CloudFormation入門

1. CloudFormation って何？

- AWSリソース（EC2やRDS、S3等）をテンプレート化して自動で構築できる。
- CloudFormation自体は利用無料（利用したインスタンス等のみ料金がかかる）

テンプレート再利用のメリット

- 同じテンプレートを使いまわせる。
- 検証と本番で同じテンプレートを利用することにより、構成を同じにできる。
- AWSの構成を可視化出来る。
- バージョン管理も出来る。

CloudFormationの概念

- テンプレート
  JSON形式で記述する、AWSリソースのパラメータ。
- 複数のリソースを連携できる（例えば、EC2作成→EIP割り当て等）
- スタック
  テンプレートによって管理（作成、更新、削除）されるリソースをスタックと言います。
  AWSではリソースはスタック単位で管理されます。
  スタック作成時にスタックで利用するテンプレートを指定する。

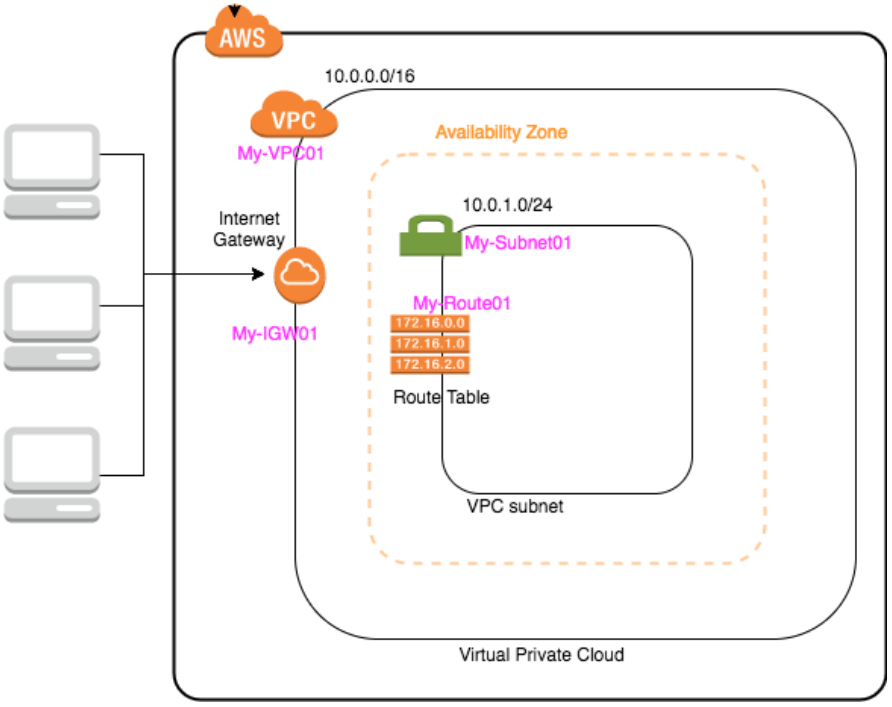
2. CloudFormation 利用時のワークフロー

1. 通常利用時のワークフロー
   - テンプレート作成
   - ローカルまたはS3へ保存
   - スタック作成（作成したテンプレートを指定）
   - 自動でプロビジョニングされる

2. スタックの更新時のワークフロー
   - テンプレートの修正
   - ローカルまたはS3へ保存
   - 更新したテンプレートを指定
   - 変更したリソースのみ更新される

3. 試行環境を準備する

図1 試行環境



以下のような内容のシンプルな環境を作っていきます。

1. VPC(10.0.0.0/16)を1つ作る。
   VPC名は「My-VPC01」とする。

2. そのサブネット(10.0.1.0/24)を1つ作ります。
   サブネット名は「My-Subnet01」とする。

3. インスタンスがインターネットにアクセスできるようにインターネットゲートウェイを作ります。
   ゲートウェイ名は「My-IGW01」とする。

4. インターネットゲートウェイへのルーティングテーブルを作ります。
   ルーティングテーブル名は「My-Route01」とします。

4. 作業の流れ

作業の大まかな流れは次の通り。

1. 環境を定義した「テンプレート」というファイルを作成する。
2. テンプレートファイルをCloudFormationに読み込ませる。

手順としては以上。

テンプレートはjson形式です。
CloudFormationは、テンプレートを作成することが全てです。

4-1. テンプレートの作成

図1 試行環境 の内容を書き起こすと以下のような感じになります。

```
{
  "AWSTemplateFormatVersion" : "2010-09-09",
   "Resources" : {
     "MyVPC" : {
        "Type" : "AWS::EC2::VPC",
        "Properties" : {
         "CidrBlock" : "10.0.0.0/16",
         "EnableDnsSupport" : "true",
         "EnableDnsHostnames" : "true",
           "InstanceTenancy" : "default",
           "Tags" : [ {"Key" : "Name", "Value" : "My-VPC01"} ]
         }
     },
    "MyIGW" : {
        "Type" : "AWS::EC2::InternetGateway",
        "Properties" : {
           "Tags" : [ {"Key" : "Name", "Value" : "My-IGW01"}]
         }
    },
    "MyAttachGateway" : {
       "Type" : "AWS::EC2::VPCGatewayAttachment",
       "Properties" : {
         "VpcId" : { "Ref" : "MyVPC" },
         "InternetGatewayId" : { "Ref" : "MyIGW" }
       }
    },
    "MySubnet" : {
        "Type" : "AWS::EC2::Subnet",
        "Properties" : {
           "VpcId" : { "Ref" : "MyVPC" },
           "CidrBlock" : "10.0.0.0/20",
           "AvailabilityZone" : "ap-northeast-1a",
           "Tags" : [ { "Key" : "Name", "Value" : "My-Subnet01" } ]
           }
    },
    "MyRouteTable" : {
        "Type" : "AWS::EC2::RouteTable",
        "Properties" : {
           "VpcId" : { "Ref" : "MyVPC" },
```

```
          "Tags" : [ { "Key" : "Name", "Value" : "My-Route01" } ]
        }
    },
    "MySubnetAttache" : {
      "Type" : "AWS::EC2::SubnetRouteTableAssociation",
      "Properties" : {
        "RouteTableId" : { "Ref" : "MyRouteTable" },
        "SubnetId" : { "Ref" : "MySubnet" }
      }
    },
    "MyRoute" : {
       "Type" : "AWS::EC2::Route",
       "Properties" : {
          "RouteTableId" : { "Ref" : "MyRouteTable" },
          "DestinationCidrBlock" : "0.0.0.0/0",
          "GatewayId" : { "Ref" : "MyIGW" }
        }
    }
  }
}
```

全体の構造は以下のようなJSON構造になっており、定義の一塊(リソース)を列挙していく。

```
{
  "AWSTemplateFormatVersion" : "フォーマットバージョン",

  "Description" : "説明文",

  "Metadata" : {
    テンプレートのメタデータ
  },

  "Parameters" : {
    スタック起動時にテンプレート渡す値
  },

  "Mappings" : {
    キーと名前付きの値を対応付け
  },

  "Conditions" : {
    入力パラメータによって評価される条件
  },

  "Resources" : {
    テンプレート化するAWSリソース（VPC,EC2など）を記載
  },

  "Outputs" : {
    リソースを追加、変更、削除する変更時に出力する内容
  }
}
```

4-2. Resources の設定

Resources にはAWSリソースの列挙をします。

```
  "Resources" : {
     <リソース>,
     <リソース>,...
     <リソース>
 }
```

各リソースは以下のような構造です。
リソースはリソースタイプ毎に作ります。
リソースタイプとは「何を作るか」の種類の指定の事で「Type」で指定します。

```
<リソースID>:{
  "Type" : "AWS::aws-product-name::data-type-name",
  "Properties" : {
        設定値
   }
}
```

指定できるリソースタイプの種類は公式ドキュメントのリファレンスに詳しく書かれています。
AWS リソースプロパティタイプのリファレンス

4-3. リソースID(論理ID)

最初の「MyVPC」がこのリソースID(論理ID)になります。
これはリソースの名前のことで、利用可能な文字種は、英数字(A-Za-z0-9)のみ。
テンプレート内で一意であれば任意で良い。

マネジメントコンソールで操作する時は、リソースID を押下して値設定などの操作を
することになります。

4-4. リソースの参照

テンプレートの中にチラホラ出てくる「Ref」の文字ですが、これが先ほど書いた「リソースID」を
参照するという意味です。
下記の部分で説明すると「MyVPC」が参照されていますが、参照されるのは
リソース名「MyVPC」で定義されているリソース内容になります。

```
   "MyAttachGateway" : {
      "Type" : "AWS::EC2::VPCGatewayAttachment",
      "Properties" : {
        "VpcId" : { "Ref" : "MyVPC" },
        "InternetGatewayId" : { "Ref" : "MyIGW" }
          }
     },
```

「MyAttachGateway」のリソースでは、
　インターネットゲートウェイ「MyIGW」に
　VPC「MyVPC」をアタッチしようとしています。
実際に作成される際は、「MyIGW」「MyVPC」で定義されたリソース内容を参照して
作成されることになります。

4-5. テンプレートの記述する順序

テンプレートを書く順番は特には決まっていません。
VPC作ってからサブネット作るという流れに沿って書かなくても、CloudFormation側で内容を
解釈して実行してくれます。
しかし、人がメンテしやすいようにある程度順番を意識して書くと、後で見た時にわかりやすい。

4-6. テンプレート記述のまとめ

ここまでの内容でまとめると、以下になります。

- 論理IDを適当に宣言する。
- 何を作るのか「リソースタイプ」Typeを指定する。
- 各Type毎に必要な要素を「Properties」で指定する。
- 他のリソースの参照が必要な場合は、参照したい論理IDを指定する。

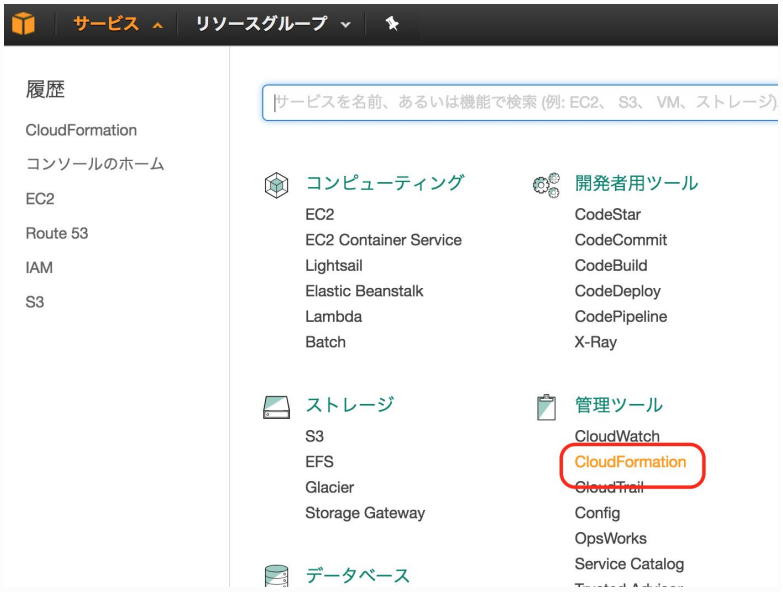
今回のテンプレートの構造は大まかに以下の通り。

```
{
  "AWSTemplateFormatVersion" : "2010-09-09",
   "Resources" : {
     "MyVPC" : {
        "Type" : "AWS::EC2::VPC",
     },
    "MyIGW" : {
        "Type" : "AWS::EC2::InternetGateway",
    },
    "MyAttachGateway" : {
        "Type" : "AWS::EC2::VPCGatewayAttachment",
    },
    "MySubnet" : {
        "Type" : "AWS::EC2::Subnet",
    },
    "MyRouteTable" : {
        "Type" : "AWS::EC2::RouteTable",
    },
    "MySubnetAttache" : {
       "Type" : "AWS::EC2::SubnetRouteTableAssociation",
    },
    "MyRoute" : {
        "Type" : "AWS::EC2::Route",
    }
  }
}
```

---

5. CloudFormation実行

作成したテンプレートで実際に環境を作っていきます。



CloudFormationの画面で「新しいスタックの作成」を押下。

今回は先ほどのテンプレートをファイルに保存して、ローカルからアップロードします。
S3に保存しておいてそのURLを指定することもできます。

指定したjsonのテンプレートに誤りがあると、ここでエラーになります。

エラー内容も合わせて表示してくれるので、修正してやり直します。
エラーとしては、jsonフォーマットの間違い(カンマ,の抜け漏れ等)や、参照IDのスペル間違いなどです。

参考：JSONチェッカー
https://lab.syncer.jp/Tool/JSON-Viewer/

テンプレート適用前の検証

テンプレートを利用する前に検証をする。

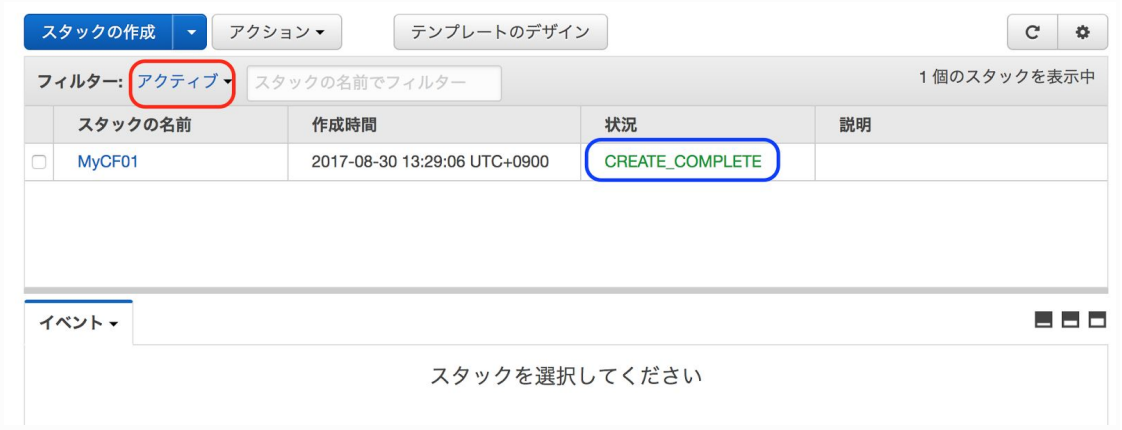
- 構文エラー、循環などのチェック
- コンソールから利用する場合は、自動で検証する
- CLI利用時は、テンプレート検証アクションを実行する必要あり

テンプレートに問題がなければ次の画面になるので、適当な名前をつけ、「次へ」を押下。

タグの指定、IAMロール選択などは必要に応じて指定します。

最後に確認して「作成」を押下すれば実際の作成が開始されます。

状況が「CREATE_COMPLETE」になれば作成完了です。
フィルターで状況を指定して絞り込み内容を変更して確認します。



途中で失敗すれば、作成途中でも全てロールバックされます。
途中で失敗するのは、例えば、指定したリソースタイプで使えない「Properties」要素が
入ってしまっている場合などです。

注意事項

- 作成権限
  自動でプロビジョニングさせるにはIAMでの権限が必要
  →EC2作成権限がなければ作成はできずに失敗する

- 失敗時
  途中で失敗した場合、ロールバックされ、途中まで作成したものは全て削除されます。

6. スタックの削除

スタック一覧で、該当スタックを選択、「アクション」→「スタックの削除」でスタックが削除されます。

スタックを削除すると、スタック内のAWSリソースはすべて削除される。
→これで、環境一式を作ったり消したりといったことが簡単に出来ます。

削除ポリシー
削除ポリシーにより、削除したくないリソースを削除しないように出来ます。
（Retain、スナップショット）

7. その他

Stack作成時にユーザが動的に値を指定できる「パラメータ」や
リージョン毎にAMIなどを指定できる「マッピング」
といった便利機能もあります。
詳細、応用は別途まとめる。

参考資料

CloudFormation超入門
http://dev.classmethod.jp/beginners/chonyumon-cloudformation/

AWS CloudFormation とは
https://docs.aws.amazon.com/ja_jp/AWSCloudFormation/latest/UserGuide/Welcome.html

AWS リソースプロパティタイプのリファレンス
https://goo.gl/kFjA8S

【AWS】CloudFormationまとめ
http://qiita.com/iron-breaker/items/a12d1228de12663e7d32

CloudFormationのスタックをDeleteしてもデータを残す方法
http://dev.classmethod.jp/cloud/aws/how-to-keep-datas-when-deleting-stacks/

AWS設計図作成ツール
https://www.draw.io/

draw.io awsアイコン
https://www.draw.io/?splash=0&libs=aws2