HITACHI
Inspire the Next

DevOps開発プラットフォームの構築と本プラットフォームにおけるRubyの優位性

2016/11/4

株式会社日立ソリューションズ
技術統括本部 技術開発本部 生産技術部

牧　俊男

HITACHI
Inspire the Next

1. DevOps開発プラットフォームの構築

1-1 モチベーション

HITACHI
Inspire the Next

開発ツールは年々大型化している傾向にあり、
ツールの運用自体はクラウドサービスに任せる傾向がある

当社でもこれらのサービスを利用した開発を推奨したかったが、適用が難しかった

例えば、
- 権利関係の問題 → 当社(もしくは当社顧客)の著作物を社外に置いてもよいか
- 機密情報保護の問題 → 何らかのミスで情報が流出する恐れがないか
- 輸出管理の問題 → 海外のサーバにソースコードを置くため、輸出管理を行う必要
- セキュリティの問題 → 誤って外部のサイトを攻撃してしまう可能性はないか

社内にDevOpsの考え方を広めていくため、
自社管理のDevOps環境を構築して、社内に展開する活動を始めた

1-2 社内DevOps環境の構築

HITACHI
Inspire the Next

高速開発プロセス実現のためのツール基盤

- プルリクエスト型のコードレビュー
- テスト・デプロイの自動化
- PaaS環境での評価

評価・テスト

フィードバック

回帰テスト

課題登録

ビルド

CLOUDFOUNDRY

PaaS環境へ導入

drone

コードレビュー

ソース
更新

GitLab

反映依頼
（プルリクエスト）

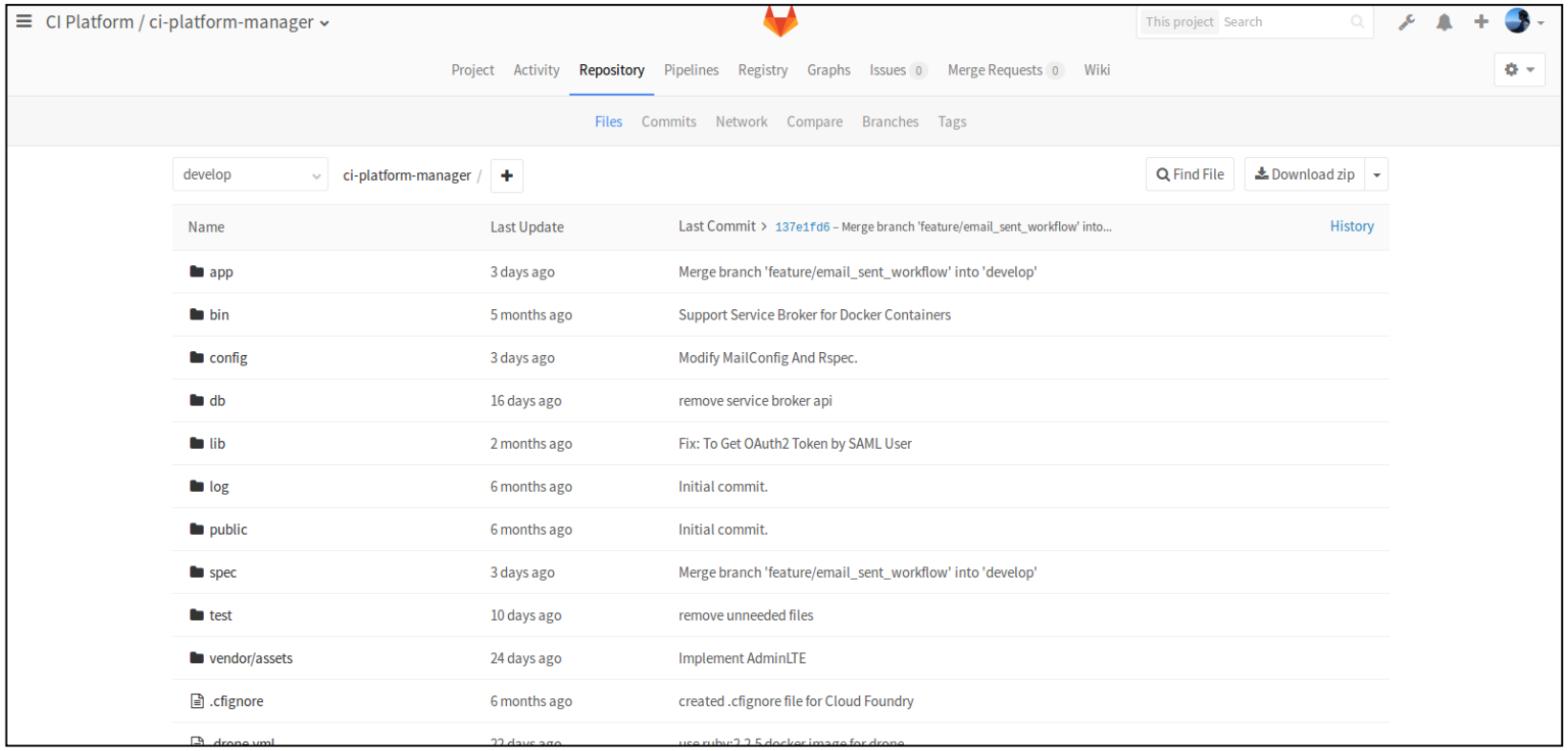
1-3 GitLab

HITACHI
Inspire the Next

GitHubと同等の機能を提供するソースコード管理ツール

GitLab

- Gitによるソースコードリポジトリの提供
- Issue管理(バグ報告や機能追加の受付をするもの)
- マージリクエスト管理(変更のためのワークフロー)
- 最近Docker Registoryが統合され、Docker Imageのリポジトリとしても利用可能

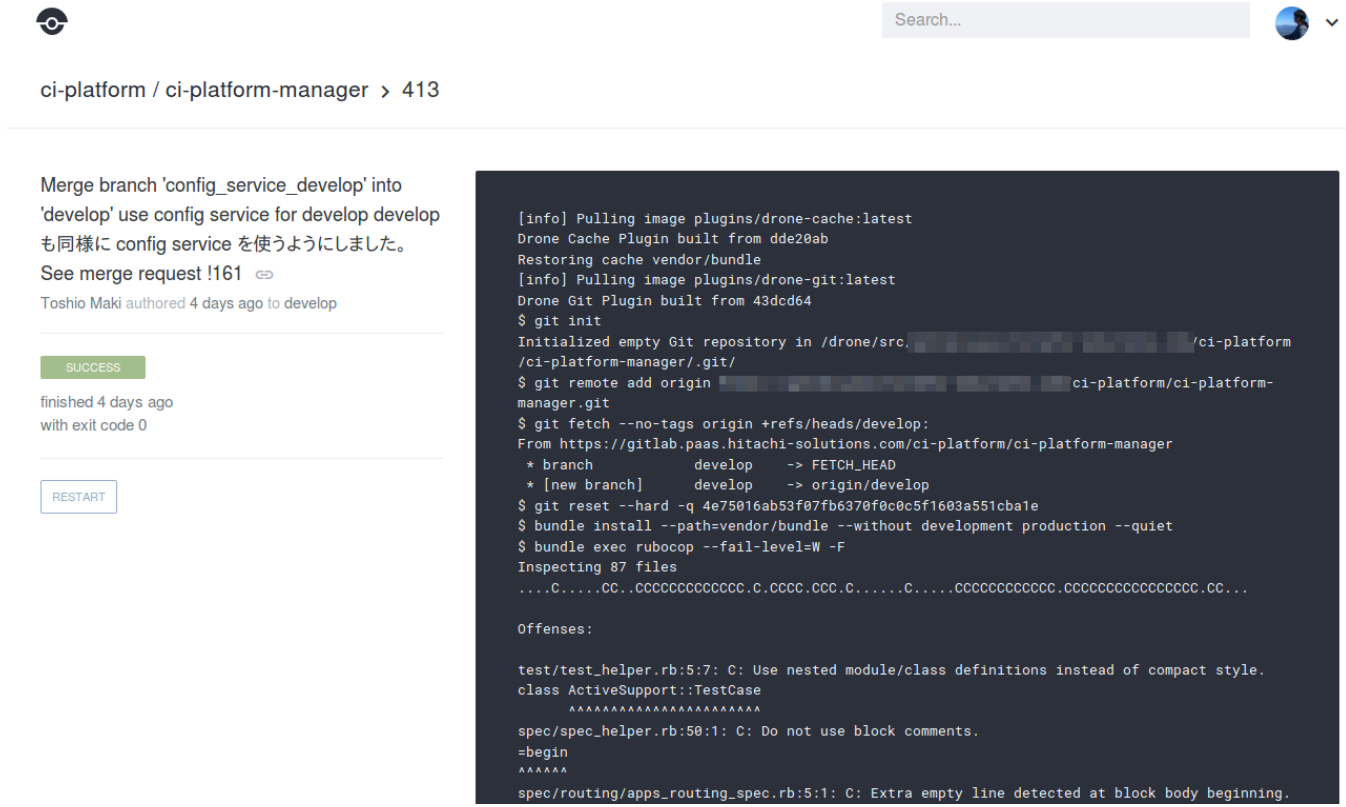


1-4 drone

HITACHI
Inspire the Next

継続的インテグレーションを実現するための自動テスト・デプロイ実行基盤

- GitLabのイベントにフックして、自動的に予定されたイベントを実行
- リポジトリ内に配置した設定ファイルでdroneを制御できる
  - Dockerコンテナを使ったクリーンなビルド・テスト
  - PaaS環境へのデプロイ自動化



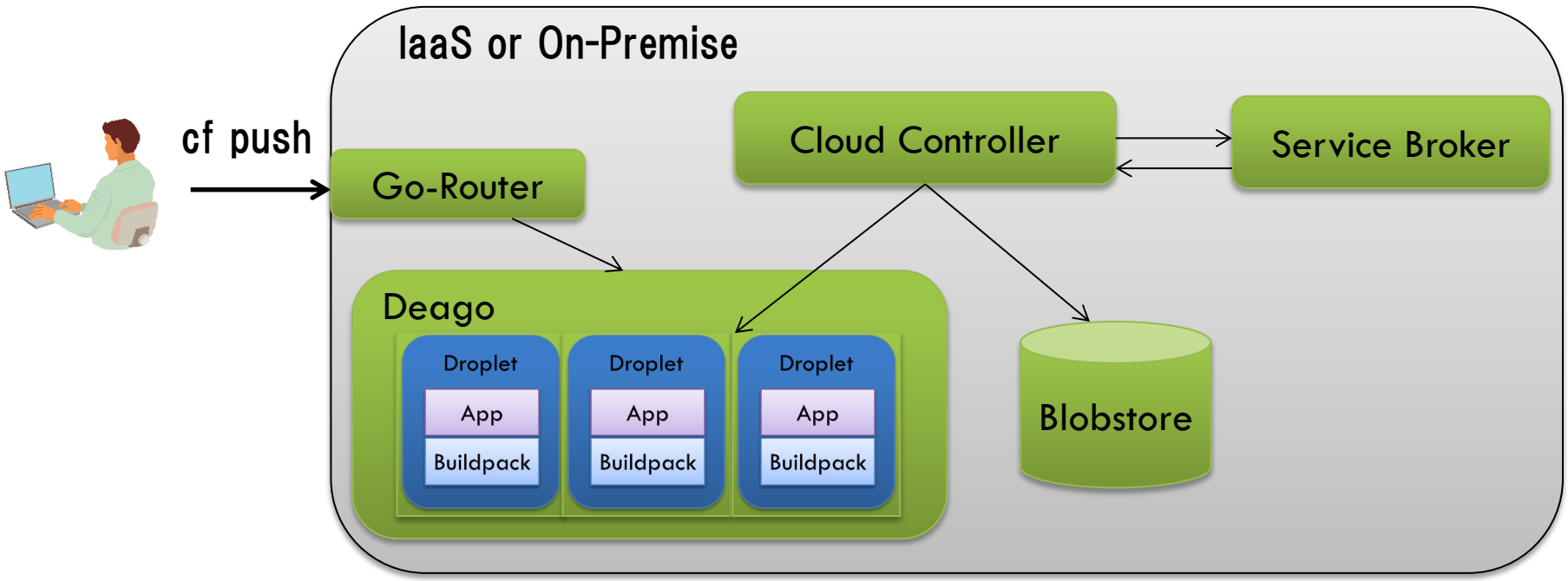
1-5 Cloud Foundry

HITACHI
Inspire the Next

継続的デリバリーを実現するためのPaaS基盤

- 1コマンドでのシンプルなデプロイプロセス
- インスタンスのスケールアップ・数の増減が簡単に可能
- 死活監視や負荷分散など非機能要件のほとんどをサポートする
- MySQL等のミドルウェアや自前のサービスをカタログサービスとして、高速に提供できる

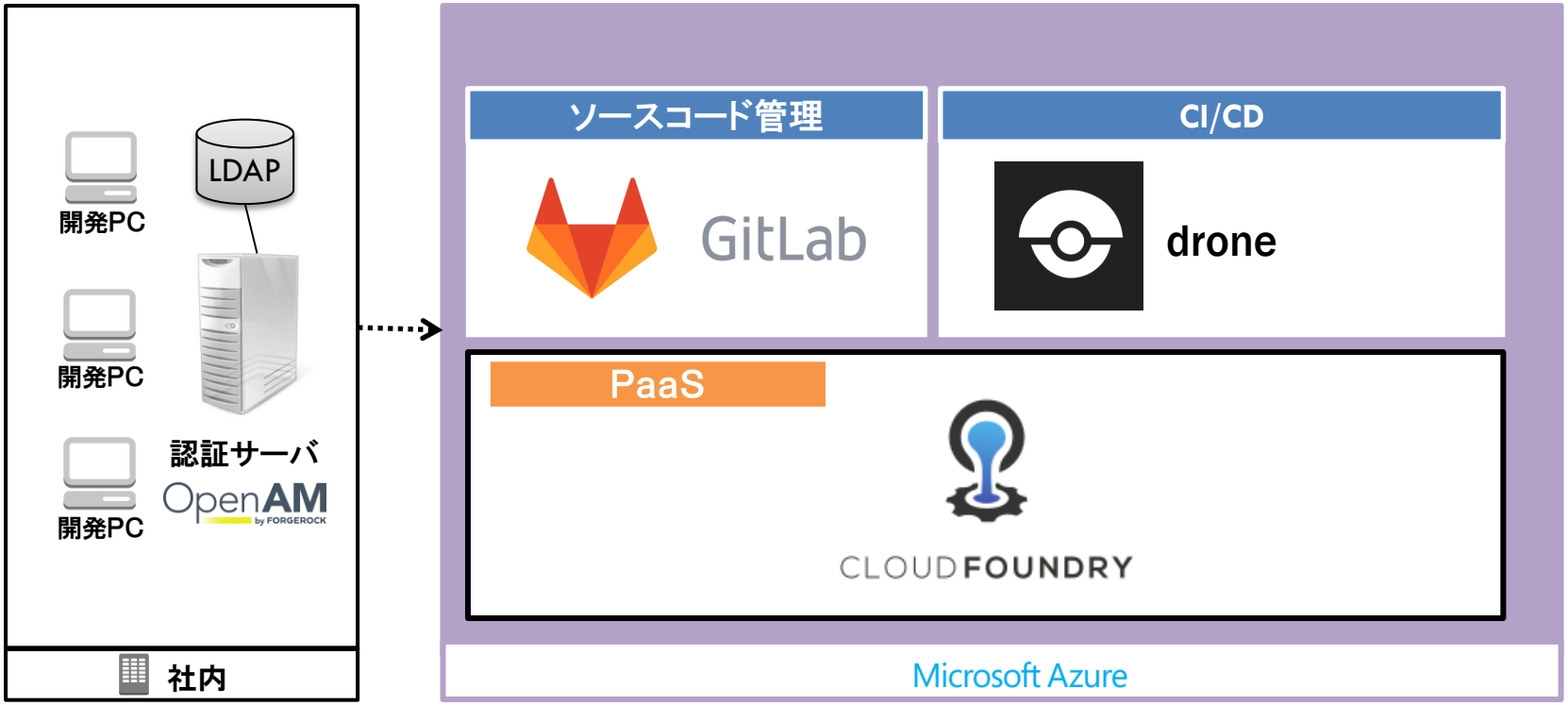
CLOUDFOUNDRY



1-6 構成(2016/03時点)

HITACHI
Inspire the Next

- IaaS基盤としてMicrosoft Azureを利用(日本リージョン)
  (パブリッククラウドを利用するための申請は、基盤運営チームでまとめて実施)
- 当社プロキシサーバからのアクセス(HTTPS)のみを許可、それ以外のアクセスをブロック
- 各サービスへの認証処理は社内のIDPサーバを利用して、LDAP+SAML認証でSSOを実現



HITACHI
Inspire the Next

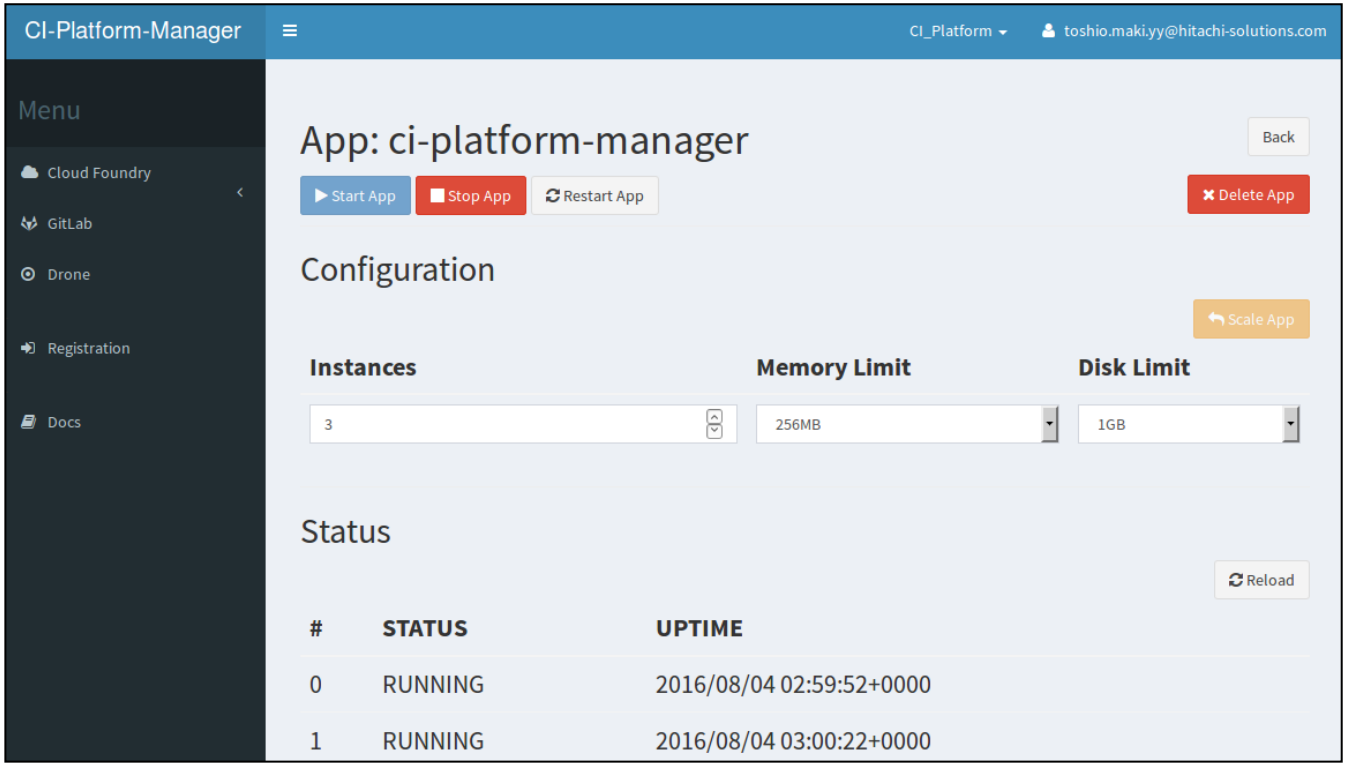
2. 本プラットフォーム上におけるDevOpsの実践

2-1 本プラットフォーム上におけるDevOpsの実践

HITACHI
Inspire the Next

プラットフォーム上で何ができればよいのかを検証するため、DevOpsを実践しながら構築

- プラットフォーム上で稼働する管理ポータルやサービスの開発
  - アプリケーションの起動・停止やリソース管理
  - Cloud Foundryのマネージドサービスの管理とアプリへのバインド処理
  - Cloud Foundry上で提供するサービスの開発
- アジャイル開発　2週間を1イテレーションとして、実装する機能を決定

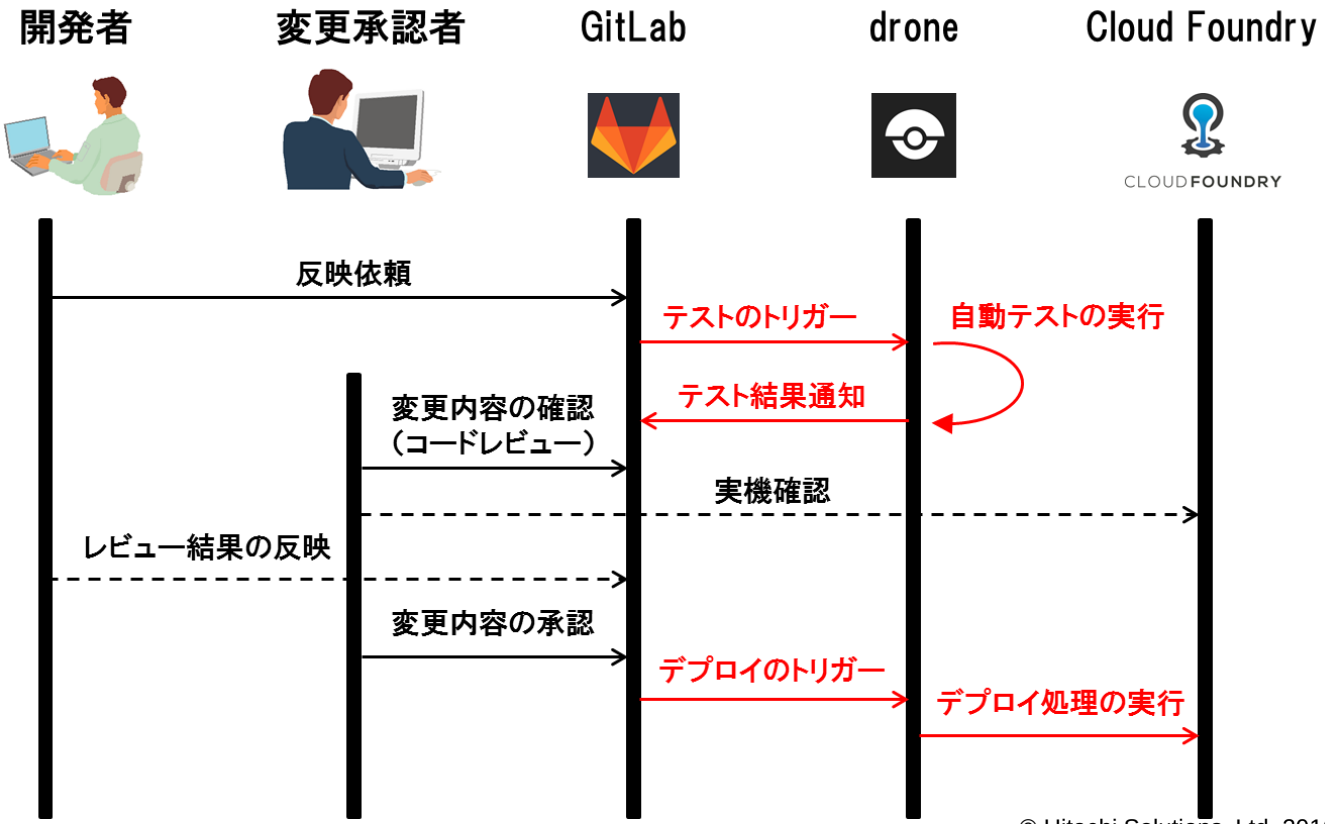


2-2 コード取り込みのフローを決定

HITACHI
Inspire the Next

GitLabのマージリクエストを中心とした、今時のフローを採用

- Git-Flowの開発方式を採用し、develop・masterブランチへの直接pushを禁止
- GitLabのマージリクエスト機能を使って、作成した機能を提出する
- コードレビューは自分以外の誰かが行う
- マージリクエストで承認したら、ステージング環境に自動デプロイされる
- ステージング環境でテストして問題なければ、本番環境へのマージリクエストを出す



2-3 CI/CDのサイクルの流れを作成

HITACHI
Inspire the Next

.drone.yml 1.01 KB

```
cache:
  mount:
    - vendor/bundle
clone:
  skip_verify: true
build:
  image: "[redacted]"
  environment:
    - LANG=C.UTF-8
  commands:
    - bundle install --path=vendor/bundle --without production
    - bundle exec rubocop --fail-level=W
    - bundle exec rake db:migrate
    - bundle exec rake db:test:prepare
    - bundle exec rspec spec/
deploy:
  cloudfoundry:
    image: "[redacted]"
    api: "api.[redacted]"
    user: "ci-platform-manager"
    password: "password"
    org: "CI_Platform"
    space: "CI_Platform"
    name: "ci-platform-manager-$$BRANCH-$$BUILD_NUMBER"
    when:
      branch: "!master"
  cloudfoundry:
    image: "[redacted]"
    api: "api[redacted]"
    user: "ci-platform-manager"
    password: "password"
    org: "CI_Platform"
    space: "CI_Platform"
    name: "ci-platform-manager"
    when:
      branch: "master"
```

ビルドプロセスを記述
- ベースとなるDockerコンテナの指定
- ソースのビルド
- 依存ライブラリの取得
- Rubocopによるソースコードのチェック
- db:migrateの実行
- RSpecの実行

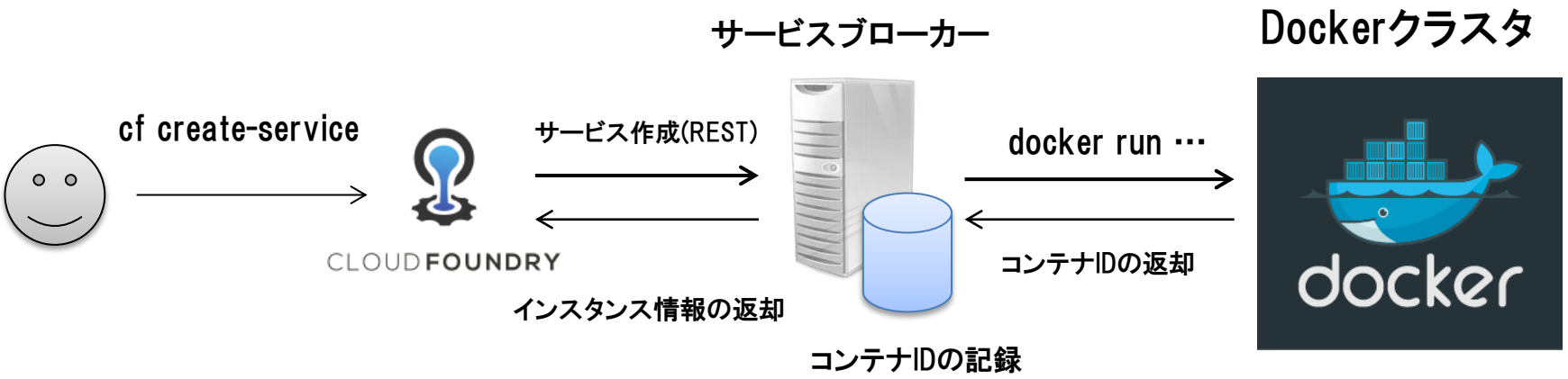
デプロイプロセスを記述
- DroneのCloud Foundryプラグインを利用
- Ruby on Railsはディレクトリをpushすればok
- masterブランチが変更されたとき
  - 本番環境へcf pushする
- それ以外のブランチが変更されたとき
  - ビルド番号を付与してcf pushする

2-4 Docker Service Brokerの実装

HITACHI
Inspire the Next

DockerコンテナをCloud Foundryに統合する仕組み

- 予めCloud Foundryのサービスとして動作させたいDockerイメージを登録
- Cloud Foundry上でサービスインスタンスを作成するコマンドが用意されている
  - 実行するとCloud FoundryからサービスブローカーにREST APIを投げる
  - APIを受けてサービスブローカーが、コンテナをDockerクラスタに配置する
  - インスタンスへの接続情報をCloud Foundryに返す
- Cloud Foundryアプリケーションとサービスインスタンスのバインドは1コマンドで実行
- 管理ポータル機能の一機能として実装

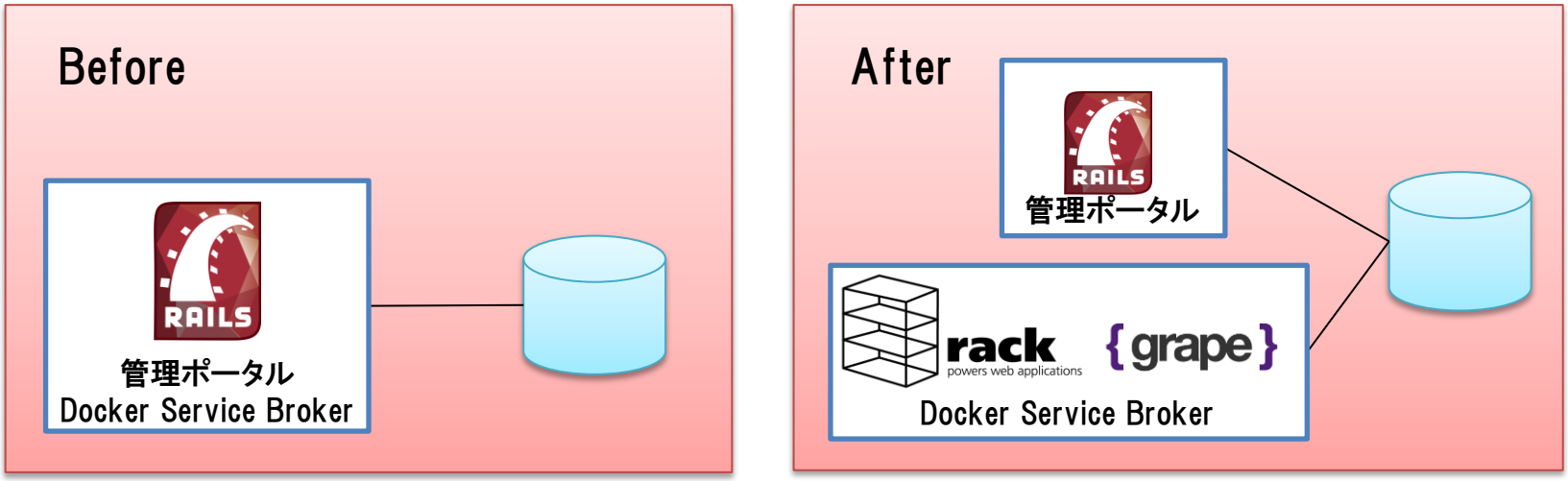


2-5 管理ポータルとDocker Service Brokerの分割

HITACHI
Inspire the Next

アプリケーションが大きくなってきたので分割したくなった

- 目的の違う管理ポータルとDocker Service Brokerとのコードの混在が気になってきた
  - ソースコードの可読性の観点
  - アプリケーションの可用性の観点
- プロジェクトを2つのリポジトリに分割
  - APIしか投げないDocker Service BrokerはRack＋Grapeに書き換えた
  - 管理ポータルのAPサーバをPassengerからPumaへ変更
- Request Specを書いていたため、テストケースが全部通るのを確認ながら分割

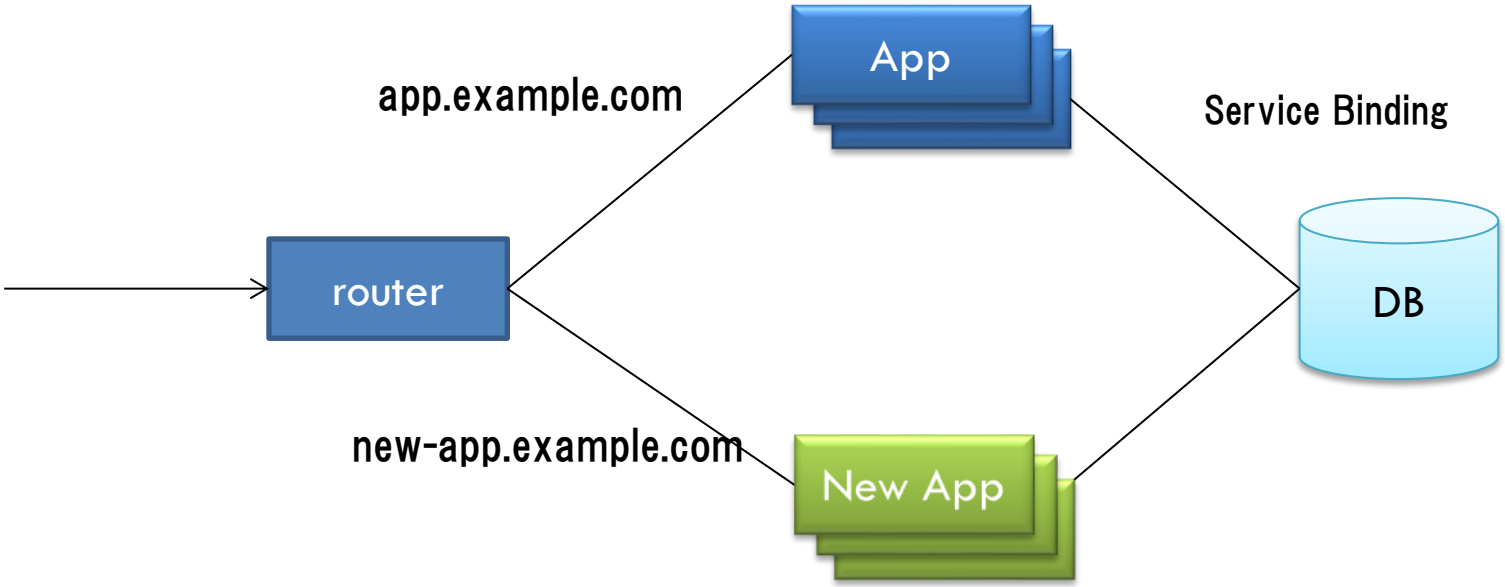


2-6 Blue-Green-Deploymentの実践

HITACHI
Inspire the Next

Blue-Green-Deployementとはアプリ切り替え時のダウンタイムを“0”にする技術

- マーチン・ファウラー氏の著書「Continuous Delivery」で紹介されている
- 一時的に新旧のアプリを混在させ、ルーティングを差し替える
- 問題があった場合は、古いアプリにルーティングを戻す
- 問題がなかった場合は、古いアプリを削除する
- Cloud Foundryの場合はcf push → cf map-route → cf unmap-routeの手順で実現可能

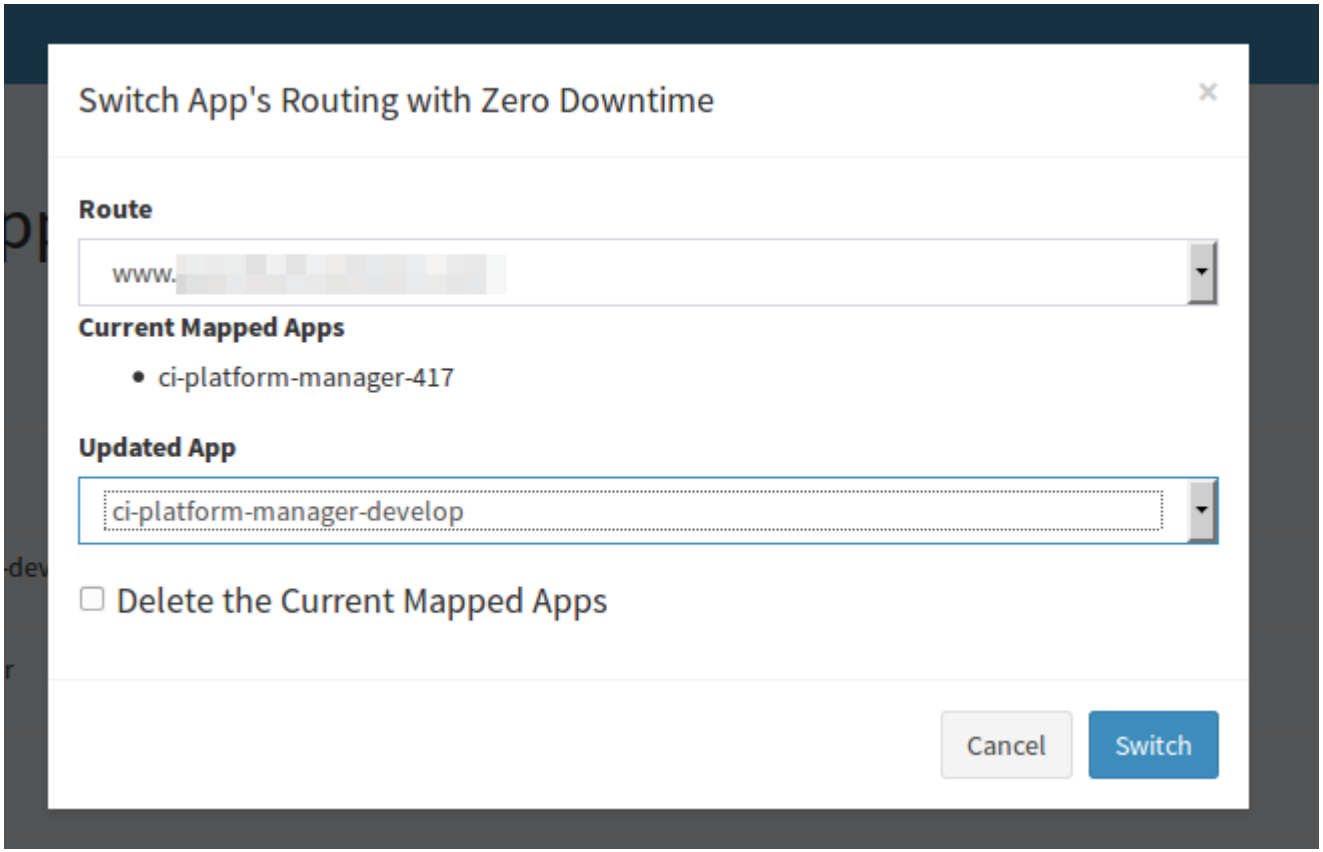


2-7 Blue-Green-Deployment機能の実装

HITACHI
Inspire the Next

コマンドラインから手動でも実現できるが、便利なので管理ツール側にも実装した

- Droneがデプロイする際に、アプリ名にDroneのビルド番号を付与して、デプロイ
- 指定したRouteに紐づいているアプリと、更新するアプリを切り替える

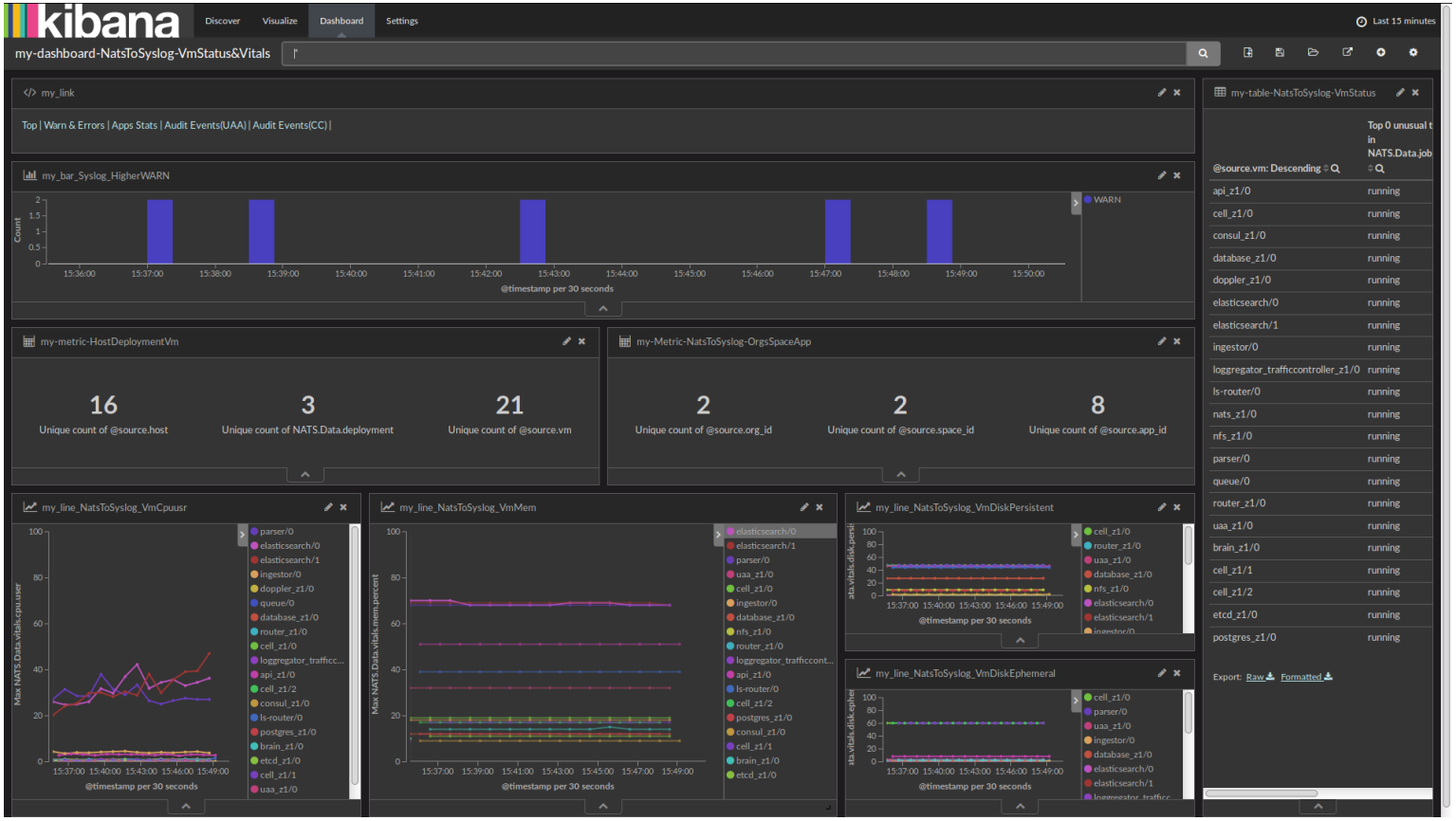


2-8 アプリケーションの運用監視

HITACHI
Inspire the Next

プラットフォーム上の動態監視を行うための仕組み作り

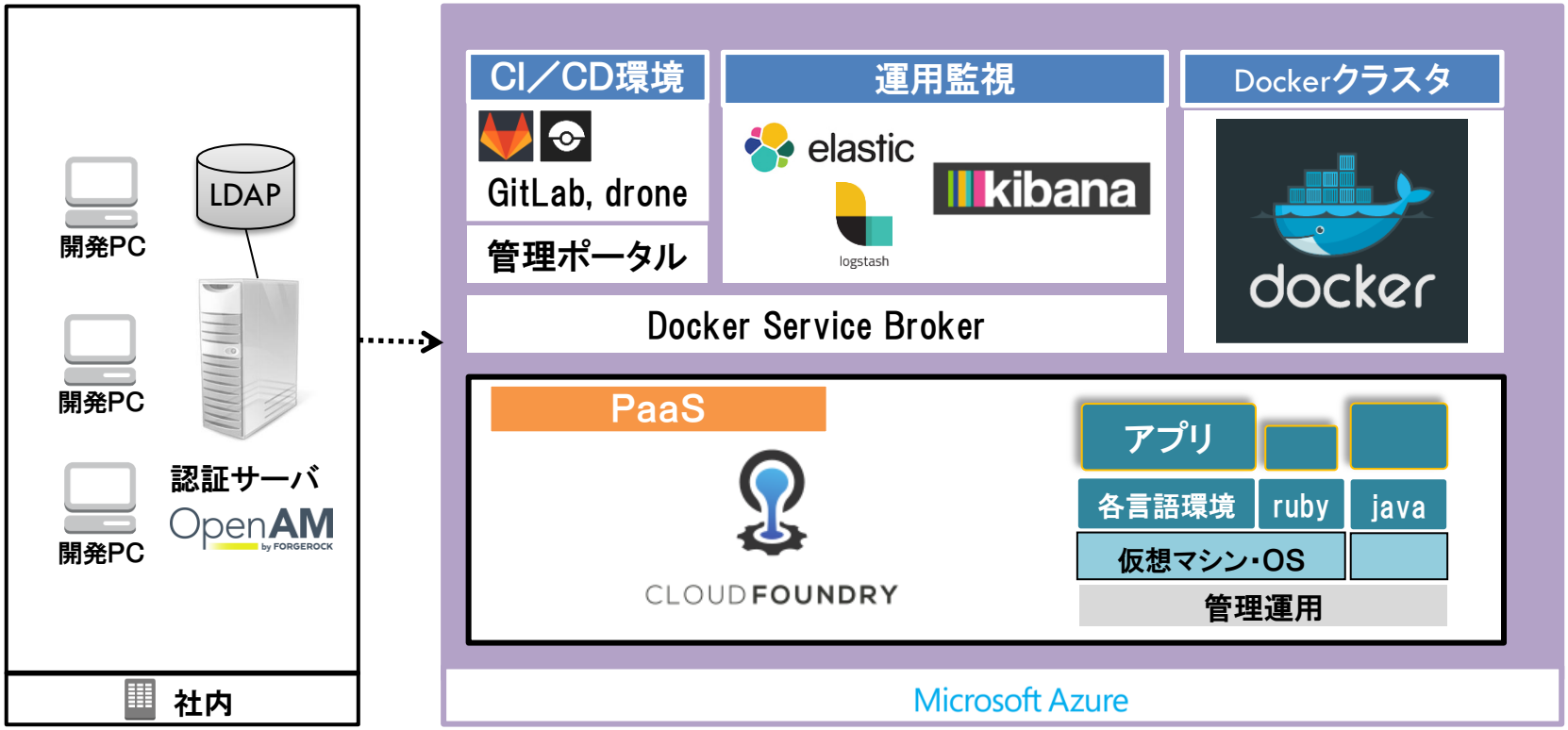
- Elasticsearch + Logstash + Kibana
- アクセスログやセキュリティ監査ログなどを統合
- アプリ側からは標準出力にログを出力するだけで、この仕組みに載るように構築
- それ以外のコンポーネントはsyslogから収集して統合



2-9 現行の構成

HITACHI
Inspire the Next

- GitLabでコード管理することで、コードの取り込みがシステムで管理できるようになった
- DroneをつかってCI/CDの基本的な仕組みが構築できた
- Blue-Green-Deploymentで、メンテナンス告知なしにアプリの更新ができるようになった
- Docker Service Brokerにより簡単にミドルウェアサービスを提供できるようになった
- ログ管理もプラットフォームに統合されたことで、運用が楽になった



HITACHI
Inspire the Next

3. 本プラットフォーム上におけるRubyの優位性

3-1本プラットフォーム上におけるRubyの優位性

HITACHI
Inspire the Next

- アーキテクチャ面での優位性
- 文化面での優位性

3-2 アーキテクチャ面での優位性

HITACHI
Inspire the Next

Ruby on RailsとThe Twelve-Factor App

- クラウドネイティブなアプリケーションを作るための方法論
  https://12factor.net/ja/
- Heroku社の創業者であるAdam Wiggins氏が提唱
- Cloud Foundry上で動くアプリケーションのベストプラクティスとして紹介されている
- RubyGemにもrails_12factorというgemが公開されている

THE TWELVE-FACTOR APP

はじめに

現代では、ソフトウェアは一般にサービスとして提供され、*Webアプリケーション*や*Software as a Service*と呼ばれる。Twelve-Factor Appは、次のようなSoftware as a Serviceを作り上げるための方法論である。

- セットアップ自動化のために **宣言的な** フォーマットを使い、プロジェクトに新しく加わった開発者が要する時間とコストを最小化する。
- 下層のOSへの **依存関係を明確化** し、実行環境間での **移植性を最大化** する。
- モダンな **クラウドプラットフォーム** 上への **デプロイ** に適しており、サーバー管理やシステム管理を不要なものにする。
- 開発環境と本番環境の **差異を最小限** にし、アジリティを最大化する **継続的デプロイ** を可能にする。
- ツール、アーキテクチャ、開発プラクティスを大幅に変更することなく **スケールアップ** できる。

Twelve-Factorの方法論は、どのようなプログラミング言語で書かれたアプリケーションにでも適用できる。また、どのようなバックエンドサービス(データベース、メッセージキュー、メモリキャッシュなど)の組み合わせを使っていても適用できる。

背景

このドキュメントへの寄稿者は、何百ものアプリケーションの開発とデプロイに直接関わり、Herokuプラットフォーム上での仕事を通して、何百何千ものアプリケーションの開発・運用・スケールに間接的に立ち会った。

このドキュメントは、多種多様なSaaSアプリケーション開発現場での私たちの経験と観察をすべてまとめたものである。これは、アプリケーション開発における理想的なプラクティスを見出すための三角測量である。特に、アプリケーションが時間と共に有機的に成長す

3-3 12 factor Appのフレームワーク別の対応状況

HITACHI
Inspire the Next

#	エッセンス	Ruby on Rails(Ruby)	Spring Boot(Java)	Struts(Java)
1	コードベース	◎	◎	◎
2	依存関係	○	○	△(Gradle/Maven)
3	設定	○	○	×
4	バックエンドサービス	△(rails_12factor)	○	×
5	ビルド、リリース、実行	◎	◎	◎
6	プロセス	△(activerecord-session等)	○	△(memcached-session-manager等)
7	ポートバインディング	○	○	○
8	並行性	○	△(Jetty等)	△(Jetty等)
9	廃棄容易性	○	×※	×※
10	開発／本番一致	◎	◎	◎
11	ログ	○(rails_12factor)	○	△(Log4j等)
12	管理プロセス	○	△(Flyway等)	△(Flyway等)

◎…DevOps開発プラットフォームでサポートされる
○…フレームワークレベルで対応
△…外部ツールで対応
×…フレームワークやツールでの支援はなく、自前で対応する必要がある

3-4 文化面での優位性

HITACHI
Inspire the Next

DevOpsは開発効率を上げていくための改善活動

- 改善していくためには、改善内容に明確なメリットが見えないといけない
- 常に改善するために、変化への適用が辛くならないように工夫する
- Rubyを採用しているプロジェクトには、変化に前向きなマインドがある
- Ruby以外のプロジェクトにも少しずつ考えを広めていきたい

ツール	求められる変化	工夫内容
GitLab	ブランチ管理ポリシーの導入 コードレビューの必須化	ブランチが増えすぎないように、管理するブランチを最小にする レビュアーを分散させて負荷が集中しないようにする
Drone	ビルドスクリプトの用意 テストコード文化の導入	カバレージ率を気にするより、まず少しずつ書くことから始める
Cloud Foundry	12factor Appの設計思想の導入	全てを一気に入れるのではなく、効果を見ながら少しずつ対応していく

3-4 まとめ

HITACHI
Inspire the Next

社内に限定された環境でDevOpsを実現する環境を構築した

- 社外のサービスを利用するのと同等レベルのことは実現できるようになった
- DevOpsを推進するための環境は整った
- 当社の製品やサービスなどをこの仕組みに載せ、サービスの拡充をめざす
- 今後は運用回りについてもノウハウを蓄積していく

DevOpsの実践を通して、Rubyの優位性を実感した

- 既にRubyを採用しているチームには導入障壁は高くないので、推進したい
- これらの文化に馴染みのないプロジェクトに推進していくために
  どうすればよいかは、今後の課題としたい

商標等

HITACHI
Inspire the Next

- CircleCIはCircleCI社の米国およびその他の国における登録商標または商標です。
- DockerはDocker Inc.の米国およびその他の国における登録商標または商標です。
- DroneはDrone.io社の米国およびその他の国における登録商標または商標です。
- Elastic、logstash、kibanaは、Elasticsearch BVの米国およびその他の国における登録商標または商標です。
- GitHubはGitHub Inc.,の米国およびその他の国における登録商標または商標です。
- GitLabはGitlab BVの米国およびその他の国における登録商標または商標です。
- Herokuはsalesforce.comの米国およびその他の国における登録商標または商標です。
- OracleとJavaは，Oracle Corporation 及びその子会社，関連会社の米国及びその他の国における登録商標です。
- OpenAMはオープンソース・ソリューション・テクノロジ株式会社の登録商標です。
- Pivotal Web Services、Cloud Foundry、Spring Bootは,米国および日本、その他の地域における Pivotal Software,Inc. の登録商標または商標です。
- Ruby on Railsは、David Heinemeier Hansson氏の米国およびその他の国における商標または登録商標です。
- Microsoft Azureは，米国Microsoft Corporationの米国およびその他の国における商標または登録商標です。
- その他記載の会社名、製品名は、それぞれの会社の商号、商標もしくは登録商標です。

HITACHI
Inspire the Next

END

DevOps開発プラットフォームの構築と
本プラットフォームにおけるRubyの優位性

2016/11/04

株式会社日立ソリューションズ
技術統括本部 技術開発本部 生産技術部

牧　俊男