うぶんちゅ！

まがじん ざっぱ～ん♪

vol.6

体験版

うぶんちゅ！　まがじん ざっぱ〜ん♪ Vol.6【体験版】

team zpn　著

2017-04-09 版　　team zpn 発行

目次

第 1 章　OpenNebula の Private MarketPlaceApp

おおたあきひこ

ココミネ家は OpenNebula で家庭内クラウドを構築し、ココナとパピカの二人で計算機リソースをやりくりしています[*1]。

おや？ パピカが何やら困っています。

「どうしたのパピカ」

「MarketPlace がおそーい、App のダウンロードがいつまで待っても終わらないー」

どうやら新しく構築した OpenNebula 環境に MarketPlace から Ubuntu 16.04 の App をダウンロードしようとして時間がかかっているようです。デフォルトの Public MarketPlace は混雑していることが多く、約 350MB の Ubuntu 16.04 qcow2 イメージのダウンロードに数時間かかる場合もあり、ココナもパピカも大弱りです。

1.1 前置き

「あの」
はい何でしょうココナさん。
「ざっぱ〜ん Vol.5 までとキャラ設定が変わっている気がするんですけど」
ピュアイリュージョンではよくあることです。気にせずに進めましょう。
「えー」

- OpenNebula 環境での、OpenNebula サーバープロセスが稼働するマシンを「フロントエンド」と呼称します。
- Private MarketPlace を構築するマシンを「Private MarketPlace マシン」と呼称します。（そのまんまです）
- フロントエンド、Private MarketPlace マシンはいずれも Ubuntu 16.04 LTS server で構築しています。
- OpenNebula は OpenNebula.org 公式リポジトリのバージョン 5.2.1 を使用します。
- OpenNebula が一通りセットアップされ、各種リソースが登録されているものとします。

1.2 OpenNebula の MarketPlace

大抵の商用クラウドサービスでは、そのサービスで利用可能な各種 OS のディスクイメージが事前に用意されていて、利用者が好きなときに用途にあったディスクイメージを選んで仮想マシンインスタンスを作成できるようになっています。

OpenNebula では、これを実現する仕組みとして、仮想マシンディスクイメージの共有や配布を行う「MarketPlace」が用意されています。OpenNebula のリソース情報にはデフォルトで OpenNebula Systems の運営する「Public MarketPlace[*2]」が登録されており、フロントエンドを構築した直後から CLI や Web インタフェースの Sunstone でアクセスできます。

Public MarketPlace では、Ubuntu、Debian、CentOS、Arch などの各種ディストリビューションや、VyOS、Alpine Linux ベースのルーターなどのディスクイメージファイルが公開され、自由にダウンロードして利用できます[*3]。これらは MarketPlaceApp、または単に App[*4] と呼ばれます。

[*1] おばあちゃんは修理中です。

[*2] Public MarketPlace はインターネット上で公開されており、`http://marketplace.opennebula.systems/appliance` からアクセスできます。

[*3] `one-context` パッケージや `cloud-utils` パッケージなどがインストールされ、OpenNebula 環境で利用する準備が整った状態で公開されています。

[*4] Application の App だと思っていたのですが、どうも Appliance の App のようです。しまった vol.6 のテーマから外

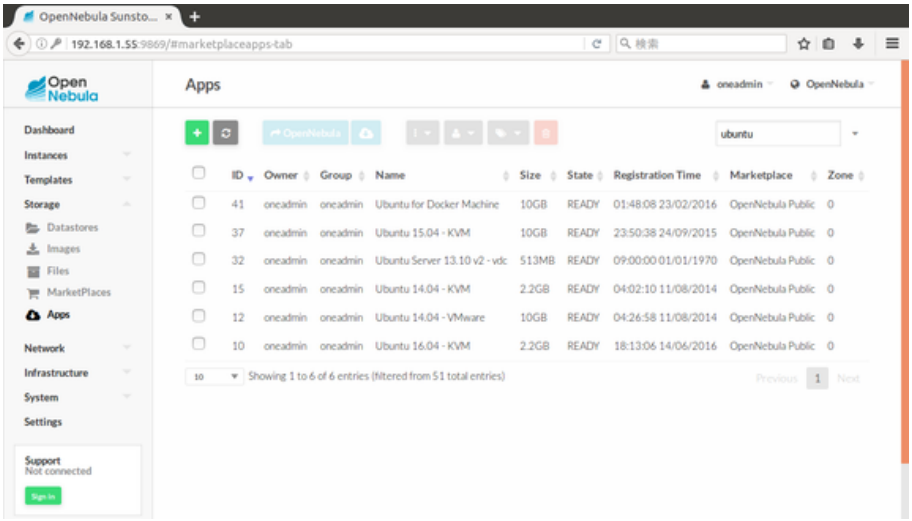

図 1.1: Public MarketPlace で Ubuntu の App を検索したところ

OpenNebula の管理者や一般ユーザーは、Public MarketPlace から OpenNebula 環境に App をダウンロードし、自分の仮想マシンインスタンスを作成できます[*5]。

1.2.1 App ってなに？

「qcow2 形式や raw 形式、vmdk 形式のディスクイメージファイル」と、「仮想マシンの構成情報」をまとめたものを、OpenNebula では App と呼んでいます。App を OpenNebula 環境にダウンロードすると、ディスクイメージは Image リソースとして、構成情報は Template リソース[*6]として登録されます。

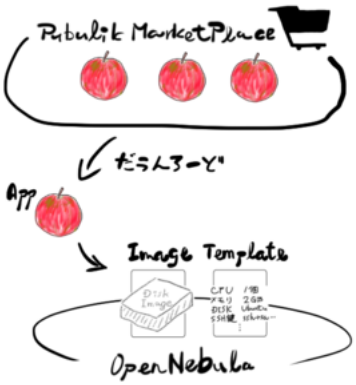

図 1.2: Public MarketPlace から App をダウンロード

れてしまった……

[*5] GitHub の Public Marketplace metadata リポジトリ（https://github.com/OpenNebula/marketplace）に pull request して自作の App を投稿することもできるようです。

[*6] OpenNebula では仮想マシンを構成するリソース情報を Template リソースにまとめて管理します。Template には、仮想マシンの起動元となるディスクイメージ（Image リソース）、仮想マシンが繋がるネットワークや IP アドレスレンジ（Virtual Network リソース）、CPU コア数、メモリサイズ、SSH ログインに使用する公開鍵などを記載できます。

1.2.2 Public と Private

デフォルトで登録されている Public MarketPlace に加えて、個々の OpenNebula 環境で独自に MarketPlace を構築することも可能です。こちらは Private MarketPlace と呼ばれます。

Private MarketPlace では、管理者や一般ユーザーが Image リソースと仮想マシン構成情報を App として登録し、環境内の他の利用者に公開できます。

表 1.1: Public と Private の比較

種別	特徴
Public	OpenNebula Systems が運営・公開している。App のダウンロードのみ可能。
Private	必要に応じて構築し、ローカルで利用する。App のアップロード/ダウンロードどちらも可能。

デフォルトの Public MarketPlace は環境構築直後から使えて便利なのですが、冒頭でも触れたように混雑していることが多く、App のダウンロードに時間がかかりがちです。そこで今回は Private MarketPlace を構築し、App を登録して環境内で共有してみます。

1.3 Private MarketPlace の構築

Private MarketPlace はバックエンドに HTTP サーバーまたは AWS S3 互換ストレージを使用できます。今回は Private MarketPlace 専用マシンを用意し、そこに HTTP サーバーを構築することにします[*7]。

使用する HTTP サーバーは Apache か Nginx が推奨されていますが、その他の HTTP サーバーでも要件を満たせば何とかなると思います。

1.3.1 HTTP バックエンドの仕組み

Private MarketPlace はディスクイメージのダウンロードとアップロードを処理します。HTTP バックエンドの場合、Private MarketPlace マシンからのディスクイメージダウンロードは HTTP で、フロントエンドからのディスクイメージアップロードは `oneadmin` 権限での scp で行います。

表 1.2: HTTP バックエンドのディスクイメージ転送方法

転送方向	転送プロトコル
Private MarketPlace → フロントエンド	HTTP
フロントエンド → Private MarketPlace	ssh (scp)

1.3.2 Private MarketPlace マシンの設定

Private MarketPlace マシンは以下で構築が完了します。

1. HTTP サーバーの公開設定
2. `oneadmin` グループ/アカウントの作成
3. `oneadmin` アカウントでフロントエンドから scp するための SSH 公開鍵登録
4. HTTP サーバーの公開ディレクトリの権限設定

例えば `apt install nginx` で Nginx をインストールし、リスト 1.1 のような設定を書いて[*8]、

リスト 1.1: Nginx 設定

[*7] フロントエンドマシンに HTTP サーバーを立てて同居させることも可能です。

[*8] `/etc/nginx/sites-available/default` のコメントを省いて `server_name` を書いただけのものです。

第 2 章　いかにして翻訳をするか

柴田充也

FLOSS（Free/Libre and Open Source Software）コミュニティで求められている作業のひとつに「UI やドキュメントの翻訳」があります。世界に数多ある言語への翻訳は、それがたとえ主要な言語であっても、到底ひとりで成し遂げられるものではありません。ゆえに FLOSS の世界における翻訳は、もっともコミュニティによる協力が不可欠な作業領域です。本記事ではこの翻訳における心構えについて、「Code of Conduct」を交えながら紹介します。

2.1 ソフトウェアを翻訳するということ

ソフトウェアの特に UI の翻訳は、修正すべき箇所がわかりやすいこと、英語と日本語の知識があれば始められること、多くの FLOSS においてはウェブ UI から翻訳可能であることなどから、比較的手掛けやすい**「貢献」**のひとつとしてあげられています。

しかしながら「UI やドキュメントの翻訳」は行うためには、単なる外国語の知識だけでなく、翻訳するふたつの言語の関係性、UI やドキュメントにおける文脈、他のソフトウェア・定訳との一貫性、ソースコード上の文字列の扱いなどを考慮しなくてはなりません。簡単なようでいて実は広範囲にわたる知識が必要な難しい作業なのです。

そのため大抵のソフトウェアプロジェクトでは、翻訳にあたってのルールや初心者向けのガイドラインを作成しています。またルール・ガイドラインの遵守、翻訳品質の維持を目的として、翻訳者の作業に対して確認を行うレビューワーが存在することも一般的です。翻訳者は何度となくルールやガイドラインを読み直しつつ、レビューワーにも指導してもらいながら作業を行うことになるでしょう。

2.2 翻訳に向いている人・向いていない人

どんな人にも向き・不向きな作業は存在します。その人に向いていない作業を行うことは、その人にとっても他の作業者にとっても不幸な結果を招きかねません。特に趣味やボランティアとして参加する FLOSS コミュニティであれば、自分にとって向いている作業で貢献したいところです。

では、どのような人が翻訳の作業に向いているのでしょうか。コミュニティの性質によって変わり得ますが、一般的には次の項目に該当する人が向いていると言えるでしょう。

- すでに存在するルールにそのまま従うことができる
- 作業の成果に対する反応がなくても気にしない
- 英語と日本語の読み書きが辞書を使える程度には堪能である
- わからない部分は翻訳しない忍耐力を持っている
- 上記のどれかひとつでも「自分はそこまでできない」と思ってしまう

なぜこれらの項目が出てくるのか、それは次節以降で説明します。

さて、上記を踏まえて言い換えると、次のような人は他の人と協力するような翻訳作業には向いていないと言えるかもしれません。

- ルールは自分で作りたい・変えたい

- 作業の成果はできるだけスピーディーに反映したい
- 英語は滅ぶべきであると確信している
- 細かいことはいいからとにかく翻訳させろ

極端な話、ルールを積極的に変更したり、自身の成果を迅速にコミットしたいのであれば、既存のプロジェクト・コミュニティに参加するよりも、新規にプロジェクトを立ち上げるほうが手っ取り早いです。自分が主体のプロジェクトを作れば、ルールもコミットも思いのままです。幸い FLOSS であれば、ソフトウェアプロジェクトを分離する自由が保証されていますので、新規プロジェクトの立ち上げそのものは簡単に行えるでしょう。当たり前のことながら、その立ち上げたプロジェクトを軌道に乗せることは非常に大変なのですけれども。

もちろん翻訳以外にもたくさんの作業があります。翻訳に向いていなければそちらで貢献するのもひとつの手です。実装や不具合修正といったソースコードの改変に関わる作業はもちろんのこと、不具合の報告、誤字脱字の訂正、ドキュメントの作成、資金や場所・雇用の提供など、できることは多岐にわたります。

特に寄付という形での資金の提供は、お金さえあれば誰でもできる上に、そのお金自体は自分の得意な手段で稼げば良いのでおすすめです。コミュニティイベントなどで開発者に遭遇した際に、美味しい飲み物や食べ物をおごるという手もありますね。

なぜ数ある選択肢の中から翻訳という作業を選んだのか、それを明確に説明できるかどうかも向いている条件のひとつかもしれません。

2.3 Code of Conduct（行動規範）を活用する

コミュニティが円滑に機能するためには、一定のルールが必要です。人数が少ないうちは「常識」や「空気」と言った明文化されていない暗黙のルールのような何かだけでもうまくいくことでしょう。しかしながらある程度コミュニティの規模が大きくなってくると、メンバー間の衝突を公正に調停するためにも、ルールを明文化する必要がでてきます。これは結果としてコミュニティ運営の透明性を向上させ、新規にコミュニティに参加する人への心理的障壁を下げる効果もあるのです。

Ubuntu は非常に大きなコミュニティです。FLOSS の文化について詳しい開発者だけでなく、コンピューターそのものに不慣れな純粋な初心者もかなりの割合で存在します。さらに他の FLOSS コミュニティと同じように、世界中の異なる常識をもった人々が協力して作業を行う必要があります。よって明文化されたルールの必要性は明白でした。

そこで Ubuntu では**「Code of Conduct」**（CoC：行動規範）[*1]と呼ばれる、Ubuntu コミュニティの一員としての振る舞いについて述べた文書を作成し、メンバーにこの CoC に基づいた行動を心がけるように呼びかけました。

CoC の現在のバージョンは 2012 年に更新された 2.0 です[*2]。Ubuntu Japanese Team のサイトには 1.1 の日本語参考訳も存在します[*3]。内容は平易な英語を使うように心がけているため、できるかぎりより新しい原文を読むのがいいでしょう[*4]。

---

*1 https://www.ubuntu.com/about/about-ubuntu/conduct

*2 https://launchpad.net/codeofconduct/2.0

*3 https://www.ubuntulinux.jp/community/conduct

*4 2.0 では、1.1 に残っていた Ubuntu 固有の話や開発者という表現を極力取り除き CC-BY-SA 3.0 でライセンスしています。これにより他のコミュニティでもこの CoC をそのままあるいは改変して使えるようになりました。また、これまで別扱いだったコミュニティリーダー向けの CoC が統合され、重複している部分は削除されました。ちなみに 1.1 より前はもうちょっと軽い感じで「完璧だと思われている人もいません（SABDFL を除いては）」（SABDFL は Ubuntu 創始者

CoC に書かれている内容は、とても「当たり前」のことです。社会生活を行う上で、たいていの人が「常識的な振る舞い」として考えているであろう内容を、そのまま文章にしています。読む人によっては、正論や理想論をただ並べただけのつまらない文章に見えるかもしれません。でも、この「当たり前」をおぼえて実践するだけで、本当にたったそれだけで、コミュニティからは良識のある人として信頼を得ることができるのです。これを個人の行動の指針として活用しない手はありませんね。

CoC の内容については実際の条文を読んでもらうことにして、ここではその項目と概要のみを列挙しましょう。

他者を思いやれ（Be considerate）

コミュニティでの決定や成果物は、他のコミュニティでも使われる可能性があることを考えて行動しましょう。

他者を尊重せよ（Be respectful）

敬意をもった対応を心がけましょう、意見の不一致が攻撃的になってよい理由にはなりません。

発言や行動に責任を持て（Take responsibility for our words and our actions）

誰でも間違いは起こすもの。間違ったときにどう行動するかが重要です。

協力せよ（Be collaborative）

他のコミュニティとの協力は不可欠です。そのためには他のコミュニティの目的や作業を理解する努力が必要です。

決断・透明性・合意を尊重せよ（Value decisiveness, clarity and consensus）

意見が一致しないことは普通のことではあります。いかに建設的にその不一致を解決し、その決定を尊重するかが重要です。

自信がないときは助けを求めよ（Ask for help when unsure）

何でも知っている人なんていません。適切な場所で積極的に質問を行いましょう。

退任するときは慎重に（Step down considerately）

プロジェクトを離脱するときは、プロジェクトへの影響が少なくなるよう配慮しましょう。

先ほど述べた「翻訳に向いている人」のリストをひとつずつ説明し、CoC を元にした行動指針と照らし合わせることで、CoC で使われている用語の有用性を明らかにしていくことにしましょう。

2.3.1 すでに存在するルールにそのまま従うことができる

コミュニティに参加する場合、まずは既存のルールを理解して作業しましょう。これはすでに得られた合意を尊重することに他なりません（**決断・透明性・合意を尊重せよ**）。

あなたが優秀であれば、ルールに疑問を抱くことや、より良い方法を導き出すこともあるでしょう。ただし既存のルールには、少なくとも導入された当時には何らかの意味があったはずです。闇雲にルールの改定を提案するのではなく、まずは一通りの作業を行うことで、そのルールの必要性や経緯を理解することに努めましょう。

「他者を思いやれ」では、自身のコミュニティにおいて何らかの決定を行う際に他のコミュニティへの影響を考慮するよう述べています。これは言い換えると個人が何らかの決定を行う際に自身のコミュニティへの影響を考慮することも重要であるとも解釈できます。

ただしルールを変更するためのルールが決まっているのであれば、その限りではありません。もちろんルールの意図を理解できそうにないのであれば、質問するのも有効な手段です（**自信がないときは助**

のマーク・シャトルワースのこと）なんて冗談交じりの文言もあったぐらいです。

第 3 章　Ubuntu 音楽再生アプリ探訪

長南 浩（Ubuntu Japanese Team）

Ubuntu 16.10 が去る 2016 年 10 月 13 日 にリリースされました。いまのところ流動的なところもありますが、Ubuntu の次世代のデスクトップ環境（Unity 8）の足音が大きくなってきています。ここで、現在の Unity デスクトップ環境で使える音楽再生アプリをいくつか紹介してみます。

こう書くと「総まとめ」のようにも見えますが、2021 年 4 月までサポートされる 16.04 LTS を使っている方には音楽再生アプリを選ぶために役にたつのではないかと思います。

音楽再生アプリはコンピュータに向きあう日常の一部を豊かにするものということで、ゆるくおつきあいください。

3.1 大きな変化を遂げた音楽鑑賞環境

Ubuntu に限ったことではなくて恐縮ですが、読者の皆さんは音楽をどこから入手しているでしょうか？ ショップに出かけて CD を購入し、CD プレーヤで再生しているというケースはもはや少数派で iTunes ストアやその他の音楽配信サイトで購入したものを聴いたり、もしかしたら YouTube やニコニコ動画を BGM 代わりに垂れ流しているケースも多いと思います。

20 世紀の頃の昔、音楽はレコードやら CD を取り扱うショップで購入し、家でレコードプレーヤや CD デッキを使って再生するのが普通の楽しみ方でした。しかし、20 世紀も終わりになると不可逆ながらネットワークやメモリデバイスに載せられる mp3 音声データフォーマットが注目を浴び、場合によってはアンダーグラウンドで非合法に音楽データがやりとりされるようなこともありました。[*1]

PC 上で mp3 ファイルで音楽を聴くというのはアンダーグランドで背徳感のある行為だったし、CD-R で音楽 CD の複製を作るといったことも技術的に容易になった時代がしばらく続いたのですが、iTunes（2003 年 4 月サービス開始）の登場で、アングラっぽいものは Apple（そしてその他の音楽配信事業者）のビジネスになってしまいました。

今は音楽配信ビジネスは一般的なものになり、多くの人が CD ではなくデジタルファイルで音楽を楽しむ時代です。音楽 CD から音声データを抽出するというのも、アンダーグランド的な行為から iTunes の機能になってしまいました。[*2]

そんなデジタル音楽を Ubuntu 上で楽しむことができるアプリをいくつか紹介します。この他にもたくさんの音楽再生アプリが Ubuntu では使えるので、好みのものを探して活用してください。

3.2 Ubuntu 16.10 で使える音楽再生アプリ

3.2.1 Rhythmbox

現在の Ubuntu 標準の音楽プレーヤが Rhythmbox。標準音楽プレーヤとしての実績は大きく、11.04 と 11.10 で一時的に Banshee にその座を奪われたものの、Ubuntu で音楽を再生したことがあるユーザは一度は起動したことがあるアプリです。

文字中心のオペレーションで UI の見た目が少々味気ないですが、その分軽快に動作するのがありがたい、シンプルさが魅力のアプリです。

[*1] http://akiba-pc.watch.impress.co.jp/hotline/980501/mpman.html によると 1998 年に携帯 mp3 プレーヤが発売されたようです。この頃から「あきばおー」があったということも驚きですね...

[*2] 不正競争防止法に抵触しそうなところですが、一般的な音楽 CD は技術的保護手段を持っていないので抵触しないことになっています。著作権法の私的複製の範囲内で一般的な音楽 CD をリッピングするのは法的な問題はないものと思われます。

図 3.1: Rhythmbox の音楽再生画面

3.2.2 Banshee

11.04 と 11.10 で Ubuntu 標準に採用された音楽プレーヤです。文字主体の UI だった Rhythmbox に比べて、アルバムアートが表示できたり、外部サービスと連携できたりする高機能音楽プレーヤで、音楽だけではなく動画も再生できるメディアセンター的な位置づけを狙ったアプリです。

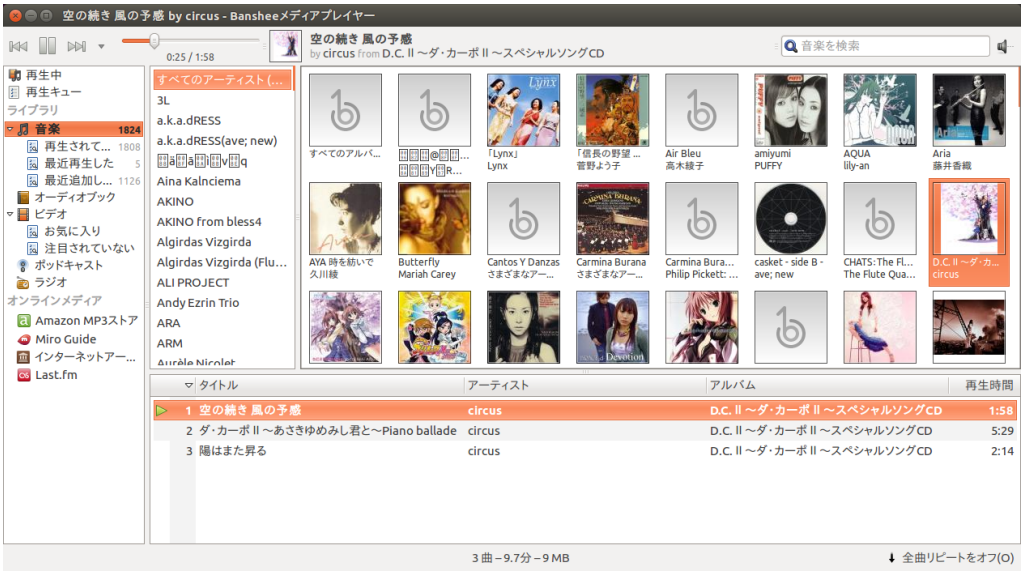

図 3.2: Banshee の音楽再生画面。iTunes のような画面構成が特徴

3.2.3 Amarok

KDE ベースの音楽プレーヤ。曲間のクロスフェードやポッドキャスティングのサポート、データベース連携など、多彩な機能をもつ音楽再生アプリです。KDE ベースなので Unity デスクトップと若干 GUI のテイストが異なり、インストールに必要なパッケージも若干多いところが評価を分けるかもしれません。

図 3.3: Amarok の音楽再生画面。スペースを広く使う UI は好みの別れどころかもしれない

3.2.4 Clementine

Amarok にインスパイヤされて作られたのが Clementine です。キューシートに対応していたり、マルチプラットホームで Windows や Mac でも利用できるのが特徴です。

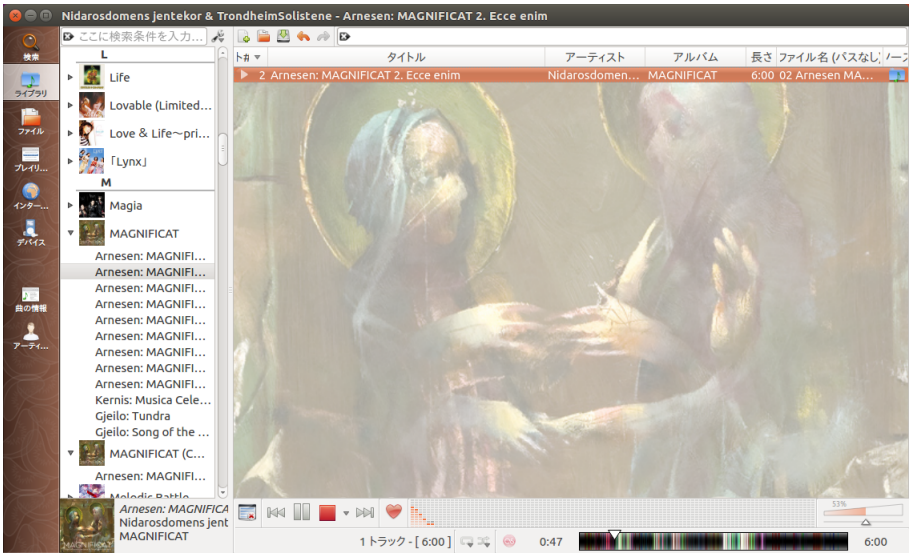

図 3.4: Clementine の音楽再生画面。今回紹介した中では最もモダンな UI のプレーヤ。細かいコダワリを感じさせる

第 4 章　2017 年の Ubuntu 録画環境

kazken3(@kazken3)

＊人は寂しくなるとアニメに手を出す。これはそんな男の物語である＊

4.1 これまでのおはなし

これまでのざっぱ〜ん上における録画関連、こんな感じでやってきました。

- Chinachu&Rygel を導入し人権を得る (vol.1)
- エンコーディングとか USB マウントとか (vol.2)
- Chatbot で録画させる (vol.3)

わりとやってきたんですねえ。もうやめてしまったこととともわりと多いですが…さて、今回は 2017 年初頭の録画環境は、こんな感じでやっています。

- Chinachu γ の導入
- Rygel ふたたび
- pushbullet で録画を通知

ディストリビューションは Ubuntu 16.10 Server を利用しましたが、Ubuntu 16.04 Server でも問題ないはずです。事前準備として以下のパッケージ群をインストールしておきましょう。

```
$ sudo apt install autoconf build-essential dkms git pkg-config ¥
curl libssl-dev yasm libtool libpcsclite-dev libpcsclite1 ¥
pcscd cmake linux-headers-`uname -r`
```

今回はドライバーや、ライブラリ、レコーダーの導入について本記事には掲載しませんが、参考までに gist[*1] に置いておきます。3 年前からの更新なのでビルドや導入に多少の変更はありますが、基本の流れは変わりません。

4.2 Chinachu γ の導入

2016 年の年末に、Chinachu Air における新機能の先行実装版の形で Chinachu γ[*2]がリリースされました。これまでも Chinachu Beta を利用してきていたのですが、今回はまっさらな状態から新たに Chinachu γ をインストールすることにしました[*3]。

4.2.1 Chinachu の構成

Chinachu Beta のころは、Chinachu 単体で録画クライアントとチューナー処理を一緒に行っていましたが、Mirakurun が登場し、以下の分担が行われるようになりました。

- Mirakurun:チューナーサーバー
- Chinachu γ:録画クライアント

録画の実行部分と、録画クライアントの部分を切り離した感じでしょうか。それではまず、Mirakurun をインストールしていきます。

[*1] https://gist.github.com/kazken3/4ede523d0add3193f34b691ceb73fc77
[*2] https://r2.ag/chinachu-gamma/
[*3] SSD 交換しました。ふふふ。

4.2.2 Mirakurun インストール

Node.js のインストールを行います。適当な作業ディレクトリを作成して、安定版をインストールします。

```
$ mkdir ~/work
$ cd ~/work
$ curl -sL https://deb.nodesource.com/setup_6.x | sudo -E bash -
$ sudo apt install -y nodejs
```

node -v でインストールできていることを確認できれば OK です。

```
$ node -v
v6.9.4
```

Node.js 上のアプリケーションをデーモン化する PM2 のインストールします。

```
$ sudo npm install pm2 -g
```

ここで Mirakurun のインストールします。このタイミングで PM2 にも登録されます。インストールが終わったらステータスを確認して online であればインストールは完了です。

```
$ sudo npm install mirakurun -g --unsafe --production
$ sudo mirakurun status
```

b25 のストリームデコーダーもインストールします[*4]。

```
$ sudo npm install arib-b25-stream-test -g --unsafe
```

チューナーの設定を行います。最初は isDisabled がすべて true です。必要なものだけ false にしましょう[*5]。

```
$ sudo mirakurun config tuners
```

あわせて、チャネルの設定も行います。デフォルト設定として東京の地上波や BS、CS が入っていますが、Wikipedia のページなど[*6]を参考に、設定を追加しましょう。

```
$ sudo mirakurun config channels
```

こんな感じで、設定してます。
設定の追加が終わったら、mirakurun を再起動します。

```
$ sudo mirakurun restart
```

次は Chinachu γ のインストールです。

[*4] ここで気づいたんですけど、ここでデコードするなら recpt1 上でのデコードっていらないっすよね…まあいずれにしても動作はします。
[*5] PT3 がチューナーの場合の設定は Chardev 版であれば有効にするだけでしょう。
[*6] https://ja.wikipedia.org/wiki/%E3%83%86%E3%83%AC%E3%83%93%E5%91%A8%E6%B3%A2%E6%95%B0%E3%83%81%E3%83%A3%E3%83%B3%E3%83%8D%E3%83%AB

4.2.3 Chinachu γ インストール

```
$ git clone git://github.com/kanreisa/Chinachu.git ~/chinachu
$ cd ~/chinachu/
$ ./chinachu installer
# Auto を選択
```

設定ファイルをサンプルファイルからコピーして編集します。

```
$ cp config.sample.json config.json
$ vim config.json
```

録画機能自体は Mirakurun に渡したので、以下のように設定ファイルはシンプルになっています。気をつけておくべきところは、wuiOpenHost がデフォルトでは省略されているので、追加してサーバの IP アドレスを記入しておいたほうがよいといったところでしょうか[*7]。その他では recordedFormat を少しいじったこと、recordedCommand を追加したことぐらいです[*8]。

```
[
{
  "mirakurunPath": "http+unix://%2Fvar%2Frun%2Fmirakurun.sock/",

  "recordedDir" : "./recorded/",

  "wuiUsers": [
  ],
  "wuiAllowCountries": ["JP"],

  "wuiPort": null,
  "wuiHost": "0.0.0.0",
  "wuiTlsKeyPath": null,
  "wuiTlsCertPath": null,
  "wuiTlsRequestCert": false,
  "wuiTlsRejectUnauthorized": true,
  "wuiTlsCaPath": null,
  "wuiOpenServer": true,
  "wuiOpenHost": "サーバーの IP アドレス",
  "wuiOpenPort": 20772,
  "wuiXFF": false,
  "wuiDLNAServerEnabled": false,
  "wuiMdnsAdvertisement": true,

  "recordedFormat": "[<date:yymmdd-HHMM>][<type><channel>][<channel-name>]<title>ts.mp4",
  "recordedCommand": "/home/hoge/sc.sh",
  "storageLowSpaceThresholdMB": 3000,
  "storageLowSpaceAction": "remove",
  "storageLowSpaceNotifyTo": null,
  "storageLowSpaceCommand": ""
}

]
```

空ルールファイルの追加を行います。

```
$ echo [] > rules.json
```

これで、大まかな導入は完了です。動作確認のため、以下のコマンドでエラーが出てないことを確認しましょう。

[*7] wuiPort は今後廃止予定で、wuiOpenHost も IP アドレスを自動検出するようですが、無難に動作させたい場合は wuiOpenHost に IP アドレスを入れたほうが良さそうです。

[*8] これは Pushbullet の項で説明します。Pushbullet 連携が不要な方は外しておいたほうが無難です。

第 5 章　Cyclograph で GPS を使わず自転車ライフログ

Rakugou

一部のサイクルコンピュータには搭載されていない自転車の走行ルートの履歴の作成について、Ubuntu のアプリケーションである Cyclograph を用いて作成や画像データの出力、データの修正までやってみようと思います。

5.1 サイクルコンピュータのあれやこれや

Ubuntu をお使いのスポーツ自転車好きの皆様、サイクルコンピュータはどのようなものをお使いでしょうか。最高時速、走行距離、積算走行距離を計測できるベーシックなものや、加えて消費カロリー、心拍数やペース配分をサポートしてくれたり、はたまたスマートフォンと連動してメールや着信の通知をしてくれたり様々な機能が搭載されているものがあります。値段も 2,000 円台から購入できるものもあれば、30,000 円を超えるモデルまで様々なものがあります。

さて、クロスバイクやロードバイクに代表されるスポーツ自転車及びそれに搭載されているパーツ群は盗難のリスクが多いことで知られています。自転車本体には鍵を複数個かけて駐輪したり、街路樹や手すり等にチェーンキーを括りつけてロックするなど[*1]、自転車本体の盗難に対する工夫は種々ありますが、サイクルコンピュータは鍵をかける部分がないため取り外しが容易で、かつ高価なものもあるため盗難のリスクは高いです。

もちろんサイクルコンピュータは自転車生活を便利にしてくれるものでありますが、出費可能な予算と、必要な機能と、盗難された時の心理的苦痛を勘案して決めることが重要でしょう。そのような中で、GPS 非搭載で走行履歴を記録できないサイクルコンピュータを使われている方で、どうしても自転車の走行履歴を後付けでも残しておきたいという場合、Cyclograph というアプリケーションが有効です。

5.2 地図を作成する

Cyclograph はソフトウェアセンターからダウンロードできます。末尾に Qt と Gtk になっているものがありますが、どちらかお好みのものをインストールしてください。ちなみに、今回は末尾が Qt のものを、Ubuntu のバージョンは 16.04 LTS を用いて作成しました。

「New」をクリックしてデータを新規作成します。その後、「Create」を選択すると、地図を作成する画面が出現します（図 5.1）。左側のメニューから、「Draw on the map」を選択します。初期設定ではヨーロッパが基準となっていますが、左上のプラスマイナスボタンで拡大、縮小、移動をしながら、日本にフォーカスを向けるようにしましょう。

ちなみに、他のメニューだと、出発地と目的地を入力し、その間をナビゲートしてもらう形式になりますが、自転車ログ的に活用する場合は、初めて行く道であれば迷ったりしてナビゲート通りには走っていないことも多々有るでしょうし、よく知っている道であればナビゲートで推奨されている道よりも、ショートカットできる路地や細道などを走行して時間短縮を図ることもあるでしょう。そのため、手動でルートを描いていくほうがより現実に則した地図になるでしょう。

ここから自転車走行の軌跡を記録していきましょう。今回の例では、京阪電鉄・大阪市営地下鉄御堂筋線淀屋橋駅から天保山まで自転車で向かうルートを想定してみましょう。まずは地図を淀屋橋駅に合わせます。すると図 5.2 のような感じになります。

[*1] 俗に「地球ロック」と呼ばれています。

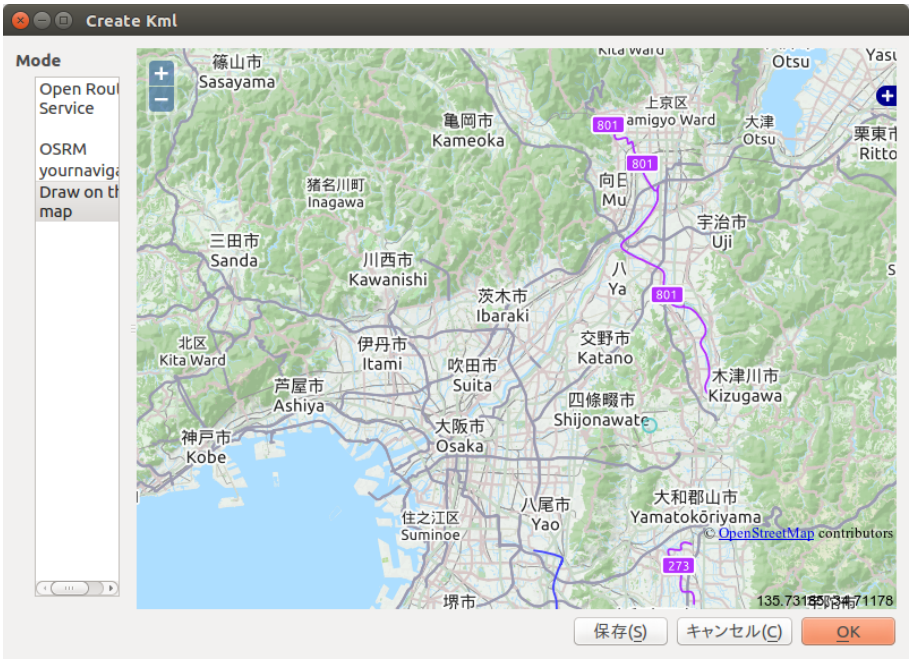

図 5.1: 地図選択画面、地図上に 801 と紫色に輝いているのが京都八幡木津自転車道路。273 となっているのが、大和郡山田原本橿原自転車道路になります。

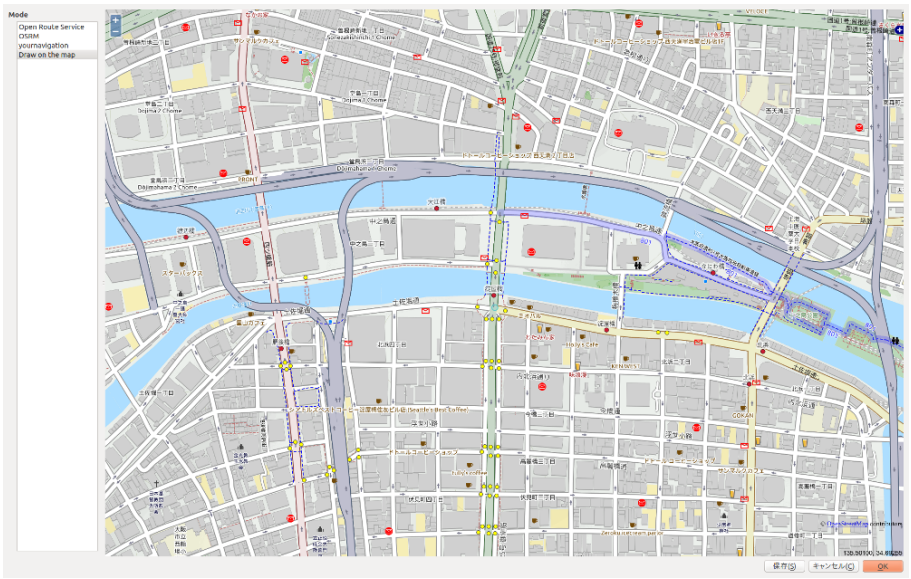

図 5.2: 淀屋橋駅が起点となる。なお、青くハイライトされている大阪吹田自転車道路は今回は利用しない。

まず、起点を設定するために、設定する場所をダブルクリックします。その後カーソルを動かすと、青い線が起点からカーソルについていくように現れるようになります。そして、クリックすると中継地点を設定できるようになります。それを目的地まで繋いでいきます。ゆるやかに曲がる場合は、細かく

中継地点を設定して進めていきましょう。図 5.3 のようになるはずです。

図 5.3: 肥後橋交差点付近まで中継地点を設定した様子。まだまだ先は長い。

なお、右クリックを長押しした状態でカーソルを移動させると、先の地図を出現させることができます。クリックを続けても目的地に到達すると、ダブルクリックして描画を終了させます。すると青い線がオレンジに変色します。

図 5.4: 目的地まで到達した結果。なお最後の右折部分を左折すれば天保山マーケットビレッジや海遊館へ向かうことができます。

そこからオレンジになっている OK ボタンをクリックすると描画データを記録していきます。距離が長いとそれだけ時間がかかるので気長に待ちましょう。記録が完了すると、一旦記録を保存するために、Save ボタンをクリックして保存しましょう。`cgx` という拡張子がついて保存されます。

第 6 章　Docker のアプリコンテナとして Re:VIEW を動かそう

柴田充也

Re:VIEW[*1] は複数のフォーマットに対応した書籍作成フレームワークです。本誌も vol.1 の頃から Re:VIEW を使って EPUB や PDF を生成しています。そこで本記事はこの Re:VIEW を Docker を用いて「アプリ」コンテナ化し、誰でも簡単に実行環境を構築できる方法を紹介しましょう。

6.1 Docker とアプリコンテナ

Docker はカーネルのコンテナ技術などを利用して，アプリケーションをサンドボックス環境の中で動かす仕組みです。`Dockerfile` としてイメージファイルの構築方法を記述し、一度作成したイメージをベースにコンテナを立ち上げアプリケーションを実行します。イメージファイルはインターネットを経由して共有できますので、誰かが作ったイメージファイルをそのまま流用することや、イメージファイルをベースに新たなイメージファイルを作ることも可能です。

Docker は主にサーバー分野で使われてきました。たとえば何がしかのサービスをデプロイするとき、まずは OS をインストールしてからサービスごとにさまざまなソフトウェアをインストールし、設定ファイルを記述しなくてはなりません。デプロイに際して行ったことは手順書として残しておかないと、別のマシンに同じサービスをデプロイするときに苦労することになります。その手順書は、基本的に OS をインストールした直後のクリーンな環境を前提に記述されていることでしょう。つまり同じマシンに複数のサービスをデブロイするとなると、複数の手順書をもとに辻褄をあわせなくてはなりません。また複数のサービス間で、手順書の一部を共有できる場合もあります。

一台のマシンに複数のサービスを立ち上げるのであれば、サービスごとに仮想マシンを作るという手もあります。この方法であれば手順書の辻褄はあわせなくても「きちんと動く」はずです。しかしながら仮想マシンは立ち上げた数だけリソースを消費します。どんなにハイエンドなマシンであっても、数十個の仮想マシンが限度でしょう。しかもすべてのサービスが常に必要だとは限りません。気軽に仮想マシンをオン・オフする仕組みも欲しいところです。

Linux のコンテナ技術を使えば「より軽量な仮想環境」を構築できます。プロセスの起動と同じような感覚で、ホストからは隔離された仮想環境を「起動」できるのです。Docker は Linux のコンテナ技術を、サーバーアプリケーションの管理者向けに使いやすくしたツールです。Docker を使えば、ミドルレンジなスペックのマシンであっても、簡単に数百個のサービスを個別の「コンテナ」として立ち上げることができます。つまり同じマシンに複数のサービスをデプロイする際に、他のサービスを気にせずに手順書を実施すれば良いことになります。

さらにデプロイの手順書自体も「`Dockerfile`」として、人間からも機械からも読みやすいフォーマットで記述できます。`Dockerfile` さえ正しく記述できていれば、同じ環境をコマンドひとつで再構築できるのです。`Dockerfile` は他の構築済みのコンテナイメージをベースイメージとして指定できますので、手順書の共有も簡単です。サーバーアプリケーションによっては、フロントエンドとデータベースなど異なる役割（ロール）を持った複数のサービスを組み合わせる必要もあるかもしれません。Docker を使えばロールごとにコンテナを作り、コンテナ間を連携させれば容易に実現できます。

このように Docker はサーバーアプリケーションと非常に相性が良い仕組みを実装しています。実際、最近はサーバーアプリケーションを提供する側がいわゆるインストールガイドとは別に `Dockerfile` や構築済みのイメージを提供する流れになってきました。管理者側も Docker の使い方さえ覚えておけば、コマンドひとつでさまざまなサーバーアプリケーションをコンテナ上で試せる時代になっている

[*1] https://github.com/kmuto/review

のです。

さて Docker が「サーバーアプリケーション」を気軽に試す上で便利なツールであることを説明しました。しかしながら実は「サーバー」を省いた「アプリケーション」であっても状況は変わりません。GUI アプリケーションの場合はデスクトップ環境やツールキット、各種バスサービスとの連携が必要であるため、Docker だと容易には実現できませんが、ひとつのコマンドで実現できる CUI アプリケーションであれば、Docker を使って実行環境を提供するという方法も現実的になります。実際のところ、これまで「Python+PIP」や「Node.js+npm」、「Ruby+Gem+Bundler」などように言語ごとに実行環境を構築する仕組みは作られてきました。Docker は、これらアプリケーションの実行環境をコンテナの中に閉じ込めて「アプリコンテナ」とすることで、システムとは隔離した環境として利用できるのです。これによりホストシステムには依存しない方法でソフトウェアを導入できるようになりますし、同じマシン上で異なるバージョンの共存も簡単になります。

今回紹介する Re:VIEW は Ruby を利用したソフトウェアです。Ubuntu の場合は、まず公式パッケージ版の Ruby を使うのか本家の Ruby の最新版を導入するのかを考えなくてはなりません。また、PDF の生成には TeX を利用していますので、どのバージョンの TeX をどのようにインストールしなくてはならないかという点も考慮が必要です。執筆者がぼっちであれば、ぼっちマシン上で環境を構築すれば事足りますが、複数人で執筆・編集する場合は各自の環境で EPUB や PDF を生成したいところです。そうなると Re:VIEW だけでなく、Ruby や TeX のバージョンもあわせなければならないケースも出てくるでしょう。

Docker を使って、Re:VIEW 環境構築済みのイメージを配布すれば、各執筆者の環境を簡単に統一できます。これが Re:VIEW を「アプリコンテナ」化する意味です。

6.2 まずは実践

Re:VIEW コンテナの中身の解説はあとまわしにして、まずは実際に動かしてみましょう。

Docker を使う上で最低限インストールしなくてはならないものが Docker デーモンと Docker クライアントです。このふたつは同じホスト上にある必要はなく、たとえばリモートの Ubuntu サーバー上に Docker デーモンをインストールし、ローカルマシンの Docker クライアントから操作するといった使用方法も可能ではあります。ただ Re:VIEW のようなソフトウェアの場合は、ローカルですべて完結するでしょうし、今回は Ubuntu 16.04 LTS にすべてインストールする前提で話を進めましょう。ちなみに macOS や Windows でも Docker は使えます。

Ubuntu に Docker をインストールするなら、公式リポジトリの `docker.io` パッケージを利用する方法がいちばん簡単です。比較的新しいバージョンのパッケージが用意されていますので、Docker 公式の最新版でなくても良いのであればこの方法をおすすめします。手順としては、パッケージをインストールし、`docker` グループにユーザーを追加し、ログインし直すだけです。`docker` グループに所属しておくと、以降の `docker` コマンドは `sudo` なしに実行できるようになります。

```
$ sudo apt install docker.io
$ getent group docker
docker:x:136:
$ systemctl is-enabled docker
enabled
$ sudo usermod -aG docker $USER
(ログインし直す)
$ docker info
(Docker デーモンの状態が表示される)
```

Docker のインストールに関する詳細な説明は Ubuntu Weekly Recipe の第 458 回[*2]に記述していますのでそちらも参照してください。第 458 回では、Docker 公式のパッケージを導入する方法や、プロキシ環境下での注意点なども記載しています。

[*2] http://gihyo.jp/admin/serial/01/ubuntu-recipe/0458

Docker コマンドで最初に覚えるべきことは「docker run で指定したコンテナを立ち上げる」ことです。Docker Hub にはすでに構築済みの Re:VIEW コンテナが存在しますので、それを使って Re:VIEW のコマンドを実行してみましょう。なお 1.3GB ほどのイメージファイルをダウンロードしますので、ネットワーク帯域には注意してください。

```
$ docker pull mshibata/docker-review
Using default tag: latest
latest: Pulling from mshibata/docker-review
(中略)
$ docker run --rm mshibata/docker-review review-pdfmaker --help
Usage: review-pdfmaker configfile
        --help                  Prints this message and quit.
        --[no-]debug            Keep temporary files.
        --ignore-errors         Ignore review-compile errors.
```

「mshibata/docker-review」がイメージファイル（Docker イメージ）の名前です。Docker イメージはローカルに構築されたイメージサーバーから検索します。もしローカルになければ、パブリックなイメージサーバーである Docker Hub を検索し、取得します。コマンド実行時の「Pulling from」が実際に取得している様子ですね。コンテナイメージにはハッシュ値やタグが付けられていて、以降はその名前を使って操作することになります。ローカルにダウンロードされたイメージは「docker images」コマンドで確認できます。

```
$ docker images
REPOSITORY              TAG                 IMAGE ID            CREATED             SIZE
mshibata/docker-review  latest              aa66e927999b        About an hour ago   1.269 GB
```

「--rm」オプションは、コマンドが完了したら（コンテナが終了したら）そのコンテナを削除するためのオプションです。Docker は「コンテナを立ち上げる」とき、指定したコンテナイメージから新しいコンテナを作成し、その上でコマンドの実行やファイルの読み書きを行います。ただし読み書きする領域は、元になった Docker イメージの上に新しく読み書きできるレイヤーという形で作成しています。このため同じ Docker イメージから複数のコンテナを立ち上げてもストレージはほぼ消費しません。「--rm」オプションはコンテナ終了時に、そのインスタンスで使用したレイヤーの部分だけを削除します。「Docker には『イメージ』と『コンテナ』という二種類の概念がある」ということだけは覚えておきましょう。

「docker run」の「review-pdfmaker --help」の部分がコンテナの中で実行するコマンドです。Docker は指定したコマンド、もしくはそのコンテナイメージの Dockerfile にある CMD フィールドで指定されたコマンドを PID=1 のプロセスとして起動します。また PID=1 のプロセスが終了したらコンテナを終了します。つまりいわゆる OS が起動するときの init デーモンは呼びだされません。サーバーアプリケーションのような、デーモンとして常駐するプログラムの場合は、そのデーモンが動き続ける限りは、Docker コンテナが動いているということになります。

さて、Docker はイメージからコンテナインスタンスを作成し、インスタンス上のレイヤーにデータを読み書きすることがわかりました。イメージもレイヤーもその中身はホストからは直接見えませんし、コンテナからもホストのファイルシステムにはアクセスできません。永続的なデータの保存領域を用意したい場合、コンテナからホストのファイルを読み書きしたい場合は「ボリューム」を利用します。Docker における「ボリューム」はコンテナとは独立したデータを保存する領域です。このボリュームの使い方のひとつに、「ホストのファイルシステムをボリュームとしてマウントする」方法が存在します。実際に試してみましょう。

```
$ docker run --rm -it -v ~/:/mnt mshibata/docker-review bash
root@9da00a7f3778:/data# ls /mnt
ダウンロード  ビデオ          テンプレート  ピクチャ
デスクトップ  ミュージック    ドキュメント  公開
root@9da00a7f3778:/data# exit
```

第 7 章　Ubuntu で血迷ってアダルトサイトを作ってみた話

長南 浩（Ubuntu Japanese Team）

本業が忙しすぎて頭がおかしく（？）なっていたときに、血迷ってアダルトアフィリエイトサイトを作った話を紹介します。

ここで紹介するということは読者のみなさんの「お察し」の通り、すでにサイトを閉鎖してネタとして熟成したということになるわけで、あまり資料も残っていないヨタ話です。こんな話を特集に書いていいのか謎ですが。

でも私と同じように無修正サイトのアフィリエイトに手を出すのはいろいろリスクが高そうなので、やめておいたほうが良いと思います。

7.1 プロローグ

その男は憔悴しきっていた。彼は半年ほど前から歯車が狂い放置されてきた職場で Web アプリを書いていた。いや「書いていた」というのは語弊があり、マネジメント・顧客との打ち合わせ・Excel という悪魔のアプリケーションを使った資料づくり・事務所内の PC のトラブルシュートといった雑務の合間になんとかつじつまを合わせるように Web アプリのコードを書いていたのだ。

なぜこれまでに苦労し、モンスターに育ててしまった「お客様」に媚びへつらい、「理不尽なシステムを作る姿勢」を見せねばならぬのか。なぜそこまでして働かねばならぬのか。最低限食うに困らないだけの収入があれば、誰も幸せにならない Web アプリを作る必要もないだろう。

彼にとって当面必要なのは収入であることは間違いなかったが、心身を傷つけてまで取り組むことではないことは認識していた。幸い転職先が見つかりつつある時であったが、それにしても自身がおかれた状況はストレスが多く、何か破天荒なことをやらねば収まりがつかぬ状態だった。そんな折に彼はふと

「アダルトアフィリエイトサイトでも作るか」

と考えた。変にやり方を間違えて大儲けすればそれで良し、そうでなくても気分転換にはなるだろうし、後でなにかしらのネタにしてやろう。彼は重い体を鈍く動かしながらサイトを開く準備をはじめたのだった。

彼の名は「長南 浩」。それは 2016 年 1 月の出来事だった。

7.2 外部サービスの選定

アダルトアフィリエイトサイトを作るにあたり、利用する外部サービスを選定する必要がありました。大きく考えないといけないのは下記の 3 項目です。

- アフィリエイト・サービス・プロバイダ（ASP）
- Web サーバ運営インフラ
- サービス用のドメインと、ドメインレジストラ選定

一言で ASP と言ってもいろいろな分野があります。アフィリエイトということで少し考えた結果、人間の本能に訴え、かつ可能な限りエグいということで無修正 AV 動画配信で有名な「カリビアンコム」の広告を取り扱っている DTI アフィリエイトを使うことにしました。

Web サーバとドメインについては、普通のサイトであれば何の考えもなく、安い事業者や管理に慣れているところを選定すれば良いのですが、国内の VPS 事業者はアダルトサイトに利用することを規約で禁じているところが大半です。

リソースの使われ方や評判リスクの観点からアダルトサイトの運営を禁止しているのでしょうけれど

も、私としても問題があったときに辿られてしまうのは、何か事が起きた時に非常にマズイことになるのは想像できたので、linode を使うことにしました。もちろんサーバ OS は Ubuntu 16.04 LTS を使うことが前提条件です。

Web サーバと同様、ドメインも足がつかないように海外のサービスを使うことを考え、今回は GoDaddy を使ってみました。この GoDaddy は面白いことに日本語での情報を多く発信していて、Web サイトやコントロールパネルも日本語が使えるという、妙なところに気合が入っています。なおドメイン名は「パージする集まり」的な名前で、我ながら酷い命名だと自己嫌悪に陥りました。

なお、サーバを借りたりドメインを取るために経費が発生しますが、カネの流れから首謀者を特定できるのもイヤなので、手数料がかかるのは承知の上でプリペイドクレジットカードの「バニラ Visa オンライン」を使ってみました。このサービスでは Google Play や Webmoney のようにコンビニでプリペイドカードを買い登録することで決済できるようになる、匿名性の高いサービスなのに、私のように悪用する人間が増えたのか、2016 年 12 月にサービスそのものを閉じてしまい、現在は利用することができない状況となっています。

7.3 開発環境の整備

アフィリエイトサイトを作るということで、漠然と「Web アプリを書く」という観点で進めていたので、開発環境は普通の Ubuntu デスクトップ環境に JetBrains の IntelliJ IDEA Ultimate を使いました。

Web アプリを書くという観点からすれば、フレームワークや言語によって特化した IDE を使ってもよいのですが、どうせ 1 つの言語だけで済む話ではなくなる上にライセンス的にサブスクリプションを継続すると、2 年目・3 年目以降のライセンス価格がディスカウントされるという魅力があり、全部入りのものを購読しています。

インストールも Ubuntu make を使うとこのように簡単にインストールできるので非常に便利です。

```
$ sudo apt-add-repository ppa:ubuntu-desktop/ubuntu-make
$ sudo apt-get update
$ sudo apt-get install ubuntu-make
```

なお、Web アプリのフレームワークは個人的な趣味で Laravel を使いました。サーバのセットアップも特に普通の LAMP スタックな Web サーバを作るということで面白みもない感じで淡々と進めました。この記事を読むような方にはイマサラな感じなので、ざっくり省略してしまいます。

7.4 特筆すべき「ておくれ」項目

このプロジェクトというか私のストレス発散場所ですが、時間の開いているときに少しずつ機能などを追加していく形で開発していたのですが、サイトを閉鎖してしまった今はソースコードレポジトリくらいしか資料が残っていない状況です。基本的に、DTI で準備している動画をそのままサイトで流して、気に入ったら「サイトで登録する」という流れです。

そんな中、色々頭がおかしいフィーチャーを思い出しながら紹介します。

7.4.1 HTML・CSS・Javascript の圧縮

あまり「ておくれ」とは言えないところかもしれませんが、アダルトサイトでは通信量を 1 バイトでも削りたい欲求があるのかと考え、HTML や CSS や Javascript は最小化してサーバに配備することにしました。

しかし単に「最小化」と言っても手作業でやってる暇はありません。Laravel では Laravel Elixir という基本的な Gulp タスクが実装されているので、これに乗っかる形でリスト 7.1 のような Gulpfile を買いてみました。無駄に `require('laravel-elixir-html-minify');` しているところ、我ながら中二病的な何かを感じます。

リスト 7.1: minify 指定している Gulpfile

```
process.env.DISABLE_NOTIFIER = true;
var elixir = require('laravel-elixir');

require('laravel-elixir-html-minify');
... (略) ...
elixir(function(mix) {
    ... (略) ...
    // Composer サードパーティライブラリのインストール
    mix.copy(
            'vendor/components/bootstrap/css/bootstrap-theme.min.css',
            'public/css/bootstrap-theme.min.css')
        .copy(
            'vendor/components/bootstrap/fonts',
            'public/fonts')
    ... (略) ...
}
```

7.4.2 ヘンタイな URL Route

Laravel では、リクエストがあった URL に対してどのコントローラに制御を渡すのかというのを Route という概念で制御します。

リスト 7.1 の /home のように記述するのが普通なのですが、その後ろの指定は何なんでしょう？

これは表 7.1 のように動画のカテゴリに対して数字を対応付けているのですが、/category/ の後に数字をたくさんつなげて動画の絞り込みを行うために作ったものです。無限にいくらでもつけられればよかったのですが、うまいやり方が見つからなかったので、最大 50 個までカテゴリを検索対象に追加できるようにしていました。

表 7.1: 動画のカテゴリ名

ID	カテゴリ名	作成日時	更新日時	対象動画数
1	美尻	2016-05-05 14:45:08	2016-06-01 08:17:26	174
2	美乳	2016-05-05 14:45:08	2016-06-01 08:17:26	210
	（以下自粛）			

/category/1/2 へのアクセスは「美尻」かつ「美乳」のカテゴリタグの両方を持つ動画をリストアップする形です。

リスト 7.2: ヘンタイな Route 記述

```
... (略) ...
Route::get('/home', 'HomeController@index');
... (略) ...
for ( $i = 50 ; $i >= 1 ; $i-- ) {
    $path_str = '';
    for ( $j = 1 ; $j < $i ; $j++) {
        $path_str .= '/{id' . $j . '}';
    }
    Route::get('/category' . $path_str, 'CategoryController@show');
}
```

7.4.3 バッチ系コマンド

一般的な Web アプリの場合、アプリケーションの挙動の機転は何らかのユーザの操作なのですが、それ以外に「勝手に動いてほしいバッチ系の処理」というのも存在します。

Laravel の場合には、CLI 実行コマンドを準備する機構が含まれていて比較的簡単にバッチ系のコマンドを書くことができるのですが、今回のサイトではこんなものを準備していたようです（app/console/Commands）。

「うぶんちゅ! まがじん ざっぱ〜ん♪」バックナンバー

vol.1（2013 年 12 月 20 日発行）

表紙 瀬尾浩史
プチ帰ってきた『行っとけ! Ubuntu 道場!』
特別編 hito
Ubuntu で作るおうち録画環境 kazken3
Unity から自由を奪還せよ 柴田充也
これで完璧!! Ubuntu で印刷 again! おがさわらなるひこ
Enju Leaf でつくるオレオレ蔵書管理サーバ 長南 浩
Ubuntu マシンで艦これを動かして遠隔プレイできる環境を作ってみた 鶴ノ子餅すあま
SD 連載 Ubuntu Monthly Report における動物の変遷 SoftwareDesign 編集部 金田冨士男
This is me, with Ubuntu. おしえたかし
うぶんちゅ まがじん ざっぱーん Vol.01 発刊おめでとう 水野源
Ubuntu と私 あわしろいくや
終わりに あわしろいくや
価格：500 円
販売サイト・体験版：
http://zapppaaan.freepub.jp/article/82821893.html

vol.2（2014 年 11 月 15 日発行）

表紙 Ubuntu 4.10
プチ帰ってきた『行っとけ! Ubuntu 道場!』
特別編 第二回 hito
あのプロダクトは今!? 柴田充也
Ubuntu を「未来をうかがう」道具にする 長南 浩
普通の社会人が【録画環境】を (もう少し) やってみた kazken3
Ubuntu 10 歳、CUPS 15 歳 おがさわらなるひこ
Ubuntu 14.04/14.10 でも ATOK X 3 を動かす あわしろいくや
Ubuntu Studio フレーバー 7 周年 坂本 貴史
Ubuntu で Blink(1) mk2 を動かしてみる kazken3
Ubuntu8.04 はこんなだった あわしろいくや
人と Ubuntu と私 芝田 静間
第二弾おめでとうございます Ueno
私が愛してきた OS 達 長南 浩
普通の大学教員が【 Ubuntu 】やってみた。 おしえたかし
終わりに あわしろいくや
価格：700 円
販売サイト・体験版：
http://zapppaaan.freepub.jp/article/105631657.html

vol.3（2015 年 7 月 25 日発行）

表紙 写真：水野源
OpenNebula で PCI passthrough おおたあきひこ
Let’s note CF-RZ4 に Ubuntu をインストールしてみる話 Rakugou
21 世紀の Device Tree 柴田充也
ちょーなんさんちのノート PC 事情 長南 浩
かよちんとボクと、時々、録画 kazken3
磁気センサーを使った冷蔵庫監視システムの構築 水野源
新し目の Mozc をビルドする あわしろいくや
著者紹介
編集後記 あわしろいくや
価格：700 円
販売サイト・体験版：
http://zapppaaan.freepub.jp/article/156663726.html

vol.4（2016 年 1 月 30 日発行）

表紙 イラスト：よかぜ
艦これで学ぶ通信傍受入門 柴田充也
お得でおいしい SSL 証明書のとりかた 長南 浩
Xymon Maniax Hajime MIZUNO
Ubuntu と知の巨象、Evernote との共存をめぐって Rakugou
Ubuntu GNOME の再インストールと棚卸し あわしろいくや
Blu-ray を Ubuntu で観よう kazken3
海外カンファレンスの楽しみ おがさわらなるひこ
著者紹介
記録に残るものが記憶に残るもの〜あとがきに代えて〜 あわしろいくや
価格（Gumroad）：500 円
価格（DLsite）：756 円
販売サイト・体験版：
http://zapppaaan.freepub.jp/article/173095293.html

vol.5（2016 年 8 月 21 日発行）

価格（Gumroad・BOOTH）：700 円
価格（DLsite）：972 円
販売サイト・体験版：
http://zapppaaan.freepub.jp/article/176563093.html

総集編 1（2017 年 4 月 9 日発行）

価格（Gumroad・BOOTH）：700 円
価格（DLsite）：972 円
販売サイト・体験版：
http://zapppaaan.freepub.jp/article/179274130.html

うぶんちゅ！　まがじん ざっぱ～ん♪ Vol.6【体験版】

2017 年 4 月 9 日　v1.0.0 発行
著　者　team zpn
発行所　team zpn

発行● team zpn