Engineer

# Ansible

サーバーを文書で定めた状態にするツール

## どのようなものか

Ansibleは、SSHでマシンへ接続し、YAMLファイルに記述した設定へ整える、オープンソースの自動化ソフトウェアです。インストールするパッケージ、存在するユーザーやファイル、動かすサービスとその設定を定めます。実行元をコントロールノード、設定対象を管理対象ノードと呼びます。対象にエージェントを入れる必要はありません。OSの認証情報でログインし、その接続を通じて作業するためです。ファイルには結果を列挙し、Ansibleが各ホストで、すでにその結果になっているかを判断します。

構成要素には明快な名前があります。インベントリーは管理対象ノードを並べ、`web` や `db` などのグループに分け、グループ全体や個別ホストの変数を持ちます。プレイブックには一つ以上のプレイがあり、各プレイが、ホスト群とタスク一覧を結び付けます。タスクは、パッケージ、ファイル、サービス、ユーザー、ファイアウォールルールなど、一種類の対象の管理方法を知る小さなプログラム、モジュールを呼びます。公式文書は、プレイブックを、繰り返し利用できる単純な構成管理・複数マシンへのデプロイの仕組みと説明しています。ロールは、タスク、テンプレート、既定の変数、ハンドラーを一つの名前にまとめます。「PostgreSQLサーバー」や「防御を強化したSSHデーモン」を一度記述し、必要なホストへ適用できます。

短いプレイで、その形が分かります。

```
- name: ウェブサーバーがサイトを配信する
  hosts: web
  become: true
  tasks:
    - name: nginxがインストールされている
      ansible.builtin.package:
        name: nginx
        state: present
```

```
    - name: nginxが現在稼働し、再起動後も起動する
      ansible.builtin.service:
        name: nginx
        state: started
        enabled: true
```

各タスクは操作ではなく状態を示します。多くのモジュールは、まずその状態になっているかを確認し、なっていれば何もせず終了します。そのため、設定済みのホストへ二度目に実行しても、何も変えず、そう報告します。この冪等性によって、ホストごとのスクリプト適用状況を追跡せずに、同じプレイブックを全体へ毎回、安全に実行できます。

Ansibleの検討では、よく三つの近い選択肢と比較します。シェルスクリプトとの違いは、スクリプトは必要かどうかにかかわらず手順を再実行する一方、モジュールは実行前に確認することです。TerraformやBicepとは、対象の層が異なります。それらがネットワーク、データベース、仮想マシンなどのクラウドリソースを作るのに対し、Ansibleは、存在するマシンの中で動くものを設定します。両方を使うチームも多くあります。PuppetやChefとは、モデルが異なります。通常、それらは各マシンのエージェントが中央サーバーから設定を取得しますが、Ansibleは、起動元から設定を送り、接続先に必要なのはSSHだけです。

概念を判断に生かす

「Ansible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

手作業で設定したサーバーの再現性は、設定した人の記憶に依存します。半年後、カーネルパラメーターの値の理由、固定したパッケージとその理由、再構築で先に行うことを、自信を持って言える人がいなくなります。設定をプレイブックに書けば、その記憶を、読み、プルリクエストでレビューし、新しいマシンで同じ結果を得るために再実行できるファイルへ変えられます。

文書化すると、ずれの意味も変わります。プレイブックがホストの記述なら、稼働中のマシンにチェックモードで実行して、両者の不一致を報告できます。`--diff` を付けると、各ファイルの変更予定行も示します。定期的に宣言状態へ合わせるサーバー群は記述に近い状態を保ち、手動で編集したホストは、次の実行で変更として現れます。

このモデルには代償もあります。Ansibleは実行時だけ動くため、実行間のずれは次まで残ります。タスクは対象の全ホストに対して順番に一つずつ動くため、多数のマシンへの大きなプレイブックは、一つのことだけをするスクリプトより遅くなります。YAMLとJinja2テンプレートのロジックは、ロールの条件分岐が増えると追いにくくなります。また、ロールバック用のトランザクションがないため、途中で止まると、ホストを一部変更したまま残します。

## 実際の活用例

このサイトの複数のプロジェクトは、文書化した環境が時間とともにどう育ち、正しさを保つために何が必要かを示しています。

自社のコードによるインフラ管理は、公開済みのサービスと、他の構成要素の設定を対象とします。バックアップやメールの自動化も含みますが、有効化は別の手順です。記録された実装は、19のプレイブックと34のロール、12,065行のYAMLからなります。この数値は、その時点の状態であり、現在のリポジトリの規模ではありません。ホストは階層のグループに属し、その変数で適用するロールを宣言するため、引数なしの構築処理で、全ホストを宣言状態へ合わせます。冪等性は契約として扱います。整合したホストは変更ゼロを報告し、そう言えないプレイは未完成です。四つのコマンド名前空間で、ホストに触れない品質ゲートから、意図してホストを変更して判定する検証まで、それぞれの保証を区別しています。代償も率直に示しています。どの変更も、サーバーのファイルを直接編集するより時間がかかり、途中まで適用する収束処理は、拒否するものより悪いということです。

アクセス制御も、文書化が役立つ場面です。1,000のRBACルールをAnsibleで設定し、開発環境、アプリケーションインスタンス、PostgreSQL、MS SQL、Linuxサーバーにまたがる各ルールを、一つの役割とシステムに結び付けました。記録された成果は、文書化したロールベースのルール群と、それを繰り返し適用・確認する方法です。記録が裏付けるのは、その範囲であり、セキュリティリスクの低減量や設定時間の削減量を数値化していません。

試行実行を正直にすることにも、独自の作業が必要でした。プラットフォーム全体でチェックモードが事実を示すよう修正したきっかけは、2026年9月の試行実行でした。ホストについて誤ったアサーションで止まり、セキュリティサンドボックスを緩めるフラグを勧めていました。調べると、六つのプローブが、ホストから取得していない値で判断していました。実行がまだ触れていない状態を読むプローブは、現在はチェックモードでも動かします。それ以外は、未検証の内容を明示してスキップします。このルールを開発ガイドに加えました。

## 問題が起きるところ

Ansibleの問題の多くは、報告を、その背後の仕組みより信頼することから起きます。

**実行していないコマンドに、試行実行が問題なしと報告します。** チェックモードは模擬実行です。対応モジュールは変更を予測しますが、未対応のものは何も報告せず、何もしないと文書にあります。`command` と `shell` は部分対応で、`creates` や `removes` を与えなければ、コマンドの実行をスキップし、空の出力を結果として登録します。後続の `when:` がそれを読むと、測定されていない値で判断します。ホストを読むだけのプローブは、実行するよう指示でき、実行できないものはそう述べるべきです。

```
- name: チェックモードでもファイアウォールの状態を読む
  ansible.builtin.command: ufw status
  register: fw
  check_mode: false
  changed_when: false

- name: 試行実行で未検証の内容を示す
  ansible.builtin.debug:
    msg: "チェックモード：サービスの再起動は検証していない"
  when: ansible_check_mode
```

**九番目のタスクで止まり、最初の八つは適用済みです。** ホストでタスクが失敗すると、Ansibleは、そのホストを残りの実行から外し、それまでの変更を残します。触れる前にできる検査は、収束処理の前に置きます。自社環境のシークレット変更後も、そうしました。事前のプレイが、まずローカルでVaultを検証します。本当の収束処理と同じタスクファイルを約一秒で実行し、シークレットを解決できなければ、稼働中のサーバーを一部変更する代わりに、ノートパソコンで二秒ほどで拒否します。同じページは注意点も述べています。一つのプレイで設定したfactsは後にも残るため、一度の呼び出しに検査と収束を連結すると、再取得すべき値を共有してしまう場合があります。

**触っていないホストが、毎回変更を報告します。** `command` と `shell` は、コマンドが何を変更したかを知る方法がないため、毎回変更を報告します。それらばかりのプレイブックは、変更ゼロに収束せず、実際の変更が雑音に埋もれます。対象を直接管理するモジュールを優先し、`creates` や `removes` を加えるか、`changed_when` で、どの出力が変更を示すかを伝えます。

**パスワードがログやリポジトリに出ます。** 暗号化しなければ変数は平文です。Ansible Vaultは、変数ファイルや単独の値を暗号化してリポジトリに置けるようにし、タスクの `no_log: true` は引数と結果を出力から除きます。どちらも信頼する前に、テスト環境で一度、詳細な出力で実行し、内容を読みます。

## 参考資料

- Ansible Community Documentation：インストール、概念、各ガイドへの公式の入り口です。
- Ansibleのプレイブック：プレイ、タスク、モジュール、実行順序、実務上の冪等性を説明します。
- タスクの検証：チェックモードと差分モード：試行実行で分かること・分からないことと、タスクごとの設定です。
- インベントリーの作成方法：ホスト、グループ、それらの変数を説明します。
- Ansible Vaultによる機密データの保護：プレイブックと一緒に置くファイルと値の暗号化を説明します。
- ansible.builtin.command：部分的なチェックモード対応を含む、モジュールのリファレンスです。

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

「Ansible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/ansible/