
vol.7
Ubunchu!
Magazine
Zapppaan
体験版

うぶんちゅ！ まがじん ざっぱ〜ん♪ vol.7【体験版】

team zpn　著

2018-02-11 版　　team zpn 発行

目次

表紙イラスト：よかぜ
装幀：team zpn

第 1 章　OpenNebula で Nextcloud サーバー構築

おおたあきひこ

青羽家は OpenNebula で家庭内クラウドを構築し、ここなちゃんとママの二人で計算機リソースをやりくりしています。

おや？ 居間の Ubuntu 17.10 デスクトップマシンに向かっているママが何やらお困りのようです。

「変ねぇ、オンラインストレージに写真がアップロードできないわ…」

どうやら商用オンラインストレージサービスの無料で使える容量を使い切ってしまったみたいです。今月は支出が多く、ストレージ容量の追加購入にも踏み切れません。ここなちゃんも、ほのかちゃんに撮ってもらった夢馬くん画像がアップロードできなくて大弱りです。

1.1 OpenNebula ノススメ サードシーズン

今回は、どこのご家庭でもおなじみの OpenNebula なクラウド環境で仮想マシンによる Nextcloud サーバーを構築し、Ubuntu 17.10 デスクトップの GNOME オンラインアカウントに登録してみます。OpenNebula とは何ぞやについては、gihyo.jp の Ubuntu Weekly Recipe で紹介記事を書きましたので[*1][*2]、そちらをご参照ください。

「あの」

Nextcloud は ownCloud 派生のファイル同期サーバー/クライアントです。なんとかボックスやなんとかドライブといった、いわゆるオンラインストレージ的なサービスを簡単に構築できます。こちらも Ubuntu Weekly Recipe であわしろいくやさんが紹介されていますので[*3]、併せてご参照ください。

「ざっぱ〜ん♪ vol.6 で他の作品に浮気されてませんでした？」

ひぃぃぃゴメンナサイゴメンナサイ。3 期の話がなかなか聞こえてこなくて心が挫けてしまったんですぅぅ…

1.2 前置き

- デスクトップ環境は Ubuntu 17.10 の GNOME 3.26 を使用します。
- OpenNebula は OpenNebula.org 公式リポジトリのバージョン 5.4.6 を使用します。
- OpenNebula 環境がひととおりセットアップされているものとします。

1.3 仮想マシンイメージの用意

Nextcloud は Ubuntu 16.04 LTS 以降であれば Nextcloud 社公式の snap パッケージが利用でき、コマンドひとつでセットアップが完了します。つまり Ubuntu 16.04 LTS 以降の仮想マシンイメージが用意できれば Nextcloud の構築は八割方終わったようなものといっても過言ではありません。

OpenNebula では MarketPlace にて Ubuntu を含む各種ディストリビューションの仮想マシンイメージが配布されており[*4]、`onemarketplaceapp` コマンドや Web インタフェースの "Sunstone" から手元の OpenNebula 環境に登録できます。これを利用すれば非常に楽ではあるのですが、楽なもの (snap) に楽なもの (MarketPlace) を重ね掛けすると原稿のボリュームが著しく痩せてしまうので、今

[*1] Ubuntu Weekly Recipe 第 483 回 https://gihyo.jp/admin/serial/01/ubuntu-recipe/0483
[*2] Ubuntu Weekly Recipe 第 484 回 https://gihyo.jp/admin/serial/01/ubuntu-recipe/0484
[*3] Ubuntu Weekly Recipe 第 476 回 https://gihyo.jp/admin/serial/01/ubuntu-recipe/0476
[*4] OpenNebula MarketPlace https://marketplace.opennebula.systems/appliance

回は少し趣向を変えて Ubuntu 公式のクラウド向けイメージ「Ubuntu Cloud Images」[*5] を利用することにします。

OpenNebula フロントエンドにログインし、wget コマンドなどで Ubuntu 17.10 server (Artful Aardvark) の current のイメージをダウンロードします。ダウンロード先は/var/tmp とします[*6]。

```
$ wget -P /var/tmp 'https://cloud-images.ubuntu.com/artful/current/artful-server-cloudimg-amd64.img'
```

1.3.1 cloud-init へのパッチ当て

Ubuntu Cloud Images のイメージには、各種クラウド向け仮想マシン初期設定ユーティリティの "cloud-init" がプリインストールされています[*7]。cloud-init は OpenNebula にも対応しており、仮想マシン起動時に OpenNebula から渡される IP アドレスやディスクサイズといった構成情報を受け取ってよしなに設定してくれる、ことになっています。

が、Ubuntu 16.04 LTS〜17.10 の cloud-init version 17.1 には OpenNebula 向けの実装部分に不具合があり、OpenNebula 環境ではネットワークが正しく設定されません。とりあえず修正パッチを適用します[*8]。

Launchpad の以下の URL から wget コマンド等でパッチをダウンロードします。

リスト 1.1: cloud-init の OpenNebula 向け修正パッチ URL

```
https://launchpadlibrarian.net/338298258/DataSourceOpenNebula.network_config.patch
https://launchpadlibrarian.net/338331568/support.PredictableNetworkInterfaceNames.patch
https://launchpadlibrarian.net/348301526/fix_return_empty_string.patch
```

Ubuntu 17.10 のイメージをローカルマウントします。qcow2 形式なので、nbd ドライバをロードして qemu-nbd コマンドで nbd デバイスにアタッチし、/dev/nbd0p1 をマウントします。

```
$ sudo modprobe nbd
$ sudo qemu-nbd -c /dev/nbd0 /var/tmp/artful-server-cloudimg-amd64.img
$ sudo mount /dev/nbd0p1 /mnt
```

以下のディレクトリまで降りて、patch コマンドで DataSourceOpenNebula.py にパッチを当てます。

```
$ cd /mnt/usr/lib/python3/dist-packages/cloudinit/sources/
$ sudo patch -p3 < /var/tmp/DataSourceOpenNebula.network_config.patch
$ sudo patch -p3 < /var/tmp/support.PredictableNetworkInterfaceNames.patch
$ sudo patch -p3 < /var/tmp/fix_return_empty_string.patch
```

アンマウントしてイメージを nbd デバイスからデタッチします。

```
$ cd ~/
$ sudo umount /mnt
$ sudo qemu-nbd -d /dev/nbd0
```

[*5] Ubuntu Cloud Images https://cloud-images.ubuntu.com/

[*6] ローカルファイルシステムからイメージを登録する場合、OpenNebula のデフォルト設定では登録元のディレクトリを/var/tmp のみに制限しています。

[*7] 対して MarketPlace で公開されているイメージには OpenNebula 専用の仮想マシン初期設定ユーティリティ "one-context" がプリインストールされています。仮想マシンに追加で NIC をアタッチした際の自動 IP アドレス設定など、OpenNebula のいくつかの機能は one-context でのみサポートされています。

[*8] この修正パッチは現在は merge されており、version 18.1 以降の cloud-init ではパッチ当て作業は不要です。それ以前のバージョンにもそのうちバックポートされるかもしれません。

1.3.2 仮想マシンイメージの登録

cloud-init にパッチを当てた仮想マシンイメージを OpenNebula のデータストアに登録します。Web インタフェースの Sunstone からでも登録できますが、今回は oneimage コマンドから登録することにします。まずは以下のような定義ファイルを書きます。

リスト 1.2: イメージ定義ファイル

```
NAME          = "artful"
PATH          = "/var/tmp/artful-server-cloudimg-amd64.img"
TYPE          = "OS"
DESCRIPTION   = "Artful Aardvark Cloud Image"
DEV_PREFIX    = "vd"
DRIVER        = "qcow2"
```

表 1.1: 定義ファイルのパラメータ

変数名	値
NAME	イメージの名称
PATH	登録するイメージファイルのファイルパス
TYPE	仮想マシンイメージの場合は "OS" と指定
DESCRIPTION	イメージの説明文
DEV_PREFIX	仮想マシンでディスクアクセスに virtio ドライバを使用する場合は "vd" と指定
DRIVER	イメージファイルのフォーマット。今回は "qcow2" と指定

この定義ファイルを oneimage create コマンドの引数に指定して実行します。-d オプションは登録先データストアの指定です。今回はデフォルトで用意されている default データストアを指定します。

```
$ oneimage create image.one -d default
```

1.4 仮想マシンへの IP アドレス予約

OpenNebula のデフォルトの動作では、仮想マシンをデプロイするタイミングで、ネットワークリソースのアドレスレンジから未使用の IP アドレスが仮想マシンに割り当てられます[9]。

サーバー用途などの仮想マシンを専用の IP アドレスで固定したい場合、事前に onevnet reserve コマンドで IP アドレスを予約しておきます。

たとえば、192.168.1.169〜176 の 8 個のアドレスレンジ (アドレスレンジ ID: 0) を持つネットワークリソース private から、IP アドレス 192.168.1.169 を nextcloud という名称で予約する場合は、以下のコマンドを実行します。

```
$ onevnet reserve private -n nextcloud -a 0 -s 1 -i 192.168.1.169
```

onevnet reserve コマンドで予約後、onevnet list コマンドでネットワークリソース一覧を見ると、ネットワークリソース nextcloud が作成されていることがわかります。

```
$ onevnet list
  ID USER            GROUP         NAME              CLUSTERS   BRIDGE   LEASES
   0 oneadmin        oneadmin      private           0          br0           1
   1 oneadmin        oneadmin      nextcloud         0          br0           0
```

[9] テンプレートの IP 属性にて IP アドレスを明示的に記述することもできます。ただし、この場合は他の仮想マシンが先にそのアドレスを使用しているとデプロイに失敗します。

第 2 章　サーバーレスでオレオレ画像アップローダーを作る

水野源

Ubuntu Weekly Recipe 第 413 回[*1]では、Ubuntu サーバーを使ってプライベートな Gyazo サーバー[*2]を構築する方法を解説しました。本記事では同様の機能をサーバーレスアーキテクチャで実装する方法を紹介します。

2.1 概要と設計

Gyazo はアップロードされた画像を受け取り、保存し、URL で閲覧可能にするだけの単純なサービスですから、個人や少人数での利用であれば、サーバーのリソースはほとんど必要ありません。とはいえ、サービスである以上は 24 時間 365 日リクエストに応える必要があり、サーバーを常時稼動させておかねばならないという二律背反が発生します。

ご存知の通り、クラウドサービスというものはコスト的には割高です。しかし計算機資源の動的な割り当てが可能であることから、使っていない時間にサービスを縮小することで、結果的に安く上げることができるわけです。つまり、リクエストがあった時だけサービスを起動できれば理想的です。

これを実現するのが、今流行りの**サーバーレスアーキテクチャ**です。AWS の機能を使い、自前でサーバーを用意することなく、Gyazo 的なサービスを作ってみましょう。

まず独自ドメインでサービスを運用するため、ドメインを取得していることを前提に解説します。また後述する ACM での認証操作を簡単にするため、ドメインは Route 53 でホストしていることを想定しています。

画像の保存には **Amazon S3** を使います。S3 には静的な Web サイトをホスティングする機能[*3]があり、Web サーバーなしで画像や HTML の公開が可能なため、本サービスの根幹である「画像の保存」と「URL 指定による画像へのアクセス」を S3 単体で実現可能です。また独自ドメインの URL を提供できるのも魅力です[*4]。

クライアントから画像を受け取り、S3 に格納する処理には **AWS Lambda** を使います。Lambda はサーバーのプロビジョニングなしで、コードを実行できるサービスです。インフラの管理を AWS に丸投げして、自分は実行するコードのみを考慮すればいいため、今回のような用途には最適です。

Lambda 関数は AWS 内の様々なイベントをトリガーとして起動されます。これを REST API から起動できるよう、エンドポイントを準備してあげる必要があります。これには **Amazon API Gateway** を使います。また **AWS Certificate Manager** で SSL 証明書を発行し、独自ドメインの API エンドポイントに SSL でアクセスできるようにします。

[*1] https://gihyo.jp/admin/serial/01/ubuntu-recipe/0413
[*2] https://gyazo.com/ja
[*3] http://docs.aws.amazon.com/ja_jp/AmazonS3/latest/dev/WebsiteHosting.html
[*4] ただし CloudFront を併用しない場合、独自ドメインに SSL でアクセスすることはできません。

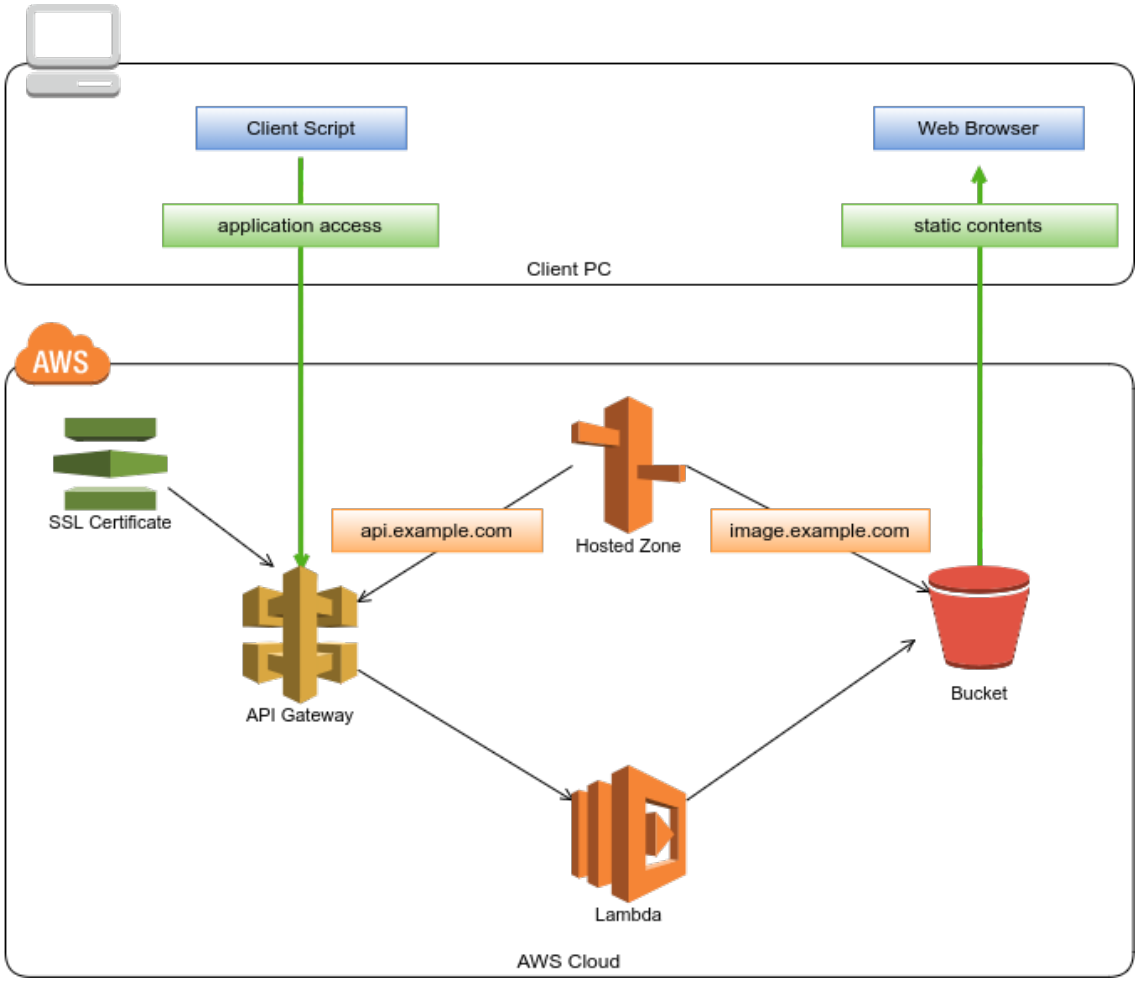

図 2.1: サービスの構成図。API Gateway が提供するエンドポイントに対して画像を POST すると、Lambda 関数が S3 に保存する。保存した画像には S3 の静的ウェブホスティング機能を利用して、Web ブラウザーからアクセスできる。API やバケットの URL には、Route 53 でホストする独自のドメインを利用する。

2.2 S3 バケットを作る

まず画像を保存する S3 バケットを作ります。バケット名は画像を公開する FQDN と同一 (例: `image.example.com`) にします。これは独自ドメインを使って S3 で静的ウェブサイトをホストする際の、AWS の仕様です[*5]。

[*5] http://docs.aws.amazon.com/ja_jp/AmazonS3/latest/dev/website-hosting-custom-domain-walkthrough.html#root-domain-walkthrough-s3-tasks

図 2.2: 画像を公開する FQDN と同じ名前のバケットを作る。

バケットの「アクセス権限」→「バケットポリシー」を開き、リスト 2.1 の内容を記述します。これで、このバケットに対するパブリックな読み込み権限が付与されます。

リスト 2.1: バケット内の全オブジェクトに対して GetObject を許可するバケットポリシー。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "PublicReadGetObject",
            "Effect": "Allow",
            "Principal": "*",
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::バケット名/*"
        }
    ]
}
```

バケットの「プロパティ」→「Static website hosting」を開き、ウェブサイトのホストを許可します。インデックスドキュメント名を指定し (適当で構いません) 保存します。

これで、オブジェクトを公開できるバケットの準備はできました。次に独自ドメインでバケットにアクセスできるよう、DNS レコードを作成します。

Route 53 で対象のゾーンを開き、「Create Record Set」をクリックします。バケット名に使用したのと同じサブドメインを入力し、タイプは A レコードの Alias を選択します。Alias Target は S3 website hosting のエンドポイントを指定してください。

第 3 章　BuildStream で GNOME アプリをビルド

柴田充也

GNOME のアプリケーションは GNOME 製のさまざまなライブラリに依存しています。次期リリースに向けて開発中のアプリは、同じく次期リリースに向けて開発中のライブラリを使うことになります。しかしながら、ホストマシンに開発中の不安定なライブラリをインストールすると、開発マシンが不安定になり最悪デスクトップ環境が動かなくなります。でも、アプリの新機能を実装するためには、リリース前の開発中のライブラリが必要です。本記事ではそんな開発環境の悩みを解消し、より GNOME 開発者向けの継続的インテグレーションをもたらすために開発された**「BuildStream」**を紹介しましょう。

3.1 GNU Network Object Model Environment

当初の GNOME は**「GNU Network Object Model Environment」**の頭字語であると紹介されていました。これはネットワーク透過型のオブジェクトモデルによって構築された環境という意味で、要するにアプリを含むさまざまなコンポーネントがネットワーク越しに連携して動作するようなモダンなデスクトップ環境を構築しようとしていたわけです。とは言え「頭字語であること」はほどなく取り下げられ、なんの略称でもないと説明されることになります。COM だの UNO だのオブジェクト指向がやたらと流行った前世紀末のフレームワークやツールと同様に、「名前にオブジェクトという単語を入れなければならない病気」に罹っていただけかもしれません。

そんなデスクトップ環境である GNOME は、アプリやツールキットなどを含む UI や多種多様なソフトウェアの集合体です。ユーザーが直接触れるであろうほぼすべてのパーツを自前で用意し、UI のガイドラインを設けることで、一貫したユーザーエクスペリエンスを提供しています。個々のアプリは独立していますし、コアライブラリはコンポーネント化されていますので、基本的に個別に開発が進みます。しかしながら冒頭でも説明したように、開発を進める中で依存関係が発生することもありますし、リリースまでのどこかのタイミングで統合テストを行う必要があります。ディストリビューションが作成するパッケージよりも細かい粒度で、ビルド＆テストを繰り返さなければなりません。

つまり Linux ディストリビューションには依存せずに、フルスクラッチで最新の GNOME 環境を構築できるシステムが必要になるのです。

これまでは JHBuild と呼ばれるシステムが使われてきました[*1]。これは GNOME アプリの最新版を取得し、ホストの環境にあわせて依存パッケージをインストールし、ビルドするシステムです。アプリだけでなく GNOME システムそのものを作る機能も存在します。しかしながら、JHBuild はその仕組み上ホストのツールキット（GCC など）に依存します。つまりディストリビューションをまたいで同じ結果になるとは限らないということです。また、ビルドの再現性を確保する手段が簡単ではなかったり、依存関係を人類には難しすぎる XML で記述しなければならなかったりと、JHBuild にはいくつかの問題も存在しました。

そこで新規に開発されたのが「BuildStream」です[*2]。JHBuild はもとより、Yocto や Buildroot などの既存のビルドシステムに加えて、OSTree や Bubblewrap などのコンテナ技術を参考に、ビルドから CI までを担うフレームワークとして実装されました。2018 年 1 月には待望の 1.0 がリリースされ、今後は GNOME 開発のインフラシステムとなるべく、精力的に開発を進めているようです。

BuildStream は、ホストの環境にほぼ依存しません。必要なソフトウェアは Python 3.4 以上と OSTree、Bubblewrap ぐらいです[*3]。簡単に言うと、OSTree で取得したミニマルなイメージをベー

[*1] JHBuild については、Ubuntu Weekly Recipe の第 244 回で松澤さんが紹介してくれています。https://gihyo.jp/admin/serial/01/ubuntu-recipe/0244

[*2] https://wiki.gnome.org/Projects/BuildStream

[*3] Bubblewrap を使う都合上、LXD 上で BuildStream は動作しません。もしくは LXD 上で動かすためにそれなりの設定が必要です。

スシステムとして、GNOME をビルドするスクリプト群です。ビルド環境はユーザーのホームディレクトリー以下に構築するため、管理者権限は不要です。また pyenv や ndenv のように、構築した環境の中に「ログイン」してビルドしたアプリを実行する仕組みも備わっています。つまり、ある種の SDK のような機能を一通り備えているわけです。

将来的には JHBuild のようにブータブルな GNOME イメージを構築したり、プラグインを導入することで Flatpak パッケージを構築することが可能になるようです。

3.2 BuildStream のインストール

BuildStream のインストールは非常に簡単です[*4]。これはホストの環境に極力依存しないように作られているためです。ただし OSTree を使う必要があることから、「インストールの簡単さ」を享受するためには Ubuntu 17.10 以上を使う必要があります。

ちなみにビルドだけなら、デスクトップ版でもサーバー版でも違いはありません。ただしビルドしたあとに GNOME アプリを実行してテストすることを考えると、デスクトップ版を用意したほうがいいでしょう。CPU はできるだけコア数が多く、高速なものをおすすめします。またひたすらビルドをするため、メモリも多いに越したことはありません。ストレージは最低 20GiB 程度、できれば 50GiB 程度の空きは欲しいところです。ビルド時のデータは「~/.cache/」に保存されるので、ホームディレクトリに余裕が必要です。

ホストマシンには次のパッケージをインストールしてください。

```
$ sudo apt install ¥
    python3-dev python3-pip git ¥
    python3-gi gir1.2-ostree-1.0 ostree ¥
    bubblewrap python3-ruamel.yaml fuse libfuse2
```

BuildStream の最新版をダウンロードし、必要な Python パッケージをインストールします。

```
$ git clone https://gitlab.com/BuildStream/buildstream.git
$ cd buildstream
$ pip3 install --user .
```

あとは「~/.local/bin」以下のコマンドを実行できるようにしておきます。「~/.bashrc」に書いておいてもいいでしょう。

```
$ export PATH=${PATH}:~/.local/bin
```

これだけでインストール完了です。BuildStream のコマンドは「bst」になります。

```
$ bst --version
bst, version 1.0.1.dev59+g88260c7

$ bst --help
Usage: bst [OPTIONS] COMMAND [ARGS]...

  Build and manipulate BuildStream projects

  Most of the main options override options in the user preferences
  configuration file.

Options:
  --version                       Show the version and exit.
  -c, --config PATH               Configuration file to use
```

[*4] https://buildstream.gitlab.io/buildstream/install.html

```
  -C, --directory DIRECTORY       Project directory (default: current
                                  directory)
  --on-error [continue|quit|terminate]
                                  What to do when an error is encountered
  --fetchers INTEGER              Maximum simultaneous download tasks
  --builders INTEGER              Maximum simultaneous build tasks
  --pushers INTEGER               Maximum simultaneous upload tasks
  --network-retries INTEGER       Maximum retries for network tasks
  --no-interactive                Force non interactive mode, otherwise this
                                  is automatically decided
  --verbose / --no-verbose        Be extra verbose
  --debug / --no-debug            Print debugging output
  --error-lines INTEGER           Maximum number of lines to show from a task
                                  log
  --message-lines INTEGER         Maximum number of lines to show in a
                                  detailed message
  --log-file FILENAME             A file to store the main log (allows storing
                                  the main log while in interactive mode)
  --colors / --no-colors          Force enable/disable ANSI color codes in
                                  output
  --strict / --no-strict          Elements must be rebuilt when their
                                  dependencies have changed
  -o, --option <TEXT TEXT>...     Specify a project option
  -h, --help                      Show this message and exit.

Commands:
  build          Build elements in a pipeline
  checkout       Checkout a built artifact
  fetch          Fetch sources in a pipeline
  pull           Pull a built artifact
  push           Push a built artifact
  shell          Shell into an element's sandbox environment
  show           Show elements in the pipeline
  source-bundle  Produce a build bundle to be manually executed
  track          Track new source references
  workspace      Manipulate developer workspaces
```

3.3 strict と track

BuildStream はビルド対象となるソフトウェアとそこから依存するソフトウェアを Git などを利用してダウンロードし、ビルドするシステムです。ソフトウェアごとにビルドする方法を記載した YAML 形式のファイルを**「Element」**と呼びます。しかしながら、すべての依存関係にある Element の最新版を毎回ダウンロード・ビルドしていては開発作業そのものに支障をきたします。そこでいくつかの「リビジョンを固定する」仕組みが存在します。

ひとつめは「strict」オプションです。これは特定の Element をリビルドする際に、依存する他の Element もリビルドするかどうかを設定します。BuildStream は CI などで利用するビルドシステムを想定しているため、特に指定していない場合はリビルドする設定になっています。特定のソフトウェアのテスト目的としてビルドを繰り返す場合は非効率なので、設定ファイルで strict を無効化してしまいましょう。「~/.config/buildstream.conf」に、リスト 3.1 の内容を記述しておきます。

リスト 3.1: ~/.config/buildstream.conf の内容

```
projects:
  gnome:
    strict: False
```

もうひとつの概念が「track」です。BuildStream では Git などを利用してビルド対象のソフトウェアの最新版を取得します。ビルドした時点でどのリビジョンを使用したかを記録し、必要に応じてそのリビジョンで固定したり、最新版を取得し直すのが track 機能です。こちらは主に bst コマンドのオプションとして使用します。よってその使い方は、後ほど見ていくことにしましょう。

第 4 章　LibreOffice 6.0 Writer で縦書き小冊子を作成する

あわしろいくや

LibreOffice 6.0 Writer で縦書き小冊子を作成し、印刷する方法を紹介します。

4.1 小冊子の左とじと右とじ

LibreOffice Writer には小冊子印刷機能があります。小冊子印刷は横置きページの左右に 1 ページずつ印刷する際に中央でとじても違和感がないようにできます。

[ファイル]-[印刷]-[ページレイアウト] タブに「パンフレット」がありますが、これが Writer で言うところの小冊子印刷機能です（図 4.1）。

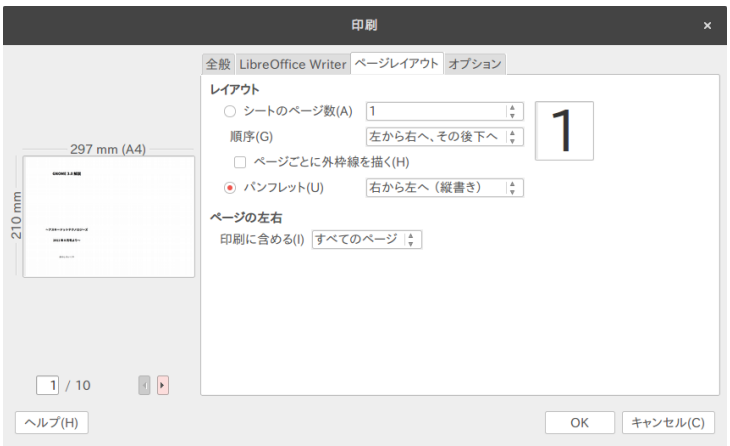

図 4.1:「パンフレット」が小冊子印刷機能

5.2 以前では左とじ（左から右へ）にしかなりませんでしたが、5.3 からは右とじ（右から左へ）もできるようになり、縦書きの小冊子の印刷が Writer 単体で行えるようになりました。同人小説の印刷などに便利なのではないでしょうか。

昨今は家庭用のプリンターないし複合機でも両面印刷機能がついているものが多いため、手軽に小冊子印刷ができるのは嬉しいところです。

というわけで、ここでは「パンフレット」印刷を行うために必要なレイアウトや設定を紹介していきます。

4.2 前準備

今回は文章を流し込んでからレイアウトを変更しますので、まずはレイアウトの前に必要なことを行っていきます。

4.2.1 使用する LibreOffice のバージョン

今回はスケジュールの都合で 6.0.0.2 という中途半端なバージョンを使用しましたが、可能な限り新しいバージョンであるのが望ましいです。具体的なインストール方法は省略しますが、Ubuntu 18.04

LTS を使用するか、AppImage 版[*1]を使用すると簡単です。

4.2.2 原稿

実際に作業に入る前にかなりの前準備が必要となります。最も大切なのは原稿を用意することです。原稿はテキストファイルでも Writer で執筆したものでも何でもいいですが、今回は後者とします。筆者の手元にも例とする適切な原稿が思いつかず、また新たに書き下ろす労力もなかったため、月刊アスキードットテクノロジーズ 2011 年 8 月号[*2]で執筆した原稿を掘り起こしてきました。

4.2.3 フォントのインストール

Ubuntu のリポジトリにある最も印刷に適しており、かつ Writer で扱えるフォントは IPAex 明朝[*3]ということで、作業の前にインストールします。[fonts-ipaexfont-mincho] パッケージか、ゴシック体も一緒にインストールするのであれば [fonts-ipaexfont] を事前にインストールしてください。

4.2.4 LibreOffice の設定

Ubuntu Weekly Recipe 第 440 回[*4]の『日本語レイアウト』を参考に、カーニングと文字間隔の調整を変更してください（図 4.2）[*5]。

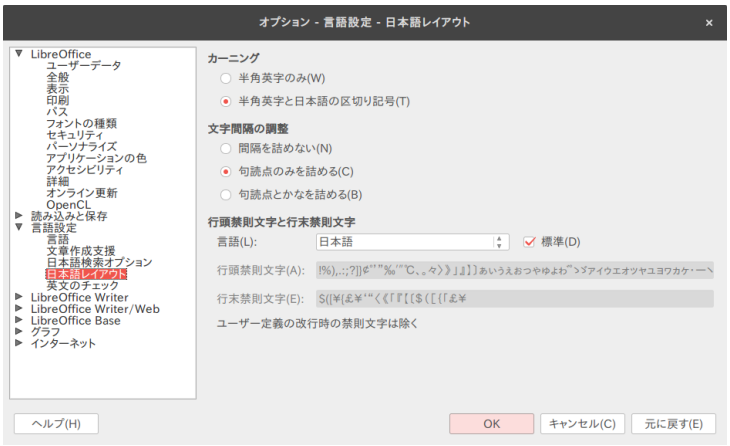

図 4.2:「カーニング」と「文字間隔の調整」を変更する

4.2.5 書式設定

サイドバーの「スタイルと書式設定」を開き、上部の 6 つあるスタイルアイコンの左から 4 番目をクリックし、ページスタイルを表示します（図 4.3）。続いて使用するスタイル（今回は「標準スタイル」）を右クリックし、「変更」をクリックします。

「インデントと間隔」（図 4.4）、「配置」（図 4.5）、「日本語の体裁」（図 4.6）、「フォント」（図 4.7）の各タブを図を参考に変更してください。

[*1] https://libreoffice.soluzioniopen.com/
[*2] https://bookwalker.jp/de491903ee-8d17-483b-a07b-5839712fe5cd/
[*3] http://ipafont.ipa.go.jp/old/ipaexfont/download.html
[*4] http://gihyo.jp/admin/serial/01/ubuntu-recipe/0440
[*5] 現在調査中ですが、少なくとも日本語ではカーニングは機能していないように見えます

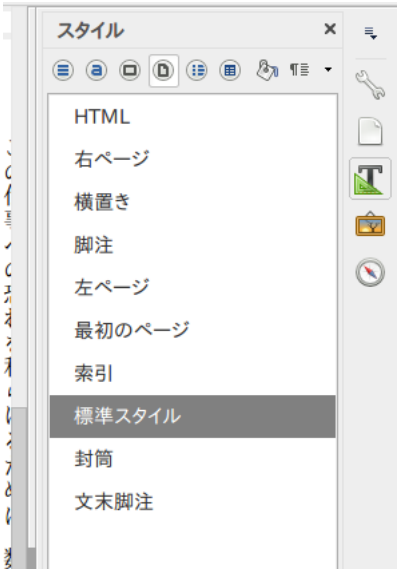

図 4.3:「ページスタイル」の「標準スタイル」はこれ

図 4.4:「インデントと間隔」タブでは既定値では狭すぎるので「行間」を変更する。

第 5 章　Ubuntu を macOS High Sierra 風に変えてみる。

Rakugou

Ubuntu が 17.10 からデスクトップシェルが Unity から GNOME Shell にチェンジするというニュースをゲットした筆者は、Unity 漬けの毎日からグラデュエーションしないと時代にスポイルされるのではないかと思いました。Unity にオプティマイズされたブレインは、すぐさま GNOME とジョインすることはディフィカルトですが、これを機に GNOME を Mac っぽくチェンジすることでスタイリッシュに、かつ Ubuntu のソフトウェアを使うことでユーザビリティもアップできるのでは、ということでスタバでドヤリングしても空間にマッチするエレガントな GNOME にドレスアップしていきます。

... すみません。わかりにくいですね。

5.1 MacBook に対する憧れ

[自分のノートパソコン] というのを意識し始めたのが大学生になってからで、ちょうど大学生であれば持っている人は一人一台ノートパソコンを持っているくらいのご時世でした。イケてない大学生であった筆者は、当時軽めの部類とされる 2kg を超えるパソコンを引っさげながらイケてる大学生が薄くて軽くて高機能でスタイリッシュな MacBook を持っているのを羨ましく眺めていたものです。それから月日が経ち、MacBook を持つ夢は未だに叶ってはいないですが、今、目の前には Ubuntu がインストールされているノートパソコンがあります。Ultrabook の台頭もあり Mac 以外の薄型で軽くスタイリッシュなノートパソコンも巷でよく見かけるようになりました。機が熟した今なら、GNOME を使ってノートブックを Mac っぽくしてみて、スタイリッシュなパソコンを**見かけだけでも**手にして夢を叶えようではありませんか。

5.2 まずは GNOME Tweaks で変えてみる

はじめに、GNOME Tweaks をインストールして、最も見た目に影響するテーマやアイコン、カーソルをインストールして変えてみます。テーマとカーソルはホームフォルダに [.themes]、アイコンは [.icons] のフォルダを作成し、GNOME-LOOK.ORG[*1] からダウンロードして解凍したフォルダを格納します。その後、GNOME Tweaks の [テーマ] の項目のそれぞれのタブの項目を変更し、合わせていきます。

5.2.1 .themes に GN-OSX-High-Sierra を入れる

テーマは執筆当時最新の macOS である High Sierra に似せたものになるように Gn-OSX H.Sierra[*2] にしました。出たばかりの OS を模したテーマがここまで早く世に出ていると思ってはいなかったです。すごい。ただ一つ注意点ですが、このテーマは開発が継続されないようで、同じく High Sierra に似せたテーマとして、Gnome-OSX[*3] が存在します。

[*1] https://www.gnome-look.org/
[*2] https://www.gnome-look.org/p/1180505/
[*3] https://www.gnome-look.org/p/1171688/

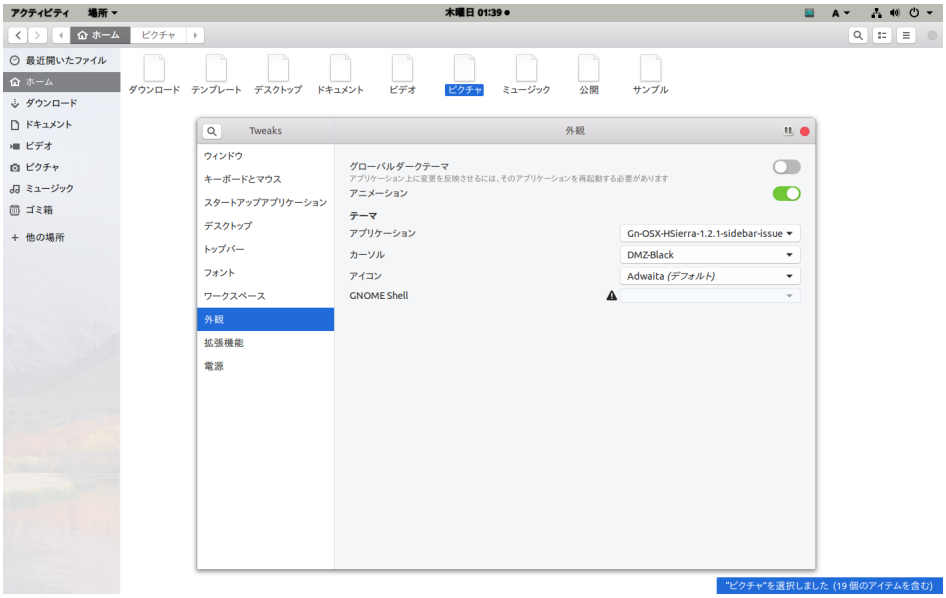

図: 単純にテーマ変更をしただけのもの、これだけでもそれっぽく見えてくる。

5.2.2 .icons にアイコンセットを入れる

アイコンセットはものによっては本家とあえて見た目を遠ざけているものがあり、比較的本家に近い雰囲気を持つアイコンセットである MacOS[*4] にしました。テーマとアイコンで設定しただけですが、雰囲気がそれらしくなってきました。

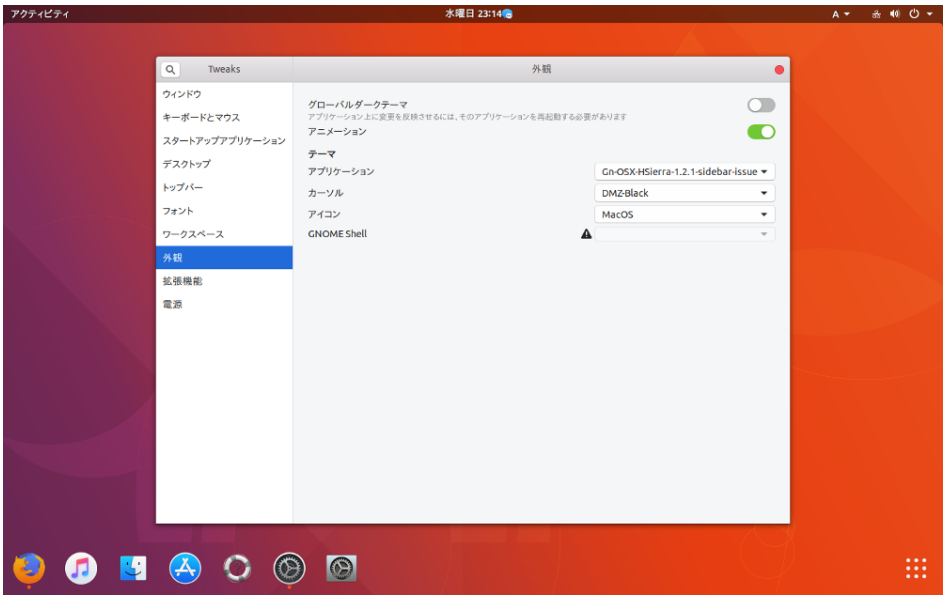

図: アイコンを MacOS へ変更。アイコンの変更をわかりやすくするために、後述する Dash to Dock という拡張機能を先にインストールしている。

[*4] https://www.gnome-look.org/p/1102582/

5.2.3 .icons に OSX-ElCap カーソルセットを導入する

カーソルセットは El Capitan を模した Capitaine Cursors[*5] を使うことにしました。ロード中のカーソルなど、それっぽくなっているのが分かるでしょう。この 3 つを変更するだけでもかなり Mac らしい外面になっていて、「Mac 風にしてみる」という意味ではほぼ近い状態になっていると思います。

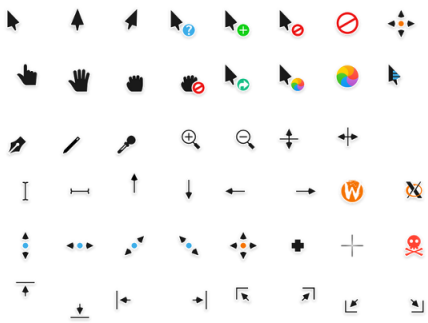

図: カーソルのスクリーンショットは撮るのが大変なので、カーソル一式のサンプル画像となります。

5.2.4 最大化、最小化ボタンの有効

Mac のカラフルな 3 色のボタン（閉じる、最小化、最大化）を表示させるため、GNOME Tweaks を使って設定します。GNOME Tweaks の [ウィンドウ] → [タイトルバー] の [最大化][最小化] のタブがあるので、それぞれ有効にします。すると赤と黄色の最大化、最小化ボタンが出現します。続いて、Mac に合わせて各種ボタンを左側に置きます。

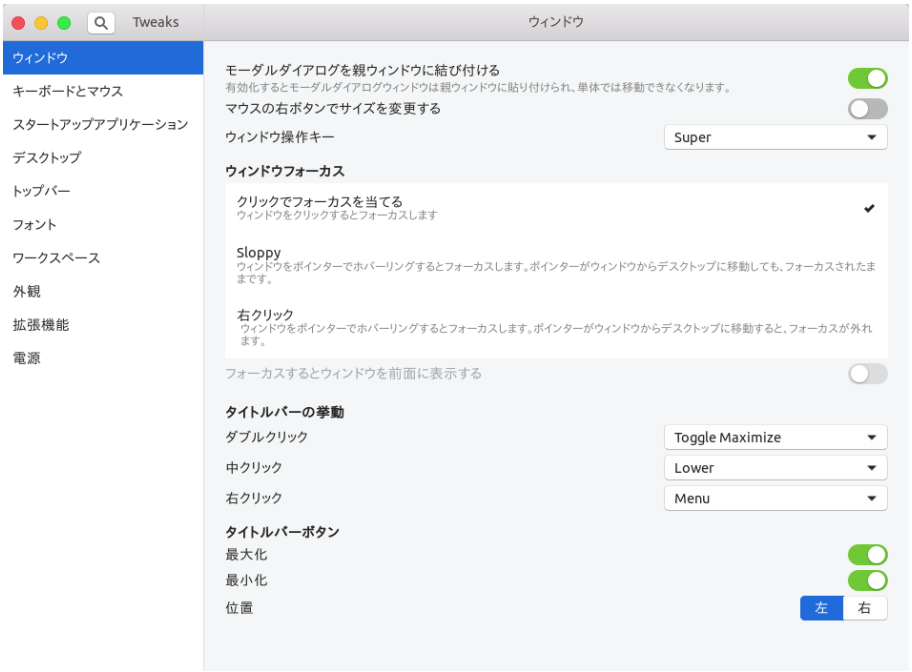

図: 3 色揃うとそれだけで Mac らしい雰囲気になる。

[*5] https://www.gnome-look.org/p/1148692/

第 6 章　ざっぱ〜んを支える技術：ダウンロード編

柴田充也

ざっぱ〜んの総集編 1 と vol.6 では、技術書典 2 に参加するために QR コードを利用した電子版ダウンロードシステムを構築しました。ダウンロードシステムと言っても、あらかじめ頒布数分の QR コードを生成しておき、正しい QR コードからアクセスしていれば PDF をダウンロードさせるというシンプルなシステムです。本記事では将来再び技術書典に参加したときにどういう作りだったか思い出せるよう、未来の自分のためにシステムの詳細を解説します。

6.1 みんなで幸せになりたい

この書籍を購入された方なら、何らかの電子書籍や音楽配信を購入されたことがあるかと思います。そのほとんどは DRM を始めとした著作権管理機構が備わっていたり、ダウンロードにあたって何らかの制限が設けられていたことでしょう。個人的な意見となりますが、DRM を施すことそのものに異論はありません。ただ DRM によって購入者が不便になることは、避けてほしいと考えています。

たとえば特定の OS やプラットフォームでしか使用できなかったり、サービスが終了したらデータも利用できなくなったり、私的利用の範囲内における複製や加工ができなかったりするのは、電子版のメリットを大いに損なっていると感じるのです。だからこそ音楽配信における iTunes Store 以降の DRM 廃止の流れや、技術評論社の Gihyo Digital Publishing[*1] などの電子書籍ストアでも採用されいているソーシャル DRM への移行は、購入者の立場からすると大歓迎でした。

同時に購入者の善意に完全に依存することは難しいと考える、著作権者や著作権者に対して責任を負うコンテンツ配信事業者の気持ちもわからなくはありません。特に後者は「何かあったとき」の説明責任がある以上、「こういう対策を講じていました」と述べる何かが必要になるという意見も理解はできます。特に完全に善意に頼るシステムにおいては、一人でも不埒者が現れると、正規の代金で購入してくれた正直者が馬鹿を見る（もしくはそう感じる人が増える）ことになってしまいます。「The honest will pay, but honestly doesn’t pay」な状態になるのです。そのため、ある程度の管理システムは、購入者の保護にもつながります。結局のところ購入者の利便性と権利者の保護のバランスが重要になってくるわけです[*2]。

正直な話、今回作成するざっぱ〜んのダウンロードシステムは著作権者と配信事業者が一致しているため、極端に購入者の利便性に寄せる（権利者の保護は考えない）実装になるよう調整することも可能ではありました。しかしながら技術者である以上、社会的な要求に即したサービスを構築したいという願望には抗えません。そこでダウンロードシステムを作るにあたって、以下のような要求仕様をもたせることにしました。

1. アカウント登録は不要であること
2. 購入者ごとにダウンロード URL は異なること
3. ダウンロード URL は推測不能であること
4. URL の通知は書面（QR コード）で提供すること
5. ダウンロードに期限を設定できること
6. 期限内であれば無制限にダウンロードできること
7. 厳密な運用は求めないこと

「1. アカウント登録は不要であること」は購入者と販売側の双方の手間を省きます。今回構築するシステムはあくまで技術書典で電子版を販売することが目的でした。つまり恒常的な販売は必要ありませ

[*1] https://gihyo.jp/dp

[*2] 本来の目的は「権利者の保護」ではなく「権利者への適切な対価の提供」であることに注意が必要です。DRM の必要性に関する議論は、著作権管理技術を利用し、不正なコピーを防止することで、権利者が本来得るべき金額を徴収できるという前提に立っています。この前提が正しいかどうかはまた別の話です。

ん。これまでも一般的な電子書籍販売サイトに任せていたためです。よってできるだけステートレスにすることでお互いの手間を省けます。売りたいんじゃない読んでほしいんだ、というイベントにおける頒布を主目的とした同人誌ならではの条件ですね。

「2. 購入者ごとにダウンロード URL は異なること」と**「3. ダウンロード URL は推測不能であること」**は密接に関わっています。ダウンロード URL を購入者ごとに変えることで、不正への心理的なハードルを上げています[*3]。URL が連番のような推測可能なものだと変える意味をなしません。また推測不能な URL は、お行儀の悪いクローラーなどの餌食になってしまう可能性があるのでその点も注意する必要があります。ちなみに購入者ごとに URL を変えることで、ダウンロードされたかどうかの判断が行えたり、ダウンロードできなかったときのようなエラーを追跡するにあたっての情報源として利用できるという副次的なメリットもあります。

推測不能な URL はだいたい長い文字列です。つまり購入者にとってはその URL を間違えずに手入力しなければならないという不利益が発生します。そこで出てくるのが**「4. URL の通知は書面（QR コード）で提供すること」**です。スマートフォンのカメラを用いて QR コードで取得した URL からそのままダウンロードするも良し、URL をメールなどで PC に通知するも良し、利用者の都合の良い使い方をしてもらえます。

「5. ダウンロードに期限を設定できること」は販売側の都合ですね。サーバーの維持費などメンテナンスコストを考えると、未来永劫ダウンロード可能なんて約束はまずできません。よって明示的にダウンロード期限を設けています。今回は一年程度にしましたが、ダウンロード状況を見ているとちょっと長すぎたかなという感じはあります。一年前に買った本のことなんて覚えてないですしね。その代わり**「6. 期限内であれば無制限にダウンロードできること」**にしています。とりあえずダウンロードしてみたもののどこに保存したっけと忘れてしまうことはよくあります。そこで期限内であればいつでもダウンロードできるようにしておき、どこかのタイミングでクラウドストレージなどに保存してもらえればということを考えていました。

最後の**「7. 厳密な運用は求めないこと」**は、同人誌ならではのゆるさです。厳密さを求めれば求めるほどコストは跳ね上がっていきますので、「何かあったら人間がなんとかする」という方向に持っていくつもりでいました。そのためダウンロードカードにもうまくいかなかったときの連絡先として、メールアドレスを記載してあります。

6.2 Nginx の SecureLink モジュール

ざっぱ〜んの電子版（PDF や EPUB）には DRM をつけていません。また購入者ごとの電子透かしを付けることもしていません。単に PDF と EPUB をアーカイブしたものを配布しています。つまり配布だけであればウェブサーバーがあれば十分です。そこで今回は Nginx を使用します。必要な機能は次の二つです。

1. 複数の特定の URL へアクセスされたら同じコンテンツを返す
2. 特定の URL には期限を設定する

1. については Nginx の `rewrite` ディレクティブを使用し、2. については SecureLink モジュールを使用します。

6.2.1 rewrite ディレクティブ

Nginx の `rewrite` ディレクティブ[*4]を用いると、特定の URL へのアクセスに対して URL を内部的に書き換えて、外に公開していない URL の結果を返すことが可能です。「URL を内部的に書き換える」点がリダイレクトと異なるポイントです。

[*3] こういうことをすると「客を疑っているのか」と怒る人や、怒らないまでも不愉快に思う人はそれなりにいるようです。今回のようなシステムを構築する場合は、どう頑張ってもそのような意見を持っている人と良好な関係を築けないので、関わり合わないことがお互いのためでしょう。

[*4] http://nginx.org/en/docs/http/ngx_http_rewrite_module.html#rewrite

リスト 6.1: rewrite ディレクティブの例

```
location /foo/ {
        rewrite ^/foo/bar$ /secure/contents.zip last;
        return 403;
}
location /secure/ {
        root /var/www/internal/;
        internal;
}
```

リスト 6.1 では「/foo/bar」にアクセスしたとき「/var/www/internal/contents.zip」の内容を返す設定になっています。HTTP クライアントからはあくまで「/foo/bar」の結果として見えます。rewrite ディレクティブの書式は以下のとおりです。

```
rewrite マッチングルール 書き換え先 [フラグ];
```

マッチングルールにマッチした場合は、URL を書き換え先の内容に書き換えた上で処理を進めます。フラグに「last」を指定していると、マッチしたらその箇所で処理を止めて、書き換え先の location ディレクティブへとジャンプします。もし「/foo/」で始まるアクセスであるにも関わらずマッチしない場合は、最後の「return 403」が処理されます。つまり「Forbidden」ですね。

「/secure/」では internal ディレクティブを指定しています。これにより「/secure/」には直接サーバーの外からアクセスできない状態になります。もしアクセスした場合は 404 が返ります。

ここまでで異なる URL へのアクセスを一つのコンテンツに集約することができました。

6.2.2 SecureLink モジュール

SecureLink モジュール[*5]を使うと、アカウント登録時などにメールで送られてくる「一定時間なら有効な URL」を簡単に作成できます。

しかしながら SecureLink は、Nginx を「--with-http_secure_link_module」でビルドする必要があります。Nginx のビルドオプションは次のように確認できます。

```
$ nginx -V 2>&1 | tr ' ' '¥n' | grep secure
--with-http_secure_link_module
```

Ubuntu の場合、nginx パッケージをそのままインストールしたら、このオプションがついていないバージョンになります。Ubuntu の nginx パッケージは若干特殊で、同じ Nginx サーバーを提供する異なるパッケージが複数存在するのです。

nginx-light
: HTTP サーバーとして必要最低限の機能のみを有効化した最も小さいバージョンです。

nginx-core
: Nginx の一般的に使われる標準モジュールを一通り有効化したバージョンです。

nginx-full
: Nginx のすべての標準モジュールを有効化したバージョンです。

nginx-extras
: nginx-full に加えてさらに多くのサードパーティモジュールを有効化したバージョンです。

nginx パッケージを指定したら nginx-core パッケージがインストールされます。しかしながら SecureLink モジュールは nginx-etras でのみ有効化されています。よってこのモジュールを使いたいなら nginx-extras をインストールしてください。ちなみにすでに nginx-core をインストール済み

[*5] https://nginx.org/en/docs/http/ngx_http_secure_link_module.html

第 7 章　Flatpak で Ubuntu 16.04 LTS 時代の gedit をよみがえらせよう

あわしろいくや

Flatpak を武器に、ハンバーガーメニューに抵抗するレジスタンスの物語。

7.1 GNOME の方針に対する愚痴

GNOME はデスクトップ環境であり、その役割のうちの一つは各種アプリケーションのルック＆フィールを統一するというものです。確かにそうすることによって全く使ったことのないアプリケーションでも何となく使えるような気がしてきます。

もちろんそんなものは幻想であって、適材適所アプリケーションの役割に合ったユーザーインターフェースが必要に決まっているのですが、そんなの知らねぇというのが現在の GNOME の方針です。たぶん。知らんけど。

実際、多くのアプリケーションは現在の HIG（ヒューマンインターフェースガイドライン）にして使い勝手が上がっているものが多いのですが、gedit は例外です。というか、ほぼすべてのメニューをたどるためにいちいちハンバーガーメニュー（右側に表示される三みたいなアイコン）をクリックしないといけないというのはめんどくさすぎます。おまいら本当に gedit 使っているのかよ。

そんな GNOME の方針はさておき Ubuntu では長らくデフォルトのデスクトップシェルが Unity だったので、GNOME のこのポリシーとは合わないよねということで、旧来のメニューバーを表示するスタイルを維持してきました。そしてそれがアップストリームの gedit にも取り込まれたのですが、いつの間にやらその機能も削られ、Ubuntu 17.10 では Unity 環境下でも gedit はハンバーガーメニューありで起動するようになりました。

GNOME のポリシーに共感しない人たちはどうやら多いらしく、その筆頭は MATE デスクトップ環境で、GNOME 2.x 時代のアプリケーションも含めてまるごとフォークしているという反骨精神満載のプロジェクトです。もちろん gedit もフォークしており、pluma[*1] という名前になっていますが、gedit はフォークしたものの gedit-plugins はフォークしていません。よって機能が足りません。Linux Mint が始めた Xed[*2] というエディターもありますが、これは pluma の機能削減版です。一体何がしたいんだ Linux Mint よ。

結局現状だといい感じに gedit を使用する方法がないということになります。

7.2 Flatpak と snap

最近 Flatpak や snap の話をよく耳にします。これらは新しいパッケージディストリビューションの仕組みで、次のような特徴があります。

- Linux ディストリビューションやバージョンに依存しない
- 既存のパッケージシステムに影響を与えない
- サンドボックス環境で動作する

ぱっと思いつく Flatpak と snap の共通点はこのくらいで、あとは全然別物です。というか本来は同じジャンルに分類してもいいものなのかよくわかりません。Flatpak はデスクトップアプリケーションのみに対応しますが、snap はもっと広くシステム全体に使用できますし、世代管理ができるのも snap だけです。サポート状況も、snap は Canonical がやっているだけあって Ubuntu では乳母日傘（死

[*1] https://github.com/mate-desktop/pluma
[*2] https://github.com/linuxmint/xed

語）ですが、Flatpak は塩対応です。Fedora では当然逆ですし、Debian では Flatpak のほうがアクティブにメンテナンスされているようです。どうでもいいですけど塩対応ってネットスラングではないのでしょうか。

どちらもそれなりに使えるようになってきているので、じゃあ Ubuntu 16.04 LTS 時代の gedit 3.18 をどちらかでビルドすればいいのではないだろうかと思いついて、まずは Ubuntu なのでということで snap で挑戦したものの、どうやってもメニューを日本語にできなくて断念しました。もちろんほかにもいろいろ問題はあったのですが。というわけで今回は snap は全く関係ないのですが、検証した結果としては既存のパッケージを snap パッケージに組み込むことができるのは一長一短あります。あとは日本語メニューにすることすらもできないほどの強力なセキュリティというかサンドボックスは本当に必要なんでしょうか。ほかにもデスクトップアプリケーションではいろいろ厳しいです。

一方 Flatpak では概ねいい感じにできたので、ここでは getit 3.18 を Flatpak でパッケージにする方法を紹介します。ついでに自分にとって便利なようにいくつかのカスタマイズを加えています。

余談ですが、検索していると「A Snap of Gedit 3.10 for the toolbar fans」という GitHub のリポジトリ[*3]が見つかり、作者は Will Cooke[*4]です。古い形式なので今の snapcraft ではビルドできないはずであり、現在も継続して利用されているとは思われませんが、やはり同じことを考える人もいるのだなということでここで紹介しました。

7.3 ビルド方法

解説は後回しにして、まずはビルド方法を紹介します。ビルド環境は Ububuntu 17.10 を想定しています。18.04 LTS 以降の場合は最初の PPA 追加は必要ありません。

```
$ sudo add-apt-repository ppa:ikuya-fruitsbasket/flatpak
$ sudo apt install flatpak-builder
$ sudo flatpak remote-add --if-not-exists flathub https://flathub.org/repo/ flathub.flatpakrepo
$ sudo flatpak install flathub org.freedesktop.Platform//1.6 org.freedesktop.Sdk//1.6
$ sudo flatpak install flathub org.gnome.Sdk//3.26 org.gnome.Platform//3.26
$ git clone https://github.com/ikunya/flatpak-gedit318.git
$ cd flatpak-gedit318
$ flatpak-builder gedit org.gnome.gedit.json
```

ビルドした gedit を試しに起動してみる場合は、次のコマンドを実行してください。

```
$ flatpak-builder --run gedit org.gnome.gedit.json gedit
```

インストール方法は後ほど解説します。

7.4 解説

Flatpak 版 gedit のパッケージはすでに存在しており、今回はそれをカスタマイズしていくことにします。一旦前述のビルド方法については忘れて、まずは初期セットアップから行うことにします。

```
$ sudo add-apt-repository ppa:ikuya-fruitsbasket/flatpak
$ sudo apt install flatpak-builder
$ sudo flatpak remote-add --if-not-exists flathub https://flathub.org/repo/ flathub.flatpakrepo
$ sudo flatpak install flathub org.freedesktop.Platform//1.6 org.freedesktop.Sdk//1.6
$ sudo flatpak install flathub org.gnome.Sdk//3.26 org.gnome.Platform//3.26
$ git clone https://github.com/ikunya/flatpak-gedit318.git
$ cd flatpak-gedit318
```

[*3] https://github.com/8none1/gedit310
[*4] Canonical 社員でデスクトップ部門の偉い人

元にした設定ファイルはリポジトリに入れてあるので、何はなくてもビルドしてみます。

```
$ flatpak-builder gedit org.gnome.gedit.orig.json
```

すると次のようなエラーを表示して止まります。

```
error: org.gnome.Sdk 3.24 not installed
Failed to init: Unable to find sdk org.gnome.Sdk version 3.24
```

ここで使用している GNOME の SDK はバージョン 3.26 なので、次のような変更を加えます。

```
@@ -1,7 +1,7 @@
 {
     "app-id": "org.gnome.gedit",
     "runtime": "org.gnome.Platform",
-    "runtime-version": "3.24",
+    "runtime-version": "3.26",
     "branch": "stable",
     "sdk": "org.gnome.Sdk",
     "command": "gedit",
```

2 回目以降にビルドする場合は次のコマンドに変更してください。

```
$ flatpak-builder --force-clean gedit org.gnome.gedit.orig.json
```

これでビルドが完走するはずです。
続いて gedit-plugins を組み込んでみます。

```
@@ -72,6 +72,16 @@
                     "sha256": "aa7bc3618fffa92fdb7daf2f57152e1eb7962e68561a9c92813d7bbb7fc9492b"
                 }
             ]
+        },
+        {
+            "name": "gedit-plugins",
+            "sources": [
+                {
+                    "type": "archive",
+                    "url": "https://download.gnome.org/sources/gedit-plugins/3.22/gedit-plugins-
3.22.0.tar.xz",
+                    "sha256": "83a73088de73478841b9a216bd89c2e478aa302b3579a2a8685893d7a6a48fdc"
+                }
+            ]
         }
     ]
 }
```

SHA256 のハッシュがあるのがポイントです。自分でダウンロードして生成してもいいですが、ダウンロード元のフォルダーに gedit-plugins-3.22.0.sha256sum というファイルがあり、ここに書かれているのでこれをコピーするのが簡単です。
gedit 3.22 でビルドできたので、3.18 に切り替えます。

```
@@ -68,8 +68,8 @@
             "sources": [
                 {
                     "type": "archive",
-                    "url": "https://download.gnome.org/sources/gedit/3.22/gedit-3.22.1.tar.xz",
```

「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

vol.6（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/179274142.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.7【体験版】

2018 年 2 月 11 日　v1.0.0 発行
著　者　team zpn
発行所　team zpn

1. OpenNebulaでNextcloudサーバー構築
2. サーバーレスでオレオレ画像アップローダーを作る
3. BuildStreamでGNOMEアプリをビルド
4. LibreOffice 6.0 Writerで縦書き小冊子を作成する
5. UbuntuをmacOS High Sierra風に変えてみる。
6. ざっぱ〜んを支える技術：ダウンロード編
7. FlatpakでUbuntu 16.04 LTS時代のgeditをよみがえらせよう

発行 ● team zpn