
Serverless Reference Architecture: Mobile Backend

README Languages: [DE](#) | [ES](#) | [FR](#) | [IT](#) | [JP](#) | [KR](#) | [PT](#) | [RU](#) | [CN](#) | [TW](#) ## Introduction

The Mobile Backend reference architecture (diagram) demonstrates how to use AWS Lambda along with other services to build a serverless backend for a mobile application. The specific example application provided in this repository enables users to upload photos and notes using Amazon Simple Storage Service (Amazon S3) and Amazon API Gateway respectively. The notes are stored in Amazon DynamoDB, and are processed asynchronously using DynamoDB streams and a Lambda function to add them to an Amazon CloudSearch domain. In addition to the source code for the Lambda functions, this repository also contains a prototype iOS application that provides examples for how to use the AWS Mobile SDK for iOS to interface with the backend resources defined in the architecture.

Running the example

To run the full example application, you must first deploy the backend resources, and then compile and run the example iOS application.

Deploying the Backend

The provided AWS CloudFormation template creates most of the backend resources that you need for this example, but you still need to create the Amazon CloudSearch domain, API Gateway REST API, and Cognito identity pool outside of AWS CloudFormation.

Step 1: Create a CloudSearch Domain

1. Using the AWS CLI, create a new CloudSearch domain providing a domain name of your choice.

```
1 aws cloudsearch create-domain --domain-name [YOUR_DOMAIN_NAME]
```

2. Note the ARN of the new domain in the output document. You will use this as an input when launching the CloudFormation stack.
3. Define indexes for the `headline` and `note_text` fields.

```
1 aws cloudsearch define-index-field --name headline --type text --  
  domain-name [YOUR_DOMAIN_NAME]  
2 aws cloudsearch define-index-field --name note_text --type text --  
  domain-name [YOUR_DOMAIN_NAME]
```

Note: You will need to reindex your Domain via CLI or Console. Here is the CLI command:

```
1 aws cloudsearch index-documents --domain-name [YOUR_DOMAIN_NAME]
```

Step 2: Create an API Gateway REST API

1. Using the AWS CLI, create a new API providing a name of your choice.

```
1 aws apigateway create-rest-api --name [YOUR_API_NAME]
```

2. Note the **API ID** provided in the output document. You will use this as an input when launching the CloudFormation stack.

Step 3: Create an Amazon Cognito Identity Pool

1. Using the AWS CLI, create a new identity pool providing a name of your choice.

```
1 aws cognito-identity create-identity-pool --allow-unauthenticated-identities --identity-pool-name [YOUR_POOL_NAME]
```

2. Note the **IdentityPoolId** from the output document. You will use this as a parameter when launching the CloudFormation stack.

Step 4: Launch the CloudFormation Template You can deploy the entire example in the us-east-1 region using the provided CloudFormation template and S3 bucket by Launching the stack below. If you would like to deploy the template to a different region, you must create an Amazon S3 bucket in that region, then in the file `lambda-functions/upload`, modify the `BUCKET` variable with the bucket you created, run `./upload`, and copy `config-helper.template` and `mobile-backend.template` into the bucket.

Choose **Launch Stack** to launch the template in the us-east-1 region in your account:



When prompted, enter the parameter values for the CloudSearch domain, CloudSearch Search/Document Endpoints, API Gateway REST API ID, and Amazon Cognito identity pool resources you created in the previous steps.

Details about the resources created by this template are provided in the *CloudFormation Template Resources* section of this document.

Step 5: Update Your API Gateway REST API After you have created the CloudFormation stack, you need to update the API you created previously to use the newly created [NotesApiFunction](#).

1. In the Amazon API Gateway Console, choose your API.
2. Choose **Create Resource** to create a new child resource under /.
3. Type [notes](#) as the resource name and [/notes](#) as the resource path.
4. Choose **Create Resource**.
5. With the new [/notes](#) resource selected, choose **Create Method**.
6. Choose [POST](#) and select the check box.
7. Choose **Lambda Function** as the integration type and then choose the region where you launched the CloudFormation stack as the Lambda region.
8. in **Lambda Function**, type [NotesApiFunction](#) and then select the function created by the CloudFormation Stack.
9. Choose **Save** and grant API Gateway permissions to execute the Lambda function.
10. Choose **Method Request** to edit the request configuration.
11. For **Authorization type**, select [AWS_IAM](#).
12. For **API Key Required**, select [true](#).
13. Select the Actions dropdown, Choose **Deploy API**.
14. For **Deployment stage**, choose [New Stage](#) and then type a name in **Stage name**.
15. Note the **Invoke URL** for the new stage. You will use this value when running the sample iOS app.

Step 6: Create an API Key

1. In the Amazon API Gateway Console, choose **API Keys**.
2. Choose **Create API Key**.
3. Type a name for the key and then select **Save**.
4. Note the **API key**. You will use this when running the mobile application.
5. Click on **Usage Plans** and type a name for the plan. Disable Throttling and Quotas. Click **Next**.
6. From the dropdowns, select your API and Stage created in Step 4. Click the Checkbox and then **Next**.
7. Click **Add API Key to Usage Plan** and enter the Key you created earlier then choose **Done**.

Step 7: Update Your Amazon Cognito Identity Pool

1. In the Amazon Cognito Console, select your identity pool.
2. Choose **Edit Identity Pool**.
3. For both the **Unauthenticated role** and the **Authenticated role**, select the **MobileClientRole** created by the CloudFormation stack. The full ARN for the role is provided in the outputs of the

-
- stack.
4. Choose **Save Changes**.

Running the Sample iOS Application

Prerequisites To run the provided iOS sample application, you must be running Mac OS X 10.10 (Yosemite) or a more recent version. You must also have the latest version of Xcode and Cocoa Pods installed.

Building and Running the Application

1. Check out or download the source code for the **ios-sample** in this repository.
2. Update `MobileBackendIOS/Constants.swift` with the values for your backend deployment. Most of the values can be found in the outputs of the CloudFormation stack. The API Gateway key and endpoint URL values are available in the details for your API in the AWS Management Console.
3. Run Cocoa Pods from the root `ios-sample` directory.

```
1 pod install
```

4. Open the generated `MobileBackendIOS.xcworkspace` file in Xcode.

```
1 open -a Xcode MobileBackendIOS.xcworkspace
```

5. Build and run the project from Xcode by clicking the play button at the top of the window.

Testing the Application

The sample application provides two features: uploading an image and posting a note.

To upload an image

1. Choose **Upload Image** in the application.
2. Choose the camera icon, select an image from the camera roll, and then choose **Choose**.
3. Choose the **Upload** button.

Validating that the image was uploaded You should see a log entry that the image has been uploaded to Amazon S3 in the output pane of Xcode.

You can also browse the bucket created by the CloudFormation stack using the AWS Management Console to verify that the image was correctly uploaded.

To post a note

1. Choose **Post a Note**.
2. In the note, type a headline and text.
3. Choose **Save Note**.

Validating that the note was posted You should see a log entry that the note was saved successfully in the output pane of Xcode.

When the note is uploaded, the `NotesApiFunction` is invoked by the mobile application. You can view the logs for this function in Amazon CloudWatch.

When the function is successfully invoked, it adds an entry to the DynamoDB table created in the CloudFormation stack. You can verify that the note you posted in the application has been persisted in the created table.

Finally, when the note is persisted in the DynamoDB table a record is added to the table's stream which is in turn processed by the `DynamoStreamHandlerFunction`. You can view the logs for this function in CloudWatch and verify that a new document has been added to the CloudSearch domain you created.

Cleaning Up the Application Resources

To remove all resources created by this example, do the following:

1. Delete all objects from the S3 bucket created by the CloudFormation stack.
2. Delete the CloudFormation stack.
3. Delete the Amazon Cognito identity pool, API Gateway, and CloudSearch domain.
4. Delete the CloudWatch log groups associated with each Lambda function created by the CloudFormation stack.

CloudFormation Template Resources

Lambda Functions

- **NotesApiFunction** - A function to handle posted notes from the mobile application via API Gateway.
- **SearchApiFunction** - A function that uses the CloudSearch domain to find indexed notes based on search terms.
- **DynamoStreamHandlerFunction** - A function that adds an indexed document to the provided CloudSearch domain based on records in the [PhotoNotesTable](#) stream.

AWS Identity and Access Management (IAM) Roles

- **NotesApiRole** - A role for the [NotesApiFunction](#). This role grants permission for logging and working with items in the [PhotoNotesTable](#).
- **SearchApiRole** - A role for the [SearchApiFunction](#). This role grants permissions for logging and searching the provided CloudSearch domain.
- **DynamoStreamHandlerRole** - A role for the [DynamoStreamHandlerFunction](#). This role grants permissions for logging and adding documents to the provided CloudSearch domain.
- **MobileClientRole** - A role used by your Amazon Cognito identity pool for both unauthenticated and authenticated users. This role provides access to the provided API Gateway REST API as well as permissions for putting objects to the [MobileUploadsBucket](#).

Other Resources

- **MobileUploadsBucket** - An S3 bucket for user uploaded photos.
- **CloudFrontDistribution** - A CDN distribution with the [MobileUploadsBucket](#) configured as an origin.
- **PhotoNotesTable** - A DynamoDB table that stores notes uploaded by users from the mobile application.

Configuration

- **ConfigTable** - A DynamoDB table to hold configuration values read by the various Lambda functions. The name of this table, “MobileRefArchConfig”, is hard coded into each function’s code and cannot be modified without updating the code as well.

-
- **ConfigHelperStack** - A sub-stack that creates a custom resource for writing entries to the [ConfigTable](#). This stack creates a Lambda function and execution role that grants UpdateItem permission on the [ConfigTable](#).
 - **NotesTableConfig** - A configuration entry that identifies the [PhotoNotesTable](#) name.
 - **SearchEndpointConfig** - A configuration entry that identifies the search endpoint of the CloudSearch domain passed as a parameter.
 - **DocumentEndpointConfig** - A configuration entry that identifies document endpoint of the CloudSearch domain passed as a parameter.

License

This reference architecture sample is licensed under Apache 2.0.