OpenStack 運用ガイド

[FAMILY Given]

概要

本書は OpenStack クラウドの設計および運用に関する情報を提供します。

謝辞

OpenStack Foundationは、オースチンへの航空券、(暴風後の停電によるドキドキの夜を含む)宿、そして美味しい食事で、この本の作成をサポートしました。約10,000USドルで、Rackspaceのオースチンオフィスの同じ部屋の中で、私たちは1週間で集中的に共同作業をすることができました。著者たちはすべてOpenStack Foundationのメンバーであり、それにはあなたも参加することができます。FoundationのWebサイト (http://openstack.org/join) に行ってみてください。

私たちは、オースチンの Rackspace での素晴らしいホスト Rackersに感謝したい:

- Rackspace ゲストリレーションズの Emma Richards は、私たちのランチの注文を素晴らしく面倒を見てくれて、更に壁から剥がれ落ちた付箋紙の山を脇においてくれました。
- 熱狂的なエグゼクティブアシスタントの Betsy Hagemeier は、部屋の改造の面倒を見てくれて、1週間で解決する手助けをしてくれました。
- 「The Victors」としても知られている、オースチンの Rackspace の不動産チームは、素晴らしい応答をしてくれました。
- Rackspace IT部門 の Adam Powell は、私たちに毎日のネットワーク帯域を提供してくれました。また、より多くのスクリーンが必要となったため、セカンドモニタを提供してくれました。
- 水曜日の夜、オースチン OpenStack ミートアップグループと楽しく幸せな時間を過ごし、Racker Katie Schmidt は私たちのグループを素晴らしい世話をしてくれました。

私たちは部屋の外から、いくつかの素晴らしいインプットを得ました。

- CERNの Tim Bell は、私たちが作業を開始する前に、その概要についてフィードバックを与えてくれて、週の半ばにはレビューをしてくれました。
- Sébastien Han は素晴らしいブログを書いてくれて、寛大にも再利用の許可を与えてくれました。
- Oisin Feeley は、このマニュアルを読んで、いくつかの編集をし、私たちが問い合わせをした際には、E-mailでのフィードバックをくれました。

Inside the book sprint room with us each day was our book sprint facilitator Adam Hyde. Without his tireless support and encouragement, we would have thought a book of this scope was impossible in five days. Adam has proven the book sprint method effectively again and again. He creates both tools and faith in collaborative authoring at www.booksprints.net.

私たちは、これらの多大な協力的な援助と励まし無しには、これを成し遂げることはできなかったでしょう。

目次

はじめに 3
OpenStack の概要 3
この本の対象読者 4
この本の構成 6
この本をなぜ書いたか？どうやって書いたか？ 8
この本の作成に参加するには 11
1. プロビジョニングとデプロイメント 13
自動デプロイメント 13
自動環境設定 16
リモート管理 17
2. クラウドコントローラーの設計 19
ハードウェアの考慮事項 20
サービスの分離 22
データベース 22
メッセージキュー 22
Application Programming Interface (API) 23
API 拡張 23
スケジューラー 25
イメージ 25
ダッシュボード 26
認証と認可 26
ネットワークの考慮事項 28
3. スケーリング 29
出発点 29
コントローラーノードの追加 31
クラウドの分離 32
スケーラブルハードウェア 35
4. コンピュートノード 37
CPU の選択 37
ハイパーバイザーの選択 38
インスタンスストレージのソリューション 38
オーバーコミット 43
ロギング 43
ネットワーク 44
5. ストレージ選定 45
ストレージのコンセプト 45
ストレージバックエンドの選択 48
OpenStack Object Storage の注意事項 52
6. ネットワーク設計 55
管理ネットワーク 55
パブリックアドレスの選択肢 55

IP アドレス計画 56
ネットワークトポロジー 58
ネットワーク関係のサービス 60
7. 参考アーキテクチャ 63
概要 63
設定指針 64
詳細な説明 66
さらなる拡張 69
8. 環境の把握 71
クライアントコマンドラインツール 71
ネットワーク 79
ユーザーとプロジェクト 79
稼働中のインスタンス 81
9. プロジェクトとユーザーの管理 83
プロジェクトかテナントか? 83
プロジェクトの管理 83
クォータ 85
ユーザー管理 90
新規ユーザーの作成 90
プロジェクトへのユーザーの割り当て 91
10. ユーザーによる運用 97
イメージ 97
フレーバー 99
セキュリティグループ 101
ブロックストレージ 105
インスタンス 106
セキュリティグループの割り当て 110
Floating IP 111
ブロックストレージの接続 111
スナップショットの取得 112
データベースにあるインスタンス 116
11. メンテナンス、故障およびデバッグ 119
クラウドコントローラーとストレージプロキシの故障とメンテナンス 119
コンピュートノードの故障とメンテナンス 121
ストレージノードの故障とメンテナンス 127
完全な故障の対処 129
構成管理 130
ハードウェアの取り扱い 130
データベース 132
HDWMY 133
故障しているコンポーネントの特定 134
アップグレード 136

アンインストール 137
12. ネットワークのトラブルシューティング 139
「ip a」を使ってインタフェース状態をチェックする 139
クラウド上のネットワークトラフィック 140
経路上の障害を見つける 141
tcpdump 141
iptables 143
データベースにあるネットワーク設定 144
DHCP の問題をデバッグする 145
DNS の問題をデバッグする 148
13. ロギングと監視 151
ログはどこにあるのか？ 151
ログの読み方 152
インスタンスリクエストの追跡 153
カスタムログの追加 154
RabbitMQ Web管理インターフェイス および rabbitmqctl 154
ログの集中管理 155
StackTach 157
監視 157
14. バックアップとリカバリー 165
バックアップ対象 165
データベースのバックアップ 165
ファイルシステムバックアップ 166
バックアップのリカバリー 168
15. カスタマイズ 169
DevStack 169
ミドルウェア例 173
Nova スケジューラーの例 179
ダッシュボード 184
16. OpenStack コミュニティ 185
助けを得る 185
バグ報告 186
OpenStack コミュニティに参加する 189
機能と開発ロードマップ 190
ドキュメント作成への貢献の仕方 192
セキュリティ情報 193
さらに情報を見つける 194
17. 高度な設定 195
ドライバーによる違い 195
周期的タスク 196
設定に関する個別のトピック 197
A. 事例 199
NeCTAR 199

MIT CSAIL 200
DAIR 201
CERN 201
B. ハリウッド^H^H^H^H^Hクラウドナイトメア 203
ダブル VLAN 203
「あの問題」 206
イメージの消失 208
バレンタインデーのコンピュートノード大虐殺 210
ウサギの穴に落ちて 212
C. リソース 215
用語集 217

表の一覧

5.1. OpenStackのストレージ 45
9.1. コンピュートのクォータの説明 85
9.2. ブロックストレージのクォータの設定 88
11.1. サービス復旧優先度一覧の例 129

ドキュメント変更履歴

このバージョンのドキュメントでは、それ以前のバージョンをすべて置き換えています。以下の変更履歴には、それ以降の最近の変更を記載しています。

Revision Date	Summary of Changes
December 12, 2013	• Additions to acknowledgements and preface to situate this guide relative to other OpenStack guides.
May 13, 2013	• アベイラビリティゾーンの説明を更新しました。
April 2, 2013	• 画面例がページや注記に収まるように修正しました。
March 22, 2013	• HTML 出力において分割することを止めました。
March 20, 2013	• 編集における変更をしました。 • glossterm タグを用語集の用語に追加しました。 • コード例におけるフォーマットを整理しました。 • 未来時制を削除しました。
March 11, 2013	• ファイルを OpenStack github リポジトリに移動しました。

はじめに

OpenStack の概要 3
この本の対象読者 4
この本の構成 6
この本をなぜ書いたか？どうやって書いたか？ 8
この本の作成に参加するには 11

OpenStack はオープンソースプラットフォームで、OpenStack を使うと、コモディティハードウェア上で動作する Infrastructure as a Service (IaaS) クラウドを自分で構築できます。

OpenStack の概要

OpenStack believes in open source, open design, open development, all in an open community so anyone can participate. The long-term vision for OpenStack is to produce a ubiquitous open source cloud computing platform that meets the needs of public and private cloud providers regardless of size. OpenStack services control large pools of compute, storage, and networking resources throughout a data center.

Each service provides a REST API so that all these resources can be managed through a dashboard that gives administrators control while empowering users to provision resources through a web interface, through a command-line client, or through software development kits that support the API. Many OpenStack APIs are extensible, meaning you can keep compatibility with a core set of calls while providing access to more resources and innovating through API extensions. The OpenStack project is a global collaboration of developers and cloud computing technologists producing the open standard cloud computing platform for both public and private clouds. The project aims to deliver solutions for all types of clouds by being simple to implement, massively scalable, and feature-rich. The technology consists of a series of interrelated projects delivering various components for a cloud infrastructure solution.

Okay, you say, OpenStack is open source cloud. Are there others? Yes, Eucalyptus offers an open source version of their private cloud offering that provides services compatible with Amazon Web Services (AWS) APIs such as EC2 and S3. With Ubuntu Server

11.10 release, OpenStack became supported by default rather than Eucalyptus for Ubuntu Server releases. The product targets private cloud deployers who want to maintain compatibility with Amazon Web Services by partnering with Amazon. Apache CloudStack is another example of an open source cloud project, but it only runs on Ubuntu 10.04. It offers computing as a core service, with connections to storage services. Citrix purchased the project technology for $200 million in 2011 when buying cloud.com, then released it to Apache in April 2012. The API compatibility for CloudStack centers on AWS APIs.

OpenStack はスケーラビリティを意識して設計されており、あなたのクラウドが大きくなるのにあわせて新たなコンピュートリソースやストレージリソースを簡単に追加することができます。HP や Rackspace などの組織では OpenStack を使って巨大なパブリック・クラウドを構築しています。OpenStack は、そのまま実行する単なるソフトウェアパッケージではなく、数多くの様々な技術を組み合わせてクラウドを構築することができます。このアプローチは非常に柔軟性がありますが、このため多くの選択肢があり最初は面食らうかもしれません。

この本の対象読者

このガイドは、Linux ディストリビューションの Ubuntu、SQL データベースや仮想化に関してよく知っていることを前提にしています。複数台の Linux マシンのネットワーク設定・管理にも慣れている必要があります。MySQL データベースのインストールと管理を行い、場合によってはデータベースに対して SQL クエリーを実行することもあります。

OpenStack クラウドの最も複雑な点の一つにネットワーク設定があります。DHCP、Linux ブリッジ、VLAN、iptables といった考え方をよく理解していなければなりません。OpenStack クラウドで必要となるスイッチやルータを設定できるネットワークハードウェアの専門家と話をする必要もあります。

This book is for those of you starting to run OpenStack clouds as well as those of you who were handed a running one and want to keep it running well. Perhaps you're on a devops team, perhaps you are a system administrator starting to dabble in the cloud, or maybe you want to get on that OpenStack cloud team at your company. This book is for all of you.

OpenStack のドキュメントウェブサイト docs.openstack.org には他にも本があり、助けになることでしょう。

OpenStack の各種ガイド

OpenStack インストールガイド	Describe a manual install process, as in, by hand, no automation, for multiple distributions based on packaging system: • Debian 7.0 向けインストールガイド • openSUSE および SUSE Linux Enterprise Server 向けインストールガイド • Red Hat Enterprise Linux, CentOS, Fedora 向けインストールガイド • Ubuntu 12.04 (LTS) Server 向けインストールガイド
OpenStack 設定レファレンス	Contains a reference listing of all configuration options for core and integrated OpenStack services by release version.
OpenStack クラウド管理者ガイド	Contains how-to information for managing an OpenStack cloud as needed for your use cases, such as storage, computing, or software-defined-networking.
OpenStack 高可用性ガイド	OpenStack サービス、関連するコントローラやデータストアを高可用にするために取りうる方策に説明しています。
OpenStack セキュリティガイド	OpenStack クラウドを安全にするためのベストプラクティスと基本的な考え方について書かれています。
仮想マシンイメージガイド	OpenStack で利用可能な仮想マシンイメージを取得、作成、更新する方法について説明されています。

OpenStack エンドユーザーガイド	Shows OpenStack end users how to create and manage resources in an OpenStack cloud with the OpenStack dashboard and OpenStack client commands.
OpenStack 管理ユーザーガイド	Shows OpenStack administrators how to create and manage resources in an OpenStack cloud with the OpenStack dashboard and OpenStack client commands.
OpenStack API クイックスタート	OpenStack サービスのエンドポイントに REST API リクエストをどのように送信するかについての概要が説明されています。

この本の構成

この本は 2つの部分から構成されています。一つは OpenStack クラウド設計においてアーキテクチャーの決定に関してで、もうひとつは稼働中の OpenStack クラウドで繰り返し行う運用に関してです。

Chapter 1: Provisioning and Deployment: While this book doesn't describe installation, we do recommend automation for deployment and configuration, discussed in this chapter.

Chapter 2: Cloud Controller Design: The cloud controller is an invention for the sake of consolidating and describing which services run on which nodes. The chapter discusses hardware and network considerations as well as how to design the cloud controller for performance and separation of services.

Chapter 3: Scaling: This chapter discusses the growth of your cloud resources through scaling and segregation considerations.

Chapter 4: Compute Nodes: This chapter describes the compute nodes, which are dedicated to run virtual machines. Some hardware choices come into play here as well as logging and networking descriptions.

Chapter 5: Storage Decisions: Along with other architecture decisions, storage concepts within OpenStack take a lot of consideration, and this chapter lays out the choices for you.

Chapter 6: Network Design: Your OpenStack cloud networking needs to fit into your existing networks while also enabling the best design for your users and administrators, and this chapter gives you in-depth information about networking decisions.

Chapter 7: Example Architecture: Because of all the decisions the previous chapters discuss, this chapter describes the decisions made for this particular book and much of the justification for the example architecture.

Chapter 8: Lay of the Land: This chapter is written to let you get your hands wrapped around your OpenStack cloud through command line tools and understanding what is already set up in your cloud.

Chapter 9: Managing Projects and Users: This chapter walks through those user-enabling processes that all admins must face to manage users, give them quotas to parcel out resources, and so on.

Chapter 10: User-facing Operations: This chapter moves along to show you how to use OpenStack cloud resources and train your users as well.

Chapter 11: Maintenance, Failures, and Debugging: This chapter goes into the common failures the authors have seen while running clouds in production, including troubleshooting.

Chapter 12: Network Troubleshooting: Because network troubleshooting is especially difficult with virtual resources, this chapter is chock-full of helpful tips and tricks to tracing network traffic, finding the root cause of networking failures, and debugging related services like DHCP and DNS.

Chapter 13: Logging and Monitoring: This chapter shows you where OpenStack places logs and how to best to read and manage logs for monitoring purposes.

Chapter 14: Backup and Recovery: This chapter describes what you need to back up within OpenStack as well as best practices for recovering backups.

Chapter 15: Customize: When you need to get a specialized feature into OpenStack, this chapter describes how to use DevStack to write custom middleware or a custom scheduler to rebalance your resources.

Chapter 16: Upstream OpenStack: Because OpenStack is so, well, open, this chapter is dedicated to helping you navigate the community and find out where you can help and where you can get help.

Chapter 17: Advanced Configuration: Much of OpenStack is driver-oriented, where you can plug in different solutions to the base set of services. This chapter describes some advanced configuration topics.

Appendix A: Use Cases: You can read a small selection of use cases from the OpenStack community with some technical detail and further resources.

Appendix B: Tales From the Cryp^H^H^H^H Cloud: These are shared legendary tales of image disappearances, VM massacres, and crazy troubleshooting techniques to share those hard-learned lessons and wisdom.

Appendix C: Resources: So many OpenStack resources are available online due to the fast-moving nature of the project, but there are also listed resources the authors found helpful while learning themselves.

Glossary: A list of terms used in this book is included, which is a subset of the larger OpenStack Glossary available online.

この本をなぜ書いたか？どうやって書いたか？

私たちは少なくとも1年以上 OpenStack クラウドを構築し運用してきました。そこで得られた知識を多くの人と共有するために、この本を書きました。 OpenStack クラウドの責任者として数ヶ月がたつと、そのドキュメントを渡しておけば、システム管理者に日々のクラウドの運用をどのように行なえばよいかが分かるようなドキュメントが欲しくなりました。また、クラウドを構築する際に選択したやり方のより詳細な技術情報を共有したいと思いました。

次のような場面であなたの助けとなるように、この本を書きました。

- 初めての本格的な OpenStack クラウドのアーキテクチャの設計と構築。この本を読み終えると、コンピュート、ネットワーク、ストレージリソースを選ぶにはどんな質問を自分にすればよいのか、どのように組み上げればよいのかや、どんなソフトウェアパッケージが必要かが分かることでしょう。

- クラウドを管理する上で必要となる日々のタスクの実行。

私たちはこの本を Book Sprint で執筆しました。 Book Sprint は短い期間で本を建設的に作成できるメソッドです。詳しい情報は、 Book Sprint のサイト を参照して下さい。著者らは2013年2月の5日間でこの本をまとめあげました。カフェインと、テキサス州オースティンの素晴らしいテイクアウトの食事は力になりました。

最初の日に、アイデアを色とりどりのポストイットでホワイトボードいっぱいに書き出し、クラウドを設計し運用するという漠然とした話題を扱った本の作成を開始しました。

私たちは一心不乱に自分たちの経験に基づき執筆を行い、互いに意見をぶつけ合いました。一定の間隔で、本の現在の状況や構成をレビューし、本を作り上げていき、今皆さんが見ているものができあがりました。

以下が執筆チームのメンバーです。

- Tom Fifield. LHC で ATLAS のような素粒子物理学実験でコンピューティングのスケーラビリティの経験を積んだ後、現在はオーストラリアの公的な研究部門を支援するプロダクションの OpenStack クラウドに携わっていました。現在は OpenStack のコミュニティマネージャーを務めており、空いた時間で OpenStack ドキュメントプロジェクトに参加しています。

- Diane Fleming. 彼女は OpenStack API ドキュメントプロジェクトで非常に熱心に活動しています。このプロジェクトでは自分ができるところであれば、どこでも取り組んでくれました。

- Anne Gentle. 彼女は OpenStack のドキュメントコーディネーターで、2011年の Google Doc Summit では individual contributor （個人コントリビュータ） を努め Open Street Maps チームとともに活動しました。Adam Hyde が進めていた FLOSS Manuals の以前の doc sprint にも参加しています。テキサス州オースティンに住んでいます。

- Lorin Hochstein. アカデミック出身のソフトウェア開発者・運用者である彼は、Nimbis Services でクラウドサービスの Lead Architect として働いています。Nimbis Service では彼は技術計算アプリケーション用の OpenStack を運用しています。 Cactus リリース以来 OpenStack に携わっています。以前は、University of Southern California's Information Sciences Institute (USC-ISI) で OpenStack の high-performance computing 向けの拡張を行いました。

- Adam Hyde. 彼は今回の Book Sprint をリードしました。 Book Sprint メソッドを創設者でもあり、一番経験豊富な Book Sprint のファシリテーターです。詳しい情報は http://www.booksprints.net/ を見て下さい。 3000人もの参加者がいるフリーソフトウェアのフリーなマニュアルを作成するコミュニティである FLOSS Manuals の創設者です。また、Booktype の創設者でプロジェクトマネージャーです。Booktype はオンラインで本の執筆、編集、出版を行うオープンソースプロジェクトです。

- Jonathan Proulx. 彼は MIT Computer Science and Artificial Intelligence Lab で上級技術アーキテクトとして OpenStack クラウドを運用し、研究者が必要なだけの計算能力を使えるようにしています。 OpenStack の勉強を加速しようと思い、OpenStack ドキュメントへの貢献とドキュメントのレビューを始めました。

- Everett Toews. 彼は Rackspace の Developer Advocate で、OpenStack や Rackspace Cloud を使いやすくする仕事をしていま

す。ある時は開発者、ある時は advocate、またある時は運用者です。彼は、ウェブアプリケーションを作成し、ワークショップを行い、世界中で公演を行い、教育界やビジネスでプロダクションユースとして使われる OpenStack を構築しています。

- Joe Topjian. 彼は Cybera で複数のクラウドの設計と構築を行って来ました。 Cybera は、非営利でカナダのアルバータ州の起業家や研究者を支援する電子情報インフラを構築しています。また、システムアーキテクトとしてこれらのクラウドの維持・運用を活発に行なっており、その経験からクラウド環境でのトラブルシューティングの豊富な知識を持っています。

この本の作成に参加するには

この本の元は人が集まったイベントで作成されましたが、今やこの本はみなさんも貢献できる状態になっています。 OpenStack のドキュメント作成は、バグ登録、調査、修正を繰り返して行うというコーディングの基本原則に基いて行われています。我々はこの本のソースコンテンツを Github にも置いており、レビューシステムである OpenStack Gerrit 経由で協力をお待ちしています。この本の O'Reilly 版では、我々は O'Reilly の Atlas システムを使用していますが、ソースコンテンツは Github にも格納され、コントリビュータ間での共同作業ができるようになっています。

OpenStack ドキュメント作成プロジェクトに参加する方法については、Documentation How To (http://wiki.openstack.org/Documentation/HowTo) を見て下さい。

バグを見つけたが、どのように直せばよいか分からない場合や本当にドキュメントのバグか自信が持てない場合は、 OpenStack Manuals (http://bugs.launchpad.net/openstack-manuals) にバグを登録して、バグの Extra オプションで ops-guide タグを付けて下さい。 ops-guide タグは、そのバグがこのガイドに関するものであることを示します。どのように直せばよいか分かる場合には、そのバグの担当者を自分に割り当てることもできます。また、OpenStack doc-core チームのメンバーがドキュメントバグを分類することもできます。

第1章 プロビジョニングとデプロイメント

自動デプロイメント 13
自動環境設定 16
リモート管理 17

クラウドのスケーラビリティにおける重要な部分の一つは、クラウドを運用するのに必要な労力にあります。クラウドの運用コストを最小化するために、デプロイメントと環境設定を自動化する基盤を構築して利用しましょう。

この基盤には、初期構成のオペレーティングシステム自動的にインストールすること、その後すべてのサービスの環境設定を自動的かつ集中的に調整することが含まれます。これにより、手操作の労力と誤りが介在する余地を減らすことができます。

自動デプロイメント

自動デプロイメントシステムは、新しいサーバーに対して、物理的なラッキング、MAC アドレスへの IP アドレスの割り当て、電源設定といった必要最小限の手操作の後、人手の介在なしに、オペレーティングシステムのインストールと環境設定を行います。典型的な方法では、PXE ブートと TFTP サーバーを使った仕組みを元にしてオペレーティングシステムの基本的なインストールを行い、その後、自動環境設定管理システムに制御を渡します。

Ubuntu と Red Hat Linux はどちらも、preseed や kickstart といった、ネットワークブートの後に利用できる、オペレーティングシステムを設定するための仕組みを持っています。これらは、典型的には自動環境設定システムを自力で立ち上げるために使われます。他の方法としては、systemimager のようなイメージベースのオペレーティングシステムのデプロイメント手法を使うこともできます。これらの手法は、例えば物理インフラと制御サービスを分離するために仮想マシンを使う時のように、いずれも仮想化基盤といっしょに使うことができます。

デプロイメントの計画を立てる際には、後から修正するのはとても困難な、いくつかの重要な領域に集中してください。

ディスクのパーティショニングと RAID

どんなオペレーティングシステムでも、もっとも根本的な部分として、それがインストールされるハードディスクがあります。

サーバーのハードディスクに対して、以下の環境設定を完了させなければなりません。

- パーティショニング
- RAID アレイへの追加

もっとも簡単な選択肢は、1 台のハードディスクを 2 つのパーティションに分割することです。

- ファイルシステム
- スワップ領域

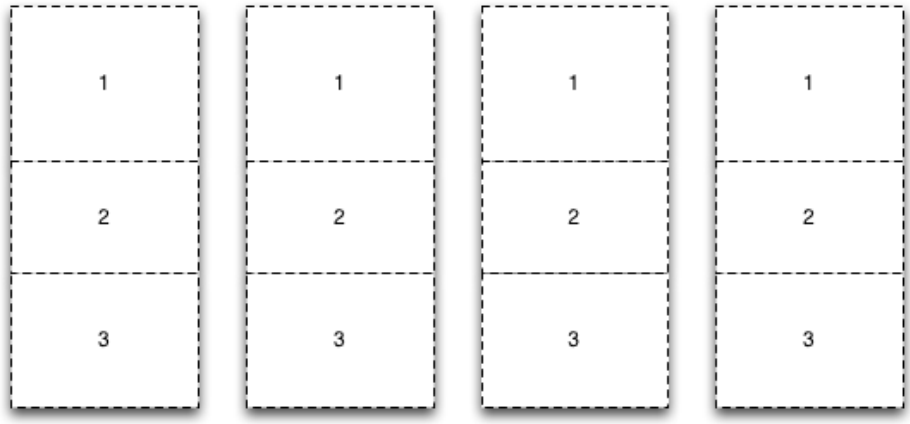
この場合は RAID は使用しません。

注記

この選択肢は、ハードディスクが故障するとサーバー全体がダウンしてしまうため、商用環境には推奨されません。代わりに、2 台以上のディスクを使用することを推奨します。使用するディスクの台数によって、構成する RAID アレイの種類が決まります。

以下に挙げる複数のディスクの選択肢から選ぶことを推奨します。

- オプション 1: 次の図に示すように、すべてのハードディスクをまったく同じようにパーティショニングします。



  このオプションでは、パーティションごとに異なる RAID アレイにおくことができます。例えば、ディスク 1 とディスク 2 のパーティション 1 を /boot パーティションのミラーとして、すべてのディスクのパーティション 2 をルートパーティションのミラーとして、すべてのディスクのパーティション 3 を RAID10 アレイの上の cinder-volumes の LVM パーティションとして割り当てることができます。

  この例にあるディスク 3 と 4 のパーティション 1 のように使用しないパーティションが残ることになるかもしれないとしても、ディスク領域の利用率を最大化することができます。すべてのディスクがすべてのタスクで利用されることにより、I/O 性能が問題になるかもしれません。

- オプション 2: すべてのディスクを 1 つの大きな RAID アレイに追加します。ここでは、ソフトウェア RAID でもハードウェア RAID でもかまいません。この大きな RAID アレイを boot、root、swap、そして LVM 領域に分割します。この選択肢はシンプルですべてのパーティションを利用することができますが、I/O 性能に悪影響があるかもしれません。

- オプション 3: 全ディスク領域を特定のパーティションで占有させます。例えば、ディスク 1 と 2 を RAID1 ミラーで boot、root、swap パーティションに割り当て、そしてディスク 3 と 4 は、すべてをやはり RAID1 ミラーの LVM パーティションに割り当てます。この場合は、I/O を特定の目的に集中させるため、より良いディスク I/O 性能

を得ることができるでしょう。ただし、LVM パーティションはだいぶ小さくなります。

ほとんどのアーキテクチャ上の選択のように、正しい答えは環境に依存します。

ネットワーク設定

ネットワーク設定は、この本の複数の部分にまたがるとても大きな話題です。ここでは、用意したサーバーが PXE ブートできることと、デプロイメントサーバーと正常に通信できることを確認しておいてください。

例えば、PXE ブートの際には、通常は VLAN の設定は行えません。さらに、通常は bonding された NIC から PXE ブートを行うこともできません。このような状況の場合、クラウド内でのみ通信できるネットワークで、シンプルな 1Gbps のスイッチを使うことを検討してください。

自動環境設定

自動環境設定管理の目的は、人間の介在なしにシステムの一貫性を確立し、維持することにあります。同じクラウド環境を毎回繰り返し作るために、デプロイメントにおける一貫性を維持したいでしょう。自動環境設定管理ツールを正しく利用することによって、デプロイメントと環境設定の変更を全体に展開する作業を単純にすることに加えて、クラウドのシステムを構成するコンポーネントが特定の状態にあるように保証することができます。

これらのツールは、完全に繰り返して動作可能であるため、変更点のテストやロールバックを行うことができます。OpenStack コミュニティでは、この手法で多くの作業が便利に実行されています。設定管理ツールの 1 つである Puppet の場合、OpenStack 用のモジュールも提供されています。

設定管理システムと、それによって管理する項目は、切っても切れない関係にあります。自動的に管理したい項目と管理したくない項目のすべてを注意深く検討するべきです。

リモート管理

経験上、ほとんどの事業者はクラウドを動かすサーバーのすぐ横に席があるわけではありません。また、多くの人が必ずしもデータセンターに行くことを楽しんでいるわけではありません。OpenStack は完全にリモートから環境設定できますが、時にこの通りにならないこともあります。

この場合、OpenStack が動くノードに対して外側からアクセスできるようにすることが重要です。ここでは、IPMI プロトコルがデファクトスタンダードです。完全自動のデータセンタを実現するために、IPMI をサポートしたハードウェアを購入することを強く推奨します。

さらに、リモート電源管理装置も検討してください。通常、IPMI はサーバーの電源状態を制御しますが、サーバーが接続されている PDU にリモートアクセスできれば、すべてが手詰まりに見えるような状況で非常に役に立ちます。

第2章 クラウドコントローラーの設計

ハードウェアの考慮事項 20
サービスの分離 22
データベース 22
メッセージキュー 22
Application Programming Interface (API) 23
API 拡張 23
スケジューラー 25
イメージ 25
ダッシュボード 26
認証と認可 26
ネットワークの考慮事項 28

OpenStack は非常に水平的にスケーラブルに設計されています。これにより、すべてのサービスを広く分散させることができます。しかし、この話を単純にするために、このガイドでは単一のクラウドコントローラーの概念を使い、より集中的な性質のサービスとして議論することにしました。

全体的なアーキテクチャの詳細については、7章参考アーキテクチャ [63]の章を参照してください。

このガイドに記述されているように、クラウドコントローラーは、データベース、メッセージキューサービス、認証・認可サービス、イメージ管理サービス、ユーザーダッシュボード、API エンドポイントを収容する１台のノードです。

クラウドコントローラーは、複数ノードで構成される OpenStack デプロイメントに対する集中管理機能を提供します。典型的には、クラウドコントローラーは認証および、メッセージキューを通じたメッセージのやりとりを管理します。

私たちの例では、クラウドコントローラーが nova-* コンポーネント群を持ちます。これは、クラウドのグローバルな状態を表し、認証のようなサービスとやりとりし、クラウドの情報をデータベースで維持し、メッセージキューを通してすべてのコンピュートノードとストレージワーカーと通信し、 API アクセスを提供します。可用性やスケーラビリティのために、あるクラウドコントローラーで動作するそれぞれのサービスを別々のノードに分離することができます。

ハードウェアの考慮事項

クラウドコントローラー用のハードウェアはコンピュートノードのものと同じでかまいませんが、運用するクラウドのサイズとタイプに基づいて、もっと指定したいかもしれません。

クラウドコントローラーが管理するすべての、または一部のサービス、たとえばメッセージキューに対して仮想マシンを使うことも可能です。このガイドでは、すべてのサービスが直接クラウドコントローラー上で実行されるものと仮定します。

クラウドコントローラーのサーバーを正しくサイジングするため、および一部を仮想化するかどうか決定するため、以下を見積もるべきです。

- 予期されるインスタンス実行数
- コンピュートノードの数
- コンピュートサービスまたはストレージサービスにアクセスするユーザー数
- ユーザがクラウドへ REST API でアクセスするのかダッシュボードを使うのか
- ユーザーは外部のシステムに対して認証を行うのか（例えば、LDAP や Active Directory）
- 1 つのインスタンスがどのくらい実行され続けるのか

考慮事項	派生問題
同時に何インスタンス実行されますか？	データベースサーバーを負荷に応じてサイジングしてください。もし、多数のインスタンスが同時に状態を報告したり、CPU能力が必要な新規インスタンス起動のスケジューリングを行う場合は、１台のクラウドコントローラーを超えてスケールアウトしてください。
同時にコンピュートノードが何ノード実行されますか？	メッセージキューが正しくリクエストを処理することを保証し、適切にサイジングしてください。
どのくらいの数のユーザがAPIにアクセスしますか？	もし多数のユーザが複数のリクエストを発行するのであれば、クラウドコントローラーがそれらを扱えるよう、CPU負荷を確認してください。
どのくらいの数のユーザが ダッシュボードにアクセスしますか？	ダッシュボードは、APIアクセスよりもさらに多くのリクエストを発行します。そのため、もしユーザに対するインタフェースがダッシュボードなのであれば、より多くのCPUを追加してください。
あなたのクラウドで、何個のnova-apiサービ	サービスごとに１コア割り当ててコントローラーをサイジングする必要があります。

考慮事項	派生問題
スを同時に実行しますか？	
１つのインスタンスがどのくらい実行され続けますか？	インスタンスの起動と停止は コンピュートノードに負荷をかけますが、それだけでなく、すべてのAPI処理とスケジューリングの必要性のために、コントローラーノードにも負荷をかけます。
認証システムは外部に確認を行いますか？	クラウドコントローラーと外部の認証システムの間のネットワーク接続性が良好であることと、クラウドコントローラーがリクエストを処理するのに十分なCPU能力があることを確認してください。

サービスの分離

私たちの例には、一か所にすべての集中的なサービスが含まれていますが、サービスを異なる物理サーバーに分離するのは可能ですし、多くの場合、本当に良いアイデアです。以下は、私たちが見たデプロイメントのシナリオと、その妥当性です。

glance-* サーバーを、 swift-proxy サーバーの上で実行する	このデプロイメントでは、オブジェクトストレージのプロキシサーバーの I/O の空きは十分でした。そして、Glance のイメージ配信部分は、それが利用しているオブジェクトストレージと同じ物理マシンの上にあり、バックエンドに対して良好な接続性をもつことにより、恩恵を得られました。
専用の中央データベースサーバーを運用する	このデプロイメントでは、すべてのサービスに対するデータベースサービスを提供する専用サーバを設置しました。これにより、データベースサーバーのアップデートを分離でき、運用がシンプルになりました。また、フェイルオーバーのためのスレーブデータベースサーバーの設置が単純になりました。
サービスごとに 1 台の VM を動作させる	このデプロイメントでは、KVM を実行しているサーバー群において集中的なサービス群を動作させています。 各サービス（nova-scheduler、rabbitmq、データベース等）に対して専用 VM が作成されています。これにより、各仮想マシンにかかった負荷（構築中にはよく分からなかったこと）に応じてリソースをチューニングすることができました。
外部ロードバランサーを使う	このデプロイメントでは、組織内に高価なハードウェアロードバランサーを持っていました。彼らは複数の nova-api とswift-proxy サーバーを異なる物理サーバーで動作させ、それらの間の振り分けにロードバランサーを使いました。

いつも問題になる一つの選択は、仮想化するかどうかです。nova-compute、swift-proxy そして swift-object といったサーバーは仮想化するべきではありません。しかし、コントローラーサーバーは問題なく仮想化することができます。通常、性能ペナルティは、単により多くのサービスを動作させることにより相殺することができます。

データベース

OpenStack Compute のほとんどの中央サービスは、現在は nova-compute も含めて、状態の情報を保存するのにデータベースを利用します。この機能を喪失することはエラーにつながります。ですので、耐故障性のために、なんらかの方法でデータベースをクラスタ化することを推奨します。

メッセージキュー

OpenStack Compute のほとんどのサーバーは、メッセージキューを利用してお互いに通信しあっています。一般論として、このメッセージ

キューが故障するかアクセスできなくなると、Nova のクラスタは、最後のメッセージが送信されたところの情報でとどまったまま、ゆっくり停止し、「リードオンリー」状態になります。したがって、メッセージキューもクラスタ化することを推奨します。RabbitMQ には、これを行う機能が組み込まれています。

Application Programming Interface (API)

直接であっても、コマンドラインクライアントであっても、Web ベースのダッシュボードであっても、一般ユーザーからのアクセスはAPIサービスを利用します。 API リファレンスは http://api.openstack.org/ にあります。

Amazon EC2 互換 API をサポートしたいか、OpenStack API だけなのか、選択しなければなりません。両方の API を運用する場合、イメージとインスタンスを参照する際の見え方が違うことが一つの論点になります。

例えば、EC2 API では、16 進数を含む ID を使ってインスタンスを参照するのに対して、OpenStack API では名前と数値を使います。類似の話題として、EC2 API は仮想マシンに接続するのに DNS エイリアスに頼る傾向がありますが、OpenStack では典型的には IP アドレスを使います。

もし OpenStack が正しく設定されていなければ、不正な DNS エイリアスしか得られないことによりユーザがインスタンスに接続できないという事態が容易に発生します。こうした事情にもかかわらず、EC2 互換性はユーザーをお使いのクラウドに移行させるのに役立ちます。

データベースやメッセージキューのように、１台より多くの API サーバー を置くのは良いことです。nova-api サービスを高可用にするために、伝統的な HTTP 負荷分散技術を利用することができます。

API 拡張

API Specifications (http://docs.openstack.org/api/api-specs.html) に OpenStack API コアのアクション、ケーパビリティ、メディアタイプが定義されています。クライアントはいつでもこのコアAPIが使えることに依存することができますし、実装者はいつでも完全にサポートすることが求められます。コアAPIの厳守を要求することにより、クライアントは同じAPIの複数の実装とやりとりする際に、最小限のレベルの機能に依存することができます。

OpenStack Compute API は拡張可能です。ある拡張は、ある API にコア定義を超えたケイパビリティを追加します。新機能、新しい MIME タイプ、アクション、状態、ヘッダ、パラメータ、そしてリソースの導入は、コア API の拡張によって達成することができます。これにより、API に対してバージョンを変更することなく新機能を導入することができ、ベンダー固有の特定の機能を導入することもできます。

スケジューラー

さまざまなサイズ(異なる フレーバー)の仮想マシンを、異なる能力の物理 nova-compute ノードに配置するのは、一般的にコンピュータ科学でパッキング問題として研究されている難しい問題です。

この問題を解くことは本書のスコープ外ですが、この問題に対しては様々なテクニックを使うことができます。様々なスケジューリングの選択肢に対応するため、OpenStack Compute はいくつかのタイプのスケジューリングドライバーを提供しています。詳細は、OpenStack 設定レファレンスのスケジューリング (http://docs.openstack.org/havana/config-reference/content/section_compute-scheduler.html) を参照してください。

可用性の目的のため、もしくは非常に大規模な環境や非常に頻繁にスケジューラーが呼び出される環境の場合には、複数の nova-scheduler サービスを動作させることを検討するべきです。nova-scheduler は完全にメッセージキューを使って通信を行うため、特別な負荷分散は必要ありません。

イメージ

OpenStack Image Catalog and Delivery サービスは、 glance-api と glance-registry の２つの部分から構成されています。前者は コンピュートノードへのイメージの配送に責任を持ち、コンピュートノードはバックエンドからイメージをダウンロードするために使います。後者は、仮想マシンのイメージに関連するメタデータ情報を管理し、データベースを必要とします。

glance-api は、バックエンドの選択を可能にする抽象化レイヤであり、現在は以下をサポートしています。

- OpenStack Object Storage。イメージをオブジェクトとして保存することができます。
- ファイルシステム。イメージをファイルとして保存するのに、任意の伝統的なファイルシステムを使用します。
- S3。Amazon S3 からイメージを取得することができます。
- HTTP。Webサーバーからイメージを取得することができます。このモードでは、イメージの書き込みはできません。

OpenStack Object Storage サービスがある場合には、イメージを保存するスケーラブルな場所としてこれを利用することを推奨します。また、OpenStack 経由で新しいイメージをアップロードする必要がある場合には、他の選択肢としては、十分な性能を持ったファイルシステムや Amazon S3 があります。

ダッシュボード

OpenStack ダッシュボードは、Apache httpdの中で実行される Python の Web アプリケーションとして実装されています。したがって、そこから ネットワーク経由で（admin エンドポイントを含む）API サーバーにアクセスできるという条件の下、他の任意の Web アプリケーションと同じように取り扱うことができます。

認証と認可

OpenStack の認証・認可を支える概念は、よく理解され、類似の性質のシステムで広く使用されているものから得られています。ユーザは、認証に使えるクレデンシャルを持ち、１つ以上のグループ(プロジェクトまたはテナントして知られています)のメンバとなることができます。

例えば、クラウドのユーザは自分の現在のグループに属するインスタンスのみが見えるのに対して、クラウドの管理者はそのクラウドのすべてのインスタンスの一覧をとることができるでしょう。利用可能なコア数、ディスク容量等のリソースのクォータはプロジェクトに対して関連づけられています。

OpenStack Identity Service（Keystone）は、認証の判定とユーザの属性情報を提供する場となり、他の OpenStack サービスから認可のために使用されます。ポリシーは policy.json で記述されます。これらを設定するための情報については、9章プロジェクトとユーザーの管理 [83] を参照してください。

Identity サービスは、バックエンド認証と情報保持のために種々のプラグインをサポートしています。これらの選択肢は、純粋なストレージの選択から、外部認証システムにわたり、現在は以下が含まれています。

- インメモリキーバリューストア
- SQL データベース
- PAM
- LDAP

多くのデプロイメントで SQL データベースが使われていますが、既存の認証インフラとインテグレーションする必要のある環境では、LDAP もポピュラーな選択肢です。

ネットワークの考慮事項

クラウドコントローラーは非常に多くのサービスを取り扱うため、到来するトラフィックを処理できなければなりません。例えば、クラウドコントローラー上に OpenStack Image サービスを乗せることにした場合、そのクラウドコントローラーは許容可能な速度でイメージを転送できなければなりません。

別の例としては、クラウドコントローラーがすべてのインスタンスのゲートウェイとなるような単一ホストネットワークモデルを使うことにした場合、クラウドコントローラーは外部インターネットとあなたのクラウドの間でやりとりされるすべてのトラフィックを支えられなければなりません。

10GbE のような高速な NIC を使うことを推奨します。また、10GbE NIC を2枚使って ボンディングすることもできます。束ねられた 20Gbps の速度をフルに使うことはできないかもしれませんが、異なる送信ストリームは異なる NIC を使います。例えば、クラウドコントローラーが2つのイメージを送信する場合、それぞれのイメージが別の NIC を使い、10Gbps の帯域をフルに使うことができます。

第3章 スケーリング

出発点 29
コントローラーノードの追加 31
クラウドの分離 32
スケーラブルハードウェア 35

あなたのクラウドが成功していれば、いずれ増加する需要に対応するためにリソースを追加しなければなりません。OpenStack は水平的にスケールできるよう設計されています。より大きなサーバーに取り換えるのではなく、もっと多くのサーバーを購入すればよいのです。理想的には、機能的に同一のサービス間でスケールアウトして負荷分散します。

出発点

クラウドのスケーラビリティを決定することと、それをどう改善するのかは、多くの変数のバランスをとる問題です。万人のスケーラビリティ目標に適した解決策は存在しませんが、いくつかの指標を観察しておくと役に立つでしょう。

ほとんどの場合の出発点は、クラウドのコア数です。なんらかの比率を掛けることにより、実行される仮想マシン(VM)の数の期待値についての情報を得られます。（(オーバーコミット率 × cores) / インスタンスごとの仮想コア数), 必要なストレージ容量は（フレーバーごとのディスクサイズ × インスタンス数). あなたのクラウドにどの程度の追加機材が必要なのか、これらの比率で判断することができます。

OpenStack のデフォルトフレーバー:

名前	仮想コア数	メモリ	ディスク	エフェメラル
m1.tiny	1	512 MB	1 GB	0 GB
m1.small	1	2 GB	10 GB	20 GB
m1.medium	2	4 GB	10 GB	40 GB
m1.large	4	8 GB	10 GB	80 GB
m1.xlarge	8	16 GB	10 GB	160 GB

以下の構築例では、（200 / 2）× 16 = 1600 VM インスタンスをサポートし、 /var/lib/nova/instances 以下に80TBのストレージ領域が必要なものとします。

- 200物理コア
- ほとんどのインスタンスのサイズは m1.medium（仮想コア数2、ストレージ50GB)とします。
- デフォルトの CPU オーバーコミット率（cpu_allocation_ratio in nova.conf）は 16:1 とします。

しかし、APIサービスやデータベースサーバー、MQサーバーがおそらく遭遇する負荷を見積もるためには、コア数以外の検討も行う必要があります。クラウドの利用パターンも考慮しなければなりません。

特定の例としては、マネージドWebホスティングプラットフォームをサポートするクラウドと、コードコミットごとに仮想マシンを１つ作成するような開発プロジェクト用の結合テストを動かしているクラウドを比較してみましょう。前者では、VMを作成する重い処理は数か月に一度しか発生しないのに対して、後者はクラウドコントローラーに定常的に重い処理を発生させます。一般論として、VMの平均寿命が長いということは、クラウドコントローラーの負荷が軽いことを意味するため、平均的なVMの寿命を検討しなければなりません。

仮想マシンの起動、停止以外では、ユーザーアクセス、特に nova-api と関連データベースへのアクセスの影響を検討しなければなりません。インスタンス一覧を取得する処理は膨大な量の情報を収集しますし、ユーザーがこの処理を頻繁に行うと、ユーザー数が多いクラウドではこの負荷が著しく上昇します。この負荷は、ユーザーが知らずに発生します。つまり、ブラウザのOpenStack ダッシュボードのインスタンスタブを開いたままにすると、30秒ごとに仮想マシンの一覧が更新されます。

これらの要素を検討した後、クラウドコントローラーにどのくらいのコア数が必要なのか決定することができます。上記の注意事項において、典型的には、ラック1本分のコンピュートノードに対して8コア、メモリ8GBのサーバーで充分です。

ユーザーの仮想マシンの性能のために、ハードウェアのキーになるスペックも考慮に入れなければなりません。予算と性能への需要を検討するのです。例としては、ストレージ性能(スピンドル / コア)、メモリ(メモリ量 / コア)、ネットワーク帯域(Gbps / コア)、そして全般的なCPU性能（CPU / コア)です。

クラウドをどうスケールさせるのか決定するために観察すべき指標については、13章ロギングと監視［151］を参照してください。

コントローラーノードの追加

クラウドの水平的な拡張は、ノード追加によって容易に実行することができます。コンピュートノードの追加は簡単です。新規のコンピュートノードは既存のシステムから簡単に認識されます。しかし、クラスタを高可用なものにするためには、設計の際にいくつかの重要なポイントを検討しなければなりません。

クラウドコントローラーは、いくつかの異なるサービスを実行することを思い出してください。拡張のための新しいサーバーには、 nova-scheduler や nova-console のようなメッセージキューを用いて内部でのみ通信するサービスをインストールすることができます。しかし、他の不可欠な部分についてはもっと注意が必要です。

ダッシュボードやnova-api 、Object Storage proxy のようなユーザー向けのサービスは負荷分散するべきです。任意の標準的なHTTP負荷分散方法（DNSラウンドロビン、ハードウェアロードバランサ、Pound や HAProxy のようなソフトウェア）を使ってください。ダッシュボードに関する注意事項の一つは、VNC proxy が使う WebSocket プロトコルです。これは、L7ロードバランサで苦労することになるかもしれません。以下のリンクも参照してください。 Horizon session storage (http://docs.openstack.org/developer/horizon/topics/deployment.html#session-storage).

nova-api や glance-api のようなサービスは、設定ファイルのフラグを変更することによって複数プロセスで処理させるように設定できます。これによって一台のサーバの複数のコアの間で処理を共有できるようになります。

MySQLの負荷分散にはいくつかのオプションがありますし、RabbitMQ はクラスタリング機能を持っています。これらや他の多くのサービスの設定方法に関する情報は運用の章で見つけることができます。

クラウドの分離

クラウドを分離するためには、次に挙げる OpenStack の方法を使います。 セル，リージョン，ゾーン そしてホストアグリゲート です。これらは以下の表で述べるように、それぞれ異なる機能を提供します。

	セル	リージョン	アベイラビリティゾーン	ホストアグリゲート
用途	コンピュート資源に対する単一の API エンドポイント、もしくは２段階スケジューリングが必要な場合	リージョンごとに別々のAPIエンドポイントが必要で、リージョン間で協調する必要がない場合	物理的な隔離や冗長性のために、Nova デプロイメントの中で論理的な分離が必要な場合	共通の機能を持ったホストのグループに対してスケジューリングしたい場合
例	複数サイトで構成されるクラウドで、仮想マシンを「任意のサイト」または特定のサイトにスケジューリングしたい場合	複数サイトで構成されるクラウドで、仮想マシンを特定のサイトに対してスケジューリングでき、かつ共有インフラを利用したい場合	単一サイトのクラウドで、分離された電源供給ラインを持つ設備で構成される場合	トラステッドコンピューティング機能に対応したホスト群に対してスケジューリングしたい場合
オーバーヘッド	• nova-cells という新しいサービスが必要 • 各セルには nova-api 以外の全 nova サービスが必要	• リージョン毎に別々のAPIエンドポイントが必要 • 各リージョンにフルセットのNovaサービスが必要	• nova.conf の設定変更が必要	• nova.conf の設定変更が必要
共有サービス	Keystone nova-api	Keystone	Keystone すべての Nova サービス	Keystone すべての Nova サービス

この選択肢の表は２つに分けると一番良く理解することができます。１つは、別々の Nova デプロイメントが動作します（セルとリージョン）。もう一つは、単一の Nova デプロイメントを分割するだけです。(アベイラビリティゾーンとホストアグリゲート).

セルとリージョン

OpenStack Compute のセルは、より複雑な技術を持ち込むことなしに、また既存のNovaシステムに悪影響を与えることなしに、クラウドを分散された環境で運用することができるように設計されています。１つのクラウドの中のホストは、セルと呼ばれるグループに分割されます。セルは、木構造に構成されてます。最上位のセル（「API セル」）はnova-apiサービスを実行するホストを持ちますが、nova-compute サービスを実行するホストは持ちません。それぞれの子セルは、nova-apiサービス以外の、普通のNovaシステムに見られる他のすべての典型的な nova-* サービスを実行します。それぞれのセルは自分のメッセージキューとデータベースサービスを持ち、またAPIセルと子セルの間の通信を制御するnova-cellsサービスを実行します。

これによって、複数のクラウドシステムに対するアクセスを、１つのAPIサーバで制御することができます。通常のnova-schedulerによるホストの選択に加えて、第二段階のスケジューリング(セルの選択)を導入することにより、仮想マシンを実行する場所の制御の柔軟性が大きく向上します。

セルをリージョンと比較してみましょう。リージョンは、クラウドごとに別々のAPIエンドポイントを持ち、より関連性の低い分離を実現できます。サイトをまたがってインスタンスを実行したいユーザーは、明示的にリージョンを指定しなければなりません。しかし、新しいサービスを実行するという、さらなる複雑さは必要ありません。

現在のところ、OpenStack ダッシュボード（Horizon） は１つのリージョンだけを操作対象とします。したがって、リージョンごとにダッシュボードサービスを実行するべきです。リージョンは、高度な故障耐性を実現しつつ、複数の OpenStack Compute のシステム間でインフラのいくらかの部分を共有するための堅牢な方法です。

アベイラビリティゾーンとホストアグリゲート

一つの nova デプロイメントを分割するのに、アベイラビリティゾーン、ホストアグリゲート、もしくはその両方を使うことができます。

アベイラビリティゾーンは、ホストアグリゲートを利用して実装されており、ホストアグリゲートと同様の方法で設定します。

しかし、アベイラビリティゾーンとホストアグリゲートは別の理由で使用します。

- アベイラビリティゾーン: OpenStack Compute ホストを論理グループにまとめて、（独立した電源系統やネットワーク装置を使うことなど

で）他のアベイラビリティゾーンとのある種の物理的な分離や冗長性を実現できます。

サーバー毎に、指定したサーバーが所属するアベイラビリティゾーンを定義します。一般に、アベイラビリティゾーンは共通の性質を持つサーバーの集合を識別するために使われます。例えば、データセンターの一部のラック群がある一つの独立した電源系統につながっている場合には、これらのラック群に設定されたサーバーを専用のアベイラビリティゾーンに入れることができます。また、アベイラビリティゾーンは異なるクラスのハードウェアを分割するのにも使うことができます。

ユーザーはリソースを作成する際に、インスタンスを作成するアベイラビリティゾーンを指定することができます。これによって、クラウドの利用者は、自分のアプリケーションのリソースが異なるマシンに分散して配置されることを保証でき、ハードウェア故障が発生した場合でも高可用性を達成することができます。

- ホストアグリゲート: 負荷分散やインスタンスの分散配置のために、OpenStack Compute のデプロイメントを論理グループに分割することができるようになります。ホストアグリゲートを使って一つのアベイラビリティゾーンをさらに分割することもできます。例えば、ホストアグリゲートを使うことで、一つのアベイラビリティゾーンを、ストレージやネットワークなどの共通のリソースを共有するホストのグループや、トラステッドコンピューティングハードウェアのような特別な性質を備えたホストのグループ、に分割することができるでしょう。

  ホストアグリゲートのよくある使い方は nova-scheduler で利用する情報を提供することです。例えば、ホストアグリゲートを使って、特定のフレーバーやイメージを共有するホストの集合を作成することができます。

注記

以前はすべてのサービスにアベイラビリティゾーンがありました。現在では nova-compute サービスだけが自分のアベイラビリティゾーンを持っています。 nova-scheduler, nova-network, nova-conductor といったサービスは常にすべてのアベイラビリティゾーンにまたがる形になります。

次の操作のいずれかを実行すると、これらのサービスは内部アベイラビリティゾーン (CONF.internal_service_availability_zone) 内に表示されます。

- nova host-list (os-hosts)
- euca-describe-availability-zones verbose
- nova-manage service list

内部アベイラビリティゾーンは（非冗長モードの）euca-describe-availability_zones では表示されません。

CONF.node_availability_zone は CONF.default_availability_zone に名前が変更されました。このオプションは nova-api サービスと nova-scheduler サービスでのみ使用されます。

CONF.node_availability_zone は今も機能しますが、非推奨扱いです。

スケーラブルハードウェア

OpenStack をインストールしてデプロイするのに有用な情報源が既にいくらか存在していますが、自分のデプロイメントで事前に計画を立てておくことは非常に重要です。このガイドでは、OpenStack用にラックを少なくとも１本用意しておくことを想定してしますが、いつ、何をスケールさせるのかについてのアドバイスも行っています。

ハードウェア調達

「クラウド」とは、サーバーが作成されたり終了されたりするという揮発性の環境であると説明されてきました。これは正しいかもしれませんが、あなたのサーバーが揮発性でなければならないという意味ではありません。クラウドを構成するハードウェアを安定させ、正しく設定されていることを保障するということは、あなたのクラウド環境が稼働中であり動作していることを意味します。基本的に、ユーザーが必要な時に確保でき揮発性なものとして扱えるようなクラウドを運営できるよう、安定したハードウェア環境を作ることに注力してください。

OpenStack は、この本の参考アーキテクチャで使われている Ubuntu 12.04 のように、OpenStack と互換性のある Linux ディストリビューションでサポートされたハードウェアにデプロイできます。

ハードウェアはまったく同じでなければいけないことはありませんが、少なくともインスタンスマイグレーションが可能であるような同じタイプのCPUを装備しているべきです。

OpenStack に使うのに推奨される典型的なハードウェアは、ほとんどのハードウェアベンダが提供している、「金額に見合う価値をもった（value-for-money）」とても標準的なハードウェアです。調達を「コンピュート」や「オブジェクトストレージ」そして「クラウドコントローラー」のような構成要素に分割し、それぞれ必要な数だけ要求するのは分かりやすい方法でしょう。これ以上費用をかけることができない場合でも、代わりに、もし既存のサーバーがあって、これらが性能や仮想化技術の要件を満たしていれば、高い確率で OpenStack を動作させられます。

キャパシティプランニング

OpenStackは、単純な方法でサイズを拡大できるように設計されています。 スケーラビリティ の章の、特にクラウドコントローラーのサイジングに関する検討を考慮に入れ、必要に応じて追加のコンピュートノードやオブジェクトストレージノードを調達できるようにすべきです。新しいノードは、既存ノードと同じスペックである必要はありませんし、同じベンダーである必要すらありません。

コンピュートノードについては、nova-scheduler がコア数やメモリ量のサイジングに関する違いを吸収しますが、CPUの速度が違うことによって、ユーザーの使用感が変わることを考慮するべきです。オブジェクトストレージノードを追加する際には、 そのノードのケーパビリティ を反映するウェイト を指定するべきです。

リソース利用状況の監視とユーザー増加の監視によって、（追加機材の）調達時期を知ることができます。監視の章でいくつかの有用な監視項目を詳しく解説します。

エージング試験

サーバーは、そのライフタイムの最初と最後にハードウェア故障の確率が高くなります。結論として、初期故障を誘発する適切なエージングテストを行うことによって、運用中の故障に対応するための多くの労力を避けることができます。一般的な原則は、限界まで負荷をかけることです。エージング試験の例としては、数日間にわたってCPUやディスクベンチマークを走行させることが含まれます。

第4章 コンピュートノード

CPU の選択 37
ハイパーバイザーの選択 38
インスタンスストレージのソリューション 38
オーバーコミット 43
ロギング 43
ネットワーク 44

コンピュートノードは OpenStack Compute クラウドのリソースの中核を構成し、インスタンスを動作させるためのプロセッシング、メモリ、ネットワーク、ストレージの各リソースを提供します。

CPU の選択

コンピュートノードの CPU 種別は非常に重要な選択です。まず、CPU は Intel チップでは VT-x、 AMD チップでは AMD-v の仮想化に対応している必要があります。

CPU のコア数も選択に影響します。最近のCPUでは最大12コアあるのが一般的です。さらに、CPU がハイパースレッディングをサポートしていれば、12コアは2倍の24コアになります。複数のCPUを持つサーバーを購入すれば、コア数はさらに掛け算で増えます。

CPUでハイパースレッディングを有効にするかどうかはユースケースに依存します。ハイパースレッディングがオン、オフの両方の状態であなたの用途に応じた負荷で性能試験を行い、どちらがユースケースに適しているかを判断することをお薦めします。

ハイパーバイザーの選択

OpenStack Compute は多数のハイパーバイザーをサポートしており、その程度も様々です。 サポートされているハイパーバイザーは、KVM, LXC, QEMU, UML, VMWare ESX/ESXi, Xen, PowerVM, Hyper-V です。

おそらく、ハイパーバイザーの選択で最も重要な要素は、現在の使用法やこれまでの経験でしょう。それ以外では、同等の機能の実用上の懸念、ドキュメント、コミュニティでの経験量などあると思います。

例えば、 KVM は OpenStack コミュニティでは最も多く採用されているハイパーバイザーです。 KVM 以外では、Xen、LXC、VMWare、Hyper-V を使っているシステムが、（サポート）リストにある他のハイパーバイザーよりは多いです。しかしながら、これらのハイパーバイザーはどれもある機能のサポートがなかったり、OpenStack と組み合わせての使い方に関するドキュメントが最新版に追従していなかったりします。

ハイパーバイザー選択の参考になる情報は、Hypervisor Support Matrix (https://wiki.openstack.org/wiki/HypervisorSupportMatrix) と 設定レファレンス (http://docs.openstack.org/trunk/config-reference/content/section_compute-hypervisors.html) です。

注記

ホストアグリゲートやセルを使うと一つの OpenStack システムで複数のハイパーバイザーを動かすこともできますが、一つのコンピュートノードで同時に実行できるのは1種類のハイパーバイザーだけです。

インスタンスストレージのソリューション

コンピュートクラスタを調達する際に、作成したインスタンスの（仮想）ディスク用のストレージを決めなければいけません。この一時ストレージの提供方法には主に3つのアプローチがあり、その意味を理解することが重要です。

次の3つの方法があります。

- コンピュートノード外のストレージ （共有ファイルシステム）
- コンピュートノード上のストレージ （共有ファイルシステム）

- コンピュートノード上のストレージ （非共有ファイルシステム）

一般的には、ストレージを選択する際には次のような質問をされます。

- 実現したいプラッター数（ディスク容量）はどれくらいか？
- ネットワークアクセスがあったとしても、ディスク数が多い方が良い I/O 性能が得られるか？
- 何があなたが目指すコストパフォーマンスのシナリオはどれか？
- 運用上ストレージをどのように管理したいのか？

コンピュートノード外のストレージ （共有ファイルシステム）

多くの運用者はコンピュートホストとストレージホストを分離して使用しています。コンピュートサービスとストレージサービスには異なる要件があり、コンピュートホストでは通常はストレージホストよりも多くのCPUとRAMが必要です。そのため、一定の予算の中では、コンピュートホストとストレージホストで異なる構成として、コンピュートホストに多くのCPUとRAMを持たせ、ストレージホストに多くのブロックストレージを持たせるのは、理にかなっています。

また、コンピュートホストとストレージホストを分離しておけば、コンピュートホストを「ステートレス」（状態を保持しないもの）として扱うことができます。これにより、コンピュートホストの管理を単純にすることができます。コンピュートホスト上で動作しているインスタンスがない限り、クラウドの他の部分に影響を与えずにそのノードをオフラインにしたり取り除いたりすることができます。

一方、クラウドの構築に使用できる物理ホスト数に制限があり、できるだけ多くのホストをインスタンスの実行に使えるようにしたい場合は、同じマシンでコンピュートホストとストレージホストを動作させるのは理にかなっています。

この方法では、実行中のインスタンスの状態を格納するディスクはコンピュートホスト外のサーバーに置かれます。この方法には以下のようなメリットもあります。

- コンピュートホストが故障した場合、通常インスタンスは簡単に復旧できます。
- 専用のストレージシステムを動作させることで、運用がシンプルになります。

- ディスク数がスケーラブルになります。
- 外部ストレージを他の用途と共有できる可能性があります。

この方法の主なマイナス面は以下の点です。

- 設計次第では、一部のインスタンスの I/O が非常に多い場合に、無関係のインスタンスに影響が出る場合があります。
- ネットワークを使用するため、性能低下が起こる可能性があります。

コンピュートノード上のストレージ （共有ファイルシステム）

この方法では、各 nova-compute ノードには多数のディスクが接続されますが、分散ファイルシステムにより各コンピュートノードのディスクは1つのマウントポイントにまとめられます。この方法の主なメリットは、追加のストレージが必要になった際に外部ストレージを利用してスケールできる点です。

しかし、この方法にはいくつかマイナス点があります。

- 分散ファイルシステムを動作させるため、非共有ストレージと比較してデータの局所性が失われます。
- 複数の物理ホストが関係するため、インスタンスの復旧が複雑になります。
- コンピュートノードの筐体サイズによって、コンピュートノードに搭載できるディスク数が制限されます。
- ネットワークを使用するため、性能低下が起こる可能性があります。

コンピュートノード上のストレージ （非共有ファイルシステム）

この方法では、各 nova-compute ノードには、そのホストで動作するインスタンスを収容するのに十分な量のディスクが接続されます。この方法には次の2つのメリットがあります。

- あるコンピュートノード上での I/O が非常に多い場合でも、他のコンピュートノードのインスタンスに影響がありません。
- I/O アクセスが直接行われるので、性能向上が図れます。

この方法には次のようなマイナス点があります。

- コンピュートノードが故障すると、そのノードで実行中のインスタンスが失われてしまいます。
- コンピュートノードの筐体サイズによって、コンピュートノードに搭載できるディスク数が制限されます。
- あるノードから別のノードへのインスタンスのマイグレーションが複雑になります。また、マイグレーション方法も開発が継続されるか分からない方法に依存することになります。
- 追加のストレージが必要になった際に、この方法はスケールしません。

ライブマイグレーションに関する問題

我々はライブマイグレーションはクラウドの運用に不可欠なものだと考えています。この機能により、インスタンスをある物理ホストから別の物理ホストに停止せずに移動し、コンピュートホストの再起動を必要とするアップグレードを実行することができるようになります。しかし、ライブマイグレーションを行うには共有ストレージがなければなりません。

ライブマイグレーションは、KVM ライブブロックマイグレーションとして知られる機能を用いて、非共有ストレージでも実行できます。以前のバージョンでの KVM と QEMU におけるブロックマイグレーションの実装は信頼性がないと思われていましたが、OpenStack とも互換性のある QEMU 1.4 と libvirt 1.0.2 には、信頼性が向上した新しいライブブロックマイグレーションの実装があります。しかしながら、このガイドの執筆陣は誰もライブブロックマイグレーションを使用したことがありません。

ファイルシステムの選択

共有ストレージを使ったライブマイグレーションをサポートしたい場合には、分散ファイルシステムを構成する必要があります。

次のような選択肢があります。

- NFS（Linux でのデフォルト）
- GlusterFS
- MooseFS

- Lustre

我々はこれら全ての実例を見たことがありますが、一番運用方法を知っているものを選択することをお薦めします。

オーバーコミット

OpenStack では、コンピュートノードの CPU と RAM をオーバーコミットすることができます。これにより、インスタンスの性能が下がるものの、クラウド上で動作可能なインスタンス数を増やすことができます。OpenStack Compute でのデフォルト値は次のようになっています。

- CPU 割当比: 16

- RAM 割当比: 1.5

CPU 割当比のデフォルト値 16 は、スケジューラーが1つのノードで物理コア1つあたり最大16個の仮想コアを割り当てることを意味します。例えば、ある物理ノードのコア数が12の場合、スケジューラが最大で192個の仮想コアをインスタンスに割り当てることになります（例えば、各インスタンスの仮想コアが4個の場合には、48インスタンス割り当てられます）。

同様に、RAM 割当比のデフォルト値 1.5 は、インスタンスに割り当てられた RAM の総量がその物理ノードで利用できるメモリ量の1.5倍未満であれば、スケジューラーがその物理ノードにインスタンスを割り当てることを意味します。

例えば、物理ノードに 48GB の RAM がある場合、そのノード上のインスタンスに割り当てられた RAM の合計が 72GB に達するまでは、スケジューラーはそのノードにインスタンスを割り振ることになります（例えば、各インスタンスのメモリが 8GB であれば、9 インスタンス割り当てられます）。

あなた自身のユースケースに合わせて、適切な CPU と RAM の割当比を選択しなければなりません。

ロギング

ロギングについては 「ロギング」 [43] で詳しく説明しています。しかし、ロギングはクラウドの運用を開始前に考慮しておくべき重要な検討事項です。

OpenStack は非常に多くの有用なログ情報を出力しますが、運用時にログ情報を有効活用するためには、ログを集積するログサーバーや、

（logstash といった）ログ解析/分析システムを用意することを検討すべきでしょう。

ネットワーク

OpenStack のネットワークは複雑で、検討すべき点がたくさんあります。 6章ネットワーク設計［55］を参照して下さい。

第5章 ストレージ選定

ストレージのコンセプト 45
ストレージバックエンドの選択 48
OpenStack Object Storage の注意事項 52

ストレージは OpenStack スタックの多くの部分に存在し、これらのタイプの違いにより経験豊富なクラウド技術者でさえ混乱する事があります。本章では、あなたのクラウドで設定可能な永続的ストレージに焦点を当てます。

ストレージのコンセプト

表5.1 OpenStackのストレージ

	エフェメラルストレージ	ブロックストレージ	オブジェクトストレージ
使用目的	OS を起動し、空き領域に記録する	永続的なストレージを仮想マシン（VM）へ追加する	データを保存する（VMイメージも含む）
アクセス方法	ファイルシステム	パーティション作成、フォーマット、マウントされたblock device（/dev/vdc など）	REST API
アクセス可能な場所	VM内	VM内	どこからでも
管理元	OpenStack Compute (Nova)	OpenStack Block Storage (Cinder)	OpenStack Object Storage (Swift)
データの残存期間	VM終了まで	ユーザーが削除するまで	ユーザーが削除するまで
容量の指定	管理者がサイズ設定（フレーバーとも呼ばれる）を用意する	ユーザが指定する	利用可能な物理ディスクの総量で決まる
典型的な利用例	10GBの1台目ディスク、30GBの2台目ディスク	1TBディスク	数十TBのデータセットストレージ

Novaのみを構成した場合、デフォルトではユーザにはあらゆる種類の永続ストレージへのアクセス方法がありません。この状態でVMに割り当てられるディスクは「エフェメラル」であり、これは仮想マシンが削除された時にディスクが削除される事を意味します。そのためユーザに対してどんなタイプの永続的ストレージがサポートされているか明示しておく必要があります。

現在、OpenStackでは二つの永続的ストレージ（object storage と block storage）がサポートされています。

オブジェクトストレージ

オブジェクトストレージでは、ユーザはREST APIを経由してバイナリオブジェクトへアクセスします。有名なオブジェクトストレージにAmazon S3があります。ユーザがアーカイブ領域や大容量のデータセットを必要としたときにオブジェクトストレージは有効です。またOpenStackはファイルシステムの代わりにオブジェクトストレージに仮想マシンイメージを保存する事が可能です。

ブロックストレージ

ブロックストレージ(ボリュームストレージとも呼ばれる)はユーザにブロックデバイスを提供します。ユーザは実行中のVMにボリュームをアタッチして、ブロックストレージを利用します。

このボリュームは永続的です。データを残したまま仮想マシンからデタッチし、別の仮想マシンへ再アタッチすることができます。OpenStackでは、ブロックストレージは OpenStack Block Storage （Cinder）で実装されており、複数のバックエンドストレージをドライバという形式でサポートします。あなたが選択するストレージバックエンドは、Block Storage のドライバによりサポートされている必要があります。

多くのストレージドライバはインスタンスが直接ストレージハードウェアのブロックデバイスへアクセスできるようにします。これは リード/ライト I/O 性能の向上に役立ちます。

Folsomリリースではファイルをボリュームとして利用するための実験的サポートを開始しました。これは、最初は Cinder で NFS を使用するためのリファレンスドライバとしてスタートしたものです。Grizzly リリースまでに、このドライバはGlusterFS ドライバと同様、完全な NFS ドライバに拡張されました。

これらのドライバーは従来のブロックストレージドライバとは少々異なる動作をします。NFSやGlusterFSでは1つのファイルが作成され、インスタンスに対して「仮想」ボリュームとしてマッピングされます。このマッピング/変換は/var/lib/nova/instances 下に保存される、QEMUのファイルベースの仮想マシンの、OpenStackによる扱い方と同様です。

ファイルレベルストレージ

ファイルレベルストレージでは、ユーザはOSのファイルシステムインターフェースを使ってデータへアクセスします。ほとんどのユーザは（以前ネットワークソリューションを使用した経験があった場合）この

種類のネットワークストレージに遭遇したことがあります。UnixではNFSが一般的で、WindowsではCIFS(旧 SMB)が一般的です。

OpenStackではエンドユーザがファイルレベルストレージを目にすることはありません。しかし、クラウド設計時、/var/lib/nova/instances 下のインスタンス保存用にファイルレベルストレージを検討する事は重要です。なぜなら、ライブマイグレーションをサポートしたい場合、共有ファイルシステムが必須だからです。

ストレージバックエンドの選択

storage back-end 選択における一般的な考慮事項：

- ユーザがブロックストレージを必要とするか？
- ユーザがオブジェクトストレージを必要とするか？
- 管理者がライブマイグレーションを必要とするか？
- 永続的ストレージをコンピュートノード内に持つべきか？それとも外部ストレージに持つべきか？
- 実現可能な容量は？ネットワークアクセスでも、より多くのディスクがより良い I/O 性能に繋がるか？
- どちらが自分の意図した最高のコストパフォーマンスシナリオを実現するか？
- ストレージの運用管理をどうするか？
- ストレージの冗長性と分散をどうするか？ストレージノード障害でどうなるか？災害時、自分のデータ消失をどの程度軽減できるのか？

コモディティハードウェアを利用したストレージ環境の構築に、下記に表に示したいくつかのオープンソースパッケージを利用可能です。

	オブジェクトストレージ	ブロックストレージ	ファイルレベルストレージ*（ライブマイグレーションサポート）
Swift	✓		
LVM		✓	
Ceph	✓	✓	実験的
Gluster	✓		✓
NFS		✓	✓
ZFS		✓	
Sheepdog		実験的	

* このOSS ファイルレベル共有ストレージのリストは完全ではなく、他にもOSSが存在します(MooseFS)。あなたの組織では既に利用可能なファイルレベル共有ストレージがあるかもしれません。

OSSに加えて、OpenStack Block Storageではいくつかのプロプライエタリなストレージを公式にサポートしています。それらは以下のベンダーによって提供されています。

- IBM (Storwize family/SVC, XIV)
- NetApp
- Nexenta
- SolidFire

こちらのリンクからドライバごとにサポートされている機能マトリックスを参照できます。 OpenStack wiki (https://wiki.openstack.org/wiki/CinderSupportMatrix)

クラウド内でオブジェクトストレージの利用を検討する必要があります。コンピュートクラウドで提供されるオブジェクトストレージの一般的な利用方法は以下の二つです。

- ユーザに永続的ストレージの仕組みを提供する
- スケーラブルで信頼性のある仮想マシンイメージデータストアとして利用する

コモディティハードウェア上の ストレージ技術

このセクションでは様々な コモディティハードウェアを利用するストレージ技術の差異について、上位レベルの概要を提供します。

- OpenStack Object Storage (Swift)。OpenStack公式のオブジェクトストア実装です。これはRackspace Cloud Files で採用されており、既に数年間の商用実績を持つ成熟した技術です。高度なスケーラビリティを備え、ペタバイトストレージの管理に適しています。OpenStack Object Storageの利点はOpenStackとの統合(OpenStack Identityとの統合、OpenStack Dashboardインターフェースでの操作)と、非同期の結果整合性レプリケーションによる複数データセンターのサポートです。

  従って、将来的に複数データセンターにまたがった分散ストレージクラスタを計画する場合や、コンピュートとオブジェクトストレージ間で統一されたアカウントを必要とする場合、または OpenStack Dashboard を使ってオブジェクトストレージを操作したい場合などに OpenStack Object Storage を検討します。OpenStack Object Storage のより詳細な情報は以後のセクションで記載します。

- Ceph。コモディティなストレージノード間でデータレプリケーションを行う、スケーラブルなストレージソリューションです。Ceph は元々DreamHost の創設者の一人が開発し、現在はDreamHost の商用サービスで利用されています。

  Ceph はエンドユーザに対して異なるストレージインターフェースが利用できるよう設計されています： オブジェクトストレージ、ブロックストレージ、ファイルシステムをサポートしていますが、ファイルシステムはまだ商用利用可能な状態ではありません。Cephはオブジェクトストレージでswiftと同じAPIをサポートし、Cinder ブロックストレージのバックエンドとしても利用でき、Glance用イメージのバックエンドストレージとしても利用できます。Cephはcopy-on-wirteを使って実装されたシンプロビジョニングをサポートしています。

  ボリューム作成が非常に高速なため、boot-from-volume に有効です。またCephはKeystoneベースの認証(version 0.56等)をサポートするため、デフォルトのOpenStack Swift との置き換えをシームレスに行えます。

  Cephのメリットは、管理者がデータの分散とレプリケーションを細かく計画する事ができること、ブロックストレージとオブジェクトストレージを統合できること、シンプロビジョニングを使ってインスタンスのboot-from-volumeを高速で行えること、 商用利用にはまだ推奨されていませんが（http://ceph.com/docs/master/faq/）分散ファイルシステムのインターフェースを利用できることです。

  単一システムでブロックストレージとオブジェクトストレージを管理したい場合や、高速なboot-from-volumeをサポートしたい場合はCephの利用を検討してください。

- Gluster 分散共有ファイルシステムです。Gluster 3.3の時点で、オブジェクトストレージとファイルストレージを統合して利用でき、これはGluster UFOと呼ばれています。Gluster UFOは、Glusterをバックエンドとして使うようにカスタマイズされたSwiftを利用しています。

  正規のSwift経由でGluster UFOを使う利点は、分散ファイルシステムをサポートしたい時や、共有ストレージによるライブマイグレーションのサポートや、個別サービスとしてエンドユーザにGluster UFOを提供できる事です。単一ストレージシステムでオブジェクトストレージとファイルストレージを管理したい場合はGluster UFOを検討します。

- LVM 論理ボリュームマネージャ。オペレーティングシステムへ論理ディスクを公開するために、物理ディスク上の抽象化レイヤーを提供するLinuxベースの仕組みです。LVM(論理ボリュームマネージャ)の

バックエンドは、LVM論理パーティションとしてブロックストレージを提供します。

ブロックストレージを収容する各ホストでは、管理者は事前にブロックストレージ専用のボリュームグループを作成しておく必要があります。ブロックストレージはLVM論理ボリュームから作られます。

注記

LVMはレプリケーションを提供しません。通常、管理者はブロックストレージとしてLVMを利用するホスト上にRAIDを構成し、ここのハードディスク障害からブロックストレージを保護します。しかしRAIDではホストそのものの障害には対応できません。

- ZFS. Solarisの OpenStack Block Storage用のiSCSIドライバーはZFSを実態としたブロックストレージを実装しています。ZFSはボリュームマネージャ機能を持ったファイルシステムです。これはボリュームマネージャ(LVM)とファイルシステム(ext3, ext4, xfs, btrfsのような)が分離しているLinuxとは異なっています。ZFSはデータ整合性チェックを含み、ext4より多くの利点を持っています。

  OpenStack Block Storage用のZFSバックエンドは Illumos 等の Solaris ベースのシステムのみをサポートします。Linuxにポーティングされ ZFSもありますが、標準的なLinuxディストリビューションには含まれておらず、OpenStack Block Storage でもテストされていません。LVMと同様にZFSはホスト間のレプリケーション機能を提供していませんので、ストレージノードの障害に対応するためには、ZFS上にレプリケーション機能を追加する必要があります。

  ここではLinuxベースのシステムを前提としているので、これまでにZFSの構築実績が無ければ、Solarisの知識を前提とするZFSはあえてお薦めしません。

- Sheepdog KVMのインスタンスにブロックストレージを提供する事に特化した新しいプロジェクトで、ホスト間のレプリケーションもサポートします。ただし、作者であるNTT研究所では、Sheepdogは実験的な技術と考えており、商用クラウドサービスでの利用はお薦めしません。

OpenStack Object Storage の注意事項

OpenStack Object Storage は従来のファイルシステムの制約を一部緩和することで、高いスケーラビリティと可用性を実現しています。その設計のためには、動作のキーコンセプトを理解することが重要です。このタイプのストレージはあらゆる場所・レベルにおいてハードウェア障害が発生する、という考えに基づいて構築されています。他のストレージシステムでは動作不能になってしまうような、まれに発生するRAIDカードやホスト全体の障害に対しても OpenStack Object Storage は正常に動作します。

オブジェクトストレージのアーキテクチャについて the developer documentation (http://docs.openstack.org/developer/swift/overview_architecture.html) に記述されています。まずアーキテクチャを理解し、プロキシサーバーとZoneがどのように働くか知る必要があります。重要なポイント見逃さないように注意してください。

クラスタの設計には耐久性と可用性を検討する必要があります。耐久性と可用性はハードウェアの信頼性ではなく、データの分布と配置が重要です。デフォルトのレプリカ数3について考えます。これはオブジェクトが書き込まれた時に少なくとも2つのコピーが存在する事を意味します。1台のサーバーへの書き込みが失敗した場合、3つ目のコピーは書き込み操作が返った直後には存在するかもしれないし、存在しないかもしれません。レプリカ数を増やすとデータの堅牢性は増しますが、利用できるストレージの総量は減ってしまいます。次にサーバーの配置を見てみます。データセンター全体でネットワークや電源の障害箇所とサーバーの分布を考えてください。その障害ではラック、サーバー、ディスクのどこが影響を受けますか？

オブジェクトストレージのネットワークのトラフィックは通常とは異なっているかもしれません。以下のトラフィックを考慮してください。

- object server 、 container server 、 account server 間
- object/container/account server と proxy server の間
- proxy server と 利用者の間

オブジェクトストレージはデータを保持するノード間で頻繁に通信を行います。 小さなクラスタでさえ、これは数MB/sのトラフィックで主に他ホストにオブジェクトの存在確認を行いっています。相手ノードにオブジェクトが無い場合はレプリケーションが開始されます。

サーバー障害で3つのコピーを保つために、24TBのデータ転送が必要になる場合を考えてみてください。これはネットワークに大きな負荷を発生させます。

忘れてはいけない事として、レプリカがあるためファイルがアップロードされたときに、proxy server は多くのストリームを書き出す必要があることです。これは3レプリカの場合、10Gbpsのアップロードに対して、30Gbpsのダウンロードになります。これはパブリック側のネットワークよりも、プライベート側のネットワークがより多くの帯域を必要とすることを意味しています。OpenStack Object Storage はパフォーマンスのために非暗号化、未認証のrsync通信を行います。そのためプライベートネットワークは非公開である必要があります。

残りのポイントはパブリック側のネットワーク帯域になります。swift-proxyはステートレスなため、ノードを追加し、HTTPロードバランスを使うことで帯域の増加と可用性の向上を容易に行うことができます。

ストレージ側の性能が十分であれば、proxy server の増加は帯域の増加になります。

第6章 ネットワーク設計

管理ネットワーク 55
パブリックアドレスの選択肢 55
IP アドレス計画 56
ネットワークトポロジー 58
ネットワーク関係のサービス 60

OpenStack は高度なネットワーク環境を提供します。本章では、クラウドを設計するときに考慮すべき要件と選択肢について詳細に説明します。

これがあなたの組織で初めてのクラウド基盤構築であれば、この章を読んだ後、最初にあなたの（組織の）ネットワーク管理チームと相談すべきです。クラウド運用におけるネットワークの使用は伝統的なネットワーク構築とはかなり異なり、接続性とポリシーレベルの両面で破壊的な結果をもたらす可能性があるからです。

例えば、管理インフラだけでなくゲストインスタンス用のIPアドレスの数も計画しなければなりません。加えて、プロキシサーバーやファイアウォールを経由してのクラウドネットワークの接続性を調査・議論する必要があります。

管理ネットワーク

管理ネットワークを用意するのはお薦めの選択肢です。通常、管理ネットワークは専用のスイッチと NIC で構成します。ネットワークを分離することで、システム管理と監視システムアクセスが、ゲスト自身が生成するトラフィックによって邪魔されることがなくなります。

メッセージキューや OpenStack Compute といった OpenStack 内部のコンポーネント間の通信用に別のプライベートネットワークを作成することを検討して下さい。 VLAN はこれらのシナリオに非常に適しています。

パブリックアドレスの選択肢

ゲストの仮想マシン用の IP アドレスは、固定 IP とフローティング IP の2種類に大別できます。固定 IP はインスタンス起動時にインスタンスに割り当てられ、フローティング IP はユーザ操作によりインスタンスへの割り当てを変更できます。どちらのタイプの IP アドレスについて

も、パブリックアドレス、プライベートアドレスのいずれかをあなたの用途に合わせて選択することができます。

固定 IP アドレスは必須ですが、フローティング IP はなくても OpenStack を実行することができます。フローティング IP の最も一般的な用途の 1 つは、利用可能なIPアドレス数が限られているプライベートクラウドでパブリックIPアドレスを利用できるようにすることです。他の用途としては、パブリッククラウドのユーザが、インスタンスがアップグレードや移動した際でも割り当て直すことができる「静的」 IP アドレスを利用できるようにすることです。

固定IPアドレスは、プライベートクラウドではプライベートアドレスに、パブリッククラウドではパブリックアドレスにすることが出来ます。インスタンスが削除される際、そのインスタンスの固定IPは割当を解除されます。IPアドレスが使い終わったらすぐに解放されてしまうという動作に、クラウドコンピューティングの初心者がストレスを感じる可能性があることに注意しましょう。

IP アドレス計画

OpenStack のインストールでは、潜在的に、多数のサブネットと、サブネット毎に異なる種類のサービスが存在する可能性があります。IPアドレス計画は、ネットワーク分割の目的とスケーラビリティに関する理解を共有するのに役立ちます。コントロールサービスはパブリックIPアドレスとプライベートIPアドレスを持つ場合があり、上記の通り、インスタンスのパブリックIPアドレスには2種類のオプションが存在します。

IPアドレス計画は以下のセクションに分類できるでしょう。

サブネットルータ	このサブネットから出て行くパケットはこのアドレスを経由して出て行きます。このアドレスは、専用ルータにすることも、nova-network サービスにすることもできます。
コントロールサービスのパブリックインターフェース	swift-proxy, nova-api, glance-api, horizon へのパブリックアクセスはこれらのアドレス宛に行われます。これらのアドレスはロードバランサの一方か、個々のマシンに割り当てられます。
Object Storage クラスタ内の通信	object/account/container サーバー間、またはこれらのサーバーとプロキシサーバーの内側のインターフェースとの間の通信は、このプライベートネットワークを使用します。
コンピュートとストレージ間の通信	一時ディスクまたはブロックストレージがコンピュートノード以外にある場合、このネットワークが使用されます。
アウトバンドのリモート管理	専用のリモートアクセスコントローラーチップがサーバーに搭載されている場合、多くの

	場合、これらは独立したネットワーク上に置かれます。
インバンドのリモート管理	多くの場合、システム管理者や監視ツールからホストへのアクセスは、パブリックインタフェース経由ではなく、コンピュートノード、ストレージノードの（1Gbps などの）追加のインタフェース経由で行われます。
将来の拡張用の予備のアドレス空間	パブリック側に置かれる制御サービスやゲストインスタンスのIPの追加は、必ずアドレス計画の一部として入れておくべきです。

例えば、OpenStack Compute と Object Storage の両方を使用し、プライベートアドレス範囲として 172.22.42.0/24 と 172.22.87.0/26 が利用できる場面を考えます。一例として、アドレス空間を以下のように分割することができます。

```
172.22.42.0/24
172.22.42.1   - 172.22.42.3   - subnet routers
172.22.42.4   - 172.22.42.20  - spare for networks
172.22.42.21  - 172.22.42.104 - Compute node remote access controllers (inc spare)
172.22.42.105 - 172.22.42.188 - Compute node management interfaces (inc spare)
172.22.42.189 - 172.22.42.208 - Swift proxy remote access controllers (inc spare)
172.22.42.209 - 172.22.42.228 - Swift proxy management interfaces (inc spare)
172.22.42.229 - 172.22.42.252 - Swift storage servers remote access controllers (inc spare)
172.22.42.253 - 172.22.42.254 - spare
172.22.87.0/26:
172.22.87.1  - 172.22.87.3    - subnet routers
172.22.87.4  - 172.22.87.24   - Swift proxy server internal interfaces (inc spare)
172.22.87.25 - 172.22.87.63   - Swift object server internal interfaces (inc spare)
```

パブリックIPアドレスの場合でも同様のアプローチが取れます。但し、ゲストインスタンス用のIPとして使用する場合には、大きなフラットなアドレスレンジの方が好まれることに注意した方がよいでしょう。また、OpenStack のネットワーク方式によっては、ゲストインスタンス用のパブリックIPアドレスレンジのうち一つが nova-compute ホストに割り当てられることも考慮する必要があります。

ネットワークトポロジー

OpenStack Compute では、数種類のネットワークマネージャーが用意されており、それぞれ長所と短所があります。どのネットワークマネージャーを選択するかは利用するネットワークトポロジーにより変わります。そのため、慎重に選択する必要があります。

種別	長所	短所
Flat	極めてシンプル。 DHCP ブロードキャストなし。	インスタンスに対するファイルインジェクションが必須。 特定の Linux ディストリビューションしか利用できない。 設定の難易度は高く、非推奨。
FlatDHCP	比較的シンプルな構成。 標準的なネットワーク。 すべてのオペレーティングシステムが利用できる。	専用の DHCP ブロードキャストドメインが必要。
VlanManager	各テナントが専用の VLAN で分離される。	少し複雑な構成。 専用の DHCP ブロードキャストドメインが必要。 一つのポートに多数の VLAN をトランクが必要。 標準的な VLAN 数の上限。 802.1q VLAN タギングに対応したスイッチが必要。
FlatDHCP Multi-host HA	ネットワーク障害が影響を受けるハイパーバイザー上の VM に限定される。 DHCP トラフィックは個々のホスト内に閉じ込めることができる。 ネットワークトラフィックをコンピュートノード全体に分散できる。	少し複雑な構成。 デフォルトでは、各コンピュートノードにパブリック IP アドレスが必要となる。 ライブマイグレーションでネットワークが動作するようにするためは、オプションを慎重に設定する必要がある。

VLAN

VLAN 設定は要求に応じて単純にも複雑にもなり得ます。 VLAN を使用すると、各プロジェクトのサブネットとブロードキャストを他のプロジェクトから分離できるというメリットがあります。 OpenStack が VLAN を

効率的に利用できるようにするには、ある範囲の VLAN（1プロジェクト1VLAN）を割り当て、各コンピュートノードのスイッチポートをトランクポートに設定する必要があります。

例えば、あなたのクラウドでサポートする必要があるプロジェクト数が最大で100と見込まれる場合、ネットワークインフラで現在使用されていない VLAN の範囲を選んで下さい（例えば VLAN 200 - 299)。この VLAN の範囲を OpenStack に設定するとともに、この範囲の VLAN トラフィックを許可するようにスイッチポートを設定しなければいけません。

マルチNIC

OpenStack Compute には、一つのインスタンス複数の NIC を割り当てる機能があり、プロジェクト単位に制御できます。一般的には、これは高度な機能で、普段から必要になるものではありません。また、この機能をリクエスト単位で使うことも簡単にできます。しかしながら、2つ目のNICを使うと、サブネット、つまり VLAN が一つまるごと必要になる点に注意して下さい。これにより、全体で収容できるプロジェクト数が一つ減ることになるからです。

マルチホストネットワークとシングルホストネットワーク

nova-network は、マルチホストモードでもシングルホストモードでも動作させることができます。マルチホストモードでは、各コンピュートノードで nova-network サービスを動作させ、コンピュートノード上のインスタンスはそのコンピュートノードをインターネットへのゲートウェイとして使用します。コンピュートノードはそのノード上のインスタンスに対してフローティングIPとセキュリティグループ機能も提供します。シングルホストモードでは、1台の中央のサーバー（例えば、クラウドコントローラー）で nova-network を動かします。全コンピュートノードがインスタンスからのトラフィックをクラウドコントローラーに転送し、クラウドコントローラーはトラフィックをインターネットに転送します。クラウドコントローラーは、クラウド上の全インスタンスに対してフローティングIPとセキュリティグループ機能を提供します。

どちらのモードにもメリットがあります。シングルホストモードには、単一障害点というマイナス面があります。クラウドコントローラーが利用できなくなると、インスタンスはネットワークと通信できなくなります。マルチホストモードでは、この状況にはなりませんが、各コンピュートノードはインターネットと通信するためのパブリックIPアドレスが必要となります。十分な大きさのパブリックIPアドレスブロックを取得できない場合には、マルチホストモードは選択肢にならないかもしれません。

ネットワーク関係のサービス

OpenStack も、他のネットワークアプリケーション同様、 DNS や NTP など標準的に考慮すべき点が多くあります。

NTP

時刻同期は OpenStack のコンポーネントの継続的な動作を保証するためには不可欠な項目です。正しい時刻は、インスタンスのスケジューリング、オブジェクトストアでのオブジェクト複製や、デバッグ時のログのタイムスタンプの突き合わせなどでのエラーを避けるために必要です。

OpenStack のコンポーネントが動作している全てのサーバーから適切な NTP サーバにアクセスできるようにすべきです。 NTP サーバーは自分で用意するか、 http://www.pool.ntp.org/ に載っている公開 NTPサーバーを使うこともできます。

DNS

nova-network ホスト上で動作する dnsmasq デーモンを除くと、現時点では、OpenStack は DNS サービスを提供していません。ダイナミック DNS サービスを提供して、インスタンスが新しいIPアドレスで DNS エントリを更新できるようにすることを検討する価値があります。また、インスタンスの IP アドレスに対して、vm-203-0-113-123.example.com のような、汎用的な順引き、逆引き DNS マッピング、を行うことを検討してもよいでしょうか。

第7章 参考アーキテクチャ

概要 63
設定指針 64
詳細な説明 66
さらなる拡張 69

OpenStack は設定の自由度が高く、多くの異なるバックエンドとネットワーク設定オプションがあり、考えられる OpenStack の構成をすべて網羅するドキュメントを作成するのは困難です。そのため、このガイドでは参考アーキテクチャを定義することで、このガイドの説明範囲を明確にするとともに、ドキュメント作成を単純化しています。こうすることで、、著者らが実際に経験したことのある構成での設定に焦点を当てることができます。

概要

OpenStack リリース	Folsom
ホストのオペレーティングシステム	Ubuntu 12.04 LTS
OpenStack パッケージリポジトリ	Ubuntu Cloud Archive (https://wiki.ubuntu.com/ServerTeam/CloudArchive) *
ハイパーバイザー	KVM
データベース	MySQL*
メッセージキュー	RabbitMQ
ネットワークサービス	nova-network
ネットワークマネージャー	FlatDHCP
nova-network がシングルホストかマルチホストか？	マルチホスト*
Image Service (glance) のバックエンド	file
Identity Service (keystone) のドライバー	SQL
Block Storage Service (cinder) のバックエンド	LVM/iSCSI
ライブマイグレーションのバックエンド	NFS を使った共有ストレージ *
オブジェクトストレージ	OpenStack Object Storage (swift)

アスタリスク（*）は、参考アーキテクチャでの設定がデフォルトのインストールの設定とは違うことを示します。

注記

以下の OpenStack の機能がこのガイドの参考アーキテクチャではサポートされていますが、これらは必須項目ではありません。

- ダッシュボード
- ブロックストレージ
- フローティング IP アドレス
- ライブマイグレーション
- オブジェクトストレージ

設定指針

この参考アーキテクチャは、OpenStack Folsom の現在のデフォルト機能をベースとして、安定性を重視して選択されました。特に、 OpenStack Folsom リリースの構成で、この文書の著者の誰も使ったことがないバックエンドや設定については、この参考アーキテクチャでは取り上げないことにしました。きっと、現在本番環境で OpenStack を使っている多くのクラウドでは同様の選択がなされていることでしょう。

まず最初に、すべての物理ノードで動作させるオペレーティングシステムを選ばなければなりません。いくつかの Linux ディストリビューションが OpenStack をサポートしていますが、我々は Ubuntu 12.04 LTS (Long Term Support) を使うことにしました。 Ubuntu 12.04 LTS は開発コミュニティの大半の人が使っており、他のディストリビューションと比較して機能の完成度が高く、今後のサポート計画もはっきりしています。

Ubuntu のデフォルトの OpenStack インストールパッケージではなく、Ubuntu Cloud Archive (https://wiki.ubuntu.com/ServerTeam/CloudArchive) を使うことをお薦めします。Cloud Archive は Canonical がサポートしているパッケージレポジトリで、このレポジトリを使うことで Ubuntu 12.04 を使い続けながら新しい OpenStack リリースにアップグレードすることができます。

Ubuntu を選択した場合、 ハイパーバイザー として KVM が最も適切です。サポートの観点ではぴったりの組み合わせであり、また (著者も含め) OpenStack の開発コミュニティからの関心も非常に高いからです。著者のほとんどが KVM を使っています。また、 KVM は機能が揃っていて、ライセンスの課金も制限もありません。

MySQL も同様の理由から選ばれました。最近開発元が変わりましたが、このデータベースは OpenStack との組み合わせで最もテストされており、 Ubuntu での動かし方についても非常によくドキュメントにまとまっています。デフォルトのデータベースである SQLite を使っていませんが、それは SQLite が本番環境での利用には適したデータベースではないからです。

OpenStack では AMQP 互換の選択肢としては ZeroMQ や Qpid などのサポートが進んでいますが、 RabbitMQ を選んだのは、Ubuntu での使いやすさと、本番環境で十分にテストされているのが理由です。また、RabbitMQ は Compute Cell といった機能でサポートされている唯一の選択肢です。 RabbitMQ はクラスタ構成にすることを推奨します。それは、メッセージキューは OpenStack システムで不可欠のコンポーネントで、RabbitMQ 自体で元々サポートされているためかなり簡単に実現することができるからです。

前の章に議論したように、OpenStack Compute のネットワークにはいくつかの選択肢があります。我々は FlatDHCP を選択し、高可用性のために マルチホスト モードを使うことをお薦めします。マルチホストモードでは、nova-network デーモンを OpenStack Compute ホスト毎に一つ動作させます。この堅牢性のメカニズムでは、ネットワーク障害が各コンピュートホスト内に閉じることが保証され、各ホストがハードウェアのネットワークゲートウェイと直接通信できます。

ライブマイグレーション は共有ストレージを使うことでサポートされます。分散ファイルシステムとして NFS を使います。

多くの小規模の構成では Object Storage サービスを仮想マシンイメージのストレージのためだけに使うのはコストがかかることなので、OpenStack Image Service (Glance) のバックエンドとしてファイルバックエンドを選択しました。あなたが設計しているクラウドで Object Storage も動かすつもりであれば、代わりに Object Storage をバックエンドとして使うのは簡単なので、使用することを是非お薦めします。

Identity サービス (keystone) のバックエンドとして、LDAP などの他の選択肢ではなく SQL バックエンドを選択しました。 SQL バックエンドはインストールが簡単で、堅牢性があります。多くのシステムで既存のディレクトリサービスと接続したいという要望があることは理解しています。その場合は、設定オプションリスト (http://docs.openstack.org/trunk/config-reference/content/ch_configuring-openstack-identity.html#configuring-keystone-for-ldap-backend) を注意深く理解してから使って下さい。

Block Storage サービス (cinder) は外部のストレージノードに直接インストールされ、LVM/iSCSI プラグイン を使用します。ほとんどの

Block Storage サービスのプラグインは個々のベンダー製品や実装に依存しており、そのハードウェアプラットフォームを持っているユーザしか利用できませんが、 LVM/iSCSI は堅牢性と安定性があり、一般的なハードウェアで利用できます。

このクラウドは OpenStack ダッシュボード なしで動かすことはできますが、我々はダッシュボードを、クラウドのユーザ操作のためだけでなく、オペレータ向けのツールとしてもなくてはならないものと考えています。まだ、ダッシュボードは Django を使っているため拡張しやすい柔軟なフレームワークとなっています。

なぜ OpenStack Networking Service (neutron) を使わないか？

我々はこのガイドで OpenStack Networking Service (neutron) について触れていません。それは、このガイドの著者達が nova-network を使った本番環境の経験しかないからです。それに加えて、neutron はまだマルチホストネットワークをサポートしていないことも理由の一つです。

なぜマルチホストネットワークを使うか？

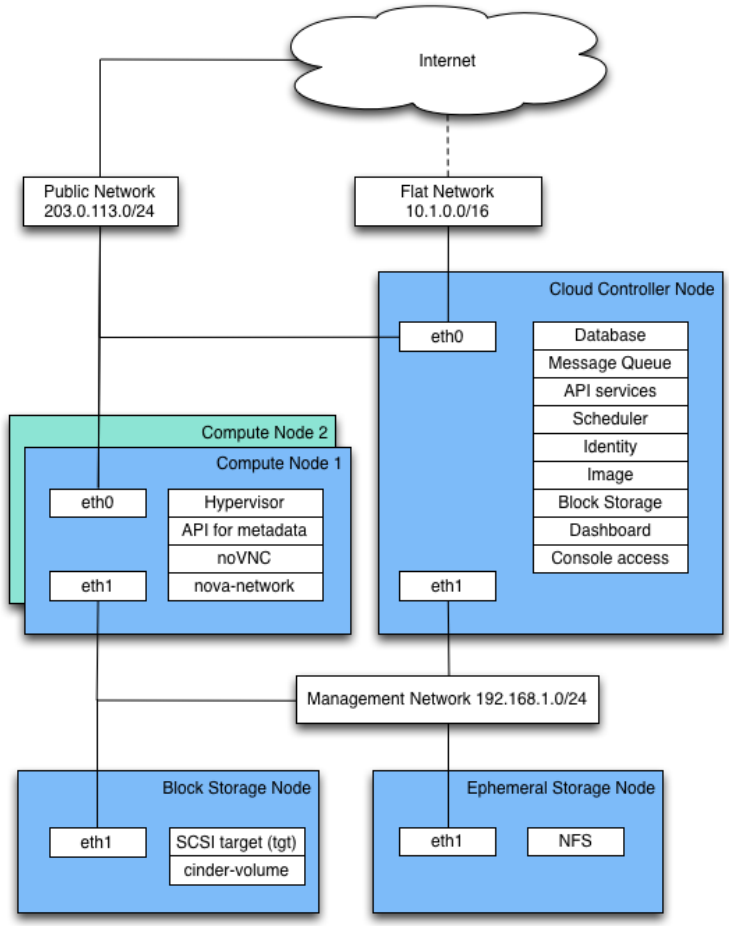
デフォルトの OpenStack の構成では、nova-network サービスはクラウド内（通常はクラウドコントローラー）で一つだけ動作し、ゲストインスタンスにネットワークアドレス変換 (NAT)、DHCP、DNS などのサービスを提供します。 nova-network サービスが動作している1台のノードがダウンすると、ユーザーはインスタンスにアクセスできなくなり、インスタンスはインターネットにアクセスできなくなります。クラウドで送受信されるネットワークトラフィックが多くなり過ぎると、nova-network サービスが動作する1台のノードがボトルネックになる可能性があります。

マルチホスト (http://docs.openstack.org/folsom/openstack-compute/admin/content/existing-ha-networking-options.html#d6e8906) は、ネットワーク設定の高可用性オプションで、 nova-network サービスを1台のノードだけで動かすのではなく、各コンピュートノードで動作させるものです。

詳細な説明

参考アーキテクチャは複数のコンピュートノード、クラウドコントローラー、インスタンスストレージ用の外部の NFS ストレージサーバー、ボ

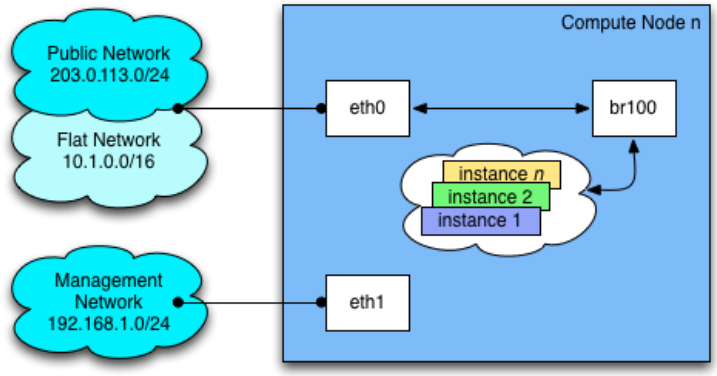
リューム ストレージ用の OpenStack Block Storage サーバーで構成されています。ネットワーク時刻サービス (Network Time Protocol, NTP) により全ノードの時刻が同期されます。ネットワークでは、マルチホストモードの FlatDHCPManager を使用しています。



クラウドコントローラーでは、ダッシュボード、API サービス、データベース (MySQL)、メッセージキューサーバー (RabbitMQ)、コンピューティングリソースの選択を行うスケジューラー (nova-scheduler)、Identity サービス (keystone, nova-consoleauth), Image Service (glance-api, glance-registry), ゲストへのコンソールアクセス用サービス、ストレージリソースのスケジューラーを含む Block Storage サービス(cinder-api と cinder-scheduler) が動作します。

コンピュートノードはコンピューティングリソースを提供する場所です。この参考アーキテクチャでは、コンピュートノードで、ハイパーバイザー（KVM)、 libvirt（ハイパーバイザー用のドライバーで、ノード間でのライブマイグレーションを可能にします)、 nova-compute、nova-api-metadata（通常はマルチホストモードの場合のみ使用されます。インスタンス固有のメタデータの取得に使われます)、 nova-vncproxy、 nova-network が動作します。

ネットワークは2つのスイッチで構成され、一つは管理やプライベートトラフィック用で、もう一つはフローティングIPなどのパブリックアクセスに使用されます。この構成をとるため、クラウドコントローラーと各コンピュートノードにはNICを2つ用意しています。OpenStack Block Storage と NFS ストレージサーバーはプライベートネットワークだけにアクセスできればよく、そのため必要なNICは1つです。ただし、可能であれば複数のNICを bonding 設定で動作させることを推奨します。フローティング IP アクセスはインターネットと直結になりますが、フラット IP アクセスは NAT 経由となります。



さらなる拡張

この参考アーキテクチャを以下のように拡張することができます。

- 追加でクラウドコントローラーを増やす（11章メンテナンス、故障およびデバッグ [119] を参照）。
- OpenStack Storage サービスを追加する（http://docs.openstack.org/havana/install-guide/install/apt/content/ch_swift.html）
- 追加で OpenStack Block Storage ホストを増やす（see 11章メンテナンス、故障およびデバッグ [119] 参照）。

次はどうする？

おめでとうございます！ ここまでで、あなたのクラウドのしっかりとした基本設計ができたことでしょう。ここまで来たら、OpenStack Install and Deploy Manual - Ubuntu（http://docs.openstack.org/havana/install-guide/install/apt/）を見ることをお薦めします。このマニュアルには、OpenStack パッケージと依存モジュールをあなたのクラウドに手動でインストールする方法がステップバイステップで説明されています。

運用者が OpenStack を構築する一つ一つの手順を詳しく知ることも大事ですが、 Puppet や Chef といった設定管理ツールを評価することを強くお薦めします。これらのツールは構築手順を自動化する上で助けとなることでしょう。

このガイドの残りでは、OpenStack クラウドはうまく構築でき、イメージの追加、インスタンスの起動、ボリュームの接続といった基本的な操作は実行できるものとして話を進めます。

あなたの興味は安定した運用をどのように行うかに移っていると思いますので、この本の残りの部分にどんなことが書かれているかざっと眺めることをお薦めします。いくつかの内容は前もって読んでおくと役に立ちます。ベストプラクティスを実行することで、長い目で見ると運用が楽になることでしょう。他の内容は、電源故障や特定の問題のトラブルシューティングといった予期しない現象が起こったときに参考になります。

第8章 環境の把握

クライアントコマンドラインツール 71
ネットワーク 79
ユーザーとプロジェクト 79
稼働中のインスタンス 81

ここからは、あなたがOpenStackが稼働している環境を持っていると想定して進めます。この節は環境の構築と情報収集に役立つでしょう。

クライアントコマンドラインツール

OpenStack command line interface (CLI) クライアントツールとOpenStackダッシュボードを組み合わせて使うことをおすすめします。他のクラウド技術の経験を持つユーザーは、EC2互換APIを使っているかも知れませんが、それとは命名規則が少々異なります。それらの違いは必要に応じて補足します。

コマンドラインツールは、UbuntuやFedoraのパッケージからではなく、Python Package Index(PyPI) (https://pypi.python.org/) から導入することを強くおすすめします。クライアントツールは開発が活発であり、OSベンダーが配布しているパッケージのバージョンが古くなりがちなためです。

「pip」ユーティリティはPyPIアーカイブからパッケージを導入するために使われます。そして、多くのLinuxディストリビューションでは「pip」ユーティリティは「python-pip」パッケージに含まれています。各OpenStackプロジェクトはそれぞれクライアントを持ちますので、あなたの環境で動かすサービスに合わせて、以下のパッケージを導入してください。

- python-novaclient (nova CLI)
- python-glanceclient (glance CLI)
- python-keystoneclient (keystone CLI)
- python-cinderclient (cinder CLI)
- python-swiftclient (swift CLI)
- python-neutronclient (neutron CLI)

ツールの導入

PyPIアーカイブからpipを使ってパッケージをインストール（もしくはアップグレード)するには、rootとして、

```
# pip install [--upgrade] <package-name>
```

パッケージを削除するには、

```
# pip uninstall <package-name>
```

もし新しいバージョンのクライアントが必要な場合、-eフラグを指定することで、アップストリームのgitリポジトリから直接導入できます。その際は、Python egg名を指定しなければいけません。例えば、

```
# pip install -e git+https://github.com/openstack/python-novaclient.git#egg=python-novaclient
```

もしEC2 APIを使いたい場合、「euca2ools」パッケージなどのEC2 API対応ツールで実現できます。EC2 APIベースのツールはこのガイドの範囲外ですが、それを使うための認証情報の取得手段については触れることにします。

管理系コマンドラインツール

*-manageという名のコマンドラインツールがいくつかあります。

- nova-manage
- glance-manage
- keystone-manage
- cinder-manage

前述のツールと違って、*-manageツールはクラウドコントローラーからroot権限で実行されなければいけません。なぜなら、それらのツールは/etc/nova/nova.confなどの設定ファイルを読み取り、また、OpenStack API エンドポイントではなく、データベースに対し直接クエリーを発行するからです。

-manageツールが存在しているのは、歴史的な経緯からです。OpenStackプロジェクトは最終的に、-manageの全機能を通常のクライアントツールへ移行する予定です。それまでは、cloud controller nodeへSSHし、必要な*-manageを使ってメンテナンス作業を行う必要があります。

認証情報の取得方法

OpenStackクラウドに対してクエリを発行するためにコマンドラインツールを使いたいのであれば、適切な認証情報を持っている必要があります。コマンドラインツールで使う認証情報を得るための最も簡単な方法は、Horizonダッシュボードです。ユーザー設定ページを表示するには、ページ右上のナビゲーションバーから、設定のリンクを選択し、ユーザー設定を開きます。ユーザー設定では、言語やタイムゾーンも設定できます。ここで重要なのは、このページ左側のナビゲーションにあるOpenStack API と EC2 認証情報 リンクです。このリンクから、あなたのサービスエンドポイントや認証情報といった環境変数を設定するファイルを生成できます。

OpenStackネイティブのツールを使うには、 OpenStack API リンクを選択します。ページ上部にはサービスエンドポイントのURLリストが、その下にタイトルOpenStack RCファイルのダウンロードがあります。管理者としてクラウドにアクセスする場合、ドロップダウンメニューから選択できます。ここから認証情報を取得したいプロジェクトを選択し、RCファイルのダウンロードをクリックします。以下のような、openrc.shファイルが生成されます。

```
#!/bin/bash

# With the addition of Keystone, to use an openstack cloud you should
# authenticate against keystone, which returns a **Token** and **Service
# Catalog**. The catalog contains the endpoint for all services the
# user/tenant has access to - including nova, glance, keystone, swift.
#
# *NOTE*: Using the 2.0 *auth api* does not mean that compute api is 2.0.
# We use the 1.1 *compute api*
export OS_AUTH_URL=http://203.0.113.10:5000/v2.0

# With the addition of Keystone we have standardized on the term **tenant**
# as the entity that owns the resources.
export OS_TENANT_ID=98333aba48e756fa8f629c83a818ad57
export OS_TENANT_NAME="test-project"

# In addition to the owning entity (tenant), openstack stores the entity
# performing the action as the **user**.
export OS_USERNAME=test-user

# With Keystone you pass the keystone password.
echo "Please enter your OpenStack Password: "
read -s OS_PASSWORD_INPUT
export OS_PASSWORD=$OS_PASSWORD_INPUT
```

注記

あなたのパスワードはプレーンテキストに保存されません。これはいいことです。このスクリプトを読み込み、実行する際、パスワードの入力が要求され、環境変数OS_PASSWORDに設定されます。対話式に要求することが重要です。もしあなたが非対話式に操作したいのであれば、値を直接スクリプトに

書くことは可能です。しかし、このファイルのセキュリティとパーミッションに細心の注意を払う必要があります。

EC2互換クレデンシャルは左側ナビゲーションバー内の「EC2 認証情報」リンクからダウンロードできます。対象認証情報のプロジェクトを選択し、「EC2 認証情報のダウンロード」をクリックすると、x509証明書とシェルスクリプトを含むzipファイルが生成されます。クラウドにアクセスするために必要な認証情報を含んでいるため、openrcとは異なり、安全な場所へ新しいディレクトリを作り、そこへzipファイルを展開してください。cacert.pem、 cert.pem、 ec2rc.sh そして pk.pemがあるはずです。 ec2rc.shは以下のようになるでしょう。

```
#!/bin/bash

NOVARC=$(readlink -f "${BASH_SOURCE:-${0}}" 2>/dev/null) ||¥
NOVARC=$(python -c 'import os,sys; ¥
print os.path.abspath(os.path.realpath(sys.argv[1]))' "${BASH_SOURCE:-${0}}")
NOVA_KEY_DIR=${NOVARC%/*}
export EC2_ACCESS_KEY=df7f93ec47e84ef8a347bbb3d598449a
export EC2_SECRET_KEY=ead2fff9f8a344e489956deacd47e818
export EC2_URL=http://203.0.113.10:8773/services/Cloud
export EC2_USER_ID=42 # nova does not use user id, but bundling requires it
export EC2_PRIVATE_KEY=${NOVA_KEY_DIR}/pk.pem
export EC2_CERT=${NOVA_KEY_DIR}/cert.pem
export NOVA_CERT=${NOVA_KEY_DIR}/cacert.pem
export EUCALYPTUS_CERT=${NOVA_CERT} # euca-bundle-image seems to require this

alias ec2-bundle-image="ec2-bundle-image --cert $EC2_CERT --privatekey ¥
$EC2_PRIVATE_KEY --user 42 --ec2cert $NOVA_CERT"
alias ec2-upload-bundle="ec2-upload-bundle -a $EC2_ACCESS_KEY -s ¥
$EC2_SECRET_KEY --url $S3_URL --ec2cert $NOVA_CERT"
```

EC2認証情報を環境変数として有効にするには、ec2rc.shファイルを(source コマンドで）読み込みます。

コマンドラインのこつとはまりどころ

コマンドラインツールは、--debugフラグを渡すことでOpenStack API コールを表示することができます。例えば、

```
# nova --debug list
```

この例は、クライアントからのHTTPリクエストとエンドポイントからのレスポンスを表示しています。これはOpenStack APIを使ったカスタムツールを作る際に役立ちます。

Keyring Support (https://wiki.openstack.org/wiki/KeyringSupport)は、このガイドの執筆時点では混乱の元になるかもしれません。このバグレポート (https://bugs.launchpad.net/python-novaclient/+bug/1020238)はオープンされた後、無効(invalid)としてクローズされ、何度かやり取りがあって再度オープンされています。

その問題とは、ある条件下でコマンドラインツールがPythonのkeyringを認証情報キャッシュとして使おうとし、さらにある条件が重なった時、使用するたびにツールがkeyringパスワードを要求することです。もしあ

なたが運悪くその現象に遭遇した場合、認証情報キャッシュを回避するため、--no-cacheフラグを追加するか、OS_NO_CACHE=1を環境変数に設定してください。

注記

no-cache を設定すると、コマンドラインツールがクラウドとやり取りを行う度に認証が行われます。

cURL

コマンドラインツールの内部ではOpenStack APIが使用されます。OpenStack APIはHTTP上で動作するRESTful APIです。時に、APIと直接やりとりしたい、CLIツールのバグを疑っている、ということがあるでしょう。その際に最適な方法は、cURL (http://curl.haxx.se/)と、JSONレスポンスを解析するjq (http://stedolan.github.com/jq/)などのツールを組み合わせることです。

まずはじめに、クラウドの認証が必要です。あなたの認証情報を用いて認証トークンを入手してください。

あなたの認証情報はユーザー名、パスワード、テナント(プロジェクト)の組み合わせです。前述したopenrc.shで、それらの値を得ることができます。そのトークンのおかげで、リクエストのたびに再認証することなく、サービスエンドポイントとやりとりすることができます。トークンは通常24時間有効です。失効の際は401 (Unauthorized)レスポンスにて警告されますが、別トークンを要求することができます。

1. それではOpenStack サービスカタログを見てみましょう。

```
$ curl -s -X POST http://203.0.113.10:35357/v2.0/tokens ¥
-d '{"auth": {"passwordCredentials": {"username":"test-user", "password":"test-password"}, "tenantName":"test-project"}}' ¥
-H "Content-type: application/json" | jq .
```

2. JSONレスポンスを読むことで、カタログを把握することができます。

   これ以降のリクエストを楽にするため、トークンを環境変数へ設定します。

```
$ TOKEN=`curl -s -X POST http://203.0.113.10:35357/v2.0/tokens ¥
-d '{"auth": {"passwordCredentials": {"username":"test-user", "password":"test-password"}, "tenantName":"test-project"}}' ¥
-H "Content-type: application/json" |  jq -r .access.token.id`
```

   コマンドラインから、トークンを$TOKENとして参照できるようになりました。

3. あなたのサービスカタログからコンピュートなどのエンドポイントを選び、試してみましょう。例えば、(サーバー)インスタンスのリスト出力など。

```
$ curl -s ¥
-H "X-Auth-Token: $TOKEN" ¥
http://203.0.113.10:8774/v2/98333aba48e756fa8f629c83a818ad57/servers | jq .
```

APIリクエストがどのような構造かを知るには、OpenStack API Reference (http://api.openstack.org/api-ref.html)を参照してください。また、jqを使ってレスポンスを理解するには、jq Manual (http://stedolan.github.com/jq/manual/)も参考になるでしょう。

cURLの-sフラグを使うことで、プログレスメーター非表示にできます。もしcURLコマンドの実行に問題がある場合、それらを消したくなるでしょう。いっぽう、-vフラグは詳細を出力するため、cURLコマンドのトラブル解決に役立ちます。非常に便利な機能がcURLには多くあるので、manページでそれらのオプションを参照してください。

サーバーとサービス

管理者として、あなたのOpenStackクラウドがどのような状態か、ツールを使って把握する方法が、いくつかあります。この節では、あなたのクラウドの概要、形、大きさ、状態を得るアイデアをお伝えします。

まず、あなたのOpenStackクラウドに属し、稼働しているサーバーを把握することができます。

```
$ nova-manage service list | sort
```

出力は以下のようになります。

```
Binary           Host              Zone Status  State Updated_At
nova-cert        cloud.example.com nova enabled  :-)  2013-02-25 19:32:38
nova-compute     c01.example.com   nova enabled  :-)  2013-02-25 19:32:35
nova-compute     c02.example.com   nova enabled  :-)  2013-02-25 19:32:32
nova-compute     c03.example.com   nova enabled  :-)  2013-02-25 19:32:36
nova-compute     c04.example.com   nova enabled  :-)  2013-02-25 19:32:32
nova-compute     c05.example.com   nova enabled  :-)  2013-02-25 19:32:41
nova-consoleauth cloud.example.com nova enabled  :-)  2013-02-25 19:32:36
nova-network     cloud.example.com nova enabled  :-)  2013-02-25 19:32:32
nova-scheduler   cloud.example.com nova enabled  :-)  2013-02-25 19:32:33
```

この出力は、5つのコンピュートノードと1つのクラウドコントローラーがあることを示しています。:-) のような笑顔は、サービスが起動し、稼働中であることを表しています。もしサービスが動作していない場合、:-)はXXXへと変化します。なぜサービスがダウンしているのか、問題解決すべき印です。

Cinderを使っているのであれば、以下のコマンドを実行すると同様のリストが見られます。

```
$ cinder-manage host list | sort
```

```
host              zone
c01.example.com   nova
c02.example.com   nova
c03.example.com   nova
c04.example.com   nova
c05.example.com   nova
cloud.example.com nova
```

これら2つの表で、どのサーバーとサービスがあなたのクラウドを構成しているのか、概要を知ることができました。

また、認証サービス (Keystone)を使い、あなたのクラウドでどのサービスが使えるのか、また、どのエンドポイントがサービス向けに構成されているか、知ることができます。

下記コマンドを実行するには、あなたのシェル環境で正しく管理系の変数が設定されている必要があります。

```
$ keystone service-list
```

```
+-----+----------+----------+----------------------------+
| id  | name     | type     | description                |
+-----+----------+----------+----------------------------+
| ... | cinder   | volume   | Cinder Service             |
| ... | glance   | image    | OpenStack Image Service    |
| ... | nova_ec2 | ec2      | EC2 Service                |
| ... | keystone | identity | OpenStack Identity Service |
| ... | nova     | compute  | OpenStack Compute Service  |
+-----+----------+----------+----------------------------+
```

上記の出力は、5つのサービスが構成されていることを示しています。

各サービスのエンドポイントを見るには、

```
$ keystone endpoint-list
```

```
---+-----------------------------------------+--
   |               publicurl                 |
---+-----------------------------------------+--
   | http://example.com:8774/v2/%(tenant_id)s |
   |         http://example.com:9292         |
   |        http://example.com:8000/v1       |
   |       http://example.com:5000/v2.0      |
---+-----------------------------------------+--

---+-----------------------------------------+--
   |                adminurl                 |
---+-----------------------------------------+--
   | http://example.com:8774/v2/%(tenant_id)s |
   |         http://example.com:9292         |
   |        http://example.com:8000/v1       |
   |       http://example.com:5000/v2.0      |
---+-----------------------------------------+--
```

この例では、多くの出力のうち 2つの列だけを示しています。service と endpointは1対1でマッピングされます。いくつかのサービスではパブリックURLと管理URLが異なるので注意してください。

nova-manage コマンドを使ってインストールされている Compute のバージョンを確認できます。

```
$ nova-manage version list
```

コンピュートノードの詳細状況の取得

サーバー ID を指定して nova diagnostics コマンドを実行することで、実行中の仮想マシンに関するさらなる情報を取得できます。インスタンス毎の CPU 使用状況、メモリ、ディスク I/O、ネットワーク I/O が取得できます。

```
$ nova diagnostics <serverID>
```

このコマンドの出力はハイパーバイザーにより異なります。ハイパーバイザーが Xen の場合の出力は以下のようになります。

```
+----------------+-----------------+
|    Property    |      Value      |
+----------------+-----------------+
| cpu0           | 4.3627          |
| memory         | 1171088064.0000 |
| memory_target  | 1171088064.0000 |
| vbd_xvda_read  | 0.0             |
| vbd_xvda_write | 0.0             |
| vif_0_rx       | 3223.6870       |
| vif_0_tx       | 0.0             |
| vif_1_rx       | 104.4955        |
| vif_1_tx       | 0.0             |
+----------------+-----------------+
```

このコマンドは libvirt 経由で制御されるハイパーバイザーであれば (KVM, QEMU, LXC など) どれであっても動作するはずですが、テストされているのは KVM だけです。ハイパーバイザーが KVM の場合の出力は以下のようになります。

```
+------------------+------------+
| Property         | Value      |
+------------------+------------+
| cpu0_time        | 2870000000 |
| memory           | 524288     |
| vda_errors       | -1         |
| vda_read         | 262144     |
| vda_read_req     | 112        |
| vda_write        | 5606400    |
| vda_write_req    | 376        |
| vnet0_rx         | 63343      |
| vnet0_rx_drop    | 0          |
| vnet0_rx_errors  | 0          |
| vnet0_rx_packets | 431        |
| vnet0_tx         | 4905       |
| vnet0_tx_drop    | 0          |
| vnet0_tx_errors  | 0          |
```

```
| vnet0_tx_packets | 45         |
+------------------+------------+
```

ネットワーク

次に、あなたのクラウドでどのような固定IPネットワークが設定されているかを見てみましょう。nova novaコマンドラインクライアントを使って、構成されているIPアドレス空間を得ることができます。

```
$ nova network-list
+--------------------------------------+--------+--------------+
| ID                                   | Label  | Cidr         |
+--------------------------------------+--------+--------------+
| 3df67919-9600-4ea8-952e-2a7be6f70774 | test01 |  10.1.0.0/24 |
| 8283efb2-e53d-46e1-a6bd-bb2bdef9cb9a | test02 |  10.1.1.0/24 |
+--------------------------------------+--------+--------------+
```

nova-manage ツールを使うと、もう少し詳しい情報が表示されます。

```
$ nova-manage network list
id IPv4        IPv6 start address DNS1 DNS2 VlanID project   uuid
1  10.1.0.0/24 None 10.1.0.3      None None 300    2725bbd   beacb3f2
2  10.1.1.0/24 None 10.1.1.3      None None 301    none      d0b1a796
```

この出力は、2つのネットワークが構成され、それぞれが255のIPアドレス（/24 サブネット)を持つことを表しています。1つ目のネットワークはあるプロジェクトに割り当てられており、2つ目はまだ割り当てられていません。手動でプロジェクトに割り当てることもできますし、プロジェクトの1つ目のインスタンスを起動する際、自動的に割り当てることもできます。

利用可能なフローティングIPを確認するには、

```
$ nova-manage floating list
```

```
2725bbd458e2459a8c1bd36be859f43f 1.2.3.4 None                                    nova vlan20
None                             1.2.3.5 48a415e7-6f07-4d33-ad00-814e60b010ff nova vlan20
```

2つのフローティングIPが利用可能であることがわかります。1つめはプロジェクトに割り当て済み、もうひとつは未割り当てです。

ユーザーとプロジェクト

クラウドに追加されたプロジェクトのリストを見るためには、

```
$ keystone tenant-list
```

```
+-----+----------+---------+
| id  | name     | enabled |
+-----+----------+---------+
| ... | jtopjian | True    |
```

```
| ... | alvaro   | True    |
| ... | everett  | True    |
| ... | admin    | True    |
| ... | services | True    |
| ... | jonathan | True    |
| ... | lorin    | True    |
| ... | anne     | True    |
| ... | rhulsker | True    |
| ... | tom      | True    |
| ... | adam     | True    |
+-----+----------+---------+
```

ユーザーのリストを見るためには、

```
$ keystone user-list
```

```
+-----+----------+---------+-----------------------------+
| id  | name     | enabled | email                       |
+-----+----------+---------+-----------------------------+
| ... | everett  | True    | everett.towne@backspace.com |
| ... | jonathan | True    | jon@sfcu.edu                |
| ... | nova     | True    | nova@localhost              |
| ... | rhulsker | True    | ryan.hulkster@cyberalbert.ca |
| ... | lorin    | True    | lorinhoch@nsservices.com    |
| ... | alvaro   | True    | Alvaro.Perry@cyberalbert.ca |
| ... | anne     | True    | anne.green@backspace.com    |
| ... | admin    | True    | root@localhost              |
| ... | cinder   | True    | cinder@localhost            |
| ... | glance   | True    | glance@localhost            |
| ... | jtopjian | True    | joe.topjian@cyberalbert.com |
| ... | adam     | True    | adam@ossmanuals.net         |
| ... | tom      | True    | fafield@univm.edu.au        |
+-----+----------+---------+-----------------------------+
```

注記

ユーザーとグループが1対1でマッピングされることもあります。これはcinder、glance、novaやswiftなど標準のシステムアカウントや、グループに属すのが1ユーザーのみの場合がこれにあたります。

稼働中のインスタンス

以下のコマンドで、稼働中のインスタンスのリストが得られます。

```
$ nova list --all-tenants
```

```
+-----+------------------+--------+--------------------------------------------+
| ID  | Name             | Status | Networks                                   |
+-----+------------------+--------+--------------------------------------------+
| ... | Windows          | ACTIVE | novanetwork_1=10.1.1.3, 199.116.232.39     |
| ... | cloud controller | ACTIVE | novanetwork_0=10.1.0.6; jtopjian=10.1.2.3  |
| ... | compute node 1   | ACTIVE | novanetwork_0=10.1.0.4; jtopjian=10.1.2.4  |
| ... | devbox           | ACTIVE | novanetwork_0=10.1.0.3                     |
| ... | devstack         | ACTIVE | novanetwork_0=10.1.0.5                     |
| ... | initial          | ACTIVE | nova_network=10.1.7.4, 10.1.8.4            |
| ... | lorin-head       | ACTIVE | nova_network=10.1.7.3, 10.1.8.3            |
+-----+------------------+--------+--------------------------------------------+
```

残念ながら、このコマンドは稼働中のインスタンスの詳細を出力しません。例えば、どのコンピュートノードでインスタンスが動いているのか、フレーバーは何か、などなど。あなたは以下のコマンドでインスタンス個別の詳細情報を得ることができます。

```
$ nova show <uuid>
```

例えば、

```
# nova show 81db556b-8aa5-427d-a95c-2a9a6972f630
```

```
+-------------------------------------+----------------------------------+
| Property                            | Value                            |
+-------------------------------------+----------------------------------+
| OS-DCF:diskConfig                   | MANUAL                           |
| OS-EXT-SRV-ATTR:host                | c02.example.com                  |
| OS-EXT-SRV-ATTR:hypervisor_hostname | c02.example.com                  |
| OS-EXT-SRV-ATTR:instance_name       | instance-00000029                |
| OS-EXT-STS:power_state              | 1                                |
| OS-EXT-STS:task_state               | None                             |
| OS-EXT-STS:vm_state                 | active                           |
| accessIPv4                          |                                  |
| accessIPv6                          |                                  |
| config_drive                        |                                  |
| created                             | 2013-02-13T20:08:36Z             |
| flavor                              | m1.small (6)                     |
| hostId                              | ...                              |
| id                                  | ...                              |
| image                               | Ubuntu 12.04 cloudimg amd64 (...) |
| key_name                            | jtopjian-sandbox                 |
| metadata                            | {}                               |
| name                                | devstack                         |
| novanetwork_0 network               | 10.1.0.5                         |
| progress                            | 0                                |
| security_groups                     | [{u'name': u'default'}]          |
| status                              | ACTIVE                           |
| tenant_id                           | ...                              |
| updated                             | 2013-02-13T20:08:59Z             |
| user_id                             | ...                              |
+-------------------------------------+----------------------------------+
```

第9章 プロジェクトとユーザーの管理

プロジェクトかテナントか? 83
プロジェクトの管理 83
クォータ 85
ユーザー管理 90
新規ユーザーの作成 90
プロジェクトへのユーザーの割り当て 91

OpenStack クラウドはユーザーがいないと、あまり価値がありません。本章はユーザー、プロジェクトおよびクォータの管理に関する話題を説明します。

プロジェクトかテナントか?

OpenStack のユーザーインターフェースおよびドキュメントにおいて、ユーザーのグループはプロジェクトまたはテナントと呼ばれます。これらの語は同じ意味で用いられます。

OpenStack Compute サービス (Nova) の初期実装は独自の認証システムを持ち、プロジェクトという用語を使用していました。認証が OpenStack Identity サービス (Keystone) プロジェクトに移行したとき、ユーザーのグループを意味する用語としてテナントという用語が使用されました。このような経緯のため、いくつかの OpenStack ツールはプロジェクトを使用し、いくつかはテナントを使用します。

このガイドはプロジェクトという用語を使用します。テナントという用語を使用するツールとやりとりする例もあります。

プロジェクトの管理

ユーザーは少なくとも一つのプロジェクトに所属しなければいけませんが、複数のプロジェクトに所属することもできます。そのため、ユーザーを追加する前に、少なくとも一つのプロジェクトを追加する必要があります。

プロジェクトの追加

ダッシュボードからプロジェクトを追加する方法:

1. 管理ユーザーとしてログインします。

2. 左側のナビゲーションバーにある「プロジェクト」リンクを選択します。

3. 右上にある「プロジェクトの作成」ボタンをクリックします。

プロジェクト名と説明の入力が求められます（説明の入力は任意ですが、入力をお薦めします）。このプロジェクトを有効にするには、フォームの下にあるチェックボックスを選択します。これは標準で有効になっています。

プロジェクトメンバーを追加し、プロジェクトのクォータを調整することもできます。後述しますが、これらの操作を一度にすべて処理するほうが、実践的には非常に便利です。

コマンドラインインターフェース（CLI）からプロジェクトを作成する方法:

コマンドラインからのプロジェクト追加には、keystone ユーティリティを使用する必要があります。ここで「プロジェクト」の代わりに「テナント」を使用します。

```
# keystone tenant-create --name=demo
```

このコマンドは「demo」という名前のプロジェクトを作成します。--description tenant-description を追加で指定することで、説明の文字列を付与できます。これは非常に便利です。また、コマンドに --enabled false を追加することにより、無効状態のグループを作成することもできます。標準で、プロジェクトは有効状態で作成されます。

クォータ

To prevent system capacities from being exhausted without notification, you can set up quotas. Quotas are operational limits. For example, the number of gigabytes allowed per tenant can be controlled to ensure that a single tenant cannot consume all of the disk space. Quotas are currently enforced at the tenant (or project) level, rather than by user.

コマンドラインインターフェースを使って、OpenStack Compute サービスと Block Storage サービスのクォータを管理できます。

Typically, default values are changed because a tenant requires more than the OpenStack default of 10 volumes per tenant, or more than the OpenStack default of 1TB of disk space on a Compute node.

注記

全てのテナントを表示するには、以下のコマンドを実行します。

```
$ keystone tenant-list
+----------------------------------+----------+---------+
|                id                |   name   | enabled |
+----------------------------------+----------+---------+
| a981642d22c94e159a4a6540f70f9f8d |  admin   |   True  |
| 934b662357674c7b9f5e4ec6ded4d0e7 | tenant01 |   True  |
| 7bc1dbfd7d284ec4a856ea1eb82dca80 | tenant02 |   True  |
| 9c554aaef7804ba49e1b21cbd97d218a | services |   True  |
+----------------------------------+----------+---------+
```

コンピュートサービスのクォータの設定

管理ユーザーは、既存のテナントの Compute サービスのクォータを更新できます。また、新規テナントのクォータのデフォルト値を更新することもできます。

表9.1 コンピュートのクォータの説明

クォータ	説明	項目名
固定 IP	テナント毎の固定 IP アドレスの最大数。この数はテナント毎の最大インスタンス数以上にしなければなりません。	fixed-ips

クォータ	説明	項目名
Floating IP	テナントごとの最大 Floating IP 数	floating-ips
injected file のバイト数	injected file あたりの最大バイト数	injected-file-content-bytes
injected file のパス長	injected file のパス長の最大バイト数	injected-file-path-bytes
injected file	injected file の最大数	injected-files
インスタンス	テナントごとの最大インスタンス数	instances
キーペア	ユーザーごとの最大キーペア数	key-pairs
メタデータ項目数	インスタンスごとのメタデータ項目数	metadata-items
メモリー	テナントごとのインスタンスの RAM 容量（メガバイト単位）	ram
セキュリティグループルール	セキュリティグループごとのセキュリティルール数	security-group-rules
セキュリティグループ	テナントごとのセキュリティグループ数	security-groups
仮想 CPU	テナントごとのインスタンスのコア数	cores

Compute サービスのテナント（プロジェクト）の クォータの表示と更新

管理ユーザーは nova quota-* コマンドを使って、テナントのクォータを表示したり更新したりできます。コマンドは python-novaclient パッケージに含まれます。

デフォルトのクォータ値の表示と更新

1. 全テナントに対するクォータのデフォルト値を全て表示するには、以下のようにします。

```
$ nova quota-defaults
```

例えば

```
$ nova quota-defaults
+-----------------------------+-------+
| Property                    | Value |
+-----------------------------+-------+
| metadata_items              | 128   |
| injected_file_content_bytes | 10240 |
| ram                         | 51200 |
| floating_ips                | 10    |
| key_pairs                   | 100   |
| instances                   | 10    |
| security_group_rules        | 20    |
| injected_files              | 5     |
```

```
| cores                       | 20    |
| fixed_ips                   | -1    |
| injected_file_path_bytes    | 255   |
| security_groups             | 10    |
+-----------------------------+-------+
```

2. 新規テナントに対するクォータのデフォルト値を更新するには、以下のようにします。

```
$ nova quota-class-update default key value
```

例えば

```
$ nova quota-class-update default instances 15
```

テナント（プロジェクト）のクォータ値の表示

1. テナント ID を変数に格納しておきます。

```
$ tenant=$(keystone tenant-list | awk '/tenantName/ {print $2}')
```

2. テナントの現在のクォータ値を一覧表示します。

```
$ nova quota-show --tenant $tenant
```

例えば

```
$ nova quota-show --tenant $tenant
+-----------------------------+-------+
| Property                    | Value |
+-----------------------------+-------+
| metadata_items              | 128   |
| injected_file_content_bytes | 10240 |
| ram                         | 51200 |
| floating_ips                | 12    |
| key_pairs                   | 100   |
| instances                   | 10    |
| security_group_rules        | 20    |
| injected_files              | 5     |
| cores                       | 20    |
| fixed_ips                   | -1    |
| injected_file_path_bytes    | 255   |
| security_groups             | 10    |
+-----------------------------+-------+
```

テナント（プロジェクト）のクォータ値の更新

1. テナント ID を取得します。

```
$ tenant=$(keystone tenant-list | awk '/tenantName/ {print $2}')
```

2. 指定したクォータ値を更新します。

```
# nova quota-update --quotaName quotaValue tenantID
```

例えば

```
# nova quota-update --floating-ips 20 $tenant
# nova quota-show --tenant $tenant
+-----------------------------+-------+
| Property                    | Value |
+-----------------------------+-------+
| metadata_items              | 128   |
| injected_file_content_bytes | 10240 |
| ram                         | 51200 |
| floating_ips                | 20    |
| key_pairs                   | 100   |
| instances                   | 10    |
| security_group_rules        | 20    |
| injected_files              | 5     |
| cores                       | 20    |
| fixed_ips                   | -1    |
| injected_file_path_bytes    | 255   |
| security_groups             | 10    |
+-----------------------------+-------+
```

注記

quota-update コマンドのオプションリストを表示するには、以下のようにします。

```
$ nova help quota-update
```

ブロックストレージのクォータの設定

管理ユーザーは、既存のテナントの Block Storage サービスのクォータを更新できます。また、新規テナントのクォータのデフォルト値を更新することもできます。

表9.2 ブロックストレージのクォータの設定

項目名	説明
gigabytes	テナントごとのボリューム容量の最大値（単位はギガバイト）
snapshots	テナントごとのブロックストレージスナップショット数
volumes	テナントごとのブロックストレージボリューム数

Block Storage サービスのテナント（プロジェクト）のクォータの表示と更新

管理ユーザーは cinder quota-* コマンドを使って、テナントのクォータを表示したり更新したりできます。コマンドは python-cinderclient パッケージに含まれます。

Block Storage のデフォルトのクォータ値の表示と更新

1. 全テナントに対するクォータのデフォルト値を全て表示するには、以下のようにします。

```
$ cinder quota-defaults
```

例えば

```
$ cinder quota-defaults
+-----------+-------+
|  Property | Value |
+-----------+-------+
| gigabytes |  1000 |
| snapshots |   10  |
|  volumes  |   10  |
+-----------+-------+
```

2. 新規テナントのクォータのデフォルト値を更新するには、/etc/cinder/cinder.conf ファイルの対応する項目を更新します。

テナントの Block Storage クォータを表示する

- 特定のテナントのクォータを表示するには以下のようにします。

```
# cinder quota-show tenantName
```

例えば

```
# cinder quota-show tenant01
+-----------+-------+
|  Property | Value |
+-----------+-------+
| gigabytes |  1000 |
| snapshots |   10  |
|  volumes  |   10  |
+-----------+-------+
```

テナントの Block Storage クォータを更新する

1. テナント ID を変数に格納しておきます。

```
$ tenant=$(keystone tenant-list | awk '/tenantName/ {print $2}')
```

2. 指定したクォータ値を更新します。

```
# cinder quota-update --quotaName NewValue tenantID
```

例えば

```
# cinder quota-update --volumes 15 $tenant
# cinder quota-show tenant01
+-----------+-------+
|  Property | Value |
+-----------+-------+
| gigabytes |  1000 |
| snapshots |   10  |
|  volumes  |   15  |
+-----------+-------+
```

ユーザー管理

直接コマンドラインツールを使ってユーザーを管理することは面倒です。一つの作業を完了するために複数のコマンドを実行する必要がありますし、多くの項目でシンボル名の代わりに UUID が使用されています。現実的に、人間はこれらのツールをそのまま使用しません。幸運なことに、OpenStack Dashboardが便利なインターフェースを提供しています。さらに、多くのサイトは個別の要求を満たすために独自ツールを作成し、サイト固有のポリシーを適用し、パッケージツールでは実現できないレベルのセルフサービスをユーザーに提供しています。

新規ユーザーの作成

ユーザーを作成するには、以下の情報が必要です。

- ユーザー名
- 電子メールアドレス
- パスワード
- 主プロジェクト
- 役割

ユーザー名と電子メールアドレスは見たとおりです。あなたのサイトは従うべき独自ルールがあるかもしれません。Identity サービスにおいて

パスワードを設定および変更するには、管理者権限が必要です。Folsom リリースまでは、ユーザーが自分のパスワードを変更できません。これは独自のツールを作成する大きな理由になります。また、パスワードを割り当ておよび配布するときに気をつける必要があります。主プロジェクトは、単にそのユーザーが割り当てられる最初のプロジェクトです。このプロジェクトはユーザーを作成する前に存在している必要があります。役割は多くの場合ずっと「メンバー」のままになります。標準の状態で、OpenStack は次の 2 つの役割が定義されています。

- 「member」: 一般的なユーザー。
- 「admin」: すべてのプロジェクトにわたり全権限を持つ管理ユーザー。非常に注意して使用する必要があります。

他の役割を定義できますが、一般的にはそうしません。

一旦これらの情報が揃ったら、ダッシュボードでのユーザーの作成は、これまでに見てきた他の Web フォームと同じです。「管理」ナビゲーションバーの「ユーザー」リンクにあります。そして、右上にある「ユーザーの作成」ボタンをクリックします。

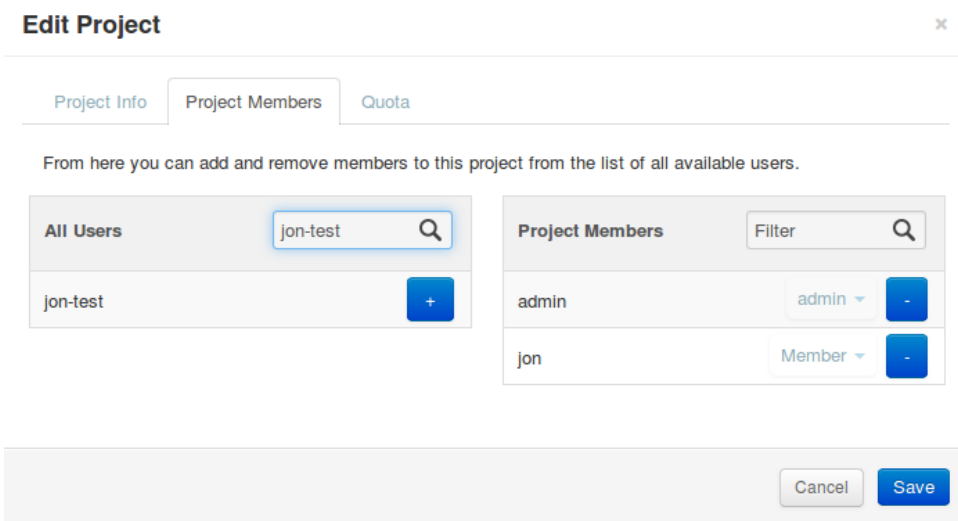
ユーザー情報の変更は、この「ユーザー」ページから実行することもできます。多数のユーザーがいる場合、このページにはたくさんのユーザーが表示されてしまいます。ページの上部にある「フィルター」検索ボックスを使うと、表示されるユーザーの一覧を絞り込むことができます。変更しようとしているユーザーの行末にあるアクションドロップダウンメニューの「編集」を選択することにより、ユーザー作成ダイアログと非常に似ているフォームを表示できます。

プロジェクトへのユーザーの割り当て

多くのサイトは一つのプロジェクトのみに割り当てられているユーザーで実行しています。これは、管理者にとってもユーザーにとっても、より保守的で分かりやすい選択です。管理の面では、ユーザーからインスタンスやクォータに関する問題の報告があった場合、どのプロジェクトに関するものかが明確です。ユーザーが一つのプロジェクトのみに所属している場合、ユーザーがどのプロジェクトで操作しているのかを気にする必要がありません。ただし、既定の設定では、どのユーザーも同じプロジェクトにいる他のユーザーのリソースに影響を与えることができることに注意してください。あなたの組織にとって意味があるならば、ユーザーを複数のプロジェクトに割り当てることも可能です。

ダッシュボードの「プロジェクト」ページから「アクション」列の「ユーザーの変更」を選んで、既存のユーザーへの追加のプロジェクト

の割り当てや、古いプロジェクトからのユーザー削除を実行することができます。



このビューから、多くの有用な操作、いくつかの危険な操作を実行できます。

「すべてのユーザー（All Users)」という見出しが付けられた、このフォームの最初の列に、このプロジェクトにまだ割り当てられていない、クラウドのすべてのユーザーが一覧表示されます。2 列目には、すべての割り当て済みユーザーが一覧表示されます。これらの一覧は非常に長くなる可能性がありますが、それぞれの列の上部にあるフィルターフィールドに、探しているユーザー名の部分文字列を入力することにより、表示を絞り込むことができます。

ここから、プロジェクトにユーザーを追加するには + アイコンをクリックします。削除するには - をクリックします。

危険な点としては、メンバーの役割を変更する機能があることです。「プロジェクトメンバー」一覧のユーザー名の後ろにあるドロップダウンリストです。実際すべての場合で、この値は「Member」に設定すべきです。この例では意図的に、この値が 「admin」になっている管理ユーザーを示しています。

重要

「admin」は各プロジェクトの管理者ではなく、システム全体の管理者です。そのため、ユーザーに管理者の役割を与える

ことにより、クラウド全体にわたる管理者権限を与えることになります。

一般的な使用法は、一つのプロジェクトだけに管理ユーザーを所属させることです。慣例により、「admin」プロジェクトがクラウド環境のセットアップ中に標準で作成されます。管理ユーザーもクラウドを使用してインスタンスの起動、管理を行う場合には、管理アクセスと一般アクセス用に別々のユーザーアカウントを使用し、それらのユーザーを別々のプロジェクトにすることを強く推奨します。

権限のカスタマイズ

デフォルトの認可設定では、管理ユーザーのみが他のプロジェクトのリソースを作成できます。OpenStack では以下の 2 種類の認可ポリシーを使うことができます。

- 操作ベース: 特定の操作に対するアクセス基準を指定するポリシー。特定の属性に対する詳細な制御も可能です。
- リソースベース: リソースに対して設定されたパーミッションに基づいて、特性のリソースに対するアクセスを許可するかを決定する（今のところネットワークリソースでのみ利用可能)。OpenStack により強制される実際の認可ポリシーは、導入の仕方により異なります。

ポリシーエンジンは policy.json ファイルから項目を読み込みます。このファイルの実際の位置はディストリビューションにより異なります。一般的に Nova 用の設定ファイルは /etc/nova/policy.json にあります。システムの実行中に項目を更新でき、サービスを再起動する必要がありません。今のところ、ポリシーファイルの編集がこのようなポリシーを更新する唯一の方法です。

OpenStack サービスのポリシーエンジンがポリシーと直接照合を行います。ルールはそのようなポリシーの要素の評価を意味します。たとえば、compute:create: [["rule:admin_or_owner"]] 文において、ポリシーは compute:create で、ルールは admin_or_owner です。

ポリシーのいずれかが OpenStack API 操作、もしくは指定された操作で使用されている特定の属性に一致する場合、ポリシーが OpenStack ポリシーエンジンにより呼び出されます。たとえば、ユーザーが POST /v2/{tenant_id}/servers リクエストを OpenStack Compute API サーバーに送信したときに必ず、エンジンが create:compute ポリシーを評価します。ポリシーは特定の API 拡張に関連づけることもできます。たとえば、ユーザーが compute_extension:rescue のような拡張に対して要求を行った場合、プロバイダー拡張により定義された属性は、その操作に対するルールテストを呼び出します。

認可ポリシーは、一つまたは複数のルールにより構成できます。複数のルールを指定した場合、いずれかのルールが成功と評価されれば、ポリシーの評価は成功となります。API 操作が複数のポリシーに一致すると、すべてのポリシーが成功と評価される必要があります。また、認可ルールは再帰的にもできます。あるルールにマッチした場合、これ以上展開できないルールに達するまで、そのルールは別のルールに展開されます。以下のルールが定義できます。

- 役割に基づいたルール: リクエストを出したユーザーが指定された役割を持っていれば、成功と評価されます。たとえば、リクエストを出しているユーザーが管理者ならば、”role:admin” が成功します。
- 項目に基づいたルール: 現在のリクエストに指定されたリソースの項目が指定された値と一致すれば、成功と評価されます。たとえば、ネットワークリソースの shared 属性が True に設定されている場合、”field:networks:shared=True”が成功します。
- 一般的なルール: リソースの属性をユーザーのセキュリティクレデンシャルから抽出した属性と比較し、一致した場合に成功と評価されます。たとえば、リソースのテナント識別子がリクエストを出したユーザーのテナント識別子と一致すれば、”tenant_id:%(tenant_id)s” が成功します。

これは標準の nova policy.json ファイルの抜粋です。

```
{
”context_is_admin”:  [[”role:admin”]],
”admin_or_owner”:  [[”is_admin:True”], [”project_id:%(project_id)s”]], [1]
”default”: [[”rule:admin_or_owner”]], [2]
”compute:create”: [ ],
”compute:create:attach_network”: [ ],
”compute:create:attach_volume”: [ ],
”compute:get_all”: [ ],
   ”admin_api”: [[”is_admin:True”]],
”compute_extension:accounts”: [[”rule:admin_api”]],
”compute_extension:admin_actions”: [[”rule:admin_api”]],
”compute_extension:admin_actions:pause”: [[”rule:admin_or_owner”]],
”compute_extension:admin_actions:unpause”: [[”rule:admin_or_owner”]],
...
”compute_extension:admin_actions:migrate”: [[”rule:admin_api”]],
”compute_extension:aggregates”: [[”rule:admin_api”]],
”compute_extension:certificates”: [ ],
...
”compute_extension:flavorextraspecs”: [ ],
”compute_extension:flavormanage”: [[”rule:admin_api”]],  [3]
}
```

[1] 現在のユーザーが、管理者、またはリクエストで指定されたリソースの所有者（テナント識別子が同じ）であれば、成功であると評価されるルールを表します。

[2] API 操作が policy.json のどのポリシーとも一致しなかった場合に、必ず評価される規定のポリシーを表します。

[3] フレーバーを操作する権限を、管理 API を使用する管理者だけに限定するポリシーを表します。

いくつかの場合では、いくつかの操作を管理者のみに制限すべきです。そこで、次の例では、ユーザーが自分のフレーバーを作成できるようにするシナリオの場合に、このサンプルのポリシーファイルをどのように変更すればよいかを示します。

```
"compute_extension:flavormanage": [ ],
```

他のユーザーに悪影響を与えるユーザー

クラウドのユーザーは他のユーザーに悪影響を与える場合があります。意図的に悪意を持って行われる場合もあれば、偶然起こる場合もあります。状況を理解することで、このような混乱への対処方法について、よりよい判断ができるようになります。

例えば、あるユーザーのグループが、非常に計算負荷の高い作業用に大量のコンピュートリソースを使うインスタンスを持っているとします。これにより、コンピュートノードの負荷が高くなり、他のユーザーに影響が出ます。この状況では、ユーザーのユースケースを精査する必要があります。計算負荷が高いシナリオがよくあるケースだと判明し、ホスト集約やリージョンなど、クラウドを適切に分割することを計画すべき場合もあるでしょう。

別の例は、あるユーザーが非常に多くの帯域を消費する場合です。繰り返しですが、ユーザーが実行していることを理解することが重要です。多くの帯域を必ず使用する必要がある場合には、他のユーザーに影響を与えないように通信帯域を制限する、または、より多くの帯域を利用可能な別の場所に移動させる必要があるかもしれません。一方、ユーザーのインスタンスが侵入され、DDOS 攻撃を行っているボットネットの一部になっているかもしれません。この問題の解決法は、ネットワークにある他のサーバーが侵入された場合と同じです。ユーザーに連絡し、対応する時間を与えます。もし対応しなければ、そのインスタンスを停止します。

最後の例は、ユーザーがクラウドのリソースに繰り返し悪影響を与える場合です。ユーザーと連絡をとり、何をしようとしているのか理解します。ユーザー自身が実行しようとしていることを正しく理解していない可能性があります。または、アクセスしようとしているリソースに問題があり、リクエストがキューに入ったり遅れが発生している場合もあります。

システム管理の見過ごされがちな大事な要素の一つに、エンドユーザのためにシステム管理者が存在するという点があります。BOFH (Bastard

Operator From Hell; 「地獄から来た最悪の管理者」）の道に入って、問題の原因となっているユーザーを全員停止させるようなことはしないでください。ユーザーがやりたいことを一緒になって理解し、どうするとあなたの環境がユーザーが目的を達成するのにもっと支援できるかを見つけてください。

第10章 ユーザーによる運用

イメージ 97
フレーバー 99
セキュリティグループ 101
ブロックストレージ 105
インスタンス 106
セキュリティグループの割り当て 110
Floating IP 111
ブロックストレージの接続 111
スナップショットの取得 112
データベースにあるインスタンス 116

このガイドは OpenStack の運用者向けです。ユーザー向けの膨大なリファレンスを目指すものではありません。しかし運用者として、クラウド設備を使用する方法について基本的な理解を持つことが重要です。本章は、基本的なユーザーの観点から OpenStack を見ていきます。ユーザーが必要とすることを理解する手助けになります。また、トラブルのチケットを受け取ったときに、ユーザーの問題またはサービスの問題のどちらかを判断する手助けになります。取り扱っている主な概念はイメージ、フレーバー、セキュリティグループ、ブロックストレージおよびインスタンスです。

イメージ

OpenStack のイメージはしばしば「仮想マシンテンプレート」と考えることができます。イメージは ISO イメージのような標準的なインストールメディアの場合もあります。基本的に、インスタンスを起動するために使用されるブート可能なファイルシステムを含みます。

イメージの追加

いくつかの構築済みイメージが存在します。簡単に Image Service の中にインポートできます。追加する一般的なイメージは、非常に小さく、テスト目的に使用される CirrOS イメージです。このイメージを追加するには、単に次のようにします。

```
# wget https://launchpad.net/cirros/trunk/0.3.0/+download/cirros-0.3.0-x86_64-disk.img # glance
 image-create --name='cirros image' --is-public=true --container-format=bare --disk-format=qcow2 <
cirros-0.3.0-x86_64-disk.img
```

glance image-create コマンドでは、イメージに指定できる多数のオプションが用意されています。たとえば、min-disk オプションは、特定

の容量のルートディスクを必要とするイメージ（例: 大きな Windows イメージ）のために有用です。これらのオプションを表示するには、次のようにします。

```
$ glance help image-create
```

location オプションは注意する意味があります。Glance にイメージ全体のコピーを行わず、そのイメージがある元の位置への参照を保持します。イメージのインスタンスを起動するとき、Glance が指定された場所からイメージにアクセスします。

copy-from オプションは、指定された位置から /var/lib/glance/images ディレクトリの中にコピーします。例に示されたように STDIN リダイレクションを使用するときに、同じことが実行されます。

既存のイメージのプロパティを表示するために、以下のコマンドを実行します。

```
$ glance details
```

イメージの削除

イメージを削除するには、ただ次のとおり実行します。

```
$ glance image-delete <image uuid>
```

注記

イメージを削除しても、そのイメージがベースになっているインスタンスやスナップショットには影響がありません。

他の CLI オプション

オプションの完全な一覧は、以下を使用して見つけられます。

```
$ glance help
```

または OpenStack Image Service CLI Guide. (http://docs.openstack.org/cli/quick-start/content/glance-cli-reference.html)

イメージサービスおよびデータベース

Glance がデータベースに保存しない唯一のものがイメージそのものです。Glance データベースは主な 2 つのテーブルを持ちます。

- images

- image_properties

データベースと SQL クエリーを直接使うことで、Glance イメージの独自のリストやレポートを得ることができます。一般には、推奨されませんが、技術的にはデータベース経由でイメージのプロパティを更新できます。

イメージサービスのデータベースクエリーの例

興味深い例の一つは、イメージとそのイメージの所有者の表の表示内容を変更することです。これは、所有者のユニーク ID を表示するようにするだけで実現できます。この例はさらに一歩進め、所有者の読みやすい形式の名前を表示します。

```
$ mysql> select glance.images.id, glance.images.name, keystone.tenant.name, is_public from glance.
images inner join keystone.tenant on glance.images.owner=keystone.tenant.id;
```

もう一つの例は、特定のイメージに関するすべてのプロパティを表示することです。

```
$ mysql> select name, value from image_properties where id = <image_id>
```

フレーバー

仮想ハードウェアのテンプレートは、OpenStack において「フレーバー」と呼ばれます。これは、RAM、ディスク、コア数などを定義します。標準のインストールでは、5 種類のフレーバーが提供されます。これらは管理ユーザーにより編集可能です（これは設定可能であり、nova-api サーバーにおいて /etc/nova/policy.json にある compute_extension:flavormanage に対するアクセス制御を再定義することにより権限委譲できます）。お使いのシステムにおいて利用可能なフレーバーの一覧を取得するには、次のとおり実行します。

```
$ nova flavor-list
```

```
+----+-----------+-----------+------+-----------+¥+-------+-¥+-------------+
| ID | Name      | Memory_MB | Disk | Ephemeral |/| VCPUs | /| extra_specs |
+----+-----------+-----------+------+-----------+¥+-------+-¥+-------------+
| 1  | m1.tiny   | 512       | 1    | 0         |/| 1     | /| {}          |
| 2  | m1.small  | 2048      | 10   | 20        |¥| 1     | ¥| {}          |
| 3  | m1.medium | 4096      | 10   | 40        |/| 2     | /| {}          |
| 4  | m1.large  | 8192      | 10   | 80        |¥| 4     | ¥| {}          |
| 5  | m1.xlarge | 16384     | 10   | 160       |/| 8     | /| {}          |
+----+-----------+-----------+------+-----------+¥+-------+-¥+-------------+
```

nova flavor-create コマンドにより、権限のあるユーザーが新しいフレーバーを作成できます。さらなるフレーバーの操作コマンドは次のコマンドを用いて表示できます:

```
$ nova help | grep flavor
```

フレーバーは数多くの要素を定義します。

項目	説明
ID	一意な数値 ID。
名前	慣習として xx.size_name などの内容を表す名前を使用しますが、必須ではありません。いくつかのサードパーティツールはその名称に依存しているかもしれません。
Memory_MB	メガバイト単位の仮想マシンメモリー。
ディスク	ギガバイト単位の仮想ルートディスク容量。これはベースイメージがコピーされる一時ディスクです。永続的なボリュームからブートするとき、これは使用されません。「0」という容量は特別な値で、一時ルートボリュームの容量としてベースイメージのネイティブ容量をそのまま使用することを意味します。
エフェメラル	二次的な一時データディスクの容量を指定します。これは空の、フォーマットされていないディスクです。インスタンスの生存期間だけ存在します。
スワップ	インスタンスに割り当てられるスワップ空間。これはオプションです。
仮想 CPU	インスタンスに存在する仮想 CPU 数。
RXTX_Factor	作成したサーバーが接続されたネットワークにおける定義と異なる帯域制限を持てるようにするプロパティ。これはオプションです。この要素はネットワークの rxtx_base プロパティの倍数です。既定の値は 1.0 です（つまり、接続されたネットワークと同じです）。
Is_Public	論理値。フレーバーがすべてのユーザーに利用可能か、または作成されたプロジェクト内のみであるか。標準で真（True）です。
extra_specs	フレーバーを実行できるコンピュートノードに関する追加の制限。これはオプションです。これは、コンピュートノードにおいて対応するキー/バリューペアとして実装され、コンピュートノードでの対応するキー/バリューペアと一致するものでなければいけません。(GPU ハードウェアを持つコンピュートノードのみにおいて実行するフレーバーのように）特別なリソースのようなものを実装するために使用できます。

どのように既存のフレーバーを変更しますか?

残念ながら、OpenStack ではフレーバーを変更するインターフェースは提供されておらず、作成および削除のインターフェースだけがあります。OpenStack ダッシュボードは、既存のフレーバーを削除し、同じ名前の新しいものを作成することにより、フレーバーを変更する機能を模倣しています。

セキュリティグループ

OpenStack の新しいユーザーがよく経験する問題が、インスタンスを起動するときに適切なセキュリティグループを設定できず、その結果、ネットワーク経由でインスタンスにアクセスできないというものです。

セキュリティグループは、インスタンスのネットワークに適用される、IP フィルタールールの組です。それらはプロジェクト固有です。プロジェクトメンバーがそれらのグループの標準ルールを編集でき、新しいルールを追加できます。すべてのプロジェクトが「default」セキュリティグループを持ちます。他のセキュリティグループが定義されていないインスタンスには「default」セキュリティグループが適用されます。「default」セキュリティグループは、ルールを変更しない限り、すべての受信トラフィックを拒否します。

nova.conf のオプション allow_same_net_traffic（標準で true）は、同じネットワークを共有するホストにルールを適用するかを制御します。このオプションはシステム全体に影響するグローバルオプションです。true に設定したとき、同じサブネットにあるホストはフィルターされず、それらの間ですべての種類の通信が通過できるようになります。フラットなネットワークでは、これにより、全プロジェクトの全インスタンスが通信をフィルターされなくなります。VLAN ネットワークでは、これにより、同じプロジェクト内のインスタンス間でアクセスが許可されます。allow_same_net_traffic が false に設定されていると、セキュリティグループがすべての通信に対して強制されます。この場合、既定のセキュリティグループをそれらのサブネットからのすべての通信を許可するよう設定することにより、プロジェクトが allow_same_net_traffic をシミュレートできます。

現在のプロジェクトのセキュリティグループが、Horizon ダッシュボードの「アクセス & セキュリティ」にあります。既存のグループの詳細を表示するには、セキュリティグループの「編集」を選択します。自明ですが、この「編集」インターフェースから既存のグループを変更できます。新しいグループを作成するための「セキュリティグループの作成」ボタンが、メインの「アクセス & セキュリティ」ページにあります。同等のコマンドラインを説明するとき、これらの項目において使用される用語について説明します。

コマンドラインからは、以下の nova コマンドを使って、現在のプロジェクトのセキュリティグループのリストを取得できます。

```
$ nova secgroup-list
```

```
+---------+-------------+
| Name    | Description |
+---------+-------------+
| default | default     |
| open    | all ports   |
+---------+-------------+
```

「open」セキュリティグループの詳細を表示する方法:

```
$ nova secgroup-list-rules open
```

```
+-------------+-----------+---------+-----------+--------------+
| IP Protocol | From Port | To Port | IP Range  | Source Group |
+-------------+-----------+---------+-----------+--------------+
| icmp        | -1        | 255     | 0.0.0.0/0 |              |
| tcp         | 1         | 65535   | 0.0.0.0/0 |              |
| udp         | 1         | 65535   | 0.0.0.0/0 |              |
+-------------+-----------+---------+-----------+--------------+
```

標準で拒否されるので、これらのルールはすべて「許可」形式のルールです。1 番目の項目は IP プロトコル（icmp, tcp, udp のどれか）です。2 番目と 3 番目の項目は対象となるポート番号の範囲を指定します。4 番目の項目は CIDR 形式の IP アドレスの範囲を指定します。この例では、すべてのプロトコルの全ポート番号について、すべての IP からのトラフィックを許可しています。

前の章で述べたとおり、セキュリティグループごとのルール数は quota_security_group_rules により制御されます。また、プロジェクトごとに許可されるセキュリティグループ数は quota_security_groups クォータにより制御されます。

新しいセキュリティグループを追加するとき、内容を表す簡潔な名前をつけるべきです。この名前はインスタンスの簡単な説明など、より長い説明フィールドが使用されないところで使用されます。インスタンスがセキュリティグループ「http」を使っているのを見れば、「bobs_group」や「secgrp1」よりはずっと理解しやすいことでしょう。

例のとおり、インターネットのどこからでも Web 通信を許可するセキュリティグループを作成しましょう。このグループを「global_http」と呼ぶことにします。許可されるものと許可されるところを要約した、明白で簡潔な名前になっています。コマンドラインから、

```
$ nova secgroup-create global_http "allow web traffic from the internet"
```

```
+-------------+-----------------------------------+
| Name        | Description                       |
+-------------+-----------------------------------+
| global_http | allow web traffic from the internet |
+-------------+-----------------------------------+
```

やりたいことを行うための空のセキュリティグループが作成されます。いくつかのルールを追加する必要があります。

```
$ nova secgroup-add-rule <secgroup> <ip-proto> <from-port> <to-port>
                         <cidr>
$ nova secgroup-add-rule global_http tcp 80 80 0.0.0.0/0
+-------------+-----------+---------+-----------+--------------+
| IP Protocol | From Port | To Port | IP Range  | Source Group |
+-------------+-----------+---------+-----------+--------------+
| tcp         | 80        | 80      | 0.0.0.0/0 |              |
+-------------+-----------+---------+-----------+--------------+
```

引数の順番が決まっていることに注意してください。そして、「from-port」と「to-port」の引数は許可されるローカルのポート範囲を指定し、接続の送信元ポートと宛先ポートではないことに注意してください。nova secgroup-add-rule を複数回呼び出すことで、より複雑なルールセットを構成できます。たとえば、http と https の通信を通過させたい場合、

```
$ nova secgroup-add-rule global_http tcp 443 443 0.0.0.0/0
```

```
+-------------+-----------+---------+-----------+--------------+
| IP Protocol | From Port | To Port | IP Range  | Source Group |
+-------------+-----------+---------+-----------+--------------+
| tcp         | 443       | 443     | 0.0.0.0/0 |              |
+-------------+-----------+---------+-----------+--------------+
```

新しく追加されたルールのみが出力されますが、この操作は追加操作です。

```
$ nova secgroup-list-rules global_http
```

```
+-------------+-----------+---------+-----------+--------------+
| IP Protocol | From Port | To Port | IP Range  | Source Group |
+-------------+-----------+---------+-----------+--------------+
| tcp         | 80        | 80      | 0.0.0.0/0 |              |
| tcp         | 443       | 443     | 0.0.0.0/0 |              |
+-------------+-----------+---------+-----------+--------------+
```

逆の操作が secgroup-delete-rule です。secgroup-delete-rule のコマンドラインは同じ形式です。セキュリティグループ全体を secgroup-delete を用いて削除できます。

インスタンスのクラスター向けにセキュリティグループのルールを作成する方法:

ソースグループは許可するソースの CIDR を動的に定義する特別な方法です。ユーザーがソースグループ（セキュリティグループ名）を指定します。これにより、指定されたソースグループを使用する、ユーザーの他のインスタンスが動的にすべて選択されます。これにより、クラスターのそれぞれの新しいメンバーを許可する、個別のルールが必要なくなります。

使用法: nova secgroup-add-group-rule <secgroup> <source-group> <ip-proto> <from-port> <to-port>

```
$ nova secgroup-add-group-rule cluster global-http tcp 22 22
```

「cluster」ルールにより、「global-http」グループを使用する他のすべてのインスタンスから SSH アクセスが許可されます。

ブロックストレージ

OpenStack のボリュームは、インスタンスから接続および切断できる、永続的なブロックストレージデバイスです。ただし、一度に接続できるのは 1 インスタンスだけです。外部ハードディスクと似ています。ネットワークファイルシステムやオブジェクトストアがしているような共有ストレージは提供されません。ブロックデバイス上にファイルシステムを構築し、それをマウントするかどうかは、インスタンス内のオペレーティングシステムに任されます。

他のリムーバブルディスク技術と同じように、ディスクを取り外す前に、オペレーティングシステムがそのディスクを使用しないようにすることが重要です。Linux インスタンスにおいて、一般的にボリュームからマウントされているすべてのファイルシステムをアンマウントする必要があります。OpenStack Volume Service は、インスタンスから安全にボリュームを取り外すことができるかはわかりません。そのため、指示されたことを実行します。ボリュームに書き込み中にインスタンスからボリュームの切断を、ユーザーが Volume Service に指示すると、何らかのレベルのファイルシステム破損が起きる可能性があります。それだけでなく、デバイスを使用していたインスタンスの中のプロセスがエラーを起こす可能性もあります。

ブロックデバイスにアクセスするために、インスタンスのオペレーティングシステムにおいて必要となる手順に、OpenStack 固有の事項はありません。初めて使用するときにフォーマットが必要になる、デバイスを取り外すときに注意する、などが考えられます。固有の事項は、新しいボリュームを作成し、それらをインスタンスに接続および切断する方法です。これらの操作は、ダッシュボードの「ボリューム」ページからすべて実行できます。または、cinder コマンドラインクライアントを使用します。

新しいボリュームを追加する際に必要なのは、名前とギガバイト単位のボリューム容量だけです。これらを「ボリュームの作成」Web フォームに記入します。または、コマンドラインを使用します。

```
$ cinder create --display-name test-volume 10
```

これは「test-volume」という名前の 10GB のボリュームを作成します。既存のボリュームの一覧を表示するには以下のようにします。それらが接続されているインスタンスがあれば、インスタンス情報も表示されます。

```
$ cinder list
```

```
+-----------+--------+-------------------+------+-------------+-------------+
|     ID    | Status |    Display Name   | Size | Volume Type | Attached to |
+-----------+--------+-------------------+------+-------------+-------------+
| 0821...19f | active |    test-volume    |  10  |     None    |             |
+-----------+--------+-------------------+------+-------------+-------------+
```

Block Storage Service では、ボリュームのスナップショットを作成することもできます。これはブロックレベルのスナップショットであることを覚えておいてください。これはクラッシュに対する一貫性があります。そのため、スナップショットが取得されるとき、ボリュームがインスタンスに接続されていないことが最良です。ボリュームが接続されたインスタンスにおいて使用されていなければ、次に良いです。ボリュームが高負荷にある場合、スナップショットによりファイルシステムの不整合が起こる可能性があります。実際、デフォルト設定では、Volume Service はイメージに接続されたボリュームのスナップショットを取得しません。ただし、強制的に実行することができます。ボリュームのスナップショットを取得するには、ダッシュボードの「ボリューム」ページにおいて、ボリューム名の隣にあるアクション項目から「スナップショットの作成」を選択します。または、コマンドラインから次のようにします。

```
usage: cinder snapshot-create [--force <True|False>]
[--display-name <display-name>]
[--display-description <display-description>]
<volume-id>
Add a new snapshot.
Positional arguments:  <volume-id>           ID of the volume to snapshot
Optional arguments:  --force <True|False>  Optional flag to indicate whether to snapshot a volume
                     even if its attached to an instance. (Default=False)  --display-name <display-
name>                       Optional snapshot name. (Default=None)
--display-description <display-description>
Optional snapshot description. (Default=None)
```

ブロックストレージの作成エラー

ユーザーがボリュームを作成しようとし、すぐにエラー状態になれば、トラブル解決のために最適な方法は Cinder ログファイルをボリュームの UUID で grep することです。まずクラウドコントローラーにあるログファイルを調べます。次に、ボリュームを作成しようとしたストレージノードのログファイルを調べます。

```
# grep 903b85d0-bacc-4855-a261-10843fc2d65b /var/log/cinder/*.log
```

インスタンス

インスタンスは OpenStack クラウドの中で実行中の仮想マシンです。このセクションは、インスタンス、インスタンスが使用するイメージ、インスタンスのネットワークプロパティを扱うための方法について取り扱います。また、それらがデータベースでどのように表現されているかについて取り扱います。

インスタンスの起動

インスタンスを起動するには、イメージ、フレーバーおよび名前を選択する必要があります。名前は一意である必要がありませんが、名前が一意である限りは、多くのツールが UUID の代わりに名前を使用できるので、シンプルにできます。インスタンスの起動はダッシュボードにおいて、「インスタンス」ページにある「インスタンスの起動」ボタン、または「イメージ & スナップショット」ページにあるイメージまたはスナップショットの隣にある「起動」アクションから実行できます。

コマンドラインでは次のようにします。

```
$ nova boot --flavor <flavor> --image <image> <name>
```

指定できる多くのオプション項目があります。インスタンスを起動しようとする前に、このインスタンスのセクションを最後まで読んでください。しかし、これが今から説明する詳細の基本となるコマンドです。

ダッシュボードからインスタンスを削除するには、「インスタンス」ページにおいてインスタンスの隣にある「インスタンスの終了」アクションを選択します。または、コマンドラインから次のとおり実行します。

```
$ nova delete <instance-uuid>
```

注意すべき大事な点は、インスタンスの電源オフは、OpenStack 的な意味でのインスタンスの終了ではないということです。

インスタンスの起動失敗

インスタンスの開始に失敗し、すぐに「エラー」状態になるならば、何が問題なのかを追跡するために、いくつかの異なる方法があります。いくつかの方法は通常のユーザーアクセスで実行でき、他の方法ではログサーバーやコンピュートノードへのアクセスが必要です。

ノードが起動に失敗する最も簡単な理由は、クォータ違反、またはスケジューラーがインスタンスを実行するのに適したコンピュートノードを見つけられなかった場合です。これらの場合、失敗したインスタンスに対して nova show を実行するとエラーが表示されます。

```
$ nova show test-instance
```

```
+-----------------------+-----------------------------------------------------¥
| Property              | Value                                               /
+-----------------------+-----------------------------------------------------¥
| OS-DCF:diskConfig     | MANUAL                                              /
| OS-EXT-STS:power_state | 0                                                  ¥
```

```
| OS-EXT-STS:task_state  | None                                                  /
| OS-EXT-STS:vm_state    | error                                                 ¥
| accessIPv4             |                                                       /
| accessIPv6             |                                                       ¥
| config_drive           |                                                       /
| created                | 2013-03-01T19:28:24Z                                  ¥
| fault                  | {u'message': u'NoValidHost', u'code': 500, u'created/
| flavor                 | xxl.super (11)                                        ¥
| hostId                 |                                                       /
| id                     | 940f3b2f-bd74-45ad-bee7-eb0a7318aa84                  ¥
| image                  | quantal-test (65b4f432-7375-42b6-a9b8-7f654a1e676e) /
| key_name               | None                                                  ¥
| metadata               | {}                                                    /
| name                   | test-instance                                         ¥
| security_groups        | [{u'name': u'default'}]                               /
| status                 | ERROR                                                 ¥
| tenant_id              | 98333a1a28e746fa8c629c83a818ad57                      /
| updated                | 2013-03-01T19:28:26Z                                  ¥
| user_id                | a1ef823458d24a68955fec6f3d390019                      /
+------------------------+-------------------------------------------------------¥
```

この場合、「fault」メッセージに NoValidHost が表示されています。NoValidHost はスケジューラーがインスタンスの要件を満たせなかったことを意味します。

nova show が十分な失敗の理由が表示されていない場合、そのインスタンスがスケジューリングされたコンピュートノードの nova-compute.log やスケジューラーホストの nova-scheduler.log を、インスタンスの UUID で検索するのが、より低レベルの問題を調査する良い出発点となります。

管理ユーザーとして nova show を使用すると、インスタンスがスケジュールされたコンピュートノードが hostId として表示されます。インスタンスがスケジュール中に失敗していれば、この項目が空白です。

インスタンス固有のデータ

カスタムデータを注入する方法は、認証済みキー注入、ユーザーデータ、メタデータサービス、ファイル注入（file injection）などいろいろな方法があります。

ユーザーデータとメタデータの違いを明確にするには、「ユーザーデータ」がデータの塊で、インスタンスが実行されていないときに設定することを理解する必要があります。このユーザーデータは、インスタンスが実行中に、インスタンスの中からアクセスできます。設定、スクリプト、またはテナントが必要とする任意のものを保存するために、このユーザーデータを使用します。

Compute では、インスタンスのメタデータはインスタンスと関連付けられたキーバリューペアの集まりです。エンドユーザーがこれらのキーバリューペアを読み書きするために Compute API を使用するとき、Compute がインスタンスの生存期間中にインスタンスの内外からこれらを読み書きします。しかしながら、Amazon EC2 メタデータサービス

と互換性のあるメタデータサービス経由で、インスタンスに関連付けられたキーバリューペアをクエリーできません。

ユーザーが nova コマンドを使用して SSH 鍵を生成および登録できます。

```
$ nova keypair-add mykey > mykey.pem
```

これにより、インスタンスと関連付けられる mykey という名前の鍵が生成されます。mykey.pem というファイルが秘密鍵です。これは、mykey 鍵が関連付けられたインスタンスに root アクセスできるので、安全な場所に保存すべきです。

このコマンドを使用して、既存の公開鍵を OpenStack に登録できます。

```
$ nova keypair-add --pub-key mykey.pub mykey
```

この鍵と関連付けられたインスタンスにアクセスするために、対応する秘密鍵を持つ必要があります。

起動時にインスタンスに鍵を関連付けるには、たとえば、コマンドラインに --key_name mykey を追加します。

```
$ nova boot --image ubuntu-cloudimage --flavor 1 --key_name mykey
```

サーバーを起動するとき、他の実行中のインスタンスと区別しやすくするために、メタデータを追加することもできます。--meta オプションを key=value ペアとともに使用します。ここで、キーとバリューの両方の文字列を指定することができます。たとえば、説明とサーバーの作成者を追加できます。

```
$ nova boot --image=test-image --flavor=1 smallimage --meta description='Small test image'
```

サーバーの情報を表示するとき、メタデータ行に含まれるメタデータを参照できます。

```
$ nova show smallimage
```

```
+------------------------+----------------------------------------+
|     Property           |                  Value                 |
+------------------------+----------------------------------------+
|   OS-DCF:diskConfig    |               MANUAL                   |
| OS-EXT-STS:power_state |                 1                      |
| OS-EXT-STS:task_state  |                None                    |
|  OS-EXT-STS:vm_state   |               active                   |
|    accessIPv4          |                                        |
|    accessIPv6          |                                        |
|      config_drive      |                                        |
|     created            |            2012-05-16T20:48:23Z        |
|      flavor            |              m1.small                  |
|      hostId            |             de0...487                  |
|        id              |             8ec...f915                 |
|      image             |             natty-image                |
|     key_name           |                                        |
|     metadata           | {u'description': u'Small test image'}  |
```

```
|       name             |             smallimage2                |
|    private network     |            172.16.101.11               |
|     progress           |                 0                      |
|     public network     |             10.4.113.11                |
|      status            |                ACTIVE                  |
|    tenant_id           |             e83...482                  |
|     updated            |            2012-05-16T20:48:35Z        |
|     user_id            |           de3...0a9                    |
+------------------------+----------------------------------------+
```

ユーザーデータはメタデータサービスにおける特別なキーで、ゲストインスタンス内のクラウド対応アプリケーションがアクセスできるファイルを保持します。たとえば、 cloudinit (https://help.ubuntu.com/community/CloudInit) は、このユーザーデータを使用するクラウドインスタンスの初期設定を処理する、Ubuntu のオープンソースパッケージです。

このユーザーデータは、ローカルマシンのファイルに保存され、--user-data <user-data-file> フラグを用いてインスタンスの生成時に渡されます。例えば、

```
$ nova boot --image ubuntu-cloudimage --flavor 1 --user-data mydata.file
```

--file <dst-path=src-path> オプションを用いて、任意のローカルファイルを生成時にインスタンスのファイルシステムの中に置けます。5 ファイルまで保存できます。たとえば、何らかの理由で通常の SSH 鍵の注入ではなく、special_authorized_keysfile という名前の特別な authorized_keys をインスタンスに置きたい場合、以下のコマンドを使用できます。

```
$ nova boot --image ubuntu-cloudimage --flavor 1 --file /root/.ssh/authorized_keys=
special_authorized_keysfile
```

セキュリティグループの割り当て

プロジェクトの規定のセキュリティグループを新たな通信を許可するように変更していない限り、インスタンス向けのネットワーク通信を許可するには、これまでに議論してきたセキュリティグループが必要です。

セキュリティグループの追加は、一般的にインスタンスの起動時に実行されます。ダッシュボードから起動するとき、これは「インスタンスの起動」ダイアログの「アクセス & セキュリティ」タブにあります。コマンドラインから起動する場合には、--security-groups にセキュリティグループのコンマ区切り一覧を指定します。

インスタンスを実行中にセキュリティグループを追加および削除することもできます。現在、コマンドラインツールからのみ利用可能です。

```
$ nova add-secgroup <server> <securitygroup>
```

```
$ nova remove-secgroup <server> <securitygroup>
```

Floating IP

プロジェクトはクォータで管理された数の Floating IP を持ちます。しかしながら、これらは、使用可能になる前にユーザーにより確保する必要があります。Floating IP をプロジェクトに確保するために、ダッシュボードの「アクセス & セキュリティ」ページにある「プロジェクトへの IP の確保」ボタンがあります。または、コマンドラインにおいて次を使用します。

```
$ nova floating-ip-create
```

一度確保すると、Floating IP を実行中のインスタンスに割り当てることができます。ダッシュボードでは、「アクセス & セキュリティ」ページにある IP の隣にある、アクションドロップダウンから「Floating IP の割り当て」を選択することにより実行できます。または、「インスタンス」ページにおいて割り当てたいインスタンスの隣にある、同じアクションから実行できます。逆の動作「Floating IP の解放」は「アクセス & セキュリティ」ページからのみ利用可能です。インスタンスのページから利用できません。

これらの作業を完了するために、コマンドラインの場合、以下のコマンドにより同じことができます。

```
$ nova add-floating-ip <server> <address>
```

```
$ nova remove-floating-ip <server> <address>
```

ブロックストレージの接続

ダッシュボードから ボリューム ページにおいて、ブロックストレージをインスタンスに接続できます。接続したいボリュームの隣にある 接続の編集 アクションをクリックします。

このアクションをコマンドラインから実行するには、以下のコマンドを実行します

```
$ nova volume-attach <server> <volume>
```

nova コマンドラインクライアントから以下のように、インスタンスの起動時にブロックストレージのマッピングを指定することもできます。

```
--block-device-mapping <dev-name=mapping>
```

ブロックデバイスのマッピング形式は <dev-name=<id>:<type>:<size(GB)>:<delete-on-terminate> です。ここで、

dev-name	そのボリュームはシステムで /dev/dev_name に接続されます。
id	起動するボリュームの ID です。nova volume-list の出力に表示されます。
type	ボリュームがスナップショットから作成されたことを意味する snap、または snap 以外の何か（空文字列も有効）です。上の例では、ボリュームがスナップショットから作成されていません。そのため、この項目を以下の例において空白にしてあります。
size (GB)	ボリュームのギガバイト単位の容量。このフィールドを空欄にして、Compute サービスに容量を推定させるのが安全です。
delete-on-terminate	インスタンスが終了したときに、ボリュームが削除されるかどうかを指示する論理値です。真は True または 1 として指定できます。偽は False または 0 として指定できます。

ブート可能なファイルシステムイメージでブロックストレージを事前に準備していると、永続ブロックストレージからブートすることもできます。以下の例は ID=13 のボリュームからブートしようとしています。終了時に削除されません。--key-name を有効なキーペア名で置き換えてください。

```
$ nova boot --flavor 2 --key-name mykey --block-device-mapping vda=13:::0 boot-from-vol-test
```

バグ 1163566 (https://bugs.launchpad.net/nova/+bug/1163566) のため、Horizon においてボリュームからブートするとき、このイメージを使用しないにも関わらず、イメージを指定しなければいけません。

普通にイメージからブートし、ブロックストレージを接続するには、vda 以外のデバイスをマッピングします。

スナップショットの取得

OpenStack のスナップショット機能により、実行中のインスタンスから新しいイメージを作成できます。これは、基本イメージをアップグレードしたり、公開イメージを取得してローカル向けにカスタマイズしたり

する場合に非常に便利です。CLI を利用して、実行中のインスタンスのスナップショットを取得し、イメージを作成する方法:

```
# nova image-create <instance name or uuid> <name of new image>
```

「イメージ & スナップショット」のページでは、内容が以下のように分類されているので、ダッシュボードのスナップショットのインターフェースは紛らわしいです。

- イメージ
- インスタンスのスナップショット
- ボリュームのスナップショット

しかしながら、インスタンスのスナップショットはイメージです。Glance に直接アップロードしたイメージと、スナップショットにより作成したイメージとの唯一の違いは、スナップショットにより作成されたイメージが glance データベースにおいて追加のプロパティを持つことです。これらのプロパティは image_properties テーブルで確認でき、次の項目を含みます。

名前	value
image_type	スナップショット
instance_uuid	<スナップショットされたインスタンスの UUID>
base_image_ref	<スナップショットされたインスタンスの元イメージの UUID>
image_location	スナップショット

ライブスナップショット

Live snapshots is a feature that allows users to snapshot the running virtual machines without pausing them. These snapshots are simply disk-only snapshots. Snapshotting an instance can now be performed with no downtime (assuming QEMU 1.3+ and libvirt 1.0+ are used).

スナップショットの一貫性の保証

Sébastien Han さんの OpenStack: Perform Consistent Snapshots ブログエントリー (http://www.sebastien-han.fr/blog/2012/12/10/openstack-perform-consistent-snapshots/) からのコンテンツです。

スナップショットは、ファイルシステムの状態をキャプチャーしますが、メモリーの状態をキャプチャーしません。そのため、スナップショットに期待するデータが含まれることを確実にするために、次のことを確実にする必要があります。

- 実行中のプログラムがコンテンツをディスクに書き込んだこと
- ファイルシステムが「ダーティー」バッファーを持たないこと: 「ダーティー」バッファーがあるとは、プログラムがディスクに書き込むためにコマンドを発行しましたが、オペレーティングシステムがまだ書き込みを完了していないことです。

(データベースのような) 重要なサービスがコンテンツをディスクに書き込んだことを保証するために、それらのアプリケーションのドキュメントを読んで、コンテンツをディスクに同期させるためにどのコマンドを発行する必要があるかを調べることを推奨します。ディスクに同期させるための方法がはっきり分からない場合、最も安全な方法は単にこれらの実行中のサービスを通常通り停止することです。

「ダーティー」バッファーの問題を解決するために、スナップショットの前に sync コマンドを使用することを推奨します。

```
# sync
```

sync を実行することにより、ダーティーバッファー (変更されたが、ディスクに書き込まれていないバッファー済みブロック) をディスクに書き込みます。

ファイルシステムが一貫性を持つことを保証するためには、単に sync を実行するだけでは不十分です。fsfreeze ツールを使用することを推奨します。これは、ファイルシステムに対する新規アクセスを停止し、スナップショットに適した安定したイメージをディスクに作成します。fsfreeze は ext3, ext4 および XFS を含むいくつかのファイルシステムをサポートします。仮想マシンのインスタンスが Ubuntu において実行されていれば、fsfreeze を取得するために util-linux パッケージをインストールします。

```
# apt-get install util-linux
```

お使いのオペレーティングシステムに利用可能なバージョンの fsfreeze がなければ、代わりに xfs_freeze を使用できます。これは Ubuntu の xfsprogs パッケージにおいて利用可能です。「xfs」という名前にもかかわらず、xfs_freeze は Linux カーネル 2.6.29 またはそれ以降を使用していれば ext3 や ext4 においても動作します。それは 2.6.29 において開始された仮想ファイルシステム (VFS) レベルで動作するためです。xfs_freeze は fsfreeze と同じ名前のコマンドライン引数をサポートします。

永続ブロックストレージのスナップショットを取得したい例を検討します。ゲストオペレーティングシステムにより /dev/vdb として認識され、/mnt にマウントされているとします。fsfreeze コマンドが 2 つの引数を受け取ります。

- -f: システムをフリーズします
- -u: システムを解凍（フリーズ解除）します

スナップショットの準備においてボリュームをフリーズするには、インスタンスの中で root として次のとおり実行します。

```
# fsfreeze -f /mnt
```

fsfreeze コマンドを実行する前に、ファイルシステムをマウントする必要があります。

「fsfreeze -f」コマンドが発行された場合、ファイルシステム内で進行中のすべてのトランザクションが完了することが認められます。新規書き込みのシステムコールは停止されます。そして、ファイルシステムを変更する他のコールは停止されます。最も重要なこととしては、すべてのダーティーデータ、メタデータ、およびログ情報がディスクに書き込まれることです。

ボリュームがフリーズ状態になったら、ボリュームの読み書き命令が止まってしまうので、ボリュームの読み書きを行わないようにしてください。オペレーティングシステムがすべての I/O 操作を停止し、すべての I/O 試行がファイルシステムがフリーズ解除されるまで遅延させられます。

fsfreeze コマンドを発行すると、スナップショットを実行しても安全です。たとえば、インスタンスが mon-instance という名前で、mon-snapshot という名前のイメージにスナップショットを取得したければ、以下のとおり実行します。

```
$ nova image-create mon-instance mon-snapshot
```

スナップショットの作成が終わったら、インスタンスの中で root として以下のコマンドを用いて、ファイルシステムをフリーズ解除できます。

```
# fsfreeze -u /mnt
```

ルートファイルシステムをバックアップしたければ、プロンプトがフリーズしてしますので、上のコマンドを単純に実行できません。代わりに、インスタンスの中で root として以下の 1 行を実行します。

```
# fsfreeze -f / && sleep 30 && fsfreeze -u /
```

データベースにあるインスタンス

インスタンス情報は多くのデータベースのテーブルに保存されますが、ユーザーのインスタンスに関連して運用者が参照する必要が最もありそうなテーブルは「instances」テーブルです。

インスタンスのテーブルは、実行中および削除済みの両方のインスタンスに関連する情報のほとんどを保持しています。データベースで完全なリストを見ると、このテーブルには目が回るほどたくさんのフィールドがあることがわかります。以下に、クエリーを行おうとしている運用者にとって非常に有用なフィールドを挙げます。

「deleted」フィールドは、インスタンスが削除されていると「1」がセットされます。削除されていなければ NULL です。これはクエリーから削除済みインスタンスを除外するために重要です。

「uuid」フィールドはインスタンスの UUID です。データベースにある他の表において外部キーとして使用されます。この ID は、インスタンスを一意に識別するために、ログ、ダッシュボードおよびコマンドラインツールにおいて表示されます。

外部キーはインスタンスの関連を見つけるために利用可能です。これらの中で最も有用なものは、「user_id」および「project_id」です。これらは、インスタンスを起動したユーザー、およびそれが起動されたプロジェクトの UUID です。

「host」フィールドは、どのコンピュートノードがインスタンスをホストしているかを示します。

「hostname」フィールドは、インスタンスが起動したときのインスタンス名を保持します。「display-name」は、最初は hostname と同じですが、nova rename コマンドを使って再設定することができます。

多くの時刻関連のフィールドは、いつ状態の変化がインスタンスに起こったかを追跡する際に役に立ちます。

- created_at
- updated_at
- deleted_at
- scheduled_at
- launched_at

- terminated_at

第11章 メンテナンス、故障およびデバッグ

クラウドコントローラーとストレージプロキシの故障とメンテナンス 119
コンピュートノードの故障とメンテナンス 121
ストレージノードの故障とメンテナンス 127
完全な故障の対処 129
構成管理 130
ハードウェアの取り扱い 130
データベース 132
HDWMY 133
故障しているコンポーネントの特定 134
アップグレード 136
アンインストール 137

停止時間（計画的なものと予定外のものの両方）はクラウドを運用するときに確実に発生します。本章は、プロアクティブまたはリアクティブに、これらの出来事に対処するために有用な情報を提供することを目的としています。

クラウドコントローラーとストレージプロキシの故障とメンテナンス

想定内の場合も想定外の場合も停止時間が発生した場合の挙動が、クラウドコントローラーとストレージプロキシは互いに似ています。クラウドコントローラーとストレージプロキシはそれぞれクラウドで一つ実行されるので、動作していない場合、非常に目立ちます。

クラウドコントローラーの場合、良いニュースとしては、クラウドがFlatDHCP マルチホスト HA ネットワークモードを使用していれば、既存のインスタンスとボリュームはクラウドコントローラーがオフラインの間も動作を継続するという点があります。しかしながら、ストレージプロキシの場合には、サーバーが元に戻され動作状態になるまで、ストレージとの通信ができません。

計画メンテナンス

クラウドコントローラーやストレージプロキシのメンテナンスを計画する一つの方法は、単に午前 1 時や 2 時のような利用の少ない時間帯に

実行することです。この戦略はあまり多くのユーザーに影響を与えません。クラウドコントローラーやストレージプロキシが、いかなる時間帯においても、サービスが利用できないことによる影響が大きければ、高可用性オプションについて検討する必要があります。

クラウドコントローラーとストレージプロキシの再起動

多くの場合、「reboot」コマンドを発行するだけです。オペレーティングシステムが正常にサービスをシャットダウンし、自動的に再起動します。万全を期したい場合、再起動する前にバックアップジョブを実行してください。

クラウドコントローラーまたはストレージプロキシの再起動後

クラウドコントローラーを再起動した後、すべての必要なサービスが正常に起動したことを確認します。

```
# ps aux | grep nova-
# grep AMQP /var/log/nova/nova-*.log
# ps aux | grep glance-
# ps aux | grep keystone
# ps aux | grep cinder
```

また、すべてのサービスが正しく機能していることを確認します。

```
# source openrc
# glance index
# nova list
# keystone tenant-list
```

ストレージプロキシの場合、Object Storage サービスが再開していることを確認します。

```
# ps aux | grep swift
```

また、正しく機能していることを確認します。

```
# swift stat
```

全体的なクラウドコントローラーの故障

残念ながら、これはひどい状況です。クラウドコントローラーはクラウドの不可欠な部分です。コントローラーが一つだけならば、多くのサービスが失われます。

この状況を避けるために、高可用なクラウドコントローラークラスターを作成します。このことは、このドキュメントの範囲外です

が、ドラフト版の OpenStack High Availability Guide が http://docs.openstack.org/trunk/openstack-ha/content/ch-intro.html にあります。

次に最も優れている方法は、クラウドコントローラーを自動的に構築するために Puppet のような構成管理ツールを使用することです。利用可能な予備サーバーがあれば、15 分もかかりません。コントローラーを再構築後、取得したすべてのバックアップを復元します（バックアップとリカバリー の章を参照してください)。

実際には、コンピュートノードの nova-compute サービスがときどき、コントローラー上で動作している rabbitmq に正しく再接続されない場合があります。時間のかかるリブートから戻ってきた場合や、コンピュートノードのnova サービスを再起動する必要がある場合です。

コンピュートノードの故障とメンテナンス

コンピュートノードは、予期せずクラッシュしたり、メンテナンスのために再起動が必要になったりすることがときどきあります。

計画メンテナンス

（ソフトウェアやハードウェアのアップグレードのように）計画されたメンテナンスのために、コンピュートノードを再起動する必要があれば、まずホストしている全インスタンスがノード外に移動していることを確認します。クラウドが共有ストレージを利用していれば、nova live-migration コマンドを使用します。初めに、移動させる必要があるインスタンスの一覧を取得します。

```
# nova list --host c01.example.com --all-tenants
```

次に、それらを一つずつマイグレーションします。

```
# nova live-migration <uuid> c02.example.com
```

共有ストレージを使用していない場合、--block-migrate オプションを使用できます。

```
# nova live-migration --block-migrate <uuid> c02.example.com
```

すべてのインスタンスをマイグレーションした後、nova-compute サービスが停止していることを確認します。

```
# stop nova-compute
```

Puppet などの構成管理システムを使って、nova-compute サービスが確実に実行されているようにしている場合、init ファイルを一時的に移動します。

```
# mkdir /root/tmp
# mv /etc/init/nova-compute.conf /root/tmp
# mv /etc/init.d/nova-compute /root/tmp
```

続けて、コンピュートノードを停止し、メンテナンスを実行し、ノードを元に戻します。先のコマンドを逆に実行することにより、nova-compute サービスを再び有効化できます。

```
# mv /root/tmp/nova-compute.conf /etc/init
# mv /root/tmp/nova-compute /etc/init.d/
```

そして nova-compute サービスを起動します。

```
# start nova-compute
```

インスタンスを元のコンピュートノードにマイグレーションすることもできます。

コンピュートノードの再起動後

コンピュートノードを再起動した場合、まず正常に起動したことを確認します。これには nova-compute サービスが実行していることを確認することが含まれます。

```
# ps aux | grep nova-compute
# status nova-compute
```

AMQP サーバーに正常に接続できることも確認します。

```
# grep AMQP /var/log/nova/nova-compute
2013-02-26 09:51:31 12427 INFO nova.openstack.common.rpc.common [-] Connected to AMQP server on 199.
116.232.36:5672
```

コンピュートノードが正常に実行された後、そのコンピュートノードでホストされているインスタンスはどれも動作していないので、そのコンピュートノードにおいてホストされているインスタンスを処理する必要があります。ユーザーや顧客に対する SLA によっては、各インスタンスを開始し、正常に起動していることを確認する必要がある場合もあるでしょう。

インスタンス

以下のコマンドを実行することにより、コンピュートノードにおいてホストされているインスタンスの一覧を作成できます。

```
# nova list --host c01.example.com --all-tenants
```

一覧を取得した後、各インスタンスを起動するために nova コマンドを使用できます。

```
# nova reboot <uuid>
```

注記

予期せずシャットダウンしたときは、ブートに問題があるかもしれません。たとえば、インスタンスがルートパーティションにおいて fsck を実行する必要があるかもしれません。もしこうなっても、これを修復するためにダッシュボード VNC コンソールを使用できます。

インスタンスがブートしなければ、つまりブートしようとしても virsh list がインスタンスを表示しなければ、コンピュートノードにおいて以下のとおり実行します。

```
# tail -f /var/log/nova/nova-compute.log
```

再び nova reboot コマンドを実行してみてください。インスタンスがなぜブートできないかについて、エラーメッセージを確認すべきです。

多くの場合、libvirt の XML ファイル（/etc/libvirt/qemu/instance-xxxxxxxx.xml）の何かがすでに存在しないことで、エラーが発生する。次のとおり実行することにより、インスタンスを再起動するのと同時に、強制的に XML ファイルを再作成できます。

```
# nova reboot --hard <uuid>
```

故障したインスタンスからの検査とデータ復旧

いくつかのシナリオでは、インスタンスが実行中であるにも関わらず、SSH 経由でアクセスできず、あらゆるコマンドに反応がありません。VNC コンソールがブート失敗やカーネルパニックのエラーメッセージを表示している可能性があります。これは仮想マシン自身においてファイルシステム破損の意味する可能性があります。ファイルを復旧したりインスタンスの中身を調査したりする必要があれば、qemu-nbd を使ってディスクをマウントできます。

注記

ユーザーのコンテンツやデータにアクセスしたり表示したりする場合は、まず承認をもらってください!

インスタンスのディスク（/var/lib/nova/instances/instance-xxxxxx/disk）にアクセスするには、以下の手順に従う必要があります。

1. virsh コマンドを使用してインスタンスをサスペンドします。

2. qemu-nbd デバイスをディスクに接続します。

3. qemu-nbd デバイスをマウントします。

4. デバイスを調査後、アンマウントします。

5. qemu-nbd デバイスを切断します。

6. インスタンスを再開します。

手順 4-6 を省略すると、OpenStack Compute がインスタンスを管理できなくなります。OpenStack Compute により発行されるすべてのコマンドに対する応答が失敗し、シャットダウンしているように見えます。

ディスクファイルをマウントすると、それにアクセスでき、ファイルとディレクトリ構造を持つ通常のディレクトリのように取り扱えます。しかしながら、どのファイルの編集も操作もしないことをお薦めします。なぜなら、それにより ACL が変更されたり、起動できるインスタンスが起動できなくなってしまう場合があるからです。

1. virsh コマンドを使用してインスタンスを一時停止します - 内部 ID を記録します。

```
root@compute-node:~# virsh list
Id Name                 State
----------------------------------
1 instance-00000981    running
2 instance-000009f5    running
30 instance-0000274a    running

root@compute-node:~# virsh suspend 30
Domain 30 suspended
```

2. qemu-nbd デバイスをディスクに接続します。

```
root@compute-node:/var/lib/nova/instances/instance-0000274a# ls -lh
total 33M
-rw-rw---- 1 libvirt-qemu kvm  6.3K Oct 15 11:31 console.log
-rw-r--r-- 1 libvirt-qemu kvm   33M Oct 15 22:06 disk
-rw-r--r-- 1 libvirt-qemu kvm  384K Oct 15 22:06 disk.local
-rw-rw-r-- 1 nova         nova 1.7K Oct 15 11:30 libvirt.xml
root@compute-node:/var/lib/nova/instances/instance-0000274a# qemu-nbd -c /dev/nbd0 `pwd`/disk
```

3. qemu-nbd デバイスをマウントします。

   qemu-nbd デバイスはインスタンスのディスクの個々のパーティションを別々のデバイスとしてエクスポートしようとします。たとえば、ディスクが vda で、ルートパーティションが vda1 の場合、qemu-nbd はそれぞれ /dev/nbd0 と /dev/nbd0p1 としてデバイスをエクスポートします。

```
#mount the root partition of the device
root@compute-node:/var/lib/nova/instances/instance-0000274a# mount /dev/nbd0p1 /mnt/
# List the directories of mnt, and the vm's folder is display
# You can inspect the folders and access the /var/log/ files
```

セカンダリディスクや一時ディスクを調査する際に、プライマリディスクとセカンダリディスクを同時にマウントしたければ、別のマウントポイントを使用してください。

```
# umount /mnt
# qemu-nbd -c /dev/nbd1 `pwd`/disk.local
# mount /dev/nbd1 /mnt/
```

```
root@compute-node:/var/lib/nova/instances/instance-0000274a# ls -lh /mnt/
total 76K
lrwxrwxrwx.  1 root root    7 Oct 15 00:44 bin -> usr/bin
dr-xr-xr-x.  4 root root 4.0K Oct 15 01:07 boot
drwxr-xr-x.  2 root root 4.0K Oct 15 00:42 dev
drwxr-xr-x. 70 root root 4.0K Oct 15 11:31 etc
drwxr-xr-x.  3 root root 4.0K Oct 15 01:07 home
lrwxrwxrwx.  1 root root    7 Oct 15 00:44 lib -> usr/lib
lrwxrwxrwx.  1 root root    9 Oct 15 00:44 lib64 -> usr/lib64
drwx------.  2 root root  16K Oct 15 00:42 lost+found
drwxr-xr-x.  2 root root 4.0K Feb  3  2012 media
drwxr-xr-x.  2 root root 4.0K Feb  3  2012 mnt
drwxr-xr-x.  2 root root 4.0K Feb  3  2012 opt
drwxr-xr-x.  2 root root 4.0K Oct 15 00:42 proc
dr-xr-x---.  3 root root 4.0K Oct 15 21:56 root
drwxr-xr-x. 14 root root 4.0K Oct 15 01:07 run
lrwxrwxrwx.  1 root root    8 Oct 15 00:44 sbin -> usr/sbin
drwxr-xr-x.  2 root root 4.0K Feb  3  2012 srv
drwxr-xr-x.  2 root root 4.0K Oct 15 00:42 sys
drwxrwxrwt.  9 root root 4.0K Oct 15 16:29 tmp
drwxr-xr-x. 13 root root 4.0K Oct 15 00:44 usr
drwxr-xr-x. 17 root root 4.0K Oct 15 00:44 var
```

4. 調査を完了すると、マウントポイントをアンマウントし、qemu-nbd デバイスを解放します。

```
root@compute-node:/var/lib/nova/instances/instance-0000274a# umount /mnt
root@compute-node:/var/lib/nova/instances/instance-0000274a# qemu-nbd -d /dev/nbd0
/dev/nbd0 disconnected
```

5. virsh を使用してインスタンスを再開します。

```
root@compute-node:/var/lib/nova/instances/instance-0000274a# virsh list
Id Name                 State
----------------------------------
1 instance-00000981    running
2 instance-000009f5    running
30 instance-0000274a    paused

root@compute-node:/var/lib/nova/instances/instance-0000274a# virsh resume 30
Domain 30 resumed
```

ボリューム

影響のあったインスタンスがボリュームを接続していれば、まずインスタンスとボリュームの UUID 一覧を生成します。

```
mysql> select nova.instances.uuid as instance_uuid, cinder.volumes.id as volume_uuid, cinder.
volumes.status,
cinder.volumes.attach_status, cinder.volumes.mountpoint, cinder.volumes.display_name from cinder.
volumes
inner join nova.instances on cinder.volumes.instance_uuid=nova.instances.uuid
 where nova.instances.host = 'c01.example.com';
```

以下のような結果を確認できます。

```
+--------------+-------------+------+--------------+-----------+--------------+
|instance_uuid |volume_uuid |status |attach_status |mountpoint | display_name |
+--------------+-------------+------+--------------+-----------+--------------+
|9b969a05      |1f0fbf36    |in-use |attached      |/dev/vdc   | test         |
+--------------+-------------+------+--------------+-----------+--------------+
1 row in set (0.00 sec)
```

次に、ボリュームを手動で切断し、再接続します。

```
# nova volume-detach <instance_uuid> <volume_uuid>
# nova volume-attach <instance_uuid> <volume_uuid> /dev/vdX
```

ここで、X には適切なマウントポイントを指定します。上記を実行する前に、インスタンスが正常に起動し、ログイン画面になっていることを確認します。

コンピュートノード全体の故障

コンピュートノードが故障し、2〜3時間もしくはそれ以上たっても復旧できないと見込まれる場合、/var/lib/nova/instances に共有ストレージを使用していれば、故障したノードで動作していたインスタンスをすべて再スタートすることができます。

これを実行するために、nova データベースにおいて以下のクエリーを実行することにより、故障したノードにおいてホストされているインスタンスの UUID の一覧を生成します。

```
mysql> select uuid from instances where host = 'c01.example.com' and deleted = 0;
```

次に、c01.example.com においてホストされていたすべてのインスタンスが、今度は c02.example.com でホストされることを Nova に教えます。

```
mysql> update instances set host = 'c02.example.com' where host = 'c01.example.com' and deleted = 0;
```

その後、nova コマンドを使って、c01.example.com にあったすべてのインスタンスを再起動します。起動する際にインスタンスの XML ファイルを再生成します。

```
# nova reboot --hard <uuid>
```

最後に、ボリュームの節で説明されているのと同じ方法を用いて、ボリュームを再接続します。

/var/lib/nova/instances

コンピュートノードの故障の話題に関連して、このディレクトリについては説明しておく価値があるでしょう。このディレクトリには、コンピュートノードにホストされているインスタンス用の libvirt KVM の

ファイル形式のディスクイメージが置かれます。共有ストレージ環境でクラウドを実行していなければ、このディレクトリはコンピュートノード全体で一つしかありません。

/var/lib/nova/instances には 2 種類のディレクトリがあります。

一つ目は _base ディレクトリです。ここには、そのコンピュートノードで起動されたそれぞれのイメージに関して、glance から取得したすべてのベースイメージのキャッシュが置かれます。_20（または他の番号）で終わるファイルは一時ディスクのベースイメージです。

もう一つのディレクトリは instance-xxxxxxxx という名前です。これらのディレクトリはコンピュートノードにおいて実行中のインスタンスと対応します。中にあるファイルは _base ディレクトリにあるファイルのどれかと関連があります。これらは基本的に、元々の _base ディレクトリからの変更点のみ含む、差分ベースのファイルです。

/var/lib/nova/instances にあるすべてのファイルとディレクトリは一意に名前が付けられています。_base にあるファイルは元となった glance イメージに対応する一意に名前が付けられています。また、instance-xxxxxxxx という名前が付けられたディレクトリは特定のインスタンスに対して一意にタイトルが付けられています。たとえば、あるコンピュートノードにある /var/lib/nova/instances のすべてのデータを他のノードにコピーしたとしても、ファイルを上書きすることはありませんし、また同じ一意な名前を持つイメージにダメージを与えることもありません。同じ一意な名前を持つものは本質的に同じファイルだからです。

この方法はドキュメントに書かれておらず、サポートされていない方法ですが、コンピュートノードが完全にオフラインになってしまったが、インスタンスがローカルに保存されているときに、この方法を使用できます。

ストレージノードの故障とメンテナンス

オブジェクトストレージの高い冗長性のため、オブジェクトストレージのノードに関する問題を処理することは、コンピュートノードに関する問題を処理するよりも簡単です。

ストレージノードの再起動

ストレージノードの再起動が必要ならば、単に再起動します。そのノードにホストされているデータに対する要求は、サーバーが再起動している間、他のコピーに転送されます。

ストレージノードのシャットダウン

ストレージノードを少し長い間（1 日以上）シャットダウンする必要があれば、ノードをストレージリングから削除することを検討します。例:

```
# swift-ring-builder account.builder remove <ip address of storage node>
# swift-ring-builder container.builder remove <ip address of storage node>
# swift-ring-builder object.builder remove <ip address of storage node>
# swift-ring-builder account.builder rebalance
# swift-ring-builder container.builder rebalance
# swift-ring-builder object.builder rebalance
```

次に、ring ファイルを他のノードに再配布します。

```
# for i in s01.example.com s02.example.com s03.example.com
> do
> scp *.ring.gz $i:/etc/swift
> done
```

これらの操作はストレージノードをストレージクラスターから効率的に外せます。

ノードがクラスターに参加できるようになったら、ただリングに再度追加するだけです。swift-ring-builder を使用して Swift クラスターにノードを追加するための構文は、元々クラスターを作成したときに使用した元々のオプションに強く依存します。作成時に使用したコマンドをもう一度見てください。

Swift ディスクの交換

Object Storage ノードのハードディスクが故障した場合、その交換は比較的簡単です。Object Storage 環境が正しく設定され、故障したディスクに保存されているデータが Object Storage 環境内の他のディスクにも複製されていることを前提にしています。

この例では、/dev/sdb が故障したと仮定します。

まず、ディスクをアンマウントします。

```
# umount /dev/sdb
```

次に、ディスクを物理的にサーバーから取り外し、正常なディスクと入れ替えます。

オペレーティングシステムが新しいディスクを認識していることを確認します。

```
# dmesg | tail
```

/dev/sdb に関するメッセージを確認したほうがいいです。

Swift ディスクではパーティションを使用しないことが推奨されるので、単にディスク全体をフォーマットします。

```
# mkfs.xfs /dev/sdb
```

最後に、ディスクをマウントします。

```
# mount -a
```

Swift は新しいディスクを認識します。また、データが存在しないことを認識します。そうすると、他の既存の複製からディスクにデータを複製しはじめます。

完全な故障の対処

データセンターの電力消失のような、完全なシステム故障から復旧する一般的な方法は、各サービスに優先度を付け、順番に復旧することです。

表11.1 サービス復旧優先度一覧の例

1	内部ネットワーク接続性
2	バックエンドのストレージサービス
3	ユーザーの仮想マシンに対するパブリックネットワーク接続性
4	nova-compute, nova-network, cinder ホスト
5	ユーザーの仮想マシン
10	メッセージキューとデータベースのサービス
15	Keystone サービス
20	cinder-scheduler
21	イメージカタログとイメージ配信のサービス
22	nova-scheduler サービス
98	cinder-api
99	nova-api サービス
100	ダッシュボードサービス

この例にある優先度一覧を使用すると、きちんと安定した状態になる前であっても、できる限り早くユーザーに影響するサービスを復旧させることができます。もちろん、1 行の項目として一覧化されていますが、各ステップは多大な作業が必要です。たとえば、データベースを開始した後、その完全性を確認すべきです。また、Nova サービスを開始した後、ハイパーバイザーがデータベースに一致しているかを確認し、不一致があれば修正すべきです。

構成管理

OpenStack クラウドをメンテナンスするには、複数の物理サーバーを管理することが必要です。そして、この数は日々増えていきます。ノードを手動で管理することはエラーを起こしやすいので、構成管理ツールを使用することを強く推奨します。これらのツールはすべてのノードが適切に設定されていることを保証するプロセスを自動化します。また、これらを使うことで、(パッケージや設定オプションといった) 構成情報のバージョン管理されたリポジトリでの管理が行いやすくなります。

いくつかの構成管理ツールがあります。このガイドでは特定のものを推奨しません。OpenStack コミュニティで人気があるものは Puppet (https://puppetlabs.com/) と Chef (http://opscode.com/chef) の 2 つで、OpenStack 用の設定集がそれぞれ OpenStack Puppet modules (http://github.com/puppetlabs/puppetlabs-openstack) と OpenStack Chef recipes (https://github.com/opscode/openstack-chef-repo) にあります。比較的新しい他のツールとしては、Juju (https://juju.ubuntu.com/)、Ansible (http://ansible.cc) や Salt (http://saltstack.com) があります。もう少し成熟したツールとしては CFEngine (http://cfengine.com) や Bcfg2 (http://bcfg2.org) があります。

ハードウェアの取り扱い

初期導入時と同じように、本番環境に追加する前に、すべてのハードウェアについて適切な通電テストを行うべきでしょう。ハードウェアを限界まで使用するソフトウェアを実行します。RAM、CPU、ディスク、ネットワークを限界まで使用します。多くのオプションが利用可能であり、通常はベンチマークソフトウェアとの役割も果たします。そのため、システムのパフォーマンスに関する良いアイディアを得ることもできます。

コンピュートノードの追加

コンピューティングリソースのキャパシティ限界に達した、または達しそうとわかれば、さらなるコンピュートノードの追加を計画すべきです。さらなるノードを追加することは簡単です。ノードを追加する手順は、最初にコンピュートノードをクラウドに導入したときと同じです。自動配備システムを使ってベアメタルサーバーにオペレーティングシステムのインストールと起動を行い、次に構成管理システムにより OpenStack Compute サービスのインストールと設定を行います。他のコンピュートノードと同じ方法で Compute サービスのインストールと設

定が終わると、自動的にクラウドに接続されます。クラウドコントローラーが新しいノードを検知し、そこにインスタンスを起動するようスケジュールし始めます。

OpenStack ブロックストレージノードがコンピュートノードから分離いる場合、同じキュー管理とポーリングのシステムが両方のサービスで使用されるので、同じ手順が適用できます。

新しいコンピュートノードとブロックストレージノードには、同じハードウェアを使用することを推奨します。最低限、ライブマイグレーションが失敗しないように、コンピュートノードでは CPU は同様のものにしてください。

オブジェクトストレージノードの追加

新しいオブジェクトストレージノードの追加は、コンピュートノードやブロックストレージノードの追加とは異なります。サーバーの設定は、これまで通り自動配備システムと構成管理システムを使って行えます。完了した後、オブジェクトストレージノードのローカルディスクをオブジェクトストレージリングに追加する必要があります。これを実行するコマンドは、最初にディスクをリングに追加するのに使用したコマンドと全く同じです。オブジェクトストレージプロキシサーバーにおいて、このコマンドを、新しいオブジェクトストレージノードにあるすべてのディスクに対して、再実行するだけです。これが終わったら、リングの再バランスを行い、更新されたリングファイルを他のストレージノードにコピーします。

注記

新しいオブジェクトストレージノードのディスク数が元々のノードのディスク数と異なる場合には、新しいノードを追加するコマンドが元々のコマンドと異なります。これらのパラメーターは環境により異なります。

コンポーネントの交換

クラウドインフラなどの大規模環境では、ハードウェアの故障はよくあることです。作業内容を考慮し、可用性と時間の節約のバランスを取ります。たとえば、オブジェクトストレージクラスターは、十分な容量がある場合には、ある程度の期間は死んだディスクがあっても問題なく動作します。また、（クラウド内の）コンピュートノードに空きがある場合には、問題に対処する時間が取れるまで、ライブマイグレーションで RAM が故障したホストから他のホストへインスタンスを移動させることも考慮するとよいでしょう。

データベース

ほとんどすべての OpenStack コンポーネントは、永続的な情報を保存するために内部でデータベースを使用しています。このデータベースは通常 MySQL です。通常の MySQL の管理方法がこれらのデータベースに適用できます。OpenStack は特別な方法でデータベースを設定しているわけではありません。基本的な管理として、パフォーマンス調整、高可用性、バックアップ、リカバリーおよび修理などがあります。さらなる情報は標準の MySQL 管理ガイドを参照してください。

より迅速に情報を取得したり、データ不整合のエラーを修正したりするために、データベースでいくつかの小技を実行できます。たとえば、インスタンスが終了していたが、データベースの状態が更新されていなかった、という状況です。こうした小技がこのドキュメント全体を通して議論されています。

データベース接続性

コンポーネントの設定ファイルを確認して、それぞれの OpenStack コンポーネントが対応するデータベースにどのようにアクセスするかを把握ください。sql_connection またはただの connection を探します。

```
# grep -hE "connection ?=" /etc/nova/nova.conf /etc/glance/glance-*.conf
/etc/cinder/cinder.conf /etc/keystone/keystone.conf
        sql_connection = mysql://nova:nova@cloud.alberta.sandbox.cybera.ca/nova
        sql_connection = mysql://glance:password@cloud.example.com/glance
        sql_connection = mysql://glance:password@cloud.example.com/glance
    sql_connection=mysql://cinder:password@cloud.example.com/cinder
        connection = mysql://keystone_admin:password@cloud.example.com/keystone
```

connection 文字列は以下の形式をとります。

```
mysql:// <username> : <password> @ <hostname> / <database name>
```

パフォーマンスと最適化

クラウドが大きくなるにつれて、MySQL がさらに使用されてきます。MySQL がボトルネックになってきたことが疑われる場合、MySQL 最適化の調査から始めるとよいでしょう。MySQL のマニュアルでは、Optimization Overview (http://dev.mysql.com/doc/refman/5.5/en/optimize-overview.html) というセクションがあり、一つのセクション全部をあててこの話題を扱っています。

HDWMY

これらは、毎時間、日、週、月および年に実行する To Do 項目の簡単な一覧です。これらのタスクは必要なものでも、絶対的なものでもありませんが、役に立つものばかりです。

毎時

- 監視システムのアラートを確認し、それらに対処します。
- チケットキューの新しいチケットを確認します。

日次

- 故障または異常になっているインスタンスを確認し、理由を調査します。
- セキュリティパッチを確認し、必要に応じて適用します。

週次

- クラウドの使用量を確認します:
  - ユーザークォータ
  - ディスク領域
  - イメージ使用量
  - 大きなインスタンス
  - ネットワーク使用量（帯域および IP 使用量）
- アラート機能が動作していることを確認します。

月次

- この 1 か月における使用量および傾向を確認します。
- 削除すべきユーザーアカウントを確認します。
- 削除すべきオペレーターアカウントを確認します。

四半期ごと

- この四半期における使用量および傾向を確認します。
- 使用量と統計に関する四半期レポートを準備します。
- クラウドの追加の必要性を検討し、計画を立てます。
- OpenStack のメジャーアップグレードの内容を確認し、その計画を立てます。

半年ごと

- OpenStack をアップグレードします。
- OpenStack のアップグレード後に後始末を行います（未使用または新しいサービスを把握していますか?）

故障しているコンポーネントの特定

OpenStack は、異なるコンポーネント同士が互いに強く連携して動作しています。たとえば、イメージのアップロードでは、nova-api, glance-api, glance-registry, Keystone が連携する必要があります。 swift-proxy も関係する場合があります。その結果、時として問題が発生している箇所を正確に特定することが難しくなります。これを支援することがこのセクションの目的です。

最新ログの確認

最初に確認する場所は、実行しようとしているコマンドに関連するログファイルです。たとえば、nova list が失敗していれば、Nova ログファイルを tail 表示しながら、次のコマンドを再実行してください。

端末 1:

```
# tail -f /var/log/nova/nova-api.log
```

端末 2:

```
# nova list
```

何らかのエラーまたはトレースをログファイルで探します。詳細は ロギングとモニタリング の章を参照してください。

エラーから問題が他のコンポーネントにあることが分かる場合には、そのコンポーネントのログファイルに表示を切り替えます。nova が glance にアクセスできなければ、glance-api ログを確認します。

端末 1:

```
# tail -f /var/log/glance/api.log
```

端末 2:

```
# nova list
```

問題の根本となる原因を見つけるまで、洗い出し、精査し、繰り返します。

コマンドラインでのデーモンの実行

残念ながら、ときどきエラーがログファイルに表れない場合があります。このような場合、作戦を変更し、違うコマンドを使用します。おそらくコマンドラインにおいて直接サービスを実行することです。たとえば、glance-api サービスが起動しなかったり、実行状態にとどまらない場合は、コマンドラインからデーモンを起動してみます。

```
# sudo -u glance -H glance-api
```

これにより、エラーと問題の原因が表示されるかもしれません。

注記

sudo を用いてデーモンを実行するとき、-H フラグが必要です。いくつかのデーモンは、ユーザーのホームディレクトリーからの相対パスのファイルに書き込みを行うため、-H がないと、この書き込みが失敗してしまいます。

複雑な例

ある朝、あるノードでインスタンスの実行がすべて失敗するようになりました。ログファイルがすこしあいまいでした。特定のインスタンスが起動できなかったことを示していました。これは最終的に偽の手掛かりであることがわかりました。単にそのインスタンスがアルファベット順で最初のインスタンスだったので、nova-compute が最初に操作したのがそのインスタンスだったというだけでした。

さらなるトラブルシューティングにより、libvirt がまったく動作していないことがわかりました。これは大きな手がかりです。libvirt が動作していないと、KVM によるインスタンスの仮想化ができません。libvirt を開始させようとしても、libvirt は何も表示せずすぐに停止しました。libvirt のログでは理由がわかりませんでした。

次に、libvirtd デーモンをコマンドラインにおいて実行しました。最終的に次のような役に立つエラーメッセージが得られました。d-bus

に接続できませんでした。このため、滑稽に聞こえるかもしれませんが、libvirt 、その結果として nova-compute も D-Bus に依存していて、どういう訳か D-Bus がクラッシュしました。単に D-Bus を開始するだけで、一連のプログラムがうまく動くようになり、すぐに全部が元に戻り動作状態になりました。

アップグレード

Object Storage 以外では、OpenStack をあるバージョンから別のバージョンへアップグレードするには、非常に労力を伴います。

一般的に、アップグレード作業は以下の手順で行います。

1. リリースノートとドキュメントを読みます。

2. 異なるバージョン間の非互換性を確認します。

3. アップグレードスケジュールの計画を立て、テストクラスターで手順どおりにアップグレードができることを確認します。

4. アップグレードを実行します。

ユーザーのインスタンスが実行中のまま、アップグレードを実行することができます。しかしながら、この方法は危険です。ユーザーに対する適切な通知を忘れないようにしてください。そして、バックアップも忘れないでください。

最も成功すると考えられる一般的な順番は次のとおりです。

1. OpenStack Identity サービス（keystone）をアップグレードします。

2. OpenStack Image サービス（glance）をアップグレードします。

3. すべての OpenStack Compute（nova）サービスをアップグレードします。

4. すべての OpenStack Block Storage（cinder）サービスをアップグレードします。

これらのステップそれぞれに対して、以下のサブステップを完了します。

1. サービスを停止します。

2. 設定ファイルとデータベースのバックアップを作成します。

3. ディストリビューションのパッケージマネージャーを用いてパッケージをアップグレードします。

4. リリースノートに従って設定ファイルを更新します。

5. データベースのアップグレードを適用します。

6. サービスを再起動します。

7. すべてが正しく動作することを確認します。

おそらく、すべて中で最も大切なステップは事前のアップグレードテストです。とくに新しいバージョンのリリース後すぐにアップグレードする場合、未発見のバグによってアップグレードがうまくいかないこともあるでしょう。管理者によっては、最初のアップデート版が出るまで待つことを選ぶ場合もあります。しかしながら、重要な環境の場合には、リリース版の開発やテストに参加することで、あなたのユースケースでのバグを確実に修正することもできるでしょう。

インスタンスを実行したまま、OpenStack Compute のアップグレードを行うには、ハイパーバイザーの機能のライブマイグレーションを使って、アップグレードを実行している間はインスタンスを他のマシンに移動し、終わったら元のマシンに戻す方法を取ることができます。しかしながら、データベースの更新を確実に行うことが非常に重要です。さもないと、クラスターが一貫性のない状態になってしまいます。

Performing some 'cleaning' of the cluster prior to starting the upgrade is also a good idea, to ensure the state is consistent. For example some have reported issues with instances that were not fully removed from the system after their deletion. Running a command equivalent to:

```
$ virsh list --all
```

to find deleted instances that are still registered in the hypervisor and removing them prior to running the upgrade can avoid issues.

アンインストール

我々は常に、自動配備システムを使って、まっさらの状態からシステムを再インストールすることを進めていますが、時として OpenStack を地道にシステムから削除しなければならない場合もあるでしょう。その場合には以下の手順となります。

- すべてのパッケージを削除する

- 残っているファイルを削除する
- データベースを削除する

これらの手順はお使いのディストリビューションにより異なりますが、一般には aptitude purge ~c $package のようなパッケージマネージャーの「purge（完全削除）」コマンドを探すとよいでしょう。その後で、このガイドの中に出てきたディレクトリにある不要なファイルを探します。データベースを適切にアンインストールする方法については、使用しているソフトウェアの適切なマニュアルを参照して下さい。

第12章 ネットワークのトラブルシューティング

「ip a」を使ってインタフェース状態をチェックする 139
クラウド上のネットワークトラフィック 140
経路上の障害を見つける 141
tcpdump 141
iptables 143
データベースにあるネットワーク設定 144
DHCP の問題をデバッグする 145
DNS の問題をデバッグする 148

ネットワークのトラブルシューティングは、残念ながら、非常に難しくややこしい作業です。ネットワークの問題は、クラウドのいくつかの場所で問題となりえます。論理的な問題解決手順を用いることは、混乱の緩和や迅速な切り分けに役立つでしょう。この章は、あなたがものにしたい情報を提供することを目標とします。

「ip a」を使ってインタフェース状態をチェックする

コンピュートノード上でnova-networkが動いている場合、次のコマンドでIPやVLAN、また、インターフェイスがUPしているか、などインターフェイス関連の情報を見られます。

```
# ip a
```

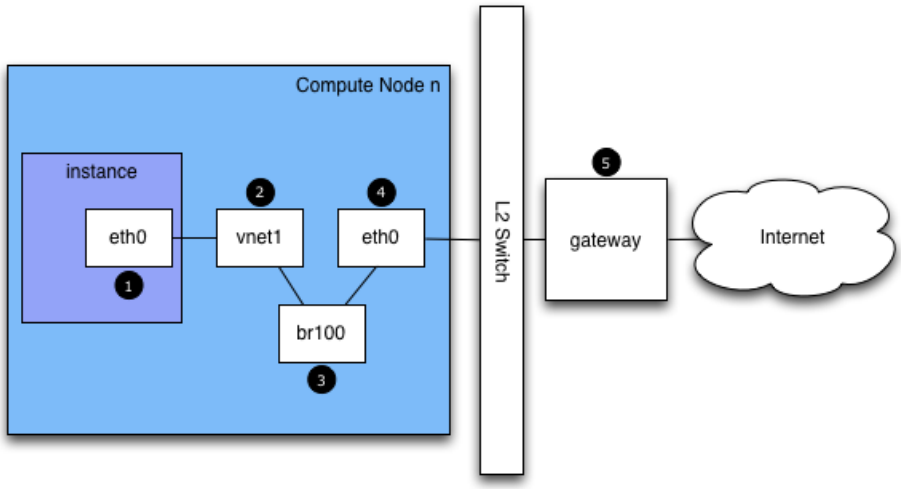
もしあなたがネットワークの問題に直面した場合、まず最初にするとよいのは、インターフェイスがUPになっているかを確認することです。例えば、

```
$ ip a | grep state
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue state UNKNOWN
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
 qlen 1000
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast master
 br100 state UP qlen 1000
4: virbr0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state
 DOWN
6: br100: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
```

virbr0の状態は無視することができます。なぜならそれはlibvirtが作成するデフォルトのブリッジで、OpenStackからは使われないからです。

クラウド上のネットワークトラフィック

もしあなたがインスタンスにログインしており、外部ホスト、例えばgoogle.comにpingした場合、そのpingパケットは下記経路を通ります。



1. インスタンスはパケットを生成し、インスタンス内の仮想NIC、例えばeth0にそれを渡します。

2. そのパケットはコンピュートホストの仮想NIC、例えば vnet1に転送されます。vnet NICの構成は、/etc/libvirt/qemu/instance-xxxxxxxx.xml を見ることで把握できます。

3. パケットはvnet NICからコンピュートノードのブリッジ、例えばbr100に転送されます。

   もしFlatDHCPManagerを使っているのであれば、ブリッジはコンピュートノード上に一つです。VlanManagerであれば、VLANごとにブリッジが存在します。

   下記コマンドを実行することで、パケットがどのブリッジを使うか確認できます。

   ```
   $ brctl show
   ```

   vnet NICを探してください。また、nova.confのflat_network_bridgeオプションも参考になります。

4. パケットはコンピュートノードの物理NICに送られます。このNICはbrctlコマンドの出力から、もしくはnova.confのflat_interfaceオプションから確認できます。

5. パケットはこのNICに送られた後、コンピュートノードのデフォルトゲートウェイに転送されます。パケットはこの時点で、おそらくあなたの管理範囲外でしょう。図には外部ゲートウェイを描いていますが、マルチホストのデフォルト構成では、コンピュートホストがゲートウェイです。

pingの応答経路は、これと逆方向です。

この経路説明によって、あなたはパケットが4つの異なるNICの間を行き来していることがわかったでしょう。これらのどのNICに問題が発生しても、ネットワークの問題となるでしょう。

経路上の障害を見つける

ネットワーク経路のどこに障害があるかを素早く見つけるには、pingを使います。まずあなたがインスタンス上で、google.comのような外部ホストにpingできるのであれば、ネットワークの問題はないでしょう。

もしそれができないのであれば、インスタンスがホストされているコンピュートノードのIPアドレスへpingを試行してください。もしそのIPにpingできるのであれば、そのコンピュートノードと、ゲートウェイ間のどこかに問題があります。

もしコンピュートノードのIPアドレスにpingできないのであれば、問題はインスタンスとコンピュートノード間にあります。これはコンピュートノードの物理NICとインスタンス vnet NIC間のブリッジ接続を含みます。

最後のテストでは、2つ目のインスタンスを起動し、2つのインスタンス間でお互いにpingを実行します。実行できるのであれば、この問題にはコンピュートノード上のファイアウォールが関係しているかもしれません。

tcpdump

ネットワーク問題の解決を徹底的に行う方法のひとつは、tcpdumpです。tcpdumpを使い、ネットワーク経路上の数点、問題のありそうなところから情報を収集することをおすすめします。もしGUIが好みであれ

ば、Wireshark (http://www.wireshark.org/)を試してみてはいかがでしょう。

例えば、以下のコマンドを実行します。

```
tcpdump -i any -n -v 'icmp[icmptype] = icmp-echoreply or
icmp[icmptype] = icmp-echo'
```

このコマンドは以下の場所で実行します。

1. クラウド外部のサーバー上

2. コンピュートノード上

3. コンピュートノード内のインスタンス上

例では、この環境には以下のIPアドレスが存在します

```
                    Instance
                    10.0.2.24
                    203.0.113.30
                    Compute Node
                    10.0.0.42
                    203.0.113.34
                    External Server
                    1.2.3.4
```

次に、新しいシェルを開いてtcpdumpの動いている外部ホストへpingを行います。もし外部サーバーとのネットワーク経路に問題がなければ、以下のように表示されます。

外部サーバー上

```
12:51:42.020227 IP (tos 0x0, ttl 61, id 0, offset 0, flags [DF], proto ICMP
 (1), length 84)
    203.0.113.30 > 1.2.3.4: ICMP echo request, id 24895, seq 1, length 64
12:51:42.020255 IP (tos 0x0, ttl 64, id 8137, offset 0, flags [none], proto
 ICMP (1), length 84)
    1.2.3.4 > 203.0.113.30: ICMP echo reply, id 24895, seq 1, length 64
```

コンピュートノード上

```
12:51:42.019519 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto ICMP
 (1), length 84)
    10.0.2.24 > 1.2.3.4: ICMP echo request, id 24895, seq 1, length 64
12:51:42.019519 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto ICMP
 (1), length 84)
    10.0.2.24 > 1.2.3.4: ICMP echo request, id 24895, seq 1, length 64
12:51:42.019545 IP (tos 0x0, ttl 63, id 0, offset 0, flags [DF], proto ICMP
 (1), length 84)
    203.0.113.30 > 1.2.3.4: ICMP echo request, id 24895, seq 1, length 64
12:51:42.019780 IP (tos 0x0, ttl 62, id 8137, offset 0, flags [none], proto
 ICMP (1), length 84)
    1.2.3.4 > 203.0.113.30: ICMP echo reply, id 24895, seq 1, length 64
12:51:42.019801 IP (tos 0x0, ttl 61, id 8137, offset 0, flags [none], proto
 ICMP (1), length 84)
    1.2.3.4 > 10.0.2.24: ICMP echo reply, id 24895, seq 1, length 64
12:51:42.019807 IP (tos 0x0, ttl 61, id 8137, offset 0, flags [none], proto
 ICMP (1), length 84)
    1.2.3.4 > 10.0.2.24: ICMP echo reply, id 24895, seq 1, length 64
```

インスタンス上

```
12:51:42.020974 IP (tos 0x0, ttl 61, id 8137, offset 0, flags [none], proto
 ICMP (1), length 84)
 1.2.3.4 > 10.0.2.24: ICMP echo reply, id 24895, seq 1, length 64
```

外部サーバーはpingリクエストを受信し、pingリプライを送信しています。コンピュートノード上では、pingとpingリプライがそれぞれ成功していることがわかります。また、見ての通り、コンピュートノード上ではパケットが重複していることもわかるでしょう。なぜならtcpdumpはブリッジと外向けインターフェイスの両方でパケットをキャプチャするからです。

iptables

Novaはiptablesを自動的に管理します。コンピュートノード上のインスタンス間でのパケット送受信、フローティングIPトラフィック、security groupのルール管理もそれに含まれます。

iptablesの現在の構成を見るには、以下のコマンドを実行します。

```
# iptables-save
```

注記

もしiptablesの構成を変更した場合、次のnova-network再起動時に前の状態に戻ります。iptablesの管理にはOpenStackを使ってください。

データベースにあるネットワーク設定

novaデータベースのテーブルには、いくつかのネットワーク情報が含まれています。

- fixed_ips: Novaに登録されたサブネットで利用可能なIPアドレス。このテーブルはfixed_ips.instance_uuid列で instances テーブルと関連付けられます。
- floating_ips: Novaに登録されたフローティングIPアドレス。このテーブルはfloating_ips.fixed_ip_id列でfixed_ipsテーブルと関連付けられます。
- instances: ネットワーク特有のテーブルではありませんが、fixed_ipとfloating_ipを使っているインスタンスの情報を管理します。

これらのテーブルから、フローティングIPが技術的には直接インスタンスにひも付けられておらず、固定IP経由であることがわかります。

手動でフローティングIPの関連付けを解除する

しばしば、フローティングIPを正しく開放しないままインスタンスが終了されることがあります。するとデータベースは不整合状態となるため、通常のツールではうまく開放できません。解決するには、手動でデータベースを更新する必要があります。

まず、インスタンスのUUIDを確認します。

```
mysql> select uuid from instances where hostname = 'hostname';
```

次に、そのUUIDから固定IPのエントリーを探します。

```
mysql> select * from fixed_ips where instance_uuid = '<uuid>';
```

関連するフローティングIPのエントリーが見つかります。

```
mysql> select * from floating_ips where fixed_ip_id = '<fixed_ip_id>';
```

最後に、フローティングIPを開放します。

```
mysql> update floating_ips set fixed_ip_id = NULL, host = NULL where
 fixed_ip_id = '<fixed_ip_id>';
```

また、ユーザプールからIPを開放することもできます。

```
mysql> update floating_ips set project_id = NULL where fixed_ip_id =
 '<fixed_ip_id>';
```

DHCP の問題をデバッグする

よくあるネットワークの問題に、インスタンスが起動しているにも関わらず、dnsmasqからのIPアドレス取得に失敗し、到達できないという現象があります。dnsmasqはnova-nwtworkサービスから起動されるDHCPサーバです。

もっともシンプルにこの問題を特定する方法は、インスタンス上のコンソール出力を確認することです。もしDHCPが正しく動いていなければ、下記のようにコンソールログを参照してください。

```
$ nova console-log <instance name or uuid>
```

もしインスタンスがDHCPからのIP取得に失敗していれば、いくつかのメッセージがコンソールで確認できるはずです。例えば、Cirrosイメージでは、このような出力になります。

```
udhcpc (v1.17.2) started
Sending discover...
Sending discover...
Sending discover...
No lease, forking to background
starting DHCP forEthernet interface eth0 [ [1;32mOK[0;39m ]
cloud-setup: checking http://169.254.169.254/2009-04-04/meta-data/instance-id
wget: can't connect to remote host (169.254.169.254): Network is unreachable
```

インスタンスが正しく起動した後、この手順でどこが問題かを切り分けることができます。

DHCPの問題はdnsmasqの不具合が原因となりがちです。まず、ログを確認し、その後該当するプロジェクト(テナント)のdnsmasqプロセスを再起動してください。VLANモードにおいては、dnsmasqプロセスはテナントごとに存在します。すでに該当のdnsmasqプロセスを再起動しているのであれば、もっともシンプルな解決法は、マシン上の全てのdnsmasqプロセスをkillし、nova-networkを再起動することです。最終手段として、rootで以下を実行してください。

```
# killall dnsmasq
# restart nova-network
```

注記

RHEL/CentOS/Fedora では openstack-nova-network ですが、Ubuntu/Debian では nova-network です。

nova-networkの再起動から数分後、新たなdnsmasqプロセスが動いていることが確認できるでしょう。

```
# ps aux | grep dnsmasq
nobody 3735 0.0 0.0 27540 1044 ? S 15:40 0:00 /usr/sbin/dnsmasq --strict-
order --bind-interfaces --conf-file=
    --domain=novalocal --pid-file=/var/lib/nova/networks/nova-br100.pid --
listen-address=192.168.100.1
    --except-interface=lo --dhcp-range=set:'novanetwork',192.168.100.2,
static,120s --dhcp-lease-max=256
    --dhcp-hostsfile=/var/lib/nova/networks/nova-br100.conf --dhcp-script=/
usr/bin/nova-dhcpbridge --leasefile-ro
root 3736 0.0 0.0 27512 444 ? S 15:40 0:00 /usr/sbin/dnsmasq --strict-order
 --bind-interfaces --conf-file=
     --domain=novalocal --pid-file=/var/lib/nova/networks/nova-br100.pid --
listen-address=192.168.100.1
     --except-interface=lo --dhcp-range=set:'novanetwork',192.168.100.2,
static,120s --dhcp-lease-max=256
     --dhcp-hostsfile=/var/lib/nova/networks/nova-br100.conf --dhcp-script=/
usr/bin/nova-dhcpbridge --leasefile-ro
```

もしまだインスタンスがIPアドレスを取得できない場合、次はdnsmasqがインスタンスからのDHCPリクエストを見えているか確認します。dnsmasqプロセスが動いているマシンで、/var/log/syslogを参照し、dnsmasqの出力を確認します。なお、マルチホストモードで動作している場合は、dnsmasqプロセスはコンピュートノードで動作します。もしdnsmasqがリクエストを正しく受け取り、処理していれば、以下のような出力になります。

```
Feb 27 22:01:36 mynode dnsmasq-dhcp[2438]: DHCPDISCOVER(br100)
 fa:16:3e:56:0b:6f
Feb 27 22:01:36 mynode dnsmasq-dhcp[2438]: DHCPOFFER(br100) 192.168.100.3
 fa:16:3e:56:0b:6f
Feb 27 22:01:36 mynode dnsmasq-dhcp[2438]: DHCPREQUEST(br100) 192.168.100.3
 fa:16:3e:56:0b:6f
Feb 27 22:01:36 mynode dnsmasq-dhcp[2438]: DHCPACK(br100) 192.168.100.3
 fa:16:3e:56:0b:6f test
```

もしDHCPDISCOVERが見つからなければ、dnsmasqが動いているマシンがインスタンスからパケットを受け取れない何らかの問題があります。もし上記の出力が全て確認でき、かついまだにIPアドレスを取得できないのであれば、パケットはインスタンスからdnsmasq稼働マシンに到達していますが、その復路に問題があります。

もし他にこのようなメッセージを確認できたのであれば、

```
Feb 27 22:01:36 mynode dnsmasq-dhcp[25435]: DHCPDISCOVER(br100)
 fa:16:3e:78:44:84 no address available
```

これはdnsmasqの、もしくはdnsmasqとnova-network両方の問題です。(例えば上記では、OpenStack Compute データベース上に利用可能な固定IPがなく、dnsmasqがIPアドレスを払い出せない問題が発生しています)

もしdnsmasqのログメッセージで疑わしいものがあれば、コマンドラインにてdnsmasqが正しく動いているか確認してください。

```
$ ps aux | grep dnsmasq
```

出力は以下のようになります。

```
108 1695 0.0 0.0 25972 1000 ? S Feb26 0:00 /usr/sbin/dnsmasq -u libvirt-
dnsmasq --strict-order --bind-interfaces
 --pid-file=/var/run/libvirt/network/default.pid --conf-file= --except-
interface lo --listen-address 192.168.122.1
 --dhcp-range 192.168.122.2,192.168.122.254 --dhcp-leasefile=/var/lib/
libvirt/dnsmasq/default.leases
 --dhcp-lease-max=253 --dhcp-no-override
nobody 2438 0.0 0.0 27540 1096 ? S Feb26 0:00 /usr/sbin/dnsmasq --strict-
order --bind-interfaces --conf-file=
 --domain=novalocal --pid-file=/var/lib/nova/networks/nova-br100.pid --
listen-address=192.168.100.1
 --except-interface=lo --dhcp-range=set:'novanetwork',192.168.100.2,static,
120s --dhcp-lease-max=256
 --dhcp-hostsfile=/var/lib/nova/networks/nova-br100.conf --dhcp-script=/usr/
bin/nova-dhcpbridge --leasefile-ro
root 2439 0.0 0.0 27512 472 ? S Feb26 0:00 /usr/sbin/dnsmasq --strict-order
 --bind-interfaces --conf-file=
 --domain=novalocal --pid-file=/var/lib/nova/networks/nova-br100.pid --
listen-address=192.168.100.1
 --except-interface=lo --dhcp-range=set:'novanetwork',192.168.100.2,static,
120s --dhcp-lease-max=256
 --dhcp-hostsfile=/var/lib/nova/networks/nova-br100.conf --dhcp-script=/usr/
bin/nova-dhcpbridge --leasefile-ro
```

もし問題がdnsmasqと関係しないようであれば、tcpdumpを使ってパケットロスがないか確認してください。

DHCPトラフィックはUDPを使います。そして、クライアントは68番ポートからサーバーの67番ポートへパケットを送信します。新しいインスタンスを起動し、機械的にNICをリッスンしてください。トラフィックに現れない通信を特定できるまで行います。tcpdumpでbr100上のポート67、68をリッスンするには、こうします。

```
# tcpdump -i br100 -n port 67 or port 68
```

また、「ip a」や「brctl show」などのコマンドを使って、インターフェイスが実際にUPしているか、あなたが考えたとおりに設定されているか、正当性を検査をすべきです。

DNS の問題をデバッグする

あなたがインスタンスにsshできるけれども、プロンプトが表示されるまで長い時間(約1分)を要する場合、DNSに問題があるかもしれません。sshサーバーが接続元IPアドレスのDNS逆引きをおこなうこと、それがこの問題の原因です。もしあなたのインスタンスでDNSが正しく引けない場合、sshのログインプロセスが完了するには、DNSの逆引きがタイムアウトするまで待たなければいけません。

DNS問題のデバッグをするとき、そのインスタンスのdnsmasqが動いているホストが、名前解決できるかを確認することから始めます。もしホストができないのであれば、インスタンスも同様でしょう。

DNSが正しくホスト名をインスタンス内から解決できているか確認する簡単な方法は、hostコマンドです。もしDNSが正しく動いていれば、以下メッセージが確認できます。

```
$ host openstack.org
openstack.org has address 174.143.194.225
openstack.org mail is handled by 10 mx1.emailsrvr.com.
openstack.org mail is handled by 20 mx2.emailsrvr.com.
```

もしあなたがCirrosイメージを使っているのであれば、「host」プログラムはインストールされていません。その場合はpingを使い、ホスト名が解決できているか判断できます。もしDNSが動いていれば、ping結果の先頭行はこうなるはずです。

```
$ ping openstack.org
PING openstack.org (174.143.194.225): 56 data bytes
```

もしインスタンスがホスト名の解決に失敗するのであれば、DNSに問題があります。例えば、

```
$ ping openstack.org
ping: bad address 'openstack.org'
```

OpenStackクラウドにおいて、dnsmasqプロセスはDHCPサーバに加えてDNSサーバーの役割を担っています。dnsmasqの不具合は、インスタンスにおけるDNS関連問題の原因となりえます。前節で述べたように、dnsmasqの不具合を解決するもっともシンプルな方法は、マシン上のすべてのdnsmasqプロセスをkillし、nova-networkを再起動することです。しかしながら、このコマンドは該当ノード上で動いているすべてのインスタンス、特に問題がないテナントにも影響します。最終手段として、rootで以下を実行します。

```
# killall dnsmasq
# restart nova-network
```

dnsmasq再起動後に、DNSが動いているか確認します。

dnsmasqの再起動でも問題が解決しないときは、tcpdumpで問題がある場所のパケットトレースを行う必要があるでしょう。DNSサーバーはUDPポート53番でリッスンします。あなたのコンピュートノードのブリッジ(br100など)上でDNSリクエストをチェックしてください。コンピュートノード上にて、tcpdumpでリッスンを開始すると、

```
# tcpdump -i br100 -n -v udp port 53
tcpdump: listening on br100, link-type EN10MB (Ethernet), capture size 65535
 bytes
```

インスタンスへのssh、ping openstack.orgの試行にて、以下のようなメッセージが確認できるでしょう。

```
16:36:18.807518 IP (tos 0x0, ttl 64, id 56057, offset 0, flags [DF], proto
 UDP (17), length 59)
 192.168.100.4.54244 > 192.168.100.1.53: 2+ A? openstack.org. (31)
16:36:18.808285 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto UDP
 (17), length 75)
 192.168.100.1.53 > 192.168.100.4.54244: 2 1/0/0 openstack.org. A 174.143.
194.225 (47)
```

第13章 ロギングと監視

ログはどこにあるのか？ 151
ログの読み方 152
インスタンスリクエストの追跡 153
カスタムログの追加 154
RabbitMQ Web管理インターフェイス および rabbitmqctl 154
ログの集中管理 155
StackTach 157
監視 157

OpenStackクラウドは、様々なサービスから構成されるため、多くのログファイルが存在します。このセクションでは、それぞれのログの場所と取り扱い、そしてシステムのさらなる監視方法について説明します。

ログはどこにあるのか？

Ubuntu では、ほとんどのサービスが /var/log ディレクトリ以下のディレクトリにログファイルを出力するという慣習に従っています。

クラウドコントローラー

サービス	ログの場所
nova-*	/var/log/nova
glance-*	/var/log/glance
cinder-*	/var/log/cinder
keystone	/var/log/keystone
horizon	/var/log/apache2/
その他 (Swift, dnsmasq)	/var/log/syslog

コンピュートノード

libvirt: /var/log/libvirt/libvirtd.log

VMインスタンスのコンソール（ブートメッセージ): /var/lib/nova/instances/instance-<instance id>/console.log

ブロックストレージ ノード

cinder: /var/log/cinder/cinder-volume.log

ログの読み方

OpenStack サービスは標準のロギングレベルを利用しています。重要度のレベルは次の通りです(重要度の低い順): DEBUG、INFO、AUDIT、WARNING、ERROR、CRTICAL、TRACE。特定のログレベルより「重要」な場合のみメッセージはログに出力されます。ログレベルDEBUGの場合、すべてのログが出力されます。TRACEの場合、ソフトウェアがスタックトレースを持つ場合にのみログに出力されます。INFOの場合、情報のみのメッセージも含めて出力されます。

DEBUG レベルのロギングを無効にするには、以下のように /etc/nova/nova.conf を編集します。

```
debug=false
```

Keystoneは少し異なる動作をします。ロギングレベルを変更するためには、/etc/keystone/logging.confを編集し、 logger_root と handler_file を修正する必要があります。

Horizon のロギング設定は /etc/openstack_dashboard/local_settings.py で行います。Horizon は Django web アプリケーションですので、Django Logging (https://docs.djangoproject.com/en/dev/topics/logging/) フレームワークの規約に従います。

エラーの原因を見つけるための典型的な最初のステップは、CRTICAL、TRACE、ERRORなどのメッセージがログファイルの終わりで出力されていないかを確認することです。

トレース(Pythonのコールスタック)付きのCRITICALなログメッセージの例は次の通りです。

```
2013-02-25 21:05:51 17409 CRITICAL cinder [-] Bad or unexpected response from the storage volume
 backend API: volume group
 cinder-volumes doesn't exist
2013-02-25 21:05:51 17409 TRACE cinder Traceback (most recent call last):
2013-02-25 21:05:51 17409 TRACE cinder File "/usr/bin/cinder-volume", line 48, in <module>
2013-02-25 21:05:51 17409 TRACE cinder service.wait()
2013-02-25 21:05:51 17409 TRACE cinder File "/usr/lib/python2.7/dist-packages/cinder/service.py",
 line 422, in wait
2013-02-25 21:05:51 17409 TRACE cinder _launcher.wait()
2013-02-25 21:05:51 17409 TRACE cinder File "/usr/lib/python2.7/dist-packages/cinder/service.py",
 line 127, in wait
2013-02-25 21:05:51 17409 TRACE cinder service.wait()
2013-02-25 21:05:51 17409 TRACE cinder File "/usr/lib/python2.7/dist-packages/eventlet/greenthread.
py", line 166, in wait
2013-02-25 21:05:51 17409 TRACE cinder return self._exit_event.wait()
2013-02-25 21:05:51 17409 TRACE cinder File "/usr/lib/python2.7/dist-packages/eventlet/event.py",
 line 116, in wait
2013-02-25 21:05:51 17409 TRACE cinder return hubs.get_hub().switch()
2013-02-25 21:05:51 17409 TRACE cinder File "/usr/lib/python2.7/dist-packages/eventlet/hubs/hub.py",
 line 177, in switch
2013-02-25 21:05:51 17409 TRACE cinder return self.greenlet.switch()
```

```
2013-02-25 21:05:51 17409 TRACE cinder File "/usr/lib/python2.7/dist-packages/eventlet/greenthread.
py", line 192, in main
2013-02-25 21:05:51 17409 TRACE cinder result = function(*args, **kwargs)
2013-02-25 21:05:51 17409 TRACE cinder File "/usr/lib/python2.7/dist-packages/cinder/service.py",
 line 88, in run_server
2013-02-25 21:05:51 17409 TRACE cinder server.start()
2013-02-25 21:05:51 17409 TRACE cinder File "/usr/lib/python2.7/dist-packages/cinder/service.py",
 line 159, in start
2013-02-25 21:05:51 17409 TRACE cinder self.manager.init_host()
2013-02-25 21:05:51 17409 TRACE cinder File "/usr/lib/python2.7/dist-packages/cinder/volume/manager.
py", line 95,
 in init_host
2013-02-25 21:05:51 17409 TRACE cinder self.driver.check_for_setup_error()
2013-02-25 21:05:51 17409 TRACE cinder File "/usr/lib/python2.7/dist-packages/cinder/volume/driver.
py", line 116,
 in check_for_setup_error
2013-02-25 21:05:51 17409 TRACE cinder raise exception.VolumeBackendAPIException(data=
exception_message)
2013-02-25 21:05:51 17409 TRACE cinder VolumeBackendAPIException: Bad or unexpected response from
 the storage volume
 backend API: volume group cinder-volumes doesn't exist
2013-02-25 21:05:51 17409 TRACE cinder
```

この例では、ボリュームのバックエンドがストレージボリュームをセットアップができなかったため、cinder-volumesが起動に失敗し、スタックトレースを出力しています。おそらく、設定ファイルで指定されたLVM ボリュームが存在しないためと考えられます。

エラーログの例:

```
2013-02-25 20:26:33 6619 ERROR nova.openstack.common.rpc.common [-] AMQP
 server on localhost:5672 is unreachable:
 [Errno 111] ECONNREFUSED. Trying again in 23 seconds.
```

このエラーでは、novaサービスがRabbitMQへの接続に失敗していました。接続が拒否されたというエラーが出力されています。

インスタンスリクエストの追跡

インスタンスが正しく動作していない場合、インスタンスに関連したログを調べる必要があります。これらのログは複数のnova-*サービスが出力しており、クラウドコントローラーとコンピュートノードの両方に存在します。

一般的な方法はインスタンスのUUIDをキーにして、各サービスのログを追跡することです。

次のような例を考えてみましょう。

```
ubuntu@initial:~$ nova list
+--------------------------------------+--------+--------+-------------------------+
| ID                                   | Name   | Status | Networks                |
+--------------------------------------+--------+--------+-------------------------+
| fafed8-4a46-413b-b113-f1959ffe | cirros | ACTIVE | novanetwork=192.168.100.3|
+--------------------------------------+--------+--------+-------------------------+
```

ここで、インスタンスのUUIDは faf7ded8-4a46-413b-b113-f19590746ffeです。クラウドコントローラー上の /var/log/nova-*.logファイルをこの文字列で検索すると、nova-api.logと nova-scheduler.logで見つかります。同様にコンピュートノードで検索した場合、nova-network.log と nova-compute.logで見つかります。もし、ERRORやCRITICALのメッセージが存在しない場合、最後のログエントリが、何が悪いかのヒントを示しているかもしれません。

カスタムログの追加

もし、ログに十分な情報がない場合、あなた自身でカスタマイズしたログを nova-*サービスに追加することができます。

ソースファイルは /usr/lib/python2.7/dist-packages/novaに存在します。

ログステートメントを追加するには、次の行をファイルの先頭に置きます。ほとんどのファイルでは、これらは既に存在します。

```
from nova.openstack.common import log as logging
LOG = logging.getLogger(__name__)
```

DEBUGログステートメントを追加するには次のようにします。

```
LOG.debug("This is a custom debugging statement")
```

以下に例を示しますが、全てのログメッセージはアンダースコアで始まり、括弧で括られていることに気づいたでしょうか?

```
LOG.debug(_("Logging statement appears here"))
```

これは、ログメッセージを異なる言語に翻訳するためにgettext (http://docs.python.org/2/library/gettext.html) 国際化ライブラリ を利用しているためです。カスタムログには必要ありませんが、もし、OpenStackプロジェクトにログステートメントを含むコードを提供する場合は、アンダースコアと括弧でログメッセージを囲わなければなりません。

RabbitMQ Web管理インターフェイス および rabbitmqctl

接続失敗の問題は別とすると、RabbitMQログファイルはOpenStackに関連した問題のデバッグにあまり有用ではありません。その代わりにRabbitMQ Web管理インターフェイスを推奨します。 クラウドコントローラーにおいて、以下のコマンドで有効になります。

```
# /usr/lib/rabbitmq/bin/rabbitmq-plugins enable rabbitmq_management
# service rabbitmq-server restart
```

RabbitMQ Web管理インターフェイスは、クラウドコントローラーから http://localhost:55672 でアクセスできます。

注記

Ubuntu 12.04はRabiitMQのバージョン2.7.1を55672番ポートを使うようにインストールします。RabbitMQバージョン3.0以降では15672が利用されます。Ubuntuマシン上でどのバージョンのRabbitMQが実行されているかは次のように確認できます。

```
$ dpkg -s rabbitmq-server | grep "Version:"
Version: 2.7.1-0ubuntu4
```

RabbitMQ Web 管理インターフェイスを有効にするもう一つの方法としては、 rabbitmqctl コマンドを利用します。例えば rabbitmqctl list_queues| grep cinder は、キューに残っているメッセージを表示します。メッセージが存在する場合、CinderサービスがRabbitMQに正しく接続できてない可能性があり、再起動が必要かもしれません。

RabbitMQで監視すべき項目としては、各キューでのアイテムの数と、サーバーでの処理時間の統計情報があります。

ログの集中管理

クラウドは多くのサーバーから構成されるため、各サーバー上にあるイベントログを繋ぎあわせて、ログをチェックしなければなりません。よい方法は全てのサーバーのログを一ヶ所にまとめ、同じ場所で確認できるようにすることです。

Ubuntuはrsyslog をデフォルトのロギングサービスとして利用します。rsyslog はリモートにログを送信する機能を持っているので、何かを追加でインストールする必要はなく、設定ファイルを変更するだけです。リモート転送を実施する際は、盗聴を防ぐためにログが自身の管理ネットワーク上を通る、もしくは暗号化VPNを利用することを考慮する必要があります。

rsyslog クライアント設定

まず始めに、全てのOpenStackコンポーネントのログを標準ログに加えてsyslogに出力するように設定します。また、各コンポーネントが異なるsyslogファシリティになるように設定します。これによりログサーバー上で、個々のコンポーネントのログを分離しやすくなります。

nova.conf:

```
use_syslog=True
syslog_log_facility=LOG_LOCAL0
```

glance-api.conf および glance-registry.conf:

```
use_syslog=True
syslog_log_facility=LOG_LOCAL1
```

cinder.conf:

```
use_syslog=True
syslog_log_facility=LOG_LOCAL2
```

keystone.conf:

```
use_syslog=True
syslog_log_facility=LOG_LOCAL3
```

Swift

デフォルトでSwiftはsyslogにログを出力します。

次に、 /etc/rsyslog.d/client.confに次の行を作成します。

```
*.* @192.168.1.10
```

これは、rsyslogに全てのログを指定したIPアドレスに送るように命令しています。この例では、IPアドレスはクラウドコントローラーを指しています。

rsyslog サーバー設定

集中ログサーバーとして使用するサーバーを決めます。ログ専用のサーバーを利用するのが最も良いです。 /etc/rsyslog.d/server.conf を次のように作成します。

```
# Enable UDP
$ModLoad imudp
# Listen on 192.168.1.10 only
$UDPServerAddress 192.168.1.10
# Port 514
$UDPServerRun 514

# Create logging templates for nova
$template NovaFile,"/var/log/rsyslog/%HOSTNAME%/nova.log"
$template NovaAll,"/var/log/rsyslog/nova.log"



# Log everything else to syslog.log
$template DynFile,"/var/log/rsyslog/%HOSTNAME%/syslog.log"
*.* ?DynFile
```

```
# Log various openstack components to their own individual file
local0.* ?NovaFile
local0.* ?NovaAll
& ~
```

この設定例はnovaサービスのみを扱っています。はじめに rsyslog を UDP 514番ポートで動作するサーバーとして設定します。次に一連のログテンプレートを作成します。ログテンプレートは受け取ったログをどこに保管するかを指定します。上記の例を用いると、c01.example.comから送られるnovaのログは次の場所に保管されます。

- /var/log/rsyslog/c01.example.com/nova.log
- /var/log/rsyslog/nova.log

c02.example.comから送られたログはこちらに保管されます。

- /var/log/rsyslog/c02.example.com/nova.log
- /var/log/rsyslog/nova.log

全てのノードからのnovaのログを含む集約されたログだけでなく、個々のコンピュートノードのログも持つことになります。

StackTach

StackTachはRackspaceによって作られたツールで、novaから送られた通知を収集してレポートします。通知は本質的にはログと同じですが、より詳細な情報を持ちます。通知の概要のよい資料は、System Usage Data(https://wiki.openstack.org/wiki/SystemUsageData)にあります。

novaで通知の送信を有効化するには次の行を nova.confに追加します。

```
notification_topics=monitor
notification_driver=nova.openstack.common.notifier.rabbit_notifier
```

novaが通知を送信するように設定後、StackTachをインストールします。StackTachは新しく、頻繁に更新されるので、インストール手順はすぐに古くなります。StackTach GitHub repo (https://github.com/rackerlabs/stacktach) とデモビデオを参照して下さい。

監視

二つの監視のタイプがあります。問題の監視と、利用傾向の監視です。前者は全てのサービスが動作していることを保証するものであり、後者は時間に沿ったリソース利用状況を監視することで、潜在的なボトルネックの発見とアップグレードのための情報を得るものです。

プロセス監視

基本的なアラート監視は、単純に必要なプロセスが実行されているかどうかをチェックすることです。例えば、 nova-apiサービスがクラウドコントローラーで動作していることを確認します。

```
[ root@cloud ~ ] # ps aux | grep nova-api
nova 12786 0.0 0.0 37952 1312 ? Ss Feb11 0:00 su -s /bin/sh -c exec nova-api --config-
file=/etc/nova/nova.conf nova
nova 12787 0.0 0.1 135764 57400 ? S Feb11 0:01 /usr/bin/python /usr/bin/nova-api --
config-file=/etc/nova/nova.conf
nova 12792 0.0 0.0 96052 22856 ? S Feb11 0:01 /usr/bin/python /usr/bin/nova-api --
config-file=/etc/nova/nova.conf
nova 12793 0.0 0.3 290688 115516 ? S Feb11 1:23 /usr/bin/python /usr/bin/nova-api --
config-file=/etc/nova/nova.conf
nova 12794 0.0 0.2 248636 77068 ? S Feb11 0:04 /usr/bin/python /usr/bin/nova-api --
config-file=/etc/nova/nova.conf
root 24121 0.0 0.0 11688 912 pts/5 S+ 13:07 0:00 grep nova-api
```

NagiosとNRPEを使って、クリティカルなプロセスの自動化されたアラートを作成することが可能です。nova-compute プロセスがコンピュートノードで動作していることを保証するために、Nagiosサーバー上で次のようなアラートを作成します。

```
define service {
    host_name c01.example.com
    check_command check_nrpe_1arg!check_nova-compute
    use generic-service
    notification_period 24x7
    contact_groups sysadmins
    service_description nova-compute
}
```

そして、対象のコンピュートノードにおいて、次のようなNRPE設定を作成します。

```
command[check_nova-compute]=/usr/lib/nagios/plugins/check_procs -c 1: -a nova-compute
```

Nagiosは常に一つ以上の nova-computeサービスが動作しているかをチェックします。

リソースのアラート

リソースのアラートは、一つ以上のリソースが極めて少なくなった場合に通知するものです。監視の閾値はインストールされたOpenStackの環境に合わせて設定すべきですが、リソース監視の利用方法は、OpenStackに固有の話ではありません。一般的なアラートのタイプが利用できます。

監視項目に含む幾つかのリソースをあげます。

- ディスク使用量

- サーバー負荷
- メモリ使用量
- ネットワーク I/O
- 利用可能な vCPU 数

例として、コンピュートノード上のディスク容量をNagiosを使って監視する場合、次のようなNagios設定を追加します。

```
define service {
    host_name c01.example.com
    check_command check_nrpe!check_all_disks!20% 10%
    use generic-service
    contact_groups sysadmins
    service_description Disk
}
```

コンピュートノード上では、次のようなNRPE設定を追加します。

```
command[check_all_disks]=/usr/lib/nagios/plugins/check_disk -w $ARG1$ -c $ARG2$ -e
```

Naigosは、80%のディスク使用率でWARNING、90%でCRITICALを警告します。%i_dummy%f_dummy

OpenStack固有のリソース

メモリ、ディスク、CPUのような一般的なリソースは、全てのサーバー(OpenStackに関連しないサーバーにも)に存在するため、サーバーの状態監視において重要です。OpenStackの場合、インスタンスを起動するために必要なリソースが確実に存在するかの確認という点でも重要です。OpenStackのリソースを見るためには幾つかの方法が存在します。

最初は nova コマンドを利用した例です。

```
# nova usage-list
```

このコマンドはテナント上で実行されるインスタンスのリストと、インスタンス全体の簡単な利用統計を表示します。クラウドの簡単な概要を得るのに便利なコマンドですが、より詳細な情報については表示しません。

次に nova データベースは 利用情報に関して3つのテーブルを持っています。

nova.quotasと nova.quota_usages テーブルはクォータの情報が保管されています。もし、テナントのクォータがデフォルト設定と異なる場合、nova.quotasに保管されます。以下に例を示します。

```
mysql> select project_id, resource, hard_limit from quotas;
+----------------------------------+-----------------------------+------------+
| project_id                       | resource                    | hard_limit |
+----------------------------------+-----------------------------+------------+
| 628df59f091142399e0689a2696f5baa | metadata_items              | 128        |
| 628df59f091142399e0689a2696f5baa | injected_file_content_bytes | 10240      |
| 628df59f091142399e0689a2696f5baa | injected_files              | 5          |
| 628df59f091142399e0689a2696f5baa | gigabytes                   | 1000       |
| 628df59f091142399e0689a2696f5baa | ram                         | 51200      |
| 628df59f091142399e0689a2696f5baa | floating_ips                | 10         |
| 628df59f091142399e0689a2696f5baa | instances                   | 10         |
| 628df59f091142399e0689a2696f5baa | volumes                     | 10         |
| 628df59f091142399e0689a2696f5baa | cores                       | 20         |
+----------------------------------+-----------------------------+------------+
```

nova.quota_usagesテーブルはどのくらいリソースをテナントが利用しているかを記録しています。

```
mysql> select project_id, resource, in_use from quota_usages where project_id like '628%';
+----------------------------------+--------------+--------+
| project_id                       | resource     | in_use |
+----------------------------------+--------------+--------+
| 628df59f091142399e0689a2696f5baa | instances    | 1      |
| 628df59f091142399e0689a2696f5baa | ram          | 512    |
| 628df59f091142399e0689a2696f5baa | cores        | 1      |
| 628df59f091142399e0689a2696f5baa | floating_ips | 1      |
| 628df59f091142399e0689a2696f5baa | volumes      | 2      |
| 628df59f091142399e0689a2696f5baa | gigabytes    | 12     |
| 628df59f091142399e0689a2696f5baa | images       | 1      |
+----------------------------------+--------------+--------+
```

リソース利用をテナントのクォータと結合することで、利用率を求めることが出来ます。例えば、テナントが、10個のうち1つのFloating IPを利用しているとすると、10%のFloating IP クォータを利用することになります。これを行なうことで、レポートを作ることができます。

```
+----------------------------------+-----------+------------+--------------+
| some_tenant                                                              |
+----------------------------------+-----------+------------+--------------+
| Resource                         | Used      | Limit      |              |
+----------------------------------+-----------+------------+--------------+
| cores                            | 1         | 20         |          5 % |
| floating_ips                     | 1         | 10         |         10 % |
| gigabytes                        | 12        | 1000       |          1 % |
| images                           | 1         | 4          |         25 % |
| injected_file_content_bytes      | 0         | 10240      |          0 % |
| injected_file_path_bytes         | 0         | 255        |          0 % |
| injected_files                   | 0         | 5          |          0 % |
| instances                        | 1         | 10         |         10 % |
| key_pairs                        | 0         | 100        |          0 % |
| metadata_items                   | 0         | 128        |          0 % |
| ram                              | 512       | 51200      |          1 % |
| reservation_expire               | 0         | 86400      |          0 % |
| security_group_rules             | 0         | 20         |          0 % |
| security_groups                  | 0         | 10         |          0 % |
| volumes                          | 2         | 10         |         20 % |
+----------------------------------+-----------+------------+--------------+
```

上記の出力はこのGithub（https://github.com/cybera/novac/blob/dev/libexec/novac-quota-report）にあるカスタムスクリプトを用いて作成しました。

注記

このスクリプトは特定のOpenStackインストール環境向けなので、自身の環境に適用する際には変更しなくてはいけませんが、ロジックは簡単に変更できるでしょう。

インテリジェントなアラート

インテリジェントなアラートは運用における継続的インテグレーションの要素の一つです。例えば、以下のような方法で簡単にGlanceが起動しているかを簡単に確認できます。glance-apiとglance-registryプロセスが起動していることを確認する、もしくはglace-apiが9292ポートに応答するといった方法です。

しかし、イメージサービスにイメージが正しくアップロードされたことをどのように知ればいいのでしょうか? もしかしたら、イメージサービスが保管しているイメージのディスクが満杯、もしくはS3のバックエンドがダウンしているかもしれません。簡易的なイメージアップロードを行なうことでこれをチェックすることができます。

```
#!/bin/bash
#
# assumes that reasonable credentials have been stored at
# /root/auth


. /root/openrc
wget https://launchpad.net/cirros/trunk/0.3.0/+download/cirros-0.3.0-x86_64-disk.img
glance image-create --name='cirros image' --is-public=true --container-format=bare --
disk-format=qcow2 < cirros-0.3.0-x8
6_64-disk.img
```

このスクリプトを(Nagiosのような)監視システムに組込むことで、イメージカタログのアップロードが動作していることを自動的に確認することができます。

注記

毎回テスト後にイメージを削除する必要があります。イメージサービスからイメージが削除できるかのテストにしてしまえば、さらによいです。

インテリジェントなアラートは、この章で述べられているの他のアラートよりも計画、実装にかなり時間を要します。インテリジェントなアラートを実装する流れは次のようになります。

- 構築したクラウドにおいて一般的なアクションをレビューする

- それらのアクションに対して自動テストを作成する
- それらのテストをアラートシステムに組み込む

インテリジェントなアラートのその他の例としては以下があります。

- インスタンスの起動と削除が可能か?
- ユーザの作成は可能か?
- オブジェクトの保存と削除は可能か?
- ボリュームの作成と削除は可能か?

トレンド

トレンドはどのようにクラウドが実行されているかの素晴しい見通しを与えてくれます。例えば、負荷の高い日がたまたま発生したのか、新しいコンピュートノードを追加すべきか、などです。

トレンドはアラートとは全く異なったアプローチです。アラートは0か1かの結果(チェックが成功するか失敗するか)に注目しているのに対して、トレンドはある時点での状態を定期的に記録します。十分な量が記録されれば、時系列でどのように値が変化するかを確認できます。

これまでに示した全てのアラートタイプは、トレンドレポートに利用可能です。その他のトレンドの例は以下の通りです。

- 各コンピュートノード上のインスタンス数
- 使用中のフレーバー
- 使用中のボリューム数
- 1時間あたりの Object Storage リクエスト数
- 1時間あたりの nova-api リクエスト数
- ストレージサービスの I/O の統計

例として、nova-apiの使用を記録することでクラウドコントローラーをスケールする必要があるかを追跡できます。nova-apiのリクエスト数に注目することにより、nova-apiプロセスを追加するか、もしくは、nova-apiを実行するための新しいサーバーを導入することまで行なうかを決定することができます。リクエストの概数を取得するには/var/log/nova/nova-api.logのINFOメッセージを検索します。

```
# grep INFO /var/log/nova/nova-api.log | wc
```

成功したリクエストを検索することで、更なる情報を取得できます。

```
# grep ” 200 ” /var/log/nova/nova-api.log | wc
```

このコマンドを定期的に実行し結果を記録することで、トレンドレポートを作ることができます。これにより/var/log/nova/nova-api.logの使用量が増えているのか、減っているのか、安定しているのか、を知ることができます。

collectdのようなツールはこのような情報を保管することに利用できます。collectdはこの本のスコープから外れますが、collectdでCOUNTERデータ形として結果を保存するのがよい出発点になります。より詳しい情報はcollectdのドキュメント (https://collectd.org/wiki/index.php/Data_source)を参照してください。

第14章 バックアップとリカバリー

バックアップ対象 165
データベースのバックアップ 165
ファイルシステムバックアップ 166
バックアップのリカバリー 168

OpenStackバックアップポリシーを作成する際、標準的なバックアップのベストプラクティスが適用できます。例えば、どの程度の頻度でバックアップを行なうかは、どのくらい早くデータロスから復旧させる必要があるかに密接に関連しています。

注記

もし、いかなるデータロスも許されない場合、バックアップに加えて高可用性（High Avaialability）についても検討すべきです。

さらにバックアップの考慮点として以下があげられます。

- いくつのバックアップを持つべきか?
- オフサイトにバックアップを置くべきか?
- どの程度の頻度でバックアップをテストすべきか?

バックアップポリシーと同じくらい大事なことは、リカバリーポリシーです（少なくともリカバリーのテストは必要です）。

バックアップ対象

OpenStackは多くのコンポーネントから構成され、注意を払うべき箇所もたくさんありますが、大事なデータのバックアップは非常に単純です。

この章では、OpenStackコンポーネントを動作させるのに必要な設定ファイルとデータベースについてのバックアップ方法のみを説明します。オブジェクトストレージ内のオブジェクトや、ブロックストレージ内のデータのバックアップについては説明していません。一般的にこれらの領域はユーザー自身でバックアップを行います。

データベースのバックアップ

参考アーキテクチャーでは、クラウドコントローラーを MySQL サーバにしています。このMySQL サーバーは Nova, Glance, Cinder, そして

Keystone のデータベースを保持しています。全てのデータベースが一ヶ所にある場合、データベースバックアップは非常に容易となります。

```
# mysqldump --opt --all-databases >
                openstack.sql
```

もし、単一のデータベースのみバックアップする場合は次のように実行します。

```
# mysqldump --opt nova > nova.sql
```

ここで nova はバックアップ対象のデータベースです。

以下のようなcronジョブを一日に一度実行することで、簡単に自動化することも出来ます。

```
#!/bin/bash
backup_dir="/var/lib/backups/mysql"
filename="${backup_dir}/mysql-`hostname`-`eval date +%Y%m%d`.sql.gz"
# Dump the entire MySQL database
/usr/bin/mysqldump --opt --all-databases | gzip > $filename
# Delete backups older than 7 days
find $backup_dir -ctime +7 -type f -delete
```

このスクリプトは MySQL データベース全体をダンプし、7日間より古いバックアップを削除します。

ファイルシステムバックアップ

このセクションは、サービスにより構成される、定期的にバックアップすべきファイルとディレクトリについて議論します。

コンピュート

クラウドコントローラー、および、コンピュートノードの /etc/novaディレクトリは定期的にバックアップされるべきです。

/var/log/nova については、全てのログをリモートで集中管理しているのであれば、バックアップの必要はありません。ログ集約システムの導入か、ログディレクトリのバックアップを強く推奨します

/var/lib/nova がバックアップする他の重要なディレクトリです。これの例外がコンピュートノードにある /var/lib/nova/instances サブディレクトリです。このサブディレクトリには実行中のインスタンスの KVM イメージが置かれます。このディレクトリをバックアップしたいと思うのは、すべてのインスタンスのバックアップコピーを保持する必要がある場合だけでしょう。多くの場合において、これを実行する必要がありません。ただし、クラウドごとに異なり、サービスレベルによっても異

なる可能性があります。稼働中の KVM インスタンスのバックアップは、バックアップから復元したときでも、正しく起動しない可能性があることに気をつけてください。

イメージカタログと配布

/etc/glanceと/var/log/glanceはnovaの場合と同じルールに従います。

/var/lib/glanceもバックアップすべきです。 /var/lib/glance/imagesには特段の注意が必要です。もし、ファイルベースのバックエンドを利用しており、このディレクトリがイメージの保管ディレクトリならば特にです。

このディレクトリの永続性を保証するために二つの方法があります。一つ目はRAIDアレイ上にこのディレクトリを置くことで、ディスク障害時にもこのディレクトリが利用できます。二つ目の方法はrsyncのようなツールを用いてイメージを他のサーバーに複製することです。

```
# rsync -az --progress /var/lib/glance/images backup-server:/var/lib/glance/images/
```

認証

/etc/keystoneと/var/log/keystoneは他のコンポーネントと同じルールになります。

/var/lib/keystoneは、使用されるデータは含まれていないはずですが、念のためバックアップします。

ブロックストレージ

/etc/cinderと/var/log/cinderは他のコンポーネントと同じルールです。

/var/lib/cinderもまたバックアップされるべきです。

オブジェクトストレージ

/etc/swiftは非常に重要ですのでバックアップが必要です。このディレクトリには、Swiftの設定ファイル以外に、RingファイルやRingビルダーファイルが置かれています。これらのファイルを消失した場合はクラスター上のデータにアクセスできなくなります。ベストプラクティスとしては、ビルダーファイルを全てのストレージノードにringファイルと共

に置くことです。この方法でストレージクラスター上にバックアップコピーが分散されて保存されます。

バックアップのリカバリー

バックアップのリカバリーは単純です。始めにリカバリー対象のサービスが停止していることを確認します。例を挙げると、クラウドコントローラー上のnovaの完全リカバリーを行なう場合、最初に全ての nova サービスを停止します。

```
# stop nova-api
      # stop nova-cert
      # stop nova-consoleauth
      # stop nova-novncproxy
      # stop nova-objectstore
      # stop nova-scheduler
```

以前にバックアップしたデータベースをインポートします。

```
# mysql nova < nova.sql
```

同様に、バックアップした nova のディレクトリをリストアします。

```
# mv /etc/nova{,.orig}
      # cp -a /path/to/backup/nova /etc/
```

ファイルをリストア後、サービスを起動します。

```
# start mysql
      # for i in nova-api nova-cert nova-consoleauth nova-novncproxy nova-objectstore nova-scheduler
      > do
      > start $i
      > done
```

他のサービスもそれぞれのディレクトリとデータベース名で同じ処理となります。

第15章 カスタマイズ

DevStack 169
ミドルウェア例 173
Nova スケジューラーの例 179
ダッシュボード 184

OpenStack はあなたが必要とするすべてのことをしてくれるわけではないかもしれません。この場合、主に2つのやり方のうちのいずれかに従ってください。まず最初に、貢献するには (https://wiki.openstack.org/wiki/How_To_Contribute) を学び、Code Review Workflow (https://wiki.openstack.org/wiki/GerritWorkflow) に従って、あなたの修正を上流の OpenStack プロジェクトへコントリビュートしてください。もし、あなたが必要な機能が既存のプロジェクトと密にインテグレーションする必要がある場合、これが推奨される選択肢です。コミュニティは、いつでも貢献に対して開かれていますし、機能開発ガイドラインに従う新機能を歓迎します。

代替え案としては、もしあなたが必要とする機能が密なインテグレーションを必要としないのであれば、OpenStack をカスタマイズする他の方法があります。もし、あなたの機能が必要とされるプロジェクトが Python Paste フレームワークを使っているのであれば、そのための ミドルウェアを作成し、環境設定を通じて組み込めばよいのです。例えば OpenStack Compute の新しいスケジューラーや、カスタマイズされたダッシュボードを作成するといった、プロジェクトをカスタマイズする特定の方法もあるかもしれません。この章では、OpenStack をカスタマイズする後者の方法にフォーカスします。

OpenStack をこの方法でカスタマイズするためには、開発環境が必要です。開発環境を手軽に動作させる最良の方法は、クラウドの中で DevStack を動かすことです。

DevStack

ドキュメンテーションはすべて DevStack (http://devstack.org/) のウェブサイトにあります。 どのプロジェクトをカスタマイズしたいかによって、つまり Object Storage (swift) なのか他のプロジェクトなのかによって、DevStack の環境設定が異なります。以下のミドルウェアの例では、Object Storage を有効にしてインストールしなければなりません。

インスタンス上で、Folsom の安定版用の DevStack を動作させるためには：

1. ダッシュボード、または nova のコマンドラインインタフェース(CLI)から、以下のパラメータでインスタンスを起動してください。

   - 名前: devstack
   - イメージ: Ubuntu 12.04 LTS
   - メモリサイズ: 4 GB RAM（おそらく 2 GB でもなんとかなるでしょう）
   - ディスクサイズ: 最低 5 GB

   nova コマンドを使っているのであれば、適切なメモリ量とディスクサイズを得るために nova boot コマンドに --flavor 6 を指定してください。

2. 利用するイメージで root ユーザしか使えない場合、「stack」ユーザを作成しなければなりません。でなければ、stack.sh スクリプトに「stack」ユーザを作成させる時に、screen に関するパーミッションの問題にぶつかります。利用するイメージに、既に root 以外のユーザがあるのであれば、このステップは省略できます。

   a. ssh root@<IP Address>
   b. adduser --gecos "" stack
   c. プロンプトに対して新しいパスワードを入力します。
   d. adduser stack sudo
   e. grep -q "^#includedir.*/etc/sudoers.d" /etc/sudoers || echo "#includedir /etc/sudoers.d" >> /etc/sudoers
   f. (umask 226 && echo "stack ALL=(ALL) NOPASSWD:ALL" > /etc/sudoers.d/50_stack_sh)
   g. exit

3. stack ユーザとしてログインし、DevStack の設定を行います。

   a. ssh stack@<IP address>

b. プロンプトに対して、stack ユーザに作ったパスワードを入力します。

c. sudo apt-get -y update

d. sudo apt-get -y install git

e. git clone https://github.com/openstack-dev/devstack.git -b stable/folsom devstack/

f. cd devstack

g. vim localrc

- Swift のみ、ミドルウェア例 で使用された, 以下の [1] Swift only localrc の例を参照してください。
- 他のすべてのプロジェクトについて、 Nova Scheduler Example で使用された, 以下の [2] All other projects localrc の例を参照してください。

h. ./stack.sh

i. screen -r stack

注記

- stack.sh の実行には、しばらく時間がかかります。できれば、この時間を使って OpenStack ファウンデーションに参加してください (http://www.openstack.org/join/)。
- stack.sh を実行する際、“ERROR: at least one RPC back-end must be enabled” というエラーメッセージが出るかもしれません。これは心配しないでください。swift と keystone はRPC (AMQP) バックエンドを必要としないのです。同様に、ImportErrors はすべて無視できます。
- Screen は、多くの関連するサービスを同時に見るための便利なプログラムです。GNU screen quick reference. (http://aperiodic.net/screen/quick_reference) を参照してください。

以上で OpenStack の開発環境を準備できましたので、運用環境にダメージを与えることを心配せずに自由にハックできます。Swift のみの環境では ミドルウェア例 に、‘他のすべてのプロジェクトでは Nova Scheduler Example に進んでください。

[1] Swift only localrc

```
ADMIN_PASSWORD=devstack
MYSQL_PASSWORD=devstack
RABBIT_PASSWORD=devstack
SERVICE_PASSWORD=devstack
SERVICE_TOKEN=devstack

SWIFT_HASH=66a3d6b56c1f479c8b4e70ab5c2000f5
SWIFT_REPLICAS=1

# Uncomment the BRANCHes below to use stable versions


# unified auth system (manages accounts/tokens)
KEYSTONE_BRANCH=stable/folsom
# object storage
SWIFT_BRANCH=stable/folsom

disable_all_services
enable_service key swift mysql
```

[2] All other projects localrc

```
ADMIN_PASSWORD=devstack
MYSQL_PASSWORD=devstack
RABBIT_PASSWORD=devstack
SERVICE_PASSWORD=devstack
SERVICE_TOKEN=devstack

FLAT_INTERFACE=br100
PUBLIC_INTERFACE=eth0

VOLUME_BACKING_FILE_SIZE=20480M

# For stable versions, look for branches named stable/[milestone].

# compute service
NOVA_BRANCH=stable/folsom

# volume service
CINDER_BRANCH=stable/folsom

# image catalog service
GLANCE_BRANCH=stable/folsom

# unified auth system (manages accounts/tokens)
KEYSTONE_BRANCH=stable/folsom

# django powered web control panel for openstack
HORIZON_BRANCH=stable/folsom
```

ミドルウェア例

ほとんどの OpenStack プロジェクトは Python Paste(http://pythonpaste.org/) フレームワークに基づいています。A Do-It-

Yourself Framework (http://pythonpaste.org/do-it-yourself-framework.html) は、このアーキテクチャの最良の紹介です。このフレームワークを使っているため、コードに一切手をいれずに、プロジェクトの処理パイプラインになんらかのカスタム・コードを入れて機能追加を行うことができます。

このように OpenStack をカスタマイズすることを説明するため、Swift用に、あるコンテナに対して、そのコンテナのメタデータで指定される一群のIPアドレスのみからアクセスできるようにするミドルウェアを作成してみます。このような例は多くの状況で有用です。例えば、一般アクセス可能なコンテナを持っていますが、実際に行いたいことはホワイトリストに基づいた一群のIPアドレスにのみ制限したい場合です。

警告

この例は実証目的のみのためにあります。さらなる作りこみと広範なセキュリティテストなしにコンテナのIPホワイトリスト・ソリューションとして使用するべきではありません。

stack.sh が screen -r stack で作成したセッションに join すると、Swift インストール用の localrc を使ったのであれば、3つの screen セッションが見えます。

```
0$ shell*  1$ key  2$ swift
```

*（アスタリスク)は、どの screen にいるのかを示します。

- 0$ shell. 何か作業することができる shell セッションです。
- 1$ key. keystone サービス。
- 2$ swift. swift プロキシサービス。

ミドルウェアを作成して Paste の環境設定を通して組み込むためには：

1. すべての OpenStack のコードは /opt/stack にあります。shell セッションの screen の中で swift ディレクトリに移動し、あなたのミドルウェアモジュールを編集してください。

   a. cd /opt/stack/swift

   b. vim swift/common/middleware/ip_whitelist.py

2. 以下のコードをコピーしてください。作業が終わったら、ファイルを保存して閉じてください。

   ```
   # Licensed under the Apache License, Version 2.0 (the "License");
   ```

```
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#

#    http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
# implied.
# See the License for the specific language governing permissions and
# limitations under the License.
import socket

from swift.common.utils import get_logger
from swift.proxy.controllers.base import get_container_info
from swift.common.swob import Request, Response

class IPWhitelistMiddleware(object):
    """
    IP Whitelist Middleware

    Middleware that allows access to a container from only a set of IP
    addresses as determined by the container's metadata items that start
    with the prefix 'allow'. E.G. allow-dev=192.168.0.20
    """

    def __init__(self, app, conf, logger=None):
        self.app = app

        if logger:
            self.logger = logger
        else:
            self.logger = get_logger(conf, log_route='ip_whitelist')

        self.deny_message = conf.get('deny_message', "IP Denied")
        self.local_ip = socket.gethostbyname(socket.gethostname())

    def __call__(self, env, start_response):
        """
        WSGI entry point.
        Wraps env in swob.Request object and passes it down.

        :param env: WSGI environment dictionary
        :param start_response: WSGI callable
        """
        req = Request(env)

        try:
            version, account, container, obj = req.split_path(1, 4, True)
        except ValueError:
            return self.app(env, start_response)

        container_info = get_container_info(
            req.environ, self.app, swift_source='IPWhitelistMiddleware')

        remote_ip = env['REMOTE_ADDR']
        self.logger.debug(_("Remote IP: %(remote_ip)s"),
```

```
                        {'remote_ip': remote_ip})

        meta = container_info['meta']
        allow = {k:v for k,v in meta.iteritems() if k.startswith('allow')}
        allow_ips = set(allow.values())
        allow_ips.add(self.local_ip)
        self.logger.debug(_("Allow IPs: %(allow_ips)s"),
                          {'allow_ips': allow_ips})

        if remote_ip in allow_ips:
            return self.app(env, start_response)
        else:
            self.logger.debug(
                _("IP %(remote_ip)s denied access to Account=%(account)s "
                  "Container=%(container)s. Not in %(allow_ips)s"), locals())
            return Response(
                status=403,
                body=self.deny_message,
                request=req)(env, start_response)


def filter_factory(global_conf, **local_conf):
    """
    paste.deploy app factory for creating WSGI proxy apps.
    """
    conf = global_conf.copy()
    conf.update(local_conf)

    def ip_whitelist(app):
        return IPWhitelistMiddleware(app, conf)
    return ip_whitelist
```

env と conf には、リクエストについて何をするのか判断するのに使える有用な情報が多数含まれています。どんなプロパティが利用可能なのかを知るには、以下のログ出力文を __init__ メソッドに挿入してください。

```
self.logger.debug(_("conf = %(conf)s"), locals())
```

そして以下のログ出力分を __call__ メソッドに挿入してください。

```
self.logger.debug(_("env = %(env)s"), locals())
```

3. このミドルウェアを Swift のパイプラインに組み込むには、設定ファイルを1つ編集する必要があります。

```
vim /etc/swift/proxy-server.conf
```

4. [filter:ratelimit] セクションを探し、以下の環境定義セクションを貼り付けてください。

```
[filter:ip_whitelist]
paste.filter_factory = swift.common.middleware.ip_whitelist:filter_factory
# You can override the default log routing for this filter here:
# set log_name = ratelimit
# set log_facility = LOG_LOCAL0
# set log_level = INFO
# set log_headers = False
# set log_address = /dev/log
deny_message = You shall not pass!
```

5. [pipeline:main] セクションを探し、このように ip_whitelist リストを追加してください。完了したら、ファイルを保存して閉じてください。

```
[pipeline:main]
pipeline = catch_errors healthcheck cache ratelimit ip_whitelist authtoken
 keystoneauth proxy-logging proxy-server
```

6. Swift にこのミドルウェアを使わせるために、Swift プロキシサービスを再起動します。swift の screen セッションに切り替えてはじめてください。

   a. Ctrl-A の後で 2 を押します。ここで、2 は screen セッションのラベルです。Ctrl-A の後で n を押し、次の screen セッションに切り替えることもできます。

   b. Ctrl-C を押し、サービスを終了させます。

   c. 上矢印キーを押し、最後のコマンドを表示させます。

   d. Enter キーを押し、実行します。

7. Swift の CLI でミドルウェアのテストをしてください。shell の screen セッションに切り替えてテストを開始し、swift の screen セッションにもどってログ出力をチェックして終了します。

   a. Ctrl-A の後で 0 を押します。

   b. cd ~/devstack

   c. source openrc

   d. swift post middleware-test

   e. Ctrl-A の後で 2 を押します。

8. ログの中に以下の行があるでしょう。

```
proxy-server ... IPWhitelistMiddleware
proxy-server Remote IP: 203.0.113.68 (txn: ...)
proxy-server Allow IPs: set(['203.0.113.68']) (txn: ...)
```

基本的に、最初の３行はこのミドルウェアが Swift の他のサービスとやりとりする際に、再度認証を行う必要がないことを示しています。最後の２行は、このミドルウェアによって出力されており、リクエストが DevStack インスタンスから送られており、許可されていることを示しています。

9. DevStack 環境の外の、DevStack 用インスタンスにアクセス可能なリモートマシンからミドルウェアをテストします。

   a. swift --os-auth-url=http://203.0.113.68:5000/v2.0/ --os-region-name=RegionOne --os-username=demo:demo --os-password=devstack list middleware-test

   b. Container GET failed: http://203.0.113.68:8080/v1/AUTH_.../middleware-test?format=json 403 Forbidden You shall not pass!

10. 再び Swift のログをチェックすると、以下の行が見つかるでしょう。

```
proxy-server Invalid user token - deferring reject downstream
proxy-server Authorizing from an overriding middleware (i.e: tempurl) (txn: ...)
proxy-server ... IPWhitelistMiddleware
proxy-server Remote IP: 198.51.100.12 (txn: ...)
proxy-server Allow IPs: set(['203.0.113.68']) (txn: ...)
proxy-server IP 198.51.100.12 denied access to Account=AUTH_... Container=None.
 Not in set(['203.0.113.68']) (txn: ...)
```

ここで、リモートIPアドレスが、許可されたIPアドレスの中になかったため、リクエストが拒否されていることがわかります。

11. DevStack用インスタンスに戻り、リモートマシンからのリクエストを許可するようなコンテナのメタデータを追加します。

    a. Ctrl-A の後で 0 を押します。

    b. swift post --meta allow-dev:198.51.100.12 middleware-test

12. ステップ 9 [178] に記載したコマンドをもう一度試すと、今度は成功します。

このような機能試験は、正しいユニットテストと結合テストの代わりになるものではありませんが、作業を開始することはできます。

Python Paste フレームワークを使う他のすべてのプロジェクトで、類似のパターンに従うことができます。単純にミドルウェアモジュールを作成し、環境定義によって組み込んでください。そのミドルウェアはプロジェクトのパイプラインの一部として順番に実行され、必要に応じて他のサービスを呼び出します。プロジェクトのコア・コードは一切修正しません。Paste を使っているプロジェクトを確認するには、/etc/<project> に格納されている、プロジェクトの conf または ini 環境定義ファイルの中で pipeline 変数を探してください。

あなたのミドルウェアが完成したら、オープンソースにし、OpenStack メーリングリストでコミュニティに知らせることを薦めます。コミュニティの人々はあなたのコードを使い、フィードバックし、おそらくコントリビュートするでしょう。もし十分な支持があれば、公式なSwift ミドルウェア（https://github.com/openstack/swift/tree/master/swift/common/middleware）に追加するように提案することもできるでしょう。

Nova スケジューラーの例

多くの OpenStack のプロジェクトでは、ドライバ・アーキテクチャを使うことによって、特定の機能をカスタマイズすることができます。特定のインターフェースに適合するドライバを書き、環境定義によって組み込むことができます。例えば、簡単に nova に新しいスケジューラーを組み込むことができます。nova の既存のスケジューラーは、フル機能であり、スケジューリング（http://docs.openstack.org/trunk/config-reference/content/section_compute-scheduler.html）によくドキュメントされています。しかし、あなたのユーザのユース・ケースに依存して、既存のスケジューラで要件が満たせないかもしれません。この場合は、新しいスケジューラーを作成する必要があるでしょう。

スケジューラーを作成するには、nova.scheduler.driver.Scheduler クラスを継承しなければなりません。オーバーライド可能な５つのメソッドのうち、以下の「*」で示される２つのメソッドをオーバーライドしなければなりません。

- update_service_capabilities
- hosts_up
- schedule_live_migration
- * schedule_prep_resize
- * schedule_run_instance

OpenStack のカスタマイズをデモするために、リクエストの送信元IPアドレスとホスト名のプレフィックスに基づいてインスタンスを一部のホ

ストにランダムに配置するようなNova のスケジューラーの例を作成します。この例は、１つのユーザのグループが１つのサブネットにおり、インスタンスをホスト群の中の一部のサブネットで起動したい場合に有用です。

注記

この例は実証目的のみのためにあります。さらなる作りこみと広範なテストなしにNovaのスケジューラーとして使用するべきではありません。

stack.sh が screen -r stack で作成したセッションに join すると、多数の screen セッションが見えます。

```
0$ shell*  1$ key  2$ g-reg  3$ g-api  4$ n-api  5$ n-cpu  6$ n-crt  7$ n-net  8-$ n-sch ...
```

- 0$ shell. 何か作業することができる shell セッションです。
- 1$ key. keystone サービス。
- g-*. glance サービス。
- n-*. nova サービス。
- n-sch。nova スケジューラーサービス。

スケジューラーを作成して、設定を通して組み込むためには：

1. OpenStack のコードは /opt/stack にあるので、nova ディレクトリに移動してあなたのスケジューラーモジュールを編集します。

   a. cd /opt/stack/nova

   b. vim nova/scheduler/ip_scheduler.py

2. 以下のコードをコピーしてください。作業が終わったら、ファイルを保存して閉じてください。

```
# vim: tabstop=4 shiftwidth=4 softtabstop=4
# Copyright (c) 2013 OpenStack Foundation
# All Rights Reserved.
#
#    Licensed under the Apache License, Version 2.0 (the "License"); you may
#    not use this file except in compliance with the License. You may obtain
#    a copy of the License at
#
#   http://www.apache.org/licenses/LICENSE-2.0
#
#    Unless required by applicable law or agreed to in writing, software
#    distributed under the License is distributed on an "AS IS" BASIS, WITHOUT
#    WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the
#    License for the specific language governing permissions and limitations
#    under the License.
```

```
"""
IP Scheduler implementation
"""

import random

from nova import exception
from nova.openstack.common import log as logging
from nova import flags
from nova.scheduler import driver

FLAGS = flags.FLAGS
LOG = logging.getLogger(__name__)


class IPScheduler(driver.Scheduler):
    """
    Implements Scheduler as a random node selector based on
    IP address and hostname prefix.
    """

    def _filter_hosts(self, hosts, hostname_prefix):
        """Filter a list of hosts based on hostname prefix."""

        hosts = [host for host in hosts if host.startswith(hostname_prefix)]
        return hosts

    def _schedule(self, context, topic, request_spec, filter_properties):
        """
        Picks a host that is up at random based on
        IP address and hostname prefix.
        """

        elevated = context.elevated()
        hosts = self.hosts_up(elevated, topic)

        if not hosts:
            msg = _("Is the appropriate service running?")
            raise exception.NoValidHost(reason=msg)

        remote_ip = context.remote_address

        if remote_ip.startswith('10.1'):
            hostname_prefix = 'doc'
        elif remote_ip.startswith('10.2'):
            hostname_prefix = 'ops'
        else:
            hostname_prefix = 'dev'

        hosts = self._filter_hosts(hosts, hostname_prefix)
        host = hosts[int(random.random() * len(hosts))]

        LOG.debug(_("Request from %(remote_ip)s scheduled to %(host)s")
                  % locals())

        return host

    def schedule_run_instance(self, context, request_spec,
                              admin_password, injected_files,
                              requested_networks, is_first_time,
                              filter_properties):
        """Attempts to run the instance"""
        instance_uuids = request_spec.get('instance_uuids')
        for num, instance_uuid in enumerate(instance_uuids):
            request_spec['instance_properties']['launch_index'] = num
            try:
                host = self._schedule(context, 'compute', request_spec,
```

```
                                    filter_properties)
                updated_instance = driver.instance_update_db(context,
                                                             instance_uuid)
                self.compute_rpcapi.run_instance(context,
                                                 instance=updated_instance, host=host,
                                                 requested_networks=requested_networks,
                                                 injected_files=injected_files,
                                                 admin_password=admin_password,
                                                 is_first_time=is_first_time,
                                                 request_spec=request_spec,
                                                 filter_properties=filter_properties)
            except Exception as ex:
                # NOTE(vish): we don't reraise the exception here to make sure
                # that all instances in the request get set to
                # error properly
                driver.handle_schedule_error(context, ex, instance_uuid,
                                             request_spec)

    def schedule_prep_resize(self, context, image, request_spec,
                             filter_properties, instance, instance_type,
                             reservations):
        """Select a target for resize."""
        host = self._schedule(context, 'compute', request_spec,
                              filter_properties)
        self.compute_rpcapi.prep_resize(context, image, instance,
                                        instance_type, host, reservations)
```

context と request_spec と filter_propertiesには、どこにインスタンスをスケジュールするのか決定するのに使える有用な情報が多数含まれています。どんなプロパティが利用可能なのかを知るには、以下のログ出力文を上記の schedule_run_instance メソッドに挿入してください。

```
LOG.debug(_("context = %(context)s") % {'context': context.__dict__})LOG.debug(_("request_spec
 = %(request_spec)s") % locals())LOG.debug(_("filter_properties = %(filter_properties)s") %
 locals())
```

3. このスケジューラーを Nova に組み込むには、設定ファイルを1つ編集する必要があります。

```
LOG$ vim /etc/nova/nova.conf
```

4. compute_scheduler_driver 設定を見つけ、このように変更してください。

```
LOGcompute_scheduler_driver=nova.scheduler.ip_scheduler.IPScheduler
```

5. Nova にこのスケジューラーを使わせるために、Nova スケジューラーサービスを再起動します。 n-sch screen セッションに切り替えてはじめてください。

   a. Ctrl-A の後で 8 を押します。

   b. Ctrl-C を押し、サービスを終了させます。

   c. 上矢印キーを押し、最後のコマンドを表示させます。

d. Enter キーを押し、実行します。

6. Nova の CLI でスケジューラーのテストをしてください。shell の screen セッションに切り替えてテストを開始し、n-sch screen セッションにもどってログ出力をチェックして終了します。

   a. Ctrl-A の後で 0 を押します。

   b. cd ~/devstack

   c. source openrc

   d. IMAGE_ID=`nova image-list | egrep cirros | egrep -v "kernel|ramdisk" | awk '{print $2}'`

   e. nova boot --flavor 1 --image $IMAGE_ID scheduler-test

   f. Ctrl-A の後で 8 を押します。

7. ログの中に以下の行があるでしょう。

```
LOG2013-02-27 17:39:31 DEBUG nova.scheduler.ip_scheduler [req-... demo demo] Request from 50.
56.172.78 scheduled to
devstack-nova from (pid=4118) _schedule /opt/stack/nova/nova/scheduler/ip_scheduler.py:73
```

このような機能試験は、正しいユニットテストと結合テストの代わりになるものではありませんが、作業を開始することはできます。

ドライバ・アーキテクチャを使う他のすべてのプロジェクトで、類似のパターンに従うことができます。単純に、そのドライバ・インタフェースに従うモジュールとクラスを作成し、環境定義によって組み込んでください。あなたのコードはその機能が使われた時に実行され、必要に応じて他のサービスを呼び出します。プロジェクトのコア・コードは一切修正しません。ドライバ・アーキテクチャを使っているプロジェクトを確認するには、/etc/<project> に格納されている、プロジェクトの環境定義ファイルの中で「driver」変数を探してください。

あなたのスケジューラーが完成したら、オープンソースにし、OpenStack メーリングリストでコミュニティに知らせることをお薦めします。もしかしたら他の人も同じ機能を必要としているかもしれません。彼らはあなたのコードを使い、フィードバックし、おそらくコントリビュートするでしょう。もし十分な支持があれば、もしかしたら公式なNova スケジューラー（https://github.com/openstack/nova/tree/master/nova/scheduler）への追加を提案してもよいでしょう。

ダッシュボード

ダッシュボードは、Python Django (https://www.djangoproject.com/) Webアプリケーションフレームワークに基づいています。カスタマイズのための最良のガイドは既に執筆されており、 Build on Horizon (http://docs.openstack.org/developer/horizon/topics/tutorial.html) にあります。

第16章 OpenStack コミュニティ

助けを得る 185
バグ報告 186
OpenStack コミュニティに参加する 189
機能と開発ロードマップ 190
ドキュメント作成への貢献の仕方 192
セキュリティ情報 193
さらに情報を見つける 194

OpenStack は非常に活発なコミュニティの上に成り立っており、あなたの参加をいつでも待っています。この節では、コミュニティの他の人と交流する方法について詳しく説明します。

助けを得る

助けてもらうにはいくつかの方法があります。コミュニティの助けを得るのが一番早い方法です。 Q&A サイト、メーリングリストのアーカイブを検索したり、バグリストにあなたが抱えている問題に似た問題がないか探します。必要な情報が見つからなかった場合、この後の節で説明する手順にしたがってバグ報告を行ったり、以下のサポートチャネルのどれかを使うことになります。

最初に確認すべき場所は OpenStack の公式ドキュメントです。 http://docs.openstack.org にあります。

ask.openstack.org では、すでに回答されている質問を見ることができます。

メーリングリスト (https://wiki.openstack.org/wiki/Mailing_Lists) もアドバイスをもらうのに素晴らしい場所です。 Wiki ページにメーリングリストの詳しい情報が載っています。運用者としては、確認しておくべき主要なメーリングリストは以下です。

- 一般向けメーリングリスト: openstack@lists.openstack.org. このリストでは、OpenStackの現行リリースに関する話題を扱います。流量は非常に多く、1日にとてもたくさんのメールが流れます。
- 運用者向けリスト: openstack-operators@lists.openstack.org. このリストは、あなたのように、実際にOpenStackクラウドの運用者間での議論を目的としています。現在のところ、メールの流量はかなり少なく、１日に１通といったレベルです。

- 開発者向けリスト: openstack-dev@lists.openstack.org. このリストでは、OpenStackの今後についての話題を扱います。流量は多く、1日に何通もメールが流れます。

一般向けリストと運用者向けリストを購読するのがお薦めです。一般向けリストの流量に対応するにはフィルタを設定する必要があるでしょうが。 Wiki のメーリングリストのページにはメーリングリストのアーカイブへのリンクがあり、そこで議論の過程を検索することができます。

一般的な質問用や開発者の議論用など、複数の IRC チャネル (https://wiki.openstack.org/wiki/IRC) があります。一般的な議論用のチャネルは irc.freenode.net の #openstackです。

バグ報告

運用者として、あなたは、あなたのクラウドでの予期しない動作を報告できる非常によい立場にいます。 OpenStack は柔軟性が高いので、ある特定の問題を報告するのはあなた一人かもしれません。すべての問題は重要で修正すべきものです。そのためには、簡単にバグ報告を登録する方法を知っておくことは欠かせません。

全ての OpenStack プロジェクトでは、バグ管理に Launchpad を使っています。バグ報告を行う前に、Launchpad のアカウントを作る必要があります。

Launchpad のアカウントを作った後は、バグを報告するのは、問題の原因となったプロジェクトを特定するのと同じくらい簡単です。場合によっては、思っていたよりも難しいこともありますが、最初のバグ報告が適切な場所でなかったとしても、バグの分類を行う人がバグ報告を適切なプロジェクトに割り当て直してくれます。

- Nova のバグ報告 (https://bugs.launchpad.net/nova/+filebug)
- python-novaclient のバグ報告 (https://bugs.launchpad.net/python-novaclient/+filebug)
- Swift のバグ報告 (https://bugs.launchpad.net/swift/+filebug)
- python-swiftclient のバグ報告(https://bugs.launchpad.net/python-swiftclient/+filebug)
- Glance のバグ報告 (https://bugs.launchpad.net/glance/+filebug)
- python-glanceclient のバグ報告(https://bugs.launchpad.net/python-glanceclient/+filebug)

- Keystone のバグ報告（https://bugs.launchpad.net/keystone/+filebug）
- python-keystoneclient のバグ報告(https://bugs.launchpad.net/python-keystoneclient/+filebug)
- Neutron のバグ報告（https://bugs.launchpad.net/neutron/+filebug）
- python-neutronclient のバグ報告(https://bugs.launchpad.net/python-neutronclient/+filebug)
- Cinder のバグ報告（https://bugs.launchpad.net/cinder/+filebug）
- python-cinderclient のバグ報告(https://bugs.launchpad.net/python-cinderclient/+filebug)
- Horizon のバグ報告（https://bugs.launchpad.net/horizon/+filebug）
- ドキュメントdocumentation に関するバグ報告（http://bugs.launchpad.net/openstack-manuals/+filebug）
- API ドキュメント に関するバグ報告（http://bugs.launchpad.net/openstack-api-site/+filebug）

よいバグ報告を書くには、次に述べる手順を踏むことが不可欠です。まず、バグを検索し、同じ問題に関してすでに登録されているバグがないことを確認します。見つかった場合は、「This bug affects X people. Does this bug affect you?」（このバグは X 人に影響があります。あなたもこのバグの影響を受けますか?）をクリックするようにして下さい。同じ問題が見つからなかった場合は、バグ報告で詳細を入力して下さい。少なくとも以下の情報を含めるべきです。

- 実行しているソフトウェアのリリースやマイルストーン、コミット ID。
- あなたがバグを確認したオペレーティングシステムとそのバージョン。
- バグの再現手順。何がおかしいかも含めて下さい。
- 期待される結果の説明（出くわした現象ではなく）。
- ログファイルから関連部分のみを抜粋したもの。

バグ報告をすると、バグは次のステータスで作成されます。

- Status: New

バグのコメントでは、既知のバグの修正方法に関して案を出したり、バグの状態を Triaged（有効な報告か、対応が必要かなどを分類した状態）にセットすることができます。バグを直接修正することもできます。その場合は、バグを自分に割り当てて、状態を In progress（進行中）にセットし、コードのブランチを作り、トランクにマージしてもらうように変更を提案するという流れになります。でも、先走りし過ぎないようにしましょう。バグを分類するという仕事もありますから。

バグの確認と優先付け

このステップでは、バグが実際に起こるかのチェックやその重要度の判定が行われます。これらのステップを行うには、バグ管理者権限が必要なものがあります（通常、バグ管理者権限はコア開発者チームだけが持っています）。バグ報告に、バグを再現したり、バグの重要度を判定したりするのに必要な情報が不足している場合、バグは

- Status: Incomplete

にセットされます。問題を再現できた場合（もしくはこの報告がバグであると 100% 確信を持てる場合）で、セットする権限を持っている場合には、%c_dummy

- Status: Confirmed

にセットして下さい。

- Importance: <バグ影響度>

バグ影響度は以下のカテゴリに分かれています。

1. Critical このバグにより、目玉となる機能が全てのユーザで正常に動作しない場合（または簡単なワークアラウンドでは動作しない場合）、もしくはデータロスにつながるような場合

2. High このバグにより、目玉となる機能が何人かユーザで正常に動作しない場合（または簡単なワークアラウンドで動作する場合）

3. Medium このバグにより、ある程度重要な機能が正常に動作しない場合

4. Low このバグが多くの場合で軽微な場合

5. Wishlist 実際にはバグではないが、そのように動作を変更した方よいものの場合

バグ報告に解決方法やパッチが書かれている場合、そのバグの状態は Triaged にセットされる。

バグ修正

この段階では、開発者が修正を行います。修正を行っている間、作業の重複を避けるため、バグの状態を以下のようにセットすべきです。

- Status: In progress
- Assignee: <あなた自身>

修正が用意できたら、開発者は変更を提案し、レビューを受けます。

変更が受理された後

変更がレビューされ、受理されて、マスターブランチにマージされると、バグの状態は自動的に以下のようになります。

- Status: Fix committed

修正がマイルストーンやリリースブランチに取り込まれると、バグの状態は自動的に以下のようになります。

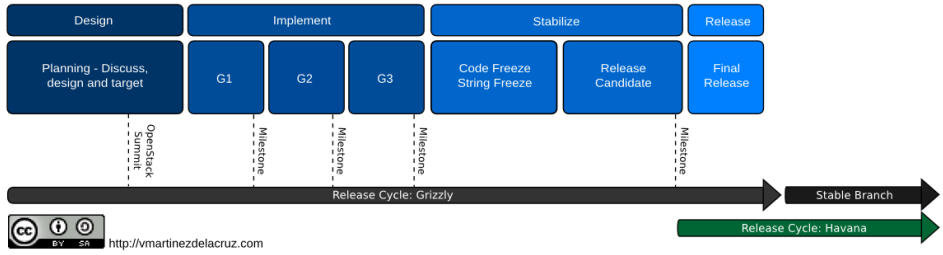
- Milestone: このバグの修正が入ったマイルストーン
- Status: Fix released

OpenStack コミュニティに参加する

この本をここまで読んで来たので、あなたはコミュニティの正式な個人メンバーになって、Join The OpenStack Foundation (https://www.openstack.org/join/) に加入することを考えていることでしょう。OpenStack Foundation は、共有リソースを提供し OpenStack の目的の達成を支援する独立組織で、OpenStack ソフトウェア、およびユーザ、開発者、エコシステム全体といった OpenStack を取り巻くコニュニティを守り、活力を与え、普及の促進を行います。我々全員がこのコミュニティをできる限りよいものにしていく責任を共有します。メンバーになることはコミュニティに参加する最初のステップです。ソフトウェア同様、 OpenStack Foundation の個人会員は無料で誰でもなることができます。

機能と開発ロードマップ

OpenStack は6ヶ月のリリースサイクルを取っており、通常は4月と10月にリリースが行われます。各リリースサイクルの最初に、OpenStack コミュニティは一ヶ所に集まりデザインサミットを行います。サミットでは、次のリリースでの機能が議論され、優先度付けと計画が行われます。以下はリリースサイクルの一例で、サミットが行われて以降のマイルストーンリリース、Code Freeze、String Freeze などが記載されています。マイルストーンはリリースサイクル内での中間リリースで、テスト用にパッケージが作成され、ダウンロードできるようになります。Code Freeze では、そのリリースに向けての新機能の追加が凍結されます。String Freeze は、ソースコード内の文字列の変更が凍結されることを意味します。



機能追加リクエストは、通常 Etherpad で始まります。Etherpad は共同編集ツールで、デザインサミットのその機能に関するセッションで議論を整理するのに使われます。続けて、プロジェクトの Launchpad サイトに blueprint が作成され、blueprint を使ってよりきちんとした形で機能が規定されていきます。 この後、blueprint はプロジェクトメンバーによって承認され、開発が始まります。

あなたの機能追加リクエストを新機能として検討してもらうのに一番早い方法は、アイデアを書いた Etherpad を作成し、デザインサミットのセッションを提案することです。デザインサミットが終わっている場合には、blueprint を直接作成することもできます。 Victoria Martínez の blueprint での開発の進め方についてのブログ (http://vmartinezdelacruz.com/how-to-work-with-blueprints-without-losing-your-mind/) を是非読んで下さい。 OpenStack 開発者のインターンの視点で書かれた記事です。

次のリリースに向けたロードマップと開発状況は Releases (http://status.openstack.org/release/) で見ることができます。

将来のリリースで検討されている機能を確認したり、過去に実装された機能を見るには、OpenStack Compute (nova) Blueprints (https://

blueprints.launchpad.net/nova), OpenStack Identity (keystone) Blueprints (https://blueprints.launchpad.net/keystone) などの既存の Blueprint やリリースノートを見て下さい。

リリースノートは OpenStack Wiki で管理されています。

シリーズ	状態	リリース番号	リリース日
Grizzly	開発中、リリーススケジュール	リリース予定	2013年4月4日
Folsom	現在の安定版リリース、セキュリティアップデート対象	2012.2	2012年9月27日
		2012.2.1	2012年11月29日
		2012.2.2	2012年12月13日
		2012.2.3	2012年1月31日
Essex	コミュニティによるサポートが行われている、セキュリティアップデート対象	2012.1	2012年4月5日
		2012.1.1	2012年6月22日
		2012.1.2	2012年8月10日
		2012.1.3	2012年10月12日
Diablo	コミュニティによるサポートが行われている	2011.3	2011年9月22日
		2011.3.1	2012年1月19日
Cactus	非推奨	2011.2	2011年4月15日
Bexar	非推奨	2011.1	2011年2月3日
Austin	非推奨	2010.1	2010年10月21日

ドキュメント作成への貢献の仕方

OpenStack のドキュメント作成活動は、運用管理ドキュメント、API ドキュメント、ユーザドキュメントなどに渡ります。

この本の元は人が集まったイベントで作成されましたが、今やこの本はみなさんも貢献できる状態になっています。 OpenStack のドキュメント作成は、バグ登録、調査、修正を繰り返し行うというコーディングの基本原則に基いて行われています。

コードと全く同じく、docs.openstack.org サイトは Gerrit レビューシステムを使って常に更新されています。ソースは、GitHub の openstack-manuals (http://github.com/openstack/openstack-manuals/) レポジトリと api-site (http://github.com/openstack/api-site/) レポジトリに、それぞれ DocBook 形式で保存されています。

公開される前にドキュメントのレビューを行うには、OpenStack Gerrit サーバー review.openstack.org に行って、project:openstack/openstack-manuals や project:openstack/api-site を検索して下さい。

ドキュメントのレビューや変更を最初に行うのに必要な手続きについての詳しい情報は、Wiki の How To Contribute (https://

wiki.openstack.org/wiki/How_To_Contribute）ページを参照して下さい。

セキュリティ情報

我々は、コミュニティとして、セキュリティを非常に重要だと考えており、潜在的な問題の報告は決められたプロセスに基いて処理されます。修正を用心深く追跡し、定期的にセキュリティ上の問題点を取り除きます。あなたは発見したセキュリティ上の問題をこの決められたプロセスを通して報告できます。OpenStack 脆弱性管理チームは、OpenStackコミュニティから選ばれた脆弱性管理の専門家で構成されるごく少人数のグループです。このチームの仕事は、脆弱性の報告を手助けし、セキュリティ上の修正を調整し、脆弱性情報の公開を続けていくことです。特に、このチームは以下の役割を担っています。

- 脆弱性管理: コミュニティメンバー（やユーザ）が発見した全ての脆弱性をこのチームに報告できます。
- 脆弱性追跡: このチームはバグ追跡システムに登録された脆弱性に関連する問題の管理を行います。問題の中には、このチームと影響を受けるプロジェクトの責任者のみが参照できる場合もありますが、問題の修正が用意されると、全ての脆弱性は公開されます。
- Responsible Disclosure（責任ある情報公開）: セキュリティコミュニティと仕事をする義務の一つとして、セキュリティ情報の公開が、確実に、OpenStack のセキュリティ問題を責任をもって扱うセキュリティ研究者の適切なお墨付きを得て行われるようにします。

OpenStack 脆弱性管理チームに問題を報告する方法は2種類用意されており、問題の重要度に応じて使い分けて下さい。

- Launchpad でバグ報告を行い、'security bug' のマークを付ける。マークを付けると、そのバグは一般には公開されなくなり、脆弱性管理チームだけが参照できるようになります。
- 問題が極めて慎重に扱うべき情報の場合は、脆弱性管理チームのメンバーの誰かに暗号化したメールを送って下さい。メンバーの GPG 鍵は OpenStack Security（http://www.openstack.org/projects/openstack-security/）で入手できます。

あなたが参加できるセキュリティ関連のチームのリストは セキュリティチーム（http://wiki.openstack.org/SecurityTeams）にあります。脆弱性管理プロセスはすべてドキュメントにまとめられており、脆弱性管理（https://wiki.openstack.org/wiki/VulnerabilityManagement）で参照できます。

さらに情報を見つける

この本以外にも、OpenStack に関する情報源はたくさんあります。OpenStack ウェブサイト（http://www.openstack.org）は最初に見るといいでしょう。ここには、OpenStack ドキュメント（http://docs.openstack.org）と OpenStack API ドキュメント（http://api.openstack.org）があり、OpenStack に関する技術情報が提供されています。 OpenStack wiki には、OpenStack の各プロジェクトの範囲にとどらまらない汎用的な情報が多数あり、例えば お薦めツールのリスト（https://wiki.openstack.org/wiki/OperationsTools ）といった情報も載っています。最後に、Planet OpenStack（http://planet.openstack.org）には多くのブログが集められています。

第17章 高度な設定

ドライバーによる違い 195
周期的タスク 196
設定に関する個別のトピック 197

OpenStack は、非常に小さなプライベートクラウドから大規模なパブリッククラウドまで様々な構成でうまく動くことを意図して作られています。これを実現するため、開発者はコードに設定オプションを用意し、要件にあわせて種々のコンポーネントの振る舞いを細かく調整できるようにしています。しかし、残念ながら、デフォルトの設定値ですべてのデプロイメントに対応することはできません。

執筆時点では、OpenStack には 1,500 以上の設定オプションがあります。これらのオプションは OpenStack 設定レファレンスガイドに記載されています。この章では、これら全てについて説明するつもりはありませんが、さらなる情報が必要なときにどこを探せばよいか分かるように、大事な考え方について説明したいと思います。

ドライバーによる違い

多くの OpenStack プロジェクトではドライバー層が実装され、各ドライバーはドライバー固有の設定オプションが実装されています。例えば、OpenStack Compute (Nova) では、libvirt, xenserver, hyper-v, vmware などの種々のハイパーバイザードライバーが実装されていますが、これらのハイパーバイザードライバーすべてが同じ機能を持っている訳ではなく、異なるチューニング要件もドライバー毎に異なります。

注記

現在実装されているハイパーバイザーの一覧は OpenStack のドキュメントウェブサイトにあります。OpenStack Compute (Nova) のハイパーバイザードライバーの各種機能の対応状況は、OpenStack Wiki の Hypervisor support matrix ページに記載されています。

ここで言っておきたいことは、オプションが存在するからといって、そのオプションがあなたが選んだドライバーに関係するとは限らないということである。通常は、ドキュメントには、その設定オプションが適用されるドライバーについての記載があります。

周期的タスク

様々な OpenStack プロジェクトに共通する別の考え方として、周期的タスク（periodic task）があります。周期的タスクは伝統的な Unix システムの cron ジョブに似ていますが、OpenStack プロセスの内部で実行されます。例えば、OpenStack Compute（Nova）はローカルキャッシュからどのイメージを削除できるかを決める必要がある際に、これを行うために周期的タスクを実行します。

周期的タスクは、OpenStack が使用しているスレッドモデルにおける制限を理解する上で重要です。OpenStack は Python の協調スレッドを使用しています。このことは、何か長く複雑な処理が実行された場合、その処理が自発的に別の協調スレッドに実行を譲らない限り、そのプロセス内の他のタスクの実行が停止されることを意味します。

これの具体的な例が nova-compute プロセスです。libvirt でイメージキャッシュを管理するために、nova-compute はイメージキャッシュの内容をスキャンする周期的なプロセスを用意します。このスキャンの中で、各イメージのチェックサムを計算し、チェックサムが nova-compute が期待する値と一致することを確認します。しかしながら、イメージは非常に大きく、チェックサムを生成するのに長い時間がかかる場合があります。このことがバグとして報告され修正される前の時点では、nova-compute はこのタスクで停止し RPC リクエストに対する応答を停止してしまっていました。この振る舞いは、インスタンスの起動や削除などの操作の失敗としてユーザーに見えていました。

これから分かることは、OpenStack のプロセスが少しの間「停止」したように見えて、それから通常通りの動作を継続するような状況を見つけた場合、周期的タスクが問題になっていないかを確認すべきだということです。取りうる一つの方法は、間隔を 0 に設定して周期的タスクを無効にすることです。また、周期的タスクの実行頻度を設定することもできます。デフォルトとは異なる頻度で周期的タスクを実行する方が意味がある場合もあります。

実行頻度は周期的タスク別に定義されています。したがって、OpenStack Compute（Nova）ではすべての周期的タスクは無効にするためには、多くの設定オプションを 0 に設定する必要があることでしょう。現在のところ 0 に設定する必要がある設定オプションの一覧は以下のとおりです。

- bandwidth_poll_interval
- sync_power_state_interval
- heal_instance_info_cache_interval

- host_state_interval
- image_cache_manager_interval
- reclaim_instance_interval
- volume_usage_poll_interval
- shelved_poll_interval
- shelved_offload_time
- instance_delete_interval

設定オプションを 0 に設定するには、nova.conf に image_cache_manager_interval=0 のような行を入れてください。

上記のリストはリリース間で変更されることもあります。最新の情報については設定ガイドを参照してください。

設定に関する個別のトピック

この節では、調整を検討した方がよい設定オプションの個別の具体例を扱います。

OpenStack Compute (Nova)

周期的タスクの頻度

Grizzly リリースより前では、周期的タスクの頻度はタスクの実行間の秒数で指定していました。これは、周期的タスクの実行に30分かかり、頻度が1時間に一度に設定されていた場合、その周期的タスクは実際には90分に一回実行されることを意味します。そのタスクは実行完了後にもう一度実行されるまでに1時間待つからです。この動作は Grizzly で変更され、現在は、周期的タスクの頻度はそのタスクの実行の開始からの時間で計算されます。そのため、30分かかる周期的タスクは 1時間毎に実行され、最初の実行の完了から次の実行の開始までの待ち時間は30分になります。

付録A 事例

目次

NeCTAR 199
MIT CSAIL 200
DAIR 201
CERN 201

この節ではコミュニティからを事例をいくつか紹介します。これらでは通常より多くの技術的詳細情報が提供されています。他の事例はOpenStack ウェブサイト (https://www.openstack.org/user-stories/)で探して下さい。

NeCTAR

利用者：オーストラリアの公的資金による研究部門からの研究者。用途は、シンプルな Web サーバー用のインスタンスから高スループットコンピューティング用の数百のコア使用まで、多種多様な専門分野に渡ります。

デプロイ

OpenStack Compute Cells を使用して、NeCTAR クラウドは８サイトに及び、１サイトあたり約4,000コアがあります。

各サイトは（OpenStack Compute のセル設定におけるリソースセルとして）異なる設定で実行されています。数サイトは複数データセンターに渡り、コンピュートノード外のストレージを共有ストレージで使用しているサイトもあれば、コンピュートノード上のストレージを非共有型ファイルシステムで使用しているサイトもあります。各サイトは Object Storage バックエンドを持つ Image Service をデプロイしています。中央の Identity Service、Dashboard、Compute API サービスが使用されています。Dashboard へのログインが Shibboleth の SAML ログインのトリガーになり、SQL バックエンドの Identity サービスのアカウントを作成します。

コンピュートノードは 24～48コアがあり、１コアあたり 4GB 以上の RAM があり、１コアあたり約 40GB 以上の一時ストレージがあります。

全サイトは Ubuntu 12.04 をベースにしており、ハイパーバイザとして KVM を使用しています。使用している OpenStack のバージョンは基本的に安定バージョンであり、5〜10%のコードが開発コードからバックポートされたか、修正されています。

リソース

- OpenStack.org Case Study (https://www.openstack.org/user-stories/nectar/)
- NeCTAR-RC GitHub (https://github.com/NeCTAR-RC/)
- NeCTAR Web サイト (https://www.nectar.org.au/)

MIT CSAIL

利用者：マサチューセッツ工科大学コンピュータ科学・人工知能研究所の研究者。

デプロイ

CSAIL クラウドは現在 64 物理ノード、768 物理コア、3,456 GB のメモリがあります。クラウドリソースがコンピュータリソースに焦点をあてているため、永続データストレージの大部分は、クラウド外の NFS 上にあります。現在 65 ユーザと 23 プロジェクトがあり、拡張が期待される約 90% 近くの利用率があります。

ソフトウェアスタックは Ubuntu 12.04 LTS と Ubuntu Cloud Archive からの OpenStack Folsom です。KVM がハイパーバイザで、FAI(http://fai-project.org/)と Puppet を設定管理に使用してデプロイされています。FAI と Puppet の組み合わせはOpenStack のみならず研究所全体で使用されています。単一のクラウドコントローラーノードがあり、他のコンピュータ・ハードウェアはコンピュートノードに使用されています。用途が計算集約型の為、nova.conf 中の物理 CPU・RAM と仮想 CPU・RAM の比は 1:1 です。しかしながら、ハイパースレッディングが有効であり（Linux がその方式を複数 CPU でカウントする為）実際には通常 2:1 になっています。

ネットワーク面では、物理システムは２つのネットワーク・インターフェースと、独立したIPMI管理用のマネージメントカードがあります。OpenStack ネットワークサービスはマルチホストネットワーク、FlatDHCP を使用しています。

DAIR

利用者：DAIR は新しい情報通信技術（ICT）と他のデジタル技術を開発・評価するための CANARIE ネットワークを活用した統合仮想環境です。このシステムは、先進的なネットワーク、クラウドコンピューティング、ストレージといったデジタルインフラから構成されており、革新的な ICT アプリケーション、プロトコル、サービス、の開発・評価環境の作成、デプロイのスケールに関する実験の実施、市場へのより早期の投入促進を目的としています。

デプロイ

DAIR はカナダの2つの異なるデータセンタ（１つはアルバータ州、もう1つはケベック州）でホスティングされています。各拠点にはそれぞれクラウドコントローラがありますが、その１つが「マスター」コントローラーとして、認証とクォータ管理を集中して行うよう設計されています。これは、特製スクリプトと OpenStack の軽微な改造により実現されています。DAIR は現在、Folsom で運営されています。

オブジェクトストレージ用に、各リージョンには Swift 環境があります。

各リージョンでは、ブロックストレージとインスタンスストレージの両方でNetApp アプライアンスが使用されています。これらのインスタンスを NetApp アプライアンスから Ceph または GlusterFS といった分散ファイルシステム上に移動する計画があります。

ネットワーク管理は VlanManager が広範囲に使用されています。全てのサーバーは２つの冗長化（bonding）された 10GB NIC があり、２つの独立したスイッチに接続されています。DAIR はクラウドコントローラーが全コンピュートノード上の全インスタンス用のゲートウェイとなる、単一ノードのネットワーキングを使用する設定がされています。内部の OpenStack 通信（例：ストレージ通信）はクラウドコントローラーを経由していません。

リソース

- DAIR ホームページ（http://www.canarie.ca/en/dair-program/about）

CERN

利用者：高エネルギー物理学研究を管理する CERN（欧州原子核共同研究機関）の研究者。

デプロイ

この環境は、大部分は Red Hat 互換の Scientific Linux 6 ベースです。主なハイパーバイザとして KVM を使用していますが、一方 Windows Server 2008 上の Hyper-V を使用したテストも進行中です。

我々は Compute、Image Service、Identity Service、Dashboard の設定に Puppet Labs のOpenStack モジュールを使用しています。

ユーザとグループは Active Directory で管理され、LDAP を使用して Identity Service にインポートされます。CLI は nova と euca2ools が使用可能です。

CERN では現在 3 つのクラウドが稼働しており、合計で約 3400 台の Nova コンピュートノード、 60,000 コアがあります。 CERN IT クラウドは 2015年までに 300,000 コアにまで拡張される予定です。

リソース

- OpenStack in Production: A tale of 3 OpenStack Clouds (openstack-in-production.blogspot.com/2013/09/a-tale-of-3-openstack-clouds-50000.html)
- Review of CERN Data Centre Infrastructure (http://cern.ch/go/N8wp)
- CERN Cloud Infrastructure User Guide (http://information-technology.web.cern.ch/book/cern-private-cloud-user-guide)

付録B ハリウッド^H^H^H^H^Hクラウドナイトメア

目次

ダブル VLAN 203
「あの問題」 206
イメージの消失 208
バレンタインデーのコンピュートノード大虐殺 210
ウサギの穴に落ちて 212

ここにあるのは、OpenStack クラウドオペレータ達の苦闘の抜粋である。これを読み、彼らの叡智を学ぶが良い。

ダブル VLAN

私は、新しい OpenStack クラウドのセットアップをするため、カナダのブリティッシュコロンビア州ケロウナの現地にいた。デプロイ作業は完全に自動化されていた。Cobbler が物理マシンに OS をデプロイし、それを起動し、その後は Puppet が引き継いだ。私は練習で幾度もデプロイシナリオを実行してきたし、もちろん全て正常であった。

ケロウナの最終日、私はホテルから電話会議に参加していた。その裏で、私は新しいクラウドをいじっていた。私はインスタンスを１つ起動し、ログインした。全ては正常に思えた。退屈しのぎに、私は ps aux を実行したところ、突然そのインスタンスがロックアップしてしまった。

これは単なる１回限りの問題と思ったので、私はインスタンスを削除して、新しいインスタンスを起動した。その後電話会議は終了し、私はデータセンターを離れた。

データセンターで、私はいくつかの仕事を済ませると、ロックアップのことを思い出した。私は新しいインスタンスにログインし、再度 ps aux を実行した。コマンドは機能した。ふぅ。私はもう一度試してみることにした。今度はロックアップした。何だこれは？

何度か問題が再現した後、私はこのクラウドが実は問題を抱えているという不幸な結論に至った。更に悪いことに、私がケロウナから出発する時間になっており、カルガリーに戻らなければならなかった。

どこかであなたはこのような障害調査を行ったことがあるだろうか？インスタンスはコマンドを打つ度に全くランダムにロックアップしてしまう。元になったイメージの問題か？No－全てのイメージで同じ問題が発生する。コンピュートノードの問題か？No－全てのノードで発生する。インスタンスはロックアップしたのか？No！新しいSSH接続は問題なく機能する！

我々は助けを求めた。ネットワークエンジニアは、これは MTU の問題ではないかというのだ。素晴らしい！MTU! 事態は動き始めた! MTU とは何で、何故それが問題になるのだろうか？

MTU とは最大転送単位（Maximum Transmission Unit）である。これは、各パケットに対してそのインターフェースが受け取る最大バイト数を指定する。もし２つのインターフェースが異なる MTU であった場合、バイトは尻切れトンボとなって変なことが起こり始める…例えばセッションのランダムなロックアップとか。

注記

すべてのパケットサイズが 1500 に収まるわけではない。SSH 経由の ls コマンド実行は 1500 バイト未満のサイズのパケット１つで収まるかもしれない。しかし、 ps aux のように多大な出力を行うコマンドを実行する場合、1500 バイトのパケットが複数必要とある。

OK。では MTU の問題はどこから来るのか？なぜ我々は他のデプロイでこの問題に遭遇しなかったのか？この状況は何が新しいのか？えっと、新しいデータセンター、新しい上位リンク、新しいスイッチ、スイッチの新機種、新しいサーバー、サーバーの新機種…つまり、基本的に全てが新しいものだった。素晴らしい。我々は様々な領域で MTU の増加を試してみた。スイッチ、コンピュータのNIC、インスタンスの仮想NIC、データセンターの上位リンク用のインターフェースのMTUまでいじってみた。いくつかの変更ではうまくいったが、他はダメだった。やはり、この線の障害対策はうまくいってないようだった。我々はこれらの領域のMTUは変更すべきではないようだ。

結局、我々のネットワーク管理者（Alvao）と私自身は４つのターミナルウィンドウ、１本の鉛筆と紙切れを持って座った。１つのウインドウで我々は ping を実行した。２つ目のウインドウではクラウドコントローラー上の tcpdump、３つ目ではコンピュートノード上の tcpdump、４つ目ではインスタンス上の tcpdump を実行した。前提として、このクラウドはマルチノード、非マルチホスト構成である。

１つのクラウドコントローラーが全コンピュートノードのゲートウェイの役割を果たしていた。ネットワーク設定には VlanManager が使われて

いた。これは、クラウドコントローラーと全コンピュートノードで、各 OpenStack プロジェクトが異なる VLAN を持つことを意味する。パケットサイズ変更のため、ping の -s オプションを使用していた。パケットが全て戻ってくる時もあれば、パケットが出ていったきり全く戻って来ない時もあれば、パケットはランダムな場所で止まってしまう時もある、という状況だった。tcpdump を変更し、パケットの16進ダンプを表示するようにした。外部、コントローラー、コンピュート、インスタンスのあらゆる組み合わせの間で ping を実行した。

遂に、Alvaro が何かを掴んだ。外部からのパケットがクラウドコントローラーを叩いた際、パケットは VLAN で設定されるべきではない。我々はこれが正しいことを検証した。パケットがクラウドコントローラーからコンピュートノードに行く際、パケットはインスタンス宛の場合にのみ VLAN を持つべきである。これもまた正しかった。ping のレスポンスがインスタンスから送られる際、パケットは VLAN 中にいるべきである。ＯＫ。クラウドコントローラーからパブリックインターネットにパケットが戻る際、パケットには VLAN を持つべきではない。ＮＧ。うぉっ。まるで パケットの VLAN 部分が削除されていないように見える。

これでは意味が無かった。

このアイデアが我々の頭を駆け巡る間、私はコンピュートノード上でコマンドをランダムに叩いていた。

```
$ ip a
…
10: vlan100@vlan20: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
 master br100 state UP
…
```

「Alvaro、VLAN 上に VLAN って作れるのかい？」

「もしやったら、パケットに余計に４バイト追加になるよ…」

やっと事の全容が判明した…

```
$ grep vlan_interface /etc/nova/nova.conf
vlan_interface=vlan20
```

nova.conf 中で、vlan_interface は OpenStack が全ての VLAN をアタッチすべきインターフェースがどれかを指定する。正しい設定はこうだった:

```
vlan_interface=bond0
```

これはサーバーの冗長化された（bonded）NIC であるべきだからだ。

vlan20 はデータセンターが外向けのパブリックなインターネットアクセス用に我々に付与した VLAN である。これは正しい VLAN で bond0 にアタッチされている。

ミスにより、私は全てのテナント VLAN を bond0 の代わりに vlan20 にアタッチするよう OpenStack を設定した。それによって１つの VLAN が別の VLAN の上に積み重なり、各パケットに余分に４バイトが追加され、送信されるパケットサイズが 1504 バイトになる原因となった。これがパケットサイズ 1500 のみ許容するインターフェースに到達した際、問題の原因となったのだった！

全力でこの問題を修正した結果、全てが正常に動作するようになった。

「あの問題」

2012年8月の終わり、カナダ アルバータ州のある大学はそのインフラを OpenStack クラウドに移行した。幸か不幸か、サービスインから1〜2日間に、彼らのサーバーの1台がネットワークから消失した。ビッ。いなくなった。

インスタンスの再起動後、全ては元通りに動くようになった。我々はログを見直し、問題の箇所（ネットワーク通信が止まり、全ては待機状態になった）を見た。我々はランダムな事象の原因はこのインスタンスだと判断した。

数日後、それは再び起こった。

我々はログのセットを両方見直した。頻発したログの１つは DHCP だった。当時、OpenStack はデフォルトでは DHCP リース期間を 1分に設定していた（現在は 2分）。これは、各インスタンスが固定 IP アドレスを更新するためにクラウドコントローラー（DHCP サーバー）に接続することを意味する。幾つかの理由で、このインスタンスはその IP アドレスを更新できなかった。インスタンスのログとクラウドコントローラー上のログを突き合わせ、並べてやりとりにしてみた。

1. インスタンスはIPアドレスを更新しようとする。
2. クラウドコントローラーは更新リクエストを受信し、レスポンスを返す。
3. インスタンスはそのレスポンスを「無視」して、更新リクエストを再送する。
4. クラウドコントローラーは２度めのリクエストを受信し、新しいレスポンスを返す。

5. インスタンスはクラウドコントローラーからのレスポンスを受信しなかったため、更新リクエストを255.255.255.255に送信し始める。

6. クラウドコントローラーは 255.255.255.255 宛のリクエストを受信し、３番めのレスポンスを返す。

7. 最終的に、インスタンスはIPアドレス取得を諦める。

この情報により、我々は問題が DHCP 実行に起因するものと確信した。何らかの理由でインスタンスが新しいIPアドレスを取得できず、その結果IPアドレスがなくなり、インスタンスは自分自身をネットワークから切り離した、と考えた。

ちょっと Google 検索した結果、これを見つけた。VLAN モードでの DHCPリースエラー（https://lists.launchpad.net/openstack/msg11696.html）。この情報はその後の我々の DHCP 方針の支えになった。

最初のアイデアは、単にリース時間を増やすことだった。もしインスタンスが毎週１回だけIPアドレスを更新するのであれば、毎分更新する場合よりこの問題が起こる可能性は極端に低くなるだろう。これはこの問題を解決しないが、問題を単に取り繕うことはできる。

我々は、このインスタンス上で tcpdump を実行して、操作で再びこの現象に遭遇するか見てみることにした。実際、我々はやってみた。

tcpdump の結果は非常に奇妙だった。一言で言えば、インスタンスが IP アドレスを更新しようとする前に、まるでネットワーク通信が停止しているように見えた。１分間のリース期間で大量の DHCP ネゴシエーションがあるため、確認作業は困難を極めた。しかし、パケット間のたった数ミリ秒の違いであれ、あるパケットが最初に到着する際、そのパケットが最初に到着し、そのパケットがネットワーク障害を報告した場合、DHCP より前にネットワーク障害が発生していることになる。

加えて、問題のインスタンスは毎晩非常に長いバックアップジョブを担っていた。「あの問題」（今では我々はこの障害をこう呼んでいる）はバックアップが行われている最中には起こらなかったが、（数時間たっていて）「あの問題」が起こるまであと少しのところだった。

それから何日か過ぎ、我々は「あの問題」に度々遭遇した。我々は「あの問題」の発生後、dhclient が実行されていないことを発見した。今、我々は、それが DHCP の問題であるという考えに立ち戻った。/etc/init.d/networking restart を実行すると、全ては元通りに実行されるようになった。

探し続けてきた Google の検索結果が突然得られたという事態をお分かりだろうか？えっと、それがここで起こったことだ。私は dhclient の

情報と、何故 dhclient がそのリースを更新できない場合に死ぬのかを探していて、我々が遭遇したのと同じ問題についての OpenStack と dnsmasq の議論の束を突然発見した。

高負荷ネットワークと Dnsmasq における問題（http://www.gossamer-threads.com/lists/openstack/operators/18197）

DHCPOFFER が無いために起動中のインスタンスが IP アドレスを失う問題（http://www.gossamer-threads.com/lists/openstack/dev/14696）

マジ？Google。

このバグ報告は全ての鍵だった。 KVMイメージがブリッジネットワークで接続を失う（https://bugs.launchpad.net/ubuntu/+source/qemu-kvm/+bug/997978）

レポートを読むのは楽しかった。同じ奇妙なネットワーク問題にあった人々であふれていたが、全く同じ説明はなかった。

つまり、これは qemu/kvm のバグである。

バグ報告を発見すると同時に、同僚が「あの問題」を再現することに成功した！どうやって？彼は iperf を使用して、インスタンス上で膨大なネットワーク負荷をかけた。30分後、インスタンスはネットワークから姿を消した。

パッチを当てた qemu と再現方法を携えて、我々は「あの問題」を最終的に解決したかを確認する作業に着手した。インスタンスにネットワーク負荷をかけてから丸48時間後、我々は確信していた。その後のことは知っての通りだ。あなたは、joe へのバグ報告を検索し、私のコメントと実際のテストを見つけることができる。

イメージの消失

2012年の終わり、Cybera （カナダ アルバータ州にある、サイバーインフラのデプロイを監督する権限を持つ非営利団体）が、彼らの DAIR プロジェクト（http://www.canarie.ca/en/dair-program/about）用に新しい OpenStack クラウドをデプロイした。サービスインから数日後、あるコンピュートノードがロックアップした。問題のノードの再起動にあたり、私は顧客の権限でインスタンスを起動するため、そのノード上で何のインスタンスがホスティングされていたかを確認した。幸運にも、インスタンスは１つだけだった。

nova reboot コマンドは機能しなかったので、virsh を使用したが、すぐに仮想ディスクが見つからないとのエラーが返ってきた。この場合、

仮想ディスクは Glance イメージで、イメージが最初に使用する際に /var/lib/nova/instances/_base にコピーされていた。何故イメージが見つからないのか？私はそのディレクトリをチェックし、イメージがないことを知った。

私は nova データベースを見直し、nova.instances テーブル中の当該インスタンスのレコードを見た。インスタンスが使用しているイメージは virsh が報告したものと一致した。よって、ここでは矛盾は発見されなかった。

私は Glance をチェックし、問題のイメージがそのユーザの作成したスナップショットであることに注目した。最終的に、それはグッドニュースだった。このユーザが影響を受けた唯一のユーザだった。

最後に、私は StackTack をチェックし、ユーザのイベントを見直した。彼らはいくつかのスナップショットを作ったり消したりしていた－ありそうな操作ではあるが。タイムスタンプが一致しないとはいえ、彼らがインスタンスを起動して、その後スナップショットを削除し、それが何故か /var/lib/nova/instances/_base から削除されたというのが私の結論だった。大した意味は無かったが、それがその時私が得た全てだった。

コンピュートノードがロックアップした原因はハードウェアの問題だったことが判明した。我々はそのハードウェアを DAIR クラウドから取り外し、修理するよう Dell に依頼した。Dell が到着して作業を開始した。何とかかんとか（あるいはタイプミス）で、異なるコンピュートノードを落としてしまい、再起動した。素晴らしい。

そのノードが完全に起動した際、インスタンスが起動した時に何が起こるのかを見るため、私は同じシナリオを実行して、インスタンスを復旧した。インスタンスは全部で４つあった。３つは起動し、１つはエラーになった。このエラーは以前のエラーと同じだった。「unable to find the backing disk.」マジ、何で？

再度、イメージがスナップショットであることが判明した。無事に起動した他の３インスタンスは標準のクラウドイメージであった。これはスナップショットの問題か？それは意味が無かった。

DAIR のアーキテクチャは /var/lib/nova/instances が共有 NFS マウントであることに注意したい。これは、全てのコンピュートノードがそのディレクトリにアクセスし、その中に _base ディレクトリが含まれることを意味していた。その他の集約化エリアはクラウドコントローラーの /var/log/rsyslog だ。このディレクトリは全コンピュートノードの全ての OpenStack ログが収集されていた。私は、virsh が報告したファイルに関するエントリがあるのだろうかと思った。

```
dair-ua-c03/nova.log:Dec 19 12:10:59 dair-ua-c03
2012-12-19 12:10:59 INFO nova.virt.libvirt.imagecache
[-] Removing base file:
/var/lib/nova/instances/_base/7b4783508212f5d242cbf9ff56fb8d33b4ce6166_10
```

あっはっは！じゃぁ、OpenStack が削除したのか。でも何故？

Essex で、_base 下の任意のファイルが使用されていないかどうか定期的にチェックして確認する機能が導入された。もしあれば、Nova はそのファイルを削除する。このアイデアは問題がないように見え、品質的にも良いようだった。しかし、この機能を有効にすると最終的にどうなるのか？Essex ではこの機能がデフォルトで無効化されていた。そうあるべきであったからだ。これは、Folsom で有効になることが決定された (https://bugs.launchpad.net/nova/+bug/1029674)。私はそうあるべきとは思わない。何故なら

何かを削除する操作はデフォルトで有効化されるべきではない。

今日、ディスクスペースは安価である。データの復元はそうではない。

次に、DAIR の共有された /var/lib/nova/instances が問題を助長した。全コンピュートノードがこのディレクトリにアクセスするため、全てのコンピュートノードは定期的に _base ディレクトリを見直していた。あるイメージを使用しているインスタンスが１つだけあり、そのインスタンスが存在するノードが数分間ダウンした場合、そのイメージが使用中であるという印を付けられなくなる。それゆえ、イメージは使用中に見えず、削除されてしまったのだ。そのコンピュートノードが復帰した際、そのノード上でホスティングされていたインスタンスは起動できない。

バレンタインデーのコンピュートノード大虐殺

この物語のタイトルは実際の事件よりかなりドラマティックだが、私はタイトル中に「バレンタインデーの大虐殺」を使用する機会が再びあるとは思わない（し望まない）。

この前のバレンタインデーに、クラウド中にあるコンピュートノードが最早動いていないとの警告を受け取った。つまり

```
$nova-manage service list
```

の出力で、この特定のノードの状態が XXX になっていた。

実に奇妙なことだが、私はクラウドコントローラーにログインし、問題のコンピュートノードに ping と SSH の両方を実行できた。通常、この種の警告を受け取ると、コンピュートノードは完全にロックしていてアクセス不可になる。

数分間のトラブル調査の後、以下の詳細が判明した。

- 最近、あるユーザがそのノード上で CentOS のインスタンスを起動しようとした。
- このユーザはそのノード（新しいノード）上の唯一のユーザだった。
- 私が警告を受け取る直前、負荷率は８に急増した。
- 冗長化された 10Gb ネットワークデバイス(bond0) は DOWN 状態だった。
- 1Gb NICはまだ生きていて、有効だった。

私は bonding ペアの両方の NIC の状態を確認し、両方ともスイッチポートへの通信ができないことを知った。bond 中の各 NIC が異なるスイッチに接続されていることを知り、私は、各スイッチのスイッチポートが同時に死ぬ可能性はまずないと思った。私は 10Gb デュアルポート NIC が死んで、交換が必要だと結論づけた。私は、そのノードがホスティングされているデータセンターのハードウェアサポート部門に宛てたチケットを作成した。私は、それが新しいノードで、他のインスタンスがまだそのノード上でホスティングされていないことを幸運に思った。

１時間後、私は同じ警告を受信したが、別のコンピュートノードだった。拍手。OK、問題は間違いなく現在進行中だ。元のノードと全く同様に、私は SSH でログインすることが出来た。bond0 NIC は DOWN だったが、1Gb NIC は有効だった。

そして、最も重要なこと。同じユーザが CentOS インスタンスを作成しようとしたばかりだった。何だと？

私はこの時点で完全に混乱した。よって、私はネットワーク管理者に対して、私を助けられるか聞いてみるためメールした。彼は両方のスイッチにログインし、すぐに問題を発見した。そのスイッチは２つのコンピュートノードから来たスパニングツリーパケットを検出し、スパニングツリーループを回避するため、即時にそれらのポートをダウンさせたのだ。

```
Feb 15 01:40:18 SW-1 Stp: %SPANTREE-4-BLOCK_BPDUGUARD: Received BPDU packet
 on Port-Channel35 with BPDU guard enabled. Disabling interface. (source mac
 fa:16:3e:24:e7:22)
```

```
Feb 15 01:40:18 SW-1 Ebra: %ETH-4-ERRDISABLE: bpduguard error detected on
 Port-Channel35.
Feb 15 01:40:18 SW-1 Mlag: %MLAG-4-INTF_INACTIVE_LOCAL: Local interface Port-
Channel35 is link down. MLAG 35 is inactive.
Feb 15 01:40:18 SW-1 Ebra: %LINEPROTO-5-UPDOWN: Line protocol on Interface
 Port-Channel35 (Server35), changed state to down
Feb 15 01:40:19 SW-1 Stp: %SPANTREE-6-INTERFACE_DEL: Interface Port-Channel35
 has been removed from instance MST0
Feb 15 01:40:19 SW-1 Ebra: %LINEPROTO-5-UPDOWN: Line protocol on Interface
 Ethernet35 (Server35), changed state to down
```

彼はスイッチポートを再度有効にしたところ、２つのコンピュートノードは即時に復活した。

不幸にも、この話にはエンディングがない…我々は、なぜ CentOS イメージがスパニングツリーパケットを送信し始める原因をいまだ探している。更に、我々は障害時にスパニングツリーを軽減する正しい方法を調査している。これは誰かが思うより大きな問題だ。スパニングツリーループを防ぐことはスイッチにとって非常に重要であるが、スパニングツリーが起こった際に、コンピュートノード全体がネットワークから切り離されることも大きな問題である。コンピュートノードが 100 インスタンスをホスティングしていて、そのうち１つがスパニングツリーパケットを送信した場合、そのインスタンスは事実上他の 99 インスタンスを DDoS（サービス不能攻撃）したことになる。

これはネットワーク業界で進行中で話題のトピックである（特に仮想マシンと仮想スイッチで発生する）。

ウサギの穴に落ちて

稼働中のインスタンスからコンソールログを取得可能なユーザはサポートの恩恵となる（インスタンスの中で何が起こっているのか何度も確認できるし、あなたが悩まずに問題を修正することができる）。不幸なことに、過剰な失敗の記録は時々、自らの問題となり得る。

報告が入った。VM の起動が遅いか、全く起動しない。標準のチェック項目は？Nagios 上は問題なかったが、RabbitMQ クラスタの現用系に向かうネットワークのみ高負荷を示していた。捜査を開始したが、すぐに RabbitMQ クラスタの別の部分がざるのようにメモリリークを起こしていることを発見した。また警報か？RabbitMQ サーバーの現用系はダウンしようとしていた。接続は待機系にフェイルオーバーした。

この時、我々のコントロールサービスは別のチームによりホスティングされており、我々には現用系サーバー上で何が起こっているのかを調査するための大したデバッグ情報がなく、再起動もできなかった。このチームは警報なしで障害が起こったと連絡してきたが、そのサーバーの

再起動を管理していた。１時間後、クラスタは通常状態に復帰し、我々はその日は帰宅した。

翌朝の継続調査は別の同様の障害でいきなり始まった。我々は急いで RabbitMQ サーバーを再起動し、何故 RabbitMQ がそのような過剰なネットワーク負荷に直面しているのかを調べようとした。nova-api のデバッグログを出力することにより、理由はすぐに判明した。tail -f /var/log/nova/nova-api.log は我々が見たこともない速さでスクロールしていた。CTRL+C でコマンドを止め、障害を吐き出していたシステムログの内容をはっきり目にすることが出来た。－我々のユーザの１人のインスタンスからのシステムログだった。

インスタンスIDの発見後、console.log を探すため /var/lib/nova/instances にアクセスした。

```
adm@cc12:/var/lib/nova/instances/instance-00000e05# wc -l console.log
92890453 console.log
adm@cc12:/var/lib/nova/instances/instance-00000e05# ls -sh console.log
5.5G console.log
```

思った通り、ユーザはダッシュボード上のコンソールログページを定期的に更新しており、ダッシュボードに向けて5GB のファイルが RabbitMQ クラスタを通過していた。

我々はユーザを呼び、しばらくダッシュボードの更新を止めるよう申し入れた。すると、恐ろしい VM の破壊は止み、彼らは大いに喜んだ。その後、我々はコンソールログのサイズを監視するようになった。

今日に至るまで、この問題（https://bugs.launchpad.net/nova/+bug/832507）には完全な解決策がないが、我々は次回のサミットの議論に期待している。

付録C リソース

OpenStack

OpenStack 設定レファレンス (http://docs.openstack.org/trunk/config-reference/content/section_compute-hypervisors.html)

OpenStack インストールガイド - Ubuntu (http://docs.openstack.org/havana/install-guide/install/apt/content/)

OpenStack クラウド管理ガイド (http://docs.openstack.org/admin-guide-cloud/content/)

OpenStack セキュリティガイド (http://docs.openstack.org/security-guide/content/)

OpenStack Cloud Computing Cookbook (http://www.packtpub.com/openstack-cloud-computing-cookbook-second-edition/book)

クラウド（全般）

NIST Cloud Computing Definition (http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf)

Python

Dive Into Python (http://www.diveintopython.net)

ネットワーク

TCP/IP Illustrated (http://www.pearsonhighered.com/educator/product/TCPIP-Illustrated-Volume-1-The-Protocols/9780321336316.page)

The TCP/IP Guide (http://nostarch.com/tcpip.htm)

A tcpdump Tutorial and Primer (http://danielmiessler.com/study/tcpdump/)

システム管理

UNIX and Linux Systems Administration Handbook (http://www.admin.com/)

仮想化

The Book of Xen (http://nostarch.com/xen.htm)

設定管理

Puppet Labs Documentation (http://docs.puppetlabs.com/)

Pro Puppet (http://www.apress.com/9781430230571)

用語集

OpenStack 関連の用語や言い回しの定義を確認するために、この用語集を使用してください。

この用語集に追加するには、OpenStack の貢献プロセスを通して、 github.com にある openstack/openstack-manuals リポジトリをフォークして、ソースファイルを更新します。

A

アカウント
: Swift のアカウントコンテキスト、または Active Directory、/etc/passwd、OpenLDAP、Keystone などの Identity サービスのアカウント。

account auditor
: バックエンドの SQLite データベースに対して問合せを実行した特定の Swift アカウント中の消失したレプリカ、不正、破壊されたオブジェクトのチェック。

アカウントデータベース
: アカウントサーバーからアクセスされる、Swift アカウントと関連メタデータを含むSQLite データベース、又は、アカウントを含む Keystone バックエンド。

account reaper
: アカウントサーバー上で削除済みとマークされたアカウントデータベースをスキャン／削除する Swift のワーカープロセス。

account server
: Swift 中のコンテナの一覧を表示し、アカウントデータベース中のコンテナ情報を保存する。

account service
: 一覧、作成、修正、監査等のアカウントサービスを提供するSwift のコンポーネント。Keystone、OpenLDAP、又は同様のユーザアカウントサービスと混同しないこと。

Active Directory
: OpenStack においてサポートされる、LDAP に基づいた、Microsoft による認証および識別サービス。

アドレスプール
: Nova の 1 プロジェクトに紐付けられ、プロジェクト内の VM インスタンスに割り当てる事で使用可能な固定 IP アドレスと Floating IP アドレスの組。

管理 API
: 認証済み管理者がアクセス可能で一般に公共のインターネットからのエンドユーザアクセスが出来ない API コールのサブセット。独立したサービス（Keystone）として存在したり、別の API（Nova）のサブセットになり得る。

Amazon Kernel Image (AKI)
: 仮想マシンコンテナー形式および仮想マシンディスク形式。glance がサポートしている。

Amazon Machine Image (AMI)
: 仮想マシンコンテナー形式および仮想マシンディスク形式。glance がサポートしている。

Amazon Ramdisk Image (ARI)
: 仮想マシンコンテナー形式および仮想マシンディスク形式。glance がサポートしている。

Apache
: 現在インターネットにおいて使用されている最も一般的な Web サーバーのソフトウェア。HTTPd としても知られている。

Apache License 2.0
: すべての OpenStack コアプロジェクトは Apache License 2.0 ライセンスの条件で提供されている。

API エンドポイント
: クライアントが API にアクセスするために通信するデーモン、ワーカーまたはサービス。OpenStack では、認証、イメージの追加、仮想マシンの起動、ボリュームの追加といったサービスは API エンドポイントで提供される。

API 拡張
: 独自モジュールがコア API を拡張できるようにする Nova と Neutron の機能。

API 拡張プラグイン
: neutron プラグインまたは neutron API 拡張の別名。

API サーバー
: API エンドポイントを提供するデーモンまたはワーカーを実行するあらゆるノード。

API バージョン
: OpenStack において、プロジェクト向けの API バージョンは URL の一部である。たとえば、 example.com/nova/v1/foobar。

Application Programming Interface (API)
: サービス、アプリケーション、プログラムへのアクセスに使用される仕様の集合。サービス呼出、各呼出に必要なパラメーター、想定される戻り値を含む。

arptables
: ファイアウォールサービスを提供する為に Nova 中で iptables, ebtables, ip6tables と一緒に使用される。

Asynchronous JavaScript and XML (AJAX)
: 非同期 Web アプリケーションを作成する為にクライアント側で使用される相互関係のある Web 開発技術の集合。Horizon で広く使用されている。

アタッチ（ネットワーク）
: 論理ポートへのインターフェースIDの紐付け。インターフェースをポートに差し込む。

auditor
: Swift オブジェクト、コンテナ、アカウントの完全な状態を検証するワーカープロセス。Auditor は Swift account auditor, container auditor, object auditor の総称。

Austin
: OpenStack の最初のリリースのプロジェクト名。

認証
: ユーザー、プロセスまたはクライアントが、秘密鍵、秘密トークン、パスワード、フィンガープリントまたは同様の方式により示されている主体と本当に同じであることを確認するプロセス。AuthN と略される。

認証トークン
: 認証後にクライアントに提供されるテキスト文字列。API エンドポイントに続くリクエストにおいて、ユーザーまたはプロセスにより提供される必要がある。

認可
: ユーザー、プロセス、またはクライアントが、ある操作を実行する権限があることを検証する動作。Swift オブジェクトの削除、Swift コンテナーの一覧表示、Nova 仮想マシンの起動、パスワードのリセットなどの操作です。AuthZ と略される。

アベイラビリティゾーン
: クラウドデプロイの独立したエリア。

B

バックエンドカタログ
: クライアントが利用可能なAPI エンドポイントに関する情報の保存、取得の為に Keystone カタログサービスで使用されるストレージ方式。例：SQL データベース、LDAP データベース、KVS バックエンドを含む。

バックエンドストア
: glance が仮想マシンイメージを取得および保管するために使用する永続的なデータストア。オプションに swift、ローカルファイルシステム、S3 および HTTP がある。

bare
: VM イメージ用にコンテナ（保存先）がない事を示す Glance のコンテナフォーマット。

Bexar
: 2011年２月に登場した OpenStack 関連プロジェクトのまとまったリリース。Compute（Nova）と Object Storage（Swift）のみが含まれていた。

ブロックデバイス
: ブロック状態のデータを移動するデバイス。これらのデバイスノードにはハードディスク、CD-ROM ドライブ、フラッシュドライブ、その他のアドレス可能なメモリの範囲等がある。

ブロックマイグレーション
: ユーザが移動を指示してから非常に短いダウンタイムであるあるホストから別のホストにインスタンスを移動させる為に KVM で使用される VM ライブマイグレーション方式。共有ストレージを必要としない。Nova でサポート。

ブータブルディスクイメージ
: 単独の、ブート可能なファイルとして存在する仮想マシンイメージの形式。

ビルダーファイル
: Swift リング用の設定情報を含み、リングの再設定や深刻な障害後に１からリング情報を作成するのに使用される。

C

cache pruner
: Glance VM イメージキャッシュを設定された最大サイズ以下に保持する為に使用される実行可能プログラム。

Cactus
: 2011年春に登場した OpenStack 関連のプロジェクトリリース。Compute (Nova), Object Storage (Swift), Image Service (Glance) が含まれていた。

ケーパビリティ
: CPU、ストレージ、ネットワークを含むセルのリソースを定義する。1 セルやセル全体に含まれる特定のサービスに適用可能。

capacity cache
: 各ホストの現在の負荷、空き RAM 量、実行中の VM 数を含む、Nova バックエンドデータベース中のテーブル。どのホストで VM を起動するかを決定するのに使用される。

capacity updater
: VM インスタンスを監視し、必要に応じて容量キャッシュを更新する通知ドライバ。

カタログ
: Keystone での認証後に、ユーザに返される利用可能な API エンドポイントの一覧。

カタログサービス
: Keystone での認証後にユーザに返される利用可能な API エンドポイントの一覧を提供する Keystone サービス。

ceilometer
: OpenStack 用のメータリング・課金基盤を提供する育成プロジェクト。

セル
: 親子関係で Nova リソースの論理パーティションを提供する。親セルが要求されたリソースを提供できない場合、親セルからのリクエストは子セルに渡される。

セルフォワーディング
: 親が要求されたリソースを提供できない場合、親セルがリソース要求を子セルに渡す事を可能にするNovaオプション。

セルマネージャー
: セル中の各ホストの現在のキャパシティの一覧を含み、リクエストを適切な方法でルーティングする Nova コンポーネント。

Ceph
: オブジェクトストア、ブロックストア、および POSIX 互換分散ファイルシステムから構成される大規模スケール可能分散ストレージシステム。OpenStack 互換。

CephFS
: Ceph により提供される POSIX 互換ファイルシステム。

認証局
: cloudpipe VPN と VM イメージ復号用に Nova が提供するシンプルな認証局。

チャンススケジューラー
: プールから利用可能なホストをランダムに選択する為に Nova が使用するスケジュール方式。

changes-since
: 新しいデータセットからのダウンロードや古いデータに対する比較の代わりに、あなたの最後のリクエストから要求されたアイテム変更をダウンロードする事を可能にするNova API パラメーター。

Chef
: OpenStack をサポートする構成管理ツール。

子セル
: CPU 時間、ディスクストレージ、メモリ等の要求されたリソースが親セルで利用不可の場合、リクエストは親セルに紐付けられた子セルに転送される。子セルがリクエストに対応可能な場合、子セルはそのリクエストを処理する。対応不可の場合、そのリクエストを自分の子セルに渡そうとする。

cinder
: 仮想マシンのインスタンスに接続できるブロックデバイスを管理する OpenStack ブロックストレージサービス。

クラウドアーキテクト
: クラウドの作成を計画、設計および監督する人。

クラウドコントローラーノード
: network, volume, API, scheduler, Image service を実行するノード。スケーラビリティや可用性の為、各サービスは独立したノード群に分割され得る。

cloud-init
: メタデータサービスから取得したSSH 公開鍵やユーザデータ等の情報を使用して、起動後にインスタンスの初期化を実行する為にVMイメージに一般にインストールされるパッケージ。

cloudpipe
: プロジェクト単位の VPN 作成用に Nova が使用するサービス。

cloudpipe イメージ
: cloudpipe サーバとしてサービスを行う為の、予め用意された VM イメージ。本質的には Linux 上で実行される OpenVPN。

コマンドフィルタ
: Nova rootwrap 機構中で許可されたコマンドの一覧。

コミュニティプロジェクト
: OpenStack Foundation で公認されていないプロジェクト。プロジェクトが充分成功した場合、育成プロジェクトに昇格し、その後コアプロジェクトに昇格する事がある。あるいはメインの code trunk にマージされる事もある。

Compute API
: Nova サービスへのアクセスを提供する nova-api デーモン。Amazon EC2 API のようないくつかの外部 API でも通信可能。

Compute API extension
: Nova API extension の別名。

コンピュートコントローラー
: VM インスタンスを起動する適切なホストを選択する Nova のコンポーネント。

コンピュートノード
: nova-compute デーモンと仮想マシンインスタンスを実行するノード。

コンピュートサービス
: VM を管理する Nova コンポーネントの別名。

連結オブジェクト
: Swift 中で分割された大きなオブジェクト。再度結合されてから、クライアントに送信される。

一貫性ウインドウ
: 全クライアントがアクセス可能になる為に、新しい Swift オブジェクトに必要な時間。

コンソールログ
: Nova 中の Linux VM コンソールからの出力を含む。

コンテナ
: Swift 中でオブジェクトの構造化、格納に使用される。Linux のディレクトリと同様のコンセプトだが、階層構造にできない。Glance コンテナフォーマットの別名。

コンテナオーディタ
: SQLite バックエンドデータベースへのクエリを通して、指定された Swift コンテナ中の失われたレプリカや不完全なオブジェクトをチェックする。

コンテナデータベース
: Swift コンテナと関連メタデータを含む SQLite データベース。container サーバからアクセスされる。

コンテナフォーマット
: VM イメージの保存用に Glance が使用する「封筒」。マシン状態、OS ディスクサイズ等のメタデータに紐付けられる。

コンテナサーバー
: コンテナを管理する Swift のコンポーネント。

コンテナサービス
: 作成、削除、覧等のコンテナサービスを提供する Swift コンポーネント。

コントローラーノード
: クラウドコントローラーノードの別名。

コアAPI
: 文脈に依存する。コアAPI は OpenStack API またはNova, Neutron, Glance 等の特定のコアプロジェクトのメイン API のいずれかである。

コアプロジェクト
: OpenStack の公式プロジェクト。現在、Compute (nova), Object Storage (swift), Image Service (glance), Identity (keystone), Dashboard (horizon), Networking (neutron), Volume (cinder) が含まれる。

クレデンシャル
: ユーザーだけが知っているデータもしくはユーザーだけがアクセスできるデータで、認証時にサーバーに示され、ユーザーが誰であるかの確認に使用される。例：パスワード、シークレットキー、デジタル認証、フィンガープリントなど。

Crowbar
: クラウドの迅速なデプロイ用に全ての必要なサービスを提供する用途の、Dell によるオープンソースコミュニティプロジェクト。

カレントワークロード
: 指定されたホスト上で現在進行中の build, snapshot, migrate, resize の操作数を元に計算される、Nova のキャパシティキャッシュの 1 要素。

カスタムモジュール
: ダッシュボードのルックアンドフィールを変更する為に Horizon がロードする、ユーザが作成した Python モジュール。

D

ダッシュボード
: OpenStack 用 Web ベース管理インターフェース。Horizon の別名。

データベースレプリケータ
: 他ノードにアカウント、コンテナ、オブジェクトデータベースの変更をコピーする Swift のコンポーネント。

デフォルトパネル
: ユーザが Horizon ダッシュボードにアクセスした際に表示されるパネル。

デフォルトテナント
: ユーザ作成時にテナントが指定されなかった場合、新しいユーザはこの Keystone テナントに割り当てられる。

デフォルトトークン
: 特定のテナントに紐付けられず、スコープ付きトークン用と交換される Keystone トークン。

遅延削除
: イメージを即時削除する代わりに、あらかじめ定義された秒数後に削除する Glance のオプション。

デリバリモード
: Nova RabbitMQ メッセージ配信モード用設定。transient（一時）又は persistent（永続）のいずれかを設定できる。

デバイス
: Swift の文中では、下位のストレージ装置を指す。

デバイス ID
: Swift パーティション⇒物理ストレージ装置への対応表。

デバイスウェイト
: Swift デバイス間のパーティション分散に使用される。通常、分散はデバイスのストレージ容量に比例する。

DevStack
: 完全な OpenStack 開発環境を迅速にデプロイする為のシェルスクリプトを使用するコミュニティプロジェクト。

Diablo
: 2011年秋に登場した OpenStack 関連プロジェクトのリリース。Compute (nova 2011.3), Object Storage (swift 1.4.3), Image service (glance) が含まれる。

ディスクフォーマット
: Glance バックエンドストアで格納される VM 用ディスクイメージのフォーマット。 AMI, ISO, QCOW2, VMDK など。

dispersion
: Swift で、フォールトトレラントの確認の為に、オブジェクトとコンテナの分散をテスト、確認するツール。

Django
: Horizon 中で広く使用される Web フレームワーク。

dnsmasq
: DNS, DHCP, BOOTP, TFTP サービスを提供するデーモン。Nova の VLAN マネージャーと FlatDHCP マネージャーが使用する。

DNS レコード
: 特定のドメインに関する情報を指定し、ドメインに所属するレコード。

動的ホスト設定プロトコル (DHCP)
: ホスト起動時に自動的にネットワークを設定する方式。Neutron と Nova の両方が提供する。

E

ebtables
: ファイアウォールを作成し、ネットワーク通信の分離を確立する為に arptables, iptables, ip6tables と一緒に Nova で使用される。

EC2
: Amazon Elastic Compute Cloud。Nova と同様の機能を提供するアマゾンによるパブリッククラウド。

EC2 アクセスキー
: Nova の EC2 API へのアクセスの為に、EC2 シークレットキーと一緒に使用される。

EC2 API
: OpenStack は Nova で Amazon EC2 API によるアクセスをサポートする。

EC2 互換API
: OpenStack による Amazon EC2 による通信を可能にする Nova コンポーネント。

EC2 シークレットキー
: Nova の EC2 API による通信時に EC2 アクセスキーと一緒に使用される。各リクエストのデジタル署名に使用される。

Elastic Block Storage (EBS)
: Amazon の商用ブロックストレージ製品。 Cinder と似た機能を持つ。

エンドポイント
: API エンドポイントを参照。

エンドポイントレジストリ
: keystone カタログの別名。

エンドポイントテンプレート
: Object Storage、Compute、Identity などサービスがアクセスできる場所を示す、URL とポート番号のエンドポイントの一覧。

エンティティ
: Neutron（ネットワーク接続サービス）により提供されるネットワークサービスへの通信必要なハードウェア又はソフトウェア部品。エンティティは VIF を実装する為に Neutron に使用される。

エフェメラルストレージ
: 仮想マシンインスタンスにアタッチされたストレージボリューム。インスタンスが削除された後に永続しない。

Essex
: 2012年４月に登場した OpenStack 関連プロジェクトのリリース。Compute (nova 2012.1), Object Storage (swift 1.4.8), Image (glance), Identity (keystone), Dashboard (horizon) が含まれる。

ESX
: OpenStack がサポートしている、VMware のハイパーバイザーの１つ。

ESXi
: OpenStack がサポートしている、VMware のハイパーバイザーの１つ。

ETag
: Swift 中でのオブジェクトの MD5 ハッシュ。データの完全性の保証に使用される。

euca2ools
: 仮想マシンを管理するためのコマンドラインツール集。ほとんどは OpenStack と互換性がある。

evacuate
: １つまたは全ての仮想マシン（VM）インスタンスをあるホストから別のホストにマイグレーションする処理。共有ストレージのライブマイグレーションとブロックマイグレーション両方と互換がある。

エクステンション
: Nova API エクステンション又はプラグインの別名。Keystone の文脈では、OpenID 用追加サポートのような実装を示す呼び方。

拡張仕様
: ユーザが新しいインスタンスをリクエストする際に指定可能な、追加の要求事項。例：最小ネットワーク帯域、GPU。

F

FakeLDAP
: keystone と nova をテストするためにローカル LDAP ディレクトリを作成する簡単な方法。Redis が必要。

充填優先スケジューラー
: 様々なホスト上で新しい VM を起動するよりも、なるべく一つのホストを埋めようとする Nova スケジューリング手法。

フィルター
: VM を実行できないホストを排除し、選択されないようにする Nova のスケジューリング処理の段階。

ファイアウォール
: ホストノード間の通信を制限する為に使用される。iptables, arptables, ip6tables, ebtables を使用して Nova により実装される。

固定 IP アドレス
: インスタンス起動時に毎回同じインスタンスに割当られるIPアドレス（一般に、エンドユーザやパブリックインターネットからはアクセス出来ない）。インスタンスの管理に使用される。

FlatDHCP マネージャー
: OpenStack クラウド中の全サブネット用に単一のレイヤ２ドメインを提供する Nova のネットワークマネージャー。nova-network インスタンス毎に１つの DHCP サーバーを提供し、全インスタンスの IP アドレスを管理する。

Flat マネージャー
: 許可されたノードにIP アドレスを提供する Nova コンポーネント。DHCP、DNS、ルーティング設定、他の何かによって提供されるサービスを前提とする。

フラットモードインジェクション
: インスタンスが起動する前に VM イメージにOS ネットワーク設定を挿入する Nova ネットワーク方式。

フラットネットワーク
: 全インスタンスが同じサブネットに IP アドレスを持つ Nova のネットワーク構成。フラットネットワークは VLAN を使用しない。

フレーバー
: ユーザが利用可能な様々な仮想マシンイメージのパラメーター（CPU、ストレージ、メモリ等を含む）を示す。インスタンスタイプとしても知られている。

フレーバー ID
: Nova 又は Glance VM 用の、フレーバーまたはインスタンスタイプのUUID。

Floating IP アドレス
: Nova プロジェクトが 1 VM に紐付け可能な IP アドレス。これにより、インスタンス起動時に毎回同じパブリック IP アドレスがインスタンスに割り当てできる。DNS 割当管理用の一貫した IP アドレスを管理する為に、Floating IP アドレスのプールを作成し、VM 起動時にインスタンスに Floating IP アドレスを割り当てできる。

Folsom
: 2012年秋に登場した OpenStack 関連プロジェクトのリリース。Compute (nova), Object Storage (swift), Identity (keystone), Networking (neutron), Image service (glance)、Volumes 又は Block Storage (cinder) が含まれる。

FormPost
: ユーザが Web ページの Form を介してイメージをアップロード（ポスト）できるようにする為の Swift ミドルウェア。

G

glance
: OpenStack イメージサービスを提供するコアプロジェクト。

glance API サーバー
: VM 用クライアントリクエストの処理、registry サーバー上での Glance メタデータの更新、バックエンドストアから VM イメージをアップロードする為のストアアダプタによる通信。

グローバルエンドポイントテンプレート
: 全テナントが利用可能なサービスを含む Keystone のエンドポイントテンプレート。

GlusterFS
: オープンソース、分散、共有ファイルシステム。

Grizzly
: OpenStack の 7 回目リリースのプロジェクト名。

ゲスト OS
: ハイパーバイザーの管理下で実行しているオペレーティングシステムのインスタンス。

H

handover
: ドライブ障害の為、新しいオブジェクトのレプリカが自動的に作成される、Swift のオブジェクト状態。

ハードリブート
: きちんとした正常なOSのシャットダウンを行わず、物理又は仮想電源ボタンを押すタイプの再起動。

Heat
: OpenStack に複数のクラウドアプリケーションをオーケストレーションする為に開発されたプロジェクト。

horizon
: OpenStack ダッシュボードを提供するプロジェクトの名前。

ホスト
: 物理的なコンピュータ（ノードとしても知られる）。対比：インスタンス。

ホストアグリゲート
: アベイラビリティゾーンをホスト群の集合に分割する為の手法。

Hyper-V
: OpenStack によりサポートされるハイパーバイザーの一つ。Microsoft により開発された。

ハイパーバイザー
: VM のアクセスを実際の下位ハードウェアに仲介して制御するソフトウェア。

ハイパーバイザープール
: ホストアグリゲートにより一緒にグループ化されたハイパーバイザーの集合。

I

ID 番号
: Keystone 中で各ユーザに割り当てられた一意な数値 ID。Linux や LDAP の UID と同様の概念。

Identity API
: Identity サービス API の別名。

Identity バックエンド
: Keystone がユーザー情報を取得する情報源。例えば OpenLDAP サーバー。

Identity サービス
: 認証サービスを提供する（Keystone としても知られる）。

Identity サービス API
: Keystone が提供する OpenStack Identity サービスアクセスに使用される API。

イメージ
: サーバーの作成、再構築に使用する特定のオペレーティングシステム（OS）用のファイルの集合。起動したサーバーからカスタムイメージ（又はスナップショット）を作成する事ができる。

Image API
: 仮想マシンイメージの管理向け glance API エンドポイント。

イメージキャッシュ
: ユーザが要求したイメージを毎回他のイメージサーバーからイメージを再ダウンロードする代わりに、ローカルホスト上のイメージを利用できるようにする為にGlance により使用される。

イメージ ID
: イメージ API を介してGlance イメージアクセスに使用される URI と UUID の組み合わせ。

イメージメンバーシップ
: Glance 中で与えられた VM イメージへのアクセスが可能なテナントの一覧。

イメージ所有者
: Glance 仮想マシンイメージを所有する Keystone テナント。

イメージレジストリ
: Glance 経由で利用可能な VM イメージの一覧。

Image サービス API
: Glance イメージ API の別名。

イメージ状態
: Glance 中の VM イメージの現在の状態。稼働中のインスタンスの状態と混同しない事。

イメージストア
: VM イメージの保存用に Glance が使用するバックエンドストア。選択肢には Swift、ローカルファイルシステム、S3、HTTP がある。

イメージ UUID
: Glance が各 VM イメージを一意に識別するために使用する UUID。

インキュベートプロジェクト
: 育成プロジェクト。コミュニティプロジェクトがこの状態に昇格する事があり、その後コアプロジェクトに昇格する。

イングレスフィルタリング
: 外部から内部へのネットワーク通信のフィルタリング処理。Nova がサポートする。

インジェクション
: インスタンスが起動する前に、仮想マシンイメージ中にファイルを配置する処理。

インスタンス
: ハードウェアサーバーのように使用できる、実行中の仮想マシン、またはサスペンド中のような既知の状態にある仮想マシン。

インスタンス ID
: 実行中の各 nova 仮想マシンインスタンスを特定する一意な ID。

インスタンス状態
: Nova の VM イメージの現在の状態。

インスタンスタイプ
: フレーバーの別名。

インスタンスタイプ ID
: フレーバー ID の別名。

インスタンス UUID
: 各 nova 仮想マシンインスタンスに割り当てられた一意な ID。

インターフェース ID
: UUID 形式の、Neutron VIF 又は仮想 NIC 用の一意な ID。

ip6tables
: Nova の中でファイアウォールを作成するのに、arptables, ebtables, iptables と一緒に使用される。

iptables
: Nova 中でファイアウォールを作成する為に arptables, ebtables, ip6tables と一緒に使用される。

J

JavaScript Object Notation (JSON)
: OpenStack API 用にサポートされたレスポンスフォーマットの１つ。

Jenkins
: OpenStack 開発で自動的にジョブを実行するために使用されるツール。

K

kernel-based VM (KVM)
: OpenStack がサポートするハイパーバイザの１つ。

keystone
: OpenStack Identity サービスを提供するプロジェクト。

Kickstart
: Red Hat、Fedora、CentOS ベースの Linux ディストリビューションのインストールとシステム設定を自動化するツール。

L

ラージオブジェクト
: 5GB より大きな Swift 中のオブジェクト。

Launchpad
: OpenStack 用コラボレーションサイト。

レイヤ2 ネットワーク
: データリンクレイヤ用の OSI ネットワークアーキテクチャで使用される用語。

libvirt
: KVM、QEMU、LXC を含む多数の対応ハイパーバイザと通信する為の、OpenStack で使用される仮想化 API ライブラリ。

Linux ブリッジ
: Nova 中で、複数の VM が単一の物理 NIC を共用する事を可能にする為に使用されるソフトウェア。

Linux ブリッジ Neutron プラグイン
: Neutron のポート、インターフェース接続、その他抽象概念を認識する、Linux ブリッジ用のプラグイン

Linux コンテナー （LXC）
: OpenStack がサポートするハイパーバイザーの１つ。

ライブマイグレーション
: 実行中の仮想マシンインスタンスを、切替時の短時間のサービス中断だけであるホストから別ホストに移動させる為の、Nova 中の機能。

M

マネジメント API
: 管理 API（admin API）の別名。

管理ネットワーク
: 管理者が使用するネットワークセグメント（パブリックインターネットからはアクセス不可）。

マニフェスト
: Swift 中でラージオブジェクトのセグメントを追跡する為に使用される。

マニフェストオブジェクト
: ラージオブジェクトのマニフェストを含む特別な Swift オブジェクト。

メンバーシップ
: Glance VM イメージとテナント間の関連付け。指定されたテナント間でイメージを共有できるようにする。

メンバーシップリスト
: Glance 中で指定された VM イメージにアクセスできるテナントの一覧を含む。

メモリーオーバーコミット
: 実行中の各インスタンスが利用可能と考えている RAM 量に基づく判断をベースにする代わりに、ホスト上の実際のメモリ使用量をベースにした、新しい VM インスタンスを起動する機能。

メッセージブローカー
: Nova 中で AMQP メッセージ機能を提供する為に使用されるソフトウェアパッケージ。デフォルトは RabbitMQ。

メッセージバス
: Nova 中でクラウド間通信用の全 AMQP メッセージに使用される主要な仮想通信線。

メッセージキュー
: クライアントからのリクエストを適切なワーカーに渡し、ジョブが完了した際にクライアントに出力を返す。

マイグレーション
: VM インスタンスをあるホストから別のホストに移動させる処理。

マルチ NIC
: 各仮想マシンインスタンスが複数の VIF を持つことを可能にする Nova の機能。

N

ネットワーク ID
: Neutron 中で各ネットワークセグメントに割り当てられた一意な ID。

ネットワークマネージャー
: ファイアウォールルール、IP アドレス割り当て等の様々なネットワークコンポーネントを管理する Nova のコンポーネント。

ネットワークノード
: network ワーカーデーモンを実行する任意の Nova ノード。

ネットワークセグメント
: Neutron 中の仮想の独立した OSI レイヤ２サブネットを表現する。

ネットワーク UUID
: Neutron ネットワークセグメントの一意な ID。

ネットワークワーカー
: nova-network ワーカーデーモン。起動中の Nova インスタンスに IP アドレスを付与する等のサービスを提供する。

非永続ボリューム
: エフェメラルボリュームの別名。

nova
: Compute サービスを提供する OpenStack プロジェクト。

nova API
: nova Compute API の別名。

nova-network
: IP アドレス割り当て、ファイアウォール、その他ネットワーク関連タスクを管理する Nova コンポーネント。

O

オブジェクト
: Swift が保持する BLOB データ。フォーマットは任意。

オブジェクト API
: swift オブジェクト API の別名。

オブジェクトオーディター
: あるオブジェクトサーバー用の全オブジェクトを開き、各オブジェクトの MD5 ハッシュ、サイズ、メタデータを検証する。

オブジェクト有効期限
: 指定された時間経過後、又は指定日になった際に自動的にオブジェクトを削除するための Swift 中の設定オプション。

オブジェクトハッシュ
: Swift オブジェクトの一意な ID。

オブジェクトパスハッシュ
: リング中でオブジェクトの配置を決定する為に Swift が使用する。オブジェクトとパーティションの対応表。

オブジェクトレプリケーター
: フォールトトレラント用にオブジェクトをリモートパーティションにコピーする Swift のコンポーネント。

オブジェクトサーバー
: オブジェクト管理に責任を持つ Swift コンポーネント。

Object サービス API
: swift オブジェクト API の別名。

オブジェクトストレージ
: 結果整合性（eventually consistent）、ストレージ冗長化、静的デジタルコンテンツ取得、といった機能を提供する。

オブジェクトバージョニング
: コンテナ中の全オブジェクトがバージョニングしている事を Swift コンテナ上でユーザが示す為のフラグ設定を可能にする。

運用者
: OpenStack インストールを計画し、管理する責任者。

P

親セル
: 要求されたリソース（CPU時間、ディスクストレージ、メモリ）が親セルで利用不可の場合、そのリクエストは紐付けられた子セルに転送される。

パーティション
: オブジェクトを保存し、デバイスの上位に位置し、フォールトトレラント用にレプリケーションされる、Swift 中のストレージ単位。

パーティションインデックス
: リング中で全 Swift パーティションの位置を含む。

パーティションシフト値
: どのパーティションデータが配置されるべきかを決定する為に Swift に使用される。

ポーズ
: VM の変更が発生しない VM 状態（メモリ変更なし、ネットワーク通信停止、他）。VM は停止しているがシャットダウンしていない。

永続ボリューム
: 単独の仮想マシンインスタンスのライフタイムの後に永続するディスクボリューム。対比：一時ストレージ（ephemeral storage）

プラグイン
: Neutron API 用、又は Compute API 用の実際の実装を提供するソフトウェアコンポーネント。文脈に依存する。

ポリシーサービス
: ルール管理インターフェースとルールベースの認可エンジンを提供する Keystone コンポーネント。

ポート
: neutron で定義される仮想ネットワークポート。仮想インターフェース / 仮想 NIC がポートに接続される。

ポート UUID
: neutron ポートの一意な ID。

preseed
: Debian ベースの Linux ディストリビューションでシステム設定とインストールの自動化ツール。

プライベートイメージ
: 特定のテナントだけが利用できる glance の VM イメージ。

プロジェクト
: Nova におけるユーザーの論理的なグループ。仮想マシンイメージに対するクォータやアクセス権を定義するために使用される。

プロジェクト ID
: nova におけるユーザー定義の英数字文字列。プロジェクトの名前。

プロジェクト VPN
: cloudpipe の別名。

プロキシノード
: Swift プロキシサービスを提供するノード。

プロキシサーバー
: Swift のユーザーは、リング中にあるリクエストされたデータの場所を参照してユーザに結果を返すプロキシサーバーを介して Swift サービスに通信を行う。

パブリック API
: サービス間の通信とユーザとのやり取りの両方に使用される API エンドポイント。

パブリックイメージ
: 全テナントが利用可能な Glance VM イメージ。

パブリック IP アドレス
: エンドユーザがアクセス可能な IP アドレス。

パブリックネットワーク
: compute サーバーがパブリックネットワークと相互通信できるよう、ネットワークコントローラーが仮想ネットワークを提供する。全マシンにはパブリックとプライベートのネットワークインターフェースがなければならない。パブリックネットワークインターフェースは public_interface オプションにより制御される。

Puppet
: OpenStack をサポートする構成管理ツール。

Python
: OpenStack において幅広く使用されるプログラミング言語。

Q

neutron
: OpenStack のコアプロジェクトで、OpenStack Compute に対してネットワーク接続の抽象化レイヤーを提供する。

neutron API
: neutron へのアクセスに使用する API で、独自のプラグインが作成できる拡張性を持ったアーキテクチャになっている。

neutron マネージャー
: nova と neutron を統合し、neutron が nova VM のネットワークを管理できるようにする。

neutron プラグイン
: neutron 内部のインタフェースで、QoS、ACL、IDS といった先進的な機能を持った独自のプラグインを作成できるようになっている。

隔離
: swift が壊れたオブジェクト、コンテナー、アカウントを見つけた際に、そのデータはこの状態にセットされる。この状態にセットされたデータは、複製されず、クライアントが読み出すこともできなくなり、正しいコピーが再複製される。

Quick EMUlator (QEMU)
: OpenStack がサポートするハイパーバイザーの一つ。一般に、開発目的で使用される。

クォータ
: プロジェクト単位に使用できるリソース上限を設定できる nova の機能。

R

RAM フィルター
: RAM オーバーコミットの有効／無効を制御する nova の設定。

RAM オーバーコミット
: 実行中の各インスタンスが利用可能と考えている RAM 量に基づく判断をベースにする代わりに、ホスト上の実際のメモリ使用量をベースにした、新しい VM インスタンスを起動する機能。

レートリミット
: アカウント単位、コンテナー単位のデータベースの書き込みレートの上限を指定する、swift の設定オプション。

リバランス
: リングに登録されたすべてのドライブに swift のパーティションを分散させる処理。最初にリングを作成する際やリングの設定を変更した後に実行される。

recon
: メトリックス（品質）の収集に使われる swift のコンポーネント。

レコード ID
: 変更が行われる度に増加するデータベース内の数値。 swift が複製を行う際に使用する。

レジストリサーバー
: VM イメージのメタデータ情報をクライアントに提供する glance サービス。

レプリカ
: swift のオブジェクト、アカウント、コンテナーのコピーを作成することで、データ冗長性や耐障害性を実現する。これにより、バックエンドのストレージが故障した場合でもデータは失わない。

レプリカ数
: swift リングでのデータのレプリカ数。

レプリケーション
: 別の物理デバイスにデータをコピーする処理。耐障害性や性能のために行われる。

レプリケーター
: オブジェクトレプリカの作成、管理を行う swift のバックエンドプロセス。

リクエスト ID
: nova に送られた各リクエストに割り当てられる一意な ID。

リング
: swift データのパーティションへのマッピングを行う。アカウント、オブジェクト、コンテナーというサービス単位に別々のリングが存在する。

リングビルダー
: swift のリングの作成、管理を行い、パーティションのデバイスへの割り当てを行い、他のストレージノードに設定を転送する。

ロール ID
: keystone のロール毎に割り当てられる英数字の ID。

rootwrap
: 特権を持たない「nova」ユーザーが指定されたリストにあるコマンドを Linux root ユーザーで実行できるようにする nova の機能。

RPC ドライバー
nova が利用するメッセージキューソフトウェアを変更できるようにする仕組み。例えば、 RabbitMQ を ZeroMQ や Qpid に変更できる。

S

S3
Amazon のオブジェクトストレージサービス。 swift に似た機能を持つ。glance の VM イメージのバックエンドストレージとして利用できる。

スケジューラーマネージャー
どこで VM インスタンスを起動するかを決定する nova のコンポーネント。様々なスケジューラーが利用できるようにモジュラー型の設計になっている。

スコープ付きトークン
特定のテナントと関連付けされた keystone API アクセストークン。

シークレットキー
ユーザーだけが知っている文字列。 nova API へのリクエスト時にアクセスキーとともに使用される。

セキュリティグループ
nova インスタンスに適用される、ネットワークトラフィックのフィルタルールの集合。

分割オブジェクト
複数に分割された swift のラージオブジェクト。再構成されたオブジェクトは「concatenated object」（連結オブジェクト）と呼ばれる。

サーバーイメージ
VM イメージの別名。

サーバー UUID
各 nova 仮想マシンインスタンスに割り当てられた一意な ID。

サービスカタログ
keystone カタログの別名。

サービス ID
各サービスに割り当てられる一意な ID。 keystone カタログで使用される。

サービス登録
nova などのサービスをカタログに自動的に登録する keystone の機能。

サービステナント
カタログリストにあるすべてのサービスが所属する特別な keystone テナント。

サービストークン
: 管理者が定義するトークンで、 nova が keystone と安全に通信するのに使用される。

セッションバックエンド
: Horizon でクライアントセッションの追跡に使用される、ローカルメモリ、クッキー、データベース、memcached などのストレージ。

セッション持続性
: 負荷分散サービスの機能。そのノードがオンラインである限り、あるサービスへのそれ以降の接続を同じノードにリダイレクトする。

セッションストレージ
: クライアントセッションの保持と追跡を行う Horizon のコンポーネント。Django のセッションフレームワークを用いて実装されている。

共有ストレージ
: 同時に複数のクライアントからアクセス可能なブロックストレージ。 NFS など。

SmokeStack
: OpenStack のコア API に対して自動テストを実行する。 Rails で書かれている。

スナップショット
: OpenStack ストレージボリュームやイメージの、ある時点でのコピー。ストレージのボリュームスナップショットは、ボリュームをバックアップするために使用する。イメージスナップショットは、データのバックアップを行ったり、新しいサーバー用の「ゴールド」イメージ（設定済みイメージ）としてバックアップしたりするのに使用する。

分散優先スケジューラー
: 新しい VM を最も負荷が低いホストで起動しようとする、nova の VM スケジューリングアルゴリズム。

SQLAlchemy
: OpenStack で使われている、オープンソースの Python 用 SQL ツールキット。

SQLite
: 軽量 SQL データベース。多くの OpenStack サービスでデフォルトの永続ストレージとして使用されている。

StackTach
: nova の AMQP 通信のキャプチャーを行うコミュニティプロジェクト。デバッグに役立つ。

静的 IP アドレス
: 固定 IP アドレスの別名。

StaticWeb
: コンテナーのデータを静的なウェブページとして扱う swift の WSGI ミドルウェア。

ストレージバックエンド
: サービスが永続ストレージで使用する方式。 iSCSI、NFS、ローカルディスクなどがある。

ストレージノード
: アカウントサービス、コンテナーサービス、オブジェクトサービスを提供する swift ノード。アカウントデータベース、コンテナーデータベース、オブジェクトストレージの制御を行う。

ストレージマネージャー
: XenAPI の一つのコンポーネントで、様々な種類の永続ストレージバックエンドをサポートするための取り外し可能な（pluggable）インタフェースを提供する。

ストレージマネージャーバックエンド
: iSCSI や NFS といった、XenAPI がサポートする永続ストレージ手段。

ストレージサービス
: swift のオブジェクトサービス、コンテナーサービス、アカウントサービスの総称。

swift
: オブジェクトストレージサービスを提供する OpenStack コアプロジェクト。

swift All in One (SAIO)
: 完全な swift の開発環境を1台のVM内に作成できる。

swift ミドルウェア
: 追加機能を実現するための swift のコンポーネントの総称。

swift プロキシサーバー
: swift への出入口として動作する。ユーザを認証する役割も持つ。

swift ストレージノード
: swift のアカウントサービス、コンテナーサービス、オブジェクトサービスが動作するノード。

同期ポイント
: swift ノード間で最後にコンテナーデータベースとアカウントデータベースの同期が行われた時点。

T

TempAuth
: swift 自身が認証と認可を行えるようにする swift の認証機構。もっぱらテストや開発で使用される。

Tempest
: OpenStack コアプロジェクトの trunk ブランチに対してテストを実行するために設計された自動ソフトウェアテストスイート。

TempURL
: ユーザが一時的なオブジェクトアクセス用の URL を作成できるようにする swift のミドルウェア。

テナント
: Nova リソースへのアクセスを分離するために使用されるユーザーのグループ。Nova のプロジェクトの別名。

テナントエンドポイント
: 一つ以上のテナントと関連付けされる keystone API エンドポイント。

テナント ID
: keystone 内で各テナントに割り当てられる一意な ID。 nova のプロジェクト ID は keystone のテナント ID にマッピングされる。

トークン
: OpenStack API やリソースへのアクセスに使用される英数字文字列。

tombstone
: 削除済みの swift オブジェクトを示す印として使われる。これにより、そのオブジェクトが削除後に別のノードで更新されないことを保証する。

トランザクション ID
: 各 swift リクエストに割り当てられる一意な ID。デバッグや追跡に使用される。

U

スコープなしトークン
: keystone のデフォルトトークンの別名。

アップデーター
: キューイングされたり失敗したりした、コンテナやオブジェクトの更新要求の処理を行う swift コンポーネントの総称。

ユーザー
: keystone では、各ユーザは一つ以上のテナントに所属する。 nova では、テナントはロール、プロジェクトおよびその両方と関連付けられる。

ユーザーデータ
: インスタンス起動時にユーザが指定できる blob データ。インスタンスはこのデータに metadata サービスや config driver 経由でアクセスできる。通常、インスタンスがブート時に実行するシェルスクリプトを渡すのに使用される。

V

VIF UUID
: neutron VIF に割り当てられる一意な ID。

仮想CPU（vCPU）
: 物理 CPU をいくつかに分割し、分割された部分をインスタンスが使用する。仮想コアとも呼ばれる。

仮想マシン（VM）
: ハイパーバイザー上で動作するオペレーティングシステムインスタンス。一台の物理ホストで同時に複数の VM を実行できる。

仮想ネットワーク
: neutron の レイヤ2 ネットワークセグメント。

仮想ネットワークインタフェース（VIF）
: neutron ネットワークのポートに接続されるインタフェース。通常は仮想ネットワークインタフェースは VM に所属する。

仮想ポート
: 仮想ネットワークへの仮想インタフェースの接続ポイント。

仮想プライベートネットワーク（VPN）
: nova では cloudpipe の形で提供される。 cloudpipe では、特別なインスタンスを使って、プロジェクト毎に VPN が作成される。

仮想サーバー
: VM やゲストの別名。

仮想スイッチ（vSwitch）
: ホスト上で動作し、ハードウェアのネットワークスイッチと同じ機能や動作を行うソフトウェア。

仮想 VLAN
: 仮想ネットワークの別名。

VLAN マネージャー
: nova ネットワークマネージャーの一つで、サブネットを分割し、テナント毎に異なる VLAN を割り当て分離された Layer 2 セグメントを実現する。VLAN 毎に、インスタンスに IP アドレスの払い出しを行う DHCP サーバーが用意される。

VLAN ネットワーク
: ネットワークコントローラーは、コンピュートサーバー間、およびコンピュートサーバーとパブリックネットワークとの通信を行う仮想ネットワークを用意する。すべての物理マシンにはパブリック側とプライベート側のネットワークインタフェースが必要。VLAN ネットワークはプライベート側のネットワークインタフェースで、VLAN マネージャーの vlan_interface オプションで指定される。

VM イメージ
: イメージの別名。

VNC プロキシ
: VNC や VMRC を使って VM インスタンスのコンソールにユーザがアクセスできるようにする nova のコンポーネント。

ボリューム
: ディスクベースのデータストレージで、ほとんどの場合 iSCSI ターゲットとして表現され、拡張属性をサポートするファイルシステムを用いて構成される。永続的ストレージと一時的ストレージの両方がある。通常は、ブロックデバイスの同義語として使用される。

Volume API
: コンピュート VM 用のブロックストレージの作成、接続、接続解除を行うための API で、独立したエンドポイントとして提供される。

ボリュームコントローラー
: ストレージのボリューム操作の監視と調停を行う nova のコンポーネント。

ボリュームドライバー
: ボリュームプラグインの別名。

ボリューム ID
: nova 配下の各ストレージボリュームに割り当てられる一意な ID。

ボリュームマネージャー
: 永続ストレージボリュームの作成、接続、接続解除を行う nova のコンポーネント。

ボリュームノード
: cinder-volume デーモンが動作する nova ノード。

ボリュームプラグイン
: nova ボリュームマネージャーのプラグイン。プラグインにより新しい種類や特別な種類のバックエンドストレージのサポートが提供される。

Volume サービス API
: Block Storage API の別名。

ボリュームワーカー
: バックエンドストレージとやり取りを行い、ボリュームの作成、削除、コンピュートボリュームの作成を行う nova のコンポーネント。 nova-volume デーモンにより提供される。

W

ウェイト
: swift ストレージデバイスが、そのジョブに適したストレージデバイスを判定するのに使用する。デバイスはサイズで重み付けされる。

重み付けコスト
: nova で新しい VM インスタンスをどこで起動するかを決定するときに使用される、個々のコストの合計値。

計量 (weighing)
: ジョブ用の VM インスタンスとしてあるホストが適しているかを判定する nova プロセス。例えば、そのホストには十分なメモリがない、CPU が割り当てられすぎている、など。

ワーカー
: タスクを実行するデーモン。例えば、 nova-volume ワーカーはストレージを VM インスタンスに接続する。ワーカーは、キューを待ち受け、新しいメッセージが到着すると操作を行う。

Z

Zuul
: OpenStack 開発で使用されているツールで、変更のテストを正しい順番を保証しながら並列に実行する。