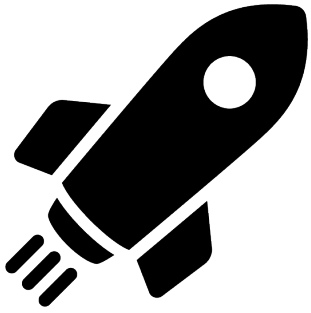
Baikonur

2020 年 10 月 16 日

目次



BAIKONU

第 1 章

Baikonur とは

Baikonur は、 Terraform モジュール 、ナレッジ、インフラ周りのツールなどを含むオープンソースプロジェクトです。

Baikonur は 2018 年に 株式会社サイバーエージェント での社内プロジェクトとしてスタートし、すぐに 50 モジュールを超えました。最も人気のある、ユニークなモジュールをここでオープンソース化からスタートしました。

ここでリリースされているモジュールは、 Terraform Public Module Registry にも公開されています。

第 2 章

目次

2.1 よくある質問

2.1.1 このプロジェクトの名前がなぜ「Baikonur (バイコヌール)」になっていますか?

Baikonur プロジェクトの名前は、人類初の人工衛星の打ち上げと人類初の有人宇宙飛行で知られるカザフスタンにある宇宙港である バイコヌール宇宙基地 に由来しています。そのため、プロジェクトのロゴはロケットです。

そもそも宇宙のテーマを選んだ理由は非常に単純です。 Ansible Galaxy と GitHub Universe に倣いました。

2.1.2 このプロジェクトは Terraform モジュールに限定されていますか?

いいえ。Baikonur プロジェクトは、サイバーエージェント社内で利用されていた Terraform モジュールのオープンソース化のためのプラットフォームとして始まりましたが、最近は Python ライブラリ、CLI ツールをリリースし、インフラ周りのナレッジを公開する準備を進めています。

2.1.3 コントリビュータ/メンテナ/コアメンバーになるには、サイバーエージェントの従業員である必要がありますか?

必要 ありません。

2.2 Baikonur プロジェクトレポジトリー覧

2.2.1 Amazon Kinesis と AWS Lambda を活用した Baikonur ロギング Terraform モジュール

lambda-kinesis-to-fluent

Kinesis Data Streams のデータを Fluent エンドポイントへ転送するためのモジュール

lambda-kinesis-to-s3

Kinesis Data Streams のデータを S3 に保存するためのモジュール

lambda-kinesis-to-es

Kinesis Data Streams のデータを Elasticsearch に格納するためのモジュール

lambda-es-cleaner

Elasticsearch Service の古いインデックスを自動的に削除するためのモジュール

lambda-kinesis-forward

一つの Kinesis Data Streams から別の Kinesis Data Streams にデータを転送、ルーティンするためのモジュール

2.2.2 eden: Amazon ECS Dynamic Environment Manager (Amazon ECS 動的環境マネージャ)

aws-eden-cli

eden CLI

lambda-eden-api

eden API

2.2.3 ツールとライブラリ

amazon_kinesis_utils

Amazon Kinesis でのデータ処理に役立つ Python ライブラリ。 **ロギングモジュール** で使われている。

aws-eden-core

eden **モジュール** 用の内部ライブラリ。

2.2.4 その他の AWS Terraform モジュール

iam-nofile

AWS IAM ロールの作成を容易にするためのモジュール。インライン（ヒアドキュメント）構文でロールポリシードキュメントを記述できるため、変数を使用するために ‘テンプレートレンダリング <template rendering>‘_を使用する必要はありません。

fargate-scheduled-task

CloudWatch イベントを使用して、ECS スケジュールタスクを簡単に作成するためのモジュール (バッチ実行などのような一定のスケジュールで実行されるアプリケーション向け)

2.3 ナレッジベース

工事中。

2.4 Amazon Kinesis と AWS Lambda を活用した Baikonur ロギング

2.4.1 前提条件

なぜ Kinesis を活用するのか?

Kinesis Data Streams は、ストリーミングデータを扱うための高スループット、低レイテンシのフルマネージドサービスです。ストリームは複数のシャードから構築され、一つのシャードは、 1 MB/秒の読み取りと 2 MB/秒の書き込みキャパシティを提供します 。 Kinesis Data Stream に書き込まれたデータは、データ保持期間パラメーターで指定された期間保存されます。したがって、キャパシティプランニングでは、格納、取り出しのピークスループットさえ知っていれば十分です。

注釈: Kinesis Data Streams にはオートスケーリング機能がありません。 amazon-kinesis-scaling-utils を使用すると、Kinesis Data Stream のシャード数を自動的に変更することができます。

なぜ Kinesis 上のデータを Lambda で処理するのか?

Kinesis Data Streams 上のデータを (イベントソースマッピングを用いて) Lambda 関数で処理することで実装コストと管理コストを抑えることができます:

1. Lambda では、サーバーや OS などを管理する必要はありません。

2. Kinesis Data Streams からデータを読み取るためのロジックを実装する必要はありません。データのバッチは、Lambda 関数の実行時に ‘‘event‘‘オブジェクトで渡されます。

3. イベントソースマッピングでは、Kinesis Data Stream 上の処理済みデータの位置が自動的に管理されます。各シャードでどの位置までのデータが正常に処理されたかが記録されます。Lambda の実行ががエラーなしで完了した場合にのみ、位置が更新されます。

3. Kinesis Data Stream にマッピングされた Lambda 関数がエラーで終了した場合、実行が成功するか、データの期限が切れるま Lambda 関数が同じバッチで再度実行されます。データは保持期間が超過したときにのみ Kinesis Data Stream から削除されます。

    - したがって、再試行ロジックを実装する必要はありません。ただし、同じバッチで Lambda が一定回数失敗し続けている場合、そのバッチをキュー (または S3) に保存し、データ処理パイプラインが停止しないようにエラーなしで終了することをお勧めします。

なぜロギングに Kinesis と Lambda を活用するか?

Kinesis と Lambda の連携により、ログ処理のコントロールを維持しながら、自前のロギングクラスターをマネージドサービスに置き換えることができます。

Kinesis と Lambda を使うとで、モジュール式で拡張可能な、スケーラブルなロギングアーキテクチャが実現できます。ログ転送の信頼性を向上するメリットもあります。データが Kinesis Data Stream に正常に書き込まれれば、欠損することはありません。

2.4.2 共通スキーマ要件

「Baikonur ログ周り構成」とは、Kinesis Data Streams と次の Baikonur ログ周り Lambda モジュールを 1 つ以上の組み合わせを使っているものを指す。

- terraform-aws-lambda-kinesis-forward
- terraform-aws-lambda-kinesis-to-es
- terraform-aws-lambda-kinesis-to-s3
- terraform-aws-lambda-kinesis-to-fluent

これらのモジュールには、次の共通スキーマ要件があります。

- すべてのデータは JSON である必要があります。ルート要素の型は `object` でなければなりません。
- すべてのデータには、次のキーが含まれている必要があります。
    - データ種類識別子 (デフォルトのキー名: `log_type`)
    - 一意の識別子。例: `uuid.uuid4()` (デフォルトのキー名: `log_id`)

  - dateutil でパース可能なタイムスタンプ (デフォルトのキー名: `time`)

注釈: すべてのキー名はカスタマイズ可能です。

共通スキーマ要件により、次のことが可能になります:

1. より簡単なパース
2. 異なる Lambda モジュール間の互換性の向上
   - 異なるモジュールを同じ Kinesis Data Stream に接続することができます。
3. 共通スキーマ要件のキーを使った機能を実装することができます:
   - 最も重要な機能の 1 つは、データ種類識別子の値に基づいたログのフィルタリング機能です。
   - terraform-aws-lambda-kinesis-to-s3 では、ファイル名の一意性を保証するために log_id が必要で、日付でログを仕分けるためにタイムスタンプフィールドを必要です。
   - terraform-aws-lambda-kinesis-to-es では、Elasticsearch で日別のインデックスを作成するためにタイムスタンプが必要です (例: index-20200314)。

注釈: データが共通スキーマ要件を満たしている限り、このページで説明されているアーキテクチャとモジュールは、高速で信頼性を担保する必要があるあらゆるデータ転送に適用できます。例えば、マイクロサービス間通信に活用できます。

2.4.3 アーキテクチャ例

1 つストリーム - 1 つの転送先

宛先ごとに Kinesis Data Stream を作成します。たとえば、すべてのログデータを S3 に保存し、一部のみを Elasticsearch に保存する場合は、2 つのストリームを作成し、terraform-aws-lambda-kinesis-to-s3 および terraform-aws-lambda-kinesis-to-es モジュールをデプロイしてマップします。それらをそれぞれのストリームに:

```
   Kinesis API
       v
[App] ---> [KDS "s3"] ---> [kinesis-to-s3] ---> S3
     \
      ---> [KDS "es"] ---> [kinesis-to-es] ---> ES
```

S3 と Elasticsearch の両方にログを保存する場合は、両方のストリームにデータを書き込みます。

1 つのストリーム - 複数の転送先

上記の例では、Elasticsearch にログを保存するためには、両方のストリームに同じデータを書き込む必要があります。 Elasticsearch 用のストリームに terraform-aws-lambda-kinesis-to-s3 を追加することで、重複格納をなくすことができます:

```
   Kinesis API
       v
[App] ---> [KDS "s3"] ---> [kinesis-to-s3] ---> S3
     \
      ---> [KDS "es"] ---> [kinesis-to-es] ---> ES
                      \
                       ---> [kinesis-to-s3] ---> S3
```

これで、各ログイベント複数回格納されることはありません。

Kinesis ルーティングパターン

ルータと呼ばれる一つの Kinesis のストリームにデータを書き込みます。さらに、転送先ごとに出力ストリームを作成します。転送モジュール (terraform-aws-lambda-kinesis-forward) を追加しホワイトリスを設定することで、出版-購読型モデル に類似したアーキテクチャを作成できます。ここで、トピックはデータ種類識別子であり、出力ストリームは購読グループを表します:

```
   Kinesis API
       v
[App] ---> [KDS "router"] ---> [kinesis-forward] ---> [KDS "A"]
                          \
                           ---> [kinesis-forward] ---> [KDS "B"]
                           \
                            --> [kinesis-forward] ---> [KDS "C"]
```

このパターンはマイクロサービス間の通信にも活用できます。

各出力ストリームには、それぞれの Lambda モジュールの組み合わせや購読者を追加することが可能です:

```
   Kinesis API
       v
[App] ---> [KDS "router"] ---> [kinesis-forward] ---> [KDS "A"] ---> [S3]
                          \
                           ---> [kinesis-forward] ---> [KDS "B"] ---> [ES]
                           \
                            --> [kinesis-forward] ---> [KDS "C"] <--- [External␣
↪subscriber]
```

CloudWatch Logs サブスクリプションフィルタを活用した Kinesis ルーティングパターン

Kinesis **ルーティングパターン** をさらに拡張します。「ルーター」ストリームへのデータ格納を CloudWatch Logs サブスクリプションフィルタで行うよう変更します。これで、すでにログを CloudWatch に出力しているユースケースに置いて、アプリケーションでの PutRecord/PutRecords の実装が必要がなくなります。たとえば、ECS で awslogs ロギングドライバーを使用しているユースケースでは、出力先ロググループにサブスクリプションフィルタを設定すると、ログ周りアーキテクチャが次のようになります:

```
 stdout->awslogs      Subscription filter
       v                       v
[App] ---> [CloudWatch Logs] ---> [KDS "router"] ---> [kinesis-forward] ---> [KDS "A"]␣
↪---> [S3]
                                                  \
                                                   ---> [kinesis-forward] ---> [KDS "B"]␣
↪---> [ES]
                                                    \
                                                     --> [kinesis-forward] ---> [KDS "C"]
↪<--- [External subscriber]
```

2.5 eden: Amazon ECS Dynamic Environment Manager (Amazon ECS 動的環境マネージャ)

Amazon ECS を使った環境を簡単に複製するためのツールです。 eden に参照するための Amazon ECS Service を指定すると、eden がそれを複製します。

eden は高速です。作成/削除コマンドの実行は 5 秒以内に完了します。

2.5.1 なぜ?

eden は、多くのプレビュー開発環境が必要だがデータベースやその他のリソースが共有可能なユースケースのために作られています。

2.5.2 どうやって?

eden は、参照 ECS サービスからタスク定義、ECS サービス、およびターゲットグループを複製します。

eden は、すべてのクローンされたサービスに対して 1 つの共通 ALB を使用することでコストと実行時間を削減します（ALB の作成には最大 5 分かかる場合があります)。 eden はターゲットグループを作成し、サービスを複製します。作成したターゲットグループを共通の ALB にアタッチし、共通の ALB を指す Route 53 A ALIAS レコードを作成します。

リソース作成順序

1. ECS タスク定義
   - 参照サービスから複製
2. ALB ターゲットグループ
   - 設定は、参照サービスに接続されたターゲットグループからコピーされます
3. ECS サービス
   - 参照サービスと同じクラスターに作成
4. ALB リスナールール
   - ホストヘッダールール
5. Route 53 A ALIAS レコード
   - 共通 ALB を指す
6. Environments JSON ファイルへの追加

注釈: リソースの削除は逆の順序で実行されます。作成と削除のいずれもの実行が 5 秒以内に完了します。

2.5.3 前提条件

1. S3 バケット内の Environments JSON ファイル
   - ファイル構成や詳細は **こちら**<*eden_envs_json_*> で説明されています。
2. ターゲットグループがアタッチされた参照先となる ECS サービス
3. eden が管理するサービスのための共通 ALB
   - ホストヘッダーリスナールールを用いて全ての環境によって再利用されるます
   - 参照サービスが使用するものとは別のもの
   - HTTPS リスナーが必要です
   - リスナーには、ターゲットダイナミックゾーンのワイルドカード証明書が必要です
4. 単純な ALB の利用
   - 複数のパスルールを利用しない、など
   - 1 つの ECS サービスごとに 1 つの ALB

2.5.4 準備

工事中

2.5.5 Environments JSON ファイル

Environments JSON ファイルは次の目的で使用されます:

1. 存在する環境とそのエンドポイントの確認
2. クライアントアプリケーションに利用可能な環境を知らせる

Environments JSON ファイルの例:

```
{
    "environments": [
        {
            "env": "dev",
            "name": "dev-dynamic-test",
            "api_endpoint": "api-test.dev.example.com"
        }
    ]
}
```

上記の例では、 `config_update_key = api_endpoint` を前提としています。

同じ名前で複数の環境を作成できます。単一の eden 環境内に複数のエンドポイントを作れるようにするために、`config_update_key` 設定が異なるプロファイルを使用すればよいです。たとえば、API、管理ツール、およびフロントエンドサービスを単一の環境として作成したいというユースケースがあるとします。

Let's say we have three profiles, `api`, `admin`, and `frontend`. These profiles are pre-configured with `config_update_key` equal to `api_endpoint`, `admin_endpoint`, `frontend_endpoint` respectively.

```
$ eden create -p api --name foo --image-uri xxxxxxxxxx.dkr.ecr.ap-northeast-1.
↪amazonaws.com/api:latest
$ eden create -p admin --name foo --image-uri xxxxxxxxxx.dkr.ecr.ap-northeast-1.
↪amazonaws.com/admin:latest
$ eden create -p frontend --name foo --image-uri xxxxxxxxxx.dkr.ecr.ap-northeast-1.
↪amazonaws.com/frontend:latest
```

このとき、Environments JSON ファイルは以下のようになります:

```
{
    "environments": [
        {
            "env": "dev",
```

(次のページに続く)

(前のページからの続き)

```
            "name": "dev-dynamic-test",
            "api_endpoint": "api-test.dev.example.com",
            "admin_endpoint": "admin-test.dev.example.com",
            "frontend_endpoint": "test.dev.example.com"
        }
    ]
}
```

Environments JSON ファイル内では、単一の JSON オブジェクトに複数のエンドポイントが存在します。このオブジェクトにある最後のエンドポイントが削除されると、このオブジェクトが Environments JSON ファイルから削除されます。

警告: 上記の例のように単一の環境で複数のエンドポイントを扱う場合、エンドポイントの作成/削除の時間差により、環境が不完全になる可能性があることに注意してください (必要なすべてのエンドポイントが揃っていない環境が存在しうる)。

2.5.6 CLI と API

eden は CLI と API として利用できます。

まず、eden CLI を試すことをお勧めします。eden を CI/CD パイプラインに追加する準備ができたら、eden API の利用を推奨します。API のための **プロファイル** のプッシュには CLI が必要であることに注意してください。

eden CLI のインストール

```
$ pip3 install aws-eden-cli

$ eden -h
usage: eden [-h] {create,delete,ls,config} ...

ECS Dynamic Environment Manager. Clone Amazon ECS environments easily.

positional arguments:
  {create,delete,ls,config}
    create              Create environment or deploy to existent
    delete              Delete environment
    ls                  List existing environments
    config              Configure eden

optional arguments:
  -h, --help            show this help message and exit
```

ヒント: サブコマンドでも -h を使用できます。

```
$ eden config -h
 usage: eden config [-h] {setup,check,push,remote-rm} ...

positional arguments:
  {setup,check,push,remote-rm}
    setup               Setup profiles for other commands
    check               Check configuration file integrity
    push                Push local profile to DynamoDB for use by eden API
    remote-rm           Delete remote profile from DynamoDB

optional arguments:
  -h, --help            show this help message and exit

$ eden config push -h
usage: eden config push [-h] [-p PROFILE] [-c CONFIG_PATH] [-v]
                        [--remote-table-name REMOTE_TABLE_NAME]

optional arguments:
  -h, --help            show this help message and exit
  -p PROFILE, --profile PROFILE
                        profile name in eden configuration file
  -c CONFIG_PATH, --config-path CONFIG_PATH
                        eden configuration file path
  -v, --verbose
  --remote-table-name REMOTE_TABLE_NAME
                        profile name in eden configuration file
```

eden CLI の設定

まず、プロファイルを作成してみましょう。プロファイルを利用すれば、コマンドを実行する際に毎回すべてのパラメーターを指定する必要がなくなり、プロファイル名の指定のみで十分です。

```
$ eden config setup --endpoint-s3-bucket-name servicename-config
$ eden config setup --endpoint-s3-key endpoints.json
$ eden config setup --endpoint-name-prefix servicename-dev
$ eden config setup --endpoint-update-key api_endpoint
$ eden config setup --endpoint-env-type dev
$ eden config setup --domain-name-prefix api
$ eden config setup --dynamic-zone-id Zxxxxxxxxxxx
$ eden config setup --master-alb-arn arn:aws:elasticloadbalancing:ap-northeast-
↪1:xxxxxxxxxxx:loadbalancer/app/dev-alb-api-dynamic/xxxxxxxxxx
$ eden config setup --name-prefix dev-dynamic
$ eden config setup --reference-service-arn arn:aws:ecs:ap-northeast-
↪1:xxxxxxxxxxx:service/dev/dev01-api
$ eden config setup --target-cluster dev
```

設定は ~/.eden/config に保存されます。上記のコマンドで default というプロファイルが作成されました:

```
$ cat ~/.eden/config
[api]
name_prefix = dev-dynamic
reference_service_arn = arn:aws:ecs:ap-northeast-1:xxxxxxxxxxxx:service/dev/dev01-api
target_cluster = dev
domain_name_prefix = api
master_alb_arn = arn:aws:elasticloadbalancing:ap-northeast-1:xxxxxxxxxxxx:loadbalancer/
↪app/dev-alb-api-dynamic/xxxxxxxxxx
dynamic_zone_name = dev.example.com.
dynamic_zone_id = Zxxxxxxxxxxxx
config_bucket_name = servicename-config
config_bucket_key = endpoints.json
config_update_key = api_endpoint
config_env_type = dev
config_name_prefix = servicename-dev
target_container_name = api
```

設定ファイルの整合性をチェックしましょう:

```
$ eden config check
No errors found
```

eden のプロファイル

複数のプロファイルを使用できます。設定で複数のプロファイルを作成し、コマンドを実行する際にの -p **プロファイル名** で使用するプロファイルを指定できます。

```
$ eden config check -p api
No errors found
```

ローカルプロファイルを DynamoDB に保存することで、eden API でそのプロファイルが使用できるようになります。

```
$ eden config push -p api
Waiting for table creation...
Successfully pushed profile api to DynamoDB
```

注釈: eden のプロファイルを保存するための DynamoDB テーブルが存在しない場合、eden CLI がテーブルを自動的に作成します

同じコマンドで既存のプロファイルが上書きされます (既存のプロファイルにプッシュすると上書きされます):

```
$ eden config push -p api
Successfully pushed profile api to DynamoDB table eden
```

remote-rm コマンドでリモートのプロファイルを削除することができます:

```
$ eden config remote-rm -p api
Successfully removed profile api from DynamoDB table eden
```

コマンドの実行

環境の作成:

```
$ eden create -p api --name foo --image-uri xxxxxxxxxx.dkr.ecr.ap-northeast-1.
↪amazonaws.com/api:latest
Checking if image xxxxxxxxxx.dkr.ecr.ap-northeast-1.amazonaws.com/api:latest exists
Image exists
Retrieved reference service arn:aws:ecs:ap-northeast-1:xxxxxxxxxx:service/dev/api
Retrieved reference task definition from arn:aws:ecs:ap-northeast-1:xxxxxxxxxx:task-
↪definition/api:20
Registered new task definition: arn:aws:ecs:ap-northeast-1:xxxxxxxxxx:task-definition/
↪dev-dynamic-api-foo:1
Registered new task definition: arn:aws:ecs:ap-northeast-1:xxxxxxxxxx:task-definition/
↪dev-dynamic-api-foo:1
Retrieved reference target group: arn:aws:elasticloadbalancing:ap-northeast-
↪1:xxxxxxxxxx:targetgroup/api/xxxxxxxxxxx
Existing target group dev-dynamic-api-foo not found, will create new
Created target group arn:aws:elasticloadbalancing:ap-northeast-
↪1:xxxxxxxxxx:targetgroup/dev-dynamic-api-foo/xxxxxxxxxxx
ELBv2 listener rule for target group arn:aws:elasticloadbalancing:ap-northeast-
↪1:xxxxxxxxxx:targetgroup/dev-dynamic-api-foo/xxxxxxxxxxx and host api-foo.dev.
↪example.com does not exist, will create new listener rule
ECS Service dev-dynamic-api-foo does not exist, will create new service
Checking if record api-foo.dev.example.com. exists in zone Zxxxxxxxx
Successfully created CNAME: api-foo.dev.example.com -> dev-alb-api-dynamic-297517510.
↪ap-northeast-1.elb.amazonaws.com
Updating config file s3://example-com-config/endpoints.json, environment example-api-
↪foo: nodeDomain -> api-foo.dev.example.com
Existing environment not found, adding new
Successfully updated config file
Successfully finished creating environment dev-dynamic-api-foo
```

注釈: create、delete コマンドは、リモートステート (DynamoDB テーブル) を更新します。`eden config push` と同様に、テーブルが存在しない場合、テーブルが自動的に作成されます。

作成の確認:

```
$ eden ls
Profile api:
dev-dynamic-api-foo api-foo.dev.example.com (last updated: 2019-11-20T19:44:10.179760)
```

注釈: この一覧は、 *Environments JSON* ファイルではなく、リモートステートの DynamoDB テーブルから生成されます。最後に更新されたタイムスタンプは、create での環境の新規作成、すでに存在する環境へのデプロイで更新されます。

環境を削除し、削除を確認します:

```
$ eden delete -p api --name foo
Updating config file s3://example-com-config/endpoints.json, delete environment␣
↪example-api-foo: nodeDomain -> api-foo.dev.example.com
Existing environment found, and the only optional key is nodeDomain,deleting␣
↪environment
Successfully updated config file
Checking if record api-foo.dev.example.com. exists in zone Zxxxxxxxxx
Found existing record api-foo.dev.example.com. in zone Zxxxxxxxxx
Successfully removed CNAME record api-foo.dev.example.com
ECS Service dev-dynamic-api-foo exists, will delete
Successfully deleted service dev-dynamic-api-foo from cluster dev
ELBv2 listener rule for target group arn:aws:elasticloadbalancing:ap-northeast-
↪1:xxxxxxxxxx:targetgroup/dev-dynamic-api-foo/xxxxxxxxxxxx and host api-foo.dev.
↪example.com found, will delete
Deleted target group arn:aws:elasticloadbalancing:ap-northeast-
↪1:xxxxxxxxxx:targetgroup/dev-dynamic-api-foo/xxxxxxxxxxxx
Deleted all task definitions for family: dev-dynamic-api-foo, 1 tasks deleted total
Successfully finished deleting environment dev-dynamic-api-foo

$ eden ls
No environments available
```

2.5.7 API への移行

CLI と API の両方が DynamoDB テーブルでステートを管理します。このテーブルは CLI によってのみ作成されます。また、API はリモートステートに格納されたプロファイルのみを使用できます。 API を初めて実行する前に、 `eden config --push` を実行して、API のためのプロファイルをプッシュしてください。Terraform で API を構築する際に、リモートステートテーブルが存在しない場合、terraform apply の実行が失敗します。

API 内部

eden API は以下の要素で構成されます:

1. Lambda 関数 (API 本体)
2. API Gateway と API を保護するための API キー
3. ステート管理用の DynamoDB テーブル
    - デフォルトのテーブル名は「eden」です。

Terraform を使用した eden API の構築

```
module "eden_api" {
  source  = "baikonur-oss/lambda-eden-api/aws"
  version = "0.2.0"

  lambda_package_url = "https://github.com/baikonur-oss/terraform-aws-lambda-eden-api/
↪releases/download/v0.2.0/lambda_package.zip"
  name               = "eden"

  # eden API Gateway variables
  api_acm_certificate_arn = "${data.aws_acm_certificate.wildcard.arn}"
  api_domain_name         = "${var.env}-eden.${data.aws_route53_zone.main.name}"
  api_zone_id             = "${data.aws_route53_zone.main.zone_id}"

  endpoints_bucket_name = "somebucket"

  dynamic_zone_id = "${data.aws_route53_zone.dynamic.zone_id}"
}
```

警告: ステート管理のための DynamoDB テーブルは eden CLI によって作成されます。`terraform apply` を実行する前に、一度 `eden config --push` を実行してください。

複数のプロファイルを使うことで、1 つのアカウント/リージョンに対して 1 つの eden API で十分です。詳しくは **プロファイル** セクションを参照してください。

eden API のコマンド

eden API には、create と delete という 2 つの API コマンドのみあります。

GET /api/v1/create

必須クエリパラメータ:

- name: 環境名

- image_uri: デプロイする ECR イメージの URI。すでに ECR にプッシュされ、同じアカウントにある必要があります (eden API は、デプロイ前にイメージ URI が有効であるかを確認します)

任意のクエリパラメータ:

- profile: eden のリモートプロファイルの指定 (デフォルト値は `default`)。プロファイルには、コマンドを実行するために必要なすべての設定が含まれています。リモートプロファイルは `eden config --push` コマンドで作成できます (詳細は **こちら** を参照)。

GET /api/v1/delete

必須クエリパラメータ:

- name: 環境名

任意のクエリパラメータ:

- profile: eden のリモートプロファイルの指定 (デフォルト値は `default`)。プロファイルには、コマンドを実行するために必要なすべての設定が含まれています。リモートプロファイルは `eden config --push` コマンドで作成できます (詳細は **こちら** を参照)。

eden API キー

eden API Terraform モジュールは、1 つの API キーを作成します。API Gateway コンソールから API キーが確認できます。

API にアクセスするには、このキーを指定しなければなりません。

API キーを HTTP ヘッダーとして指定する必要があります:

```
x-api-key: YOURAPIKEY
```

API 使用例

`api` という名前のリモートプロファイルを使って create API を実行してみましょう:

```
curl https://eden.example.com/api/v1/create?name=test-create&image_uri=xxxxxxxxxxxx.
↪dkr.ecr.ap-northeast-1.amazonaws.com/servicename-api-dev:latest&profile=api -H "x-
↪api-key:YOURAPIKEY"
```

API の Lambda 関数が出力したログを見てみましょう:

```
2019-04-08T20:32:05.151Z INFO     [main.py:check_cirn:382] Checking if image␣
↪xxxxxxxxxxxx.dkr.ecr.ap-northeast-1.amazonaws.com/servicename-api-dev:latest exists
2019-04-08T20:32:05.270Z INFO     [main.py:check_cirn:401] Image exists
```

(次のページに続く)

(前のページからの続き)

```
2019-04-08T20:32:05.446Z INFO     [main.py:create_env:509] Retrieved reference service␣
↪arn:aws:ecs:ap-northeast-1:xxxxxxxxxxxx:service/dev/dev01-api
2019-04-08T20:32:05.484Z INFO     [main.py:create_task_definition:58] Retrieved␣
↪reference task definition from arn:aws:ecs:ap-northeast-1:xxxxxxxxxxxx:task-
↪definition/dev01-api:15
2019-04-08T20:32:05.557Z INFO     [main.py:create_task_definition:96] Registered new␣
↪task definition: arn:aws:ecs:ap-northeast-1:xxxxxxxxxxxx:task-definition/dev-dynamic-
↪test-create:1
2019-04-08T20:32:05.584Z INFO     [main.py:create_target_group:112] Retrieved␣
↪reference target group: arn:aws:elasticloadbalancing:ap-northeast-
↪1:xxxxxxxxxxxx:targetgroup/dev01-api/9c68a5f91f34d9a4
2019-04-08T20:32:05.611Z INFO     [main.py:create_target_group:125] Existing target␣
↪group dev-dynamic-test-create not found, will create new
2019-04-08T20:32:06.247Z INFO     [main.py:create_target_group:144] Created target␣
↪group
2019-04-08T20:32:06.310Z INFO     [main.py:create_alb_host_listener_rule:355] ELBv2␣
↪listener rule for target group arn:aws:elasticloadbalancing:ap-northeast-
↪1:xxxxxxxxxxxx:targetgroup/dev-dynamic-test-create/b6918e6e5f10389d and host api-
↪test.dev.example.com does not exist, will create new listener rule
2019-04-08T20:32:06.361Z INFO     [main.py:create_env:554] ECS Service dev-dynamic-
↪test-create does not exist, will create new service
2019-04-08T20:32:07.672Z INFO     [main.py:check_record:414] Checking if record api-
↪test.dev.example.com. exists in zone Zxxxxxxxxxxx
2019-04-08T20:32:08.133Z INFO     [main.py:create_cname_record:477] Successfully␣
↪created ALIAS: api-test.dev.example.com -> dev-alb-api-dynamic-xxxxxxxxx.ap-
↪northeast-1.elb.amazonaws.com
2019-04-08T20:32:08.134Z INFO     [main.py:create_env:573] Successfully finished␣
↪creating environment dev-dynamic-test-create
```

環境のステートがリモートの DynamoDB テーブルで管理されているため、eden CLI で環境の作成を確認できます:

```
$ eden ls
Profile api:
dev-dynamic-test-create api-test.dev.example.com (last updated: 2019-04-08T20:32:08.
↪134469)
```

次のコマンドを実行して、この環境を削除しましょう:

```
curl https://eden.example.com/api/v1/delete?name=test&profile=api -H "x-api-
↪key:YOURAPIKEY"
```

API Lambda ログは次のようになります:

```
2019-04-10T23:11:38.515Z INFO     [main.py:check_record:495] Checking if record api-
↪test.dev.example.com. exists in zone Zxxxxxxxxxxx
2019-04-10T23:11:38.752Z INFO     [main.py:check_record:506] Found existing record api-
↪test.dev.example.com. in zone Zxxxxxxxxxxx
```

(次のページに続く)

(前のページからの続き)

```
2019-04-10T23:11:38.996Z INFO     [main.py:delete_cname_record:596] Successfully␣
↪removed ALIAS record api-test.dev.example.com
2019-04-10T23:11:39.245Z INFO     [main.py:delete_env:665] ECS Service dev-dynamic-
↪test exists, will delete
2019-04-10T23:11:39.401Z INFO     [main.py:delete_env:670] Successfully deleted␣
↪service dev-dynamic-test from cluster dev
2019-04-10T23:11:39.573Z INFO     [main.py:delete_alb_host_listener_rule:397] ELBv2␣
↪listener rule for target group arn:aws:elasticloadbalancing:ap-northeast-
↪1:xxxxxxxxxxxx:targetgroup/dev-dynamic-test/xxxxxxxx and host api-test.dev.example.
↪com found, will delete
2019-04-10T23:11:40.483Z INFO     [main.py:delete_env:697] Deleted all task␣
↪definitions for family: dev-dynamic-test, 5 tasks deleted total
2019-04-10T23:11:40.483Z INFO     [main.py:delete_env:700] Successfully finished␣
↪deleting environment dev-dynamic-test
```

第 3 章

目次と索引

- genindex
- search