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グラフィカル ユーザー インターフェイス, アプリケーション

   自動的に生成された説明
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グラフィカル ユーザー インターフェイス, テキスト, アプリケーション

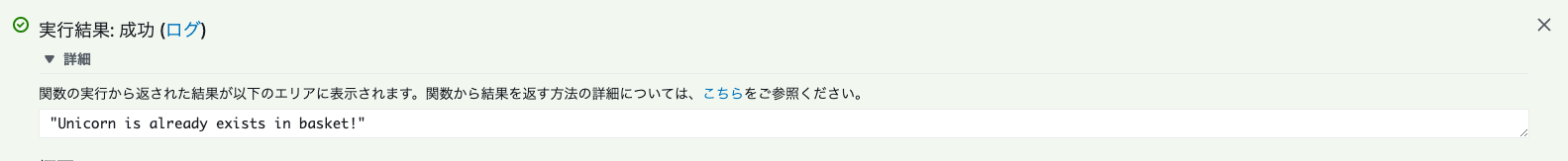
   自動的に生成された説明
4. Lambdaマネジメントコンソールに移動して、「関数の作成」ボタンをクリックしますグラフィカル ユーザー インターフェイス, テキスト, アプリケーション

   自動的に生成された説明
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」に設定しますグラフィカル ユーザー インターフェイス, テキスト, アプリケーション, メール

   自動的に生成された説明
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。グラフィカル ユーザー インターフェイス, テキスト, アプリケーション, メール

   自動的に生成された説明
7. 「関数の作成」ボタンをクリックします
8. 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
9. 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)モニター画面に映るウェブサイトのスクリーンショット

   自動的に生成された説明
10. 「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
11. 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。グラフィカル ユーザー インターフェイス, テキスト, アプリケーション, メール, Teams

    自動的に生成された説明
12. 「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
13. 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

1. Refactor Spacesの左ペインで「サービスを作成」をクリックします
2. 先ほどと同様に環境は「unistore-dev」、アプリケーションは「unistore」を設定しますグラフィカル ユーザー インターフェイス, テキスト, アプリケーション

   自動的に生成された説明
3. サービス名は「AddToCartService」を設定しますグラフィカル ユーザー インターフェイス, アプリケーション

   自動的に生成された説明
4. 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」を選択しますグラフィカル ユーザー インターフェイス, テキスト, アプリケーション, メール

   自動的に生成された説明
5. 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

グラフィカル ユーザー インターフェイス, テキスト, アプリケーション

自動的に生成された説明

1. 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
2. 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

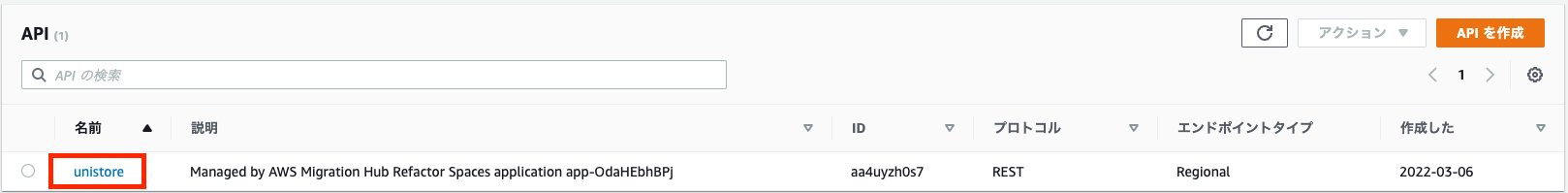
　エンドポイント：Lambda

　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

1. 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
2. APIの「unistore」をクリックします
3. 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

グラフィカル ユーザー インターフェイス

低い精度で自動的に生成された説明

1. 左ペインの「モデル」をクリックして「作成」をクリックしますグラフィカル ユーザー インターフェイス, アプリケーション

   自動的に生成された説明
2. モデル名に「UnicornBasket」と入力して、コンテンツタイプに「application/json」と入力しますグラフィカル ユーザー インターフェイス, アプリケーション

   自動的に生成された説明
3. testcommand.txtのUnicornBasket GetSchemaのJsonをコピーして「モデルの作成」をクリックしますテーブル

   自動的に生成された説明
4. 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

1. 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グラフィカル ユーザー インターフェイス, アプリケーション

   自動的に生成された説明
2. /unicorns/basket/{proxy+}配下のGETをクリックしますグラフィカル ユーザー インターフェイス, アプリケーション

   自動的に生成された説明
3. 「統合リクエスト」をクリックしますグラフィカル ユーザー インターフェイス, アプリケーション

   自動的に生成された説明
4. 「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をクリックしますグラフィカル ユーザー インターフェイス, アプリケーション

   自動的に生成された説明
5. 「/unicorns/basket」のPOSTとDELETEに対しても同様の作業を行います
6. 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」を開きますグラフィカル ユーザー インターフェイス, テキスト, アプリケーション, メール

   自動的に生成された説明
7. 「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グラフィカル ユーザー インターフェイス, テキスト, アプリケーション, メール

   自動的に生成された説明
8. 「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グラフィカル ユーザー インターフェイス, テキスト, アプリケーション, メール

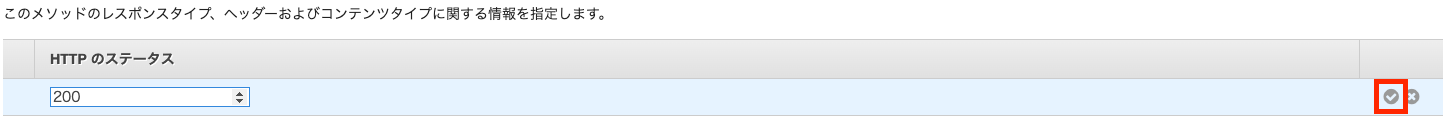
   自動的に生成された説明
9. 「テンプレートの生成」ドロップダウンから「UnicornBasket」を選びますグラフィカル ユーザー インターフェイス, テキスト, アプリケーション

   自動的に生成された説明
10. 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グラフィカル ユーザー インターフェイス, テキスト, アプリケーション

    自動的に生成された説明
11. 「/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！！) グラフィカル ユーザー インターフェイス

    自動的に生成された説明
12. 以下の構成になっていることを確認しますグラフィカル ユーザー インターフェイス

    低い精度で自動的に生成された説明
13. 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」をクリックしますグラフィカル ユーザー インターフェイス, アプリケーション

    自動的に生成された説明
14. 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
15. 「/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
16. 「/unicorns/basket」パスをクリックして「アクション→CORSの有効化」を選択しますグラフィカル ユーザー インターフェイス, アプリケーション

    自動的に生成された説明
17. 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」をクリックしますグラフィカル ユーザー インターフェイス, テキスト, アプリケーション, メール

    自動的に生成された説明
18. 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」をクリックしますグラフィカル ユーザー インターフェイス, テキスト, アプリケーション, メール

    自動的に生成された説明
19. 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グラフィカル ユーザー インターフェイス, テキスト, アプリケーション, メール

    自動的に生成された説明
20. 「/unicorns/basket/{proxy+}」を選択し同様にCORSの有効化を行います
21. ルートパス( / )を選択し、「アクション→APIのデプロイ」を選択しますグラフィカル ユーザー インターフェイス, テキスト, アプリケーション

    自動的に生成された説明
22. デプロイされるステージは「prod」を選択し「デプロイ」をクリックしますグラフィカル ユーザー インターフェイス, テキスト, アプリケーション

    自動的に生成された説明
23. 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宣読み替えていただければ一通りお試しいただけるかと思います。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。