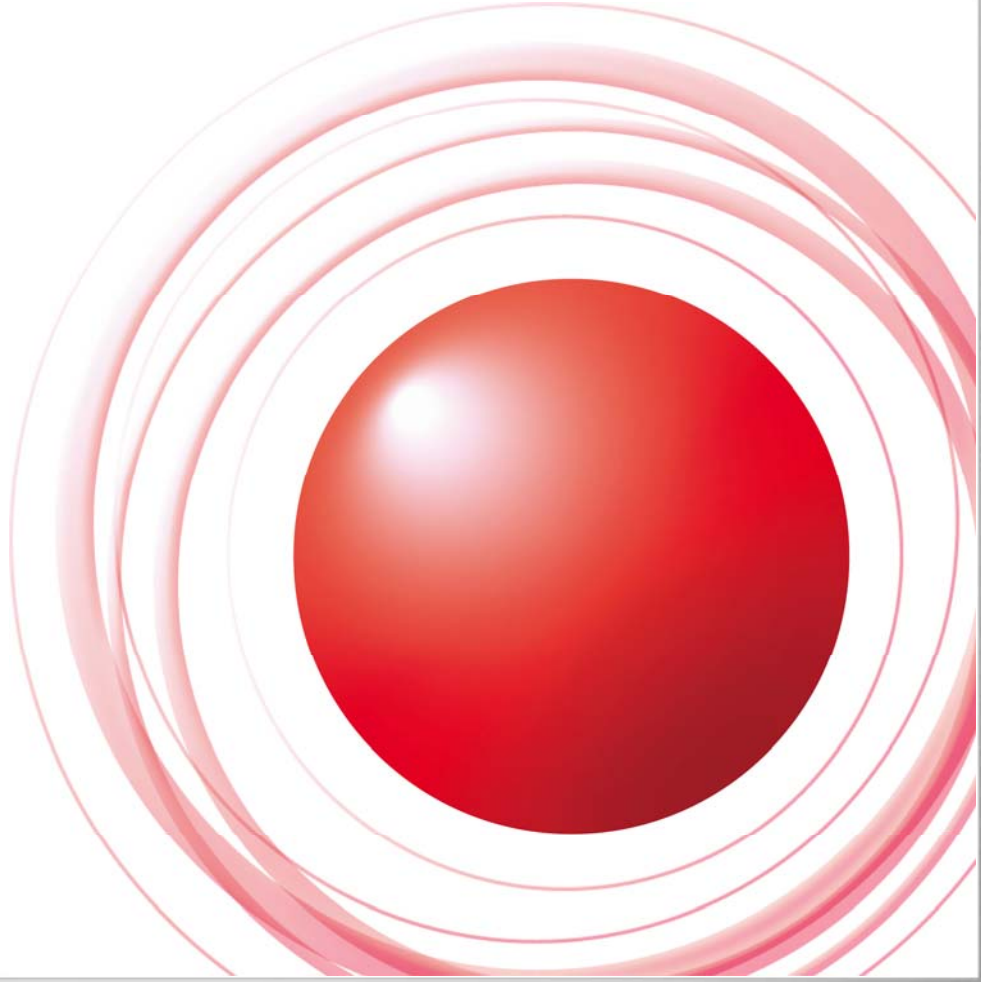
Dockerの概要とIIJでの利用例

IIJ
Internet Initiative Japan



前橋 孝広
maebashi@iij.ad.jp
株式会社インターネットイニシアティブ

Ongoing Innovation

本日の話

- Dockerとは
  - 概要
  - Dockerfile
  - Docker Hub
- 仕組み
- IIJでの利用例

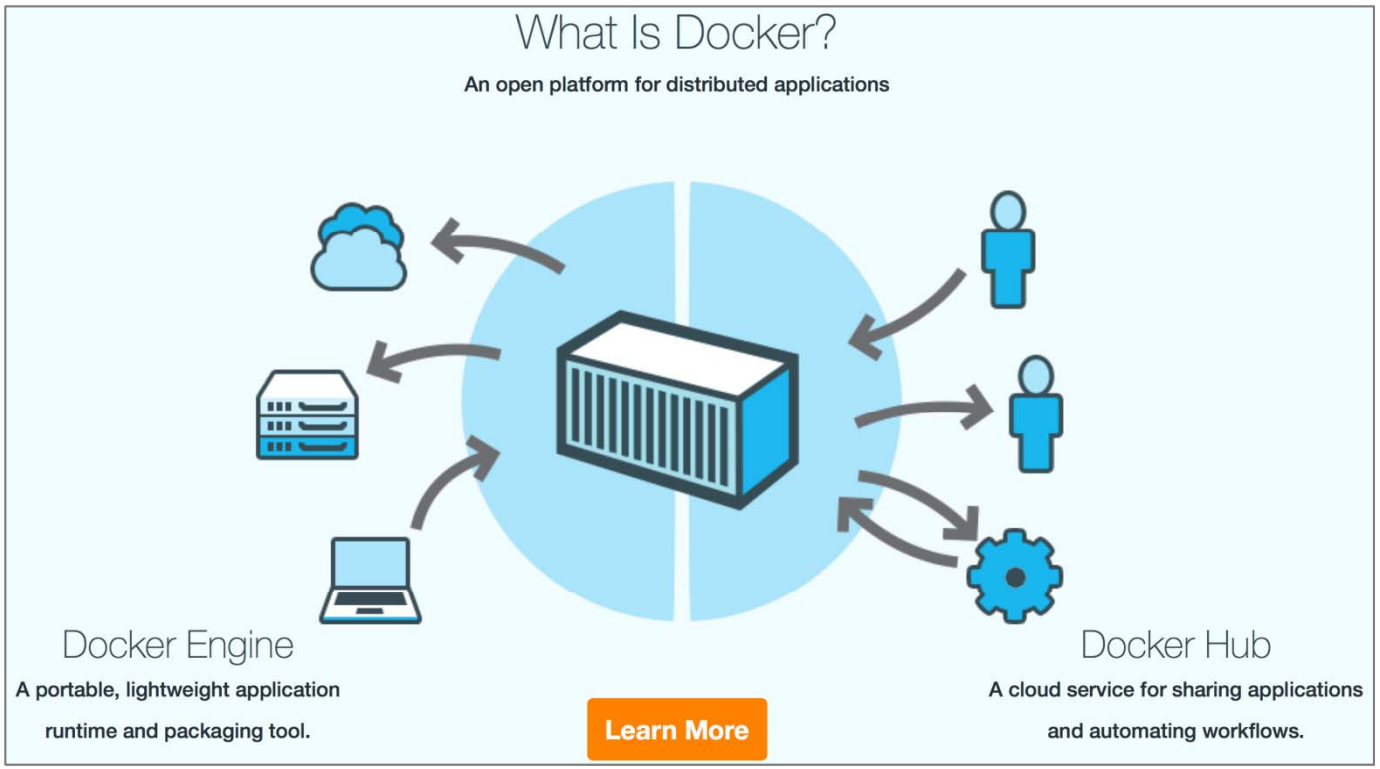
特に断りがない限り、本資料の説明は
CentOS 6.5/6.6 + docker 1.3.1を前提としている

Dockerとは?

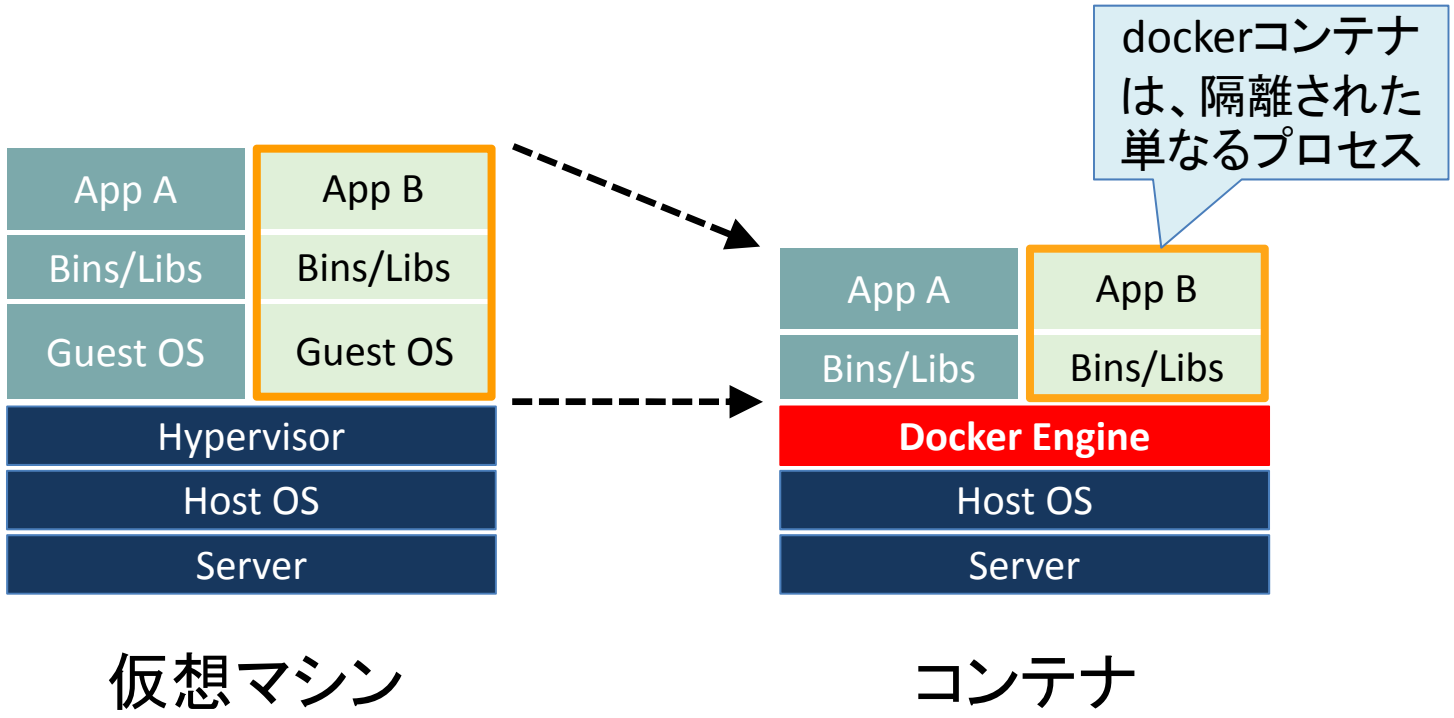
- オープンソースのコンテナ管理ソフトウェア
  - Linuxホストの中に、別の独立したLinux環境を作り出す
- 技術的に新しいものではない
  - 「chroot + α」とか「 jail + α」とかよく言われる
  - ディスクイメージ管理と、アプリを含めたイメージのパッケージング方式がキモ

本家(www.docker.com)の説明

Docker (プラットフォーム) = Docker Engine and Docker Hub



Hypervisor型仮想化 vs コンテナ



何がうれしいのか?

- 必要なアプリを含めたイメージを簡単に作れる
  - それをDocker HUBでソーシャルに共有
- ディスクイメージはgitのコミットグラフのように管理される
  - 差分だけ新たなディスク領域を使用

実行環境を高速に再現可能

インストール(CentOS6.5の場合)

```
$ sudo rpm -ivh
http://ftp.iij.ad.jp/pub/linux/fedora/epel/6/x86
_64/epel-release-6-8.noarch.rpm
$ sudo yum install docker-io
(/etc/sysconfig/docker編集)
$ sudo service docker start
```

/etc/sysconfig/dockerの例

```
other_args="-H=unix:///var/run/docker.sock -e native"
ulimit -n 1048576
ulimit -u unlimited
```

無保証版 docker 1.3.1 rpm

CentOS 6.5/6.6 (64bit版) 用

```
$ wget https://github.com/maebashi/docker-rpm-
el6/releases/download/v1.3.1/docker-io-1.3.1-
1.el6.x86_64.rpm.zip
$ unzip docker-io-1.3.1-1.el6.x86_64.rpm.zip
$ sudo rpm -ivh docker-io-1.3.1-1.el6.x86_64.rpm
(/etc/sysconfig/docker編集)
$ sudo service docker start
```

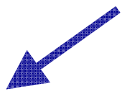
ソース(specファイル、Dockerfileなど):
https://github.com/maebashi/docker-rpm-el6

Docker Hub:
https://registry.hub.docker.com/u/maebashi/docker-rpm-el6/

Hello, world

docker pull

（イメージを取ってくる）



リポジトリ名:タグ名
（イメージ名）

```
# docker pull centos:centos6
# docker images
REPOSITORY  TAG       IMAGE ID      CREATED       VIRTUAL SIZE
centos      centos6   70441cac1ed5  8 hours ago   215.8 MB
```

docker run

（コンテナ起動）

```
# docker run centos:centos6 echo Hello, world
Hello, world
#
```

bashの実行

```
$ docker run -i -t centos:centos6 /bin/bash
[root@a61514500d21 /]# ls
bin  home   lost+found  opt   sbin     sys  var
dev  lib    media       proc  selinux  tmp
etc  lib64  mnt         root  srv      usr
[root@a61514500d21 /]# yum install -y ruby
...
Installed:
  ruby.x86_64 0:1.8.7.374-2.el6

Dependency Installed:
  compat-readline5.x86_64 0:5.2-17.1.el6   ruby-libs.x86_64
0:1.8.7.374-2.el6
Complete!
[root@a61514500d21 /]#
```

shellを終了せずに tty から切り離すにはCtrl-P Ctrl-Q

イメージいろいろ

```
# docker images
REPOSITORY   TAG       IMAGE ID       CREATED        VIRTUAL SIZE
centos       centos6   70441cac1ed5   8 hours ago    215.8 MB
centos       centos7   ae0c2d0bdc10   8 hours ago    224 MB
centos       latest    ae0c2d0bdc10   8 hours ago    224 MB
ubuntu       14.04     5506de2b643b   11 days ago    197.8 MB
ubuntu       latest    5506de2b643b   11 days ago    197.8 MB
```

- 同じイメージ名(リポジトリ:タグ)でも、pullする時期によってイメージ内容は異なることがある
- 自分で(既存イメージを継承せずに)イメージを作る方法もある

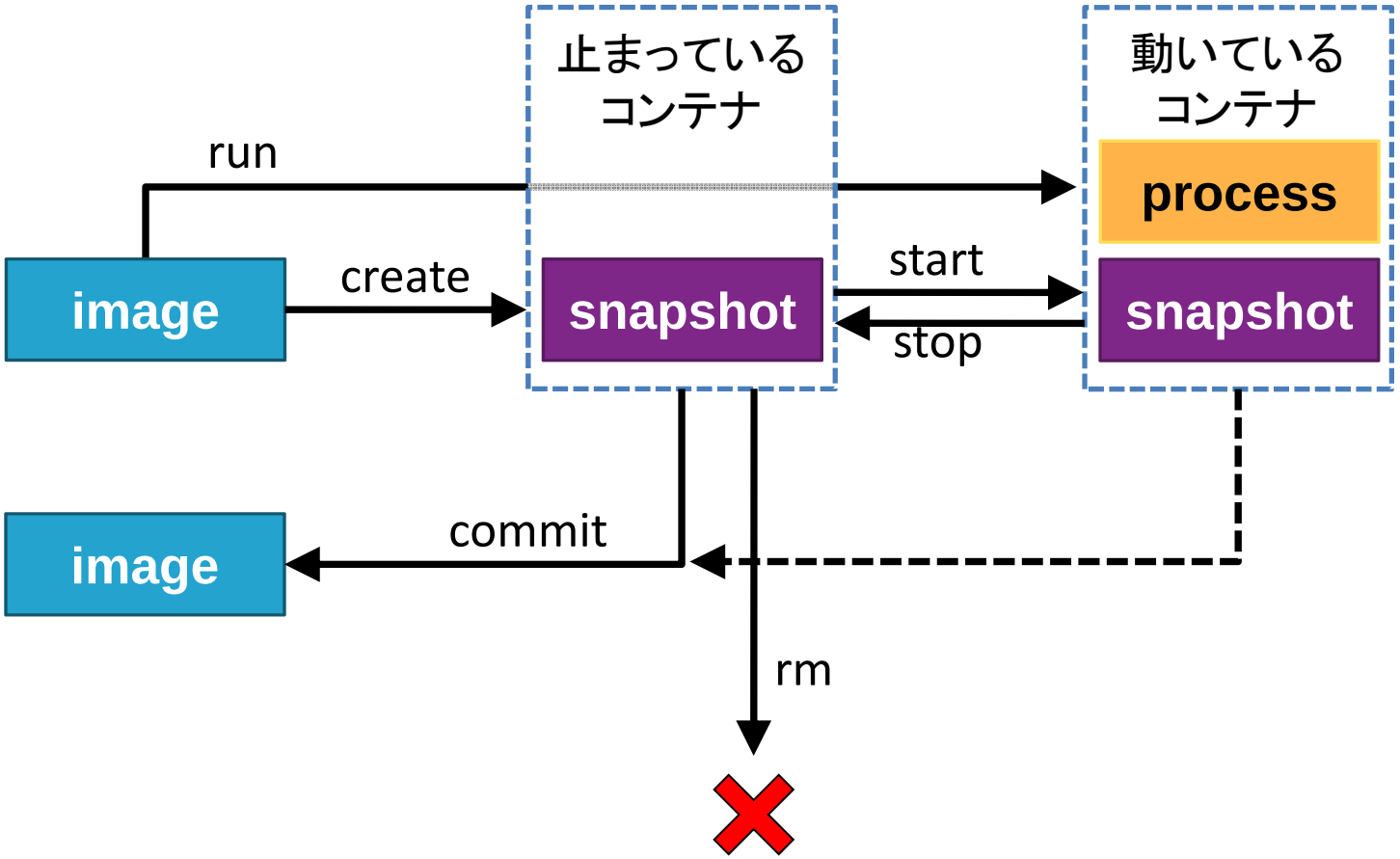
イメージの格納場所

CentOS 6.5の場合 /var/lib/docker/devicemapper/devicemapper/

```
# cd /var/lib/docker/devicemapper/devicemapper/
# ls -lh
total 550M
-rw-------. 1 root root 100G Nov  4 14:59 data
-rw-------. 1 root root 2.0G Nov  4 14:59 metadata
#
```

- 上記dataファイルがブロックプール
  - data(とmetadata)の中に複数のイメージを格納
  - devicemapper(dm)により論理デバイスとして使える
  - dmで論理デバイスのスナップショットを作れる
  - イメージは合計100Gバイトまで(変更可)

コンテナのライフサイクル



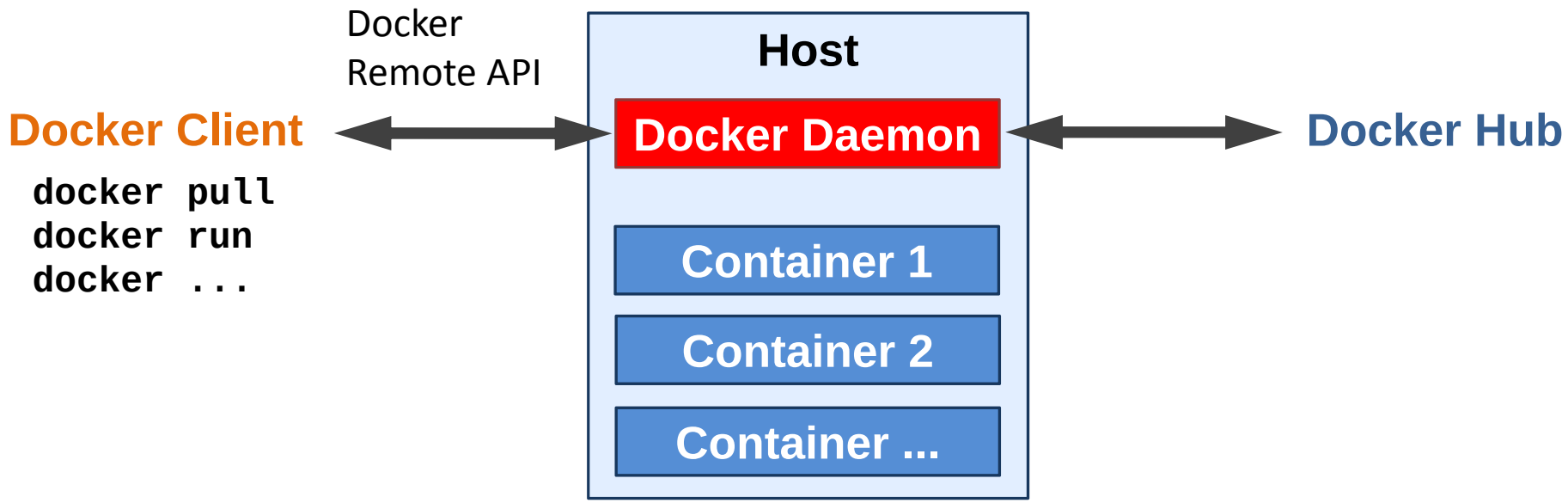
dockerコンテナのネットワーク

- (デフォルトでは)コンテナ毎に個別のEthernetインタフェースが作られ、(ホスト側で使っていない)プライベートIPアドレスが振られる

```
$ docker run -i -t centos:centos6 /bin/bash
bash-4.1# ip a
4: lo: <LOOPBACK,UP,LOWER_UP> mtu 1500 qdisc noqueue state
UNKNOWN
...
5: eth0: <BROADCAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast
state UP qlen 1000
    link/ether aa:22:97:e5:b5:b5 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.2/16 scope global eth0
    inet6 fe80::a822:97ff:fee5:b5b5/64 scope link
       valid_lft forever preferred_lft forever
bash-4.1#
```

- IPマスカレードで、ホスト経由で外に出て行く

dockerクライアントとdockerデーモン



Dockerfile

Dockerfile

- dockerイメージの構成内容を記述するテキストファイル
  - OS、ライブラリ、アプリケーションをパッケージング
  - Dockerfileにより実行環境を再現可能
  - インフラをコードとして扱える

DockerfileとDocker build

- Dockerfileの例(centos6ベースでrubyを入れたイメージを作る)

```
FROM centos:centos6
RUN yum install -y ruby
```

- docker build
  - 上記内容のDockefileを作り...

```
# docker build -t ruby - < Dockerfile
...
Complete!
 ---> b3a4de744050
Removing intermediate container 04afb30a1c34
Successfully built b3a4de744050
```

docker buildでやっていること

- FROMで指定されたイメージがなければpull
- RUNやADD等の行ごとに
  - コンテナを起動
    - ディスクイメージは直前の行で生成されたイメージのスナップショット
  - RUNやADDの実行
  - コンテナを停止(ディスクイメージは残っている)
- 最後に出来たイメージに名前をセットする

buildしたイメージの確認

build前

```
# docker images
REPOSITORY  TAG      IMAGE ID      CREATED         VIRTUAL SIZE
centos      centos6 70441cac1ed5  8 hours ago     215.8 MB
# du -sh /var/lib/docker/devicemapper/devicemapper/*
549M    /var/lib/docker/devicemapper/devicemapper/data
976K    /var/lib/docker/devicemapper/devicemapper/metadata
```

build後

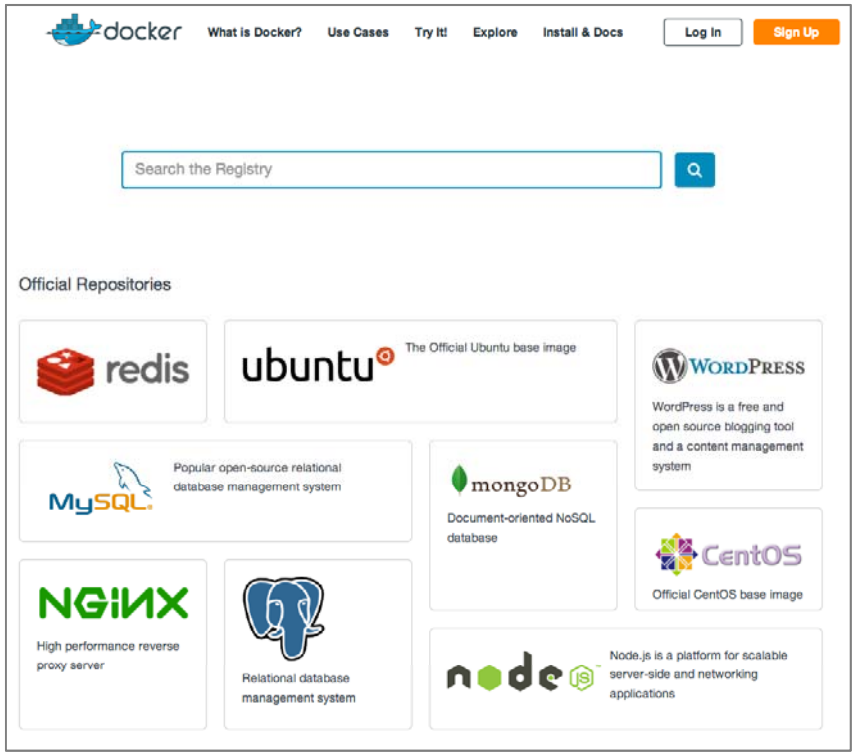
```
# docker images
REPOSITORY  TAG      IMAGE ID      CREATED         VIRTUAL SIZE
ruby        latest   583c11ebc182  24 seconds ago 253.7 MB
centos      centos6 70441cac1ed5  8 hours ago     215.8 MB
# du -sh /var/lib/docker/devicemapper/devicemapper/*
590M    /var/lib/docker/devicemapper/devicemapper/data
1.4M    /var/lib/docker/devicemapper/devicemapper/metadata
```

Docker Hub

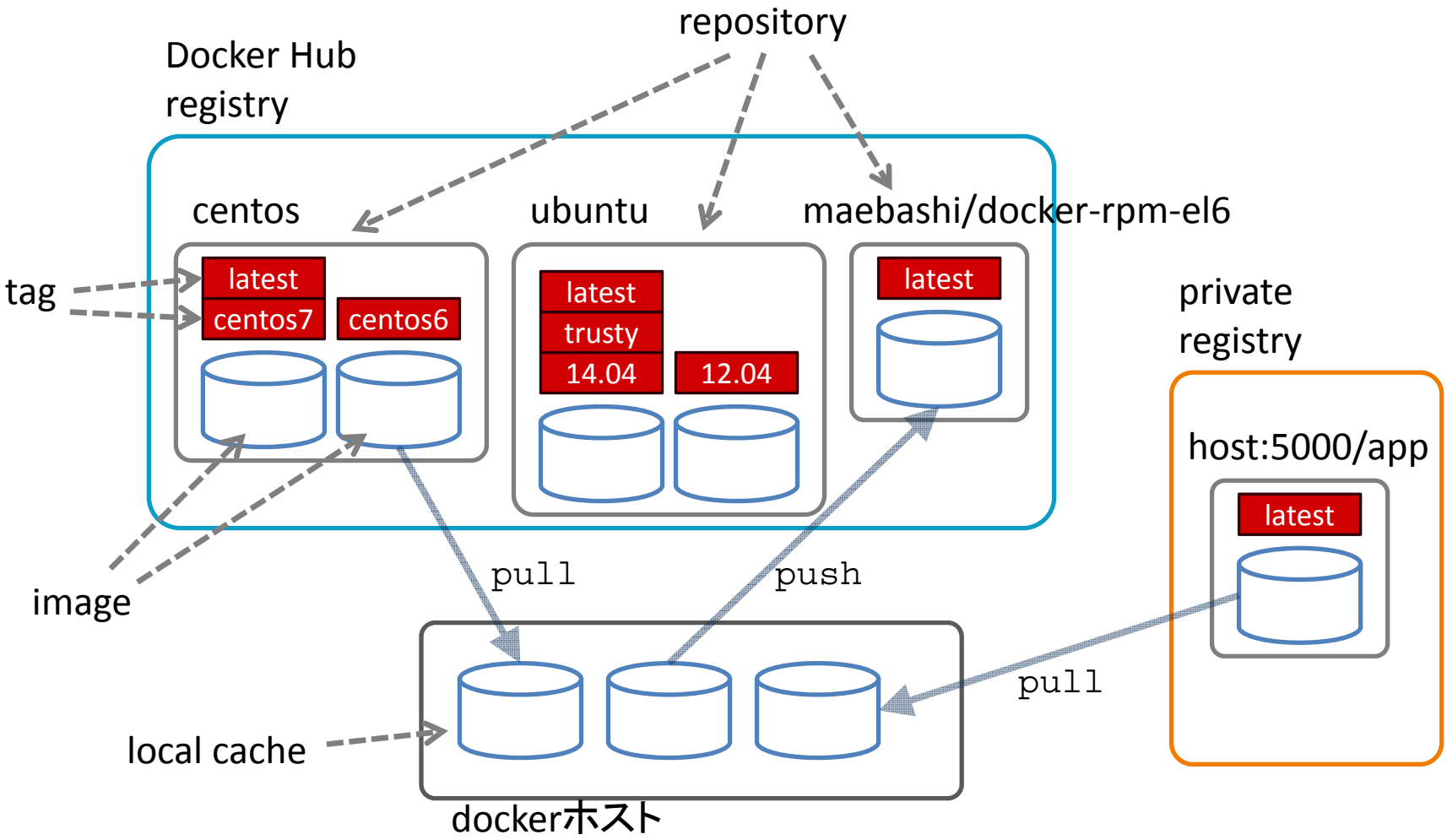
Docker Hub とは?

- dockerのイメージを共有出来るサービス

https://registry.hub.docker.com/

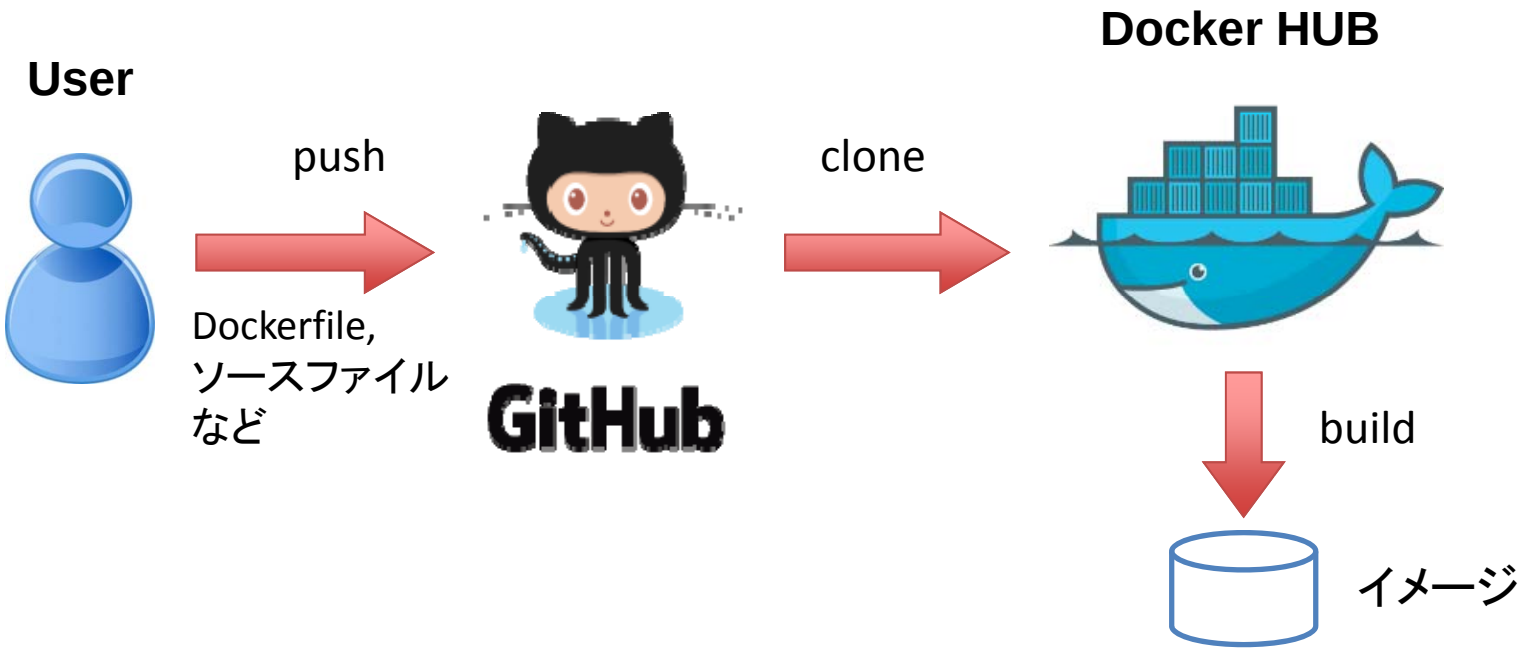


registry, repository 概念図



Docker Hub Automated Build

- GitHub/Bitbucketと連動し、自動的にイメージをbuildする



Automated Build の利用例

- 無保証版docker rpmは、Docker Hub上でAutomated Buildされている
  - ↓これでrpmを取り出すことができる

```
# docker run maebashi/docker-rpm-el6 tar cf – ¥
-C /rpmbuild RPMS | tar xf -
```

  - Docker Hubからmaebashi/docker-rpm-el6イメージ取得
  - このイメージのbuild中にrpmが生成されるようになっているので、それを取り出すだけ

docker-rpm-el6のDockerfile

```
FROM centos:centos6

RUN yum install -y tar git hg rpmdevtools gcc glibc-static device-
mapper-devel
RUN rpm -ivh
http://ftp.iij.ad.jp/pub/linux/fedora/epel/6/x86_64/epel-release-6-
8.noarch.rpm
RUN yum install -y pandoc 'golang(github.com/gorilla/mux)'
'golang(github.com/kr/pty)'
'golang(code.google.com/p/go.net/websocket)'
'golang(code.google.com/p/gosqlite/sqlite3)'
'golang(github.com/syndtr/gocapability/capability)'
'golang(github.com/godbus/dbus)' 'golang(github.com/coreos/go-
systemd)' 'golang(github.com/tchap/go-patricia/patricia)'

ADD rpmbuild /rpmbuild
RUN wget -P /rpmbuild/SOURCES
https://github.com/docker/docker/archive/v1.3.1.tar.gz
RUN rpmbuild -bb /rpmbuild/SPECS/docker-io.spec -D "_topdir
/rpmbuild"
```

Docker Engineの仕組み

コンテナとは何か?

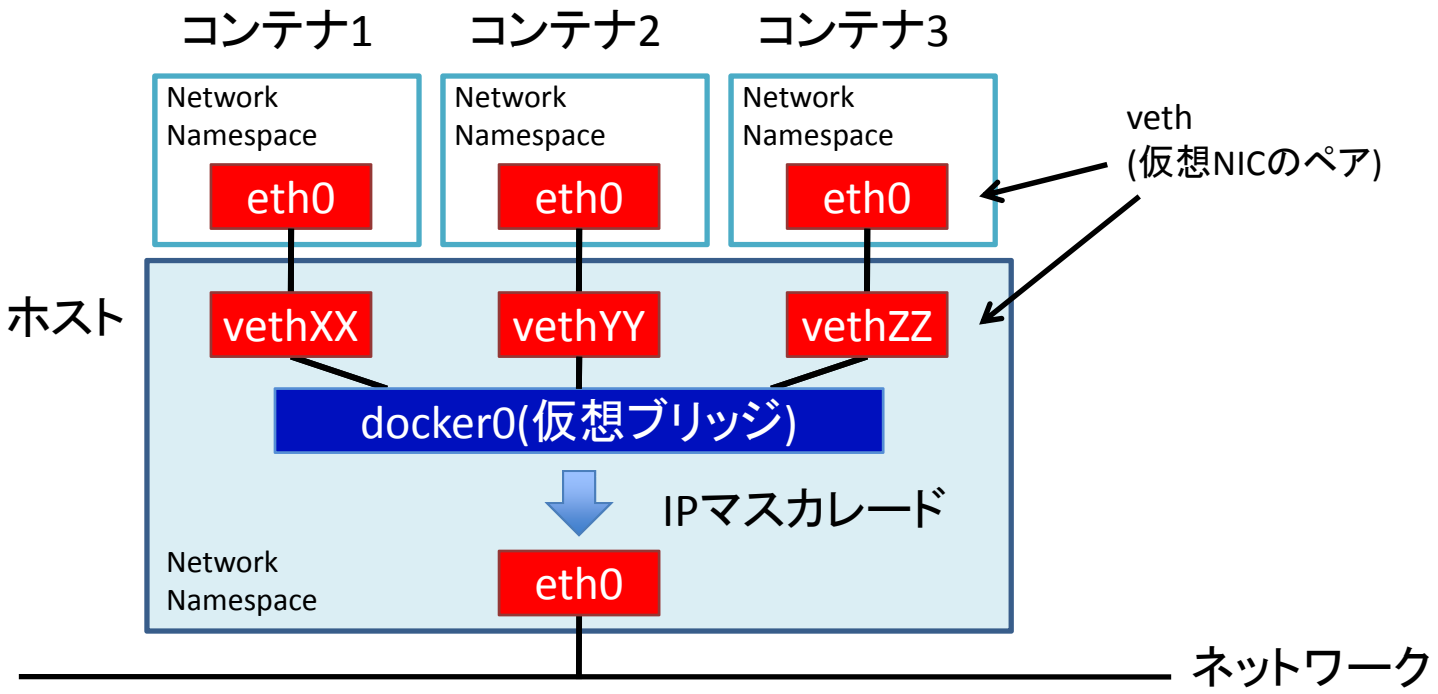
- 複数のLinux標準機能の組み合わせで実現する隔離環境
  - Namespaces – ネットワーク環境やファイルシステム等の分離
  - Cgroups – メモリやCPUなどのリソース制限
  - Capabilities – 権限降格
  - iptables – ネットワークパケットフィルタ and NAT

Namespace

- コンテナ実現のための中核となる機能
- プロセスが動作する空間を分離する
  - いくつかの種類がある
    - Network Namespace – ネットワーク環境の分離
    - Mount Namespace – ファイルシステムの分離
    - PID Namespace – プロセステーブルの分離
    - UTS Namespace – hostnameの分離
    - IPC Namespace – IPCの分離
    - User Namespace – UID/GIDの分離(docker 1.3では未使用)

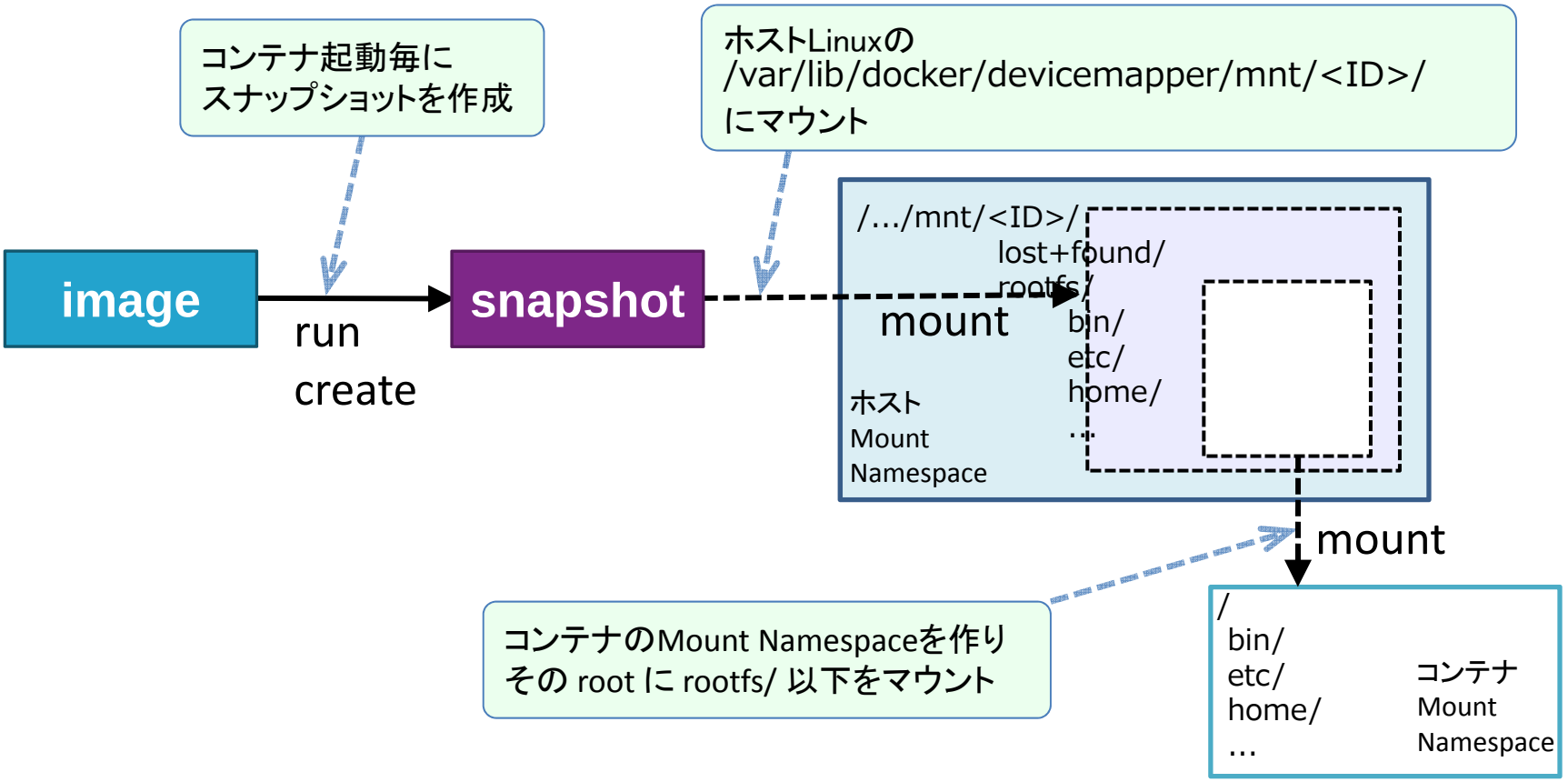
Network Namespace

- コンテナ毎にネットワーク環境を分離
  - 分離しないこともできる(ホストと共有)



Mount Namespace

- コンテナ毎にファイルシステムを分離



Mount Namespace (2)

- 一部のファイルは個別にbind mountされている (docker 1.2より前はread only)
  - /etc/resolv.conf
  - /etc/hosts
  - /etc/hostname
- 実体との対応は docker inspect <ID> 等でわかる

cgroups

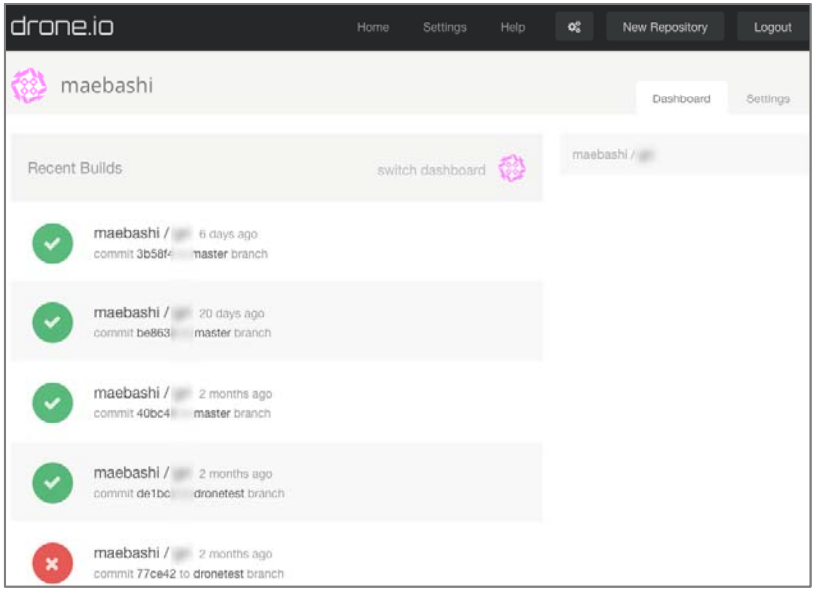
- プロセスグループのリソース(CPU、メモリ、ディスクI/Oなど)の利用を制限
  - dockerではcpu(set), memoryについて制限可
  - デバイスへのアクセスも制限
- リソース使用状況の統計値がとれる

IIJ社内での利用例

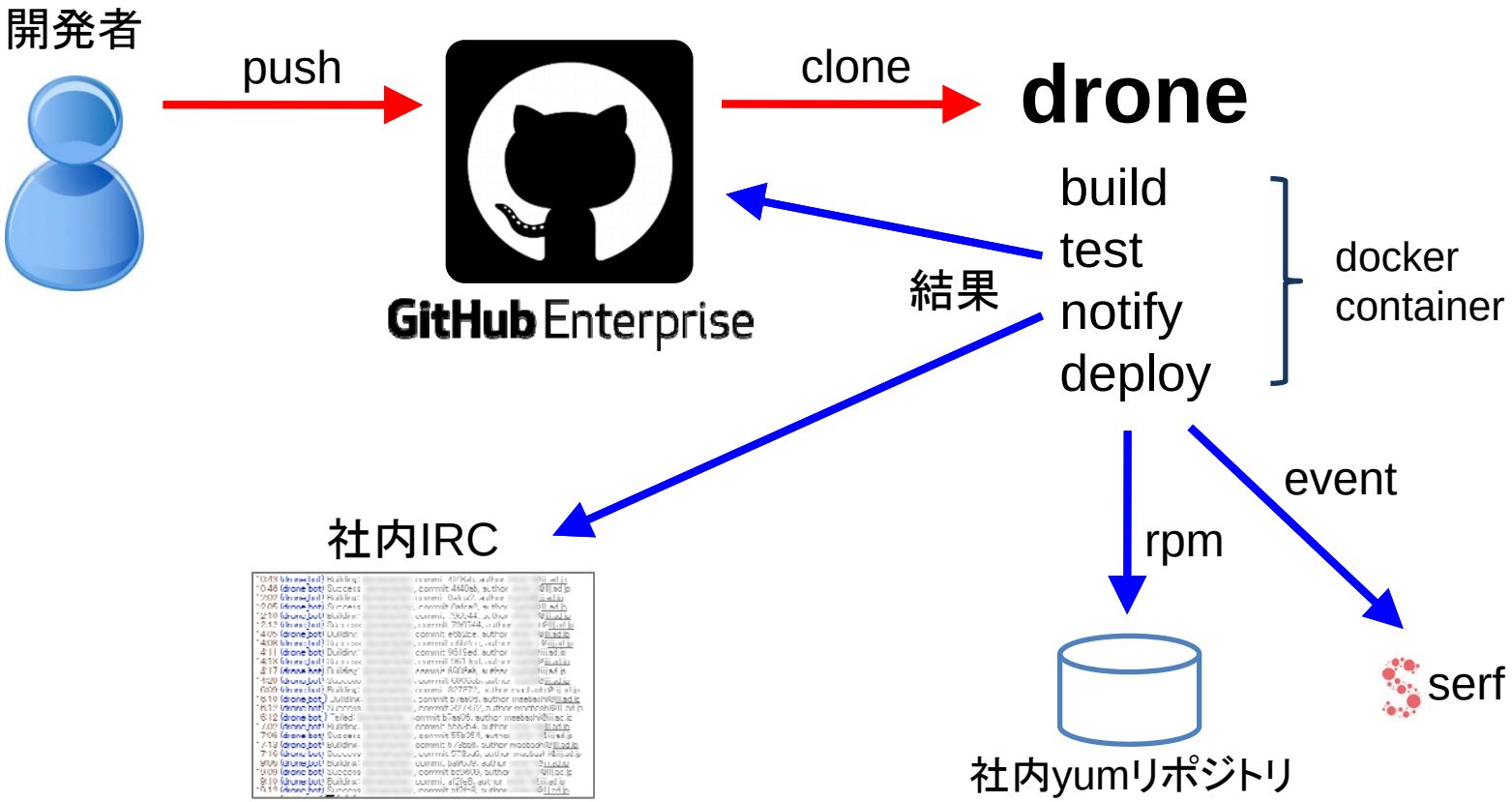
継続的インテグレーション
クラスタリング
モニタリング

継続的インテグレーション(CI)

- drone
  - オープンソースの CI サーバ
  - dockerコンテナ内のクリーンな環境でビルド、テストを実行する
  - GitHub/GitHub Enterpriseと連携

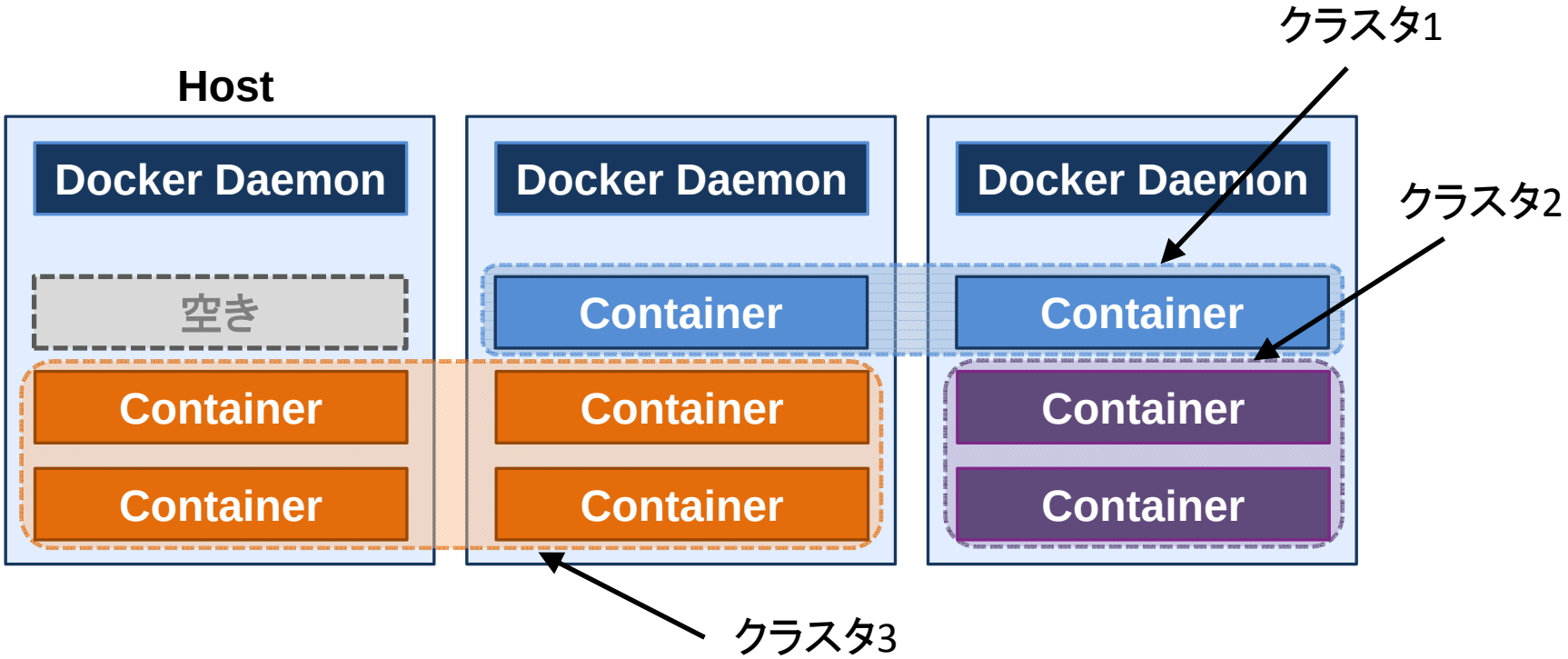


継続的インテグレーション: 流れ



コンテナのクラスタリング

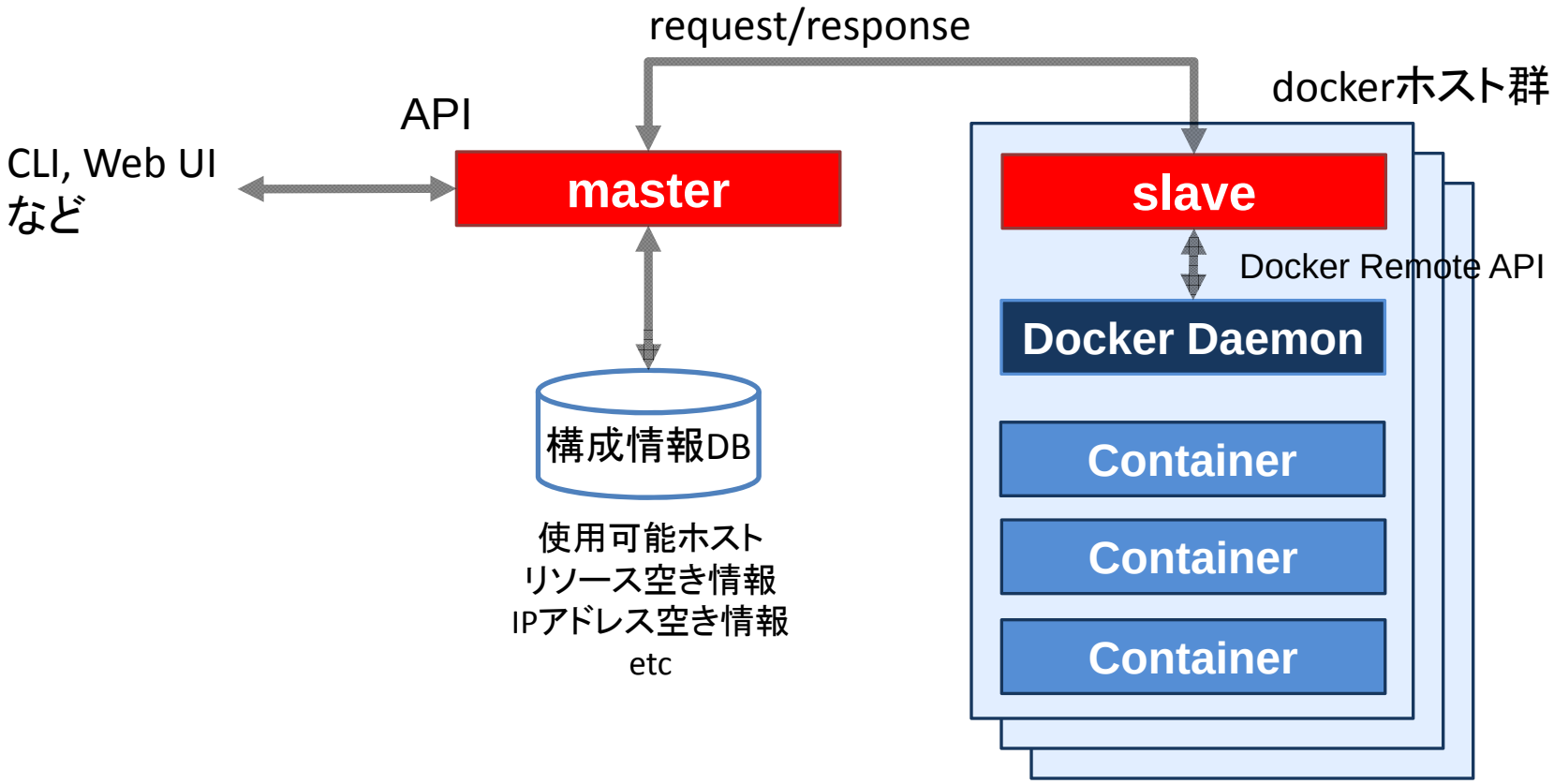
- 多数のDockerホスト上の多数のコンテナを管理したい



クラスタ管理ツール

- fig
  - 開発環境用、ホスト1台のみ対象
- Apache Mesos
  - 複数ホストのリソース管理
- Kubernetes
  - Google Container Engineで使われている
- flynn, dokku
  - PaaS用

IIJ内製 docker manager 構成図



dockerコンテナのモニタリング

- 個々のコンテナのメトリクスを収集したいがコンテナ毎にsnmpdやその他エージェントを入れたくない
- dockerコンテナのメトリクス収集
  - http://blog.docker.com/2013/10/gathering-lxc-docker-containers-metrics/
  - cgroupsの統計情報を使うことにより、ホスト上で(コンテナの外から)メトリクスを収集可能
  - コンテナのNetwork Namespaceに切り替えて/proc/net/devを参照することでネットワークの統計情報を得られる

docker-metricsd

- 各dockerホストに常駐、dockerコンテナの情報(cgroupsの統計情報等)を収集して返す
  - 似たようなもの
    - Google cAdvisor等

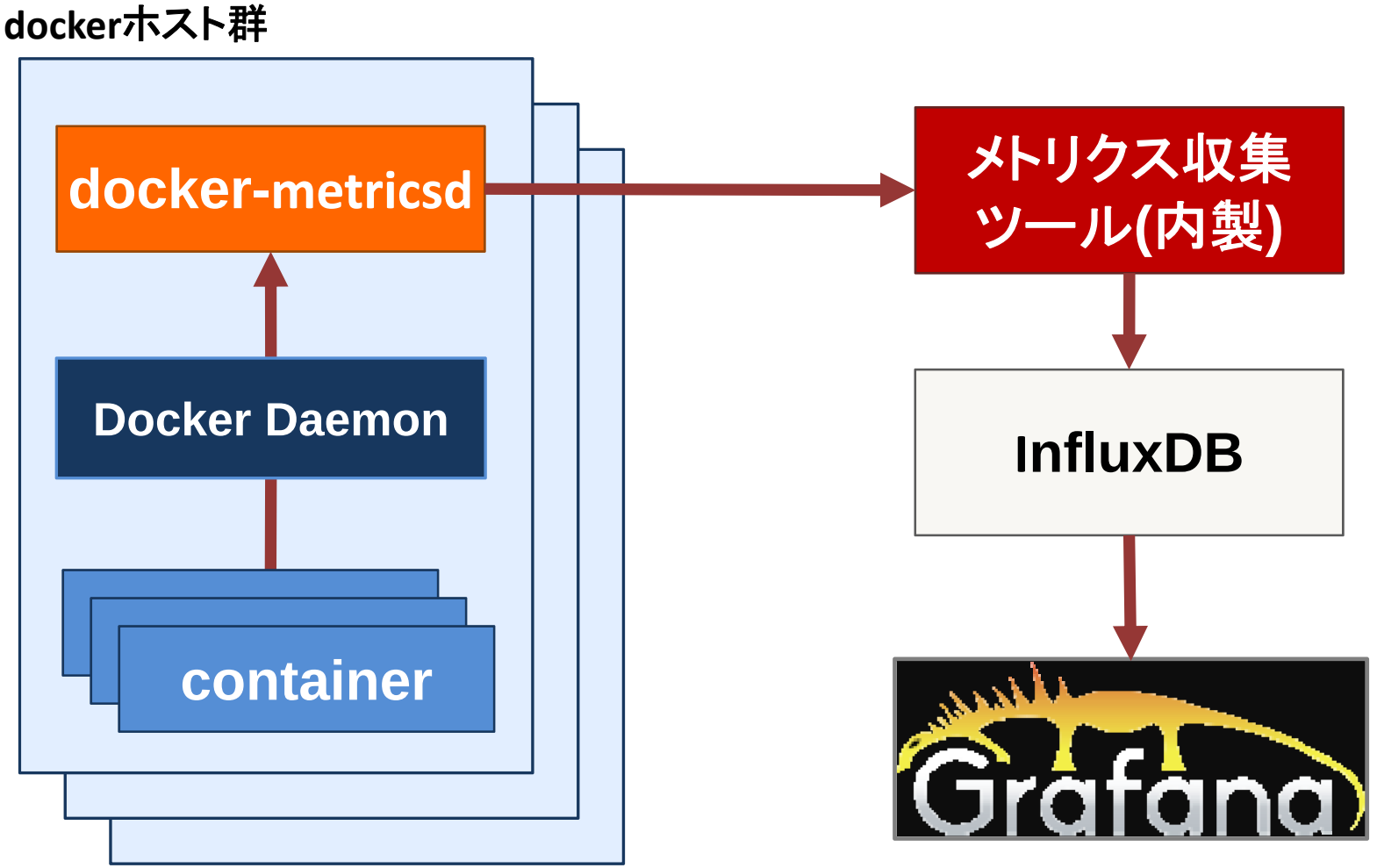
- インストール & 実行

```
# docker run -v /usr/local/bin:/target maebashi/docker-metricsd
# /usr/local/bin/docker-metricsd &
```

docker-metricsdはメトリクスをJSON形式で返す


```
"memory": {
      "failcnt": 0,
      "stats": {
        "unevictable": 0,
        "total_unevictable": 0,
        "total_swap": 0,
        "total_rss": 380928,
        "total_pgpgout": 681084,
        "total_pgpgin": 697086,
        "total_mapped_file": 1433600,
        "total_inactive_file": 38936576,
        "mapped_file": 1433600,
        "inactive_file": 38936576,
        "inactive_anon": 0,
        "hierarchical_memsw_limit":
9223372036854776000,
        "hierarchical_memory_limit":
9223372036854776000,
        "cache": 102838272,
        "active_file": 63901696,
        "active_anon": 380928,
        "pgpgin": 697086,
        "pgpgout": 681084,
        "rss": 380928,
        "swap": 0,
        "total_active_anon": 380928,
        "total_active_file": 63901696,
        "total_cache": 102838272,
        "total_inactive_anon": 0
      },
      "max_usage": 139268096,
      "usage": 104189952
    },
    "interfaces": {
      "outpackets.0": 3021223,
      "inbytes.0": 8228044607,
      "indrop.0": 0,
      "inerrs.0": 0,
      "inpackets.0": 6429687,
      "name.0": "eth0",
      "outbytes.0": 199687042,
      "outdrop.0": 0,
      "outerrs.0": 0
    },
    "cpuacct": {
      "throlling_data": {},
      "cpu_usage": {
        "usage_in_usermode": 2.668e+10,
        "usage_in_kernelmode": 8.181e+10,
        "percpu_usage": [
          22599918565,
          987624379,
          65146098,
          36705600,
          18221767943,
          5890326,
          22768005795,
          4033968,
          218211598,
          4302652,
          5437296326,
          23781278563,
          18992360915,
          18712487134,
          55742891,
          18522722054
        ]
      }
    }
```

定期的にメトリクスを収集してGrafanaでグラフ化



結果

↓CPU Accounting

↓Network traffic

Memory→

まとめ

- dockerにより実行環境を高速に再現できる
- Docker HUB便利
- コンテナは、Linux標準機能のNamespaces, Cgroups等を使用して実現
- IIJではコンテナ関連ツールをいくつか自作して利用