



[日付]

# はじめてのサーバーレス

[文書のサブタイトル]

KAMEDA, HARUNOBU

AMAZON CORPORATE

# 目次

目次.....	1
1. はじめに.....	2
1.1. 本ハンズオンのゴール.....	2
1.2. 準備事項.....	2
2. ハンズオンの概要.....	3
2.1. ハンズオン全体を通しての注意事項.....	3
2.2. ハンズオンの構成.....	3
3. 準備.....	5
3.1. Cloud9 環境の作成.....	5
3.1.1. AWS マネジメントコンソールへのログイン .....	5
3.1.2. Cloud9 環境の作成 .....	5
4. サーバーレスアプリケーションを <i>step-by-step</i> で構築する.....	10
4.1. DynamoDB および Lambda を使用して動作確認を行う.....	10
4.1.1. DynamoDB テーブルの作成 .....	10
4.1.2. Lambda 用 IAM ポリシーの作成.....	12
4.1.3. Lambda 用 IAM ロールの作成.....	16
4.1.4. Cloud9 上での Lambda 関数の作成とテスト(Write 関数) .....	19
4.1.5. Cloud9 上での Lambda 関数の作成とテスト(Read 関数) .....	27
4.1.6. Lambda 関数のデプロイ .....	34
4.2. API Gateway を追加して動作確認を行う .....	36
4.2.1. API Gateway REST API 作成.....	36
4.2.2. API に GET メソッドを追加.....	40
4.2.3. API に POST メソッドを追加.....	50
4.2.4. CORS の設定.....	58
4.2.5. API のデプロイ .....	63
4.2.6. Cloud9 からの動作確認 .....	66
4.2.7. Web ブラウザからの動作確認 .....	69
4.3. Cognito の動作確認を行う .....	73
4.3.1. Cognito ユーザープールの作成 .....	73
4.3.2. Cognito ID プールの作成 .....	79
4.3.3. Web ブラウザからの動作確認 .....	83
4.4. Cognito による認証と API Gateway を組み合わせる.....	91
4.4.1. API Gateway に認証の設定を追加 .....	91
4.4.2. IAM ロールにポリシーを追加 .....	99
4.4.3. Web ブラウザからの動作確認 .....	111
5. Amplify を使ってサーバーレスアプリケーションを構築する.....	115
6. 後片付け.....	エラー! ブックマークが定義されていません。

# 1. はじめに

## 1.1. 本ハンズオンのゴール

AWS では「サーバーレスアプリケーション」を構築するためのさまざまなサービスが提供されています。例えば以下のようなものがあります。

- コンピューティング : AWS Lambda
- データストア : Amazon DynamoDB
- API の発行と管理 : Amazon API Gateway
- ユーザー認証・認可 : Amazon Cognito

サーバーレスに初めて触れる方にとっては、これらのサービスをどのように組み合わせることによってサーバーレスアプリケーションを構築できるのか、それぞれのサービスをどのように設定して利用すればよいのか、分からぬ点が多いのではないかと思います。

本ハンズオンでは、これらのサービスを一つずつ構築していくことで、step-by-step でサーバーレスアプリケーションの構築方法について理解して頂くことをゴールとしています。

## 1.2. 準備事項

- AWS を利用可能なネットワークに接続された PC (Windows, Mac OS, Linux 等)
- 事前に用意していただいた AWS アカウント
- ブラウザ (Firefox もしくは Chrome を推奨)

## 2. ハンズオンの概要

### 2.1. ハンズオン全体を通しての注意事項

本ハンズオンは、基本的に「東京」、「バージニア北部」、「オレゴン」、「シンガポール」を前提に記載されています。リソースなどの上限に引っかかってしまった場合は、上記のリージョンのどれかの環境で作成することが可能です。作業は特に指定されない限りは東京リージョンを使ってください。

各章で配置されている「補足説明」につきましては、本ハンズオンを進めていただく上では必須手順ではありません。参考資料としてください。

同じ AWS アカウントで複数人が同時に本ハンズオンを実施される場合、適宜名前などが重複しないようにご留意ください。

各手順において、「任意」と記載のあるものについては自由に名前を変更いただくことができますが、ハンズオン中に指定した名前がわからなくなないように、ハンズオン実施中は基本的にはそのままの名前で進めることを推奨いたします。

### 2.2. ハンズオンの構成

本ハンズオンは 2 つのラボで構成されています。

- 準備
  - AWS サービス : AWS Cloud9
- サーバーレスアプリケーションを step-by-step で構築する
  - AWS サービス : Amazon DynamoDB、AWS Lambda、Amazon API Gateway、Amazon Cognito
- Amplify を使ってサーバーレスアプリケーションを構築する
  - AWS サービス : AWS Amplify、Amazon DynamoDB、AWS Lambda、Amazon API Gateway、Amazon Cognito

これらのハンズオンを通じて、サーバーレスアプリケーションを構成する代表的なサービスについての理解と、それらを組み合わせて実際にサーバーレスアプリケーションの動作を確認することができます。

### 3. 準備

本日のハンズオンで使用する **Cloud9** の環境を構築します。

Cloud9 は、コードの記述・実行・デバッグが行えるクラウドベースの統合開発環境（IDE）です。

このハンズオンでは、EC2 インスタンスを利用して新規に Cloud9 の環境を構築します。

#### 3.1. Cloud9 環境の作成

##### 3.1.1. AWS マネジメントコンソールへのログイン

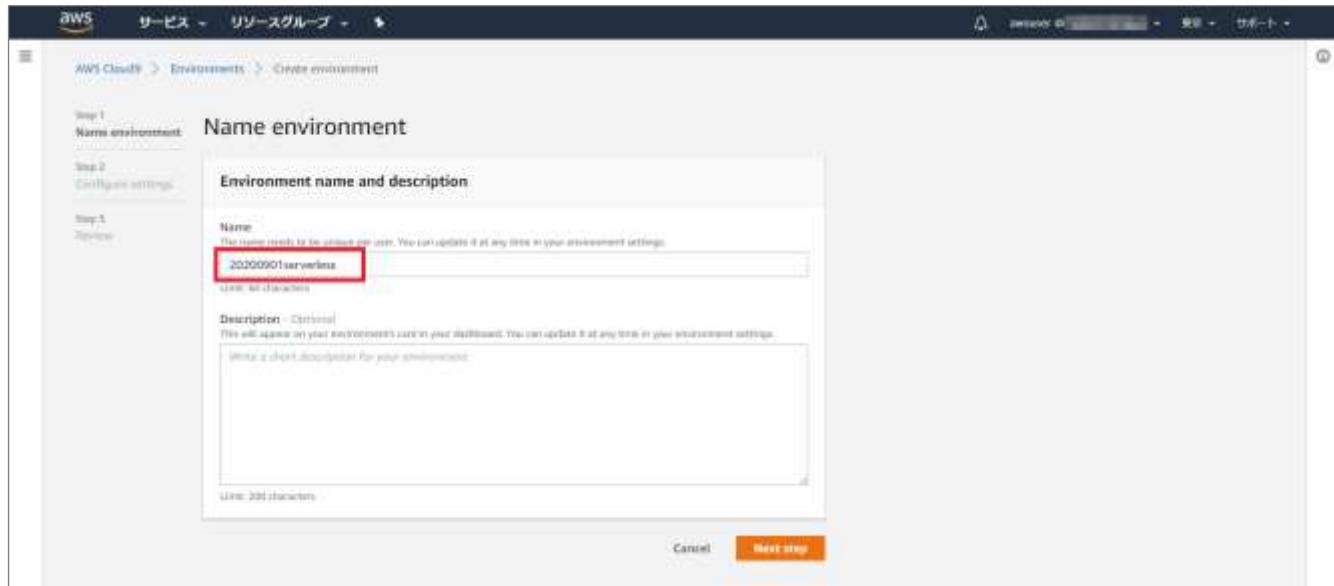
1. AWS マネジメントコンソールにログインします。
2. ログイン後、画面右上のヘッダー部のリージョン選択にて、**利用を指示されたリージョン** となっていることを確認します。  
注：[東京]、[バージニア]、[オレゴン]、[シンガポール] のどれかになります。

##### 3.1.2. Cloud9 環境の作成

1. AWS マネジメントコンソールのサービス一覧から **[Cloud9]** を選択します。
2. **[Create environment]** をクリックします。



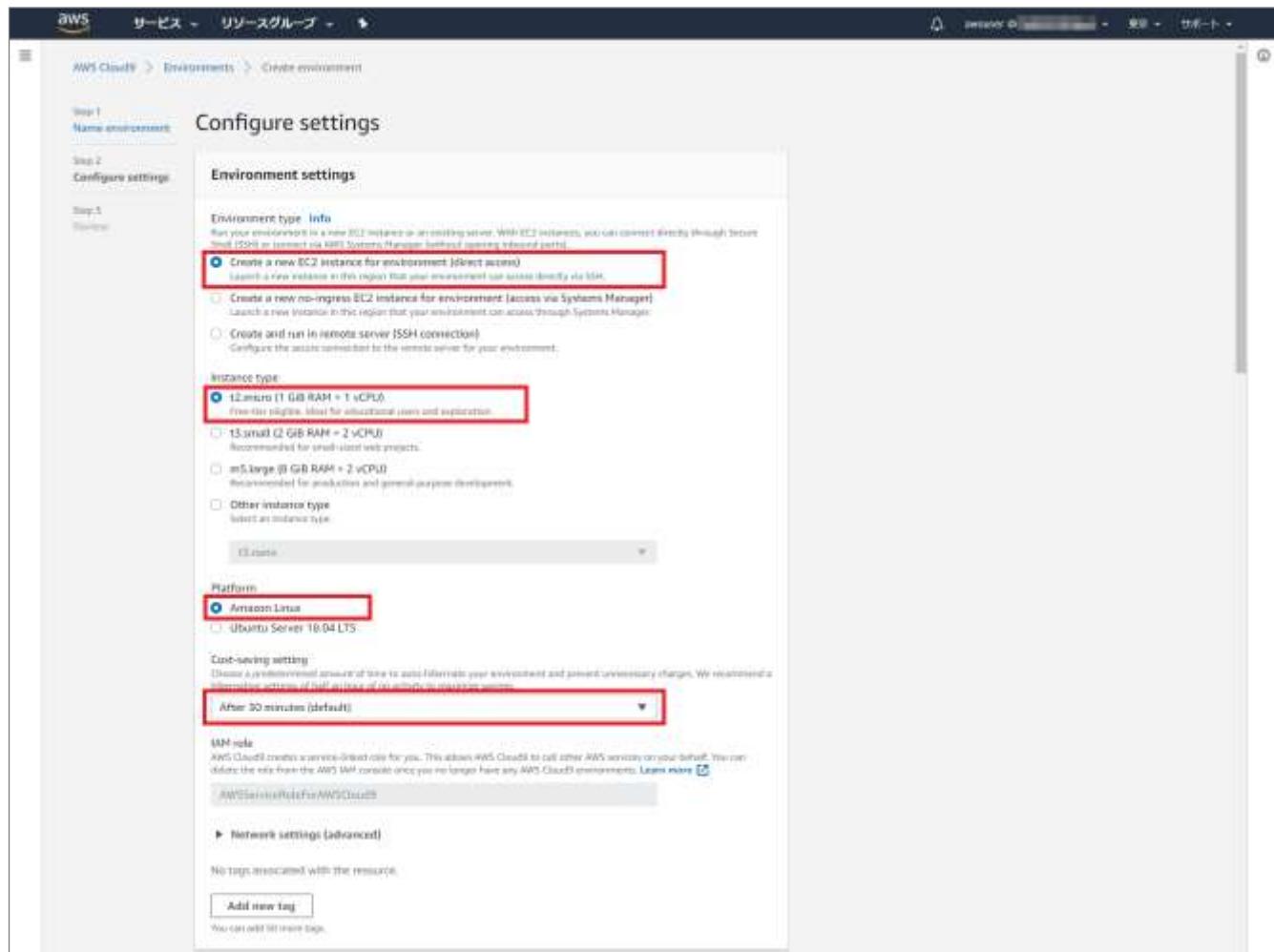
3. [Name] 欄に [YYYYMMDDserverless] と入力します。 (YYYYMMDD は本日の日付)



4. [Next Step] をクリックします。

5. 以下のように設定します。 (基本的にデフォルトのままで構いません)

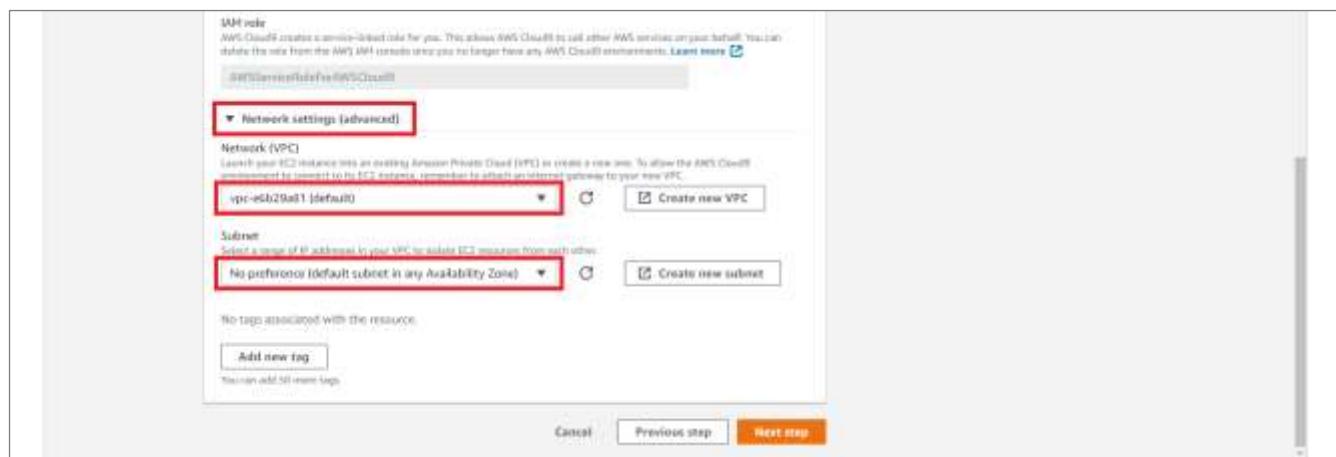
- **Environment type:** [Create a new EC2 instance for environment (direct access)]
- **Instance type:** [t3.small]
- **Platform:** [Amazon Linux]
- **Cost-saving setting:** [After 30 minutes] (アイドル状態が 30 分続くと自動的に EC2 インスタンスを停止する設定です)
- **IAM Role:** [AWSServiceRoleForAWSCloud9] (変更できません)



6. [Network settings (advanced)] をクリックして展開します。

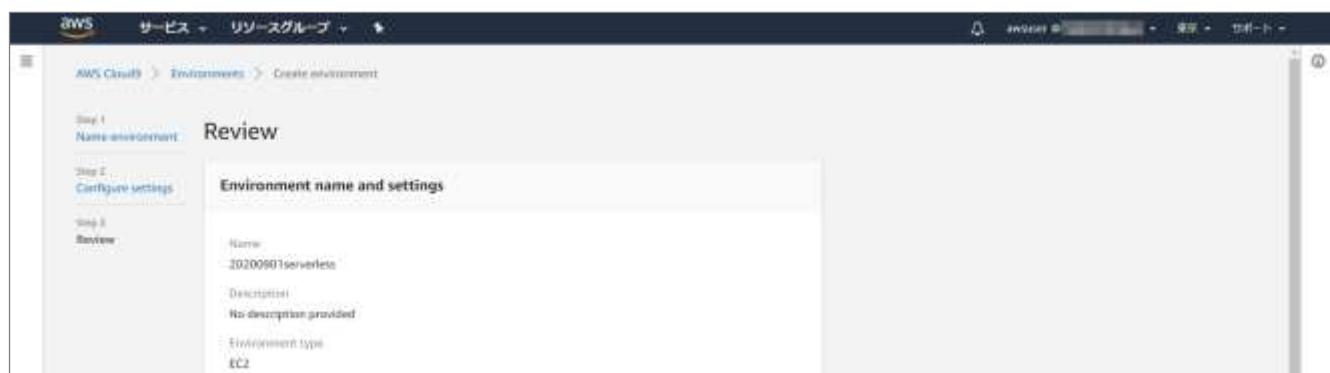
Cloud9 環境を構築する VPC および サブネット を指定できますので、任意の VPC を指定してください。サブネットは「パブリックサブネット」であるものを指定してください。

不明な場合はデフォルト VPC（名前に「(default)」が付いた VPC）を選択すれば問題ありません。

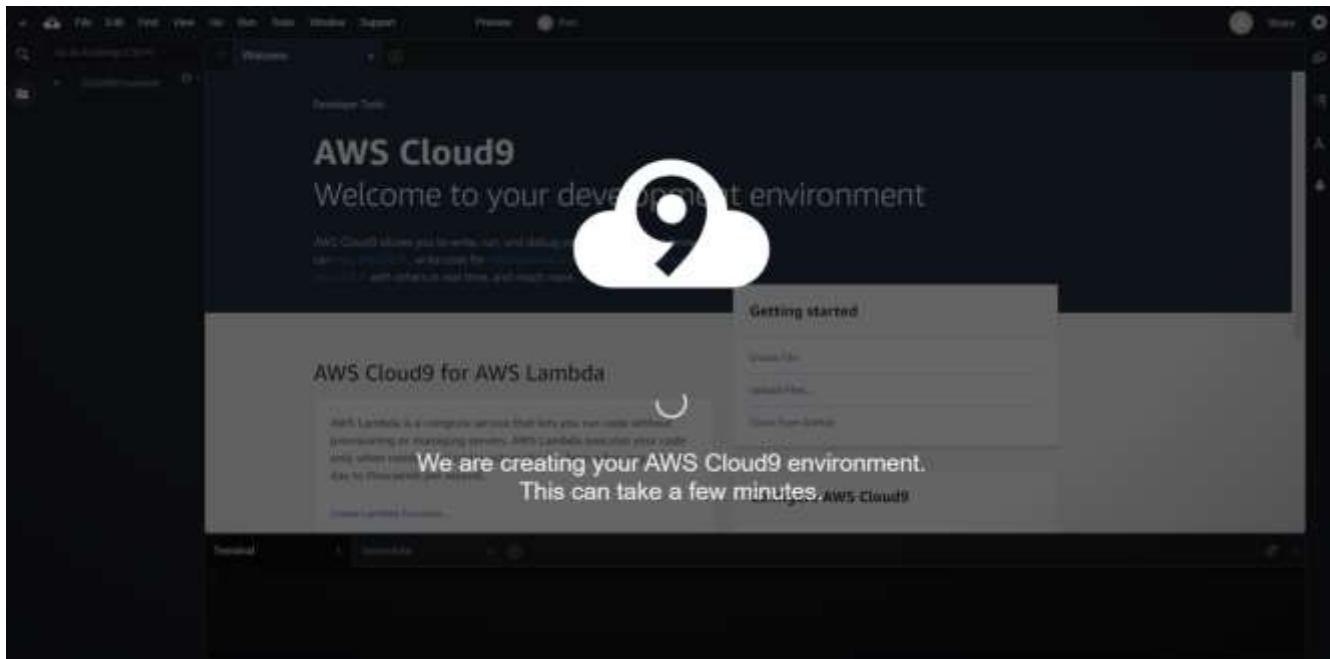


7. [Next Step] をクリックします。

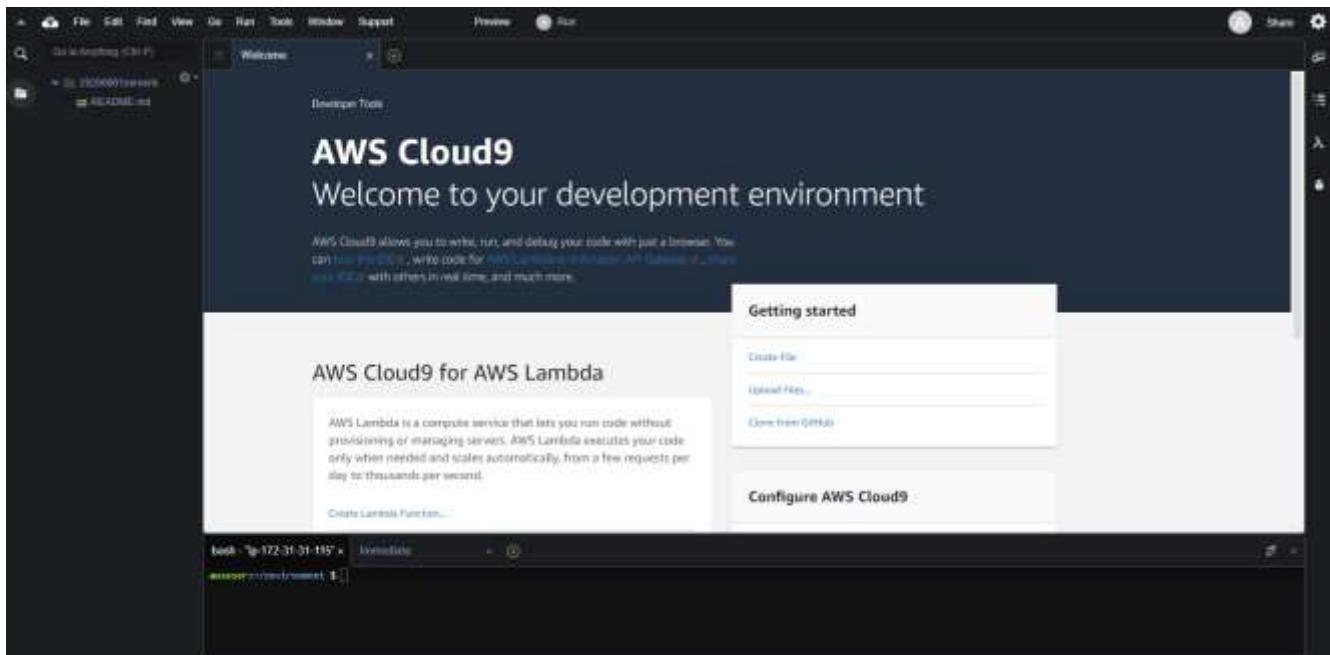
8. 確認画面になりますので、[Create environment] をクリックします。



9. Cloud9 環境の作成が始まります。完成まで数分間かかります。



10. Cloud9 環境が作成されました。



## 4. サーバーレスアプリケーションを step-by-step で構築する

### 4.1. DynamoDB および Lambda を使用して動作確認を行う

#### 4.1.1. DynamoDB テーブルの作成

1. AWS マネジメントコンソールのサービス一覧から **[DynamoDB]** を選択します。

**[テーブルの作成]** をクリックします。



2. 以下の通り入力します。

- テーブル名: [YYYYMMDDserverless] (YYYYMMDD は本日の日付)
- プライマリーキー: [Artist] と入力、右側のプルダウンから [文字列] を選択
- [ソートキーの追加] にチェックを入れる
- ソートキー: [Title] と入力、右側のプルダウンから [文字列] を選択



3. [作成] をクリックします。

4. テーブルが作成されるまで 1~2 分程度かかります。

5. テーブルが作成されたら、[項目] タブをクリックします。

下図のように [Artist]、[Title] の各属性が構成されていることを確認します。



## 4.1.2. Lambda 用 IAM ポリシーの作成

1. AWS マネジメントコンソールのサービス一覧から [IAM] を選択します。

画面左側のメニューから [ポリシー] を選択します。

The screenshot shows the AWS IAM Management Console. On the left sidebar, under the 'Identity and Access Management (IAM)' section, the 'Policies' option is highlighted with a red box. The main pane displays the 'Identity and Access Management へようこそ' (Welcome to Identity and Access Management) page. It includes sections for IAM ユーザーのサインインリンク, IAM リソース (User: 2, Group: 0, Role: 0, IAM プロバイダー: 0), and セキュリティステータス (5 項目中 5 項目が完了しています). A sidebar on the right lists additional resources like IAM ヘストプラグイン, IAM ドキュメント, and Policy Simulator.

2. [ポリシーの作成] をクリックします。

The screenshot shows the 'Create New Policy' screen in the AWS IAM Management Console. The left sidebar shows the same navigation as before. The main area has a blue box around the 'ポリシーの作成' (Create New Policy) button. Below it, a table lists various pre-defined policies with columns for Name, Type, and Description. The table header includes 'ポリシー名', 'タイプ', '既として使用', and '説明'.

ポリシー名	タイプ	既として使用	説明
AccessAnalyzerServiceRolePolicy	AWS による管理	なし	Allow Access Analyzer to analyze resource metadata
AdministratorAccess	権限範囲	Permissive policy (2)	Provides full access to AWS services and resources.
AlexaForBusinessDeviceSetup	AWS による管理	なし	Provide device setup access to AlexaForBusiness services
AlexaForBusinessFullAccess	AWS による管理	なし	Grants full access to AlexaForBusiness resources and access to related AWS...
AlexaForBusinessGatewayExecution	AWS による管理	なし	Provide gateway execution access to AlexaForBusiness services
AlexaForBusinessLifecycleDelegated	AWS による管理	なし	Provide access to LifeCycle AWS devices

### 3. [サービス] をクリックして展開します。



The screenshot shows the AWS IAM Policy Editor interface. At the top, there's a navigation bar with 'AWS' and other options like 'サービス', 'リソースグループ', etc. Below the navigation is a title 'ポリシーの作成'. A horizontal tab bar has 'ビジュアルエディタ' (selected) and 'JSON'. On the right, there's a link '管理ポリシーのインポート'. The main area has a search bar 'すべて検索 | すべて折りたたむ' and a dropdown menu '▼ サービスの選択' which is currently expanded, showing 'サービス サービスの選択' as the selected item. Below this are sections for 'アクション', 'リソース', and 'リクエスト条件'. At the bottom right, there's a note 'さらにアクセス許可を追加する'.

### 4. 検索欄に [dynamodb] と入力します。

表示された [DynamoDB] をクリックします。



This screenshot shows the same AWS IAM Policy Editor interface as the previous one, but with the search term 'dynamodb' entered into the search bar. The search results are displayed below the search bar, showing 'dynamodb' as the selected service and other services like 'DynamoDB' and 'DynamoDB Accelerator'. The rest of the interface remains the same, with the 'Services' dropdown still expanded.

5. [アクション] を展開して、[アクセスレベル] から [読み込み] と [書き込み] にチェックを入れます。



6. [リソース] を展開して、[table] 列の右端にある [このアカウント内のいずれか] にチェックを入れます。

▼ リソース  指定  すべてのリソース

backup	backup リソース ARN を指定します。 <a href="#">DescribeBackup</a> および、さらに 1 つのアクション。 <a href="#">ARN の追加</a> アクセスを制限	<input type="checkbox"/> このアカウント内のいずれか
global-table	global-table リソース ARN を指定します。 <a href="#">UpdateGlobalTable</a> および、さらに 1 つのアクション。 <a href="#">ARN の追加</a> アクセスを制限	<input type="checkbox"/> このアカウント内のいずれか
index	タイプが index であるリソースを指定していません <a href="#">ARN の追加</a> アクセスを制限	<input type="checkbox"/> このアカウント内のいずれか
stream	stream リソース ARN を指定します。 <a href="#">GetRecords</a> および、さらに 1 つのアクション。 <a href="#">ARN の追加</a> アクセスを制限	<input type="checkbox"/> このアカウント内のいずれか
table	arn:aws:dynamodb:*:294963776963:table/*	<input checked="" type="checkbox"/> このアカウント内のいずれか

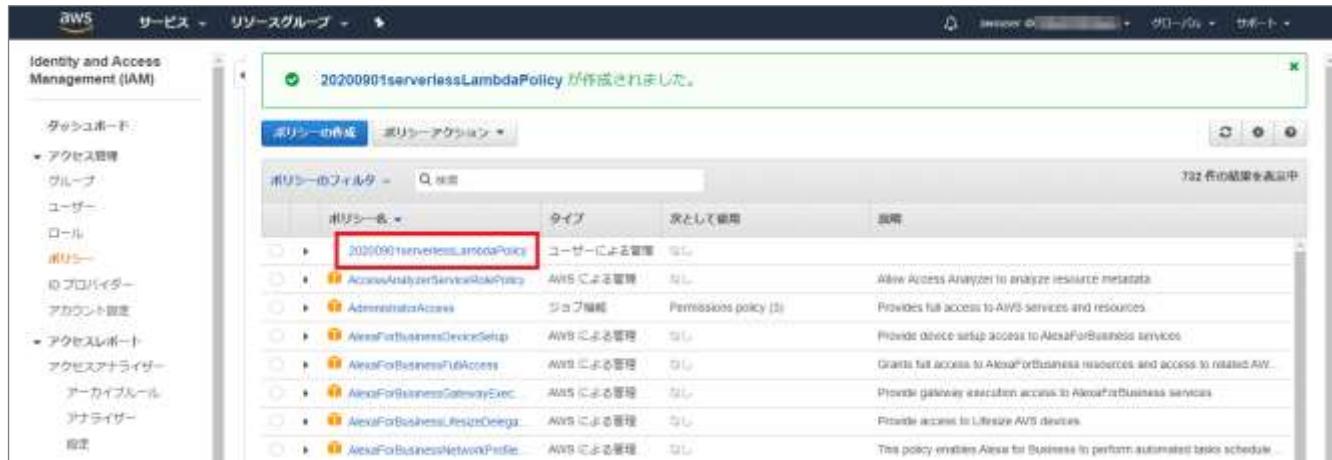
7. [ポリシーの確認] をクリックします。

8. [名前] 欄に [YYYYMMDDserverlessLambdaPolicy] と入力します。 (YYYYMMDD は本日の日付)



9. [ポリシーの作成] をクリックします。

10. ポリシー [YYYYMMDDserverlessLambdaPolicy] が作成されたことを確認します。



### 4.1.3. Lambda 用 IAM ロールの作成

1. [IAM] 画面で、画面左側のメニューから [ロール] を選択します。



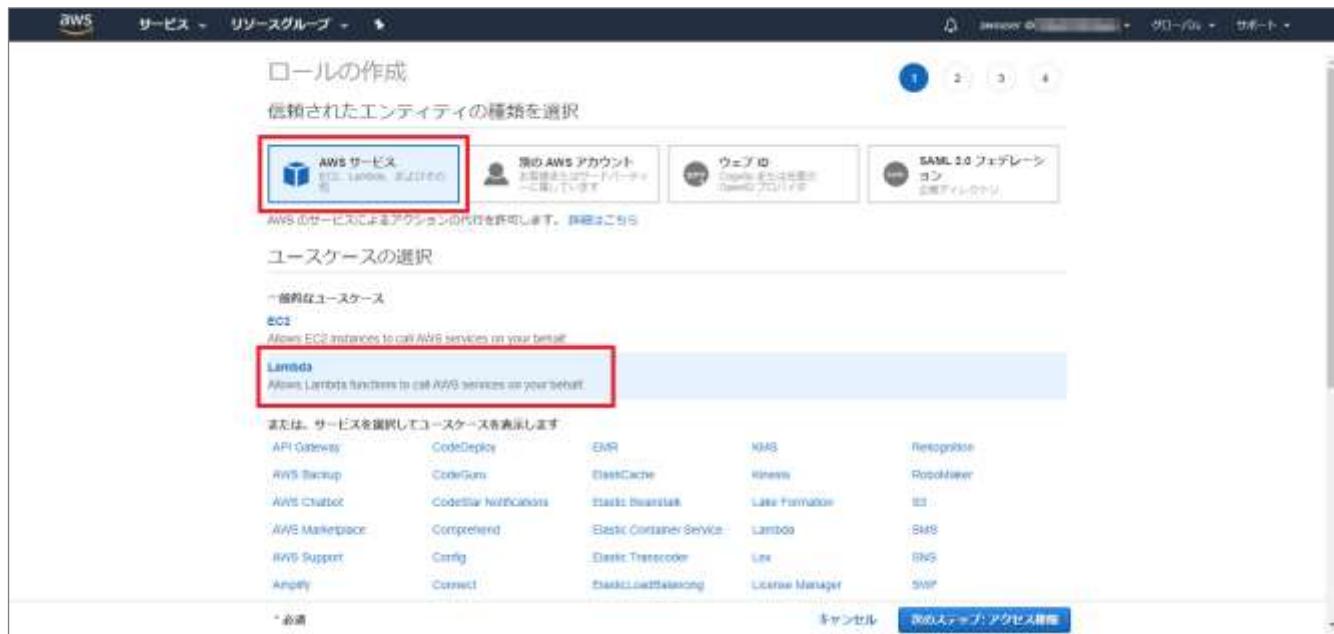
The screenshot shows the AWS IAM console interface. On the left, there is a navigation sidebar with several options: 'Identity and Access Management (IAM)', 'ダッシュボード', 'アクセス管理', 'グループ', 'ユーザー', 'ロール' (which is highlighted with a red box), 'ポリシー', 'ID プロバイダー', 'アカウント設定', 'アクセスレポート', 'アクセスアナライザー', 'アーカイブルール', 'アナライザー', '設定', and '認証履歴レポート'. The main content area is titled 'Identity and Access Management へようこそ' and contains sections for 'AWS ユーザーのサインインリンク', 'IAM リソース' (listing 'ユーザー: 2', 'グループ: 0', and 'カスタマーベンダー: 0'), and 'セキュリティステータス' (with several status items like 'ルートアカウントの MFA を有効化' and '個々の IAM ユーザーの作成'). On the right side, there is a '追加情報' panel with links to 'IAM ヘストプラグイン', 'IAM ドキュメント', 'AWS ID フェデレーションのフレイグラウンド', 'Policy Simulator', and '動画: IAM リソース 説明: よび追加のリソース'.

2. [ロールの作成] をクリックします。



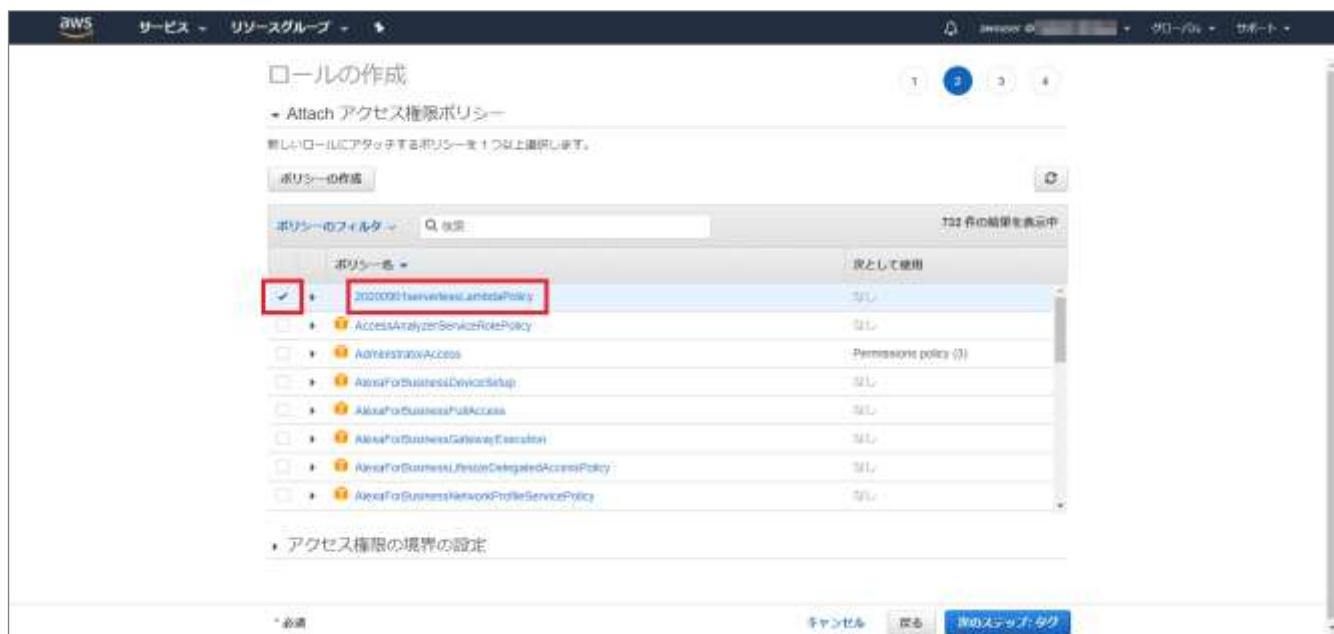
The screenshot shows the 'Roles' page in the AWS IAM console. The left sidebar is identical to the previous screenshot. The main area has two tabs: 'ロールの概要' (selected) and 'ロールの検索'. Below these tabs is a search bar labeled 'Q 検索'. A table lists several existing roles: 'AmazonDSSParsePutDataAccessRole' (AWS サービス: dss-parse-pots, 最後のアクティビティ: 2021/07/20), 'AmazonPersonalizeExecutorRole-135230770300' (AWS サービス: personalize, なし), 'AWSDeepRacerCloudFormationAccessRole' (AWS サービス: cloudformation, なし), 'AWSDeepRacerLambdaAccessRole' (AWS サービス: lambda, なし), 'AWSDeepRacerRobotAccessRole' (AWS サービス: roborunner, なし), and 'AWSDeepRacerSageMakerAccessRole' (AWS サービス: sagemaker, なし). A large blue rectangular box highlights the 'Create New Role' button at the bottom of the table.

3. [信頼されたエンティティの種類] で [AWS サービス] が選択されていることを確認します。  
[ユースケースの選択] で [Lambda] をクリックして選択状態にします。



4. [次のステップ: アクセス権限] をクリックします。

5. ポリシーの一覧から、前手順で作成した [YYYYMMDDserverlessLambdaPolicy] を探して、左側のチェックボックスにチェックを入れます。



6. [次のステップ: タグ] をクリックします。

7. [タグの追加] ページでは何も入力せず、[次のステップ: 確認] をクリックします。
8. [ロール名] 欄に [**YYYYMMDDserverlessLambdaRole**] と入力します。 (**YYYYMMDD** は本日の日付)



The screenshot shows the final step of creating a new AWS Lambda role. The role name is set to "20200901serverlessLambdaRole". The policy selected is "20200901serverlessLambdaPolicy". There are no tags added. The "Next Step" button is highlighted in blue.

9. [ロールの作成] をクリックします。
10. ロール [**YYYYMMDDserverlessLambdaRole**] が作成されたことを確認します。



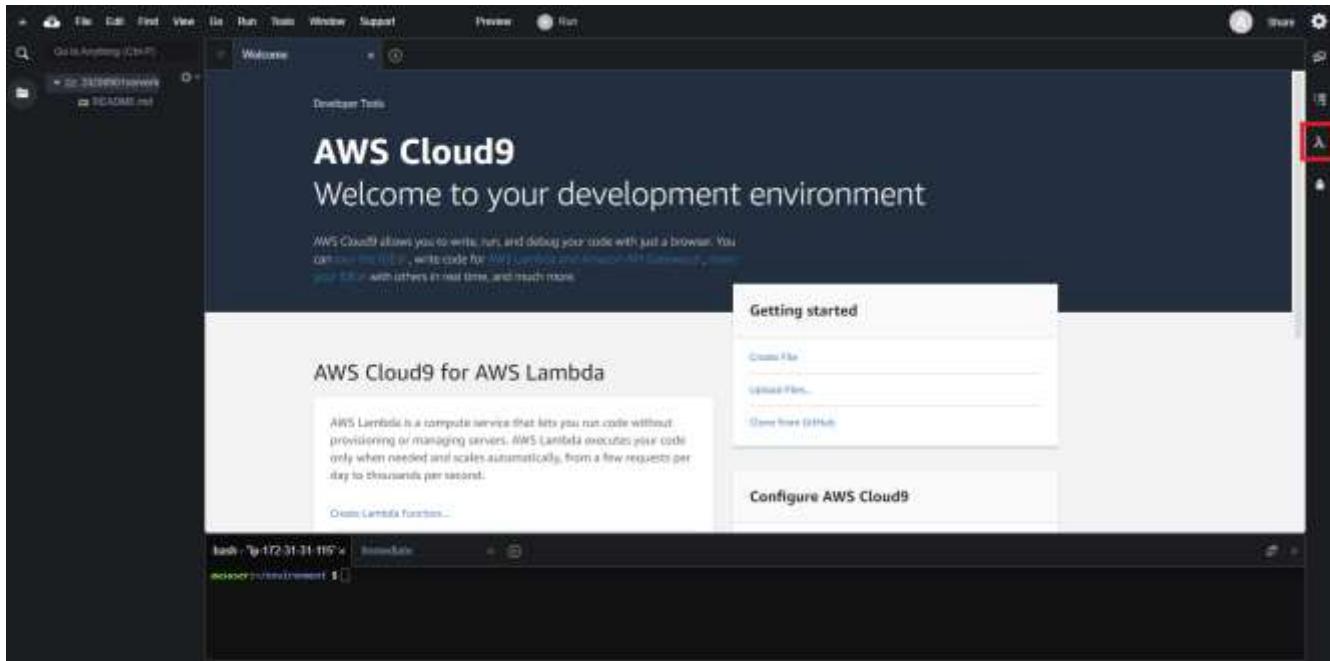
The screenshot shows the AWS IAM Roles list. A new role named "20200901serverlessLambdaRole" has been created and is listed at the top. It is associated with the "lambda" service and has a status of "なし" (None).

ロール名	権限されたエンティティ	最後のアクティビティ
20200901serverlessLambdaRole	AWS サービス: lambda	なし
AmazonEKSFargateFullExecutionRole	AWS サービス: eks-fargate-jobs	260 日間
AmazonPersonalize-ExecutionRole-190318790823	AWS サービス: personalize	なし
AWSDeepRacerCloudFormationAccessRole	AWS サービス: cloudformation	なし
AWSDeepRacerLambdaAccessRole	AWS サービス: lambda	なし
AWSDeepRacerRobobuilderAccessRole	AWS サービス: robomaker	なし
AWSDeepRacerStageManagerAccessRole	AWS サービス: stepfunctions	なし
AWSDeepRacerServiceRole	AWS サービス: stepfunctions	29 日間

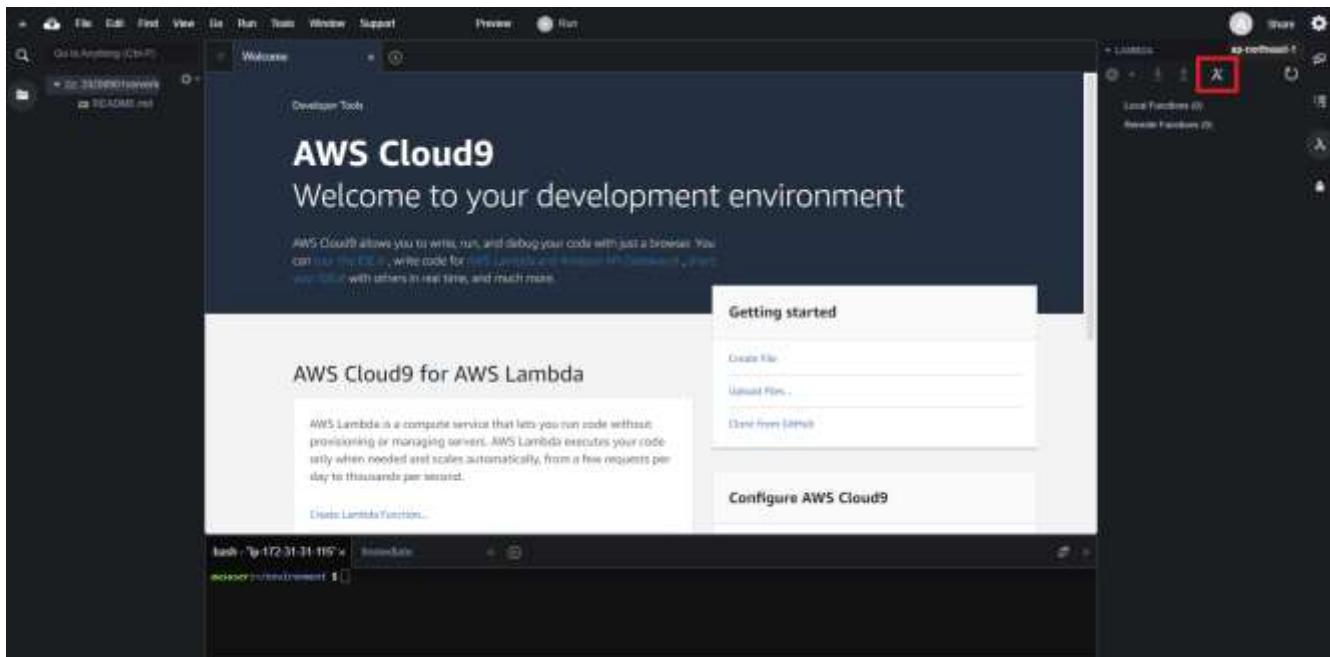
#### 4.1.4. Cloud9 上での Lambda 関数の作成とテスト (Write 関数)

1. Cloud9 の画面へ移動します。

画面右側のツールバーから [Lambda] ボタンをクリックします。表示が異なる場合は[AWS Resource]を選んでください。

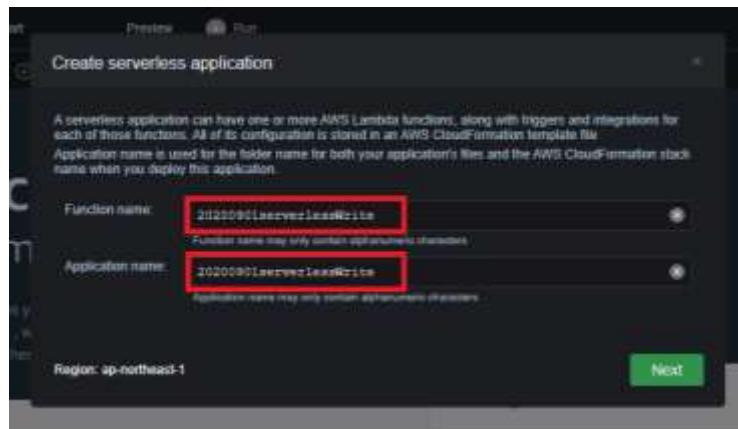


2. [Create] ボタン (Lambda アイコンに[+]印が付いたもの) をクリックします。



3. [Function name] 欄に [YYYYMMDDserverlessWrite] と入力します。 (YYYYMMDD は本日の日付)

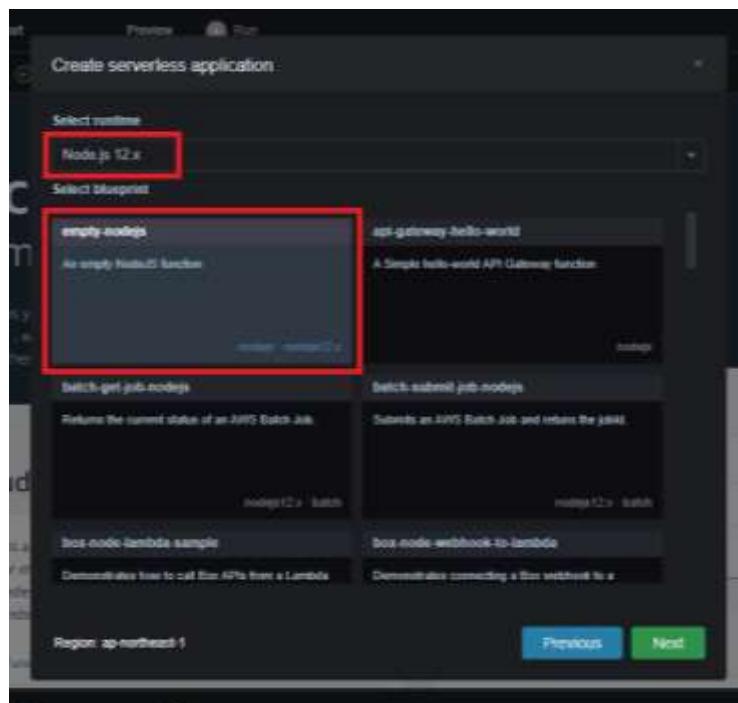
[Application name] 欄は [Function name] 欄と同じ内容が自動的に入力されますので、そのままにします。



4. [Next] をクリックします。

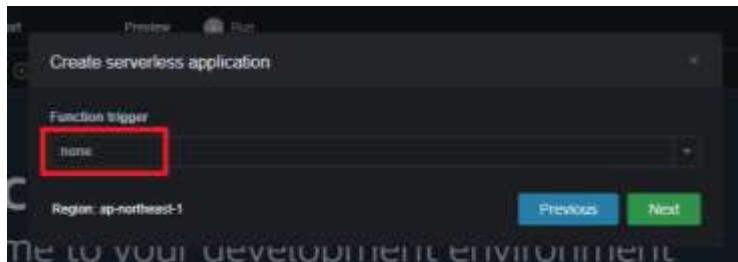
5. [Select runtime] で [Node.js 12.x] を選択します。

[Select blueprint] の中から [empty-nodejs] をクリックして選択状態にします。



6. [Next] をクリックします。

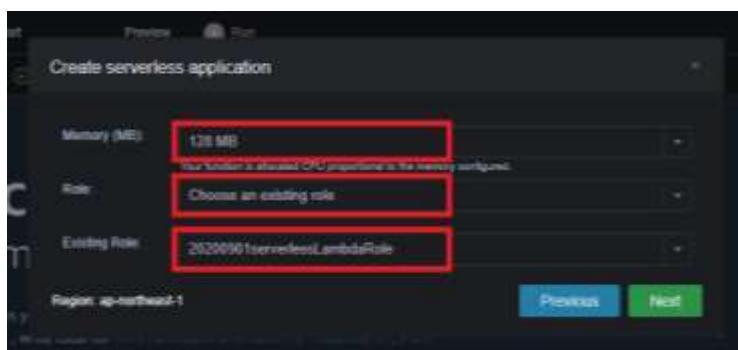
7. [Function trigger] で [none] を選択します。



8. [Next] をクリックします。

9. 以下の通りに選択します。

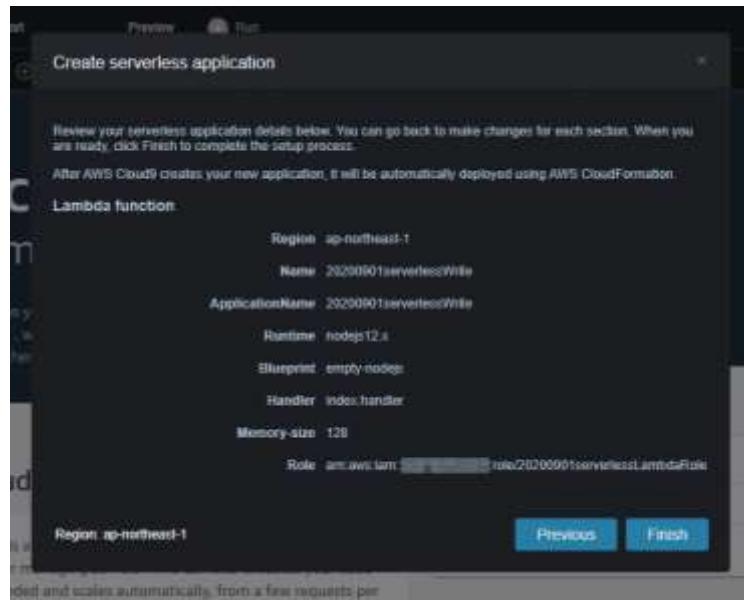
- **Memory:** [128 MB] (デフォルトのまま)
- **Role:** [Choose an existing role]
- **Existing Role:** 前手順で作成した [**YYYYMMDDserverlessLambdaRole**] を選択



注意：この手順で先に作成した IAM ロールが表示されない場合、IAM ロールに以下のポリシーを追加で作成して再度試してください。[[AmazonDynamoDBFullAccess](#)] [[AWSCloud9Administrator](#)]

10. [Next] をクリックします。

11. 確認画面になりますので、[Finish] をクリックします。



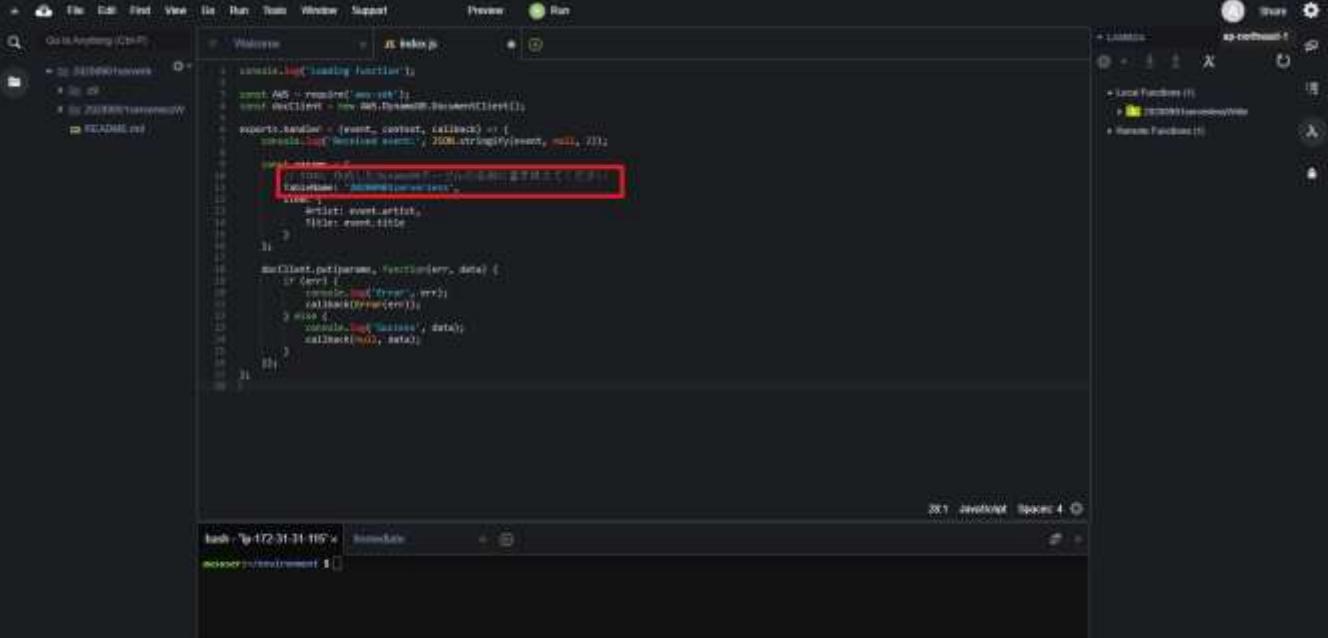
12. Lambda 関数コード (index.js) の編集画面が表示されます。

The screenshot shows the AWS Cloud9 code editor with the 'index.js' file open. The code defines a Lambda function handler:

```
exports.handler = (event, context, callback) => {
    console.log('Hello');
    callback();
};
```

13. 初期入力されているサンプルコードを一旦全て削除します。

フォルダ内の **[lambda\_function\_write.txt]** の内容をコピーして編集画面にペーストします。コード中に DynamoDB のテーブル名が記述されている箇所がありますので、実際に作成したテーブルの名前に書き換えます。

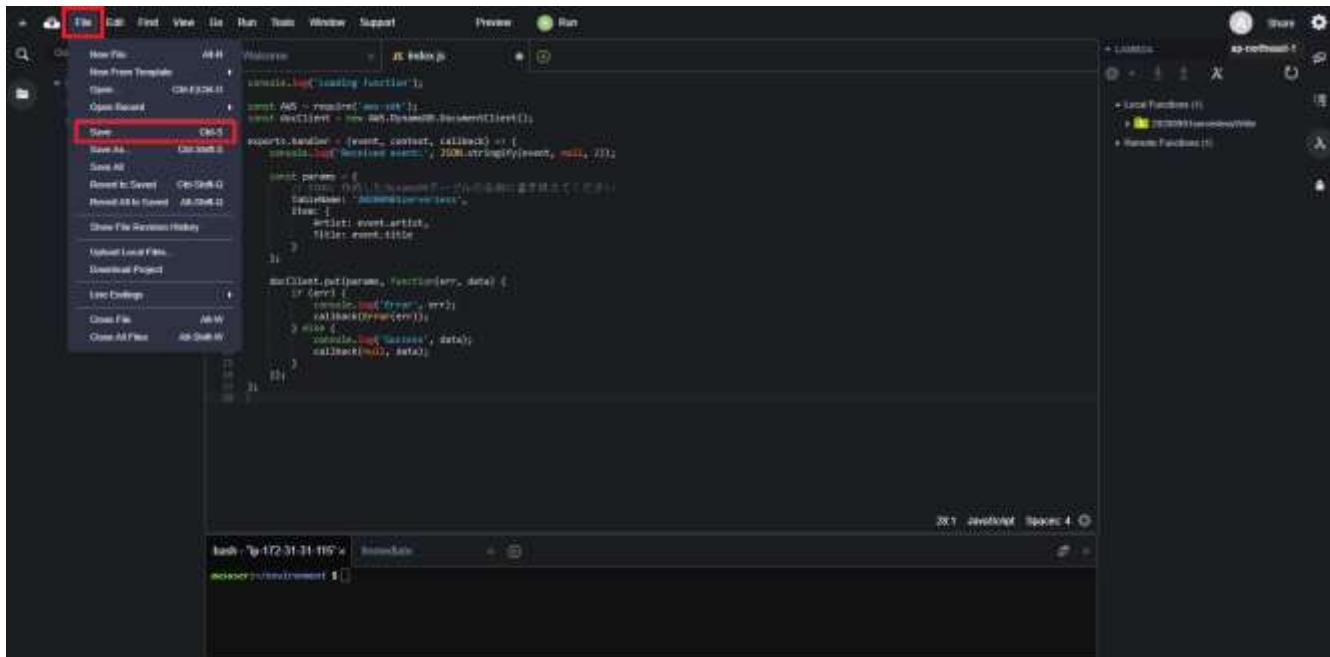


The screenshot shows the AWS Lambda Function Editor interface. The code editor window contains a JavaScript file named 'index.js' with the following content:

```
const AWS = require('aws-sdk');
const docClient = new AWS.DynamoDB.DocumentClient();
exports.handler = (event, context, callback) => {
    console.log(`Received event: ${JSON.stringify(event, null, 2)}`);
    const params = {
        TableName: 'DynamoDBTutorial',
        Item: {
            artist: event.artist,
            title: event.title
        }
    };
    docClient.put(params, function(err, data) {
        if (err) {
            callback(err);
        } else {
            callback(null, data);
        }
    });
};
```

The 'Artist' field in the 'Item' object is highlighted with a red box. Below the code editor is a terminal window showing the command 'awscloudformation \$'.

14. コードの編集が終わりましたら、画面上部のメニューバーから **[File]→[Save]** の順に選択してファイルを保存します。



15. 画面上部の **[Run]** をクリックします。

The screenshot shows the AWS Lambda function editor interface. On the left, the file structure is visible with 'lambda\_function\_write\_test.js' selected. The main area contains the following code:

```
const AWS = require('aws-sdk');
const docClient = new AWS.DynamoDB.DocumentClient();
exports.handler = (event, context, callback) => {
    console.log(`Received event: ${JSON.stringify(event, null, 2)}`);
    const params = {
        TableName: 'DynamodbMovies',
        Item: {
            artist: event.artist,
            title: event.title
        }
    };
    docClient.put(params, function(err) {
        if (err) {
            console.error(`Error ${err}`);
            callback(err);
        } else {
            console.log(`Success!`, params);
            callback(null, params);
        }
    });
};
```

The 'Run' button is highlighted in red at the top right of the editor.

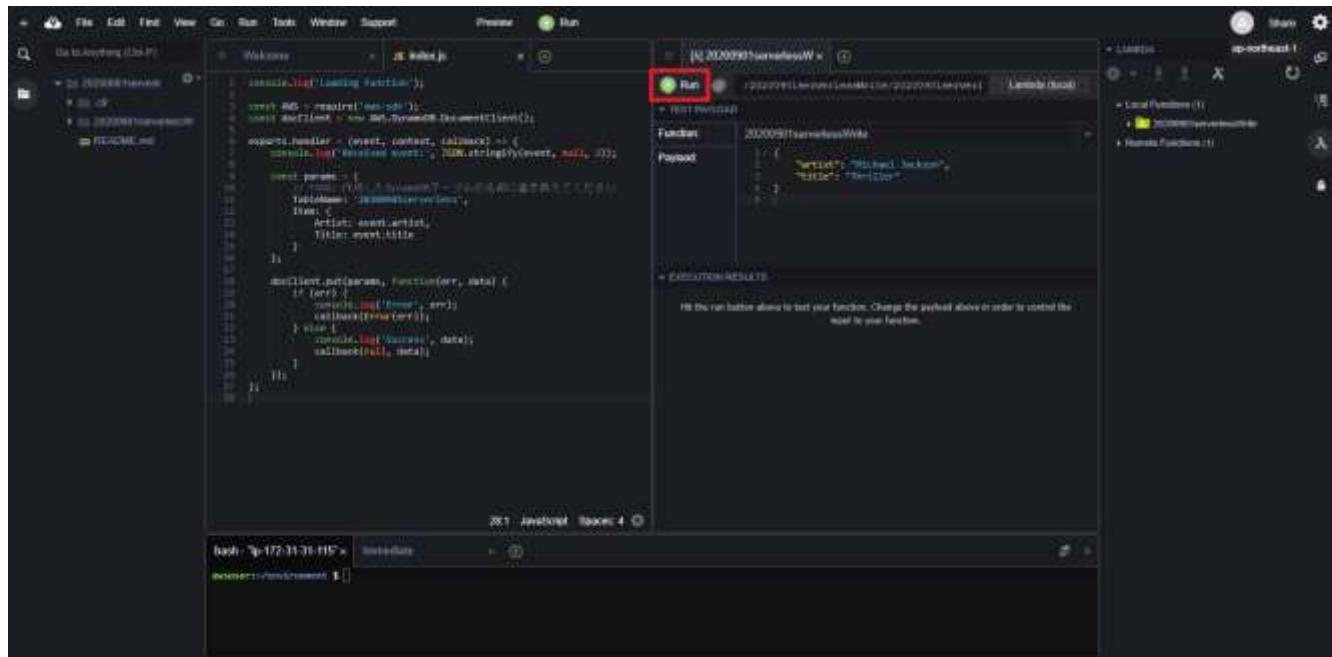
16. [lambda\_function\_write\_test.txt] の内容をコピーして [Payload] 欄にペーストします。

The screenshot shows the AWS Lambda function editor interface after pasting the payload from 'lambda\_function\_write\_test.txt' into the 'Payload' field. The 'Payload' field now contains:

```
"Artist": "Michael Jackson",
"Title": "Thriller"
```

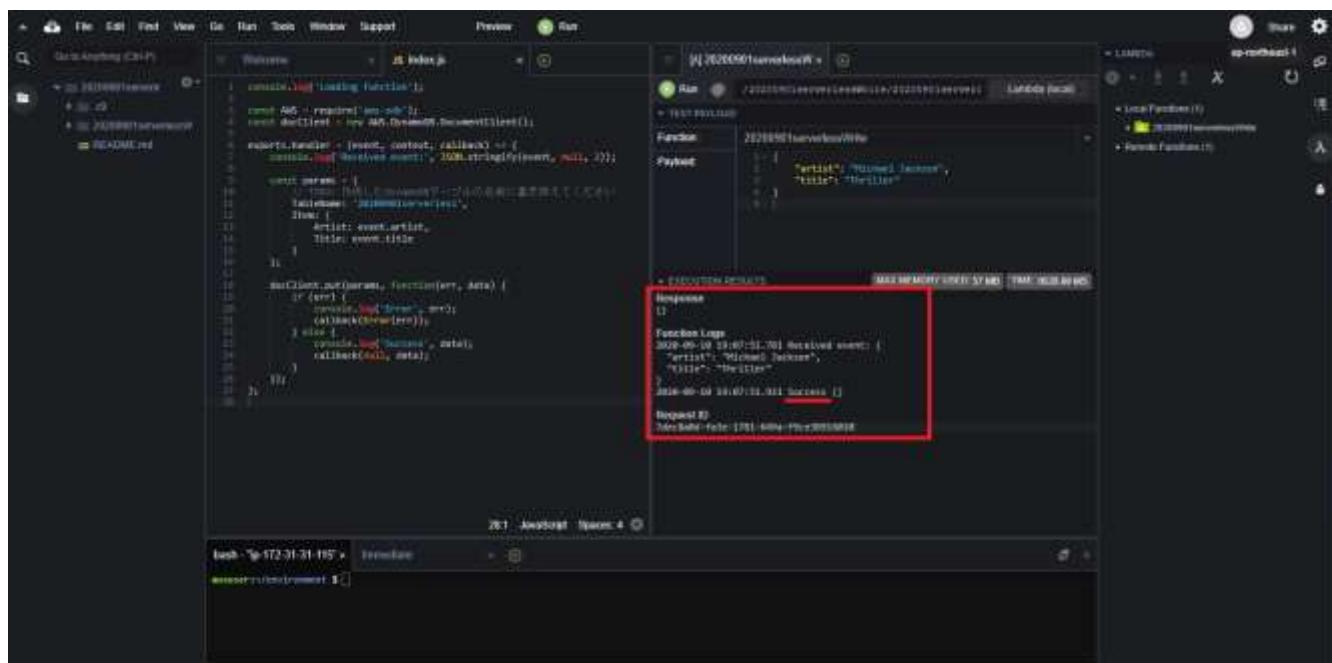
The rest of the code and interface remain the same as in the previous screenshot.

17. Lambda 関数実行タブの [Run] をクリックします。



18. [EXECUTION RESULTS] 欄に実行結果が表示されます。

エラー等が発生しておらず、[Success] と表示されていれば成功です。



19. DynamoDB の画面に移動して、テーブルの【項目】タブを表示します。

リロードボタンをクリックして、Lambda 関数によって書き込まれたデータが表示されていることを確認します。

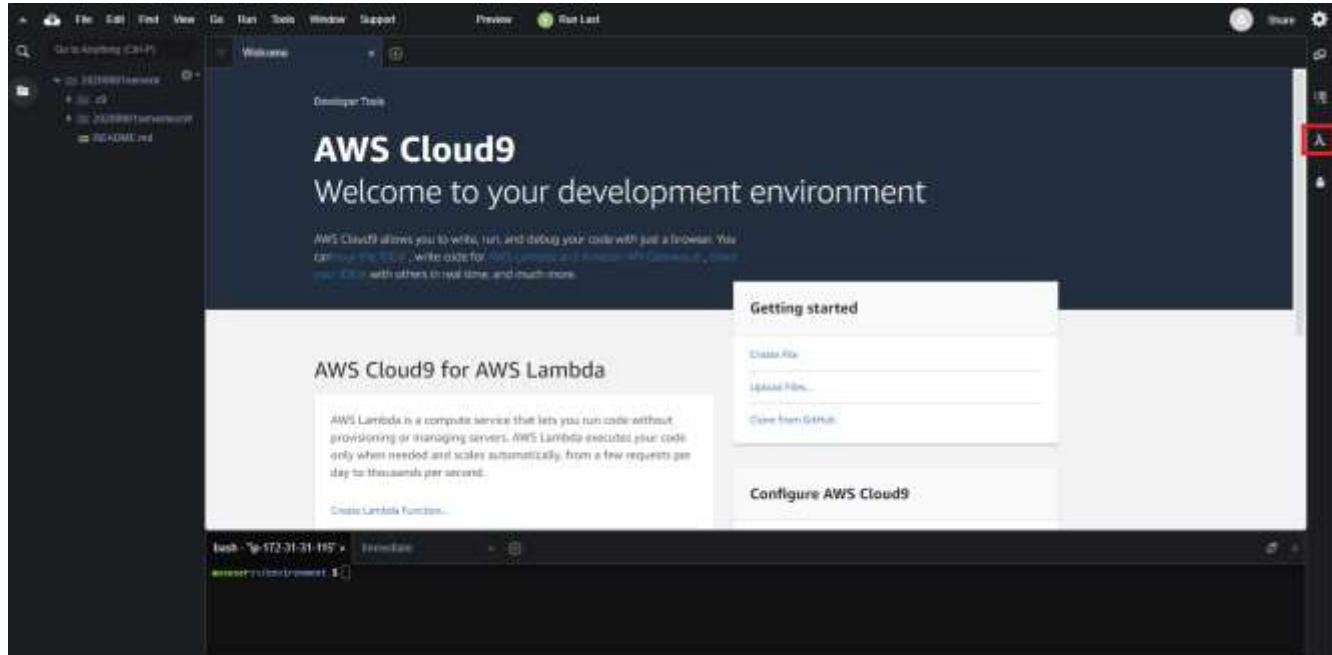
The screenshot shows the AWS DynamoDB console interface. On the left, the navigation pane is visible with options like 'DynamoDB', 'ダッシュボード', 'テーブル', 'バックアップ', etc. The main area displays a table titled '20200901serverless' with one item listed:

Artist	Title
Michael Jackson	Thriller

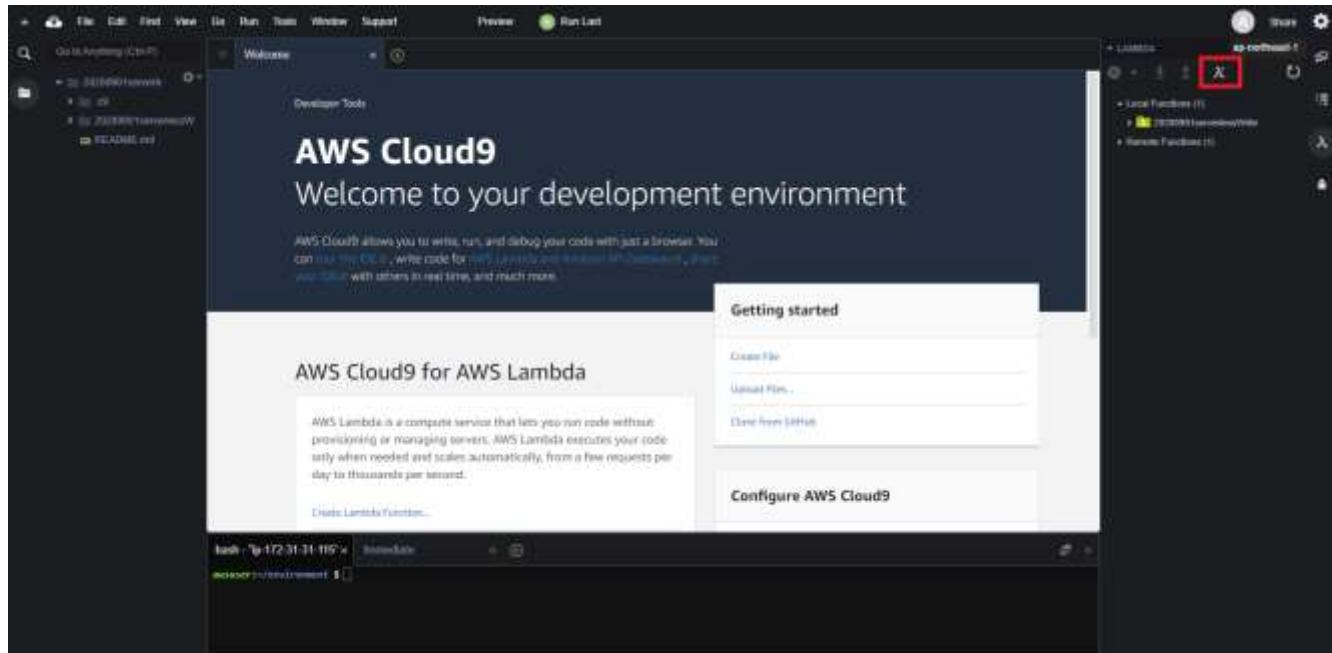
A red box highlights the 'Michael Jackson' entry in the 'Artist' column. In the top right corner of the main window, there is a red box around the refresh icon (a circular arrow).

#### 4.1.5. Cloud9 上での Lambda 関数の作成とテスト (Read 関数)

1. 画面右側のツールバーから [Lambda] ボタンをクリックします。表示されない場合は[AWS Resources]を押します。

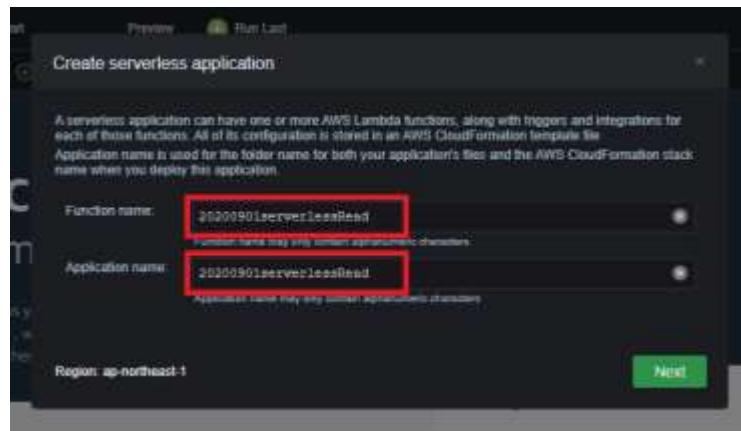


2. [Create] ボタン (Lambda アイコンに[+]印が付いたもの) をクリックします。



3. [Function name] 欄に [YYYYMMDDserverlessRead] と入力します。 (YYYYMMDD は本日の日付)

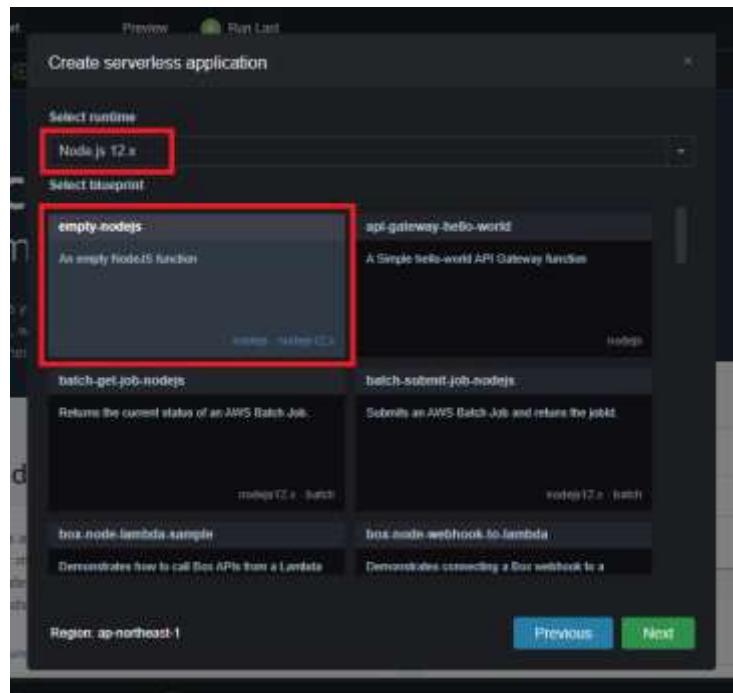
[Application name] 欄は [Function name] 欄と同じ内容が自動的に入力されますので、そのままにします。



4. [Next] をクリックします。

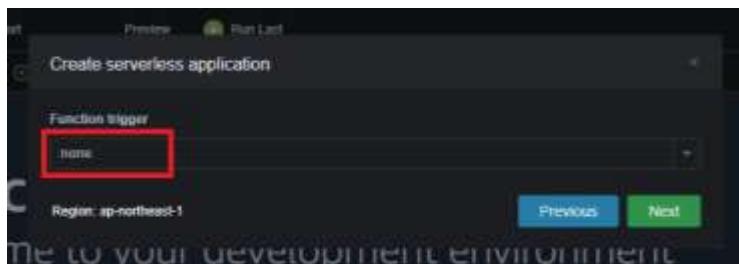
5. [Select runtime] で [Node.js 12.x] を選択します。

[Select blueprint] の中から [empty-nodejs] をクリックして選択状態にします。



6. [Next] をクリックします。

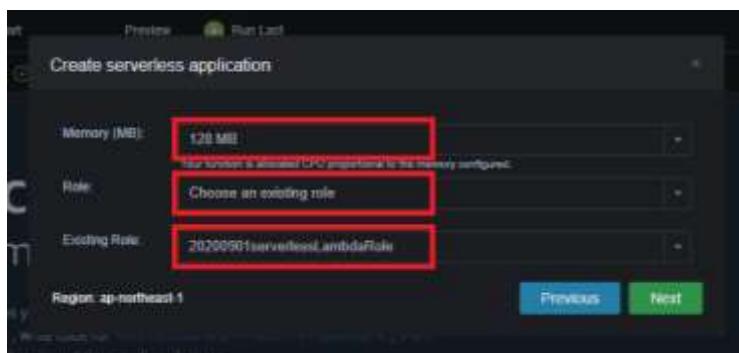
7. [Function trigger] で [none] を選択します。



8. [Next] をクリックします。

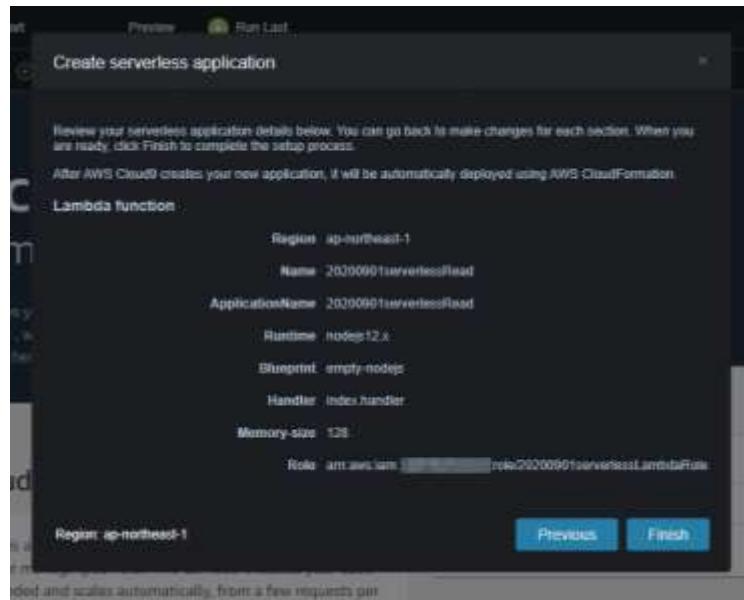
9. 以下の通りに選択します。

- **Memory:** [128 MB] (デフォルトのまま)
- **Role:** [Choose an existing role]
- **Existing Role:** 前々項で作成した [**YYYYMMDDserverlessLambdaRole**] を選択



10. [Next] をクリックします。

11. 確認画面になりますので、[Finish] をクリックします。



12. Lambda 関数コード (index.js) の編集画面が表示されます。

The screenshot shows the AWS Cloud9 code editor displaying the 'index.js' file. The code is as follows:

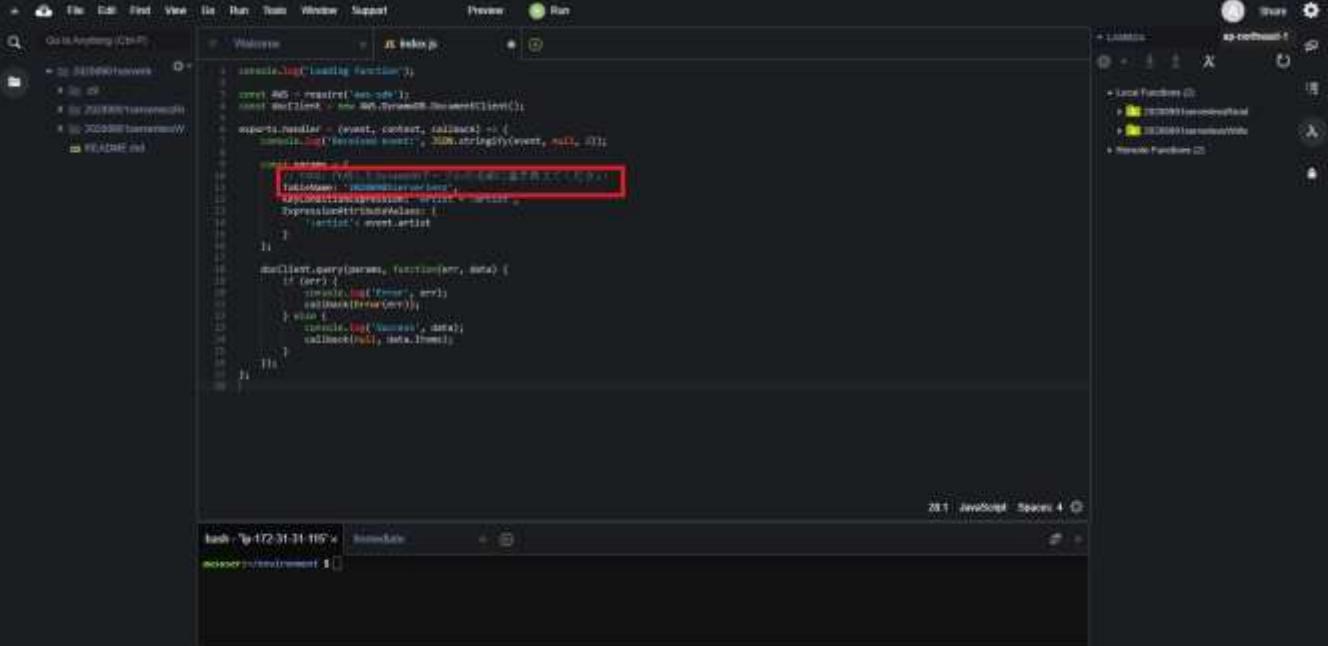
```
exports.handler = (event, context, callback) => {
    callback(null);
}
```

The code editor interface includes a sidebar with project files like '20200901serverless' and 'README.md', and a bottom navigation bar with tabs for 'Index.js' and 'Logs'.

13. 初期入力されているサンプルコードを一旦全て削除します。

[lambda\_function\_read.txt] の内容をコピーして編集画面にペーストします。

コード中に DynamoDB のテーブル名が記述されている箇所がありますので、実際に作成したテーブルの名前に書き換えます。

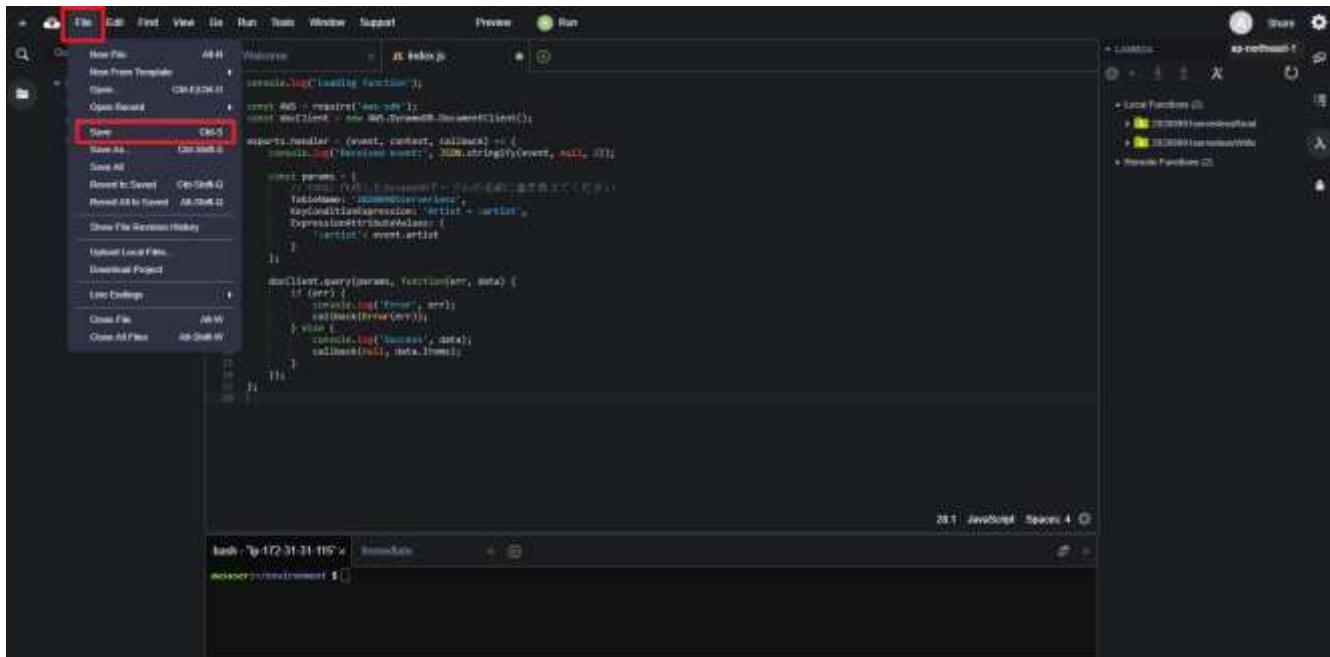


The screenshot shows the AWS Lambda Function Editor interface. The code editor window contains the following JavaScript code:

```
const AWS = require('aws-sdk');
const ddbClient = new AWS.DynamoDB.DocumentClient();
exports.handler = (event, context, callback) => {
    const query = event.query;
    console.log(`DynamoDB: ${query}`);
    const artistName = query.artist;
    const expressionAttributeValues = {
        artist: event.artist
    };
    const expressionAttributeNames = {
        artist: `:${artistName}`
    };
    ddbClient.query(query, function(err, data) {
        if (err) {
            console.error(`Error: ${err}`);
            callback(err);
        } else {
            console.log(`Success: ${data.Items}`);
            callback(null, data.Items);
        }
    });
};
```

The line `const artistName = query.artist;` is highlighted with a red rectangle.

14. コードの編集が終わりましたら、画面上部のメニューバーから [File]→[Save] の順に選択してファイルを保存します。



15. 画面上部の [Run] をクリックします。

The screenshot shows the AWS Lambda function editor. The left pane displays the function's code in a text editor:

```
console.log('Loading function');

const AWS = require('aws-sdk');
const docClient = new AWS.DynamoDB.DocumentClient();
exports.handler = (event, context, callback) => {
    console.log('Received event:', JSON.stringify(event, null, 2));
    const params = {
        TableName: 'DocumentDBExample',
        KeyConditionExpression: 'Artist = :Artist',
        ExpressionAttributeValues: {
            ':Artist': event.artist
        }
    };

    docClient.query(params, function(err, data) {
        if (err) {
            callback(err);
        } else {
            console.log("Success", data);
            callback(null, data.Items);
        }
    });
};
```

The right pane shows the Lambda function configuration, including the function name and a Local Variables panel.

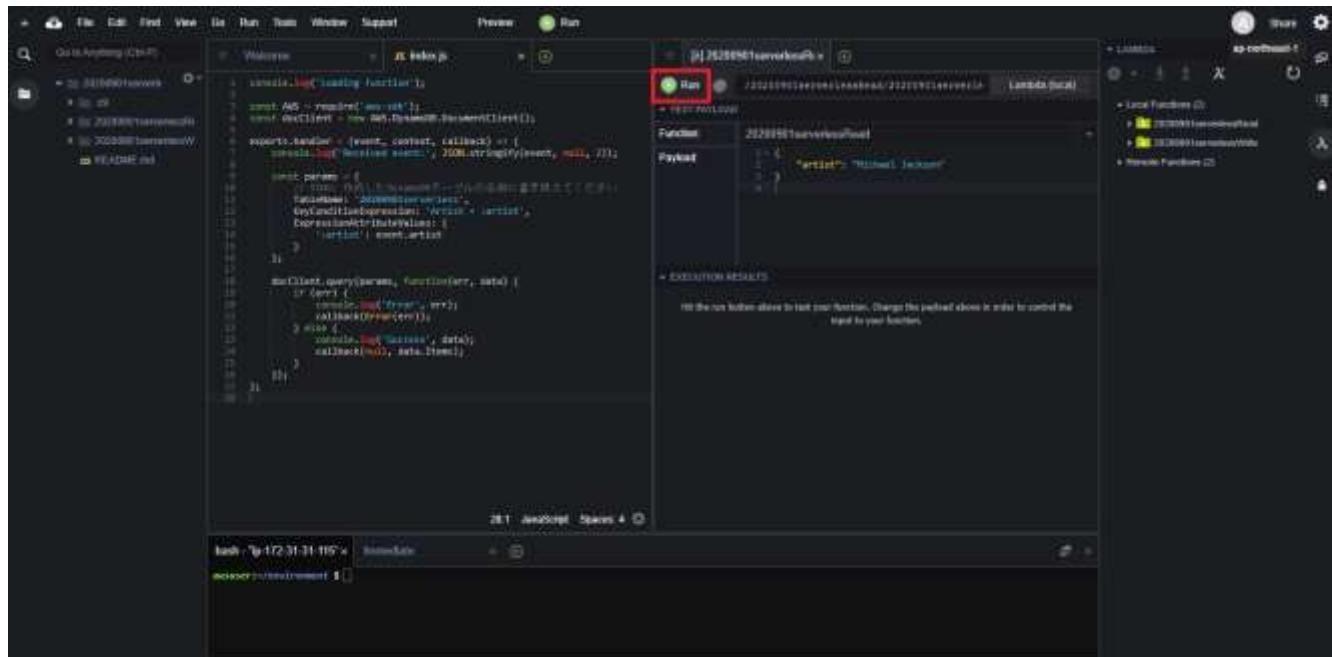
16. [lambda\_function\_read\_test.txt] の内容をコピーして [Payload] 欄にペーストします。

The screenshot shows the AWS Lambda function editor with a payload value pasted into the Payload field:

Payload: "Artist": "Michael Jackson"

The rest of the interface is identical to the previous screenshot, showing the function code and configuration.

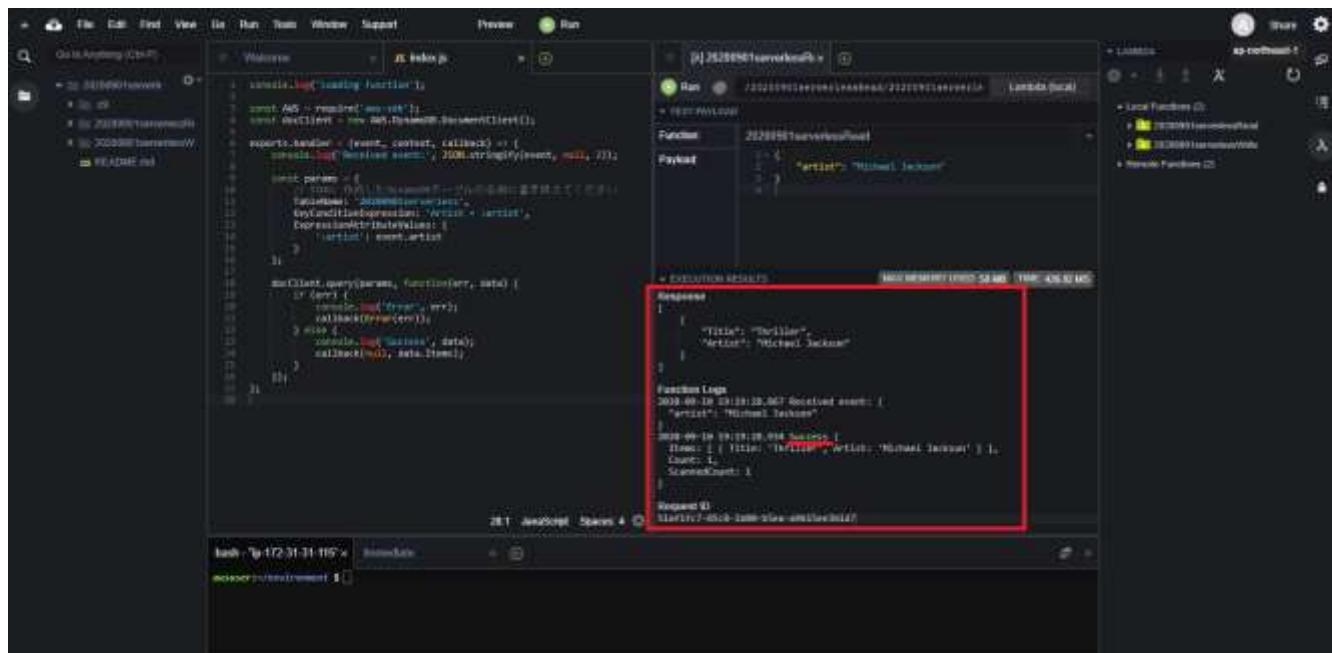
17. Lambda 関数実行タブの [Run] をクリックします。



18. [EXECUTION RESULTS] 欄に実行結果が表示されます。

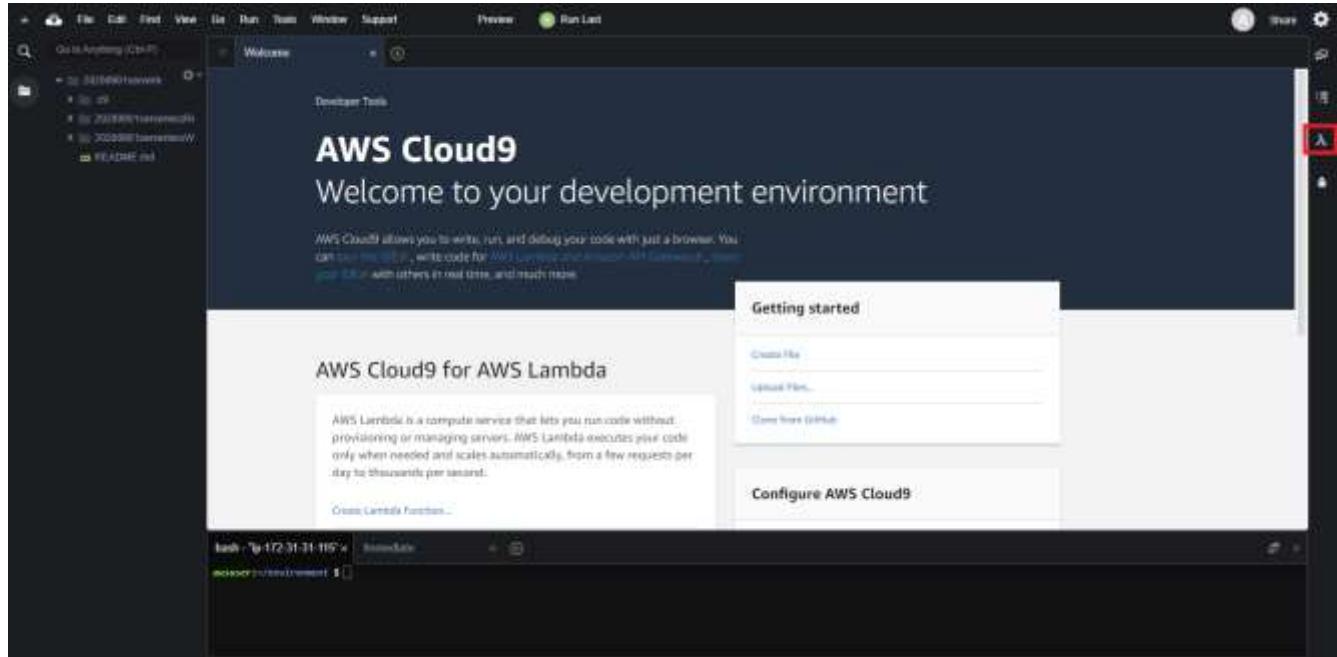
エラー等が発生しておらず、[Success] と表示されていれば成功です。

DynamoDB のデータが [Response] に表示されていることを確認します。



#### 4.1.6. Lambda 関数のデプロイ

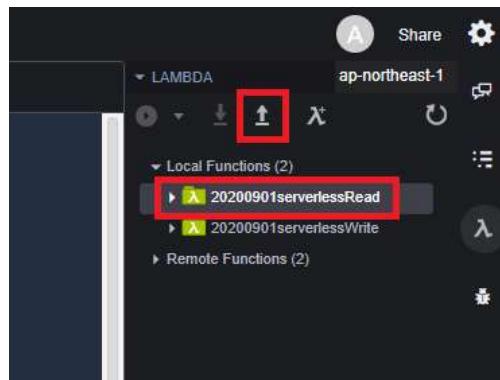
1. 画面右側のツールバーから [Lambda] ボタンをクリックします。



2. [Local Functions] (=Cloud9 上に存在する関数) 配下に 2 つのフォルダがあります。

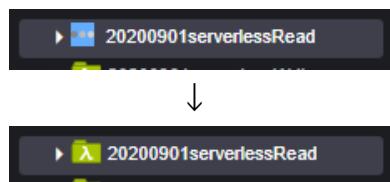
まず、[YYYYMMDDserverlessRead] の方をクリックして選択状態にします。

[Deploy] ボタン ([↑]) をクリックします。

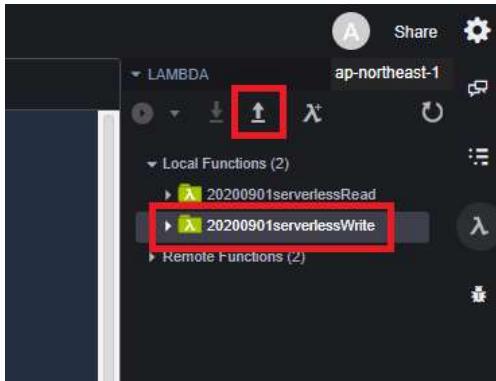


3. Lambda 関数のデプロイが行われます。

フォルダアイコンが「デプロイ中」の動きとなりますので、終わるまで待ちます。



4. 同様にして、[YYYYMMDDserverlessWrite] をクリックして選択状態にしてから、  
[Deploy] ボタンをクリックします。



5. デプロイが終わるまで待ちます。

6. AWS マネジメントコンソールのサービス一覧から [Lambda] を選択します。

関数の一覧に [cloud9-YYYYMMDDserverless…] で始まる名前の関数が 2 つ存在することを確認します。

関数名	説明	ランタイム	コードサイズ	最終更新日時
cloud9-20200901serverless-20200901serverlessRead-HC3DEAWGOD08		Node.js 12.x	1.6 kB	1 分前
cloud9-20200901serverlessW-20200901serverlessWrite-106A59CH6725A		Node.js 12.x	1.6 kB	20 秒前

## 4.2. API Gateway を追加して動作確認を行う

### 4.2.1. API Gateway REST API 作成

1. AWS マネジメントコンソールのサービス一覧から **[API Gateway]** を選択します。

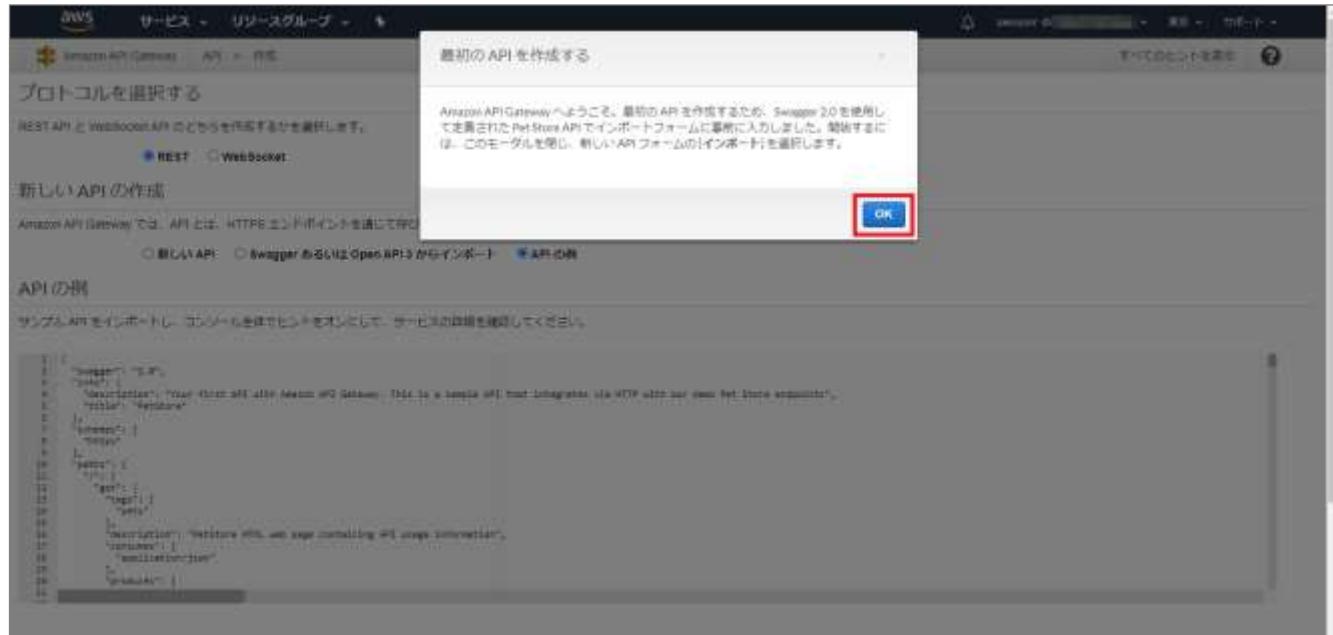
The screenshot shows the AWS Management Console with the search bar at the top containing 'API Gateway'. On the left, there's a sidebar with 'API Gateway' selected. The main content area has a title 'Amazon API Gateway' and a subtitle '規模を問わず API を作成、維持、保護する'. Below this, a section titled 'API タイプを選択' contains two options: 'HTTP API' and 'WebSocket API'. The 'HTTP API' section is expanded, showing its description and a note about CORS support. A red box highlights the 'REST API' option in the second section.

2. **[API タイプを選択]** の中から **[REST API]** の **[構築]** をクリックします。

※ **[REST API プライベート]** の方ではありませんので注意してください

The screenshot shows the 'REST API' section from the previous screen. A red box highlights the 'REST API' button. The 'REST API' section is expanded, showing its description and a note about security. A red box highlights the '構築' (Build) button. Below it, another section for 'REST API プライベート' is partially visible.

3. 初めて API Gateway を利用する場合は下図のダイアログが表示されますので、[OK] をクリックします。



[新しい API の作成] の選択肢では [新しい API] を選択します。

[API 名] 欄の入力ができるようになるので、[YYYYMMDDserverless] と入力します。

(YYYYMMDD は本日の日付)

[エンドポイントタイプ] は [リージョン] のままにします。

新しい API の作成

Amazon API Gateway では、API とは、HTTP エンドポイントを通じて呼び出し可能なリソースおよびメッシュの構造体のこととします。

新しい API  Swagger あるいは Open API からインポート  API の例

名前と説明

API のフレンドリー名と説明を選択します。

API 名:

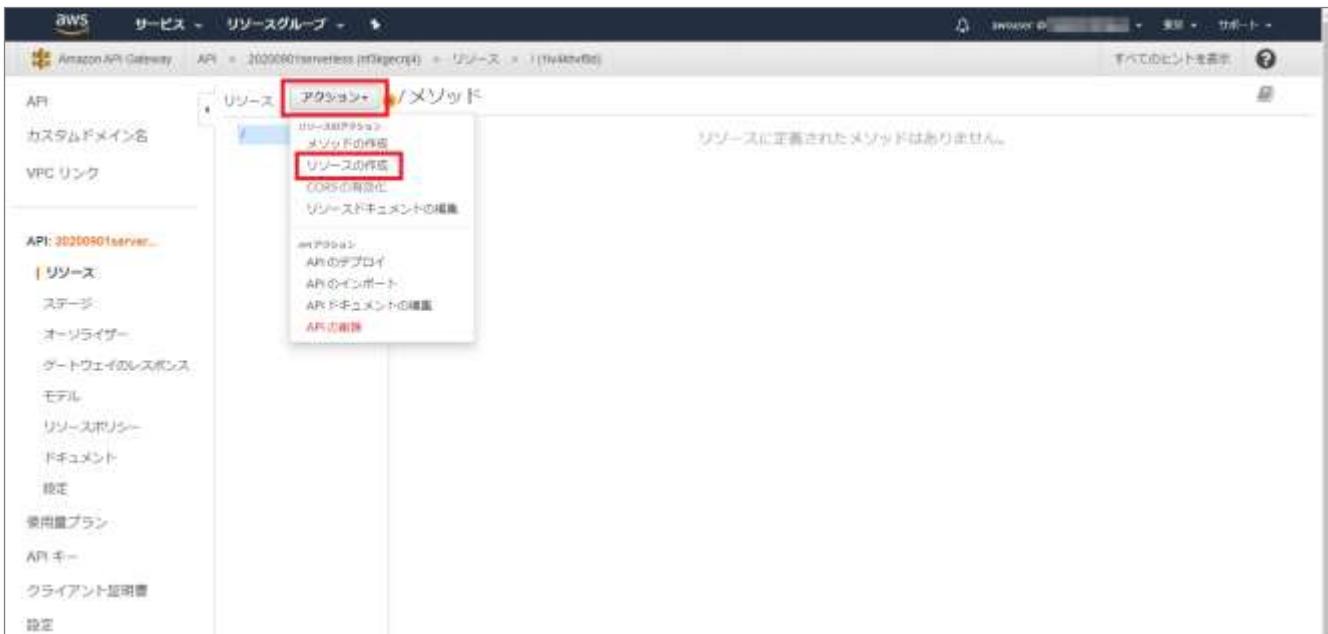
説明:

エンドポイントタイプ:

次へ API の作成

4. [API の作成] をクリックします。

5. 作成した API の画面になりますので、[アクション] プルダウンから [リソースの作成] を選択します。



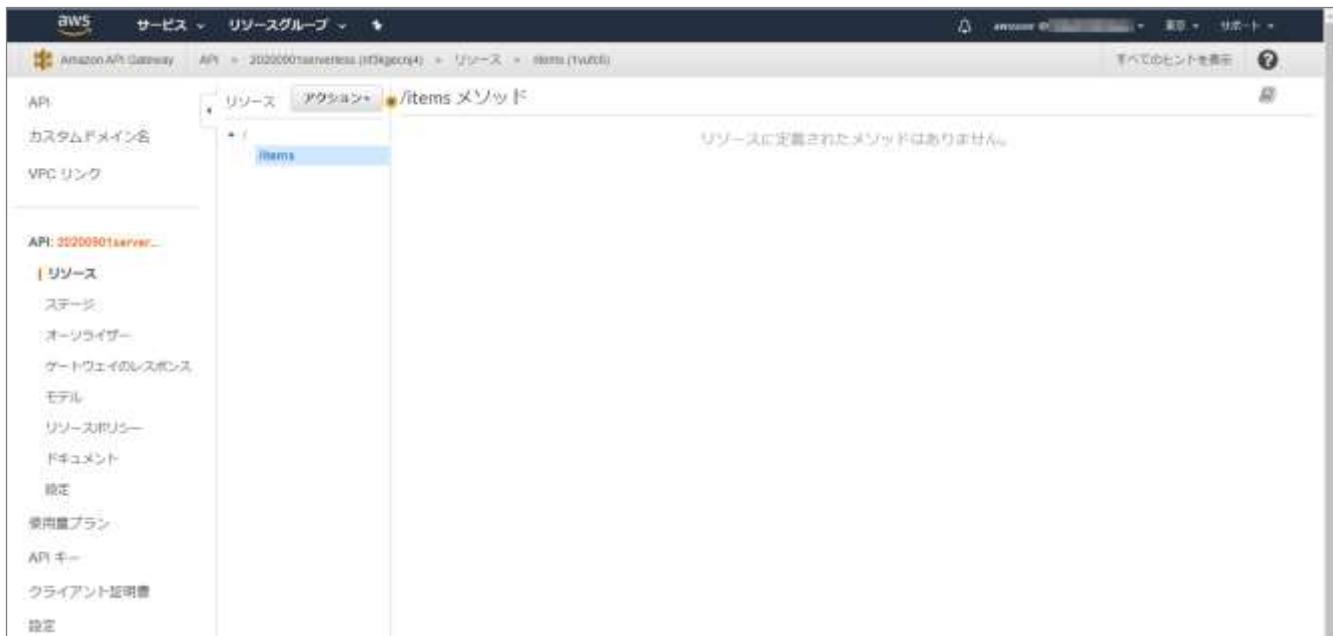
6. [リソース名] 欄に [items] と入力します。

[リソースパス] 欄は [リソース名] 欄と同じ内容が自動的に入力されますので、そのままにします。



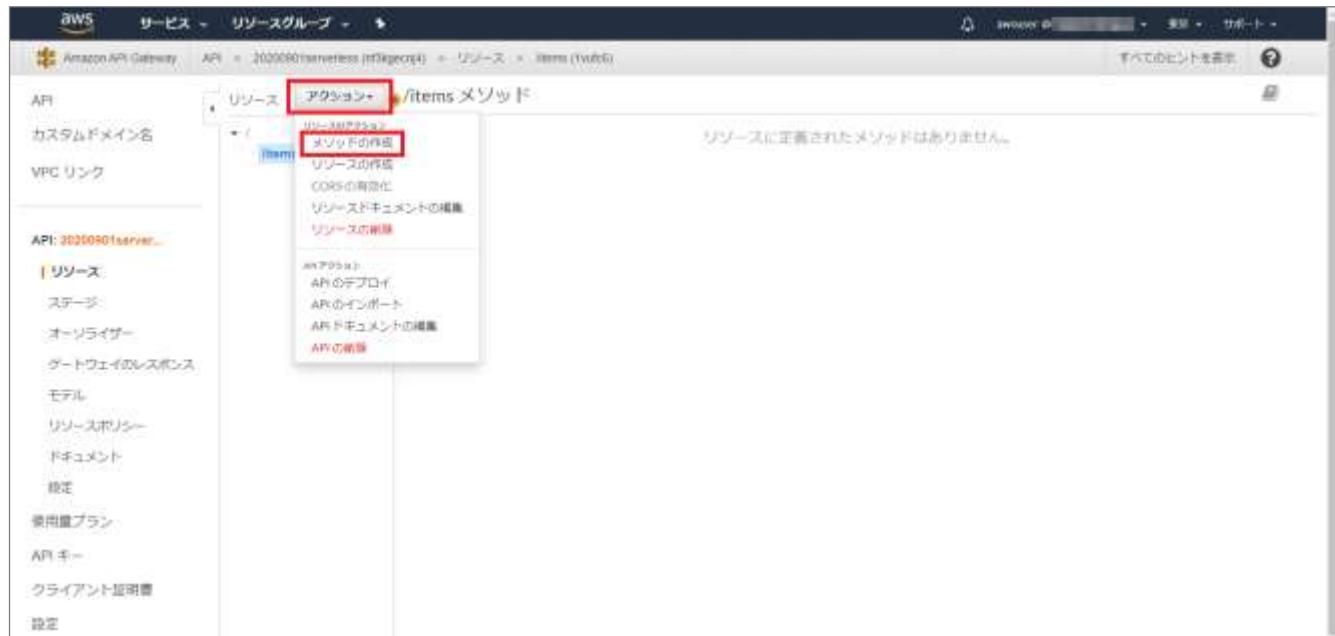
7. [リソースの作成] をクリックします。

8. リソース [/items] が作成されました。

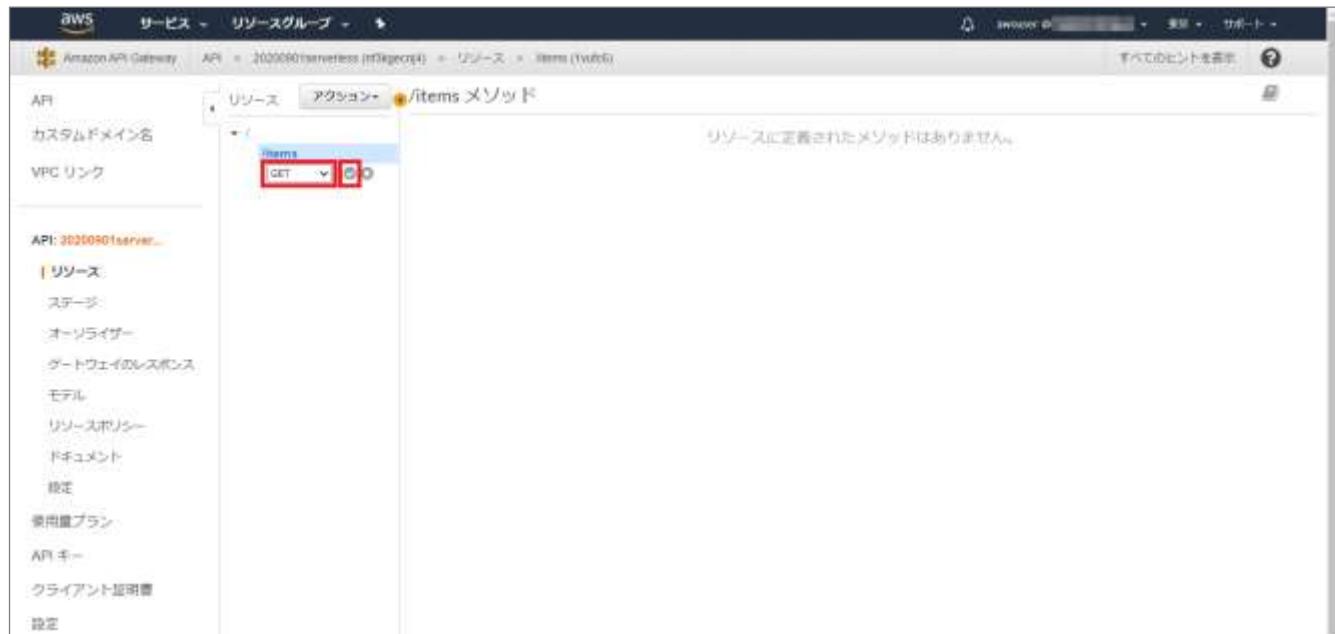


## 4.2.2. API に GET メソッドを追加

1. [/items] リソースの画面で、[アクション] プルダウンから [メソッドの作成] を選択します。



2. メソッド種類のプルダウンから [GET] を選択して、右側のチェックボタンをクリックします。



3. 【統合タイプ】はデフォルトの【Lambda 関数】のままにします。

【Lambda プロキシ統合の使用】のチェックは外したままにします。

【Lambda 関数】欄に、前節で作成した Lambda 関数のうち「Read」の方の関数名を入力します。（関数名の一部、例えば【YYYYMMDD】を入力すると候補が表示されて容易に入力できます）（Read と Write は間違えやすいので注意です）

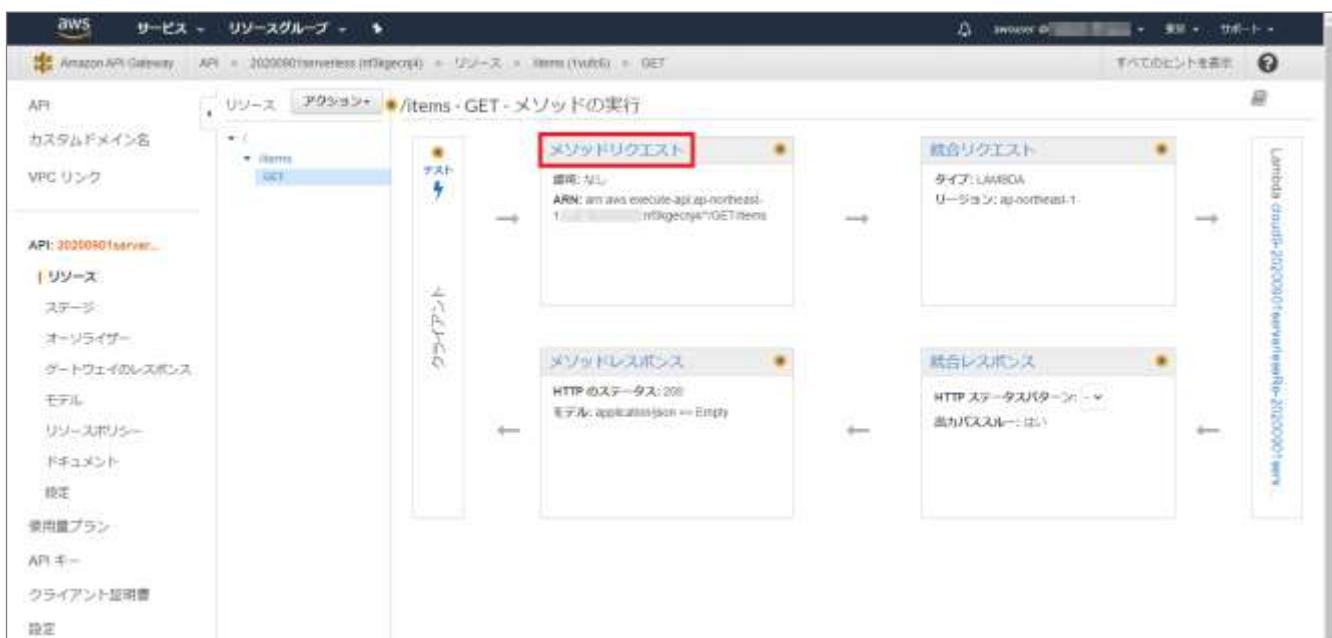
The screenshot shows the AWS Lambda function configuration interface. On the left, there's a sidebar with various options like API, VPC Link, and Lambda functions. The main area shows an API endpoint: /items - GET. Under the '統合タイプ' (Integration Type) section, the 'Lambda 関数' (Lambda Function) option is selected. Below it, the 'Lambda 関数' dropdown is set to 'cloud9-20200901serverlessRe-20200901serverlessRead-MC3X6MS0DKI0'. A red box highlights this dropdown. At the bottom right, there's a '保存' (Save) button.

This is a zoomed-in view of the Lambda function selection dropdown from the previous screenshot. The dropdown is labeled 'Lambda 関数' and contains the value '20200901'. A red box highlights this input field. Below the dropdown, a tooltip displays two Lambda function names: 'cloud9-20200901serverlessRe-20200901serverlessRead-MC3X6MS0DKI0' and 'cloud9-20200901serverlessW-20200901serverlessWrite-1X6A5XCM67J5A'. The entire configuration panel is enclosed in a light gray border.

4. [保存]を押すと、権限追加の確認ダイアログが表示されますので、[OK] をクリックします。



5. 作成した [GET] メソッドの画面になりますので、[メソッドリクエスト] をクリックします。



6. [URL クエリ文字列パラメータ] をクリックして展開します。

The screenshot shows the AWS API Gateway configuration interface. On the left, there's a sidebar with various service options like 'API', 'カスタムドメイン名', 'VPC リンク', etc. The main area shows an API named '20200901server...' with a single resource '/items'. Underneath it, there's a 'GET' method. To the right of the method, there's a '設定' (Settings) tab. Below the settings tab, there's a section titled 'URL クエリ文字列/パラメータ' (Query String Parameters). This section is currently expanded, showing a table with three columns: '名前' (Name), '必須' (Required), and 'キャッシュ' (Cache). There is one row in the table with the value 'クエリ文字列なし' (No query string parameters). Other collapsed sections include 'HTTP リクエストヘッダー', 'リクエスト本文書', and 'SDK 設定'.

7. [クエリ文字列の追加] をクリックします。

This screenshot is identical to the previous one, but the 'クエリ文字列' (Query String) button within the 'Query String Parameters' section is highlighted with a red box. This indicates the user has clicked on it to add a new parameter.

8. [名前] 欄に [artist] と入力して、右端のチェックボタンをクリックします。



The screenshot shows the AWS API Gateway console. On the left, there's a sidebar with various options like API, カスタムドメイン名, VPC リンク, etc. The main area shows an API named '20200901server...' with a single resource '/items'. Under '/items', there's a 'GET' method. The 'Actions' tab is selected. In the 'Query Parameters' section, there's a table with one row. The 'Name' column contains 'artist', which is highlighted with a red box. The '必須' (Required) column has a checkbox that is checked and also highlighted with a red box.

9. 画面上部の [ $\leftarrow$ メソッドの実行] をクリックして、前の画面に戻ります。



This screenshot is identical to the previous one, showing the AWS API Gateway configuration for the '/items' endpoint. The 'Name' field ('artist') and its 'Required' checkbox are still highlighted with red boxes. Additionally, the 'Execute Method' button at the top of the page is also highlighted with a red box.

## 10. [統合リクエスト] をクリックします。

The screenshot shows the AWS Lambda function configuration interface. On the left sidebar, under the 'API' section, '統合リクエスト' (Integration Request) is selected. The main area displays the flow of the integration:

- メソッドリクエスト**:
  - 操作: テスト
  - ARN: arn:aws:execute-api:ap-northeast-1:32000000000000000000:items/GET
  - クエリ文字列: item
- 統合リクエスト**:
  - 操作: テスト
  - タイプ: LAMBDA
  - リージョン: ap-northeast-1
- メソッドレスポンス**:
  - HTTP のステータス: 200
  - モデル: application/json -- Empty
- 統合レスポンス**:
  - HTTP ステータスバージョン: 1.1
  - 出力バスルート: /

## 11. 一番下の [マッピングテンプレート] をクリックして展開します。

The screenshot shows the expanded 'Mapping Template' section for the integration request step. The 'マッピングテンプレート' button at the bottom of the list is highlighted with a red box.

This section contains the following configuration:

- 統合タイプ**:
  - Lambda 関数 (selected)
  - HTTP
  - Mock
  - AWS サービス
  - VPC リンク
- Lambda プロビジョニングの使用**:
  - Lambda リージョン: ap-northeast-1
  - Lambda 関数: cloud9-20200911serverlessRe-20200001serverlessRead-HCQWwAS0K0
- 実行ロール**:
  - 実行者の認証情報を使用した呼び出し
  - 認証情報キャッシュ: 対象者の認証情報をキャッシュし、再利用する
- デフォルトタイムアウトの使用**:
  - URL/バス/パラメータ
  - URL クエリ文字列/パラメータ
  - HTTP ヘッダー
- マッピングテンプレート**: This section is expanded, showing the mapping template code.

12. [テンプレートが定義されていない場合(推奨)] を選択します。

[マッピングテンプレートの追加] をクリックします。



13. [Content-Type] 欄欄に [application/json] と入力して、右側のチェックボタンをクリックします。 (タイプミスに注意！)



14. [apigateway\_get\_mapping.txt] の内容をコピーしてテンプレートの入力欄にペーストします。

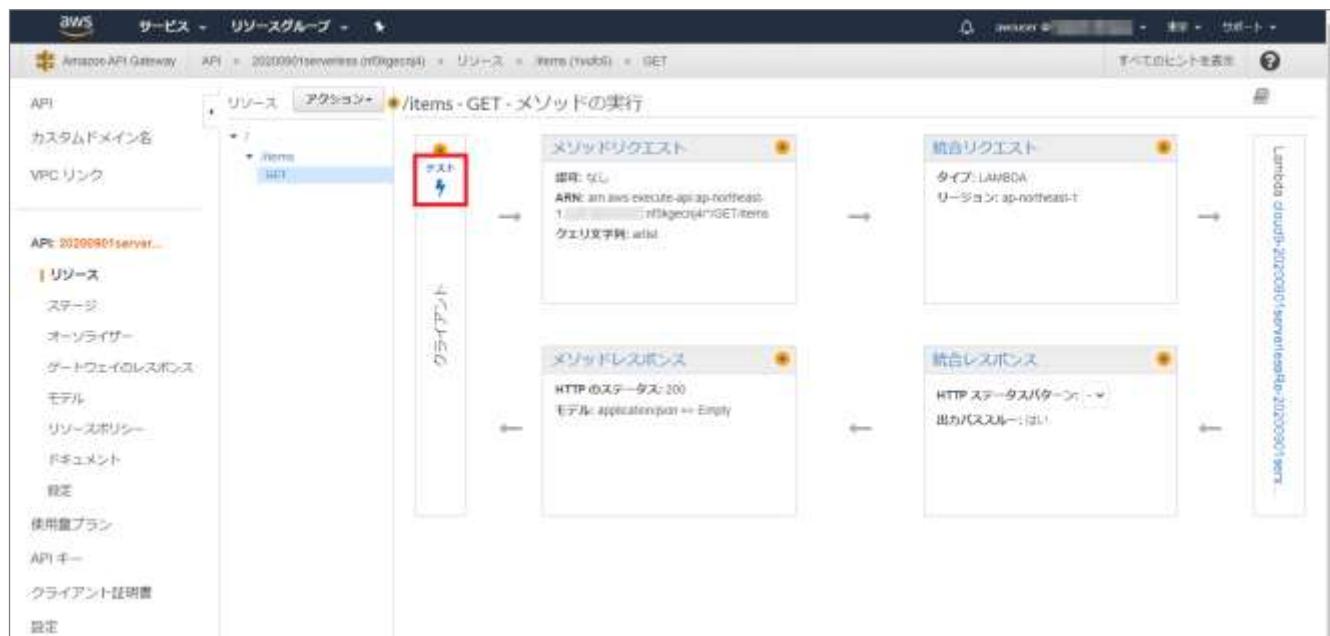


15. [保存] をクリックします。 (画面は変わりません)

16. 画面上部の [ $\leftarrow$ メソッドの実行] をクリックして、前の画面に戻ります。



17. [テスト] をクリックします。



18. [クエリ文字列] 欄に [artist=Michael Jackson] と入力します。

[◀ メソッドの実行 /items - GET - メソッドテスト](#)

指定された入力で メソッドに対してテストコールを行います。

パス

このリソースに対するパス/パラメータは存在しません。パス  
パラメータは、リソースパスにおいて構文  
{myPathParam} を使い定義します。

クエリ文字列

{items}

artist=Michael Jackson

ヘッダー

{items}

ヘッダー名と値を区切るときはコロン  
(:) を使用し、複数のヘッダーを宣言  
するときは改行を使用します。例:  
Accept:application/json

19. [テスト] をクリックすると、右側に結果が表示されます。

[ステータス] が [200] と表示されていれば成功です。

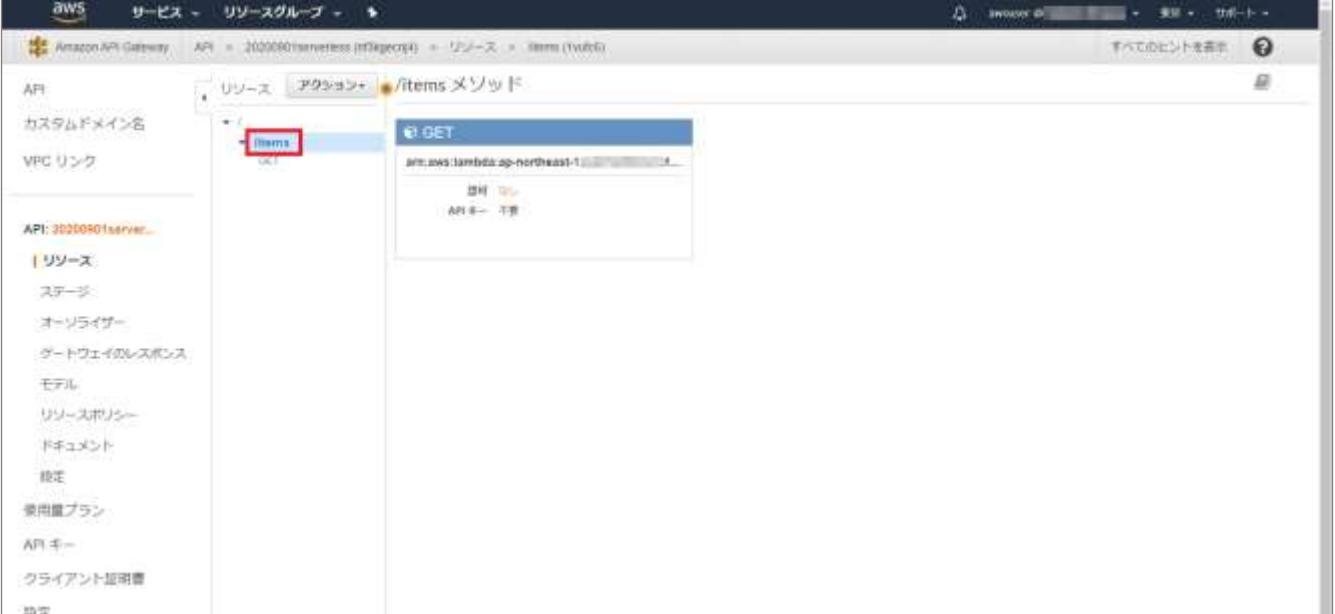
[レスポンス本文] に DynamoDB テーブルの内容が表示されていることを確認してください。

The screenshot shows the AWS API Gateway Test API interface. The left sidebar lists various API resources and stages. The main area shows a successful test run for the '/items' endpoint using the 'GET' method. The response body is highlighted with a red box and contains the following JSON:

```
[{"id": 1, "title": "Thriller", "Artist": "Michael Jackson"}]
```

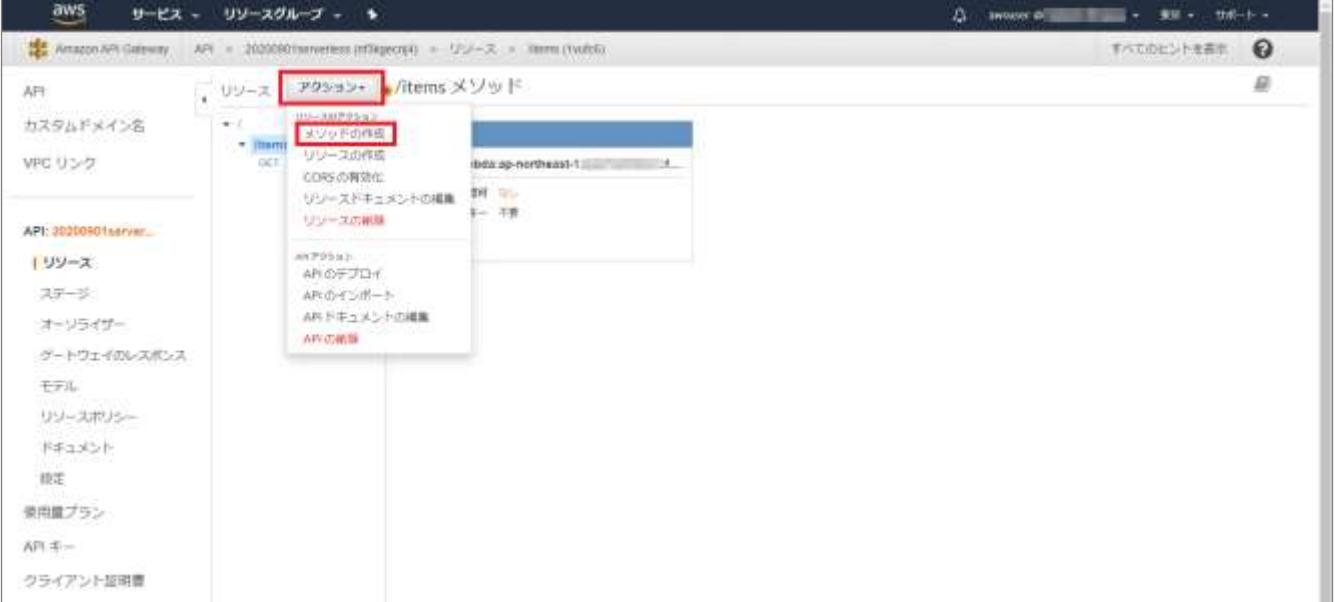
### 4.2.3. API に POST メソッドを追加

1. リソースのツリー階層から [/items] をクリックして選択します。



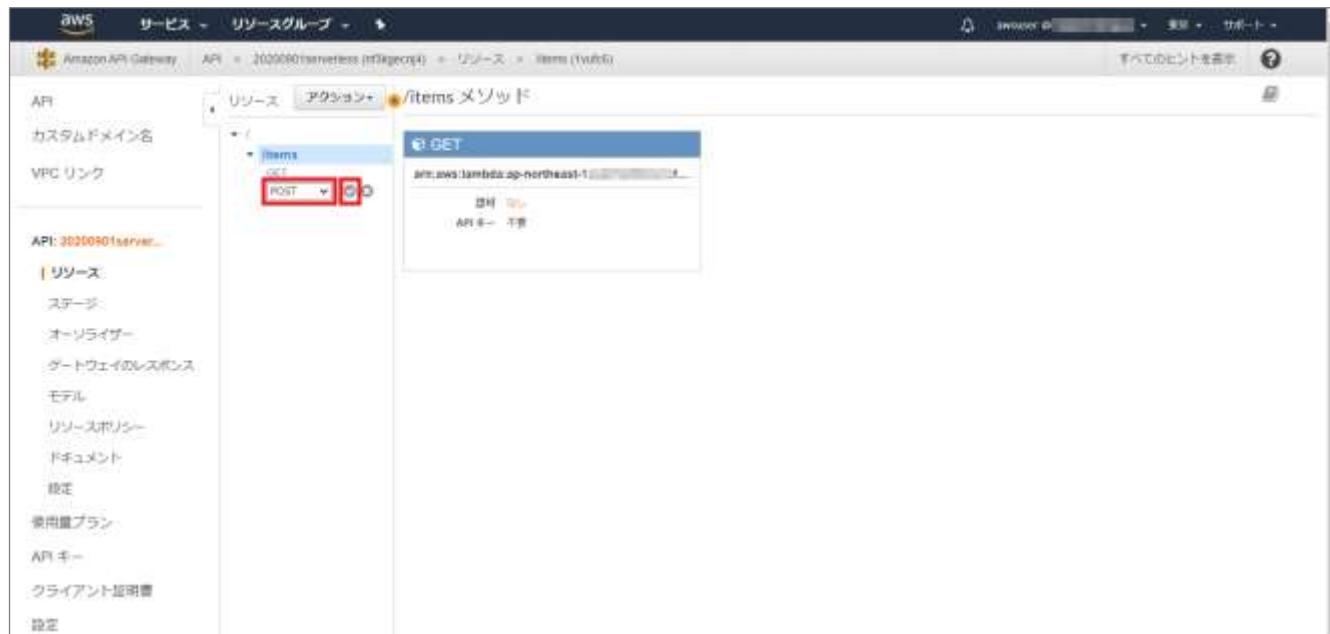
The screenshot shows the AWS Lambda console interface. On the left, there's a sidebar with various navigation options like 'API', 'カスタムドメイン名', 'VPC リンク', etc. The main area displays the API resource tree. A red box highlights the path element '/items'. To the right of the tree, there's a detailed view of the GET method configuration for the 'arn:aws:lambda:ap-northeast-1:...' function, which is currently set to 'APIキー 不要'.

2. [アクション] プルダウンから [メソッドの作成] を選択します。



This screenshot shows the same AWS Lambda console interface as the previous one, but the 'Actions' dropdown menu is open. A red box highlights the 'メソッドの作成' (Create Method) option. Other visible options in the dropdown include 'リソースの作成' (Create Resource), 'CORS の有効化' (Enable CORS), 'リソースドキュメントの構造' (Resource Document Structure), and 'リソースの削除' (Delete Resource). Below the dropdown, the right panel shows the same API configuration details as the first screenshot.

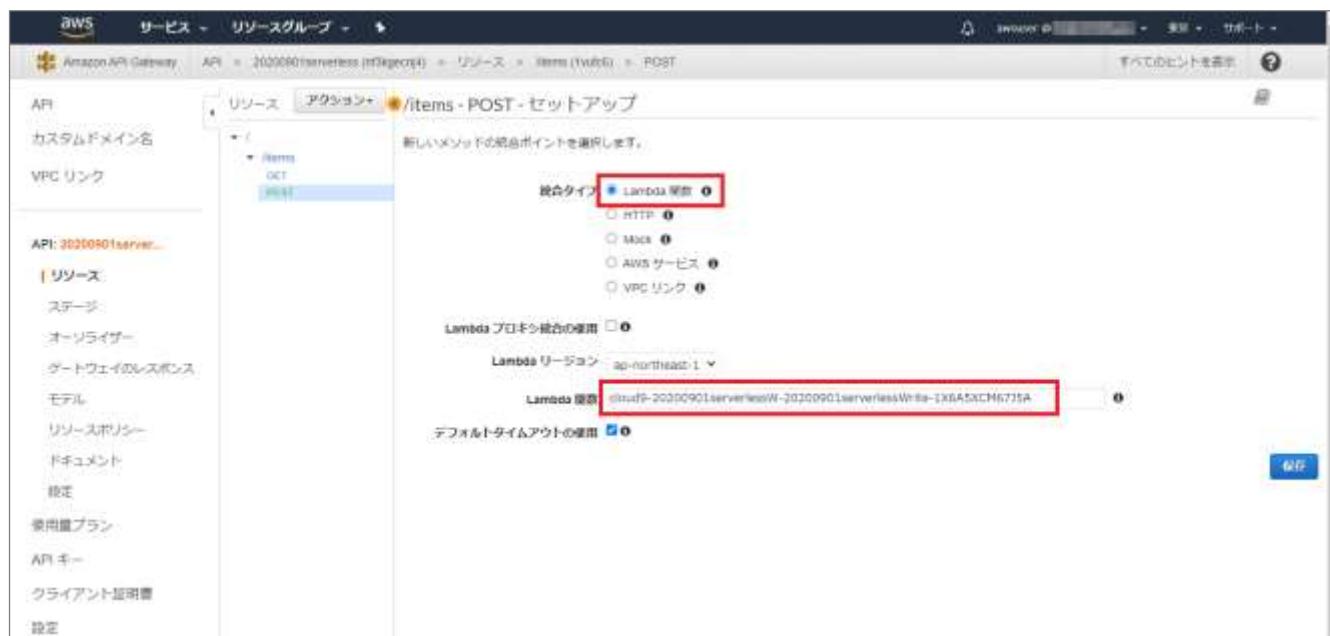
3. メソッド種類のプルダウンから [POST] を選択して、右側のチェックボタンをクリックします。



4. [統合タイプ] はデフォルトの [Lambda 関数] のままにします。

[Lambda プロキシ統合の使用] のチェックは外したままにします。

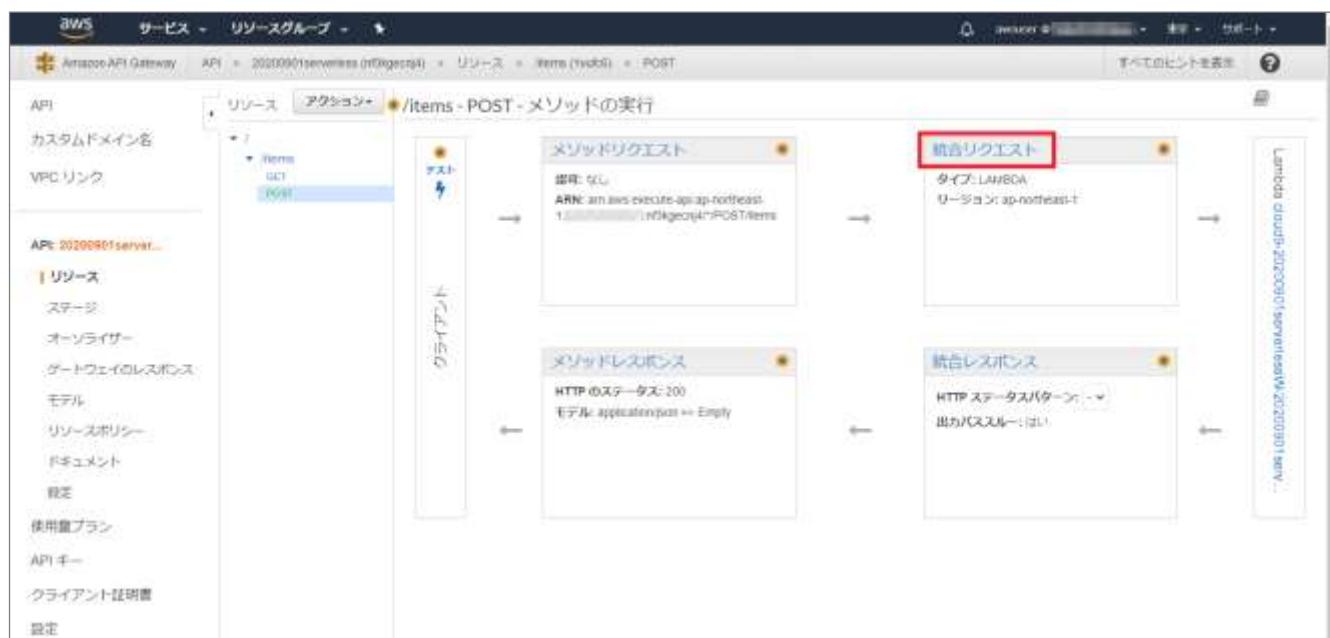
[Lambda 関数] 欄欄に、前節で作成した Lambda 関数のうち「Write」の方の関数名を入力します。



5. [保存]を押すと権限追加の確認ダイアログが表示されますので、[OK] をクリックします。



6. 作成した [POST] メソッドの画面になりますので、[統合リクエスト] をクリックします。



7. 一番下の [マッピングテンプレート] をクリックして展開します。

The screenshot shows the AWS Lambda function configuration in the AWS API Gateway console. The 'Mapping Template' section at the bottom is highlighted with a red box. The configuration includes:

- API: 20200901serverless (Integrate)
- リソース: /items - POST - 統合リクエスト
- 拡張タイプ: Lambda 関数 (selected)
- HTTP リージョン: ap-northeast-1
- 実行ロール: Lambda 角度: cloud9-20200901serverlessW-20200901serverlessWrite-1X6A5XCM67.JSA
- デフォルトタイムアウトの使用: 有効
- マッピングテンプレート: (展開)

8. [テンプレートが定義されていない場合 (推奨)] を選択します。

[マッピングテンプレートの追加] をクリックします。

The screenshot shows the 'Mapping Template' configuration dialog. The 'Template for requests with no mapping template defined (recommended)' radio button is selected and highlighted with a red box. The configuration includes:

- リクエスト本文のパススルー: リクエストの Content-Type ヘッダーに一致するテンプレートがない場合 (selected)
- なし: (未選択)

Content-Type: マッピングテンプレートが定義されていません。リクエスト本文は既定エンドポイントに渡されます。

マッピングテンプレートの追加

9. [Content-Type] 欄に [application/json] と入力して、右側のチェックボタンをクリックします。



10. [apigateway\_post\_mapping.txt] の内容をコピーしてテンプレートの入力欄にペーストします。



11. [保存] をクリックします。 (画面は変わりません)

12. 画面上部の [**←メソッドの実行**] をクリックして、前の画面に戻ります。

The screenshot shows the AWS Lambda function configuration for the POST method of the /items API resource. The 'Lambda関数' (Lambda Function) is selected as the integration type. The Lambda function name is 'lambda-items'. The Lambda function ARN is 'arn:aws:lambda:ap-northeast-1:123456789012:function:lambda-items'. The Lambda function region is 'ap-northeast-1'. The execution role is 'lambda-execute-role'. The timeout is set to 3 seconds. The memory size is 128 MB. The handler is 'lambda-items.handler'. The runtime is Node.js 8.10. The environment variables are: 'AWS\_LAMBDA\_FUNCTION\_NAME': 'lambda-items', 'AWS\_LAMBDA\_FUNCTION\_MEMORY\_SIZE': '128', 'AWS\_LAMBDA\_FUNCTION\_TIMEOUT': '3', 'AWS\_LAMBDA\_FUNCTION\_HANDLER': 'lambda-items.handler', 'AWS\_LAMBDA\_FUNCTION\_REGION': 'ap-northeast-1'. The test event is defined as follows:

```
var item = { id: "1", name: "test" };
```

The 'Test' button is highlighted with a red box.

13. [テスト] をクリックします。

The screenshot shows the AWS Lambda function configuration page after clicking the 'Test' button. The 'Test' button is highlighted with a red box. The test event is displayed in the 'メソッドリクエスト' (Method Request) section. The response is shown in the 'メソッドレスポンス' (Method Response) section. The execution details are shown in the '統合リクエスト' (Integration Request) and '統合レスポンス' (Integration Response) sections. The Lambda function ARN is 'arn:aws:lambda:ap-northeast-1:123456789012:function:lambda-items'.

14. [apigateway\_post\_test.txt] の内容をコピーして [リクエスト本文] 欄にペーストします。

The screenshot shows the AWS API Gateway test interface. On the left, there's a sidebar with various API settings like custom domains, VPC links, and stages. The main area shows a POST method for the '/items' resource. The 'Request Body' field is highlighted with a red box, containing the following JSON:

```
1: {  
2:   "artist": "Beyoncé",  
3:   "title": "like a virgin"  
4: }
```

Below the request body is a blue 'Test' button.

15. [テスト] をクリックすると、右側に結果が表示されます。

[ステータス] が [200] と表示されていれば成功です。

The screenshot shows the AWS API Gateway Test interface. On the left, the API path is set to `/items` and the method is `POST`. The body of the request is set to `{ "Artist": "Madonna", "Title": "Like a Virgin" }`. The response on the right shows a status code of **ステータス: 200**, a latency of **1265 ms**, and a response body containing JSON log output.

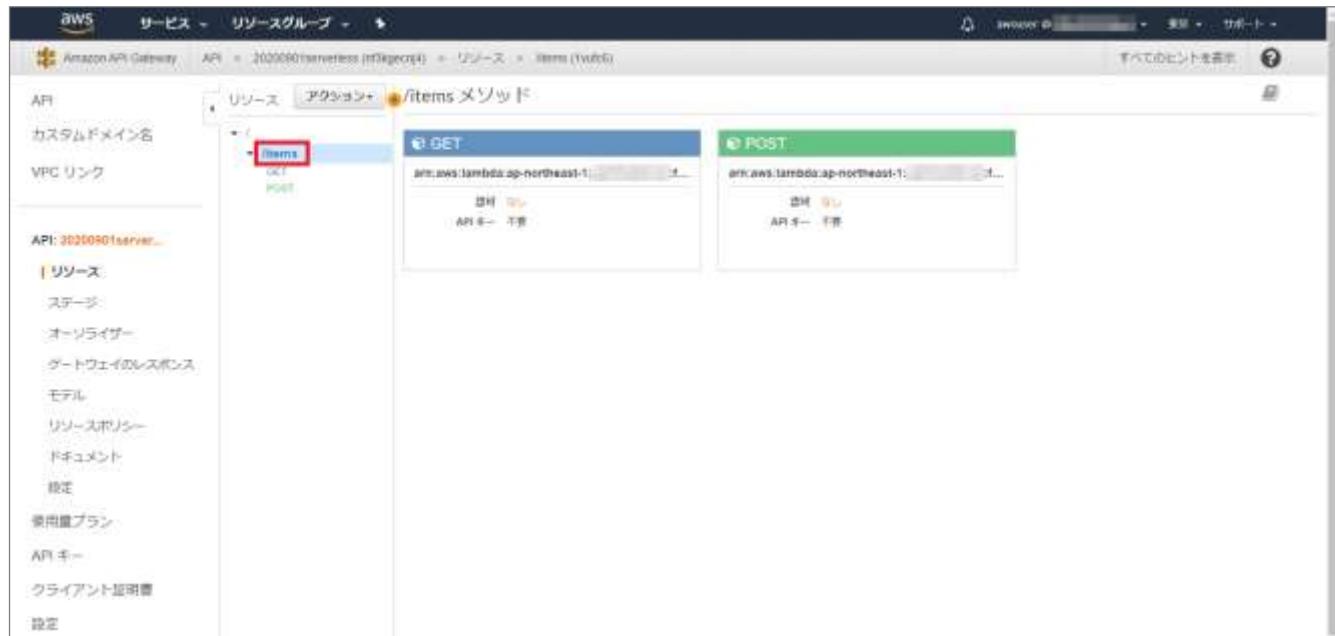
16. DynamoDB の画面に移動して、テーブルの [項目] タブを表示します。

リロードボタンをクリックして、POST メソッドから呼び出された Lambda 関数によって書き込まれたデータが表示されていることを確認します。

The screenshot shows the AWS DynamoDB table `20200901serverless` in the `Items` tab. A new item has been added with the primary key `Artist` set to `Madonna` and the attribute `Title` set to `Like a Virgin`. The table also contains the original item from the database, `Michael Jackson` with `Thriller`.

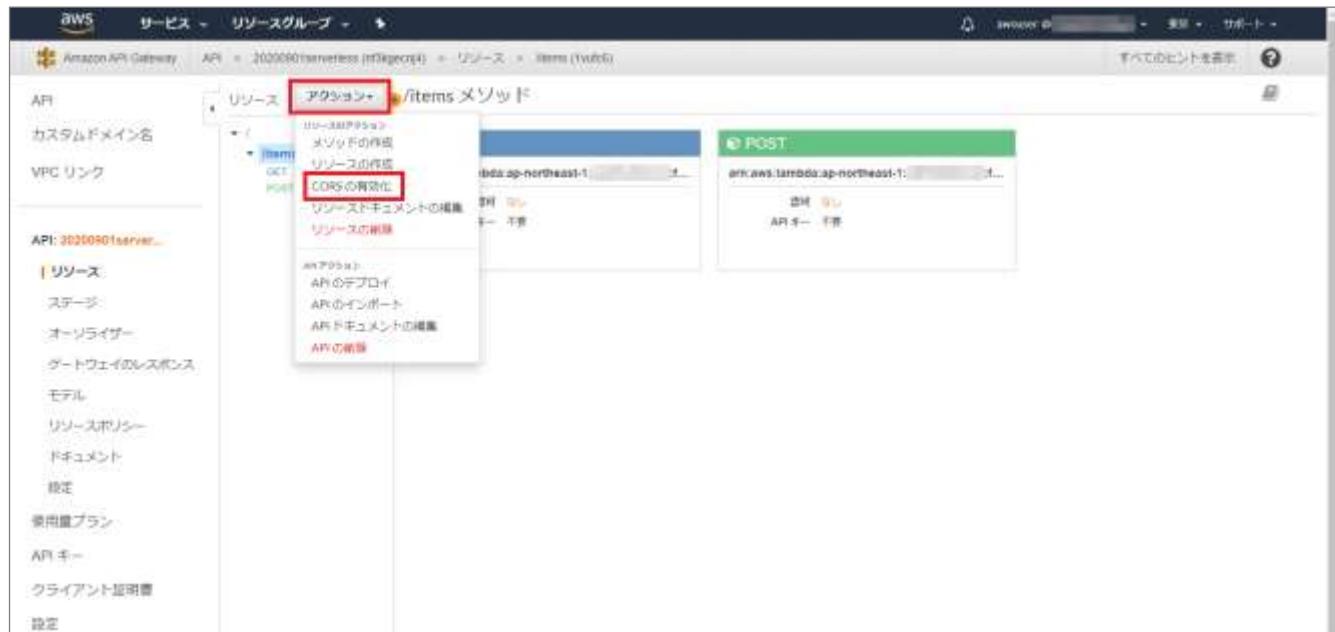
#### 4.2.4. CORS の設定

1. リソースのツリー階層から [/items] をクリックして選択します。



The screenshot shows the AWS API Gateway console. On the left, there's a sidebar with various options like 'カスタムドメイン名' (Custom Domain Name), 'VPC リンク' (VPC Link), and 'API' (API: 20200901server...). The main area shows a tree structure under 'リソース'. A red box highlights the '/items' node, which is selected. To the right of the tree, there are two cards: 'GET' and 'POST', both showing the endpoint 'arn:aws:lambda:ap-northeast-1:...' and a status of '未実装' (Not Implemented).

2. [アクション] プルダウンから [CORS の有効化] を選択します。



This screenshot is similar to the previous one, but the 'Actions' dropdown menu has been opened over the '/items' node. A red box highlights the 'Enable CORS' option in the dropdown menu. The other options listed are 'リソースアタッチ' (Resource Attach), 'リソースの作成' (Create Resource), 'リソースの削除' (Delete Resource), 'API アクション', 'API のデプロイ', 'API のインポート', 'API ドキュメントの編集', and 'API の削除'.

3. [CORS を有効にして既存の CORS ヘッダーを置換] をクリックします。

The screenshot shows the AWS Lambda API Gateway interface. On the left, there's a sidebar with options like 'API', 'カスタムドメイン名', 'VPC リンク', etc. The main area is titled 'CORS の有効化' (Enable CORS). It shows a list of resources: 'Items' (selected), 'GET', and 'POST'. Under 'Items', there are fields for 'Access-Control-Allow-Methods' (set to 'GET, OPTIONS, POST'), 'Access-Control-Allow-Headers' (set to 'Content-Type, X-Amz-Date, Authorization'), and 'Access-Control-Allow-Origin' (set to '\*'). A red box highlights the 'CORS を有効にして既存の CORS ヘッダーを置換' (Enable CORS and replace existing CORS header) button at the bottom right.

4. 確認ダイアログが表示されますので、[はい、既存の値を置き換えます] をクリックします。

The screenshot shows a confirmation dialog box titled 'メソッド変更の確認' (Method Change Confirmation). The text inside the box explains the changes: 'この変更は、このリソースのメソッドに対して行われ、すべての既存の値は置き換えられます。続行してよろしいですか？' (This change will be applied to this resource's methods, replacing all existing values. Do you want to proceed?). Below the text is a list of steps: 'OPTIONS メソッドを作成する', '200 メソッドレスポンスを空のレスポンスモデルとともに OPTIONS メソッドに追加する', 'Mock 論理を OPTIONS メソッドに追加する', '200 総合レスポンスを OPTIONS メソッドに追加する', 'Access-Control-Allow-Methods, Access-Control-Allow-Origin メソッドレスポンスヘッダーを OPTIONS メソッドに追加する', 'Access-Control-Allow-Headers, Access-Control-Allow-Methods, Access-Control-Allow-Origin 総合レスポンスヘッダーマッピングを OPTIONS メソッドに追加する', 'Access-Control-Allow-Origin メソッドレスポンスヘッダーを GET メソッドに追加する', 'Access-Control-Allow-Origin 総合レスポンスヘッダーマッピングを GET メソッドに追加する', 'Access-Control-Allow-Origin メソッドレスポンスヘッダーを POST メソッドに追加する', and 'Access-Control-Allow-Origin 総合レスポンスヘッダーマッピングを POST メソッドに追加する'. At the bottom of the dialog, there are two buttons: 'キャンセル' (Cancel) and 'はい、既存の値を置き換えます' (Yes, replace the existing value), with the latter being highlighted by a red box.

5. CORS の有効化処理が実行されて、すべての結果に緑のチェックが付くことを確認します。

The screenshot shows the AWS Lambda function configuration page. The left sidebar lists various settings like API, VPC, and Resource Groups. The main area shows the function's code and configuration. A red box highlights the 'Actions' tab, which contains a list of successful steps under the heading 'CORS の有効化'. The steps include: 'OPTIONS メソッドを作成する', '200 メソッドレスポンスを割りレスポンスマルトとともに OPTIONS メソッドに追加する', 'Mock 組合せ OPTIONS メソッドに追加する', '200 組合せレスポンスを OPTIONS メソッドに追加する', 'Access-Control-Allow-Headers, Access-Control-Allow-Methods, Access-Control-Allow-Origin メソッドレスポンスヘッダーを OPTIONS メソッドに追加する', 'Access-Control-Allow-Headers, Access-Control-Allow-Methods, Access-Control-Allow-Origin 組合せレスポンスヘッダーマッピングを OPTIONS メソッドに追加する', 'Access-Control-Allow-Origin メソッドレスポンスヘッダーを GET メソッドに追加する', 'Access-Control-Allow-Origin 組合せレスポンスヘッダーマッピングを GET メソッドに追加する', 'Access-Control-Allow-Origin メソッドレスポンスヘッダーを POST メソッドに追加する', and 'Access-Control-Allow-Origin 組合せレスポンスヘッダーマッピングを POST メソッドに追加する'. Below the list, a note says 'リリースは CORS に対して設定されました。上位の出力にエラーが表示される場合は、エラーメッセージを確認し、必要に応じて実行したステップをメソッドエディターを通して手動で実行してみてください。'.

6. 画面左側のメニューから [ゲートウェイのレスポンス] をクリックします。

The screenshot shows the AWS Lambda function configuration page. The left sidebar lists various settings like API, VPC, and Resource Groups. The main area shows the function's code and configuration. A red box highlights the 'Gateway Responses' section in the left sidebar. The right panel displays the 'Gateway Responses' configuration, which includes a list of response types and a note: 'ゲートウェイのレスポンスを選択してください'.

7. 一番上の【アクセスが拒否されました】を選択して、画面右上の【編集】をクリックします。

The screenshot shows the AWS Lambda function configuration page. On the left, there's a sidebar with various settings like 'API', 'カスタムドメイン名', 'VPC リンク', etc. The main area is titled 'Gateway Responses' and contains a section for 'ゲートウェイのレスポンス'. A radio button labeled 'アクセスが拒否されました' is selected and highlighted with a red box. To the right, there's a 'Edit' button at the top right of the response configuration panel.

8. 【レスポンスヘッダーの追加】をクリックします。

The screenshot shows the same AWS Lambda function configuration page as the previous one, but with a different focus. The 'Edit' button from the previous step has been clicked, and now the 'Edit' button in the 'レスポンスヘッダー' section is highlighted with a red box. This section is used to add custom headers to the API response.

9. 以下の通り入力します。

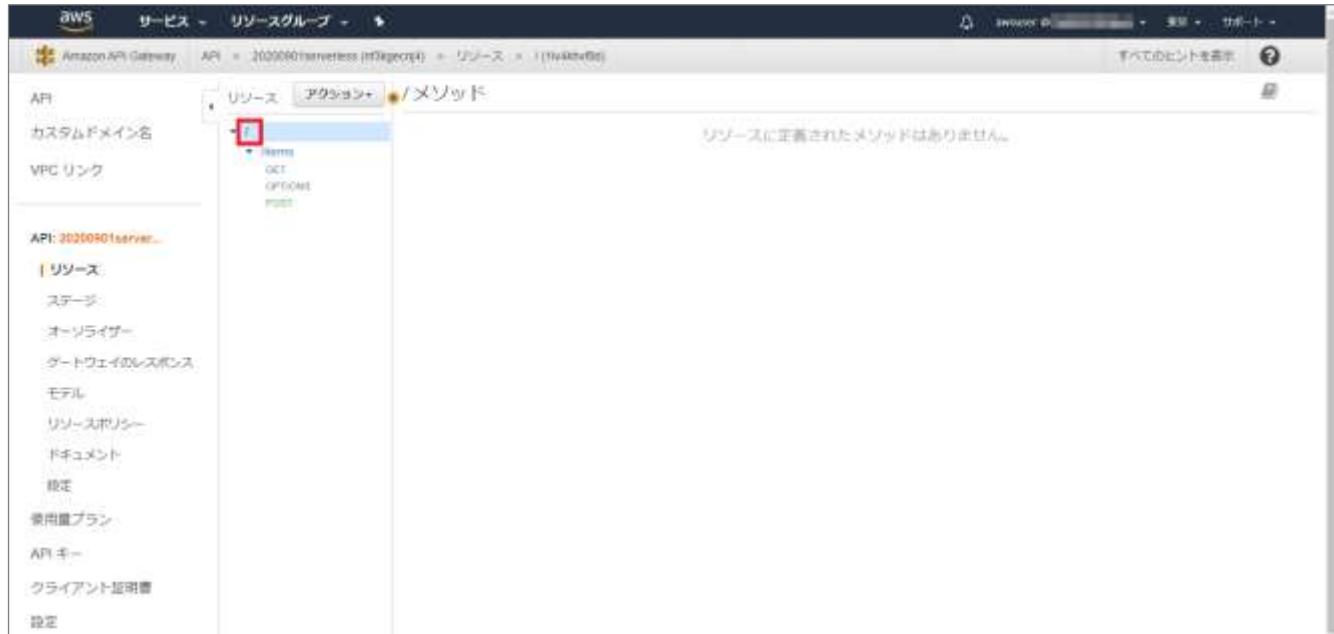
- レスポンスヘッダー: [Access-Control-Allow-Origin] と入力
- 値: ['\*'] と入力 (シングルクォーテーション、アスタリスク、シングルクォーテーション)

The screenshot shows the AWS Lambda function configuration interface. On the left, there's a sidebar with various tabs like 'Lambda Functions', 'Logs', 'Metrics', 'Actions', 'Tracing', 'CloudWatch Metrics', 'CloudWatch Logs', and 'CloudWatch Metrics Insights'. The main area has tabs for 'Overview', 'Code', 'Environment', 'Runtime Configuration', 'Tracing', and 'Logs'. Under 'Logs', it says 'No logs have been generated yet.' and 'Logs are off by default. Turn them on to view logs for this function.' There are 'Edit' and 'Save' buttons at the bottom.

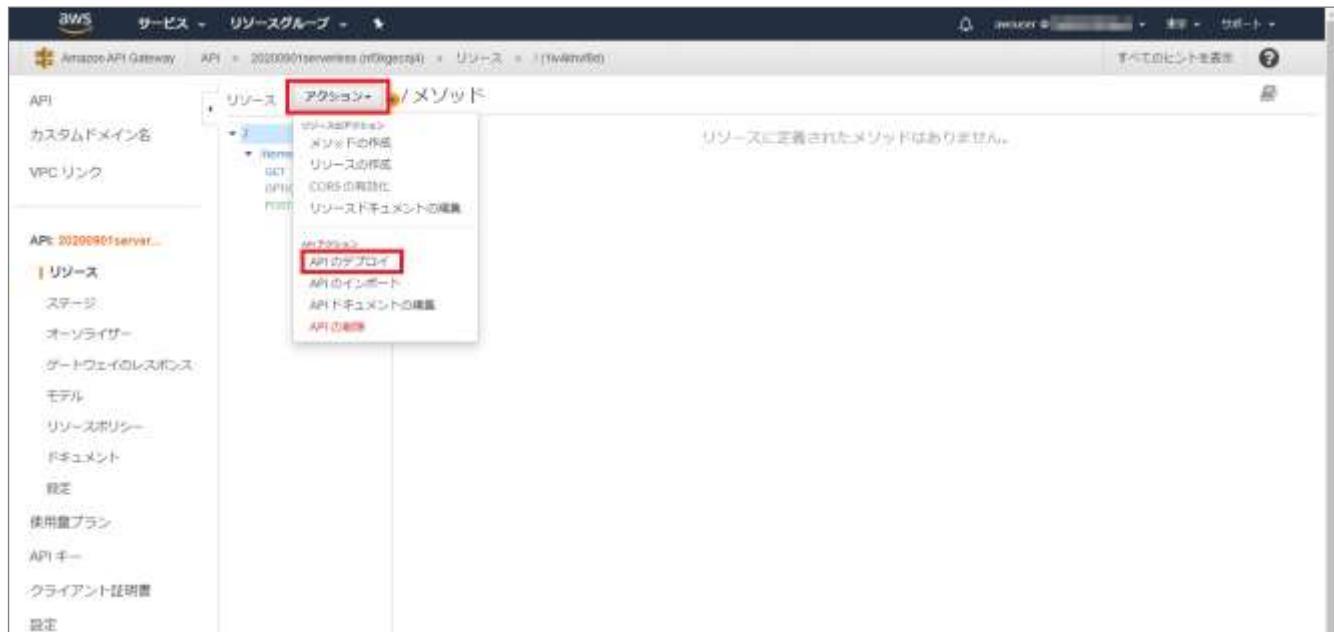
10. [保存] をクリックします。

#### 4.2.5. API のデプロイ

1. リソースのツリー階層から [/] をクリックして選択します。



2. [アクション] プルダウンから [API のデプロイ] を選択します。



3. 以下の通り入力します。

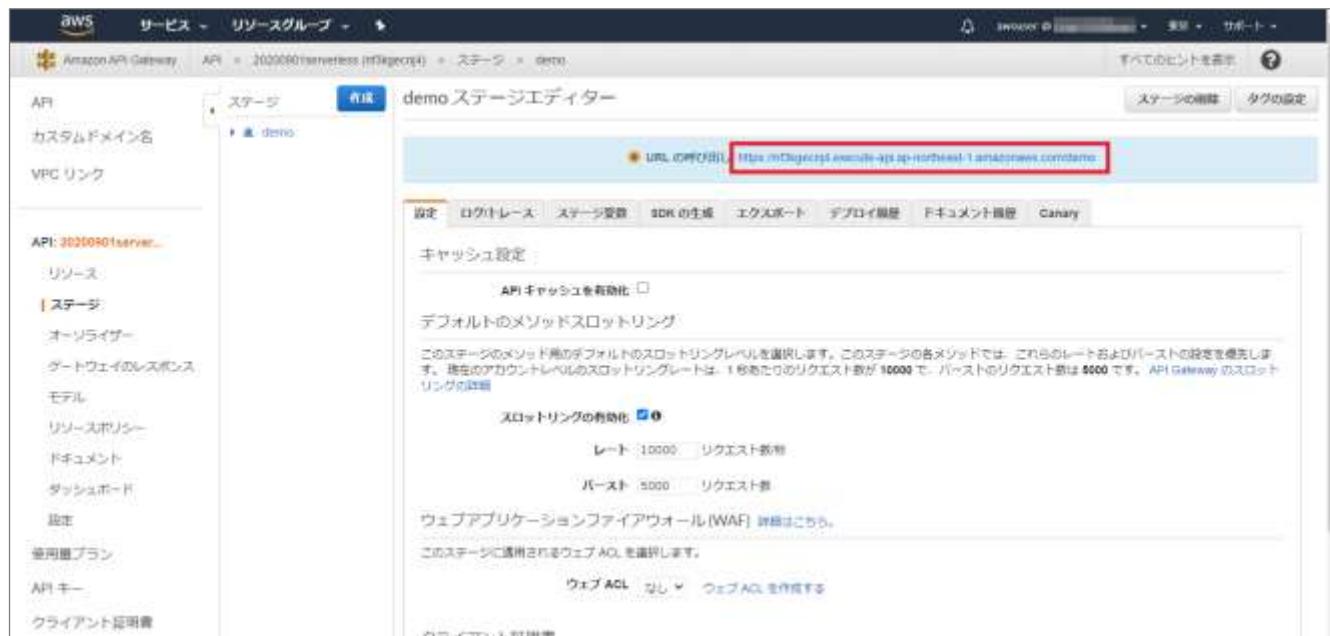
- **デプロイされるステージ:** [新しいステージ] を選択
- **ステージ名:** [demo] と入力
- **ステージの説明:** (省略)
- **デプロイメントの説明:** (省略)



4. [デプロイ] をクリックします。

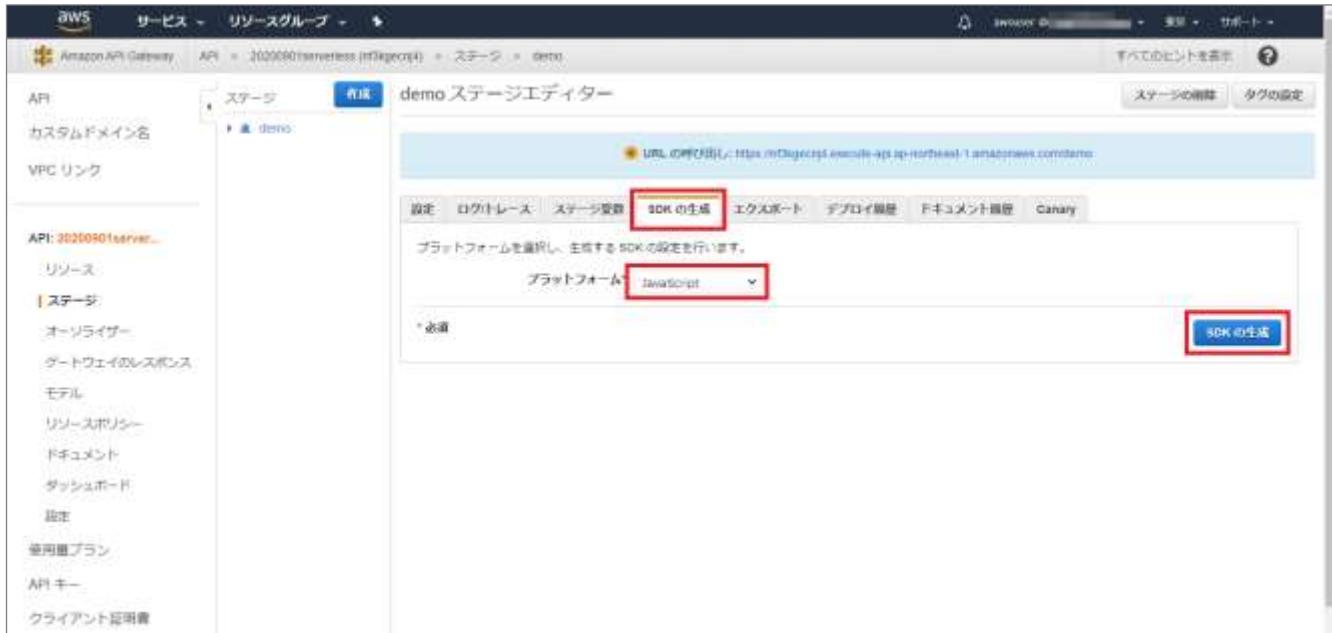
5. デプロイが行われ、ステージ [demo] の画面が表示されます。

画面上部には API を呼び出す際の URL が表示されています。（後の手順で使用するためメモしておいてください）



6. [SDK の生成] タブをクリックします。

[プラットフォーム] で [JavaScript] を選択して、[SDK の生成] をクリックします。



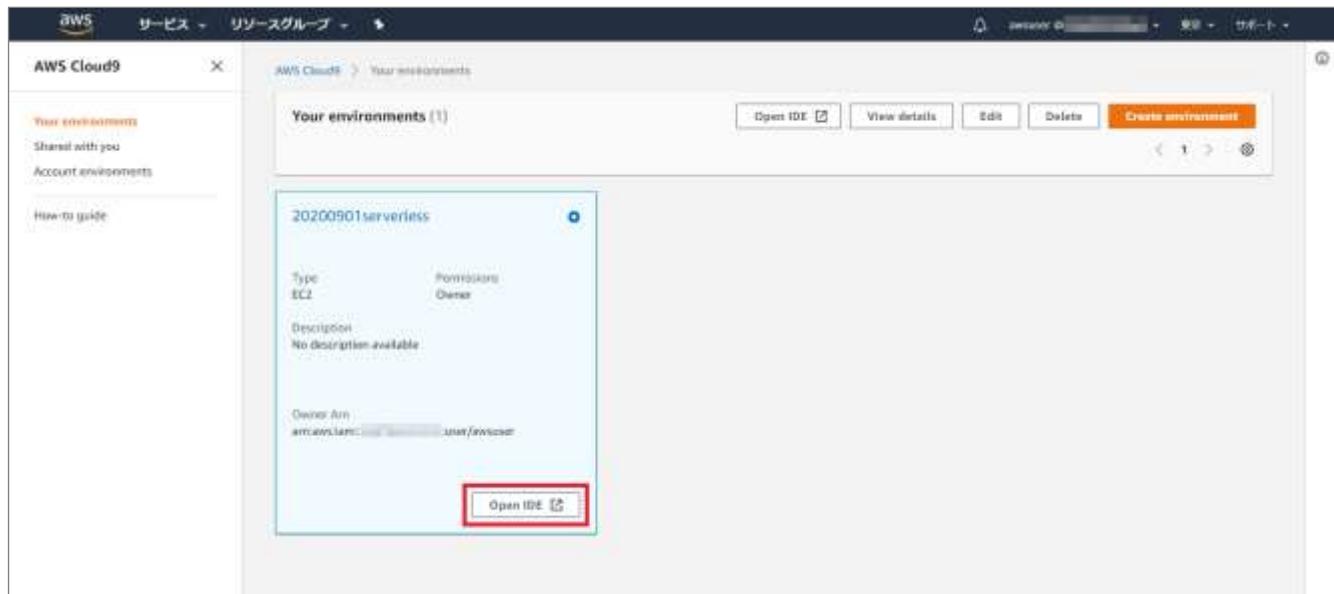
7. ファイル **[javascript\_YYYY-MM-DD\_HH-MMZ.zip]** のダウンロードが行われるので、任意の場所に保存します。 (後の手順で使用します)

#### 4.2.6. Cloud9 からの動作確認

1. Cloud9 の画面へ移動します。

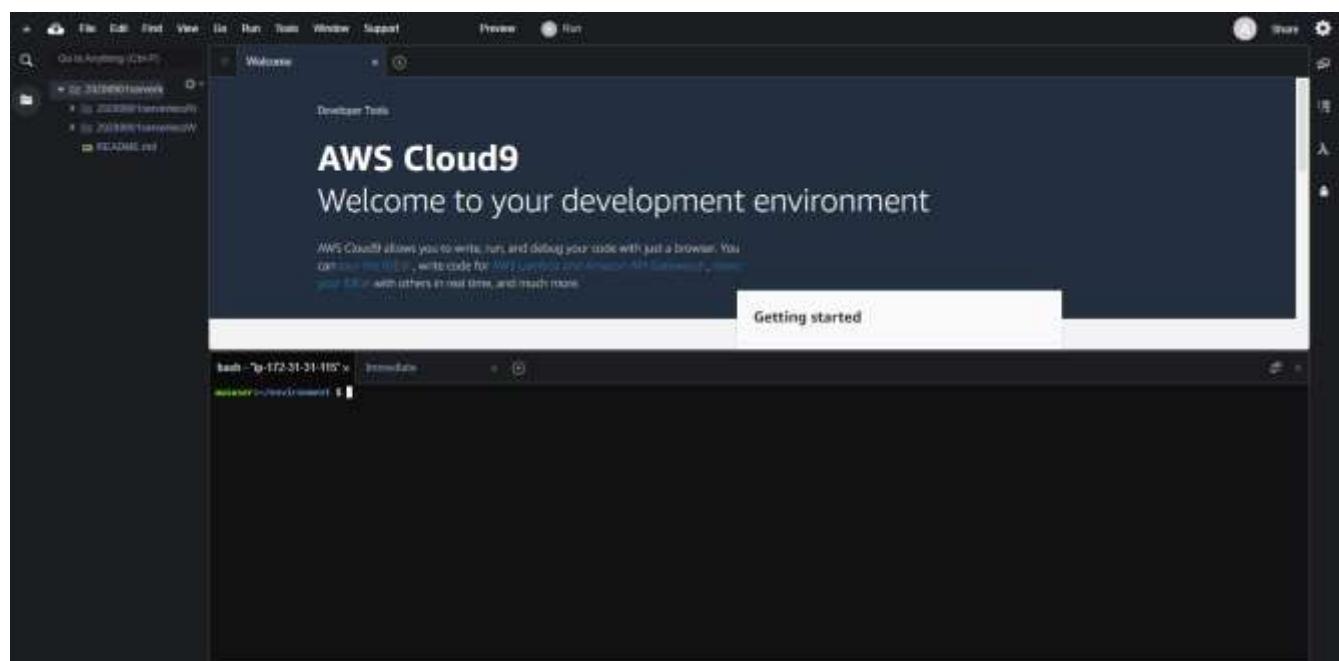
時間が経っている場合には Cloud9 の EC2 インスタンスが停止している可能性があります。

その際は、Web ブラウザをリロードするか、一旦画面を閉じて Cloud9 の管理画面を開き、対象の Cloud9 環境を指定して **[Open IDE]** をクリックしてください。



2. Cloud9 の画面下部にある **[Terminal]** ウィンドウを使用してコマンドを実行します。

画面が狭い場合は、境界線をドラッグして画面を大きくして頂いて構いません。



3. ここからの一連のコマンド実行内容は、**[apigateway\_curl\_test.txt]** にも記載していますので参考にしてください。

まず、前節で確認した **[API呼び出し URL]** を環境変数にセットします。 (URL はご自身の環境に合わせて入力してください)

```
$ export API_URL=https://XXXXXXXXXX.execute-api.AWS_REGION.amazonaws.com/demo
```

4. GET メソッドの動作を確認します。

以下のようにコマンドを入力します。

```
$ curl -X GET -i ${API_URL}/items?artist=Michael+Jackson
```

5. API 呼び出しに成功すると、以下のように結果が返ってくるはずです。

```
$ curl -X GET -i ${API_URL}/items?artist=Michael+Jackson
HTTP/2 200
date: Tue, 1 Sep 2020 05:15:01 GMT
content-type: application/json
content-length: 49
x-amzn-requestid: 4c75f8ea-fc5b-47f7-a323-799bc98d7664
access-control-allow-origin: *
x-amz-apigw-id: Sr4pIEKSNjMFqIg=
x-amzn-trace-id: Root=1-5f5b07d3-5154bf8e204e3de2231150b5;Sampled=0

[{"Title": "Thriller", "Artist": "Michael Jackson"}]
```

## 6. POST メソッドの動作を確認します。

以下のようにコマンドを入力します。

```
$ curl -X POST -i \
-H 'Content-Type: application/json' \
-d '{"artist":"Michael Jackson","title":"Billie Jean"}' \
${API_URL}/items
```

## 7. API 呼び出しに成功すると、以下のように結果が返ってくるはずです。

```
$ curl -X POST -i \
>   -H 'Content-Type: application/json' \
>   -d '{"artist":"Michael Jackson","title":"Billie Jean"}' \
>   ${API_URL}/items
HTTP/2 200
date: Tue, 1 Sep 2020 05:17:42 GMT
content-type: application/json
content-length: 2
x-amzn-requestid: 0eaf5dac-0ae8-4790-9556-5d314083b5be
access-control-allow-origin: *
x-amz-apigw-id: Sr5CXH0jtjMFuTQ=
x-amzn-trace-id: Root=1-5f5b0875-0a3206c732fed15649481e99;Sampled=0
{}

{}
```

## 8. DynamoDB の画面に移動して、テーブルの【項目】タブを表示します。

リロードボタンをクリックして、POST メソッドから呼び出された Lambda 関数によって書き込まれたデータが表示されていることを確認します。

The screenshot shows the AWS DynamoDB console with the '20200901serverless' table selected. The 'Items' tab is active, displaying the following data:

Artist	Title
Michael Jackson	Billie Jean
Michael Jackson	Thriller
Marlene	Like a Virgin

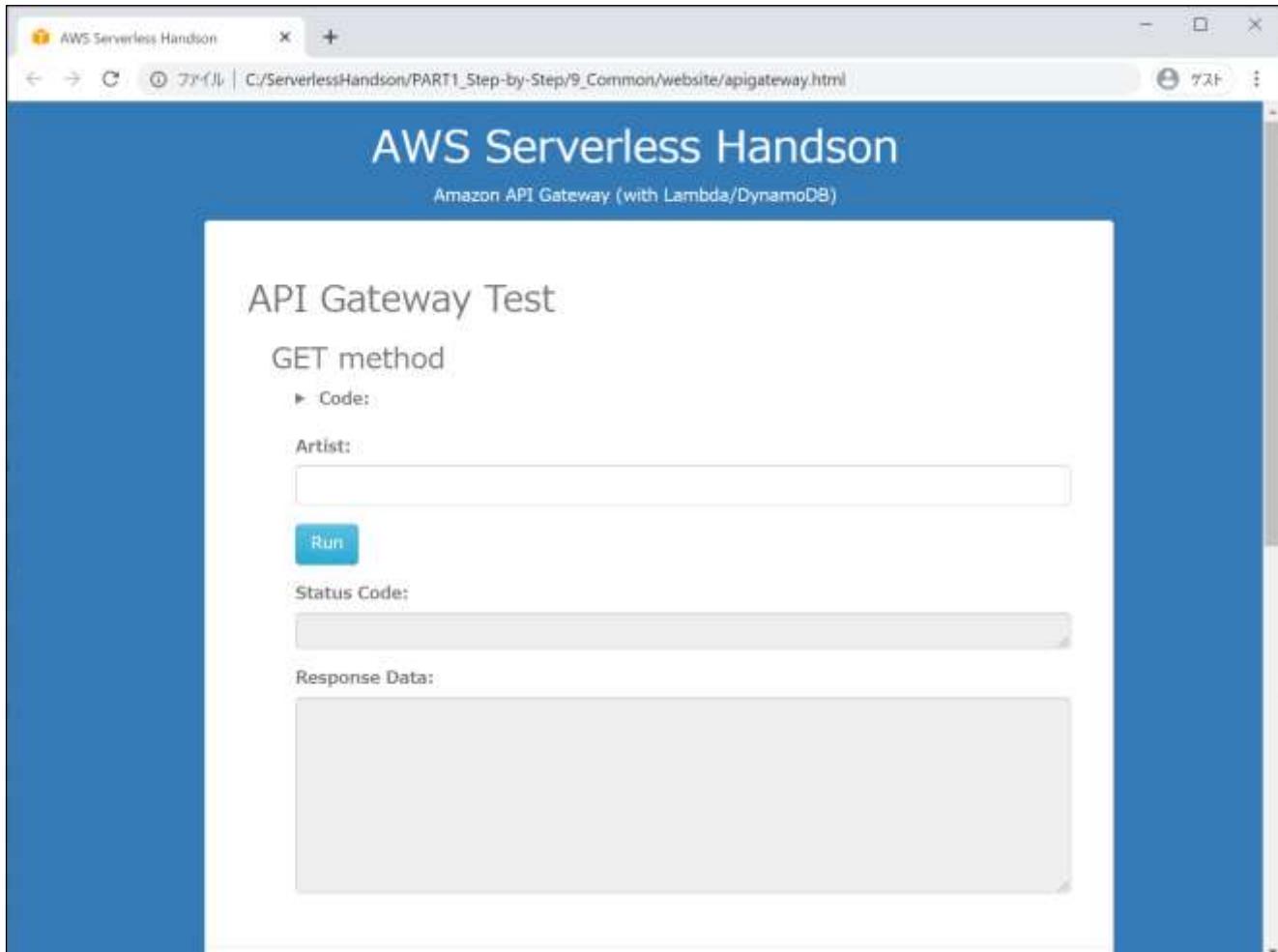
#### 4.2.7. Web ブラウザからの動作確認

1. [webpage] フォルダに以下のファイルを[427]のフォルダからコピーします。
  - [apigateway.html]
  - [apigateway.js]
2. 「4.2.5. API のデプロイ」でダウンロードした [javascript\_YYYY-MM-DD\_HH-MMZ.zip] を zip 展開します。  
展開して出来たフォルダ [apiGateway-js-sdk] の直下に [apigClient.js] というファイルがあります。  
このファイルを [webpage]-[lib]-[apigateway] フォルダの直下にコピーします。
3. この時点で、[webpage] フォルダ配下の構成は以下のようになっているはずです。

```
[website]
  |- [img] ... アイコン等の画像ファ
  |- [lib] ... 各種ライブラリ/SDK (必要なものを格納済み、ただし apigClient.js を除く)
    |- [apigateway]
      |- [lib]
        |- apigClient.js ... API Gateway アクセスクライアントのクラス定義
      |- [awssdk]
      |- [cognito]
  |- [style] ... CSS ファイル
  |- apigateway.html ... Web ブラウザで表示するページ
  |- apigateway.js ... API Gateway へアクセスを行う処理を記述した JavaScript コード
```

その他のファイルは存在していても邪魔にはならないので無視します。 (後ほど使います。)

4. [apigateway.html] を Web ブラウザで開きます。



5. まず、GET メソッドを試します。

[Artist] 欄に検索する文字列を入力して、[Run] をクリックします。

API Gateway Test

GET method

▶ Code:

Artist:  
Michael Jackson

Run

Status Code:

Response Data:

6. [Status Code] 欄に [200] と表示されることを確認します。

[Response Data] 欄に GET メソッドの結果が表示されていることを確認します。

API Gateway Test

GET method

▶ Code:

Artist:  
Michael Jackson

Run

Status Code:  
200

Response Data:

```
[{"Title": "Billie Jean", "Artist": "Michael Jackson"}, {"Title": "Thriller", "Artist": "Michael Jackson"}]
```

7. 続いて POST メソッドを試します。

[Artist] 欄および [Title] 欄に登録するデータを入力して、[Run] をクリックします。

POST method

▶ Code:

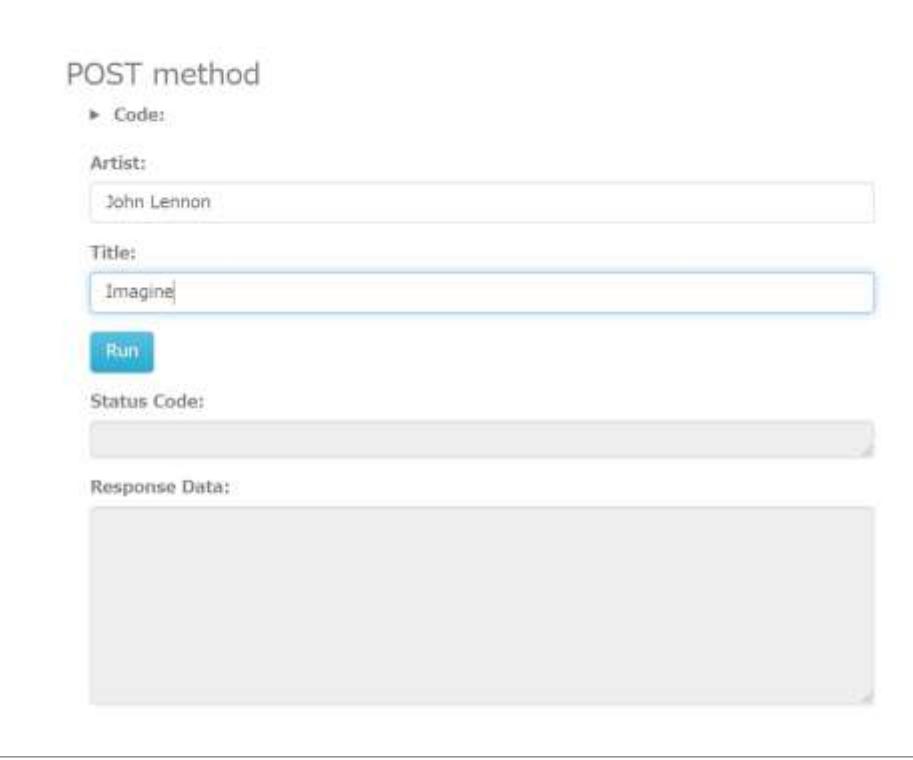
Artist:  
John Lennon

Title:  
Imagine

**Run**

Status Code:

Response Data:



8. [Status Code] 欄に [200] と表示されることを確認します。

DynamoDB の画面から、登録したデータが正しく書き込まれていることを確認します。

POST method

▶ Code:

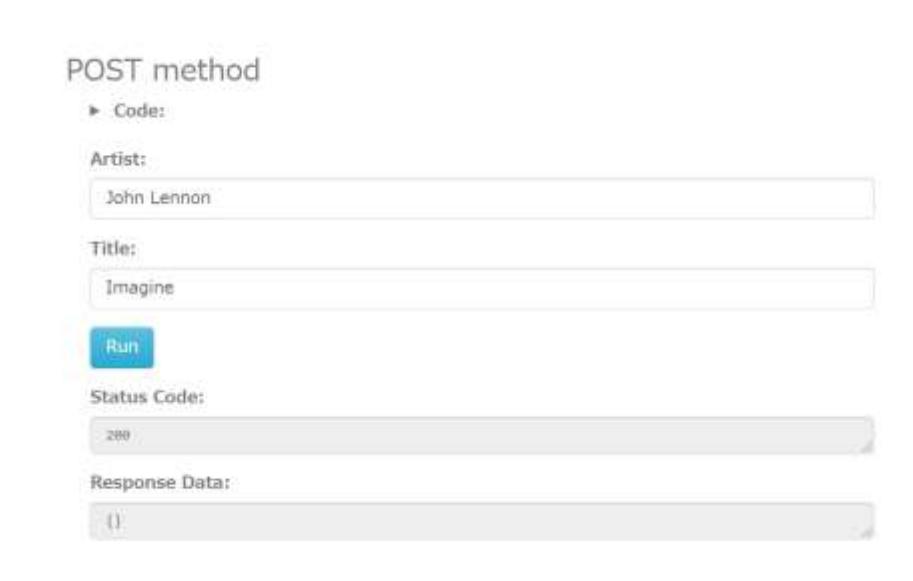
Artist:  
John Lennon

Title:  
Imagine

**Run**

Status Code:  
200

Response Data:  
()



## 4.3. Cognito の動作確認を行う

### 4.3.1. Cognito ユーザープールの作成

1. AWS マネジメントコンソールのサービス一覧から **[Cognito]** を選択します。

**[ユーザープールの管理]** をクリックします。

The screenshot shows the AWS Management Console with the Cognito service selected. The main heading is "Amazon Cognito". Below it, a text block explains that Cognito provides user pools and identity pools. A red box highlights the "User Pools" tab. Two sections are shown below: "Sign Up and Sign In" (with an icon of two people and a plus sign) and "Grant access to AWS services" (with an icon of a computer monitor and a lock). Each section has a descriptive text block.

2. **[ユーザープールを作成する]** をクリックします。

The screenshot shows the "Create User Pool" wizard. The top navigation bar includes "AWS", "サービス", "リソースグループ", and "User Pools". The main area displays the message "User pool does not exist. Click here to create a user pool." A red box highlights the "Create User Pool" button at the bottom right.

3. [プール名] 欄に [YYYYMMDDserverless] と入力します。 (YYYYMMDD は本日の日付)  
[デフォルトを確認する] をクリックします。

The screenshot shows the first step of the AWS User Pool creation wizard. The title is "User Poolを作成する". On the left, there's a sidebar with options: 名前 (highlighted), 属性, ポリシー, MFAとして確認, メッセージのカスタマイズ, タグ, デバイス, アプリクライアント, トリガー, 和訳. The main area has two sections: "User Poolにどのような名前を付けますか?" and "User Poolをどのように作成しますか?". In the first section, the "Pool Name" input field contains "20200901serverless" and is highlighted with a red box. In the second section, there are two buttons: "デフォルトを確認する" (highlighted with a red box) and "ステップに進む" (with a smaller red box).

4. 画面左側のメニューから [ポリシー] を選択します。

The screenshot shows the second step of the AWS User Pool creation wizard. The title is "User Poolを作成する". The left sidebar shows "名前" (highlighted), "属性", "ポリシー" (highlighted with a red box), "MFAとして確認", "メッセージのカスタマイズ", "タグ", "デバイス", "アプリクライアント", "トリガー", and "和訳". The main area has several sections: "Pool Name: 20200901serverless", "必須の属性: email", "エイリアス属性: ユーザー名属性の確認", "ユーザー名属性: ユーザー名属性の確認", "大文字と小文字を区別しないことを有効にしますか?: はい", "カスタム属性: カスタム属性の確認", "パスワードの最小数: 8", "パスワードポリシー: 大文字, 小文字, 特殊文字, 数字", "ユーザーのサインアップを許可しますか?: ユーザーは自己サインアップできます", "送信元 Eメールアドレス: デフォルト", "Amazon SESによる Eメール配信: はい", "MFA: MFAが有効化", "確認: Eメール", and "タグ: ユーザープールのタグを選択".

5. [数字を必要とする]、[特殊文字を必要とする]、[大文字を必要とする]、[小文字を必要とする]の各項目のチェックを全て外します。（これはテスト目的です。運用環境では強固なポリシーを設定してください）

画面下部の【変更の保存】をクリックします。

The screenshot shows the 'Create User Pool' configuration screen. On the left, a sidebar lists options: 'Name', 'Attributes', 'Policy', 'MFA and Security', 'Message Customization', 'Tags', 'Delegation', 'Application Client', 'Trigger', and 'Regions'. The 'Policy' section is highlighted with a yellow box. In the main area, there are three sections:

- Password Strength Requirements:** A box highlights the dropdown set to '8' and the checkbox for 'Requires numbers'.
- Allow User Sign-in:** A box highlights the radio button for 'Allow users to sign in with their own accounts'.
- Temporary Password Expiry:** A box highlights the dropdown set to '7'.

At the bottom right, a blue-bordered button labeled 'Change' is highlighted with a red box.

6. 画面左側のメニューから【アプリクライアント】を選択します。

The screenshot shows the 'Create User Pool' configuration screen. The 'Application Client' section is highlighted with a red box. It contains the following fields:

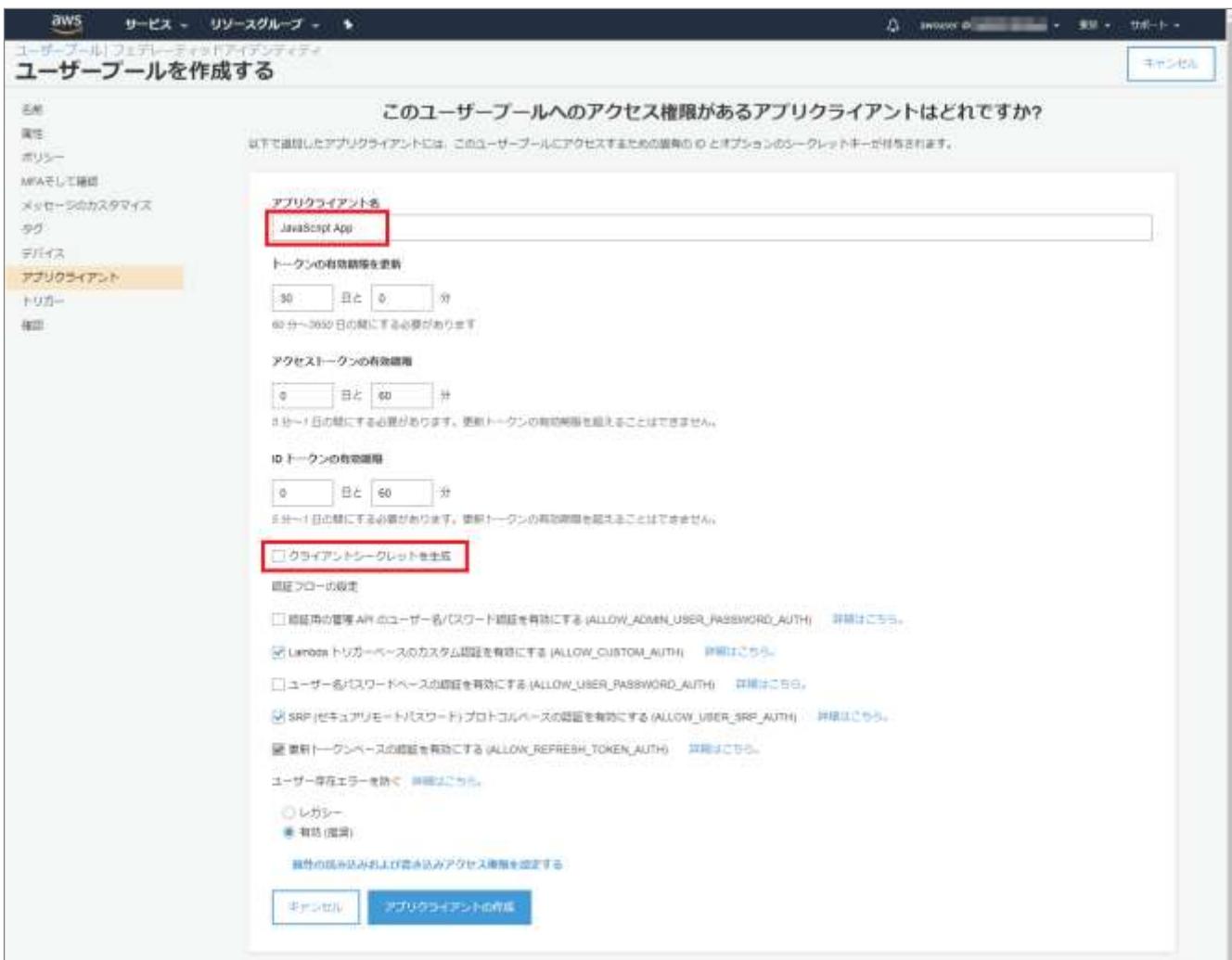
- Pool Name:** 20200801serverless.
- Attribute Mappings:** Shows '必番の属性: email' and 'エイリアス属性: エイリアス属性の選択'.
- User Name Attribute:** 'ユーザー名属性: ユーザー名属性の選択'.
- Character Case:** '大文字と小文字を区別しないことを有効に': 'オフ' (Off).
- Custom Attribute:** 'カスタム属性: カスタム属性の選択'.
- Password Policy:** '最小長: 8' and 'ポリシーなし'.
- User Sign-in:** 'ユーザーのサインアップを許可しますか?: ユーザーは自己サインアップできます'.
- Delivery Options:** '送信先 Eメールアドレス: デフォルト' and 'Amazon SESによる Eメール配信: (はい)'.
- MFA:** 'MFA の有効化'.
- Auth:** 'Eメール'.
- Tags:** 'タグ: ユーザーポールのタグを追加'.

7. [アプリケーションの追加] をクリックします。



8. [アプリケーション名] 欄に [JavaScript App] と入力します。

[クライアントシークレットを生成] のチェックを外します。

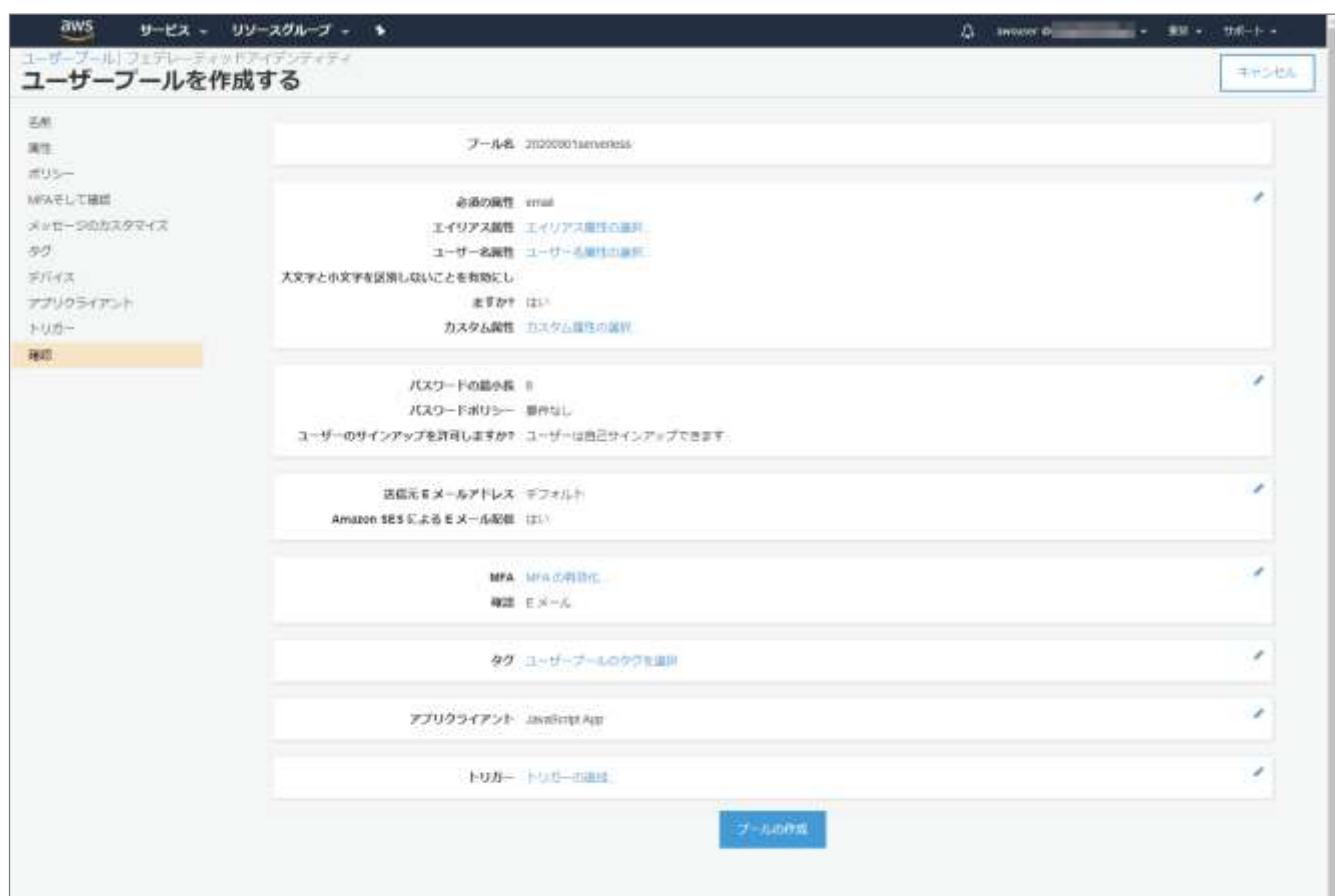


9. [アプリケーションの作成] をクリックします。

10. 画面左側のメニューから [確認] を選択します。



11. 確認画面になりますので、[ポールの作成] をクリックします。



## 12. ユーザープールが作成されました。

ユーザープールの [プール ID] が表示されています。 (後の手順で使用するためメモしておいてください)

The screenshot shows the AWS Cognito User Pool creation page. The main content area displays a green success message: "ユーザープールは正常に作成されました。". Below this, the "Pool ID" is highlighted with a red box, showing "ap-northeast-1\_9OcRWhos". The "Pool ARN" is also visible as "arn:aws:cognito-idp:ap-northeast-1:...". On the left sidebar, the "アプリの認証" section is selected, and under it, the "アプリクライアント" tab is also selected. Other tabs like "ユーザーとグループ", "属性", "ポリシー", etc., are visible but not selected.

## 13. 画面左側のメニューから [アプリクライアント] を選択します。

[アプリクライアント ID] が表示されています。 (こちらも後の手順で使用するためメモしてください)

The screenshot shows the AWS Cognito User Pool configuration page. The main content area asks "このユーザープールへのアクセス権限があるアプリクライアントはどれですか?". Below this, a table lists an application named "JavaScript App" with its "App Client ID" highlighted with a red box, showing "Set2cuc00iv1mpedjdgf". The "Add another client" button is visible at the bottom of the table. On the left sidebar, the "アプリの認証" section is selected, and under it, the "アプリクライアント" tab is selected. Other tabs like "ユーザーとグループ", "属性", "ポリシー", etc., are visible but not selected.

### 4.3.2. Cognito ID プールの作成

1. [Cognito] 画面で、[ID プールの管理] をクリックします。

The screenshot shows the AWS Cognito service management interface. At the top, there's a navigation bar with 'aws' logo, 'サービス', 'リソースグループ', and other options. Below the navigation is the Cognito logo and the heading 'Amazon Cognito'. A descriptive text explains that Cognito provides user pools and ID pools for sign-up/sign-in and identity federation. Two main buttons are at the bottom: 'ユーザー プールの管理' (User Pool Management) and 'ID プールの管理' (ID Pool Management), with the latter being highlighted by a red box. Below these buttons are two sections: 'サインアップとサインインの追加' (Add Sign-up and Sign-in) and 'ユーザーに AWS のサービスへのアクセス権を付与' (Grant access to AWS services to users). Each section has a small icon and some explanatory text.

14. [ID プール名] 欄に [**YYYYMMDDserverless**] と入力します。 (YYYYMMDD は本日の日付)

[認証されていない ID に対してアクセスを有効にする] にチェックを入れます。

This screenshot shows the 'Create New ID Pool' wizard, Step 1: Create ID Pool. The title is '新しいID プールの作成' (Create New ID Pool). It asks for the ID pool name, which is '20200901serverless' with a red box around it. Below is a note about unauthenticated identities. A checkbox labeled '認証されていない ID に対してアクセスを有効にする' (Enable access for unauthenticated identities) is checked, also with a red box around it. A note below explains that this allows users to log in without a provider. The next section is '認証フローの設定' (Authentication Flow Settings), which includes a note about using OAuth 2.0 for mobile clients. At the bottom are '次へ' (Next) and '一括選択' (Select All) buttons.

## 15. [認証プロバイダー] をクリックして展開します。

The screenshot shows the 'AWS' navigation bar with 'サービス', 'リソースグループ', and a search bar. Below it is the '使用開始ウィザード' (Getting Started Wizard) header. The main content area is titled '新しいID プールの作成' (Create New IAM Pool). It shows the first step: 'ID プールを作成する' (Create an IAM pool). A dropdown menu labeled '認証プロバイダー' (Authentication Provider) is open, with 'Cognito' selected. Other options like 'Amazon' and 'Apple' are also visible.

## 16. [Cognito] タブが選択されていることを確認します。（これは「Cognito ユーザープールを認証プロバイダーとして利用する」ということを意味します）

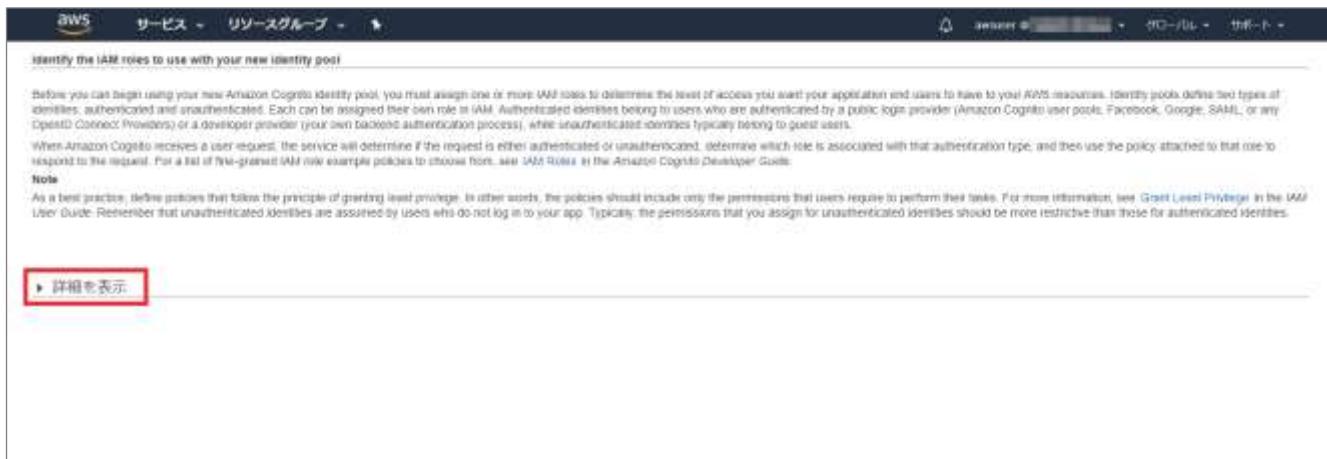
[ユーザープール ID] および [アプリクライアント ID] 欄に、前手順で確認したユーザープールの各 ID を入力します。

The screenshot shows the '認証プロバイダー' (Authentication Provider) section. The 'Cognito' tab is selected, indicated by a red border. Below it, there's a note about using Cognito as a public provider. Two input fields are present: 'ユーザープール ID' (User Pool ID) containing 'ap-southeast-1\_2OcRf0tress' and 'アプリクライアント ID' (App Client ID) containing '6+Jzunm4eTmpeA9dje'. Both fields are also highlighted with red boxes.

## 17. [プールの作成] をクリックします。

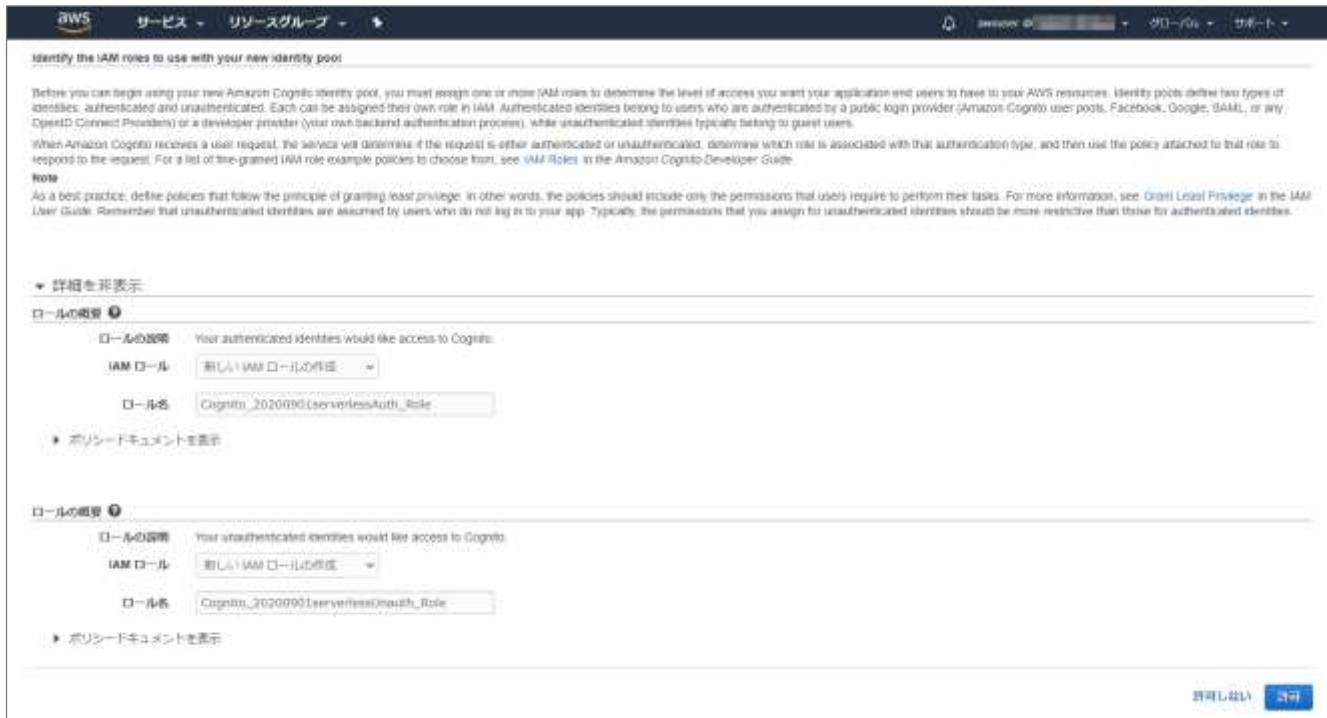
18.Cognito のウィザードが自動的に IAM ロールを作成することに対して許可を求められます。

[詳細を表示] をクリックして展開します。



19.2つのロール [**Cognito\_YYYYMMDDserverlessAuth\_Role**]（認証された ID 用のロール）および [**Cognito\_YYYYMMDDserverlessUnauth\_Role**]（認証されていない ID 用のロール）が作成されることを確認します。

[許可] をクリックします。



20.ID プールが作成されました。

[AWS 認証情報の取得] のサンプルコードに [ID プールの ID] が記述されています。（後の手順で使用するためメモしておいてください）

The screenshot shows the AWS Cognito Identity Pool configuration page. On the left, there's a sidebar with options: ID プール (selected), ダッシュボード, サンプルコード, and ブラウザ。The main content area has a title "Amazon Cognito での作業開始" and a section "AWS SDK のダウンロード" with a link "AWS SDK for Android をダウンロード". Below it is a section "AWS 認証情報の取得" containing a code snippet:

```
CognitoIdentityCredentialsProvider credentialsProvider = new CognitoIdentityCredentialsProvider(Region.getRegion(Regions.US_EAST_1), "arn:aws:cognito-idp:us-east-1:123456789012:my-pool-id", null);
```

The URL "arn:aws:cognito-idp:us-east-1:123456789012:my-pool-id" is highlighted with a red box.

Below the code, there's a note "次に、認証情報プロバイダーを初期化します。" followed by a link "Cognito ID での作業開始" and a "ダッシュボードに移動" button.

### 4.3.3. Web ブラウザからの動作確認

1. [webpage] フォルダに「4.2. API Gateway を追加して動作確認を行う」で使用したファイルが残っていると思います。[433]のフォルダからファイルを[webpage]の下にコピーします。
2. [webpage] フォルダに以下のファイルがあります。
  - [cognito.js]
  - [config.js]
  - [login.html]
  - [mypage.html]
3. [config.js] をテキストエディタで開きます。

[userPoolId]、[appClientId]、[identityPoolId] の各値を、前節・前々節で確認した [ユーザープール ID]、[アプリクライアント ID]、[ID プール ID] の値にそれぞれ書き換えます。

※ 東京以外のリージョンを利用している場合は、[region] の値も併せて書き換えてください

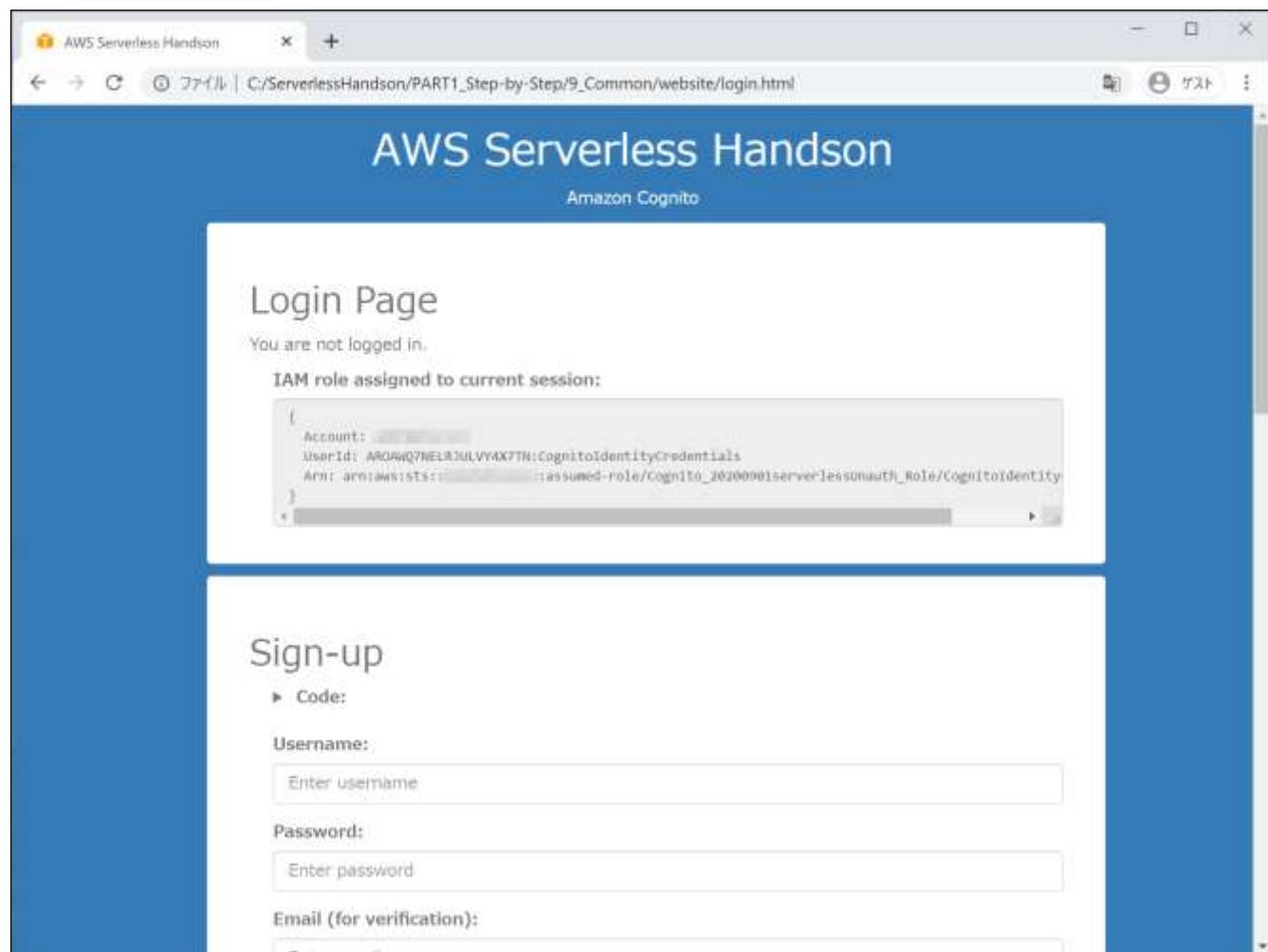
```
const CognitoConfig = {  
    region: 'ap-northeast-1',  
  
    // User Pool  
    userPoolId: 'ap-northeast-1 XXXXXXXX',  
    appClientId: 'XXXXXXXXXXXXXXXXXXXXXX',  
  
    // Federated Identity  
    identityPoolId: 'ap-northeast-1:XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXX',  
}
```

4. この時点で、[webpage] フォルダ配下の構成は以下のようになっているはずです。

```
[website]
├─ [img] ... アイコン等の画像ファイル
├─ [lib] ... 各種ライブラリ/SDK (必要なものを格納済み、ただし apigClient.js を除く)
│  ├─ [apigateway]
│  │  ├─ [lib]
│  │  └─ apigClient.js ... API Gateway アクセスクライアントのクラス定義
│  ├─ [awssdk]
│  └─ [cognito]
├─ [style] ... CSS ファイル
├─ cognito.js ... Cognito ハーアクセスを行う処理を記述した JavaScript コード
├─ config.js ... Cognito の ID 情報等を記述したファイル
├─ login.html ... Web ブラウザで最初に表示するページ
└─ mypage.html ... ログイン処理が行われた後に遷移する Web ページ
```

他のファイルは存在していても邪魔にはならないので無視します。（後ほど使います。）

5. [webpage] フォルダ直下の [login.html] を Web ブラウザで開きます。



6. 画面上部に「Cognito ID プールによって現在のブラウザセッションに割り当てられた IAM ロール」の情報が表示されています。

まだ認証が行われていませんので **[Cognito\_YYYYMMDDserverlessUnauth\_Role]** が割り当てられていることが分かります。

The screenshot shows a "Login Page" with the message "You are not logged in." Below it, a box displays the "IAM role assigned to current session:" information. The JSON output is partially visible, showing the ARN of the assumed role: "arn:aws:sts::[REDACTED]:assumed-role/cognito\_20200901serverlessUnauth\_Role/CognitoIdentity".

7. まず、サインアップ（ユーザー登録）を行います。

ユーザー名、パスワード、メールアドレスを入力して、**[Sign-up]** をクリックします。

※ 確認メールが送信されますので、受信可能なメールアドレスを指定してください。

The screenshot shows the "Sign-up" page. It has fields for "Username" (user1), "Password" (\*\*\*\*\*), and "Email (for verification)" (user1@example.com). A blue "Sign-up" button is visible. Below the form, there is a "Result:" section which is currently empty.

8. [Result] 欄にエラーが出力されたりせず、下図のように表示されればサインアップ成功です。

Sign-up

▶ Code:

Username:  
Enter username

Password:  
Enter password

Email (for verification):  
Enter email

Sign-up

Result:

(Mon Sep 14 2020 23:55:18 GMT+0900 (日本標準時))

```
{ "username": "user1" }
```

9. AWS マネジメントコンソールで Cognito ユーザープールの画面を開き、[ユーザーとグループ] を選択します。

サインアップを行ったユーザーが登録されていることが確認できます。（表示されない場合は、右上のリロードボタンをクリックして表示を更新してください）

アカウントのステータスが [UNCONFIRMED]（未確認）であることを確認してください。

The screenshot shows the AWS Management Console interface for a Cognito User Pool. The left sidebar has a 'User Pools' section with a single pool named '20200901serverless'. The main area is titled 'ユーザーとグループ' (Users and Groups). On the left, there's a navigation bar with 'ユーザーとグループ' selected. The main content area has tabs for 'ユーザー' (User) and 'グループ' (Group), with 'ユーザー' selected. There are buttons for 'ユーザーをインポート' (Import User) and 'ユーザーの作成' (Create User). A search bar at the top right contains 'user'. Below the search bar, a table lists users. The first row in the table is highlighted with a red border. The columns are 'ユーザー名' (User Name), '有効' (Enabled), 'アカウントのステータス' (Account Status), 'Eメール確認済み' (Email Verified), '電話番号確認済み' (Phone Number Verified), '更新済み' (Last Updated), and '作成日' (Created Date). The data for the first user is: user, Enabled, UNCONFIRMED, false, -, Sep 14, 2020 12:33:17 PM, Sep 14, 2020 12:33:17 PM.

10. 以下のような確認メールが届いていることを確認します。

[confirmation code] (確認コード、6桁の数字) は次の手順で必要になります。



11. 次に、アクティベーション（メールアドレスによる本人確認）を行います。

ユーザー名、および、メールで送られてきた [confirmation code] を入力して、[Activete] をクリックします。

The image shows a screenshot of an activation form titled "Activation". The form includes the following fields:

- A "Code:" label followed by a text input field containing "user1".
- A "Confirmation Code:" label followed by a text input field containing "343587".
- A blue rectangular button labeled "Activate".
- A large, empty gray rectangular area labeled "Result:".

12. [Result] 欄に [success] と表示されればアクティベーション成功です。

Activation

▶ Code:

Username:

Confirmation Code:

Activate

Result:

(Mon Sep 14 2020 22:01:58 GMT+0900 (日本標準時))

"SUCCESS"

13. Cognito ユーザープールの [ユーザーとグループ] 画面で、リロードして表示を更新します。

アカウントのステータスが [CONFIRMED] (確認済み) に変わっていることを確認します。

ユーザー名	有効	アカウントのステータス	Eメール確認済み	電話番号確認済み	登録済み	作成日
user	Enabled	CONFIRMED	No	—	Sep 14, 2020 1:01:57 PM	Sep 14, 2020 12:58:17 PM

14. ユーザーがアクティベートされたので、サインイン（ログイン）を行います。

ユーザー名とパスワードを入力して、[Sign-in] をクリックします。

The screenshot shows a 'Sign-in' form. It has fields for 'Username' (containing 'user1') and 'Password' (containing '\*\*\*\*\*'). Below the password field is a blue 'Sign-in' button.

15. サインインが成功すると、[My Page] に遷移します。

The screenshot shows a web browser window titled 'AWS Serverless Handson'. The address bar indicates the file is located at 'C:/ServerlessHandson/PART1\_Step-by-Step/9\_Common/website/mypage.html'. The main content area is titled 'AWS Serverless Handson' and 'Amazon Cognito'. It displays a 'My Page' section with the message 'You are logged in.' and an 'IAM role assigned to current session:' table. The table shows the following data:

Account:	userId: AROAWQ7NELIJR8U09M25H:CognitoIdentityCredentials
Arns:	arn:aws:sts:::assumed-role/Cognito_20200901serverlessAuth_Role/CognitoIdentity

Below this is an 'About current user' section with the heading 'How to get current CognitoUser'. It contains the following JavaScript code:

```
var UserPool = new AWS.Cognito.CognitoIdentityServiceProvider.CognitoUserPool({
    UserPoolId: UserPoolId,
    ClientId: AppClientId
});

var currentCognitoUser = UserPool.getCurrentUser();
```

At the bottom of this section is a 'Reference username' link.

16. My Pageにおいても、画面上部に「Cognito ID プールによって現在のブラウザセッションに割り当てられた IAM ロール」の情報が表示されます。

認証が行われましたので **[Cognito\_YYYYMMDDserverlessAuth\_Role]** が割り当てられています。



17. [About current user] には、Cognito ユーザープールから現在のブラウザセッションが取得した情報が表示されています。

- **Reference username**
- **Identity Token**
- **ID Token Expiration**
- **Access Token**

18. 画面を一番下までスクロールすると [Sign-out] ボタンがあります。

サインアウトすると [Login Page] に戻ります。

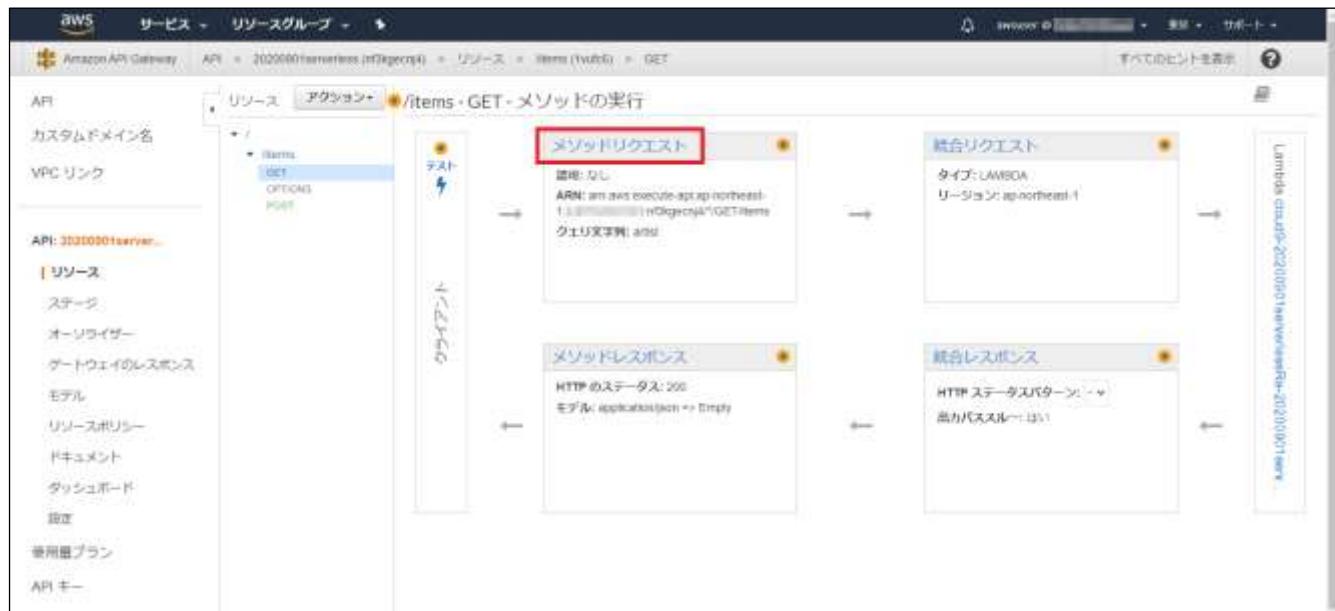


## 4.4. Cognito による認証と API Gateway を組み合わせる

### 4.4.1. API Gateway に認証の設定を追加

1. AWS マネジメントコンソールで **[API Gateway]** を開き、GET メソッド実行の設定画面を開きます。

**[メソッドリクエスト]** をクリックします。(ステージではなく、リソースを指していることを確認してください)



2. **[認可]** の右側にある鉛筆ボタンをクリックします。

Amazon API Gateway API: 20200901server... リソース / items (Twinkl) GET

API: カスタムドメイン名 VPC リンク

リソース アクション+ + メソッドの実行 /items - GET - メソッドリクエスト

このメソッドが認可設定と、許可可能なパラメータに関する情報を指定します。

認可なし  リクエストの検証なし  APIキーの必要性 false

URL クエリ文字列パラメータ  HTTP リクエストヘッダー  リクエスト本文書  SDK 設定

3. プルダウンから [AWS\_IAM] を選択して、右側のチェックボタンをクリックして確定します。

Amazon API Gateway API: 20200901server... リソース / items (Twinkl) GET

API: カスタムドメイン名 VPC リンク

リソース アクション+ + メソッドの実行 /items - GET - メソッドリクエスト

このメソッドが認可設定と、許可可能なパラメータに関する情報を指定します。

認可 AWS\_IAM  リクエストの検証なし  APIキーの必要性 false

URL クエリ文字列パラメータ  HTTP リクエストヘッダー  リクエスト本文書  SDK 設定

4. 画面上部の [] をクリックして、前の画面に戻ります。

aws サービス リソースグループ

Amazon API Gateway API: 20200801serverless (InfraGeoAPI) リソース / Items (Items) GET

すべてのヒントを表示 ?

API リソース アクション + **メソッドの実行 /Items - GET - メソッドリクエスト**

このメソッドの認可設定と、使用可能なパラメータに関する情報を見つけるには、このセクションを確認してください。

設定

認可 AWS\_IAM

リクエストの検証なし

APIキーの必要性なし

URL クエリ文字列パラメータ

HTTP リクエストヘッダー

リクエスト本文書

SDK 設定

API: 20200801serverless

リソース:

- ステータス
- オーバーライダー
- ゲートウェイのレスポンス
- モデル
- リソースポリシー
- ドキュメント
- ダッシュボード
- 設定
- 使用回数プラン
- API キー
- クライアント証明書

5. GET メソッド呼び出しの ARN が表示されています。後の手順で使用するためメモしておいてください。

The screenshot shows the AWS API Gateway console. On the left, the navigation pane lists various API-related options like 'API', 'リソース', 'ステージ', etc. The main area displays the execution of a GET method. A central box labeled 'メソッドリクエスト' shows the ARN of the Lambda function: 'arn:aws:lambda:ap-northeast-1:xxxxxxxxxxxx:infogeonam:GETitems'. This ARN is highlighted with a red box. To the right, there are boxes for '統合リクエスト' and '統合レスポンス', both of which are currently empty.

6. POST メソッド実行の設定画面を開きます。

[メソッドリクエスト] をクリックします。

This screenshot shows the same AWS API Gateway interface, but for a POST method. The central 'メソッドリクエスト' box now displays the ARN of the Lambda function: 'arn:aws:lambda:ap-northeast-1:xxxxxxxxxxxx:infogeonam:POSTitems'. This ARN is also highlighted with a red box. The other components of the flow (統合リクエスト and 統合レスポンス) remain empty.

7. 【認可】の右側にある鉛筆ボタンをクリックします。

The screenshot shows the AWS API Gateway console. On the left, there's a sidebar with various options like 'API', 'カスタムドメイン名', 'VPC リンク', etc. The main area shows a resource named 'Items' with a single POST method. Under the 'Authorization' section, the dropdown menu is open, showing 'None' as the selected option. A red box highlights the pencil icon next to the dropdown.

8. プルダウンから [AWS\_IAM] を選択して、右側のチェックボタンをクリックして確定します。

This screenshot is identical to the previous one, but the 'Authorization' dropdown has been changed. Now, 'AWS\_IAM' is selected, and a red box highlights the checkmark button to its right, indicating that the change has been confirmed.

9. 画面上部の [ メソッドの実行] をクリックして、前の画面に戻ります。

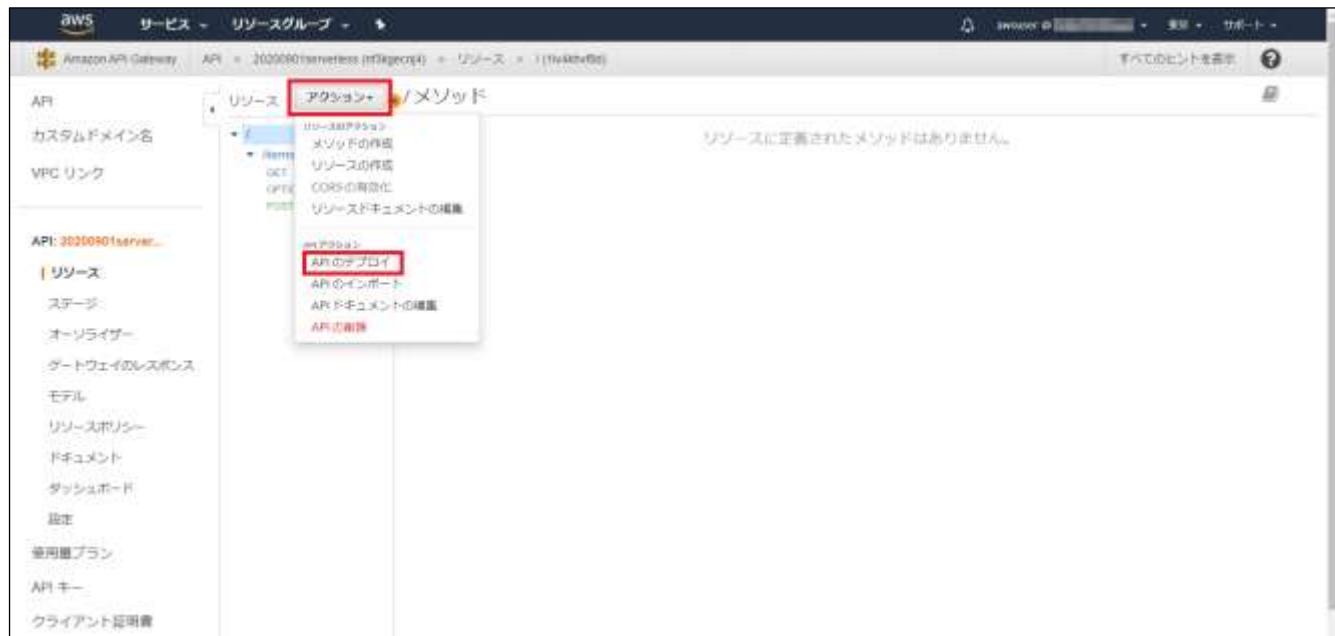
The screenshot shows the AWS API Gateway configuration interface. The left sidebar lists various API resources like 'API: 20200901server...', 'リソース', 'ステージ', etc. The main panel is titled 'アクション+' and shows the path '/Items - POST - メソッドリクエスト'. A red box highlights the '← メソッドの実行' button at the top left of this panel. Below it, there are several configuration sections: 'リクエストの検証なし', 'APIキーの必要性なし', 'URL クエリ文字列/パラメータ', 'HTTP リクエストヘッダー', 'リクエスト本文書', and 'SDK 設定'.

10. POST メソッド呼び出しの ARN が表示されています。後の手順で使用するためメモしておいてください。

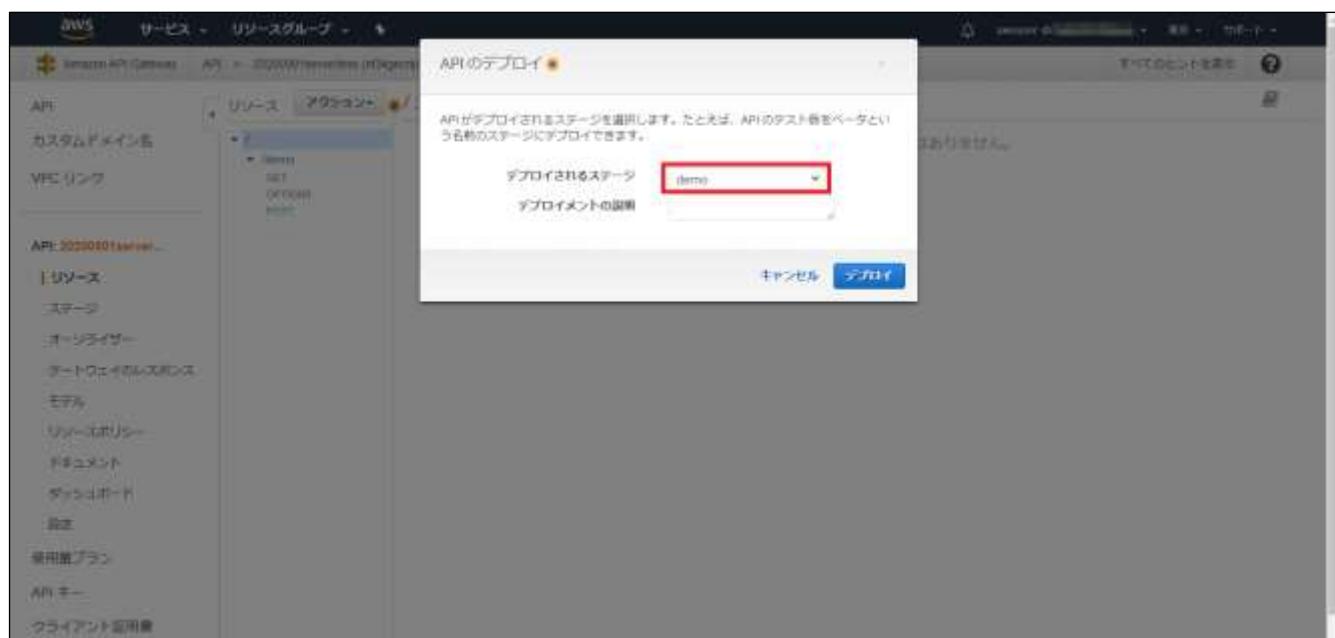
The screenshot shows the AWS API Gateway execution history. It displays a sequence of steps: 'メソッドリクエスト' (Method Request) with the ARN 'arn:aws:execute-api:ap-northeast-1:123456789012:20200901server-/Items-POST' highlighted with a red box; 'Lambda' (Lambda function); 'Lambda' (Lambda function); and '統合レスポンス' (Integration Response). The Lambda functions are associated with the ARN 'arn:aws:lambda:ap-northeast-1:123456789012:LambdaFunctionName'.

11. リソースのツリー階層から [/] をクリックして選択します。

[アクション] プルダウンから [API のデプロイ] を選択します。



12. [デプロイされるステージ] で [demo] を選択して、[デプロイ] をクリックします。



### 13. デプロイが行われました。

The screenshot shows the AWS API Gateway Stage Editor for the 'demo' stage of the '20200801serverless' API. The left sidebar lists various API configuration options like 'API', 'カスタムドメイン名', and 'VPC リンク'. The main area displays deployment details for the 'demo' stage:

- API: 20200801serverless**
- リソース:** (none)
- ステージ:** demo (selected)
- キャッシュ設定:** API キャッシュを有効化:
- デフォルトのメソッドスロットリング:** このステージのメソッド毎のデフォルトのスロットリングレベルを選択します。このステージの各メソッドでは、これらのレートおよびバーストの設定を優先します。現在のアクションレベルのスロットリングレートは、1秒あたりのリクエスト数が 10000 で、バーストのリクエスト数は 5000 です。API Gateway のスロットリングの詳細。
- スロットリングの有効化:** レート: 10000 リクエスト数  
バースト: 5000 リクエスト数
- ウェブアプリケーションファイアウォール (WAF):** 詳細はこちら。このステージに適用されるウェブ ACL を選擇します。
- ウェブ ACL:** なし  ウェブ ACLを作成する

#### 4.4.2. IAM ロールにポリシーを追加

1. AWS マネジメントコンソールで [IAM ポリシー] の画面を開きます。

[ポリシーの作成] をクリックします。

The screenshot shows the AWS Management Console with the 'Identity and Access Management (IAM)' service selected. In the top navigation bar, the 'Policies' tab is highlighted with a red box. Below the navigation bar, there is a search bar and a table listing several policies. The columns in the table are 'Policy Name', 'Type', 'Who can use it', and 'Description'. The policies listed are: '20230907removalLimitPolicy' (User-managed, Permissions policy), 'AccessAnalyzerServiceRolePolicy' (AWS-managed, AWS-managed), 'AdministratorAccess' (AWS-managed, Permissions policy), 'AlexaForBusinessDeviceSetup' (AWS-managed, AWS-managed), and 'AlexaForBusinessHealthAccess' (AWS-managed, AWS-managed). A note at the bottom right of the table says '732 件の結果を表示中'.

2. [サービス] をクリックして展開します。

The screenshot shows the 'Create Policy' wizard, Step 1: Set the service. The title is 'Policy Creation'. It includes a note about using the visual editor or JSON. There are tabs for 'Visual Editor' (selected) and 'JSON'. Below the tabs, there are sections for 'Actions', 'Resources', and 'Request Conditions'. A large button labeled 'Next Step' is at the bottom right.

3. 検索欄に [executeapi] と入力します。

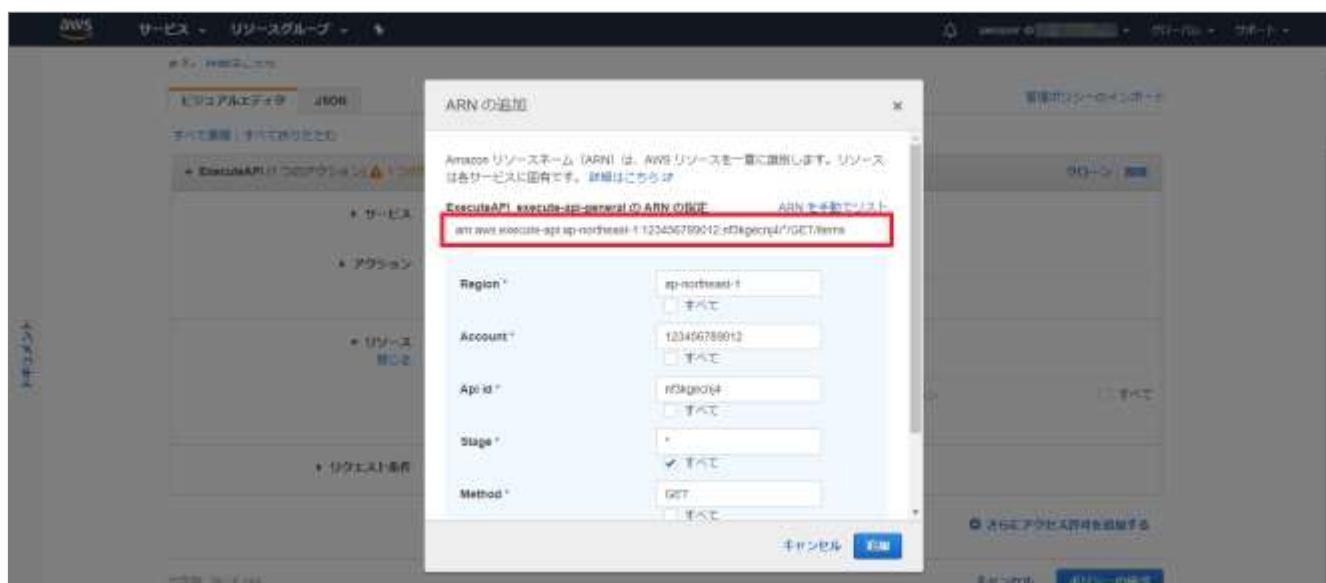
表示された [ExecuteAPI] をクリックします

The screenshot shows the 'Create Policy' wizard, Step 1: Set the service, with the search term 'executeapi' entered in the search bar. The results list shows 'executeapi' with a red box around it. Other results like 'executeapi' (Lambda) and 'executeapi' (CloudWatch Metrics) are also listed. The 'Actions', 'Resources', and 'Request Conditions' sections are visible below the search results.

4. [アクション] を展開して、[書き込み]-[Invoke] にチェックを入れます。



5. [ExecuteAPI execute-api-general の ARN の指定] 欄に、前節で確認した「GET メソッド呼び出しの ARN」を入力します。[追加] をクリックします。

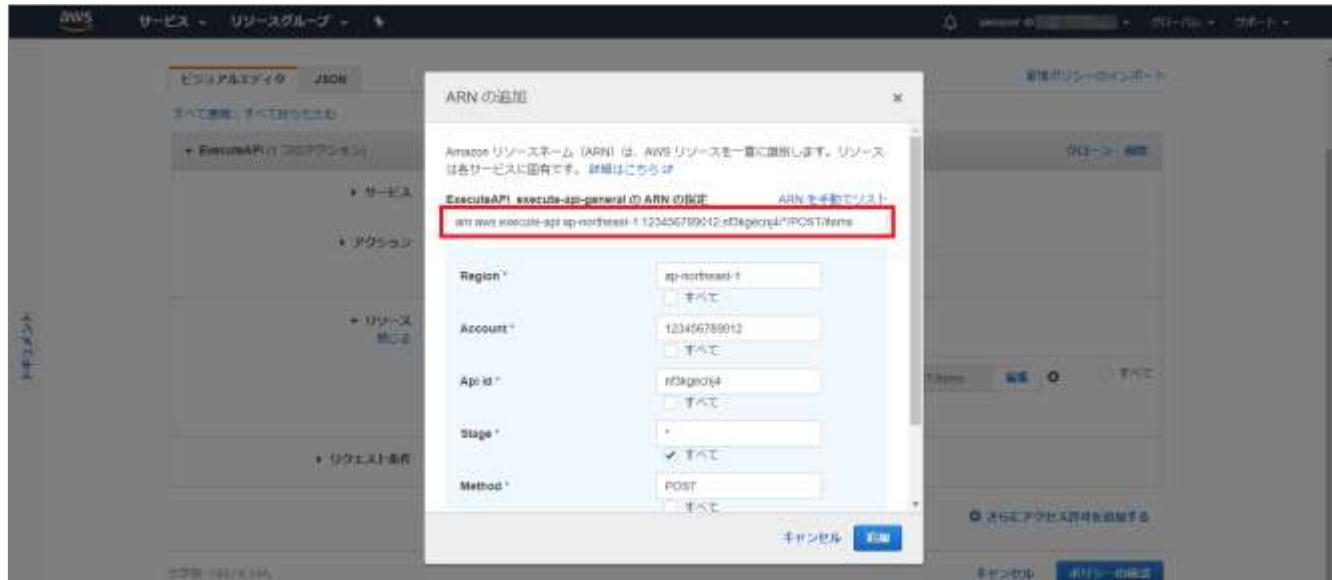


6. GET メソッド呼び出しの ARN が追加されたことを確認します。

続けて [ARN の追加] をクリックします。



7. [ExecuteAPI execute-api-general の ARN の指定] 欄に、前節で確認した「POST メソッド呼び出しの ARN」を入力します。[追加] をクリックします。



8. POST メソッド呼び出しの ARN が追加されたことを確認します。



9. [ポリシーの確認] をクリックします。

10. [名前] 欄に [**YYYYMMDDserverlessAPIGatewayGetAndPostPolicy**] と入力します。  
(**YYYYMMDD** は本日の日付)



11. [ポリシーの作成] をクリックします。

12. GET メソッドおよび POST メソッドを呼び出す権限を定義したポリシー

[**YYYYMMDDserverlessAPIGatewayGetAndPostPolicy**] が作成されました。

作成されたポリシーをクリックして定義内容を確認します。



13. IAM ポリシーの定義内容を JSON 形式で表示します。

以下のような定義内容となっていることを確認します。

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "VisualEditor0",  
            "Effect": "Allow",  
            "Action": "execute-api:Invoke",  
            "Resource": [  
                "arn:aws:execute-api:ap-northeast-1:123456789012:nf3kgecnj4/*/GET/items",  
                "arn:aws:execute-api:ap-northeast-1:123456789012:nf3kgecnj4/*/POST/items"  
            ]  
        }  
    ]  
}
```

14. もう 1 つ IAM ポリシーを作成します。[ポリシーの作成] をクリックします。

The screenshot shows the AWS Identity and Access Management (IAM) Policies page. On the left, there's a sidebar with options like 'ダッシュボード', 'サービス', 'リソースグループ', and a search bar. The main area has a heading 'ポリシーの作成' (Create New Policy) with a red box around it. Below it is a search bar and a table with columns 'ポリシー名', 'タイプ', '渡として使用', and '説明'. The table lists several built-in policies such as 'AWSLambdaBasicExecutionRole', 'AdministratorAccess', and 'AmazonDynamoDBFullAccess'. A note at the bottom right says 'TSS 許可の結果を表示中'.

15. 前の手順と同様にしてポリシーの設定を行います。

- サービス: [ExecuteAPI] を選択
- アクション: [書き込み]-[Invoke] を選択
- リソース: 「GET メソッド呼び出しの ARN」のみを指定 (POST メソッドは指定しません)

The screenshot shows the AWS Lambda function configuration page for a function named 'execute-api-general'. It has tabs for 'ビジュアルエディタ' and 'JSON'. The main area shows the configuration for the 'ExecuteAPI' layer, with 'サービス' set to 'ExecuteAPI' and 'アクション' set to '書き込み' (Write) and 'Invoke'. Under 'リソース', the 'ARN' field contains 'arn:aws:execute-api::ap-northeast-1:xxxxxxxxxxxx:execute-api/\*:GET/\*'. There are buttons for 'クリーン' (Clean), 'すべて' (All), and '確認' (Review). At the bottom, there are buttons for 'キャンセル' (Cancel) and 'ポリシーの確認' (Review Policy).

16. [ポリシーの確認] をクリックします。

17. [名前] 欄に [**YYYYMMDDserverlessAPIGatewayGetOnlyPolicy**] と入力します。  
 (YYYYMMDD は本日の日付)

18. [ポリシーの作成] をクリックします。

19. GET メソッドのみを呼び出す権限を定義したポリシー

[**YYYYMMDDserverlessAPIGatewayGetOnlyPolicy**] が作成されました。

作成されたポリシーをクリックして定義内容を確認します

ポリシー名	タイプ	次として使用	説明
20200901serverlessAPIGateway...	ユーザーによる管理	なし	
<b>20200901serverlessAPIGateway...</b>	ユーザーによる管理	なし	
20200901serverlessLambdaPolicy	ユーザーによる管理	Permissions policy (1)	
AccessAnalyzerServerlessPolicy	AWS による管理	なし	Allow Access Analyzer to analyze resource metadata
AdministratorAccess	ジョブ権限	Permissions policy (1)	Provides full access to AWS services and resources
AlexaForBusinessDeviceSetup	AWS による管理	なし	Provide device setup access to AlexaForBusiness services
AlexaForBusinessFullAccess	AWS による管理	なし	Grants full access to AlexaForBusiness advances and access to related AWS services
AlexaForBusinessGatewayDevice...	AWS による管理	なし	Provides gateway destination access to AlexaForBusiness services
AlexaForBusinessFeatureDelegat...	AWS による管理	なし	Provide access to Utilize AWS devices
AlexaForBusinessNetworkProfile...	AWS による管理	なし	This policy enables Alexa for Business to perform automated tasks schedule
AlexaForBusinessRegionAccess	AWS による管理	なし	Provide access to Poly AWS devices
AlexaForBusinessReadOnlyAccess	AWS による管理	なし	Provide read-only access to AlexaForBusiness services
AmazonAPIGatewayAdministrator	AWS による管理	なし	Provides full access to create/delete APIs in Amazon API Gateway via the AWS Lambda interface
AmazonAPIGatewayInvokeFull	AWS による管理	なし	Provides full access to invoke APIs in Amazon API Gateway

20. IAM ポリシーの定義内容を JSON 形式で表示します。

以下のような定義内容となっていることを確認します

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "VisualEditor0",  
            "Effect": "Allow",  
            "Action": "execute-api:Invoke",  
            "Resource": [  
                "arn:aws:execute-api:ap-northeast-1:123456789012:nf3kgecnj4/*/GET/items"  
            ]  
        }  
    ]  
}
```

21. AWS マネジメントコンソールで [IAM ロール] の画面を開きます。

検索ボックスに [Cognito] と入力して、Cognito ID プールに紐付くロールを表示します。

[Cognito\_YYYYMMDDserverlessAuth\_Role] (認証された ID 用のロール) をクリックします。

The screenshot shows the AWS IAM Roles page. On the left, there's a sidebar with navigation links like 'Identity and Access Management (IAM)', 'Groups', 'Users', 'Roles', 'Policies', etc. The main area has a search bar at the top with 'Cognito' typed in. Below it, there are two rows of results:

Role Name	Assumed by Entity	Last Activity
Cognito_20200901serverlessAuth_Role	ID Provider: cognito-identity.amazonaws.com	Today
Cognito_20200901serverlessAuth_Role	ID Provider: cognito-identity.amazonaws.com	Today

22. [ポリシーをアタッチします] をクリックします。

The screenshot shows the detailed view of the 'Cognito\_20200901serverlessAuth\_Role'. The left sidebar is identical to the previous screenshot. The main area shows the role's ARN, creation date, and other metadata. At the bottom, there's a section titled 'Permissions policies (1 適用済みポリシー)' with a red box around the 'ポリシーをアタッチします' button. Below it, there's another section for 'Permissions boundary (not set)'.

23. ポリシーの一覧から [YYYYMMDDserverlessAPIGatewayGetAndPostPolicy] にチェックを入れます。

The screenshot shows the AWS IAM Policies list for the role 'Cognito\_20200901serverlessAuth\_Role'. A specific policy, '20200901serverlessAPIGatewayGetAndPostPolicy', is selected and highlighted with a red box. This policy is listed under the 'inline policies' section.

ポリシー名	タイプ	次として使用
20200901serverlessAPIGatewayGetAndPostPolicy	ユーザーによる管理	なし
20200901serverlessAPIGatewayGetOnlyPolicy	ユーザーによる管理	なし
3200001serverlessAuthPolicy	ユーザーによる管理	Permissions policy (1)
AdministratorAccess	ジョブ権限	Permissions policy (2)
AlexaForBusinessDeviceSetup	AWS による管理	なし
AlexaForBusinessFullAccess	AWS による管理	なし
AlexaForBusinessGatewayExecution	AWS による管理	なし
AlexaForBusinessRoleDelegatedAccessPolicy	AWS による管理	なし
AlexaForBusinessPolicyDelegatedAccessPolicy	AWS による管理	なし
AlexaForBusinessReadOnlyAccess	AWS による管理	なし
AmazonAPIGatewayAdministrator	AWS による管理	なし

24. [ポリシーのアタッチ] をクリックします。

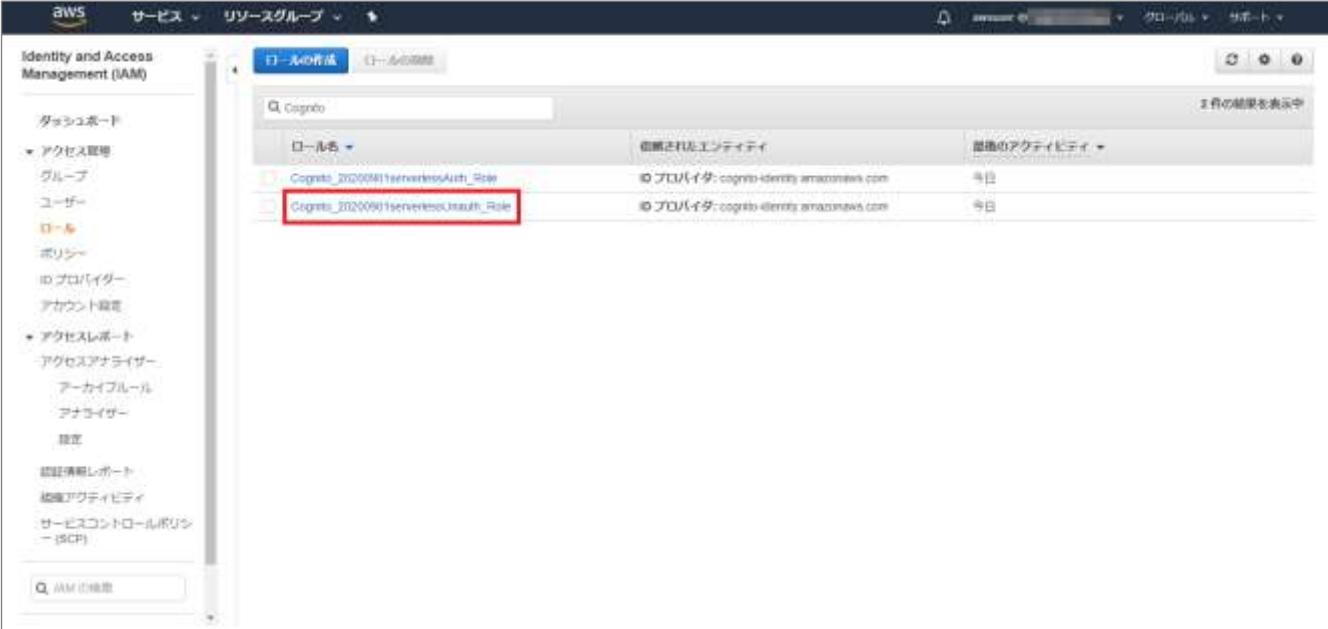
25. ポリシーがアタッチされたことを確認します。

The screenshot shows the 'Role Details' page for the role 'Cognito\_20200901serverlessAuth\_Role'. In the 'Permissions' tab, the message 'Cognito\_20200901serverlessAuth\_Role にポリシー 20200901serverlessAPIGatewayGetAndPostPolicy がアタッチされました。' is displayed. The 'Permissions policies' section lists the attached inline policy '20200901serverlessAPIGatewayGetAndPostPolicy'.

ポリシー名	ポリシータイプ
20200901serverlessAPIGatewayGetAndPostPolicy	inline policy
oneCloud_Cognito_20200901serverlessAuth_Role_1599642518049	inline policy

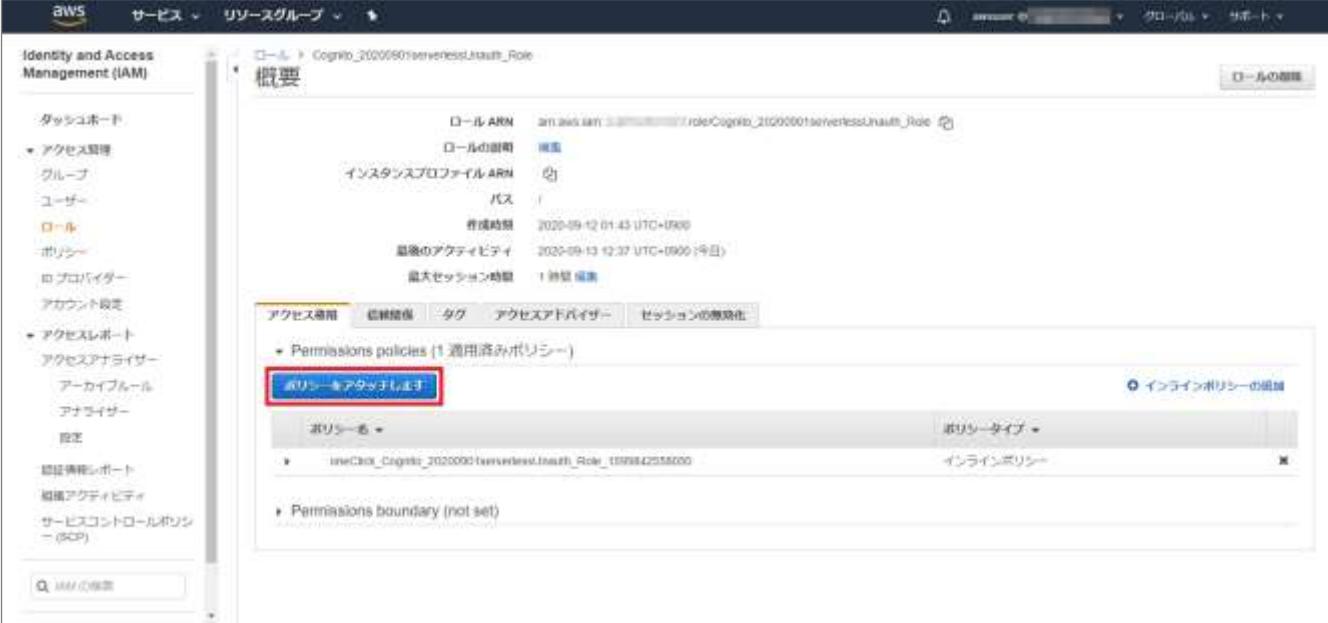
26. [IAM ロール] の画面に戻ります。

[Cognito\_YYYYMMDDserverlessUnauth\_Role] (認証されていない ID 用のロール) をクリックします。



The screenshot shows the AWS IAM service dashboard. On the left, there's a sidebar with navigation links like 'Identity and Access Management (IAM)', 'ダッシュボード', 'アクセス管理', 'グループ', 'ユーザー', 'ロール' (which is highlighted in orange), 'ポリシー', 'ID プロバイダー', 'アカウント設定', and 'アクセスレポート'. The main area has a search bar at the top with 'Cognito' typed in. Below it, there are two tabs: 'ロールの作成' and 'ロールの検索'. A table lists roles, with one row for 'Cognito\_20200901serverlessUnauth\_Role' which is highlighted with a red box. The table columns include 'ロール名' (Role Name), '信頼されたエンティティ' (Trusted Entity), and '最後のアクティビティ' (Last Activity). Both rows show 'ID プロバイダー: cognito-identity.amazonaws.com' and '今日' (Today) under 'Last Activity'.

27. [ポリシーをアタッチします] をクリックします。



The screenshot shows the detailed view of the 'Cognito\_20200901serverlessUnauth\_Role'. The left sidebar is identical to the previous screenshot. The main area is titled '概要' (Overview) and shows the ARN, creation date, and last activity of the role. Below this, there's a tabbed section for 'アクセス権限' (Access Permissions) which is currently selected. It contains a table with one row: 'inlineCloud\_Cognito\_20200901serverlessUnauth\_Role\_1598842558000'. There are tabs for '信頼関係' (Trust Relationship), 'タグ' (Tags), 'アクセスアドバイザー' (Access Advisor), and 'セッションの有効化' (Enable Session). At the bottom of the table, there's a button labeled 'ポリシーをアタッチします' (Attach policy) which is highlighted with a red box. To the right of the table, there's a note about inline policies and a 'Edit' button.

28. ポリシーの一覧から [YYYYMMDDserverlessAPIGatewayGetOnlyPolicy] にチェックを入れます。

The screenshot shows the AWS IAM Policies list. A search bar at the top right contains the text 'YYYYMMDDserverlessAPIGatewayGetOnlyPolicy'. On the left, a sidebar lists policies under 'Cognito\_20200901serverlessUnauth\_Role'. One policy, '320200901serverlessAPIGatewayGetOnlyPolicy', is selected and highlighted with a red box around its checkbox. This policy is also expanded, showing its details: Type: User-managed policy (1), Source: Cognitoによる管理, Status: Enabled.

29. [ポリシーのアタッチ] をクリックします。

30. ポリシーがアタッチされたことを確認します。

The screenshot shows the AWS IAM Role Details page for 'Cognito\_20200901serverlessUnauth\_Role'. In the top right, a message box says 'Cognito\_20200901serverlessUnauth\_Role にポリシー 320200901serverlessAPIGatewayGetOnlyPolicy がアタッチされました。' Below it, the role's ARN is listed as 'arn:aws:iam::[REDACTED]:role/Cognito\_20200901serverlessUnauth\_Role'. The 'Permissions policies' section shows two inline policies attached: '320200901serverlessAPIGatewayGetOnlyPolicy' (User-managed policy) and 'oneCloud\_Cognito\_320200901serverlessUnauth\_Role\_1599942518000' (Inline policy). The 'Permissions boundary' section is shown as '(not set)'.

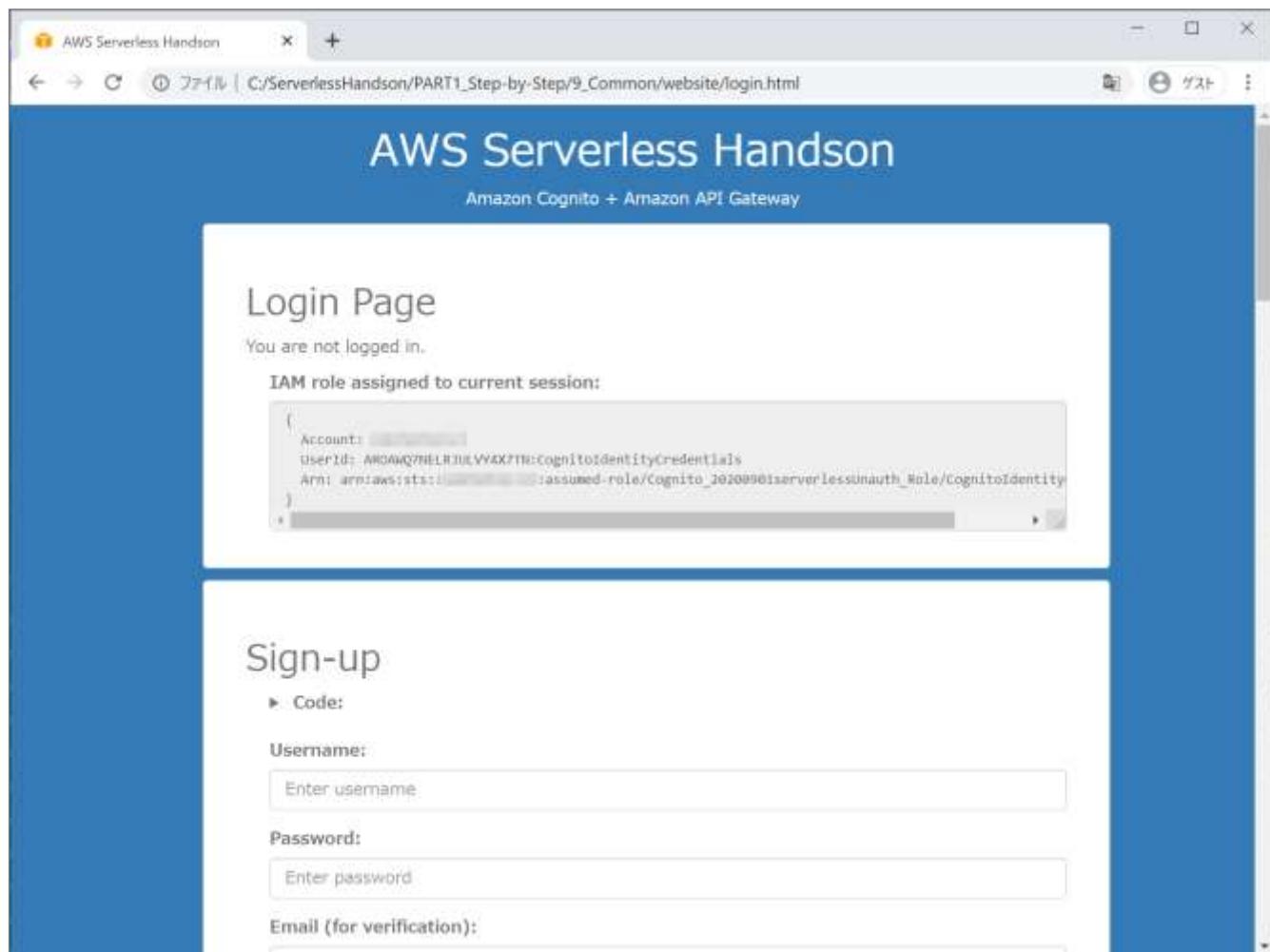
#### 4.4.3. Web ブラウザからの動作確認

0. [webpage] フォルダの [login.html][mypage.html][apigateway.js] を適当な名前にリネームします。 [login.html-][mypage.html-][apigateway.js-] 等
1. [443] のフォルダから以下のファイルを [webpage] フォルダにコピーします。
  - [apigateway.js]
  - [login.html]
  - [mypage.html]
2. この時点で、 [webpage] フォルダ配下の構成は以下のようになっているはずです。

```
[website]
├── [img] ... アイコン等の画像ファイル
├── [lib] ... 各種ライブラリ/SDK (必要なものを格納済み、ただし apigClient.js を除く)
│   ├── [apigateway]
│   │   ├── [lib]
│   │   │   └── apigClient.js ... API Gateway アクセスクライアントのクラス定義
│   ├── [awssdk]
│   └── [cognito]
├── [style] ... CSS ファイル
└── apigateway.js ... API Gateway 処理の JavaScript コード (Cognito 認証処理を追加)
├── cognito.js ... Cognito ヘアクセスを行う処理を記述した JavaScript コード
├── config.js ... Cognito の ID 情報等を記述したファイル
├── login.html ... Web ブラウザで最初に表示するページ (API Gateway の項目を追加)
└── mypage.html ... ログイン処理が行われた後に遷移する Web ページ (同上)
```

その他のファイルは存在していても邪魔にはならないので無視します。

3. [webpage] フォルダ直下の [login.html] を Web ブラウザで開きます



4. 画面を下にスクロールしていくと、Cognito のテスト項目の下に API Gateway のテスト項目が追加されています。

まず、ログインしていない状態で API Gateway のテストを実施します。ログイン状態となっている場合、Sign Out のボタンを押してください。セッションが切れます。

5. GET メソッドの実行を試みます。正常に実行できることを確認します。

API Gateway Test

GET method

▶ Code:

Artist:

Michael Jackson

Run

Status Code:

200

Response Data:

```
[{"Title": "Billie Jean", "Artist": "Michael Jackson"}, {"Title": "Thriller", "Artist": "Michael Jackson"}]
```

6. POST メソッドの実行を試みます。

権限が無いため、ステータスコード [403] が返ってきます。

POST method

▶ Code:

Artist:

Mariah Carey

Title:

All I Want for Christmas Is You

Run

Status Code:

403

Response Data:

```
{"message": "User: arn:aws:sts::1111222233445566 assumed-role/cognito_20200001serverlessauth_Rule/CognitoIdentityCredentials is not authorized to perform: execute-api:Invoke on resource: arn:aws:execute-api:ap-northeast-1:1111222233445566:infkgcnnj4/demo/POST/item"}
```

7. 登録済みのユーザーでサインインを行います。  
[My Page] に遷移した後、同様に API Gateway のテストを実施します。
8. GET メソッドの実行を試みます。正常に実行できることを確認します。

API Gateway Test

GET method

▶ Code:

Artist:  
Michael Jackson

Run

Status Code:  
200

Response Data:

```
[  
  {  
    "title": "Billie Jean",  
    "Artist": "Michael Jackson"  
  },  
  {  
    "title": "Thriller",  
    "Artist": "Michael Jackson"  
  }]
```

9. POST メソッドの実行を試みます。正常に実行できることを確認します。

POST method

▶ Code:

Artist:  
Mariah Carey

Title:  
All I Want for Christmas Is You

Run

Status Code:  
200

Response Data:  
()

# 5. ラボ 2： Amplify を使ってサーバーレスアプリケーションを構築する

本ラボでは、AWS が提供する Web アプリケーション・モバイルアプリケーション構築の統合フレームワークである AWS Amplify を使用して、サーバーレスアプリケーションを構築します。

Amplify では、大きく以下の 3 つの機能が提供されます。

- **Amplify CLI:**

- アプリケーションを構成する各種リソースを対話形式で設定して構築することができる、コマンドラインユーザーインターフェイス

- **Amplify Library:**

- Amplify を使ったアプリケーションを構築する際に、クライアント側のアプリケーション（Web、モバイル）に組み込んで用いる SDK/ライブラリー

- **Amplify Console:**

- GitHub や AWS CodeCommit と統合された CI/DI 機能や、マネージドなアプリケーションホスティング機能を提供

ラボ 2 では、ラボ 1 で使用した AWS サービス「Amazon DynamoDB」「AWS Lambda」「Amazon API Gateway」「Amazon Cognito」を引き続き使用して、サーバーレスアプリケーションを構築していきます。

## 5.1. Amplify を使用する準備を行う

### 5.1.1. Cloud9 環境のディスク容量拡張

ラボ 1 で作成した Cloud9 環境は、ユーザーが利用可能なディスク領域の容量が「10GB」となっています。

これから実施するラボ 2 ではディスク容量が不足する可能性がありますので、予めディスク容量を拡張しておきます。

1. ハンズオン資料のフォルダに、以下のファイルがあります。

- [resize.sh]

このファイルを Cloud9 上にアップロードします。

Cloud9 の画面左側に表示されているディレクトリツリーから、ルートディレクトリ（「**YYYYMMDDserverless - /home/ec2-user/environment**」と表示されています）をクリックして選択状態にします。

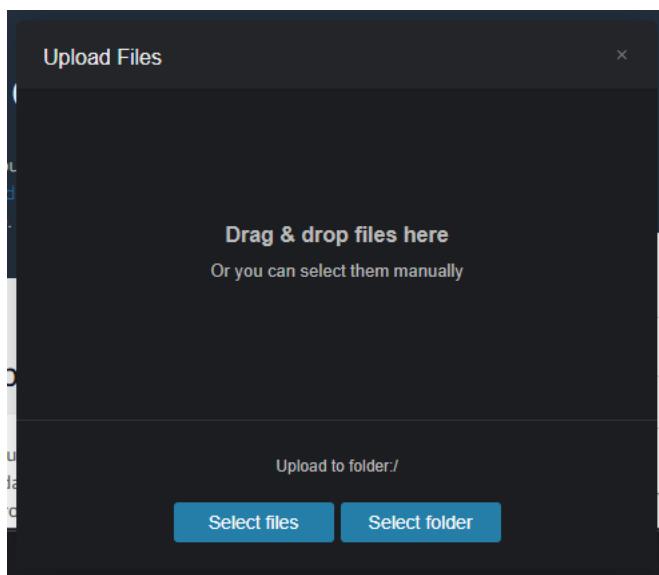


2. 画面上部のメニューから [File]-[Upload Local Files...] を選択します。



3. 下図のウィンドウが表示されますので、アップロードするファイル（resize.sh）をウィンドウ内にドラッグ&ドロップします。

もしくは、[Select files] をクリックしてアップロード対象ファイルを指定しても構いません。



4. ls コマンドを実行して、ファイルがアップロードされたことを確認します。

```
$ ls -l
total 8
-rw-r--r-- 1 ec2-user ec2-user 569 Aug 28 05:46 README.md
-rw-r--r-- 1 ec2-user ec2-user 1304 Sep 1 12:28 resize.sh
```

5. ディスク容量を「20GB」に拡張するために、以下のコマンドを実行します

```
$ sh resize.sh 20
```

6. df コマンドを実行して、/dev/xvda1 のサイズが 20GB に拡張されたことを確認します。

```
$ df
Filesystem      1K-blocks    Used   Available  Use% Mounted on
devtmpfs        493872       60     493812   1% /dev
tmpfs           504564       0     504564   0% /dev/shm
/dev/xvda1      20509288 9630940  10778100  48% /
```

### 5.1.2. 前提環境の確認

1. Cloud9 の画面下部にある **[Terminal]** ウィンドウを使用してコマンドを実行します。

画面が狭い場合は、境界線をドラッグして画面を大きくして頂いて構いません。

<図>

2. Amplify CLI をインストールするには、以下の前提条件を満たしておく必要があります。

- **Node.js** : バージョン **10.x** 以降
- **npm** : バージョン **6.x** 以降

Cloud9 にはこれらのツールが最初から導入されていますが、念のためにバージョンを確認しておきましょう。（“-”は文字コードの関係でコピペでは動作しないケースがありますので手打ちしてください）

```
$ node --version  
v10.22.0
```

```
$ npm --version  
6.14.6
```

Cloud9 を作成したタイミングによっては上記と異なるバージョンになっている場合もありますが、Amplify CLI の前提条件を満たしていれば問題ありません。

### 5.1.3. Amplify CLI のインストール

1. Amplify CLI は「npm」を使用してインストールします。

以下の通りコマンドを実行します。

```
$ npm install -g @aws-amplify/cli
```

2. Amplify CLI のインストールが行われます。

「Successfully installed the Amplify CLI」と出力されればインストール成功です。

3. Amplify CLI が実行できることを確認します。

以下の通りコマンドを実行して、バージョンが表示されれば OK です。 ("-"がコピペでは動作しないので手打ちしてください)

```
$ amplify -version
Scanning for plugins...
Plugin scan successful
4.29.2
```

### 5.1.4. AWS 認証情報の設定

Amplify CLI でコマンドを実行するためには、IAM ユーザーの認証情報が必要になります。

今回は、Amplify CLI の実行時に AWS CLI のプロファイルを指定する方法を採用します。

Cloud9 上で AWS CLI のプロファイルを作成するために、以下の手順を実行します。

1. 以下の通りコマンドを実行します。

```
$ aws configure
```

2. アクセスキーID、シークレットアクセスキー、リージョンの確認を求められますので、何も入力せずに Enter キーを押します。

```
AWS Access Key ID [*****XXXX]:  
AWS Secret Access Key [*****XXXX]:  
Default region name [ap-northeast-1]:
```

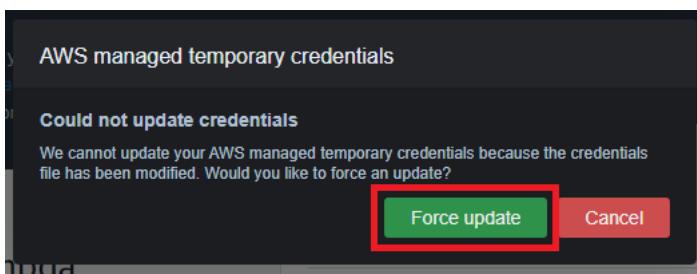
3. 「Default output format」に対して **[json]** と入力して Enter キーを押します。

```
Default output format [None]: json
```

4. これで AWS CLI のプロファイル「default」が Cloud9 上に保存されました。

このタイミングで以下のようなダイアログが表示される場合があります。

その際は **[Force update]** をクリックしてください。



註) PC や Mac で Amplify CLI を利用する場合は、AWS CLI のプロファイルを指定する方法の他に、IAM ユーザーのアクセスキーを Amplify CLI に直接設定する方法もあります。

詳しい手順は下記リンク先を参照してください。

<https://docs.amplify.aws/cli/start/install>

## 5.2. REST API を使った Web アプリケーションを作成する

### 5.2.1. React アプリケーションの作成

1. 以下のコマンドを実行します。 (YYYYMMDD は本日の日付)

```
$ npx create-react-app YYYYMMDDamplify
```

2. 作成したプロジェクトのディレクトリに移動します。

```
$ cd YYYYMMDDamplify
```

3. React を使った Web アプリケーションが動作することをテストします。

以下のコマンドを実行してアプリケーションを起動します。

```
$ npm start
```

4. アプリケーションが起動すると、以下のように表示されます。

```
Compiled successfully!
```

```
You can now view 20200901amplify in the browser.
```

```
Local:          http://localhost:8080
On Your Network:  http://172.XX.XX.XX:8080
```

```
Note that the development build is not optimized.
To create a production build, use npm run build.
```

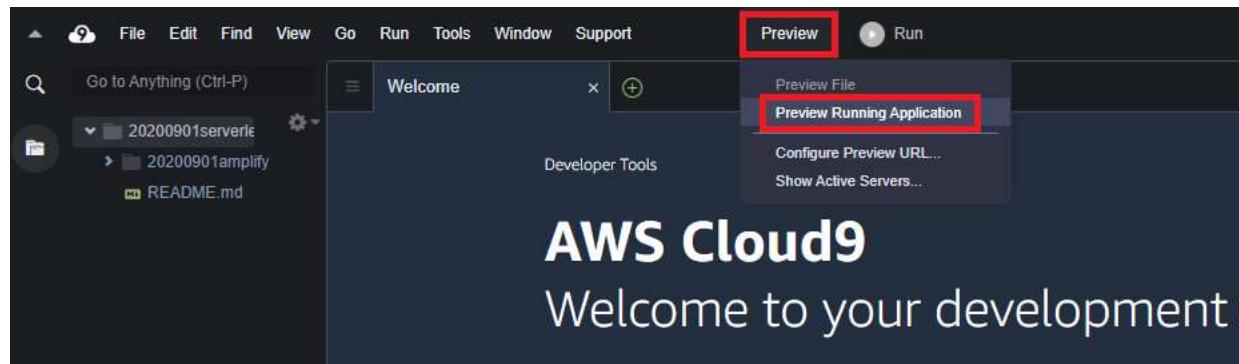
## 5. 起動したアプリケーションに接続を行ってみます。

アプリケーションは Cloud9 上で起動しているので、Web ブラウザのアドレスバーに

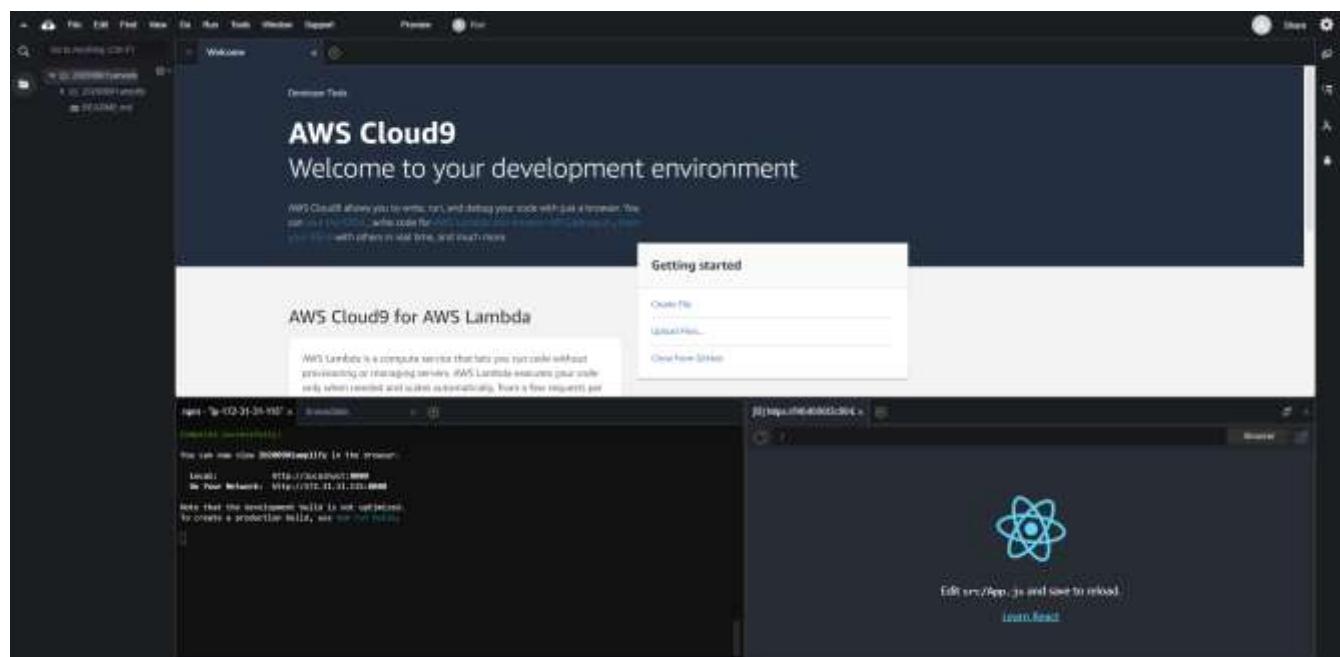
「<http://localhost:8080>」と入力して接続を試みても、接続することはできません。

そこで、Cloud9 の「Preview」機能を使うことでアプリケーションに接続します。

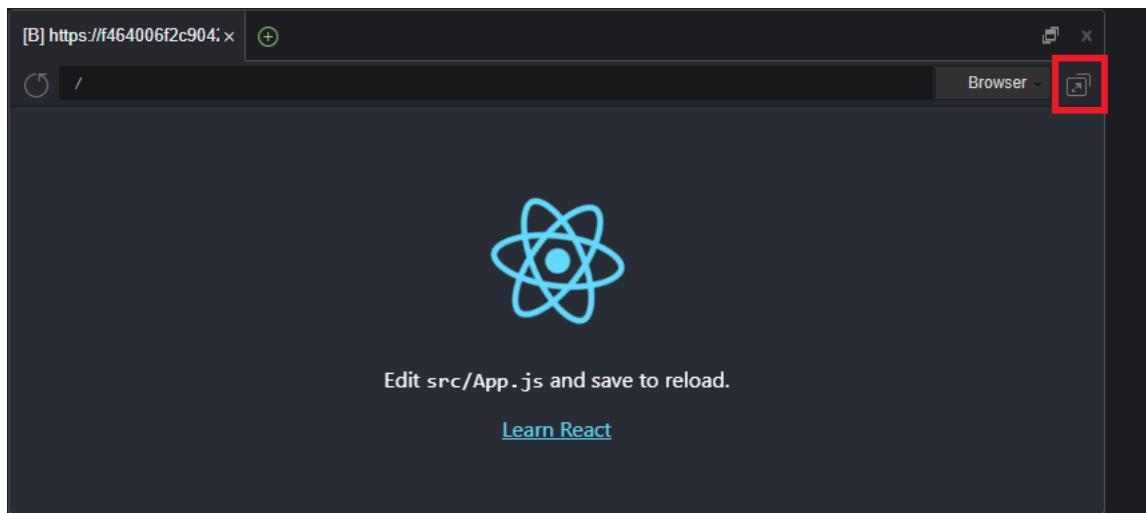
画面上部の [Preview] をクリックして [Preview Running Application] を選択します。



## 6. ウィンドウ内に React サンプルアプリケーションの画面が表示されれば動作確認 OK です。



7. 画面を大きく表示したい場合は下図のボタンをクリックすると、別ウィンドウでアプリケーションの画面が開きます。



8. 動作確認が終わりましたら、ターミナルの画面（「npm start」を実行したウィンドウ）で **[Ctrl]+[C]** を入力して、アプリケーションを停止します。  
アプリケーションを停止すると、プレビューが表示できなくなります。  
(すぐに表示が消えることはありませんが、プレビュー画面をリロードすると「Oops! No application seems to be running here!」と表示されることが分かると思います)

## 5.2.2. Amplify プロジェクトの初期化

1. 以下のコマンドを実行します。

```
$ amplify init
```

2. ここからは、対話形式で Amplify プロジェクトの初期化の設定を行っていきます。

最初に、プロジェクトの名前の指定を求められます。

現在のディレクトリの名前が「デフォルト値」として表示されていることを確認して、何も入力せずに Enter キーを押します。

```
? Enter a name for the project (20200901amplify)
```

※ このようにデフォルト値から変更しない場合は、何も入力せずに Enter キーを押します

3. 環境の名前を入力します。

デフォルト値として「dev」が表示されていますが、今回は [demo] と入力して Enter キーを押します。

```
? Enter a name for the environment (dev) demo
```

※ デフォルト値から変更する場合は、キーボードから値を入力して Enter キーを押します

4. Amplify CLI で使用するエディタを指定します。

カーソルキーの上下で [Vim] を選択して Enter キーを押します。

```
? Choose your default editor (Use arrow keys)
  Visual Studio Code
  Atom Editor
  Sublime Text
  IntelliJ IDEA
  > Vim (via Terminal, Mac OS only)
  Emacs (via Terminal, Mac OS only)
  None
```

※ 選択肢から選択する場合は、カーソルキーの上下で選択して Enter キーを押します

5. 構築するアプリケーションの種類を指定します。

[**javascript**] を選択して Enter キーを押します。

```
? Choose the type of app that you're building (Use arrow keys)
  android
  ios
> javascript
```

6. JavaScript で使用するフレームワークを指定します。

[**react**] を選択して Enter キーを押します。

```
? What javascript framework are you using (Use arrow keys)
  angular
  ember
  ionic
> react
  react-native
  vue
  none
```

7. プロジェクトのビルド、実行についていくつかの指定を求められます。

全てデフォルト値のままで構いませんので、何も入力せずに Enter キーを押します。

```
? Source Directory Path: (src)
? Distribution Directory Path: (build)
? Build Command: (npm run-script build)
? Start Command: (npm run-script start)
```

8. AWS CLI のプロファイルを使用するかどうか聞いてきます。

[**y**] を入力して Enter キーを押します。

```
? Do you want to use an AWS profile? (Y/n) y
```

ここで「Setup new user?」などと聞かれた場合は、AWS CLI のプロファイルが Amplify CLI から認識されていない可能性があります。前節の手順を見直してください。

9. 使用する AWS CLI のプロファイルを指定します。

[**default**] を選択して Enter キーを押します。

```
? Please choose the profile you want to use (Use arrow keys)
> default
```

10. 全ての項目の指定が終わりましたので、初期化の処理が始まります。

初期化の処理が終わるまで数分程度かかりますので、終わるまで待ちます。（画面上では Amplify の初期化に伴って関連する AWS リソースを作成する CloudFormation が実行されている状況が確認できると思います）

初期化の処理が正常に終わると、以下のメッセージが表示されます。

Your project has been successfully initialized and connected to the cloud!

### 5.2.3. API リソースの追加

Amplify CLI では、アプリケーションを構成する要素を「リソース」と呼びます。

最初に、Amplify プロジェクトの中核となる「**API**」リソースを作成します。

「API」リソースには **AWS AppSync** を使用する「**GraphQL**」と、**Amazon API Gateway** を使用する「**REST**」のいずれかを選択することができますが、今回は「REST」を使用します。

また、「API」リソースを作成する過程で、API のバックエンドとなる Lambda 関数を構成する「**Function**」リソース、データストアとなる DynamoDB を構成する「**Storage**」リソースも併せて作成します。

1. 以下のコマンドを実行します。

```
$ amplify add api
```

2. 作成する API の種類を選択します。

[**REST**] を選択して Enter キーを押します。

```
? Please select from one of the below mentioned services: (Use arrow keys)
  GraphQL
  > REST
```

3. 「API リソース」の名前を指定します。

[**YYYYMMDDamplify**] と入力して Enter キーを押します。 (**YYYYMMDD** は本日の日付)

```
? Provide a friendly name for your resource to be used as a label for this
category in the project: (apiXXXXXXX) YYYYMMDDamplify
```

4. API のパスを指定します。

デフォルト値 [**items**] のまま Enter キーを押します。

```
? Provide a path (e.g., /book/{isbn}): (/items)
```

5. ここからは「Function」リソースに関する設定を行います。

Lambda 関数を新規に作成しますので、[Create a new Lambda function] を選択して Enter キーを押します。

```
? Choose a Lambda source (Use arrow keys)
> Create a new Lambda function
```

6. 「Function」リソースの名前を指定します。

[YYYYMMDDamplify] と入力して Enter キーを押します。 (YYYYMMDD は本日の日付)

```
? Provide a friendly name for your resource to be used as a label for this
category in the project: (20200901amplifyXXXXXXX) YYYYMMDDamplify
```

7. Lambda 関数の名前を指定します。

デフォルト値 [YYYYMMDDamplify] のまま Enter キーを押します。

```
? Provide the AWS Lambda function name: (20200901amplify)
```

8. Lambda 関数で使用するランタイムを指定します。

[NodeJS] を選択して Enter キーを押します。

```
? Choose the runtime that you want to use: (Use arrow keys)
.NET Core 3.1
Go
Java
> NodeJS
Python
```

9. 使用する Lambda 関数のテンプレートを指定します。

[CRUD function for DynamoDB] を選択して Enter キーを押します。

```
? Choose the function template that you want to use: (Use arrow keys)
> CRUD function for DynamoDB (Integration with API Gateway)
Hello World
Lambda trigger
Serverless ExpressJS function (Integration with API Gateway)
```

10. ここからは「Storage」リソースに関する設定を行います。

DynamoDB テーブルを新規に作成しますので、[Create a new DynamoDB table] を選択して Enter キーを押します。

```
? Choose a DynamoDB data source option (Use arrow keys)
  Use DynamoDB table configured in the current Amplify project
> Create a new DynamoDB table
```

11. 「Storage」リソースの名前を指定します。

[YYYYMMDDamplify] と入力して Enter キーを押します。 (YYYYMMDD は本日の日付)

```
? Please provide a friendly name for your resource that will be used to
label this category in the project: (dynamoXXXXXXXXXX) YYYYMMDDamplify
```

12. DynamoDB テーブルの名前を指定します。

デフォルト値 [YYYYMMDDamplify] のまま Enter キーを押します。

```
? Please provide table name: (20200901amplify)
```

13. DynamoDB の最初の列（属性）を指定します。

[Artist] と入力して Enter キーを押します。

```
? What would you like to name this column: Artist
```

14. 属性のデータ型を指定します。

[string] を選択して Enter キーを押します。

```
? Please choose the data type: (Use arrow keys)
> string
  number
  binary
  boolean
  list
  map
  null
(Move up and down to reveal more choices)
```

15. 更に列（属性）を追加するか聞かれますので、[y] を入力して Enter キーを押します。

```
? Would you like to add another column? (Y/n) y
```

16. DynamoDB の 2 番目の列（属性）を指定します。

[Title] と入力して Enter キーを押します。

```
? What would you like to name this column: Title
```

17. 属性のデータ型を指定します。

[string] を選択して Enter キーを押します。

```
? Please choose the data type: (Use arrow keys)
> string
  number
  binary
  boolean
  list
  map
  null
(Move up and down to reveal more choices)
```

18. 更に列（属性）を追加するか聞かれますので、[n] を入力して Enter キーを押します。

```
? Would you like to add another column? (Y/n) n
```

19. パーティションキーに設定する属性を指定します。

[Artist] を選択して Enter キーを押します。

```
? Please choose partition key for the table: (Use arrow keys)
> Artist
  Title
```

20. ソートキーを設定するか聞かれますので、[y] を入力して Enter キーを押します。

```
? Do you want to add a sort key to your table? (Y/n) y
```

21. ソートキーに設定する属性を指定します。

[Title] を選択して Enter キーを押します。

```
? Please choose sort key for the table: (Use arrow keys)
> Title
```

22. グローバルセカンダリインデックスを設定するか聞かれますので、[n] を入力して Enter キーを押します。

```
? Do you want to add global secondary indexes to your table? (Y/n) n
```

23. DynamoDB に Lambda トリガーを設定するか聞かれますので、[n] を入力して Enter キーを押します。

```
? Do you want to add a Lambda Trigger for your Table? (y/N) n
```

24. 「Storage」リソースの設定が一通り終わりましたので、ここからは「Function」リソースの設定に戻ります。

Lambda 関数からアクセスするリソースを更に追加するか聞かれますので、[n] を入力して Enter キーを押します。

```
? Do you want to access other resources in this project from your Lambda
function? (Y/n) n
```

25. Lambda 関数を繰り返しスケジュール実行するか聞かれますので、[n] を入力して Enter キーを押します。

```
? Do you want to invoke this function on a recurring schedule? (y/N) n
```

26. Lambda レイヤーの設定を行うか聞かれますので、[n] を入力して Enter キーを押します。

```
? Do you want to configure Lambda layers for this function? (y/N) n
```

27. Lambda 関数の内容を編集することができますが、今回は作成されたテンプレートをそのまま使いますので、[n] を入力して Enter キーを押します。

```
? Do you want to edit the local lambda function now? (Y/n) n
```

28. 「Function」リソースの設定が一通り終わりましたので、最後に「API」リソースの設定に戻ります。

APIへのアクセスを制限するかどうか聞かれますので、[n] を入力して Enter キーを押します。  
(Cognito を使ったアクセス制限は次のステップで行います)

```
? Restrict API access (Y/n) n
```

29. API にパスを追加するか聞かれますので、[n] を入力して Enter キーを押します。

```
? Do you want to add another path? (y/N) n
```

30. リソースの追加が正常に行われると、以下のメッセージが表示されます。

```
Successfully added resource 20200901amplify locally
```

## 5.2.4. API リソースのデプロイ

前の手順で Amplify プロジェクトに「リソース」を追加しました。

この時点では、リソースは PC (今回は Cloud9) のローカル上に Amplify プロジェクトの定義情報として存在しているのみです。

定義情報に基いて AWS 環境に実際にリソースを構築するには「デプロイ」操作を行います。

1. 以下のコマンドを実行します。

```
$ amplify push
```

2. 現在 Amplify プロジェクトに定義されている「リソース」の一覧とステータスが表示されます。

Operation 欄に表示されている「Create」は「定義が作成されたが、まだデプロイされていない」状態であることを示します。

これらのリソースを AWS 環境へデプロイするために、[y] を入力して Enter キーを押します。

```
✓ Successfully pulled backend environment demo from the cloud.
```

```
Current Environment: demo
```

Category	Resource name	Operation	Provider plugin
Storage	20200901amplify	Create	awscloudformation
Function	20200901amplify	Create	awscloudformation
Api	20200901amplify	Create	awscloudformation

```
? Are you sure you want to continue? (Y/n) y
```

3. AWS 環境へのデプロイ処理が始まります。

デプロイ処理が終わるまで数分程度かかりますので、終わるまで待ちます。 (画面上では AWS の各リソースを作成する CloudFormation が実行されている様子が確認できると思います)

デプロイ処理が成功すると、以下のメッセージが表示されます。

```
✓ All resources are updated in the cloud
```

4. デプロイによって、以下の AWS リソースが作成されています。

作成された各リソースは既に設定も行われており、特に内容を確認したり変更したりする必要はありませんが、時間のある方は作成されたリソースの内容を確認するとよいでしょう。

- API Gateway REST API
  - **YYYYMMDDamplify**
- Lambda 関数
  - **YYYYMMDDamplify-demo**
- IAM ロール（Lambda 実行ロール）
  - **YYYYMMDDamplifyLambdaRoleXXXXXXXXX-demo**
- DynamoDB テーブル
  - **YYYYMMDDamplify-demo**

### 5.2.5. Amplify Library のインストール

「Amplify Library」は、Amplify を使ったアプリケーションを構築する際に、クライアント側のアプリケーション（Web、モバイル）に組み込んで用いる SDK/ライブラリーです。

Web クライアントアプリケーション向けには、JavaScript 用の SDK に加えて「React」「Vue」「Angular」などの JavaScript フレームワーク用の UI コンポーネントが提供されており、UI コンポーネントを利用することでアプリケーションにリッチな UI を容易に実装することができます。

今回は、Amplify プロジェクトに「JavaScript 向けの Amplify Library」と「React 向けの UI コンポーネント」をインストールして利用可能にします。

1. 以下のコマンドを実行します。

```
$ npm install aws-amplify @aws-amplify/ui-react
```

2. エラーが発生することなくインストールが完了することを確認します。

## 5.2.6. Web クライアントアプリケーションの構成

1. [src] フォルダに以下のファイルがあります。

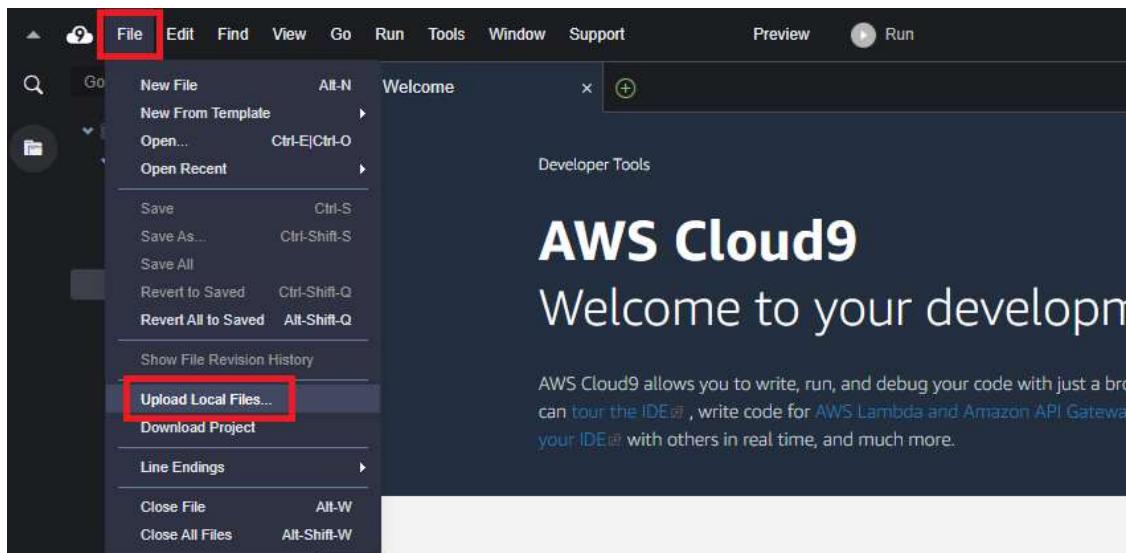
- [ApiGet.js]
- [ApiPost.js]
- [App.css]
- [App.js]
- [aws-logo.png]
- [config.js]

これらのファイルを Amplify プロジェクトの [YYYYMMDDamplify]-[src] フォルダの直下にコピーします。

画面左側のディレクトリツリーから [(ルートディレクトリ)]-[YYYYMMDDamplify]-[src] の階層を開き、[src] ディレクトリをクリックして選択状態にします。



2. 画面上部のメニューから [File]-[Upload Local Files...] を選択します。

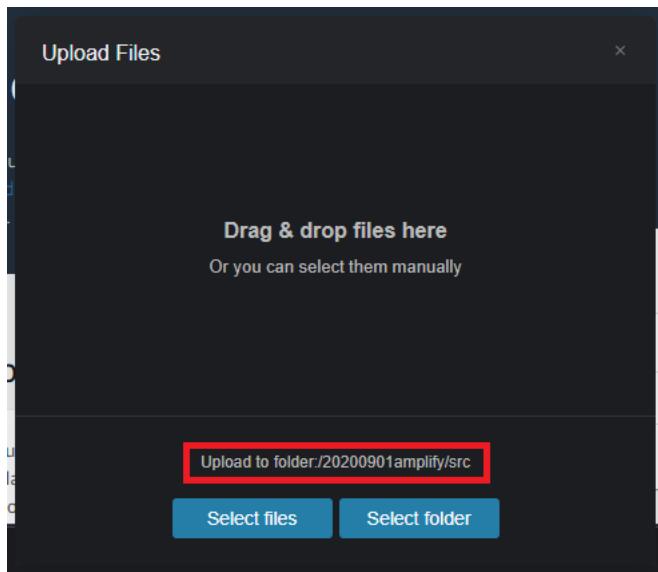


3. 下図のウィンドウが表示されます。

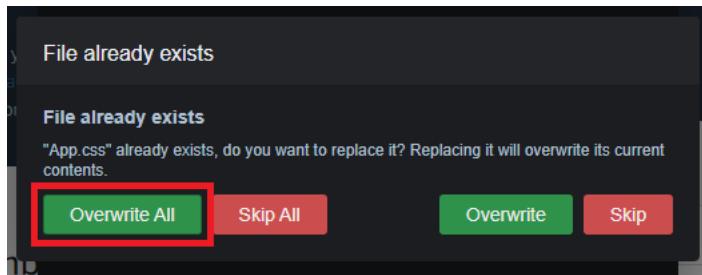
アップロード先ディレクトリが「/YYYYMMDDamplify/src」であることを確認します。

アップロードするファイル群をウィンドウ内にドラッグ&ドロップします。

もしくは、[Select files] をクリックしてアップロード対象ファイルを指定しても構いません。



4. [App.css]、[App.js] の2ファイルは、アップロード先に既に同名ファイルが存在していますので上書きの確認を求められます。[Overwrite All] をクリックして上書きします。

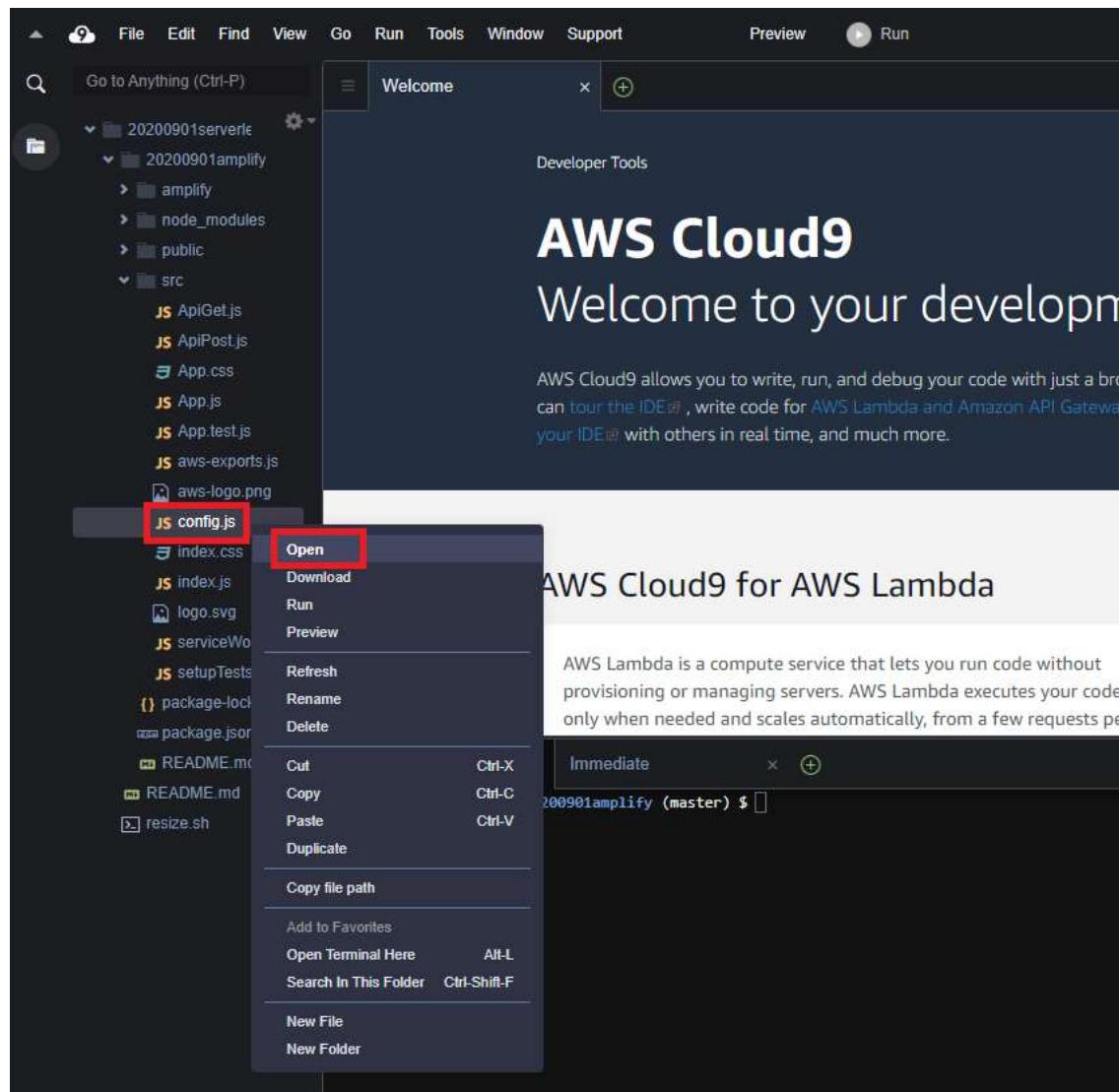


5. ls コマンドを実行して、ファイルがアップロードされたことを確認します。

```
$ ls -l src
total 56
-rw-r--r-- 1 ec2-user ec2-user 2626 Sep  1 13:07 ApiGet.js
-rw-r--r-- 1 ec2-user ec2-user 2293 Sep  1 13:07 ApiPost.js
-rw-rw-r-- 1 ec2-user ec2-user 1350 Sep  1 13:07 App.css
-rw-rw-r-- 1 ec2-user ec2-user 1003 Sep  1 13:07 App.js
-rw-rw-r-- 1 ec2-user ec2-user 280 Sep  1 12:51 App.test.js
-rw-rw-r-- 1 ec2-user ec2-user 654 Sep  1 13:01 aws-exports.js
-rw-r--r-- 1 ec2-user ec2-user 2724 Sep  1 13:07 aws-logo.png
-rw-r--r-- 1 ec2-user ec2-user 160 Sep  1 13:07 config.js
-rw-rw-r-- 1 ec2-user ec2-user 366 Sep  1 12:51 index.css
-rw-rw-r-- 1 ec2-user ec2-user 503 Sep  1 12:51 index.js
-rw-rw-r-- 1 ec2-user ec2-user 2671 Sep  1 12:51 logo.svg
-rw-rw-r-- 1 ec2-user ec2-user 5086 Sep  1 12:51 serviceWorker.js
-rw-rw-r-- 1 ec2-user ec2-user 255 Sep  1 12:51 setupTests.js
```

6. リソース名を記述した設定ファイルを編集します。

画面左側のディレクトリツリーから [src] ディレクトリ直下のファイル [config] を右クリックして、[Open] を選択します。 (vi コマンドを使用して編集しても構いません)



7. API 名を実際に作成した名前に書き換えて、ファイルを保存します。

(パス名は、ここまで手順通りに実施していれば変更する必要はありません)

```
// TODO: 作成した REST API の名前およびパスに書き換えてください
export const apiName = 'YYYYMMDDamplify';
export const basePath = '/items';
```

### 5.2.7. Web クライアントアプリケーションの動作確認

- 以下のコマンドを実行します。

```
$ npm start
```

- アプリケーションが起動すると、以下のように表示されます。（数分間かかるので待ちます）

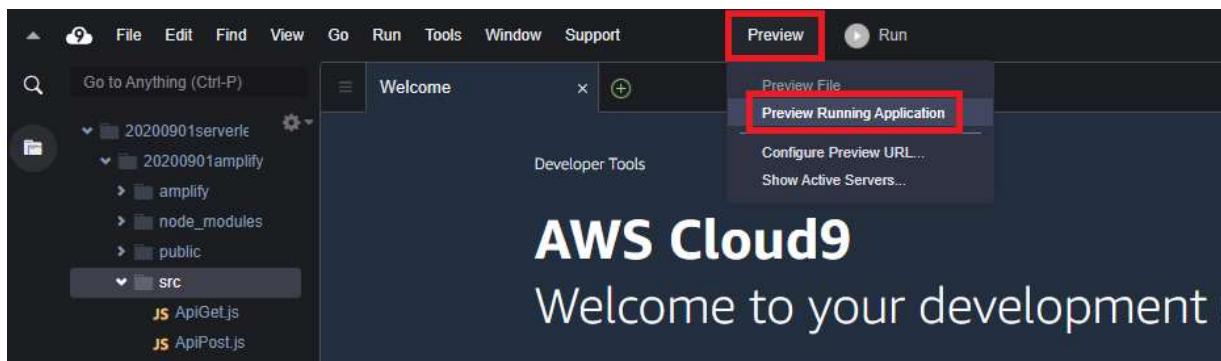
```
Compiled successfully!

You can now view 20200901amplify in the browser.

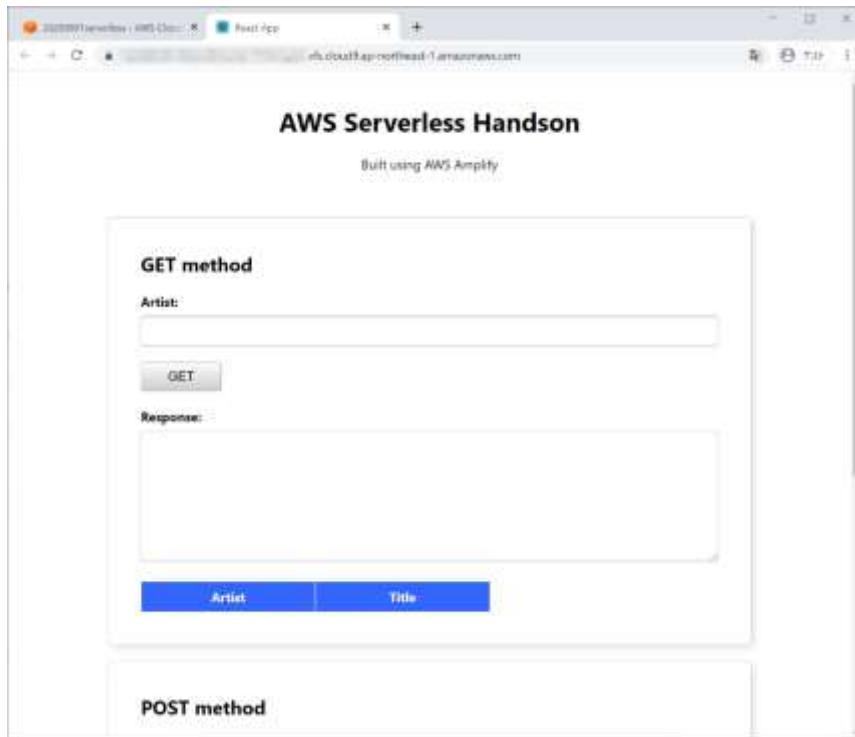
Local:          http://localhost:8080
On Your Network:  http://172.XX.XX.XX:8080

Note that the development build is not optimized.
To create a production build, use npm run build.
```

- 画面上部の [Preview] をクリックして [Preview Running Application] を選択します。



4. 下図のように Web クライアントアプリケーションが正常に表示されることを確認します。



5. 「GET メソッド」 「POST メソッド」 がそれぞれ正常に動作することを確認します。

(まず「POST メソッド」を実行してデータを登録してから、「GET メソッド」を実行するとよいでしょう)

The screenshot shows the "GET method" section of the application. The "Artist:" input field is populated with "Michael Jackson". The "Response:" text area displays the following JSON data:

```
[{"Title": "Billie Jean", "Artist": "Michael Jackson"}, {"Title": "Thriller", "Artist": "Michael Jackson"}]
```

Below this is a table with columns "Artist" and "Title". It contains two rows, both corresponding to Michael Jackson's songs.

Artist	Title
Michael Jackson	Billie Jean
Michael Jackson	Thriller

## 5.3. Cognito による認証を Web プリケーションへ追加する

### 5.3.1. Auth リソースの追加

作成した Web アプリケーションへ **Cognito** による認証機能を追加します。

Amplify プロジェクトに認証機能を提供する「**Auth**」リソースを追加します。

0. Ctl + c で実行を停止します。

1. 以下のコマンドを実行します。

```
$ amplify add auth
```

2. 設定をどのように行うのかを指定します。

[Default configuration] を選択して Enter キーを押します。

```
Do you want to use the default authentication and security configuration?  
(Use arrow keys)  
➤ Default configuration  
  Default configuration with Social Provider (Federation)  
  Manual configuration  
  I want to learn more.
```

3. サインインに用いる項目を指定します。

[Username] を選択して Enter キーを押します。

```
How do you want users to be able to sign in? (Use arrow keys)  
➤ Username  
  Email  
  Phone Number  
  Email or Phone Number  
  I want to learn more.
```

4. 詳細な設定を行うかどうか聞かれます。

今回は標準の設定を用いますので、[No, I am done.] を選択して Enter キーを押します。

```
Do you want to configure advanced settings? (Use arrow keys)  
➤ No, I am done.  
  Yes, I want to make some additional changes.
```

- リソースの追加が正常に行われると、以下のメッセージが表示されます。

```
Successfully added auth resource 20200901amplifyXXXXXXX locally
```

### 5.3.2. API リソースの更新

「Auth」リソースによる認証機能を追加しましたので、前節で作成した「API」リソースを認証に対応させる必要があります。

作成済みのリソースの設定を変更するには「amplify update」コマンドを使用します。

1. 以下のコマンドを実行します。

```
$ amplify update api
```

2. 変更対象の API の種類を指定します。

[REST] を選択して Enter キーを押します。

```
? Please select from one of the below mentioned services: (Use arrow keys)
GraphQL
> REST
```

3. 変更対象の API リソースを指定します。

[YYYYMMDDamplify] を選択して Enter キーを押します。

```
? Please select the REST API you would want to update (Use arrow keys)
> 20200901amplify
```

4. 変更の内容を指定します。

既存の API パスの設定を変更しますので、[Update path] を選択して Enter キーを押します。

```
? What would you like to do (Use arrow keys)
Add another path
> Update path
Remove path
```

5. 変更対象の API パスを指定します。

[/items] を選択して Enter キーを押します。

```
? Please select the path you would want to edit (Use arrow keys)
> /items
```

6. 変更後のパス名を指定します。

パス名は変更しないため、何も入力せずに Enter キーを押します。

```
? Provide a path (e.g., /book/{isbn}): (/items)
```

7. Lambda 関数を新たに作成するのか、既存のものを使用するのか指定します。

既存の Lambda 関数を使用するため、[Use a Lambda function already added in the current Amplify project] を選択して Enter キーを押します。

```
? Choose a Lambda source (Use arrow keys)
  Create a new Lambda function
> Use a Lambda function already added in the current Amplify project
```

8. 使用する Lambda 関数を指定します。

[YYYYMMDDamplify] を選択して Enter キーを押します。

```
? Choose the Lambda function to invoke by this path (Use arrow keys)
> 20200901amplify
```

9. API へのアクセスを制限するかどうかを指定します。

Cognito 認証と連係したアクセス制限を行うため、[y] を入力して Enter キーを押します。

```
? Restrict API access (Y/n) y
```

10. 認証されたユーザーのみにアクセスを許可するのか、ゲストユーザーにもアクセスを許可するのかを指定します。

今回は認証されたユーザーのみにアクセスを許可するため、[Authenticated users only] を選択して Enter キーを押します。

```
? Who should have access? (Use arrow keys)
> Authenticated users only
  Authenticated and Guest users
```

11. 認証されたユーザーに対してどの API 操作を許可するのかを指定します。

[create]、[read]、[update]、[delete] の全てを選択状態にして、Enter キーを押します。

```
? What kind of access do you want for Authenticated users? (Press <space> to select, <a> to toggle all, <i> to invert selection)
>● create
  ● read
  ● update
  ● delete
```

※ 複数の項目を選択する場面では、カーソルキーの上下で項目を選択してスペースキーを押すことで選択状態にします（もう一度スペースキーを押すと選択状態が解除されます）  
選択が終わりましたら Enter キーを押します

12. リソースの更新に成功すると、以下のメッセージが表示されます。

```
Successfully updated resource
```

### 5.3.3. Auth リソースおよび API リソースのデプロイ

前節と同様に、Amplify プロジェクト上にリソースを追加した後は、AWS 環境に実際にリソースを構築するために「デプロイ」操作を行います。

また、作成済みの「API」リソースの更新についても、更新内容を AWS 環境に反映するために「デプロイ」操作を行う必要があります。

1. 以下のコマンドを実行します。

```
$ amplify push
```

2. 現在 Amplify プロジェクトに定義されている「リソース」の一覧とステータスが表示されます。

「Auth」リソースの Operation 欄が「Create」と表示されています。

また、「Api」リソースの Operation 欄は「Update」と表示されており、これは「定義が更新されたが、まだデプロイされていない」状態であることを示します。

これらのリソースを AWS 環境へデプロイするために、[y] を入力して Enter キーを押します。

```
✓ Successfully pulled backend environment demo from the cloud.
```

```
Current Environment: demo
```

Category	Resource name	Operation	Provider plugin
Auth	20200901amplifyXXXXXXXX	Create	awscloudformation
Api	20200901amplify	Update	awscloudformation
Storage	20200901amplify	No Change	awscloudformation
Function	20200901amplify	No Change	awscloudformation

```
? Are you sure you want to continue? (Y/n) y
```

3. AWS 環境へのデプロイ処理が始まります。

デプロイ処理が終わるまで数分程度かかりますので、終わるまで待ちます。（画面上では AWS の各リソースを作成する CloudFormation が実行されている状況が確認できると思います）

デプロイ処理が成功すると、以下のメッセージが表示されます。

```
✓ All resources are updated in the cloud
```

4. デプロイによって、以下の AWS リソースが作成されています。

作成された各リソースは既に設定も行われており、特に内容を確認したり変更したりする必要はありませんが、時間のある方は作成されたリソースの内容を確認するとよいでしょう。

- Cognito ユーザープール
  - **YYYYMMDDamplifyXXXXXXX\_userpool\_XXXXXXX-demo**
- Cognito ID プール
  - **YYYYMMDDamplifyXXXXXXX\_identitypool\_XXXXXXX\_\_demo**
- IAM ロール
  - snsXXXXXXXXXXXXX-demo
  - upClientLambdaRoleXXXXXXXXXXXXX-demo
- Lambda 関数
  - amplify-**YYYYMMDDamplify-demo-UserPoolClientLambda-XXXXXXX**

また、以下の既存の AWS リソースに対して変更が行われています。

大きな変更点は「API Gateway REST API」に対して「AWS\_IAM」の認可設定が追加されることです。

- API Gateway REST API
  - **YYYYMMDDamplify**
- Lambda 関数
  - **YYYYMMDDamplify-demo**
- IAM ロール（認証されたユーザーに割り当てられる IAM ロール）
  - amplify-**YYYYMMDDamplify-demo-XXXXX-authRole**

「認証されたユーザーに割り当てられる IAM ロール」については、ここまで説明していませんでしたが、実は最初の「Amplify プロジェクトの初期化 (amplify init)」の際に既に作成されています。（同時に「認証されていないユーザーに割り当てられる IAM ロール」も作成されています）「API」リソースに対してアクセス制限を設定したことにより、これらの IAM ロールに対して API Gateway REST API の呼び出し許可を与えるポリシーが登録されています。

### 5.3.4. Web クライアントアプリケーションの構成

- API を認証に対応させるために、JavaScript ファイル **[App.js]** の内容を修正します。Cloud9 のエディタ、または vi コマンドを使って、**[src]** ディレクトリ直下の **[App.js]** ファイルを開きます。
- 修正箇所は 3 点あります。

まず、ファイル冒頭の「import」の記述群に以下の行を追加します。

```
import React from 'react';
import Amplify from 'aws-amplify';
import awsconfig from './aws-exports';
import { withAuthenticator, AmplifySignOut } from '@aws-amplify/ui-react';
```

この行によって、認証を行う 2 つのコンポーネントをインポートします。

- 次に、ファイルの 16 行目付近にある「<div className="App">」の行の次に、以下の行を追加します。

```
class App extends React.Component {
  render() {
    return (
      <div className="App">
        <AmplifySignOut />
        <header className="App-header">
```

この行によって、画面に「サインアウト」ボタンのコンポーネントを配置します。

- 最後に、ファイルの最終行にある以下の行を、下記のように修正します。

```
export default App;
```

↓

```
export default withAuthenticator(App);
```

この行によって、アプリケーションに「サインアップ」「サインイン」などの認証機能を提供するコンポーネントを適用します。

全ての修正が終わりましたら、ファイルを保存します。

### 5.3.5. Web クライアントアプリケーションの動作確認

- 以下のコマンドを実行します。

```
$ npm start
```

- アプリケーションが起動すると、以下のように表示されます。（数分間かかりますので待ちます）

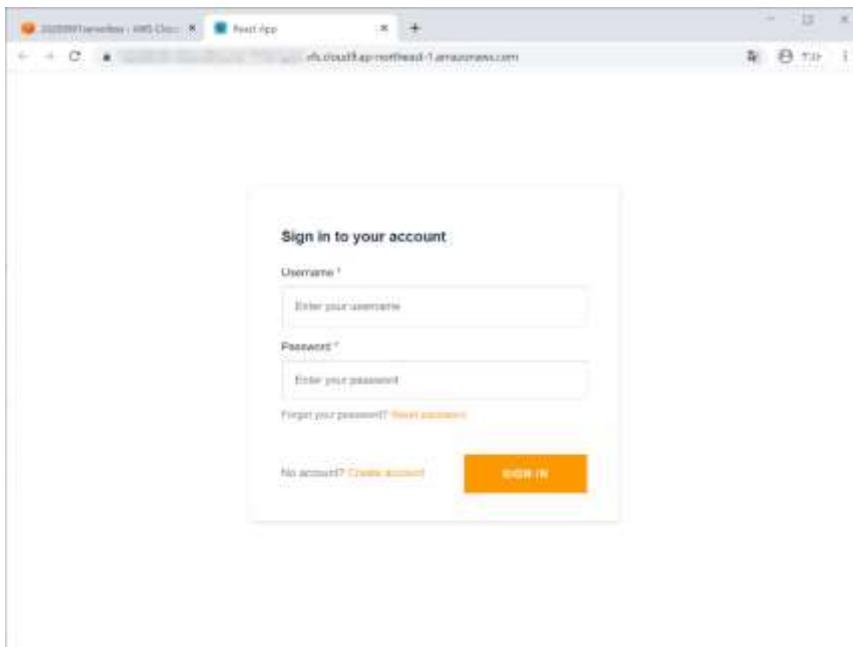
```
Compiled successfully!

You can now view 20200901amplify in the browser.

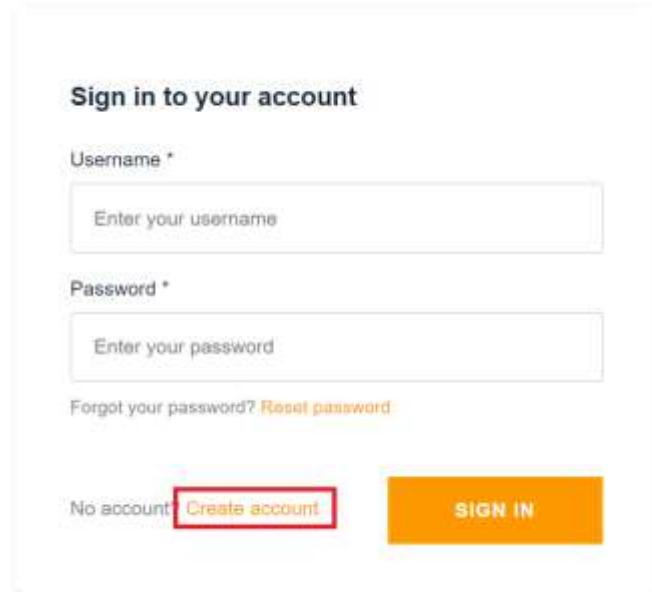
Local:          http://localhost:8080
On Your Network:  http://172.XX.XX:8080

Note that the development build is not optimized.
To create a production build, use npm run build.
```

- 画面上部の [Preview] をクリックして [Preview Running Application] を選択します。
- 下図のように Web クライアントアプリケーションが正常に表示されることを確認します。

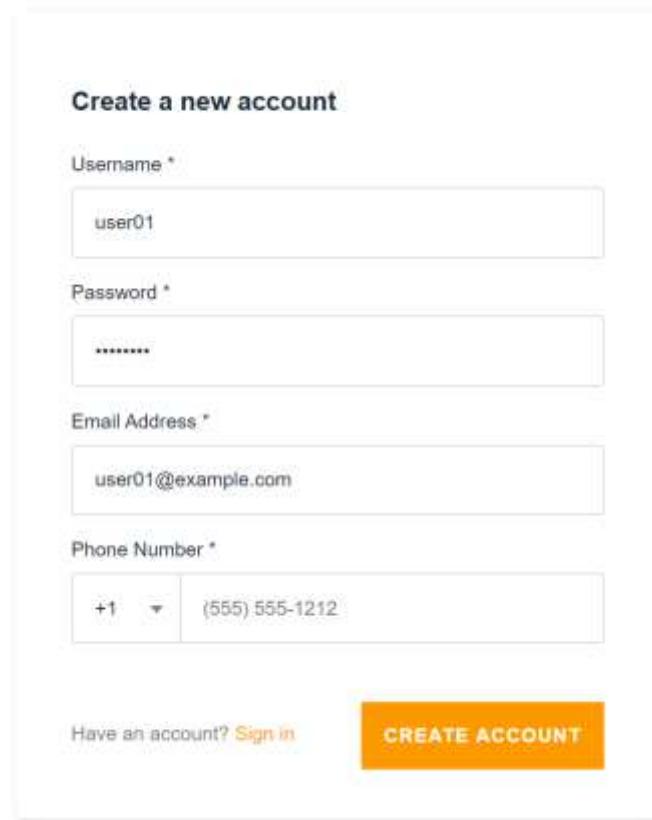


5. まず、[Create account] をクリックしてサインアップを行います。



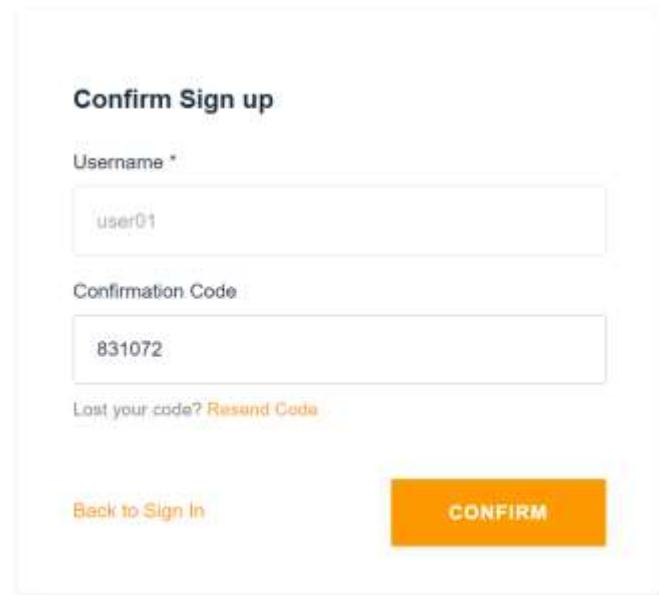
The image shows a sign-in form titled "Sign in to your account". It includes fields for "Username \*" and "Password \*", both with placeholder text "Enter your username" and "Enter your password" respectively. Below these fields is a link "Forgot your password? [Reset password](#)". At the bottom left is a link "No account? [Create account](#)" and at the bottom right is a large orange button labeled "SIGN IN".

6. ユーザー名、パスワード、メールアドレスを入力して [CREATE ACCOUNT] をクリックします。 (電話番号の欄は未入力で構いません)



The image shows a create account form titled "Create a new account". It includes fields for "Username \*" (with value "user01"), "Password \*" (with placeholder "....."), "Email Address \*" (with value "user01@example.com"), and "Phone Number \*" (with value "+1 (555) 555-1212"). At the bottom left is a link "Have an account? [Sign in](#)" and at the bottom right is a large orange button labeled "CREATE ACCOUNT".

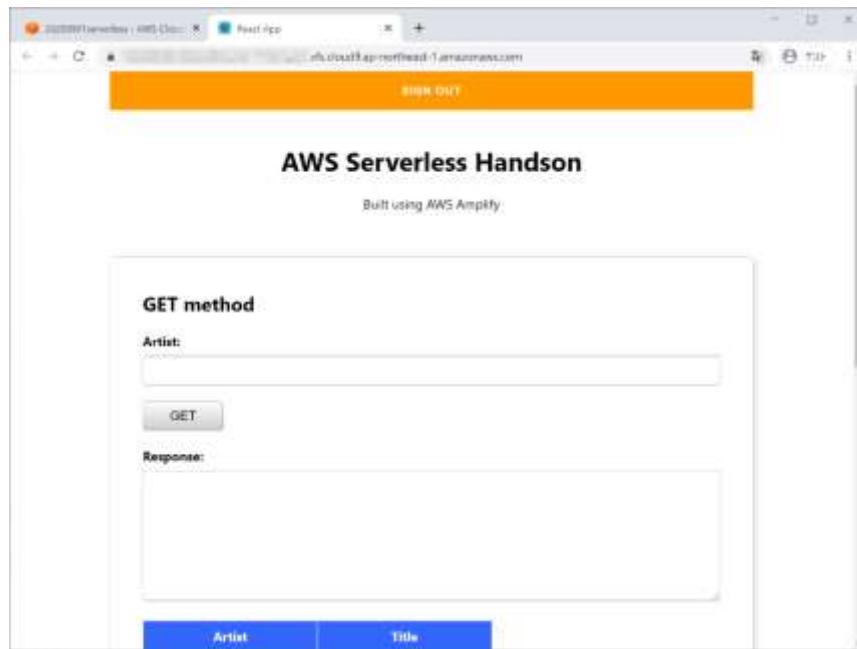
7. メールで確認コードが送られてきますので、入力して **[CONFIRM]** をクリックします。



The screenshot shows a 'Confirm Sign up' form. It has two input fields: 'Username \*' containing 'User01' and 'Confirmation Code' containing '831072'. Below the fields is a link 'Lost your code? [Resend Code](#)'. At the bottom are two buttons: 'Back to Sign In' (orange) and a large orange 'CONFIRM' button.

8. 自動的にサインインが行われて、API テストの画面が表示されます。

「GET メソッド」 「POST メソッド」 が実行可能なことを確認します。



9. 画面上部の **[SIGN OUT]** をクリックすると、サインアウトが行われ、最初のサインイン画面に戻ります。

このように、JavaScript に 3 行ほど追加するのみで、サインアップ・サインイン等のフォームやコードを記述することなく、アプリケーションに認証機能を追加することができました。

今回は「認証されたユーザー」のみに API のアクセス権限を与えていましたが、ラボ 1 のように「認証されたユーザー」「認証されていないユーザー」それぞれに異なるアクセス権限を与えることも可能です。

その場合は、今回より少し複雑なコードを記述する必要があります。

## 5.4. Amplify アプリケーションをホスティングする

ここまで手順では、作成した Amplify アプリケーションをローカル環境（Cloud9）で実行してきましたが、本番運用では公開 Web サイトとして公開する必要があります。

「Amplify Console」のマネージドなホスティング機能を使うと、S3 静的ウェブホスティングや CloudFront といったホスティング環境を自分で用意することなく、簡単に Amplify アプリケーションを公開することができます。

（なお、Amplify Console にはホスティング機能の他にも、GitHub や AWS CodeCommit と統合可能な CI/DI 機能がありますが、今回は利用しません）

### 5.4.1. ホスティングの設定

0. Ctl + c で停止します。cd でディレクトリを environment から yyyyymmddamplify に移動します。
1. 以下のコマンドを実行します。

```
$ amplify add hosting
```

2. ホスティングの方式を指定します。

今回は Amplify Console が提供するマネージドなホスティング機能を利用しますので、

[Hosting with Amplify Console] を選択して Enter キーを押します。

```
? Select the plugin module to execute (Use arrow keys)
> Hosting with Amplify Console (Managed hosting with custom domains,
  Continuous deployment)
  Amazon CloudFront and S3
```

3. CI/CD の方式を指定します。

今回は Git ベースの CI/CD 機能を利用しませんので、[Manual deployment] を選択して Enter キーを押します。

```
? Choose a type (Use arrow keys)
  Continuous deployment (Git-based deployments)
> Manual deployment
  Learn more
```

4. ホスティングの設定が正常に行われると、以下のメッセージが表示されます。

```
You can now publish your app using the following command:
```

```
Command: amplify publish
```

## 5.4.2. アプリケーションの公開

- 以下のコマンドを実行します。

```
$ amplify publish
```

- リソースの「デプロイ」を行う時と同様に、以下のような画面が表示されます。

ホスティングの設定は「リソース」として扱われており、Operation 欄が「Create」と表示されています。

ホスティング環境を公開するために、[y] を入力して Enter キーを押します。

```
✓ Successfully pulled backend environment demo from the cloud.
```

```
Current Environment: demo
```

Category	Resource name	Operation	Provider plugin
Hosting	amplifyhosting	Create	awscloudformation
Storage	20200901amplify	No Change	awscloudformation
Function	20200901amplify	No Change	awscloudformation
Api	20200901amplify	No Change	awscloudformation
Auth	20200901amplifyXXXXXXXXX	No Change	awscloudformation

```
? Are you sure you want to continue? (Y/n) y
```

- ホスティング環境を公開する処理が始まります。

処理が終わるまで数分程度かかりますので、終わるまで待ちます。

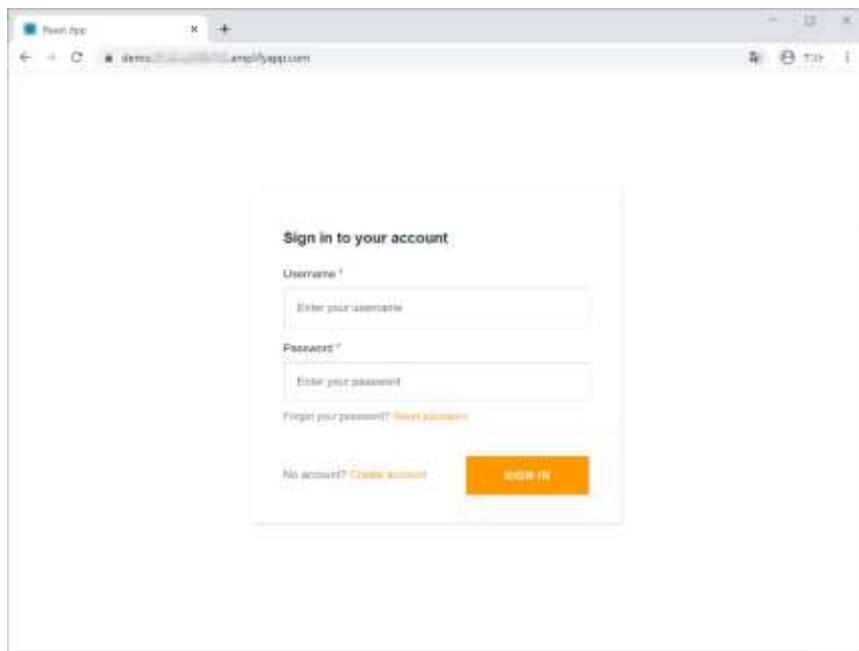
処理が成功すると、以下のメッセージが表示されます。

公開サイトの URL が表示されていることを確認します。

```
✓ Deployment complete!
```

```
https://demo.XXXXXXXXXXXXXXX.amplifyapp.com
```

4. PC の Web ブラウザで、表示された公開サイトの URL へアクセスします。  
アプリケーションの画面が表示されることを確認します。  
また、サインインや API の操作を行い、ローカル環境と同様に操作できることを確認します。



# 6. 後片付け

下記、後片付けを行います。

この作業まで完了していない場合、継続して課金が発生しますので、必ず実施してください。

- 4. ラボ 1 : サーバーレスアプリケーションを step-by-step で構築する
  - Cognito ID プールの削除
    - ✧ [YYYYMMDDserverless]
  - Cognito ユーザープールの削除
    - ✧ [YYYYMMDDserverless]
  - API Gateway (API) の削除
    - ✧ [YYYYMMDDserverless]
  - Lambda 関数の削除
    - ✧ [YYYYMMDDserverlessRead]
    - ✧ [YYYYMMDDserverlessWrite]
  - DynamoDB テーブルの削除
    - ✧ [YYYYMMDDserverless]
  - IAM ロールの削除
    - ✧ [YYYYMMDDserverlessLambdaRole]
    - ✧ [Cognito\_YYYYMMDDserverlessAuth\_Role]
    - ✧ [Cognito\_YYYYMMDDserverlessUnauth\_Role]
  - IAM ポリシーの削除
    - ✧ [YYYYMMDDserverlessLambdaPolicy]
    - ✧ [YYYYMMDDserverlessAPIGatewayGetAndPostPolicy]
    - ✧ [YYYYMMDDserverlessAPIGatewayGetOnlyPolicy]
- 5. ラボ 2 : Amplify を使ってサーバーレスアプリケーションを構築する
  - Amplify プロジェクトを削除するコマンドを実行することで、AWS 環境にデプロイしたリソースを全て削除することができます。

Amplify プロジェクトのディレクトリに移動して、以下のコマンドを実行します。

```
$ amplify delete
```

- Cloud9 環境の削除
- ✧ [YYYYMMDDserverless]

後片付けは以上です。

