Engineer

Terraform

インフラをコードで定義し、変更を計画・適用するHashiCorpのツール

どのようなものか

Terraformは、HashiCorpのInfrastructure as Codeツールです。クラウドやオンプレミスのリソースをプレーンテキストのファイルに記述し、バージョン管理します。Terraformは、稼働中のインフラをそのファイルに合わせるために、実際のオブジェクトをどう作成、変更、削除するかを判断します。プログラムを呼び出す名前は小文字の `terraform` で、コマンドやスクリプトにもこの表記が使われます。

言語は宣言型です。HashiCorpのドキュメントでは、「目標に到達する手順ではなく、到達したい目標を記述する」と説明しています。設定には、ある大きさのマシンが特定のネットワークに存在することを書き、そのマシンを作るためのAPI呼び出しは列挙しません。ファイルの拡張子は `.tf` で、JSONベースの `.tf.json` 形式もあります。ネイティブ構文はHCLという構文に基づき、主な構成要素は二つです。ほかの内容を収めるブロックと、名前に値を割り当てる引数です。

ツールの残りの部分を理解する鍵は、三つあります。

プロバイダーはプラグインです。設定にインストールすべきプラグインを指定すると、 `terraform init` が検索、ダウンロード、インストールします。HashiCorpによれば、アクセス可能なAPIを持つほぼあらゆるプラットフォームやサービスを、プロバイダーを通じて扱えます。公開されているものはTerraform Registryに掲載されています。各プロバイダーは、そのプラットフォームのリソース型を定義します。そのため、同じ中核プログラムで、異なる三つの事業者のDNSゾーン、仮想マシン、監視ルールを管理できます。

モジュールはディレクトリです。一つのディレクトリにまとめて置かれた `.tf` と `.tf.json` ファイルの集まりがモジュールであり、Terraformを実行する作業ディレクトリがルートモジュールです。

状態は記憶です。Terraformは実際のインフラを状態ファイルに記録し、公式の入門資料ではこれを環境の状態を判断する基準となる情報源と呼んでいます。状態は、設定内の各リソースと、それが管理する実際のオブジェクトをメタデータとともに対応付けます。既定では `terraform.tfstate` というローカルファイルで、その隣に前の版が `terraform.tfstate.backup` として保存されます。

ライセンスは変更されています。HashiCorpは2023年8月10日、以後の製品リリースすべてについて、ソースコードのライセンスをMozilla Public License 2.0からBusiness Source License 1.1へ変更すると発表しました。一方、API、SDK、その他のライブラリのほぼすべてはMPL 2.0のままです。OpenTofuは、Gruntwork、Spacelift、Harness、Env0、Scalrなどが始め、Linux Foundationの管理下で開発されるTerraformのフォークと説明されています。リポジトリはMPL 2.0で公開されており、公式サイトは、既存のワークフローと設定を維持できるTerraformの代替として紹介しています。

概念を判断に生かす

「Terraformとは？」を実際の判断に生かしましょう。その概念を、開発または購入するシステムに結び付けて考えるお手伝いをします。 構成全体をファイルに記述し、特定の人の記憶だけに頼らず、まったく同じ環境を再現できるようにします。

サービスを見る お問い合わせ →

用語の背後にある前提を検証する

解説とリンク先の実績を参考に、プロジェクトについての質問を整理してください。ツールを選ぶ前に、前提を一緒に検討できます。 Linuxサーバーを守る1,000件のRBACルールを作成 — 1,000 セキュリティ上の穴を塞ぐアクセスルール

Linuxサーバーを守る1,000件のRBACルールを作成 お問い合わせ →

なぜ重要なのか

HashiCorpは基本のワークフローを三段階で説明しています。記述では、複数のプロバイダーやサービスにまたがることもあるリソースを定義します。計画では、Terraformが作成、変更、削除する対象を判断し、実行計画として示します。適用では、承認された計画を実行します。その順序は、リソースグラフから導いたリソース間の依存関係に従います。

価値の多くは中間の段階にあります。計画は、何かが起こる前に、変更を書いた本人以外も読める事前確認資料です。レビュー担当者が見るのはファイルの差分だけではありません。二つのリソースを作成し、一つを更新し、ときには誰も触るつもりのなかった一つを削除する、という結果も分かります。最後の行をプルリクエストで指摘する負担は小さく、実行後に気付く負担は非常に大きくなります。

二つ目の理由は対応範囲です。各プラットフォームの知識はそれぞれのプロバイダーが持つため、一つの言語、一つのワークフロー、共通のレビュー手順で、チームが使うすべてのプロバイダーに対応できます。二つのクラウドと別のDNSホストを使う会社なら、その三つを一つのリポジトリに記述し、同じ方法でレビューできます。MicrosoftのBicepは、Azure Resource Managerのワークフローで別の選択をしています。デプロイの状態はAzureが保存するため、独自の状態ファイルを持ちません。ただし、BicepがAzure専用の言語という意味ではありません。拡張機能は、KubernetesやMicrosoft Graphなど、Resource Manager以外のリソースにも対応しています。

三つ目の理由は状態です。Terraformは、どの実際のオブジェクトがどのブロックに属するかを記録するため、自分が管理するものと、まだ見たことがないものを区別し、記録と稼働中の実態を比較できます。その記録は、このツールを使う代償でもあります。チーム全員がアクセスできる場所に置き、二つの実行が同時に書き込まないようロックする必要があります。また、秘密情報を含むことがあるため、秘密情報自体と同じ厳しさのアクセス制御が必要です。

より小さな負担もあります。HCLという言語を一つ覚える必要があり、各プロバイダーには固有のリソース型や癖があります。Terraformを基盤に製品を作るチームには、ライセンスを確認する必要もあります。HashiCorpのライセンスFAQは、何が競合する製品やサービスに当たるかを独立した質問として扱っています。OpenTofuは、同じワークフローをMPL 2.0で提供します。

実際の活用例

HashiCorpのリソース構文のページには、次の例が掲載されています。

```
resource "aws_instance" "web" {
  ami           = "ami-a1b2c3d4"
  instance_type = "t2.micro"
}
```

`resource` がブロック型で、 `aws_instance` がAWSプロバイダーの定義するリソース型です。 `"web"` はローカル名で、Terraformが状態ファイル内でリソースを追跡するラベルです。実際のマシンの設定には影響しません。ブロック内の二行は引数です。

次の三つのコマンドで、このファイルから稼働するマシンを作ります。

```
terraform init
terraform plan -out=tfplan
terraform apply tfplan
```

`terraform init` は作業ディレクトリを準備します。設定で指定されたプロバイダープラグインをインストールし、状態の保存先を設定します。何度実行しても安全で、公式ドキュメントは依存関係のロックファイルをバージョン管理にコミットするよう案内しています。

`terraform plan -out=tfplan` は計画を算出し、人がそのまま読めない形式のファイルに保存します。この計画作成によって、提案されたインフラ変更が適用されるわけではありません。その後、 `terraform apply tfplan` がそのファイル内の操作を確認なしで実行します。HashiCorpは、保存した計画を渡すこと自体を承認と見なすからです。同じ理由で、計画ファイルを指定した場合は `-auto-approve` フラグが無視されます。

チームでは、役割を自然に分けられます。CIジョブがプルリクエストに対して計画を生成してファイルを保存し、人が計画と差分の両方を読み、マージ後に別のジョブが同じファイルを適用します。読んだ内容がそのまま実行される仕組みです。applyのドキュメントは、忘れやすい条件も加えています。Terraformのワークフロー外で誰もインフラを変更できないようにする必要があります。そうでなければ、計画が記述する環境は、すでに実態とずれています。

このサイトの技術構成のページでは、TerraformとAnsibleを並べています。どちらもマシンのあるべき姿を記述し、環境の実態を後からたどり直すのではなく、文書として読めるようにします。別のツールで同じ考えを実践した事例が、Azureインフラのコードによるプロビジョニングです。Bicepを使い、テスト、ステージング、本番の各環境を同じテンプレート群から構築しています。テンプレートはパラメーターを受け取るため、環境の追加や変更はプルリクエストになります。この事例には、先ほどの選択の違いが表れています。BicepはAzureが記録を持つため独自の状態ファイルを必要とせず、Terraformは独自の状態を保持する代わりに複数のクラウドを扱います。

問題が起きるところ

Terraformの事故の多くは、同じ形を取ります。設定は有効で、実行も成功したのに、実際のリソースが消えたり変わったりしています。

リソースの名前を変えると、削除される。 ラベルを `"web"` から `"frontend"` に変えると、リソースのアドレスが変わります。HashiCorpのリファクタリングガイドによれば、Terraformは既定で、アドレスの変更を、既存のオブジェクトを削除して新しいものを作る指示と解釈します。使い捨てのマシンなら数分の損失ですが、データベースならデータを失います。対策は、何も削除せずにアドレスを更新する `moved` ブロックです。

```
moved {
  from = aws_instance.web
  to   = aws_instance.frontend
}
```

役目を終えた後も、このブロックは残します。HashiCorpは過去の `moved` ブロックを保持することを強く推奨し、削除は破壊的変更になると明記しています。

二人が別々のローカル状態のコピーに対して適用する。 初期状態では、各ワークスペースの状態はローカルファイルです。一人が一台のノートパソコンで使うには適しています。しかし、同僚が別のコピーからTerraformを実行すると、二つの状態ファイルは存在するものについて食い違い、それぞれの実行が自分の版に基づいて動きます。HashiCorpの推奨どおり、状態はHCP Terraformか、ロックとアクセス制御を備えたリモートストレージに保存します。そうするとTerraformは、状態を書き換える可能性のあるすべての操作で自動的にロックし、取得できなければ処理を進めません。すべての保存先がロックに対応するわけではないため、選択前に確認してください。 `-lock=false` は使わずにおきます。存在するフラグですが、公式ドキュメントは使用を推奨していません。

誰かがまだ作業中なのに、残ったロックを強制解除する。 自動解除が失敗するとロックが残り、 `force-unlock` コマンドはTerraformが表示したロックIDを使って解除します。ロックのドキュメントは、細心の注意を求めています。他人が保持するロックを強制解除すると、同じ状態に二つの実行が書き込む可能性があります。このコマンドは、自動解除に失敗した自分のロックだけを解除するためのものです。強制操作の前に、誰のロックで、その実行がまだ動いているかを調べます。

状態ファイルから秘密情報が漏れる。 HashiCorpは、状態に秘密情報が含まれる可能性があり、安全なアクセス制御のないシステムに保存すると漏えいし得ると警告しています。誰でも読めるバケットに置いたり、ビルド成果物にコピーした

りした状態ファイルは、認証情報の漏えいです。リポジトリには入れず、含まれる秘密情報と同じアクセス制御を適用し、手作業では編集しません。必要な修正には `terraform state` コマンドがあります。

承認が形だけになる。 `-auto-approve` は計画の対話的な承認を省略します。保存済みの計画ファイルを渡す場合も、そのファイルが承認と見なされるため省略されます。applyのドキュメントは、リソースの削除を含む破壊的な変更も対象になると警告しています。レビュー済みの計画なしでマージのたびに適用するCIジョブは、人が削除を事前に見つけられる唯一の機会を取り除いています。対策は、先ほどの保存済み計画を使う流れで、計画作成ジョブと適用ジョブの間に人のレビューを必須にすることです。自動化で `-auto-approve` を使う場合も、HashiCorp自身が、まず計画を生成してレビューするよう勧めています。

参考資料

- Terraformとは — HashiCorpの入門資料。記述、計画、適用、プロバイダー、状態を説明しています。
- terraform applyコマンド — 保存した計画、 `-auto-approve` 、自動化でのコマンド実行を扱います。
- 状態のロック — 自動ロック、非対応の保存先、 `force-unlock` の注意事項です。
- モジュールのリファクタリング — アドレス変更が削除と作成を意味する理由と、 `moved` ブロックを説明しています。
- HashiCorpによるBusiness Source Licenseの採用 — MPL 2.0からBSL 1.1への移行を発表した2023年8月の告知です。
- OpenTofu — Linux Foundationの下で保守されているTerraformのフォークです。

用語集

実際の制約に合わせて選ぶ

デプロイの流れ、運用上のリスク、緊急対応の担当者をお知らせください。検査、復旧手段、明確な責任の所在を伴う変更の範囲を定められます。

料金を見る お問い合わせ →

用語を実務に結び付ける

「Terraformとは？」がお客様の組織にどう当てはまるかをご相談ください。現在のシステムの具体例が一つあれば、話を始められます。

サービスを見る

welcome@engineer.company +45 50 10 61 84 デンマーク、コペンハーゲン

このページを持ち歩く

連絡先カード

心を込めて丁寧に制作： D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

著作権 © 2025 – 現在 · Engineer ApS · すべての権利を留保します。

印刷元： https://engineer.company/ja/glossary/terraform/