

Webサービスを止めずに 継続的な開発をするためのソフトウェア技術

コンテナ化・テスト自動化・分散型バージョン管理

Sakura Internet, Inc.
技術本部ミドルウェアグループ
Masahito Zembutsu @zembutsu

Nov 19, 2019
ORIST技術セミナー・ビジネスマッチングブログ(BMB)第47回勉強会



@zembutsu
前佛 雅人

 @zembutsu  zembutsu

さくらインターネット株式会社 技術本部
Technology Evangelist / Developer Advocate

- 石狩市への小学校プログラミング教育支援プロジェクト → さくらの学校支援プロジェクト (2019～)
- 仮想化基盤チーム(クラウドチーム・VPSチーム)
- エバンジェリストチーム

何をしたいの?: 普通の人々が当たり前計算資源を活用して、豊かな生活を育めるように支援する

● 略歴

- ・ 1986年 はじめてパソコンに触る
- ・ 1993年 富山高専(電気工学科)で学び始める
パソコン通信・インターネットとの遭遇
- ・ 2000年 富山のIT企業に就職
 - ・ とあるホスティング&ISPで勤務
NOC対応・運用・サポート業務に携わる
- ・ 2007年 東京へ
 - ・ システム運用とか監視をいろいろ、企画
 - ・ 自動化ツール
- ・ 2019年 大阪在住

http://docs.docker.jp/



高品質で持続的な開発・運用を行うため

「コンテナ化」

「バージョン管理」

「テスト自動化」

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

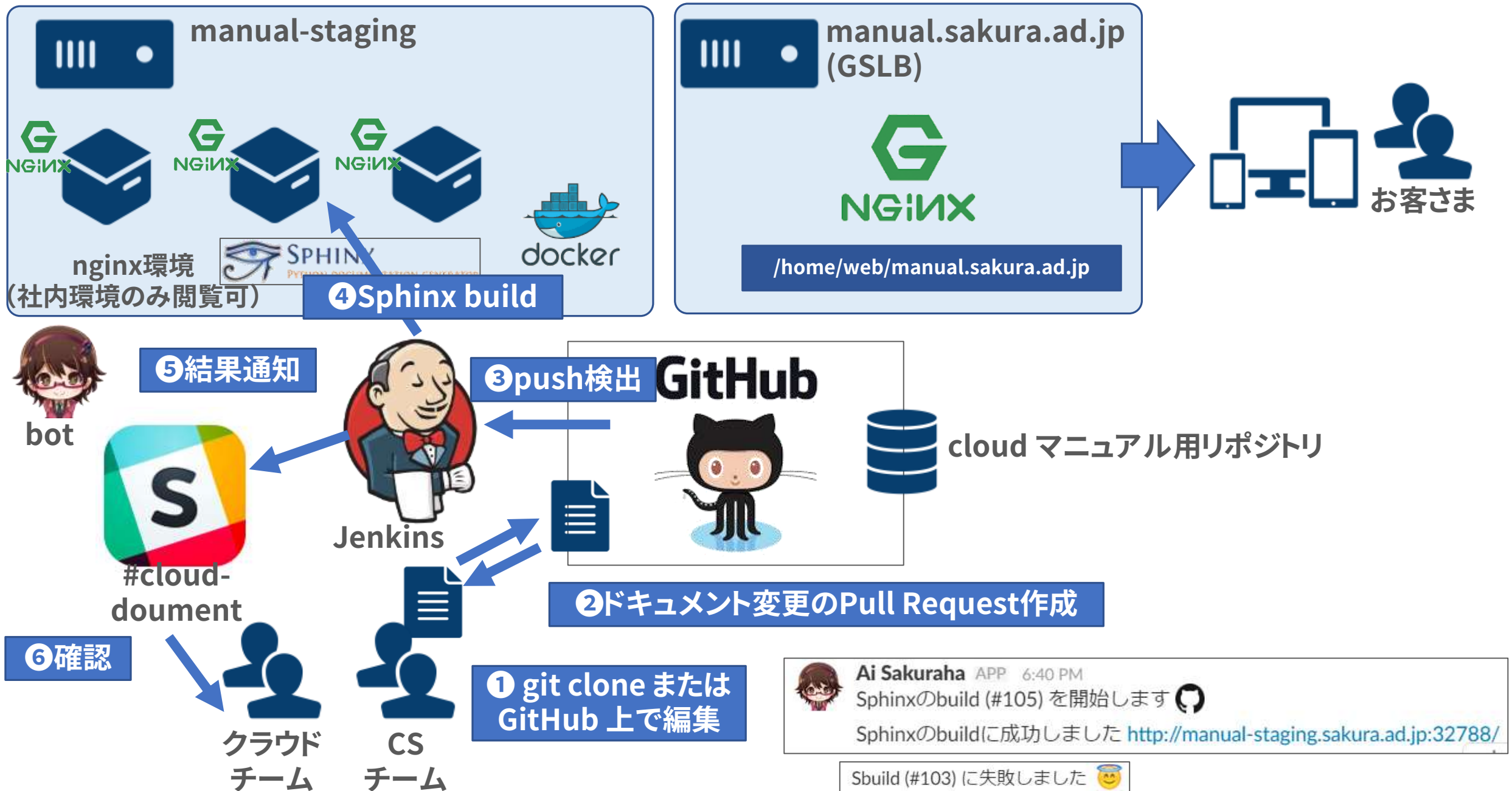
- バージョン管理

- 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

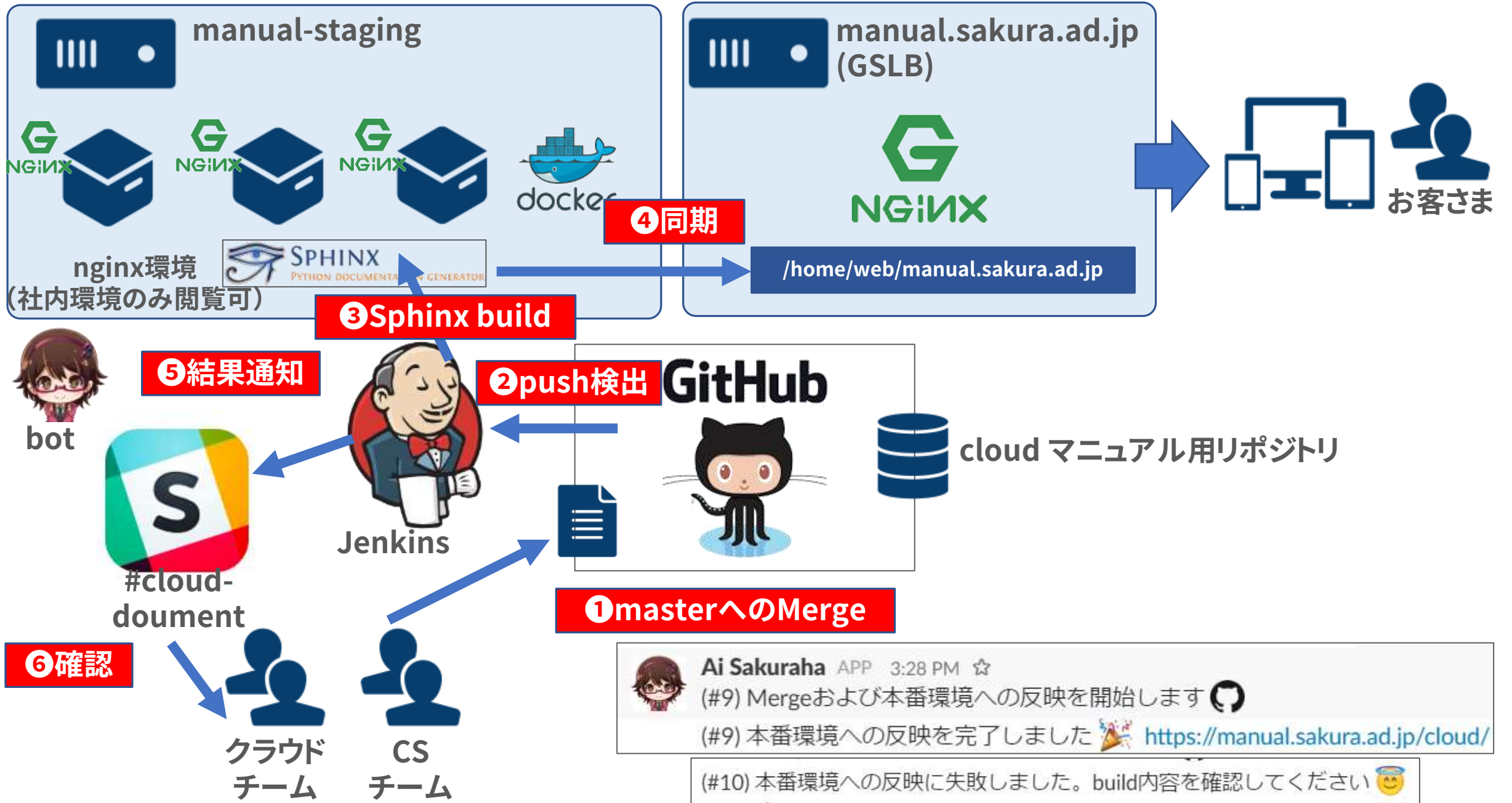
- テスト自動化

- 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

manual.sakura.ad.jp 更新の流れ(PR)

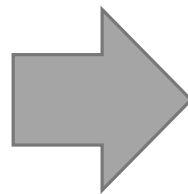


manual.sakura.ad.jp 本番反映の流れ(Merge)



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。



VPSSとクラウドの違いは？

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じ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

NATIONAL BESTSELLER

THE Nicholas Carr
AUTHOR OF *THE SHALLOWS*
BIG SWITCH

"COMPULSIVELY READABLE." —*FAST COMPANY*



REWIRING THE WORLD
**FROM EDISON
TO GOOGLE**

THE DEFINITIVE GUIDE TO THE
CLOUD COMPUTING REVOLUTION

WITH A NEW AFTERWORD

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



- ・電力網を通じた家電の普及・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じ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

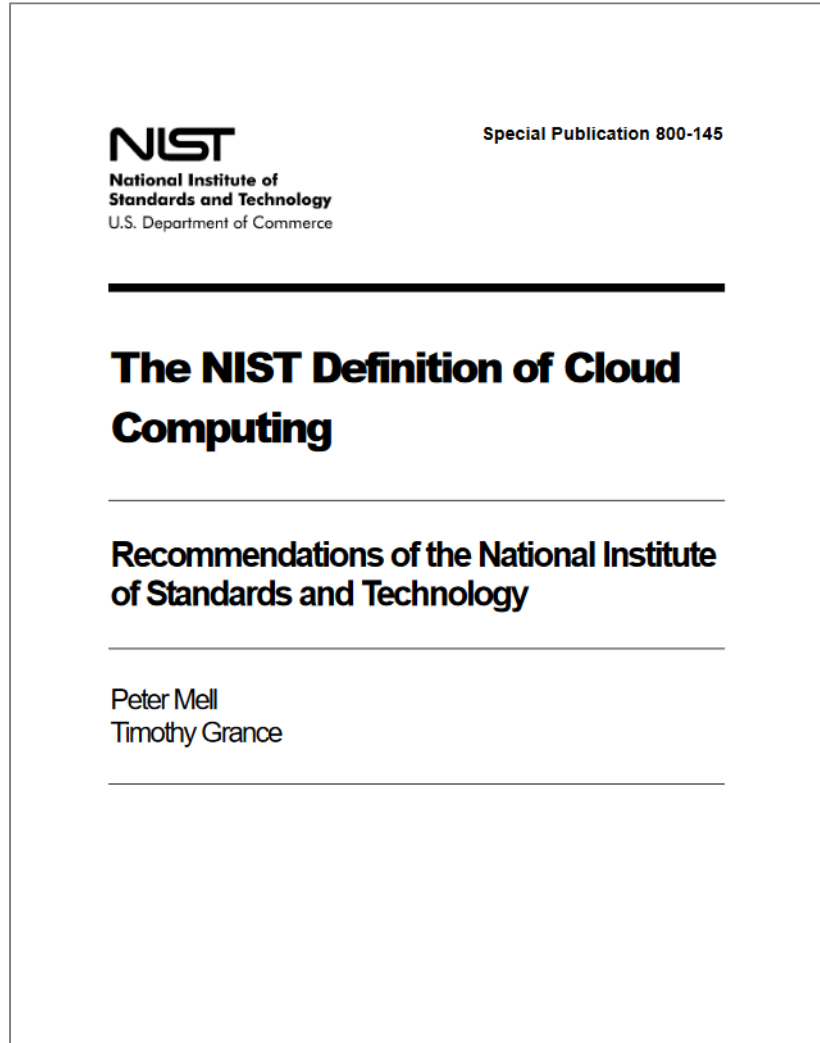


- 個々のマシンやデータセンタ上のサーバにデータを持ち、システムを動かす時代の終焉を予測
- ただし、1つのマシン(One Machine)に集約されるのではなく、分散した

<http://www.rough.type.com/?p=8314>

クラウドの定義

SP 800-145, The NIST Definition of Cloud Computing



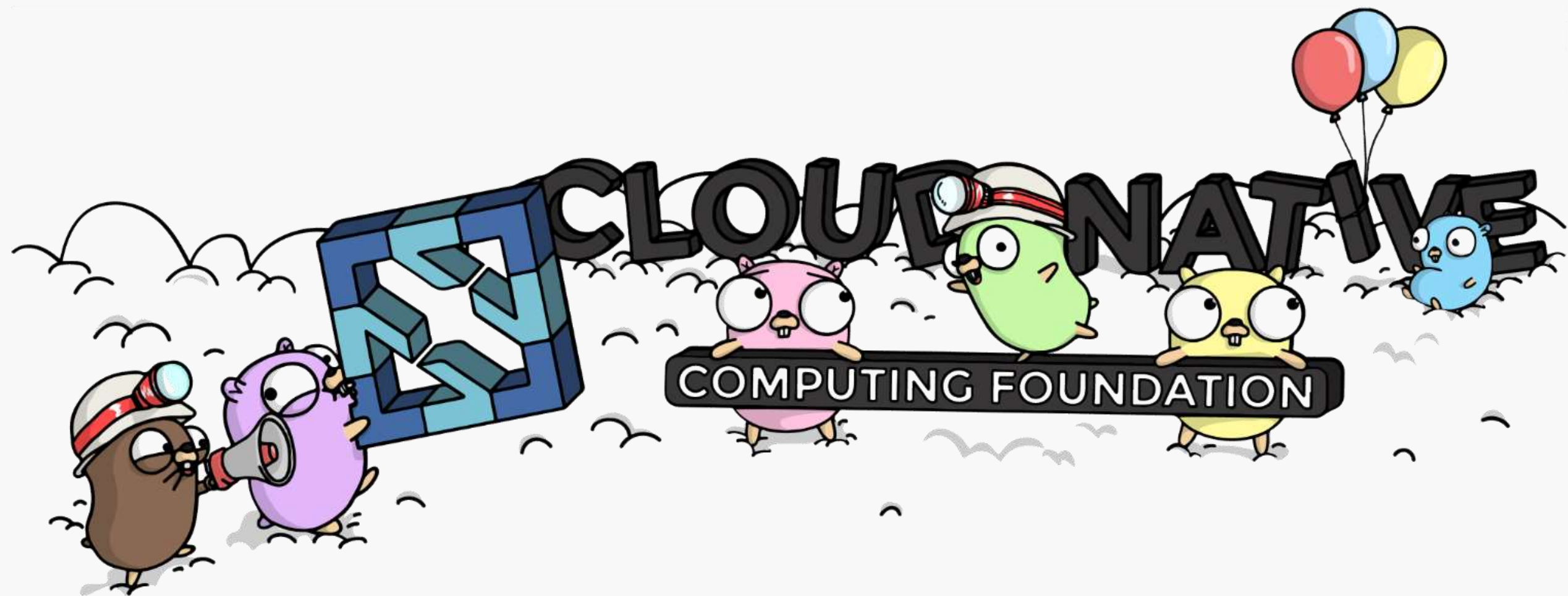
- 2011年9月に最終版を策定
- 本質的特質
 - On-Demand self-service
 - Broad network access
 - Resource pooling
 - Rapid elasticity
 - Measured service
- サービス・モデル
 - Software as a Service (SaaS)
 - Platform as a Service (PaaS)
 - Infrastructure as a Service (IaaS)

クラウド VPS

ここで質問：違いをご存じですか？

From “Cloud Computing”,
To “Cloud Native”

クラウド・ネイティブの時代



仮想化からクラウド・ネイティブへ

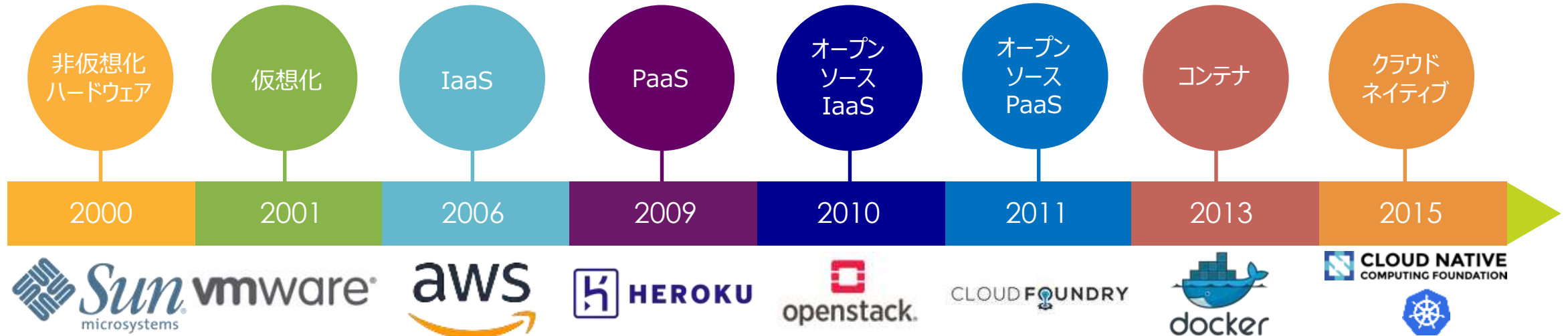
From Virtualization to Cloud Native



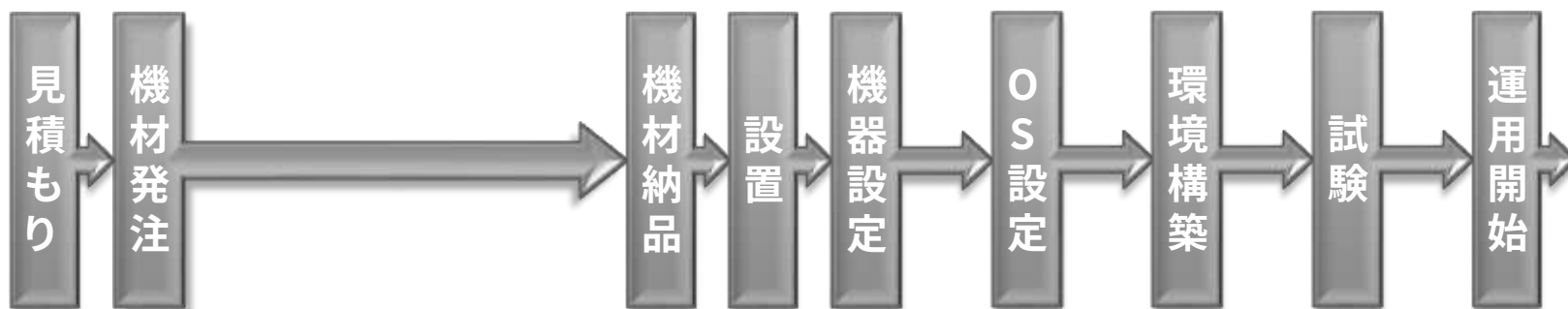
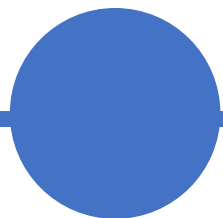
kubernetes

・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：

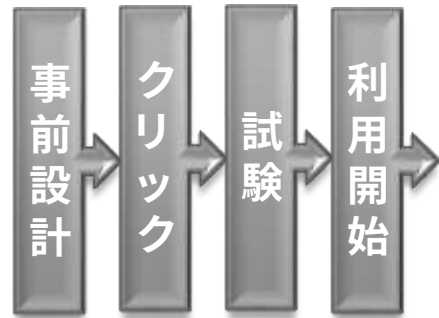
- アプリケーションをマイクロサービス(*microservices*)に分割し、
- 各パーツ自身をコンテナにパッケージし、
- リソース利用を最適化するために、動的に統合/オーケストレーション(*orchestrate*)する

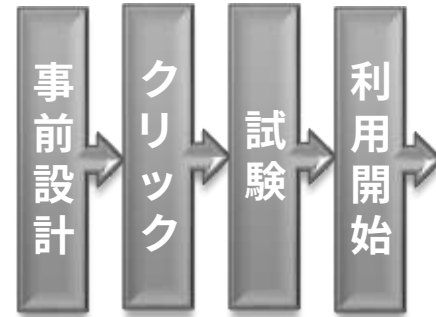
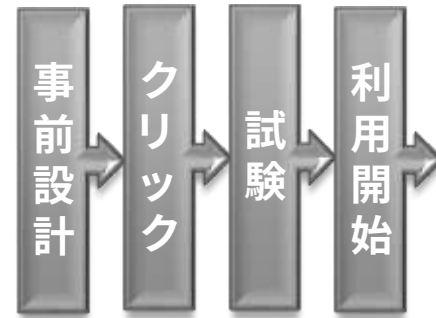
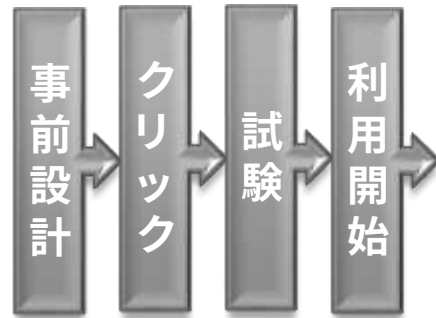
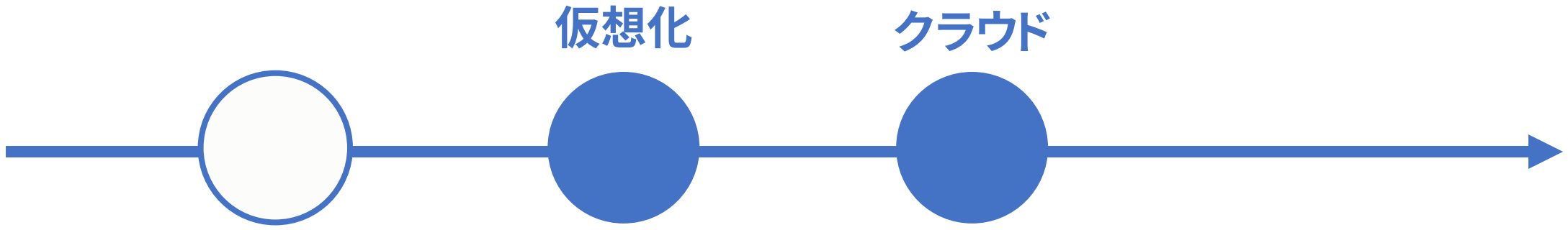


物理サーバ時代



仮想化





“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



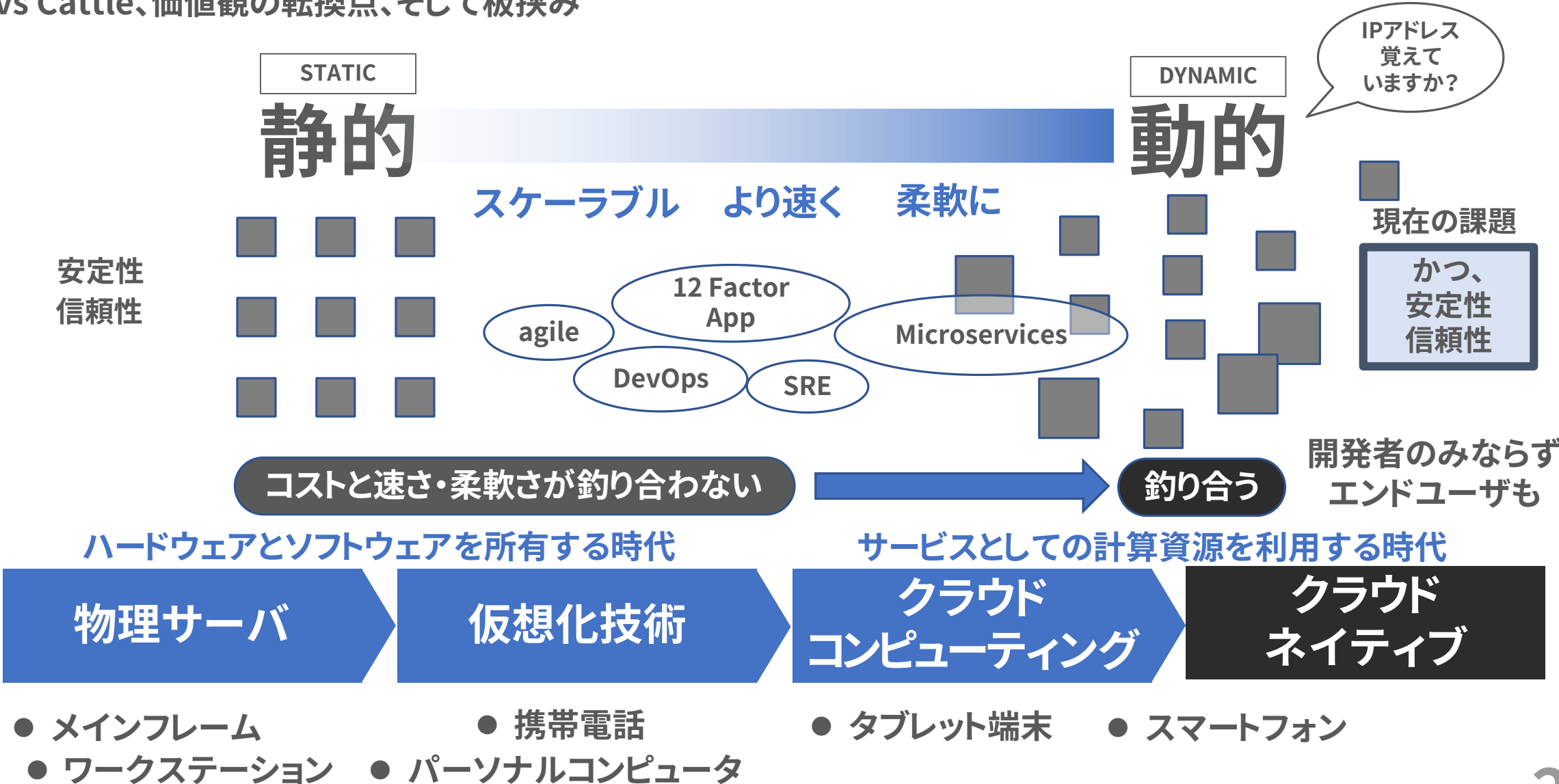
attrib: Bill Baker, Distinguished Engineer, Microsoft
* added by yours truly ...

19

cloudscaling

システム基盤も、サービス側も変わり続ける前提

Pet vs Cattle、価値観の転換点、そして板挟み



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部分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

CLOUD NATIVE TRAIL MAP

The Cloud Native Landscape Lollipop has a large number of options. This Cloud Native Trail Map is a recommended series for leveraging open source, cloud-native technologies. At each step, you can choose a vendor-supported offering or do it yourself, and everything after step #3 is optional based on your circumstances.

HELP ALONG THE WAY

A. Training and Certification

Consider taking offerings from CNCF and then take the exam to become a Certified Kubernetes Administrator or a Certified Kubernetes Application Developer.

cncf.io/training

B. Consulting Help

If you want assistance with Kubernetes and the surrounding ecosystem, consider engaging a Kubernetes Certified Service Provider.

cncf.io/csp

C. Join CNCF's End User Community

For companies that don't offer cloud-native services externally.

cncf.io/enduser

WHAT IS CLOUD NATIVE?

Cloud-native technologies, such as containers and microservices, empower organizations to develop and deploy scalable, agile applications and services in dynamic, distributed environments. By taking into account these characteristics, such systems are designed to be resilient, elastic, and loosely coupled, via manageable abstractions and declarative APIs, thereby enabling effective, reliable automation. This allows engineers to observe the applications and to safely make impactful changes, and results in processes and workflows that fully take advantage of these environments and workflow tool.

The Cloud Native Computing Foundation seeks to drive adoption of these techniques by fostering an ecosystem of open-source, vendor-neutral projects that align with these objectives, and which are portable to public, private, and hybrid clouds. We disseminate the state-of-the-art patterns and practices to ensure innovations remain open and accessible for everyone.

l.cncf.io

091180425



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 out whole applications and writing future functionality as microservices.

3. ORCHESTRATION

- Kubernetes is the market-leading orchestration solution.
- You should select a Certified Kubernetes Distribution, Hosted Platform, or Installer.
- cncf.io/kubernetes



5. SERVICE MESH AND DISCOVERY

- CoreDNS is a fast and flexible tool that is useful for service discovery.
- Envoy and Linkerd each enable service mesh and features.
- They offer health checking, routing, and load balancing.



7. DISTRIBUTED DATABASE

When you need more resiliency and scalability than you can get from a single database, Vitess is a good option for running MySQL at scale through sharding.



9. CONTAINER RUNTIME

You can use alternative container runtimes. The most common, all of which are OCI-compliant, are containerd, rkt and CRI-O.



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 rollbacks, roll backs and testing.

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.



6. NETWORKING

To enable more flexible networking, use a CNCF-compliant network project like Calico, Flannel, or Weave Net.



1. コンテナ化

- Docker コンテナで一般的に行う
- あらゆる大きさのアプリケーションと依存関係(エミュレータ上で実行する PDP-11 のコードですら)をコンテナ化できる
- 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

CONTAINERIZATION

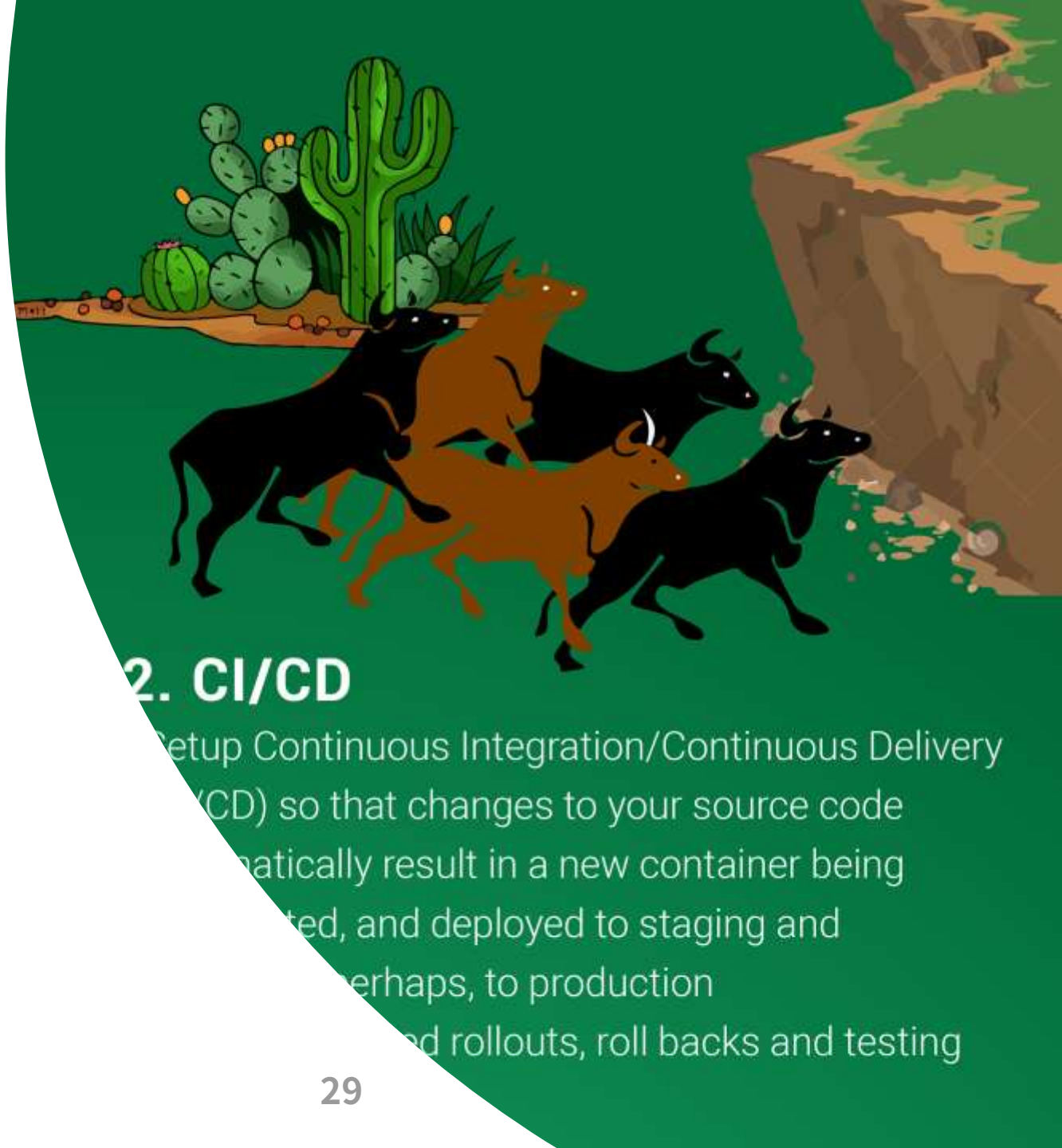
Commonly done with Docker containers

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 microse



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。



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 production environments. Perhaps, to production environments. Automated rollouts, roll backs and testing

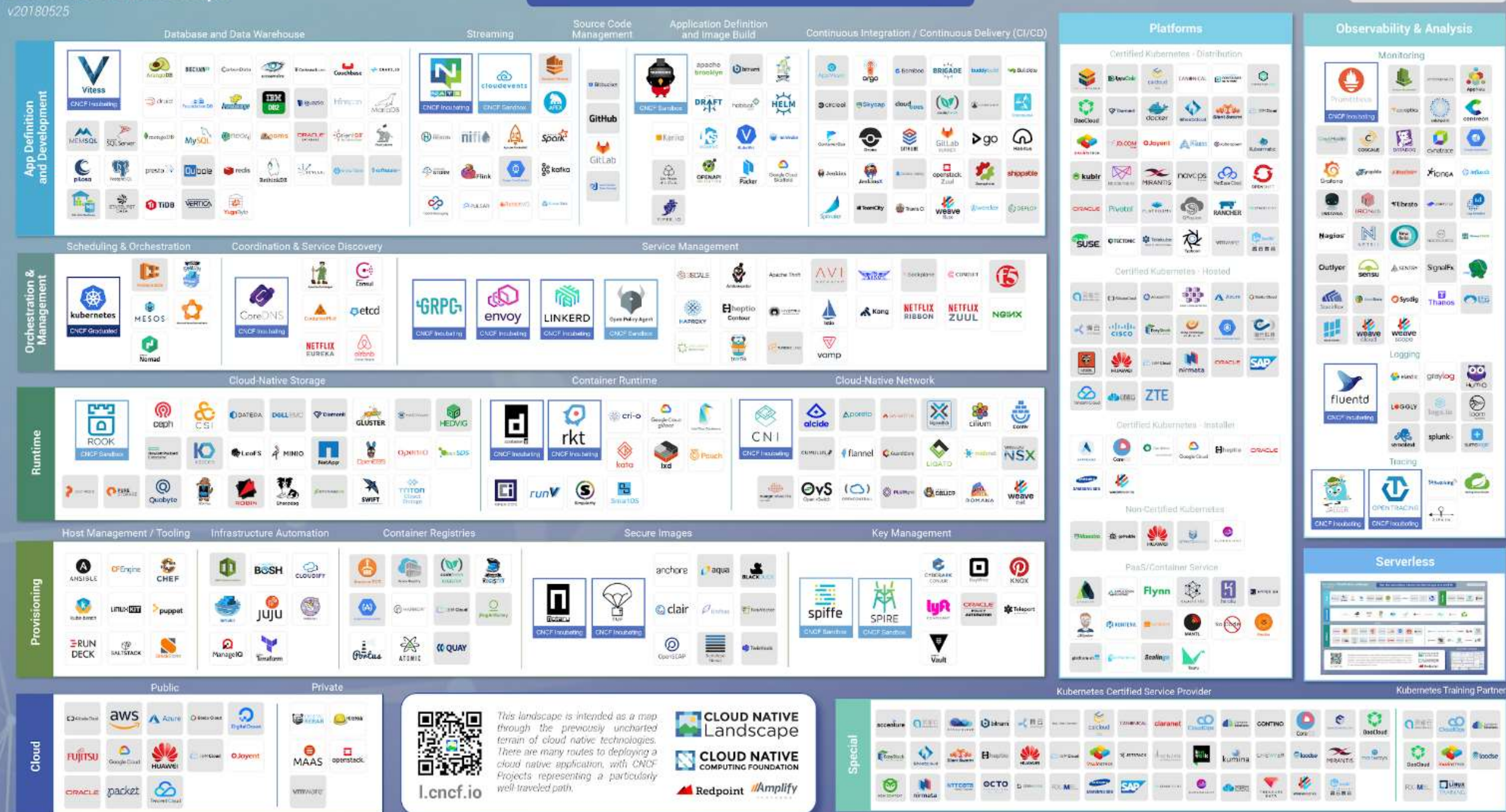
3. オーケストレーション

- 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。

HESTRATION

is the market-leading orchestration
select a Certified Kubernetes Distr
form, or Installer





Cloud Native 参照アーキテクチャ





Why Docker?

Docker の本質的な存在意義

Dockerとは？

What is the Docker?

Containerization Namespace・Cgroup
プロセスを簡単にコンテナ化(isolate)し、
「プロセス・ファイルシステム・ネットワーク・等々」に対して

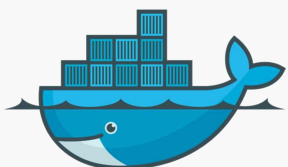
Build Ship Run

簡単かつ素早く開発・移動・実行できるプラットフォームが Docker

1

Dockerコンテナは
実行に必要な全て
をパッケージして、
簡単に動かせる

- アプリケーションを簡単に開発し、移動し、実行するためのプログラムとプラットフォームを提供するのが Docker
- クライアント・サーバ型



container



<https://docker.com>

2

Dockerイメージは
複数のイメージ・レイ
ヤとメタ情報の積み
重なり

- イメージ・レイヤ (image layer) は読み込み専用
- 親子関係がある
- イメージに対する変更は Copy on Write (CoW) 処理が走る
- コンテナ実行にはイメージが必要で、Docker Hub から得られる
- コンテナ実行時のみ、読み書きが可能なレイヤを追加

3

コンテナのプロセス
はデフォルトで
isolate (隔離・分離)
された状態

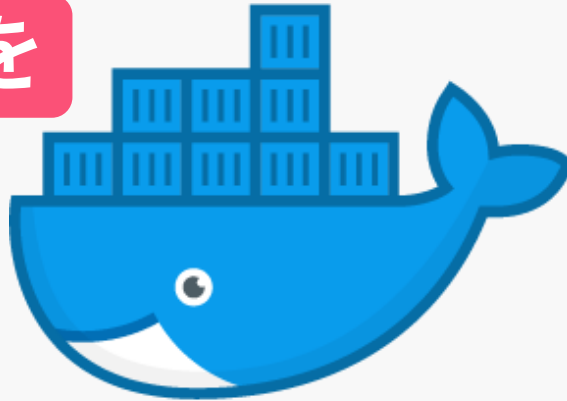
- namespace (名前空間) でプロセス空間やファイルシステムやネットワーク等を分ける技術と、cgroups (コントロール・グループ) でリソースの利用上限を指定
- コンテナはポートをデフォルトで開かない
- ネットワークはブリッジ、ホスト、none の3種類
- ボリュームはコンテナ間でファイルシステムを共有できる。名前付き (named) とホスト・ボリューム

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.”

コンテナ?



コンピュータ

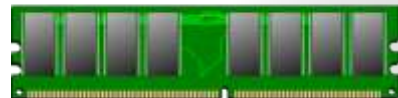
(物理または仮想)



CPU



メモリ



記憶装置



Linux カーネル + システム・ライブラリ(libc)等

オペレーティングシステム (OS)の領域

一般的な
プロセス実行

コンピュータ

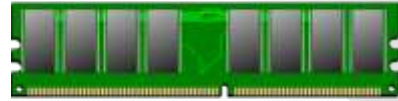
(物理または仮想)



CPU



メモリ



記憶装置



ユーザ空間

プロセス



プロセス



システム空間

Linux カーネル + システム・ライブラリ(libc)等



プログラム



設定ファイル



ソースコード



データ

オペレーティングシステム (OS) の領域

コンテナは
特別な状態の
プロセスのこと

コンピュータ

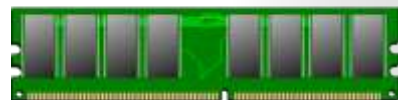
(物理または仮想)



CPU



メモリ



記憶装置



isolate(隔離)

プロセス



isolate(隔離)

プロセス



システム空間

Linux カーネル + システム・ライブラリ(libc)等



プログラム



設定ファイル



ソースコード



データ

オペレーティングシステム (OS) の領域



namespace
“名前空間”技術を使って

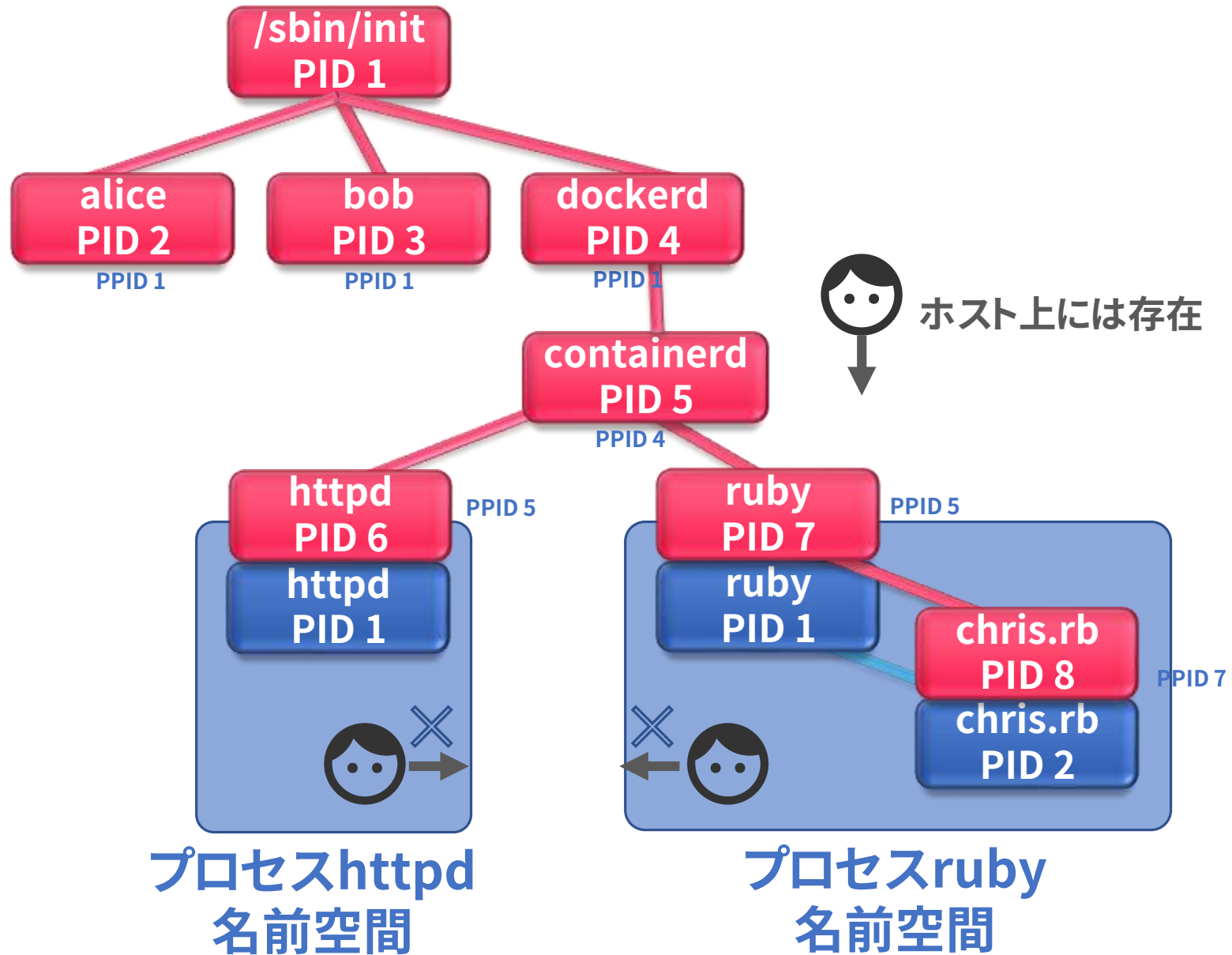
アイソレート
isolate(分離)する

insulatus→isolated→isolate

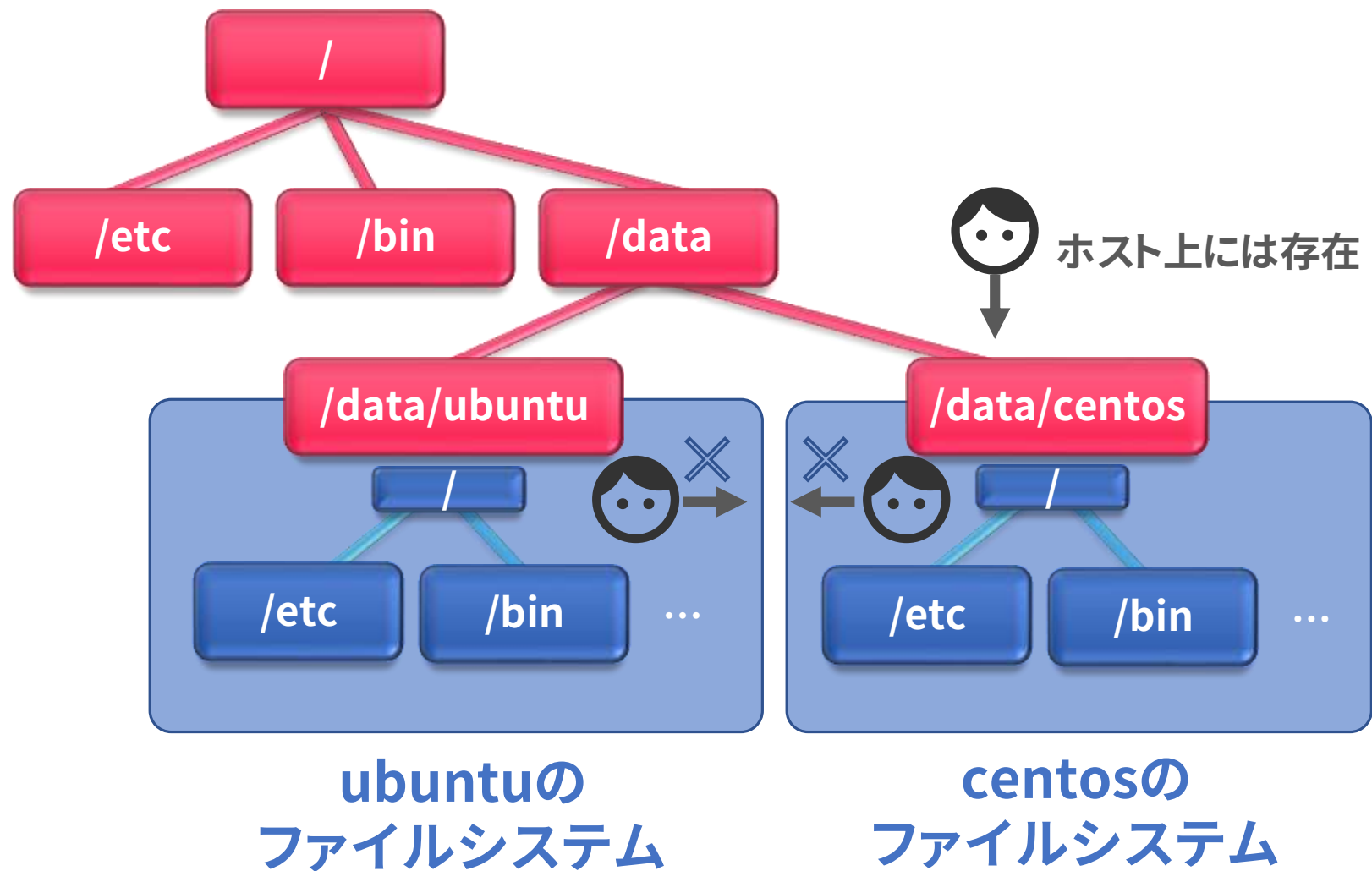
isle island



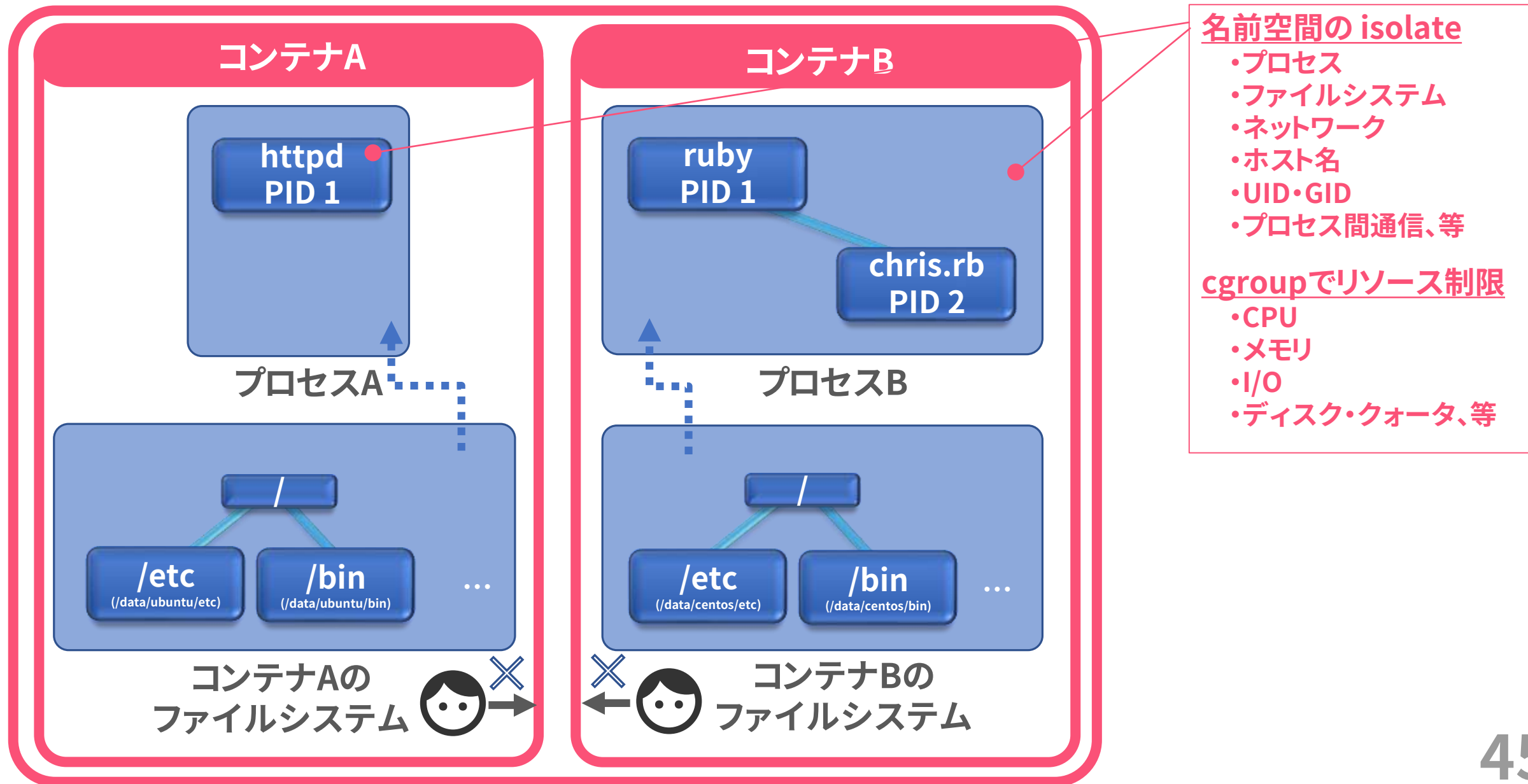
PID名前空間



ファイルシステムを分ける (chroot)

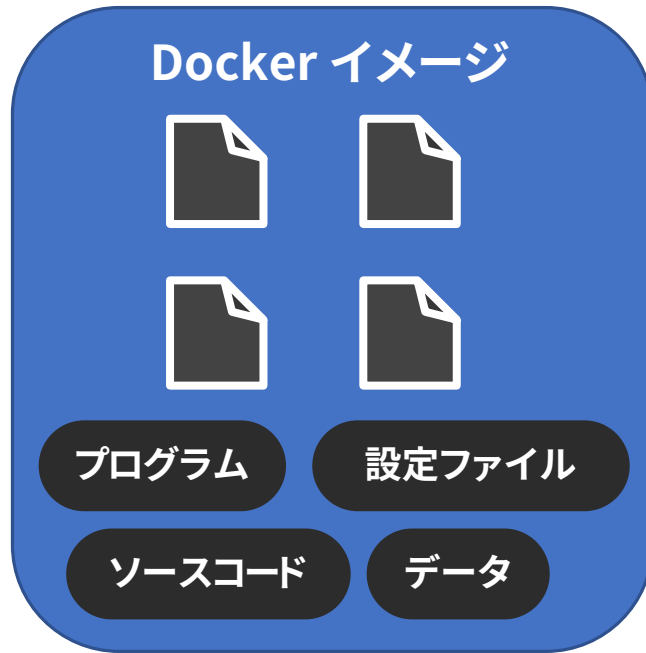


コンテナは特別なプロセスの状態



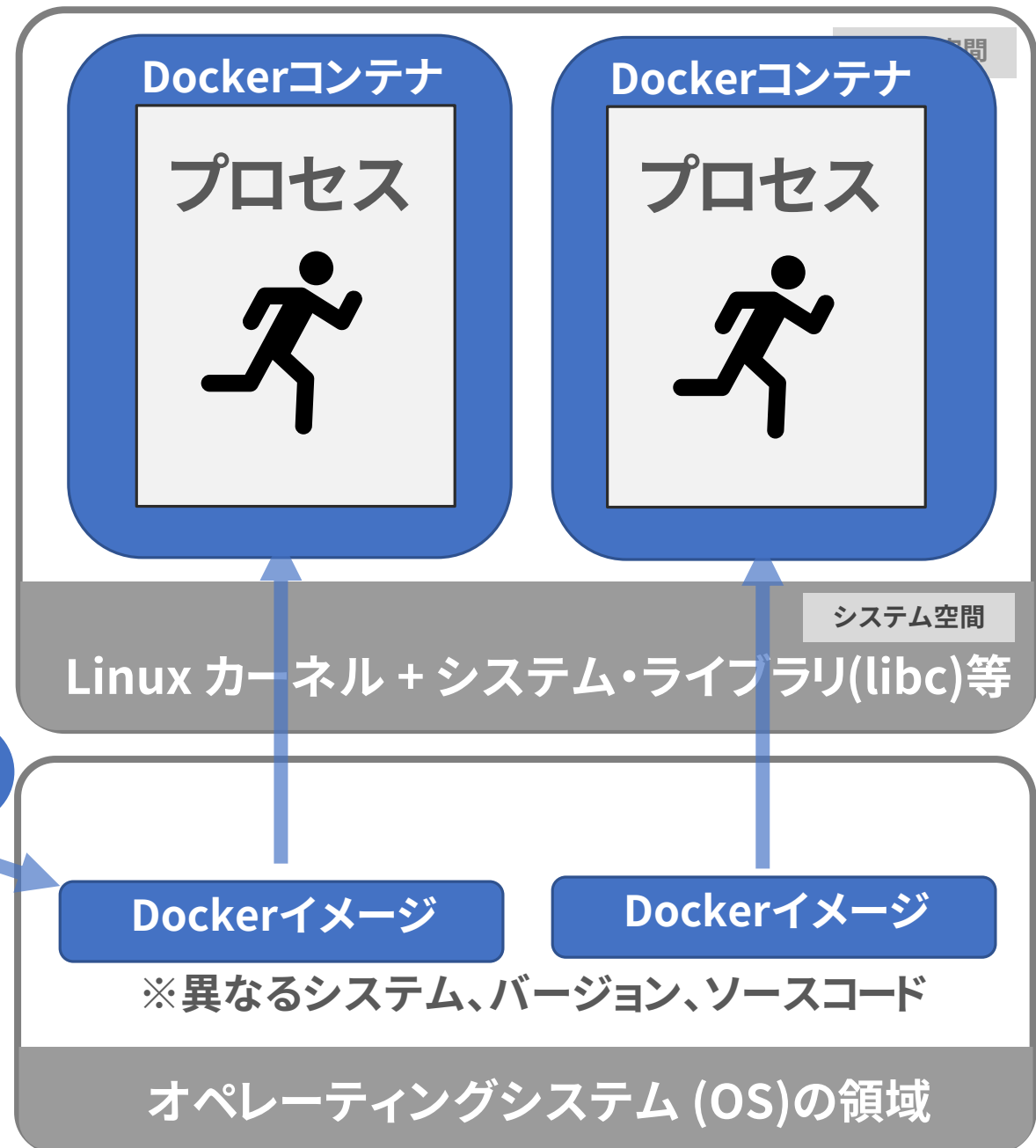
では、 Dockerは？

Dockerイメージを使って
Dockerコンテナ(隔離状態)のプロセスを起動

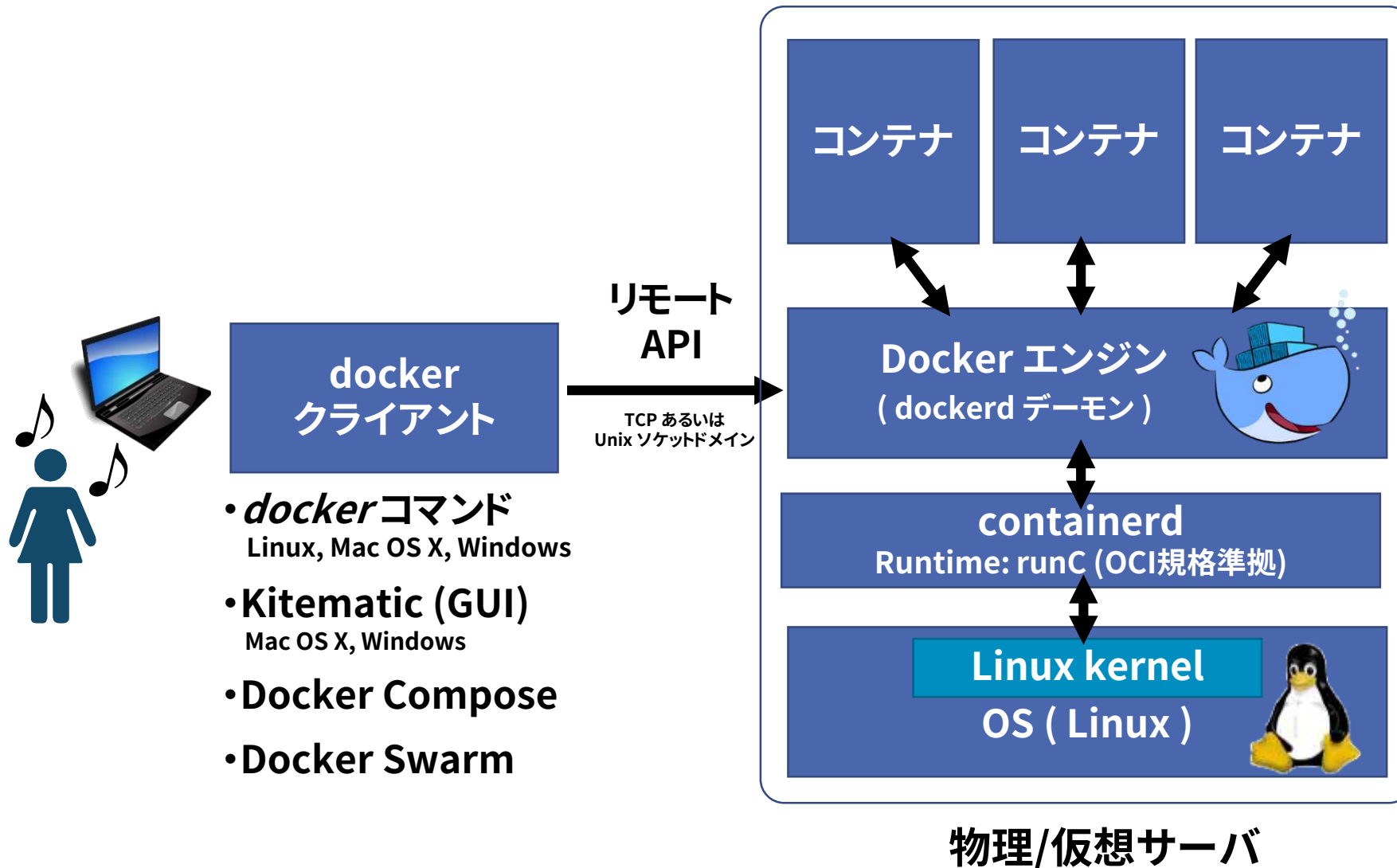


ビルド

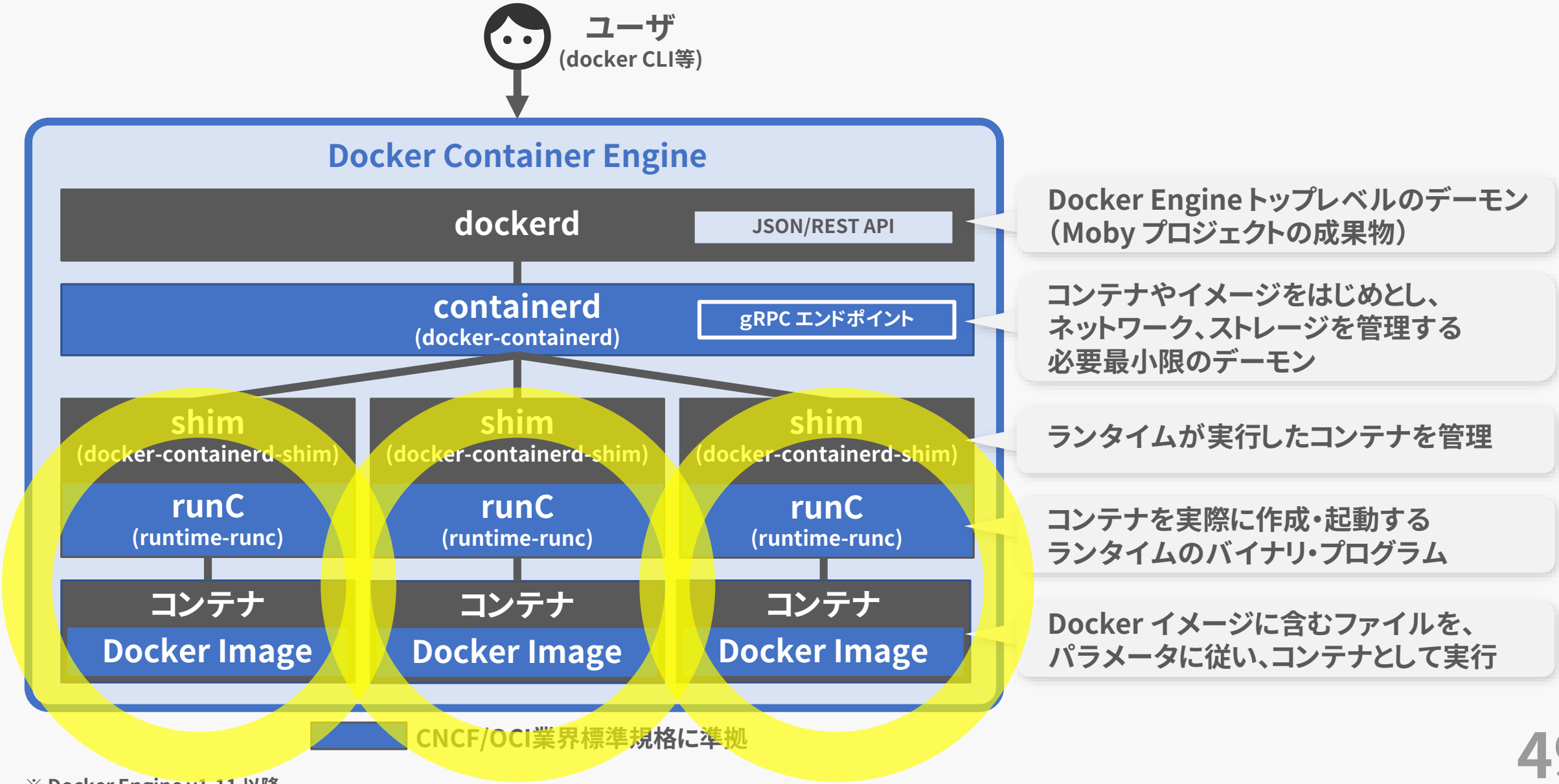
pull



Dockerはサーバ・クライアント型モデル

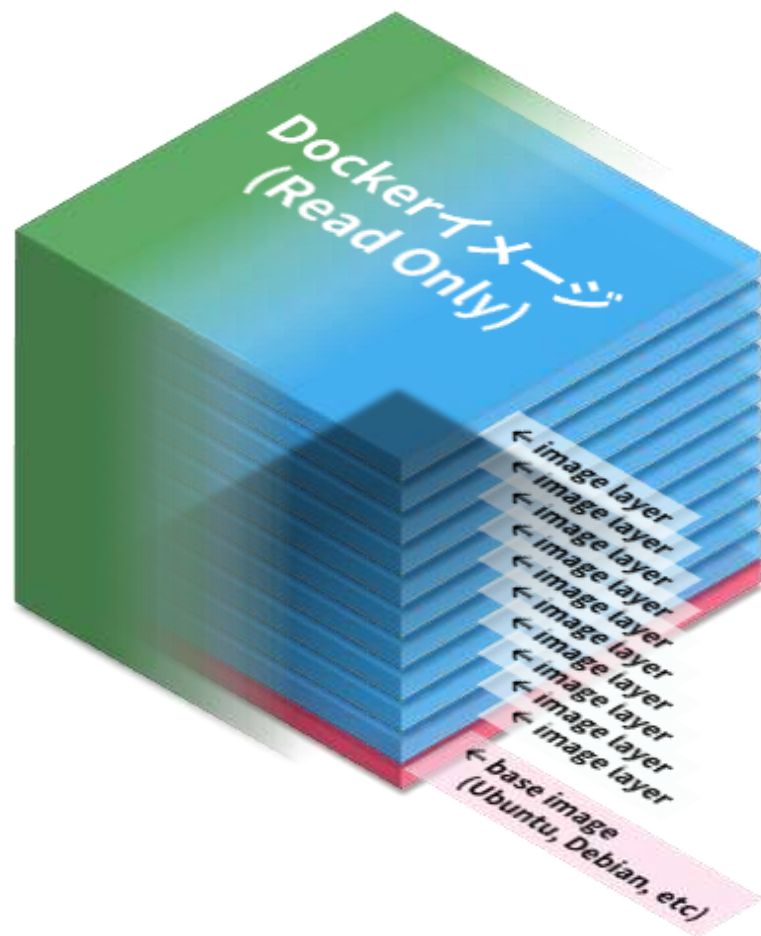


参考: Docker Engineのアーキテクチャ

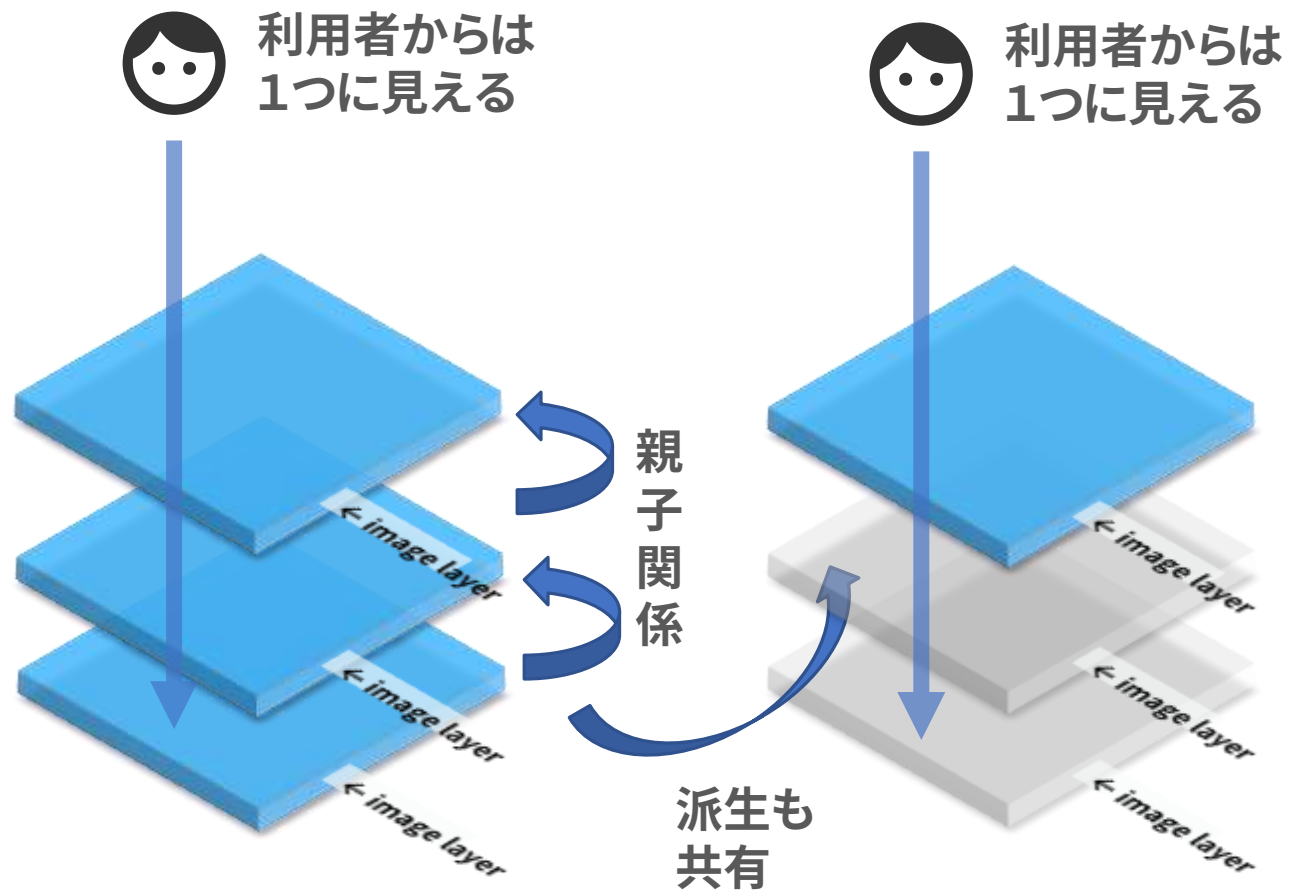


※ Docker Engine v1.11 以降

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

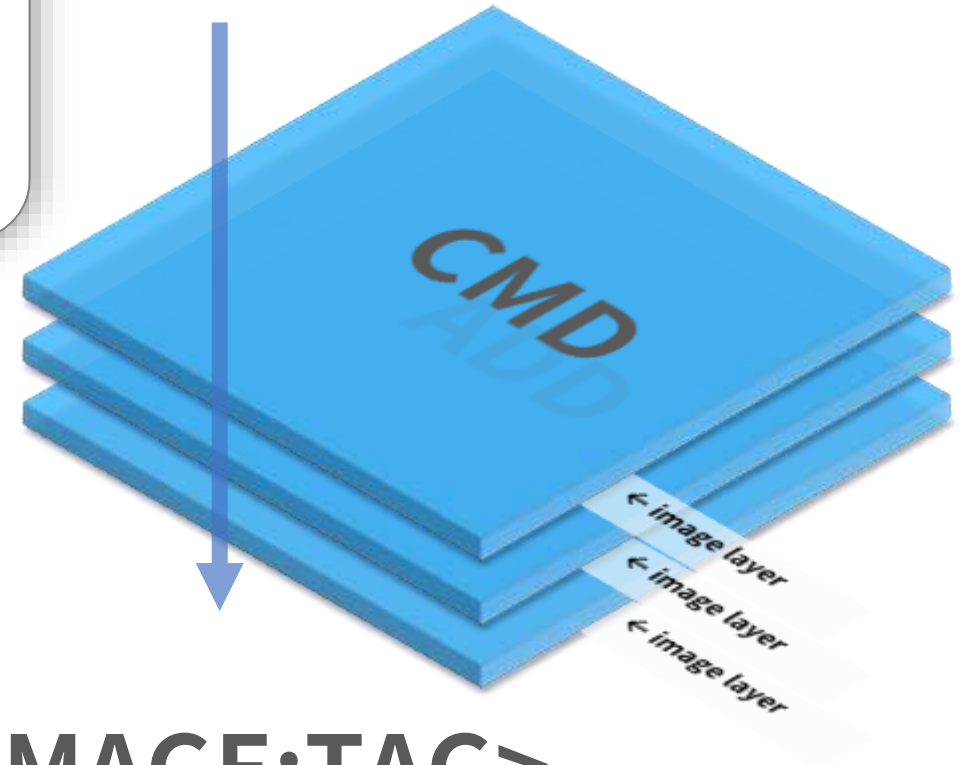
4 lines (3 sloc) | 49 Bytes

```
1  FROM scratch
2  ADD rootfs.tar.xz /
3  CMD ["/bin/sh"]
```

alpine



利用者からは
1つに見える



docker image build -t <IMAGE:TAG> .

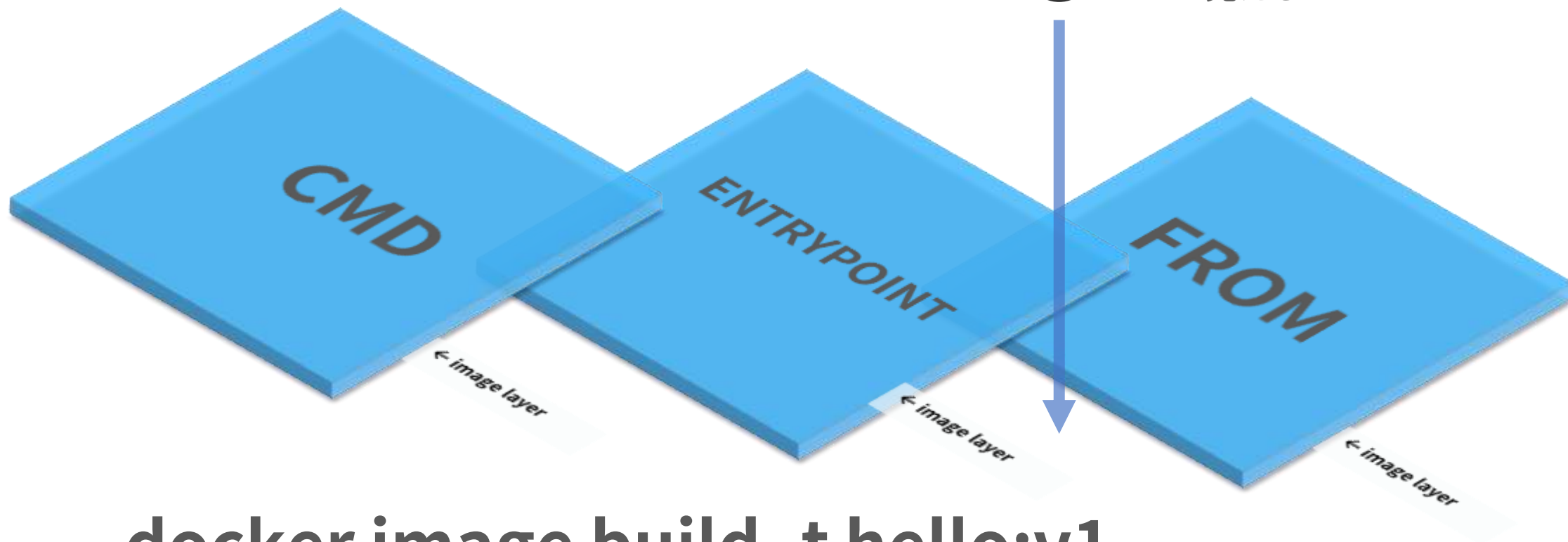
(docker build)

```
FROM alpine
ENTRYPOINT ["/bin/echo"]
CMD ["こんにちは!こんにちは!"]
```

hello:v1



利用者からは
1つに見える

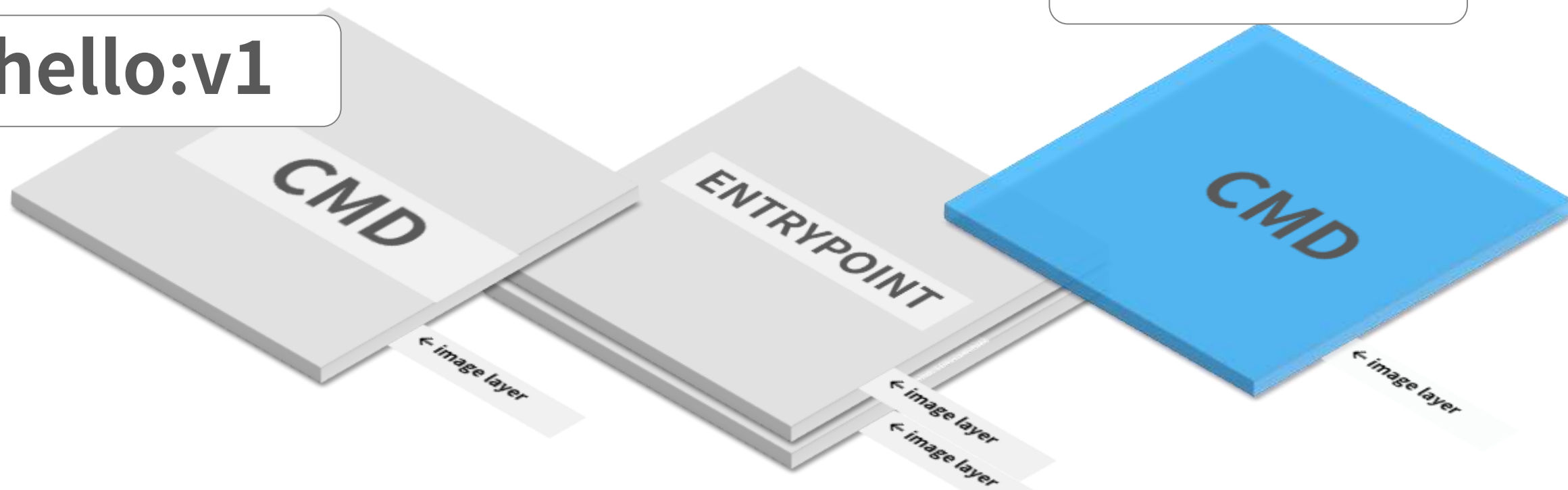


docker image build -t hello:v1 .
(docker build)

```
FROM alpine  
ENTRYPOINT ["/bin/echo"]  
CMD ["おはよう!おはよう!"]
```

hello:v1

hello:v2



```
docker image build -t hello:v2 .  
(docker build)
```


Alpine Linux

<https://www.alpinelinux.org>



Small. Simple. Secure.

Alpine Linux is a security-oriented, lightweight Linux distribution based on musl libc and busybox.

gliderlabs / docker-alpine

gliderlabs / docker-alpine

Watch 146

Star 4,714

Fork 424

Code

Issues 68

Pull requests 4

Projects 0

Insights

Tree: c14b86580b docker-alpine / versions / library-3.8 / x86_64 /

Create new file

Upload files

Find file

History

gliderbot release image version library-3.8/x86_64 for build 906

Dockerfile

Add 3.8

options

Tag 3.7 images as

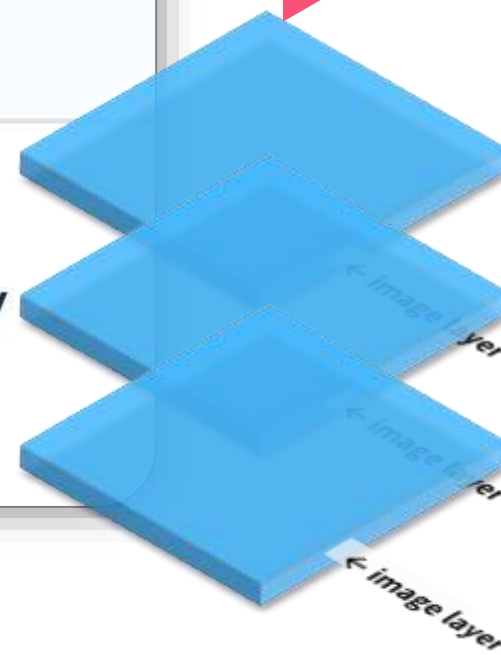
rootfs.tar.xz

release image ver

4 lines (3 sloc) | 49 Bytes

```
1 FROM scratch
2 ADD rootfs.tar.xz /
3 CMD ["/bin/sh"]
```

イメージ・レイヤ

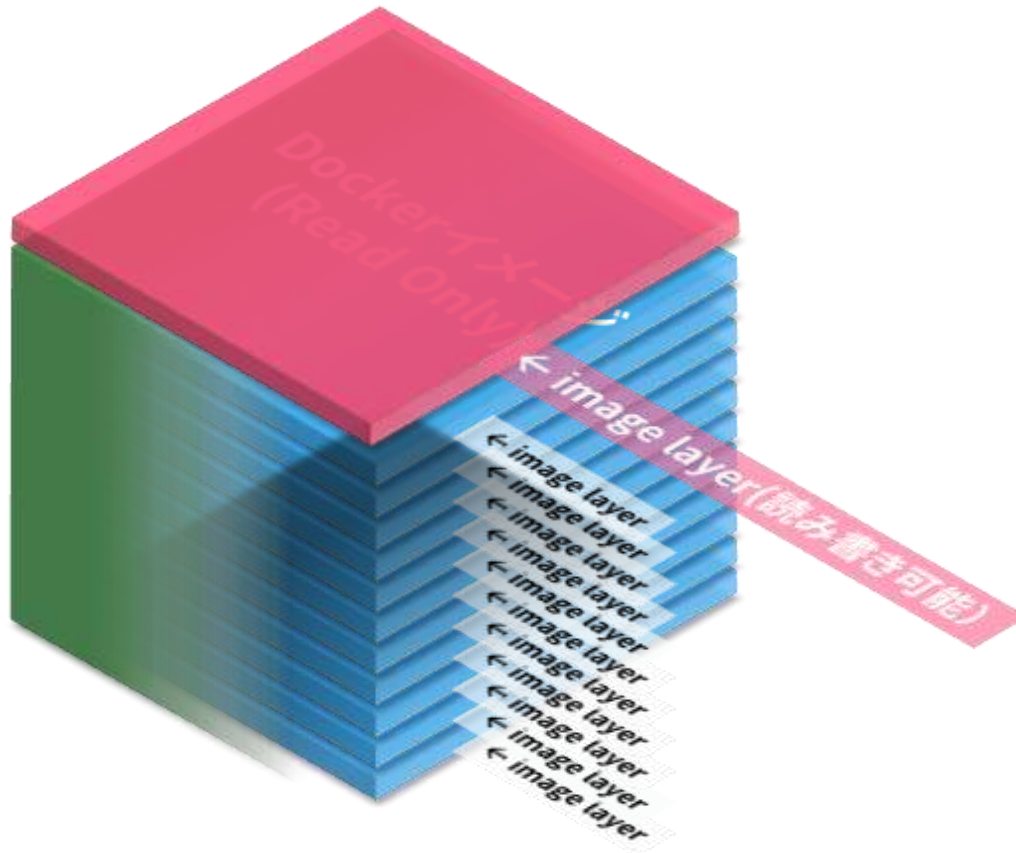


docker image pull (docker pull)

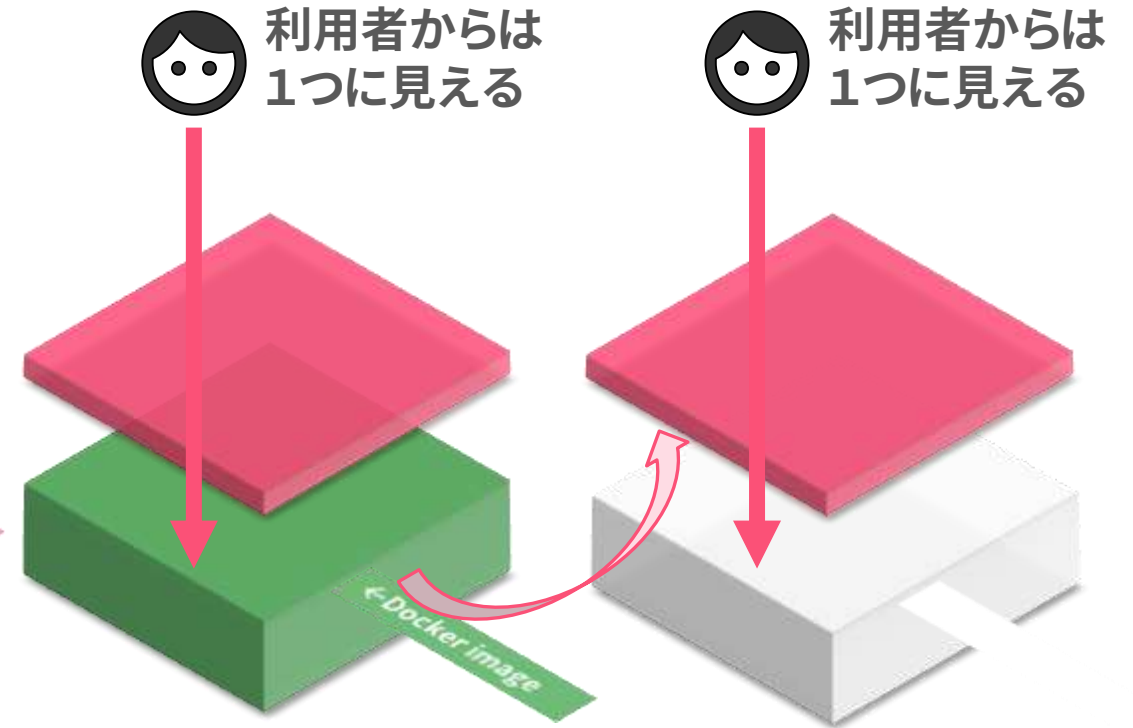


```
$ docker pull hello-world
Using default tag: latest
latest: Pulling from library/hello-world
d1725b59e92d: Pull complete
Digest: sha256:0add3ace90ecb4adbf7777e9aacf18357296e799f81cabc9fde470971e499788
Status: Downloaded newer image for hello-world:latest
```

DockerコンテナはDockerイメージを実行



元のレイヤに対する変更情報を記録
Copy on Write の性質

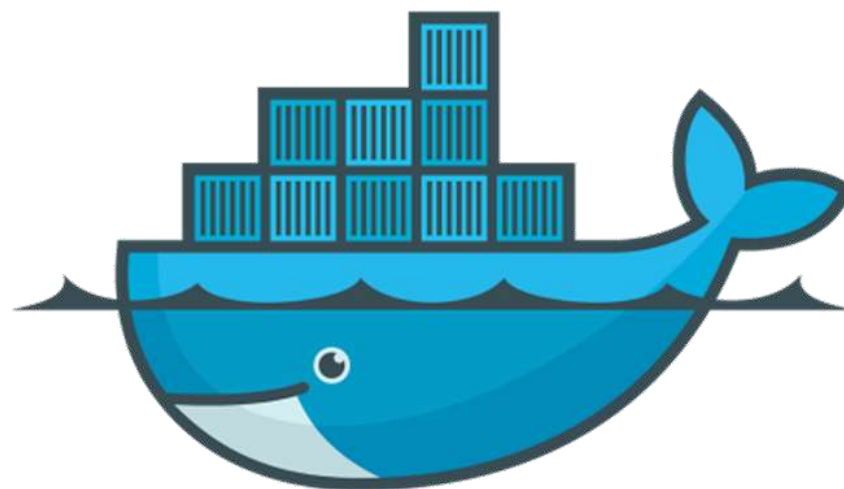
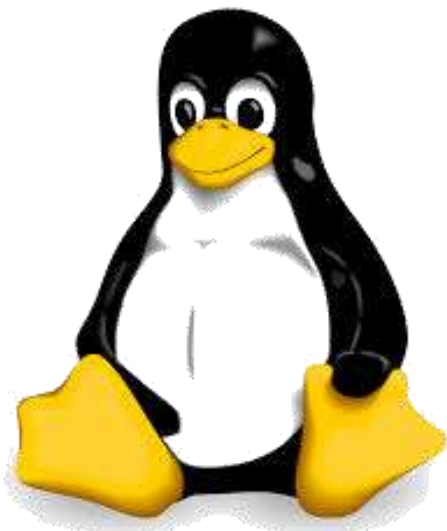


だから高速に移動できる・開発を高速化できる
リソースを有効に使える“lightweight”な性質

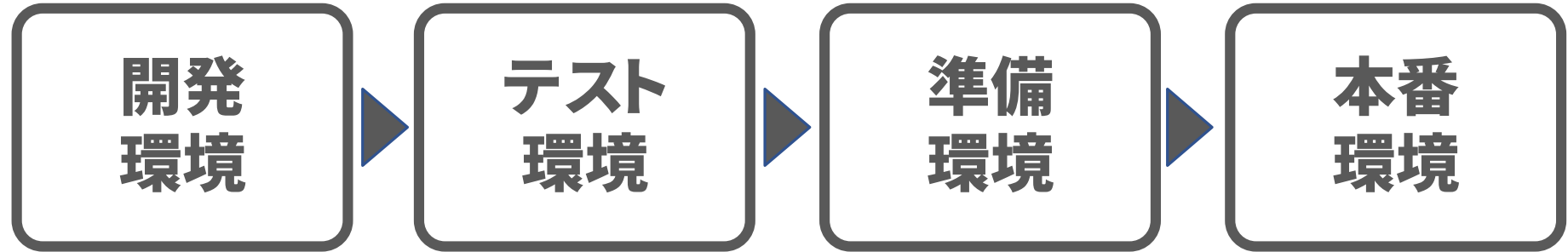
コンテナ
Technology

Docker
Specification

技術と仕様



Dockerは構築・移動・実行のためのプラットフォーム



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

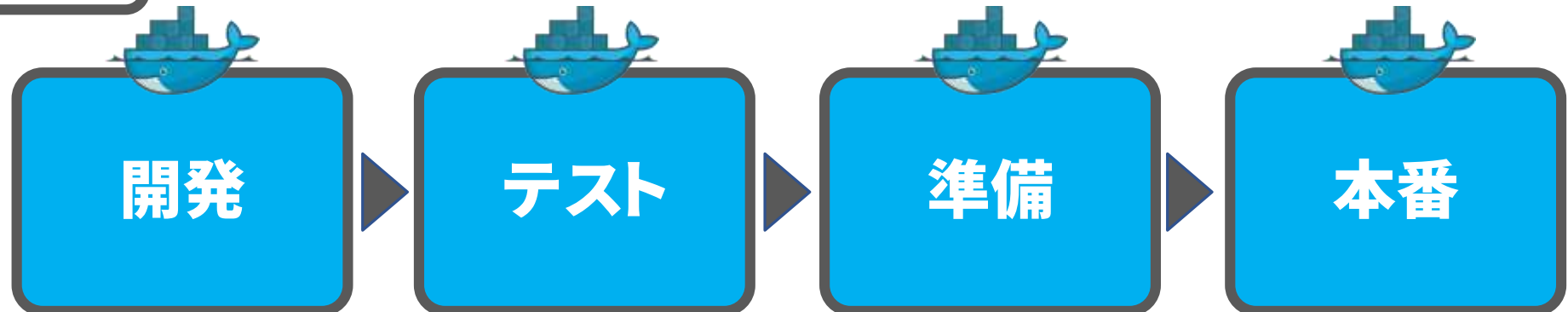
課題 時間がかかる

課題 遅い

課題 面倒



コンテナ



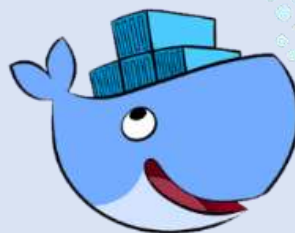
docker クライアント

```
$ docker container run hello-world
```



run

docker エンジン



コンテナ化した
hello-worldの実行



イメージ

pull

レジストリ

Docker Hub

hello-world レポジトリ



イメージ

latest
タグ

Hello from Docker.
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:

1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
3. The Docker daemon created a new container from that image which runs the executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it to your terminal.

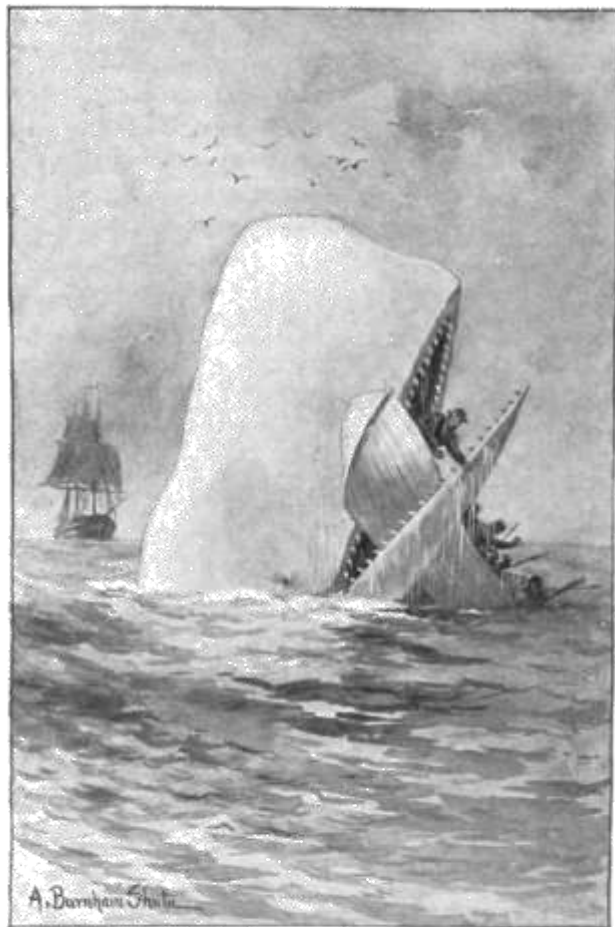
To try something more ambitious, you can run an Ubuntu container with:
\$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker Hub account:
<https://hub.docker.com>

For more examples and ideas, visit:
<https://docs.docker.com/userguide/>

Dockerイメージを取得したり、共有するための「公開レジストリ」が「Docker Hub」。

ちなみにDocke^{モビー}rのキャラクターの名前は“Moby”



“Both jaws, like enormous shears, bit the craft completely in twain.”
—Page 510.



QUEEQUEG AND HIS HARPOON.

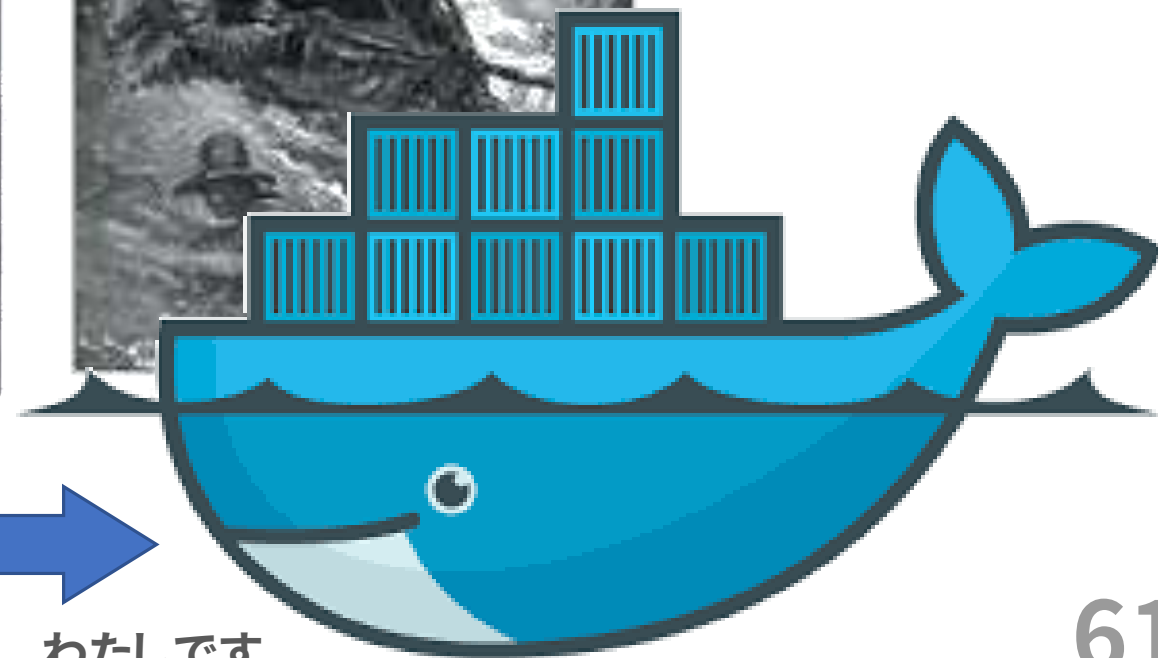


白鯨 – Moby-Dick

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



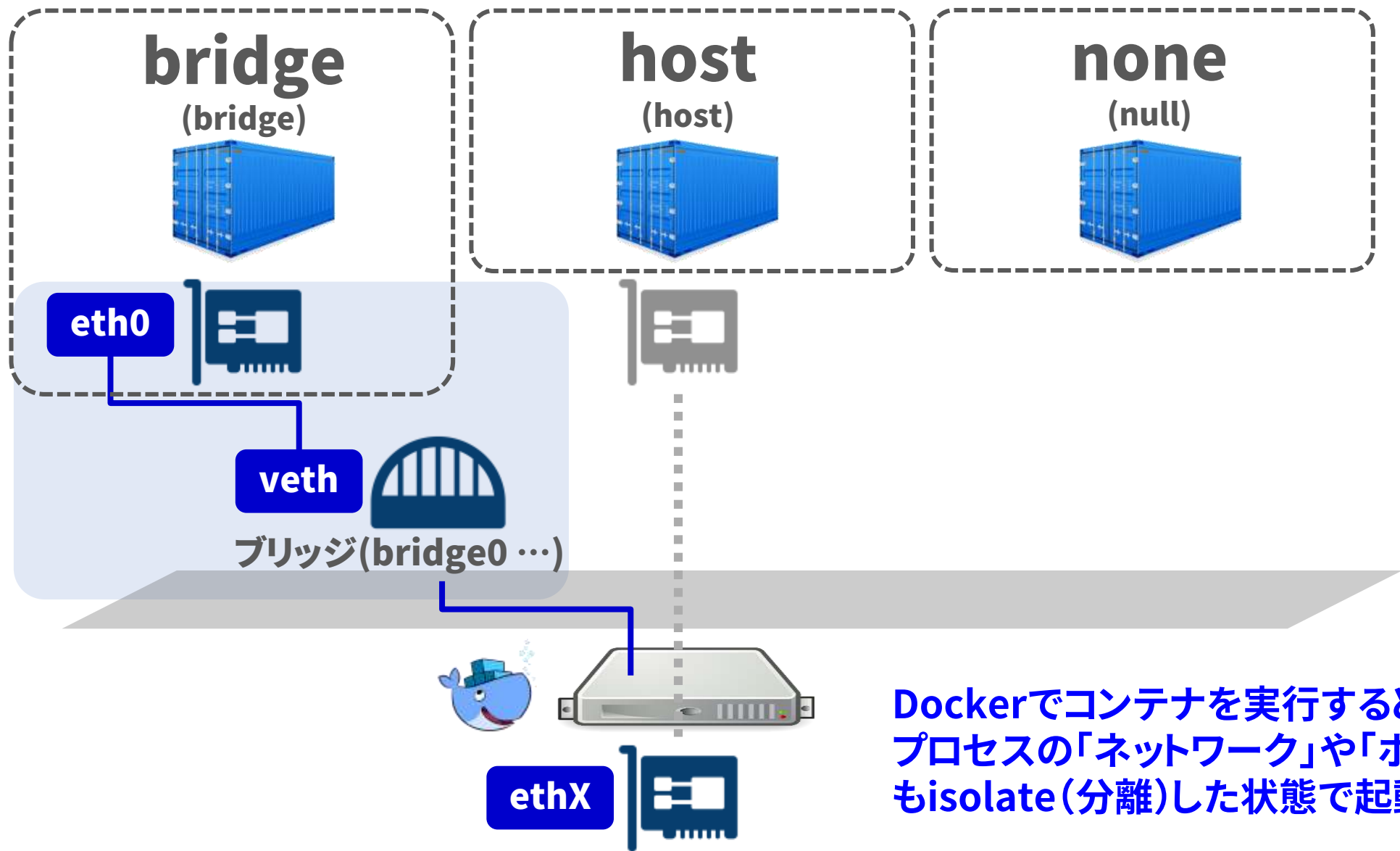
わたしです



Docker network

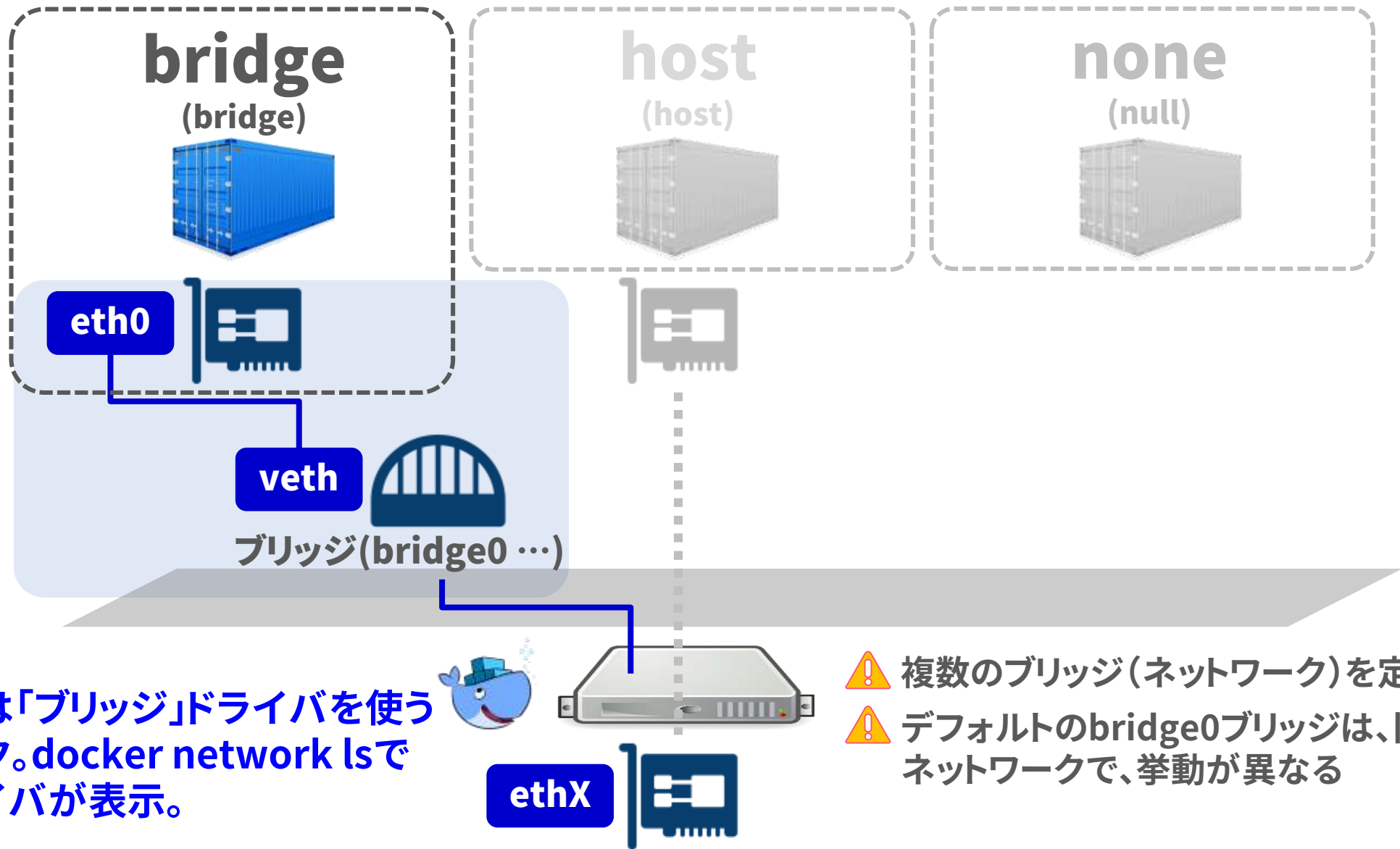
Docker ネットワーク概要

3つの Docker 標準ネットワークモデル



Dockerでコンテナを実行すると、プロセスの「ネットワーク」や「ホスト名」もisolate(分離)した状態で起動。

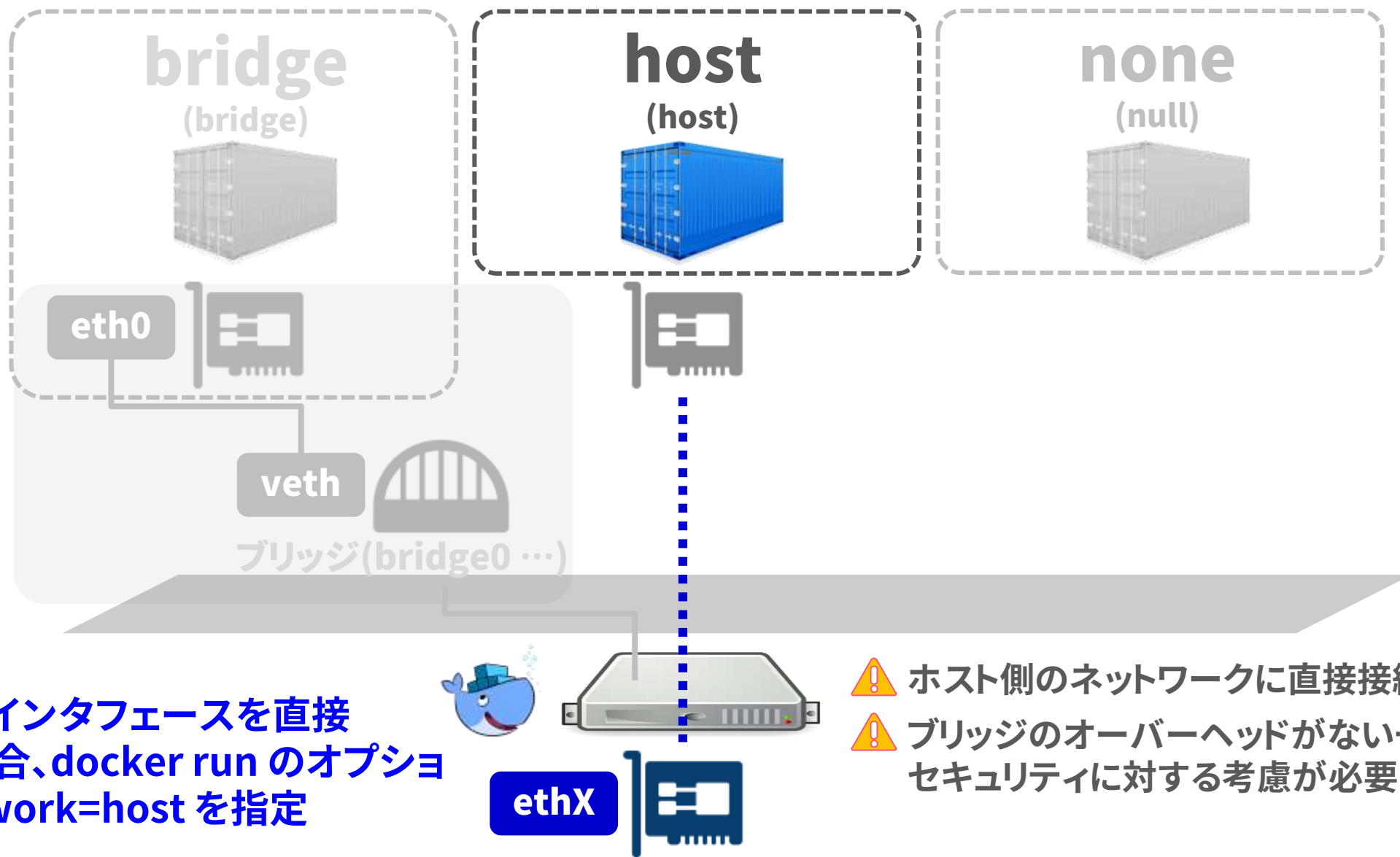
3つの Docker 標準ネットワークモデル



デフォルトは「ブリッジ」ドライバを使うネットワーク。`docker network ls`で3つのドライバが表示。

- ⚠ 複数のブリッジ(ネットワーク)を定義できる
- ⚠ デフォルトのbridge0ブリッジは、旧仕様のネットワークで、挙動が異なる

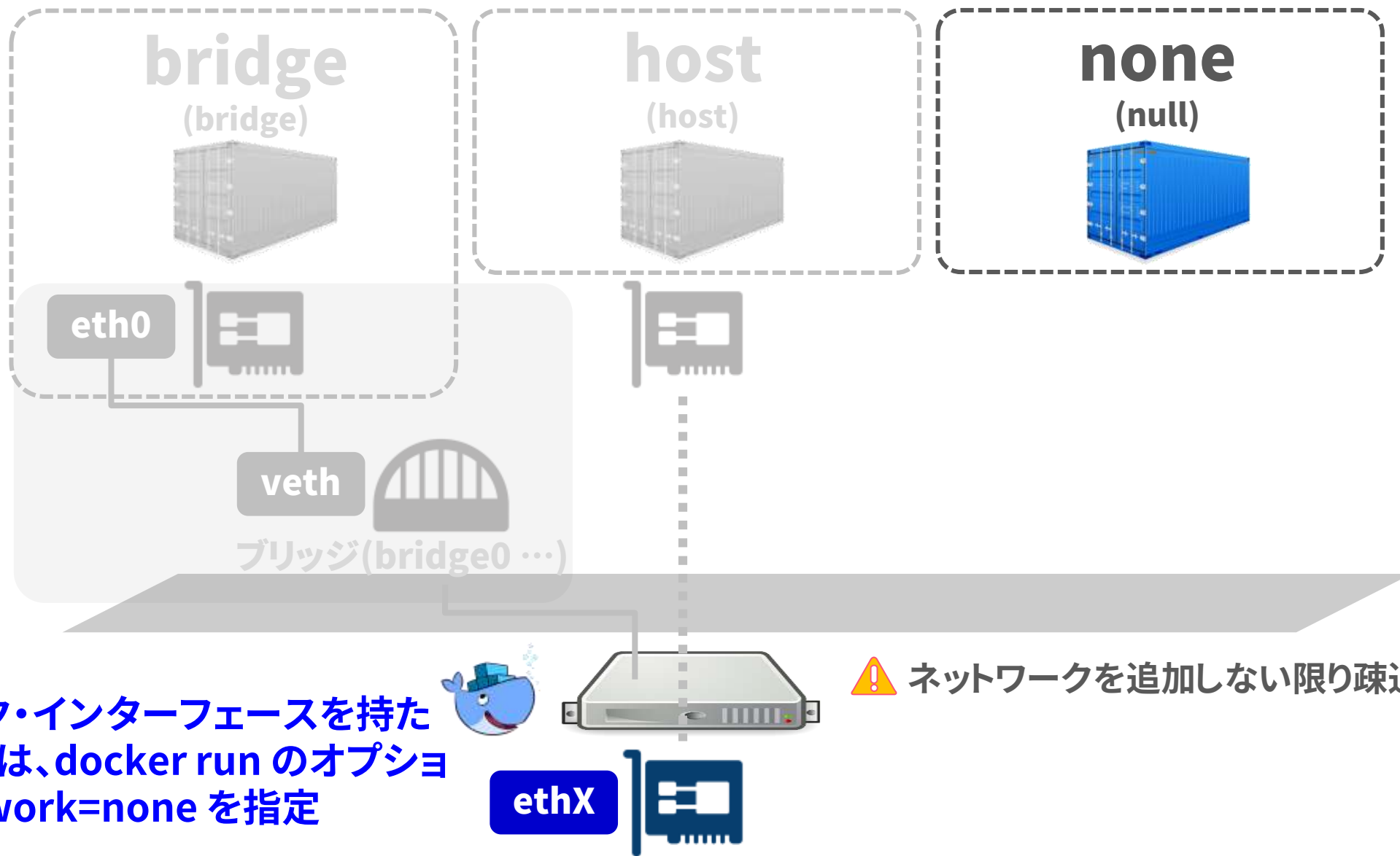
3つの Docker 標準ネットワークモデル



ホスト側のインタフェースを直接
使いたい場合、`docker run` のオプションで `--network=host` を指定

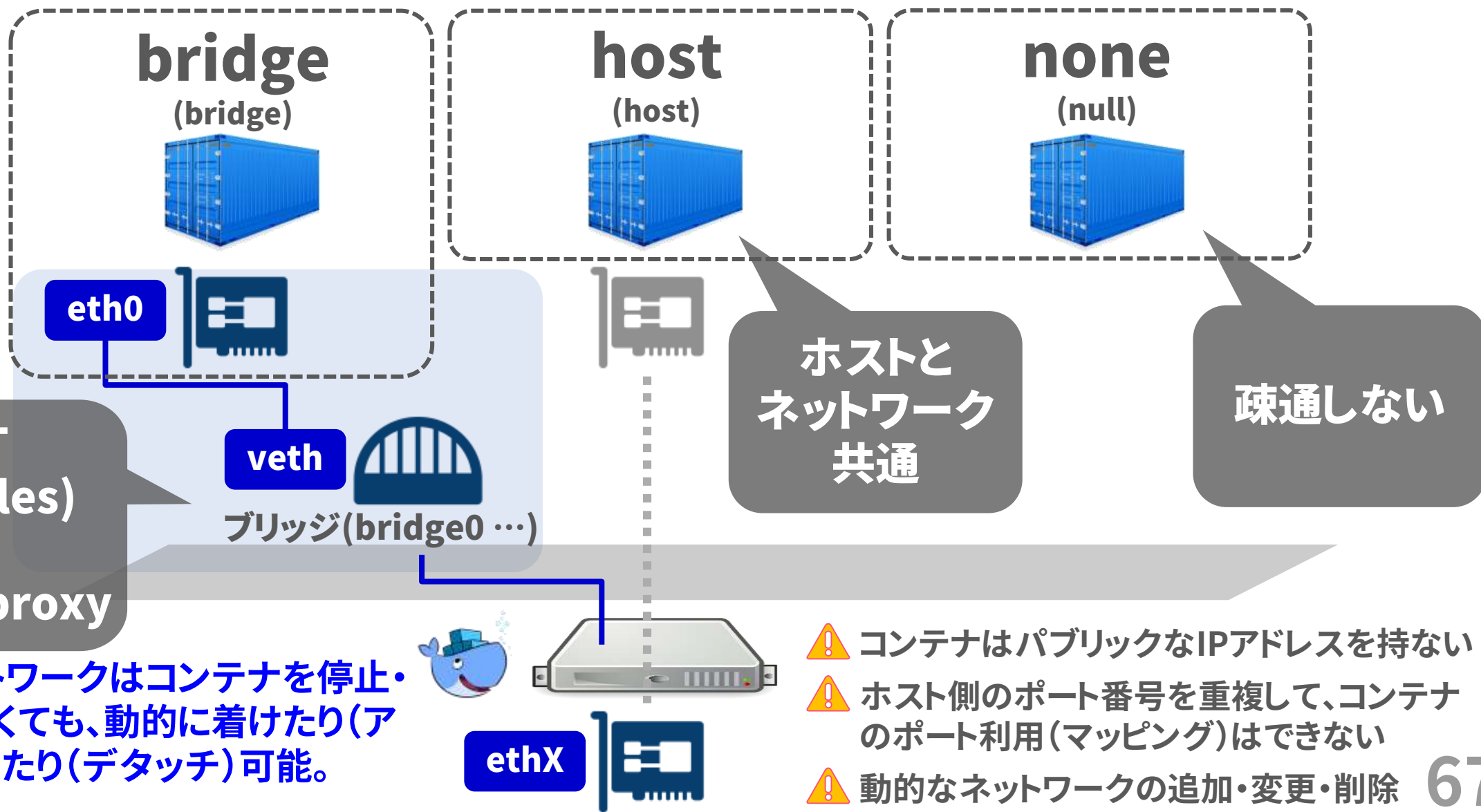
- ⚠ ホスト側のネットワークに直接接続
- ⚠ ブリッジのオーバーヘッドがない一方で、セキュリティに対する考慮が必要

3つの Docker 標準ネットワークモデル

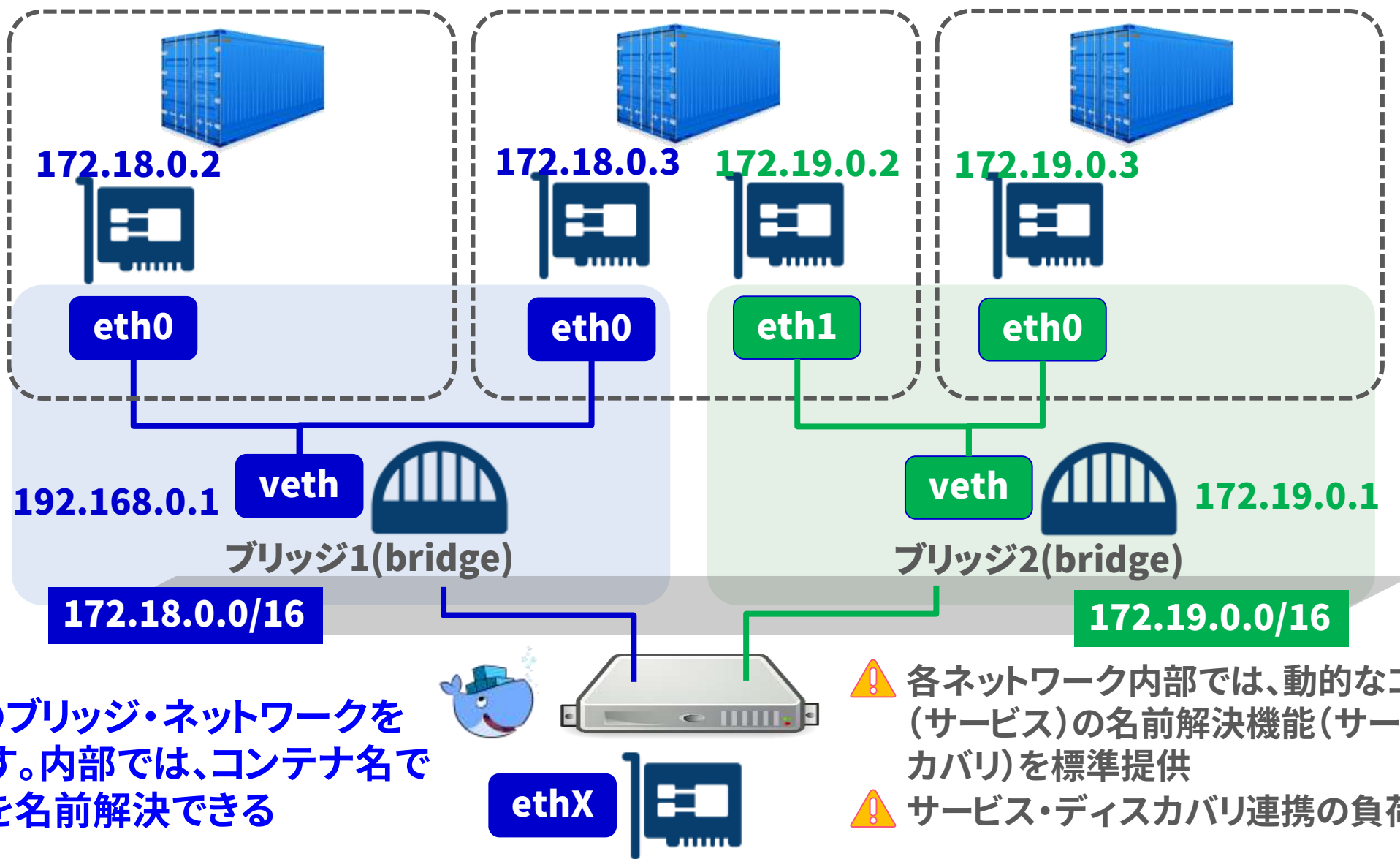


ネットワーク・インターフェースを持たせない場合は、**docker run** のオプションで **--network=none** を指定

3つの Docker 標準ネットワークモデル



コンテナは複数のネットワーク(ブリッジ)に接続できる



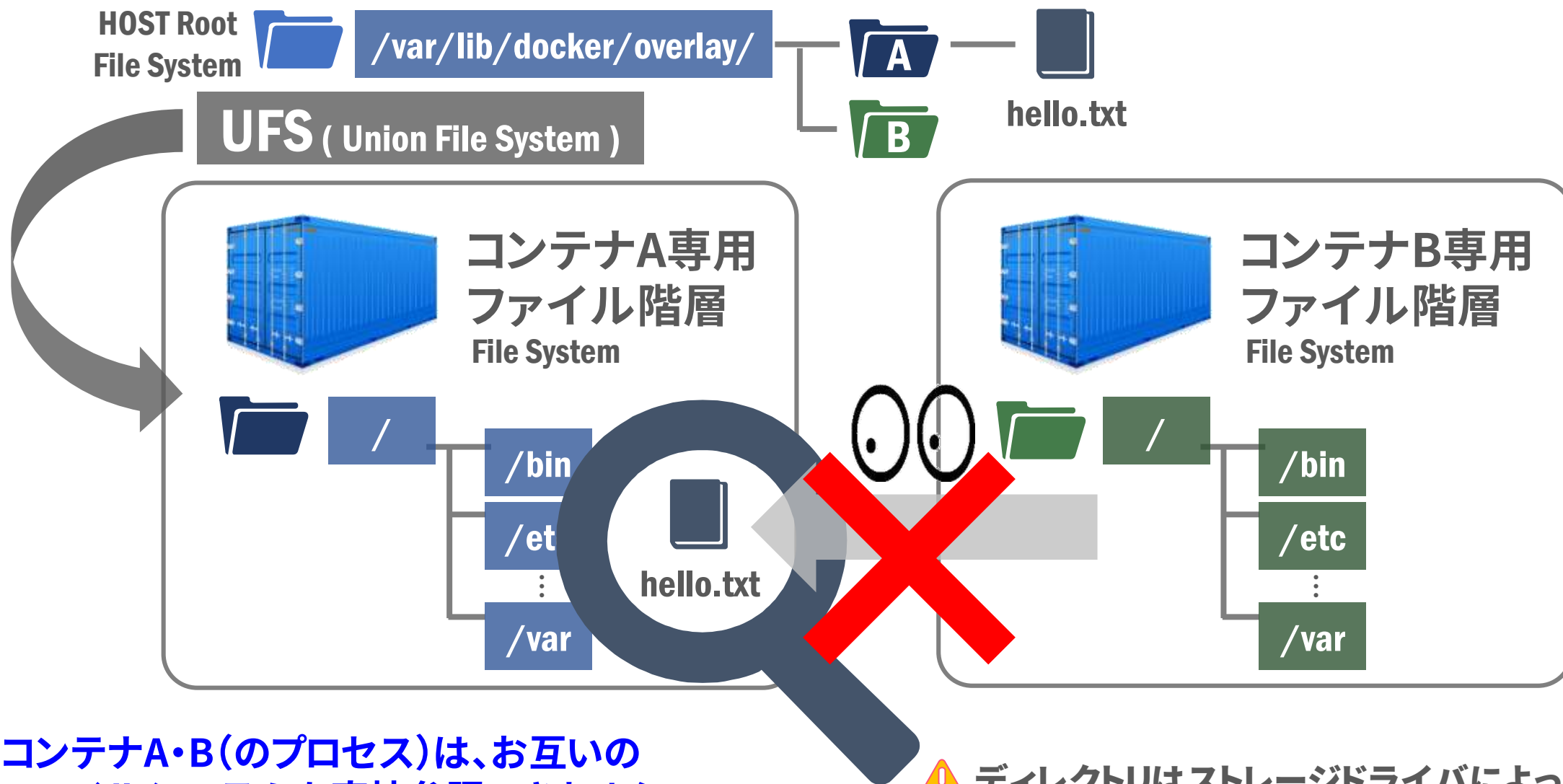
また、複数のブリッジ・ネットワークを定義できます。内部では、コンテナ名でIPアドレスを名前解決できる

- ⚠ 各ネットワーク内部では、動的なコンテナ名(サービス)の名前解決機能(サービス・ディスカバリ)を標準提供
- ⚠ サービス・ディスカバリ連携の負荷分散

Docker volume

Docker ボリューム概要

データの扱い

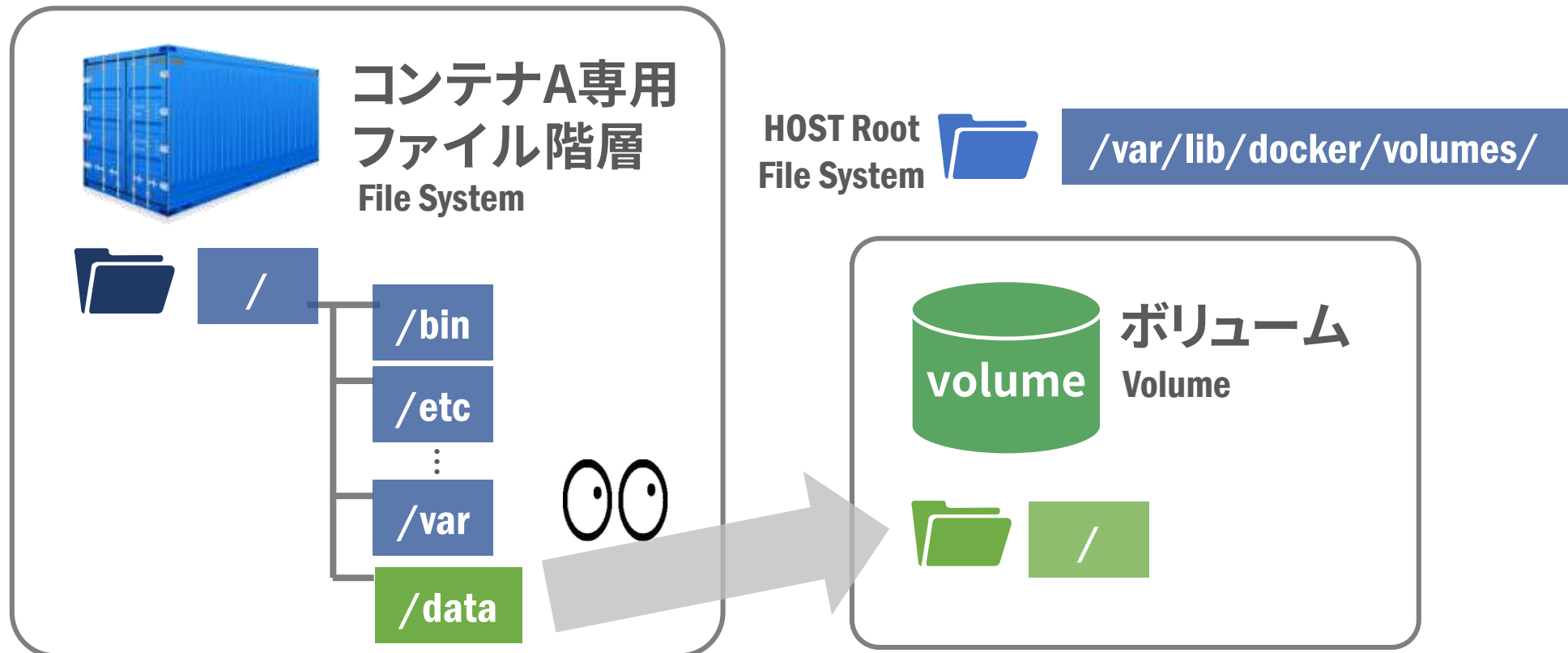


コンテナA・B(のプロセス)は、お互いの
ファイルシステムを直接参照できません。

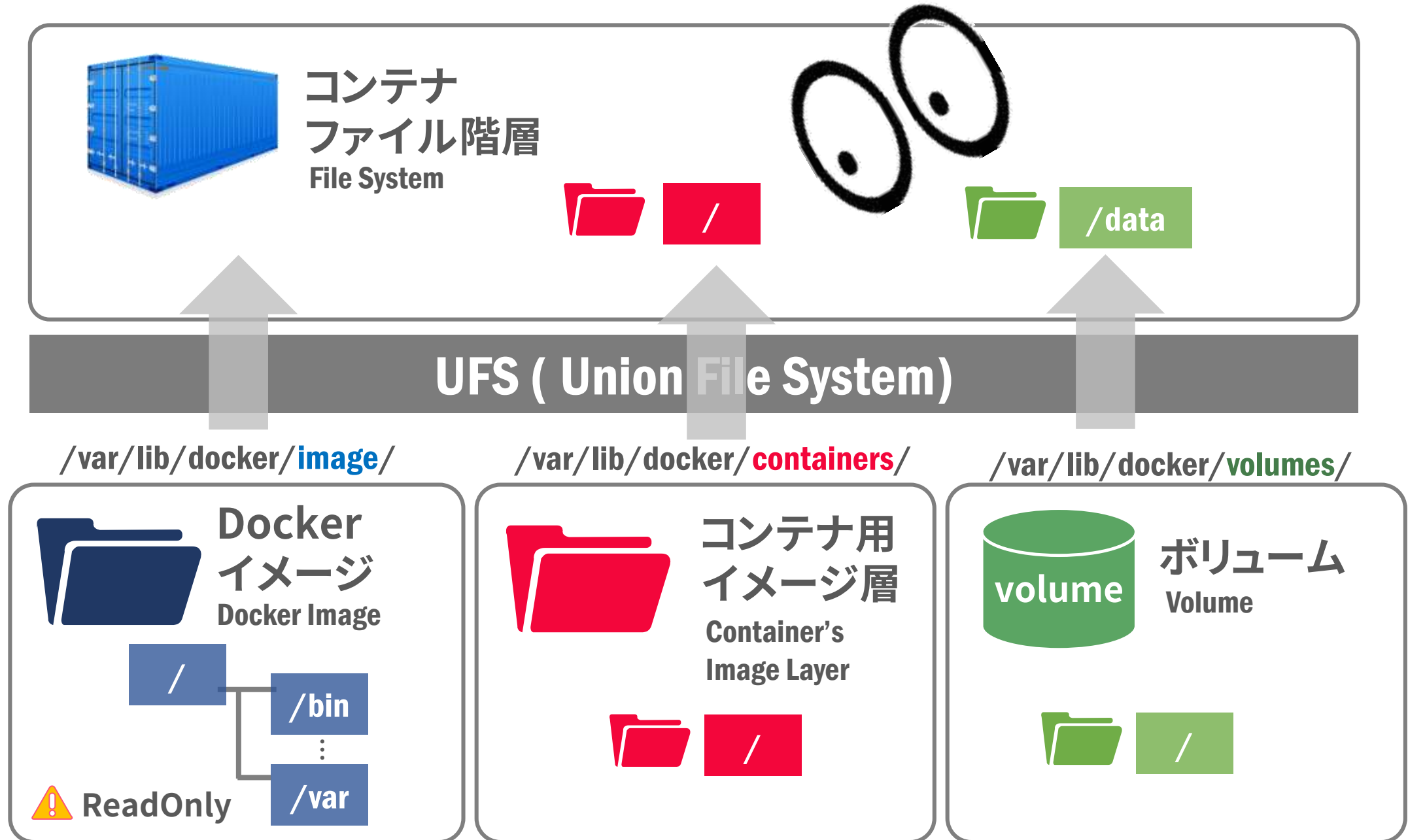
⚠️ ディレクトリはストレージドライバによって異なる70

データ・ボリューム

コンテナからはUFSを通してデータ領域が見える
ストレージ・ドライバのオーバーヘッドを受けない
複数のコンテナでボリュームを共有できる



コンテナは「ボリューム」と呼ぶ領域(ディレクトリまたはファイル)をマウントできる



ボリュームは3分類

ホストをマウント



/data

/docker/data

ホスト上のディレクトリ

名前付き



/data

volume

名前無し



/data

/etc

volume

ボリュームとして、ホスト上のファイルを直接マウントできるだけでなく(`docker run -v ホスト側:コンテナ内`)、ボリュームを複数のコンテナで共有可能。読み書きするデータは、通常このボリュームを作成し、利用する。



- ⚠ ボリュームはコンテナ間でデータを**共有**できる
- ⚠ ボリュームの**実体は、ホスト上**のディレクトリ `/var/lib/docker/volumes`

Why? There are some reasons to manage containers.

Docker Compose が必要な理由

システム
コンテナ

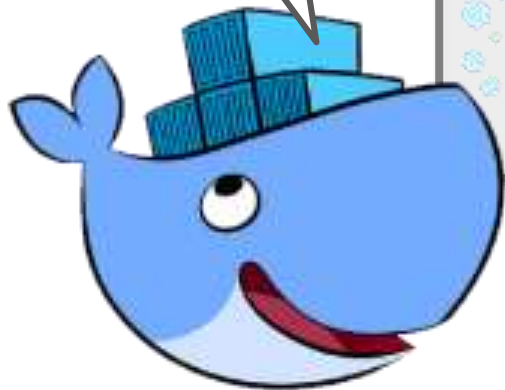
コンテナ

プロセス
A

プロセス
B

プロセス
C

スケールしづらさ
管理のしづらさから
Dockerやk8sには
向かないね



アプリケーション
コンテナ

ソフトウェアごとに
コンテナが別だから
スケールしやすいし
差し替えも簡単

コンテナ

プロセス
A

docker run ...

コンテナ

プロセス
B

docker run ...

コンテナ

プロセス
C

docker run ...



もしも コンテナ が 20こ ひつよう なら



- 環境構築が大変
- 環境の維持・再現が大変
- 本末転倒

docker run ...

docker run ...

docker run ...

docker run ...

docker run ...

docker run ...

docker run ...

docker run ...

docker run ...

docker run ...

docker run ...

docker run ...

docker run ...

docker run ...

docker run ...

docker run ...

docker run ...

docker run ...



docker-compose up

利用者側のメリット:

- compose ファイルがあれば、とにかく動くので(環境の再現性が高い)

開発者側のメリット:

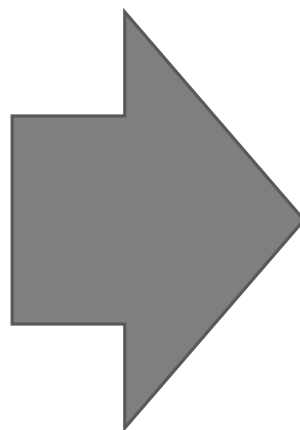
- 1つのマシン上で、複数の環境を切り替えやすい

- 環境はYAMLで管理
- docker コマンドと類似



Fig

<http://www.fig.sh>



Docker Compose

複数のコンテナを一括管理する Docker Compose

Docker Composeとは？

What is Docker Compose?

multi-container applications
複数のコンテナで構成するアプリケーションを

define and run
定義と実行するためのツール

1

Composeは
アプリケーションの
サービスをファイル
で定義する

- Compose ファイル (YAML形式) で Docker コンテナのサービスを定義できるため、再利用性が高い

- コンテナの状態
- イメージ
- ネットワーク
- ボリューム
- メタ情報



2

Dockerコマンドと
高い親和性がある
ため、学習コストが
比較的に低い

- docker-compose CLI のコマンド体系は docker に準拠
 - 例: 「docker run -d」は「docker-compose up -d」

3

Swarm モードに
サービスをデプロイ
できるオーケストレー
ション機能

- Docker swarm モードを使えば、サービス状態を定義できるので常に期待状態を維持できる
 - docker stack deploy
- Ingress ネットワークの活用

<https://github.com/docker/compose>

Docker Compose 活用場面

- 利用者視点
 - docker-compose.yml があれば、すぐに何でも実行できる
 - Docker Hub 公式イメージ
 - PWD
- 開発者視点
 - 環境の再構築が簡単
 - バージョン違いの環境を作りやすい

Dockerのセットアップ

Linuxで検証用途に手軽な方法

```
$ curl https://get.docker.com | head
```

※コマンドの確認



```
$ curl -fsSL get.docker.com -o get-docker.sh  
# sh ./get.docker.com
```

※リポジトリの自動セットアップ



```
# systemctl enable docker  
# systemctl start docker
```

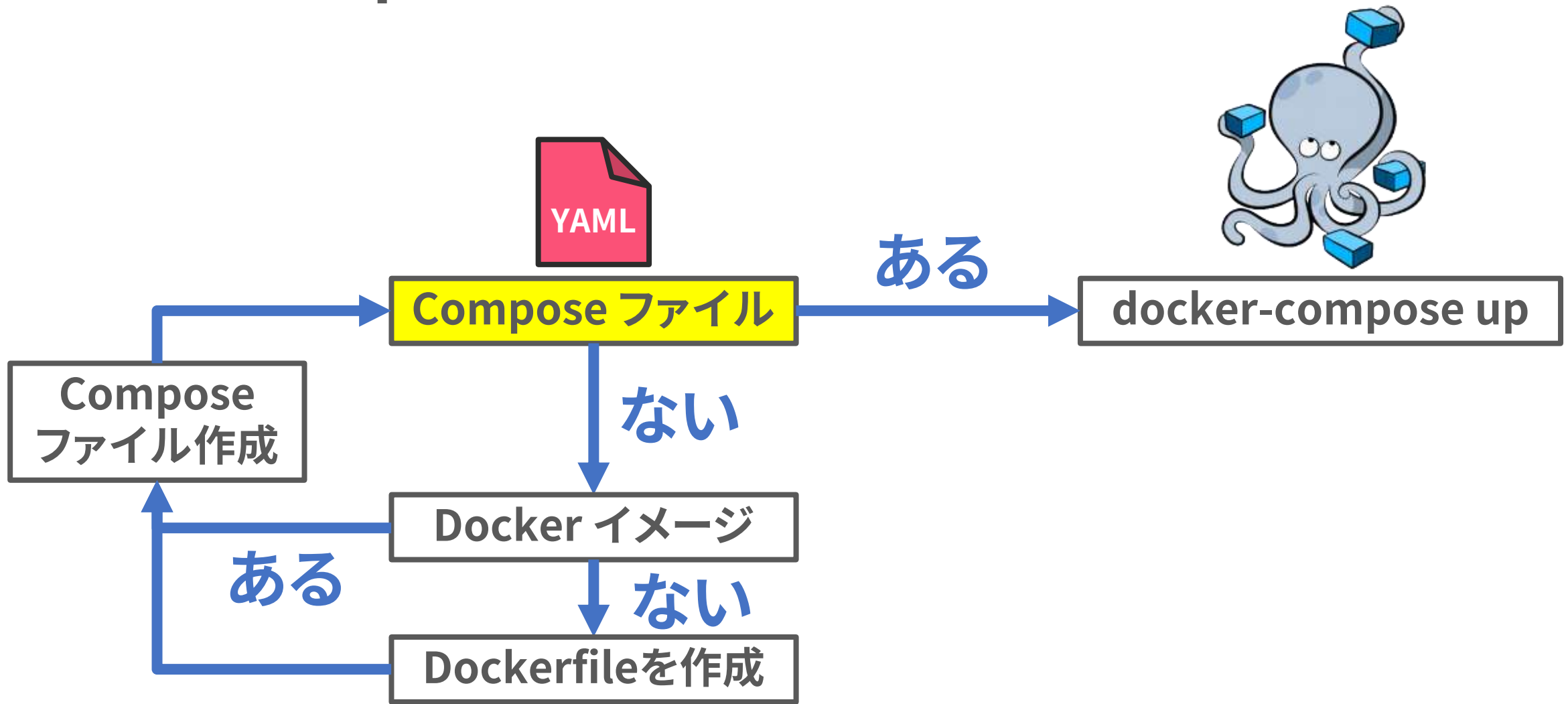
※systemd系の場合

Docker Compose セットアップ方法

- Docker Desktop for Mac / Windows
 - docker-compose も同梱されているので、すぐに使える
- Linux 版
 - バイナリをリポジトリからローカルにダウンロードし、パーミッションを設定
 - <https://github.com/docker/compose/releases>
 - “Assets”からアーキテクチャごとにダウンロードできる
 - Linuxは “docker-compose-Linux-x86-64”

```
curl -L https://github.com/docker/compose/releases/download/1.23.2/docker-compose-Linux-x86_64 ¥  
-o /usr/local/bin/docker-compose  
chmod 755 /usr/local/bin/docker-compose
```

Docker Composeを使うための基本ステップ



「サービス」「ネットワーク」「ボリューム」などをYAML形式で記述した、Composeファイルが必要。 85

docker-compose.yml

services:

networks:

volumes:

- 「サービス」「ネットワーク」「ボリューム」を定義
 - image … イメージ ID やイメージ名
 - build … Dockerfile のパス (イメージを構築する場合)
 - command … デフォルトの (イメージ実行時) コマンドを上書き
 - restart … コンテナの再起動ポリシー (例: always)
 - ports … ポートのマッピング
 - environment: , env_file: 環境変数の指定

WordPress公式の Docker Compose例

version: '3.1' ◀ ネットワークやボリュームを使う場合、通常2以上
swarm modeにデプロイする場合は3を想定

services: ◀ サービスを定義

wordpress: ◀ サービス「wordpress」

image: wordpress ◀ 使用するDockerイメージは「wordpress:latest」

restart: always ◀ 停止時、常にプロセス再起動

ports:

- 8080:80 ◀ ホスト側ポート8080を、コンテナ内ポート80に割り当て

environment: ◀ コンテナ内で使える環境変数を定義

WORDPRESS_DB_HOST: db ◀ サービス「db」で名前解決できる

WORDPRESS_DB_USER: exampleuser

WORDPRESS_DB_PASSWORD: examplepass

WORDPRESS_DB_NAME: exampledb

db: ◀ サービス「db」

image: mysql:5.7 ◀ 使用するDockerイメージは「mysql:5.7」

restart: always ◀ 停止時、常にプロセス再起動

environment: ◀ コンテナ内で使える環境変数を定義

MYSQL_DATABASE: exampledb ◀ WordPress用のデータベース、
MySQL_USER: exampleuser ユーザ、ユーザのパスワードを定義

MYSQL_PASSWORD: examplepass

MYSQL_RANDOM_ROOT_PASSWORD: '1' ◀ 初回実行時rootパス自動生成

※ネットワークを定義していないが
docker-compose up時に自動的に、
このプロジェクト専用の bridge
ネットワークが作成される。

※wordpressとdbのコンテナ間は
ポートの公開を明示しなくても、
同一 bridge ネットワークに所属
しているため、お互い内部通信可能

別例:

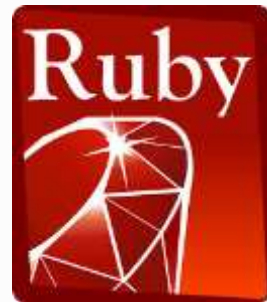
「web」という名称のサービスを
定義(内部ネットワークで名前
解決できる)▶

```
version: '3' ◀ ネットワークやボリュームを使う場合、通常2以上
swarm modeにデプロイする場合は3を想定

services:
  web:
    image: zembutsu/docker-sample-nginx ◀ 使用するDockerイメージ
    deploy: ◀ docker stack deploy時の挙動
      replicas: 3 ◀ コンテナのレプリカ(複製)を3つ起動
    resources:
      limits:
        cpus: "0.1" ◀ CPUリソース制限
    restart_policy:
      condition: on-failure ◀ 再起動ポリシーを指定:障害時に再起動
    ports:
      - "80:80" ◀ ホスト側のポート80を、コンテナ内のポート80にマッピング
    networks:
      internal: ◀ 「internal」という名称のブリッジネットワークに接続し、
        aliases: frontend というエイリアス(別名)を指定
        - frontend
    volumes:
      - /etc/localtime:/etc/localtime:ro ◀ ボリューム定義
                                                /etc/localhost を読み込み専用で

networks: ◀ ネットワーク定義
  internal: internalという名称のブリッジ・ネットワーク
```


Mastodon



services:



<https://github.com/tootsuite/mastodon>

サービス

redis



イメージ

redis
:alpine

サービス

postgres



イメージ

postgres
:alpine

サービス

frontend



イメージ

nginx
:alpine

サービス

web



イメージ

gargron/mastodon
:v1.4.6

サービス

streaming



イメージ

gargron/mastodon
:v1.4.6

サービス

sidekiq



イメージ

gargron/mastodon
:v1.4.6

サービス

mta



イメージ

namshi/smtp
:latest

volumes:



mastodon_redis



mastodon_postgres



certbot
(external)



assets



packs



system

networks:

80

443

ネットワーク

192.168.0.0/16 mastdon (external)

ホスト側ネットワーク

eth0等

80

443

```
version: '3'

services:
  web:
    image: zembutsu/docker-sample-nginx
    deploy:
      replicas: 3
      resources:
        limits:
          cpus: "0.1"
      restart_policy:
        condition: on-failure
    ports:
      - "80:80"
    networks:
      internal:
        aliases:
          - web
    volumes:
      - /etc/localtime:/etc/localtime:ro

networks:
  internal:
```

docker-compose.yml

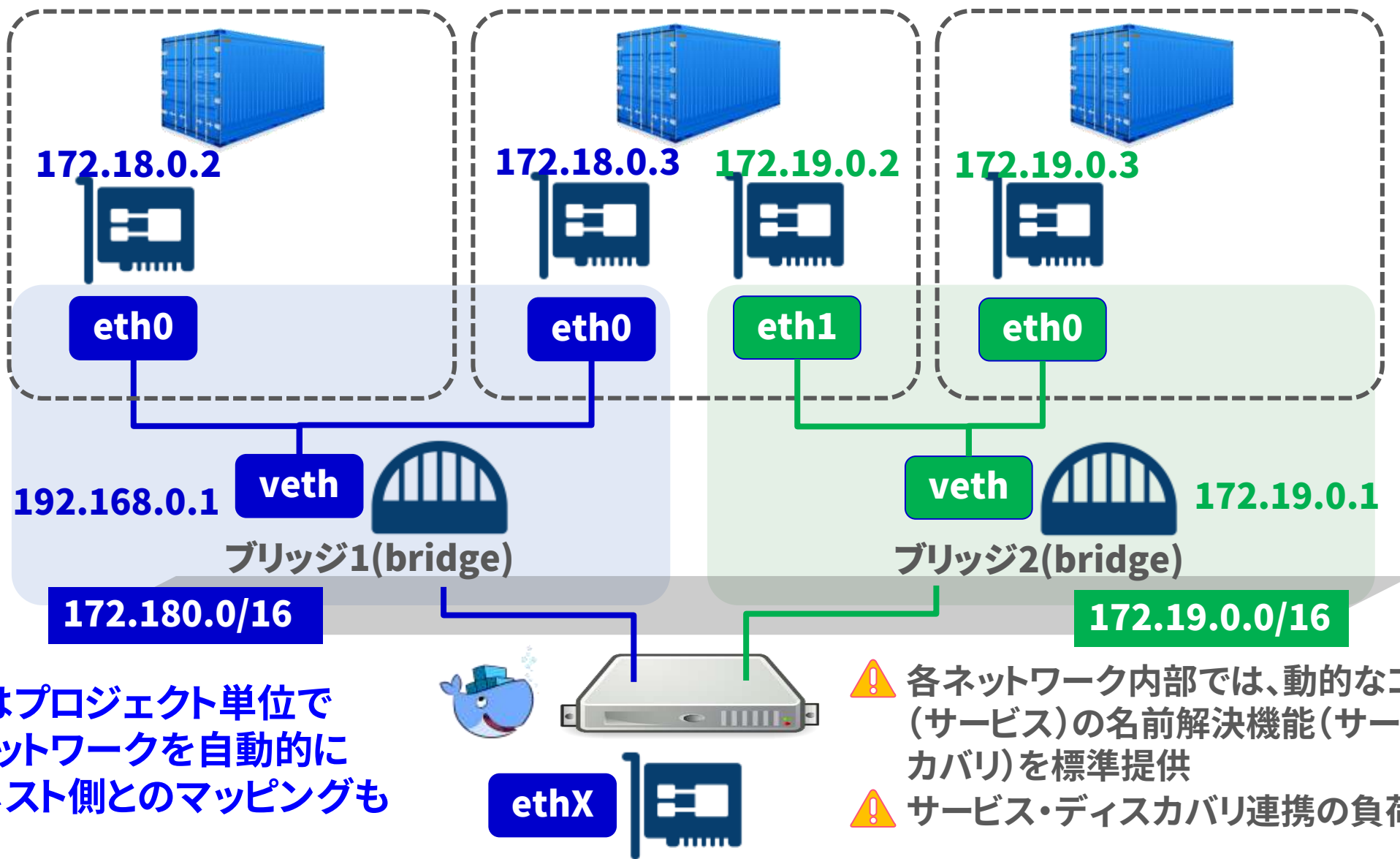
services:

networks:

volume:

- 「サービス」「ネットワーク」「ボリューム」を定義
 - image … イメージ ID
 - build … Dockerfile のパス
 - command … デフォルトのコマンドを上書き
 - restart … 再起動ポリシー(例: always)
 - ports … ポート
 - environment: , env_file: 環境変数

コンテナは複数のネットワーク(ブリッジ)に接続できる



Composeプロジェクトの「ネットワーク」定義

```
networks:  
  external_network:  
  internal_network:  
    internal: true
```

```
services:  
  web:  
    ...  
    networks:  
      hostnet: {}  
  
networks:  
  hostnet:  
    external: true  
    name: host
```

「networks」でネットワークを定義。”external:true”は、この Compose プロジェクト外で(手動・自動的に存在しているネットワークと接続)。”driver: overlay”の場合、”internal:true”で、複数ホスト間をつなぐローカルネットワークも定義できる。

ボリュームは3分類

ホストをマウント



/data

/docker/data

ホスト上のディレクトリ

名前付き



/data

volume

名前無し



/data

/etc

volume



⚠ ボリュームはコンテナ間でデータを**共有**できる

⚠ ボリュームの**実体は、ホスト上**のディレクトリ
/var/lib/docker/volumes

コンテナ(のプロセス)でマウントするボリューム(領域)も、Composeで定義できる。

Composeプロジェクトの「ボリューム」定義

volumes:

- /var/lib/mysql
- cache:/tmp/cache
- ./configs:/etc/configs/:ro

「volumes:」に“- ディレクトリまたはファイル名”の場合は名前無し(volume idが割り振られる)。主に使い捨て用途

“- 名前:パス”は名前付きボリューム。ホスト上での管理を容易にするため(バックアップや参照で)

“- パス:パス”はホスト上を直接マウント。
“:ro”は Read-Only オプション

services:

db:

image: postgres:9.4

volumes:

- db-data:/var/lib/postgresql/data

個々のサービスの中で定義する場合は、そのサービス(のコンテナ)だけが参照

プロジェクト全体または個々のサービスに定義。毎回 docker volume create等の管理が不要に。 96

Run & Tips

動かし方とコツ的な

Docker Compose Run & Tips

- `docker-compose up -d --build`
 - `docker-compose down -v --rmi all / local`
 - `docker-compose exec <サービス名> コマンド`
 - `docker-compose -f <ファイル名.yml>`
 - `docker-compose -p <プロジェクト名>`
 - `docker-compose --config`
-
- `docker system prune`
 - `docker volume rm $(docker volume ls -q)`

Compose ファイル形式バージョン

- networks: volumes: は v2 , v3 で利用可能
- v3 を使うには Docker v1.13 以上
- Swarm mode を使うには v3 (同じオプションでも v2 と挙動が異なる)
- 細かな差違はリファレンス

<https://docs.docker.com/compose/compose-file/>

env_file & environment:

services:

wordpress:

image: wordpress

restart: always

ports:

- 8080:80

environment:

WORDPRESS_DB_HOST: db

WORDPRESS_DB_USER: exampleuser

WORDPRESS_DB_PASSWORD:

examplepass

WORDPRESS_DB_NAME: exampledb

services:

wordpress:

image: wordpress

restart: always

ports:

- 88:80

env_file:

- **wordpress.env**

WORDPRESS_DB_HOST=db

WORDPRESS_DB_USER=exampleuser

WORDPRESS_DB_PASSWORD=examplepass

WORDPRESS_DB_NAME=exampledb

arg

```
$ docker-compose build --build-arg="text=hello"
```

```
version: '3'
```

```
services:
```

```
  apache:
```

```
    build:
```

```
      context: ./service-httpd/
```

```
      dockerfile: Dockerfile.httpd
```

```
      args:
```

```
        - text=plain
```

```
    image: localhttpd:1.0
```

```
    ports:
```

```
      - "80:80"
```

```
FROM httpd:2.4-alpine
```

```
RUN apk add tzdata && cp /usr/share/zoneinfo/Asia/Tokyo /etc/localtime && apk del tzdata
```

```
ARG text
```

```
RUN echo $text > /usr/local/apache2/htdocs/index.html
```

Orchestration

オーケストレーションと Swarm mode

Cloud Native 参照アーキテクチャ



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



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

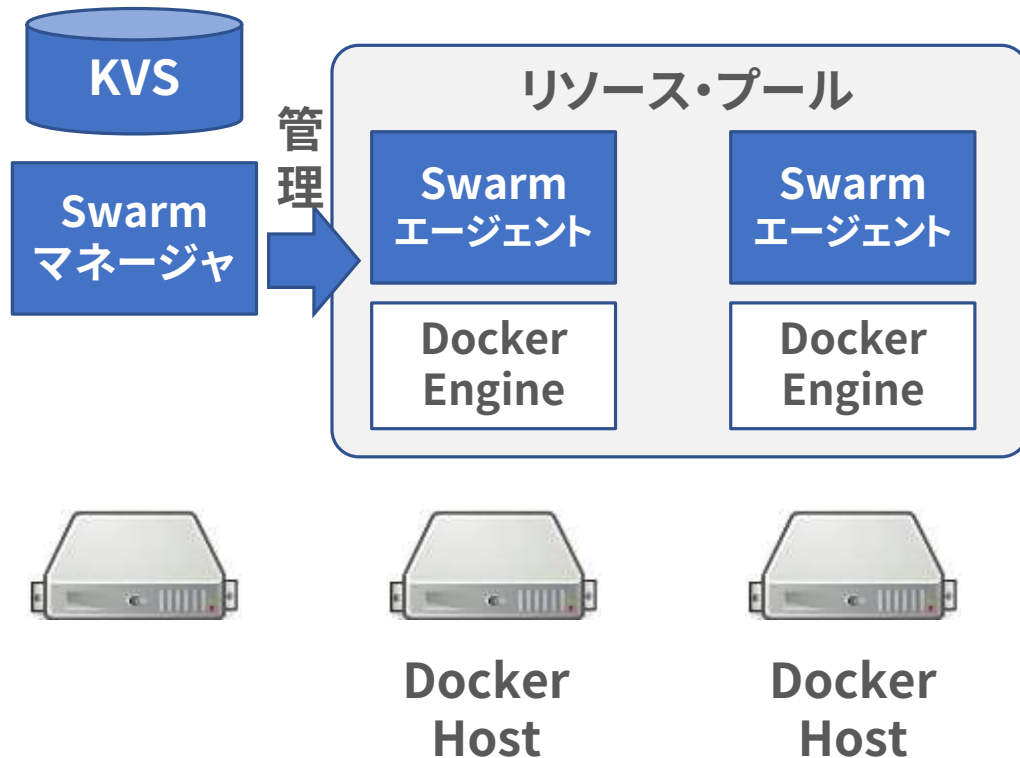
Δ = 状態から期待状態への収束



Docker Swarm vs Docker Swarm mode

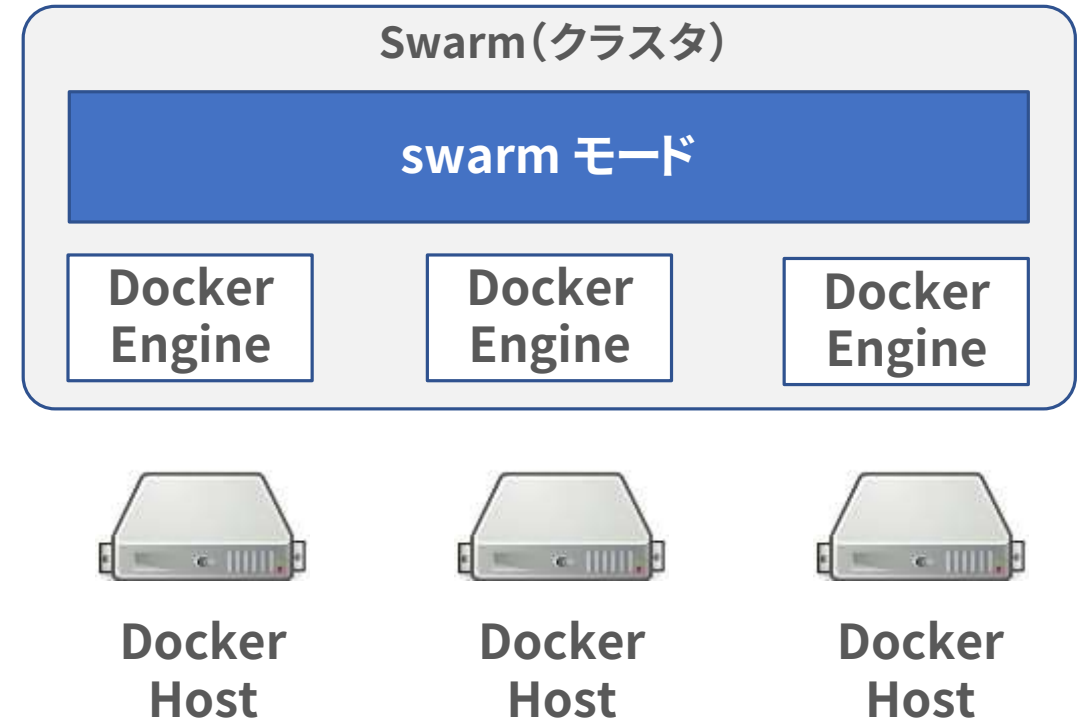
standalone swarm

管理用マネージャ・エージェントや KVS が別途必要

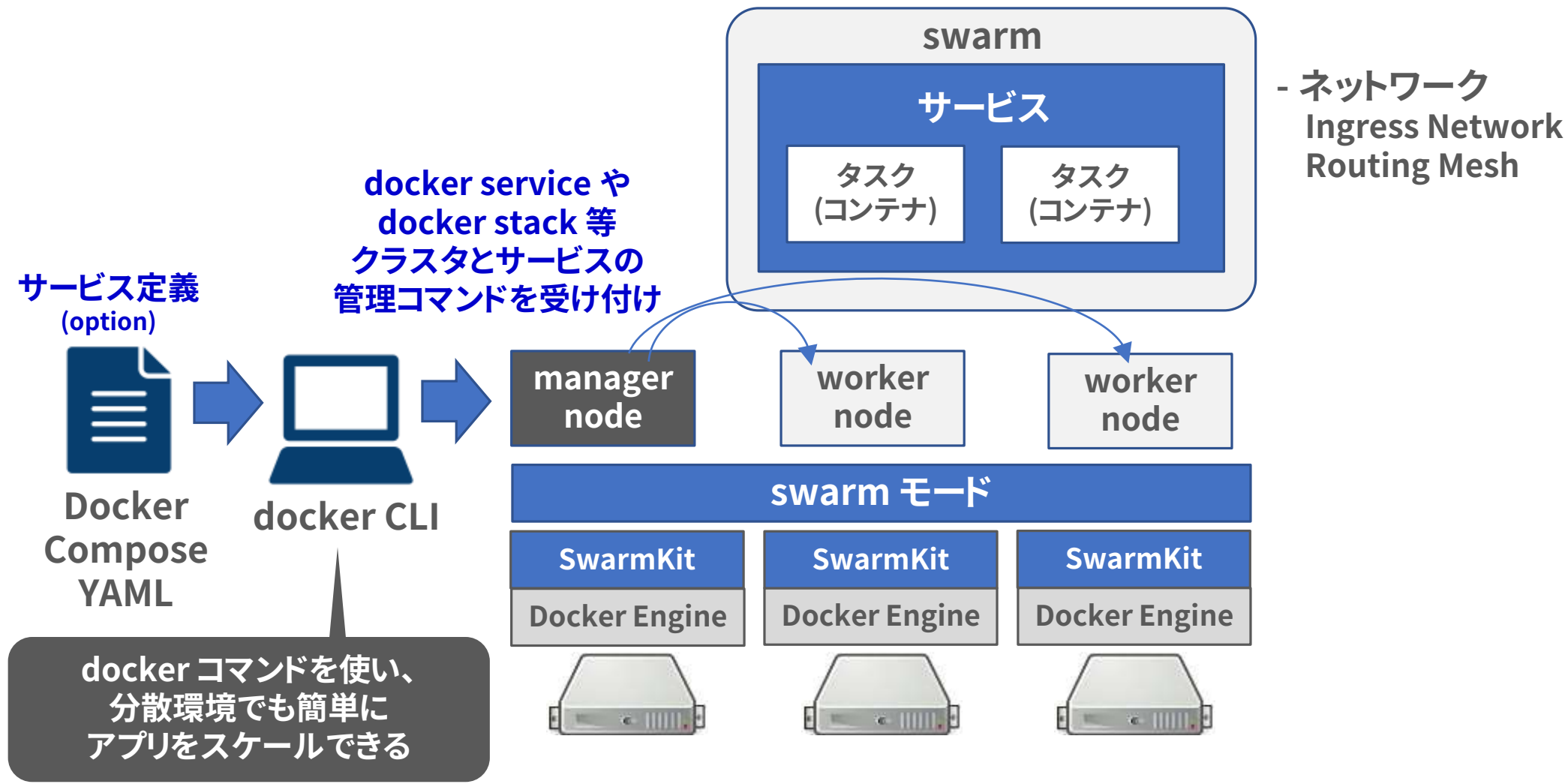


SwarmKit

Docker 1.12からクラスタ管理機能を内蔵



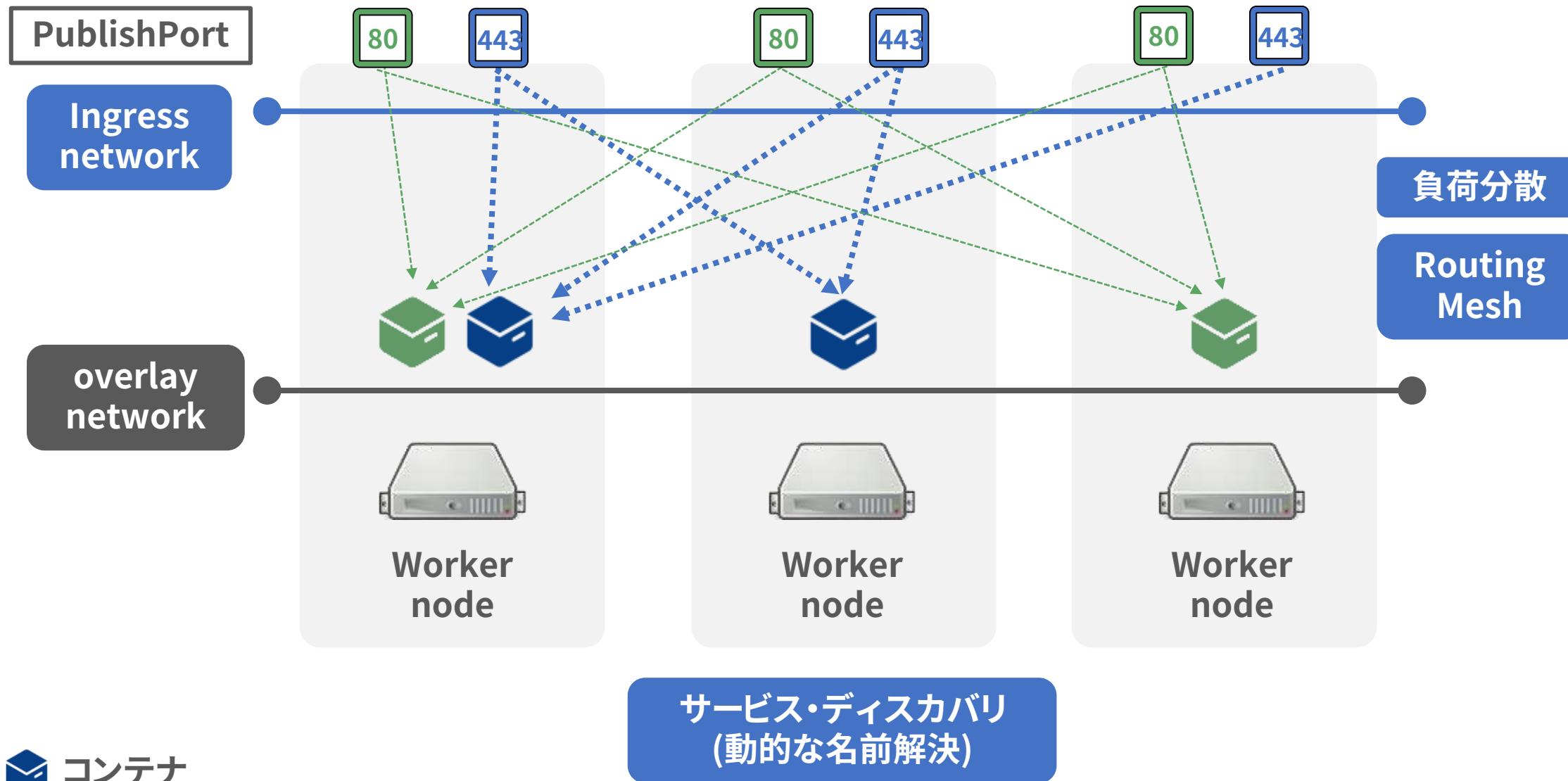
Swarm mode の構成要素



docker service や docker stack 系コマンドの実行は、kubectlと互換性を持つ
※ Docker for Mac で Experimental かつ Kubernetes 有効化の場合

swarm mode のネットワーク機能

Multi Host Networking



クラスタ初期化と サービス実行

Dockerのセットアップ

Linuxで検証用途に手軽な方法

```
$ curl https://get.docker.com | head
```

※コマンドの確認



```
$ curl -fsSL get.docker.com -o get-docker.sh  
# sh ./get.docker.com
```

※リポジトリの自動セットアップ



```
# systemctl enable docker  
# systemctl start docker
```

※systemd系の場合

クラスタ初期化と join 手順



manager

```
$ docker swarm init
```

Swarm initialized: current node (hhzcdnj2r43ywjcmjcwbgvwa7) is now a manager.

To add a worker to this swarm, run the following command:

```
docker swarm join --token SWMTKN-1-0w2m8k41xh1tbapub..... <IP_ADDR>:2377
```

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.



worker

```
$ docker swarm join ¥  
--token SWMTKN-1-0w2m8k41xh1tbapub..... <IP_ADDR>:2377
```

managerのIPアドレスとポート

This node joined a swarm as a worker.

クラスタ join 時のトークン確認



manager

```
$ docker swarm join-token worker
```

To add a worker to this swarm, run the following command:

```
docker swarm join --token SWMTKN-1-0w2m8k41xh1tbapubwl7sd7j7x.... <IP_ADDR>:2377
```



manager

```
$ docker swarm join-token manager
```

To add a manager to this swarm, run the following command:

```
docker swarm join --token SWMTKN-1-0w2m8k41xh1tbapubwl7sd0m0.... <IP_ADDR>:2377
```

クラスタ状態確認



manager

\$ docker node ls

\$ docker node ls

ID	HOSTNAME	STATUS	AVAILABILITY	MANAGER STATUS
hhzcdnj2r43ywjcmjcwbgvwa7 *	node-01	Ready	Active	Leader



\$ docker node ls

ID	HOSTNAME	STATUS	AVAILABILITY	MANAGER STATUS
hhzcdnj2r43ywjcmjcwbgvwa7 *	node-01	Ready	Active	Leader
znmguxtqwyywhja9chkkaa6a7y	node-02	Ready	Active	
2mgqqmgt0dlv9zc932nz9rkat	node-03	Ready	Active	

Compose file でサービス作成・操作



manager

```
$ vi docker-compose.yml
```

```
$ docker stack deploy -c ./docker-compose.yml  
demo
```

```
Creating network demo_internal  
Creating service demo_web
```

```
$ docker stack ls  
$ docker stack ps demo  
$ docker service scale demo_web=5  
$ docker service stop demo
```

```
$ docker stack rm demo
```

```
version: '3'
```

```
services:
```

```
  web:
```

```
    image: zembutsu/docker-sample-nginx
```

```
    deploy:
```

```
      replicas: 3
```

```
      resources:
```

```
        limits:
```

```
          cpus: "0.1"
```

```
        restart_policy:
```

```
          condition: on-failure
```

```
    ports:
```

```
      - "80:80"
```

```
    networks:
```

```
      internal:
```

```
        aliases:
```

```
          - web
```

```
    volumes:
```

```
      - /etc/localtime:/etc/localtime:ro
```

```
networks:
```

```
  internal:
```

まとめ

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

Dockerとは？

Why Docker?

Containerization Namespace・Cgroup
プロセスを簡単にコンテナ化(isolate)し、
「プロセス・ファイルシステム・ネットワーク・等々」に対して

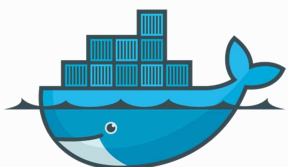
Build Ship Run

簡単かつ素早く開発・移動・実行できるプラットフォームが Docker

1

Dockerコンテナは
実行に必要な全て
をパッケージして、
簡単に動かせる

- アプリケーションを簡単に開発し、移動し、実行するためのプログラムとプラットフォームを提供するのが Docker
- クライアント・サーバ型



container



<https://docker.com>

2

Dockerイメージは
複数のイメージ・レイ
ヤとメタ情報の積み
重なり

- イメージ・レイヤ (image layer) は読み込み専用
- 親子関係がある
- イメージに対する変更は Copy on Write (CoW) 処理が走る
- コンテナ実行にはイメージが必要で、Docker Hub から得られる
- コンテナ実行時のみ、読み書きが可能なレイヤを追加

3

コンテナのプロセス
はデフォルトで
isolate (隔離・分離)
された状態

- namespace (名前空間) でプロセス空間やファイルシステムやネットワーク等を分ける技術と、cgroups (コントロール・グループ) でリソースの利用上限を指定
- コンテナはポートをデフォルトで開かない
- ネットワークはブリッジ、ホスト、none の3種類
- ボリュームはコンテナ間でファイルシステムを共有できる。名前付き (named) とホスト・ボリューム

Docker Composeとは？

What is Docker Compose?

multi-container applications
複数のコンテナで構成するアプリケーションを

define and run
定義と実行するためのツール

1

Composeは
アプリケーションの
サービスをファイル
で定義する

- Compose ファイル (YAML形式) で Docker コンテナのサービスを定義できるため、再利用性が高い

- コンテナの状態
- イメージ
- ネットワーク
- ボリューム
- メタ情報



2

Dockerコマンドと
高い親和性がある
ため、学習コストが
比較的に低い

- docker-compose CLI のコマンド体系は docker に準拠
 - 例: 「docker run -d」は「docker-compose up -d」

3

Swarm モードに
サービスをデプロイ
できるオーケストレー
ション機能

- Docker swarm モードを使えば、サービス状態を定義できるので常に期待状態を維持できる
 - docker stack deploy
- Ingress ネットワークの活用

<https://github.com/docker/compose>

Q&A

- 何か気になるところがありますか？
m-zembutsu@sakura.ad.jp
- 過去の資料
<https://slideshare.net/zembutsu>
- Dockerドキュメント日本語訳
<http://docs.docker.jp>
- 公式ドキュメント
<https://docs.docker.com>

