記事
Mihoko Iijima · 2020年4月28日 7m read

GitLabを使用したInterSystemsソリューションの継続的デリバリー - パートV：コンテナを使用する理由

この連載記事
では、InterSystemsの技術とGitLabを使用したソフトウェア開発に向けて実現可能な複数の手法を紹介し、議論したいと思います。 次のようなトピックについて説明します。

- Git 101
- Gitフロー（開発プロセス）
- GitLabのインストール
- GitLabワークフロー
- 継続的デリバリー
- GitLabのインストールと構成
- GitLab CI/CD
- **コンテナを使用する理由**
- コンテナを使用するGitLab CI/CD

第1回の記事
では、Gitの基本、Gitの概念を高度に理解することが現代のソフトウェア開発にとって重要である理由、Gitを使用してソフトウェアを開発する方法について説明していますました。

第2回の記事
では、アイデアからユーザーフィードバックまでの完全なソフトウェアライフサイクルプロセスであるGitLabワークフローについて説明しています。

第3回の記事では、GitLabのインストールと構成ならびに利用環境のGitLabへの接続について説明しています。

第4回の記事では、CDの構成を作成しています。

この記事では、コンテナとその使用方法（および使用する理由）について説明します。

この記事は、Dockerとコンテナ
の概念を熟知していることを前提としています。 コンテナとイメージ
について知りたい場合は、@Luca Ravazzoloからの記事を確認してください。

メリット

コンテナを使用することには次のような多くのメリットがあります。

- 移植性
- 効率
- 分離
- 軽量
- 不変性

これらについて詳しく説明しましょう。

移植性

コンテナは構成ファイルや依存関係など、アプリケーションの実行に必要なすべてのものを格納しています。 これにより、ローカルデスクトップ、物理サーバー、仮想サーバー、テスト、ステージング環境、プロダクション環境、パブリックまたはプライベートクラウドなどのさまざまな環境でアプリケーションを簡単かつ確実に実行できるようにしています。

移植性のもう1つのメリットは、いったんDockerイメージを構築して正常に動作することを確認したら、それはdockerが動作する環境（最近のWindows、Linux、MacOSサーバー）であればどこでも動作するということです。

効率

すべてのシステムソフトなどを実行する代わりに、自分のアプリケーションプロセスだけを本当に実行させる必要があります。 コンテナはまさにこの要件を満たすものです。コンテナは明示的に必要なプロセスのみを実行し、それ以外は何も実行しません。 コンテナは個別のオペレーティングシステムを必要としないため、使用するリソースが少なくなります。 VMは多くの場合数ギガバイトサイズですが、コンテナは通常数百メガバイトしかないため、単一のサーバーではVMよりも多くのコンテナを実行できます。 コンテナは基盤となるハードウェアをより高度に利用するため、必要なハードウェアが少なくなり、結果としてベアメタルのコストとデータセンターのコストが削減されます。

分離

コンテナはアプリケーションを他のすべてのものから分離します。また、同じサーバー上で複数のコンテナを実行できますが、それらは互いに完全に独立しています。 そのため、コンテナ間の通信は明示的に宣言する必要があります。 あるコンテナに障害が発生しても、それが他のコンテナには影響することはありません。障害が発生したコンテナはすぐに再起動できます。 このような分離によって、セキュリティ上のメリットも生まれます。 例えば、ベアメタルのウェブサーバーの脆弱性が悪用された場合は攻撃者がサーバー全体にアクセスできる可能性があります。しかし、コンテナの場合は攻撃者がアクセスできるのはウェブサーバーコンテナだけです。

軽量

コンテナは個別のOSを必要としないため、数秒で起動、停止、再起動できます。そのため、関連するすべての開発パイプラインと本番稼働までの時間が短縮されます。 そのため、より早く作業を開始し、設定に費やす時間をなくすことができます。

不変性

イミュータブルインフラストラクチャはインプレースで更新されるのではなく、デプロイするたびに置き換えられる不変のコンポーネントで構成されています。 これらのコンポーネントはデプロイのたびに1回作成され、テストと検証が可能な共通のイメージから起動されます。不変性によって不整合が抑制され、異なるアプリケーションの状態間で容易に複製や移動を行えるようになります。 不変性に関する詳細をお読みください。

新しい可能性

これらすべてのメリットにより、インフラストラクチャとワークフローをまったく新しい方法で管理できます。

オーケストレーション

ベアメタル環境やVM環境には問題があります。これらには個別性があるため、予期できず、望ましくない多くの問題を、いずれもたらすことになります。

の問題に対する答えは、DevOpsチームがソースコードに使用するのと同じバージョン管理を使用した、記述モデルでのインフラストラクチャの管理手法であるInfrastructure as Code（コードとしてのインフラストラクチャ）です。

Infrastructure as Codeでは、デプロイコマンドは環境の起動状態に関係なく、常にターゲット環境を同じ構成に設定します。 これは、既存のターゲットを自動的に構成するか、既存のターゲットを破棄して新しい環境を再作成することによって実現されます。

したがって、チームはInfrastructure as Codeを使用して環境の記述を変更し、構成モデルにバージョンを付けます。これは、一般的にはJSONなどの適切にドキュメント化されたコード形式で記述されます。 リリースパイプラインは、モデルを実行してターゲット環境を構成します。 チームが変更を加える必要がある場合、チームはターゲットではなくソースを編集します。

これらはすべて実現可能であり、コンテナを使用する方がはるかに簡単です。 コンテナをシャットダウンして新しいコンテナを起動するには数秒かかりますが、新しいVMのプロビジョニングには数分かかります。そしてサーバーをクリーンな状態にロールバックすることさえ可能です。

スケーリング

これまでの内容から、Infrastructure as Codeはそれ自体が静的であると理解されるかもしれません。 それは正しくありません。オーケストレーションツールは、現在のワークロードに基づいて水平スケーリング（さらに同じ環境をプロビジョニングすること）も提供できるためです。 現在必要なものだけを実行し、ニーズに応じてアプリケーションをスケーリングすれば良いのです。 これにより、コストを削減することもできます。

まとめ

コンテナは開発パイプラインを合理化できます。 環境間の不整合が解消され、テストとデバッグが容易になります。 オーケストレーションを使用すると、スケーラブルなアプリケーションを構築できます。不変な任意の履歴ポイントへのデプロイやロールバックを簡単に実行できます。

組織は、上記のすべての問題が解決済みで、スケジューラーやオーケストレーターがより多くのことを自動処理するような高度な作業を望んでいます。

次の内容

次の記事では、コンテナを使用したプロビジョニングについて説明し、InterSystems IRIS Dockerコンテナを活用するCD構成を作成します。

#Git #継続的デリバリー #その他

ソースURL:
https://jp.community.intersystems.com/post/gitlab%E3%82%92%E4%BD%BF%E7%94%A8%E3%81%97%E3%81%9Fintersystems%E3%82%BD%E3%83%AA%E3%83%A5%E3%83%BC%E3%82%B7%E3%83%A7%E3%83%B3%E3%81%AE%E7%B6%99%E7%B6%9A%E7%9A%84%E3%83%87%E3%83%AA%E3%83%90%E3%83%AA%E3%83%BC-%E3%83%91%E3%83%BC%E3%83%88v%EF%BC%9A%E3%82%B3%E3%83%B3%E3%83%86%E3%83%8A%E3%82%92%E4%BD%BF%E7%94%A8%E3%81%99%E3%82%8B%E7%90%86%E7%94%B1