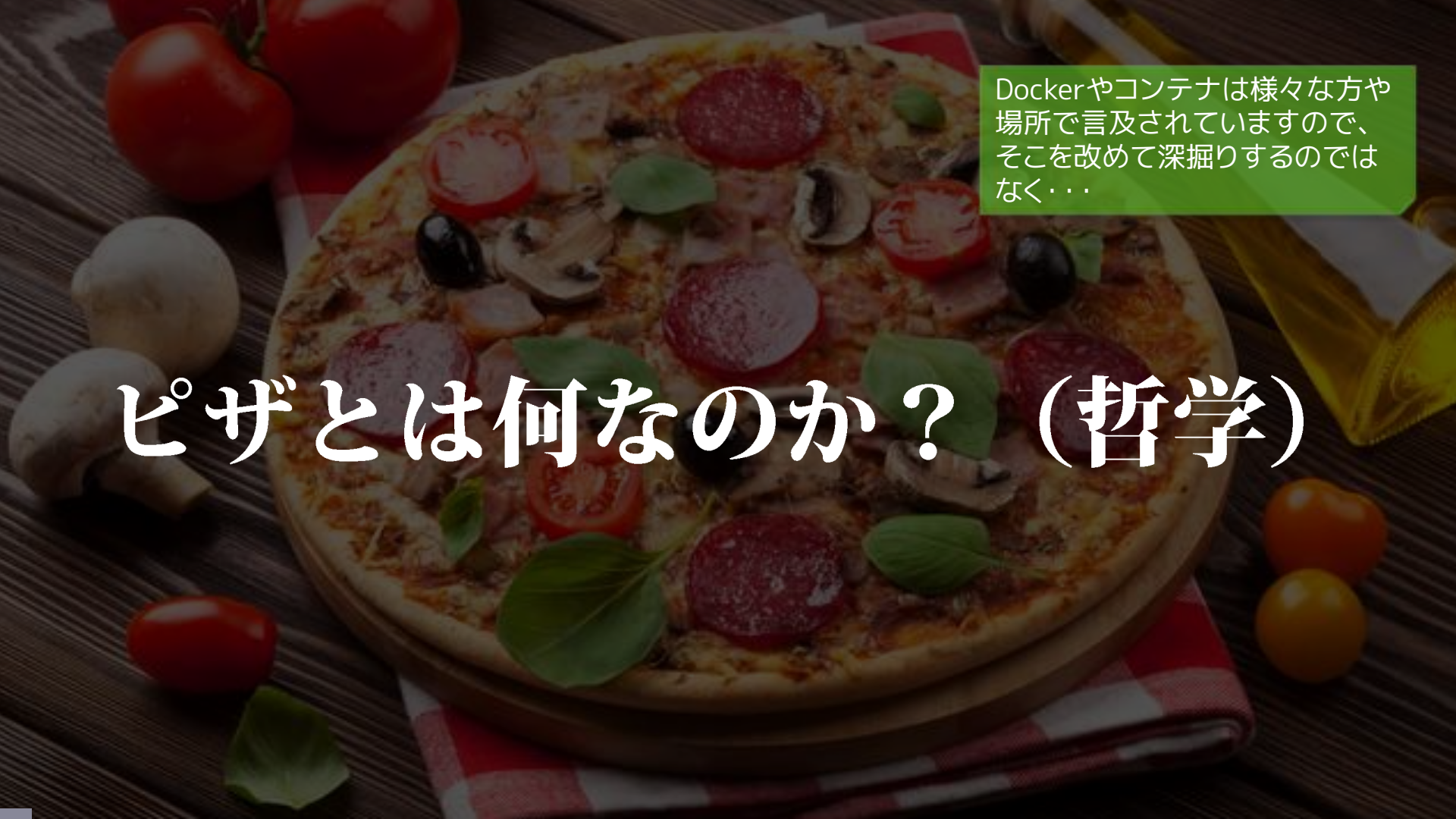
Docker概要：コンテナ技術と普及がシステム・インテグレータに与える影響

2016年3月4日(金)

@zembutsu

Technology Evangelist; Creationline, Inc.


Dockerやコンテナは様々な方や場所で言及されていますので、そこを改めて深掘りするのではなく・・・
ピザとは何なのか？（哲学）


いまここにあるDockerを、どのようにして使っていけば良いのか？
ピザであれば「どう食べるのか」という視点で、皆さんと共有したいと思っています。
ピザをどう食べたらいいのか？

コンテナ時代にシステムインテグレータがこの先生きのこるには?

2016年3月4日(金)

第35回NCWG発表資料

@zembutsu

Technology Evangelist; Creationline, Inc.


システム要件に
コンテナ？
こマ？
どうすんだ
これ…

本資料の想定読者

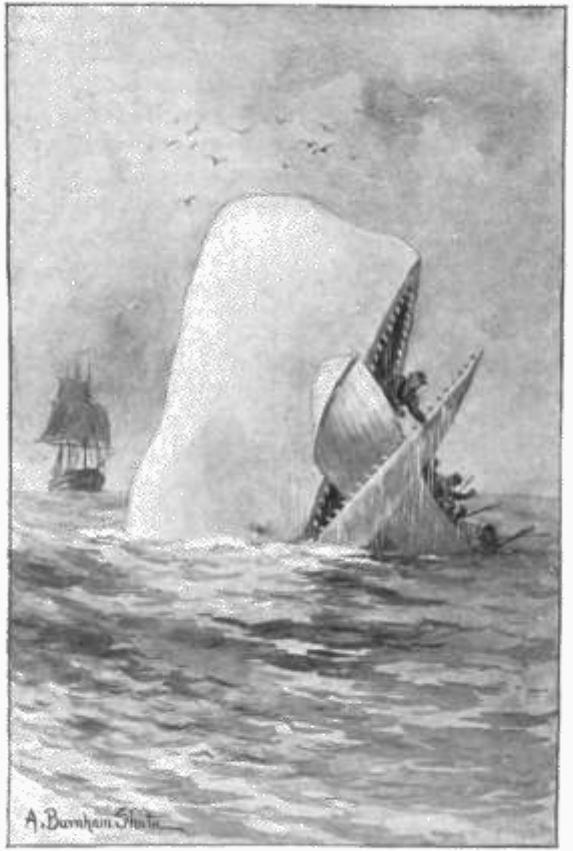
- 営業）お客さまが「Docker」を**使いたい**と言われたら？
- 上長）部下が「Docker」を**使いたい**と言ってきたら？
- 現場）上長・役員に「Docker」を**使うべき**と伝えるには？

本資料における主張

- 「Docker」を**導入するだけでは価値を生まない**。
- 「Docker」は技術選択要素の１つにすぎないので、
  業務フローに一致するならば、積極的に導入すべきだ。
- **継続的**な技術評価を行うべきだ。

.

「SIがコンテナ時代に、この先生きのこるために」



"Both jaws, like enormous shears, bit the craft completely in twain."

—*Page 510.*

QUEEQUEG AND HIS HARPOON.

https://en.wikipedia.org/wiki/Moby-Dick


FORTUNE
PANAMA
炎上
不可避

本スライドの内容

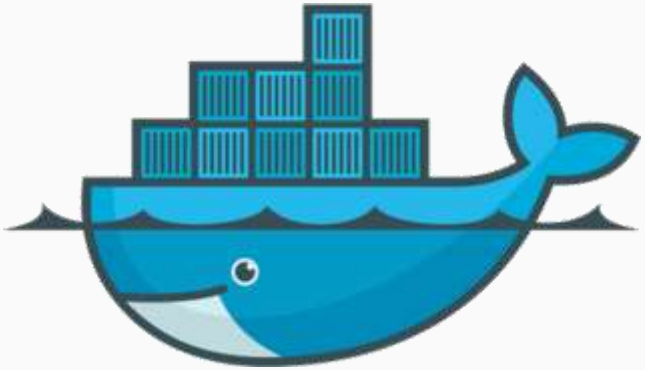
- 10分で分かる「Docker」の世界
- なぜDockerが必要なのか、業務に何をもたらすのか？
- 導入時の技術選択要素と誤解（都市伝説）
- 質疑応答

1

10分で分かるDockerの世界

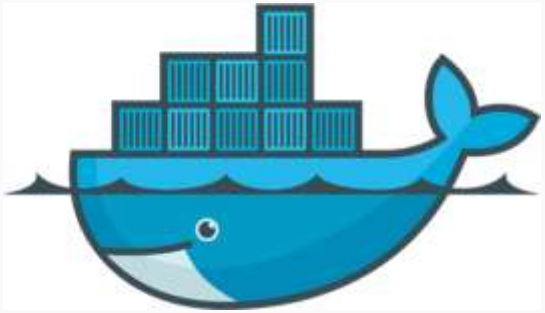
■□□□ What is the "Docker" ?

アクセントの悩み



docker

／ドッ＼カー　　ドッ／カー→



docker

Docker プロジェクト

- 開発者やシステム管理者が、アプリを開発・移動・実行するプラットフォーム
- Docker, inc. はスポンサー
  primary contributor, maintainer

Docker デーモン

- イメージとコンテナを管理
- ホスト上で動くプロセス
- Linuxコンテナ技術を使う
  namespaces, cgroups

※Docker Glossary https://docs.docker.com/engine/reference/glossary/

Docker, Inc.

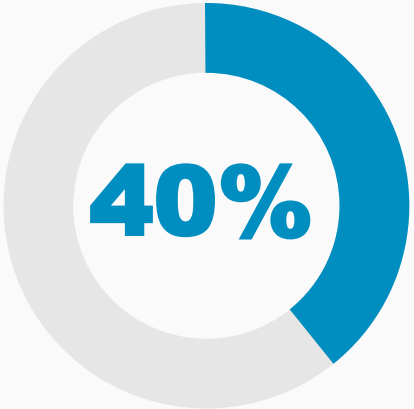
商用Dockerソリューションの提供

- 構築・運搬・実行を統合するソリューション
- 公式プロバイダによる商用技術サポート (Docker, IBM)

Docker プロジェクトのスポンサー

- Docker オープン・ソースに対するプライマリ・コントリビュータ及びメンテナ
- 10億以上のイメージをダウンロード
- 1500人以上の貢献者（コントリビュータ）
- 200,000以上の Docker 対応アプリケーション

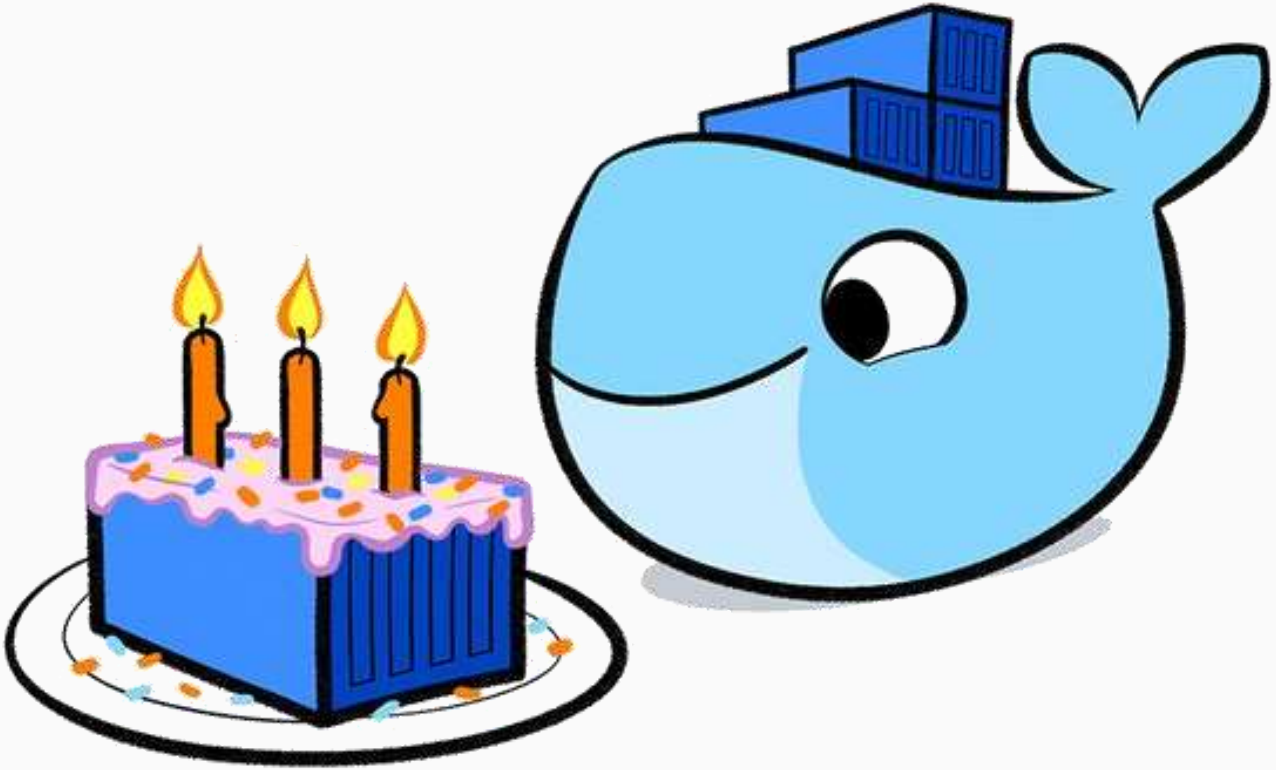
Docker を既に
プロダクションで利用中



Gerber, Anna. “The State of Containers and the Docker Ecosystem: 2015” O’Reilly, September 2015

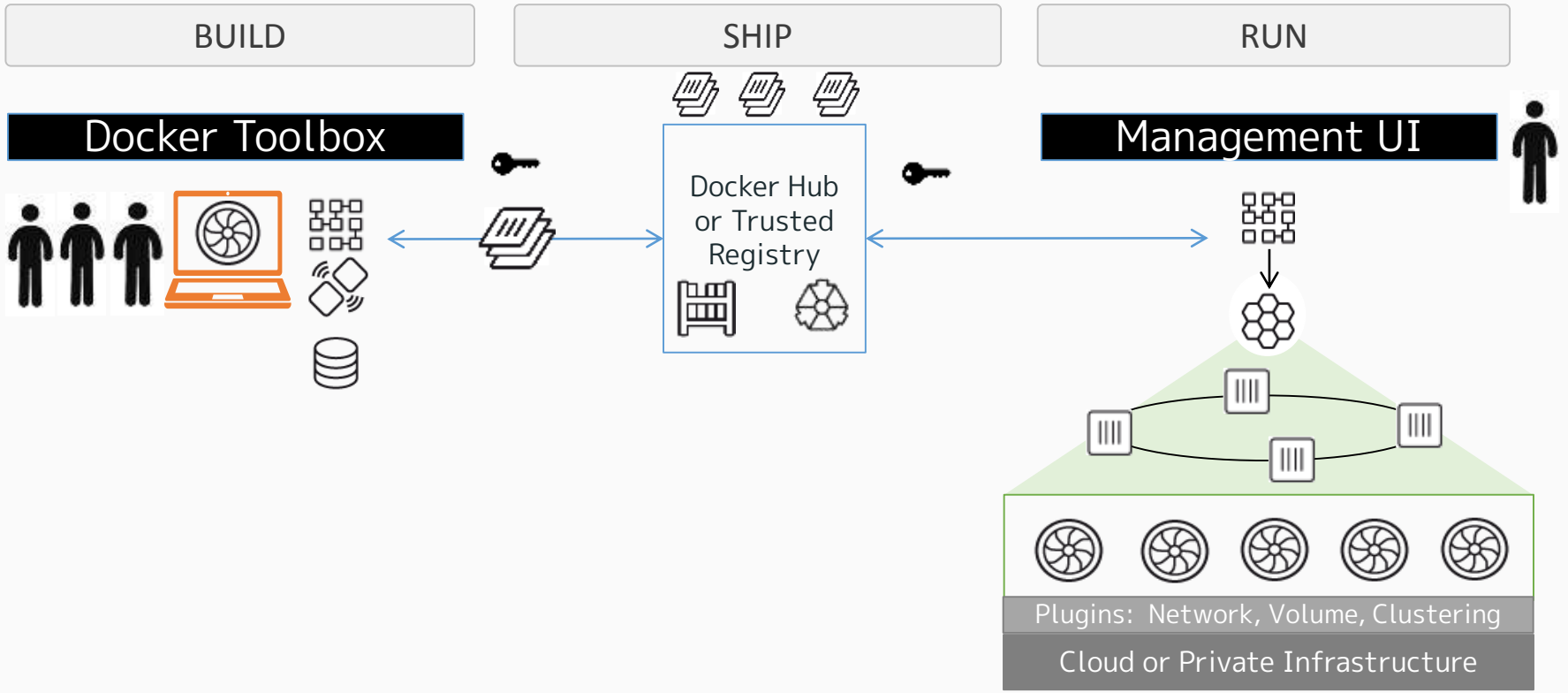
分野	OSS	使用率
OS	Linux 系	67.3%
	BSD 系	12.9%
RDBMS	MySQL	53.1%
	PostgreSQL	35.0%
アプリケーションサーバー	Tomcat	35.6%
	JBoss	12.0%
システム運用管理	Zabbix	16.2%
	Nagios	7.1%
	Chef	3.9%
	Hinemos	1.9%
システムソフトウェア	Samba	21.4%
	BIND	13.6%
ハイパーバイザー	Xen	16.2%
	KVM	10.7%
クラウド基盤	OpenStack	6.1%
	Docker	4.5%
	CloudStack	3.6%
	Cloud Foundry	2.9%
データ分散処理	Hadoop	6.8%
	Spark	1.3%
NoSQL	MongoDB	4.5%
	Scalaris	4.2%
	Cassandra	2.6%
	Hypertable	2.6%

引用：国内企業におけるオープンソースソフトウェアの利用実態調査結果を発表
http://www.idcjapan.co.jp/Press/Current/20160204Apr.html



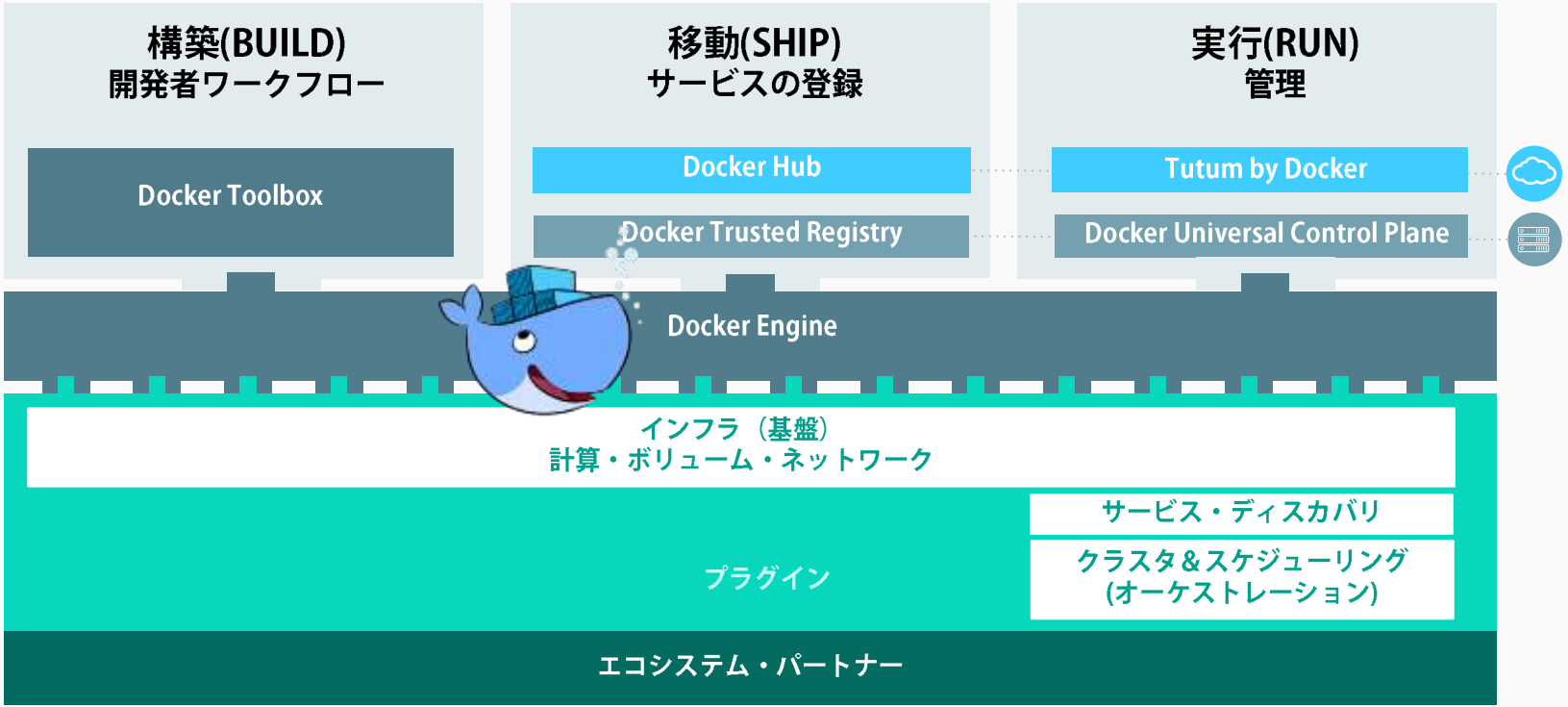
プラットフォーム

「Docker」とは、アプリケーションを動かすための仕組みを提供



プラットフォーム

開発者が簡単にアプリケーションを開発・デプロイできる仕組み



Docker Engine

コンテナ

"docker" デーモン

ユーザ空間

ユーザプロセス

ユーザプロセス

ユーザ空間

ユーザプロセス

TCP:2375, TCP:2376(TSL), Unix Socket

OS のカーネル空間

サーバ

Docker サーバ

docker run …

Docker クライアント

- **Cgroups**
- **Namespaces**
  - ✓ mount
  - ✓ UTS
  - ✓ IPC
  - ✓ PID
  - ✓ Network
  - ✓ User

Ubuntu レポジトリ

tag:latest

tag:14.04

…

MySQL レポジトリ

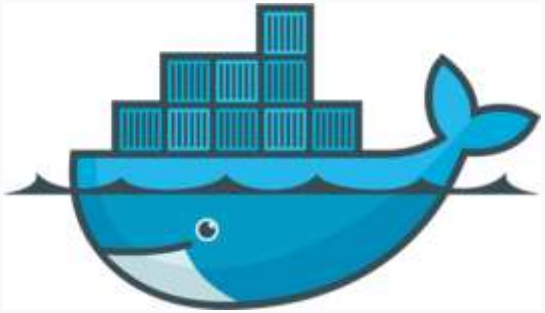
tag:latest

tag:5.7

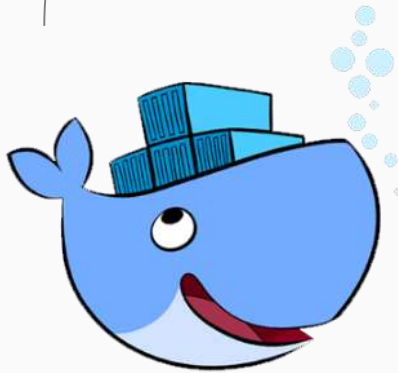
…

Docker Hub (レジストリ)

https://hub.docker.com/



docker



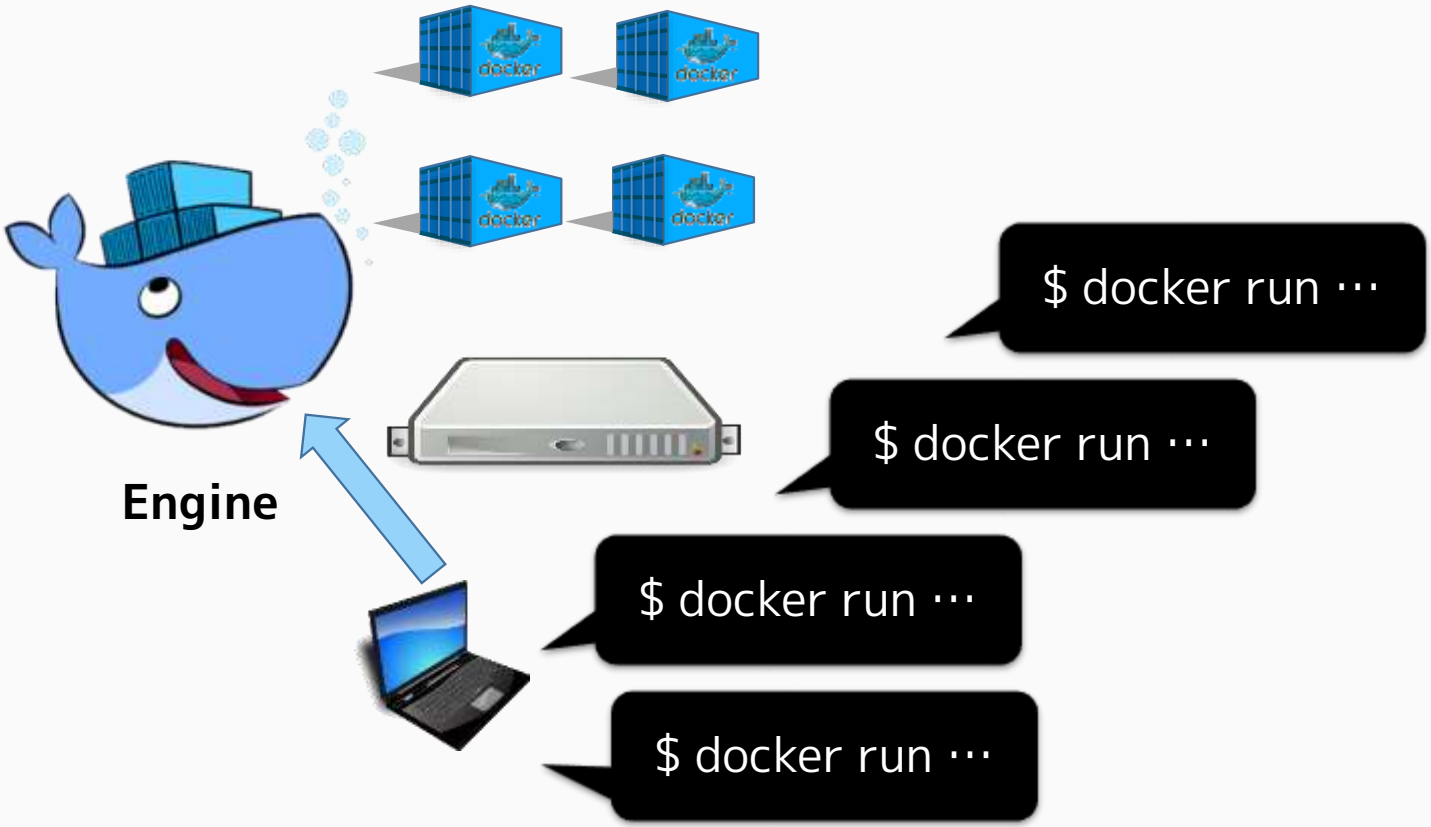

docker
MACHINE

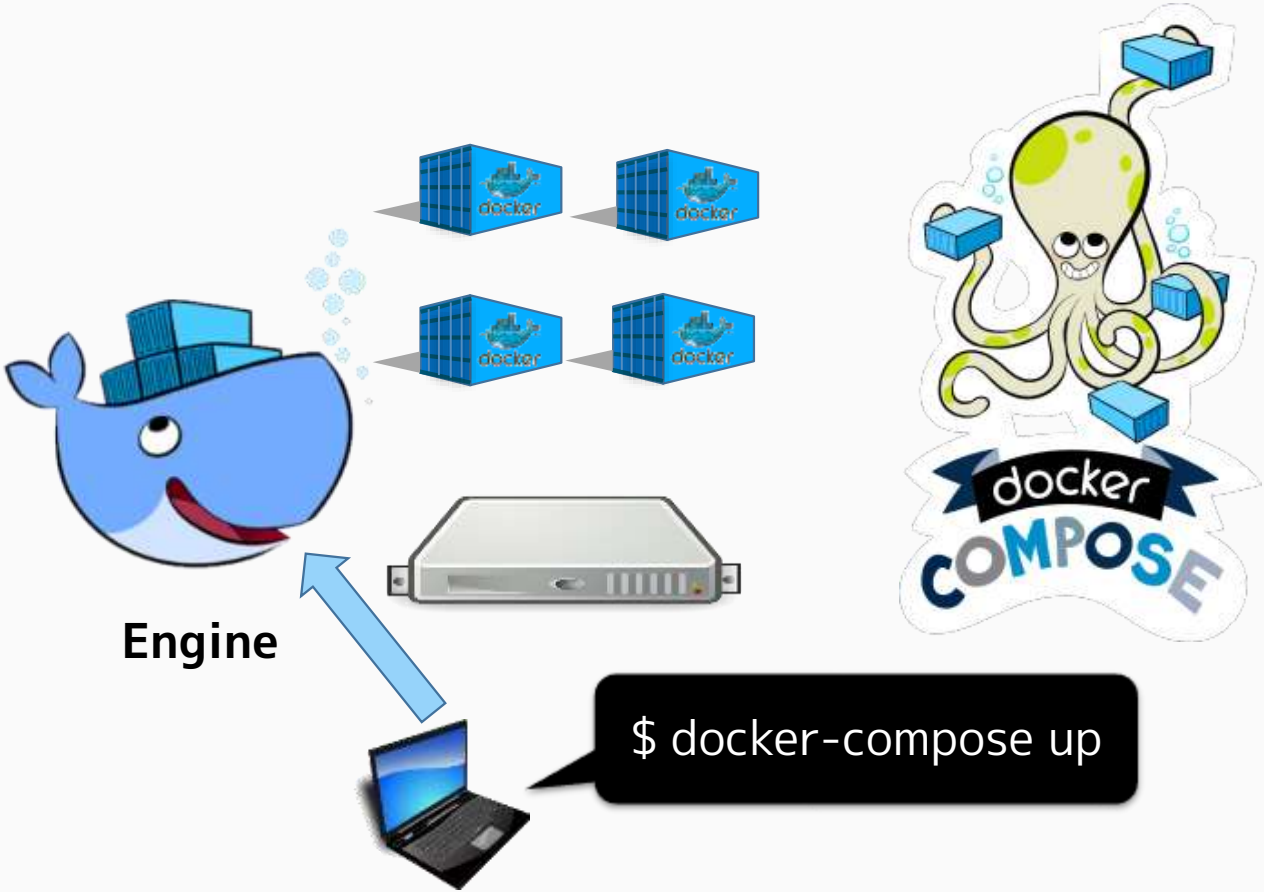

docker
SWARM

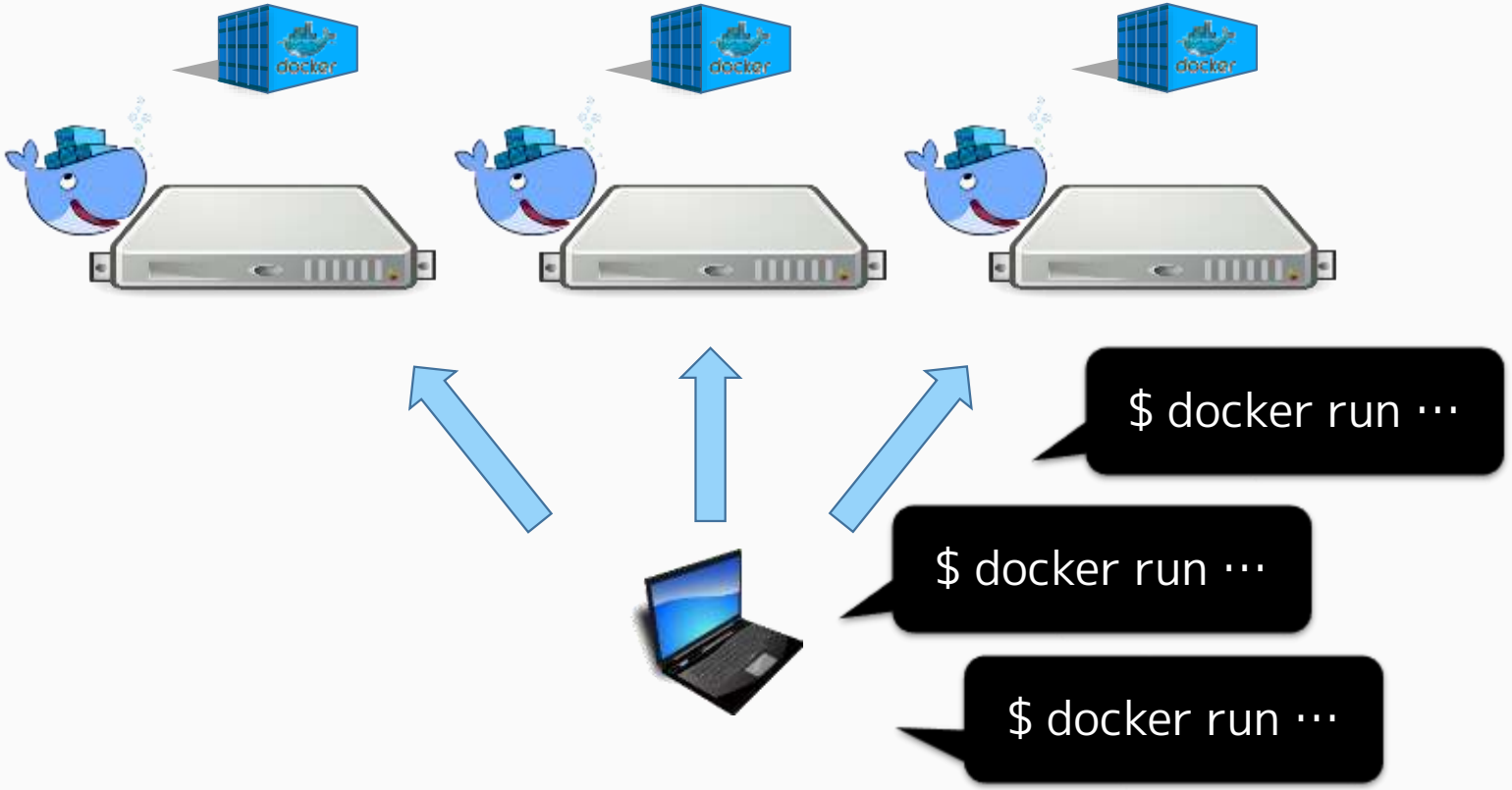

docker
COMPOSE

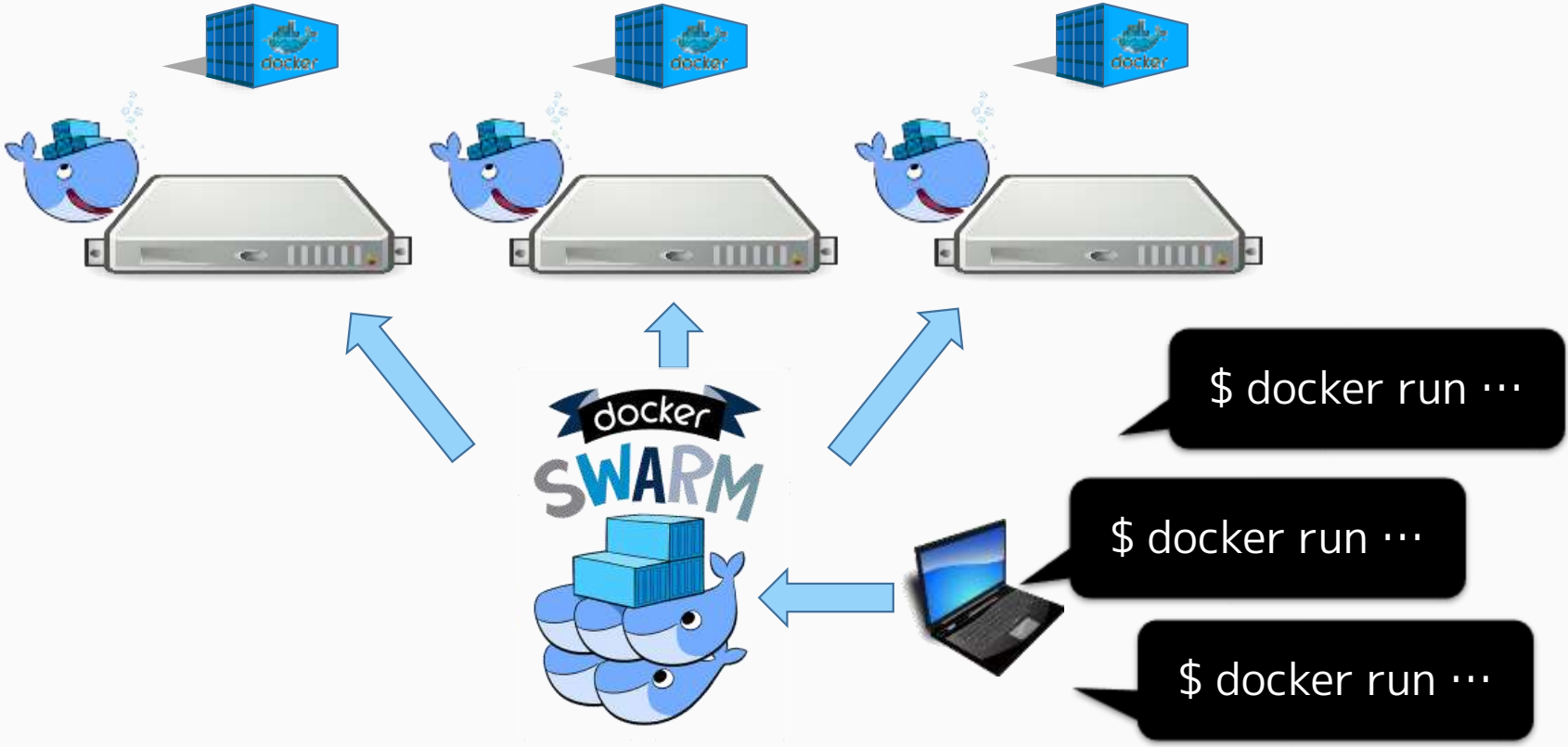

docker
REGISTRY


docker
DOCKER
DATACENTER


$ docker run ⋯
$ docker run ⋯
$ docker run ⋯
$ docker run ⋯
Engine


docker
docker
docker
docker
docker
COMPOSE
Engine
$ docker-compose up


docker
docker
docker
$ docker run ⋯
$ docker run ⋯
$ docker run ⋯


docker
docker
docker
docker
SWARM
$ docker run ⋯
$ docker run ⋯
$ docker run ⋯

tutum

https://www.tutum.co/

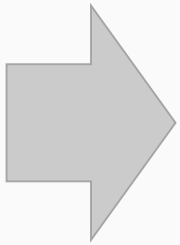
Docker社が買収(2015年10月)

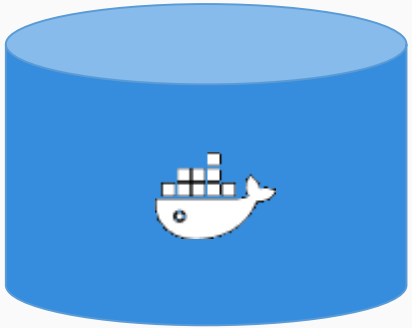
AWS, Digital Ocean, Azure, SoftLayer, Packet と連携

コンテナのデプロイと管理

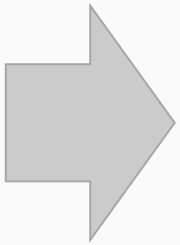
Docker Hub を補完する役割

パブリック環境でコンテナを実行





Docker Hub



http://cloud.docker.com

(旧Tutum)

より安全にコンテナを実行

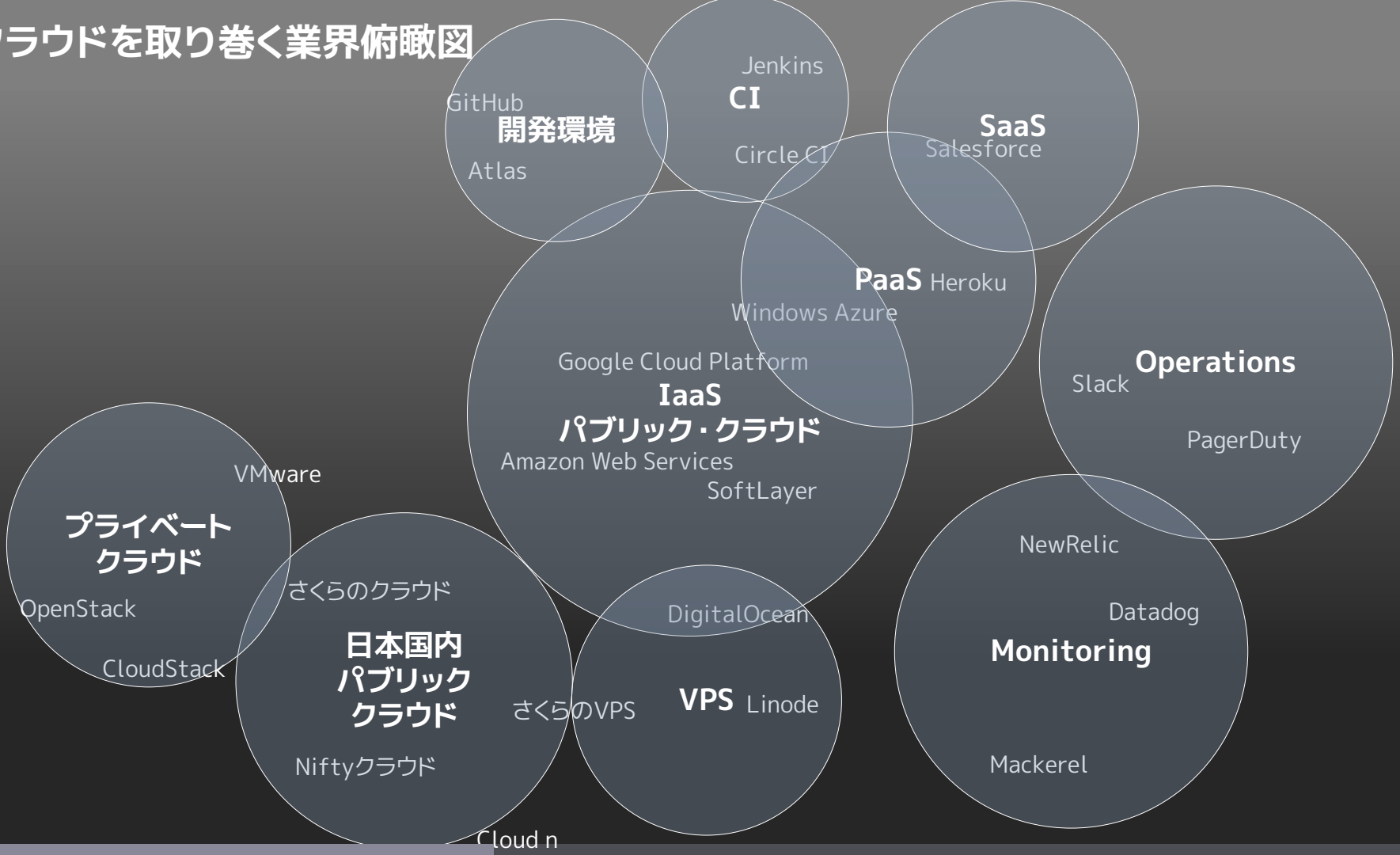
DTR
Docker Trusted Registry

Orca (beta) + UCP
Universal Control Plane

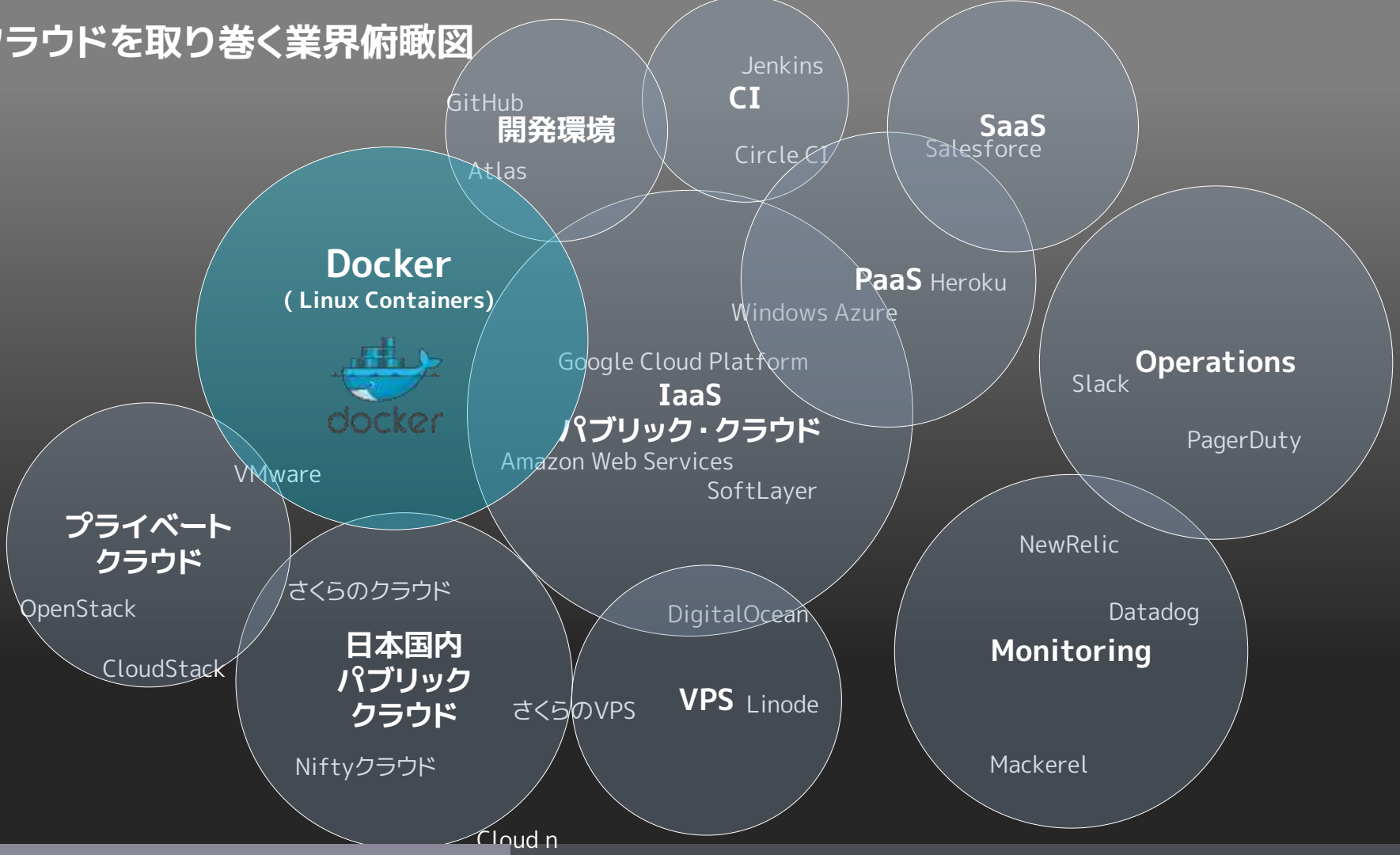
https://www.docker.com/universal-control-plane

クラウドを取り巻く業界俯瞰図

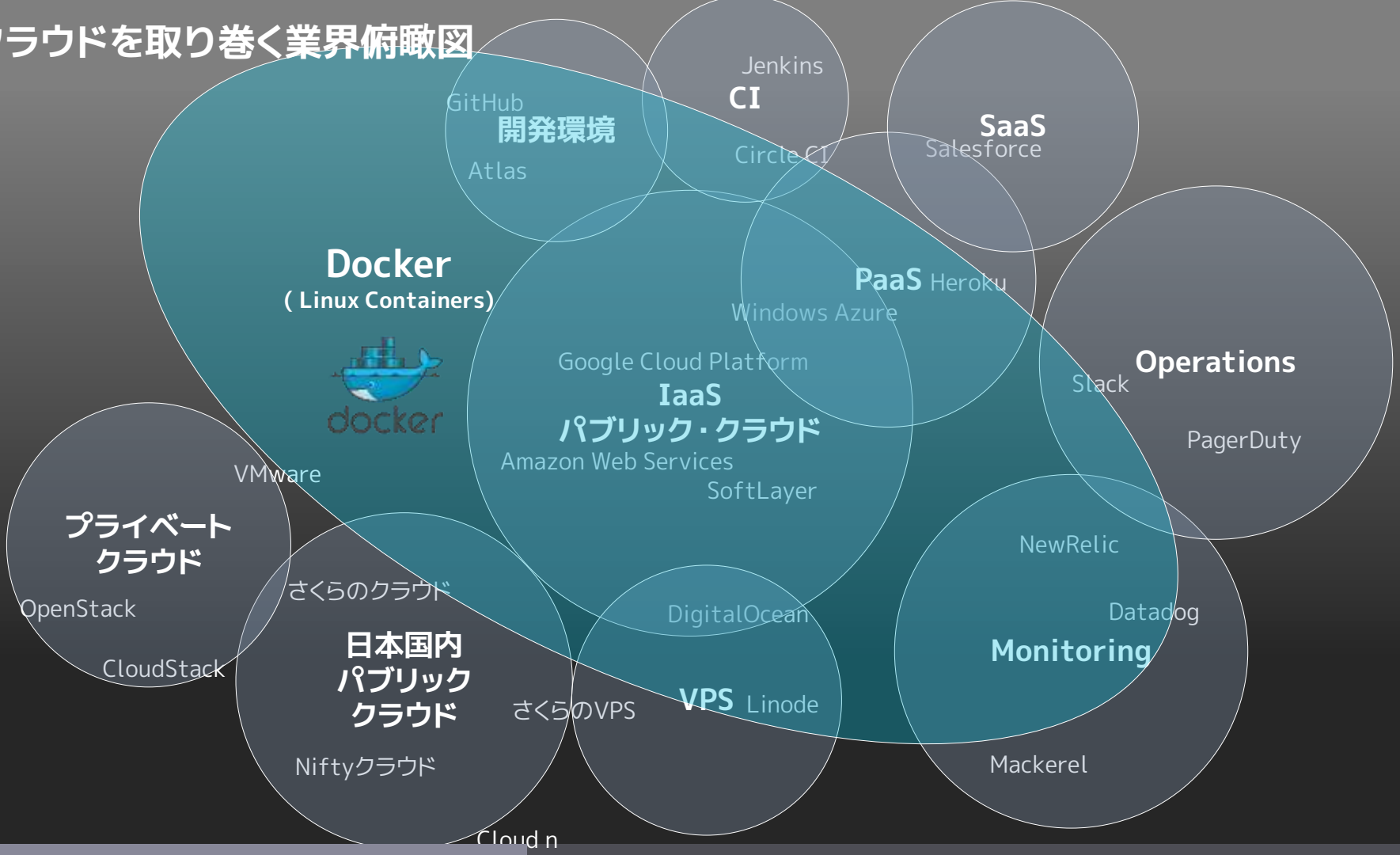
クラウドを取り巻く業界俯瞰図



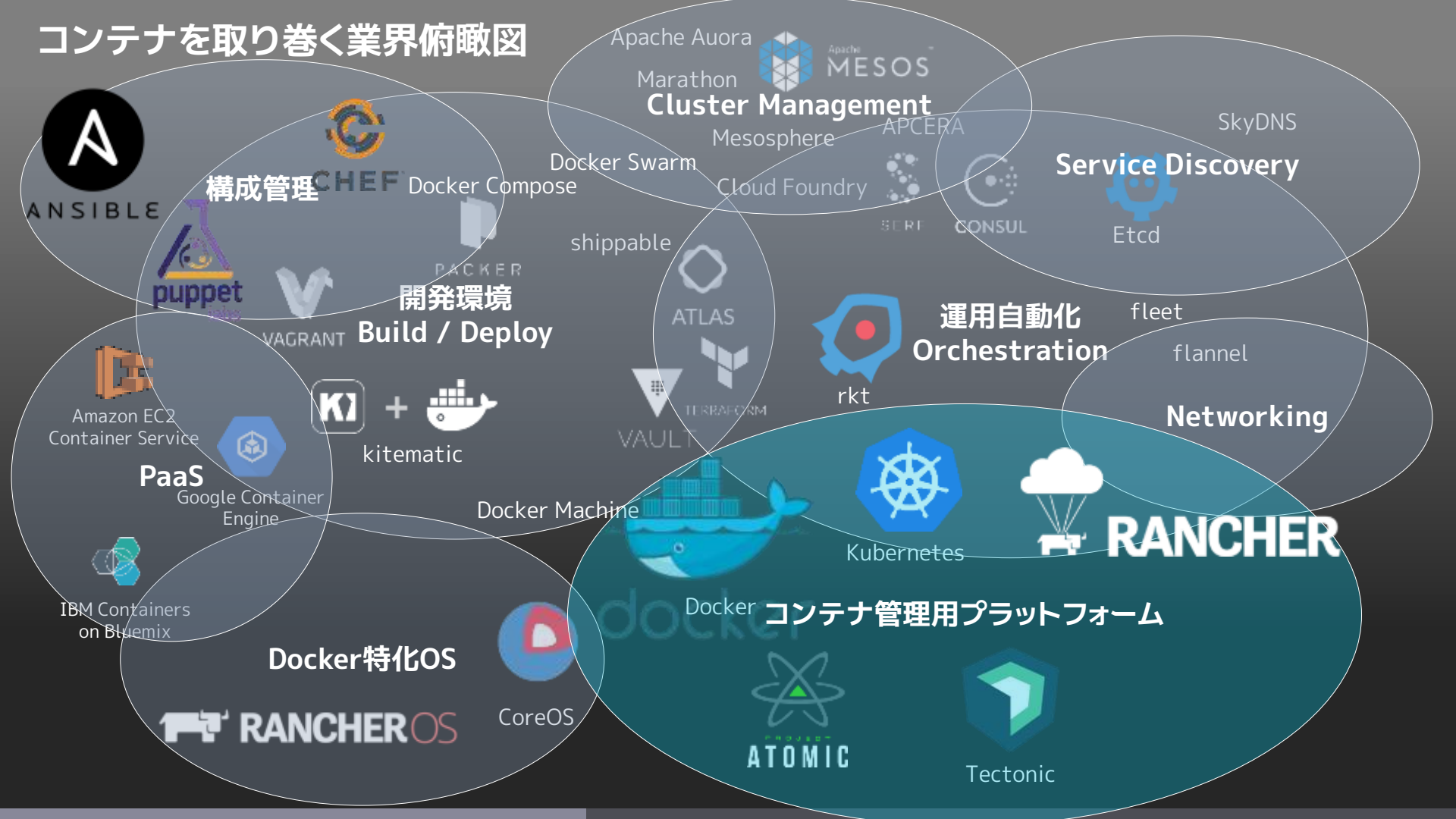
クラウドを取り巻く業界俯瞰図

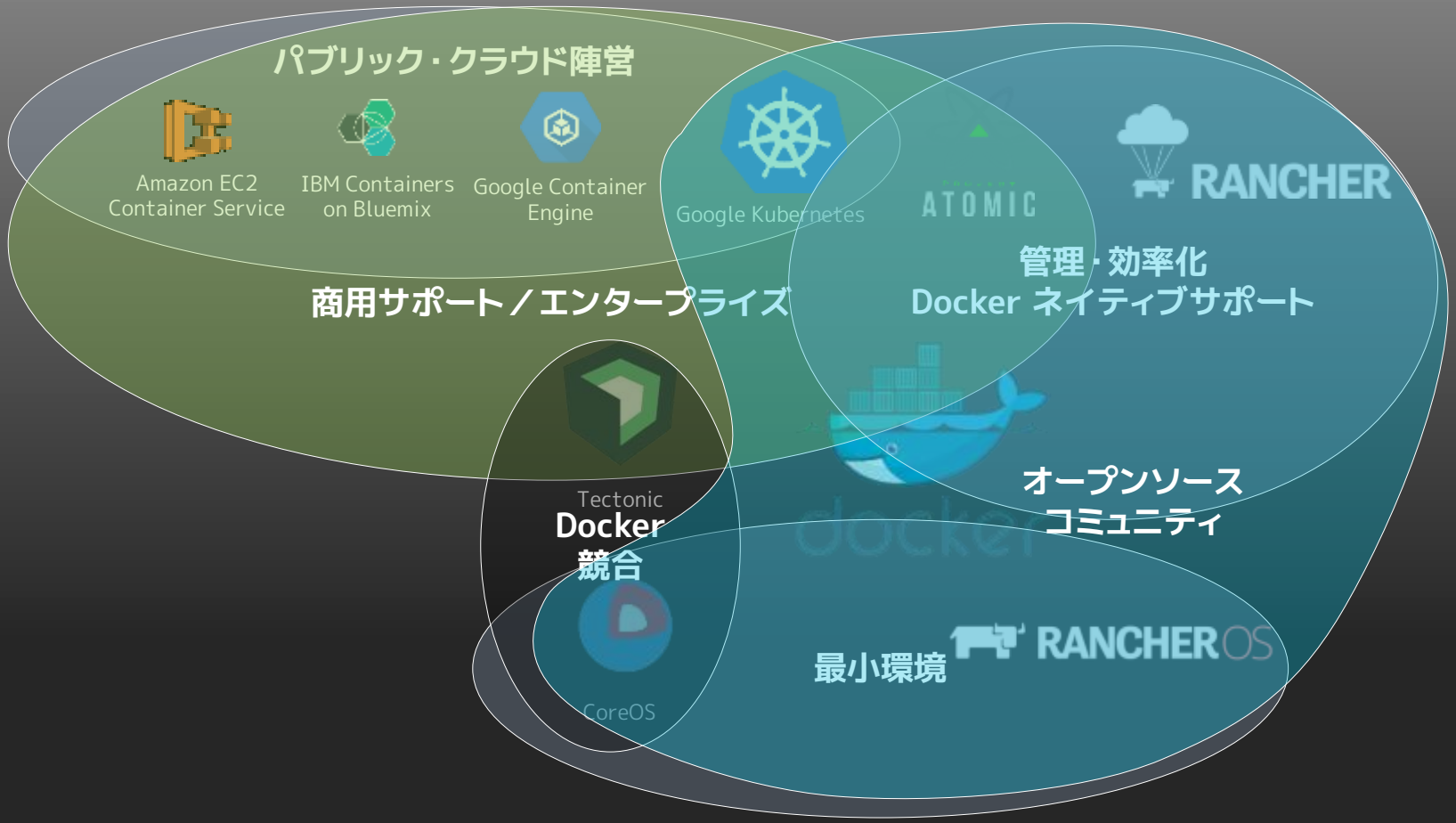


クラウドを取り巻く業界俯瞰図



コンテナを取り巻く業界俯瞰図




パブリック・クラウド陣営
Amazon EC2
Container Service
IBM Containers
on Bluemix
Google Container
Engine
Google Kubernetes
ATOMIC
RANCHER
管理・効率化
Docker ネイティブサポート
商用サポート／エンタープライズ
Tectonic
Docker
競合
オープンソース
コミュニティ
docker
最小環境
RANCHEROS
CoreOS

DockerCon 2015 から流れが変わる

http://www.dockercon.com/

- **Docker, Inc.主催**
  - ➡ サンフランシスコ
  - ➡ 6月22日･23日の2日間
  - ➡ セッション、ハンズオン、トレーニング、ブース出展

The Open Container Project (OCP)

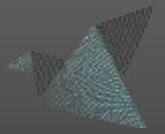
Linux Foundation 事務局として協力

Orchestration

weave

Google Kubernetes

PROJECT ORCA



Amazon ECS

IBM Containers on Bluemix

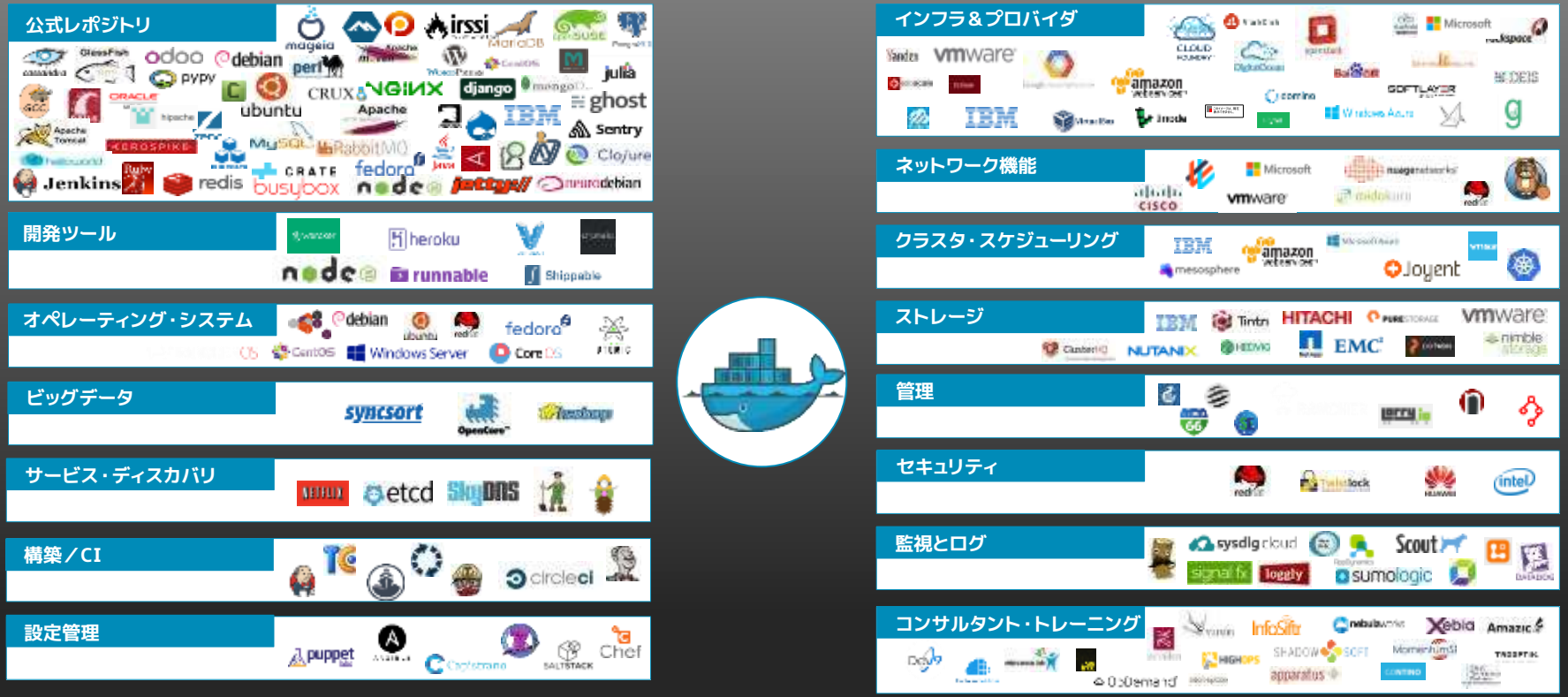
Google Container Engine

× libcontainer 廃止

CoreOS

runC

appcは独自に継続


公式レポジトリ
irssi
debian
perl
julia
ubuntu
Apache
IBM
Sentry
Clojure
fedora
Jenkins
redis
busybox
node
開発ツール
heroku
runnable
Shippable
オペレーティング・システム
debian
ubuntu
fedora
CentOS
Windows Server
CoreOS
ビッグデータ
syncsort
OpenCore
サービス・ディスカバリ
etcd
SkyDNS
構築／CI
circleci
設定管理
puppet
Chef
SALTSTACK
インフラ＆プロバイダ
vmware
amazon
IBM
Microsoft
ネットワーク機能
cisco
vmware
クラスタ・スケジューリング
IBM
amazon
mesosphere
Joyent
ストレージ
IBM
HITACHI
vmware
NUTANIX
EMC
管理
セキュリティ
intel
監視とログ
sysdig cloud
Scout
signalfx
loggly
sumologic
コンサルタント・トレーニング
InfoSiftr
Xebia

ここまでのまとめ

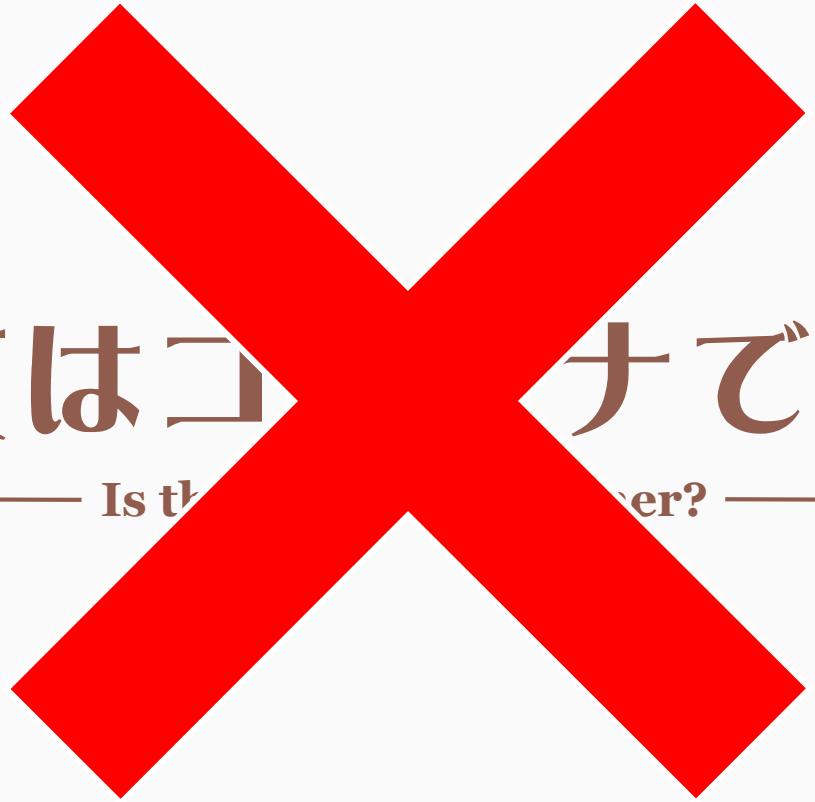
- Dockerは組織でもあり、技術でもある
- 簡単にアプリケーションを開発･移動･実行するための
  プラットフォームを提供する（≠インフラ）
- Dockerには用途に応じた様々なツールが存在する
- Dockerを中心としたエコシステムが形成されつつある

2 なぜDockerが必要になったのか？

■■□□ Why we need "Docker" ?

ご注文はコンテナですか？

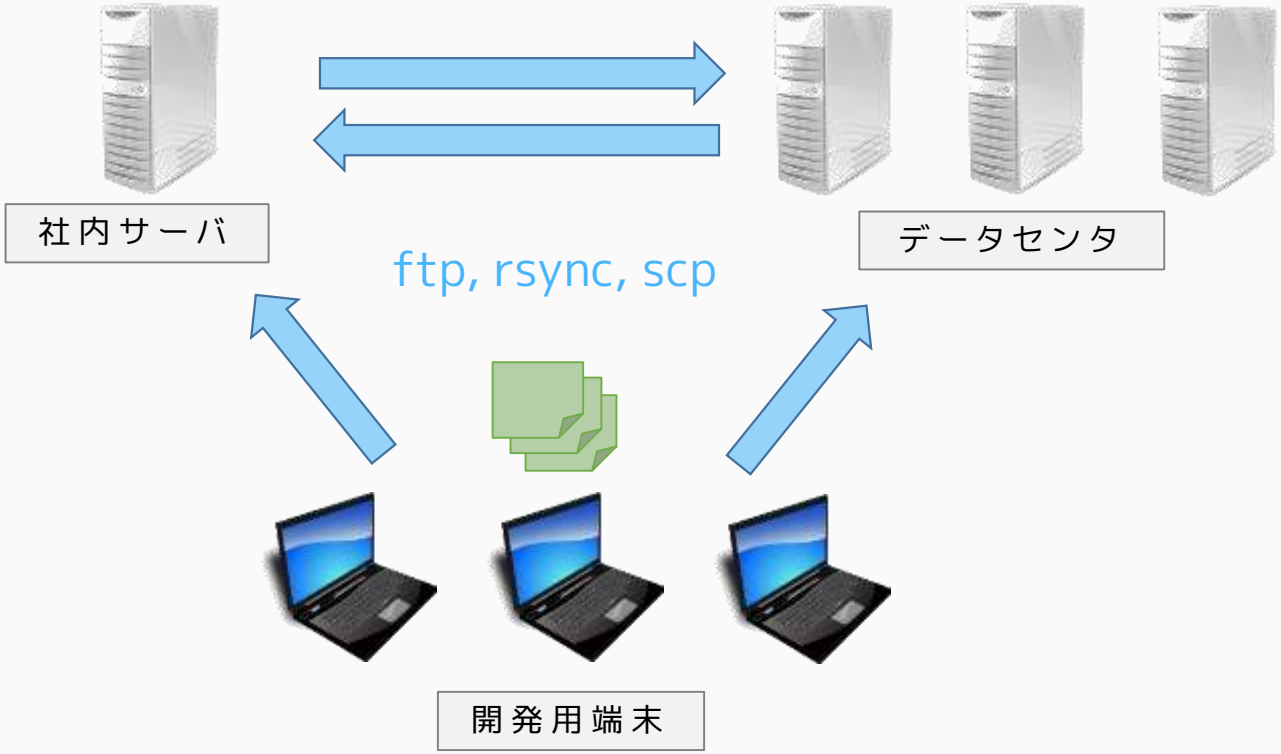
Is the order a container?



ご注文はコーヒーですか？

Is the coffee for here or to go?

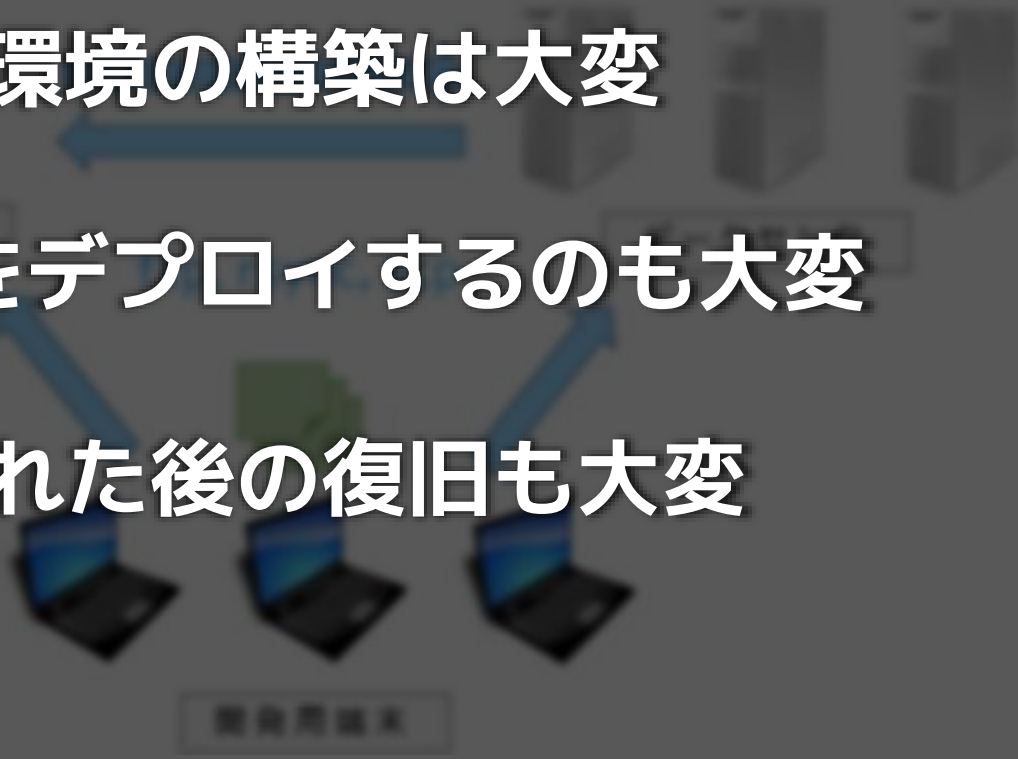
物理サーバ・ネットワーク環境の全盛時代



環境の構築は大変

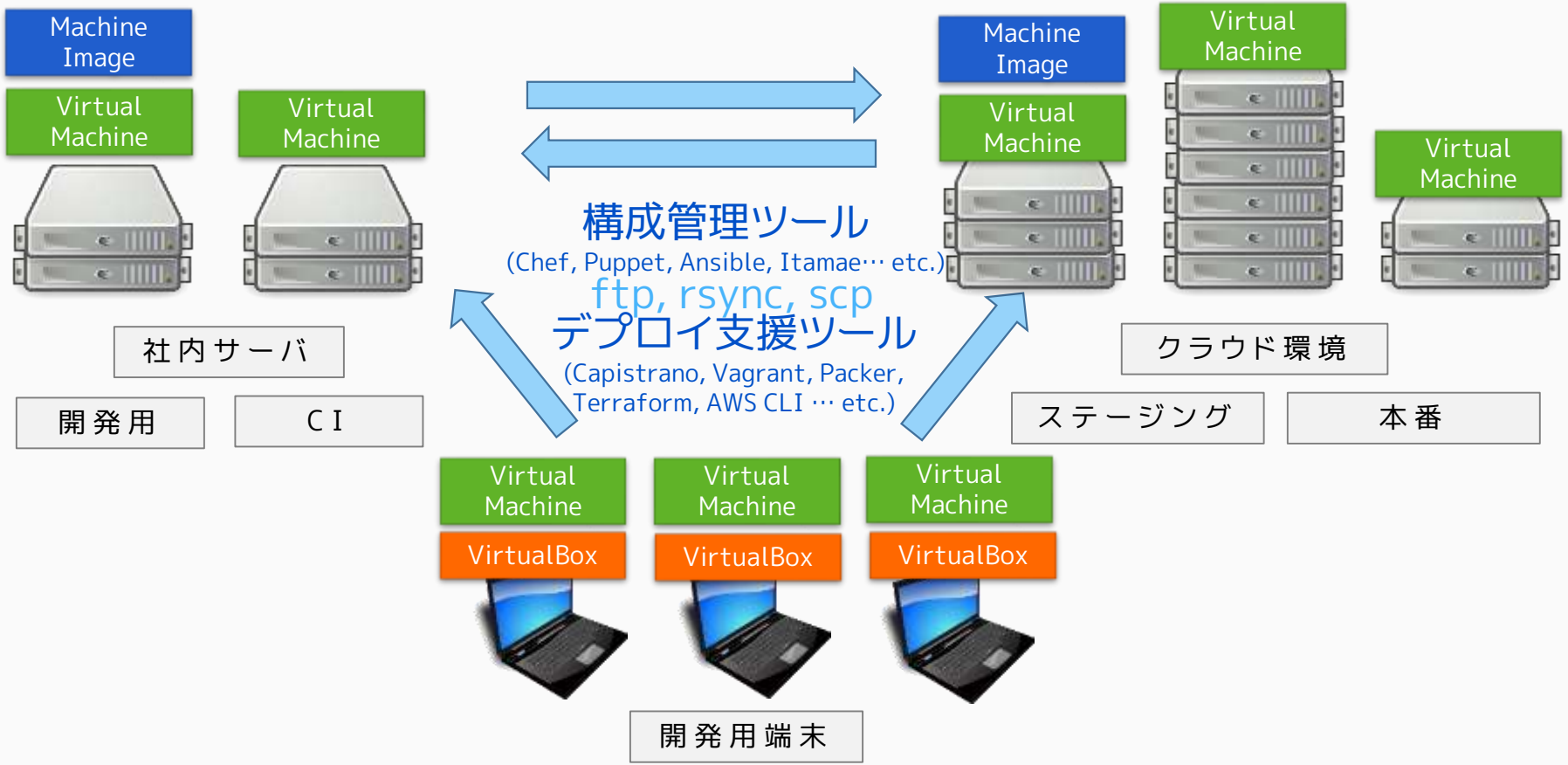
環境をデプロイするのも大変

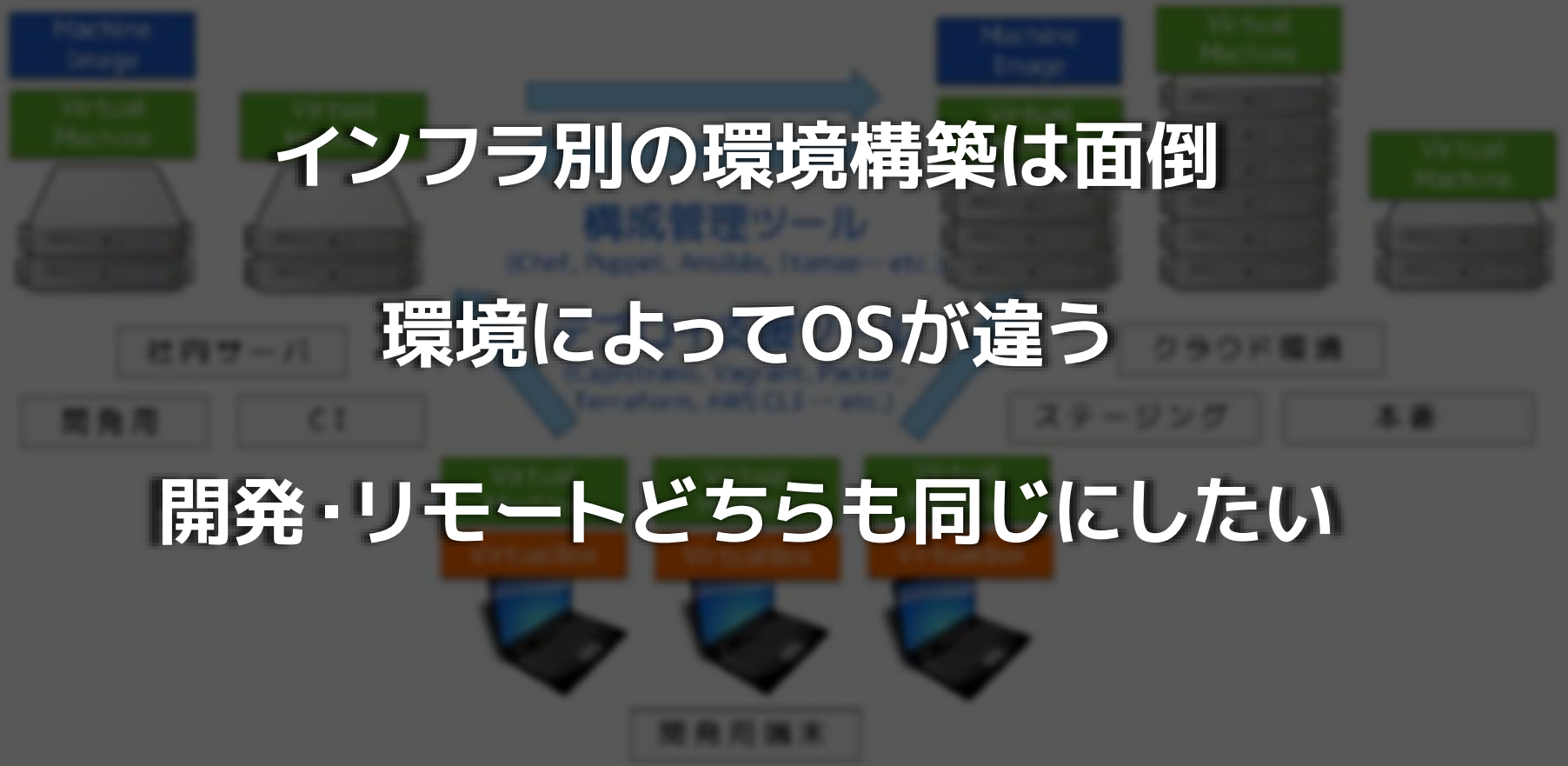
壊れた後の復旧も大変



突然の死

仮想化・クラウド時代

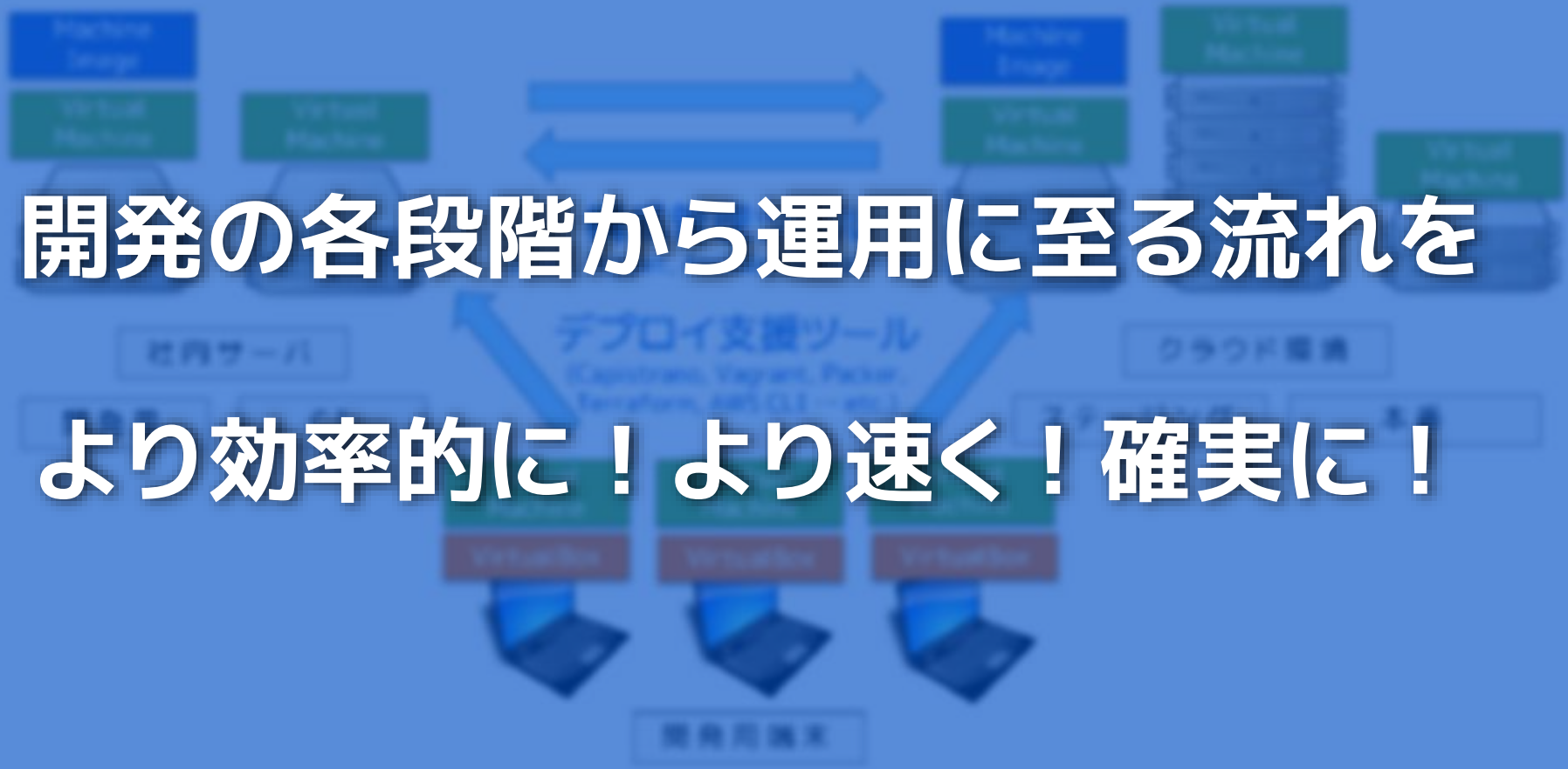

Machine Image
Virtual Machine
Virtual Machine
社内サーバ
開発用
CI
構成管理ツール
(Chef, Puppet, Ansible, Itamae… etc.)
ftp, rsync, scp
デプロイ支援ツール
(Capistrano, Vagrant, Packer, Terraform, AWS CLI … etc.)
Machine Image
Virtual Machine
Virtual Machine
Virtual Machine
クラウド環境
ステージング
本番
Virtual Machine
VirtualBox
Virtual Machine
VirtualBox
Virtual Machine
VirtualBox
開発用端末



インフラ別の環境構築は面倒

環境によってOSが違う

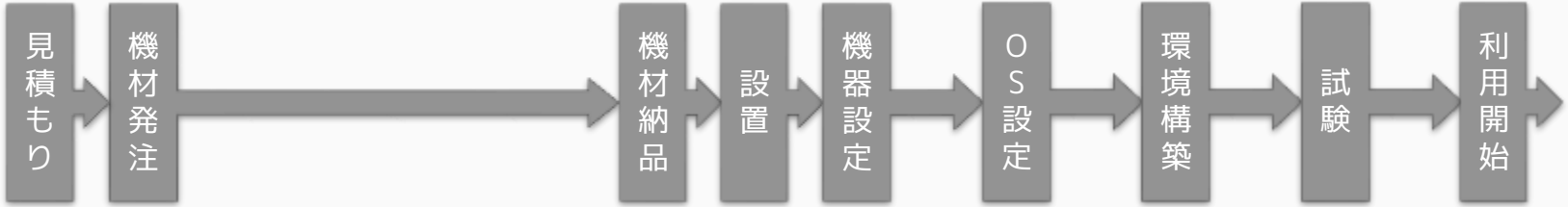
開発・リモートどちらも同じにしたい



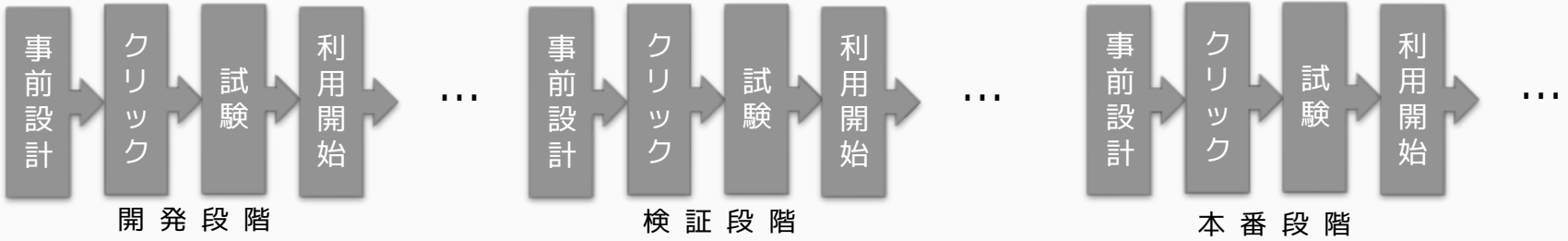
開発の各段階から運用に至る流れを

より効率的に！より速く！確実に！

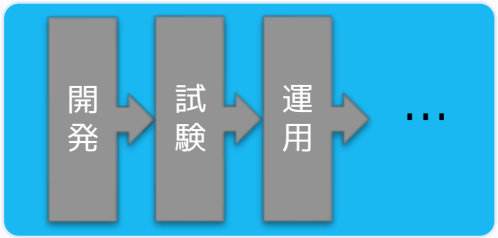
暗黒^H^H物理時代



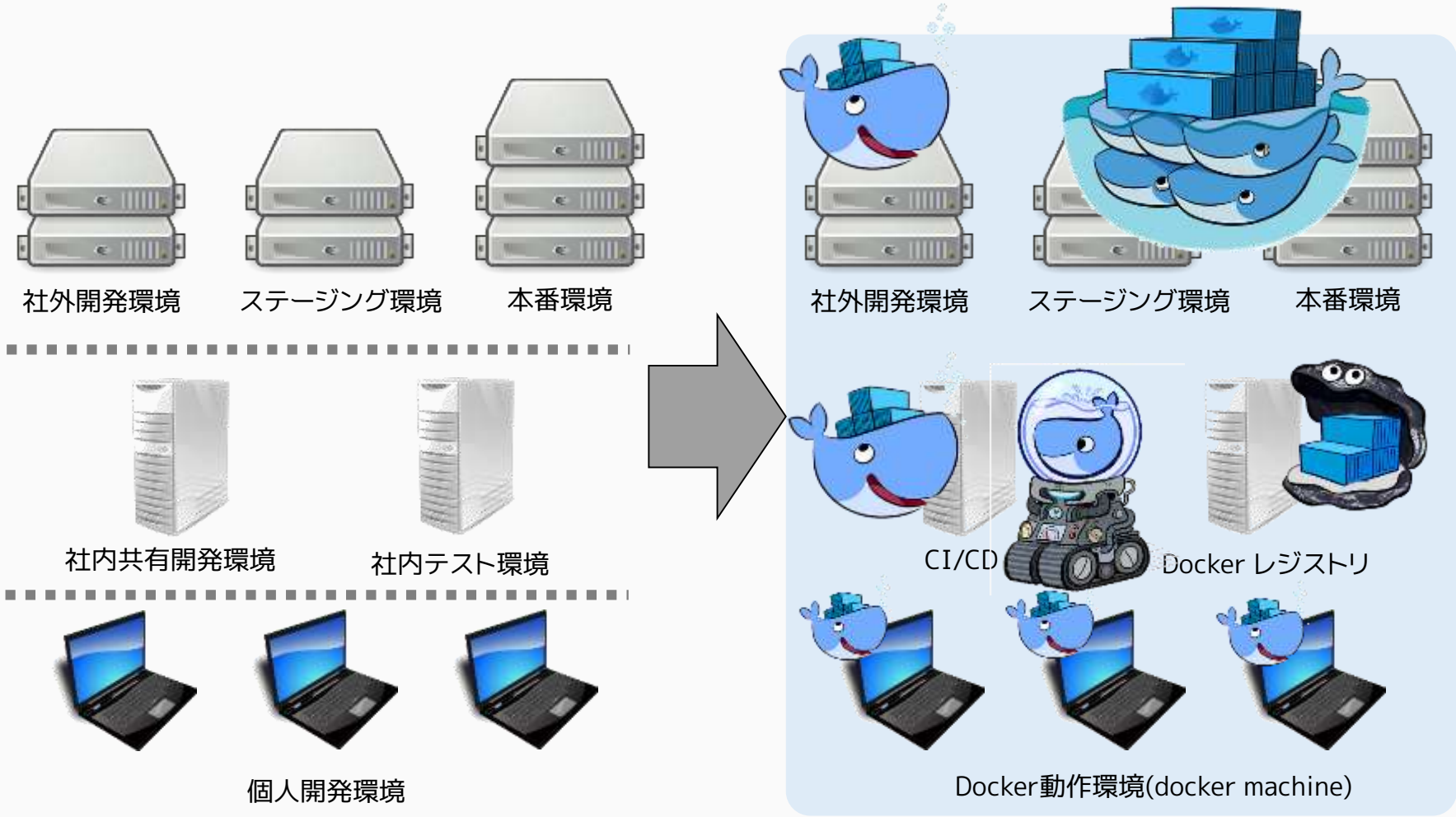
仮想化・クラウド時代



コンテナ時代



すべてを迅速に、一貫した環境で行いやすい← New


社外開発環境
ステージング環境
本番環境
社内共有開発環境
社内テスト環境
個人開発環境
社外開発環境
ステージング環境
本番環境
CI/CD
Docker レジストリ
Docker動作環境(docker machine)

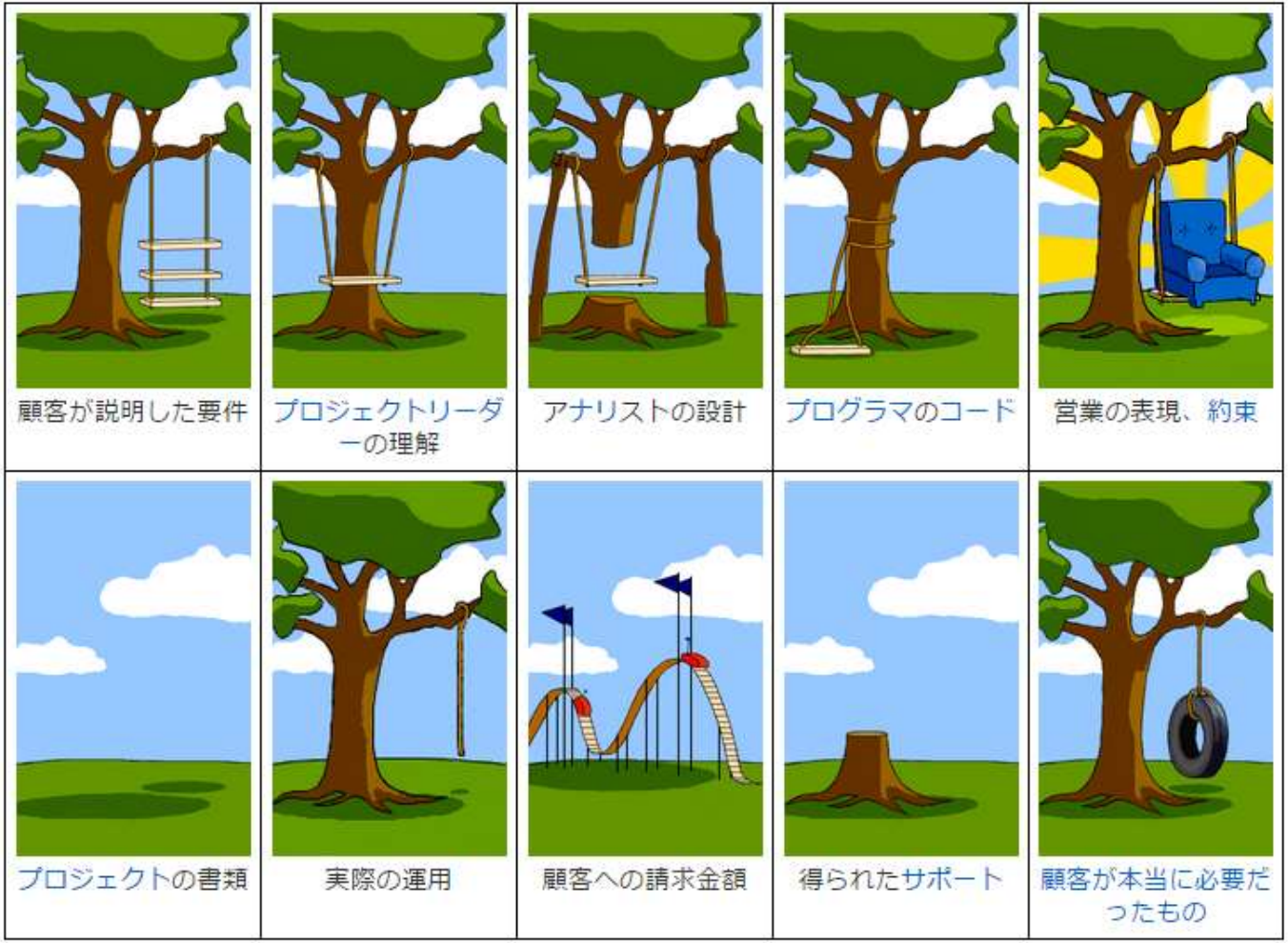
ここまでのまとめ

- 迅速な業務フローが求められている
- ハードウェア視点：インフラを迅速かつ継続的に使いたい
- ソフトウェア視点：継続的な開発･運用を続けたい

3 Docker導入時の考慮点

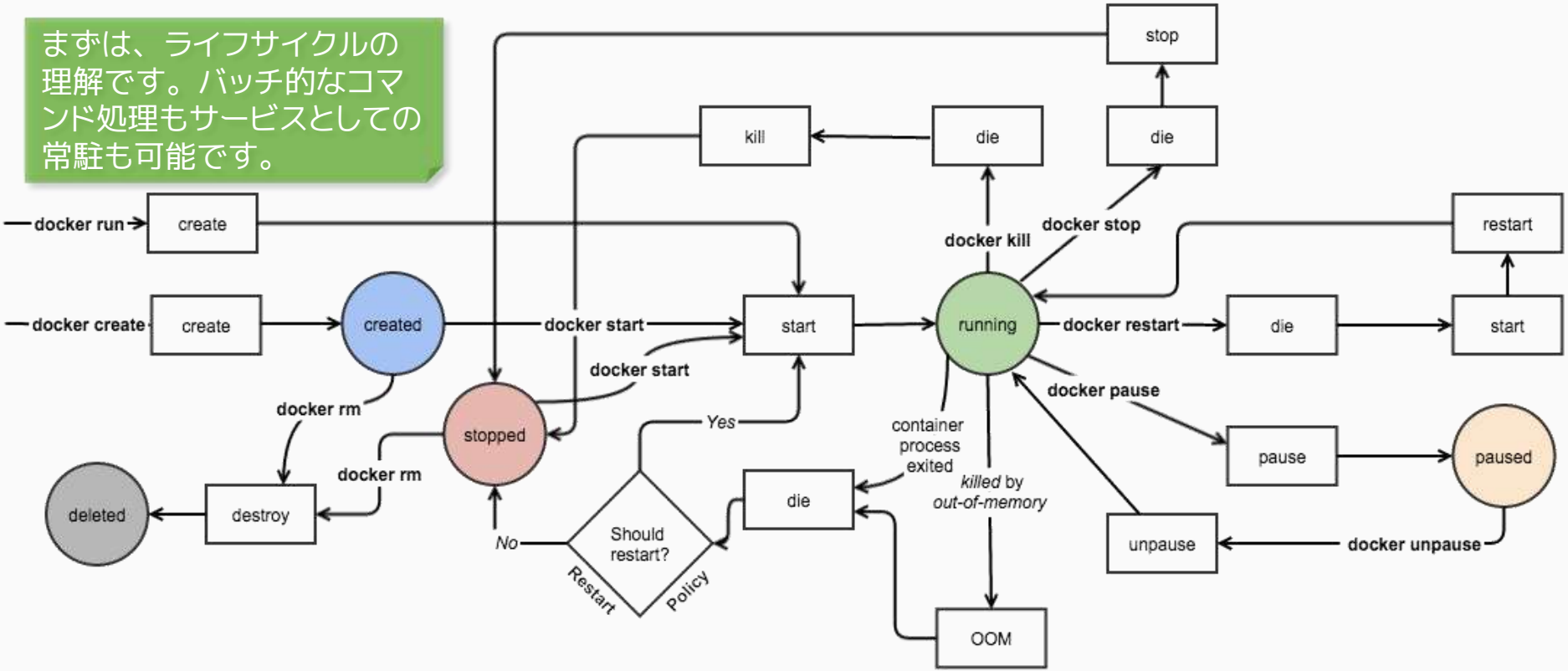
■■■□ Consideration

？？？「すべてがコンテナになる」


顧客が説明した要件
プロジェクトリーダーの理解
アナリストの設計
プログラマのコード
営業の表現、約束
プロジェクトの書類
実際の運用
顧客への請求金額
得られたサポート
顧客が本当に必要だったもの

Dockerコンテナのライフサイクル

まずは、ライフサイクルの理解です。バッチ的なコマンド処理もサービスとしての常駐も可能です。



※ Docker Docs より引用：https://docs.docker.com/engine/reference/api/docker_remote_api/

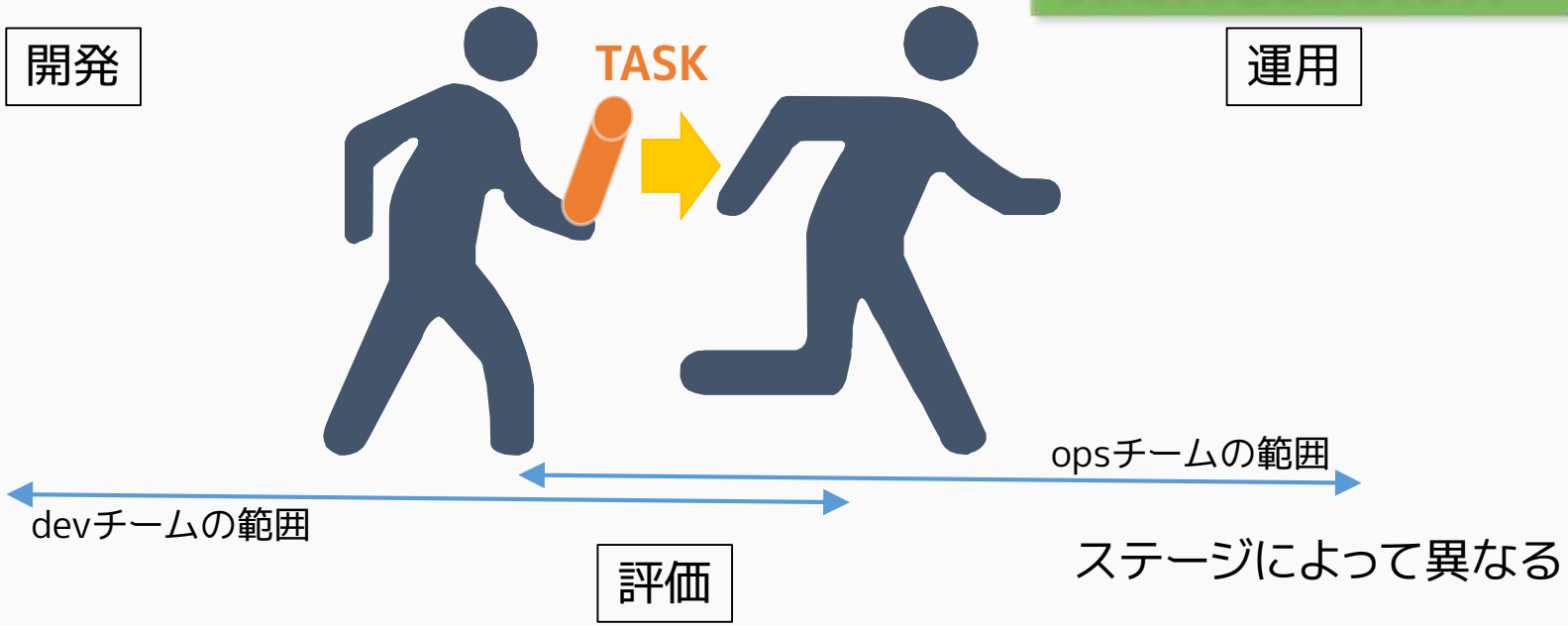
利用シーンの範囲を検討

- 開発環境のみ
- 開発環境 ～ CI / テスト環境
- 開発環境 ～ 本番環境

Dockerの基本的ライフサイクルやコマンドの使い方などの理解は前提として、次に、どのような場面でDockerを使うのかで視点が変わってきます。

利用シーン

例えば開発チームと運用チームが分かれているかもしれません。駅伝やリレーのように、各チームだけでは完結しません。お互い、歩み寄りが必要になります。



ステージによって異なる

利用シーン

あるいは、自分一人（または、特定のチーム内だけ）で全てが完結する可能性も有りますし・・・

開発

運用

評価

ステージによって異なる

利用シーン

工程ごとに様々な人がいる場合も考えられます。

ステージによって異なる

検討項目

1. 利用者は誰で、用途は何か？
2. インフラをどうするのか？
3. OSを何にするのか？
4. ディストリビューションを何にするのか？
5. ストレージ･ドライバを何にするのか？
6. ベースイメージは何にするのか？
7. セキュリティをどのように考慮すべきか？
8. 性能をどのように担保するのか？

★重要な警告★

本章に含まれる以降の検討要素は私個人の見解であり頻繁にいただく質問に対する一般的見解をまとめたものです。所属会社およびDocker, Inc. の意見を代表するものではありませ。また、Docker 1.10 現在の情報です。今後の Docker のバージョンによっては判断材料として適切ではない場合が有り得ます。常に、最新情報をご確認願います。

技術選択にあたっての基本ポイント

まず始めに誰が何のために使うのか？という視点が必須です。

- **コンテナを誰が使うのか？**

  開発チーム？　運用チーム？　テスト？　検証？　コミュニティ？

  自社内？　お客さま？

- **何のために使うのか？**

  システム開発？　運用？　商用サービス？　検証？


たとえばピザで考えてみます！

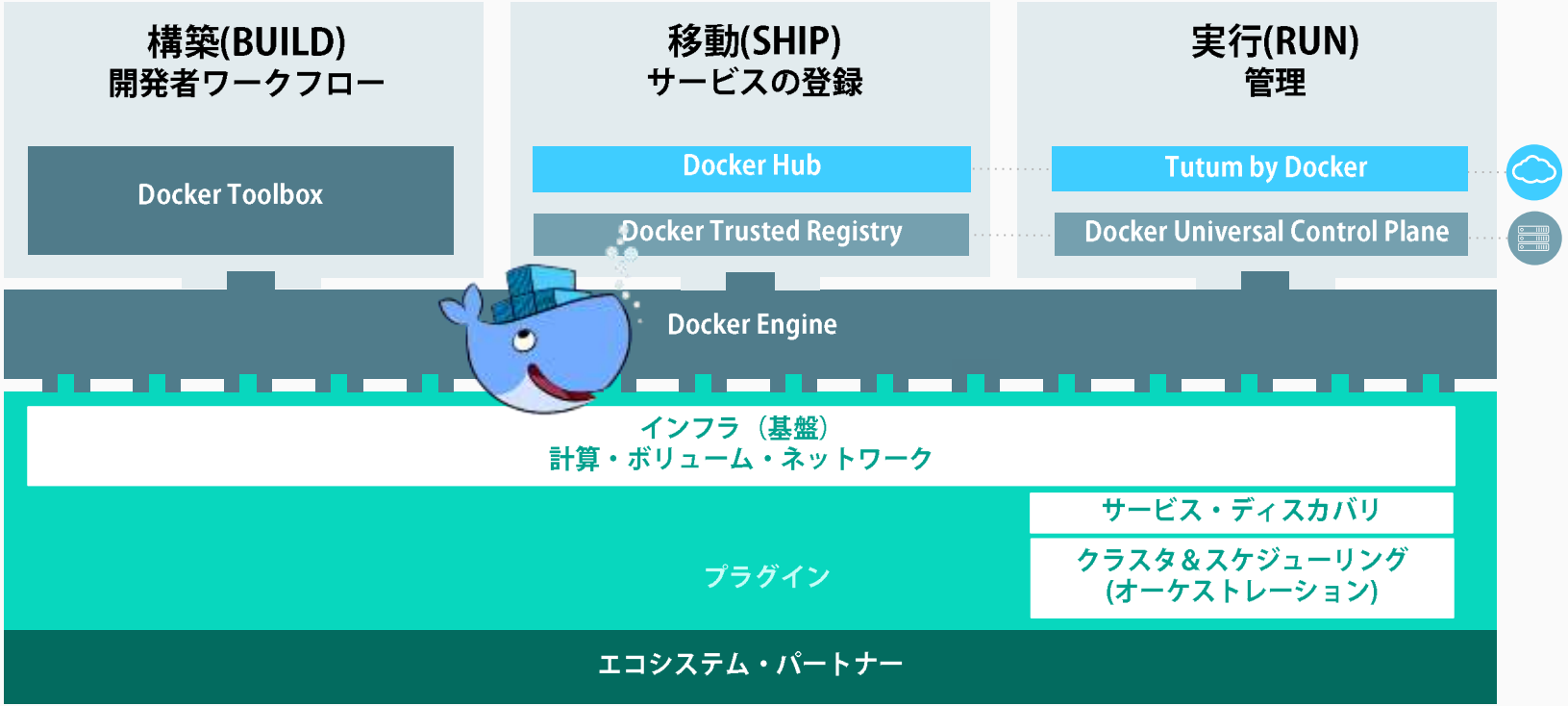
例えば冷凍ピザ！自分がお腹空いた、とにかくピザを今すぐ食べたい！というのであれば冷凍ピザでも良いですよね。料理の腕前に覚えがなくとも、電子レンジですぐにできます。簡単です。味もソコソコでしょう。これは、Dockerコンテナの使い勝手と似ている所があると思います。


しかしお客さまに商品としてピザを出す（サービス提供する）のであれば、Dockerを使うべきでないシーンがあるかもしれません。

ちゃんとしたモノが必要であれば、それなりに手間暇をかけるべき場合も考えられます。改めて、誰が何のために使うのか？を考慮するのは非常に重要だと考えています。

特に、仕事で使うのであれば、最終的にピザを食べるのは誰か＝Dockerコンテナを使用･運用するのは誰かという視点は欠かせないものでしょう。

利用者や目的が定まれば、インフラに何を使うべきかは自ずと決まると思われます。利用シーンであったり、環境によって異なるでしょう。どれが優れているか優れていないかは、個々のユースケースに依存します。



Linuxディストリビューションの選択

- **Docker Engine 商用サポートの有無**
  - Docker Commercial Suport （商用サポート）版対応（2015年2月現在）
    - Red Hat Enterprise Linux 7.0 , 7.1
    - CentOS 7.1
    - Ubuntu 14.04 LTX
  - 最新リストは https://www.docker.com/compatibility-maintenance
- **ベンダーによる保守サポート有無**
- **コミュニティによるサポートの有無**

ストレージ・ドライバ

ストレージ・ドライバを何にするかはDocker独特の判断ポイントです。こちらもどのような用途に使うのか、あるい経験を持っているかによって異なります。

- **Docker イメージの内部管理方式**
  - aufs, Devicemapper, btrfs, overlayfs, zfs …

  ➡ **全てのユースケースを満たすものは存在しないため、特性に応じて使い分ける必要がある**

- **Dockerのドキュメントでは**
  - これまで使ったことがあるドライバを優先すべき（経験重視）
  - 次に、ディストリビューションごとの標準ドライバ（コミュニティがサポート）
  - なお、CS 版 (Commercial Support) 版 Dockerは（商用／ベンダのサポート重視）
    - RHEL7.0, 7.1, Ubuntu 14.04 LTS, CentOS 7.1　※2016年2月現在
    - https://www.docker.com/compatibility-maintenance

https://docs.docker.com/engine/userguide/storagedriver/selectadriver/

AUFS 安定 プロダクション対応 メモリ使用量が適切 Docker利用がスムーズ 重たい書き込み PaaS風の動作

Devicemapper (loop) 安定 カーネルの主流 Docker利用がスムーズ プロダクション 性能 調査用のテスト

Devicemapper (direct-lvm) 安定 プロダクション対応 カーネルの主流 Docker利用がスムーズ PaaS風の動作

Btrfs カーネルの主流 重たい書き込み コンテナの撹拌（激しい利用） ビルド・ツール

Overlay 安定 メモリ使用量が適切 カーネルの主流 コンテナの撹拌（激しい利用） 調査用のテスト

ZFS native (Zol) PaaS風の動作

ZFS FUSE 安定 調査用のテスト プロダクション

ポイント

特徴がある 特徴

ユースケースに相応しい ユースケース

ユースケースに適しない ユースケース

一般的な指針として、このような区分はされていますが、鵜呑みはしないほうが良いと思います。扱うファイルサイズだったりOSやハードウェアにも影響を受けます。なので個々の環境におけるベンチマークは必要と思います。

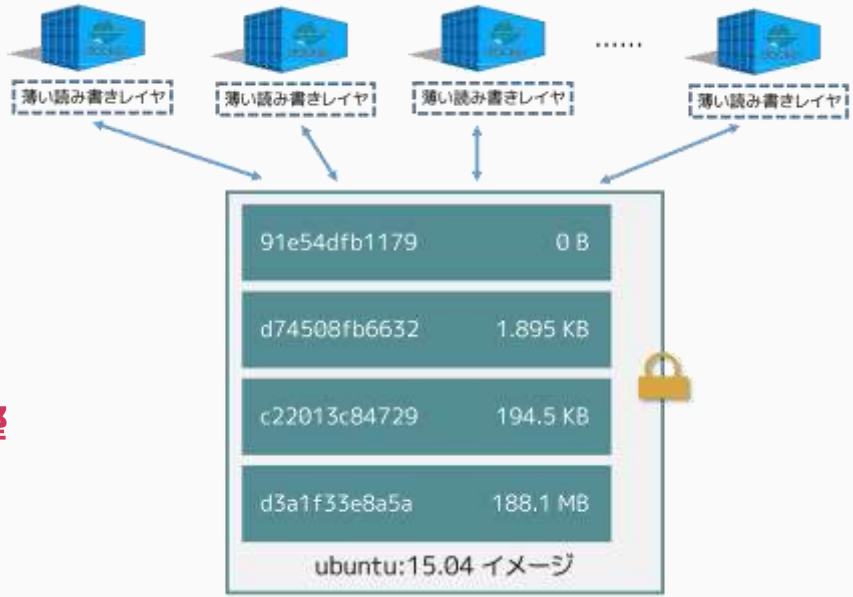
コピー・オン・ライト方式

なぜならDockerはイメージ管理に独特の処理を行っているため。

- **CoW; Copy on Write**
  - 効率的にイメージを使う仕組み
    1. ファイル更新が発生する（初回）
    2. イメージからコンテナ用領域にファイルをコピーする
    3. コピーしたファイルを編集

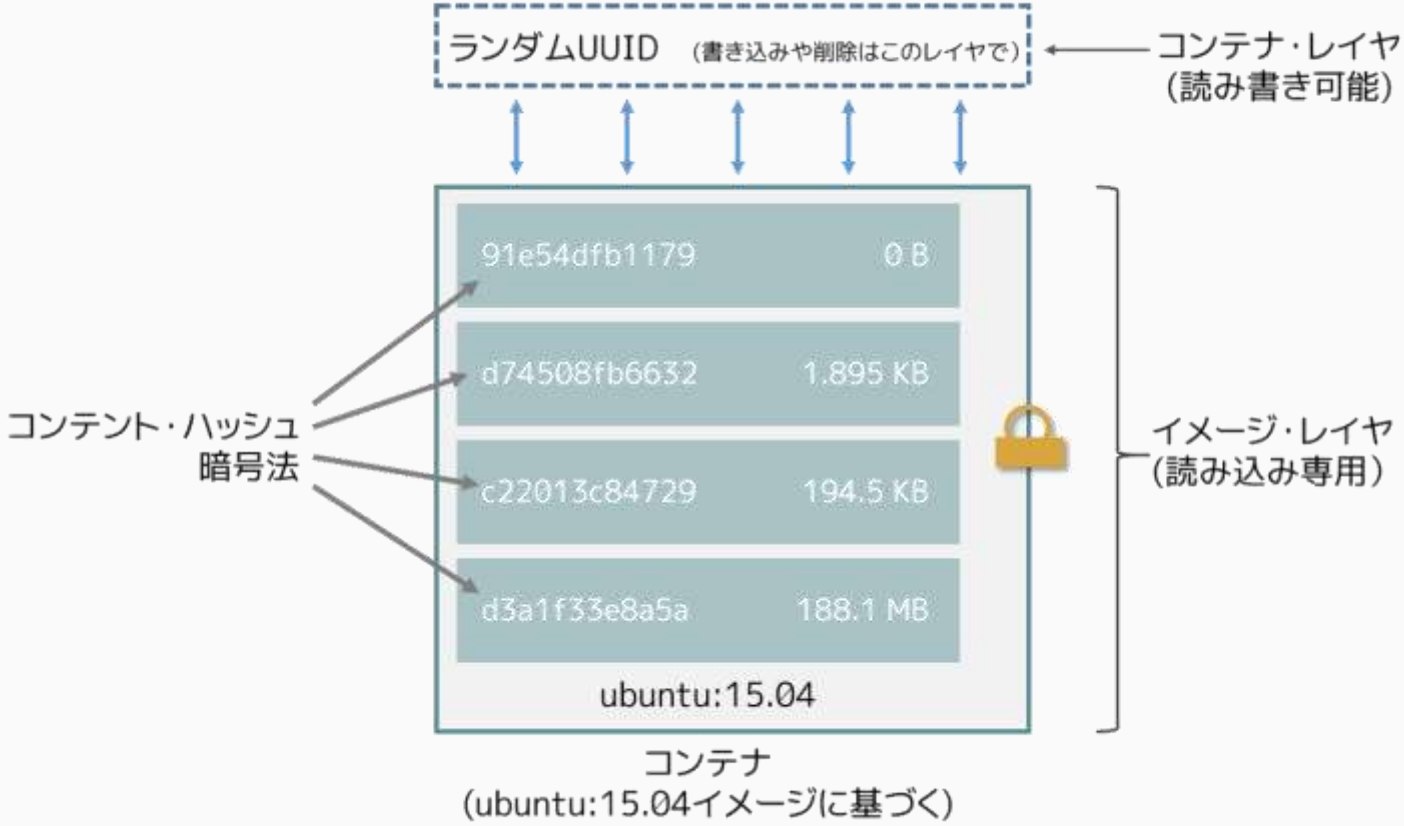
➡ **CoW処理がパフォーマンスに影響**

➡ **デバイス・ドライバの選択により挙動が異なるので注意が必要**



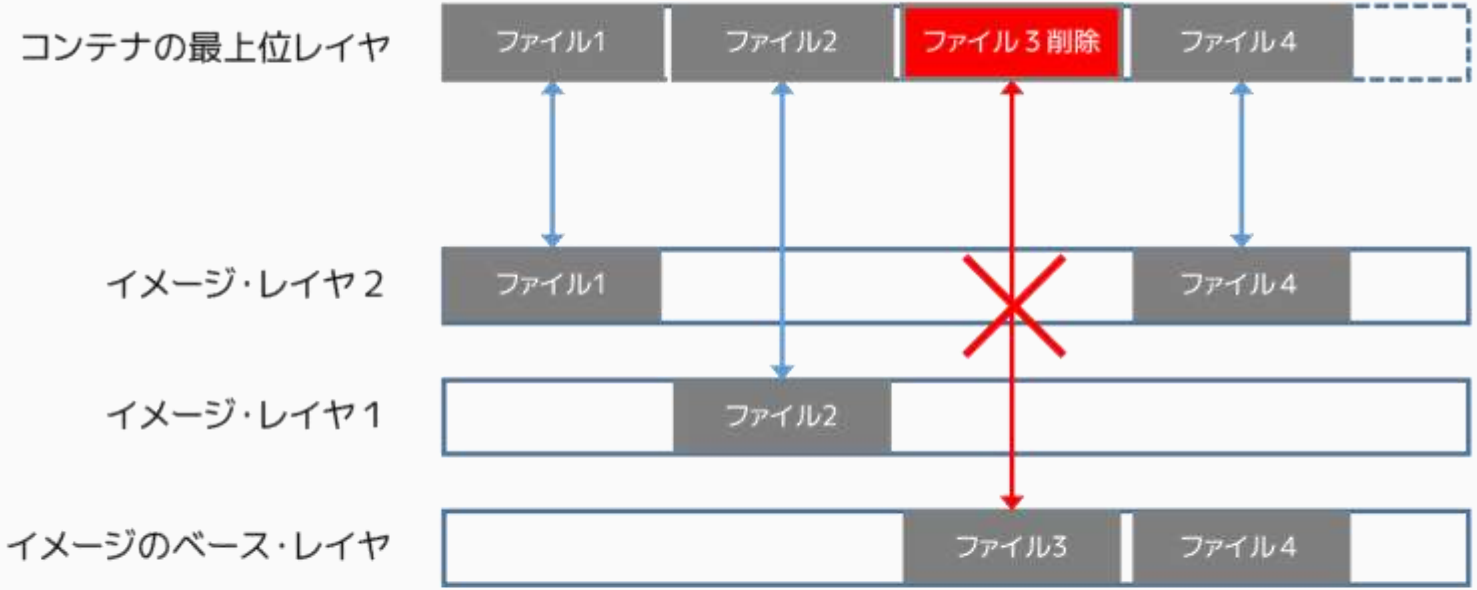
詳細：イメージ、コンテナ、ストレージ・ドライバの理解 — Docker-docs-ja 1.10.0b ドキュメント
http://docs.docker.jp/engine/userguide/storagedriver/imagesandcontainers.html

こちらはレイヤを抽象化したもの。



AUFS

AUFSはコンテナに書き込みを行う時、初回にコピーが発生します。1GBのファイルに追記する場合、1GBのファイルをコピー後に処理が進行します。

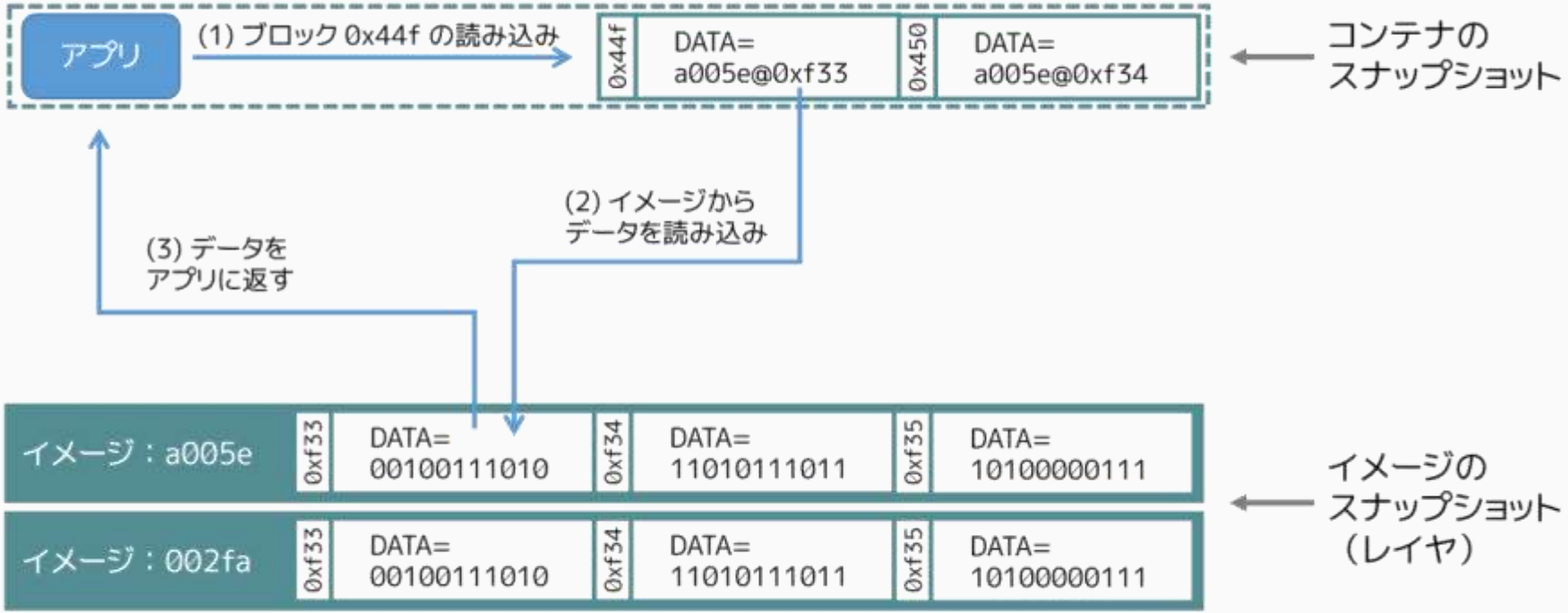


Docker コンテナ
(AUFS ストレージ･ドライバのホワイトアウト･ファイルの説明用)

AUFS ストレージ･ドライバを使う — Docker-docs-ja 1.10.0b ドキュメント
http://docs.docker.jp/engine/userguide/storagedriver/aufs-driver.html

devicemapper

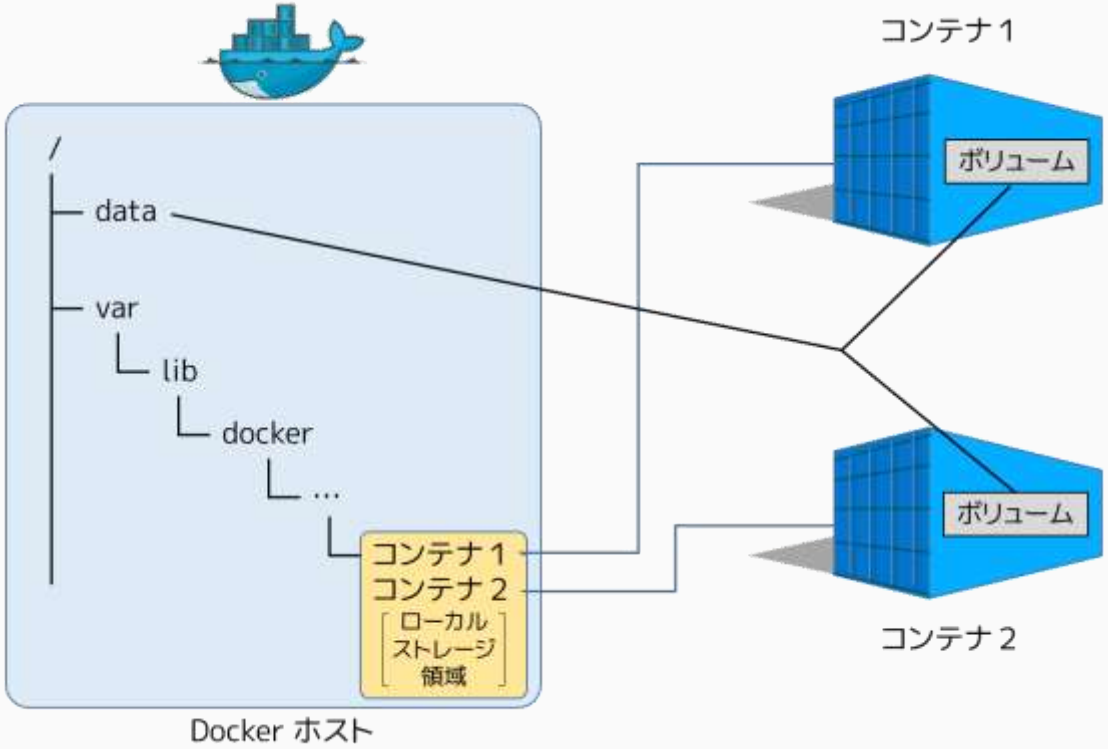
一方、RHEL系の場合はコピーは発生しません。64KB単位で処理が行われます。



Device Mapper ストレージ・ドライバを使う — Docker-docs-ja 1.10.0b ドキュメント
http://docs.docker.jp/engine/userguide/storagedriver/device-mapper-driver.html

コンテナのボリューム

ボリュームはCoW処理を行わず、
ホスト上のI/O性能が直接反映。



コンテナでデータを管理する — Docker-docs-ja 1.10.0b ドキュメント
http://docs.docker.jp/engine/userguide/containers/dockervolumes.html

主なセキュリティ考慮点

- **ベース・イメージ**
  - 誰がセキュリティのメンテナスを行っているのか。コミュニティ or 独自？
- **Dockerコンテナ**
  - Dockerコンテナの操作には事実上の root ユーザ権限が必要
  - SELinux, AppArmor など OS 側の対応状況
  - ホスト上の UID とコンテナが内部で使う UID が不意に一致する恐れ
    - v1.10 で user namespace をサポートしたことで、 --userns-remap を使った subuid, subgid を活用可能に
  - Docker デーモンの脆弱性発生リスクと対処手段
    - ベンダによるサポートの有無
  - 特権ユーザの処理　http://blog.docker.com/2013/09/docker-can-now-run-within-docker/
  - デバイス・ドライバ毎のバグが許容できるかどうか
    - 参考：aufs/overlayfs/btrfs bugs – Qiita (AkihiroSuda 氏 の投稿)
      http://qiita.com/AkihiroSuda/items/7db89f48d50b98fa1fa8
- **ネットワーク通信**
  - クライアント・デーモン間は TLS 通信が推奨

検討項目まとめ

1. 利用者は誰で、用途は何か？
2. インフラをどうするのか？
3. OSを何にするのか？
4. ディストリビューションを何にするのか？
5. ストレージ･ドライバを何にするのか？
6. ベースイメージは何にするのか？
7. セキュリティをどのように考慮すべきか？
8. 性能をどのように担保するのか？

参考にすべきはドキュメント

- 公式
  - ➡ https://docs.docker.com/
- 日本語訳
  - ➡ http://docs.docker.jp

Docker Engine オススメのドキュメント

- **ユーザガイド**
  - http://docs.docker.jp/engine/userguide/index.html
  - Dockerfile ベスト・プラクティス
    - http://docs.docker.jp/engine/userguide/eng-image/dockerfile_best-practice.html
  - ストレージ・ドライバ
    - http://docs.docker.jp/engine/userguide/storagedriver/index.html
  - Docker ネットワーク機能の概要
    - http://docs.docker.jp/engine/userguide/networking/index.html
- **リファレンス**
  - docker run リファレンス
    - http://docs.docker.jp/engine/reference/run.html
  - コマンドライン・リファレンス
    - コマンドラインで使う http://docs.docker.jp/engine/reference/commandline/cli.html
    - daemon http://docs.docker.jp/engine/reference/commandline/daemon.html
    - run http://docs.docker.jp/engine/reference/commandline/run.html

ここまでのまとめ

- Dockerはプラットフォームであり、**全てを解決しない**
- Dockerコンテナのライフサイクルに対する理解が必要
- 検討要素
  - 最終的な利用者は**誰**なのか
  - 業務範囲や用途は**何**なのか

技術選択のポイント

Dockerは全てを解決するインフラではない。Dockerの管理領域と、それ以外を区別して組み合わせを考える必要がある。

1. 利用者は誰で、用途は何か？
2. インフラをどうするのか？
3. OSを何にするのか？
4. ディストリビューションを何にするのか？
5. ストレージ・ドライバを何にするのか？
6. ベースイメージは何にするのか？
7. セキュリティをどのように考慮すべきか？
8. 性能をどのように担保するのか？

4 まとめ

■■■■ Wrap up

本資料における主張

- 「Docker」を**導入するだけでは価値を生まない**。
- 「Docker」は技術選択要素の１つにすぎないので、
  業務フローに一致するならば、積極的に導入すべきだ。
- **継続的**な技術評価を行うべきだ。

.

誰が？
何のために？

技術選択のポイント

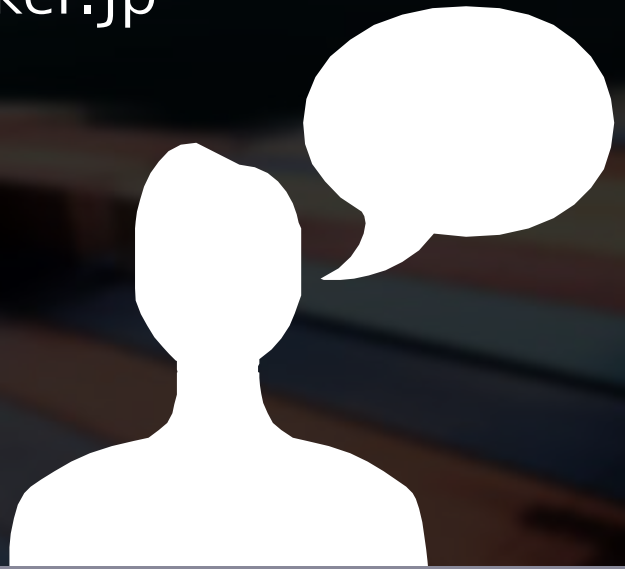
1. 利用者は誰で、用途は何か？
2. インフラをどうするのか？
3. OSを何にするのか？
4. ディストリビューションを何にするのか？
5. ストレージ・ドライバを何にするのか？
6. ベースイメージは何にするのか？
7. セキュリティをどのように考慮すべきか？
8. 性能をどのように担保するのか？

より詳しい技術情報

http://docs.docker.jp
http://slideshare.net/zembutsu

質疑応答

- 何か気になる所はありますか？
- 日本語ドキュメント http://docs.docker.jp
- @zembutsu



私は誰なのか？

Why am I here?

+MasahitoZembutsu

- **@zembutsu a.k.a. 前佛雅人**
  - Technology Evangelist; Creationline, Inc. – 2 yrs
  - Technical Trainer; Docker Authorized Trainer – 0.5 yr
  - Data Center Operations Engineer – 15+ yrs

➡ **興味：運用監視自動化、趣味でOSSやクラウド系の検証・情報発信**

  - SlideShare http://slideshare.net/zembutsu
  - Blog http://pocketstudio.jp/log3