Engineer

イミュータブルインフラ

❄ 稼働中のサーバーを直接変更せず、置き換えて更新する運用

どのようなものか

イミュータブルインフラは、マシンを一度稼働させたら、変更しないサーバー運用です。HashiCorpのWell-Architected Frameworkは、「一度デプロイしたら変更せず、置き換えるだけ」のインフラと説明します。アプリケーションの新版、ライブラリーの更新、セキュリティパッチ、設定の変更を、すべて同じ方法で届けます。新しいイメージから構築したマシンやコンテナーで、古いものを置き換えます。その後、古いものを停止し、削除します。

反対は、長く使うサーバーを、その場で更新するミュータブルインフラです。Chad Fowlerは、2013年6月、何年も更新したサーバーの姿を説明しました。何度もパッチを重ね、障害の最中に手で変えた設定や、一人しか知らないcronジョブを抱え、ついには、チームが変更も置き換えも恐れるようになります。イミュータブルデプロイメントと呼んだ解決策は、すべてのシステムを自動化で構築し、後は手を触れないことでした。更新や新リリースは、新しいサーバーやイメージを作り、古いものを捨てることになります。

この方法を実用的にするのが、二種類のイメージです。Packerの文書によると、マシンイメージは、事前設定したOSとインストール済みソフトウェアを持つ、一つの静的な単位です。コミュニティーのツールPackerは、一つの元設定から、複数のプラットフォーム向けに、同じ構成のイメージを作ります。二つ目はコンテナーイメージです。Dockerの文書は明快で、一度作ったイメージは変更できず、新しいイメージを作るか、新たな層として上へ変更を足すしかありません。

この言葉は、インストールする製品ではなく、システムの運用方法を指します。境界もあります。アプリケーションサーバーは毎リリースで置き換えられますが、扱うデータは、置き換えの後も残る場所に置く必要があります。実際の設計作業の大半は、この境界へ向きます。

概念を判断に生かす

「イミュータブルインフラとは？」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

第一の理由は、構成のずれです。HashiCorpの枠組みは、手作業の更新や、その場での変更で、サーバーが元の設定から徐々に離れると明示します。同じ構成で、一年違いに始めた二台でも、修正、再試行、手編集の順序が異なれば、最後に同じとは限りません。どのサーバーも、一つのイメージの新しいコピーなら、イメージが完全な仕様になり、ずれる余地が残りません。

第二は、ロールバックです。リリースが不調なら、この枠組みの答えは、稼働中のサーバーを直すのではなく、前の成果物を再デプロイすることです。前のイメージは、すでに本番で動いた既知の状態なので、切迫した中で、記憶やシェル履歴を頼りに、昨日のサーバーを再構成する必要がありません。

第三は、本番へ手を加える人が減ることです。枠組みは、本番サーバーへ接続して手で更新すると、入力ミスや手順忘れなどの人為的な誤りを招く一方、この方法なら、誰も手でログインする必要がないと指摘します。稼働中のマシンへのログイン権限が必要な人自体が減り、持つ人も、使う必要はほとんどありません。

第四は、監査証跡です。同じページは、監査人が、本番を承認済みの手順外で変更していない証拠を求め、手作業では記録が不完全になると説明します。各変更が新しい成果物なら、版管理とCI/CDのログが、何をいつ変え、誰が承認したかを示します。セキュリティパッチも機能リリースと同じ経路を通り、稼働中のものを置き換える新しい成果物になるため、同じように構築、検証、記録します。

代償も伴います。枠組みは、仮想マシンの起動は、コンテナーより時間と資源を使う一方、コンテナーには運用するオーケストレーターが必要と記します。設定一行でも、すべての変更に完全なビルドと置き換えが必要なので、ビルドを自動化し、誰も省きたくならないほど速くする必要があります。状態は、最初の置き換えの前に、置き換え対象のマシンから出さなければなりません。後からではありません。

実際の活用例

Packerのテンプレートは、マシンイメージの構築方法を記します。以下は、Amazon EBSビルダーの文書にある例を基に、buildブロックのリファレンスに示す形で、`build`ブロックを加えたものです。

```
source "amazon-ebs" "web" {
  access_key    = var.aws_access_key
  secret_key    = var.aws_secret_key
  region        = "us-east-1"
  source_ami    = "ami-fce3c696"
  instance_type = "t2.micro"
  ssh_username  = "ubuntu"
  ami_name      = "web {{timestamp}}"
}

build {
  sources = ["source.amazon-ebs.web"]

  provisioner "shell" {
    scripts = fileset(".", "scripts/{install,secure}.sh")
  }
}
```

`source`ブロックは、ビルダーと元のイメージ`source_ami`を指定します。そのルートボリュームをコピーし、稼働中のインスタンスで設定します。`build`ブロックは、どのソースを構築し、どう設定するかを記します。shellプロビジョナーは、そのマシンを静的な成果物へ変える前に、中で二つのスクリプトを動かします。成果物は、新しいイメージです。ここでは、二つの変数の宣言を省いています。一つのコマンドで、テンプレートの全ビルドを実行し、成果物を作ります。

```
packer build web.pkr.hcl
```

デプロイが、もう半分です。HashiCorpの枠組みは、二つのツールを並べます。Packerでイメージを作り、Terraformでデプロイし、更新が必要になったら、丸ごと置き換えます。その場で変更できない場合、Terraformは、lifecycleのリファレンスによると、既存オブジェクトを削除してから、置き換えを作るのが既定です。この流れで更新する際は、稼働中の本番サーバーにログインする必要はありません。スクリプトを変えると新しいイメージができ、そこから新しいマシンができ、古いものを取り除きます。

同じ規律は、一台にも適用できます。PodmanとQuadletでコンテナーを動かす事例では、プロビジョニングとデプロイを分けます。rootがデプロイ経路を一度だけ用意し、その後のリリースは、特別な権限のない一般ユーザーが行います。そのユーザーは、ちょうど一つのスクリプトだけを実行でき、ほかは実行できません。リリースは、新しいイメージとサービスの再起動であり、rootシェルも、自由なコマンドも使いません。GitHub ActionsによるCI/CDの事例は、イメージ側をさらに進めます。本番フロントエンドは、シェルもパッケージマネージャーもないdistrolessの土台で動き、ビルドを環境から次の環境へ進めるのは、人ではなくワークフローです。ログイン先にできるものはほぼなく、何かをインストールするツールもないため、後継イメージを作るのが、変更のための唯一妥当な方法になります。

ゼロから再構築できるサーバーの事例は、もう一つの運用モデルを慎重に示します。Ansibleが、稼働中のホストをリポジトリーに書いた状態へ整え、各プレイは冪等です。リポジトリーには、宣言した環境を再構築するための設定が入っています。事例は、その場で変える費用も明示します。どの変更も、サーバー上の直接編集より時間がかかり、途中で止まった実行は、開始を拒否した場合よりも、ホストを悪い状態に残します。そのため、今では事前確認のプレイを先に動かします。半分だけ変わったホストは、Fowlerの述べた壊れやすいサーバーへの一歩であり、変更せず置き換えるのは、その危険を避ける別の方法です。

問題が起きるところ

このモデルは説明しやすい一方、崩れやすいものです。失敗の大半は、そのときは無害に見えた、一つの例外から始まります。

SSHでの緊急修正が、構成のずれを再び招きます。 切迫した中で、誰かが稼働中のサーバーへ入り、ファイルを編集します。次のデプロイでマシンを置き換えるまで修正は動き、その後、必要だった理由の記録とともに消えます。それまでは、サーバーがイメージと一致せず、まさにこの方法が防ぐ状態です。緊急のものも含め、すべての変更をイメージのビルドへ通し、それが最短経路になるほど速く保ちます。HashiCorpの指針は、手編集や更新のために稼働中のサーバーへ接続することを、一切認めません。

マシンへ書いたデータが、一緒に消えます。 Dockerの文書は、コンテナーを削除すると、その中で行ったファイルの変更も消えると説明します。置き換えるマシンにある、アップロード、生成ファイル、ローカルデータベースは、次のリリースでなくなります。置き換える部分の外へ、状態を保存します。Dockerのボリ

ュームは、一つのコンテナーの寿命を超えてデータを残し、同じボリュームへつなぐ新しいコンテナーにも、ファイルが見つかります。マシンでは、イメージに含まれない別ディスクや、データベースサービスが、その役を担います。

置き換えで、サービスが止まります。 Terraformの既定順序は、先に既存オブジェクトを削除し、次に置き換えを作るため、何も動かない時間が生じます。`create_before_destroy` をリソースの `lifecycle` ブロックに設定すると、順序を逆にできます。

```
resource "aws_instance" "web" {
  # ...
  lifecycle {
    create_before_destroy = true
  }
}
```

HashiCorpが明示的な選択にしているのは、一つのオブジェクトのコピー二つが、必ずしも並存できないためです。多くの種類で名前の一意性や、ほかの制約があります。リファレンスは、この設定を頼る前に、各リソースの種類ごとの制約を調べるよう求めます。

再構築が、黙って新しい土台から始まります。 Amazon EBSビルダーは、固定の `source_ami` の代わりに、 `source_ami_filter` から値を取り、 `most_recent` がtrueなら、一致する最新のイメージを選べます。文書は、日々のディストリビューション用ビルドに最も有用とします。最新に保つには便利ですが、同じテンプレートを異なる日に二回ビルドすると、それぞれ別の土台になり、古いリリースの再構築では再現できなくなります。リリース用では、 `source_ami` を特定のイメージへ固定し、レビュー済みのコミットで、意図的に更新します。

秘密情報を、イメージへ焼き込みます。 HashiCorpの枠組みは、データベースのパスワード、API鍵、暗号鍵などの秘密情報を、成果物に絶対に含めてはならないと明言します。成果物を読めれば取り出せ、認証情報を変えるたびに、すべての成果物の再構築と再デプロイが必要になるためです。マシンやコンテナーの起動時に、秘密情報を取得します。HashiCorp自身のパターンはVaultを使い、起動するマシンが認証して、有効期限のある一時的な認証情報を受け取ります。Vault内で認証情報を入れ替え、再構築は不要です。上のテンプレートも、ファイル内の文字列ではなく、変数で鍵を受け取ることで、同じ規則に従います。

必要になったとき、前の成果物がありません。 ロールバックは、前の成果物を再デプロイするため、まだ存在することが前提です。古いイメージを消す掃除処理や、各ビルドで同じイメージ名を使い回す習慣は、ロールバックを、変わったかもしれない土台の上で、古いソースから再構築する作業へ変えます。本番で動いた最近のリリースのイメージを残し、再利用しない名前を一つずつ与え、必要な日が来る前に、どれかへのロールバックを予行演習します。

参考資料

- イミュータブルインフラの作成 — 構成のずれ、ロールバック、イメージの種類、秘密情報に関するWell-Architected Frameworkです。
- サーバーを捨て、コードを焼く：イミュータブルインフラと使い捨ての構成要素 — Internet Archiveのコピーで読む、Chad Fowlerの2013年6月の記事です。
- Packerの紹介 — Packerが作るものと、マシンイメージの内容です。
- Amazon EBSビルダー — 上で使ったビルダーと、`source_ami`、`source_ami_filter`です。
- lifecycleメタ引数のリファレンス — Terraformの、削除してから作成する既定順序と、`create_before_destroy`です。
- コンテナーデータの永続化 — コンテナー内へ書いたファイルが一緒に消える理由と、ボリュームで残す方法です。

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

「イミュータブルインフラとは？」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/immutable-infrastructure/