CLOUD SERVERLESS COMPUTING

21CS3281R/ 21CS3281A/ 21CS3281P

SKILL WORKBOOK

STUDENT ID:

STUDENT NAME:

ACADEMIC YEAR: 2023-24

Table of Contents

A.Y. 2023-24 LAB/SKILL CONTINUOUS EVALUATION

S.No	Date	Experiment Name	Pre-Lab (10M)	In-Lab (25M)			Post-Lab (10M)	Viva Voce (5M)	Total (50M)	Faculty Signature
				Program/ Procedure (5M)	Data and Results (10M)	Analysis & Inference (10M)				
1.		Introductory Session	-NA-							
2.		Create a Simple Chat Bot Serverless ApplicatinWith Amazon LEX #1								
3.		Create a AWS S3 Bucket and Configuring as a Static Website Hosting #2								
4.		Create a AWS CloudFront Distribution for Content Delivery Network to restrict s3 bucket #3								
5.		Implement Amazon API Gateway EndPoint and Writing it up to our WebSite #4								
6.		Create an AWS Lambda Function to create a test case for our Lambda Mock #5								
7.		Create A DynamoDB Table for seeding wheather table from CSV file for API. #6								
8.		Build aserverless web application for Dynamicwebsite hosting using S3 , Route53 ,CloudFront and Certificate Manager #7								
9.		Develop Serverless WebApplication on AWS using GET-METHOD getemployeedeails by email #8								

S.No	Date	Experiment Name	Pre-Lab (10M)	In-Lab (25M)			Post-Lab (10M)	Viva Voce (5M)	Total (50M)	Faculty Signature
				Program/ Procedure (5M)	Data and Results (10M)	Analysis & Inference (10M)				
10.		Develop Serverless WebApplicationon AWS using POST-METHOD postemployeedeails using email #9								
11.		Build a Serverless Application for Amazon Rekognition Service for given Image using Lambda, S3Bucket,IAM role and API Gateway. #10								
12		AZURE- Create a WEB APP in Blob Storage and upload images with Azure Functions #11								
13.		GCP- Building Serverless Front-End Applications Using Google Cloud Platform #12								
14.		Create a Simple Chat Bot Serverless ApplicatinWith Amazon LEX integrate with Telegram/Whatsapp #13 (Adv/Peer)								
15.		Develop Real time full stack serverless web application using s3,lambda, apigateway, dynamodb #14 (Adv/Peer)								
16.		Develop Serverless Web Application on AWS Translate using amplify AWS Amplify, AWS IAM, AWS Lambda, AWS API Gateway, AWS Translate #15(Adv/Peer).								
17.		Build serverless application using Kubernetes #16 (Adv/Peer)								

S.No	Date	Experiment Name	Pre-Lab (10M)	In-Lab (25M)			Post-Lab (10M)	Viva Voce (5M)	Total (50M)	Faculty Signature
				Program/ Procedure (5M)	Data and Results (10M)	Analysis & Inference (10M)				
18.		Create serverless application using Openshift container platform web console #17 (Adv/Peer)								
19.		Real time bot integration with facebook #18 (Adv/Peer)								
20.		develop serverless voting web application #19 (Adv/Peer)								
21		develop serverless web application using get and post method for real time data #20 (Adv/Peer)								

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

#1 Experiment Title: Create a Simple Chat Bot Serverless Application With Amazon LEX

Aim/Objective:

To create a serverless chatbot application using Amazon Lex to answer user queries related to a specific topic.

Description:

Creating a serverless chatbot application using Amazon Lex to answer user queries related to a specific topic is a structured process that requires careful planning and execution. To begin, it's crucial to define the purpose and scope of your chatbot. Understand the specific queries or questions it will address and establish the boundaries of its capabilities. Once you have a clear objective in mind, proceed to create an Amazon Lex bot.

Pre-Requisites:

AMAZON LEX, AWS FREE TIER ACCOUNT

Pre-Lab:

1) Is Amazon Lex a managed service?

Ans:

2) When using AWS Lex, why is it important to ensure that all fields contain valid values?

Ans:

3) List most common use cases of Amazon lex.

Ans:

4) Which service enables you to build the workflows that are required for human review of machine learning predictions?
   a) Amazon Aurora
   b) Amazon Augmented AI
   c) Amazon Lex
   d) Amazon Textract

Ans:

5) What are the different ways you can use session attributes while building bots on AWS Lex?

Ans:

In-Lab:

1. Create a Simple Chatbot in the Amazon LEX Using below procedure?

- **Procedure/Program:**

The goal of this first lab is to create a simple chatbot in the LEX console that can gather basic information from a user. which we can “eliciting slots”.

When we ask the chatbot for the weather it will not have enough 1nformat1on to provide an answer. It Will need to further establish the city your cat lives m, and then confirm with you that the question you asked was understood correctly.

For example:

You	Weatherbot
Is It too cold for my Cat?	
	What city?
chicago	
	So you want to know if your cat c.an go outside today in Chicago. is that correct?
yes	
	Fullfilment success. [i.e. end conversation]

And that's it...for now at least!

We use this chatbot later in week 4 where we will get to enhance it significantly making it much smarter. So it provides the weather for the respective city.

Once built, we will integrate It with our web application that you will build next.

For now. let's Just build that basic chatbot, and get the fundamentals out of the way.

Please follow all these steps below one by one <u>in order very carefully</u>.

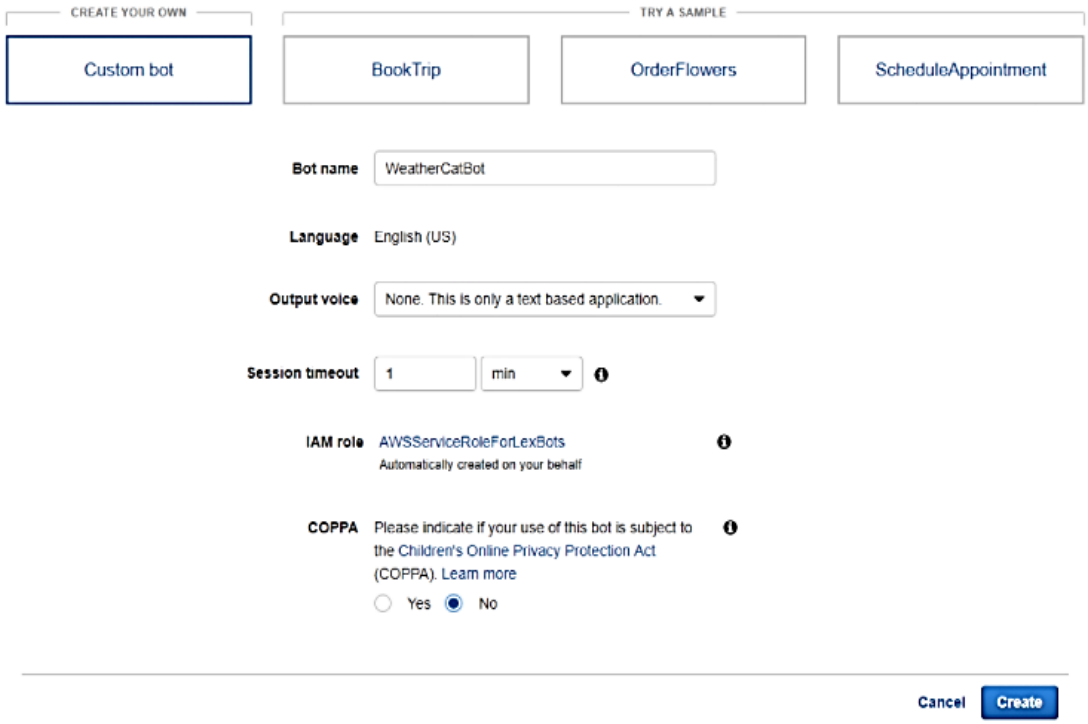
It's very easy to accidentally miss a step, and it then not work as expected.

1. Steps for creating a Lex Bot

- Sign in to the AWS Management Console and in the **Find Services** search box type lex

and choose **Amazon Lex**.

- Make sure you are in the **N. Virginia** region at the top right.
- GO TO VERSION V1 Console
- If this is your first bot, choose **Get Started**; otherwise, on the **Bots** page, choose **Create**.
- Under **Create Your Own** choose **custom Bot**.
  - Name the Bot: **WeatherCatBot**
  - For **Output voice**. Leave it as: **None**. **This is only a text based application.**
  - Change the **Session timeout** to **1** minute.
  - Leave the IAM role as **AWSServiceRoleForLexBots**
  - For **COPPA** choose **No.**



- Click **Create.**

Now we have created a bot we need to prepare it for the type of questions we might throw at it.
If we prepare LEX correctly with a good sample set of questions (that are basically saying the same thing) it can use it as "training data".

The idea is that over time, and the more it is used, it can figure out what you are trying to implly. Or what you mean when you talk to it. Pretty cool right?

Even if you ask it a question that is not in the sample questions you supply, it is usually smart enough to figure out your intent anyway.

So lets say you want to know if it is too cold for you cat, there are a gazillion edge case ways that you might phase that conversationally right?

"Hey silly bot, cool enough for my cat yet?"

"Cat bot, I bet it is too hot for· my cat in Arizona, isn't it?"

"Can my cat go outside yet? He's really bored.. oh yeah we're in Texas."

...the list goes on.

They call these common question samples "utterances".

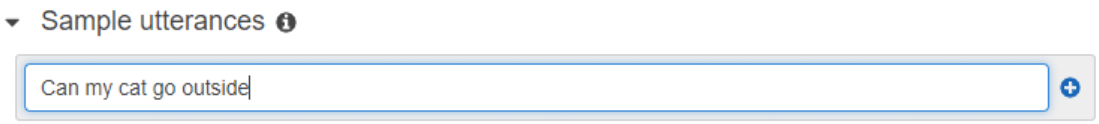
We obviously can't type every conceivable utterance, and we don't need to. LEX is pretty smart, and gets smarter over time.

We just need to provide a good sample of commonly phased questions that you think users would say. These are related to the INTENT of finding out if your cat should go out or not. And let LEX over time, figure out all the fancy humanized edge case intents.

Let's do that now.

2. Steps for creating an Intent.

- Click the **Create Intent** button.
- On the Add intent pop-up click **Create intent**.
- Name the Intent. **catweatber** and click **Add**.
- The box will disappear
- Under **Sample utterances** open it up (there will be an small black arrow on the left to expand).



NOTE: There is a + symbol at the far right of the text entry area. Either press that after adding text or press Enter before entering the next utterance.

Enter these five utterances one at a time, In order:

1. Can my cat go outside
2. Is it warm enough for my cat
3. Can I let my cat out in {city_str}
4. Should my cat wear booties in {city_str}
5. Will my cat stay dry in {city_str}

We are *not* going to talk about synonyms in the course, but I recommend you check chat concept out to make your bot super human. https://aws.amazon.com/blogs/mac'line-learning/use-synonym s-and-slot-value-validation-in-your-amazon-lex-chatbots/

Did you notice the {city_str}? Think of this as a variable, as the user can say any city they want.

This maps to what we call a **SLOT**.

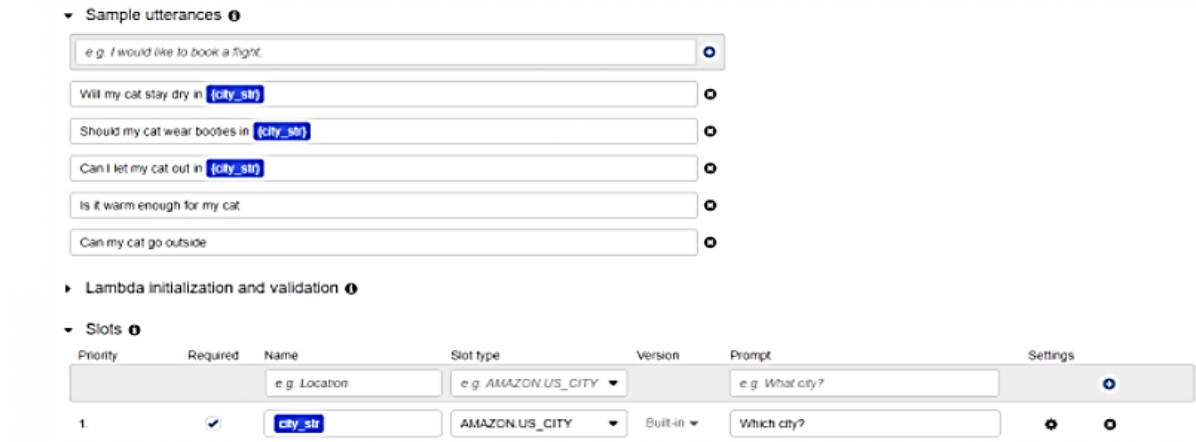
The advantage in putting this in our utterances, is that if a slot is used in the initial user utterance the LEX bot has all it needs. It no longer needs to elicit that slot and ask the user what city, because it has just been volunteered.

Are you ready to create the slot that we used as a placeholder as city_str in some of our utterances? Yeah, you're ready ;)

3. Steps for creating a Slot.

- Scroll down to the slots sections and expand it using the drop down arrow.

- Type city_str under **Name**.
- For **Slot type** start typing in **AMAZON.u** and choose **AMAZON.US_CITY** .
- Type which city? under **Prompt** and click '+' or hit **Enter**.
- {city_str} will now be colorized in the **Sample utterances** drop down.



Before you save your intent. we want to humanize it a little more by configuring a confirmation step with "So you want to know if your cat can go out today in {city_str}?".

Let's add that now.

- Expand **Confirmation prompt**.
  - o Check **Confirmation prompt**.
  - o Under **Confirm** type in the following:

So you want to know if your cat can go out today in {city_str}?

- Under Cancel (if the user says "no", ie. that's wrong) type in:

Sorry,can you please repeat your initial question?

- Expand **Fulfillment** and verify **Return parameters to client** is selected.
- Leave Response as it is.
- Click **Save Intent** at the bottom.
- Click **Build** at the top right.
- At the Build your bot pop-up click **Build**.
- This can take a few minutes. *Usually less than 30 seconds, f your lucky.*

Believe it or not. that's it…

Lets test it in the helpful Lex console.

4. Steps for testing the bot in the Lex console.

- Expand the chatbot on the right. Where it says Test Chatbot .
- Try these conversations, so you can see how LEX responds to you in different scenarios.

user	chatbot
Will my cat stay dry in seattle?	<editor note> likely not 😊
	So you want to know if your cat can go out today in seattle?
Yes	
	Intent CatWeater is ReadyForFullfilment: city_str:Seattle
Can I let my cat out in Arizona?	
	So you want to know if your cat can go out in Arizona?
Is it warm enough for my cat?	
	Which city?
Denver	
	So you want to know if your cat can go out in denver?
Sure	
	Intent CatWeater is ReadyForFullfilment: city_str:Denver
Is it too cold for my cat	
	Which city?
Las vegas	
	So you want to know if your cat can go out in las vegas?
Yes	
	Intent CatWeater is ReadyForFullfilment: city_str:las vegas

NOTE: Notice we didn't include "Is it w cold for my cat?" in the utterances. Yet our chat bot is intelligent enough to prompt us for the city.

I think that is so cool. LEX can actually figure out what you are trying to ask.

Awesome job!

Let's look at our course todo checklist.

Congratulations

Refer:

1. https://www.coursera.org/learn/aws-fundamentals-building-serverless-applications/supplement/rn0YB/exercise-1-1-creating-a-simple-bot-with-lex

- **Data and Results:(keep screenshot of output)**

- **Analysis and Inferences:(write your own observations)**

Sample VIVA-VOCE Questions (In-Lab):

1. What is an "intent"?
   a) The response from Amazon Lex back to the user.
   b) An Amazon Lex feature that does natural langauge processing.
   c) A particular goal that the user wants to achieve.

Ans:

2. What is a slot?
   a) Data that the user must provide to fulfill the intent.
   b) The response from Amazon Lex back to the user.

Ans:

3.What service does Amazon Lex use for text-to-speech?
   a) Amazon Alexa
   b) Amazon Poly
   c) Amazon Rekognition
   d) Amazon Alexa

Ans:

Post-Lab:

1. **Create a Banking Bot/Order Pizza with support for English and Spanish on the Amazon Lex V2 console**

- Procedure/Program:

- **Data and Results:**

- **Analysis and Inferences:**

Evaluator Remark (if Any):	**Marks Secured: _____out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.

#2. Experiment Title: Creating a S3 Bucket and Configuring as a Static Website Hosting

Aim/Objective: building static database driven web application to figure out the temperature using S3bucket

Description:

What I'd like us to focus on next is the building of a database driven web application that figures out the temperature when we tell it a particular city. As you would Imagine. creating an application like this involves many moving parts. We will need:

- a front end web site
- a backend API
- some sort of logic (functional intelligence)
- a data source.

If we take it stage by stage and first focus on just the web front end, and not concern ourselves yet with wiring It up to a backend. We only need to worry about hosting some HTML.CSS and little bit of JavaScript

We already have a website from end all prepared for you in a zip tile. I did promise you no coding remember:)

All you need to do 1s download this zip file, unzip it and push it into the cloud. We will use 53 (Simple Storage Service) which can happily "host" these kinds of static websites.

Later on, after we have this website uploaded and everything is configured *for* website hosting. We can gee a little fancy and add a content delivery network 1n front of it. Ready for global domination:). Adam will talk to you about that in one of the upcoming videos. Al things in good time though, let's get this static website up and running on 53 first.

You'll need to create what we call a "bucket" to store your web objects (HTML. CSS, and JS files).and set it up correctly using a bucket policy. so it can act as a host for out static website. It's not difficult to do this. just follow these steps carefully, and your site will be up in no time.

Pre-Requisites:

AWS FREE TIER ACCOUNT, S3 BUCKET CREATION, BUCKET VERSION

Pre-Lab:

1) What is key in AWS S3?

Ans:

2) Is Simple Storage Service considered as DFS?

Ans:

3) What is static web hosting in S3?

Ans:

4) Object storage systems require less _____ than file systems to store and access files.
   a) Big data
   b) Metadata
   c) Master data
   d) Exif data

Ans:

5) How many buckets can you create in AWS by default?

   a) 100 buckets
   b) 200 buckets
   c) 110buckets
   d) 125 buckets

Ans:

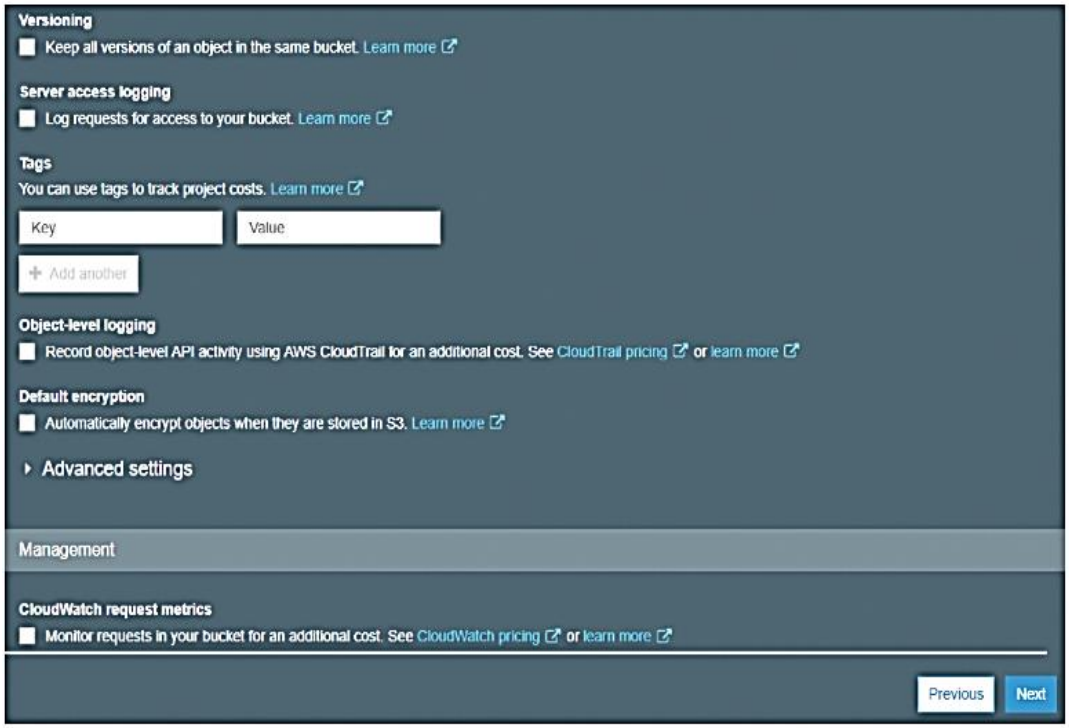
In-Lab:

Creating a S3 Bucket and Configuring as a Static Website

- **Procedure/Program:**

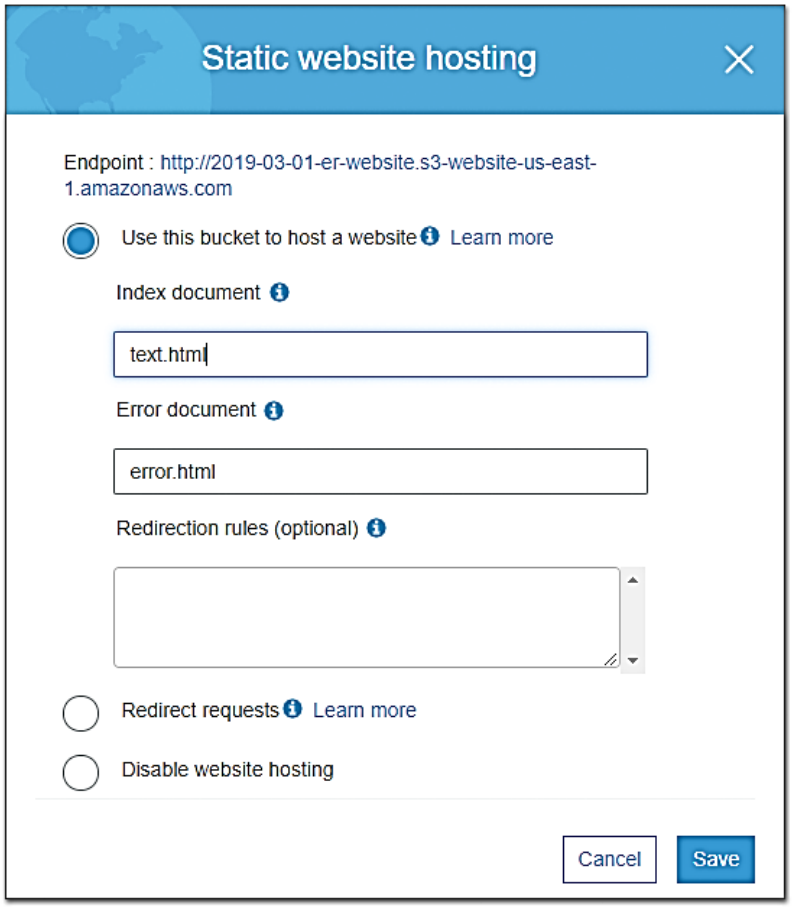
1. **Steps to create a S3 bucket and configure it as an static website.**

- Sign in to the AWS Management Console and in the **Find Services** search box type s3 and choose **S3**.
- Click Create bucket.
  - In the **Bucket name** field, type a unique DNS-compliant name for your new bucket.
  - Example: **2019-03-01-er-website** *IMPORTANT: Useyour own initials and the current date.*
  - For **Region** choose US East (N.Virginia).
- Click **Next**. Leave the current settings alone:



- Click **Next** again.
- Under **Public access settings for this bucket** de-select the following:
  - **Block new public ACLs and uploading public objects.**
  - **Remove public access granted through public ACLs.**
  - **o Block new public bucket policies.**
  - **Block public and cross-account access If bucket has public policies.**

- Click **Next.**
- Click **Create bucket.**
- Click on the bucket and select the **Properties** pane, choose **Static Website Hosting**.
  - Select **Use this bucket to host a website**.
  - Under **Index document** type in **text.html** . *NOT index.html*
  - Under **Error document** type in **error .html**.



- Click the endpoint. It should open in a new tab and show an error 403 forbidden. Which is exactly what we would expect right now, as no permissions have been added yet.

403 Forbidden

- Code: AccessDenied
- Message: Access Denied
- RequestId: 9D930D9176FB6D3E
- HostId: xPjHzQf8gIq1jB9yJ4b5YpgFCsYx0kvTZPcqShr4QbD1889nn95kc6FLVPpGI665fpB+LTNX4bI=

An Error Occurred While Attempting to Retrieve a Custom Error Document

- Code: AccessDenied
- Message: Access Denied

- Click **Save**. So we can move on to adding public permissions to this bucket.

- Select the **Permissions** tab and choose **Bucket Policy**.
- Paste the following code below in the Bucket policy editor. Make sure to change the Resource arn with your bucket name.
- You must change arn:aws :s3:::20 19-03-01-er-website/* to your bucket. e.g arn:aws :s3:::2019-mm-dd-xx-website/ *

```
{
"Version": "2012-10-17",
"Statement": [
        {
        "Sid": "AddPerm",
        "Effect": "Allow",
        "Principal": "*",
        "Action": [
                "s3:GetObject"
        ],
        "Resource": [
                "arn:aws:s3:::2019-03-01-er-website/*"
        ]
        }
]
}
```

- Click **Save.**
- This bucket will now have public access as it will host our web objects.

Try it now. Go to that tab where your website is hosted (where we had the 403 forbidden message), and press refresh.

You will notice you can now see absolutely nothing. i.e you will get a 404 error.

This is because we have nothing in out bucket to show, so the browser is telling us, "nope can't find anything here". At least this proves that the bucket is now public and we did everything right so far
#winning.

Now let's get the website up there.

2. Steps to upload objects to an existing S3 bucket.

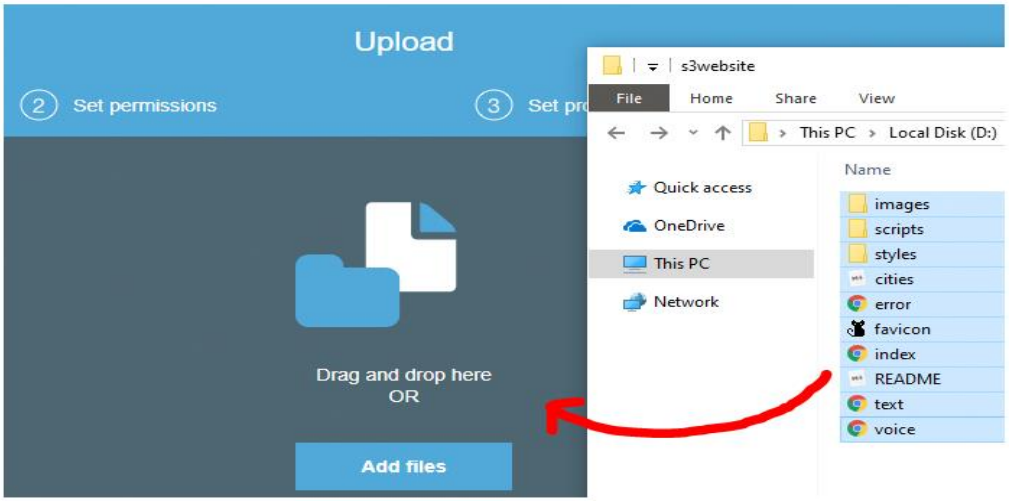
- Download the ZIP file at:

https://s3.amazonaws.com/awsu-hosting/CSA-TF-100-SCSRVL-10-EN/s3website.zip

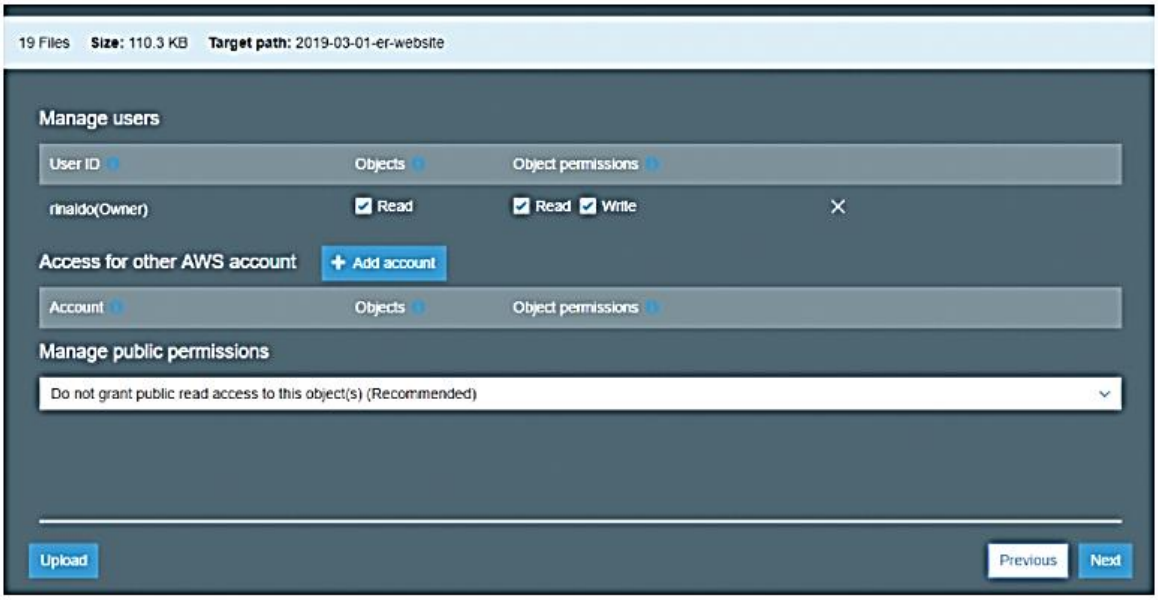
- Unzip it on your local machine, and you will see the following folder structure:

+ images
  - mobile.jpg
+ scripts
    - config.js
  - helper.js
  - jquery.js
  - text.js
  - voice.js
+ styles
  - reset.css
  - text.css
  - voice.css
+ cities.md
+ error.html
+ favicon.ico
+ index.html
+ README.md
+ text.html
+ voice.html

- Click the **Overview** tab and click on your bucket **2019-mm-dd-xx-website .** (Keep in mind your bucket will have its own unique name that you gave it).
- Click **Upload** and drag the files needed for the site from your local file explorer.
- TIP: You can press "add files" button, but we recommend dragging (like in this image below)



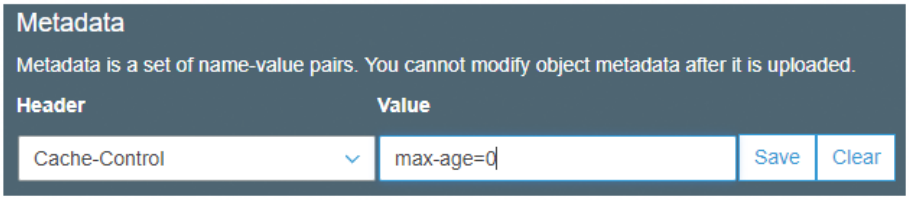
- Click **Next.**
- Leave the permissions settings alone as the defaults.



- Click **Next.**
- Scroll down to **Metadata.**

This next step about max age is **critical**! If you miss this, you will have problems later on in the future exercises. So type carefully and do not miss this step.

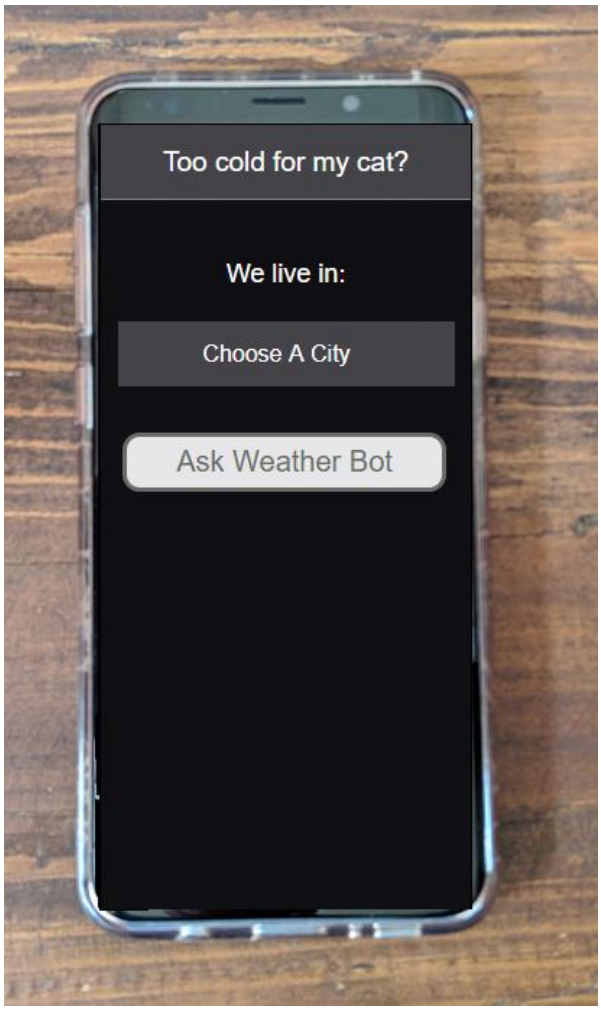
- For **Header** choose **cache-control** and for the **Value** enter **max-age=O** and click **Save**.



- Click **Next**.
- **Double check it ;)**
- Click **Upload.**
- Verify connectivity by browsing to your tab (website} where you had the 404 error.

You should see your shiny new text based static web based application (see image below).

PLEASE USE OR SWITCHTO CHROME for this, as later on we use **chrome only** features for the application.



- Press the section that says "Choose a city" it will give you a choice of cities, then press **Ask Weather Bot.**

As of right now, every city you choose will return **20 degrees.** This is because this is just a static website. Fear not we will add backend intelligence to this later on.

3. Steps to make local changes and upload your adjustments.

We want to make sure that when we make changes to local files that they are propagated correctly. So please make the following change to your local copy of text .html

From

```
<h1>Too cold for my cat?</h1>
```

To

```
<h1>Too hot for my cat?</h1>
```

- Upload the edited **text .html** to the S3 bucket:
  - Click **Upload**.
  - Click **Add files**.
  - Browse to the location of **text.html** on your local machine or drag and drop like before.
  - o Click **Next**. Leave the permissions alone and click **Next**.
  - Again scroll down to **Metadata** and set the **Header** to cache-control and set the **Value** to max-age=O.
  - **DOUBLE CHECK IT.**
  - Click **Save** and click **Next.**
  - Finally click **Upload**.
- Refresh your website

You should now see the changes. Awesome !
That's another one off our goa l checklists done, let's see our progress.

- **Data and Results:**

- **Analysis and Inferences:**

Sample VIVA-VOCE Questions (In-Lab):

1. What is Amazon S3 (Simple Storage Service), and how is it typically used?

Ans:

2. Can you explain the concept of a static website? How does it differ from a dynamic website?

Ans:

3. What is the purpose of creating an S3 bucket as a static website?

Ans:

4. What are the steps involved in creating an S3 bucket in AWS.

Ans:

5. How do you configure an S3 bucket to function as a static website?

Ans:

Post-Lab:

T1.Protect Data on Amazon S3 Against Accidental Deletion or Application Bugs Using S3 Versioning, S3 Object Lock, and S3 Replication

- **Procedure/Program:**

- **Data and Results:**

- **Analysis and Inferences:**

Evaluator Remark (if Any):	**Marks Secured: _____out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.

#3. Experiment Title: Create an AWS CloudFront Distribution for Content Delivery Network to restrict s3 bucket.

Aim/Objective:

The aim of this task is to create an AWS CloudFront distribution that acts as a Content Delivery Network (CDN) to serve and protect the content stored in an S3 bucket.

Description:

Creating an AWS CloudFront distribution to serve as a Content Delivery Network (CDN) for an S3 bucket is a strategic move to optimize content delivery and enhance the security of the data stored within that bucket. AWS CloudFront, as a globally distributed CDN, accelerates the delivery of web content, including images, videos, and other assets, by caching content at edge locations close to the end-users. To ensure security, access to the content in the S3 bucket is restricted, and this is where AWS Identity and Access Management (IAM) plays a crucial role.

Pre-Requisites:

AWS FREE TIER ACCOUNT, CLOUDFRONT,S3

Pre-Lab:

1)How does CDN or AWS Cloudfront work?

Ans:

2)What are the use cases of using CloudFront Functions?

Ans:

3)You support a web application that uses a CloudFront distribution. A banner ad that was posted the previous night at midnight has an error in it, and you've been tasked with removing the ad so that users don't see the error. What steps should you take? (Choose two)

A. Delete the banner image from S3.

B. Remove the ad from the website.

C. Wait for 24 hours and the edge locations will automatically expire the ad from their caches.

D. Clear the cached object manually.

4)You are building a large-scale confidential documentation web server on AWS and all of its documentation will be stored on S3. One of the requirements is that it should not be publicly accessible from S3 directly, and CloudFront would be needed to accomplish this. Which of the methods listed below would satisfy the outlined requirements? Choose an answer from the options below.

A. Create an Identity and Access Management (IAM) user for CloudFront and grant access tothe objects in your S3 bucket to that IAM User.

B. Create an Origin Access Identity (OAI) for CloudFront and grant access to the objectsin your S3 bucket to that OAI.

C. Create individual policies for each bucket the documents are stored in, and grant access only to CloudFront in these policies.

D. Create an S3 bucket policy that lists the CloudFront distribution ID as the Principaland the target bucket as the Amazon Resource Name (ARN).

Ans:

In-Lab:

Create a AWS CloudFront Distribution for Content Delivery Network to restrict s3 bucket

- **Procedure/Program:**

We already have a working static website. How can we make it perform better?

As that was last week, bring up the website in Chrome and remind yourself of what we have done.

We would now like to put a content delivery network (CloudFront} in-front of our static website host (53) which we call the "origin".

Doing this will give our users around the world a lower latency, as Adam talked about. It will also alleviate the load that is placed on the origin bucket.

Once we configure the next few steps for setting up the content delivery network (Amazon CloudFront) it usually takes around 10 to 15 minutes to fully propagate. During that time we will go back to the 53

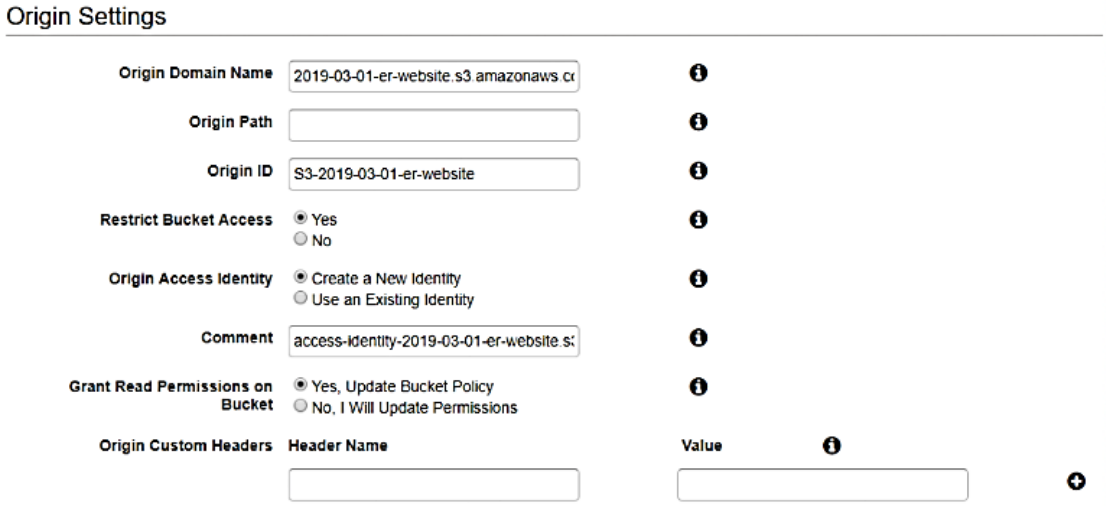
bucket Origin and prevent people from accessing content directly from 53. It will literally force the users

to go via CloudFront, which will be much better for them. Let's start with configuring CloudFront for our website.

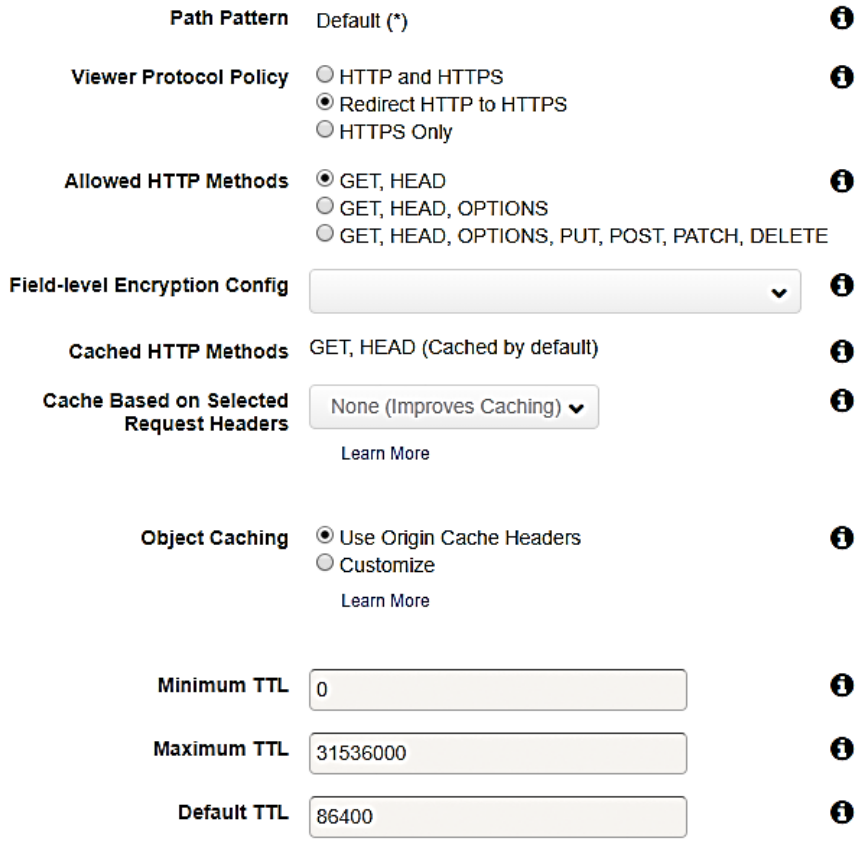
1. **Steps for creating a Cloud Front distribution**

- Sign in to the AWS Management Console and in the **Find Services** search box type cloud and choose **Cloudfront**.
- You should **Global** for the region at the top right.
- Click **Create Distribution.**
- Under **Web** click **Get Started.**

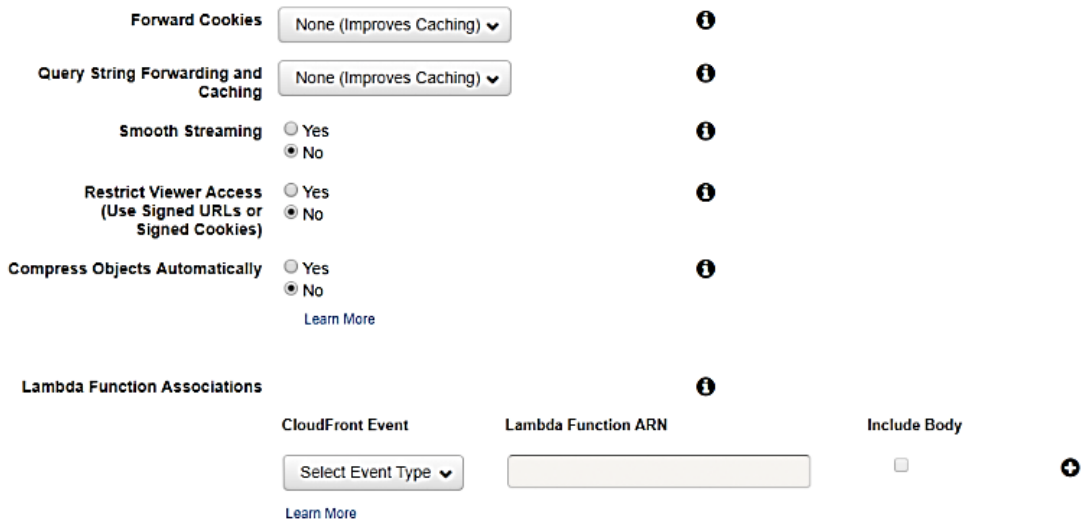
- For **Origin Domain Name** once you place the cursor in there you should see your available S3 buckets.
- Pick the website bucket you created.
- If it's not listed type it in:e.g 2019-03-01-er-website. s3.amazonaws .com *Usingyour bucket name*
- Leave **Origin Path** blank.
- The **Origin ID** should have been pre-populated when you chose your bucket.
- Click **Yes** to **Restrict Bucket Access**.
- Under **Origin Access- Legacy access Identity** select **Create a New Identity**.
- It will pre-populate the **Comment** and append the bucket name.
- For **Grant Read Permissions on Bucket** check **Yes, Update Bucket Policy**. This will update the bucket policy for us.
- Leave the **Origin Custom Headers** blank.



- For the **Default Cache Behavior Settings** section:
- Under **Viewer Protocol Policy** select **Redirect HTTP to HTTPS**.
- For **Allowed HTTP Methods** choose **GET,HEAD.**
- Leave **Field-level Encryption Config** blank.
- Leave **GET,HEAD (Cached by default)** for **Cached HTTP Methods.**
- For **Cache Based on Selected Request Headers** leave it as the default **None (Improves Caching)**.
- For **Object Caching** also leave it at the default **Use Origin Cache Headers.**

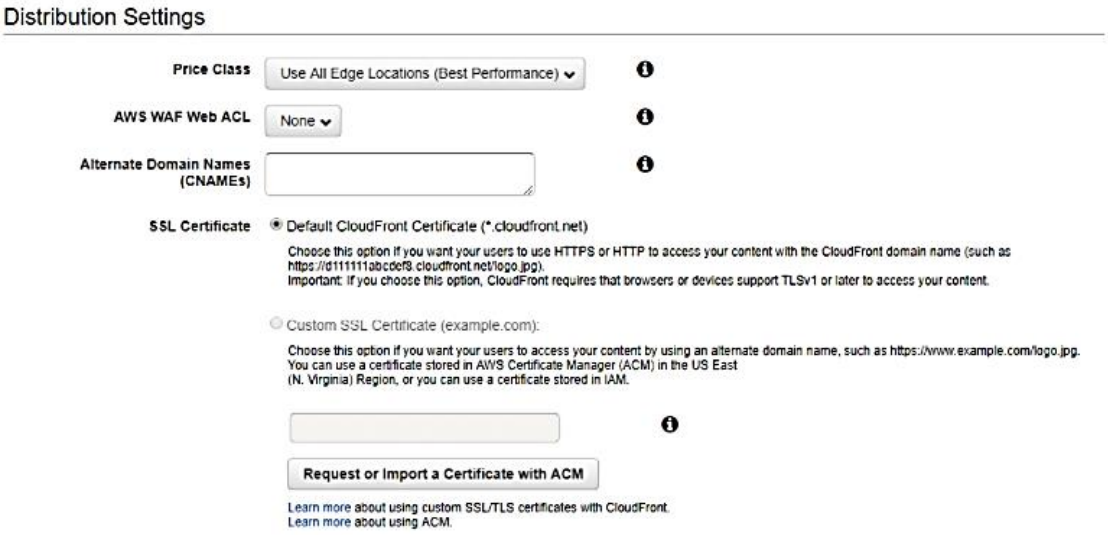


- Under **Forward Cookies** leave it as **None (Improves Caching).**
- Also for **Query String Forwarding and Caching** leave as **None (Improves Caching).**
- For **Smoothing Streaming** select **No.**
- For **Restrict Viewer Access (Use Signed URLs or Signed Cookies)** select **No.**
- Also leave **Compress Objects Automatically** as **No.**
- We can also leave **Lambda Function Associations** as the default.

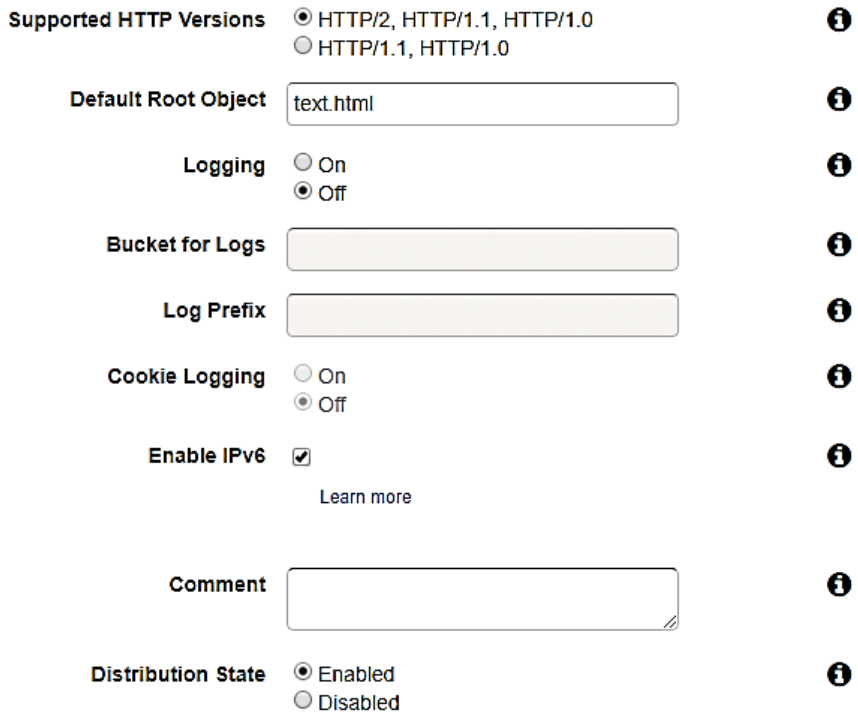


- Scroll down to **Distribution Settings.**
- For **Price Class** leave the default **Use All Edge Locations (Best Performance).**

- We will not be using WAF so for **AWS WAF Web ACL** leave it as **None.**
- Also leave **Alternate Domain Names (CNAMEs)** blank.
- We will also use the **Default CloudFront Certificate** for **SSL Certificate.**



- For **Supported HTTP Versions** leave as **HTTP/2,HTTP/1.1,HTTP/1.0.**
- Under **Default Root Object** type in **text .html .**
- We can leave **Logging** set to **Off.**
- Leave **Enable 1Pv6** checked.
- Finally set **Distribution State** to **Enabled.**



- Click **Create Distribution.**

- Click on **Distributions** at the top left to see your CloudFront distribution being built.
- This can take 15-20 minutes to complete.

While we wait. we ill head over to S3 and lock down access to only allow calls from CloudFront.

2. Restrict our S3 bucket policy to CloudFront

- Click **Services** at the top left and type in S3 or select it from History.
- Click your bucket **2019-mm-dd-xx -website** . IMPORTANT: Your bucket will have a different name.
- Click **Permissions.**
- Select **Bucket Policy.**
- We can see that CloudFront has added what we call an "Origin Access Identity" to the policy.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AddPerm",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::2019-03-01-er-website/*"
    },
    {
      "Sid": "2",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::cloudfront:user/CloudFront Origin Access Identity E1KO2GAPI
WFF7X"
      },
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::2019-03-01-er-website/*"
    }
  ]
}
```

- Remove the public S3 access section so it looks more like the following:

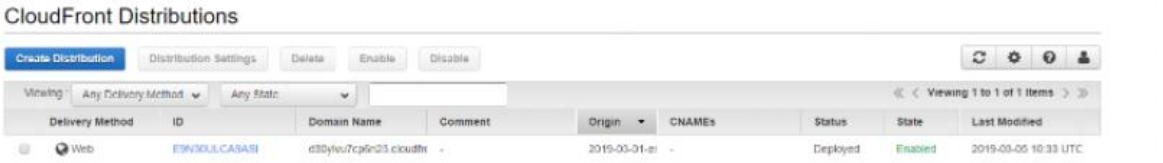
```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "2",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::cloudfront:user/CloudFront Origin Access Identity E1KO2GAPI
WFF7X"
```

```
        },
        "Action": "s3:GetObject",
        "Resource": "arn:aws:s3:::2019-03-01-er-website/*"
      }
    ]
}
```

- This will only allow our specific CloudFront distribution access to our S3 bucket which is what we want.
- Click **Save** and grab a cup of coffee while we wait for the CloudFront Distribution to finish baking.

3. **Steps for testing that we successfully locked down S3 from public view**
   - Browse to **your** S3 endpoint: *Example: http://2019-03-01-er-website.s3-website-us-east-1.amazonows.com/*
   - You will see a **403 Forbidden** as we effectively removed public access via the bucket policy.

- Click on the cloudFront distribution ID.(The blue hyperlink)



- Copy the URL under **Domain Name.**
- Browse to that URL and you should now see the **text.html** page.

Remember the distribution may take up to 15 minutes to complete. Next we will wire up our static website to a backend API. Awesome, we are moving though our exercise goal list nicely.

- **Data and Results:**

- **Analysis and Inferences:**

Sample VIVA-VOCE Questions (In-Lab):

1. What is AWS CloudFront, and how does it work as a content delivery network (CDN)?

Ans:

2. Why might you want to use AWS CloudFront in conjunction with an S3 bucket?

Ans:

3. How does AWS CloudFront help improve the performance and security of content delivery?

Ans:

4. What are the key components of an AWS CloudFront distribution?

Ans:

5. How can you configure CloudFront to restrict access to an S3 bucket?

Ans:

Post-Lab:

Deliver Content Faster with Amazon CloudFront

Refer link: https://aws.amazon.com/getting-started/hands-on/deliver-content-faster/?ref=gsrchandson

- **Procedure/Program:**

- **Data and Results:**

- **Analysis and Inferences:**

Evaluator Remark (if Any):	**Marks Secured: _____out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.

#4.Experiment Title: Implement Amazon API Gateway EndPoint and Writing it up to our Website

Aim/Objective:

The aim of this task is to implement an Amazon API Gateway endpoint and integrate it into our website.

Description:

We have a website sitting on S3 being exclusively delivered via CloudFront, but the problem is that the website is static. This means when you are interacting with it is "faking it". It Is hard-coded In the javascript to replay with "the temperature in X is 20 degrees".

We need to have our website's JavaScript hit an endpoint in the cloud and return data from that. Perhaps different temperatures say,69 degrees. That way when we test it we will know it is hitting our API successfully. Creating a backend API is easy using Amazon API gateway. Before we create an API that does anything fancy, such as look up weather data from a database or run function, we want to MOCK our API.

This is a fancy way of saying that will make a request to the API and get a temperature of 60 degrees back regardless of what information we pass to the back end. l.e A hard coded mock API. Once the mock is wired up and we have tested our website, we can start thinking about putting functionality behind it. We will look at server-less functions next week with John, for now, let's get this mock API built and tested.

Pre-Requisites:

AMAZON FREE TIER ACCOUNT, API GATEWAY, CLOUDFRONT, S3 BUCKET

Pre-Lab:

1) Describe are the Features of API Gateway

Ans:

2) What is Mapping Template?

Ans:

3) What is API Caching in API Gateway?

Ans:

4) A company owns an API which currently gets 1000 requests per second. The company wants to host this in a cost effective manner using AWS. Which one of the following solution is best suited for this?
   A. Use API Gateway with the backend services as it is.
   B. Use the API Gateway along with AWS Lambda
   C. Use CloudFront along with the API backend service as it is.
   D. Use ElastiCache along with the API backend service as it is.

Ans:

5) A company runs an online voting system for a weekly live television program. During broadcasts,users submit hundreds of thousands of votes within minutes to a front-end fleet of Amazon EC2 instances that run in an Auto Scaling group. The EC2 instances write the votes to an Amazon RDS database. However, the database is unable to keep up with the requests that come from the EC2 instances. A solutions architect must design a solution that processes the votes in the most efficient manner and without downtime. Which solution meets these requirements?

A) Migrate the front-end application to AWS Lambda. Use Amazon API Gateway to route user requests to the Lambda functions.

B) Scale the database horizontally by converting it to a Multi-AZ deployment. Configure the front-end application to write to both the primary and secondary DB instances.

C) Configure the front-end application to send votes to an Amazon Simple Queue Service (Amazon SQS) queue. Provision worker instances to read the SQS queue and write the vote information to the database.

D) Use Amazon EventBridge (Amazon CloudWatch Events) to create a scheduled event to re-provision the database with larger, memory optimized instances during voting periods. When voting ends, re-provision the database to use smaller instances

Ans:

In-Lab:

Amazon API Gateway EndPoint and Writing it up to our WebSite

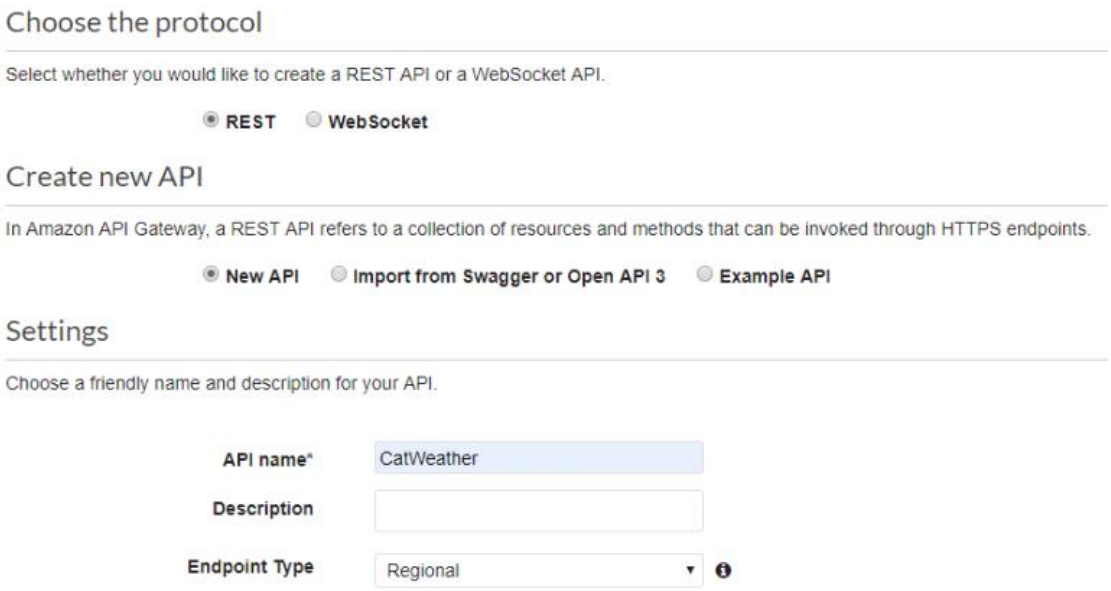
- **Procedure/Program:**

1. Steps for creating a REST endpoint with API Gateway

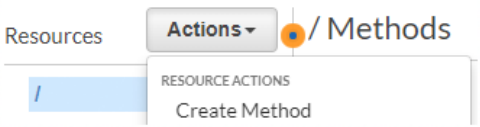
- Sign in to the AWS Management Console and in the **Find Services** search box type api and choose **API Gateway**.
- Make sure you are in the **N. Virginia** region at the top right.
- Click **Get Started** and click **OK** to remove the Create Example API pop-up.
  - If you already have APIs click **Create API**
- Choose protocol **REST.**
- Under Create new API choose **New API.**

For settings use the following:

- API name: **CatWeather.**
- Optionally, add a brief description in **Description.**
- Endpoint Type: **Regional.**



- Click **Create APL**
- Click **Actions** and **Create Method.**



- Choose **POST** from the list under the */* resource and click the **check mark icon.**

- For **Integration type** choose **Mock** and click **Save.**
- Click **TEST.**

- Under **Request Body** enter in:

```
{
    "city_str": "VEGAS"
}
```

- Click **Test** and you should get back something similar in the logs section:

Under Response Body: no data

And under Response Headers:

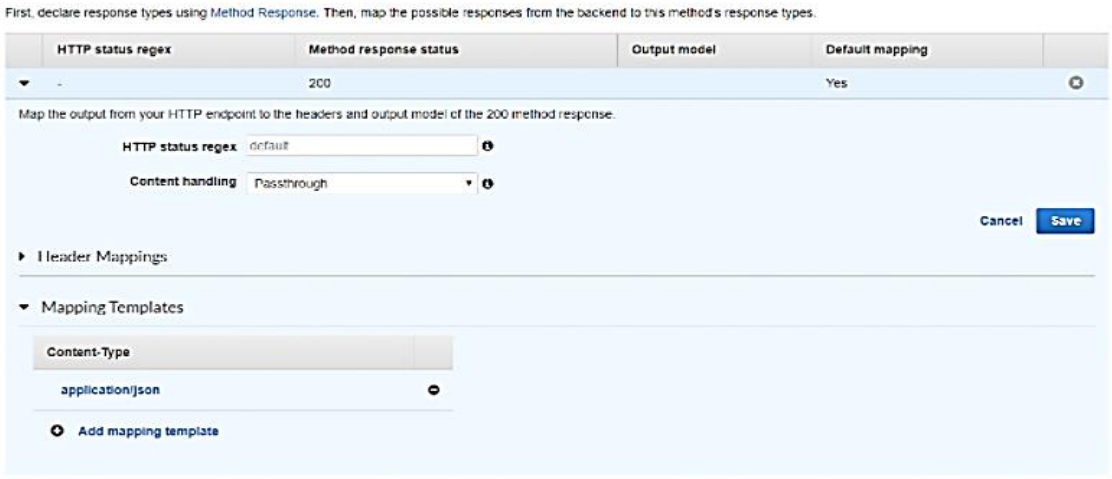
```
{"Content-Type":"application/json"}
```

Logs should look similar to:

HTTP Method: POST, Resource Path: /

```
Method request path: {}
Method request query string: {}
Method request headers: {}
Method request body before transformations: {
  "city_str": "VEGAS"
}
Method response body after transformations:
Method response headers: {Content-Type=application/json}
Successfully completed execution
Method completed with status: 200
```

- Scroll up and click **Method Execution** '←Method Execution' to go back to the test settings.
- Click **Integration Response**.
- Click the arrow to open up the response properties.
- Click the drop down arrow for **Mapping Templates.**

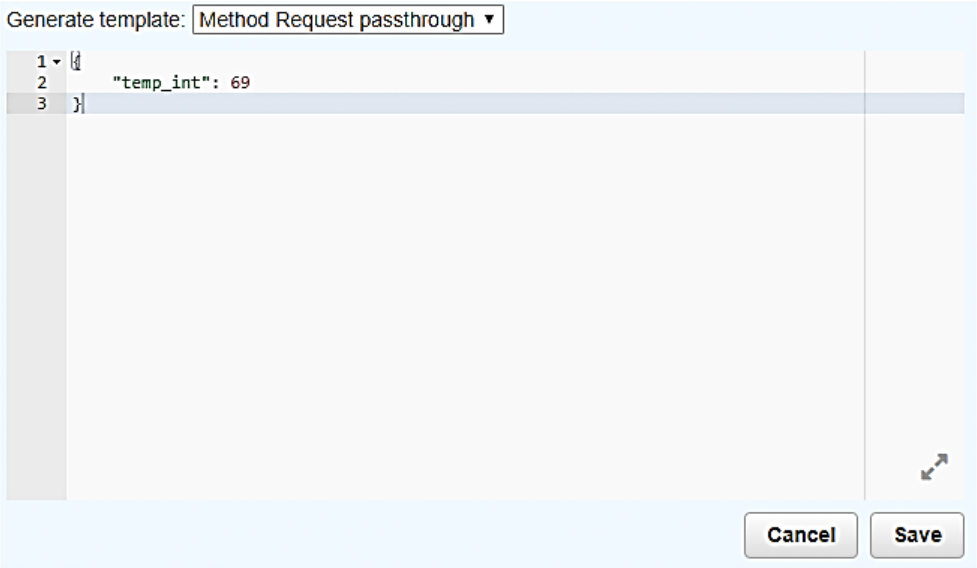


- Click on **application/json** blue hyperlink under **Content-Type**.

- Under **General template** choose **Method Request passthrough**.
- Replace it all with:

```
{

   "temp_int": 69
}
```

- Click **Save** at the bottom right.



- **Also** click **Save** at the top right.
- Choose **Method Execution** again.
- Click **Test.**
- Set the request body as

```
{

  "city_str": "CHICAGO"
}
```

- Click **Test** again**.**
- You should see the following

Under the Response Body:

```
{

   "temp_int": 69
}
```

Response Header:

```
{"Content-Type":"application/json"}
```

Logs should show something similar to:

```
Execution log for request bc5f8d95-3f6b-11e9-9e4c-a7f1e746c2c6

Starting execution for request: bc5f8d95-3f6b-11e9-9e4c-a7f1e746c2c6
HTTP Method: POST, Resource Path: /
Method request path: {}
Method request query string: {}
Method request headers: {}
Method request body before transformations:  {
```

```
    "city_str": "CHICAGO"
}
Method response body after transformations: {
 "temp_int": 69
}
Method response headers: {Content-Type=application/json}
Successfully completed execution
Method completed with status: 200
```


Awesome! We have a working MOCK API that returns hard

coded data. Now we need to wire this up to our website, but

there's a problem.

The problem is that our website is being hosted via a CloudFront domain name, and this API (once we publish it) will be available at a different API Gateway domain names, and the browser will bark at you if you try to go across domains.

Basically, if you try to bring in content from a different domain the browser needs to be sure that it is allowed to be pulled into that (cross domain) website.

The way we let the browser know that it is ok for a website of a different domain (CF) to get information from us our API Gateway endpoint is to set up what we call CORS (Cross Origin Resource Sharing).

Luck for us, this is very easy to do via the API Gateway console. For simplicity we will say any website (using the asterix *) that request information from our endpoint will be allowed to consume our content.

2. **Steps to enable CORS in the API gateway console.**

- Click on the resource / (NOT Post) ... /
- Select **Actions** and **Enable CORS.**
- Select **DEFAULT 4XX and DEFAULT SXX.** Leave the other settings as the default.

- Click **Enable CORS and replace existing CORS headers.**
- Click **Yes, replace existing values** on the **Confirm method changes** pop-up.

- Click **Actions** and choose **Deploy API** under API Actions.

- On the **Deploy API** settings pop-up select:
  - Deployment stage **[New Stage)**
  - Stage name test
  - Stage description test
  - Click **Deploy.**
- Copy down the **Invoke URL.** eg .https://j5c1lgd.Jf6.execute·aP-1.us-east-1.amazonaws.com/test
- This is a post request so IF you simply visit that URL in the browser you will get this error:

```
{
  "message": "Missing Authentication Token"
}
```

Now our mock API is tested, CORS is enabled and fully deployed. We need to have our website call that endpoint to get weather data (always 69 degrees as it is a MOCK remember).

3. **Steps to wire up your mock API to your website**

You don't have to write any code for this next bit. All you need to do is change the API_GATEWAY_lIRL_STR value in the website's /scripts/config.js file. Save it. and re-upload it back to 53, CloudFront will pick up the changes to 53 automatically.

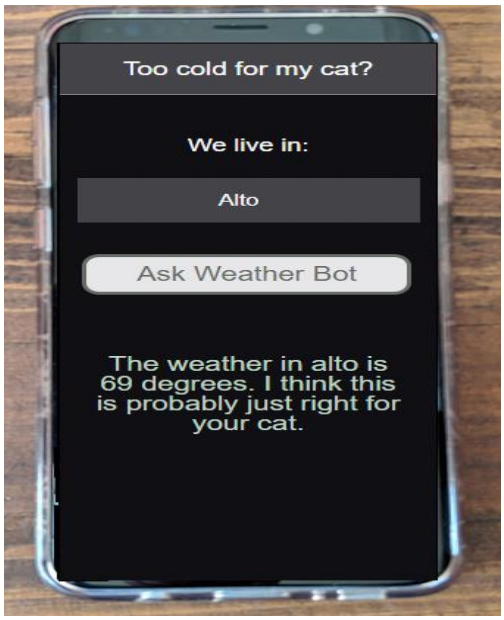
- On your local machine, open the folder where you unzipped the file and edit the file s3website/scripts/config.js
- Replace the var API_GATEWAY_URL_STR with the **Invoke URL** you copied from step 2.
- It will look like this:

```
var API_GATEWAY_URL_STR = null;
```

- Use your own deployed API gateway URL {above}. Like in this example:

```
var API_GATEWAY_URL_STR = "https://Your-Invoke-URL.execute-api.us-east-1.amazonaws.com/test";
```

- Save the file and upload it back to 53 into the **scripts** folder, remembering to set cache-Control to max age=O . ***(Refer to week 1 if you don't know how to upload items to*** *53)*
- Browse to your website {i.e. your CloudFront distribution URL)
- Choose a city, and press **Ask Weather Bot.**



- The weather should always return 69 degrees no matter which city is chosen. We now have a love MOCK API interacting with our website. ··Next week we will look at wiring up our API Gateway to a function (lambda), and will use IAM. and Cloudwatch.

See you next week!

- **Data and Results:**

- **Analysis and Inferences:**

Sample VIVA-VOCE Questions (In-Lab):

1. What is Amazon API Gateway, and how does it fit into the AWS ecosystem?

Ans:

2. What are the key use cases for Amazon API Gateway?

Ans:

3. What are the components and concepts involved in setting up an API Gateway endpoint.

Ans:

4. How do you create a new API in Amazon API Gateway?

Ans:

5. What are the different types of API endpoints that you can create in API Gateway?
Ans:

Post-Lab:

Build a Serverless Web Application with AWS Lambda, Amazon API Gateway, AWS Amplify, Amazon DynamoDB, and Amazon Cognito

- **Procedure/Program:**

- **Data and Results:**

- **Analysis and Inferences:**

Evaluator Remark (if Any):	**Marks Secured: _____out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.

#5. Experiment Title: Create an AWS Lambda Function to create a test case for our Lambda Mock

Aim/Objective: The aim of this task is to create an AWS Lambda function that generates test cases specifically designed for our Lambda Mock environment.

Description:

Currently we have our website communicating with a hard coded API mock.

We need to start adding functionality to the APL Such as having a function run when then API is hit that calls a database for weather information.

However to introduce you to the concept of server-less functions and the plumbing that runs it we shall start simple and create the worlds simplest function that returns. You guessed it, a hard coded value back through API Gateway all the way to the browser.

We need to create a server-less function that regardless of what city name you send, it will reply with a fixed temperature and echo back the city name.

This is essentially a Lambda Mock, to replace the API Gateway mock.

This is a useful learning step before making the Lambda function do anything intelligent. Even just creating a simple Lambda mock will introduce a few new services to you. Such as CloudWatch logs for debugging, and IAM for establishing both invocation permissions (what can trigger it [API gateway]) and the execution permission. Basically what Lambda can write to [CloudWatch logs]).

Pre-Requisites:

AWS FREE TIER ACCOUNT, LAMBDA FUCTION, API GATEWAY

Pre-Lab:

1) What is automation deployment in AWS lambda?

Ans:

2) How does lambda function handle failure during event processing?

Ans:

3) What are the disadvantages of implementing a serverless approach?

Ans:

4) When creating an AWS CloudFront distribution, which of the following is not an origin?
   A. Elastic Load Balancer
   B. AWS S3 bucket
   C. AWS MediaPackage channel endpoint
   D. AWS Lambda

5) You are a solutions architect working for an online retailer. Your online website uses REST API calls via API Gateway and Lambda from your Angular SPA front-end to interact with your DynamoDB data store. Your DynamoDB tables are used for customer preferences, account, and product information. When your web traffic spikes, some requests return a 429 error response. What might be the reason your requests are returning a 429 response (choose all the right answers)

A. Your Lambda function has exceeded the concurrency limit
B. DynamoDB concurrency limit has been exceeded
C. Your Angular service failed to connect to your API Gateway REST endpoint
D. Your Angular service cannot handle the volume spike
E. Your API Gateway has exceeded the steady-state request rate and burst limits

Ans:

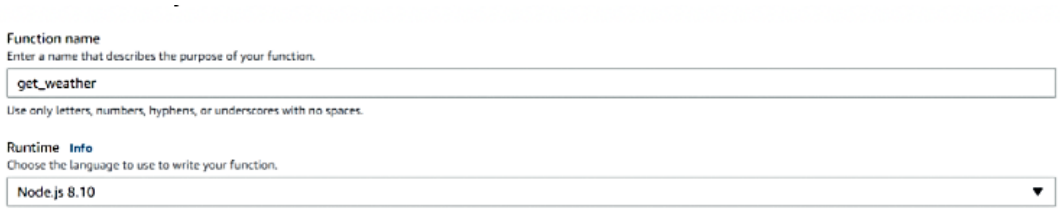
In-Lab:

Create an AWS Lambda Function to create a test case for our Lambda Mock

- **Procedure/Program:**

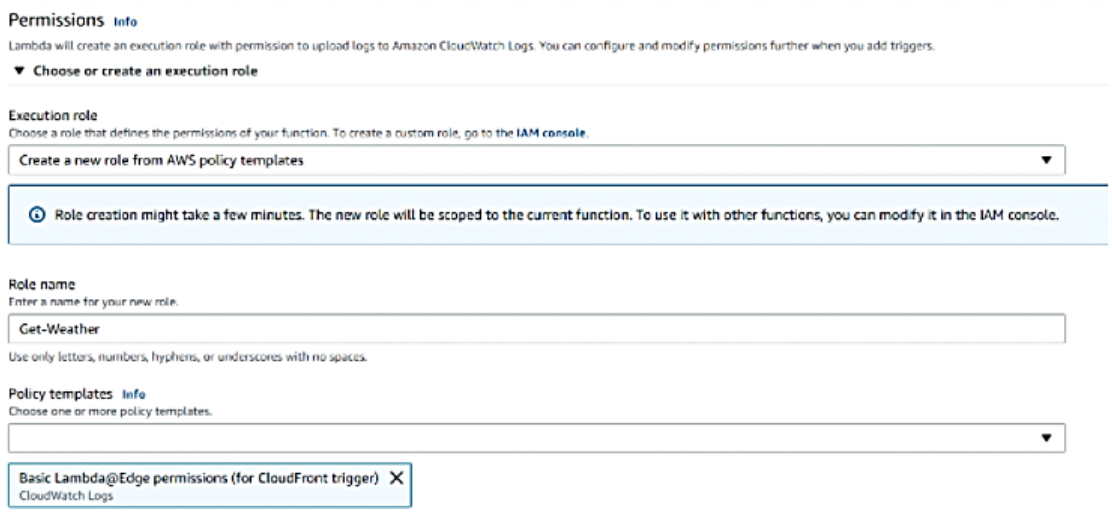
1. **Steps for create a simple Lambda function (our new mock)**

- Sign in to the AWS Management Console and in the **Find Services** search box type lambda and choose **Lambda**.
- Make sure you are in the **N.Virginia** region at the top right.
- Click **Create function**.
- Select **Author from scratch**.
  - o Name the function qet_weather .
  - o Select **Node.js 8.10** from the **Runtime** list.



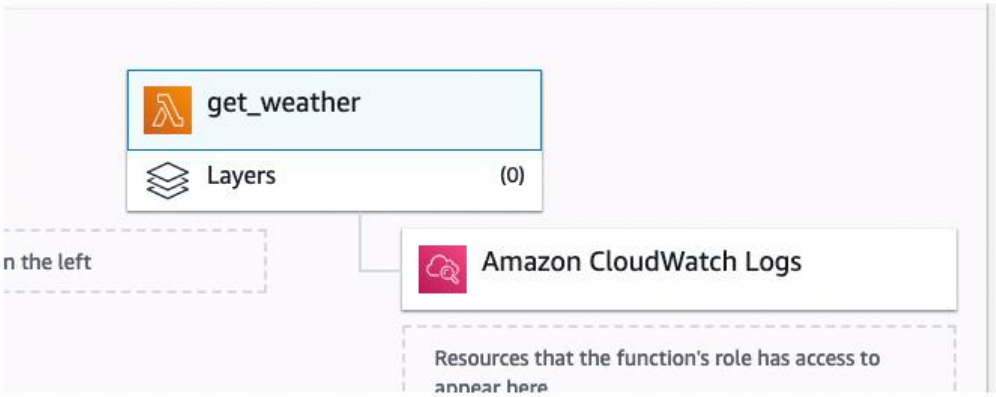
- o For **Execution role** choose **Create a new role from AWS policy templates**.
- o For **Role name** name the role **Get-weather** .
- o From the **Policy templates** list scroll down and choose **Basic Lambda@Edge permissions (For CloudFront trigger)**.

We use the **Basic Lambda@Edge permission** because it is a very managed simple policy that allows us to write to CloudWatch Logs, which is all we really need right now,in terms of executions permissions at least.



- Click **Create function**.
- This will give us permission to write to CloudWatch logs as well as create a log group.

```
{
        "Version": "2012-10-17",
        "Statement": [
        {
                "Effect": "Allow",
                "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents"
                ],
                "Resource": [
                        "arn:aws:logs:*:*:*"
                ]
        }
        ]
}
```



- Scroll down to the inline code editor.
- Paste the following code into the **index.js** tab replacing the existing code:

```
function handler(event, context, callback){
  var
     city_str = event.city_str,
          response = {
   city_str: city_str,
   temp_int: 74
 };
  console.log(response);
  callback(null, response);
}
exports.handler = handler;
```

This code simply takes the city as a string and will echo it back along with a 74 temperature, as a Number then exits successfully. You will notice we console log out the response too. This is so you can view this information in CloudWatch logs, if for example you needed to debug it. API Gateway can interact with this function, essentially acting as an intermediary between the clients browser and your Lambda function.

- No need to change any of the defaults.
- Click **Save.**

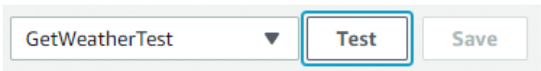
Again, later on we will make this Lambda function smarter, for now a mock is fine to get all the plumbing in place and to be able to test it all. Talking of testing, let's create a test case for our Lambda function and make sure it works as intended if we send it a random city.

2. Steps to create a test case for our Lambda Mock

- Click the dropdown that says **"select a test event"**
- Choose **configure test events**
- Keep the default **Create new test event** selected.
- For **Event template** leave it as **Hello World.**
- For **Event name** type in **GetWeatherTest.**
- Paste the following into the inline code editor:

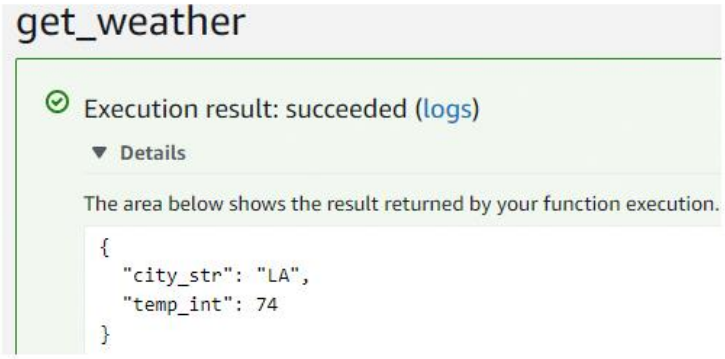
```
{

   "city_st r" : " LA"
}
```

- Click **Create.**
- The test case should now be saved at the top right.

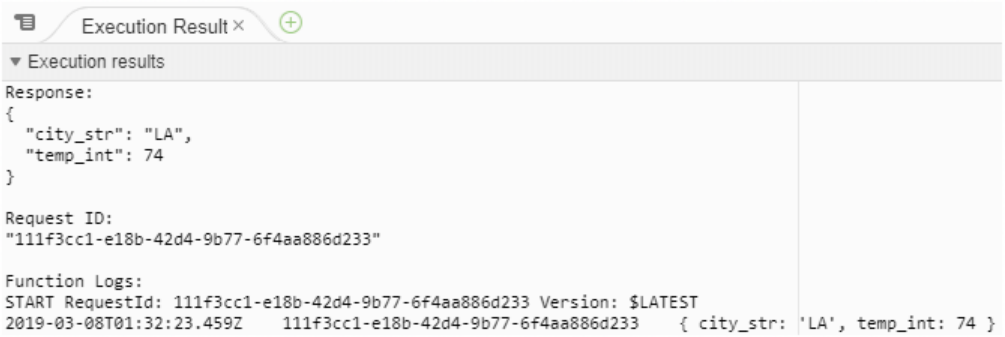


- Click **Test.**

You should now see at the top the execution succeeded:



You can also view this at the bottom of the inline code editor under the **Execution Result** tab.



You could edit the city_str with a new city if you like, but at this point, it should be working as expected.

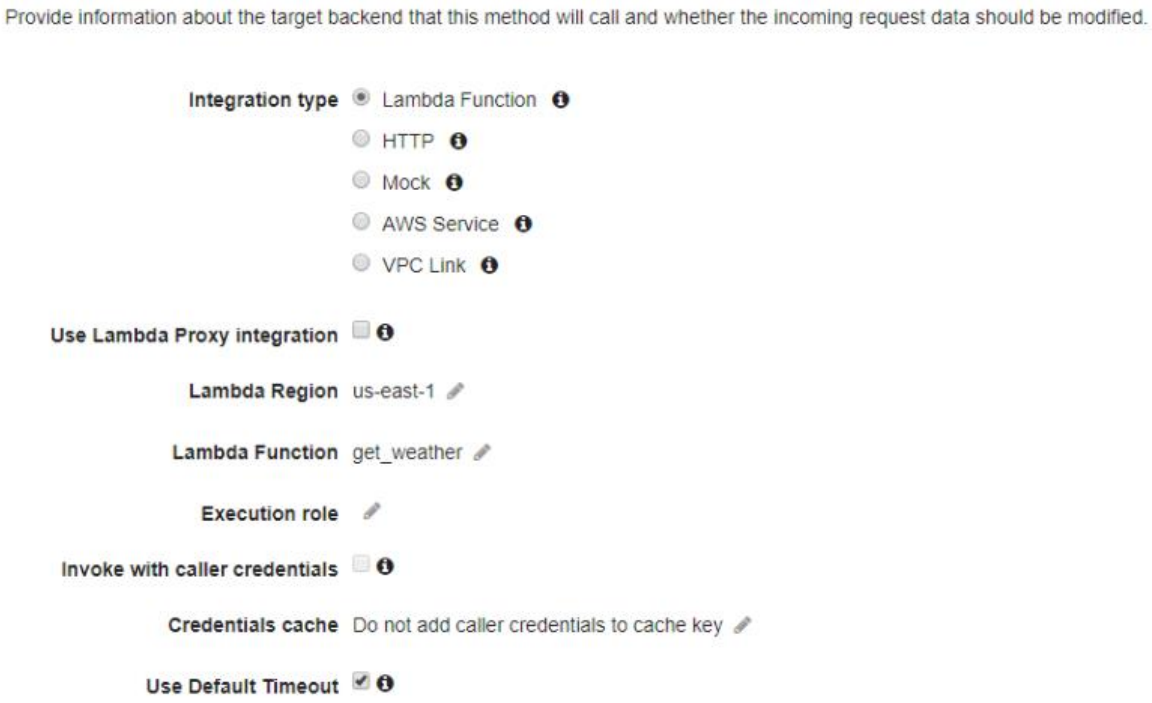
Currently our webs will still return 69 degrees because the API Gateway that it is pointing to is a hard coded mock. All we need to do is go into the API gateway console and replace the hard coded text mock that we created in week 2 with this new lambda mock.

We only need to make a small change to our API's configuration in the console, and then we should be able to test our website and get 74 degrees returned instead of 69.

3. **Steps to replace the API mock endpoint with the get_weather Lambda function**

- Click **Services** and search for API, and then select **API Gateway**.
- Select the **CatWeather** API under **APIs**.
- Under resources choose **POST** and click **Integration Request**.
- For the **Integration type** change **Mock** to **Lambda Function**.
- ENSURE **Use Lambda Proxy Integration** is **de-selected**.
- Choose the **Lambda Region**: us-east-1.
- Type in the name of the Lambda function **get_weather** . (Once you type in g you should be able to select **get_weather** from the list).
- Leave **Use Default Timeout** checked.
- Click **Save**.
- At the **Switch to Lambda integration** pop-up. Click **OK**.

- It will tell you that you are adding **Permissions to Lambda Function**. Click **OK.**



4. Test

- Click **Method Execution** at the top: **←Method Execution**

- Click **Test:**
- Paste the following into the **Request Body:**

```
{
      " c ty_str " : "SEATTLE "
}
```

- Click **Test.**
- Under **Response Body** we can see the desired result:

```
{
"city_str": "SEATTLE",
"temp_int": 74
}
```

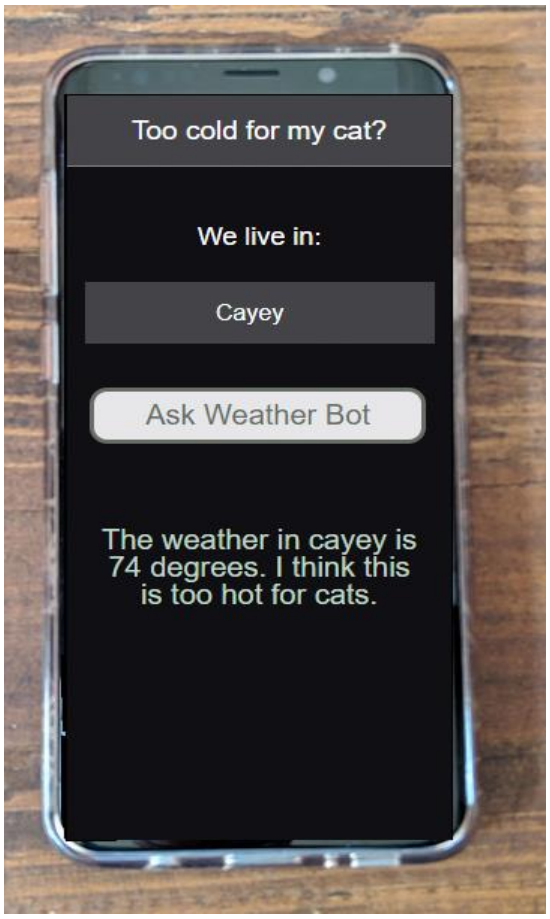
- We can also see in the **Logs** section that the request was indeed sent to our Lambda function: Similar to:

Sending request to https://lambda.us-east-1.amazonaws.com/2015-03-31/functions/arn:aws:lambda:us-east-1:179741345863:function:get_weather/invocations

Method response body after transformations: {"city_str":"SEATTLE","temp_int":74}

- We now need to re-enable **CORS** by going to • /
- Click **Actions** and **Enable CORS.** Again select **DEFAULT 4XX** and **DEFAULT sxx.**

- Click **Enable CORS and replace existing CORS headers.**
- Click **Yes, replace existing values** on the **Confirm method changes** pop-up.
- Click **Actions** again and **Deploy API.**
- Set **Deployment stage to test.**
- Click **Deploy.**
- We can now browse back to our CloudFront website.
- Choose a city,and click **Ask Weather Bot.**
- You should now see the update from Lambda.



Every city will now return 74 degrees.

Awesome you now have a server-less website that hits a functional (albeit a bit a bit dumb) backend. We will now work to make it smarter by asking a database for real weather information based on city. Of course we will need a database for that, so we will do that next. We are doing well on our checklist tho'!

- **Data and Results:**

- **Analysis and Inferences:**

Sample VIVA-VOCE Questions (In-Lab):

1. What is an AWS Lambda function, and what are its primary use cases?

Ans:

2. Can you briefly describe the typical structure of a Lambda function written in a programming language like Python or Node.js?

Ans:

3. What is the significance of test cases in software development, and why might you want to create test cases for Lambda mocks?

Ans:

4. How do you configure and set up a test event in the AWS Lambda Console, and why is it useful when testing Lambda functions?

Ans:

5. What is the purpose of using JSON format for representing test cases, and how can you pass input data to a Lambda function for generating test cases?

Ans:

Post-Lab:

Schedule a Serverless Workflow

- **Procedure/Program:**

- **Data and Results:**

- **Analysis and Inferences:**

Evaluator Remark (if Any):	**Marks Secured: _____out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.

#6. Experiment Title: Create A DynamoDB Table for seeding weather table from CSV file for API

Aim/Objective:

The aim of this task is to establish an Amazon DynamoDB table specifically designed for storing weather data imported from a CSV file. By accomplishing this, we aim to create a reliable and scalable data repository that will serve as the backend storage for our weather data API.

Description:

We have our Lambda function all wired up, so all we need to do is change the function code to query a database. Which we don't have. Yet ;)

We are going to create a DynamoDB table, and use lambda to seed it with data that we will provide in CSV format. Then test query the database, and run a few test cases for Lambda to make sure it is all wired up correctly.

As far as the dynamo table goes we are not going to create a sort key or an index. We are also foregoing a clever schema, because this lab is a one trick pony that only needs to do one simple search: "What is the temp of this city".

Pretty easy structure, its almost spreadsheet like! Out Table will look a bit like this

Primary Key [SC]	**T**
NORTH LAS VEGAS	66
CHICAGO	0
SEATILE	46

We will save all the really cool Dynamo deep dive stuff and schema design for another dedicated course.

Pre-Requisites:

AMAZON FREE TIER ACCOUNT, DYNAMODB, APIGATEWAY

Pre-Lab:

1) What do you understand about DynamoDB Streams?

Ans:

2) Are there any limitations to using DynamoDB? If yes, then what are they?

Ans:

3)Describe about global secondary indexes in DynamoDB

Ans:

4)An application currently writes a large number of records to a DynamoDB table in one region. There is a requirement for a secondary application to retrieve new records written to the DynamoDB table every 2 hours and process the updates accordingly. Which of the following is an ideal way to ensure that the secondary application gets the relevant changes from the DynamoDB table?

A. Insert a timestamp for each record and then scan the entire table for the timestamp asper the last 2 hours.

B. Create another DynamoDB table with the records modified in the last 2 hours.

C. Use DynamoDB Streams to monitor the changes in the DynamoDB table.

D. Transfer records to S3 which were modified in the last 2 hours.

Ans:

5)A team is building an application that must persist and index JSON data in a highly available data store. Latency of data access must remain consistent despite very high application traffic.What service should the team choose for the above requirement?

Ans:

In-Lab:

CREATING A DynamoDB TABLE for seeding weather table from CSV file for API.

- **Procedure/Program:**

We have our Lambda function all wired up, so all we need to do is change the function code to query a database.

Which we don't have. Yet ;)

We are going to create a DynamoDB table, and use lambda to seed it with data that we will provide in CSV format. Then test query the database, and run a few test cases for Lambda to make sure it is all wired up correctly.

As far as the dynamo table goes we are not going to create a sort key or an index. We are also foregoing a clever schema, because this lab is a one trick pony that only needs to do one simple search: "What is the temp of this city".

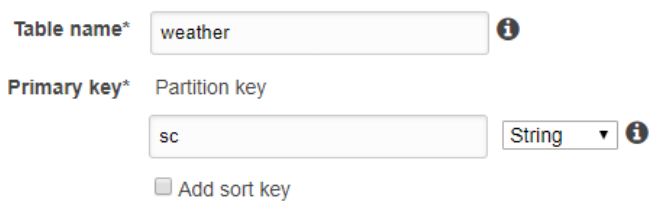
Pretty easy structure, its almost spreadsheet like! Out Table will look a bit like this

Primary Key [SC]	T
NORTH LAS VEGAS	66
CHICAGO	0
SEATILE	46

We will save all the really cool Dynamo deep dive stuff and schema design for another dedicated course.

1. Steps for creating a simple Dynamo DB table

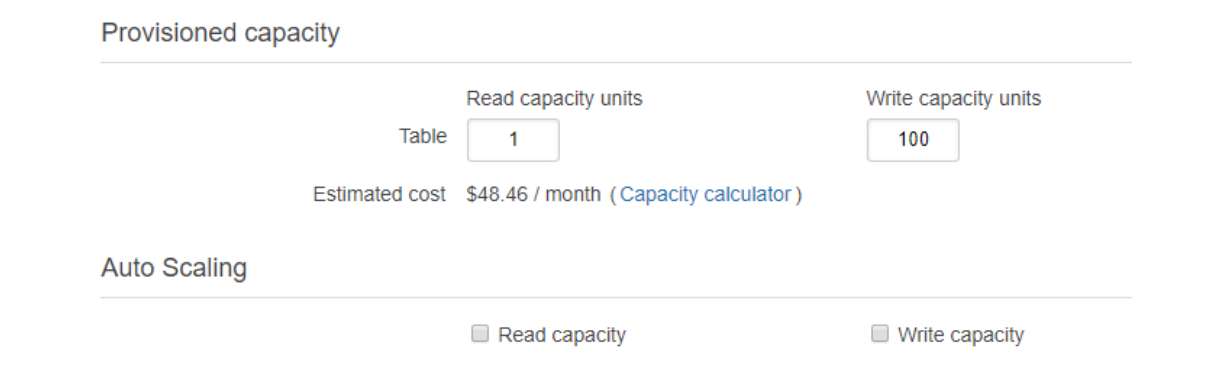
- Sign in to the AWS Management Console and in the Find Services search box type dynamo and choose **DynamoDB.**
- Make sure you are in the **N. Virginia** region at the top right.
- Click **Create table.**
- For **Table name** type weather .
- For **Primary key** type in sc (for searchable city) and leave **String** selected.



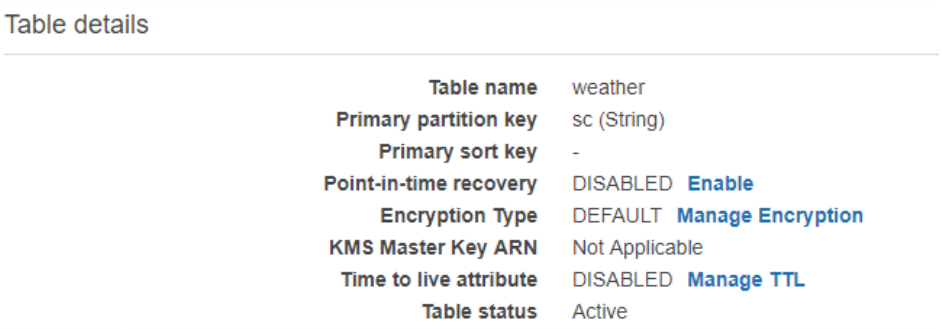
- Remove the check for **Use default settings.**
- Under Auto Scaling remove the check for **Read capacity** and **Write capacity.**
- Above that under **Provisioned capacity** change the **Read capacity** units to 1 and

change the **Write capacity units** to 100 .

We will dial this back down shortly, we only need to have it set to 100 while we seed the table.



- Leave **DEFAULT** selected under **Encryption At Rest.**
- Click **Create.** Wait and verify the **Table status** becomes **Active.**



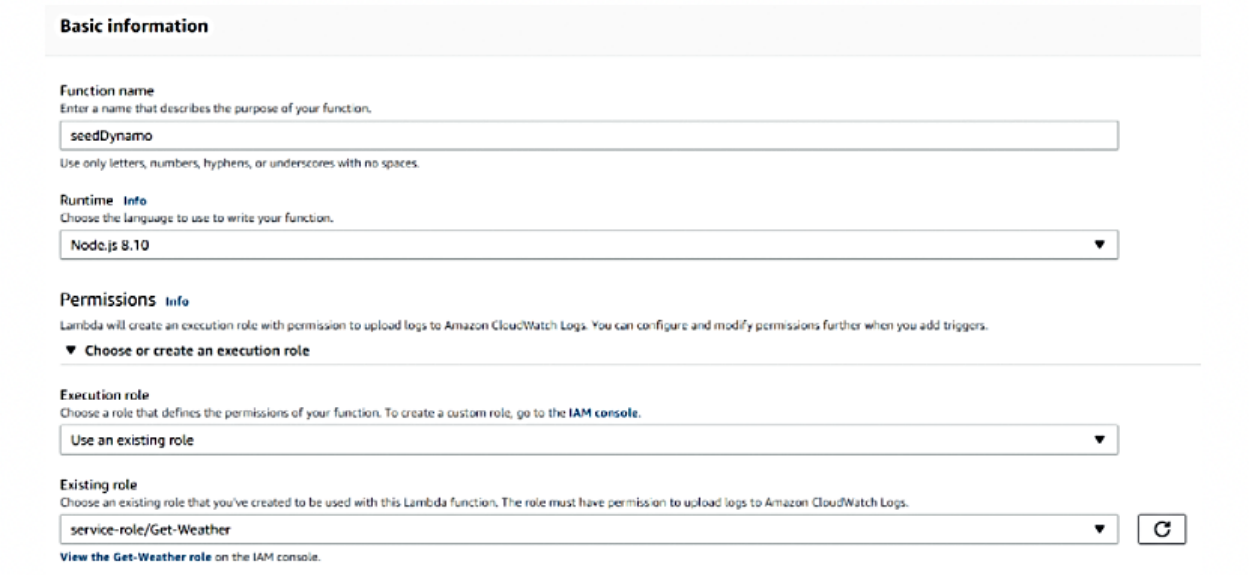
Sometimes it takes a few minutes before the table is active. We must wait for it to say active (keep refreshing) before we try and add data to it

Once you have it Active we can create a Lambda function that will take a provided CSV file and parse it and throw it into Dynamo. That way we have weather data in our database and should be able to do basic queries on it, getting the temperature for that city.

This CSV file is not live data, so in a way it feels like we are cheating. However in the real world you would likely update Dynamo in real time based upon hitting a third party API. This CSV data is fine for our purposes though. Atleast each city has a different temperatures now, so this will help our website "fake it" pretty well.

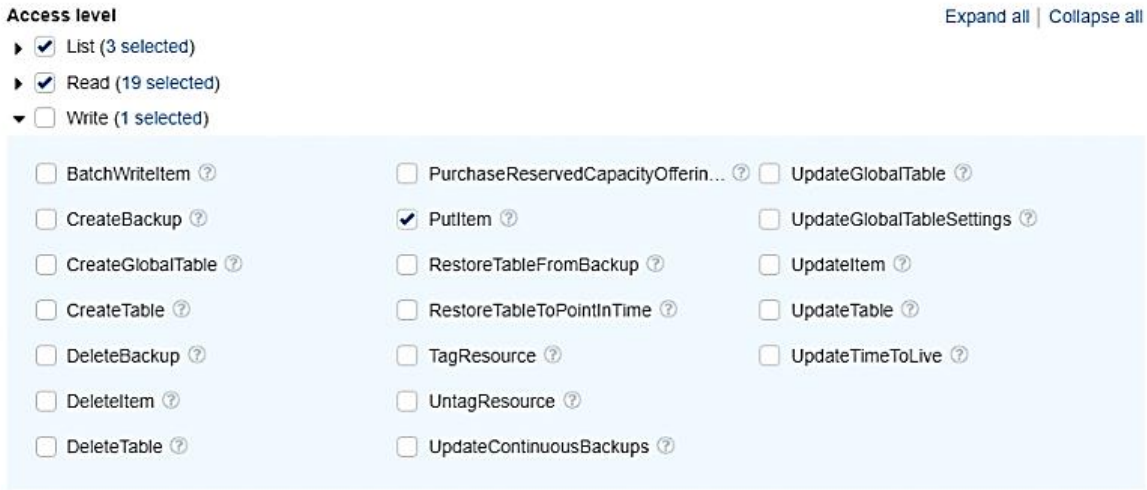
2. Steps for seeding the weather table from a CSV

- Click **Services** and type lambda in the **Find Services** search box and choose **Lambda.**
- Click **create function.**
- For **Function name** type in seedDynamo .
- Leave **Node.js 8.10** for **Runtime.**
- For **Execution role** select **Use an existing role.**
- For **Existing role** select our **service-role/Get-Weather.**



- Click View the Get-Weather rote on the lAM console. This will pivot to the IAM console.
- Expand the policy by clicking on the drop down arrow.

- Click **Edit policy .**
- Click **Add additional permissions**. As Lambda will need to write co DynamoDB.
- Click **Choose a service.**
- Type dynamo in the **Find a service** search box and choose **DynamoDB.**
- Under **Access level** select **List** and **Read.**
- Expand **Write** and choose **PutItem.**



- Under **Resources** select **All resources.**

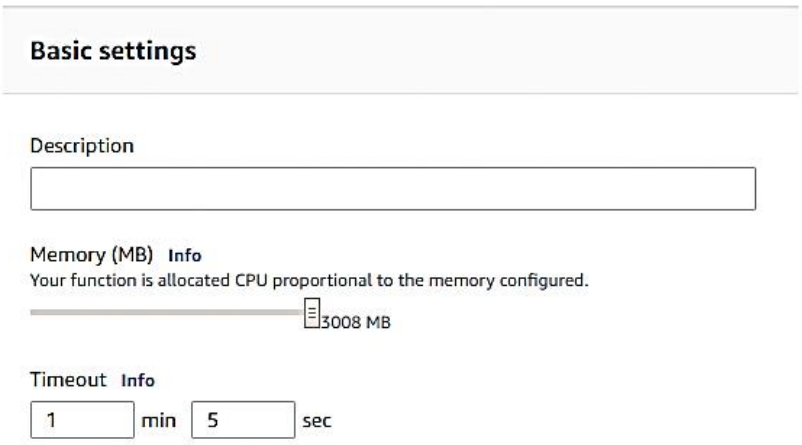
- Click **Review policy.**
- Click **Save changes.**
- Go back to your **Lambda** tab and click **Create function.**

- Paste the following in the **index.js** tab:

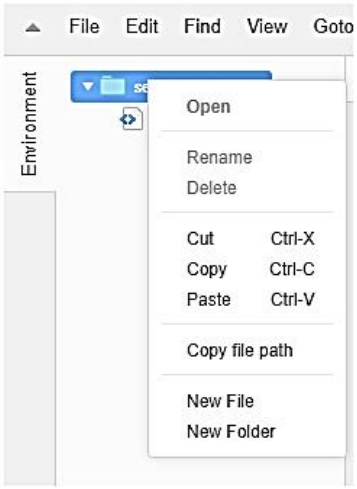
```
exports.handler = function(event, context, callback) {
  var
    AWS = require("aws-sdk"),
    fs = require("fs"),
    item = {},
    some_temp_int = 0,
     params = {},
    DDB = new AWS.DynamoDB;

  AWS.config.update({
    region: "us-east-1"
  });
  fs.readFileSync("cities.csv", "utf8").split('\n').map(function(item_str){
    params.ReturnConsumedCapacity = "TOTAL";
    params.TableName = "weather";
    params.Item = {
      "sc": {
        "S": item_str.split(",")[0]
      },
      "t": {
        "N": String(item_str.split(",")[1])
      }
    };
    DDB.putItem(params, function(err, data){
      if(err){
        console.error(err);
      }else{
        //ignore output
      }
    });
  });
 setTimeout(function(){
   callback(null, "ok");
 }, 1000 * 10);
}
```

This code simply reads a CSV file parses it, and inserts the data into the table. When it is done, it exits.

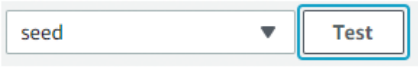
- Scroll down to **Basic settings** and change **Timeout** to 1 min and s sec and set the **Memory** to 3008 MB.



- Click **Save** at the top right.
- At the left of the inline code editor where it says **Environment**. Right click the **seedDynamo** folder and click **New File.**



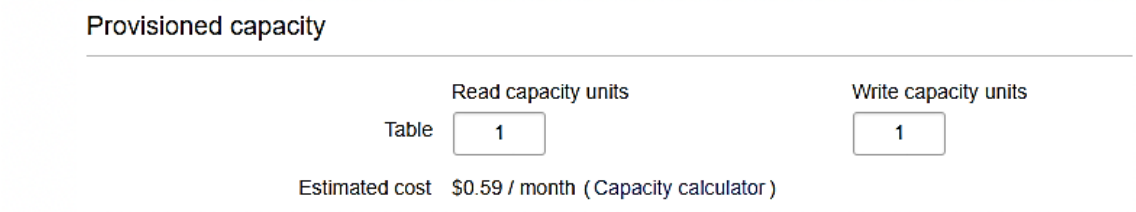
- Name the file cities.csv .
- Double click the **cities.csv** file and it will open in a new tab.
- Copy and paste the contents from the **cities_template.md** file which is located in the website zip file you downloaded in week 1.
- Click **save.**
- Click **Test.** Any test payload we provide will trigger the function, which is all we need to do. The payload you provide is irrelevant.
- For **Event template** leave the **Hello World** setting.
- Under **Event name** type in seed .
- Click **Create.**
- Once its populated:



- Click **Test.** This will populate our **weather** table with the list of cities from our csv file.

This can take up to a full minute, before you see this in the Lambda console.

- We want to verify that our DynamoDB table has been populated properly. We also want to change the capacity back to 1 . **As you don't want to be charged for 100WCUs on an ongoing basis**
- Click on **Services** type in dynamo in the search box or choose **DynamoDB** from the **History.**
- Click **Tables** and click our **weather** table.
- Select the **Capacity** tab and under **Provisioned capacity** set the **Write capacity units** to 1.



- Click **Save.**

Now let's check our table items.

- Click the **Items** tab.
- Change Scan to Query and enter **ALTO** as the value. Then click **Start search**.

- We can see the query returned the correct value **47** from the table.
- We can now remove our **seedDynamo** function.
- Click **Services** and choose **Lambda** from the **History** list.
- Select **seedDynamo** under **Functions.**
- Click **Actions** and **Delete.** On the pop-up window once again click **Delete.**

You notice we cleaned up after ourselves by deleting that seed function. We recommend at the end of the course that you remove all your created assets otherwise costs (albeit small) will occur if you go outside of the free tier.

Ok we have a database, and we have tested that we can query it. Now lets change the Lambda function mock code to something a little more interesting. We will make our Lambda function collect the city that is passed (like before), and instead of just returning it with a random temperature, it is going to issue a request to DynamoDB. Passing the city and hopefully getting a temperature back which it can use to send all the way though API Gateway and back to the users browser.

3. Steps to adjust the Lambda function code so it will query DynamoDB

- We should be in our **Lambda** console already.
  - But if you somehow browsed away from it. Click **Services** and search for lambda or choose **Lambda** from the **History.**
- Click our **get_weather** function.
- In the inline code editor replace the contents of **index.js** with:

```
function handler(event, context, callback){
  var
    AWS = require("aws-sdk"),
    DDB = new AWS.DynamoDB({
      apiVersion: "2012-08-10",
      region: "us-east-1"
    }),

    city_str = event.city_str.toUpperCase(),
    data = {
      city_str: city_str,
      temp_int_str: 72
    },
    response = {},
    params = {
      TableName: "weather",
      KeyConditionExpression: "sc = :v1",
      ExpressionAttributeValues: {
        ":v1":{
          S: city_str
        }
      }
    };

  DDB.query(params, function(err, data){
    var
      item = {},
      response = {
```

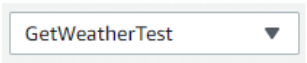
```
            statusCode: 200,
            headers: {},
            body: null
          };
        if(err){
          response.statusCode = 500;
          console.log(err);
          response.body = err;
        }else{
          // console.log(data.Items[0]);
          var data = data.Items[0];
          if(data && data.t){
            console.log(data.sc.S + " and " + data.t.N);
            item = {
              temp_int:Number(data.t.N),
              city_str: data.sc.S
            };
          }else{
            item = {
              city_str: event.city_str
             //when we don't return a temp, the client can say city not found
            };
          }
        }
        response = item;
       // console.log(response);
        callback(null, response);
      });
    }
    exports.handler = handler;
```

This code takes the city and passes it to Dyna mo as the partition key. This enables the Dynamo SDK to find the right temperatu re and return it along with the city.

- Change the **Timeout** to **15** seconds
- Click **Save** at the top right.

Let's create a test case for it, and ensure we can get a different temp for each city.

- We can use our **GetWeatherTest** case. Click the drop down arrow.



- Choose **Configure test events** and change to a desired city. For example:

```
{
  "city_str": "DENVER"
}
```

- Click **Save.**
- Click **Test.**
- We should now see the following data returned, and temp is coming from our **weather** table:

```
{
  "temp_int": 38,
  "city_str": "DENVER"
}
```

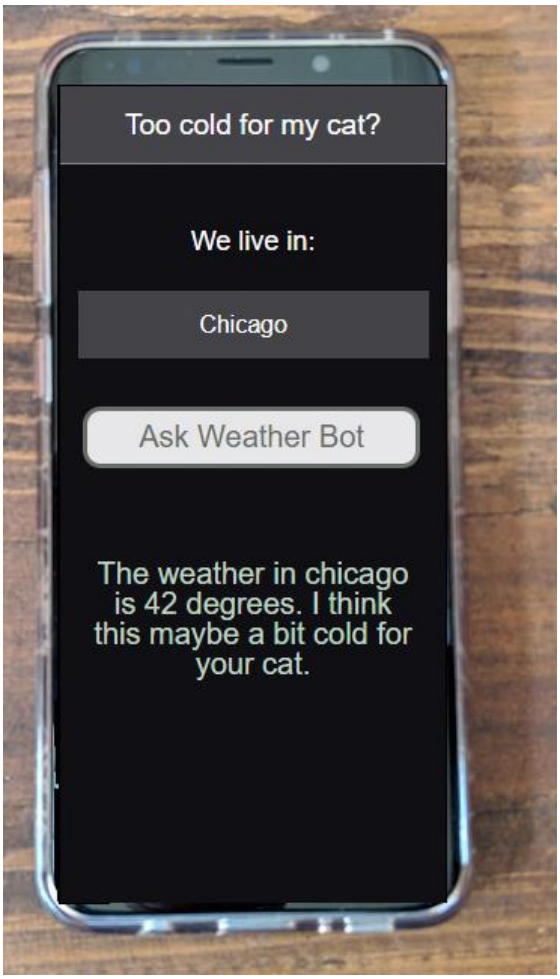

- Feel free to play with the **GetWeatherTest** event and type in different cities. Use Capitals


```
{
  "city_str": "PITTSBURGH"
}
```


Should return:


```
{
  "temp_int": 78,
  "city_str": "PITTSBURGH"
}
```


- Keep in mind the city must be in our **cities.csv** file or it will simply return **city_str**. Which is a cue to the front end website to say something like: "No city found, please try again".
- We can now visit our website (i.e your CloudFront URL)
- Now when we choose a city and click **Ask Weather Bot** we should get data returned to us from our **weather** DynamoDB table:
- Depending upon the weather temp, the JavaScript in the front end website will add a 2 cent comment, suggesting that it is either too hot, too cold, or just right for your cat.



We are faking it pretty well right now, we have a server-less data driven text weather app running in a global content delivery network for low latency.

- **Data and Results:**

- **Analysis and Inferences:**

Sample VIVA-VOCE Questions (In-Lab):

1. Can you explain the steps involved in creating a DynamoDB table for seeding a weather table with data from a CSV file for use with an API?

Ans:

2. What factors should be considered when designing the schema of the DynamoDB table to store weather data imported from a CSV file?

Ans:

3. How would you define the primary key for the DynamoDB table when seeding weather data, and what is the significance of choosing the right primary key attributes?

Ans:

4. Can you describe the process of importing weather data from a CSV file into the DynamoDB table, and are there any considerations for handling data transformations during this process?

Ans:

5. In the context of an API, how would you use the DynamoDB table you've created to serve weather data, and what AWS services or components might be involved in connecting the DynamoDB table to the API?

Ans:

Post-Lab:

Build a turn-based game with Amazon DynamoDB and Amazon SNS

- **Procedure/Program:**

- **Data and Results:**

- **Analysis and Inferences:**

Reference:

https://aws.amazon.com/tutorials/turn-based-game-dynamodb-amazon-sns/module-seven/

Evaluator Remark (if Any):	**Marks Secured: _____out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.

#7. Experiment Title: Build aserverless web application for Dynamic website hosting using S3 , Route53 ,CloudFront and Certificate Manager

Aim/Objective:

The aim of this experiment is to construct a serverless web application that offers dynamic and scalable website hosting.

Description:

Building a serverless web application for dynamic website hosting is a comprehensive project that takes full advantage of the AWS ecosystem's capabilities. This approach allows us to provide users with a seamless and secure web experience while focusing on cost efficiency and ease of management. Leveraging AWS services, our application will harness the power of Amazon S3 for reliable content storage and access, making it possible to host dynamic website elements. The integration of AWS Route 53 enables us to manage domain names and direct traffic to the application seamlessly.

To ensure optimal performance and a global reach, AWS CloudFront, a content delivery network (CDN), will be utilized to distribute website content from edge locations around the world.

Pre-Requisites:

AMAZON FREE TIER ACCOUNT, Route53 , APIGateway, Cloudfront, S3, Certificate Manager, freenom domain name

Pre-Lab:

1)What is AWS Certificate Manager?

Ans:

2)What is AWS Route 53 traffic flow?

Ans:

3)List the differences between AWS Route 53 and API Gateway

Ans:

4)A company currently hosts their architecture in the US region. They now need to duplicate this architecture to the Europe region and extend the application hosted on this architecture to the new region. In order to ensure that users across the globe get the same seamless experience from either setups, what among the following needs to be done?

A. Create a Classic Elastic Load Balancer setup to route traffic to both locations.

B. Create a weighted Route 53 policy to route the policy based on the weightage for each location

C. Create an Application Elastic Load Balancer setup to route traffic to bothlocations.

D. Create a Geolocation Route 53 Policy to route the policy based on the location.

Ans:

5) Your company is planning on using Route 53 as the DNS provider. There is a need to ensure that the company's domain name points to an existing CloudFront distribution. How can this be achieved?

A. Create an Alias record which points to the CloudFront distribution.

B. Create a host record which points to the CloudFront distribution.

C. Create a CNAME record which points to the CloudFront distribution

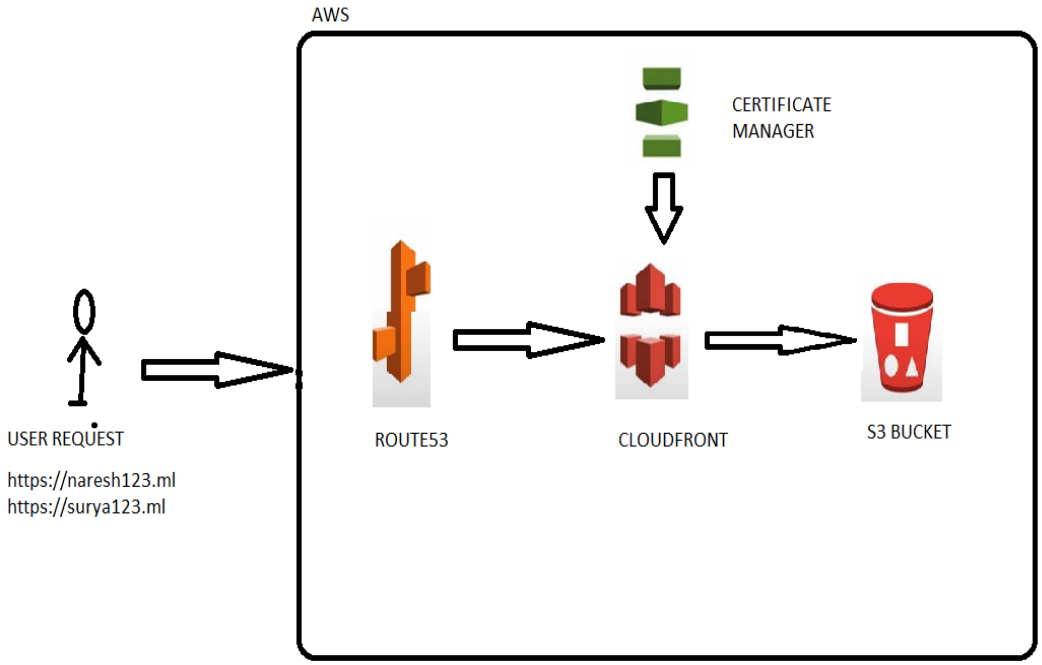
D. Create a Non-Alias Record which points to the CloudFront distribution.

Ans:

In-Lab:

Build a serverless web application for Dynamic website hosting using S3 , Route53 ,CloudFront and Certificate Manager

- **Procedure/Program:**

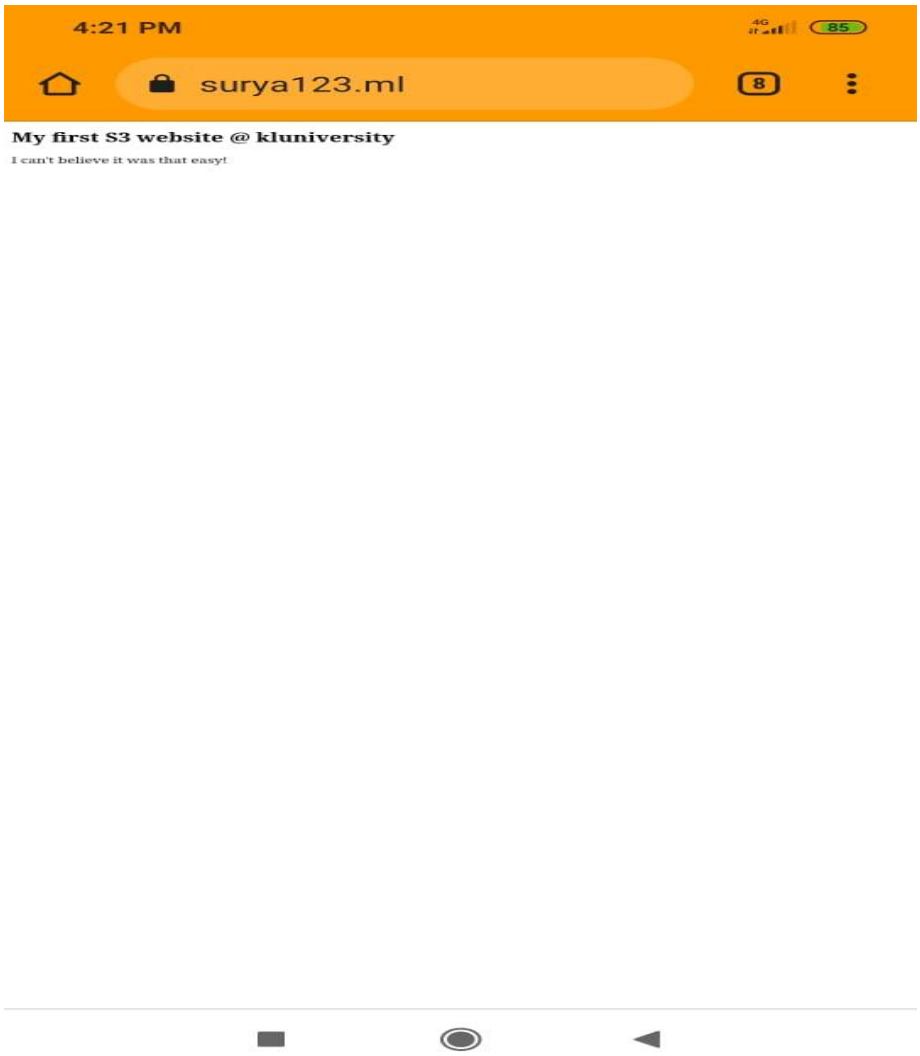


STEP BY STEP:

1. Initially will create s3 bucket in N.Virginia
2. Upload index.html file and make it public
3. Enable static webhosting (index.html)
4. Bucket policy /*(copy only after https://) in browser
5. Freenom website(naresh123.ml)
6. Route53(naresh123.ml)-NS&SOA
7. CloudFront(*.naresh123.ml, naresh123.ml)
8. Redirect http to https
9. Request for Certificate Manager(N.Virginia)
10. Request CNAME into Route53
11. Certificate issue
12. Add certificate and add naresh123.ml
13. Copy cloudfront domainname in browser
14. Create A record in Route53 (alias choos cloudfront)
15. Paste https://naresh123.ml in browser

REFER:

https://kluniversityin-my.sharepoint.com/:f:/g/personal/naresh_vurukonda_kluniversity_in/EhaiQ0-b7xVBgUV06dW_Iw8B5FeWf4_RHuKwU6JB47OAFA?e=qRdpqV



- **Data and Results:**

- **Analysis and Inferences:**

Sample VIVA-VOCE Questions (In-Lab):

1. How do you host a dynamic website using serverless components like S3, Route53, CloudFront, and Certificate Manager?

Ans:

2. What is the purpose of Amazon S3 in hosting a serverless web application?

Ans:

3. How does Amazon CloudFront improve website performance in a serverless architecture?

Ans:

4. What is the role of AWS Route53 in connecting your domain to a serverless web application?

Ans:

5. Why is AWS Certificate Manager used for securing a serverless web application?

Ans:

Post-Lab:

Protect Data on Amazon S3 Against Accidental Deletion or Application Bugs Using S3 Versioning, S3 Object Lock, and S3 Replication

- **Procedure/Program:**

- **Data and Results:**

- **Analysis and Inferences:**

Evaluator Remark (if Any):	**Marks Secured: _____out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

#8. Experiment Title: Develop Serverless Web Application on AWS using GET-METHOD getemployeedetails by email

Aim/Objective:

The aim of this project is to create a serverless web application on Amazon Web Services (AWS) that utilizes the HTTP GET method to retrieve employee details based on their email address.

Description:

Developing a serverless web application on AWS that employs the HTTP GET method to retrieve employee details by email represents a forward-thinking approach to managing HR-related data and enhancing organizational efficiency. This project leverages the power of AWS Lambda functions, AWS API Gateway, and a data storage service like Amazon DynamoDB to create a scalable, cost-effective, and easily maintainable application

Pre-Requisites:

AWS FREE TIER ACCOUNT, API GATEWAY, LAMBDA, CLOUDFRONT, DYNAMODB,S3

Pre-Lab:

1) What does an Amazon API Gateway resource policy entail?

Ans:

2) How can we build HTTP APIs using API Gateway?

Ans:

3) How do you enable allow CORS for API gateway?

Ans:

4) A gaming company planned to launch their new gaming application that will be in both web and mobile platforms. The company considers using GraphQL API to securely query or update data through a single endpoint from multiple databases, microservices, and several other API endpoints. They also want some portions of the data to be updated and accessed in real-time. The customer prefers to build this new application mostly on serverless components of AWS. As a Solutions Architect, which of the following AWS services would you recommend the customer to develop their GraphQL API?
   A. Kinesis Data Firehose
   B. Amazon Neptune
   C. Amazon API Gateway
   D. AWS AppSync

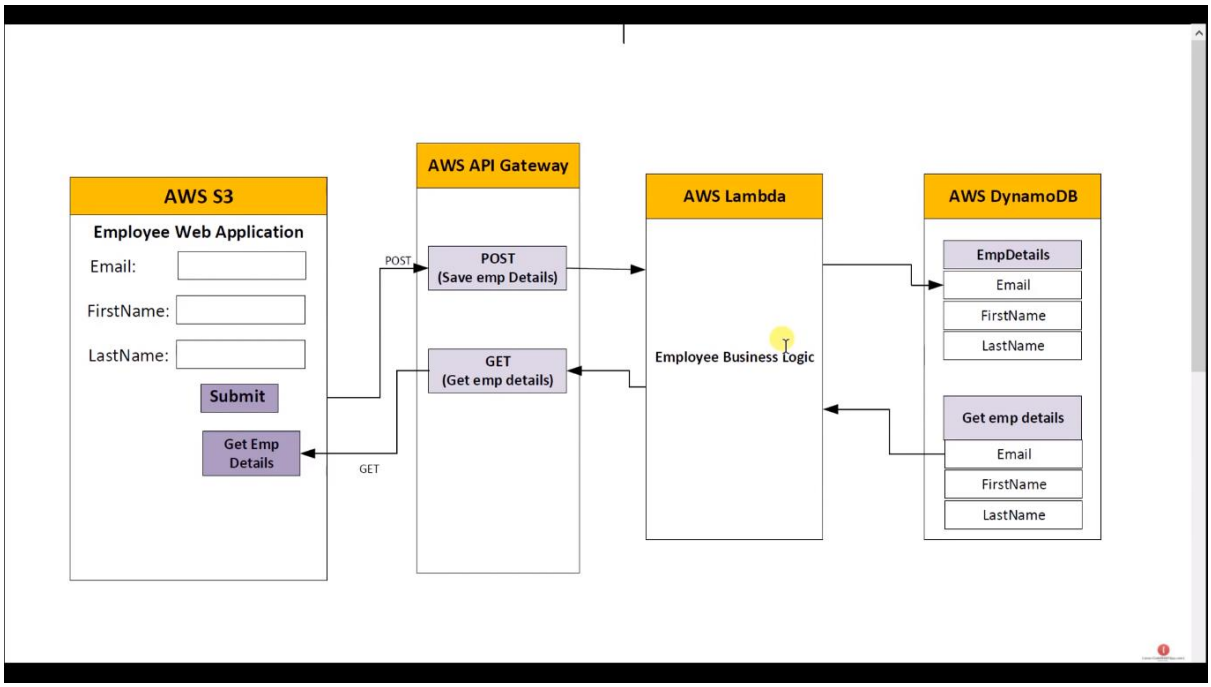
5) You are a solutions architect working for an online retailer. Your online website uses REST API calls via API Gateway and Lambda from your Angular SPA front-end to interact with your DynamoDB data store. Your DynamoDB tables are used for customer preferences, account, and product information. When your web traffic spikes, some requests return a 429 error response. What might be the reason your requests are returning a 429 response
A. Your Lambda function has exceeded the concurrency limit
B. DynamoDB concurrency limit has been exceeded
C. Your Angular service failed to connect to your API Gateway REST endpoint
D. Your Angular service cannot handle the volume spike
E. Your API Gateway has exceeded the steady-state request rate and burst limits

Ans:

In-Lab:

Develop Serverless WebApplication on AWS using GET-METHOD getemployeedeails by email .

- **Procedure/Program:**



REFER: https://kluniversityin-my.sharepoint.com/:f:/g/personal/naresh_vurukonda_kluniversity_in/EqZzfykCMDBMixQkQqfnuoUBgJKsYik1kWDBUyHvCTsXfQ?e=BalCVT

- **Data and Results:**

- **Analysis and Inferences:**

Sample VIVA-VOCE Questions (In-Lab):

1. How would you design a serverless web application on AWS to implement a GET method for retrieving employee details by email?

Ans:

2. What AWS services and components might be involved in creating a serverless web application with a "getemployeedetails" endpoint?

Ans:

3. What are the key considerations when designing the API for retrieving employee details using the email as a parameter?

Ans:

4. How can you ensure security and access control for the GET method to protect employee data in your serverless web application?

Ans:

5. What are some common use cases for implementing serverless web applications with specific GET methods like "getemployeedetails by email" on AWS?

Ans:

Post-Lab:

Publish Amazon SNS Messages Privately With Amazon SNS, Amazon VPC, Amazon EC2, Amazon CloudFormation, and AWS Lambda

https://docs.aws.amazon.com/sns/latest/dg/sns-vpc-tutorial.html

- **Procedure/Program:**

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

- **Data and Results:**

- **Analysis and Inferences:**

Evaluator Remark (if Any):	**Marks Secured: _____out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.

#9. Experiment Title: Develop Serverless Web Application on AWS using POST-METHOD postemployeedetails using email

Aim/Objective:

The aim of this project is to create a serverless web application on Amazon Web Services (AWS) that utilizes the HTTP POST method to add employee details based on their email address

Description:

Developing a serverless web application on AWS that leverages the HTTP POST method to add employee details by email is a forward-thinking approach to streamlining HR data management and improving organizational efficiency. This project harnesses the capabilities of AWS Lambda functions, AWS API Gateway, and a data storage service like Amazon DynamoDB to create a scalable, cost-effective, and easily maintainable application.

Pre-Requisites:

AWS FREE TIER ACCOUNT, API GATEWAY, LAMBDA, CLOUDFRONT, DYNAMODB,S3

Pre-Lab:

1) Does API gateway invoke Lambda synchronously? If so explain?
   Ans:

2) What are different ways to trigger Lambda?
   Ans:

3) What advantages does an API Gateway offer?
   Ans:

4) A team is building an HTML form that is hosted in a public Amazon S3 bucket. The form uses JavaScript to post data to an Amazon API Gateway API endpoint. The API endpoint is integrated with AWS Lambda functions. The team has tested each method in the API Gateway console and has received valid responses.Which combination of steps must the team complete so that the form can successfully post to the API endpoint and receive a valid response? (Select TWO.)
   A) Configure the S3 bucket to allow cross-origin resource sharing (CORS).
   B) Host the form on Amazon EC2 rather than on Amazon S3.
   C) Request a quota increase for API Gateway.
   D) Enable cross-origin resource sharing (CORS) in API Gateway.
   E) Configure the S3 bucket for web hosting.

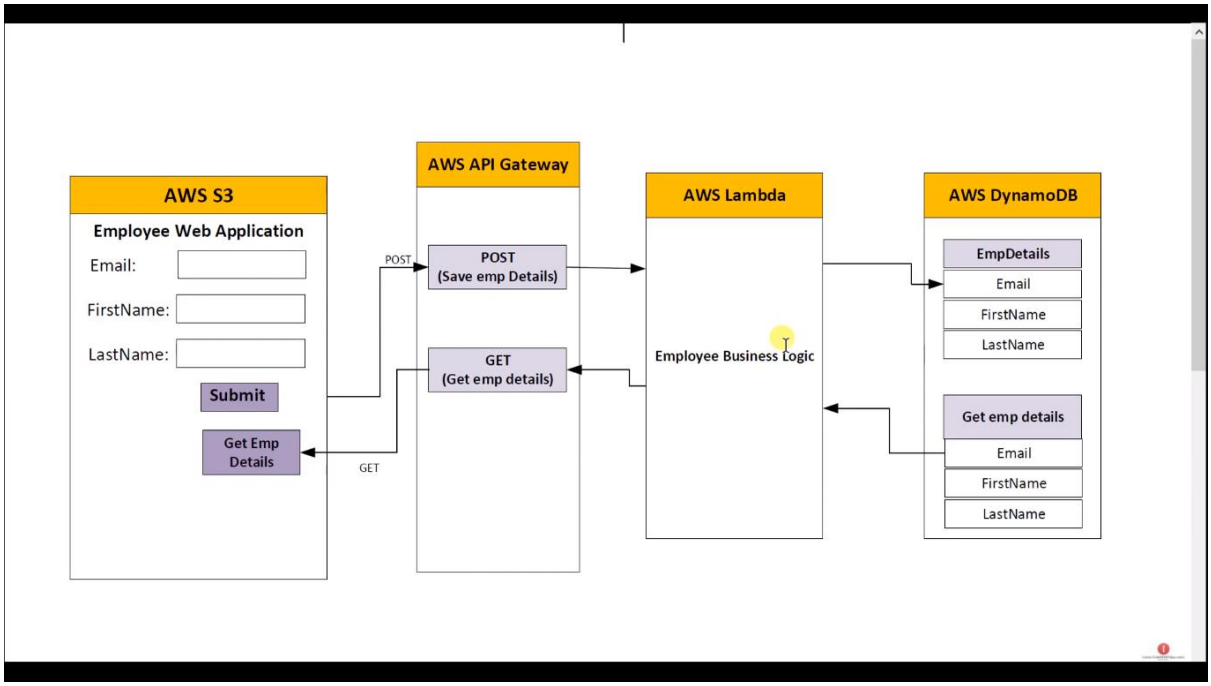
5) A company runs a serverless mobile app that uses Amazon API Gateway, AWS Lambda functions,Amazon Cognito, and Amazon DynamoDB. During large surges in traffic, users report intermittent system failures. The API Gateway API endpoint is returning HTTP status code 502 (Bad Gateway) errors to valid requests. Which solution will resolve this issue?

   A) Increase the concurrency quota for the Lambda functions. Configure Amazon CloudWatch to send notification alerts when the ConcurrentExecutions metric approaches the quota.
   B) Configure notification alerts for the quota of transactions per second on the API Gateway API endpoint. Create a Lambda function that will increase the quota when the quota is reached.
   C) Shard users to Amazon Cognito user pools in multiple AWS Regions to reduce user authentication latency.
   D) Use DynamoDB strongly consistent reads to ensure that the client application always receives the most recent data

In-Lab:

Develop Serverless WebApplication on AWS using POST-METHOD postemployeedeails using email

- **Procedure/Program:**



REFER: https://kluniversityin-my.sharepoint.com/:f:/g/personal/naresh_vurukonda_kluniversity_in/ErJU96i8QlBHuCMkV6HTkq0Bomak4hqMjWcqDqUiIpyMCw?e=kCuiMi

- **Data and Results:**

- **Analysis and Inferences:**

Sample VIVA-VOCE Questions (In-Lab):

1. How would you design a serverless web application on AWS to implement a POST method for posting employee details using email as a parameter?

Ans:

2. What AWS services and components might be involved in creating a serverless web application with a "postemployeedetails" endpoint?

Ans:

3. What is the purpose of using a POST method to add employee details to a serverless web application, and how does it differ from a GET method?

Ans:

4. How can you ensure data validation and security when handling POST requests with employee details, including email, in your serverless web application?

Ans:

5. What are some best practices for implementing serverless web applications with specific POST methods like "postemployeedetails using email" on AWS?

Ans:

Post-Lab:

Combine AWS Batch & Step Functions to create a video processing workflow

- **Procedure/Program:**

- **Data and Results:**

- **Analysis and Inferences:**

Evaluator Remark (if Any):	**Marks Secured: _____out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.

#10. Experiment Title: Build a Serverless Application for Amazon Rekognition Service for given Image using Lambda, S3Bucket,IAM role and API Gateway.

Aim/Objective:

The aim of this project is to develop a serverless application that integrates the Amazon Rekognition service for image recognition.

Description:

Creating a serverless application that harnesses the capabilities of the Amazon Rekognition service is a strategic move in developing image recognition and analysis solutions. The foundation of this project is built on AWS Lambda, a serverless compute service, allowing us to run code in response to events. This serverless approach ensures efficient resource utilization and cost-effectiveness.

The project integrates Amazon S3, a scalable object storage service, to store and retrieve the images to be analyzed. Users can upload their images to an S3 bucket, and the application will automatically trigger AWS Lambda functions for image recognition.

Pre-Requisites:

AWS FREE TIER ACCOUNT, AWS REKOGNITION , S3 BUCKET

Pre-Lab:

1) Which feature of Amazon recognition can assist with saving time?

Ans:

2) Can I use Amazon Rekognition to find cropped versions of an image?

Ans:

3) What benefits does Amazon Rekognition provide?

Ans:

4) A website experiences inconstant traffic, and the database cannot keep up with the write requests during peak traffic times. What AWS Service helps to decouple the web application from the database?
   a) AWS Lambda
   b) Amazon s3
   c) Amazon EFS
   d) Amazon SQS

5) How does image resolution affect the quality of Rekognition Image API results ?

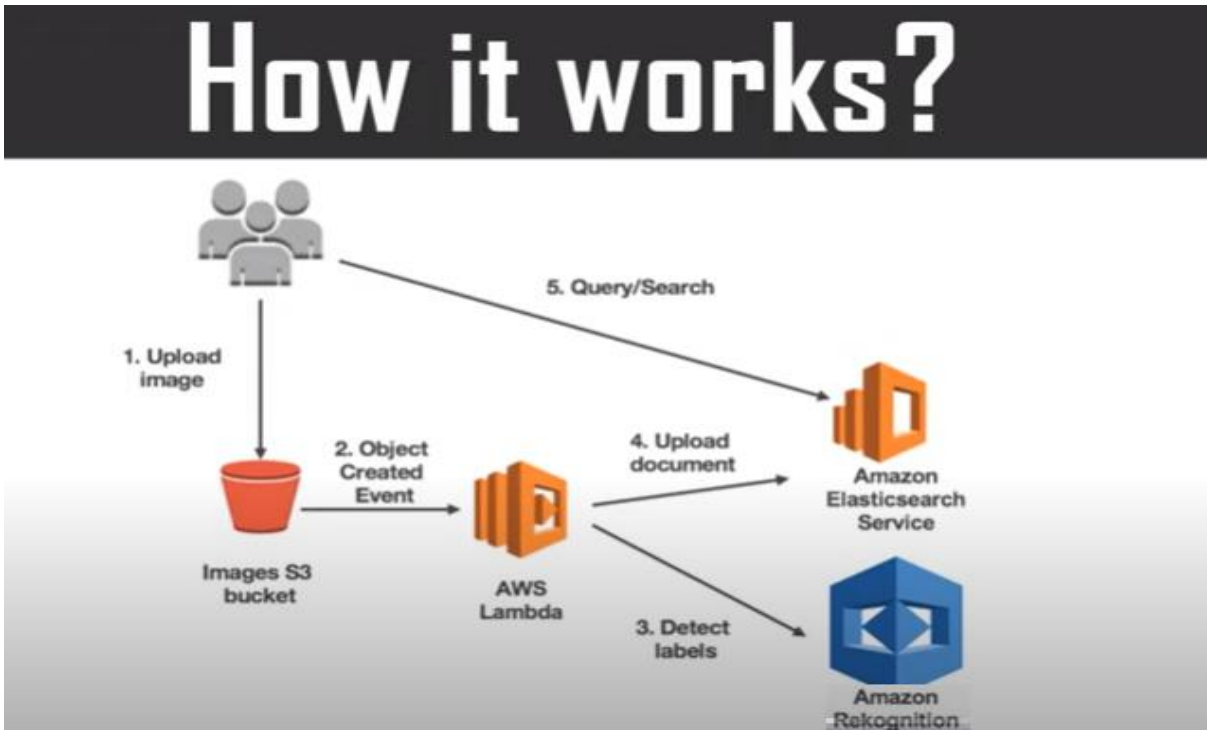
Ans:

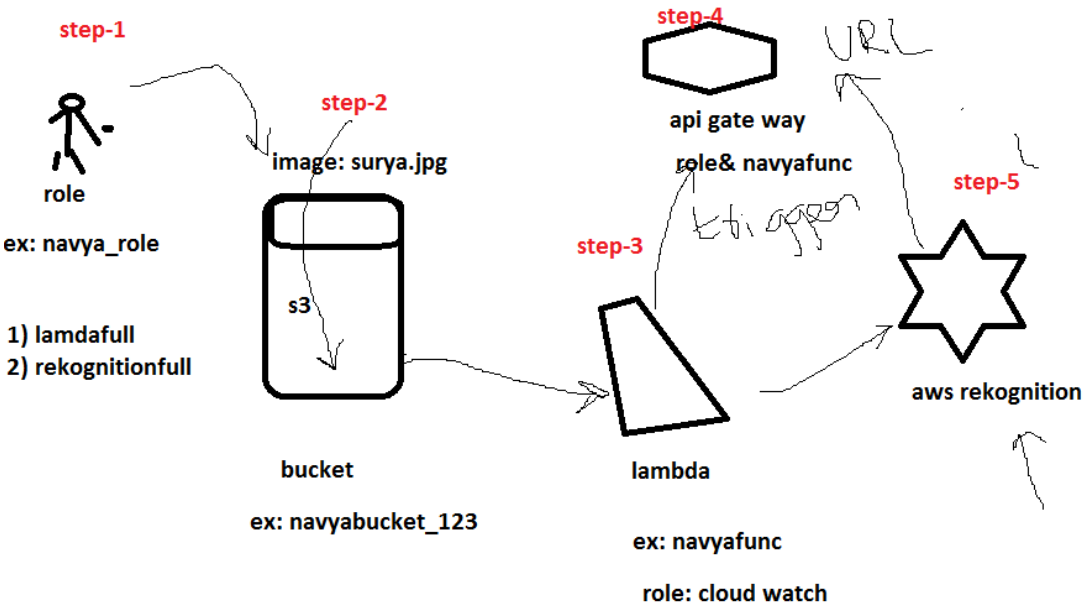
Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

In-Lab:

Build a Serverless Application for Amazon Rekognition Service for given Image using Lambda, S3Bucket,IAM role and API Gateway

- **Procedure/Program:**





REFER:
https://kluniversityin-my.sharepoint.com/:f:/g/personal/naresh_vurukonda_kluniversity_in/EmpPHQ43P1JLkfU9dal9oecBAD9EgFqmeFmRS6E6_V_HiQ?e=a0D5mM

- **Data and Results:**

- **Analysis and Inferences:**

Sample VIVA-VOCE Questions (In-Lab):

1. How can you use Lambda, S3 bucket, IAM role, and API Gateway to create a serverless application for Amazon Rekognition service to analyze a given image?

Ans:

2. What is the role of Amazon Rekognition in a serverless application for image analysis, and how does it integrate with other AWS services?

Ans:

3. What are the key steps involved in setting up an S3 bucket to store and retrieve images for processing by Amazon Rekognition in a serverless application?

Ans:

4. How can you grant the necessary permissions to AWS Lambda functions through IAM roles to access the Rekognition service and the S3 bucket?

Ans:

5. How does API Gateway play a role in providing a RESTful API endpoint for triggering image analysis with Amazon Rekognition in your serverless application?

Ans:

Post-Lab:

Orchestrate Queue-based Microservices with AWS Step Functions and Amazon SQS

- **Procedure/Program:**

- **Data and Results:**

- **Analysis and Inferences:**

Evaluator Remark (if Any):	**Marks Secured: _____out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

#11. Experiment Title: AZURE- Create a WEB APP in Blob Storage and upload images with Azure Functions

Aim/Objective:

To Create a WEB APP in Blob Storage and upload images with Azure Functions

Description:

Creating a web app hosted in Azure Blob Storage and integrating it with Azure Functions for image uploads is a versatile and cost-effective solution for web-based image management and processing.

1. **Azure Blob Storage:** Azure Blob Storage is used as the primary storage solution for the web app. It's highly scalable and cost-effective, making it ideal for storing images and other files. You can organize the data into containers and access it securely using Azure's authentication and authorization mechanisms.

2. **Web App in Blob Storage:** The web app is hosted directly within Azure Blob Storage by serving static HTML, CSS, and JavaScript files from containers. This allows for a serverless, scalable, and low-latency web hosting solution. Users can access the web app via a public URL.

3. **Azure Functions:** Azure Functions are used to handle image uploads. When a user uploads an image through the web app, an Azure Function is triggered. The function can perform various tasks, such as resizing, compressing, or validating the image, before storing it in Blob Storage. This provides a serverless and event-driven approach to image processing.

4. **Scalability and Cost Efficiency:** The combination of Blob Storage and Azure Functions ensures high scalability and cost efficiency. Resources are allocated dynamically based on demand, reducing operational overhead.

5. **Security:** Azure's built-in security features, such as authentication, authorization, and encryption, provide robust protection for both the web app and stored images.

6. **Monitoring and Logging:** Azure provides comprehensive monitoring and logging capabilities, allowing you to track and analyze the performance and usage of your web app and image uploads.

This architecture offers a flexible and cost-efficient way to create a web app for image uploads and processing. It leverages Azure's serverless capabilities, enabling you to focus on the application's functionality while Azure handles the underlying infrastructure. It's an excellent choice for scenarios where you need to manage and process images within a web application efficiently.

Pre-Requisites:

This Section contains the list of Software/Tools or required knowledge (Glossary) to complete the task under the Laboratory Session.

Pre-Lab:

1. What is Azure Blob Storage, and how can it be used for web application hosting?
2. How do Azure Functions enhance the functionality of a web application hosted in Blob Storage?
3. What is the purpose of creating a web app in Blob Storage for image hosting?
4. How can you set up Azure Functions to automate the process of image uploads in the web application?

5. What are the potential advantages of using Azure Functions in combination with Blob Storage for a web app?

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

In-Lab:

- **Procedure/Program:**

- **Data and Results:**

- **Analysis and Inferences:**

Sample VIVA-VOCE Questions (In-Lab):

1. Explain the concept of hosting a web app in Azure Blob Storage. How does it differ from traditional web hosting, and what are the benefits of this approach?

Ans:

2. How can Azure Functions be utilized to enable image uploads in your web app? What is the role of Azure Functions in processing and storing uploaded images?

Ans:

3. Discuss the key steps involved in creating and configuring a web app hosted in Azure Blob Storage. What Azure services and resources are required to set up this architecture?

Ans:

4. Explain how Azure Functions can be triggered when an image is uploaded to the web app. What triggers or bindings are commonly used in Azure Functions for this purpose?

Ans:

5. Walk me through the process of uploading an image to your web app and how Azure Functions handle the storage and processing of the image. What are the typical storage solutions for image files in Azure Blob Storage?

Ans:

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Post-Lab:

1. Image Management with Azure Blob Storage and Web Apps

- **Procedure/Program:**

  This Section is meant for the student to Write the program/Procedure for Experiment

  (Leave at least 2-3 Pages for each Procedure/ Program/ Solution)

- **Data and Results:**

This Section is meant for the students to collect, record the results generated during the Program/Experiment execution. Include instructions on how to present the results, such as creating tables, graphs, or visualizations.

(Leave at least 1 Page for recording the results)

- **Analysis and Inferences:**

This Section is meant for the students to analyse their data, perform calculations Include questions or prompts to encourage critical thinking and interpretation of the data.

(Leave at least 1 Page for recording the analysis and inferences)

Evaluator Remark (if Any):	**Marks Secured: _____out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.

#12. Experiment Title: GCP- Building Serverless Front-End Applications Using Google Cloud Platform

Aim/Objective:

To Build a Serverless Front-End Applications Using Google Cloud Platform Description.

Description:

Building serverless front-end applications using Google Cloud Platform (GCP) is a modern approach that offers scalability, cost-efficiency, and flexibility. This architecture enables developers to create responsive, dynamic, and easily maintainable web applications without the burden of managing traditional server infrastructure.

1. **Google Cloud Functions:** Google Cloud Functions are at the core of this serverless architecture. They allow developers to run code in response to various events, including HTTP requests. For a front-end application, Cloud Functions can serve as serverless back-end logic, handling user requests, authentication, and data processing.

2. **Google Cloud Storage:** Cloud Storage can be used to host static assets, such as HTML, CSS, JavaScript, and media files. It offers high availability, scalability, and low-latency access, making it an excellent choice for front-end resources.

3. **Firebase:** Firebase, which is closely integrated with GCP, provides a real-time database, authentication, and hosting services. It's an ideal complement to a serverless front-end, as it allows developers to build responsive and collaborative applications effortlessly.

4. **Google Cloud Pub/Sub:** For real-time communication or event-driven applications, Cloud Pub/Sub can be employed to facilitate messaging and data synchronization between front-end clients and serverless back-end services.

5. **Scalability and Cost Efficiency:** The serverless nature of this architecture means that resources are automatically scaled to meet demand, which can result in cost savings. Developers only pay for the resources they actually use.

6. **Security and Authentication:** GCP provides robust security features, including identity and access management (IAM), which ensures that data and services are protected. Firebase offers authentication services to secure user access.

7. **Monitoring and Analytics:** GCP offers comprehensive monitoring and analytics tools, such as Google Cloud Monitoring and Google Cloud Trace, allowing developers to track the performance and usage of their applications.

Thus building serverless front-end applications on GCP is an efficient and agile approach. It leverages the power of cloud computing to create highly responsive, scalable, and cost-effective applications while minimizing infrastructure management complexities. This architecture is well-suited for a wide range of web and mobile applications, including e-commerce sites, social platforms, and collaborative tools.

Pre-Requisites:

This Section contains the list of Software/Tools or required knowledge (Glossary) to complete the task under the Laboratory Session.

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Pre-Lab:

1. What is the fundamental concept of serverless computing, and how does it apply to front-end applications?

Ans:

2. What are the key Google Cloud Platform services commonly used to build serverless front-end applications?

Ans:

3. How does serverless architecture in GCP promote cost-efficiency and scalability for front-end applications?

Ans:

4. What are the advantages of building front-end applications that can scale automatically based on demand?

Ans:

5. How does Google Cloud's real-time database service, Firebase, complement serverless front-end applications?

Ans:

In-Lab:

This Section must contain at least 2 Programs/Experiments that links the lecture to be performed during the laboratory Session.

- **Procedure/Program:**

  This Section is meant for the student to Write the program/Procedure for Experiment

(Leave at least 2-3 Pages to record the Procedure/Program)

- **Data and Results:**

  This Section is meant for the students to collect, record the results generated during the Program/Experiment execution. Include instructions on how to present the results, such as creating tables, graphs, or visualizations.

  (Leave at least 1 Page to record the results)

- **Analysis and Inferences:**

  This Section is meant for the students to analyse their data, perform calculations Include questions or prompts to encourage critical thinking and interpretation of the data

  (Leave at least 1 Page for each Program)

Sample VIVA-VOCE Questions (In-Lab):

1. Explain the concept of serverless front-end applications and how it differs from traditional server-based front-end deployments. What are the key advantages of a serverless approach?

Ans:

2. How can you use Google Cloud Functions in the context of serverless front-end applications? Describe the role of Cloud Functions in responding to frontend requests and enabling serverless functionality.

Ans:

3. Discuss the significance of Firebase in building serverless front-end applications. What Firebase features are commonly used to develop and host frontend components?

Ans:

4. Explain how Google Cloud Storage is used to store and serve static assets for a serverless frontend. What benefits does it offer compared to traditional web hosting?

Ans:

5. Walk me through the process of creating and deploying a serverless frontend application using Google Cloud Platform. What are the key steps involved in setting up and hosting the frontend project?

Ans:

Post-Lab:

1. Unlocking Scalable Front-End Capabilities with Serverless on Google Cloud Platform

- **Procedure/Program:**

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

- **Data and Results:**

- **Analysis and Inferences:**

Evaluator Remark (if Any):	**Marks Secured: _____out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.

#13. Experiment Title: Create a Simple Chat Bot Serverless Applicatin With Amazon LEX integrate with Telegram/Whatsapp

Aim/Objective:

To Create a Simple Chat Bot Serverless Applicatin With Amazon LEX integrate with Telegram/Whatsapp

Description:

Creating a simple chatbot serverless application integrated with popular messaging platforms like Facebook, Telegram, and WhatsApp is a strategic move in today's digital landscape. At the heart of this application is Amazon Lex, a robust tool for natural language understanding and processing. Lex enables the development of conversational interfaces and facilitates chatbot dialog management. The serverless architecture underpinning this application leverages AWS Lambda for chatbot logic and AWS API Gateway for seamless interaction with the chosen messaging platforms. This approach not only ensures scalability to accommodate varying workloads but also brings cost-efficiency, as resources are allocated dynamically, reducing operational overhead and infrastructure management complexities. The integration with Facebook, Telegram, and WhatsApp is a pivotal element. By setting up webhooks and APIs specific to each platform, the chatbot can both send and receive messages, making it accessible to users on their preferred messaging apps. This multi-channel access enhances user engagement and convenience, catering to a wider audience.

In designing the chatbot's conversation flow, Amazon Lex empowers developers to craft responses, manage actions, and fulfill user requests. Security measures are integrated for user authentication and authorization, ensuring data protection and privacy. Monitoring and analytics tools, such as AWS CloudWatch and Amazon CloudTrail, provide valuable insights into chatbot performance, usage patterns, and user interactions. This data can be used for continuous improvement and refinement of the chatbot's capabilities.

In essence, building a serverless chatbot application integrated with Facebook, Telegram, and WhatsApp streamlines user interactions, automates responses, and offers a user-friendly, multi-channel communication platform. It's a strategic tool for organizations seeking to provide efficient and accessible services, improve user experiences, and enhance engagement on popular messaging platforms.

Pre-Requisites:

This Section contains the list of Software/Tools or required knowledge (Glossary) to complete the task under the Laboratory Session.

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Pre-Lab:

1. What is the primary purpose of integrating Amazon Lex with Telegram, and WhatsApp?

Ans:

2. How does Amazon Lex facilitate the creation of a serverless chatbot application?

Ans:

3. What are the key advantages of using serverless architecture for a chatbot application?

Ans:

4. What user interactions can be supported by the chatbot when integrated with these messaging platforms?

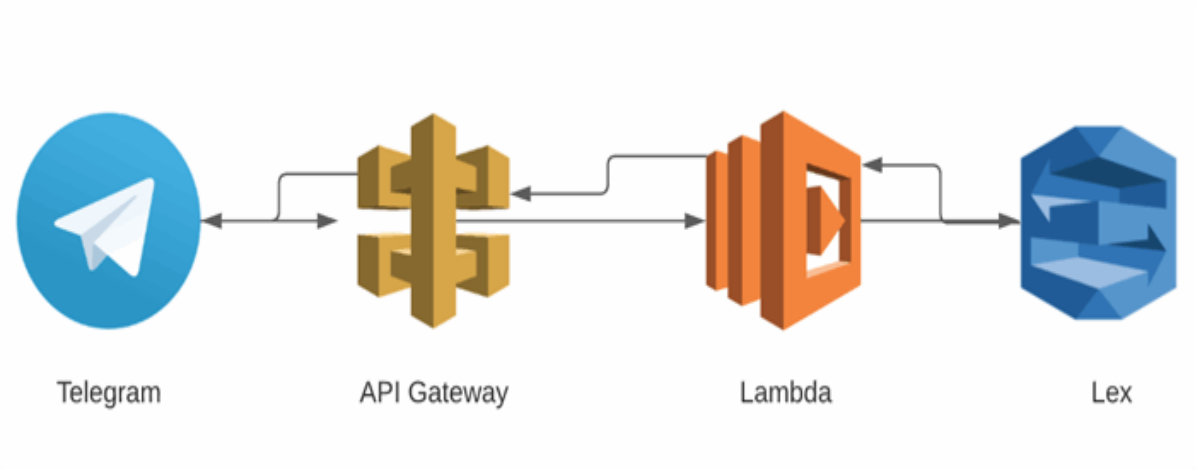
Ans:

5. How can Amazon Lex help improve the user experience in a multi-platform chatbot application?

Ans:

In-Lab:

- **Create a Simple Chat Bot Serverless Applicatin With Amazon LEX integrate with Telegram/Whatsapp**



Post-Lab:

1. Explain the components and architecture of a simple chatbot serverless application using Amazon Lex. What are the core building blocks of this application?

Ans:

2. How can you integrate Amazon Lex with messaging platforms like Telegram, and WhatsApp? What are the common methods for connecting the chatbot to these services?

Ans:

3. Discuss the role of serverless functions in your chatbot application. How are AWS Lambda functions used to interact with Amazon Lex and facilitate communication with the messaging platforms?

Ans:

4. What are the authentication and security considerations when integrating chatbots with messaging platforms like Telegram, and WhatsApp? How do you ensure data privacy and security?

Ans:

5. Walk me through a typical user interaction with your chatbot. How does a user initiate a conversation on one of the messaging platforms, and how is Amazon Lex used to respond to user queries?

Ans:

Evaluator Remark (if Any):	**Marks Secured: _____out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.

#14. Experiment Title: Develop Real time full stack serverless web application using s3, lambda, apigateway, dynamodb

Aim/Objective:

To Develop a Real time full stack serverless web application using s3, lambda, apigateway, dynamodb

Description:

Developing a real-time full-stack serverless web application using Amazon S3, AWS Lambda, API Gateway, and Amazon DynamoDB is a modern approach that offers scalability, cost-efficiency, and rapid development. At the core of this architecture is Amazon S3, which serves as a highly available and durable storage solution for hosting the application's static assets, including HTML, CSS, JavaScript, and media files. This not only ensures low-latency access to these resources but also capitalizes on S3's ability to handle variable loads seamlessly.

AWS Lambda functions play a pivotal role in this setup, as they handle the serverless back-end logic. These functions can be triggered by various events, including HTTP requests from the web application, allowing for dynamic processing and data retrieval from Amazon DynamoDB, a NoSQL database that stores the application's data. DynamoDB is scalable, fast, and capable of handling both real-time and large-scale data with low latency.

API Gateway acts as the interface between the client-side web application and Lambda functions, securely managing user requests and responses. The architecture also leverages the real-time capabilities of AWS WebSocket API to facilitate instant, bidirectional communication between the web application and the serverless back end. This is crucial for real-time features like chat, notifications, or live updates.

This serverless approach to full-stack web application development minimizes operational overhead, ensuring that resources scale automatically based on demand. It allows developers to focus on application functionality and rapidly iterate on features. In addition, AWS security mechanisms, identity and access management, and data encryption are used to safeguard user data and the application's integrity.

Monitoring and analytics services like AWS CloudWatch and AWS X-Ray are deployed to track and optimize application performance, as well as to detect and address issues proactively. This real-time full-stack serverless web application architecture is a powerful choice for projects requiring dynamic, scalable, and cost-effective solutions, particularly those with real-time requirements, such as collaborative tools, dashboards, or interactive web applications.

Pre-Requisites:

This Section contains the list of Software/Tools or required knowledge (Glossary) to complete the task under the Laboratory Session.

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Pre-Lab:

1. What is the role of Amazon S3 in hosting a full stack serverless web application?

Ans:

2. How do AWS Lambda functions facilitate real-time processing in a serverless web application?

Ans:

3. What is the purpose of API Gateway in the architecture of a full stack serverless web application?

Ans:

4. How does DynamoDB store and manage data in a serverless web application?

Ans:

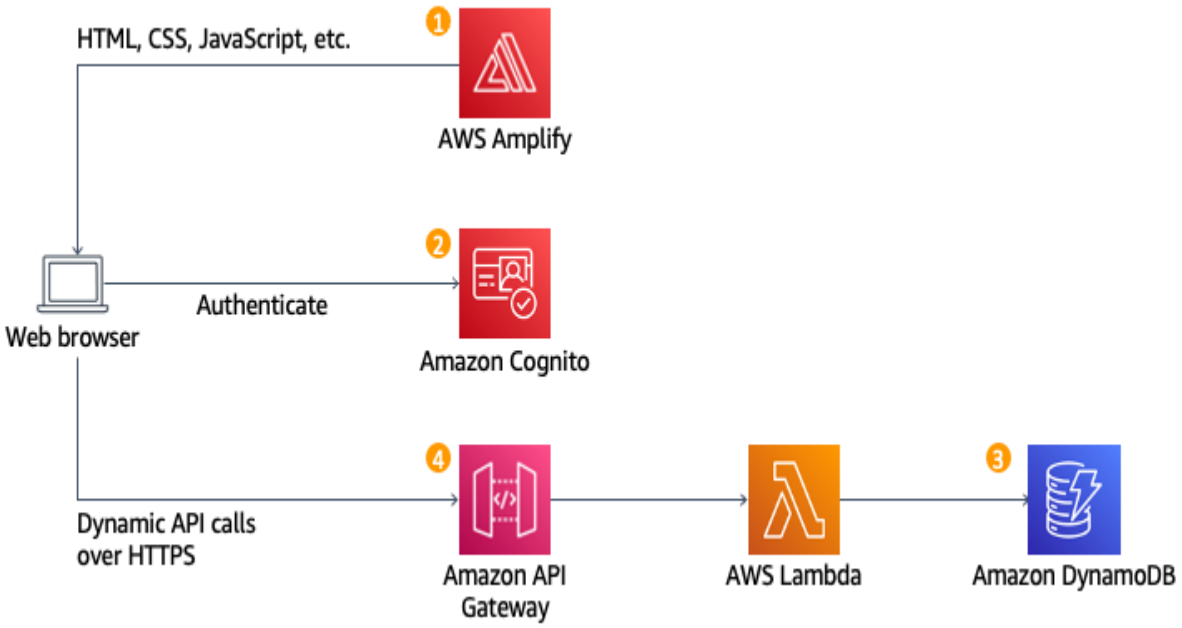
5. What advantages does serverless architecture offer for real-time, full stack web applications?

Ans:

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

In-Lab:

Develop Real time full stack serverless web application using s3, lambda, apigateway, dynamodb



Sample VIVA-VOCE Questions (In-Lab):

1. Explain the components involved in building a full-stack serverless web application using Amazon S3, AWS Lambda, API Gateway, and DynamoDB. What are the roles of these services in the architecture?

Ans:

2. How can Amazon S3 be used to host the frontend of a serverless web application? What are the benefits of using Amazon S3 for this purpose?

Ans:

3. Describe the role of AWS Lambda in a serverless application. How do you implement real-time functionality using Lambda functions, and what triggers these functions?

Ans:

4. Discuss the purpose of Amazon API Gateway in the architecture. How does it help in exposing and managing APIs for your serverless application, including real-time data updates?

Ans:

5. Explain how Amazon DynamoDB is utilized as a data store in your serverless application. How can it handle real-time data, and what features of DynamoDB are beneficial for this purpose?

Ans:

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Post-Lab:

How did the serverless architecture components, including S3, Lambda, API Gateway, and DynamoDB, work together to enable real-time capabilities in the web application, and what are the advantages of this approach over traditional server-based solutions?

- **Procedure/Program:**

  This Section is meant for the student to Write the program/Procedure for Experiment

  (Leave at least 2-3 Pages for each Procedure/ Program/ Solution)

- **Data and Results:**

This Section is meant for the students to collect, record the results generated during the Program/Experiment execution. Include instructions on how to present the results, such as creating tables, graphs, or visualizations.

(Leave at least 1 Page for recording the results)

- **Analysis and Inferences:**

This Section is meant for the students to analyse their data, perform calculations Include questions or prompts to encourage critical thinking and interpretation of the data.

(Leave at least 1 Page for recording the analysis and inferences)

Evaluator Remark (if Any):	**Marks Secured: _____out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.

#15 Experiment Title: Develop Serverless Web Application on AWS Translate using amplify AWS Amplify, AWS IAM, AWS Lambda, AWS API Gateway, AWS Translate.

Aim/Objective:

To Develop a Serverless Web Application on AWS using amplify AWS Amplify, AWS IAM, AWS Lambda, AWS API Gateway, AWS Translate.

Description:

Creating a serverless project for translating languages using AWS services involves several steps. Here's a high-level overview of how you can approach this project:

Pre-Requisites:

Components:

API Gateway: For handling API requests.

Lambda Function: To execute code for language translation.

AWS Translate: For language translation.

S3 Bucket: For storing translation logs or results.

IAM Role: To grant necessary permissions to Lambda and Translate service.

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Pre-Lab:

1. What is the role of AWS Amplify in developing a serverless web application?

Ans:

2. How does the GET method using Amplify retrieve employee details?

Ans:

3. What is the purpose of the POST method in a serverless web application for posting employee details?

Ans:

4. How can email be used effectively as an identifier for employee data management in a serverless web application?

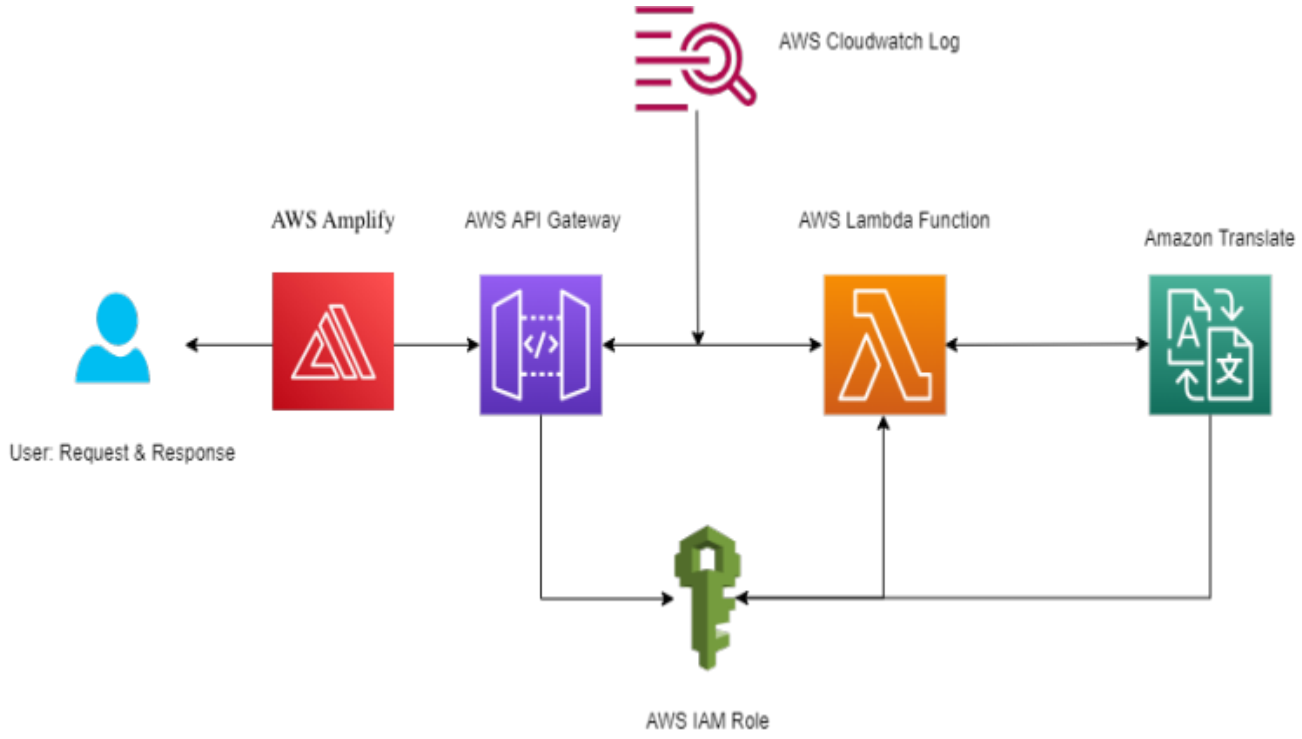
Ans:

5. What advantages does serverless architecture offer in building web applications that handle employee data efficiently?

Ans:

In-Lab:

Develop Serverless Web Application on AWS Translate using amplify AWS Amplify, AWS IAM, AWS Lambda, AWS API Gateway, AWS Translate.



Post-Lab:

1. Explain what AWS Amplify is and how it simplifies the development of web and mobile applications on AWS. What are the core components of AWS Amplify?

Ans:

2. Describe the purpose of implementing the GET and POST methods in a serverless web application. How do these methods enable you to post employee details using email?

Ans:

3. How can AWS Amplify be used to create the frontend of a web application? What are the steps involved in setting up the frontend project using AWS Amplify?

Ans:

4. Walk me through the process of creating and deploying serverless backend functions using AWS Lambda for handling GET and POST requests in an AWS Amplify project.

Ans:

5. Explain the role of Amazon DynamoDB and AWS Cognito in your serverless web application. How do these services enable data storage and user authentication, respectively?

Ans:

Evaluator Remark (if Any):	**Marks Secured: _____out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.

#16. Experiment Title: Build serverless application using Kubernetes

Aim/Objective:

To Build a serverless application using Kubernetes.

Description:

Building a serverless application using Kubernetes may seem counterintuitive, as Kubernetes is typically associated with container orchestration, but it is possible to leverage Kubernetes in a serverless context. Kubernetes can be adapted to serve as a platform for running serverless workloads, offering a more managed and scalable environment for applications without the need for traditional server management.

To create a serverless application with Kubernetes, you can employ serverless frameworks like Knative or Kubeless, which extend Kubernetes to enable event-driven, auto-scaling, and pay-per-use computing. These frameworks provide abstractions and automated scaling, allowing developers to focus on code and application logic rather than infrastructure management.

Kubernetes' ability to manage and scale containers makes it well-suited for running serverless functions packaged in containers. The serverless functions can be deployed as pods within the Kubernetes cluster and triggered by events, such as HTTP requests or messages from a message queue.

By combining the scalability, resource efficiency, and event-driven capabilities of Kubernetes with serverless frameworks, developers can achieve serverless-like benefits while maintaining the flexibility to manage more complex workloads within a familiar Kubernetes environment. This approach bridges the gap between traditional containerized applications and serverless computing, providing a unique solution for different application requirements.

Pre-Requisites:

This Section contains the list of Software/Tools or required knowledge (Glossary) to complete the task under the Laboratory Session.

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Pre-Lab:

1. How does Kubernetes traditionally differ from serverless computing?

Ans:

2. What is Knative, and how can it be used for serverless workloads in Kubernetes?

Ans:

3. What are the benefits of integrating serverless capabilities with Kubernetes?

Ans:

4. How does Kubernetes offer scalability and resource management for serverless applications?

Ans:

5. What challenges might you encounter when building serverless applications in a Kubernetes environment?

Ans:

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

In-Lab:

Build serverless application using Kubernetes

Post-Lab:

1. Explain what serverless computing is and how it differs from traditional server-based application deployments. How does serverless fit into the Kubernetes ecosystem?

Ans:

2. How does Kubernetes support the deployment of serverless functions or applications? Describe the key components or concepts in Kubernetes that are relevant to building serverless applications.

Ans:

3. What is a Kubernetes-based serverless framework, and how does it simplify the deployment and scaling of serverless functions? Can you provide an example of such a framework?

Ans:

4. Walk me through the process of creating and deploying a serverless function using a Kubernetes-based serverless framework. What are the essential steps and Kubernetes resources involved?

Ans:

5. Discuss a real-world use case where building a serverless application using Kubernetes would be advantageous. How does Kubernetes enable automatic scaling and event-driven behaviors for serverless functions in this scenario?

Ans:

Evaluator Remark (if Any):	**Marks Secured: _____out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.

#17. Experiment Title: Create serverless application using Openshift container platform web console

Aim/Objective:

To Create a serverless application using Openshift container platform web console.

Description:

Creating a serverless application using the OpenShift Container Platform web console may sound unconventional, as OpenShift is primarily known for container orchestration. However, OpenShift's flexibility and extensibility make it possible to build serverless applications using its web console. OpenShift, based on Kubernetes, can host serverless workloads within its cluster, providing an environment that simplifies application management and scales resources automatically.

To create a serverless application in OpenShift, developers can leverage OpenShift Serverless, an add-on for the platform. OpenShift Serverless offers a serverless framework known as Knative, which can be used to deploy and manage serverless functions within OpenShift clusters. The OpenShift web console provides a user-friendly interface to define and deploy serverless functions, making it accessible for developers and operators.

Serverless applications built on OpenShift can benefit from the platform's resource allocation, scaling, and ease of deployment. Functions can be triggered by HTTP requests or events, responding to user interactions and system events without requiring manual provisioning or scaling of server instances. The combination of OpenShift and serverless computing offers an attractive solution for applications that need to efficiently handle dynamic workloads, while the web console simplifies the management and monitoring of these serverless functions, providing a more intuitive and streamlined development experience.

Pre-Requisites:

This Section contains the list of Software/Tools or required knowledge (Glossary) to complete the task under the Laboratory Session.

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Pre-Lab:

1. How does Openshift Container Platform typically relate to container orchestration?

Ans:

2. What is Openshift Serverless, and how does it extend the platform for serverless workloads?

Ans:

3. How does the web console of Openshift Container Platform streamline the creation of serverless applications?

Ans:

4. What are the advantages of building serverless applications in Openshift in terms of scalability and resource management?

Ans:

5. What security considerations are important when creating serverless applications using Openshift?

Ans:

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

In-Lab:

Create serverless application using Openshift container platform web console

Sample VIVA-VOCE Questions (In-Lab):

1. What is serverless computing, and how does it differ from traditional server-based application deployments?

Ans:

2. Explain the role of the OpenShift Container Platform in the context of serverless applications. How does it simplify the deployment and scaling of serverless functions?

Ans:

3. How can you use the OpenShift web console to create and manage serverless applications? Walk me through the steps to deploy a serverless function using the web console.

Ans:

4. What is Knative, and how does it relate to serverless applications in OpenShift? How does it enable automatic scaling and event-driven behaviors in serverless functions?

Ans:

5. Discuss a real-world use case where a serverless application deployed on OpenShift could provide significant benefits. How does the platform support this use case, and what are the advantages of using OpenShift for serverless computing?

Ans:

Evaluator Remark (if Any):	**Marks Secured: _____out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.

#18. Experiment Title: Real time bot integration with facebook Messenger

Aim/Objective:

To integrate a bot in Real time with facebook Messenger

Description:

Real-time bot integration with Facebook Messenger using AWS is a dynamic and efficient approach to enhance user engagement and automate interactions across these popular messaging platforms. AWS, with its extensive suite of cloud services, offers a robust infrastructure for deploying and managing real-time chatbots, allowing organizations to deliver timely and personalized experiences to their users.

To implement real-time bot integration with AWS, developers can leverage services like Amazon Lex, AWS Lambda, and API Gateway. Amazon Lex serves as the core for building chatbots, providing natural language understanding and dialogue management capabilities. AWS Lambda, a serverless compute service, executes the chatbot's business logic, ensuring efficient and scalable processing of user interactions. API Gateway facilitates secure and efficient communication between the messaging platforms and the chatbot, managing requests and responses.

Real-time capabilities are crucial for prompt responses and engaging user experiences. AWS provides scalable and resilient infrastructure to handle the varying workloads associated with real-time communication on these messaging platforms. AWS IoT can also be utilized to facilitate real-time communication and notification features, ensuring users receive instant updates, alerts, and notifications through their preferred messaging apps.

Security and data privacy are paramount in real-time bot integration, and AWS offers robust identity and access management (IAM) and encryption mechanisms to safeguard sensitive information and protect user data. With real-time bot integration on Facebook Messenger using AWS, organizations can deliver dynamic and responsive services, streamline communication, and automate interactions, ultimately enhancing user satisfaction and engagement across these popular messaging platforms.

Pre-Requisites:

This Section contains the list of Software/Tools or required knowledge (Glossary) to complete the task under the Laboratory Session.

Pre-Lab:

1. How can real-time bot integration enhance user engagement on messaging platforms?

Ans:

2. What are the key features and capabilities provided by Facebook, Messenger for bot integration?

Ans:

3. What are the potential benefits of real-time interactions with users in a bot integration scenario?

Ans:

4. How do real-time chatbots enhance the user experience and responsiveness on messaging apps?

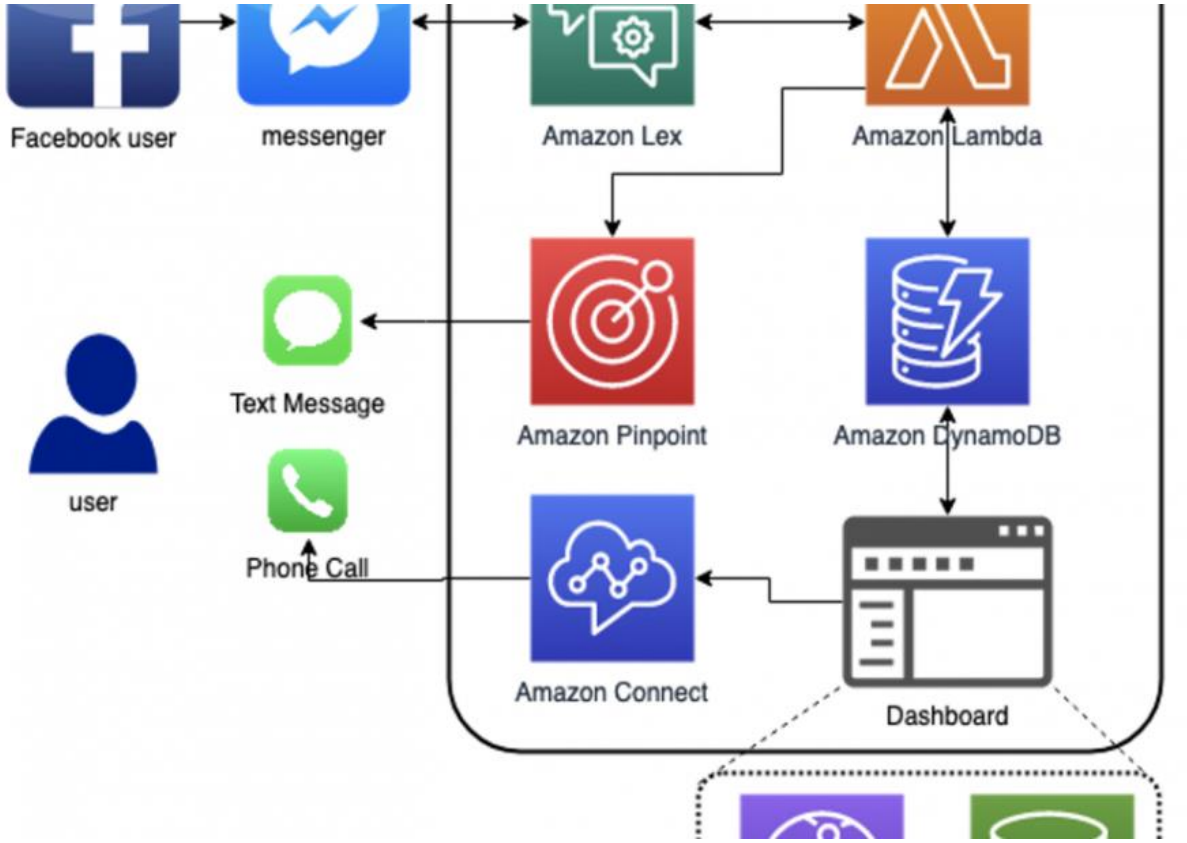
Ans:

5. What security measures should be considered when integrating bots with messaging platforms like Facebook Messenger?

Ans:

In-Lab:

Real time bot integration with facebook Messenger



Post-Lab:

1. What is a real-time bot integration, and why is it important for messaging platforms like Facebook, Telegram, and WhatsApp?

Ans:

2. Explain the fundamental differences in integrating a bot with Facebook Messenger, Telegram, and WhatsApp. What are the common challenges in each case?

Ans:

3. How do you use webhooks or callbacks to achieve real-time interactions with bots on these messaging platforms? Can you provide an example of how a webhook works in this context?

Ans:

4. Discuss the importance of authentication and security in real-time bot integrations. What measures should be taken to protect user data and ensure the bot's identity?

Ans:

5. What are some typical use cases for real-time bot integration on these platforms? Can you provide examples of how businesses or organizations can benefit from such integrations?

Ans:

Evaluator Remark (if Any):	**Marks Secured: _____out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.

#19. Experiment Title: develop serverless voting web application

Aim/Objective:

To develop a serverless voting web application

Description:

This Section must contain detailed information pertaining to the Aim/Objective of the Laboratory Session

Pre-Requisites:

This Section contains the list of Software/Tools or required knowledge (Glossary) to complete the task under the Laboratory Session.

Pre-Lab:

1. What is the main objective of a serverless voting web application?

Ans:

2. How does serverless architecture improve the scalability of a voting web application?

Ans:

3. What are the core components required for building a serverless voting app?

Ans:

4. How can data security and integrity be ensured in a serverless voting application?

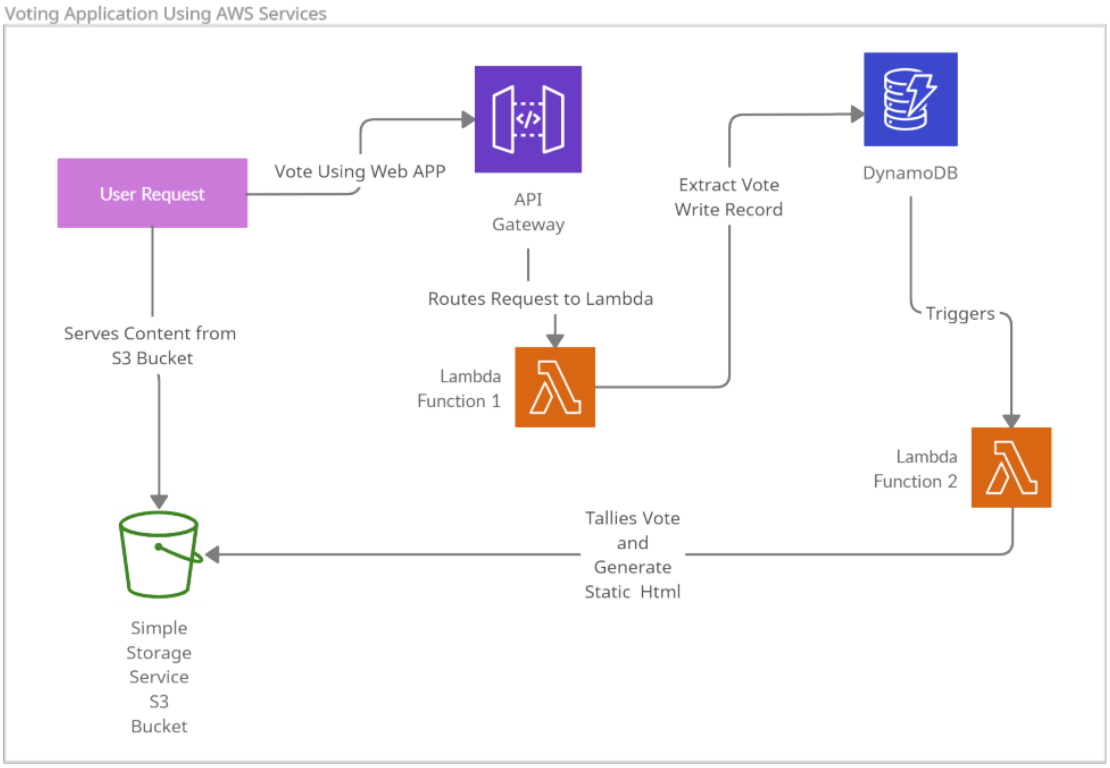
Ans:

5. What advantages does serverless technology offer for rapid development and cost-efficiency in a voting application?

Ans:

In-Lab:

Develop serverless voting web application



Post-Lab:

1. What is a serverless voting web application, and why might it be a suitable choice for such an application compared to traditional architectures?

Ans:

2. Explain the key components of a serverless voting web application, including the frontend, backend, and data storage.

Ans:

3. How can you use HTTP GET and POST methods in a serverless voting application? Describe their roles in retrieving and submitting votes.

Ans:

4. Discuss the considerations for securing a serverless voting web application. How can you ensure the integrity and confidentiality of votes?

Ans:

5. What are some advantages of using serverless computing in a voting application, and what challenges might you face when designing and deploying such an application?

Ans:

Evaluator Remark (if Any):	**Marks Secured: _____out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.

#20. Experiment Title: develop serverless web application using get and post method for real time data

Aim/Objective:

To develop a serverless web application using get and post method for real time data

Description:

This Section must contain detailed information pertaining to the Aim/Objective of the Laboratory Session

Pre-Requisites:

This Section contains the list of Software/Tools or required knowledge (Glossary) to complete the task under the Laboratory Session.

Pre-Lab:

1. How can a serverless web application facilitate real-time data interactions?

Ans:

2. What is the role of the GET method in retrieving real-time data in a serverless web application?

Ans:

3. How does the POST method enable real-time data submission in a serverless web application?

Ans:

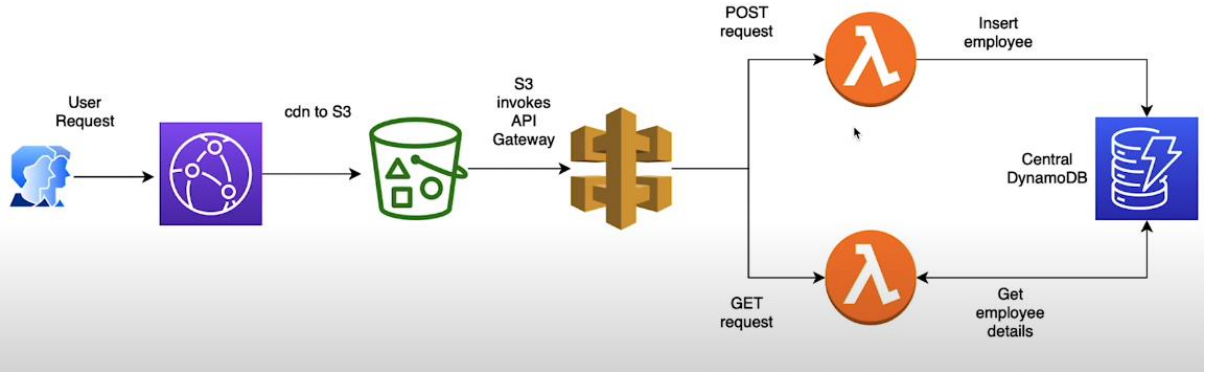
4. What are the primary use cases for real-time data processing in serverless web applications?

Ans:

5. What benefits does serverless architecture bring to the development of real-time data-driven applications?

Ans:

In-Lab: Develop serverless web application using get and post method for real time data



Post-Lab:

1. What is a serverless web application, and how does it differ from a traditional web application architecture?

Ans:

2. Explain the purpose of using HTTP GET and POST methods in a web application. How do these methods differ in terms of data transfer?

Ans:

3. What is the role of a serverless function (e.g., AWS Lambda or Azure Functions) in a serverless web application, and how is it triggered by HTTP requests?

Ans:

4. How can you achieve real-time data updates in a serverless web application? What technologies or protocols are commonly used for real-time communication?

Ans:

5. What is the significance of using an API Gateway in a serverless web application, and how does it relate to handling HTTP requests like GET and POST?

Ans:

Evaluator Remark (if Any):	**Marks Secured: _____out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.