



## 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

---

# 1. DevOps開発プラットフォームの構築

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、

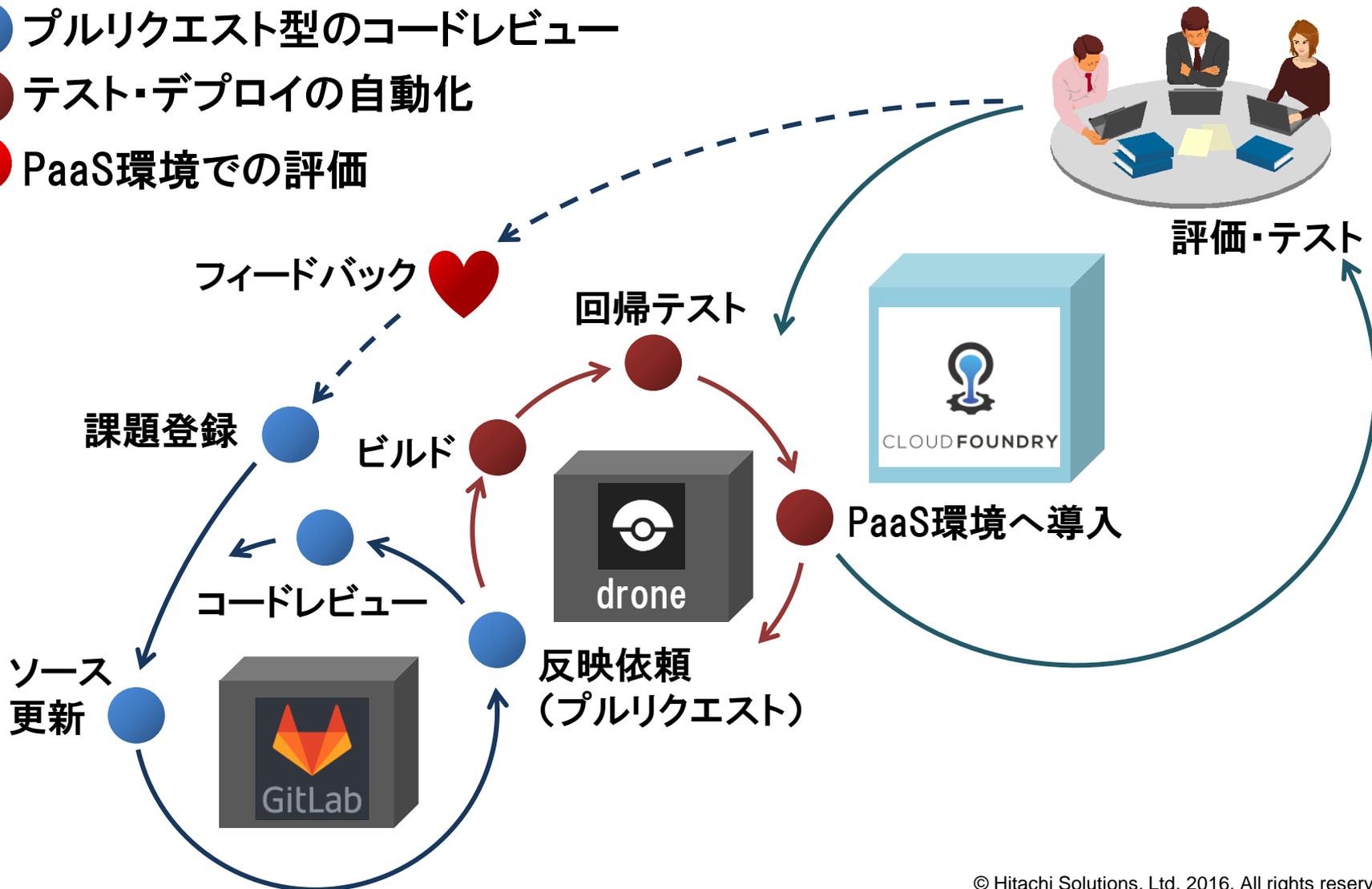
- 権利関係の問題 → 当社(もしくは当社顧客)の著作物を社外に置いてもよいか
- 機密情報保護の問題 → 何らかのミスで情報が流出する恐れがないか
- 輸出管理の問題 → 海外のサーバにソースコードを置くため、輸出管理を行う必要
- セキュリティの問題 → 誤って外部のサイトを攻撃してしまう可能性はないか

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

## 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



## GitHubと同等の機能を提供するソースコード管理ツール

- 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 Registryが統合され、Docker Imageのリポジトリとしても利用可能

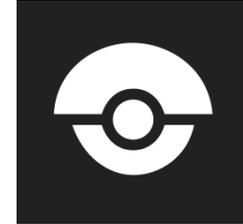


GitLab

Name	Last Update	Last Commit > 137e1fd6 - Merge branch 'feature/email_sent_workflow' into...	History
app	3 days ago	Merge branch 'feature/email_sent_workflow' into 'develop'	
bin	5 months ago	Support Service Broker for Docker Containers	
config	3 days ago	Modify MailConfig And Rspec.	
db	16 days ago	remove service broker api	
lib	2 months ago	Fix: To Get OAuth2 Token by SAML User	
log	6 months ago	Initial commit.	
public	6 months ago	Initial commit.	
spec	3 days ago	Merge branch 'feature/email_sent_workflow' into 'develop'	
test	10 days ago	remove unneeded files	
vendor/assets	24 days ago	Implement AdminLTE	
.cfignore	6 months ago	created .cfignore file for Cloud Foundry	
docker-compose.yml	22 days ago	use ruby:2.2.5 docker image for docker	

## 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



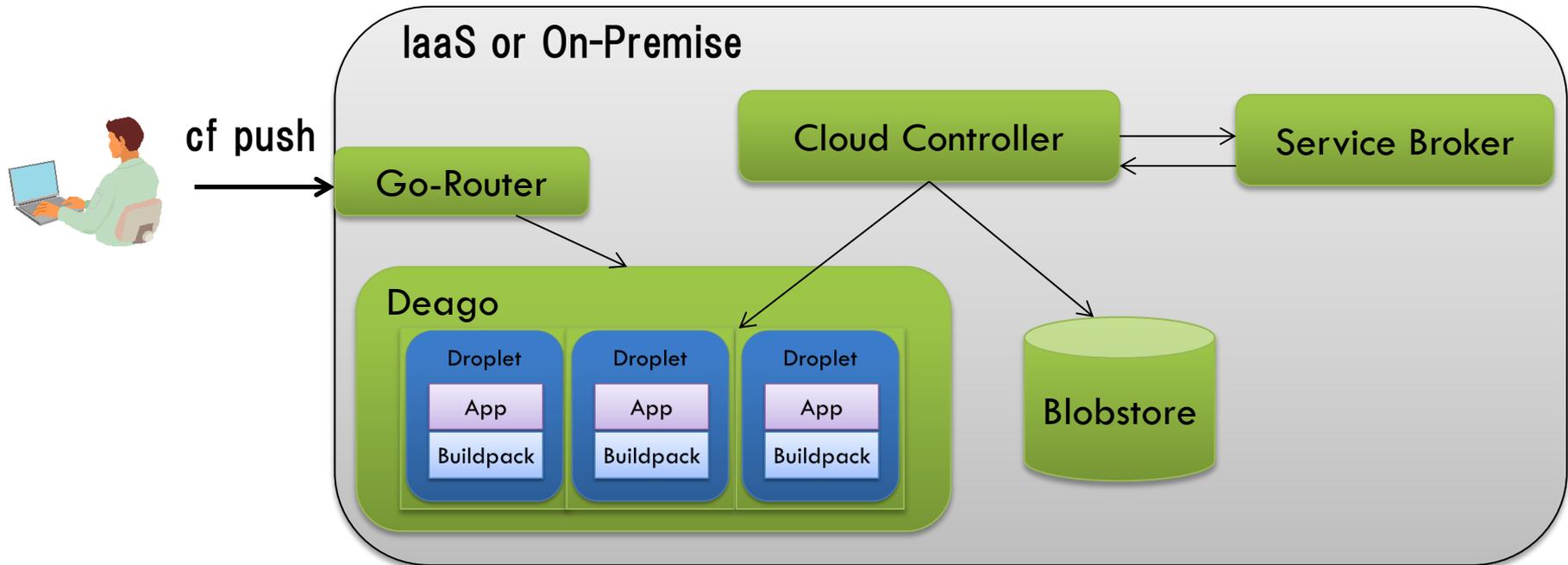
The screenshot shows a GitLab merge request interface. At the top, there's a search bar and a user profile icon. Below that, the path 'ci-platform / ci-platform-manager > 413' is visible. The main content area shows a merge request from 'Toshio Maki' 4 days ago. A green 'SUCCESS' badge is present, along with the text 'finished 4 days ago with exit code 0'. A 'RESTART' button is also visible. On the right side, there's a dark terminal window showing the output of a Drone CI pipeline. The terminal output includes information about pulling images for 'drone-cache:latest' and 'drone-git:latest', initializing a Git repository, adding a remote origin, fetching the 'develop' branch, resetting the working directory, and installing dependencies with 'bundle install'. It also shows linting results for Ruby files, with offenses such as 'Use nested module/class definitions instead of compact style' and 'Do not use block comments'.

## 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

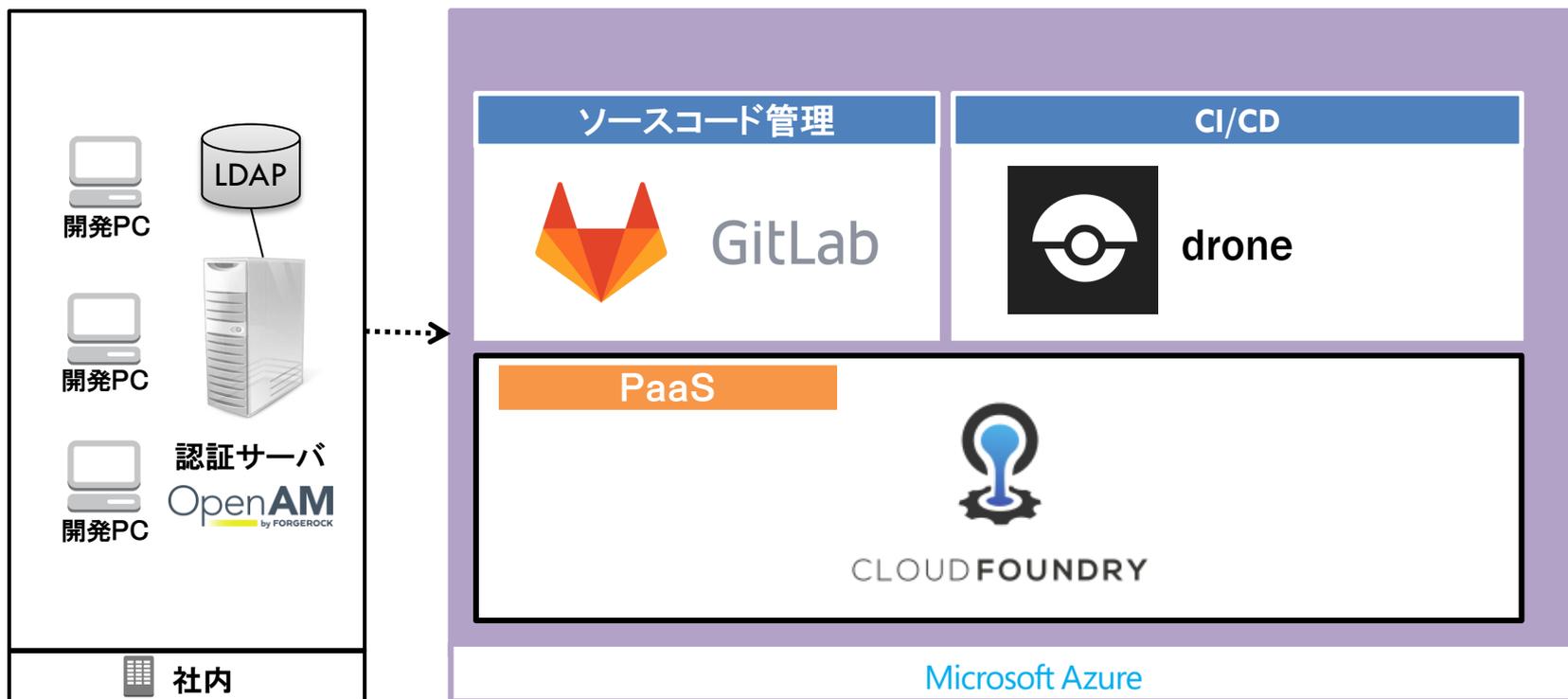


CLOUDFOUNDRY



# 1-6 構成(2016/03時点)

- 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



---

## 2. 本プラットフォーム上におけるDevOpsの実践

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

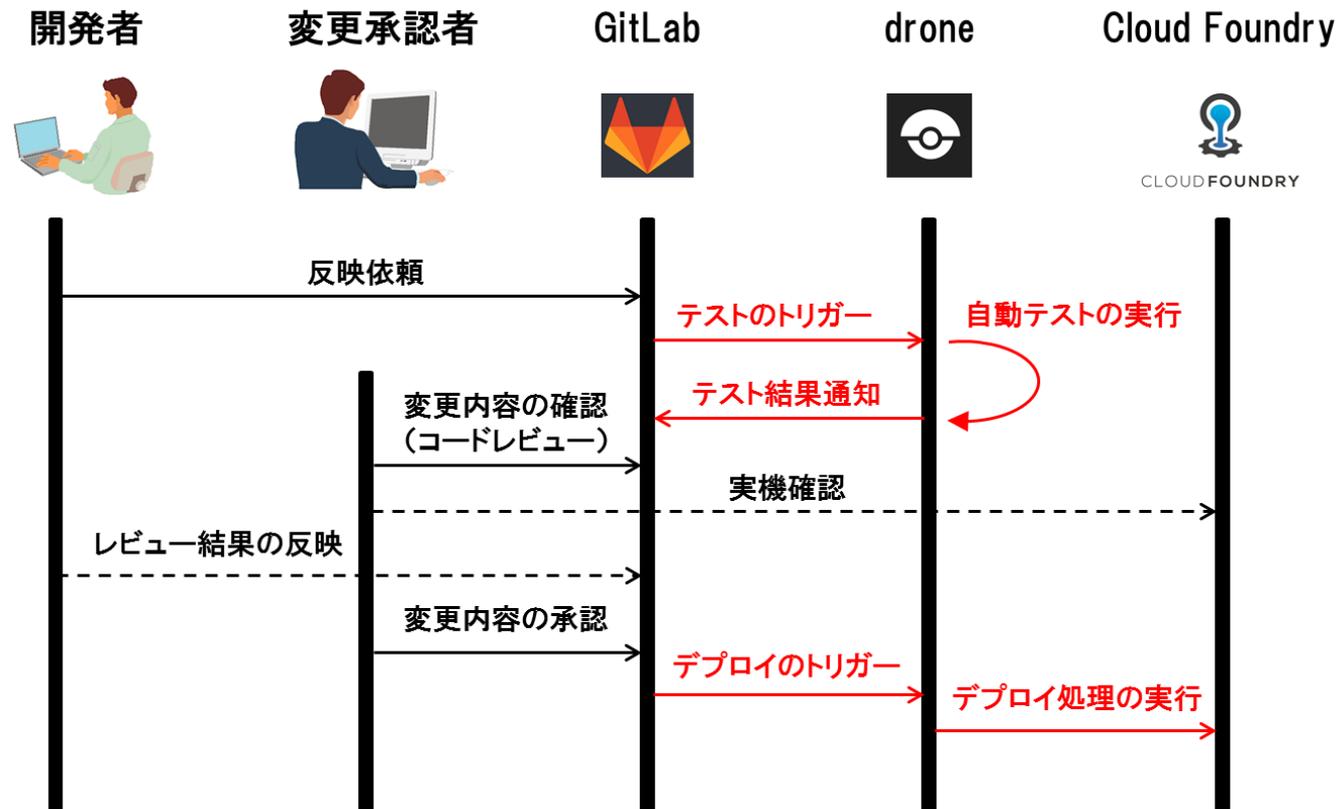
The screenshot displays the CI-Platform-Manager web interface. The top navigation bar includes the title 'CI-Platform-Manager', a hamburger menu icon, and the user profile 'toshio.maki.yy@hitachi-solutions.com'. A dark sidebar on the left contains a 'Menu' with items: Cloud Foundry, GitLab, Drone, Registration, and Docs. The main content area is titled 'App: ci-platform-manager' and features control buttons: 'Start App', 'Stop App', 'Restart App', and 'Delete App'. Below this is the 'Configuration' section with a 'Scale App' button and three input fields: 'Instances' (set to 3), 'Memory Limit' (set to 256MB), and 'Disk Limit' (set to 1GB). The 'Status' section includes a 'Reload' button and a table with the following data:

#	STATUS	UPTIME
0	RUNNING	2016/08/04 02:59:52+0000
1	RUNNING	2016/08/04 03:00:22+0000

## 2-2 コード取り込みのフローを決定

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

.drone.yml 1.01 KB

```
1  cache:
2    mount:
3      - vendor/bundle
4  clone:
5    skip verify: true
6  build:
7    image: "████████████████████████████████████████"
8    environment:
9      - LANG=C.UTF-8
10   commands:
11     - bundle install --path=vendor/bundle --without production
12     - bundle exec rubocop --fail-level=W
13     - bundle exec rake db:migrate
14     - bundle exec rake db:test:prepare
15     - bundle exec rspec spec/
16  deploy:
17    cloudfoundry:
18      image: "████████████████████████████████████████"
19      api: "api.████████████████████"
20      user: "ci-platform-manager"
21      password: "password"
22      org: "CI_Platform"
23      space: "CI_Platform"
24      name: "ci-platform-manager-$$BRANCH-$$BUILD_NUMBER"
25      when:
26        branch: "!master"
27    cloudfoundry:
28      image: "████████████████████████████████████████"
29      api: "api.████████████████████"
30      user: "ci-platform-manager"
31      password: "password"
32      org: "CI_Platform"
33      space: "CI_Platform"
34      name: "ci-platform-manager"
35      when:
36        branch: "master"
```

### ビルドプロセスを記述

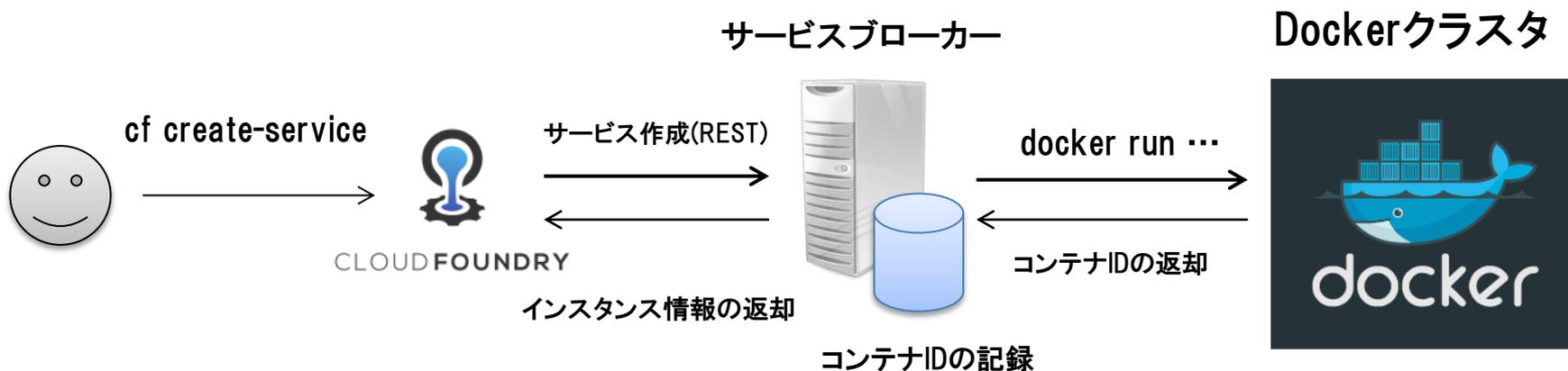
- ベースとなるDockerコンテナの指定
- ソースのビルド
- 依存ライブラリの取得
- Rubocopによるソースコードのチェック
- db:migrateの実行
- RSpecの実行

### 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

### DockerコンテナをCloud Foundryに統合する仕組み

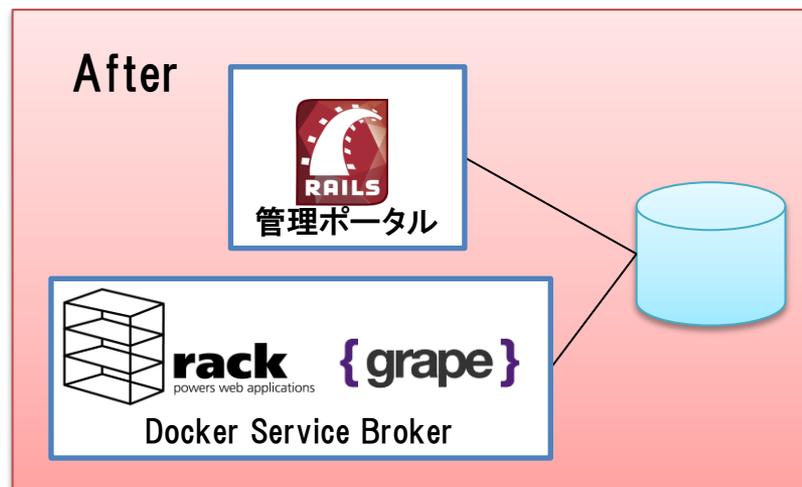
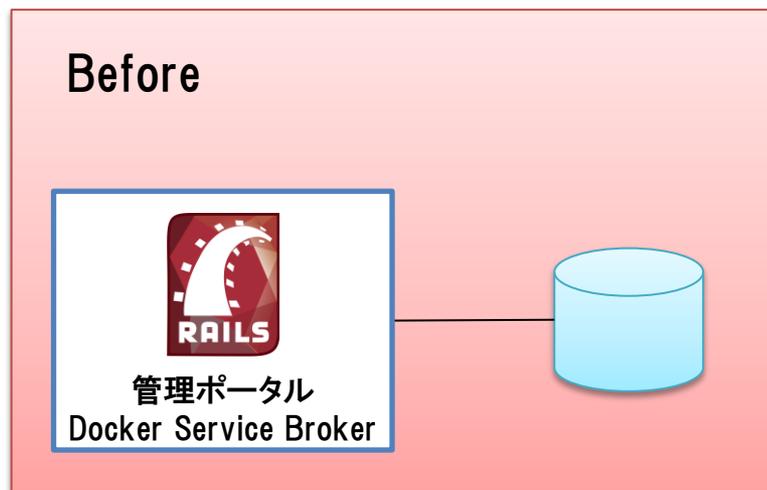
- 予めCloud Foundryのサービスとして動作させたいDockerイメージを登録
- Cloud Foundry上でサービスインスタンスを作成するコマンドが用意されている
  - 実行するとCloud FoundryからサービスブローカーにREST APIを投げる
  - APIを受けてサービスブローカーが、コンテナをDockerクラスタに配置する
  - インスタンスへの接続情報をCloud Foundryに返す
- Cloud Foundryアプリケーションとサービスインスタンスのバインドは1コマンドで実行
- 管理ポータル機能の一機能として実装



## 2-5 管理ポータルとDocker Service Brokerの分割

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

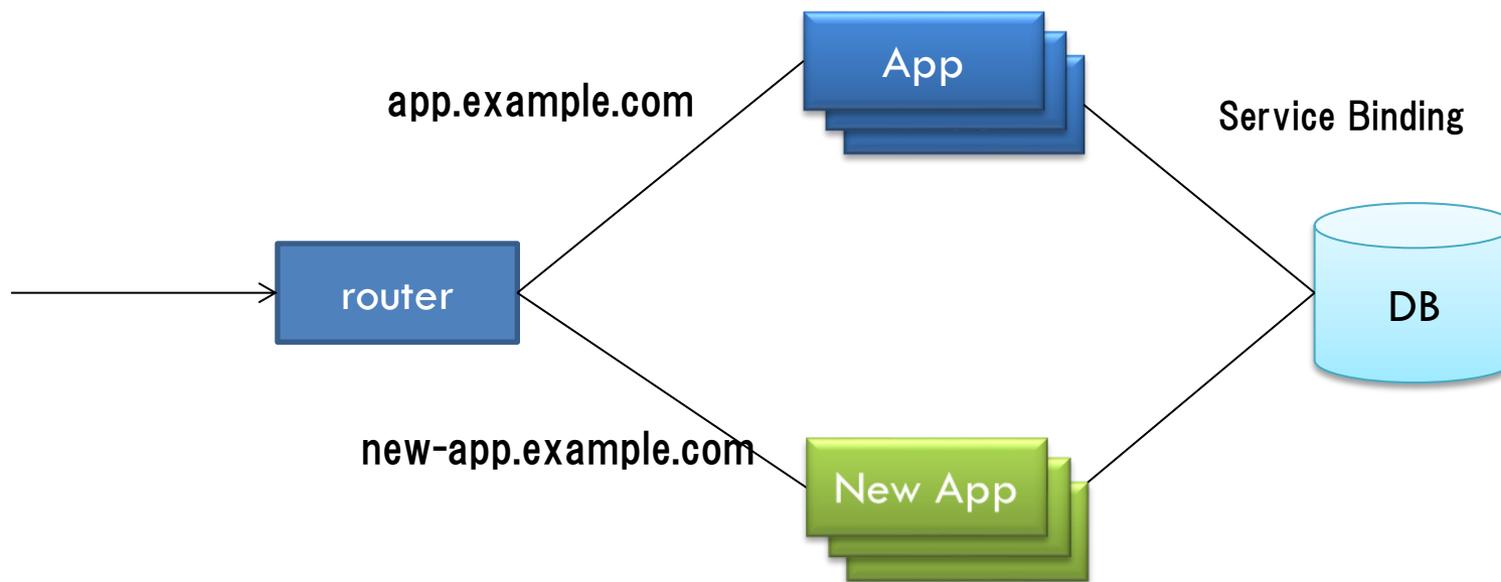
- 目的の違う管理ポータルとDocker Service Brokerとのコードの混在が気になってきた
  - ソースコードの可読性の観点
  - アプリケーションの可用性の観点
- プロジェクトを2つのリポジトリに分割
  - APIしか投げないDocker Service BrokerはRack + Grapeに書き換えた
  - 管理ポータルのAPサーバをPassengerからPumaへ変更
- Request Specを書いていたため、テストケースが全部通るのを確認ながら分割



## 2-6 Blue-Green-Deploymentの実践

Blue-Green-Deployment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

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

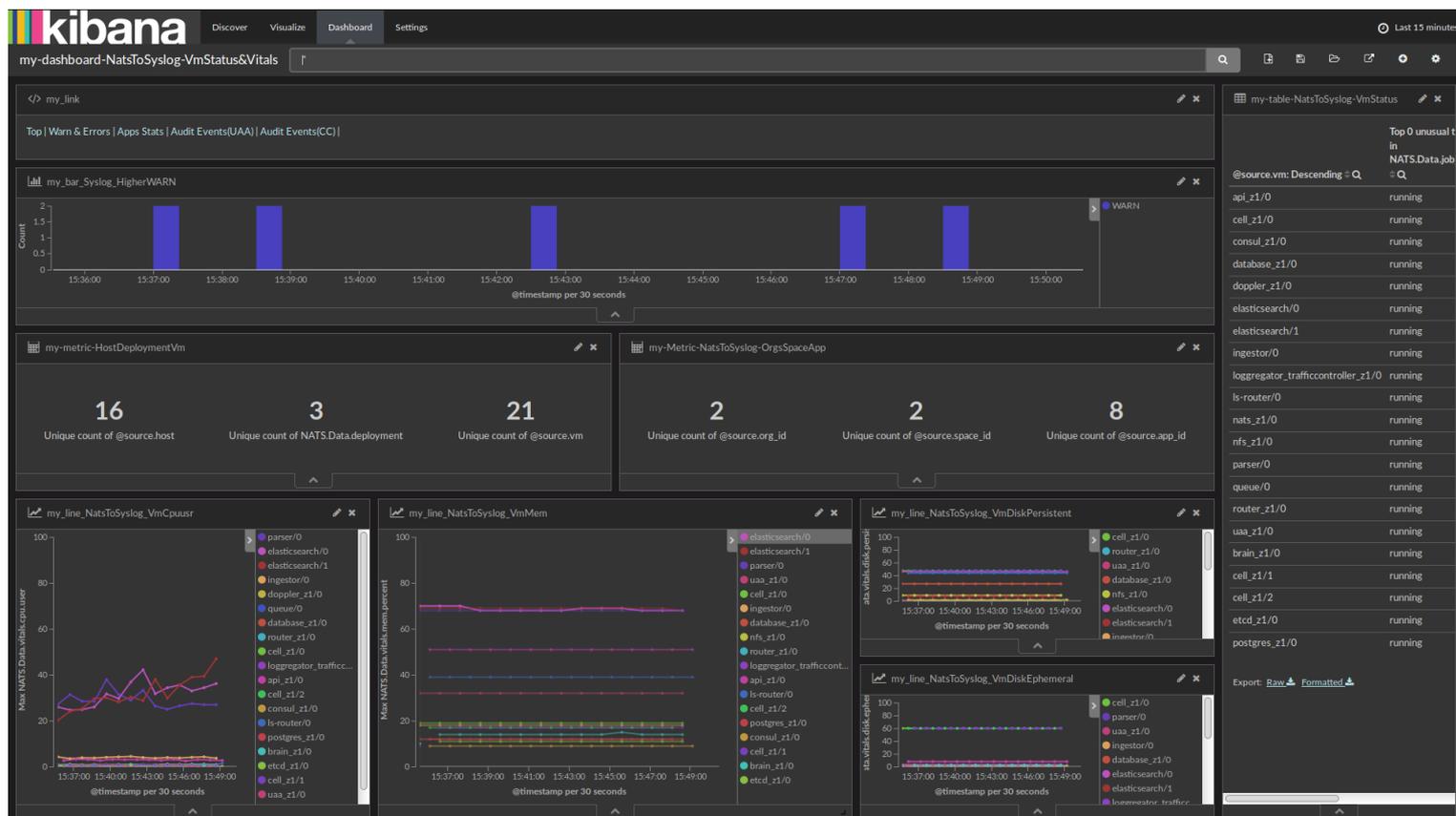
- Droneがデプロイする際に、アプリ名にDroneのビルド番号を付与して、デプロイ
- 指定したRouteに紐づいているアプリと、更新するアプリを切り替える

The screenshot shows a modal window titled "Switch App's Routing with Zero Downtime". It contains the following elements:

- Route:** A dropdown menu showing "www. [redacted]".
- Current Mapped Apps:** A list containing "ci-platform-manager-417".
- Updated App:** A dropdown menu showing "ci-platform-manager-develop".
- Delete the Current Mapped Apps**
- Buttons:** "Cancel" and "Switch".

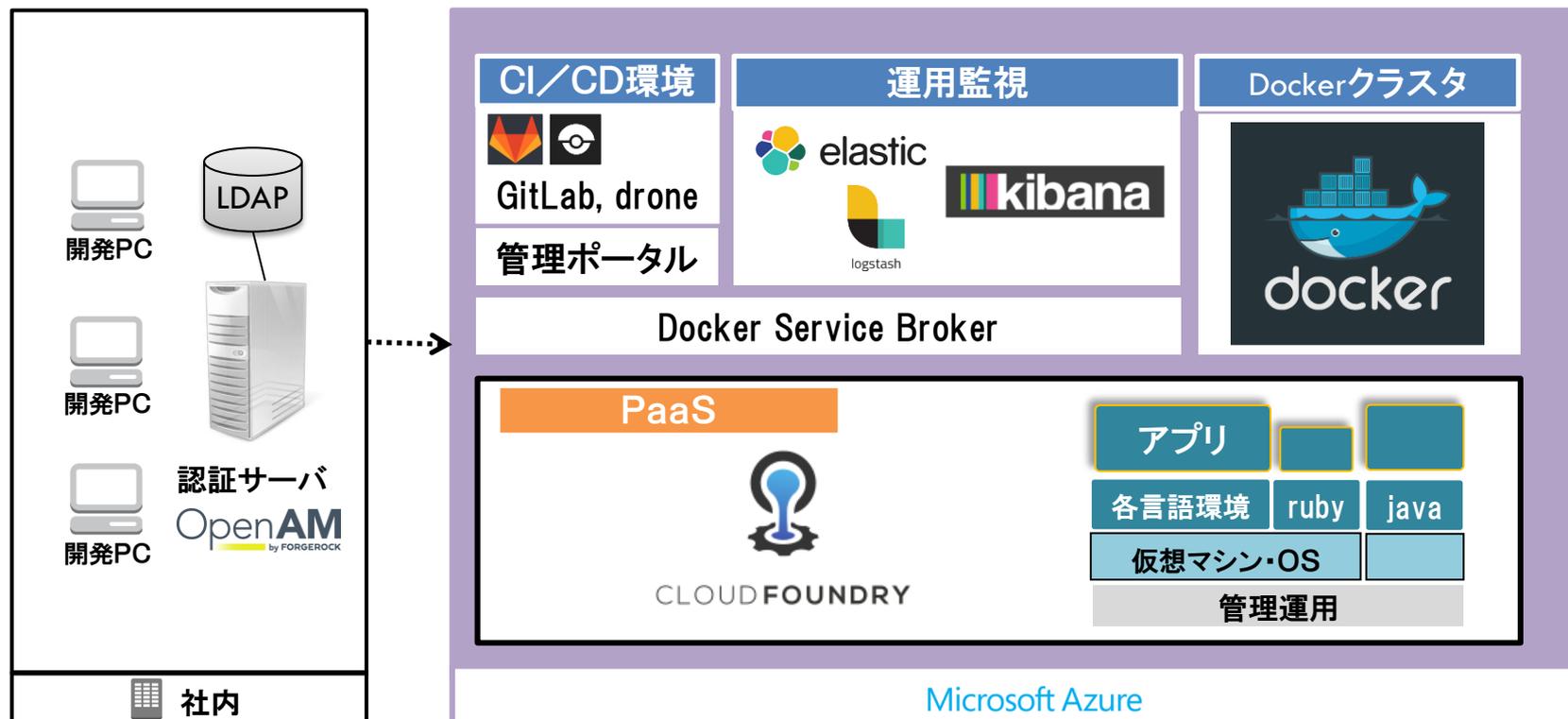
## プラットフォーム上の動態監視を行うための仕組み作り

- 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

- 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を使って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



---

### 3. 本プラットフォーム上におけるRubyの優位性

- **アーキテクチャ面での優位性**
- **文化面での優位性**

### 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



#### 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

#	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

### 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	カバレッ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

社内限定された環境でDevOpsを実現する環境を構築した

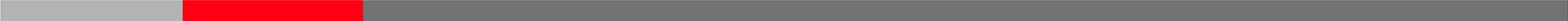
- 社外のサービスを利用するのと同レベルのことは実現できるようになった
- DevOpsを推進するための環境は整った
- 当社の製品やサービスなどをこの仕組みに載せ、サービスの拡充をめざす
- 今後は運用回りについてもノウハウを蓄積していく

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

- 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・テクノロジー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。

**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**