

[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.....	168

# 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ちらかの環境で作成することができます。作業は特に指定されない限りは東京うリージョンを使ってください。

コメントの追加【青柳1】：CloudFormationを使用していないためリージョンの制約は無いと思いますが、ある程度限定しておいた方がよいかもしれません。

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。

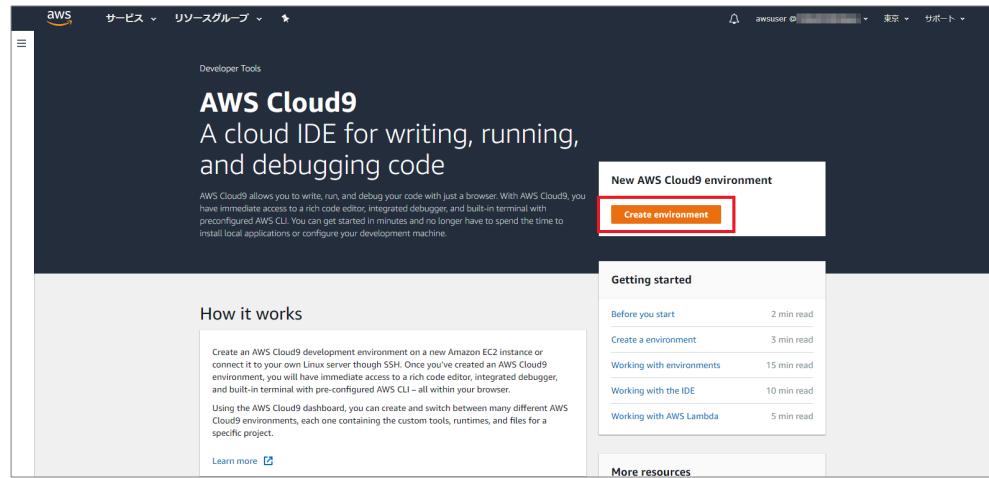
#### 3.1. Cloud9 環境の作成

##### 3.1.1. AWS マネジメントコンソールへのログイン

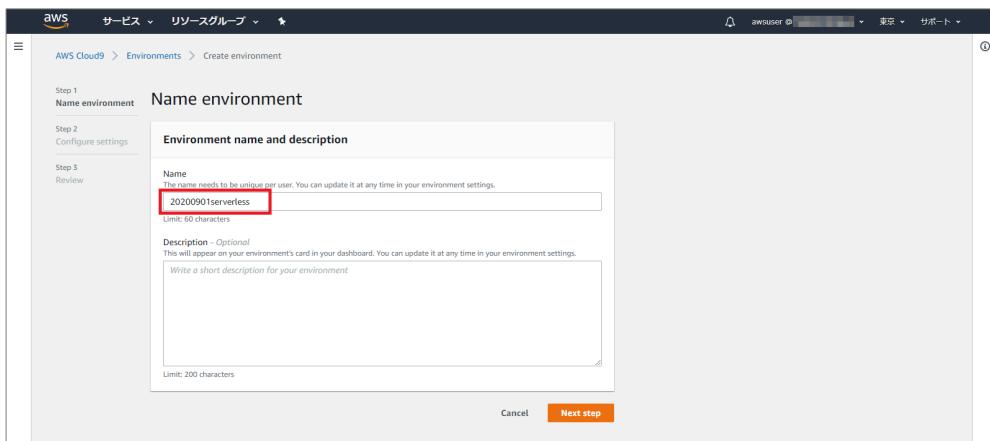
1. AWS マネジメントコンソールにログインします。
2. ログイン後、画面右上のヘッダー部のリージョン選択にて、**利用を指示されたリージョン** となっていることを確認します。  
注：[東京]、[バージニア]、[オレゴン]、[シンガポール] のどれかになります。

##### 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)



The screenshot shows the 'Name environment' step of the AWS Cloud9 'Create environment' wizard. The 'Name' field is highlighted with a red border and contains the text '20200901serverless'. The 'Description' field is empty. At the bottom, there are 'Cancel' and 'Next step' buttons.

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:** [t2.micro]
- **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)

aws サービス リソースグループ 東京 サポート

AWS Cloud9 > Environments > Create environment

Step 1 Name environment Step 2 Configure settings Step 3 Review

**Configure settings**

**Environment settings**

**Environment type** Info  
Run your environment in a new EC2 instance or an existing server. With EC2 Instances, you can connect directly through Secure Shell (SSH) or connect via AWS Systems Manager (without opening inbound ports).

Create a new EC2 instance for environment (direct access)  
Launch a new instance in this region that your environment can access directly via SSH.

Create a new no-ingress EC2 instance for environment (access via Systems Manager)  
Launch a new instance in this region that your environment can access through Systems Manager.

Create and run in remote server (SSH connection)  
Configure the secure connection to the remote server for your environment.

**Instance type**  
 t2.micro (1 GB RAM + 1 vCPU)  
Free-tier eligible. Ideal for educational users and exploration.  
 t2.small (2 GB RAM + 2 vCPU)  
Recommended for small-scale web projects.  
 m5.large (8 GB RAM + 2 vCPU)  
Recommended for production and general-purpose development.  
 Other instance type  
Select an instance type:  
t3.nano

**Platform**  
 Amazon Linux  
 Ubuntu Server 18.04 LTS

**Cost-saving setting**  
Choose a predetermined amount of time to auto-hibernate your environment and prevent unnecessary charges. We recommend a minimum setting of half an hour of inactivity to maximize savings.  
After 30 minutes (default)

**IAM role**  
AWS Cloud9 creates a service-linked role for you. This allows AWS Cloud9 to call other AWS services on your behalf. You can delete the role from the AWS IAM console once you no longer have any AWS Cloud9 environments. [Learn more](#)

AWSServiceRoleForAWSCloud9

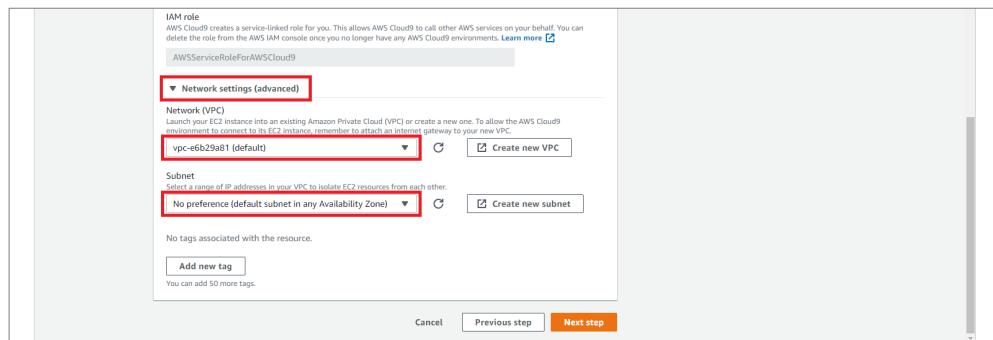
▶ Network settings (advanced)

No tags associated with the resource.  
Add new tag  
You can add 50 more tags.

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

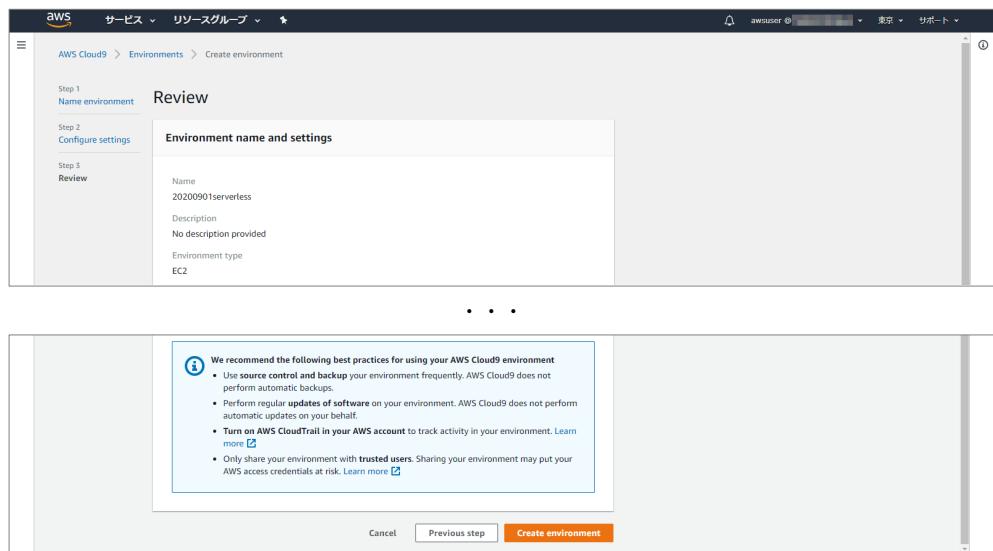
Cloud9 環境を構築する VPC および サブネット を指定できますので、任意の VPC を指定してください。サブネットは「パブリックサブネット」であるものを指定してください。

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。

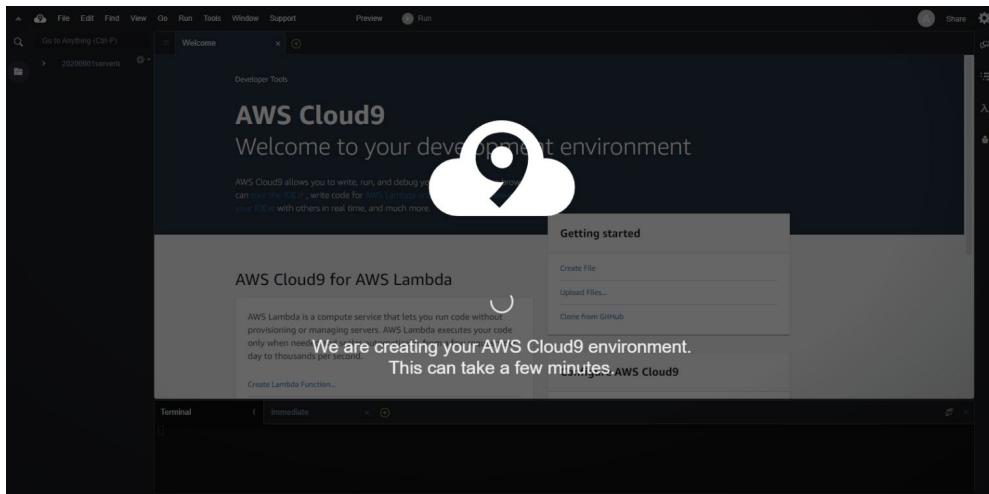


7. [Next Step] をクリックします。

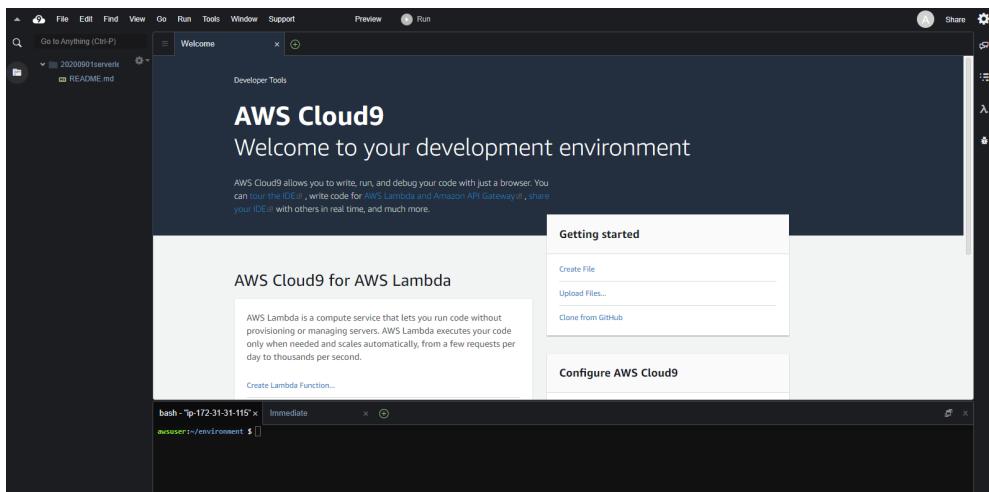
8. 確認画面になりますので、[Create environment] をクリックします。



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。



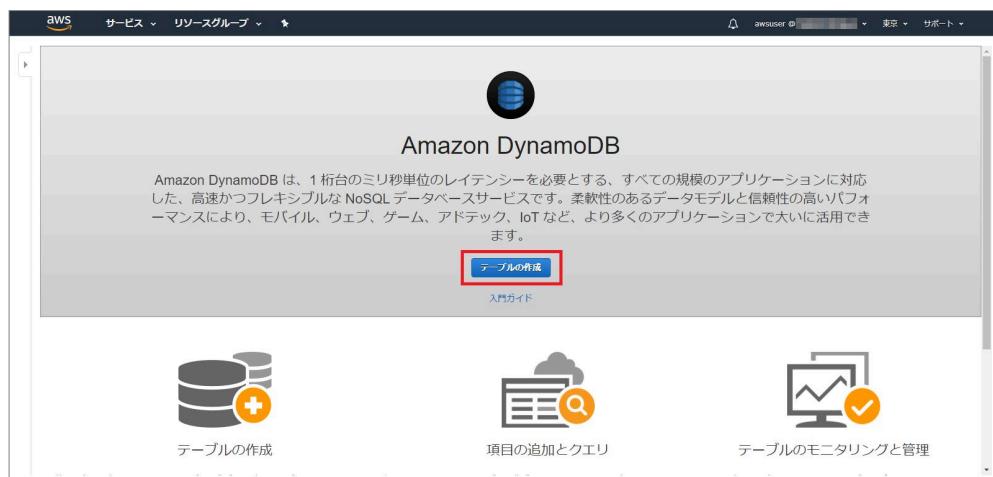
## 4. サーバーレスアプリケーションを step-by-step で構築する

### 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

DynamoDB テーブルの作成

テーブル名: 20200901serverless

プライマリキー: パーティションキー  
Artist 文字列

ソートキーの追加

ソートキー: Title 文字列

テーブル設定

デフォルト設定は、最も手軽にテーブルの使用を開始する手段です。これらのデフォルト設定は、今すぐまたはテーブルの作成後に変更できます。

デフォルト設定の使用

- セカンダリインデックスなし。
- Auto Scaling のキヤバシティーを AnyScale のデフォルト機能を示すテーブルの作成ページで 70% のターゲット使用率 (最低キヤバシティーは 5つの読み込みと 5つの書き込み) に設定。
- デフォルトの暗号化タイプによる保護時の暗号化。

+ タグの追加 NEW!

CloudWatch または Simple Notification Service で AWS 料料利用枠の範囲を超えた場合は、追加の課金が適用される可能性があります。再度なアラーム設定は CloudWatch マネジメントコンソールにて設定可能です。

キャンセル 作成

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ましたら、【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。

20200901serverless 閉じる

項目

スキーマ: [テーブル] 20200901serverless: Artist, Title

スキャナ: [テーブル] 20200901serverless: Artist, Title

Artist > Title

1つの項目は 1 つ以上の属性で構成されます。各属性は名前、データ型、値で構成されます。項目の読み込みまたは書き込み時、必須の属性はプライマリキーを構成する属性のみです。 詳細

#### 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 service dashboard. The left sidebar has a 'Policies' section highlighted with a red box. The main area displays 'Identity and Access Management へようこそ' and 'IAM リソース' information: 2 users, 0 groups, 64 roles, and 0 ID providers. Below this is a 'セキュリティステータス' section with several items, some of which have orange warning icons.

2. **[ポリシーの作成]** をクリックします。

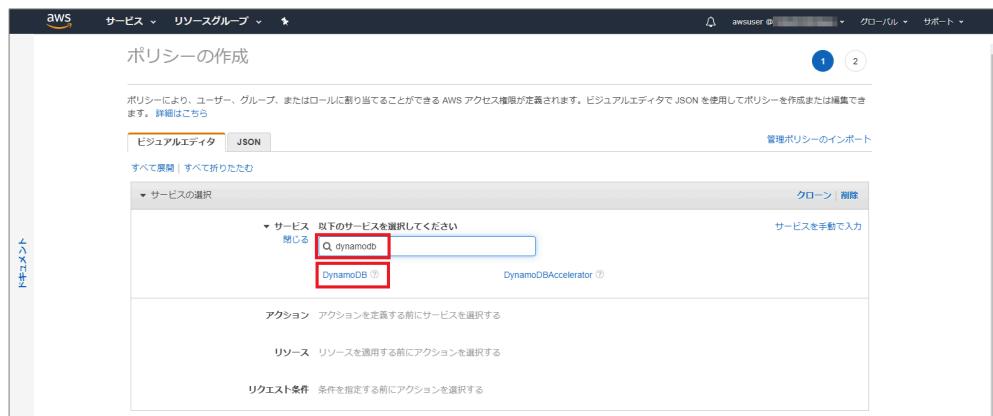
The screenshot shows the 'Create New Policy' wizard. Step 1 is 'Set Policy Name'. The 'Policy Name' input field contains 'LambdaPolicy'. The 'Type' dropdown is set to 'AWSによる管理'. The 'Next Step' button is visible at the bottom right.

3. [サービス] をクリックして展開します。



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。



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  
閉じる  すべてのリソース

backup ⑦	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 ⑦	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 ⑦	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 ⑦	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 ⑦	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)

ポリシーの確認

名前: 20200901serverlessLambdaPolicy  
英数字と「\_」を使用します。最大 128 文字。

説明

最大 1000 文字。英数字と「\_」を使用します。

概要

このポリシーはアクセス許可を提供しないいくつかのアクション、リソース、条件が定義されています。アクセス権限を付与するには、ポリシーに該当するリソースまたは条件を持つアクションがある必要があります。詳細については、[権利の表示](#)、[詳細はこちら](#)を選択します。

サービス	アクセスレベル	リソース	リクエスト条件
許可 (239 サービス中 1) 残りの 238 を表示	DynamoDB	完全:書き込み:制限:読み込み	複数

\* 必須

キャンセル 戻る ポリシーの作成

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。

Identity and Access Management (IAM)

ポリシー名: 20200901serverlessLambdaPolicy

ポリシー名	タイプ	次として使用	説明
20200901serverlessLambdaPolicy	ユーザーによる管理	なし	ユーザーによる管理
AccessAnalyzerServiceRolePolicy	AWS による管理	なし	Allow Access Analyzer to analyze resource metadata
AdministratorAccess	ショットカット	Permissions policy (3)	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 AW...
AlexaForBusinessGatewayExecution	AWS による管理	なし	Provide gateway execution access to AlexaForBusiness services
AlexaForBusinessLifecycleDeleg...	AWS による管理	なし	Provide access to Lifesize AVS devices
AlexaForBusinessNetworkProfile...	AWS による管理	なし	This policy enables Alexa for Business to perform automated tasks schedule...

#### 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. The left sidebar has 'Identity and Access Management (IAM)' selected. Under 'Roles', the 'Roles' link is highlighted with a red box. The main area displays 'IAM リソース' (Resources) with sections for 'ユーザー' (Users), 'グループ' (Groups), and 'カスタマーマネジメントポリシー' (Customer-managed policies). Below this is a 'セキュリティステータス' (Security status) section with several items, some of which have yellow warning icons.

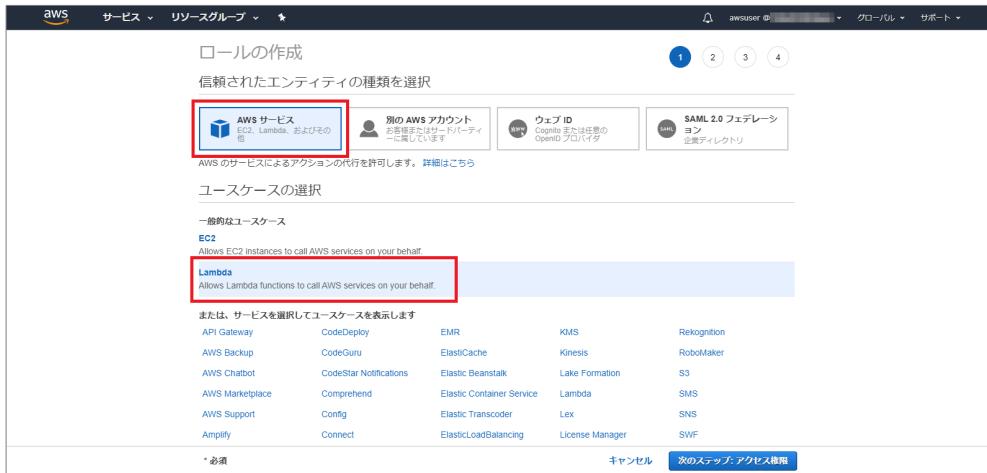
2. [ロールの作成] をクリックします。

The screenshot shows the 'Roles' page in the AWS IAM console. The 'Create New Role' button is highlighted with a red box. The table below lists existing roles, including their names, service principals, and last activity times. A search bar at the top is also visible.

Role Name	Assumed by Entity	Last Activity
AmazonEKSFargatePodExecutionRole	AWS Service: eks-fargate-pods	260 days ago
AmazonPersonalize-ExecutionRole-1562387868689	AWS Service: personalize	None
AWSDeepRacerCloudFormationAccessRole	AWS Service: cloudformation	None
AWSDeepRacerLambdaAccessRole	AWS Service: lambda	None
AWSDeepRacerRoboMakerAccessRole	AWS Service: robomaker	None
AWSDeepRacerSageMakerAccessRole	AWS Service: sagemaker	None

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。

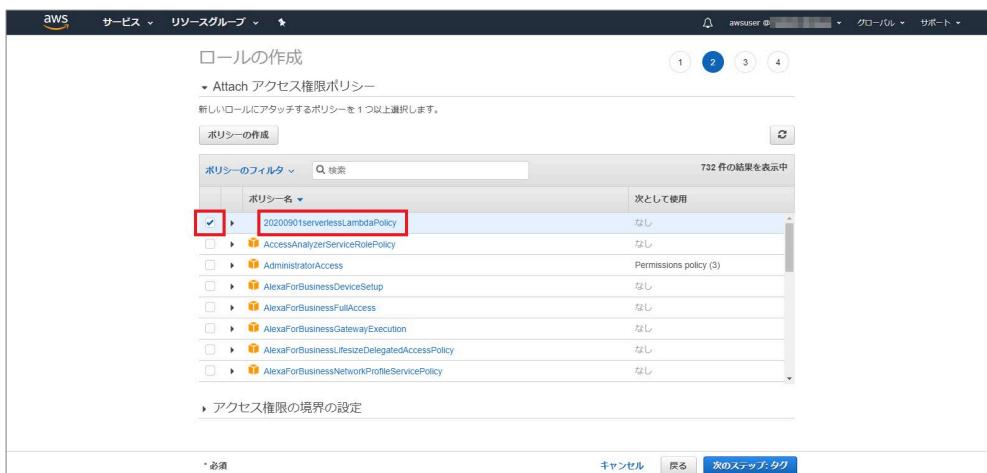
[ユースケースの選択] で [Lambda] をクリックして選択状態にします。



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] 欄欄に [**YYYYMMDDserverlessLambdaRole**] と入力します。 (YYYYMMDD は本日の日付)

The screenshot shows the 'Role creation' step of the AWS Lambda role creation wizard. The role name is set to '20200901serverlessLambdaRole'. The role description is 'Allows Lambda functions to call AWS services on your behalf.' The policy selected is '20200901serverlessLambdaPolicy'. There are no tags added. The bottom right button is 'Create role'.

9. [ロールの作成] をクリックします。

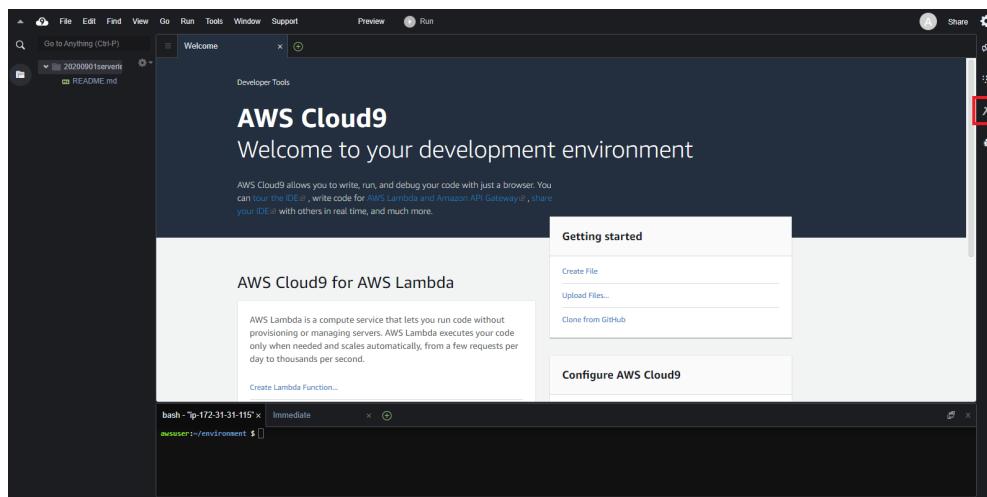
10. ロール [**YYYYMMDDserverlessLambdaRole**] が作成されたことを確認します。

ロール名	信頼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
AmazonEKSFargatePodExecutionRole	AWS サービス: eks-fargate-pods	280 日間
AmazonPersonalize-ExecutionRole-156238768689	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
AWSDeepRacerRoboMakerAccessRole	AWS サービス: robomaker	なし
AWSDeepRacerSageMakerAccessRole	AWS サービス: sagemaker	なし
AWSDeepRacerServiceRole	AWS サービス: deepracer	29 日間

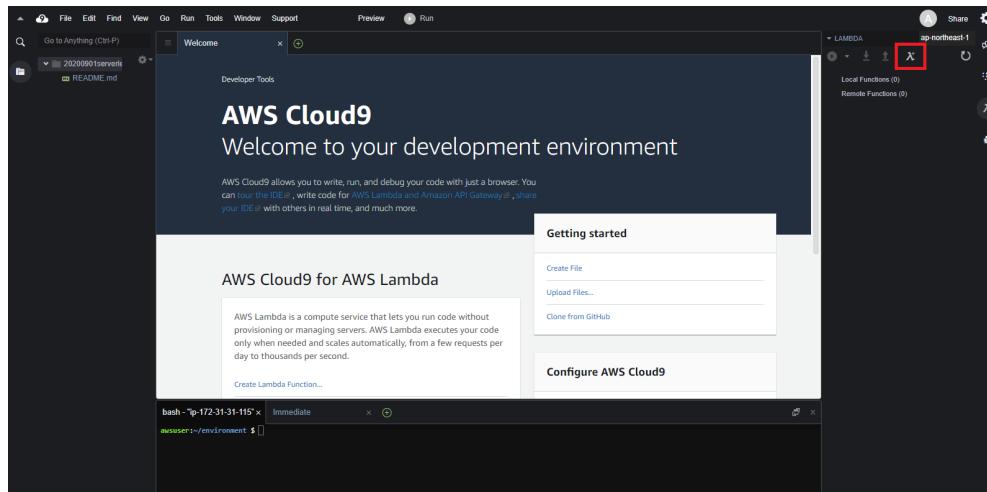
#### 4.1.4. Cloud9 上での Lambda 関数の作成とテスト (Write 関数)

##### 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 Resourcse]を選んでください。

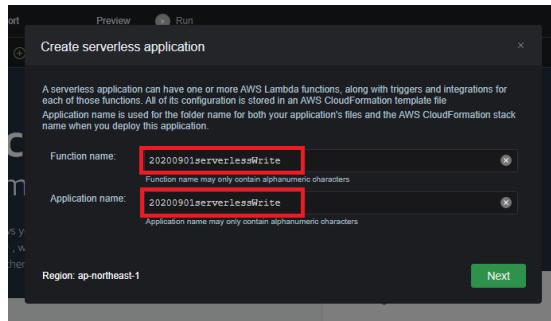


##### 2. [Create] ボタン (Lambda アイコンに[+]印が付いたもの) をクリックします。



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

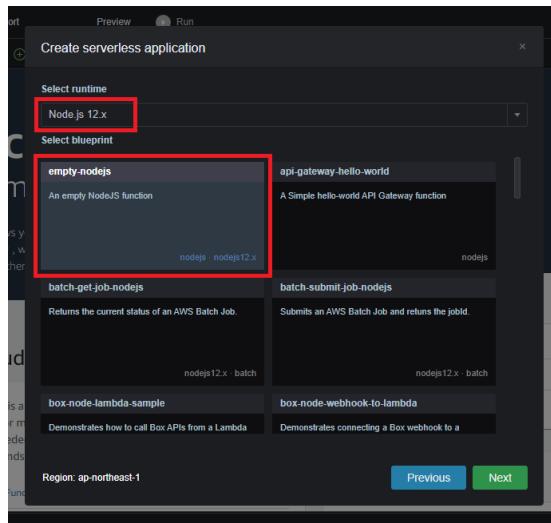
[Application name] 欄は [Function name] 欄と同じ内容が自動的に入力されますので、そのままにします。



4. [Next] をクリックします。

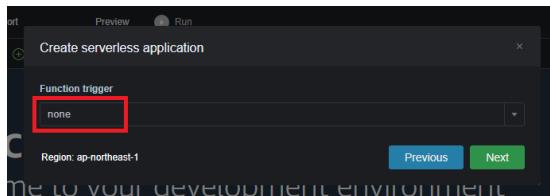
5. [Select runtime] で [Node.js 12.x] を選択します。

[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。

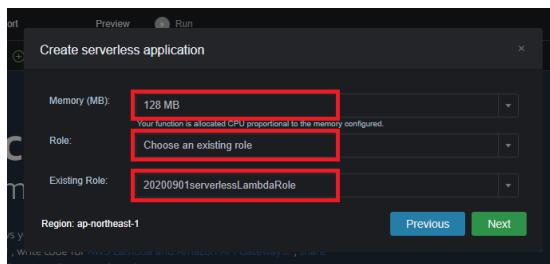
7. [Function trigger] で [none] を選択します。



8. [Next] をクリックします。

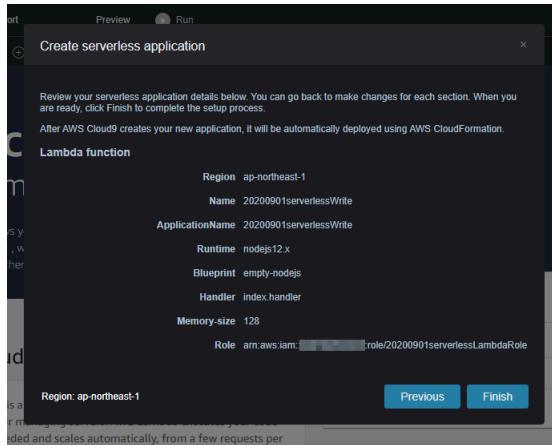
9. 以下の通りに選択します。

- **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

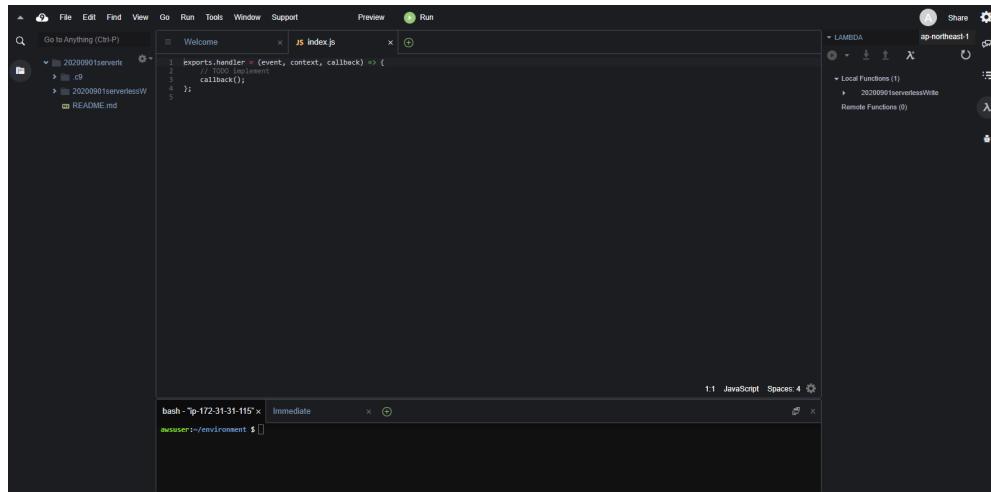


10. [Next] をクリックします。

11. 確認画面になりますので、[Finish] をクリックします。

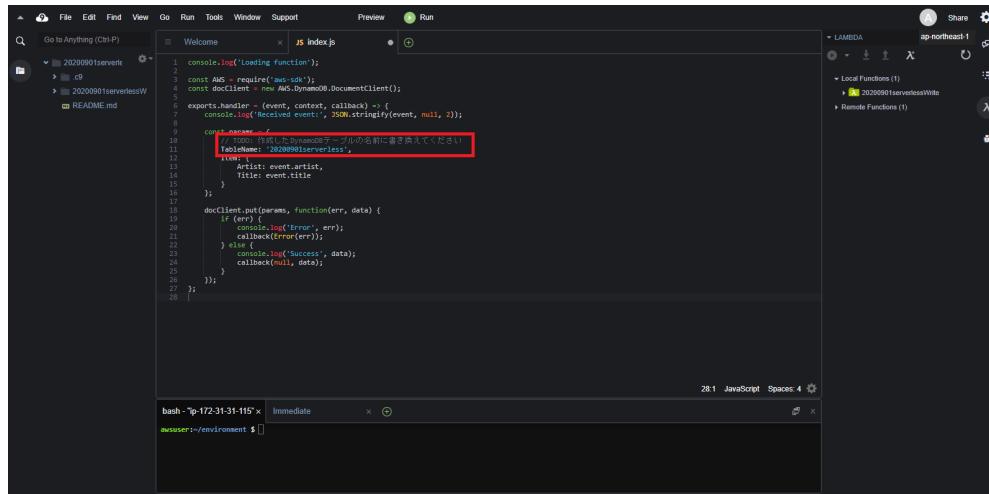


12. Lambda 関数コード (index.js) の編集画面が表示されます。



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。



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

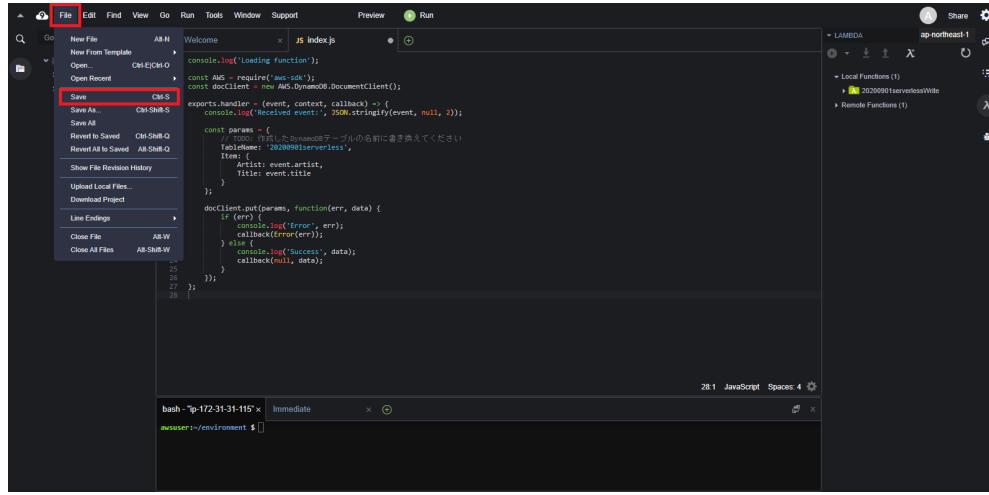
const AWS = require('aws-sdk');
const docClient = new AWS.DynamoDB.DocumentClient();

exports.handler = (event, context, callback) => {
    console.log('Received event:', JSON.stringify(event, null, 2));

    const params = {
        // TODO: 作成したDynamoDBテーブルの名前に書き換えてください!
        TableName: '20200901serverless',
        Item: {
            artist: event.artist,
            title: event.title
        }
    };

    docClient.put(params, function(err, data) {
        if (err) {
            console.log('Error', err);
            callback(err);
        } else {
            console.log('Success', data);
            callback(null, data);
        }
    });
};
```

14. コードの編集が終わりましたら、画面上部のメニューから **[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。

```

File Edit Find View Go Run Tools Window Support Preview Run
Q Go to Anything (Ctrl-P)
20200901serverless
  c9
  20200901serverlessW
README.md

Welcome JS index.js
1 console.log('Loading function');
2
3 const AWS = require('aws-sdk');
4 const docClient = new AWS.DynamoDB.DocumentClient();
5
6 exports.handler = (event, context, callback) => {
7   console.log('Received event:', JSON.stringify(event, null, 2));
8
9   const params = {
10     TableName: '20200901serverless',
11     Item: {
12       Artist: event.artist,
13       Title: event.title
14     }
15   };
16
17   docClient.put(params, function(err, data) {
18     if (err) {
19       console.log('Error', err);
20       callback(err);
21     } else {
22       console.log('Success', data);
23       callback(null, data);
24     }
25   });
26 };
27
28

```

bash - ip-172-31-31-115 ~ Immediate awsuser:~/environment \$

## 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。

```

File Edit Find View Go Run Tools Window Support Preview Run
Q Go to Anything (Ctrl-P)
20200901serverless
  c9
  20200901serverlessW
README.md

[A] 20200901serverlessW Lambda (local)

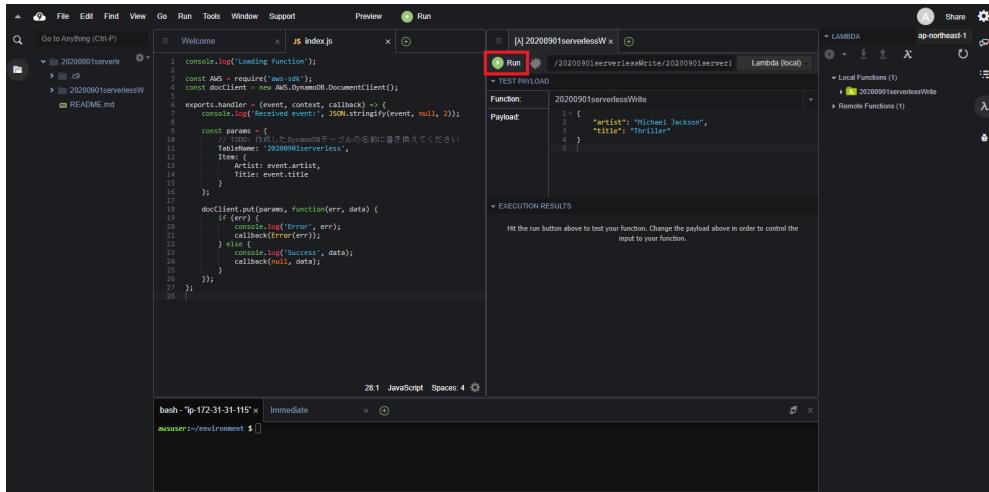
Welcome JS index.js
1 console.log('Loading function');
2
3 const AWS = require('aws-sdk');
4 const docClient = new AWS.DynamoDB.DocumentClient();
5
6 exports.handler = (event, context, callback) => {
7   console.log('Received event:', JSON.stringify(event, null, 2));
8
9   const params = {
10     TableName: '20200901serverless',
11     Item: {
12       Artist: event.artist,
13       Title: event.title
14     }
15   };
16
17   docClient.put(params, function(err, data) {
18     if (err) {
19       console.log('Error', err);
20       callback(err);
21     } else {
22       console.log('Success', data);
23       callback(null, data);
24     }
25   });
26 };
27
28

Function: 20200901serverlessWrite
Payload: [
  {
    "Artist": "Michael Jackson",
    "Title": "Thriller"
  }
]

EXECUTION RESULTS
Hit the run button above to test your function. Change the payload above in order to control the input to your function.

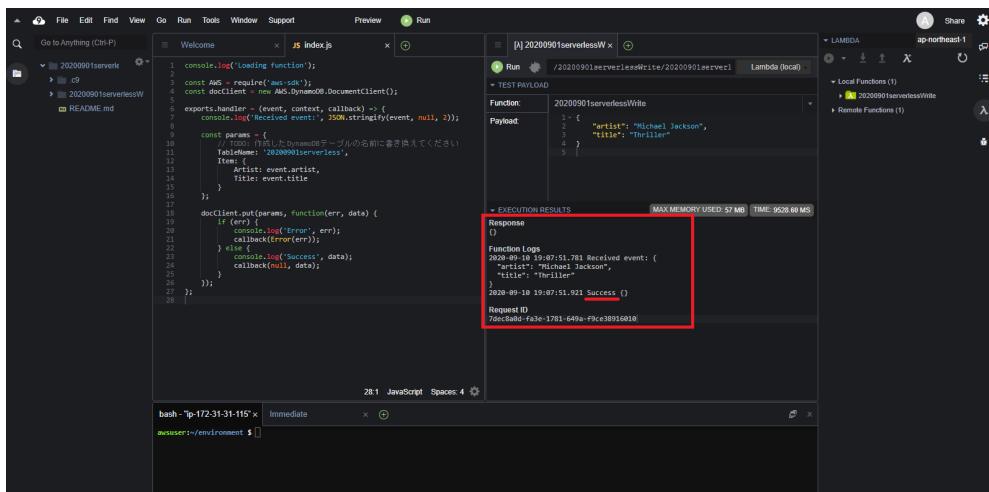
bash - ip-172-31-31-115 ~ Immediate awsuser:~/environment $
```

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。

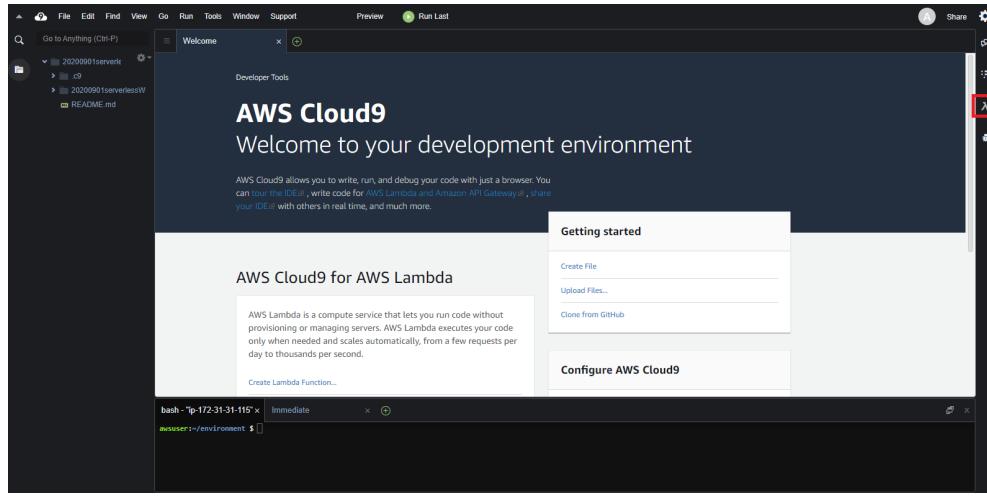
リロードボタンをクリックして、Lambda 関数によって書き込まれたデータが表示されていることを確認します。

The screenshot shows the AWS DynamoDB console interface. On the left, the navigation menu is visible with options like 'DynamoDB', 'DAX', and 'Events'. The main area displays the '20200901serverless' table. The top navigation bar has tabs for '概要', '項目' (Item), 'メトリックス', 'アラーム', 'キャパシティ', 'インデックス', 'グローバルテーブル', 'バックアップ', '投稿者のインサイト', and 'さらに'. The '項目' tab is selected. Below the tabs, there are filters for 'スキャン' and '(テーブル) 20200901serverless: Artist, Title'. A red box highlights the '再読み込み' (Reload) button in the top right corner of the item list header. The table itself shows one item: Michael Jackson with the title Thriller.

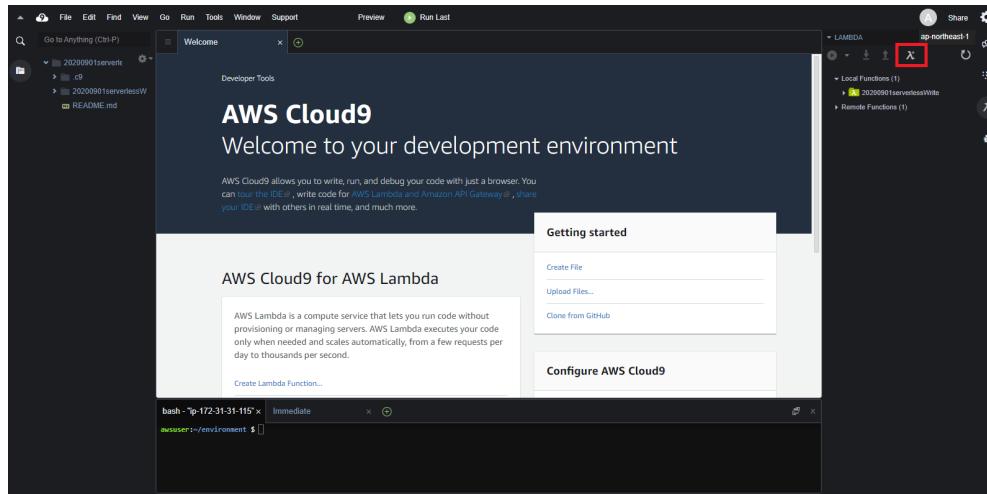
Artist	Title
Michael Jackson	Thriller

#### 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。

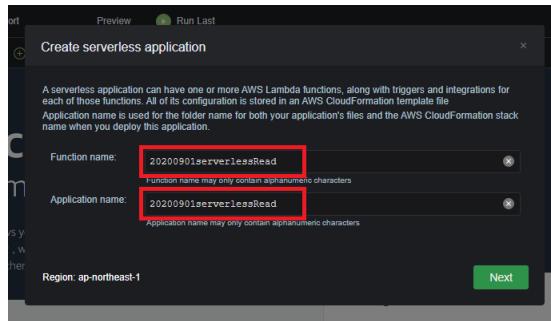


2. [Create] ボタン (Lambda アイコンに[+]印が付いたもの) をクリックします。



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

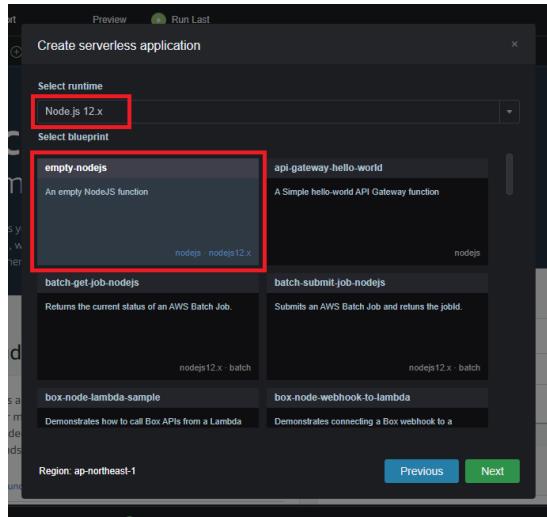
[Application name] 欄は [Function name] 欄と同じ内容が自動的に入力されますので、そのままにします。



4. [Next] をクリックします。

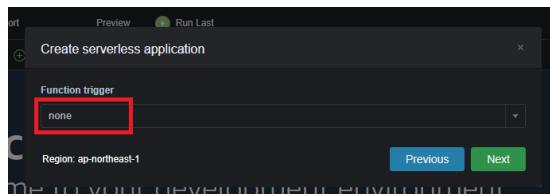
5. [Select runtime] で [Node.js 12.x] を選択します。

[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。

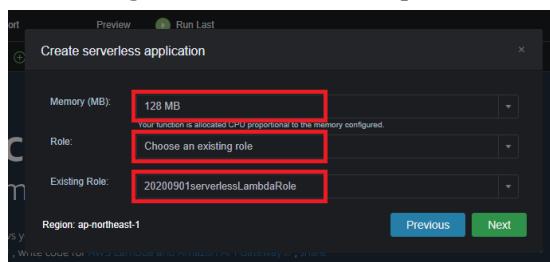
7. [Function trigger] で [none] を選択します。



8. [Next] をクリックします。

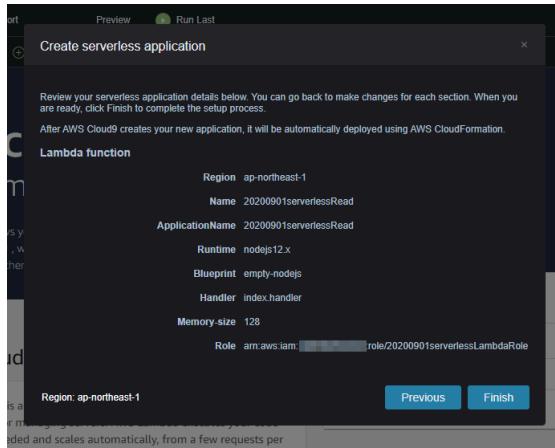
9. 以下の通りに選択します。

- **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

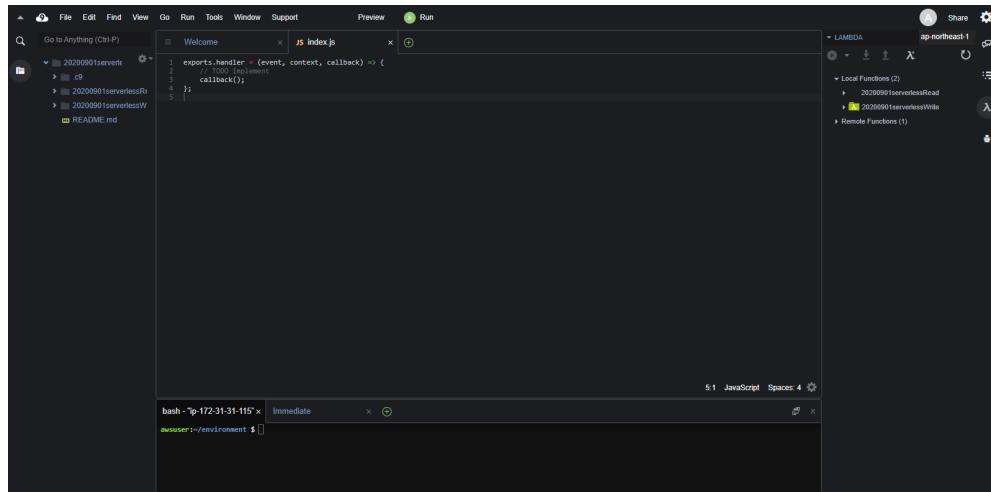


10. [Next] をクリックします。

11. 確認画面になりますので、[Finish] をクリックします。



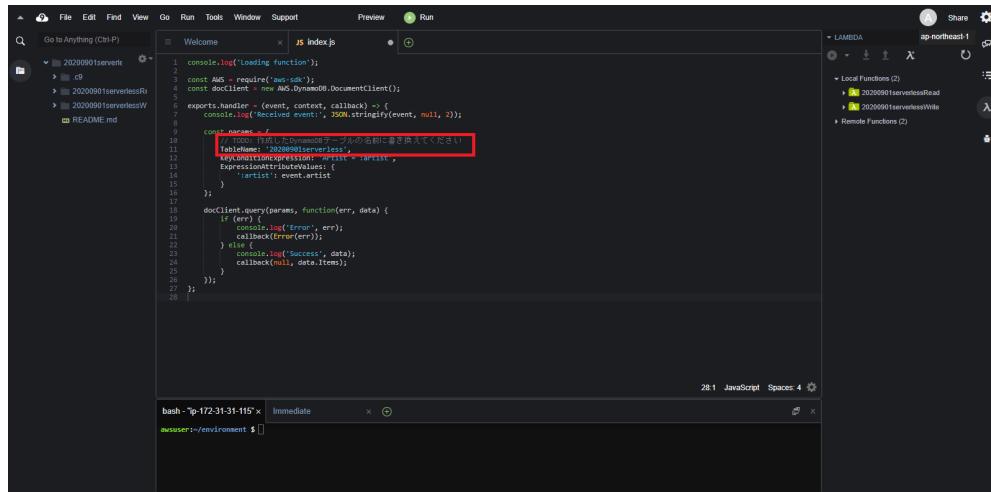
12. Lambda 関数コード (index.js) の編集画面が表示されます。



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。



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

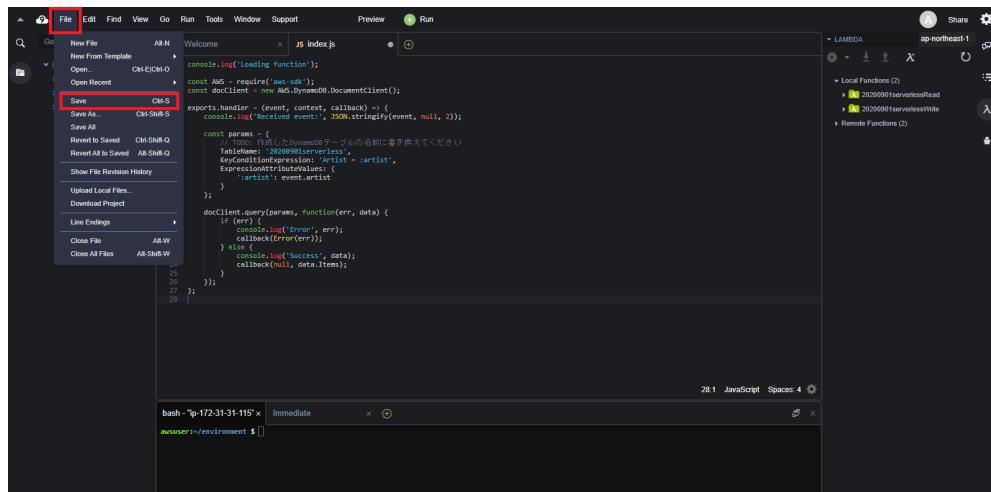
const AWS = require('aws-sdk');
const docClient = new AWS.DynamoDB.DocumentClient();

exports.handler = (event, context, callback) => {
    console.log('Received event:', JSON.stringify(event, null, 2));

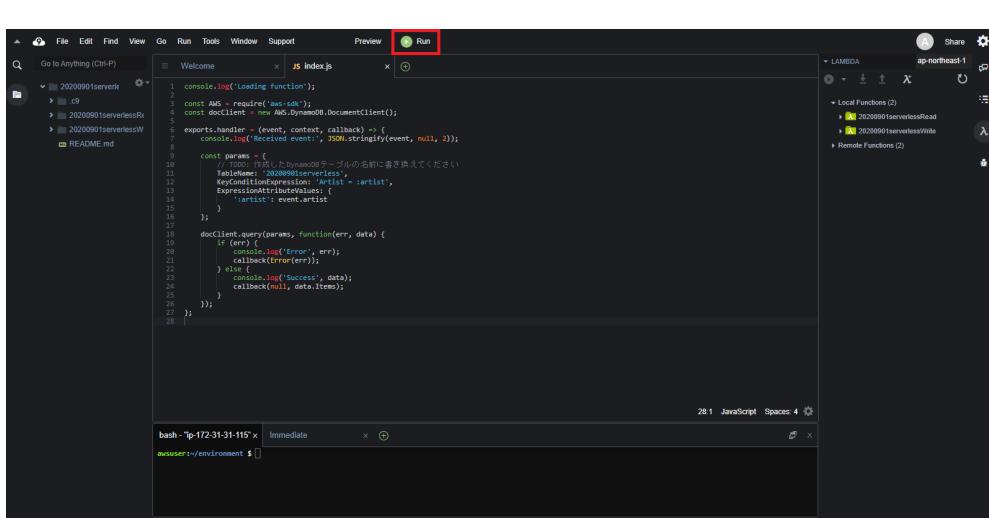
    const params = {
        TableName: '20200901serverless',
        KeyConditionExpression: '#artist = :artist',
        ExpressionAttributeValues: {
            ':artist': event.artist
        }
    };

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

14. コードの編集が終わりましたら、画面上部のメニューから [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 console interface. On the left, there's a sidebar with project navigation. The main area has tabs for 'Welcome' and 'index.js'. The 'index.js' tab contains the following code:

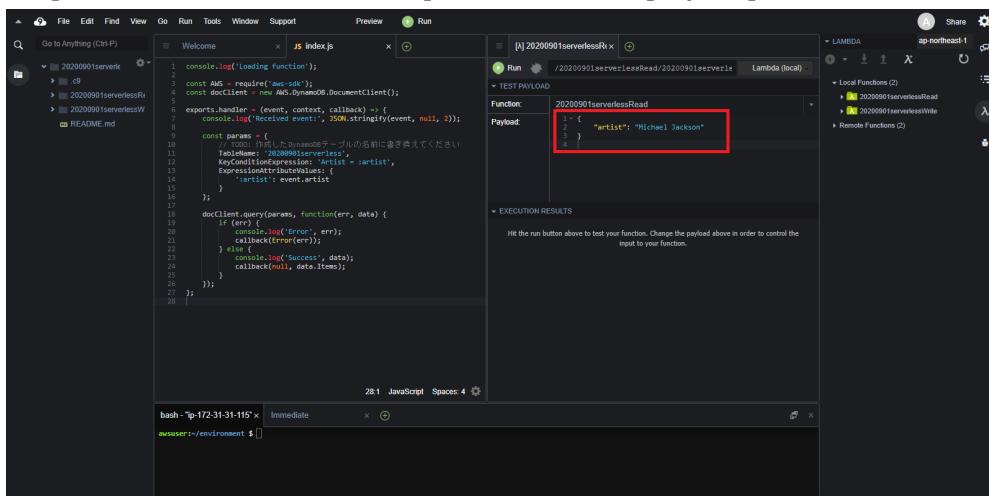
```

1 console.log('Loading function');
2
3 const AWS = require('aws-sdk');
4
5 const docClient = new AWS.DynamoDB.DocumentClient();
6
7 exports.handler = (event, context, callback) => {
8   const params = {
9     TableName: '20200901serverless',
10    KeyConditionExpression: 'Artist = :artist',
11    ExpressionAttributeValues: {
12      ':artist': event.artist
13    }
14  };
15
16  docClient.query(params, function(err, data) {
17    if (err) {
18      console.error('Error', err);
19      callback(err);
20    } else {
21      console.log('Success', data);
22      callback(null, data.Items);
23    }
24  });
25}
26
27
28

```

On the right, the Lambda service details are shown, including 'Local Functions (2)' and 'Remote Functions (2)'. Below the editor, there's an 'Immediate' terminal window with the command 'awsuser@~/environment \$'.

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。



This screenshot shows the Lambda function configuration page for the '20200901serverlessRead' function. The 'Payload' field is highlighted with a red box and contains the following JSON:

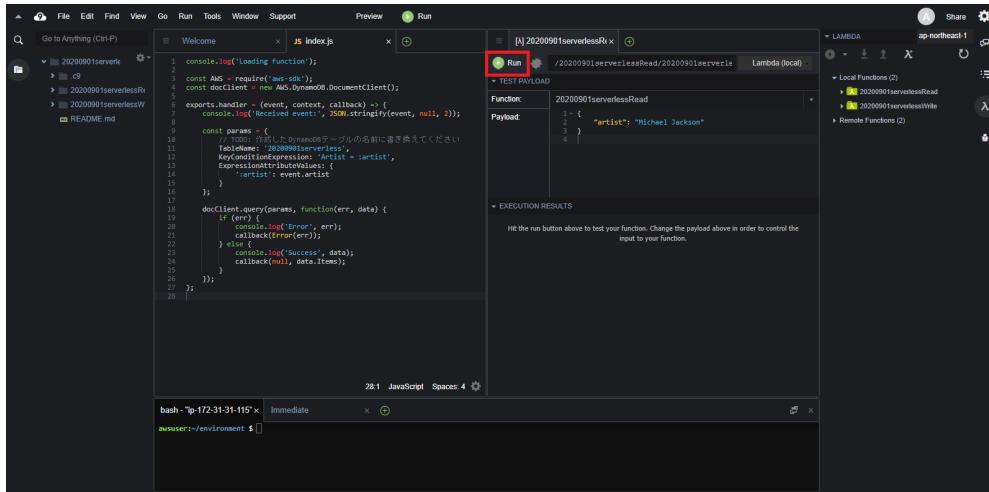
```

1 { "Artist": "Michael Jackson" }

```

The rest of the interface is similar to the first screenshot, showing the code editor and terminal window.

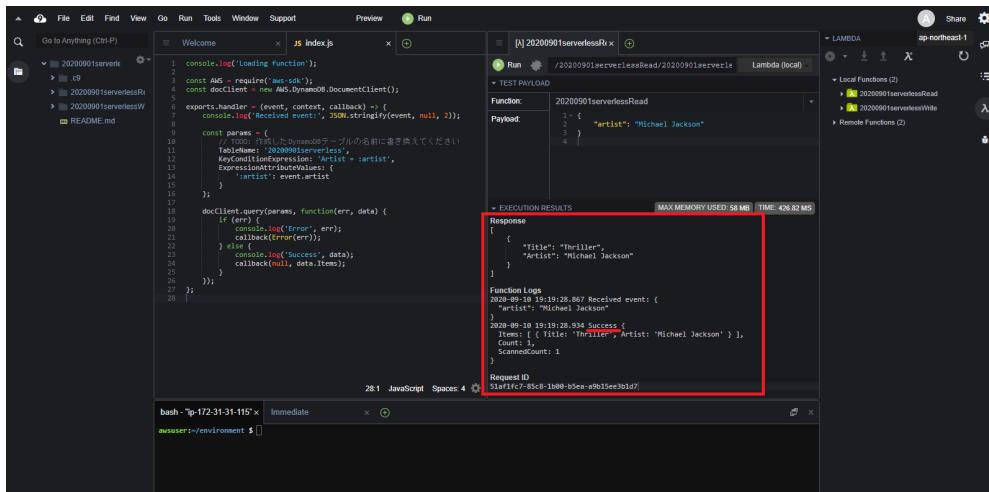
17. Lambda 関数実行タブの [Run] をクリックします。



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

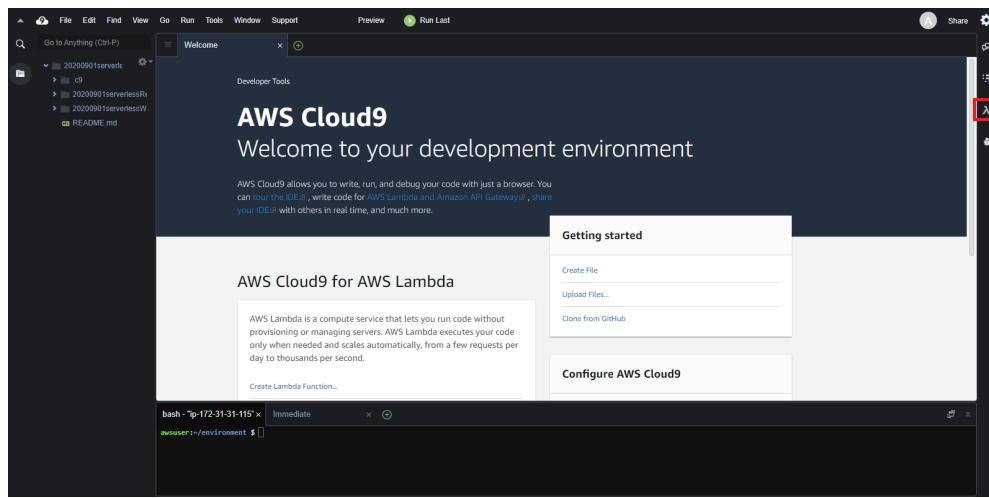
エラー等が発生しておらず、[Success] と表示されていれば成功です。

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

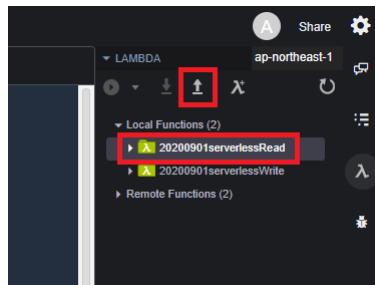
1. 画面左側のツールバーから [Lambda] ボタンをクリックします。



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。



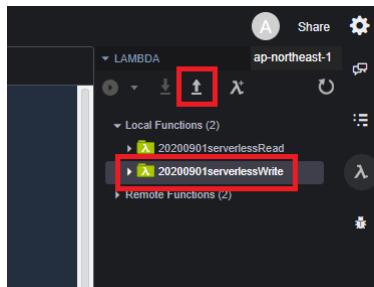
3. Lambda 関数のデプロイが行われます。

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
cloud9-20200901serverlessRe-20200901serverlessRead-MC3X6MS0OKIO		Node.js 12.x	1.6 kB	1 分前
cloud9-20200901serverlessW-20200901serverlessWrite-1XGA5XCM67JSA		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。

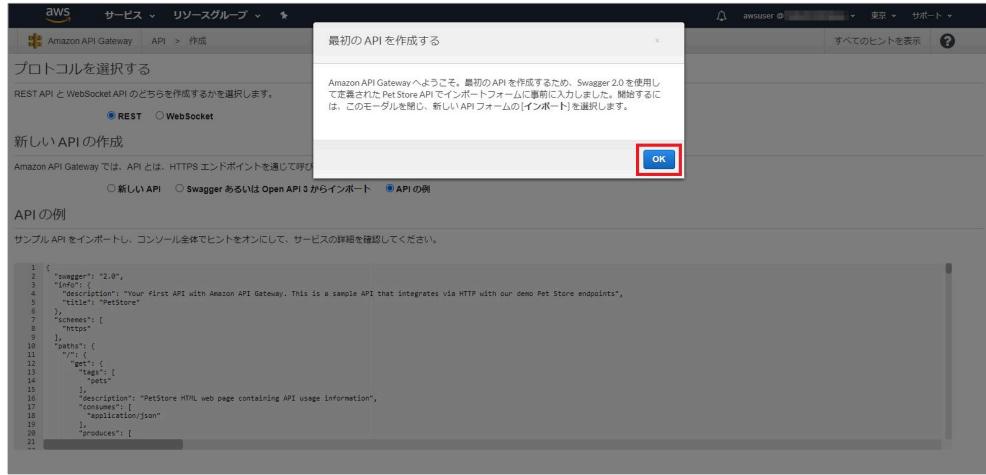


2. [API タイプを選択] の中から [REST API] の [構築] をクリックします。

※ [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



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。

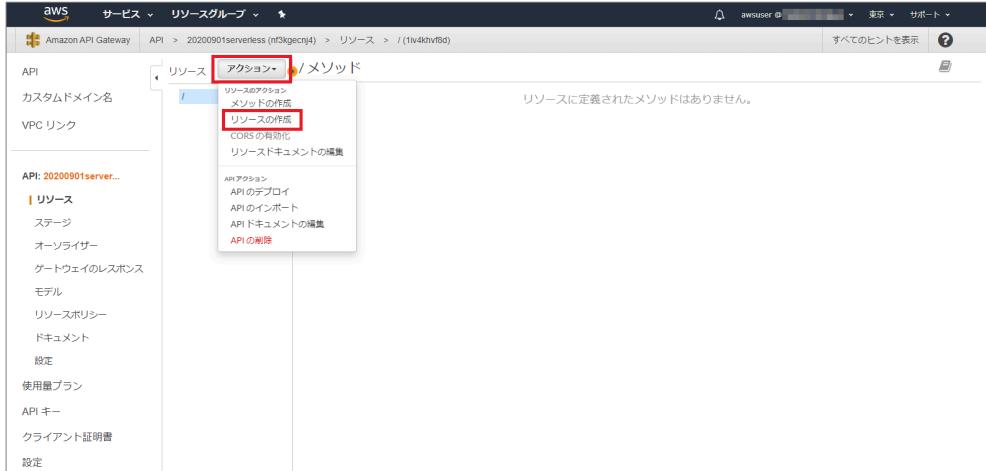
(YYYYMMDD は本日の日付)

[エンドポイントタイプ] は [リージョン] のままにします。

The screenshot shows the 'Create New API' form. The 'API Name' field is filled with '20200901serverless'. The 'Endpoint Type' dropdown is set to 'Region'. Both the 'API Name' field and the 'Endpoint Type' dropdown are highlighted with red boxes.

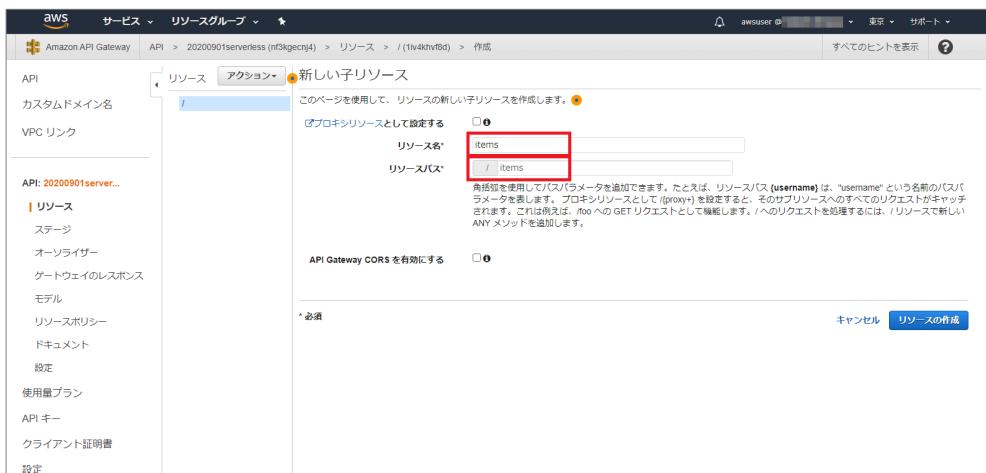
4. [APIの作成] をクリックします。

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。



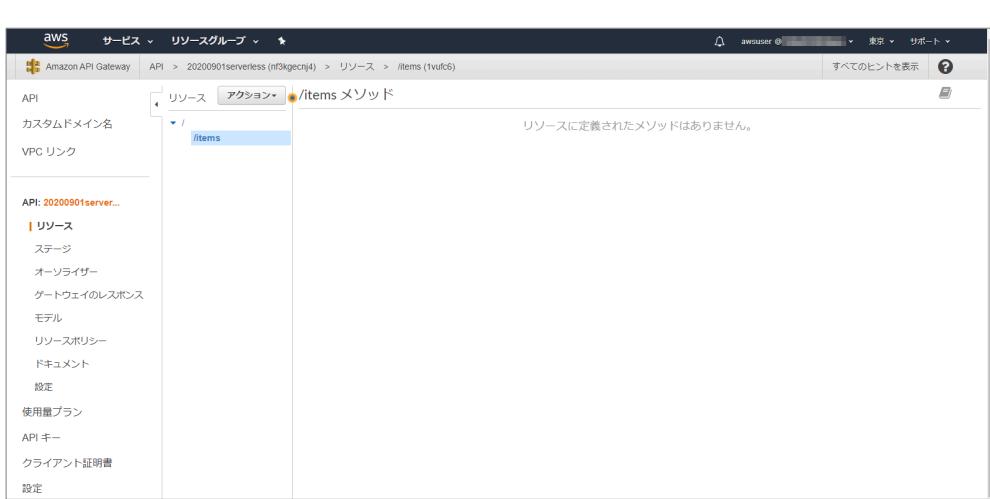
6. [リソース名] 欄に [items] と入力します。

[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。



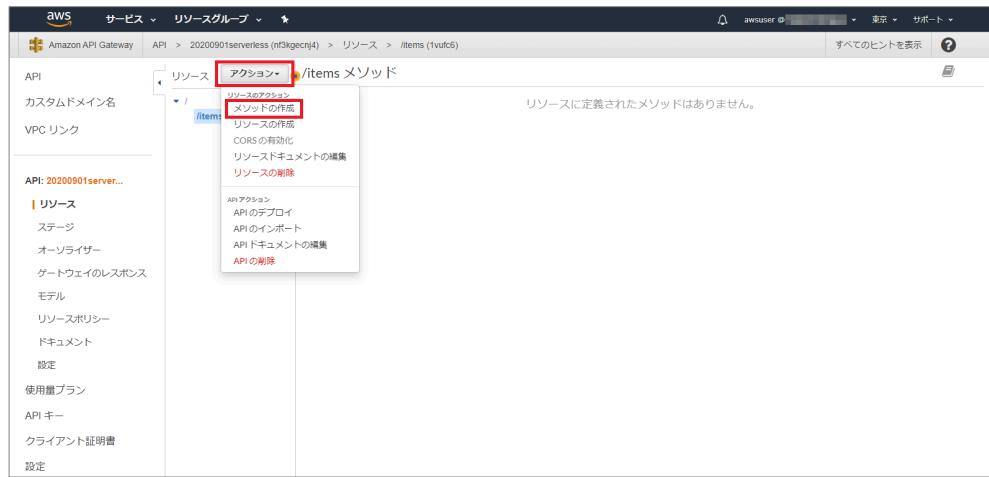
7. [リソースの作成] をクリックします。

8. リソース [/items] が作成されました。

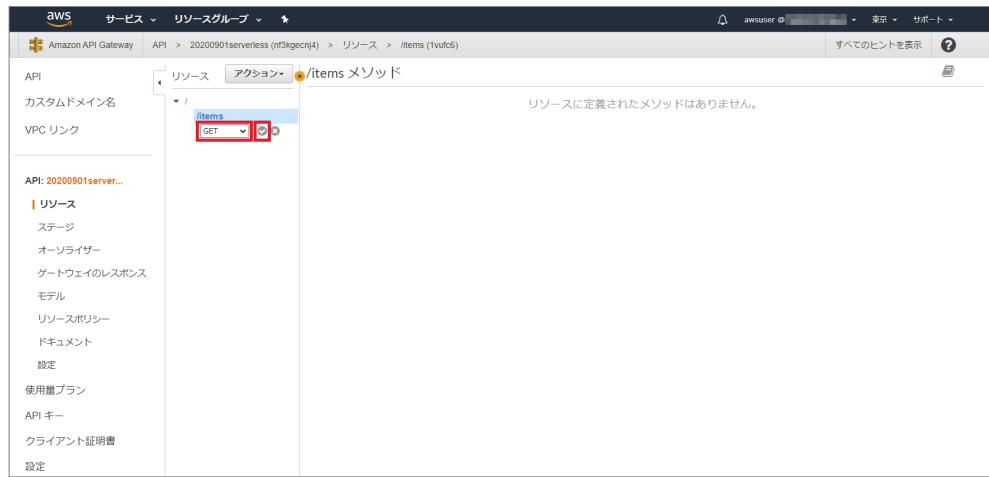


#### 4.2.2. API に GET メソッドを追加

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。



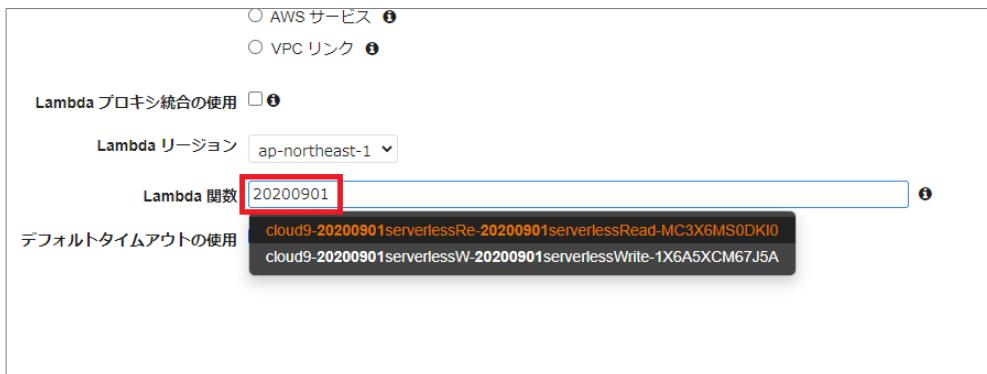
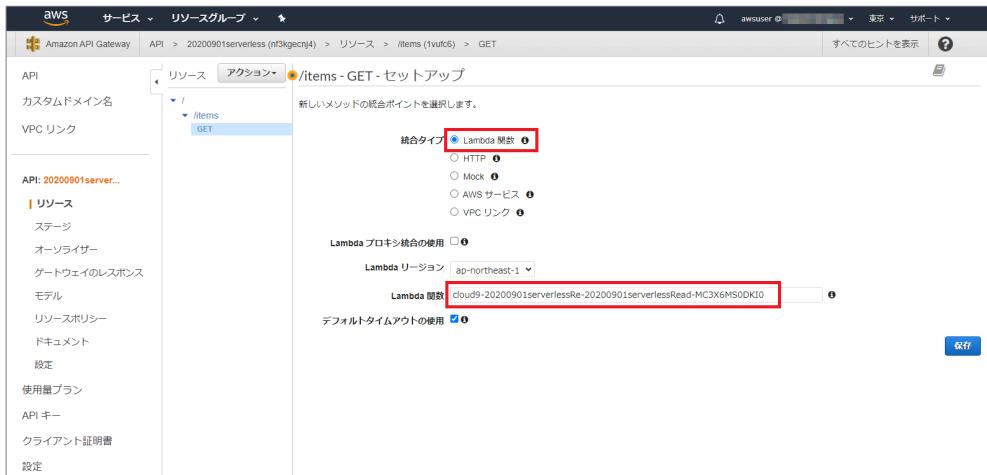
2. メソッド種類のプルダウンから [GET] を選択して、右側のチェックボタンをクリックします。



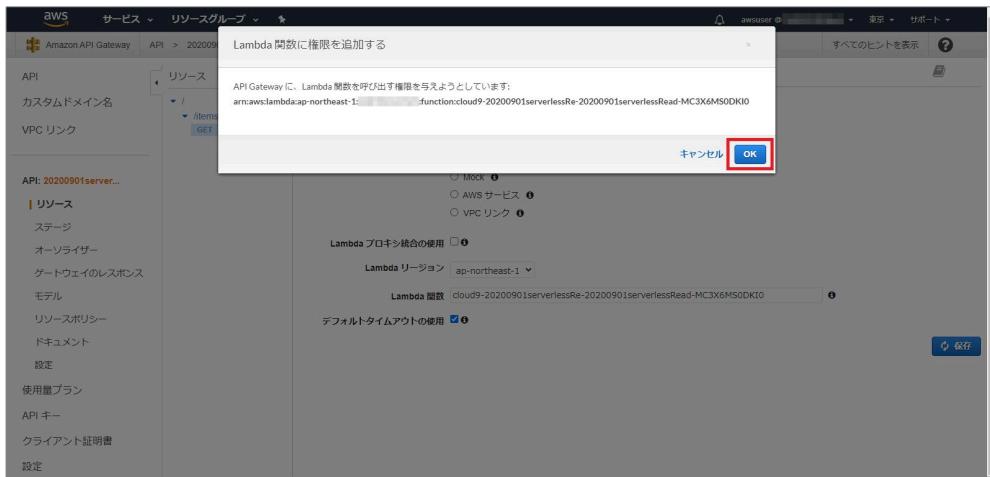
3. [統合タイプ] はデフォルトの [Lambda 関数] のままにします。

[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。

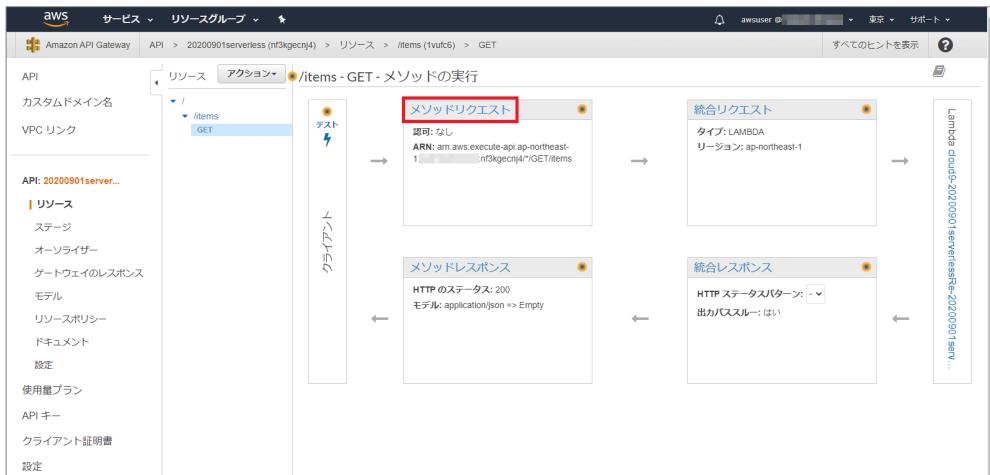
[Lambda 関数] 欄に、前節で作成した Lambda 関数のうち「Read」の方の関数名を入力します。（関数名の一部、例えば [YYYYMMDD] を入力すると候補が表示されて容易に入力できます）（Read と Write は間違えやすいので注意です）



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 console. On the left, there's a navigation sidebar with options like 'API', 'カスタムドメイン名', 'VPC リンク', and 'API: 20200901server...'. The main area shows a tree structure with 'リソース' and 'アクション' selected. Under 'アクション', a 'GET' method for the '/items' resource is chosen. The '設定' tab is open, revealing the 'Query String Parameters' section. This section has three columns: '名前' (Name), '必須' (Required), and 'キャッシュ' (Cache). A red box highlights the '名前' column of the first row, which is currently empty. Below this table, there's a button labeled 'クエリ文字列の追加' (Add Query String Parameter) with a red box around it.

7. [クエリ文字列の追加] をクリックします。

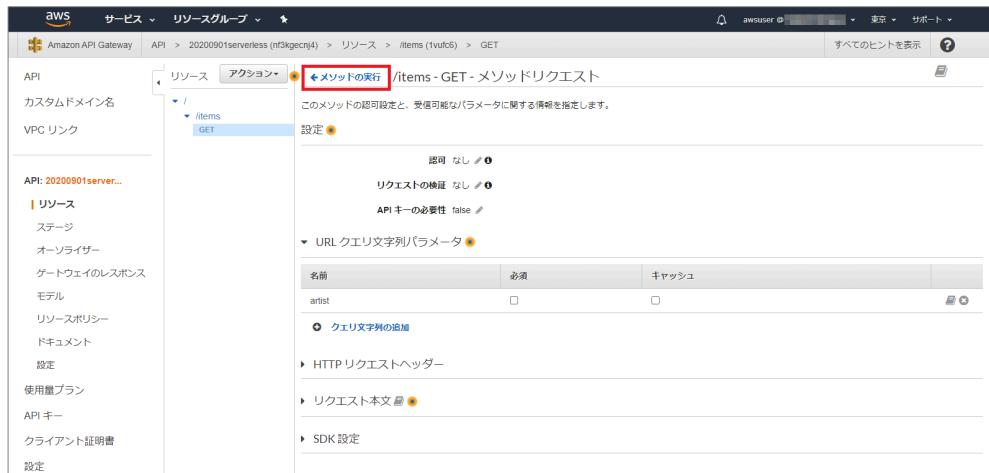
This screenshot is identical to the previous one, but the 'クエリ文字列の追加' button has been clicked. A red box now surrounds the first row of the 'Query String Parameters' table, indicating that a new parameter has been added but its name is still empty.

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。



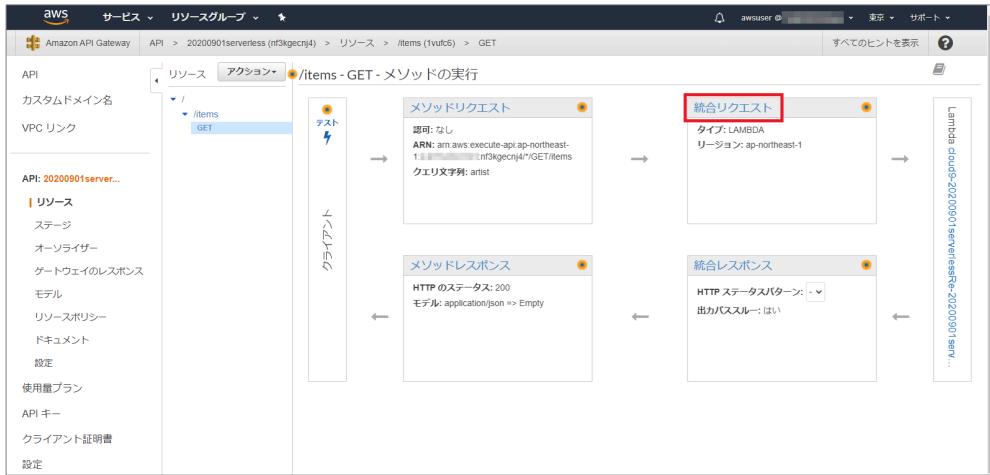
The screenshot shows the AWS API Gateway console. On the left, a sidebar lists various API settings like Custom Domain Name, VPC Link, and Resource Stage. The main panel shows an API named 'API: 20200901serverless'. Under the '/items' resource, a 'GET' action is selected. The 'Actions' tab is active. A table for 'URL クエリ文字列パラメータ' (URL Query String Parameters) is shown, with a row for '名前' (Name) containing 'artist'. The '必須' (Required) checkbox is checked and highlighted with a red box. The 'チェックボックス' (checkbox) icon at the end of the row is also highlighted with a red box.

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。

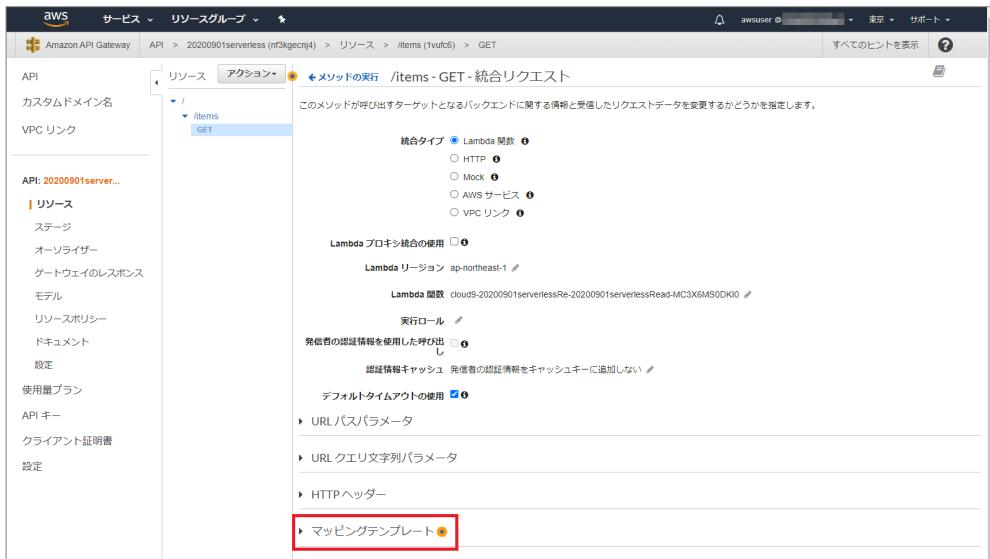


This screenshot shows the same API configuration as the previous one, but the 'Name' field in the URL query parameter table is now empty. The '必須' (Required) checkbox is unchecked. The 'checkbox' icon at the end of the row is also visible.

10. [統合リクエスト] をクリックします。



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。



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。

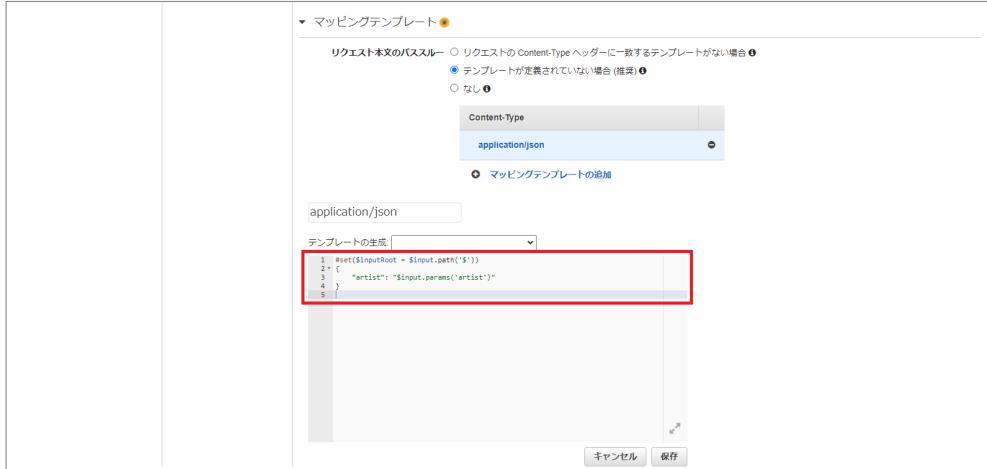
The screenshot shows the 'Mapping Template' configuration screen. On the left, there's a sidebar with 'APIキー', 'クライアント証明書', and '設定'. The main area has sections for 'URLパラメータ', 'URL クエリ文字列パラメータ', 'HTTP ヘッダー', and 'マッピングテンプレート'. Under 'マッピングテンプレート', there are two radio button options: 'リクエスト本文のバスルート' (selected) and 'テンプレートが定義されていない場合 (推薦)' (highlighted with a red box). Below these is a 'Content-Type' dropdown containing 'application/json' (also highlighted with a red box), and a 'マッピングテンプレートの追加' button at the bottom.

13. [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。 (タイプミスに注意！)

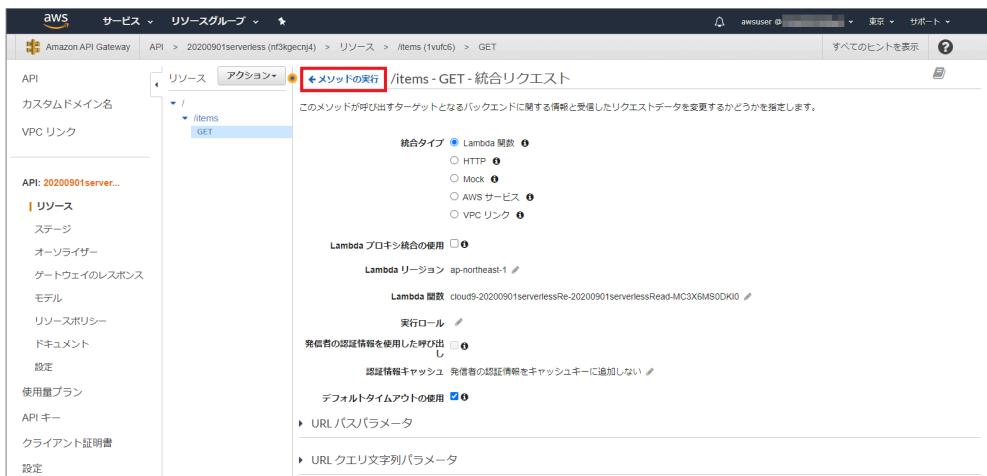
This screenshot shows the same configuration screen as the previous one, but with changes made to the 'Content-Type' field. The 'Content-Type' dropdown now contains 'application/json' (highlighted with a red box). To its right, there is a small checkbox (also highlighted with a red box) which is now checked. The rest of the interface remains the same, with the 'Template not defined' (推荐) option still selected under 'Mapping Template'.

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。

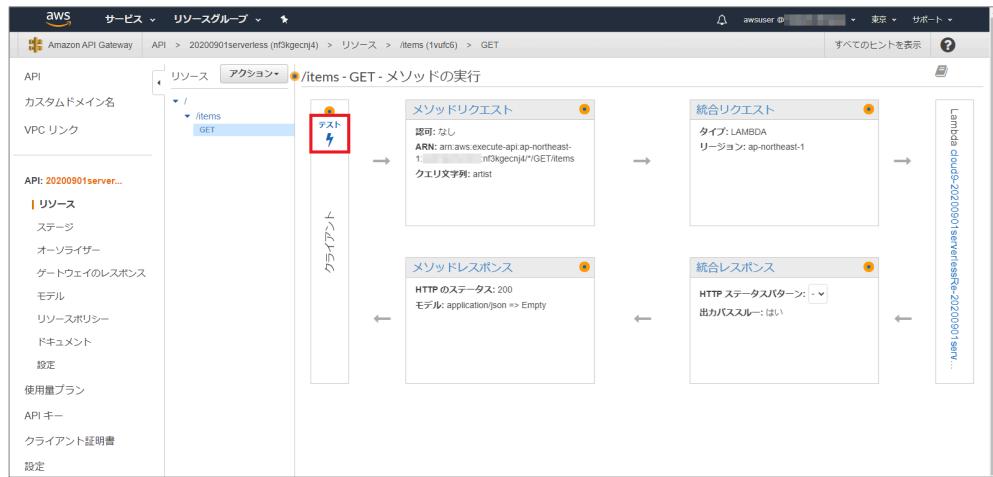


15. [保存] をクリックします。 (画面は変わりません)

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} を使用し定義します。

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 interface. The URL is `https://nfskgecnj4.execute-api.ap-northeast-1.amazonaws.com/test/items?artist=Michael%20Jackson`. The response status is **200** and the response body is:

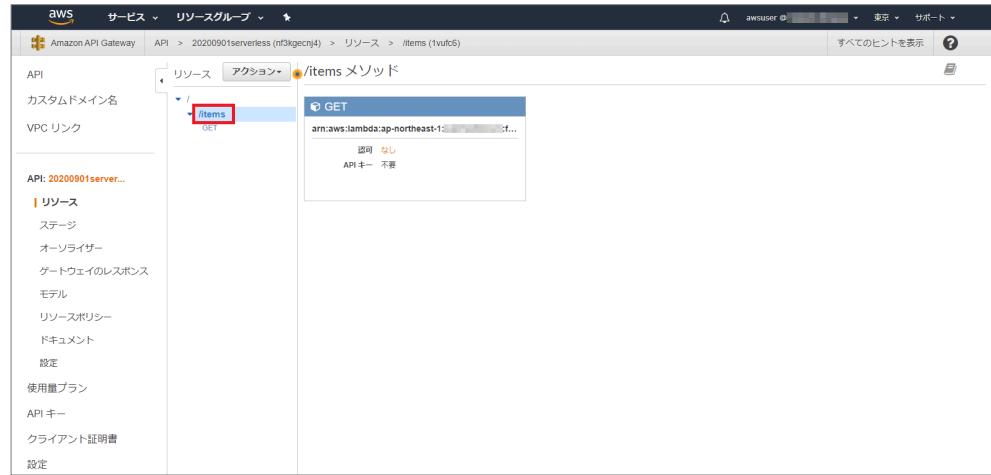
```
[{"Title": "Thriller", "Artist": "Michael Jackson"}]
```

The 'Logs' section shows the execution trace:

```
Execution log for request 2e900d4-364a-4763-9bb6-8a4f4bf7f163
Fri Sep 11 00:56:01 UTC 2020 : Starting execution for request: 2e900d4-364a-4763-9bb6-8a4f4bf7f163
Fri Sep 11 00:56:01 UTC 2020 : HTTP Method: GET, Resource Path: /items
Fri Sep 11 00:56:01 UTC 2020 : Method request path: []
Fri Sep 11 00:56:01 UTC 2020 : Method request query string: [artist=Michael Jackson]
Fri Sep 11 00:56:01 UTC 2020 : Method request headers: []
Fri Sep 11 00:56:01 UTC 2020 : Method request body before transformation:
Fri Sep 11 00:56:01 UTC 2020 : Endpoint request URL: https://ynamodb.ap-northeast-1.amazonaws.com/2015-03-31/functions/invocations?FunctionArn=20200901serverlessfn-20200901serverlessfn-MC39M500K10/invoc
```

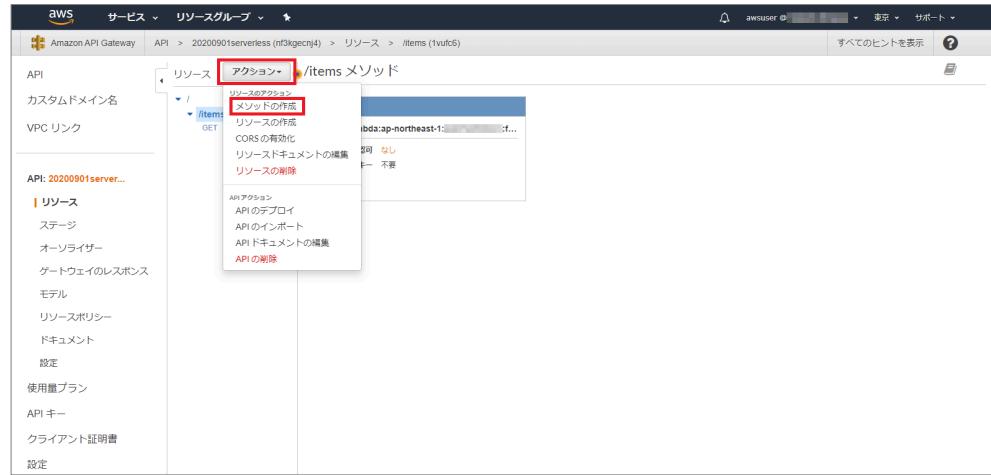
#### 4.2.3. API に POST メソッドを追加

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。



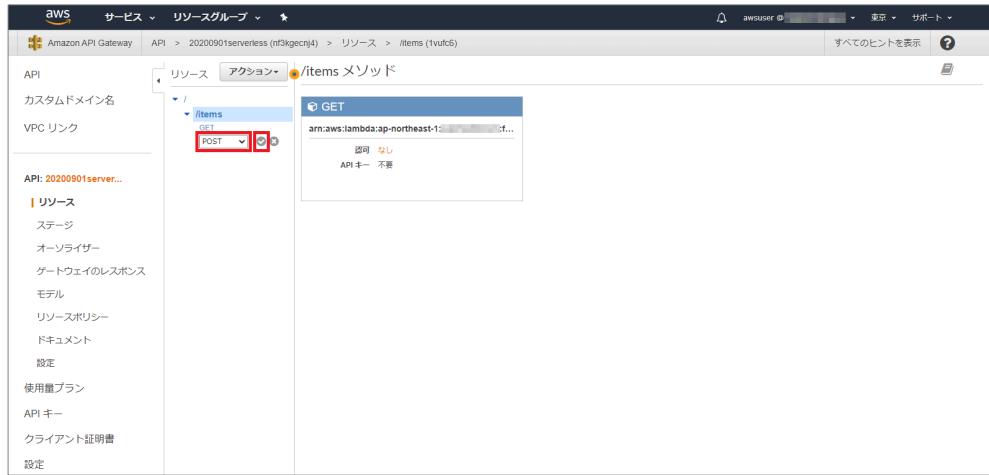
The screenshot shows the AWS Lambda function configuration interface. On the left, a sidebar lists various service options like API Gateway, Lambda, CloudWatch, etc. The main area shows a tree structure under 'API'. A red box highlights the 'Items' node under the '/' node. To the right, there's a detailed view of the 'GET' method for the '/items' resource, showing the ARN and CORS settings.

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 is similar to the previous one, but the 'Actions' dropdown for the '/items' resource is now open. A red box highlights the 'Create Method' option in the dropdown menu. Other options visible include 'Edit Method', 'Delete Method', and 'API Permissions'.

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。

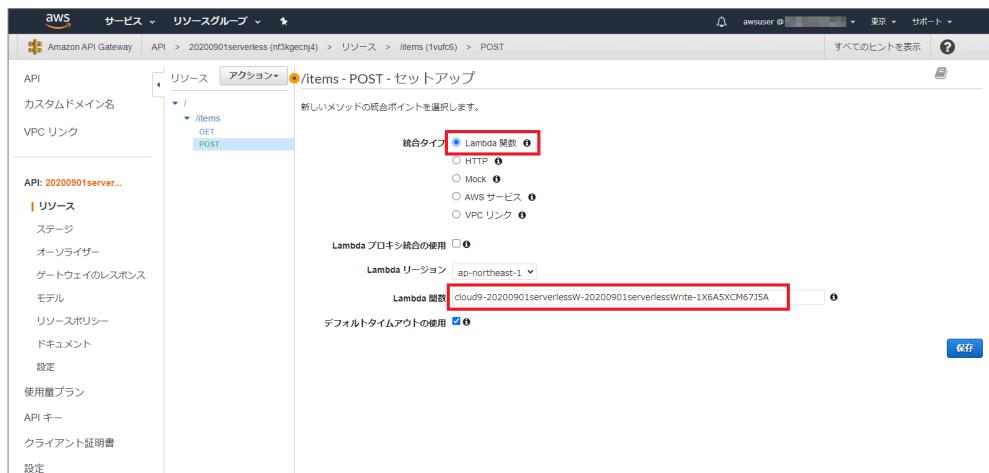


The screenshot shows the AWS Lambda function configuration interface. On the left, a sidebar lists various settings like API, VPC, and Lambda functions. In the main area, a function named '20200901serverless' is selected. Under the 'Actions' tab, there's a dropdown menu for the 'POST' method. A red box highlights this dropdown, and a smaller red box highlights the checked checkbox next to it.

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

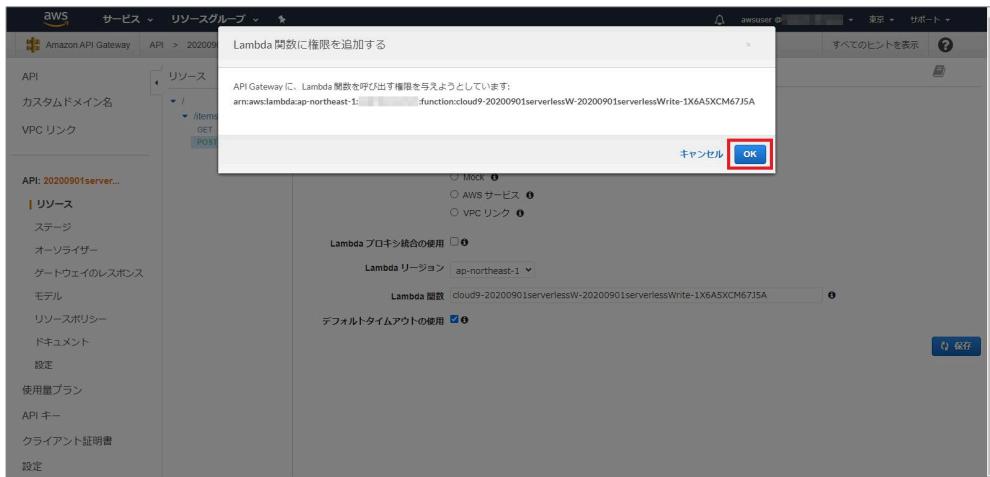
[Lambda プロキシ統合の使用] のチェックは外したままにします。

[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。

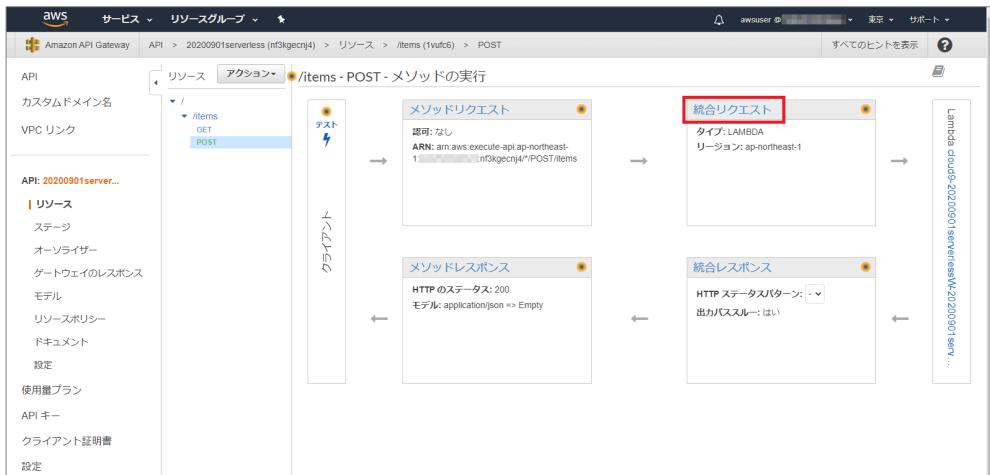


The screenshot shows the AWS Lambda function configuration interface for the 'POST' method. Under the 'Actions' tab, the 'Lambda 関数' (Lambda Function) option is selected. A red box highlights the 'Lambda 関数' button. Below it, the 'Lambda プロキシ統合の使用' (Proxy Integration) checkbox is unchecked. The 'Lambda リージョン' (Region) dropdown is set to 'ap-northeast-1'. A red box highlights the 'Lambda 関数' input field, which contains the ARN 'cloud9-20200901serverlessW-20200901serverlessWrite-1X6A5XCM675A'. At the bottom right, a blue '保存' (Save) button is visible.

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 for the POST /items endpoint. The 'Mapping Template' section at the bottom is highlighted with a red box.

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 type' dropdown is set to 'Template not defined (recommended)' and the 'Add mapping template' button is highlighted with a red box.

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。

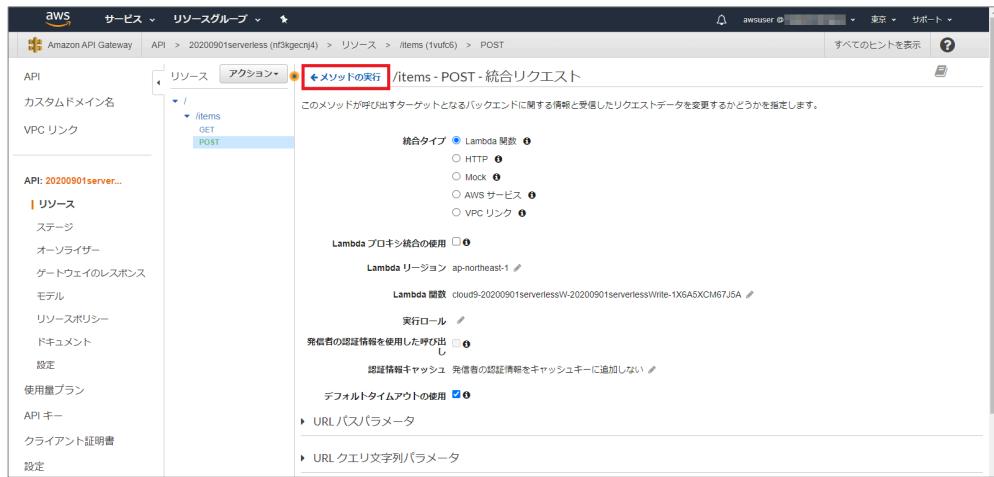


10. [apigateway\_post\_mapping.txt] の内容をコピーしてテンプレートの入力欄にペーストします。

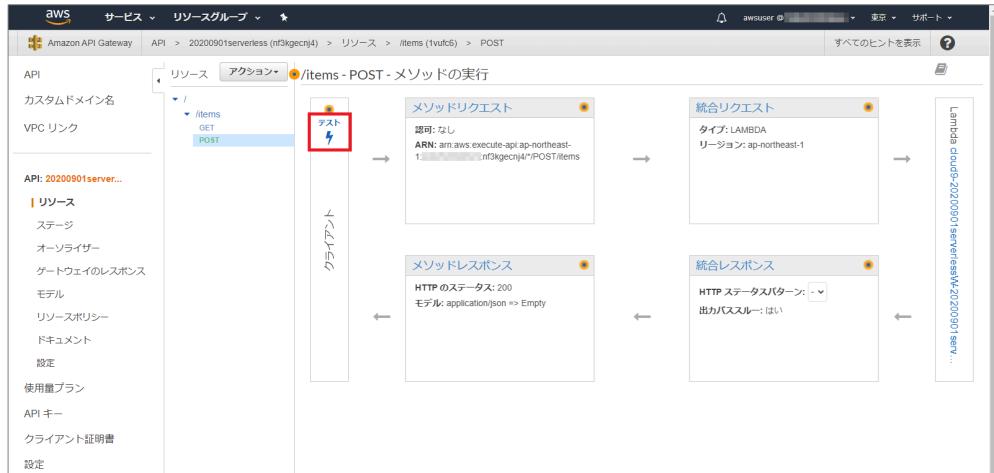


11. [保存] をクリックします。 (画面は変わりません)

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。



13. 【テスト】をクリックします。



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 console. On the left, a sidebar lists various API settings like custom domains, VPC links, and stages. The main area shows an API named '20200901serverless' with a single resource path '/items'. Under this path, there are two methods: 'GET' and 'POST'. The 'POST' method is selected. A large text input field labeled 'リクエスト本文' (Request Body) contains the following JSON:

```
1: {
2:   "artist": "Madonna",
3:   "title": "Like a Virgin"
4: }
```

This JSON is highlighted with a red rectangular box.

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 structure is visible with a POST method selected for the /items endpoint. In the main area, the test results are displayed. The status code is shown as "ステータス: 200". The response body is empty, indicated by "[]". The log panel shows the execution details of the Lambda function, including the request ID and timestamp. The log output shows the JSON payload sent to the Lambda function: {"Artist": "Madonna", "Title": "Like a Virgin"}

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 console. The left sidebar shows the table named "20200901serverless". The main area displays the table's contents under the "項目" tab. A search bar at the top is set to "スキャン" and filters for the table "20200901serverless". The results show two items: "Artist": "Madonna", "Title": "Like a Virgin" and "Artist": "Michael Jackson", "Title": "Thriller". The item "Like a Virgin" is highlighted with a red box.

#### 4.2.4. CORS の設定

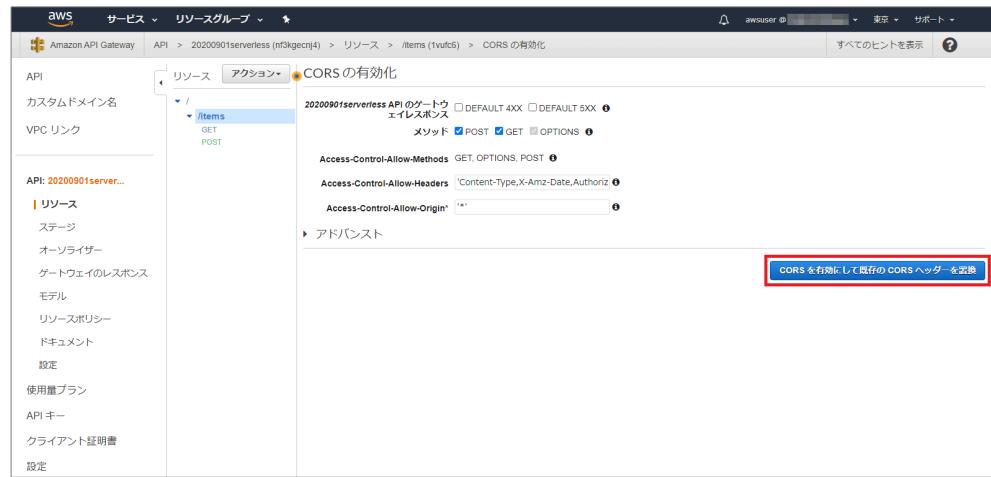
1. リソースのツリー階層から **[/items]** をクリックして選択します。

The screenshot shows the AWS API Gateway console. On the left, there's a navigation sidebar with options like 'API', 'カスタムドメイン名', 'VPC リンク', and 'API: 20200901server...'. Under 'API' is a 'リソース' section with a tree view. A red box highlights the '/items' node under the '/' node. To the right, there are two method cards: 'GET' and 'POST', both pointing to 'arm:aws:lambda:ap-northeast-1:...' and requiring 'なし' (None) for both '認可' (Authorization) and 'API キー' (API Key). There's also a small note 'API メソッドを変更するには'.

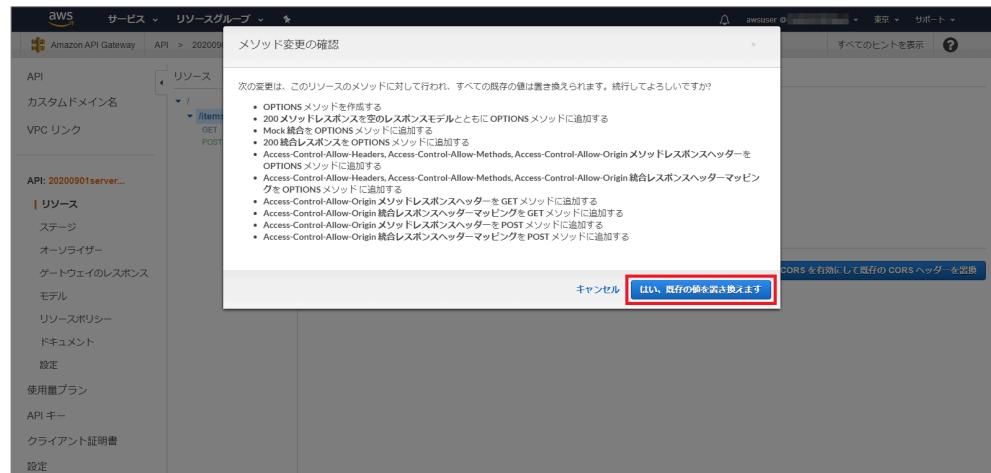
2. [アクション] プルダウンから **[CORS の有効化]** を選択します。

This screenshot is from the same AWS API Gateway interface as the previous one. The '/items' resource is still selected. In the 'Actions' dropdown menu (indicated by a red box), the 'CORS の有効化' (Enable CORS) option is highlighted. The other options in the dropdown include 'リソースの作成' (Create resource), 'リソースの削除' (Delete resource), 'API の作成' (Create API), 'API の削除' (Delete API), and 'API の複数選択' (Select multiple APIs).

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。



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。



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 the function's settings: API, カスタムドメイン名, VPC リンク, API: 20200901server..., リソース, ステージ, オーソライザー, 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, 設定, 使用量プラン, API キー, クライアント証明書, and 設定. The main pane displays the 'CORS の有効化' (Enable CORS) step, which has been successfully completed. A red box highlights the green checkmarks next to the status messages indicating successful execution of various CORS-related steps.

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 the function's settings: API, カスタムドメイン名, VPC リンク, API: 20200901server..., リソース, ステージ, オーソライザー, ゲートウェイのレスポンス (highlighted with a red box), 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, 使用量プラン, API キー, クライアント証明書, and VPC リンク. The right pane shows the 'Gateway Responses' configuration, specifically the '設定' (Settings) tab. It contains a list of response types with radio buttons, and a note at the top asking to select a gateway response.

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 API Gateway console. On the left, a sidebar lists various API settings like 'Custom Domain Name', 'VPC Link', and 'APIs'. The main area is titled 'Gateway Responses' and shows a list of responses for the '20200901serverless' API. The '403' response is currently selected. On the right, there's a detailed configuration panel for the '403' response. At the top of this panel, there's a 'Edit' button which is highlighted with a red box.

8. [レスポンスヘッダーの追加] をクリックします。

This screenshot is from the same interface as the previous one, but the 'Responses' tab is now selected. The 'Edit' button remains highlighted. At the bottom of the 'Responses' section, there is a button labeled 'Add Response Header' which is also highlighted with a red box.

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（シングルレクオーテーション、アスタリスク、シングルレクオーテーション）

The screenshot shows the AWS API Gateway interface. On the left, a sidebar lists various API settings like 'リソース', 'ステージ', and 'モデル'. The main area is titled 'ゲートウェイのレスポンス' (Gateway Response) for a specific API endpoint. It shows a dropdown for the reason code '設定 - アクセスが拒否されました' (Access denied) set to '403'. Below it, there's a table for 'レスポンスヘッダー' (Response Headers). A new row is being added, with '名前' (Name) set to 'Access-Control-Allow-Origin' and '値' (Value) set to '\*'. The '保存' (Save) button is visible at the top right.

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。

The screenshot shows the AWS Lambda console with the API Gateway service selected. The left sidebar lists various API components: API, カスタムドメイン名, VPC リンク, API: 20200901server..., リソース, ステージ, オーナライザー, 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, 設定, 使用量プラン, API キー, クライアント証明書, and 設定. The 'リソース' section is expanded, showing a list of actions: GET, OPTIONS, POST, and items. A red box highlights the 'items' action, which is currently selected.

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。

The screenshot shows the AWS Lambda console with the API Gateway service selected. The left sidebar lists various API components: API, カスタムドメイン名, VPC リンク, API: 20200901server..., リソース, ステージ, オーナライザー, 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, 設定, 使用量プラン, API キー, クライアント証明書, and 設定. The 'リソース' section is expanded, showing a list of actions: GET, OPTIONS, POST, and items. A red box highlights the 'items' action, which is currently selected. The 'Actions' dropdown menu is open, showing several options: リソースのアクション, メソッドの作成, リソースの作成, CORSの有効化, リソースドキュメントの構築, APIアクション, APIのデプロイ, APIのインポート, APIドキュメントの選択, and APIの削除. The 'APIのデプロイ' option is highlighted with a red box.

3. 以下の通り入力します。

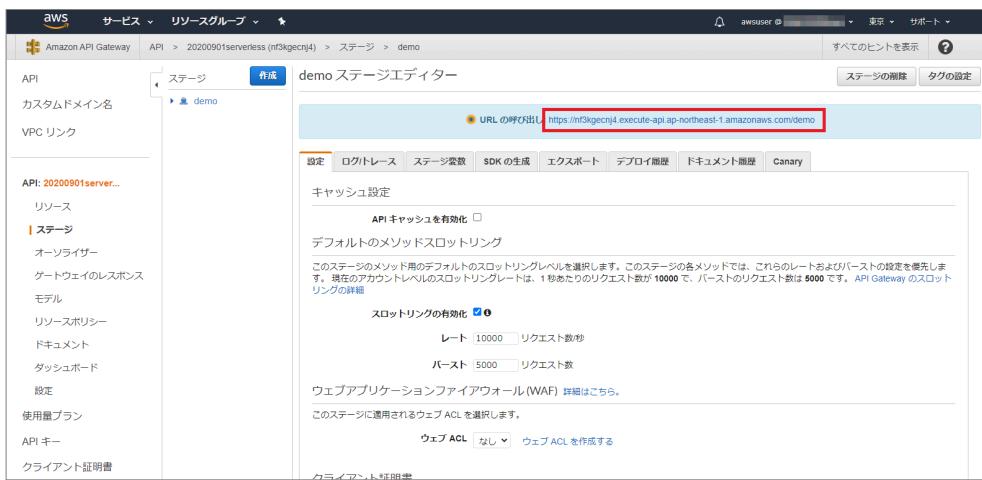
- **デプロイされるステージ:** [新しいステージ] を選択
- **ステージ名:** [demo] と入力
- **ステージの説明:** (省略)
- **デプロイメントの説明:** (省略)



4. [デプロイ] をクリックします。

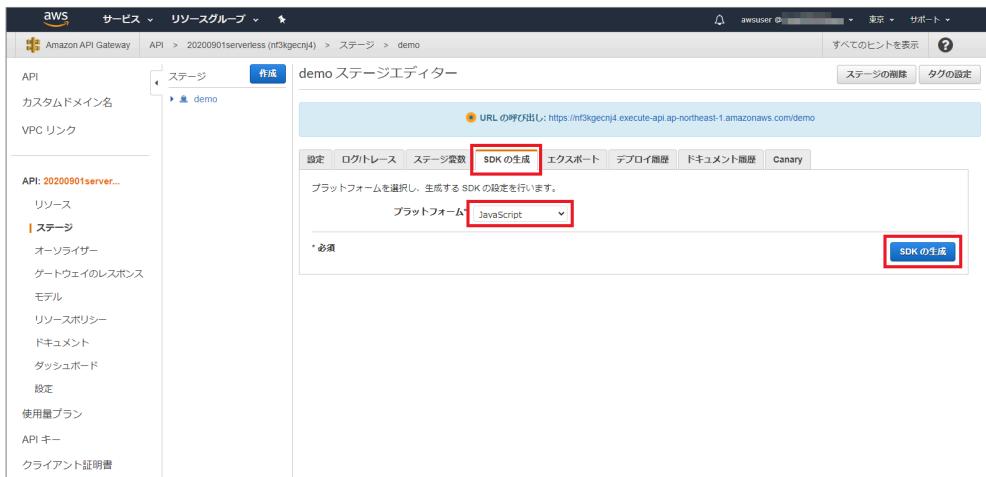
5. デプロイが行われ、ステージ [demo] の画面が表示されます。

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

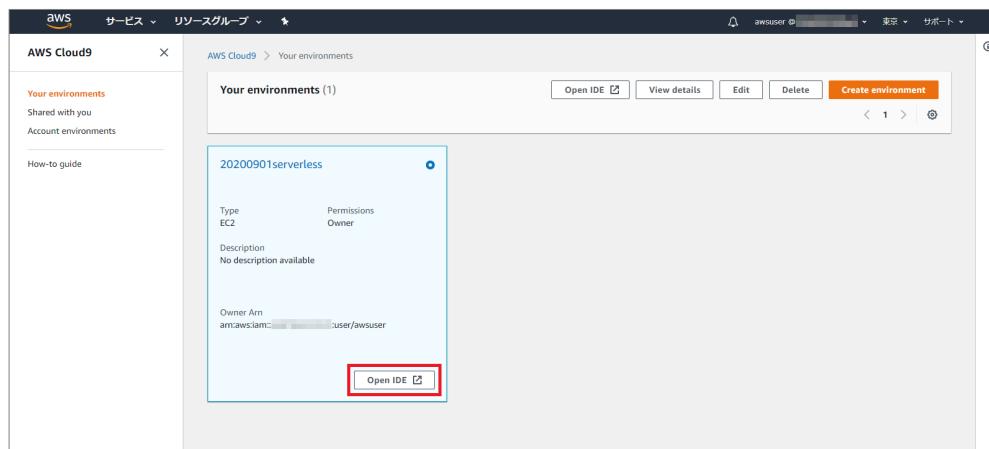
の場所に保存します。 (後の手順で使用します)

#### 4.2.6. Cloud9からの動作確認

##### 1. Cloud9の画面へ移動します。

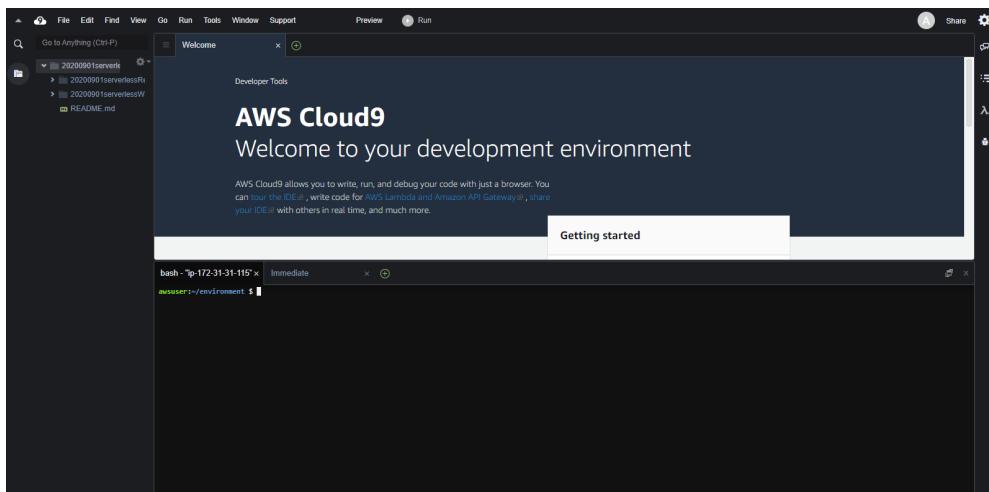
時間が経っている場合には Cloud9 の EC2 インスタンスが停止している可能性があります。

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 table '20200901serverless' selected. The 'Items' section displays the following data:

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

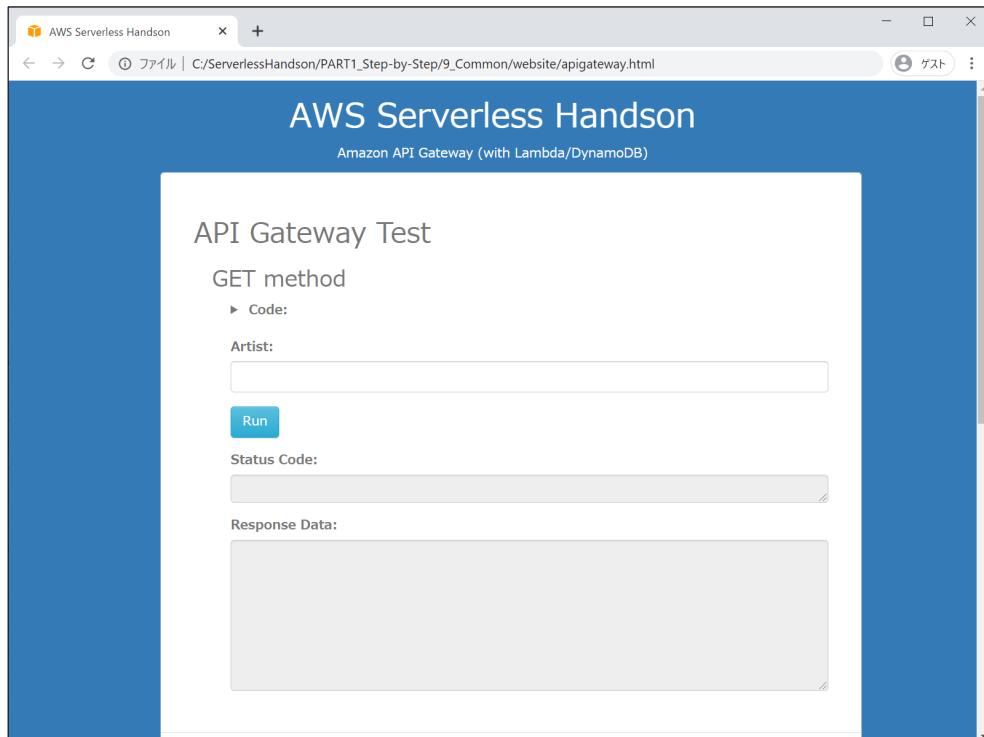
#### 4.2.7. Web ブラウザからの動作確認

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. この時点では、[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client 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

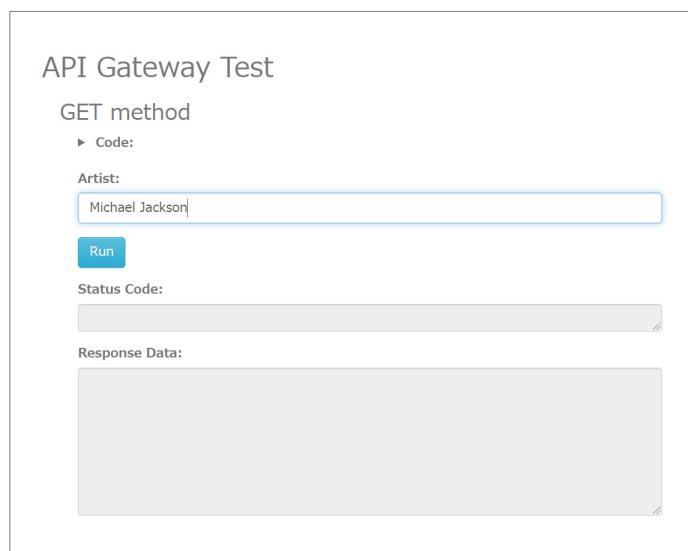
▶ Code:

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:

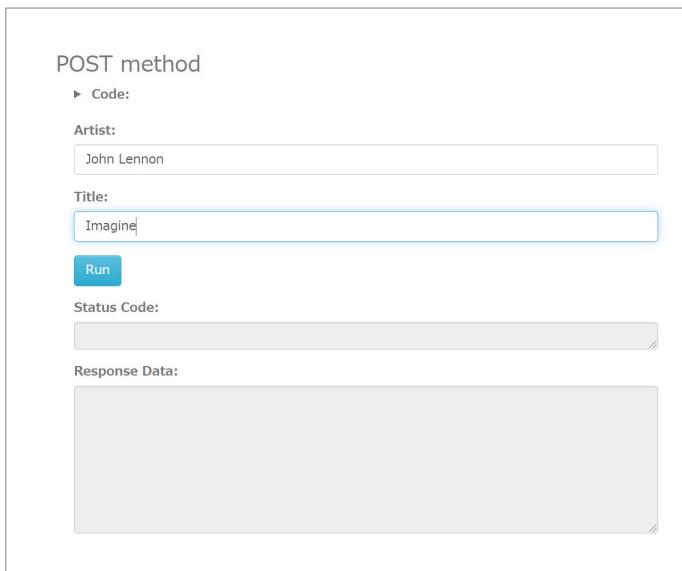
Artist:  
John Lennon

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:

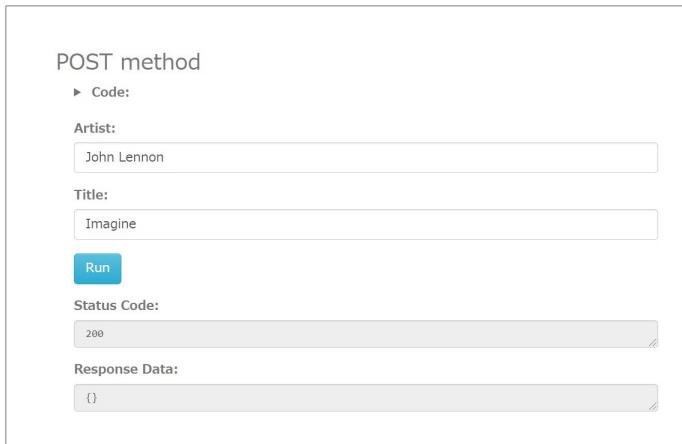
Artist:  
John Lennon

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 Amazon Cognito service page. At the top, there's a navigation bar with 'aws' logo, 'サービス', 'リソースグループ', and a user dropdown. Below the navigation is a large purple circular icon with a white 'C' inside. The main title is 'Amazon Cognito'. A descriptive text block explains that Cognito provides user pools and ID pools for sign-up and sign-in operations. Two buttons are at the bottom: 'ユーザープールの管理' (User Pool Management) which is highlighted with a red box, and 'ID プールの管理' (ID Pool Management). 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) with an icon of two people and a plus sign, 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) with an icon of a computer monitor and a lock.

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 a user dropdown. The main title is 'ユーザープール' (User Pool). A sub-header says 'ユーザープール|フェデレーション|ID プール|アイデンティティ'. On the right side, a blue button labeled 'ユーザープールを作成する' (Create User Pool) is highlighted with a red box. Below the button, a note says 'ユーザープールがありません。ユーザープールを作成するには、ここをクリックします。' (No user pool exists. Click here to create a user pool.)

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。

aws サービス リソースグループ 東京 サポート キャンセル

ユーザー プール | フェデレーション ID アイデンティティ

ユーザー プールを作成する

名前  
属性  
ポリシー  
MFA そして 確認  
メッセージのカスタマイズ  
タグ  
デバイス  
アプリケーション  
トリガー  
確認

Pool Name: 20200901serverless

ユーザー プールにどのような名前を付けますか?  
今後簡単に特定できるように、ユーザー プールにわかりやすい名前をつけてください。

ユーザー プールをどのように作成しますか?

デフォルトを確認する  
デフォルトを確認してから必要なに応じてカスタマイズする

ステップに従って設定する  
各設定をステップに従ってを選択する

4. 画面左側のメニューから [ポリシー] を選択します。

aws サービス リソースグループ 東京 サポート キャンセル

ユーザー プール | フェデレーション ID アイデンティティ

ユーザー プールを作成する

名前  
属性  
ポリシー  
MFA そして 確認  
メッセージのカスタマイズ  
タグ  
デバイス  
アプリケーション  
トリガー  
確認

Pool Name: 20200901serverless

必須の属性 email  
エイリアス属性 エイリアス属性の選択...  
ユーザー名属性 ユーザー名属性の選択...  
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 カスタム属性の選択...

パスワードの最小長: 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 の有効化...  
確認 E メール

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 page. In the 'Password Requirements' section, the 'Minimum Length' is set to 8. The 'Character Requirements' dropdown is expanded, showing four checkboxes: '数字を必要とする' (Check), '特殊文字を必要とする' (Uncheck), '大文字を必要とする' (Uncheck), and '小文字を必要とする' (Uncheck). A note below says: 'Amazonでは、セキュリティを強化するために8文字以上の大文字、小文字、数字で構成されるパスワードをお勧めします。' (Amazon recommends using a password consisting of 8 or more uppercase letters, lowercase letters, numbers, and symbols). In the 'User Sign-up' section, the radio button 'ユーザーに自己サインアップを許可する' (Allow users to sign up) is selected. In the 'Temporary Password' section, the 'Expiration Duration' is set to 7 days. At the bottom right, the 'Change' button 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。

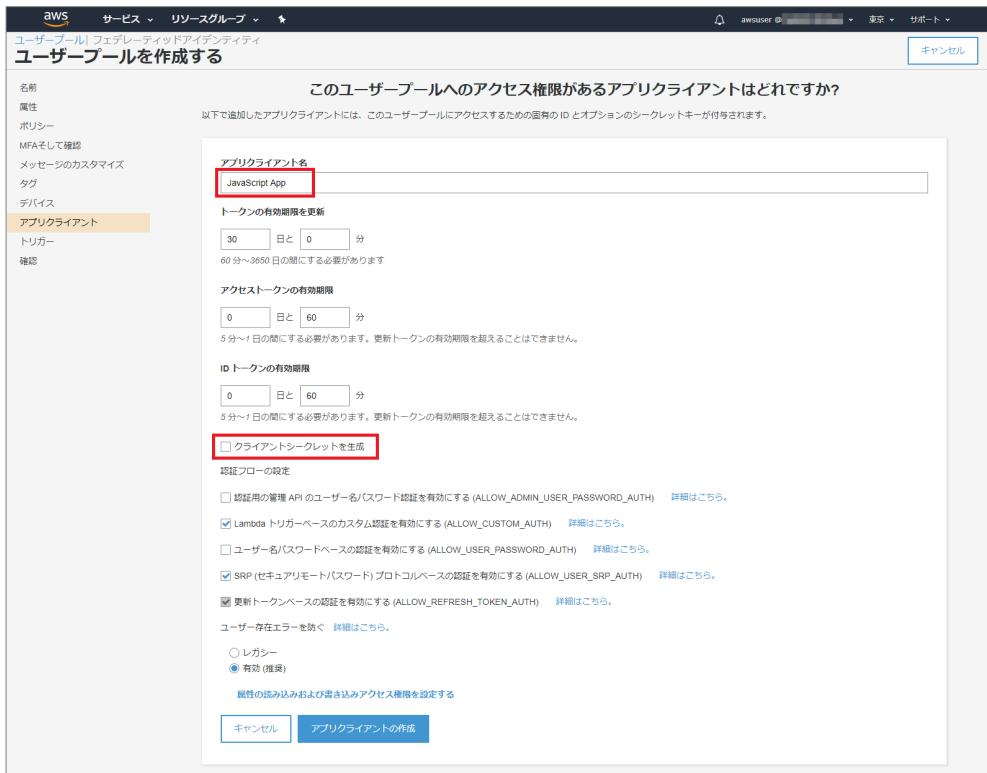
The screenshot shows the 'Create User Pool' configuration page. In the 'Client Settings' section, the 'Name' is '20200901serverless'. Under 'Default User Pool Attributes', the 'Email' attribute is selected. In the 'Custom User Pool Attributes' section, the 'Email' attribute is selected. In the 'Sign-up' section, the 'Email' attribute is selected. In the 'Email Configuration' section, the 'From Address' is 'DEFAULT' and 'Amazon SESによるEメールを有効化' (Enable Amazon SES email) is checked. In the 'MFA' section, 'MFAの有効化' (Enable MFA) is checked. In the 'Tags' section, 'タグ' (Tags) is selected. The 'Create User Pool' button is highlighted with a red box at the bottom right.

7. [アプリケーションの追加] をクリックします。



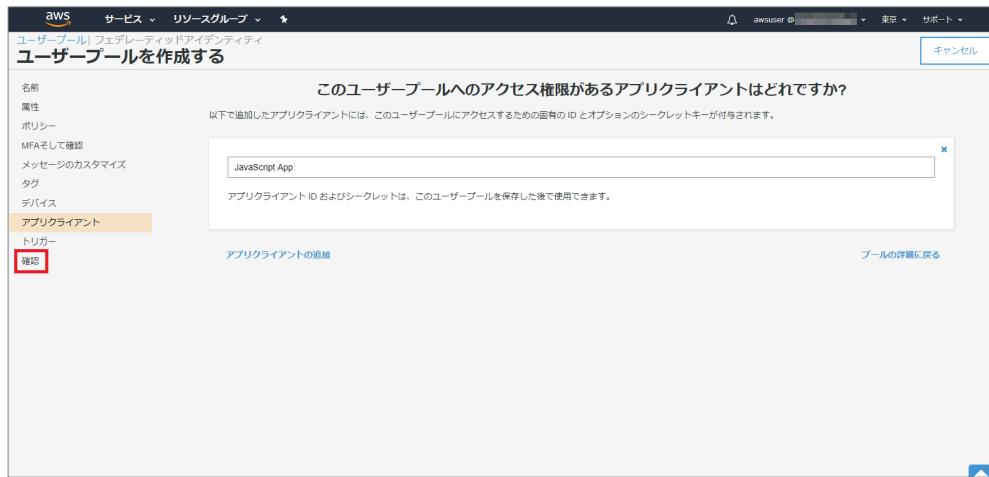
8. [アプリケート名] 欄に [JavaScript App] と入力します。

[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。

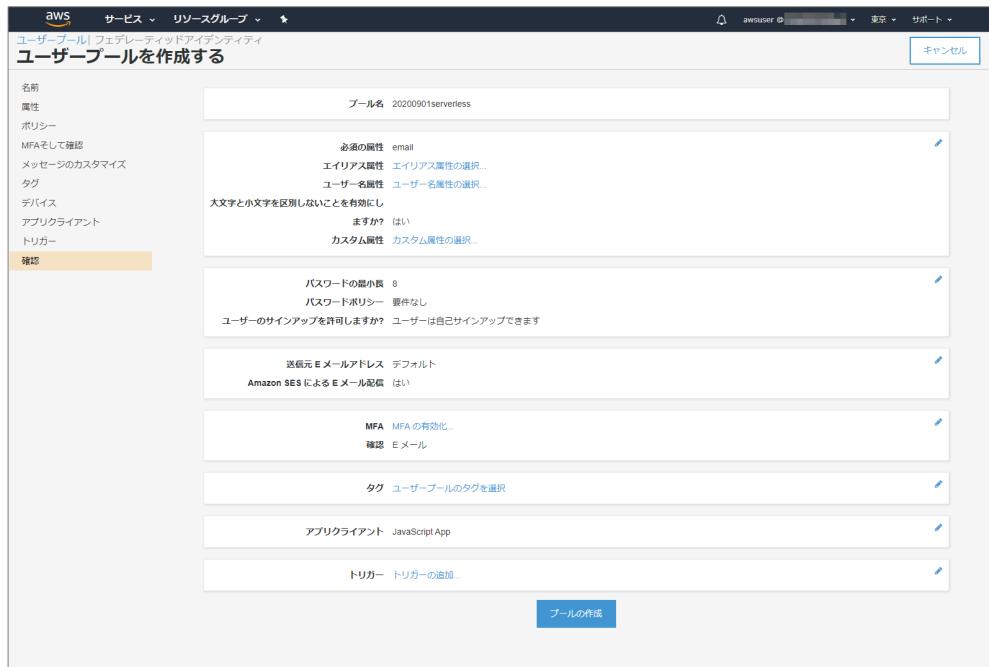


9. [アプリケーションの作成] をクリックします。

10. 画面左側のメニューから【確認】を選択します。

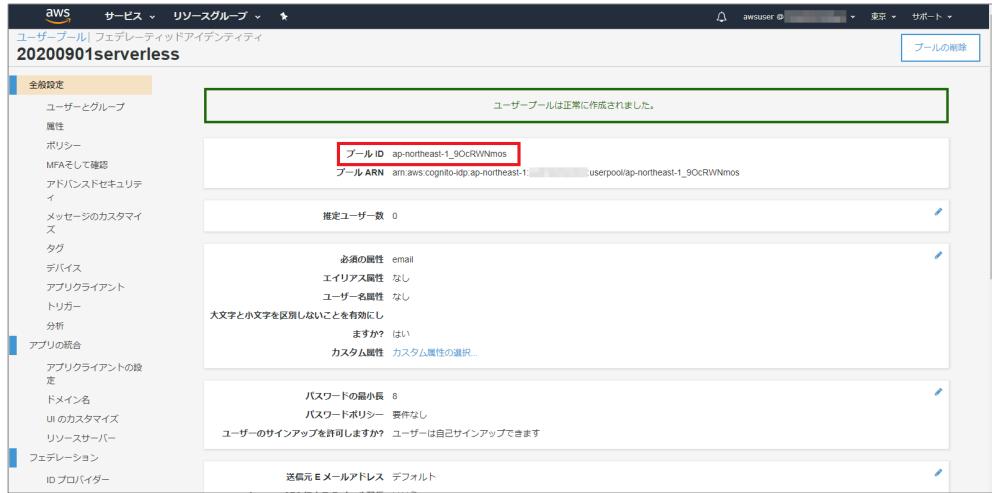


11. 確認画面になりますので、【ポールの作成】をクリックします。



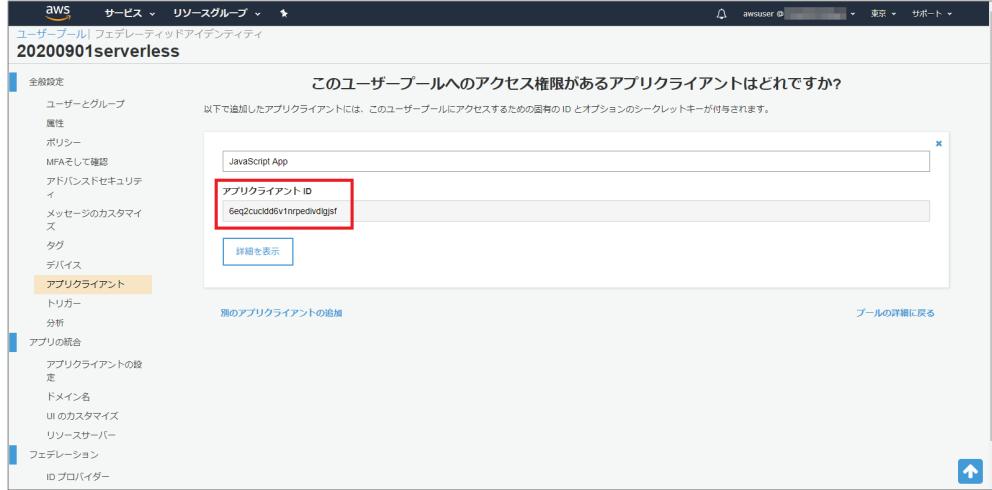
12. ユーザープールが作成されました。

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)



13. 画面左側のメニューから [アプリケイント] を選択します。

[アプリケイント 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おいてください)



#### 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 Amazon Cognito service page. At the top, there are tabs for 'User Pools' and 'ID Pools'. The 'ID Pools' tab is highlighted with a red box. Below the tabs, there are two sections: 'Sign-in and Sign-up' (with an icon of two user profiles) and 'AWS Service Access' (with an icon of a computer monitor and a cloud). The 'AWS Service Access' section contains text about using Cognito ID Pools for temporary AWS credentials.

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. It has two steps: 'Create ID Pool' and 'Configure access for your users'. In Step 1, the 'ID Pool Name' field is filled with '20200911serverless'. Below it, there's a note: 'Example: My App Name'. Under the heading 'Enable unauthenticated ID', there's a checkbox labeled 'Check to enable unauthenticated ID to have access'. This checkbox is checked. A note next to it says: 'Amazon Cognito is an identity provider that provides temporary AWS credentials to support unauthenticated users or sign-in users. If you enable this option, users can log in to your application without being prompted for a password. You can also grant them access to your application by enabling the "Allow unauthenticated ID to have access" option.' At the bottom, there are 'Cancel' and 'Create Pool' 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 'Create New Pool' step of the AWS IAM User Pools wizard. In the 'Authentication Provider' section, the 'Cognito' provider is selected. The 'User Pool ID' field contains 'ap-northeast-1\_9OcRWNmcs'. The 'Client ID' field contains '6eq2cuidd6v1mrpedivdglgf'. The '必須' (Required) label is present.

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。

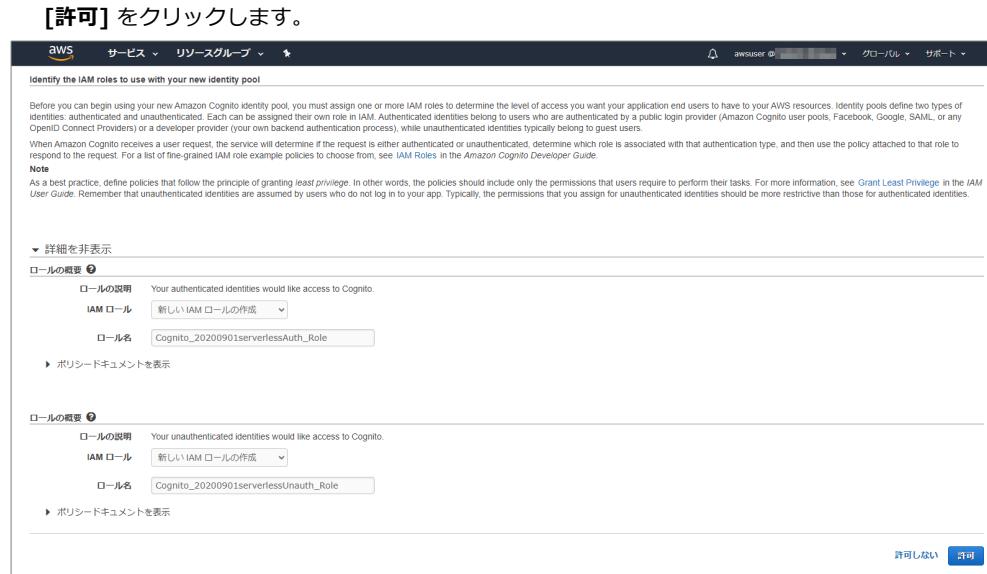
The screenshot shows the 'Create New Pool' step of the AWS IAM User Pools wizard. In the 'Authentication Provider' section, the 'Cognito' provider is selected. The 'User Pool ID' field contains 'ap-northeast-1\_9OcRWNmcs' and the 'Client ID' field contains '6eq2cuidd6v1mrpedivdglgf'. The '必須' (Required) label is present.

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。



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。



## 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)

Amazon Cognito での作業開始

プラットフォーム Android ▾

▼ AWS SDK のダウンロード

[AWS SDK for Android をダウンロード](#) 開発者ガイド

▼ AWS 認証情報の取得

```
// Amazon Cognito 認証情報プロバイダーを初期化します
CognitoIdentityProvider credentialsProvider = new CognitoCachingCredentialsProvider(
    "us-east-1:18954a9a-9d7b-463b-8f10-94cf30f678d8", // ID プールの ID
    regions.US_EAST_1);
regions.setDefaultRegion("us-east-1"); // リージョン
```

▼ 次に、認証情報プロバイダーを初期化します。

- Cognito ID での作業開始

[ダッシュボードに移動](#)

#### 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

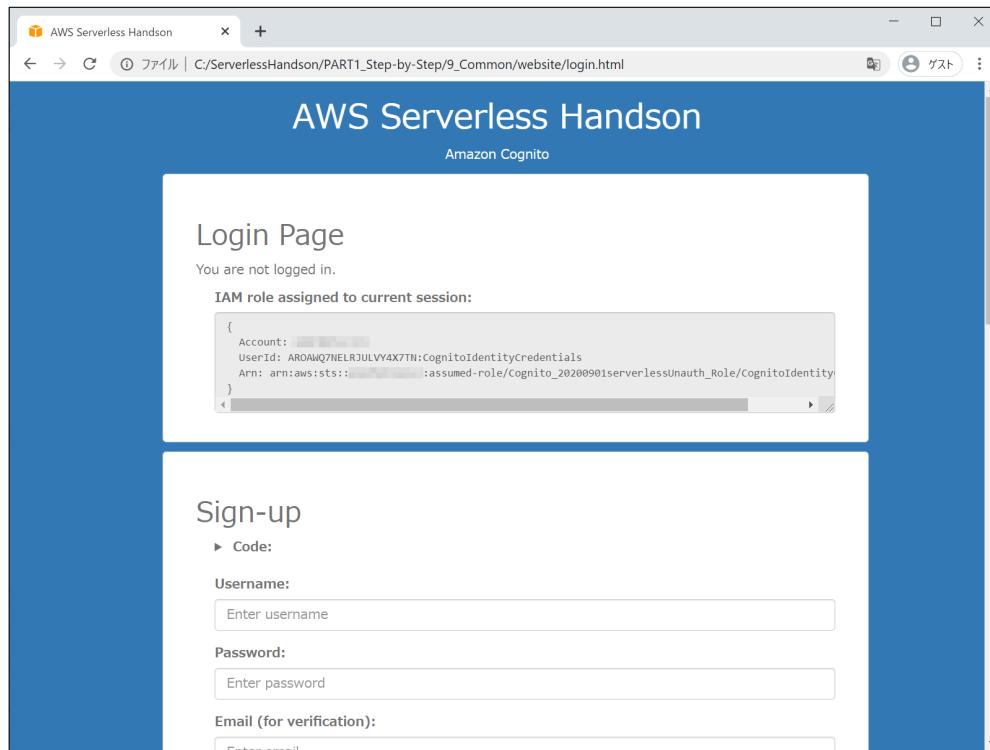
```
const CognitoConfig = {  
    region: 'ap-northeast-1',  
  
    // User Pool  
    userPoolId: 'ap-northeast-1 XXXXXXXXX',  
    appClientId: 'XXXXXXXXXXXXXXXXXXXXXX',  
  
    // Federated Identity  
    identityPoolId: 'ap-northeast-1:XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX',  
}
```

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 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
└ mypage.html ... ログイン処理が行われた後に遷移する Web ページ
```

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。）

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 browser window titled "Login Page". Below it, a message says "You are not logged in." A section titled "IAM role assigned to current session:" displays the following JSON object:

```
{  
  Account: [REDACTED]  
  UserId: AROAWQ7NELRJULVY4X7TN:CognitoIdentityCredentials  
  Arn: 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 a "Sign-up" form. It includes fields for "Code" (with a "▶" icon), "Username" (containing "user1"), "Password" (containing "\*\*\*\*\*"), "Email (for verification)" (containing "user1@example.com"), and a "Sign-up" button. Below the form is a "Result:" section with a large, empty text area.

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 21:55:18 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。

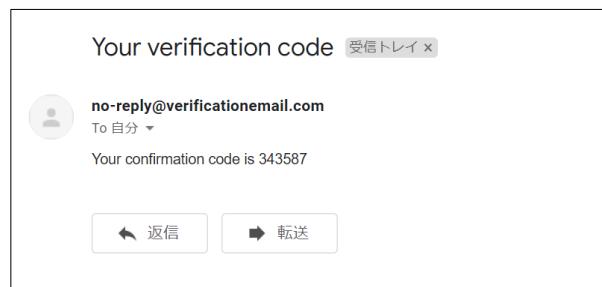
サインアップを行ったユーザーが登録されていることが確認できます。（表示されない場合は、右上のリロードボタンをクリックして表示を更新してください）

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。

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	更新済み	作成日
user1	Enabled	UNCONFIRMED	false	-	Sep 14, 2020 12:55:17 PM	Sep 14, 2020 12:55:17 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 screenshot shows an "Activation" form. It has fields for "Code:" (Username: user1, Confirmation Code: 343587), an "Activate" button, and a "Result:" section which is currently empty.

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:  
Enter username

Confirmation Code:  
Enter 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。

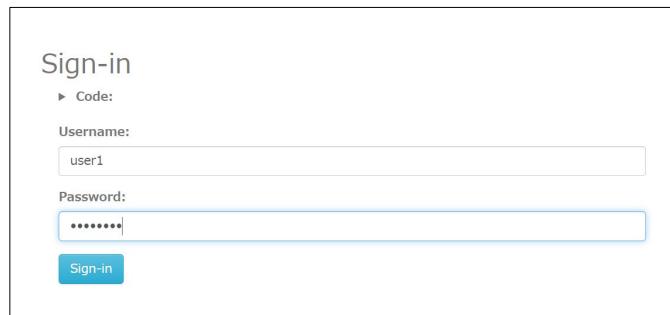
アカウントのステータスが [CONFIRMED] (確認済み) に変わっていることを確認します。

The screenshot shows the AWS Cognito User Pools interface. On the left, there's a sidebar with options like '全般設定', '属性', 'ポリシー', 'MFA そして 確認', 'アドバイスとセキュリティ', 'メッセージのカスタマイズ', 'タグ', 'デバイス', 'アプリケーション', and 'トリガー'. The main area has tabs for 'ユーザー' and 'グループ', with 'ユーザー' selected. There are buttons for 'ユーザーをインポート' and 'ユーザーの作成'. A search bar labeled 'User name' is present. Below these, a table lists users. The first row shows a user named 'user1' with the status 'CONFIRMED' highlighted in a red box. Other columns in the table include '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', '更新済み', and '作成日'.

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	更新済み	作成日
user1	Enabled	CONFIRMED	true	-	Sep 14, 2020 1:01:57 PM	Sep 14, 2020 12:55:17 PM

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

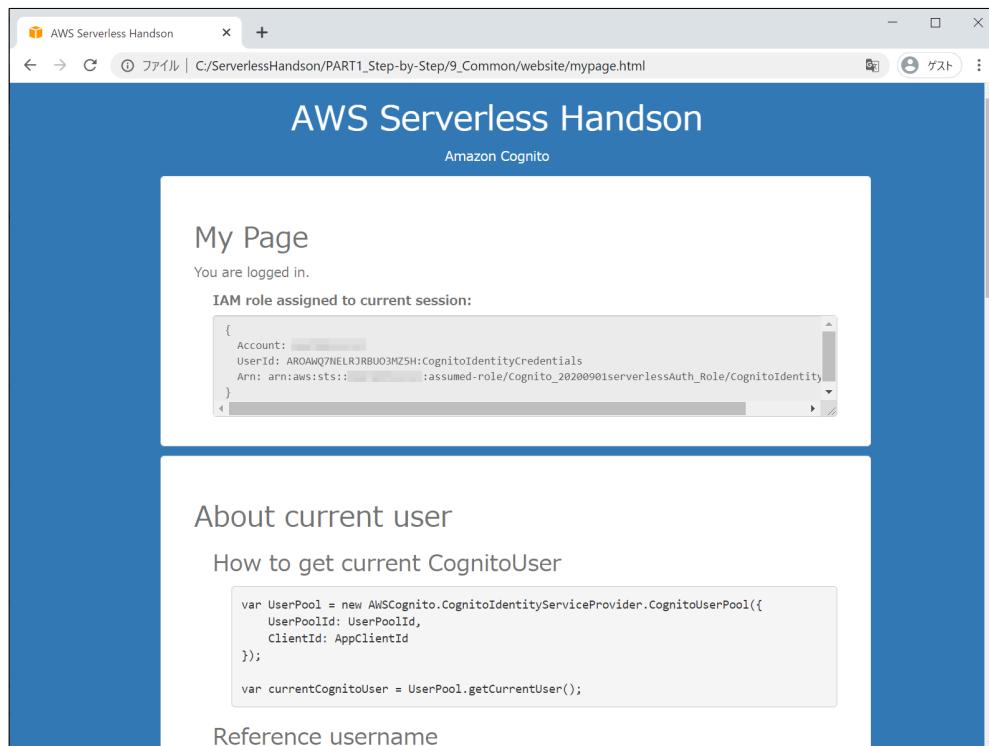
ユーザー名とパスワードを入力して、[Sign-in] をクリックします。



The screenshot shows a 'Sign-in' form with the following fields:

- Code: (dropdown menu)
- Username: user1
- Password: (redacted)
- 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 browser window titled 'AWS Serverless Handson' with the URL 'C:/ServerlessHandson/PART1\_Step-by-Step/9\_Common/website/mypage.html'. The page displays the following content:

## AWS Serverless Handson

Amazon Cognito

### My Page

You are logged in.

IAM role assigned to current session:

```
{  
  Account: [REDACTED]  
  UserId: AROAWQ7NELRJRB03M25H:CognitoIdentityCredentials  
  Arn: arn:aws:sts::[REDACTED]:assumed-role/cognito_20200901serverlessAuth_Role/CognitoIdentity
```

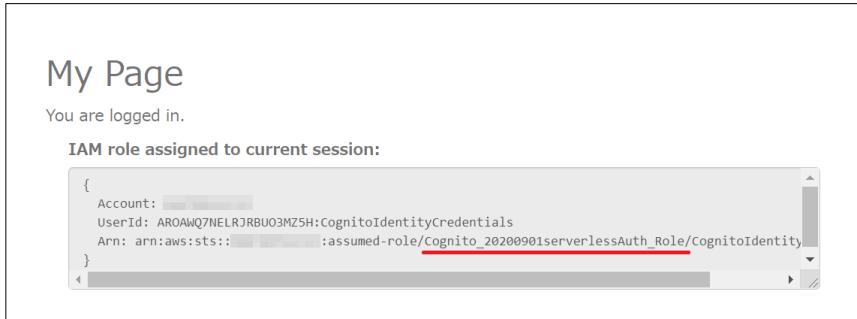
### About current user

How to get current CognitoUser

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

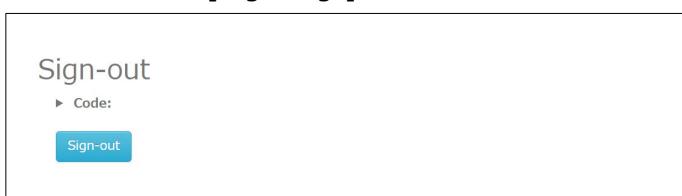
Reference username

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

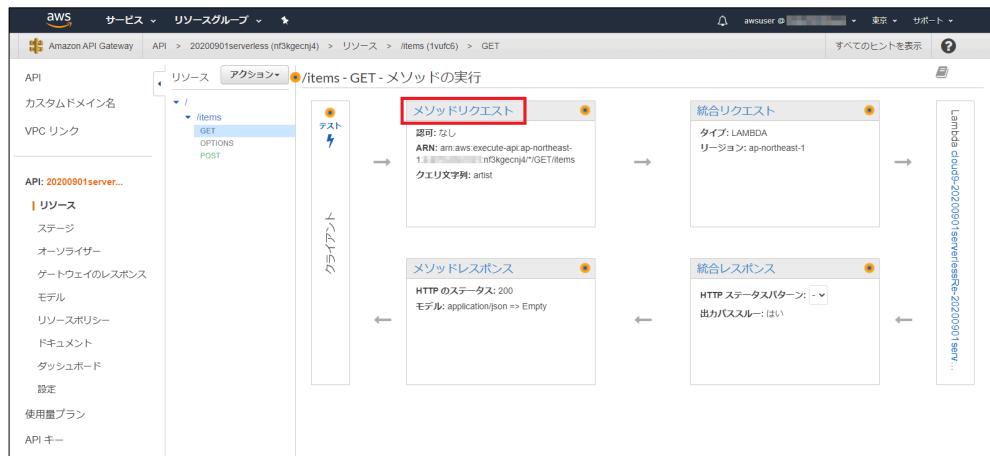
18. 画面を一番下までスクロールすると [Sign-out] ボタンがあります。  
サインアウトすると [Login Page] に戻ります。



## 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)



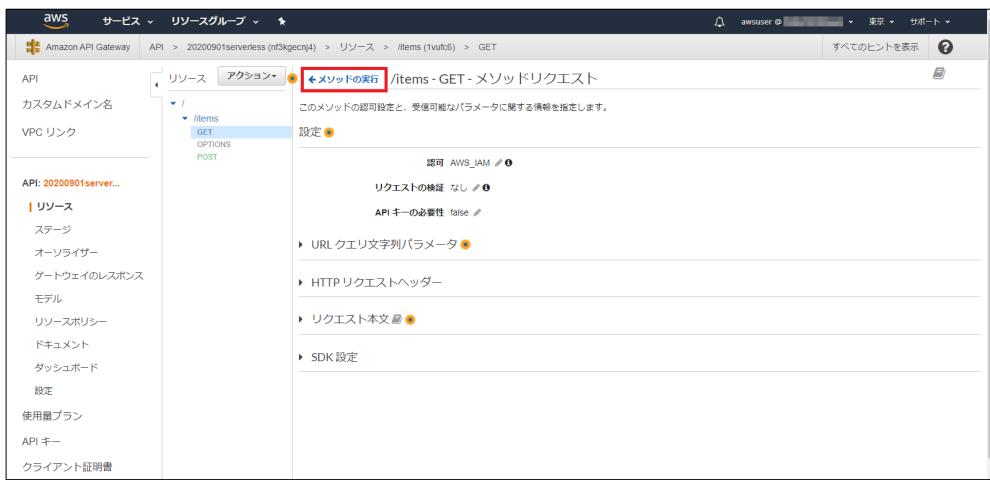
2. [認可] の右側にある鉛筆ボタンをクリックします。

The screenshot shows the AWS API Gateway console. On the left, there's a navigation sidebar with options like 'API', 'カスタムドメイン名', 'VPC リンク', and 'API: 20200901server...'. The main area shows a tree structure with 'リソース' expanded, showing '/Items' and its 'アクション' (Actions). Under 'Actions', 'GET' is selected. A callout box highlights the '認可' (Authorization) dropdown, which is currently set to 'なし' (None).

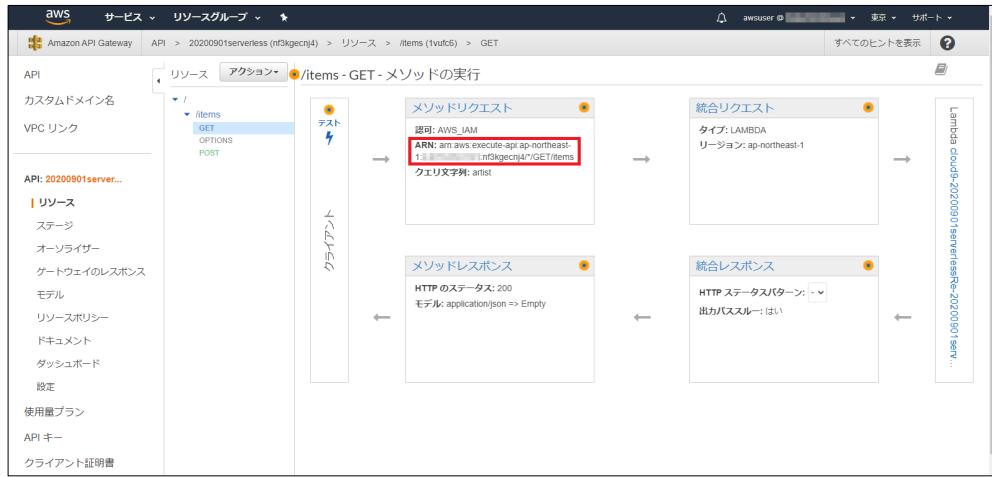
3. プルダウンから [AWS\_IAM] を選択して、右側のチェックボタンをクリックして確定します。

This screenshot is identical to the one above, but the '認可' (Authorization) dropdown has been changed to 'AWS\_IAM'. A red box highlights the dropdown, and another red box highlights the checked checkbox next to 'AWS\_IAM'.

4. 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。

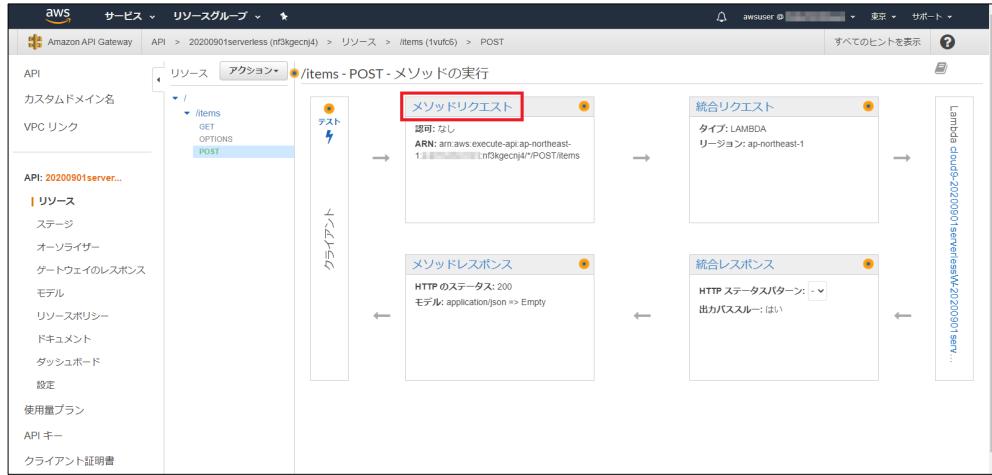


5. GET メソッド呼び出しの ARN が表示されています。後の手順で使用するためメモしておいてください。



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。



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 navigation options like 'API', 'リソース', 'アクション', '設定', etc. The main area shows a tree structure for an API named '20200901serverless'. Under 'リソース', there's a 'POST' method selected. In the center, there's a section titled 'メソッドの実行 /Items - POST - メソッドドリクエスト'. Under '認可', it says 'なし' (None) and has a pencil icon to its right. Other sections include 'リクエストの検証 なし', 'APIキーの必要性 false', and several expandable sections for URL parameters, HTTP headers, request body, and SDK settings.

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 to 'AWS\_IAM'. The pencil icon is still present next to the dropdown. The rest of the interface remains the same, showing the execution details and various configuration options for the POST method.

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。

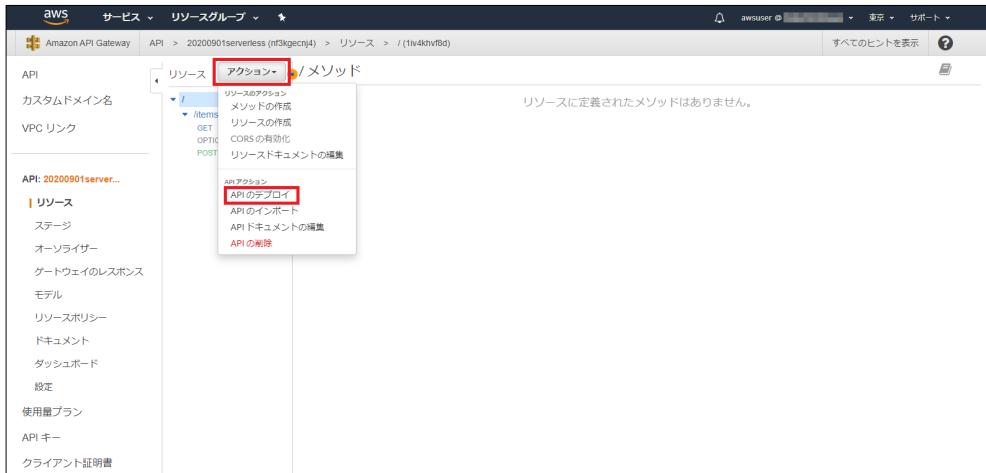
The screenshot shows the AWS API Gateway console. On the left, the navigation pane includes 'API', 'カスタムドメイン名', 'VPC リンク', and a list of API resources like 'API: 20200901server...'. The main area shows the 'POST' method configuration for the '/items' resource. A red box highlights the '← メソッドの実行' button at the top left of the configuration panel. The configuration panel itself contains sections for '設定' (Authorization), 'リクエストの検証' (Request Validation), 'API キーの必要性' (API Key Required), 'URL クエリ文字列/パラメータ' (Query String Parameters), 'HTTP リクエストヘッダー' (Request Headers), 'リクエスト本文' (Request Body), and 'SDK 設定' (SDK Settings).

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。

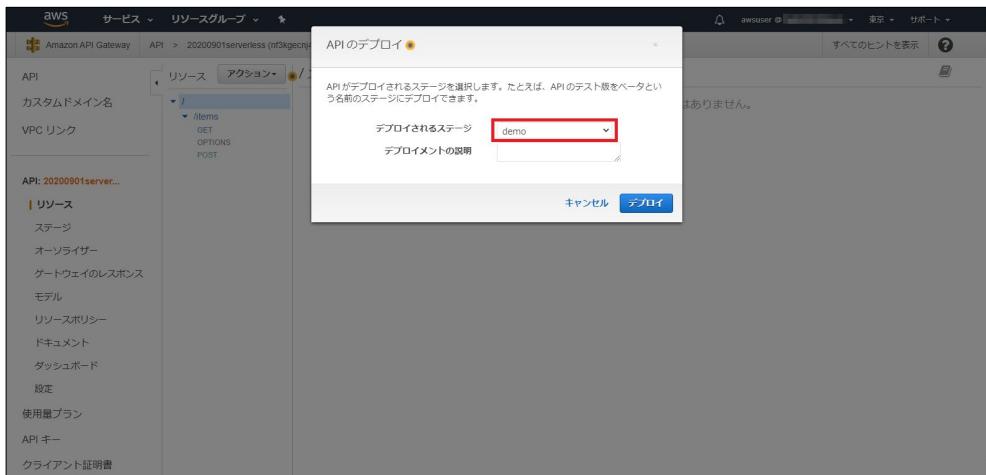
This screenshot shows the execution of the POST method for the '/items' endpoint. The 'Test' button in the configuration panel has been clicked, and the results are displayed in four boxes: 'メソッド実行' (Method Execution) which shows the ARN 'arn:aws:execute-api:ap-northeast-1:nf3kgecnj4:POST:items'; '統合リクエスト' (Integrated Request) which shows the Lambda function details; 'メソッドレスポンス' (Method Response) which shows the response status '200' and model 'application/json => Empty'; and '統合レスポンス' (Integrated Response) which shows the response status '200' and model 'application/json => Empty'. A red box highlights the ARN value in the 'メソッド実行' box.

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 interface. On the left, a sidebar lists various API settings like 'API', 'リソース', 'ステージ', etc. The main area is titled 'demo ステージエディター'. It displays the URL 'https://nf3kgecnj4.execute-api.ap-northeast-1.amazonaws.com/demo'. Below this, there are tabs for '設定' (Configuration), 'ログトレース' (Logs), 'ステージ要数' (Stage Count), 'SDK の生成' (Generate SDK), 'エクスポート' (Export), 'デプロイ履歴' (Deployment History), 'ドキュメント履歴' (Documentation History), and 'Canary'. The '設定' tab is selected. Under '設定', there's a section for 'キャッシュ設定' (Cache Settings) with a checkbox for 'API キャッシュを有効化' (Enable API Cache). Below that is a section for 'デフォルトのメソッドスロットリング' (Default Method Slot Limiting) with a note about account-level limits. There's also a section for 'スロットリングの有効化' (Slot Limiting Enabled) with rate and burst settings: 'レート' (Rate) is set to 10000 requests per second, and 'バースト' (Burst) is set to 5000 requests. At the bottom, there's a note about WAF integration and a link to 'ウェブ ACL を作成する' (Create Web ACL).

#### 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 IAM Policies creation interface. The top navigation bar includes 'awsuser' and 'グローバル' (Global). The left sidebar lists 'Identity and Access Management (IAM)' services: ダッシュボード, アクセス管理, グループ, ユーザー, ロール, **ポリシー**, ID プロバイダー, and AWS Organizations. The main area has tabs 'ポリシーの作成' (highlighted with a red box) and 'ポリシーアクション'. A search bar and a table with 732 results are present. The table columns are 'ポリシーネーム' (Policy Name), 'タイプ' (Type), '次として使用' (Use as Next), and '説明' (Description). Several policies are listed, such as '20200901serverlessLambdaPolicy', 'AccessAnalyzerServiceRolePolicy', 'AdministratorAccess', 'AlexaForBusinessDeviceSetup', and 'AlexaForBusinessFullAccess'.

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. It displays instructions about defining policies for users, groups, or roles. It includes tabs for 'Visual Editor' (selected) and 'JSON'. Below are buttons for 'Manage Policy Import' and 'Import Policy'. The 'Services Selection' section is expanded, showing a list of services. The 'Services' tab is highlighted with a red box. Other tabs include 'Actions', 'Resources', and 'Request Conditions'.

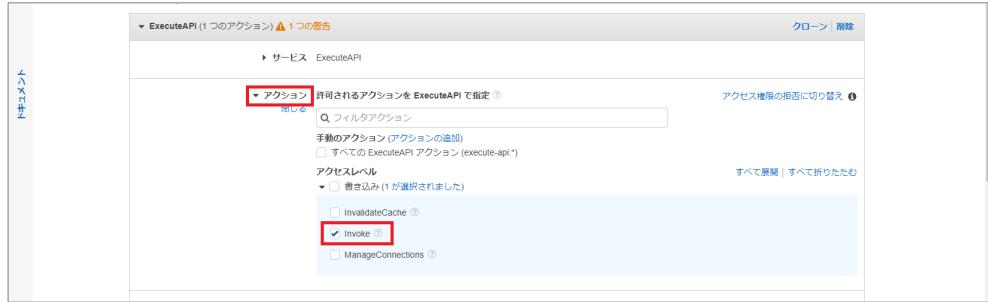
3. 検索欄に [executeapi] と入力します。

表示された [ExecuteAPI] をクリックします

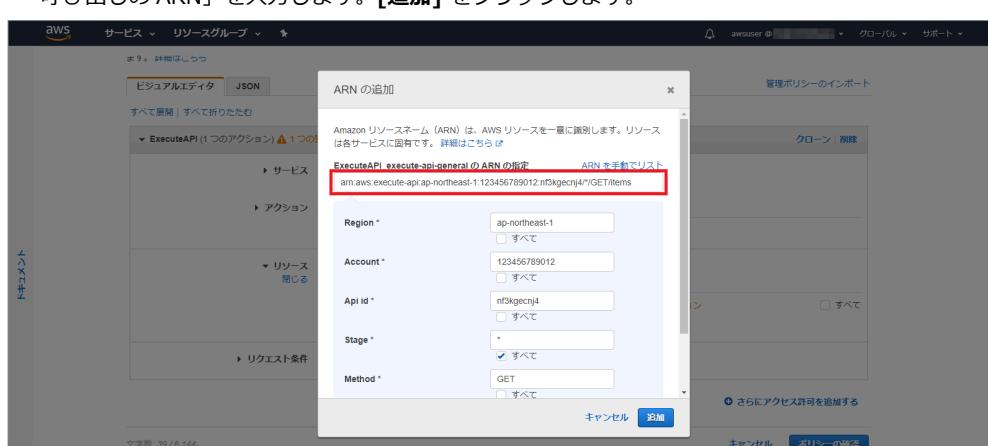


The screenshot shows the 'Services Selection' dropdown expanded. A search bar contains 'executeapi'. The 'ExecuteAPI' service is listed and highlighted with a red box. Other services like 'Amazon CloudWatch Metrics' and 'Amazon CloudWatch Metrics Insights' are also visible.

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。

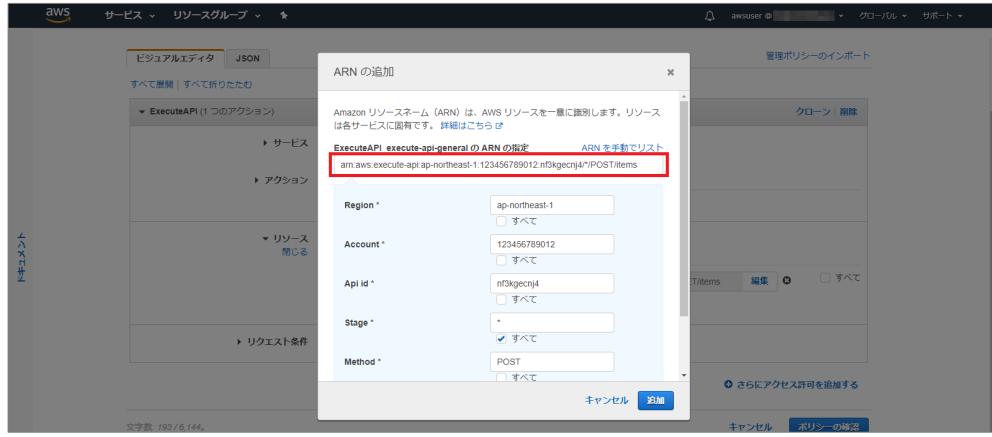


6. GET メソッド呼び出しの ARN が追加されたことを確認します。

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)

The screenshot shows the 'Policy creation' wizard step 2. The 'Policy name' field contains '20200901serverlessAPIGatewayGetAndPostPolicy'. The 'Description' field is empty. The 'Summary' section shows the following permissions:

サービス	アクセスレベル	リソース	リクエスト条件
許可 (239 サービス中 1) 残りの 238 を表示	ExecuteAPI	制限 書き込み	権限 なし

At the bottom, there are buttons for 'キャンセル' (Cancel), '戻る' (Previous), and 'ポリシーの作成' (Create Policy).

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。

The screenshot shows the 'Identity and Access Management (IAM)' service's 'Policies' page. A success message '20200901serverlessAPIGatewayGetAndPostPolicy が作成されました。' is displayed. The table lists policies, with the newly created one highlighted:

ポリシー名	タイプ	次として使用	説明
20200901serverlessAPIGatewayGetAndPostPolicy	ユーザーによる管理	なし	
20200901serverlessLambdaPolicy	ユーザーによる管理	Permissions policy (1)	
AccessAnalyzerServiceRolePolicy	AWS による管理	なし	Allow Access Analyzer to analyze resource metadata
AdministratorAccess	ショット機能	Permissions policy (3)	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 AW...
AlexaForBusinessGatewayExec...	AWS による管理	なし	Provide gateway execution access to AlexaForBusiness services
AlexaForBusinessLifesizeDeleg...	AWS による管理	なし	Provide access to Lifesize AVS devices

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。

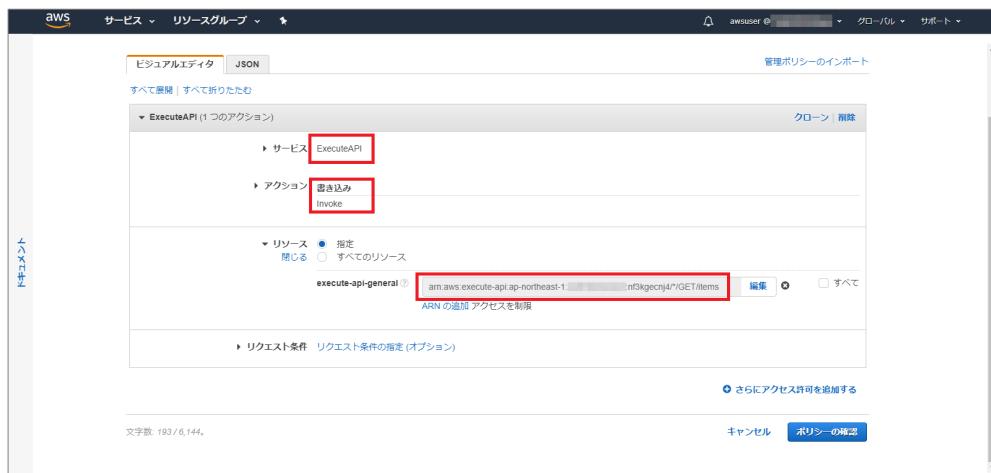
```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Sid": "VisualEditor0",  
      "Effect": "Allow",  
      "Action": "execute-api:Invoke",  
      "Resource": [  
        "arn:aws:execute-api:ap-northeast-1:123456789012:nf3kgecnj4/*/GET/items",  
        "arn:aws:execute-api:ap-northeast-1:123456789012:nf3kgecnj4/*/POST/items"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

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。



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)



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)

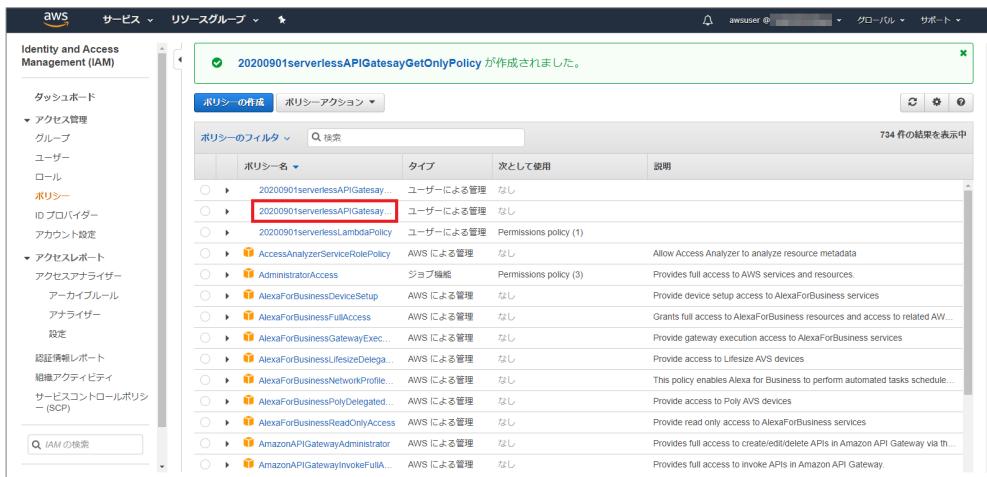


18. [ポリシーの作成] をクリックします。

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



20. IAM ポリシーの定義内容を JSON 形式で表示します。

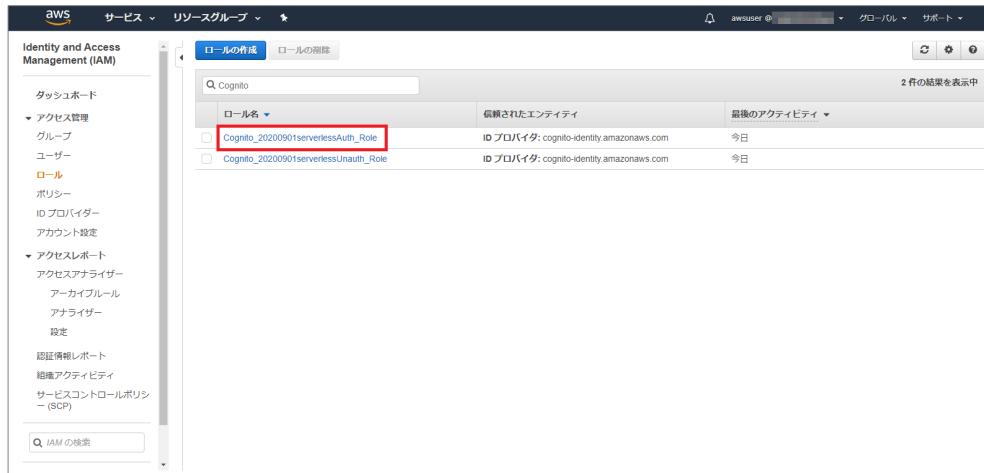
以下のような定義内容となっていることを確認します

```
{  
  "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。

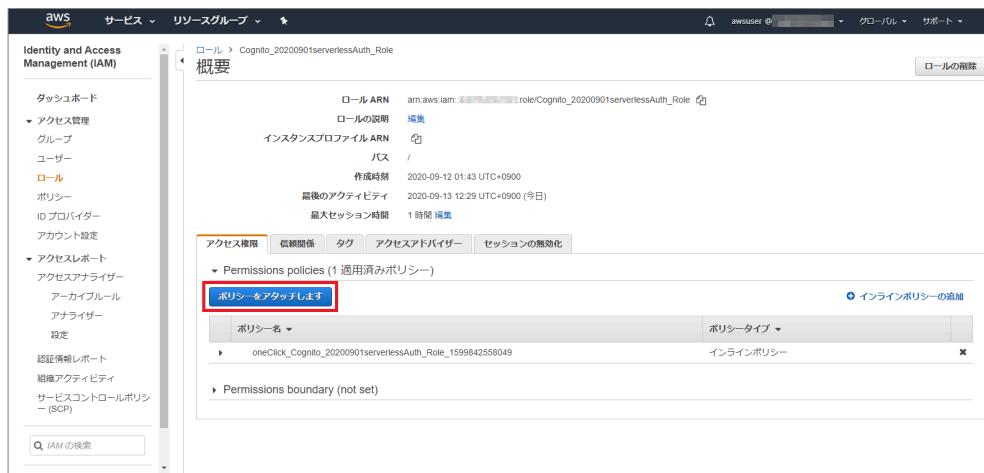
[Cognito\_YYYYMMDDserverlessAuth\_Role] (認証された ID 用のロール) をクリックします。



The screenshot shows the AWS IAM service dashboard. On the left, there's a sidebar with various navigation options like Groups, Users, Roles, Policies, and Access Reports. The main area is titled 'Identity and Access Management (IAM)' and shows a search bar with 'Cognito'. Below it, a table lists two roles:

ロール名	信頼されたエンティティ	最後のアクティビティ
Cognito_20200901serverlessAuth_Role	ID プロバイダ: cognito-identity.amazonaws.com	今日
Cognito_20200901serverlessUnauth_Role	ID プロバイダ: cognito-identity.amazonaws.com	今日

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'. It includes fields for Role ARN, Description, and Instance Profile ARN. Under the 'Permissions policies' section, there is a button labeled 'ポリシーをアタッチします' (Attach policy), which is highlighted with a red box.

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 Policy Attachments interface. A search bar at the top right contains the text 'YYYYMMDDserverlessAPIGatewayGetAndPostPolicy'. Below it is a table with columns 'ポリシー名' (Policy Name), 'タイプ' (Type), and '次として使用' (Used as Next). The first row, '20200901serverlessAPIGatewayGetAndPostPolicy', has a checked checkbox in the 'ポリシーの作成' (Create Policy) column and is highlighted with a red box. Other rows in the table represent various AWS managed policies.

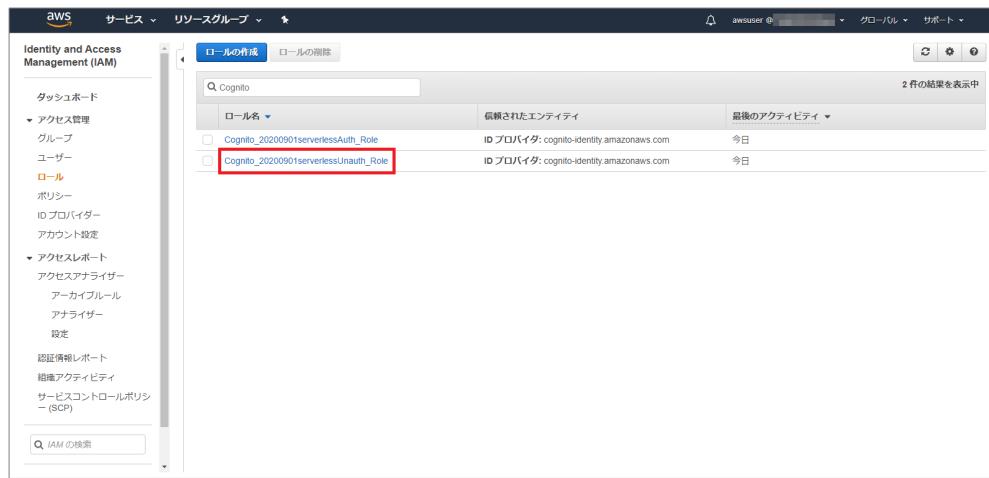
24. [ポリシーのアタッチ] をクリックします。

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 AWS IAM Role Overview interface. On the left, a sidebar menu is open under 'Identity and Access Management (IAM) > ロール'. The main area displays the role 'Cognito\_20200901serverlessAuth\_Role' with its ARN and creation date. A green message box indicates that the policy '20200901serverlessAPIGatewayGetAndPostPolicy' has been attached. Below this, the 'Permissions policies' section lists the attached policies: '20200901serverlessAPIGatewayGetAndPostPolicy' and 'oneClick\_Cognito\_20200901serverlessAuth\_Role\_1599842558049'. A blue button labeled 'ポリシーをアタッチします' (Attach Policy) is visible at the bottom of the policy list.

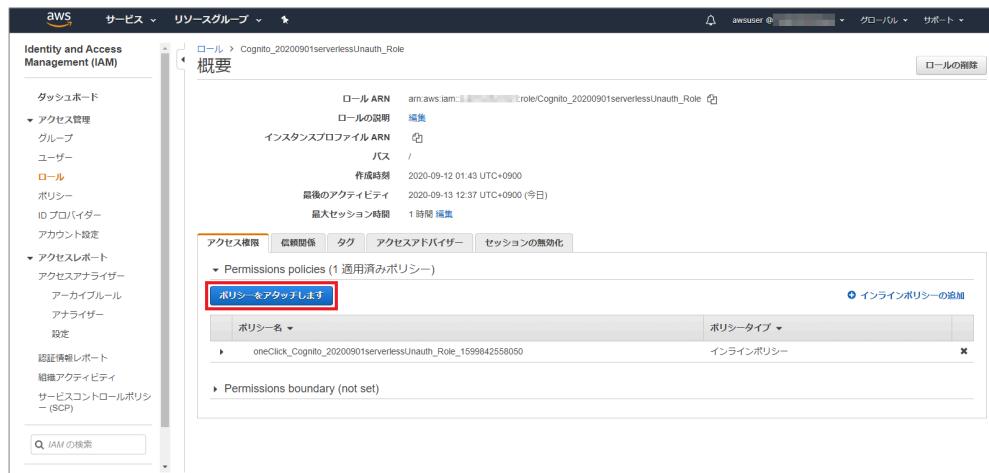
26. [IAM ロール] の画面に戻ります。

[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 Identity and Access Management (IAM) service interface. The left sidebar is collapsed. The main area displays a list of roles under the heading 'ロール'. There are two entries: 'Cognito\_20200901serverlessAuth\_Role' and 'Cognito\_20200901serverlessUnauth\_Role'. The second entry is highlighted with a red box. The top navigation bar includes the AWS logo, service dropdown, resource groups dropdown, and support dropdown.

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' role. The left sidebar is collapsed. The main area has a tab bar with '概要' (Overview) selected. Below it, there are sections for 'Role ARN', 'Role description', 'Instance profile ARN', 'Creation time', and 'Last activity'. Under the 'Permissions policies' section, there is a single inline policy named 'oneClick\_Cognito\_20200901serverlessUnauth\_Role\_1599842558050'. A red box highlights the 'Attach policy' button at the bottom of this section. The top navigation bar includes the AWS logo, service dropdown, resource groups dropdown, and support dropdown.

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 service interface. A search bar at the top contains the text "YYYYMMDDserverlessAPIGatewayGetOnlyPolicy". Below it, a table lists policies. One policy, "20200901serverlessAPIGatesayGetOnlyPolicy", is selected and highlighted with a red box. The table includes columns for "ポリシー名" (Policy Name), "タイプ" (Type), and "次として使用" (Used as Next). The "タイプ" column shows "Permissions policy (1)" for the selected policy.

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 service interface, specifically the "Role Overview" page for the role "Cognito\_20200901serverlessUnauth\_Role". The left sidebar shows the navigation menu for Identity and Access Management (IAM). The main pane displays the attached policy "20200901serverlessAPIGatesayGetOnlyPolicy". The policy details show it was attached on 2020-09-12 01:43 UTC+0900. The "Permissions policies" section indicates "(2 適用済みポリシー)" (2 policies applied) and shows the two attached policies: "20200901serverlessAPIGatesayGetOnlyPolicy" and "oneClick\_Cognito\_20200901serverlessUnauth\_Role\_1599842558050".

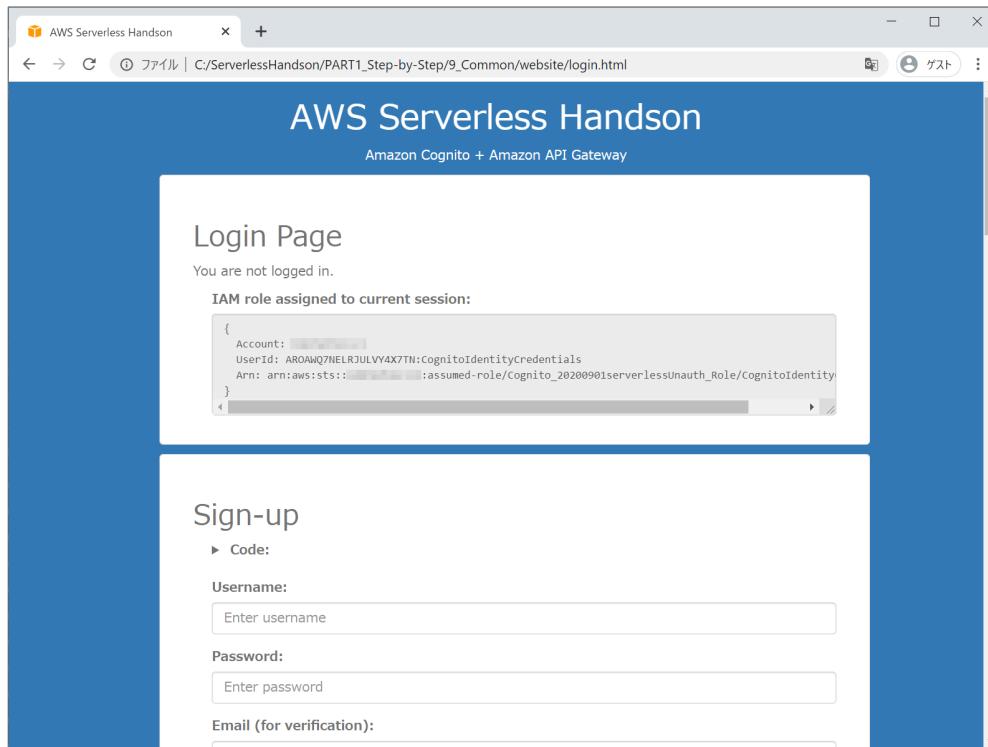
#### 4.4.3. Web ブラウザからの動作確認

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. この時点では、**[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client のクラス定義
    │   ├── [awssdk]
    │   └── [cognito]
└── [style] ... CSS ファイル
└── apigateway.js ... API Gateway 構造の JavaScript コード (Cognito 認証処理を追加)
└── cognito.js ... Cognito へ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:

**Run**

Status Code:

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:

Title:

**Run**

Status Code:

Response Data:

```
{"message": "User: arn:aws:sts::██████████:assumed-role/cognito_20200901serverlessunauth_Role/cognitoIdentityCredentials is not authorized to perform: execute-api:Invoke on resource: arn:aws:execute-api:ap-northeast-1:████████████:fn3kgecnj4/demo/POST/items"}
```

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/CD 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。

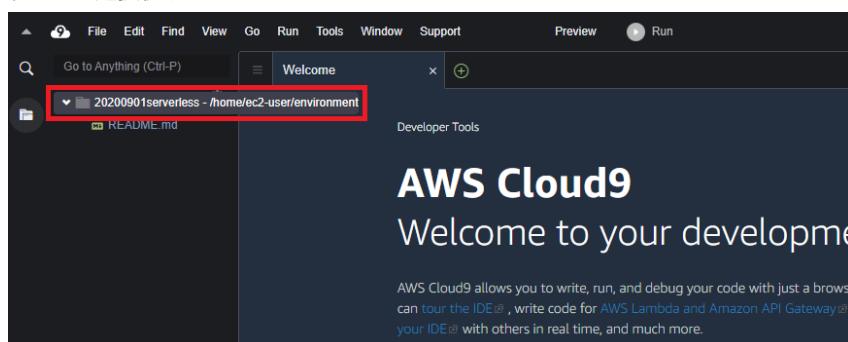
これから実施するラボ 2 ではディスク容量が不足する可能性がありますので、予めディスク容量を拡張しておきます。

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。

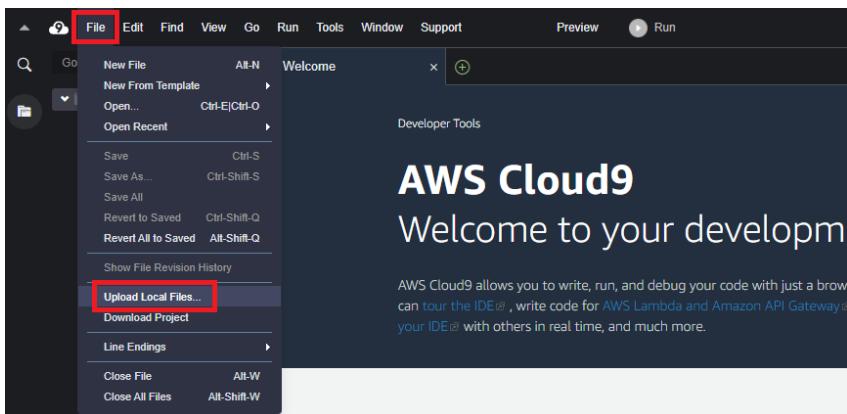
- [resize.sh]

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。

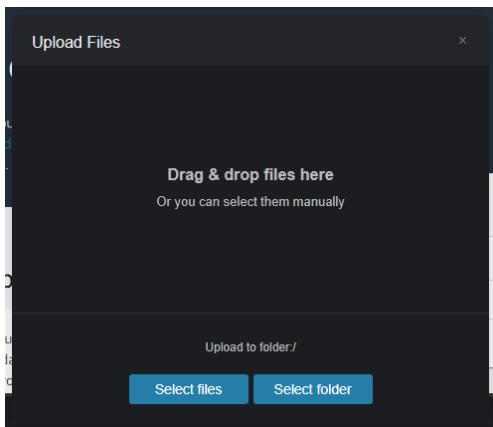


2. 画面上部のメニューから [File]-[Upload Local Files...] を選択します。



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]:
```

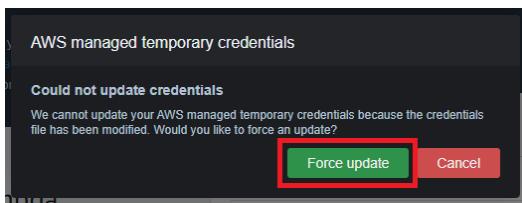
3. 「Default output format」に対して **[json]** と入力して Enter キーを押します。

```
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:8080

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

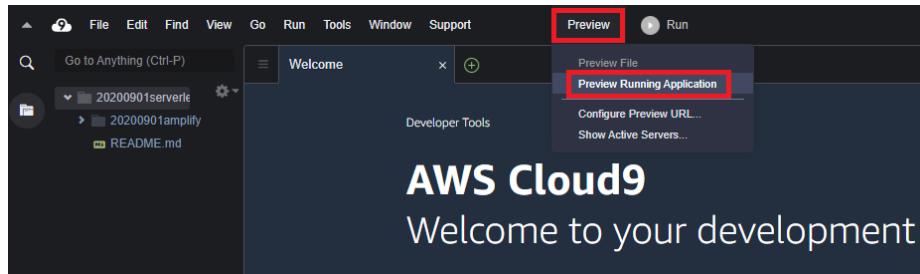
5. 起動したアプリケーションに接続を行ってみます。

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

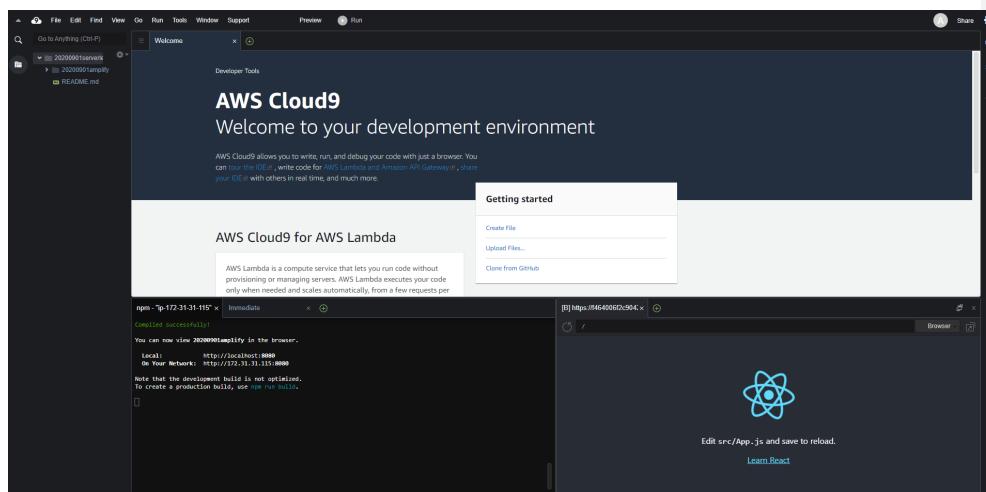
「<http://localhost:8080>」と入力して接続を試みても、接続することはできません。

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。

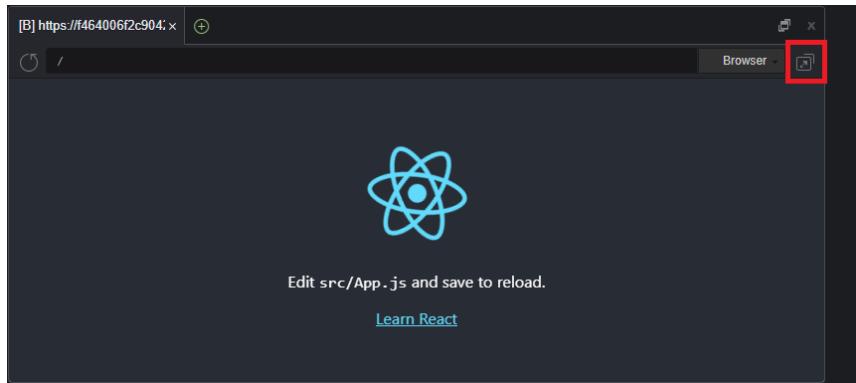
画面上部の [Preview] をクリックして [Preview Running Application] を選択します。



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、実行に関する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: (dynamodbXXXXXXXXXX) 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。

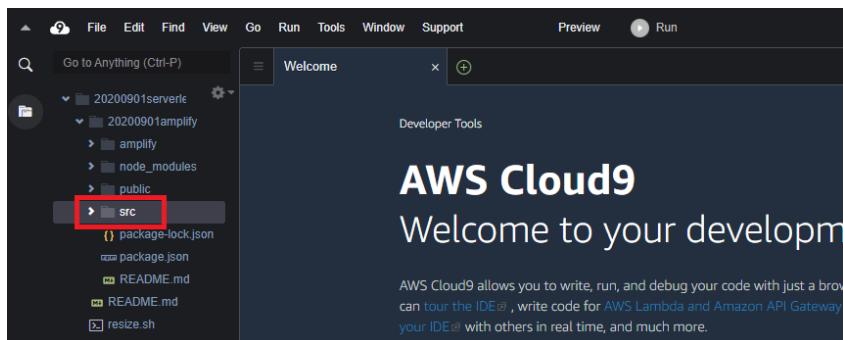
## 5.2.6. Web クライアントアプリケーションの構成

1. [src] フォルダに以下のファイルがあります。

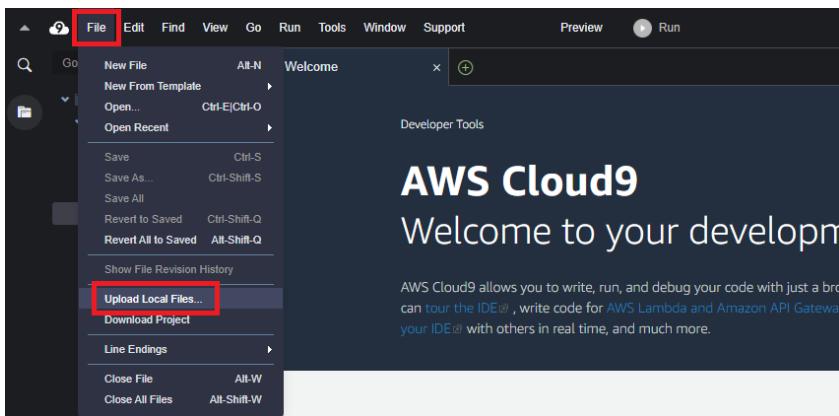
- [ApiGet.js]
- [ApiPost.js]
- [App.css]
- [App.js]
- [aws-logo.png]
- [config.js]

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。

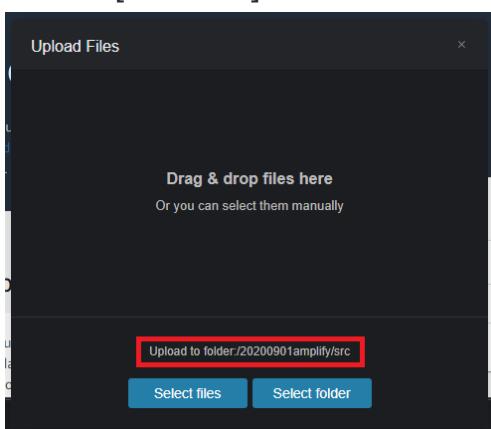


3. 下図のウィンドウが表示されます。

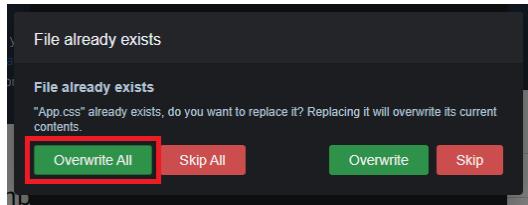
アップロード先ディレクトリが「/YYYYMMDDamplify/src」であることを確認します。

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。

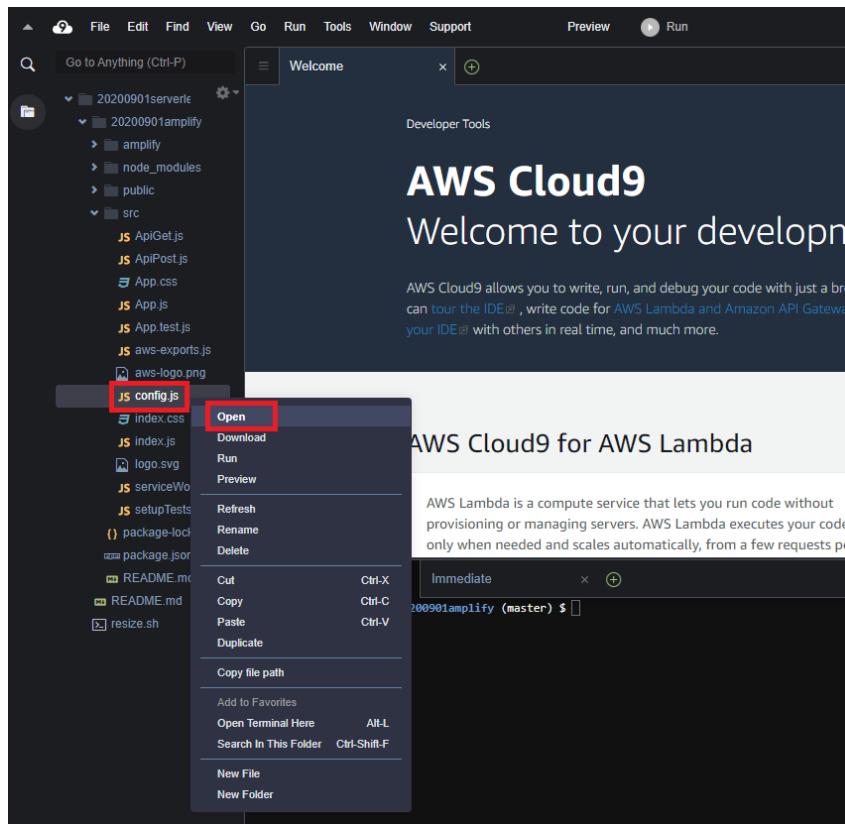


5. ls コマンドを実行して、ファイルがアップロードされたことを確認します。

```
$ 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）

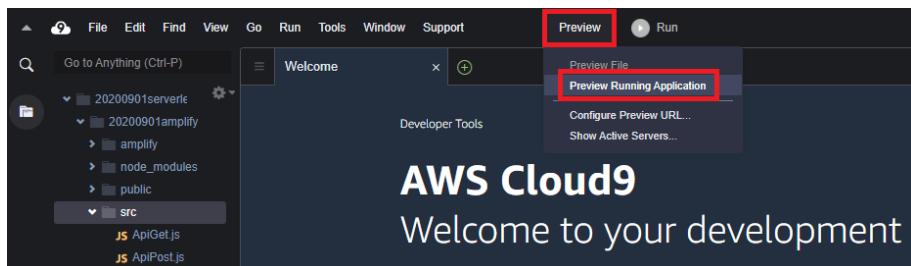
```
Compiled successfully!

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。

**AWS Serverless Handson**  
Built using AWS Amplify

**GET method**

Artist:

**GET**

Response:

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

Artist	Title
Michael Jackson	Billie Jean
Michael Jackson	Thriller

**POST method**

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)

**GET method**

Artist:

**GET**

Response:

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

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)
>❶ create
❷ read
❸ update
❹ 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	20200901amplifyXXXXXXX	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
  - **YYYYMMDD**amplifyXXXXXXXXX\_userpool\_XXXXXXX-demo
- Cognito ID プール
  - **YYYYMMDD**amplifyXXXXXXXXX\_identitypool\_XXXXXXX\_demon
- IAM ロール
  - snsXXXXXXXXXXXXXX-demon
  - upClientLambdaRoleXXXXXXXXXXXXXX-demon
- Lambda 関数
  - amplify-**YYYYMMDD**amplify-demo-UserPoolClientLambda-XXXXXXXXXXXXXX

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
  - **YYYYMMDD**amplify
- Lambda 関数
  - **YYYYMMDD**amplify-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-**YYYYMMDD**amplify-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

1. 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。

2. 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。

3. 次に、ファイルの 16 行目付近にある「<div className="App">」の行の後に、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。

4. 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

1. 以下のコマンドを実行します。

```
$ npm start
```

2. 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!

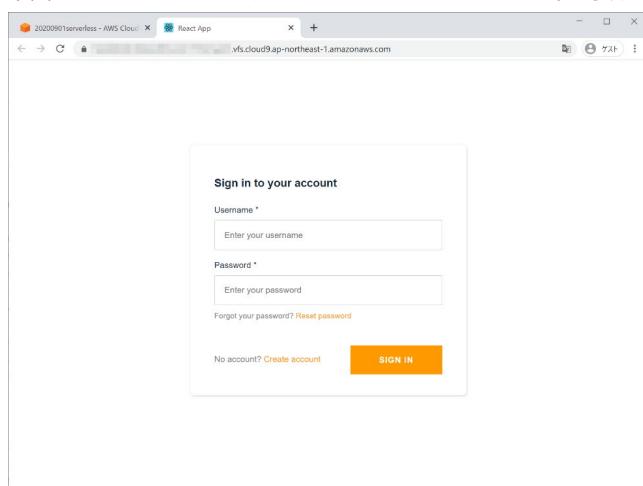
You can now view 20200901amplify in the browser.

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

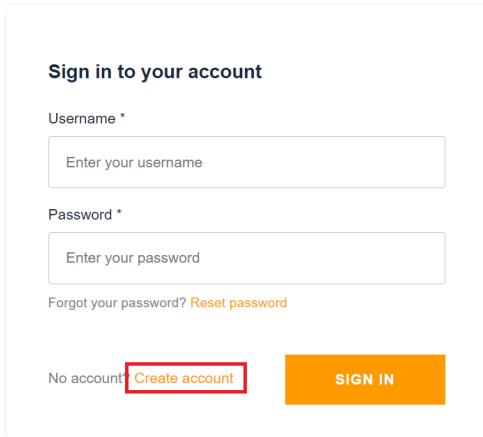
Note that the development build is not optimized.
To create a production build, use npm run build.
```

3. 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。

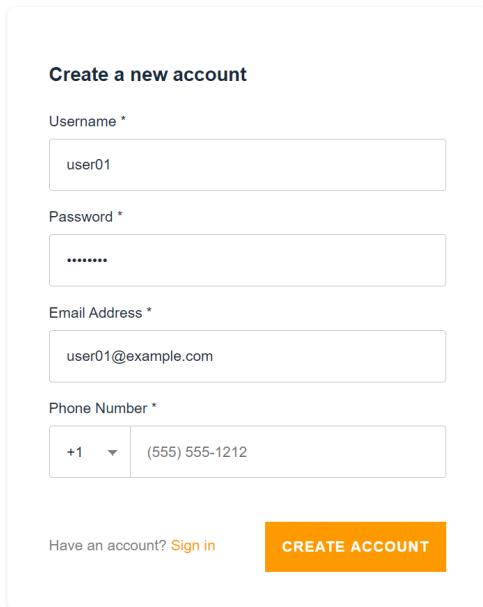


5. まず、[Create account] をクリックしてサインアップを行います。



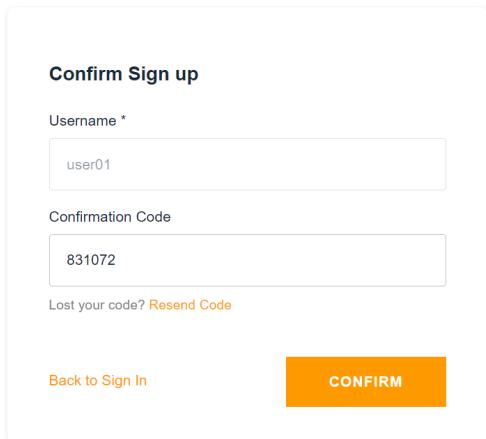
The image shows a sign-in form titled "Sign in to your account". It contains fields for "Username \*" and "Password \*", both with placeholder text "Enter your username" and "Enter your password" respectively. Below these fields is a link "Forgot your password? [Reset password](#)". At the bottom left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

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

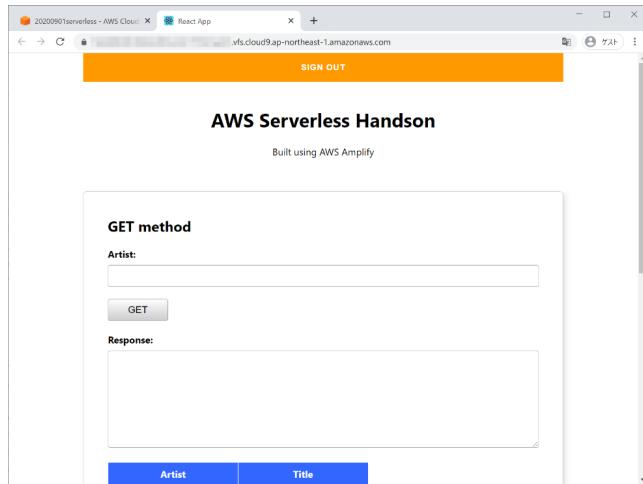
7. メールで確認コードが送られてきますので、入力して **[CONFIRM]** をクリックします。



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 left is a 'Back to Sign In' link, and at the bottom right is 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。



The screenshot shows a web browser window titled '20200901serverless - AWS CloudWatch Logs' with the URL 'vfls.cloud9.ap-northeast-1.amazonaws.com'. The page is titled 'AWS Serverless Handson' and states 'Built using AWS Amplify'. It features a 'SIGN OUT' button at the top. Below it is a section titled 'GET method' with a form field 'Artist:' and a 'GET' button. A scrollable 'Response:' area is shown below. At the bottom are tabs for 'Artist' and 'Title'.

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すが、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	20200901amplifyXXXXXXX	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。

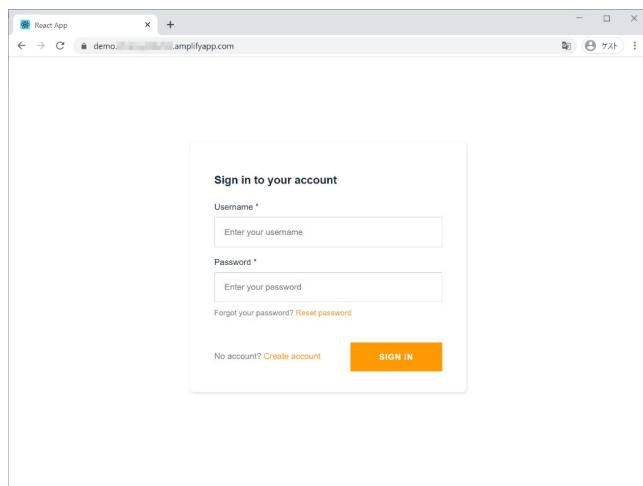
```
✓ Deployment complete!
```

```
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。

- 3. 準備

- Cloud9 環境の削除
  - ✧ [YYYYMMDDserverless]

- 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]

コメントの追加 [青柳2]：他にも削除すべきリソースがあるかもしれませんので、精査します

- **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
```

後片付けは以上です。

## 7. 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。

- 3. 準備
  - Cloud9 環境の削除
    - ✧ [YYYYMMDDserverless]
- 4. 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. Amplify を使ってサーバーレスアプリケーションを構築する

後片付けは以上です。

コメントの追加 [青柳3]: 他にも削除すべきリソースがあるかもしれませんので、精査します