インタークラウドを実現する技術
～デファクトスタンダードからの視点～

Ver1.0 中井悦司
Twitter @enakai00

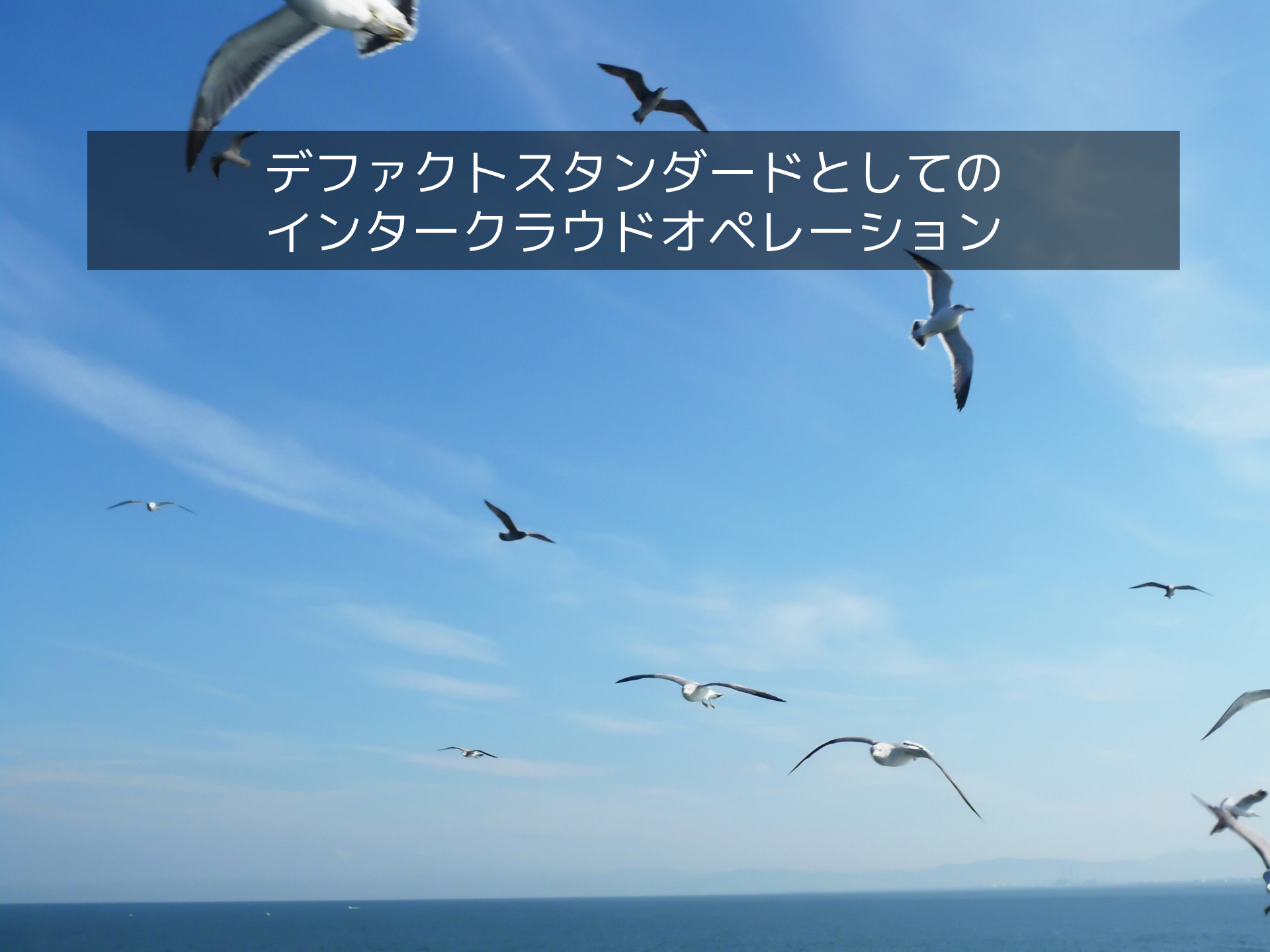
自己紹介

- 中井悦司（なかいえつじ）
  - Twitter @enakai00

- 日々の仕事
  - Senior Solution Architect and Cloud Evangelist at Red Hat K.K.
    企業システムでオープンソースの活用を希望される
    お客様を全力でご支援させていただきます。

- 昔とった杵柄
  - 素粒子論の研究（超弦理論とか）
  - 予備校講師（物理担当）
  - インフラエンジニア（Unix/Linux専門）

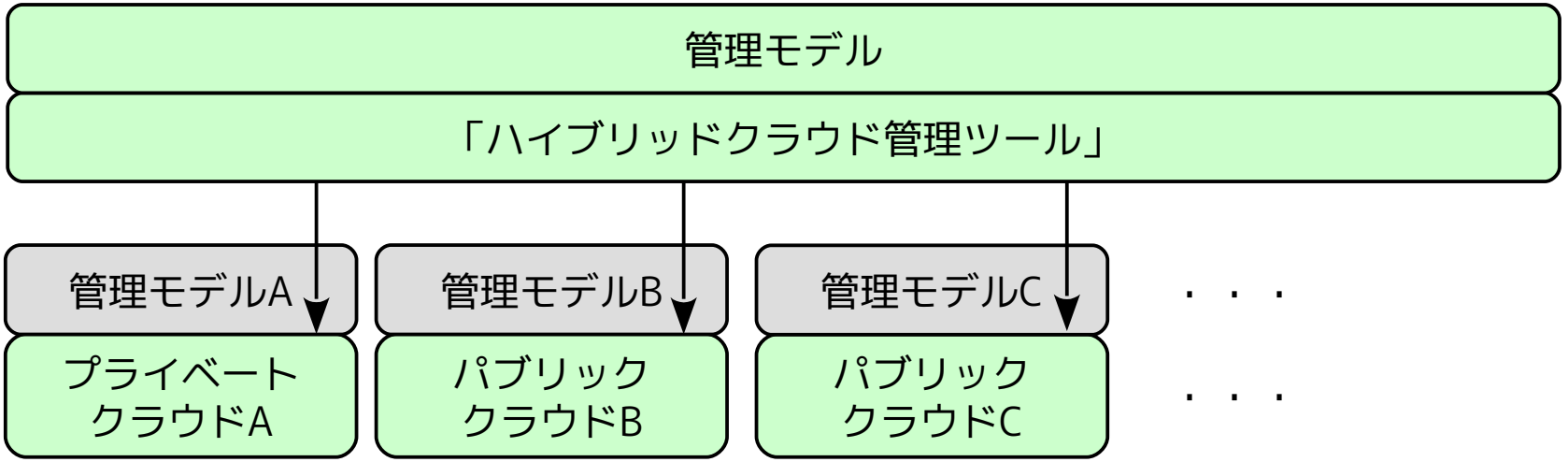
好評発売中！


デファクトスタンダードとしての
インタークラウドオペレーション

「ハイブリッドクラウド管理製品」の課題

- 独自の「クラウド管理モデル」を複数のクラウドに共通適用するアプローチ。
- 利用者からは、配下のクラウドの管理モデルは隠蔽されて見えなくなる。
  - 個々のクラウドの特性を意識した管理が困難。
  - 個々のクラウドの既存の利用者には、管理モデルの変化は受け入れがたい。

⇒ 製品ベンダー主導のトップダウンの管理モデルでは、なかなかうまくいかない。（世の中に普及しない）



クラウドをより便利に使うためのボトムアップの動き

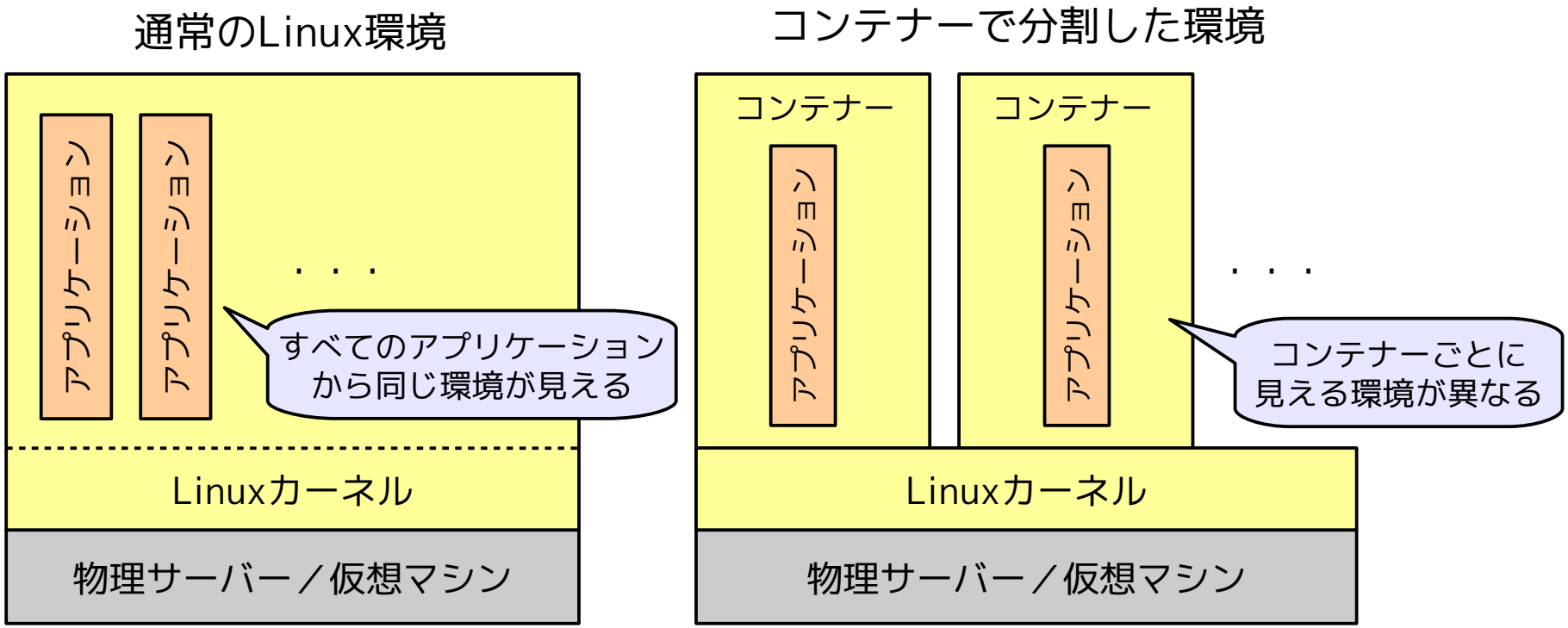
- 既存のIaaS型クラウドをより便利に使うためのいくつかのツールがデファクトスタンダードとして普及を見せ始めています。
  - Docker：アプリケーションデプロイの標準化
  - Kubernetes：マイクロサービスアーキテクチャーの実行基盤
  - Ansible：インフラオペレーションの自動化
  - Jupyter：Literate Computing（手順書と自動化の橋渡し）の実現
- これらのツールは、特定のクラウドインフラを前提としない汎用性を持つため、今後、これらを組み合わせた、デファクトスタンダードとしての「インタークラウドオペレーション」が実現する可能性が高いと考えられます。
- 個々のクラウドの特性を意識した、よりIntelligentな「クラウド群連成基盤」のアーキテクチャー設計／運用設計においては、このような「インタークラウドオペレーション」の理解が重要な役割を果たすものと考えています。


Docker：アプリケーションデプロイの標準化

（参考）Linuxコンテナーの仕組み

- 「Linuxコンテナー」は、プロセスグループごとに独立したOS環境を見せる技術
  - ローカルディスクの内容（ディレクトリー内のファイル）
  - ネットワーク環境（NIC、IPアドレス）
  - CPU、メモリー割り当て

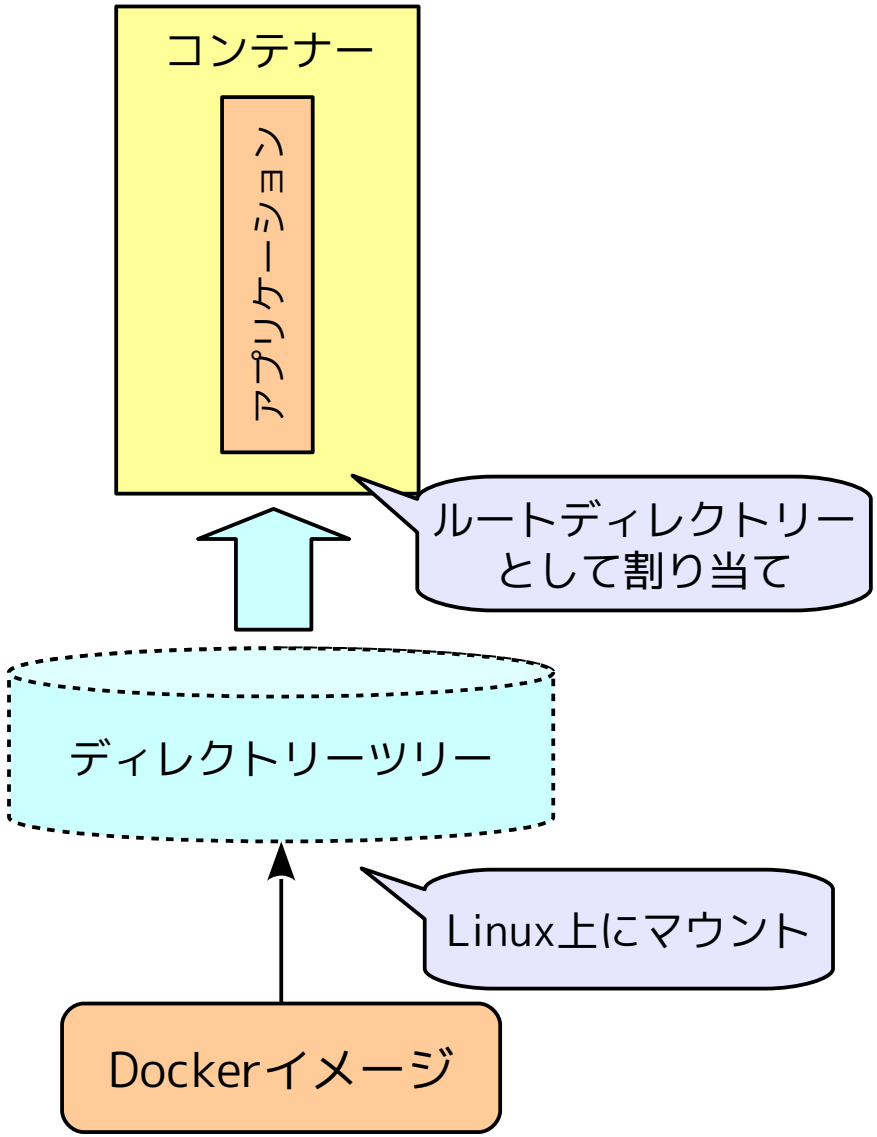
※ Dockerよりもずっと古くから存在する技術です。



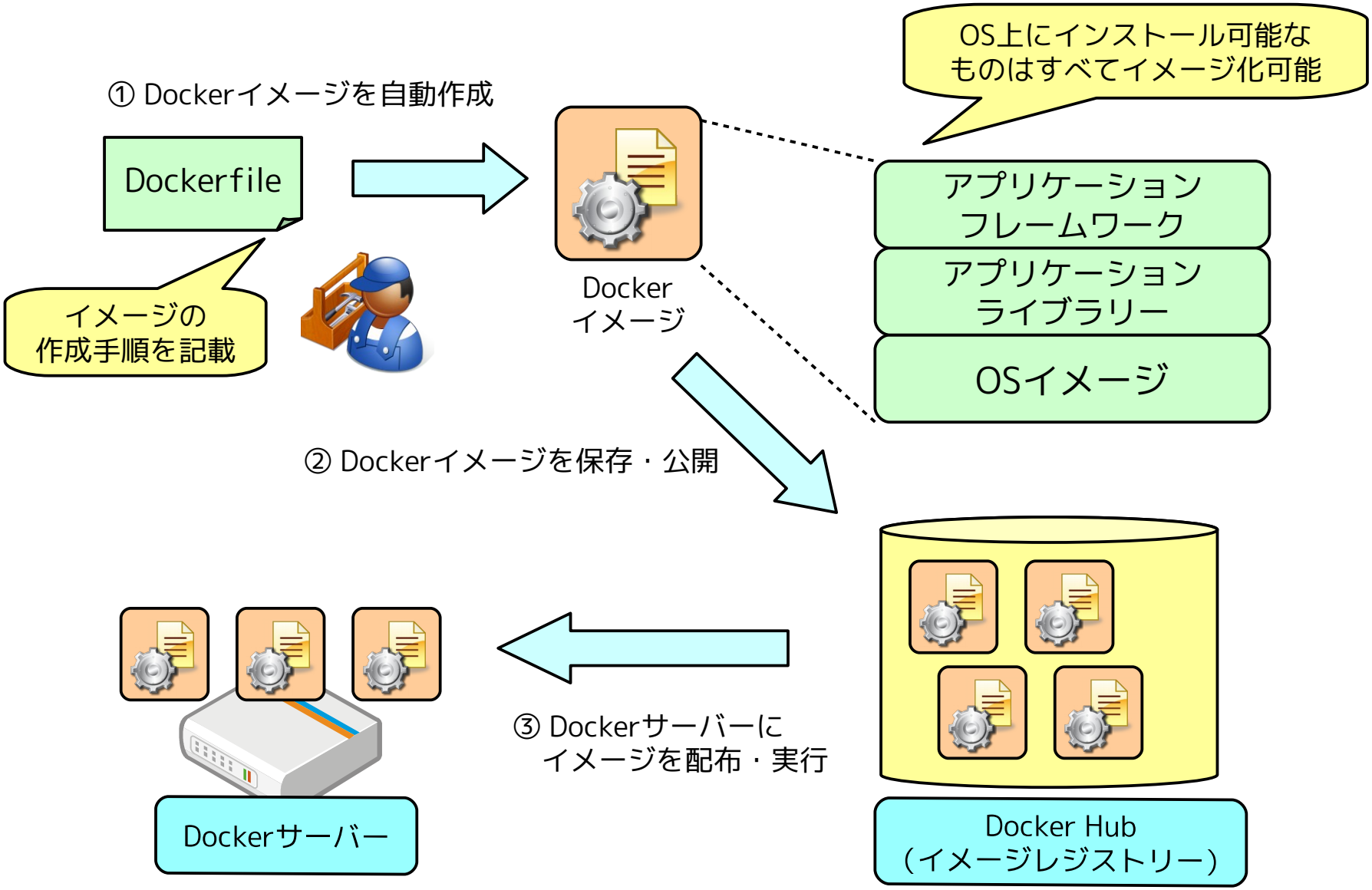
Dockerとコンテナの関係

- 「Dockerイメージ」の実体は、コンテナーに割り当てるディスクイメージに、ネットワーク設定などの環境情報を付与したものにすぎません。

- Dockerの真の価値は、次のような「イメージ管理機能」にあります。
  - Dockerfile：
    Dockerイメージを自動作成する仕組み
  - Docker Hub：
    Dockerイメージを共有・配布する仕組み

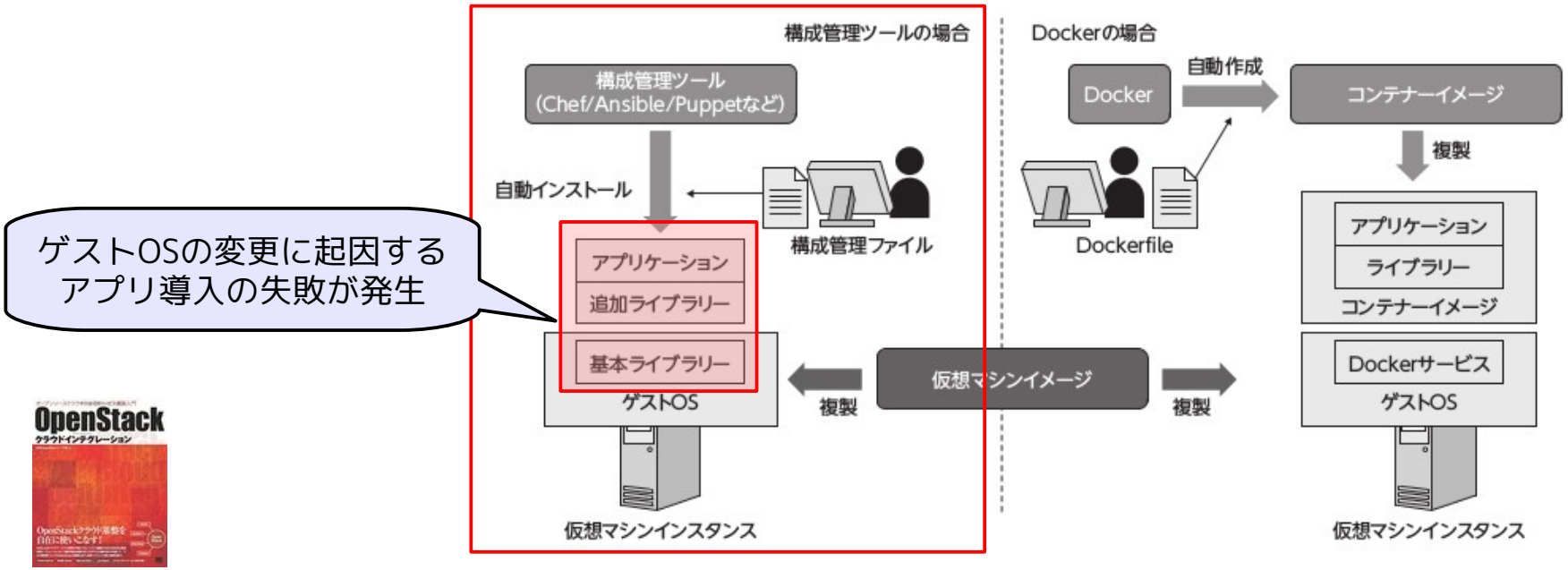


Dockerが提供する基本機能



OpenStackによる自動化（オーケストレーション）手法

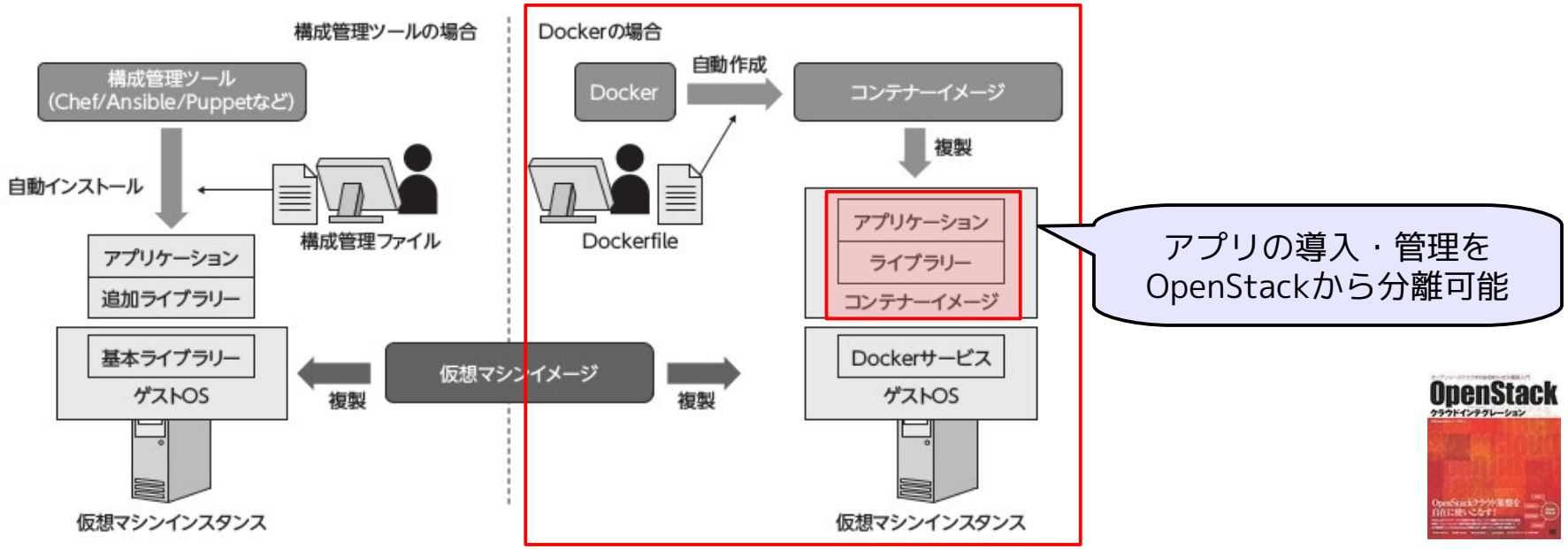
- Dockerが無かった時代は・・・
  - 仮想マシン、ストレージ、ネットワークなどのインフラは、OpenStackで自動構成
  - ゲストOS上のアプリはChef/Ansible/Puppetなどの構成管理ツールで自動構成
- ゲストOSとアプリの管理が別れているため「Immutable」な運用が困難！
  - ゲストOSのテンプレートはOpenStack側で管理
  - 仮想マシン起動時に動的にアプリの導入・設定を実施



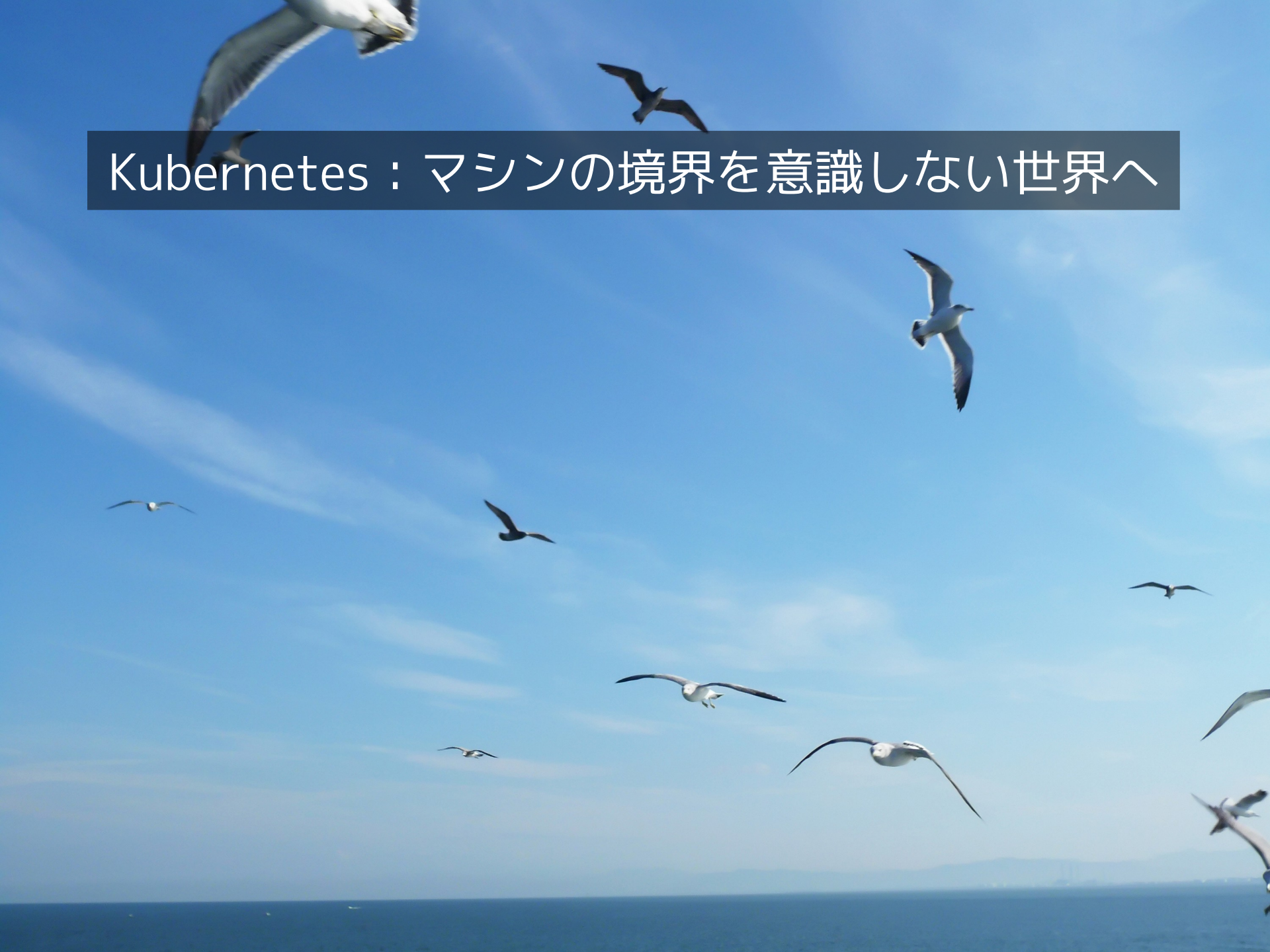
「第14章 Dockerを利用したアプリケーション展開」より引用

OpenStackとDockerの組み合わせ手法

- Dockerを用いた運用だと・・・
  - OpenStackは、「インフラ＋DockerホストOS」の提供に専念
  - アプリの実行環境は、Dockerイメージで作成・管理・デプロイ
- インフラとアプリの管理を分離することで「Immutable」な運用が容易に！
  - ゲストOSのテンプレートはDockerの稼働環境を提供
  - 事前作成済みのDockerイメージを配布してアプリを起動



「第14章 Dockerを利用したアプリケーション展開」より引用


Kubernetes：マシンの境界を意識しない世界へ

参考文献

- Large-scale cluster management at Google with Borg
  - http://research.google.com/pubs/pub43438.html
- Borg, Omega, and Kubernetes
  - http://research.google.com/pubs/pub44843.html

おすすめ

Large-scale cluster management at Google with Borg

Abhishek Verma† Luis Pedrosa‡ Madhukar Korupolu
David Oppenheimer Eric Tune John Wilkes
Google Inc.

manager that runs hun-
any thousands of differ-
clusters each with up to

mbining admission con-
mmitment, and machine
nce isolation. It supports
ntime features that min-
duling policies that re-
failures. Borg simplifies
arative job specification
, real-time job monitor-
te system behavior.
Borg system architecture
ions, a quantitative anal-
s, and a qualitative ex-
a decade of operational

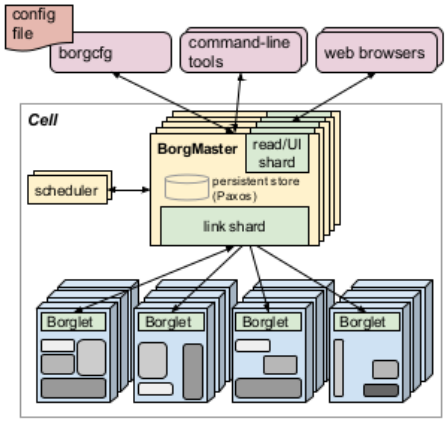


Figure 1: The high-level architecture of Borg. *Only a tiny fraction of the thousands of worker nodes are shown.*

Borg, Omega, and Kubernetes

LESSONS LEARNED FROM THREE CONTAINER-MANAGEMENT SYSTEMS OVER A DECADE

BRENDAN BURNS, BRIAN GRANT, DAVID OPPENHEIMER, ERIC BREWER, AND JOHN WILKES, GOOGLE INC.

Though widespread interest in software containers is a relatively recent phenomenon, at Google we have been managing Linux containers at scale for more than ten years and built three different container-management systems in that time. Each system was heavily influenced by its predecessors, even though they were developed for different reasons. This article describes the lessons we've learned from developing and operating them.

Application Oriented Infrastructure

- 「コンテナによる隔離と依存性の最小化は、グーグル社内で非常に効果的であることがわかったので、グーグルの社内インフラでは、唯一コンテナだけを利用できるようにした」

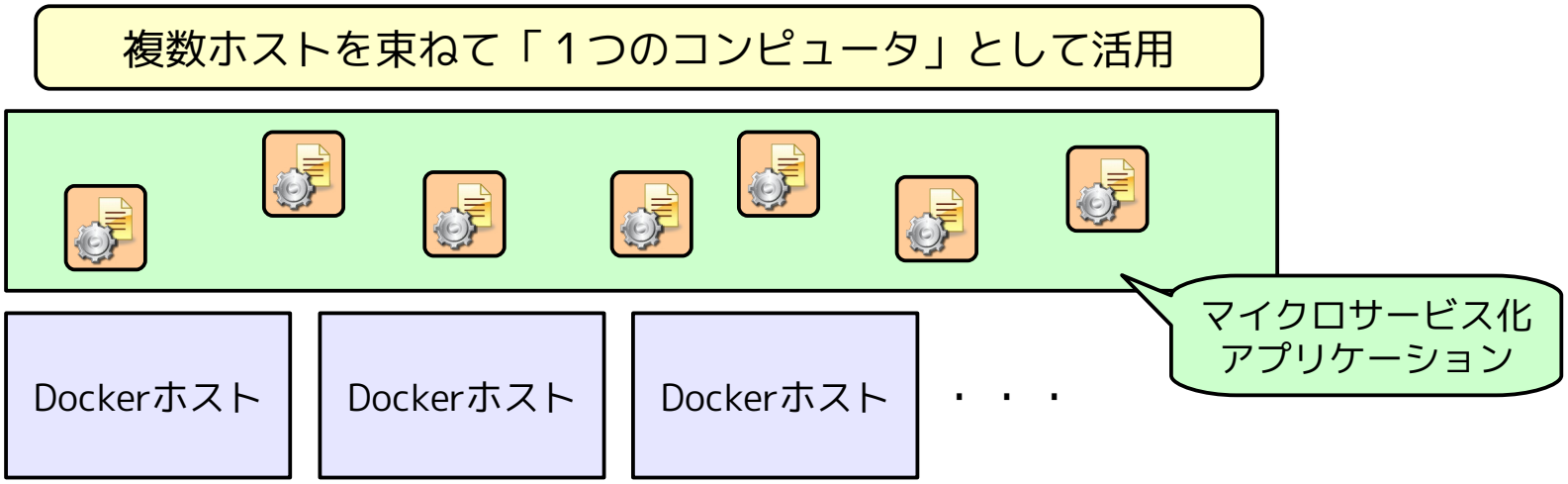
- 「個々のサーバーではなく、コンテナを管理するAPIを用意することで、データセンターの『プライマリーキー』をサーバーからアプリケーションへ変化させた」

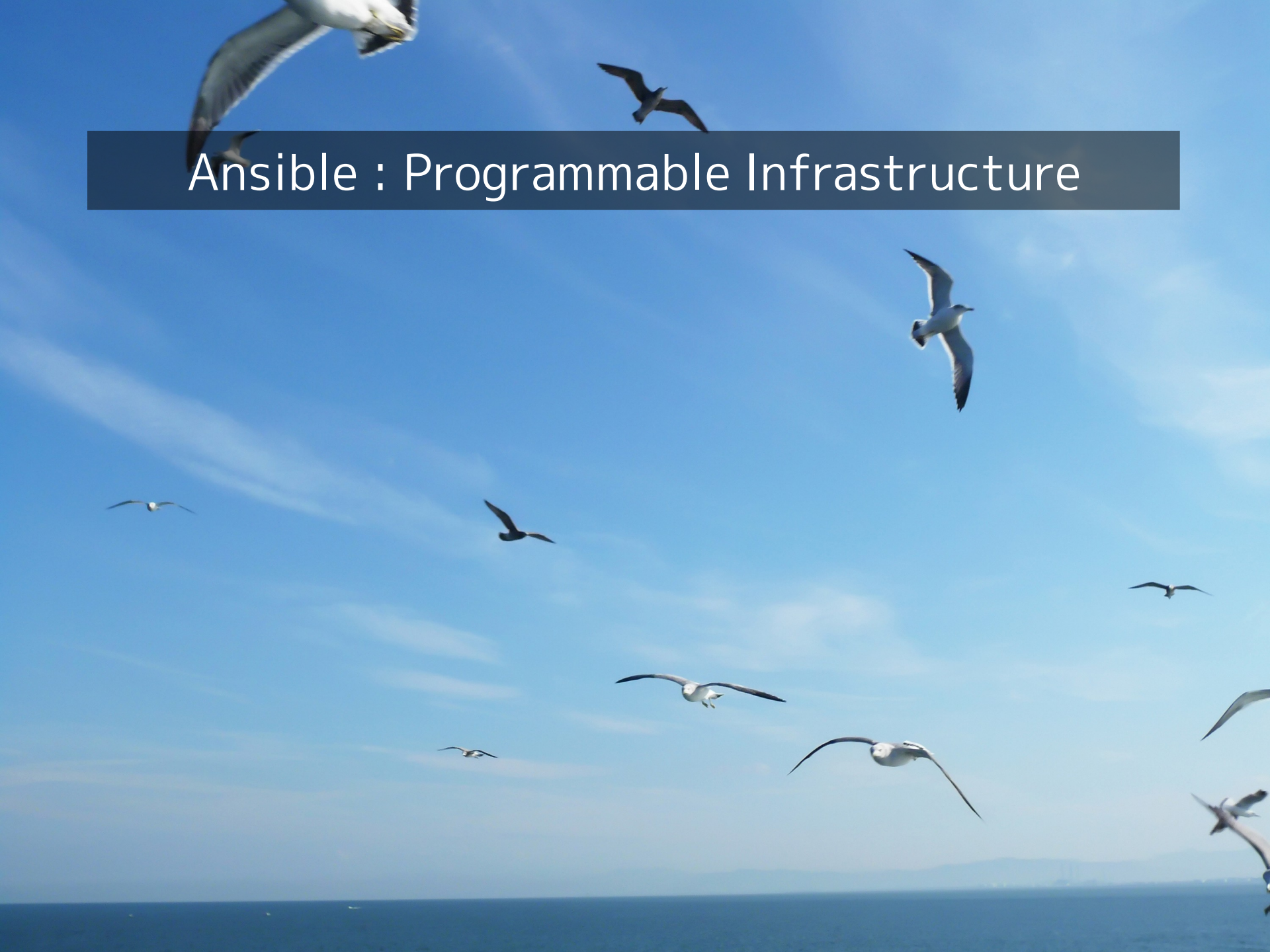
  – アプリケーション開発者、および、運用担当者は、個々のサーバーやOSの設定を気にする必要がなくなった。
  – インフラチームは、実行中のアプリケーション、あるいは、その開発者に大きな影響を与えることなくハードウェアやOSの更新を実施できるようになった。
  – リソースのモニタリングをサーバーではなく、アプリケーション単位で実施するように変わり、アプリケーション監視や問題判別の精度が向上した。

⇒ 個々のハードウェアやOSの違いをアプリケーション開発者に意識させず、「1つのコンピューターであるかのように利用可能にした」

サーバーの境界を意識しないアプリケーションデプロイ

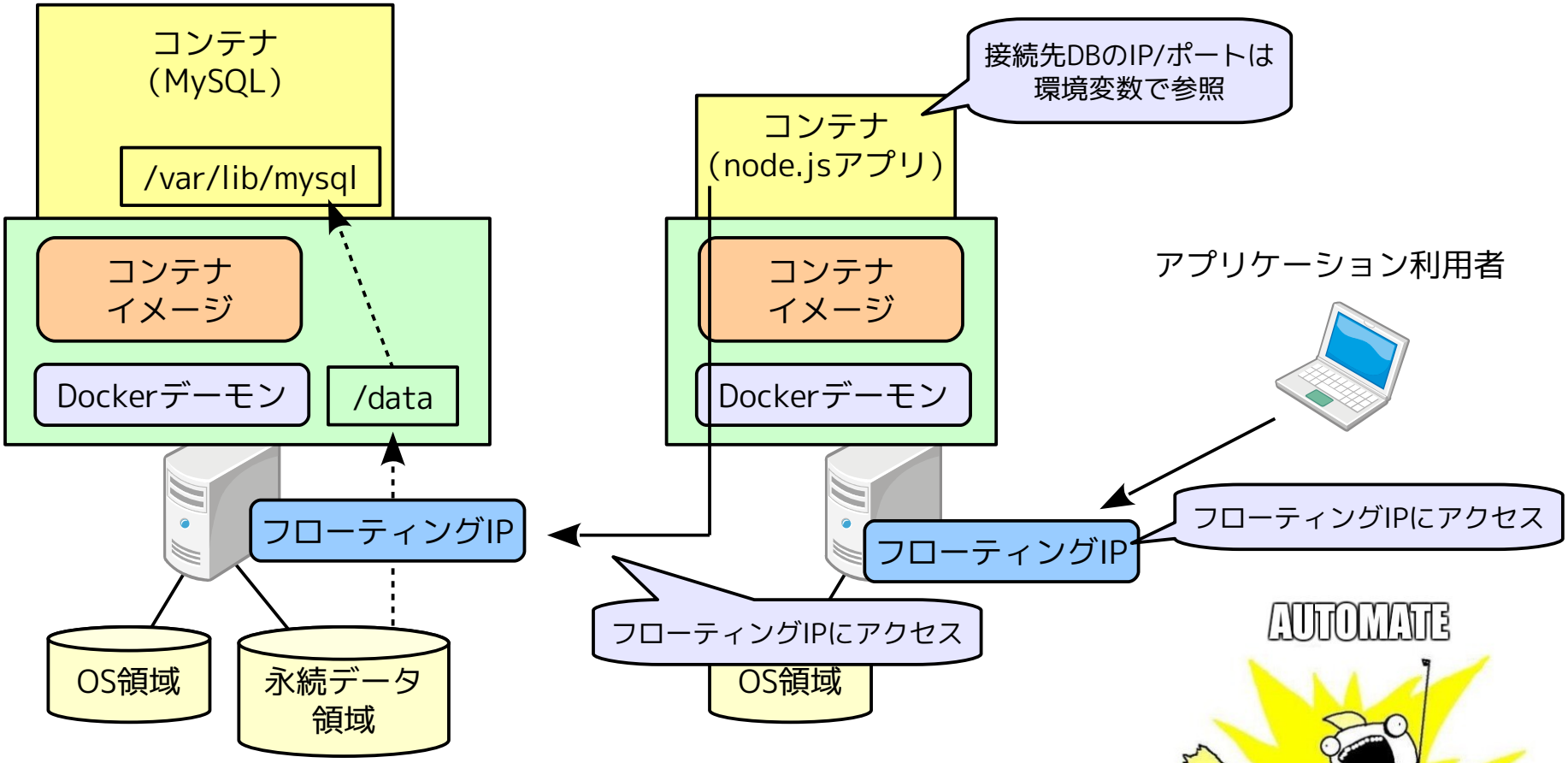
- コンテナの配置先を自動的に振り分ける仕組みを用いて、複数ホストを「１つのコンピューティングリソース」として活用します。
- アプリケーションを機能単位に分割してコンテナ化することで、さらなるメリットが得られます。
  - 必要な機能を負荷に応じてオートスケールします。
  - 機能単位でコンテナを入れ替えることにより、稼働中のアプリケーションの動的な機能変更が可能になります。




Ansible : Programmable Infrastructure

Ansibleによる複数インスタンス環境のオーケストレーション

- Ansibleを利用すると、OpenStack APIによる仮想インフラの構成とDockerによるアプリケーション配布のワークフローをまとめて自動化も可能に



（参考）Ansibleのプレイブック構成

- インフラ構築に関連する個々の作業を「サブルーチン化」したプレイブックを組み合わせることで、作業全体をまとめて自動化します。
  - このデモで使用するプレイブックは、下記で公開しています。
    - https://github.com/enakai00/ansible_eplite

```
- include: lib/prep_tenant.yml

- include: lib/create_instances.yml
  vars:
    servers:
      - name: epmysql
        meta: "managed=yes"
      - name: eplite
        meta: "managed=yes"

- include: lib/attach_volumes.yml
  vars:
    volumes:
      - name: mysql_volume
        volsize: 2
        server: epmysql
```

OpenStackのテナント環境を初期設定

仮想マシンインスタンスを起動

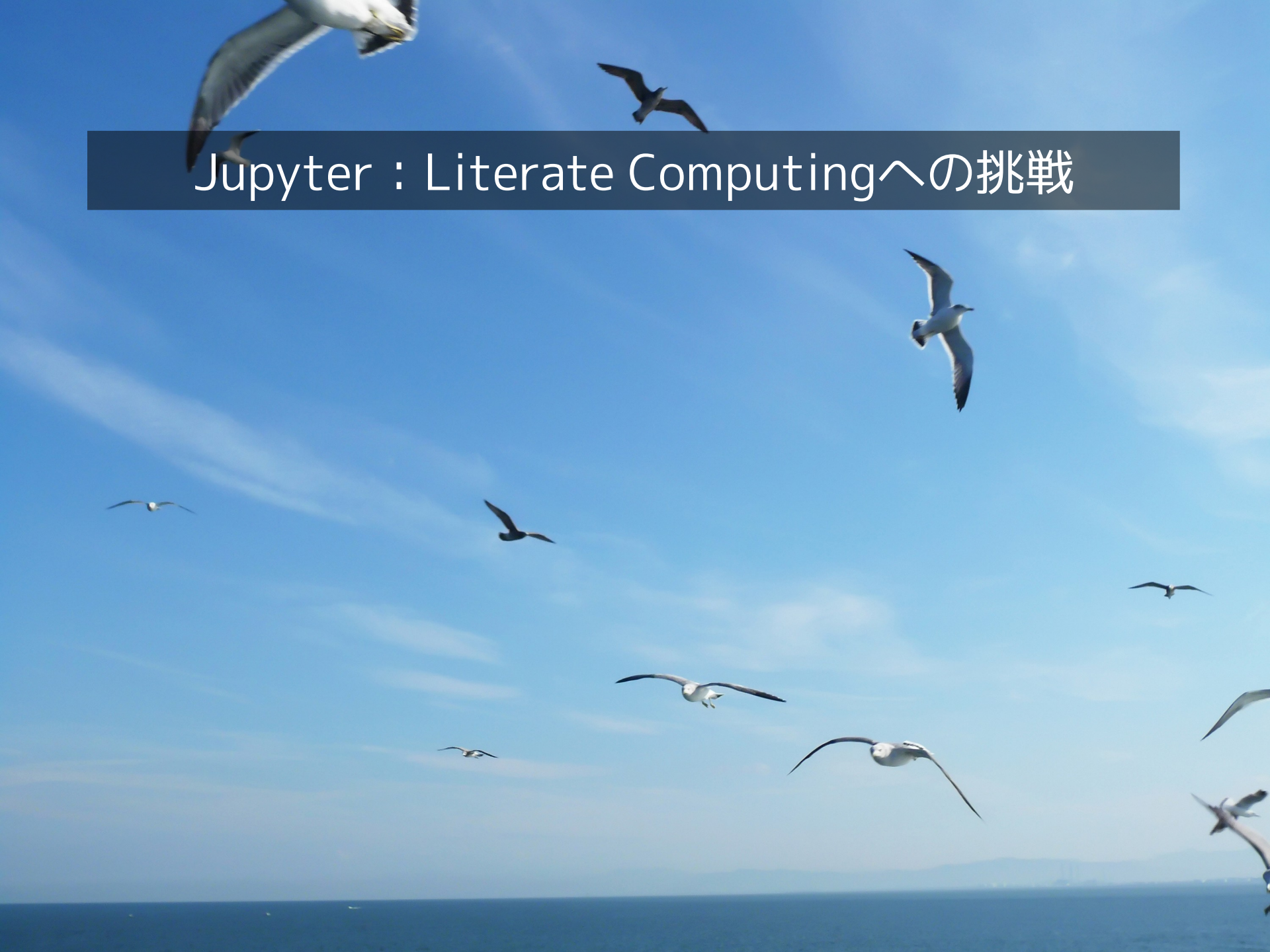
ブロックボリュームを作成して接続

```
- include: lib/check_reachable.yml
  vars:
    servers:
      - epmysql
      - eplite

- include: lib/mount_volume.yml
  vars:
    server: epmysql

- include: lib/deploy_eplite.yml
```

ゲストOSの起動を確認

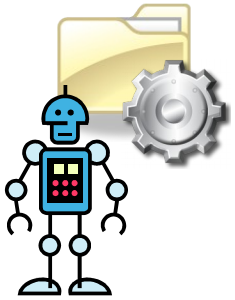
ブロックボリュームをフォーマットしてマウント

Dockerイメージを配信してアプリケーションを起動


Jupyter：Literate Computingへの挑戦

Ansibleのメリットと課題

- Ansibleは、従来の「作業手順書」を「プレイブック」と呼ばれる一連のコードに置き換えます。
  - 手作業で行っていた一連の作業をまとめて自動化。
  - 複数の対象に対して同じ作業を繰り返し、もしくは、並列に実行。
  - 操作対象が多数ある場合の作業効率は確実に向上します。



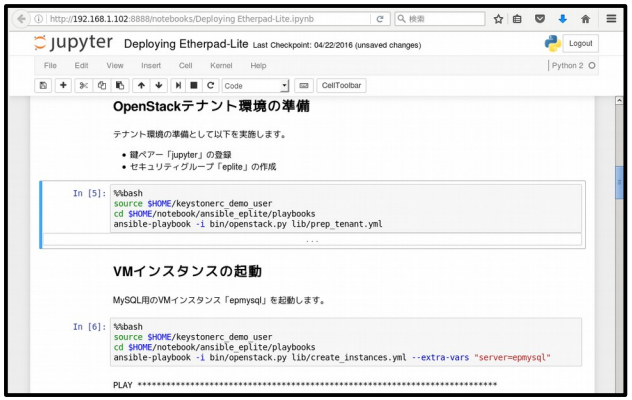
- 一方、Ansibleだけでは解決できない問題もあります。
  - プレイブックそのもののメンテナンス機能は提供しません。
  - どのマシンにどのプレイブックを適用したかという作業記録は、別途、人間が管理する必要があります。

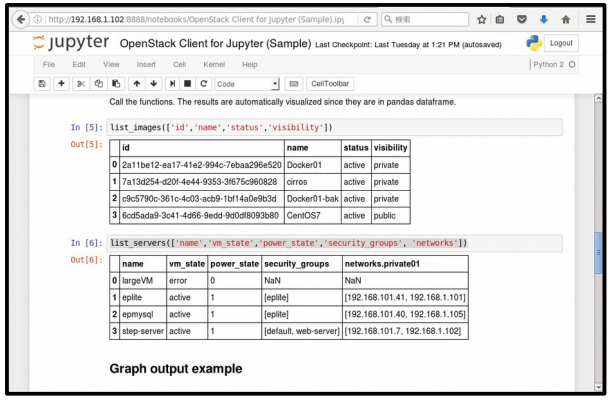
- Ansibleだけでは幸せになれない本質的な理由
  - 自動化されているかどうかに関わらず、作業手順の中身には、管理者／作業者の暗黙知が含まれており、それを明示的にコミュニケートして、改善していく手段が必要。

手順書をコードに置き換えるのではなく、
コードのように自動実行／リファクタリング可能な
「新しい形式の手順書」を作ればよいのでは？！

Jupyter Notebookで実現する「実行可能な手順書」

- 見た目は従来通りの手順書
  - 作業内容の説明とそれを実施するコマンドをセットで記述
- 記載されたコマンドはそのまま実行可能
  - 端末にコマンドをコピペして指差し確認するという非生産的作業を排除
  - 環境に応じてコマンドのオプションを修正して実行するなども可能
  - コマンドの実行結果をそのまま作業履歴として保存可能
- 一定のまとまった処理をAnsibleのプレイブックで実施
  - 複数マシンに対する作業をまとめて実行
  - プレイブックの粒度を適切に保つことで、ブラックボックス化を防止





https://github.com/enakai00/ansible_eplite/blob/jupyter/Deploying%20Etherpad-Lite.ipynb
https://github.com/enakai00/jupyter_samples/blob/master/OpenStack%20Client%20for%20Jupyter%20(Sample).ipynb

参考資料

- Literate Computing for Infrastructure - インフラ・コード化の実践における IPython (Jupyter) Notebookの適用
  - http://www.slideshare.net/nobu758/literate-computing-for-infrastructure-ipython-jupyter-notebook
- Literate Automation（文芸的自動化）についての考察
  - http://enakai00.hatenablog.com/entry/2016/04/22/204125

まとめ

- 個々のクラウドの特性を隠蔽するのではなく、利用者自身がそれぞれの特性を意識した上で、適切に組み合わせることを支援する技術が必要

- 上記の観点で、デファクトスタンダードのツールが果たす役割を理解することが大切
  - Docker：環境に依存しないアプリケーションデプロイの標準ツール
  - Kubernetes：マイクロサービス化により機能単位でデプロイ先をコントロール
  - Ansible + Jupyter：環境の違いを意識したオペレーションを適切な粒度で自動化（Literate Computingの実現）

Thank You!

中井悦司
Twitter @enakai00