
じょしだいせい
JDと学ぶ
OpenStack
入門編

ここはどこかの大学の研究室。クラウドコンピューティングサービスについて研究しています。
研究室に訪ねてきた男子大学生のあなたが主人公です。

Keysotne（操作ユーザーの認証・認可を担当）

研究室の准教授。永遠の 17 歳。
研究室メンバーを選んだ張本人。
先生の機嫌を損ねると研究室から締め出される。
「あらようこそいらっしゃい♪　あなた専攻は何かしら？」

Horizon（OpenStack を操作する GUI）

keystone 先生が飼ってるワインレッドの猫。
先生の胸の谷間におさまるのがお気に入り。
先生の指示棒でお腹を撫でるとディスプレイにトランスフォームする

Nova(仮想マシンの管理)

本作のメインヒロイン。3 年生。
ボケる Cinder を突っ込むも今ひとつ。
後ろでいつも先生に苦笑いをされている。
面倒見がよく、後輩の Glance ちゃんをかわいがっている。
「ちょっと Glance！カタログより先にお茶出しでしょ！もう 」

Glance（仮想マシンで使える OS を管理）

研究室に最近入り浸るようになった、怖いもの知らずの 2 年生。
肩にかけたバッグには OS カタログがびっしり。
研究室に来るお客様に
「ねえねえ見て見て！どれにする？？」
とカタログを開いて迫ってくるけど選んだら選んだで
「ほんとにそれでいいの・・・？」と上目遣いであざとく聞いてくる。

Cinder(仮想マシンにボリュームを追加)

世間知らずの天然さん。3 年生。
お金持ちのお嬢様。お家は広い宮殿。
研究室の備品は全て彼女の提供品。
「最近おこづかいきつくて・・・バイト増やさないと。。。」
「え？どうしてじいやに頼まないの？」
「普通の家にはいないんだよ Cinder ちゃん・・・」

Neutron(ネットワーク接続を管理)

主人公の友人関係までバッチリ把握済みの情報通。3 年生。
ケンカは見ているだけのことが多いけど、時々鋭いことを言う。
「Glance ちゃんも Cinder ちゃんもいいかげんにしなよ？
ふたりとも Nova ちゃんのこと好きなだけだよね？見てたらわかるよ？」
しかし男関係についてはどうやら来る者拒まずの清楚系ビ○チ。
「こないだの彼？とっくに別れたわよ、3 人と浮気してるの見えちゃったんだもん」

— 見た目も性格もバラバラな 5 人だけど、やりたいことに手をあげたら、Openstack ができました。

第 1 章

はじめに

OpenStack という単語を聞いたことがある人が、この本を手に取って頂いたのだと思います。あるいは表紙がかわいいから、という理由かもしれません。

11 月某日にみさきさんという女の子に出会いました。その子にコミケ出展の話を偶然したことをきっかけに、今回かわいい表紙や挿絵を書いてくれたくろろろろさんという絵師の女の子にも出会うことが出来ました。くろろろろさんの絵はラフの段階でとても素晴らしい出来でした。この 2 人がいなければ、このような形で本書は完成していなかったでしょう。「縁に気づいて、縁を生かす」、という言葉があります。本書はまさにその言葉通りの行動の結果だと思います。まずは、みさきさんとくろろろろさんに感謝の意を述べたいと思います。心から「ありがとう」。

上述に加え、メンバーはあと 2 人います。実際のところ、このサークルのエンジンとなったのは、作者であるわれわれ 2 人ではなく、そのメンバーでしょう。やたら盛り上がりにかける編集長を締め切りのようなもので殴った犯人達です。テンション低めですみませんでした。当日は一緒に本を売ってくれる予定です。ありがとうございます。

この本は以下の内容で構成されています。第 2 章からは OpenStack の概要、第 3 章は devstack、第 4 章は VM が起動するまでの旅路を、第 5 章は周辺技術について話ができたらと思います。やたら長い後書きは、同人誌にはよくあることです。

ググれば分かることはそちらに任せ、それとは少し違う、かゆいところに手が届く内容にしたいと思います。OpenStack に関する本はまだまだ少ないと言えます。ネット上の情報も十分とは言えないでしょう。それにしては、簡単に試すには酷なソフトウェア群です。これはつまり、これでお金を動かすことができるということです。それにも関わらず、この本には複数の方の協力を得ました。給料は出ません。有り難い話です。この本を真に受けてしまうのはおすすめしませんが、ネットを調べてもなかなか載っていないだろう情報が、多少はあると思います。

偶然かどうかわかりませんが、この本を編集している時に、さきほどの引用にある本を読んでいました。OpenStack は、長らく職人の世界であり、良くも悪くも雇用を守ってきたインフラの聖域に少し入り込んだ、新参者です。だからこそ流行し、だからこそ流行させられようとし、そして嫌われ始めています。OpenStack の Alternative が発生するのは時間の問題でしょう。ただしそれは、貴方が手に取った「本」に対しても、かつて同じことが起こっていたと思います。いつだって最後には、些細なことです。

これはかつてクラウドと呼ばれたインフラに関する事柄、あるいはたった一つの冴えたやりかたの存在を考えるシリーズの、第一巻となる予定です。Terraform や KVM と、書かれるべきものはたくさんありますが、執筆者のみでは不可能であり、まず、他のメンバーにお伺いをたてなければなりません。

第 2 章

OpenStack ことはじめ

2.1 OpenStack 以前 クラウドのベンダーロックイン問題

近年、クラウド市場は大きな伸びを見せています。特に日本では、2011 年に AWS（Amazon Web Services）が Tokyo リージョンを開設してから日本においてもクラウドに注目する企業が増えていたりします。しかし、それと同時にさまざまな問題も露呈しはじめています。その一つがパブリッククラウドのブラックボックス化です。企業にとってクラウドを利用する目的は、例えばオンプレミス環境からの脱却を図るためであったりするため、利用しているサービスがブラックボックスであっても、一見特に問題ないように思えてしまいます。

しかし、サービス事業者は独自の技術や手法を用いて構築したサービスを推進して、当然他事業者との差別化を図る戦略を取ろうとします。もちろん、サービスの高度化をもたらしユーザのビジネス拡大につながる可能性があるんですが、ユーザの立場から見ると、システムの再構築や改修、他事業者への移行が容易ではない、あるいは、利用コストが掛かり過ぎる、将来的なコスト予測がつきにくい、といった問題を引き起こす可能性があるということです。わがままなもんです。幸運が三度姿を現すように、不運もまた三度兆候を示すそうです。見たくないから見ない、気がついても言わない、言ってもきかない。そして破局を迎える。

これをクラウドのベンダーロックイン問題と呼びます。この問題が露呈するのきっかけとなったのは、Google App Engine の正式サービス化でしょう。正式サービスに伴い 2011 年 9 月に、実質的な大幅値上げが行われることが判明しました。その値上げに対して、当然ユーザからの反発を招いたが、それと同時に、迅速な他のクラウドへの乗り換えが困難であることが露呈、その結果さまざまな議論を巻き起こしました。特に Google App Engine は PaaS であるため、その環境でのみ動作するコードが必要となります。そのため、他のクラウドに移行するには膨大なコード改修に伴う工数とコストがかかってしまう。こういったリスクは、アプリケーションのコードレベルだけではなく IaaS であればより低レイヤーでの互換性など、様々な側面で発生します。

クラウドサービスはまだまだ標準化もオープン化もされていません。この状況は、かつて 80 年代オープン化推進以前のベンダー囲い込みに酷似した状況です。その後 90 年代に花開いたオープンシステムは 20 年の時を経て幼年期の終わりを告げました。さて、2010 年代のクラウドサービスはどうなることやら……。

2.1.1 国内事例

2014 年 2 月 13 14 日に開催された OpenStack Days Tokyo 2014 では、グリー、Yahoo! JAPAN、NEC がぞれぞれ自社の OpenStack 導入事例を公開しました。

Yahoo! Japan: 内製クラウド環境からの移行

ショッピング、ヤフオク、知恵袋、トラベル、不動産、ブックストア、ゲームなどが利用

GREE: 大規模ゲーム基盤として採用

運用効率化のため物理サーバの提供から仮想環境の提供へシフトするために OpenStack を選択

NEC: NEC が提供する NEC Cloud IaaS というパブリッククラウドサービスとして提供

スタンダード版で OpenStack が活用されているハードウェアには NEC データセンター専用省電力高集積サーバを使用

また、上記に加えて GMO インターネットがすでに事例を出しています。

GMO インターネット:

Conoha VPS サービス、PaaS サービスの基盤として採用

海外では、CERN、eBay、Yahoo!、AT&T などがすでに事例を出しています。これら事例から以下のことが重要であると考えられます。

- 大規模クラウドを実現するための拡張性
- SS による高品質サービスの実現

毎年、5 月と 11 月に開催されている OpenStack Summit では、世界中のエンジニアが様々なトピックについて議論を繰り広げています。特に、大規模なクラウド環境の構築や高可用性のある設計について、白熱した議論が行われています。エンジニアが会社を跨いで、他社の事例や取り組みを入手でき、かつ議論できることは、OSS ベースである OpenStack を利用することの魅力の一つだと考えられます。ホントかな……。

2.1.2 オープンクラウドエコシステムと幼年期の終わり

選択肢は多様性を産みすぎる

ほんの数年前には、プライベートクラウドを構築するソフトウェアは、Eucalyptus、CloudStack、OpenStack などいくつかあった。Eucalyptus は AWS との API 互換性で、CloudStack は黎明期から商用に耐えうる安定性に、OpenStack はオープンソースのあり方とそのコミュニティ/マーケティングの手法により、三者三様シェアを取り合いクラウド戦争を起こしていた。そう、2012 年には確かに勝者はわからなかった……。

オープンクラウドは幼年期の終わりを告げる

この時代のクラウドソフトウェアはまだどれもベンダーの囲い込みがあった。OpenStack だけは、囲い込みはなく、ただ単純に安定性の欠如の問題があったのである。初期のユーザは主にパブリッククラウドの運営会社だ。だから数年前までは安定性のある CloudStack のシェアが一歩リードしていた。状況が変わったのは 2013 年だ。そのころから OpenStack の安定性が劇的に向上した。なぜなら、RedHat や HP など大手 OSS 企業のコントリビューションが増えたからだ。そこで、CloudStack はより一層商用サポートと言う名のベンダー囲い込みをアピールしていった。しかし、CloudStack は洗濯を間違えた。なぜなら、ユーザが求めていたのはベンダー囲い込みからの自由であり、システムの主導権をユーザ自身が握ることである。また、このタイミングからクラウド事業者ではなく Yahoo! Japan、GREE といったウェブ企業が OpenStack を利用するようになった。彼らが求めていたものは、やはり機能や性能ではなかった。結局、OpenStack とそれを取り囲むエコシステムが提供するポータビリティーやベンダーからの自由、独立性などを求めていたのである。こういったユーザのニーズにオープンソースの本来のあり方を貫いた OpenStack がハマり、結果クラウドソフトウェアの勝者となったのである。この状況は、15 年前の Lunux への移行時期の状況と非常に酷似している。

単にクラウドソフトウェアをう変えれば良かった、という幼年期の時代から、これからはクラウドソフトウェアであろうとも自由という独立性を根幹とした OpenStack を中心としたエコシステムによる製品やサービスが提供される時代、これをなんと呼べばよいかわかりませんが、いまそういう時代へ明らかにシフトしているのだと考えられます。

2.2 OpenStack コンポーネント

OpenStack は複数のコンポーネントにより構成されており、その各々のコンポーネントが特定の機能に特化しています。ユーザは、目的に応じて必要なコンポーネントをインストールすることで柔軟な設計をすることが可能です。2014 年 4 月にリリースされた Icehouse というバージョンでは 11 の主要コンポーネントが存在します。以下のその各コンポーネントの中でも特に重要な 5 つのコンポーネントについて説明します。最初は名前と顔が一致しなくて、本当に困りました。ここで紹介するのはあくまでコアとなるコンポーネントのみです。他にもたくさんのコンポーネントが存在し、開発中なのですが、どれも名前とサービス内容が一致しなくて困ります。

2.2.1 Compute Service Code-name: Nova

Nova は、コンピュートリソースの運用とプロビジョニング自動化のために設計されています。つまり、Nova 自身はハイパーバイザーそのものではなく、仮想マシンの管理ソフトウェアということです。加えて、Nova は仮想化技術とし KVM だけではなく Xen、VMware ESX(i)、Hyper-V、それにベアメタルをサポートします。コンピュートリソースは、開発者に対して API を通して、また管理者やユーザに対して Web のインターフェースを通して利用可能です。コンピュートの設計は、コモディティ化された物理サーバ群を利用しスケールアウトすることが可能です。Icehouse からは Nova 自身のローリングアップグレードが可能となっており、Nova それ自身を稼働中の仮想 VM をシャットダウンさせることなく新しいバージョンにアップグレードすることが可能となっています。また、Nova の API は AWS の EC2 の API と互換性を保っています。

2.2.2 Block Storage Service Code-name: Cinder

Cinder を利用することでブロックストレージの管理が可能になります。それにより各インスタンスに対してストレージ領域の拡張を可能にします。具体的には拡張用ブロックの作成や作成したボリュームのアタッチ、デタッチを管理します。その他 Cinder の主要な機能としては、伝統的なエンタープライズの共有ストレージシステムを統合管理することです。例えば、Linux NFS、LVM、Ceph、NetApp、Nexenta、SoliddFire、EMC、3PAR などがその対象です。これは AWS の EBS と同等の機能を提供するサービスと考えることができます。

2.2.3 Networking Service Code-name: Neutron

Neutron は OpenStack に対してネットワークに紐付いたサービスを提供します。これは他のコンポーネントに比べて特に API-Driven で開発されているシステムであり、かつ管理者やユーザは様々なタイプのネットワーク (例えば、Flat Networks、VLANs、VPN など) を API を使用してオンデマンドでカスタマイズすることが可能です。また、専用の IP アドレスや Floating IP アドレスの提供も可能です (特に後者はメンテナンスは障害時にトラフィックの経路変更をするときに役に立ちます)。また OpenFlow SDN やプラグイン形式でロードバランサーサービスやファイアーウォールサービスの提供もサポートしています。

2.2.4 Image Service Code-name: Glance

クラウドプラットフォームの鍵となるメリットの一つはユーザが好きなときにすぐに仮想マシンを立ち上げることができることです。Glance は仮想マシンのテンプレートを作成すること、かつ仮想マシンの再作成やコピー、スナップショットの取得でこの役割を助けます。つまり、管理者は、ユーザが自身でプロビジョニングす

るときに選択する仮想マシンのテンプレートのカタログを設定するだけでよいということです。また既存イメージのバックアップや保存にも活用され、かつ、イメージの保存先として Cinder と連携することが可能です。

2.2.5 Identity Service Code-name: Keystone

Keystone はユーザ認証とロールベースのアクセスコントロールを OpenStack に提供するためのコンポーネントです。Keystone は LDAP と連携することでユーザコントロールやポリシーの設定が可能になります。また、伝統的な ID/Password によるログインに加えてトークンベースのログインも可能です。

2.3 OpenStack を動かしてみる

というわけで、OpenStack をセットアップしてみましょう。念入りに準備をして、その準備から外れたことはしない人いますけど、それが責任感とでも思っているんですかね。何でもやってみなくちゃ始まりません。なんと deb でも rpm でもパッケージが提供されています。ディストリビューションのリポジトリを探してみましょう。そのパッケージをインストールすればよいのです。できましたか？設定ファイルなどを書き直す必要はありますが……まぁパッケージだしデフォルトでそれなりに動くようになってはいるでしょう。じゃあ起動してみましょう。

はい、できませんね。僕もできませんでした。何を起動するの？ってか何をしたら OpenStack をセットアップしたことになるの？というかそもそも、何をインストールするの？

”openstack”で探してみました？前章で出てきた名前のパッケージが見えますね。どれをインストールしました？いや、その前に AMQP インストールしました？ついでに、そもそもなんですが、あなたのカーネルって KVM モジュール効いてます？って実は無くても動くんですけどね。あ、libxml-devel が必要かもしれません。どこにも書いてないけど。まぁまずはドキュメントを読んでみましょう：

http://docs.openstack.org/

の、どれだよ、って？しょうがないなぁ Fedora と Ubuntu のを教えてあげよう：

- http://docs.openstack.org/juno/install-guide/install/yum/content/
- http://docs.openstack.org/juno/install-guide/install/apt/content/

詳細なドキュメントですね。これをすべてやらないと OpenStack は動きません。読んでない人のために、ここにまとめを書いておきましょう。三行で言うと：

1. この手順では
2. Linux が
3. 3 台必要

よろしい。ならば戦争だ。

大きな物語を失った若者が右傾化すると言われる今日このごろですが、そんなあなたに devstack というものがあります。OpenStack を試してみたい場合に便利です、という売り文句ですが、最新のソースコードを元にセットアップするため、OpenStack そのものの開発環境を作る際にも便利なものになっています。中でやっていることは：

- OpenStack ソースコードの取得
- 依存パッケージのインストール
- 初期設定

- テストデータの投入
- 起動・終了

となっています。devstack 自体はシェルスクリプトの集合体なのですが、コマンド一発で OpenStack がそれなりに動作する環境を作ることができます。設定ファイルを編集することで動作を変えることもできます。早速試してみましょう。

2.4 devstack を使ってみる

まず Linux が必要です。devstack のコードを眺めてみると、サポートされているディストリビューションは debian 系・Redhat 系・SUSE 系のようですが、ここは Fedora 21 で試してみましょう。執筆時点では Ubuntu 14.10 はサポート外のようでした。ちょっとコードにパッチをあてれば動く事には動くのですが、VPNaaS（VPN as a Service）を構築する際に必要になる openswan というパッケージが Ubuntu 14.10 のソースに無かったりして、力を出し切ることはできなさそうでした。Fedora は 21 から Server・Workstation・Cloud とそれぞれに特化したバージョンを公開していますが、どれでもよいと思います。今回は Fedora Server 21 beta を使用しました。GNOME 3 採用の時ほどにはあまり知られていないのですが、yum が廃止の方向でして、代わりに dnf というパッケージ管理システムが導入されています。まぁ結局扱うのは RPM なんですけどね。なんですけど、Server は yum で Workstation は DNF なんですよね。Server で dnf って打ったらそんなコマンド無いって言われたよ。

お試しで動かす場合に SELinux は邪魔でしかありませんので、Disable、は負けを認めた感じになりますので、Permissive にしておきましょう

```
vim /etc/selinux/config
reboot
```

OS のインストールとセットアップに関しては割愛しますが、スペックは大きめにしておく必要があります。僕は 4GB RAM + 4 Cores + 32GB Btrfs で開発していますが、割とギリギリです。最低限のコンポーネントを起動させておくには問題ありませんが、devstack がサポートするすべてのコンポーネントを同時に起動させておくには 8GB ほどメモリを用意しておくほうがよいでしょう。

OS のインストールが終わったら、まずは devstack を取得してみましょう。devstack は残念ながらパッケージでは用意されていません。git clone してくる必要があります。

```
git clone https://github.com/openstack-dev/devstack.git
```

え？ git が無いって？貴方 Ubuntu ユーザーですね？ Ubuntu はいつだってそうでした。

```
apt-get install build-essential
```

Fedora では以下のパッケージをインストールしておくと後の作業が楽でしょう。

```
yum group install 'Development Tools'
```

あ、そもそもネットワークが繋がってないです？まぁネットワークアクセスが無くても OpenStack の意義はどこかにある気もしますが、そうでないなら、さっさと Lan ケーブルなり WiFi なりにつなぎましょう。DHCP でもいいです。それでも動きます。

devstack ディレクトリを ls してみましょう。ええ、何が何だかわかりませんね。ドキュメントには localrc を書いて./stack.sh しろって言ってますが、all_in_one とそうじゃないのとが書いてあったり、VM と Machine で違ったり、何か IP レンジ決めろとか書いてあるし、やる気なくなりますね。でも大丈夫、必要ありません。しかし、設定ファイルは必要です。それが localrc というファイルです。ありますか？ありませんね。そう、自分

で作らないといけません。ひな形ぐらい用意しておいて欲しいですね。例えば以下のように書いて保存してみましょう：

```
disable_service n-net
enable_service neutron q-svc q-agt q-dhcp q-l3 q-meta
```

この設定が何かと言いますと、nova-network を無効化して、代わりに Neutron を有効化しています。勘のよい方は気付くかと思いますが、そうです、これはシェルスクリプトです。disable_service・enable_service は Bash の関数として定義されています。

暗号かよ。disable/enable はなんとなくわかるけど……それなら enable_neutron とかじゃないの？ enable_service neutron とか？

いい質問ですね。いろいろ回答できるいい質問です。かつて、VM のネットワーク管理をする部分は nova-network として nova の一部分として組み込まれていました。ところが、ネットワーク大好きレガシーおじさん達のせいで、どんどんネットワーク周辺のコードが肥大化していくことが見え透き、ついに nova の一機能から quantum という名で独立し、個別にネットワークの機能を拡張していくことになりました。その quantum も成熟し、ついに OpenStack の中核を成すコンポーネントに成長し、こうなるってくると急に「実は Quantum って商標なんですよねー出すもん出してもらわないとーチラッチラッ」みたいな話があったかなかったか、Neutron と名を変え、OpenStack 内屈指の難解コンポーネントになったのでした。

もうおわかりですね。n-net は nova-network のことです。これは古き良きものなので disable します。かわりに enable_service neutron します。neutron を構成するサブコンポーネントの起動もします。q-*がそれです。q とは？そう、旧名の quantum のことです。え、n じゃねーの？だってそりゃ、n じゃ nova と被るじゃん。

……やってられませんね。人間味溢れるシステム、それが OpenStack なのです。初心者お断りです。さて上の 2 行を書いたらあとは./stack.sh するだけです。それだからいけないのか、それだけのことだからいけないのか。

2.4.1 devstack を動かす

ところで devstack が何をやっているかと言いますと、OpenStack の公式 Git リポジトリからソースコードを Clone してきて、必要な Python パッケージのインストールをし、設定ファイルを自動生成し、コンポーネントを順番に起動していきます。起動したら、テスト用の初期データの投入や、テスト用の OS イメージファイルなどのダウンロードとそのイメージの登録をします。これで OpenStack が何となく動く状態にまで持っていってくれます。

先ほど localrc も書いたし、とりあえず devstack を実行してみましょう。pwd を devstack にして：

```
./stack.sh
```

おっと延々とメッセージが出はじめましたね。それぞれが何をやっているかはあとで説明するとして、ちょっと待ってみましょう。プロンプトが出て、何か聞かれませんか？まずはこれでしょう：

```
################################################################################
ENTER A PASSWORD TO USE FOR THE DATABASE.
################################################################################
This value will be written to your localrc file so you don't have to enter it
again.  Use only alphanumeric characters.
If you leave this blank, a random default value will be used.
Enter a password now:
```

OpenStack は必要なデータを保存しておくシステムとして、データベースシステムを必要とします。MySQL でも sqlite でも postgresql でも（たぶん）いいのですが、devstack は MySQL を使おうとします。つまり、勝手にインストールします。そして、勝手に起動します。（自動起動はしない、と思う。）ます聞かれたのは、このデータベースの root パスワードです。次はこれ：

```
################################################################################
ENTER A PASSWORD TO USE FOR RABBIT.
################################################################################
This value will be written to your localrc file so you don't have to enter it
again.  Use only alphanumeric characters.
If you leave this blank, a random default value will be used.
Enter a password now:
```

出ましたねラビット。これが何かと言いますと、DBus みたいなもんだと僕は思っていますが、教科書的には「キュー」です。「メッセージキュー」とも言います。

OpenStack はさまざまなコンポーネントやプロセス、あるいは上述の RDBMS や MongoDB のような別のデータストアを有機的に組み合わせて稼働するシステムです。このキューはそれらプロセス間の通信を仲介する役目を担います。それぞれのプロセスは相手の様子を鑑みず、どんどん相手に要求を差し込むため、これを取り持ってキューイングしておく必要があります。相手が要求を受け入れられる状態にあるならば送りますし、暇な奴がキューから自律的に取り出して処理したりします。

と、それはそうなのですが、このキューの本当の利点は、実はこの点においてではありません。このキューがあることで、送信者と受信者の数を自在に増減することができるのです。ものすごい量の VM 作成依頼が急に来たらプロセスを増やし、逆に増やしすぎて油を売っている状態になっていたら送受信者を減らすこともできます。受信者だけ増やすこともできます。つまり、システムの強度を柔軟に変更することができるようになるのです。年末だからって予算を使い切らないといけない、なんてこともありません。とりあえず今必要な分だけで構築し、困ったら少しづつ増強していく運用スタイルをすることができます。

素晴らしい組織体系ですね。あなたの会社はどうですか？まぁどっちにしても、そのような予算の組み方に依存して生活を営んでいる人で構成された国ですので、それを守るのが正義なんじゃないですかね。まぁいいや。次はこれです：

```
################################################################################
ENTER A SERVICE_TOKEN TO USE FOR THE SERVICE ADMIN TOKEN.
################################################################################
This value will be written to your localrc file so you don't have to enter it
again.  Use only alphanumeric characters.
If you leave this blank, a random default value will be used.
Enter a password now:
```

これから以下は、デフォルトのパスワード設定です。ADMIN TOKEN は管理者用の操作を行うための特別パスワードみたいなものです。次もパスワードを聞いてきています：

```
################################################################################
ENTER A SERVICE_PASSWORD TO USE FOR THE SERVICE AUTHENTICATION.
################j###############################################################
This value will be written to your localrc file so you don't have to enter it
again.  Use only alphanumeric characters.
If you leave this blank, a random default value will be used.
Enter a password now:
```

各コンポーネントは OpenStack システムを担うサブシステムですが、それでもその連携には OpenStack のエンドユーザーと同様の認証と認可が必要になります。つまり、各コンポーネントもユーザー名と正しいパスワードを持っている必要があります。しかし、さすがに普通のユーザーではなく、ロールが違います。各コンポーネントは service という特殊なロールを割り当てられており、Keystone による認証と認可を受けます。というわけで、サービスロールを持つ OpenStack コンポーネントユーザーのパスワードをここで決めます。で、最後は：

```
################################################################################
ENTER A PASSWORD TO USE FOR HORIZON AND KEYSTONE (20 CHARS OR LESS).
################################################################################
This value will be written to your localrc file so you don't have to enter it
again.  Use only alphanumeric characters.
If you leave this blank, a random default value will be used.
Enter a password now:
```

これは OpenStack お試し環境のための、テストユーザー用のパスワードです。

たくさんパスワードありましたね。まぁとりあえず全部 password とかにしておきましょう。あぁ、そういえば、ちょっと前に自分の目標をパスワードにすると生活が捗るっていうのを Facebook でみかけましたね。そんなことが普通になる世界を想像してみましょう。ああ、なんというユートピア。

さて、必要なパッケージのインストール後、OpenStack のリポジトリから各コンポーネントのソースコードが git clone されていくのが見えると思います。その後、各コンポーネントが起動され、最後にテストデータが次々と投入されていきます。これが終われば、devstack は皆さんを以下の言葉で優しく包んでくれるでしょう：

```
Horizon is now available at http://192.168.226.131/
Keystone is serving at http://192.168.226.131:5000/v2.0/
Examples on using novaclient command line is in exercise.sh
The default users are: admin and demo
The password: password
This is your host ip: 192.168.226.131
2014-11-25 13:48:02.914 | stack.sh completed in 1159 seconds.
```

動きましたね。地獄へようこそ。

2.5 devstack で OpenStack を使ってみる

では最期の devstack からの言葉の通り、ブラウザで localhost にアクセスしてみましょう。（先ほどの結果では 192.168.122.13）ログイン画面が出ます。ユーザー名とパスワードは devstack のログに書いてあります。まぁとりあえず admin ユーザーで入ってみましょう。

ログインできましたか？ここまでくれば、あとは Google 検索しながら操作できるかと思います。というより、ここまで来てようやく、OpenStack の仕組みや使い方を調べることが意味を持ちます。ここまで来た後の情報はネットに多く転がっているのです。地獄って 270 個くらいの苦しみがあるみたいですが、全部こなすと、逆に地獄に住めるようになるらしいですよ。

2.5.1 devstack 動いてる？

動いてる？見てみましょう。新しい端末を開いて：

```
screen -x stack
```

お、何か起こりましたね。screen です。screen から説明するのはめんどくさいのでやめます。devstack では OpenStack の各コンポーネントを、この screen の中で動かすことにより、ログの表示や再起動をしやすくして

います。./stack.sh 時に stack という名前でセッションを作成するので、そのセッションを attach すればよいというわけです。ちょっと ps してみましょう：

```
ps aux -H
```

たくさんのプロセスが screen から生えているのが分かると思います。screen の中で起動し、ログはその各スクリーンに標準出力されます。何かコードを変更し、その動作を確かめるときには各スクリーンでおもむろに Ctrl-C でプロセスを Kill し、上カーソルで同じ起動コマンドをもう一度叩けば、再度プロセスが起動します。開発にはとても便利な環境です。試しに Horizon のログを見ながらブラウザから Horizon を操作してみましょう。ログがどどっと流れていき、通信される他のコンポーネントのログも同時に流れているのが見えると思います。

また、他にも多くのサービスが知らない内に起動していると思います。MySQL や RabbitMQ、OpenVSwitch などだと思いますが、これらは OpenStack を動かす際に必要となる外部システムです。MySQL はメタデータの永続化、RabbitMQ はメッセージキュー、OpenVSwitch は Neutron が依存しています。これらも./stack.sh 時に自動でインストールされ、自動的に起動します。また、テストデータも投入済みとなっています。

2.6 devstack の中身

まず、devstack は巨大な bash スクリプトの集合体です。そして、devstack には多くの設定項目が存在します。しかし、そのほとんどには適切なデフォルト値が設定されています。そのため、特に設定を変更せずとも、それなりに動くようになりますが、2 つ問題があります。デフォルトでは起動されないコンポーネントがあることと、デフォルトの値では結構なスペックが必要となることです。たとえば、以下のように設定ファイルを書くと、devstack で起動できるすべてのコンポーネントを起動することができます。

```
enable_service tempest
enable_service ceilometer ceilometer-acompute ceilometer-acentral \
               ceilometer-anotification ceilometer-collector ceilometer-api
enable_service s-proxy s-object s-container s-account
enable_service trove tr-api tr-tmgr tr-cond
enable_service sahara
enable_service zaqar-server
disable_service heat h-eng h-api h-api-cfn h-api-cw
```

tempest は結合テスト、ceilometer はリソース使用量の解析、s-で始まるのはオブジェクトストレージサービスの Swift、trove はデータベースサービス、sahara は Hadoop クラスタをデプロイする機能、zaqar はキューイングサービス、heat はサーバーのオーケストレーションです。これらを全部起動するとどうなるかと言うと、ものっすごいメモリを消費します。8GB くらいは欲しいですね。

他にも便利な設定値があります

RECLONE=yes

RECLONE は、devstack 起動時に最新のソースコードを毎回 clone してくるか否かです。yes にすると、./stack.sh を実行するたびに本家のリポジトリからソースコードを clone してきます。常に最新の機能を使うことができますが、まぁこれは OpenStack そのものの開発者しか必要ないでしょう。とはいえ、最新のログを追うのもコミュニティの息づかいを感じることができて楽しいものです。ほら、あれだよ。デグレったりとかさ。

VOLUME_BACKING_FILE_SIZE=1G

Cinder の volume 用に作成されるファイルのサイズなのですが、これがまたデフォルトではデカいんだな。devstack 起動するだけで急にディスクの使用率が上がったりします。1G もあればお試しには十分じゃないのかなぁ。df と話し合って決めてください。

SCREEN_HARDSTATUS

デフォルトの Screen のステータスバーがちょっと見にくい。この変数で変えることができます。.screenrc を書いておけばプレフィックスキーなども変更することができます。が、 hardstatus は上書かれてしまうようです。

```
SCREEN_HARDSTATUS='%{= .b}%-Lw%{=b .r}%>%n%f %t*%{= .b}%+Lw%< %-=%{g}(%{d}%H/%l%{g})'
```

ENABLE_HTTPD_MOD_WSGI_SERVICES=False

さきほどの ps の出力をよく見てみましょう。実は、Horizon は apache でリクエストを受け、mod_wsgi を使って処理をする Web アプリケーションとして起動しています。そして、Keystone も同様の起動スタイルをしています。やたら Apache からプロセス生えてますね。Keystone の動作確認やパッチ当て等の開発をしている時には、そのたびに Apache の再起動が必要になってしまいます。それはまぁ無駄といえば無駄なので、このオプションで apache で受けないようにすることができます。まぁこれも開発用途と言えるでしょう。

ところで、これらの設定項目ってどこに書いてあると思いますか？はい、正解です！そう、ソースコードです。「これ不便だな。変えたいな。」って思ったら、grep かけてみましょう。何か見つかるでしょう。

2.6.1 devstack メモ

Fedora、またお前か

Fedora にはいつものことですが、パッケージが壊れてます。Fedora 21 Server の beta で devstack 走らせてみると、pcre-devel と pcre の依存が壊れてます。

```
% sudo yum install pcre-devel読み込んだプラグイン
:langpacks依存性の解決をしています

--> トランザクションの確認を実行しています。
---> パッケージ pcre-devel.x86_64 0:8.35-7.fc21 をインストール
--> 依存性の処理をしています: pcre(x86-64) = 8.35-7.fc21 のパッケージ: pcre-devel-8.35-7.fc21.x86_64
--> 依存性解決を終了しました。エラー
: パッケージ: pcre-devel-8.35-7.fc21.x86_64 (fedora)要求
              : pcre(x86-64) = 8.35-7.fc21インストール
             : pcre-8.35-8.fc21.x86_64 (@updates-testing)
                 pcre(x86-64) = 8.35-8.fc21利用可能
             : pcre-8.35-7.fc21.x86_64 (fedora)
                 pcre(x86-64) = 8.35-7.fc21問題を回避するために
  --skip-broken を用いることができます。これらを試行できます
 : rpm -Va --nofiles --nodigest
```

beta だし、こんなことで心折れていてはいけません。リラダンの『未来のイヴ』を読む方がよっぽど大変ですよ。あれは大変でした。まぁとりあえず即座に：

```
yum downgrade pcre
```

Cenlometer 試してみたらディスクフルで天使が見えた

devstack は Ceilomerter もサポートしています。で、Ceilometer は計測データの保管に mongodb を使うのですが、mongodb は起動時に各データベースごと 5GB のディスクを確保しようとします。Ceilometer は DB を 3 つ必要としますので、あら大変、15GB も消費されてしまうことになります。Ceilometer を試す際には、/etc/mongodb.conf に以下を追記し、大きなファイルを作成しないようにしておきましょう。

```
smallfiles = true
```

Oh, Proxy.

あぁ、Proxy ですね。ソースコードを Clone してくるのに git プロトコルを使うのですが、これを変えたい時は instead of 設定をしましょう。というのはすぐ思いつくのですが、–system オプションでユーザーの設定に依存しないところまで広く設定しておきましょう。devstack はカレントユーザーの git 設定は読みません。

OS のセットアップについて

何をもってして Linux のインストールが終わったか、というのは人によるでしょう。ここでは、インストーラーでの作業が終了し、パッケージのアップデートが終わっている状態とします。(apt-get update && apt-get upgrade, yum update, dnf upgrade) 普通は他にもやりたいことがあるかと思います。(vim, emacs, zsh, screen, tmux, tig ...) 執筆時には以下をインストールしました：vim zsh tmux tig

ホストサーバーのリソースについて

Fedora 21 Server には Cockpit というサーバーリソース管理サービスがデフォルトでインストールされています。有効化はされていないので、以下で起動を指定し、9090 ポートを Web ブラウザで開くと、リソース使用状況を綺麗なページで見ることができます。まぁ、出来はまだまだですが。

systemctl start cockpit

CPU、メモリ、ディスク、ネットワーク、起動中のサービスなどの基本情報に加えて、ユーザー情報、Docker サービス、Docker コンテナ、journalctl なんかも Web から見ることができます。

2.7 ブート時のインスタンスの設定

2.7.1 config_drive

インスタンスをブートする際に、そのインタンスに対してメタデータを書き込んだ特別な設定用ドライブをアタッチして OpenStack の設定を行うことができます。その際、アタッチする設定用仮想ドライブをコンフィグドライブと呼びます。インスタンスはこのドライブをマウントしてファイルを読み情報を取得することが可能です。例えば、インスタンスへ IP アドレスを割り当てる際に DHCP を使用しない場合、ネットワーク設定が書き込まれているメタデータファイルをコンフィグドライブとしてマウントして固定 IP アドレスの設定をすることができます。コンフィグドライブは、インスタンスにネットワーク設定を施す前にマウントしてメタデータへのアクセスが可能になるが故、IP アドレスの設定が可能になるのです。

2.7.2 cloud-init

コンフィグドライブから情報を取得し、ブート時にインスタンスの設定を行う際、cloud-init との連携が必要になります。cloud-init は OpenStack 固有のテクノロジーではありません。マシンイメージをマルチクラウド上に展開することができるようにするため、複数のクラウドサービスをサポートできるように設計されています。

2.7.3 インストール

1. RHEL6/CentOS6

```
rpm -Uvh http://dl.fedoraproject.org/pub/epel/6/x86_64/epel-release-6-8.noarch.rpm
yum install cloud-init
```

2. Ubuntu/Debian

```
apt-get install cloud-init
```

2.7.4 実用例

cloud-init を利用することでインスタンスの起動時に任意スクリプトを実行することができます。

例として Glance イメージから Nova でインスタンスをブートする際に固定 IP を設定する手順を紹介します。まず、設定ファイルの以下の項目を追記します。

```
# cat /etc/cloud/cloud.cfg
cloud_final_modules:
- scripts-per-once
- scripts-per-boot
- scripts-per-instance
```

追加した内容は下記となります。

scripts-per-once

”/var/lib/cloud/scripts/per-once/”以下のスクリプトを実行

* 初回起動時に実行 (OpenStack 環境ではユーザが独自イメージを作成する際に実行される)

scripts-per-boot

”/var/lib/cloud/scripts/per-boot/”以下のスクリプトを実行

* リブート時に実行

scripts-per-instance

”/var/lib/cloud/scripts/per-instance/”以下のスクリプトを実行

* Nova でインスタンスを初回ブートする時に実行

インスタンスへの固定 IP の設定は初回ブート時に必要な作業なので、scripts-per-instance を利用する。以下のとおり、スクリプトを配置します。

```
# cd /var/lib/cloud/scripts
```

```
# tree `pwd`
/var/lib/cloud/scripts
├── per-instance
│   └── per-instance.sh
```

スクリプトの中身は以下のとおりです。

```
# cat per-instance.sh
#!/bin/bash
/root/bootstrap/networking.py
```

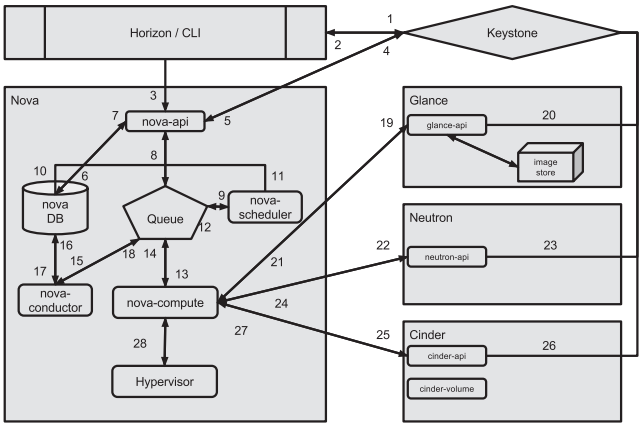
/root/bootstrap 配下に固定 IP を設定するための Python スクリプトを配置します。

```
# cd /root/bootstrap
# tree `pwd`
/root/bootstrap
├── lib
│   ├── __init__.py
│   ├── cmd_args.py
│   ├── exception.py
│   ├── gateway.py
│   ├── ip_addr.py
│   └── util.py
└── networking.py
```

処理の流れとしては、config_drive をマウントし、ipconfig コマンド、route コマンドを使い IP 設定を行い、最後に config_drive をアンマウントするという流れです。当然、次回リブートすると設定は消えてしまうので、ブート後ネットワークの設定ファイルを修正する必要がありますが、このスクリプトにより初回ブート時に自動で ssh できるようになります。Python スクリプトの中身は以下で紹介していますので、ご参考下さい。

https://github.com/Project-VI/OpenStack/blob/master/5-1-userdata_py.txt

最後に、第 5 章では、より詳細な OpenStack でインスタンスを生成した際の内部で起こっているブートシーケンスについて順を追って説明します。なお、ここでは API への通信は Horizon、もしくは novaclient による CLI 操作を想定しています。



1. **Horizon or CLI** ユーザの認証情報を取得し REST の通信により Keystone に認証をかけます
2. **Keystone** ユーザ情報を認証し、認証トークンを生成してユーザにその値を返します。認証トークンは、その他のコンポーネントに REST 通信を通してリクエストを投げるためのものです。
3. **Horizon or CLI Horizon** もしくは CLI は、REST API により'launch instance'、もしくは'nova-boot'と定義される新しいインスタンスを生成するリクエストを変換して、それを nova-api に送ります。
4. **nova-api** nova-api は、Keystone に対してトークンとユーザのパーミッションの確認を行うリクエストを投げます。
5. **Keystone** Keystone は、トークンを確認し、更新された認証用ヘッダーをロール情報とパーミッションとともに送ります。
6. **nova-api** nova-api は Nova のバックグラウンドのデータベースとのセッションを確立します。
7. **Database** 新しいインスタンス用のレコードをデータベースに登録します。
8. **nova-api** nova-api は nova-scheduler に対して nova-compute のホスト ID を決定付けるインスタンスの更新エントリーを除く rpc.call のリクエストを投げます。
9. **nova-scheduler** nova-scheduler はメッセージングキューを介してリクエストを受けます。
10. **nova-scheduler** nova-scheduler は Nova のバックエンドのデータベースとセッションを確立し、フィルターと重み付け機能を通してインスタンス生成にふさわしい nova-compute のホストを決定します。
11. **nova-scheduler** フィルターと重み付け機能を通して決定された nova-compute のホスト情報を付与したインスタンスのエントリーを返します。
12. **nova-scheduler** nova-scheduler は、決定された nova-compute のホストに対して'launch instance' の rpc.cast のリクエストを送ります。
13. **nova-compute** nova-compute はメッセージングキューを介してリクエストを受けます。
14. **nova-compute** nova-compute は nova-conductor に対してインスタンスのホストの ID や Flavor(メモリ、CPU、ディスク) の情報を取ってくるために、nova-conductor に対して rpc.call のリクエストを投げます。
15. **nova-conductor** nova-conductor はメッセージングキューを介してリクエストを受けます。
16. **nova-conductor** nova-conductor は Nova のバックグラウンドのデータベースとのセッションを確立します。
17. **Database** データベースはインスタンスの情報を返します。
18. **nova-compute** nova-compute はメッセージングキューを介してリクエストを受けます。
19. **nova-compute** nova-compute は glance-api に対して認証トークンを通してブート対象のイメージ ID を持ったイメージの URI を取得しイメージ用ストレージからイメージをアップロードするための REST のコールを行います。
20. **glance-api** glance-api は Keystone に対して認証トークンを確認しにいきます。
21. **nova-compute** nova-comupte はイメージのメタデータと取得します。
22. **nova-compute** nova-compute は Network API に対して認証トークンを通してインスタンスに紐付けるネットワークの割り当てと設定を行うための REST のコールを行います。
23. **neutron-server** neutron-server は Keystone に対して認証トークンを確認しにいきます。
24. **nova-compute** nova-compute はネットワーク情報を取得します。
25. **nova-compute** nova-compute は Volume API に対して認証トークンを通してインスタンスにボリュームをアタッチするための REST のコールを行います。
26. **cinder-api** cinder-api は Keystone に対して認証トークンを確認しにいきます。
27. **nova-comupte** nova-compute はブロックストレージの情報を取得します。

28. **nova-compute** nova-compute はハイパーバイザードライバーのためのデータを生成し、libvirt もしくは API を通してハイパーバイザーに対してリクエストを実行します。

以下の表は、上記の各プロビジョニングステップにおけるインスタンスのステータスの変化を表しています。

Status	Task	Power state	Steps
Build	scheduling	None	3-12
Build	networking	None	22-24
Build	block_device_mapping	None	25-27
Build	spawing	None	28
Active	none	Running	

第 3 章

あとがき

3.1 こじろー

3.1.1 ミームの死骸を待ちながら

「組織も人も、特殊化の果てにあるのはゆるやかな死、それだけよ」

草薙素子『GHOST IN THE SHELL / 攻殻機動隊』

オープンクラウドという名で、まさに組織が抱えるゆるやかな死を阻止すべく登場したのが OpenStack である。インフラストラクチャーの構築において知の共有がなされていなかった時代、各企業はそれぞれ独自の手法を見出し特殊化していった。2010 年代に入り、特殊化が自らのシステムをゆるやかに死に追いやっていることに皆気付き始めた。隔離的に画一化されたシステムは何らかのアクシデントで全滅する可能性がある。考えうる唯一の冴えたやり方は、グローバルな共有知の創出により集団的知性の発生を促すことである。

15 年前に GNU/Linux が成し遂げた夢を、クラウドの分野で再度叶えるのである。この時代にそれを成すためには、ミーム学的アプローチにより、互いに依存しあうミーム複合体 (memeplex) を構築することが必要である。偶然か必然か、そのアプローチを取ったのが、OpenStack である。GNU/Linux 上に構築されたクラウドシステム (ミーム複合体) とそれと連携するドライバモデル (ミーム) は、まさにミームが寄り集まってミーム複合体を形成する様に酷似しているのである。オープンソースであることはもちろんのこと、単一企業による開発コミュニティの独占が成されていないことにより、共有知の創出を圧倒的速度で促すことに成功したのである。OpenStack によりクラウドのあり方は変わるであろう。が、それと同時に僕は、ミームの死骸に恋い焦がれているのである。

3.2 まっきー

3.2.1 OpenStack は革命戦士の夢を見るか、あるいはすべてがエスになる世界について

否定性は、止揚の不可欠の一契機である。怒り、等々の否定性の感情は、人が今ある存在を実践的にのりこえ、変換してゆくときに不可欠の前提である。しかしこの怒り、等々の感情は、それがあまりにも直接的で、なまなましくわれわれをおそうとき、われわれの超越への意志を緊縛し、あらがいがたい強力をもって、われわれを内在性のうちにひき戻してしまう。情況にたいしていいったん距離をおき、これを明晰に相対的に把握することを許さない。怒りはわれわれを、情況のたんなる否定性、同位対立者としての境位に呪縛してしまう。（中略）人間的な実践の弁証法は解体し否定性の実存だけがむきだしに残る。そ

> して現代の支配体制の柔構造は、このような無援の実存を愚弄し、吸収するための幾重もの装置をもっている。
>
> 見田宗介『まなざしの地獄』

世界を変えるには少なくとも二つの方法がある。一つは、その世界の外に出ること。もう一つは、その世界を変えるだけの地位に付くことだ。かつて集団をもってして世界を変えようとしたかれらに、なぜ我々は手を貸さなかったのか。彼らの臨界点は何だったか。われわれは世界を本当に変えてしまっていいのか。われわれは世界を変えることを望んでいるのか。このあとがきは、主にかつて共に暮らした友人達への謝罪のために書くものである。

その前に OpenStack の話をしよう。OpenStack は確かに良いものであろう。ただ、残念ながら OpenStack で成功を得られるかと言うと、それは何とも答えようがない。まず成功が何かを考えなければならないし、さらに残念なことには、そのようなことは大抵のユーザーにとってはさほど重要なことではない。重要ではないものを話されると何が起きるかというと、時には相手に対する怒りがわくというわけだ。不快だもんね。

革命戦士達に手を貸さなかった理由は、すくなくとも僕にとっては、それが重要ではなかったからだった。重要ではないというのは、重要になるほど不快ではなかったという意味だ。ありていに言えば、別に今の世界に不満はないということだ。従って、僕にとって OpenStack は革命戦士ではない。私は革命を望んではいない。ビッグブラザーは消えたことにしてしまえ。となれば、愛は友人に向けられるべきではないのか。友達が笑っていれば、それでいいのだ。そこはまなざしの地獄かもしれないけれど。

世界の外に出てしまえば、世界を出た自分を認識せねばならない。地位を上るには、地位の高さを自分で認識せねばならない。なるほど是非ともみなやるべきだが、少なくとも僕はやりたくない。それで腹がふくらむわけでもあるまいし。世界の外には無援の実存を愚弄するものも、アブゾーブしてくれる装置もない。だが、世界の中には、両方ある。そこが良い場所だと訓練すれば世界は平和だ。よい場所だと思わせるのが監獄だ。規律と訓練。あぁ、これは語り尽くされたことだった。われわれはそこから出ることは学ばなかったし、出ようとも思わない。

世界をよりよくしたいという思いの居場所を考えないといけないわれわれは、その適用と探索の範囲を、自身にまつわる狭い範疇で認識すればよいものとし、それを世界と呼んだ。見たまえ、やっぱり世界は変わってないじゃないか。君たちのやっていることは迷惑か無意味ではないか。僕は今日も仕事に行く。

時には世の中に違和感を覚え、その解決に行動を起こしたくなる時もあるが、それは僕の生活にとってあまり重要ではなかった。食うに困ったら役所にでも書類を出しにいくか、とか思っている。

得てして、良き友は今でもただの良き友のままであり、僕は革命戦士の夢を見る。怒りがある分には、まだ OpenStack もまた、革命戦士の夢をみてもバチはあたらない気がする。

3.3 くろろろろ

イラスト担当のくろろろろです。人様の原稿に絵を付けたのはこれが初めてなので、とても緊張しました。表紙を見て OpenStack に少しでも興味を持って貰えると嬉しく思います。


入門編
じょしだいせい
JDと学ぶ
OpenStack