

AWS Migration Hub Refactor Spaces ワークショップ(後半用)

はじめに：

この資料は AWS Migration Hub Refactor Spaces を使ってモノリシックなウェブアプリケーションをマイクロサービスに移行する手順を紹介します。事前に以下の URL で紹介されている手順でモノリシックアプリケーションの構築を完了させてください。

<https://github.com/harunobukameda/AWS-Migration-Hub-Refactor-Spaces>

上記のハンズオンでは、マイクロサービスに移行する方法として Lambda を使って移行する方法を紹介していますが、この Lambda は大きい Java を使用しています。Lambda を Java で動かそうとするとランタイムの立ち上がりに時間がかかるため、全体の動作時間がとても遅くなります。

そこで立ち上がりに時間がかからないスクリプト言語として Python を使用して Lambda 関数を再実装し、若干手順をアレンジしたのが本レポジトリの内容です。

マイクロサービス環境の構築：

それでは早速カート処理機能実行する環境をマイクロサービスで構築していきます。

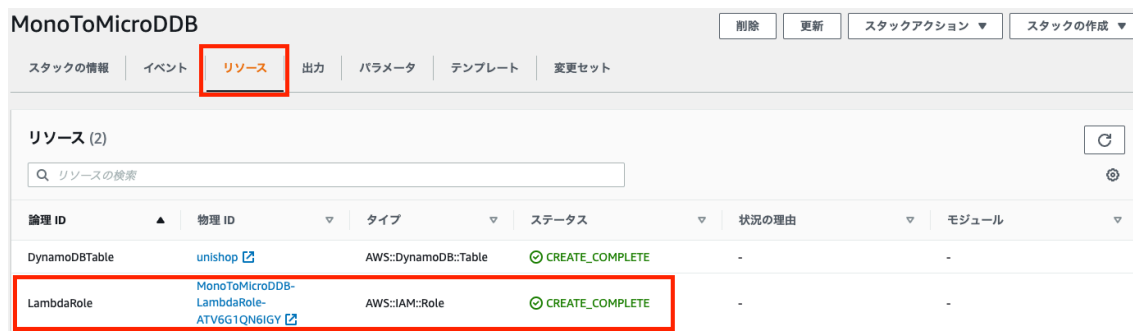
1. 「MonoToMicroCFDDB.yaml」をレポジトリからダウンロードします
2. 前半パートと同様に CFn にてスタックを作成します。スタック名は「MonoToMicroDDB」としてください。「CREATE_COMPLETE」となったら、念の為 Dynamo DB のマネジメントコンソールで Dynamo DB を開いて「unishop」というテーブルが作成されていることを確認します

The screenshot shows the AWS Management Console interface for the 'unishop' table in DynamoDB. The left sidebar shows the 'unishop' table selected under 'DynamoDB > テーブル > unishop'. The main content area displays the table's details, including its name 'unishop', partition key 'uuid (String)', sort key '-', and capacity mode 'オンデマンド'. The table status is 'Active' with 'アクティブなアラームなし'. The '項目の概要' (Summary) section shows 0 items, 0 bytes of table size, and 0 bytes of average item size.

一般的な情報			
パーティションキー uuid (String)	ソートキー -	キャパシティーモード オンデマンド	テーブルの状態 Active アクティブなアラームなし

項目の概要		
項目数 0	テーブルサイズ 0 バイト	項目の平均サイズ 0 バイト

3. CFn のリソースタブから作成された IAM ロールの名前をコピーしておきます



4. Lambda マネジメントコンソールに移動して、「関数の作成」ボタンをクリックします



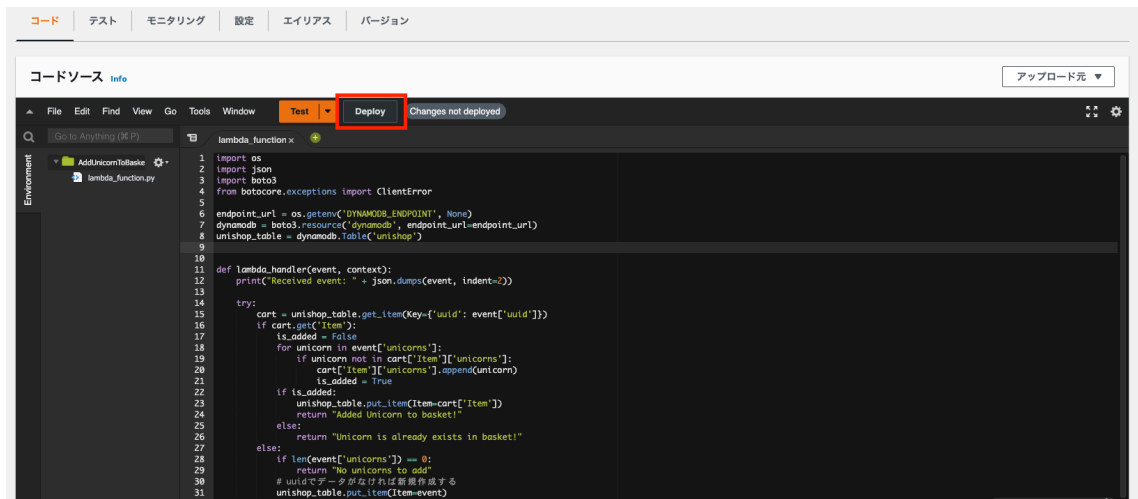
5. 「一から作成」を選択して、関数名を「AddUnicornToBasket」、ランタイムは「Python 3.9」に設定します



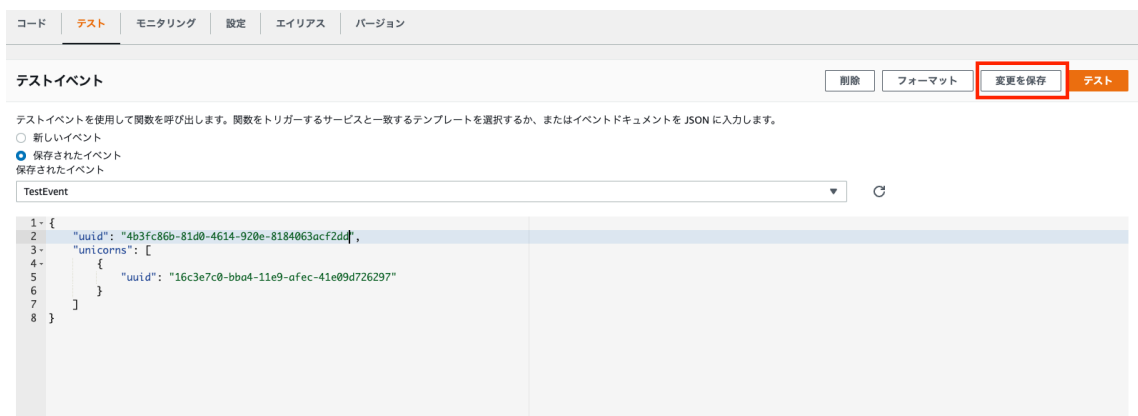
6. 「デフォルトの実行ロールの変更」をクリックして、「既存のロールを使用する」を選択して、先程 CFn を作成したときにコピーした名前の IAM ロールを Lambda の実行ロールとして設定します。



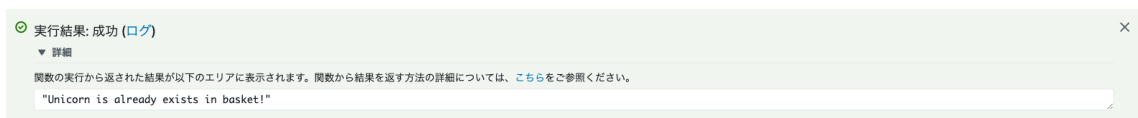
- 「関数の作成」ボタンをクリックします
- レポジトリの「AddUnicornToBasketV2」ディレクトリ内にある「lambda_function.py」の中身をコピーします
- コピーしたら先ほど作成した Lambda 関数のテキストエディタに貼り付けて「Deploy」ボタンをクリックします(画面が黒いのは作者の好みでダークテーマにしています)



- 「テスト」タブをクリックして、テストイベントの設定画面を開きます
- レポジトリの「testcommand.txt」を開き、「AddUnicornToBasket」に書かれている Json をコピーして Json エディタに貼り付けたらイベント名を「TestEvent」にして「変更を保存」ボタンをクリックします。



- 「テスト」ボタンをクリックしてテストを実行します。以下のように実行結果が「成功」と表示されたら設定は完了です



- 4 から 12 の手順であと 2 つ関数を作成します

名前：GetUnicornsBasket

コード：GetUnicornsBasketV2/lambda_function.py

名前：RemoveUnicornFromBasket

コード：RemoveUnicornFromBasketV2/lambda_function.py

ストラングラーフィグパターンの実装：

Lambda 関数ができましたので、API Gateway の向き先を変更していきます。現在 Refactor Spaces ではデフォルトとしてすべてがモノリシック環境へルーティングされていますが、関数単位で Lambda 関数へルーティングを変更していきます

14. Refactor Spaces の左ペインで「サービスを作成」をクリックします

15. 先ほどと同様に環境は「unistore-dev」、アプリケーションは「unistore」を設定します

サービスを作成 情報

環境内のアプリケーションにはサービスが含まれています。各サービスには、エンドポイントがあります (HTTP URL または AWS Lambda 関数のいずれか)。サービスのエンドポイントにルートを追加して、既存のアプリケーションから新しいサービスにトラフィックを再ルーティングできます。

アプリケーションを選択

環境
お客様が所有する環境とお客様と共有されている環境。

unistore-dev ▼ 

アプリケーション
選択した環境でアプリケーションを選択します。

unistore ▼ 

16. サービス名は「AddToCartService」を設定します

サービスの詳細

サービス名

AddToCartService

サービスの説明 - オプション

説明を入力

17. サービスエンドポイントタイプは「Lambda」を選択して、使用する Lambda は先程作成した「AddUnicornToBasket」を選択します

サービスエンドポイントを設定

サービスエンドポイントタイプを選択

☐ VPC
サービスは VPC の URL エンドポイントです。

☒ Lambda
サービスは Lambda 関数です。

❗ 別のアカウントで Lambda をお探しですか? その別の AWS アカウントにサインインして、そのアカウントからサービスを追加してください。

現在のアカウントの Lambda を選択

Lambda の説明

AddUnicornToBasket

18. ルーティングは以下の設定をしてください

ソースパス : /unicorns/basket

子パスを含める : チェックを外す

動詞 : POST

このサービスにトラフィックをルーティング - オプション

❗ ルートが作成されると、そのルートはすぐにアクティブになり、トラフィックがサービスに送信されます。ソースパスのすべてのトラフィックは、アプリケーションのデフォルトルートではなく、このサービスにルーティングされます。

ソースパス
ルートへのパス。

/unicorns/basket

☐ 子パスを含める

動詞

オプションを選択

POST X

19. 一通り設定したら、「サービスを作成」ボタンをクリックしてサービスの作成を完了させます

20. 15 から 19 の手順でサービスをもう一つ作成します

名前 : RemoveCartService

エンドポイント : Lambda

Lambda : RemoveUnicornFromBasket

ソースパス：/unicorns/basket

子パスを含める：チェックを外す

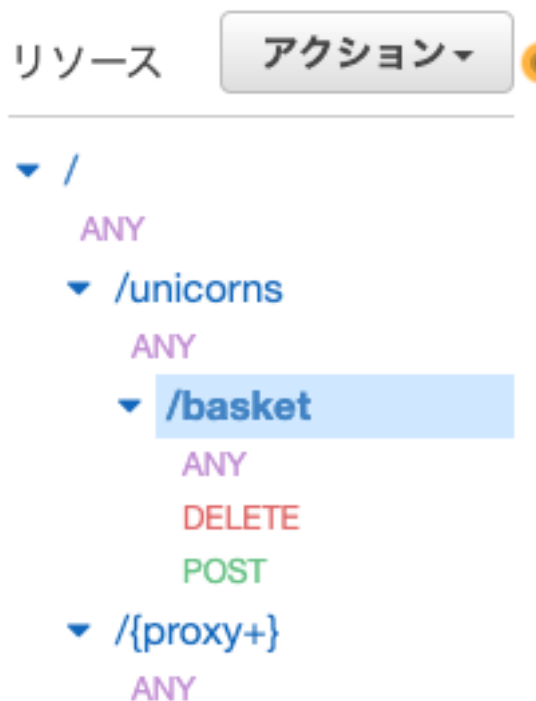
動詞：DELETE

21. ブラウザの別タブで API Gateway のマネジメントコンソールを開きます

22. API の「unistore」をクリックします

API (1)						
<input type="text" value="API の検索"/>						
名前 ▲	説明	ID	プロトコル	エンドポイントタイプ	作成した	
<input type="radio"/> unistore	Managed by AWS Migration Hub Refactor Spaces application app-OdaHEbhBPJ	aa4uyzh0s7	REST	Regional	2022-03-06	

23. 以下のように先程作成した DELETE と POST が表示されていることが確認できます



24. 左ペインの「モデル」をクリックして「作成」をクリックします

API
カスタムドメイン名
VPC リンク

モデル

作成

</> Empty
</> Error

API: **unistore**
リソース
ステージ
オーソライザー
ゲートウェイのレスポンス

モデル

リソースポリシー
ドキュメント
ダッシュボード
設定

25. モデル名に「UnicornBasket」と入力して、コンテンツタイプに「application/json」と入力します

モデル名*

UnicornBasket

コンテンツタイプ*

application/json

モデルの説明

26. testcommand.txt の UnicornBasket GetSchema の Json をコピーして「モデルの作成」をクリックします

モデルのスキーマ*

```

1- {
2  "$schema": "http://json-schema.org/draft-04/schema#",
3  "title": "UnicornBasket",
4  "type": "object",
5  "properties": {
6    "uuid": { "type": "string" }
7  }
8 }

```

* 必須

キャンセル

モデルの作成

27. Refactor Spaces に戻り先程と同様に 3 回目の「サービス作成」を行います。以下の設定を行います

名前：GetCartService

エンドポイント：Lambda

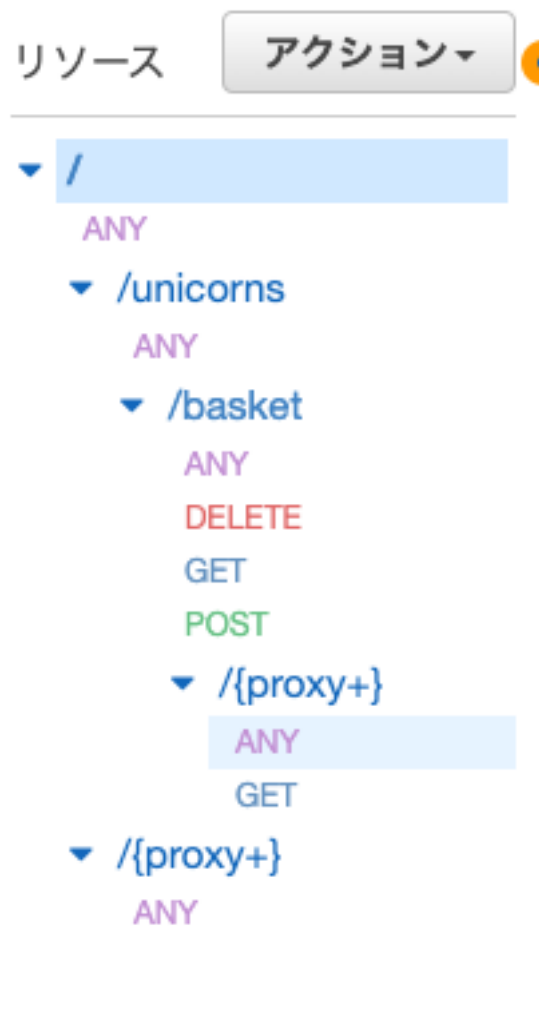
Lambda：GetUnicornBasket

ソースパス：/unicorns/basket

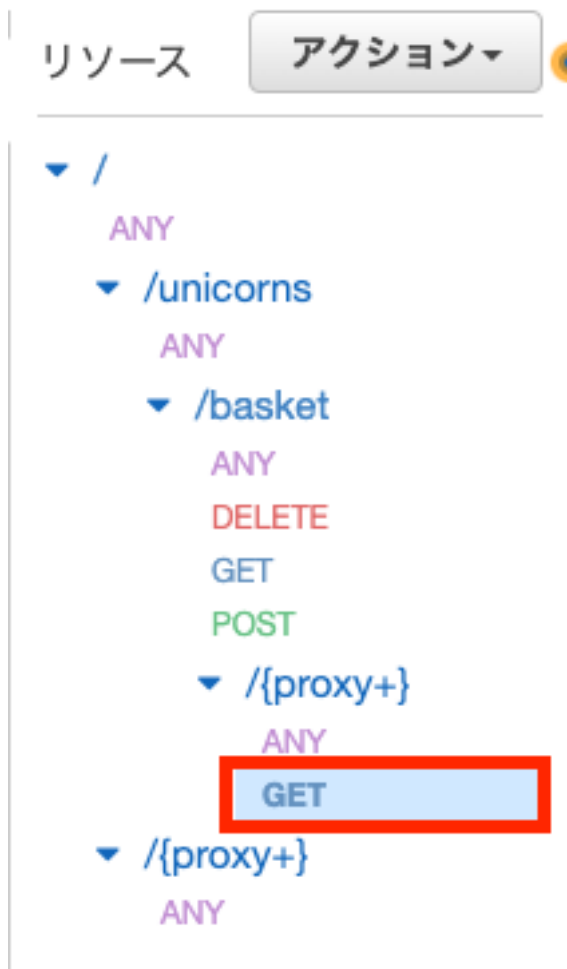
子パスを含める：チェックを付ける(今までと異なるので要注意)

動詞：GET

28. API Gateway のマネジメントコンソールから unistore リソースを開いて以下の構成になっていれば問題ありません。もし、どこかずれていた場合はサービスの作成ミスがあるので、デフォルトのサービス以外を削除してサービスの作成をやり直してください



29. /unicorns/basket/{proxy+}配下の GET をクリックします



30. 「統合リクエスト」をクリックします



31. 「Lambda プロキシ統合の使用」のチェックをはずします。このときいくつか確認ダイアログが出ますが、全て OK をクリックします

←メソッドの実行 /unicorns/basket/{proxy+} - GET - 統合リクエスト

このメソッドが呼び出すターゲットとなるバックエンドに関する情報と受信したリクエストデータを変更するかどうかを指定します。

統合タイプ ☒ Lambda 関数 ⓘ

☐ HTTP ⓘ

☐ Mock ⓘ

☐ AWS サービス ⓘ

☐ VPC リンク ⓘ

Lambda プロキシ統合の使用 ☐ ⓘ

Lambda リージョン us-east-1 ✎

Lambda 関数 GetUnicornsBasket ✎

32. 「/unicorns/basket」の POST と DELETE に対しても同様の作業を行います

33. 再度「/unicorns/basket/{proxy+}」の GET の画面に戻り統合リクエスト画面を開きます。画面一番下の「マッピングテンプレート」を開きます

←メソッドの実行 /unicorns/basket/{proxy+} - GET - 統合リクエスト

このメソッドが呼び出すターゲットとなるバックエンドに関する情報と受信したリクエストデータを変更するかどうかを指定します。

統合タイプ ☒ Lambda 関数 ⓘ

☐ HTTP ⓘ

☐ Mock ⓘ

☐ AWS サービス ⓘ

☐ VPC リンク ⓘ

Lambda プロキシ統合の使用 ☐ ⓘ

Lambda リージョン us-east-1 ✎

Lambda 関数 GetUnicornsBasket ✎

実行ロール ✎

発信者の認証情報を使用した呼び出し ☐ ⓘ

認証情報キャッシュ 発信者の認証情報をキャッシュキーに追加しない ✎

デフォルトタイムアウトの使用 ☒ ⓘ

▶ 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

▶ マッピングテンプレート ⓘ

34. 「テンプレートが定義されていない場合(推奨)」を選択し、「マッピングテンプレートの追加」のプラスマークをクリックします

▼ マッピングテンプレート 🟡

リクエスト本文のパススルー ☐ リクエストの Content-Type ヘッダーに一致するテンプレートがない場合 ⓘ
☒ テンプレートが定義されていない場合 (推奨) ⓘ
☐ なし ⓘ

Content-Type

マッピングテンプレートが定義されていません。リクエスト本文は統合エンドポイントに渡されます

+ マッピングテンプレートの追加

35. 「Content-Type」は「application/json」を入力してチェックボタンをクリックします

リクエスト本文のパススルー ☐ リクエストの Content-Type ヘッダーに一致するテンプレートがない場合 ⓘ
☒ テンプレートが定義されていない場合 (推奨) ⓘ
☐ なし ⓘ

Content-Type

application/json

+ マッピングテンプレートの追加

36. 「テンプレートの生成」ドロップダウンから「UnicornBasket」を選びます

application/json

テンプレートの生成 ✓

メソッドリクエストのパススルー
モデル
Empty
Error
UnicornBasket

37. testcommand.txt の「UnicornBasket model」の値をコピーして、テンプレートのエディタに貼り付けて「保存」をクリックします。このとき、

「保存」をクリックしても何も変化が起きませんが正常に保存できています

application/json

テンプレートの生成:

```
1 #set($inputRoot = $input.path('$'))
2 {
3   "uuid" : "$input.params('proxy')"
4 }
```

キャンセル 保存

38. 「/unicorns/basket」の GET を選択して、「アクション→メソッドの削除」でメソッドを削除します(間違って「/unicorns/basket/{proxy+}」配下の GET メソッドを削除しないようにしてください！！)



39. 以下の構成になっていることを確認します



40. 再度「/unicorns/basket/{proxy+}」の GET メソッドをクリックして「メソッドレスポンス」をクリックします



41. ステータスコード「200」を入力し、チェックボタンをクリックします

このメソッドのレスポンスタイプ、ヘッダーおよびコンテンツタイプに関する情報を指定します。

The screenshot shows the 'メソッドレスポンス' (Method Response) configuration screen. The 'HTTP ステータス' (HTTP Status) field is set to 200, and the '出力バースループ' (Output Screenshot) button is highlighted.

42. 「/unicorns/basket」配下の POST メソッドと DELETE メソッドに対しても同様にメソッドレスポンスを設定します
43. 「/unicorns/basket」パスをクリックして「アクション→CORSの有効化」を選択します



44. 設定は特にいじらずに「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

CORSの有効化

unistore API のゲートウェイレスポンス ☐ DEFAULT 4XX ☐ DEFAULT 5XX ⓘ

メソッド ☒ DELETE ☒ POST ⓘ OPTIONS ⓘ

Access-Control-Allow-Methods DELETE, GET, HEAD, OPTIONS, PATCH, POST, PUT ⓘ

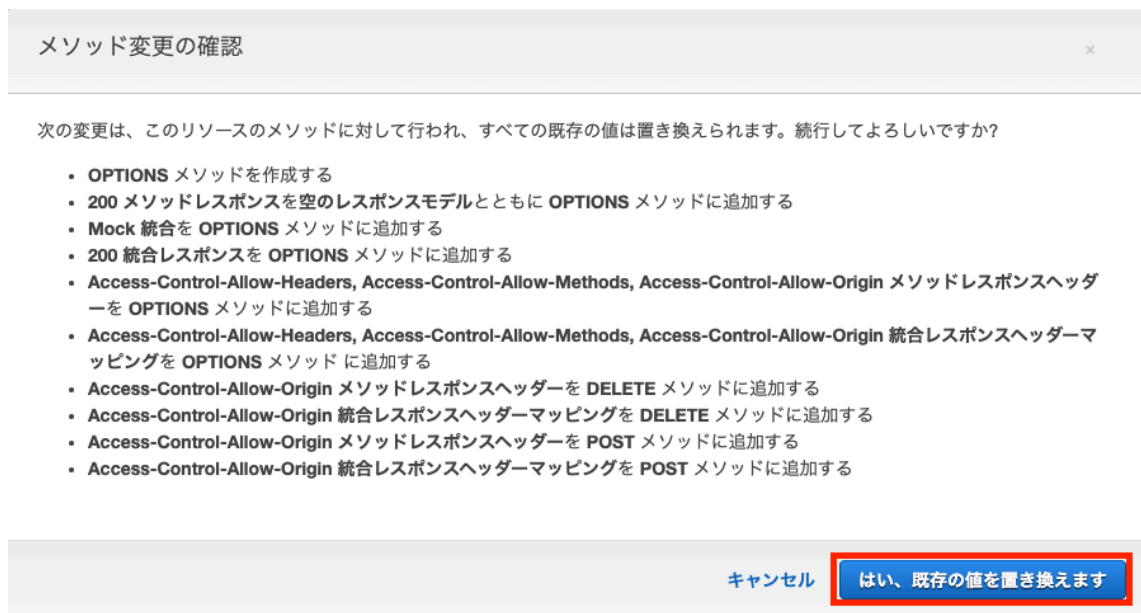
Access-Control-Allow-Headers 'Content-Type,X-Amz-Date,Authoriz ⓘ

Access-Control-Allow-Origin* "" ⓘ

▶ アドバンスド

CORSを有効にして既存の CORS ヘッダーを置換

45. 確認のダイアログが表示されるので「はい、既存の値を置き換えます」をクリックします



46. ここまでの設定が正しければ以下のようにすべての項目にチェックが入ります。もし何かエラーが出たときにはメソッドの設定を再度かくにんしてください



47. 「/unicorns/basket/{proxy+}」を選択し同様に CORS の有効化を行います

48. ルートパス(/)を選択し、「アクション→API のデプロイ」を選択します



49. デプロイされるステージは「prod」を選択し「デプロイ」をクリックします

API のデプロイ

×

API がデプロイされるステージを選択します。たとえば、API のテスト版をベータという名前のステージにデプロイできます。

デプロイされるステージ

prod

▼

デプロイメントの説明

↵

キャンセル

デプロイ

50. 以上ですべての設定が完了です。カートへの追加、削除、一覧の表示のみ Lambda へ移行されていることを確認できます。本当に動作しているのか気になる方は DynamoDB の「unishop」テーブルの中身にデータが

保存されているかで確認できます(移行前に追加されたカート情報は引き継がれないのでご注意ください)

お疲れさまでした！

削除は以下を行ってください

- Lambda 関数 3 つ
- Refactor Spaces のルート(ソースパス「/」は最後にけしてください)
- Refactor Spaces のサービス(legacy は最後に消してください)
- Refactor Spaces のアプリケーション
- Refactor Spaces の環境
- S3 のバケット 2 つを空にする(バケット自体は CFn の削除時に一緒に削除されます)
- CFn2 つ
- Cloud Watch Logs ロググループ(MonoToMicro-InstaceLogGroup)

Appendix :

今回のハンズオンではカート機能をマイクロサービスに移行しましたが、このアプリケーションには他にもいくつか機能がありますが、その機能たちもマイクロサービスに移植してみたいという方のために Appendix フォルダにいくつかファイルを用意しました。詳細な使い方は省略しますが、今回のハンズオンの手順書の内容を適宜読み替えていただければ一通りお試しください。できるかと思います。余裕のある方はお試しください。

Cloudformation

- MonoToMicroCFDDBAppnedix.yaml : 追加で必要になる Dynamo DB のテーブル、及び IAM ロールを用意しています
 - MasterTable : 商品情報を登録するテーブルです
 - UserTable : ユーザー情報を登録するテーブルです
 - LambdaMasterRole : 商品情報で設定しているテーブルに Lambda でアクセスするときに使用するテーブルです
 - LambdaUserRole : MasterTable で設定しているテーブルに Lambda でアクセスするときに使用するロールです

Dynamo DB

- insert_unicorn.py : Dynamo DB に商品データを登録するときに使用する

コードです。「dynamodb:BatchWriteItem」と「dynamodb:PutItem」の権限を付与した IAM ユーザーを作成してお使いください(IAM ユーザーの作成方法は省略します)

Lambda

- LoginUnicorn : ログイン処理を行う Lambda 関数です
 - 使用する IAM ロール : LambdaUserRole
 - Refactor Spaces でのサービス名 : LoginUnicorn
 - ソースパス : /user/login
- SignupUnicorn : サインアップ処理を行う Lambda 関数です
 - 使用する IAM ロール : LambdaUserRole
 - Refactor Spaces でのサービス名 : SignupUnicorn
 - ソースパス : /user
- GetUnicorn : Dynamo DB に移植した商品データを取得するための Lambda 関数です(「insert_unicorn.py」で Dynamo DB にデータ追加する必要があります)
 - 使用する IAM ロール : LambdaMasterRole
 - Refactor Spaces でのサービス名 : GetUnicorn
 - ソースパス : /unicorns

何でもかんでもマイクロサービスにすればいいわけではありませんが、もし全機能を完全にマイクロサービスにするにはどうすればできるのかを Appendix でなんとなく感じていただけますと幸いです。