Engineer

インフラのコード化

版管理した定義ファイルからインフラを管理する手法

どのようなものか

インフラのコード化、Infrastructure as Codeは、システムを動かすマシン、ネットワーク、ファイアウォール規則、ロードバランサー、DNSレコードを、テキストファイルへ記述し、版管理し、ツールに実環境を構築させる方法です。ファイルが正本です。インフラの変更は、ファイルの変更として始まり、アプリケーションコードと同じレビューを通り、ツールが適用して初めて、稼働中のシステムへ届きます。infrastructure-as-codeとハイフン付きで書き、IaCとも略します。

NISTの特別刊行物（Special Publication）800-204Cは、マイクロサービスで作るアプリケーションのDevSecOps指針です。アプリケーションを配置する資源を、「繰り返し可能で、一貫した方法で」用意し、設定するコードと説明します。同じ指針は、アプリ自身のソースと同じツールで、これらの定義を版管理すると記します。再現性こそが目的です。ファイルから作れる環境は、最初の環境の隣でも、災害の後でも、火曜の午後の試験用アカウントでも、再び作れます。

定義は、大きく二つに分かれます。宣言的な定義は、特定の大きさのサーバーを、特定のネットワークへ置き、ポートを一つ開く、といった最終結果を示し、何を作り、変え、触らずに置くかはツールへ任せます。命令的な定義は、これを作り、それを接続し、その後ポートを開く、という手順一覧です。SP 800-204Cは、IaCを宣言的なコードに分類します。必要なものを書き、ツールが稼働するインスタンスへ変えるためです。HashiCorpもTerraform言語を、到達する手順ではなく、意図する目標の記述と説明します。違いは二回目に現れます。手順一覧は、一度どう行ったかを記録しますが、宣言は、何が存在するべきかを記録します。一部がすでに存在するとき、ツールに必要なのは後者です。

一つの言語に決まっていません。SP 800-204Cは、ツールによって、Python、JavaScript、TypeScriptなどの汎用スクリプト言語か、HashiCorpのHCLなどの独自設定言語で書き、後者がJSONなどの標準形式と互換かどうかは、それぞれだと記します。Terraformの言語は後者で、`.tf`で終わるファイルに書きます。

Azureは、JSONか、より短いBicep言語のResource Managerテンプレートを受け付けます。Ansibleは、YAMLでプレイブックを書き、相手へエージェントを入れず、既存のOSの認証情報を使い、SSHで接続します。

同じ名前に含まれる二つの仕事を、分けると分かりやすくなります。プロビジョニングは、仮想マシン、ディスク、ネットワーク、ファイアウォールを作ります。構成管理は、すでにあるマシンの中で動く、パッケージ、アカウント、サービスファイルを整えます。どちらへ寄るかはツールによって異なり、境界は厳密ではありません。どちらも、定義をリポジトリーへ置き、実システムをツールで一致させることが、IaCの条件です。

作ったものの記憶方法も、ツールで違います。Terraformは、定義した資源と作成した実オブジェクトを対応付ける状態ファイルを持ち、文書は、その状態情報を、環境についての正本と説明します。Bicepには、Azure自身が配置したものを記録するため、状態ファイルがありません。記録の場所で、二人が同時実行したときや、記録を失ったときの結果が変わります。

概念を判断に生かす

「インフラのコード化とは？」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

IaC以前は、各環境を手で用意し、手で修正していました。ステージングだけへの修正、試験用に開いたままのポート、三台のうち一台だけ新しいパッケージなどです。SP 800-204Cは、実際の設定が意図した運用状態を実現しなくなることを、構成のずれと呼びます。よくある入口も、コードを通らないウェブコンソールやコマンドラインの変更と記します。ステージングで動くリリースが本番で失敗するとき、よくある原因です。

IaCは、正本を稼働中のシステムの外へ移します。チームが定義を変え、システムがそれに従います。ログインしてファイアウォール規則を変えるのではなく、定義ファイルを変え、ツールに実行させます。そこから、三つのことが生まれます。

第一は、レビューと履歴です。ネットワークの変更は、何かが起きる前に、別の人が読むプルリクエストの差分になります。数か月後、誰がなぜポートを開いたかは、記憶ではなく、コミットログが答えます。取り消しは、変更を戻すコミットと、ツールの再実行です。

第二は、再現性です。適切な定義は冪等で、一致する環境へ適用しても変わらず、ずれた環境へ適用すれば戻します。二つの方法があります。今あるものを調整するか、捨てて、定義から作り直すかです。後者を規則とするのが、イミュータブルインフラです。

第三は、低コストの環境です。SP 800-204Cは、ツールが、資源の用意から撤去まで、インフラの寿命全体を自動化し、短命すぎて、保守する価値のないものもあると説明します。必要に応じて複数の試験環境を作り、再び取り壊せます。コード自体も、何かへ触れる前に検査できます。指針は、マージ前に、すべての部分に脆弱性検査を行うよう求めます。ブランチに本番構成の短命なコピーを与え、承認前に、レビュアーへ変更のプレビューを見せられます。

導入に伴う負担も、明確に述べる必要があります。プルリクエスト経由のファイアウォール規則変更は、ウェブコンソールより遅く、障害中には、その時間が大きな負担に感じられます。ツールごとの言語と、状態のモデルを学ぶ必要もあります。また、環境を作れるファイルは、消すこともできます。どちら向きにも同じだけ作用するため、プレビューは、その後の実行と同じほど重要です。

実際の活用例

CIジョブがツールを実行し、プルリクエストへドライランの結果を投稿する構成を例に、一つの編集が、稼働中のシステムへ届く流れを示します。

```
1日目 09:10 ネットワーク定義を編集: 事務所からTCP 443を許可
      09:12 プルリクエスト作成。CIが試行実行（計画）
      09:14 計画をプルリクエストへ投稿: 規則一つ追加、削除なし
      10:30 レビュアーが計画と差分の両方を読み、承認
      10:31 マージ。CIがレビュー済みの計画を適用
      10:33 同じ定義の二回目の実行が、変更なしを報告
40日目      443の開放理由を問われ、コミットログで実行者と理由を確認
```

計画が、その役割を果たします。差分は、著者が変えるつもりだったものを示し、計画は、削除しようとする資源など、著者の予想外も含め、ツールがこれから行うことを示します。Terraformは、`-out`で計画をファイルへ保存でき、

`terraform apply` へ渡すと、その中の操作を実行します。そのため、10:31に実行するのは、10:30に承認したものです。10:33の変更なしという二回目の結果は、定義とシステムの一致を確認します。

このサイトのIaCサービスのページでは、誰かの記憶に頼るのではなく、バージョン管理された設定を基に、再現可能な形でサーバーを構築する方法を説明しています。

安全に到達するには、主に順序が重要です。2026年9月の事例が、その理由を示します。別の事業者に二台目を追加します。一台目は、手で作り、後で保護したため、しばらくの間、新規ホストが初期パスワードポリシーのまま、インターネットへ開いていました。今回は、リポジトリーから作り、マシンができる前に保護を用意しました。サーバーのプレイより先にファイアウォールのプレイを動かし、二つ目の事業者では、クラウドファイアウォールより先にタグを割り当てます。タグ向けの規則は、タグがなければ対象を持たないためです。作成ロールが既存マシンを引き継ぐため、再実行しても害はありません。費用が発生する唯一のロールなので、専用の文書へ、リージョン、種類と仕様、ユーロ建ての月額を記録します。事例は、試行で見えないものも明示します。両ファイアウォールは、汎用HTTPタスクから事業者のREST APIを呼んで設定し、そのタスクはチェックモードに対応しないため、試行では省きます。本当の予行演習は、費用がかからず、取り消せる、ファイアウォールのプレイ単体の実行です。

問題が起きるところ

インフラコードは、成功に見える形で失敗しがちです。ツールが正常終了し、CIも成功し、ファイルと実環境が、黙って食い違います。

対応中の手修正が、次回の実行で戻されます。 午前二時にコンソールでポートを開き、障害が終わり、一週間後の定期実行で、定義ファイルに変更を反映していないため、再び閉じます。反対に、再び実行しないことで、ファイルが実態を表さなくなることもあります。緊急変更は同日中にコードへ入れ、ファイルと稼働中のものを比較する定期検査を行います。SP 800-204Cは、各デプロイの直後と運用中の一定間隔で、その比較を求めます。Terraformでは、planコマンドの詳しい終了コードで、毎晩実行する短いジョブにできます。

```
terraform plan -detailed-exitcode
# 終了コード0: 変更なし、1: エラー、2: 変更あり
```

すでに適用したブランチで、終了コード2なら、実環境がコードからずれたことを意味し、誰かが理由を調べるべきです。

ファイルから資源を消しても、削除されるとは限りません。 定義から資源がなくなった場合の扱いは、ツールで異なります。Azure Resource Managerテンプレートについて、Microsoftのwhat-if文書は、存在するのにテンプレートにない資源を、完全モードの場合だけ削除すると記します。それ以外では、動き、課金され続けます。削除規則を調べてから頼り、恐れる削除だけでなく、期待する削除もプレビューで読みます。

試行では、すべては見えません。 プレビューは、変更のモデルであり、保証ではありません。Microsoftのwhat-if文書は、配置時にAzureが既定値を入れるだけでも、削除と報告するプロパティーを挙げ、その結果をノイズと呼びます。`reference` 関数で設定したプロパティーは、毎回変更と出ます。前の事例は、もう一つの隙間を示します。チェックモード非対応のタスクは、試行で省かれ、何も分かりません。既知のノイズを理解し、見えないものは、低コストで取り消せる対象での予行演習まで、未検証と扱います。

秘密情報が、リポジトリーや状態へ入ります。 定義ファイルのパスワードは、全クローンの履歴へ残ります。見落としやすい点として、Terraformの文書は、状態に秘密情報が含まれる場合があり、既定の保存先は `terraform.tfstate` というローカルファイルだと警告します。秘密情報は専用の保管庫へ置いて名前で参照し、状態は、HashiCorpの勧めどおり、ロックとアクセス制御を備えた遠隔ストレージへ置きます。状態ファイルを手で編集してはなりません。必要な変更には、`terraform state` があります。

スクリプトを、定義に見せかけます。 たとえば、サーバーを作るだけのシェルスクリプトをリポジトリーへ置けば、版管理はされていますが、冪等にはなりません。二回動かせば、二台になるか、途中でエラーになります。宣言的な資源か、前述の、既存マシンを引き継ぐロールのように、動く前に確認する手順を使います。Ansibleの文書は、基準を明確にします。システムが、すでにプレイブックの記述する状態なら、再実行しても何も変わりません。

参考資料

- サービスメッシュを使うマイクロサービスアプリケーションでのDevSecOpsの実装（NIST SP 800-204C） — 宣言的で版管理するコードとしてのインフラ、構成のずれ、検査です。
- Terraform言語の文書 — 宣言的な言語、資源、ブロックです。
- planコマンド — 試行、保存した計画、詳しい終了コードです。

- 状態 — Terraformが記録するもの、保存場所、機密性が高い理由です。
- Ansibleの紹介 — SSHで行うエージェント不要の自動化、プレイブック、望む状態です。
- テンプレート配置のwhat-if — Azureの配置プレビュー、変更の種類、既知のノイズです。

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

「インフラのコード化とは？」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/infrastructure-as-code/