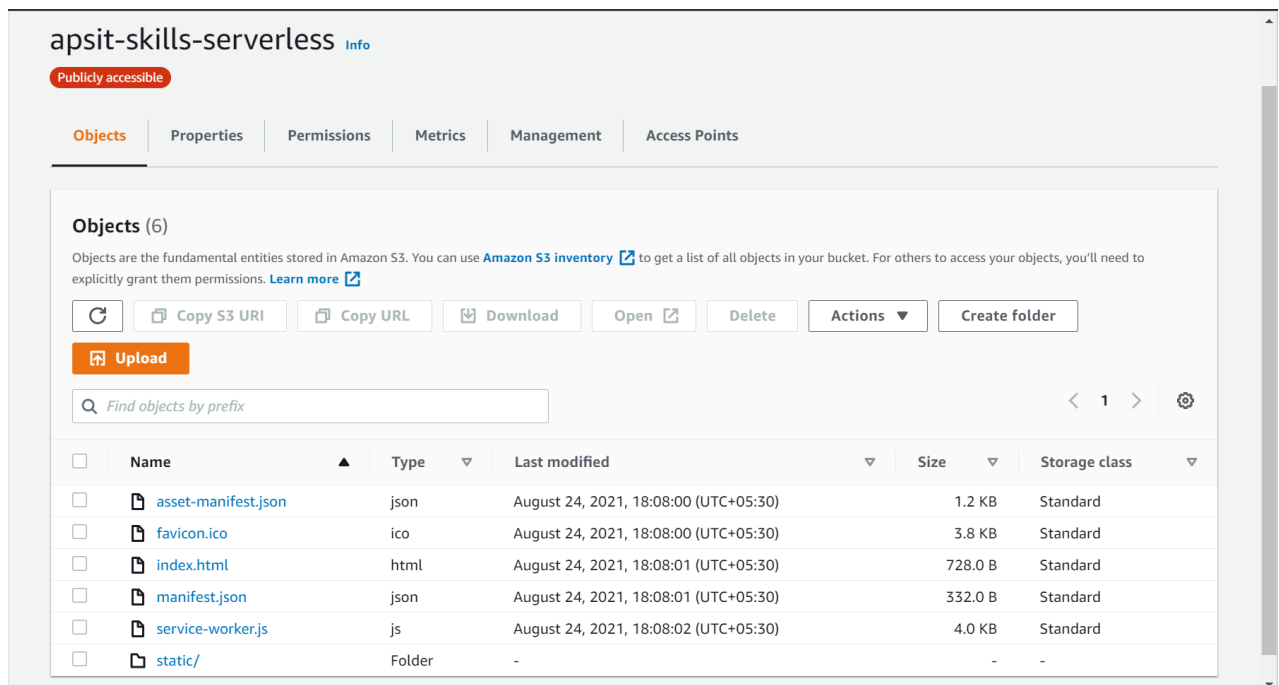
Buckets (2) [Info](#)

Buckets are containers for data stored in S3. [Learn more](#)

[Refresh](#) [Copy ARN](#) [Empty](#) [Delete](#) [Create bucket](#)

	Name ▲	AWS Region ▼	Access ▼	Creation date ▼
<input type="radio"/>	apsit-skills-serverless	Asia Pacific (Mumbai) ap-south-1	Public	August 22, 2021, 03:30:16 (UTC+05:30)
<input type="radio"/>	logs-serverless-apsit-skills	Asia Pacific (Mumbai) ap-south-1	Bucket and objects not public	August 24, 2021, 17:42:38 (UTC+05:30)

Fig. 5a - S3 buckets for static content and logs



apsit-skills-serverless [Info](#)

Publicly accessible

[Objects](#) [Properties](#) [Permissions](#) [Metrics](#) [Management](#) [Access Points](#)

Objects (6)

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. [Learn more](#)

[Refresh](#) [Copy S3 URI](#) [Copy URL](#) [Download](#) [Open](#) [Delete](#) [Actions](#) [Create folder](#)

[Upload](#)

<input type="checkbox"/>	Name ▲	Type ▼	Last modified ▼	Size ▼	Storage class ▼
<input type="checkbox"/>	asset-manifest.json	json	August 24, 2021, 18:08:00 (UTC+05:30)	1.2 KB	Standard
<input type="checkbox"/>	favicon.ico	ico	August 24, 2021, 18:08:00 (UTC+05:30)	3.8 KB	Standard
<input type="checkbox"/>	index.html	html	August 24, 2021, 18:08:01 (UTC+05:30)	728.0 B	Standard
<input type="checkbox"/>	manifest.json	json	August 24, 2021, 18:08:01 (UTC+05:30)	332.0 B	Standard
<input type="checkbox"/>	service-worker.js	js	August 24, 2021, 18:08:02 (UTC+05:30)	4.0 KB	Standard
<input type="checkbox"/>	static/	Folder	-	-	-

Fig. 5b - S3 bucket objects of static content

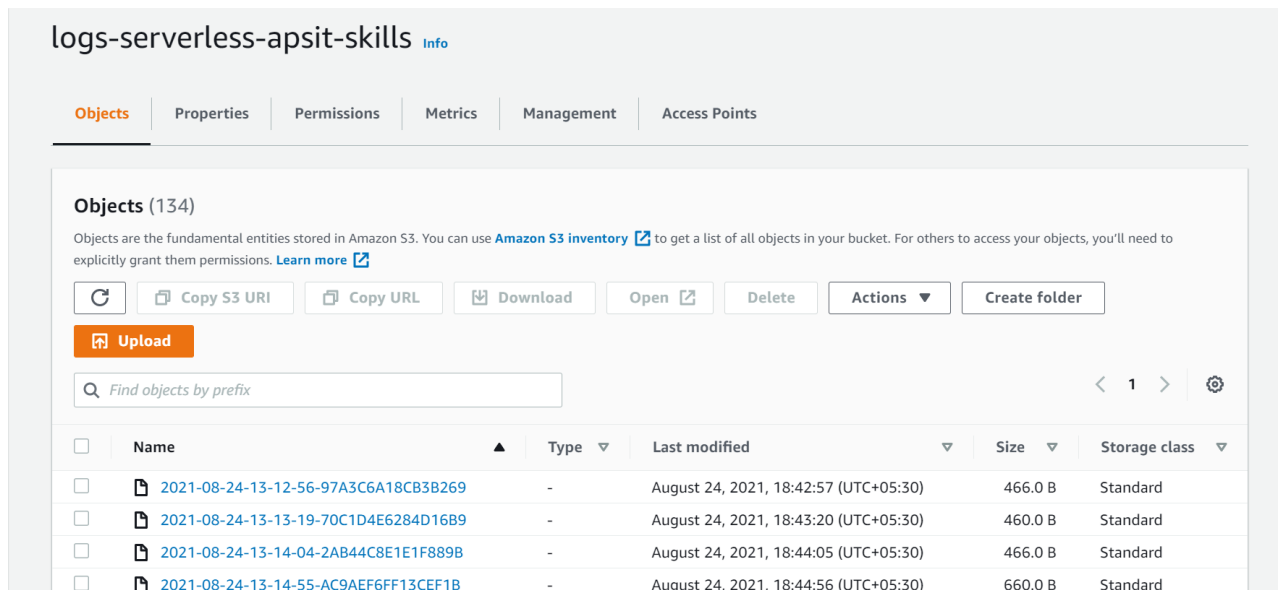


Fig. 5a - S3 bucket objects of logs

6) AWS CloudFront

Finally, we use the CDN service from AWS. We connect our s3 bucket that contains our static files. We set index.html as the default root object and create the distribution. Thus it then provides us with a domain name for our web application.

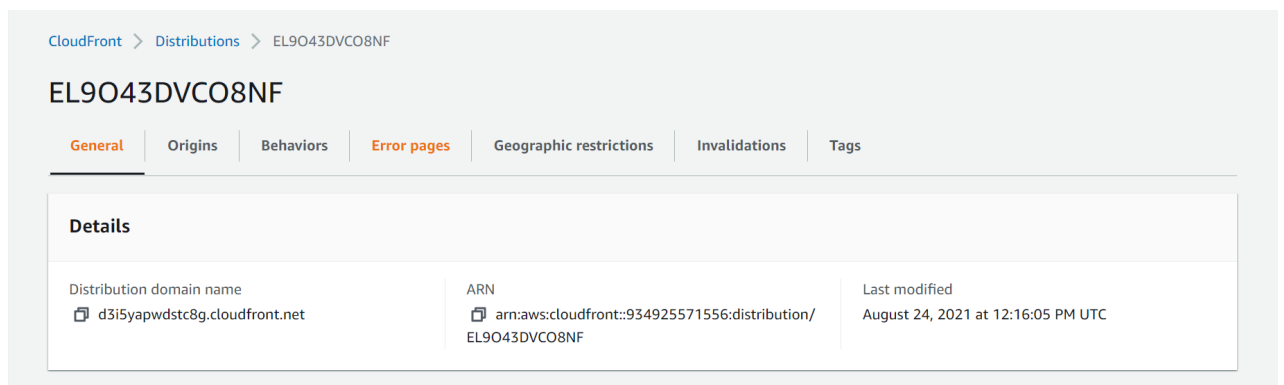


Fig. 6 - CloudFront distribution

OUTPUT

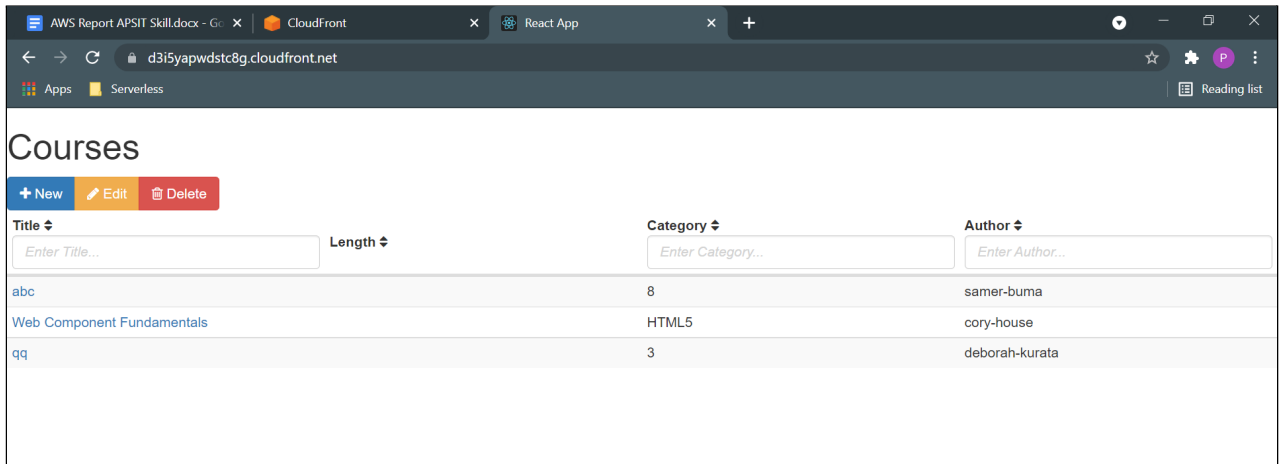


Fig. 7 - Web application Home page with URL as the distribution domain name

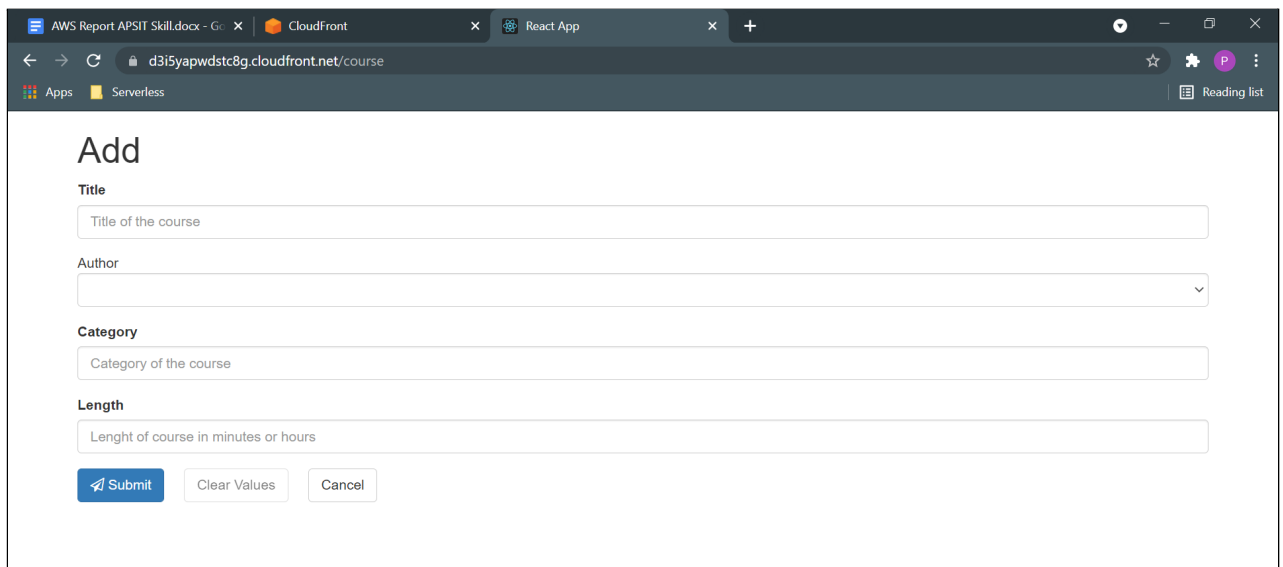


Fig. 7 - Web page to add a new course

ADVANTAGES AND DISADVANTAGES

Advantages

- No server management is necessary. It is managed by the vendor (AWS). This reduces the time and capital investment for the DevOps team.
- Dynamic provisioning is precise and real-time backed with a pay-as-you-go billing model broken down to milliseconds. This can drastically reduce the cost for some applications.
- Inherent scalability enables the application to handle unusually high spikes in workloads.
- Quick deployments and updates make it possible to quickly update, patch, fix, or add new features to the application.
- Combined with a CDN, the application has reduced latency because requests from the user no longer have to travel to the origin server.

Disadvantages

- Testing and Debugging are more complicated because developers do not have visibility into backend processes, and because the application is broken up into separate, smaller functions.
- Since clients are not assigned their discrete physical servers, serverless providers will often be running code from several of their customers on a single server at any given time. This is called multitenancy and can be a security concern if the application involves sensitive data.
- It is not built for long-running processes and can cost a lot more.
- Vendor lock-in is a big risk. If the vendor's service goes down the client application goes down as well. This also makes it difficult to switch vendors since every vendor has slight differences in their features and workflows.

APPLICATIONS

- To-do list application,
- Real-time file processing,
- To build an Extract Transfer Load (ETL) workflow for implementation of batch processing,
- Serverless document repository.