



BLACKBUCKS INTERNSHIP REPORT

HOSTING SERVERLESS STATIC AND DYNAMIC WEBSITES USING AWS

SUBMITTED BY

K Ganesh 20B91A5725

B Shanmukh 20B91A5708

K Vikram 20B91A5726

UNDER THE GUIDANCE OF MR. AASHU DEV

**B.Tech , AWS Accredited Solution Architect (2x) , AWS
Academy educator**

Blackbuck Engineers Pvt. Ltd

**Road No 36, Jubilee Hills,
Hyderabad**

BLACKBUCK INTERNSHIP WORK

Team Members:

K Ganesh (20B91A5725)

B Shanmukh (20B91A5708)

K Vikram (20B91A5726)

Title:

Using the Cloud shell repository, we pull the repository file from the code commit to cloud9 and then push it to third - party repository Git Hub.

Abstract:

Usually, To host dynamic or a static website we need a server. But with using AWS we can host them without a physical server. All we need to know is how to use AWS services that would let us host a static or dynamic website publicly. The websites that are hosted through AWS are called as serverless websites that means the person that's hosting doesn't need to invest in a physical server they can just use AWS services.

Contents

Team Members -	4
Project Aim.....	5
Project tasks	5
Overview about AWS	6
Project AWS architecture.....	7
Hosting a static website using AWS	8
AWS S3	8
Process for hosting a static website through S3	12
Static website code-	15
Hosting a dynamic website using AWS	17
AWS Amplify	17
AWS Lambda	20
Amazon API Gateway.....	24
Amazon DynamoDB	26
AWS IAM (Identity and Access Management).....	30
Process for hosting a dynamic website in AWS	32
Codes used-	47
Conclusion.....	54

Team Members-

Names	Reg No
B Shanmukh	20B91A5708
K Ganesh	20B91A5725
K Vikram	20B91A5726

Project Aim

Our aim in this project is to host serverless static and dynamic websites using AWS.

Project tasks

Hosting a serverless static website using AWS

- Needed services
 - S3 buckets

Hosting a serverless dynamic website using AWS

- Needed services
 - AWS Amplify
 - AWS Lambda
 - Amazon Api Gateway
 - Amazon DynamoDB
 - AWS IAM

Overview about AWS

Amazon Web Services, AWS, uses a lot of acronyms and codenames for different services that can be confusing when being new to the service. The list below gives a quick summary of the services we use and discuss in the wiki for the Relativity Server in the Cloud feature.

Services

EC2- short for Elastic Cloud Computing, lets you run virtual machines, essentially full virtual server computers to run your software on. We use it to run Relativity Server. EC2 instances should be treated as "discordable" and a well-designed system will not store any data on them.

S3- short for Simple Storage Service, lets you create "buckets" where you can store data safely and reliably. Data stored in S3 can be publicly available or private to yourself, and S3 includes data safety guarantees and redundancy so that you can (within reason) rely on data in S3 not being lost due to hardware failures. We use S3, among other things, to store Relativity server configurations.

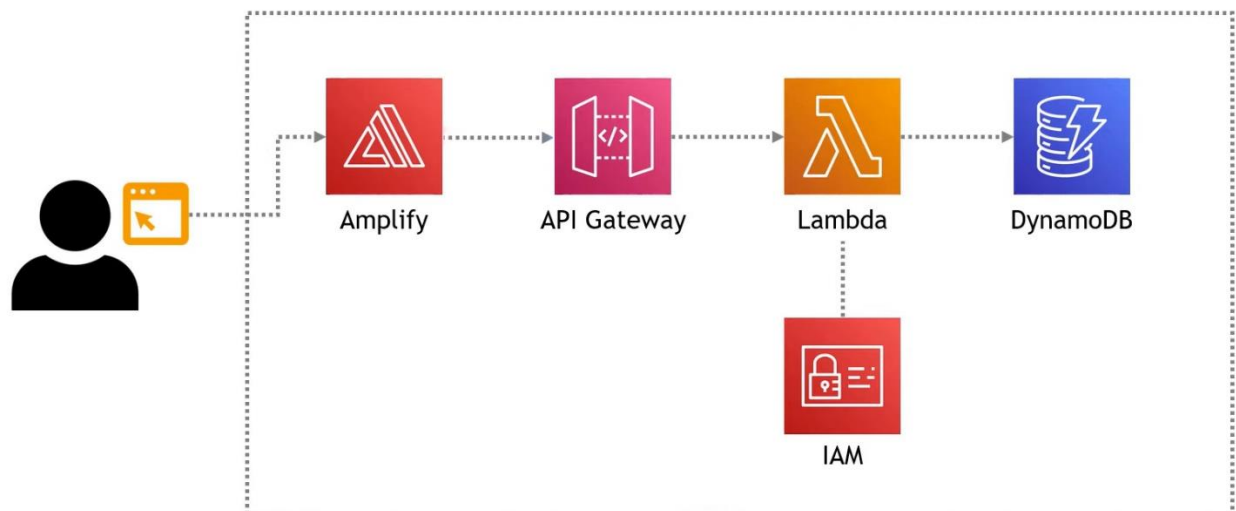
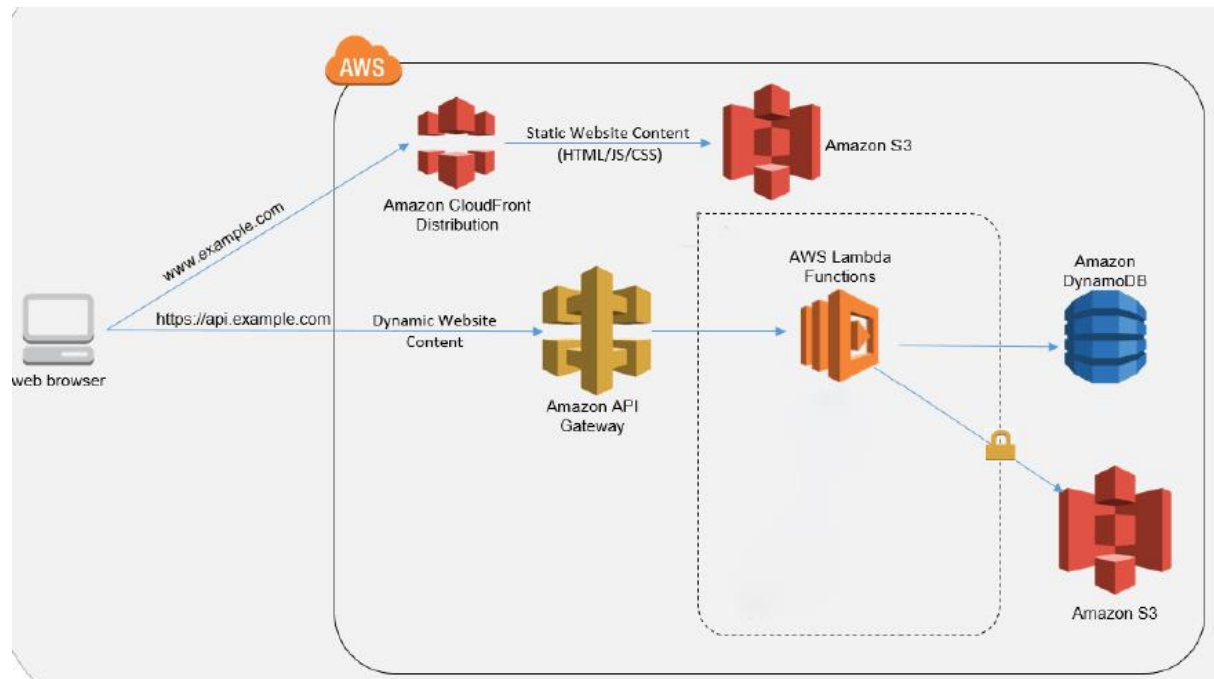
RDS- short for Relational Database Service, is a way to host databases such as Microsoft SQL Server, Oracle, MySQL or PostgreSQL on AWS, without maintaining EC2 machines to host database engine yourself. RDS supports automatic data replication and failover to backup servers, as well as automatic backups to S3.

DynamoDB- It is a no-SQL database system with built-in data protection across availability zones.

IAM- short for Identity and Access Management, is AWS's portal for managing users and access to other AWS services.

AMI- short for AWS Machine Image, is the name for a pre-configured image that can be used to boot up EC2 instances. We provide a ready-configured AMI with Relativity Server that only needs to be booted up and is ready to work, without any additional installation of Relativity server or prerequisites (see Setting up Relativity Server on AWS).

Project AWS architecture



Hosting a static website using AWS

To host a static website through AWS, we need to use S3 bucket.

AWS S3

Amazon Simple Storage Service (Amazon S3) is an object storage service that offers industry-leading scalability, data availability, security, and performance. Customers of all sizes and industries can use Amazon S3 to store and protect any amount of data for a range of use cases, such as data lakes, websites, mobile applications, backup and restore, archive, enterprise applications, IoT devices, and big data analytics. Amazon S3 provides management features so that you can optimize, organize, and configure access to your data to meet your specific business, organizational, and compliance requirements.

Features of Amazon S3

Storage classes

Amazon S3 offers a range of storage classes designed for different use cases. For example, you can store mission-critical production data in S3 Standard for frequent access, save costs by storing infrequently accessed data in S3 Standard-IA or S3 One Zone-IA, and archive data at the lowest costs in S3 Glacier Instant Retrieval, S3 Glacier Flexible Retrieval, and S3 Glacier Deep Archive.

You can store data with changing or unknown access patterns in S3 Intelligent-Tiering, which optimizes storage costs by automatically moving your data between four access tiers when your access patterns change. These four access tiers include two low-latency access tiers optimized for frequent and infrequent access, and two opt-in archive access tiers designed for asynchronous access for rarely accessed data.

For more information, see [Using Amazon S3 storage classes](#). For more information about S3 Glacier Flexible Retrieval, see the [Amazon S3 Glacier Developer Guide](#).

Storage management

Amazon S3 has storage management features that you can use to manage costs, meet regulatory requirements, reduce latency, and save multiple distinct copies of your data for compliance requirements.

- S3 Lifecycle – Configure a lifecycle configuration to manage your objects and store them cost effectively throughout their lifecycle. You can transition objects to other S3 storage classes or expire objects that reach the end of their lifetimes.
- S3 Object Lock – Prevent Amazon S3 objects from being deleted or overwritten for a fixed amount of time or indefinitely. You can use Object Lock to help meet regulatory requirements that require *write-once-read-many (WORM)* storage or to simply add another layer of protection against object changes and deletions.

- S3 Replication – Replicate objects and their respective metadata and object tags to one or more destination buckets in the same or different AWS Regions for reduced latency, compliance, security, and other use cases.
- S3 Batch Operations – Manage billions of objects at scale with a single S3 API request or a few clicks in the Amazon S3 console. You can use Batch Operations to perform operations such as **Copy**, **Invoke AWS Lambda function**, and **Restore** on millions or billions of objects.

Access management and security

Amazon S3 provides features for auditing and managing access to your buckets and objects. By default, S3 buckets and the objects in them are private. You have access only to the S3 resources that you create. To grant granular resource permissions that support your specific use case or to audit the permissions of your Amazon S3 resources, you can use the following features.

- S3 Block Public Access – Block public access to S3 buckets and objects. By default, Block Public Access settings are turned on at the bucket level. We recommend that you keep all Block Public Access settings enabled unless you know that you need to turn off one or more of them for your specific use case.
- AWS Identity and Access Management (IAM) – IAM is a web service that helps you securely control access to AWS resources, including your Amazon S3 resources. With IAM, you can centrally manage permissions that control which AWS resources users can access. You use IAM to control who is authenticated (signed in) and authorized (has permissions) to use resources.
- Bucket policies – Use IAM-based policy language to configure resource-based permissions for your S3 buckets and the objects in them.
- Amazon S3 access points – Configure named network endpoints with dedicated access policies to manage data access at scale for shared datasets in Amazon S3.
- Access control lists (ACLs) – Grant read and write permissions for individual buckets and objects to authorized users. As a general rule, we recommend using S3 resource-based policies (bucket policies and access point policies) or IAM user policies for access control instead of ACLs. Policies are a simplified and more flexible access control option. With bucket policies and access point policies, you can define rules that apply broadly across all requests to your Amazon S3 resources.
- S3 Object Ownership – Take ownership of every object in your bucket, simplifying access management for data stored in Amazon S3. S3 Object Ownership is an Amazon S3 bucket-level setting that you can use to disable or enable ACLs. By default, ACLs are disabled. With ACLs disabled, the bucket owner owns all the objects in the bucket and manages access to data exclusively by using access-management policies.

- IAM Access Analyzer for S3 – Evaluate and monitor your S3 bucket access policies, ensuring that the policies provide only the intended access to your S3 resources.

Data processing

To transform data and trigger workflows to automate a variety of other processing activities at scale, you can use the following features.

- S3 Object Lambda – Add your own code to S3 GET, HEAD, and LIST requests to modify and process data as it is returned to an application. Filter rows, dynamically resize images, redact confidential data, and much more.
- Event notifications – Trigger workflows that use Amazon Simple Notification Service (Amazon SNS), Amazon Simple Queue Service (Amazon SQS), and AWS Lambda when a change is made to your S3 resources.

Storage logging and monitoring

Amazon S3 provides logging and monitoring tools that you can use to monitor and control how your Amazon S3 resources are being used. For more information, see [Monitoring tools](#).

Automated monitoring tools

- Amazon CloudWatch metrics for Amazon S3 – Track the operational health of your S3 resources and configure billing alerts when estimated charges reach a user-defined threshold.
- AWS CloudTrail – Record actions taken by a user, a role, or an AWS service in Amazon S3. CloudTrail logs provide you with detailed API tracking for S3 bucket-level and object-level operations.

Manual monitoring tools

- Server access logging – Get detailed records for the requests that are made to a bucket. You can use server access logs for many use cases, such as conducting security and access audits, learning about your customer base, and understanding your Amazon S3 bill.
- AWS Trusted Advisor – Evaluate your account by using AWS best practice checks to identify ways to optimize your AWS infrastructure, improve security and performance, reduce costs, and monitor service quotas. You can then follow the recommendations to optimize your services and resources.

Analytics and insights

Amazon S3 offers features to help you gain visibility into your storage usage, which empowers you to better understand, analyze, and optimize your storage at scale.

- Amazon S3 Storage Lens – Understand, analyze, and optimize your storage. S3 Storage Lens provides 29+ usage and activity metrics and interactive dashboards to

aggregate data for your entire organization, specific accounts, AWS Regions, buckets, or prefixes.

- Storage Class Analysis – Analyze storage access patterns to decide when it's time to move data to a more cost-effective storage class.
- S3 Inventory with Inventory reports – Audit and report on objects and their corresponding metadata and configure other Amazon S3 features to take action in Inventory reports. For example, you can report on the replication and encryption status of your objects. For a list of all the metadata available for each object in Inventory reports.

Strong consistency

Amazon S3 provides strong read-after-write consistency for PUT and DELETE requests of objects in your Amazon S3 bucket in all AWS Regions. This behavior applies to both writes of new objects as well as PUT requests that overwrite existing objects and DELETE requests. In addition, read operations on Amazon S3 Select, Amazon S3 access control lists (ACLs), Amazon S3 Object Tags, and object metadata (for example, the HEAD object) are strongly consistent. For more information.

How Amazon S3 works

Amazon S3 is an object storage service that stores data as objects within buckets. An *object* is a file and any metadata that describes the file. A *bucket* is a container for objects.

To store your data in Amazon S3, you first create a bucket and specify a bucket name and AWS Region. Then, you upload your data to that bucket as objects in Amazon S3. Each object has a *key* (or *key name*), which is the unique identifier for the object within the bucket.

S3 provides features that you can configure to support your specific use case. For example, you can use S3 Versioning to keep multiple versions of an object in the same bucket, which allows you to restore objects that are accidentally deleted or overwritten.

Buckets and the objects in them are private and can be accessed only if you explicitly grant access permissions. You can use bucket policies, AWS Identity and Access Management (IAM) policies, access control lists (ACLs), and S3 Access Points to manage access.

Buckets

A bucket is a container for objects stored in Amazon S3. You can store any number of objects in a bucket and can have up to 100 buckets in your account.

Every object is contained in a bucket. For example, if the object named photos/puppy.jpg is stored in the DOC-EXAMPLE-BUCKET bucket in the US West (Oregon) Region, then it is addressable by using the URL <https://DOC-EXAMPLE-BUCKET.s3.us-west-2.amazonaws.com/photos/puppy.jpg>.

When you create a bucket, you enter a bucket name and choose the AWS Region where the bucket will reside. After you create a bucket, you cannot change the name of the bucket or its Region. Bucket names must follow the bucket naming rules. You can also configure a bucket to use S3 Versioning or other storage management features.

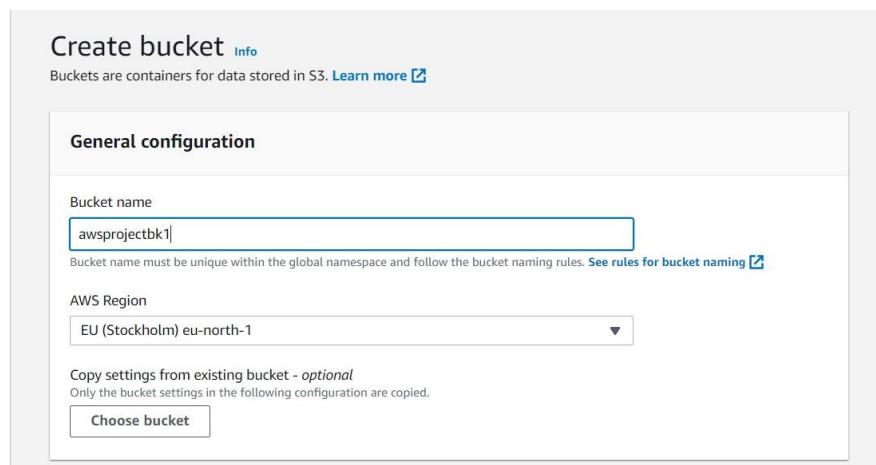
Buckets also:

- Organize the Amazon S3 namespace at the highest level.
- Identify the account responsible for storage and data transfer charges.
- Provide access control options, such as bucket policies, access control lists (ACLs), and S3 Access Points, that you can use to manage access to your Amazon S3 resources.
- Serve as the unit of aggregation for usage reporting.

Process for hosting a static website through S3

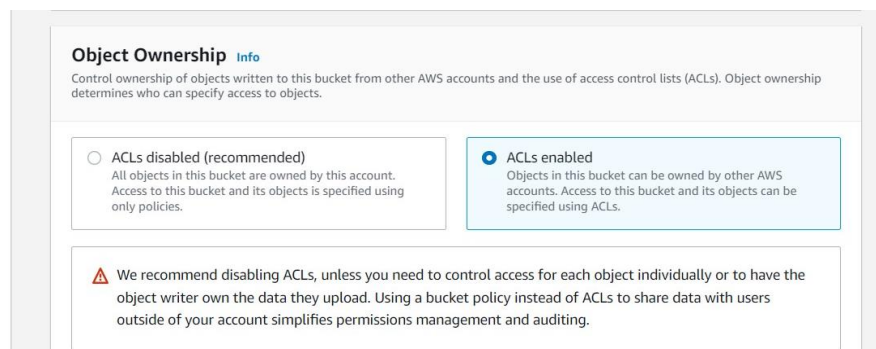
1. Creating a S3 Bucket

Bucket creation follows as



The screenshot shows the 'Create bucket' page in the AWS Management Console. At the top, it says 'Create bucket' with an 'Info' link. Below that, a note states 'Buckets are containers for data stored in S3. [Learn more](#)'. The main section is titled 'General configuration'. It contains a 'Bucket name' field with the text 'awsprojectbk1' and a note: 'Bucket name must be unique within the global namespace and follow the bucket naming rules. [See rules for bucket naming](#)'. Below this is the 'AWS Region' dropdown menu, currently set to 'EU (Stockholm) eu-north-1'. Further down, there is a section for 'Copy settings from existing bucket - optional' with a note: 'Only the bucket settings in the following configuration are copied.' and a 'Choose bucket' button.

Changing ownership of objects for global access



The screenshot shows the 'Object Ownership' page in the AWS Management Console. It has an 'Info' link and a description: 'Control ownership of objects written to this bucket from other AWS accounts and the use of access control lists (ACLs). Object ownership determines who can specify access to objects.' There are two radio button options: 'ACLs disabled (recommended)' and 'ACLs enabled'. The 'ACLs disabled' option is selected. Below these options is a warning box with a triangle icon and the text: 'We recommend disabling ACLs, unless you need to control access for each object individually or to have the object writer own the data they upload. Using a bucket policy instead of ACLs to share data with users outside of your account simplifies permissions management and auditing.'

Giving public access for the bucket

Block Public Access settings for this bucket

Public access is granted to buckets and objects through access control lists (ACLs), bucket policies, access point policies, or all. In order to ensure that public access to this bucket and its objects is blocked, turn on Block all public access. These settings apply only to this bucket and its access points. AWS recommends that you turn on Block all public access, but before applying any of these settings, ensure that your applications will work correctly without public access. If you require some level of public access to this bucket or objects within, you can customize the individual settings below to suit your specific storage use cases. [Learn more](#)

☐ **Block all public access**
Turning this setting on is the same as turning on all four settings below. Each of the following settings are independent of one another.

☐ **Block public access to buckets and objects granted through new access control lists (ACLs)**
S3 will block public access permissions applied to newly added buckets or objects, and prevent the creation of new public access ACLs for existing buckets and objects. This setting doesn't change any existing permissions that allow public access to S3 resources using ACLs.
☐ **Block public access to buckets and objects granted through any access control lists (ACLs)**
S3 will ignore all ACLs that grant public access to buckets and objects.
☐ **Block public access to buckets and objects granted through new public bucket or access point policies**
S3 will block new bucket and access point policies that grant public access to buckets and objects. This setting doesn't change any existing policies that allow public access to S3 resources.
☐ **Block public and cross-account access to buckets and objects through any public bucket or access point policies**
S3 will ignore public and cross-account access for buckets or access points with policies that grant public access to buckets and objects.

Bucket creation is done.

2. Uploading website files to bucket.

Amazon S3

>

Buckets

>

awsprojectbk1

awsprojectbk1

Info

Objects

Properties

Permissions

Metrics

Management

Access Points

Objects (0)

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

Copy S3 URI

Copy URL

Download

Open

Delete

Actions

Create folder

Upload

Find objects by prefix

< 1 >

Name	Type	Last modified	Size	Storage class
No objects				
You don't have any objects in this bucket.				
Upload				

Files and folders (1 Total, 229.0 B)

Remove

Add files

Add folder

All files and folders in this table will be uploaded.

Find by name

< 1 >

Name	Folder	Type	Size
index.html	-	text/html	229.0 B

Destination

Destination

s3://awsprojectbk1

Destination details

Bucket settings that impact new objects stored in the specified destination.

Permissions

Grant public access and access to other AWS accounts.

3. Enabling Static website hosting

Hosting type

☒ **Host a static website**

Use the bucket endpoint as the web address. [Learn more](#)

☐ **Redirect requests for an object**

Redirect requests to another bucket or domain. [Learn more](#)

i For your customers to access content at the website endpoint, you must make all your content publicly readable. To do so, you can edit the S3 Block Public Access settings for the bucket. For more information, see [Using Amazon S3 Block Public Access](#)

Index document

Specify the home or default page of the website.

index.html

Error document - *optional*

This is returned when an error occurs.

error.html

Redirection rules - *optional*

Redirection rules, written in JSON, automatically redirect webpage requests for specific content. [Learn more](#)

Edit static website hosting Info

Static website hosting

Use this bucket to host a website or redirect requests. [Learn more](#)

Static website hosting

☐ Disable

☒ **Enable**

Hosting type

☒ **Host a static website**

Use the bucket endpoint as the web address. [Learn more](#)

☐ **Redirect requests for an object**

Redirect requests to another bucket or domain. [Learn more](#)

i For your customers to access content at the website endpoint, you must make all your content publicly readable. To do so, you can edit the S3 Block Public Access settings for the bucket. For more information, see [Using Amazon S3 Block Public Access](#)

Index document

Specify the home or default page of the website.

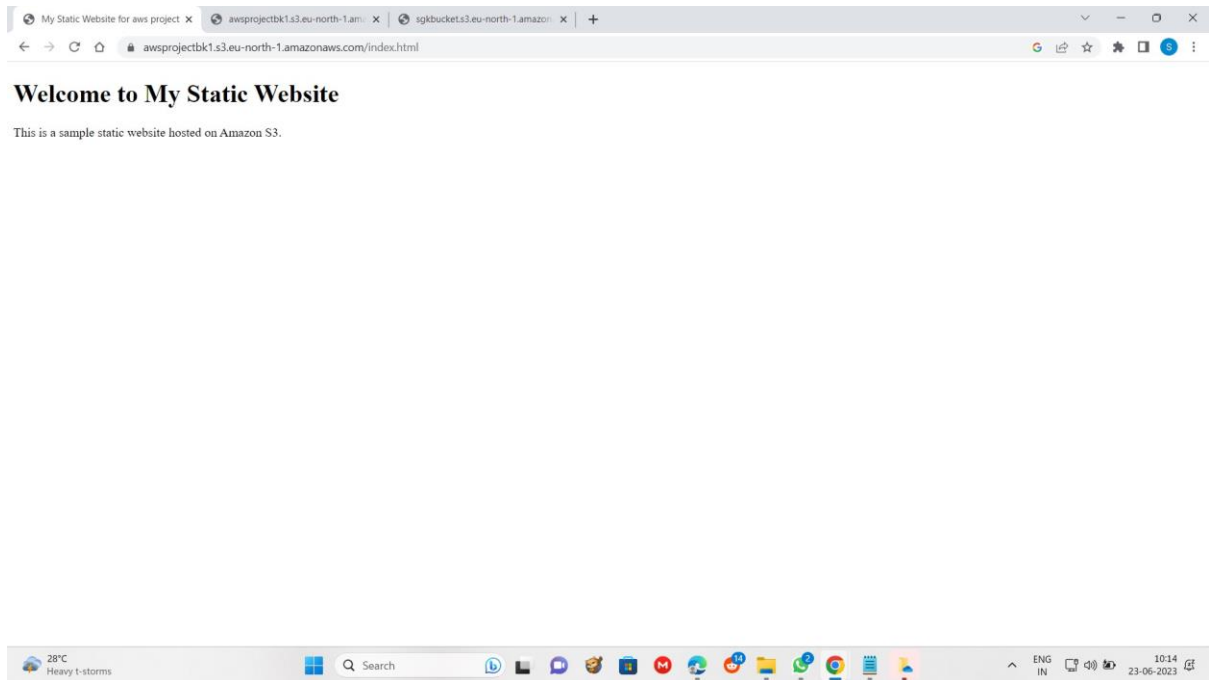
index.html

Static website code-

```
<!DOCTYPE html>
<html>
<head>
  <title>My Static Website for aws project </title>
</head>
<body>
  <h1>Welcome to My Static Website</h1>
  <p>This is a sample static website hosted on Amazon S3.</p>

</body>
</html>
```

Hosted static website

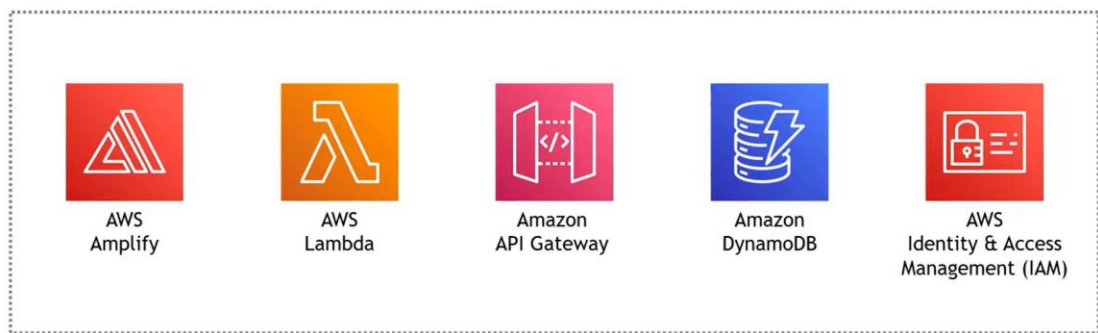


Hosting a dynamic website using AWS

To host a dynamic website in AWS, we use AWS services like

- AWS Amplify
- AWS Lambda
- Amazon Api Gateway
- Amazon DynamoDB
- AWS IAM

Services We'll be Using



Now let's look at each of the above-mentioned services in details

AWS Amplify

AWS Amplify was launched in 2017 and is a full-suite package of tools and services specifically designed to help developers create and launch apps with ease.

It can also include code libraries, ready-to-use components, and a built-in CLI. This tool's most significant benefit is that it allows you to integrate a myriad of functions quickly and securely for everything from API to AI.

Another reason behind the launch of AWS Amplify is the user experience. User experience on any application is the most integral component that needs to be taken care of. AWS Amplify was built to unify the UX across multiple platforms, including web and mobile.

It provides users with the flexibility to select the platform they are most comfortable building with and is especially beneficial when it comes to front end development. Most Amplify users also claim that it makes full-stack development much more comfortable with its scalability factor.

Now, let us peek into how AWS Amplify works.

How does AWS Amplify work?

You can think of AWS Amplify as a JavaScript library that lets you build and deploy serverless applications in the cloud. It is a full-stack application platform that is a combination of both client-side and server-side code. In a nutshell, AWS Amplify consists of three major components:

- Libraries
- UI
- CLI Toolchain.

All these components work collaboratively to manage the application development lifecycle. Here is a brief look into each element:

The Library: This component lets you add, integrate, and interact with the AWS cloud services. The library also makes it easy to add secure authentication, file storage, data storage, serverless APIs, analytics, push notifications, AR/VR, and multiple other features to your apps.

UI: These are pre-built UI components that are designed around cloud workflows in your application, including the authentication higher-order component.

CLI Toolchain: This last component helps in scaling your application. If you ever need to add more cloud services and features, the easy-to-use CLI commands can efficiently make changes to your AWS-managed backends.

In the next section, let us look at the main benefits of using AWS Amplify.

Advantages of Using AWS Amplify

- Easy and UI driven Development

AWS Amplify provides a simple, fast, and modern UI-driven approach to building mobile and web applications. The out-of-the-box UI component provides everything, so you do not have to code one. The design of the CLI processes and workflows is also seamless, which accelerates app development.

- Usage-Based Payment

Like many other paid AWS Services, the payment model for AWS Amplify is very flexible and cost-efficient as you only pay for the services you use.

- Backend Support

AWS Amplify improves app performance by offering built-in support for backend management.

- It's Free to Start

There are many free and impressive tiers with AWS Amplify that offer multiple benefits and zero cost. Only when you reach a high threshold of technical requirements would you need to set up a paid tier.

- Web-Based Analytics

AWS Amplify comes with a web-based analytics dashboard that is extremely useful for developers, designers, and project managers. It not only tracks user sessions and attributes but also offers in-app metrics. The analytics always stay up to date, allowing teams to manage and track projects.

Now that we know the main benefits of AWS Amplify let us dive into some of its downsides now.

AWS Amplify Limitations

According to the article [What is AWS Amplify? Advantages and Disadvantages of AWS Amplify](#) here are some limitations of AWS Amplify.

- Higher Learning Curve

If you are an AWS Amplify newbie, you might realize that the time you save in writing code will be used towards learning the platform. Sometimes for beginners, it is a bit difficult to find the right method in the documentation, and the multiple methods and versions can sometimes make it difficult to navigate.

- Consistent Changes

This is a continuously changing platform and as such new changes and features are rolled out consistently. This means that an AWS Amplify user must always stay on top of things to keep themselves up to date and to keep exploring the platform.

- Cost

Since AWS Amplify is a managed service, the end-user has less control over the environment and installed packages that can affect your website and will most likely cost more vs. handling the backend by yourself. You also must contend with other disadvantages that come with managed services.

- Traffic Distribution

You cannot use load balancers to distribute traffic when using AWS Amplify. This can be a big drawback in certain cases like managing traffic spikes and latency issues.

AWS Lambda

AWS Lambda is a serverless computing service provided by Amazon Web Services (AWS). Users of AWS Lambda create functions, self-contained applications written in one of the supported languages and runtimes, and upload them to AWS Lambda, which executes those functions in an efficient and flexible manner.

The Lambda functions can perform any kind of computing task, from serving web pages and processing streams of data to calling APIs and integrating with other AWS services.

The concept of “serverless” computing refers to not needing to maintain your own servers to run these functions. AWS Lambda is a fully managed service that takes care of all the infrastructure for you. And so “serverless” doesn’t mean that there are no servers involved: it just means that the servers, the operating systems, the network layer and the rest of the infrastructure have already been taken care of, so that you can focus on writing application code.

How does AWS Lambda work?

Each Lambda function runs in its own container. When a function is created, Lambda packages it into a new container and then executes that container on a multi-tenant cluster of machines managed by AWS. Before the functions start running, each function’s container is allocated its necessary RAM and CPU capacity. Once the functions finish running, the RAM allocated at the beginning is multiplied by the amount of time the function spent running. The customers then get charged based on the allocated memory and the amount of run time the function took to complete.

The entire infrastructure layer of AWS Lambda is managed by AWS. Customers don’t get much visibility into how the system operates, but they also don’t need to worry about updating the underlying machines, avoiding network contention, and so on—AWS takes care of this itself.

And since the service is fully managed, using AWS Lambda can save you time on operational tasks. When there is no infrastructure to maintain, you can spend more time working on the application code—even though this also means you give up the flexibility of operating your own infrastructure.

One of the distinctive architectural properties of AWS Lambda is that many instances of the same function, or of different functions from the same AWS account, can be executed concurrently. Moreover, the concurrency can vary according to the time of day or the day of the week, and such variation makes no difference to Lambda—you only get charged for the compute your functions use. This makes AWS Lambda a good fit for deploying highly scalable cloud computing solutions.

Why is AWS Lambda an essential part of the Serverless architecture?

When building Serverless applications, AWS Lambda is one of the main candidates for running the application code. Typically, to complete a Serverless stack you'll need:

- a computing service;
- a database service; and
- an HTTP gateway service.

Lambda fills the primary role of the compute service on AWS. It also integrates with many other AWS services and, together with API Gateway, DynamoDB and RDS, forms the basis for Serverless solutions for those using AWS. Lambda supports many of the most popular languages and runtimes, so it's a good fit for a wide range of Serverless developers.

What are the most common use cases for AWS Lambda?

Due to Lambda's architecture, it can deliver great benefits over traditional cloud computing setups for applications where:

1. individual tasks run for a short time;
2. each task is generally self-contained;
3. there is a large difference between the lowest and highest levels in the workload of the application.

Some of the most common use cases for AWS Lambda that fit these criteria are:

Scalable APIs. When building APIs using AWS Lambda, one execution of a Lambda function can serve a single HTTP request. Different parts of the API can be routed to different Lambda functions via Amazon API Gateway. AWS Lambda automatically scales individual functions according to the demand for them, so different parts of your API can scale differently according to current usage levels. This allows for cost-effective and flexible API setups.

Data processing. Lambda functions are optimized for event-based data processing. It is easy to integrate AWS Lambda with datasources like Amazon DynamoDB and trigger a Lambda function for specific kinds of data events. For example, you could employ Lambda to do some work every time an item in DynamoDB is created or updated, thus making it a good fit for things like notifications, counters and analytics.

Task automation-

With its event-driven model and flexibility, AWS Lambda is a great fit for automating various business tasks that don't require an entire server at all times. This might include running scheduled jobs that perform cleanup in your infrastructure, processing data from forms submitted on your website, or moving data around between different datastores on demand.

Supported languages and runtimes-

As of now, AWS Lambda doesn't support all programming languages, but it does support a number of the most popular languages and runtimes. This is the full list of what's supported:

- Node.js 8.10
- Node.js 10.x (normally the latest LTS version from the 10.x series)
- Node.js 12.x (normally the latest LTS version from the 12.x series)
- Python 2.7
- Python 3.6
- Python 3.7
- Python 3.8
- Ruby 2.5
- Java 8 - This includes JVM-based languages that can run on Java 8's JVM — the latest Clojure 1.10 and Scala 2.12 both run on Java 8 so can be used with AWS Lambda
- Java 11
- Go 1.x (latest release)
- C# — .NET Core 1.0
- C# — .NET Core 2.1
- PowerShell Core 6.0

All these runtimes are maintained by AWS and are provided in an Amazon Linux or Amazon Linux 2 environment. For each of the supported languages, AWS provides an SDK that makes it easier for you to write your Lambda functions and integrate them with other AWS services.

A few additional runtimes are still in the pre-release stage. These runtimes are being developed as a part of AWS Labs and are not mentioned in the official documentation:

- Rust 1.31
- C++

The C++ runtime also serves as an example for creating custom runtimes for AWS Lambda. See the AWS docs for the details of how to create a custom runtime if your language isn't supported by default.

Benefits of using AWS Lambda-

AWS Lambda has a few unique advantages over maintaining your own servers in the cloud. The main ones are:

Pay per use. In AWS Lambda, you pay only for the compute your functions use, plus any network traffic generated. For workloads that scale significantly according to time of day, this type of billing is generally more cost-effective.

Fully managed infrastructure. Now that your functions run on the managed AWS infrastructure, you don't need to think about the underlying servers—AWS takes care of this for you. This can result in significant savings on operational tasks such as upgrading the operating system or managing the network layer.

Automatic scaling. AWS Lambda creates the instances of your function as they are requested. There is no pre-scaled pool, no scale levels to worry about, no settings to tune—and at the same time your functions are available whenever the load increases or decreases. You only pay for each function's run time.

Tight integration with other AWS products. AWS Lambda integrates with services like DynamoDB, S3 and API Gateway, allowing you to build functionally complete applications within your Lambda functions.

Limitations of AWS Lambda-

While AWS Lambda has many advantages, there are a few things you should know before using it in production.

Cold start time-

When a function is started in response to an event, there may be a small amount of latency between the event and when the function runs. If your function hasn't been used in the last 15 minutes, the latency can be as high as 5-10 seconds, making it hard to rely on Lambda for latency-critical applications. There are ways to work around it, including a method we wrote about in our blog.

Function limits-

The Lambda functions have a few limits applied to them:

Execution time/run time. A Lambda function will time out after running for 15 minutes. There is no way to change this limit. If running your function typically takes more than 15 minutes, AWS Lambda might not be a good solution for your task.

Memory available to the function. The options for the amount of RAM available to the Lambda functions range from 128MB to 3,008MB with a 64MB step.

Code package size. The zipped Lambda code package should not exceed 50MB in size, and the unzipped version shouldn't be larger than 250MB.

Concurrency. By default, the concurrent execution for all AWS Lambda functions within a single AWS account are limited to 1,000. (You can request a limit increase for this number by contacting AWS support.)

Any Lambda executions triggered above your concurrency limit will be throttled and will be

forced to wait until other functions finish running.

Payload size. When using Amazon API Gateway to trigger Lambda functions in response to HTTP requests (i.e. when building a web application), the maximum payload size that API Gateway can handle is 10MB.

Not always cost-effective-

On AWS Lambda, you pay only for the used function runtime (plus any associated charges like network traffic). This can produce significant cost savings for certain usage patterns, for example, with cron jobs or other on-demand tasks. However, when the load for your application increases, the AWS Lambda cost increases proportionally and might end up being higher than the cost of similar infrastructure on AWS EC2 or other cloud providers.

Limited number of supported runtimes-

While AWS Lambda allows adding custom runtimes, creating them can be a lot of work. So if the version of the programming language you are using isn't supported on Lambda, you might be better off using AWS EC2 or a different cloud provider.

Amazon API Gateway

Amazon API Gateway is an AWS service for creating, publishing, maintaining, monitoring, and securing REST, HTTP, and WebSocket APIs at any scale. API developers can create APIs that access AWS or other web services, as well as data stored in the AWS Cloud. As an API Gateway API developer, you can create APIs for use in your own client applications. Or you can make your APIs available to third-party app developers. For more information, see [Who uses API Gateway?](#).

API Gateway creates RESTful APIs that:

- Are HTTP-based.
- Enable stateless client-server communication.
- Implement standard HTTP methods such as GET, POST, PUT, PATCH, and DELETE.

For more information about API Gateway REST APIs and HTTP APIs, see [Choosing between REST APIs and HTTP APIs](#), [Working with HTTP APIs](#), [Use API Gateway to create REST APIs](#), and [Creating a REST API in Amazon API Gateway](#).

API Gateway creates WebSocket APIs that:

- Adhere to the WebSocket protocol, which enables stateful, full-duplex communication between client and server.
- Route incoming messages based on message content.

Architecture of API Gateway

The following diagram shows API Gateway architecture.



This diagram illustrates how the APIs you build in Amazon API Gateway provide you or your developer customers with an integrated and consistent developer experience for building AWS serverless applications. API Gateway handles all the tasks involved in accepting and processing up to hundreds of thousands of concurrent API calls. These tasks include traffic management, authorization and access control, monitoring, and API version management.

API Gateway acts as a "front door" for applications to access data, business logic, or functionality from your backend services, such as workloads running on Amazon Elastic Compute Cloud (Amazon EC2), code running on AWS Lambda, any web application, or real-time communication applications.

Features of API Gateway

Amazon API Gateway offers features such as the following:

- Support for stateful (WebSocket) and stateless (HTTP and REST) APIs.
- Powerful, flexible authentication mechanisms, such as AWS Identity and Access Management policies, Lambda authorizer functions, and Amazon Cognito user pools.
- Canary release deployments for safely rolling out changes.
- CloudTrail logging and monitoring of API usage and API changes.
- CloudWatch access logging and execution logging, including the ability to set alarms. For more information, see [Monitoring REST API execution with Amazon CloudWatch metrics](#) and [Monitoring WebSocket API execution with CloudWatch metrics](#).
- Ability to use AWS CloudFormation templates to enable API creation. For more information, see [Amazon API Gateway Resource Types Reference](#) and [Amazon API Gateway V2 Resource Types Reference](#).
- Support for custom domain names.
- Integration with AWS WAF for protecting your APIs against common web exploits.

- Integration with AWS X-Ray for understanding and triaging performance latencies.

For a complete list of API Gateway feature releases, see Document history.

Accessing API Gateway

You can access Amazon API Gateway in the following ways:

- AWS Management Console – The AWS Management Console provides a web interface for creating and managing APIs. After you complete the steps in Prerequisites for getting started with API Gateway, you can access the API Gateway console at <https://console.aws.amazon.com/apigateway>.
- AWS SDKs – If you're using a programming language that AWS provides an SDK for, you can use an SDK to access API Gateway. SDKs simplify authentication, integrate easily with your development environment, and provide access to API Gateway commands. For more information, see Tools for Amazon Web Services.
- API Gateway V1 and V2 APIs – If you're using a programming language that an SDK isn't available for, see the Amazon API Gateway Version 1 API Reference and Amazon API Gateway Version 2 API Reference.
- AWS Command Line Interface – For more information, see Getting Set Up with the AWS Command Line Interface in the *AWS Command Line Interface User Guide*.
- AWS Tools for Windows PowerShell – For more information, see Setting Up the AWS Tools for Windows PowerShell in the *AWS Tools for Windows PowerShell User Guide*.

Part of AWS serverless infrastructure

Together with AWS Lambda, API Gateway forms the app-facing part of the AWS serverless infrastructure.

For an app to call publicly available AWS services, you can use Lambda to interact with required services and expose Lambda functions through API methods in API Gateway. AWS Lambda runs your code on a highly available computing infrastructure. It performs the necessary execution and administration of computing resources. To enable serverless applications, API Gateway supports streamlined proxy integrations with AWS Lambda and HTTP endpoints.

Amazon DynamoDB

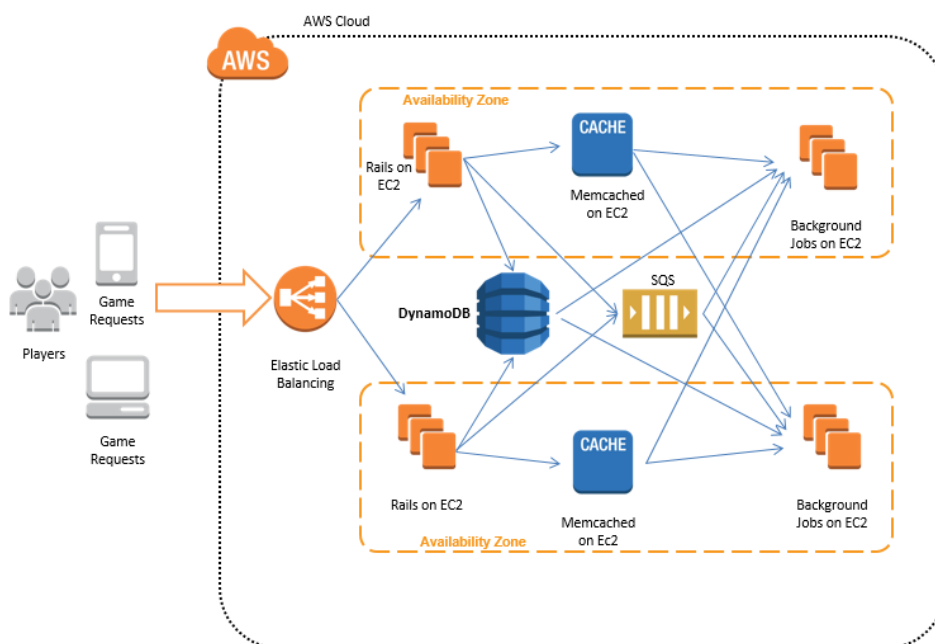
Amazon DynamoDB is a cloud-native NoSQL primarily key-value database. Let's define each of those terms.

- DynamoDB is cloud-native in that it does not run on-premises or even in a hybrid cloud; it only runs on Amazon Web Services (AWS). This enables it to scale as needed without requiring a customer's capital investment in hardware. It also has attributes

common to other cloud-native applications, such as elastic infrastructure deployment (meaning that AWS will provision more servers in the background as you request additional capacity).

- DynamoDB is NoSQL in that it does not support ANSI Structured Query Language (SQL). Instead, it uses a proprietary API based on JavaScript Object Notation (JSON). This API is generally not called directly by user developers, but invoked through AWS Software Developer Kits (SDKs) for DynamoDB written in various programming languages (C++, Go, Java, JavaScript, Microsoft .NET, Node.js, PHP, Python and Ruby).
- DynamoDB is primarily a key-value store in the sense that its data model consists of key-value pairs in a schemaless, very large, non-relational table of rows (records). It does not support relational database management systems (RDBMS) methods to join tables through foreign keys. It can also support a document store data model using JavaScript Object Notation (JSON).

DynamoDB's NoSQL design is oriented towards simplicity and scalability, which appeal to developers and devops teams respectively. It can be used for a wide variety of semistructured data-driven applications prevalent in modern and emerging use cases beyond traditional databases, from the Internet of Things (IoT) to social apps or massive multiplayer games. With its broad programming language support, it is easy for developers to get started and to create very sophisticated applications using DynamoDB.



What is a DynamoDB Database?

Outside of Amazon employees, the world doesn't know much about the exact nature of this database. There is a development version known as DynamoDB Local used to run on

developer laptops written in Java, but the cloud-native database architecture is proprietary closed-source.

While we cannot describe exactly what DynamoDB *is*, we can describe how you *interact* with it. When you set up DynamoDB on AWS, you do not provision specific servers or allocate set amounts of disk. Instead, you provision throughput — you define the database based on provisioned capacity — how many transactions and how many kilobytes of traffic you wish to support per second. Users specify a service level of read capacity units (RCUs) and write capacity units (WCUs).

As stated above, users generally do not directly make DynamoDB API calls. Instead, they will integrate an AWS SDK into their application, which will handle the back-end communications with the server.

DynamoDB data modeling needs to be denormalized. For developers used to working with both SQL and NoSQL databases, the process of rethinking their data model is nontrivial, but also not insurmountable.

History of the Amazon DynamoDB Database

DynamoDB was inspired by the seminal Dynamo white paper (2007) written by a team of Amazon developers. This white paper cited and contrasted itself from Google's Bigtable (2006) paper published the year before.

The original Dynamo database was used internally at Amazon as a completely in-house, proprietary solution. It is a completely different customer-oriented Database as a Service (DBaaS) that runs on Amazon Web Services (AWS) Elastic Compute Cloud (EC2) instances. DynamoDB was released in 2012, five years after the original white paper that inspired it.

While DynamoDB was inspired by the original paper, it was not beholden to it. Many things had changed in the world of Big Data over the intervening years since the paper was published. It was designed to build on top of a “core set of strong distributed systems principles resulting in an ultra-scalable and highly reliable database system.”

DynamoDB Release History

Since its initial release, Amazon has continued to expand on the DynamoDB API, extending its capabilities. DynamoDB documentation includes a document history as part of the DynamoDB Developer Guide to track key milestones in DynamoDB's release history:

Jan 2012 — Initial release of DynamoDB

Aug 2012 — Binary data type support

Apr 2013 — New API version, local secondary indexes

Dec 2013 — Global secondary indexes

Apr 2014 — Query filter, improved conditional expressions

Oct 2014 — Larger item sizes, flexible scaling, JSON document model

Apr 2015 — New comparison functions for conditional writes, improved query expressions

Jul 2015 — Scan with strongly-consistent reads, streams, cross-region replication

Feb 2017 — Time-to-Live (TTL) automatic expiration

Apr 2017 — DynamoDB Accelerator (DAX) cache
Jun 2017 — Automatic scaling
Aug 2017 — Virtual Private Cluster (VPC) endpoints
Nov 2017 — Global tables, backup and restore
Feb 2018 — Encryption at rest
Apr 2018 — Continuous backups and Point-in-Time Recovery (PITR)
Nov 2018 — Encryption at rest, transactions, on-demand billing
Feb 2019 — DynamoDB Local (downloadable version for developers)
Aug 2019 — NoSQL Workbench preview

DynamoDB Database Overview

DynamoDB Design Principles

As stated in Werner Vogel's initial blog about DynamoDB, the database was designed to build on top of a "core set of strong distributed systems principles resulting in an ultra-scalable and highly reliable database system." It needed to provide these attributes:

- Managed — provided 'as-a-Service' so users would not need to maintain the database
- Scalable — automatically provision hardware on the backend, invisible to the user
- Fast — support predictive levels of provisioned throughput at relatively low latencies
- Durable and highly available — multiple availability zones for failures/disaster recovery
- Flexible — make it easy for users to get started and continuously evolve their database
- Low cost — be affordable for users as they start and as they grow

The database needed to "provide fast performance at any scale," allowing developers to "start small with just the capacity they need and then increase the request capacity of a given table as their app grows in popularity." Predictable performance was ensured by provisioning the database with guarantees of throughput, measured in "capacity units" of reads and writes. "Fast" was defined as single-digit milliseconds, based on data stored in Solid State Drives (SSDs).

It was also designed based on lessons learned from the original Dynamo, SimpleDB and other Amazon offerings, "to reduce the operational complexity of running large database systems." Developers wanted a database that "freed them from the burden of managing databases and allowed them to focus on their applications." Based on Amazon's experience with SimpleDB, they knew that developers wanted a database that "just works." By its design users no longer were responsible for provisioning hardware, installing operating systems, applications, containers, or any of the typical devops tasks for on-premises deployments. On the back end, DynamoDB "automatically spreads the data and traffic for a table over a sufficient number of servers to meet the request capacity specified by the customer."

Furthermore, DynamoDB would replicate data across multiple AWS Availability Zones to provide high availability and durability.

As a commercial managed service, Amazon also wanted to provide a system that would make it transparent for customers to predict their operational costs.

DynamoDB Data Storage Format

To manage data, DynamoDB uses hashing and b-trees. While DynamoDB supports JSON, it only uses it as a transport format; JSON is not used as a storage format. Much of the exact implementation of DynamoDB's data storage format remains proprietary. Generally a user would not need to know about it. Data in DynamoDB is generally exported through streaming technologies or bulk downloaded into CSV files through ETL-type tools like AWS Glue. As a managed service, the exact nature of data on disk, much like the rest of the system specification (hardware, operating system, etc.), remains hidden from end users of DynamoDB.

[AWS IAM \(Identity and Access Management\)](#)

You can use AWS Identity and Access Management Roles Anywhere to obtain temporary security credentials in IAM for workloads such as servers, containers, and applications that run outside of AWS. Your workloads can use the same IAM policies and IAM roles that you use with AWS applications to access AWS resources. Using IAM Roles Anywhere means you don't need to manage long-term credentials for workloads running outside of AWS.

To use IAM Roles Anywhere, your workloads must use X.509 certificates issued by your certificate authority (CA). You register the CA with IAM Roles Anywhere as a trust anchor to establish trust between your public-key infrastructure (PKI) and IAM Roles Anywhere. You can also use AWS Private Certificate Authority (AWS Private CA) to create a CA and then use that to establish trust with IAM Roles Anywhere. AWS Private CA is a managed private CA service for managing your CA infrastructure and your private certificates. For more information, see [What is AWS Private CA](#).

Topics

- [IAM Roles Anywhere concepts](#)
- [Accessing IAM Roles Anywhere](#)

[IAM Roles Anywhere concepts](#)

Learn the basic terms and concepts used in IAM Roles Anywhere.

- [Trust anchors](#)

You establish trust between IAM Roles Anywhere and your certificate authority (CA) by creating a *trust anchor*. A trust anchor is a reference to either AWS Private CA or an external CA certificate. Your workloads outside of AWS authenticate with the trust anchor using

certificates issued by the trusted CA in exchange for temporary AWS credentials. For more information, see IAM Roles Anywhere trust model.

- Roles

An IAM role is an IAM identity that you can create in your account that has specific permissions. A role is intended to be assumable by anyone who needs it. For IAM Roles Anywhere to be able to assume a role and deliver temporary AWS credentials, the role must trust the IAM Roles Anywhere service principal. For more information, see Role trusts.

- Profiles

To specify which roles IAM Roles Anywhere assumes and what your workloads can do with the temporary credentials, you create a profile. In a profile, you can define permissions with IAM managed policies to limit the permissions for a created session.

Accessing IAM Roles Anywhere

AWS Management Console

You can manage your IAM Roles Anywhere resources using the browser-based console at <https://console.aws.amazon.com/rolesanywhere/>.

AWS Command Line Tools

You can use the AWS command line tools to issue commands at your system command line to perform IAM Roles Anywhere and other AWS tasks. This can be faster and more convenient than using the console. The command line tools can be useful if you want to build scripts to perform AWS tasks.

AWS provides the AWS Command Line Interface (AWS CLI). For information about installing and using the AWS CLI, see the AWS Command Line Interface User Guide .





AWS SDKs

The AWS software development kits (SDKs) consist of libraries and sample code for various programming languages and platforms including Java, Python, Ruby, .NET, iOS and Android, and others. The SDKs include tasks such as cryptographically signing requests, managing errors, and retrying requests automatically.

Process for hosting a dynamic website in AWS

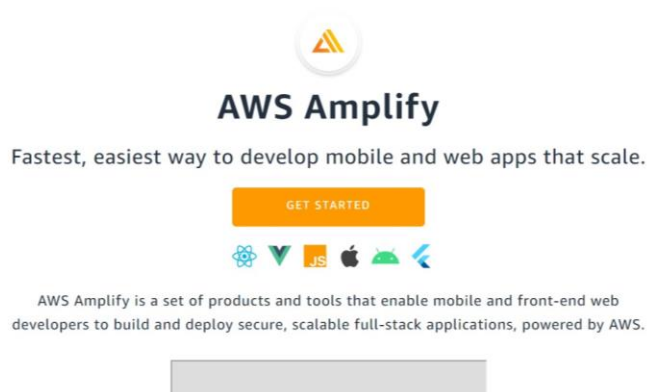
Gathering website files to upload-

We need to gather the files that belongs to the dynamic website that we are hosting though AWS.

 index1	23-06-2023 11:16	Compressed (zipp...	1 KB
 index1	23-06-2023 11:16	Chrome HTML Do...	1 KB
 index	23-06-2023 10:04	Chrome HTML Do...	1 KB
 free-aws-3215369-2673787.webp	23-06-2023 09:51	WEBP File	6 KB

Upload website file to AWS Amplify-

Now, we upload website files to AWS Amplify through a AWS S3 bucket as shown earlier.



Amplify Hosting

Get started

Amplify Studio



Build an app

Build an app backend with auth, data, and storage, and create custom UI components. Then integrate them in your app with just a few steps.



Get started

Amplify Hosting



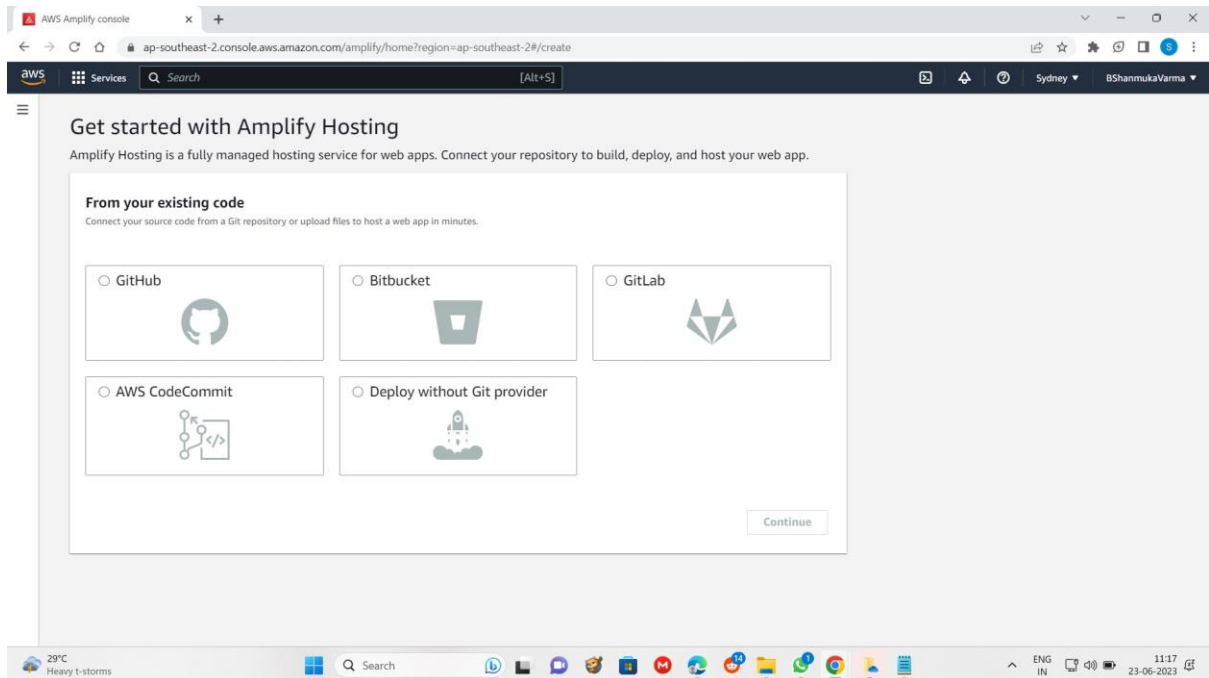
Host your web app

Connect your Git repository to continuously deploy your frontend and backend. Host it on a globally available CDN.

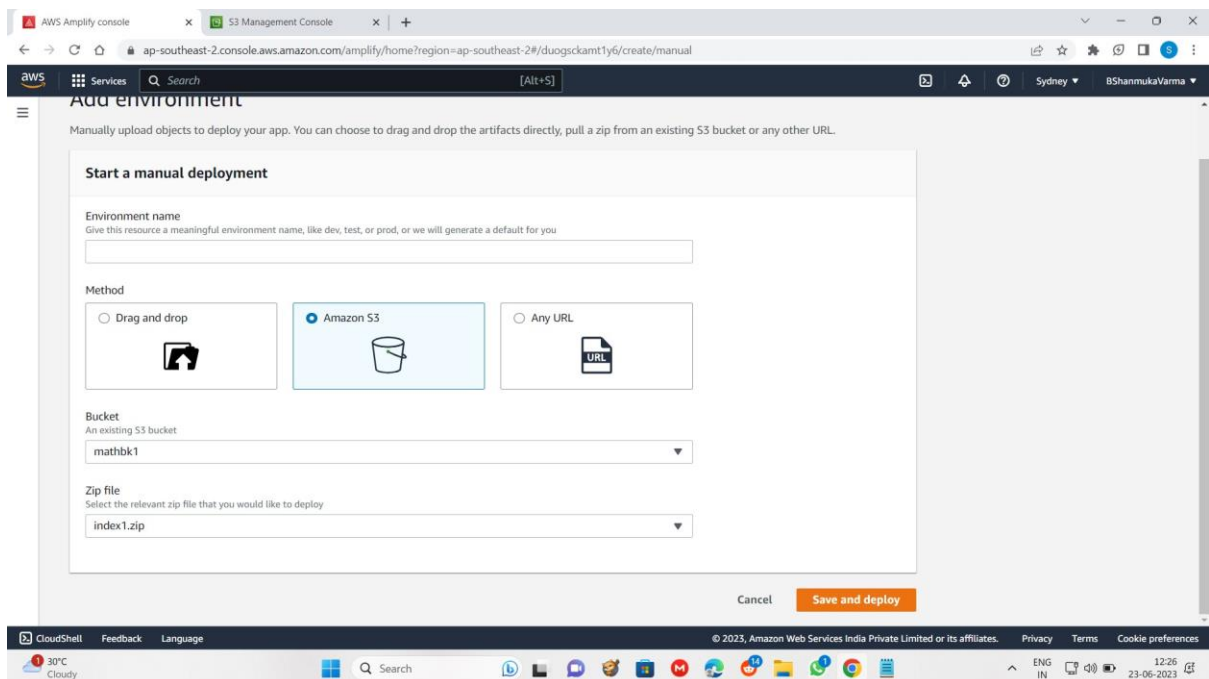


Get started

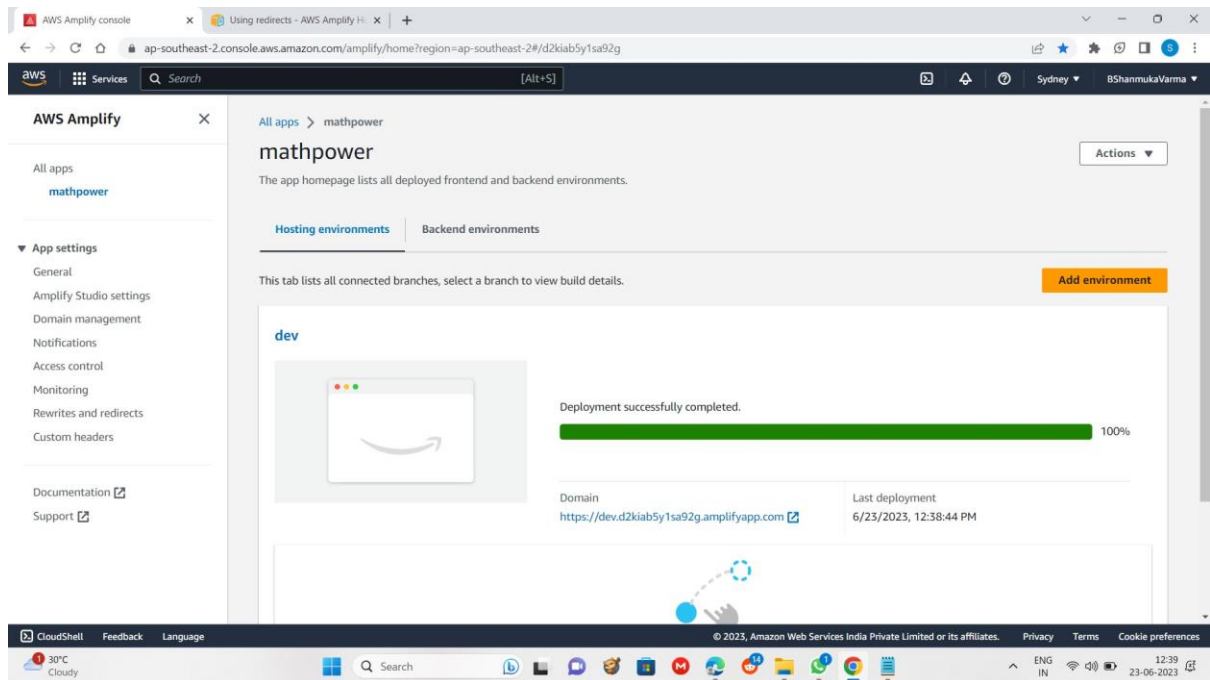
Select “deploy without git provider”



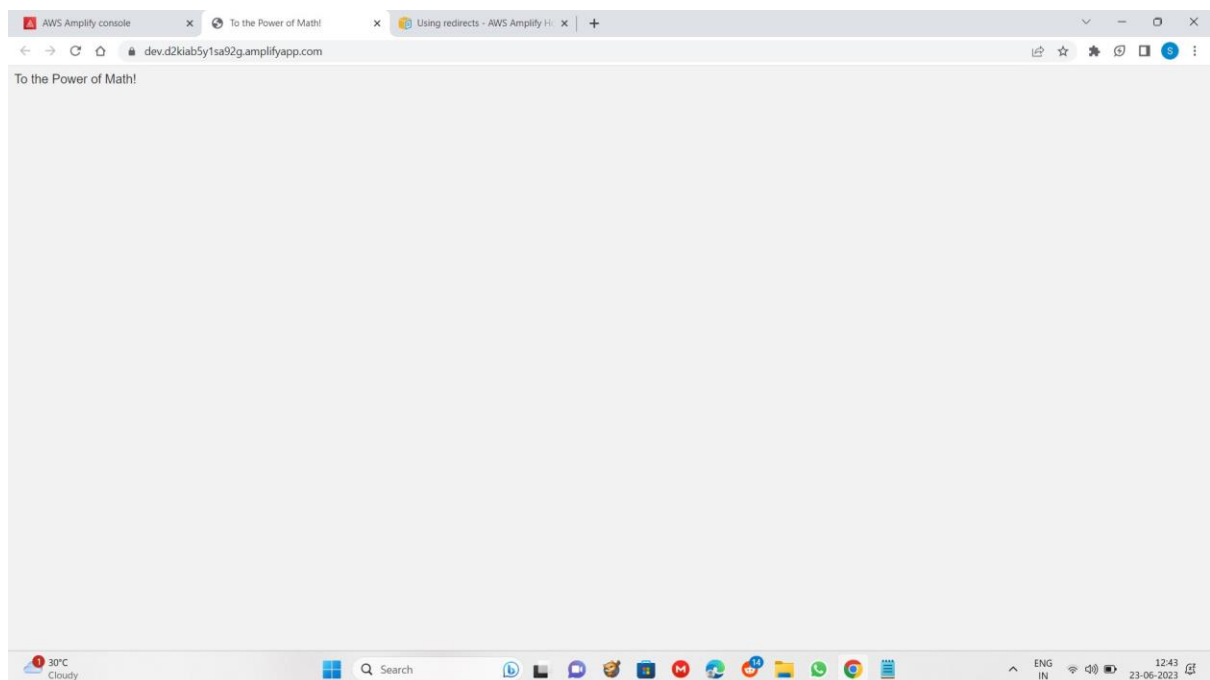
Using Amazon s3 bucket to upload the zipped website file



Upload and deployment completed

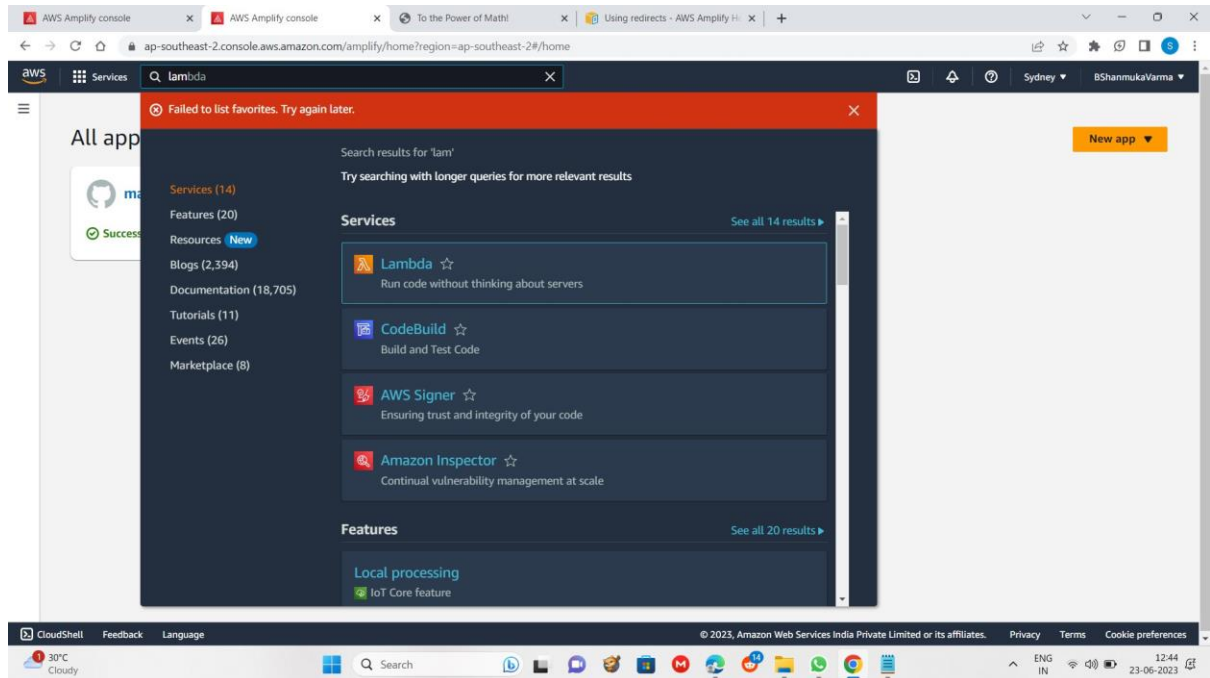


Basic deployment of the website



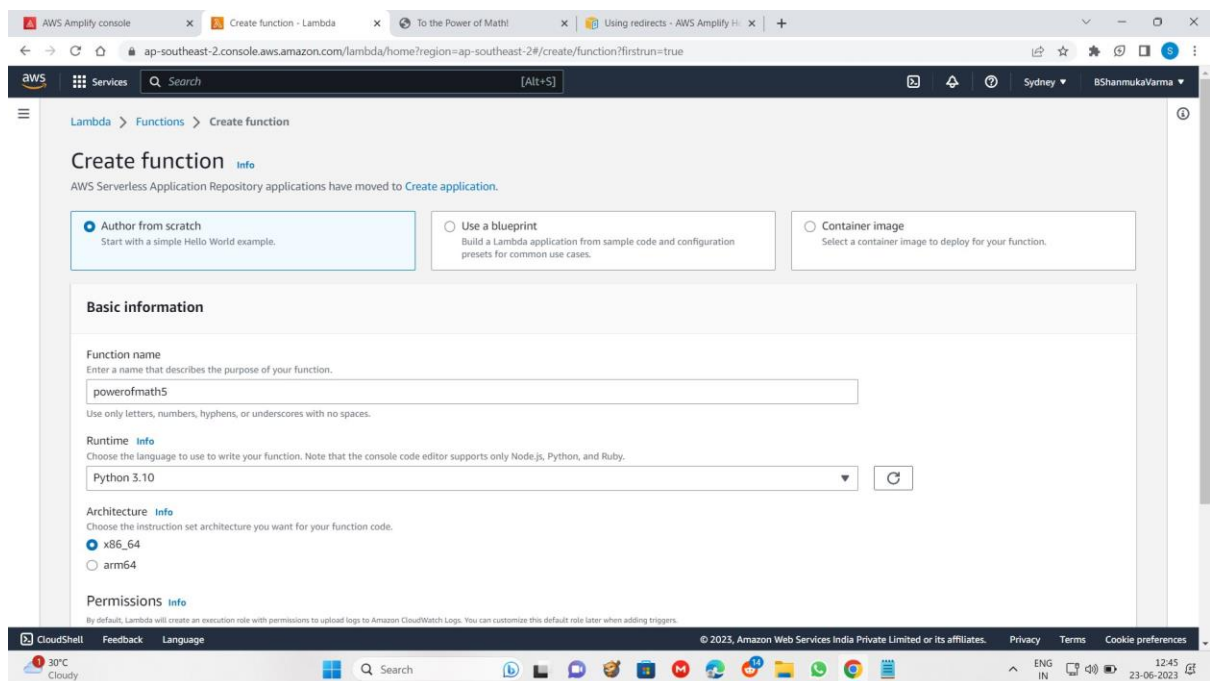
Creating a Lambda function-

Code that runs upon some triggers

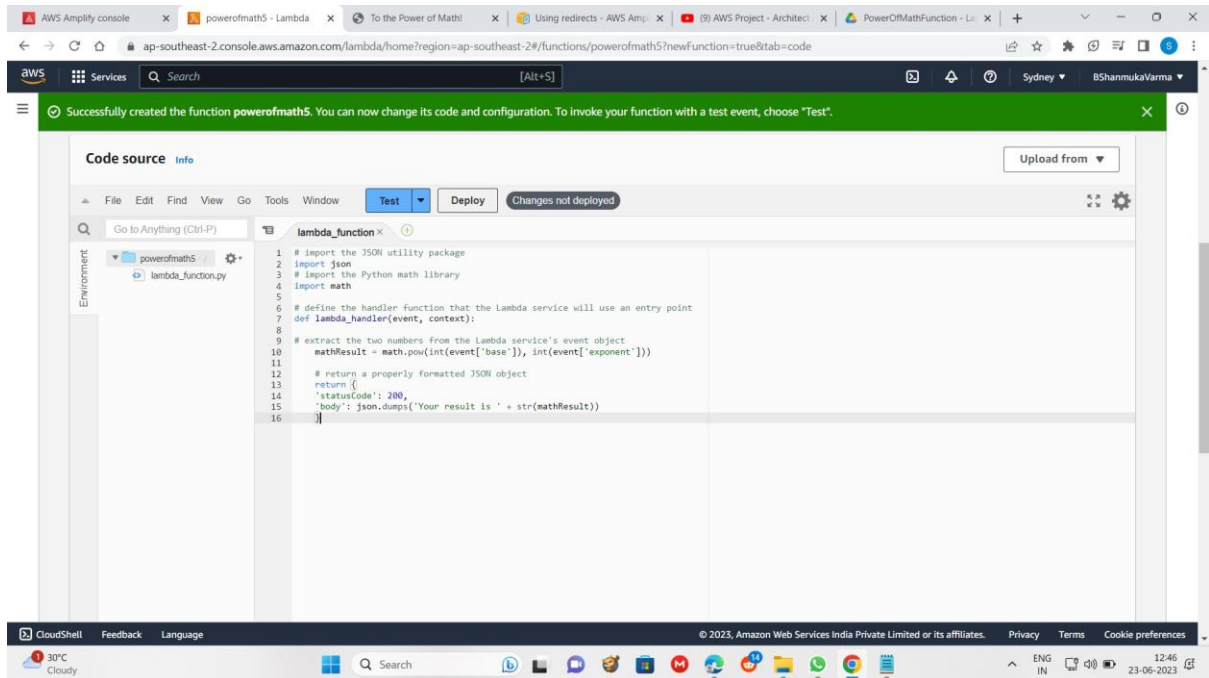


Whenever lambda gets triggered it executes a certain code that was given to it.

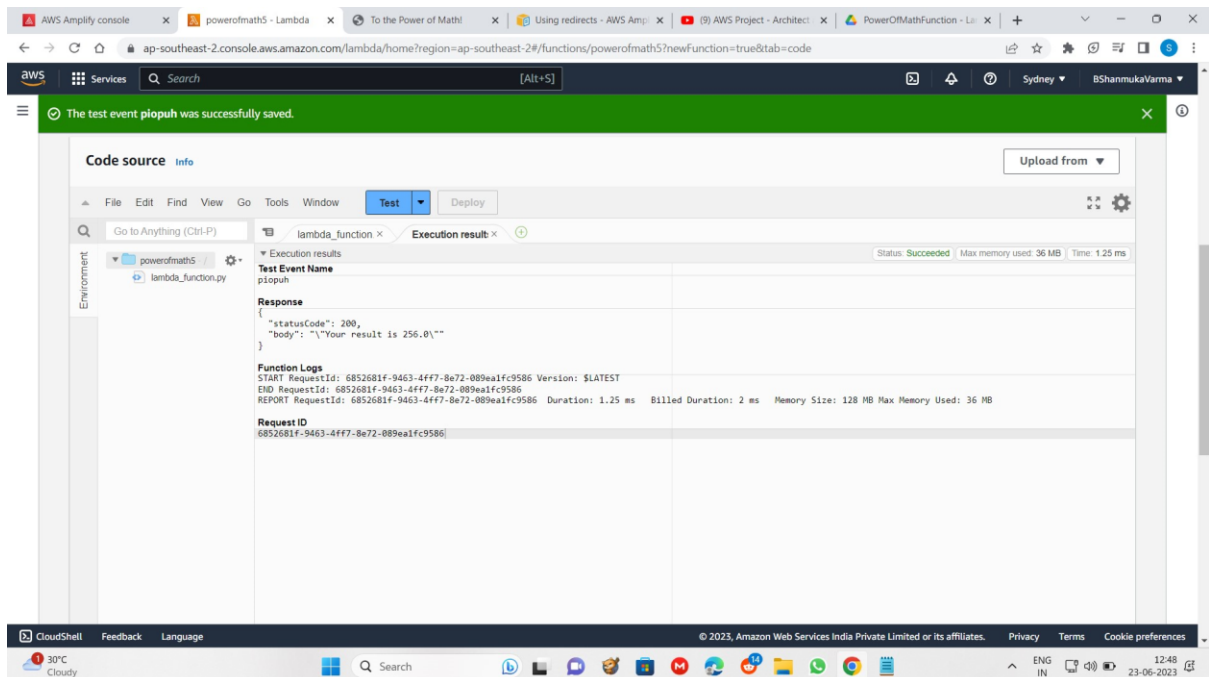
Creating a new function



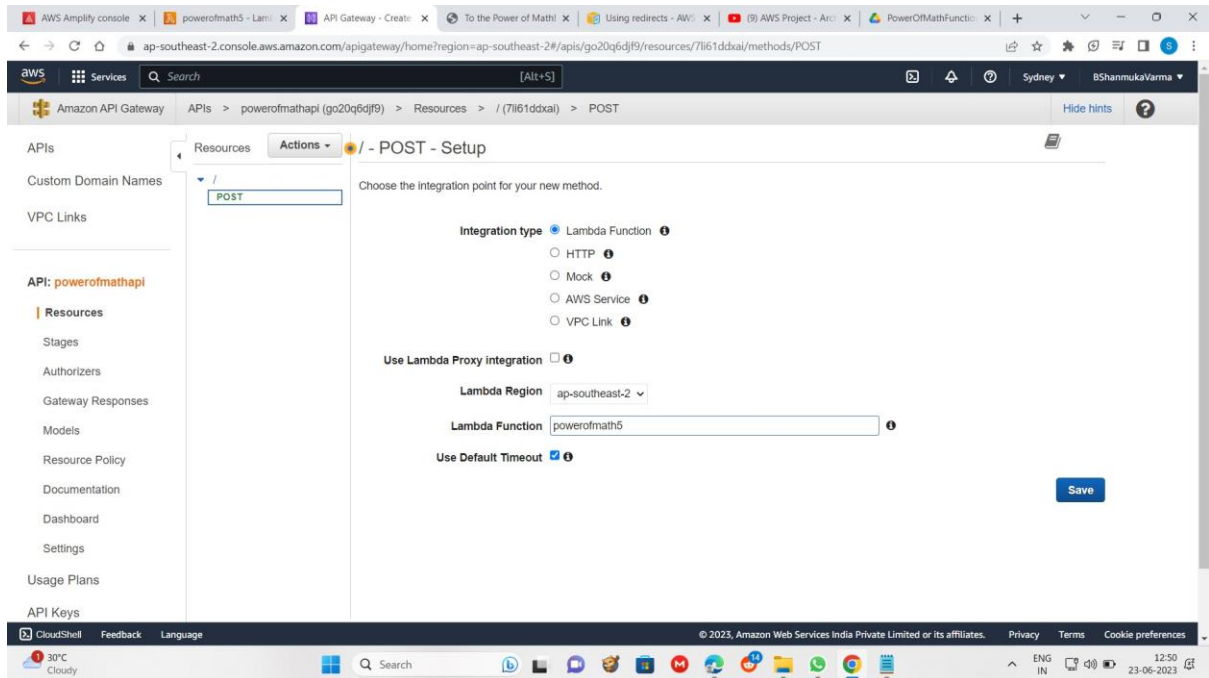
Adding trigger code to the function



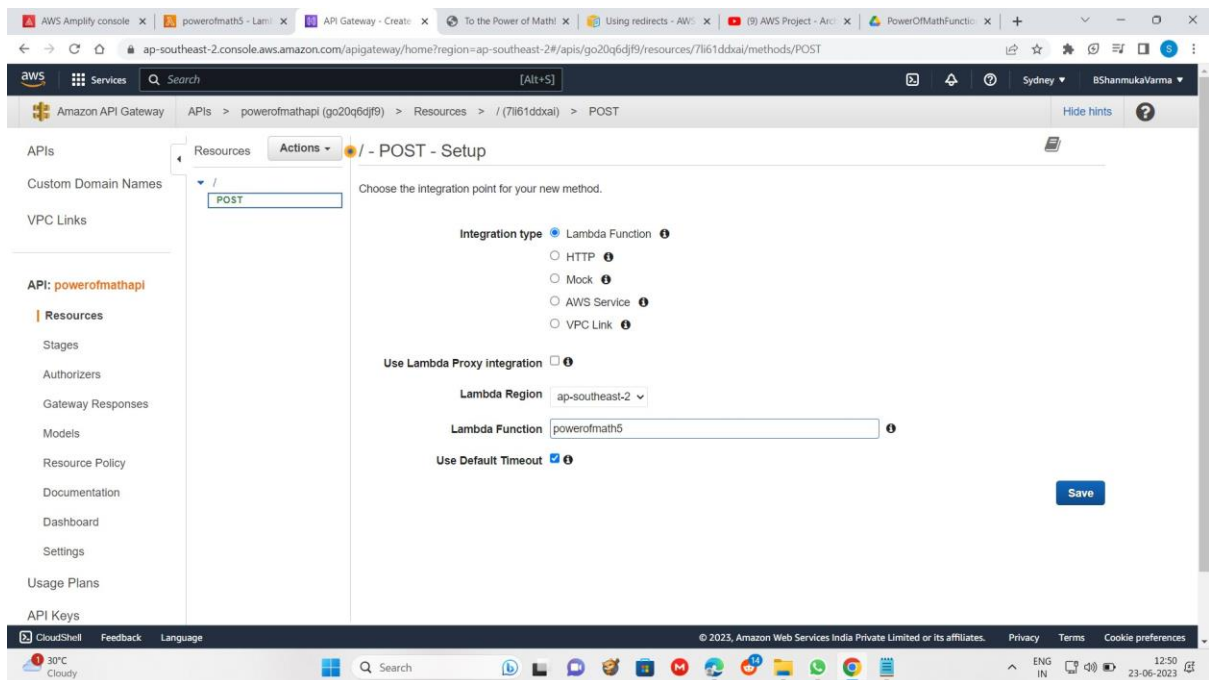
Now we deploy it



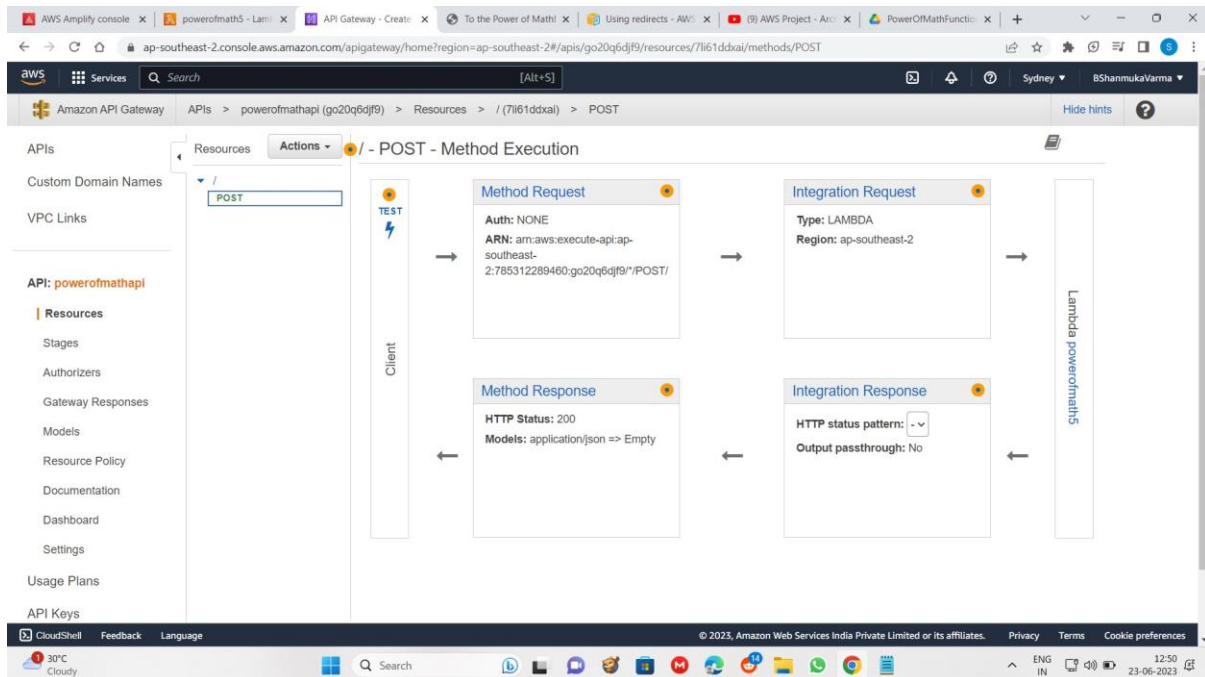
Integrating Amazon Api gateway-



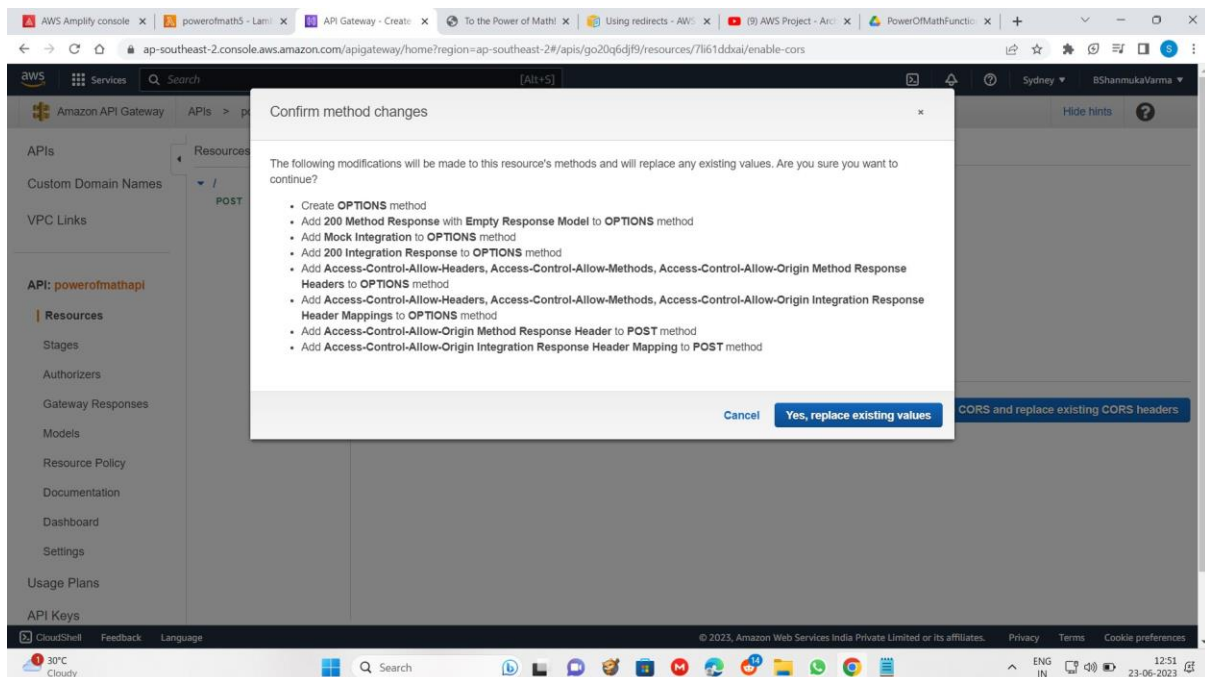
Setup



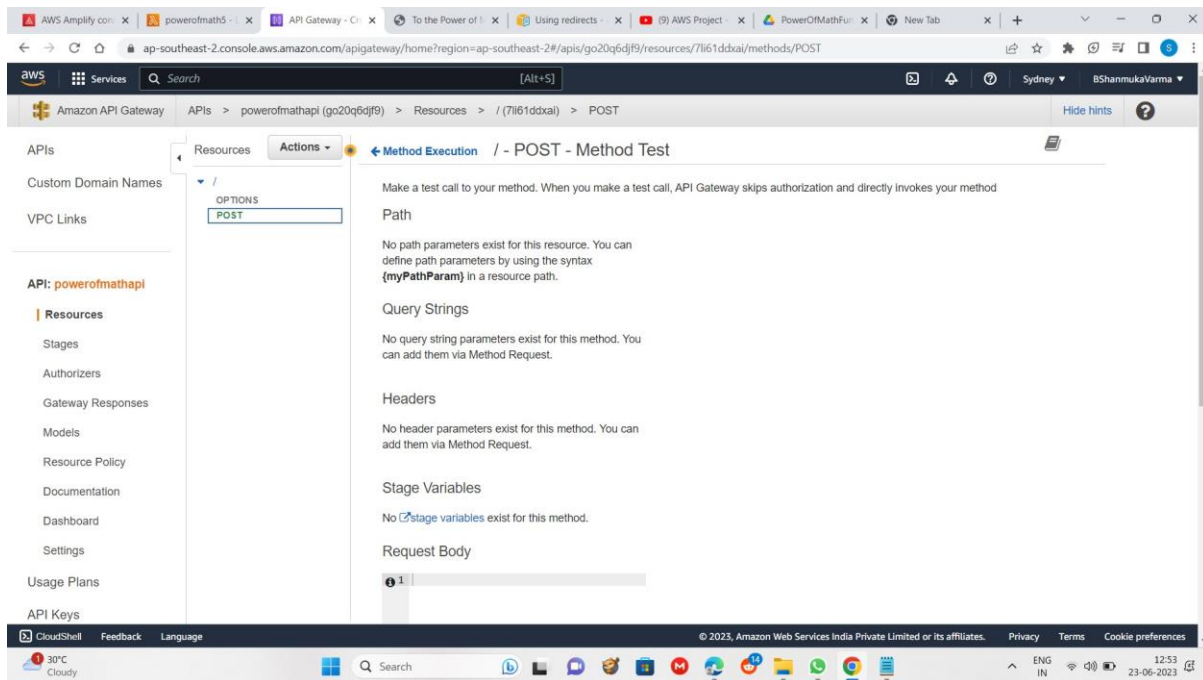
POST Method execution



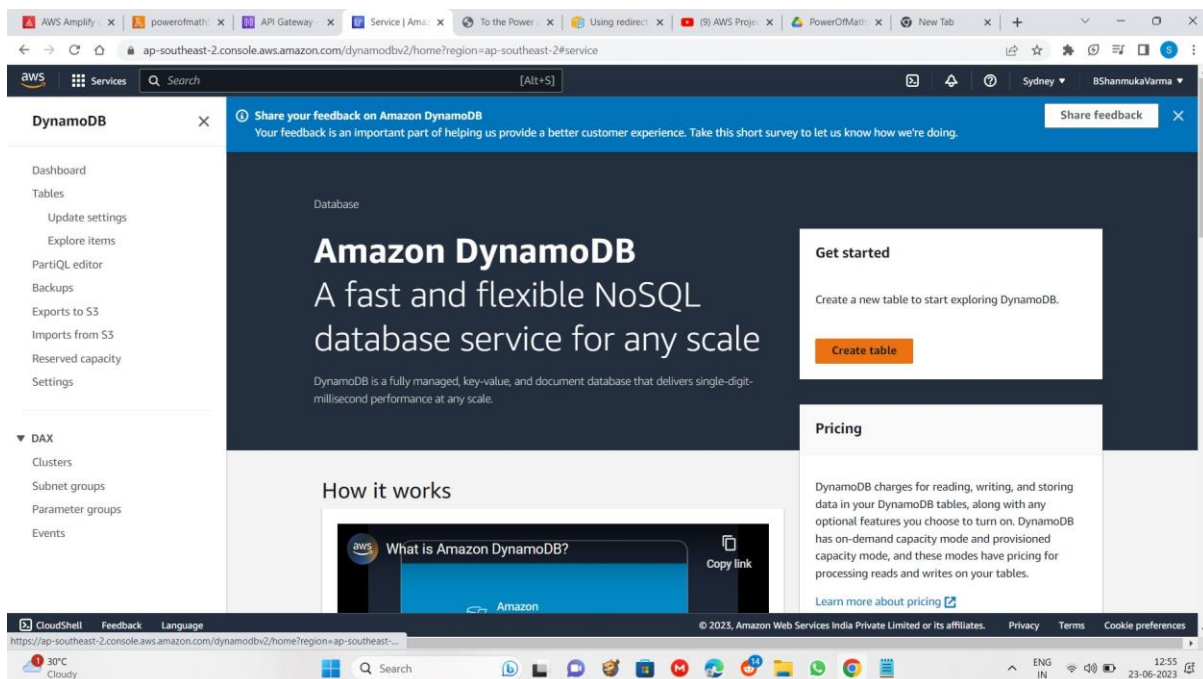
Confirming changes made



Post method overview



Using DynamoDB as database-



Creating a new table in DynamoDB

The screenshot shows the 'Create table' page in the AWS Management Console. The browser address bar shows the URL: `ap-southeast-2.console.aws.amazon.com/dynamodbv2/home?region=ap-southeast-2#create-table`. The page has a blue header with a feedback survey. The main content area is titled 'Create table' and contains a 'Table details' section. The 'Table name' field is 'powerofmath'. The 'Partition key' is 'ID' with a 'String' data type. The 'Sort key - optional' field is empty. The bottom of the page shows the AWS logo, 'CloudShell', 'Feedback', 'Language', and a copyright notice for 2023.

Table details [info](#)

DynamoDB is a schemaless database that requires only a table name and a primary key when you create the table.

Table name
This will be used to identify your table.

Between 3 and 255 characters, containing only letters, numbers, underscores (_), hyphens (-), and periods (.).

Partition key
The partition key is part of the table's primary key. It is a hash value that is used to retrieve items from your table and allocate data across hosts for scalability and availability.

1 to 255 characters and case sensitive.

Sort key - optional
You can use a sort key as the second part of a table's primary key. The sort key allows you to sort or search among all items sharing the same partition key.

1 to 255 characters and case sensitive.

Store the ARM URL for future use

The screenshot shows the 'View table' page in the AWS Management Console for the 'powerofmath' table. The browser address bar shows the URL: `ap-southeast-2.console.aws.amazon.com/dynamodbv2/home?region=ap-southeast-2#table?name=powerofmath`. The page has a blue header with a feedback survey. The main content area is titled 'General information' and contains a table with details about the table. The 'Table status' is 'Active'. The 'Additional info' section shows the 'Table class' as 'DynamoDB Standard', 'Indexes' as '0 globals, 0 locals', 'DynamoDB stream' as 'Off', 'Time to Live (TTL)' as 'Off', 'Replication Regions' as '0 Regions', 'Encryption' as 'Owned by Amazon', 'Date created' as 'June 23, 2023, 12:55:56 (UTC+05:30)', and 'Deletion protection' as 'Off'. The 'Items summary' section shows the 'Item count' as '0', 'Table size' as '0 bytes', and 'Average item size' as '0 bytes'. The bottom of the page shows the AWS logo, 'CloudShell', 'Feedback', 'Language', and a copyright notice for 2023.

General information

Partition key ID (String)	Sort key -	Capacity mode Provisioned	Table status Active
Alarms No active alarms	Point-in-time recovery (PITR) Off		

Additional info

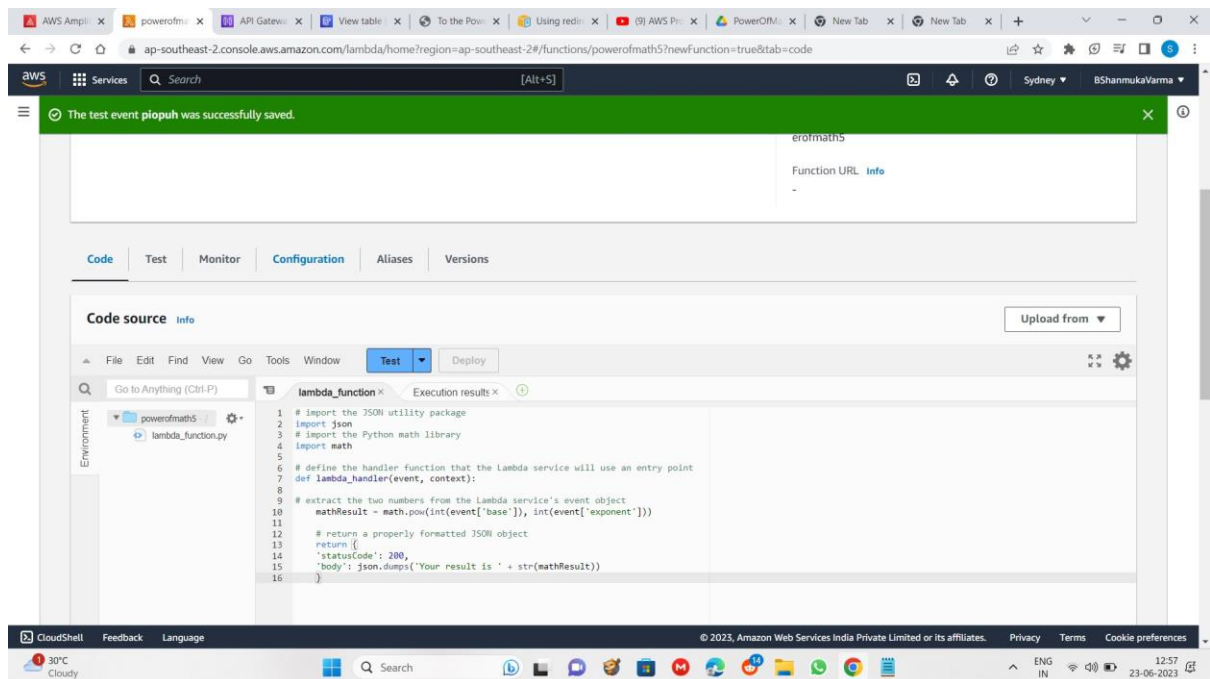
Table class DynamoDB Standard	Indexes 0 globals, 0 locals	DynamoDB stream Off	Time to Live (TTL) Off
Replication Regions 0 Regions	Encryption Owned by Amazon	Date created June 23, 2023, 12:55:56 (UTC+05:30)	Deletion protection Off

Amazon Resource Name (ARN)
`arn:aws:dynamodb:ap-southeast-2:785312289460:table/powerofmath`

Items summary
DynamoDB updates the following information approximately every six hours.

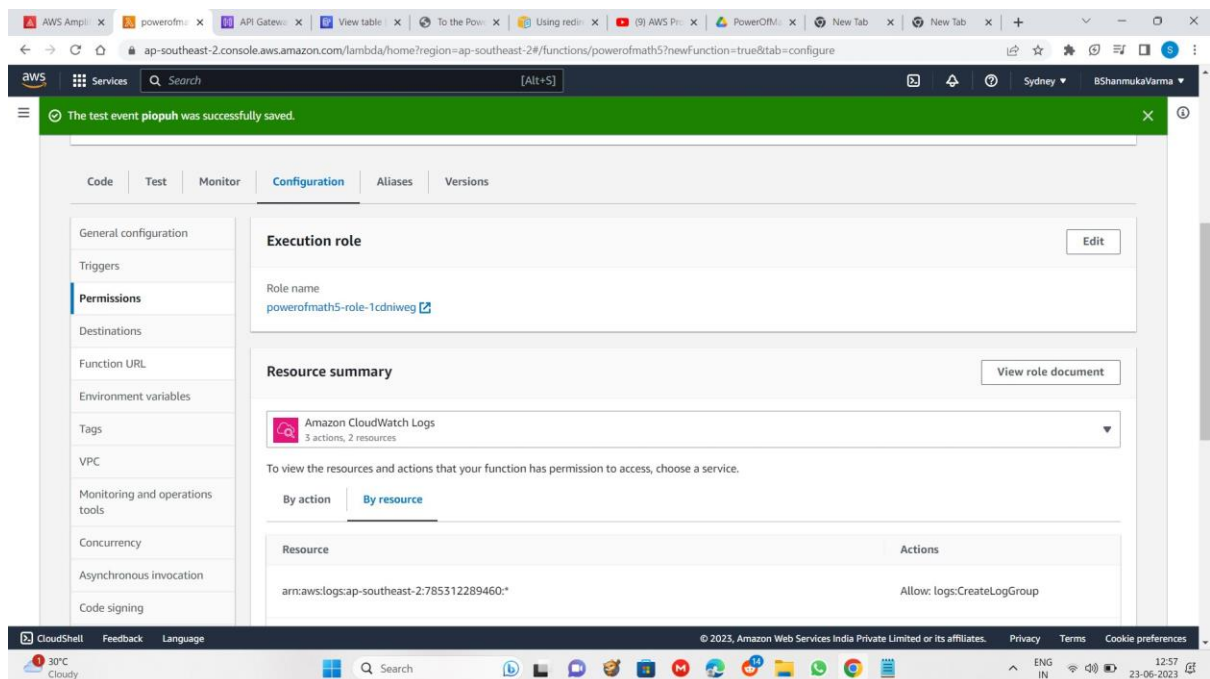
Item count 0	Table size 0 bytes	Average item size 0 bytes
-----------------	-----------------------	------------------------------

Here we have to make some changes to the lambda function that we previously discussed



We have to change some configurations here to lambda accessible to dynamodb

Changing configurations



Using IAM to give permissions-

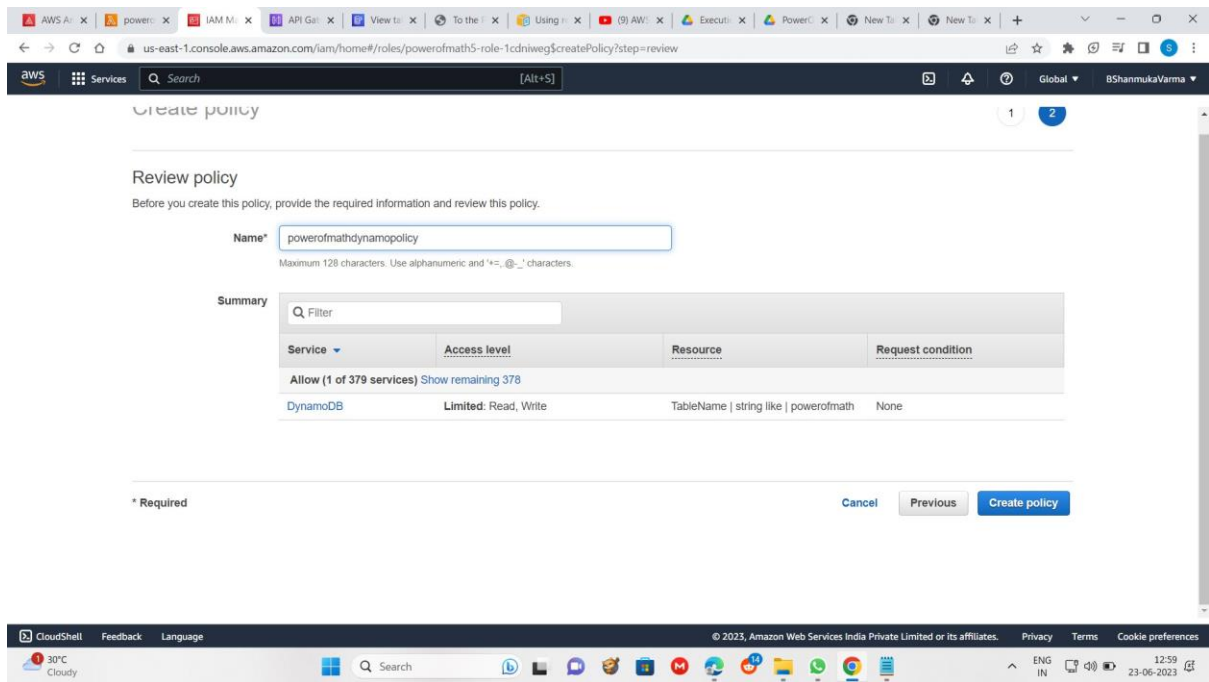
The screenshot shows the AWS IAM console interface. The left sidebar contains the 'Identity and Access Management (IAM)' menu with options like Dashboard, Access management, Users, Roles, Policies, Identity providers, Account settings, Access reports, Access analyzer, Archive rules, Analyzers, Settings, Credential report, Organization activity, and Service control policies (SCPs). The main content area displays the details for the role 'powerofmath5-role-1cdniweg'. The 'Summary' tab is active, showing the creation date (June 23, 2023, 12:45 UTC+05:30), last activity (None), ARN (arn:aws:iam::785312289460:role/service-role/powerofmath5-role-1cdniweg), and maximum session duration (1 hour). Below the summary, there are tabs for Permissions, Trust relationships, Tags, Access Advisor, and Revoke sessions. The 'Permissions policies (1)' section shows a table with one policy attached: 'AWSLambdaBasicExecutionRole-a90a3e19b-47a6-bbf5-3ca3d3b51087', which is 'Customer managed'. The bottom of the screen shows the Windows taskbar with various application icons and the system clock indicating 12:57 on 23-06-2023.

Creating policy

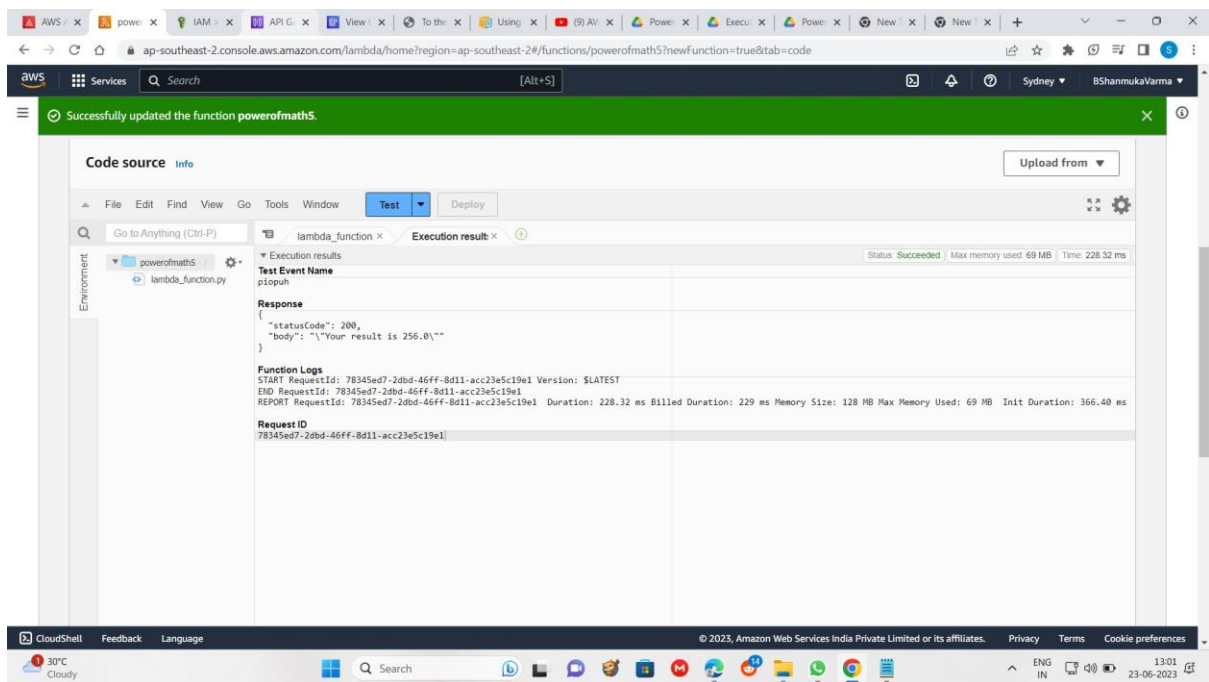
The screenshot shows the 'Create policy' page in the AWS IAM console. The page has two tabs: 'Visual editor' and 'JSON'. The 'JSON' tab is selected, and the policy document is displayed in a code editor. The policy is named 'powerofmath5-role-1cdniweg\$createPolicy' and is in the 'step=edit' state. The JSON document defines a policy with the following structure:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "VisualEditor0",
      "Effect": "Allow",
      "Action": [
        "dynamodb:PutItem",
        "dynamodb:DeleteItem",
        "dynamodb:GetItem",
        "dynamodb:Scan",
        "dynamodb:Query",
        "dynamodb:UpdateItem"
      ],
      "Resource": "arn:aws:dynamodb:ap-southeast-2:785312289460:table/powerofmath"
    }
  ]
}
```

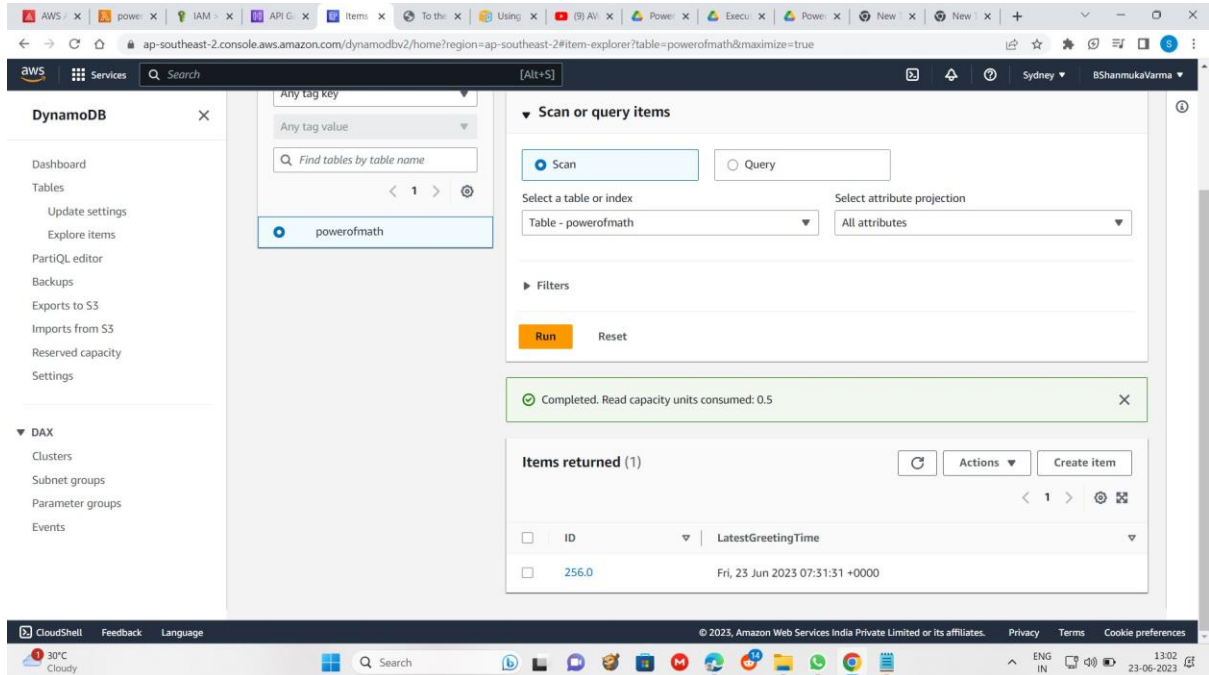
The bottom of the screen shows the Windows taskbar with various application icons and the system clock indicating 12:59 on 23-06-2023.



Testing the weather we getting the results or not-



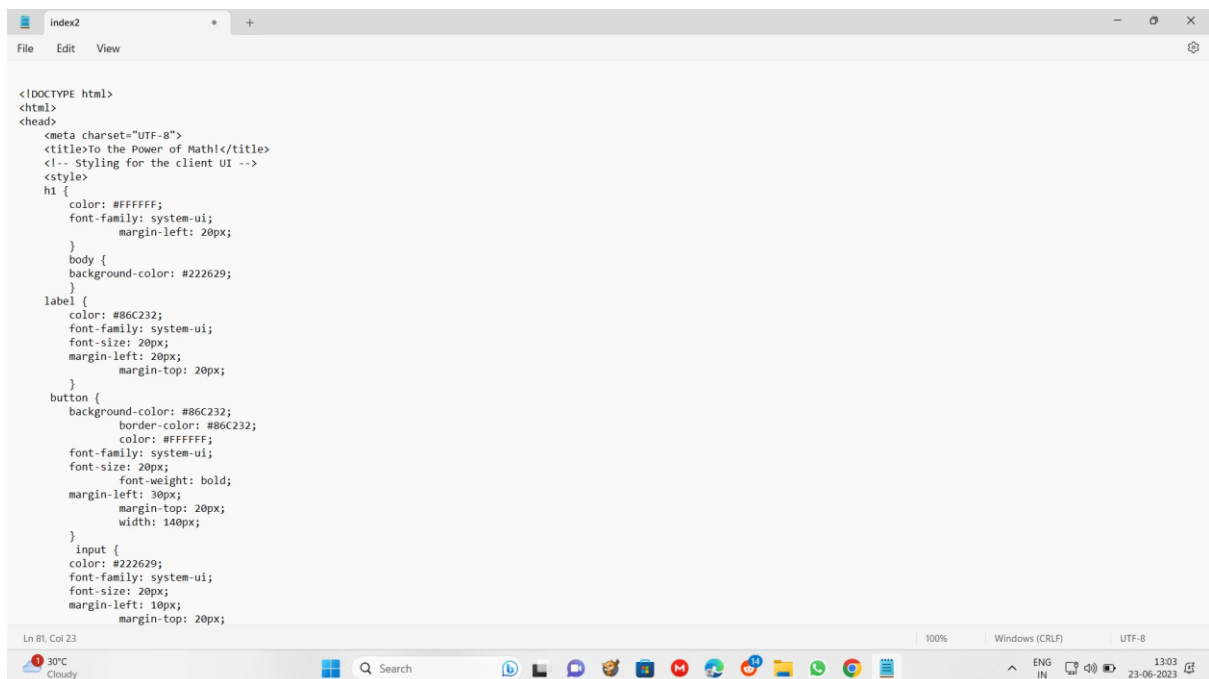
Result stored in DynamoDB



The screenshot displays the AWS Management Console for DynamoDB. The left sidebar shows the navigation menu with options like Dashboard, Tables, Update settings, Explore items, PartiQL editor, Backups, Exports to S3, Imports from S3, Reserved capacity, Settings, DAX, Clusters, Subnet groups, Parameter groups, and Events. The main content area is titled 'Scan or query items' and shows a successful scan of the 'powerofmath' table. The results section, 'Items returned (1)', displays a single item with the following details:

ID	LatestGreetingTime
256.0	Fri, 23 Jun 2023 07:31:31 +0000

Updating website's index code-

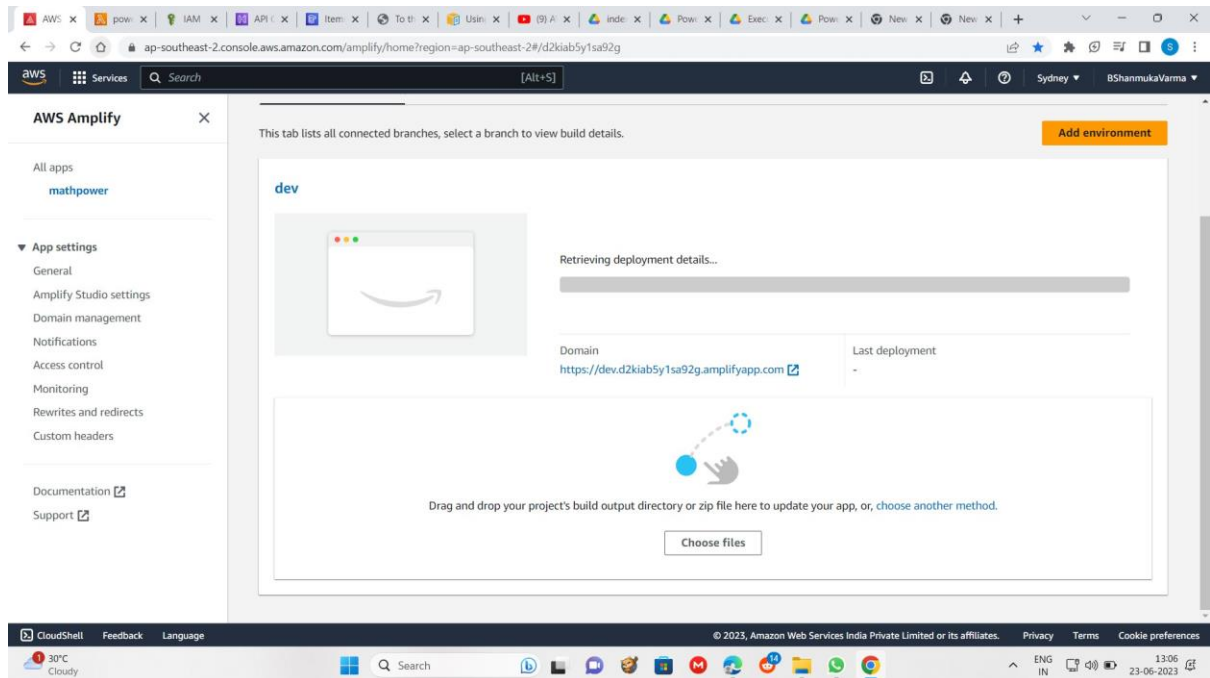


The screenshot shows a code editor with the file 'index2.html'. The code is as follows:

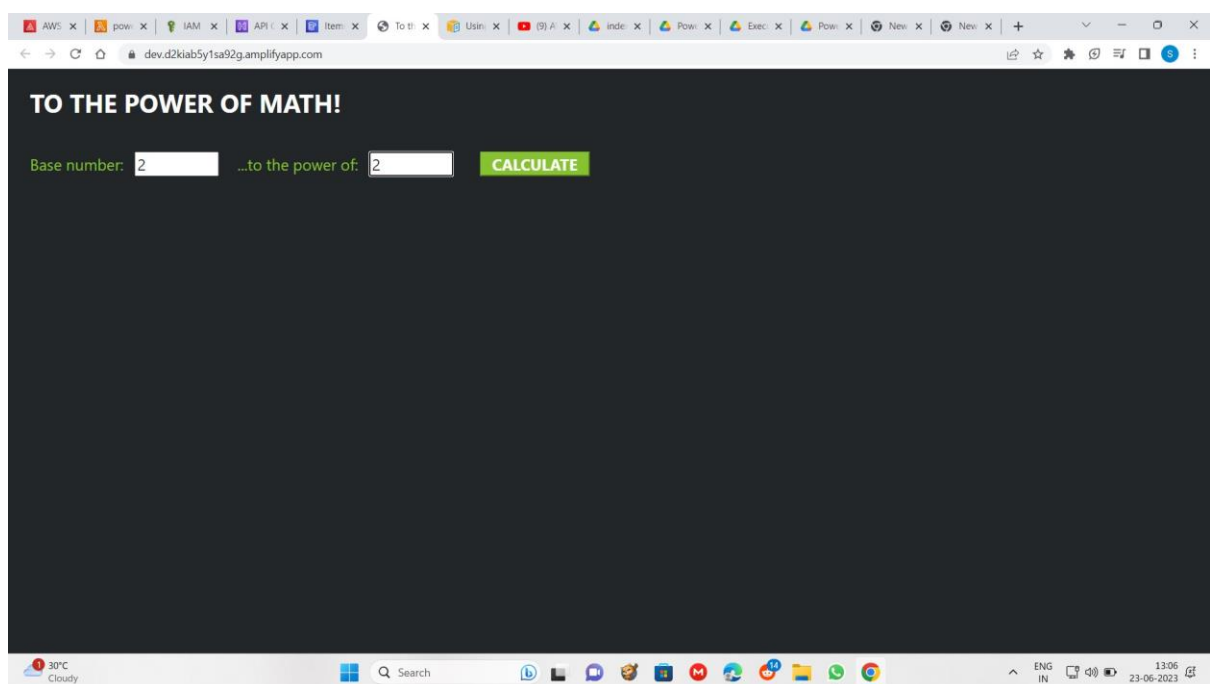
```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>To the Power of Math!</title>
  <!-- Styling for the client UI -->
  <style>
    h1 {
      color: #FFFFFF;
      font-family: system-ui;
      margin-left: 20px;
    }
    body {
      background-color: #222629;
    }
    label {
      color: #86C232;
      font-family: system-ui;
      font-size: 20px;
      margin-left: 20px;
      margin-top: 20px;
    }
    button {
      background-color: #86C232;
      border-color: #86C232;
      color: #FFFFFF;
      font-family: system-ui;
      font-size: 20px;
      font-weight: bold;
      margin-left: 30px;
      margin-top: 20px;
      width: 140px;
    }
    input {
      color: #222629;
      font-family: system-ui;
      font-size: 20px;
      margin-left: 10px;
      margin-top: 20px;
    }
  </style>

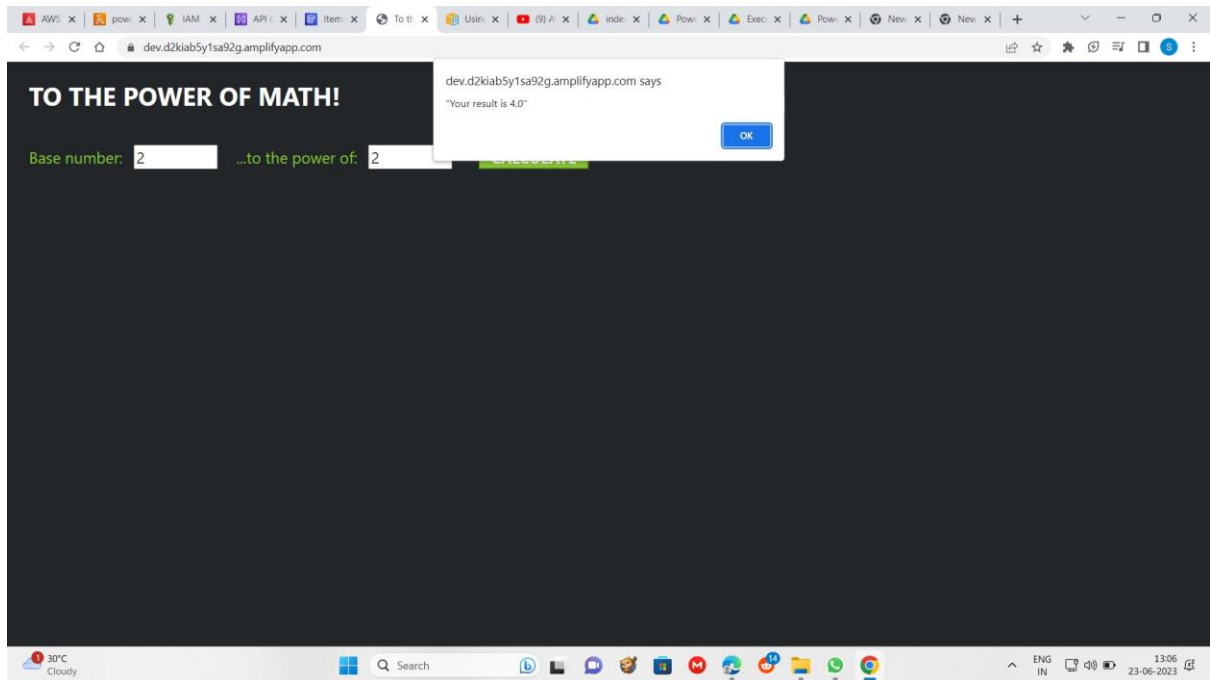
```

Now, we upload that to amplify



Deployed dynamic website-





Codes used-

Dynamic website code

```
<!DOCTYPE html>

<html>

<head>

  <meta charset="UTF-8">

  <title>To the Power of Math!</title>

  <!-- Styling for the client UI -->

  <style>

h1 {

  color: #FFFFFF;

  font-family: system-ui;

      margin-left: 20px;

}

  body {

    background-color: #222629;

  }

label {

  color: #86C232;

  font-family: system-ui;

  font-size: 20px;

  margin-left: 20px;

      margin-top: 20px;

}

button {

  background-color: #86C232;

      border-color: #86C232;

      color: #FFFFFF;

  font-family: system-ui;
```

```

font-size: 20px;
        font-weight: bold;
margin-left: 30px;
        margin-top: 20px;
        width: 140px;
    }
    input {
color: #222629;
font-family: system-ui;
font-size: 20px;
margin-left: 10px;
        margin-top: 20px;
        width: 100px;
    }
</style>
<script>
    // callAPI function that takes the base and exponent numbers as parameters
    var callAPI = (base,exponent)=>{
        // instantiate a headers object
        var myHeaders = new Headers();
        // add content type header to object
        myHeaders.append("Content-Type", "application/json");
        // using built in JSON utility package turn object to string and store in a variable
        var raw = JSON.stringify({"base":base,"exponent":exponent});
        // create a JSON object with parameters for API call and store in a variable
        var requestOptions = {
            method: 'POST',
            headers: myHeaders,
            body: raw,
            redirect: 'follow'
        };

```



```

    // make API call with parameters and use promises to get response
    fetch("YOUR API GATEWAY ENDPOINT", requestOptions)
      .then(response => response.text())
      .then(result => alert(JSON.parse(result).body))
      .catch(error => console.log('error', error));
  }
</script>
</head>
<body>
  <h1>TO THE POWER OF MATH!</h1>
  <form>
    <label>Base number:</label>
    <input type="text" id="base">
    <label>...to the power of:</label>
    <input type="text" id="exponent">
    <!-- set button onClick method to call function we defined passing input values as parameters -->
    <button type="button"
      onclick="callAPI(document.getElementById('base').value,document.getElementById('exponent').value)">CALCULATE</button>
  </form>
</body>
</html>

```

Lambda original code-

```
# import the JSON utility package
```

```
import json
```

```
# import the Python math library
```

```
import math
```

```
# define the handler function that the Lambda service will use as an entry point
```

```
def lambda_handler(event, context):
```

```
# extract the two numbers from the Lambda service's event object
```

```
    mathResult = math.pow(int(event['base']), int(event['exponent']))
```

```
# return a properly formatted JSON object
```

```
    return {
```

```
        'statusCode': 200,
```

```
        'body': json.dumps('Your result is ' + str(mathResult))
```

```
    }
```

PowerOfMathFunction – Lambda

```
# import the JSON utility package
import json

# import the Python math library
import math

# import the AWS SDK (for Python the package name is boto3)
import boto3

# import two packages to help us with dates and date formatting
from time import gmtime, strftime

# create a DynamoDB object using the AWS SDK
dynamodb = boto3.resource('dynamodb')

# use the DynamoDB object to select our table
table = dynamodb.Table('PowerOfMathDatabase')

# store the current time in a human readable format in a variable
now = strftime("%a, %d %b %Y %H:%M:%S +0000", gmtime())

# define the handler function that the Lambda service will use as an entry point
def lambda_handler(event, context):

# extract the two numbers from the Lambda service's event object
    mathResult = math.pow(int(event['base']), int(event['exponent']))

# write result and time to the DynamoDB table using the object we instantiated and save response in
# a variable
    response = table.put_item(
        Item={
            'ID': str(mathResult),
            'LatestGreetingTime': now
        }
```

```
}}
```

```
# return a properly formatted JSON object
```

```
return {  
    'statusCode': 200,  
    'body': json.dumps('Your result is ' + str(mathResult))  
}
```

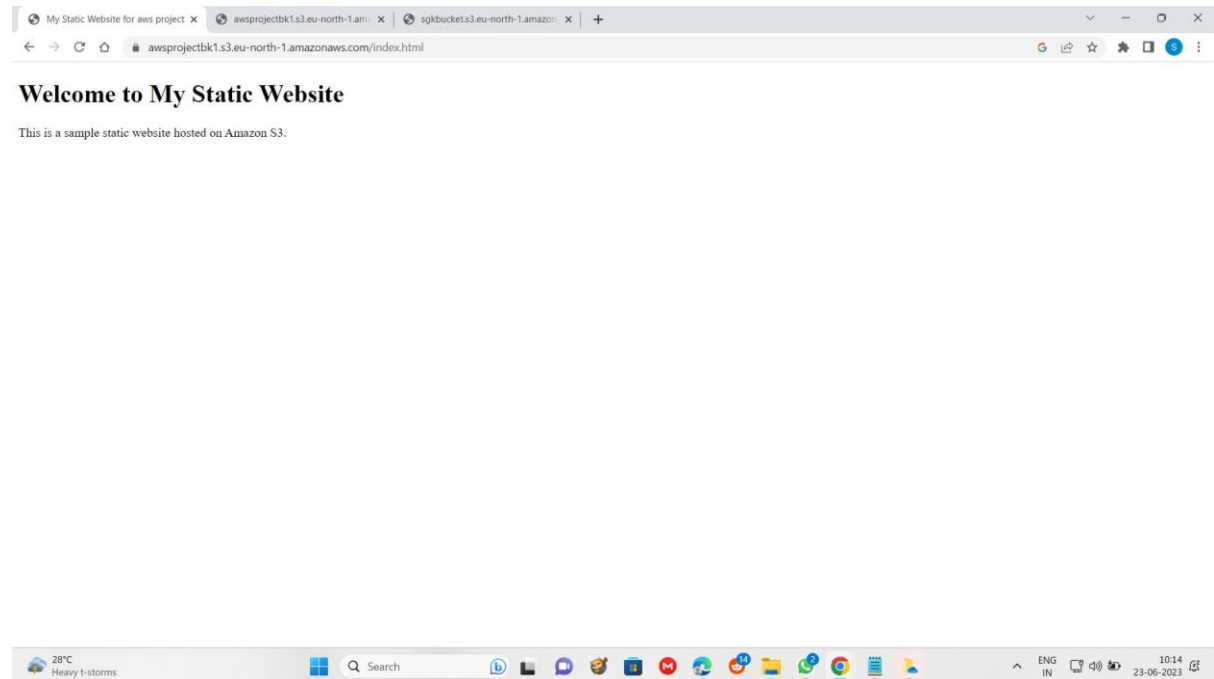
Execution Role Policy JSON.txt -

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "VisualEditor0",
      "Effect": "Allow",
      "Action": [
        "dynamodb:PutItem",
        "dynamodb:DeleteItem",
        "dynamodb:GetItem",
        "dynamodb:Scan",
        "dynamodb:Query",
        "dynamodb:UpdateItem"
      ],
      "Resource": "YOUR-TABLE-ARN"
    }
  ]
}
```

Conclusion

We deployed a serverless Static and a serverless dynamic website using AWS

Hosted static website



Hosted dynamic website

