

[日付]

はじめてのサーバーレス

[文書のサブタイトル]

KAMEDA, HARUNOBU

AMAZON CORPORATE

目次

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

1. はじめに

1.1. 本ハンズオンのゴール

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

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

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

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

1.2. 準備事項

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

2. ハンズオンの概要

2.1. ハンズオン全体を通しての注意事項

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

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

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

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

2.2. ハンズオンの構成

本ハンズオンは 2 つのラボで構成されています。

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

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

3. 準備

本日のハンズオンで使用する **Cloud9** の環境を構築します。

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

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

3.1. Cloud9 環境の作成

3.1.1. AWS マネジメントコンソールへのログイン

1. AWS マネジメントコンソールにログインします。
2. ログイン後、画面右上のヘッダー部のリージョン選択にて、**利用を指示されたリージョン** となっていることを確認します。

注：[東京]、[バージニア]、[オレゴン]、[シンガポール] のどれかになります。

3.1.2. Cloud9 環境の作成

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

The screenshot shows the AWS Cloud9 service page. At the top right, there is a user profile with the name "awsuser" and a location dropdown set to "東京". Below the header, the main title is "AWS Cloud9" with the subtitle "A cloud IDE for writing, running, and debugging code". A call-to-action button "Create environment" is highlighted with a red box. To the left, a section titled "How it works" contains text about creating a development environment on an Amazon EC2 instance or connecting to an existing Linux server via SSH. It also mentions the availability of a rich code editor, integrated debugger, and built-in terminal. A "Learn more" link is present. On the right, there is a sidebar titled "Getting started" with several links and their estimated reading times: "Before you start" (2 min read), "Create a environment" (3 min read), "Working with environments" (15 min read), "Working with the IDE" (10 min read), and "Working with AWS Lambda" (5 min read). A "More resources" link is at the bottom of the sidebar.

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 box and contains the value '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)

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 GiB RAM + 1 vCPU)
Free-tier eligible. Ideal for educational users and exploration.

t3.small (2 GiB RAM + 2 vCPU)
Recommended for small-sized web projects.

m5.large (8 GiB 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 hibernation setting of half an hour of no activity 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.

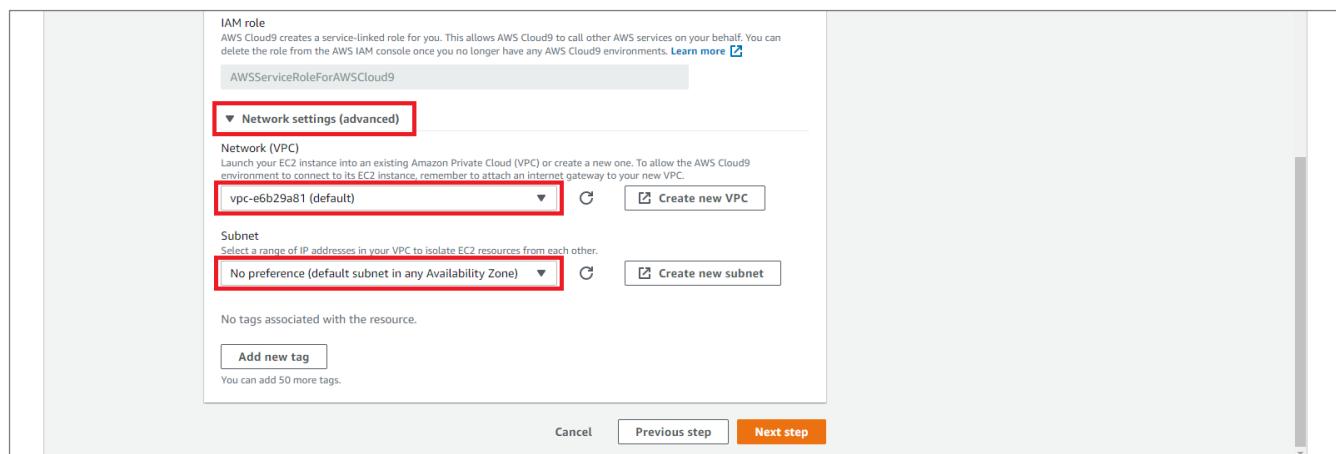
Add new tag

You can add 50 more tags.

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

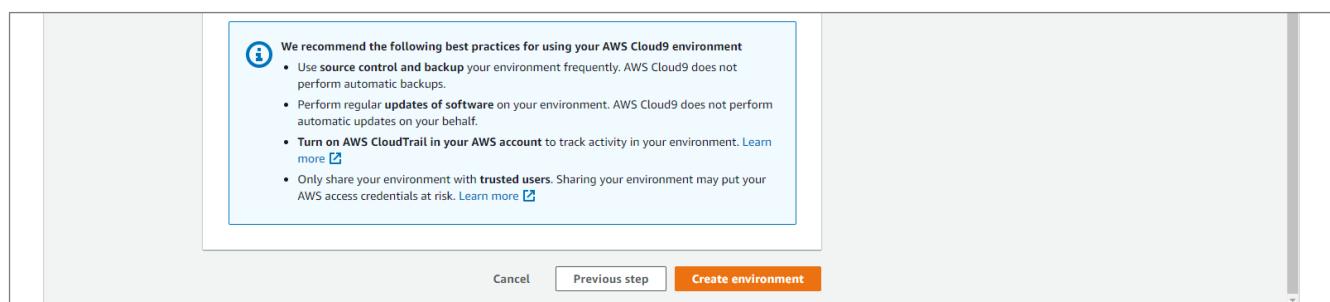
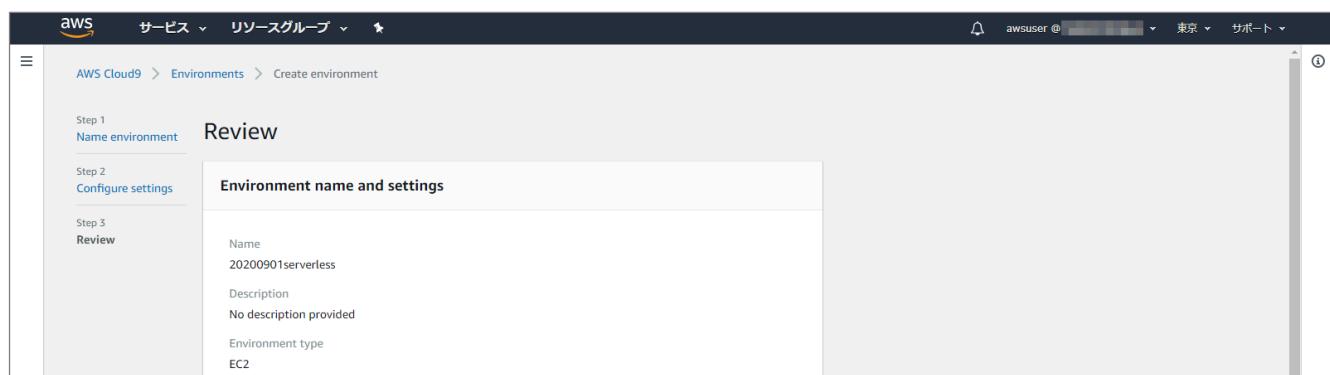
Cloud9 環境を構築する VPC および サブネット を指定できますので、任意の VPC を指定してください。サブネットは「パブリックサブネット」であるものを指定してください。

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。

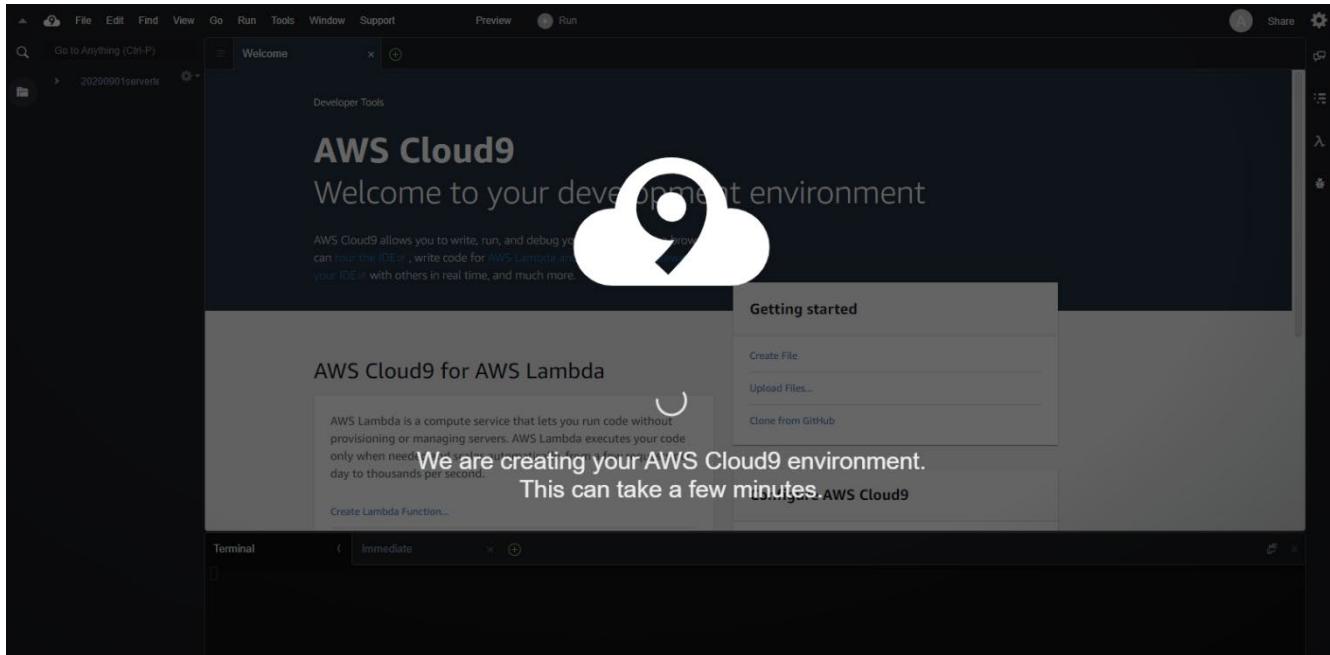


7. [Next Step] をクリックします。

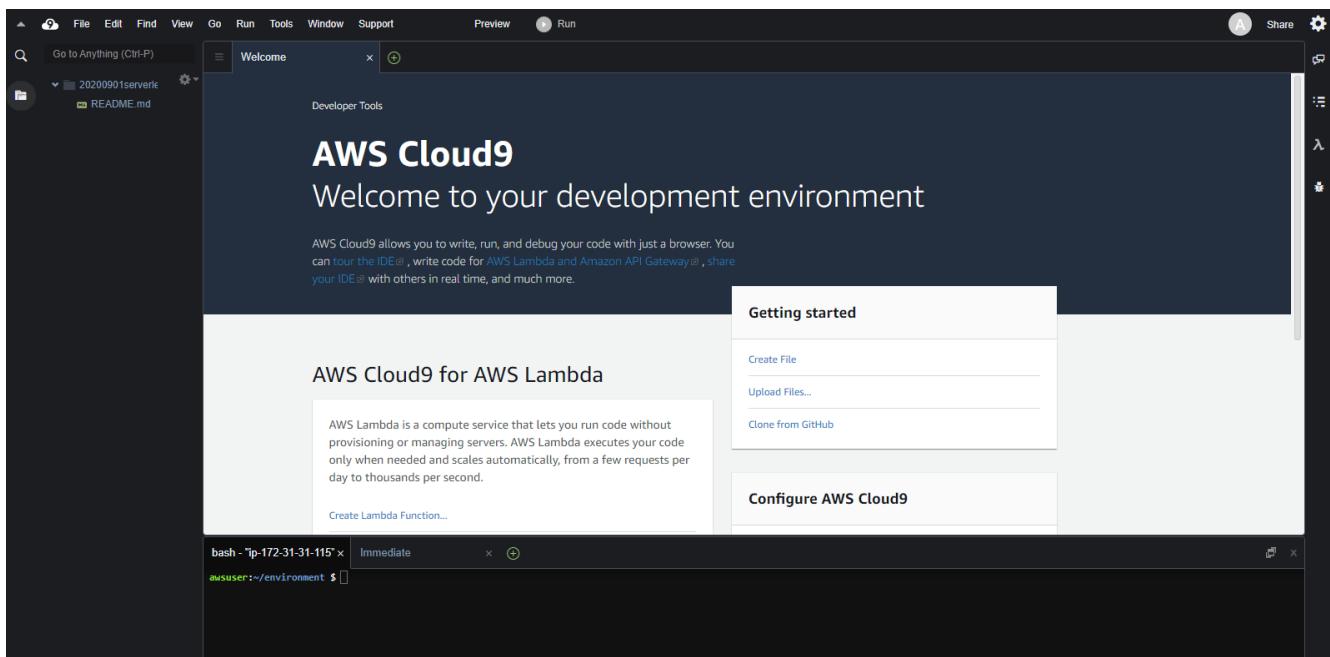
8. 確認画面になりますので、[Create environment] をクリックします。



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。



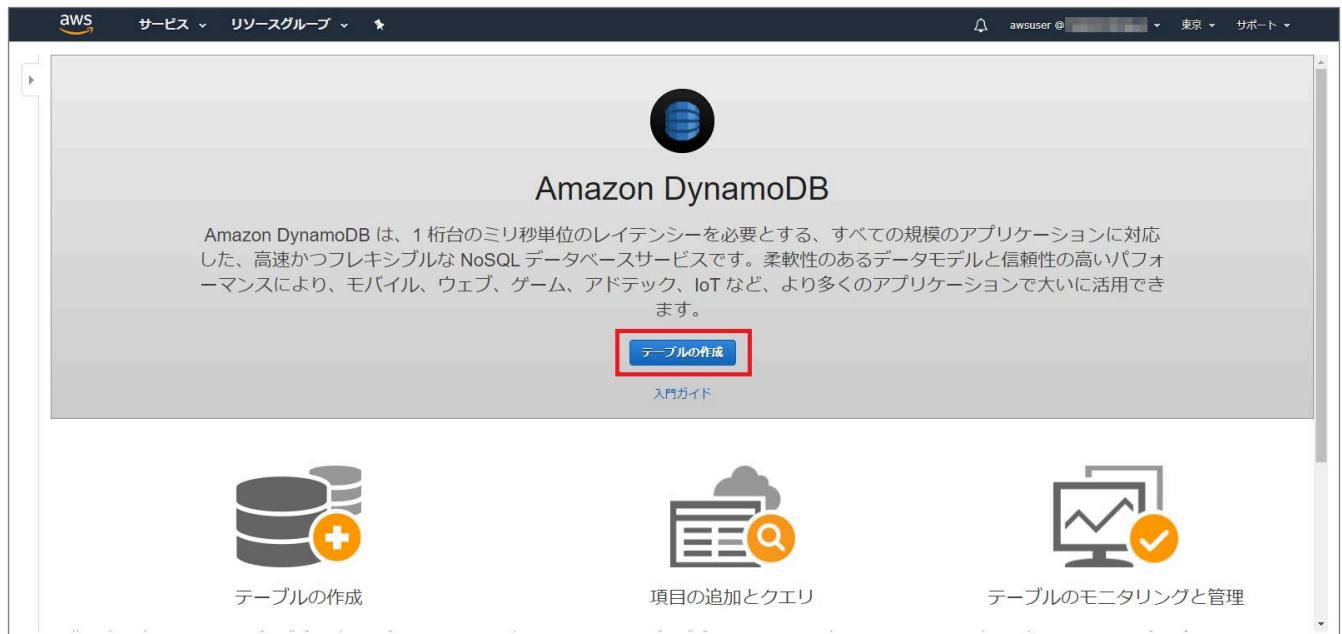
4. サーバーレスアプリケーションを step-by-step で構築する

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

The screenshot shows the 'DynamoDB テーブルの作成' (Table Creation) wizard. In the 'テーブル名' (Table Name) field, '20200901serverless' is entered. Under 'プライマリキー' (Primary Key), 'Artist' is selected as the key name and '文字列' (String) as the type. The 'ソートキーの追加' (Add Sort Key) checkbox is checked, and 'Title' is selected as the sort key name with '文字列' (String) as the type. At the bottom, the 'デフォルト設定' (Default Settings) section is expanded, showing options like 'セカンダリインデックスなし' (No secondary indexes). The '作成' (Create) button is visible at the bottom right.

3. 【作成】をクリックします。

4. テーブルが作成されるまで 1~2 分程度かかります。

5. テーブルが作成されたら、【項目】タブをクリックします。

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

The screenshot shows the '20200901serverless' table overview. The '項目' (Attributes) tab is selected. Under 'スキヤン: [テーブル] 20200901serverless: Artist, Title へ', the 'スキャン' (Scan) dropdown is set to 'テーブル'. The 'Artist' and 'Title' attributes are listed with their respective data types: 'Artist' is of type 'String' and 'Title' is also of type 'String'. A note at the bottom states: '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

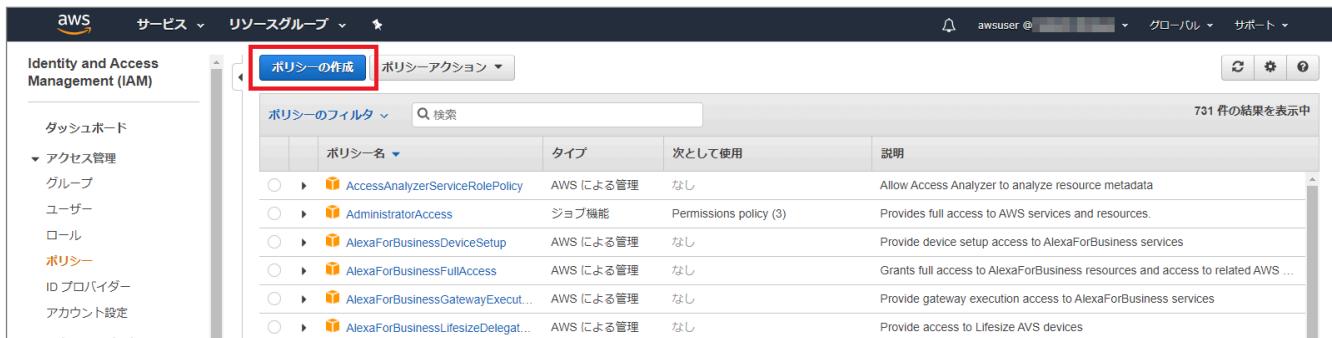
1. AWS マネジメントコンソールのサービス一覧から [IAM] を選択します。

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. On the left sidebar, under the 'Identity and Access Management (IAM)' section, the 'Policies' option is highlighted with a red box. The main content area displays the 'Identity and Access Management へようこそ' (Welcome to Identity and Access Management) page. It includes sections for IAM ユーザーのサインインリンク (Sign-in links for IAM users), IAM リソース (IAM resources), and セキュリティステータス (Security status). A progress bar at the bottom indicates '5 項目中 3 項目が完了しています' (3 items completed out of 5).

2. [ポリシーの作成] をクリックします。



The screenshot shows the 'Create Policy' screen within the AWS IAM service. The 'Policies' menu item from the previous screenshot is also highlighted with a red box. The main content area displays a table of existing policies, with the first row selected. The table columns include 'Policy Name', 'Type', 'Next to use', and 'Description'. A search bar labeled '検索' (Search) is visible above the table, and a note at the top right says '781 件の結果を表示中' (781 results displayed).

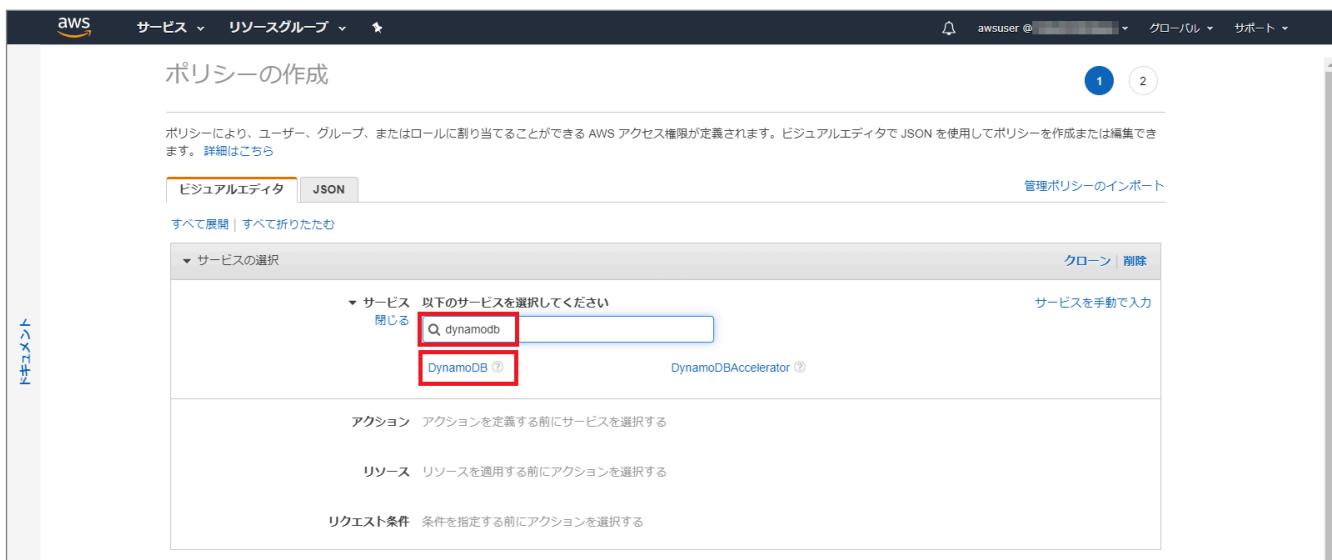
3. [サービス] をクリックして展開します。



The screenshot shows the AWS IAM Policy Editor interface. At the top, there are tabs for 'ビジュアルエディタ' (Visual Editor) and 'JSON'. Below the tabs, there are two numbered circles: '1' and '2'. A sidebar on the left has a blue 'ヘルプ' (Help) button. The main area is titled 'ポリシーの作成' (Create Policy). It contains sections for 'アクション', 'リソース', and 'リクエスト条件'. A large 'サービス' (Service) section is expanded, showing a search bar with 'dynamodb' typed into it. Other service options like 'DynamoDB' and 'DynamoDB Accelerator' are listed below. A red box highlights the 'サービス' section title and the search bar.

4. 検索欄に [dynamodb] と入力します。

表示された [DynamoDB] をクリックします。



This screenshot continues from the previous one, showing the search results for 'dynamodb'. The 'サービス' dropdown is still expanded, and the search bar now displays 'dynamodb'. The result 'DynamoDB' is highlighted with a red box. Other services like 'DynamoDB Accelerator' are also visible. The rest of the policy editor interface remains the same, with sections for 'アクション', 'リソース', and 'リクエスト条件'.

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。

DynamoDB (1つのアクション) ▲ 1つの警告 クローン | 削除

サービス: DynamoDB

アクション: 許可されるアクションを DynamoDB で指定 ①

手動のアクション (アクションの追加)

すべての DynamoDB アクション (dynamodb:*)

アクセスレベル

リスト 読み込み (21 が選択されました)

タグ付け 書き込み (21 が選択されました)

すべて展開 | すべて折りたたむ

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 ②	backup リソース ARN を指定します。 DescribeBackup および、さらに 1 つのアクション。 ARN の追加 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 ②	global-table リソース ARN を指定します。 UpdateGlobalTable および、さらに 1 つのアクション。 ARN の追加 アクセスを制限	<input type="checkbox"/> このアカウント内のいずれか
index ②	タイプが index であるリソースを指定していません ARN の追加 アクセスを制限	<input type="checkbox"/> このアカウント内のいずれか
stream ②	stream リソース ARN を指定します。 GetRecords および、さらに 1 つのアクション。 ARN の追加 アクセスを制限	<input type="checkbox"/> このアカウント内のいずれか
table ②	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)

The screenshot shows the 'Policy Name' field populated with '20200901serverlessLambdaPolicy'. The 'Description' field is empty. The 'Summary' section indicates the policy grants full access to DynamoDB. The 'Permissions Summary' table shows 'DynamoDB' with actions '完全: 書き込み 制限: 読み込み' and conditions '複数'.

サービス	アクセスレベル	リソース	リクエスト条件
DynamoDB	完全: 書き込み 制限: 読み込み	複数	なし

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。

The screenshot shows the '20200901serverlessLambdaPolicy' policy listed in the IAM Policies table. The policy is highlighted with a red box. The table includes columns for Policy Name, Type, and Description.

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	
AccessAnalyzerServiceRolePolicy	AWS による管理	なし	Allow Access Analyzer to analyze resource metadata
AdministratorAccess	ジョブ機能	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
AlexaForBusinessLifesizeDelega...	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...

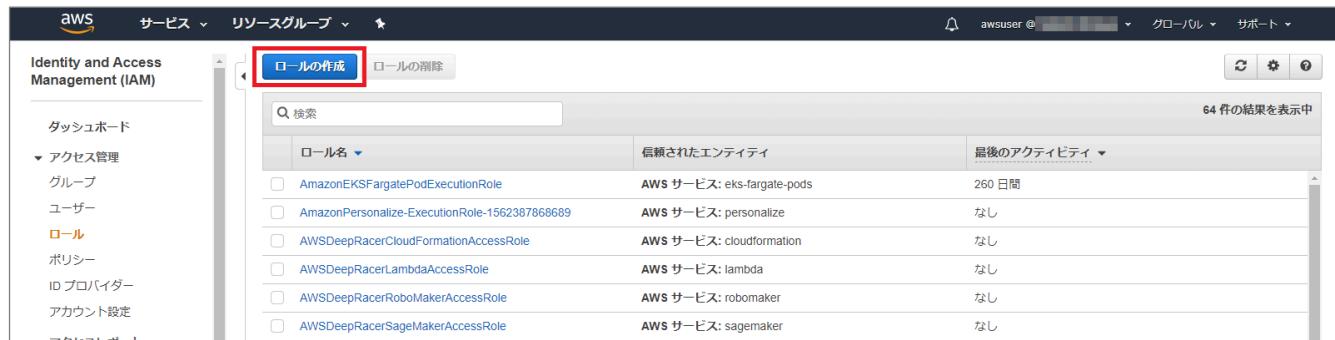
4.1.3. Lambda 用 IAM ロールの作成

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 interface. The left sidebar has 'Identity and Access Management (IAM)' selected. Under 'Roles', the 'Roles' option is highlighted with a red box. The main content area displays 'Identity and Access Management へようこそ' and 'IAM ユーザーのサインインリンク'. It shows 'IAM リソース' with 'ユーザー: 2', 'グループ: 0', and 'ID プロバイダ: 0'. Below this is a section titled 'セキュリティステータス' with five items: 'ルートアカウントの MFA を有効化' (checked), '個々の IAM ユーザーの作成' (checked), 'グループを使用したアクセス許可の割り当て' (warning), 'IAM パスワードポリシーの適用' (warning), and 'アクセスキーのローデーション' (checked). A progress bar indicates '5 項目中 3 項目が完了しています'.

2. [ロールの作成] をクリックします。



The screenshot shows the 'Create New Role' page in the AWS IAM console. The left sidebar shows 'Identity and Access Management (IAM)' with 'Roles' selected. The main area has a search bar and a table with one row. The table columns are 'Role Name', 'Assumed by Entity', and 'Last Activity'. The single row shows 'AmazonEKSFargatePodExecutionRole' under 'AWS Service: eks-fargate-pods', 'AWSサービス: personalize' under 'AWS Service', and '260 日間' under 'Last Activity'. A blue box highlights the 'Create New Role' button at the top of the page.

Role Name	Assumed by Entity	Last Activity
AmazonEKSFargatePodExecutionRole	AWSサービス: eks-fargate-pods AWS Service: personalize	260 日間 なし

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。

□ ロールの作成

信頼されたエンティティの種類を選択

AWS サービス EC2、Lambda、およびその他

別の AWS アカウント お客様またはゲートウェイパートナーに接続しています

ウェブ ID Cognito または任意の OpenID プロバイダ

SAML 2.0 フェデレーション ヨン 企業ディレクトリ

AWS のサービスによるアクションの代行を許可します。詳細は[こちら](#)

ユースケースの選択

一般的なユースケース

EC2
Allows EC2 instances to call AWS services on your behalf.

Lambda
Allows Lambda functions to call AWS services on your behalf.

または、サービスを選択してユースケースを表示します

API Gateway	CodeDeploy	EMR	KMS	Rekognition
AWS Backup	CodeGuru	ElastiCache	Kinesis	RoboMaker
AWS Chatbot	CodeStar Notifications	Elastic Beanstalk	Lake Formation	S3
AWS Marketplace	Comprehend	Elastic Container Service	Lambda	SMS
AWS Support	Config	Elastic Transcoder	Lex	SNS
Amplify	Connect	Elastic Load Balancing	License Manager	SWF

* 必須 キャンセル 次のステップ: アクセス権限

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。

□ ロールの作成

▼ Attach アクセス権限ポリシー

新しいロールにアタッチするポリシーを 1 つ以上選択します。

ポリシーの作成

ポリシー名 フィルタ Q 検索 732 件の結果を表示中

<input checked="" type="checkbox"/> 20200901serverlessLambdaPolicy	なし
<input type="checkbox"/> AccessAnalyzerServiceRolePolicy	なし
<input type="checkbox"/> AdministratorAccess	Permissions policy (3)
<input type="checkbox"/> AlexaForBusinessDeviceSetup	なし
<input type="checkbox"/> AlexaForBusinessFullAccess	なし
<input type="checkbox"/> AlexaForBusinessGatewayExecution	なし
<input type="checkbox"/> AlexaForBusinessLifesizeDelegatedAccessPolicy	なし
<input type="checkbox"/> AlexaForBusinessNetworkProfileServicePolicy	なし

▼ アクセス権限の境界の設定

* 必須 キャンセル 戻る 次のステップ: タグ

6. [次のステップ: タグ] をクリックします。

- [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。
- [ロール名] 欄に [YYYYMMDDserverlessLambdaRole] と入力します。 (YYYYMMDD は本日の日付)

以下に必要な情報を指定してこのロールを見直してから、作成してください。

ロール名* 英数字と「+=_,@_」を使用します。最大 64 文字。

ロールの説明 Allows Lambda functions to call AWS services on your behalf.

最大 1000 文字。英数字と「+=_,@_」を使用します。

信頼されたエンティティ AWS のサービス: lambda.amazonaws.com

ポリシー 20200901serverlessLambdaPolicy

アクセス権限の境界 アクセス権限の境界が設定されていません

追加されたタグはありません。

* 必須 キャンセル 戻る ロールの作成

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

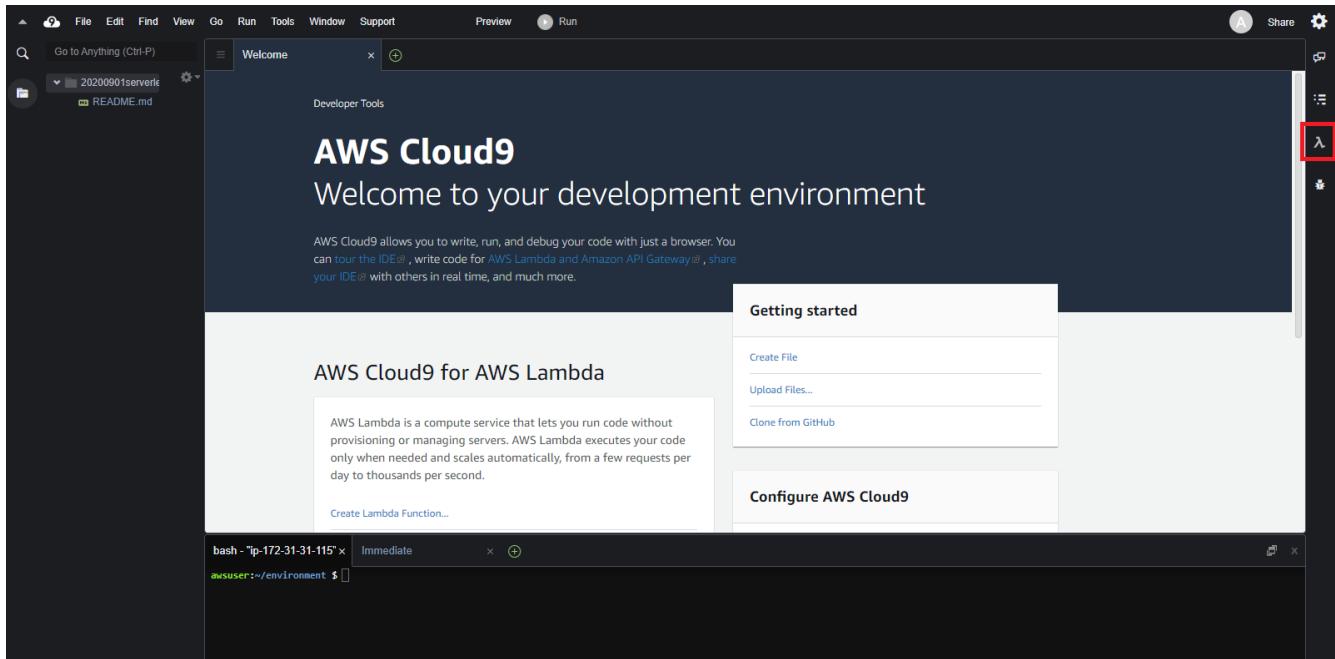
Identity and Access Management (IAM)

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	260 日間
AmazonPersonalizeExecutionRole-156238786869	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

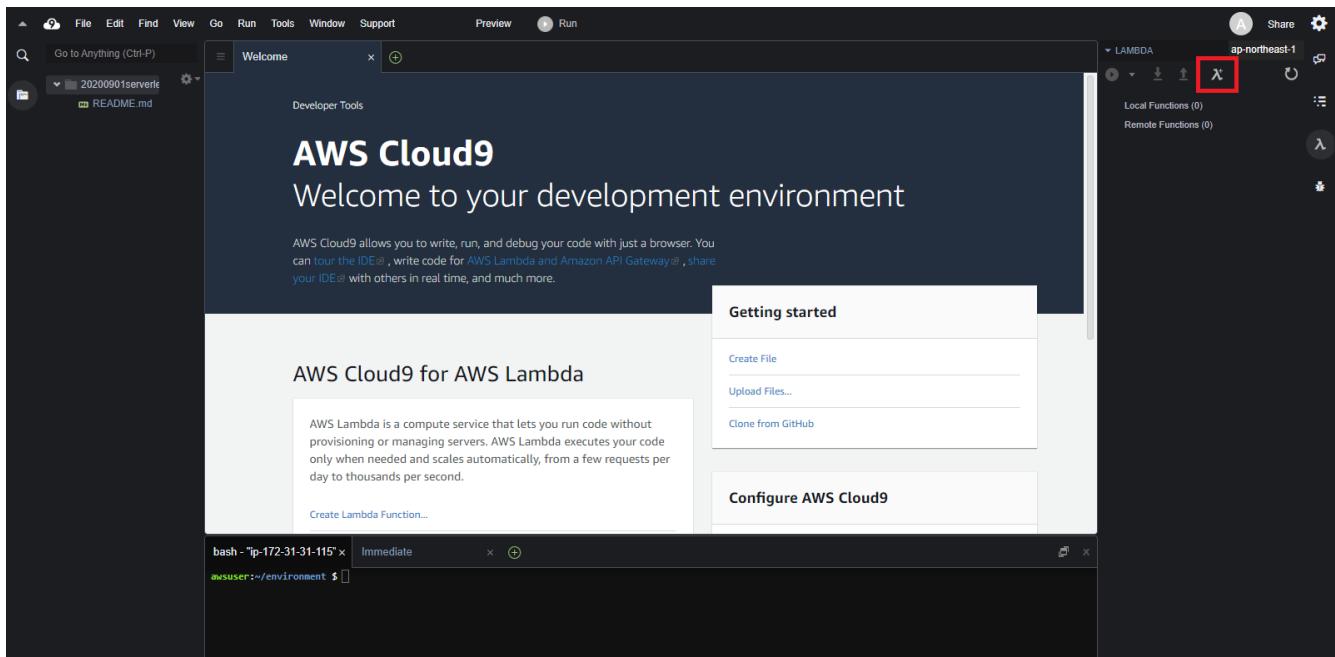
4.1.4. Cloud9 上での Lambda 関数の作成とテスト (Write 関数)

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 Resource]を選んでください。

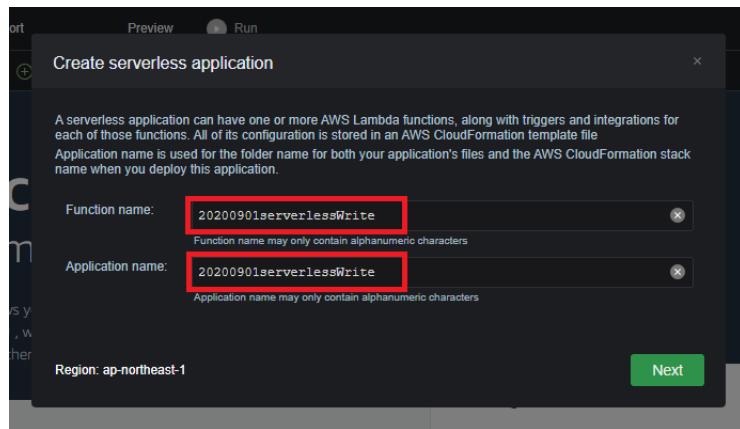


2. [Create] ボタン (Lambda アイコンに[+]印が付いたもの) をクリックします。



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

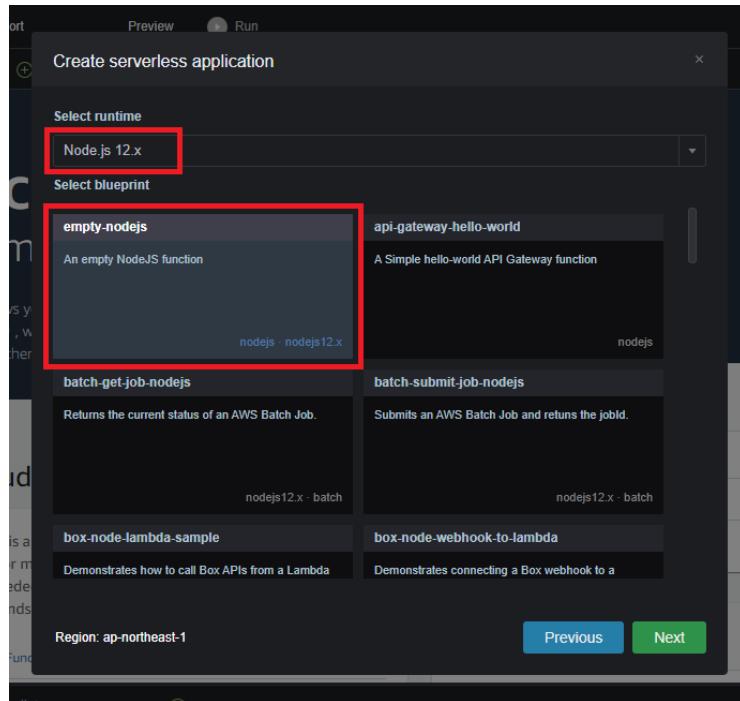
[Application name] 欄は [Function name] 欄と同じ内容が自動的に入力されますので、そのままにします。



4. [Next] をクリックします。

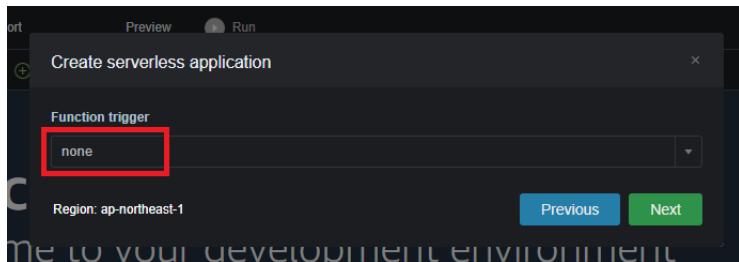
5. [Select runtime] で [Node.js 12.x] を選択します。

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



6. [Next] をクリックします。

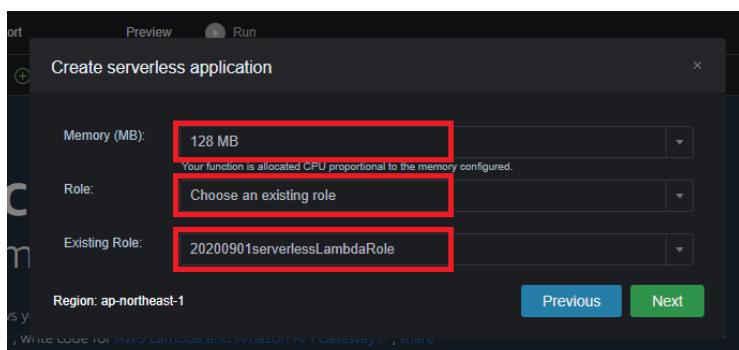
7. [Function trigger] で [none] を選択します。



8. [Next] をクリックします。

9. 以下の通りに選択します。

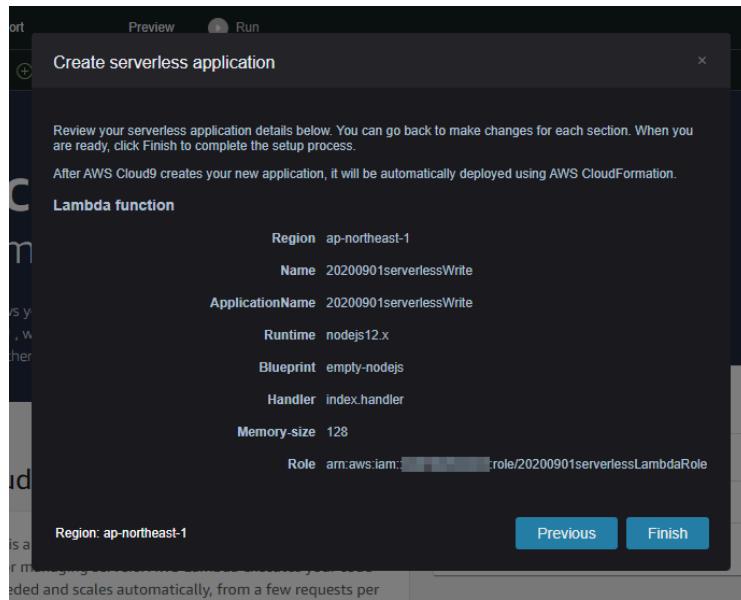
- **Memory:** [128 MB] (デフォルトのまま)
- **Role:** [Choose an existing role]
- **Existing Role:** 前手順で作成した [**YYYYMMDDserverlessLambdaRole**] を選択



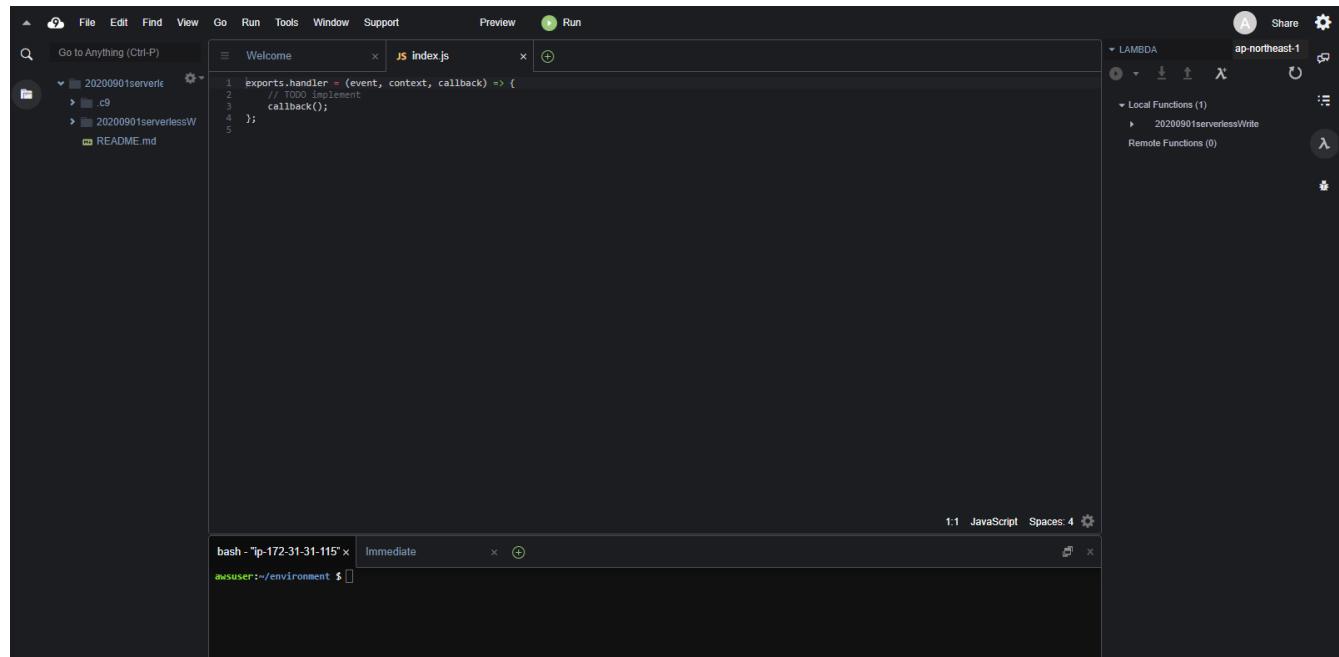
注意：この手順で先に作成した IAM ロールが表示されない場合、IAM ロールに以下のポリシーを追加で作成して再度試してください。[[AmazonDynamoDBFullAccess](#)] [[AWSCloud9Administrator](#)]

10. [Next] をクリックします。

11. 確認画面になりますので、[Finish] をクリックします。

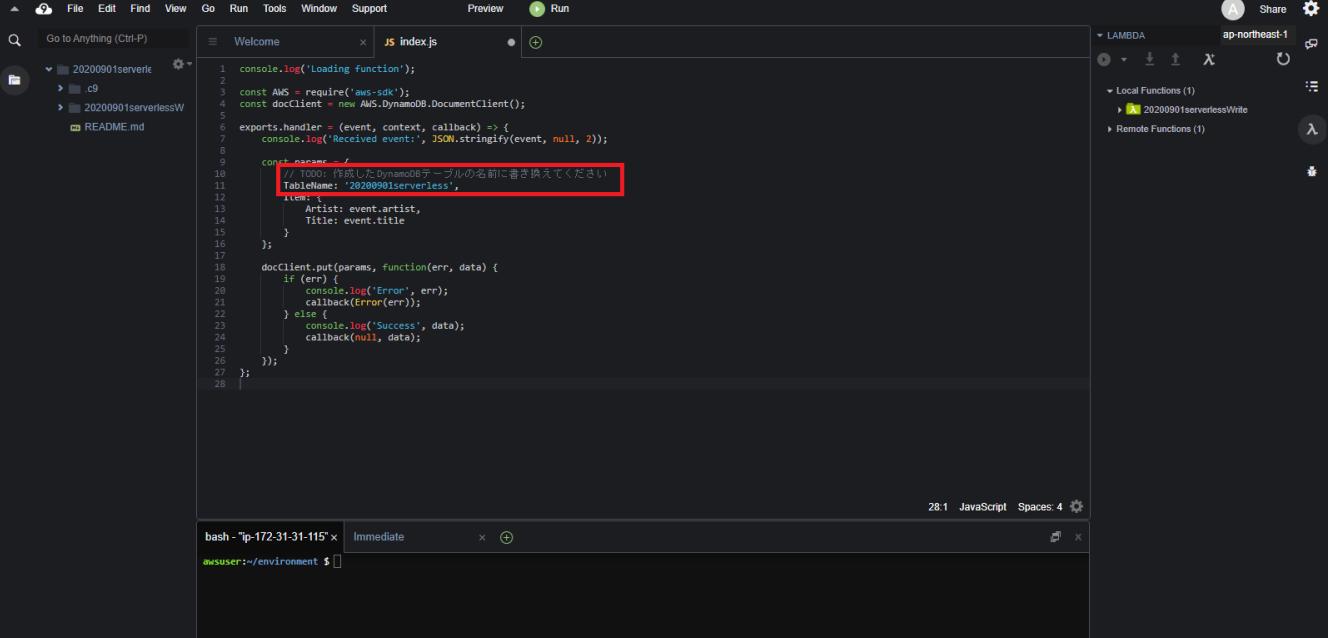


12. Lambda 関数コード (index.js) の編集画面が表示されます。



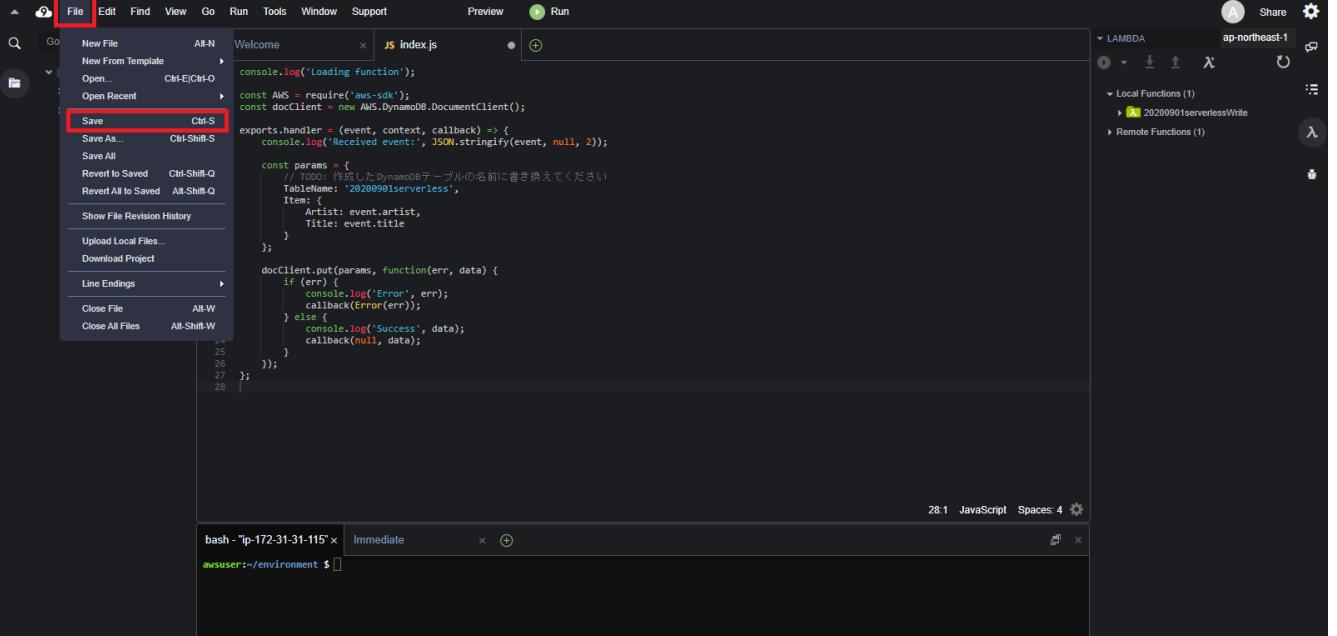
13. 初期入力されているサンプルコードを一旦全て削除します。

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。



```
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         // TODO: 作成したDynamoDBテーブルの名前に書き換えてください
11         TableName: '20200901serverless',
12         Item: {
13             Artist: event.artist,
14             Title: event.title
15         }
16     };
17
18     docClient.put(params, function(err, data) {
19         if (err) {
20             console.log('Error', err);
21             callback(Error(err));
22         } else {
23             console.log('Success', data);
24             callback(null, data);
25         }
26     });
27 }
28
```

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



```
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         // TODO: 作成したDynamoDBテーブルの名前に書き換えてください
11         TableName: '20200901serverless',
12         Item: {
13             Artist: event.artist,
14             Title: event.title
15         }
16     };
17
18     docClient.put(params, function(err, data) {
19         if (err) {
20             console.log('Error', err);
21             callback(Error(err));
22         } else {
23             console.log('Success', data);
24             callback(null, data);
25         }
26     });
27 }
28
```

15. 画面上部の **[Run]** をクリックします。

The screenshot shows the AWS Lambda function editor interface. On the left, there's a sidebar with project files: README.md, c9, 20200901serverless, and 20200901serverlessW. 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 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         // TODO: 作成したDynamoDBテーブルの名前に書き換えてください
11         TableName: '20200901serverless',
12         Item: {
13             Artist: event.artist,
14             Title: event.title
15         }
16     };
17
18     docClient.put(params, function(err, data) {
19         if (err) {
20             console.log('Error', err);
21             callback(Error(err));
22         } else {
23             console.log('Success', data);
24             callback(null, data);
25         }
26     });
27 };
28
```

Below the code editor is a terminal window titled 'bash - "ip-172-31-31-115" x' with the command 'awsuser:~/environment \$'.

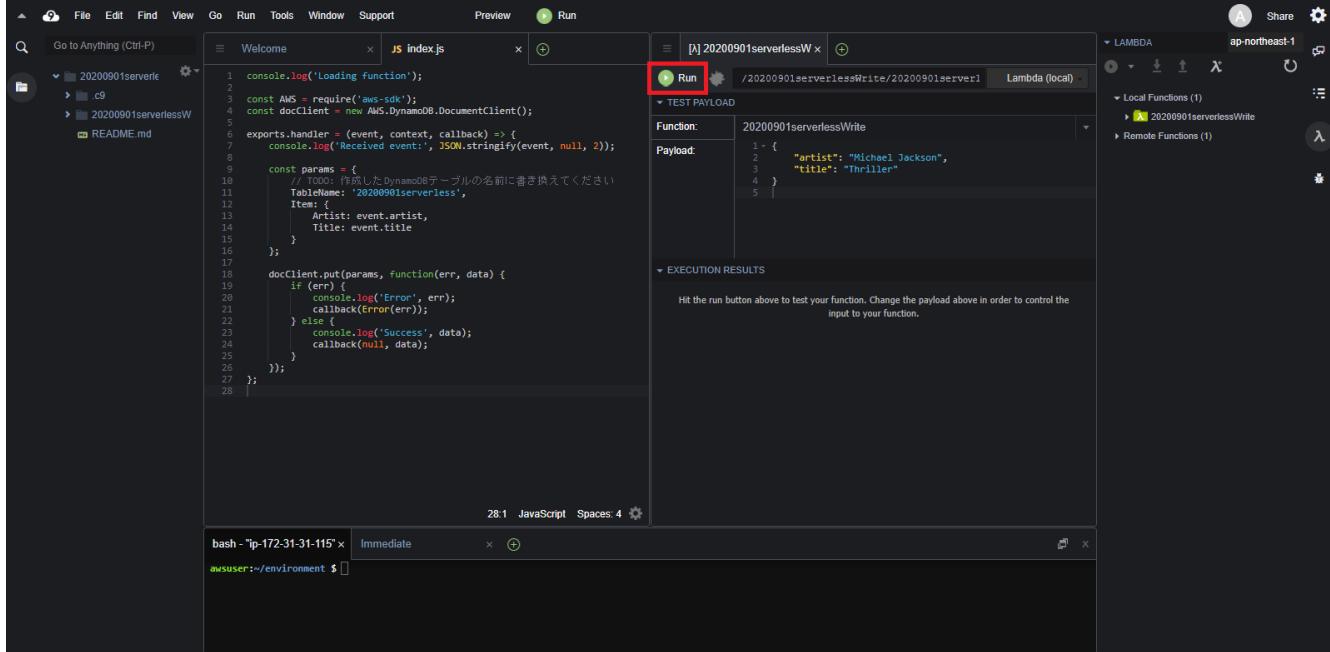
16. [lambda_function_write_test.txt] の内容をコピーして [Payload] 欄にペーストします。

The screenshot shows the AWS Lambda function editor interface. The 'index.js' tab contains the same code as the previous screenshot. On the right, there's a 'TEST PAYLOAD' section with a 'Function:' dropdown set to '20200901serverlessW' and a 'Payload:' input field containing the following JSON:

```
{ "artist": "Michael Jackson", "title": "Thriller"}
```

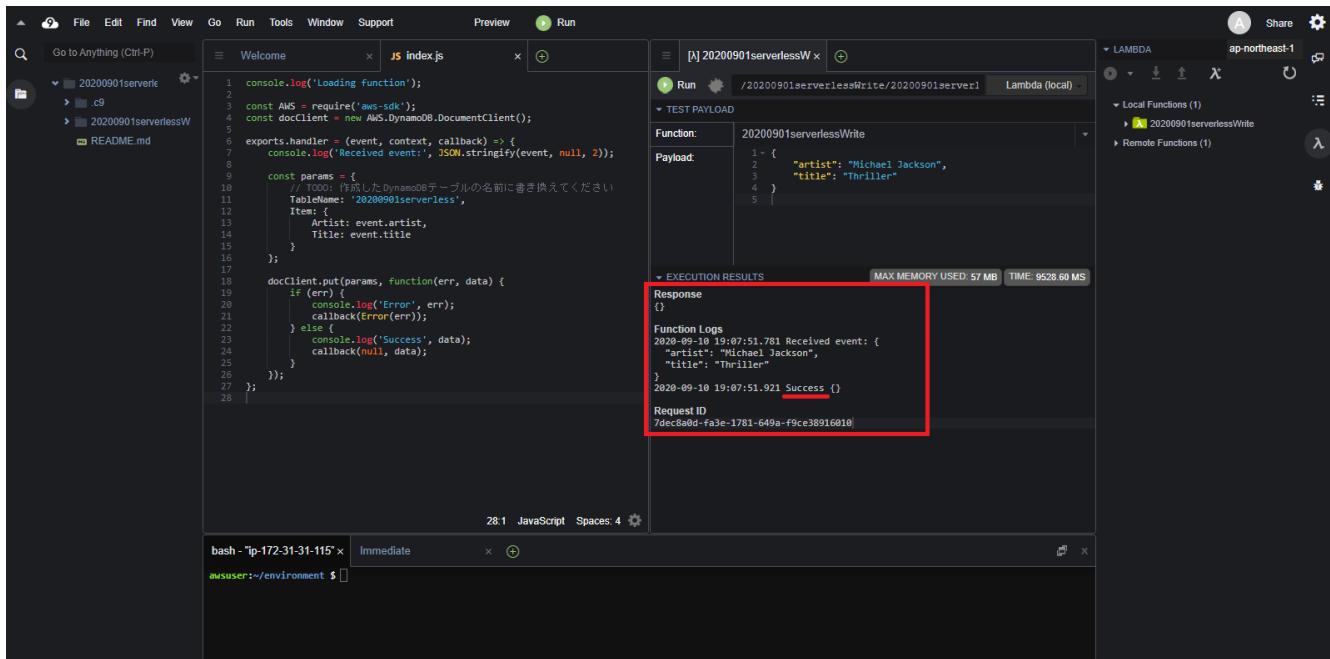
Below the payload editor is an 'EXECUTION RESULTS' section with the message: 'Hit the run button above to test your function. Change the payload above in order to control the input to your function.'

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。



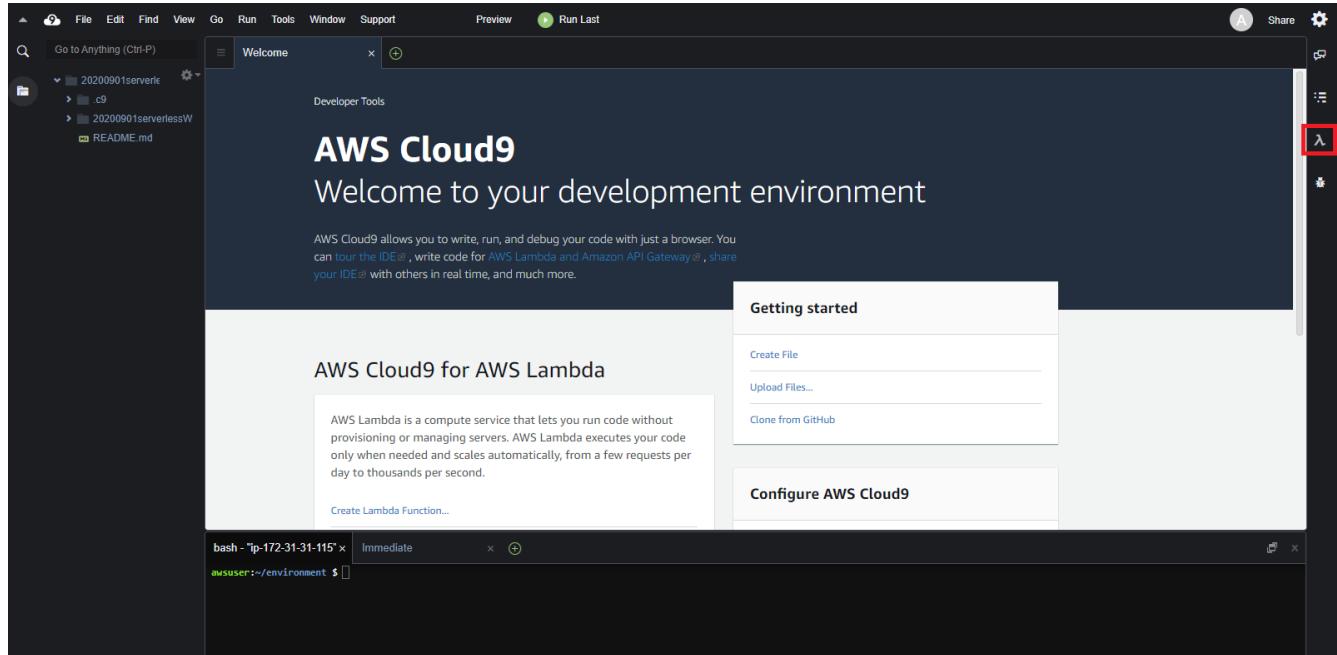
19. DynamoDB の画面に移動して、テーブルの【項目】タブを表示します。

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。

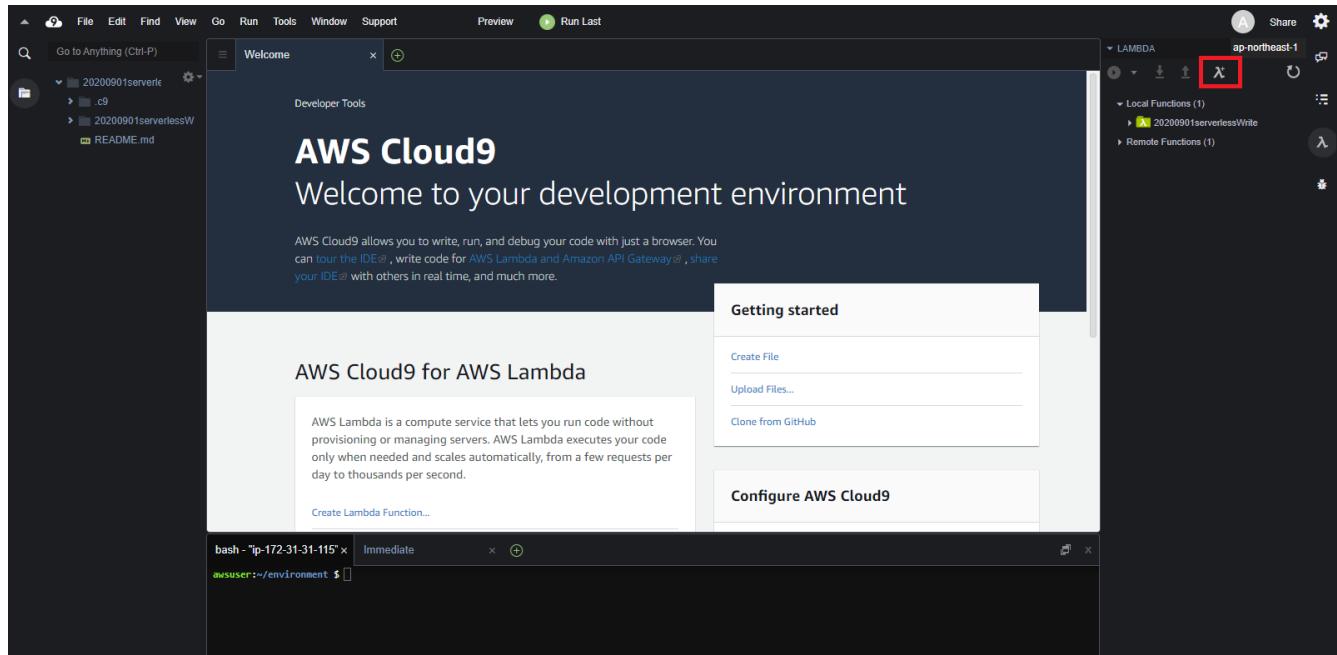
The screenshot shows the AWS DynamoDB console interface. On the left, the navigation menu is visible with options like DynamoDB, ダッシュボード, テーブル, バックアップ, リザーブドキャバディ, 設定, DAX, and イベント. The main area displays a table titled "20200901serverless" with the "項目" tab selected. A search bar at the top of the table header allows filtering by "テーブル名によるフィルター" (Table name filter) and "アクション" (Action). The table has two columns: "Artist" and "Title". A single row is listed: "Artist" is "Michael Jackson" and "Title" is "Thriller". The entire row is highlighted with a red box. In the top right corner of the table area, there is a refresh icon (a circular arrow) which is also highlighted with a red box.

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。

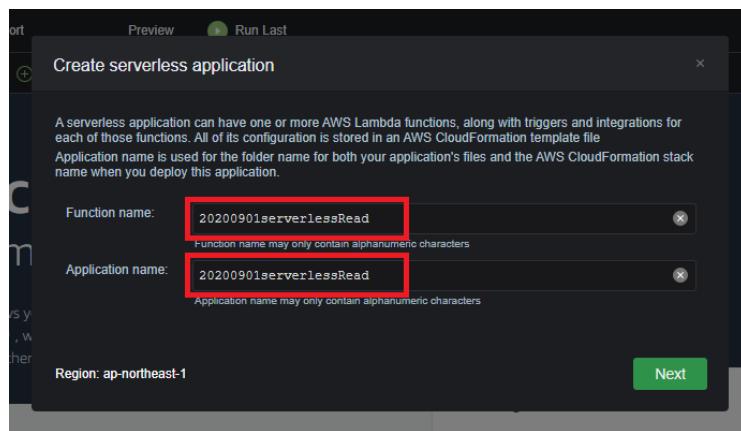


2. [Create] ボタン (Lambda アイコンに[+]印が付いたもの) をクリックします。



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

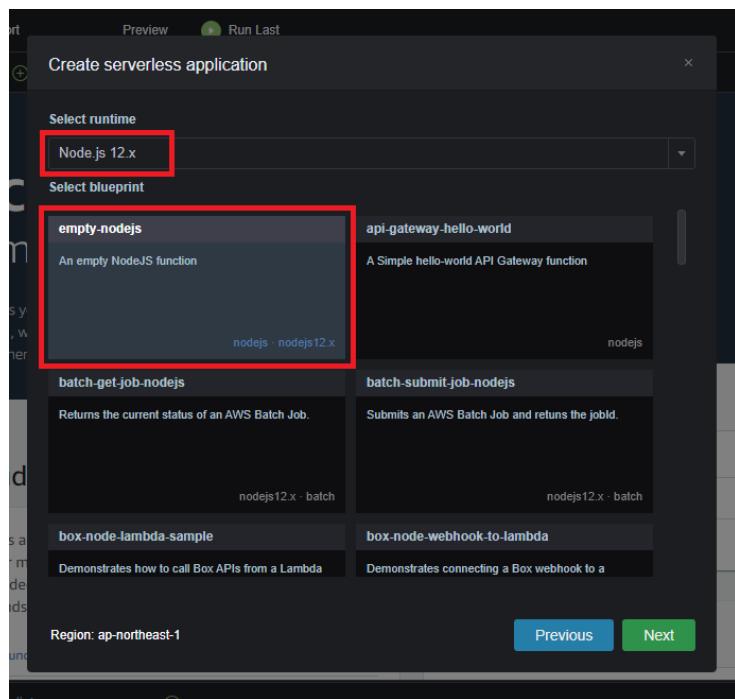
[Application name] 欄は [Function name] 欄と同じ内容が自動的に入力されますので、そのままにします。



4. [Next] をクリックします。

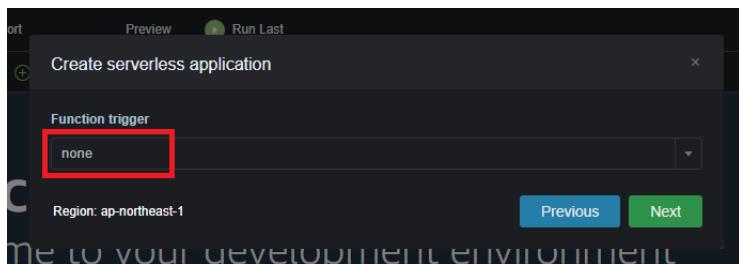
5. [Select runtime] で [Node.js 12.x] を選択します。

[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。

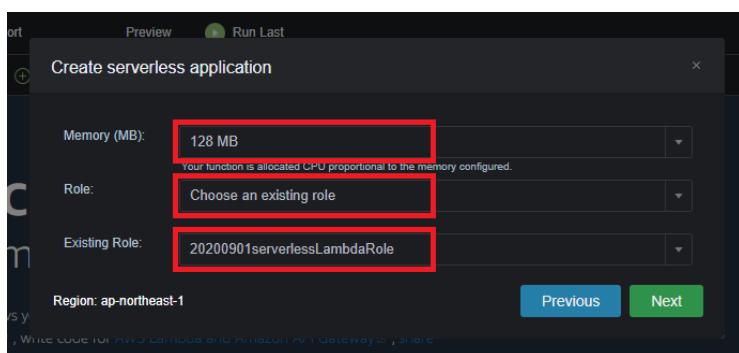
7. [Function trigger] で [none] を選択します。



8. [Next] をクリックします。

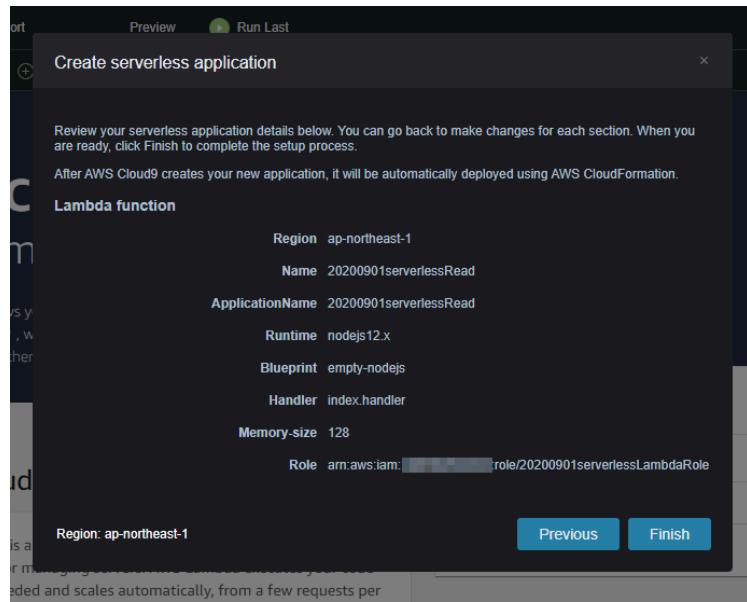
9. 以下の通りに選択します。

- **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

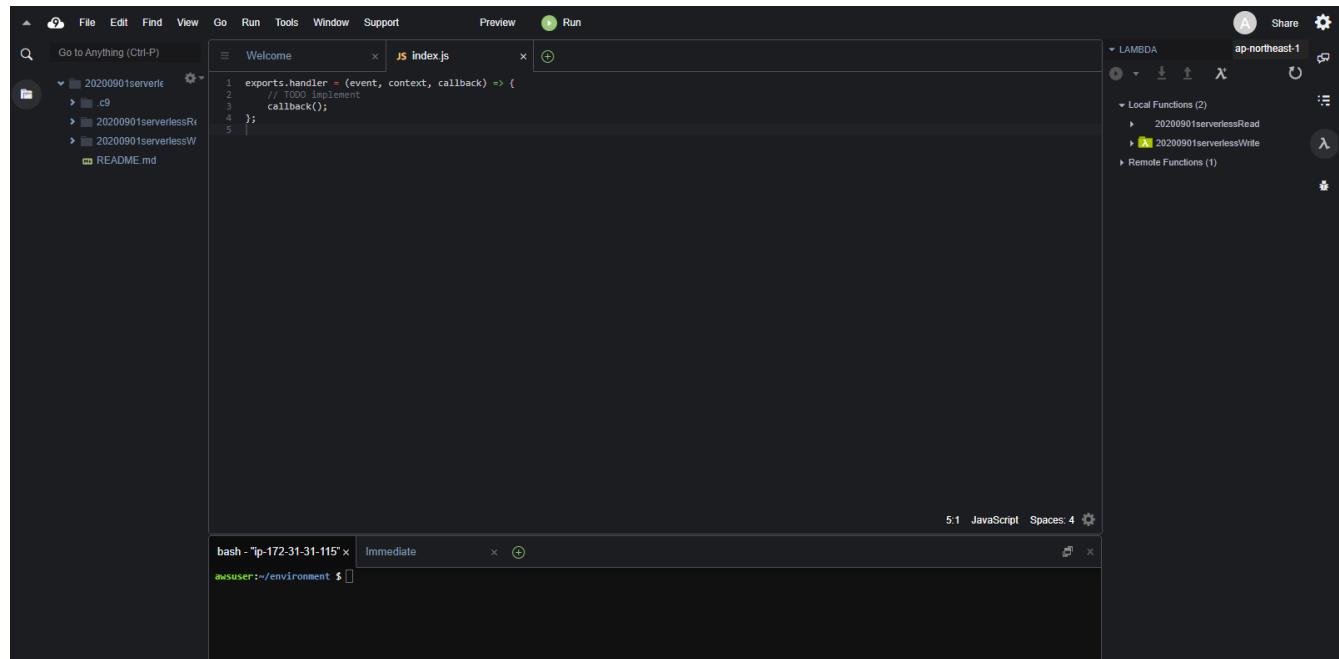


10. [Next] をクリックします。

11. 確認画面になりますので、[Finish] をクリックします。



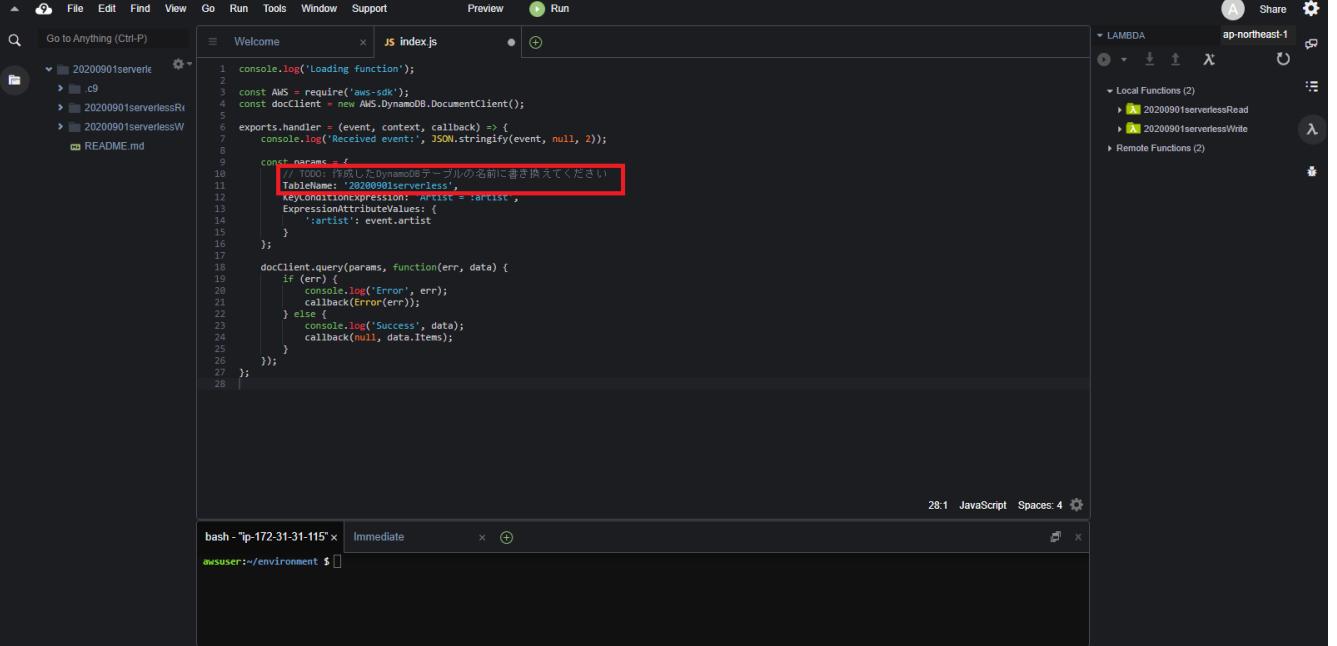
12. Lambda 関数コード (index.js) の編集画面が表示されます。



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');

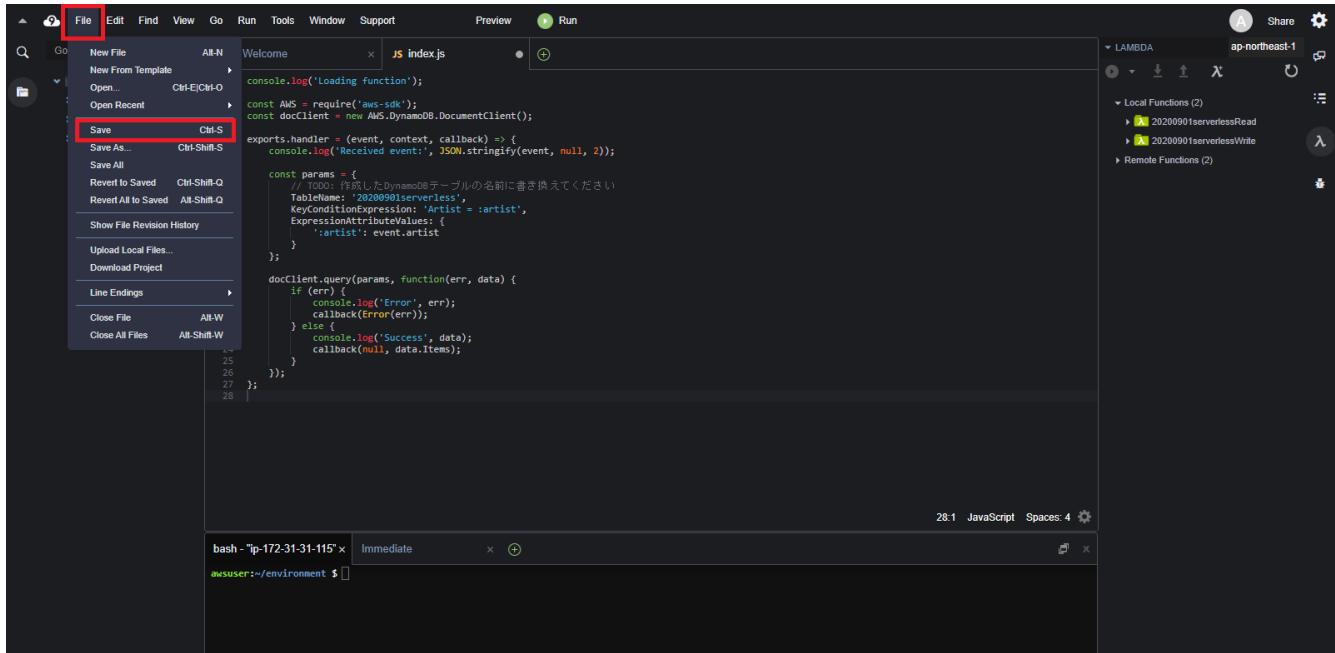
const AWS = require('aws-sdk');
const docClient = new AWS.DynamoDB.DocumentClient();

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

    const params = {
        // 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',
        KeyConditionExpression: 'Artist = :Artist',
        ExpressionAttributeValues: {
            ':Artist': event.artist
        }
    };

    docClient.query(params, function(err, data) {
        if (err) {
            console.log('Error', err);
            callback(Error(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バーから [File]→[Save] の順に選択してファイルを保存します。



15. 画面上部の [Run] をクリックします。

The screenshot shows the AWS Lambda function editor interface. In the center, there is a code editor window titled "index.js" containing the following JavaScript code:

```

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         // TODO: 作成したDynamoDBテーブルの名前に書き換えてください
11         TableName: '20200901serverless',
12         KeyConditionExpression: 'Artist = :artist',
13         ExpressionAttributeValues: {
14             ':artist': event.artist
15         }
16     };
17
18     docClient.query(params, function(err, data) {
19         if (err) {
20             console.log('Error:', err);
21             callback(Error(err));
22         } else {
23             console.log('Success', data);
24             callback(null, data.Items);
25         }
26     });
27 };
28

```

At the top of the editor, there is a toolbar with various icons. The "Run" button, which is green with a play icon, is highlighted with a red box.

On the right side of the interface, there is a sidebar titled "LAMBDA" showing "Local Functions (2)" and "Remote Functions (2)". Below the sidebar, there is a terminal window titled "bash - [ip-172-31-31-115]" 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。

The screenshot shows the AWS Lambda function editor interface, similar to the previous one but with a different payload. In the center, there is a code editor window titled "index.js" and a "TEST PAYLOAD" panel on the right.

The "TEST PAYLOAD" panel has a "Function:" dropdown set to "20200901serverlessRead" and a "Payload:" text area. The payload object is pasted into the "Payload:" field:

```

{
    "artist": "Michael Jackson"
}

```

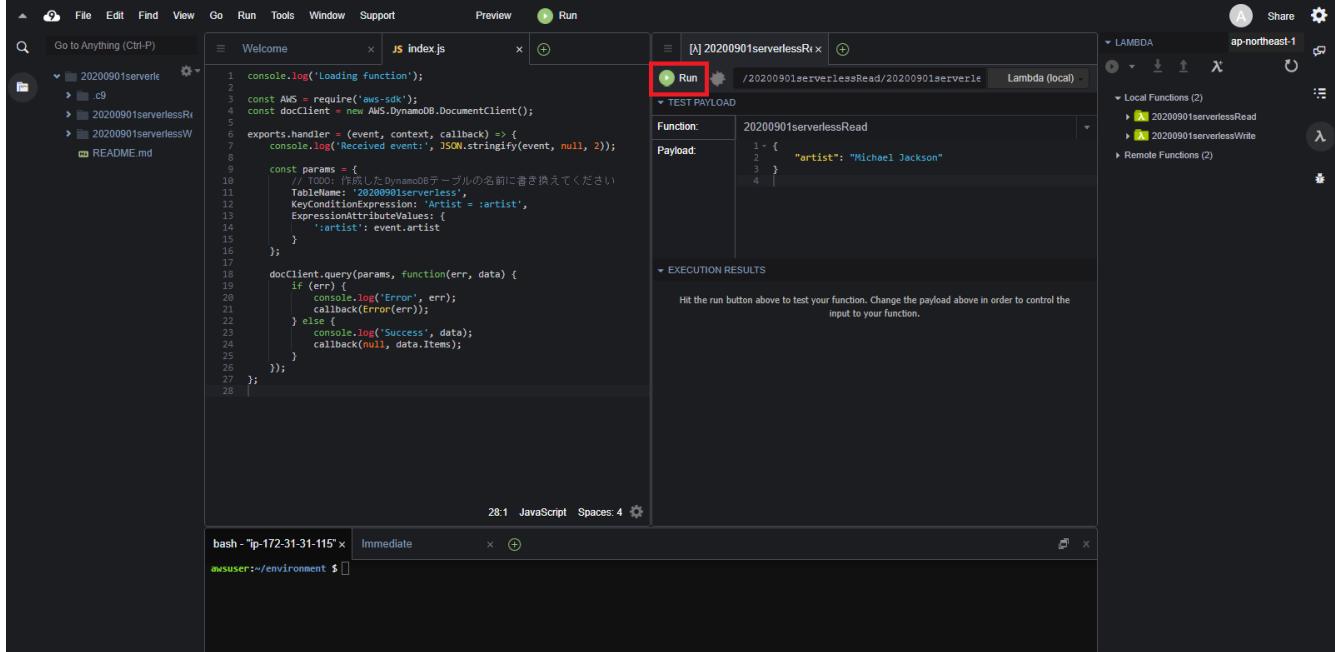
The "Payload:" field is highlighted with a red box.

Below the "TEST PAYLOAD" panel, there is a "EXECUTION RESULTS" section with the following text:

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

At the bottom of the editor, there is a terminal window titled "bash - [ip-172-31-31-115]" with the command "awsuser:~/environment \$".

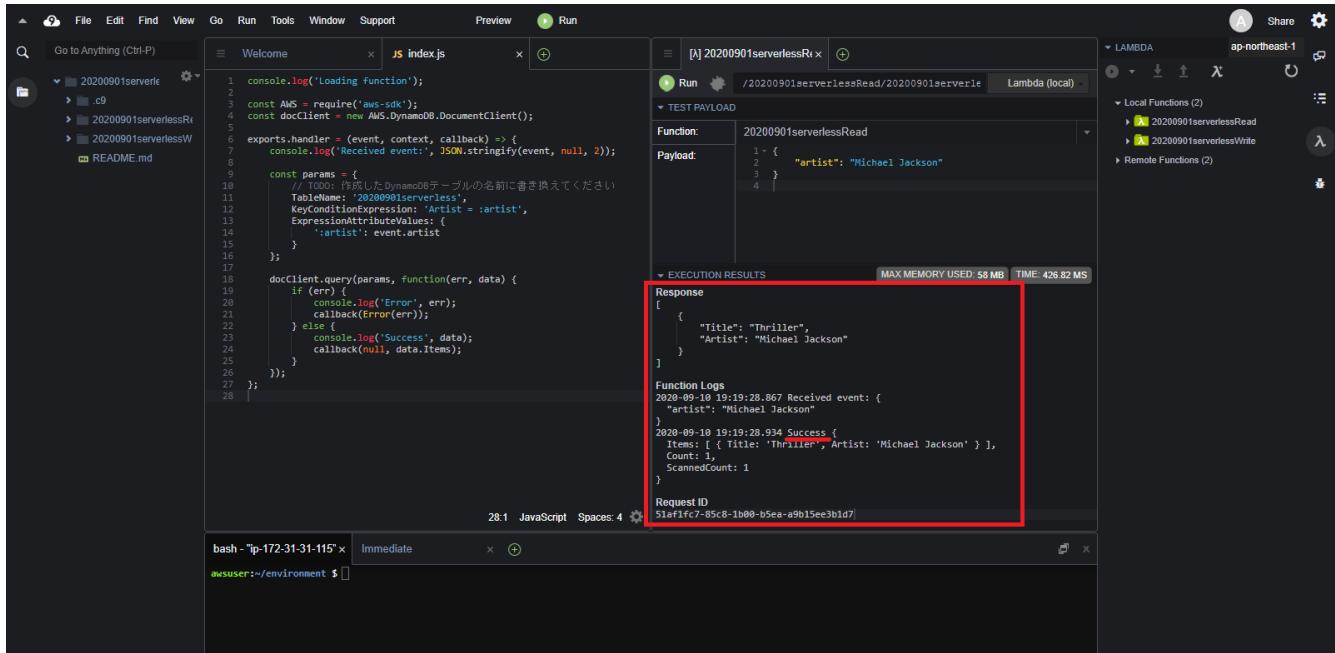
17. Lambda 関数実行タブの [Run] をクリックします。



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

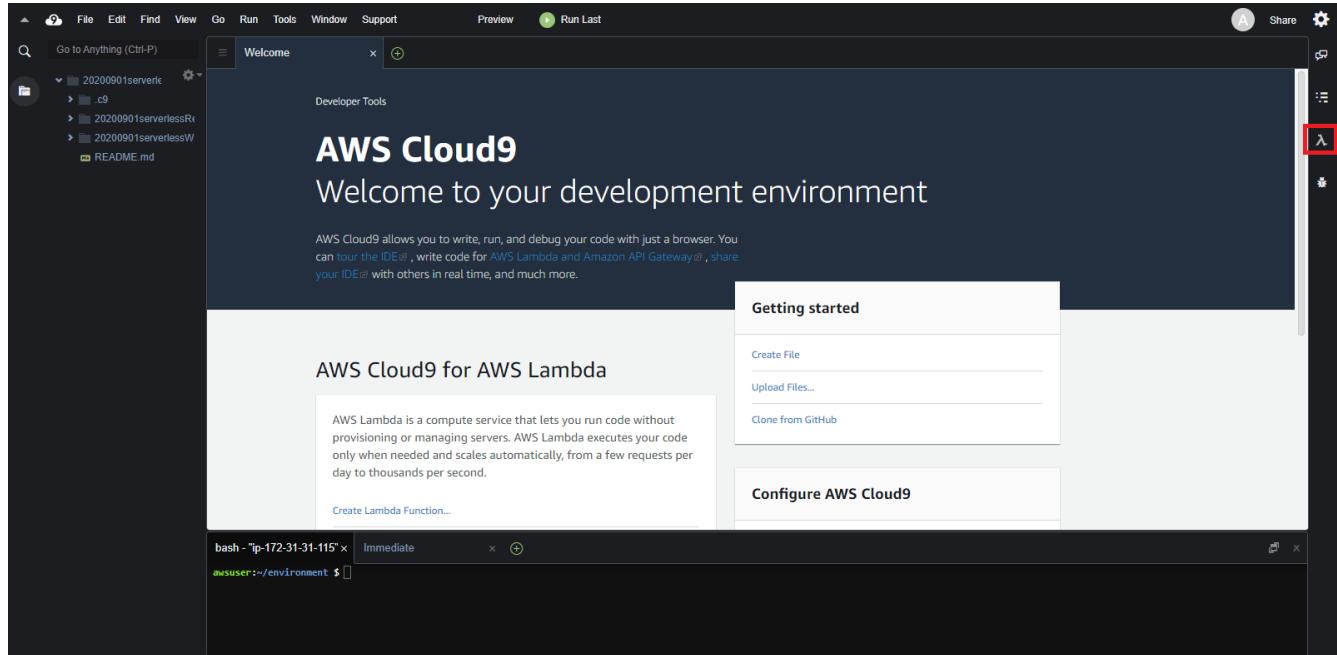
エラー等が発生しておらず、[Success] と表示されていれば成功です。

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



4.1.6. Lambda 関数のデプロイ

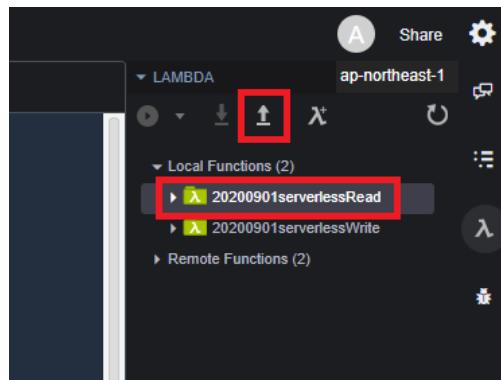
1. 画面右側のツールバーから [Lambda] ボタンをクリックします。



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

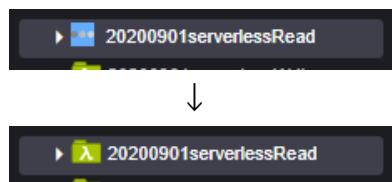
まず、[YYYYMMDDserverlessRead] の方をクリックして選択状態にします。

[Deploy] ボタン ([↑]) をクリックします。

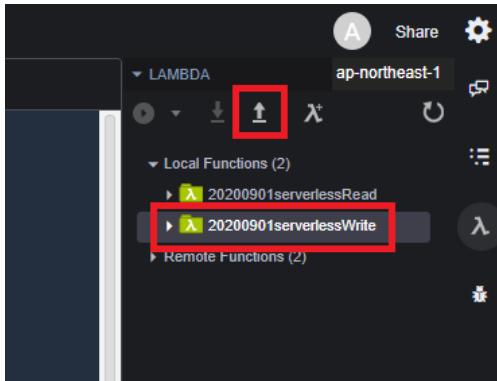


3. Lambda 関数のデプロイが行われます。

フォルダアイコンが「デプロイ中」の動きとなりますので、終わるまで待ちます。



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



5. デプロイが終わるまで待ちます。

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

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

関数名	説明	ランタイム	コードサイズ	最終更新日時
cloud9-20200901serverlessRe-20200901serverlessRead-MC3XGMS0DK10		Node.js 12.x	1.6 kB	1 分前
cloud9-20200901serverlessW-20200901serverlessWrite-1X6A5XCM67J5A		Node.js 12.x	1.6 kB	20 秒前

4.2. API Gateway を追加して動作確認を行う

4.2.1. API Gateway REST API 作成

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

The screenshot shows the AWS Management Console with the API Gateway service selected. The main content area displays the following information:

- Amazon API Gateway**: 規模を問わず API を作成、維持、保護する
- A brief description: Amazon API Gateway は開発者が Amazon EC2、AWS Lambda、またはパブリックにアクセス可能なウェブサービスで動作しているバックエンドシステムへの API を作成し、管理するのに役立ちます。Amazon API Gateway を使用すると、API のカスタムクライアント SDK を生成して、バックエンドシステムをモバイル、ウェブ、およびサーバーアプリケーションまたはサービスに接続できます。
- API タイプを選択** section:
 - HTTP API**: OIDC や OAuth2 などの組み込み機能およびネイティブ CORS サポートを使用して、低レイテンシーでコスト効率の高い REST API を構築します。
以下で動作します。
Lambda、HTTP バックエンド
 - 構築** button

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

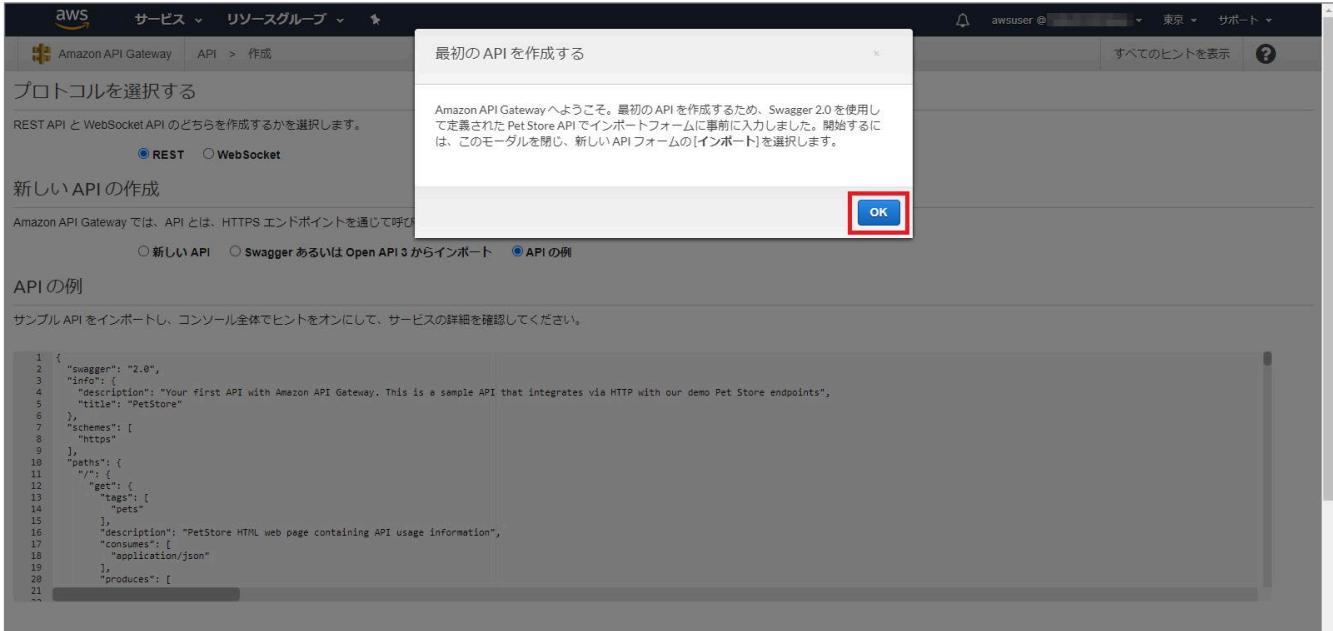
※ **[REST API プライベート]** の方ではありませんので注意してください

The screenshot shows the AWS Management Console with the API Gateway service selected. The main content area displays three options under **API タイプを選択**:

- WebSocket API**: チャットアプリケーションやダッシュボードなど、リアルタイムのユースケース向けの永続的な接続を使用する WebSocket API を構築します。
以下で動作します。
Lambda、HTTP、AWS サービス
- REST API**: API 管理機能とともに、リクエストとレスポンスを完全に制御できる REST API を開発します。
以下で動作します。
Lambda、HTTP、AWS サービス
- REST API プライベート**: VPC 内からのみアクセス可能な REST API を作成します。
以下で動作します。
Lambda、HTTP、AWS サービス

The **REST API** section is highlighted with a red box, and the **構築** button is also highlighted with a red box.

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。

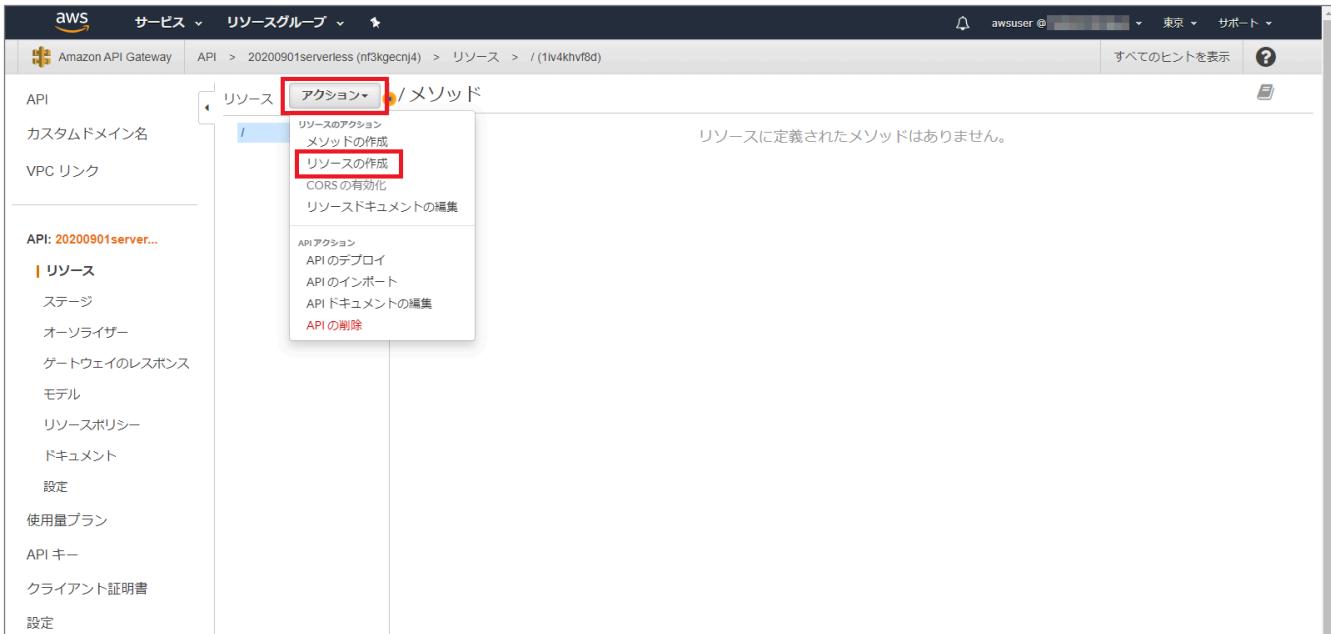
(YYYYMMDD は本日の日付)

[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 'new API' radio button is selected. The 'API Name' field contains '20200901serverless'. The 'Endpoint Type' dropdown is set to 'Region'. The 'Required' label is at the bottom left, and the 'Create API' button is at the bottom right.

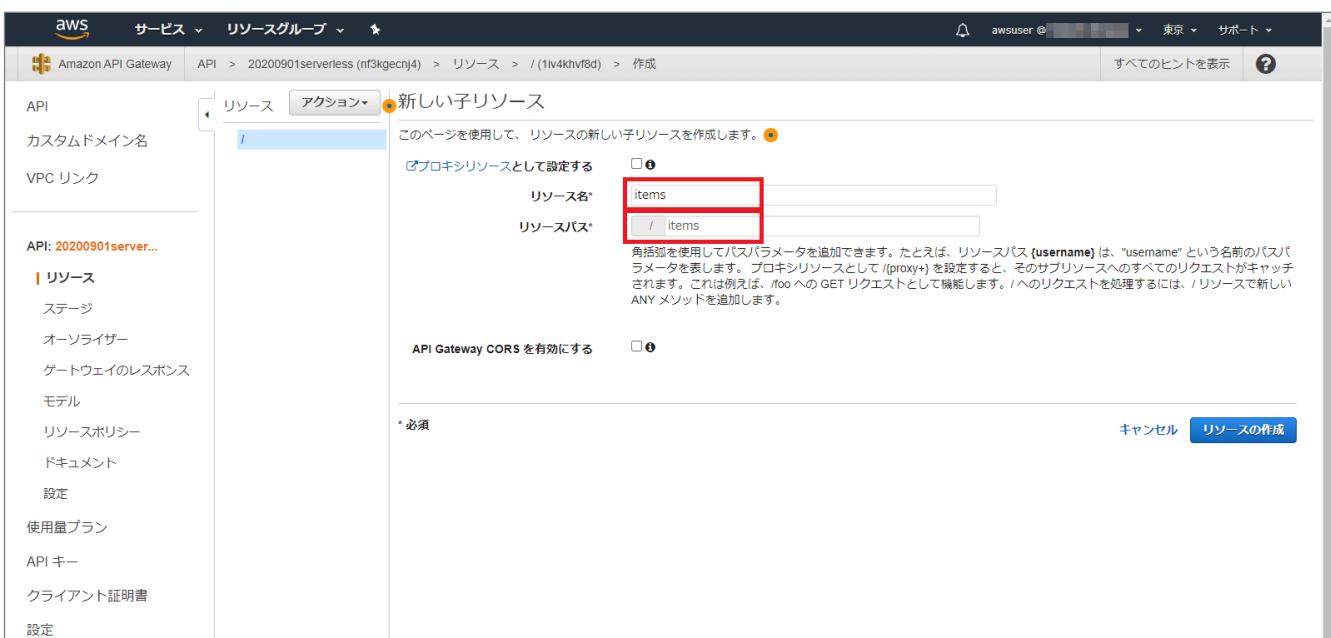
4. [APIの作成] をクリックします。

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。

The screenshot shows the AWS API Gateway console. The navigation bar at the top includes the AWS logo, a 'サービス' dropdown, a 'リソースグループ' dropdown, and user information 'awsuser @ [REDACTED] 東京 サポート'. Below the navigation is a search bar with placeholder text 'すべてのヒントを表示' and a help icon.

The main content area has a left sidebar with a tree view:

- API (selected)
- カスタムドメイン名
- VPC リンク

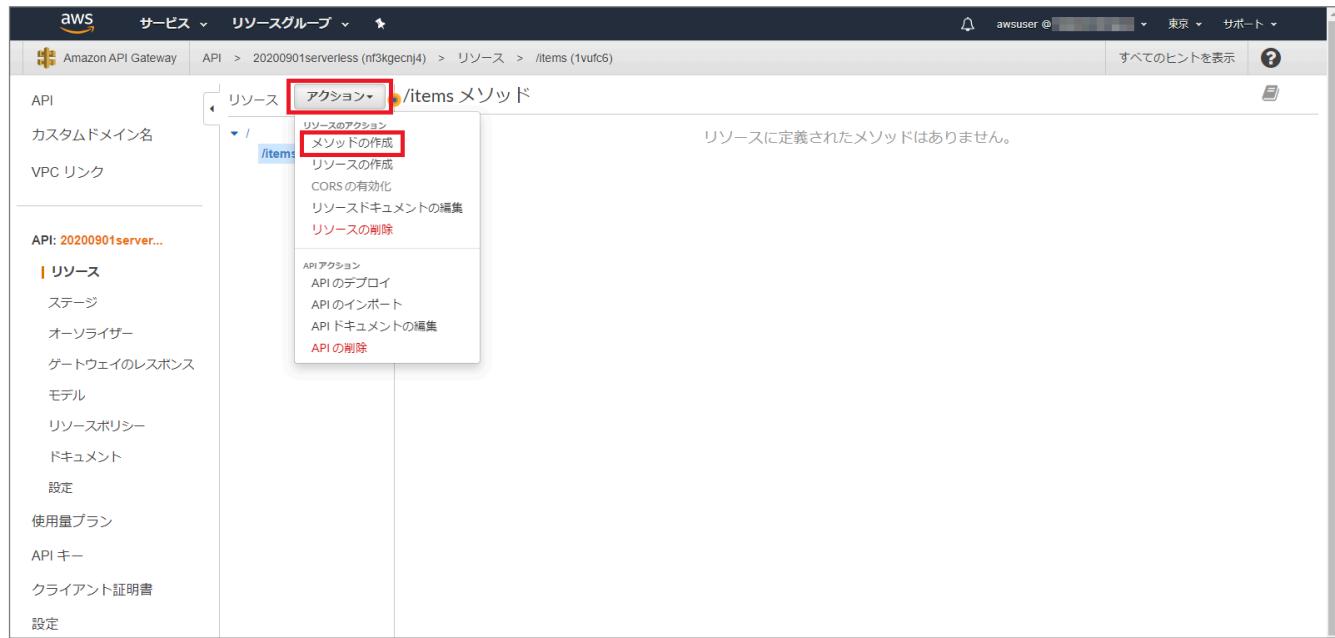
Below the sidebar, the text 'API: 20200901server...' is followed by a list of options:

- | リソース (selected)
- ステータス
- 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
- 設定

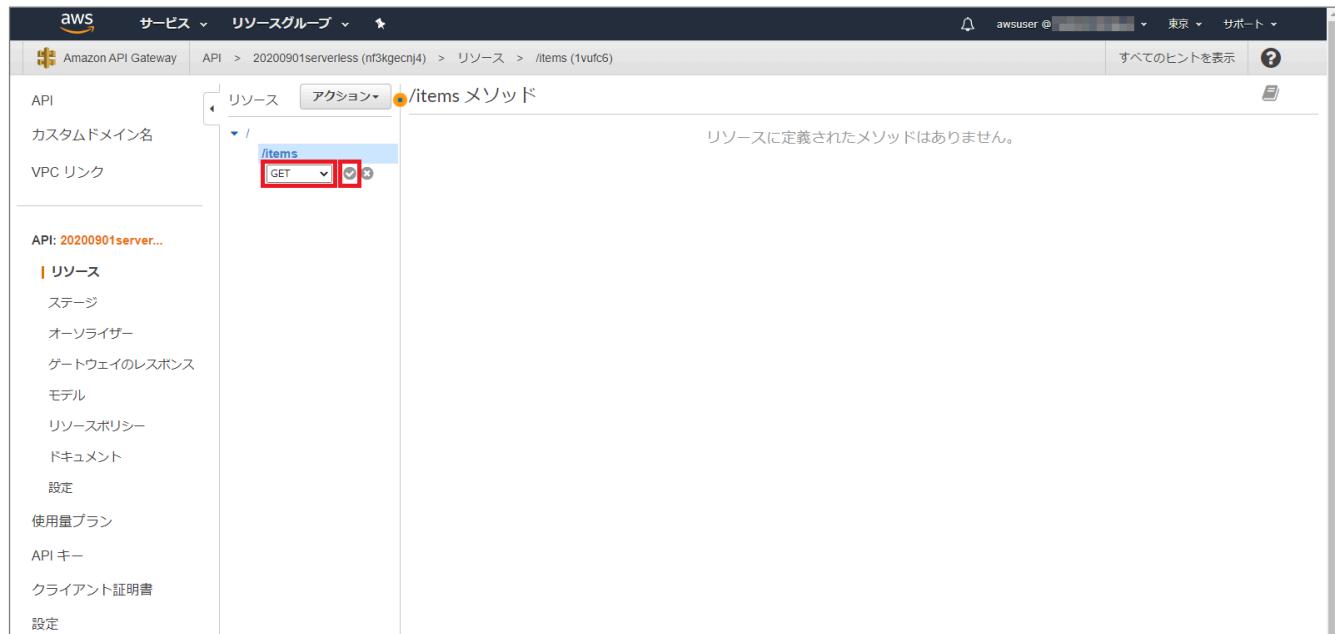
The main panel displays the path '/items' under the 'アクション' tab. A message states 'リソースに定義されたメソッドはありません。' (No methods defined for this resource).

4.2.2. API に GET メソッドを追加

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



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



3. 【統合タイプ】はデフォルトの【Lambda 関数】のままにします。

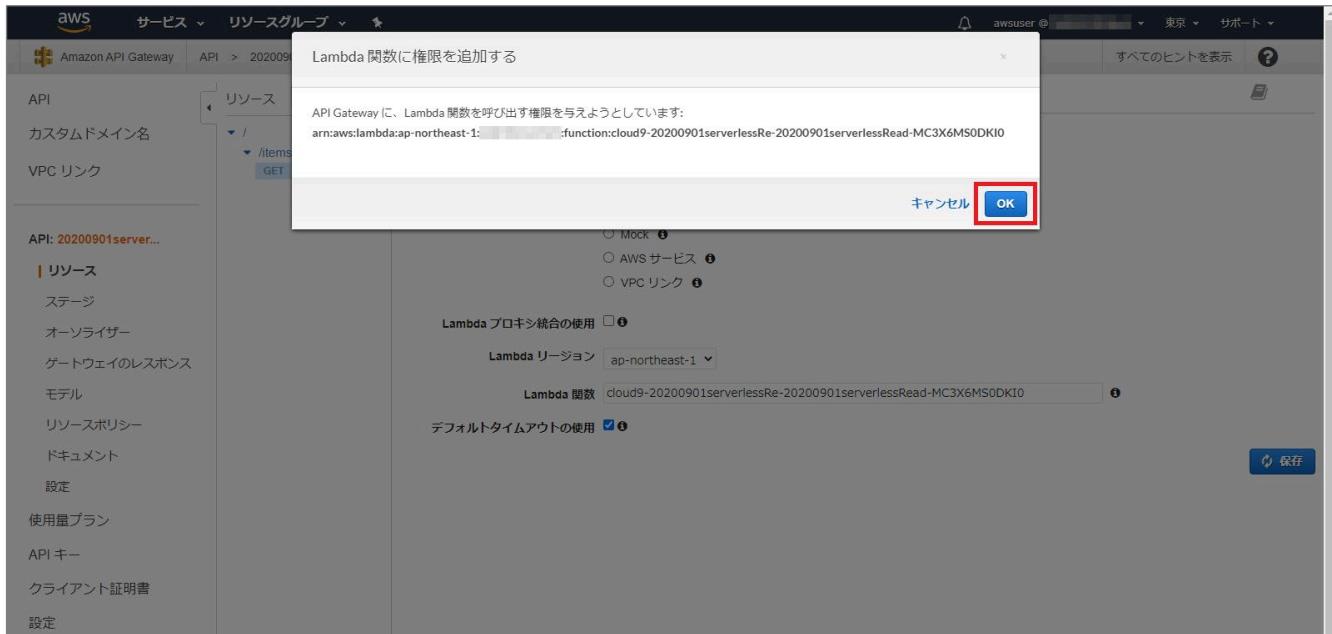
【Lambda プロキシ統合の使用】のチェックは外したままにします。

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

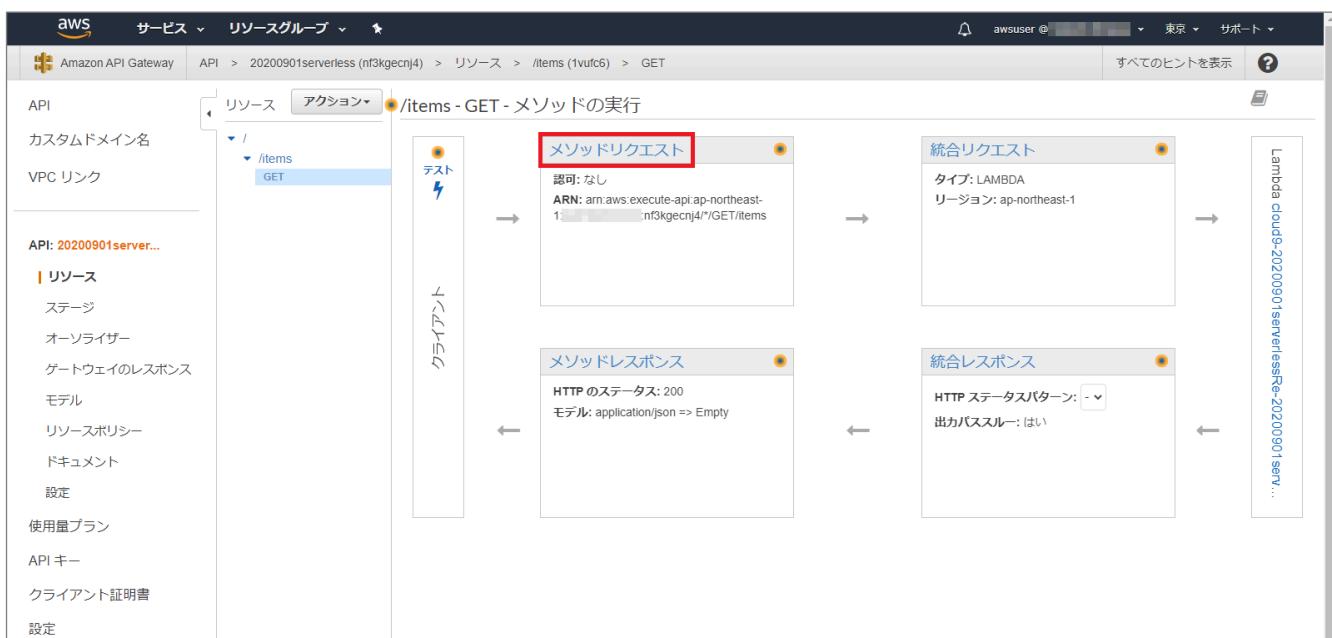
The screenshot shows the AWS Lambda function configuration interface. On the left, there's a sidebar with various API settings like 'カスタムドメイン名' and 'VPC リンク'. The main area shows an API named '20200901serverless...' with a resource path '/items' and an action 'GET'. Underneath, it says '新しいメソッドの統合ポイントを選択します' (Select a new integration point for the method). A dropdown menu for '統合タイプ' (Integration Type) is open, with 'Lambda 関数' selected (indicated by a red box). Below that, there's a checkbox for 'Lambda プロキシ統合の使用' (Proxy integration), which is unchecked. The 'Lambda リージョン' (Region) is set to 'ap-northeast-1'. The 'Lambda 関数' field contains the value 'cloud9-20200901serverlessRe-20200901serverlessRead-MC3X6MS0DKI0', also highlighted with a red box. At the bottom right, there's a blue '保存' (Save) button.

This is a zoomed-in view of the Lambda function selection dropdown from the previous screenshot. It shows two options: 'AWS サービス' and 'VPC リンク', both with their respective icons. Below that is a section titled 'Lambda プロキシ統合の使用' with a checkbox. Underneath is a 'Lambda リージョン' dropdown set to 'ap-northeast-1'. The 'Lambda 関数' dropdown is open, showing the value '20200901' (highlighted with a red box). Below the dropdown, a tooltip displays two Lambda function names: 'cloud9-20200901serverlessRe-20200901serverlessRead-MC3X6MS0DKI0' and 'cloud9-20200901serverlessW-20200901serverlessWrite-1X6A5XCM67J5A'. At the bottom, there's a 'デフォルトタイムアウトの使用' (Use default timeout) checkbox.

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 Lambda API Gateway console. The left sidebar lists various service options like API, VPC Link, and API Keys. The main panel shows a hierarchical path: API > 20200901serverless (nt3kgecnj4) > リソース > /items (1vufc6) > GET. On the right, under the 'アクション' tab, there's a 'メソッドの実行' section for '/items - GET - メソッドドリクエスト'. Below it, the '設定' section is expanded, revealing the 'URL クエリ文字列パラメータ' section. This section contains a table with columns '名前', '必須', and 'キャッシュ'. A single row is present with the value 'クエリ文字列なし'. There are also sections for 'HTTP リクエストヘッダー', 'リクエスト本文', and 'SDK 設定'.

7. [クエリ文字列の追加] をクリックします。

This screenshot is identical to the previous one, showing the AWS Lambda API Gateway console with the same navigation and settings. The difference is that the 'クエリ文字列の追加' button within the 'URL クエリ文字列パラメータ' section is now highlighted with a red box, indicating the user action.

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

The screenshot shows the AWS API Gateway console. On the left, there's a sidebar with various options like API, カスタムドメイン名, VPC リンク, etc. The main area shows an API named '20200901serverless'. Under the 'リソース' tab, there's a 'アクション' dropdown set to 'GET' for the '/items' endpoint. The '設定' section is expanded, showing '認可なし', 'リクエストの検証なし', and 'APIキーの必要性 false'. Below this, the 'URL クエリ文字列パラメータ' section is expanded, showing a table with one row. The first column is '名前' (Name), containing 'artist', which is highlighted with a red box. The second column is '必須' (Required), with a checkbox also highlighted with a red box. The third column is 'キャッシュ' (Cache). At the bottom right of the table, there are two small icons: a pencil and a delete symbol.

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

This screenshot is similar to the previous one but shows a different state. The '必需' (Required) checkbox for the 'artist' parameter is now unchecked. A tooltip 'クエリ文字列の追加' (Add query string parameter) is visible above the '名前' (Name) input field. The rest of the interface is identical to the previous screenshot.

10. [統合リクエスト] をクリックします。

The screenshot shows the AWS API Gateway console. On the left, the navigation pane includes options like 'API', 'カスタムドメイン名', 'VPC リンク', and 'API: 20200901server...'. The main area displays a flow diagram for a GET request to '/items'. The steps are: 'メソッド実行' (Method Execution) -> '統合リクエスト' (Integration Request) -> 'メソッドレスポンス' (Method Response) -> '統合レスポンス' (Integration Response). The '統合リクエスト' step is highlighted with a red box.

11. 一番下の [マッピングテンプレート] をクリックして展開します。

The screenshot shows the expanded configuration for the '統合リクエスト' (Integration Request) step. It includes fields for '統合タイプ' (Integration Type) set to 'Lambda 関数', 'Lambda リージョン' (Region) set to 'ap-northeast-1', and '実行ロール' (Execution Role). There are also sections for '発信者の認証情報を使用した呼び出し' (Use caller authentication information) and '認証情報キャッシュ' (Cache authentication information). At the bottom, the 'マッピングテンプレート' (Mapping Template) section is highlighted with a red box.

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。

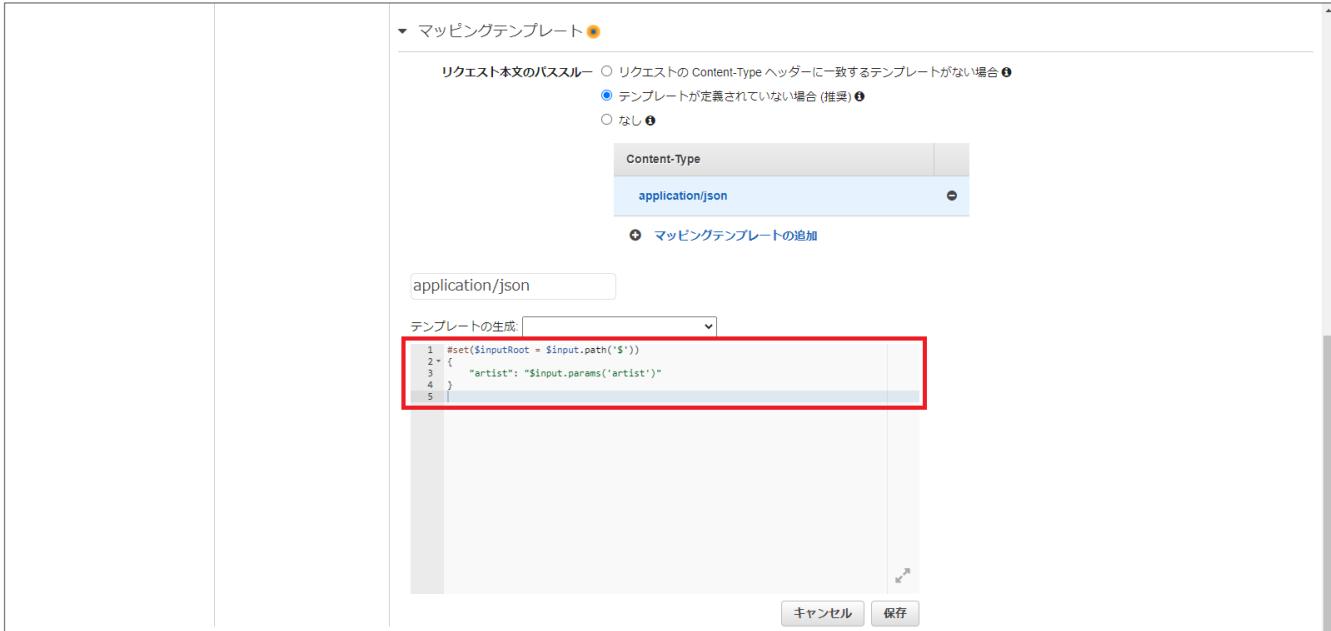
[マッピングテンプレートの追加] をクリックします。

The screenshot shows the 'Mapping Template' configuration screen. On the left, there are navigation links: 'APIキー', 'クライアント証明書', and '設定'. The main area has sections for 'URL パスパラメータ', '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 is a dropdown menu with three options: 'リクエスト本文のバスルート' (selected), 'テンプレートが定義されていない場合 (推奨)' (highlighted with a red box), and 'なし'. Below this is a 'Content-Type' input field containing 'application/json'. At the bottom right of the input field is a small red box around the 'マッピングテンプレートの追加' button.

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

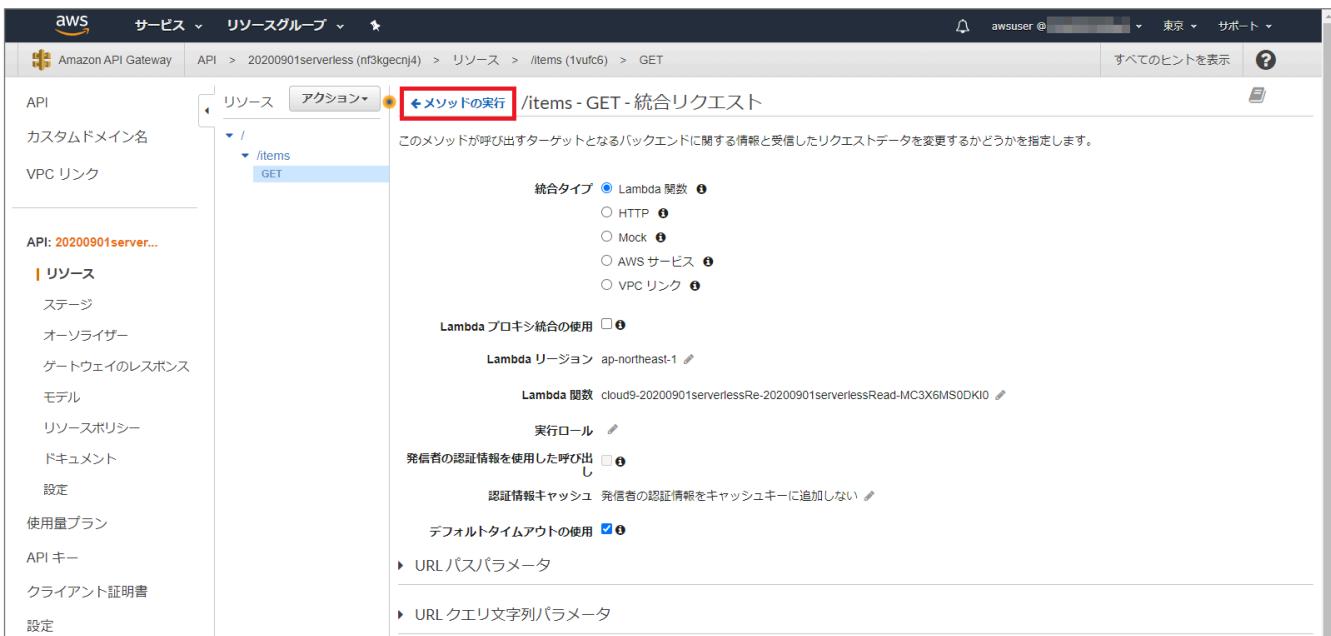
This screenshot shows the same configuration screen as the previous one, but with the 'Content-Type' input field now containing 'application/json'. A red box highlights the checkbox to the right of the input field, which is also checked. The rest of the interface is identical to the first screenshot.

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。

The screenshot shows the AWS API Gateway test interface. On the left, the navigation pane lists various API settings like custom domains, VPC links, and stages. The main area shows a flow diagram with four boxes: 'メソッド実行' (Method Execution) containing a 'Test' button (which is highlighted with a red box), '統合リクエスト' (Integrated Request) with Lambda details, 'メソッドレスポンス' (Method Response) with an empty body, and '統合レスポンス' (Integrated Response) with status code and CORS settings. Arrows indicate the flow from the request to the response.

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

←メソッドの実行 /items - GET - メソッドテスト

指定された入力で メソッドに対してテストコールを行います。

パス

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

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。

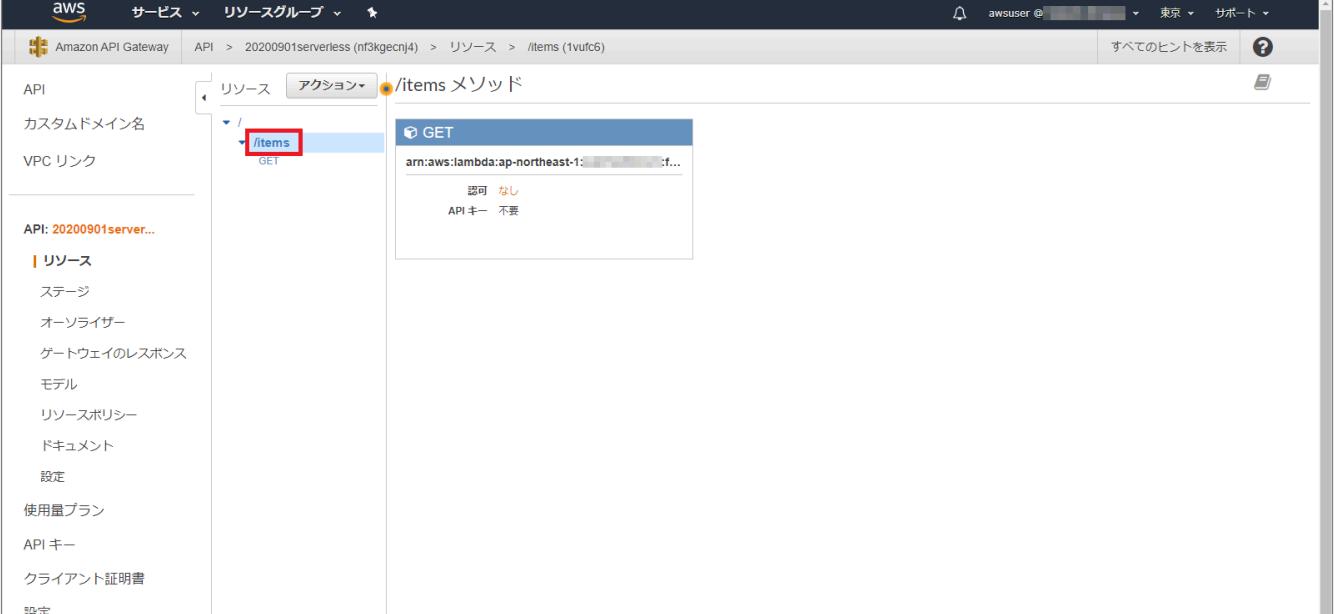
[ステータス] が [200] と表示されていれば成功です。

[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 Lambda function configuration interface. On the left, the sidebar lists various settings like API Gateway, Lambda functions, and CloudWatch Metrics. In the main area, a specific Lambda function is selected. The 'Handler' dropdown is set to 'index.handler'. Below it, the 'Role' dropdown is set to 'lambda-role'. Under the 'Environment' section, there is a 'Variables' table with one entry: 'Artist' with the value 'Michael Jackson'. At the bottom of the page, there is a large blue button labeled 'Test' with a lightning bolt icon, which is highlighted with a red box.

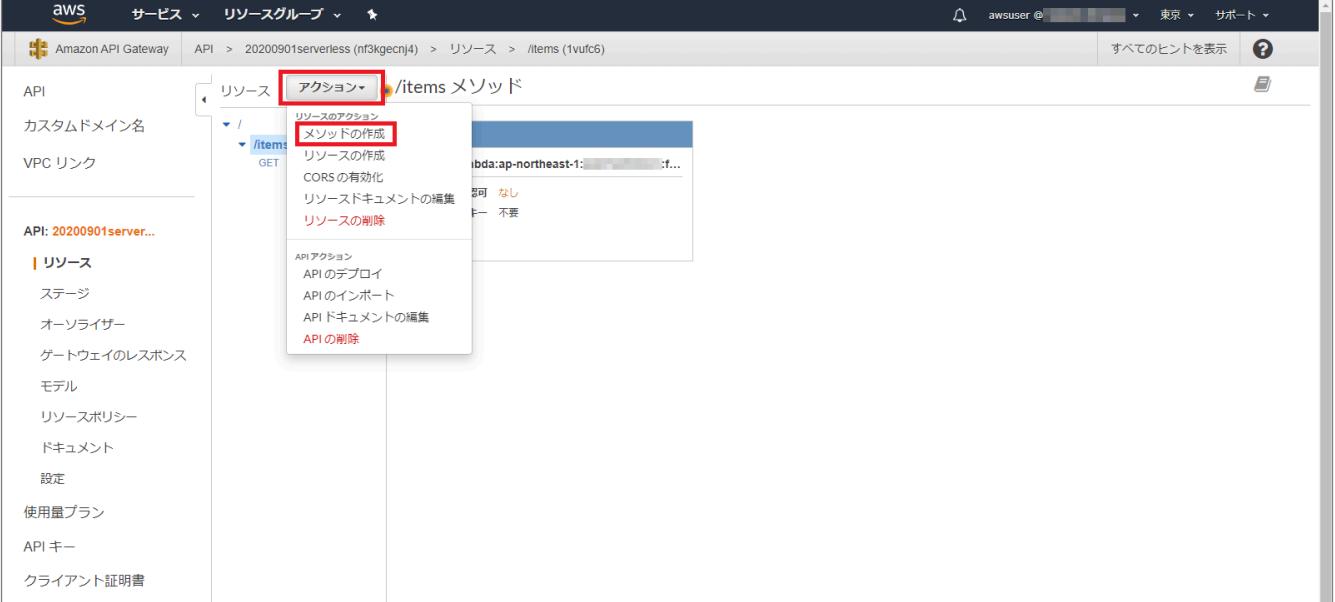
4.2.3. API に POST メソッドを追加

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



The screenshot shows the AWS Lambda console interface. On the left, there's a sidebar with various navigation options like Services, Resource Groups, and APIs. The main area shows an API named '20200901serverless'. Under the 'Resources' section, there's a tree view with a node labeled '/items'. This node is highlighted with a red box. To the right of the tree, there's a detailed view of a GET method for the '/items' resource, showing its ARN, authentication requirements (none), and API key requirements (none). The entire interface has a light gray background with blue and white UI elements.

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 from the same AWS Lambda console as the previous one, but it shows a different state. The '/items' resource is still selected. A red box highlights the 'Actions' dropdown menu, which is open. Inside the dropdown, the 'Create Method' option is also highlighted with a red box. The rest of the dropdown menu includes other options like 'Resource Creation', 'Method Configuration', 'Resource Document Edit', and 'Resource Deletion'. The overall layout is identical to the first screenshot, with the same sidebar and main API details.

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, there's a sidebar with navigation links like 'API', 'カスタムドメイン名', 'VPC リンク', etc. The main area shows a function named 'cloud9-20200901serverlessW-20200901serverlessWrite-1X6A5XCM67J5A'. Under the 'Actions' tab, there's a dropdown menu for selecting a method. The 'POST' option is highlighted with a red box. To the right, there's a detailed view of the function's configuration, including its ARN, region, and execution role.

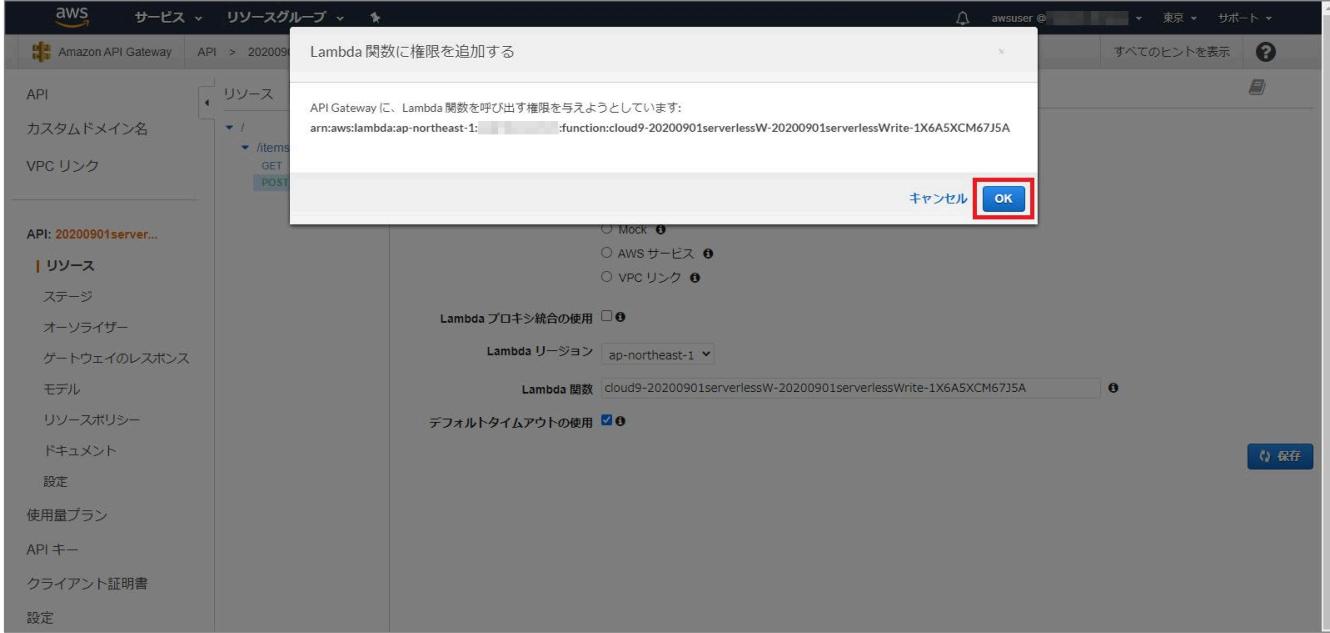
4. [統合タイプ] はデフォルトの [Lambda 関数] のままにします。

[Lambda プロキシ統合の使用] のチェックは外したままにします。

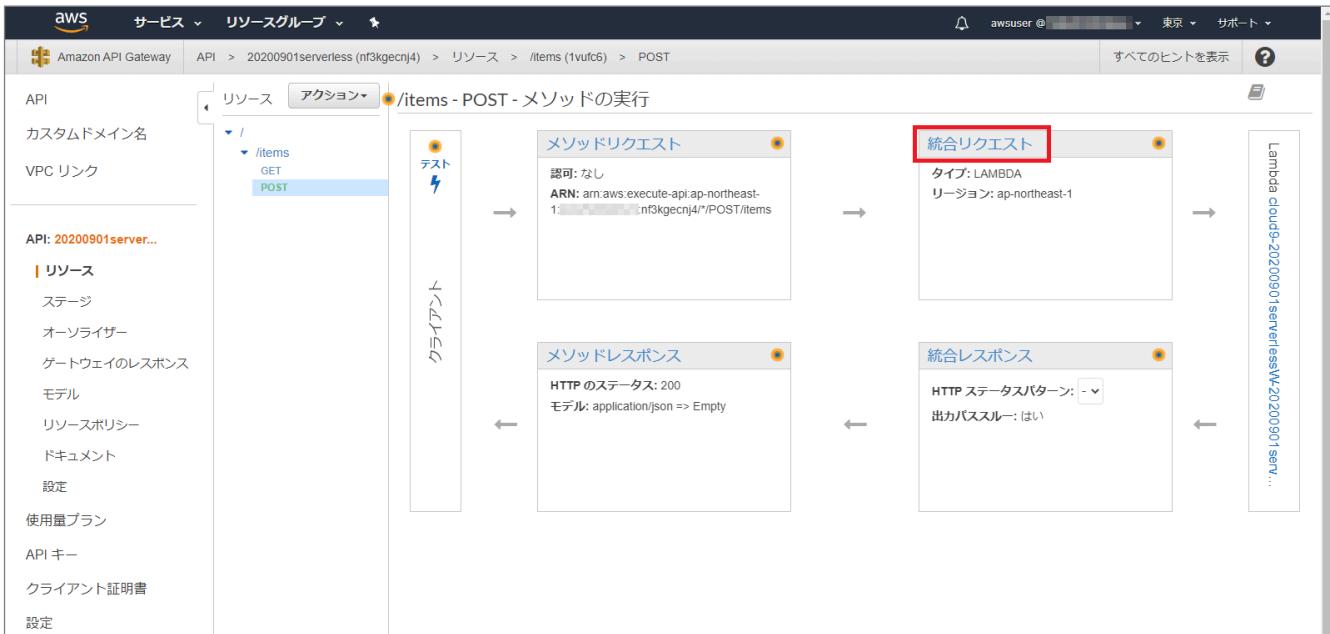
[Lambda 関数] 欄欄に、前節で作成した Lambda 関数のうち「Write」の方の関数名を入力します。

The screenshot shows the AWS API Gateway configuration interface. It's similar to the Lambda interface but for a different service. The main area shows a resource path '/items' with a POST method selected. Under the 'Integration' section, the '統合タイプ' (Integration Type) is set to 'Lambda 関数' (Lambda Function), which is highlighted with a red box. Below it, there's a field for 'Lambda 関数' (Lambda Function) containing the value 'cloud9-20200901serverlessW-20200901serverlessWrite-1X6A5XCM67J5A', also highlighted with a red box. There are other options like 'HTTP', 'Mock', 'AWS サービス', and 'VPC リンク' available but not selected. A checkbox for 'Lambda プロキシ統合の使用' (Proxy Integration) is unchecked. A '保存' (Save) button is visible at the bottom right.

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



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



7. 一番下の [マッピングテンプレート] をクリックして展開します。

The screenshot shows the AWS Lambda function configuration in the AWS API Gateway console. The left sidebar lists various API-related settings. The main panel shows a POST method for the '/items' resource. Under 'Lambda Function' settings, the 'Mapping Template' section is highlighted with a red box. This section contains a dropdown menu with options: 'テンプレートが定義されていない場合 (推奨)' (selected), 'なし' (None), and 'リクエスト本文のパスルール'.

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. It includes sections for 'URL Path Parameters', 'URL Query String Parameters', 'HTTP Headers', and 'Mapping Template'. The 'Mapping Template' section is expanded, showing a dropdown menu with three options: 'テンプレートが定義されていない場合 (推奨)' (selected and highlighted with a red box), 'なし' (None), and 'リクエスト本文のパスルール'. Below the dropdown is a 'Content-Type' input field containing the placeholder text 'マッピングテンプレートが定義されていません。リクエスト本文は結合エンドポイントに渡されます'.

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。

The screenshot shows the AWS Lambda function configuration interface. On the left, there's a sidebar with 'APIキー', 'クライアント証明書', and '設定' options. The main area has sections for 'URL パスパラメータ', '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'. The 'マッピングテンプレート' section is expanded, showing three radio button options: 'リクエスト本文のバスルール' (selected), 'リクエストの Content-Type ヘッダーに一致するテンプレートがない場合', and 'なし'. Below this is a 'Content-Type' input field containing 'application/json', with a red box highlighting it. To the right of the input field is a small circular icon with a red border, also highlighted with a red box. At the bottom of the template section is a link 'マッピングテンプレートの追加'.

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。

This screenshot shows the 'マッピングテンプレート' editor. It has the same structure as the previous screenshot, with the 'Content-Type' input field containing 'application/json' and the circular icon highlighted. Below the input field is a 'テンプレートの生成:' dropdown menu. Underneath the menu is a code editor window containing the following JSON template:

```
1 $input.json('$')
```

The entire code editor window is highlighted with a large red box. At the bottom of the editor are two buttons: 'キャンセル' and '保存'.

11. [保存] をクリックします。 (画面は変わりません)

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

The screenshot shows the AWS API Gateway configuration interface. On the left, the navigation sidebar includes options like 'API', 'カスタムドメイン名', 'VPC リンク', and 'API: 20200901server...'. The main panel displays the configuration for the POST method of the '/items' resource. A red box highlights the '←メソッドの実行' button at the top left of the configuration area. The configuration details include:

- リソース**: /items - POST - 統合リクエスト
- 統合タイプ**: Lambda 関数 (selected)
- Lambda リージョン**: ap-northeast-1
- 実行ロール**: cloud9-20200901serverlessW-20200901serverlessWrite-1X6A5XCM67J5A
- デフォルトタイムアウトの使用**: 有効
- URL パスパラメータ** (and URL クエリ文字列パラメータ)

13. [テスト] をクリックします。

The screenshot shows the AWS API Gateway test interface. It displays the flow of data between the API endpoint and the Lambda function. A red box highlights the 'テスト' (Test) button on the left side of the flow diagram. The flow consists of the following components:

- リソース**: /items - POST - メソッドの実行
- メソッドリクエスト**: 認可: なし, ARN: arn:aws:execute-api:ap-northeast-1:**cloud9-20200901serverlessW-20200901serverless**/POST/items
- 統合リクエスト**: タイプ: LAMBDA, リージョン: ap-northeast-1
- メソッドレスポンス**: HTTP のステータス: 200, モデル: application/json => Empty
- 統合レスポンス**: HTTP ステータスパターン: 200, 出力バスルート: はい

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. The URL in the address bar is `API > 20200901serverless (nf3kgecnj4) > リソース > /items (1vufc6) > POST`. The left sidebar has 'API' selected. The main area shows the 'POST' method under the '/items' resource. The 'リクエスト本文' (Request Body) field contains the following JSON:

```
1: {  
2:   "artist": "Madonna",  
3:   "title": "Like a Virgin"  
4: }  
5: |
```

This field is highlighted with a red 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: API > 20200901serverless (nf3kgecnj4) > リソース > /items > POST. The right panel shows the test results for the POST method. The status is highlighted with a red box as "ステータス: 200". The log panel shows the execution log for the request, confirming a successful execution.

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 navigation menu for DynamoDB. The main area displays the 'Artist, Title' table. A specific item, 'Like a Virgin' by Madonna, is selected and highlighted with a red box. The top right corner of the main window has a refresh button, which is also 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。

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

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

The screenshot shows the AWS 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, etc. The main area shows an API named "20200901serverless" with a resource path "/items". Under this path, methods "GET" and "POST" are listed. A sub-menu titled "CORS の有効化" is open, showing configuration for "20200901serverless API のゲートウェイレスポンス". It lists "メソッド" (POST, GET, OPTIONS) and "Access-Control-Allow-Methods" (GET, OPTIONS, POST). There's also a section for "Access-Control-Allow-Headers" with the value "Content-Type,X-Amz-Date,Authorization" and "Access-Control-Allow-Origin" set to "*". A red box highlights the button "CORS を有効にして既存の CORS ヘッダーを置換" (Enable CORS and replace existing CORS header).

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

The screenshot shows a confirmation dialog box titled "メソッド変更の確認" (Method Change Confirmation) overlying the API Gateway interface. The dialog contains a list of changes: "OPTIONS メソッドを作成する", "200 メソッドレスポンスを空のレスポンスモデルとともに OPTIONS メソッドに追加する", etc. At the bottom right of the dialog is a button labeled "はい、既存の値を置き換えます" (Yes, replace existing values), which is highlighted with a red box.

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

The screenshot shows the AWS Lambda function configuration for the '20200901serverless' function. In the 'Actions' tab, the 'CORSの有効化' (Enable CORS) action is selected. A red box highlights the status message: 'リソースは CORS に対して設定されました。上記の出力にエラーが表示される場合は、エラーメッセージを確認し、必要に応じて失敗したステップをメソッドエディターを通じて手動で実行してみてください。' (The resource has been set up for CORS. If an error is displayed in the output, please check the error message and manually run the failed step in the method editor if necessary.)

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 API Gateway settings page for the '20200901serverless' API. In the left sidebar, the 'Gateway Responses' option under the 'Responses' section is highlighted with a red box. On the right, the 'Gateway Responses' section is visible, showing various response types such as 'DEFAULT 4XX', 'DEFAULT 5XX', and various error codes like 'アクセスが拒否されました' (Access denied). A red box highlights the 'Gateway Responses' section.

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

The screenshot shows the AWS Lambda function configuration page. On the left, there's a sidebar with various options like 'API', 'カスタムドメイン名', 'VPC リンク', etc. The main area is titled 'Gateway Responses' and shows a list of response types. The first item, 'アクセスが拒否されました' (Access Denied), is selected and highlighted with a red box. In the top right corner, there are two buttons: 'デフォルトにリセット' (Reset to Default) and a larger '編集' (Edit) button, which is also 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。

The screenshot shows the same AWS Lambda function configuration page as the previous one, but now the 'レスポンスヘッダーの追加' (Add Response Header) button at the bottom of the 'レスポンスヘッダー' (Response Headers) section is highlighted with a red box. The rest of the interface is identical to the previous screenshot.

9. 以下の通り入力します。

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

The screenshot shows the AWS Lambda API Gateway interface. On the left, a sidebar lists various API settings like 'カスタムドメイン名' and 'VPC リンク'. The main area is titled 'ゲートウェイのレスポンス' (Gateway Responses) and shows a list of response types. One response type, '403 Forbidden', is currently selected. In the 'レスポンスヘッダー' (Response Headers) section, a new header 'Access-Control-Allow-Origin' is being added. The 'Value' field contains the value '*' and is highlighted with a red box. The 'Save' button at the top right is also highlighted with a red box.

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 interface. On the left, there's a sidebar with various service links like 'Lambda', 'API Gateway', 'CloudWatch Metrics', etc. The main area shows an 'API' named '20200901serverless'. Under 'API' is a 'リソース' (Resource) section. A red box highlights the path element '/'. Below it, under 'アクション' (Actions), are 'GET', 'OPTIONS', and 'POST'. A message at the bottom right says 'リソースに定義されたメソッドはありません' (No methods defined for this resource).

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。

This screenshot is similar to the previous one, but the 'Actions' dropdown menu is open over the selected '/'. The menu items are: 'リソースのアクション' (Resource Actions), 'メソッドの作成' (Create Method), 'リソースの作成' (Create Resource), 'CORSの有効化' (Enable CORS), and 'リソースドキュメントの編集' (Edit Resource Document). At the bottom of the dropdown, another menu is open with the option 'API のデプロイ' (Deploy API), which is highlighted with a red box.

3. 以下の通り入力します。

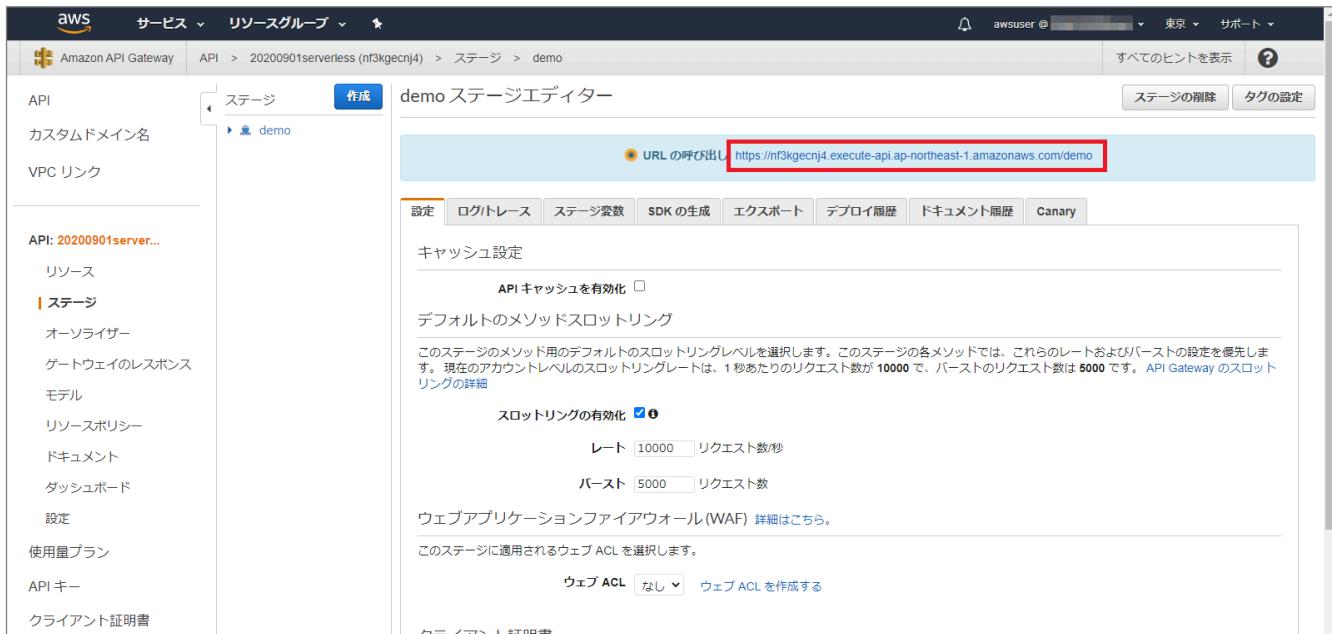
- **デプロイされるステージ:** [新しいステージ] を選択
- **ステージ名:** [demo] と入力
- **ステージの説明:** (省略)
- **デプロイメントの説明:** (省略)



4. [デプロイ] をクリックします。

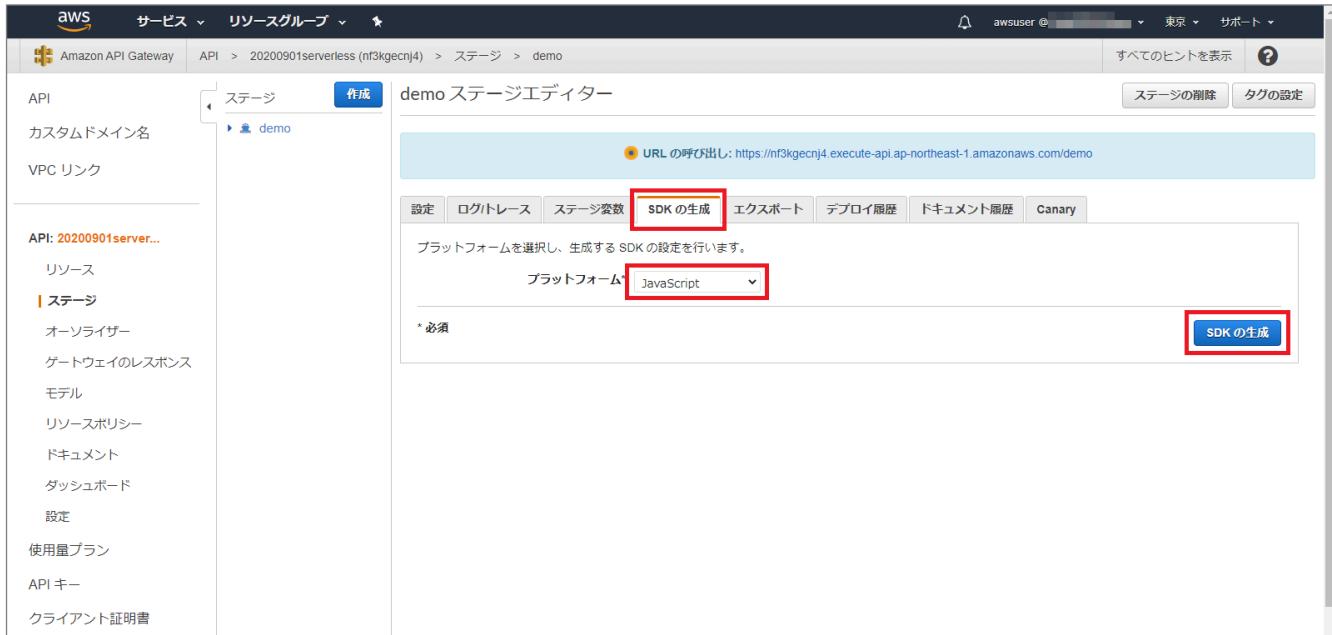
5. デプロイが行われ、ステージ [demo] の画面が表示されます。

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。



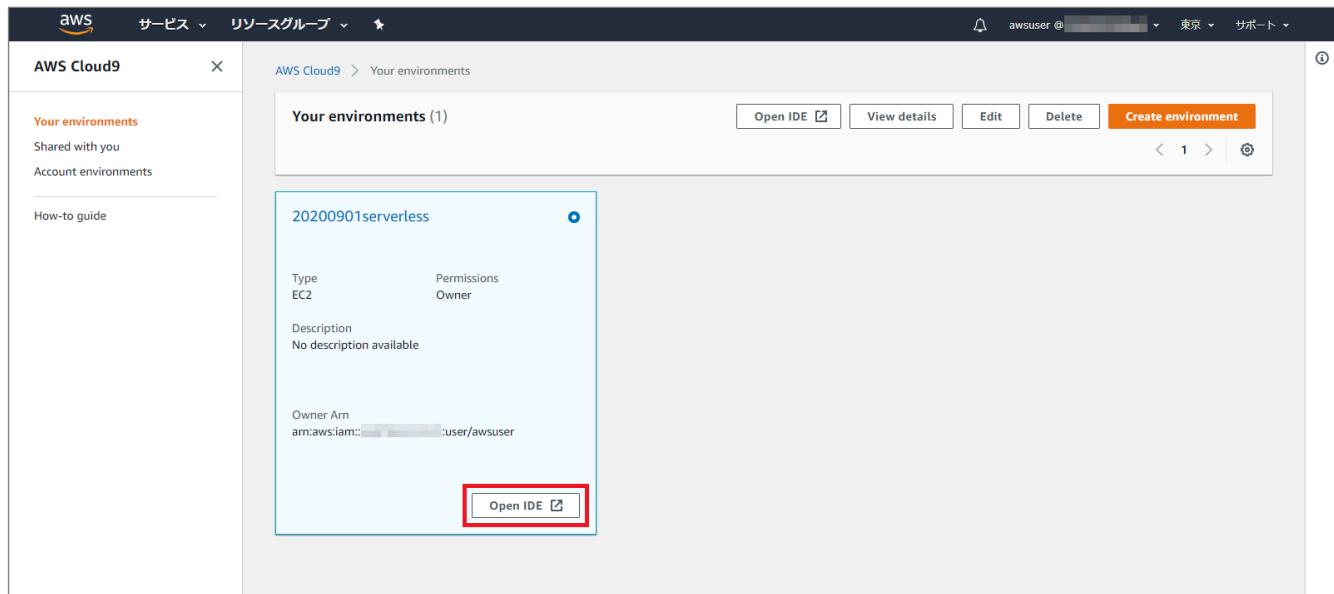
7. ファイル **[javascript_YYYY-MM-DD_HH-MMZ.zip]** のダウンロードが行われるので、任意の場所に保存します。 (後の手順で使用します)

4.2.6. Cloud9 からの動作確認

1. Cloud9 の画面へ移動します。

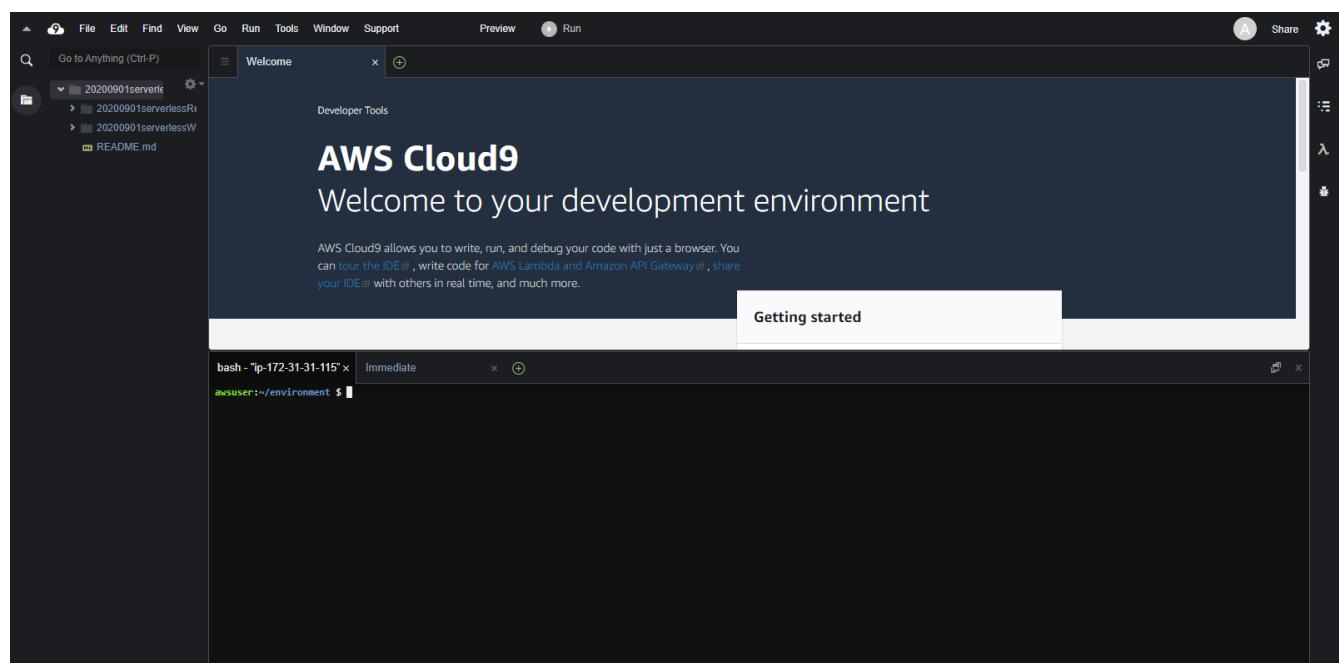
時間が経っている場合には Cloud9 の EC2 インスタンスが停止している可能性があります。

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。

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

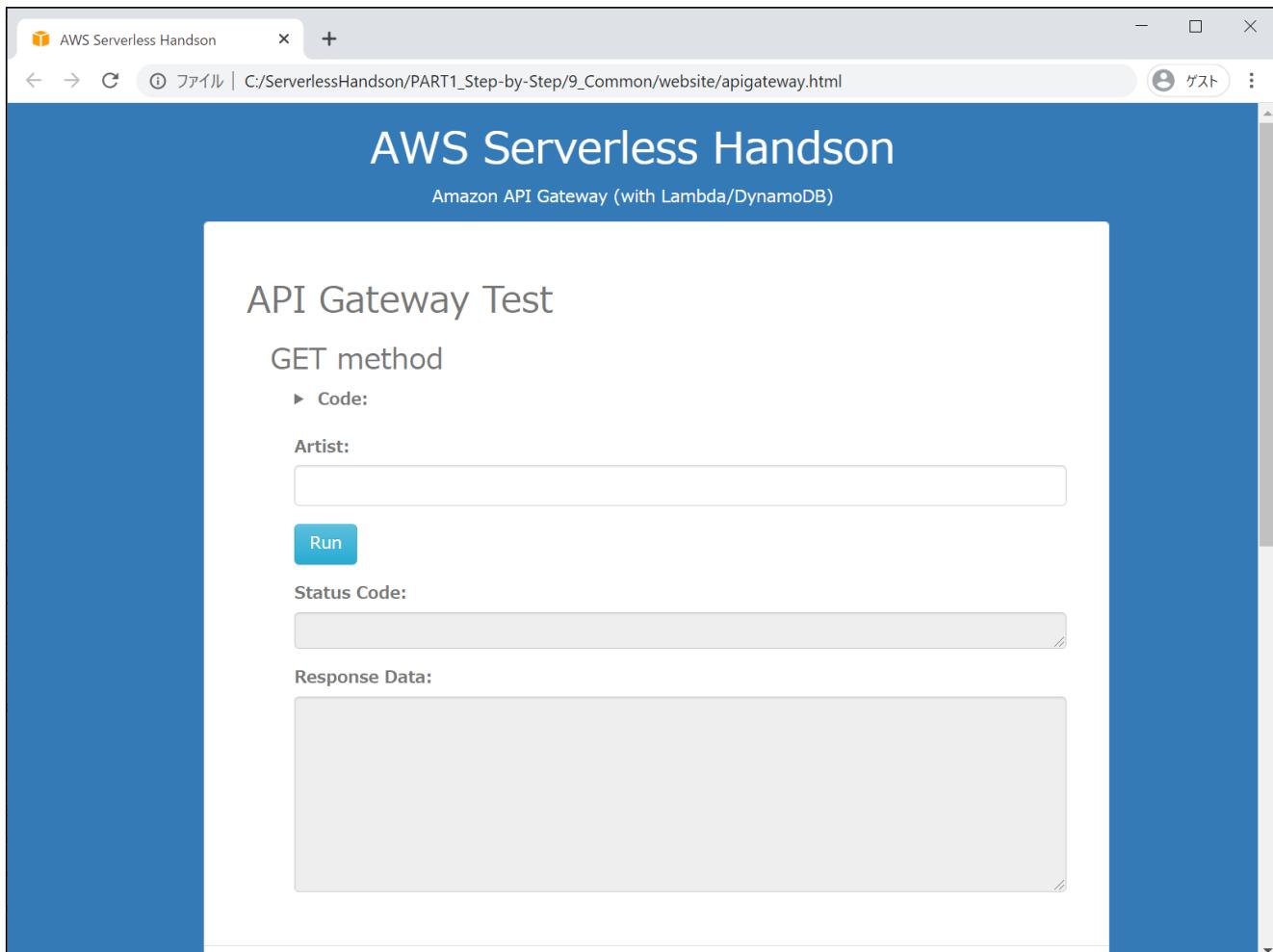
4.2.7. Web ブラウザからの動作確認

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、[webpage] フォルダ配下の構成は以下のようになっているはずです。

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

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

4. [apigateway.html] を Web ブラウザで開きます。



5. まず、GET メソッドを試します。

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

API Gateway Test

GET method

▶ Code:

Artist:

Michael Jackson

Run

Status Code:

Response Data:

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

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

API Gateway Test

GET method

▶ Code:

Artist:

Michael Jackson

Run

Status Code:

200

Response Data:

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

7. 続いて POST メソッドを試します。

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

POST method

▶ Code:

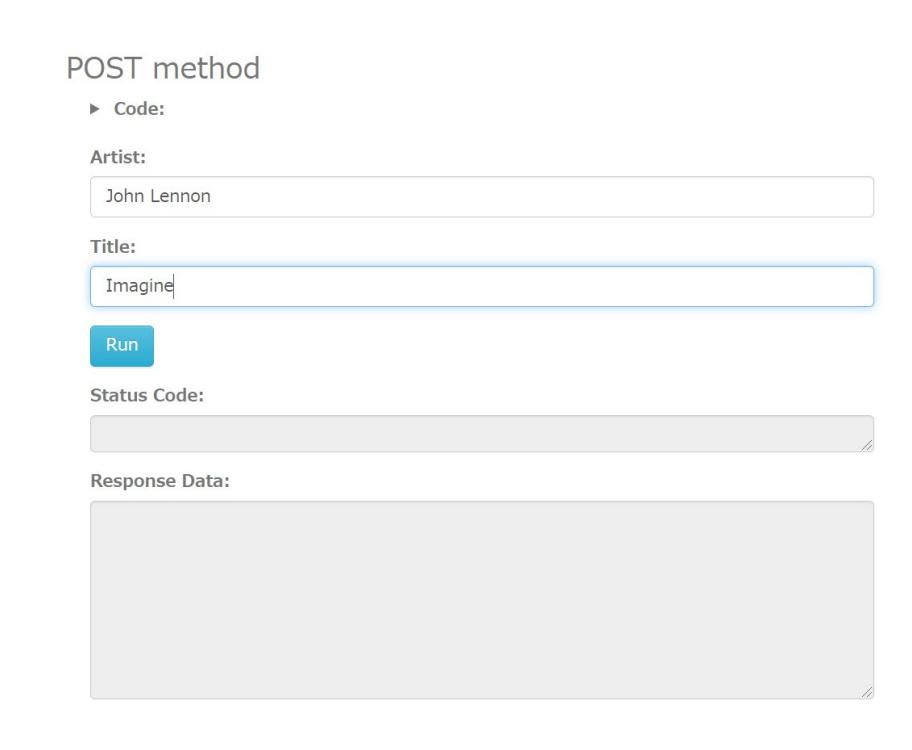
Artist:
John Lennon

Title:
Imagine

Run

Status Code:

Response Data:



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

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

POST method

▶ Code:

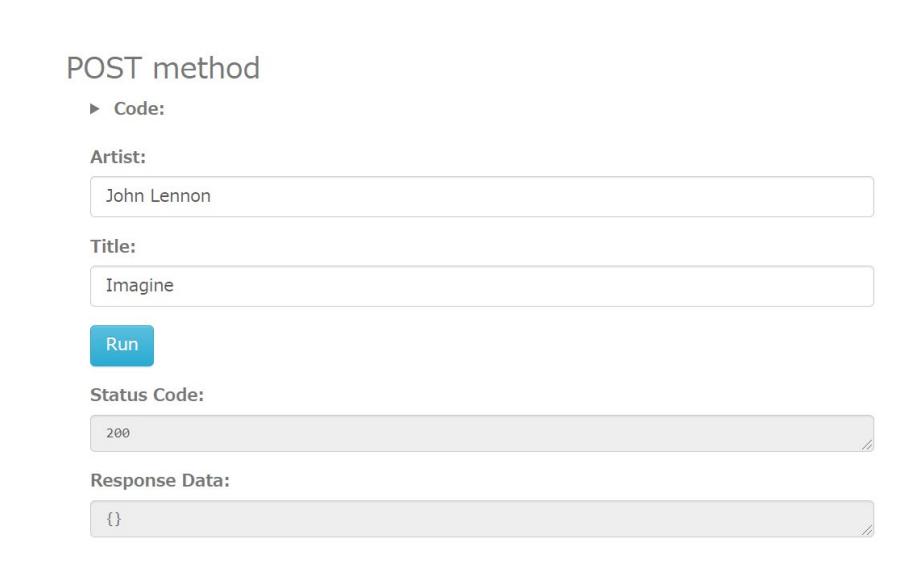
Artist:
John Lennon

Title:
Imagine

Run

Status Code:
200

Response Data:
{}



4.3. Cognito の動作確認を行う

4.3.1. Cognito ユーザープールの作成

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

[ユーザープールの管理] をクリックします。

The screenshot shows the AWS Management Console with the Cognito service selected. The main heading is "Amazon Cognito". Below it, a text block explains that Cognito provides user pools and ID pools for app users. It highlights that user pools support sign-up and sign-in, while ID pools allow access to other AWS services. Two buttons are present: "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 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 AWS service access to users) 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 "User Pool Management" section of the Cognito console. A message at the top states "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 one.). 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.

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

The screenshot shows the 'Create User Pool' wizard. On the left, a sidebar lists options: 名前 (highlighted), 属性, ポリシー (highlighted), MFAとして確認, メッセージのカスタマイズ, タグ, デバイス, アプリクライアント, トリガー, 確認. The main area has two sections: 'User pool name' (Pool Name: 20200901serverless) and 'How to create a user pool'. The 'Confirm Default' button is highlighted with a red box.

4. 画面左側のメニューから [ポリシー] を選択します。

The screenshot shows the 'Create User Pool' wizard. The sidebar shows 'Policy' selected (highlighted). The main area displays policy settings: 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... . Other sections shown include Password (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), Email (送信元: デフォルト, Amazon SESによるEメール配信: はい), MFA (MFAの有効化...), 和 Tag (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。

サービス リソースグループ ★

awsuser @ 東京 サポート

ユーザー プール フェデレーティッドアイデンティティ
ユーザー プールを作成する キャンセル

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

最小長
8

数字を必要とする
 特殊文字を要求する
 大文字を必要とする
 小文字を必要とする

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。

ユーザーに自己サインアップを許可しますか?

管理者のみユーザーの作成を許可するか、ユーザーに自己サインアップを許可するかを選択できます。 詳細は[こちら](#)。

管理者のみユーザーの作成を許可する
 ユーザーに自己サインアップを許可する

管理者が設定した一時パスワードが使用されない期間がどれくらい続くと、有効期限が切れますか?

管理者が設定した一時パスワードが使用されないまま有効期限になるまでの期間は選択できます。これには、管理者が作成したアカウントが含まれます。

有効期限(日数)
7

キャンセル 变更の保存

6. 画面左側のメニューから [アプリケーション] を選択します。

aws サービス リソースグループ ★

awsuser @ [REDACTED] 東京 サポート

ユーザープール フェデレーティッドアイデンティティ
ユーザープールを作成する キャンセル

名前 プール名: 20200901serverless

属性 必須の属性: email

属性 エイリアス属性: エイリアス属性の選択...

属性 ユーザー名属性: ユーザー名属性の選択...

属性 大文字と小文字を区別しないことを有効にし

属性 ますか? [はい]

属性 カスタム属性: カスタム属性の選択...

デバイス

トリガー

確認

確認 パスワードの最小長: 8

確認 パスワードポリシー: 要件なし

確認 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

確認 送信元 E メールアドレス: デフォルト

確認 Amazon SES による E メール配信: はい

確認 MFA: MFA の有効化...

確認 確認: E メール

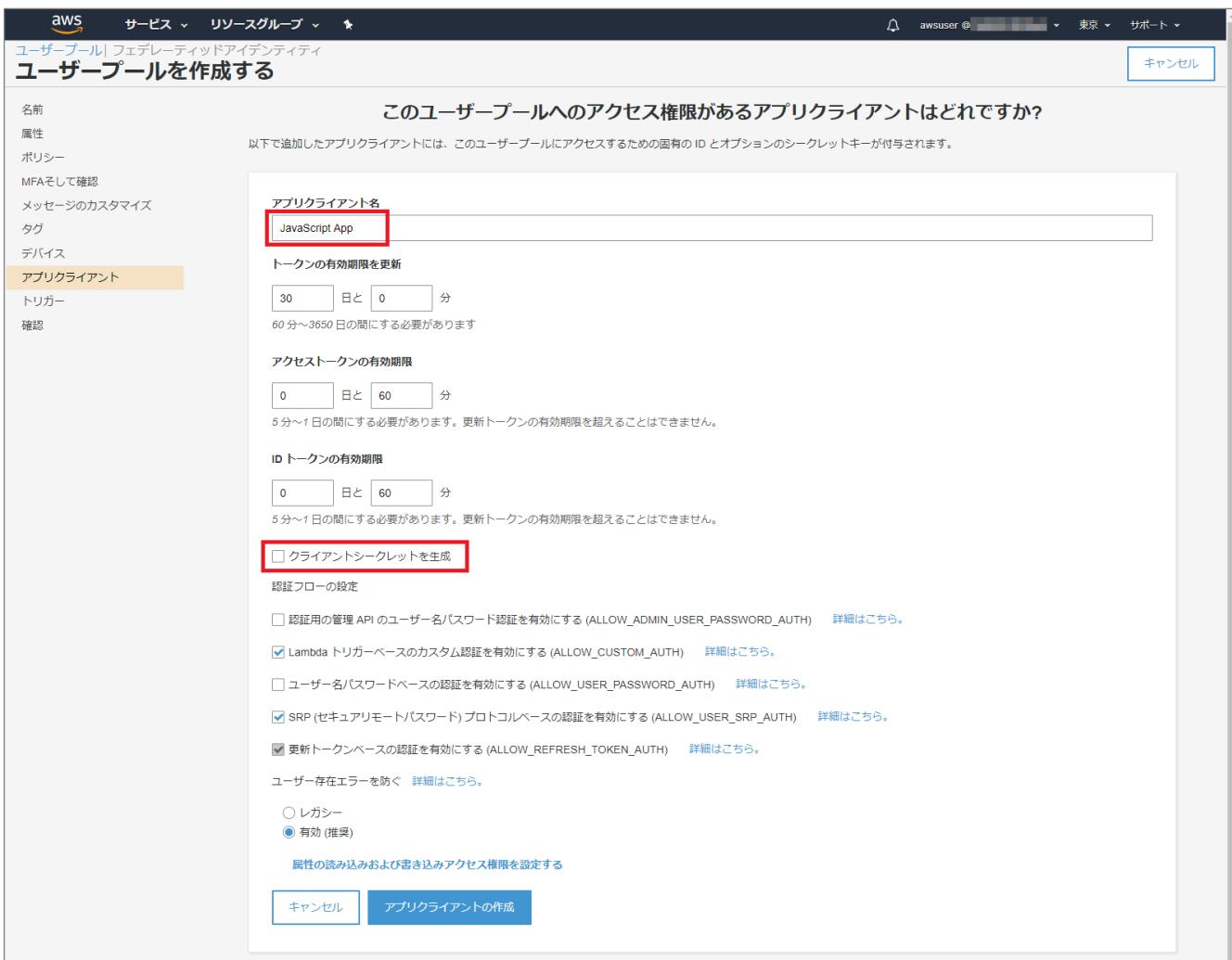
タグ タグ: ユーザープールのタグを選択

7. [アプリケーションの追加] をクリックします。



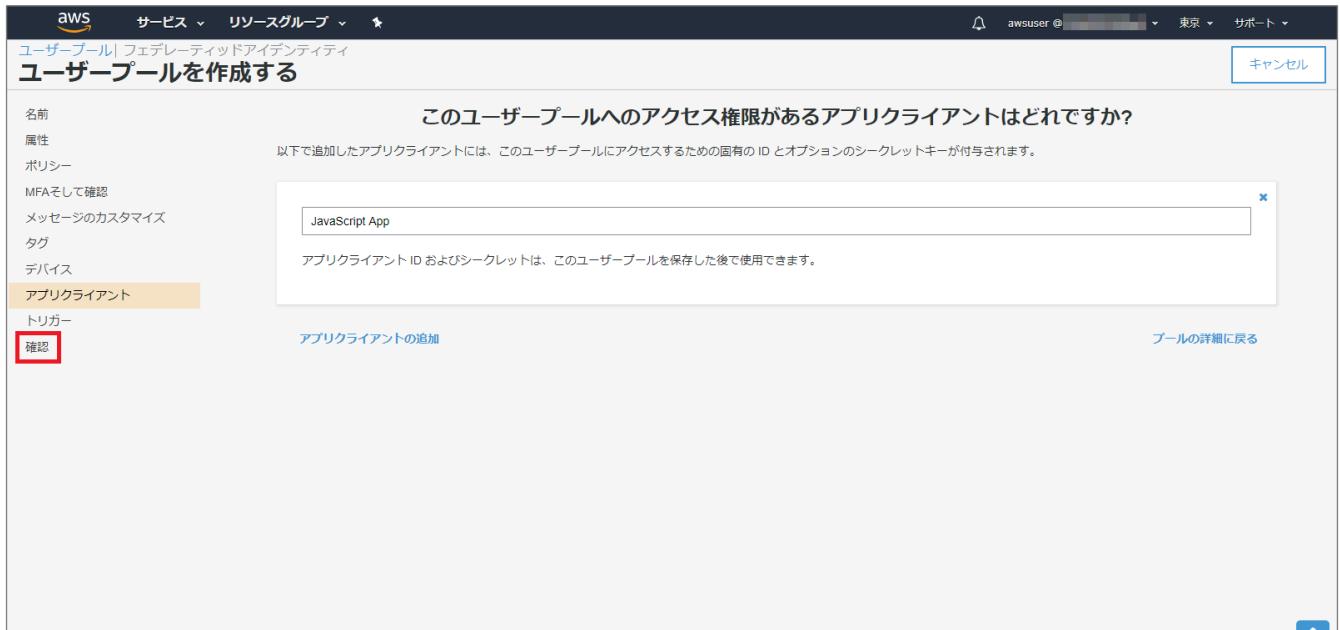
8. [アプリケーション名] 欄に [JavaScript App] と入力します。

[クライアントシークレットを生成] のチェックを外します。

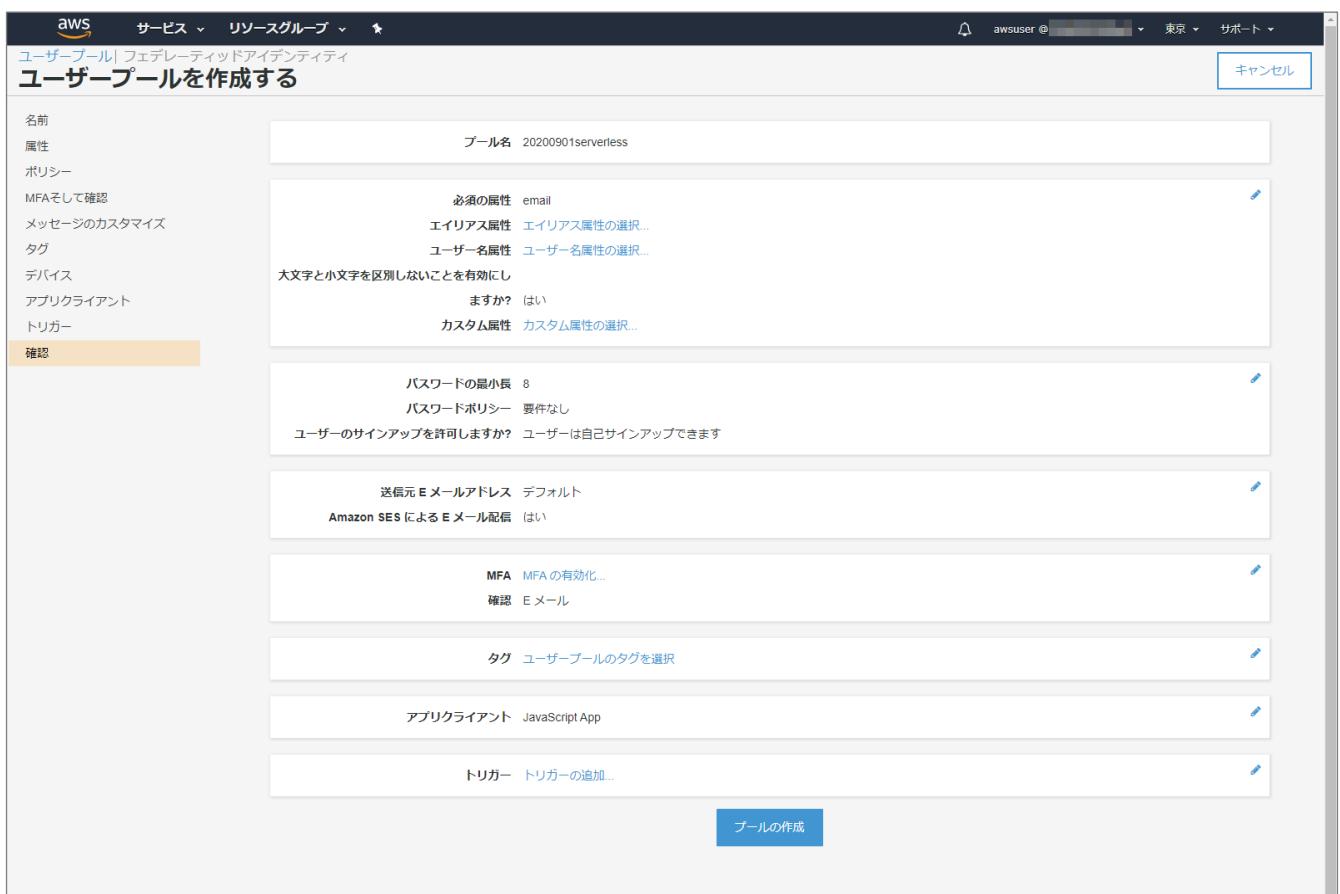


9. [アプリケーションの作成] をクリックします。

10. 画面左側のメニューから [確認] を選択します。



11. 確認画面になりますので、[プールの作成] をクリックします。



12. ユーザープールが作成されました。

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

The screenshot shows the AWS Cognito User Pool creation page. The top navigation bar includes 'aws', 'サービス', 'リソースグループ', and account information 'awsuser @ [REDACTED] 東京 サポート'. The main title is 'ユーザープール| フェデレーティッドアイデンティティ' and the sub-title is '20200901serverless'. A green success message box says 'ユーザープールは正常に作成されました。'. Below it, the '全般設定' section displays the following details:

- プール ID: ap-northeast-1_9OcRWNNmos (highlighted with a red box)
- プール ARN: arn:aws:cognito-idp:ap-northeast-1:[REDACTED]:userpool/ap-northeast-1_9OcRWNNmos
- 推定ユーザー数: 0
- 必須の属性: email
- エイリアス属性: なし
- ユーザー名属性: なし
- 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: 要件なし
- 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

The left sidebar lists various settings like 'ユーザーとグループ', '属性', 'ポリシー', etc.

13. 画面左側のメニューから [アプリクライアント] を選択します。

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

The screenshot shows the '全般設定' section of the User Pool settings. The 'アプリクライアント' tab is selected. A modal window titled 'このユーザープールへのアクセス権限があるアプリクライアントはどれですか?' is open, listing a single client named 'JavaScript App'. The 'アプリクライアント ID' field contains the value '6eq2cucldd6v1nrpedivdgjsf' (highlighted with a red box). Other fields in the modal include '詳細を表示' and '別のアプリクライアントの追加'. The bottom right corner of the modal has a link 'プールの詳細に戻る'. The left sidebar shows other tabs like 'ユーザーとグループ', '属性', 'ポリシー', etc.

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's a navigation bar with 'aws', 'サービス', 'リソースグループ', and other account details. Below the navigation is the Cognito logo and the text 'Amazon Cognito'. A descriptive paragraph explains that Cognito provides user pools and ID pools for sign-up/sign-in and identity providers. Two main buttons are at the bottom: 'User Pool Management' and 'ID Pool Management', with 'ID Pool Management' being highlighted with a red box. Below these buttons are two sections: 'Sign Up and Sign In' (with an icon of two people and a plus sign) and 'User Access to AWS Services' (with an icon of a computer monitor and a lock).

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。

The screenshot shows the 'Create New ID Pool' wizard, Step 1: Create ID Pool. It has two tabs: 'Step 1: Create ID Pool' (selected) and 'Step 2: Set Access Restrictions'. The 'Step 1' tab has a section for 'New ID Pool Name' where '20200901serverless' is entered. Below it is a section for 'Authentication' with a checkbox labeled 'Enable authentication for unauthenticated identities' which is checked. There are also sections for 'AWS Lambda triggers' and 'AWS Lambda roles'. The bottom of the screen shows a note about 'AWS Lambda triggers' and a 'Next Step' button.

15. [認証プロバイダー] をクリックして展開します。

The screenshot shows the 'AWS' navigation bar, user account 'awsuser @ [REDACTED]', and a dropdown for selecting a region. The main title is '使用開始ウィザード' (Get Started Wizard). The current step is '新しい ID プールの作成' (Create a new ID pool). On the left, there are two steps: 'ステップ 1: ID プールを作成する' (Step 1: Create an ID pool) and 'ステップ 2: アクセス権限の設定' (Step 2: Set access permissions). The '認証されていない ID' (Unauthenticated ID) section is expanded, showing a checkbox for enabling it. The '認証フローの設定' (Authentication flow settings) section is also expanded, showing a checkbox for enabling basic (classic) flows. A red box highlights the '認証プロバイダー' (Authentication provider) button under the '認証フローの設定' section. At the bottom right are 'キャンセル' (Cancel) and 'プールの作成' (Create pool) buttons.

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

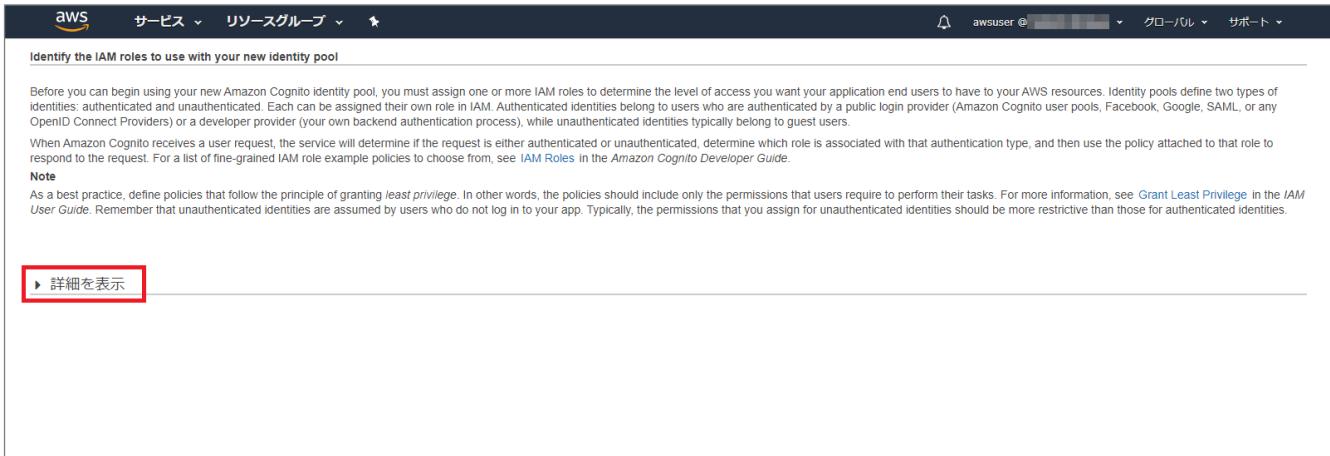
[ユーザープール ID] および [アプリクライアント ID] 欄に、前手順で確認したユーザープールの各 ID を入力します。

The screenshot shows the '認証プロバイダー' (Authentication provider) section. It lists various public providers: Cognito (selected), Amazon, Apple, Facebook, Google+, Twitter / Digits, OpenID, SAML, and Custom. Below this, there's a note about using Cognito ID pools for authentication. The 'ユーザープール ID' (User pool ID) field contains 'ap-northeast-1_9OcRWNmos' and the 'アプリクライアント ID' (App client ID) field contains '6eq2cucldd6v1nrpedivdgjsf'. Both fields have red boxes around them. At the bottom right are 'キャンセル' (Cancel) and 'プールの作成' (Create pool) buttons.

17. [プールの作成] をクリックします。

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

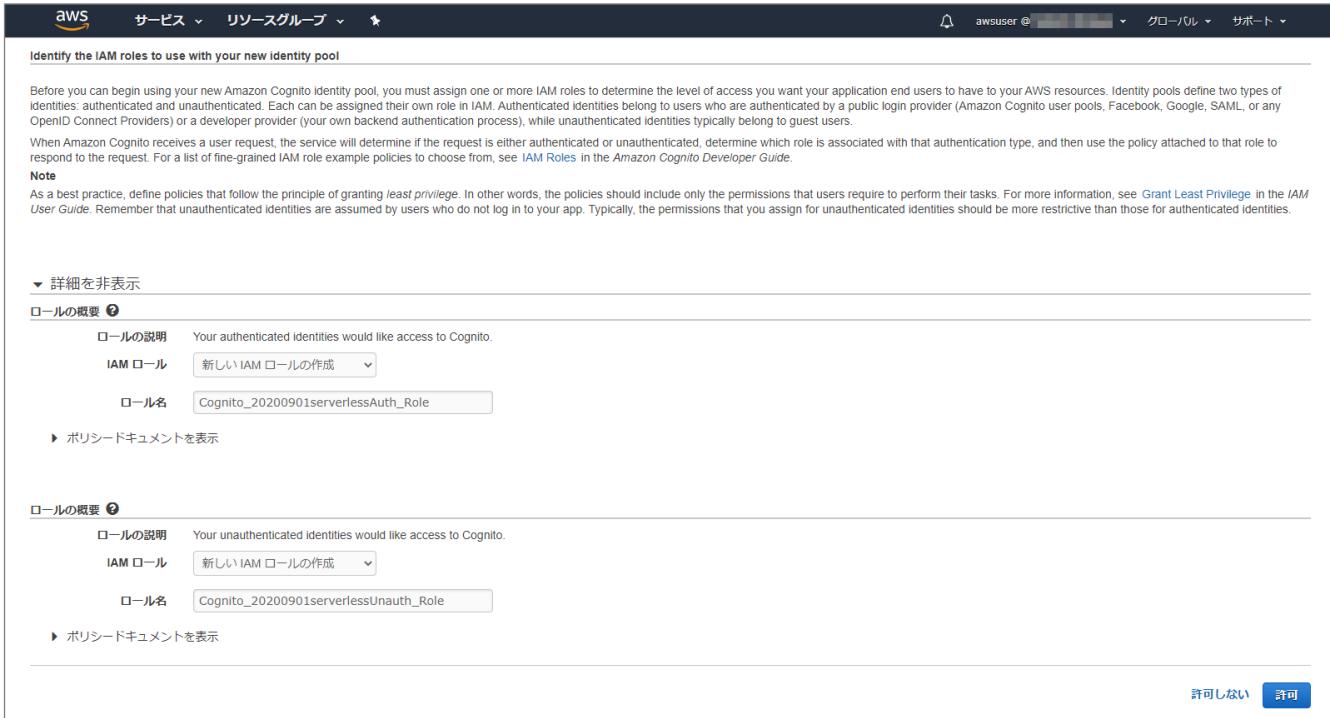
[詳細を表示] をクリックして展開します。



The screenshot shows the AWS Cognito Identity Pool configuration wizard. The current step is "Identify the IAM roles to use with your new identity pool". It contains instructions about assigning IAM roles to determine access levels. A red box highlights the "▶ 詳細を表示" (Show details) button.

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

[許可] をクリックします。



The screenshot shows the IAM Role creation wizard. It displays two roles: "Cognito_20200901serverlessAuth_Role" and "Cognito_20200901serverlessUnauth_Role". The "Cognito_20200901serverlessAuth_Role" section is expanded, showing its role summary. The "Cognito_20200901serverlessUnauth_Role" section is collapsed. At the bottom right, there are "許可しない" and "許可" buttons.

20.ID プールが作成されました。

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

The screenshot shows the AWS Cognito ID Pool sample code page. The left sidebar has links for ID Pool, Dashboard, Sample Code, and Browser. The main content area is titled "Amazon Cognito での作業開始". It shows a dropdown for "Platform" set to "Android". Under "AWS SDK のダウンロード", there is a link to "AWS SDK for Android をダウンロード" and a "Developer Guide". Below that is a section for "AWS 認証情報の取得" containing Java code. A red box highlights the placeholder text "ID プールの ID". The code is as follows:

```
// Amazon Cognito 認証情報プロバイダーを初期化します
CognitoCachingCredentialsProvider credentialsProvider = new CognitoCachingCredentialsProvider(
    applicationContext,
    "ap-northeast-1:835a350a-8d7b-4d3b-83f0-84cf39f67888", // ID プールの ID
    regions.ap_northeast_1 // リージョン
);
```

Below the code, it says "次に、認証情報プロバイダーを初期化します。" with a link to "Cognito ID での作業開始". At the bottom is a "ダッシュボードに移動" button.

4.3.3. Web ブラウザからの動作確認

1. [webpage] フォルダに「4.2. API Gateway を追加して動作確認を行う」で使用したファイルが残っていると思います。[433]のフォルダからファイルを[webpage]の下にコピーします。
2. [webpage] フォルダに以下のファイルがあります。
 - [cognito.js]
 - [config.js]
 - [login.html]
 - [mypage.html]
3. [config.js] をテキストエディタで開きます。
[userPoolId]、[appClientId]、[identityPoolId] の各値を、前節・前々節で確認した [ユーザープール ID]、[アプリクライアント ID]、[ID プール ID] の値にそれぞれ書き換えます。
※ 東京以外のリージョンを利用している場合は、[region] の値も併せて書き換えてください

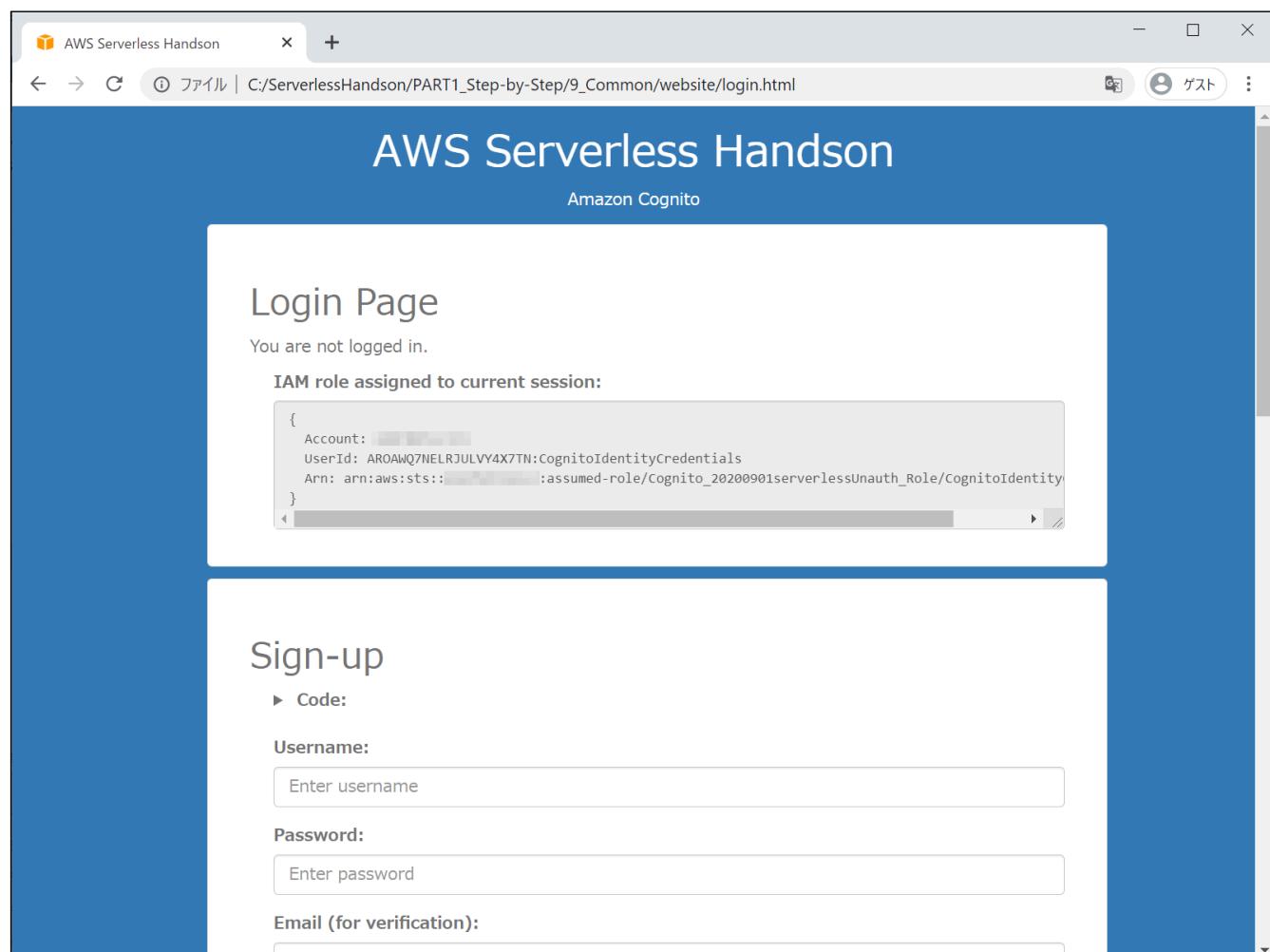
```
const CognitoConfig = {  
    region: 'ap-northeast-1',  
  
    // User Pool  
    userPoolId: 'ap-northeast-1 XXXXXXXX',  
    appClientId: 'XXXXXXXXXXXXXXXXXXXXXX',  
  
    // Federated Identity  
    identityPoolId: 'ap-northeast-1:XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXX',  
}
```

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

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

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:" contains a 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:", "Username" (set to "user1"), "Password" (redacted), "Email (for verification)" (set to "user1@example.com"), and a "Sign-up" button. Below the form is a "Result:" section which is currently empty.

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

Sign-up

▶ Code:

Username:
Enter username

Password:
Enter password

Email (for verification):
Enter email

Sign-up

Result:

```
(Mon Sep 14 2020 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）

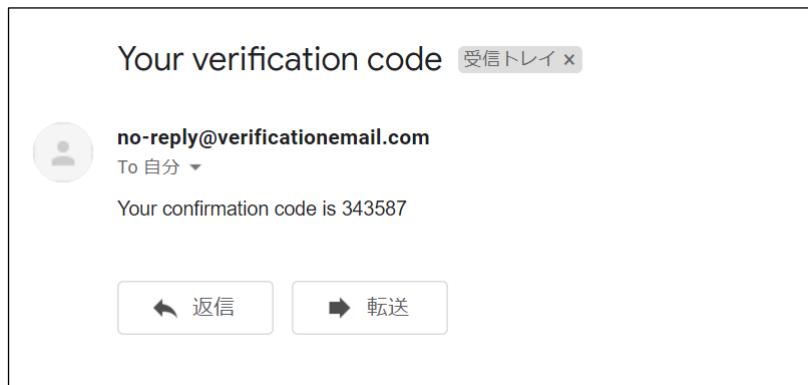
アカウントのステータスが [UNCONFIRMED]（未確認）であることを確認してください。

The screenshot shows the AWS Management Console interface for the Cognito User Pool service. The top navigation bar includes the AWS logo, a 'Services' dropdown, and account information ('awsuser @ [REDACTED] 東京 サポート'). Below the navigation is a breadcrumb trail: 'ユーザーとグループ' > 'フェデレーティッドアイデンティティ' > '20200901serverless'. On the left, a sidebar menu lists various settings like '全般設定', 'ユーザーとグループ' (which is currently selected and highlighted in orange), '属性', 'ポリシー', etc. The main content area has tabs for 'ユーザー' and 'グループ', with 'ユーザー' selected. It features two buttons: 'ユーザーをインポート' and 'ユーザーの作成'. A search bar with 'User name' and a '検索' button is present. Below these are filter options: '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'. A table lists users with the following columns: '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'. A single row for 'user1' is shown, with its 'アカウントのステータス' field explicitly highlighted with a red border. The table data is as follows:

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。

Activation

▶ Code:

Username:
user1

Confirmation Code:
343587

Activate

Result:

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

Activation

▶ Code:

Username:

Confirmation Code:

Activate

Result:

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

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

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

The screenshot shows the AWS Cognito User Pool interface. The left sidebar has a 'User and Groups' tab selected. The main area shows a table with one user entry:

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

The 'CONFIRMED' status in the third column is highlighted with a red box.

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

ユーザー名とパスワードを入力して、[Sign-in] をクリックします。

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

15. サインインが成功すると、[My Page] に遷移します。

The screenshot shows a web browser window titled 'AWS Serverless Handson'. The address bar indicates the page is at 'C:/ServerlessHandson/PART1_Step-by-Step/9_Common/website/mypage.html'. The main content area displays two sections:

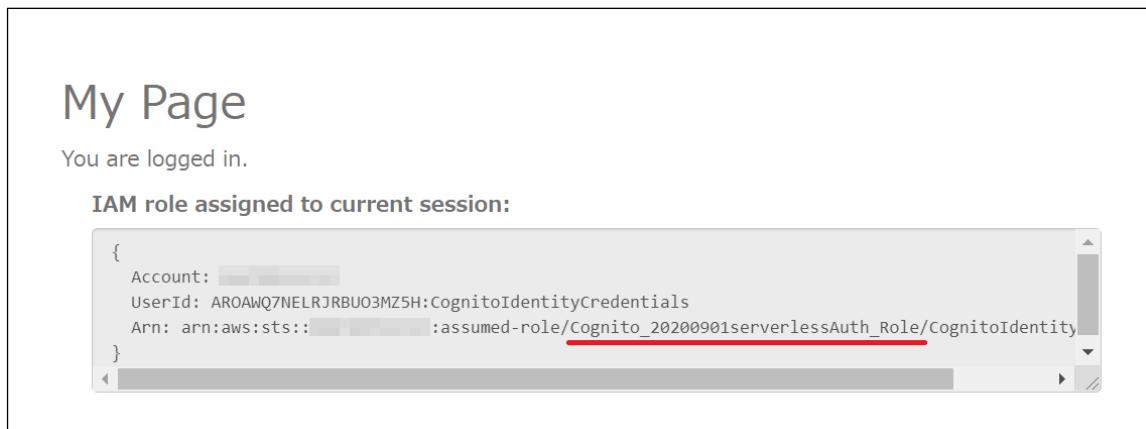
- My Page**: Shows the message "You are logged in." and a JSON representation of the "IAM role assigned to current session":

```
{  Account: "████████",  UserId: "AROAWQ7NELRJRB0U3MZ5H:CognitoIdentityCredentials",  Arn: "arn:aws:sts::████████:assumed-role/Cognito_20200901serverlessAuth_Role/CognitoIdentity"}
```
- About current user**: Contains the heading "How to get current CognitoUser" and a code snippet:

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

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。

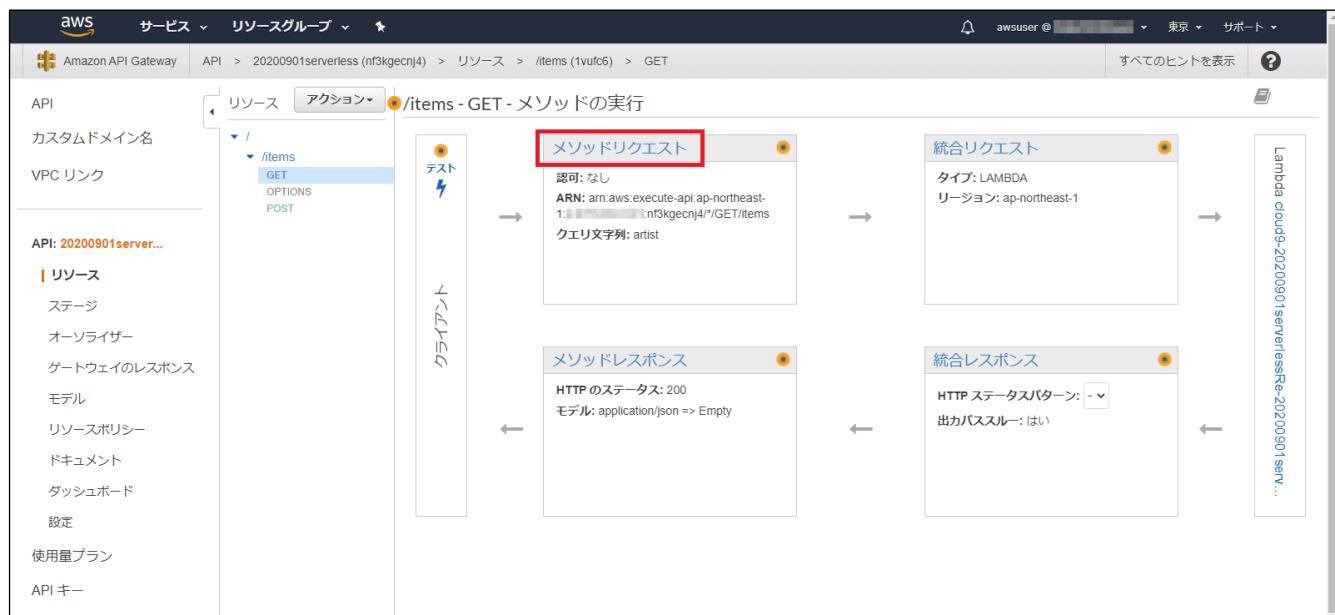


4.4. Cognito による認証と API Gateway を組み合わせる

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 for resources: '/ > /items > GET'. A red box highlights the '認可' (Authorization) dropdown in the '設定' (Settings) section, 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 shows the same API configuration screen as the previous one, but with changes made to the authorization settings. The '認可' (Authorization) dropdown has been changed to 'AWS_IAM', and a red box highlights the '確認' (Confirm) button next to it. The rest of the settings remain the same as in the first screenshot.

4. 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。

The screenshot shows the AWS API Gateway console. The navigation path is: API > 20200901serverless (nf3kgecnj4) > リソース > /items > GET. The left sidebar shows the API ID: 20200901server... and a list of resources like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, ダッシュボード, 設定, 使用量プラン, APIキー, and クライアント証明書. The main panel shows the configuration for the GET method on the /items resource. The title is 'リソース' and the action is 'アクション' (GET). A red box highlights the 'メソッドの実行' button. Below it, the documentation states: 'このメソッドの認可設定と、受信可能なパラメータに関する情報を指定します。' Under '設定', the '認可' section is set to 'AWS_IAM'. Other sections include 'リクエストの検証' (なし), 'APIキーの必要性' (false), 'URL クエリ文字列パラメータ', 'HTTP リクエストヘッダー', 'リクエスト本文', and 'SDK 設定'.

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

The screenshot shows the AWS API Gateway console. The URL in the address bar is `Amazon API Gateway > API > 20200901serverless (nf3kgecnj4) > リソース > /items (1vufc6) > GET`. On the left sidebar, under the 'API' section, 'リソース' is selected. In the main area, the path `/items` is expanded, and the 'GET' method is selected. A 'クライアント' (Client) section shows a 'メソッドリクエスト' (Method Request) box. Inside this box, the 'ARN' field is highlighted with a red box, showing the value `arn:aws:execute-api:ap-northeast-1:1:...:nf3kgecnj4/*/GET/items`. Below the ARN, it says 'クエリ文字列: artist'. To the right of the client section is a '統合リクエスト' (Integrated Request) box, which contains the text 'タイプ: LAMBDA' and 'リージョン: ap-northeast-1'. At the bottom of the client section is a 'メソッドレスポンス' (Method Response) box, which shows 'HTTP のステータス: 200' and 'モデル: application/json => Empty'. To the right of the client section is a '統合レスポンス' (Integrated Response) box, which shows 'HTTP ステータスパターン: -' and '出力バスルート: はい'. The right side of the screen has a vertical scroll bar.

6. POST メソッド実行の設定画面を開きます。

[メソッドリクエスト] をクリックします。

This screenshot is similar to the previous one but for the POST method. The URL in the address bar is `Amazon API Gateway > API > 20200901serverless (nf3kgecnj4) > リソース > /items (1vufc6) > POST`. The 'リソース' section is selected on the left. The path `/items` is expanded, and the 'POST' method is selected. In the 'クライアント' section, the 'メソッドリクエスト' box is highlighted with a red box. Inside, the 'ARN' field shows the value `arn:aws:execute-api:ap-northeast-1:1:...:nf3kgecnj4/*/POST/items`. The 'クエリ文字列' field is empty. To the right of the client section is a '統合リクエスト' box with 'タイプ: LAMBDA' and 'リージョン: ap-northeast-1'. Below the client section is a 'メソッドレスポンス' box with 'HTTP のステータス: 200' and 'モデル: application/json => Empty'. To the right of the client section is a '統合レスポンス' box with 'HTTP ステータスパターン: -' and '出力バスルート: はい'. The right side of the screen has a vertical scroll bar.

7. 【認可】の右側にある鉛筆ボタンをクリックします。

The screenshot shows the AWS API Gateway console. On the left, there's a sidebar with various options like 'API', 'リソース', 'ステージ', etc. The main area shows a tree structure for an API named '20200901serverless'. Under the 'POST' method for the '/items' resource, the 'Authorization' section is highlighted. A red box surrounds the 'Authorization' dropdown, which is currently set to 'なし' (None). Below it, other settings like 'Request authentication' and 'API key requirement' are shown.

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. A red box highlights the dropdown, which now shows 'AWS_IAM' instead of 'なし'. The rest of the interface remains the same, with the 'Request authentication' and 'API key requirement' fields visible below.

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。

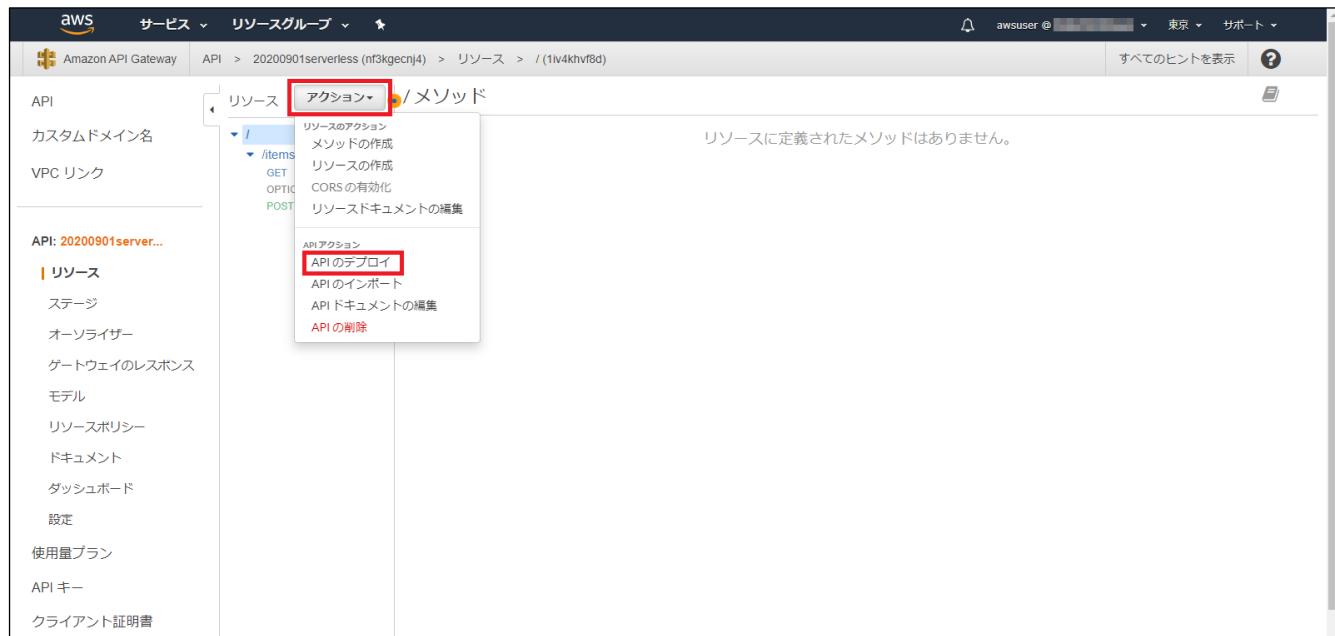
The screenshot shows the AWS API Gateway console. The URL in the address bar is `API > 20200901serverless (nf3kgecnj4) > リソース > /items (1vufc6) > POST`. On the left sidebar, under the 'API' section, 'API: 20200901server...' is selected. In the main pane, the 'アクション' dropdown is open, and the 'POST' option is highlighted. A red box highlights the '← メソッドの実行' button, which is located next to the 'リソース' tab. The right side of the screen displays configuration settings for the POST method, including IAM authentication and 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。

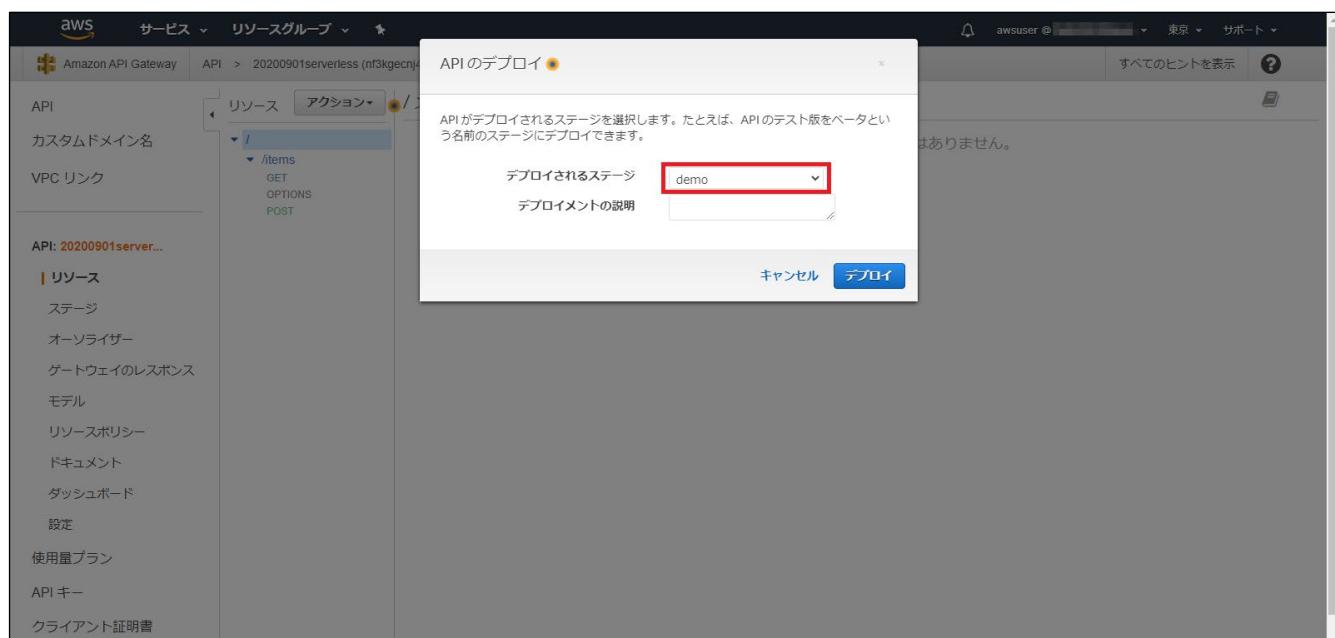
The screenshot shows the AWS API Gateway console after executing the POST method. The URL in the address bar is the same as the previous screenshot. The 'アクション' dropdown now shows the result of the execution: '/items - POST - メソッドの実行'. A red box highlights the 'ARN: arn:aws:execute-api:ap-northeast-1:1:nf3kgecnj4:POST/items' value, which is the Lambda function ARN. The rest of the interface remains similar to the previous screenshot, showing the configuration tabs and settings.

11. リソースのツリー階層から [/] をクリックして選択します。

[アクション] プルダウンから [API のデプロイ] を選択します。



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



13. デプロイが行われました。

The screenshot shows the AWS API Gateway Stage Editor for the 'demo' stage of the '20200901serverless' API. The URL of the stage is <https://nf3kgecnj4.execute-api.ap-northeast-1.amazonaws.com/demo>. The 'Setting' tab is selected. Under the 'Throttling' section, the 'Rate' is set to 10000 requests/second and the 'Burst' is set to 5000 requests. There is also a note about WAF settings.

API: 20200901serverless

ステージ: demo

URL の呼び出し: https://nf3kgecnj4.execute-api.ap-northeast-1.amazonaws.com/demo

設定 ログトレース ステージ変数 SDK の生成 エクスポート デプロイ履歴 ドキュメント履歴 Canary

キャッシュ設定

API キャッシュ有効化

デフォルトのメソッドスロットリング

このステージのメソッド用のデフォルトのスロットリングレベルを選択します。このステージの各メソッドでは、これらのレートおよびバーストの設定を優先します。現在のアカウントレベルのスロットリングレートは、1秒あたりのリクエスト数が 10000 で、バーストのリクエスト数は 5000 です。API Gateway のスロットリングの詳細

スロットリングの有効化

レート: 10000 リクエスト数/秒

バースト: 5000 リクエスト数

ウェブアプリケーションファイアウォール (WAF) 詳細はこちら。

このステージに適用されるウェブ ACL を選択します。

ウェブ ACL: なし

4.4.2. IAM ロールにポリシーを追加

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

[ポリシーの作成] をクリックします。

The screenshot shows the AWS Management Console with the Identity and Access Management (IAM) service selected. In the left sidebar, 'ポリシー' (Policies) is expanded. At the top, there is a search bar and a button labeled 'ポリシーの作成' (Create Policy). Below the search bar, a table lists various policies with columns for 'ポリシー名' (Policy Name), 'タイプ' (Type), '次として使用' (Used as), and '説明' (Description). One policy, '20200901serverlessLambdaPolicy', is highlighted.

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: Service Selection. The title is 'ポリシーの作成'. It includes a note about defining permissions using the visual editor or JSON. There are tabs for 'ビジュアルエディタ' (Visual Editor) and 'JSON'. Below the tabs, there are buttons for 'すべて展開' (Expand All) and 'すべて折りたたむ' (Collapse All). A section titled 'サービスの選択' (Select Service) contains a dropdown menu with 'サービス' (Service) selected, and a sub-menu 'サービスの選択' (Select Service) with an arrow pointing to it. Buttons for 'クローン' (Clone) and '削除' (Delete) are at the top right of this section.

3. 検索欄に [executeapi] と入力します。

表示された [ExecuteAPI] をクリックします

The screenshot shows the 'Create Policy' wizard, Step 1: Service Selection, with the search term 'executeapi' entered in the search bar. The results list shows 'ExecuteAPI' as the first result, which is highlighted with a red box. Other results include 'サービス' (Service) and '閉じる' (Close).

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。

The screenshot shows the AWS Lambda function configuration interface. In the 'Actions' section, there is a checkbox labeled 'Invoke' which is checked and highlighted with a red box. Other options like 'InvalidateCache' and 'ManageConnections' are also listed.

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。

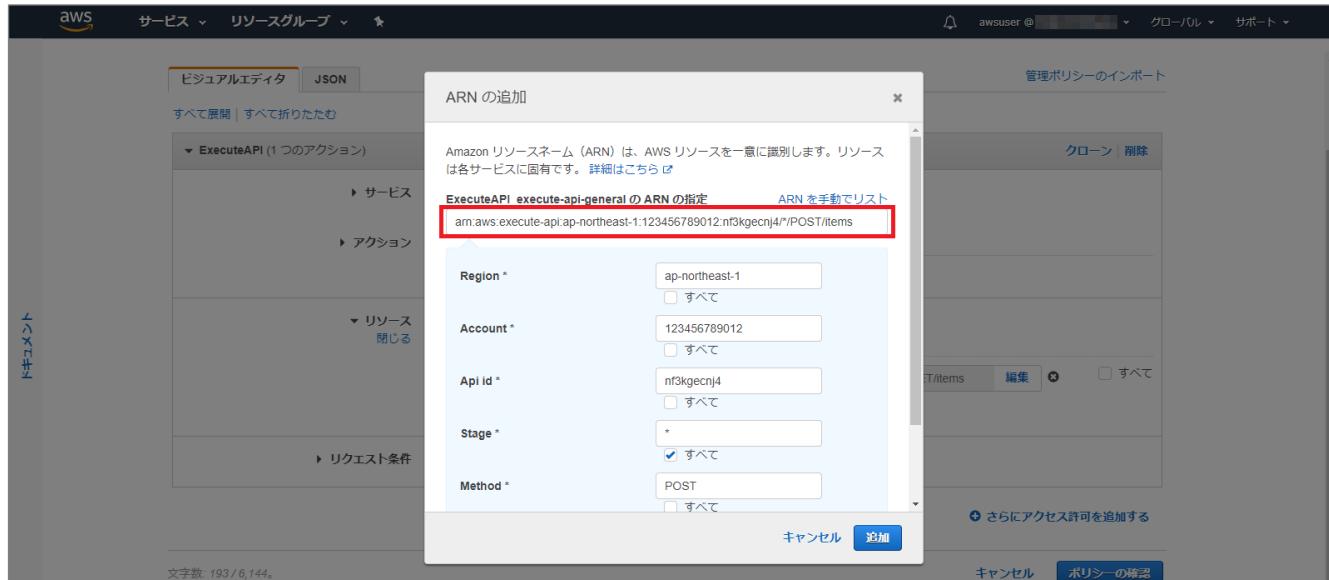
The screenshot shows the AWS Lambda function configuration interface with a modal dialog titled 'ARN の追加'. In the 'ARN を手動でリスト' field, the ARN 'arn:aws:execute-api:ap-northeast-1:123456789012:nf3kgcncj4/*/*GET/items' is entered and highlighted with a red box. The dialog also contains fields for Region, Account, API ID, Stage, and Method, all of which are set to their respective values.

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。

The screenshot shows the AWS Lambda function configuration interface with the 'ARN の追加' dialog still open. The 'ARN の追加' button at the bottom right of the dialog is highlighted with a red box.

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。

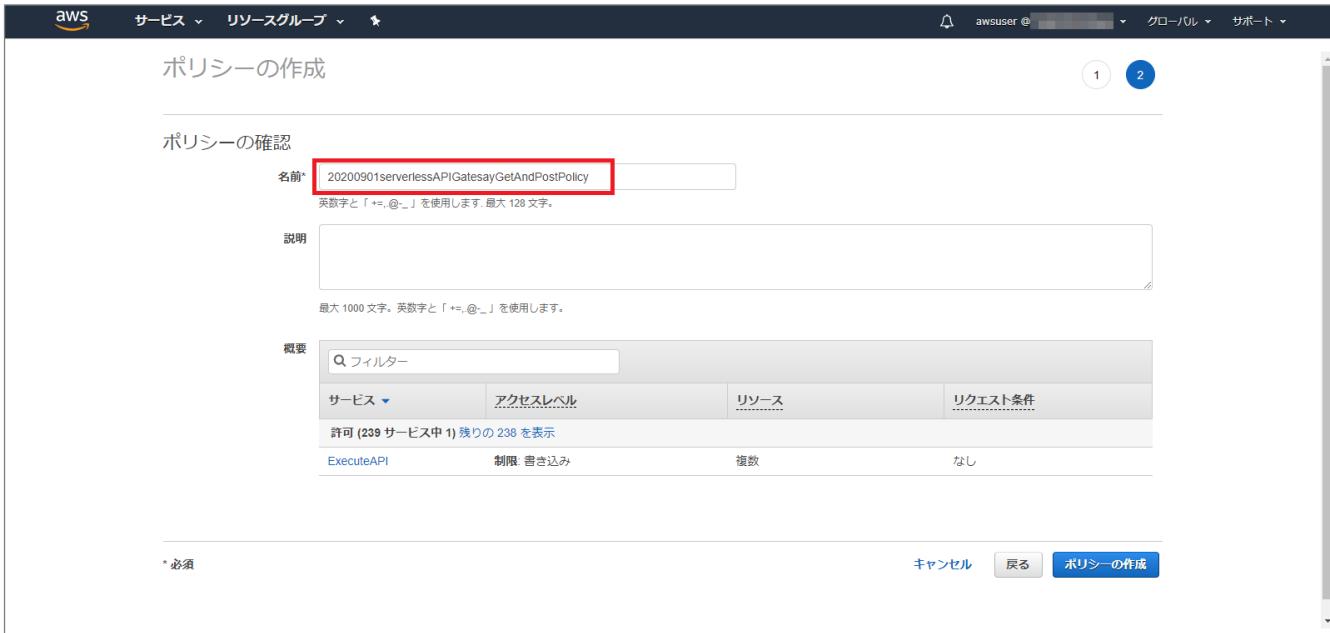


8. POST メソッド呼び出しの ARN が追加されたことを確認します。



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 'Name' field contains the value '20200901serverlessAPIGatesayGetAndPostPolicy'. The 'Description' field is empty. The 'Summary' section shows the following details:

サービス	アクセスレベル	リソース	リクエスト条件
ExecuteAPI	制限: 書込み	複数	なし

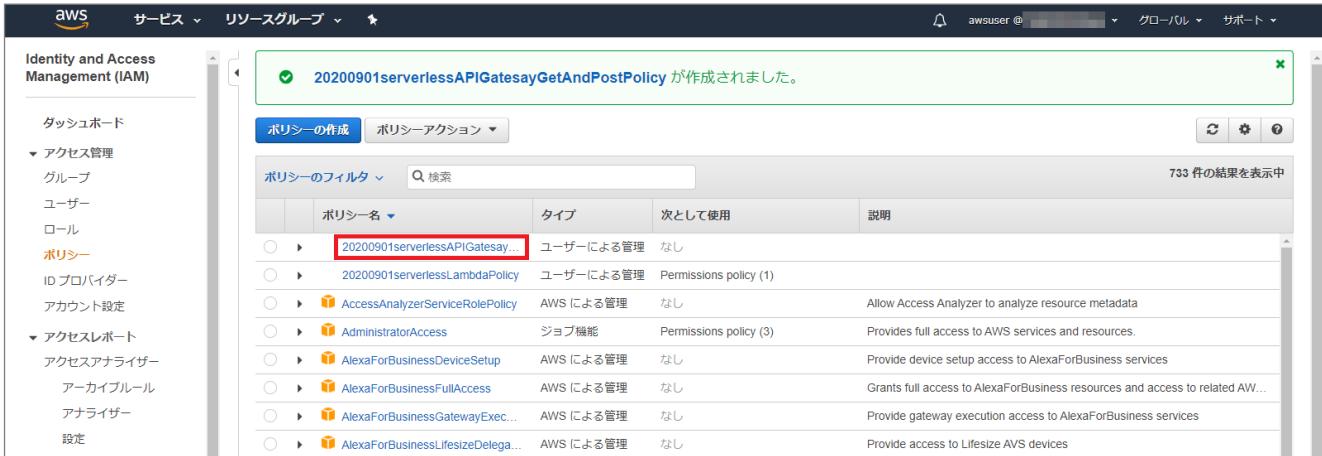
At the bottom, there are buttons for 'キャンセル', '戻る', and a blue 'ポリシーの作成' button.

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 'Policies' list in the IAM console. A success message at the top states '20200901serverlessAPIGatesayGetAndPostPolicy が作成されました.' The table lists policies, with the newly created one highlighted in red:

ポリシー名	タイプ	次として使用	説明
20200901serverlessAPIGatesay...	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	ジョブ機能	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
AlexaForBusinessLifesizeDelega...	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。

The screenshot shows the AWS Identity and Access Management (IAM) Policies page. The top navigation bar includes 'aws' logo, 'サービス' (Services), 'リソースグループ' (Resource Groups), a bell icon for notifications, the user 'awsuser @...', 'グローバル' (Global), and 'サポート' (Support). On the left, a sidebar menu is open under 'Identity and Access Management (IAM)', showing 'ダッシュボード', 'アクセス管理' (Access Management) expanded to show 'グループ', 'ユーザー', 'ロール', 'ポリシー' (Policy) selected, 'ID プロバイダー' (ID Provider), and 'AWS 証明書'. The main content area has tabs 'ポリシーの作成' (Create Policy) and 'ポリシーアクション' (Policy Actions), with 'ポリシーの作成' highlighted by a blue box. Below is a search bar and a filter dropdown 'ポリシーのフィルタ' (Filter Policies). A table lists 732 results, with columns: 'ポリシー名' (Policy Name), 'タイプ' (Type), '次として使用' (Use as Next), and '説明' (Description). The table rows include:

	ポリシー名	タイプ	次として使用	説明
<input type="radio"/>	20200901serverlessLambdaPolicy	ユーザーによる管理	Permissions policy (1)	
<input type="radio"/>	AccessAnalyzerServiceRolePolicy	AWS による管理	なし	Allow Access Analyzer to analyze resource metadata
<input type="radio"/>	AdministratorAccess	ジョブ機能	Permissions policy (3)	Provides full access to AWS services and resources.
<input type="radio"/>	AlexaForBusinessDeviceSetup	AWS による管理	なし	Provide device setup access to AlexaForBusiness services
<input type="radio"/>	AlexaForBusinessFullAccess	AWS による管理	なし	Grants full access to AlexaForBusiness resources and access to related AWS ...

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

ポリシーネーム	タイプ	次として使用	説明
20200901serverlessAPIGatesay...	ユーザーによる管理	なし	
20200901serverlessAPIGatesay...	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	ジョブ機能	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
AlexaForBusinessLifesizeDelega...	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...
AlexaForBusinessPolyDelegated...	AWS による管理	なし	Provide access to Poly AVS devices
AlexaForBusinessReadOnlyAccess	AWS による管理	なし	Provide read only access to AlexaForBusiness services
AmazonAPIGatewayAdministrator	AWS による管理	なし	Provides full access to create/edit/delete APIs in Amazon API Gateway via th...
AmazonAPIGatewayInvokeFullA...	AWS による管理	なし	Provides full access to invoke APIs in Amazon API Gateway

20. IAM ポリシーの定義内容を JSON 形式で表示します。

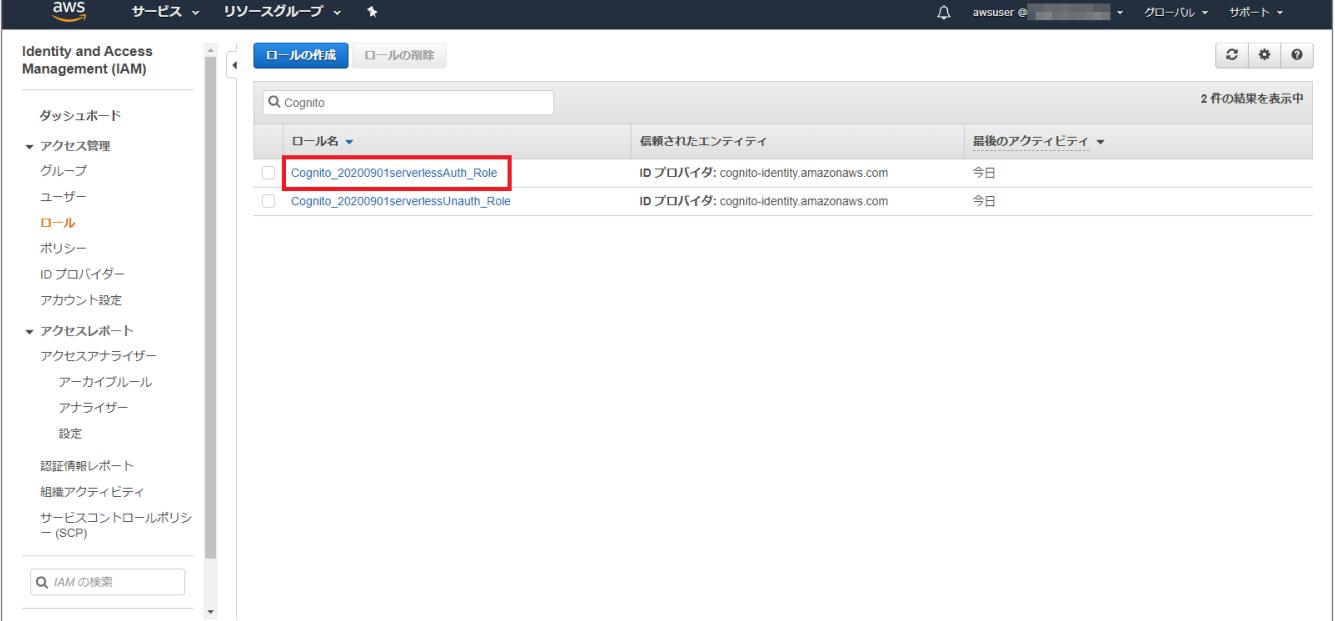
以下のような定義内容となっていることを確認します

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

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

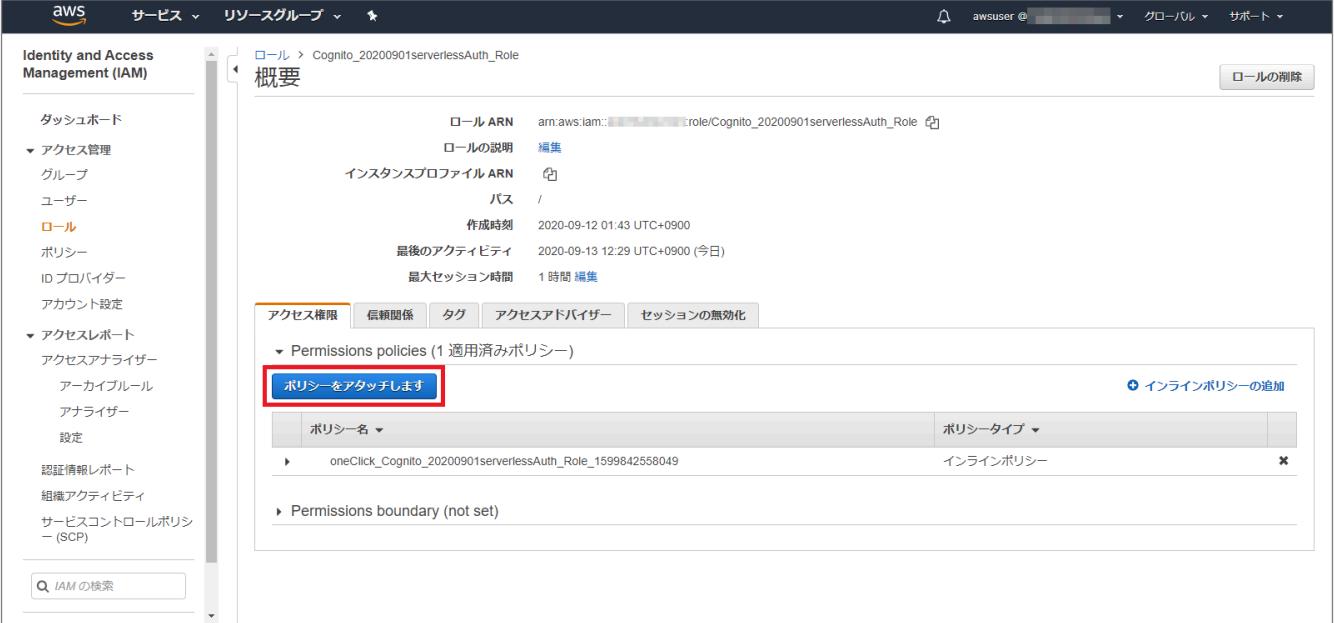
検索ボックスに [Cognito] と入力して、Cognito ID プールに紐付くロールを表示します。

[**Cognito_YYYYMMDDserverlessAuth_Role**] (認証された ID 用のロール) をクリックします。



The screenshot shows the AWS IAM Roles list page. The left sidebar navigation includes 'Identity and Access Management (IAM)' under 'サービス' (Services). The main content area has a search bar at the top with 'Cognito' typed in. Below it is a table with two rows. The first row, 'Cognito_20200901serverlessAuth_Role', is highlighted with a red box. The second row is 'Cognito_20200901serverlessUnauth_Role'. The table columns are 'ロール名' (Role Name), '信頼されたエンティティ' (Trusted Entity), and '最後のアクティビティ' (Last Activity).

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 IAM Role details page for 'Cognito_20200901serverlessAuth_Role'. The left sidebar navigation is identical to the previous screenshot. The main content area shows the role's ARN, description, and creation date. Below this, the 'Permissions policies' section is expanded, showing one policy attached: 'oneClick_Cognito_20200901serverlessAuth_Role_1599842558049'. At the bottom of this section is a blue 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 Policies list page. A search bar at the top right contains the text "Cognito_20200901serverlessAuth_Role にアクセス権限を追加する". Below the search bar 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, "20200901serverlessAPIGatesayGetAndPostPolicy", has a checked checkbox in the "タイプ" column and is highlighted with a red border. Other rows include "20200901serverlessAPIGatesayGetOnlyPolicy", "20200901serverlessLambdaPolicy", and several Alexa-related 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 page for "Cognito_20200901serverlessAuth_Role". The left sidebar shows navigation options like "Identity and Access Management (IAM)", "ロール", and "ポリシー". The main area displays the role's ARN, description, and creation date. A summary message states "Cognito_20200901serverlessAuth_Role にポリシー 20200901serverlessAPIGatesayGetAndPostPolicy がアタッチされました。". Below this, the "Permissions policies" section lists the attached policy "20200901serverlessAPIGatesayGetAndPostPolicy".

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. On the left, there's a navigation sidebar with options like Dashboard, Groups, Users, Roles, Policies, and more. The 'Roles' option is currently selected. The main content area displays a list of roles. A search bar at the top says '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

The second row, 'Cognito_20200901serverlessUnauth_Role', has a red box around it, indicating it is the target role.

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 identical to the previous screenshot. The main area shows the role's ARN, description, and creation details. Below this, there's a tabbed section for 'Permissions policies'. The 'Attach policy' button is highlighted with a red box. The policy list shows one inline policy named 'oneClick_Cognito_20200901serverlessUnauth_Role_1599842558050'.

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

The screenshot shows the AWS IAM Policies list. A search bar at the top right contains the text 'Cognito_20200901serverlessUnauth_Role' and a note below it says 'Cognito_20200901serverlessUnauth_Role にアクセス権限を追加する'. Below the search bar is a 'ポリシーの作成' button. The main table lists policies with columns for 'ポリシー名', 'タイプ', and '次として使用'. One policy, '20200901serverlessAPIGatesayGetOnlyPolicy', is selected and highlighted with a red box around its checkbox and row. This policy is described as a 'Permissions policy (1)'.

ポリシー名	タイプ	次として使用
20200901serverlessAPIGatesayGetAndPostPolicy	ユーザーによる管理	なし
20200901serverlessAPIGatesayGetOnlyPolicy	ユーザーによる管理	なし
20200901serverlessLambdaPolicy	ユーザーによる管理	Permissions policy (1)
AdministratorAccess	ジョブ機能	Permissions policy (3)
AlexaForBusinessDeviceSetup	AWS による管理	なし
AlexaForBusinessFullAccess	AWS による管理	なし
AlexaForBusinessGatewayExecution	AWS による管理	なし
AlexaForBusinessLifesizeDelegatedAccessPolicy	AWS による管理	なし
AlexaForBusinessPolyDelegatedAccessPolicy	AWS による管理	なし
AlexaForBusinessReadOnlyAccess	AWS による管理	なし
AmazonAPIGatewayAdministrator	AWS による管理	なし

29. [ポリシーのアタッチ] をクリックします。

30. ポリシーがアタッチされたことを確認します。

The screenshot shows the AWS IAM Role Overview page for 'Cognito_20200901serverlessUnauth_Role'. The left sidebar shows navigation options like 'Identity and Access Management (IAM)', 'ロール', and 'ポリシー'. The main area displays role details: 'Role ARN' (arn:aws:iam:::role/Cognito_20200901serverlessUnauth_Role), 'Role Description' (福集), 'Instance Profile ARN' (None), 'Path' (/), 'Created At' (2020-09-12 01:43 UTC+0900), and 'Last Activity' (2020-09-13 12:37 UTC+0900). Below this, the 'Permissions' tab is selected, showing two attached policies: '20200901serverlessAPIGatesayGetOnlyPolicy' (Management Policy) and 'oneClick_Cognito_20200901serverlessUnauth_Role_1599842558050' (Inline Policy). A note at the top right says 'Cognito_20200901serverlessUnauth_Role にポリシー 20200901serverlessAPIGatesayGetOnlyPolicy がアタッチされました。' (The policy 20200901serverlessAPIGatesayGetOnlyPolicy has been attached to Cognito_20200901serverlessUnauth_Role).

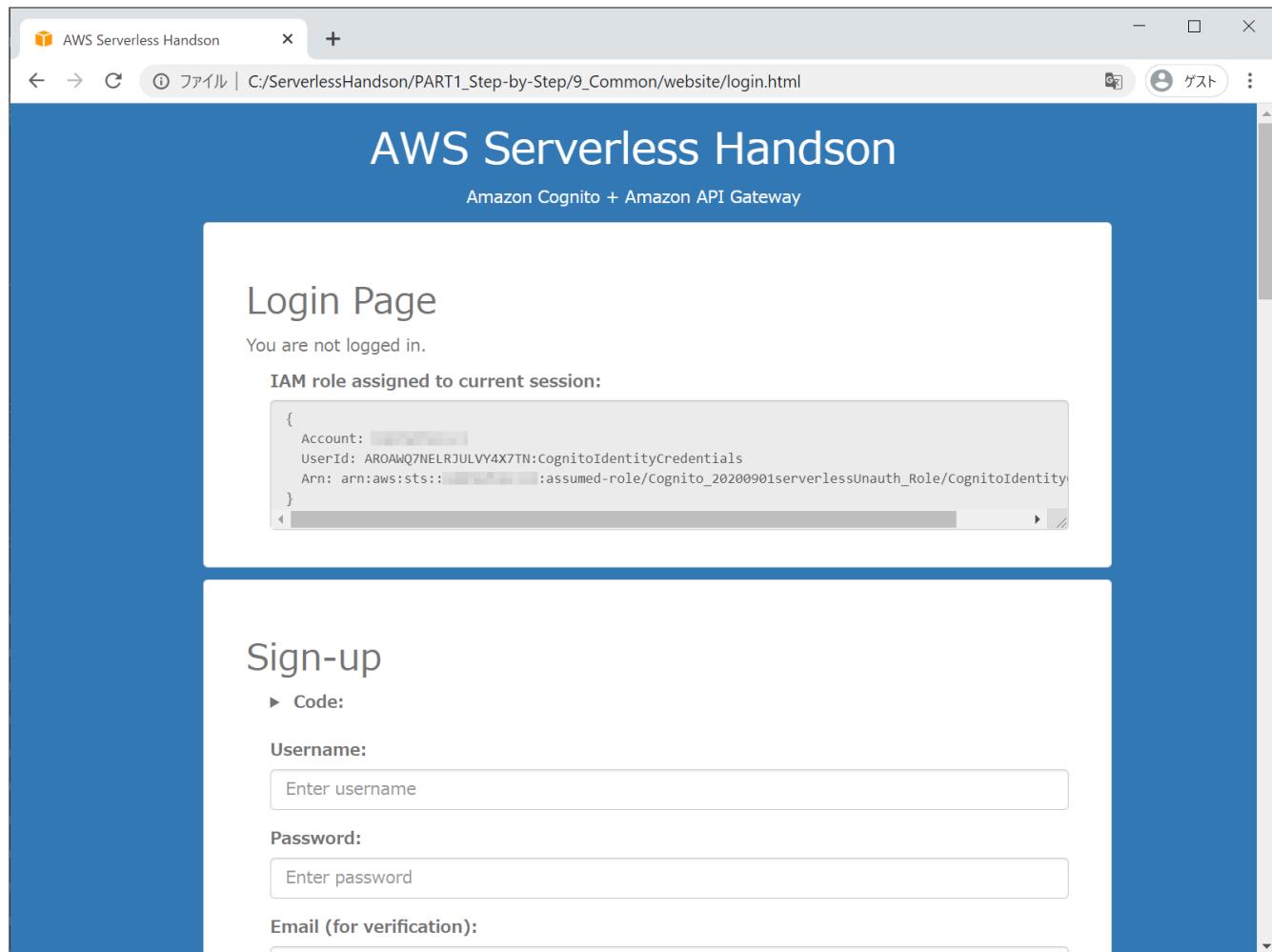
4.4.3. Web ブラウザからの動作確認

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

```
[website]
├── [img] ... アイコン等の画像ファイル
├── [lib] ... 各種ライブラリ/SDK (必要なものを格納済み、ただし apigClient.js を除く)
│   ├── [apigateway]
│   │   ├── [lib]
│   │   │   └── apigClient.js ... API Gateway アクセスclient のクラス定義
│   │   ├── [awssdk]
│   │   └── [cognito]
├── [style] ... CSS ファイル
├── apigateway.js ... API Gateway 処理の JavaScript コード (Cognito 認証処理を追加)
├── cognito.js ... Cognito 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:

Michael Jackson

Run

Status Code:

200

Response Data:

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

6. POST メソッドの実行を試みます。

権限が無いため、ステータスコード [403] が返ってきます。

POST method

▶ Code:

Artist:

Mariah Carey

Title:

All I Want for Christmas Is You

Run

Status Code:

403

Response Data:

```
{  
  "message": "User: arn:aws:sts::██████████:assumed-role/Cognito_20200901serverlessUnauth_Role/CognitoIdentityCredentials is not authorized to perform: execute-api:Invoke on resource: arn:aws:execute-api:ap-northeast-1:██████████:nf3kgecnj4/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/DI 機能や、マネージドなアプリケーションホスティング機能を提供

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

5.1. Amplify を使用する準備を行う

5.1.1. Cloud9 環境のディスク容量拡張

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

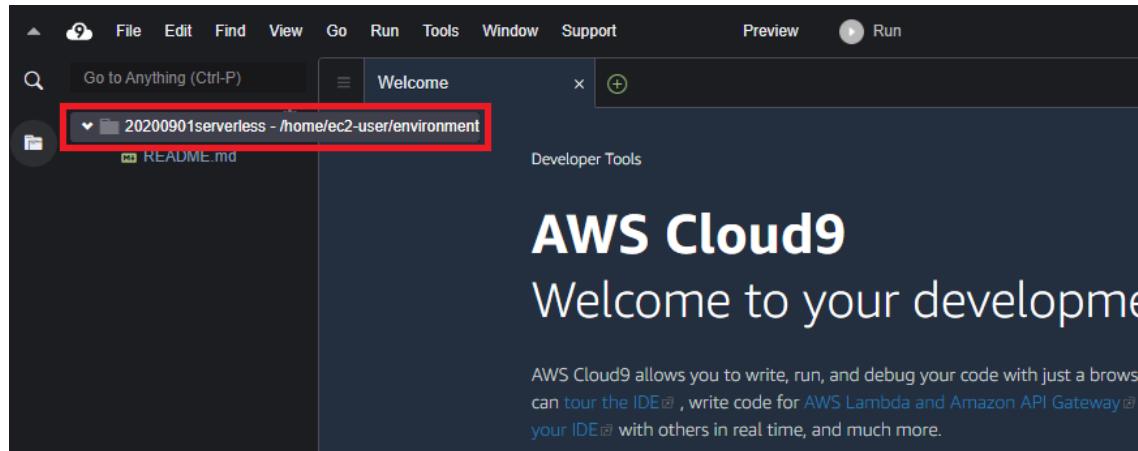
これから実施するラボ 2 ではディスク容量が不足する可能性がありますので、予めディスク容量を拡張しておきます。

1. ハンズオン資料のフォルダに、以下のファイルがあります。

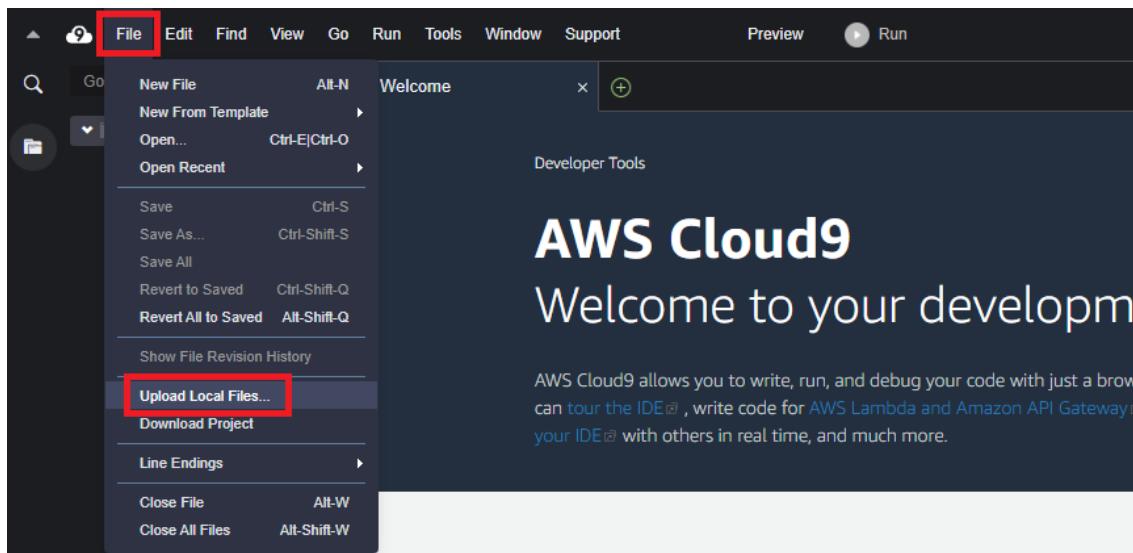
- [resize.sh]

このファイルを Cloud9 上にアップロードします。

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

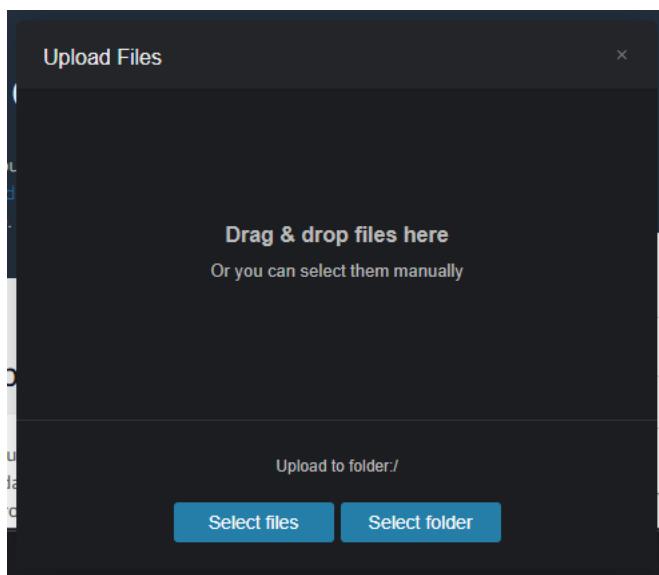


2. 画面上部のメニューから [File]-[Upload Local Files...] を選択します。



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

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



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

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

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

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

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

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

5.1.2. 前提環境の確認

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

画面が狭い場合は、境界線をドラッグして画面を大きくして頂いて構いません。

<図>

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

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

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

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

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

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

5.1.3. Amplify CLI のインストール

1. Amplify CLI は「npm」を使用してインストールします。

以下の通りコマンドを実行します。

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

2. Amplify CLI のインストールが行われます。

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

3. Amplify CLI が実行できることを確認します。

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

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

5.1.4. AWS 認証情報の設定

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

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

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

1. 以下の通りコマンドを実行します。

```
$ aws configure
```

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

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

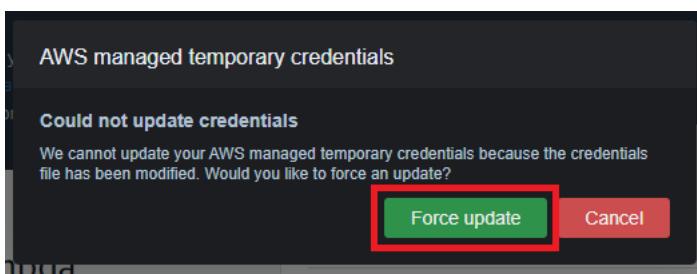
3. 「Default output format」に対して **[json]** と入力して Enter キーを押します。

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

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

このタイミングで以下のようなダイアログが表示される場合があります。

その際は **[Force update]** をクリックしてください。



註) PC や Mac で Amplify CLI を利用する場合は、AWS CLI のプロファイルを指定する方法の他

に、IAM ユーザーのアクセスキーを Amplify CLI に直接設定する方法もあります。

詳しい手順は下記リンク先を参照してください。

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

5.2. REST API を使った Web アプリケーションを作成する

5.2.1. React アプリケーションの作成

1. 以下のコマンドを実行します。 (YYYYMMDD は本日の日付)

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

2. 作成したプロジェクトのディレクトリに移動します。

```
$ cd YYYYMMDDamplify
```

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

以下のコマンドを実行してアプリケーションを起動します。

```
$ npm start
```

4. アプリケーションが起動すると、以下のように表示されます。

```
Compiled successfully!

You can now view 20200901amplify in the browser.

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

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

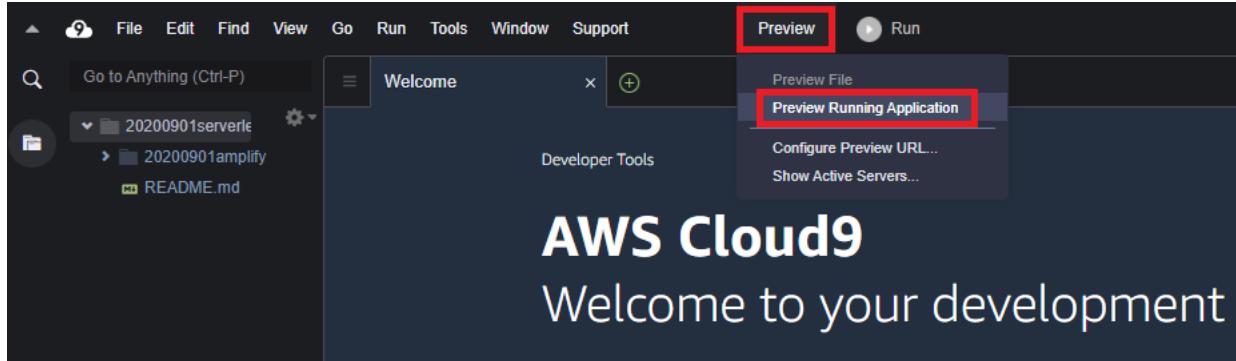
5. 起動したアプリケーションに接続を行ってみます。

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

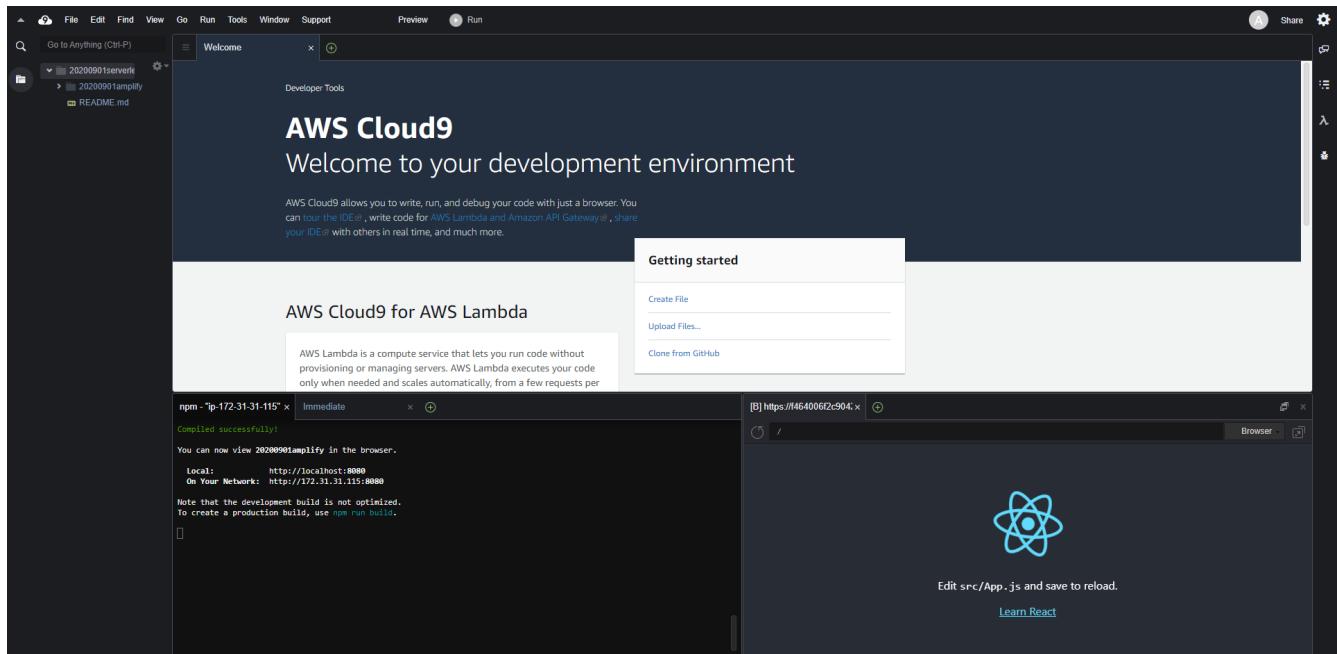
「<http://localhost:8080>」と入力して接続を試みても、接続することはできません。

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

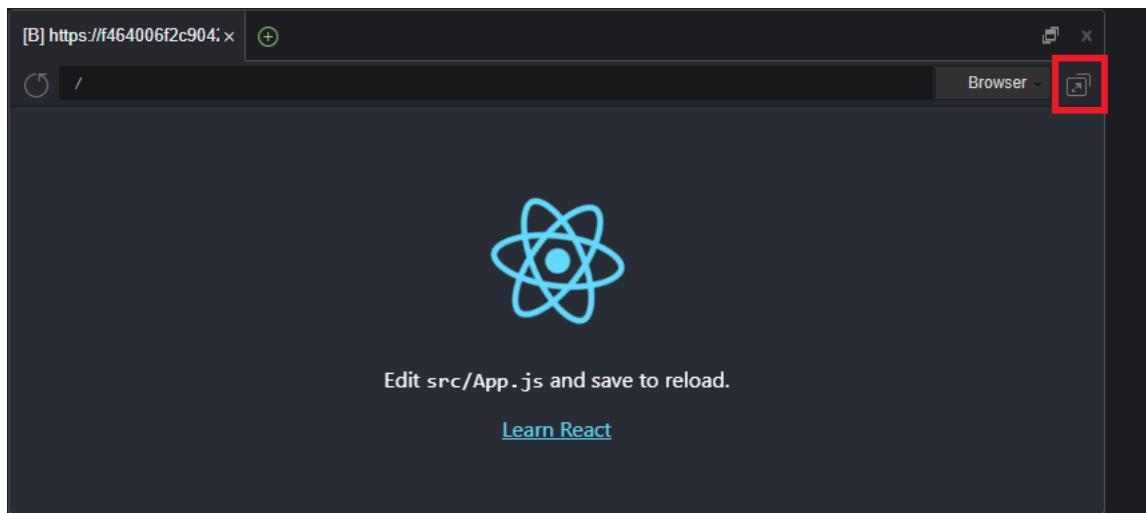
画面上部の [Preview] をクリックして [Preview Running Application] を選択します。



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



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



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

5.2.2. Amplify プロジェクトの初期化

1. 以下のコマンドを実行します。

```
$ amplify init
```

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

最初に、プロジェクトの名前の指定を求められます。

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

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

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

3. 環境の名前を入力します。

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

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

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

4. Amplify CLI で使用するエディタを指定します。

カーソルキーの上下で [Vim] を選択して Enter キーを押します。

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

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

5. 構築するアプリケーションの種類を指定します。

[**javascript**] を選択して Enter キーを押します。

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

6. JavaScript で使用するフレームワークを指定します。

[**react**] を選択して Enter キーを押します。

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

7. プロジェクトのビルド、実行についていくつかの指定を求められます。

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

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

8. AWS CLI のプロファイルを使用するかどうか聞いてきます。

[**y**] を入力して Enter キーを押します。

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

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

9. 使用する AWS CLI のプロファイルを指定します。

[**default**] を選択して Enter キーを押します。

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

10. 全ての項目の指定が終わりましたので、初期化の処理が始まります。

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

初期化の処理が正常に終わると、以下のメッセージが表示されます。

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

5.2.3. API リソースの追加

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

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

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

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

1. 以下のコマンドを実行します。

```
$ amplify add api
```

2. 作成する API の種類を選択します。

[**REST**] を選択して Enter キーを押します。

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

3. 「API リソース」の名前を指定します。

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

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

4. API のパスを指定します。

デフォルト値 [**items**] のまま Enter キーを押します。

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

5. ここからは「Function」リソースに関する設定を行います。

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

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

6. 「Function」リソースの名前を指定します。

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

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

7. Lambda 関数の名前を指定します。

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

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

8. Lambda 関数で使用するランタイムを指定します。

[NodeJS] を選択して Enter キーを押します。

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

9. 使用する Lambda 関数のテンプレートを指定します。

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

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

10. ここからは「Storage」リソースに関する設定を行います。

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

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

11. 「Storage」リソースの名前を指定します。

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

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

12. DynamoDB テーブルの名前を指定します。

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

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

13. DynamoDB の最初の列（属性）を指定します。

[Artist] と入力して Enter キーを押します。

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

14. 属性のデータ型を指定します。

[string] を選択して Enter キーを押します。

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

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

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

16. DynamoDB の 2 番目の列（属性）を指定します。

[Title] と入力して Enter キーを押します。

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

17. 属性のデータ型を指定します。

[string] を選択して Enter キーを押します。

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

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

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

19. パーティションキーに設定する属性を指定します。

[Artist] を選択して Enter キーを押します。

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

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

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

21. ソートキーに設定する属性を指定します。

[Title] を選択して Enter キーを押します。

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

30. リソースの追加が正常に行われると、以下のメッセージが表示されます。

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

5.2.4. API リソースのデプロイ

前の手順で Amplify プロジェクトに「リソース」を追加しました。

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

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

1. 以下のコマンドを実行します。

```
$ amplify push
```

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

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

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

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

```
Current Environment: demo
```

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

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

3. AWS 環境へのデプロイ処理が始まります。

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

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

4. デプロイによって、以下の AWS リソースが作成されています。

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

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

5.2.5. Amplify Library のインストール

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

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

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

1. 以下のコマンドを実行します。

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

2. エラーが発生することなくインストールが完了することを確認します。

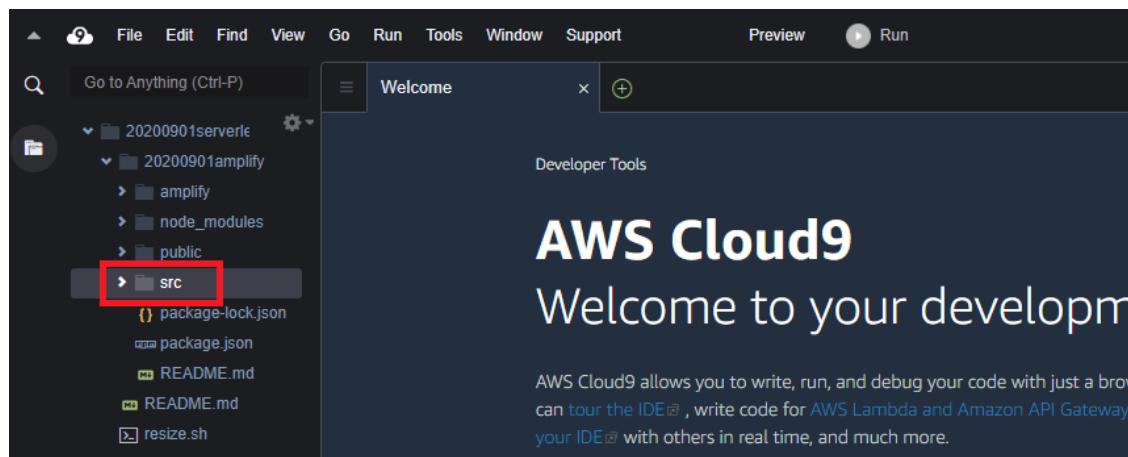
5.2.6. Web クライアントアプリケーションの構成

1. [src] フォルダに以下のファイルがあります。

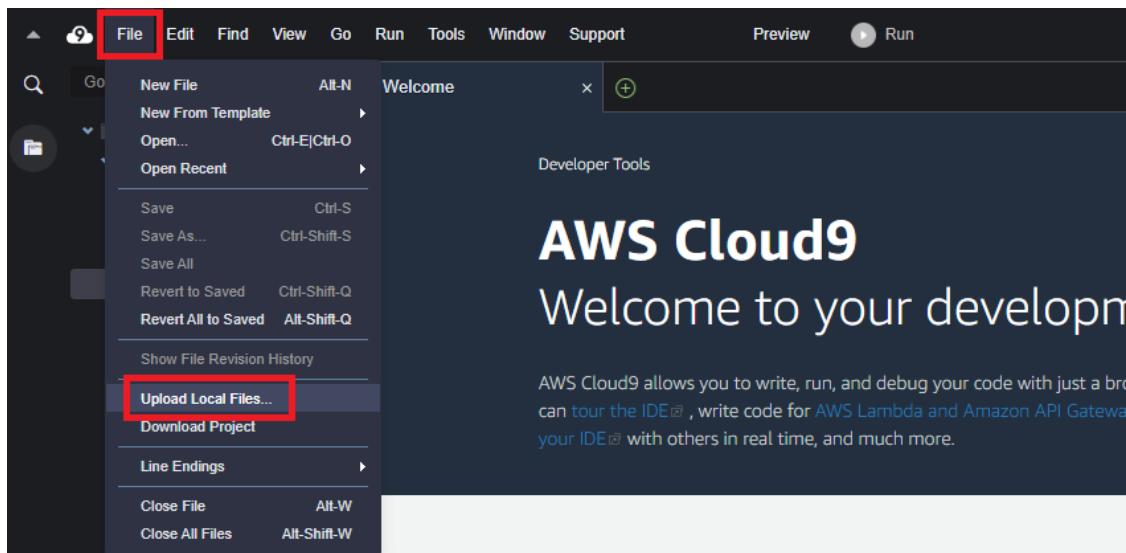
- [ApiGet.js]
- [ApiPost.js]
- [App.css]
- [App.js]
- [aws-logo.png]
- [config.js]

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

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



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

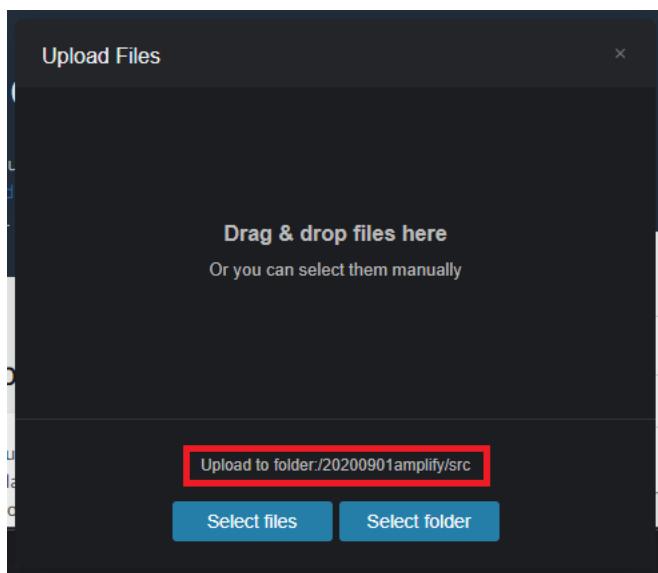


3. 下図のウィンドウが表示されます。

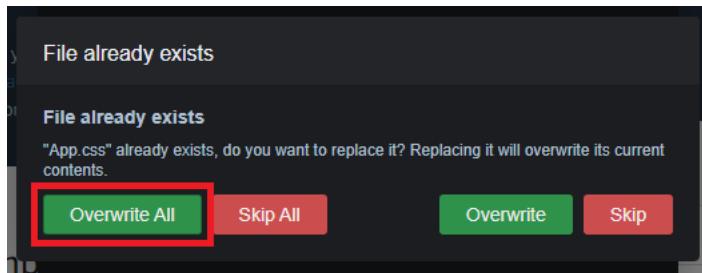
アップロード先ディレクトリが「/YYYYMMDDamplify/src」であることを確認します。

アップロードするファイル群をウィンドウ内にドラッグ&ドロップします。

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



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

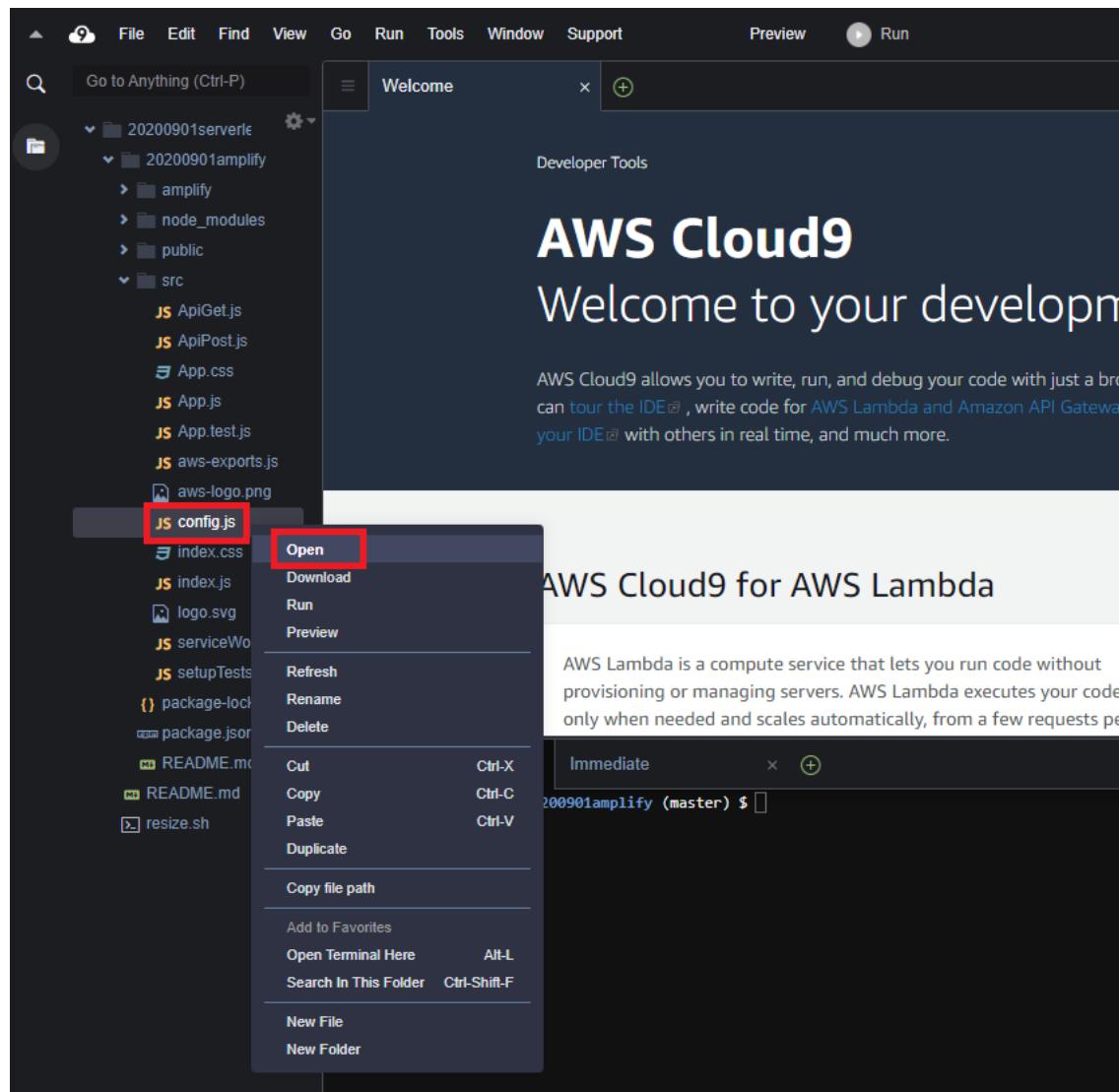


5. ls コマンドを実行して、ファイルがアップロードされたことを確認します。

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

6. リソース名を記述した設定ファイルを編集します。

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



7. API 名を実際に作成した名前に書き換えて、ファイルを保存します。

(パス名は、ここまで手順通りに実施していれば変更する必要はありません)

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

5.2.7. Web クライアントアプリケーションの動作確認

- 以下のコマンドを実行します。

```
$ npm start
```

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

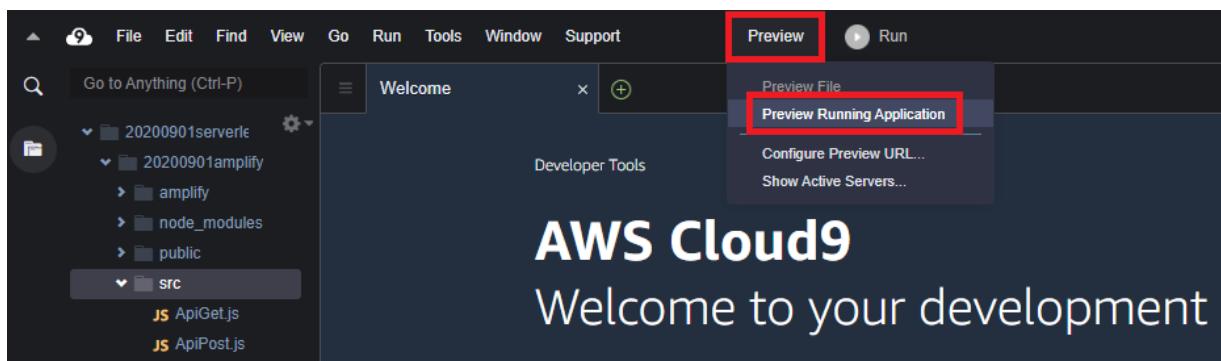
```
Compiled successfully!

You can now view 20200901amplify in the browser.

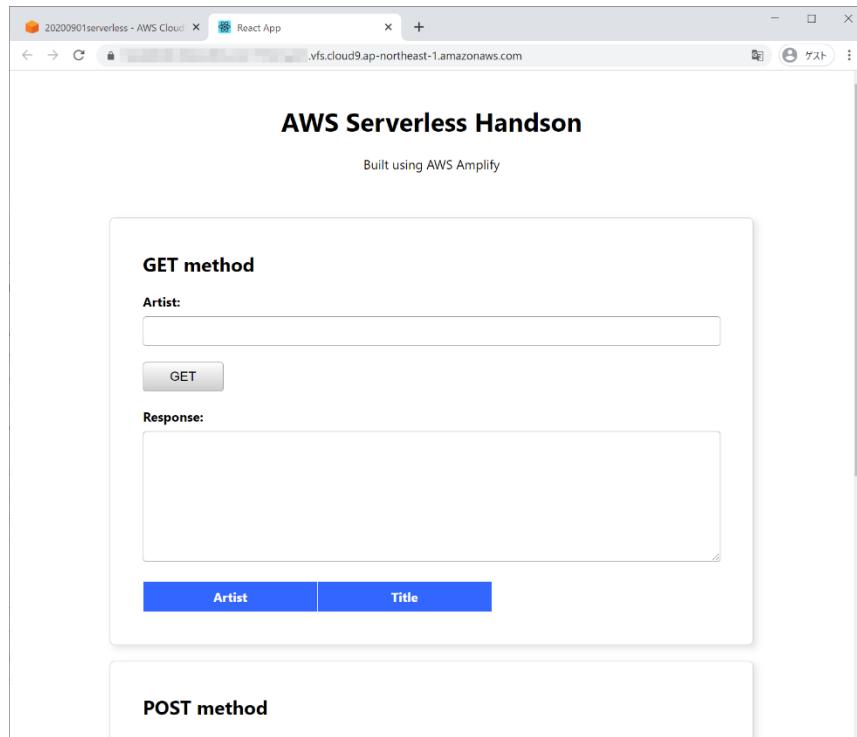
Local:          http://localhost:8080
On Your Network:  http://172.XX.XX.XX:8080

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

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



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



5. 「GET メソッド」 「POST メソッド」 がそれぞれ正常に動作することを確認します。
(まず「POST メソッド」を実行してデータを登録してから、「GET メソッド」を実行するとよいでしょう)

GET method

Artist:

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)
>● create
  ● read
  ● update
  ● delete
```

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

選択が終わりましたら Enter キーを押します

12. リソースの更新に成功すると、以下のメッセージが表示されます。

```
Successfully updated resource
```

5.3.3. Auth リソースおよび API リソースのデプロイ

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

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

1. 以下のコマンドを実行します。

```
$ amplify push
```

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

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

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

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

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

```
Current Environment: demo
```

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

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

3. AWS 環境へのデプロイ処理が始まります。

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

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

4. デプロイによって、以下の AWS リソースが作成されています。

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

- Cognito ユーザープール
 - **YYYYMMDDamplifyXXXXXXX_userpool_XXXXXXX-demo**
- Cognito ID プール
 - **YYYYMMDDamplifyXXXXXXX_identitypool_XXXXXXX__demo**
- IAM ロール
 - snsXXXXXXXXXXXXX-demo
 - upClientLambdaRoleXXXXXXXXXXXXX-demo
- Lambda 関数
 - amplify-**YYYYMMDDamplify-demo-UserPoolClientLambda-XXXXXXX**

また、以下の既存の AWS リソースに対して変更が行われています。

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

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

「認証されたユーザーに割り当てられる IAM ロール」については、ここまで説明していませんでしたが、実は最初の「Amplify プロジェクトの初期化 (amplify init)」の際に既に作成されています。（同時に「認証されていないユーザーに割り当てられる IAM ロール」も作成されています）

「API」リソースに対してアクセス制限を設定したことにより、これらの IAM ロールに対して API Gateway REST API の呼び出し許可を与えるポリシーが登録されています。

5.3.4. Web クライアントアプリケーションの構成

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

まず、ファイル冒頭の「import」の記述群に以下の行を追加します。

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

この行によって、認証を行う 2 つのコンポーネントをインポートします。

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

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

この行によって、画面に「サインアウト」ボタンのコンポーネントを配置します。

- 最後に、ファイルの最終行にある以下の行を、下記のように修正します。

```
export default App;
```

↓

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

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

全ての修正が終わりましたら、ファイルを保存します。

5.3.5. Web クライアントアプリケーションの動作確認

- 以下のコマンドを実行します。

```
$ npm start
```

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

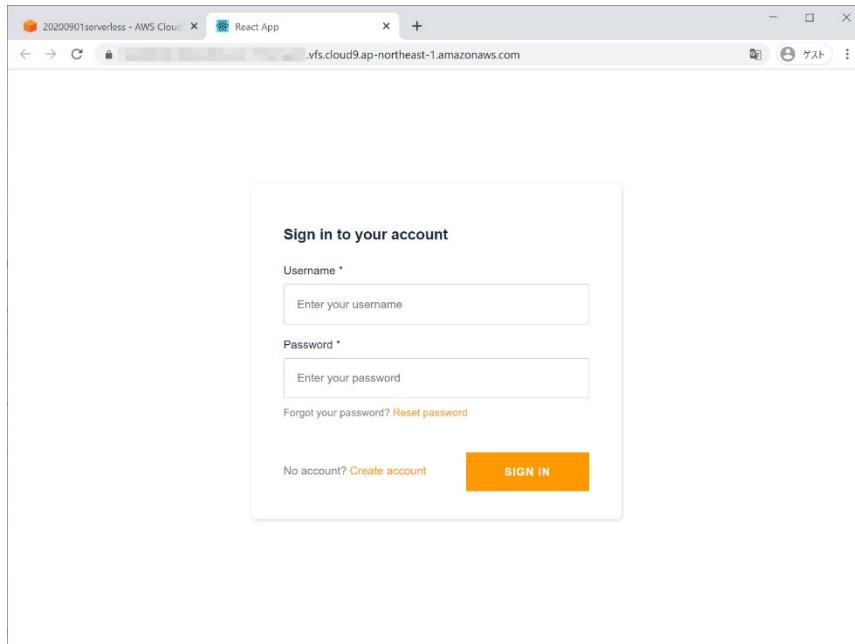
```
Compiled successfully!

You can now view 20200901amplify in the browser.

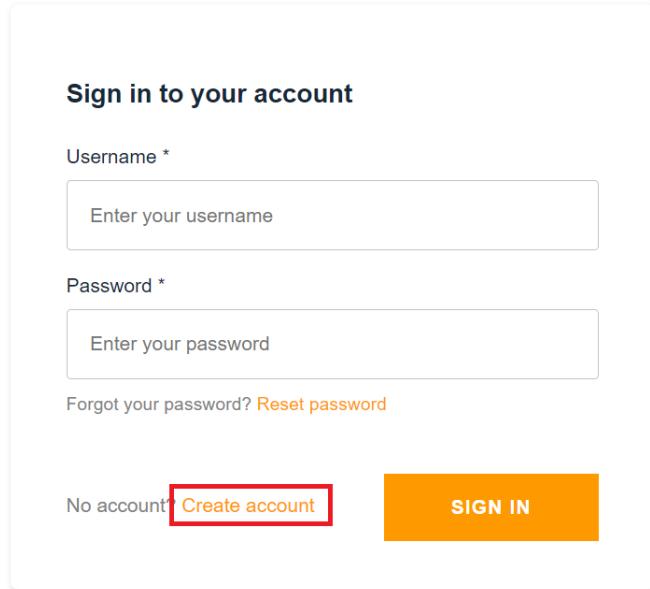
Local:          http://localhost:8080
On Your Network:  http://172.XX.XX:8080

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

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

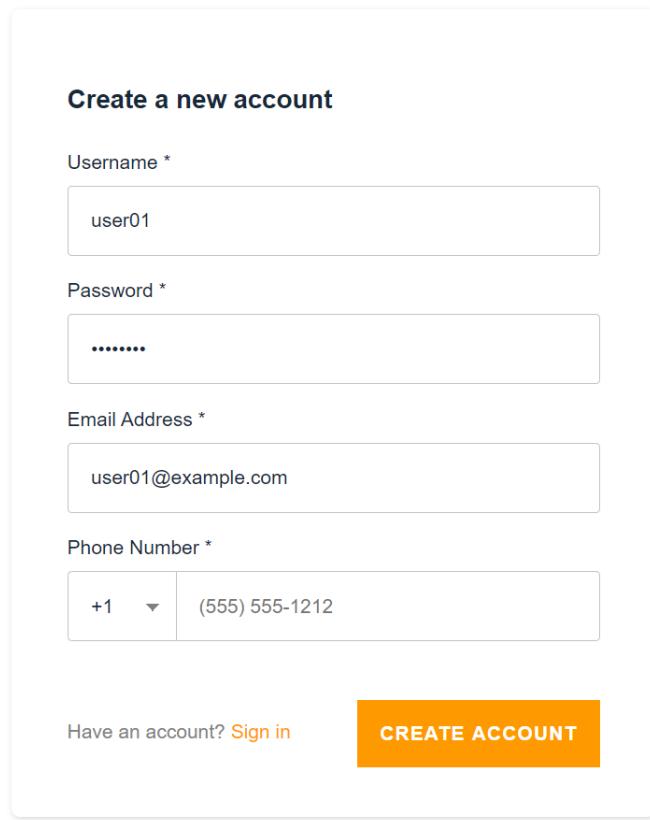


5. まず、[Create account] をクリックしてサインアップを行います。



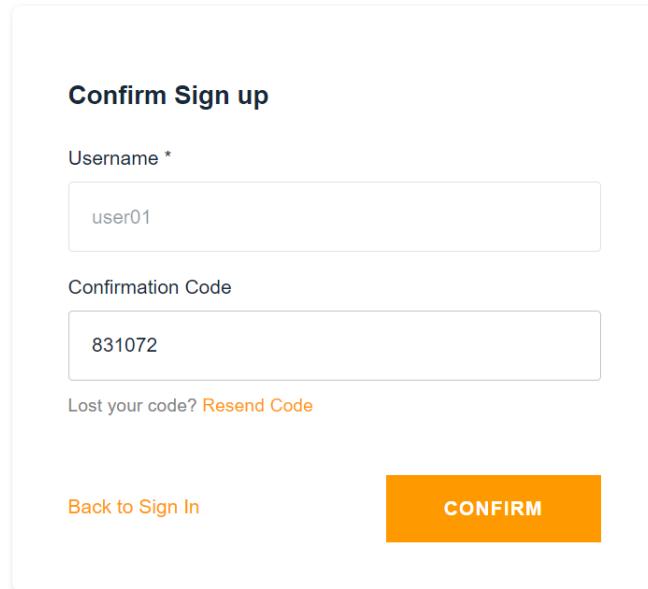
The image shows a sign-in form titled "Sign in to your account". It includes fields for "Username *" and "Password *", both with placeholder text "Enter your username" and "Enter your password" respectively. Below these fields is a link "Forgot your password? [Reset password](#)". At the bottom left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

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



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

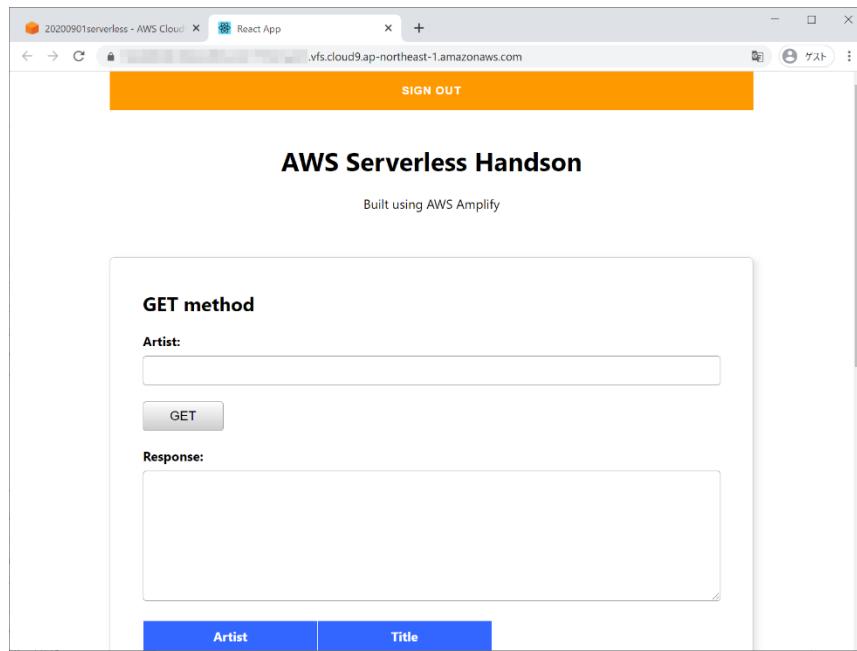
7. メールで確認コードが送られてきますので、入力して **[CONFIRM]** をクリックします。



The screenshot shows a 'Confirm Sign up' form. It has two input fields: 'Username *' containing 'user01' and 'Confirmation Code' containing '831072'. Below the fields is a link 'Lost your code? [Resend Code](#)'. At the bottom are two buttons: 'Back to Sign In' (orange) and a large orange 'CONFIRM' button.

8. 自動的にサインインが行われて、API テストの画面が表示されます。

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



The screenshot shows a web browser window titled '20200901serverless - AWS Cloud' with the URL 'vfs.cloud9.ap-northeast-1.amazonaws.com'. The page displays the 'AWS Serverless Handson' logo and 'Built using AWS Amplify'. Below this is a 'GET method' section with a 'Artist:' input field and a 'GET' button. A 'Response:' text area is also present. At the bottom, there 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したが、ラボ 1 のように「認証されたユーザー」「認証されていないユーザー」それぞれに異なるアクセス権限を与えることも可能です。

その場合は、今回より少し複雑なコードを記述する必要があります。

5.4. Amplify アプリケーションをホスティングする

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

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

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

5.4.1. ホスティングの設定

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

```
$ amplify add hosting
```

2. ホスティングの方式を指定します。

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

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

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

3. CI/CD の方式を指定します。

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

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

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

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

```
Command: amplify publish
```

5.4.2. アプリケーションの公開

- 以下のコマンドを実行します。

```
$ amplify publish
```

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

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

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

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

```
Current Environment: demo
```

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

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

- ホスティング環境を公開する処理が始まります。

処理が終るまで数分程度かかりますので、終わるまで待ちます。

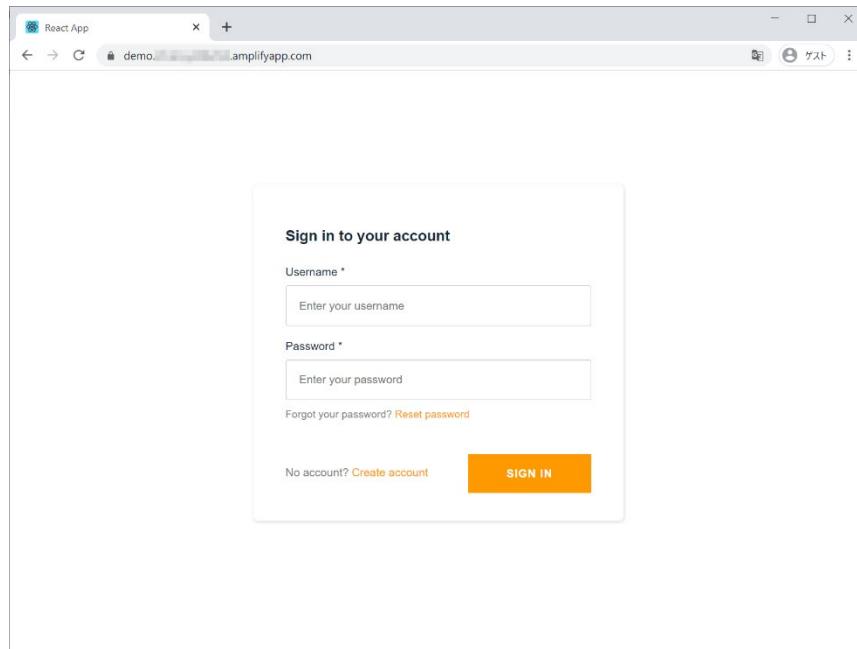
処理が成功すると、以下のメッセージが表示されます。

公開サイトの URL が表示されていることを確認します。

```
✓ Deployment complete!
```

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

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



6. 後片付け

下記、後片付けを行います。

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

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

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

```
$ amplify delete
```

- Cloud9 環境の削除
 - ✧ [YYYYMMDDserverless]

後片付けは以上です。

