INSIDER REVIEW – PREDICTION TOOL FOR EMPLOYEE ATTRITION

A MINI PROJECT REPORT

Submitted by

PRADHUSHA A [20CS120]

PRAKALYA M [20CS121]

OBULI O [20CS116]

in partial fulfillment

of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

KPR INSTITUTE OF ENGINEERING AND TECHNOLOGY, COIMBATORE

(MAY 2023)

BONAFIDE CERTIFICATE

Certified that this project report "SERVERLESS WEB APPLICATION WITH HTTP API" is the Bonafide work of **"PRADHUSHA A (20CS120), PRAKALYA M (20CS121), OBULI O (20CS116)"** who carried out the project work under my supervision.

Dr.YUVARAJ.N

HEAD OF THE DEPARTMENT

Computer Science and Engineering

KPR Institute of Engineering and Technology

Arasur, Coimbatore – 641407

Mrs. KIRUTHIKA J K

SUPERVISOR,

Assistant professor

Computer Science and Engineering

KPR Institute of Engineering and Technology

Arasur, Coimbatore – 641407

Submitted for the university project Viva – Voce held on _____

INTERNAL EXAMINER

EXTERNAL EXAMINER

ACKNOWLEDGEMENT

The success shines when there is a team work and cooperation among the students and the faculties. We take this opportunity to express our gratitude and thanks to everyone who helped us during the project through every phase.

We would like to thank our beloved Chairman **Dr. K. P. Ramasamy** for his academic interest shown towards the students, by providing us excellent facilities and the constant support.

We would like to express our deep gratitude and sincere regards to our beloved and honorable Principal **Dr. M. Akila,** for providing us excellent facilities and encouragement during the course of study and project.

We owe a genuine gratitude to **Dr. Yuvaraj.N** Professor and Head, Department of Computer Science and Engineering for her encouragement and inspiration through every course.

We owe a sincere thanks to **Mrs. Kiruthika JK,** Assistant Professor, Department of Computer Science and Engineering for her support and guidance throughout the project as an Internal Guide.

Lastly, we thank our parents for their constant encouragement without which this assignment would not have been possible.

ABSTRACT

The rise of serverless computing has enabled developers to build modern web applications without worrying about the underlying infrastructure. This project proposes an architecture for building a serverless website using HTTP API in AWS that utilizes various AWS services such as Amazon S3, AWS Lambda, AWS API Gateway, Amazon CloudFront, and DynamoDB for data storage. The proposed architecture leverages modern web development technologies such as React, HTML, and CSS to provide a highly responsive and flexible solution.

The proposed architecture is designed to provide a scalable and cost-effective solution for building modern web applications. By utilizing serverless computing, developers can reduce the overhead associated with managing servers, while still benefiting from high performance and reliability. The use of AWS services such as Amazon S3, AWS Lambda, AWS API Gateway, and DynamoDB enables developers to easily integrate with other AWS services and third-party APIs, further enhancing the flexibility of the solution.

The proposed architecture also provides high scalability, as it automatically scales to handle any amount of traffic or data, making it suitable for applications with unpredictable traffic patterns. Additionally, it provides high availability through multi-region replication, automatic failover, and continuous backups, ensuring that data is always available and protected.

Overall, the proposed architecture provides a powerful and flexible solution for building modern web applications. Its use of serverless computing and AWS services such as Amazon S3, AWS Lambda, AWS API Gateway, Amazon CloudFront, and DynamoDB enables developers to build highly responsive .

A serverless web application using HTTP API, AWS Lambda, and Amazon DynamoDB can be abstracted using a multi-layered architecture. The presentation layer is responsible for handling user requests and can be implemented using an HTTP API. AWS Lambda can be used for the business logic layer, which implements the core logic of the application, while Amazon DynamoDB can be used for the data access layer, which interacts with the data storage layer. By using a multi-layered architecture, each layer can be changed independently without affecting the other layers, allowing for simplified development, reduced complexity, and improved scalability. Additionally, design patterns such as dependency inversion can further abstract and simplify the application. Overall, this approach provides a flexible and scalable solution for building serverless web applications.

The architecture for a serverless web application using HTTP API, AWS Lambda, and Amazon DynamoDB can be abstracted using a multi-layered architecture. This approach provides several benefits, including:

- Simplified development: By dividing the application into multiple layers, developers can focus on the specific responsibilities of each layer, resulting in simpler and more maintainable code.

- Reduced complexity: Each layer can be developed and tested independently, resulting in reduced complexity and fewer dependencies.

- Improved scalability: Because each layer can be scaled independently, the application can be more easily scaled to handle increases in traffic and usage.

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

AWS	AMAZON WEB SERVICES
CC	CLOUD COMPUTING
VM	VIRTUAL MACHINE
IAAS	INFRASTRUCTURE AS A SERVICE
PAAS	PLATFORM AS SERVICE
SAAS	SOFTWARE AS A SERVICE
S3	SIMPLE STORAGE SYSTEM
API	APPLICATION PROGRAMMING INTERFACE
HTTPS	HYPER TEXT TRANSFER PROTOCOL
REST API	REPRESENTATIONAL STATE TRANSFER APPLICATION PROGRAMMING INTERFACE

CHAPTER 1
INTRODUCTION

1.1 OVERVIEW

Serverless computing has emerged as a popular approach for building modern web applications due to its scalability, cost-effectiveness, and ease of deployment. The project proposes an architecture for building a modern web application using serverless computing and various AWS services such as Amazon S3, AWS Lambda, AWS API Gateway, Amazon CloudFront, and DynamoDB. The proposed architecture is designed to provide a scalable, cost-effective, and highly responsive solution that can handle any amount of traffic and data. By leveraging serverless computing and AWS services, developers can reduce the overhead associated with managing servers and focus on building the business logic for their application. Additionally, the use of AWS services enables developers to easily integrate with other AWS services and third-party APIs, further enhancing the flexibility of the solution. The architecture provides high scalability, reliability, and availability through automatic scaling, multi-region replication, automatic failover, and continuous backups. Overall, the proposed architecture provides a powerful and flexible solution for building modern web applications.

1.2 CLOUD COMPUTING

Cloud computing refers to the delivery of computing services, including servers, storage, databases, software, and networking, over the internet (the "cloud"). Instead of hosting these resources on-premises, organizations can use cloud computing services provided by third-party providers. Cloud computing allows for easy access to computing resources on-demand, without the need for extensive hardware or infrastructure investment. It provides a highly flexible and scalable solution for organizations of all sizes, while also offering benefits such as cost-effectiveness, reliability, security, and agility.

Cloud computing provides a wide range of deployment models, service ssmodels, and deployment options, allowing organizations to choose the cloud computing solution that best meets their specific needs. Popular cloud computing deployment models include public cloud, private cloud, and hybrid cloud, while service models include Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS).

1.3 AWS

AWS (Amazon Web Services) is a cloud computing platform provided by Amazon. It offers a wide range of cloud computing services, including compute, storage, database, analytics, machine learning, artificial intelligence, networking, security, and more. AWS provides customers with a highly scalable, reliable, and secure infrastructure that can be used to host web applications, mobile applications, IoT (Internet of Things) applications, and much more.

Some of the popular services provided by AWS include EC2 (Elastic Compute Cloud) for virtual machine instances, S3 (Simple Storage Service) for object storage, RDS (Relational Database Service) for managed relational databases, Lambda for serverless computing, and many others. AWS is used by organizations of all sizes, from startups to large enterprises, in a wide range of industries, including healthcare, finance, education, media, and more. AWS provides a highly flexible and scalable platform that can be tailored to meet the specific needs of each customer, and offers a pay-as-you-go pricing model that allows organizations to only pay for the services they use, without incurring any upfront costs. Overall, AWS is a powerful tool for organizations looking to take advantage of the benefits of cloud computing.

1.3 SERVERLESS WEB APPLICATION

A serverless web application is a type of application that is built and hosted using cloud computing platforms that provide serverless computing services. In a serverless architecture, the cloud provider manages the infrastructure and automatically scales the application based on usage, which means that developers do not need to worry about managing and provisioning servers.

Serverless web applications are typically built using a combination of backend services, such as AWS Lambda, Azure Functions, or Google Cloud Functions, and frontend services, such as AWS Amplify or Firebase. These services allow developers to focus on writing code and building features, rather than worrying about server infrastructure.

One of the benefits of serverless web applications is that they can be highly scalable and cost-effective. The cloud provider automatically manages the scaling of resources, which means that the application can handle large volumes of traffic without the need for manual intervention. Additionally, serverless applications typically only incur costs when they are in use, which can result in significant cost savings compared to traditional server-based applications.

There are also some challenges associated with building serverless web applications. For example, serverless architectures can introduce new complexities around managing dependencies, testing, and debugging. Additionally, some applications may not be well-suited for serverless architectures due to factors such as long-running processes or high resource usage.

Overall, serverless web applications can be a powerful tool for building scalable, cost-effective web applications, but they require careful consideration of the specific requirements of the application and the capabilities of the chosen cloud provider.

1.4 DynomoDB

Amazon DynamoDB is a managed NoSQL database service provided by Amazon Web Services (AWS). DynamoDB is designed to provide fast and predictable performance at any scale, and is highly scalable, reliable, and fully managed. It is a fully managed database service, which means that AWS handles the operational aspects of the database, such as infrastructure provisioning, software patching, and data backups.

DynamoDB is a NoSQL database, which means that it does not use the traditional SQL query language for data retrieval and manipulation. Instead, DynamoDB uses a key-value and document data model, with each item in the database having a unique primary key, and the ability to store multiple attributes and values within the item. DynamoDB can handle millions of requests per second, making it a great fit for high-traffic, high-scale applications.One of the unique features of DynamoDB is its ability to provide consistent and predictable performance, even at scale. This is achieved through the use of partitioning, which allows data to be distributed across multiple servers to ensure that the workload is evenly balanced. Additionally, DynamoDB offers various levels of consistency, ranging from eventual consistency to strong consistency, depending on the specific needs of the application.

DynamoDB also offers a range of advanced features, such as global secondary indexes, which allow developers to create additional indexes for data lookup beyond the primary key, and DynamoDB Streams, which can be used to capture changes to

data in real-time and enable use cases such as data replication and real-time analytics.

Overall, DynamoDB is a powerful and flexible database service that can be used for a wide range of use cases, including web and mobile applications, gaming, IoT, and more. However, it is important to carefully consider the specific requirements of the application and to understand the pricing model, which is based on a combination of provisioned throughput capacity and storage usage.

1.5 LAMBDA FUNCTION

Lambda function is a compute service offered by Amazon Web Services (AWS) that allows developers to run code without provisioning or managing servers. It is a serverless computing service that enables developers to write, deploy, and execute code in response to specific events or requests.

Lambda functions can be written in a variety of programming languages, including Python, Java, Node.js, and C#, among others. They are triggered by a specific event, such as an API request, a change in an Amazon S3 bucket, or a message on an Amazon SNS topic.

Lambda functions are billed based on the number of requests and the amount of time the code is running. This means that developers only pay for the compute time used to run their code and do not need to pay for any idle time when the function is not in use.

Lambda functions can be used for a variety of use cases, including serverless web applications, real-time stream processing, data processing and analysis, and more. They can be used in conjunction with other AWS services, such as Amazon S3, Amazon DynamoDB, Amazon SNS, and Amazon API Gateway to create highly

scalable and flexible applications.

One of the benefits of using Lambda functions is that developers do not need to manage any infrastructure, such as servers or operating systems. This allows them to focus on writing code and building features, rather than worrying about managing servers or scaling infrastructure.

1.6 HTTP API

HTTP API, or HTTP Application Programming Interface, is a type of API that enables communication between applications or services over the Hypertext Transfer Protocol (HTTP). HTTP APIs are designed to be lightweight and simple, and are often used for building microservices and serverless architectures.

Unlike traditional REST APIs, which often require complex resource mapping and multiple endpoints, HTTP APIs provide a simplified interface for creating and managing APIs. They support a limited set of HTTP methods, including GET, POST, PUT, PATCH, and DELETE, and typically rely on JSON or XML payloads for data exchange.

HTTP APIs are often used in conjunction with other serverless technologies, such as AWS Lambda or Azure Functions, to create highly scalable and flexible applications. They can be used to build serverless web applications, IoT applications, and other types of microservices.

One of the benefits of using HTTP APIs is that they can be highly efficient and performant, due to their lightweight and simplified design. Additionally, they can be easier to manage and deploy than traditional REST APIs, as they often require fewer resources and have fewer moving parts.

1.7 REST API

REST API stands for Representational State Transfer Application Programming Interface. It is an architectural style used for building web services that allow communication between different systems over the internet. RESTful web services are based on the principles of REST, which is a set of guidelines for building scalable and flexible web applications. RESTful web services provide a set of HTTP methods, such as GET, POST, PUT, and DELETE, that can be used to create, retrieve, update, and delete resources. Each resource is identified by a unique URL, and is represented in a specific format, typically JSON or XML.

One of the key principles of REST is that it is stateless, meaning that each request to the API contains all of the information necessary to complete the request. This allows the API to be highly scalable and flexible, as requests can be processed in parallel without requiring any shared state.RESTful web services are often used in conjunction with other web technologies, such as OAuth for authentication and authorization, and Swagger for API documentation. They can be used to build a wide range of applications, including web and mobile applications, IoT devices, and more.

One of the benefits of using RESTful web services is that they are widely supported and can be integrated with a variety of different programming languages and frameworks.

1.8 S3 BUCKET

Amazon S3 (Simple Storage Service) is an object storage service provided by Amazon Web Services (AWS). It is a highly scalable and durable storage service that allows you to store and retrieve data from anywhere on the web. S3 buckets are the containers used to store objects in S3. An object in S3 can be a file, an image, a video, or any other type of data. Each object is stored in a bucket and is assigned a unique identifier called an object key.

Key features and benefits of using Amazon S3:

- Scalability: S3 can store an unlimited amount of data, and you can easily scale your storage resources up or down as your needs change.

- Durability: S3 is designed to provide 99.999999999% durability of objects over a given year. This means that even if one copy of an object is lost or damaged, S3 automatically retrieves another copy.

- Security: S3 provides several security features, including encryption, access control, and audit logs, to help protect your data.

- Performance: S3 is optimized for high performance, with low latency and high throughput for both read and write operations.

- Cost-effective: S3 pricing is based on the amount of data stored, the number of requests made, and the amount of data transferred. S3 also provides cost optimization tools to help you reduce your storage costs.

CHAPTER 2

LITERATURE SURVEY

1. "Building Serverless Web Applications" by Sam Kroonenburg and Ryan Kroonenburg: This book provides a comprehensive overview of serverless web application development, with a focus on AWS Lambda, API Gateway, DynamoDB, and other AWS services.

2. "Serverless Architectures on AWS" by Peter Sbarski: This book provides an in-depth look at how to design and implement serverless architectures using AWS Lambda, API Gateway, DynamoDB, and other AWS services.

3. "Serverless Applications with Node.js" by Slobodan Stojanovic and Aleksandar Simovic: This book provides a detailed guide to building serverless applications using Node.js and AWS Lambda, with a focus on integrating with other AWS services such as API Gateway and DynamoDB.

4. "Practical Serverless Architectures" by Jianping Zhu: This book provides a practical guide to designing and implementing serverless architectures, with a focus on AWS Lambda, API Gateway, and DynamoDB.

CHAPTER 3

SYSTEM DESIGN

3.1 SERVERLESS WEB APPLICATION WORKFLOW IN HTTP API

The workflow of a serverless web application using HTTP API in AWS typically follows these steps:

1. Design the architecture: First, you need to design the architecture of the serverless web application, deciding on the services you will use, such as AWS Lambda, API Gateway, DynamoDB, and others.

2. Develop the code: Next, you need to develop the code for the various functions and APIs that make up the web application, using the appropriate programming languages and tools.

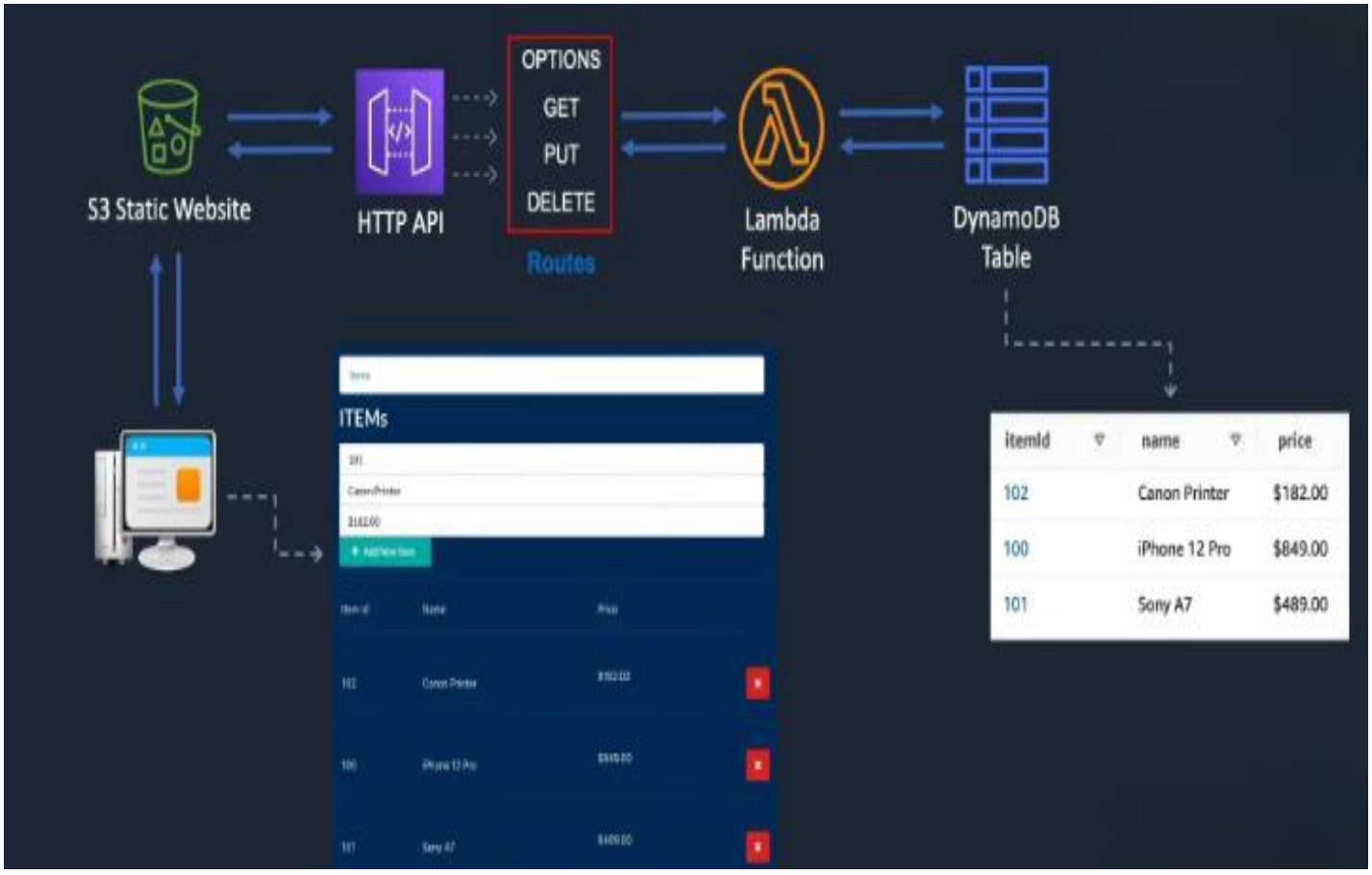
3. Test the application: After developing the code, you need to test the application to ensure that it is functioning as expected and meeting the requirements.

4. Deploy the application: Once you have tested the application, you can deploy it to AWS using the appropriate deployment tools and techniques.

5. Monitor the application: After deploying the application, you need to monitor it to ensure that it is running smoothly, and to detect and resolve any issues that arise

6. Maintain and update the application: Finally, you need to maintain and update the application over time, making changes and improvements as needed to ensure that it continues to meet the needs of users.

3.2 ARCHITECTURE DIAGRAM



CHAPTER 4

IMPLEMENTATION

4.1 PROPOSED SYSTEM

- The proposed system for a serverless website using HTTP API AWS involves utilizing various AWS services, including Amazon S3, AWS Lambda, API Gateway, CloudFront, and DynamoDB for Data storage.

- The system is designed to be completely serverless, meaning that there are no servers to manage or provision. Instead, AWS Lambda functions are used to handle HTTP requests and responses, while API Gateway is used to manage API endpoints and create custom domain names.

- Amazon S3 is used to store static website assets such as HTML, CSS, and JavaScript files, while CloudFront is used to distribute the content and provide faster access to users through a global network of edge locations.

- DynamoDB is used for data storage, allowing for fast and scalable storage of data in a NoSQL format. The data is stored in tables and accessed using Lambda functions, which can be triggered by API Gateway endpoints.

- The proposed system provides a highly scalable and efficient way to build and host serverless websites using AWS services.

4.2 METHODOLOGY

STEP 1: Configuration of CloudFormation Stack

1. Go to CloudFormation main page.
2. Create CloudFormation stack.
3. Choose upload template file.
4. Select YAML file from local drive.
5. Optional - click View in Designer if want to see AWS stack in designer.
6. Click Next.
7. Acknowledge and click Create Stack.
8. Stack creation in progress.
9. Stack created.

STEP 2: Creation of S3 Bucket

1. Create S3 bucket named "item-frontend-static-hosting". This bucket's URL has to be specified in the backend API CORS configuration for allowed origins.
2. Choose appropriate region as well as allow Public access
3. Enable Static website hosting on this bucket.
4. Check Enable, Host Static Website. And specify index.html as index document.
5. Edit Bucket Policy.

STEP 3: API Gateway Configuration

1. Go to API Gateway section to check API created as part of CF stack.
2. Need to copy Invoke URL to be configured in frontend config
3. Need to modify four things: Routes, Integrations, Stages, CORS and then Deploy.

4. Click Stage on left navigation. Then Click Create.
5. New stage name "prod". Click Create at bottom
6. Manage Integrations tab. A default integration exists but we click Create to create a new one
7. Specify integration type, AWS region and Lambda function. This Lambda function is also created as part of CF stack. Click Create at bottom
8. New integration is ready. Note integration ID
9. Now need to create 6 routes: /items (OPTIONS, GET, PUT) and /items/{id} (OPTIONS, DELETE, GET)

GET /items

PUT /items

GET /items/{id}

DELETE(/items/{id}

OPTIONS /items

OPTIONS /items/{id}

10. Attach the NEW integration to the Lambda function to each of the 6 routes. Click Route. Then click Attach integration.
11. Select correct integration matching integration ID. Click Attach integration
12. Repeat for all 6 routes
13. Configure CORS. All 6 fields should be configured.

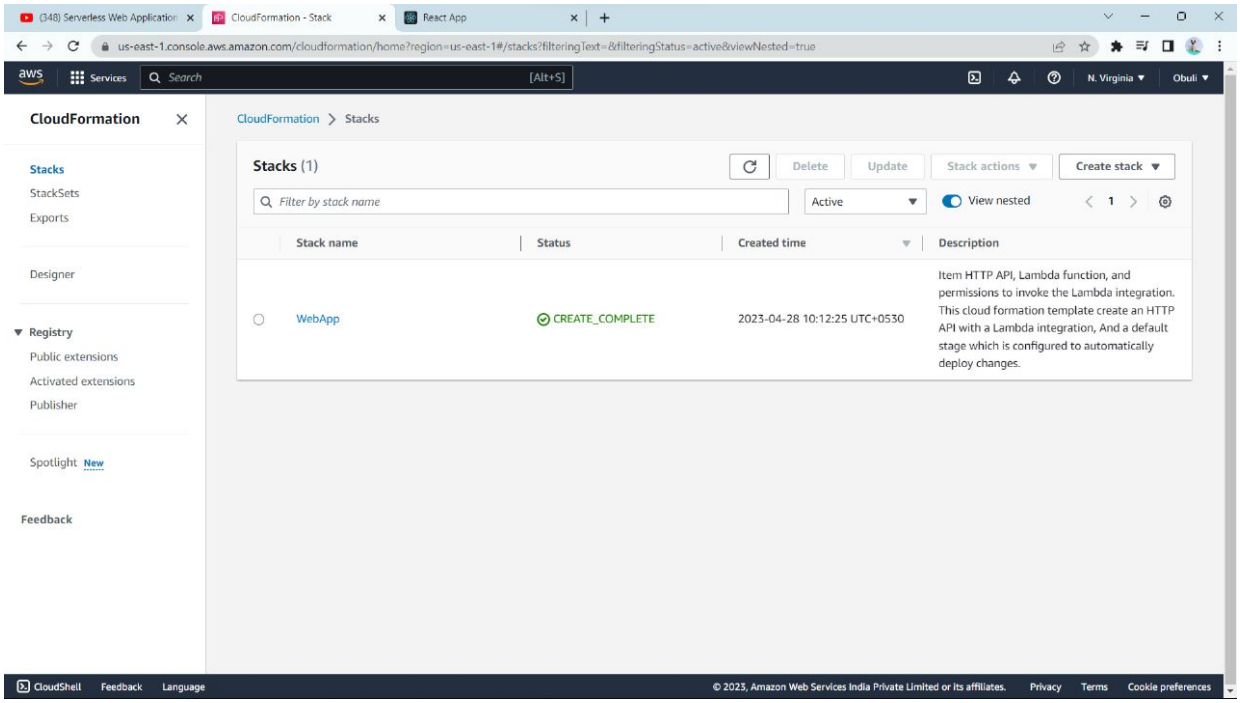
STEP 4: NODEJS

1. The invoke URL's API Id must be configured in the frontend config.
2. From a command prompt, go to client folder and install all dependencies by running "npm install".
3. Now run "npm run build" to create a production build in "build" subfolder.
4. Upload the build subfolder content for the frontend client application into the bucket under Objects tab.

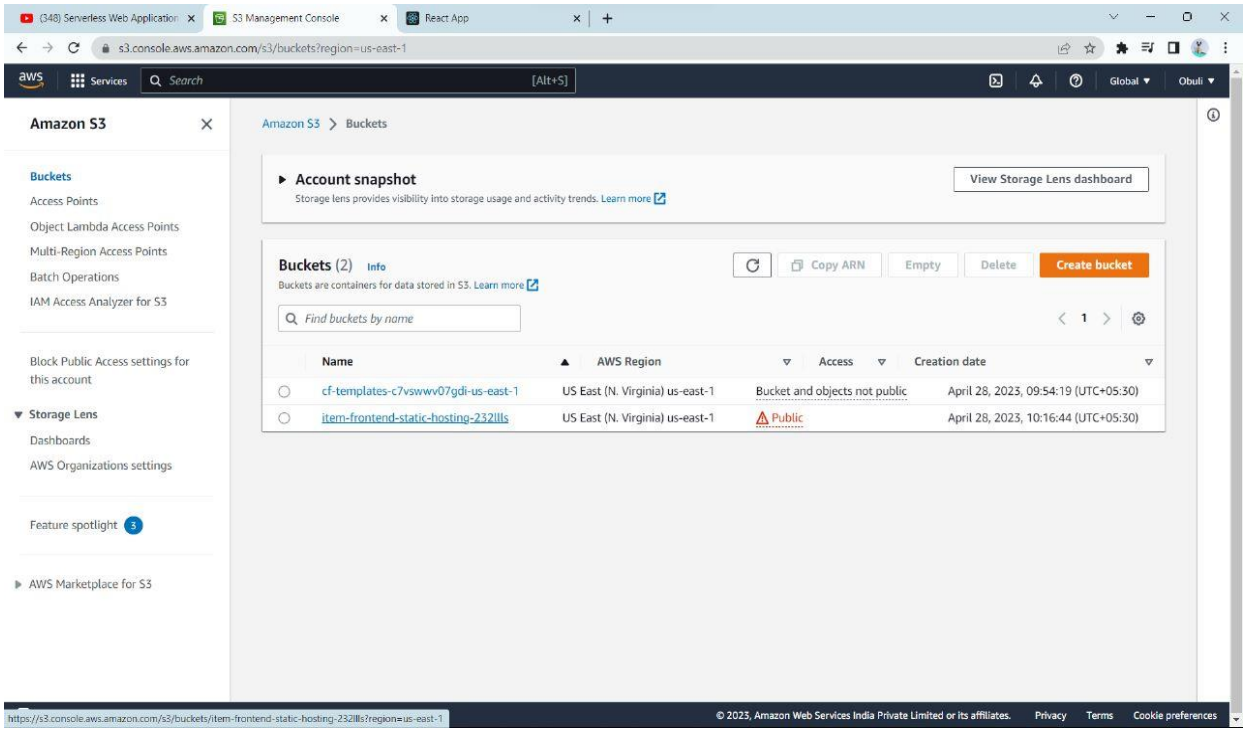
5. Under "build" there is "static" folder which itself has three subfolders "css", "js" and "media". Make sure the proper structure is created in S3 and Files are uploaded into respective folders on the local machine.
6. After all uploading is done. open index.html at the root of S3 bucket. Inside it shows an HTTPS URL. Use that to access the frontend application.
7. The frontend application by default opens a dashboard page which shows the Items link where at the top, a form allows you to add new items by specifying a unique ID, name and price. And below that a grid shows the items that have been added. Items can be deleted. But can't be updated.

4.3 IMAGES

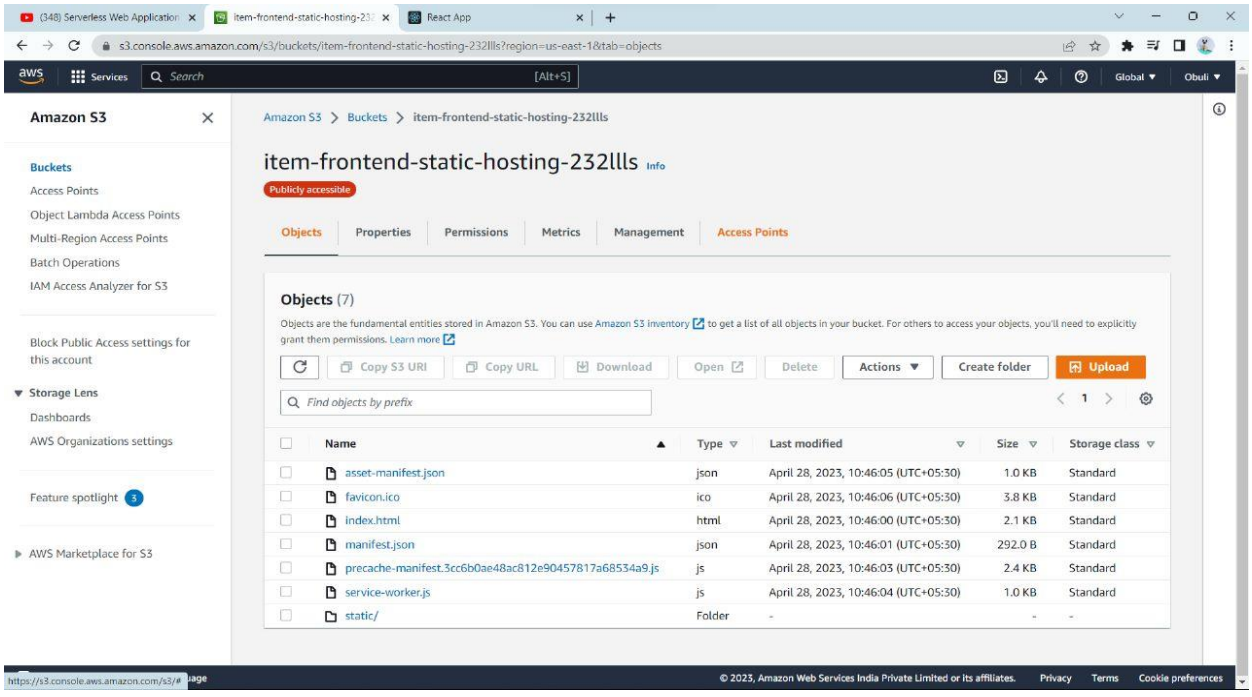
STEP 1 : CREATING STACK IN CLOUD FORMATION



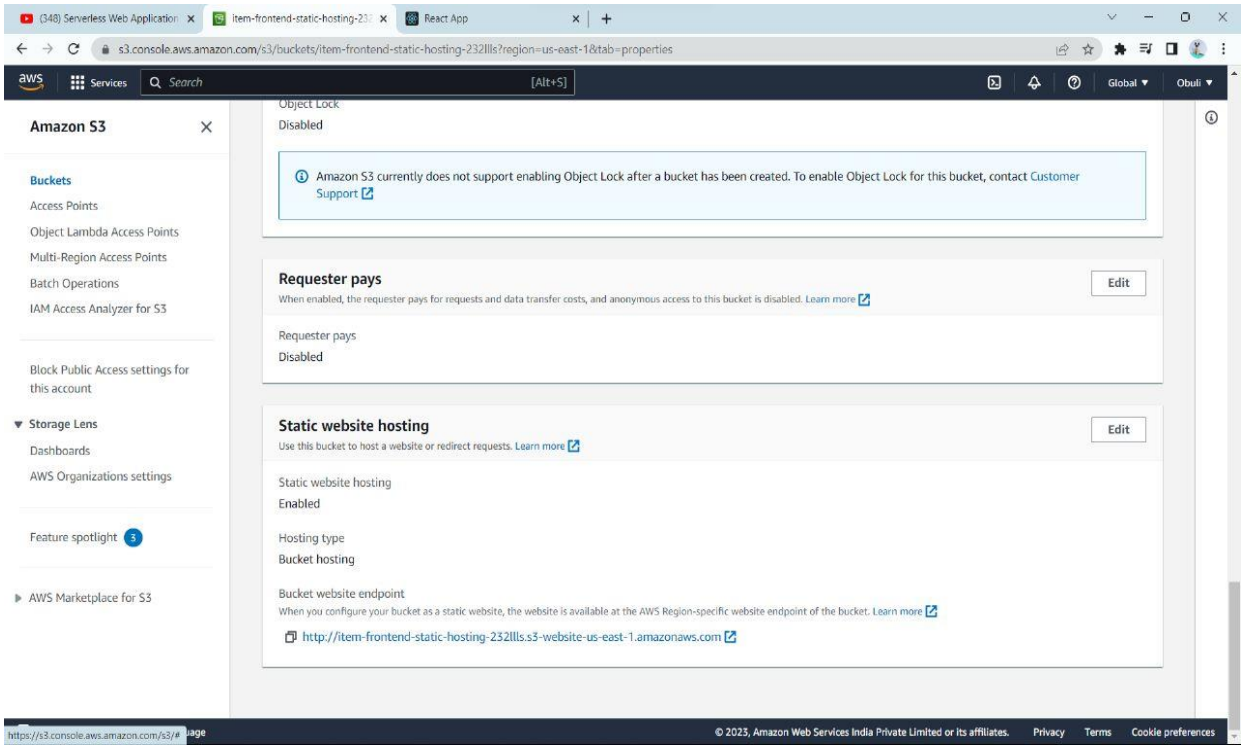
STEP 2 - CREATION OF BUCKETS



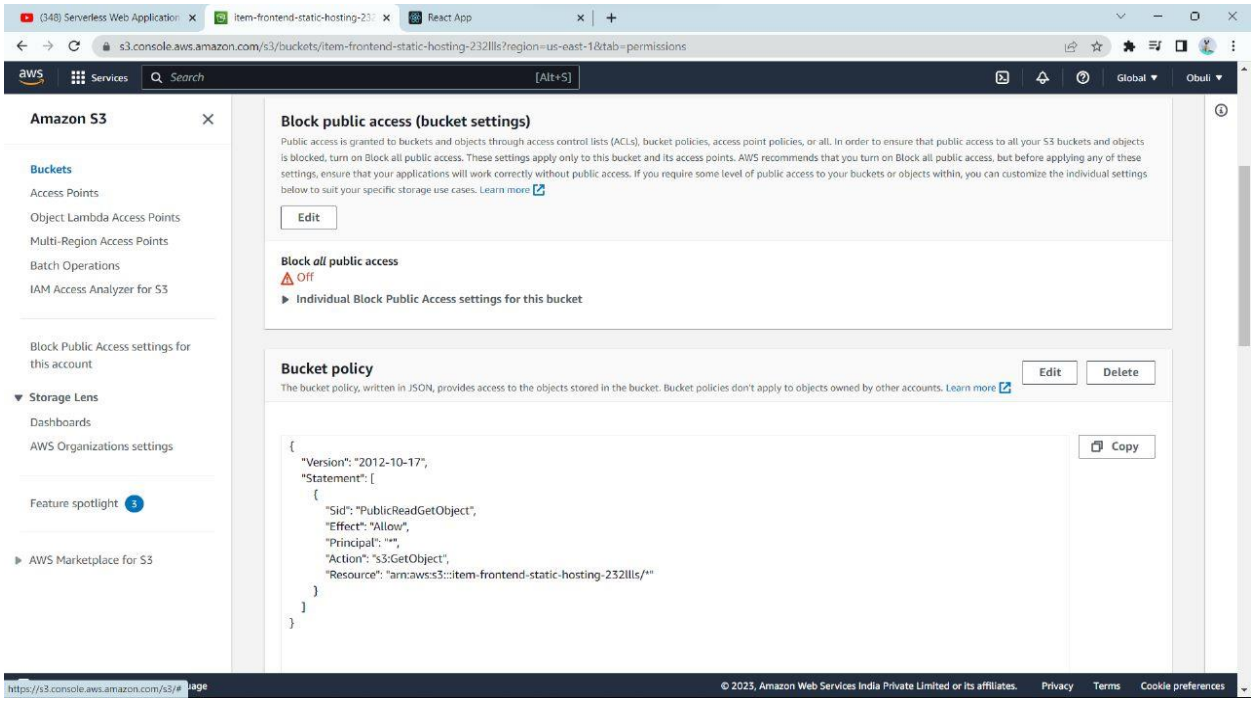
STEP 3:FILES UPLOADED IN S3 BUCKET



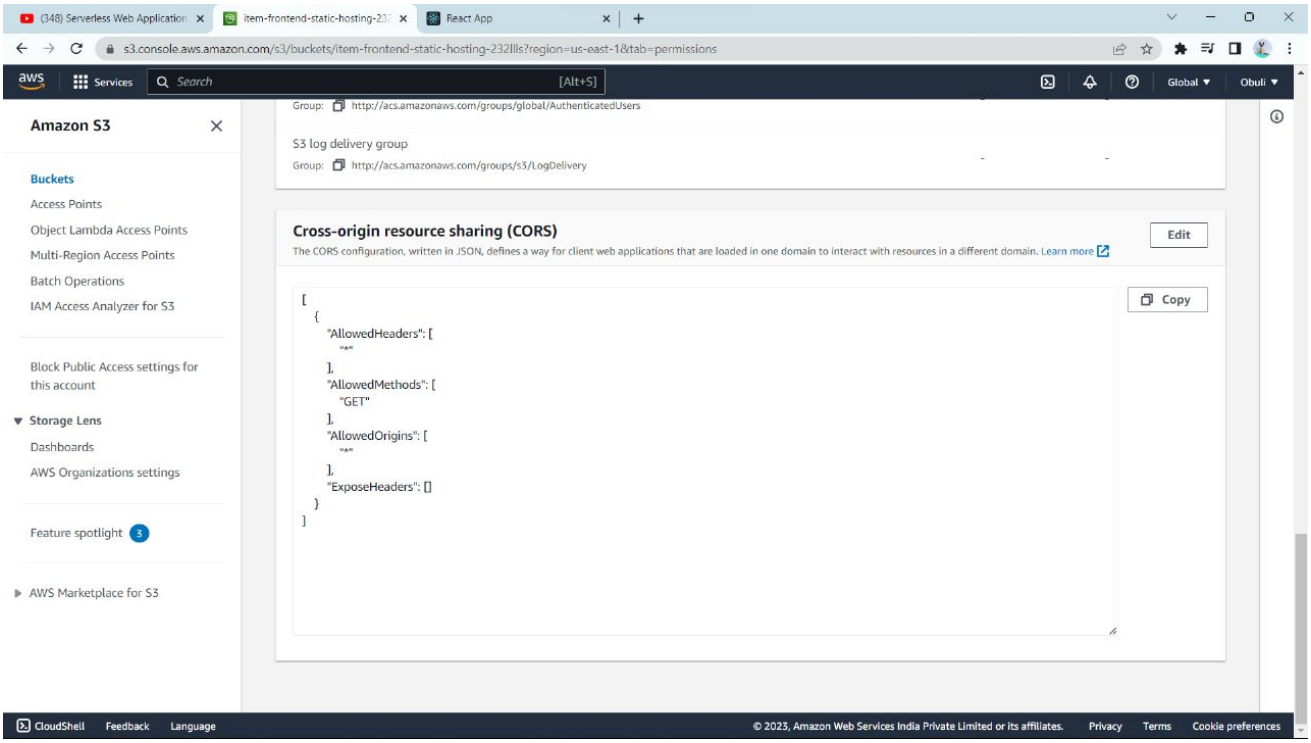
STEP 4: ENABLING STATIC WEBSITE PROPERTY IN S3 BUCKET



STEP 5: DISABLING BLOCK PUBLIC ACCESS AND SETTING BUCKET POLICY

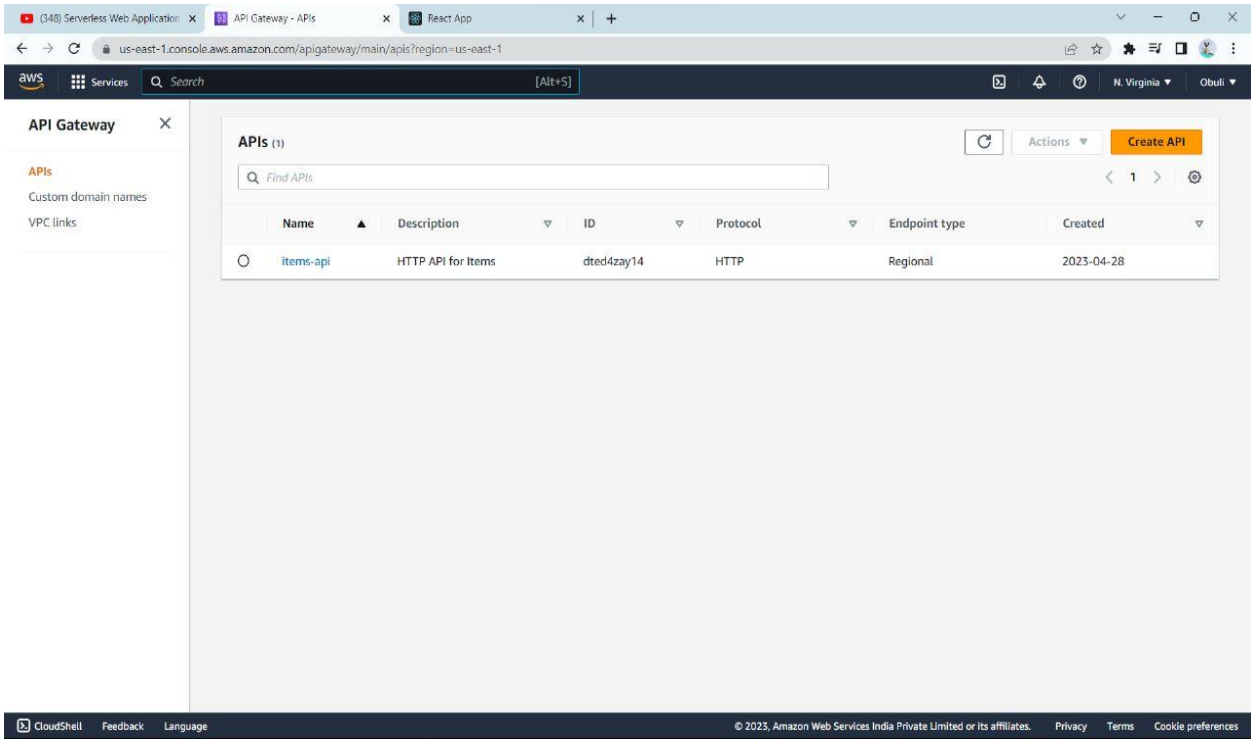


STEP 6: SETTING UP CORS

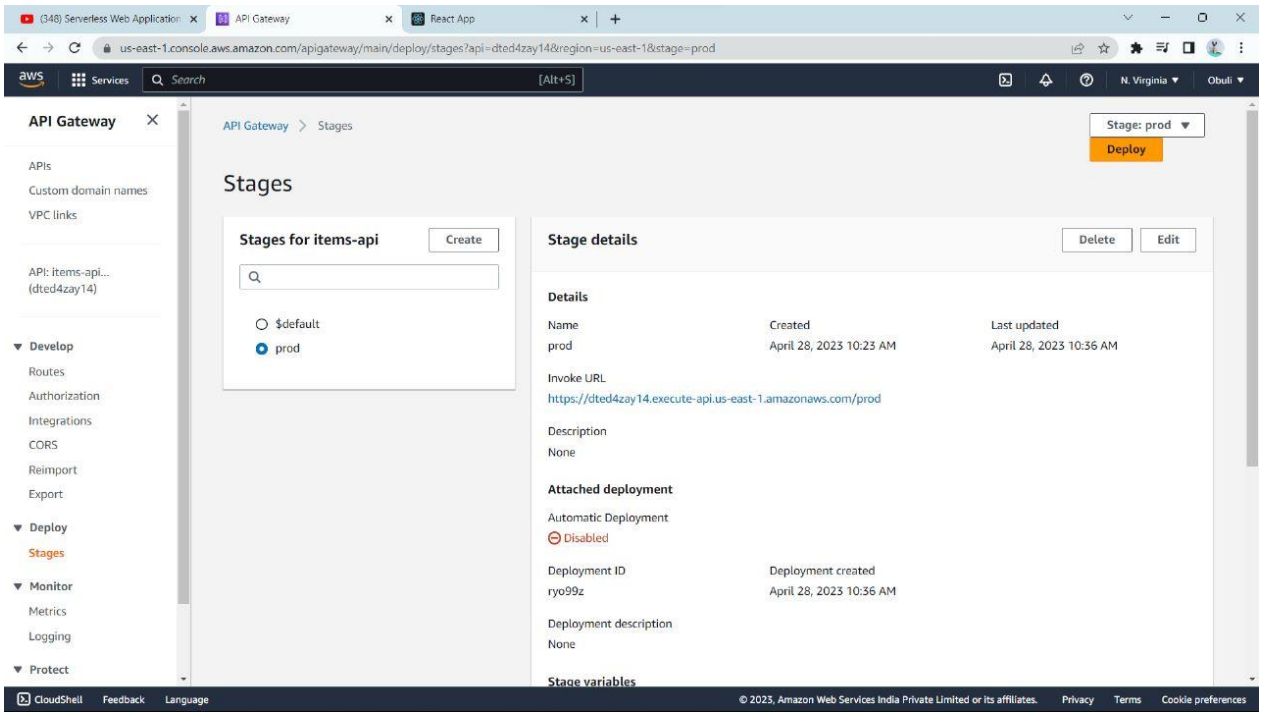



```
# API Gateway Configuration

Go to API Gateway section to check API created as part of CF stack. items-api is created
Need to copy Invoke URL to be configured in frontend config
Need to modify four things: Routes, Integrations, Stages, CORS and then Deploy
Click Stage on left navigation. Then Click Create
New stage name "prod". Click Create at bottom
Go to Integrations
Manage Integrations tab. A default integration exists but we click Create to create a new one
Specify integration type, AWS region and Lambda function. This Lambda function is also created as part of CF stack. Click Create at bottom
New integration is ready. Note integration ID
Now need to create 6 routes: /items (OPTIONS, GET, PUT) and /items/{id} (OPTIONS, DELETE, GET)
    GET /items
    PUT /items
    GET /items/{id}
    DELETE/items/{id}
    OPTIONS /items
    OPTIONS /items/{id}
Attach the NEW integration to the Lambda function to each of the 6 routes. Click Route. Then click Attach integration.
Select correct integration matching integration ID. Click Attach integration
Repeat for all 6 routes
Configure CORS. All 6 fields should be configured:
    The bucket URL (for N.Virgina buckets) should be: https://item-frontend-static-hosting-232llls.s3.amazonaws.com
    "Access-Control-Allow-Origin" has to be specified after creating the S3 bucket as its name is used in the URL. Click Save.
Now click Deploy at top right and select prod Stage for deployment
💡
* Invoke URL: https://dted4zay14.execute-api.us-east-1.amazonaws.com/prod

* Integration ID: niudy51
```

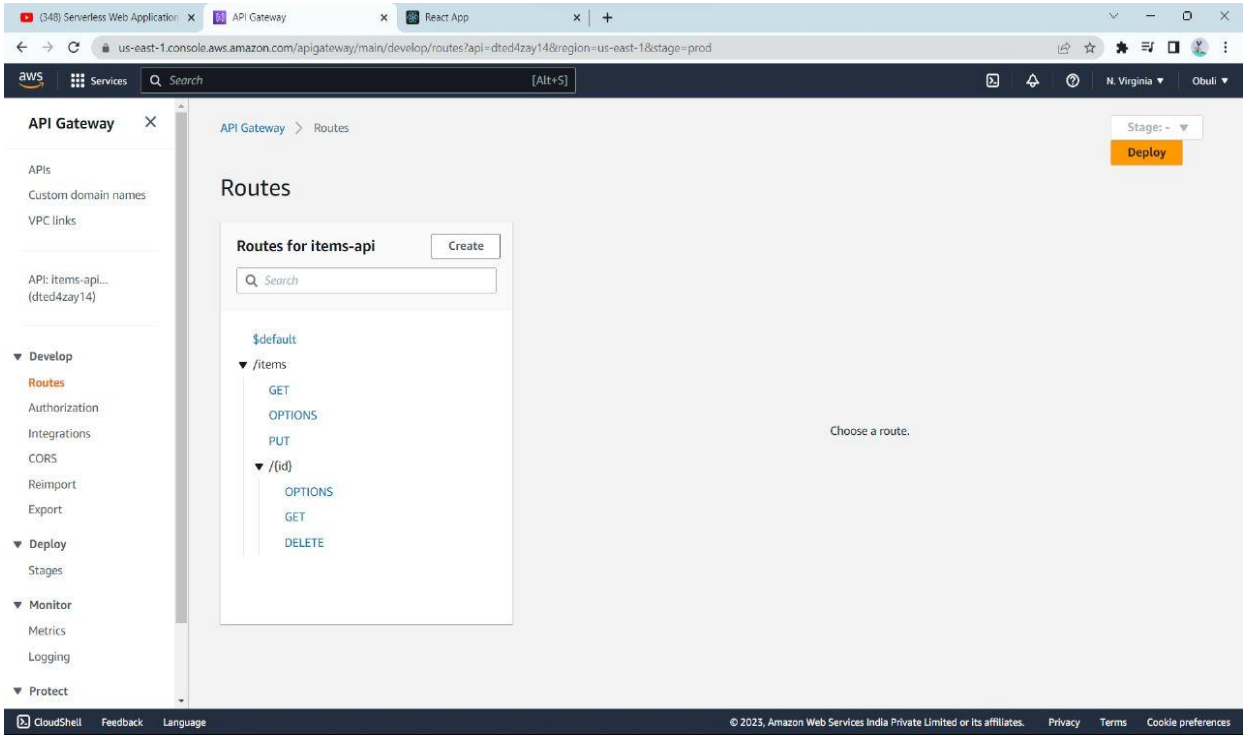
STEP 7: ITEMS-API IS CREATED



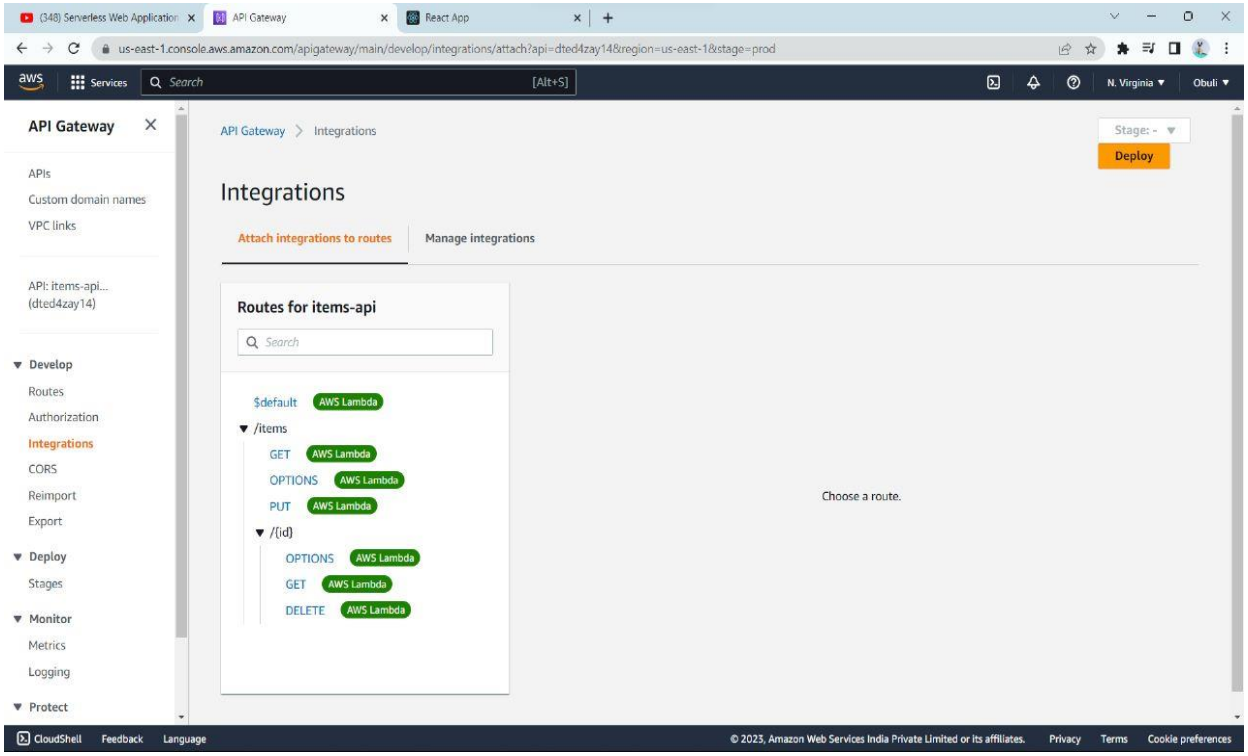
STEP 8: PROD IS CREATED



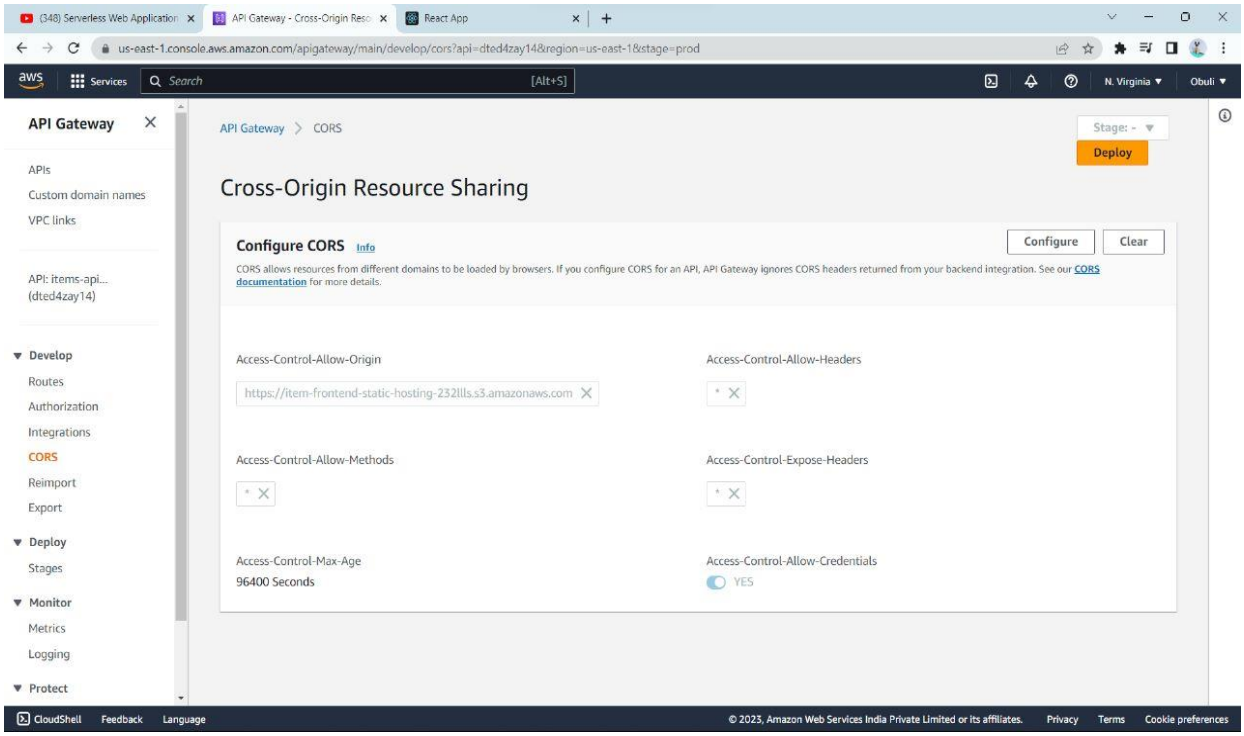
STEP 9: ROUTES USED IN WEBAPP IS CREATED



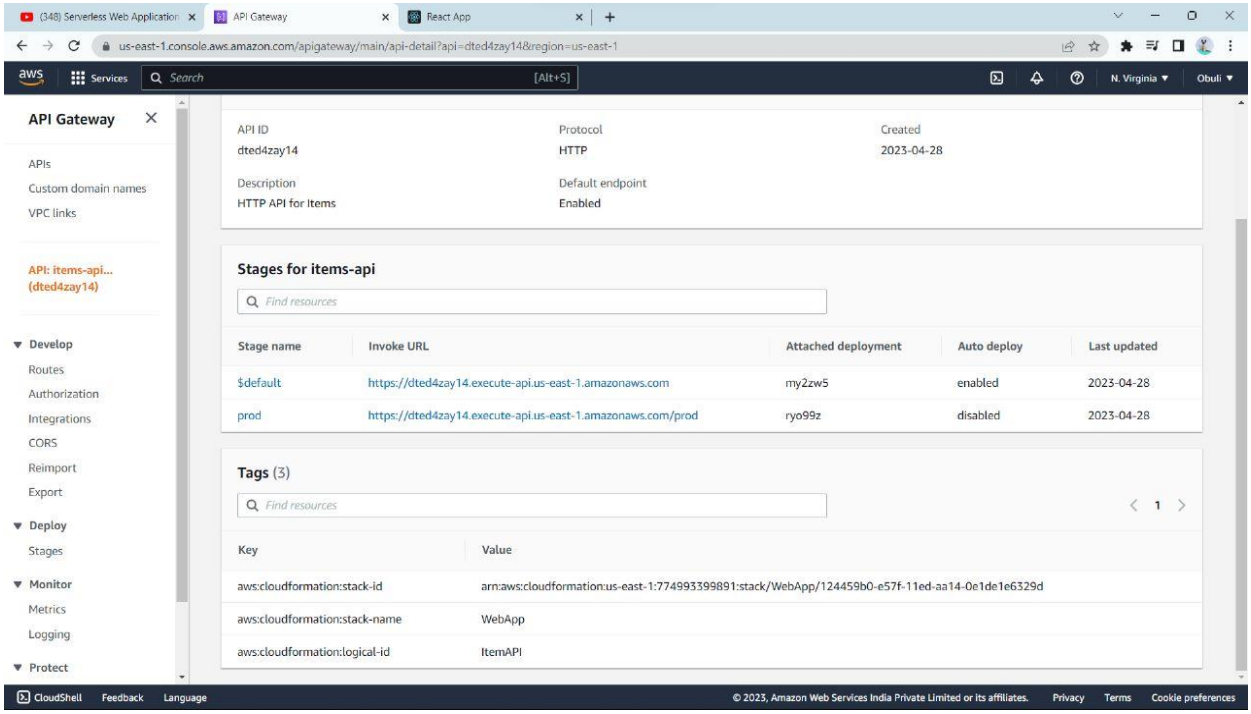
STEP 10: INTEGRATIONS ATTACHED TO ROUTES

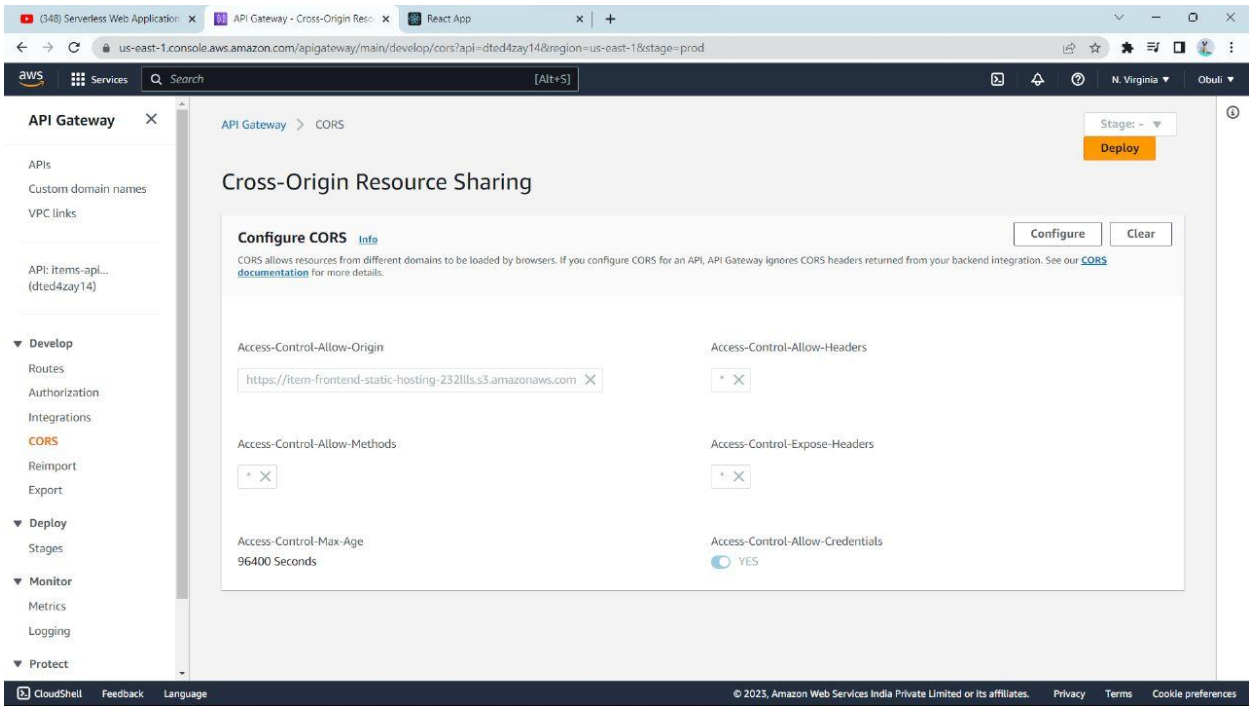


STEP 11: CONFIGURATION OF CORS

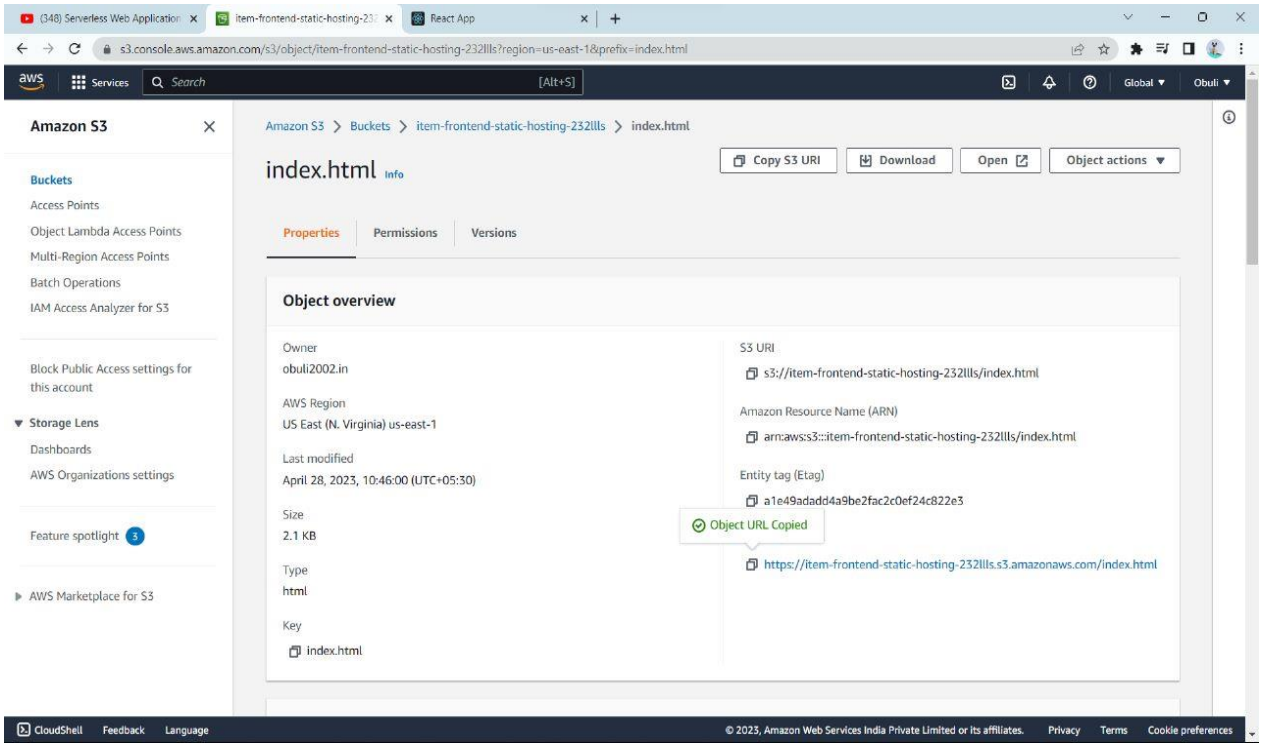


STEP 12: STAGE PROD WITH INVOKE URL

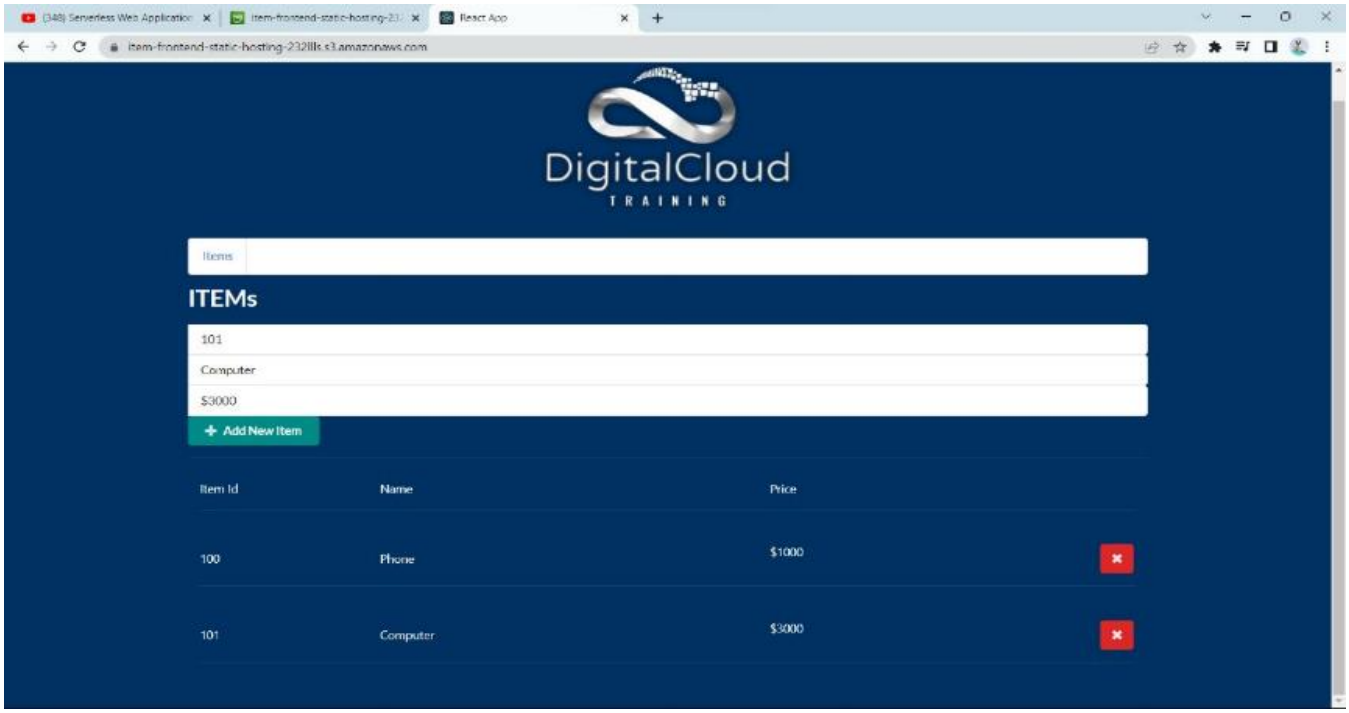


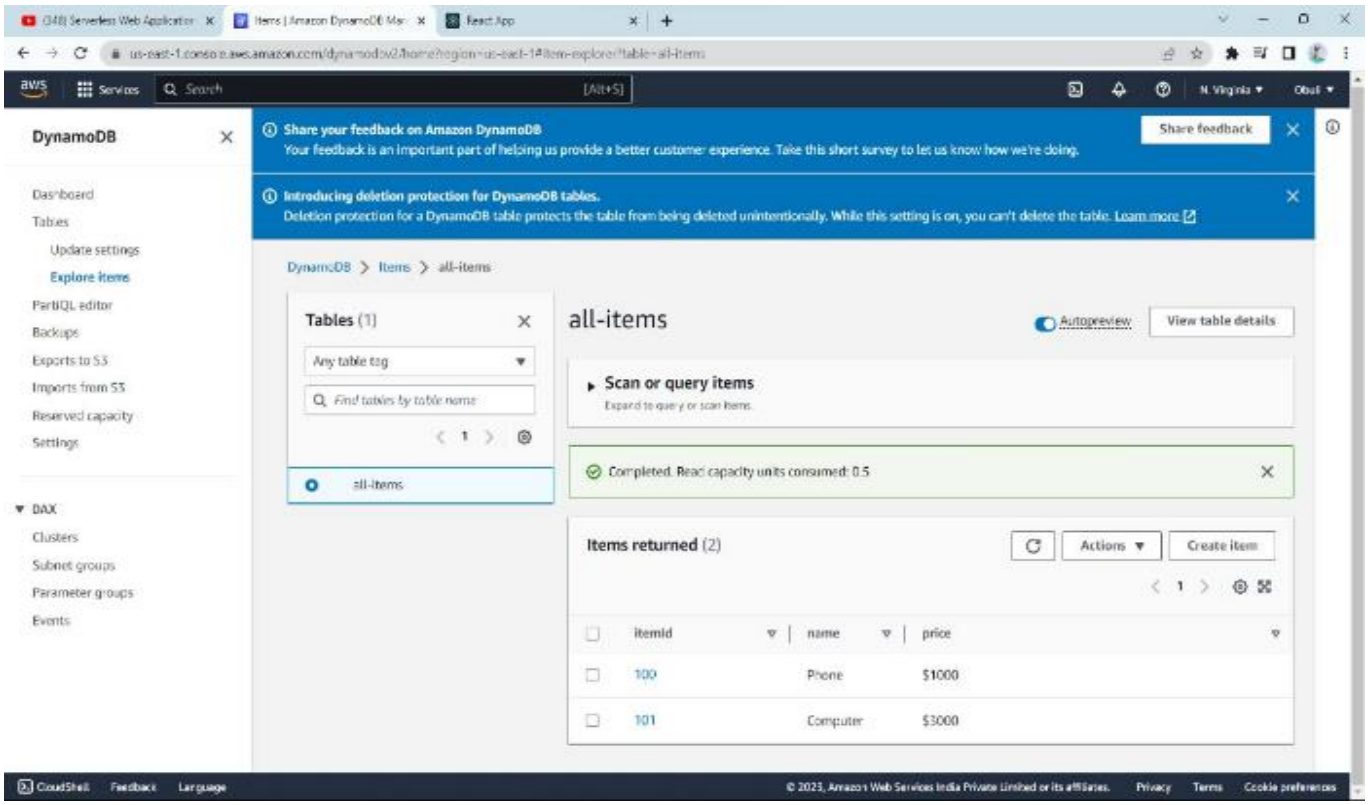


STEP 13: OBJECT URL TO RUN WEB APP



WEBAPP OUTPUT





CHAPTER 5
SYSTEM REQUIREMENTS

5.1 SOFTWARE SPECIFICATION

1. AWS Account: An AWS account is required to use the AWS services, including AWS Lambda, DynamoDB, and HTTP API.

2. AWS Lambda: AWS Lambda is a serverless compute service that allows you to run code without provisioning or managing servers. It will be used to handle user registration, login, user profile creation and updating, content creation and retrieval, and notifications.

3. AWS DynamoDB: AWS DynamoDB is a NoSQL database service that provides fast and predictable performance. It will be used to store and manage user data and content data.

4. AWS HTTP API: AWS HTTP API is a fully managed API Gateway service that makes it easy to create, deploy, and manage APIs at any scale. It will be used to handle incoming HTTP requests and route them to the appropriate AWS Lambda function.

5. AWS SDK: The AWS SDK provides a set of libraries and tools to build AWS applications using various programming languages. It will be used to interact with AWS services such as Lambda and DynamoDB.

6. Frontend Framework: A frontend framework such as React, Angular, or Vue.js will be used to build the user interface.

5.2 HARDWARE SPECIFICATION

As a serverless website using AWS Lambda, DynamoDB, and HTTP API, the hardware requirements are managed by AWS, and the website does not require any dedicated hardware. AWS will automatically provision and manage the necessary infrastructure and resources to handle the incoming requests and execute the Lambda functions.

However, it is recommended to have a reliable and fast internet connection to ensure optimal performance and user experience. Also, developers will need a computer with sufficient resources to run the necessary software, such as an IDE, text editor, and the AWS SDK. The recommended hardware specifications for a developer's computer are:

- Processor: Intel Core i5 or higher
- RAM: 8 GB or higher
- Storage: 256 GB SSD or higher
- Internet connection: 10 Mbps or higher

Additionally, it is important to consider the hardware requirements for load testing and performance testing, as well as disaster recovery and backup mechanisms.

CHAPTER 6
SOURCE CODE

Edit Bucket Policy:

Make sure the bucket name is correctly specified and save

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "PublicReadGetObject",
         "Effect": "Allow",
         "Principal": "*",
         "Action": "s3:GetObject",
         "Resource": "arn:aws:s3:::items-frontend-static-hosting/*"
      }
   ]
    }
```

Edit bucket CORS configuration.

```
[
   {
      "AllowedHeaders": [
         "*"
```

```
        ],
          AllowedMethods": [
          "GET"
],
        "AllowedOrigins": [
           "*"
        ],
        "ExposeHeaders": []
    }
     ]
```

HTTP API ROUTES:

- GET /items
- PUT /items
- GET /items/{id}
- DELETE(/items/{id}
- OPTIONS /items
- OPTIONS /items/{id}

CHAPTER 7
RESULT

7.1 RESULT ANALYSIS

The performance and results of a serverless website using HTTP API, DynamoDB, and Lambda can be analyzed using various metrics and tools..

1. Latency: The latency of the website can be measured using tools such as AWS CloudWatch and X-Ray. CloudWatch can monitor the API Gateway and Lambda function execution times, while X-Ray can trace the performance of individual requests and identify any bottlenecks.

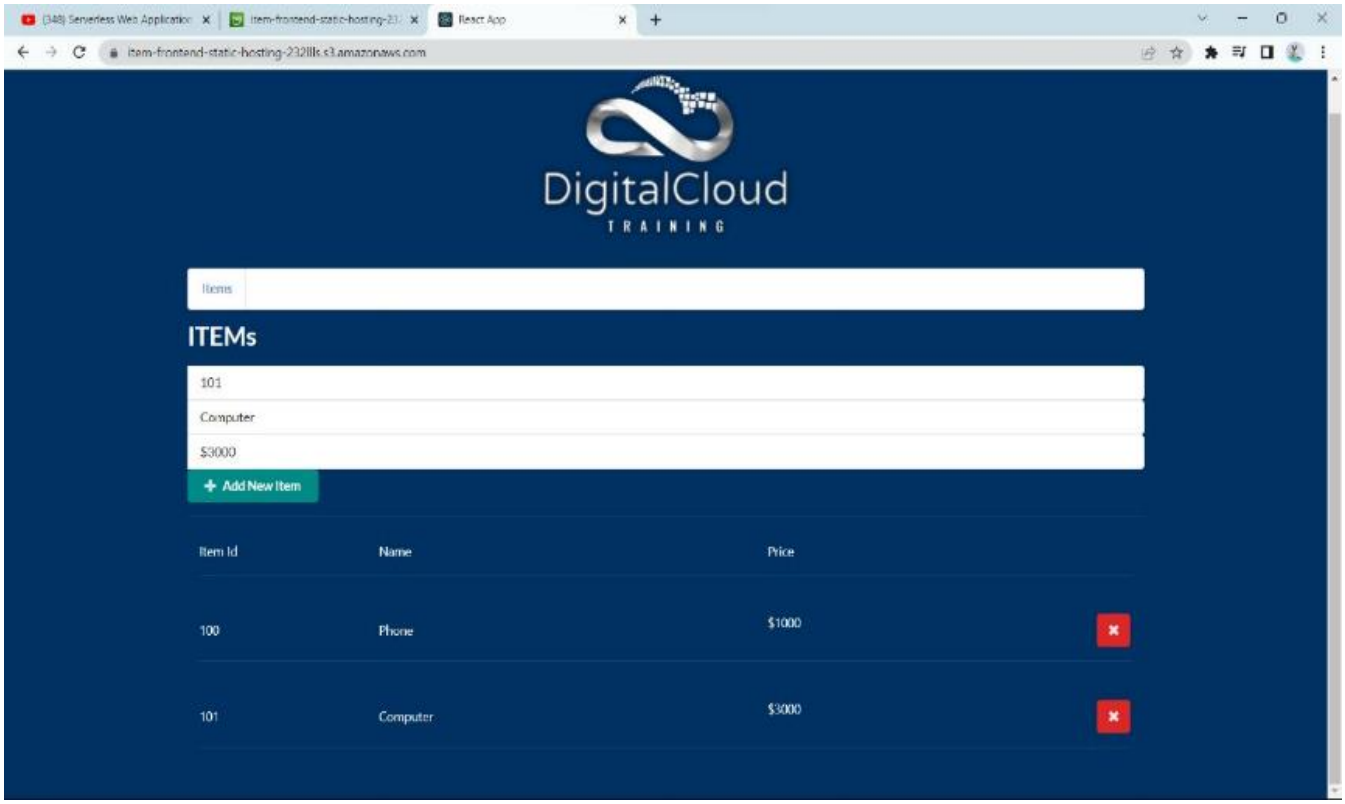
2. Cost: One of the benefits of using a serverless architecture is the cost-effectiveness. The cost of running the website can be monitored using the AWS Cost Explorer and Billing Dashboard.

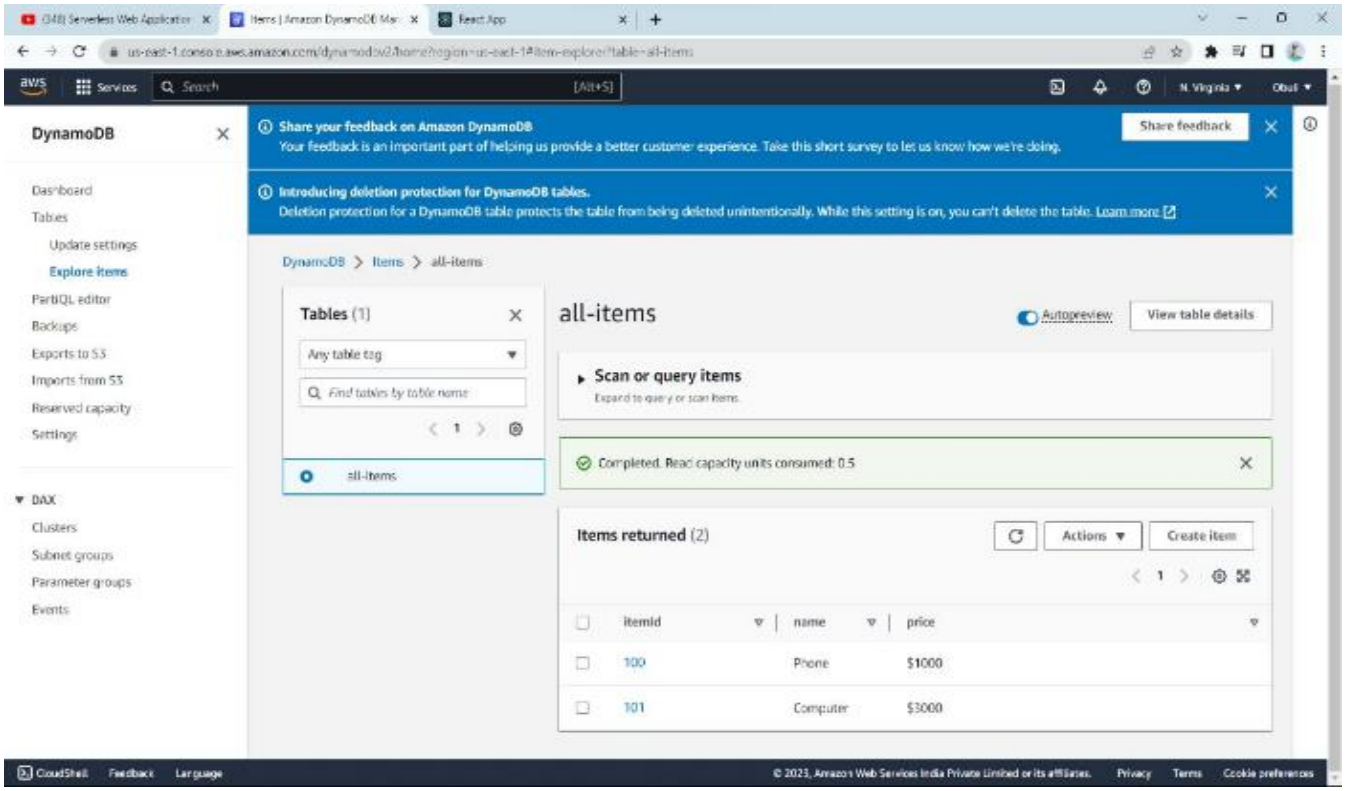
3. Scaling: The ability of the website to scale automatically in response to changing traffic can be monitored using CloudWatch and the API Gateway dashboard. The number of requests per second and concurrent executions can be tracked to ensure that the website can handle high levels of traffic.

4. Error Rates: The number of errors and error rates of the website can be monitored using CloudWatch logs and X-Ray. This can help identify and troubleshoot any issues with the website.

5. User Engagement: The engagement of users with the website can be analyzed using analytics tools such as Google Analytics or AWS Pinpoint.

7.2 OUTPUT


DigitalCloud
TRAINING
Items
ITEMs
101
Computer
$3000
+ Add New Item
Item Id
Name
Price
100
Phone
$1000
101
Computer
$3000


DynamoDB
Share your feedback on Amazon DynamoDB
Your feedback is an important part of helping us provide a better customer experience. Take this short survey to let us know how we're doing.
Share feedback
Introducing deletion protection for DynamoDB tables.
Deletion protection for a DynamoDB table protects the table from being deleted unintentionally. While this setting is on, you can't delete the table. Learn more
DynamoDB > Items > all-items
Tables (1)
all-items
Autopreview
View table details
Scan or query items
Completed. Read capacity units consumed: 0.5
Items returned (2)
Actions
Create item
itemId
name
price
100
Phone
$1000
101
Computer
$3000

CHAPTER 8
CONCLUSION

8.1 CONCLUSION

In conclusion, a serverless website using HTTP API, DynamoDB, and Lambda provides a highly scalable, reliable, and cost-effective solution for building modern web applications. With AWS managing the infrastructure, developers can focus on writing code and delivering features without worrying about hardware, networking, or server maintenance.

The architecture of the website allows for easy scaling and high availability, ensuring that the website can handle large volumes of traffic and is always accessible to users. The use of DynamoDB as a NoSQL database allows for flexible and efficient data storage, while Lambda functions provide a serverless compute environment for executing code.

To ensure optimal performance and user experience, it is recommended to monitor key metrics such as latency, cost, scaling, error rates, and user engagement. By analyzing these metrics, developers can identify areas for improvement and optimize the website for maximum efficiency.

Overall, a serverless website using HTTP API, DynamoDB, and Lambda is a powerful and flexible solution for building modern web applications that can scale to meet the needs of users and businesses alike.

8.2 FUTURE SCOPE

1. Multi-Region Deployment: By deploying the website in multiple regions, users in different parts of the world can access the website with lower latency, improving the user experience.

2. User Personalization: By integrating the website with AWS Pinpoint or a similar service, personalized content and notifications can be sent to users based on their preferences, location, and behavior.

3. API Gateway Caching: Caching responses from the API Gateway can reduce the number of requests made to Lambda and DynamoDB, improving performance and reducing costs.

4. Automated Testing: By using AWS Lambda to automate testing, the website can be more easily maintained and updated, ensuring high quality and reducing the likelihood of errors.

5. Scheduled Lambda Functions: Scheduled Lambda functions can be used to perform tasks such as database backups, data processing, and report generation, improving efficiency and reducing the workload on developers.

These are just a few simple future scopes for a serverless website using HTTP API, DynamoDB, and Lambda. By implementing these and other enhancements, the website can continue to evolve and provide a better experience for users.

REFERENCES

https://aws.amazon.com/getting-started/hands-on/build-serverless-web-app-lambda-apigateway-s3-dynamodb-cognito/

https://www.pluralsight.com/guides/building-a-serverless-web-app-on-aws-services

https://www.youtube.com/watch?v=BFC16uM15Cg

https://www.youtube.com/results?search_query=serverless+webapp+in+aws

https://chat.openai.com/

https://ijirt.org/master/publishedpaper/IJIRT151299_PAPER.pf

https://docs.aws.amazon.com/apigateway/latest/developerguide/http-api-dynamo-db.html