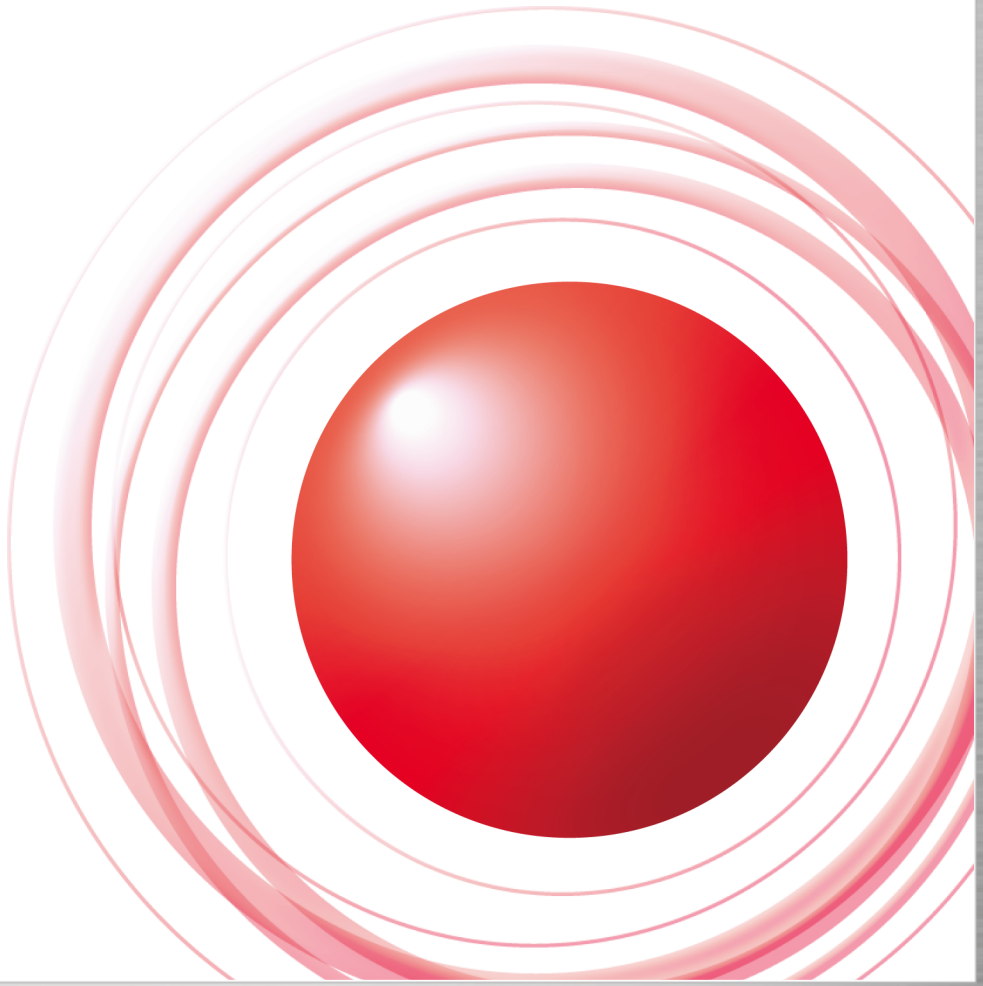
Dockerの商用サービスでの利用事例紹介

IIJ
Internet Initiative Japan



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

前置き	サービス概要 Dockerを商用サービスの基盤に使うにあたって
実装方法	多数のコンテナの管理 コンテナのリソース制限 コンテナ間のネットワーク その他

サービス概要

http://www.iij.ad.jp/biz/storage/

クラウド　メール・Web

ビッグデータ向け高品質クラウドストレージ

IIJ GIO

IIJ GIOストレージ&アナリシスサービス

IIJ GIO（ジオ）ストレージ&アナリシスサービスは、豊富なREST APIのインタフェースを持つ保存容量無制限のクラウドストレージサービスです。データの収集・蓄積・解析の環境をワンストップで実現できるため、ビッグデータの活用にも最適です。

ビッグデータ活用に最適なクラウドサービスのラインアップへ

- 堅牢かつ大容量保存のクラウドストレージ
- 初期費用0円。使用した分だけのお支払い
- ビッグデータ活用の実現

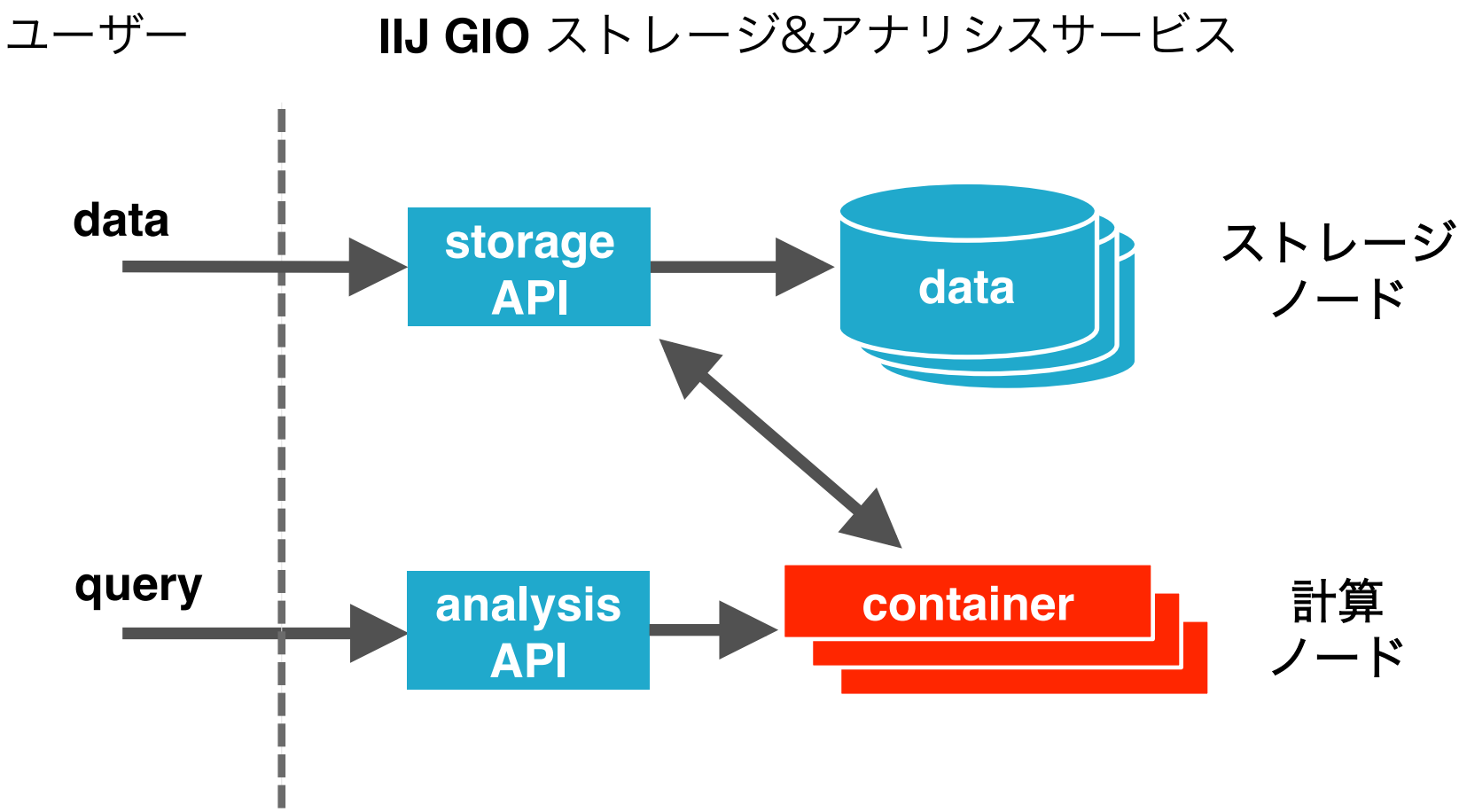
IIJ GIO ストレージ&
アナリシスサービス

REST API(AWS S3互換)を持つ
クラウドストレージサービス

＋

Hadoop/Hiveを用いた
データ解析機能(オプション)

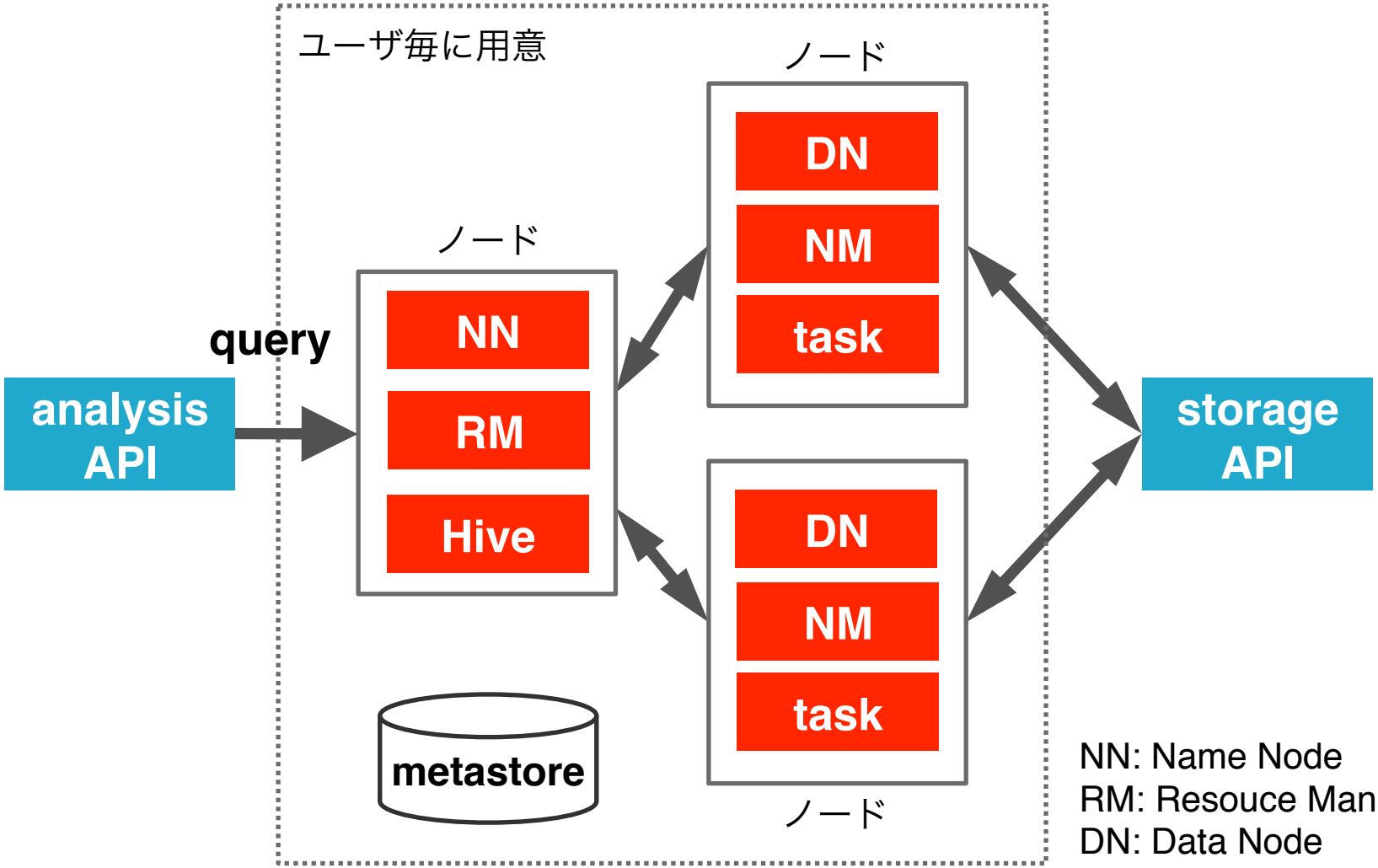
サービス全体図



計算ノード

- Hadoop + Hiveを使用
- マルチテナント

計算ノード(図)



NN: Name Node
RM: Resouce Manager
DN: Data Node
NM: Node Manager

計算ノード部分 要件

- ユーザ毎に実行環境が隔離されていること
- 公平なリソースの共有
- ノードはユーザにより増減可能なこと

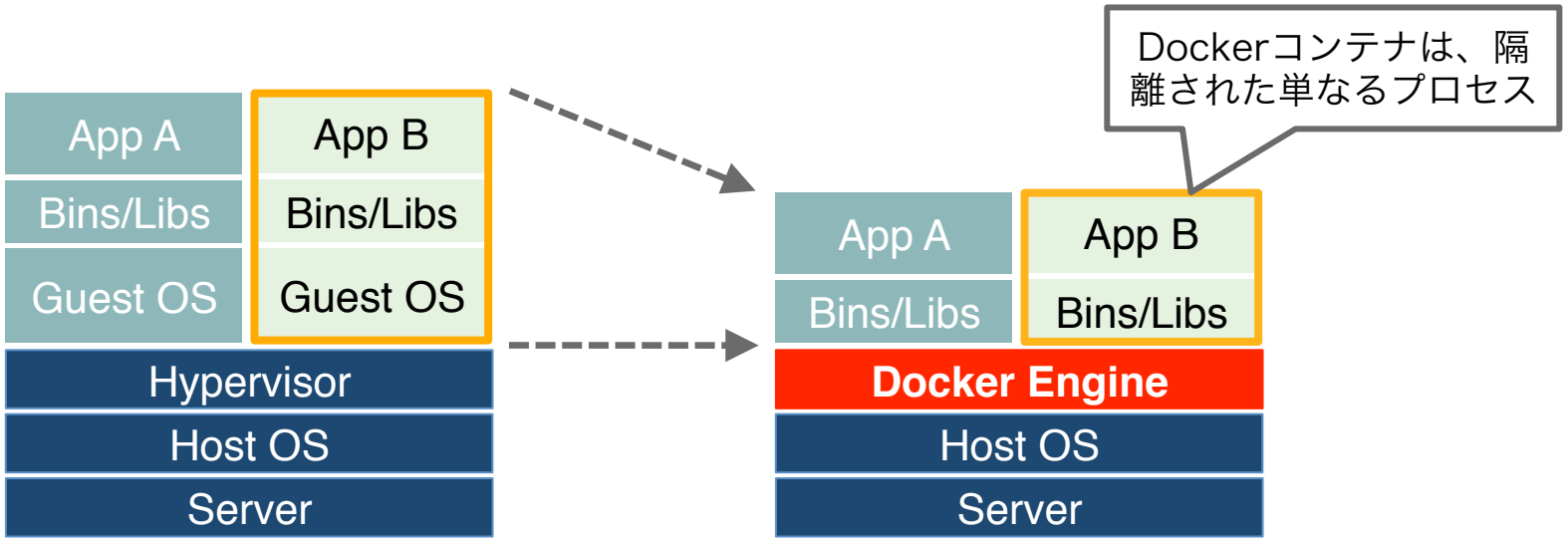
サービス検討時の候補

- Hypervisor型仮想マシンによる隔離
  - 隔離という面では一番確実
  - 起動が遅い、オーバーヘッドがある
- コンテナによる隔離
- アプリケーションレベルでの隔離
  - Hiveだけ(UDF禁止)等、やれることを限定すれば可能

Dockerとは?

- コンテナ上で動くアプリケーションの実行環境を構築・管理
- オープンソース
- コンテナとは?
  - 複数のLinux標準機能の組み合わせで実現する隔離環境
    - Namespaces, Cgroups, Capabilities など

仮想マシン vs コンテナ



Hypervisor型仮想マシン

ゲスト毎に任意のOS使用可
高い隔離性

Dockerコンテナ

オーバヘッドが少ない
起動が速い

こちらを採用

実際の本サービスの制限

- 現時点では、ユーザが任意のDockerイメージを動かすことは出来ない
  - 任意のMapReduceプログラムを動かすこともできない
  - 使えるのはHiveQLのみ
- Dockerの利用は将来への布石

Dockerを商用サービスの基盤に使うにあたって

Dockerが向いていること

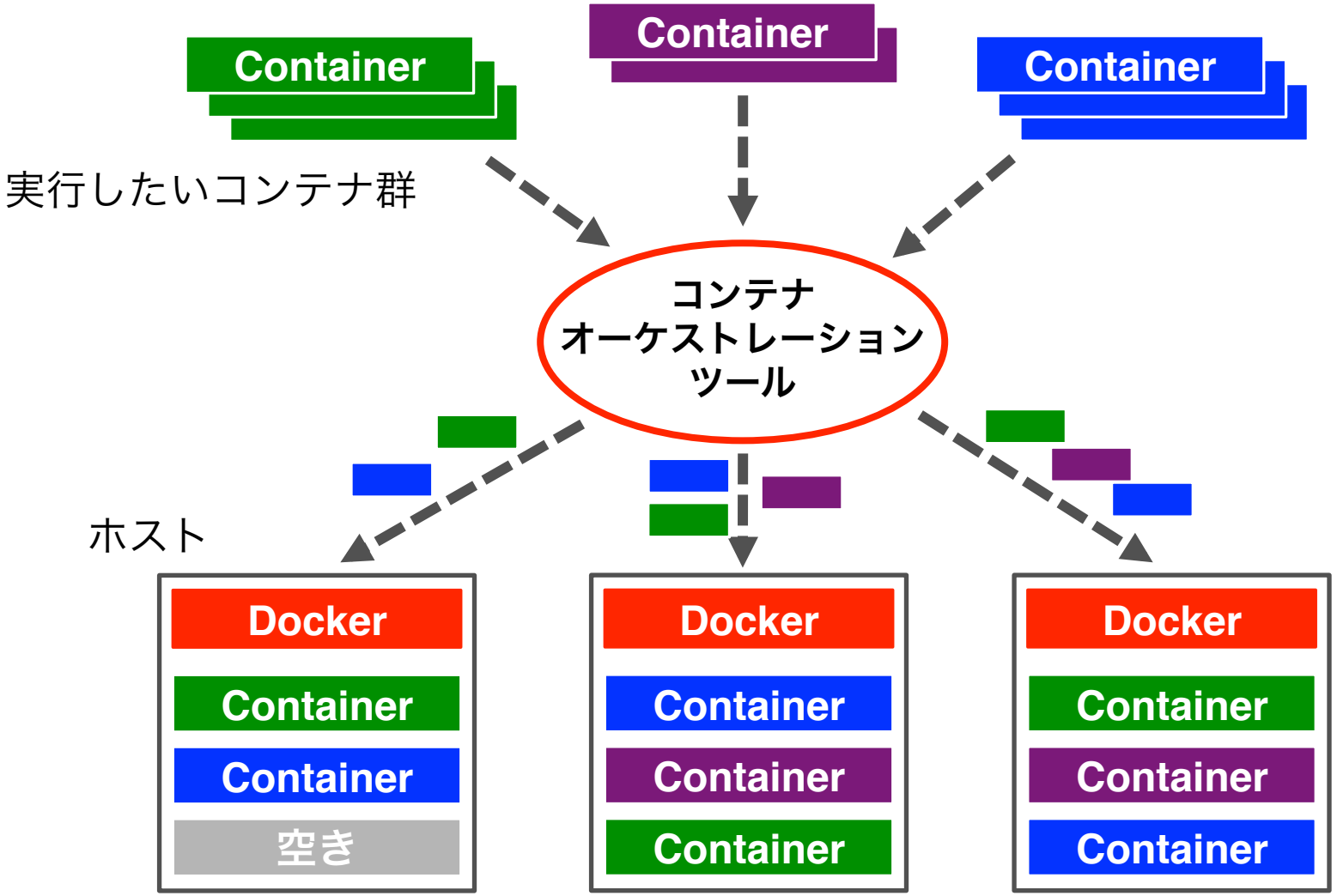
- 単体のホスト上での開発・ビルド・テスト環境
- 環境のパッケージングと配布

Dockerの課題

- 複数ホスト上の多数のコンテナの管理(オーケストレーション)
- 複数ホストをまたいだネットワーク
- コンテナのログの管理
- コンテナの監視、モニタリング

多数のコンテナの管理

オーケストレーション



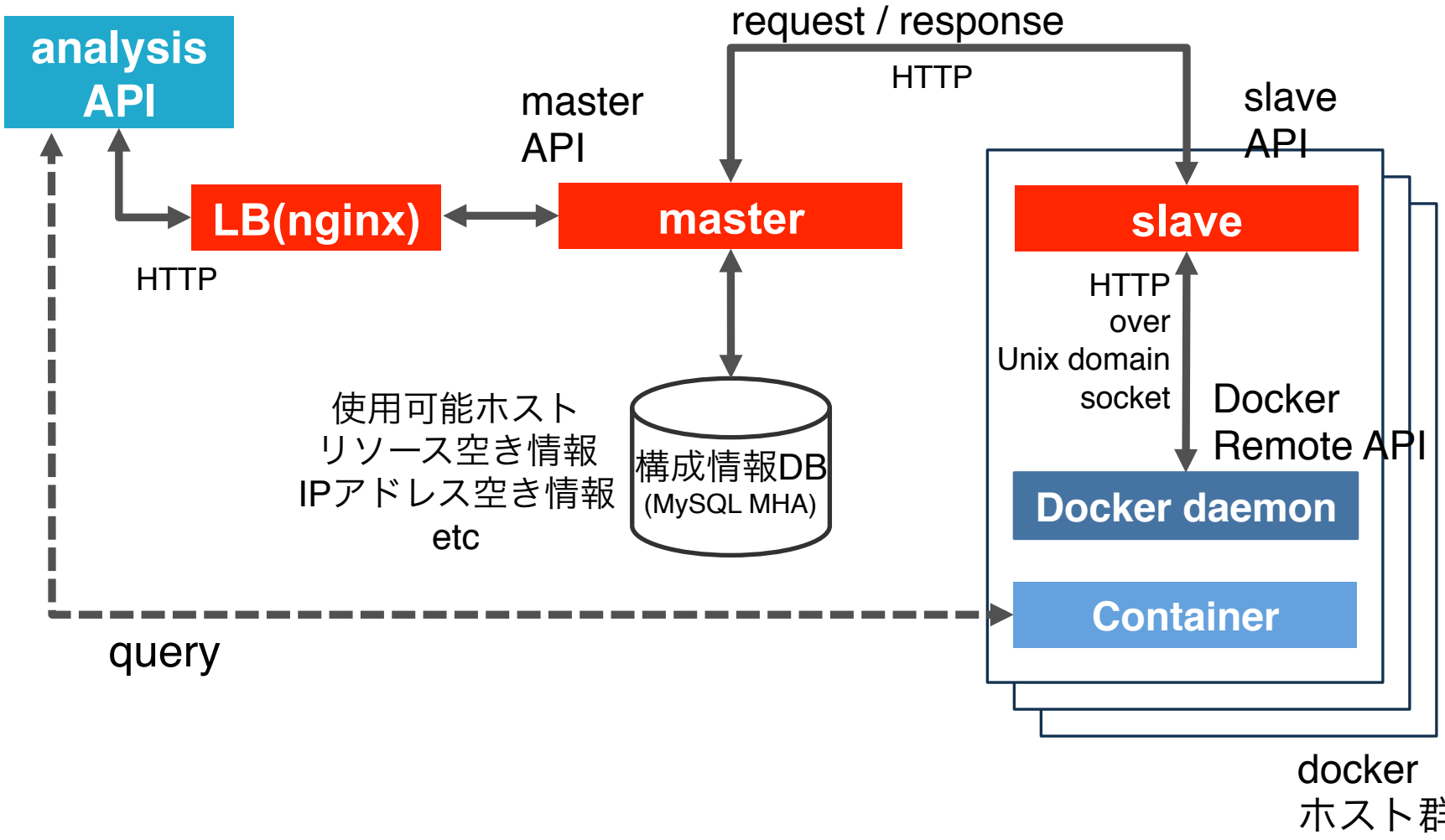
オーケストレーションツール

- 例
  - Docker Swarm
  - Kubernetes
  - Apache Mesos
  - Fleet
  - Nomad
  - ...

doma(docker manager)とは?

- 独自開発
- 多数のDockerコンテナを管理する
  - 複数ホストをリソースプールとする
  - analysis APIからの要求によりプールから必要数のコンテナを自動確保し起動

doma 構成図

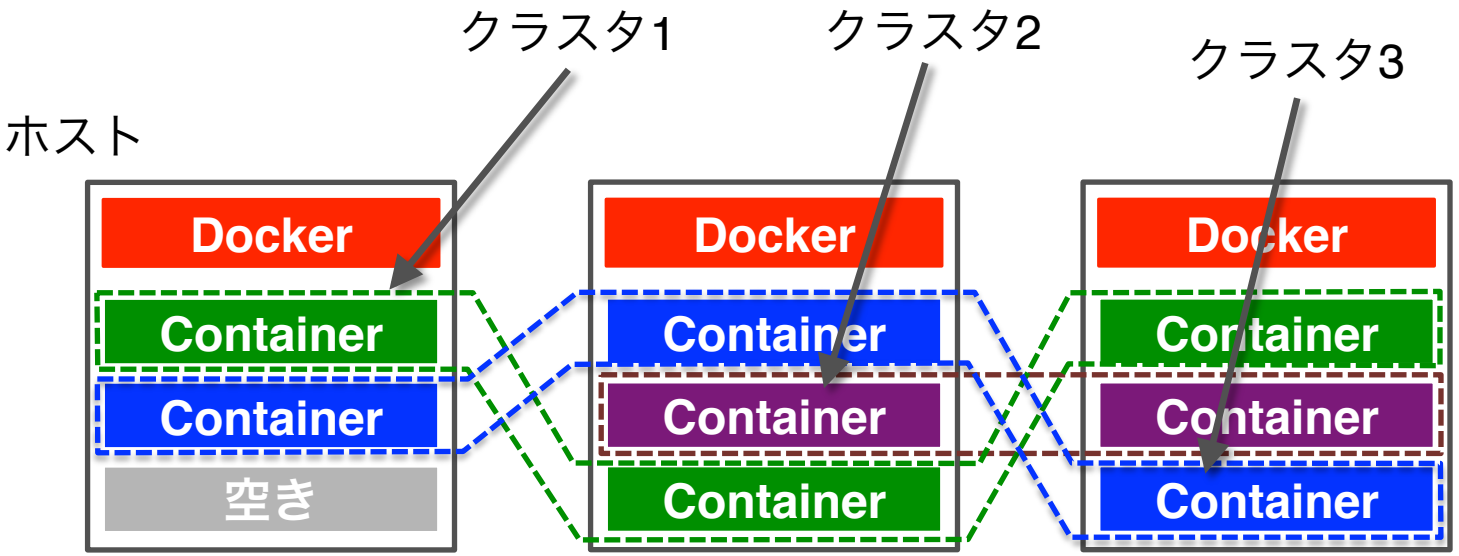


master

- 多数のコンテナをクラスタ単位で管理
  - クラスタ ＝ コンテナの集合
- クラスタ操作
  - 予約、アップデート、起動、停止、再起動、解放(削除)
- Ruby + Rack + EventMachine

クラスタ

- 互いに通信できるコンテナの集合
  - 1つのクラスタは1つのユーザに属する
  - 別のクラスタとは通信できない



masterが管理するもの

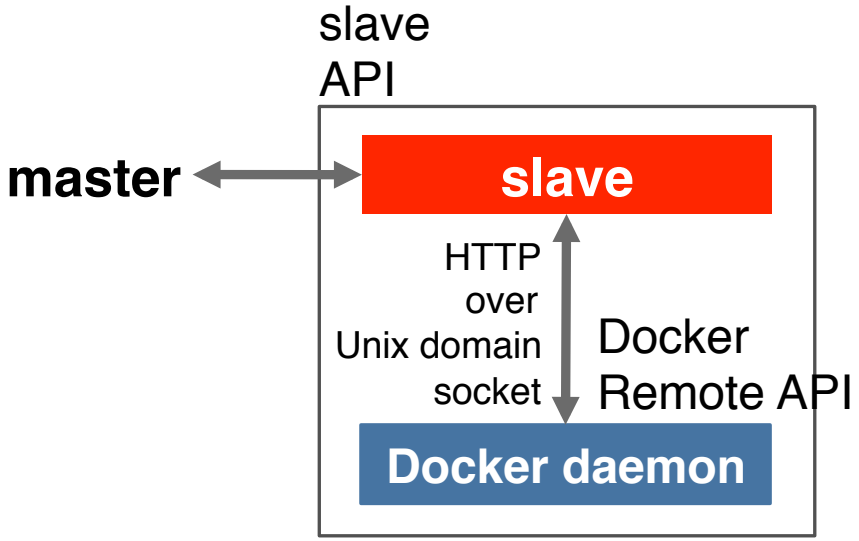
- ホスト
  - dockerが動く物理ホスト
  - CPU、メモリ割当て状況
- IPアドレス
  - コンテナは1個ずつ(プライベート)IPアドレスを持つ

コンテナの種類

- グレード
  - 性能を決める(CPUコア数、メモリ量)
- タイプ
  - dockerイメージ名に対応
  - アプリケーション毎に複数用意されている

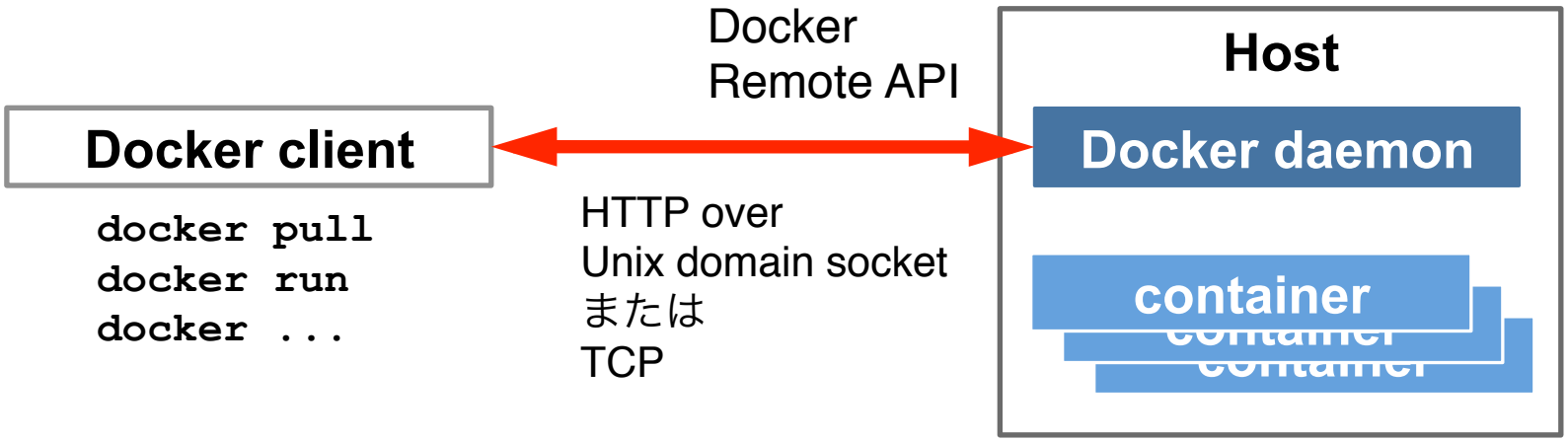
slave

- Docker daemonのwrapper
  - masterからは、ほぼDocker daemonに見える
  - いくつかAPIを拡張
- Goで記述



Docker Remote API

- HTTP、REST、JSONベース
- dockerコマンドはすべてこのAPIを経由している



slave追加機能

- コンテナ起動関連
  - サイズ制限されたディスク領域提供
  - ネットワーク設定
  - iptablesチェイン設定
- コンテナのメトリクス取得
- コンテナ内にファイルを送り込む
- コンテナ非同期削除

コンテナのリソース制限

リソース制限概略

- 主にcgroupを使う
- CPU
  - cpuset
- メモリ
  - memory
- ディスク使用量
  - cgroupではできない

CPU制限

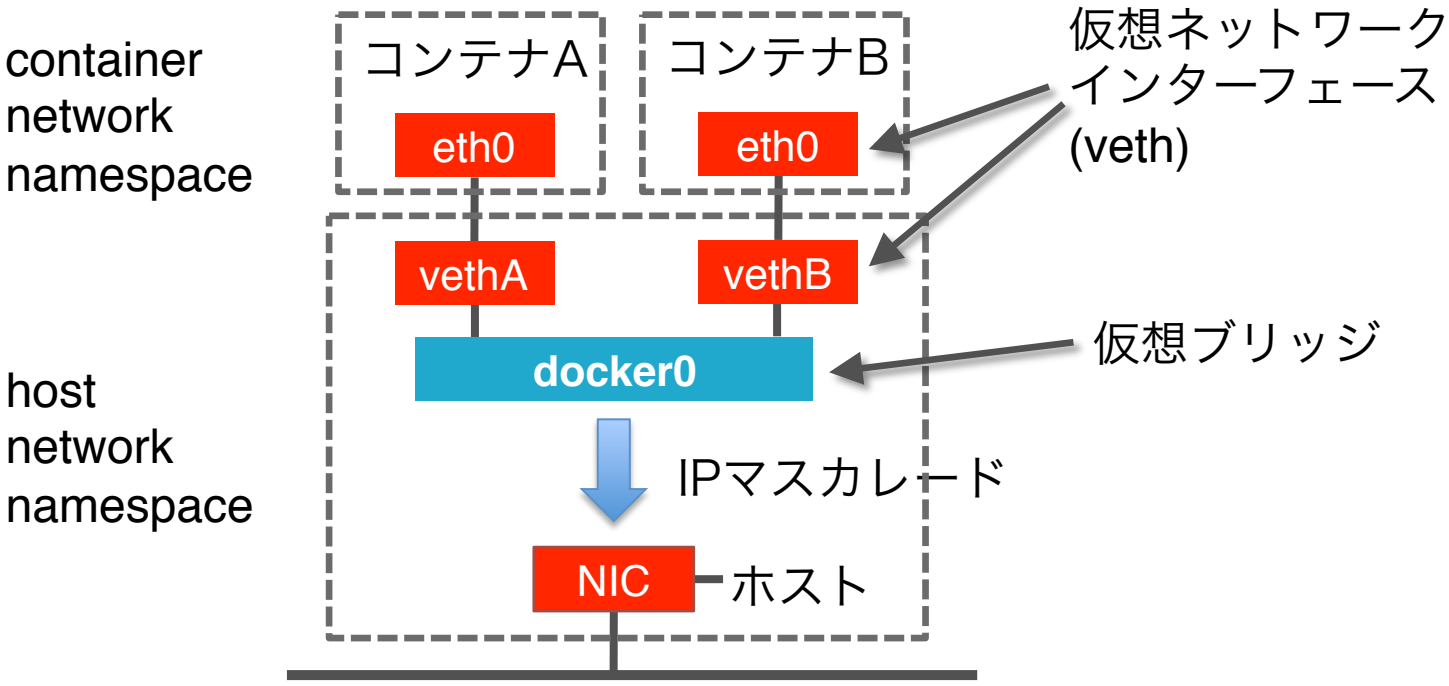
- (Docker Remote API経由で)cgroupのcpuset.cpusを使う
  - コンテナのプロセスが実行されるCPUコアの番号を指定
  - 例: "CpusetCpus":"0-2,7"
- 各コンテナおよびシステムプロセスそれぞれが使うCPUコアを分離

ディスク使用量制限

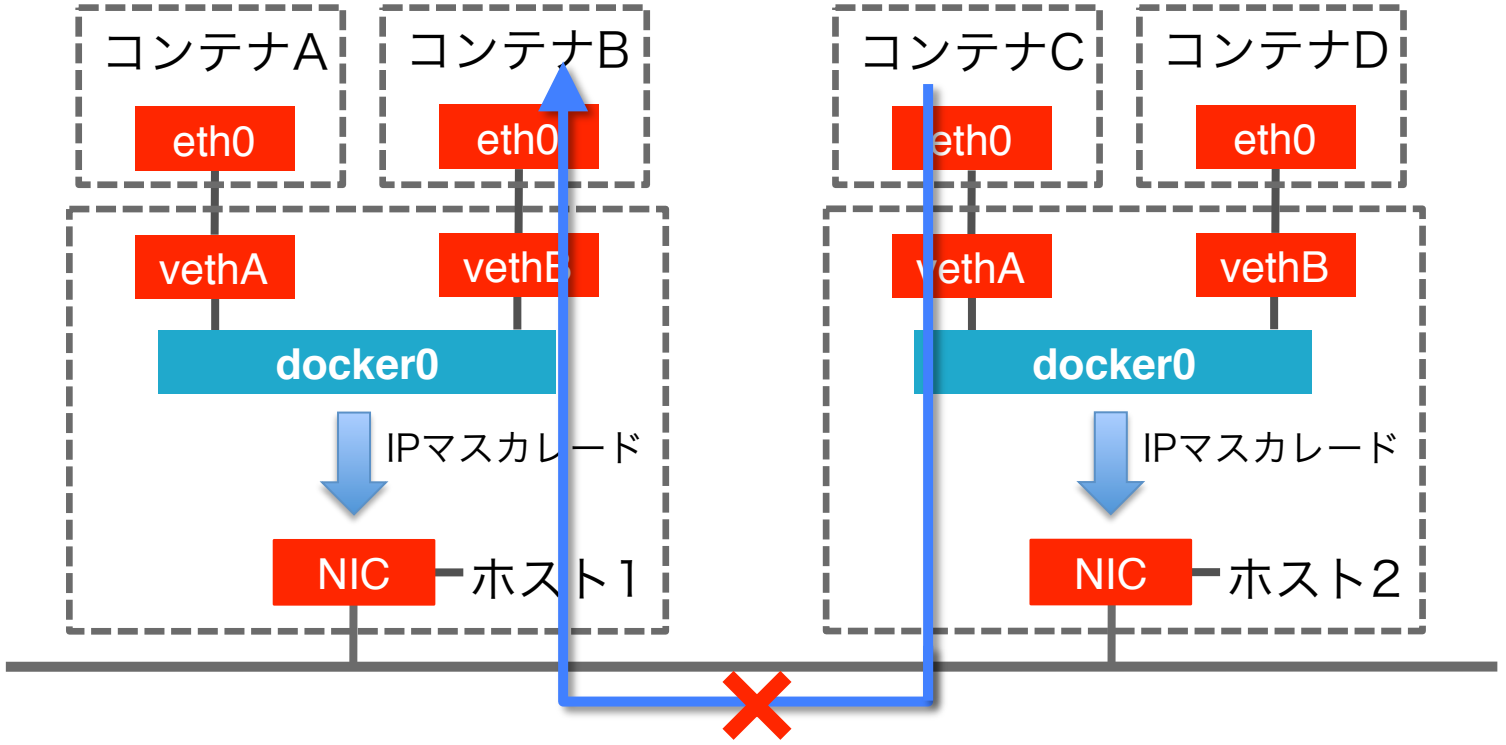
- 特定サイズのloopbackデバイス用ファイルを作って、それをmount
  - あらかじめmkfs済sparseイメージ入りtarファイルを用意
  - コンテナ起動時にそれを展開してmount(0.1〜0.2秒くらい)
  - コンテナ解放時はファイルを1個削除するだけ

コンテナ間のネットワーク

通常のDockerのネットワーク



通常のDockerのネットワーク

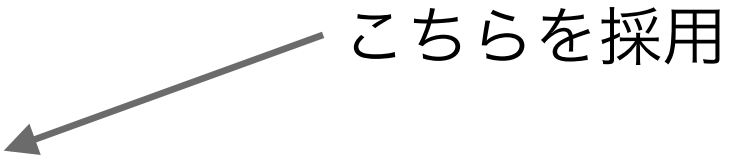


ホストをまたいだコンテナ間通信は困難

(ポートが固定されていればEXPOSEでできるが
Hadoopと相性が悪い)

解決策

- 例
  - pipework
  - weave
  - flannel
  - libnetwork overlay driver
  - ...
- または独自に頑張る

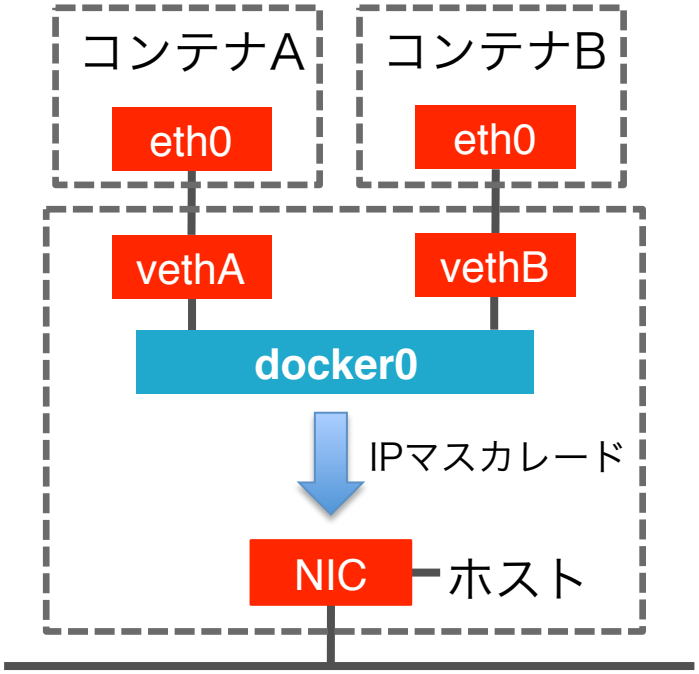


本サービスのネットワーク

- dockerのネットワーク設定は使用せず
  - ("NetworkDisabled": true)
- 代わりにslaveがコンテナ起動時にネットワーク設定をする

本サービスのネットワーク

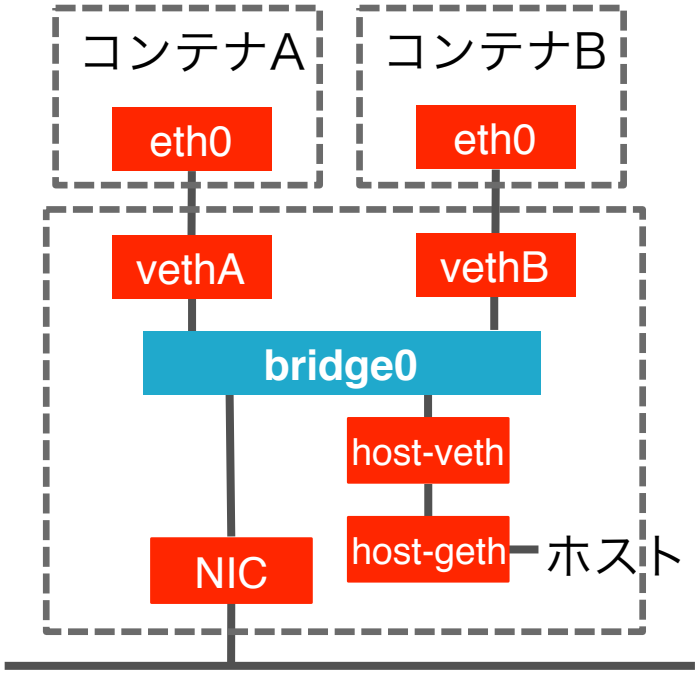
通常のdockerの
ネットワーク



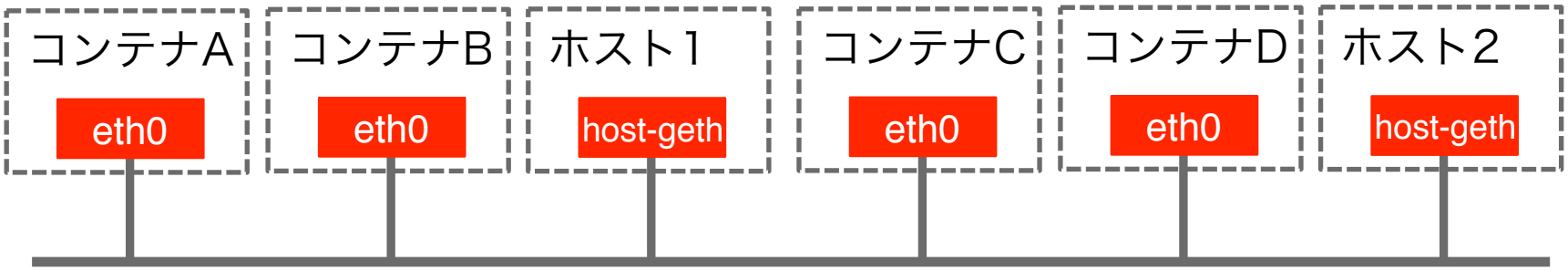
container
network
namespace

host
network
namespace

本サービスの
ネットワーク



論理的にはこんな感じ



- コンテナはホストと同じネットワークに直接つながる
- コンテナのIPアドレスはmasterが決定しslaveが設定する

ネットワークの隔離

- クラスタ間はiptablesで隔離
  - コンテナはCAP_NET_RAWを禁止

その他

Dockerイメージ

- Docker HUBのイメージは使用せず
  - 同じrepository名:tag名でも内容が変わっていることがある
  - (docker 1.6～はContent Addressable Image Identifiersにより一意に指定可)
- 独自にOSイメージ作成
  - febootstrap で CentOS 6ベースで作成
- そのOSイメージからJDK, Hive, Hadoop入りなどをDockerfileで生成

ログの管理

- コンテナ内の各種ログは、コンテナ内の特定ディレクトリに一時保存
- ホスト上(コンテナ外)のfluentdで外に飛ばす
- 本サービスのストレージ部分(管理用)に保存

(今ならDockerのlog driverを使えば良いと思う)

監視

- ホストが障害を起こしたらdomaはコンテナ割り当て対象から除外
  - 障害コンテナを別ホストに自動移動したり再起動はしない
- コンテナのアプリケーションレイヤの監視はanalysis APIが行っている

コンテナのモニタリング

- slaveに、コンテナ単位のメトリクスを返す機能追加
- cgroupの統計情報と、コンテナ内の /proc/net/dev 等からメトリクスを得る

(今なら cAdvisor を使えば良いと思う)

コンテナのモニタリング

↓CPU Accounting

↓Network traffic

Memory→

ホストOSについて

- CentOS 6を使用中
  - だがDocker 1.8以降、CentOS 6は対象外
  - Red Hat は、DockerをRHEL 6で動かすことを推奨していない
- コンテナ用OS
  - CoreOS, Project Atomic, Snappy, RancherOS

まとめ

- Hadoop/Hiveをマルチテナントで動かすためにDockerを採用
- Dockerは多数のコンテナを扱うには課題がある
  - 解決のためいくつかのツール類を開発
  - 今ならもっと楽な別のやり方があるはず

別のやり方

項目	本サービス	別のやり方(例)
コンテナオーケストレーション	独自 doma(docker manager)	Kubernetes, Swarm, Mesosなど
ネットワーク	独自	libnetwork, flannel, weave など
ロギング	外付けfluentd	docker log driver
モニタリング	独自	cAdvisor
OS	CentOS 6	CoreOS, Atomic Hostなど