記事
Mihoko Iijima · 2020年4月28日 8m 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VI：コンテナインフラストラクチャ

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 Flow（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
- **コンテナインフラストラクチャ**
- 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。

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。

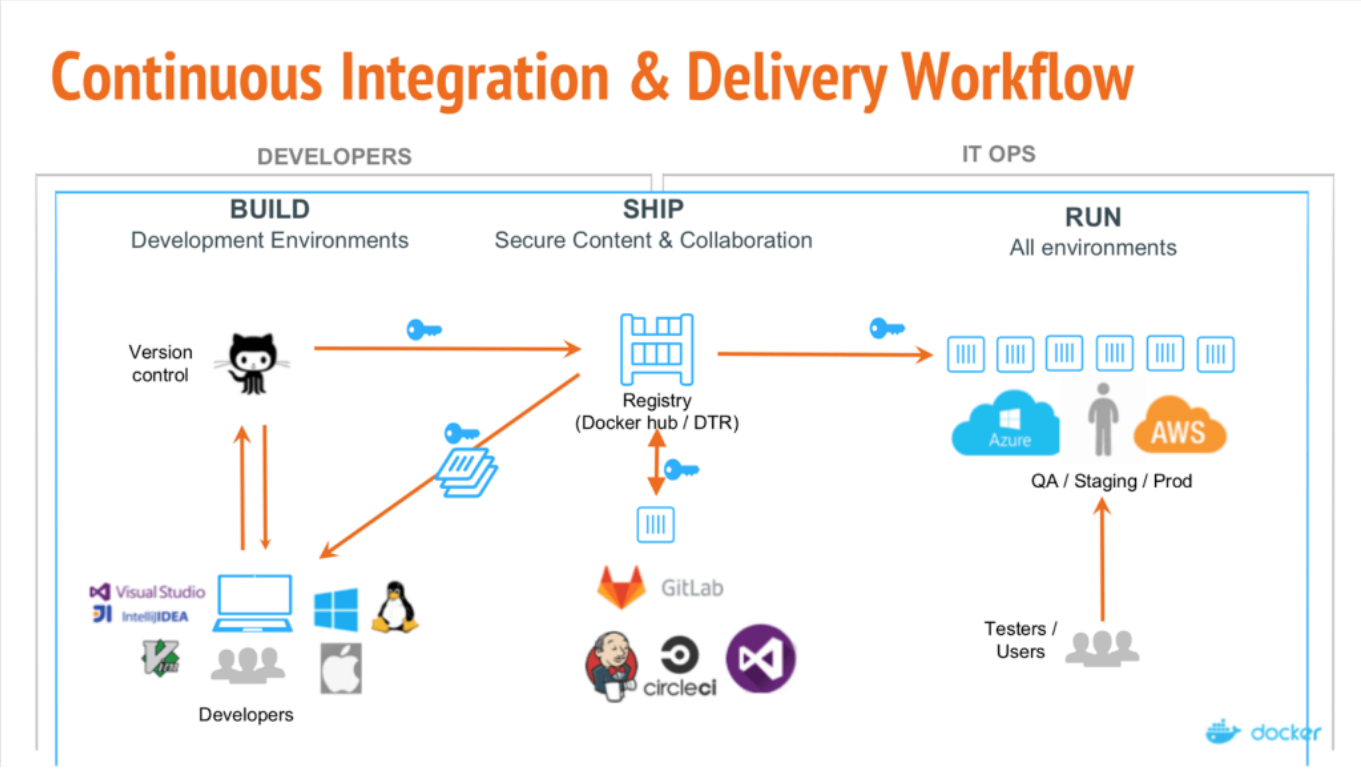
第3回の記事では、GitLabのインストールと構成ならびに利用環境のGitLabへの接続について説明しています。

第4回の記事では、CDの構成を作成しています。

第5回の記事では、コンテナとその使用方法（および使用する理由）について説明しています。

この記事では、コンテナを使用して継続的デリバリーのパイプラインを実行する必要がある主なコンポーネントと、それらすべての連携の仕組みについて説明します。

次のような構成になります。



ここでは、次の3つの主なステージに分かれていることが分かります。

- Build
- Ship
- Run

Build

前パートのビルドは多くの場合は増分ビルドであり、現在の環境と現在のコードベースの違いを計算してコードベースに対応するように環境が修正されていました。 コンテナの場合、各ビルドはフルビルドになります。
ビルドの成果物は、依存関係に関係なくどこでも実行できるイメージです。

Ship

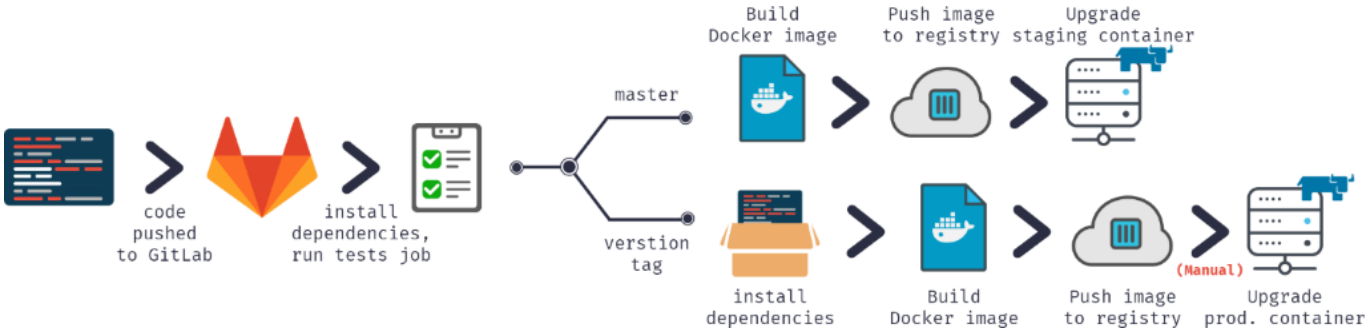
イメージが構築されてテストに合格すると、該当イメージが Docker イメージをホストする専用サーバーであるレジストリにアップロードされます。 そこで、同じタグを持つ前のイメージを置き換えることができます。 例えば、masterブランチに新たにコミットが行われたために新しいイメージ（project/version:master）を構築したとします。そして該当イメージがテストに合格した場合はレジストリ内のイメージを同じタグを持つ新しいイメージと置き換えることができるため、project/version:masterをプルする人全員が新しいバージョンを取得することになります。

Run

最後に、イメージがデプロイされます。 GitLabなどのCIソリューションはこの動作や専用のオーケストレーターを制御できますが、ポイントは同じです。一部のイメージが実行され、定期的に正常性がチェックされ、新しいイメージのバージョンが利用可能になると更新されます。

これらの各種ステージを説明したDockerのウェビナーをご覧ください。

あるいは、コミットの観点から見た場合は以下のようになります。



デリバリーの構成では以下を実施します。

- GitLabリポジトリにコードをプッシュする
- Dockerイメージをビルドする
- テストする
- Dockerレジストリにイメージを公開する
- 古いコンテナをレジストリの新しいバージョンと入れ替える

そのためには、次の内容が必要です。

- Docker
- Dockerレジストリ
- 登録済みドメイン（任意ですが推奨します）
- GUIツール（任意）

Docker

まず、どこかでDockerを実行する必要があります。 Ubuntu、RHEL、SUSEといった主流のLinuxフレーバーを搭載した1台のサーバーから開始することをお勧めします。
CoreOSやRancherOSといったクラウド指向のディストリビューションは使用しないでください。
これらはまったく初心者向けではありません。 ストレージドライバー
をdevicemapperに必ず切り替えてください。

大規模なデプロイの場合はKubernetes、Rancher、Swarmなどのコンテナオーケストレーションツールを使用するとほとんどのタスクを自動化できますが、ここではそれらについては（少なくともこのパートでは）説明しません。

Dockerレジストリ

これは実行する必要がある最初のコンテナであり、Dockerイメージを格納および配布できるステートレスでスケーラブルなサーバーサイドアプリケーションです。 次の場合はレジストリを使用する必要があります。

- イメージの保存場所を厳密に管理したい
- イメージの配信パイプラインを完全に所有したい
- イメージの保存と配布を社内の開発ワークフローにしっかりと統合したい

こちらにレジストリのドキュメントがあります。

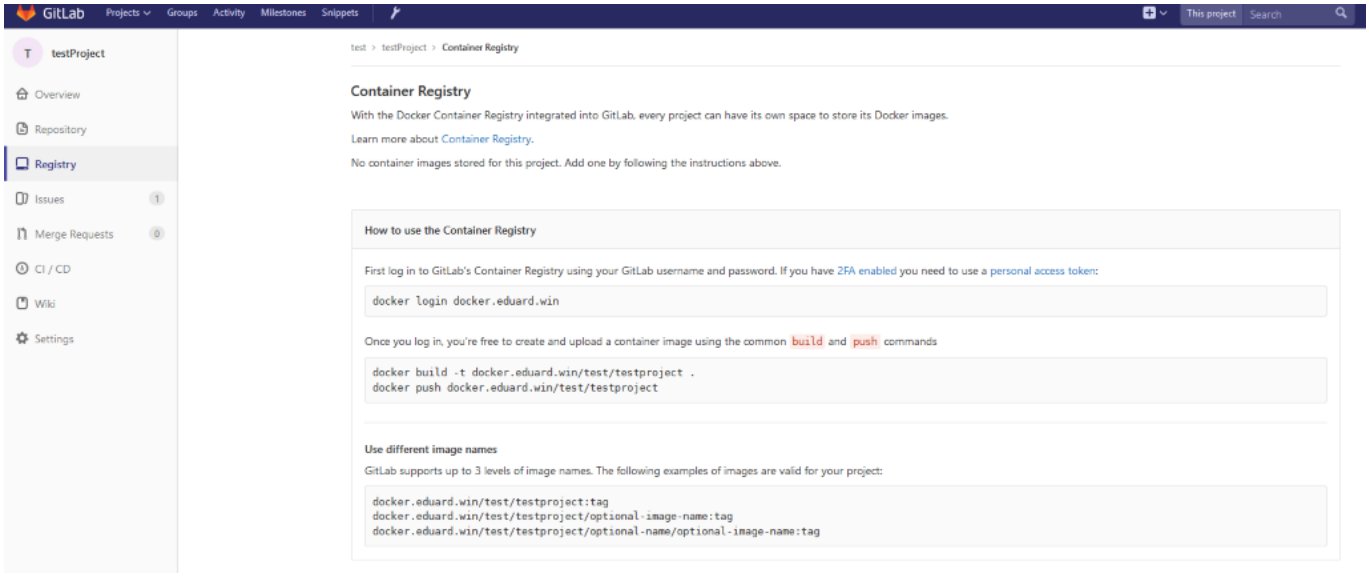
レジストリとGitLabの接続

注意：GitLabにはレジストリの機能が組み込まれています。

このレジストリを外部レジストリの代わりに実行できます。
この段落でリンクされているGitLabのドキュメントをお読みください。

レジストリをGitLabに接続するには、HTTPSサポート付き
でレジストリを
実行する必要があります。私は
Let's Encryptを使用して証明書を取得してからこのGistに従い、証明書を取得してコンテナに渡しました。 レジストリがHTTPS経由で利用可能であることを確認した後（ブラウザから確認できます）、レジストリをGitLabに接続する際のこちらの手順に従ってください。これらの手順
は、要件とGitLabのインストール形態によって異なります。私の場合は構成にレジストリの証明書と（適切な名前を付け、適切な権限を付与した）鍵を/etc/gitlab/sslに追加し、以下の行を/etc/gitlab/gitlab.rbに追加しました。
registryexternalurl ' https://docker.domain.com'
gitlabrails['registryapiurl'] = " https://docker.domain.com"

そしてGitLabを再構成
すると、新しくビルドしたイメージに適切にタグを付ける方法に関する情報を含む新しいレジストリタブが表示されました。このように表示されていました。



ドメイン

継続的デリバリーの構成ではブランチごとにイメージを自動的にビルドし、イメージがテストに合格した場合にそれをレジストリに公開して自動的に実行したため、アプリケーションはすべての「状態」で自動的に利用できるようになりました。例えば、以下を利用できます。

- <featureName>.docker.domain.com のいくつかのフィーチャーブランチ
- master.docker.domain.com のTestバージョン
- preprod.docker.domain.com のPreprodバージョン
- prod.docker.domain.com のProdバージョン

そのために
はドメイン名が必要で
、*.docker.domain.com を docker.domain.co
m のIPアドレスに向けるワイルドカードDNSレコードを追加する必要があります。
その他、異なるポートを使用するという選択肢もあります。

Nginxプロキシ

いくつかのフィーチャーブランチがある場合、サブドメインを正しいコンテナに自動的にリダイレクトする必要が

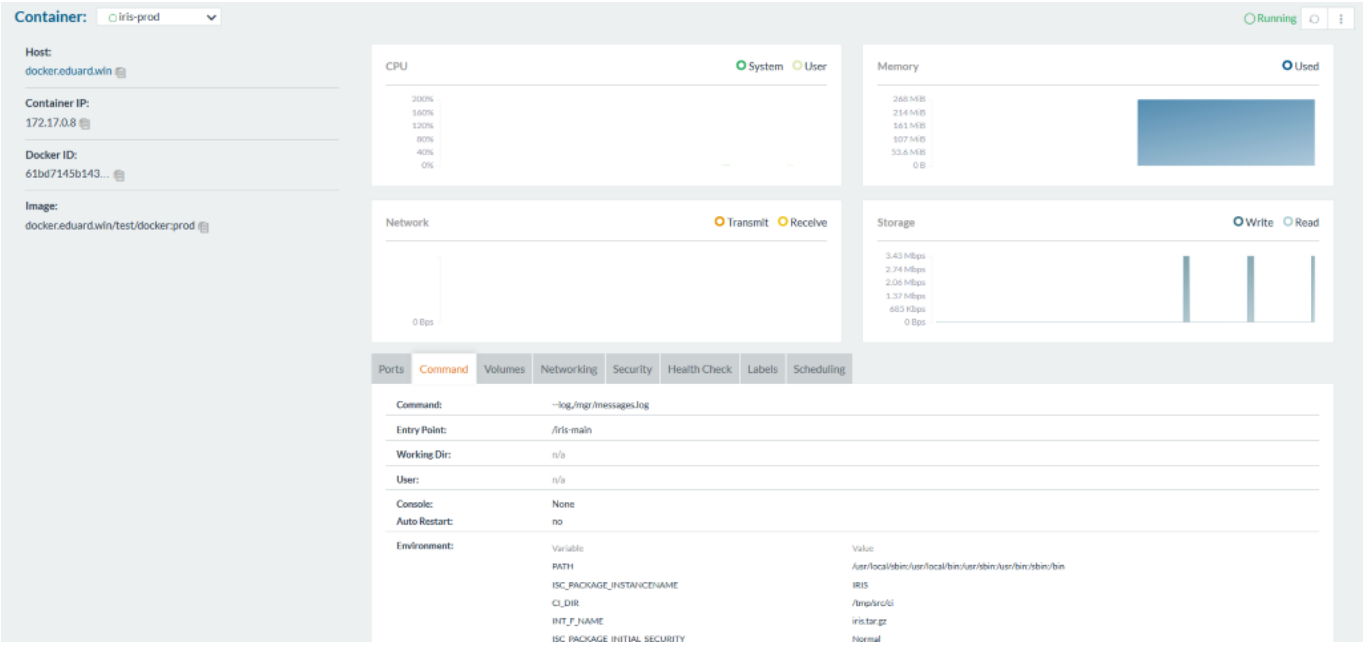
あります。 そのためには、Nginxをリバースプロキシとして使用できます。 こちらにガイドがあります。

GUIツール

コンテナの操作を開始するには、コマンドラインまたはGUIインターフェイスのいずれかを使用できます。
多くのツールを利用できますが、例えば次のようなものがあります。

- Rancher
- MicroBadger
- Portainer
- Simple Docker UI
- HYPERLINK "https://www.google.com/search?q=gui+container+managers"...

これらを使用すると、CLIではなくGUIからコンテナを作成し、管理できます。
Rancherは次のような外観をしています。



GitLab Runner

前と同じように、他のサーバーでスクリプトを実行するにはGitLab Runnerをインストールする必要があります。
この件は第3回目の記事で説明しました。

Docker executorではなく、Shell executorを使用する必要があることに注意してください。 Docker executorはイメージ内側から何かが必要な場合に使用されます。例えば、JavaコンテナでAndroidアプリケーションを構築していて、apkのみが必要な場合があります。
この場合はコンテナ全体が必要になり、そのためにShell executorが必要になります。

まとめ

コンテナの実行を開始するのは簡単で、選択できるツールは多数あります。

コンテナを使用した継続的デリバリーは、次のような点で通常の継続的デリバリー構成とは異なります。

- ビルド時に依存関係が満たされるため、イメージがビルドされた後に依存関係について考える必要はあり

ません。
- 再現性 - 同じコンテナをローカルで実行することで、既存の環境を簡単に再現できます。
- 速度 - コンテナには明示的に追加したもの以外は何も含まれないため、高速にビルドすることができます。また、さらに重要なことには一度ビルドしてしまえば必要なときにいつでも使用できます。
- 効率 - 上記と同様、コンテナはVMなどよりもオーバーヘッドが少なくなります。
- スケーラビリティ - オーケストレーションツールを使用すると、アプリケーションをワークロードに合わせて自動的にスケーリングし、現在必要なリソースのみを消費できます。

次の内容

次の記事では、InterSystems IRIS Dockerコンテナを活用するCD構成の作成について説明します。

#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%88vi%EF%BC%9A%E3%82%B3%E3%83%B3%E3%83%86%E3%83%8A%E3%82%A4%E3%83%B3%E3%83%95%E3%83%A9%E3%82%B9%E3%83%88%E3%83%A9%E3%82%AF%E3%83%81%E3%83%A3