ホワイトペーパー

ANSIBLE による 継続的インテグレーションおよびデリバリー

はじめに

Ansible は、極めて強力なオープンソースの自動化言語です。Ansible が他の管理ツールと違う点は、同時にデプロイおよびオーケストレーションのツールでもあるということです。自動化に関係するさまざまな課題に対処する上で、Ansible が目指すものは、生産性の大幅な向上です。Ansible は、他の自動化ソリューションの各種コア機能よりも生産性に優れた代替機能を数多く備えているだけでなく、IT 関連の主要な課題の解決にも貢献します。

これらの課題の 1 つとして、ゼロダウンタイムで行う継続的インテグレーションおよび継続的デプロイ (CI/CD) の実現が挙げられます。この目標を達成するためには多くの場合、大量のカスタマイズを伴うコーディング、複数のソフトウェア・パッケージを使用した作業、そしてそれらをつなぎ合わせるための社内開発が必要でした。Ansible は設計の初期段階から、まさにこのような各種シナリオのオーケストレーションを目的として設計されており、これらすべてに必要な機能をワンストップで提供します。

CI/CD を採用する理由

過去 10 年間にわたり、ソフトウェアと IT プラクティスの分析により明らかになったのは、短いサイクルで頻繁にリリースする方式 (「反復」または「アジャイル」) と比べて、リリースサイクルが長い方式 (「ウォーターフォール型プロジェクト」) では、劇的にオーバーヘッドが大きくなるということでした。たとえば、リリースの開始時点では、品質保証チームは万全の体制でテスト作業のために待機する一方で、IT チームにはデプロイできるものがありません。そして、リリースサイクルの終わりが近づくと、全速力で QA が実行され、IT チームも忙しくなりますが、開発チームはバグ修正と次回リリースの計画との間で引き裂かれた状態になります。そのため、コンテキストスイッチ (思考の対象の切り替え) が頻繁に起こります。

さらに深刻なのは、リリースサイクルが長いと、バグ発見からその修正までの時間差が長くなることです。これは、1 つ問題があると何百万というユーザーに影響があるような、高トラフィックの Web プロパティにおいて特に問題になります。この問題を解決するため、ソフトウェア開発業界は「アジャイル・ソフトウェア開発」という標語のもと、「短期かつ高頻度でのリリース」体制へと急速に移行しつつあります。短期かつ高頻度でリリースを行うということは、どのチームのメンバーにとっても、業務体制のギアチェンジが少なくなる

ことを意味します。これにより、従来より遥かに短時間で変更、品質保証、デプロイを行うことが可能となります。このようなテクニックから得られる効果は、QA 自動化エンジニアリングやテスト駆動開発 (TDD) などの各種アプローチを用いることで、さらに向上します。「短期かつ高頻度でのリリース」体制への移行が進歩であるなら、CI/CD は、理想の状態と呼べるかもしれません。アジリティは段階的に実現されていくものですが、組織がアジリティの実現を目指して取り組みを行うならば、すばらしい結果が期待できることでしょう。アジリティ実現の鍵となるのは、自動化です。そのため、所要時間を短縮し、人間の介入を必要最小限に抑えるためのテクノロジーが極めて重要です。

この資料では、そのようなプロセスを可能とするために複数のコンピュータ・システムを接続する上で、Ansible と Red Hat® Ansible® Tower が理想的なツールである理由を説明します。

ゼロダウンタイムを目指すべき理由

ダウンタイムやサービスの停止時間が発生すると、収益の損失や顧客の不満につながります。世界中のあらゆるタイムゾーンにユーザーが存在する大手 Web アプリケーションの場合、ダウンタイムの発生は、極めて重要かつ複雑なアップグレード・プロセスのため以外では避ける必要があります。アプリケーションのバージョン・アップデートを行うだけでダウンタイムが発生するようではいけません。大企業の社内アプリケーション (会計システムやイントラネット・システムなど) の場合においても、基幹システムが停止すると、生産性に大きな影響を及ぼす可能性があります。あらゆる自動化プロセスは、運用能力に影響を与えることなく更新を有効にすることを目標とすべきです。

あらゆる自動化プロセスは、運用能力に影響を与えることなく更新を有効にすることを目標とすべきです。

ゼロダウンタイムは実現可能ですが、現実的な労力で実現するためには、ツールによる支援が必要となります。ここで役立つのが、Ansible のような、複数の階層やステップに対応した強力なオーケストレーション・エンジンです。

他のビルドシステム

継続的デリバリー (CD) を実現するには、最初のステップとして継続的インテグレーション (CI) を実現する必要があります。簡単に言えば、CI システムは、さまざまなソースコントロール・リポジトリにおける変化を監視し、適用されるテストをすべて実行し、ソースコントロールの各変更に基づいてアプリケーションの最新バージョンを自動的にビルドする (そして、理想的にはテストも行う) ビルドシステムです。その一例として Jenkins (jenkins.io) が挙げられます。

CD における重要なハンドオフは、ビルドが成功した際、ビルドシステムにより Ansible を起動できるという点です。ビルドの結果としてコードの単体テストまたは統合テストも実行するユーザーの場合は、一歩先を行くことになります。Jenkins では、Ansible Tower を活用することでビルドの成果物を複数の環境にデプロイできますが、本番環境とよく似た QA/ステージング環境を使用することで、テストの精度が上がり、ライフサイクルにおける予測可能性が大幅に向上します。その後、Ansible から返されるデータを参照し、ビルドシステムのジョブで Ansible Tower ジョブに直接関連付けることができます。

事実、後述のように、ステージング環境に対して Ansible Tower を使用することにより、本番環境へのデプロイメントに先立ってテストを実行することができます。

そのような複雑なワークフローも、Ansible が実現する複数の階層やステップに対応したユニークなオーケストレーション機能を、プッシュベースのアーキテクチャと組み合わせることにより、迅速に実行することができます。

ローリングアップデートと多層アプリケーション

CD システムの絶対的要件として、1 つのアプリケーションに含まれる各種アーキテクチャ層のローリングアップデートのオーケストレーションを行える必要があります。そのような複雑なワークフローも、Ansible が実現する複数の階層やステップに対応したユニークなオーケストレーション機能を、プッシュベースのアーキテクチャと組み合わせることにより、迅速に実行することができます。1 つの層が更新されると、すぐに次の層での処理が始まり、層と層の間でのデータ共有も容易です。

このような処理を可能にする Ansible のコア機能の 1 つは、「play」を定義する機能です。これにより、きめ細かなホストグループを選択し、それぞれに実行すべきタスク (または「ロール」) を割り当てることができます。タスクとは通常、特定のリソースを特定の状態にすべきという宣言です。たとえば、パッケージを特定のバージョンでインストールする、ソースコントロール・リポジトリを特定のバージョンでチェックアウトする、といったことが挙げられます。

大半の Web アプリケーション・トポロジーにおいては、特定の層を厳密な順序で更新することが必要です。たとえば、アプリケーション・サーバーを更新する前に、データベース・スキーマを移行し、キャッシュサーバーをフラッシュすることが必要かもしれません。

また、さらに重要なこととして、本番稼働中のシステムすべてのアプリケーションや構成の管理を同時に行うべきではありません。

サービスが再起動しても、それらがすぐに使用可能にならない可能性があります。同様に、アプリケーションを別のバージョンで置き換える処理も、一瞬では終わらない場合があります。アップデートに先立って、ロードバランシングされたプールからマシンを取り出せる必要があります。そして、プールからのマシンの出し入れを自動化する機能も重要です。

Ansible を利用すれば、「serial」というキーワードにより、ローリングアップデートの際、一度に処理するマシンの数を制御することができます。また、Ansible によるローリングアップデート処理のスマートな性質として、あるバッチ処理が失敗すると、ローリングアップデートは停止し、インフラストラクチャの残りの部分はオンラインのままになります。

管理コンテンツの継続的デプロイメント

運用サービスの CD (継続的デリバリー) に加えて、Ansible Playbook の内容自体の継続的デプロイも可能です。これにより、システム管理者と開発者は、Ansible Playbook のコードを一元的なリポジトリにコミットし、ステージング環境でいくつかのテストを実行してから、プロダクションへと移行する際にそれらの構成を自動適用することができます。これにより、ソフトウェアのデプロイに使用するのと同じソフトウェアプロセスを構成管理にも適用でき、同様のメリットが得られます。

ソフトウェアや構成の変更は、予期せぬダウンタイムの原因の中でも主要なものです。そのため、自動化されたテストと人間による確認はいずれも、実施することが有用な対策です。これを Gerrit (gerritcodereview.com) などのコードレビュー・システムと組み合わせることにより、変更の適用前に複数ユーザーの承認を要求することが可能になります。

ローリングアップデート中のロードバランシングおよびモニタリング

Ansible は、ローリングアップデート中におけるロードバランサーなどのツールとの連携性に優れています。Playbook の「ホストループ」の中では、「このアクションをシステム X 上でホスト Y の代理として実行」というような指定が可能です。

ここには、あらゆる種類のロードバランサーとの通信に加えて、特定のホストに対して停止フラグを立てることで、現在更新中のホストにするアラートのモニタリングをオフにすることも含まれます。「モニタリングをオフ - プールから削除 - 層を更新 - プールへの追加 - モニタリングをオン」といったような単純なイディオムにより、手動介入なしでの自動アップデートを、ダウンタイムを発生させず、アラーム・ページャー・アラートの誤作動もなく実現することができます。

ANSIBLE で行う統合されたステージングテスト

複数のインベントリソースとともに Ansible Tower を使用することにより、ステージング環境で Ansible Playbook のテストを実行することや、本番環境へのロールアウトの前にアップデートの実行の成功を必須条件とすることが可能です。このような設定は必須ではありませんが、実際の現場のシステムをアップデートするよりも前に運用環境アップデートのシナリオを (おそらく小規模に) 形成する上で理想的なものです。Ansible Playbook に対して「-i」パラメータを使用することで Playbook の実行対象となるインベントリソースを指定し、ステージングと本番環境の両方に同じ Playbook を使用するだけで、これを実現できます。

バージョン管理に基づくデプロイメント

さまざまな組織が各種 OS パッケージ (RPM、deb など) によりアプリケーションをパッケージ化することを好みますが、多くの場合、特に動的 Web アプリケーションにおいて、そのようなパッケージ化は不要です。そのため、Ansible には、バージョン管理とデプロイの作業を連携させるためのモジュールが数多く含まれています。Playbook では、特定のタグまたはバージョンでリポジトリをチェックアウトするように要求することができます。その場合、システムは、各種サーバーにおいてそうなっていることを確認します。Ansible では、ソフトウェアのバージョン変更が必要な場合に、変更イベントを報告して追加の対処動作をトリガーすることができるため、関連するサービスを不必要に再起動することがありません。

モニタリングツールとの統合

Ansible はオーケストレーション・システムであるため、各種 APM (アプリケーション・パフォーマンス管理) モニタリングツールとも統合が可能です。たとえば、アプリケーションの統合テストまたはデプロイメントにおいて、ある APM システムのエージェントのインストールまたは更新が必要になったとします。Ansible では、アプリケーション・スタックに対するエージェントのインストールを処理するロールを設けることが可能です。エージェントがオンラインになれば、Ansible は、モニタリング・スタックの中でアラートを設定できます (まだ存在していない場合)。このモニタリング設定により、デプロイメントに関心のあるチームに即時データを返すことが可能です。これにより、アプリケーションが問題なく実行されているかどうかを通知できます。

アップグレード後に本番環境でアプリケーションに何かが起こった場合、モニタリングツールにより Ansible を起動して、そのアプリケーションの過去のビルドのうち、良好とわかっているものに戻すことができます。ただし当然ながら、これが可能なのは、アプリケーションが過去のビルドへの変更を許容する場合のみです。

コールバックとプラグイン可能性

CI/CD のカルチャーにおいては、しばしば、イベント発生時にアラートを提供することが役立ちます。Ansible には、そのための機構が複数含まれており、その中には統合されたメールモジュールも含まれます。さらに、コールバック機構は、任意のシステムによる通知のプラグインに対応しており、チャット・ブロードキャスト (IRC、Campfire など)、カスタムロギング、社内ソーシャルメディアなどが利用可能です。

デプロイメントにおける「DESIRED STATE」リソースモデルの適用

ソフトウェア・デプロイメントにおいて Ansible を興味深いものとしている主な機能の 1 つは、ソフトウェアをアップデートするプロセスの中で、状態ベースのリソースモデルが使用されていることです。このようなモデルは、構成管理ツールにおいて普及しています。従来、一部のオープンソース・ツールは追加のデプロイメント・ソフトウェアやスクリプトと組み合わせて使う必要がありましたが、Ansible の場合は不要です。

なぜなら、Ansible であれば、各種システムの異なる層の間でイベントの順序をきめ細かく制御したり、他のシステムに対してアクションの権限委任を行ったり、リソースモデルの命令 (「パッケージ X の状態は Y でなければならない」) と従来のコマンドパターン (「run script.sh」) を同じプロセスの中で混在させたりすることが可能であるためです。

また、Ansible では、さまざまな条件についてテストするコマンドを実行した後、それらのコマンドの結果に基づいて条件判定を行うことも簡単です。構成とデプロイを1 つのツールチェーンで一元化することにより、各種ツールの管理におけるオーバーヘッドを低減するとともに、OS とアプリケーション・ポリシーの間の統一性を向上することができます。

構成とデプロイを1 つのツールチェーンで一元化することにより、各種ツールの管理におけるオーバーヘッドを低減するとともに、OS とアプリケーション・ポリシーの間の統一性を向上することができます。

テストをデプロイメントに組み込む

大きな権限には、大きな責任が伴います。自動化された CD システムをセットアップすることで発生する危険の 1 つは、悪い構成がすべてのシステムへと伝播する可能性があるということです。このような事態を防ぐため、Ansible では、何か不具合があった場合にローリングアップデートを中止するためのテストを、Playbook そのものの中に追加することができます。この場合、「command」ないし「script」モジュールでデプロイされたテストや、あるいは Ansible のネイティブモジュールとして作成されたテストをデプロイすることで、各種条件のチェックを行えます。チェック項目には、サービスの稼働状態を含めることが可能です。

任意の時点に「fail」モジュールを使用することで、ホストに対する実行を中止することができます。これにより、ローリングアップデート中の初期段階でエラーを検出することが簡単になります。たとえば、ステージング環境と本番環境の間の差から生じた構成エラーにより、本番環境でのデプロイメントにのみ障害が発生したとしましょう。この場合、Ansible の Playbook を記述する際に、ローリングアップデート中の第 1 段階でアップデート・プロセスを停止するようにできます。サーバー数が 100、そして一度にアップデートされる台数が 10 であれば、アップデートが停止して介入が可能になります。エラーの修正後は、ただ Playbook を再実行すればアップデートが続行されます。

Ansible ならば、障害が発生した場合に動作を続けてシステムを不完全に構成してしまうようなことはありません。Ansible はエラーを報告して注意を促すため、エラーへの対応が可能となります。また、アップデート・サイクルの中におけるどのホストで問題が発生したのか、そして各プラットフォームでどれだけの変更が実行されたのかについても、要約を出力します。

コンプライアンス・テスト

多くの環境では、実際に何らかの変更が必要な場合を除き、構成の変更を適用するべきではありません。これは、実行の指示を出す「ボタンを押す」よりも前に、理解し、納得しておく必要のあることです。しかし、このようなケースにおいても、いくつかの点に注意することで、CD システムを活用することは可能です。

たとえば、Ansible の「--check」フラグで利用できるドライランモードでは、Playbook プロセスを実行するとどのような変更が加えられることになるかの報告が出力されます。この場合、実際にコマンドが実行されるわけではなく、エラー発生の可能性があるスクリプトについても考慮されないため、あくまで参考にすぎませんが、極めて有用な目安となるため、覚えておくと役立つ方法です。

Ansible の「--check」フラグで利用できるドライランモードでは、Playbook プロセスを実行するとどのような変更が加えられることになるかの報告が出力されます。

新しいビルドの製品が利用可能になった際の継続的デプロイを行っていても、さらに高い頻度でコンプライアンス・チェックを実行することが可能です。これにより、手動介入により本番環境に変更が加えられて、別の Playbook の実行により修正を行うが必要が生じた場合にも、報告を受けられるようにできます。こうすることで、ソフトウェアやパーミッションの修正が必要かもしれない状況の検出が容易になる場合があります。

デプロイメントの自動化

Ansible には、CI/CD ソリューションとして理想的なツールや機能が数多く含まれています。たとえば、Ansible なら、ゼロダウンタイムのローリングアップデートを行うワークフローの中で、複数の層やステップにわたるプロセスのきめ細やかなオーケストレーションが可能です。これは、Ansible のユニークなアーキテクチャと、エージェントレスなシステムにより実現されています。エージェントレスであるということは、一段上のセキュリティを意味するほか、管理作業の管理も必要ありません。そして、従来ならば手動で行う必要のあった複雑な IT プロセスを、シンプルかつヒューマンリーダブルに記述することができます。もはや、会議室に閉じこもった運用チームが手動部分と自動部分の混在する作業に従事するような光景は、過去のものとなりました。Ansible を使うことで、デプロイ作業の完全自動化が可能です。

そして、この場合の「完全自動化」が意味するのは、真のアジャイルプロセスの実現です。つまり、変更のデリバリーをより迅速に、より少ないエラーで行うとともに、コスト効率に悪影響を及ぼすコンテキストシフトの頻度も抑えることができます。ダウンタイム、そして特に手動変更によるミスが排除されるということが、中核的な社内 IT サービスや高トラフィックの Web プロパティにおいて重要な意味を持つことは明白です。

Ansible は、アーキテクチャの各種機能に加えて、Jenkins などの CI システムとの密接な統合により、構成作業を自動化できるだけでなく、IT プロセスのあらゆる面を自動化できます。Ansible が単なる設定管理やソフトウェア・デプロイメントという分類ではなく、オーケストレーション・エンジンとして位置づけられるのは、このことを理由としています。

例と詳細情報

Ansible コンテンツの基本的な例
github.com/ansible/ansible-examples

Ansible Tower を今すぐ試す
ansible.com/tower

ANSIBLE

RET HAT ANSIBLE TOWER について

Ansible は、Red Hat をスポンサーとするオープンソースのコミュニティ・プロジェクトであり、IT 自動化を実現するための最もシンプルな手段であるとともに、システム管理者やネットワーク管理者から開発者および経営陣に至るまで、さまざまな IT チームが使用することのできる唯一の自動化言語です。Red Hat® Ansible® Automation は、アプリケーション・ライフサイクル全体を自動化するためのエンタープライズに対応したソリューションを提供し、サーバーからクラウドやコンテナ、そしてその間に存在するすべてのものを自動化します。Red Hat® Ansible® Tower は、複数の層で構成される複雑なデプロイ作業をサポートする商用製品であり、Ansible 導入環境に管理/制御、知識、そして権限委任の機能を追加します。

RED HAT について

オープンソースソリューションのプロバイダーとして世界をリードする Red Hat は、コミュニティとの協業により高い信頼性と性能を備えるクラウド、Linux、ミドルウェア、ストレージおよび仮想化テクノロジーを提供、さらにサポート、トレーニング、コンサルティングサービスも提供しています。Red Hat は、お客様、パートナーおよびオープンソースコミュニティのグローバルネットワークの中核として、成長のためにリソースを解放し、IT の将来に向けた革新的なテクノロジーの創出を支援しています。

北米
1 888 REDHAT1
www.redhat.com

ヨーロッパ、中東、アフリカ
00800 7334 2835
europe@redhat.com

アジア太平洋地域
+65 6490 4200
apac@redhat.com

中南米
+54 11 4329 7300
info-latam@redhat.com

日本
03 5798 8510

redhat.