SRA の技術情報誌
GS LetterNeo
Vol.201　2026 年 5 月

Terraform でクラウド環境を構築する

沖 康輔

技術本部 技術開発室

1 はじめに

筆者は物理・仮想サーバーおよび AWS の構築・運用に携わる中で、GUI による手作業でも一定の品質は確保できるものの、構成の複雑化や作業頻度の増加に伴い、以下のような課題があると感じており、改善に向けて取り組みたいと考えていました。

・作業前後の差分管理やドキュメント修正の負荷

・文書化・チーム内での共有の難しさ

・設定内容の確認が行いにくい点や、作業結果の把握しづらさ

・手順書を用いても作業者によってつまずくポイントが異なる　など

これらの課題に対し、インフラ構成をコードとして管理する IaC（Infrastructure as Code）に注目していました。

IaC はサーバーやネットワークなどのインフラをコードとして扱うことで、構成内容を明示的に把握しながら適用できる特性があります。これにより、構成状態の再現性を高めるとともに、意図しない操作の予防、コードの再利用、レビューや変更管理が容易になり、GUI 操作ならではの課題を解消できると考えたためです。

そこで、これらの特性が実務でどの程度有効に働くかを確認するため、技術検証を行いました。

具体的には、IaC のツール「Terraform」で AWS リソースをコードとして定義し、環境構築の再現性や AWS リソースのモジュール化を行い、実際の構築の場面でも活用できるように構成しました。

そのうえで、Terraform による初期化（init）から変更内容の確認（plan）、AWS リソースのデプロイ（apply）、AWS リソース削除（destroy）までの一連の流れを通して技術検証を行いました。あわせて、実行手順を固定化することでコマンド操作手順を統一できるかについても検討しました。

具体的には Terraform の基本コマンド（init、plan、apply、destroy）を PowerShell スクリプトでラップし、スクリプト内で入力値チェックや確認プロンプトを組み込みました。

本技術検証では、AWS 環境を再現性高く構築・運用するための基本構成と考え方を整理し、実務で流用可能な Terraform を用いた AWS 環境構築のための土台となる構成を固めることを意識しました。また、IaC のツールはマルチクラウド対応や汎用性・導入事例の豊富さを評価し「Terraform(※1)」で検証することにしました。

※1
Terraform はクラウドサービスの設定をマウス操作ではなく、コードに記述して実行できる仕組みです。
マルチクラウドに対応しており、代表的なクラウドサービスであるAzure、GCPなどにも対応しています。
公式ホームページ：https://developer.hashicorp.com/terraform

2 本検証の範囲

まず、本検証の方針を以下の通り定めました。

- 開発／ステージング／本番で環境を分離し、運用する構成を対象とする。
- 実務利用を想定し、詳細は詰めすぎず、デプロイする AWS リソースは最小限かつ現実的な適用範囲に限定して検証する。

2.1 対象とする範囲

次に、本検証で扱う基本的な範囲を定めます。

- Terraform CLI を用いた AWS リソースの構築・更新・削除

- 環境（開発／ステージング／本番）ごとに分離したステート管理とリソース構成
- モジュール化を前提とした Terraform コード構成
- PowerShell による Terraform 操作の簡易化および操作手順統一化の検討

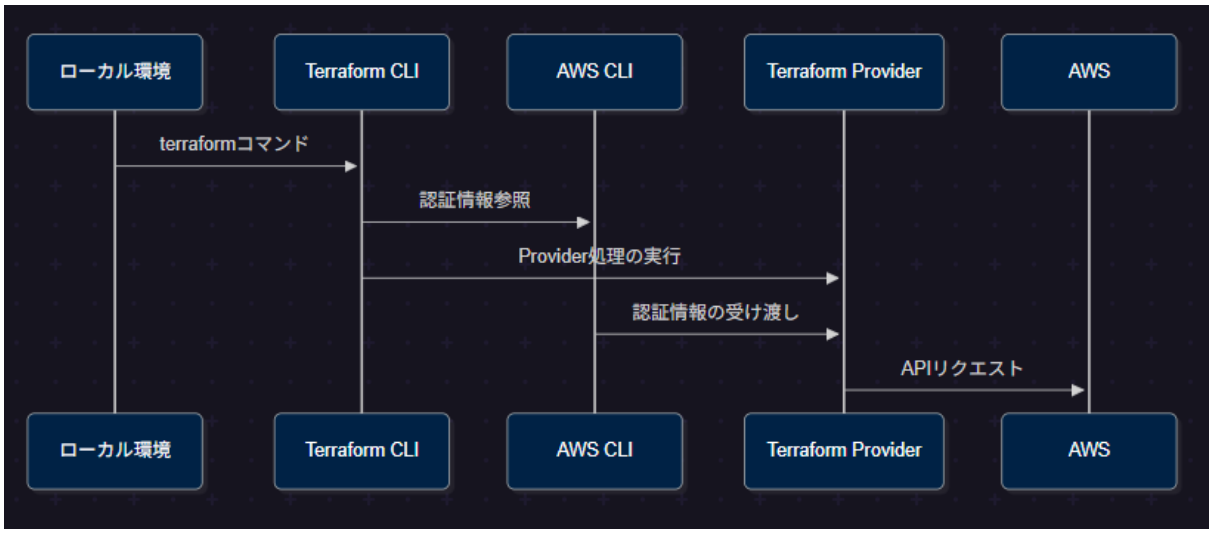
2.2 対象外範囲

また、以下の要素は本番稼働サービスでは重要な検討要素になりますが 本検証の目的からは外れるため、本検証では扱わない範囲としています。

- CI/CD パイプライン（GitHub Actions 等）、Terraform Cloud などのワークフローの利用
- 本番運用を前提とした高度なセキュリティ設計（マルチアカウント、Organizations、SCP、IAM 設計 等）
- 大規模環境や高可用性・災害対策（DR）を前提とした構成
- デプロイするリソースの詳細オプションおよび設定

3 Terraform 実行時の AWS 連携フロー

この章では、ローカル環境で Terraform を実行した際に、AWS に対してどのようにリクエストが送られ、リソースが作成されるかという処理の流れを説明します。



図の左側はローカル PC（Terraform の実行環境）、右側が AWS 側の処理を表しています。矢印は Terraform の操作を起点に、認証情報を取得し、Terraform Provider を通じて AWS API に伝わり、処理されるかを示しています。

Terraform は実行時に、まず認証情報（プロファイル、環境変数、SSO など）を参照します。その後、Terraform Provider に処理が渡され、Provider が AWS の API にリクエストを送信します。AWS 側では受け取った内容（リソースの作成・検証・変更・削除など）が実行され、結果として Terraform の設定ファイルに基づいた構成が実際の AWS 環境に反映されます。

4 基本方針

4.1 共通方針

共通方針として「環境単位、Terraform コマンドの実行単位、リソース作成単位を明確に分離する」と定めました。

4.2 環境および実行に関する方針

実行環境を明確に分離し、運用負荷を抑えるため、方針は次の通りに定めました。

- バックエンドストレージは「S3」とし(※2)、「暗号化」および「バージョニング」を有効化する
- Terraform のステートファイル更新時の S3 の排他制御オプション「use_lockfile = true」を有効化する
- 開発・ステージング・本番の各環境の分離は 1 バケット+プレフィックス(パス/ディレクトリ)で分離する(※3)
- 環境ごとの差異は terraform.tfvars（環境毎の個別変数定義ファイル） および backend.hcl（Terraform のステート保存先（S3 等）を定義する設定ファイル）で個別指定する
- Terraform コードは環境に依存しない形で共通化する

※2 バックエンドストレージの補足
この記事ではAWSのためバックエンドストレージにS3(AWSのオブジェクトストレージサービス)を使用しています。Terraformはマルチクラウドに対応しているため、Azure は Azure Blob Storage、GCP は Google Cloud Storageなど、利用するクラウドに応じて適切なバックエンドストレージを選択することができます。

※3 環境分離の補足
この記事では1バケットでプレフィックス(パス)を分けて環境ごとにステートファイルを分離しています。S3バケットを分離する方式もあります。
参考 Hashicorp社公式HP：
https://developer.hashicorp.com/terraform/language/backend/s3#multi-account-aws-architecture

4.3 コード構成と命名規則

コード構成と命名に関しては、検証の過程で複数のパターンを比較しながら検討を進め、最終的に以下の方針に整理しました。

- Terraform コードは役割を明確にした上でファイルを分割する

  （例：プロバイダー設定、変数定義、ローカル変数、モジュール呼び出し）

- リソースの命名規則は locals.tf に集約し、プロジェクト名・環境名・リソース種別を人が判別しやすい命名規則とする（※4）
- 再利用性を考慮しつつ、煩雑にならない範囲でモジュール化を行う

※4 環境分離の補足
AWS では多数のリソースが並ぶため、ARN(リソースの固有値)による識別は人の目では難しく、リソース名から役割や環境を把握するために、環境名や用途を含めた一貫した命名規則を設定することがベストプラクティスとなっています。

4.4 TERRAFORM の実行方法

操作手順を統一化する観点から、次のように方針を決めました。

- Terraform コマンドは直接実行せず、PowerShell スクリプト経由で実行する
- スクリプトでは環境選択や事前確認を行い、誤操作を防止する
- ユーザーに環境を明示的に確認するロジックを入れる
- 環境選択や確認処理は PowerShell で実装する
- Terraform コードは環境や実行方法に依存しない形で共通化する

4.5 ディレクトリおよびファイル構成

「4.1」～「4.4」の方針を踏まえ、初めに Terraform を用いた AWS の環境（開発／ステージング／本番）を再現性高く構築・運用するための必要な要素（基本設定、環境ごとの設定、AWS リソースのモジュール化、操作スクリプトなど）を整理し、Terraform の公式文書も参考にして構成案を複数作成し、比較を行いました。

最終的には以下のディレクトリおよびファイル構成にまとめました。

```
C:¥Users¥user¥dev¥terraform_template (コードの配置場所は任意です)
├── main.tf                    # メイン（主にTerraform とプロバイダー設定）
├── module.tf                  # モジュール呼び出し
├── variables.tf               # ルート変数定義
├── locals.tf                  # ローカル変数
├── outputs.tf                 # 出力定義
├── README.md                  # 利用方法
├── environments¥
│   ├── 01-dev¥
│   │   ├── backend.hcl        # Dev(開発)環境バックエンド設定
│   │   └── terraform.tfvars   # Dev(開発)環境変数値
│   ├── 02-stg¥
│   │   ├── backend.hcl        # Stg(ステージング)環境バックエンド設定
│   │   └── terraform.tfvars   # Stg(ステージング)環境変数値
│   └── 03-prd¥
│       ├── backend.hcl        # Prd(本番)環境バックエンド設定
│       └── terraform.tfvars   # Prd(本番)環境変数値
├── modules¥
│   ├── sns¥
│   │   ├── main.tf            # Amazon SNS リソース定義
│   │   └── variables.tf       # Amazon SNS モジュール変数
│   └── sqs¥
│       ├── main.tf            # Amazon SQS リソース定義
│       └── variables.tf       # Amazon SQS モジュール変数
└── scripts¥
    ├── init.ps1               # terraform初期化スクリプト
    ├── deploy.ps1             # リソースデプロイスクリプト
    ├── plan.ps1               # リソースプランスクリプト
    └── destroy.ps1            # リソース削除スクリプト
```

5 検証して分かったこと

5.1 設計・構成面

Terraform は記述の自由度が高く、1 ファイルに設定をまとめて書いても動作しますし、ファイルを分割しても問題ありません。

一方で、「どこに何を書くべきか」の環境ごとの分け方や階層構造・モジュールの分離の仕方についてはあらかじめ設計し、明確にしておく必要があると感じました。

Terraform は構成要素をファイル分割して柔軟に書くことができるため、全体構成や役割分担を意識しておかないと、後から整理するつもりで進めてしまい、結果的に構成の整理が後回しになりやすい点には注意が必要です。

また、運用フェーズに入ると、管理担当者が変わるケースも想定されます。そのため、構成や設計意図が読み取れる状態にしておく必要があると感じました。こうした点を踏まえると Terraform を導入する段階で設計しておくことが重要であると考えています。

さらに、稼働後に構成変更を行う場合、Terraform のステートと実際の AWS リソースが乖離（ドリフト）するリスクや意図しない変更が適用される可能性もあります。実際に運用しないと実感しにくいところですが、事前にモジュール構成・環境分離の方針について決定しておくことが重要なポイントです。

■設計不足により運用フェーズで発生し得るリスク

・運用に入ってからの修正コストが増える

・コード自体は読めても、全体構成の把握が難しくなる（特に開発と運用が分かれる場合）

・後から環境を分離する場合、意図しない環境に変更が及ぶ可能性がある（※3）

※3 Terraform の運用では、環境（開発／ステージング／本番）ごとにステートファイルなどを切り替えて利用することが一般的です。そのため、環境設定の切り替えを誤ると、意図していない環境に対して実行される可能性もあるため、環境を分離する際のリスクとして注意する必要があります。

こうした点を踏まえて、設計時に考えておくべきポイントを以下のように整理しました。

・ステートファイルを環境ごとに管理するか、すべての環境を共通で管理するか

- ファイルをまとめるか、モジュールとして分割するか（Terraform のベストプラクティスではモジュール化が推奨されています。）
- 変数をどのファイルに定義するか（Terraform が扱う設定値を、全体で共有する variables.tf に集約するか、環境ごとに terraform.tfvars に持たせるか）
- ポリシーやファイル間の依存関係

5.2 ステートファイル管理に関する課題

Terraform の公式ドキュメントでは、チーム運用や本番運用では、リモートバックエンドの利用とステートロックの有効化が強く推奨されています。[1]

一方で、単独・短期の PoC などのクラウドインフラを触る人が限定される小規模システムでは、例外的にステートファイルをローカルで管理する選択肢もあり得ると感じました。

例えば、

- クラウドインフラを操作する人が限られている
- 大学・研究用途・PoC などの小規模な環境

このようなケースでは、前提条件次第ではローカルでステートを管理する運用も選択肢になると感じました。もちろん推奨構成ではありませんが、「シンプルさを優先する」という判断もあり得ると考えております。

5.3 ステートファイル管理方式の検討

今回はベストプラクティスに沿って、リモートバックエンド環境でステートファイルは「S3」で管理する前提で検証しました。チームで共有管理することを想定した構成です。

ただ、実際の現場では統制や権限の都合でローカル管理を求められるケースも想定され、選択肢として入れたいと考えました。そのため、ベストプラクティスとして推奨される「リモート＋環境分離」を前提にしつつも現場の制約とどうバランスとるかという視点で判断軸を以下の通りに整理しました。

【統制上の制約】

- Terraform 実行者をどこまで限定する必要があるか

  例えば、S3 利用が認められないケースでは、ローカル管理も選択肢になる
- S3 や IAM の権限付与が可能か
- 手順・承認・監査をどの程度強制する必要があるか

【変更頻度と運用負荷】

- Apply（変更）頻度
- 変更時の影響範囲・大きさ（特に本番環境）

例えば、以下のようなシナリオが考えられます。

- 小規模開発 → 担当者 PC でステートファイルをローカル管理
- チームで運用管理するがスキルセットにばらつきがある場合

  ⇒ 実行者を限定してローカル管理
- 大規模だが操作権限を絞りたい場合

  ⇒ 実行者を限定してローカル管理
- 本番環境で安全にデプロイすることが最優先

  ⇒ リモートバックエンド＋S3 で管理

6. まとめ

Terraform を用いた IaC による AWS 環境構築について、環境分離・コード構成・ステート管理を中心に実務利用を想定した検証を行いました。本検証を通じて、IaC に期待した効果と検証結果を以下の通りまとめます。

- コードを共通化することで環境ごとに管理する負荷を低減できることを確認

⇒環境ごとの差分は固有の変数ファイルとして切り出せるため、作成工数や煩雑さを低減することにつながりました。

- 環境構築の再現性を高められることを確認

  ⇒環境構成をコードとして管理することで、構成内容を明示的に定義・確認しながら環境を構築できるようになり、安定して環境を再現しやすくなりました。

- 作業手順の統一化

  ⇒Terraform の基本コマンドを PowerShell スクリプトでラップすることで、コマンド操作を統一できるようになりました。あわせて、入力値チェックや確認プロンプトといったフェイルセーフを組み込むことで、操作のばらつきを抑え、習熟度にかかわらず安定した作業につなげられることを確認しました。実際に採用するかは判断が分かれると

ころだと思いますが、厳密なオペレーションを求められる環境や、チーム内の習熟度にばらつきがある現場環境では選択肢の一つになるのではないでしょうか。

Terraform を習得するための学習コストはかかりますが、基本的な実行フロー(「init：初期化」→「plan：変更内容の検証」→「apply：変更の反映」→「destroy：削除」)の理解自体は比較的とらえやすいと思います。

また、チームで運用する場合には段階的に IaC 管理へ切り替えを進め、その過程でメンバーも IaC を習得するのも、現実的なシナリオであると想定しています。

今後の展望としては環境ごとのバックエンドの切り分け、モジュールの充実化、変数定義の切り分けなど、設計段階で判断が必要となるポイントをさらに深掘りしていきたいと考えています。

最後までお読みいただきましてありがとうございました。

GSLetterNeo Vol.201
2026 年 5 月 20 日発行
発行者　株式会社 SRA 技術本部 先端技術研究室

編集者　熊澤努　方 学芳
バックナンバー https://www.sra.co.jp/public/sra/gsletter/
お問い合わせ　gsneo@sra.co.jp

株式会社SRA
〒171-8513 東京都豊島区南池袋 2-32-8

夢を。Yawaraka Innovation
やわらかいのべーしょん