ソリューションアーキテクト

2025年1月28日　　9:08

DNS(30点)

2025年1月28日　　9:08

クリア手順

①Route53でホストゾーンをプライベートで作成する[internl.news.org]
②[Internal News Server]のプライベートIPアドレスのAレコードを作成する[thewhitepaper.internl.news.org]
③作成したAレコードのCNAMEレコードを作成する

④CNAMEレコードを入力して検証する

サーバーレス基礎(10点)

2025年1月28日　　9:38

sample_co
de
<file://C:¥Users¥1609844¥Downloads¥sample_code.py>

①コードを編集する

emoji_type = 0 → feeling: 'positive'

emoji_type = 1 → feeling: 'neutral'

else → feeling: 'negative'

レスポンスに`message`と`feeling`を含んでいる必要がある

→課題に指定されたfeelingの値を返すコードを追加する

```
# Please review the comments for each code block to help you understand the execution of this Lambda function.
# At the time you create a Lambda function, you specify a handler,
# which is a function in your code, that AWS Lambda can invoke when the service executes your code.
# The Lambda handler is the first function that executes in your code.
# Python programming model – https://docs.aws.amazon.com/lambda/latest/dg/python-programming-model.html
def lambda_handler(event, context):

    # context – AWS Lambda uses this parameter to provide runtime information to your handler.
    # event – AWS Lambda uses this parameter to pass in event data to the handler. This parameter is usually of the Python dict type.
    # AWS Lambda function Python handler – https://docs.aws.amazon.com/lambda/latest/dg/python-handler.html

    # When you invoke your Lambda function you can determine the content and structure of the event.
    # When an AWS service invokes your function, the event structure varies by service.
    #
    # In this lab, the lambda_handler "event" parameter has the following structure of name/value pairs:
    # {
    #     "emoji_type": number,
    #     "message": "string"
    # }

    # In a name-value pair you can extract the "value" of the pair using the "name" of the pair.
    # Here the id value is extracted from the event Lambda parameter and passed to a variable called emoji_type.
    emoji_type = event["emoji_type"]
    # Here the message value is extracted from the event Lambda parameter and passed to a variable called message.
    message = event["message"]

    # The variables are printed here, which means the variable values will be displayed in CloudWatch logs and the Execution Results panel.
    print(emoji_type)
    print(message)

    # In Python, you can create a variable with no value using the Built-in constant None. This means the variable "custom_message" currently has no value.
    custom_message = None

    # In Python, we use the if-elif-else to create a conditional execution. https://docs.python.org/3/reference/compound_stmts.html#the-if-statement
    # That is, if the value of emoji_type is equal to 0, we execute the statement inside its block
    if emoji_type == 0:
        # Only execute if id is equal to 0
        # The variable custom_message combines "Message for code 0:" string with the variable message
        custom_message = "Message for code 0: " + message
```

```
        custom_feeling = "positive"
    elif emoji_type == 1:
        # The variable custom_message combines "Message for code 1:" string with the variable message
        custom_message = "Message for code 1: " + message
        custom_feeling = "neutral"
    else:
        # The variable custom_message combines "Message for all other codes:" string with the variable message
        custom_message = "Message for all other codes: " + message
        custom_feeling = "negative"
    # Optionally, the handler can return a value. What happens to the returned value depends on the invocation type you use when invoking the
Lambda function
    # In this lab we use synchronous execution, so we need to create a response for the lambda function.
    # We created a "response" variable that has a structure of name-value pairs using the id and custom_message created earlier.
    response = {
        # In this name-value pair, the literal value "message" will store the value of the variable custom_message.
        "message": message,
        # In this name-value pair, the literal value "id" will store the value of the variable id.
        "custom_message": custom_message,
        "feeling": custom_feeling
    }

    # Finally, we return the values from response variable to the caller, which could be a test event or an AWS service performing a synchronous call.
    # The execution of the lambda function is finished.
    return response
```

②lambda関数のarnを入力して検証する

CloudFormation による自動化(40点)

2025年1月28日　　9:10

sample_code
<file://C:¥Users¥1609844¥Downloads¥sample_code.txt>

##FULL STACK CODE##

```
Resources:
  RobotAppServer:
    Type: 'AWS::EC2::Instance'
    Properties:
      InstanceType: t2.micro
      ImageId: ami-087c17d1fe0178315
      SecurityGroups:
      - !Ref RobotAppSecurityGroup
  RobotAppSecurityGroup:
    Type: 'AWS::EC2::SecurityGroup'
    Properties:
      GroupDescription: Enable SSH access via port 22
      SecurityGroupIngress:
      - IpProtocol: tcp
        FromPort: '22'
        ToPort: '22'
        CidrIp: 0.0.0.0/0
  RobotS3Bucket:
    Type: 'AWS::S3::Bucket'
    DeletionPolicy: Delete
```

このJSONコードをコピーして貼り付けてスタックを作成する

IAMロールは指定せずに作成する

クリア手順
①cloudformationでスタックを作成

※Infrastructure Composerからビルド

②Infrastructure ComposerのTemplateにコードを記述する

sample_code.txtの
##FULL STACK CODE##以下のコードをコピーしてInstanceTypeをt2.smallに変更してから作成する**(JSON)**

```
Resources:
  RobotAppServer:
    Type: 'AWS::EC2::Instance'
    Properties:
      InstanceType: t2.small
      ImageId: ami-087c17d1fe0178315
      SecurityGroups:
      - !Ref RobotAppSecurityGroup
  RobotAppSecurityGroup:
    Type: 'AWS::EC2::SecurityGroup'
    Properties:
      GroupDescription: Enable SSH access via port 22
      SecurityGroupIngress:
      - IpProtocol: tcp
        FromPort: '22'
        ToPort: '22'
        CidrIp: 0.0.0.0/0
  RobotS3Bucket:
    Type: 'AWS::S3::Bucket'
    DeletionPolicy: Delete
```

③スタック名などを設定

IAMロールは指定しない！

④スタック名を入力して検証する

RESTful APIのデプロイ(20点)

2025年1月28日　　18:35

sample_co
de (1)

vehicles_fu
nction
<file://C:¥Users¥1609844¥Downloads¥vehicles_function.py>

①vehicles_function.pyをコピーしてLmabda関数を作成する

②REST APIを作成してリソース /vehicles を作成する

③リソース /vehicles/{id}を作成する

④APIをデプロイする

⑤<API URL>/vehiclesを入力して検証する

API とデータベース(30点)

2025年1月28日　　9:53

sample_co
de (2)

①Dynamodbテーブルを作成する

テーブル名 locations

パーティションキー id(文字列)

②sample_codeからLamda関数を作成する

ロールは既存のロールを使用

③REST APIを作成してリソース /vehicles を作成する

④リソース /vehicles/{id}を作成する

⑤

URL

API Gateway呼び出しURL

<api gatewayのステージURL>/vehicles

DynamoDBテーブルの名前

locations

を入力して検証する

ネットワークトラフィックの分析(30点)

2025年1月28日　13:40

①s3バケットに保存されているテキストファイルに書かれているIPアドレスをコピーする

②ネットワークACIでアウトバウンドルールを作成する

ルール番号　10
タイプ　全てのトラフィック
送信先　<コピーしたIPアドレス>
許可/拒否　拒否

データ取り込み方法(40点)

2025年1月28日　　11:24

①Kinesis Firehose を作成する

ソース Direct PUT

送信先 Amazon S3

データ変換をオンにする

Lambda関数 DataProcessingFunction

s3 databucket選択

バケットプレフィックス processed_data/

エラープレフィックス error/

既存のIAMロールを選択 LabKinesisFirehoseRole

バッファヒントを変更する **60秒**

②EC2インスタンスのアプリにアクセスしてFirehoseにデータを出力する

ブラウザから　http://<インスタンスのパブリックIP>/firehose

firehoseのストリーム名を入力してSubmitする

→firehoseにデータが送信される

③Lambda関数でFirehoseから受け取ったデータをDynamodbへ登録するコードを記述する

属性にvalueは必須

Lambda関数にdynamodbにレコードを追加するコードを追加する

```
    dynamodb = boto3.resource('dynamodb', region_code='us-east-1')
    table = dynamodb.Table('OutputTable')
```

```
item = {
    "timestamp": str(datetime.now()),
    "recordId": record['recordId'],
    "result": "Ok",
```

```
        "value": base64.b64encode(df.to_csv(header=False, index=False).encode('utf-8'))
    }
    table.put_item(Item=item)
```

④テーブルにデータが登録されたことを確認してからテーブル名を入力して検証する

リソースモニタリング(20点)

2025年1月28日　　10:03

mem_usedをメトリクスとして選択して
ダッシュボードとアラームを作成する
アラームのアクションでインスタンスを再起動するアクションを選択する
アラームのメトリクスで使用しているインスタンスIDで検証

①ダッシュボードを作成する
　　メトリクス **mem_used** を追加する

②アラームを作成する
　　インスタンス名(Server1)のmem_usedをメトリクスとして追加する

　　条件
　　　　しきい値　静的
　　　　mem_usedが300000000よりも大きい

　　アクション
　　　　通知　新しいトピックスを作成する

　　　　EC2アクション　このインスタンスを再起動

データをバックアップする(20点)

2025年1月28日　　9:13

①バックアッププランを作成する
　　頻度は12時間毎

②リソースの割り当て
　　ロール backup_role
　　タグ　Buckup-DIY = True

③バックアッププランIDを入力して検証する

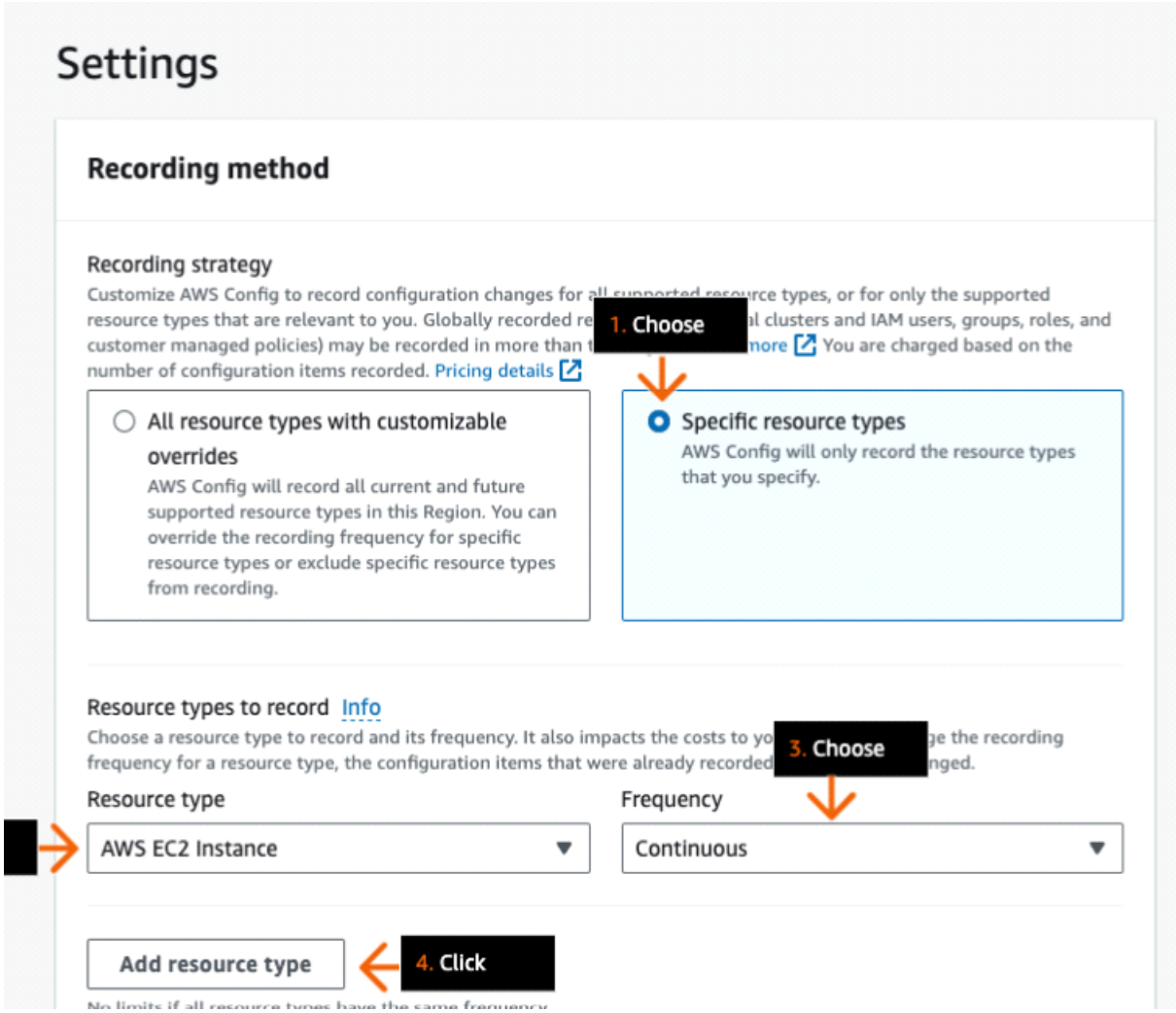
リソースガバナンス(40点)

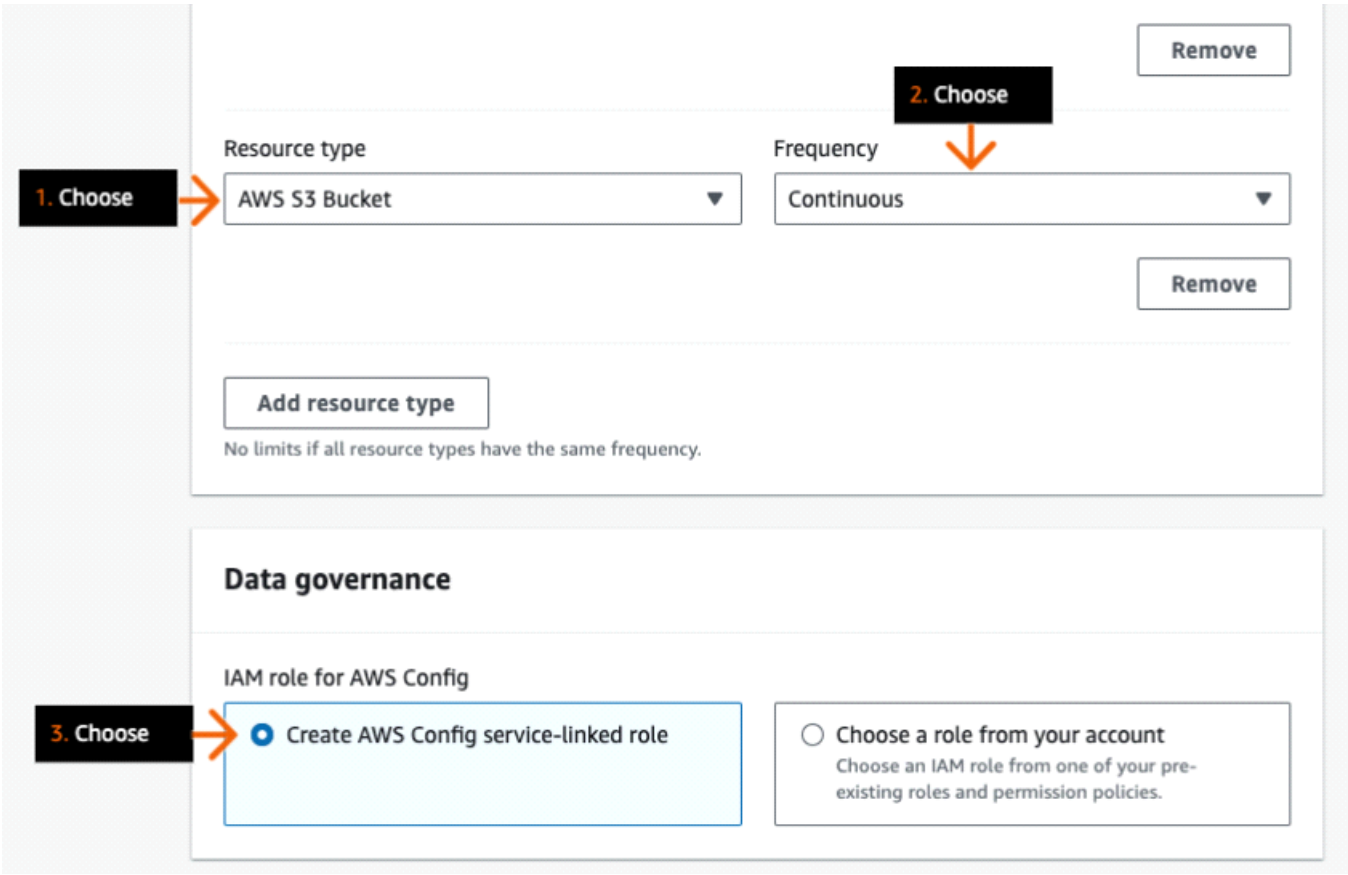
2025年1月28日　　10:16

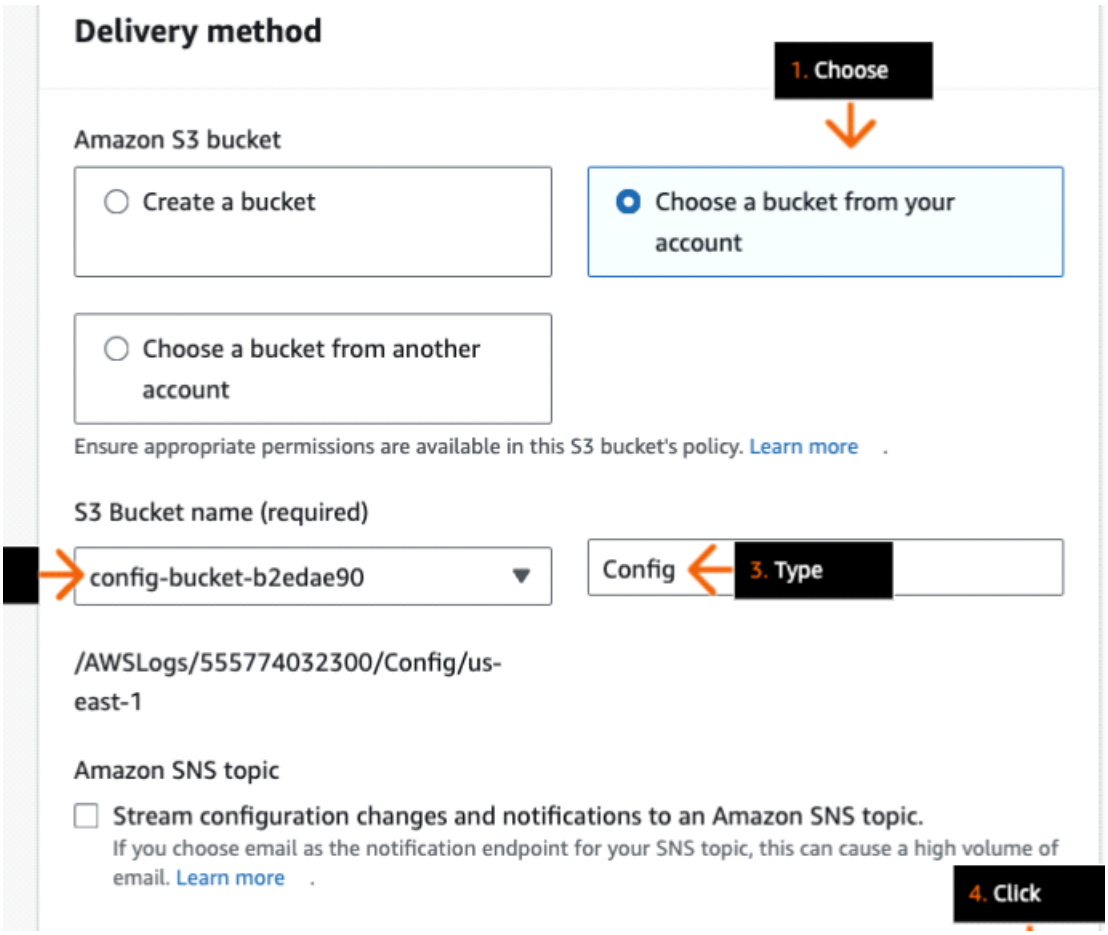
①s3バケットのバージョニングを有効にする

②s3バケットの暗号化設定で既存のKMSキーを使用する

③Configから「今すぐ始める」でルールを作成する
　　ルールには**bucket-versioning-enabled**を選択




Remove
2. Choose
Resource type
Frequency
1. Choose
AWS S3 Bucket
Continuous
Remove
Add resource type
No limits if all resource types have the same frequency.
Data governance
IAM role for AWS Config
3. Choose
Create AWS Config service-linked role
Choose a role from your account
Choose an IAM role from one of your pre-existing roles and permission policies.


Delivery method
1. Choose
Amazon S3 bucket
Create a bucket
Choose a bucket from your account
Choose a bucket from another account
Ensure appropriate permissions are available in this S3 bucket's policy. Learn more
S3 Bucket name (required)
config-bucket-b2edae90
Config
3. Type
/AWSLogs/555774032300/Config/us-east-1
Amazon SNS topic
Stream configuration changes and notifications to an Amazon SNS topic.
If you choose email as the notification endpoint for your SNS topic, this can cause a high volume of email. Learn more
4. Click

④ルール名とs3バケット名を入力して検証する

コンテナサービス(20点)

2025年1月28日　　13:21

①ec2からリポジトリを作成

ec2にセッションマネージャーで接続する

sudo su ec2-user

dockerコマンドインストール

./install_scripts/install_docker.sh

解凍する

unzip lab-codes.zip

cd second_app

aws ecr create-repository --repository-name "${repo_name}"

ecrのプッシュコマンドでプッシュする

②タスク定義を作成してパブリックサブネットで起動させる

セキュリティグループで8443番を全許可する

コンテンツ配信ネットワーク(30点)

2025年1月28日　　17:56

①オリジンにs3を指定してディストリビューションを作成する
　　オリジンアクセスはPublic

②ディストリビューションがデプロイされたことを確認後
　　<URL>/index.html を入力して検証する

アプリケーションのデカップリング(30点)

2025年1月28日　　11:30

sample_co
de (1)

<URL>/index.html
で検証

ダウンロードしたファイルのキューポリシーを14行目に貼り付ける

```
{
  "Version": "2012-10-17",
  "Id": "__default_policy_ID",
  "Statement": [
    {
      "Sid": "__owner_statement",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::759030240478:root"
      },
      "Action": "SQS:*",
      "Resource": "arn:aws:sqs:us-east-1:759030240478:sqs"
    },
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "sns.amazonaws.com"
      },
      "Action": "sqs:SendMessage",
      "Resource": "arn:aws:sqs:us-east-1:759030240478:sqs",
      "Condition": {
        "ArnEquals": {
          "aws:SourceArn": "arn:aws:sns:us-east-1:759030240478:sns"
        }
      }
```

```
    },
    {
      "Sid": "topic-subscription-arn:aws:sns:us-east-1:759030240478:sns",
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
      },
      "Action": "SQS:SendMessage",
      "Resource": "arn:aws:sqs:us-east-1:759030240478:sqs",
      "Condition": {
        "ArnLike": {
          "aws:SourceArn": "arn:aws:sns:us-east-1:759030240478:sns"
        }
      }
    }
  ]
}
```

作成したsnsトピックをサブスクリプションとして登録する

sqsの**url**とsnsのarnを入力して検証する

生成AIでのクラウドインフラストラクチャ(40点)

2025年1月28日　　13:52

SageMaker Studioを開く

Code Editor

Applicationを開く

Open Folderで/home/sagemaker-user/を開く

Tarminalを開く

mkdir cdkapp

cdk init -a app -l=python

source .venv/bin/activate

pip3 install -r requirements.txt

s3バケットへ移動し、userdata.shのS3URLをコピーする

cd cdkapp

aws s3 cp <S3URL> .

cdk.jsonファイル編集
"@aws-cdk/aws-ec2:restrictDefaultSecurityGroup": true,

[true] の値を [false] に変更します。

cdkapp_stack.pyを編集

../

sdk synth

sdk deploy

→cloudformationでスタックが作成されて動作する

ターゲットグループにインスタンスを追加する

スタック名とインスタンスidを入力して検証する

cdkapp_stack

シングルページアプリケーション(110点)

2025年1月28日　　13:26

policy

①バケットをパブリックアクセスに変更する

②APIGatewayを作成

/vehicles　GETメソッドでLambda関数(find_all_vehicles)を選択する

/vehicles/{id}　GETメソッドでLambda関数(find_vehicles)を選択する

③lambdaの権限を設定(両方)

lambdaの「AWSポリシーテンプレートから新しいロールを作成」でDynamodbのシンプルポリシーを追加する