

Serverless Web Application in AWS



Venkatgiri Sasanapuri · [Follow](#)

10 min read · Apr 21, 2024

[Listen](#)

[Share](#)

[More](#)

[Open in app](#) ↗



[Search](#)



Serverless Project Workflow



Image created using draw.io

The idea of using serverless comes when we want to get rid of the hardware limitations, server provisioning, and software installations needed to run our application.

Before jumping into the project, the prerequisites are:

1. AWS Account.
2. Visual Studio Code or any similar code editors.

We are creating a serverless web application using AWS services. As mentioned in the above diagram, we are using:

Amazon s3: To store static files. i.e., html, javascript files. We enable static hosting with the necessary permissions.

API Gateway: An API gateway manages incoming requests and routes them based on key factors such as request path, headers, and query parameters. In short, when we submit our details on the webpage, the API gateway manages to send this data to backend services like Lambda.

AWS Lambda: AWS lambda contains the backend code required for our application. This service is the backbone, and the name serverless is said because of this. We are neither creating servers nor installing any software here. We select the software environment needed and add our code.

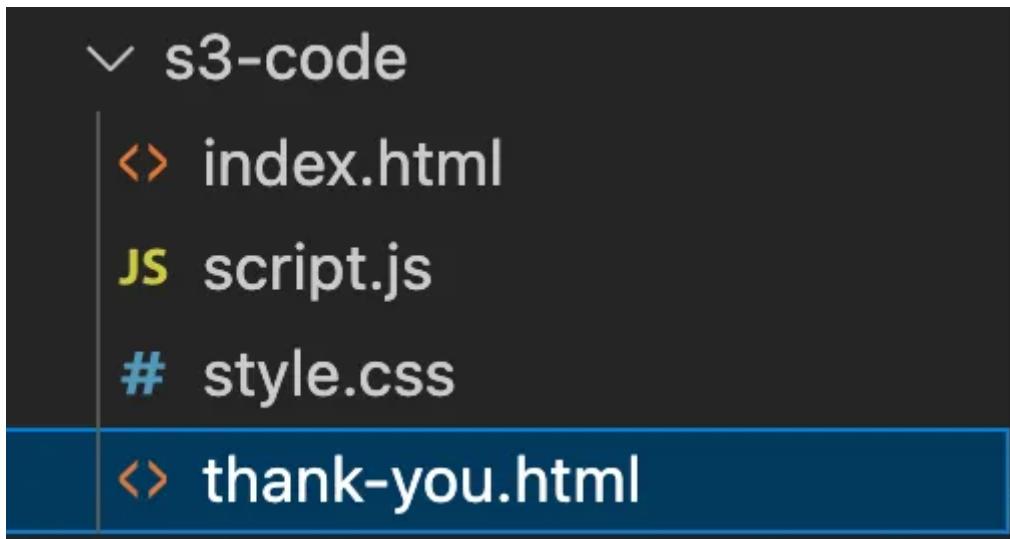
DynamoDB: This is a no-SQL(non-relational) database where the data is stored. For ex., When we fill out the HTML form, the data is processed as shown in the diagram and stored in the database.

Identity Access and Management (IAM): IAM is a service that helps manage resource access. We can perform user administration, resource administration, etc., here.

Cloudwatch: This is a monitoring tool that helps manage and monitor resources effectively.

Step 1: Create a Static Website using S3:

Let's start creating our bucket. I will list the files and the code I used to store them in s3.



Index.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Contact Form</title>
    <link rel="stylesheet" type="text/css" href="style.css">
</head>
<body>
    <div class="container">
        <h1>Contact Form</h1>
        <form id="contact-form">
            <label for="name">Name:</label>
            <input type="text" id="name" name="name" required>

            <label for="email">Email:</label>
            <input type="email" id="email" name="email" required>

            <label for="subject">Subject:</label>
            <input type="text" id="subject" name="subject" required>

            <label for="message">Message:</label>
            <textarea id="message" name="message" rows="5" required></textarea>

            <button type="submit">Submit</button>
        </form>
    </div>

    <script src="script.js"></script>
</body>
</html>
```

style.css

```
body {
    margin: 0;
    padding: 0;
    font-family: Arial, sans-serif;
    background-size: cover; /* Cover the entire background */
    background-position: center; /* Center the background image */
}

.container {
    max-width: 500px;
    margin: 50px auto; /* Center the container vertically and horizontally */
    padding: 20px;
    background-color: rgba(255, 255, 255, 0.8); /* Add a semi-transparent white background */
    border-radius: 8px; /* Add some rounded corners */
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1); /* Add a subtle shadow */
}

h1 {
    text-align: center;
}

form label {
    display: block;
    margin-bottom: 8px;
}

form input,
form textarea {
    width: 100%;
    padding: 10px;
    margin-bottom: 15px;
    border: 1px solid #ccc; /* Add a border to input fields */
    border-radius: 4px; /* Add some rounded corners to input fields */
    box-sizing: border-box; /* Ensure padding is included in the width */
}

form button {
    width: 100%;
    padding: 10px 20px;
    background-color: #007bff;
    color: #fff;
    border: none;
    border-radius: 4px;
    cursor: pointer;
}

form button:hover {
    background-color: #0069d9;
}
```



script.js

```
document.getElementById('contact-form').addEventListener('submit', function(event) {
  event.preventDefault(); // Prevent form submission

  // Validate form inputs
  var name = document.getElementById('name').value.trim();
  var email = document.getElementById('email').value.trim();
  var subject = document.getElementById('subject').value.trim();
  var message = document.getElementById('message').value.trim();

  // Email validation using a regular expression
  var emailRegex = /^[\S+@\S+\.\S+$/;
  if (!name || !email || !subject || !message || !emailRegex.test(email)) {
    alert('Please fill in all fields with valid inputs.');
    return;
  }

  // Construct the form data
  var formData = {
    name: name,
    email: email,
    subject: subject,
    message: message
  };

  // Perform form submission
  submitForm(formData);
});

function submitForm(formData) {
  // Make an API request to the backend (API Gateway) for form submission
  fetch('https://xxxxxxxx.execute-api.us-east-1.amazonaws.com/dev/submit', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json'
    },
    body: JSON.stringify(formData)
  })
  .then(function(response) {
    if (response.ok) {
      // Redirect to the thank you page
      window.location.href = 'thank-you.html';
    } else {
      throw new Error('Form submission failed.');
    }
  })
  .catch(function(error) {
    console.error(error);
    alert('Form submission failed. Please try again later.');
  });
}
```

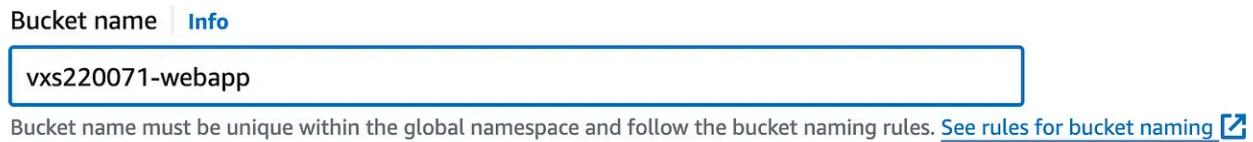
```
});  
}
```

thank-you.html

```
<!DOCTYPE html>  
<html>  
<head>  
<title>Thank You</title>  
<link rel="stylesheet" type="text/css" href="style.css">  
</head>  
<body>  
<div class="container">  
<h1>Thank You!</h1>  
<p>Your message has been submitted successfully.</p>  
</div>  
</body>  
</html>
```

Let's create an s3 bucket and then enable the static website.

1. Open s3 in the AWS Console. Create a bucket with the name of your choice, leaving rest of the options to default.



2. Once the bucket is created, upload the index.html, script.js, style.css and thank-you.html files.

vxs220071-webapp Info Publicly accessible

Objects	Properties	Permissions	Metrics	Management	Access Points
Objects (4) <small>Info</small>					
<input type="button" value="C"/> <input type="button" value="Copy S3 URI"/> <input type="button" value="Copy URL"/> <input type="button" value="Download"/> <input type="button" value="Open"/> <input type="button" value="Delete"/> <input type="button" value="Actions"/> <input type="button" value="Create folder"/> <input type="button" value="Upload"/>					
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					
<input type="text"/> <small>Find objects by prefix</small>					
<input type="checkbox"/>	Name	Type	Last modified	Size	Storage class
<input type="checkbox"/>	index.html	html	April 19, 2024, 22:05:15 (UTC-05:00)	720.0 B	Standard
<input type="checkbox"/>	script.js	js	April 20, 2024, 21:55:51 (UTC-05:00)	1.6 KB	Standard
<input type="checkbox"/>	style.css	css	April 19, 2024, 22:05:16 (UTC-05:00)	1.1 KB	Standard
<input type="checkbox"/>	thank-you.html	html	April 20, 2024, 21:58:15 (UTC-05:00)	262.0 B	Standard

3. Open the permissions tab and scroll down to unblock the public access.

Edit Block public access (bucket settings) Info**Block public access (bucket settings)**

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 all your S3 buckets and 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 your buckets 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.

Cancel**Save changes**

4. Edit the bucket policy and add the bucket below code. Make sure to change the bucket name.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "PublicReadGetObject",  
            "Effect": "Allow",  
            "Principal": "*",  
            "Action": "s3:GetObject",  
            "Resource": "arn:aws:s3:::<your-bucket>/*"  
        }  
    ]  
}
```

5. Go to the properties tab and scroll down till the end. Enable static website hosting. Give the name of your index document. It's index.html. Save the changes.

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](#)

ⓘ 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

6. Click on the url created after enabling the static website.

The screenshot shows the AWS Static Website Hosting configuration page. It displays the following details:

- Static website hosting**: A section header.
- Use this bucket to host a website or redirect requests. [Learn more](#)
- Static website hosting
- Enabled
- Hosting type
- Bucket hosting
- Bucket website endpoint
- When you configure your bucket as a static website, the website is available at the AWS Region-specific website endpoint of the bucket. [Learn more](#)
- A link to the endpoint: <http://vxs220071-webapp.s3-website-us-east-1.amazonaws.com>

7. Congrats, we have created our static website. We have not set up the backend framework to store data in the database. So, even though I try to submit, the data won't be stored.

The screenshot shows a web browser window with the following details:

- Address bar: Not Secure vxs220071-webapp.s3-website-us-east-1.amazonaws.com
- Form title: Contact Form
- Fields:
 - Name: Venkata Giri Sasanapuri
 - Email: thisisstaticwebsite@gmail.com
 - Subject: Static Website
 - Message: We have learnt till the creation of static website.
- Submit button

Step 2: Working on lambda and Dynamo DB

1. In this step, we create a database to store the form data.

2. Let's create a DynamoDB with the name of your choice and the Partition key as mentioned. Leave the rest of the options to default and create the table.

DynamoDB > Tables > Create table

Create table

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.

2. Let's create the lambda function. Give a name of your choice and leave the rest of the options to default. This will also create a lambda role with basic execution permissions.

Create function Info

Choose one of the following options to create your function.

Author from scratch

Start with a simple Hello World example.

Use a blueprint

Build a Lambda application from sample code and configuration presets for common use cases.

Container image

Select a container image to deploy.

Basic information

Function name

Enter a name that describes the purpose of your function.

vxs220071-webapp-function

Use only letters, numbers, hyphens, or underscores with no spaces.

Runtime Info

Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.

Node.js 20.x



Architecture Info

Choose the instruction set architecture you want for your function code.

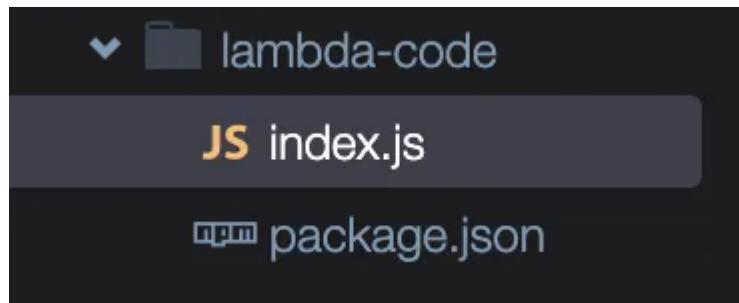
x86_64

arm64

3. We must do certain steps to upload the code to lambda.

Let's add our code to lambda. We have two options for uploading our code: s3 and direct upload. Let's use s3 to follow best practices.

Before doing that, open the VS code and save the code as shown below.



index.js

Make sure to change the DynamoDB table name.

```
const AWS = require('aws-sdk');
const dynamodb = new AWS.DynamoDB.DocumentClient();
const { v4: uuidv4 } = require('uuid');

exports.handler = async (event) => {
  try {
    console.log('Raw input data:', event); // Add this line to log the raw input
  }
```

```

const formData = {
  name: event.name,
  email: event.email,
  subject: event.subject,
  message: event.message,
};

const item = {
  SubmissionId: generateUUID(), // Generate a UUID
  ...formData, // Use the form data as attributes
};

// Store the form data in DynamoDB
await storeFormData(item);

return {
  statusCode: 200,
  body: JSON.stringify({ message: 'Form submitted successfully' }),
};
} catch (error) {
  console.error(error);
  return {
    statusCode: 500,
    body: JSON.stringify({ message: 'Error submitting the form' }),
  };
}
};

async function storeFormData(item) {
  const params = {
    TableName: '<dynamodbtable>',
    Item: item,
  };

  await dynamodb.put(params).promise();
}

function generateUUID() {
  return uuidv4();
}

```

package.json

```
{
  "name": "serverless-contact-form",
  "version": "1.0.0",
}
```

```
"description": "Lambda function for handling the serverless contact form",
"main": "index.js",
"dependencies": {
    "aws-sdk": "^2.1386.0"
}
}
```

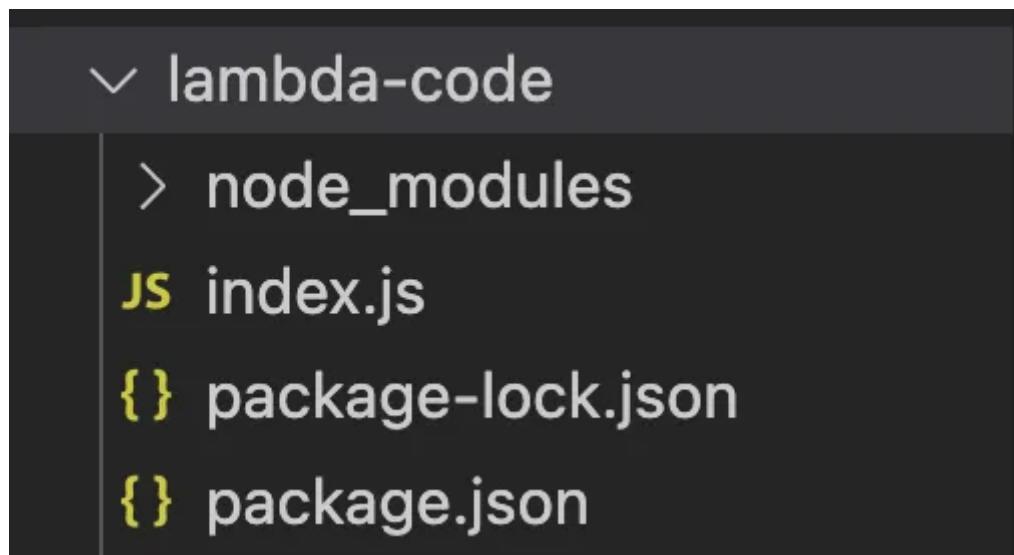
Open the terminal in VS code and make sure you are in the *lambda-code* directory. I had to move to *lambda-code* by typing the cd command as my parent directory is different.

```
venkatgirisasanapuri@Venkatgiris-MacBook-Air Javascript-testing.js % cd lambda-code
venkatgirisasanapuri@Venkatgiris-MacBook-Air lambda-code %
```

Run the npm install command to install the modules required to run our code.

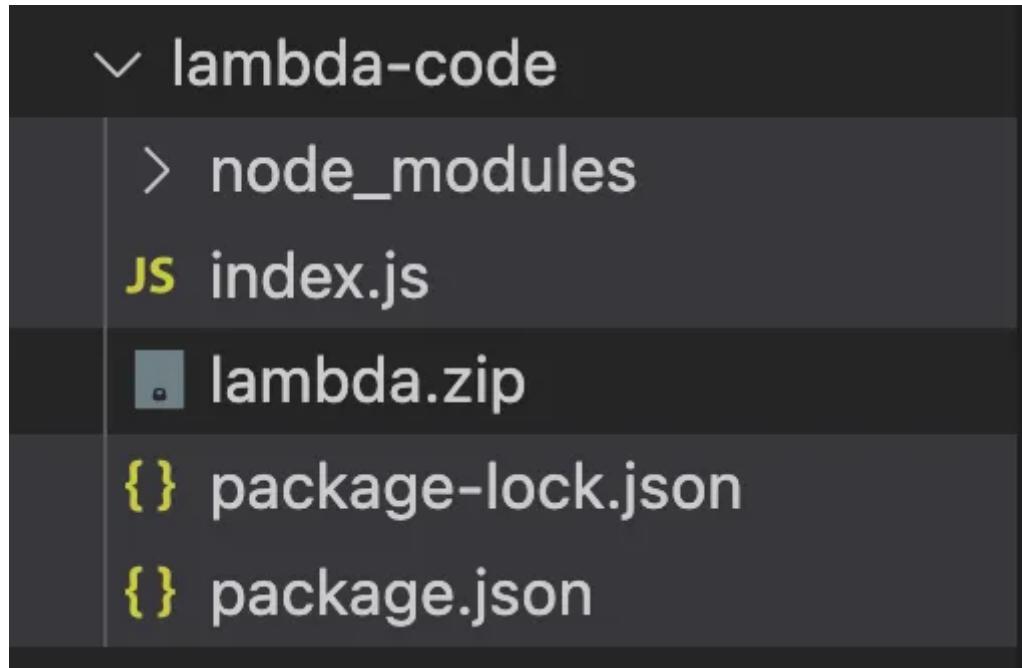
```
npm install
```

After installation, the folder structure should be like,



We should zip all the files and folders within the lambda code. We can use the command given below: (Make sure you are in the *lambda-code* directory)

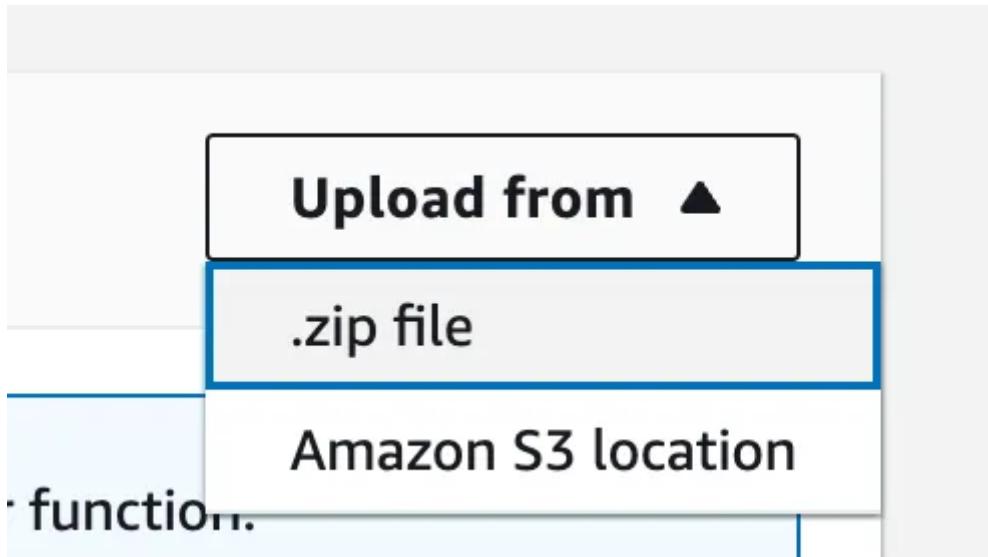
```
zip -r lambda.zip .
```



4. Let's create an s3 bucket to upload this code. Create the bucket with the name of your choice and with the default options, and upload the zip file.

The screenshot shows the Amazon S3 console with the URL 'Amazon S3 > Buckets > vxs220071-lambda-code'. The 'Info' tab is selected. On the left, there are tabs for 'Objects', 'Properties', 'Permissions', 'Metrics', 'Management', and 'Access Points'. The 'Objects' tab is active. At the top of the main area, there's a toolbar with buttons for 'Actions' (with dropdown menus for 'Copy S3 URI', 'Copy URL', 'Download', 'Open', 'Delete', 'Create folder', and 'Upload'), a search bar ('Find objects by prefix'), and navigation controls ('< 1 >'). Below the toolbar, a table lists the object 'lambda.zip'. The table has columns for 'Name' (lambda.zip), 'Type' (zip), 'Last modified' (April 20, 2024, 11:07:44 (UTC-05:00)), 'Size' (13.3 MB), and 'Storage class' (Standard).

5. Copy the S3 URI, open lambda, and select upload from the Amazon s3 location.



Let's test our backend code by saving our data manually using the test option in Lambda. We will create a test event.

The screenshot shows the AWS Lambda function configuration interface with the 'Test' tab selected. At the top, there are tabs for 'Code', 'Test' (which is active), 'Monitor', 'Configuration', 'Aliases', and 'Versions'. Below the tabs, there is a section titled 'Test event' with 'Info' and 'Save' buttons. A note says 'To invoke your function without saving an event, configure the JSON event, then choose Test.' There are two options: 'Create new event' (selected) and 'Edit saved event'. Under 'Event name', the value 'backend-testing' is entered. A note below says 'Maximum of 25 characters consisting of letters, numbers, dots, hyphens and underscores.'

Add the below json-code and save the test event for future uses.

```
{  
  "name": "myname",  
  "email": "mymail@example.com",  
  "subject": "lambda-testing",  
  "message": "A message from lambda"  
}
```

Event name
backend-testing

Maximum of 25 characters consisting of letters, numbers, dots, hyphens and underscores.

Event sharing settings
 Private
 This event is only available in the Lambda console and to the event creator. You can configure a total of 10. [Learn more](#)

Shareable
 This event is available to IAM users within the same account who have permissions to access and use shareable events. [Learn more](#)

Template - *optional*
hello-world

Event JSON

```

1 ~ [{}]
2   "name": "myname",
3   "email": "myemail@example.com",
4   "subject": "Lambda-testing",
5   "message": "A message from lambda"
6 ]

```

Format JSON

6. Test the event. This should save the details in the DynamoDB, but due to permission issues, I got an error saying the lambda role doesn't have permission to save data.

Executing function: succeeded ([logs](#))

▼ Details

The area below shows the last 4 KB of the execution log.

```
{
  "statusCode": 500,
  "body": "{\"message\":\"Error submitting the form\"}"
}
```

The section below shows the logging calls in your code. [Click here](#) to view the corresponding CloudWatch log group.

```

email: 'myemail@example.com',
subject: 'Lambda-testing',
message: 'A message from lambda'
}
2024-04-20T16:41:03.285Z 96b2a334-88d6-482b-a1d3-94ed8dde3acc ERROR AccessDeniedException: User: arn:aws:sts::506960532205:assumed-role/vxs220071-
webapp-function-role-pvgalbr3/vxs220071-webapp-function is not authorized to perform: dynamodb:PutItem on resource: arn:aws:dynamodb:us-east-
1:506960532205:table/vxs220071-webapp-db because no identity-based policy allows the dynamodb:PutItem action
at Request.extractError (/var/task/node_modules/aws-sdk/lib/protocol/json.js:80:27)
at Request.callListeners (/var/task/node_modules/aws-sdk/lib/sequential_executor.js:106:20)
at Request.emit (/var/task/node_modules/aws-sdk/lib/sequential_executor.js:78:10)
at Request.emit (/var/task/node_modules/aws-sdk/lib/request.js:686:14)

```

Let's modify the permissions of the lambda role.

Click on the Configuration tab and then on Permissions from the left menu to view the role. Click on the role name.

Code | Test | Monitor | **Configuration** | Aliases | Versions

General configuration	Execution role	Edit View role document
Triggers		
Permissions	Role name vxs220071-webapp-function-role-pvgalbr3	
Destinations		

vxs220071-webapp-function-role-pvgalbr3 [Info](#)[Delete](#)**Summary**[Edit](#)

April 20, 2024, 10:07 (UTC-05:00)

ARN

arn:aws:iam::506960532205:role/service-role/vxs220071-webapp-function-role-pvgalbr3

Last activity

-

Maximum session duration

1 hour

[Permissions](#)[Trust relationships](#)[Tags](#)[Access Advisor](#)[Revoke sessions](#)**Permissions policies (1) [Info](#)**

You can attach up to 10 managed policies.

[Simulate](#) [Remove](#)[Add permissions](#)

Filter by Type

 Search

All types

< 1 >

 Policy name

Type

Attached entities

[AWSLambdaBasicExecutionRole-Of5...](#)

Customer managed

1

Click on Add Permissions and select Attach Policies. Search for Amazon DynamoDBFullAccess and Add permissions.

Other permissions policies (1/939) Dyna

Filter by Type

All types

4 matches

< 1 >

 Policy name

Type

Description

 [AmazonDynamoDBFullAccess](#)

AWS managed

Provides full access to Amazon Dynam...

 [AmazonDynamoDBReadOnlyAccess](#)

AWS managed

Provides read only access to Amazon D...

 [AWSLambdaDynamoDBExecutionRole](#)

AWS managed

Provides list and read access to Dynam...

 [AWSLambdaInvocation-DynamoDB](#)

AWS managed

Provides read access to DynamoDB Str...

[Cancel](#)[Add permissions](#)[Permissions](#)[Trust relationships](#)[Tags](#)[Access Advisor](#)[Revoke sessions](#)**Permissions policies (2) [Info](#)**[Simulate](#) [Remove](#)[Add permissions](#)

You can attach up to 10 managed policies.

Filter by Type

All types

 Search Policy name

Type

Attached entities

 [AmazonDynamoDBFullAccess](#)

AWS managed

2

 [AWSLambdaBasicExecutionRole-Of5...](#)

Customer managed

1

Let's open lambda to test whether the data is being saved. This time, my data is saved in DynamoDB.

The screenshot shows the AWS Lambda function configuration page. The 'Test' tab is selected. A green checkmark icon indicates the function executed successfully. The log output shows a JSON response with a status code of 200 and a body message "Form submitted successfully".

The screenshot also includes a separate window showing the AWS DynamoDB Items returned (1) table. The table has columns: SubmissionId (String), email, message, name, and subject. One item is listed with SubmissionId: 40a1b795-8ab6-46fd-8ad..., email: mymail@ex..., message: A message ..., name: myname, and subject: lambda-testing.

Step 3: Let's create the bridge to bring the data from S3 to Lambda. That is API Gateway.

1. Open API gateway in the console and choose the REST API type. Give the name of your choice and leave other options to default.

Create REST API

API details

New API

Create a new REST API.

Clone existing API

Create a copy of an API in this AWS account.

Import API

Import an API from an OpenAPI definition.

Example API

Learn about API Gateway with an example API.

API name

vxs220071-gateway

Description - *optional*

API endpoint type

Regional APIs are deployed in the current AWS Region. Edge-optimized APIs route requests to the nearest CloudFront Point of Presence. Private APIs are only accessible from VPCs.

Regional

[Cancel](#)

[Create API](#)

2. Click on Create resource and give the resource name. Enable CORS for the enhanced security.

Create resource

Resource details

Proxy resource [Info](#)

Proxy resources handle requests to all sub-resources. To create a proxy resource use a path parameter that ends with a plus sign, for example {proxy+}.

Resource path

Resource name

CORS (Cross Origin Resource Sharing) [Info](#)

Create an OPTIONS method that allows all origins, all methods, and several common headers.

[Cancel](#)

[Create resource](#)

3. Create method and select the method type POST, and select your lambda function. Leave the rest of the options to default and Create method.

Method details

Integration type

Lambda function

Integrate your API with a Lambda function.



HTTP

Integrate with an existing HTTP endpoint.



Mock

Generate a response based on API Gateway mappings and transformations.



AWS service

Integrate with an AWS Service.



VPC link

Integrate with a resource that isn't accessible over the public internet.



Lambda proxy integration

Send the request to your Lambda function as a structured event.

Lambda function

Provide the Lambda function name or alias. You can also provide an ARN from another account.

us-east-1



arn:aws:lambda:us-east-1:506960532205:function:vxs2



4. Select the Enable CORS option(click on the resource submit). Check the boxes as shown and provide your s3 static website URL. We have to update the CORS policy in s3. We will do it later.

API Gateway > APIs > Resources - vxs220071-gateway (i2ig7bq8sb)

Resources

Resource details		Delete	Update documentation	Enable CORS
Path /submit	Resource ID hlkin5			
Methods (2)				
Method type	Integration type	Authorization	API key	
OPTIONS	Mock	None	Not required	
POST	Lambda	None	Not required	

API Gateway > APIs > Resources - vxs220071-gateway (i2ig7bq8sb) > Enable CORS

Enable CORS

CORS settings Info

To allow requests from scripts running in the browser, configure cross-origin resource sharing (CORS) for your API.

Gateway responses
API Gateway will configure CORS for the selected gateway responses.

Default 4XX

Default 5XX

Access-Control-Allow-Methods

OPTIONS

POST

Access-Control-Allow-Headers
API Gateway will configure CORS for the selected gateway responses.

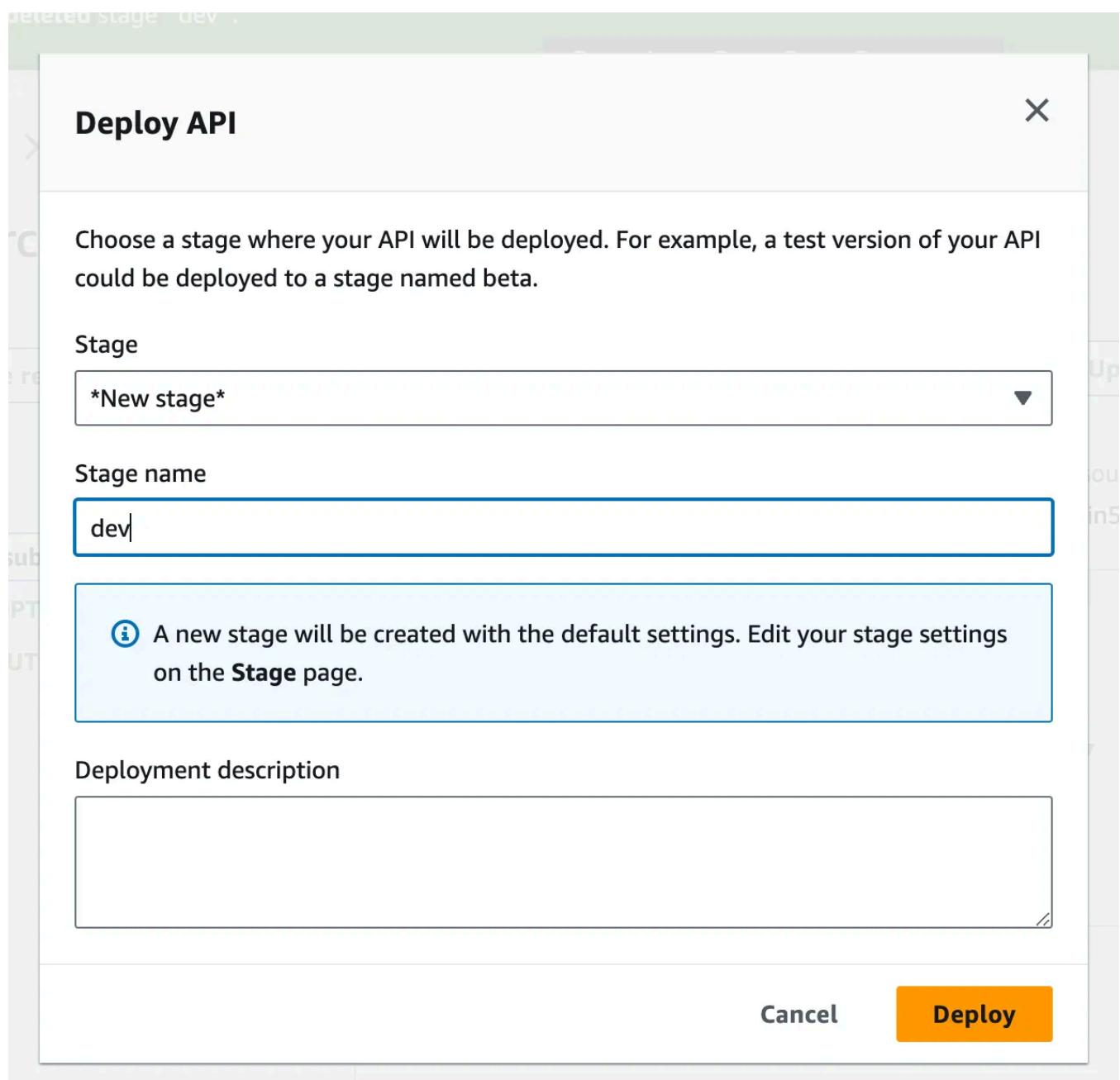
`Content-Type,X-Amz-Date,Authorization,X-Api-Key,X-Amz-Security-Token`

Access-Control-Allow-Origin
Enter an origin that can access the resource. Use a wildcard '*' to allow any origin to access the resource.

`http://vxs220071-webapp.s3-website-us-east-1.amazonaws.com`

► Additional settings

5. Click on Deploy API, select a New stage and give your stage name.



6. Once deployed, we can see the invoke URL below. This URL is responsible for receiving a response when the form is submitted. So, we have to modify the script.js file in the s3 bucket and upload the same file again.

Stages

Stage actions ▾

Create stage

Stage details Info		
Stage name dev	Rate Info -	Web ACL -
Cache cluster Info ⊖ Inactive	Burst Info -	Client certificate -
Default method-level caching ⊖ Inactive		
Invoke URL 🔗 https://i2ig7bq8sb.execute-api.us-east-1.amazonaws.com/dev		
Active deployment e9mq97 on April 20, 2024, 21:38 (UTC-05:00)		

Below is the part where you have to update the file. Once updated, upload the file again.

```
function submitForm(formData) {
    // Make an API request to the backend (API Gateway) for form submission
    fetch('<invoke-url>/submit', { // URL that represents the backend API endpoint
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify(formData)
    })
}
```

```
function submitForm(formData) {
    // Make an API request to the backend (API Gateway) for form submission
    fetch('https://i2ig7bq8sb.execute-api.us-east-1.amazonaws.com/dev/submit', { // URL that represents the backend API endpoint
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        }
    })
}
```

7. We have to update the CORS policy in s3. Open the s3 bucket and scroll down to the CORS section in the permissions tab. Replace the static URL.

Cross-origin resource sharing (CORS)

The CORS configuration, written in JSON, defines a way for client web applications that are loaded in one domain to interact with resources in a different domain. [Learn more](#)

Edit

```
[  
  {  
    "AllowedHeaders": [  
      "*"  
    ],  
    "AllowedMethods": [  
      "POST"  
    ],  
    "AllowedOrigins": [  
      "http://vs220071-webapp.s3-website-us-east-1.amazonaws.com"  
    ],  
    "ExposeHeaders": []  
  }  
]
```

Copy

Copy the CORS policy below.

```
[  
  {  
    "AllowedHeaders": [  
      "*"  
    ],  
    "AllowedMethods": [  
      "POST"  
    ],  
    "AllowedOrigins": [  
      "your-static-website-url"  
    ],  
    "ExposeHeaders": []  
  }  
]
```

Step 4: Testing

1. Let's see whether the submitted form is being saved in DynamoDB. Hit the static website in the browser and fill in the data. Once the data is submitted, we can open the db and explore items to see the data.

Contact Form

Name:

Kakashi Hatake

Email:

kakashifromleaf@gmail.com

Subject:

Naruto Character

Message:

Hello from Kakashi!

Submit

Thank You!

Your message has been submitted successfully.

Items returned (2)

<input type="checkbox"/>	SubmissionId (String)	email	message	name	subject	
<input type="checkbox"/>	8500b35a-e0a6-4c5b-b6d0-c7c0da9469bc	mymail@example.com	A message from lambda	myname	lambda-testing	 
<input type="checkbox"/>	c0b19dbd-a0cb-4c13-943e-f26fefa8aa2c	kakashifromleaf@gmail.c...	Hello from Kakashi	Kakashi Hat...	Naruto Character	 

Congrats on the completion of the project!

Connect with me on Linkedin: <https://www.linkedin.com/in/venkatgiris/>

This blog is based on Ulises's blog. Everything is pretty similar except for a few steps where I struggled.

Thank you [Ulises Magana](#).

DevOps

AWS

Cloud Computing

Serverless

Cloud



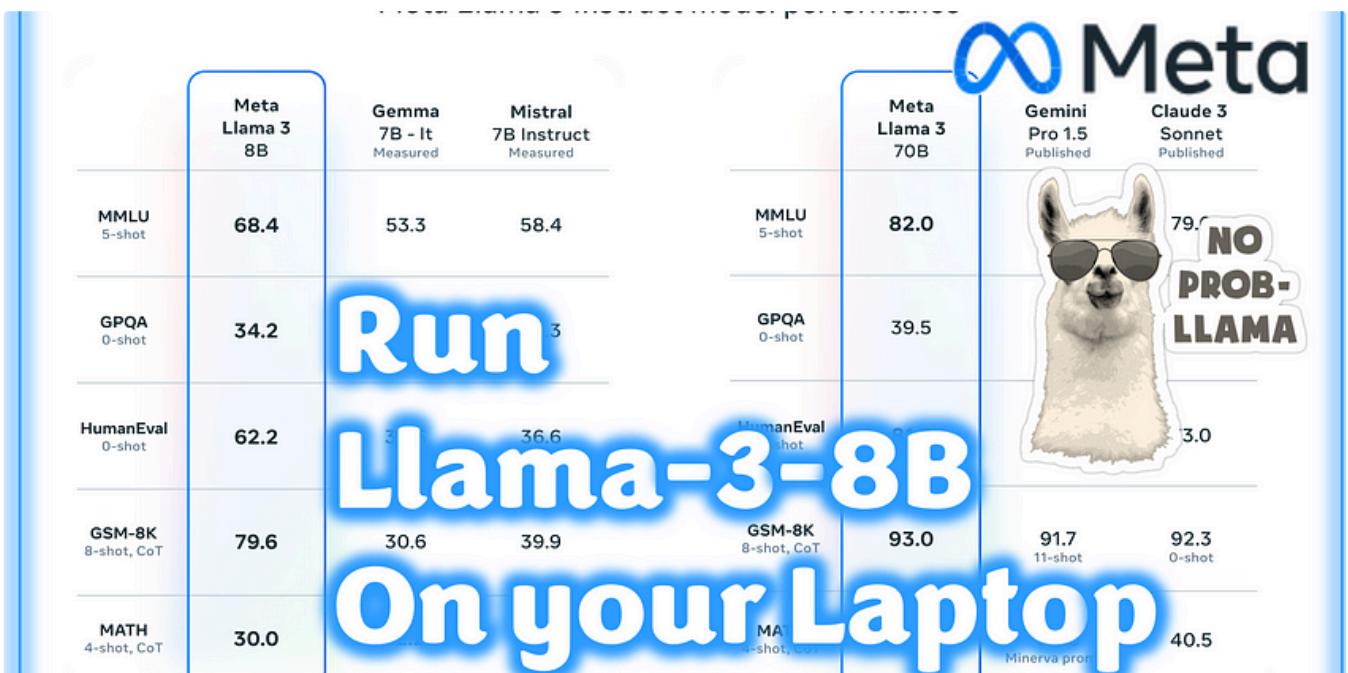
Follow



Written by Venkatgiri Sasanapuri

4 Followers

Recommended from Medium





Fabio Matricardi in Generative AI

Llama3 is out and you can run it on your Computer!

After only 1 day from the release, here is how you can run even on your Laptop with CPU only the latest Meta-AI model.

★ · 8 min read · Apr 20, 2024

👏 1.5K ⚡ 16



...

	Period	Note
	1 week	Simple landing page connected to domain of the venture. Collect emails
	2 weeks	Send emails, get feedback, and try the real need
	3 weeks	Start the building stage, with 1 single focus
	4 weeks	Launch in public



Domenico Gagliardi

How I built and sold a micro-saas in 45 days

Hello everyone,

★ · 3 min read · Apr 20, 2024

👏 271 ⚡ 9



...

Lists



General Coding Knowledge

20 stories · 1154 saves



Leadership

50 stories · 305 saves



Natural Language Processing

1409 stories · 906 saves



Productivity

240 stories · 408 saves

The dashboard displays several components:

- Request Count Over Time:** A line chart showing the number of requests over time for various API endpoints.
- Response Code Over Time:** A line chart showing the distribution of response codes over time.
- Traces:** A screenshot of a distributed tracing interface showing call graphs and latency metrics.
- Metrics:** A screenshot of a metrics monitoring interface showing time series data.
- Aliveness:** A screenshot of a Postman collection showing a GET request to `localhost:9000/well-known/alive` with a JSON response body containing `{"data": {"status": "UP"}}`.
- HTTP Response:** A screenshot of a Postman collection showing a GET request to `localhost:9000/hello` with a JSON response body containing `{"data": "Hello world!"}`.

GoFr.dev



Aryan Mehrotra in Stackademic

Why use GoFr for Golang Backend?

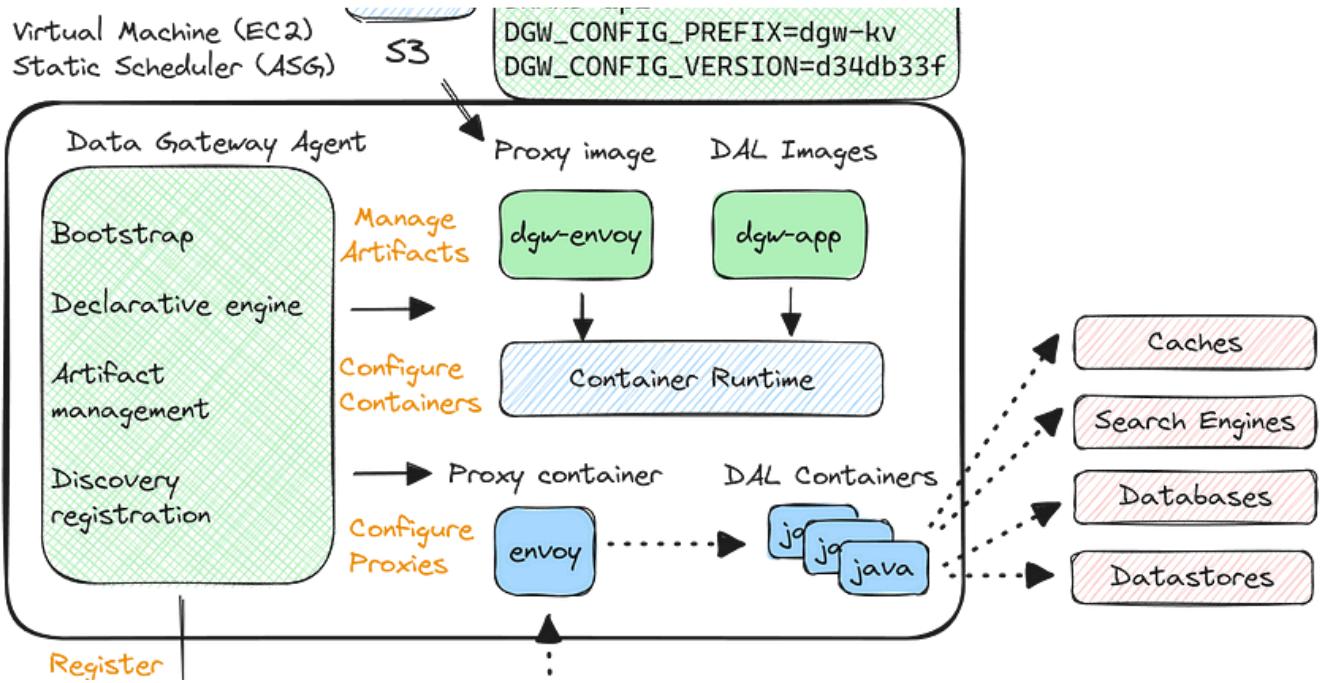
5 min read · Apr 21, 2024

👏 344

💬 1



...



 Netflix Technology Blog

Data Gateway—A Platform for Growing and Protecting the Data Tier

Shahar Zimmerman, Vidhya Arvind, Joey Lynch, Vinay Chella

13 min read · 5 days ago

 312  3



...



 Economics & Tech in Summit

The Secrets Behind Temu's Unbeatable Prices

How a Chinese e-commerce giant is using billions of dollars to disrupt the U.S. market

◆ · 7 min read · Apr 21, 2024

👏 1.3K

💬 31



...



 Alireza Yavari

The Observability Stack Part1

In the DevOps world when we talk about observability, we mean to generate, collect, and gather, perform some manipulation on it, centralize...

9 min read · Apr 20, 2024

👏 345

💬 1



...

See more recommendations