OpenStack 環境におけるブルー・グリーン・デプロイメントの実現

Implementation of the Blue Green Deployment in OpenStack Environment

佐々木　智一，吉　本　昌　平

要　約　ビジネスにスピードが求められる昨今，DevOps といったシステム開発におけるスピードを実現する考え方が注目されている．しかし，システム運用の現場では，手順書と手作業によって日々の業務が行われ，スピードとは逆行しているのが普通である．スピードを実現するために運用は自動化されるべきである．

本稿ではアプリケーションのリリースにおけるブルー・グリーン・デプロイメントという自動化手法に着目し，OpenStack のオーケストレーション機能である Heat や LBaaS などを利用して独自に開発した．シミュレーションの結果，開発手法を適用したシステムでの更新作業時間は，従来システムと較べて 8 分の 1 に短縮できることが分かった．

Abstract　As speed is required in the business, system development　such as the DevOps has attracted attention for its concept of realizing the speed of system development. However, in the system development operation, day-to-day operations are carried out by procedures and manuals. To speed up the daily operation, operation should be automated.

In this paper, we focused on the automation technique called the blue-green deployment in the application release. By using the Heat and LBaaS which are the orchestration capabilities of OpenStack, we have independently developed a blue-green deployment. As a result of simulation, the updation job time of the system which applied the blue-green deploymentwas reduced to 1/8 of the original system's job time.

1. は　じ　め　に

システム開発においてアジャイルやプロトタイプ開発手法が採用されることが多く，これは開発工程を反復することで早く要件に対応する，開発過程のリスクを早く発見するなどのスピードが開発側に求められているためである．このような開発で求められているスピードは，安全性と確実性という二つのシステム安定性の基準とともに追求される．しかし，このスピードと安全性や確実性は開発過程において相反する関係にある．それは，現在のシステム開発における多くの作業が，手順書を拠り所とした手作業で行われているためである．例えば，安全性や確実性を担保するために，手作業で二重チェックを行うなどスピードと相反する対応が必要となっている．

安全性や確実性を担保しつつスピードを実現するには，現在のシステム開発に対してパラダイムシフトを起こす必要があり，DevOps はこれを実現できる考え方である．DevOps では，より良いシステムを迅速に創りだすために，システム開発の様々な場面でその作業の自動化を推奨している．これまで開発者や運用者が行っていた手作業を自動化することで，スピード，安全性や確実性を実現できる．

本研究では DevOps を実現する種々の手法のうち，アプリケーションのリリース工程にお

ける自動化手法である，ブルー・グリーン・デプロイメントを実装することで，リリース工程を迅速に行えるか確かめることを目標と定めた．ブルー・グリーン・デプロイメントの機能を，OpenStack のオーケストレーション機能である Heat や LBaaS などを利用して独自に開発した．OpenStack は大規模なキャリアやサービス事業者にとっては，デファクトスタンダードと言える仮想化基盤であり，非常に注目を集めている．OpenStack は広く誰でも利用できる技術であるが，そのオーケストレーション機能を用いた DevOps を支援する機能の実装例は発表されていない．

2 章では，DevOps と継続的デリバリーについて説明し，さらにブルー・グリーン・デプロイメントとの関係を明らかにする．3 章ではブルー・グリーン・デプロイメントのような自動化手法が要求するソフトウェア定義について解説する．4 章では基盤として採用した OpenStack について，ブルー・グリーン・デプロイメントと関わりのある範囲での解説を行う．5 章ではブルー・グリーン・デプロイメントを OpenStack 環境で実現する手法を解説し，6 章では，本手法がシステム開発に求められるスピードを得る上で有用であるか考察する．

2. DevOps における自動化とブルー・グリーン・デプロイメント

本章では，DevOps と継続的デリバリーについて解説する．特に DevOps の構成要素として継続的デリバリーを位置付け，さらにブルー・グリーン・デプロイメントが継続的デリバリーにおける構成要素の一つであることを示す．

2. 1 DevOps と継続的デリバリー

DevOps という言葉は，「Velocity 2009」[*1] にて，Flickr 社のエンジニア John Allspaw によるプレゼンテーションがきっかけとなって広まった．プレゼンテーションでは「開発と運用が協力することで，1 日に 10 回以上のペースでリリースが可能になる」ということを，「Dev and Ops」[1] という言葉で表現した．今日における DevOps は，Dev（ソフトウェア開発者）と Ops（運用者）が協力してビジネスに対する様々な要求に対して，システムの変更を柔軟かつスピードを持って行うための考え方となっている．

DevOps には二つの側面があり，一つは文化，もう一つはツールである．そして，ツールには自動化，測定，共有の要素があり，中でも自動化が重要である．なぜなら開発者や運用者が人の手で行っていた作業を自動化することにより，安全性や確実性を担保したうえでスピードを実現できるためである．加えて自動化の重要な役割は，スピードを持った活動におけるセーフティーネットとしての役割である．例えば，リリースに失敗した際に元のバージョンに確実にロールバックできる自動化の仕組みはセーフティーネットの一つと考えられる．

継続的デリバリー[2] は価値の高いソフトウェアを短い周期で開発し，いつでもリリース可能とするためのワークフローや自動化手法を指す言葉であり，そのワークフローには様々な自動化手法が存在する．例えば，継続的インテグレーション[*2]（CI：Continuous Integration）は，不具合修正コストの最小化を狙っている自動化である．

このように，DevOps と継続的デリバリーには，自動化という共通点があり，DevOps を構成する要素の一つとして継続的デリバリーを位置づけることができる．DevOps には，1 章で述べたとおりシステム開発において様々な自動化を達成するという考え方があるが，実際の手法について明確な定義はない．

2.2　ブルー・グリーン・デプロイメントの位置づけ

ブルー・グリーン・デプロイメント[*3]は，継続的デリバリーのワークフローにおけるリリースにおいて採用できる自動化手法の一つである．前述の DevOps や継続的デリバリーをまとめたのが図 1 である．開発者のワークフロー全体は継続的デリバリーを構成し，運用者のワークフローと合わせて全体を DevOps と置くことができる．このように，ブルー・グリーン・デプロイメントは DevOps を構成する要素の一つと位置付けられる．

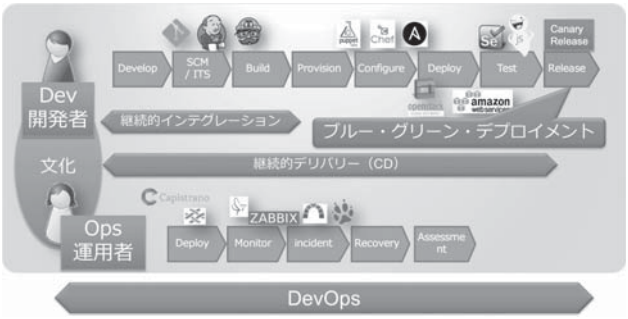

図 1　ブルー・グリーン・デプロイメントの位置づけ

2.3　ブルー・グリーン・デプロイメントとは

ブルー・グリーン・デプロイメントは，ウェブアプリケーションのリリースに対する課題を解決する．従来のウェブアプリケーションでは本番系を停止できないため，検証系や開発系といった別の環境を準備して，新しい機能の開発などに対応することがほとんどである（図 2）．検証系や開発系は本番とは別の環境であるため，リリース前に移行作業が必要であり，この周辺で種々の課題が発生する．例えば，検証環境におけるテスト中に発見された不具合に対し，プログラムを場当たり的に修正したために，移行用パッチに含めるのを忘れてしまうケースや，切り戻し手順に不具合があると，停止時間が当初想定より長引くなどの二次災害を引き起こしてしまうリスクである．これらの課題は，ブルー・グリーン・デプロイメントによって解決できる．

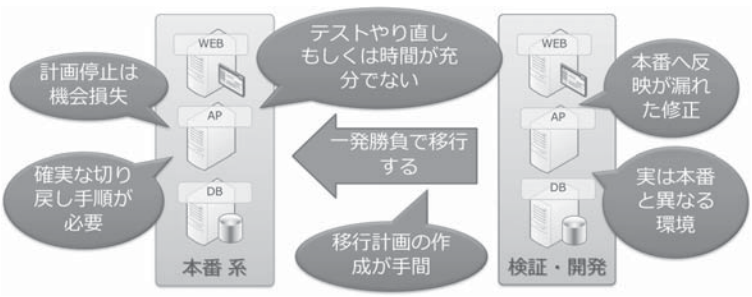

図 2　従来手法によるウェブアプリケーションのデプロイ

ブルー・グリーン・デプロイメントの特徴として，二つの本番系を交互に切り替えるといったものがある．二つの本番系の一方にはブルー，もう一方にはグリーンと名前をつけて区別する

（図 3）．ある時点ではブルー系が本番サービスを提供し，グリーン系において新機能の実装やテストを行い，十分な準備をする．新しい機能を持ったグリーン系のテスト結果が合格となった時点で，ブルー系に接続していた本番系をグリーン系に切り替える．このような切り替えを行うことで，システムを更新していく．もし，グリーン系への切り替え後にさらに問題が見つかった場合は，少し前まで動作していたブルー系に切り戻すこともできる．

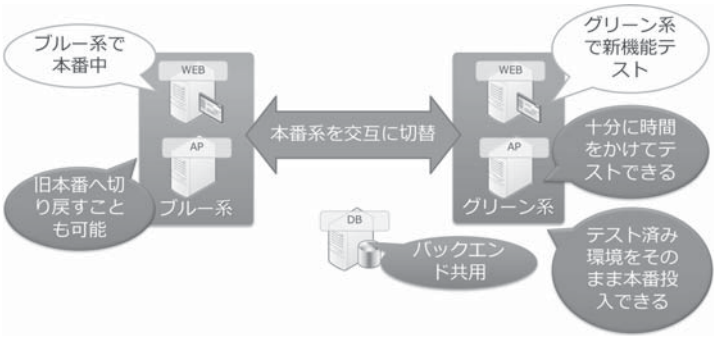

図 3　ブルー・グリーン・デプロイメントを適用したウェブアプリケーションのデプロイ

ブルー・グリーン・デプロイメントの利用事例として，amazon.com が挙げられる．2012 年の re:Invent[3]にて，「クラウドネイティブなデプロイ」という，二つの本番系を切り替えて更新するブルー・グリーン・デプロイメントと同様な手法について紹介があった．これにより，1 時間に最大 1000 回以上の更新を実現しているとも発表されていた．

ブルー・グリーン・デプロイメントでは，本番環境を二つ準備するため，ハードウェアやその設定作業に対するコストの課題があるが，仮想化によりコストの軽減が可能である．ブルー・グリーン・デプロイメントは仮想環境上に構築することを要件とするべきである．

3.　自動化を実現するソフトウェア定義（Software Defined）

自動化を行う上で重要なソフトウェア定義は，ソフトウェアによるネットワーク制御技術（SDN）から導かれた考え方である．ソフトウェア定義は自動化を容易にするために，システム基盤が他のソフトウェアから制御しやすい仕組みを提供することを指す言葉である．本章では，ソフトウェア定義について解説し，ブルー・グリーン・デプロイメントで必要とされるソフトウェア定義を示す．

3. 1　ソフトウェア定義（Software Defined）とは

ソフトウェア定義の歴史を紐解くと，ソフトウェアによるネットワーク制御技術（SDN）が認知された後に，様々な Software Defined が誕生するに至っており，例えば表 1 のようなものがある．OpenFlow[*4] から始まった SDN の本質は，制御部とデータ転送部の分離であったが，その後に派生した Software Defined は，REST（Representational State Transfer）などの外部からの操作が可能な API（Application Programming Interface）を備え，それによる各種の自動化が可能という文脈で語られることが大半である．実際にどのような定義を行うのかは別の検討が必要である．

表 1　様々な Software Defined

名称	略称	対象
Software Defined Networking	SDN	ネットワーク
Software Defined Data Center	SDDC	データセンタ
Software Defined Infrastructure	SDI	インフラストラクチャ
Software Defined Storage	SDS	ストレージ
Software Defined Everything	SDx	すべて

3. 2　ブルー・グリーン・デプロイメントにおけるソフトウェア定義

ソフトウェア定義をどのように行うのかは，DevOps や継続的デリバリーにおいて要求される，システム開発のワークフロー各場面における自動化を検討することが出発点となる．ブルー・グリーン・デプロイメントでは，三つの自動化が必要となる．

1)　仮想サーバーの起動とセットアップ，アプリケーションの配備
2)　仮想サーバーのロードバランサー上への登録
3)　ブルー系とグリーン系の切り替え

これらの自動化を実現するために，仮想サーバー基盤やネットワーク基盤が提供する API を用いて，外部のアプリケーションから仮想サーバーや仮想ネットワークの状態を定義する．

4.　OpenStack とは

OpenStack[4]はソフトウェア定義の機能を持つ，主に Python 言語で書かれたオープンソースソフトウェアの仮想化基盤である．本章では OpenStack の概要と，ブルー・グリーン・デプロイメントに関わりの深い Neutron と LBaaS（Load Balancer as a Service），Heat といった機能について解説をする．

4. 1　OpenStack が実現すること

OpenStack は仮想サーバーを素早く大量に作るための仕組みである．従来のサーバー構築では，まずハードウェアを調達し，入荷後に OS などをセットアップする必要があり，利用可能になるまでに数週間程度かかることが普通であった．しかし OpenStack のような仮想環境によって，新しい仮想サーバーを数分から数十分のうちに利用できる．

仮想環境においては，CPU やメモリなど仮想化されたハードウェアのうち，仮想的なネットワークインターフェースカード（NIC）と物理 NIC の間を，仮想ネットワークとして接続する必要がある（図 4）．仮想サーバーを素早く大量に作るためには，仮想ネットワークも同様に素早く大量に作る必要がある．OpenStack でこれを実現しているのが Neutron である．

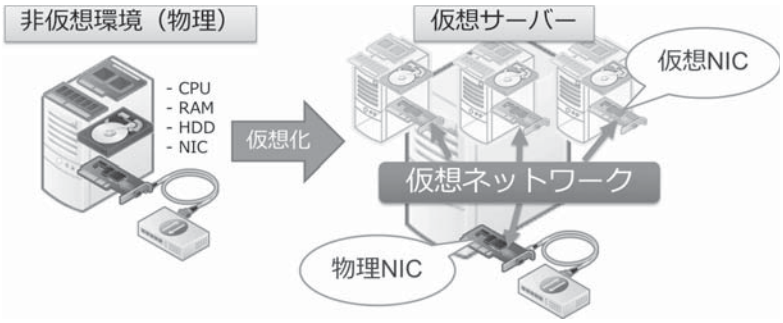

図 4　仮想ネットワーク

4. 2　OpenStack の SDN である Neutron とは

Neutron は，API によりソフトウェア定義が可能なネットワーク環境を実現し，OpenStack 上の仮想サーバーのネットワークを制御している．Neutron の重要な機能は，複数のユーザーが利用する場合に「テナント」と呼ばれる単位で仮想ネットワークを分離することである．一般的にはユーザーごとに利用可能なセグメントや IP アドレスを割り当てるような事前の調整が必要になるが，Neutron においては不要である．例えば図 5 のように，各テナントには 192.168.1.0/24 という同じセグメントを持つことができ，仮想サーバーはこのセグメントを経由してネットワークに接続する．このセグメントをここでは内部セグメントと呼ぶ．

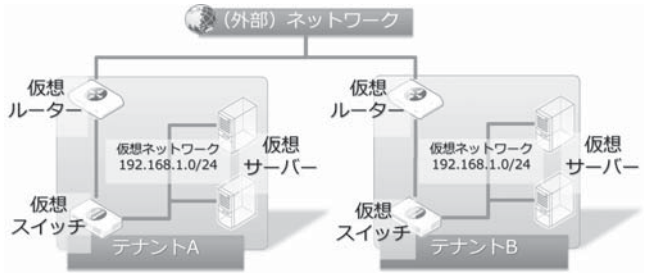

図 5　Neutron による仮想ネットワーク

仮想サーバーに外部から接続する際には，フローティング IP が必要となる．Neutron では同じ内部セグメントを複数持つことができるため，複数の仮想サーバーが同じ IP アドレスを持つ可能性がある．これでは，外部の物理ネットワークから仮想サーバーを一意に決定することができない．このため，フローティング IP を外部の物理ネットワークで確保し，仮想サーバーに関連付けることで外部からの通信を実現する．フローティング IP は仮想ルーター上のアドレス変換（NAT：Network Address Translation）によって実現される．

4. 3　OpenStack の SDx

OpenStack はソフトウェア定義の機能を Neutron 以外にも備えている．ここでは LBaaS と Heat について解説する．

LBaaS（Load Balancer as a Service）は OpenStack の仮想サーバーから利用可能なロードバランサーを構成する機能である．LBaaS は REST API 経由でロードバランサーの構成に必

要な，プール，メンバー，バーチャル IP，ヘルスモニタといった設定を行うことが可能である．LBaaS では，ドライバを準備することで，複数のロードバランサーに対応が可能であり，デフォルトの HAProxy[5] に加えて，F5 Networks[6] や A10 Networks[7] などの商用ロードバランサーも利用可能となっている．

次に，OpenStack Heat の二つの機能について解説する．一つが自動構成のためのテンプレート化であり，もう一つはオートスケーリングである．

Heat を利用すると OpenStack 上の様々な設定項目をテンプレート化して自動的に構成することができる．例えば，仮想サーバーの作成と設定，アプリケーションのデプロイ，ネットワーク設定，ストレージの設定，さらに LBaaS の設定も扱うことができる．テンプレートに対し，環境に応じたパラメーターを設定することで，同じ構成を何度でも，他のテナント内でも再現させることができる．今回構築したブルー・グリーン・デプロイメントの環境もテンプレート化されている．

オートスケーリングはスケールアウト構成を自動化する機能であり，スケールアウト構成は複数のサーバーを準備して処理を分散させ，全体として処理能力を稼ぐことを目的とする．オートスケーリングでは，Web サーバーの CPU 負荷などを監視し，例えば利用者数が増加してサーバーの負荷が高まり，ある閾値を超えた場合に新たな仮想サーバーを自動的に追加して処理を分散させ，処理能力を追加して負荷に対応することができる．逆にリソースが余った場合には仮想サーバーを自動的に減らす．これで遊休リソースの他用途への活用や，消費電力の低減を達成することができる．

5. OpenStack 環境におけるブルー・グリーン・デプロイメントの実装

本章ではブルー・グリーン・デプロイメントの機能を OpenStack 環境に構築する方法および，独自開発したブルー・グリーン・デプロイメント管理機能について解説する．

5. 1 OpenStack を選定した理由

ブルー・グリーン・デプロイメントを OpenStack 環境に構築する理由は三つある．一つは 2. 3 節で示したとおり，ブルー・グリーン・デプロイメントが仮想環境を前提とするべき手法であるため，二つ目は，OpenStack の持つ SDN や SDx 技術を活かした切り替えが可能であるため，三つ目は，だれでも入手可能なオープンな技術であるためである．

OpenStack 環境でのブルー・グリーン・デプロイメントは，ブルーとグリーンの系の切り替えを，フローティング IP を制御する Neutron API を利用したネットワークレイヤの機能として実現している．また，Heat を使うことで自動構成やオートスケーリングの機能を実現できる．

5. 2 OpenStack 環境でのブルー・グリーン・デプロイメント実装

本環境は Ubuntu 版[*5] OpenStack の Juno バージョンを用いて構築した．

ブルー・グリーン・デプロイメントを実装するには，まずブルー系とグリーン系と呼ぶ二つの本番系を準備する必要がある．OpenStack の一つのテナント内に，ブルー系とグリーン系に対応した二つの仮想ネットワークを準備し，それぞれに Web サーバーとなる仮想サーバーを稼働させる（図 6）．Web サーバーは可用性やスケールアウトのために複数の仮想サーバー

を稼働させるため，複数の Web サーバーを束ねるロードバランサーを LBaaS によって構成する．さらに外部ネットワークから接続するために，ロードバランサーのバーチャル IP（VIP）アドレスに対して，フローティング IP を対応付ける．

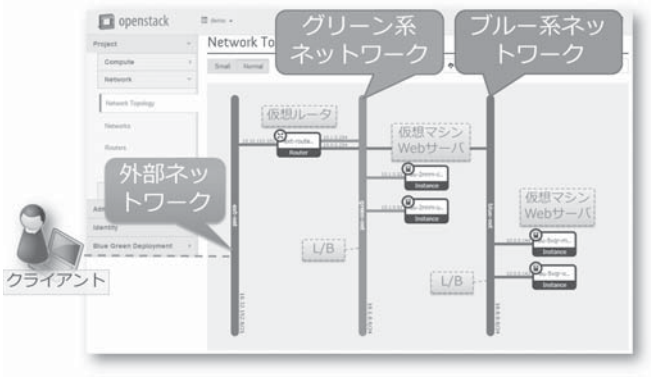

図 6　OpenStack 環境に構築したブルー・グリーン・デプロイメント

ブルーとグリーンの系の切り替えは，ロードバランサーのバーチャル IP に対応付けたフローティング IP を付け替えることで行う．つまり，ある時点ではフローティング IP はブルー系のロードバランサーの VIP に対応付けられており，切り替えはこれをグリーン系の VIP に変更することで実現する．切り替えの操作は Neutron API のうち，フローティング IP に関連するものを利用する．

系の切り替えの操作は，独自開発したブルー・グリーン・デプロイメント管理画面から行う．図 7 のように OpenStack の管理画面（Horizon）に組み込んだ．

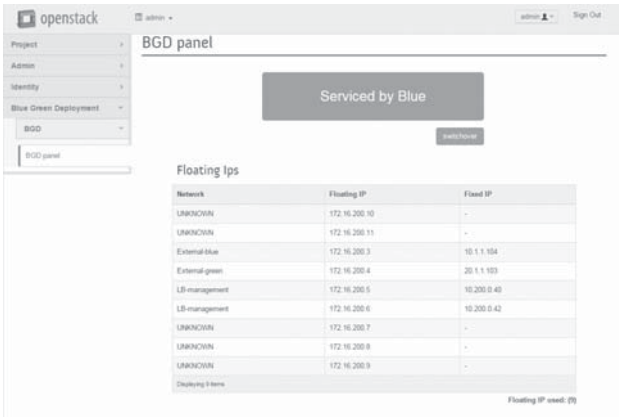

図 7　ブルー・グリーン・デプロイメント管理画面

5. 3　OpenStack 環境のブルー・グリーン・デプロイメントに対する付加機能

ブルー・グリーン・デプロイメントの付加機能として，自動構築とオートスケーリングを Heat を用いて実装した．

自動構築では，一つの操作で，仮想サーバーの起動から Web サーバーのインストールと設

定，デモ用の CGI ファイルの配備，ロードバランサーの構成とフローティング IP の設定までを自動的に行う．初期のネットワークを構成した状態から，一つのアクションでブルー系とグリーン系を構成できるように Jenkins[*6] へ組み込んだ．

また，オートスケーリングは，オートスケーリンググループとしてブルー系とグリーン系をそれぞれ構成することで，ブルー・グリーン・デプロイメントに組み込んだ．

5. 4　HTTP の Keep-Alive によって切り替わらない問題と対処

デフォルトの Neutron 実装において，ブルー・グリーン・デプロイメントを利用する場合には，HTTP の Keep-Alive[*7] の影響により切り替え不能となる問題がある．ここではその理由と対処について解説する．

デフォルトの Neutron 実装では，仮想ルーターは iptables[*8] を制御することによって実現され，フローティング IP はこの iptables の NAT テーブルによる IP アドレス変換によって実現されている．ブルー・グリーン・デプロイメントの切り替え操作を行うと，この NAT テーブルの定義が書き換わることになる．

HTTP の Keep-Alive が有効である場合，一度の TCP 接続で複数の HTTP リクエストを処理するため，Keep-Alive のタイムアウトになるまで同じ TCP 接続を利用し続ける．このため，ブルー・グリーン・デプロイメントの切り替えに伴う NAT テーブルの変更が反映されない．タイムアウトは無通信状態を計測するため，リクエストが頻繁に発生すると，切り替えが発生しないという結果となる．

さらに，LBaaS のデフォルト実装のロードバランサーである HAProxy を利用した場合，HAProxy が Keep-Alive 有効で起動するため，無効化する必要がある．LBaaS の設定ファイルなど，LBaaS の既存機能によって Keep-Alive の有効無効を制御できなかったため，今回はソースコードを修正して無効化している．

OpenStack と連携が可能な SDN 製品には，iptables 以外の仕組みで実装された仮想ルーターも存在し，本対応が不要なケースがある．逆に SDN 製品やロードバランサーの組み合わせによっては，他の考慮が発生する可能性があり，内部実装の差には注意が必要である．

6.　考察

本章では，一般的な従来システムの更新作業時間と，本手法を適用した場合の更新作業時間を机上でシミュレーションし，本手法の有用性を考察する．

一般的な従来システムについては，次の二つの条件を想定する．1）本番系と検証系という二つの独立したサーバー機器に構築されたウェブアプリケーション．2）仮想化基盤を導入しておらず，検証系で新機能をテストし，移行手順を整えた上で本番環境へ適用することでリリースを行っている．

この従来システムにおいて，さらに更新作業におけるタイムテーブル例を想定し，表 2 に示す．このシステムは毎週日曜日の 23 時から 2 時間のメンテナンスタイムを設けており，リリースはこの時間内で行われる．

次に，想定した従来システムに本手法を適用した場合のタイムテーブル例を表 3 に示す．表 3 において，メンテナンスタイムを 15 分としている理由は，切り替え後 5 分で切り戻し判断をして切り戻す時間を確保するためである．

表2　従来システムの更新作業タイムテーブル例

時刻	作業内容	備考
23時00分	メンテナンスタイム開始 本番系停止	強制ログアウト処理5分 アクセス停止処置5分 バックアップ20分
23時30分	本番系適用作業開始	適用作業30分
24時00分	本番系適用確認テスト開始	テスト可能時間30分
24時30分	切り戻し判断期限時刻	切り戻し所要時間は20分
24時55分	本番系サービス再開準備	アクセス開始処置5分
25時00分	メンテナンスタイム終了	本番系サービス再開

表3　ブルー・グリーン・デプロイメント適用システムの更新作業タイムテーブル例

時刻	作業内容	備考
23時00分	メンテナンスタイム開始 本番系停止	強制ログアウト処理5分
23時05分	本番系切り替え実施 →本番系サービス再開	強制ログアウト終了後，3秒で本番系を切り替えて終了
23時15分	メンテナンスタイム終了	切り替え後5分で切り戻し判断する前提

表3を表2と比較すると，本番系停止作業に，アクセス停止処置とバックアップが存在しない．アクセス停止処理は，強制ログアウト処理の直後に系の切り替えを行うことで省いている．これはブルー・グリーン・デプロイメントの迅速な切り替えという特徴を活かしている．また，切り戻しを想定したバックアップも，旧環境に容易に切り戻せるというブルー・グリーン・デプロイメントの特徴を活かすことで，不要としている．

また，表3には，表2にある本番系適用作業や本番系適用確認テストが含まれないが，これは更新作業前に待機系で同様の作業を完了しているためである．メンテナンスタイムから本番系適用作業を分離することで，メンテナンスタイムの短縮に加えて，時間に追われた状態での作業ミスや，限定されたテストによって引き起こされる更新失敗のリスクも回避している．

このように，従来2時間であったメンテナンスタイムを，ブルー・グリーン・デプロイメントを適用することで15分に短縮し，迅速なリリースを達成できる．さらに，想定したウェブアプリケーションにおいて強制ログアウト処理を必須としているが，これを改修し不要にできれば，明示的なメンテナンスタイム自体を無くし，利用者が意識しないうちに更新版をリリースすることも可能となる．

また，安全性や確実性，スピード以外にも，本手法には優位点があり最後にまとめておきたい．1）移行に関わる工数を削減できるといったコスト面におけるメリット．2）仮想化を採用し，適切にアプリケーションを改修することでスケールアウト構成を達成し，可用性やスケーラビリティが得られる．3）従来あった本番系と検証系の環境の差異に起因する問題が発生しない．4）自動化が促進されることによって確実性や迅速性が向上する．

7. お わ り に

本稿ではシステム開発のリリースにおける自動化手法としてブルー・グリーン・デプロイメントを取り上げ，OpenStack 環境に実装した．そして安全性や確実性を担保したうえで，システム開発に求められるスピードのうち，迅速なアプリケーションのリリースを達成できることを確かめた．また背景となる DevOps や継続的デリバリー，ソフトウェア定義といった考え方を整理した．

今後の展望として，ブルー・グリーン・デプロイメント以外の自動化手法の実装を考えている．研究をすすめることで，DevOps の未来像を描いていきたい．

＊ 1　Velocity 2009（http://velocityconf.com/velocity2009）Velocity は米国オライリー社が主催する技術カンファレンス．

＊ 2　継続的インテグレーションは，ソフトウェア開発における不具合を，開発のワークフローにおける結合テストよりも早い段階で検出しようというプラクティス．一般に不具合の検出が早いほど，修正に要するコストが小さくなるという性質を拠り所としている．自動的に最新のソースコードを取得しコンパイルやビルド，ユニットテストを行い，不具合を早期に検出する．

＊ 3　ブルー・グリーン・デプロイメントは，マーチン・ファウラーによって 2010 年に書かれたブログ記事にて注目された．http://martinfowler.com/bliki/BlueGreenDeployment.html

＊ 4　OpenFlow は，2008 年スタンフォード大学とカリフォルニア大学バークレイ校にて開始された研究によって生まれた，ネットワーク機器においてデータ転送部と制御部を分離する仕組み．データ転送部はネットワーク上のパケットをあるルールによってやりとりする仕組みを指し，制御部は転送のためのルールを決定して転送部へルールを設定する仕組みを指す．制御部を分離し集中管理を実現することにより，ネットワークをより「プログラマブル」に制御できる．

＊ 5　OpenStack には，複数のディストリビューションが存在する．例えば，Redhat や Mirantis，Ubuntu などがあげられる．Ubuntu OpenStack　http://www.ubuntu.com/cloud/ubuntu-openstack

＊ 6　Jenkins は継続的インテグレーションのためのオープンソースのツール．複数サーバーにリモートログインして任意の処理を行うといったことを WebUI 上から実施できるため，本稿では OpenStack Heat の自動構成を実行する基盤として利用している．

＊ 7　HTTP における KeepAlive は，HTTP1.1 の機能である．一つの TCP 接続で複数の HTTP リクエストとレスポンスを行うことで，TCP の 3 ウェイハンドシェイクのオーバーヘッドを削減することができる．ただし，WEB サーバーから見ると，TCP 接続がタイムアウトになるまで，常に確保され続けてしまうという問題があり，しばしば無効化される．

＊ 8　iptables は，Linux に実装されたパケットフィルタリングおよびネットワークアドレス変換（NAT）機能の設定を操作するコマンド．

参考文献　［1］ 10 deploys per day　http://www.slideshare.net/jallspaw/10-deploys-per-day-dev-and-ops-cooperation-at-flickr

［2］ David Farley，Jez Humble，和智 右桂（翻訳），高木 正弘（翻訳），継続的デリバリー 信頼できるソフトウェアリリースのためのビルド・テスト・デプロイメントの自動化，KADOKAWA/アスキー・メディアワークス，2012 年 3 月

［3］ AWS re:invent 2012 keynote Day 2　https://www.youtube.com/watch?v=PW1lhU8n5So （42 分 03 秒付近）

［4］ OpenStack　http://www.openstack.org/

［5］ HAProxy　http://www.haproxy.org/

［6］ F5 Networks　https://f5.com

［7］ A10 Networks　https://www.a10networks.com/

※上記注釈および参考文献中の URL は，2015 年 8 月 17 日時点での存在を確認．

執筆者紹介　佐々木　智一（Tomokazu Sasaki）

1999年ユニアデックス(株)入社．2005年よりIP電話ソリューション AiriPの製品企画，開発，運用，保守を担当．2013年よりSDNやOpenStackに関する技術評価，研究開発活動を担当して現在に至る．

吉　本　昌　平（Shohei Yoshimoto）

福岡のベンチャー企業でASP（Application Service Provider）の設計・開発を経験後，2006年から現職．2012年からはSDN分野の技術調査/研究開発活動を担当．現在はIoT及びSDNを担当．