WORDの編集長も生前退位します 号

目次

Docker Tutorials

文　編集部　linerlock

“遠くから見れば, 大抵のものは綺麗に見える”

— 村上春樹「1973 年のピンボール」より

1　はじめに

Docker を活用した開発環境やインフラの構築が盛んになってきている昨今ですが, その一方でまだ二の足を踏んでいる方も多くいるというのが私の肌感覚としてあります. 今回はこのようなギャップを少しでも埋めるお手伝いがしたいと思い, Docker の入門記事を書いてみることにしました. この記事は Jenkins と GitBucket を実際にデプロイしてみようという, ハンズオン的な内容になっていますので, 是非 Docker をインストールした環境を用意して, 実際に試して頂ければと思っています.

本記事の補足情報や, 記事中で用いる設定ファイルなどを次の URL で公開しています.
`http://appendices.word-ac.net/WORD41/docker-tutorial/`
誌面上では Docker 本体のインストール方法については扱いませんが, 補足資料として `docker-tutorial/doc/INSTALL.mkd` に簡単にまとめておきましたので, 参考にしてみてください.

2　Docker ことはじめ

インストールが上手くできているかの確認も兼ねて, まずは Docker における Hello, World をやってみましょう. Docker のインストールには意外と落し穴があるので, ここでしっかり確認しておくことをおすすめします. エラーが出た場合には `docker-tutorial/doc/FAQ.mkd` の情報が役に立つかもしれません.

```
$ docker pull hello-world:latest
$ docker run --rm hello-world
Hello from Docker!
This message shows that your installation appears to be working correctly.
```

このコマンドの意味についてはここでは解説しませんが, Docker 側が公式に用意している `hello-world` という Hello, World 専用の「コンテナ」を使って, Hello from Docker! を出力させています. 実際には Hello from Docker!というメッセージ以外にも初学者にとって有用な情報だと思われる情報が出力されていますので, 目を通しておいてください.

3 Jenkins コンテナの実行

Jenkins はオープンソースで開発されている, CI (継続的インテグレーション) ツールです. 既にお世話になっている方も多いと思います. 本章では Docker を使って Jenkins を立ち上げる方法をご紹介します.

```
$ docker pull jenkins:latest
$ docker run --name jenkins -d -p 8080:8080 jenkins
```

コマンドを実行したら, http://localhost:8080 にアクセスしてみてください. Jenkins のログイン画面が表示されていれば成功です.

このコマンドの意味を解説する前に少しだけ Docker に関する用語を準備しておきたいと思います. Docker における基本的な実行単位を「コンテナ」と呼びます. コンテナはあるアプリケーションと, そのアプリケーションを実行するのに必要なリソースが一通り用意された, ホストとは切り離された環境のことです. コンテナ自体は「イメージ」から作られます. 上記コマンドのうち, docker pull コマンドが, イメージの取得, docker run コマンドがコンテナの作成と実行に対応しています. それぞれ詳しく見ていきましょう.

3.1 docker pull コマンド

イメージは自分で作ることもできますが, 既に誰かが公開してくれているイメージを使えば, その手間を削減できます. docker pull コマンドはウェブ上で配布されているイメージを取得するためのコマンドです. 特に指定しない場合には DockerHub という Docker のイメージを無料で公開できるサービスから取得されます. docker pull <コンテナ名>:<タグ>という形式で記述します. タグを省略して, docker pull <コンテナ名>とすることもできます. タグは典型的にはイメージのバージョンを指定するためなどに用いられます. 上記の Jenkins を動かす例では, latest というタグが付いた jenkins という名前のイメージを取得しています.

3.2 docker run コマンド

docker run はイメージからコンテナを作成し, 実行するためのコマンドです. 相当な数のオプションが用意されているので, このオプションを憶えないといけないのが, やや面倒くさいですね. 今回は相当シンプルな例にも関わらず 3 つほどオプションがくっついています. --name オプションはコンテナに名前を与える際に使用します. コンテナに名前を与えておくと, 長くて憶えづらいコンテナ ID ではなく, 分かりやすい名前を使ってコンテナを管理できるので便利です. -d オプションはコンテナをバックグランドで実行させる際に指定するオプション, -p はホストとコンテナの間でポート・マッピングを行なう際に指定するオプションです. -p <ホストのポート>:<コンテナのポート>という構文になっています.

3.3 コンテナの停止・再開・削除

以下のコマンドでコンテナの停止・再開・削除が行なえます. 今回作った Jenkins コンテナはもう不要なので, 削除しておいて下さい. 基本的にコンテナの削除はコンテナの停止 → 削除 というステップでやることになっていますが, docker rm -f <コンテナ名>で動いているコンテナを強制的に削除することもできます.

```
$ docker stop  jenkins
```

```
$ docker start jenkins
$ docker rm    jenkins
```

3.4 データの永続化

今までの例では Jenkins のデータは永続化されていません. すなわち, コンテナを終了させると Jenkins に貯めておいたデータが全て吹っ飛んでしまいます. これではあんまりなので, Jenkins が扱うデータを永続化させてみます. Docker で永続的なデータを扱うためには-v オプションを用います. この節では複数回 Jenkins コンテナを立ち上げていますが, ポート・マッピングの問題などがあるので毎回コンテナは停止, あるいは削除するようにしてください.

```
$ docker run --name jenkins-1 -d -p 8080:8080 \
  -v /var/jenkins:/var/jenkins\_home \
  jenkins
```

-v オプションは「データボリューム」を作成するオプションです. Docker は特殊なファイル・システムの上に成り立っていて, ユーザーが特別に指定しない限り, コンテナ内で行なった変更は揮発してしまいます. 一方で, データボリュームとなったファイルやディレクトリはコンテナを終了させても揮発しません. このような特徴からデータボリュームは Docker で永続的なデータを扱う際に活躍します. 今回はコンテナ内の /var/jenkins_home ディレクトリをデータ・ボリュームとして指定し, そこにホストの/var/jenkins ディレクトリをマウントしています. Docker コンテナ内のデータは毎回揮発してしまいますが, ホスト環境のファイル・システムは (一般的に) そうではないので, ホスト環境のディレクトリを用いることでデータを永続化してやろうという作戦ですね.

他の永続化のやり方として「データボリューム・コンテナ」というものを使うやり方があります. データボリューム・コンテナの基本的なアイデアは永続的なデータを管理するためのコンテナを別の作成し, そのコンテナ自体をマウントしてやろうというものです.

```
$ docker run --name jenkins-data -v /var/jenkins busybox
$ docker run --name jenkins-2 -d -p 8080:8080 \
  --volumes-from jenkins-data}  jenkins
```

一行目のコマンドで, /var/jenkins ディレクトリをデータボリュームに指定したコンテナを作成します. データボリューム・コンテナはデータの保存だけが目的なので, 小さなイメージを用いることが多いです. 今回は busybox イメージを指定しています. docker pull していませんが, 実はローカルにないイメージを docker run しようとすると Docker 側が勝手に pull してくれるので, docker pull busybox は不要です. (もちろん, やっても良いのですが...) jenkins-data コンテナは起動するとすぐに停止してしまいますが, データボリューム・コンテナは起動している必要はないので, 問題ありません.

続いて二行目のコマンドに新しく--volumes-from というオプションが登場しています. これは他のコン

テナ内のデータボリュームをマウントするオプションです. この例では jenkins-data コンテナ内のデータボリューム, すなわち/var/jenkins_home がマウントされます.

データボリューム・コンテナを指定する方法はやや面倒ですが, ホストのディレクトリをマウントする方法に比べてコンテナのポータビリティを高くできるという利点があります. データボリューム・コンテナについての詳しい情報を得たい方は
https://docs.docker.com/v1.10/engine/userguide/containers/dockervolumes/
を参照してみて下さい.

3.5 Jenkins の設定

Docker から環境変数を渡すことで, Jenkins を設定することができます. 環境変数の設定には -e オプションを使います. あるいはファイルに記述して, --env-file オプションを用いてそれを読み込ませることもできます.

```
$ docker run -name jenkins-data -v /var/jenkins busybox
$ docker run --name jenkins-3 -d -p 8080:8080 \
  --volumes-from jenkins-data \
  -e JENKINS_PORT="8080" \
  -e JENKINS_OPTS="--prefix=/jenkins" \
  jenkins
```

JENKINS_OPTS に--prefix=/jenkins を指定すると http://localhost:8080/jenkins/ という URL で Jenkins にアクセスできるようになります.

4 GitBucket コンテナの起動

GitBucket はオープンソースの Git ウェブプラットフォームであり, Github に近い操作感や UI から人気の高いソフトウェアです. Jenkins の次は GitBucket を起動させてみましょう.

3 章では, Jenkins コンテナを起動する手順を通して, Docker の基本的な操作方法について紹介してきました. 実は Docker の操作おける基本的な部分については殆どここまでで紹介したことに尽きていて, GitBucket コンテナを立ち上げる上で新しいことはあまりありません.

```
$ docker run --name gitbucket-data -v /gitbucket busybox
$ docker pull takezoe/gitbucket
$ docker run -d -p 8081:8081 -p 29418:29418 \
  --volumes-from gitbucket-data \
  takezoe/gitbucket
```

http://localhost:8081 に移動して GitBucket のログイン画面が表示されることを確認しましょう. ちなみに root / root でログインできます.

5 Nginx コンテナを用いた別コンテナへのリバースプロキシ

3 章と 4 章でそれぞれ, Jenkins と GitBucket のコンテナ上での起動方法を紹介しました. この章では Nginx コンテナを用いたリバースプロキシを構築し, http://jenkins.localhost や http://gitbucket.localhost で Jenkins や GitBucket コンテナ内で動いている Web サービスにアクセスできるようにしたいと思います.

5.1 各コンテナの用意

Docker でコンテナ間を連携させる場合には link と呼ばれる機能を用います. 次のコマンドを実行して, Jenkins および GitBucket をリンクした Nginx コンテナを立ち上げてみます.

```
$ docker run --name jenkins-data -v /var/jenkins busybox
$ docker run --name jenkins -d \
  --volumes-from jenkins-data \
  jenkins

$ docker run --name gitbucket-data -v /gitbucket busybox
$ docker run --name gitbucket -d \
  --volumes-from gitbucket-data \
  takezoe/gitbucket

$ docker run --name nginx -d -p 80:80 \
  --link jenkins:jenkins  \
  --link gitbucket:gitbucket  \
  nginx
```

注目して欲しいのが, jenkins コンテナ及び gitbucket コンテナの起動オプションからポートマッピングの指定が消えているという点です. 以前はコンテナのポートをホスト環境側に露出させることでコンテナ内のサービスへとアクセスしていましたが, リンクを用いればその必要はありません. nginx コンテナからのみ jenkins コンテナや gitbucket コンテナ内のサービスへとアクセスできるようにし, 80 番ポートへのアクセスを Nginx を用いて振り分けます. コンテナ群が起動できたら, Nginx がリバースプロキシとして振る舞うように Nginx を設定していきたいと思います. バックグラウンドで動いているコンテナ内に入るには docker exec コマンドを利用します.

5.2 docker exec コマンド

docker exec コマンドはコンテナ内で新しいプロセスを立ち上げるためのコマンドです. このコマンドを利用して nginx コンテナ内で bash を起動します.

```
$ docker exec -it nginx /bin/bash
```

docker exec コマンドは docker exec <コンテナ名> <実行するコマンド>という構文です．ここではnginx コンテナ内で/bin/bash プロセスを起動せよ, という指示になっています. -it オプションは-i と-tの両方を指定するための省略記法ですが, それぞれ STDIN と pseudo-TTY をアタッチする役割を持っていて, shell のような対話的入出力を行なうコマンドを実行する際に必要なオプションになります. -it という指定は非常に良く用いられるイディオムです. ちなみに docker run コマンドでもシェルなどを扱う場合には同じように指定します.

5.3 --link を用いたコンテナ間の接続

nginx コンテナは jenkins コンテナと gitbucket コンテナを起動時にリンクさせています. Nginx の設定をする前に, リンクがどのような働きをするのかを確認しておきたいと思います.

/etc/hosts の内容を cat してみます. jenkins や gitbucket というフィールドが増えていることが確認できるでしょうか.

```
  $ cat /etc/hosts
127.0.0.1       localhost
...
172.17.0.4      jenkins 8205084fce20
172.17.0.5      gitbucket 5dd690d1594e
172.17.0.2      a8877b1591ff
```

(この出力は一例です. 全く同じものが出てくるわけではないので, そこで不安になる必要はありません)

続いて env コマンドで環境変数の一覧を出してみます.

```
  $ env
HOSTNAME=a8877b1591ff
JENKINS_PORT=tcp://172.17.0.4:8080
GITBUCKET_PORT_8080_TCP=tcp://172.17.0.5:8080
...
```

凄い数の環境変数が出てきますが, 注意して読むと Jenkins や GitBucket の設定などが環境変数としてマッピングされていることが分かります. このように, --link オプションはコンテナの接続に必要な情報をリンク先の環境変数や/etc/hosts に自動的に追加してくれるという機能になります. ではこれらの情報を用いてNginx の設定ファイルを書いてみましょう. 以後の作業はホスト環境側で行なうために exit コマンドを打つなりして一旦コンテナから抜けてください.

5.4 Nginx の設定

以下のような内容のファイルを default.conf という名前で作成してください. この設定におけるポイントは proxy_pass に指定する URL で, --link オプションによって/etc/hosts に各コンテナへの接続情報

が追加されているので, リンク時の名前を使って簡単にプロキシさせることができるという点です.

```
server {
  listen        80;
  server_name   jenkins.<your domain>;
  location / { proxy_pass http://jenkins:8080; }
}
server {
  listen        80;
  server_name   http://gitbucket.<your domain>;
  location / { proxy_pass http://gitbucket:8080; }
}
```

このファイルを Nginx コンテナへコピーし, Nginx をリロードします. リロードに成功したら, jenkins.<yourdomain> や gitbucket.<yourdomain> にアクセスしてうまく設定できていることを確認してください.

```
$ docker cp default.conf nginx:/etc/nginx/conf.d/
$ docker exec nginx nginx -t -c /etc/nginx/nginx.conf
$ docker exec nginx nginx -s reload
```

5.5 docker-compose を使ったコンテナを管理

--link オプションを使ってコンテナ間を接続する方法は何となく分かって頂けたでしょうか. --link オプションによる接続は単純で良いですが, 一方でコンテナの起動の順番などに影響されるので, 手動管理が面倒で仕方がないという問題があります. こういう場合には「オーケストレーション・ツール」と呼ばれる自動設定/管理ツールを使うのが一般的です. Docker におけるオーケストレーション・ツールとしては Docker-Compose というものを使うのが一般的で, これは元々 fig と呼ばれていたツールが公式化されたものです. この章では Nginx を使ったリバースプロキシ・システムに関係するコンテナ群を Docker-Compose を使ってまとめて管理する方法をご紹介します. [*1]

docker-compose コマンドは docker-compose.yml という YAML で記述された設定ファイルを元にコンテナの一元管理 (作成/起動/削除/etc..) を自動的に行なってくれるコマンドです. 以下のように YAML ファイルを設定します.

```
  nginx:
  image: nginx
  ports:
```

[*1] Docker 1.12 RC で Docker-Engine(Docker 本体) によるオーケストレーションのサポートが明言されたので, 近いうちに Docker-Compose は使われなくなるものと思われます.

```
    - 80:80
  volume:
    - /var/nginx/conf.d:/etc/nginx/conf.d
  links:
    - jenkins:jenkins
    - gitbucket:gitbucket

jenkins:
  image: jenkins
  volumes_from:
    - jenkins-data

gitbucket:
  image: takezoe/gitbucket
  ports:
    - 29418:29418
  volumes_from:
    - gitbucket-data
```

前章で作成した Nginx の設定ファイル `default.conf` を`/var/nginx/conf.d` ディレクトリ以下に配置しておいてください. これで, あとは以下のコマンドを実行するだけで一連のコンテナ群の起動/停止/削除を行うことができます.

```
$ docker-compose up -d
$ docker-compose stop
$ docker-compose down
```

6 Nginx-Proxy を使った各コンテナへのリバースプロキシの設定

ここから先は余談といいますか, 実は苦労して Nginx の設定をしてきたけど, 実は自動生成してくれるコンテナがあるよという紹介になります. ホストの`/var/run/docker.sock` というファイルをコンテナ側にマウントすると, ホストの Docker をコンテナ内から操作することができます. Docker は sudo と同じ権限で動くツールであり, コンテナ側からそれを直接弄れるというのはコンテナにホストの sudo 権を渡しているのと大体同じことなので, セキュリティ的な事を考えるとリスクが高いですが, このコンテナ自体は非常に便利なものです.

以下のような YAML ファイルを記述し, `docker-compose.yml` という名前で保存します. あとは普通に

Docker-Compose でコンテナ群を立ち上げるだけです.

```
nginx-proxy:
  image: jwilder/nginx-proxy
  ports:
    - 80:80
  volumes:
    - /var/run/docker.sock:/tmp/docker.sock:ro
jenkins:
  image: jenkins
  volumes_from:
    - jenkins-data
  environment:
    - VIRTUAL_HOST=jenkins.<your domain>
    - VIRTUAL_PORT=8080
gitbucket:
  image: takezoe/gitbucket
  volumes_from:
    - gitbucket-data
  environment:
    - VIRTUAL_HOST=gitbucket.<your domain>
    - VIRTUAL_PORT=8080
```

environment を使って VIRTUAL_HOST および VIRTUAL_PORT を設定するだけで, 自動的にリバースプロキシを構築することできました. コンテナをリンクさせる必要すらありません. 本番環境で使えるかどうかは別として, 例えば開発環境などで活用する分には非常に役に立ってくれそうです.

7 おわりに

Docker を利用して実際的なサービスを動かす方法や, その上でのリバースプロキシ構築の仕方をご紹介させて頂きました. Docker を操作する上で憶えておいた方が良い操作などは例中に一通り含めたつもりですので, ここまでつきあって下さった方は一応 Docker のイメージとして配布されているアプリケーションを手元で動かすくらいはできるようになっているはずです. たぶん.

この記事を読んで「Docker 簡単やんけ!」 と思って頂ければバッチリですが, (ここで Docker の導入に失敗した苦い経験を思い出す) 実際のところ, この記事の内容だけでは厳しいところもあると思います. Docker は参考になるドキュメント類が非常に豊富なツールですので, 適宜ウェブなどで調べながら進めて頂ければと思います.

8 落ち穂拾い

この記事では触れられなかった幾つかの点についてキーワードと参考 URL を残しておきます.

- コンテナ・イメージの作成方法, Dockerfile の記述方法
  `https://docs.docker.com/engine/tutorials/dockerimages/`
- `docker commit` コマンド Docker 上で行なった変更を反映させた新しいイメージを作成するコマンドです. イメージが完全にブラックボックス化してしまうので, Dockerfile を用いたイメージ作成の方が好まれますが, 憶えておくと役に立つことがあるかもしれません.
- `docker attach` コマンド/コンテナからの detach
  Ctrl+p Ctrl+q というキー・ストロークで, コンテナからデタッチできます. これを憶えていないと後々すごく困ったことになります. アタッチする方法については公式のドキュメントを参照してください.
  `https://docs.docker.com/engine/reference/commandline/attach/`
- Private Repository を作成する方法.
  DockerHub は無料で Public なコンテナ・イメージを公開できますが, Private なものを置きたい場合には自分で立てる必要があります.

  参考:「Registry v2 と Let's Encrypt を使って Docker Private Repository を構築」
  `http://qiita.com/linerlock/items/119b7b2c17f83173a4f7`

LXD 入門
〜利用例編〜

文　編集部　ひだるま

1　はじめに

引っ越したら部屋が 1.5 帖狭くなりました。ひだるまです。

「LXD とはなんぞや」という方は先号の記事[*1]をどうぞ。今回は「LXD ってどこで使うの？」ということで、実際の利用例について触れます。「Docker でいいじゃん」といった声もあるかと思いますが、「Docker じゃなくてもいいじゃん」「2kg 太った」「今月は 7 本買う[*2]」といった声もあるかもということで、よろしくお願いします。

ライブマイグレーションについては不勉強なので取り扱いません。

コンテナ環境の魅力はやはり取り潰しが楽なことです。環境ごと消しとばせば妙な依存は発生しませんし、ホスト環境はそのまま残せるので再構築の手間が減ります。コンテナのコピーなどを使えば更に楽ちんに。

2　VPS

先号の記事ではローカルでのみ LXD/LXC を扱いました[*3]が、今回は VPS[*4]上で動かしている例を見てみましょう。

僕はさくらの VPS2G プランを使っています。選択可能な OS に Ubuntu16.04 があるので LXD 環境構築に関しても手間が少ないでしょう。

3　動いているサービス

図 1 に構成を示しています。ここでは仮に「hogefuga.com」としています。

3.1　nginx(Host 側)

Web サーバー。各コンテナに飛ばすためにホスト側に建っています。サブドメインによって飛ばす先を変えてくれています。

3.2　GitBucket

GitHub ライクだったリポジトリ管理ツール。「似すぎ」って注意されてちょっと変えたらしいですね[*5]。Tomcat などと合わせることもできますが、単一で動かしています。

[*1] WORD40 号「LXD 入門に美少女ゲェムをした話」

[*2] その後、厨二姫、カタハネ HD、ちちないが延期したので 4 本に

[*3] 僕は GIMP、Wine などをコンテナ運用しています

[*4] Virtual Private Server

[*5] http://takezoe.hatenablog.com/entry/2016/03/21/031733

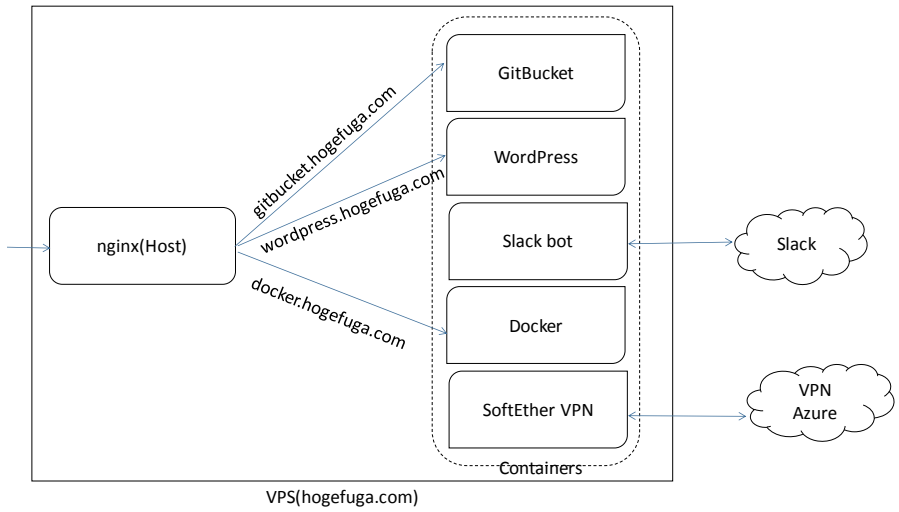


図 1: VPS 上の構成

先号の記事では ssh を多用していましたが、`lxc exec`コマンドの利用の仕方によってはその必要無くサービスを組み立てられます。

3.3 WordPress

有名 CMS。ですが、コンテナ内で建てては潰す遊びを繰り返しています。まともに建てようとするとセキュリティへの意識が大変なサービスですよね。環境を閉じこめる意味ではそこそこ LXD に向いている気がします。

3.4 Slack bot

幾つかの言語による Slack[*6] の bot を遊ばせるコンテナです。それぞれで発行したトークンを用いて Slack と通信します。

3.5 Docker

「Docker じゃん」と思った貴方。Docker です。LXD 上で動くんですよ実は。無限の可能性を感じつつも実運用はしていなかったり。VPS 上で行う際にハマりやすいのは「localhost にモノを建ててもアクセスできない」といったことでしょうか。

Docker in LXD 入門は LXD 2.0: Blog post series の該当記事[*7] をどうぞ。

[*6] 多分人気のチャットツール `https://slack.com/`

[*7] `https://www.stgraber.org/2016/04/13/lxd-2-0-docker-in-lxd-712/`

3.6 SoftEther VPN

VPN[*8] サーバーです。「LXC と SoftEther を利用してサーバー上にセキュアな作業システムを作る[*9] 」を参考に組みました。

4 Juju

コンテナでのサービス管理を更に楽にしたい方は、Canonical が開発しているサービスオーケストレーションツール Juju[*10] をどうぞ。WordPress なんかはあっという間に建ってしまいます。

僕ですか？ `lxc`コマンドと `juju`コマンドをぐちゃぐちゃに使ってたら環境が壊れたので使用を止めました。

5 終わりに

ソフトウェア紹介みたいになりましたが、こんな感じの運用例がある、という話でした。「LXD/LXC でなくてはできないこと」というのはあまりないかもしれませんが、「LXD/LXC でやってもいいこと」というのもそこそこあることを分かっていただけたなら幸いです。

そういえば Windows 10 タブレットを踏み壊しました。液晶保護シールを貼っていて良かったです (怪我をしなかった)。

[*8] Virtual Private Network

[*9] `http://ever-day-jp.blogspot.jp/2015/02/lxcsoftether.html`

[*10] `jujucharms.comarms.com`

メタデータ、整理させてください！

文　編集部　polamjag

“教えてよ 君のほんとうを 叫んでも姿は見えない”
—「Beyond the Bottom」 作詞: 辛矢凡

“MPEG” といえば皆さんの頭には何が浮かぶだろうか。たいていは動画・音声のコーデックやコンテナフォーマットの仕様の集合である MPEG-2 や MPEG-4 であろうが、それ以外にも MPEG によって策定された規格は存在する。

ここで紹介したいのは 2002 年に策定された MPEG-7 (ISO/IEC 15938): Multimedia content description interface である。これは、動画や音声の様々なメタデータを XML で記述するためのスキーマからなる。

策定当時は、曲であればアーティストやアルバム名、動画であればエピソード番号や監督・キャストといった主要なメタデータの他に、動画コンテンツのキャプションや、動画・音声の詳細なストリーム情報などを記録することも重要な目標となっていたようである。しかしながら、現在ではそのような活用はごく一部の放送局の R&D からその面影を感じとることができる[*1] のみである。これは、当時よりも計算機の性能や音声・画像の認識技術が飛躍的に向上したことで、そのような情報の自動認識が実現されつつある[*2] ことなどが原因として考えられる。

また、その他のマルチメディアのメタデータを XML で表現するための存在としては Adobe が開発した XMP[*3] が有名であるが、こちらも殆どプロフェッショナル用途で用いられているのみというのが私の肌感覚である。

1　楽曲と “タグ”

MPEG-7 は他の MPEG 標準と同様にいくつかの Part からなる。ここでは、Part 5 である Multimedia Description Schemes について考える。Part 5 は、オーディオやビデオといったメディアの種別を問わない、汎用的な “マルチメディア” のメタデータについて記述する方法を示しており、その主要な実体は 5000 行近い XML スキーマである。

1.1　み〜んなメタデータ

プリパラというアニメ・ゲームなどからなるコンテンツがあり、それに伴って楽曲が CD やダウンロード販売などにより提供されている。プリパラにおいては、一般的なアイドルアニメと同様に、声優がそれぞれキャラクターを演じ、曲によっては複数のキャラクターからなるユニットにより歌唱される。

[*1] NHK の技研などがやっています

[*2] たとえば YouTube は自動で音声認識の結果をクローズドキャプションとして表示できる

[*3] Extensible Metadata Platform. http://www.adobe.com/products/xmp.html

ところで、次に示すのはそのプリパラのとある楽曲 [1] のタグ情報を表示した画面である。

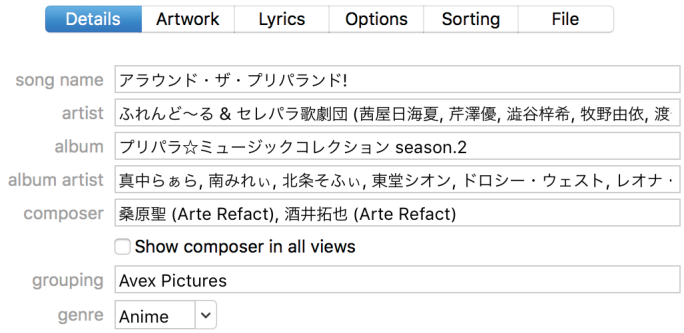


図 1: 「アラウンド・ザ・プリパランド!」のプロパティの一部

ここで、上から 2 番目のフィールドである `artist` 欄に注目をしてほしい。画像では途中で途切れてしまっているが、

> ふれんど〜る & セレパラ歌劇団 (茜屋日海夏, 芹澤優, 澁谷梓希, 牧野由依, 渡部優衣, 斎賀みつき, 久保田未夢, 山北早紀, 若井友希, 赤﨑千夏

とある。これは「ふれんど〜る」というユニットと「セレパラ歌劇団」というユニットがあり、それらに参加しているキャラクターの声優を後に続くカッコの中に列挙しているものである。

先に示した MPEG-7 では、これを正しく表現することができない。なぜなら、MPEG-7 では人間の集団をグループとして表現することができる[*4]が、キャラクター — 声優という関係性を明確に表現できないからである。

キャラクターごとに一アーティスト扱いとし、それらからなるグループを扱えるだけでも実用上は問題ないように思うが、それでもそこにキャラクター — 声優という関係が存在し、またこのスキーマ自体が極めて高度なものである以上は[*5]、それをデータとして表現して扱いたいと思うのは自然なことではないだろうか。

このように、非常に高度な MPEG-7 でさえ、完全には扱うことができないデータ構造が存在する。

1.2 アイディースリーカツドウ

現実的には、ほとんどのケースで楽曲のメタデータは ID3 タグ、並びにそれに類似したフォーマットとして表現される[*6]。しかし、ご存知のように ID3 タグに含むことができる一般的な情報はたかだか有限長の文

[*4] PersonGroupType という要素がありましてね

[*5] 複雑なので実際誰も使っていない程度には高度

[*6] 実は AAC などのコンテナとして用いられる MOV / MP4 に埋め込まれる楽曲メタデータのフォーマットが完全に標準化されてはいない

字列[*7]であり、MPEG-7 のようなフォーマットに比べると表現力の点で圧倒的に劣る。

このような状況であるので、各々が思い思いの表現法を編み出し、それを使用していくということになるが、このようにメタデータのスキーマや表現力が貧弱なために、メタデータの高度な活用など夢のまた夢という状況である。

例えば、ある CD に同一楽曲のボーカル有り・無し版がそれぞれ両方収録されていたときに、その判定がボーカル無し版のタイトルに含まれる (Off Vocal) や (Instrumental) のような文字列に依存しているなど、自明な問題が無限に存在する。

2　おわりに

ここまで示したように、楽曲のメタデータを完全にシリアライズするという試みは、端的に言って極めて厳しい状況にあるということがお分かりいただけたと思う。

次号では、このような状況を踏まえ、iTunes で利用できるメタデータを駆使することで膨大な曲数を管理し、効率的にブラウズする方法についての私の考えを解説したい。

References

[1] 真中らぁら, 南みれぃ, 北条そふぃ and others (CV: 茜屋日海夏, 芹澤優, 久保田未夢 and others)『プリパラ☆ミュージックコレクション season.2 DX』 (エイベックス・ピクチャーズ株式会社, EYCA-10983〜4, 2016) Track no. 17「アラウンド・ザ・プリパランド!」

[*7] とはいえ最近のバージョンだと長さが問題になることはほとんどありえない

続・LuaRocksで結果にコミット

文　編集部　びしょ〜じょ

1　はじめに

こんにちは、びしょ〜じょです。さて、前回 [*1] は luarocksの主要なサブコマンドについて紹介したら赤入れの時間になりました。今回はどうなるでしょうか。勝負です。

2　レッツロック! ギュイ〜ン [*2]

LuaRocks で用いられる **rockspec** ファイルを作成して、パッケージをビルドしたりアップロードしたりなんだりしよう。

2.1　rockspec

rockspec ファイルとは、RubyGems でいうところの **Gemspec** みたいなものです。PKG-VERSION.rockspec というファイル名で、パッケージのバージョンやライセンス、ファイルのリポジトリ、依存関係などを書いていきます。実は実行可能な lua ファイルで、主に複数の `table` から成り、限られた定数や関数の呼び出し、もちろん `if`文などの statement が使用可能です。

作成には `write_rockspec`サブコマンドを使用します。

```
$ luarocks write_rockspec [--output=<file> ...] [<name>] [<version>] {<url>|<path>}
```

よし、ア〜ナンダ、もうダイレクトに行くぞ。実際に作ってみましょう。

2.2　build rockspec

極めて簡単なパッケージを作ってみましょう。図 1 のような構造の物体を考えます。

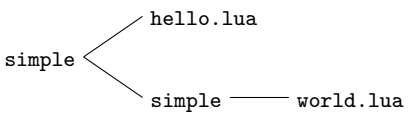


図 1　a package directory structure

[*1] http://www.word-ac.net/archives/2016/07/1252

[*2] 大人気女児アニメ『アイカツ!』の玩具、『ガーリーロックギター』に収録されている音声。正しくは「さぁみんな、レッツロック! ギュイ〜ュィ〜ン」。演奏モード、占いモードなどがある。

```
luarocks --local install word_articles
```

Listing 1　hello.lua

```
return function()
  print("hello")
end
```

Listing 2　simple/world.lua

```
return function()
  print("world")
end
```

simple をルートディレクトリとして考えていきます。

create hinagata

まずは雛形を作ります。手で書くのもアリですが、コマンドラインでオプションを渡して作るのもアリです。パッケージ名は **simple**、バージョンは **v1.0** としてみますか。

```
$ luarocks write_rockspec simple v1.0 ${PWD}
```

とルートで実行すると、simple-v1.0-1.rockspec というファイルができると思います。最後の引数で rockspec ファイルの生成場所を指定しますが、相対パスでも問題ないです。中身はこんな感じですか。

Listing 3　simple-v1.0-1.rockspec

```
package = "simple"
version = "v1.0-1"
source = {
   url = "*** please add URL for source tarball, zip or repository here ***"
}
description = {
   homepage = "*** please enter a project homepage ***",
   license = "*** please specify a license ***"
}
dependencies = {}
build = {
   type = "builtin",
   modules = {
      hello = "hello.lua",
```

```
15      ["simple.hello"] = "simple/hello.lua"
16    }
17  }
```

バージョンのケツに何かつきましたね。これは Lua のバージョニング規則に合わせたいのかなぁという気がしますね。ファイル名にあるバージョンと rockspec ファイル内の `version`が一致しないと怒られが発生するので気をつけてください。

`source`から順に見ていきましょうか。

source

パッケージのソースコードを引っ張ってきます。git サーバーのリポジトリや zip ファイルの場所を書きます。型は `table` で、`source.url`と `source.tag`という要素を持ちます。`url`にはソースコードの URL、ソースコードの URL が git のリポジトリを指している場合は `tag`でタグを指定できます。

一つ注意として、git リポジトリの URL を書く場合は、プロトコルの部分を git にしてください。例えば `https://github.com/nymphium/simple` というリポジトリにソースコードがある場合は、`url = "git://github.com/nymphium/simple"`となります。

コマンドラインからの指定も可能で、URL を指定する場合は path の位置に URL を書きます。この場合 rockspec ファイルが生成されるディレクトリはコマンドを実行した場所に生成され、ディレクトリを指定できなくなります。タグは`--tag=TAG`で指定します。

```
$ luarocks write_rockspec simple v1.0 'git://github.com/nymphium/simple' --tag=v1.0
```

```
......
source = {
   url = "git://github.com/nymphium/simple",
   tag = "v1.0"
}
......
```

イイネ・

description

説明やライセンスを記述します。これも `table` 型で、`description.summary`、`description.detailed`、`description.homepage`、`description.license`という要素を持ちます。`summary`には一行程度の説明、`detailed`には詳細、ほかも名前のとおりですね。

コマンドラインからももちろん指定可能です。上記のように URL を指定していると `homepage`はその URL

になります。git リポジトリの URL だった場合はプロトコルを http にしてくれます。

逆に指定しないと README ファイルなどから適当に引っ張ってきて作られます。本当に適当なので、指定しない場合は手動で編集しましょう。

```
$ luarocks write_rockspec simple v1.0 'git://github.com/nymphium/simple' --summary="
    this is the test package" --detailed="this is the test package\nfor the 41th
    WORD's article" --license=wtfpl
```

```
......
description = {
   summary = "this is the test package",
   detailed = "this is the test package\\nfor the 41th WORD's article",
   homepage = "http://github.com/nymphium/simple",
   license = "wtfpl"
}
......
```

`detailed`にはコマンドラインからの注入では改行してくれないようですね [*3] 。改行したい場合は手書き必須です。一般的にはエスケープが必要な文字や改行が多く含まれるので、`long string`[*4] がよく用いられます。

```
......
description.detailed = [[
this is the test package
for the 41th WORD's article
]]
......
```

dependencies

他のパッケージへの依存関係、動作する Lua のバージョンを記述します。パッケージ管理の重要な部分その 1 ですね。

依存関係を書いた `string` が 0 個以上の `table` から成ります。`string` は `PKG [OP]` という感じの書式になっています。たとえば v3.0 以降ののパッケージ foo に依存している場合は、

[*3] 本当は `printf`コマンドを挟む、つまり`--detailed="this is the $(printf "package\nfor") the......"`などするとできるが……。

[*4] `[[this is "long string"]]`、角括弧の間に`=`を 0〜5 個入れることで “レベル” を指定できる。

```
......
dependencies = {
  "foo >= v3.0"
}
......
```

といった具合です。バージョニングはパッケージによりけりなので、バージョンの指定に関しては適宜調べてください。バージョンの比較演算子は以下のようになっています。

- `==`

  等値

- `~=`

  非等値

- `>`

  真に小さい

- `<`

  真に大きい

- `<=`

  以下

- `>=`

  以上

- `~>`

  pessimistic operator というらしい。`~> 2`が`>=2 and <3`と翻訳され、`~> 2.4`は`>=2.4 and < 2.5`となる。

- `""`

  `==`に変換されるよ。

- `=`

  これも`==`

- `!=`

  驚かせて申し訳ないですが、Lua にこのような演算子はありません、`~=`に変換されるわけですね。

上記の演算子で Lua のバージョンも指定できます。

これはコマンドラインオプションで指定できないようなので、頑張ってください。

さて温まってきたところで `build`の説明と行きたいところですが、それを書くには時間が短すぎる。

3 おわりに

最悪なところで切りました。最高です。

気づけば前回の記事が春の入り口だったのにこの記事が発刊される頃には夏ですね。『りりくる Rainbow Stage!!!』[*5] の発売は 1 ヶ月延びましたが皆さん無事にプレイできたでしょうか。私は途中ですがもう最高!!! 最っ高〜〜!!! 『FLOWERS』秋篇 [*6] もまだ始めてすらないじゃん!!! プログラム書くのやめやめ〜〜っゲームしなきゃ!!!

[*5] 百合、GL をテーマにした全年齢コンテンツブランド PRATICLE による『りりくる』ドラマ CD シリーズのメンバーに、桜庭遊乃 (CV. 木村珠莉)、仁篠珠季 (CV. M・A・O)、高月紗恵香 (CV. 照井春佳) の 3 人を加えた百合 ADV。直球タイプの百合ですが、もう最高なんですね、直球でガツンと正面から殴ってくる。パワーが強い。

[*6] 言わずもがな。

ドキッ！ systemdだらけの大運動会

文　編集部　azuma962

こんにちは。皆さんはエンジョイしていますか？

systemd は非常に便利である事が知られており、その機能は既に単なる init 代替品の域を超えています。今回は、そんな systemd クンの便利なコマンド達を見ていきましょう。

1　ちょっとその前に

OS？ やら systemd のバージョン？ やらを明記しておきます。大抵のソフトウェアにおいて重要なこのファクターですが、systemd においてもそれは同じです。

Operating System　Ubuntu 16.04 LTS

systemd のバージョン　229

2　敵を知る

`systemd`の大抵のコマンドには、`systemd-`という枕詞が付けられており、非常に便利です。

Listing 1: systemd の息がかかったコマンドの一覧がスッと見れて便利

```
$ systemd-<Tab><Tab>
systemd-analyze
systemd-ask-password
systemd-cat
systemd-cgls
systemd-cgtop
systemd-delta
systemd-detect-virt
systemd-escape
systemd-hwdb
systemd-inhibit
systemd-machine-id-setup
systemd-notify
systemd-path
systemd-resolve
systemd-run
systemd-stdio-bridge
systemd-tmpfiles
```

```
systemd-tty-ask-password-agent
$ systemd-
```

あとは、*ctl系なんて物もあります。

Listing 2: *ctl をシュッ

```
$ ls /usr/bin/ /bin/ | grep 'ctl$'
journalctl
loginctl
networkctl
systemctl
wdctl
bootctl
busctl
hostnamectl
localectl
timedatectl
```

色々コントロォールできそうで善いですね。

ところで、Qiita に触れた事のある方は、systemd-nspawnや machinectl[*1] が一覧に無い事を疑問に思ったのではないでしょうか。 それは、systemd が modular[*2] な事に起因します。実は、systemd でも、コンパイル時にどのコマンドを組み込むか指定する事ができるんですよ。もう Bloated なんて言わせない、そんな心意気が伺えますね。

3 controlするヤツら

ここでは、様々な*ctlコマンドについて軽く見ていきます。

man の NAME セクション以上の情報を載せるつもりはありません。詳しく知りたければ、manしてくれよな。

3.1 bootctl

旧き良き時代に流行っていた UEFI ブートマネージャである gummibootも、今となっては、systemd の一部となり、名前も systemd-bootになってしまいました。systemd の一部になったので、それを管理するコマンドも当然存在します。

[*1] Docker や LXC/LXD に代表される、コンテナ型仮想化の systemd による実装。実力派 *IT* ポエマーこと、岡部健クンも注目しています。詳しくはコチラをクリック →http://kenokabe-techwriting.blogspot.jp/2016/02/dockersystemd-nspawnmachinectl.html

[*2] Lennart クンもそう言っています。http://0pointer.de/blog/projects/the-biggest-myths.html の “6. Myth: systemd is not modular.” をご覧下さい。

3.2 busctl

bus は D-Bus の Bus です。つまりはそういう事です。

3.3 hostnamectl

OS をインストールする時に一番悩む要素、hostname について管理するコマンドも存在します。

Listing 3: hostname, Icon name, and Chassis

```
$ hostnamectl status
   Static hostname: ubuntu
         Icon name: computer-vm
           Chassis: vm
        Machine ID: 701391523debf967b6615353576ad87b
           Boot ID: 7022de93b88c4d61bbb191ac7f448c29
    Virtualization: qemu
  Operating System: Ubuntu 16.04 LTS
            Kernel: Linux 4.4.0-24-generic
      Architecture: x86-64
```

/etc/conf.d/netにガリガリ手書きしていた時代と違って、hostname 以外にも色々管理できそうですね。

3.4 journalctl

systemd 開発チームによるログ収集プログラムの実装に、journaldという物があります。

journalctlは、journaldによって蓄積された、バイナリ形式[*3] の journal を読み出すためのコマンドです。

3.5 localectl

ハイ、もちろんロケールだって管理できますよ。

Listing 4: 順当な管理要素

```
$ localectl status
   System Locale: LANG=en_US.UTF-8
       VC Keymap: n/a
      X11 Layout: us
       X11 Model: pc105
```

3.6 loginctl

ConsoleKit の代替こと systemd-logind を管理するヤツです。機能がいっぱいで楽しそう。

[*3] 速い！！！

3.7 networkctl

バージョン 210 から、systemd にネットワーク設定機能が生えました。`networkctl`は、その機能を扱うヤツです。そう、最早 `NetworkManager`なんていうクソと付き合う必要は無くなったのです。

3.8 systemctl

読んで字の如く、system を管理します。ここで言う system は、systemd の本業である、init の事です。

3.9 timedatectl

時間の変更だってできます。バージョン 213 から systemd に NTP クライアントが生えたので、NTP についても設定できるようになったぞ。

3.10 wdctl

watchdog の状態を見る事ができます。control を想起させる `ctl`という単語ですが、watchdog をコントロールする事自体おこがましい事だと思いませんか？

4 洪水のような “systemd-*”

時間も無いので、ササッと見ていきましょう。時は金なり、Time flies like an arrow です。

4.1 systemd-analyze

システムの起動時間を計測したり、起動時のボトルネックを探し出したり、unit ファイル[*4]のシンタックスチェックができたりします。

Listing 5: systemd による素早い起動に皆にっこり

```
$ systemd-analyze time
Startup finished in 3.930s (kernel) + 3.027s (userspace) = 6.957s
```

systemd Boot Loader Interface[*5]に対応したブートローダー[*6]を使えば、Boot Loader でもたついた時間も見える化できるらしいですわよ、奥さん。

4.2 Password Agent

バージョン 12 で、ユーザーからパスワードを聞き出すための便利なツールが整備されました。`systemd-ask-password`が、そのツールです。

Listing 6: シェルスクリプトで役立ちそう

```
$ pass=$(systemd-ask-password)
Password: ****
$ echo $pass
```

[*4] systemd の設定ファイル群の総称です。

[*5] `https://www.freedesktop.org/wiki/Software/systemd/BootLoaderInterface/`

[*6] 代表的な物に、~~Gummiboot~~ systemd-boot があり、まだ、それ以外は発見されていない。

```
hoge
```

privileged なユーザーだと、もっとスゴい事ができます。

Listing 7: TTY 間通信

```
tty1 # systemd-ask-password --no-tty
Broadcast message from root@ubuntu (Fri 2016-07-08 15:21:05 JST):

Password entry required for '(null)' (PID 2297).
Please enter password with the systemd-tty-ask-password-agent tool!

hoge
tty1 #

tty2 # systemd-tty-ask-password-agent
Password: ****
tty2 #
```

systemd-ask-passwordでパスワードを要求し、systemd-tty-ask-password-agentでパスワードを返答する、そんな世界が構築できちゃうんですね。スゴォイ

他にも、systemd-ask-passwordでパスワードを kernel の keyring に格納したり、systemd-tty-ask-password-agent以外の、systemd Password Agent Specification[*7] に対応した Password Agent で返答できたりと、可能性は無限大です。

4.3 journald 関連

D-Busやら journaldでは、色々ななんやらのために、ユニークな machine-idを必要としています。systemd-machine-id-setupは、その machine-idを生成するためのコマンドです。man によると、“systemd-machine-id-setup may be used by system installer tools” らしいので、気にする必要は無いでしょう。

journal にログを流し込むコマンドだって当然存在します。それが、systemd-catです。

Listing 8: プロセス名までバッチリ

```
$ systemd-cat ls -l
$ ls -l | systemd-cat
$ journalctl -n 4
-- Logs begin at Fri 2016-07-08 15:42:09 JST, end at Fri 2016-07-08 17:32:49 JST. --
Jul 08 17:32:48 ubuntu ls[2616]: total 0
```

[*7] http://www.freedesktop.org/wiki/Software/systemd/PasswordAgents

```
Jul 08 17:32:48 ubuntu ls[2616]: -rw-r--r-- 1 root root 0 Jul 8 17:31 hello
Jul 08 17:32:49 ubuntu cat[2618]: total 0
Jul 08 17:32:49 ubuntu cat[2618]: -rw-r--r-- 1 root root 0 Jul 8 17:31 hello
```

4.4 cgroups 関連

systemd は、cgroups を使ってプロセスの監視やらなんやらを行っている訳ですが、下記 2 つのコマンドは、cgroups の状態を監視するための物です。

- systemd-cgls
- systemd-cgtop

詳しくは man を見ろ

4.5 systemd-delta

皆さんも、一度は設定ファイルの優先度で悩まされた事があるのではないでしょうか。systemd には、そんな皆さんの苛立ちを解消するためのコマンドが、当然のように存在します。

Listing 9: 設定ファイルの delta を見る

```
$ systemd-delta --no-pager
[EXTENDED] /lib/systemd/system/systemd-timesyncd.service →
    /lib/systemd/system/systemd-timesyncd.service.d/disable-with-time-daemon.conf
[EXTENDED] /lib/systemd/system/rc-local.service →
    /lib/systemd/system/rc-local.service.d/debian.conf

2 overridden configuration files found.
```

4.6 systemd-detect-virt

仮想マシンやらコンテナの上では動かしたくないソフトウェアだって、きっとある。systemd では、動作環境を確認するコマンドを御用意しております。どんな時にも便利な systemd です。

Listing 10: qemu 上で動いてるんだって！

```
$ systemd-detect-virt
qemu
```

Listing 11: 実機で動いてるんだって！

```
$ systemd-detect-virt
none
$
```

実際は Docker のコンテナ内で走らせているのに noneとか言ったりする愛嬌も、systemd の魅力だと思います。

4.7 systemd-escape

Listing 12: ASCII 文字への脱出

```
$ systemd-escape あいうえお
\xe3\x81\x82\xe3\x81\x84\xe3\x81\x86\xe3\x81\x88\xe3\x81\x8a
$ systemd-escape hello
hello
$ systemd-escape helloエヴィリワン
hello\xe3\x82\xa8\xe3\x83\xb4\xe3\x82\xa3\xe3\x83\xaa\xe3\x83\xaf\xe3\x83\xb3
```

ここで言う escape は、escape sequence で言う escape と同じ意味です。ASCII 文字以外を含む unit 名を付けたい時に便利そうですね。

4.8 systemd-hwdb

/dev管理ツールとして有名な udevですが、そんな彼には、ハードウェア特有の情報について格納するための、hwdbというデータベースがあります。systemd-hwdbは、そのデータベースを管理するためのコマンドです。

実は、udevも systemd の一部なんですよ。ご存知でした？

4.9 systemd-inhibit

systemd は、実は ACPI イベントにも反応する事ができるんですよ。画面を閉じた時にスリープする、そんなしょうもない事をするためだけに、acpidとかを入れる必要は、もはや無いんですね。

systemd-inhibitは、電源すらも制覇した systemd へ、灯を灯し続けるようお願いするためのコマンドです。

Listing 13: 寝ないでくれ〜〜〜

```
# systemd-inhibit --what=sleep sleep 10
```

4.10 systemd-path

様々な Path を写し出します。

Listing 14: systemd 配下の Path 達

```
$ systemd-path
temporary: /tmp
temporary-large: /var/tmp
system-binaries: /usr/bin
```

```
system-include: /usr/include
system-library-private: /usr/lib
system-library-arch: /usr/lib/x86_64-linux-gnu
  ...
```

4.11 ネットワーク関係

ネットワーク設定機能が生えたので、当然のようにドメイン名やらなんやらを解決する機能も追加されました。systemd-resolveが、ソレです。

Listing 15: host やら nslookup やらは必要無かったんや……

```
$ systemd-resolve google.com
google.com: 172.217.25.110

-- Information acquired via protocol DNS in 62.2ms.
-- Data is authenticated: no
```

4.12 systemd-run

古の時代には、普通のプログラムを Daemon として実行する daemontoolsが流行っていたと聞きます。systemd-runは、そんな daemontoolsの systemd 版です。

Listing 16: 簡単 daemonize

```
# systemd-run --unit=myunit sleep 100
# systemctl list-units myunit.service
UNIT LOAD ACTIVE SUB DESCRIPTION
myunit.service loaded active running /bin/sleep 100

LOAD = Reflects whether the unit definition was properly loaded.
ACTIVE = The high-level unit activation state, i.e. generalization of SUB.
SUB = The low-level unit activation state, values depend on unit type.

1 loaded units listed. Pass --all to see loaded but inactive units, too.
To show all installed unit files use 'systemctl list-unit-files'.
```

4.13 systemd-tmpfiles

systemd はテンポラリファイルも管理しています。systemd-tmpfilesは、管理されているテンポラリファイルを削除したり作ったりする、重要なコマンドです。

Listing 17: 管理社会

```
# touch /tmp/gomi
# ls /tmp
gomi
# systemd-tmpfiles --remove
# ls /tmp

#
```

/tmp/も、systemd によって管理されているので、systemd-tmpfilesであれこれを管理できるワケです。テンポラリファイルに関する設定は、/usr/lib/tmpfiles.d/等に格納されているので、是非見ましょう。

4.14 systemd-stdio-bridge

“Connect STDIO to a given bus address.” だそうです。ドキュメントが何処にあるかすら分からない、神秘的なコマンドです。

4.15 systemd-notify

“Notify the init system about service status updates.”、この文章が全てを物語っています。

daemon 達が、init[*8] に対して起動が終了した事を伝える時に使います。

5 まだまだあるぞ、systemd

systemd には様々な機能があります。コマンドに注目したこの記事で、それらを紹介する気はありません。詳しく知りたいなら、freedesktop.org の systemd に関するページ[*9] に向かいましょう。

このページから飛べる「The Biggest Myths[*10]」は、エモいのでおすすめです。

6 おわりに

このように、無限の機能を秘めた init、systemd。皆さんもこの波に飲まれてみませんか？

私は Gentoo Linux[*11] に逃げました。

[*8] もちろん systemd です。

[*9] https://www.freedesktop.org/wiki/Software/systemd/

[*10] http://0pointer.de/blog/projects/the-biggest-myths.html

[*11] 標準の init として OpenRC が採用されている。

量子にふれたよ！ #1

文　編集部　烏

1　はじめに

今日は量子コンピュータのお話デス☆マスター♡ 今年の 5 月 4 日に IBM から誰でも使える量子コンピュータが発表されましたね。今まで！ 本当に 5 月 4 日以前は！ 量子コンピュータがインターネットを通じて誰もが使えるようになるなんて、夢のまた夢だったものを..... とにかく、この天 (アメリカ) から降ってきた幸福に絶対に乗っかるしかありません。私自身、量子力学や量子コンピューティングには強い関心がありつつも、それこそ机の上、頭の中で量子ビットをクルクル回すしかなかったのが、直感的に、どころかホンモノの量子コンピュータを触れるようになったのはすごいことですよ！ これは！ (大声)

しかし、しかしですよ、せっかく様々なニュースで公開され、興味を持っている人がたくさんいるにも関わらず、皆使ってないのは本当にもったいないことです。これは機室のマシンに入って AppleScript を叩くことで Hack(イタズラ) をする行為が流行っていないのと同じくらいもったいないことです[*1]。せっかくなので、量子力学はもちろん量子コンピュータに詳しくない人も恩恵を受けられるようにと思ってこの記事を書きます。件の量子コンピュータは IBM Research Quantum Experience[*2] という名前で公開されています。実は登録してから使えるようになるまで、一週間くらいかかったりするのでさっさと登録しちゃいましょう。

これはあなたのアカウントが作られる一週間のための読み物です。

2　古典くんと量子ちゃんの違い

「分かんない... 分っかんないよ。
寿くんの言ってる事は一つも分かんないよ
寿くんがいいって言ってるもの何がいいのか分かんないよ！」

上記は、テレビアニメ『異能バトルは日常系のなかで』の第 7 話の櫛川鳩子さんが主人公の中二病を批判するシーンで放つセリフです。量子コンピュータの何がすごいのか、目の前の Macbook と何が違うのか一切わからない人がこの記事を読もうと思った時に出てくる感想はこんなものだろうと想像できます。

2.1　扱う情報が違う！

何故量子コンピュータがそんなにも魅力的なのかをお伝えしておかないと、鳩子さんのように読者の皆さんが愛想を尽かしてしまうやもしれません。この章では、とりあえず従来のコンピュータ (古典的なコンピュータと呼ばれます) と量子コンピュータの**扱う情報の違い**を皆さんにご説明いたします。

[*1] 一部の coins は音が出るタイプの Hack を恐れるあまり、ログイン時に **set volume 0** をループさせてるって本当ですか？

[*2] `https://quantumexperience.ng.bluemix.net`

古典的なコンピュータでは古典的情報である Cbit(古典ビット) を扱いますが、量子コンピュータでは Qbit(量子ビット)[*3] と呼ばれる量子情報を扱います。量子コンピュータの魅力を語る上で、Qbit についてのお話は欠かせません。Qbit は箱の中の猫[*4] と同じで、観測されるまでどちらの状態であるかわからない、というよりそのどちらでもあるという不思議な状態を持ちます。

これは Qbit の重要な性質の一つ「**重ね合わせ**」です。この性質から、Qbit は

$$|\psi\rangle = c_1 |0\rangle + c_2 |1\rangle \tag{1}$$

と表す（後述するが $|hoge\rangle$ はベクトルを表す便利な表現）とき、c_1, c_2 の条件が、

$$c_1^2 + c_2^2 = 1 \qquad c_1, c_2 \in \mathbb{C} \tag{2}$$

と規格化[*5] されます。これは Cbit が

$$「c_1 = 1 \text{ かつ } c_2 = 0」 \text{ または } 「c_1 = 0 \text{ かつ } c_2 = 1」 \tag{3}$$

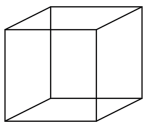
の二つの状態しか持ち得ないことに比べて、明らかに**つ よ そ う**ですよね！ ちなみに、c_1^2 と c_2^2 はそれぞれ $|0\rangle$ と $|1\rangle$ の生起確率になっています。Cbit はどちらかが 100%なのに、Qbit は条件を満たす限り無限に値を取ることができます。この扱う情報の違いが、古典的コンピュータとバトルする上で非常に強力な武器（ドーブ）となるのです。

2.2 それで、どうするの

重ね合わせの状態にある Qbit から解を得ることはできません。量子計算というのは、

1. あらゆる可能性を同時並行的に計算する
2. 目的とする結果に相当する状態だけが残るようなふるい分けを行う

という大きなくくりの 2 つの操作によって成り立ちます。量子の特性は計算にしか使えないので、解を得る段階では白黒はっきりさせます。観測すると、1 つの状態に定まってしまうという話はシュレ猫なんかで有名な話じゃないですかね。



他にも Qbit は Cbit にはない優れた性質を持っているので、以下に示します。

[*3] 一般的に使われる qubit という言葉にはマーミンが否定的な意見を述べていたので、本記事では Qbit と表す。

[*4] みんな大好きシュレ猫だよ〜((((o(*ï¿§ ▽ ï¿§*)o))))

[*5] 確率の総和が 1 となるようにする。

superposition

重ね合わせ: 並列計算に利用

interference

干渉: 解の効率的な探索に利用

entanglement

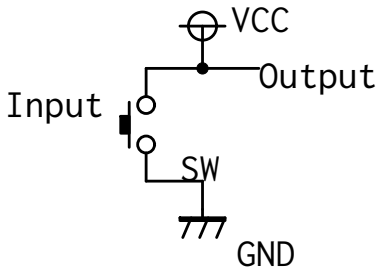
もつれ: 量子テレポーテーションや量子誤り訂正に利用

3　未知との対峙〜母なる数学

Qbit という新しい概念を知った時、あなたは恐怖を感じてしまうかもしれません。この感情を、この情報をどう処理していいかわからないのだから。

3.1　Cbit への操作

今までは簡単でした。皆さんが使っているコンピュータは 0 と 1 のデジタルな情報のみ扱えば良かったのですから。古典情報に対する操作は限られたものなので、頭の中だけで容易に把握できます。例えば、NOT ゲートのことを考えます。



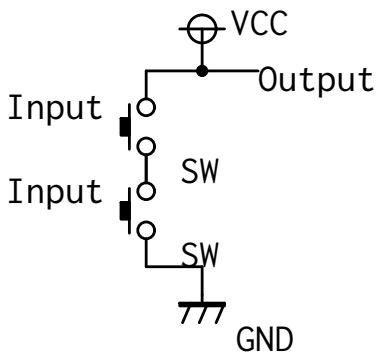
入力		出力
0	→	1
1	→	0

1 つのスイッチへ入力を行うと、Output に 1 か 0 か 1 つの結果として出力されます。

次に、2 入力 1 出力の操作は、

{00},　　{01},　　{10},　　{11}

を入力した時、出力がそれぞれ 0, 1 になるように対応するゲートを考えれば良いだけです。



入力		出力
00	→	1
01	→	1
11	→	0

これは 2 入力 1 出力の NAND ゲートの実装の 1 つです。入力の $\{01, 10\}$ に関しては交換則の成り立つ計算のみで現在コンピュータが実用に耐えていることから、$\{01\}$ を同じ入力とみなします。NAND ゲートがユニバーサルゲート[*6]であることは、各自適当な文献に当たっていただくとして、入力が 3 通りなので以下の表を満たすことを納得していただきたい。

入力	AND	XOR	OR	NOR	NOT XOR	NAND
00	0	0	0	1	1	1
01	0	1	1	0	0	1
11	1	0	1	0	1	0

順番をあえて崩すことで、3 パターンの入力をゲートに通した結果がそれぞれ 2 進 10 進変換を行ったところの 1 から 6 に対応していることを示しました。え？ 出力に $\{\{0, 0, 0\}, \{1, 1, 1\}\}$ が見当たらないって？ 入力に関係なく一定な値を取りたいなら地面にでもつないでおけばいいんじゃないですかね……（）

Cbit は 2 つの状態しか持ち得ないという性質から、とても単純な実装によって計算機に導くことができました。実際にこれで計算機が作れます。作ってみてください。

3.2 Qbit への操作

Qbit は $\{0, 1\}$ という離散値のみをとるものではありません。これでは実装を考えることが難しいだけでなく、操作を表すことすらママなりません。でも大丈夫です。数学（ママ）がいます。ママは全てを見ています。まずはママの子供になりましょう。先ほど出てきた $|\phi\rangle$ について、この記号は braket の ket[*7] と言って、ベクトルを雑に表すことのできる大変優れた記号なのですが、そのまま計算に使うには馴染みが薄すぎます。こんなものを被っていては大人になれません。ママに剥いてもらいましょう。あ、ママはもちろん数学のことです。

[*6] 全てのゲートの基本となるゲート

[*7] bra$\langle|$ と ket$|\rangle$ で braket

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \tag{4}$$

これで、ママ(数学)は Qbit を自分の子供として認識できるようになりました。まずは、Qbit の表し方を変更してみましょう。

$$|\psi\rangle = \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} \qquad c_1^2 + c_2^2 = 1 \qquad c_1, c_2 \in \mathbb{C} \tag{5}$$

1 つの Qbit に対して考えられる操作は

$$c_1 |0\rangle + c_2 |1\rangle \to c_1' |0\rangle + c_2' |1\rangle \tag{6}$$

これを数学的に表すと、

$$U \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} = \begin{pmatrix} c_1' \\ c_2' \end{pmatrix} \tag{7}$$

ここでの、U はユニタリ変換を行う操作です。Qbit はベクトルで表されているので操作は行列で表されます。ユニタリ変換は要するに数式 (2) の条件を破綻させない変換だと考えてください。Cbit への操作のように直感的に理解するのは難しいですが、1 つの Qbit に対する操作は全て記述することができました。

4 唐突な次回予告パート

2 つの Qbit に対する操作の解説も行いたかったのですが、LaTeX で数式を書くのに疲れてしまったので、次回実際に使いながら解説していこうと思います。さらに 1 つの Qbit に対する操作も、パウリ行列[*8]とかを持ち出して説明したいです。最終的にはショアのアルゴリズムの実装に関して解説できればいいな。

また、もしこれを読んでいる方の中に物理学類・応用理工学類、はたまた専門職の方がいらっしゃったら、是非アンケートにでも誤謬の指摘、知見の共有などしていただけるとありがたいです。

参考文献

[1] 慶応義塾大学 量子コンピュータ授業資料, 2009

http://www.appi.keio.ac.jp/Itoh_group/spintronics

[2] N. David Mermin (著), 木村 元 (翻訳). マーミン 量子コンピュータ科学の基礎, 2009

[3] 宮野 健次郎 (著), 古澤 明 (著). 量子コンピュータ入門 (第 2 版), 2016

[4] 中山 茂 (著). 量子アルゴリズム, 2014

[*8] コラっ！ 次回まで待ちなさい

WORD-SERVER 生前退位勅令

文　編集部　iorivur

1　ことの始まり

初めに言葉ありき

ヨハネによる福音書 1 章 1 節

polamjag: wiki**コンプラ**にレッツエンクリプト使いたくないですか（外からでも見えている Web ページなので設置は簡単）

ioriveur: （LGTM の gif-anime）

nymphium: :+1:

shinkbr: ていうか、wiki よりも word-ac.net の方がしたいんじゃない？ 外部向けなので

ioriveur: そっちもゆくゆくはやりたいと思うけど、wiki のほうが secret な感じなんで、その

図 1　Slack で火の手が上がる

この何気ない会話だが、重大な問題を孕んでいる――――

shinkbr: HTTPS 化したいという意志を持っているのが polamjag で、その権限（委譲も含めて）を持っている在学生が多分 @ioriveur と @rizaudo だ。なので、ファイッ

ioriveur: なるほど、そういうことか（俺持ってるか？ ）

ioriveur: wiki ってどこのサーバにあるの（**コンプラ**?）

shinkbr: **コンプラ**の中の lxc

ioriveur: www しか走ってなくない？

ioriveur: ```RUNNING

ioriveur: lxc-www (auto)

ioriveur: （省略）

ioriveur: ```

ioriveur: とのこと

shinkbr: www の中で動いてないっけ

図 2　不穏な空気が流れる

この時私には、そんなわけねーだろという気持ちと、 WORD のサーバならありそうという気持ちがないまぜになっている。WORD はしょせん学生組織、いくら縦横のつながりが強かろうと数年以上前のサーバ

を建てた人間はもういない——

今の会話で登場した、**コンプラ**という hostname の Git サーバにアクセスする。この Git サーバでは、この記事のような記事のソースコードや WORD 内で使われているソフトウェアのリポジトリが置いてあり、LaTeX のインテグレーションテストのための hook や trigger などを行っている [*1] 。このサーバの存在と gitolite を使ってリポジトリの管理をしているということは記事の発行の上で重要なことなので部員なら誰しも覚えている。しかしこのサーバは Ubuntu 12.04 LTS ともはや古すぎるということは知られていない。

lxc-list コマンドで**コンプラ**の上で動くコンテナに何が入っているか確認すると、 lxc-www という名前のコンテナが一つだけ動いていて、さらに動いていないたくさんのコンテナがあることが分かった。最悪に面倒くさい予感がする。

ここで一計を案じる。 polamjag に権限を与えてぐっすり眠ろう。ヨッシャ。

```
% sudo useradd -m polamjag -G wheel
useradd: group 'wheel' does not exist
```

ンア！？

[*1] 大人の都合で、**コンプラ**とは異なるコンピュータで Jenkins を動かしている

2　このサーバ、いったい何歳だ？

> 新しいぶどう酒を古い革袋に入れる者はいない。そんなことをすれば、革袋は破れ、ぶどう酒は流れ出て、革袋もだめになる。新しいぶどう酒は、新しい革袋に入れるものだ。そうすれば、両方とも長もちする
>
> マタイ福音書 9 章 17 節

wheel なむ group がない……のか。 Ubuntu だと sudo だけとかか？ そこで落ち着いて

```
% cat /etc/group
root:x:0:
daemon:x:1:
bin:x:2:
sys:x:3:
adm:x:4:[コンプラ],[コンプラ],[コンプラ],[コンプラ],[コンプラ],syslog
tty:x:5:
disk:x:6:
lp:x:7:
mail:x:8:
(snip)
```

ヌア、 Ubuntu で wheel が adm とかいうグループだったのっていったい何億年前だよ。もはや知りようがないことだが、このサーバが最初に設定されたのは相当古い可能性がある。osyoyu によれば 8.04 の頃までだったという記憶があるということだが……

まあ、古くから引き継いでいると言っても、ただ sshd 走らせて Git[*2] の push を Gitolite で hook しているだけなら、誰も何もこのサーバの設定について知らなくてもいじれるはずだ。今使っている OS のバージョンは Ubuntu 12.04 で古いといっても、LTS のおかげでサポートされているわけだし。さて、さっさと polamjag くんに全権委任してあったか布団でぐっすりしよう……

```
% sudo useradd -m polamjag -G adm^C
```

[*2] そういう VCS がある。特定のウェブサービスのほうではない

おおっと、 **コンプラ**じゃなくて、この上で動いている `www.word-ac.net` をホストしているコンテナを渡してもいいか。じゃあその前に本番環境をコピっておくか。

```
% sudo lxc-list
lxc-www lxc-www-clone lxc-www-riz-clone ...
% lxc-clone --name lxc-www lxc-www-ssl-test
```

……終わらねえ。舐めてんのか。`htop` 。無ぇ。 `apt install htop` 。 apt が無ぇ。 12.04 の頃はそういえばまだ apt-get vs. aptitude 戦争が終結してなかったな。この頃はどっちが結局推奨だったんだっけ。まあいいや。 `apt-get install htop`。さて、`lxc-clone` の `cp` が遅すぎて無限に練りをしまつ状態じゃねえか。ディスク大丈夫かな。へぇ、このマシン、 sda と sdb を束ねて RAID1[*3] にして md0 を作って、それをマウントしているのか。さすが **コンプラ**さん、 **コンプラ**でインフラやってるだけありますね [*4] 。`smartctl`[*5] を眺める。問題はないようだ……。しかしピークで 12MB/s って、ソ連の役所仕事じゃないんだぞ。[*6]

コツ、コツ。コツ、コツ。時計が `lxc-clone` 開始から 30 分たったことを示している……。暇だし lxc-www で動いてる Apache の設定でも読もう。`vim /etc/apache2/` して適当なファイルを見る。開く瞳孔。sites-available/wiki のダイジェスト認証にタダ乗りして munin が設定されてるぞ。ブラウザにバシッっと打つ。 `http://`**コンプラ**`/munin` 。あ、見える。ウケる。Slack に貼ってやろう。即座に koba789 が disk usage が 100% になってるのを目ざとく見つける。ハァ？　そんなにみんな記事書いてリポジトリ増やしてノウハウを wiki[*7] に書いてパンパンになってるのか？ みんな文字だけで HDD 埋めるとかさすが編集部員や——んなわけねーだろ。 smartctl をしたとき気づかなかったが、この md0, 80GB しか無ぇ。 SD カードだって今時もっとデカいの売ってるぞ。

ディスクが石版なみの容量しかないということは、 CPU も旧石器時代の遺物だという可能性がある。福音書にもそう書いてある。

```
% cat /proc/cpuinfo
processor       : 0
```

[*3] ディスクを束ねてミラーリングにすることでディスクが突如ぶっ壊れてデータが宇宙のどこででも存在したくないですさようならみたいにならなくする方式。

[*4] 因果関係が逆

[*5] S.M.A.R.T. という HDD のステータスを見るコマンド。ドッキリもあるよ

[*6] lxc-clone がイメージ全体をコピーするのは Ubuntu 12.04 のクソ古い lxc だからで、今では aufs でうまいことやってくれる気がする

[*7] そういうウェブベースのオーサリングツール/ドキュメントシステムがある。特定のウェブサービスのほうではない

```
3  vendor_id       : GenuineIntel
4  cpu family      : 6
5  model           : 22
6  model name      : Intel(R) Celeron(R) CPU          430  @ 1.80GHz
7  stepping        : 1
8  microcode       : 0x38
9  cpu MHz         : 1795.682
10 cache size      : 512 KB
11 (snip)
12 bogomips        : 3591.36
13 clflush size    : 64
14 cache_alignment : 64
15 address sizes   : 36 bits physical, 48 bits virtual
16 power management:
```

ファー、もう一度深呼吸、ブラウザを開き、duckduckgo[*8] で検索。Celeron 430。`ark.intel.com` をひらくまでもない。3 番目の候補に輝く Conroe-L Single-Core の文字。`ark.intel.com` によると Launch Date: Q2'07。当時、今の新入生はまだ 8, 9 歳かよ。まだママの **コンプラ**吸ってた頃じゃねえか。

[*8] 検索エンジン

3　吹き荒れるスターリニズムの嵐

私が小指をちょっと動かすだけで、チトーはこの世から消えてなくなるだろう

ヨシフ・スターリン

チューニングの要はまずベンチマークから

ヘネシー & パターソン

というわけで、プリパラ警察出動ーーーポワン。

全日本もうマシン移行したい協会に会費を払って、キーボードに向き直る。 apt のキャッシュがたまっているという珍説が Slack にかかれたが、 apt のキャッシュなんてゴキブリのウンチみたいなもので、それだけで HDD が埋まったなんてバカみたいな話は聞いたことが無い。

とにかくさっき作ったコンテナを消し、 さらに、止まってる全てのコンテナにある `/home/卒業生/tmp` `/var/www/www/...` がデカくてほぼ同一か、ふるい情報がつまっているだけのようなので、消す。消した。 disk usage が 79% になった。

さきほど発掘された munin を見ると、どうやらこの大量のデータは数年かけてゆっくり増えつづけて、少なくとも munin のログの残っているこの一年の月日のあいだには誰も容量を減らすことなく放置し続けたことがグラフから読み取れる。こういう拷問あったよな、部屋に閉じ込めて中にゆっくり注水するやつ。さて、コンスタントに増える理由を知りたい、というか一番容量を食ってるのは誰だ。

Listing 1　ディスク全体を舐めてデカいのが下になるようにソートする

```
% du / | sort -n
(snip)
3617240    /var/lib/lxc/lxc-www-test/rootfs/var/cache/bind
3619428    /var/lib/lxc/lxc-www/rootfs/var/cache/bind
3621964    /var/cache/bind
4177056    /var/lib/lxc/lxc-www-test/rootfs/var/cache
4179232    /var/lib/lxc/lxc-www/rootfs/var/cache
4533376    /home/rizaudo/lxc-www-riz-clone/rootfs/var/log/apache2
4533376    /var/lib/lxc/lxc-www-riz-clone/rootfs/var/log/apache2
4558112    /home/rizaudo/lxc-www-riz-clone/rootfs/var/log
4558112    /var/lib/lxc/lxc-www-riz-clone/rootfs/var/log
4868236    /usr/bin/sevpnserver/packet_log/WORD
```

```
13 4868240    /usr/bin/sevpnserver/packet_log
```

んー、なんで同じマシンで BIND がふたつ走ってるんだ……？？ まあ考えるのは置いておいて、ログを消そう。目についたデカいやつから消す。あれ？ /home/**コンプラ**/lxc-www-rizcloneって何だ、祖国の危機に /var/lib/lxc/ の中のものと同名のものを /home/ に抱え込んでるのは一体なんのつもりなんだ、ついでに消ッしてしまえ。[*9]

```
1 # rm /var/lib/lxc/lxc-www-{test, riz-clone, ...}/rootfs/var/cache/bind
2 # rm /var/lib/lxc/lxc-www-{test, riz-clone, ...}/rootfs/var/log/apache2
3 # rm /home/...
4         :
5         :
```

つうか待てコラ。 /usr/bin/sevpnserver/packet_log/WORDってなんだ、/usr/bin がデカいなんてデタラメ聞いたことあるかよ。なんで SoftEther VPN は /usr/bin/sevpnserver/ の中にログを 5GB も溜め込んでるんだ？ 一体誰が作ってるんだろう。まとめて殺す。普段使っていたサーバが実は古代遺跡だったことに驚きを隠せなかった私はつい気が大きくなっていたため次々とファイルを消していく。51% になった。

ioriveur: **51% になった**

ioriveur: **異常なグラフになった**

[*9] WORD では、個人的なファイルを入れておく (アンセキュアな) 場所があったりなかったりするので、外向きのサーバの個人ディレクトリに重大なファイルがあるとは予想していなかった

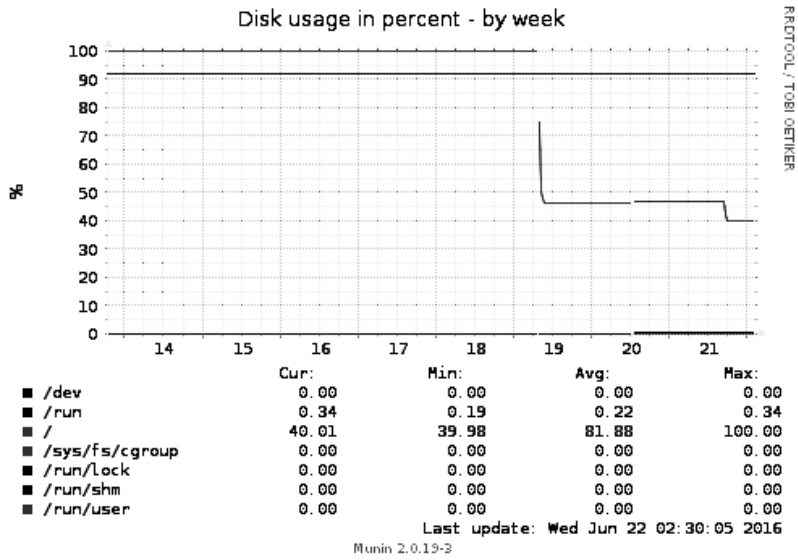


図 3　後になって保存したので若干違うグラフになっている。当時のグラフはもっと異常な形をしていた

4　1956 年 スターリン批判

このように「小指をちょっと動かした」ために、我々は高い代償を支払わなければなりませんでした

ニキータ・フルシチョフ

ioriveur: **本番のログは残してある**

koba789: **グラフがジェットコースターした**

ioriveur: **崖回**

ioriveur: sevpnserver **のログは飛ばした**

ioriveur: **まずかったかな**

ioriveur: **何も考えてなかった**

(30 **分の間**)

ioriveur: **ま、いっか！**

rizaudo: **まって、デカイからっていきなり粛清はやめてくれ。**

rizaudo: **あそこ確か** VM **やコンテナを整理した時に必要な部分を残していた筈で、削除されるとそういう遺物がなくなる。**

放棄されたコンテナと本番のログの区別がついていないパリピ状態の私に被害者がログイン即カチコミを入れてくる。そりゃそうだろう。

さて、 web 担当で管理を引き継いでいる rizaudo 君は disk usage は見てなかったのか余分なコンテナを撤去してくれてなかったが、上に挙げた名前がかぶっている謎のデカいコンテナは、過去にあったサーバのバックアップデータらしいということがその後の会話で分かった。

そのバックアップデータには WORD の Twitter bot[*10] を含むいろいろな過去の遺跡があったらしい。ではこの、今でも動いている謎の遺跡は一体何か、そしていつ作られ、何が起きている遺跡なのか、当時文字を持たなかった我々の祖先の精神世界を知ることは難しい。そして、なぜこの遺跡は、スナップショットがあるにもかかわらず今でも使われているのかも分からない。

半ばなぜ動いているのかすら分からなくなった古いサーバをカーゴカルト的に動かしつづけてきた我々、その状態をたくさんコピーしてディスクをパンパンに埋め、式年遷宮ごとに森を枯らしてきた神殿、ブチ切れて神殿の中に溜まった埃に着火していくパリピ ——————— 。

5 科学的探求の始まり

5.1 DNS の謎

さて、この du の結果をヒントに、一体このサーバで何が起きているのか調査する。まず、謎の BIND のキャッシュサーバが動いていることが分かる。しかし、この部屋の DHCP レスポンスをみると、 **コンプラ** でも lxc-www でもない別のサーバが帰ってくる。誰がこのキャッシュサーバを使っているのか。ログがある以上誰かが棲んでいるはずだ……。未知の生物への恐れが首をもたげる。しかしこれへのクエリは **コンプラ** と lxc-www しか飛ばしていないということが分かった。洞窟に徐々に光が差し込んでくる。

localhost にキャッシュサーバを立てるのは、例えば DHCP で振られる DNS が遠い場合、ミルキィホームズばりに頻繁に名前解決を行う、ブラウザなどのアプリケーションが遅くなるので、とりあえずごまかすという動機で行われがちだ。サーバにはいらない気がするが、なぜ……。というか一つのマシンに権威サーバが複数あるのも謎だが、キャッシュサーバが複数あるのはまったく意味がわからねえ。何が起きてるんだ。

5.2 munin の謎

まったく覚えてなかったが、 wiki に残っていた死海文書をヒントに古い記憶をまさぐった結果、 **コンプラ** さんが「ここに munin 大仏建てようー」みたいなこと言ってたのをほのかに思い出した。しかしいったい

[*10] かつて Twitter では BASIC 認証が使えた時代があり、その頃に作られた #wordhot タグを RT するという bot が運用されていた時期がある

具体的に誰がいつどこでどこに建てたのか、それとも建てなかったのかすら分かっていなかった我々はようやくインフラを他人に任せて惰眠を貪る危険を知ったのだ。

5.3 一次資料: 卒業生の証言

ここで古代人が里に降りてくる。

> yyu: コンプラはかつて、ふぁいさんのコンプラから出土した core i7 のマシンで動作してたんだけど、Git しか動いてないのに core i7 は必要なんですか？ という議論になり、ポンコツハードウェアに移行したという経緯だと思います。

5.4 考古学の成果

言語を絶するので次の図でのみ示す。

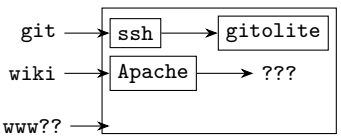


図 4 我々が想像していた コンプラ サーバ

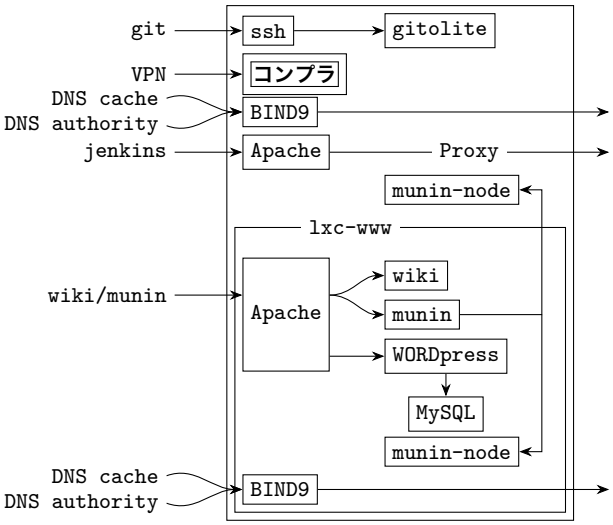


図 5 実際の コンプラ サーバ (図やセキュリティの都合上これでもはしょってある)

6　やっていき

> 規律ある民兵は自由な国家の安全保障にとって必要であるから、国民が武器を保持する権利は侵してはならない
>
> ——合衆国憲法修正第二条

やっていきが発生しているのでバシバシとしていく。

アジャイルを西海岸流、ウォーターフォールを東海岸流とするなら、WORD の開発スタイルは、テキサス流カウボーイ開発だ。思いついた人間がやっていきに応じて適当にバシバシしたり直したりとにかくやっていくというもの。修正第二条を護れ。銃を掲げよ。#2ndAmendment #2A

6.1　監視

すでに我々は munin という古代アーティファクトに光を取り戻させた。しかし、 munin は監視ツールとはいえ、リソース監視がメインで、死活監視、つまり激ヤバシチュエーションが発生した時に素早くみんなのやっていきを自動で高めるというニンジツは持っていない。もちろんリソース監視と死活監視の違いは小さいので、 munin をシバいてやっていき注入装置に転換することはできるが、どちらかというとそれはアビューズなので、できれば専用の Zabbix などを利用したい、という気持ちが盛り上がってくる。

しかし、 Zabbix を使うなら Slack インテグレーションをしたいなどの各位の様々なエモーションが集まった結果、面倒になってしまった。そこで Mackerel というサービスを紹介するはてなの回し者が現れた。外部サービスに頼ると、例えば以下のようなシナリオで役に立つ。

> polamjag: **マカレルの場合は当然メトリクスを外部に送信するので、ネットワークが死ぬと自動的に全ホストが死んだことになって通知が来て便利かも**
>
> ioriveur: **それまっさきに思いついたけど便利か便利じゃないか微妙なラインだよな**
>
> **ネットワークが死んだことは部屋にいる編集部員には自明過ぎるし、そうしたすぐ対応できる人間こそインターネットにつながってないという**
>
> osyoyu: **まあ、ネットワークの死は人間が真っ先に気づきそう……**
>
> ioriveur: **ネットワークが死ぬ→編集部員が気づいて直そうとするも失敗→いつの間にか治る→インターネットに繋がる→安堵してメーラーを開く→死んでる通知が来てるのに気づく→そうだね**

このシナリオから分かるとおり、「そうだね」というエモーションを大事にしていきたい人たちに有用なのだ。などというバカ話をしている間にわちゃわちゃしていたらあっという間に **コンプラ** がマカレルに巻かれていた。

ioriveur: added an integration to this channel: incoming-webhook

Mackerel BOT: Sample Notification from Mackerel

osyoyu: **ここで コンプラを一旦落としてみると**

ioriveur: **こういうとき猫というツールを使うと良くて**

osyoyu: **まず 3 学猫を拾ってくる**

ioriveur: **かまぼこ**

猫をサーバにいれると、猫という魚を食べる物体が鯖に入っておもしろいというほか、古代 Twitter のエラーページみたいになってエモくなるのでおすすめだ。しかし**コンプラ**塾が 3 学猫をすべて三味線にしてしまったというときは chaosmonkey を入れると良い。まあ人間と猿は大体同じなのでここでは手動 chaos 人間をする。

```
% yes yas
yas
yas
yas
:
:
```

こうするといつの間にか Slack に CRITICAL: **コンプラ**CPU % 99.0% > 80.0% LINK @channel というカチコミが飛んできて、寝てる人を叩き起こしたり、別の組織で真面目に Mackerel を使ってる人の心臓を systemctl stop させたりすることが出来る。

Mackerel のグラフを眺めていると 5 分に一度負荷が掛かっていることが分かったので謎が深まったが、頭をひねってみると実はこれは munin でグラフを生成する間隔だ。 munin が重いって本末転倒ではと思ったが、 **コンプラ**が雑魚いのが本当のところだろう。その他にも、 VPN をつないで作業をする人が現れたりすると途端に異常な Load Average を叩き出すようになる。

一日しかログがとれないと真綿でじわじわされるタイプのインシデントに気づきづらすぎるので munin を併用していく気持ちがある。

6.2 packet_log の正体

さて、どうしていやしくも /usr/bin/ の御中にログが吐かれているのか、そしてそのログは何で、どうしてこのサーバで一番デカいファイルになっているのか。それを知るため、とりあえず wc -l /usr/bin/sevpnserver/packet する。 wc が一向に終わらない。 CPU usage が 81 % になり、 Mackerel からメールと Slack 経由で警告が飛んでくる。ようやく計算が終わる。 16145708 行のログ。中を見ると、こうなっている……。

```
1 2016-06-22,00:13:30.938,コンプラ,The VPN Server is either Open-Source or
      Free version. It hasn't implemented the IP Address or TCP/UDP header
      data logging function. No IP address nor TCP/UDP header data are not
      be recorded here.,-,-,-,-,-,-,-,-,-
2 2016-06-22,00:13:30.948,コンプラ,The VPN Server is either Open-Source or
      Free version. It hasn't implemented the IP Address or TCP/UDP header
      data logging function. No IP address nor TCP/UDP header data are not
      be recorded here.,-,-,-,-,-,-,-,-,-
3 2016-06-22,00:17:06.371,コンプラ,The VPN Server is either Open-Source or
      Free version. It hasn't implemented the IP Address or TCP/UDP header
      data logging function. No IP address nor TCP/UDP header data are not
      be recorded here.,-,-,-,-,-,-,-,-,-
```

すべての行がこれだ。これが数年分に渡ってたまっている。焼いた。

7 4 年の時を駆け抜ける

Ubuntu 12.04 というのは今の一年生たちがまだ中学生で、口調が西尾維新っぽくなったり奈須きのこっぽくなっていた時代だ。あらゆることが古いし遅いし最悪なので、 do-release-upgrade コマンドを叩く。ところで古代ローマ時代には、システムのアップグレードは一族の命運をかけた一大イベントだった。そのために /etc/apt/sources.list の deb-src を手動で書き換え、 apt-get update \&\& apt-get upgrade \&\& apt-get dist-upgrade のような魔法を使ったものだ。この時代、この魔法によって何が起こるかは人間には予測不可能で神聖なものとされ、これによって一族が財をなしたり破滅したりしても仕方ないものとされた。また、ある時には apt-get install apt を適宜挿まなければ apt のデータベースが壊れることもあった。そのような状況を舞台にした、 apt-get の perl スクリプトにあるデータベースのバリデーションを手作業で破壊することでどうにか apt-get install apt だけ完走させて apt のバージョ

ンを上げ、ようやく現世の糧に繋いでリベンジを果たすという悲劇も好んで上演された。

時代は下って西暦 2012 年には `do-release-upgrade` というコマンドが使われるようになり、手動で変更されたファイルとアップグレードによって置き換わるファイルが衝突した場合に 3-way merge が出来るという機能が実験的に搭載された。ところで当時のこの機能はぶっちゃけあまり賢くないので設定ファイルのマージは手でやった方がいい……。

この作業をリモートでおこなっていたのだが、再起動をかけると意に反して全然上がって来ないため、アップグレードで致命的な失敗をしたと思い Slack に詫びを入れる。ブー垂れる shinkbr 。再度謝る自分。すると Mackerel の死活監視が **コンプラ**の社長出勤を知らせてくる……。場が和み、人々は安堵した。

7.1 nginx

こうしたアップグレードのあと、 `http://`**コンプラ**`.`**コンプラ**が Welcome to nginx! になっているというタレコミがあった。

え、Welcome to nginx じゃねえんじゃ、こちとら welcome してねえぞと思いつつ原因を探る。`service apache2 start` すると Apache のアップグレードによって設定ファイルの読み込みでコケていることが分かった。

これまで、 Apache が先に立ち上がってポートを塞いでいたせいで “Welcome to nginx!” できなかった nginx 。 Apache が設定ファイルの読み込みで死んで立ち上がらなくなった今、nginx は念願の “Welcome to nginx!” を送出する！……って、なんで nginx が上がろうとしてるんだ。よくよく思い出してみると shinkbr が Jenkins 用のリバースプロキシに nginx を使おうとしたがよくわかんねえ！ と叫んで麻雀に走り、最終的に彼が日頃から慣れている Apache で設定したという数年前の記憶が蘇ってきた。ログを見ると、一切何も役に立っていないくせに 6000 行のログがたまっている。なぜ……。

Apache の設定をやってからふきのとうかセミの幼虫のようなかわいい nginx を処刑した。Apache を再起動すると正しくリバースプロキシが動き Jenkins おじさんの顔が拝めてほっこりした。

7.2 尊属殺人事件

今度は `http://www.word-ac.net` がつながらないというタレコミがくる。嘘ツケ、さっきまで動いてたぞ。しかし確かめると確かに落ちている。 lxc-www は動いているかな……？ は……? `lxc-list` がない。`lxc-ls` に名前変わってますね、はい。気を取り直して lxc-www の状態を見てみるとちゃんと動いているので、入って Apache のステータスを見る。なるほど、怪死している。 `service` コマンドから Apache を再起動するとちゃんと上がり、外からも見ることが出来るようになった。しかしまた **コンプラ**の Apache を再起動すると……死んでる。

そこで lxc-www の Apache のログをみると、 SIGTERM シグナルを受け取って穏便に自死していることが分かった。いろいろ確かめた結果、 **コンプラ**の (つまり lxc ホストの) Apache を `service` コマン

ドで再起動するときに `service` コマンド は Apache という名を持つプロセスというだけで lxc-www の Apache を殺していることが判明した。あのぉ……ふざけんなよこの**コンプラ**野郎！

ところで尊属殺人といえば reiserfs です。

7.3 16.04 へ

そんなこんなで 14.04 にアップグレードした。しかしカレンダーをみると、今は 2016 年の 6 月だ。16.04 がとっくにリリースされている。また LTS が一つリリースされた。行こう、ここもじき腐海に沈む。

ところで 16.04 では init が upstart から systemd に変わった。しかし 14.04 から 16.04 にアップグレードすると /etc/init.d を引き継いだり、デバイス名がバージョンをまたいで変わらないよう udev のルールが自動生成されてねじ込みつつ、 upstart-socket-bridge などを使って systemd を親に起動するという非常に複雑怪奇なクトゥルー的アーティファクトが出来上がってしまい、そうした結果 init が systemd になるにもかかわらず起動もやたら遅く……というかこれなんですか……。

gitolite

今度は**コンプラ**が立てられた当初の機能である git サーバとしての機能を失ったとのカチコミがきた。その理由は gitolite2 がサポートされなくなり、 Ubuntu Server が無理やり gitolite を gitolite3 へアップグレードしたが、互換性がないのに気合でバージョンを上げたために見事完全にぶっ壊れたためだった。gitolite の設定ファイルをねじ込むリポジトリ gitolite-admin を適宜レスキューしたいが、すでに破壊された後なので落とせない。 git clone ssh:// をバイパスする適当なユーザを作ってそこから最新の gitolite-admin リポジトリを捻り出し、その後全 bare リポジトリを救出して、あとは移行マニュアルの通りに移行する。

バージョンが上がり、コマンド体系が清潔になったうえリポジトリ一覧がサクッと見られるようになった。

daemon 復活の儀

さらに www,wiki の HTTP サービス、 VPN 、 Mackerel のクライアントも死んでいたが、 init 移行に失敗して起動直後に上がるという設定がデグレっただけだったので `systemctl enable mackerel` などですぐ解決するかに思えた。

実際ほとんどのサービスはそれで復活したが、 wiki だけは一向に上がらない。おかしいと思い Slack にその旨を書くと、みんな「いや、繋がってるぞ」という反応をする。 WORD 部屋の編集用 PC で確認すると、確かに見られる。ついで、自分の PC の Firefox で確認しても見られるし、 Chromium の incognito タブで見ても見られる。普段使っている Chromium の環境でだけ、 www.word-ac.net にリダイレクトされるようになった。あまりにも意味不明だが、多分手元が悪いのだろうと思う。まず考えられるのは DNS のキャッシュがイカれてるという状況だが、その公算はない。なぜならば、同じ環境でも Firefox では見られるし、同じブラウザでも incognito mode にすれば見られるからだ。そんなこんなでわちゃわちゃしていたが、結局編集部 OB の cookie を吹っ飛ばせというアドバイスで解決した。どんな風に腐ったら cookie

のせいで permanent にリダイレクトされるようになるんだ、 Chromium ヤバすぎるな。

8 牧場へ

ところで、この激古マシンで VPN を使っていると、とにかく遅い。ファイルを出したり入れたりするとあっという間に CPU usage が 99% になる。よくみるとそのうち 26% が system に食われている時間だ。状況から考えて、ファイルの送信だけで system が 26% 食っているということになる、ありえないだろそれ、所詮 GbE だぞ。あまりにも CPU が遅すぎるために GbE すら碌に扱えないサーバを VPN にしているという異常事態に気づいた以上、ただ気分が悪いだけでなく直す必要があるという思いを新たにした。

WORD にはこうした開発用・外向き用サーバの他にもいくつかサーバがあるが、そのうちもっともスペックが高いのが VM 牧場と呼ばれるサーバだ。ここではいろいろな実験用の VM を各自が自由に立ててよいことになっているが、実際にはリソースに余裕があるままゆとり運転をしている。そこで、**コンプラ**を牧場に放って鍛えてやろうという気持ちが高まる。実際何度か人々がやろうとして途中で飽きてやめたが、今回はやっていきが最高潮に達しているのでやっていく。

8.1 牧場の NIC

スペックが高いと言っても寄付などで回っているため所詮 CPU は i7 940 、メモリは 10 GB しかなく、それなりに古い。しかも NIC は VMware ESXi でのサポートは 5 系を最後に切られてしまった Realtek 製の NIC なので、あまり良い環境とは言えなかった。

そんなある日、久しぶりに AC 部屋へいくと在学 k 年目となる**コンプラ**さんがいた。彼はゴミ袋を指して「みんな新入生が良い物を持っていったと思うけど、なんかあればあげるよ」と言う。ありがたくガサゴソと漁ってみると、 Intel PRO/1000 NIC がたくさん出てきたため、一年生もまだまだじゃのう……ほっほっほなどと言いながら頂戴してきた。そのことを koba789 に話すとまったく同じことを言って、やっぱ上級生って最悪だなと思った。

まあとにかく筐体をラックからひっぱりだして蟹 [*11] をポア、PRO/1000 を挿しておいたところ rizaudo くんが見事つられてひっかかり VMware ESXi を最新にした。 Web UI が生えてきた。いままで ESXi をイジるには Windows に謎の使いにくいクライアントを入れてワチャワチャしなければならなかったが、これで快適になった。

[*11] Realtek のチップには蟹にも見えるナスカの地上絵的なアーティファクトが描かれている

こうした経緯で新しい VMM 、まともな NIC が揃った。あとは **コンプラ** に専用のアカウントを作って ESXi 付属のツールに教えてやると、 VMware の機能で自動的に中のファイルやブート領域などが引っこ抜かれた挙句、ディスクのサイズのギャップなどにも適当に対応してそれらしいクローンが出来る。便利ー。

8.2 VPN 破壊

これで簡単に **コンプラ** のサービスが移行できた [*12] 。しかし VPN だけが壊れたという報告が上がる。VPN の設定ファイルを眺めると eth0 という文字列があったので ip コマンドで確認すると enp*msn* になっていた。これは 16.04 にしたために systemd 流のデバイス命名が行われたためだ。今まで問題が起きていなかった理由は do-release-upgrade コマンドが勝手に udev ルールを生成してねじ込んでいたからだが、いまやデバイスの MAC アドレスが変わっているのでそれから漏れてしまったという次第だ。

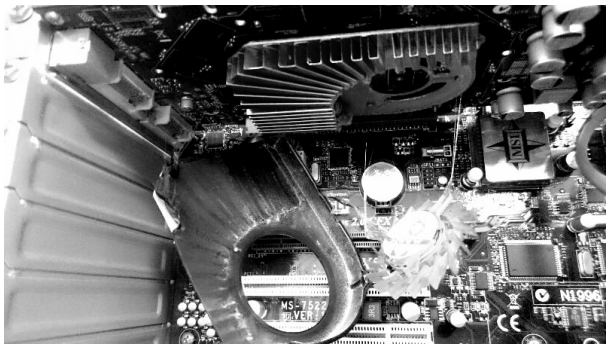
設定ファイルを vim で編集しても治らない。 VPN を再起動すると今度はファイルが巻き戻っている。なんだこれ。 SEVPN はどうやらコマンドラインツールから修正するようだ。タップに付いてるブリッジの名前を修正した。

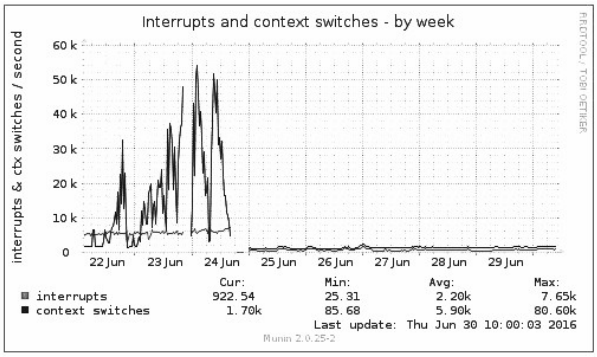
9 終わりに

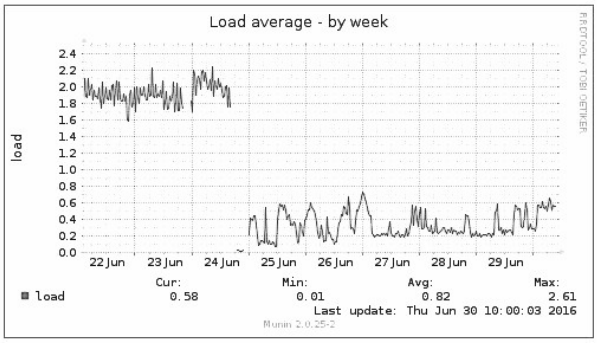
実はまだまだ問題が山積しているが、治ってから記事にしたいと思う。

ところで、これなーんだ:

[*12] グローバル IP の割り付けなどでトラブったがここでは割愛




Interrupts and context switches - by week
interrupts & ctx switches / second
60 k
50 k
40 k
30 k
20 k
10 k
0
22 Jun
23 Jun
24 Jun
25 Jun
26 Jun
27 Jun
28 Jun
29 Jun
Cur:
Min:
Avg:
Max:
interrupts
922.54
25.31
2.20k
7.65k
context switches
1.70k
85.68
5.90k
80.60k
Last update: Thu Jun 30 10:00:03 2016
Munin 2.0.25-2
PROTOOL, TOBI OETIKER


Load average - by week
load
2.4
2.2
2.0
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
22 Jun
23 Jun
24 Jun
25 Jun
26 Jun
27 Jun
28 Jun
29 Jun
Cur:
Min:
Avg:
Max:
load
0.58
0.01
0.82
2.61
Last update: Thu Jun 30 10:00:03 2016
Munin 2.0.25-2
PROTOOL, TOBI OETIKER

ブーブーを諦めてチリンチリンを買った話

文　編集部　natto

1　とある日曜日

夜に自転車を漕ぐのが好きだ。反射板は僕のためだけに光るし信号は来もしない車から僕を守ろうとする。街灯がないため自分のライトだけがたよりだ。ただ不安感はない、ライトが常に自分の場所を示し、僕の周りだけを幻想的に浮びあがらせる。調子のいいときは真ん中を走ってみる、この感覚は言葉で表せないだろう。

いつもは声に出せないことも大声で言ってみよう、どうせ聞いているのは虫と森だけだ。

ただ不幸なことに人間は想定外を想定できないのである。

良く聞くことは時間が止まるとかスローモーションになるとかだが僕にとってはその時間は無だった。ほんの少しの異変からなるがままに体が宙に浮く。つい先程までと違う、なんでうずくまってるのか、単純なことだったけど状況を飲み込むのに時間がかかった。まずわかったことは生きていること、そして頭天に真紅の花が咲いていることだった。体を軋ませながら起し「あぁ人生初か」と思いながら携帯を取り出す。ほんの数回操作するだけのことだったがそれは命と同じくらい重かった。周りを確認して事務的に相手に伝えようとする、上手く伝えられないのは衝撃のせいだろうか、僕の日本語力が稚拙だからだろうか。

否、もしものことを考えていたからであろう。

電話を終え、そんな気持ちを抑えるように動きだす、しかしこれが正しいかどうかという不安感は残った。

200m 程歩き地面に座り込んだ。「生きてるんだなぁ」と安堵した。

また赤いものが見えた、が自分から離れてしまう。「おいまってそっちじゃないんや」また電話をして bad な communication を繰り返した。僕の所へ到着して初めて感じたことは日本の社会制度のすばらしさだった、税金万歳。赤白の車だけを呼んだはずが白黒の車まで来て少々ひやりとしたが自転車を預かってもらった。

この頃には花も萎みきってどうってことなかった。

「自転車で 人生初の 119」

隊員とたわいもない話をしながら信号無視のなんとも言えない快感を味わった。

2　おわりに

初めまして、通常号は初めての natto です。いや本当に災難でしたよ。頭は 6 針縫うし未だに腰は痛い。高額な医療費（当然だが）と自転車がどうなっているかと考えるだけで胸が苦しくなる。頑張って車買ったほうがよかったと思える。そんな傷ついた心と体を癒やしてくれたのは「のぼる小寺さん」彼女はすごい。生きてるって教えてくれる。

結婚してくれ。

3　おわりたかったけどもうすこし

前節で述べた小寺さんについてもうすこし。簡単に説明すると、小寺さんはボルタリングを愛する高校生。そのひたむきな姿と不思議なところが可愛い。

うん、聞くだけでいい子だとわかる。僕は彼女のひたむきさから表れる自信が好きなのだ。彼女は特別な人間関係を築かない、それは一人でも生きて行くだけの精神力、ボルダリングへの愛故であろう。だからと言って周りに壁を作っているわけではない、ただ必要以上に人に頼らないだけだ。その姿は他人の心を動かす。それをよく思わない人間もいるだろう、きどっているだとか、かしこぶってるだとか。ただ周りを気にして自信のないことを言う子よりは真に生きているだろう。

僕は彼女の生き方に人間関係のイディアを見いだした。両手、両足のみを使い体を支える、流れるように重心移動し登っていく。ボルダリングにそっくりでないだろうか。

4　ほんとうにおわり

タイトルにもある通りいま筆者も含め WORD 編集部員はさらなる学類誌 WORD の発展のため車を購入しようとしています。廃車予定の車がありましたら一度 WORD 編集部にお立ち寄り下さい。今なら無料見積りで、もれなくメモリ型定規と WORD ステッカーをプレゼントします。

情報科学類誌

WORD

From College of Information Science

WORDの編集長も生前退位します号

発行者　情報科学類長

編集長　井上 陽介

制作・編集　筑波大学情報学群
情報科学類 WORD 編集部
（第三エリアC棟212号室）

2016年10月12日　初版第1刷発行（386部）