Cloud Computing から Cloud Native への潮流

さくらインターネット株式会社
クラウド事業本部
前佛雅人 @zembutsu
2021年10月25日

SAKURA internet

https://docs.docker.jp/

ZEMBUTSU Masahito

前佛 雅人

@zembutsu
@zembutsu_works

zembutsu

さくらインターネット株式会社 クラウド事業本部
Technology Evangelist / Developer Advocate

- ホスティング・ISP事業者のサポートおよび保守としてキャリアをスタート
- 興味範囲：運用監視、自動化、オープンソース関連技術の調査・翻訳、プログラミング教育
- 最近の趣味：ランニング、料理

私は何をしたいの？：情報処理技術者ではない普通の人が、当たり前に計算資源を活用し、豊かな生活を育めるように支援する

Cloud Computing から Cloud Native への潮流

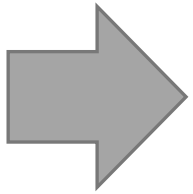
- 日本における「**Cloud Computing**」認知から10年、改めてクラウドを問う
  - ビジネスチャンス先行のための手段（例：コロナ禍で強いられるクラウド対応）
- 「**Cloud Native**」導入には、しかるべき道順(Trail)がある
  - スケール可能なサービス※実現に、**コンテナ化** → CI/CD → オーケストレーション

※広義の”サービスとしてのソフトウェア”

- 「**Docker**」や「**コンテナ**」とは？ ＆ Hacobune(β) ご紹介

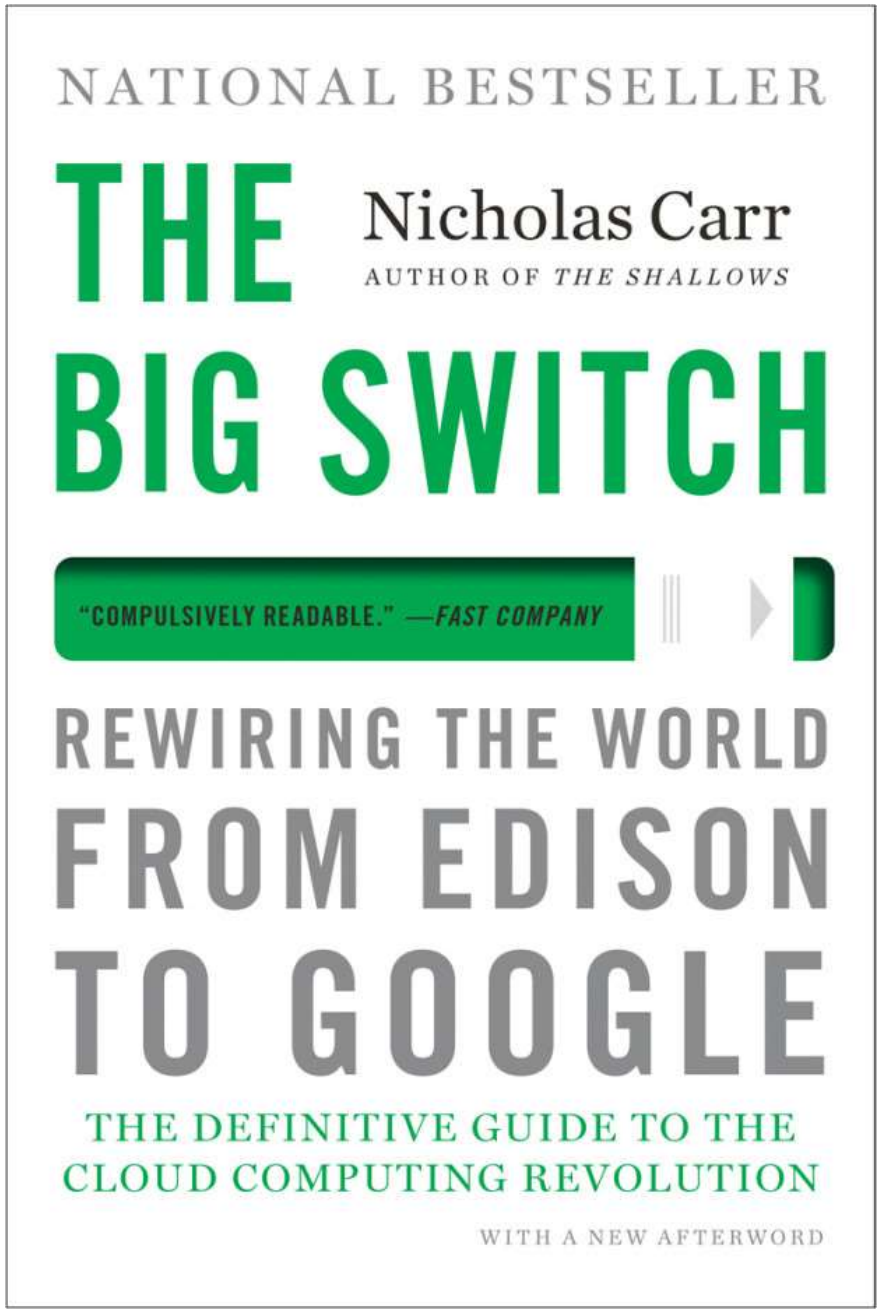
Cloud Computing

クラウド
移行
Cloud Migration



クラウド
ネイティブ
Cloud Native

- データセンタが提供するインフラサービスは、物理サーバから仮想化、そしてクラウドコンピューティングへと変わりつつあります。
- 近年は新しい要素としてコンテナ技術が注目を集めています。


NATIONAL BESTSELLER
THE BIG SWITCH
Nicholas Carr
AUTHOR OF THE SHALLOWS
"COMPULSIVELY READABLE." —FAST COMPANY
REWIRING THE WORLD FROM EDISON TO GOOGLE
THE DEFINITIVE GUIDE TO THE CLOUD COMPUTING REVOLUTION
WITH A NEW AFTERWORD

The Big Switch: Rewriting the world, From Edition to Google

- 20世紀の「電力会社」のように、インターネットを通したクラウドコンピューティング
- Nicholas G. Carr
- 2008年1月発売（邦訳10月発売）
- 翔泳社

電力会社(20世紀初頭)の例え

巨大電力
ネットワーク

ELECTRIC GRID

- 発電設備
- 送電設備
- 保守

- 電力網を通した
  家電の普及・利用
- 豊かな生活

クラウド
コンピュー
ティング

COMPUTING GRID

- サーバ設備(計算)
- ネットワーク設備
- 保守

- スマートフォンを通した
  アプリやサービスの
  普及・利用
- 豊かな生活

クラウドは特別な存在ではなくなった

ROUGH TYPE

Nicholas Carr's blog

HOME ABOUT UTOPIA IS CREEPY THE GLASS CAGE THE SHALLOWS NICHOLAS

The Big Switch: ten years on

My second book, *The Big Switch: Rewiring the World, from Edison to Google*, celebrates its tenth birthday this year. The book, which came out in January 2008, heralds the coming of the cloud and speculates on its consequences. It's hard to imagine now, but in 2008 cloud computing was a new and largely unproven concept, and the common wisdom was that it wouldn't work. Software programs running in centralized server farms and delivered over the internet to users would be too slow and balky, it was thought, to displace the programs running on hard drives inside personal computers or on servers in data centers owned by individual companies. The naysayers were wrong. The technical barriers fell, network latency evaporated, and in short order computing went from being a decentralized resource to being a centralized one — a utility, essentially. The computer scientists, engineers, and programmers who made this monumental technical shift possible still haven't received their due, and they probably never will. The real work went on behind the scenes, anonymously, in companies like Google, Salesforce.com, Amazon Web Services, Akamai, and Facebook, among many others.

The Big Switch has two parts. The first, "One Machine," draws a parallel between the building of the electric grid a hundred years ago and the building of the cloud today. In both cases, a decentralized resource essential to society (power, data processing) was centralized through the construction of a distribution network (electric grid, internet) and central plants (generation stations, server farms). The stories of the electric grid and the computing grid are both stories of technical ingenuity and fearlessness. The book's second part, "Living in the Cloud," is darker. In fact, it was during the course of writing it that my view of the future of computing changed. I began *The Big Switch* believing that the new computing grid would democratize the use of computing power even as it centralized the machinery of data processing. That is, after

http://www.roughtype.com/?p=8314

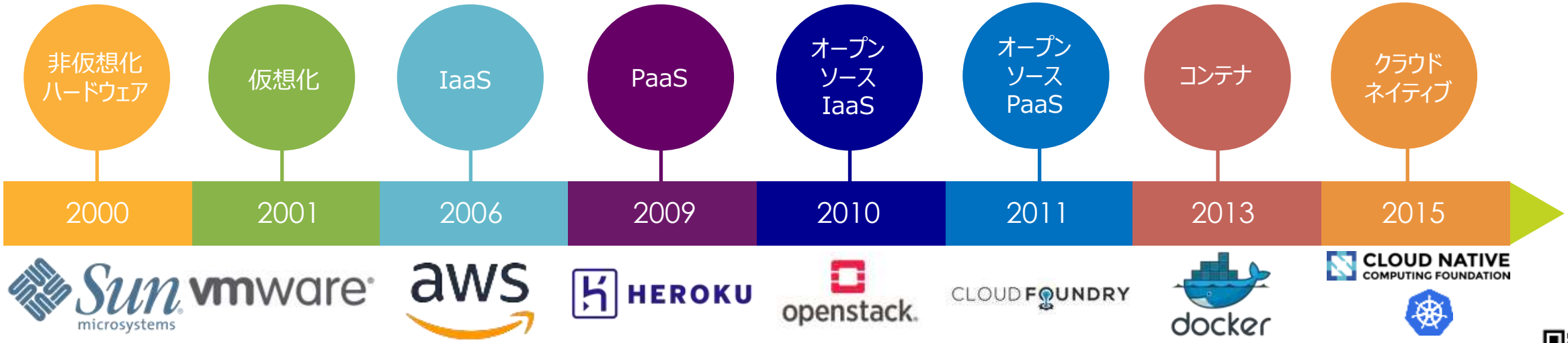
- 個々のマシンやデータセンタ上のサーバにデータを持ち、システムを動かす時代の終焉を予測
- ただし、
  1つのマシン（One Machine）に集約されるのではなく、分散した

仮想化からクラウド・ネイティブへ
From Virtualization to Cloud Native

CLOUD NATIVE COMPUTING FOUNDATION

kubernetes

- クラウド・ネイティブ・コンピューティングはオープンソースのソフトウェアを積み重ね、次のために用います：
  - アプリケーションをマイクロサービス(*microservices*)に分割し、
  - 各パーツ自身をコンテナにパッケージし、
  - リソース利用を最適化するために、動的に統合 / オーケストレート(*orchestrate*)する



※ https://github.com/cncf/presentations からの引用

“ペット vs 家畜”

Pattern:
Scale-out, not UP

Scale Up: (Virtual*)
Servers are like pets

garfield.company.com

You name them and when they get sick, you nurse them back to health

Scale Out: (Virtual*)
Servers are like cattle

web001.company.com

You number them and when they get sick, you shoot them

attrib: Bill Baker, Distinguished Engineer, Microsoft
* added by yours truly ...

19

cloudscaling

Architectures for open and scalable clouds
http://www.slideshare.net/randybias/architectures-for-open-and-scalable-clouds

Cloud Native Computing Foundation (CNCF)

ベンダロックイン無くクラウドを移動できるように

CLOUD NATIVE
COMPUTING FOUNDATION

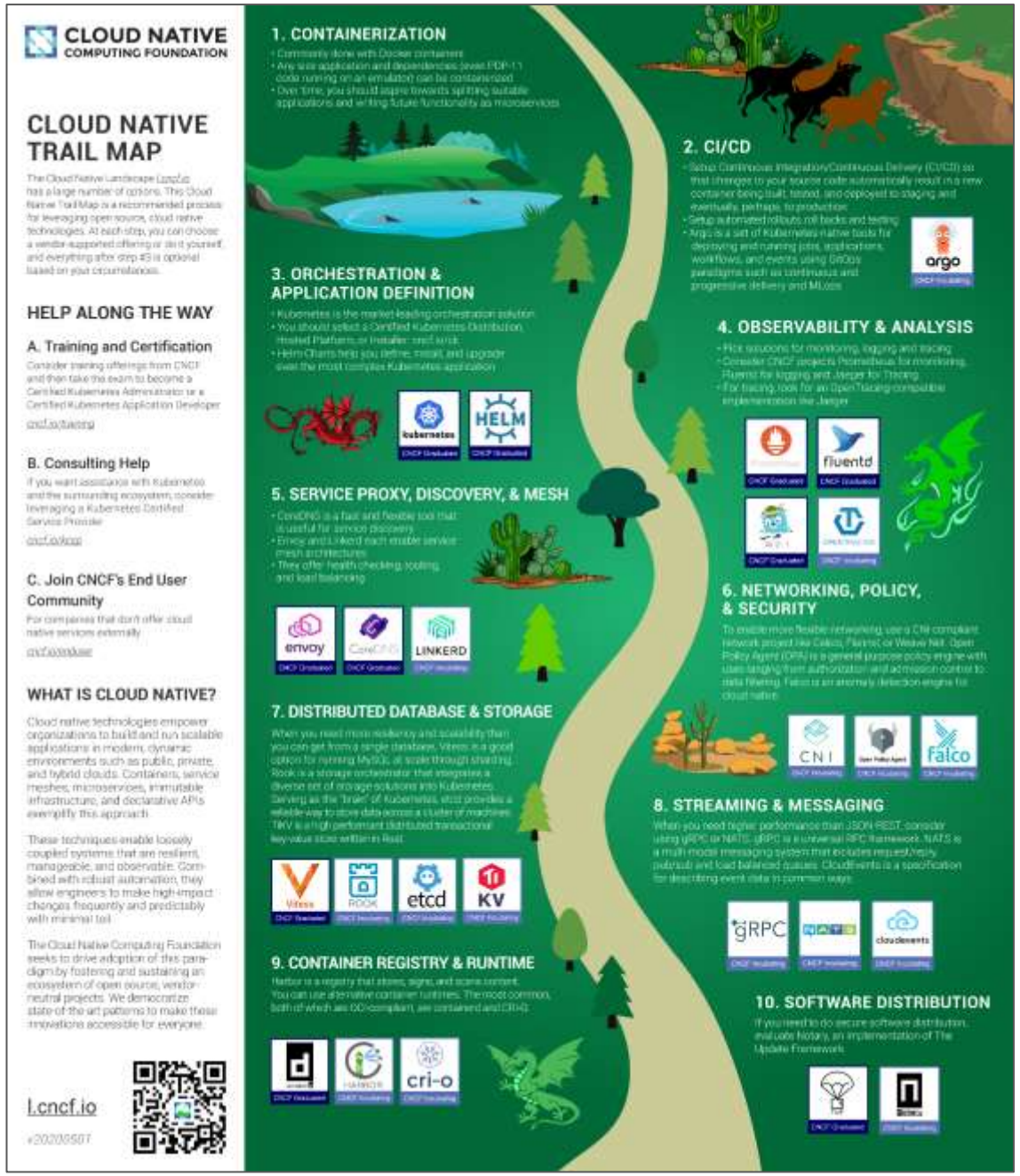
2015年設立 ⬆ プロジェクトを支援

2000年設立。Linuxを中心としたオープンソースのエコシステムを築くため、コンピュータ業界を中心に自動車業界など、50以上のサブプロジェクトを持つ。幅広く業界との調整や標準化のために努める非営利団体。

オープンソースのソフトウェアを積み重ねて：

- **コンテナ化**
  アプリケーションやプロセス等の各パートをコンテナ内にパッケージ化し、再利用性、透明性、、リソースを分離
- **動的なオーケストレーション**
  コンテナを活発にスケジュールし、リソース利用率の最適化を管理
- **マイクロサービス指向**
  アプリケーションをマイクロサービスに分割し、全体的な敏捷性(agility)とメンテナンス性を極めて向上

【参考】CNCF About https://www.cncf.io/about/faq/



CLOUD NATIVE Landscape

TRAILMAP

コンテナやマイクロサービスのようなクラウドネイティブ技術は、組織が開発とデプロイをスケーラブルに、アジャイル・アプリケーションとサービスを動的に、かつ分散環境で行えるようになります。これらの特長の考慮した設計のシステムであれば、回復力のある（resilient）、弾力的（elastic）で、疎結合（loosely coupled）となり、管理可能な抽象化と宣言型の API を通すので、つまり、効率的で信頼性のある自動化をもたらすのです。これにより、エンジニアはアプリケーションを監視できたり、影響の大きな変更も簡単に行えたりします。そして**プロセスとワークフローは結果的に、これらの環境を最大限に活かし、苦痛を最小化するのです。**

https://github.com/cncf/trailmap

1. CONTAINERIZATION

- Commonly done with Docker containers
- Any size application and dependencies (even PDP-11 code running on an emulator) can be containerized
- Over time, you should aspire towards splitting suitable applications and writing future functionality as microservices

2. CI/CD

- Setup Continuous Integration/Continuous Delivery (CI/CD) so that changes to your source code automatically result in a new container being built, tested, and deployed to staging and eventually, perhaps, to production
- Setup automated rollouts, roll backs and testing
- Argo is a set of Kubernetes-native tools for deploying and running jobs, applications, workflows, and events using GitOps paradigms such as continuous and progressive delivery and MLops

argo
CNCF Incubating

3. ORCHESTRATION & APPLICATION DEFINITION

- Kubernetes is the market-leading orchestration solution
- You should select a Certified Kubernetes Distribution, Hosted Platform, or Installer: cncf.io/ck
- Helm Charts help you define, install, and upgrade even the most complex Kubernetes application

kubernetes
CNCF Graduated

HELM
CNCF Graduated

4. OBSERVABILITY & ANALYSIS

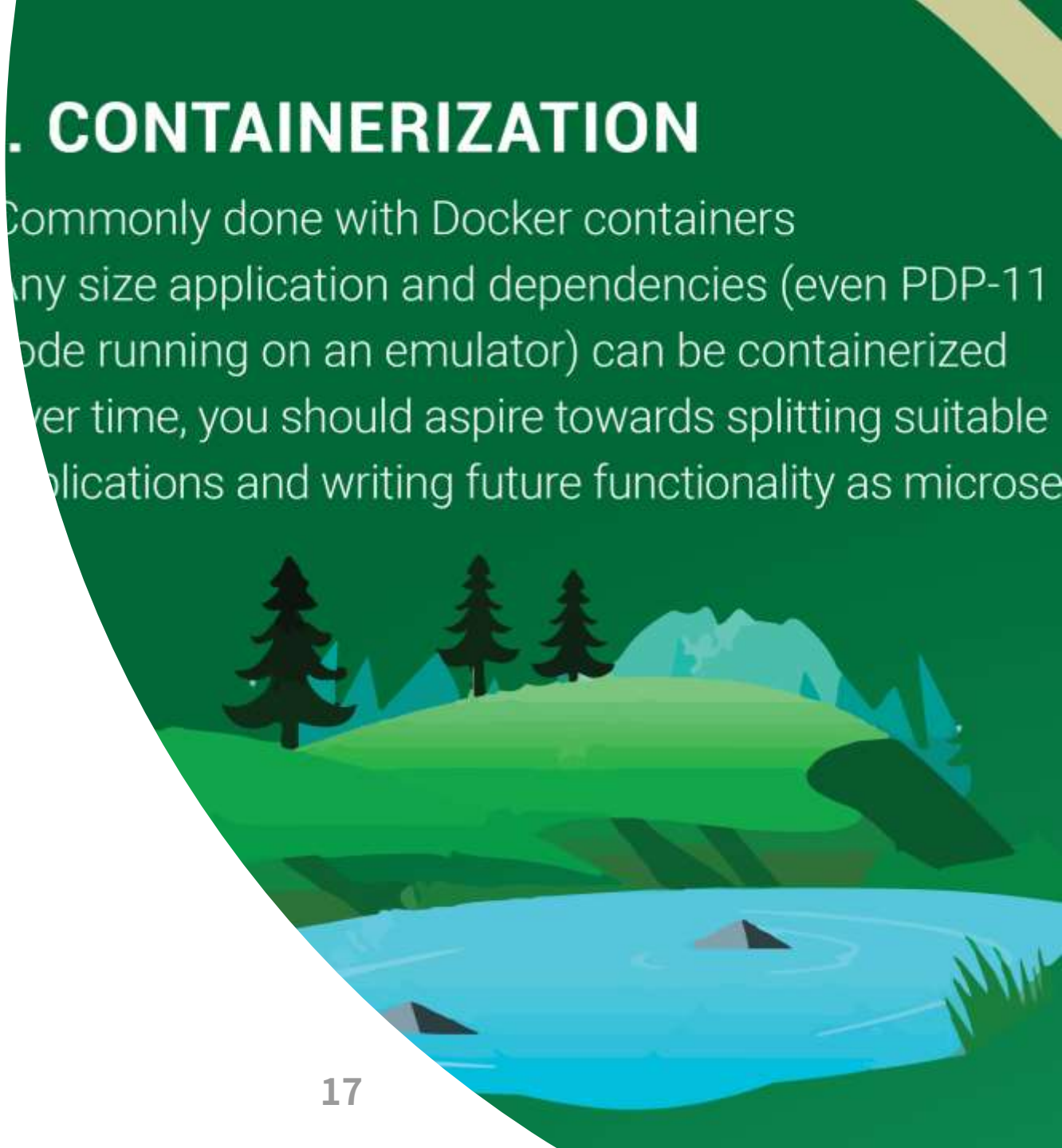
- Pick solutions for monitoring, logging and tracing
- Consider CNCF projects Prometheus for monitoring, Fluentd for logging and Jaeger for Tracing
- For tracing, look for an OpenTracing-compatible implementation like Jaeger

Prometheus
CNCF Graduated

fluentd
CNCF Graduated

1. コンテナ化

- 一般的にDocker コンテナを使う。
- あらゆる大きさのアプリケーションと依存関係を（エミュレータ上で実行する PDP-11 のコードですら）コンテナ化できる。
- 時間がたてば、適切なアプリケーションの分割を求めるようになり、次世代の機能をマイクロサービスとして書くでしょう。



開発環境 → テスト環境 → 準備環境 → 本番環境

サービスを提供したい

アプリを動かしたい

課題 都度、環境構築

課題 異なるOS環境

課題 都度、プロビジョニング

課題 動かない

課題 時間がかかる

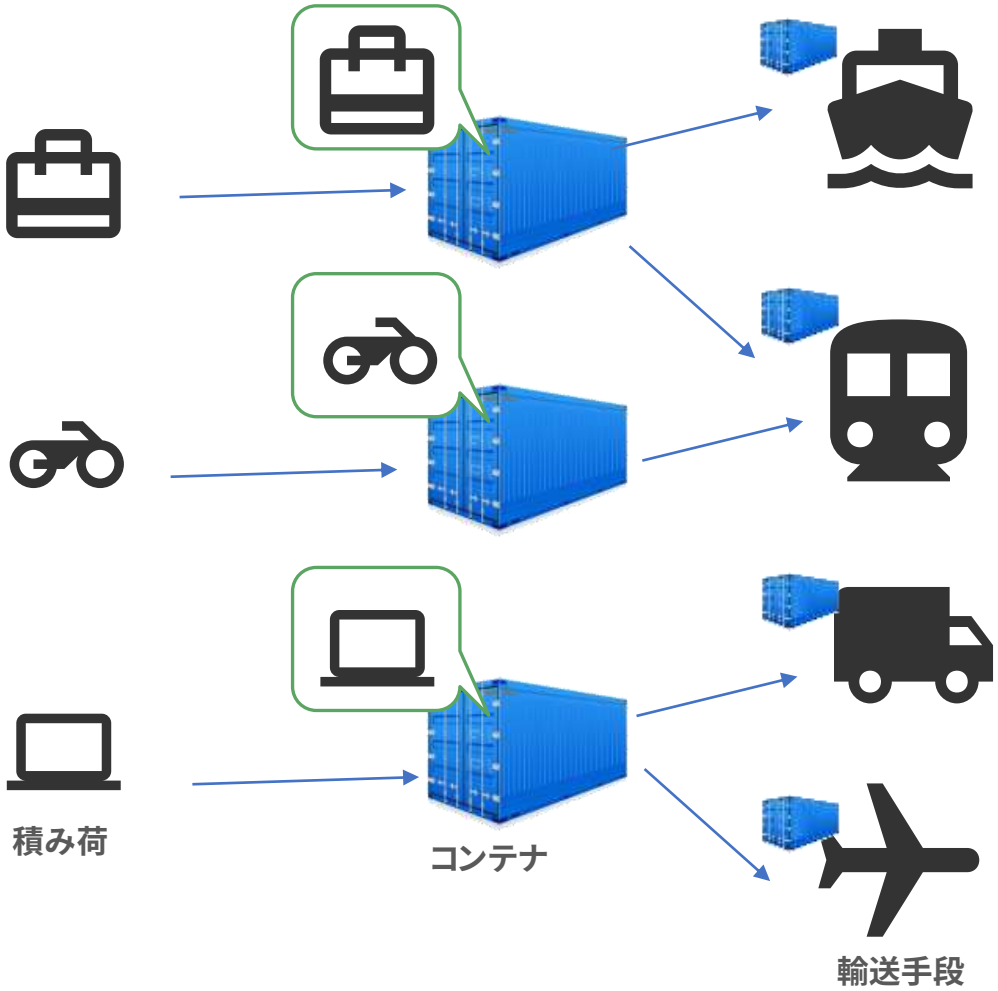
課題 遅い

課題 面倒

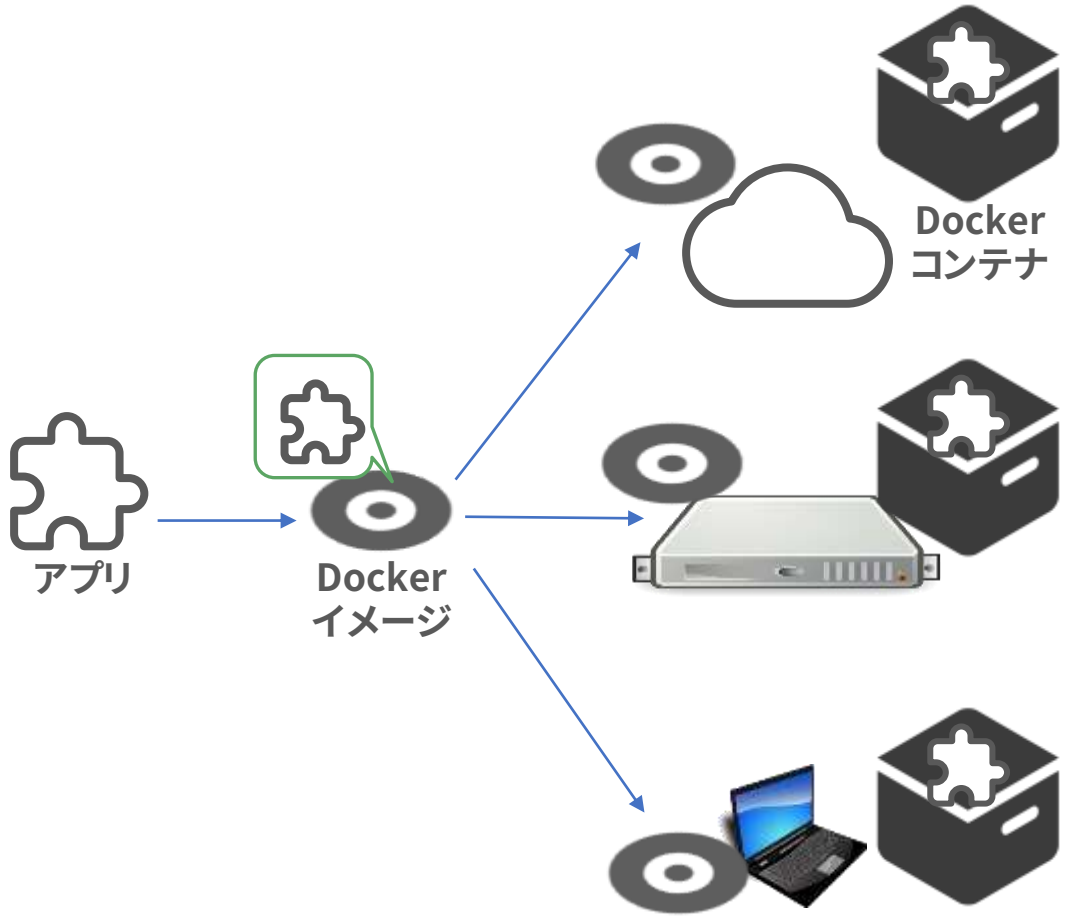
コンテナ

開発 → テスト → 準備 → 本番

どちらも「中身」を気にせず輸送・移動できる



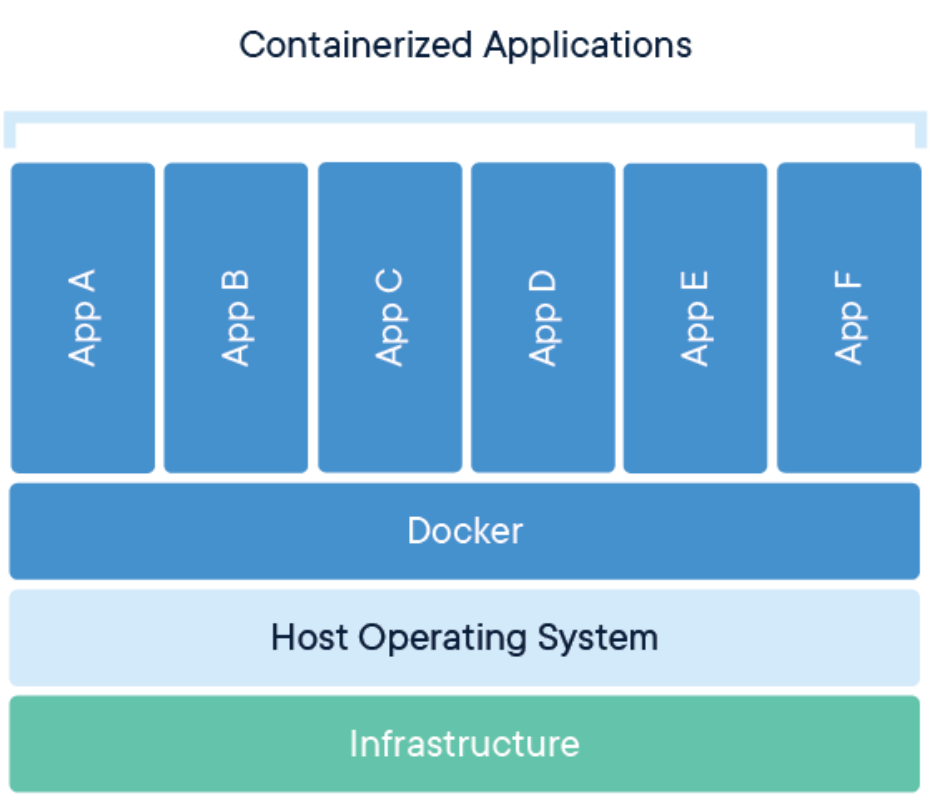
物流コンテナの輸送

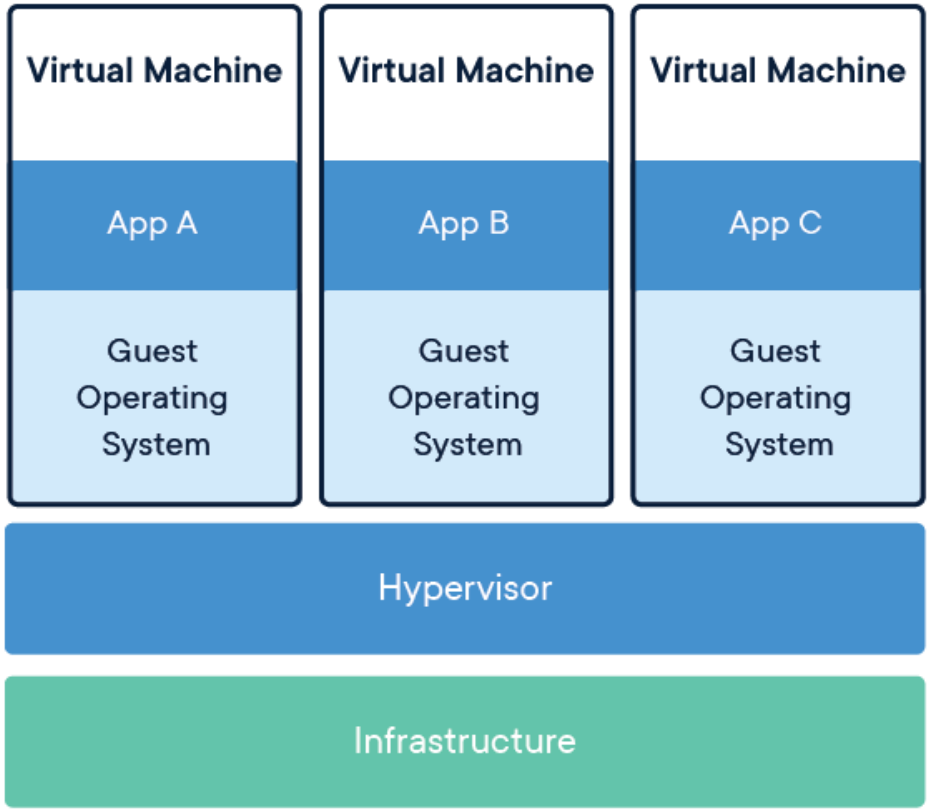


アプリケーションをDockerイメージで移動し
Dockerコンテナとして実行

Docker≠コンテナ型仮想化

- アプリケーション・コンテナ
- システム・コンテナ（コンテナ型仮想化）

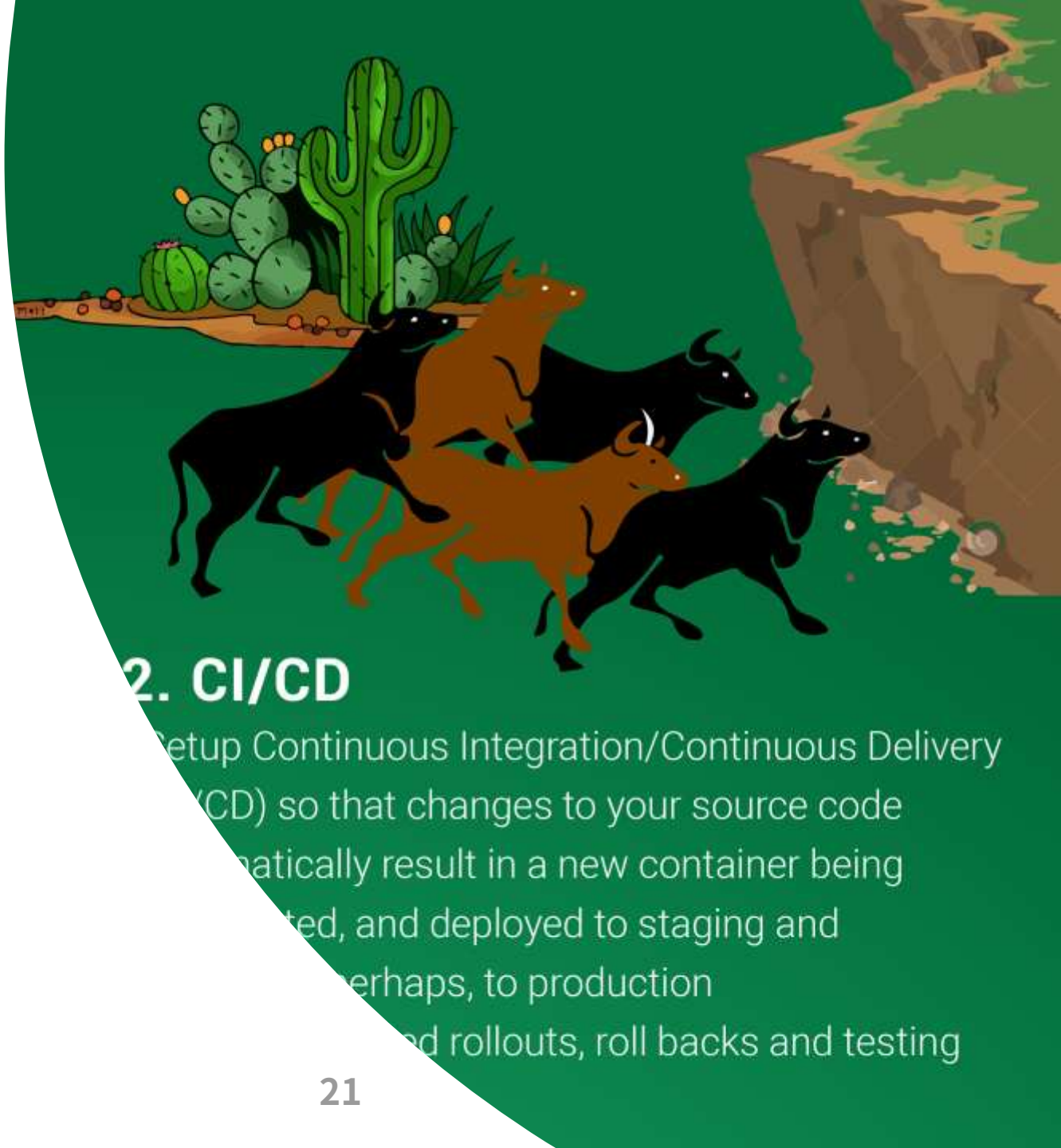




https://www.docker.com/resources/what-container

2. CI/CD

- 継続的インテグレーション/継続的デリバリ(CI/CD)をセットアップすると、ソースコードに対する変更の結果、自動的に新しいコンテナが構築、テストされ、定期的にテスト環境に展開するだけでなく、本番環境にすら展開する。
- ロールアウト、ロールバック、テストを自動的に行うようセットアップする。



高品質で持続的な開発・運用を行うため CI / CD

Continuous Integration / Continuous Delivery

- コンテナ化
  - Docker ... アプリケーションを開発・移動・実行するプラットフォーム
  - Kubernetes ... アプリケーションをデプロイ、スケーリング、管理をする
- バージョン管理
  - Git ... ソースコードの変更履歴を記録・追跡するバージョン管理システム
  - GitHub ... 内部に git を使う、GUI のソフトウェア開発プラットフォーム
- テスト自動化
  - Jenkins ... ソフトウェアのビルド、検証、インストールなどを自動化できるツール

3. オーケストレーション

- Kubernetes は業界トップのオーケストレーション・ソリューションです。
- 認証 Kubernetes ディストリビューション、対応（Hosted）プラットフォーム、インストーラーから選択すべきです。

Cloud Native 参照アーキテクチャ

【参考】https://github.com/cncf/presentations/blob/master/2016-software-circus/what-is-cloud-native/what-is-cloud-native.pdf

アプリケーションやサービスの
オーケストレーション
Orchestration

複数のホスト・システム上を横断するアプリケーションをスケール（拡大・縮小）できる機能
※コンテナに依存しない

設定ファイルをベースにサービスを定義・維持

- **Kubernetes**
- **Docker Engine (swarm mode) + Docker Compose**
- **Rancher**
- **Nomad**

計算資源の抽象化

(主に(コンテナ)やジョブの
スケジューリング
Scheduling

- **Marathon, chronos**
- **Docker swarm**
- **Deis**
- **fleet**

ホスト計算資源の
クラスタ管理
Cluster Management

- **Apache Mesos**
- **DCOS (Mesosphere)**

従来のオーケストレーション

一方通行

【参考】https://www.slideshare.net/Docker/container-orchestration-from-theory-to-practice/7

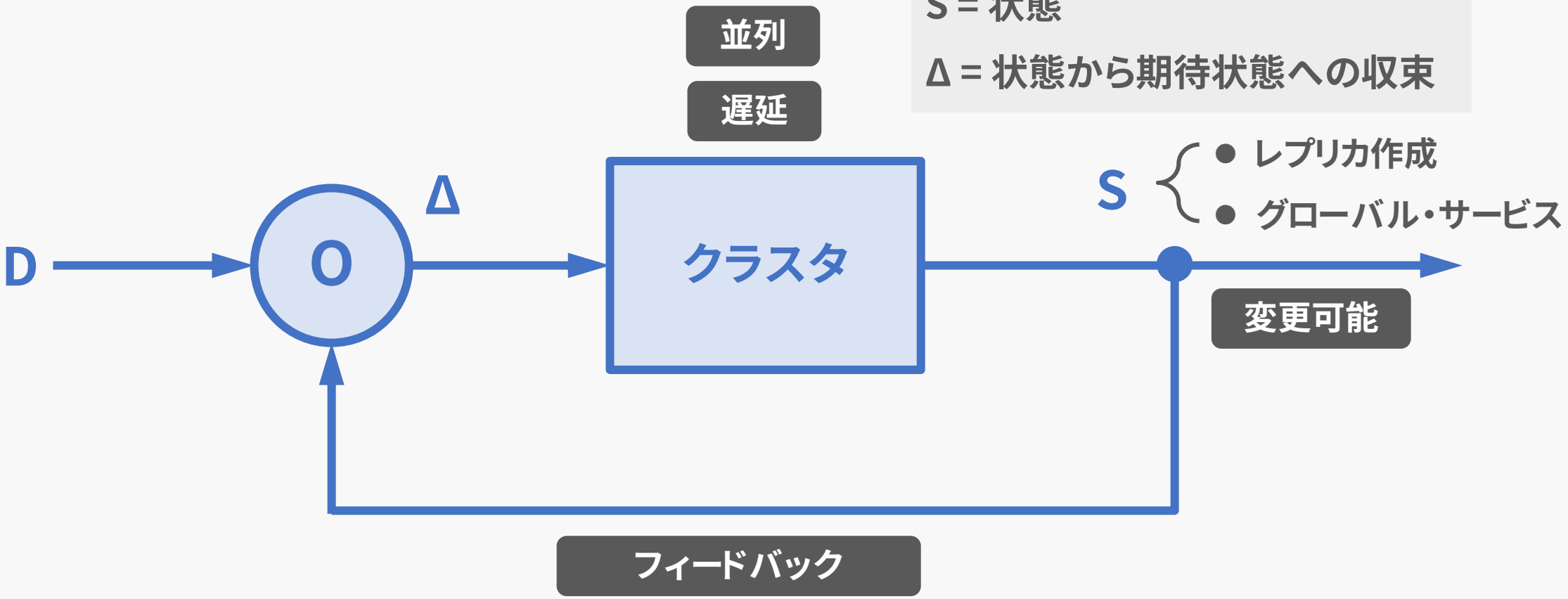
宣言型サービス・モデルのオーケストレーション
Declarative service model

D = 期待状態

O = オーケストレータ

S = 状態

Δ = 状態から期待状態への収束



【参考】https://www.slideshare.net/Docker/container-orchestration-from-theory-to-practice/7

大切なのは目的達成のための順番

システムの柔軟性が高まる一方、
開発・運用の変化を迫られる。

必要があれば、手法だけでなく、
人や組織（文化）も変える必要がある。

ここまでのポイント『手段としてのコンテナ活用』

- クラウドコンピューティングの振り返り

  1つの巨大なコンピュータに集約される予測(「クラウド化する世界」)は外れ、世界各地のデータセンタやスマートフォンに至るまで処理が分散し、生活に密着した。

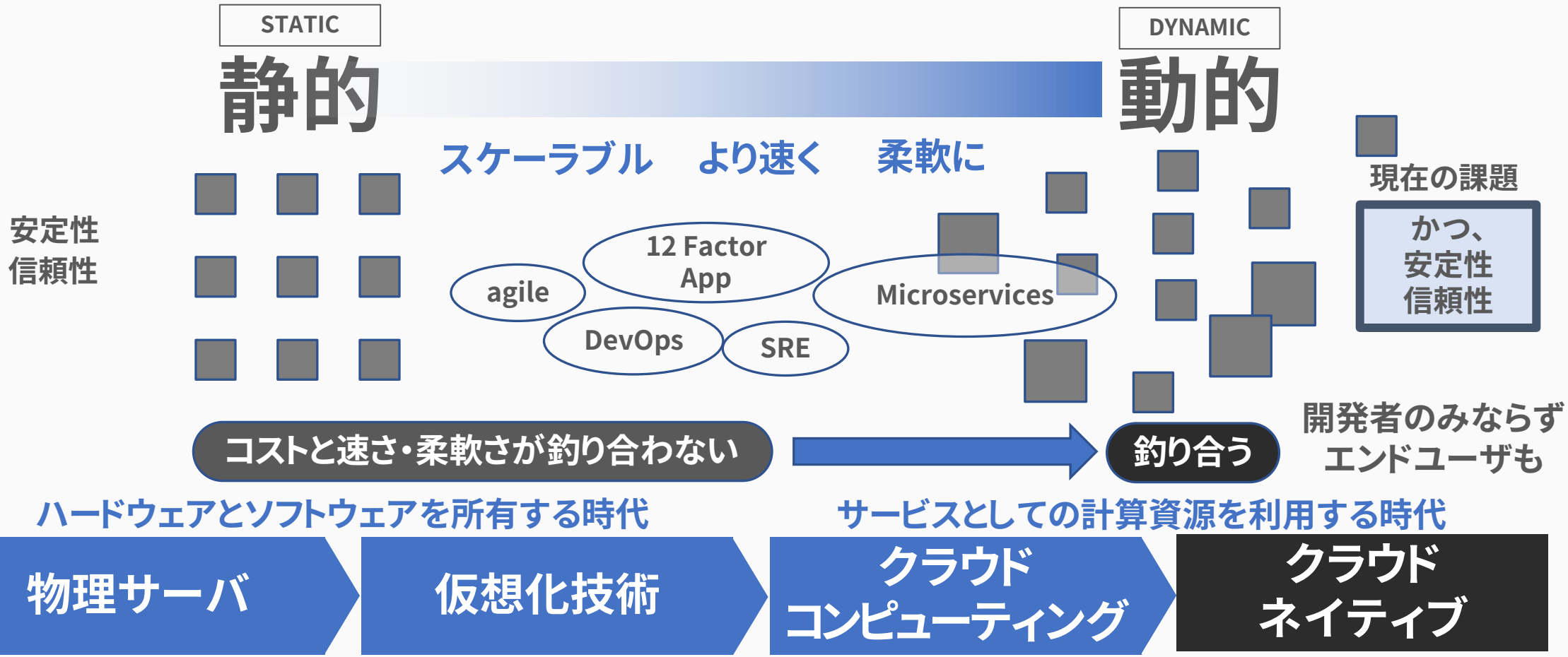
- Dockerコンテナ登場から、Kubernetesオーケストレーションに至る経緯

  クラウドネイティブに至る流れは、物理→仮想化→クラウドの延長線だが、技術は別。活用には、CNCFのCloud Native Trail Map にあるように、システムの「コンテナ化」→「CI/CD」→「オーケストレーション」の順番が大事。

- 「コンテナ」とは何か？　システムを作る場合の考慮点

  Linuxカーネルの名前空間・cgroupの分離技術を使って、素早い開発・運用を可能に。仮想化ではない。そのまま移行は無理。スケール可能なシステムへの対応が必要。近年は、技術要素よりも、開発文化がネックになりつつある。

システム基盤も、サービス側も変わり続ける前提



ちなみに：Cloud Computing / COVID-19

- 今すぐシステムが必要な場面
- 大規模アクセスが予想されるため、アプリケーションはスケール前提

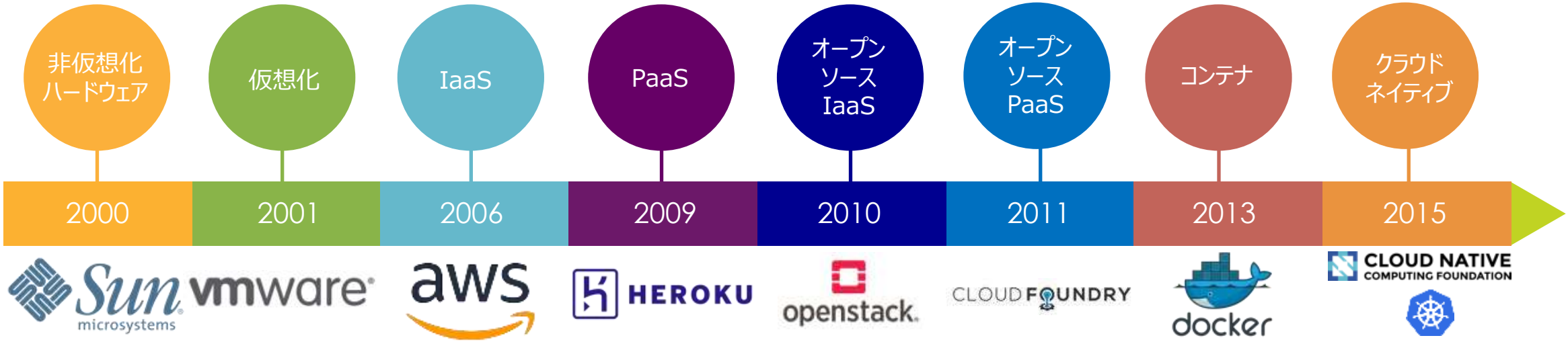
コンテナ / Docker

仮想化からクラウド・ネイティブへ
From Virtualization to Cloud Native

CLOUD NATIVE COMPUTING FOUNDATION

kubernetes

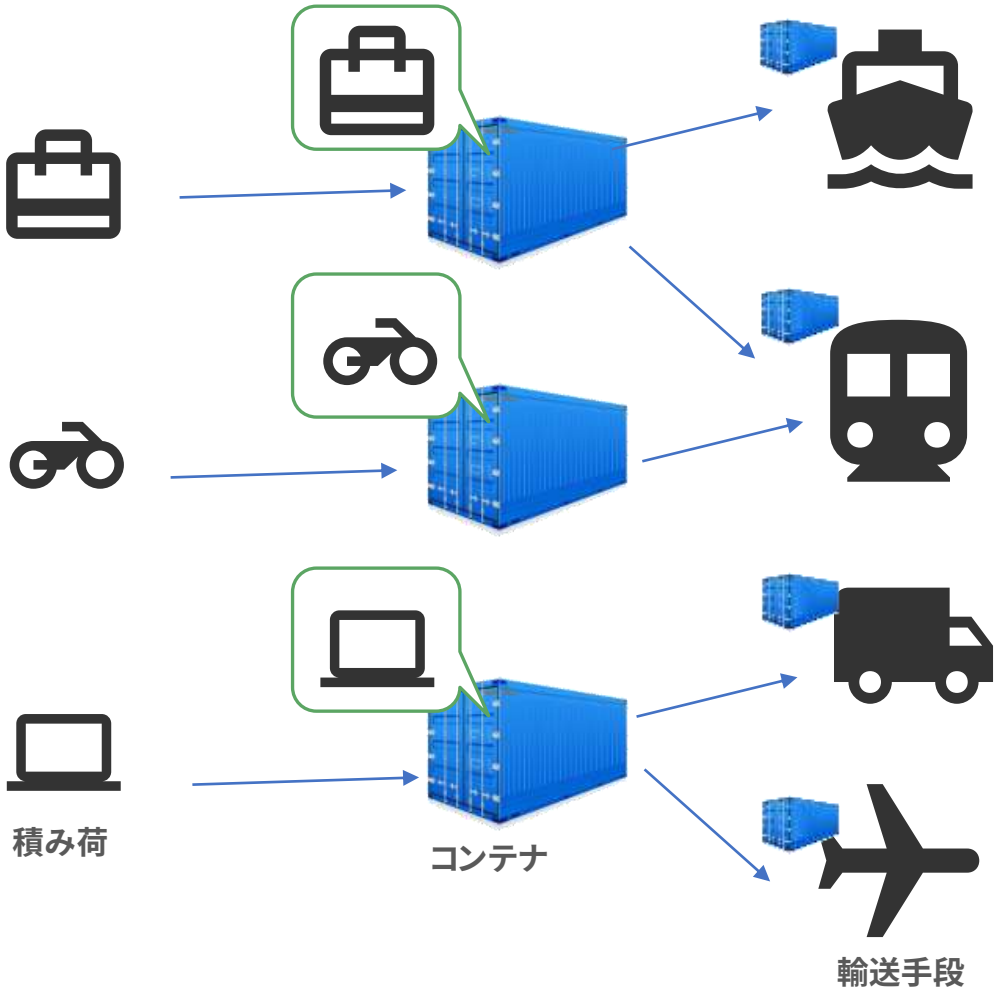
- クラウド・ネイティブ・コンピューティングはオープンソースのソフトウェアを積み重ね、次のために用います：
  - アプリケーションをマイクロサービス(*microservices*)に分割し、
  - 各パーツ自身をコンテナにパッケージし、
  - リソース利用を最適化するために、動的に統合/オーケストレート(*orchestrate*)する



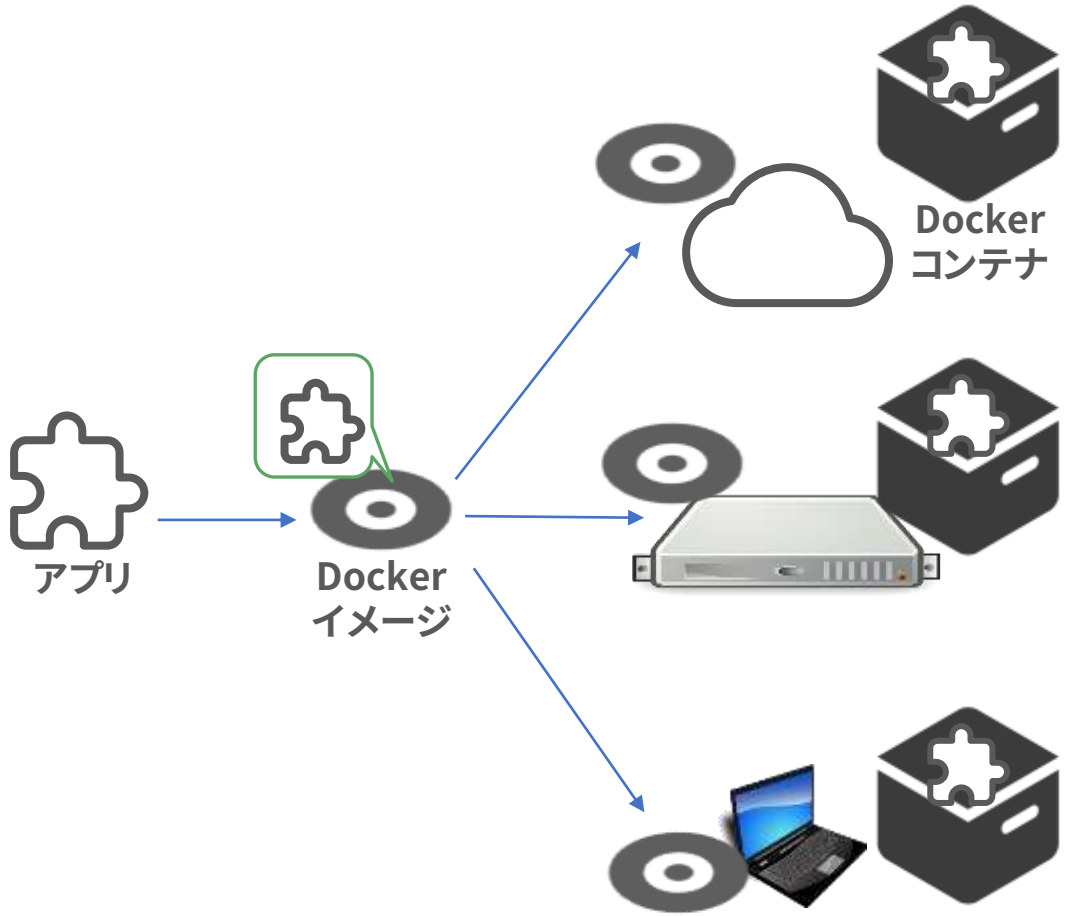
※ https://github.com/cncf/presentations からの引用

Build, Ship(Share), Run

The future of Linux Containers - YouTube
https://www.youtube.com/watch?v=wW9CAH9nSLs



物流コンテナの輸送



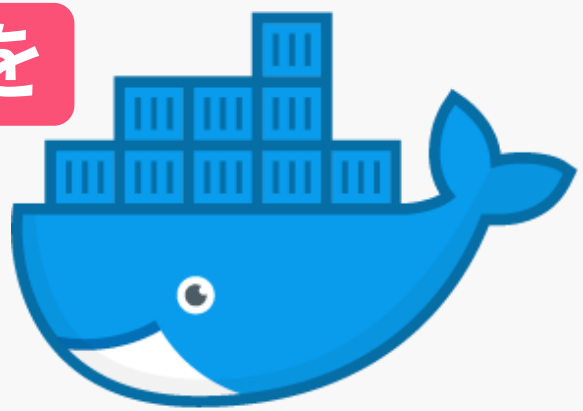
アプリケーションをDockerイメージで移動し
Dockerコンテナとして実行

Docker

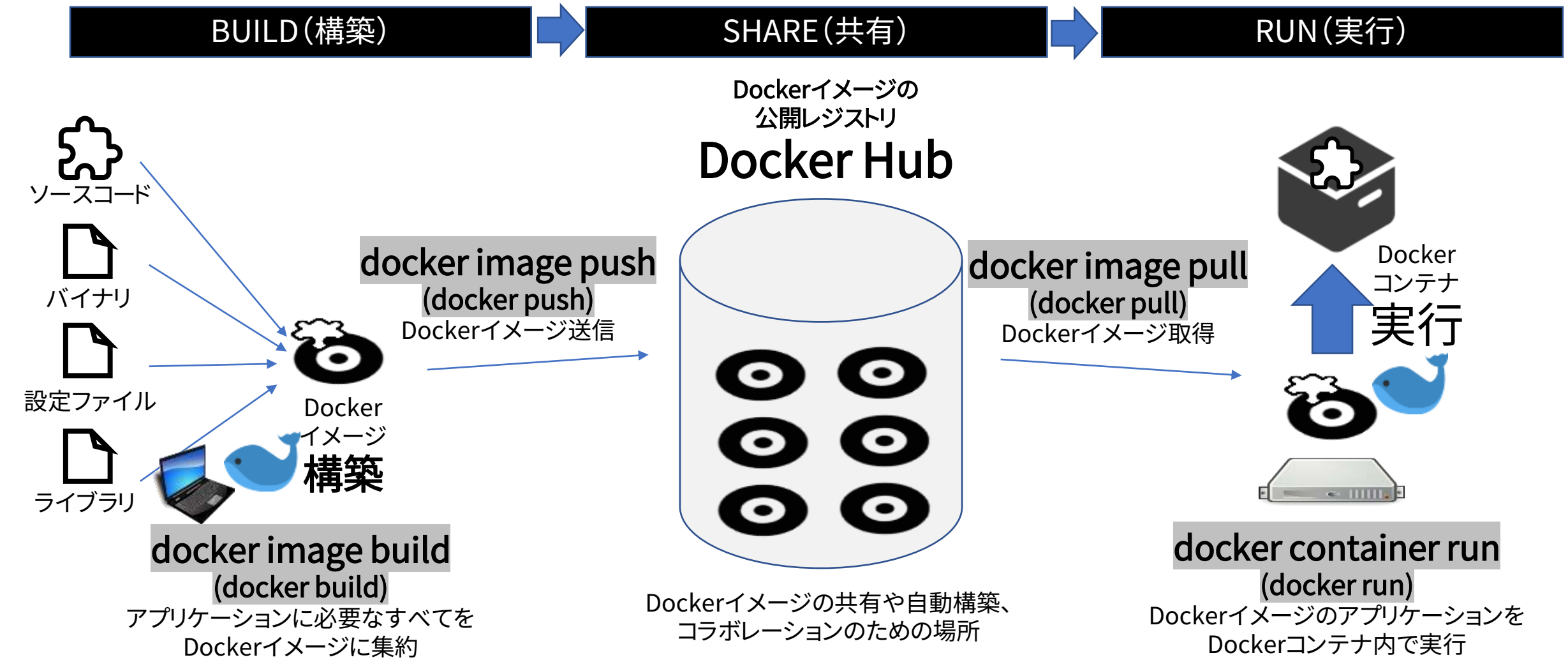
Dockerイメージとして

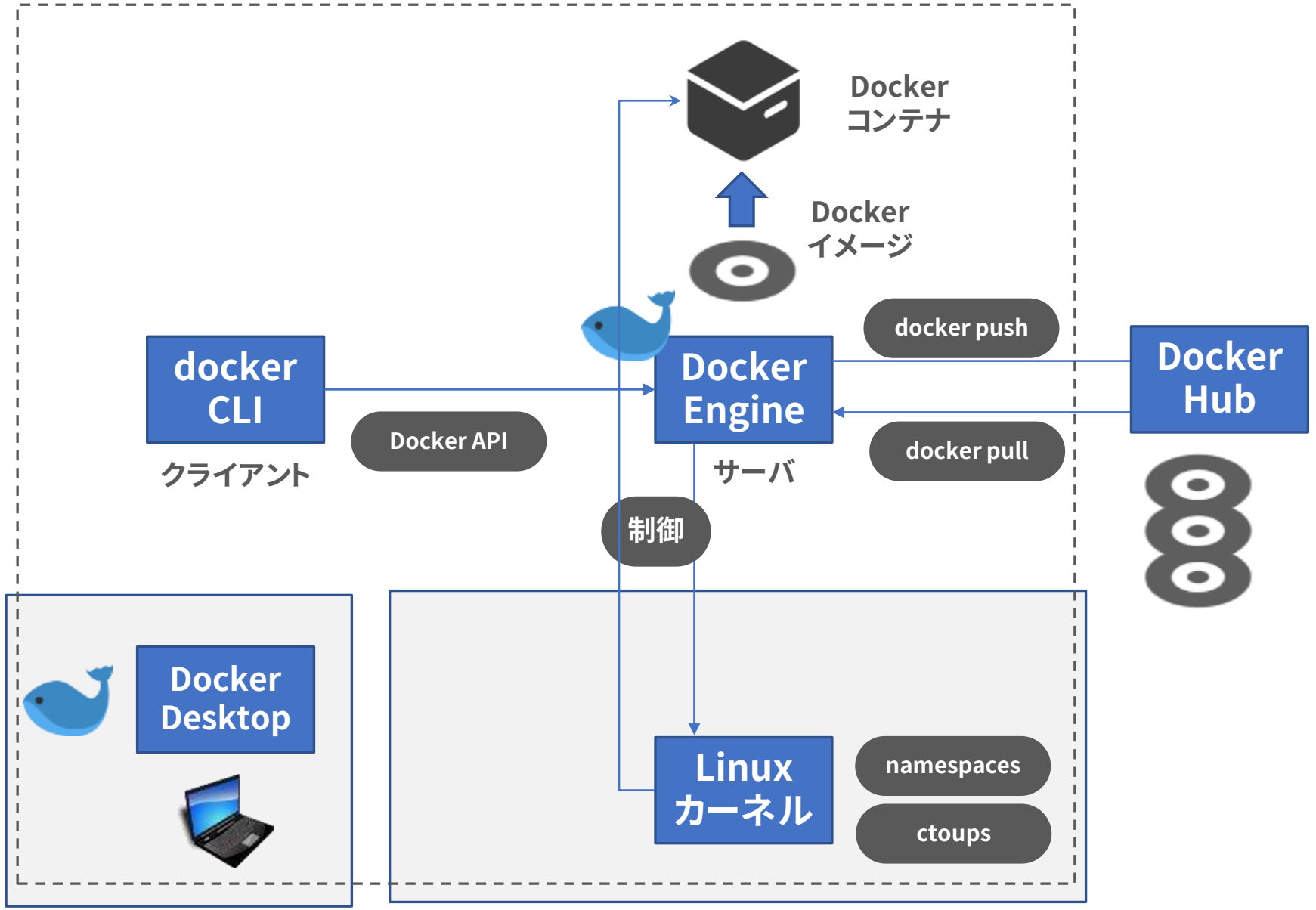
全ての依存関係をパッケージ化して、コンテナとして動かす

Linuxファイルシステムを

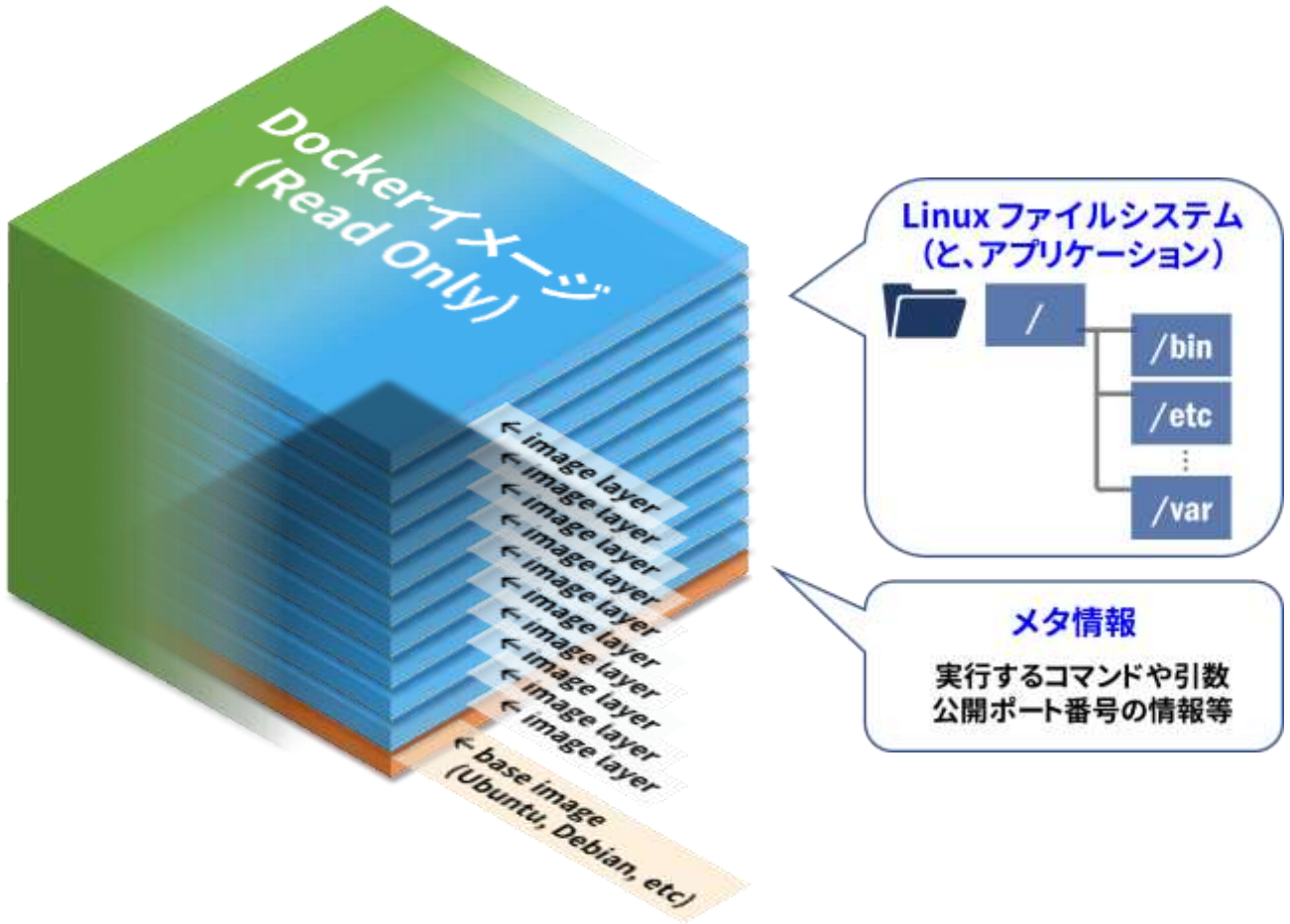


“Docker allows you to package an application with all of its dependencies into a standardized unit for software development.”


BUILD（構築）
SHARE（共有）
RUN（実行）
Dockerイメージの
公開レジストリ
Docker Hub
ソースコード
バイナリ
設定ファイル
ライブラリ
Docker
イメージ
構築
docker image build
(docker build)
アプリケーションに必要なすべてを
Dockerイメージに集約
docker image push
(docker push)
Dockerイメージ送信
docker image pull
(docker pull)
Dockerイメージ取得
Dockerイメージの共有や自動構築、
コラボレーションのための場所
Docker
コンテナ
実行
docker container run
(docker run)
Dockerイメージのアプリケーションを
Dockerコンテナ内で実行


Docker
コンテナ
Docker
イメージ
docker
CLI
クライアント
Docker API
Docker
Engine
サーバ
docker push
docker pull
Docker
Hub
制御
Docker
Desktop
Linux
カーネル
namespaces
ctoups

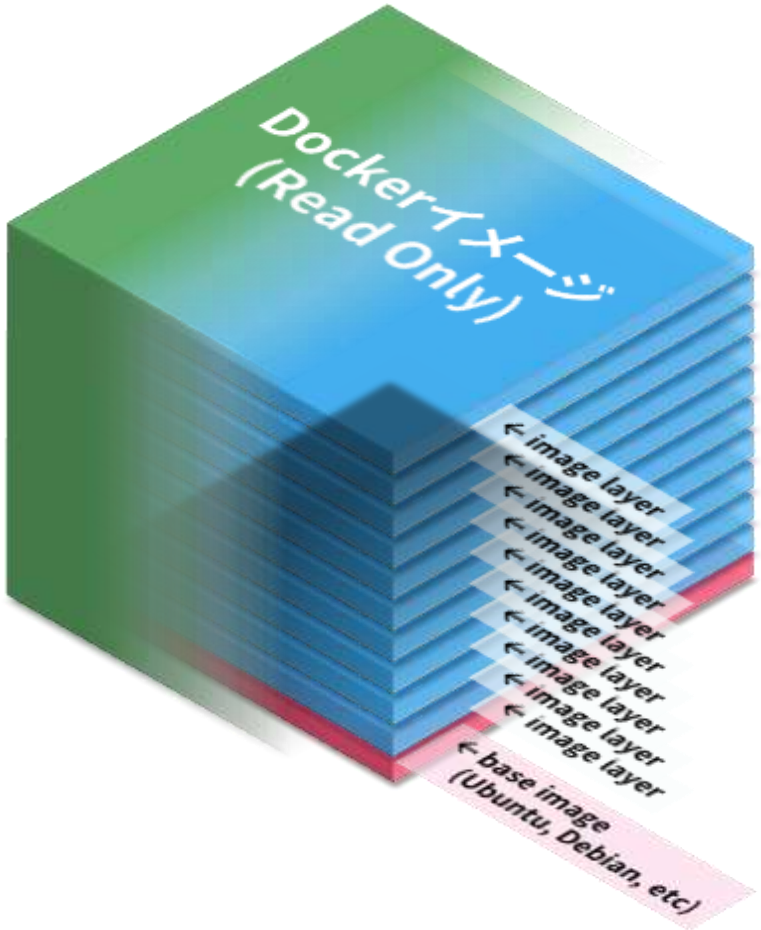
ではなく



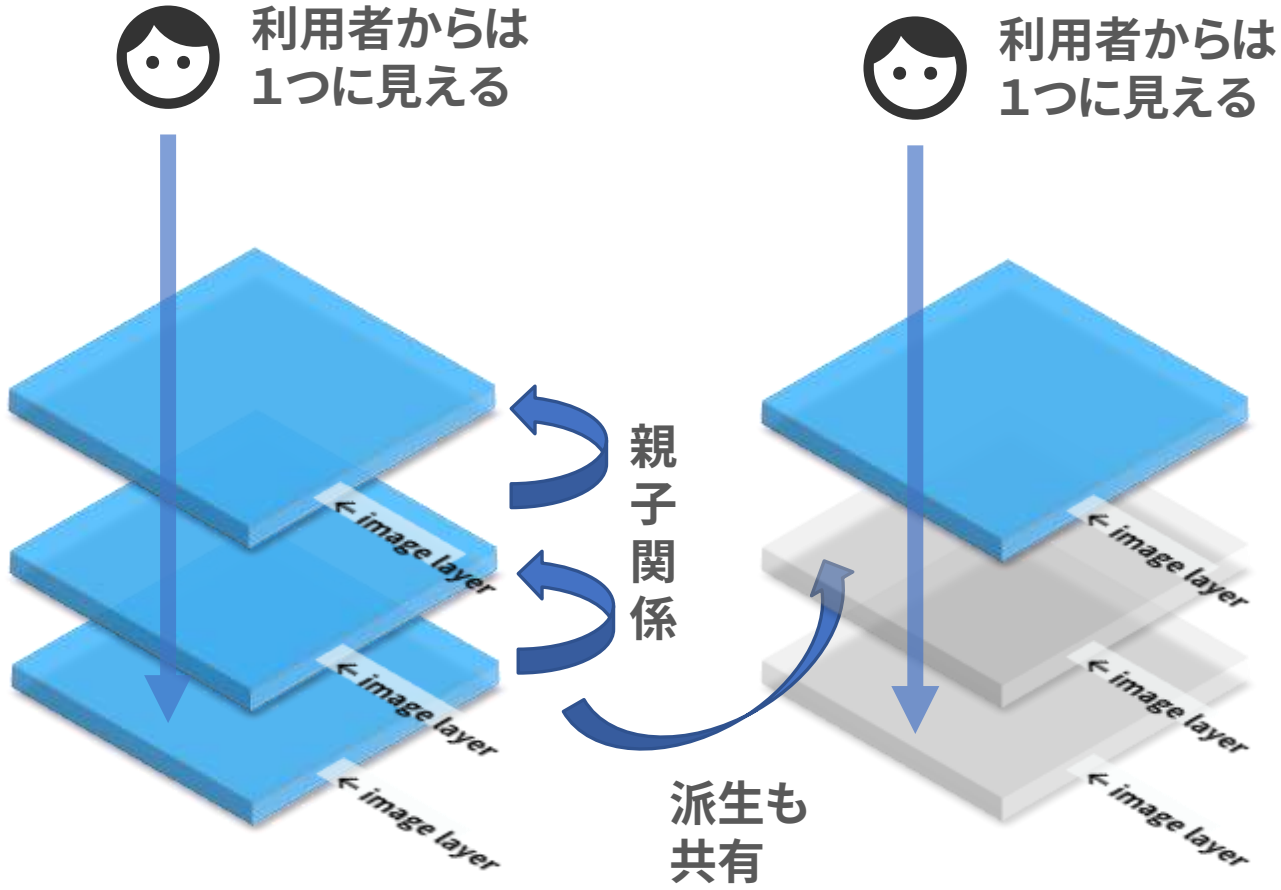
✕ docker～.img

Dockerイメージは「1つのファイル」ではなく「概念」。抽象的なイメージ・レイヤの積み重ね

Dockerイメージはイメージ・レイヤの積み重ね

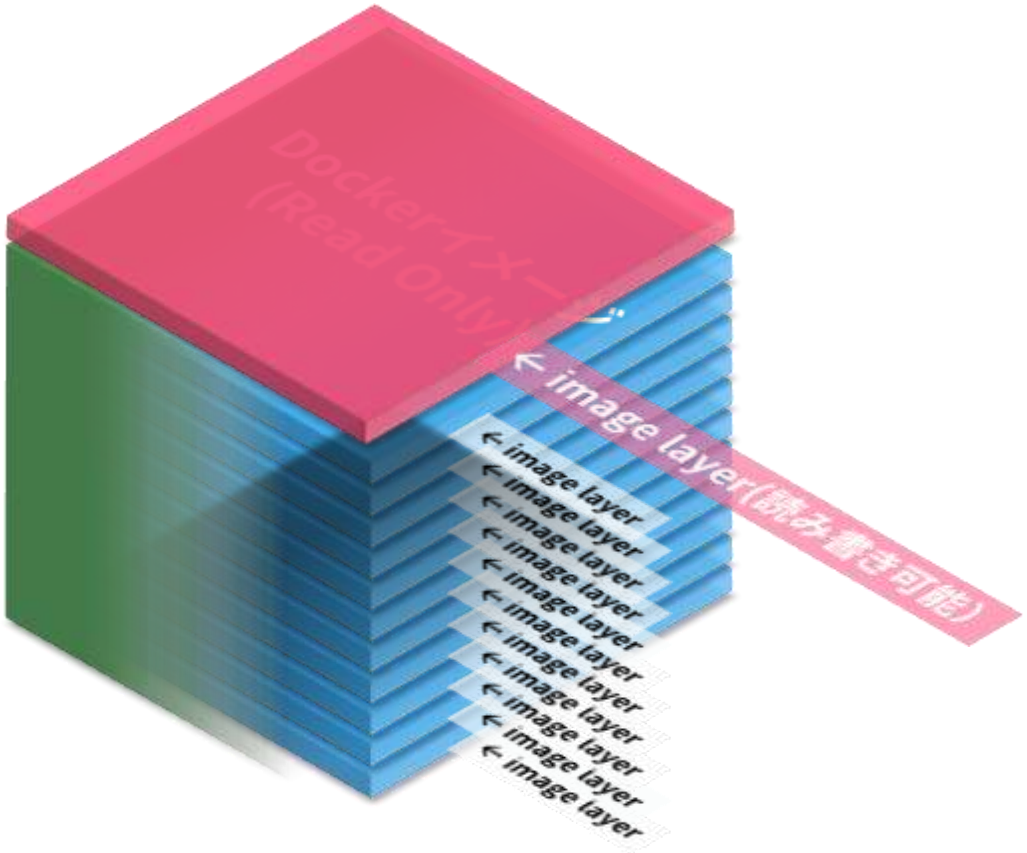


プログラムやライブラリと
メタ情報（実行するプログラムやポートなど）

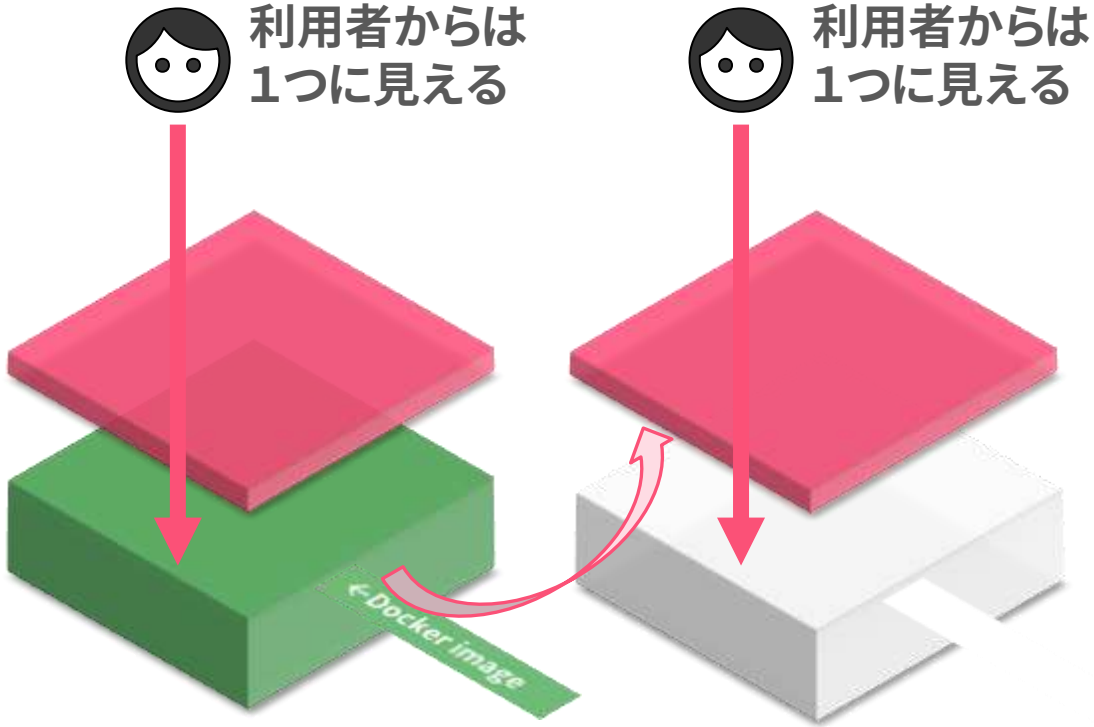


だから高速に移動できる・開発を高速化できる
リソースを有効に使える “lightweight” な性質

Dockerコンテナもイメージ・レイヤを持つ

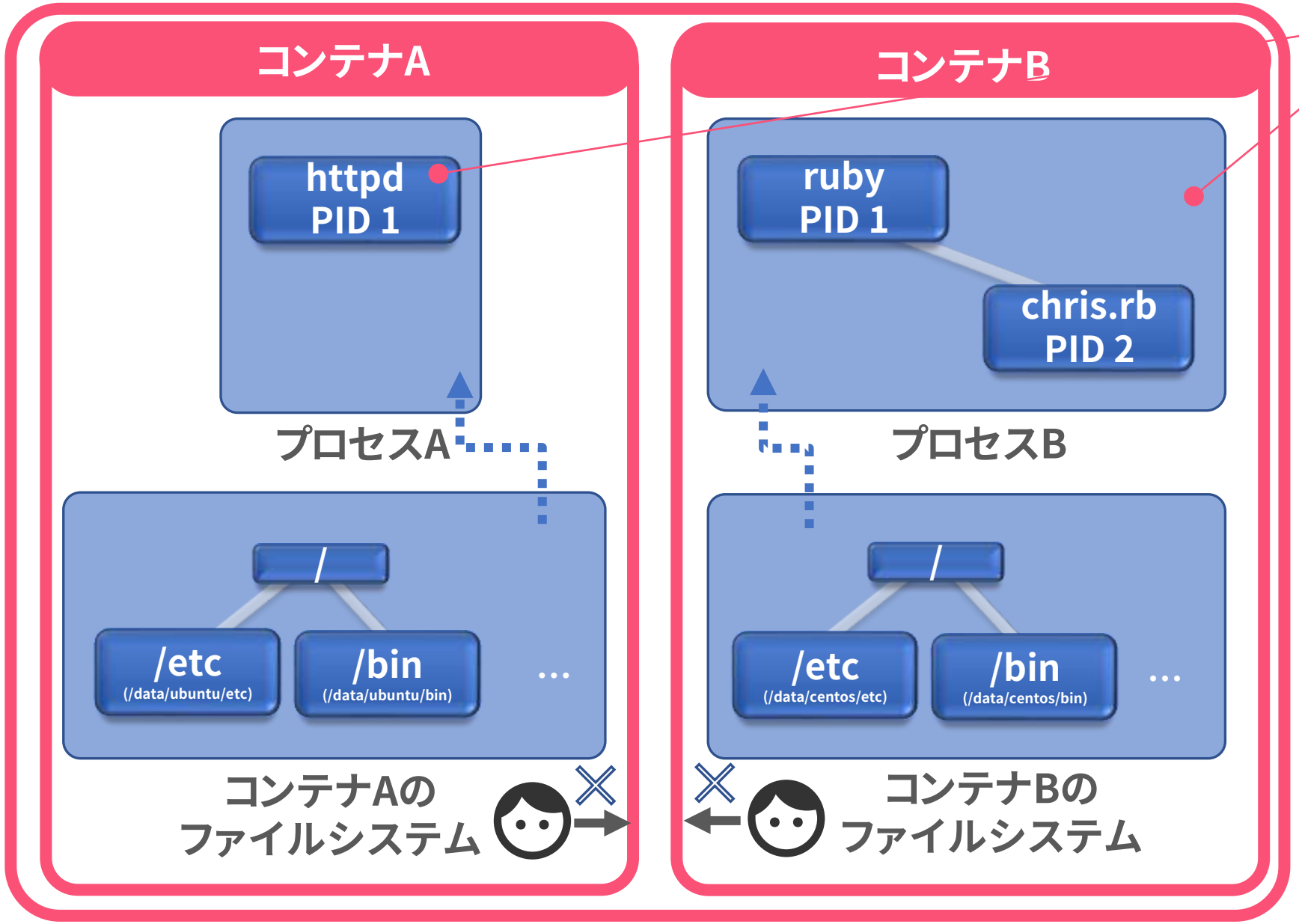


元のレイヤに対する変更情報を記録
Copy on Write の性質



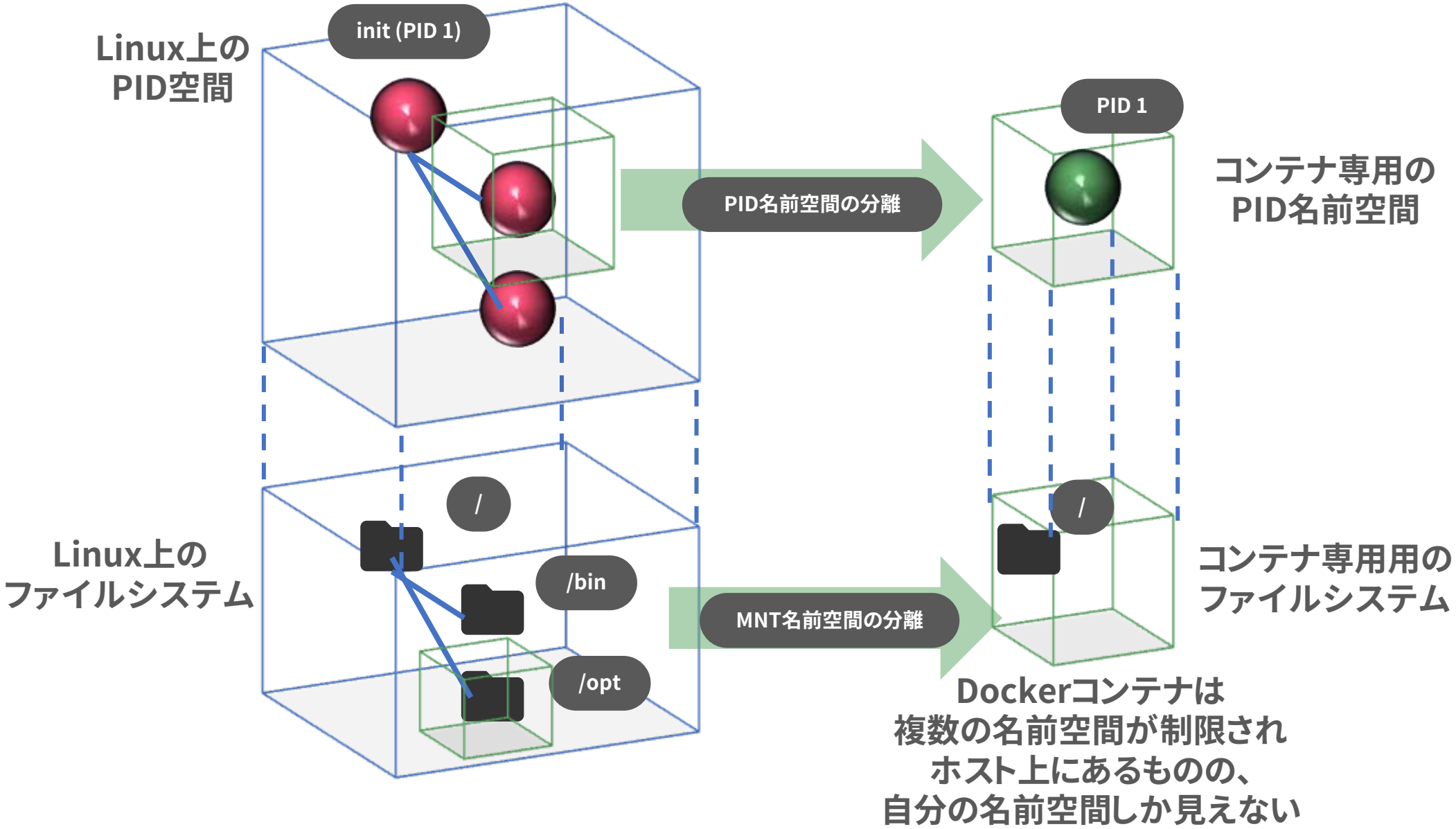
だから高速に移動できる・開発を高速化できる
リソースを有効に使える “lightweight” な性質

コンテナは特別なプロセスの状態

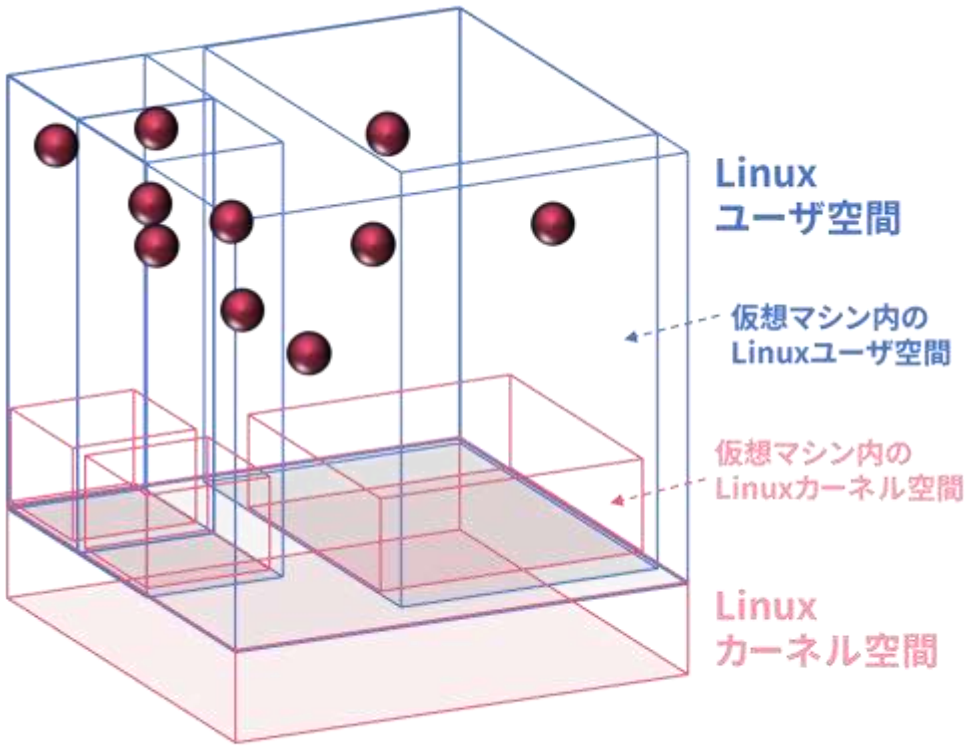


名前空間の isolate
- プロセス
- ファイルシステム
- ネットワーク
- ホスト名
- UID・GID
- プロセス間通信、等

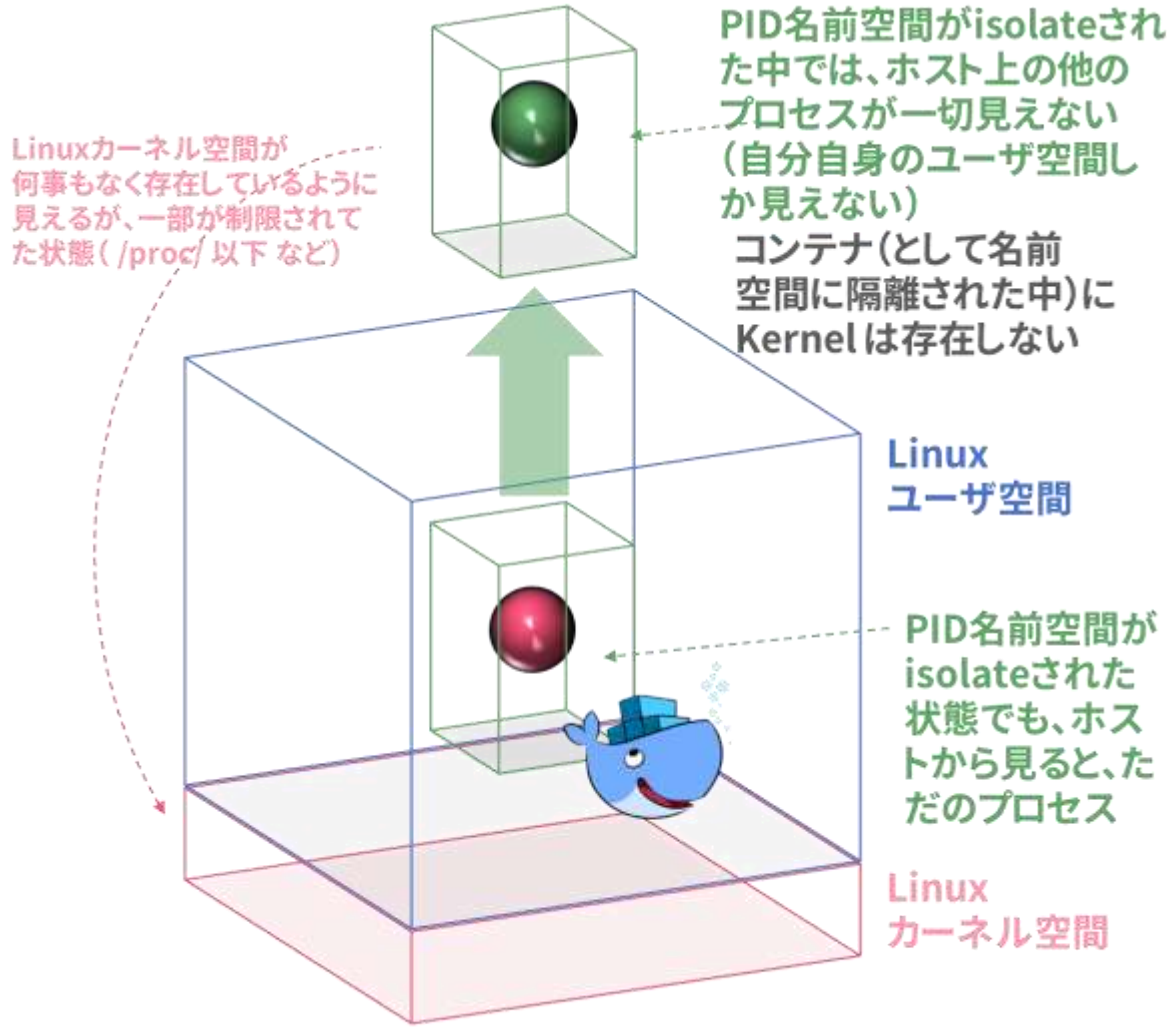
cgroupでリソース制限
- CPU
- メモリ
- I/O
- ディスク・クォータ、等



Dockerコンテナは「特別な状態」のプロセス



ハードウェアの仮想化とプロセス



Dockerコンテナとプロセス

CentOS 8 の公式イメージを使った Dockerfile 例

NG例

Dockerfile

```
FROM centos:8
RUN dnf -y install httpd php
ADD https://raw.githubusercontent.com/docker-
library/httpd/077141ee37fca63972292c562ec0f632d0f831/2.4/httpd-foreground
/usr/local/bin/
RUN chmod 700 /usr/local/bin/httpd-foreground
RUN sed -i -e 's%^#LoadModule mpm_prefork_module
modules/mod_mpm_prefork.so%LoadModule mpm_prefork_module
modules/mod_mpm_prefork.so%' /etc/httpd/conf.modules.d/00-mpm.conf
RUN sed -i -e 's%^LoadModule mpm_event_module
modules/mod_mpm_event.so%#LoadModule mpm_event_module
modules/mod_mpm_event.so%' /etc/httpd/conf.modules.d/00-mpm.conf
EXPOSE 80
COPY ./html/* /var/www/html
CMD ["httpd-foreground"]
```

PHP7 の公式イメージを使った Dockerfile 例

Dockerfile

```
FROM php:7.2-apache
COPY ./html/* /var/www/html
```

CentOS 6 の公式イメージを使った Dockerfile 例

NG例

Dockerfile

```
FROM centos:centos6.10
RUN sed -i
'/^mirrorlist/s/^/#/;/^#baseurl/{s/#//;s/mirror.centos.org¥/centos¥/$releas
ever/vault.centos.org¥/6.10/}' /etc/yum.repos.d/*B* && ¥
    yum install -y epel-release  && ¥
    yum update -y && ¥
    cp -f  /usr/share/zoneinfo/Japan /etc/localtime
RUN yum -y install php httpd
EXPOSE 80
RUN echo '<?php phpinfo(); ?>' > /var/www/html/index.php
ADD https://raw.githubusercontent.com/docker-
library/httpd/077141ee37fca63972292c562ec0f632d0f831b1/2.4/httpd-foreground
/usr/local/bin/
RUN chmod 700 /usr/local/bin/httpd-foreground
CMD ["httpd-foreground"]
```

PHP5 の公式イメージを使った Dockerfile 例

Dockerfile

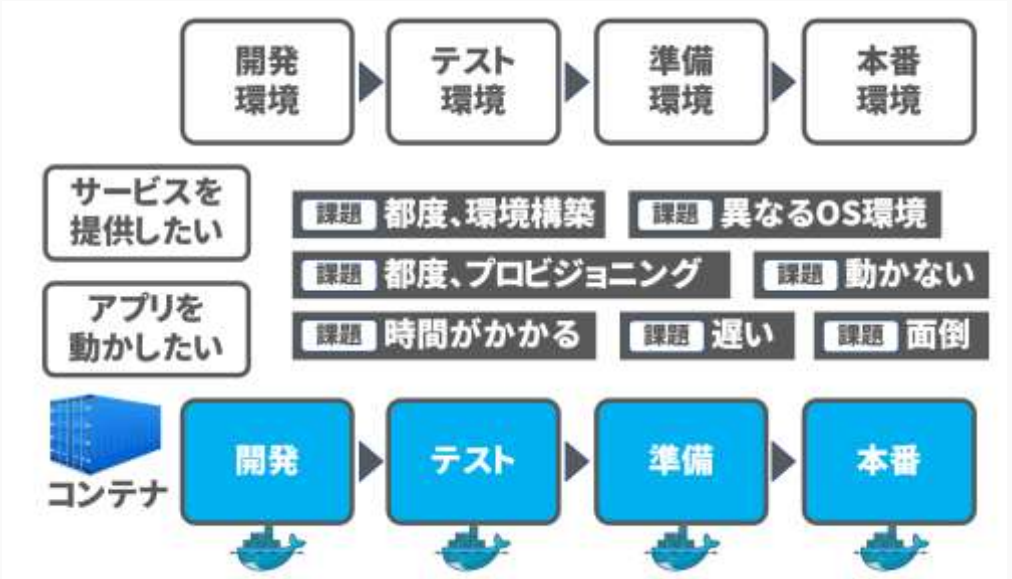
```
FROM php:5.6-apache
COPY ./html/* /var/www/html
```

Docker 化にあたっての注意点

- 【最重要】Docker は「仮想化技術」ではありません
  - 「アプリケーション・コンテナ」と「システム・コンテナ」は考え方が違う
  - 稼働中の環境を、丸っと移行は無理
    - 例：initがない
    - 例：セキュリティ対策や日々の運用をどうするか
- 既存環境をコンテナ移行ではなく、環境再構築検証の用途に
  - 従来の手順書で再現ができるかどうか
  - なければこの機会に手順化
- CloudNative 的な技術習得やチャレンジの機会に
  - いきなり未知のものをコンテナ化して取り組むより、
    よく知っているもののほうが移行しやすい
  - ただし、コンテナ活用の用途は「アプリケーション・コンテナ」に限るべきであり、
    無理矢理すべてを「そのまま」コンテナ化すべきという話ではない

“Dockerは銀の弾丸ではない”

- 前提として、Dockerは（いわゆる）「仮想化技術」とは異なる思想・技術背景
  - 「コンテナ型仮想化」として表現・比較されるが [要出典]
  - セキュリティ要件によっては、コンテナの利用より仮想化技術を選択すべき
- （特性を理解した上で）有用な場面もある
  - 技術（再）検証や手順の（再）作成
  - デプロイを素早く行いたい
  - バッチ的な処理
- どうしても使わざるを得ない場合は
  - Docker の基本概念を理解した上で利用する
  - 丸ごと移植ではなく、移行のための一時措置
  - 最新の公式ドキュメントが常に正しい



アプリケーションの“Build, Share, Run” を素早く簡単に

SAKURA internet

Lab Products

Hacobune (β)

さくらインターネット株式会社

SAKURA internet

レンタルサーバ

さくらのレンタルサーバ
さくらのマネージドサーバ

1台のサーバを複数の契約者でサーバを共有または占有することができ、管理はさくらインターネットに任せて使うサービス

1台を共有

1台を占有

VPS

さくらのVPS

仮想化技術を用い、1台の物理サーバ上に複数の仮想サーバを構築し、仮想専用サーバとして分けた領域の占有サービス

クラウド

さくらのクラウド
SAKURA CLOUD

高性能サーバと拡張性の高いネットワークを圧倒的なコストパフォーマンスで利用できるIaaS型パブリック・クラウド・サービス

専用サーバ

さくらの専用サーバ

高性能で拡張性と信頼性の高いサーバをまるごと独占して利用することができ、自由にカスタマイズして利用可能なサービス

1台～複数台

データセンター

ハウジング
リモートハウジング

データセンター内にお客様専用のハウジングスペースを確保し、ネットワーク機器やサーバなどの機材を自由に置けるサービス

新サービス

Tellus

日本発のオープン＆フリーなデータプラットフォームです。衛星から取得出来る情報を含め、世界中のあらゆるデータを集積

www.tellusxdp.com/

さくらのIoT Platformβ

通信環境とデータの保存や処理システムを一体型で提供するIoTプラットフォーム・サービス

https://iot.sakura.ad.jp/

高火力
KOUKARYOKU

高火力コンピューティング

https://www.sakura.ad.jp/koukaryoku/

サービスの主な利用用途

ウェブサイト運営、ブログ、インターネット・メール

ネットビジネス、電子商取引、動画・音楽配信、開発環境

エンタープライズ

会員制サイト、キャンペーン・サイト

SNS、ウェブ・アプリケーション、SaaS、ASP

https://www.sakura.ad.jp

さくらインターネット株式会社

データセンター・専用サーバ

東京ドーム約1個分の敷地面積（51,448㎡）
札幌から車で20～30分とアクセスも容易

さくらのクラウドのコンセプト

開発者志向のシンプルなクラウド

IaaS クラウド（サービスとしてのインフラ）基盤を、圧倒的なコストパフォーマンスで提供

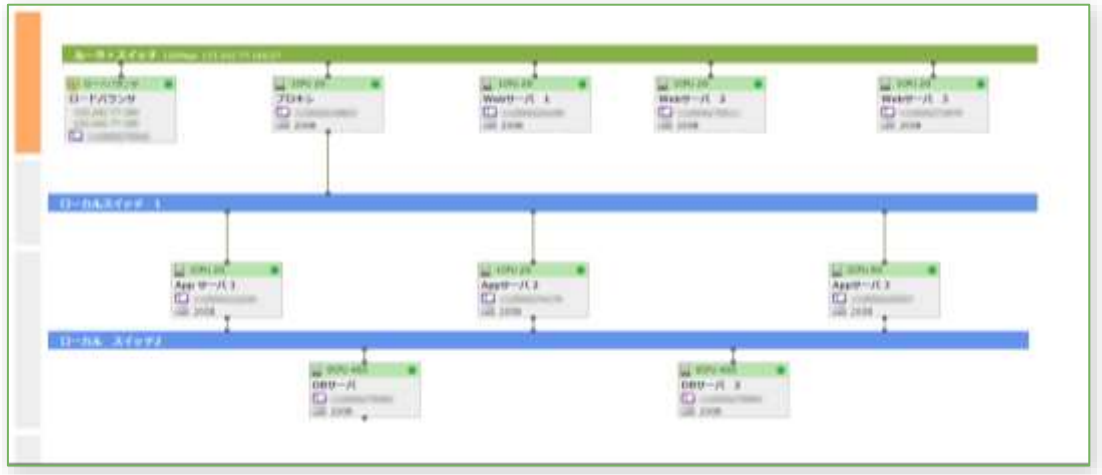
高い自由度

高性能サーバーをスケールアウト

ネットワークも物理環境のように自由に設計・実装できる

ブラウザで仮想データセンターを操作

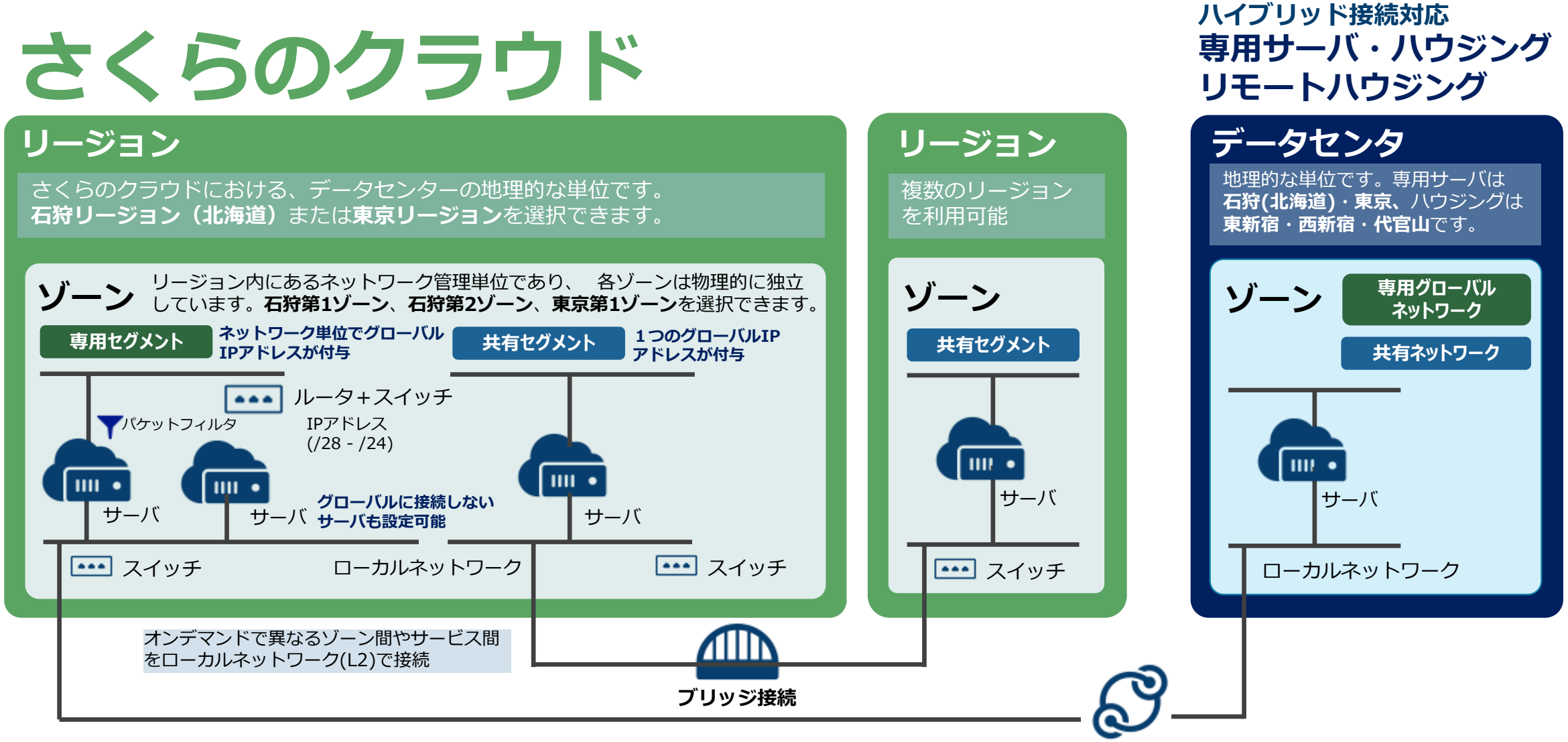
「サーバ」という概念を大切にしたい



https://cloud.sakura.ad.jp/

基本用語と解説

さくらのクラウド



SAKURA internet

企業情報 ニュース IR情報 CSR活動 採用情報

コーポレートサイトTOP > すべてのニュース > お知らせ > インフラを意識せずにSaaS開発

お知らせ Tweet B! 225

インフラを意識せずにSaaS開発ができる 次世代PaaS「Hacobune」のオープンβ版を2021年8月12日に無料提供開始 ～8月27日にオンライン発表会を開催～

2021年8月12日

さくらインターネット株式会社は、PaaS型クラウドサービス「Hacobune（はこぶね）」のβ版を、2021年8月12日よりLabプロダクト※1として無料で提供開始します。
また8月27日にこのたびの提供開始を記念して、オンライン発表会を開催いたします。

Hacobuneは、当社が「インフラを意識しない世界を実現する」をビジョンに開発したPaaS型クラウドサービスです。スタートアップ企業や少人数でのサービスの開発を行うお客さまなど、スモールスタートでの開発に適しています。Hacobuneを利用することで、インフラの構築が不要となり、お客さまはアプリケーションの開発およびアップデートのみに専念することができ、サービスリリースのサイクルを加速させることが可能となります。

インフラを意識せずにSaaS開発ができる 次世代PaaS「Hacobune」のオープンβ版を2021年8月12日に無料提供開始 ～8月27日にオンライン発表会を開催～ | さくらインターネット
https://www.sakura.ad.jp/information/announcements/2021/08/12/1968207782/

Hacobune (β)

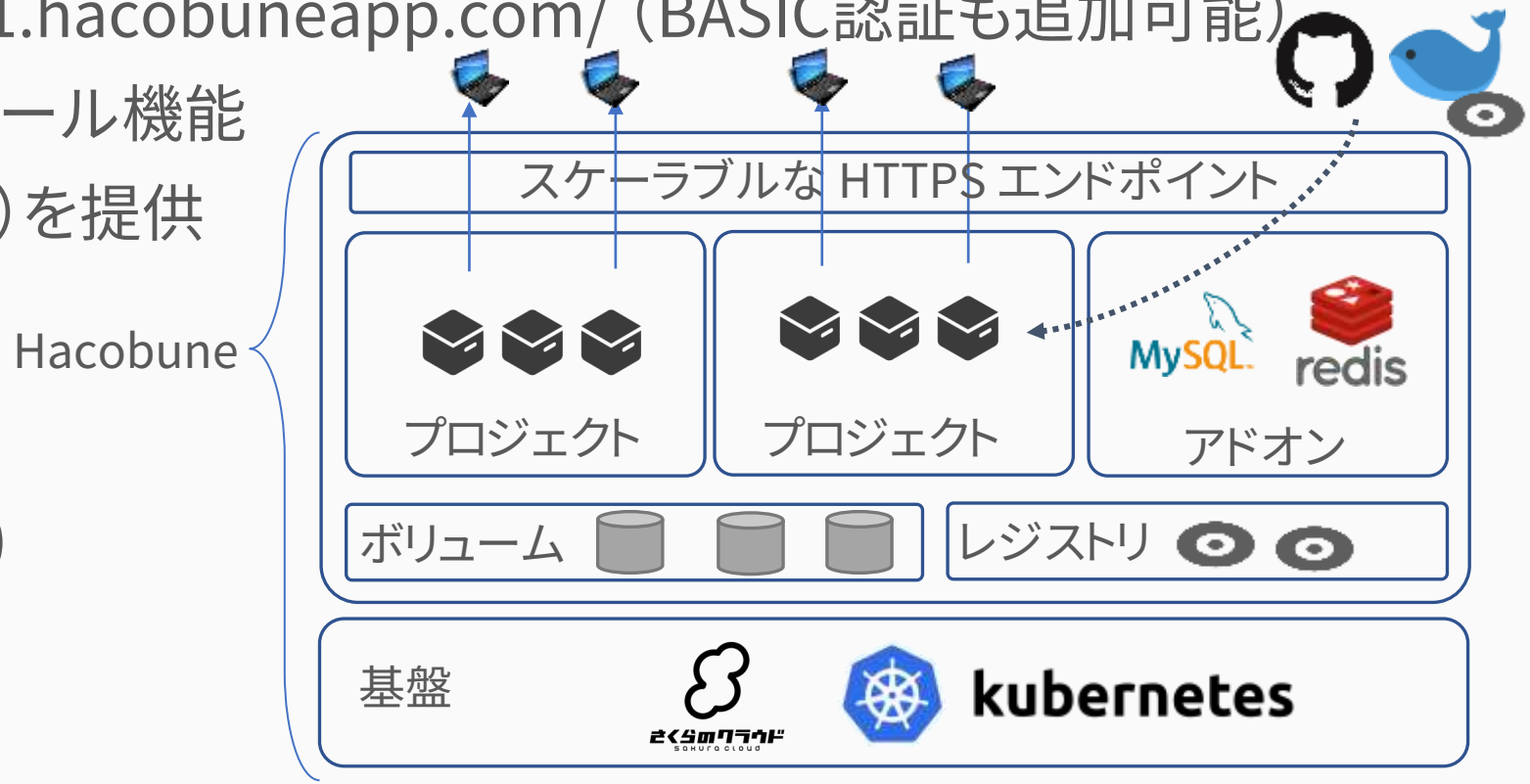
- 『小さなチームで始めるサービス開発のための Web アプリケーションプラットフォーム』
  - 高速にサービス開発からリリースまでのサイクルをまわす仕組みを提供
  - インフラ構築や運用から開発者を解放

課題例:
・ クラウド上にスケール可能なアプリケーションを、高速に開発するには?

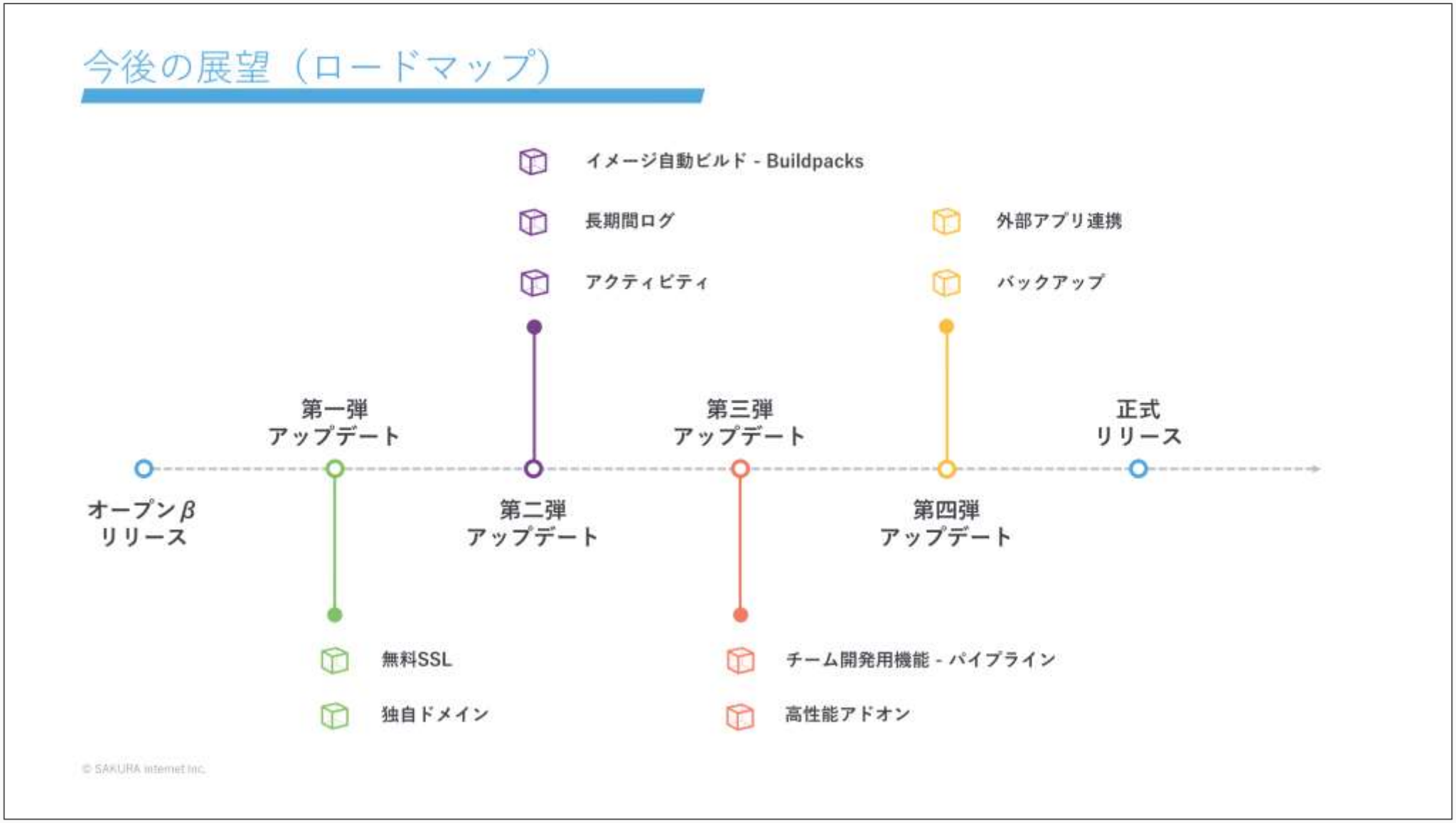
https://manual.c1.hacobuneapp.com/docs

Hacobune(β)が提供する機能

- アプリケーションのデプロイ
  - 作成元は、Docker イメージ、プライベートレジストリ、GitHub リポジトリから選択
  - Web UI からログの参照やモニタリング、再起動など操作可能
  - 公開先は https://<名前>.c1.hacobuneapp.com/ （BASIC認証も追加可能）
  - ヘルスチェックやオートスケール機能
  - データのボリューム（永続化）を提供
- 付加機能
  - アドオン (MySQL, Redis)
  - ジョブ機能 (cron的な概念)
  - コンテナレジストリ



今後のロードマップ


今後の展望（ロードマップ）
イメージ自動ビルド - Buildpacks
長期間ログ
アクティビティ
外部アプリ連携
バックアップ
第一弾
アップデート
第三弾
アップデート
正式
リリース
オープンβ
リリース
第二弾
アップデート
第四弾
アップデート
無料SSL
独自ドメイン
チーム開発用機能 - パイプライン
高性能アドオン
© SAKURA internet Inc.

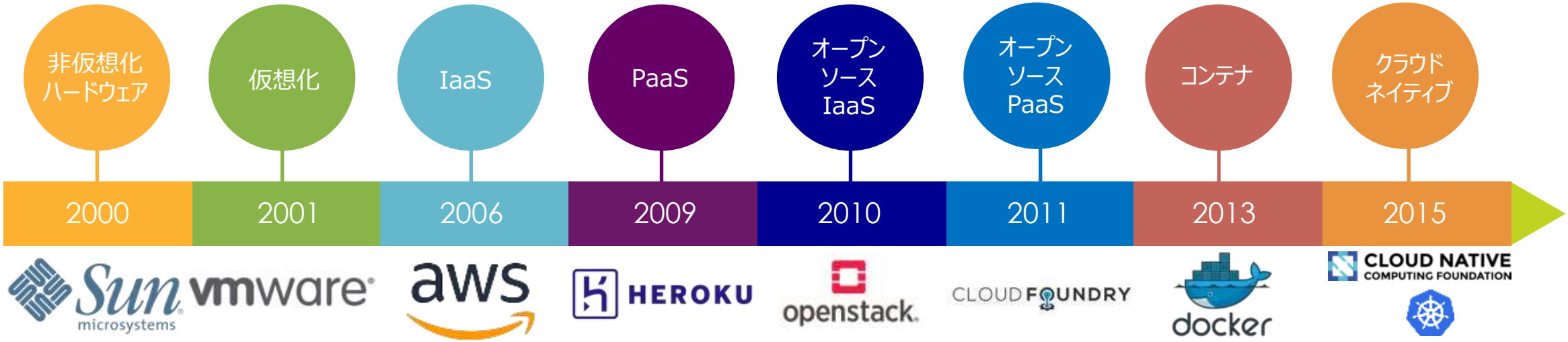
振り返り

仮想化からクラウド・ネイティブへ
From Virtualization to Cloud Native

CLOUD NATIVE COMPUTING FOUNDATION

kubernetes

- クラウド・ネイティブ・コンピューティングはオープンソースのソフトウェアを積み重ね、次のために用います：
  - アプリケーションをマイクロサービス(*microservices*)に分割し、
  - 各パーツ自身をコンテナにパッケージし、
  - リソース利用を最適化するために、動的に統合/オーケストレート(*orchestrate*)する



※ https://github.com/cncf/presentations からの引用

ここまでのポイント『手段としてのコンテナ活用』

- クラウドコンピューティングの振り返り

  1つの巨大なコンピュータに集約される予測(「クラウド化する世界」)は外れ、世界各地のデータセンタやスマートフォンに至るまで処理が分散し、生活に密着した。

- Dockerコンテナ登場から、Kubernetesオーケストレーションに至る経緯

  クラウドネイティブに至る流れは、物理→仮想化→クラウドの延長線だが、技術は別。活用には、CNCFのCloud Native Trail Map にあるように、システムの「コンテナ化」→「CI/CD」→「オーケストレーション」の順番が大事。

- 「コンテナ」とは何か？　システムを作る場合の考慮点

  Linuxカーネルの名前空間・cgroupの分離技術を使って、素早い開発・運用を可能に。仮想化ではない。そのまま移行は無理。スケール可能なシステムへの対応が必要。近年は、技術要素よりも、開発文化がネックになりつつある。

ありがとうございました。

facebook.com/zembutsu

m-zembutsu@sakura.ad.jp