

Docker を使ってみた！

LT風 - 2014/09/18

質問は隨時歓迎です。ちょっとでも
気になつたら即質問で

あ、マサカリ投げる（指摘、ツッコ
ミする）場合は私がへこまない程度
にお願いします

資料は後ほど展開するので、メモは
頑張らないで大丈夫です

今日のゴール

Docker の概要を理解して、興味を持つもらう

目次

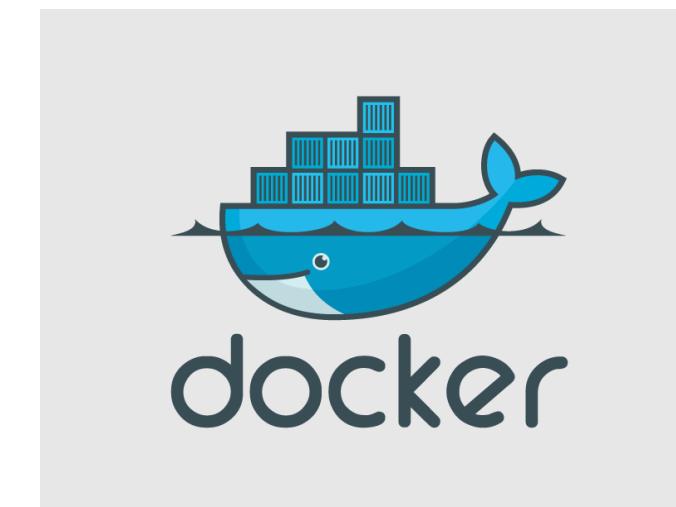
1. Docker ってなあに？
2. Docker の仕組み
3. Docker の利用方法
4. まとめ

目次

1. Docker ってなあに? ← これ
2. Docker の仕組み
3. Docker の利用方法
4. まとめ

Docker ってなあに？

Docker は Web アプリケーション等を仮想化するための技術の 1 つです。Linux KVM や VirtualBox といった「ハイパー バイザ型の仮想化」に対して、「コンテナ型の仮想化」とも呼ばれています。



('・∀・)ノハイ

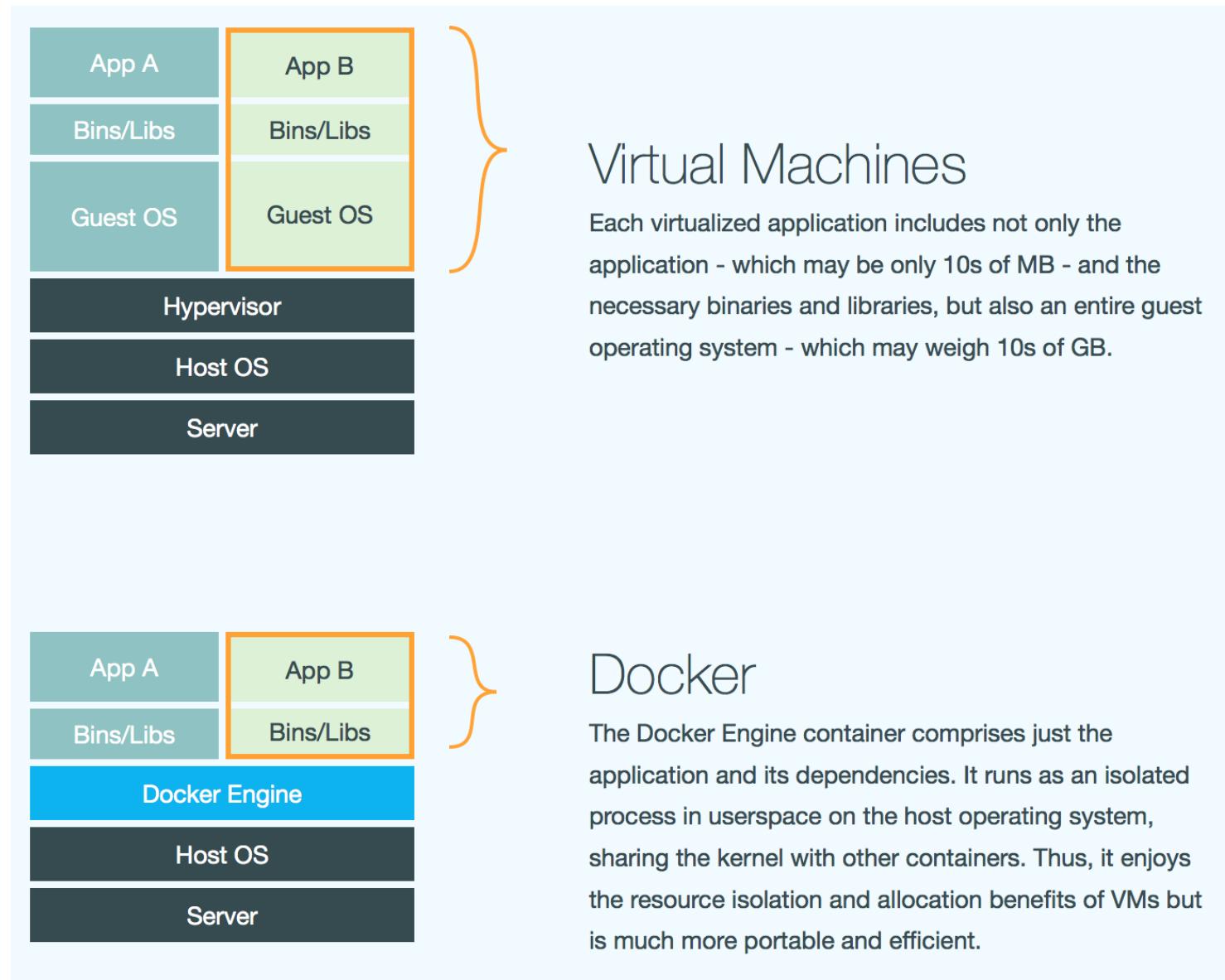


さっそく質問いいでですか

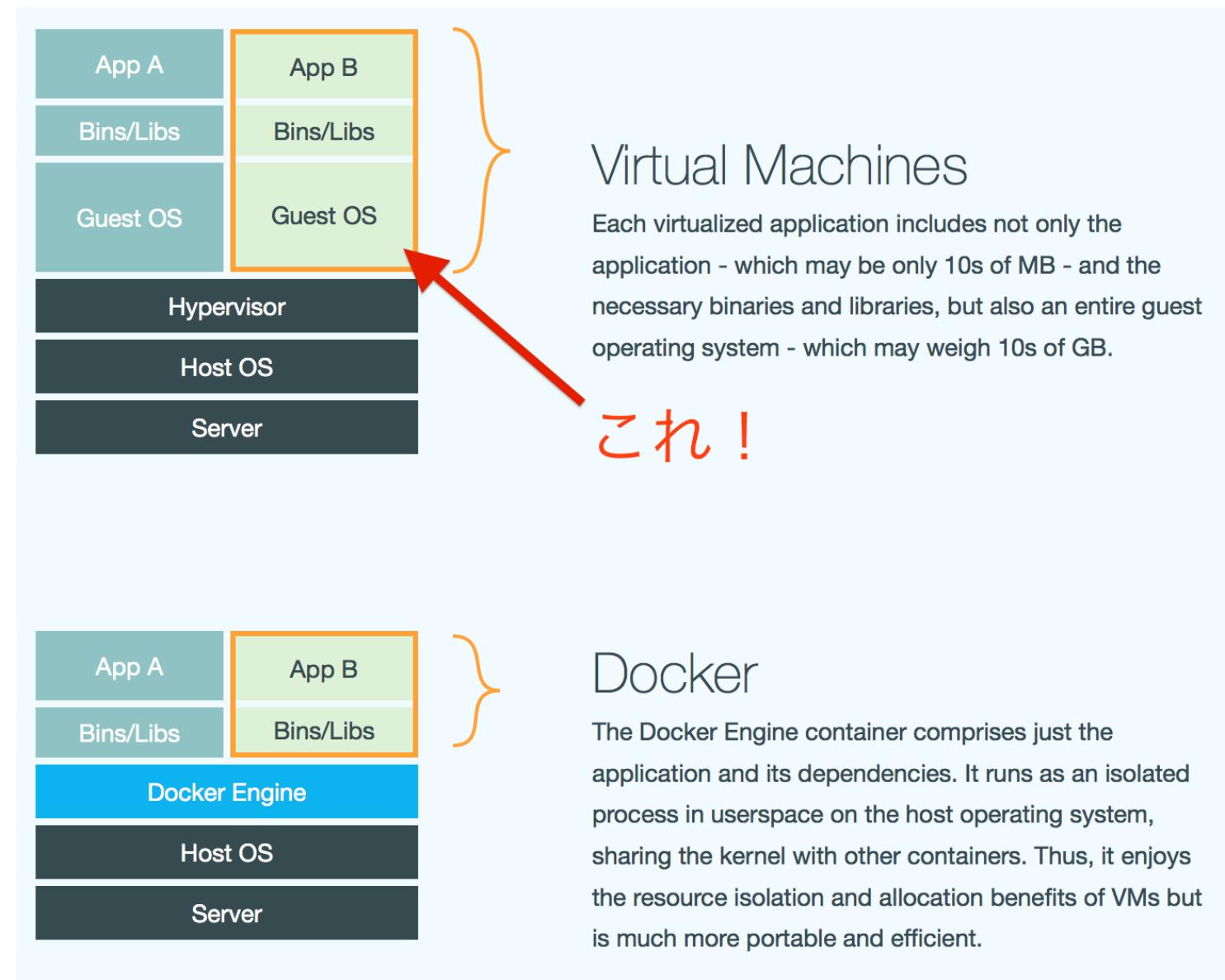
Q. Docker とハイパーバイザ型って何
が違うの？

A. ハイパーバイザの代わりにDocker Engineを使用します

図で見るとこんな感じ



ゲストOSのエミュレートは不要！



(`・△・。)ノハイ

。。。

エミュレートが不要だと
何がうれしいんですか？

うれしいこと - その1

- I. ハイパーバイザ型の場合はゲストOSをエミュレートして、その上でプロセスが動いているのに対し、**Docker Engine**は各プロセスをグループで隔離しているだけで、プロセス自体は直接ホストOS上で動かしているためオーバーヘッドが小さい。ようするに少ないリソースで仮想化が実現できる（超大事）。

うれしいこと - その2

2. オーバーヘッドが少ない分、ハイパーバイザ型と比べて起動が比較にならないほど速い！処理も速い！もう体感では物理サーバで動かしているのと変わらないくらい速い！

docker

('・∀・)ノハイ

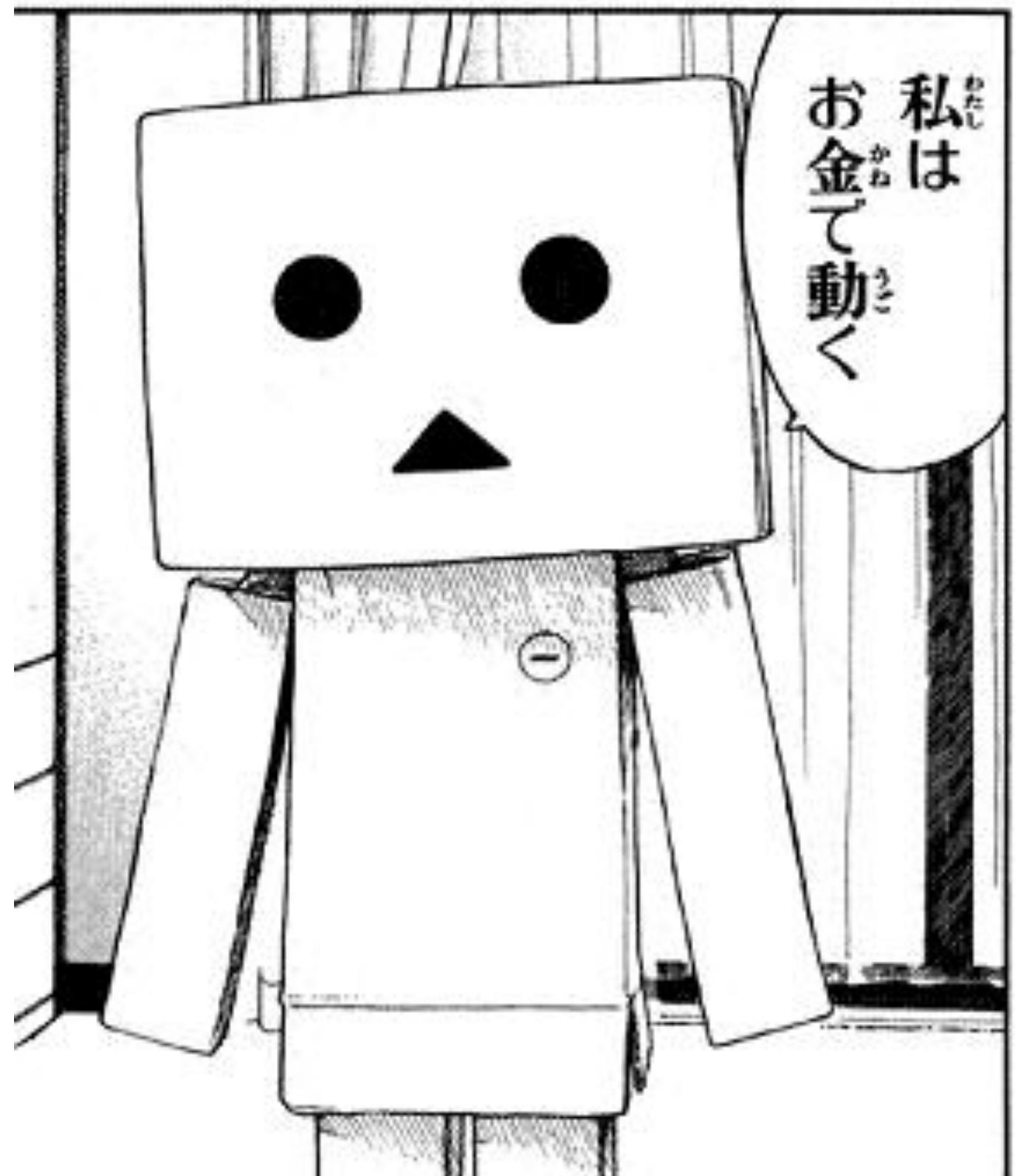


その1.について質問です

Q. 少ないリソースで仮想化実現できるのがそんなにうれしいんですか？

A. そんなにうれしいんで
す！

単純にサーバに負荷がかからないためということだけでなく、クラウド視点からすると「負荷増=コスト増」となるため、負荷はお金と直結しています。負荷が減ってお金減らせたらうれしいじゃん！



('・△・)ノハイ



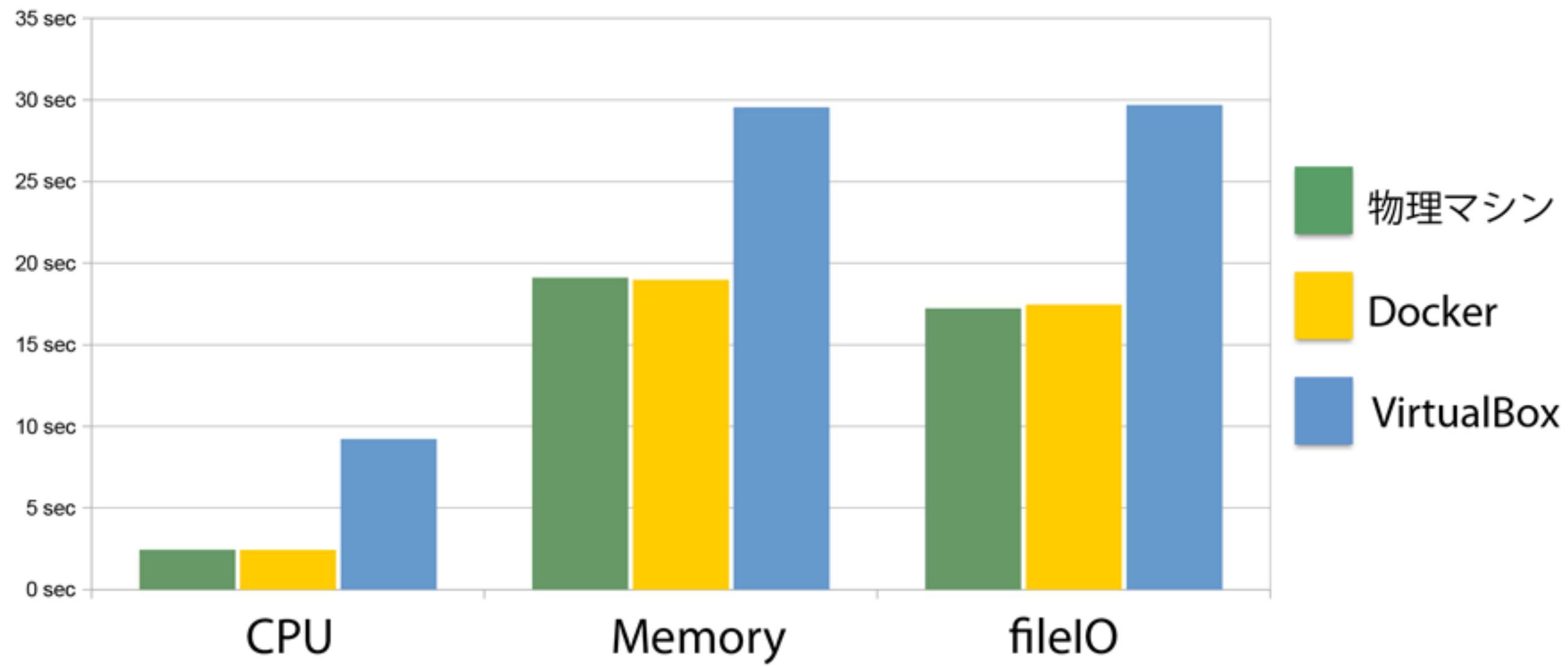
その2.について質問です

Q. 体感で速いとかいわれても分からぬいよ。実際どれくらい速いんですか？見える化してよ

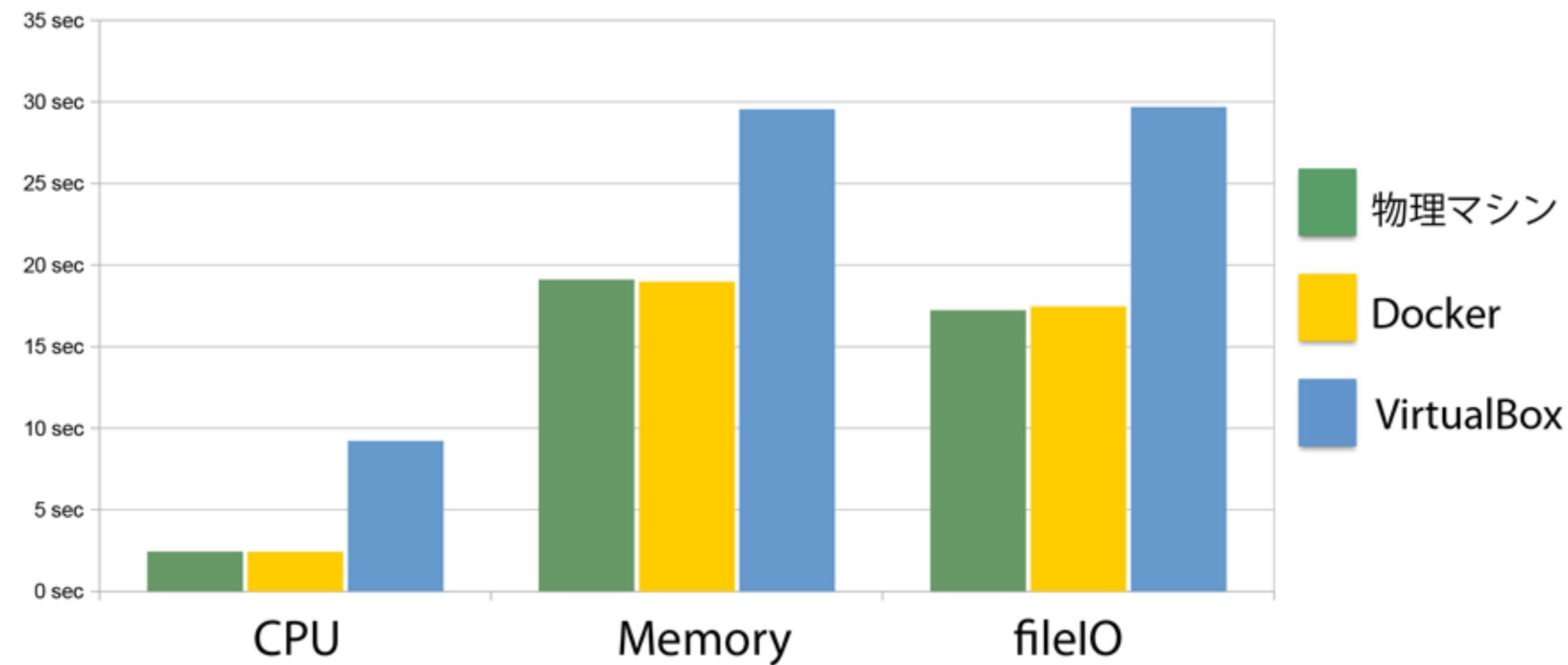
A. 速さを比較している表があった
ので貼付けてみます

..

参考：軽くて使いやすい仮想化技術「Docker」の仕組
みとエンタープライズ開発における4つの活用事例
(<http://codezine.jp/article/detail/7894>)



数値で比較しても物理サーバと違いが見られない！ようするに、



ハイパーバイザ型は

速さが足りない





ここで、Docker以外にもコンテナを実装しているものがあるので幾つかご紹介

→ LXC

Linuxのコンテナ化ソフトウェア。Docker 0.8まではLXCを利用してコンテナ化を行っていたが、Docker 0.9よりlibcontainerをベースとしたドライバがビルドインされ、このドライバによってコンテナ化を行うようになった。このため0.9以降ではLXCは必須ではなくなった（0.9以降もLXCドライバを利用してコンテナ化することは可能）。

→ OpenVZ

RHEL(Red Hat Enterprise Linux) 用のコンテナ実装。

Kernelに手を入れているため、公式には**RHEL**のみのサポートとなっている。また、ライブマイグレーションに対応している（この機能は **Docker** には存在していないのでライブマイグレーションを使いたい場合は**OpenVZ**を利用）

→ Warden

元々**VMware**が**Cloud Foundry**向けに開発した**Linux**向けコンテナ技術。現在この **Warden** を活用し、**Docker**の **libcontainer** プロジェクトの強化を行っている。

ちなみに

すでにGoogleは全部のソフトウェアをコンテナに乗せており、毎週20億個ものコンテナを起動しているらしいです。

実績もあるので、コンテナ化は今後
もっと使われそうなイメージです。

また、コンテナ化に関しては今
Docker が勢いがあるため、とりあえ
ずコンテナ化したい場合は *Docker*
を選ぶのが良いかと思います。



ということで、Docker の
仕組み

docker

目次

1. Docker ってなあに？
2. Docker の仕組み ← これ
3. Docker の利用方法
4. まとめ

Docker は既存技術を組み
合わせているだけです

主な技術

- **Go言語**：Docker は Go 言語で記述されています
- **User Namespace**：名前空間によるプロセス、ネットワークの独立化（この技術でコンテナ化を実現）
- **cgroups**：リソース(CPU、メモリ、ディスクI/Oなど)の制御
- **AUFS**：データを差分で管理するファイルシステム
(Docker Image を差分で管理するために使用)

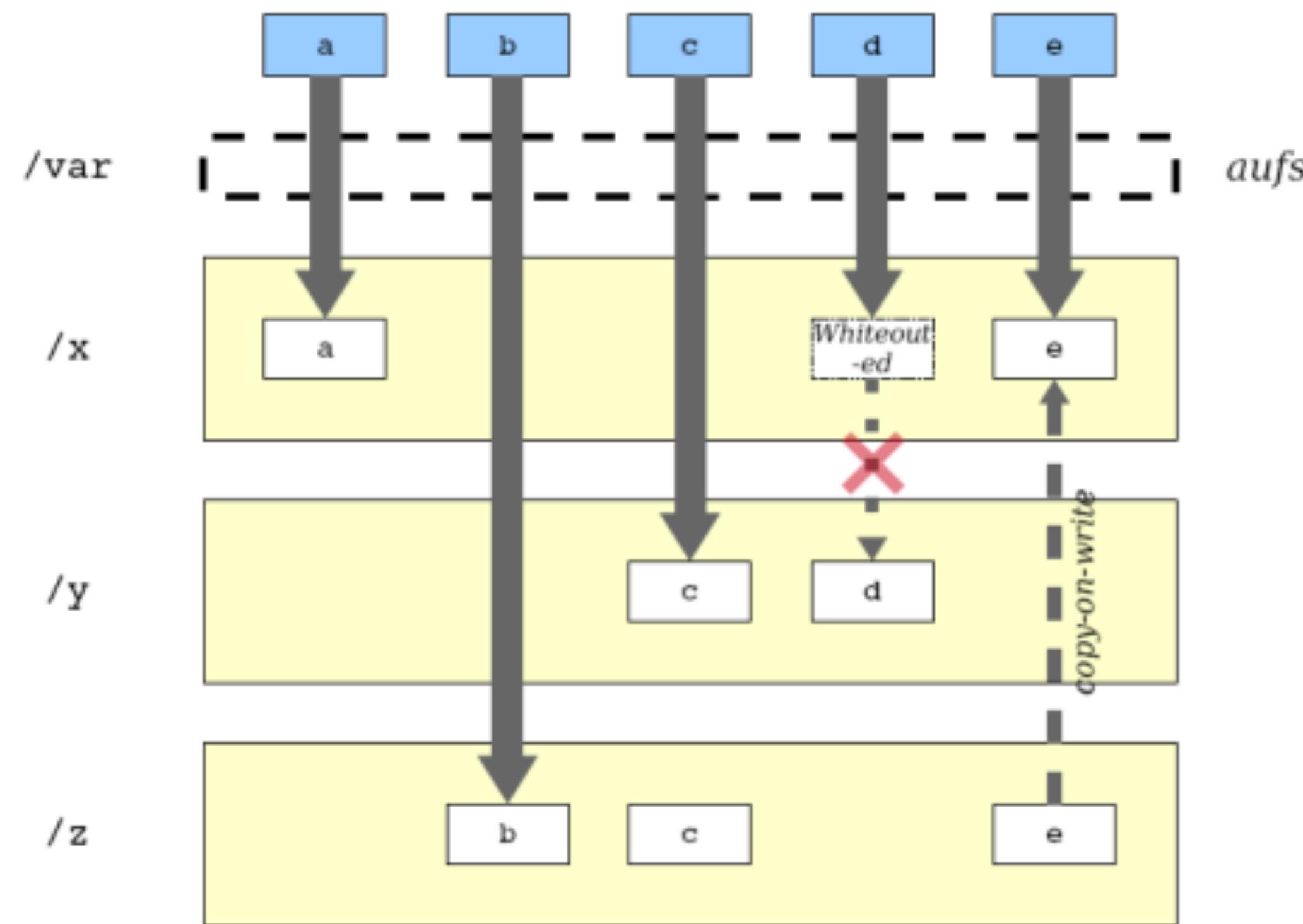
補足_1 User Namespace

→ **User Namespace** は **Kernel 3.8** 以降で実装された比較的新しい技術です（Ubuntuでいうと Version 12.04 くらい）。このため **Docker** を使用するためにはホストOSの **Kernel** が 3.8 以降である必要があります。

補足_2 AUFS

- **AUFS** は聞き慣れないファイルシステムかと思いますが、
CD Boot で **Linux**を起動した際に(**KNOPPIX**とか)、あた
かもファイルを書き換えてるかのように動かさせていたの
は、この **AUFS** を利用していたためです。
- ⇒ 具体的にいうと～

→ システム関連のファイルを書き換えようとしても、光学ディスクに書き込まれているため、物理的にファイルの内容を書き換えることができません。そこで、変更があった内容についてはメモリー上に保存しておき、差分として管理しています。図で見るとこんな感じ



また、**Docker** がコンテナ化するときは、コピーオンライト（原本をそのまま参照させておき、変更した際に初めて複製する）を行っているため、同一のイメージを複数仮想化させた際に無駄なリソースをとらないで済みます

具体的うれしいこととしては

- **IGB**のイメージを 10台立ち上げたとしても必要とするストレージは**IGB**で良い（ハイパーバイザ型の場合は**I0GB**必要となる）ためストレージを節約できる
- 同様の理由でメモリも無駄なリソースを食わないため節約できる
- メモリ上にキャッシュされているため高速でデプロイできる

仕組みは以上

次はお待ちかね？の利用
方法（デモもあるよ）

目次

1. Docker ってなあに？
2. Docker の仕組み
3. Docker の利用方法 ← これ
4. まとめ

Docker の利用方法

1. Docker をインストール(apt-get や yum)
2. Dockerfile を作成
3. Docker build コマンドで Docker Image を作成
4. Docker run コマンドで Docker Image をコンテナとして起動
 - ※ Docker stop でコンテナを停止。start でコンテナを再度起動できます

大同

ただ、Docker をインストールする際
は注意点が



繰り返しますが kernel は 3.8 以上で
ある必要があります。よく使われて
そうな CentOS を調べてみると・・・

あ、

	5.10 (最新)	i386, x86-64	2.6.18-371	2013年10月19日	2013年 9月30日
6	6.0	i386, x86-64	2.6.32-71	2011年 7月 9日	2010年11月10日
	6.1	i386, x86-64	2.6.32-131	2011年12月 9日	2011年 5月19日
	6.2	i386, x86-64	2.6.32-220	2011年12月20日	2011年12月 6日
	6.3	i386, x86-64	2.6.32-279	2012年 7月 9日	2012年 6月21日
	6.4	i386, x86-64	2.6.32-358	2013年 3月 9日	2013年 2月21日
	6.5 (最新)	i386, x86-64	2.6.32-431	2013年12月 1日	2013年11月21日
7	7.0-1406 (最新)	x86-64	3.10.0-123	2014年 7月 7日	2014年 6月10日

6系全滅だ(3.x ですらない・・)

	5.10 (最新)	i386, x86-64	2.6.18-371	2013年10月19日	2013年 9月30日
6	6.0	i386, x86-64	2.6.32-71	2011年 7月 9日	2010年11月10日
	6.1	i386, x86-64	2.6.32-131	2011年12月 9日	2011年 5月19日
	6.2	i386, x86-64	2.6.32-220	2011年12月20日	2011年12月 6日
	6.3	i386, x86-64	2.6.32-279	2012年 7月 9日	2012年 6月21日
	6.4	i386, x86-64	2.6.32-358	2013年 3月 9日	2013年 2月21日
	6.5 (最新)	i386, x86-64	2.6.32-431	2013年12月 1日	2013年11月21日
7	7.0-1406 (最新)	x86-64	3.10.0-123	2014年 7月 7日	2014年 6月10日

一応 Kernel version を上げることで
使えますが、可能であれば7系使い
ましょう(サポート期間も6系より長
い 2020年11月30日 ⇒ 2024年6月
30日)。

また、Docker 開発チームはホスト OSとしては、Ubuntu を推奨しています。これは、Docker が Ubuntu 上で開発されているためです。このため、OSが選べるのであれば Ubuntu を使うのが良いように思います。

補足として、ローカル環境で Docker を動かしたい場合は `boot2docker` といった Docker 用の軽量なディストリビューションを利用することをおすすめします。VirtualBox 上で動かします（後ほどデモでます）。

Docker の利用方法にもどります(以下はさつきと同じです)

1. Docker をインストール(`apt-get` や `yum`)
2. Dockerfile を作成
3. Docker build コマンドで Docker Image を作成
4. Docker run コマンドで Docker Image をコンテナとして起動

※ Docker stopでコンテナを停止。startでコンテナを再度起動できます

さっくり説明していきます

1. Docker をインストール

ごめんなさい、ここ見てください

→ <https://docs.docker.com/installation/#installation>

各プラットフォームに対して丁寧に解説
がされています（ムービー付きのもの
も！）

一応例として Ubuntu 14.04 ではこんな感じです。

```
$ sudo apt-get update
$ sudo apt-get install docker.io
$ sudo ln -sf /usr/bin/docker.io /usr/local/bin/docker
$ sudo sed -i '$acomplete -F _docker docker' /etc/bash_completion.d/docker.io
```

2. Dockerfile を作成

Dockerfile とは

→ **Dockerfile** は、プログラムのビルドでよく利用される **make** ツールの **Makefile** ファイルと同様に、**Docker** コンテナーの構成内容をまとめて記述するシンプルなテキスト形式のファイルです。**Dockerfile** があればどの環境でも同様な環境を容易に構築することができます。

Dockerfile の書き方

→ 1行につき1つの操作を {命令} と {引数} でスペース区切りで記述します。「#」から始まる行はコメントとして処理されます。DockerImageを作成する「**docker build**」コマンドでは、「**Dockerfile**」ファイルの上から順番に処理が実行されます。

コメント

{命令} {引数}

Dockerfile 命令概要

- **FROM** : 元となるDockerイメージの指定
- **MAINTAINER** : 作成者情報
- **RUN** : コマンドの実行
- **ADD** : ファイル／ディレクトリの追加
- **CMD** : コンテナーの実行コマンド 1
- **ENTRYPOINT** : コンテナーの実行コマンド 2
- **WORKDIR** : 作業ディレクトリの指定

- **ENV** : 環境変数の指定
- **USER** : 実行ユーザーの指定
- **EXPOSE** : ポートのエクスポート
- **VOLUME** : ボリュームのマウント

Dockerfile のシンプルな例

1. **FROM** 命令でベースとなるイメージ (OS等) を指定
2. **RUN** 命令で任意のコマンドを実行 (**apt-get** 等) し、その結果を作成するイメージにコミットする
3. **CMD** 命令で実行するプログラムを指定

nginx を例に Dockerfileを作成

```
# まずは FROMでベースとなるOSを指定  
FROM ubuntu  
# RUN で nginx をインストール  
RUN apt-get update && apt-get install -y nginx  
# 再度 RUN で index.html を作成  
RUN echo "piyopiyo" > /usr/share/nginx/html/index.html  
# CMD でnginx をバックグラウンドで起動  
CMD /usr/sbin/nginx -g 'daemon off;' -c /etc/nginx/nginx.conf
```

ということで実際に動か
してみましょう(デモ)

Dockerfile ですが、github 等で公開されています

→ 先ほどの nginx であれば <https://github.com/dockerfile/nginx/blob/master/Dockerfile> より、もっと完成度の高いものを取得できます。

Docker Image も Docker Hub より取得することができます

→ <https://hub.docker.com/>

また、Docker Hub では Docker Image をリポジトリとして利用できます



ようするに、作成した Docker Image を共有することができます。

使い方も簡単

- 共有したい Docker Image があったら
 - 1. Docker login
 - 2. Docker push [Docker Image]
- 取得したい Docker Image があったら
 - 1. Docker pull [Docker Image]

ということで実際に動か
してみましょう(デモ)

デモ終わりましたが、追加で3点ほど説明させてください

1 点目

Dockerfile と Docker Image

Q. 結局どっちを使えばいいの？そのまま使える Docker Image だけでいいのでは？

A. 環境を作る側（例：インフラ担当者）は Dockerfile で環境を定義し、Docker Image を作成しておき。環境を利用する側（例：開発者）は単純に Docker Image を pull して利用するのがよいかと思います

インフラ担当者は `Dockerfile` を用意しておくことで、一部環境を変更する必要があった場合にも `Dockerfile` の一部を書き換えるだけで対応可能になります

2点目



docker commit

コンテナは実行ごとにイメージから作成されます。そのため、現在実行中の状態は引き継がれません。現在の状態をイメージに反映するには `docker commit` コマンドを使用する必要があります

逆を返すと `docker stop` すると変更
された内容はすべて消えてしまいま
す

ようするに、DB等、変更された内容を保存しておく必要がある場合は、何かしら対策しておく必要があります

対策例をあげると

- DB を Docker で動かす場合は、データ保存場所はコンテナ外にしましょう
- アプリログも同様に、AWS S3 等の外部ストレージに保存するような処理を入れておきましょう

3 点目

Immutable Infrastructure

聞いたことある or 知つ
てる人



('・△・)ノハイ

Immutable Infrastructure とは

不变なサーバー基盤のこと。具体的には、一度サーバーを構築したらその後はサーバーのソフトウェアに変更を加えないことを意味する。

通常、サーバーにはソフトウェア構成の変更がしばしば行われ、場合によってはそれがアプリケーションの安定稼働に大きな影響をもたらすことがある。また、アプリケーションがサーバーソフトウェアに変更を加え、サーバー環境が破壊されてしまうこともある。

しかしながら、**Immutable Infrastructure**の考え方では、サーバーのソフトウェア構成を一定に固定し、アプリケーションを**Docker**などの仮想環境上で稼働させることで、ホストOS自体への変更を行わないようにして安定的にアプリケーションを稼働させる仕組みとなっている。

アプリケーションを稼働させる仮想環境 자체はいつでも廃棄・生成可能な仕様とすることで、アプリケーションを環境による問題から解放して安定稼働させることが可能になる。

by wikipedia 先生

具体的になにがうれしいかというと

- ローカル環境で気軽にTry & Errorしながら構築できる
- 環境ごとの差分を意識しなくてよい
- クリーンな環境で作業ができる
- デプロイ 자체もあらかじめ確認できるため安全に行える

Docker は Immutable Infrastructure の概念にマッチしているので、その恩恵を得ることができます

ガシガシ使って行きましょう

最後に Docker のメリット / デメリット含めてまとめてみます

目次

1. Docker ってなあに？
2. Docker の仕組み
3. Docker の利用方法
4. まとめ ← これ

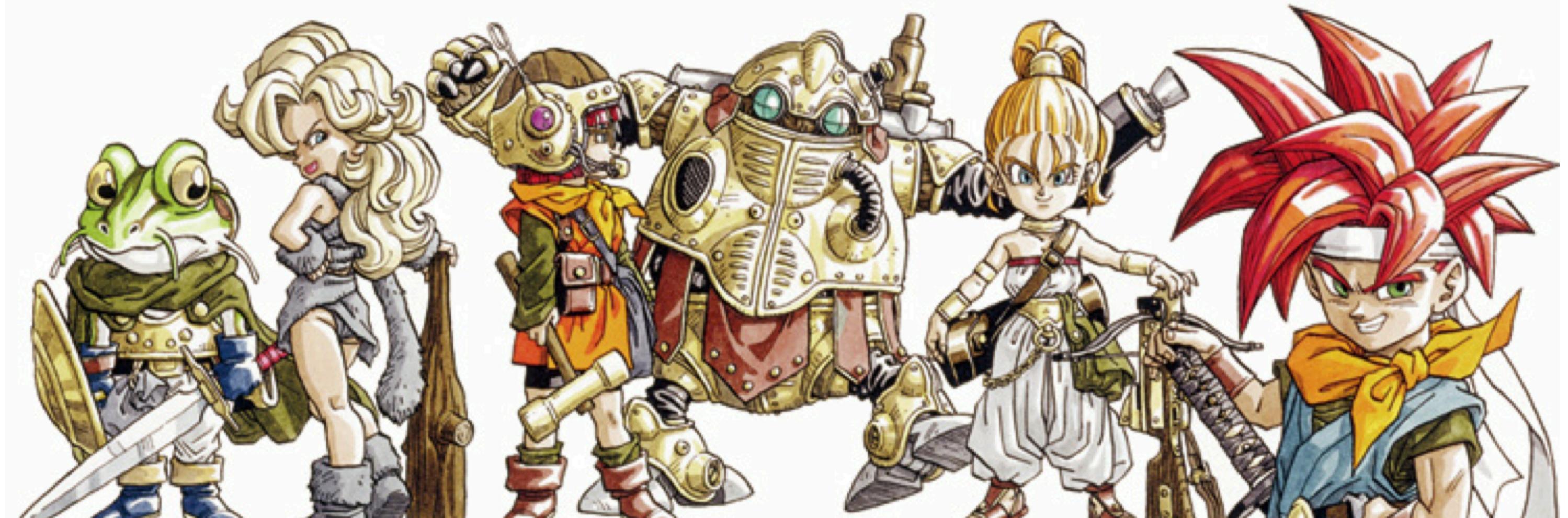
Docker のメリット

- 速い！
- 効率よくリソースが使える
- ポータビリティ性が高い
Dockerfile や *Docker Image* を利用することで、どのサーバでも同じ環境を構築することができる
- 有名どころの会社がどんどん Docker 関係で集まってきた
いる

Docker 関係で集まっている例

- 米VMwareは2014/8/25、カリフォルニア州サンフランシスコで開催の年次カンファレンス「VMworld 2014」において、米Docker、米Google、米Pivotalとの提携を発表
- 米Googleは2014/7/10、「Docker」コンテナの管理ツール「Kubernetes」（リンク先はGitHub）のプロジェクトに米Microsoft、米Red Hat、米IBM、米Docker、米Mesosphere、米CoreOS、米SaltStackが参加したと発表した。

なんかもうドリームプロジェクトな感じ



Docker のデメリット

- コンテナ内のデータが保持されないため、一手間必要な場合がある
- 複数のコンテナを管理するツールがまだ未成熟
本番で使用するにはまだ早い感じ
- **Linux** のプロセスを分けて仮想化しているため、**Linux** 以外のOSを仮想化させて動かす事はできません（もちろん **Windows** を仮想化することはできません）

まとめ



現状としては開発環境で使ったり、
今後くるであろう Docker の流れに
備えておけると良いように思います

おわり

ご清聴ありがとうございました。

参考文献

- Docker 入門 - Immutable Infrastructure を実現する -
(技術評論社)
- アプリ開発者もインフラ管理者も知っておきたいDockerの基礎知識 (<http://www.atmarkit.co.jp/ait/articles/1405/16/news032.html>)
- DockerをCentOS 7にインストールする方法 (<http://blog.yuryu.jp/2014/07/docker-centos7.html>)

- Dockerが利用しているAUFSとLXC
<http://shibayu36.hatenablog.com/entry/2013/12/30/173949>
- CentOS で aufs (another unionfs) を使う
<http://d.hatena.ne.jp/dayflower/20080714/1216010519>
- Docker基礎+docker0.9, 0.10概要
<http://www.slideshare.net/mainya/dockerdocker09-010>
- コンテナ型仮想化「Docker 0.9」リリース。LXCに依存しない独自のAPIを実装

→ Docker 0.9 リリースドキュメント日本語訳: Execution
driversとlibcontainer導入

<http://d.hatena.ne.jp/mainyaa/20140311/pl>

→ PaaS基盤「Cloud Foundry V2」内部で使われる
Buildpack、Wardenコンテナの仕組みとは？

http://www.publickey1.jp/blog/14/paascloudfoundryv2buildpackwarden_l.html

→ VMware、Docker、Google、Pivotalと協業し、企業向けコンテナ技術の導入を簡素化

<http://www.vmware.com/jp/company/news/releases/vmw-Docker-Google-Pivotal-082514>