Empowered by Innovation NEC

Okinawa Open Days 2014

OpenStackによるサービス基盤構築の実際 ～事例とDevOpsへの取り組み～

2014/12/11

NECソリューションイノベータ株式会社

自己紹介と本日のお話

塩谷 幸治（しおたに こうじ）

NECソリューションイノベータ（株）

【本日のお話】

OpenStackによるIaaS基盤にDockerを組み合わせたフルOSSによるサービス基盤と、その上でDevOpsによるサービス提供・運用を行うシステムの実現性、優位性や課題など事例を交えお話しします。

⇒SaaS等のサービスを提供する事を目的として構築した基盤のお話です

本スライドは配布用です。
セッションで使用するスライドから一部割愛していますことをご了承ください。

Agenda

1. OSS適用の考え方
2. OpenStackによるサービス基盤
3. DevOpsへの取り組み

免責事項

- 本資料に掲載されている会社名、製品名、ロゴ等は一般に各社の商標または登録商標です
- 本資料中では「™」「®」「©」は省略しています

1. OSS適用の考え方

1．1．OSS概観～技術開発プレイヤーの変化

クラウドベースの大規模Webサービス登場とともにインフラ構築やシステム開発手法、および必要な機能が大きく変化

- 世界規模でサービス提供を行うネット企業を中心に技術革新が生まれる時代へ

人智を超えた規模感への対処

自社ビジネスの課題解決のために技術開発を開始

【ネット企業】
Google、Amazon、Twitter、Facebook、など

【課題】
・大規模分散処理
・大量データの蓄積/分析
・大量のサーバを管理するための運用技術、など

エンジニアがネット企業に集結

新たなIT技術の起点がITベンダからネット企業へ移動

【ITベンダ】
企業向けシステム技術中心で、ネット企業が必要とする大規模分散処理で出遅れ。

【ネット企業】
自社エンジニアによる技術革新でネットビジネスを開拓、先端技術開発をリード。

OSSエコシステムの拡大

ITベンダがOSSコミュニティとの共創関係にシフト

OSSコミュニティを軸に開発者やITベンダ、ユーザなどが結び付いてエコシステムを形成、共に成長していくモデルが出現。
様々なOSSでエコシステムが形成され、水平的な協力関係を重視する時代に。

1．1．OSS概観～OSSが拡大し続ける理由

クラウド環境におけるライセンスコストの削減要求

- 数千～数万台のサーバ環境では、SWライセンスコストの問題からOSS活用もしくは自社開発ソフトウェアを利用せざるを得ないケースが多い

ビジネス環境の変化に対する開発スピードの要求

- 多数のエンジニアが参加するOSSコミュニティが持つ力（多種多様な考え方やスキルセット）を活用することで圧倒的にスピーディーな開発が可能

OSS公開することによる自社サービスの差別化

- 顧客に対してオープンであり、ソースコードをはじめとした提供技術の詳細を公開することで、逆に自社サービスを差別化、SWメンテナンスコストを削減
- 囲い込み戦略からOSSエコシステムへの参加による顧客の信頼獲得と将来の発展性の確保

GitHubによるOSS開発PJのソーシャル化

- オープンソースの開発者が自由に集まり、議論、開発する環境が確立

1．1．OSS概観～企業 vs OSSコミュニティ

企業が開発パワーでOSSコミュニティに対抗することは事実上不可能

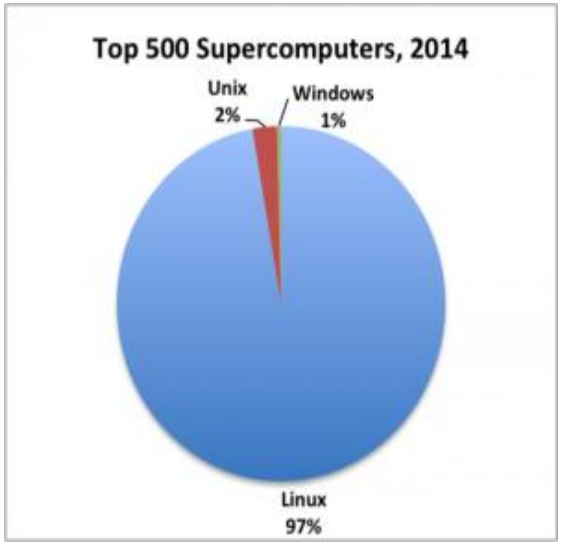
- コミュニティの開発パワーを活用することが重要

2004年

最速コンピュータTOP10のうち
5つが商用UNIX

2014年

最速コンピュータTOP500の
97%がLinux



出典：http://www.top500.org/lists/2014/06/

【OSSコミュニティ】

- OSSはコミュニティによって支えられる
- OSSの安全性は『数』によって支えられる
  ⇒コミュニティのエンジニアによるレビューが
  リスクを低減させる
- コミュニティは成果物の品質を向上させ、
  アドオンを作るコミュニティ生成を促し、エコ
  システムを成長させる

1．1．OSS概観～開発スタイルのソーシャル化（1／2）

～2005年までのOSSプロジェクト

- LinuxやApache、MySQLなど大規模PJが中心
  - センタ集中型のバージョン管理サービスを利用してMLに投稿されるパッチをメンテナーがソースコードに適用する形でプロジェクトを遂行

2005年12月 分散型バージョン管理システム・Gitリリース

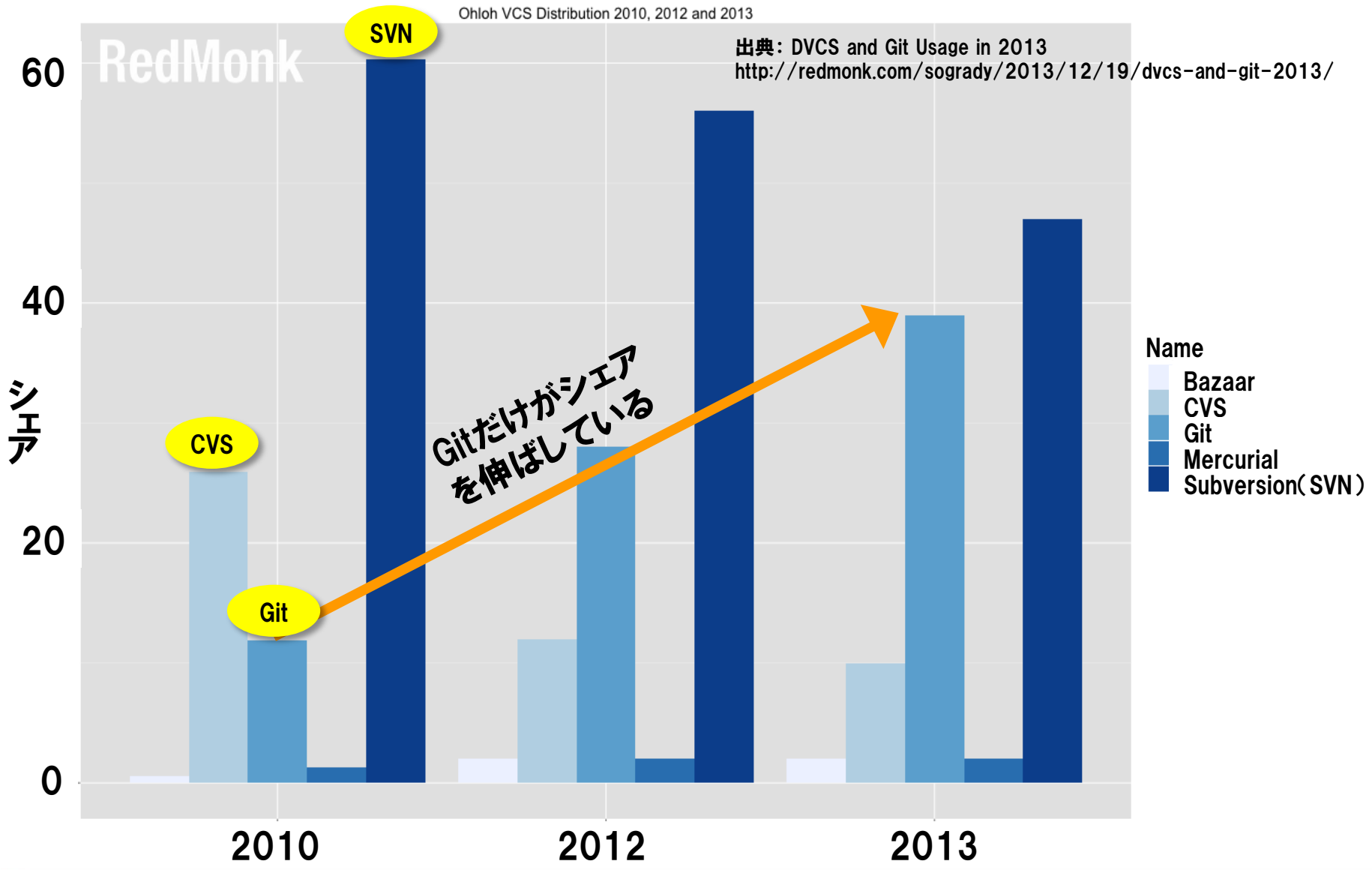
- Linux開発者 Linus Torvaldsが開発
  - 各開発者がセンタリポジトリをローカルリポジトリに複製（git clone）
  - ローカルで改修作業を実施、ローカルリポジトリに変更履歴を記録（git commit）
  - センタリポジトリの変更取込（git pull）やセンタリポジトリへの反映（git push）

2008年4月 ソーシャルコーディングサービス・GitHubリリース

- GitHub社が提供する開発PJ向けの共有Webサービス
  - GitをベースにSNS要素をプラス、コードに興味を持った人が集まって議論、改善するための機能を提供
    - Git、ユーザフォロー、スター、issueツリー、プルリクエストツリー、Wikiなど
    - PJを公開すれば無料で利用できる（非公開PJは有償）

個人によるソース公開や他開発者のソース改修がより簡単に！

参考：バージョン管理システムのシェア推移



1．2．自部門PJチームの状況（１／２）

課題 付加価値あるコア技術を持ち、高いQCDを実現しなければクラウドマーケットで戦えない・生き残れない時代にどう対処していくか？

技術

この提案でうまくいくという**根拠**は？

HWスペックやSWスタックの選択理由は？

品質（Q）

なぜ繰り返し**作業ミス**が起こるの？

担当者が変わると品質も変わるよね

コア

御社の**強み**は何ですか？

独自技術やAP開発技術、製品は？

コスト（C）

この見積ってただの**工数積み上げ**？

人が多ければいいシステムができるの？

納期（D）

よその会社は**3ヶ月**で出来るって

なぜこんなに時間がかかるの？

1．2．自部門PJチームの状況（2／2）

仮説 従来開発手法やプロセスの競争力は低下する。エンジニアがマインドシフトし、新たなやり方を獲得すれば状況を打開できるはず。

従来SI’er思考	クラウド的思考	マインドシフト
所有する	利用する	極力コードを書かない（OSS/アセット活用と自動化）
オーダーメイド	レディメイド	
ウォーターフォール	アジャイル	スピードを重視する（継続的インテグレーション）
机上設計	実機検証	
完成開発	段階的拡張	継続的な改良と機能追加（永遠のベータ版）
スケールアップ（垂直）	スケールアウト（水平）	
開発と運用の分離	開発と運用の一体化	開発と運用を統合する（DevOps）
システムサポート	ライフサイクルサポート	

クラウド時代はMTBFよりMTTRを重視する思考へ
（『壊れないシステム』から『素早く修正できるシステム』へ価値シフトへの追随）

1．3．OSS活用にあたっての準備

動作ふるまい検証（ホワイトボックス化）

- お客様に提供もしくはご利用いただく場合、OSSプロダクトの実力値を見極める必要がある

検証視点

アーキテクチャ	・機能要素とプロセス／スレッド構造を明らかにする
動作・振る舞い	・内部処理のシーケンスを明らかにする ・CPUやメモリ、ディスクI/Oなどリソース状況を把握する
MC性	・MC（ミッションクリティカル）性を検証する 性能性／拡張性／可用性／連携性／機密性／運用性

まとめ

クラウド環境においてOSS活用はMUST

- 技術革新はOSSによって生み出されている
  - コミュニティの開発パワーを活用する
  - コミュニティの開発スタイルでよい部分は積極的にPJへ取り込む
- ただし、OSSプロダクトの動作振る舞い、実力値を見極めた上で、適材適所で活用する必要がある
  - 動作振る舞い解析は障害時にも役立つ
  - 適用領域を見極めるにはMC性検証が必要

2. OpenStackによるサービス基盤

2．1．インフラ構築技術の進化

仮想化をベースとしたSoftware-Defined Infrastructure(SDI)の登場

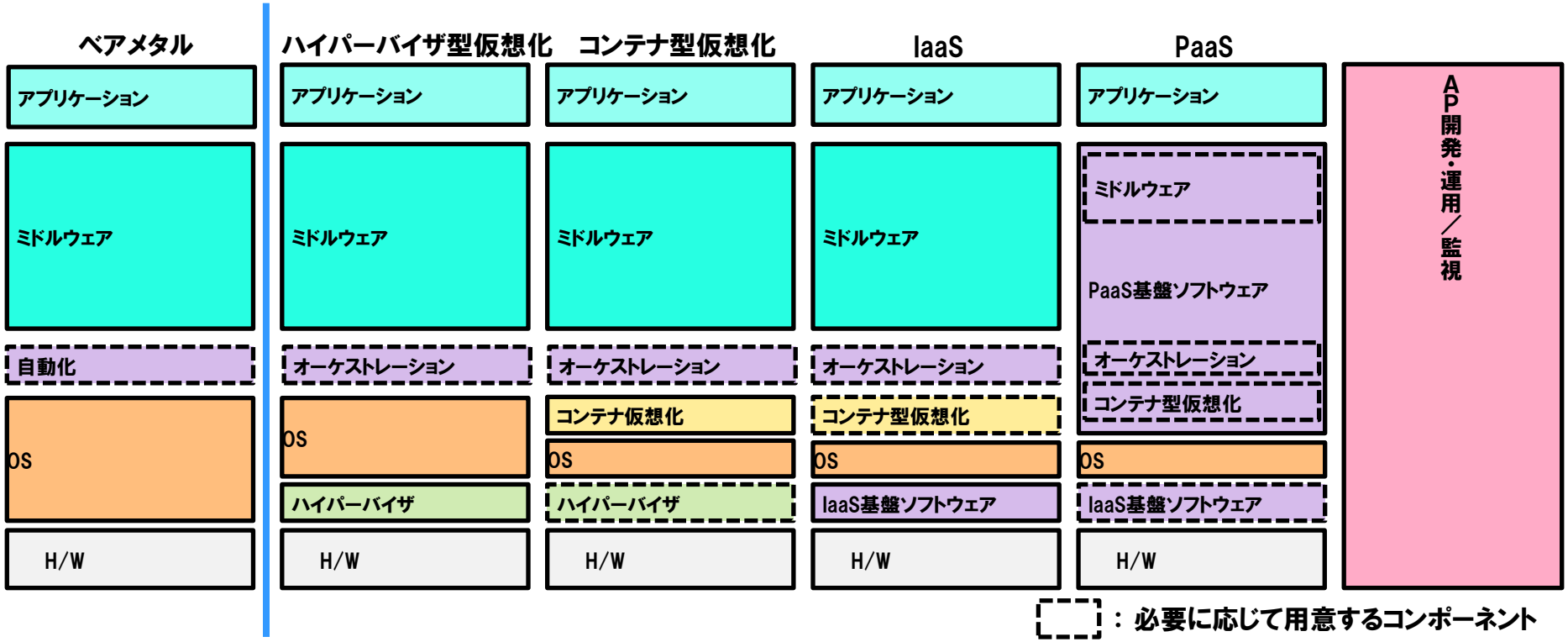
- 必要なインフラを迅速に用意するために物理リソースを論理リソース化
  - コンテナ型仮想化とHypervisor型仮想化
- 物理リソースから論理リソースまでを一元管理する運用/監視技術
  - 要求に応じて必要な論理リソースを一挙に組み合わせるオーケストレーション技術
  - 必要なソフトウェアスタックを組み上げる自動化技術
    - インフラ構築作業のコード化、自動構築とテスト
  - 動的に変化する周辺環境を制御するサービスコーディネート技術
- SDIをパッケージ化するIaaS基盤/PaaS基盤ソフトウェア

これらの技術をベースとした「使い捨てインフラ」を意味する概念である
『Immutable Infrastructure』の登場
（インフラ更新はリスクが高いため新たに作り直す方向へシフト）

2．2．サーバ基盤の変遷

仮想化技術をベースとして様々なPF基盤のバリエーションが登場

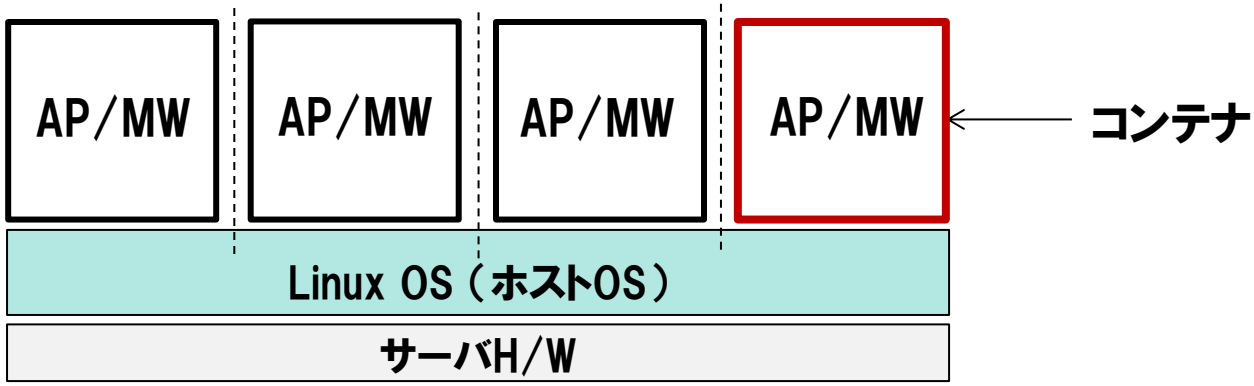
- **オンプレミスでもクラウド的な分散環境への流れが加速**
  - 多数のサーバを管理する必要性から、人的作業を軽減するためにPF構築自動化技術やオーケストレーション技術、サービスコーディネート技術が発展



2．3．コンテナ型仮想化とHypervisor型仮想化（１／２）

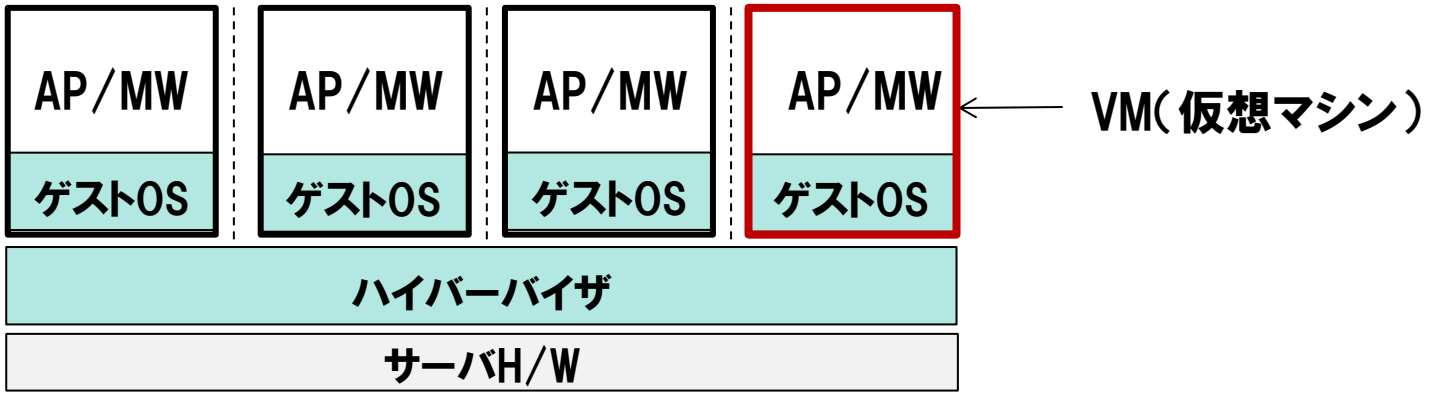
コンテナ型仮想化

- OS上にリソースが分離されたコンテナを構成する



Hypervisor型仮想化

- H/W上に複数のOS（VM（仮想マシン）を構成する



2．3．コンテナ型仮想化とHypervisor型仮想化（２／２）

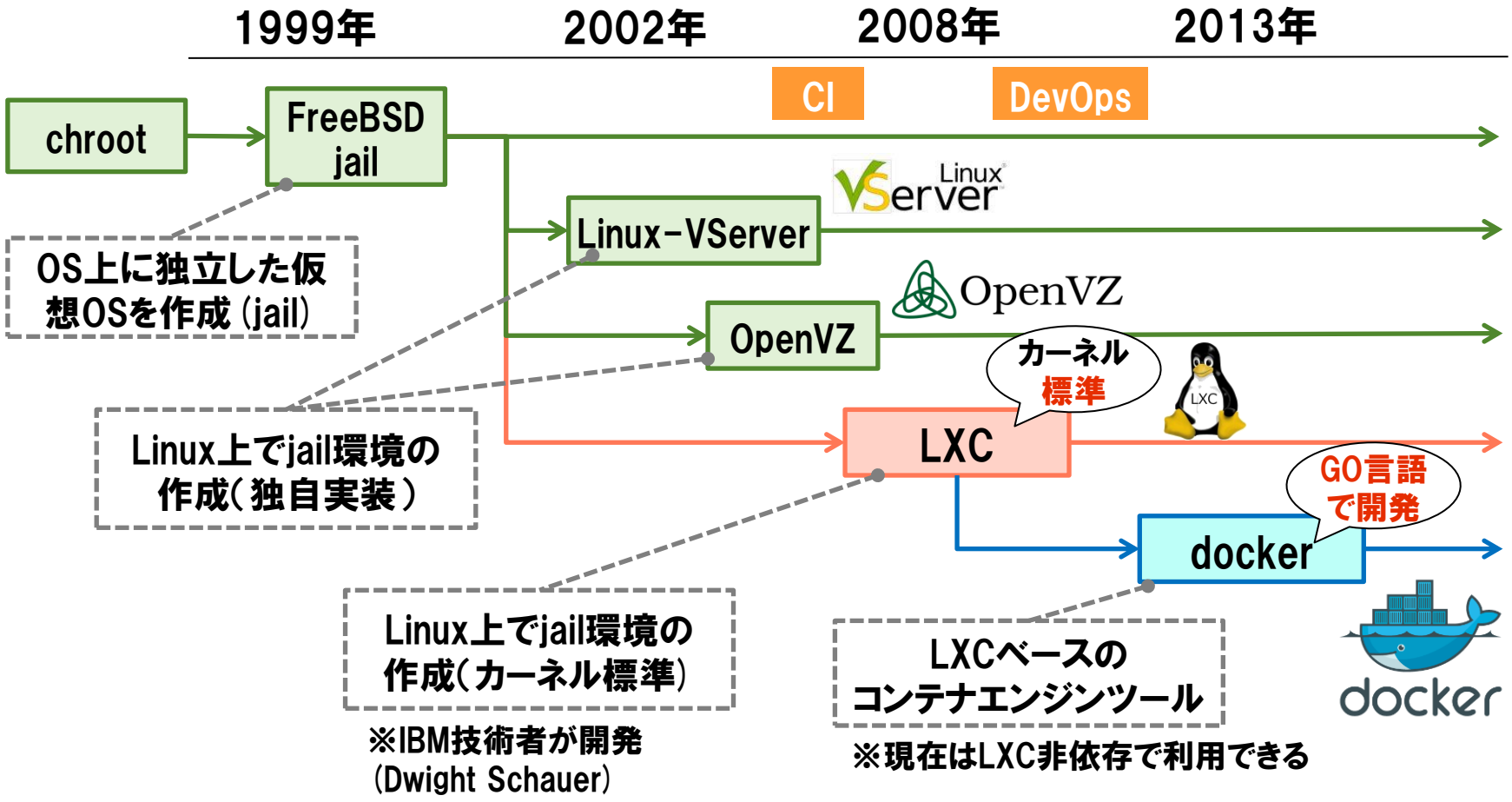
コンテナはクラウドアプリケーションに適している

- シェアードナッシング／スケールアウトする分散アプリケーション
- DevOps型ライフサイクルを持つアプリケーション
  - OSは起動済みのため、AP起動時間でコンテナを起動することができる
  - 迅速なデプロイが可能になる

項目	コンテナ型	ハイパーバイザ型
実行環境イメージ	小さい	大きい
実行環境のデプロイ	軽い（速い）	重い（遅い）
仮想化オーバーヘッド	小さい	大きい
実行環境の可搬性	良い	異種ハイパーバイザ間で課題有
対応OS	Linux（将来的にWidows）	Linux，Windows等

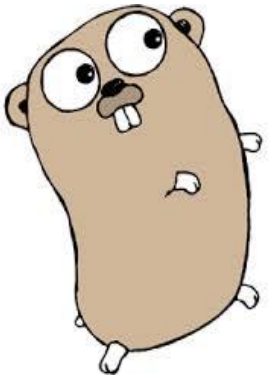
2．4．コンテナ型仮想化の歴史

CIやDevOpsの広がりとともに、ハイパーバイザ型仮想化よりも作成、起動が早く、動作も軽量なコンテナ型仮想化が浸透中。特にLinux標準のLXCを扱い易くしたDockerの登場後、急速に市場を拡大している。



参考：GO言語（１／３）

Googleが2009年に公開したコンパイル言語



GOのキャラクター
Gopher（ホリネズミ）

- **設計者**
  - Robert Griesemer（Google ChromeのV8エンジンの開発者）
  - Rob Pike（Plan9，UTF-8の開発者）
  - Ken Thompson（UNIX，C言語，UTF8の開発者）
- **最新リリース：1.3.2（2014/9/26）**
  - Linux/OS X/Windowsサポート
- **特徴**
  - シンプルな言語仕様
    - » ポインタ演算や暗黙の型変換は排除（メモリリークの回避）
    - » 例外処理の排除（複数の戻り値を返す、異常処理のためのパニック機能を整備）
    - » 宣言した変数やインポートパッケージが使用されていない場合、コンパイルエラー
  - **クロスコンパイル**のサポート
  - **Webサーバ**など充実した標準ランタイムパッケージ
    - ランタイムがリンクされるためサイズが大きくなるが**外部依存がない**
    - **サーバとアプリケーションの境界がなくなる ⇒ アプリのみでWebサービス実現可**
  - **並列処理**のサポート（goroutine・channel）

多様な書き方を認めず、
バグを生みやすい
表現は排除する設計論

参考：GO言語（２／３）

Webサービス開発言語の比較

GO言語はコンパイル言語であるため必要なライブラリはすべて静的リンクされ、**サーバ側に依存ライブラリを入れる必要がない**。
また、実行ファイルをコピーするためで動作するため、**非常にデプロイが容易**である。**コンテナ技術と組み合わせることでWebサービス開発を変革**する可能性を秘めている。

	GO	Scala	Perl	PHP	Java	Python	Ruby
性能	★★★	★	★	★★	★★	★	★
学習リソース	★★	★	★★★	★★★	★★★	★★	★★★
デプロイ容易性	★★★	★	★	★	★	★	★
事例	Google Docker	Twitter LinkedIn Foursquare	Mobage	Facebook Yahoo Wikipedia	Amazon eBay LinkedIn	Dropbox	
備考	マルチスレッド、並行処理に優れる	Javaへの依存性が高い	利用者は減少傾向	コンパイル後の性能はよい		実行速度が遅い	

参考：GO言語（３／３）

GO言語で開発された代表的なプロダクト一覧

- コンテナ型仮想化、オーケストレーション周りを中心に採用が進んでいる

OSSプロダクト名	概要
Docker	コンテナ型仮想化エンジン
Kubernetes	Googleが公開したコンテナクラスタマネージャ
InfluxDB	時系列分散DB
etcd	情報共有とサービスディスカバリに利用できる分散KVS
SkyDNS	etcdをバックエンドに持つDNS
Raft	ゴシッププロトコル、goraftによってgolang実装が提供されている
packer	単一ソースから複数のVM/コンテナイメージ生成を行うツール
serf	サービス検出・オーケストレーションツール
consul	サービス検出・設定ツール
Juju	サービスオーケストレーションツール（Pythonからgolangへ移行）

2．5．オーケストレーション（クラウドオーケストレータ）

クラウド環境のプロビジョニング自動化

- 一般にシステム基盤を構成するパーツ（サーバ・ネットワーク・ストレージ）の依存関係と設定項目をテンプレートとして定義し、一挙にクラウド上に構築する機能を提供する
  - 自社要件に合わせて、サービスコーディネータとIaCツールを組み合わせて機能実現する場合も多い

プロダクト	備考
OpenStack Heat	OpenStack専用のオーケストレーションツール
Terraform	高レイヤに特化、IaaS/PaaS/SaaSを1テンプレートに混在可
CloudConductor	TISが2013年の経産省の産業技術実用化開発事業補助金PJで開発
PrimeCloud Controller	SCSKがOSS公開、パブリッククラウド/プライベートクラウドに対応
Fig	Docker対応のコンテナオーケストレーションツール
Kubernates	GoogleがOSS公開したDocker対応のコンテナ管理ツールでオーケストレーション機能を有する

2. 6. PF構築自動化

サーバインストール/設定や仮想環境構築作業の自動化

- Infrastructure as Code(IaC)
  - ソフトウェアインストールと設定の自動化に加え、OSインストールや仮想環境準備など様々な自動化ツールが登場、市場も活性化

プロダクト	備考
Chef	RubyベースのCookbook/Recipeで構成記述、スタンドアロン・C/S型
Puppet	RubyベースのmanifestでC構成記述、スタンドアロン・C/S型
juju	Ubuntu上のサービス自動構成ツール
Ansible	YAML形式のPlay Bookで構成記述、SSHで直接命令(設定対象にAnsible不要)
Cobbler	Linux OS自動インストールツール、Kickstartと連携可能
Kickstart	RedHat系LinuxのOSインストーラ・anacondaのOS自動インストールの仕組み
Vagrant	複数仮想環境対応のテンプレート(vagrantファイル)による仮想環境構築ツール
Packer	仮想マシンイメージ作成ツール(VirtualBox、VMWare、EC2、Dockerなど)に対応

2. 7. IaaS基盤ソフトウェア

Hypervisor型仮想化技術をベースとしたクラウド環境構築用ソフトウェア

- 基本機能はVMとストレージ、ネットワークのリソースを制御し、ユーザに提供すること

プロダクト	備考
OpenStack	NASAとRackspaceが立ち上げ、大手企業の参加が多く現在最も有力
CloudStack	CitrixがOSS公開、Apacheプロジェクト
Eucalyptus	カリフォルニア大で学内クラウド用に開発開始、OSSとして公開
OpenNebula	スペインの研究PJとして開発開始、2008年にOSSとして公開
Wakame	あくしゅが開発開始、OSS公開した国産IaaS基盤ソフトウェア（wakame-vdc）

2. 8. 運用・監視技術

運用機能、監視対象によって様々なOSSが存在する

- クラウド環境の広がりと共に、増減するサーバを考慮したログ収集/メトリック可視化、サービスコーディネータが登場

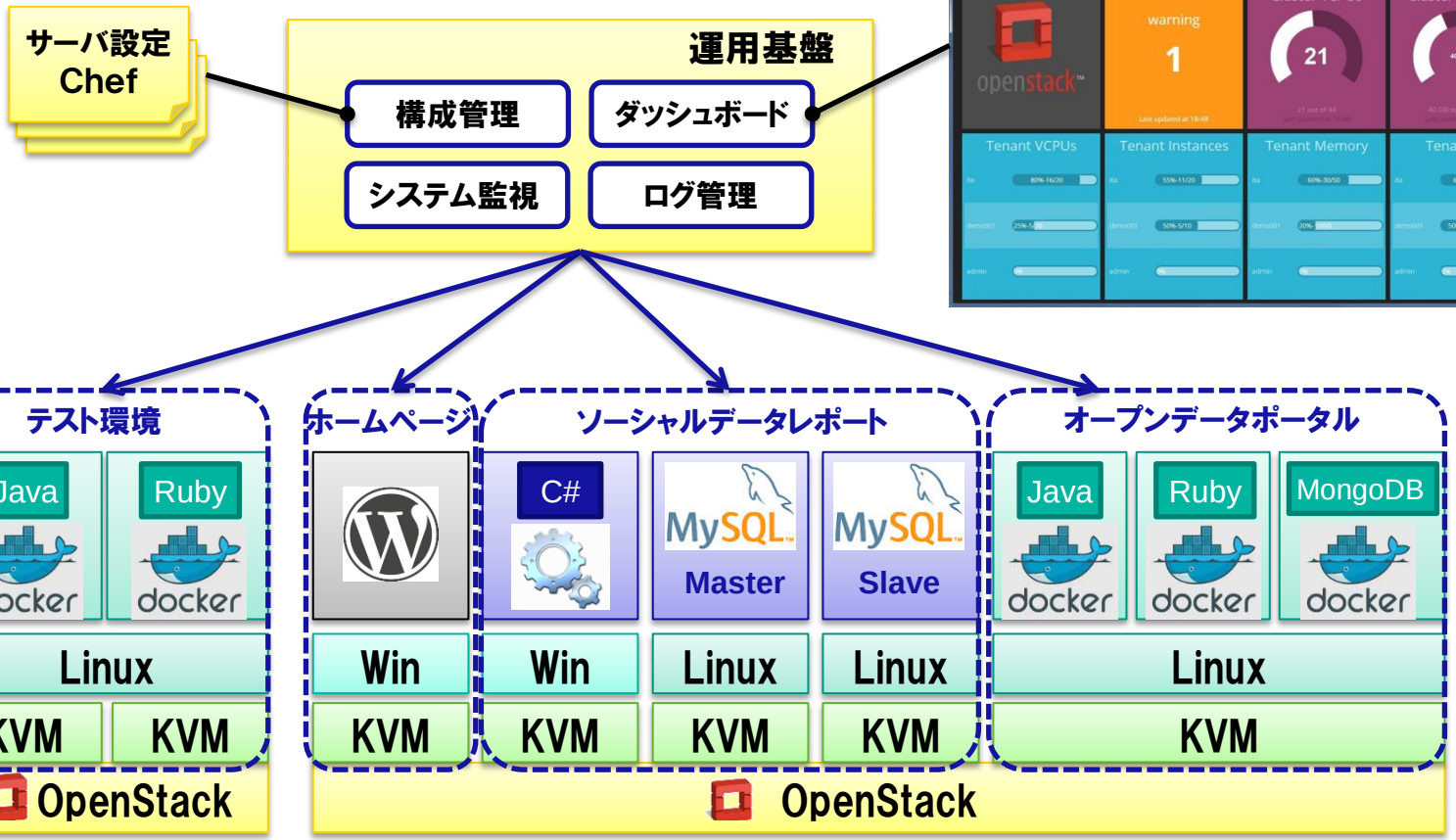
ポイントは伸縮性の実現

分類	説明
運用管理・監視	様々な機能を持つ統合製品。 【プロダクト】 Zabbix, Hinemos, Nagios, Munin, sensu(監視フレームワーク)
ログ収集	稼動ホストのログを収集して別の場所にあるデータストアへ送信する 【プロダクト】 Fluentd, FlumeNG, Scribe, Collectd
メトリック可視化	蓄積したログや稼動情報からグラフなどを生成して可視化する 【プロダクト】 Grafana, Kibana, Graphite
ジョブ管理	定型ジョブの実行制御、管理を行う 【プロダクト】 Job Scheduler ※Hinemos, Zabbixもジョブ管理機能を有する
バックアップ	データバックアップ機能を提供する 【プロダクト】 Amanda, Bacula, Clonezilla, Dick Archive, G4L
サービスコーディネータ	クラウドなど分散システムの運用に必要となるサーバ管理や情報共有、設定配布などを効率化する機能を提供する 【プロダクト】 serf, consul, ZooKeeper, etcd

2. 9. サービス基盤イメージ

全体像

- OpenStack(Grizzly)にDockerを組み合わせてSaaSサービス基盤を構築
  - OpenStackとDockerは現時点では別運用



まとめ

クラウド技術の進化とともにサーバ基盤も変遷

- 自社サービス基盤はOpenStackによるIaaS基盤＋Dockerで運用中

IaaS+Dockerの優位性

- IaaSによって払い出したVM上にコンテナを起動することで、APやミドルウェアのデプロイが迅速に行えるようになる

今後の課題

- リアクティブネスの実現
  - 現状ではサーバの動的な増減に対応して周辺環境を変化させるまでには至っていない
  - インフラ、運用監視、SaaSアプリケーションの全体でリアクティブアーキテクチャを検討する必要がある
- コンテナ運用管理の導入
  - 現在、コンテナはVMと同様の運用監視の仕組みを適用している
  - コンテナのリアクティブネスを実現するには、Kubernetesをはじめとした運用プロダクトが必要となるがまだベータの域を出ていない
  - ただし、コンテナは今後の必須技術であるためテスト環境を用いて検証する

3. DevOpsへの取り組み

3．1．DevOpsムーブメントのおさらい

2009/6/3、O'ReillyのイベントVelocityにおけるFlickerの2人の講演

10 deploys per day
Dev & ops cooperation at Flickr

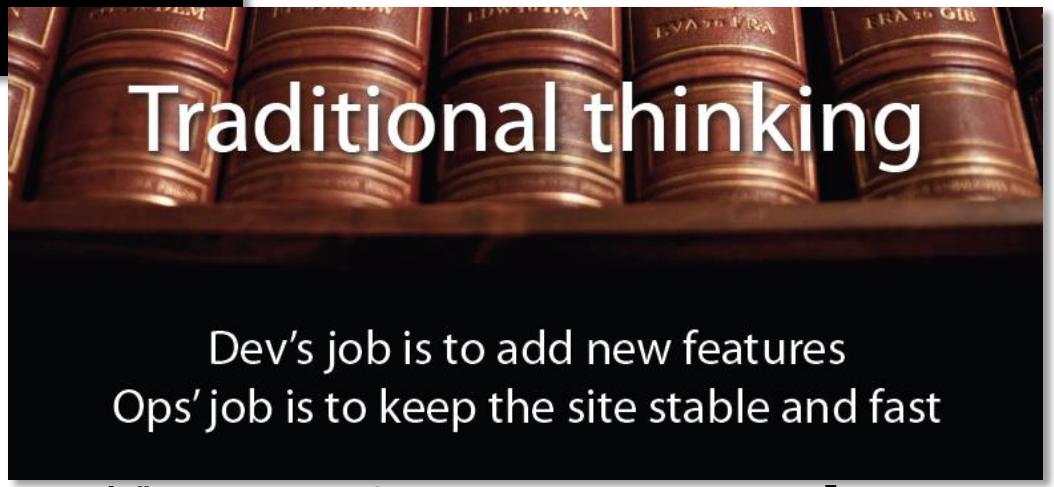
John Allspaw & Paul Hammond
Velocity 2009

Ops say

"It's not my machines, it's your code!"

Dev say

"It's not my code, it's your machines!"



理論や規約、標準などではなく、**共通のビジネスゴールを実現するには両者の協力が不可欠**であることを説いている

出典：John Allspaw & Paul Hammond Velocity 2009「10 deploys per day」

3．1．1．キーメッセージ

Ops' job is NOT to keep the site stable and fast.
Ops' job is to enable the business.
(this is dev's job too)

運用の仕事はサイトの安定と速度を維持することじゃない。
ビジネスを可能にすることだ（開発の仕事も同じ）。

The business requires change.
Lowering risk of change through tools and culture.

ビジネスは変化を要求してくる。
ツールとカルチャで変化のリスクを低減しよう。

出典：John Allspaw & Paul Hammond Velocity 2009「10 deploys per day」

3. 1. 2. ツール

1. Automated infrastructure
   (If there is only one thing you do…)
2. Shared version control
3. One step build and Deploy
4. Feature flags
   (aka branching in code)
5. Shared Metrics
6. IRC and IM Robots

1. 自動化されたインフラストラクチャ（Infrastructure as code）
2. 共有されたバージョンコントロール
3. ワンステップのビルドとデプロイ
4. 機能フラグ（一部機能をON/OFFにできる仕組み）
5. 共有された指標（評価基準）
6. IRCやIM（通知ツールの自動化）

出典：John Allspaw & Paul Hammond Velocity 2009「10 deploys per day」

3．1．3．DevOpsなツール例

・自動化されたインフラストラクチャ
⇒コードによるOSやミドルウェアの自動インストール
『Infrastructure as Code(IAC)』


・共有されたバージョンコントロール
⇒APコード、インフラコードの版管理

・ワンステップのビルドとデプロイ

参考：Chefコードサンプル

Apache2をインストールするコード例）

Recipe

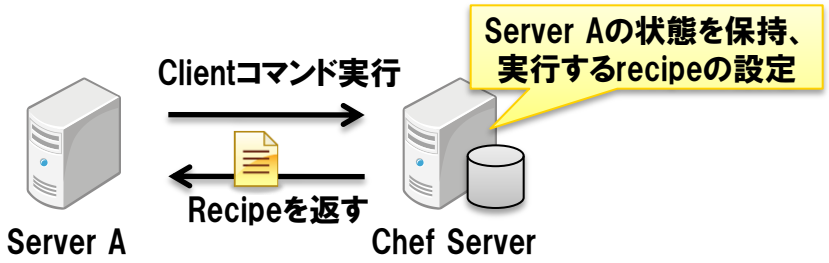
```
package 'apache2' do
    action :install
end

template
"# {node ['apache'] ['dir'] } conf/httpd.conf" do
  source    "httpd.conf.erb"
  owner     “www-data”
  mode      00644
  notifies :run, 'bash [restart apache] '
end

service " apache2 " do
    action  [ :enable , : start ]
end
```

Template（httpd.conf.erb）

```
ServerRoot "<%= node ['apache'] ['dir'] %>“
Listen <%= node ['apache'] ['port'] %>
```

Attributes

```
default ['apache'] [ ‘dir']   = "/usr/local/apache2/"
default ['apache'] ['port']  = 80
```

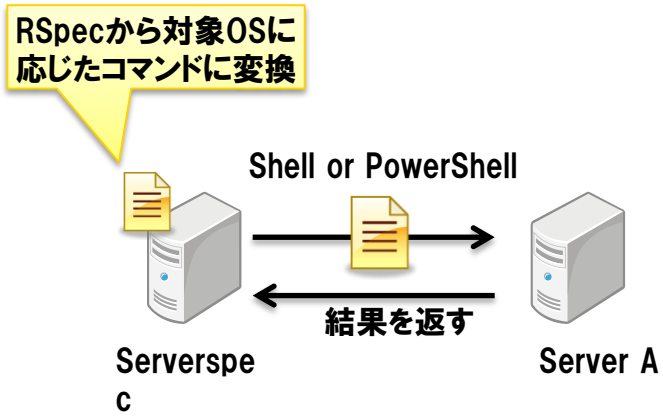


参考：Serverspecコードサンプル

Apache2をテストするコード例）

Apache_spec.rb

```
require 'spec_helper'

describe package ('apache2') do
  it { should be_installed }
end

describe service ('apache2') do
  it { should be_enabled   }
  it { should be_running   }
end

describe port (80) do
  it { should be_listening }
end

describe file ('/var/www/index.html') do
  it { should be_file }
end
```



3. 1. 4. カルチャー

1. 尊敬する
2. 信頼する
3. 障害(失敗)に対する健全な態度
4. 非難しない

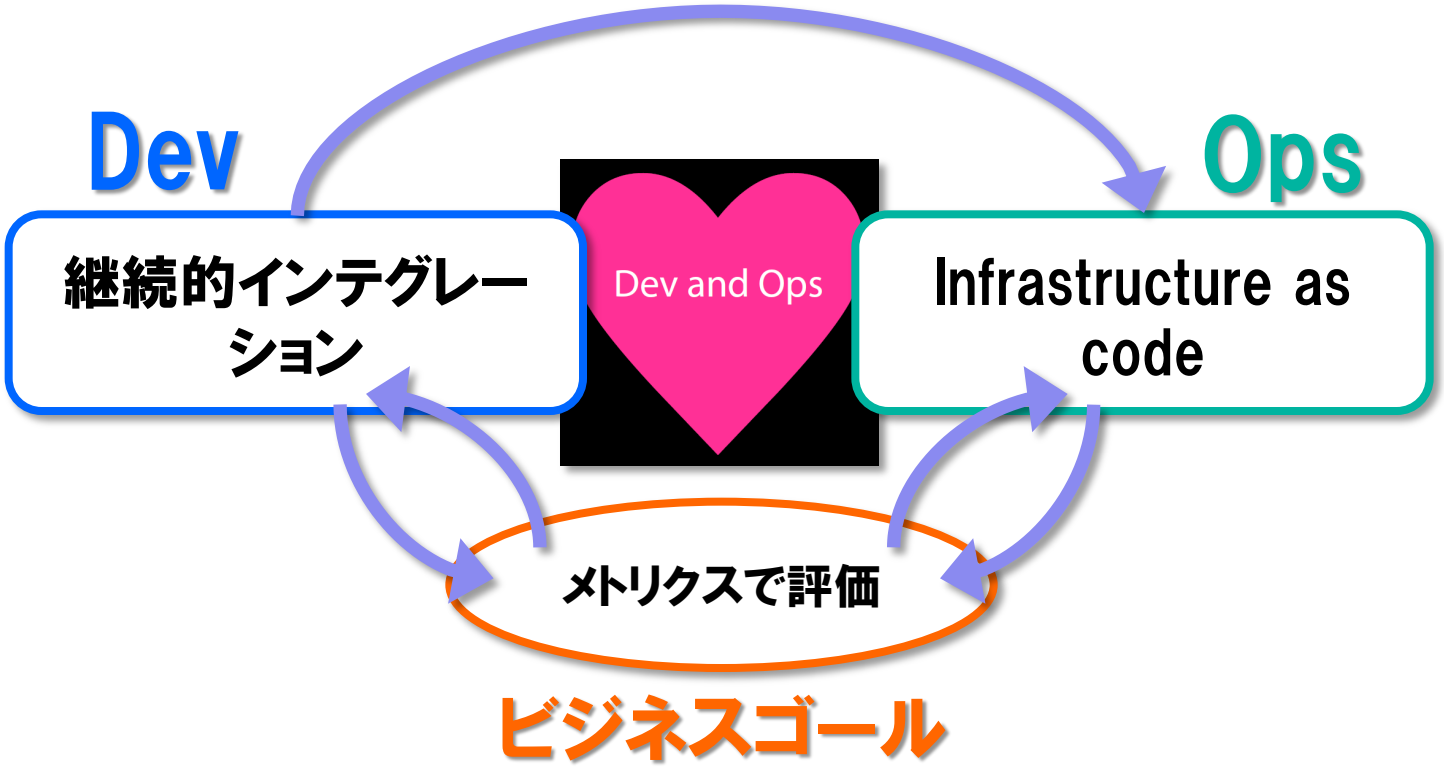
This is not easy
You could just carry on shouting at each other…

出典:John Allspaw & Paul Hammond Velocity 2009「10 deploys per day」

3．1．5. DevOpsまとめ

DevOpsって簡単にいうとどういう事？

- DevとOpsが共通のビジネスゴールに向かって協力しあい、共通の指標に従って評価・継続的改善を実施する開発/運用スタイル



3. 2. DevOpsの必要性

WebServiceの世界ではクラウド以降、変化するマーケットに合わせ、Agile and Iterative developmentが当たり前に

AP開発

アジャイル開発やCI、テスト自動化により開発期間短縮を実現

PF構築

APが求める基盤をIACにより迅速に提供、PF構築にCIを導入

The Application is the infrastructure, the infrastructure is the application

AP開発/PF構築の双方でアジャイルとCIを実現（DevOpsムーブメント）

SI/サービス開発でもAP開発/PF構築双方にアジリティが必要に

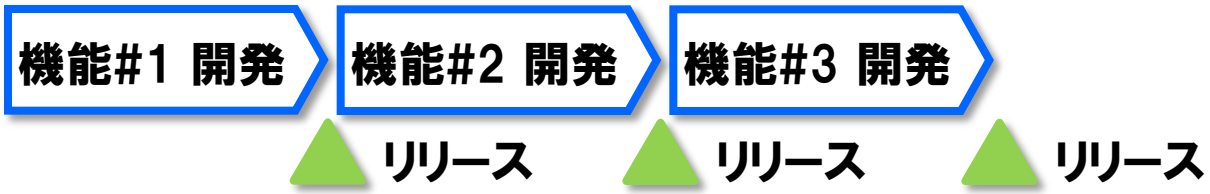
3．2．1．アジリティ

アジリティを高める ≒ 柔軟なリリースを実現すること

- 方法論の問題ではない（例：ウォーターフォール vs アジャイル）
  - 極力人手が介入しないモデルの実現（自動化）⇒人手は付加価値の高い作業に！
  - 継続的な改善（継続的インテグレーション）
- 品質を担保する仕組みも必要
- システムに関わる関係者のマインドシフトが必要

従来開発モデル

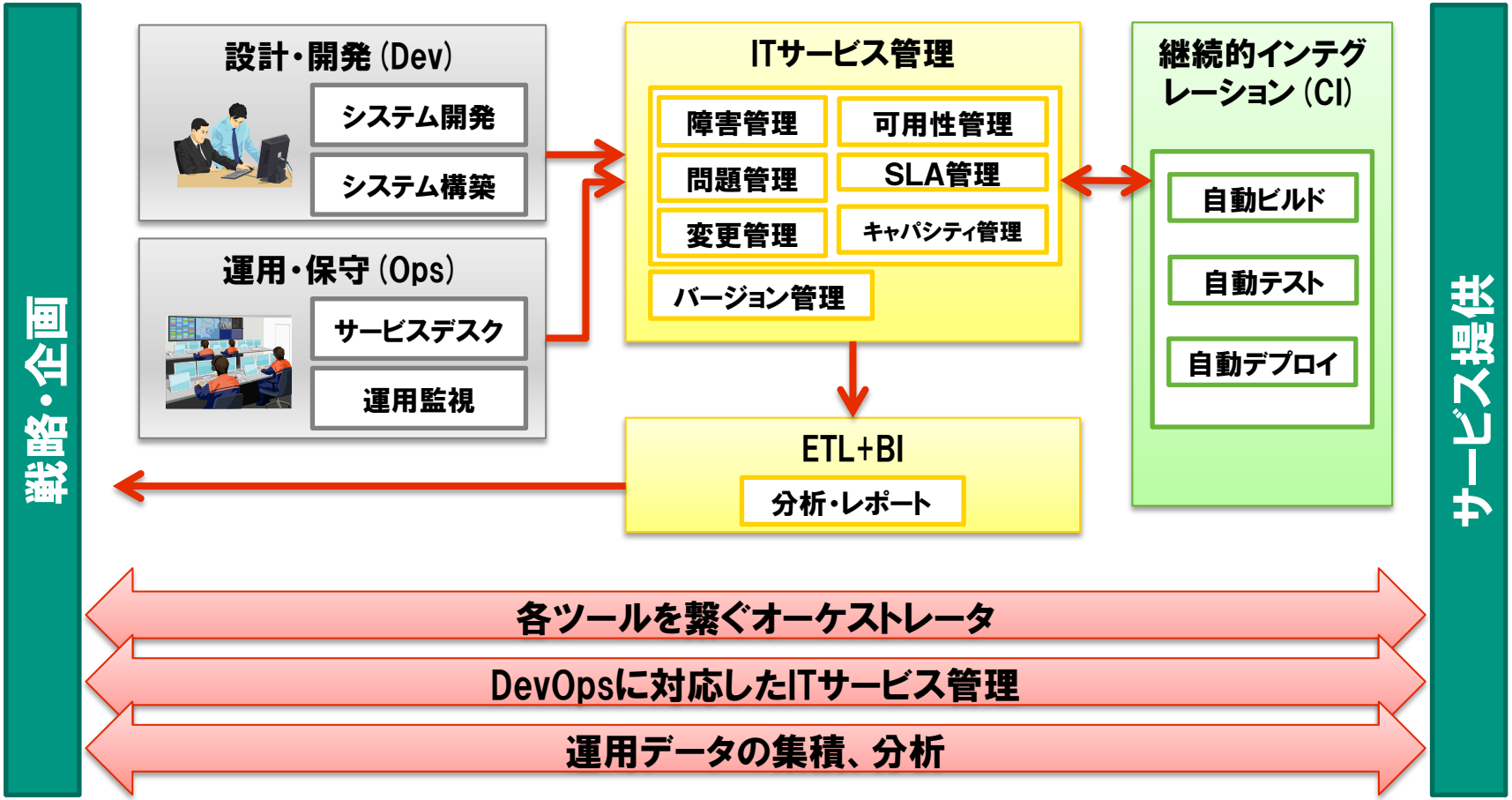
これからの開発モデル



3. 3. DevOpsのイメージ

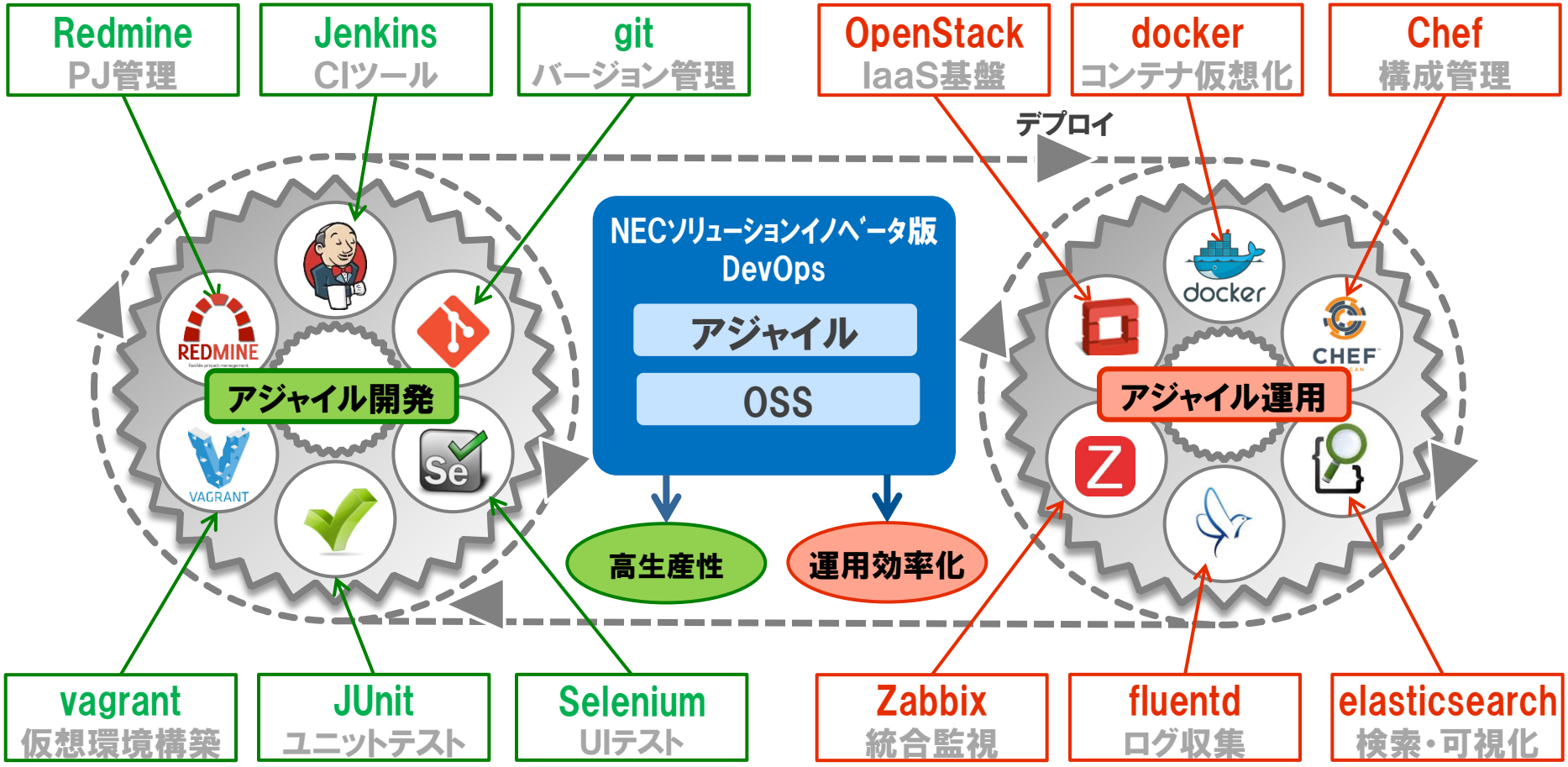
戦略・企画→開発→運用のスムーズなサイクル実現を目指す

- AP開発チームとPF構築・運用チームを一体化



3. 3. 1. 利用ツール

様々なOSSを活用してDevOpsにチャレンジ



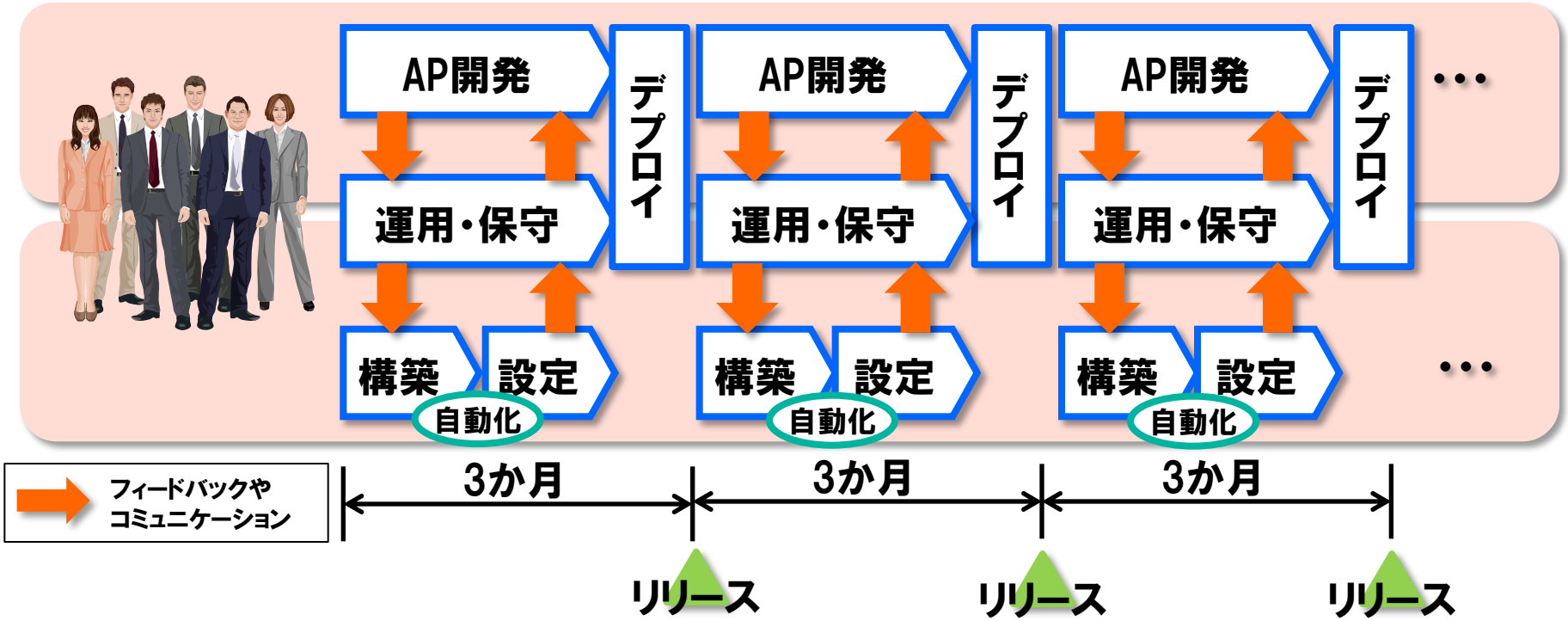
3. 3. 2. 開発フロー

既定したリリースサイクル毎のシステムリリース

- リリースタイミングはPJ毎に規定（下記例では3か月）

プログラミング作業以外は自動化

- PF構築も含めてコードによる自動化を推進する

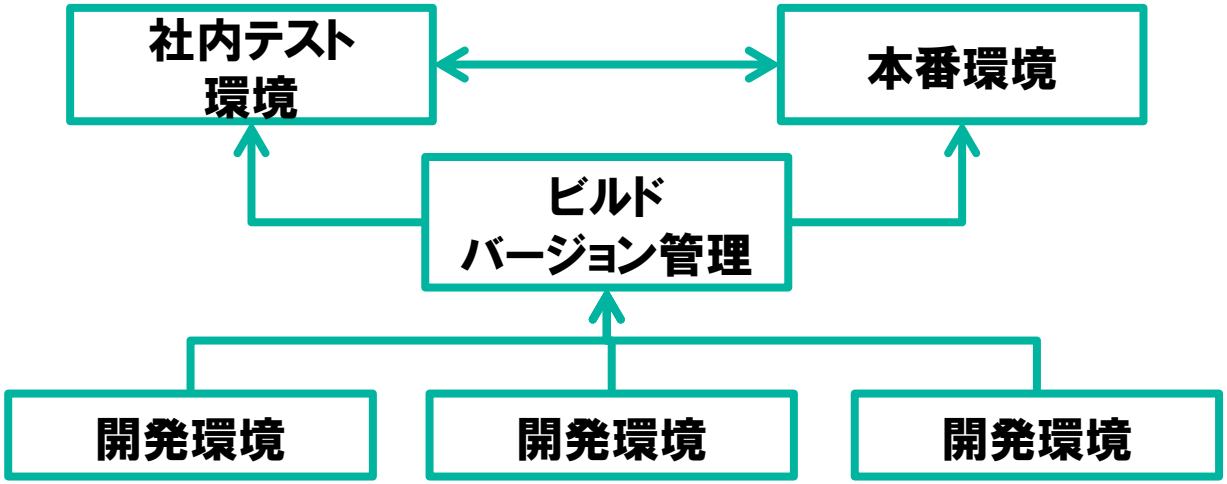


3. 3. 3. AP開発でのCI実現

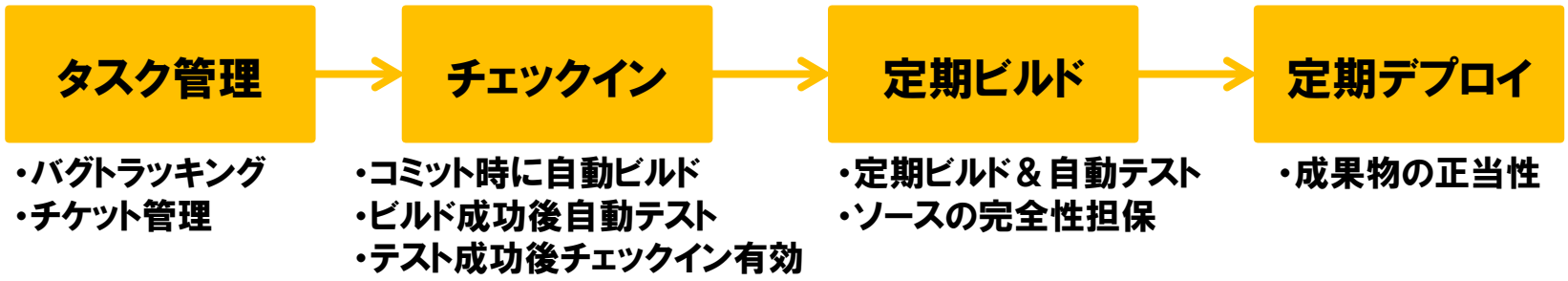
社内テスト環境-本番環境間でのデプロイ自動化

- JavaはJenkins+Junit ／ .NETはTFS（Visual Studio Team Foundation Server ）

全体イメージ

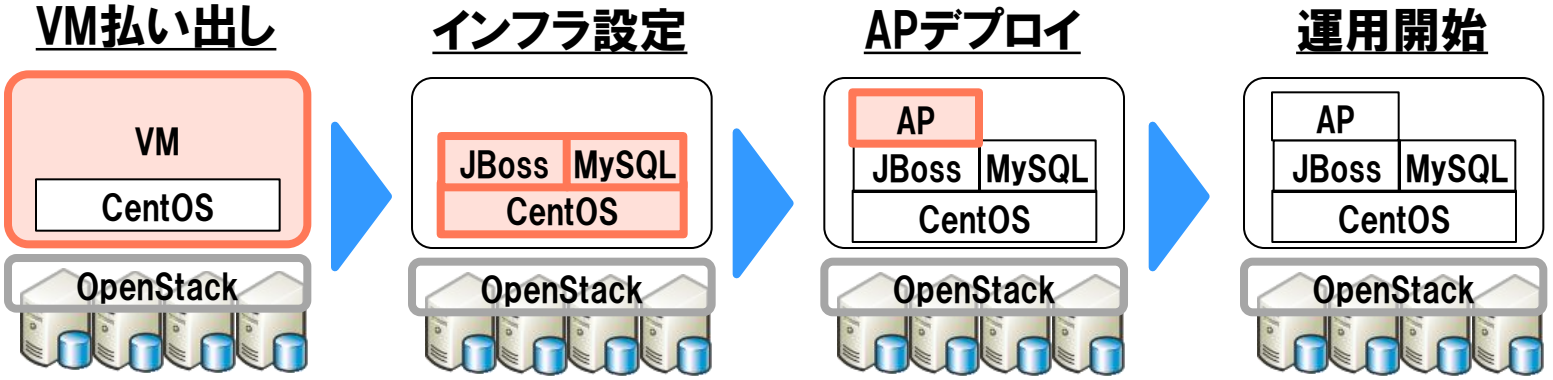


開発の流れ

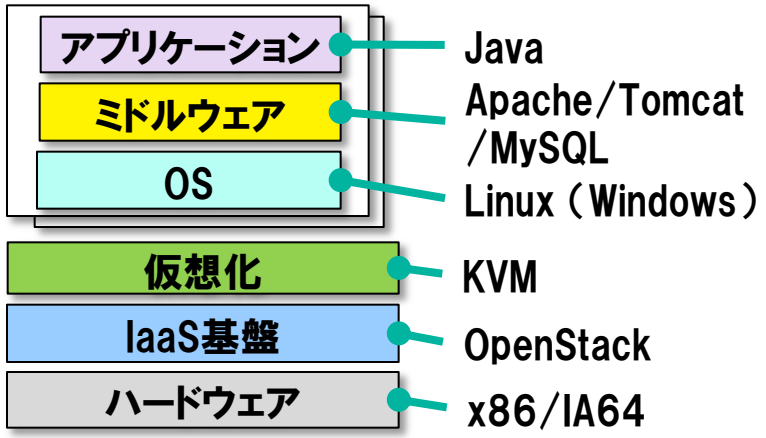


3. 3. 4. PF構築の自動化（1／2）

システムのアジリティを高めるために、人的プロセス中心の運用から
ツールを活用した自動化プロセスへの転換に取り組む



OSS基本ソフトウェアスタック



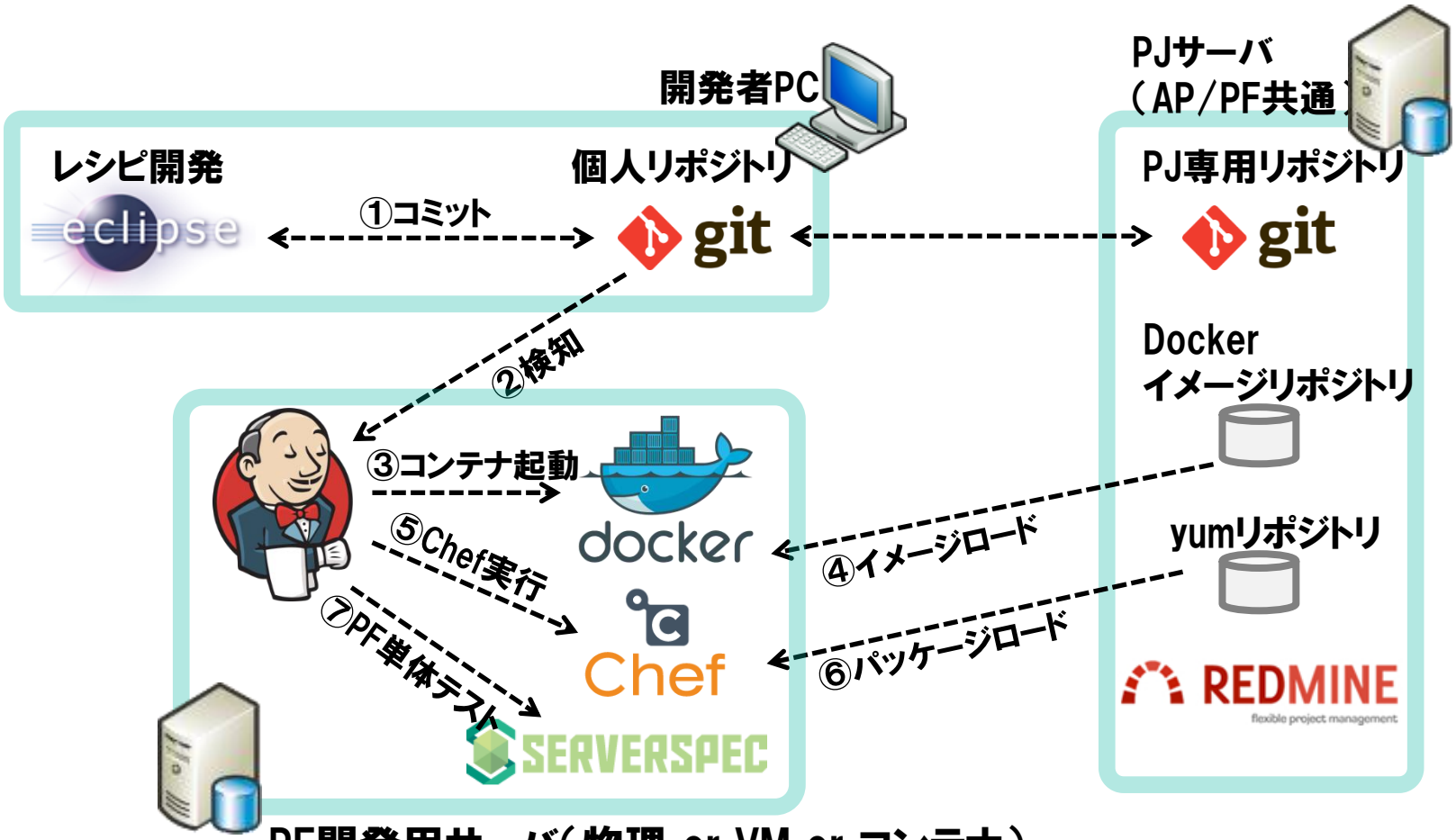
OSSでデプロイ/設定を自動化

- APビルド/デプロイ自動化
- インフラ設定自動化
- 仮想OS払い出し自動化

3. 3. 4. PF構築の自動化(2／2)

PF自動構築レシピ開発の流れ

- レシピ開発者のコミットと連動した構築テストの実施



まとめ

DevOpsの技術はサービス開発はもとよりSI開発にも適用可能

- 特にPF構築SIに効果がある
- SI開発においてはお客様との合意形成が重要
  - 何をメトリクスとしてシステムやサービスを評価するのか

DevOpsは既存の枠組みを変えなくても適用できる

- 方法論依存ではない
  - ただし、アジャイル的な開発PJによりフィットする
- IACはPF構築SIに品質管理の概念を導入することも可能となる

ご静聴ありがとうございました

Orchestrating a brighter world

世界の想いを、未来へつなげる。

未来に向かい、人が生きる、豊かに生きるために欠かせないもの。
それは「安全」「安心」「効率」「公平」という価値が実現された社会です。

NECは、ネットワーク技術とコンピューティング技術をあわせ持つ
類のないインテグレーターとしてリーダーシップを発揮し、
卓越した技術とさまざまな知見やアイデアを融合することで、
世界の国々や地域の人々と協奏しながら、
明るく希望に満ちた暮らしと社会を実現し、未来につなげていきます。

Empowered by Innovation