

## はじめに：

Migration Hub Refactor Spaces は AWS 上で動作している Rosati を持つ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へ移行させる際に、ストラングラーフイグパターンによる移行作業を簡素化するために、ネットワーク構成を構築するサービスです。現在 Preview 中で、HTTPS RestAPI のみをサポートし、また移行先は RestAPI の処理単位に対する Lambda のみをサポートしています。

本シナリオは [AWS 公式シナリオ](#) の単純な日本語化バージョンです。サービスのバージョンアップなどで動作しなくなっている場合、英語版を参照ください。その際作成者に一方いただけると幸いです。

## モノリシックアプリケーションの構築

以下の手順では S3 を用いたウェブサイトホスティングを行うため、バケットへのパブリックアクセスが必須となります。ブロック設定がオフとなっていることを確認してください。



1. [MonoToMicroCF.template.txt] を Git からダウンロードします
2. CloudFormation の画面から、[スタックの作成] をおします
3. 先程ダウンロードしたテンプレートファイルをアップロードし、[次へ] をおします

テンプレートの準備  
各スタックはテンプレートに基づきます。テンプレートとは、スタックに含む AWS リソースに関する設定情報を含む JSON または YAML ファイルです。

☒ テンプレートの準備完了
 ☐ サンプルテンプレートを使用
 ☐ デザイナーでテンプレートを作成

---

**テンプレートの指定**  
テンプレートは、スタックのリソースおよびプロパティを表す JSON または YAML ファイルです。

テンプレートソース  
テンプレートを選択すると、保存先となる Amazon S3 URL が生成されます。

☐ Amazon S3 URL
 ☒ テンプレートファイルのアップロード

テンプレートファイルのアップロード

`MonoToMicroCF.template.txt`

JSON または YAML 形式のファイル

4. [MonoToMicro]と名前をつけ、[次へ]をおします
5. 次の画面ではそのまま[次へ]をおし、最後の確認画面で以下にチェックをつけ[スタックの作成]をおします

機能

**① The following resource(s) require capabilities: [AWS::IAM::InstanceProfile, AWS::IAM::Role]**

このテンプレートには、Identity and Access Management (IAM) リソースが含まれています。これらのリソースを個別に作成し、それぞれに最小限必要な権限を与えるかどうか確認してください。さらに、カスタム名が付けられているか確認してください。カスタム名が、ご利用の AWS アカウント内で一意のものであることを確認してください。 [詳細はこちら](#)

☒ AWS CloudFormation によって IAM リソースがカスタム名で作成される場合があることを承認します。

6. 20 分ほど待つと、S3 による HTML のホスティング、EC2 上の Java アプリケーション、RDSMySQL データベースなどが作成されます。
7. 出力タブの PublicDNS の値をコピーしておきます。リソースタブから作成された VPC の ID もコピーします。
8. S3 バケットで[[monotomicro-uibucket-xxxx](#)]を特定しクリックし、[プロパティ]タブをクリックします
9. 画面の一番下の URL をブラウザで開きます

### 静的ウェブサイトホスティング

編集

このバケットを使用して、ウェブサイトをホストしたり、リクエストをリダイレクトしたりします。[詳細](#)

静的ウェブサイトホスティング  
有効

ホスティングタイプ  
バケットホスティング

バケットウェブサイトエンドポイント  
バケットを静的ウェブサイトとして設定すると、そのウェブサイトはバケットの AWS リージョン固有のウェブサイトエンドポイントで使用できます。  
[詳細](#)

<http://monotomicro-uibucket-1wygu7gcorr7m6.s3-website-ap-northeast-1.amazonaws.com>

10. 適当な名前ですignupをし、そのパスワードでLoginを行います。その後適当な商品をカートに入れたりカートから削除したり操作してみてください。

これでモノリスアプリケーションの作成は完了です。Chromeをお使いの方は、デベロッパーツールを開くと、EC2 インスタンスへ通信していることがわかります。

▶ Object	app.js:18
<a href="http://ec2-13-231-236-2.ap-northeast-1.compute.amazonaws.com">http://ec2-13-231-236-2.ap-northeast-1.compute.amazonaws.com</a>	app.js:28
▶ Array(10)	app.js:36
▶ Object	app.js:47
User not logged in	app.js:91
<a href="http://ec2-13-231-236-2.ap-northeast-1.compute.amazonaws.com/user/login">http://ec2-13-231-236-2.ap-northeast-1.compute.amazonaws.com/user/login</a>	app.js:182

## Refactor Spaces の起動：

このハンズオンでは、先に構築したモノリシックアプリケーションから、カートへの商品追加、商品の削除、カートに一時的に保存されている商品一覧の表示、3つの機能([baseUrl]/unicorns/basket)のみをマイクロサービスアーキテクチャ（Lambda へ移行します）

11. マネージメントコンソールで Migration Hub にアクセスします。MigrationHub にはデフォルトリージョンという設定が存在し、AWS アカウント単位でどこか単一のリージョンがメインのダッシュボードを提供するため、別のリージョンでアクセスしても、デフォルトリージョンのダッシュボードが表示されます。ただし Refactor Spaces は別のリージョンでも作業が可能です。
12. 左ペインから[Refactor Spaces]をクリックした後、作業を行いたいリージョンに変更します

移行

アプリケーション

更新

ツール

Refactor Spaces <sup>新</sup>

設定

ヘルプとサポ  
ート

アジアパシフィック (ジャカルタ)	ap-southeast-3
アジアパシフィック (ムンバイ)	ap-south-1
アジアパシフィック (大阪)	ap-northeast-3
アジアパシフィック (ソウル)	ap-northeast-2
アジアパシフィック (シンガポール)	ap-southeast-1
アジアパシフィック (シドニー)	ap-southeast-2
アジアパシフィック (東京)	ap-northeast-1

13. [環境]をクリックし、[環境の作成]をクリックします

< AWS Migration Hub ×  
Refactor Spaces [プレビュー](#)

通知

▼ アプリケーションリファクタリ  
ング

環境

14. 名前を” unistore-dev”と入力して、[次へ]をおします

**環境**

AWS Resource Access Manager、AWS Transit Gateway、および VPC をオーケストレートすることで、AWS アカウント間でネットワークをブリッジし、既存のアプリケーションと新しいサービスが直接通信できるようにします。

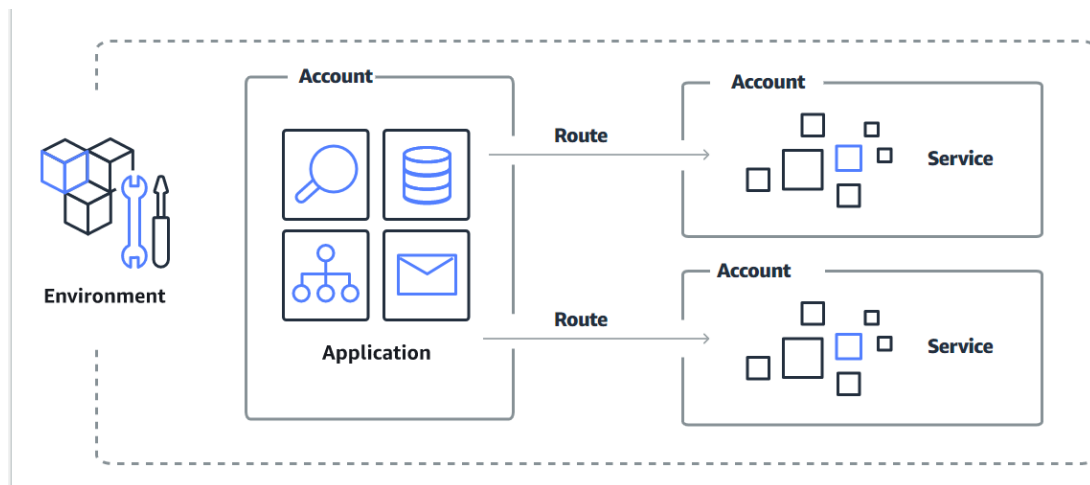
環境名

環境の説明 - オプション

環境の簡単な説明。

画面に概要図が表示されますが、右下がモノリシックなアプリケーション環境

(VPC)、右上がマイクロサービス化された Lambda 環境です。そして左側が、Refactor Spaces が作成するネットワーク環境と API Gateway になります。



15. アプリケーションに”unistore”と入力します

### アプリケーション

名前

16. 先程コピーした VPC の ID を以下のダイアログで選択し、[次へ]をおします。この Step で指定した VPC に対して、Refactor Spaces がプロキシ用ネットワークと API Gateway を構築します。

### プロキシ設定

アプリケーションのプロキシは、既存のアプリケーションから異なる AWS アカウントの新しいサービスへのトラフィックのルーティングを簡素化します。プロキシを提供するために、API Gateway REST API、API Gateway VPC のリンク、ネットワークロードバランサー、およびリソースポリシーを AWS がオーケストレーションします。

プロキシ VPC

アプリケーションのプロキシ VPC は、プロキシから URL エンドポイントを使用して Refactor Spaces サービスにトラフィックをルーティングするために使用されます。

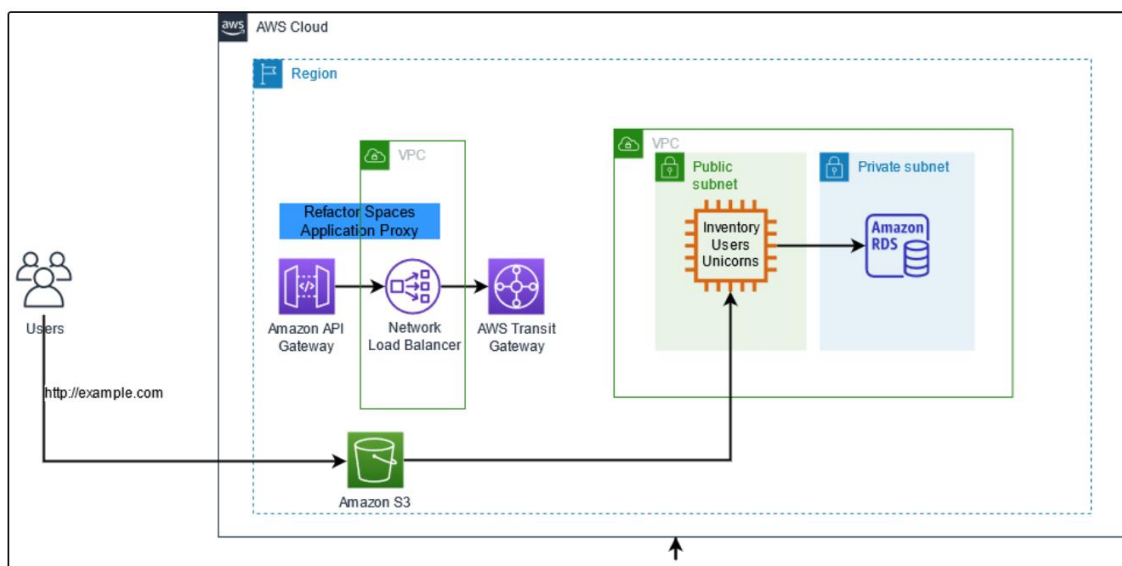
▼

プロキシエンドポイントタイプ

リージョン API は、現在のリージョン (ap-northeast-1) でパブリックにアクセス可能です。プライベート API は VPC からのみアクセスできます。

☒ リージョン別

☐ プライベート



17. 次の画面はデフォルトのまま[次へ]をおします
18. 次の画面で[環境の作成]をおします。5分程度で画面が遷移します。
19. 遷移先の画面でしばらく待ちます。



ヘルスステータスが[正常]になれば設定が完了です。

#### サービスとルートの設定：

いままでの手順でプロキシネットワークが出来上がりましたが、モノリシックアプリケーションをサービスとして登録し、そのルート設定を行うことで、ユーザーからのアクセス（リクエスト）を作成されたネットワークが処理できるようになります。

20. 左ペインから[サービスの作成]をクリックします



21. 今までの手順で作成された、環境とアプリケーションを選択します

22. サービス名に[legacy]と入力します

23. VPC に Java アプリケーションがホスティングされている EC2 の VPCID を指定し、エンドポイントに EC2 のエンドポイントを入力します。ヘルスチェックエンドポイン

トには、以下を入力します。

<http://ec2-XXX-XXX-XXX-XXX.compute-1.amazonaws.com/actuator/health>

注意点：ユーザーが S3 上の HTML を読み込んだのちアクセスする先が現在 EC2 エンドポイントと担っていますが。そのアクセス先を Refactor Spaces のネットワークに変更するための設定です。したがって、ここでの設定は S3 の URL ではなく、EC2 のエンドポイント URL を設定してください。

24. “このサービスをアプリケーション (unistore) のデフォルトルートとして設定します。”にチェックを付けます。これにより、一旦すべての Refactor Spaces へのリクエストが EC2 エンドポイントにルーティングされることとなります。

### 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

④ 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、トラフィックはサービスに送信されます。デフォルトのルートは、すべてのアプリケーショントラフィックをこのサービスに送信します。

☒ このサービスをアプリケーション (unistore) のデフォルトルートとして設定します。

25. [サービスの作成]をおします

26. しばらく待つと以下のようにサービス、ルートともに以下のように表示されれば成功です

サービス (1)

🔄

詳細を表示

削除

サービスを作成

🔍 サービスを検索

< 1 > ⚙

	ID	ステータス	名前	タイプ	エンドポイント
<input type="radio"/>	svc-dbHzks6v2D	🟢 正常	legacy	URL	<div>📄</div> http://ec2-13-231-236-2.ap-northeast-1.compute.amazon

ルート (1)

🔄

削除

ルートを作成

🔍 ルートを検索

< 1 > ⚙

	ID	ステータス	ソースパス	子パス	動詞	タイプ	ターゲット
<input type="radio"/>	rte-zUuOzkrT55	🟢 正常	/	はい	すべてを照合	デフォルト → URL	legacy

27. API Gateway のマネージメントコンソールに移動すると、API が 1 つ作成されていることがわかります。これがすべてのリクエストを一旦モノリシックなアプリケーションヘルーティングするプロキシのエンドポイント (EC2 エンドポイントに代わるもの) になります。



○
unistore

Managed by AWS Migration Hub  
Refactor Spaces application  
app-2B6WHZxNEy

4z9o5r98ej

REST

Regional

2022-01-17

- [ステージ]→[Prod]とクリックしてURLをコピーしておきます
- S3バケットの”**MonoToMicro-uibucket-xxxxx**”から”config.json”をダウンロードしエディタで開きます
- URLをEC2エンドポイントからAPI Gatewayのものに差し替え保存し、S3へ書きアップロードします。その際、必ず以下の設定を行いパブリックアクセスを可能としてください。

● パブリック読み取りアクセス権の付与  
 世界中の誰でも、指定されたオブジェクトにアクセスできます。オブジェクト所有者に読み取りおよび書き込みアクセス権があります。  
[詳細はこちら](#)

⚠
**パブリック読み取りアクセスを許可することは推奨されません**  
 世界中の誰でも、指定されたオブジェクトにアクセスできます。
 [詳細はこちら](#)

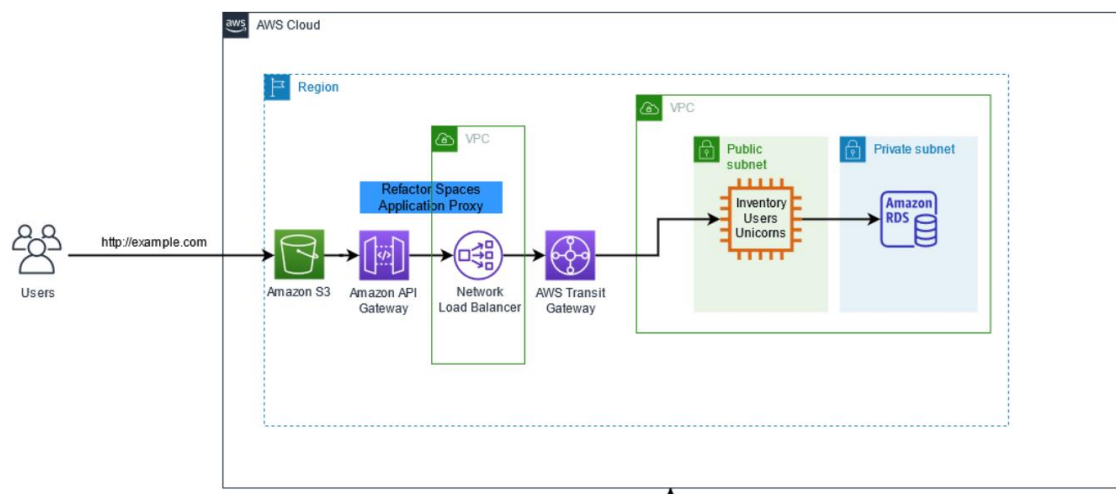
☒ 私は、指定されたオブジェクトへのパブリック読み取りアクセスを付与するリスクについて理解しています。

**▶ プロパティ**  
 ストレージクラス、暗号化設定、タグなどを指定します。

キャンセル

アップロード

31. 以下のような構成に変更されました。



API GatewayがEC2のアプリケーション（HTTPS REST API）に対するプロキシとなっていることに注目してください。

32. 先程テストしたS3上のHTMLにアクセスし、再度正しく動作するか確認してください。Chromeをお使いの方はデベロッパーツールで以下のようにアクセス先がEC2で

はなくなっていることを確認してください。

► Object

app.js:18

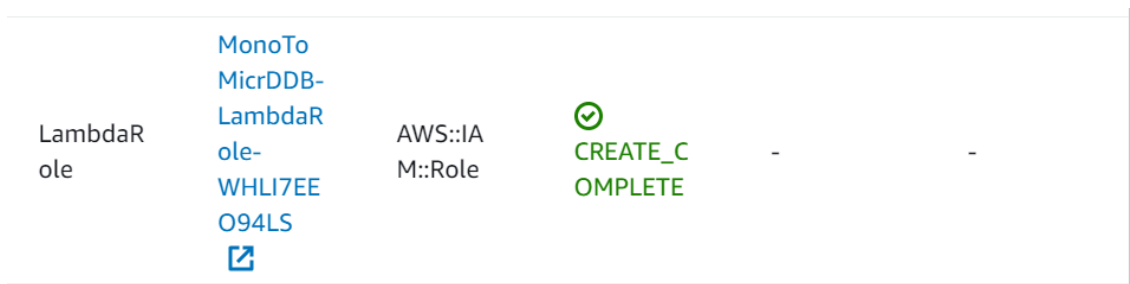
<https://4z9o5r98ej.execute-api.ap-northeast-1.amazonaws.com/prod>

app.js:28

### マイクロサービス環境の構築：

ではここから、カート処理を担うマイクロサービス環境を作成し、順次リクエストを Refactor Spaces が管理する API Gateway 経由でこちらにルーティングする設定を行います。

33. “MonoToMicroDDBCF.yaml”を Git からダウンロードします
34. 先程と同様の手順で CFn にてスタックを作成します。スタックの名前は”MonoToMicrDDB”としてください。最大 5 分程度待ちます。ステータスが”CREATE\_COMPLETE“になったら、念のためマネージメントコンソールで DynamoDB の”unishop”テーブルができていることを確認します。
35. CFn のリソースタブで、作成された IAM Role の名前をコピーしておきます



36. Lambda マネージメントコンソールに移動し、[関数の作成]ボタンをおします
37. 関数名とランタイムを以下のように設定します

関数名

関数の目的を名前として入力します。

AddUnicornToBasket

半角英数字、ハイフン、アンダースコアのみを使用でき、スペースは使用できません。

ランタイム [情報](#)

関数の記述に使用する言語を選択します。コンソールコードエディタは Node.js、Python、および Ruby のみをサポートすることに注意してください。

Java 8 on Amazon Linux 2

38. 先程作成された Lambda 用ロールを設定します。(Lambda 関数が DynamoDB へのアクセス権限を持つためのものです)

▼ デフォルトの実行ロールの変更

**実行ロール**  
関数のアクセス権を定義するロールを選択します。カスタムロールを作成するには、IAM コンソールに移動します。

☐ 基本的な Lambda アクセス権限で新しいロールを作成

☒ 既存のロールを使用する

☐ AWS ポリシーテンプレートから新しいロールを作成

**既存のロール**  
この Lambda 関数で使用するために作成した既存のロールを選択します。このロールには、Amazon CloudWatch Logs にログをアップロードするアクセス許可が必要です。

service-role/MonoToMicroDDB-LambdaRole-WHLI7EE094LS ▼ 

IAM コンソールで [MonoToMicroDDB-LambdaRole-WHLI7EE094LS](#) ロールを表示します。

39. [関数の作成]ボタンをおします
40. S3 バケット”MonoToMicro-assetbucket-xxxxx”から MonoToMicroLambda-0.0.1.jar をダウンロードし適当な個所に保存します
41. Lambda 関数のコードソースから先程ダウンロードした jar をアップロードします

**コードソース** 情報 アップロード元 ▲

① コードエディタは Java 8 on Amazon Linux 2 ランタイムをサポートしていません。

**.zip または jar ファイルをアップロード** ×

① 新しい .zip ファイル/パッケージをアップロードすると、既存のコードが上書きされます。

 **アップロード** MonoToMicroLambda-0.0.1.jar (13.0 MB)

10 MB より大きいファイルの場合は、Amazon S3 を使用したアップロードを検討してください。

キャンセル 保存

42. 以下の注意表記は正常ですので問題ありません。

**コードソース** 情報 アップロード元 ▼

① Lambda 関数「AddUnicornToBasket」のデプロイパッケージが大きすぎて、インラインコード編集を有効にできません。ただし、関数を呼び出すことはできます。

43. ランタイムの[編集]ボタンをおして、以下の値をハンドラとして設定し、[保存]をおします

com.monoToMicro.Lambda.UnicornBasketImpl::addUnicornToBasket

## ランタイム設定を編集

**ランタイム設定** 情報

ランタイム  
関数の記述に使用する言語を選択します。コンソールコードエディタは Node.js、Python、および Ruby のみをサポートすることに注意してください。

Java 8 on Amazon Linux 2 ▼

**④ 使用可能な新しいランタイム** ×  
お使いの関数の言語で、新しいランタイムを使用できます。Java 11 (Corretto)

ハンドラ 情報

com.monoToMicro.Lambda.UnicornBasketImpl::addUnicornToBasket

44. [テスト]タブをクリックして、名前に"test"と入力します。
45. "testcommand.txt"を開いて、[AddUnicornToBasket]の中身を以下にコピペし、[テスト]ボタンをおします
46. 以下の通り成功と表示されれば設定は完了です

**✓ 実行結果: 成功 (ログ)** ×

▼ 詳細

関数の実行から返された結果が以下のエリアに表示されます。関数から結果を返す方法の詳細については、[こちら](#)をご参照ください。

"Added Unicorn to basket"

47. DynamoDB のテーブルの[項目]タブを見ると値が 1 個追加されていることがわかります。

unishop [開じる](#)

概要

**項目**

メトリックス

アラーム

キャパシティー

インデックス

さらに ▼

項目の作成

アクション ▼

設定

リフレッシュ

スキャン: [テーブル] unishop: uuid ↑ 項目 1 ~ 1 を表示中

スキャン ▼

[テーブル] unishop: uuid ▼

+ フィルターの追加

開始

<input type="checkbox"/>	uuid ⓘ	unicorns
<input type="checkbox"/>	4b3fc86b-81d0-4614-920e-8184063acf2d	[{"M": {"uuid": {"S": "16c3e7c0-bba4-11e9-afec-41e09d"

48. 同様の手順でもう 2 つ Lambda 関数を作成してください。

名前 : RemoveUnicornFromBasket

ハンドラ : com.monoToMicro.Lambda.UnicornBasketImpl::removeUnicornFromBasket

名前 : GetUnicornsBasket

ハンドラ : com.monoToMicro.Lambda.UnicornBasketImpl::getUnicornsBasket

テストパラメータ (共通)

“testcommand.txt”の RemoveUnicornFromBasket / GetUnicornsBasket

### 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、APIGateway 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全てが旧モノリシック環境へのルーティングがなされていますが、関数単位 (HTTP REST API) で Lambda ヘルーティングを変更していきます

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

50. 先程と同じ値を[環境][アプリケーション]に指定します

### 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▼

↺

51. [AddToCartService]をサービス名に登録します

### 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

52. 先程と異なり、サービスエンドポイントの設定は[VPC]ではなく[Lambda]を指定し、[AddUnicornToBasket]を指定します

### 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 ▼

🔄

53. ソースパスに[/unicorns/basket]を入力し、”子パスを含める”のチェックを外してください

54. 動詞のドロップダウンから[POST]を選択し、[サービスを作成]をおします

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 ✕

キャンセル サービスを作成

55. 同様に以下の内容でサービスをもう 1 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

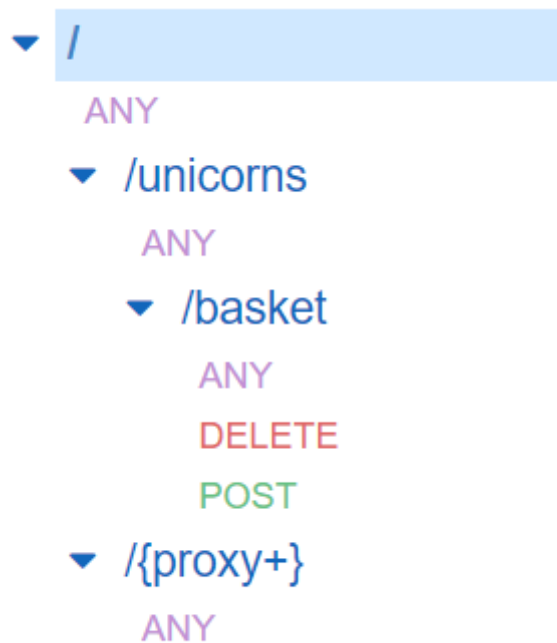
子パスを含める: オフ

動詞: DELETE

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

57. API の“unistore”をクリックします

58. 先程作成した DELETE と POST が作成されていることがわかります



59. 左ペインの[モデル]をクリックし[作成]をおします

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

60. モデル名に[UnicornBasket]と入力し、コンテンツタイプに[application/json]と入力します
61. Testcommand.txt の API Gateway Get Schema の値をコピーして、[モデルのスキーマ]にコピーして[モデルの作成]をおします
62. Refactor Spaces に戻り、先程と同様に 3 回目の[サービスの作成]を行います。以下の値を入力してください。
 

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: オン (今までと異なるので要注意)

動詞: GET
63. マネージメントコンソール API Gateway のリソースにアクセスし、以下の状態になっているか確認をしてください。



API

カスタムドメイン名

VPC リンク

リソース

▼ /

ANY

▼ /unicorns

ANY

▼ /basket

ANY

DELETE

GET

POST

▼ /{proxy+}

ANY

GET

▼ /{proxy+}

ANY

アクション▼

---

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

ずれがある場合、サービスの作成にミスがありますので、サービスを一度デフォルト以外消してやり直してください。

サービス (4)					
<div> <div>サービスを検索</div> <div> <div>刷新</div> <div>詳細を表示</div> <div>削除</div> <div>サービスを作成</div> </div> </div>					
ID	ステータス	名前	タイプ	エンドポイント	
○ <a href="#">svc-dbHzks6v2D</a>	✔ 正常	legacy	URL	<a href="#">http://ec2-13-231-236-2.ap-northeast-1.c...</a>	
○ <a href="#">svc-eE7Bwyu0Ig</a>	✔ 正常	GetCartService	Lambda	<a href="#">GetUnicornsBasket</a>	
○ <a href="#">svc-qSdaUDVDDp</a>	✔ 正常	RemoveCartService	Lambda	<a href="#">RemoveUnicornFromBasket</a>	
○ <a href="#">svc-wnlKTjrRd</a>	✔ 正常	AddToCartService	Lambda	<a href="#">AddUnicornToBasket</a>	

ルート (4)

🔄

削除

ルートを作成

🔍 ルートを検索

< 1 >

⚙️

	ID	ステータス	ソースパス	子パス	動詞	タイプ	ターゲット
<input type="radio"/>	rte-FWnFVpZ4nh	🟢 正常	/unicorns/basket	いいえ	DELETE	パス → Lambda	Re...
<input type="radio"/>	rte-efGak9HQKq	🟢 正常	/unicorns/basket	いいえ	POST	パス → Lambda	Ad...
<input type="radio"/>	rte-yQ7li5L5k5	🟢 正常	/unicorns/basket	はい	GET	パス → Lambda	Ge...
<input type="radio"/>	rte-zUuOzkrT55	🟢 正常	/	はい	すべてを照合	デフォルト → URL	leg...

64. /unicorns/basket/{proxy+}配下の GET をクリックしてください

65. [統合リクエスト]をクリックします

66. [Lambda プロキシ統合の使用]のチェックをはずします

[← 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 ap-northeast-1 

Lambda 関数 GetUnicornsBasket 

67. [/unicorns/basket]の POST と DELETE で同様の作業を行います

68. 再度[/unicorns/basket/{proxy+}/GET]の[統合リクエスト]画面に戻り、画面一番下の [マッピングテンプレート]を開きます

▶ 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 

69. [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]のプラスボタンをおします

70. [application/json]を入力し、チェックをおします

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

71. [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:

1
<div>UnicornBasket ▼</div> <div>メソッドリクエストのパススルー モデル</div> <div>Empty</div> <div>Error</div> <div>UnicornBasket</div>

72. testcommand.txt の[UnicornBasket model]の値をコピーし、[保存]をおします

テンプレートの生成: UnicornBasket ▼

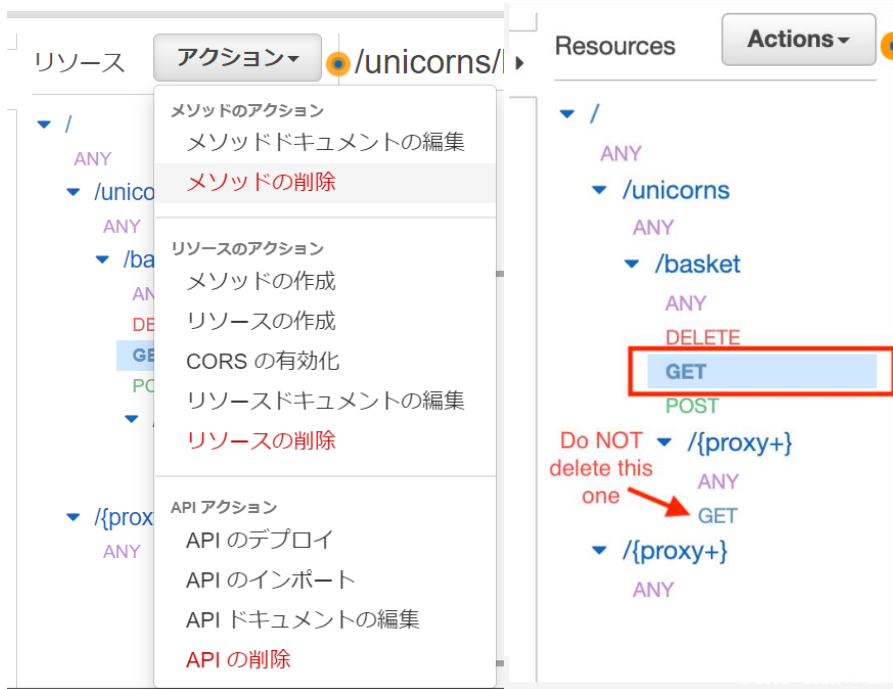
```
1 #set($inputRoot = $input.path('$'))
2 {
3   "uuid" : "$input.params('proxy')"
4 }
```

キャンセル

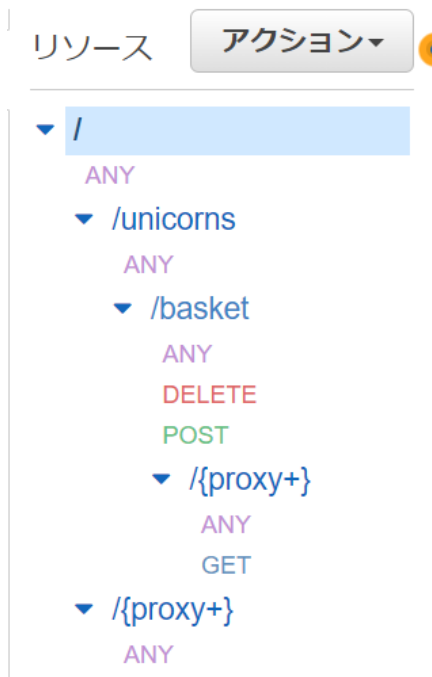
保存

[保存]をおすと、ドロップダウンから[UnicornBasket]の値が空欄に代わりますが正常です。

73. /unicorns/basket 配下の GET を選んで、アクションからメソッドの削除を選びます



74. 以下の状態になります



75. 再度 GET をクリックし、[メソッドレスポンス]をクリックします

76. [レスポンスの追加]をおして“200”と入力しチェックをおします

HTTP のステータス	
200	<input checked="" type="checkbox"/>

メソッドレスポンス

HTTP のステータス: 200

77. POST と DELETE で同じようにメソッドレスポンスを登録します。

78. [basket]を設定して、アクションから CORS の有効化を選択します

リソース

アクション

/unicorns/basket メソッド

▼ /

ANY

▼ /unicorns

ANY

▼ /unicorns/basket

ANY

DELETE

POST

▼ /{proxy}

ANY

リソースのアクション

メソッドの作成

リソースの作成

CORS の有効化

リソースドキュメントの編集

リソースの削除

API アクション

API のデプロイ

API のインポート

API ドキュメントの編集

API の削除

認可 なし

不要

認可 なし

## 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\* '\*' ⓘ

▶ アドバンスト

CORS を有効にして既存の CORS ヘッダーを置換

79. [はい、既存の値を置き換えます]をおします。

メソッド変更の確認

×

次の変更は、このリソースのメソッドに対して行われ、すべての既存の値は置き換えられます。続行してよろしいですか？

- **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
- **Mock** 統合を **OPTIONS** メソッドに追加する
- **200** 統合レスポンスを **OPTIONS** メソッドに追加する
- **Access-Control-Allow-Headers**, **Access-Control-Allow-Methods**, **Access-Control-Allow-Origin** メソッドレスポンスヘッダーを **OPTIONS** メソッドに追加する
- **Access-Control-Allow-Headers**, **Access-Control-Allow-Methods**, **Access-Control-Allow-Origin** 統合レスポンスヘッダーマッピングを **OPTIONS** メソッドに追加する
- **Access-Control-Allow-Origin** メソッドレスポンスヘッダーを **DELETE** メソッドに追加する
- **Access-Control-Allow-Origin** 統合レスポンスヘッダーマッピングを **DELETE** メソッドに追加する
- **Access-Control-Allow-Origin** メソッドレスポンスヘッダーを **POST** メソッドに追加する
- **Access-Control-Allow-Origin** 統合レスポンスヘッダーマッピングを **POST** メソッドに追加する

キャンセル

はい、既存の値を置き換えます

ここまでの設定が正しければ全てにチェックが付きます

## CORS の有効化

- ✓ **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
- ✓ **Mock 統合** を **OPTIONS** メソッドに追加する
- ✓ **200 統合レスポンス** を **OPTIONS** メソッドに追加する
- ✓ **Access-Control-Allow-Headers, Access-Control-Allow-Methods, Access-Control-Allow-Origin** メソッドレスポンスヘッダーを **OPTIONS** メソッドに追加する
- ✓ **Access-Control-Allow-Headers, Access-Control-Allow-Methods, Access-Control-Allow-Origin 統合レスポンスヘッダーマッピング** を **OPTIONS** メソッド に追加する
- ✓ **Access-Control-Allow-Origin** メソッドレスポンスヘッダーを **DELETE** メソッドに追加する
- ✓ **Access-Control-Allow-Origin 統合レスポンスヘッダーマッピング** を **DELETE** メソッドに追加する
- ✓ **Access-Control-Allow-Origin** メソッドレスポンスヘッダーを **POST** メソッドに追加する
- ✓ **Access-Control-Allow-Origin 統合レスポンスヘッダーマッピング** を **POST** メソッドに追加する

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。

80. [/unicorns/basket/{proxy+}]を選択し同様に CORS を選択します

81. ルート (/)を選択し[API のデプロイ]を選びます



82. [prod]を選択し[デプロイ]をおします



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

83. 以上で設定が完了です。カートへの追加、削除、一覧表示のみが Lambda へ移行されています。（このシナリオでは、Lambda 関数は大きい Java で作成されており、初回起動に少し時間がかかることに注意してください。

おつかれしました！

削除は以下を行って下さい

Lambda 関数 3 つ

Refactor Spaces のルート（ / 4 番目に消してください）

Refactor Spaces のサービス（legacy は 4 番目に消してください）

Refactor Spaces のアプリケーション

Refactor Spaces の環境

S3 バケット 2 つを空に（削除は CFn の削除時に行われます）

CFn 2 つ

CloudWatch Logs ロググループ MonoToMicro-InstanceLogGroup-