Маркелов А. А.

OpenStack.
Практическое знакомство с облачной операционной системой

Четвертое издание, дополненное и исправленное

Москва, 2018

УДК 004.738.5:004.451.9OpenStack
ББК 32.971.3
М25

М25 Маркелов А. А.

OpenStack. Практическое знакомство с облачной операционной системой / 4-е изд., доп. и исправ. – М.: ДМК Пресс, 2018. – 306 с.: ил.

ISBN 978-5-97060-652-0

Книга знакомит читателя с основными сервисами облачной операционной системы OpenStack на начало 2018 года (версия Queens). Рассмотрены вопросы интеграции OpenStack и системы работы с контейнерами Docker, программно-определяемой системы хранения данных Ceph, настройки производительности и высокой доступности сервисов. В четвертое издание добавлен материал по работе с сетью, настройками производительности и отказоустойчивости. В связи с переходом на сервис Gnocchi переработана глава, посвященная сервису телеметрии.

Издание рассчитано на ИТ-специалистов (системных и сетевых администраторов, а также администраторов систем хранения данных), желающих познакомиться с де-факто стандартом в области открытых продуктов построения облачной инфраструктуры типа IaaS – OpenStack.

УДК 004.738.5:004.451.9OpenStack
ББК 32.972.53

ISBN 978-5-97060-652-0

Оглавление

Предисловие

Благодарности

В первую очередь я хочу поблагодарить мою жену Лену за терпение и поддержку, проявленные во время написания этой книги и доработки текста для новых изданий. Также хотелось бы поблагодарить рецензентов этой книги – Антона Арапова и Артемия Кропачева, благодаря ценным замечаниям которых данная книга стала значительно лучше и избавилась от многих неточностей и опечаток.

Автор приносит благодарность читателям предыдущих изданий, приславшим свои замечания и исправления: Александру Румянцеву, Владимиру Манькову, Владимиру Белых, Илье Дорохову, Олегу Бабкину, Ильназу Тарханову и Евгению Дос.

Об авторе

Андрей Маркелов имеет более чем десятилетний опыт преподавания как авторских курсов, так и авторизированных курсов по ИТ-технологиям таких компаний, как Red Hat и Microsoft.

В настоящее время автор работает в качестве старшего менеджера по архитектуре компании Ericsson, специализируясь на облачных технологиях и инфраструктуре виртуализации сетевых функций (NFV-I). До этого работал в качестве старшего системного архитектора в компании Red Hat, а также в крупных системных интеграторах России, получив более чем десятилетний опыт продаж, проектирования и внедрения сетевых и инфраструктурных решений.

Около шестидесяти публикаций в отечественных ИТ-журналах («Системный администратор», «Linux Format», «PC Week» и др.). Кроме того, является автором книги «Certified OpenStack Administrator Study Guide», вышедшей в 2016 году в издательстве Apress.

Андрей является сертифицированным архитектором Red Hat (RHCA Level XI) с 2009 года и имеет сертификаты Red Hat в таких технологиях, как OpenStack, OpenShift, RHV, CloudForms, Ansible, Cloud Storage, настройка производительности, безопасности Linux-систем и др.

Кроме того, автор имеет сертификаты Microsoft Certified System Engineer, Sun Certified System Administrator, Novell Certified Linux Professional, Linux Professional Institute Certification (LPIC-1), Mirantis Certified OpenStack Administrator, OpenStack Foundation Certified OpenStack Administrator и *Cisco Certified Network Associate*.

Блог автора располагается по адресу http://markelov.blogspot.ru/. Его twitter-аккаунт: @amarkelov.

Рецензенты

Антон Арапов – руководит командой, ответственной за развитие инфраструктурных проектов в компании Acision. Помогает клиентам полностью раскрыть потенциал мобильных каналов передачи данных, а также является ответственным за эволюционное развитие сервисов в сетях LTE 4G и проекты, способствующие ускорению возврата инвестиций. Сегодня технологии виртуализации выступают основным средством для достижения поставленных целей. До Acision Антон руководил группой разработки виртуализации в ядре Linux в компании Red Hat. Антон является экспертом в технологиях виртуализации и их применении.

Артемий Кропачев – DevOps-архитектор в компании Bell Integrator. Артемий занимается проектированием и внедрением облачных решений и систем автоматизации инфраструктуры и процессов разработки и тестирования (DevOps), построенных на базе Linux, AWS, OpenStack, OpenShift, Chef и т. п. Артемий – сертифицированный архитектор Red Hat (RHCA Level XII) по направлениям Datacenter, Cloud и DevOps. До Bell Integrator работал системным архитектором в ICL Services.

Предполагаемая аудитория

Данная книга рассчитана на ИТ-специалистов (системных и сетевых администраторов, а также администраторов систем хранения данных), желающих познакомиться с де-факто стандартом в области открытых продуктов построения облачной инфраструктуры типа IaaS – OpenStack.

По аналогии с минимальными требованиями программного обеспечения к среде выполнения эта книга также предъявляет минимальные требования к читателю.

В первую очередь это желание разобраться с описываемым продуктом. Садясь за рукопись, я старался сделать книгу как можно более ориентированной на практическую работу. Лучше всего усваивается материал, который вы отработали собственными руками в лабораторном окружении. Единственной чисто теоретической главой в книге является первая, посвященная архитектуре продукта. Все остальные главы включают в себя обязательные практические упражнения.

Отсюда следует, что, прежде чем приступать к изучению продукта, написанного в первую очередь под Linux и для Linux, вам необходимы навыки работы с операционной системой. Умение работать в командной строке и знание базовых команд обязательны.

Также важными будут умение самостоятельно решать задачи и навык решения проблем – один из основных навыков хорошего ИТ-специалиста-практика. Автор сознательно старался сделать изложение большей части материала никак не привязанной к конкретным версиям операционной системы и дистрибутива OpenStack. Но почти наверняка к тому моменту, как книга доберется до вас, выйдет следующая версия OpenStack, и вы, естественно, захотите воспроизвести упражнения на актуальной версии. Также небольшие детали могут отличаться в зависимости от вашего любимого дистрибутива GNU/Linux, версии и дистрибутива OpenStack. В первую очередь это расположение конфигурационных файлов, особенности организации репозиториев с пакетами, названия пакетов и т. п.

Ну и, естественно, от ошибок и опечаток никто не застрахован, включая вашего покорного слугу.

О чем эта книга

Книга состоит из шестнадцати глав и знакомит читателя с основными сервисами облачной операционной системы OpenStack. Кроме того, рассмотрены вопросы интеграции OpenStack и системы работы с контейнерами Docker, программно-определяемой системы хранения данных Ceph, настройки производительности и высокой доступности сервисов.

Глава 1. Введение в OpenStack

Первая глава вводит читателя в предметную область облачных вычислений. Дается представление об облачных, горизонтально масштабируемых приложениях, их отличи-

ях от вертикально масштабируемых, традиционных приложений. Рассказывается об истории создания OpenStack, его основных компонентах и основных дистрибутивах, присутствующих в настоящий момент на рынке.

Глава 2. Настройка лабораторного окружения OpenStack

В данной главе рассказывается о подготовке виртуальных машин тестового окружения к установке компонентов OpenStack. Рассматриваются рекомендуемые настройки сети дистрибутива GNU/Linux CentOS 7, подключение репозиториев OpenStack RDO, установка и настройка требуемых для OpenStack компонентов.

Глава 3. Сервис идентификации Keystone

В этой главе обсуждаются концепции и терминология сервиса идентификации OpenStack, а также читатель знакомится с Keystone на практике. Читатель установит и настроит ключевой сервис облачной операционной системы и заложит фундамент для настройки остальных служб.

Глава 4. Сервис хранения образов Glance

Четвертая глава посвящена настройке каталога образов виртуальных машин. Читатель познакомится с концепциями и настройкой сервиса Glance. К концу отработки материала главы в лабораторном окружении должен появиться каталог образов с загруженным шаблоном для создания виртуальных машин.

Глава 5. Сервис блочного хранилища Cinder

В данной главе читатель познакомится с тем, как создать блочное хранилище, которое будут использовать запущенные экземпляры виртуальных машин. Настройка этого опционального сервиса позволит виртуальным машинам сохранять данные между перезагрузками.

Глава 6. Объектное хранилище Swift

В главе рассматриваются настройка и работа с одним из двух исторически первых сервисов OpenStack – объектным хранилищем Swift. Рассматриваются концепции и архитектура сервиса, а также его установка и настройка.

Глава 7. Контроллер и вычислительный узел Nova

В данной главе рассматриваются установка одного из самых важных сервисов OpenStack, непосредственно зани-

мающихся управлением виртуальными машинами. В ходе практических упражнений мы добавим два вычислительных узла в наше лабораторное окружение.

Глава 8. Службы сети Neutron

В главе, посвященной сетевым службам, рассматриваются концепции программно-определяемой сети. Читатель познакомится с вариантами организации сетевого взаимодействия в облаке и настроит на сетевом и вычислительном узле сервис Neutron.

Глава 9. Работа с виртуальными машинами из командной строки

К девятой главе читатель построит свое минимальное лабораторное окружение OpenStack. В этой главе мы рассмотрим, как на практике работать с сетями и виртуальными машинами.

Глава 10. За фасадом Neutron

В этой главе кратко рассматриваются «внутренности» сетевого сервиса OpenStack, а также такие темы, как группы безопасности и виртуальный коммутатор Open vSwitch.

Глава 11. Веб-панель управления Horizon и работа пользователя из графического интерфейса

Одиннадцатая глава, вероятно, самая дружественная к читателю. В ней мы, наконец, встанем на место пользователя облачного сервиса и познакомимся с тем, как в графическом интерфейсе веб-консоли создать проект и запустить виртуальную машину.

Глава 12. Сервис мониторинга Ceilometer

Сервис Ceilometer представляет собой централизованный источник информации по метрикам облака и данным мониторинга. Этот компонент обеспечивает возможность биллинга для OpenStack. В данной главе читатель познакомится с настройкой Ceilometer и тем, как снимать и использовать данные телеметрии OpenStack.

Глава 13. Сервис оркестрации Heat

В этой главе рассказывается о сервисе Heat – «кольце всевластия», призванном связать все компоненты облака OpenStack воедино. Читатель познакомится с шаблоном

формата HOT, при помощи которого Heat может создавать большинство типов ресурсов (виртуальные машины, тома, плавающие IP, пользователи, группы безопасности и т. д.) как единое целое, обеспечивая управление жизненным циклом приложения в облачной инфраструктуре.

Глава 14. Контейнеры и OpenStack
В четырнадцатой главе мы кратко познакомимся с технологией контейнеров и тем, как она используется совместно с OpenStack.

Глава 15. Программно-определяемая система хранения данных Ceph
Рассматриваются установка и интеграция программной СХД Ceph, которая становится для OpenStack «выбором по умолчанию» во многих новых инсталляциях.

Глава 16. Отказоустойчивость и производительность OpenStack
В этой главе автор дает обзорный материал о том, какими методами обеспечивается высокая доступность сервисов OpenStack, и рассматривает вопросы настройки производительности.

Что нового во втором, третьем и четвертом изданиях?

Текст книги расширен и обновлен, чтобы соответствовать актуальным версиям рассматриваемых проектов и компонент OpenStack на начало 2018 года.

Ниже приведены некоторые изменения четвертого издания, по сравнению с третьим:

- Обновлены главы, посвященные основным сервисам OpenStack, в соответствии с изменениями на начало 2018 года (версия Queens).
- Добавлен материал в главах, посвященных работе с сетью, настройками производительности и отказоустойчивости.
- Переработана глава, посвященная сервису телеметрии, в связи с переходом на сервис Gnocchi.
- Примеры большинства команд обновлены с использованием клиента командной строки openstack.

Изменения третьего издания, по сравнению со вторым:

- Обновлены главы, посвященные основным сервисам OpenStack, в соответствии с изменениями на начало 2017 года (версия Newton).
- Переработана глава, посвященная сервису идентификации Keystone.
- Расширены главы, посвященные сервису сети Neutron и порядку работы с виртуальными машинами из командной строки.
- Значительно переработана глава, посвященная сервису телеметрии, в соответствии с тем, что сервис оповещения (Aodh) выделен в отдельный проект.

Изменения второго издания, по сравнению с первым:

- Добавлена глава, посвященная программно-определяемой системе хранения данных Ceph и использованию Ceph совместно с OpenStack.
- Расширен и переработан материал по работе с виртуальными машинами и сетью (агрегация узлов, зоны доступности, живая миграция, создание образов виртуальных машин, работа с сетью и многое другое). С целью лучшей структуризации материал разбит на две отдельные главы.
- При описании настройки тестового окружения разделены управляющий, сетевой и вычислительный узлы, что позволяет нагляднее познакомиться с типичными ролями серверов при развертывании OpenStack.
- Значительно переработан материал по работе сети в OpenStack.
- Вопросы производительности и отказоустойчивости сервисов OpenStack выделены в отдельную главу.
- Добавлено более пятнадцати новых иллюстраций и снимков с экрана.

Что вам необходимо помимо книги

Поскольку книга называется «Практическое введение», автор подразумевает, что читатель будет не просто читать книгу, а, следуя

за изложением материала, воспроизводить представленные примеры в лабораторном окружении. Помимо самой книги, вам понадобится персональный компьютер с минимальным объемом оперативной памяти 8 Гб (желательно 16 Гб) и системой виртуализации по вашему выбору. Автор во время написания книги и тестирования примеров использовал Ubuntu Linux с гипервизором KVM, но эта информация приведена лишь в качестве примера. Любая современная система виртуализации поддерживает GNU/Linux – фундамент облака OpenStack. Конечно, никто не мешает использовать и физическое «железо».

Вам также потребуется доступ в Интернет для обращения к репозиториям с обновлениями и пакетами OpenStack, а также скачанный ISO-образ дистрибутива CentOS 7.

Глава 1

Введение в OpenStack

Что такое облачная инфраструктура?

Согласно Wikipedia, OpenStack – это свободная и открытая платформа для облачных вычислений. Для начала определимся с тем, что такое облачная платформа. Устоявшимся в индустрии определением является определение, данное National Institute of Standards and Technology (NIST):

Облачные вычисления – это модель предоставления широкодоступного, удобного доступа по сети к общему пулу настраиваемых вычислительных ресурсов по требованию (к таким, как сети, серверы, системы хранения данных, приложения и сервисы). Эти ресурсы оперативно выделяются и освобождаются при минимальных усилиях, затрачиваемых заказчиком на организацию управления и на взаимодействие с поставщиком услуг.

Этой модели присущи пять основных характеристик, три сервисные модели и четыре модели внедрения. В число характеристик входят: самообслуживание, универсальный доступ по сети, общий пул ресурсов, эластичность и учет потребления.

Сервисные модели различаются по границе контроля потребителем услуг предоставляемой инфраструктуры и включают в себя:

- инфраструктуру как сервис (IaaS) – собственно, этой сервисной модели и посвящена данная книга, поскольку OpenStack используют в основном именно для развертывания облаков этого типа. В данном случае пользователь получает контроль за всеми уровнями стека программного обеспечения, лежащими выше облачной платформы, а именно: виртуальными машинами, сетями, выделенным пользователю объемом пространства на системе хранения данных (СХД). В этом случае пользователь выступает администратором операционной системы и всего, что работает поверх, вплоть до

приложений. Примерами платформ, обеспечивающих подобную модель, помимо OpenStack, можно назвать Apache CloudStack, Eucalyptus и OpenNebula;

- программное обеспечение как сервис (SaaS) – в этом случае граница контроля пользователя – само приложение. Пользователь в данном случае может даже не знать, что такое виртуальная машина или операционная система, он просто работает с приложением. Примеры таких облачных продуктов: Google Docs, Office 365 или, например, Яндекс-почта.

Четыре модели внедрения облачной платформы включают в себя:

- платформу как сервис (PaaS) – облако, построенное по такой модели, вполне может располагаться «внутри» облака модели IaaS. В этом случае граница контроля пользователя лежит на уровне платформы построения приложений, например сервера приложения, библиотек, среды разработки или базы данных. Пользователь не контролирует и не администрирует виртуальные машины и операционные системы, установленные на них, СХД и сети. Примеры облачных платформ модели PaaS: Apache Stratos, Cloud Foundry, Deis и OpenShift Origin;
- частное облако – вся инфраструктура развернута в центре обработки данных (ЦОД) и служит подразделением одной компании или группы компаний;
- публичное облако – заказчиком облачных услуг может выступать любая компания или даже частное лицо. Это модель внедрения, на которой зарабатывают провайдеры облачных услуг;
- облако сообщества, или общественное облако. Модель, при которой потребителем является сообщество потребителей из организаций, имеющих общие задачи (например, миссии, требований безопасности, политики и соответствия различным требованиям);
- гибридное облако – это комбинация из двух или трех вышеописанных облаков, где разная нагрузка может располагаться как в частном, публичном или общественном облаке. Как правило, гибридное облако – это больше, чем просто сумма облаков, поскольку ему требуются механизмы и инструменты централизованного управления, распределения и миграции нагрузки между облачными инфраструктурами.

Что такое облачные приложения?

За время работы в компаниях, занимающихся как внедрением, так и разработкой облачных решений, автор столкнулся с тем, что потенциальные заказчики зачастую плохо представляют себе разницу в области применения облачной платформы, подобной OpenStack, и традиционных систем виртуализации. Очень часто, когда заказчик считал, что ему нужен OpenStack, на самом деле ему нужна была традиционная система промышленной виртуализации типа VMware vSphere или Microsoft Hyper-V, ориентированная не на облачные приложения, масштабируемые горизонтально, а на традиционные промышленные приложения, масштабируемые вертикально.

Проще всего объяснить различия между облачными и традиционными приложениями можно на примере аналогии.

С одной стороны, есть домашние животные – собака или кошка. Они долго живут с вами, вы знаете, как их зовут, и их характеры. Если домашнее животное болеет, вы ведете его к ветеринару. Смерть домашнего питомца будет для вас трагедией.

С другой стороны, есть стадо, например коров. Вы знаете их общее число. Они не имеют индивидуальных имен и все взаимозаменяемы. Несчастный случай с одной коровой не означает того, что вы перестали быть ковбоем и потеряли стадо.

Традиционные приложения, например база данных или почтовый сервер, – это «домашние питомцы», они требуют функционала «системы промышленной виртуализации». Этот функционал включает в себя, но не ограничивается средствами обеспечения высокой доступности, живой миграции, резервного копирования, возможность добавлять в виртуальную машину ресурсы или забирать их. Жизненный цикл таких виртуальных машин – как правило, годы.

Современные облачные приложения – это аналог «стада» виртуальных машин. Они масштабируются горизонтально, добавлением виртуальных машин. Приложение пишется таким образом, что каждая виртуальная машина сама по себе не является критичной для функционирования всего приложения и не требует высокой доступности. Также вполне возможно, что виртуальные машины работают без сохранения состояния, что не требует их резервного копирования. Жизненный цикл подобных виртуальных машин – как правило, месяцы.

Именно для обеспечения работы облачных приложений в первую очередь и создавался OpenStack. В связи с этим не удивляйтесь, что в составе функционала OpenStack отсутствует, например, высокая доступность виртуальных машин.

История OpenStack

Проект по разработке облачной операционной системы OpenStack появился в июне 2010 года как проект, объединивший разработку Национального космического агентства США (NASA) для создания виртуальных серверов Nova и программную систему хранения данных Swift от американского же хостинг-провайдера Rackspace. Первая версия под кодовым названием Austin вышла в октябре того же года.

По соглашению версии обозначаются именем, а порядковый номер в латинском алфавите первой буквы имени определяет номер версии Openstack: A – первая версия, B – вторая и т. д.

Уже в Bexar вдобавок к Nova и Swift появился третий сервис, предназначенный для хранения образов Glance. В Exssex появились веб-консоль управления Horizon и сервис идентификации Keystone. В Folsom – сервис сети, первоначально названный Quantum, но затем поменявший имя, так как оно совпадало с зарегистрированной торговой маркой, и сервис блочного хранения Cinder. В Havana добавился сервис оркестрации Heat и мониторинга Celiometer.

Важно понимать, что сам по себе OpenStack – это проект по разработке. Сайт проекта Openstack.org не предоставляет эталонного дистрибутива. Напротив, вендоры на основе кода проекта OpenStack создают свои дистрибутивы. Обзор некоторых из них приведен в разделе «Дистрибутивы и вендоры OpenStack».

Нужно отметить, что уже через год после выхода Austin к разработке OpenStack присоединились именитые ИТ-вендоры, как то: Dell, HP и Cisco.

В настоящий момент OpenStack реализуется под руководством OpenStack Foundation с числом индивидуальных членов около семидесяти тысяч и корпоративных – более шести сотен. OpenStack поддерживают практически все ИТ-лидеры рынка. Бюджет OpenStack Foundation – более шестнадцати миллионов долларов в год.

Согласно отчету Linux Foundation, на настоящий момент OpenStack – это более 20 миллионов строк кода. Основной язык

программирования – Python (71% от всего объема кода). Сам код распространяется под лицензией Apache 2.0.

К числу компаний, которые имеют самые большие инсталляции, относятся Bluehost, Canonical, CloudScaling, EasyStack, eNovance (куплена Red Hat), HP, IBM, Metacloud, Mirantis, Oracle, Piston, Rackspace, Red Hat, SUSE, SwiftStack.

Если оценивать вклад компаний в разработку проекта OpenStack, то проще всего обратиться к сайту http://stackalytics.com. Этот сервис изначально был создан компанией Mirantis для сбора статистики и вклада своих инженеров в проект и его отдельные части, а затем стал использоваться большинством компаний, разрабатывающих OpenStack.

В качестве примера на рис. 1.1 приведена диаграмма, показывающая вклад компаний в проект OpenStack за все время его существования (по состоянию на март 2018 года). Как мы видим, в пятерку топ-контрибуторов входят Red Hat, Mirantis, Rackspace, IBM и HP/HPE.

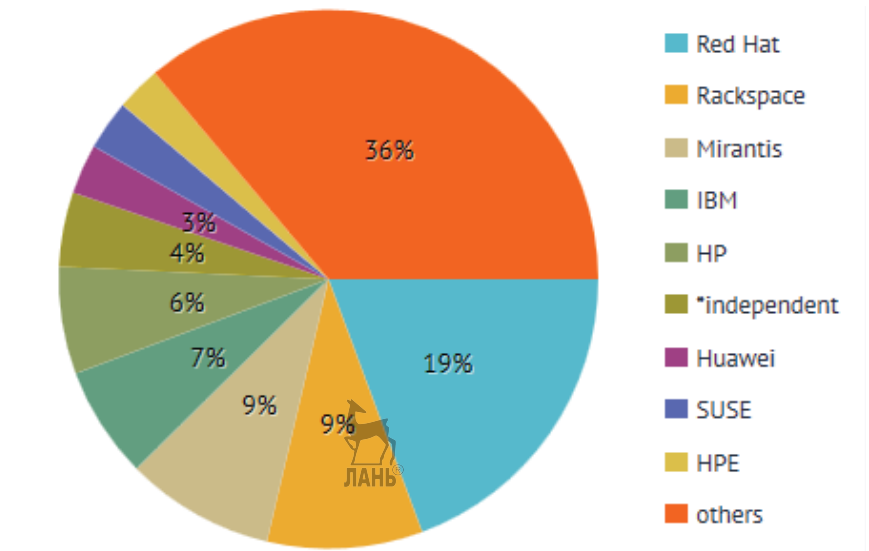


Рис. 1.1. Статистика в сервисе Stackalitics на март 2018 года

Также интересным, с точки зрения трендов, в использовании OpenStack можно считать регулярные срезы с опросника пользователей с сайта проекта https://www.openstack.org/user-survey. На момент написания книги последним был опрос от ноября 2017 го-

да. Скорее всего, после выхода книги будет доступна информация по следующему срезу статистики. Какую же интересную информацию оттуда можно получить?

- 63% принявших участие в опросе используют OpenStack для размещения производственной нагрузки;
- подавляющее большинство инсталляций – в IT-компаниях (55%) и телекоммуникационных компаниях (15%);
- Азия вышла на первое место в качестве региона проживания принявших участие в опросе (33% всех принявших участие). На втором месте – Северная Америка (33%);
- 85% всех внедрений (промышленных и тестовых) используют гипервизор KVM;
- также интересной является статистика по средам развертывания и управления облака: 20% – Puppet, 38% – Ansible, 11% – Fuel, 14% – Chef;
- 57% используют программную СХД Ceph;
- дистрибутивы ОС распределились следующим образом: 38% – Ubuntu, 31% – CentOS, 18% – RHEL.

Согласно отчету Forrester Research (http://www.openstack.org/assets/pdf-downloads/OpenStack-Is-Ready-Are-You.pdf), OpenStack в настоящее время используют многие компании из списка Fortune 100, такие как BMW, Disney и Wal-Mart.

Ну и, наконец, перед тем как двигаться дальше, возможно, читателю будет интересно ознакомиться с порталом https://www.openstack.org/enterprise/, где приведены примеры промышленной эксплуатации OpenStack на предприятиях.

Архитектура OpenStack

Проект OpenStack, который также называют облачной операционной системой, состоит из ряда отдельных проектов, разрабатывающих отдельные подсистемы. Конкретная установка OpenStack может включать в себя лишь часть из них. Некоторые подсистемы могут использоваться вообще автономно или как часть других OpenSource-проектов. В этой книге рассматривается лишь часть базовых сервисов. Их набор увеличивается от версии к версии проекта OpenStack как за счет появления новых, так и за счет разделения функционала существующих. Например, сервис nova-volume выделился в отдельный проект Cinder.

Каждый из проектов имеет свой документированный набор REST API, утилит командной строки и «родные» интерфейсы Python, предоставляющие набор функций, аналогичных утилитам командной строки.

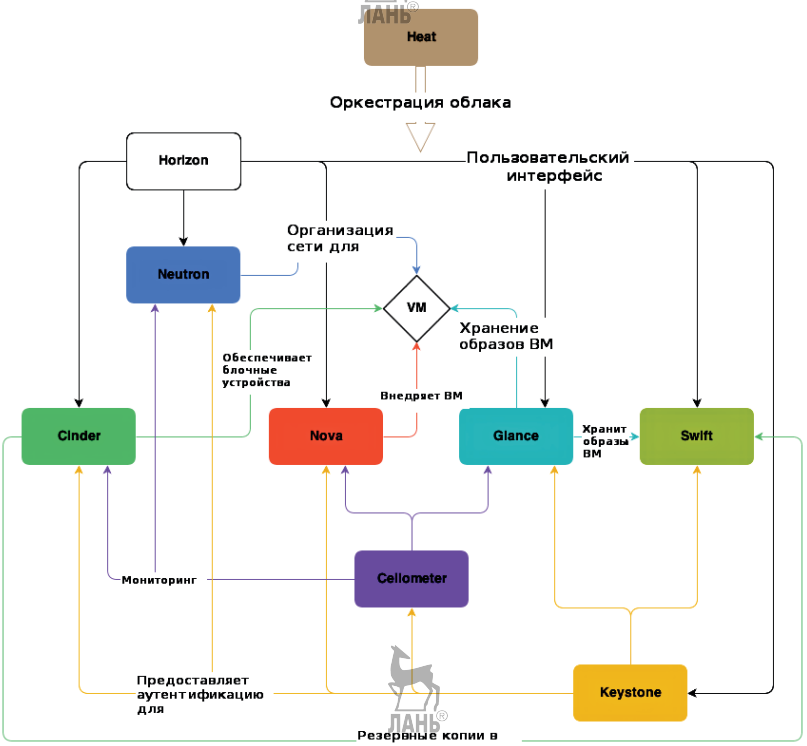


Рис. 1.2. Архитектура OpenStack

Одним из базовых сервисов является **OpenStack Compute (Nova)**. Этот сервис устанавливается на всех вычислительных узлах кластера. Он предоставляет уровень абстракции виртуального оборудования (процессоры, память, блочные устройства, сетевые адаптеры). Nova обеспечивает управление экземплярами виртуальных машин, обращаясь к гипервизору и отдавая такие команды, как, например, их запуск и остановку.

Важно отметить, что технологии OpenStack независимы от гипервизора. По адресу https://wiki.openstack.org/wiki/HypervisorSupportMatrix располагается матрица совместимости гипервизоров. Поддержка реализуется через соответствующие

драйверы в проекте Nova. Разработка и тестирование OpenStack ведутся в первую очередь для KVM. Большинство внедрений также завязано на гипервизор KVM.

Следующий сервис под названием **OpenStack Networking (Neutron)** отвечает за сетевую связанность. Пользователи могут самостоятельно создавать виртуальные сети, маршрутизаторы, назначать IP-адреса. Один из механизмов, обеспечиваемых Neutron, называется «плавающие адреса». Благодаря этому механизму виртуальные машины могут получать внешние фиксированные IP-адреса. Через механизм подключаемых модулей можно реализовать такой функционал, как балансировщик сетевой нагрузки как сервис, брандмауэр как сервис и VPN как сервис.

Служба идентификации **OpenStack Keystone** представляет собой централизованный каталог пользователей и сервисов, к которым они имеют доступ. Keystone выступает в виде единой системы аутентификации облачной операционной системы. Keystone проверяет действительность учетных записей пользователей, сопоставление пользователей проектам OpenStack и ролям и в случае успеха выдает токен на доступ к другим сервисам. Также Keystone ведет каталог служб.

OpenStack Image Service (Glance) ведет каталог образов виртуальных машин, которые пользователи могут использовать как шаблоны для запуска экземпляров виртуальных машин в облаке. Также данный сервис предоставляет функционал резервного копирования и создания моментальных снимков. Glance поддерживает множество различных форматов, включая vhd, vmdk, vdi, iso, qcow2, ami и др.

Сервис **OpenStack Block Storage (Cinder)** управляет блочным хранилищем, которое могут использовать запущенные экземпляры виртуальных машин. Это постоянное хранилище информации для виртуальных машин. Можно использовать моментальные снимки для сохранения и восстановления информации или для целей клонирования. Чаще всего с cinder используют хранилище на основе Linux-серверов, однако имеются и подключаемые модули для аппаратных хранилищ.

Сервис **OpenStack Object Storage (Swift)**, помимо Nova, является одним из двух первых проектов, появившихся в OpenStack. Изначально он назывался Rackspace Cloud Files. Сервис представляет собой объектное хранилище, позволяющее пользователям хранить файлы. Swift имеет распределенную архитектуру, обеспе-

чивая горизонтальное масштабирование, а также избыточность и репликацию для целей отказоустойчивости. Swift ориентирован преимущественно на статические данные, такие как образы виртуальных машин, резервные копии и архивы.

Сервис **OpenStack Telemetry (Celiometer)** представляет собой централизованный источник информации по метрикам облака и данным мониторинга. Этот компонент обеспечивает возможность биллинга для OpenStack.

OpenStack Orchestration (Heat) – сервис, задача которого – обеспечение жизненного цикла приложения в облачной инфраструктуре. При помощи шаблона формата AWS CloudFormation сервис управляет остальными сервисами OpenStack, позволяя создать большинство типов ресурсов (виртуальные машины, тома, плавающие IP, пользователи, группы безопасности и т. д.). Heat при помощи данных Ceilometer также может осуществлять автоматическое масштабирование приложения. Шаблоны описывают отношения между ресурсами, и это позволяет сервису Heat осуществлять вызовы API OpenStack в правильном порядке, например сначала создать сервер, а потом подключить к нему том.

И наконец, самый близкий к пользователю облака сервис **OpenStack Dashboard (Horizon)**, позволяющий управлять ресурсами облака через веб-консоль.

Также существует проект, чей код используется большинством остальных компонентов OpenStack, – это проект, объединяющий общие библиотеки компонентов облака **OpenStack Oslo**.

Гипервизор KVM и эмулятор QEMU

Как уже отмечалось, самым используемым гипервизором совместно с OpenStack является KVM (Kernel-based Virtual Machine). KVM – часть ядра Linux с 2007 года и требует аппаратной поддержки виртуализации на серверах стандартной архитектуры (AMD-V или Intel VT-x). В настоящее время KVM также адаптирован для ряда других платформ, например PowerPC. Для эмуляции устройств ввода/вывода в GNU/Linux используется эмулятор QEMU.

Обратите внимание, что аппаратная поддержка виртуализации может быть по умолчанию выключена в BIOS. Примеры включения опций в BIOS для ПК с процессорами AMD и Intel приведены на рис. 1.3, 1.4 и 1.5.

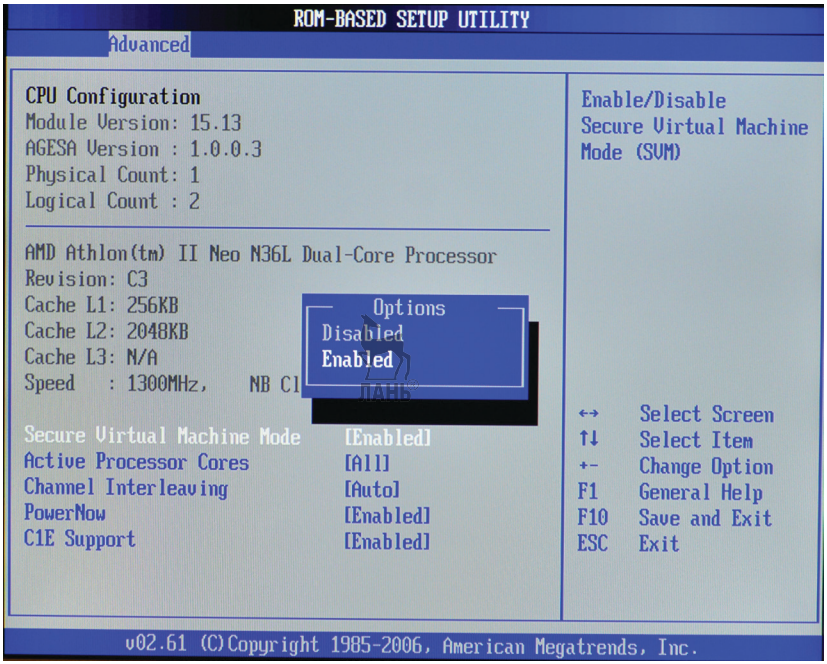


Рис. 1.3. Пример включения опции аппаратной поддержки виртуализации в BIOS (процессор AMD)

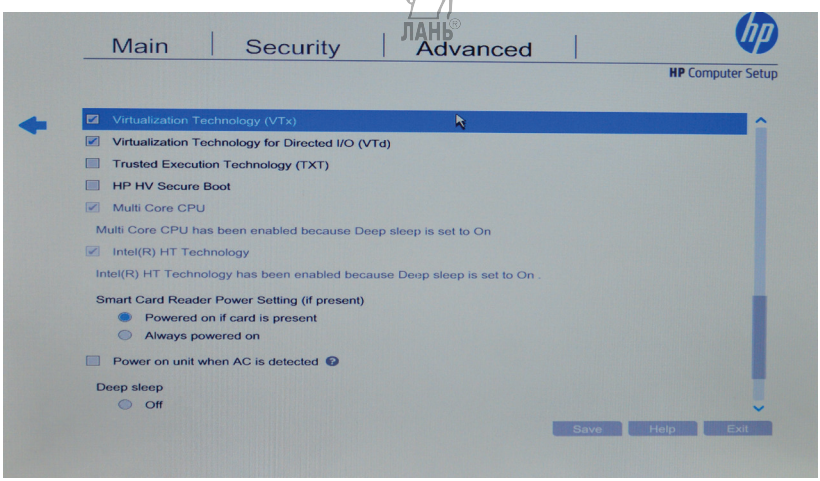


Рис. 1.4. Пример включения опции аппаратной поддержки виртуализации в BIOS ноутбука HP (процессор Intel)

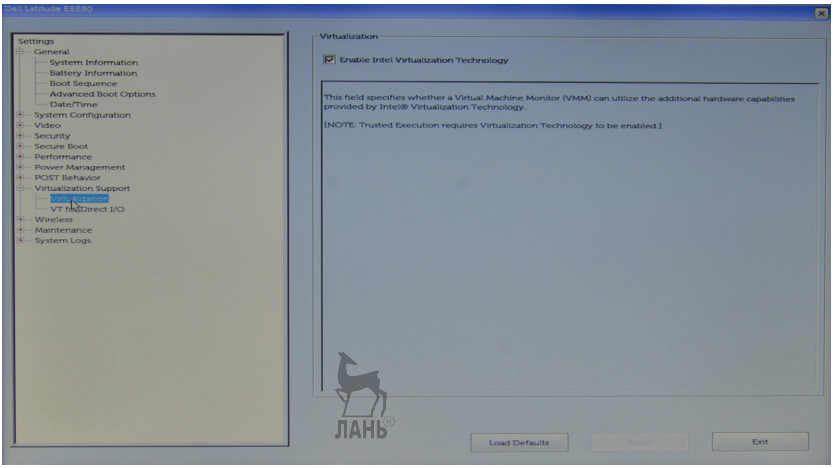


Рис. 1.5. Пример включения опции аппаратной поддержки виртуализации в BIOS ноутбука Dell (процессор Intel)

Проверить, что поддержка включена и процессор поддерживает одну из технологий, можно командой

```
$ grep -E 'svm | vmx' /proc/cpuinfo
```

Вы должны среди поддерживаемых процессором флагов увидеть `svm` или `vmx`. Также если вы дадите команду:

```
$ lsmod | grep kvm
kvm_intel              143187  3
kvm                    455843  1 kvm_intel
```

вы должны увидеть два загруженных в оперативную память модуля ядра. `kvm` – это не зависимый от производителя процессора модуль, а второй `kvm_intel` или `kvm_amd` реализует функционал VT-x или AMD-V соответственно.

Дистрибутивы OpenStack

Как уже отмечалось выше, OpenStack – это проект по созданию облачной инфраструктуры, но не продукт. Однако множество компаний, участвующих в создании OpenStack, создает на основе его кода свои продукты или дистрибутивы, зачастую используя свои проприетарные компоненты. Тут ситуация примерно аналогична созданию дистрибутивов GNU/Linux. Исходные коды в виде пакетов tar.gz доступны по адресу http://tarballs.openstack.org/.

Автор попытался дать очень краткий обзор нескольких дистрибутивов OpenStack. Обзор не претендует на всеобъемлющий охват. Также информация, приведенная ниже, актуальна на момент написания книги. Достаточно полный список основных дистрибутивов приведен в разделе Marketplace официального сайта OpenStack – https://www.openstack.org/marketplace/distros/.

Книга описывает использование дистрибутива **RDO** (https://www.rdoproject.org/), с него и начнем наш краткий обзор. RDO – спонсируемый Red Hat проект по созданию открытого дистрибутива OpenStack. В противоположность коммерческому дистрибутиву Red Hat Enterprise OpenStack Platform (RHOSP), для RDO нельзя купить поддержку. Взаимосвязь между RHOSP и RDO примерно такая же, как между Red Hat Enterprise Linux (RHEL) и Fedora. RDO призван создать сообщество вокруг разработок Red Hat. В качестве установщика изначально предлагалось использовать скрипт packstack для тестов и Foreman для промышленного развертывания. В последних версиях вместо Foreman предлагается RDO Manager, основанный на проектах OpenStack Ironic и OpenStack TripleO. RDO можно развернуть на RHEL и его производных (CentOS, Oracle Linux и других).

Второй по популярности коммерческий вендор GNU/Linux также имеет свой собственный дистрибутив OpenStack под названием **SUSE OpenStack Cloud**. В качестве дистрибутива операционной системы используется SUSE Linux Enterprise Server 12. В качестве системы хранения данных поддерживается SUSE Enterprise Storage – собственный вариант сборки программно-определяемого хранилища данных Ceph. В качестве инструмента установки используются проект Crowbar (http://crowbar.github.io) и Chef – один из распространенных инструментов управления конфигурациями в мире OpenSource.

Следующий дистрибутив – **Mirantis OpenStack (MOS)**. Как и RDO, в нем отсутствуют проприетарные компоненты, а отличительной особенностью Mirantis позиционировал систему установки Fuel, которая значительно упрощает масштабные развертывания. Также нужно отметить поддержку OpenStack Community Application Catalog, основанного на каталоге приложений Murano. В качестве дистрибутива GNU/Linux MOS требует на выбор Ubuntu или CentOS. С целью упрощения развертывания демостендов или изучения OpenStack имеются скрипты для быстрого развертывания на VirtualBox. Относительно недавно Mirantis отказался от ис-

пользования Fuel, представив новый дистрибутив Mirantis Cloud Platform.

VMware Integrated OpenStack (VIO) стоит несколько особняком от остальных рассмотренных в этом разделе, поскольку для своей работы требует развернутой проприетарной инфраструктуры VMware, включая гипервизор ESXi, платформу виртуализации сети NSX и систему управления vCenter. Все это делает VIO с учетом соответствующих лицензий относительно дорогим решением. Все компоненты OpenStack развертываются внутри виртуальных машин на ESXi в заранее созданном кластере Vmware. Поскольку в основе лежит инфраструктура VMware vSphere, то отличительным преимуществом дистрибутива будет являться «родной» функционал VMware как корпоративной системы виртуализации: HA, DRS, vMotion и т. д. Это несколько переориентирует OpenStack от облачной к традиционной нагрузке. Также VIO снижает порог вхождения для существующих инфраструктур и обученных администраторов VMware. С другой стороны, помимо необходимости приобретать лицензии на проприетарное программное обеспечение, VIO привязывает пользователя этого дистрибутива к вендору решения. Также необходимо иметь в виду, что VIO включает в себя ограниченный набор сервисов OpenStack и поддерживает только ограниченный набор подключаемых модулей и конфигураций.

Oracle OpenStack for Oracle Linux выделяется заметно недорогой, по сравнению с конкурентами, технической поддержкой в случае коммерческого использования. Он бесплатен при наличии премиальной поддержки. Из отличительных особенностей можно отметить поддержку Oracle ZFS. Также в качестве виртуальных машин поддерживается Solaris x86. Как и в случае других производителей аппаратного обеспечения, например IBM и HP, Oracle поддерживает коммерческое использование своей сборки только на своем «железе».

Ericsson Cloud Execution Environment – дистрибутив, созданный с учетом требований приложений виртуализации сетевых функций (NFV) и специфики операторов связи. В первую очередь это означает повышенную производительность сетевой подсистемы и ориентацию на приложения с требованием операций реального времени. По сравнению с дистрибутивами традиционных ИТ-компаний, ориентированный на операторов дистрибутив Ericsson Cloud Execution Environment отличается гарантированным компанией Ericsson SLA. В качестве примера функционала можно при-

вести поддержку VLAN Trunking, ускоренный при помощи библиотеки Intel DPDK виртуальный коммутатор (Ericsson Virtual Switch), мониторинг, высокую доступность виртуальных машин и многое другое. Также из отличий можно назвать свой собственный веб-интерфейс на базе Horizon. Дистрибутив создан на базе Mirantis OpenStack.

Говоря о дистрибутивах OpenStack, также необходимо упомянуть **проект OPNFV** (https://www.opnfv.org). OPNFV – это проект по построению открытой стандартной платформы для виртуализации сетевых функций (NFV). OPNFV объединяет ряд проектов, включая OpenStack, OpenDaylight, Ceph Storage, KVM, Open vSwitch и GNU/Linux. В проекте принимают участие крупнейшие телекоммуникационные компании и вендоры (AT&T, Cisco, EMC, Ericsson, HP, Huawei, IBM, Intel, NEC, Nokia, Vodafone, ZTE и многие другие). В настоящий момент проект предоставляет несколько сборок в виде ISO с различными установщиками.

Глава 2

Настройка лабораторного окружения OpenStack

На момент написания четвертого издания книги последней версией OpenStack была семнадцатая с момента появления проекта – Queens. В предыдущих трех изданиях рассматривались версии Juno, Liberty и Newton. Поскольку эта книга является введением в OpenStack, различия между релизами с точки зрения изучения OpenStack не очень велики. В книге по шагам, без использования средств автоматизации рассмотрены установка и настройка базового лабораторного окружения с использованием CentOS 7 и дистрибутива RDO.

В отличие от первого издания книги, где автор рассматривал установку большинства сервисов на одну виртуальную машину, мы будем разделять управляющий, сетевой и вычислительный узлы, что позволит нагляднее познакомиться с типичными ролями серверов при развертывании OpenStack.

С первой по пятую главу вам понадобится только машина controller минимум с двумя виртуальными процессорами и не менее чем с 4 Гб оперативной памяти. В шестой главе вам дополнительно понадобятся три виртуальные машины sw1, sw2 и sw3 для объектного хранилища Swift, каждая с 1 Гб оперативной памяти. Если ресурсы вашего стенда ограничены, то в последующих главах эти виртуальные машины можно не использовать. В седьмой главе вам необходимо будет добавить виртуальную машину compute в качестве гипервизора. Чем больше оперативной памяти вы выделите для compute, тем лучше. Для большинства примеров в книге с запуском виртуальных машин Cirros Linux вам хватит 2 Гб. Если у вас достаточно ресурсов, то вы также можете создать второй опциональный вычислительный узел compute-opt. Он нам понадобится

только для изучения живой миграции виртуальных машин. В следующей, восьмой главе добавляется виртуальная машина network для сетевых сервисов. Для нее более чем достаточно 1 Гб оперативной памяти. В пятнадцатой главе, посвященной Ceph, вам дополнительно к controller, network и compute понадобятся три виртуальные машины: ceph1, ceph2 и ceph3. Эти виртуальные машины также можно создать с 1 Гб оперативной памяти каждую.

Диски всех учебных виртуальных машин рекомендуется сделать «тонкими». В этом случае размер реально занятого места на файловой системе для каждой виртуальной машины будет от 3 до 6 Гб.

Как говорилось во введении, автор во время написания книги и тестирования примеров использовал CentOS 7 и Ubuntu Linux 14.04 LTS с гипервизором KVM (см. рис. 2.1), но эта информация приведена лишь в качестве примера.

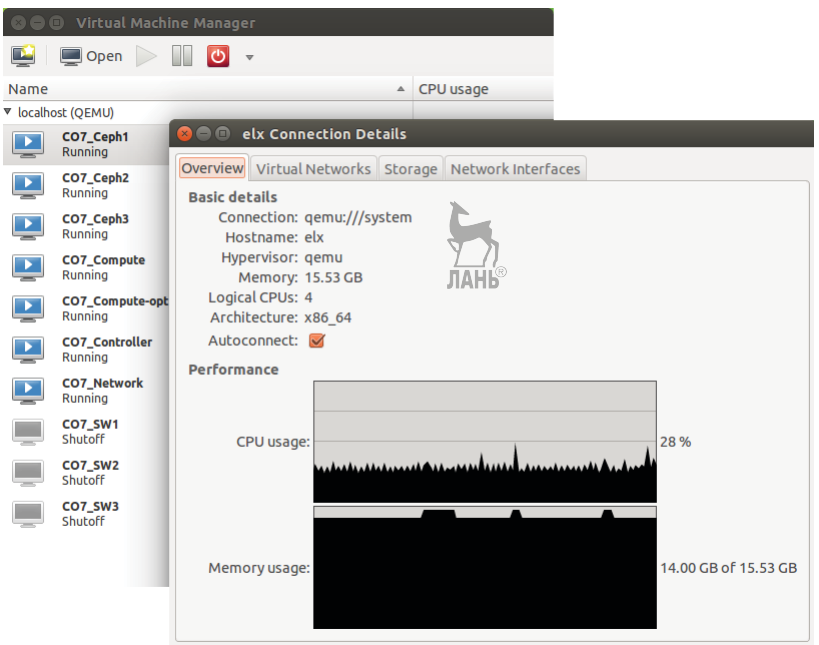


Рис. 2.1. Virtual Machine Manager с виртуальными машинами стенда в Ubuntu

Одновременно необходимо запускать от шести виртуальных машин, и для выполнения большинства упражнений книги достаточно компьютера с 8 Гб оперативной памяти, однако для комфортной работы рекомендуется 16 Гб. Подготовка операционной

системы и настройка репозиториев для всех виртуальных машин с одинаковым дистрибутивом GNU/Linux будут выполняться одинаково.

Кроме того, в табл. 1 для справки приведен список сервисов, устанавливаемых на каждый узел.

Таблица 1. Список узлов и сервисов лабораторного стенда

Узел	Сервис	Описание
Управляющий узел **controller.test.local** 192.168.122.200	rabbitmq-server	RabbitMQ – брокер сообщений протокола AMQP
	mariadb	MariaDB – база данных, используется многими сервисами OpenStack
	httpd с mod_wsgi	Keystone API, Placement API – служит для предоставления доступа к API сервиса идентификации и сервиса Placement API. Сервис Placement API появился в OpenStack версии Newton. Он отвечает за отслеживание списка ресурсов и их использование
	glance-api	Glance API – предоставляет доступ к REST API сервиса образов для поиска, хранения и получения образов
	glance-registry	Glance Registry – хранит и предоставляет образы виртуальных машин
	cinder-api	Cinder API – предоставляет доступ к REST API сервиса Cinder
	cinder-scheduler	Cinder Scheduler – сервис-планировщик Cinder
	cinder-volume	Cinder Volume – отвечает за взаимодействие с бэкэндом – блочным устройством
	cinder-backup	Cinder Backup – отвечает за создание резервных копий томов в объектном хранилище
	nova-api	Отвечает за обработку вызовов API клиентов Nova
	nova-scheduler	Сервис-планировщик. Получает из очереди запросы на запуск виртуальных машин и выбирает узел для их запуска

Узел	Сервис	Описание
	nova-conductor	Сервис выступает в качестве посредника между базой данных и nova-compute, позволяя осуществлять горизонтальное масштабирование
	nova-consoleauth	Отвечает за авторизацию для novncproxy
	nova-novncproxy	Выступает в роли VNC-прокси и позволяет подключаться к консоли виртуальных машин при помощи браузера
	neutron-server	Центральный управляющий компонент Neutron. Не занимается непосредственно маршрутизацией пакетов. С остальными компонентами взаимодействует через брокера сообщений
	openstack-aodh-evaluator	Сервис, определяющий, сработал ли триггер при достижении метриками заданных значений в течение определенного измеряемого периода
	openstack-aodh-notifier	Сервис, запускающий те или иные действия при срабатывании триггера
	openstack-aodh-listener	Сервис, определяющий, когда триггер сработает
	gnocchi-metricd	Сервис Gnocchi Metricd
	gnocchi-api	Отвечает за API сервиса Gnocchi
	ceilometer-notification	Агент, отправляющий по протоколу AMQP метрики сборщику от различных сервисов
	ceilometer-central	Агент, запускаемый на центральном сервере для запроса статистики по загрузке, не связанной с экземплярами виртуальных машин или вычислительными узлами
	ceilometer-alarm-evaluator	Сервис, проверяющий срабатывание триггеров при достижении метриками заданных значений
	ceilometer-alarm-notifier	Сервис, запускающий те или иные действия при срабатывании триггера

Узел	Сервис	Описание
	httpd	Веб-сервер для Horizon. Также может использоваться для обеспечения работы Keystone
	heat-engine	Основной сервис оркестрации, обеспечивающий обработку шаблонов и отправляющий события пользователям API
	heat-api	Сервис, отвечающий за предоставление основного REST API Heat. Сервис взаимодействует с openstack-heat-engine через вызовы RPC
	heat-api-cfn	Аналогичен предыдущему сервису, но обеспечивает работу с API, совместимым с AWS CloudFormation. Также взаимодействует с openstack-heat-engine
Сетевой узел **network.test.local** 192.168.122.220	neutron-openvswitch-agent	Openvswitch-agent – взаимодействует с neutron-server через брокер сообщений и отдает команды OVS для построения таблицы потоков
	neutron-l3-agent	Сервис обеспечивает маршрутизацию и NAT, используя технологию сетевых пространств имен
	neutron-dhcp-agent	Dhcp-agent – сервис отвечает за запуск и управление процессами dnsmasq
	neutron-metadata-agent	Metadata-agent – данный сервис позволяет виртуальным машинам запрашивать данные о себе, такие как имя узла, открытый ssh-ключ для аутентификации и др.
	neutron-lbaas-agent	Подключаемый модуль балансировщика нагрузки как сервиса
	openvswitch	Openvswitch – программный коммутатор, используемый для построения сетей. Не является проектом OpenStack, но используется им
	haproxy	Быстрый балансировщик нагрузки, написанный на языке программирования C. Используется агентом neutron-lbaas-agent

Узел	Сервис	Описание
Вычислительные узлы **compute.test.local** 192.168.122.210 **compute-opt.test.local** 192.168.122.215	nova-compute	Nova-compute – демон, управляющий виртуальными машинами через API гипервизора
	openvswitch	Openvswitch – программный коммутатор
	neutron-openvswitch-agent	Openvswitch-agent – взаимодействует с neutron-server через брокер сообщений и отдает команды OVS для построения таблицы потоков
	ceilometer-compute	Агент, запускаемый на всех вычислительных узлах для сбора статистики по узлам и экземплярам виртуальных машин
Монитор CEPH **ceph1.test.local** 192.168.122.11	ceph.service	Ceph monitors (MON) – поддерживает мастер-копию карты состояния кластера и информацию о его текущем состоянии. Все узлы кластера отправляют информацию мониторам о каждом изменении в их состоянии
Узлы хранения CEPH **ceph2.test.local** 192.168.122.12 **ceph3.test.local** 192.168.122.13	ceph.service	Object Storage Device (OSD) – отвечает непосредственно за хранение данных. Обычно один демон OSD связан с одним физическим диском
Прокси Swift **sw1.lest.local** 192.168.122.61	openstack-swift-proxy	Swift Proxy – отвечает за коммуникацию Swift с клиентами по протоколу HTTP/HTTPS и за распределение запросов на чтение и запись между узлами
Сервер хранения Swift **sw2.lest.local** 192.168.122.62	openstack-swift-account	Swift Account – отвечает за поддержку баз данных SQLite, содержащих информацию о контейнерах, доступных конкретной учетной записи
	openstack-swift-container	Swift Container – отвечает за поддержку баз данных, содержащих информацию об объектах, имеющихся в каждом контейнере
	openstack-swift-object	Swift Object – отвечает за хранение, доставку и удаление объектов

Узел	Сервис	Описание
Сервер хранения Swift **sw3.lest.local** 192.168.122.63	openstack-swift-account	Swift Account – отвечает за поддержку баз данных SQLite, содержащих информацию о контейнерах, доступных конкретной учетной записи
	openstack-swift-container	Swift Container – отвечает за поддержку баз данных, содержащих информацию об объектах, имеющихся в каждом контейнере
	openstack-swift-object	Swift Object – отвечает за хранение, доставку и удаление объектов

IP-адресацию вы можете изменить по своему усмотрению, не ориентируясь на выбор автора. Приведенные адреса выбраны лишь в качестве примера. Также убедитесь, что все взаимодействующие виртуальные машины могут разрешать имена друг друга. Можно поднять локальный DNS-сервер или, что намного проще, прописать на всех узлах имена взаимодействующих с ними лабораторных машин в файл /etc/hosts. Пример для узлов, используемых в этой книге:

```
127.0.0.1   localhost localhost.localdomain localhost4 localhost4.
localdomain4
::1         localhost localhost.localdomain localhost6 localhost6.
localdomain6
192.168.122.200 controller.test.local controller
192.168.122.210 compute.test.local compute
192.168.122.215 compute-opt.test.local compute-opt
192.168.122.220 network.test.local network
192.168.122.11 ceph1.test.local ceph1
192.168.122.12 ceph2.test.local ceph2
192.168.122.13 ceph3.test.local ceph3
192.168.122.61 sw1.test.local sw1
192.168.122.62 sw2.test.local sw2
192.168.122.63 sw3.test.local sw3
```

Подготовка CentOS 7 к использованию дистрибутива OpenStack RDO

Процесс подготовки и установки RDO на CentOS 7 не очень сильно отличается от соответствующих действий для других производных RHEL.

Первое, что необходимо, – установить саму операционную систему. Предполагается, что это не должно вызвать затруднений у читателя. В качестве варианта установки можно выбрать Minimal или Server with GUI.

После установки операционной системы обновите все установленные пакеты командой

```
# yum -y update
```

Следующее – это добавление репозиториев с пакетами OpenStack и дополнительных пакетов. Начнем с необходимого репозитория Extra Packages for Enterprise Linux (EPEL). Данный репозиторий содержит пакеты, предназначенные для RHEL и его производных, коим является CentOS. Как правило, там содержатся пакеты, которые присутствуют в Fedora, но которые компания Red Hat не включила в свой промышленный дистрибутив. В CentOS это максимально упрощено:

```
# yum -y install epel-release
```

Подключаем репозиторий дистрибутива RDO:

```
# yum install -y https://www.rdoproject.org/repos/rdo-release.rpm
```

CentOS и Fedora для управления настройками сети по умолчанию используют сервис NetworkManager. Он прекрасно подходит для настольных систем и для большинства применений в качестве средства управления сетью сервера. Однако в настоящий момент OpenStack не поддерживает работу с NetworkManager. Для корректной работы сети нам необходимо использовать традиционные скрипты network.

Отключаем сервис NetworkManager и отключаем его запуск после перезагрузки:

```
# systemctl stop NetworkManager.service
# systemctl disable NetworkManager.service
```

В конфигурационных файлах сетевых адаптеров ifcfg-eth0 и ifcfg-eth1 в директории /etc/sysconfig/network-scripts/ необходимо поправить параметры NM_CONTROLLED и ONBOOT. Также при необходимости убедитесь, что заданы статические параметры TCP/IP и правильный MAC-адрес виртуальной сетевой карты. Пример файла ifcfg-eth0 после редактирования:

```
TYPE="Ethernet"
BOOTPROTO="none"
DEFROUTE="yes"
IPV6INIT="no"
NAME="eth0"
ONBOOT="yes"
HWADDR="52:54:00:30:2F:EF"
IPADDR0="192.168.122.200"
PREFIX0="24"
GATEWAY="192.168.122.1"
DNS1="192.168.122.1"
DOMAIN="test.local"
NM_CONTROLLED=no
```

Теперь можно включить сервис `network`:

```
# systemctl start network.service
# systemctl enable network.service
```

Далее для упрощения отладки мы отключим сервис брандмауэра. Если вы не планируете отключать брандмауэр, то список используемых портов по умолчанию приведен в приложении 3 в конце книги. Начиная с CentOS 7 по умолчанию вместо `iptables` используется `firewalld`:

```
# systemctl stop firewalld.service
# systemctl disable firewalld.service
```

Важно отметить, что такая конфигурация не подходит для промышленного применения. На «боевых» узлах брандмауэр должен быть обязательно включен. При этом в любом случае в качестве брандмауэра не поддерживается `firewalld`. Необходимо использовать `iptables`. Подробнее – проконсультируйтесь с руководством по безопасности по адресу http://docs.openstack.org/security-guide/content/. Та же рекомендация относится и к настройкам системы мандатного контроля доступа SELinux. В промышленной среде при использовании производных RHEL необходимо установить пакет `openstack-selinux`:

```
# yum -y install openstack-selinux
```

Автору, в свое время потратившему много времени на пропаганду SELinux и переведшему man-страницы основных утилит SELinux для Fedora, RHEL и производных, нелегко дались эти строчки… Но для упрощения отладки в учебных целях вы можете отключить SELinux. Для этого необходимо отредактировать файл /etc/sysconfig/selinux, заменив

```
SELINUX=enforcing
```

на

```
SELINUX=disabled
```

Далее необходимо просто перезагрузить систему.

Сервисы OpenStack обычно устанавливаются на несколько отдельных узлов, и при этом все системные часы этих узлов должны быть синхронизированы. Для этого в лабораторном окружении рекомендуется использовать службу NTP. У вас не будет проблем с синхронизацией времени, если вы запускаете все виртуальные машины стенда на одном физическом сервере, однако для промышленной эксплуатации службы NTP необходимы. Множество различных проблем с функционированием сервисов OpenStack может быть вызвано различием времени на узлах, входящих в облако. При этом зачастую вы не увидите ошибок в журналах, и такие проблемы сложно диагностировать. В CetnOS 7 у вас есть выбор между использованием `ntpd` или более современной реализацией демона NTP chrony:

```
# yum -y install ntp
# systemctl start ntpd.service
# systemctl enable ntpd.service
```

или

```
# yum -y install chrony
# systemctl start chronyd
# systemctl enable chronyd
```

По умолчанию оба демона уже настроены на использование внешних источников точного времени serverX.centos.pool.ntp.org. Если у вас в организации развернут свой NTP-сервер или отсутствует доступ в Интернет с виртуальных машин стенда, то отредактируйте /etc/ntp.conf или /etc/chrony.conf.

В случае `ntpd` вам необходимо как минимум добавить в конфигурационный файл опции:

```
restrict 192.168.0.0 mask 255.255.255.0 nomodify notrap
server 127.127.1.0
fudge 127.127.1.0 stratum 10
```

На клиентах соответственно нужно исправить `server` на IP-адрес сервера NTP в локальной сети:

```
server 192.168.0.1
```

Проверить, что синхронизация времени работает с клиента, можно при помощи команды ntpq:

```
# ntpq -p
     remote           refid      st t when poll reach   delay   offset  jitter
==============================================================================
*192.168.0.1    LOCAL(0)        11 u   99 1024  377    0.275   -0.021   0.080
```

Для указания необходимости отредактировать конфигурационный файл автор в тексте книги использует утилиту `crudini` (http://www.pixelbeat.org/programs/crudini/). Имя утилиты представляет собой аббревиатуру Create, Read, Update, Delete плюс, собственно, к чему применяются данные операции. Утилита `crudini` используется только для удобства форматирования текста в книге. Вы, безусловно, для редактирования файлов можете использовать любой текстовый редактор.

Установка утилиты в CentOS производится командой

```
# yum -y install crudini
```

Синтаксис команды, который используется в примерах книги:

```
$ crudini --set /путь/к/конфигурационному/файлу СЕКЦИЯ параметр значение
```

Отличия RDO от «Upstream»

Upstream, или, как иногда говорят по-русски, используя жаргон «апстрим», «ванильный код» вследствие отсутствия короткого русского эквивалента, – это код, в котором ведется разработка, или стабилизированный вендоронезависимый код на сайте проекта. В целом RDO не содержит специфических изменений, и отличия в основном сводятся к конфигурационным файлам. Например, RDO поставляет файлы с настройками по умолчанию в /usr/share/имя_сервиса/имя_сервиса-dist.conf. Например, /usr/share/nova/nova-dist.conf или /usr/share/cinder/cinder-dist.conf.

Как установить OpenStack RDO одной командой?

Автор книги предполагает, что вы шаг за шагом пройдете вслед за автором по пути ручной установки и настройки компонентов. Это позволит вам лучше разобраться с тем, как работает OpenStack и как

связаны между собой отдельные его части. Однако если вы хотите быстро получить работающую виртуальную машину с установленными компонентами OpenStack RDO, вы можете воспользоваться утилитой Packstack. Утилита использует для своей работы Puppet и позволяет установить все компоненты на один узел. Packstack не предназначен для развертывания промышленных сред и рекомендуется только для установки лабораторного окружения.

Вам потребуется виртуальная машина, подготовленная в соответствии с инструкциями предыдущего раздела. Установите Packstack:

```
# yum install -y openstack-packstack
```

Далее можно отдать команду `packstack` с ключом `--allinone`, но лучше подготовить файл ответов, в котором можно уточнить параметры установки. Сгенерируем шаблон файла ответов:

```
# packstack --gen-answer-file ~/myfile.txt
```

Далее можно открыть полученный текстовый файл и при необходимости изменить опции установки OpenStack. Файл содержит комментарии, и в нем достаточно просто разобраться. Автор предлагает как минимум изменить следующие параметры:

```
CONFIG_DEFAULT_PASSWORD=openstack
CONFIG_KEYSTONE_ADMIN_PW=openstack
CONFIG_KEYSTONE_DEMO_PW=openstack
```

Тем самым вы укажете пароль по умолчанию для сервисов, пользователя admin и demo, а также необходимость использования репозитория EPEL. Теперь осталось выполнить команду:

```
# packstack --answer-file ~/myfile.txt
```

Через 15–30 минут, в зависимости от характеристик лабораторного окружения, все сервисы будут настроены. Во время работы утилита выдает сообщения о ходе работы:

```
Welcome to the Packstack setup utility

The installation log file is available at: /var/tmp/packstack/
20180305-124302-i2vnoU/openstack-setup.log

Installing:
Clean Up                                             [ DONE ]
```

```
Discovering ip protocol version                          [ DONE ]
Setting up ssh keys                                      [ DONE ]
...
Applying Puppet manifests                                [ DONE ]
Finalizing                                               [ DONE ]

 **** Installation completed successfully ******

Additional information:
 * Time synchronization installation was skipped. Please note
that unsynchronized time on server instances might be problem for
some OpenStack components.
 * File /root/keystonerc_admin has been created on OpenStack
client host 10.0.2.11. To use the command line tools you need to
source the file.
 * To access the OpenStack Dashboard browse to http://10.0.2.11/
dashboard .
Please, find your login credentials stored in the keystonerc_admin
in your home directory.
 * The installation log file is available at: /var/tmp/
packstack/20180305-124302-i2vnoU/openstack-setup.log
 * The generated manifests are available at: /var/tmp/
packstack/20180305-124302-i2vnoU/manifests
```

При повторном запуске Packstack пытается обновить конфигурацию. Так что в случае проблем можно просто перезапустить команду. Для подробного вывода при необходимости можно добавить опцию `-d`.

При желании можно разнести роли по разным серверам. Для указания сетевого узла используется параметр:

```
CONFIG_NETWORK_HOSTS=IP-адрес
```

Для указания вычислительных узлов – параметр:

```
CONFIG_COMPUTE_HOSTS=IP-адрес
```

В обоих случаях через запятую можно добавить несколько IP-адресов. В табл. 2 приведены некоторые другие часто используемые параметры файла ответов packstack.

Таблица 2. Параметры файла ответов packstack

Пример параметра	Описание
`CONFIG_<имя_компонента>_INSTALL=y`	Устанавливать тот или иной компонент OpenStack. Например, `CONFIG_HEAT_INSTALL=y` означает необходимость установить Heat

Пример параметра	Описание
CONFIG_DEFAULT_PASSWORD=password	Пароль по умолчанию для всех сервисов
CONFIG_NTP_SERVERS=192.168.1.1,192.168.1.2	Список NTP-серверов
CONFIG_CONTROLLER_HOST=192.168.122.200	Узел, куда устанавливаются управляющие сервисы OpenStack
CONFIG_COMPUTE_HOSTS=192.168.122.200	Узлы, куда устанавливаются сетевые сервисы OpenStack
CONFIG_NETWORK_HOSTS=192.168.122.200	Узлы, играющие роль гипервизоров
CONFIG_AMQP_BACKEND=rabbitmq	Какая служба выступает в качестве брокера AMQP. Обычно это RabbitMQ
CONFIG_USE_EPEL=y	Использовать репозиторий Extra Packages for Enterprise Linux. Необходимо задать «y», если используются производные от дистрибутива RHEL (CentOS, Oracle Linux и подобные)
CONFIG_KEYSTONE_ADMIN_PW=password	Пароль пользователя admin
CONFIG_KEYSTONE_DEMO_PW=password	Пароль пользователя demo (если установлен параметр CONFIG_PROVISION_DEMO=y)
CONFIG_GLANCE_BACKEND=file	Где хранятся образы службы Glance (допустимы варианты: file или swift)
CONFIG_CINDER_BACKEND=lvm	Что используется в качестве места хранения информации, предоставляемой сервисом Cinder (допустимы варианты: lvm, gluster, nfs, vmdk и netapp)
CONFIG_CINDER_VOLUMES_CREATE=y	Создавать ли группу LVM для Cinder. Если да, то packstack создаст файл в /var/lib/cinder и смонтирует его как loopback-устройство
CONFIG_CINDER_VOLUMES_SIZE=20G	Размер группы LVM
CONFIG_SWIFT_STORAGE_SIZE=2G	Размер loopback-файла для хранения объектов Swift
CONFIG_PROVISION_DEMO=y	Создать тестовый проект и пользователя demo

Как установить OpenStack одной командой из исходных кодов?

Второй способ предполагает установку OpenStack из исходных кодов и подходит не только для CentOS/Fedora, но и для Ubuntu, OpenSUSE и теоретически для любого дистрибутива. Для этого мы воспользуемся набором скриптов DevStack (http://devstack.org/). Обратите внимание, что минимально необходимый объем оперативной памяти для виртуальной машины – 4 Гб.

Установим Git и скачаем DevStack:

```
# yum -y install git
# git clone https://github.com/openstack-dev/devstack.git /opt/devstack/
Cloning into '/opt/devstack'...
remote: Counting objects: 35834, done.
remote: Compressing objects: 100% (21/21), done.
remote: Total 35834 (delta 8), reused 0 (delta 0), pack-reused 35813
Receiving objects: 100% (35834/35834), 10.76 MiB | 2.55 MiB/s, done.
Resolving deltas: 100% (25014/25014), done.
```

Далее при помощи скрипта `create-stack-user.sh` создадим пользователя и группу `stack`:

```
# cd /opt/devstack/
# chmod u+x tools/create-stack-user.sh
# ./tools/create-stack-user.sh
...
Creating a group called stack
Creating a user called stack
Giving stack user passwordless sudo privileges
```

Теперь изменим владельца директории и переключимся под нового пользователя:

```
# chown -R stack:stack /opt/devstack/
# sudo -i -u stack
$ cd /opt/devstack/
```

Далее нужно в директории devstack создать файл local.conf. Пример минимально достаточного файла local.conf приведен ниже:

```
[[local|localrc]]

ADMIN_PASSWORD="openstack"
SERVICE_PASSWORD="openstack"
SERVICE_TOKEN="openstack"
```

```
MYSQL_PASSWORD="openstack"
RABBIT_TOKEN="openstack"
RABBIT_PASSWORD="openstack"
SWIFT_HASH=s0M3hash1sh3r3

disable_service n-net
enable_service neutron
enable_service q-svc
enable_service q-agt
enable_service q-dhcp
enable_service q-l3
enable_service q-meta
HOST_IP=192.168.122.200

enable_service ceilometer-acompute
enable_service ceilometer-acentral
enable_service ceilometer-anotification
enable_service ceilometer-collector
enable_service ceilometer-alarm-evaluator
enable_service ceilometer-alarm-notifier
enable_service ceilometer-api

enable_service heat h-api h-api-cfn h-api-cw h-eng
enable_service s-proxy s-object s-container s-account
SWIFT_REPLICAS=1

LOGFILE=/opt/stack/logs/stack.sh.log
SCREEN_LOGDIR=/opt/stack/logs
```

Осталось только в директории devstack запустить скрипт stack.sh. В зависимости от производительности виртуальной машины и скорости доступа в Интернет установка может занять от десятка минут до получаса. Пример вывода утилиты при успешном завершении процесса:

```
=========================
DevStack Component Timing
=========================
Total runtime        4185

run_process           110
test_with_retry         5
pip_install           474
restart_apache_server  11
wait_for_service       24
yum_install           837
git_timed             350
=========================
```

```
This is your host IP address: 192.168.122.200
This is your host IPv6 address: ::1
Horizon is now available at http://192.168.122.200/dashboard
Keystone is serving at http://192.168.122.200/identity/
The default users are: admin and demo
The password: openstack
```

Как определить, какую версию OpenStack я использую?

До релиза Liberty все компоненты OpenStack, кроме Swift, имели единую версию вида «год.месяц». Например, для Kilo – 2015.1.x. Начиная с Liberty каждый компонент имеет свою версию, где первая цифра остается постоянной в рамках релиза. Определить версию установленного дистрибутива можно несколькими способами, например командами `keystone-manage` или `nova-manage` с опцией `--version`.

Пример вывода для Queens:

```
$ keystone-manage --version
2015.1.0
$ nova-manage --version
2015.1.1
```

Пример вывода для Newton:

```
$ keystone-manage --version
10.0.0
$ nova-manage --version
14.0.0
```

Пример вывода для Queens:

```
$ keystone-manage --version
13.0.0
$ nova-manage --version
17.0.0
```

В веб-интерфейсе Horizon версию можно увидеть на вкладке Admin – System Information в правом нижем углу. Пример снимка с экрана приведен в главе 11 на рис. 11.3. Сопоставление номеров версий и имен релизов можно определить по ссылкам со страницы Wiki – http://releases.openstack.org/.

Установка и настройка брокера сообщений

Вернемся к нашему стенду. В дальнейшем, рассматривая каждый сервис OpenStack, в начале главы автор будет приводить «карточку сервиса» в качестве справки. Приведем пример для RabbitMQ:

```
Название: RabbitMQ
Назначение: брокер сообщений протокола AMQP
Сайт: https://www.rabbitmq.com/
Пакет: rabbitmq-server
Имя сервиса: rabbitmq-server.service
Порты: 5672/tcp (amqp) и 5671/tcp (amqps)
Конфигурационные файлы: /etc/rabbitmq/rabbitmq.config
```

Одно из предварительных требований перед началом установки компонентов OpenStack – наличие работающего брокера сообщений. Брокер сообщений используется для координирования операций и обмена информацией между сервисами OpenStack, такими как Glance, Cinder, Nova, OpenStack Networking, Heat и Ceilometer. Как правило, брокер сообщений устанавливается на управляющем узле вместе с остальными управляющими сервисами OpenStack. Обмен сообщениями между компонентами производится по протоколу AMQP (Advanced Message Queuing Protocol). Совместно с OpenStack можно использовать несколько конкретных реализаций AMQP, например Apache Qpid или ZeroMQ.

В книге предполагается использование брокера RabbitMQ, который по умолчанию применяется в большинстве дистрибутивов OpenStack. Информация о том, как настроить другие брокеры, приведена на сайте OpenStack в разделе документации.

RabbitMQ написан на языке программирования Erlang – функциональном языке программирования, разработанном компанией Ericsson. Erlang был выбран как язык программирования, поскольку хорошо подходит для создания высоконадежных распределенных вычислительных систем.

Устанавливаем пакеты RabbitMQ, запускаем сервис и настраиваем автоматический старт после перезагрузки:

```
[root@controller ~]# yum -y install rabbitmq-server
[root@controller ~]# systemctl start rabbitmq-server.service
[root@controller ~]# systemctl enable rabbitmq-server.service
```

Аутентификацию в RabbitMQ можно настроить двумя способами: с использованием имени и пароля SASL-аутентификации (Simple Authentication and Security Layer), обеспечиваемой фреймворком Erlang, и при помощи сертификатов и SSL. Мы будем использовать первый вариант, причем для упрощения настройки все сервисы будут работать под одним пользователем RabbitMQ. Важно отметить, что такая конфигурация не подходит для промышленных внедрений.

По умолчанию в RabbitMQ разрешены два метода SASL-аутентификации: PLAIN и AMQPLAIN (нестандартная версия PLAIN). Также по умолчанию создан пользователь guest, которому по умолчанию заблокирован удаленный доступ.

Можно пойти двумя путями. Первый – это использовать существующего пользователя guest. Для этого поменяем его пароль. В качестве пароля для всех сервисов и учетных записей в этой книге будет использоваться openstack. При желании выберите более сложный или более привычный для себя «пароль по умолчанию»:

```
[root@controller ~]# rabbitmqctl change_password guest openstack
Changing password for user "guest" ...
...done.
```

Второе действие, которое необходимо сделать с пользователем guest, – это разрешить ему удаленный доступ. По умолчанию пользователю guest запрещено подключаться иначе, как с loopback-интерфейса. Необходимо создать файл /etc/rabbitmq/rabbitmq.config со следующим содержимым:

```
[{rabbit, [{loopback_users, []}]}].
```

После чего следует перезапустить сервис:

```
[root@controller ~]# systemctl restart rabbitmq-server.service
```

Проверить состояние сервиса можно командой:

```
[root@controller ~]# rabbitmqctl status
Status of node rabbit@controller ...
[{pid,1150},
 {running_applications,[{rabbit,"RabbitMQ","3.6.5"},
                        {mnesia,"MNESIA  CXC 138 12","4.14.3"},
                        {os_mon,"CPO  CXC 138 46","2.4.2"},
                        {rabbit_common,[],"3.6.5"},
                        {xmerl,"XML parser","1.3.14"},
```

```
                          {ranch,"Socket acceptor pool for TCP
                                 protocols.","1.2.1"},
                          {sasl,"SASL  CXC 138 11","3.0.3"},
                          {stdlib,"ERTS  CXC 138 10","3.3"},
                          {kernel,"ERTS  CXC 138 10","5.2"}]},
 {os,{unix,linux}},
 {erlang_version,"Erlang/OTP 19 [erts-8.3.5.3] [source] [64-bit]
[async-threads:64] [hipe] [kernel-poll:true]\n"},
 {memory,[{total,42765952},
          {connection_readers,0},
          {connection_writers,0},
          {connection_channels,0},
          {connection_other,0},
          {queue_procs,2688},
          {queue_slave_procs,0},
          {plugins,0},
          {other_proc,17355776},
          {mnesia,58976},
          {mgmt_db,0},
          {msg_index,39176},
          {other_ets,919760},
          {binary,19016},
          {code,17749737},
          {atom,752561},
          {other_system,5868262}]},
 {alarms,[]},
 {listeners,[{clustering,25672,"::"},{amqp,5672,"::"}]},
 {vm_memory_high_watermark,0.4},
 {vm_memory_limit,1261810483},
 {disk_free_limit,50000000},
 {disk_free,52476706816},
 {file_descriptors,[{total_limit,924},
                    {total_used,2},
                    {sockets_limit,829},
                    {sockets_used,0}]},
 {processes,[{limit,1048576},{used,137}]},
 {run_queue,0},
 {uptime,202},
 {kernel,{net_ticktime,60}}]
```

Второй способ – это добавить специального пользователя, которого будем использовать для настройки сервисов OpenStack:

```
[root@controller ~]# rabbitmqctl add_user openstack openstack
Creating user "openstack" ...
...done.
```

В данном случае мы добавили пользователя openstack с паролем opentsack. Теперь дадим этому пользователю права на настройку, чтение и запись:

```
[root@controller ~]#  rabbitmqctl set_permissions openstack ".*"
".*" ".*"
Setting permissions for user "openstack" in vhost "/" ...
...done.
```

В книге для настройки всех сервисов мы будем использовать именно этого пользователя openstack. Просмотреть список пользователей можно командой

```
[root@controller ~]# rabbitmqctl list_users
Listing users ...
guest   [administrator]
openstack   []
...done.
```

Добавлять пользователей и просматривать информацию о работе брокера можно через удобную графическую консоль, доступную через механизм plug-in. Для активации консоли включите подключаемый модуль `rabbitmq_management` и перезапустите сервис:

```
[root@controller ~]# /usr/lib/rabbitmq/bin/rabbitmq-plugins enable
rabbitmq_management
[root@controller ~]# systemctl restart rabbitmq-server.service
```

Далее заходим браузером на порт 15672. Как выглядит веб-консоль, показано на рис. 2.2. Если вы оставили включенным SELinux, то необходимо порту присвоить тип `amqp_port_t`. Также если вы хотите подключаться к консоли удаленно, добавьте в конфигурацию брандмауэра соответствующее правило для tcp-порта.

Можно отметить, что OpenStack несильно нагружает брокер сообщений и не рассылает широковещательных сообщений. Для каждого сервиса создается одна очередь сообщений плюс одна дополнительная.

Если далее вас интересует более тесное знакомство с RabbitMQ, то на сайте проекта находятся подробные статьи, просто и доходчиво рассказывающие о применении брокера: http://www.rabbitmq.com/getstarted.html. При внедрении RabbitMQ также будут полезными еще две ссылки: руководство по сетевому взаимодействию http://www.rabbitmq.com/n и список основных действий перед вводом RabbitMQ в эксплуатацию http://www.rabbitmq.com/p.

Еще нужно отметить, что в промышленных инсталляциях необходимо обеспечить высокую доступность сервиса, что чаще всего делается средствами самого RabbitMQ.

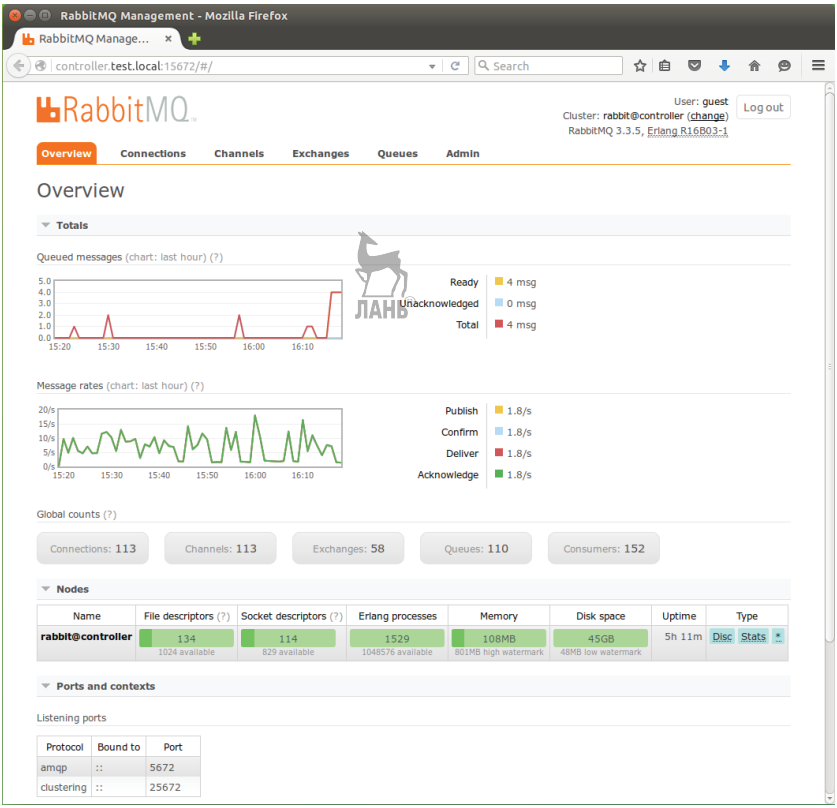


Рис. 2.2. Веб-консоль RabbitMQ

Установка и настройка базы данных

Практически все сервисы OpenStack используют базу данных. Чаще всего с OpenStack описывают работы MariaDB или MySQL, в этой книге мы поступим так же. Устанавливаем пакеты MariaDB и клиентскую библиотеку PyMySQL:

```
[root@controller ~]# yum -y install mariadb-server python2-PyMySQL
```

В OpenStack Liberty и ранее репозитории RDO не содержали пакета PyMySQL, поэтому при настройке использовалась библиотека MySQL-Python. Соответственно, строчка подключения сервисов к базе данных выглядела как «mysql://».

Создаем конфигурационный файл /etc/my.cnf.d/openstack.cnf следующего вида:

```
[mysqld]
bind-address = 192.168.122.200

default-storage-engine = innodb
innodb_file_per_table = on
max_connections = 4096
collation-server = utf8_general_ci
character-set-server = utf8
```

В опции bind-adress мы указываем IP-адрес нашего контроллера. Запускаем и включаем базу данных MariaDB:

```
[root@controller ~]# systemctl enable mariadb.service
[root@controller ~]# systemctl start mariadb.service
```

Запускаем скрипт mysql_secure_installation, который в том числе задает пароль администратора базы данных:

```
[root@controller ~]# mysql_secure_installation
```

Проверить то, что сервис работает, и получить базовую статистику можно командой:

```
[root@controller ~]# mysqladmin -uroot -popenstack status
Uptime: 3318  Threads: 21  Questions: 33979  Slow queries: 0
Opens: 123  Flush tables: 2  Open tables: 149  Queries per second
avg: 10.240
```

Переход на использование утилиты OpenStackClient

Одно из изменений в четвертом издании книги, по сравнению с предыдущими, – использование клиента командной строки openstack вместо отдельных независимых утилит. В марте 2015 года список официальных проектов OpenStack пополнился еще одним проектом – OpenStackClient. Утилита командной строки openstack представляет собой единый унифицированный клиент для доступа к OpenStack API. В целом OpenStackClient позволяет делать то же самое, что и утилиты каждого сервиса (команды keystone, nova, neutron и др.), но при помощи единой команды и унифицированного формата.

Команды строятся по принципу: взять объект1 и произвести над ним действие при помощи объекта2. Команда `openstack help -h` выведет подсказку по командам, или можно ввести `help` в интерактивном режиме. В интерактивном режиме приглашение меняется на (openstack):

```
[root@controller ~]# openstack
(openstack) help
...
Application commands (type help <topic>):
==========================================
aggregate add host      ip fixed remove        server rescue
aggregate create        ip floating add        server resize
aggregate delete        ip floating create     server resume
...
```

Тем не менее традиционные команды (кроме `keystone`) также работают, и при желании вы можете пользоваться ими. Документация на сайте OpenStack начала меняться, переходя на использование команды openstack, начиная с релиза Kilo. Большинство руководств, статей в Интернете, чей возраст ранее середины 2015 года, не использует клиента openstack.

Глава 3

Сервис идентификации Keystone

```
Название: OpenStack Identity (Keystone)
Назначение: сервис аутентификации и каталог
Пакет: openstack-keystone
Имя сервиса: httpd.service (ранее openstack-keystone.service)
Порты: 5000/tcp и 35357/tcp (openstack-id)
Конфигурационные файлы: /etc/keystone/keystone.conf
Файл журнала: /var/log/keystone/keystone.log, /var/log/httpd/keystone*
```

Служба идентификации OpenStack Keystone представляет собой централизованный каталог пользователей и сервисов, к которым они имеют доступ. Keystone выступает в виде единой системы аутентификации и авторизации облачной операционной системы. Сервис поддерживает несколько типов аутентификации, включая аутентификацию по токенам, аутентификацию при помощи пары имя пользователя/пароль и AWS-совместимую аутентификацию. Keystone поддерживает интеграцию с существующими сервисами каталогов, например LDAP. Помимо этого, Keystone является каталогом служб, доступных в OpenStack. Keystone поддерживает справочник реквизитов для обращения к API соответствующих сервисов, а также реализует политики ролевого контроля доступа.

Терминология Keystone

OpenStack – это облачная операционная система, и, как в каждой операционной системе, в OpenStack есть пользователи. После аутентификации пользователь получает токен для доступа к тем или иным ресурсам. Пользователь создается не только для представления в системе пользователя, обращающегося к ресурсам облака, но также и для сервисов.

Сервис – это одна из далее рассматриваемых служб OpenStack, например Nova, Swift или Glance. У каждого сервиса есть одна или более точек входа (endpoint). Точка входа представляет собой URL, по которому доступен этот сервис.

Токен – это строка текста, состоящая из букв и цифр и предназначенная для доступа к API и ресурсам. Токен выдается на ограниченное время и при необходимости может быть отозван до истечения срока действия. Для того чтобы пользователь получил токен, он должен либо предоставить имя и пароль, либо имя и ключ для доступа к API (API key). Токен также содержит список ролей, определяющих доступные пользователю полномочия.

Ресурсы (виртуальные машины, образы и т. д.) объединяются в проекты. В документации на английском языке также ранее использовался термин tenant, но в настоящее время он заменен на project, интуитивно более понятный термин. Проект является контейнером, который может объединять ресурсы отдельной организации, использующей публичное облако OpenStack, отдельного приложения или отдельного пользователя, – это вы решаете сами.

Пользователи и группы пользователей сами по себе не принадлежат проектам. Они получают доступ к ресурсам проектов через назначение ролей.

С появлением третьей версии Keystone API проекты объединяются в домены. Домен – это самый крупный контейнер в терминологии Keystone. Домены определяют пространство имен – область видимости объектов. Например, пользователи должны быть уникальны в рамках одного домена. По умолчанию Keystone создает домен с именем Default.

Установка и настройка Keystone

Теперь после краткого знакомства с теорией перейдем к практике. Установим пакеты сервиса, клиент openstack и дополнительный набор скриптов, упрощающих работу openstack-utils:

```
[root@controller ~]# yum -y install openstack-keystone python-
openstackclient httpd mod_wsgi
```

Перед тем как настраивать сервис, создадим базу данных keystone и дадим необходимые привилегии:

```
[root@controller ~]# mysql -u root -p
MariaDB [(none)]> CREATE DATABASE keystone;
```

```
MariaDB [(none)]> GRANT ALL PRIVILEGES ON keystone.* TO
'keystone'@'localhost' IDENTIFIED BY 'openstack';
MariaDB [(none)]> GRANT ALL PRIVILEGES ON keystone.* TO
'keystone'@'%' IDENTIFIED BY 'openstack';
MariaDB [(none)]> exit
```

Пропишем в конфигурационном файле Keystone строку подключения к базе данных:

```
[root@controller ~]# crudini --set /etc/keystone/keystone.conf
database connection mysql+pymysql://keystone:openstack@controller.
test.local/keystone
```

Далее мы должны задать формат и провайдер токенов, генерируемых сервисом Keystone. OpenStack поддерживает четыре формата токенов: UUID, PKI, PKIz и Fernet.

Первыми появились UUID-токены. Они представляют из себя строку из 32 символов, которую удобно использовать в вызовах OpenStack API, например применяя команду curl. Преимуществом этого формата токенов является их небольшой размер, а недостатком – то, что токен не содержит информации, достаточной для того, чтобы произвести локальную авторизацию. Поэтому сервисы OpenStack каждый раз должны отправлять токен сервису Keystone, для того чтобы получить информацию, какие операции разрешены с этим токеном. Таким образом, постоянные обращения к Keystone становились «бутылочным горлышком» системы в целом.

Решением этой проблемы должны были стать PKI-токены. Они содержат всю необходимую для локальной авторизации информацию и, кроме того, содержат в себе цифровую подпись и информацию об устаревании. Таким образом, другие сервисы OpenStack могут локально кэшировать эти токены. Такая архитектура значительно сократила трафик к Keystone, но увеличила размеры токенов. Их размер мог превышать 8 Кбайт, что создавало проблемы при работе с некоторыми веб-серверами, поскольку не все из них по умолчанию поддерживают HTTP-заголовки такого размера, в которых передается токен. Также большой размер токена делал неудобным работу с curl. Попытка создать сжатый вариант под названием PKIz тоже не решила проблему.

PKIZ-токены в релизе Mitaka были объявлены устаревшими и добавлен третий тип – Fernet-токены (https://github.com/fernet/spec). Они небольшого размера (до 255 символов), но содержат достаточно информации для локальной авторизации. Их не требу-

ется синхронизировать между регионами, для них не нужна база данных (токены без сохранения состояния), и процесс генерации их быстрее, чем в первых двух реализациях. Дополнительным плюсом будет отсутствие необходимости настройки memcached. Именно Fernet-токены мы и будем использовать. Зададим соответствующий провайдер:

```
[root@controller ~]# crudini --set /etc/keystone/keystone.conf token
provider fernet
```

Инициализируем базу данных и репозитории ключей Fernet:

```
[root@controller ~]# su -s /bin/sh -c "keystone-manage db_sync"
keystone
[root@controller ~]# keystone-manage fernet_setup --keystone-user
keystone --keystone-group keystone
[root@controller ~]# keystone-manage credential_setup --keystone-
user keystone --keystone-group keystone
```

Важным будет отметить, что по релиз OpenStack Liberty включительно сервис идентификации для сетевого взаимодействия использовал встроенный сервис Python Eventlet, допуская использование Apache с mod_wsgi, когда нужны лучшая масштабируемость и безопасность. Начиная с OpenStack Mitaka (апрель 2016 года) поддерживается только Apache. В конфигурационном файле вебсервера /etc/httpd/conf/httpd.conf указываем имя сервера:

```
[root@controller ~]# echo 'ServerName controller.test.local' >>
/etc/httpd/conf/httpd.conf
```

И создаем файл /etc/httpd/conf.d/wsgi-keystone.conf следующего содержания:

```
Listen 5000
Listen 35357

<VirtualHost *:5000>
    WSGIDaemonProcess keystone-public processes=5 threads=1
user=keystone group=keystone display-name=%{GROUP}
    WSGIProcessGroup keystone-public
    WSGIScriptAlias / /usr/bin/keystone-wsgi-public
    WSGIApplicationGroup %{GLOBAL}
    WSGIPassAuthorization On
    LimitRequestBody 114688
    <IfVersion >= 2.4>
      ErrorLogFormat "%{cu}t %M"
    </IfVersion>
```

```
    ErrorLog /var/log/httpd/keystone.log
    CustomLog /var/log/httpd/keystone_access.log combined

    <Directory /usr/bin>
        <IfVersion >= 2.4>
            Require all granted
        </IfVersion>
        <IfVersion < 2.4>
            Order allow,deny
            Allow from all
        </IfVersion>
    </Directory>
</VirtualHost>

<VirtualHost *:35357>
    WSGIDaemonProcess keystone-admin processes=5 threads=1
user=keystone group=keystone display-name=%{GROUP}
    WSGIProcessGroup keystone-admin
    WSGIScriptAlias / /usr/bin/keystone-wsgi-admin
    WSGIApplicationGroup %{GLOBAL}
    WSGIPassAuthorization On
    LimitRequestBody 114688
    <IfVersion >= 2.4>
      ErrorLogFormat "%{cu}t %M"
    </IfVersion>
    ErrorLog /var/log/httpd/keystone.log
    CustomLog /var/log/httpd/keystone_access.log combined

    <Directory /usr/bin>
        <IfVersion >= 2.4>
            Require all granted
        </IfVersion>
        <IfVersion < 2.4>
            Order allow,deny
            Allow from all
        </IfVersion>
    </Directory>
</VirtualHost>

Alias /identity /usr/bin/keystone-wsgi-public
<Location /identity>
    SetHandler wsgi-script
    Options +ExecCGI

    WSGIProcessGroup keystone-public
    WSGIApplicationGroup %{GLOBAL}
    WSGIPassAuthorization On
</Location>

Alias /identity_admin /usr/bin/keystone-wsgi-admin
```

```
<Location /identity_admin>
    SetHandler wsgi-script
    Options +ExecCGI

    WSGIProcessGroup keystone-admin
    WSGIApplicationGroup %{GLOBAL}
    WSGIPassAuthorization On
</Location>
```

Можно не набирать содержимое конфигурационного файла вручную, а воспользоваться примером, поставляемым в пакете openstack-keystone:

```
[root@controller ~]# ln -s /usr/share/keystone/wsgi-keystone.conf
/etc/httpd/conf.d/
```

Запускаем веб-сервер, который будет прослушивать порты 5000 и 35357:

```
[root@controller ~]# systemctl enable httpd.service
[root@controller ~]# systemctl start httpd.service
```

По команде `systemctl status` вы должны будете видеть десять процессов keystone, указанных в директивах WSGIDaemonProcess:

```
[root@controller ~]# systemctl status httpd.service
  httpd.service - The Apache HTTP Server
   Loaded: loaded (/usr/lib/systemd/system/httpd.service; enabled;
vendor preset: disabled)
   Active: active (running) since Fri 2018-03-02 14:26:15 CET; 8s ago
     Docs: man:httpd(8)
           man:apachectl(8)
 Main PID: 14629 (httpd)
   Status: "Processing requests..."
   CGroup: /system.slice/httpd.service
           ├─14629 /usr/sbin/httpd -DFOREGROUND
           ├─14630 (wsgi:keystone- -DFOREGROUND
           ├─14631 (wsgi:keystone- -DFOREGROUND
           ├─14632 (wsgi:keystone- -DFOREGROUND
           ├─14633 (wsgi:keystone- -DFOREGROUND
           ├─14634 (wsgi:keystone- -DFOREGROUND
           ├─14635 (wsgi:keystone- -DFOREGROUND
           ├─14636 (wsgi:keystone- -DFOREGROUND
           ├─14637 (wsgi:keystone- -DFOREGROUND
           ├─14638 (wsgi:keystone- -DFOREGROUND
           ├─14663 (wsgi:keystone- -DFOREGROUND
           ├─14664 /usr/sbin/httpd -DFOREGROUND
           ├─14665 /usr/sbin/httpd -DFOREGROUND
           ├─14666 /usr/sbin/httpd -DFOREGROUND
```

```
           ├─14667 /usr/sbin/httpd -DFOREGROUND
           └─14668 /usr/sbin/httpd -DFOREGROUND

Mar 02 14:26:15 controller.test.local systemd[1]: Starting The Apache
HTTP Server...
Mar 02 14:26:15 controller.test.local systemd[1]: Started The Apache
HTTP Server.
```

Для инициализации Keystone можно пойти двумя путями: использовать рекомендованную разработчиками команду `keystone-manage bootstrap`, которая выполнит инициализацию за нас, или пойти более длинным путем, в целях обучения воспользовавшись авторизационным токеном. При использовании первого варианта синтаксис команды будет следующим:

```
[root@controller ~]# keystone-manage bootstrap --bootstrap-password
openstack \
  --bootstrap-admin-url http://controller:35357/v3/ \
  --bootstrap-internal-url http://controller:35357/v3/ \
  --bootstrap-public-url http://controller:5000/v3/ \
  --bootstrap-region-id RegionOne
```

В этом случае вы можете пропустить все команды до конца раздела.

При выборе второго пути необходимо определить авторизационный токен, который необходим для первоначальной настройки. Он будет являться общим секретом между Keystone и другими сервисами, а также его можно использовать, если административный пользователь не был задан или вы забыли его пароль.

Для генерации случайного содержимого токена мы воспользуемся OpenSSL, а для работы с конфигурационным файлом сервиса /etc/keystone/keystone.conf – утилитой `crudini`:

```
[root@controller ~]# export ADM_TOKEN=$(openssl rand -hex 10)
[root@controller ~]# crudini --set /etc/keystone/keystone.conf
DEFAULT admin_token $ADM_TOKEN
```

Обычно доступ к Keystone осуществляется при помощи имени пользователя и пароля. Однако у нас пока пользователи не созданы, и мы должны воспользоваться авторизационным токеном, созданным ранее. Токен может быть передан через опции команды `keystone` или при помощи переменных окружения. Для удобства создадим скрипт `keystonerc_adm`, который можно будет использовать не только для начальной настройки, но и в дальнейшем для отладки или работы с Keystone:

```
unset OS_USERNAME OS_TENANT_NAME OS_PASSWORD OS_AUTH_URL OS_TOKEN OS_URL
# Указываем содержимое $ADM_TOKEN
export OS_TOKEN=79fec7c50bc3ae87db49
# Указываем точку входа сервиса Keystone
export OS_URL=http://controller.test.local:35357/v3
# Указываем версию API
export OS_IDENTITY_API_VERSION=3
```

Теперь можно отдать команду:

```
$ source keystonerc_adm
```

Как упоминалось ранее, одна из функций Keystone – быть каталогом доступных сервисов OpenStack. Создадим запись о самом сервисе Keystone:

```
$ openstack service create --name keystone --description
"OpenStack Identity" identity
+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description | OpenStack Identity               |
| enabled     | True                             |
| id          | 9700a30f8eb74b3b8c96e81b0e89321a |
| name        | keystone                         |
| type        | identity                         |
+-------------+----------------------------------+
```

Теперь создадим точки входа для сервиса:

```
$ openstack endpoint create identity public http://controller.
test.local:5000/v3 --region RegionOne
+--------------+----------------------------------+
| Field        | Value                            |
+--------------+----------------------------------+
| enabled      | True                             |
| id           | 1433078144ea46b187bb696ff936adfe |
| interface    | public                           |
| region       | RegionOne                        |
| region_id    | RegionOne                        |
| service_id   | 9700a30f8eb74b3b8c96e81b0e89321a |
| service_name | keystone                         |
| service_type | identity                         |
| url          | http://controller.test.local:5000/v3 |
+--------------+----------------------------------+
$ openstack endpoint create identity internal http://controller.
test.local:5000/v3 --region RegionOne
+--------------+----------------------------------+
| Field        | Value                            |
```

```
+--------------+----------------------------------------+
| enabled      | True                                   |
| id           | b81cc95780254417a8df42982ed279fa       |
| interface    | internal                               |
| region       | RegionOne                              |
| region_id    | RegionOne                              |
| service_id   | 9700a30f8eb74b3b8c96e81b0e89321a       |
| service_name | keystone                               |
| service_type | identity                               |
| url          | http://controller.test.local:5000/v3 |
+--------------+----------------------------------------+
$ openstack endpoint create identity admin http://controller.
test.local:35357/v3 --region RegionOne
+--------------+----------------------------------------+
| Field        | Value                                  |
+--------------+----------------------------------------+
| enabled      | True                                   |
| id           | 0b731c9d27354f34b874b92ae2439b86       |
| interface    | admin                                  |
| region       | RegionOne                              |
| region_id    | RegionOne                              |
| service_id   | 9700a30f8eb74b3b8c96e81b0e89321a       |
| service_name | keystone                               |
| service_type | identity                               |
| url          | http://controller.test.local:35357/v3 |
+--------------+----------------------------------------+
```

OpenStack поддерживает три варианта точек входа для каждого сервиса: admin, internal и public. В промышленных инсталляциях эти точки входа должны располагаться в различных сетях и обслуживать различные типы пользователей.

Обратите внимание, что ID сервиса присваивается динамически и при выполнении команд вывод наверняка будет различаться.

Также обратите внимание, что в команде мы использовали опцию `--region RegionOne`. Мы можем создавать несколько регионов в нашей облачной среде. В качестве отдельного региона можно использовать, например, отдельный центр обработки данных. При этом точки входа сервисов у каждого региона могут быть свои.

Работа с пользователями, ролями и проектами в Keystone

Познакомимся на практике, как создавать проекты, пользователей и роли. Сделаем это на примере администратора, который

нам будет необходим для дальнейшего управления ресурсами OpenStack. Обратите внимание, что если вы воспользовались командой `keystone-manage bootstrap`, то домен Default, пользователь и роль admin уже созданы за вас.

Первым делом создадим домен (пространство имен) Default и в нем проект admin. Напомним, что проект – это контейнер, объединяющий ресурсы облака:

```
$ source keystonerc_adm
$ openstack domain create --description "Default Domain" default
+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description | Default Domain                   |
| enabled     | True                             |
| id          | 0d4b1f2685b048639fb537f87c1e472f |
| name        | default                          |
+-------------+----------------------------------+
$ openstack project create --domain default --description "Admin
Project" admin
+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description | Admin Project                    |
| domain_id   | 0d4b1f2685b048639fb537f87c1e472f |
| enabled     | True                             |
| id          | 4372600eb0b249edaeba5eda5f4dcf81 |
| is_domain   | False                            |
| name        | admin                            |
| parent_id   | 0d4b1f2685b048639fb537f87c1e472f |
+-------------+----------------------------------+
```

Далее создадим самого пользователя admin:

```
$ openstack user create --domain default --email andrey@controller.
test.local --password openstack admin
+---------------------+----------------------------------+
| Field               | Value                            |
+---------------------+----------------------------------+
| domain_id           | 0d4b1f2685b048639fb537f87c1e472f |
| email               | andrey@controller.test.local     |
| enabled             | True                             |
| id                  | 65ec03579454407b88727eb369f3dd67 |
| name                | admin                            |
| password_expires_at | None                             |
+---------------------+----------------------------------+
```

И наконец, создадим роль admin:

```
$ openstack role create admin
+-----------+----------------------------------+
| Field     | Value                            |
+-----------+----------------------------------+
| domain_id | None                             |
| id        | c2ab46dd35ed44dd86749c8ffd668355 |
| name      | admin                            |
+-----------+----------------------------------+
```

Роль admin присутствует в OpenStack «из коробки». Роли описываются в файле /etc/keystone/policy.json. В частности, описание роли admin:

```
{
    "admin_required": "role:admin or is_admin:1"
}
```

Далее идет описание правил политики в формате: «служба:действие: условия». Например:

```
{
  "identity:list_roles": "rule:admin_required",
}
```

Вы можете создать и другие роли, но они не будут применяться, пока не описаны в файле policy.json. Файлы policy.json с описанием ролей и политик доступа индивидуальны для каждого сервиса и расположены в соответствующих сервису поддиректориях директории /etc.

Просмотреть список ролей и их идентификаторов можно командой:

```
$ openstack role list
+----------------------------------+----------+
|                id                |   name   |
+----------------------------------+----------+
| 9fe2ff9ee4384b1894a90878d3e92bab | _member_ |
| f45f68589f8445699f361ba13c37623d |  admin   |
+----------------------------------+----------+
```

Создание новых ролей не является распространенной практикой. Теперь собираем все вместе. Добавляем роль admin проекту и пользователю:

```
$ openstack role add --project admin --user admin admin
```

Важно, что роль admin – глобальная. Если вы даете ее пользователю в одном проекте, то вы даете ее пользователю во всем облаке.

Начиная с этого момента, мы можем продолжить работу пользователем admin. Создадим скрипт `keystonerc_admin` следующего содержания:

```
export OS_AUTH_TYPE=password
export OS_PROJECT_DOMAIN_NAME=Default
export OS_USER_DOMAIN_NAME=Default
export OS_PROJECT_NAME=admin
export OS_USERNAME=admin
export OS_PASSWORD=openstack
export OS_AUTH_URL=http://controller.test.local:35357/v3
export OS_IDENTITY_API_VERSION=3
```

и выполним команду

```
$ source keystonerc_admin
```

Поскольку вся необходимая для аутентификации информация присутствует в переменных окружения, мы можем запросить токен:

```
# openstack token issue
+------------+-----------------------------------------------------+
| Field      | Value                                               |
+------------+-----------------------------------------------------+
| expires    | 2018-03-02T14:59:14+0000                            |
| id         | gAAAAABamViyvyadVUQSm6BL1lLIo2n8ZkVu_RaWBkPReK6kQa1g |
|            | DtEhPPmwdXX1Pu8fjA3fXbY5fOsHxndJY-Vew6j6eroT2vQZbFuF |
|            | r0RGf7fNj1ggvmnJ_bb0c_qnDxh9tc9LpyFL8LAv4ceHoUZI2Y7c |
|            | bdjuPM0wZJnUV3JsWVCuwIvdWe0                         |
| project_id | 7fe2a6ef08df4a749f3bad1fceb055b9                    |
| user_id    | a37a67fdf3dc4e9c9e5251754b26770d                    |
+------------+-----------------------------------------------------+
```

Теперь перейдем к непривилегированному пользователю и проекту. Можно повторить команды, аналогичные ранее описанным для проекта admin, но мы воспользуемся утилитой keystone, которая использовалась до появления клиента OpenStackClient. Читателю будет полезно знакомство с утилитой keystone, если придется работать с устаревшими версиями OpenStack. Важно отметить, что на нашем стенде с релизом Queens и API v3 команда `keystone` работать не будет. Пример приведен только для устаревших релизов:

```
$ keystone tenant-create --name demo --description "Demo Tenant"
+-------------+----------------------------------+
|   Property  |              Value               |
+-------------+----------------------------------+
```

```
| description |            Demo Tenant            |
|   enabled   |               True                |
|      id     | ebf7fa585d254c749deff6385c3e816f |
|     name    |               demo                |
+-------------+-----------------------------------+

$ keystone user-create --name demo --tenant demo --pass openstack
--email user@controller.test.local
+----------+-----------------------------------+
| Property |              Value                |
+----------+-----------------------------------+
|  email   |    user@controller.test.local     |
| enabled  |               True                |
|    id    | 1fb056cf72d248b4a16f2227d2ff74ff |
|   name   |               demo                |
| tenantId | ebf7fa585d254c749deff6385c3e816f |
| username |               demo                |
+----------+-----------------------------------+
```

На нашем стенде две вышеприведенные команды необходимо заменить:

```
$ openstack project create --domain default --description "Demo
Tenant" demo
+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description | Demo Tenant                      |
| domain_id   | default                          |
| enabled     | True                             |
| id          | bc10ac4b71164550a363b8098e8ad270 |
| is_domain   | False                            |
| name        | demo                             |
| parent_id   | default                          |
| tags        | []                               |
+-------------+----------------------------------+
$ openstack user create --domain default --project demo --email
user@controller.test.local --password openstack demo
+---------------------+----------------------------------+
| Field               | Value                            |
+---------------------+----------------------------------+
| default_project_id  | bc10ac4b71164550a363b8098e8ad270 |
| domain_id           | default                          |
| email               | user@controller.test.local       |
| enabled             | True                             |
| id                  | 3b76dece42b140e092dc1a76a85c1879 |
| name                | demo                             |
| options             | {}                               |
| password_expires_at | None                             |
+---------------------+----------------------------------+
```

Создадим и добавим пользователю demo роль члена проекта:

```
$ openstack role create user
+-----------+----------------------------------+
| Field     | Value                            |
+-----------+----------------------------------+
| domain_id | None                             |
| id        | 9fe2ff9ee4384b1894a90878d3e92bab |
| name      | user                             |
+-----------+----------------------------------+
$ openstack role add --project demo --user demo user
```

Команда `keystone help` или `openstack help` покажет список ключей с «говорящими» именами, позволяющими просматривать, удалять и создавать роли, сервисы, проекты и пользователей. Кроме того, по аналогии с `keystonerc_admin` вы можете создать скрипт для пользователя demo. Единственное отличие, помимо имени пользователя и проекта, будет в переменной OS_AUTH_URL. Вместо порта 35357 используйте порт 5000, предназначенный для обычных пользователей:

```
export OS_AUTH_TYPE=password
export OS_PROJECT_DOMAIN_NAME=Default
export OS_USER_DOMAIN_NAME=Default
export OS_PROJECT_NAME=demo
export OS_USERNAME=demo
export OS_PASSWORD=openstack
export OS_AUTH_URL=http://controller.test.local:5000/v3
export OS_IDENTITY_API_VERSION=3
```

Также автор рекомендовал бы добавить в каждый из файлов keystonerc_* новое определение переменной окружения PS1, которая отвечает в bash за внешний вид приглашения. Было бы полезно, для того чтобы не запутаться, каким пользователем OpenStack вы работаете, поместить в PS1 его имя или название. Например:

```
export PS1='[\u@\h \W(Openstack_Admin)]\$ '
```

и

```
export PS1='[\u@\h \W(Openstack_Demo)]\$ '
```

После команды `source` для такого keystonerc приглашение для пользователя поменяется на:

```
[root@controller ~(Openstack_Admin)]#
```

или

```
[root@controller ~(Openstack_Demo)]#
```

Последнее, что мы сделаем, – это подготовим специальный проект для остальных сервисов OpenStack. Каждый сервис требует пользователя и роль администратора в специальном сервисном контейнере:

```
$ openstack project create --domain default --description "Service
Project" service
+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description | Service Project                  |
| domain_id   | 0d4b1f2685b048639fb537f87c1e472f |
| enabled     | True                             |
| id          | 0e8406620808466590e592df819c88ee |
| is_domain   | False                            |
| name        | service                          |
| parent_id   | 0d4b1f2685b048639fb537f87c1e472f |
+-------------+----------------------------------+
```

Соответствующий синтаксис команды `keystone` выглядит как:

```
$ keystone tenant-create --name service --description "Service Project"
```

Просмотреть список пользователей и проектов можно командами `openstack user list` и `openstack project list`. Рекомендуется отключить временный механизм аутентификации по токенам, в случае если вы использовали его для создания первого пользователя admin. Для этого необходимо отредактировать файл /usr/share/keystone/keystone-dist-paste.ini и удалить `admin_token_auth` из секций `[pipeline:public_api]`, `[pipeline:admin_api]` и `[pipeline:api_v3]`.

Итак, если вы, следуя за изложением материала, отработали все приведенные команды, то у вас установлен и настроен сервис идентификации Keystone, созданы три проекта – администратора, сервисный и demo – и заведены два пользователя – demo и admin, а также вы познакомились с устаревшей командой `keystone`. Еще раз подчеркну, что если вы работаете с последними релизами OpenStack, команда `keystone` вам не понадобится.

Посмотреть, какие сервисы доступны в вашем облаке, и точки входа в эти сервисы всегда можно командой `openstack catalog list`. Пока у нас будет только один сервис:

```
$ openstack catalog list
+----------+----------+-----------------------------------------+
| Name     | Type     | Endpoints                               |
+----------+----------+-----------------------------------------+
| keystone | identity | RegionOne                               |
|          |          |   internal: http://controller:35357/v3/ |
|          |          | RegionOne                               |
|          |          |   public: http://controller:5000/v3/    |
|          |          | RegionOne                               |
|          |          |   admin: http://controller:35357/v3/    |
|          |          |                                         |
+----------+----------+-----------------------------------------+
```

Также будет полезным знать команду `openstack service list`. Она покажет сервисы и их идентификаторы. Пример с вывода команды приведен ниже. Такой вывод вы получите ближе к концу книги:

```
$ openstack service list
+----------------------------------+------------+-----------------+
| ID                               | Name       | Type            |
+----------------------------------+------------+-----------------+
| 009af6dfa02d4e238a504e6f63c940a4 | heat       | orchestration   |
| 27c82194ea0844ed8fbd59e9798e3337 | gnocchi    | metric          |
| 3345af31a7684e259ff49a17b05b9ab7 | nova       | compute         |
| 5eff07227f47445993898a3ae18ccb94 | swift      | object-store    |
| 63e493e9b9254be59056cec2ca44ddbe | keystone   | identity        |
| 674b422d44fa435c9c05f5bc79293d67 | aodh       | alarming        |
| 717ed5d914a54a3ba8b576ea2379ffe6 | cinderv3   | volumev3        |
| 7a08a70288084664b770ec0d38af85e1 | glance     | image           |
| a8ee318c098745d1ae81cbb0a3f36f2b | placement  | placement       |
| b186a1daee4c4147abe4d454b7431298 | cinderv2   | volumev2        |
| b8068f12e01c4cb4af30dba04339bb6c | magnum     | container-infra |
| ebb95159786f48668c5fa50a10129a89 | neutron    | network         |
+----------------------------------+------------+-----------------+
```

Еще одна вещь, на которую хотелось бы обратить внимание читателя, прежде чем мы пойдем дальше. В этой главе мы работали с утилитой командной строки – клиентом keystone и openstack. Далее мы столкнемся с утилитами `glance`, `cinder`, `nova` и др. У них имеется полезный ключ `--debug`, который позволяет посмотреть, какие вызовы OpenStack API использовались при том или ином действии. Например:

```
$ openstack --debug project create --domain default --description
"My project" proj1
START with options: [u'--debug', u'project', u'create', u'--domain',
u'default', u'--description', u'My project', u'proj1']
options: Namespace(access_key='', access_secret='***', access_
```

```
token='***', access_token_endpoint='', access_token_type='',
application_credential_id='', application_credential_name='',
application_credential_secret='***', auth_type='', auth_
url='http://controller.test.local:35357/v3', cacert=None,
cert='', client_id='', client_secret='***', cloud='', code='',
consumer_key='', consumer_secret='***', debug=True, default_
domain='default', default_domain_id='', default_domain_name='',
deferred_help=False, discovery_endpoint='', domain_id='',
domain_name='', endpoint='', identity_provider='', insecure=None,
interface='', key='', log_file=None, openid_scope='', os_beta_
command=False, os_compute_api_version='', os_identity_api_
version='3', os_image_api_version='2', os_network_api_version='',
os_object_api_version='', os_project_id=None, os_project_
name=None, os_volume_api_version='', passcode='', password='***',
profile='', project_domain_id='', project_domain_name='Default',
project_id='', project_name='admin', protocol='', redirect_
uri='', region_name='', remote_project_domain_id='', remote_
project_domain_name='', remote_project_id='', remote_project_
name='', service_provider='', system_scope='', timing=False,
token='***', trust_id='', url='', user_domain_id='', user_domain_
name='Default', user_id='', username='admin', verbose_level=3,
verify=None)
...
GET call to identity for http://controller:35357/v3/domains/
default used request id req-4c0ac864-29e1-4d54-b6a7-b210cb6e7094
REQ: curl -g -i -X POST http://controller:35357/v3/projects
-H "User-Agent: python-keystoneclient" -H "Content-Type:
application/json" -H "Accept: application/json" -H "X-Auth-Token:
{SHA1}fd96ae3da4410fe6dc21791332c3b5ab734a78f2" -d '{"project":
{"enabled": true, "description": "My project", "name": "proj1",
"domain_id": "default"}}'
http://controller:35357 "POST /v3/projects HTTP/1.1" 201 289
RESP: [201] Date: Fri, 02 Mar 2018 14:34:06 GMT Server:
Apache/2.4.6 (CentOS) mod_wsgi/3.4 Python/2.7.5 Vary: X-Auth-
Token x-openstack-request-id: req-37c2237c-e157-483f-9f91-
7d0b195d1d46 Content-Length: 289 Keep-Alive: timeout=5, max=99
Connection: Keep-Alive Content-Type: application/json
RESP BODY: {"project": {"is_domain": false, "description": "My
project", "links": {"self": "http://controller:35357/v3/projects/
26aa317a74aa43a5bf40513a6fbbea28"}, "tags": [], "enabled": true,
"id": "26aa317a74aa43a5bf40513a6fbbea28", "parent_id": "default",
"domain_id": "default", "name": "proj1"}}

POST call to identity for http://controller:35357/v3/projects
used request id req-37c2237c-e157-483f-9f91-7d0b195d1d46
+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description | My project                       |
| domain_id   | default                          |
```

```
| enabled     | True                             |
| id          | 26aa317a74aa43a5bf40513a6fbbea28 |
| is_domain   | False                            |
| name        | proj1                            |
| parent_id   | default                          |
| tags        | []                               |
+-------------+----------------------------------+
clean_up CreateProject:
END return value: 0
```

Далее по адресу https://developer.openstack.org/api-guide/quick-start/index.html можно ознакомиться со справочниками API. В данном случае нас интересует раздел «Identity API v3» и вызов «Create project».

Глава 4

Сервис хранения образов Glance

```
Название: OpenStack Image Service (Glance)
Назначение: каталог образов виртуальных машин
Пакет:  openstack-glance
Имена сервисов: openstack-glance-registry, openstack-glance-api
Порт: 9292/tcp
Конфигурационные файлы: /etc/glance/glance-registry.conf, /etc/
glance/glance-api.conf
Файлы журнала: /var/log/glance/api.log, /var/log/glance/registry.log
```

Сервис Glance отвечает за ведение каталога, регистрацию и доставку образов виртуальных машин. Как правило, эти образы выполняют роль шаблонов и требуют дополнительной настройки после запуска виртуальной машины. Glance можно назвать реализацией проекта «образы виртуальных машин как сервис». При этом Glance не реализует фактического хранения образов, а через один из адаптеров использует в качестве бэкэнда ту или иную систему хранения данных. Это может быть как локальная файловая система (используется по умолчанию и описывается в этой главе), NFS, GlusterFS, так и рассматриваемые в дальнейших главах объектное хранилище Swift или блочное Ceph.

Метаданные образов, такие как размер, формат, имя и т. д., хранятся в базе данных.

Glance поддерживает целый ряд форматов хранения образов виртуальных машин: vhd, vmdk, vdi, iso, qcow2, ami и др. В качестве образов могут также выступать ядро и initrd-файл, которые при запуске виртуальной машины необходимо связывать вместе. Также необходимо отметить, что хотя утилиты OpenStack допускают указание формата контейнера (bare, aki, ovf…), в настоящее вре-

мя поддержка контейнеров для образов виртуальных машин в OpenStack отсутствует.

Сервис включает в себя две службы:

- **glance-api** – предоставляет доступ к REST API сервиса образов для поиска, хранения и получения образов;
- **glance-registry** – хранит, обрабатывает и предоставляет информацию. Непосредственно пользователи не взаимодействуют с этим сервисом, поэтому в реальной промышленной среде доступ к нему должен быть ограничен только сервисами Glance.

Взаимодействие с другими сервисами и архитектура Glance приведены на рис. 4.1.

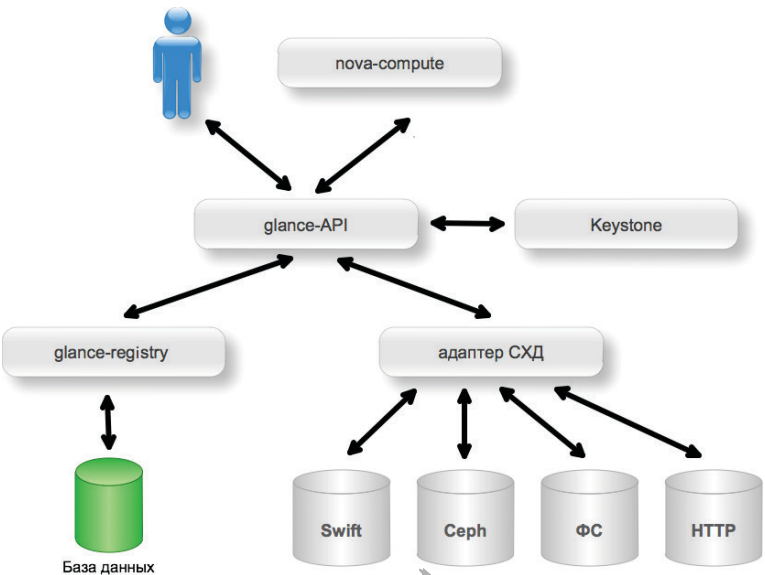


Рис. 4.1. Архитектура Glance

Процедура получения образа гипервизором при запуске виртуальной машины зависит от бэкэнда, но обычно выглядит следующим образом. Nova отправляет GET-запрос по адресу http://путь-к-сервису-glance/images/идентификатор-образа. В случае если образ найден, то glance-api возвращает URL, ссылающийся на образ. Nova передает ссылку драйверу гипервизора, который напрямую скачивает образ.

Установка и настройка сервиса Glance

Для своей работы Glance требует работающий сервис Keystone и базу данных для хранения метаданных об образах. Мы в качестве базы данных будем использовать уже установленную MariaDB.

Для начала выполним команды, знакомые нам из предыдущей главы. Создадим пользователя glance:

```
$ source keystonerc_admin
$ openstack user create --domain default --password openstack glance
+---------------------+----------------------------------+
| Field               | Value                            |
+---------------------+----------------------------------+
| domain_id           | default                          |
| enabled             | True                             |
| id                  | b8de68858b00440bb98b6b1541466794 |
| name                | glance                           |
| options             | {}                               |
| password_expires_at | None                             |
+---------------------+----------------------------------+
```

Затем присвоим роль admin пользователю glance в проекте service и создадим сам сервис glance:

```
$ openstack role add --project service --user glance admin
$ openstack service create --name glance --description "OpenStack
Image service" image
+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description | OpenStack Image service          |
| enabled     | True                             |
| id          | 7a08a70288084664b770ec0d38af85e1 |
| name        | glance                           |
| type        | image                            |
+-------------+----------------------------------+
```

Наконец, создадим для сервиса точки входа по аналогии с ранее созданной точкой для Keystone. Обратите внимание, что в этом случае все три URL совпадают:

```
$ openstack endpoint create --region RegionOne image public
http://controller.test.local:9292
+--------------+----------------------------------+
| Field        | Value                            |
+--------------+----------------------------------+
| enabled      | True                             |
| id           | bc261fcf9c54499f9b8dca58ccd0f20e |
```

```
| interface    | internal                          |
| region       | RegionOne                         |
| region_id    | RegionOne                         |
| service_id   | 7a08a70288084664b770ec0d38af85e1  |
| service_name | glance                            |
| service_type | image                             |
| url          | http://controller.test.local:9292 |
+--------------+-----------------------------------+

$ openstack endpoint create --region RegionOne image internal
http://controller.test.local:9292
+--------------+-----------------------------------+
| Field        | Value                             |
+--------------+-----------------------------------+
| enabled      | True                              |
| id           | c1e61498fd1a49c1979df288a3a2cfdf  |
| interface    | internal                          |
| region       | RegionOne                         |
| region_id    | RegionOne                         |
| service_id   | 4569bd1eccb348e787700603cdd94b10  |
| service_name | glance                            |
| service_type | image                             |
| url          | http://controller.test.local:9292 |
+--------------+-----------------------------------+

$ openstack endpoint create --region RegionOne image admin
http://controller.test.local:9292
+--------------+-----------------------------------+
| Field        | Value                             |
+--------------+-----------------------------------+
| enabled      | True                              |
| id           | b465f9d8795c404884eeeefbb328633f  |
| interface    | admin                             |
| region       | RegionOne                         |
| region_id    | RegionOne                         |
| service_id   | 7a08a70288084664b770ec0d38af85e1  |
| service_name | glance                            |
| service_type | image                             |
| url          | http://controller.test.local:9292 |
+--------------+-----------------------------------+
```

На данный момент у нас должны быть точки входа на два сервиса:

```
$ openstack endpoint list
+------+----------+-------------+-------------+---------+----------+-----------------------------------+
| ID   | Region   | Service Name | Service Type | Enabled | Interface | URL                               |
+------+----------+-------------+-------------+---------+----------+-----------------------------------+
| 0b.. | RegionOne | keystone    | identity    | True    | admin     | http://controller.test.local:35357/v3 |
| 14.. | RegionOne | keystone    | identity    | True    | public    | http://controller.test.local:5000/v3  |
| b8.. | RegionOne | keystone    | identity    | True    | internal  | http://controller.test.local:5000/v3  |
```

```
| bf.. | RegionOne | glance      | image       | True    | public    | http://controller.test.local:9292    |
| c1.. | RegionOne | glance      | image       | True    | internal  | http://controller.test.local:9292    |
| f0.. | RegionOne | glance      | image       | True    | admin     | http://controller.test.local:9292    |
+------+-----------+-------------+-------------+---------+-----------+--------------------------------------+
```

При использовании устаревшей команды `keystone` для создания точки входа необходимо указывать идентификатор сервиса, например взятый из вывода команды создания сервиса. Пример команды:

```
$ keystone endpoint-create \
>   --service-id 6a35c051d3f44cdfba24e0598e9b0152 \
>   --publicurl http:// controller.test.local:9292 \
>   --internalurl http:// controller.test.local:9292 \
>   --adminurl http:// controller.test.local:9292
+-------------+-----------------------------------------+
|   Property  |                Value                    |
+-------------+-----------------------------------------+
|   adminurl  |    http:// controller.test.local:9292 |
|      id     |   d4bc5b058eb74142ad4df0b3b98b6b1a    |
| internalurl |    http:// controller.test.local:9292 |
|  publicurl  |    http:// controller.test.local:9292 |
|    region   |             regionOne                   |
|  service_id |   6a35c051d3f44cdfba24e0598e9b0152    |
+-------------+-----------------------------------------+
```

Теперь можно установить необходимые пакеты:

```
[root@controller ~]# yum -y install openstack-glance
```

Создадим базу данных. Первым делом подключимся как пользователь root:

```
[root@controller ~]# mysql -u root -p
Enter password: openstack
```

После чего необходимо непосредственно создать базу данных

```
MariaDB [(none)]> CREATE DATABASE glance;
```

и выдать на нее правильные права:

```
MariaDB [(none)]> GRANT ALL PRIVILEGES ON glance.* TO
'glance'@'localhost'    IDENTIFIED BY 'glance';
MariaDB [(none)]> GRANT ALL PRIVILEGES ON glance.* TO
'glance'@'%' IDENTIFIED BY 'glance';
```

Теперь необходимо обновить настройки в конфигурационных файлах обоих сервисов. Для начала прописываем строку подключения к базе данных:

```
[root@controller ~]# crudini --set /etc/glance/glance-registry.
conf database connection mysql+pymysql://glance:glance@controller.
test.local/glance
[root@controller ~]# crudini --set /etc/glance/glance-api.conf
database connection mysql+pymysql://glance:glance@controller.test.
local/glance
```

И завершаем настройку базы данных:

```
[root@controller ~]# su -s /bin/sh -c "glance-manage db_sync" glance
```

Далее укажем проект, предназначенный для сервисов, имя пользователя в keystone и пароль для `glance-api`. Напомним, все сервисы OpenStack группируются в специальный сервисный проект, который мы создали в предыдущей главе и назвали service. В качестве пароля везде, за исключением MariaDB, мы используем «openstack».

```
[root@controller ~]# crudini --set /etc/glance/glance-api.conf
paste_deploy flavor keystone
[root@controller ~]# crudini --set /etc/glance/glance-api.conf
keystone_authtoken auth_uri http://controller.test.local:5000
[root@controller ~]# crudini --set /etc/glance/glance-api.conf
keystone_authtoken auth_url http://controller.test.local:35357
[root@controller ~]# crudini --set /etc/glance/glance-api.conf
keystone_authtoken auth_type password
[root@controller ~]# crudini --set /etc/glance/glance-api.conf
keystone_authtoken project_domain_name default
[root@controller ~]# crudini --set /etc/glance/glance-api.conf
keystone_authtoken user_domain_name default
[root@controller ~]# crudini --set /etc/glance/glance-api.conf
keystone_authtoken project_name service
[root@controller ~]# crudini --set /etc/glance/glance-api.conf
keystone_authtoken username glance
[root@controller ~]# crudini --set /etc/glance/glance-api.conf
keystone_authtoken password openstack
```

Теперь в секции `[glance_store]` укажем, что мы используем локальную файловую систему, и место расположения файлов образов:

```
[root@controller ~]# crudini --set /etc/glance/glance-api.conf
glance_store default_store file
[root@controller ~]# crudini --set /etc/glance/glance-api.conf
glance_store filesystem_store_datadir /var/lib/glance/images/
```

Нужно отметить, что Glance одновременно поддерживает несколько хранилищ для образов виртуальных машин. То, где создается образ, зависит от приоритета и свободного места на диске.

Например, если бы у нас в /var/lib/glance/images/ были бы смонтированы два диска, мы могли бы указать их с разным приоритетом:

```
[root@controller ~]# crudini --set /etc/glance/glance-api.conf
glance_store filesystem_store_datadirs /var/lib/glance/images/
mountA/:10
[root@controller ~]# crudini --set /etc/glance/glance-api.conf
glance_store filesystem_store_datadirs /var/lib/glance/images/
mountB/:20
```

Обратите внимание, что в данном случае указана опция `filesystem_store_datadirs`, а не `filesystem_store_datadir`.

Продолжим настройку. Те же настройки keystone, что мы задавали для `glance-api`, укажем и для `glance-registry`:

```
[root@controller ~]# crudini --set /etc/glance/glance-registry.
conf paste_deploy flavor keystone
[root@controller ~]# crudini --set /etc/glance/glance-registry.
conf keystone_authtoken auth_uri http://controller.test.
local:5000
[root@controller ~]# crudini --set /etc/glance/glance-registry.
conf keystone_authtoken auth_url http://controller.test.
local:35357
[root@controller ~]# crudini --set /etc/glance/glance-registry.
conf keystone_authtoken auth_type password
[root@controller ~]# crudini --set /etc/glance/glance-registry.
conf keystone_authtoken project_domain_name default
[root@controller ~]# crudini --set /etc/glance/glance-registry.
conf keystone_authtoken user_domain_name default
[root@controller ~]# crudini --set /etc/glance/glance-registry.
conf keystone_authtoken project_name service
[root@controller ~]# crudini --set /etc/glance/glance-registry.
conf keystone_authtoken username glance
[root@controller ~]# crudini --set /etc/glance/glance-registry.
conf keystone_authtoken password openstack
```

Добавляем в конфигурационные файлы информацию по сервису RabbitMQ:

```
[root@controller ~]# crudini --set /etc/glance/glance-api.conf
DEFAULT rabbit_password openstack
[root@controller ~]# crudini --set /etc/glance/glance-api.conf
DEFAULT rabbit_userid openstack
[root@controller ~]# crudini --set /etc/glance/glance-api.conf
DEFAULT rabbit_host controller.test.local
```

На этом этапе могут стартовать сервисы:

```
[root@controller ~]# systemctl start openstack-glance-registry
[root@controller ~]# systemctl enable openstack-glance-registry
```

```
[root@controller ~]# systemctl start openstack-glance-api
[root@controller ~]# systemctl enable openstack-glance-api
```

После чего нужно убедиться в отсутствии ошибок в файлах журналов /var/log/glance/*.

Подготовка образов виртуальных машин

Для того чтобы проверить работу нашего сервиса, мы воспользуемся образом CirrOS – минималистской операционной системой GNU/Linux, специально созданной для запуска в облаке (https://launchpad.net/cirros). Для выполнения следующих команд вам необходимо предварительно установить пакеты wget и qemu-img-ev. Размер образа – всего лишь 13 Мб:

```
$ wget -P /tmp http://download.cirros-cloud.net/0.4.0/cirros-
0.4.0-x86_64-disk.img
```

При необходимости установите пакет wget. Проверим образ при помощи утилиты `qemu-img`:

```
$ qemu-img info /tmp/cirros-0.4.0-x86_64-disk.img
image: /tmp/cirros-0.4.0-x86_64-disk.img
file format: qcow2
virtual size: 44M (46137344 bytes)
disk size: 12M
cluster_size: 65536
Format specific information:
    compat: 1.1
    lazy refcounts: false
    refcount bits: 16
    corrupt: false
```

В выводе команды:

- `file format` – формат диска;
- `virtual size` – размер диска виртуальной машины;
- `disk size` – действительный размер файла;
- `cluster_size` – размер блока (кластера) qcow;
- `format specific information` – специфичная для формата информация. В данном случае версия формата qcow2. В нашем примере указано `compat: 1.1`. Это означает новую версию образа qcow2, который поддерживается начиная с QEMU 1.1.

Нужно отметить, что в составе официальной документации OpenStack работе с образами посвящено отдельное руководство – OpenStack Virtual Machine Image Guide (http://docs.openstack.org/image-guide/content/). Документ содержит как обзор средств автоматизированного создания образов, так и примеры их создания вручную.

В качестве примера рассмотрим работу с утилитой Oz (https://github.com/clalancette/oz/wiki).

Это утилита командной строки написана на Python. На Linux-машине с гипервизором KVM и сервисом libvirtd позволяет создавать образы виртуальных машин с минимальным вмешательством пользователя. Для этого Oz использует заранее подготовленные файлы ответов неинтерактивной установки операционной системы. Например, для установки Windows используются файлы ответов unattended setup, для CentOS – kickstart-файлы и т. д. Файлы ответов для различных операционных систем расположены в директории /usr/lib/python2.7/site-packages/oz/auto/. Их можно и нужно редактировать, например для выбора правильной временной зоны. В качестве инструкций сам Oz требует файлов в формате Template Description Language (TDL). Это XML-файлы, описывающие, какая операционная система устанавливается, где находится дистрибутив, какие дополнительные изменения необходимо внести в образ.

Не рекомендуется запуск утилиты на тех же узлах, где развернуты сервисы OpenStack. Во время работы утилита запускает виртуальную машину, в которой и производится установка операционной системы по вашим инструкциям TDL шаблона с использованием общих файлов ответов.

В CentOS и Fedora установка утилиты производится командой

```
# yum -y install oz
```

Для Ubuntu в стандартных репозиториях пакета нет, поэтому вам нужно либо установить утилиту из исходных кодов, либо самостоятельно собрать пакет. Убедимся в том, что определена сеть libvirtd, используемая по умолчанию. Если вывод команды `virsh net-list` не покажет нам сеть default, то определим ее и зададим автозапуск сети:

```
# virsh net-define /usr/share/libvirt/networks/default.xml
Network default defined from /usr/share/libvirt/networks/default.xml
# virsh net-autostart default
```

```
Network default marked as autostarted
# virsh net-list
 Name                 State      Autostart     Persistent
----------------------------------------------------------
 default              active     yes           yes
```

Конфигурационный файл утилиты по умолчанию – /etc/oz/oz.cfg. Зададим в качестве типа образа формат qcow2 вместо raw:

```
# crudini --set /etc/oz/oz.cfg libvirt image_type qcow2
```

Теперь нам потребуется TDL-шаблон. Примеры поставляются с утилитой и располагаются в /usr/share/doc/oz-*/examples. Используем самый простейший, в котором определяются только путь к дистрибутиву и пароль пользователя root:

```
$ cat my-template.tdl
<template>
<name>CentOS-7</name>
<os>
<name>CentOS-7</name>
<version>1</version>
<arch>x86_64</arch>
<rootpw>openstack</rootpw>
<install type='url'>
<url>http://centos-mirror.rbc.ru/pub/centos/7/os/x86_64/</url>
</install>
</os>
<disk>
<size>20</size>
</disk>
</template>
```

Из полезных секций, которые можно найти в примерах, можно отметить: <packages>, <repositories>, <files> и <commands>. Соответственно: установка пакетов, добавление репозиториев, создание файлов и выполнение команд. Есть секции, специфичные для операционных систем. В качестве примера можно назвать ключи активации для Windows.

Запускаем утилиту:

```
# oz-install -d 2 -t 4000 my-template.tdl
Libvirt network without a forward element, skipping
Checking for guest conflicts with CentOS-7
Fetching the original media
Fetching the original install media from http://centos-mirror.rbc.
ru/pub/centos/7/os/x86_64/images/pxeboot/vmlinuz
Fetching the original media
```

```
Fetching the original install media from http://centos-mirror.rbc.
ru/pub/centos/7/os/x86_64/images/pxeboot/initrd.img
Generating 20GB diskimage for CentOS-7
Running install for CentOS-7
Generate XML for guest CentOS-7 with bootdev None
Install of CentOS-7 succeeded
Generate XML for guest CentOS-7 with bootdev hd
Cleaning up after install
Libvirt XML was written to CentOS-7Sep_30_2015-00:31:23
```

Ключ `-t 3000` говорит о том, через сколько секунд инсталлятор должен прервать установку. Также установка прервется, если в течение 300 секунд не было дисковой активности.

Ключ `-d` показывает уровень сообщений об ошибках. Для более подробного вывода укажите вместо двойки тройку.

Далее, вне зависимости, используете ли вы OZ или самостоятельно готовите образ виртуальной машины, например при помощи `virt-builder` из пакета libguestfs-tools, необходимо или вручную, или при помощи утилиты `virt-sysprep` из того же пакета убрать специфичную для конкретного экземпляра машины информацию. Для систем Windows можно воспользоваться утилитой `sysprep`. Пример использования утилиты `virt-sysprep` для образа с CentOS:

```
# virt-sysprep -a /var/lib/libvirt/images/centos7.0-1.qcow2
[   0.0] Examining the guest ...
[  54.4] Performing "abrt-data" ...
[  54.4] Performing "bash-history" ...
...
[  54.5] Setting a random seed
[  54.9] Performing "lvm-uuids" ...
```

Те же действия можно произвести и вручную. Приведем пример основных шагов для CentOS:

- Необходимо удалить или заменить на localhost.localdomain содержимое параметра HOSTNAME в файле cat /etc/sysconfig/network.
- Следует убедиться, что в файлах всех сетевых адаптеров, кроме интерфейса короткой петли, в параметре BOOTPROTO указан DHCP и отсутствуют привязки к MAC-адресам (параметр HWADDR). Имена конфигурационных файлов – /etc/sysconfig/network-scripts/ifcfg-*.
- Необходимо удалить SSH-ключи узла. Удалите /etc/ssh/ssh_host_* и /etc/ssh/moduli.

- Удалите файлы ls /etc/udev/rules.d/*persistent*, отвечающие за именование сетевых устройств.
- Убедитесь, что отсутствуют специфичные для сервисов keytab-файлы Kerberos и SSL- сертификаты.

Еще одна полезная утилита `virt-sparsify` позволяет уменьшить размер образа, превратив его в «тонкий диск/файл» (thin-provisioned/sparse). При запуске необходимо указать изначальный образ и создаваемый «тонкий»:

```
# virt-sparsify /var/lib/libvirt/images/centos7.0-1.qcow2 /var/
lib/libvirt/images/centos7.0-2.qcow2
[   0.2] Create overlay file in /tmp to protect source disk
[   0.4] Examine source disk
[   1.9] Fill free space in /dev/centos/root with zero 100%
□░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░□ 00:00
[  38.2] Clearing Linux swap on /dev/centos/swap
[  40.6] Fill free space in /dev/sda1 with zero  100%
□░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░□ --:--
[  43.3] Fill free space in volgroup centos with zero
[  64.6] Copy to destination and make sparse
[ 112.2] Sparsify operation completed with no errors.
virt-sparsify: Before deleting the old disk, carefully check that
the target disk boots and works correctly.
```

Сравним размеры исходного и полученного образов:

```
# ls -lh /var/lib/libvirt/images/centos7.0-1.qcow2 /var/lib/libvirt/
images/centos7.0-2.qcow2
-rw-------. 1 qemu qemu  11G Jan 23 21:17 /var/lib/libvirt/images/
centos7.0-1.qcow2
-rw-r--r--. 1 root root 961M Jan 23 21:21 /var/lib/libvirt/images/
centos7.0-2.qcow2
```

При необходимости изменить готовый образ виртуальной машины вы можете воспользоваться утилитой guestfish:

```
# yum -y install guestfish
```

Утилита содержит встроенную подсказку и достаточно простые команды. Ниже приведен пример для редактирования файла /etc/issue внутри образа:

```
# guestfish -a ubuntu_image.img

Welcome to guestfish, the guest filesystem shell for
editing virtual machine filesystems and disk images.

Type: 'help' for help on commands
```

```
      'man' to read the manual
      'quit' to quit the shell

><fs> run
 100% ⟦▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒⟧ 00:00
><fs> list-filesystems
/dev/sda1: ext4
><fs> mount /dev/sda1 /
><fs> edit /etc/issue
><fs> cat /etc/issue
Ubuntu 15.10
modified
><fs> exit
```

Работаем с образами виртуальных машин

Снова вернемся к нашему сервису и дистрибутиву cirros. Прежде чем продолжить работу, добавим в оба рабочих файла keystonerc_* переменную среды, определяющую версию Glance API, с которой мы будем работать:

```
export OS_IMAGE_API_VERSION=2
```

При помощи команды `openstack` загрузим образ cirros в наш сервис. При этом укажем формат образа и то, что этот образ будет доступен во всех проектах, то есть будет публичным. Доступность образа (публичный или непубличный) позже при необходимости можно поменять. Загружаем образ:

```
$ openstack image create "cirros-0.4.0-x86_64" --file /tmp/cirros-0.4.0-
x86_64-disk.img --disk-format qcow2 --container-format bare --public
+------------------+------------------------------------------------------+
| Field            | Value                                                |
+------------------+------------------------------------------------------+
| checksum         | 443b7623e27ecf03dc9e01ee93f67afe                     |
| container_format | bare                                                 |
| created_at       | 2018-03-02T15:27:23Z                                 |
| disk_format      | qcow2                                                |
| file             | /v2/images/c8ccc9b3-29bb-4220-be38-8f261ac8b99a/file |
| id               | c8ccc9b3-29bb-4220-be38-8f261ac8b99a                 |
| min_disk         | 0                                                    |
| min_ram          | 0                                                    |
| name             | cirros-0.4.0-x86_64                                  |
| owner            | 7fe2a6ef08df4a749f3bad1fceb055b9                     |
| protected        | False                                                |
| schema           | /v2/schemas/image                                    |
```

```
| size             | 12716032                                                  |
| status           | active                                                    |
| tags             |                                                           |
| updated_at       | 2018-03-02T15:27:24Z                                      |
| virtual_size     | None                                                      |
| visibility       | public                                                    |
+------------------+-----------------------------------------------------------+
```

Если возникли ошибки, проверьте файлы журнала сервиса less /var/log/glance/*. По умолчанию сервис в качестве бэкэнда использует локальную файловую систему. Проверим:

```
[root@controller ~]#  ls -l /var/lib/glance/images/
total 12420
-rw-r----- 1 glance glance 12716032 Mar  2 16:27 c8ccc9b3-29bb-
4220-be38-8f261ac8b99a
```

Далее мы продолжим пользоваться командой openstack, но обратите внимание, что взаимодействовать с сервисом можно при помощи клиента glance:

```
$ glance image-create --name "cirros-0.4.0-x86_64" --file /tmp/
cirros-0.4.0-x86_64-disk.img --disk-format qcow2 --container-format
bare --visibility public
```

Как мы видим, наш образ был загружен в директорию /var/lib/glance/images/ под именем, совпадающим с идентификатором образа. Просмотрим список доступных образов, к которым должен был добавиться наш новый cirros:

```
$ openstack image list
+--------------------------------------+---------------------+--------+
| ID                                   | Name                | Status |
+--------------------------------------+---------------------+--------+
| c8ccc9b3-29bb-4220-be38-8f261ac8b99a | cirros-0.4.0-x86_64 | active |
+--------------------------------------+---------------------+--------+
```

Просмотреть подробную информацию по образу можно при помощи команды:

```
$ openstack image show cirros-0.4.0-x86_64
+------------------+-----------------------------------------------------+
| Field            | Value                                               |
+------------------+-----------------------------------------------------+
| checksum         | 443b7623e27ecf03dc9e01ee93f67afe                    |
| container_format | bare                                                |
| created_at       | 2018-03-02T15:27:23Z                                |
| disk_format      | qcow2                                               |
```

```
| file             | /v2/images/c8ccc9b3-29bb-4220-be38-8f261ac8b99a/file  |
| id               | c8ccc9b3-29bb-4220-be38-8f261ac8b99a                  |
| min_disk         | 0                                                     |
| min_ram          | 0                                                     |
| name             | cirros-0.4.0-x86_64                                   |
| owner            | 7fe2a6ef08df4a749f3bad1fceb055b9                      |
| protected        | False                                                 |
| schema           | /v2/schemas/image                                     |
| size             | 12716032                                              |
| status           | active                                                |
| tags             |                                                       |
| updated_at       | 2018-03-02T15:27:24Z                                  |
| virtual_size     | None                                                  |
| visibility       | public                                                |
+------------------+-------------------------------------------------------+
```

К этому моменту образ готов для создания виртуальных машин. Как это делать, мы познакомимся в дальнейших главах, когда установим службы Nova и Neutron.

За счет своего малого размера CirrOS обладает очень ограниченным функционалом и рекомендуется только для тестирования облачных сервисов. Большинство сборщиков дистрибутивов GNU/Linux предоставляет готовые к использованию в облаке OpenStack образы. Вот ссылки на некоторые из них:

- Ubuntu: http://cloud-images.ubuntu.com/;
- Fedora: https://cloud.fedoraproject.org/;
- Debian: http://cdimage.debian.org/cdimage/openstack/;
- CentOS: http://cloud.centos.org/centos/7/.

Ориентированные на работу в облаке с контейнерами:

- CoreOS: https://coreos.com/os/docs/latest/booting-on-openstack.html;
- Project Atomic: http://www.projectatomic.io/download/.

При помощи команды `openstack` можно выполнять другие действия. Например, загрузим и сохраним локально образ из glance:

```
$ openstack image save cirros-0.4.0-x86_64 > cirros-0.4.0-x86_64-
disk.img
$ ls -l cirros-0.4.0-x86_64-disk.img
-rw-r--r-- 1 root root 12716032 Mar  2 16:42 cirros-0.4.0-x86_64-
disk.img
```

Вместе с образом вы можете хранить произвольные метаданные, которые потом могут быть затребованы другими сервисами

или утилитами. Также можно определять метаданные просто для идентификации образа. Например:

```
$ openstack image set --property os_name=linux --property contact_
person="Andrey Markelov" cirros-0.4.0-x86_64
$ openstack image show cirros-0.4.0-x86_64
+------------------+------------------------------------------------------+
| Field            | Value                                                |
+------------------+------------------------------------------------------+
| checksum         | 443b7623e27ecf03dc9e01ee93f67afe                     |
| container_format | bare                                                 |
| created_at       | 2018-03-02T15:27:23Z                                 |
| disk_format      | qcow2                                                |
| file             | /v2/images/c8ccc9b3-29bb-4220-be38-8f261ac8b99a/file |
| id               | c8ccc9b3-29bb-4220-be38-8f261ac8b99a                 |
| min_disk         | 0                                                    |
| min_ram          | 0                                                    |
| name             | cirros-0.4.0-x86_64                                  |
| owner            | 7fe2a6ef08df4a749f3bad1fceb055b9                     |
| properties       | contact_person='Andrey Markelov', os_name='linux'    |
| protected        | False                                                |
| schema           | /v2/schemas/image                                    |
| size             | 12716032                                             |
| status           | active                                               |
| tags             |                                                      |
| updated_at       | 2018-03-02T15:38:46Z                                 |
| virtual_size     | None                                                 |
| visibility       | public                                               |
+------------------+------------------------------------------------------+
```

При необходимости удалить те или иные метаданные можно воспользоваться командой

```
$ openstack image unset --property os_name --property contact_
person cirros-0.4.0-x86_64
```

Глава 5

Сервис блочного хранилища Cinder

```
Название: OpenStack Block Storage (Cinder)
Назначение: сервис блочного хранилища
Пакет:  openstack-cinder
Имена сервисов: openstack-cinder-api, openstack-cinder-scheduler,
openstack-cinder-volume, openstack-cinder-backup
Порт: 3260/tcp (iscsi-target), 8776/tcp
Конфигурационный файл: /etc/cinder/cinder.conf
Файлы журнала: /var/log/cinder/*
```

В состав базовых проектов OpenStack входят два принципиально разных сервиса хранения информации. OpenStack Swift, рассматриваемый в следующей главе, представляет из себя объектное хранилище, подобное Amazon S3. В этой главе мы разберемся с сервисом OpenStack Cinder, похожим на Amazon EBS.

Архитектура Cinder

Когда сервис Nova, рассматриваемый в седьмой главе, получает образ виртуальной машины из сервиса Glance, образ копируется на локальный диск вычислительного узла, где работает Nova, и из образа запускается виртуальная машина. Между перезагрузками виртуальной машины изменения, произведенные на файловой системе, сохраняются в локальной копии образа, которая по умолчанию хранится в /var/lib/nova/instances. В тот момент, когда виртуальная машина удаляется, ее оперативные данные пропадают, и у нас остается только то, что изначально содержалось в образе. Соответственно, необходимо хранилище постоянной информации, где данные сохранялись бы между перезагрузками или в случае сбоя узла с запущенными виртуальными машинами. Такое храни-

лище и предоставляет сервис Cinder – «блочное хранилище данных как сервис».

Помимо функционирования в качестве постоянного дополнительного хранилища для виртуальных машин без сохранения состояния, Cinder также может использоваться в качестве устройства начальной загрузки. Кроме того, можно создавать снимки томов, доступные в режиме «только чтение». В дальнейшем их можно использовать для создания новых томов, доступных на запись.

Блочные устройства можно одновременно подключать к нескольким виртуальным машинам. Однако, чтобы обеспечить целостность данных, они должны быть либо подключены в режиме «только чтение», либо поверх блочного устройства должна быть создана кластерная файловая система.

Архитектура Cinder представлена на рис. 5.1.

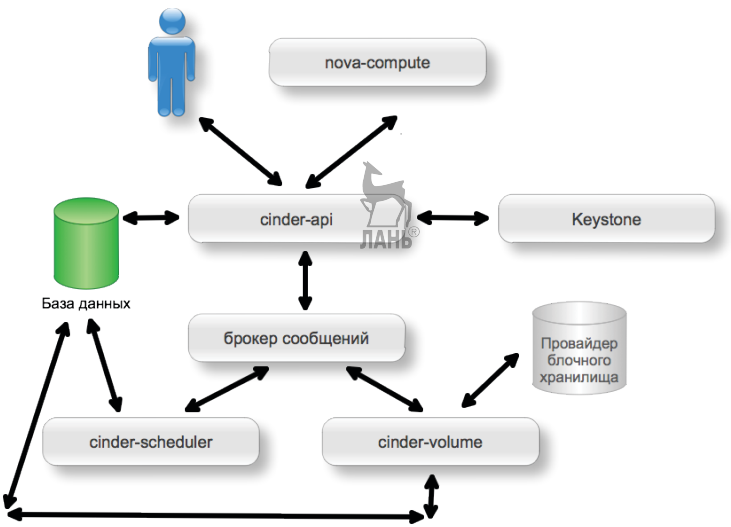


Рис. 5.1. Архитектура Cinder

Сервис состоит из четырех служб:

- **openstack-cinder-api** – точка входа для запросов в сервис по протоколу HTTP. Приняв запрос, сервис проверяет полномочия на выполнение запроса и переправляет запрос брокеру сообщений для доставки другим службам;
- **openstack-cinder-scheduler** – сервис-планировщик принимает запросы от брокера сообщений и определяет, какой

узел с сервисом openstack-cinder-volume должен обработать запрос;

- **openstack-cinder-volume** – сервис отвечает за взаимодействие с бэкэндом – блочным устройством. Получает запросы от планировщика и транслирует непосредственно в хранилище. Cinder позволяет одновременно использовать несколько бэкэндов. При этом для каждого из них запускается свой openstack-cinder-volume. И при помощи параметров CapacityFilter и CapacityWeigher можно управлять тем, какой бэкэнд выберет планировщик;
- **openstack-cinder-backup** – сервис отвечает за создание резервных копий томов в объектное хранилище.

Существует множество драйверов cinder для всевозможных систем хранения данных (СХД), как программных, так и аппаратных. Матрица совместимости доступна по адресу https://wiki.openstack.org/wiki/CinderSupportMatrix.

Настройка сервисов Cinder

Обычно внедряют много узлов с сервисом openstack-cinder-volume, которые непосредственно отвечают за доступ к данным, и несколько управляющих, обеспечивающих доступ к API и планировщику. Мы продолжим развертывать все службы на управляющем узле controller.test.local. Как самый простой вариант в качестве блочного устройства будем использовать Linux LVM, а для доступа протокол iSCSI. Для этого необходимо либо создать дополнительный диск, либо использовать свободное пространство на уже имеющемся. Поскольку предполагается, что мы работаем с виртуальной машиной, проще всего добавить еще один диск средствами используемой вами системы виртуализации. Если бы сервисы были разнесены по узлам, то эти действия мы бы выполняли только на серверах, где планируется запуск openstack-cinder-volume.

На свободном разделе выполняем команды по созданию LVM-группы. Физический том LVM создастся автоматически:

```
[root@controller ~]# vgcreate cinder-volumes /dev/vdb1
  Physical volume "/dev/vdb1" successfully created
  Volume group "cinder-volumes" successfully created
```

Устанавливаем пакеты Cinder:

```
[root@controller ~]# yum -y install openstack-cinder
```

Внесем необходимые настройки. Начнем со строки подключения к базе данных:

```
[root@controller ~]# crudini --set /etc/cinder/cinder.conf database
connection mysql+pymysql://cinder:cinder@controller.test.local/cinder
```

Укажем местоположение брокера сообщений, имя и пароль пользователя RabbitMQ, а также реквизиты пользователя и проекта в Keystone:

```
[root@controller ~]# crudini --set /etc/cinder/cinder.conf DEFAULT
transport_url rabbit://openstack:openstack@controller.test.local
```

Как и раньше, мы используем в качестве пароля «openstack», а проект для служебных пользователей называется service:

```
[root@controller ~]# crudini --set /etc/cinder/cinder.conf DEFAULT
auth_strategy keystone
[root@controller ~]# crudini --set /etc/cinder/cinder.conf
keystone_authtoken project_name service
[root@controller ~]# crudini --set /etc/cinder/cinder.conf
keystone_authtoken user_domain_name default
[root@controller ~]# crudini --set /etc/cinder/cinder.conf
keystone_authtoken project_domain_name default
[root@controller ~]# crudini --set /etc/cinder/cinder.conf
keystone_authtoken auth_type password
[root@controller ~]# crudini --set /etc/cinder/cinder.conf
keystone_authtoken username cinder
[root@controller ~]# crudini --set /etc/cinder/cinder.conf
keystone_authtoken password openstack
[root@controller ~]# crudini --set /etc/cinder/cinder.conf
keystone_authtoken auth_uri http://controller.test.local:5000
[root@controller ~]# crudini --set /etc/cinder/cinder.conf
keystone_authtoken auth_url http://controller.test.local:35357
```

По умолчанию Cinder использует в качестве хранилища именно LVM. Укажем имя группы томов LVM, в которой будут создаваться блочные устройства Cinder:

```
[root@controller ~]# crudini --set /etc/cinder/cinder.conf lvm
volume_group cinder-volumes
```

Указываем бэкэнд и параметр volume_driver:

```
[root@controller ~]# crudini --set /etc/cinder/cinder.conf DEFAULT
enabled_backends lvm
[root@controller ~]# crudini --set /etc/cinder/cinder.conf lvm
volume_driver cinder.volume.drivers.lvm.LVMVolumeDriver
```

Этот драйвер и отвечает за хранение данных при помощи локального менеджера логических томов и протокола транспорта iSCSI. В главе, посвященной Ceph, мы в том числе заменим этот драйвер на соответствующий Ceph.

Задаем параметры iSCSI:

```
[root@controller ~]# crudini --set /etc/cinder/cinder.conf lvm
iscsi_protocol iscsi
[root@controller ~]# crudini --set /etc/cinder/cinder.conf lvm
iscsi_helper lioadm
```

Указываем расположение API сервиса Glance:

```
[root@controller ~]# crudini --set /etc/cinder/cinder.conf DEFAULT
glance_api_servers http://controller.test.local:9292
```

И путь к lock-файлу:

```
[root@controller ~]# crudini --set /etc/cinder/cinder.conf oslo_
concurrency lock_path /var/lib/cinder/tmp
```

Теперь выполняем уже знакомые по сервисам Glance и Keystone операции. Создаем базу данных:

```
# mysql -u root -p
Enter password: openstack
MariaDB [(none)]> CREATE DATABASE cinder;
MariaDB [(none)]> GRANT ALL PRIVILEGES ON cinder.* TO
'cinder'@'localhost' IDENTIFIED BY 'cinder';
MariaDB [(none)]> GRANT ALL PRIVILEGES ON cinder.* TO
'cinder'@'%' IDENTIFIED BY 'cinder';
```

Подготавливаем базу данных:

```
[root@controller ~]# su -s /bin/sh -c "cinder-manage db sync" cinder
```

Заводим пользователя в Keystone и добавляем роль администратора в проекте service:

```
$ source keystonerc_admin
$ openstack user create --domain default --password openstack cinder
+---------------------+----------------------------------+
| Field               | Value                            |
+---------------------+----------------------------------+
| domain_id           | default                          |
| enabled             | True                             |
| id                  | 86c5aa25bb134478bcaa9f67fcd0feb2 |
| name                | cinder                           |
| options             | {}                               |
```

```
| password_expires_at | None                                |
+---------------------+-------------------------------------+
$ openstack role add --project service --user cinder admin
```

Или то же самое, используя команду keystone в старых версиях OpenStack:

```
$ keystone user-create --name cinder --pass openstack
$ keystone user-role-add --user cinder --role admin --tenant service
```

Теперь нам нужно, в отличие от ранее рассмотренных служб OpenStack, создать целых два сервиса: cinderv2 и cinderv3. Это связано с тем, что в настоящий момент параллельно используются две версии Cinder API. До версии Newton также использовалась первая версия:

```
openstack service create --name cinderv2 --description "OpenStack
Block Storage" volumev2
+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description | OpenStack Block Storage          |
| enabled     | True                             |
| id          | b186a1daee4c4147abe4d454b7431298 |
| name        | cinderv2                         |
| type        | volumev2                         |
+-------------+----------------------------------+
$ openstack service create --name cinderv3 --description "OpenStack
Block Storage" volumev3
+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description | OpenStack Block Storage          |
| enabled     | True                             |
| id          | 717ed5d914a54a3ba8b576ea2379ffe6 |
| name        | cinderv3                         |
| type        | volumev3                         |
+-------------+----------------------------------+
```

Создаем по три точки входа API для обоих сервисов:

```
$ openstack endpoint create --region RegionOne volumev2 public
http://controller:8776/v2/%\(project_id\)s
$ openstack endpoint create --region RegionOne volumev2 internal
http://controller:8776/v2/%\(project_id\)s
$ openstack endpoint create --region RegionOne volumev2 admin
http://controller:8776/v2/%\(project_id\)s

$ openstack endpoint create --region RegionOne volumev3 public
http://controller:8776/v3/%\(project_id\)s
```

```
$ openstack endpoint create --region RegionOne volumev3 internal
http://controller:8776/v3/%\(project_id\)s
$ openstack endpoint create --region RegionOne volumev3 admin
http://controller:8776/v3/%\(project_id\)s
```

Теперь устанавливаем и настраиваем сервис iSCSI. Опять же, если бы мы разносили сервисы по узлам, то эти действия мы выполняли бы только на серверах, где планируется запуск openstack-cinder-volume:

```
[root@controller ~]# yum -y install targetcli
[root@controller ~]# crudini --set /etc/cinder/cinder.conf
DEFAULT my_ip 192.168.122.200
[root@controller ~]# crudini --set /etc/cinder/cinder.conf
DEFAULT enabled_backends lvm
[root@controller ~]# systemctl enable target.service
[root@controller ~]# systemctl start target.service
```

Задаем автоматический старт сервисов Cinder после перезагрузки:

```
[root@controller ~]# systemctl enable openstack-cinder-api
[root@controller ~]# systemctl enable openstack-cinder-scheduler
[root@controller ~]# systemctl enable openstack-cinder-volume
[root@controller ~]# systemctl enable openstack-cinder-backup
```

Для разнообразия запустить все сервисы можно воспользоваться скриптом `openstack-service`, предварительно установив пакет `openstack-utils`:

```
[root@controller ~]# yum -y install openstack-utils
[root@controller ~]# openstack-service start cinder
```

Заодно познакомимся с еще одним удобным скриптом `openstack-status`. Он должен показать ряд общей информации о службах OpenStack, в том числе то, что все четыре сервиса Cinder запущены:

```
# openstack-status
== Glance services ==
openstack-glance-api:                active
openstack-glance-registry:           active
== Keystone service ==
openstack-keystone:                  inactive  (disabled on boot)
== Cinder services ==
openstack-cinder-api:                active
openstack-cinder-scheduler:          active
openstack-cinder-volume:             active
openstack-cinder-backup:             active
```

```
== Support services ==
mariadb:                                active
dbus:                                   active
target:                                 active
rabbitmq-server:                        active
== Keystone users ==
+----------------------------------+--------+
| ID                               | Name   |
+----------------------------------+--------+
| 3b76dece42b140e092dc1a76a85c1879 | demo   |
| 86c5aa25bb134478bcaa9f67fcd0feb2 | cinder |
| a37a67fdf3dc4e9c9e5251754b26770d | admin  |
| b8de68858b00440bb98b6b1541466794 | glance |
+----------------------------------+--------+
== Glance images ==
+--------------------------------------+---------------------+
| ID                                   | Name                |
+--------------------------------------+---------------------+
| c8ccc9b3-29bb-4220-be38-8f261ac8b99a | cirros-0.4.0-x86_64 |
+--------------------------------------+---------------------+
```

Скрипт, как вы видите, в выводе указывает, что openstack-keystone не запущен. Это нормально, поскольку данный сервис не используется. Вместо openstack-keystone мы используем Apache с `mod_wsgi`. Также можно воспользоваться командой `cinder service-list`:

```
# cinder service-list
+---------------+-------------------------+------+---------+-------+----------------------------+-----------------+
| Binary        | Host                    | Zone | Status  | State | Updated_at                 | Disabled Reason |
+---------------+-------------------------+------+---------+-------+----------------------------+-----------------+
| cinder-backup | controller.test.local   | nova | enabled | down  | 2018-03-02T17:31:31.000000 | -               |
| cinder-scheduler | controller.test.local | nova | enabled | up    | 2018-03-02T17:39:43.000000 | -               |
| cinder-volume | controller.test.local@lvm | nova | enabled | up    | 2018-03-02T17:39:49.000000 | -               |
+---------------+-------------------------+------+---------+-------+----------------------------+-----------------+
```

Как можно заметить, сервис cinder-backup находится в состоянии down. На данном этапе это нормально. Проверим журнал /var/log/cinder/backup.log:

```
2018-03-02 18:39:42.821 19623 ERROR oslo.service.loopingcall
"Could not determine which Swift endpoint to use. This "
2018-03-02 18:39:42.821 19623 ERROR oslo.service.loopingcall
BackupDriverException: Backup driver reported an error: Could
not determine which Swift endpoint to use. This can either be
set in the service catalog or with the cinder.conf config option
'backup_swift_url'.
```

Как мы видим, мы не задали конечную точку сервиса объектного хранилища Swift. Далее, когда мы добавим этот сервис в наше окружение, мы исправим эту проблему.

Создание и удаление томов Cinder

Подробнее про работу с томами мы поговорим в девятой главе. Пока в нашем стенде не хватает основного потребителя услуг Cinder – сервиса Nova. Поэтому познакомимся только с базовыми операциями. Попробуем создать том с именем testvol1 размером 1 Гб:

```
$ openstack volume create --size 1 testvol1
+---------------------+--------------------------------------+
| Field               | Value                                |
+---------------------+--------------------------------------+
| attachments         | []                                   |
| availability_zone   | nova                                 |
| bootable            | false                                |
| consistencygroup_id | None                                 |
| created_at          | 2018-03-02T17:48:27.000000           |
| description         | None                                 |
| encrypted           | False                                |
| id                  | 4e8227bf-8b45-4eb1-a4db-c7a5ea5703ac |
| migration_status    | None                                 |
| multiattach         | False                                |
| name                | testvol1                             |
| properties          |                                      |
| replication_status  | None                                 |
| size                | 1                                    |
| snapshot_id         | None                                 |
| source_volid        | None                                 |
| status              | creating                             |
| type                | None                                 |
| updated_at          | None                                 |
| user_id             | a37a67fdf3dc4e9c9e5251754b26770d     |
+---------------------+--------------------------------------+
```

Соответствующий синтаксис команды `cinder` выглядит следующим образом:

```
$ cinder create --display-name testvol1 1
```

Команды `cinder list` или `openstack volume list` (ей мы не можем воспользоваться, пока не создали сервис Nova) покажут нам, что том действительно создан, но пока не подключен ни к одному из экземпляров виртуальных машин:

```
$ cinder list
+------+-----------+--------------+------+-------------+----------+-------------+
| ID   |   Status  | Display Name | Size | Volume Type | Bootable | Attached to |
```

```
+------+-----------+--------------+------+-------------+----------+-------------+
| 4e.. | available |   testvol1   |  1   |      -      |  false   |             |
+------+-----------+--------------+------+-------------+----------+-------------+
```

Ну и под конец убедимся, что действительно была использована LVM-группа `cinder-volumes`:

```
[root@controller ~]# vgs
  VG             #PV #LV #SN Attr   VSize   VFree
  centos           1   3   0 wz--n- <99.00g   4.00m
  cinder-volumes   1   2   0 wz--n- <10.00g 484.00m
```

и на ней создан том размером 1 Гб с именем, содержащим id тома из вывода команды `cinder`:

```
[root@controller ~]# lvs
  LV                                          VG             Attr       LSize  Pool              Origin Data%  Meta%
Move Log Cpy%Sync Convert
  home                                        centos         -wi-ao---- 45.99g
  root                                        centos         -wi-ao---- 50.00g
  swap                                        centos         -wi-ao----  3.00g
  cinder-volumes-pool                         cinder-volumes twi-aotz--  9.50g                        0.00   0.62
  volume-4e8227bf-8b45-4eb1-a4db-c7a5ea5703ac cinder-volumes Vwi-a-tz--  1.00g cinder-volumes-pool       0.00
```

При наличии сервиса Nova (с ним мы познакомимся в седьмой главе) подключить имеющийся том к запущенному экземпляру виртуальной машины и начать использовать этот том как блочное устройство можно при помощи команды `nova volume-attach`, например:

```
$ nova volume-attach myinstance1 be61dc8a-3a08-4ab9-996f-3ef00db28f0d
/dev/vdb
+----------+--------------------------------------+
| Property | Value                                |
+----------+--------------------------------------+
| device   | /dev/vdb                             |
| id       | be61dc8a-3a08-4ab9-996f-3ef00db28f0d |
| serverId | 5f333f08-60ba-4b0b-80d3-de2f03125c28 |
| volumeId | 82711b99-cce1-45e4-8b7e-b91301d65dc8 |
+----------+--------------------------------------+
```

Через некоторое время в выводе команд `cinder list` или `openstack volume list` статус тома изменится на «in-use», что означает готовность к работе:

```
$ cinder list
+------+--------+--------------+------+-------------+----------+-------------+
| ID   | Status | Display Name | Size | Volume Type | Bootable | Attached to |
```

```
+------+--------+--------------+------+-------------+----------+-------------+
| be.. | in-use |   testvol1   |  1   |     None    |  false   | 5f333f08-.. |
+------+--------+--------------+------+-------------+----------+-------------+
```

Те же действия можно проделать и при помощи команды `openstack`:

```
$ openstack server add volume myinstance1 testvol1
$ openstack volume list
+------+--------------+--------+------+-------------------------------------+
| ID   | Display Name | Status | Size | Attached to                         |
+------+--------------+--------+------+-------------------------------------+
| be.. | testvol1     | in-use |    1 | Attached to myinstance1 on /dev/vdb |
+------+--------------+--------+------+-------------------------------------+
```

Отключить том можно командой `openstack server remove volume` или `nova volume-detach`, а удалить – командой `cinder delete`.

Внешний вид окна управления томами Cinder в веб-интерфейсе приведен на рис. 5.2.

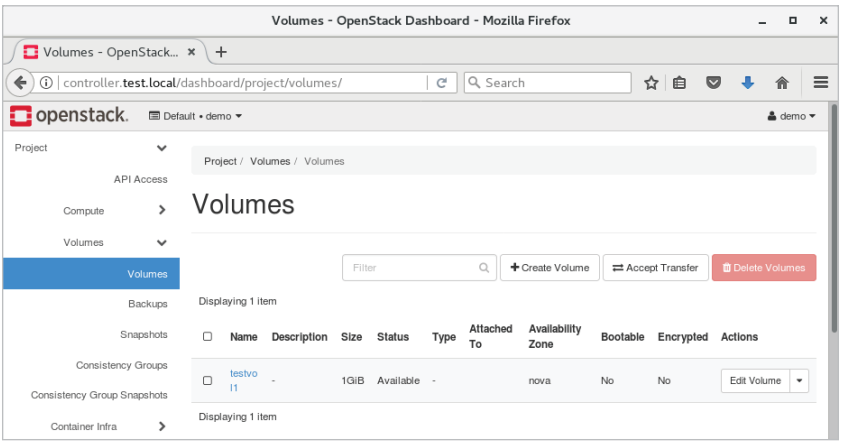


Рис. 5.2. Управление томами Cinder

Глава 6

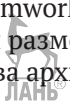
Объектное хранилище Swift

```
Название: OpenStack Object Storage
Назначение: объектное хранилище
Пакеты: openstack-swift-*
Имена сервисов:  openstack-swift-object, openstack-swift-account,
openstack-swift-container, openstack-swift-proxy
Порты: по умолчанию 8080, 6000, 6001, 6002, 873(rsync)
Конфигурационные файлы: /etc/swift/*
Файлы журнала: журнал системы /var/log/messages
```

Объектное хранилище Swift – один из двух самых первых сервисов OpenStack. Swift – это программно определяемое хранилище (software-defined storage, SDS), работающее с объектами.

Объектное хранилище, в отличие от файлового или блочного, предоставляет доступ не к файлам и блочным устройствам, а к объектам в едином пространстве имен. У объектного хранилища есть свой API, и обычно доступ к объектам осуществляется по протоколу HTTP. Такое хранилище абстрагирует объекты от их физического расположения и позволяет осуществлять масштабирование без привязки к физической инфраструктуре хранилища. Также преимуществом объектного хранилища является возможность распределять запросы по большому числу серверов, хранящих данные.

Нужно отметить, что Swift является стабильным и зрелым продуктом. Такие компании, как HP, Symantec, Softlayer, OVH, Hudson Alpha Biotech и Dreamworks, поддерживают в промышленной эксплуатации кластеры размерами в петабайты.



Основные свойства архитектуры Swift:

- линейная масштабируемость;
- отсутствие узлов с эксклюзивными ролями;

- механизмы репликации и самовосстановления. По умолчанию у каждого объекта три реплики. Соответственно, не требуется RAID-массив на узлах хранения;
- поддержка больших объектов (по умолчанию до пяти гигабайт);
- архитектура Swift обеспечивает «консистентность в конечном счете» (англоязычный термин – eventually consistent). Этот термин обозначает, что в отсутствие изменений данных в конечном счете все запросы будут возвращать последнее обновленное значение. То есть, когда вы загружаете объект в кластер, вы получаете подтверждение, что объект записан, когда придут подтверждения от двух узлов. Остальные узлы получат объект «в конечном счете». Самый распространенный пример подобной службы – DNS.

Нужно отметить, что объектное хранилище используется приложениями, а не виртуальными машинами самими по себе. Объектное хранилище, как правило, нужно, если необходимо разделять большой объем данных. Если объем небольшой, то, возможно, проще воспользоваться технологиями GlusterFS, OCFS2 или GFS2.

Архитектура Swift

Логическая структура Swift состоит из трех уровней.

- Учетная запись (account) – соответствует понятию «проект» в других сервисах OpenStack. Важно не путать учетную запись Swift с пользователем. Учетная запись сама может включать в себя множество пользователей. Ее можно сравнить с отдельной файловой системой, на которой располагаются директории-контейнеры.
- Контейнер – принадлежит учетной записи. Можно представлять себе контейнер как директорию, которой принадлежат файлы – объекты. Но, в отличие от директорий, структура контейнеров «плоская». Контейнеры не могут быть вложенными.
- Объект – единица хранения в Swift. Объектом может быть что угодно: видеофайл, образ виртуальной машины, doc-файл и т. д. С объектом также могут быть связаны метаданные.

Каждый объект можно уникально идентифицировать по имени:

```
/УчетнаяЗаписьX/КонтейнерY/объектZ
```

Физическая структура Swift не пересекается с логической и состоит из следующих компонентов:

- регион – соответствует одной площадке или центру обработки данных. Если кластер содержит несколько регионов, то Swift будет пытаться прочитать данные из ближайшего, основывая свое решение на задержке при передаче данных. Во время записи по умолчанию Swift пишет информацию во все регионы одновременно. Каждый регион имеет свою точку входа API;
- зона – набор серверов в регионе, характеризующийся общим компонентом доступности, например набор стоек, завязанных на одну линию питания, серверы, подключенные к одной паре коммутаторов, лезвия в одной корзине и т. п. Как правило, в кластере – несколько зон. Документация OpenStack рекомендует создавать минимум пять зон;
- серверы – отдельные серверы, хранящие данные;
- диски – диски серверов хранения.

Объекты хранятся как файлы на дисках с файловой системой, позволяющей хранить расширенные атрибуты файлов. Рекомендуются ext4 и XFS. Также рекомендуется достаточно большой размер inode (индексного дескриптора) для целей хранения метаданных. Например, можно задать размер, начиная от 1024.

Теперь рассмотрим, какие сервисы обслуживают Swift. Сервисы могут работать на всех узлах, или под отдельные сервисы выделяются отдельные узлы. Зачастую openstack-swift-proxy выносится на отдельные узлы, а три остальных сервиса работают на всех узлах кластера.

- **openstack-swift-proxy** – данный сервис отвечает за коммуникацию Swift с клиентами по протоколу HTTP и за распределение запросов на чтение и запись между узлами.
- **openstack-swift-account** – сервис отвечает за поддержку баз данных SQLite, содержащих информацию о контейнерах, доступных конкретной учетной записи. Для каждой учетной записи создается своя база данных, и точно так же, как и объекты, базы данных реплицируются внутри кластера.
- **openstack-swift-container** – сервис отвечает за поддержку баз данных, содержащих информацию об объектах, имеющихся в каждом контейнере. Для каждого контейнера создается своя база данных SQLite. Базы данных реплицируются внутри кластера.

- **openstack-swift-object** – отвечает за хранение, доставку и удаление объектов, сгруппированных в логические группы, называемые разделами на дисках узла. Важно, что разделы в терминологии Swift – это не разделы диска. Раздел – это директория на файловой системе, поддерживающей расширенные атрибуты файлов. Примеры таких файловых систем: XFS или ext4. Объекты хранятся в поддиректориях директории-раздела. Для идентификации объекта используется MD5-хэш пути к объекту. Метаданные, такие как, например, метка последнего доступа, хранятся в расширенных атрибутах файловой системы. Сервис позволяет хранить несколько версий одного и того же объекта, а также его срок хранения.

Помимо перечисленных сервисов, существует ряд периодически запускающихся процессов, в том числе:

- репликатор. Процесс отвечает за целостность данных. Репликацией обеспечивается то, что на всех серверах и дисках, на которых должен располагаться объект, присутствует его последняя версия. Для сравнения объектов репликатор использует хэш-функцию, а для копирования файлов – Linux-демон rsyncd;
- аудитор. Процесс периодически рассчитывает MD5-суммы файлов и сравнивает их с хранящимися в метаданных. Если обнаруживается расхождение, то файл помещается в специальную директорию на карантин, а при следующей репликации создается новая целостная копия файла;
- процесс удаления учетных записей. Удаляет учетную запись и связанные с ней контейнеры и объекты;
- процессы обновления контейнеров и объектов;
- процесс, отвечающий за автоматическое удаление объектов по прошествии определенного времени. Это позволяет задавать «срок жизни объектов».

Подготовка дополнительных серверов лабораторного окружения

Для установки сервиса OpenStack Swift мы используем три дополнительные виртуальные машины, каждой из которых мы выделим по 1 Гб оперативной памяти. Для промышленной эксплуатации

этого, конечно, мало, но для отработки материала главы достаточно.

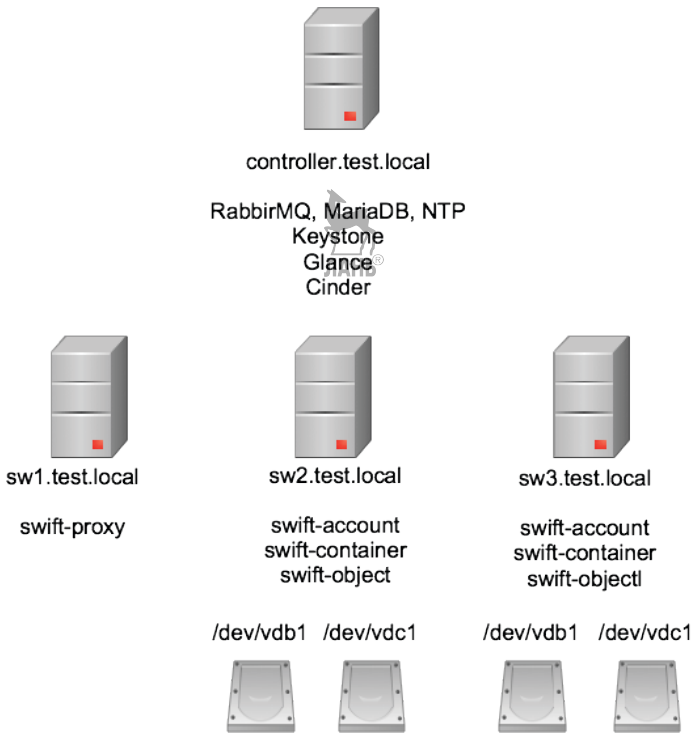


Рис. 6.1. Схема стенда Swift

Серверу sw1.test.local назначим роль Swift proxy, а два оставшихся – sw2 и sw3 – мы будем использовать в качестве серверов хранения. Для начала установим на все три виртуальные машины CentOS в варианте минимальной установки и настроим репозитории, как указано во второй главе в разделе «Подготовка CentOS 7 к использованию дистрибутива RDO». Кроме того, необходимо убедиться, что все четыре виртуальные машины могут разрешать имена друг друга и между ними есть IP-связанность. В отсутствие DNS-сервера можно просто реализовать это через общий файл /etc/hosts.

Наконец, проделаем уже знакомые нам шаги. Создаем пользователя, добавляем его с ролью admin в проект service, создаем сервис и с использованием идентификатора сервиса создаем точку

входа в сервис. Обратите внимание, что точка входа располагается на сервере sw1:

```
$ source keystonerc_admin
$ openstack user create --domain default --password openstack swift
+---------------------+----------------------------------+
| Field               | Value                            |
+---------------------+----------------------------------+
| domain_id           | default                          |
| enabled             | True                             |
| id                  | d2d83c0cbab24ba7ac7eaecd487b677d |
| name                | swift                            |
| options             | {}                               |
| password_expires_at | None                             |
+---------------------+----------------------------------+
$ openstack role add --project service --user swift admin
$ openstack service create --name swift --description "OpenStack
Object Storage" object-store
$ openstack endpoint create --region RegionOne object-store
public http://sw1.test.local:8080/v1/AUTH_%\(project_id\)s
$ openstack endpoint create --region RegionOne object-store
internal http://sw1.test.local:8080/v1/AUTH_%\(project_id\)s
$ openstack endpoint create --region RegionOne object-store admin
http://sw1.test.local:8080/v1
```

Теперь можно подключиться к виртуальной машине, где мы будем настраивать сервис Swift-proxy, – sw1.test.local.

Установка сервиса Swift-proxy

Первым делом установим необходимые пакеты на виртуальную машину sw1.test.local:

```
[root@sw1 ~]# yum -y install openstack-swift-proxy python-swiftclient
python-keystoneclient python-keystonemiddleware memcached crudini
```

В конфигурационном файле прокси пропишем сервер Keystone и другие атрибуты, необходимые для подключения к сервису идентификации. С этими настройками мы знакомы по предыдущим главам:

```
[root@sw1 ~]# crudini --set /etc/swift/proxy-server.conf
filter:authtoken paste.filter_factory keystonemiddleware.auth_
token:filter_factory
[root@sw1 ~]# crudini --set /etc/swift/proxy-server.conf
filter:authtoken auth_uri http://controller.test.local:5000
[root@sw1 ~]# crudini --set /etc/swift/proxy-server.conf
```

```
filter:authtoken auth_url http://controller.test.local:35357
[root@sw1 ~]# crudini --set /etc/swift/proxy-server.conf
filter:authtoken auth_type password
[root@sw1 ~]# crudini --set /etc/swift/proxy-server.conf
filter:authtoken project_domain_id default
[root@sw1 ~]# crudini --set /etc/swift/proxy-server.conf
filter:authtoken user_domain_id default
[root@sw1 ~]# crudini --set /etc/swift/proxy-server.conf
filter:authtoken project_name service
[root@sw1 ~]# crudini --set /etc/swift/proxy-server.conf
filter:authtoken username swift
[root@sw1 ~]# crudini --set /etc/swift/proxy-server.conf
filter:authtoken password openstack
[root@sw1 ~]# crudini --set /etc/swift/proxy-server.conf
filter:authtoken delay_auth_decision True
```

Определим, какие роли имеют право изменять данные в учетной записи (проекте):

```
[root@sw1 ~]# crudini --set /etc/swift/proxy-server.conf
filter:keystoneauth operator_roles admin,user
[root@sw1 ~]# crudini --set /etc/swift/proxy-server.conf
filter:keystoneauth use egg:swift#keystoneauth
```

Остальные параметры оставляем по умолчанию. В частности, местоположение сервиса memcached (memory cache daemon) – базы данных в памяти для кэширования информации клиентов, прошедших аутентификацию.

Установка узлов хранения Swift

Устанавливаем необходимые пакеты на две оставшиеся виртуальные машины sw2.test.local и sw3.test.local. Команды нужно выполнять на обоих серверах:

```
[root@sw2 и 3~]# yum -y install openstack-swift-account
openstack-swift-container  openstack-swift-object crudini rsync
```

Каждой из двух виртуальных машин добавим по два локальных диска и на каждом из них создадим по одному разделу. Два выберем для упрощения стенда. Имена блочных устройств зависят от системы виртуализации и типа эмулируемого устройства. В случае KVM VirtIO это /dev/vdb и /dev/vdc. Создадим на обоих разделах виртуальных машин sw2 и sw3 файловую систему XFS:

```
[root@sw2 и 3~]# mkfs.xfs /dev/vdb1
[root@sw2 и 3~]# mkfs.xfs /dev/vdc1
```

Файловые системы необходимо смонтировать в поддиректории /srv/node:

```
[root@sw2 и 3~]# mkdir -p /srv/node/vd{b,c}1
[root@sw2 и 3~]# echo "/dev/vdb1 /srv/node/vdb1 xfs noatime,nodi
ratime,nobarrier,logbufs=8 0 2" >> /etc/fstab
[root@sw2 и 3~]# echo "/dev/vdc1 /srv/node/vdc1 xfs noatime,nodi
ratime,nobarrier,logbufs=8 0 2" >> /etc/fstab
[root@sw2 и 3~]# mount -a
[root@sw2 и 3~]# chown -R swift:swift /srv/node/
```

Если вы не отключили SELinux, то восстановите контекст на директорию /srv/node:

```
[root@sw2 и 3 ~]# restorecon -vR /srv
```

Для того чтобы Swift выполнял репликацию, необходимо настроить демон rsyncd. Создаем на обоих узлах файл /etc/rsyncd.conf следующего содержания:


```
uid = swift
gid = swift
log file = /var/log/rsyncd.log
pid file = /var/run/rsyncd.pid
address = IP-адрес узла
[account]
max connections = 25
path = /srv/node/
read only = false
lock file = /var/lock/account.lock
[container]
max connections = 25
path = /srv/node/
read only = false
lock file = /var/lock/container.lock
[object]
max connections = 25
path = /srv/node/
read only = false
lock file = /var/lock/object.lock
```


Ознакомиться со значением параметров конфигурационного файла можно на man-странице rsyncd.conf (5). Запускаем и активируем сервис:

```
[root@sw2 и 3~]# systemctl enable rsyncd.service
[root@sw2 и 3~]# systemctl start rsyncd.service
```

Опять же, если вы оставили включенным SELinux, вам необходимо выполнить команду:

```
[root@sw2 и 3~]# setsebool -P rsync_full_access 1
```

Проверяем работу демона rsync для обоих серверов, подставив соответствующий IP-адрес:

```
[root@sw2 и 3~]# rsync IP-адрес::
[root@sw2 и 3~]# rsync IP-адрес::object
[root@sw2 и 3~]# rsync IP-адрес::account
[root@sw2 и 3~]# rsync IP-адрес::container
```

Теперь на каждом из двух узлов необходимо создать по три конфигурационных файла оставшихся сервисов. Для управления конфигурацией сервисов Swift используем систему Paste.deploy (http://pythonpaste.org/deploy/). Значения опций описаны на странице http://docs.openstack.org/developer/swift/deployment_guide.html. Конфигурационный файл /etc/swift/account-server.conf:

```
[DEFAULT]
bind_ip = IP-адрес сервера
bind_port = 6202
user = swift
workers = 2
swift_dir = /etc/swift
devices = /srv/node
[pipeline:main]
pipeline = healthcheck recon account-server
[app:account-server]
use = egg:swift#account
[filter:recon]
use = egg:swift#recon
recon_cache_path = /var/cache/swift

[filter:healthcheck]
use = egg:swift#healthcheck
[account-replicator]
[account-auditor]
[account-reaper]
```

Конфигурационный файл /etc/swift/object-server.conf:

```
[DEFAULT]
bind_ip = IP-адрес сервера
bind_port = 6200
user = swift
swift_dir = /etc/swift
```

```
devices = /srv/node
workers = 3
[pipeline:main]
pipeline = healthcheck recon object-server
[filter:recon]
use = egg:swift#recon
recon_cache_path = /var/cache/swift
[app:object-server]
use = egg:swift#object
[filter:healthcheck]
use = egg:swift#healthcheck
[object-replicator]
[object-updater]
[object-auditor]
```

Конфигурационный файл /etc/swift/container-server.conf:

```
[DEFAULT]
bind_ip = IP-адрес сервера
bind_port = 6201
user = swift
swift_dir = /etc/swift
devices = /srv/node
workers = 2
[pipeline:main]
pipeline = healthcheck recon container-server
[filter:recon]
use = egg:swift#recon
recon_cache_path = /var/cache/swift
[app:container-server]
use = egg:swift#container
[filter:healthcheck]
use = egg:swift#healthcheck
[container-replicator]
[container-updater]
[container-auditor]
[container-sync]
```

Обратите внимание, что, несмотря на то что в начале главы автор указал, что для служб Swift используются порты 600X, в конфигурационных файлах указаны 620X. Это связано с тем, что в дистрибутиве RDO рекомендованы и используются в компоненте openstack-puppet-modules именно эти порты. Разработчики обосновывают такой сдвиг конфликтом с X-сервером. С учетом того, что в тестовой среде действительно может одновременно работать графический сервер и Swift, это справедливо. Кроме того, именно порты 620X описаны по умолчанию в политике SELinux, что будет важно для вас, если вы не стали отключать эту подсистему.

Создание сервисных колец Swift

Следующим шагом настройки нам необходимо создать файлы сервисных колец (ring files). Эти файлы сопоставляют имена объектов их физическому местоположению. Создается по одному файлу для каждого объекта, контейнера и учетной записи. Каждый диск делится на разделы, каждый из которых представляет собой директорию. Рекомендуется от 100 разделов на диск. Для того чтобы определить, в каком разделе должен располагаться файл объекта, вычисляется хэш-функция по алгоритму MD5, и далее часть этого хэша используется как идентификатор раздела.

При создании файла кольца один из требуемых параметров – это число битов, используемых для определения раздела. Для определения числа битов проще всего прикинуть максимальное число дисков в кластере. Например, при 100 разделах на диске и 5000 дисков общее число разделов будет 5000 × 100 = 500 000. Для такого числа разделов надо определить ближайшую степень числа два, большую, чем 500 тысяч. Это будет 17, поскольку 2^17 = 524 288. Это число мы и будем использовать в качестве первого параметра утилиты `swift-ring-builder` при создании файлов колец. Вторым параметром задается число реплик файла (рекомендуется минимум 3), и, наконец, минимальное число часов, по прошествии которых раздел можно перемещать.

Проделываем шаги один раз на узле sw1. Пример показан для кольца account, и в качестве первого параметра мы возьмем число 10:

```
[root@sw1 ~]# cd /etc/swift/
[root@sw1 swift]# swift-ring-builder account.builder create 10 3 1
```

Теперь добавляем каждый из двух дисков на узлах sw2 и sw3, указывая их IP-адреса, выполнив команду `swift-ring-builder add` четыре раза.

```
[root@sw1 swift]# swift-ring-builder account.builder add --region 1
--zone 1 --ip 192.168.122.62 --port 6202 --device vdb1 --weight 100
Device d0r1z1-192.168.122.62:6202R192.168.122.62:6202/vdb1_"" with
100.0 weight got id 0
[root@sw1 swift]# swift-ring-builder account.builder add --region 1
--zone 1 --ip 192.168.122.62 --port 6202 --device vdc1 --weight 100
Device d1r1z1-192.168.122.62:6202R192.168.122.62:6202/vdc1_"" with
100.0 weight got id 1
[root@sw1 swift]# swift-ring-builder account.builder add --region 1
```

```
--zone 2 --ip 192.168.122.63 --port 6202 --device vdb1 --weight 100
Device d2r1z2-192.168.122.63:6202R192.168.122.63:6202/vdb1_""
with 100.0 weight got id 2
[root@sw1 swift]# swift-ring-builder account.builder add --region 1
--zone 2 --ip 192.168.122.63 --port 6202 --device vdc1 --weight 100
Device d3r1z2-192.168.122.63:6202R192.168.122.63:6202/vdc1_"" with
100.0 weight got id 3
```

Проверяем содержимое кольца:

```
[root@sw1 swift]# swift-ring-builder account.builder
account.builder, build version 5, id ff5952030739462a9bfcc860793825fa
1024 partitions, 3.000000 replicas, 1 regions, 2 zones, 4 devices, 0.00 balance, 0.00 dispersion
The minimum number of hours before a partition can be reassigned is 1 (0:59:46 remaining)
The overload factor is 0.00% (0.000000)
Ring file account.ring.gz is up-to-date
Devices:  id region zone      ip address:port replication ip:port  name weight partitions balance flags meta
          0      1    1 192.168.122.62:6202 192.168.122.62:6202  vdb1 100.00        768    0.00
          1      1    1 192.168.122.62:6202 192.168.122.62:6202  vdc1 100.00        768    0.00
          2      1    2 192.168.122.63:6202 192.168.122.63:6202  vdb1 100.00        768    0.00
          3      1    2 192.168.122.63:6202 192.168.122.63:6202  vdc1 100.00        768    0.00
```

Распределяем разделы по устройствам в кольце:

```
[root@sw1 swift]# swift-ring-builder account.builder rebalance
```

Затем подобные же команды повторяем для колец `container` и `object`. После этого копируем файлы колец на обе виртуальные машины sw2 и sw3:

```
[root@sw1 swift]# swift-ring-builder container.builder create 10 3 1
[root@sw1 swift]# swift-ring-builder container.builder add --region 1
--zone 1 --ip 192.168.122.62 --port 6201 --device vdb1 --weight 100
[root@sw1 swift]# swift-ring-builder container.builder add --region 1
--zone 1 --ip 192.168.122.62 --port 6201 --device vdc1 --weight 100
[root@sw1 swift]# swift-ring-builder container.builder add --region 1
--zone 2 --ip 192.168.122.63 --port 6201 --device vdb1 --weight 100
[root@sw1 swift]# swift-ring-builder container.builder add --region 1
--zone 2 --ip 192.168.122.63 --port 6201 --device vdc1 --weight 100
[root@sw1 swift]# swift-ring-builder container.builder rebalance

[root@sw1 swift]# swift-ring-builder object.builder create 10 3 1
[root@sw1 swift]# swift-ring-builder object.builder add --region 1
--zone 1 --ip 192.168.122.62 --port 6200 --device vdb1 --weight 100
[root@sw1 swift]# swift-ring-builder object.builder add --region 1
--zone 1 --ip 192.168.122.62 --port 6200 --device vdc1 --weight 100
[root@sw1 swift]# swift-ring-builder object.builder add --region 1
--zone 2 --ip 192.168.122.63 --port 6200 --device vdb1 --weight 100
[root@sw1 swift]# swift-ring-builder object.builder add --region 1
--zone 2 --ip 192.168.122.63 --port 6200 --device vdc1 --weight 100
```

```
[root@sw1 swift]# swift-ring-builder object.builder rebalance

[root@sw1 swift]# scp *.ring.gz sw2:/etc/swift/
[root@sw1 swift]# scp *.ring.gz sw3:/etc/swift/
```

Завершение настройки

В файл /etc/swift/swift.conf необходимо добавить два случайных параметра, которые следует держать в секрете. Они играют роль аналога «соли» файла /etc/shadow и будут добавляться к имени объекта для предотвращения DoS-атаки. Если злоумышленник будет знать значение этих параметров, то он сможет узнать реальное расположение объектов в разделах Swift.

```
[root@sw1 swift]# crudini --set /etc/swift/swift.conf swift-hash
swift_hash_path_suffix $(openssl rand -hex 10)
[root@sw1 swift]# crudini --set /etc/swift/swift.conf swift-hash
swift_hash_path_prefix $(openssl rand -hex 10)
```

Копия этого файла должна быть на каждом сервере:

```
[root@sw1 swift]# scp /etc/swift/swift.conf sw2:/etc/swift/swift.conf
[root@sw1 swift]# scp /etc/swift/swift.conf sw3:/etc/swift/swift.conf
```

Теперь убедимся, что все конфигурационные файлы на всех серверах принадлежат корректному пользователю и группе:

```
[root@sw1, 2 и 3 swift]#chown -R swift:swift /etc/swift
```

Наконец, командой `systemctl enable` и `systemctl start` активируем и запускаем на сервере sw1 сервисы `openstack-swift-proxy.service` и `memcached.service`, а на серверах sw2 и sw3 целых двенадцать сервисов:

```
openstack-swift-account.service
openstack-swift-account-auditor.service
openstack-swift-account-reaper.service
openstack-swift-account-replicator.service

openstack-swift-container.service
openstack-swift-container-auditor.service
openstack-swift-container-replicator.service
openstack-swift-container-updater.service

openstack-swift-object.service
openstack-swift-object-auditor.service
```

```
openstack-swift-object-replicator.service
openstack-swift-object-updater.service
```

Работа с сервисом Swift

Проверим функционирование сервиса, подключившись как пользователь demo:

```
$ source keystonerc_demo
$ swift stat
               Account: AUTH_bc10ac4b71164550a363b8098e8ad270
            Containers: 0
               Objects: 0
                 Bytes: 0
      X-Put-Timestamp: 1520072733.93450
          X-Timestamp: 1520072733.93450
           X-Trans-Id: tx9586054aea924ae2b2a38-005a9a781d
         Content-Type: text/plain; charset=utf-8
X-Openstack-Request-Id: tx9586054aea924ae2b2a38-005a9a781d
```

Теперь можно попробовать загрузить в сервис Swift файл. При этом автоматически создадим контейнер test-cont1:

```
$ swift upload test-cont1 /etc/hosts
/etc/hosts
```

Проверим список контейнеров и объектов:

```
$ swift list
test-cont1
$ swift list test-cont1
etc/hosts
```

Загрузим объект в текущую рабочую директорию:

```
$ swift download test-cont1
etc/hosts [auth 0.520s, headers 0.530s, total 0.531s, 0.055 MB/s]
$ cat etc/hosts
127.0.0.1   localhost localhost.localdomain localhost4
localhost4.localdomain4
...
```

Подключившись к серверам sw2 и sw3, поищем в директориях /srv/node файлы, заканчивающиеся на .data, – в них хранятся объекты. Поскольку у нас всего один объект и для каждого объекта должно быть три реплики, суммарно на обоих серверах должно найтись три файла:

```
[root@sw2 ~]# find /srv/node/ -type f -name *.data
```

```
/srv/node/vdb1/objects/814/f80/
cb8bc06cf9ef2cd33f3a335a472d2f80/1520073247.76181.data
/srv/node/vdc1/objects/814/f80/
cb8bc06cf9ef2cd33f3a335a472d2f80/1520073247.76181.data
[root@sw3 ~]# find /srv/node/ -type f -name *.data
/srv/node/vdc1/objects/814/f80/
cb8bc06cf9ef2cd33f3a335a472d2f80/1520073247.76181.data
```

Проверим по одному из имен найденных файлов, что это действительно файл hosts, который мы загрузили в сервис Swift:

```
[root@sw3 ~]# cat /srv/node/vdc1/objects/814/f80/
cb8bc06cf9ef2cd33f3a335a472d2f80/1520073247.76181.data
127.0.0.1   localhost localhost.localdomain localhost4
localhost4.localdomain4
...
```

В одиннадцатой главе мы рассмотрим использование веб-клиента Swift, а тут приведем пример снимка с экрана при работе с ним.

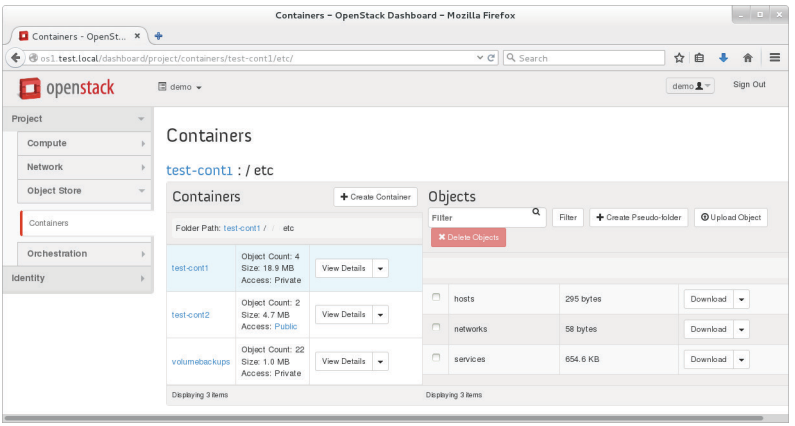


Рис. 6.2. Работа с Swift в интерфейсе веб-клиента

Для мониторинга кластера Swift можно использовать ряд утилит. В процессе установки мы настроили middleware-сервер swift-recon, который собирает и выдает в формате JSON различные метрики работы кластера. Также с ним можно использовать утилиту командной строки. Например, статистика по репликации:

```
[root@sw1 ~]# swift-recon container -r
===============================================================================
--> Starting reconnaissance on 2 hosts (container)
===============================================================================
```

```
[2018-03-03 10:38:20] Checking on replication
[replication_failure] low: 0, high: 0, avg: 0.0, total: 0, Failed: 0.0%, no_result: 0, reported: 2
[replication_success] low: 2, high: 4, avg: 3.0, total: 6, Failed: 0.0%, no_result: 0, reported: 2
[replication_time] low: 0, high: 0, avg: 0.0, total: 0, Failed: 0.0%, no_result: 0, reported: 2
[replication_attempted] low: 1, high: 2, avg: 1.5, total: 3, Failed: 0.0%, no_result: 0, reported: 2
Oldest completion was 2018-03-03 10:38:00 (19 seconds ago) by 192.168.122.63:6201.
Most recent completion was 2018-03-03 10:38:12 (7 seconds ago) by 192.168.122.62:6201.
========================================================================
```

Последнее, что мы сделаем, – это укажем корректную точку входа для сервиса Swift в настройках Cinder на сервере controller:

```
# crudini --set /etc/cinder/cinder.conf DEFAULT backup_swift_url
http://sw1.test.local:8080/v1/AUTH
```

Перезапустим сервис и убедимся, что теперь он работает:

```
[root@controller ~(Openstack_Admin)]# systemctl restart openstack-cinder-backup
[root@controller ~(Openstack_Admin)]# cinder service-list
+------------------+---------------------------+------+---------+-------+----------------------------+-----------------+
| Binary           | Host                      | Zone | Status  | State | Updated_at                 | Disabled Reason |
+------------------+---------------------------+------+---------+-------+----------------------------+-----------------+
| cinder-backup    | controller.test.local     | nova | enabled | up    | 2018-03-03T10:49:48.000000 | -               |
| cinder-scheduler | controller.test.local     | nova | enabled | up    | 2018-03-03T10:49:48.000000 | -               |
| cinder-volume    | controller.test.local@lvm | nova | enabled | up    | 2018-03-03T10:49:45.000000 | -               |
+------------------+---------------------------+------+---------+-------+----------------------------+-----------------+
```

Настройка Swift в качестве хранилища для Glance

Ну и в конце знакомства со Swift рассмотрим, как настроить объектное хранилище в качестве бэкэнда для сервиса хранения образов Glance. Предполагается, что мы работаем с конфигурационным файлом /etc/glance/glance-api.conf в том виде, в каком мы получили его к концу четвертой главы. Внесем только необходимые изменения, но перед этим сделайте резервную копию файла. Первым делом изменим хранилище по умолчанию:

```
[root@controller ~]# crudini --set /etc/glance/glance-api.conf
glance_store default_store swift
[root@controller ~]# crudini --set /etc/glance/glance-api.conf
glance_store stores swift
```

Для хранения настроек бэкэнда мы будем использовать отдельный файл etc/glance/glance-swift.conf и в нем будем ссылаться на конфигурацию ref1.

```
[root@controller ~]# crudini --set /etc/glance/glance-api.conf
glance_store default_swift_reference ref1
[root@controller ~]# crudini --set /etc/glance/glance-api.conf
glance_store swift_store_config_file /etc/glance/glance-swift.conf
```

Указываем на необходимость создания контейнера:

```
[root@controller ~]# crudini --set /etc/glance/glance-api.conf
glance_store swift_store_create_container_on_put True
```

Теперь добавляем параметры в файле /etc/glance/glance-swift.conf. Указываем опцию подключения к Keystone:

```
[root@controller ~]# crudini --set /etc/glance/glance-swift.conf
ref1 auth_address http://controller:35357/v3
[root@controller ~]# crudini --set /etc/glance/glance-swift.conf
ref1 user_domain_id default
[root@controller ~]# crudini --set /etc/glance/glance-swift.conf
ref1 project_domain_id default
[root@controller ~]# crudini --set /etc/glance/glance-swift.conf
ref1 auth_version 3
```

Указываем имя и пароль пользователя Swift:

```
[root@controller ~]# crudini --set /etc/glance/glance-swift.conf
ref1 user service:swift
[root@controller ~]# crudini --set /etc/glance/glance-swift.conf
ref1 key openstack
```

Теперь, чтобы glance мог создавать контейнеры, ему необходимо добавить роль ResellerAdmin в сервисном проекте. Если такой роли нет, то создаем ее:

```
$ source keystonerc_admin
$ openstack role create ResellerAdmin
$ openstack role add --project service --user glance ResellerAdmin
```

Рестартуем сервис Glance-api:

```
[root@controller ~]# systemctl restart openstack-glance-api
```

Наконец, можно проверить работу. Создадим новый образ, как мы это уже делали в главе, посвященной Glance:

```
$ openstack image create "cirros-0.4.0-x86_64-2" --file /tmp/cirros-0.4.0-
x86_64-disk.img --disk-format qcow2 --container-format bare --public
+------------------+------------------------------------------------------+
| Field            | Value                                                |
```

```
+------------------+--------------------------------------------------+
| checksum         | 443b7623e27ecf03dc9e01ee93f67afe                 |
| container_format | bare                                             |
| created_at       | 2018-03-04T12:12:07Z                             |
| disk_format      | qcow2                                            |
| file             | /v2/images/916374f7-be83-43a7-82ad-090f1d0aa130/file |
| id               | 916374f7-be83-43a7-82ad-090f1d0aa130             |
| min_disk         | 0                                                |
| min_ram          | 0                                                |
| name             | cirros-0.4.0-x86_64-2                            |
| owner            | 7fe2a6ef08df4a749f3bad1fceb055b9                 |
| protected        | False                                            |
| schema           | /v2/schemas/image                                |
| size             | 12716032                                         |
| status           | active                                           |
| tags             |                                                  |
| updated_at       | 2018-03-04T12:12:08Z                             |
| virtual_size     | None                                             |
| visibility       | public                                           |
+------------------+--------------------------------------------------+
```

Далее если вы хотите продолжить работу со стендом без сервиса Swift и сэкономить ресурсы, удалите только что созданный образ, восстановите файл /etc/glance/glance-api.conf из резервной копии и рестартуйте сервис openstack-glance-api.

Рекомендации по поиску неисправностей в сервисах Swift

Сервисы Swift пишут сообщения об ошибках в системный журнал /var/log/messages. Как правило, там могут быть trace-сообщения Python.

Проверьте, что все сервисы запущены, при помощи команды `lsof -i`, и что они работают на правильных портах. Если вы не нашли в списке одну из служб, то можете попытаться запустить ее вручную, указав исполняемому файлу ваш конфигурационный файл. Например:

```
[root@sw3 ~]# swift-object-server /etc/swift/object-server.conf
Traceback (most recent call last):
  File "/usr/bin/swift-object-server", line 27, in <module>
 ...
LookupError: No loader given in section 'filter:recon'
```

Сервис не запускается, и из вывода видно, что в данном случае имеется ошибка конфигурационного файла в секции recon.

В случае возникновения ошибок типа 503 Service Unavailable в первую очередь проверьте конфигурационные ошибки на опечатки, затем корректность имени и пароля в Keystone, а также корректность заведения URI точек входа в API сервиса.

В принципе, недоступность любого сервиса (Keystone, RabbitMQ или хранилищ Ceph либо GlusterFS) может вызвать ошибку 503 Service Unavailable. Проверьте соответствующие сервисы.

Ошибки в создании кольца нельзя исправить выполнением еще одной команды `swift-ring-builder add`. Предварительно необходимо отдать команду `swift-ring-builder remove`.

Глава 7

Контроллер и вычислительный узел Nova

```
Название: OpenStack Compute
Назначение: управление виртуальными машинами и сетью
Пакет:  openstack-nova
Имена сервисов: openstack-nova-api, openstack-nova-consoleauth,
openstack-nova-scheduler, openstack-nova-conductor.service,
openstack-nova-novncproxy.service, openstack-nova-compute.service
Порты: 8773, 8774, 8775, 8778, 5900-5999 (VNC)
Конфигурационные файлы: /etc/nova/nova.conf
Файлы журнала: /var/log/nova/nova-*
```

Переходим к самому главному компоненту OpenStack – сервису Nova, отвечающему за управление запущенными экземплярами виртуальных машин. Помимо управления виртуальными машинами, часть сервисов Nova также может обеспечивать управление сетью, но, хотя среди развернутых в настоящее время облаков процент использования сервиса nova-network достаточно высок, в этой книге мы не будем его рассматривать. В качестве сетевого компонента мы остановимся на более современном Neutron, который рассматривается в следующей главе. Данная глава в основном посвящена архитектуре и установке Nova. Работа с виртуальными машинами и сетями освещается дальше.

Архитектура Nova

Рассмотрим минимально необходимую часть сервисов Nova из тех, что мы будем использовать в нашем лабораторном окружении.

- **openstack-nova-api** – как и подобные службы других рассмотренных сервисов, отвечает за обработку пользовательских вызовов API.
- **openstack-nova-scheduler** – сервис-планировщик. Получает из очереди запросы на запуск виртуальных машин и выбирает узел для их запуска. Выбор осуществляется, согласно весам узлов после применения фильтров (например, необходимый объем оперативной памяти, определенная зона доступности и т. д.). Вес рассчитывается каждый раз при запуске или миграции виртуальной машины.
- **openstack-nova-conductor** – сервис появился в релизе Grizzly. Он выступает в качестве посредника между базой данных и nova-compute, позволяя осуществлять горизонтальное масштабирование. Этот сервис нельзя развертывать на тех же узлах, что и nova-compute.
- **openstack-nova-novncproxy** – выступает в роли VNC-прокси и позволяет подключаться к консоли виртуальных машин при помощи браузера.
- **openstack-nova-consoleauth** – отвечает за авторизацию для предыдущего сервиса.
- **openstack-nova-placement-api** – сервис появился в OpenStack версии Newton. Он отвечает за отслеживание списка ресурсов и их использование.
- **openstack-nova-compute** – демон, управляющий виртуальными машинами через API гипервизора. Как правило, запускается на узлах, где располагается сам гипервизор.

Также сервисам Nova требуются брокер сообщений и база данных. Мы, как всегда, будем использовать MariaDB и RabbitMQ, развернутые в виртуальной машине controller.test.local.

Первые пять сервисов мы разместим на нашей управляющей виртуальной машине, шестой – nova-compute – на новой, compute.test.local. При создании узла compute постарайтесь выделить ему как можно больше оперативной памяти.

Установка контроллера Nova

Начнем с установки пакетов на сервере controller.test.local:

```
[root@controller ~]# yum -y install openstack-nova-api openstack-
nova-conductor openstack-nova-novncproxy openstack-nova-scheduler
openstack-nova-console openstack-nova-placement-api
```

Далее проделаем уже традиционные операции. Создаем базы данных для сервиса, создаем пользователя, добавляем его с ролью admin в проект service, создаем сервис nova и с использованием идентификатора сервиса создаем точки входа в сервис:

```
# mysql -u root -p
Enter password: openstack
MariaDB [(none)]> CREATE DATABASE nova_api;
MariaDB [(none)]> CREATE DATABASE nova;
MariaDB [(none)]> CREATE DATABASE nova_cell0;

MariaDB [(none)]> GRANT ALL PRIVILEGES ON nova_api.* TO
'nova'@'localhost' IDENTIFIED BY 'nova';
MariaDB [(none)]> GRANT ALL PRIVILEGES ON nova_api.* TO
'nova'@'%' IDENTIFIED BY 'nova';
MariaDB [(none)]> GRANT ALL PRIVILEGES ON nova.* TO
'nova'@'localhost' IDENTIFIED BY 'nova';
MariaDB [(none)]> GRANT ALL PRIVILEGES ON nova.* TO 'nova'@'%'
IDENTIFIED BY 'nova';
MariaDB [(none)]> GRANT ALL PRIVILEGES ON nova_cell0.* TO
'nova'@'localhost' IDENTIFIED BY 'nova';
MariaDB [(none)]> GRANT ALL PRIVILEGES ON nova_cell0.* TO
'nova'@'%' IDENTIFIED BY 'nova';
MariaDB [(none)]> exit

$ source keystonerc_admin
$ openstack user create --domain default --password openstack nova
+---------------------+----------------------------------+
| Field               | Value                            |
+---------------------+----------------------------------+
| domain_id           | default                          |
| enabled             | True                             |
| id                  | 03b593ee9de442b985871aea39eec9b1 |
| name                | nova                             |
| options             | {}                               |
| password_expires_at | None                             |
+---------------------+----------------------------------+

$ openstack role add --project service --user nova admin
$ openstack service create --name nova --description "OpenStack
Compute Service" compute
+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description | OpenStack Compute Service        |
| enabled     | True                             |
| id          | 3345af31a7684e259ff49a17b05b9ab7 |
| name        | nova                             |
| type        | compute                          |
```

```
+-------------+----------------------------------+

$ openstack endpoint create --region RegionOne compute public
http://controller:8774/v2.1
+--------------+----------------------------------+
| Field        | Value                            |
+--------------+----------------------------------+
| enabled      | True                             |
| id           | f566c0fa6c16472ab4f08dabbfb9021a |
| interface    | public                           |
| region       | RegionOne                        |
| region_id    | RegionOne                        |
| service_id   | 3345af31a7684e259ff49a17b05b9ab7 |
| service_name | nova                             |
| service_type | compute                          |
| url          | http://controller:8774/v2.1      |
+--------------+----------------------------------+
$ openstack endpoint create --region RegionOne compute internal
http://controller:8774/v2.1
+--------------+----------------------------------+
| Field        | Value                            |
+--------------+----------------------------------+
| enabled      | True                             |
| id           | 80b44f94ed2047a2bbbfdbd4ff82befd |
| interface    | internal                         |
| region       | RegionOne                        |
| region_id    | RegionOne                        |
| service_id   | 3345af31a7684e259ff49a17b05b9ab7 |
| service_name | nova                             |
| service_type | compute                          |
| url          | http://controller:8774/v2.1      |
+--------------+----------------------------------+
$ openstack endpoint create --region RegionOne compute admin
http://controller:8774/v2.1
+--------------+----------------------------------+
| Field        | Value                            |
+--------------+----------------------------------+
| enabled      | True                             |
| id           | 52a1016b1e894755af86d5821a61f380 |
| interface    | admin                            |
| region       | RegionOne                        |
| region_id    | RegionOne                        |
| service_id   | 3345af31a7684e259ff49a17b05b9ab7 |
| service_name | nova                             |
| service_type | compute                          |
| url          | http://controller:8774/v2.1      |
+--------------+----------------------------------+
```

Теперь необходимо повторить действия по созданию пользователя и конечных точек для сервиса `placement`:

```
$ openstack user create --domain default --password openstack
placement
+---------------------+----------------------------------+
| Field               | Value                            |
+---------------------+----------------------------------+
| domain_id           | default                          |
| enabled             | True                             |
| id                  | 20ff08c488084425a1c15226d74d0bd6 |
| name                | placement                        |
| options             | {}                               |
| password_expires_at | None                             |
+---------------------+----------------------------------+
$ openstack role add --project service --user placement admin
$ openstack service create --name placement --description "OpenStack
Placement API" placement
+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description | OpenStack Placement API          |
| enabled     | True                             |
| id          | a8ee318c098745d1ae81cbb0a3f36f2b |
| name        | placement                        |
| type        | placement                        |
+-------------+----------------------------------+
$ openstack endpoint create --region RegionOne placement public
http://controller:8778
+--------------+----------------------------------+
| Field        | Value                            |
+--------------+----------------------------------+
| enabled      | True                             |
| id           | caf85b8863cb4ad2af39139cf42b9844 |
| interface    | public                           |
| region       | RegionOne                        |
| region_id    | RegionOne                        |
| service_id   | a8ee318c098745d1ae81cbb0a3f36f2b |
| service_name | placement                        |
| service_type | placement                        |
| url          | http://controller:8778           |
+--------------+----------------------------------+
$ openstack endpoint create --region RegionOne placement internal
http://controller:8778
+--------------+----------------------------------+
| Field        | Value                            |
+--------------+----------------------------------+
| enabled      | True                             |
| id           | 22ef59a2770e4485b2830069647e25cf |
| interface    | internal                         |
| region       | RegionOne                        |
| region_id    | RegionOne                        |
| service_id   | a8ee318c098745d1ae81cbb0a3f36f2b |
```

```
| service_name | placement                        |
| service_type | placement                        |
| url          | http://controller:8778           |
+--------------+----------------------------------+
$ openstack endpoint create --region RegionOne placement admin
http://controller:8778
+--------------+----------------------------------+
| Field        | Value                            |
+--------------+----------------------------------+
| enabled      | True                             |
| id           | f0fac7e439e64f00a2363bb6e6bcc1c3 |
| interface    | admin                            |
| region       | RegionOne                        |
| region_id    | RegionOne                        |
| service_id   | a8ee318c098745d1ae81cbb0a3f36f2b |
| service_name | placement                        |
| service_type | placement                        |
| url          | http://controller:8778           |
+--------------+----------------------------------+
```

Для начала установим опции в конфигурационном файле /etc/nova/nova.conf, общие для контроллера nova – виртуальной машины controller.test.local и вычислительного узла compute.test.local. Если у вас достаточно ресурсов, то вы также можете создать второй опциональный узел compute-opt.test.local.

Зададим настройки брокера сообщений RabbitMQ. Напомним, что для взаимодействия всех сервисов через брокер сообщений используется пользователь openstack с паролем openstack:

```
[root@controller ~]# crudini --set /etc/nova/nova.conf DEFAULT
transport_url rabbit://openstack:openstack@controller.test.local
```

Включаем два API: для самого сервиса и для обеспечения возможности передачи метаданных:

```
[root@controller ~]# crudini --set /etc/nova/nova.conf DEFAULT
enabled_apis osapi_compute,metadata
```

Настройки сервиса идентификации Keystone:

```
[root@controller ~]# crudini --set /etc/nova/nova.conf api auth_
strategy keystone
[root@controller ~]# crudini --set /etc/nova/nova.conf keystone_
authtoken project_name service
[root@controller ~]# crudini --set /etc/nova/nova.conf keystone_
authtoken user_domain_name default
[root@controller ~]# crudini --set /etc/nova/nova.conf keystone_
authtoken project_domain_name default
```

```
[root@controller ~]# crudini --set /etc/nova/nova.conf keystone_
authtoken auth_type password
[root@controller ~]# crudini --set /etc/nova/nova.conf keystone_
authtoken username nova
[root@controller ~]# crudini --set /etc/nova/nova.conf keystone_
authtoken password openstack
[root@controller ~]# crudini --set /etc/nova/nova.conf keystone_
authtoken auth_uri http://controller.test.local:5000
[root@controller ~]# crudini --set /etc/nova/nova.conf keystone_
authtoken auth_url http://controller.test.local:35357
```

Включаем поддержку сервиса Neutron:

```
[root@controller ~]# crudini --set /etc/nova/nova.conf DEFAULT
use_neutron True
[root@controller ~]# crudini --set /etc/nova/nova.conf DEFAULT
firewall_driver nova.virt.firewall.NoopFirewallDriver
```

и местоположение сервиса образов виртуальных машин Glance:

```
[root@controller ~]# crudini --set /etc/nova/nova.conf glance
api_servers http://controller.test.local:9292
```

Указываем имя региона для Cinder:

```
[root@controller ~]# crudini --set /etc/nova/nova.conf cinder
os_region_name RegionOne
```

Наконец, в секции `oslo_concurrency` укажем путь к файлам блокировок:

```
[root@controller ~]# crudini --set /etc/nova/nova.conf oslo_
concurrency lock_path /var/lib/nova/tmp
```

Настраиваем Placement API в секции placement:

```
[root@controller ~]# crudini --set /etc/nova/nova.conf placement
os_region_name RegionOne
[root@controller ~]# crudini --set /etc/nova/nova.conf placement
project_name service
[root@controller ~]# crudini --set /etc/nova/nova.conf placement
user_domain_name Default
[root@controller ~]# crudini --set /etc/nova/nova.conf placement
project_domain_name Default
[root@controller ~]# crudini --set /etc/nova/nova.conf placement
auth_type password
[root@controller ~]# crudini --set /etc/nova/nova.conf placement
username placement
[root@controller ~]# crudini --set /etc/nova/nova.conf placement
password openstack
```

```
[root@controller ~]# crudini --set /etc/nova/nova.conf placement
auth_url http://controller.test.local:35357/v3
```

Описанные выше опции также необходимо будет прописать и в виртуальной машине вычислительного узла compute. На данном этапе вы просто можете перенести конфигурационный файл, скопировав его на второй и опциональный третьий узел при его наличии:

```
[root@controller ~]# scp /etc/nova/nova.conf compute:~/nova.conf
[root@controller ~]# scp /etc/nova/nova.conf compute-opt:~/nova.conf
```

Продолжим настраивать опции, специфичные для контроллера. Пути к базам данных:

```
[root@controller ~]# crudini --set /etc/nova/nova.conf api_
database connection  mysql+pymysql://nova:nova@controller.test.
local/nova_api
[root@controller ~]# crudini --set /etc/nova/nova.conf database
connection mysql+pymysql://nova:nova@controller.test.local/nova
```

IP-адрес управляющего интерфейса:

```
[root@controller ~]# crudini --set /etc/nova/nova.conf DEFAULT
my_ip 192.168.122.200
```

Тот же самый адрес, где будет слушать входящие подключения VNC-сервер:

```
[root@controller ~]# crudini --set /etc/nova/nova.conf vnc
server_listen 192.168.122.200
[root@controller ~]# crudini --set /etc/nova/nova.conf vnc
server_proxyclient_address 192.168.122.200
[root@controller ~]# crudini --set /etc/nova/nova.conf vnc
enabled true
```

На момент написания книги существовала проблема, описанная в https://bugzilla.redhat.com/show_bug.cgi?id=1430540. Для исправления ошибки необходимо добавить в файл /etc/httpd/conf.d/00-nova-placement-api.conf секцию:

```
<Directory /usr/bin>
   <IfVersion >= 2.4>
      Require all granted
   </IfVersion>
   <IfVersion < 2.4>
      Order allow,deny
      Allow from all
```

```
   </IfVersion>
</Directory>
```

И перезапустить веб-сервер:

```
# systemctl restart httpd
```

Инициализируем базу данных nova-api:

```
[root@controller ~]#  su -s /bin/sh -c "nova-manage api_db sync" nova
```

Можно игнорировать предупреждающее сообщение в выводе команды. Далее регистрируем базу данных cell0 и создаем ячейку cell1:

```
[root@controller ~]# su -s /bin/sh -c "nova-manage cell_v2 map_cell0" nova
[root@controller ~]# su -s /bin/sh -c "nova-manage cell_v2 create_cell
--name=cell1 --verbose" nova
eb2201b9-0776-4d13-8aa8-93cba84f8894
```

Заполняем базу данных. Это может занять некоторое время:

```
[root@controller ~]# su -s /bin/sh -c "nova-manage db sync" nova
```

Активируем и запускаем сервисы:

```
[root@controller ~]# systemctl enable openstack-nova-api.service
openstack-nova-consoleauth.service openstack-nova-scheduler.
service openstack-nova-conductor.service openstack-nova-
novncproxy.service
[root@controller ~]# systemctl start openstack-nova-api.service
openstack-nova-consoleauth.service openstack-nova-scheduler.
service openstack-nova-conductor.service openstack-nova-
novncproxy.service
```

Можно проверить, что сервисы стартовали, и просмотреть список файлов журналов и самих сервисов, которые используют журналы:

```
[root@controller ~]# systemctl status openstack-nova-api.service  openstack-nova-consoleauth.
service openstack-nova-scheduler.service openstack-nova-conductor.service openstack-nova-
novncproxy.service -n 0
● openstack-nova-api.service - OpenStack Nova API Server
   Loaded: loaded (/usr/lib/systemd/system/openstack-nova-api.service; enabled; vendor preset:
disabled)
   Active: active (running) since Sun 2018-03-04 16:05:58 CET; 5s ago
 Main PID: 7281 (nova-api)
   CGroup: /system.slice/openstack-nova-api.service
           ├─7281 /usr/bin/python2 /usr/bin/nova-api
```

```
           ├─7336 /usr/bin/python2 /usr/bin/nova-api
           └─7338 /usr/bin/python2 /usr/bin/nova-api

● openstack-nova-consoleauth.service - OpenStack Nova VNC console auth Server
   Loaded: loaded (/usr/lib/systemd/system/openstack-nova-consoleauth.service; enabled; vendor
preset: disabled)
   Active: active (running) since Sun 2018-03-04 16:05:54 CET; 9s ago
 Main PID: 7282 (nova-consoleaut)
   CGroup: /system.slice/openstack-nova-consoleauth.service
           └─7282 /usr/bin/python2 /usr/bin/nova-consoleauth

● openstack-nova-scheduler.service - OpenStack Nova Scheduler Server
   Loaded: loaded (/usr/lib/systemd/system/openstack-nova-scheduler.service; enabled; vendor
preset: disabled)
   Active: active (running) since Sun 2018-03-04 16:05:55 CET; 8s ago
 Main PID: 7283 (nova-scheduler)
   CGroup: /system.slice/openstack-nova-scheduler.service
           └─7283 /usr/bin/python2 /usr/bin/nova-scheduler

● openstack-nova-conductor.service - OpenStack Nova Conductor Server
   Loaded: loaded (/usr/lib/systemd/system/openstack-nova-conductor.service; enabled; vendor
preset: disabled)
   Active: active (running) since Sun 2018-03-04 16:05:54 CET; 9s ago
 Main PID: 7284 (nova-conductor)
   CGroup: /system.slice/openstack-nova-conductor.service
           └─7284 /usr/bin/python2 /usr/bin/nova-conductor

● openstack-nova-novncproxy.service - OpenStack Nova NoVNC Proxy Server
   Loaded: loaded (/usr/lib/systemd/system/openstack-nova-novncproxy.service; enabled; vendor
preset: disabled)
   Active: active (running) since Sun 2018-03-04 16:05:37 CET; 26s ago
 Main PID: 7285 (nova-novncproxy)
   CGroup: /system.slice/openstack-nova-novncproxy.service
           └─7285 /usr/bin/python2 /usr/bin/nova-novncproxy --web /usr/share/novnc/

[root@controller ~]# lsof /var/log/nova/*
COMMAND    PID   USER   FD   TYPE DEVICE SIZE/OFF     NODE NAME
httpd     6709   root   14w  REG  253,0        0 67176374 /var/log/nova/nova-placement-api.log
...
nova-api  7281   nova    3w   REG  253,0     4829 68437359 /var/log/nova/nova-api.log
nova-cons 7282   nova    3w   REG  253,0     2245 68437334 /var/log/nova/nova-consoleauth.log
nova-sche 7283   nova    3w   REG  253,0     2389 68437335 /var/log/nova/nova-scheduler.log
nova-cond 7284   nova    3w   REG  253,0     2243 67174004 /var/log/nova/nova-conductor.log
nova-novn 7285   nova    3w   REG  253,0      632 68437369 /var/log/nova/nova-novncproxy.log
nova-api  7336   nova    3w   REG  253,0     4829 68437359 /var/log/nova/nova-api.log
nova-api  7338   nova    3w   REG  253,0     4829 68437359 /var/log/nova/nova-api.log
```

На этом месте переходим к настройкам вычислительного узла, где будут запускаться виртуальные машины нашего облака.

Установка вычислительных узлов Nova

Устанавливаем необходимые пакеты на новую виртуальную машину compute.test.local, подготовленную в соответствии с описанием во второй главе:

```
[root@compute ~]# yum -y install openstack-nova-compute sysfsutils libvirt
```

Если на вашем стенде достаточно ресурсов, то проделайте то же самое для второго вычислительного узла compute-opt.test.local.

Предполагаем, что у нас уже есть конфигурационный файл, перенесенный с узла controller, как это было описано в предыдущем разделе. Соответственно, остается его скопировать в /etc/nova/ и добавить настройки, уникальные для вычислительных узлов.

Первое, с чем нужно определиться, – это поддерживает ли ваша виртуальная машина или сервер аппаратную виртуализацию. Определить поможет команда:

```
[root@compute ~]# grep -E ' svm | vmx' /proc/cpuinfo
```

Если поиск grep ничего не показывает, то вам придется воспользоваться эмуляцией QEMU:

```
[root@compute ~]# crudini --set /etc/nova/nova.conf libvirt virt_type qemu
```

Однако большинство современных систем виртуализации поддерживает вложенную виртуализацию, так что автор рекомендует поискать соответствующие инструкции для вашей системы. По умолчанию OpenStack использует kvm, так что если аппаратная поддержка присутствует, то дополнительно менять настройку не нужно.

Нужно отметить, что на установленном OpenStack веб-интерфейс и утилиты командной строки будут в обоих случаях показывать в качестве типа гипервизора QEMU. Определить, используется эмуляция или виртуализация, можно опять же при помощи флагов процессора:

```
$ nova hypervisor-show compute.test.local | grep vmx
|                           | "fsgsbase", "xsave", "pge", "vmx",       |
$ nova hypervisor-show compute-opt.test.local | grep vmx
```

В данном примере на узле compute.test.local параметр `virt_type` установлен в `kvm`, а на узле compute-opt.test.local – в qemu.

Указываем адреса вычислительных узлов:

```
[root@compute ~]# crudini --set /etc/nova/nova.conf DEFAULT my_ip
192.168.122.210
```

Для compute-opt укажите адрес 192.168.122.215. Затем укажем, что VNC-сервер будет слушать подключения на всех интерфейсах и URL прокси-сервера, где будет доступен браузеру интерфейс виртуальной машины:

```
[root@compute ~]# crudini --set /etc/nova/nova.conf vnc vnc_enabled True
[root@compute ~]# crudini --set /etc/nova/nova.conf vnc vncserver_listen
0.0.0.0
[root@compute ~]# crudini --set /etc/nova/nova.conf vnc novncproxy_base_
url http://controller.test.local:6080/vnc_auto.html
```

Следующая команда должна содержать адрес конкретного вычислительного узла. Соответственно, для compute-opt укажите адрес 192.168.122.215:

```
[root@compute ~]# crudini --set /etc/nova/nova.conf vnc
vncserver_proxyclient_address 192.168.122.210
```

Активируем и запускаем сервисы libvirt и nova-compute:

```
[root@compute ~]# systemctl enable libvirtd.service openstack-
nova-compute.service
[root@compute ~]# systemctl start libvirtd.service openstack-
nova-compute.service
```

Теперь проверим работу сервисов при помощи команды `openstack compute service list` или `nova service-list`. Они выдают одинаковый результат, и мы должны получить такую картину:

```
$ source keystonerc_admin
$ nova service-list
+------+------------------+-------------------------+----------+---------+-------+------------+-----------------+------------+
| Id   | Binary           | Host                    | Zone     | Status  | State | Updated_at | Disabled Reason | Forced down |
+------+------------------+-------------------------+----------+---------+-------+------------+-----------------+------------+
| a5.. | nova-consoleauth | controller.test.local   | internal | enabled | up    | 2018-03-04.. | -               | False      |
| 09.. | nova-conductor   | controller.test.local   | internal | enabled | up    | 2018-03-04.. | -               | False      |
| 0e.. | nova-scheduler   | controller.test.local   | internal | enabled | up    | 2018-03-04.. | -               | False      |
| 2d.. | nova-compute     | compute.test.local      | nova     | enabled | up    | 2018-03-04.. | -               | False      |
| 42.. | nova-compute     | compute-opt.test.local  | nova     | enabled | up    | 2018-03-04.. | -               | False      |
+------+------------------+-------------------------+----------+---------+-------+------------+-----------------+------------+
```

Теперь регистрируем гипервизоры:

```
$ su -s /bin/sh -c "nova-manage cell_v2 discover_hosts --verbose" nova
Found 2 cell mappings.
```

```
Skipping cell0 since it does not contain hosts.
Getting compute nodes from cell 'cell1': eb2201b9-0776-4d13-8aa8-
93cba84f8894
Found 2 unmapped computes in cell: eb2201b9-0776-4d13-8aa8-
93cba84f8894
Checking host mapping for compute host 'compute.test.local':
658bd88c-8878-489c-9a9a-1239295ba61c
Creating host mapping for compute host 'compute.test.local':
658bd88c-8878-489c-9a9a-1239295ba61c
Checking host mapping for compute host 'compute-opt.test.local':
6989b8db-72db-47b7-864d-b3e1fd7b7868
Creating host mapping for compute host 'compute-opt.test.local':
6989b8db-72db-47b7-864d-b3e1fd7b7868
```

Продолжим проверку работы сервиса. Интеграция с Glance:

```
$ openstack image list
+--------------------------------------+---------------------+--------+
| ID                                   | Name                | Status |
+--------------------------------------+---------------------+--------+
| c8ccc9b3-29bb-4220-be38-8f261ac8b99a | cirros-0.4.0-x86_64 | active |
+--------------------------------------+---------------------+--------+
```

Проверяем работу Placement API:

```
# su -s /bin/sh -c "nova-status upgrade check" nova
+---------------------------+
| Upgrade Check Results     |
+---------------------------+
| Check: Cells v2           |
| Result: Success           |
| Details: None             |
+---------------------------+
| Check: Placement API      |
| Result: Success           |
| Details: None             |
+---------------------------+
| Check: Resource Providers |
| Result: Success           |
| Details: None             |
+---------------------------+
```

На будущее также нам будет полезна команда `host-describe`, позволяющая посмотреть информацию об использовании ресурсов вычислительного узла в целом и по отдельным проектам:

```
$ nova --os-compute-api-version 2.1 host-describe compute.test.local
+--------------------+------------+-----+-----------+---------+
| HOST               | PROJECT    | cpu | memory_mb | disk_gb |
+--------------------+------------+-----+-----------+---------+
```

```
| compute.test.local | (total)     | 4   | 3952       | 49      |
| compute.test.local | (used_now)  | 1   | 812        | 1       |
| compute.test.local | (used_max)  | 1   | 300        | 1       |
+--------------------+-------------+-----+------------+---------+
```

Еще одна команда, которая покажет сервисы и их распределение по узлам:

```
$ nova --os-compute-api-version 2 host-list
+------------------------+-------------+----------+
| host_name              | service     | zone     |
+------------------------+-------------+----------+
| controller.test.local  | consoleauth | internal |
| controller.test.local  | conductor   | internal |
| controller.test.local  | scheduler   | internal |
| compute.test.local     | compute     | nova     |
| compute-opt.test.local | compute     | nova     |
+------------------------+-------------+----------+
```

Наконец, при помощи команды `nova hypervisor-show` можно получить подробную информацию по гипервизору:

```
$ nova --os-compute-api-version 2 hypervisor-show compute-opt.
test.local
+--------------------------+----------------------------------------+
| Property                 | Value                                  |
+--------------------------+----------------------------------------+
| cpu_info_arch            | x86_64                                 |
| cpu_info_features        | ["pge", "avx", "xsaveopt", "clflush",  |
|                          | "sep", "syscall", "tsc_adjust", "tsc-  |
|                          | deadline", "invpcid", "tsc", "fsgsbase", |
|                          | "xsave", "smap", "vmx", "erms", "cmov", |
|                          | "smep", "fpu", "pat", "lm", "msr",     |
|                          | "adx", "3dnowprefetch", "nx", "fxsr",  |
|                          | "sse4.1", "pae", "sse4.2", "pclmuldq", |
|                          | "pcid", "fma", "vme", "mmx", "osxsave", |
|                          | "cx8", "mce", "de", "aes", "mca", "pse", |
|                          | "lahf_lm", "abm", "rdseed", "popcnt",  |
|                          | "pdpe1gb", "apic", "sse", "f16c", "pni", |
|                          | "rdtscp", "avx2", "sse2", "ss",        |
|                          | "hypervisor", "bmi1", "bmi2", "ssse3", |
|                          | "cx16", "pse36", "mtrr", "movbe",      |
|                          | "rdrand", "x2apic"]                    |
| cpu_info_model           | Broadwell-noTSX                        |
| cpu_info_topology_cells  | 1                                      |
| cpu_info_topology_cores  | 1                                      |
| cpu_info_topology_sockets | 4                                     |
| cpu_info_topology_threads | 1                                     |
| cpu_info_vendor          | Intel                                  |
| current_workload         | 0                                      |
```

```
| disk_available_least     | 48                                    |
| free_disk_gb             | 49                                    |
| free_ram_mb              | 1535                                  |
| host_ip                  | 192.168.122.215                       |
| hypervisor_hostname      | compute-opt.test.local                |
| hypervisor_type          | QEMU                                  |
| hypervisor_version       | 2009000                               |
| id                       | 2                                     |
| local_gb                 | 49                                    |
| local_gb_used            | 0                                     |
| memory_mb                | 2047                                  |
| memory_mb_used           | 512                                   |
| running_vms              | 0                                     |
| service_disabled_reason  | None                                  |
| service_host             | compute-opt.test.local                |
| service_id               | 7                                     |
| state                    | up                                    |
| status                   | enabled                               |
| vcpus                    | 4                                     |
| vcpus_used               | 0                                     |
+--------------------------+---------------------------------------+
```

Осталось добавить сеть, чем мы займемся в следующей главе, и основные компоненты нашего облака – в сборе.

Глава 8

Службы сети Neutron

```
Название:  OpenStack Networking
Назначение: обеспечение сети для виртуальных машин
Пакет:  openstack-neutron
Имена сервисов: openstack-neutron-*
Порт: 9696/tcp
Конфигурационные файлы: /etc/neutron/*
Файлы журнала: /var/log/neutron/*
```

OpenStack Networking (Neutron) – это сетевой сервис облачной операционной системы, использующий множество технологий, включая стандартные технологии коммутации (NetFlow, RSPAN, SPAN, LACP, 802.1q, туннелирование GRE и VXLAN), балансировку нагрузки, брандмауэр, VPN и др. Часть функционала реализуется через подключаемые модули сторонних производителей, часть зависит от компонентов операционной системы GNU/Linux, таких как iptables и Open vSwitch.

Архитектура Neutron

Основные абстракции, которыми оперирует Neutron:

- сеть – содержит в себе подсети. Различают внутренние, виртуальные сети, которых может быть много, и как минимум одну внешнюю. Доступ к виртуальным машинам внутри внутренней сети могут получить только машины в этой же сети или узлы, связанные через виртуальные маршрутизаторы. Внешняя сеть представляет собой отображение части реально существующей физической сети и необходима для обеспечения сетевой связанности ваших виртуальных машин внутри облака, использующих внутренние сети, и «внешнего мира». Внешние сети создаются администратором OpenStack;

- подсеть – сеть должна иметь ассоциированные с ней подсети. Именно через подсеть задается конкретный диапазон IP-адресов;
- маршрутизатор – как и в физическом мире, служит для маршрутизации между сетями. Маршрутизатор может иметь шлюз и множество интерфейсов, соединяющих подсети;
- группа безопасности – набор правил брандмауэра, применяющихся к виртуальным машинам в этой группе;
- «плавающий IP-адрес» (Floating IP) – IP-адрес внешней сети, назначаемый экземпляру виртуальной машины. Он может быть выделен только из существующей внешней сети. По умолчанию каждый проект имеет квоту в 50 адресов;
- порт – подключение к подсети. Порт на виртуальном коммутаторе. Включает в себя MAC-адрес и IP-адрес.

Справочная архитектура на сайте OpenStack выделяет четыре типа трафика (внешний, внутренний, трафик виртуальных машин, API и управления). Внутренние сети могут быть как «плоскими», так и тегированными. В случае если планируется более четырех тысяч внутренних сетей (лимит VLAN), создание оверлейных сетей может осуществляться при помощи VXLAN или GRE-инкапсуляции.

В реальной жизни может использоваться большее число изолированных типов трафика:

- публичная сеть, которой принадлежат «реальные» IP-адреса виртуальных машин;
- сеть, предоставляющая наружу публичный API и веб-интерфейс Horizon;
- управляющая сеть для операционных систем и служб (ssh и мониторинг);
- управляющая сеть для сетевой установки узлов под сервисы OpenStack;
- управляющая сеть для IPMI, DRAC, iLO, консолей коммутаторов и т. п.;
- сеть передачи данных для служб SDS (Swift, Ceph, iSCSI, NFS…);
- демилитаризованная зона.

Справочная архитектура на сайте документации OpenStack определяет три типа узлов: узел управления, сетевой узел и вычислительный узел. Как правило, сетевые узлы в реальной экс-

плуатации – физические серверы, поскольку они требовательны к ресурсам. В некоторых случаях, если реальная обработка трафика выполняется на физических коммутаторах, сетевые узлы не требуются. В таких случаях управляющий сервер Neutron через подключаемые модули управляет таким физическим «железом». В нашем лабораторном окружении роль сетевого узла будет выполнять network.test.local.

В устаревшей реализации сети nova-network использовались стандартные компоненты операционной системы GNU/Linux, такие как брандмауэр iptables, сетевые мосты и VLAN. В Neutron для реализации взаимодействия на втором уровне сетевого стека OSI появилась возможность расширять API при помощи подключаемых модулей (plug-in). В других источниках при описании использования Neutron до релиза Havana вы можете встретить модули LinuxBridge и Open vSwitch. Оба модуля монолитные и не подразумевают одновременного использования.

Начиная с релиза Icehouse они считаются устаревшими, поэтому их заменил модуль Modular Layer 2 (ML2 – OVS/LB), который и рассматривается в этой книге. Его преимуществом является возможность одновременного использования нескольких технологий второго уровня. При этом устаревшие подключаемые модули для упрощения поддержки, тестирования и переиспользования кода трансформировались в драйверы механизмов/агенты (mechanism_drivers в конфигурационном файле ml2_conf.ini).

Существует несколько типов драйверов механизмов:

- использующие агенты, например LinuxBridge и OVS, который мы рассматриваем в книге;
- использующие контроллеры SDN, например OpenDaylight, OpenContrail, VMware NSX, PLUMgrid и др.;
- использующие аппаратные коммутаторы, например Cisco Nexus, Extreme Networks и многие другие.

Если говорить о сторонних подключаемых модулях, то их около двух десятков. За подробностями автор рекомендует обратиться в wiki – https://wiki.openstack.org/wiki/Neutron_Plugins_and_Drivers.

Также существуют сервисные подключаемые модули Neutron:

- маршрутизатор;
- балансировщик нагрузки (LBaaS);
- брандмауэр (FaaS);
- виртуальные частные сети (VPNaaS).

Один узел может обслуживать несколько проектов с отдельными сетями, адресные пространства которых могут совпадать. Пересечение сетевых пространств реализуется при помощи пространств имен (namespaces). Сетевые пространства имен позволяют на одной машине иметь несколько ARP и таблиц маршрутизации, наборов правил брандмауэра, сетевых устройств и т. д.

Перечислим основные сервисы.

- Сервис узла контроллера:
  - **neutron-server** – центральный управляющий компонент. Не занимается непосредственно маршрутизацией пакетов. С остальными компонентами взаимодействует через брокер сообщений.
- Сервисы сетевого узла:
  - **neutron-openvswitch-agent** – взаимодействует с neutron-server через брокер сообщений и отдает команды OVS для построения таблицы потоков. При этом используются локальные команды OVS и протокол OpenFlow не используется;
  - **neutron-l3-agent** – обеспечивает маршрутизацию и NAT, используя технологию сетевых пространств имен;
  - **openvswitch** – программный коммутатор, используемый для построения сетей. Не является проектом OpenStack, но используется им. Подробнее об Open vSwitch – в следующей главе;
  - **neutron-dhcp-agent** – сервис отвечает за запуск и управление процессами dnsmasq. Dnsmasq – это легковесный dhcp-сервер и сервис кэширования DNS. Также neutron-dhcp-agent отвечает за запуск прокси-процессов сервера предоставления метаданных. По умолчанию каждая сеть, создаваемая агентом, получает собственное пространство имен qdhcp-UUID_сети;
  - **neutron-metadata-agent** – данный сервис позволяет виртуальным машинам запрашивать данные о себе, такие как имя узла, открытый ssh-ключ для аутентификации и др. Экземпляры виртуальных машин получают эту информацию во время загрузки скриптом, подобным cloud-init (https://launchpad.net/cloud-init), обращаясь на адрес http://169.254.169.254. Агент проксирует соответствующие запросы к openstack-nova-api при по-

мощи пространства имен маршрутизатора или DHCP. Во время настройки стенда мы для Nova и Neutron будем задавать общий секрет, используемый для подписи сообщений;

- **neutron-ovs-cleanup** – отвечает во время старта за удаление из базы данных OVS неиспользуемых мостов «старых» виртуальных машин.

❍ Сервисы вычислительного узла включают в себя уже ранее перечисленные:

- **openvswitch**;
- **neutron-openvswitch-agent**.

Работа Neutron при создании экземпляра виртуальной машины

Рассмотрим последовательность действий сервиса Neutron при создании виртуальной машины.

Во время создания экземпляра виртуальной машины сервис Nova отправляет запрос на сервис neutron-server, отвечающий за API. Тот, в свою очередь, отправляет запрос агенту DHCP на создание IP-адреса. Он обращается к сервису dnsmasq, отвечающему за подсеть, в которой создается виртуальная машина. Dnsmasq возвращает первый свободный IP-адрес из диапазона адресов подсети, после чего агент DHCP отправляет этот адрес сервису neutron-server.

После того как за виртуальной машиной закреплен IP-адрес, сервис Neutron отправляет запрос на Open vSwitch для создания конфигурации, включающей IP-адрес в существующую сеть. Требуемые параметры конфигурации возвращаются обратно на сервис Neutron при помощи шины сообщений, а далее Neutron отправляет их сервису Nova.

Установка узла управления Neutron

Начнем установку с управляющего узла, в роли которого выступает виртуальная машина controller.test.local. Ставим компоненты Neutron и подключаемый модуль Modular Layer 2:

```
[root@controller ~]# yum -y install openstack-neutron openstack-
neutron-ml2
```

Как всегда, создадим экземпляр базы данных MariaDB:

```
[root@controller ~]# mysql -u root -p
MariaDB [(none)]> CREATE DATABASE neutron;
MariaDB [(none)]> GRANT ALL PRIVILEGES ON neutron.* TO
'neutron'@'localhost' IDENTIFIED BY 'neutron';
MariaDB [(none)]> GRANT ALL PRIVILEGES ON neutron.* TO
'neutron'@'%' IDENTIFIED BY 'neutron';
```

Далее создаем пользователя, сервис и точки входа:

```
$ source keystonerc_admin
$ openstack user create --domain default --password openstack neutron
+---------------------+----------------------------------+
| Field               | Value                            |
+---------------------+----------------------------------+
| domain_id           | default                          |
| enabled             | True                             |
| id                  | bc4ad74c8d9345a2a6cbeadae067a210 |
| name                | neutron                          |
| options             | {}                               |
| password_expires_at | None                             |
+---------------------+----------------------------------+
$ openstack role add --project service --user neutron admin
$ openstack service create --name neutron --description "OpenStack
Networking" network
+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description | OpenStack Networking             |
| enabled     | True                             |
| id          | ebb95159786f48668c5fa50a10129a89 |
| name        | neutron                          |
| type        | network                          |
+-------------+----------------------------------+
$ openstack endpoint create --region RegionOne network public
http://controller.test.local:9696
+--------------+----------------------------------+
| Field        | Value                            |
+--------------+----------------------------------+
| enabled      | True                             |
| id           | c33356a2d4b1467d968eab7524457a7cc |
| interface    | public                           |
| region       | RegionOne                        |
| region_id    | RegionOne                        |
| service_id   | ebb95159786f48668c5fa50a10129a89 |
| service_name | neutron                          |
| service_type | network                          |
| url          | http://controller.test.local:9696 |
+--------------+----------------------------------+
```

```
$ openstack endpoint create --region RegionOne network internal
http://controller.test.local:9696
+--------------+----------------------------------+
| Field        | Value                            |
+--------------+----------------------------------+
| enabled      | True                             |
| id           | 9263f2263399444a97651e20a2650e4f |
| interface    | internal                         |
| region       | RegionOne                        |
| region_id    | RegionOne                        |
| service_id   | ebb95159786f48668c5fa50a10129a89 |
| service_name | neutron                          |
| service_type | network                          |
| url          | http://controller.test.local:9696 |
+--------------+----------------------------------+
$ openstack endpoint create --region RegionOne network admin
http://controller.test.local:9696
+--------------+----------------------------------+
| Field        | Value                            |
+--------------+----------------------------------+
| enabled      | True                             |
| id           | bd3e74e14e9b4086856f746b4d43767f |
| interface    | admin                            |
| region       | RegionOne                        |
| region_id    | RegionOne                        |
| service_id   | ebb95159786f48668c5fa50a10129a89 |
| service_name | neutron                          |
| service_type | network                          |
| url          | http://controller.test.local:9696 |
+--------------+----------------------------------+
```

Следующие настройки конфигурационных файлов необходимо выполнить на управляющем, сетевом и вычислительном узлах. Соответственно, файлы neutron.conf и ml2_conf.ini можно будет скопировать на узлы network, compute и compute-opt. Для удобства читателя в табл. 3 приведены используемые конфигурационные файлы Neutron и описано их назначение.

Таблица 3. Конфигурационные файлы Neutron

Конфигурационный файл	Назначение конфигурационного файла
/etc/neutron/neutron.conf	Основной конфигурационный файл Neutron
/etc/neutron/plugins/ml2/ml2_conf.ini	Конфигурационный файл подключаемого модуля Modular Layer 2 (ML2)

Конфигурационный файл	Назначение конфигурационного файла
/etc/neutron/plugin.ini	Символическая ссылка на конфигурационный файл подключаемого модуля ML2 ml2_conf.ini или в старых версиях на конфигурационный файл монолитного подключаемого модуля Open vSwitch – ovs_neutron_plugin.ini
/etc/neutron/l3_agent.ini	Конфигурационный файл L3-агента, осуществляющего маршрутизацию
/etc/neutron/dhcp_agent.ini	Конфигурационный файл DHCP-агента
/etc/neutron/metadata_agent.ini	Конфигурационный файл агента, предоставляющего метаданные виртуальным машинам
/etc/neutron/plugins/ml2/openvswitch_agent.ini	Подключаемый модуль агента Open vSwitch для модуля ML2. До Liberty назывался ovs_neutron_plugin.ini. В прошлом в файле ovs_neutron_plugin.ini располагались настройки монолитного подключаемого модуля Open vSwitch. В ранней документации на Juno и в первом издании книги предлагалось использовать ml2_conf.ini для настройки агента. Ранее некоторые дистрибутивы и утилиты развертывания OpenStack меняли стартовые скрипты агентов, для того чтобы они использовали ml2_conf.ini. Сейчас правильным методом является использование openvswitch_agent.ini

Большинство компонентов общается между собой при помощи брокера сообщений. Задаем параметры аутентификации брокера сообщений RabbitMQ:

```
[root@controller ~]# crudini --set /etc/neutron/neutron.conf
DEFAULT transport_url rabbit://openstack:openstack@controller.
test.local
```

Прописываем параметры Keystone. Тут для вас не должно быть ничего нового:

```
[root@controller ~]# crudini --set /etc/neutron/neutron.conf
DEFAULT auth_strategy keystone
[root@controller ~]# crudini --set /etc/neutron/neutron.conf
keystone_authtoken project_name service
[root@controller ~]# crudini --set /etc/neutron/neutron.conf
```

```
keystone_authtoken user_domain_name default
[root@controller ~]# crudini --set /etc/neutron/neutron.conf
keystone_authtoken project_domain_name default
[root@controller ~]# crudini --set /etc/neutron/neutron.conf
keystone_authtoken auth_type password
[root@controller ~]# crudini --set /etc/neutron/neutron.conf
keystone_authtoken username neutron
[root@controller ~]# crudini --set /etc/neutron/neutron.conf
keystone_authtoken password openstack
[root@controller ~]# crudini --set /etc/neutron/neutron.conf
keystone_authtoken auth_uri http://controller.test.local:5000
[root@controller ~]# crudini --set /etc/neutron/neutron.conf
keystone_authtoken auth_url http://controller.test.local:35357
```

Теперь укажем, какой основной подключаемый модуль для реализации взаимодействия на втором уровне сетевого стека OSI мы будем использовать:

```
[root@controller ~]# crudini --set /etc/neutron/neutron.conf
DEFAULT core_plugin ml2
```

Помимо ml2, допустимыми значениями являются: hyperv, cisco, brocade, embrance, vmware, nec и др.

Задаем, какие подгружаемые модули будем использовать (примеры возможных вариантов: router, firewall, lbaas, vpnaas, metering). Добавим только маршрутизатор:

```
[root@controller ~]# crudini --set /etc/neutron/neutron.conf
DEFAULT service_plugins router
```

Мы использовали краткие имена подключаемых модулей. Также можно было ссылаться на них по длинным наименованиям классов, например: neutron.plugins.ml2.plugin:Ml2Plugin и neutron.services.l3_router.l3_router_plugin.L3RouterPlugin.

Разрешаем пересечение IP-адресов внутри проектов:

```
[root@controller ~]# crudini --set /etc/neutron/neutron.conf
DEFAULT allow_overlapping_ips True
```

В секции oslo_concurrency укажем путь к файлам блокировок:

```
[root@controller ~]# crudini --set /etc/neutron/neutron.conf
oslo_concurrency lock_path /var/lib/neutron/tmp
```

Теперь перейдем к настройке подключаемого модуля Modular Layer 2 (ML2). Подключаемый модуль ML2 параллельно может поддерживать работу нескольких сегментов разных типов, включая

flat, GRE, VLAN, VXLAN и local. При этом local подходит только для установок типа «все в одном».

Для туннелирования с поддержкой пересечения IP-адресов в OpenStack можно использовать несколько технологий. GRE – устоявшийся в индустрии стандарт для инкапсуляции фреймов второго уровня OSI (RFC 2784 и 2890). Номер IP-протокола – 47. VXLAN – более молодой стандарт, предложенный IETF в 2011 году. В качестве транспорта используется UDP-порт 4789. Также можно использовать VLAN и «плоскую сеть».

Мы в примере настройки выберем GRE-туннелирование:

```
[root@controller ~]# crudini --set /etc/neutron/plugins/ml2/ml2_
conf.ini ml2 type_drivers flat,gre,vlan,vxlan
[root@controller ~]# crudini --set /etc/neutron/plugins/ml2/ml2_
conf.ini ml2 tenant_network_types gre
```

В реальной производственной среде также для избежания фрагментации пакетов нам бы потребовалось увеличить MTU на сетевом оборудовании.

Указываем, что в качестве драйвера механизма к подключаемому модулю ML2 используем OVS (возможные варианты: openvswitch, linuxbridge и l2population) и диапазон ID для GRE-туннелей:

```
[root@controller ~]# crudini --set /etc/neutron/plugins/ml2/ml2_
conf.ini ml2 mechanism_drivers openvswitch
[root@controller ~]# crudini --set /etc/neutron/plugins/ml2/ml2_
conf.ini ml2_type_gre tunnel_id_ranges 1:1000
```

Включаем группы безопасности, ipset:

```
[root@controller ~]# crudini --set /etc/neutron/plugins/ml2/ml2_
conf.ini securitygroup enable_security_group True
[root@controller ~]# crudini --set /etc/neutron/plugins/ml2/ml2_
conf.ini securitygroup enable_ipset True
```

Далее мы будем выполнять настройки, специфичные для управляющего узла. Поэтому скопируйте файлы /etc/neutron/neutron.conf и /etc/neutron/plugins/ml2/ml2_conf.ini на узлы network, compute и compute-opt, где мы продолжим их редактирование.

Параметры подключения к базе данных Neutron:

```
[root@controller ~]# crudini --set /etc/neutron/neutron.conf
database connection mysql://neutron:neutron@controller.test.
local/neutron
```

Далее настройки также не нуждаются в комментариях. Вносим необходимые изменения:

```
[root@controller ~]# crudini --set /etc/neutron/neutron.conf
DEFAULT notify_nova_on_port_status_changes True
[root@controller ~]# crudini --set /etc/neutron/neutron.conf
DEFAULT notify_nova_on_port_data_changes True
[root@controller ~]# crudini --set /etc/neutron/neutron.conf nova
auth_url http://controller.test.local:35357
[root@controller ~]# crudini --set /etc/neutron/neutron.conf nova
auth_type password
[root@controller ~]# crudini --set /etc/neutron/neutron.conf nova
project_domain_name default
[root@controller ~]# crudini --set /etc/neutron/neutron.conf nova
user_domain_name default
[root@controller ~]# crudini --set /etc/neutron/neutron.conf nova
region_name RegionOne
[root@controller ~]# crudini --set /etc/neutron/neutron.conf nova
project_name service
[root@controller ~]# crudini --set /etc/neutron/neutron.conf nova
username nova
[root@controller ~]# crudini --set /etc/neutron/neutron.conf nova
password openstack
```

Указываем Nova, что за сеть отвечает Neutron и что необходимо отключить сервис брандмауэра в Nova:

```
[root@controller ~]# crudini --set /etc/nova/nova.conf DEFAULT
network_api_class nova.network.neutronv2.api.API
[root@controller ~]# crudini --set /etc/nova/nova.conf DEFAULT
security_group_api neutron
[root@controller ~]# crudini --set /etc/nova/nova.conf
DEFAULT linuxnet_interface_driver nova.network.linux_net.
LinuxOVSInterfaceDriver
[root@controller ~]# crudini --set /etc/nova/nova.conf DEFAULT
firewall_driver nova.virt.firewall.NoopFirewallDriver
```

И наконец, параметры аутентификации Neutron для Nova. Тут также комментарии излишни:

```
[root@controller ~]# crudini --set /etc/nova/nova.conf neutron
url http://controller.test.local:9696
[root@controller ~]# crudini --set /etc/nova/nova.conf neutron
auth_url http://controller.test.local:35357
[root@controller ~]# crudini --set /etc/nova/nova.conf neutron
auth_type password
[root@controller ~]# crudini --set /etc/nova/nova.conf neutron
project_domain_name default
[root@controller ~]# crudini --set /etc/nova/nova.conf neutron
```

```
user_domain_name default
[root@controller ~]# crudini --set /etc/nova/nova.conf neutron
region_name RegionOne
[root@controller ~]# crudini --set /etc/nova/nova.conf neutron
project_name service
[root@controller ~]# crudini --set /etc/nova/nova.conf neutron
username neutron
[root@controller ~]# crudini --set /etc/nova/nova.conf neutron
password openstack
```

Заполняем базу данных:

```
[root@controller ~]# su -s /bin/sh -c "neutron-db-manage --config-
file /etc/neutron/neutron.conf --config-file /etc/neutron/plugins/
ml2/ml2_conf.ini upgrade head" neutron
```

Поскольку Neutron ожидает, что настройки основного подключаемого модуля задаются в файлах директории /etc/neutron/, нам необходимо создать символическую ссылку на конфигурационный файл подключаемого модуля ML2:

```
[root@controller ~]# ln -s /etc/neutron/plugins/ml2/ml2_conf.ini
/etc/neutron/plugin.ini
```

Указываем использование прокси для сервера метаданных и общий секрет, которым будут подписываться запросы к серверу метаданных. Агента мы будем настраивать в следующем разделе на узле network:

```
[root@controller ~]# crudini --set /etc/nova/nova.conf neutron
service_metadata_proxy True
[root@controller ~]# crudini --set /etc/nova/nova.conf neutron
metadata_proxy_shared_secret openstack
```

Рестартуем и включаем сервисы:

```
[root@controller ~]# systemctl restart openstack-nova-api.service
openstack-nova-scheduler.service openstack-nova-conductor.service
[root@controller ~]# systemctl enable  neutron-server.service
[root@controller ~]# systemctl start neutron-server.service
```

Проверяем, что все расширения Neutron загружены и сервис работает:

```
$ neutron ext-list
+---------------------------+-----------------------------------------------+
| alias                     | name                                          |
+---------------------------+-----------------------------------------------+
```

```
| default-subnetpools       | Default Subnetpools                              |
| network-ip-availability   | Network IP Availability                          |
| network_availability_zone | Network Availability Zone                        |
| auto-allocated-topology   | Auto Allocated Topology Services                 |
| ext-gw-mode               | Neutron L3 Configurable external gateway mode    |
| binding                   | Port Binding                                     |
| agent                     | agent                                            |
| subnet_allocation         | Subnet Allocation                                |
| l3_agent_scheduler        | L3 Agent Scheduler                               |
| tag                       | Tag support                                      |
| external-net              | Neutron external network                         |
| flavors                   | Neutron Service Flavors                          |
| net-mtu                   | Network MTU                                      |
| availability_zone         | Availability Zone                                |
| quotas                    | Quota management support                         |
| l3-ha                     | HA Router extension                              |
| provider                  | Provider Network                                 |
| multi-provider            | Multi Provider Network                           |
| address-scope             | Address scope                                    |
| extraroute                | Neutron Extra Route                              |
| subnet-service-types      | Subnet service types                             |
| standard-attr-timestamp   | Resource timestamps                              |
| service-type              | Neutron Service Type Management                  |
| l3-flavors                | Router Flavor Extension                          |
| extra_dhcp_opt            | Neutron Extra DHCP opts                          |
| standard-attr-revisions   | Resource revision numbers                        |
| pagination                | Pagination support                               |
| sorting                   | Sorting support                                  |
| security-group            | security-group                                   |
| dhcp_agent_scheduler      | DHCP Agent Scheduler                             |
| router_availability_zone  | Router Availability Zone                         |
| rbac-policies             | RBAC Policies                                    |
| standard-attr-description | standard-attr-description                        |
| router                    | Neutron L3 Router                                |
| allowed-address-pairs     | Allowed Address Pairs                            |
| project-id                | project_id field enabled                         |
| dvr                       | Distributed Virtual Router                       |
+---------------------------+--------------------------------------------------+
```

Установка сетевого узла Neutron

Начинаем с установки необходимых пакетов:

```
[root@network ~]# yum -y install openstack-neutron openstack-
neutron-ml2 openstack-neutron-openvswitch openvswitch
```

Скопируем из временной директории перенесенные в процессе установки с управляющего узла конфигурационные файлы neut-

ron.conf в /etc/neutron/neutron.conf, а ml2_conf.ini – в /etc/neutron/plugins/ml2/ml2_conf.ini.

Вносим изменения в параметры ядра при помощи sysctl. Включаем маршрутизацию пакетов и выключаем фильтрацию пакетов по их исходящему адресу, которая по умолчанию предотвращает DDoS-атаки. В нашем случае эти правила применяет Neutron для каждого экземпляра виртуальной машины при помощи iptables.

```
[root@network ~]# vi /etc/sysctl.conf
[root@network ~]# sysctl -p
net.ipv4.ip_forward = 1
net.ipv4.conf.all.rp_filter = 0
net.ipv4.conf.default.rp_filter = 0
```

Вносим изменения в конфигурационный файл подключаемого Modular Layer 2. Указываем flat-провайдер для внешней сети:

```
[root@network ~]# crudini --set /etc/neutron/plugins/ml2/ml2_conf.
ini ml2_type_flat flat_networks external
```

Поскольку Neutron ожидает, что настройки основного подключаемого модуля задаются в файлах директории /etc/neutron/, нам необходимо создать символическую ссылку на конфигурационный файл подключаемого модуля ML2:

```
[root@network ~]# ln -s /etc/neutron/plugins/ml2/ml2_conf.ini /
etc/neutron/plugin.ini
```

Теперь от подключаемого модуля ML2 переходим к используемому им агенту OVS. Напомним, что вместо агента OVS мог быть использован Linux Bridge. Локальный IP-адрес для конечной точки туннеля:

```
[root@network ~]# crudini --set /etc/neutron/plugins/ml2/
openvswitch_agent.ini ovs local_ip 192.168.122.220
```

Сопоставление внешней сети мосту br-ex:

```
[root@network ~]# crudini --set /etc/neutron/plugins/ml2/
openvswitch_agent.ini ovs bridge_mappings external:br-ex
```

Включаем GRE-туннелирование:

```
[root@network ~]# crudini --set /etc/neutron/plugins/ml2/
openvswitch_agent.ini ovs enable_tunneling True
[root@network ~]# crudini --set /etc/neutron/plugins/ml2/
openvswitch_agent.ini agent tunnel_types gre
```

Если бы мы хотели использовать VXLAN-туннелирование, то последняя команда выглядела бы как

```
[root@network ~]# crudini --set /etc/neutron/plugins/ml2/
openvswitch_agent.ini agent tunnel_types vxlan
```

Настраиваем L3-агента, осуществляющего маршрутизацию:

```
[root@network ~]# crudini --set /etc/neutron/l3_agent.ini DEFAULT
interface_driver neutron.agent.linux.interface.OVSInterfaceDriver
[root@network ~]# crudini --set /etc/neutron/l3_agent.ini DEFAULT
use_namespaces True
[root@network ~]# crudini --set /etc/neutron/l3_agent.ini DEFAULT
external_network_bridge br-ex
[root@network ~]# crudini --set /etc/neutron/l3_agent.ini DEFAULT
router_delete_namespaces True
```

Если вы хотите разрешить несколько внешних сетей для одного агента, то параметр `external_network_bridge` можно оставить с пустым значением.

Задаем минимально необходимые настройки DHCP-агента:

```
[root@network ~]# crudini --set /etc/neutron/dhcp_agent.
ini DEFAULT interface_driver neutron.agent.linux.interface.
OVSInterfaceDriver
[root@network ~]# crudini --set /etc/neutron/dhcp_agent.ini
DEFAULT dhcp_driver neutron.agent.linux.dhcp.Dnsmasq
[root@network ~]# crudini --set /etc/neutron/dhcp_agent.ini
DEFAULT use_namespaces True
[root@network ~]# crudini --set /etc/neutron/dhcp_agent.ini
DEFAULT dhcp_delete_namespaces True
```

Теперь настраиваем агента, предоставляющего метаданные виртуальным машинам. Задаем IP-адрес контроллера и общий секрет, используемый для получения метаданных:

```
[root@network ~]# crudini --set /etc/neutron/metadata_agent.ini
DEFAULT nova_metadata_host 192.168.122.200
[root@network ~]# crudini --set /etc/neutron/metadata_agent.ini
DEFAULT metadata_proxy_shared_secret openstack
```

Теперь запускаем и включаем сервис Open vSwitch (OVS):

```
[root@network ~]# systemctl enable openvswitch.service
[root@network ~]# systemctl start openvswitch.service
```

Нам необходимо создать мост br-ex, к которому мы подключаем второй, не настроенный пока сетевой адаптер:

```
[root@network ~]# ovs-vsctl add-br br-ex
[root@network ~]# ovs-vsctl add-port br-ex eth1
```

Далее необходимо добавить настройки сети в файл конфигурации адаптера /etc/sysconfig/network-scripts/ifcfg-eth1:

```
TYPE=Ethernet
NAME=eth1
DEVICE=eth1
ONBOOT=yes
BOOTPROTO=static
IPADDR=10.100.1.250
NETMASK=255.255.255.0
GATEWAY=10.100.1.1
```

Другой способ – это назначение настроек IP непосредственно на мосту. В этом случае вам необходимо создать в директории /etc/sysconfig/network-scripts/ файл, описывающий конфигурацию моста br-ex с именем ifcfg-br-ex, который будет содержать параметры, которые до этого присутствовали в конфигурационном файле сетевого адаптера. Например:

```
DEVICE=br-ex
DEVICETYPE=ovs
TYPE=OVSBridge
BOOTPROTO=static
IPADDR=10.100.1.250
NETMASK=255.255.255.0
GATEWAY=10.100.1.1
ONBOOT=yes
```

После чего заменяем текущую конфигурацию сетевого адаптера, объявив его OVS-портом:

```
DEVICE=eth0
TYPE=OVSPort
DEVICETYPE=ovs
OVS_BRIDGE=br-ex
ONBOOT=yes
```

Проверяем конфигурацию Open vSwitch:

```
[root@network ~]# ovs-vsctl show
d9eb006a-b746-4875-947a-ef9dc0190975
    Bridge br-ex
        Port "eth1"
            Interface "eth1"
        Port br-ex
```

```
            Interface br-ex
                type: internal
    ovs_version: "2.8.2"

[root@network ~]# systemctl enable neutron-openvswitch-agent.
service neutron-l3-agent.service  neutron-dhcp-agent.service
neutron-metadata-agent.service neutron-ovs-cleanup.service
[root@network ~]# systemctl start neutron-openvswitch-agent.
service neutron-l3-agent.service neutron-dhcp-agent.service
neutron-metadata-agent.service
```

Убеждаемся, что все агенты работают:

```
$ openstack network agent list
+------+--------------------+--------------------+---------+-------+-------+---------------------------+
| ID   | Agent Type         | Host               | Av Zone | Alive | State | Binary                    |
+------+--------------------+--------------------+---------+-------+-------+---------------------------+
| 02.. | Metadata agent     | network.test.local | None    | :-)   | UP    | neutron-metadata-agent    |
| 25.. | Open vSwitch agent | network.test.local | None    | :-)   | UP    | neutron-openvswitch-agent |
| 40.. | L3 agent           | network.test.local | nova    | :-)   | UP    | neutron-l3-agent          |
| 59.. | DHCP agent         | network.test.local | nova    | :-)   | UP    | neutron-dhcp-agent        |
+------+--------------------+--------------------+---------+-------+-------+---------------------------+
```

Теперь можно перейти к настройке вычислительных узлов compute и compute-opt (если он присутствует в вашей конфигурации).

Установка вычислительного узла Neutron

Выполняем дальнейшие инструкции на вычислительных узлах. Начинаем с установки необходимых пакетов. Их список аналогичен списку пакетов сетевого узла:

```
[root@compute ~]# yum -y install openstack-neutron openstack-
neutron-ml2 openstack-neutron-openvswitch
```

Еще раз скопируем из временной директории перенесенные в процессе установки с управляющего узла конфигурационные файлы neutron.conf в /etc/neutron/neutron.conf, а ml2_conf.ini – в /etc/neutron/plugins/ml2/ml2_conf.ini.

Вносим изменения в параметры ядра при помощи sysctl. Выключаем фильтрацию пакетов по их исходящему адресу, которая по умолчанию предотвращает DDoS-атаки. В нашем случае эти правила применяет Neutron для каждого экземпляра виртуальной

машины при помощи iptables. Также задаем, что пакеты с сетевого моста передаются на обработку iptables:

```
[root@compute ~]# vi /etc/sysctl.conf
[root@compute ~]# sysctl -p
net.ipv4.conf.all.rp_filter = 0
net.ipv4.conf.default.rp_filter = 0
net.bridge.bridge-nf-call-iptables = 1
```

Вносим изменения в конфигурационный файл агента OVS. Агент OVS использует подключаемый модуль Modular Layer 2. Локальный IP-адрес вычислительного узла:

```
[root@compute ~]# crudini --set /etc/neutron/plugins/ml2/
openvswitch_agent.ini ovs local_ip 192.168.122.210

[root@compute-opt ~]# crudini --set /etc/neutron/plugins/ml2/
openvswitch_agent.ini ovs local_ip 192.168.122.215
```

Включаем тип туннеля GRE в секции agent:

```
[root@compute ~]# crudini --set /etc/neutron/plugins/ml2/
openvswitch_agent.ini agent tunnel_types gre
```

Выбираем традиционный драйвер брандмауэра, реализованный с правилами iptables на специально созданном для этого мосту Linux bridge:

```
[root@compute ~]# crudini --set /etc/neutron/plugins/ml2/
openvswitch_agent.ini securitygroup firewall_driver neutron.agent.
linux.iptables_firewall.OVSHybridIptablesFirewallDriver
```

Подробнее об этой настройке мы поговорим в десятой главе. Запускаем и включаем сервис Open vSwitch (OVS):

```
[root@compute ~]# systemctl enable openvswitch.service
[root@compute ~]# systemctl start openvswitch.service
```

Указываем сервису Nova, что за сеть отвечает Neutron и что необходимо отключить службу брандмауэра в Nova:

```
[root@compute ~]# crudini --set /etc/nova/nova.conf DEFAULT
network_api_class nova.network.neutronv2.api.API
[root@compute ~]# crudini --set /etc/nova/nova.conf DEFAULT
security_group_api neutron
[root@compute ~]# crudini --set /etc/nova/nova.conf
DEFAULT linuxnet_interface_driver nova.network.linux_net.
LinuxOVSInterfaceDriver
```

```
[root@compute ~]# crudini --set /etc/nova/nova.conf DEFAULT
firewall_driver nova.virt.firewall.NoopFirewallDriver
```

Как и на управляющем узле, зададим параметры аутентификации Neutron для Nova:

```
[root@compute ~]# crudini --set /etc/nova/nova.conf neutron url
http://controller.test.local:9696
[root@compute ~]# crudini --set /etc/nova/nova.conf neutron
auth_url http://controller.test.local:35357
[root@compute ~]# crudini --set /etc/nova/nova.conf neutron
auth_type password
[root@compute ~]# crudini --set /etc/nova/nova.conf neutron
project_domain_name default
[root@compute ~]# crudini --set /etc/nova/nova.conf neutron
user_domain_name default
[root@compute ~]# crudini --set /etc/nova/nova.conf neutron
region_name RegionOne
[root@compute ~]# crudini --set /etc/nova/nova.conf neutron
project_name service
[root@compute ~]# crudini --set /etc/nova/nova.conf neutron
username neutron
[root@compute ~]# crudini --set /etc/nova/nova.conf neutron
password openstack
```

Как и ранее, нам необходимо создать символическую ссылку на конфигурационный файл подключаемого модуля ML2:

```
[root@compute ~]# ln -s /etc/neutron/plugins/ml2/ml2_conf.ini /etc/
neutron/plugin.ini
```

Рестартуем сервис Nova и включаем и стартуем агента Open vSwitch:

```
[root@compute ~]# systemctl restart openstack-nova-compute.service
[root@compute ~]# systemctl enable neutron-openvswitch-agent.service
[root@compute ~]# systemctl start neutron-openvswitch-agent.service
```

Для проверки повторим команду, показывающую список агентов, и убедимся, что добавились агенты Open vSwitch на вычислительных узлах:

```
$ openstack network agent list
+------+--------------------+-----------------------+---------+-------+-------+---------------------------+
| ID   | Agent Type         | Host                  | Av Zone | Alive | State | Binary                    |
+------+--------------------+-----------------------+---------+-------+-------+---------------------------+
| 02.. | Metadata agent     | network.test.local    | None    | :-)   | UP    | neutron-metadata-agent    |
| 1e.. | Open vSwitch agent | compute-opt.test.local| None    | :-)   | UP    | neutron-openvswitch-agent |
| 25.. | Open vSwitch agent | network.test.local    | None    | :-)   | UP    | neutron-openvswitch-agent |
| 40.. | L3 agent           | network.test.local    | nova    | :-)   | UP    | neutron-l3-agent          |
```

```
| 59.. | DHCP agent          | network.test.local     | nova    | :-)   | UP    | neutron-dhcp-agent        |
| 9b.. | Open vSwitch agent | compute.test.local     | None    | :-)   | UP    | neutron-openvswitch-agent |
+------+--------------------+------------------------+---------+-------+-------+---------------------------+
```

Основные компоненты установлены. В следующей главе мы, наконец, начнем работать с виртуальными машинами и сетями. В табл. 4 приведены файлы журналов служб Neutron:

Таблица 4. Файлы журналов служб Neutron

Служба	Месторасположение журнала
neutron-dhcp-agent	/var/log/neutron/dhcp-agent.log
neutron-l3-agent	/var/log/neutron/l3-agent.log
neutron-metadata-agent	/var/log/neutron/metadata-agent.log
neutron-openvswitch-agent	/var/log/neutron/openvswitch-agent.log
neutron-ovs-cleanup	/var/log/neutron/ovs-cleanup.log
neutron-server	/var/log/neutron/server.log

Глава 9

Работа с виртуальными машинами из командной строки

Начнем рассмотрение работы с нашим учебным облаком с изучения сетевых компонентов.

Сеть в OpenStack

Создадим из командной строки минимально необходимые компоненты сетевой инфраструктуры для запуска экземпляра виртуальной машины. Первое, что нужно сделать, – это создать внешнюю сеть, которую назовем ext-net. Ее нам необходимо создать, работая с привилегиями администратора, и для этого мы воспользуемся командой `openstack network create`. Также нужно сказать, что до сих пор доступна команда `neutron`. Работать с утилитой neutron можно как интерактивно, так и вызывая с параметрами. Команда `neutron help` выведет список всех подкоманд с описанием, а используя синтаксис neutron help <имя_подкоманды>, можно получить более подробную справку. Работать мы можем с узла, на котором установлены пакеты python-neutronclient и python-novaclient, используя традиционные утилиты или python-openstackclient для команды `openstack`.

```
$ source keystonerc_admin
$ openstack network create --external --share --provider-network-type
flat --provider-physical-network datacentre ext-net
+---------------------------+--------------------------------------+
| Field                     | Value                                |
+---------------------------+--------------------------------------+
```

```
| admin_state_up            | UP                                   |
| availability_zone_hints   |                                      |
| availability_zones        |                                      |
| created_at                | 2018-03-06T20:43:08Z                 |
| description               |                                      |
| dns_domain                | None                                 |
| id                        | 1d25a0b0-f1a4-49c2-9388-bed695c11267 |
| ipv4_address_scope        | None                                 |
| ipv6_address_scope        | None                                 |
| is_default                | False                                |
| is_vlan_transparent       | None                                 |
| mtu                       | 1500                                 |
| name                      | ext-net                              |
| port_security_enabled     | False                                |
| project_id                | 7fe2a6ef08df4a749f3bad1fceb055b9     |
| provider:network_type     | flat                                 |
| provider:physical_network | datacentre                           |
| provider:segmentation_id  | None                                 |
| qos_policy_id             | None                                 |
| revision_number           | 4                                    |
| router:external           | External                             |
| segments                  | None                                 |
| shared                    | True                                 |
| status                    | ACTIVE                               |
| subnets                   |                                      |
| tags                      |                                      |
| updated_at                | 2018-03-06T20:43:08Z                 |
+---------------------------+--------------------------------------+
```

Как мы видим, на текущий момент какие-либо подсети отсутствуют. Создадим подсеть для внешней сети, из которой у нас будут выделяться плавающие IP-адреса. В реальных внедрениях это могут быть «реальные» маршрутизируемые в Интернете IP-адреса:

```
$ openstack subnet create --network ext-net --no-dhcp --allocation-
pool start=10.100.1.100,end=10.100.1.200 --gateway 10.100.1.1
--subnet-range 10.100.1.0/24 ext-subnet
+-------------------+--------------------------------------+
| Field             | Value                                |
+-------------------+--------------------------------------+
| allocation_pools  | 10.100.1.100-10.100.1.200            |
| cidr              | 10.100.1.0/24                        |
| created_at        | 2018-03-06T20:45:22Z                 |
| description       |                                      |
| dns_nameservers   |                                      |
| enable_dhcp       | False                                |
| gateway_ip        | 10.100.1.1                           |
| host_routes       |                                      |
| id                | 18b64199-932b-46ee-9537-06bf83f6d4d7 |
| ip_version        | 4                                    |
```

```
| ipv6_address_mode | None                                 |
| ipv6_ra_mode      | None                                 |
| name              | ext-subnet                           |
| network_id        | 1d25a0b0-f1a4-49c2-9388-bed695c11267 |
| project_id        | 7fe2a6ef08df4a749f3bad1fceb055b9     |
| revision_number   | 0                                    |
| segment_id        | None                                 |
| service_types     |                                      |
| subnetpool_id     | None                                 |
| tags              |                                      |
| updated_at        | 2018-03-06T20:45:22Z                 |
+-------------------+--------------------------------------+
```

Можно посмотреть список сетей командой `openstack network list`, а подробности, касающиеся конкретной сети, – при помощи openstack network show <имя_сети>. Далее продолжим работу в качестве пользователя demo. Создадим внутреннюю сеть, которая будет использоваться в проекте demo. На этот раз для демонстрации воспользуемся командой `neutron`:

```
$ source keystonerc_demo
$ neutron net-create demo-net
neutron CLI is deprecated and will be removed in the future. Use
openstack CLI instead.
Created a new network:
+-------------------------+--------------------------------------+
| Field                   | Value                                |
+-------------------------+--------------------------------------+
| admin_state_up          | True                                 |
| availability_zone_hints |                                      |
| availability_zones      |                                      |
| created_at              | 2018-03-06T20:46:26Z                 |
| description             |                                      |
| id                      | 330c12e5-d560-4b85-abc0-05de6ee7cff4 |
| ipv4_address_scope      |                                      |
| ipv6_address_scope      |                                      |
| is_default              | False                                |
| mtu                     | 1458                                 |
| name                    | demo-net                             |
| project_id              | bc10ac4b71164550a363b8098e8ad270     |
| revision_number         | 2                                    |
| router:external         | False                                |
| shared                  | False                                |
| status                  | ACTIVE                               |
| subnets                 |                                      |
| tags                    |                                      |
| tenant_id               | bc10ac4b71164550a363b8098e8ad270     |
| updated_at              | 2018-03-06T20:46:26Z                 |
+-------------------------+--------------------------------------+
```

Обратите внимание на то, что команда neutron считается устаревшей. Затем создадим подсеть в сети demo-net:

```
$ neutron subnet-create demo-net --name demo-subnet --gateway
172.16.0.1 172.16.0.0/24
neutron CLI is deprecated and will be removed in the future. Use
openstack CLI instead.
Created a new subnet:
+-------------------+-----------------------------------------------+
| Field             | Value                                         |
+-------------------+-----------------------------------------------+
| allocation_pools  | {"start": "172.16.0.2", "end": "172.16.0.254"} |
| cidr              | 172.16.0.0/24                                 |
| created_at        | 2018-03-06T20:47:23Z                          |
| description       |                                               |
| dns_nameservers   |                                               |
| enable_dhcp       | True                                          |
| gateway_ip        | 172.16.0.1                                    |
| host_routes       |                                               |
| id                | e3729508-98d6-4fd1-856d-03602c6b178e          |
| ip_version        | 4                                             |
| ipv6_address_mode |                                               |
| ipv6_ra_mode      |                                               |
| name              | demo-subnet                                   |
| network_id        | 330c12e5-d560-4b85-abc0-05de6ee7cff4          |
| project_id        | bc10ac4b71164550a363b8098e8ad270              |
| revision_number   | 0                                             |
| service_types     |                                               |
| subnetpool_id     |                                               |
| tags              |                                               |
| tenant_id         | bc10ac4b71164550a363b8098e8ad270              |
| updated_at        | 2018-03-06T20:47:23Z                          |
+-------------------+-----------------------------------------------+
```

Мы использовали параметр --gateway. Когда подсеть будет подключена к маршрутизатору, интерфейс маршрутизатора получит заданный IP-адрес 172.16.0.1. Теперь создадим этот маршрутизатор. Вернемся к использованию команды openstack:

```
$ openstack router create demo-router
+-------------------------+--------------------------------------+
| Field                   | Value                                |
+-------------------------+--------------------------------------+
| admin_state_up          | UP                                   |
| availability_zone_hints |                                      |
| availability_zones      |                                      |
| created_at              | 2018-03-04T18:20:38Z                 |
| description             |                                      |
| distributed             | False                                |
```

```
| external_gateway_info   | None                                 |
| flavor_id               | None                                 |
| ha                      | False                                |
| id                      | 006630d4-5cb6-4d72-adee-4c8f71f5cbd4 |
| name                    | demo-router                          |
| project_id              | bc10ac4b71164550a363b8098e8ad270     |
| revision_number         | 1                                    |
| routes                  |                                      |
| status                  | ACTIVE                               |
| tags                    |                                      |
| updated_at              | 2018-03-04T18:20:38Z                 |
+-------------------------+--------------------------------------+
```

и подключим интерфейс маршрутизатора demo-router к подсети demo-subnet:

```
$ openstack router add subnet demo-router demo-subnet
```

Осталось только установить маршрутизатору demo-router в качестве шлюза внешнюю сеть ext-net:

```
$ neutron router-gateway-set demo-router ext-net
neutron CLI is deprecated and will be removed in the future. Use
openstack CLI instead.
Set gateway for router demo-router
```

На этом этапе почти все готово для запуска первой виртуальной машины. Теперь можно просмотреть список всех существующих портов:

```
$ source keystonerc_admin
$ openstack port list
+------+------+-------------------+-------------------------------------------------+--------+
| ID   | Name | MAC Address       | Fixed IP Addresses                              | Status |
+------+------+-------------------+-------------------------------------------------+--------+
| 11.. |      | fa:16:3e:79:ea:e4 | ip_address='10.100.1.108', subnet_id='18b641..' | DOWN   |
| 86.. |      | fa:16:3e:8d:ea:4b | ip_address='172.16.0.1', subnet_id='e37295..'   | ACTIVE |
| cf.. |      | fa:16:3e:c6:79:ac | ip_address='172.16.0.2', subnet_id='e37295..'   | ACTIVE |
+------+------+-------------------+-------------------------------------------------+--------+
```

Как было сказано ранее, порт – это логическое подключение ресурса, например виртуальной машины или маршрутизатора, к подсети. Просмотреть информацию о конкретном порте можно при помощи команды `neutron port-show` или `openstack port show`:

```
$ openstack port show 866cd6d8-7e1f-43a8-a25b-cffb29347ec3
+-----------------------+-------------------------------------------+
| Field                 | Value                                     |
+-----------------------+-------------------------------------------+
```

```
| admin_state_up        | UP                                                |
| allowed_address_pairs |                                                   |
| binding_host_id       | network.test.local                                |
| binding_profile       |                                                   |
| binding_vif_details   | datapath_type='system',                           |
|                       | ovs_hybrid_plug='False', port_filter='True'       |
| binding_vif_type      | ovs                                               |
| binding_vnic_type     | normal                                            |
| created_at            | 2018-03-06T20:51:49Z                              |
| data_plane_status     | None                                              |
| description           |                                                   |
| device_id             | eb5aa33d-726a-46a7-a589-d78f99d1c2eb              |
| device_owner          | network:router_interface                          |
| dns_assignment        | None                                              |
| dns_name              | None                                              |
| extra_dhcp_opts       |                                                   |
| fixed_ips             | ip_address='172.16.0.1', subnet_id='e37..         |
| id                    | 866cd6d8-7e1f-43a8-a25b-cffb29347ec3              |
| ip_address            | None                                              |
| mac_address           | fa:16:3e:8d:ea:4b                                 |
| name                  |                                                   |
| network_id            | 330c12e5-d560-4b85-abc0-05de6ee7cff4              |
| option_name           | None                                              |
| option_value          | None                                              |
| port_security_enabled | False                                             |
| project_id            | bc10ac4b71164550a363b8098e8ad270                  |
| qos_policy_id         | None                                              |
| revision_number       | 9                                                 |
| security_group_ids    |                                                   |
| status                | ACTIVE                                            |
| subnet_id             | None                                              |
| tags                  |                                                   |
| trunk_details         | None                                              |
| updated_at            | 2018-03-06T20:51:59Z                              |
+-----------------------+---------------------------------------------------+
```

Запускаем экземпляр виртуальной машины

Прежде чем создать экземпляр виртуальной машины, познакомимся с понятием flavor. Фактически это тип создаваемой виртуальной машины. До версии Mitaka при установке OpenStack создавалось несколько предустановленных типов. Начиная с Mitaka, вам необходимо создавать flavors самостоятельно так, как это описано ниже. Посмотрим, какие предустановленные типы «из коробки» мы бы увидели на стенде с одной из старых версий OpenStack:

```
$ nova flavor-list
+----+-----------+--------+------+-----+------+-------+-------------+-----------+
| ID | Name      | Mem_MB | Disk | Eph | Swap | VCPUs | RXTX_Factor | Is_Public |
+----+-----------+--------+------+-----+------+-------+-------------+-----------+
| 1  | m1.tiny   | 512    | 1    | 0   |      | 1     | 1.0         | True      |
| 2  | m1.small  | 2048   | 20   | 0   |      | 1     | 1.0         | True      |
| 3  | m1.medium | 4096   | 40   | 0   |      | 2     | 1.0         | True      |
| 4  | m1.large  | 8192   | 80   | 0   |      | 4     | 1.0         | True      |
| 5  | m1.xlarge | 16384  | 160  | 0   |      | 8     | 1.0         | True      |
+----+-----------+--------+------+-----+------+-------+-------------+-----------+
```

Из перечисленных параметров нужно прокомментировать Ephemeral – размер второго временного диска, данные которого, как и первого, теряются при выключении виртуальной машины. Swap – размер опционального swap-раздела. RXTX_Factor позволяет изменять пропускную способность сети. Значение по умолчанию 1.0 обозначает пропускную способность как у подключенной сети. Данный параметр используется только в драйвере гипервизора Xen.

В нашем случае вывод команды `nova flavor-list` или `openstack flavor list` будет пуст. Создадим `flavor` для наших экспериментов. Укажем, что этот шаблон доступен всем пользователям облака, размер диска 1 Гб и используется 1 vCPU, а размер оперативной памяти зададим 300 Мб:

```
$ source keystonerc_admin
$ openstack flavor create --ram 300 --disk 1 --vcpu 1 --public m2.tiny
+----------------------------+--------------------------------------+
| Field                      | Value                                |
+----------------------------+--------------------------------------+
| OS-FLV-DISABLED:disabled   | False                                |
| OS-FLV-EXT-DATA:ephemeral  | 0                                    |
| disk                       | 1                                    |
| id                         | a5515a35-c0a2-4d36-8997-93fcbf73d18b |
| name                       | m2.tiny                              |
| os-flavor-access:is_public | True                                 |
| properties                 |                                      |
| ram                        | 300                                  |
| rxtx_factor                | 1.0                                  |
| swap                       |                                      |
| vcpus                      | 1                                    |
+----------------------------+--------------------------------------+
```

Соответствующая команда nova выглядит так:

```
$ nova flavor-create --is-public true m2.tiny auto 300 1 1
+----------+---------+--------+------+-----+------+-------+------+-----------+
| ID       | Name    | Mem_MB | Disk | Eph | Swap | VCPUs | RXTX | Is_Public |
+----------+---------+--------+------+-----+------+-------+------+-----------+
```

```
| 98c...e3 | m2.tiny | 300    | 1    | 0   |      | 1     | 1.0  | True      |
+----------+---------+--------+------+-----+------+-------+------+-----------+
```

Для того чтобы просмотреть информацию о шаблоне, используйте команду `openstack flavor show`. По умолчанию только администратор может создавать новые шаблоны и просматривать их полный список. Для того чтобы разрешить пользователям создавать свои шаблоны, необходимо в файле /etc/nova/policy.json поменять строку

```
"compute_extension:flavormanage": "rule:admin_api",
```

на

```
"compute_extension:flavormanage": "",
```

Как уже упоминалось ранее, сервер метаданных Nova может передавать те или иные параметры экземпляру виртуальной машины. В частности, это относится к открытым ssh-ключам. Создадим пару ключей и сохраним закрытый ключ в файл:

```
$ source keystonerc_demo
$ nova keypair-add demokey1 > ~/demokey1
```

или:

```
$ source keystonerc_demo
$ openstack keypair create demokey1 > ~/demokey1
```

Проверить список ключей, доступных вам, можно командой `openstack` или `nova keypair-list`:

```
$ openstack keypair list
+----------+-------------------------------------------------+
| Name     | Fingerprint                                     |
+----------+-------------------------------------------------+
| demokey1 | b6:3e:bd:fb:81:0f:f2:2b:60:8c:e2:f1:f4:bc:c4:ed |
+----------+-------------------------------------------------+
```

Теперь попробуем запустить экземпляр виртуальной машины на основе своего шаблона m2.tiny с ключом `demokey1`:

```
$ openstack server create --image cirros-0.4.0-x86_64 --flavor m2.tiny
--key-name demokey1 --nic net-id=demo-net myinstance1
+----------------------------+--------------------------------------+
| Field                      | Value                                |
+----------------------------+--------------------------------------+
| OS-DCF:diskConfig          | MANUAL                               |
```

```
| OS-EXT-AZ:availability_zone |                                      |
| OS-EXT-STS:power_state      | NOSTATE                              |
| OS-EXT-STS:task_state       | scheduling                           |
| OS-EXT-STS:vm_state         | building                             |
| OS-SRV-USG:launched_at      | None                                 |
| OS-SRV-USG:terminated_at    | None                                 |
| accessIPv4                  |                                      |
| accessIPv6                  |                                      |
| addresses                   |                                      |
| adminPass                   | GDyA9hzW85j4                         |
| config_drive                |                                      |
| created                     | 2018-03-06T21:02:54Z                 |
| flavor                      | m2.tiny (a5515a35-c0a2-4d36-8997-9..)|
| hostId                      |                                      |
| id                          | 10e4a71b-6162-4984-a410-3dff74b1e667 |
| image                       | cirros-0.4.0-x86_64 (c8ccc9b3-29b..) |
| key_name                    | demokey1                             |
| name                        | myinstance1                          |
| progress                    | 0                                    |
| project_id                  | bc10ac4b71164550a363b8098e8ad270     |
| properties                  |                                      |
| security_groups             | name='default'                       |
| status                      | BUILD                                |
| updated                     | 2018-03-06T21:02:54Z                 |
| user_id                     | 3b76dece42b140e092dc1a76a85c1879     |
| volumes_attached            |                                      |
+-----------------------------+--------------------------------------+
```

Соответствующий синтаксис команды `nova`:

```
$ nova boot --flavor m2.tiny --image cirros-0.4.0-x86_64 --key-name
demokey1 myinstance1
```

Запуск экземпляра виртуальной машины занимает некоторое время. Можно попробовать периодически запускать команду `nova list` или `openstack server list`, пока статус экземпляра не поменяется на `active`:

```
$ openstack server list
+-------+------------+--------+--------------------+--------------------+---------+
| ID    | Name       | Status | Networks           | Image              | Flavor  |
+-------+------------+--------+--------------------+--------------------+---------+
| 10e.. | myinstance1 | ACTIVE | demo-net=172.16.0.5 | cirros-0.4.0-x86_64 | m2.tiny |
+-------+------------+--------+--------------------+--------------------+---------+
```

Можно подключиться к гипервизору и посмотреть список виртуальных машин при помощи virsh, а также параметры, с которыми через libvirt был запущен экземпляр виртуальной машины:

```
$ ssh root@192.168.122.215
# virsh list
 Id    Name                           State
----------------------------------------------------
 1     instance-0000000a              running

[root@compute-opt ~]# virsh dumpxml instance-0000000a
<domain type='kvm' id='1'>
  <name>instance-0000000a</name>
  <uuid>10e4a71b-6162-4984-a410-3dff74b1e667</uuid>
  <metadata>
    <nova:instance xmlns:nova="http://openstack.org/xmlns/libvirt/
nova/1.0">
      <nova:package version="17.0.0-1.el7"/>
      <nova:name>myinstance1</nova:name>
      <nova:creationTime>2018-03-06 21:03:07</nova:creationTime>
      <nova:flavor name="m2.tiny">
        <nova:memory>300</nova:memory>
        <nova:disk>1</nova:disk>
        <nova:swap>0</nova:swap>
        <nova:ephemeral>0</nova:ephemeral>
        <nova:vcpus>1</nova:vcpus>
      </nova:flavor>
      <nova:owner>
        <nova:user uuid="3b76dece42b140e092dc1a76a85c1879">demo</
nova:user>
        <nova:project uuid="bc10ac4b71164550a363b8098e8ad270">demo</
nova:project>
      </nova:owner>
      <nova:root type="image" uuid="c8ccc9b3-29bb-4220-be38-
8f261ac8b99a"/>
    </nova:instance>
  </metadata>
  <memory unit='KiB'>307200</memory>
  <currentMemory unit='KiB'>307200</currentMemory>
  <vcpu placement='static'>1</vcpu>
  <cputune>
    <shares>1024</shares>
  </cputune>
  <resource>
    <partition>/machine</partition>
  </resource>
  <sysinfo type='smbios'>
    <system>
      <entry name='manufacturer'>RDO</entry>
      <entry name='product'>OpenStack Compute</entry>
      <entry name='version'>17.0.0-1.el7</entry>
      <entry name='serial'>c81dde69-60f6-42b6-8ad3-4fe71ae5b698</entry>
      <entry name='uuid'>10e4a71b-6162-4984-a410-3dff74b1e667</entry>
      <entry name='family'>Virtual Machine</entry>
    </system>
```

```
  </sysinfo>
  <os>
    <type arch='x86_64' machine='pc-i440fx-rhel7.4.0'>hvm</type>
    <boot dev='hd'/>
    <smbios mode='sysinfo'/>
  </os>
  <features>
    <acpi/>
    <apic/>
  </features>
  <cpu mode='custom' match='exact' check='full'>
    <model fallback='forbid'>Broadwell-noTSX</model>
    <vendor>Intel</vendor>
    <topology sockets='1' cores='1' threads='1'/>
    <feature policy='require' name='vme'/>
    <feature policy='require' name='ss'/>
    <feature policy='require' name='f16c'/>
    <feature policy='require' name='rdrand'/>
    <feature policy='require' name='hypervisor'/>
    <feature policy='require' name='arat'/>
    <feature policy='require' name='tsc_adjust'/>
    <feature policy='require' name='xsaveopt'/>
    <feature policy='require' name='abm'/>
    <feature policy='disable' name='invpcid'/>
  </cpu>
  <clock offset='utc'>
    <timer name='pit' tickpolicy='delay'/>
    <timer name='rtc' tickpolicy='catchup'/>
    <timer name='hpet' present='no'/>
  </clock>
  <on_poweroff>destroy</on_poweroff>
  <on_reboot>restart</on_reboot>
  <on_crash>destroy</on_crash>
  <devices>
    <emulator>/usr/libexec/qemu-kvm</emulator>
    <disk type='file' device='disk'>
      <driver name='qemu' type='qcow2' cache='none'/>
      <source file='/var/lib/nova/instances/10e4a71b-6162-4984-a410-
3dff74b1e667/disk'/>
      <backingStore type='file' index='1'>
        <format type='raw'/>
        <source file='/var/lib/nova/instances/_base/2fd07305443c5
0eaf54493008b80b9f329e88b30'/>
        <backingStore/>
      </backingStore>
      <target dev='vda' bus='virtio'/>
      <alias name='virtio-disk0'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x04'
function='0x0'/>
    </disk>
    <controller type='usb' index='0' model='piix3-uhci'>
```

```
      <alias name='usb'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x01'
function='0x2'/>
    </controller>
    <controller type='pci' index='0' model='pci-root'>
      <alias name='pci.0'/>
    </controller>
    <interface type='bridge'>
      <mac address='fa:16:3e:95:67:8b'/>
      <source bridge='qbr4194b0a8-77'/>
      <target dev='tap4194b0a8-77'/>
      <model type='virtio'/>
      <alias name='net0'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x03'
function='0x0'/>
    </interface>
    <serial type='pty'>
      <source path='/dev/pts/1'/>
      <log file='/var/lib/nova/instances/10e4a71b-6162-4984-a410-
3dff74b1e667/console.log' append='off'/>
      <target port='0'/>
      <alias name='serial0'/>
    </serial>
    <console type='pty' tty='/dev/pts/1'>
      <source path='/dev/pts/1'/>
      <log file='/var/lib/nova/instances/10e4a71b-6162-4984-a410-
3dff74b1e667/console.log' append='off'/>
      <target type='serial' port='0'/>
      <alias name='serial0'/>
    </console>
    <input type='tablet' bus='usb'>
      <alias name='input0'/>
      <address type='usb' bus='0' port='1'/>
    </input>
    <input type='mouse' bus='ps2'>
      <alias name='input1'/>
    </input>
    <input type='keyboard' bus='ps2'>
      <alias name='input2'/>
    </input>
    <graphics type='vnc' port='5900' autoport='yes' listen='0.0.0.0'
keymap='en-us'>
      <listen type='address' address='0.0.0.0'/>
    </graphics>
    <video>
      <model type='cirrus' vram='16384' heads='1' primary='yes'/>
      <alias name='video0'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x02'
function='0x0'/>
    </video>
    <memballoon model='virtio'>
```

```
      <stats period='10'/>
      <alias name='balloon0'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x05'
function='0x0'/>
    </memballoon>
  </devices>
  <seclabel type='none' model='none'/>
  <seclabel type='dynamic' model='dac' relabel='yes'>
    <label>+107:+107</label>
    <imagelabel>+107:+107</imagelabel>
  </seclabel>
</domain>
```

На этом этапе можно получить доступ к консоли виртуальной машины при помощи noVNC. Для этого нам нужно получить ссылку при помощи команды `nova get-vnc-console myinstance1 novnc` или `openstack console url show myinstance1`, которую дальше нужно скопировать в адресную строку браузера:

```
$ openstack console url show myinstance1
+-------+-------------------------------------------------------------+
| Field | Value                                                       |
+-------+-------------------------------------------------------------+
| type  | novnc                                                       |
| url   | http://controller.test.local:6080/vnc_auto.html?token=c1.. |
+-------+-------------------------------------------------------------+
```

Если вы все сделали правильно, то получите доступ к консоли, подобной изображенной на рис. 9.1.

Что произошло бы в случае ошибки? Давайте смоделируем ситуацию:

```
$ nova boot --flavor m1.xlarge --image cirros-0.4.0-x86_64 --key-
name demokey1  myinstance2
```

Если вы внимательно посмотрите на строку запуска виртуальной машины, то заметите, что был выбран тип виртуальной машины m1.xlarge, для которого требуются 16 Гб оперативной памяти. В тестовом окружении автора нет вычислительного узла с требуемым объемом. Если попробовать выполнить команду `nova list`, то в статусе мы увидим «ERROR». Чтобы узнать причину ошибки, используем команду `nova show`:

```
$ nova show myinstance2 | grep fault
| fault                               | {"message": "No valid host was
found. ", "code": 500, "created": "2018-03-05T19:15:03Z"} |
```

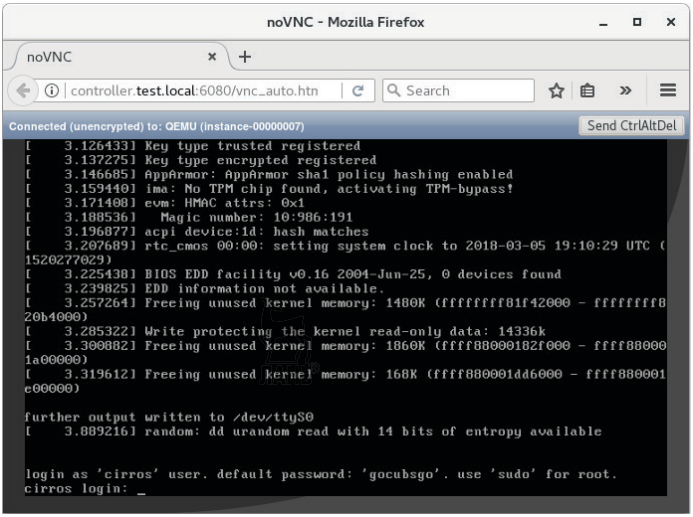


Рис. 9.1. Доступ к консоли виртуальной машины через noVNC

На двух последующих рис. 9.2 и 9.3 автор отобразил последовательность шагов при запуске виртуальной машины. В процессе запуска, как вы видите, используются практически все рассмотренные ранее сервисы OpenStack.

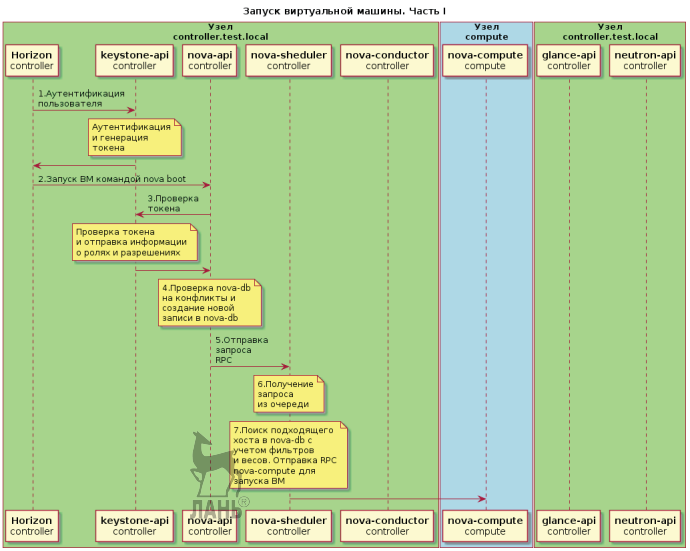


Рис. 9.2. Последовательность шагов при запуске виртуальной машины. Часть 1

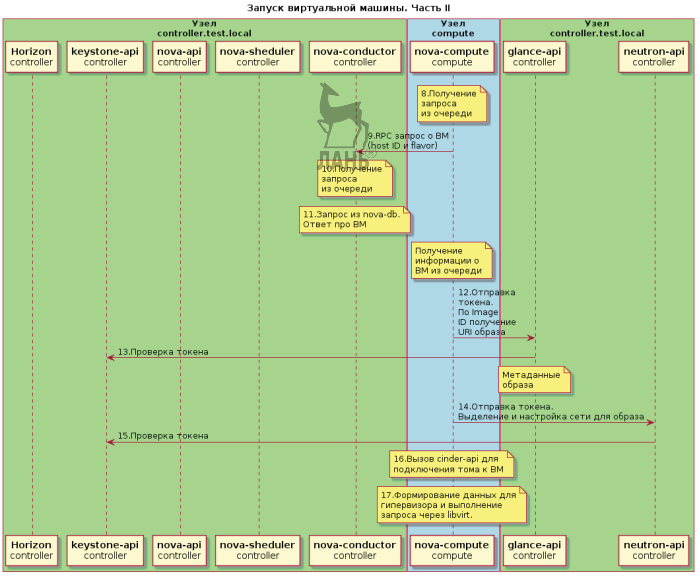


Рис. 9.3. Последовательность шагов при запуске виртуальной машины. Часть 2

Познакомимся еще с несколькими командами на примере задачи по поиску информации о том, на каком гипервизоре запущена конкретная виртуальная машина.

Получим список всех вычислительных узлов:

```
$ source keystonerc_admin
$ nova hypervisor-list
+--------------------------------------+------------------------+-------+---------+
| ID                                   | Hypervisor hostname    | State | Status  |
+--------------------------------------+------------------------+-------+---------+
| 658bd88c-8878-489c-9a9a-1239295ba61c | compute.test.local     | up    | enabled |
| 6989b8db-72db-47b7-864d-b3e1fd7b7868 | compute-opt.test.local | up    | enabled |
+--------------------------------------+------------------------+-------+---------+
```

Теперь можно проверить, какие виртуальные машины запущены на каждом из гипервизоров:

```
$ nova hypervisor-servers compute.test.local
+------------------+--------------------+---------------+------------------------+
| ID               | Name               | Hypervisor ID | Hypervisor Hostname    |
+------------------+--------------------+---------------+------------------------+
| 8d08f223-bfce-..| instance-00000008  | 658bd88c-8..  | compute.test.local     |
+------------------+--------------------+---------------+------------------------+
```

```
$ nova hypervisor-servers compute-opt.test.local
+------------------+-------------------+--------------+-----------------------+
| ID               | Name              | Hypervisor ID | Hypervisor Hostname   |
+------------------+-------------------+--------------+-----------------------+
| b7a6fd97-a37e-.. | instance-00000007 | 6989b8db-7..  | compute-opt.test.local |
+------------------+-------------------+--------------+-----------------------+
```

Естественно, такой способ поиска виртуальной машины вручную подходит только для небольших сред. Еще один способ – это использование непосредственно SQL-запросов к базе данных nova, например применяя `uuid` экземпляра виртуальной машины:

```
# mysql -unova -pnova
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 87
Server version: 10.1.20-MariaDB MariaDB Server

Copyright (c) 2000, 2016, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> use nova;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed

MariaDB [nova]> select uuid,host,launched_on,created_at from instances where
uuid='b7a6fd97-a37e-4f1f-98dc-0d182dd45e16';
+--------------------------------------+----------------------+----------------------+---------------------+
| uuid                                 | host                 | launched_on          | created_at          |
+--------------------------------------+----------------------+----------------------+---------------------+
| b7a6fd97-a37e-4f1f-98dc-0d182dd45e16 | compute-opt.test.local | compute-opt.test.local | 2018-03-05 18:48:05 |
+--------------------------------------+----------------------+----------------------+---------------------+
1 row in set (0.00 sec)
```

Еще одна команда позволяет получить суммарную статистику для каждого проекта:

```
$ nova usage-list
Usage from 2016-12-11 to 2017-01-09:
+-----------+---------+--------------+-----------+---------------+
| Tenant ID | Servers | RAM MB-Hours | CPU Hours | Disk GB-Hours |
+-----------+---------+--------------+-----------+---------------+
| 9c1258e.. | 10      | 159016.61    | 525.59    | 559.63        |
+-----------+---------+--------------+-----------+---------------+
```

Добавляем к экземпляру виртуальной машины сеть

Говоря о сети, начнем с понятия групп безопасности. Группа безопасности – это набор правил брандмауэра, разрешающих доступ к тем или иным портам виртуальной машины. По умолчанию в каждом проекте существует одна группа безопасности default, не содержащая ни одного разрешающего правила для входящего трафика. Можно добавить правила в уже существующую группу, но мы создадим свою. Работать будем в проекте demo пользователем demo:

```
$ openstack security group create demo-sgroup
+-----------------+-------------------------------------------------------------------------------
--------------------------------------------------------------------------------+
| Field           | Value
    |
+-----------------+-------------------------------------------------------------------------------
--------------------------------------------------------------------------------+
| created_at      | 2018-03-05T19:54:30Z
    |
| description     | demo-sgroup
    |
| id              | a9064626-bdc1-44b0-a4f5-9f4d5f0c2b44
    |
| name            | demo-sgroup
    |
| project_id      | bc10ac4b71164550a363b8098e8ad270
    |
| revision_number | 2
    |
| rules           | created_at='2018-03-05T19:54:30Z', direction='egress', ethertype='IPv4',
id='5d4d520c-6430-4204-9d61-29e5f5f870a5', updated_at='2018-03-05T19:54:30Z' |
|                 | created_at='2018-03-05T19:54:30Z', direction='egress', ethertype='IPv6',
id='af8340ff-2f23-462e-98b1-99cee55803c7', updated_at='2018-03-05T19:54:30Z' |
| updated_at      | 2018-03-05T19:54:30Z
    |
+-----------------+-------------------------------------------------------------------------------
--------------------------------------------------------------------------------+
```

После этого добавим одно правило, разрешающее доступ по ssh:

```
$ openstack security group rule create --protocol tcp --dst-port
22 demo-sgroup
+-------------------+--------------------------------------+
| Field             | Value                                |
+-------------------+--------------------------------------+
```

```
| created_at        | 2018-03-05T19:55:14Z                 |
| description       |                                      |
| direction         | ingress                              |
| ether_type        | IPv4                                 |
| id                | ad2e9953-76bf-46a0-ae53-06afffceaf83 |
| name              | None                                 |
| port_range_max    | 22                                   |
| port_range_min    | 22                                   |
| project_id        | bc10ac4b71164550a363b8098e8ad270     |
| protocol          | tcp                                  |
| remote_group_id   | None                                 |
| remote_ip_prefix  | 0.0.0.0/0                            |
| revision_number   | 0                                    |
| security_group_id | a9064626-bdc1-44b0-a4f5-9f4d5f0c2b44 |
| updated_at        | 2018-03-05T19:55:14Z                 |
+-------------------+--------------------------------------+
```

И второе, разрешающее протокол ICMP:

```
$ openstack security group rule create --protocol icmp  demo-sgroup
+-------------------+--------------------------------------+
| Field             | Value                                |
+-------------------+--------------------------------------+
| created_at        | 2018-03-06T21:09:43Z                 |
| description       |                                      |
| direction         | ingress                              |
| ether_type        | IPv4                                 |
| id                | 65ed89c2-49e9-4a2c-888b-597ec2f17783 |
| name              | None                                 |
| port_range_max    | None                                 |
| port_range_min    | None                                 |
| project_id        | bc10ac4b71164550a363b8098e8ad270     |
| protocol          | icmp                                 |
| remote_group_id   | None                                 |
| remote_ip_prefix  | 0.0.0.0/0                            |
| revision_number   | 0                                    |
| security_group_id | a9064626-bdc1-44b0-a4f5-9f4d5f0c2b44 |
| updated_at        | 2018-03-06T21:09:43Z                 |
+-------------------+--------------------------------------+
```

Посмотреть список групп безопасности можно командой `nova secgroup-list` или `openstack security group list`:

```
$ openstack security group list
+--------------------------------------+------------+-----------------------+------
--------------------------+
| ID                                   | Name       | Description           | Proje
ct                       |
+--------------------------------------+------------+-----------------------+------
--------------------------+
```

```
| 22eedc5e-6901-4b5d-a0f5-d04543a05305 | default | Default security group | bc10a
c4b71164550a363b8098e8ad270 |
| a9064626-bdc1-44b0-a4f5-9f4d5f0c2b44 | demo-sgroup | demo-sgroup            | bc10a
c4b71164550a363b8098e8ad270 |
+--------------------------------------+-------------+------------------------+------
----------------------------+
```

Команда `nova secgroup-list-rules` или `openstack security group rule list` выводит список правил конкретной группы:

```
$ openstack security group rule list demo-sgroup
+--------------------------------------+-------------+-----------+-----------+-----------------------+
| ID                                   | IP Protocol | IP Range  | Port Range | Remote Security Group |
+--------------------------------------+-------------+-----------+-----------+-----------------------+
| 5d4d520c-6430-4204-9d61-29e5f5f870a5 | None        | None      |           | None                  |
| 65ed89c2-49e9-4a2c-888b-597ec2f17783 | icmp        | 0.0.0.0/0 |           | None                  |
| ad2e9953-76bf-46a0-ae53-06afffceaf83 | tcp         | 0.0.0.0/0 | 22:22     | None                  |
| af8340ff-2f23-462e-98b1-99cee55803c7 | None        | None      |           | None                  |
+--------------------------------------+-------------+-----------+-----------+-----------------------+
```

Теперь вернемся к успешно стартовавшей виртуальной машине myinstance1. Группы безопасности можно добавлять и удалять динамически. Добавим группу demo-sgroup и удалим группу безопасности «по умолчанию» default:

```
$ openstack server add security group myinstance1 demo-sgroup
$ openstack server remove security group myinstance1 default
```

Проверим, что группа безопасности изменилась:

```
$ openstack server show myinstance1 | grep security_groups
| security_groups              | name='demo-sgroup'        |
```

При помощи ключа команды `nova boot` или `openstack` также можно задавать группу безопасности при старте виртуальной машины:

```
$ nova boot --flavor m2.tiny --image cirros-0.4.0-x86_64
--security-groups demo-sgroup --key-name demokey1 myinstance1
$ openstack server create --image cirros-0.4.0-x86_64 --flavor
m2.tiny --security-groups demo-sgroup --key-name demokey1
myinstance1
```

Попробуем подключиться к виртуальной машине по сети. Правило в созданной нами группе безопасности должно позволить подключение по протоколу ssh, а также у нас есть закрытый ключ,

парный которому, открытый, должен был импортироваться при старте виртуальной машины. Единственное, чего у нас не хватает, – это внешнего «плавающего» IP. Подключиться «снаружи» к внутренней сети demo-net мы не сможем. Создадим один внешний IP, который нам будет выделен из сети ext-net:

```
$ openstack floating ip create ext-net
+---------------------+--------------------------------------+
| Field               | Value                                |
+---------------------+--------------------------------------+
| created_at          | 2018-03-06T21:11:48Z                 |
| description         |                                      |
| fixed_ip_address    | None                                 |
| floating_ip_address | 10.100.1.109                         |
| floating_network_id | 1d25a0b0-f1a4-49c2-9388-bed695c11267 |
| id                  | 87a41c7d-ee46-4c22-bed7-b40b858872f2 |
| name                | 10.100.1.109                         |
| port_id             | None                                 |
| project_id          | bc10ac4b71164550a363b8098e8ad270     |
| qos_policy_id       | None                                 |
| revision_number     | 0                                    |
| router_id           | None                                 |
| status              | DOWN                                 |
| subnet_id           | None                                 |
| updated_at          | 2018-03-06T21:11:48Z                 |
+---------------------+--------------------------------------+
```

Нам нужно ассоциировать сетевой порт виртуальной машины с выделенным IP 10.100.1.103. Отдаем команду:

```
$ openstack server add floating ip myinstance1 10.100.1.103
```

Затем при помощи `nova list` или `openstack server list` проверяем, появился ли «плавающий» IP в сетях:

```
$ openstack server list
+------+------------+--------+-------------------------+--------------------+---------+
| ID   | Name       | Status | Networks                | Image              | Flavor  |
+------+------------+--------+-------------------------+--------------------+---------+
| 10e.. | myinstance1 | ACTIVE | demo-net=172.16.0.5,   |                    |         |
|      |            |        |          10.100.1.109   | cirros-0.4.0-x86_64 | m2.tiny |
+------+------------+--------+-------------------------+--------------------+---------+
```

Работу с веб-панелью управления OpenStack мы рассмотрим только в одной из следующих глав, но, для того чтобы проиллюстрировать получившуюся конфигурацию нашей сети, я приведу снимки экранов сетевой топологии в данном разделе. В версии

Newton разработчики объединили в Horizon оба вида графических представлений сети. На рис. 9.4 приведена сетевая топология Horizon, как она выглядела в версии OpenStack Kilo и более ранних. На рис. 9.5 – то, как выглядела топология в версии начиная с OpenStack Liberty.

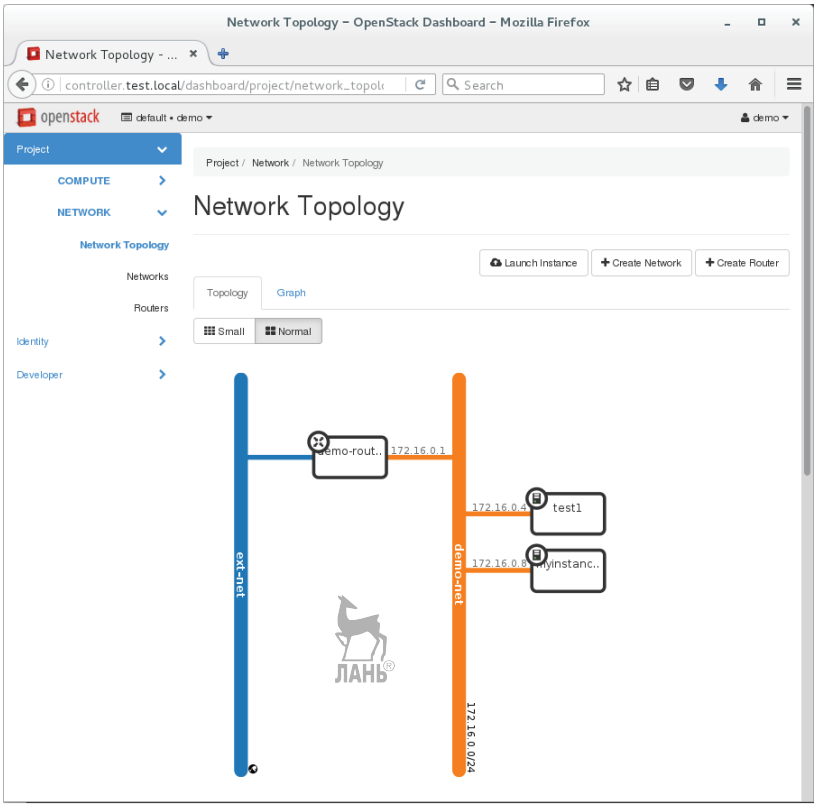


Рис. 9.4. Сетевая топология в Horizon

Последнее, что осталось проверить, – с тестовой машины, подключенной в ту же сеть, что и сетевой интерфейс eth1 виртуальной машины network, попробовать подключиться к виртуальной машине. В случае использования дистрибутива cirros подключаться надо пользователем cirros:

```
[user@test ~]$ ssh -i ~/demokey1 cirros@10.100.1.109
```

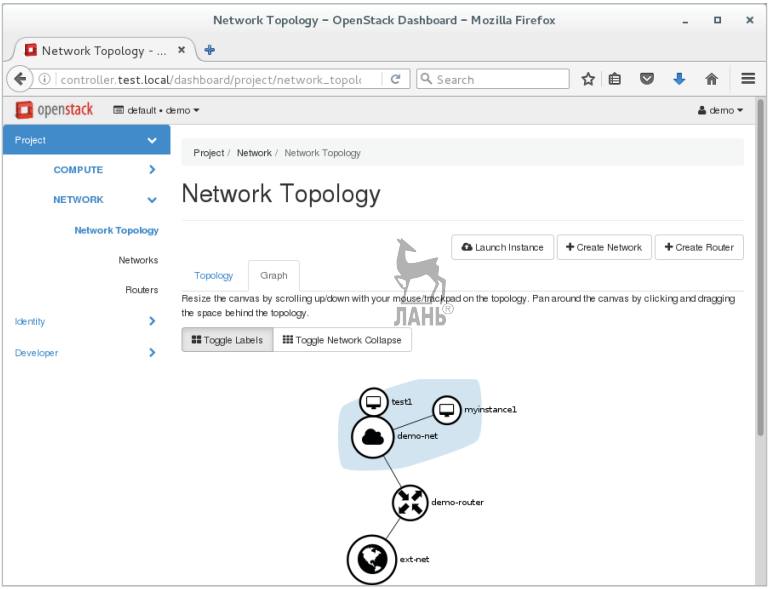


Рис. 9.5. Сетевая топология в виде графов

Моментальные снимки и резервные копии

Как мы помним из главы, посвященной сервису работы с блочными устройствами, у нас есть том с именем testvol1 и размером 1 Гб:

```
$ openstack volume list
+--------------------------------------+----------+-----------+------+-------------+
| ID                                   | Name     | Status    | Size | Attached to |
+--------------------------------------+----------+-----------+------+-------------+
| 11bddba4-ed69-4aa3-b5a9-6dd47e19d49a | testvol1 | available |    1 |             |
+--------------------------------------+----------+-----------+------+-------------+
```

Мы можем подключить его к запущенному экземпляру виртуальной машины myinstance1 и начать использовать «изнутри» машины:

```
$ openstack server add volume myinstance1 testvol1
```

Нужно дождаться, когда в выводе команды `openstack volume list` статус тома изменится на «in-use»:

```
$ openstack volume list
+--------------------------------------+---------+--------+------+--------------------------------------+
| ID                                   | Name    | Status | Size | Attached to                          |
+--------------------------------------+---------+--------+------+--------------------------------------+
| 11bddba4-ed69-4aa3-b5a9-6dd47e19d49a | testvol1 | in-use |    1 | Attached to myinstance1 on /dev/vdb |
+--------------------------------------+---------+--------+------+--------------------------------------+
```

После этого можно проверить, что устройство появилось в виртуальной машине:

```
$  cat /proc/partitions
major minor  #blocks  name

 253        0    1048576 vda
 253        1    1036192 vda1
 253       16    1048576 vdb
```

Дальше можно стандартными средствами операционной системы создать раздел и файловую систему на нем. Отключить его можно командой `nova volume-detach`. Предварительно не забудьте размонтировать устройство в виртуальной машине.

Также можно подключать блочное устройство cinder во время создания виртуальной машины командой `nova boot` при помощи опции `--block-device-mapping`.

Средствами OpenStack можно создать снимок работающей виртуальной машины, например для целей резервного копирования или для создания нового измененного базового образа. Важно отметить, что снимок не является точкой восстановления для существующей виртуальной машины. Снимок – это такой же образ, как и те, что вы загружаете в Glance. Основное различие – это несколько дополнительных полей в базе данных, например UUID виртуальной машины, из которой получен этот образ.

Сделаем какое-нибудь простое и легко проверяемое изменение на файловой системе, запущенной из машины myinstance1:

```
$ sudo cp /etc/os-release /etc/my-os-release
```

Создадим снимок с именем myinstance1_sn1:

```
$ nova image-create myinstance1 myinstance1_sn1
```

Для нашего тестового образа cirros создание снимка произойдет в считанные секунды. Для образов большого размера это займет некоторое время. Дождемся, пока статус снимка станет «ACTIVE»:

```
$ nova image-list
+--------------------------------------+---------------------+--------+---------+
| ID                                   | Name                | Status | Server  |
+--------------------------------------+---------------------+--------+---------+
| 4204660a-dc38-4430-9a68-1fa61e542522 | cirros-0.3.4-x86_64 | ACTIVE |         |
| ab8dedb7-2ec3-4aa9-978c-d081eb18e7ac | myinstance1_sn1     | ACTIVE | c70bd.. |
+--------------------------------------+---------------------+--------+---------+
```

После можно запустить новую виртуальную машину:

```
$ nova boot --flavor m2.tiny --image myinstance1_sn1 --key-name
demokey1 --security-groups demo-sgroup myinstance2
```

Если получить доступ к консоли myinstance2, то мы увидим, что изменения присутствуют в новом экземпляре:

```
$ ls /etc/my*
/etc/my-os-release
```

В том случае, если вы работаете с виртуальными машинами Fedora, CentOS, RHEL или им подобными, необходимо перед созданием снимка удалить правило udev для сети, иначе новый экземпляр не сможет создать сетевого интерфейса:

```
# rm -f /etc/udev/rules.d/70-net-persistent.rule
```

Также если виртуальная машина запущена во время создания снимка, необходимо сбросить на диск содержимое всех файловых буферов и исключить дисковый ввод/вывод каких-либо процессов. Данные команды сбросят буферы и «заморозят» доступ к файловой системе на 60 секунд:

```
# sync && fsfreeze -f / && sleep 60 && fsfreeze -u /
```

Помимо моментальных снимков, имеется функционал создания резервных копий. Основные отличия от снимков:

- резервная копия сохраняется в объектное хранилище, а не в сервис работы с образами;
- резервная копия занимает столько места, сколько занимают данные, а моментальный снимок копирует весь образ полностью;
- резервную копию можно восстановить на новый том, а из моментального снимка можно запустить новый экземпляр виртуальной машины;
- в отличие от создания моментального снимка, нельзя делать резервную копию используемого тома.

Пример создания резервной копии тома:

```
$ cinder backup-create 82711b99-cce1-45e4-8b7e-b91301d65dc8
+-----------+--------------------------------------+
|  Property |                Value                 |
+-----------+--------------------------------------+
|     id    | 66a1b5b7-4167-4a1d-b2cd-c565019cfb76 |
|    name   |                 None                 |
| volume_id | 82711b99-cce1-45e4-8b7e-b91301d65dc8 |
+-----------+--------------------------------------+
```

Если теперь посмотреть содержимое контейнера volumebackups, то мы увидим там множество объектов. Идентифицировать резервную копию можно по идентификатору тома и времени создания резервной копии:

```
$ swift list volumebackups
volume_82711b99.../20150625081540/az_nova_backup_66a1b5b7-...-00001
volume_82711b99.../20150625081540/az_nova_backup_66a1b5b7-...-00002
...
```

При восстановлении необходимо уже указывать не ID тома, а идентификатор резервной копии. В нашем случае:

```
$ cinder backup-restore 66a1b5b7-4167-4a1d-b2cd-c565019cfb76
```

Если мы посмотрим вывод команды cinder list, то увидим, что резервная копия восстанавливается в новый том с именем restore_backup_66a1b5b7-4167-4a1d-b2cd-c565019cfb76:

```
$ cinder list
+-------+-----------------+-----------------------+------+-------------+----------+-------------+
|  ID   |      Status     |      Display Name     | Size | Volume Type | Bootable | Attached to |
+-------+-----------------+-----------------------+------+-------------+----------+-------------+
| 82... |    available    |        testvol1       |  1   |     None    |  false   |             |
| d3... | restoring-backup | restore_backup_66a1...|  1   |     None    |  false   |             |
+-------+-----------------+-----------------------+------+-------------+----------+-------------+
```

Инкрементальные резервные копии в настоящий момент поддерживаются, только если вы используете в качестве бэкэнда Ceph, и эта реализация специфична именно для Ceph. Начиная с релиза Kilo инкрементальные резервные копии реализованы как общая, неспецифичная функция (для Ceph функционал существовал, начиная с Havana), но большинство драйверов на момент написания главы ее не поддерживало.

Шифрование томов Cinder

Одной из полезных опций работы с блочными устройствами является их шифрование. Настройка шифрования томов требуется со стороны двух служб: Nova и Cinder. Сделать это можно при помощи общего секрета или при помощи сервиса управления ключами Barbican, который не рассматривается в этой книге. Соответственно, пойдем по пути использования общего секрета. Нужно иметь в виду, что если он скомпрометирован, то злоумышленник получит доступ ко всем зашифрованным томам.

Зададим ключ на узле Cinder и всех вычислительных узлах:

```
[root@compute ~]# crudini --set /etc/nova/nova.conf keymgr fixed_key
123456789
[root@compute ~]# systemctl restart openstack-nova-compute
[root@controller ~]# crudini --set /etc/cinder/cinder.conf keymgr
fixed_key 123456789
[root@controller ~]# systemctl restart openstack-cinder-volume
```

Нам необходимо создать новый тип тома. Назовем его LUKS, поскольку для шифрования будет использоваться соответствующая спецификация (https://en.wikipedia.org/wiki/Linux_Unified_Key_Setup):

```
$ source keystonerc_admin
$ cinder type-create LUKS
+--------------------------------------+------+-------------+-----------+
| ID                                   | Name | Description | Is_Public |
+--------------------------------------+------+-------------+-----------+
| 14bbc318-3377-4ab3-a4f0-e1738eec8dfd | LUKS | -           | True      |
+--------------------------------------+------+-------------+-----------+
```

Следующим шагом нужно создать тип шифрования:

```
$ cinder encryption-type-create --cipher aes-xts-plain64 --key_size 512 --control_
location front-end LUKS nova.encryptors.luks.LuksEncryptor
+---------------+---------------------------------+---------------+----------+------------------+
| Volume Type ID |            Provider             |     Cipher    | Key Size | Control Location |
+---------------+---------------------------------+---------------+----------+------------------+
| 14bbc318-337.. | nova.encryptors.luks.LuksEncryptor | aes-xts-plain64 |   512    |    front-end     |
+---------------+---------------------------------+---------------+----------+------------------+
```

Теперь у нас все готово для создания зашифрованного тома. Обратите внимание на свойство encrypted при выводе команды:

```
$ source keystonerc_demo
$ cinder create --display-name myvolumeEncr --volume-type LUKS 1
```

```
+------------------------------+--------------------------------------+
| Property                     | Value                                |
+------------------------------+--------------------------------------+
| attachments                  | []                                   |
| availability_zone            | nova                                 |
| bootable                     | false                                |
| consistencygroup_id          | None                                 |
| created_at                   | 2016-12-18T14:19:56.000000           |
| description                  | None                                 |
| encrypted                    | True                                 |
| id                           | ec5612de-0116-482d-b69b-78d73138b7e9 |
| metadata                     | {}                                   |
| multiattach                  | False                                |
| name                         | myvolumeEncr                         |
| os-vol-tenant-attr:tenant_id | 9c1258e169d54a35964c4a28c380fbc3     |
| replication_status           | disabled                             |
| size                         | 1                                    |
| snapshot_id                  | None                                 |
| source_volid                 | None                                 |
| status                       | creating                             |
| updated_at                   | None                                 |
| user_id                      | 372dd23e2c6949169a7dd9b699a2bc6b     |
| volume_type                  | LUKS                                 |
+------------------------------+--------------------------------------+
```

Квоты на ресурсы

В контексте управления ресурсами и виртуальными машинами нельзя не упомянуть про квоты. Квоты можно задавать как для вычислительных ресурсов, так и для дискового пространства. Чаще всего квоты задаются в разрезе проектов, но можно их устанавливать и для пользователей. Посмотрим квоты по умолчанию:

```
$ nova quota-defaults
+-----------------------------+-------+
| Quota                       | Limit |
+-----------------------------+-------+
| instances                   | 10    |
| cores                       | 20    |
| ram                         | 51200 |
| metadata_items              | 128   |
| injected_files              | 5     |
| injected_file_content_bytes | 10240 |
| injected_file_path_bytes    | 255   |
| key_pairs                   | 100   |
| server_groups               | 10    |
| server_group_members        | 10    |
+-----------------------------+-------+
```

Из вывода команды видно, какие ограничения можно устанавливать. Для работы с отдельным проектом можно использовать подкоманды `quota-update` и `quota-show`.

Для cinder, neutron, swift и остальных сервисов нужно использовать соответствующие команды. Например:

```
$ cinder quota-show demo
+----------------------+-------+
| Property             | Value |
+----------------------+-------+
| backup_gigabytes     | 1000  |
| backups              | 10    |
| gigabytes            | 1000  |
| gigabytes_LUKS       | -1    |
| per_volume_gigabytes | -1    |
| snapshots            | 10    |
| snapshots_LUKS       | -1    |
| volumes              | 10    |
| volumes_LUKS         | -1    |
+----------------------+-------+
```

Нули означают, что квоты не заданы. Можно посмотреть текущее использование квоты:

```
$ cinder quota-usage demo
+----------------------+--------+----------+-------+
| Type                 | In_use | Reserved | Limit |
+----------------------+--------+----------+-------+
| backup_gigabytes     | 0      | 0        | 1000  |
| backups              | 0      | 0        | 10    |
| gigabytes            | 1      | 0        | 1000  |
| gigabytes_LUKS       | 0      | 0        | -1    |
| per_volume_gigabytes | 0      | 0        | -1    |
| snapshots            | 1      | 0        | 10    |
| snapshots_LUKS       | 0      | 0        | -1    |
| volumes              | 1      | 0        | 10    |
| volumes_LUKS         | 0      | 0        | -1    |
+----------------------+--------+----------+-------+
```

Ниже приведен пример для Neutron:

```
$ neutron quota-list
+------------+---------+------+-------------+--------+----------
------+---------------------+--------+------------+-----------+
| floatingip | network | port | rbac_policy | router | security_
group | security_group_rule | subnet | subnetpool | tenant_id |
+------------+---------+------+-------------+--------+----------
------+---------------------+--------+------------+-----------+
|         10 |      10 |   50 |          10 |     10 |
```

```
10 |                  100 |     10 |          -1 | 9c1258..  |
+------------+---------+------+-------------+--------+----------
------+---------------------+--------+------------+-----------+
```

Описание команд работы с квотами можно посмотреть при помощи команды `help`.

Зоны доступности и агрегирование вычислительных узлов в Nova

Еще два понятия, которые необходимо обсудить, – это зоны доступности (Availabaility Zones) и агрегаторы узлов (Host Aggregates). На самом деле в Nova все зоны доступности – это подмножество агрегаторов. Но не будем забегать вперед, а пойдем по порядку.

Зоны доступности позволяют группировать узлы OpenStack в логические группы с общим элементом доступности. Например, это могут быть две независимые площадки с раздельным питанием и доступом в Интернет, или это могут быть просто две стойки.

По умолчанию все вычислительные узлы создаются в зоне nova. Прежде чем продолжить разговор про зоны, разберемся с агрегацией узлов. Агрегаторы узлов – это также логическое объединение узлов, но с привязкой дополнительных метаданных. Агрегаторы позволяют группировать узлы по различным специфическим признакам. Посмотрим, как их можно использовать:

```
$ source keystonerc_admin
$ nova aggregate-create MyAggregate
+----+-------------+-------------------+-------+----------+
| Id | Name        | Availability Zone | Hosts | Metadata |
+----+-------------+-------------------+-------+----------+
| 1  | MyAggregate | -                 |       |          |
+----+-------------+-------------------+-------+----------+
```

Сейчас мы просто создали один агрегатор. Следующим шагом добавим в него один из наших вычислительных узлов:

```
$ nova aggregate-add-host MyAggregate compute.test.local
Host compute.test.local has been successfully added for aggregate 1
+----+-------------+-------------------+---------------------+----------+
| Id | Name        | Availability Zone | Hosts               | Metadata |
+----+-------------+-------------------+---------------------+----------+
| 1  | MyAggregate | -                 | 'compute.test.local' |          |
+----+-------------+-------------------+---------------------+----------+
```

Заметим, что один узел может входить в несколько агрегаторов. Добавим к агрегатору метаданные, которые мы дальше будем использовать для выбора узлов (в нашем случае только одного – compute.test.local) из этого агрегатора:

```
$ nova aggregate-set-metadata MyAggregate mynewmetadata=true
Metadata has been successfully updated for aggregate 1.
+----+-------------+------------+----------------------+----------------------+
| Id | Name        | Av... Zone | Hosts                | Metadata             |
+----+-------------+------------+----------------------+----------------------+
| 1  | MyAggregate | -          | 'compute.test.local' | 'mynewmetadata=true' |
+----+-------------+------------+----------------------+----------------------+
```

Как разбирали в разделе «Запускаем экземпляр виртуальной машины», создадим новый тип (flavor) виртуальной машины:

```
$ nova flavor-create --is-public true m2.mynewmdvm auto 300 1 1
+------+-------------+-----------+------+-----+------+-------+-------------+-----------+
| ID   | Name        | Memory_MB | Disk | Eph | Swap | VCPUs | RXTX_Factor | Is_Public |
+------+-------------+-----------+------+-----+------+-------+-------------+-----------+
| 64.. | m2.mynewmdvm | 300       | 1    | 0   |      | 1     | 1.0         | True      |
+------+-------------+-----------+------+-----+------+-------+-------------+-----------+
```

Добавим к созданному типу метаданные, по наличию которых при попытке запуска виртуальной машины этого типа она будет стартовать в агрегаторе MyAggregate:

```
$ nova flavor-key m2.mynewmdvm set mynewmetadata=true
```

Проверим данные:

```
$ nova flavor-show m2.mynewmdvm
+----------------------------+------------------------------+
| Property                   | Value                        |
+----------------------------+------------------------------+
| OS-FLV-DISABLED:disabled   | False                        |
| OS-FLV-EXT-DATA:ephemeral  | 0                            |
| disk                       | 1                            |
| extra_specs                | {"mynewmetadata": "true"}    |
| id                         | 647fe1cc-9eff-4405-9b9e-..   |
| name                       | m2.mynewmdvm                 |
| os-flavor-access:is_public | True                         |
| ram                        | 300                          |
| rxtx_factor                | 1.0                          |
| swap                       |                              |
| vcpus                      | 1                            |
+----------------------------+------------------------------+
```

Теперь при запуске виртуальной машины с flavor m2.mynewmdvm такая машина будет стартовать на узле compute.test.local.

Вернемся к зонам доступности. Агрегация узлов может быть представлена для пользователей в виде зоны доступности. Фактически агрегатор и есть зона доступности, необходимо только дать ему соответствующее имя. Посмотрим список зон, доступных пользователям в настоящее время:

```
$ source keystonerc_demo
$ nova availability-zone-list
+------+-----------+
| Name | Status    |
+------+-----------+
| nova | available |
+------+-----------+
```

Создадим агрегатор узлов, дав ему имя как зоне доступности:

```
$ source keystonerc_admin
$ nova aggregate-create MyAggregateZ MyZone
+----+-------------+------------------+-------+---------------------------+
| Id | Name        | Availability Zone | Hosts | Metadata                  |
+----+-------------+------------------+-------+---------------------------+
| 2  | MyAggregateZ | MyZone           |       | 'availability_zone=MyZone' |
+----+-------------+------------------+-------+---------------------------+
```

Добавим второй оставшийся узел в созданный агрегатор:

```
$ nova aggregate-add-host MyAggregateZ compute-opt.test.local
Host compute-opt.test.local has been successfully added for aggregate 2
+----+-------------+----------+-------------------------+--------------------+
| Id | Name        | Av.. Zone | Hosts                   | Metadata           |
+----+-------------+----------+-------------------------+--------------------+
| 2  | MyAggregateZ | MyZone   | 'compute-opt.test.local' | 'availability_zone |
|    |             |          |                         |           =MyZone' |
+----+-------------+----------+-------------------------+--------------------+
```

Теперь проверим видимые пользователю зоны доступности:

```
$ nova availability-zone-list
+-------------------------+----------------------------------------+
| Name                    | Status                                 |
+-------------------------+----------------------------------------+
| internal                | available                              |
| |- controller.test.local |                                        |
| | |- nova-conductor       | enabled :-) 2016-11-21T17:45:38.000000 |
| | |- nova-consoleauth     | enabled :-) 2016-11-21T17:45:38.000000 |
| | |- nova-scheduler       | enabled :-) 2016-11-21T17:45:39.000000 |
| nova                    | available                              |
| |- compute.test.local    |                                        |
| | |- nova-compute         | enabled :-) 2016-11-21T17:45:37.000000 |
| MyZone                  | available                              |
```

```
| |- compute-opt.test.local |                                          |
| | |- nova-compute         | enabled :-) 2016-11-21T17:45:37.000000 |
+---------------------------+------------------------------------------+
```

и запустим виртуальную машину с указанием имени зоны MyZone. Машина будет запущена на узлах, входящих в агрегатор MyAggregateZ. В нашем случае это – единственный узел compute-opt.test.local.

```
$ nova boot --flavor m2.tiny --image cirros-0.3.4-x86_64 --key-name
demokey1 --security-groups demo-sgroup --availability-zone MyZone
test1
```

На рис. 9.6 показан интерфейс работы с зонами и агрегаторами в Horizon.

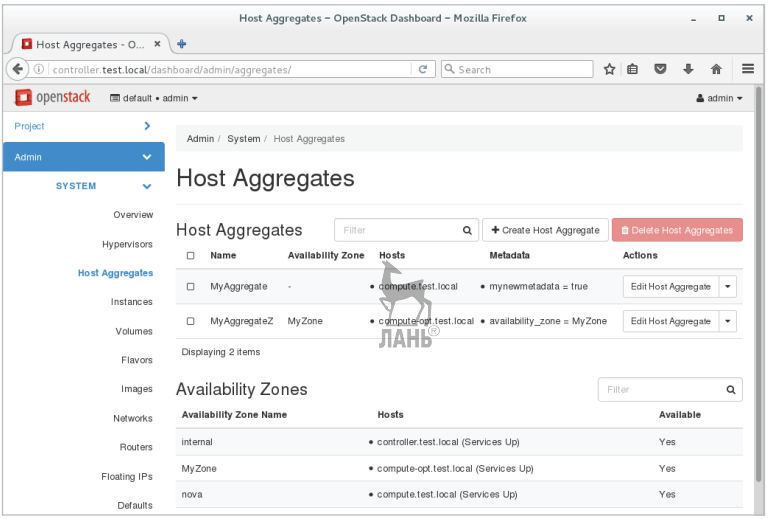


Рис. 9.6. Зоны доступности и агрегаторы узлов в веб-интерфейсе

Если необходимо в дальнейшем установить зону доступности для виртуальных машин по умолчанию, то это можно сделать при помощи параметра default_availability_zone в /etc/nova/nova.conf на управляющих узлах.

Удалим узлы из агрегаторов:

```
$ nova aggregate-remove-host MyAggregate compute.test.local
Host compute.test.local has been successfully removed from aggregate 1
+----+-------------+-------------------+-------+----------------------+
| Id | Name        | Availability Zone | Hosts | Metadata             |
+----+-------------+-------------------+-------+----------------------+
```

```
| 1  | MyAggregate | -                 |       | 'mynewmetadata=true' |
+----+-------------+-------------------+-------+----------------------+
$ nova aggregate-remove-host MyAggregateZ compute-opt.test.local
Host compute-opt.test.local has been successfully removed from aggregate 2
+----+-------------+-------------------+-------+--------------------------+
| Id | Name        | Availability Zone | Hosts | Metadata                 |
+----+-------------+-------------------+-------+--------------------------+
| 2  | MyAggregateZ | MyZone           |       | 'availability_zone=MyZone' |
+----+-------------+-------------------+-------+--------------------------+
```

После чего можно будет удалить сами агрегаторы:

```
$ nova aggregate-delete MyAggregate
Aggregate 1 has been successfully deleted.
$ nova aggregate-delete MyAggregateZ
Aggregate 2 has been successfully deleted.
```

Проверим изменения в зонах доступности:

```
$ nova availability-zone-list
+--------------------------+----------------------------------------+
| Name                     | Status                                 |
+--------------------------+----------------------------------------+
| internal                 | available                              |
| |- controller.test.local |                                        |
| | |- nova-conductor      | enabled :-) 2016-12-17T15:17:31.000000 |
| | |- nova-consoleauth    | enabled :-) 2016-12-17T15:17:32.000000 |
| | |- nova-scheduler      | enabled :-) 2016-12-17T15:17:36.000000 |
| nova                     | available                              |
| |- compute-opt.test.local |                                        |
| | |- nova-compute        | enabled :-) 2016-12-17T15:17:36.000000 |
| |- compute.test.local    |                                        |
| | |- nova-compute        | enabled :-) 2016-12-17T15:17:29.000000 |
+--------------------------+----------------------------------------+
```

Далее мы рассмотрим использование зон доступности в Cinder. В релизе Mitaka также появились зоны доступности Neutron, но в данной книге они не рассматриваются.

Зоны доступности в Cinder

Зоны доступности Cinder настраиваются в конфигурационном файле /etc/cinder/cinder.conf на узлах, где запускается сервис cinder-volume. Для создания новой зоны доступности необходимо изменить параметр storage_availability_zone, после чего рестартовать сервис openstack-cinder-volume:

```
[root@controller ~]# crudini --set /etc/cinder/cinder.conf DEFAULT storage_availability_zone cinder1
[root@controller ~]# systemctl restart openstack-cinder-volume
```

```
$ cinder service-list
+------------------+--------------------------+---------+---------+-------+-------------+----------+
| Binary           | Host                     | Zone    | Status  | State | Updated_at  | Disabled |
+------------------+--------------------------+---------+---------+-------+-------------+----------+
| cinder-backup    | controller.test.local    | nova    | enabled | up    | 2017-01-29.. | -        |
| cinder-scheduler | controller.test.local    | nova    | enabled | up    | 2017-01-29.. | -        |
| cinder-volume    | controller.test.local@lvm | cinder1 | enabled | up    | 2017-01-29.. | -        |
+------------------+--------------------------+---------+---------+-------+-------------+----------+
```

Для того чтобы установить зону доступности по умолчанию, необходимо править /etc/cinder/cinder.conf на узле, где запускается сервис cinder-api. В нашем случае этот узел – также controller:

```
[root@controller ~]# crudini --set /etc/cinder/cinder.conf
DEFAULT default_availability_zone cinder1
[root@controller ~]# systemctl restart openstack-cinder-api
```

Для одного сервиса cinder-volume можно указать несколько бэкэндов. Это возможно сделать в рамках одного узла и сервиса cinder-volume. Однако за разные зоны доступности должны отвечать разные узлы.

Живая миграция виртуальных машин

Рассмотрим на практике, как работает живая миграция виртуальных машин в OpenStack. Сразу оговоримся, что для инициации вам понадобятся привилегии администратора облака, поскольку для пользователя информация об облаке скрыта, в том числе и о конкретных гипервизорах, на которых запускаются виртуальные машины. Различные балансировки нагрузки и миграции в OpenStack – вне области ответственности пользователя.

Сервис OpenStack Nova поддерживает живую миграцию виртуальных машин в двух вариантах:

- **с общей системой хранения данных**. Виртуальная машина перемещается между двумя вычислительными узлами с общим хранилищем, к которым оба узла имеют доступ. В качестве общего хранилища может выступать, например, NFS или Ceph. Сюда же можно отнести вариант, когда не используются временные диски (ephemeral disk), а в качестве единственной системы хранения при создании виртуальных машин используется Cinder;
- **без общей системы хранения данны**х. Более простой в настройке вариант, который мы рассмотрим далее. В этом слу-

чае на миграцию требуется больше времени, поскольку виртуальная машина копируется целиком с узла на узел по сети.

Для выполнения упражнения из этого раздела вам понадобятся два работающих гипервизора (рис. 9.7). Начните с проверки IP-связанности между вычислительными узлами:

```
[root@compute ~]# ping compute-opt
PING compute-opt.test.local (192.168.122.215) 56(84) bytes of data.
64 bytes from compute-opt.test.local (192.168.122.215): icmp_seq=1
ttl=64 time=0.302 ms
64 bytes from compute-opt.test.local (192.168.122.215): icmp_seq=2
ttl=64 time=0.363 ms
^C
```

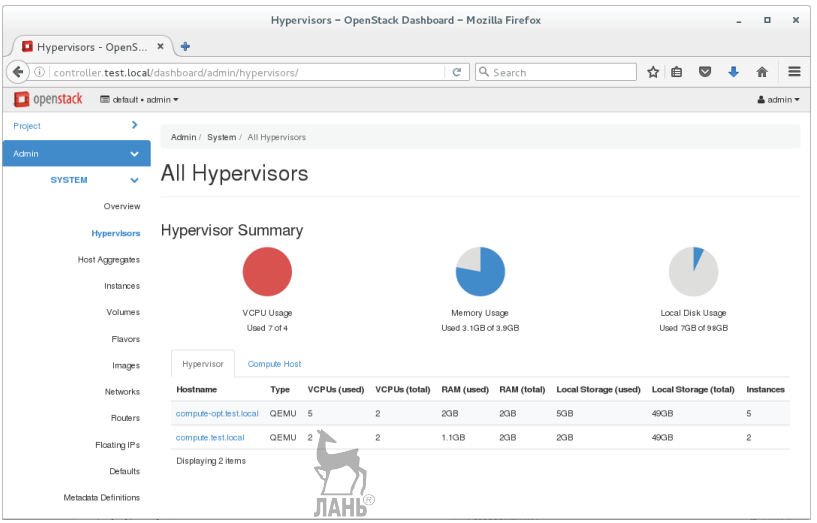


Рис. 9.7. Для миграции виртуальных машин необходимы два гипервизора

Теперь запустим виртуальную машину и при помощи команд, которые рассмотрели в разделе «Выделение вычислительных ресурсов для OpenStack», определим, на котором из узлов она работает.

```
$ nova hypervisor-servers compute.test.local
+----+------+---------------+---------------------+
| ID | Name | Hypervisor ID | Hypervisor Hostname |
+----+------+---------------+---------------------+
+----+------+---------------+---------------------+
$ nova hypervisor-servers compute-opt.test.local
+--------------------------------------+-------------------+----
---------------------------------+-----------------------+
```

```
| ID                                     | Name              |
Hypervisor ID                          | Hypervisor Hostname    |
+----------------------------------------+-------------------+----
-----------------------------------+------------------------+
| 5ac4154e-9f85-46ec-b8fd-27310cd0d10d | instance-0000002c |
6989b8db-72db-47b7-864d-b3e1fd7b7868 | compute-opt.test.local |
+----------------------------------------+-------------------+----
-----------------------------------+------------------------+
```

Мы видим, что виртуальная машина работает на узле compute-opt. Дальше определим, какой flavor использовался для этой виртуальной машины:

```
$ nova show 8nova show 5ac4154e-9f85-46ec-b8fd-27310cd0d10d |
grep flavor:original_name
| flavor:original_name                 | m2.tiny
```

и достаточно ли ресурсов на узле, куда мы хотим мигрировать виртуальную машину:

```
$ nova --os-compute-api-version 2 host-describe compute.test.local
+--------------------+------------+-----+-----------+---------+
| HOST               | PROJECT    | cpu | memory_mb | disk_gb |
+--------------------+------------+-----+-----------+---------+
| compute.test.local | (total)    | 2   | 2000      | 49      |
| compute.test.local | (used_now) | 0   | 512       | 0       |
| compute.test.local | (used_max) | 0   | 0         | 0       |
+--------------------+------------+-----+-----------+---------+
```

Как мы видим, ресурсов достаточно. Однако, прежде чем отдавать команду на миграцию, необходимо разрешить демону libvirtd слушать входящие подключения по сети. На обоих гипервизорах добавим опцию

```
LIBVIRTD_ARGS="--listen"
```

в файл /etc/sysconfig/libvirtd, отвечающую за строку запуска демона. Следующим шагом в конфигурационном файле /etc/libvirt/libvirtd.conf разрешим подключение без аутентификации и шифрования:

```
listen_tls = 0
listen_tcp = 1
auth_tcp = "none"
```

Альтернативой могло бы быть использование сертификатов или Kerberos. Рестартуем `libvirtd` на вычислительных узлах:

```
# systemctl restart libvirtd
```

Убедимся, что сервис работает и имеет необходимую опцию:

```
[root@compute-opt ~]# systemctl status libvirtd
● libvirtd.service - Virtualization daemon
   Loaded: loaded (/usr/lib/systemd/system/libvirtd.service; enabled;
vendor preset: enabled)
   Active: active (running) since Tue 2018-03-20 17:17:14 CET; 8s ago
     Docs: man:libvirtd(8)
           http://libvirt.org
 Main PID: 2493 (libvirtd)
   CGroup: /system.slice/libvirtd.service
           ├─1514 /usr/sbin/dnsmasq --conf-file=/var/lib/libvirt/
dnsmasq/default.conf --leasefile-ro --dhcp-script=/usr/libexec/libvirt_
leaseshelpe...
           ├─1515 /usr/sbin/dnsmasq --conf-file=/var/lib/libvirt/
dnsmasq/default.conf --leasefile-ro --dhcp-script=/usr/libexec/libvirt_
leaseshelpe...
           └─2493 /usr/sbin/libvirtd --listen
```

Последнее, что нужно поправить, – это флаги миграции. Делается это также на всех вычислительных узлах:

```
# crudini --set /etc/nova/nova.conf DEFAULT block_migration_flag
VIR_MIGRATE_UNDEFINE_SOURCE,VIR_MIGRATE_PEER2PEER,VIR_MIGRATE_LIVE
```

Данное изменение необходимо, поскольку флаги по умолчанию включают в себя TUNELLED, который не работает с обновленным кодом NBD (Network Block Device) в QEMU. Для применения изменений необходимо перестартовать сервис nova-compute:

```
# systemctl restart openstack-nova-compute.service
```

Теперь можно отдать команду на живую миграцию, обязательно указав опцию `--block-migrate`, которая отвечает за миграцию без общего дискового хранилища:

```
$ source keystonerc_admin
$ nova live-migration --block-migrate 5ac4154e-9f85-46ec-b8fd-
27310cd0d10d compute.test.local
```

При помощи nova show проверим, на каком узле работает виртуальная машина до и после миграции:

```
$ nova show 5ac4154e-9f85-46ec-b8fd-27310cd0d10d | grep  hypervisor
| OS-EXT-SRV-ATTR:hypervisor_hostname  | compute-opt.test.local   |
$ nova live-migration --block-migrate 5ac4154e-9f85-46ec-b8fd-27310cd0d10d
$ nova show 5ac4154e-9f85-46ec-b8fd-27310cd0d10d | grep  hypervisor
| OS-EXT-SRV-ATTR:hypervisor_hostname  | compute.test.local       |
```

Как мы видим, миграция на этот раз прошла удачно. В журнале /var/log/nova/nova-compute.log, на узле с которого осуществляется миграция, должно появиться сообщение, подобное следующему:

```
2018-03-20 17:23:10.629 2632 INFO nova.compute.manager [req-
b734f151-7aba-459e-8f31-48a696d910e3 03b593ee9de442b985871aea
39eec9b1 c19f44553ab640779672e2321364e6ce - default default]
[instance: 5ac4154e-9f85-46ec-b8fd-27310cd0d10d] Migrating
instance to compute.test.local finished successfully.
```

Во время самого процесса можно следить за ходом ее выполнения средствами OpenStack при помощи команды `nova migration-list` или на узле-источнике при помощи команды `virsh`:

```
[root@compute-opt ~]# virsh domjobinfo instance-00000017
Job type:         Unbounded
Time elapsed:     1084         ms
Data processed:   25.162 MiB
Data remaining:   1.726 GiB
Data total:       2.016 GiB
Memory processed: 25.162 MiB
Memory remaining: 1.726 GiB
Memory total:     2.016 GiB
Memory bandwidth: 240.426 MiB/s
Constant pages:   69765
Normal pages:     6276
Normal data:      24.516 MiB
Expected downtime: 46          ms
Setup time:        2           ms
```

Настройка экземпляров виртуальных машин при помощи cloud-init

При помощи cloud-init экземпляры виртуальных машин при старте могут настраивать различные параметры, такие как имя узла, открытый ssh-ключ для аутентификации, имя хоста и др. Экземпляры виртуальных машин получают эту информацию во время загрузки, обращаясь на адрес агента метаданных Neutron: http://169.254.169.254. Агент метаданных проксирует соответствующие запросы к `openstack-nova-api` при помощи пространства имен маршрутизатора или DHCP.

Конфигурационная информация передается в виртуальную машину при помощи параметров `--user-data` во время запуска. Так-

же метаданные можно передать при старте виртуальной машины из Horizon на вкладке Post-Creation.

Несмотря на то что конфигурация может быть выполнена несколькими способами, включая скрипты, наиболее простым является использование YAML-файлов с синтаксисом cloud-config. В конфигурационном файле описываются модули, исполняемые во время загрузки экземпляра виртуальной машины. Например, один модуль может настроить репозиторий с пакетами, а второй – установить требуемые пакеты. По ссылке https://cloudinit.readthedocs.io/en/latest/topics/examples.html#yaml-examples в документации на cloud-init приведено несколько примеров, а мы ограничимся простейшим экспериментом с установкой имени узла.

Создадим текстовый файл test.txt следующего содержания:

```
#cloud-config
hostname: fedora24
fqdn: fedora24.test.local
manage_etc_hosts: true
```

Далее запустим виртуальную машину из образа Fedora, в котором существует поддержка cloud-init:

```
$ nova boot --flavor m1.small --image fedora-24.x86_64 --key-name
demokey1 --user-data ./test.txt mytestvm
```

Дождемся загрузки и добавим плавающий IP к экземпляру виртуальной машины mytestvm, а после подключимся к консоли пользователем fedora при помощи ключа demokey1:

```
$ ssh -i demokey1.pem fedora@10.100.1.110
[fedora@fedora24 ~]$ hostname
fedora24.test.local
```

Как мы видим, имя узла установлено из нашего файла cloud-init. Наконец, можно убедиться, что виртуальная машина получает этот файл при помощи совместимого с EC2 API:

```
[fedora@fedora24 ~]$ curl http://169.254.169.254/2009-04-04/user-data
#cloud-config
hostname: fedora24
fqdn: fedora24.test.local
manage_etc_hosts: true
[fedora@fedora24 ~]$ curl http://169.254.169.254/latest/meta-data/
ami-id
```

```
ami-launch-index
ami-manifest-path
block-device-mapping/
hostname
instance-action
instance-id
instance-type
local-hostname
local-ipv4
placement/
public-hostname
public-ipv4
public-keys/
reservation-id
[fedora@fedora24 ~]$ curl http://169.254.169.254/latest/meta-data/
public-ipv4
10.100.1.110
```

Можно проверить файл журнала /var/log/neutron/metadata-agent.log на сервере network и убедиться, что агент метаданных предоставил последние запросы:

```
2018-03-11 13:21:58.688 1902 INFO eventlet.wsgi.server [-]
172.16.0.3,<local> "GET /latest/meta-data/ HTTP/1.1" status: 200
len: 360 time: 0.3548672
2018-03-11 13:23:29.910 1902 INFO eventlet.wsgi.server [-]
172.16.0.3,<local> "GET /latest/meta-data/public-ipv4 HTTP/1.1"
status: 200  len: 135 time: 0.3265879
```

Глава 10

За фасадом Neutron

В этой главе мы посмотрим, что скрывается за интерфейсом команд работы с сетями Neutron. Посмотрим на сетевые пространства имен, коммутатор Open vSwitch, группы безопасности и некоторые другие темы, связанные с сетью в OpenStack.

Виртуальный коммутатор Open vSwitch

Начнем с виртуального коммутатора Open vSwitch и сетевых пространств имен. Прежде чем говорить о сетевых пространствах имен, кратко рассмотрим, что такое пространства имен (Linux Namespaces) вообще.

Пространства имен контролируют доступ к структурам данных ядра. Фактически это означает изоляцию процессов друг от друга и возможность иметь параллельно «одинаковые», но не пересекающиеся друг с другом иерархии процессов, пользователей и сетевых интерфейсов. При желании разные сервисы могут иметь даже свои собственные loopback-интерфейсы.

Примеры пространств имен:

- PID, Process ID – изоляция иерархии процессов;
- NET, Networking – изоляция сетевых интерфейсов;
- PC, InterProcess Communication – управление взаимодействием между процессами;
- MNT, Mount – управление точками монтирования;
- UTS, Unix Timesharing System – изоляция ядра и идентификаторов версии.

Нас будут интересовать сетевые пространства имен. Данная команда покажет список существующих сетей:

```
$ openstack network list
+--------------------------------------+---------+--------------------------------------+
| ID                                   | Name    | Subnets                              |
```

```
+--------------------------------------+---------+--------------------------------------+
| 1d25a0b0-f1a4-49c2-9388-bed695c11267 | ext-net | 18b64199-932b-46ee-9537-06bf83f6d4d7 |
| 330c12e5-d560-4b85-abc0-05de6ee7cff4 | demo-net | e3729508-98d6-4fd1-856d-03602c6b178e |
+--------------------------------------+---------+--------------------------------------
++----------------------------------------------+
```

Сравним его со списком сетевых пространств имен на сетевом узле:

```
[root@network ~]# ip netns
qrouter-eb5aa33d-726a-46a7-a589-d78f99d1c2eb
qdhcp-330c12e5-d560-4b85-abc0-05de6ee7cff4
```

Можно заметить, что у нас есть одно пространство имен с идентификатором, совпадающим с внутренней сетью. Кроме того, существует сетевое пространство с id, совпадающим с идентификатором единственного маршрутизатора. Сравнить id можно при помощи команды `neutron router-list`. Пространства имен создаются только тогда, когда в них появляется необходимость, и автоматически не удаляются. Отсюда можно сделать вывод, что на демосистеме нет экземпляров виртуальных машин, обращающихся к ext-net.

Для сетей создаются пространства имен с префиксом «qdhcp-», а для маршрутизаторов – с префиксом «qrouter-». За их создание отвечает neutron-l3-agent. Список агентов можно вывести командой openstack network agent list.

В восьмой главе мы уже сталкивались с командой, управляющей Open vSwitch (OVS). Прежде чем смотреть на конфигурацию OVS, чуть подробнее остановимся на самом сервисе. Архитектура сервиса приведена на рис. 10.1. Обратите внимание, что мы сейчас не рассматриваем контроллер SDN, и эту часть рисунка можно пока игнорировать. Кроме того, на рисунке представлен Open vSwitch (OVS) без поддержки Intel DPDK, и начиная с версии 2.4 порт по умолчанию для OpenFlow – 6653, а OVSDB – 6640 (в соответствии с закрепленными IANA портами).

OpenFlow – это протокол, который является одним из ключевых компонентов, позволяющих строить программно определяемые сети передачи данных (SDN). При помощи данного протокола контроллер сети SDN может удаленно контролировать таблицы потоков на коммутаторах и маршрутизаторах.

Протокол Open vSwitch Database (OVSDB) изначально являлся частью OVS. Сейчас этот протокол управления коммутаторами заявлен как RFC 7047 – https://tools.ietf.org/html/rfc7047.

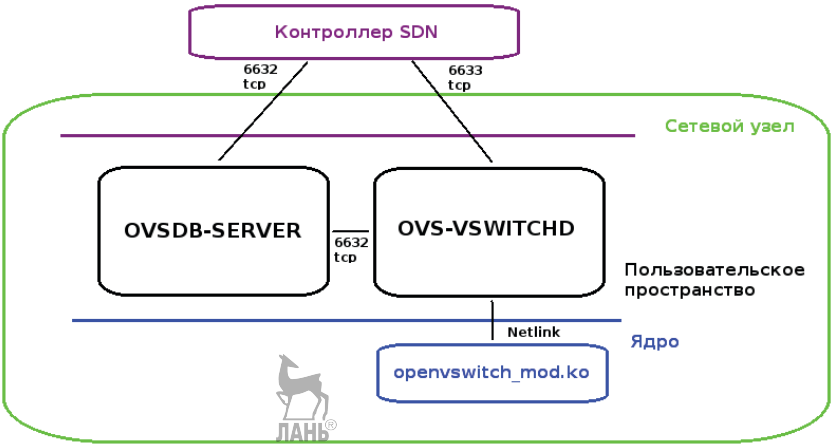


Рис. 10.1. Компоненты Open vSwitch (OVS)

Коммутатор состоит из трех основных компонентов:

- модуль ядра openswitch_mod.ko. Можно условно сравнить его с микросхемами ASIC аппаратных коммутаторов. Отвечает за работу с пакетами;
- демон ovs-vswitchd. Отвечает за управление, программирование логики пересылки пакетов, VLAN'ы и объединение сетевых карт;
- сервер базы данных ovsdb-server. Отвечает за ведение базы данных с конфигурацией.

Из полезных команд администратора при работе с Open vSwitch можно выделить:

- **ovs-vsctl show** – вывод общей информации по коммутатору;
- **ovs-vsctl add-br/del-br** – добавить или удалить мост;
- **ovs-vsctl add-port/del-port** – добавить или удалить порт;
- **ovs-ofctl dump-flows** – запрограммированные потоки для конкретного коммутатора. За их определение отвечает агент neutron-openvswitch-agent. По умолчанию они выводятся в том порядке, в каком поступили от агента. Для вывода в порядке приоритета, в порядке, в котором они обрабатываются, необходимо добавить опцию `--resort`;
- **ovsdb-tool show-log** – показать все команды настройки, отданные OVS, при помощи утилит пространства пользователя. Именно так и работает с OVS агент Neutron.

Open vSwitch поддерживает протокол OpenFlow и может выступать в качестве коммутаторов для Software Defined Network (SDN). Помимо коммутаторов, такое решение должно включать в себя контроллер, который будет выступать в качестве центральной точки управления. На настоящий момент существует как ряд коммерческих контроллеров, так и ряд проектов с открытым исходным кодом, разрабатывающих подобные решения, например OpenDaylight или OpenContrail.

Прежде чем двигаться вперед, приведем снимок с экрана сетевой топологии рассматриваемой конфигурации в Horizon. Он дан на рис. 10.2.

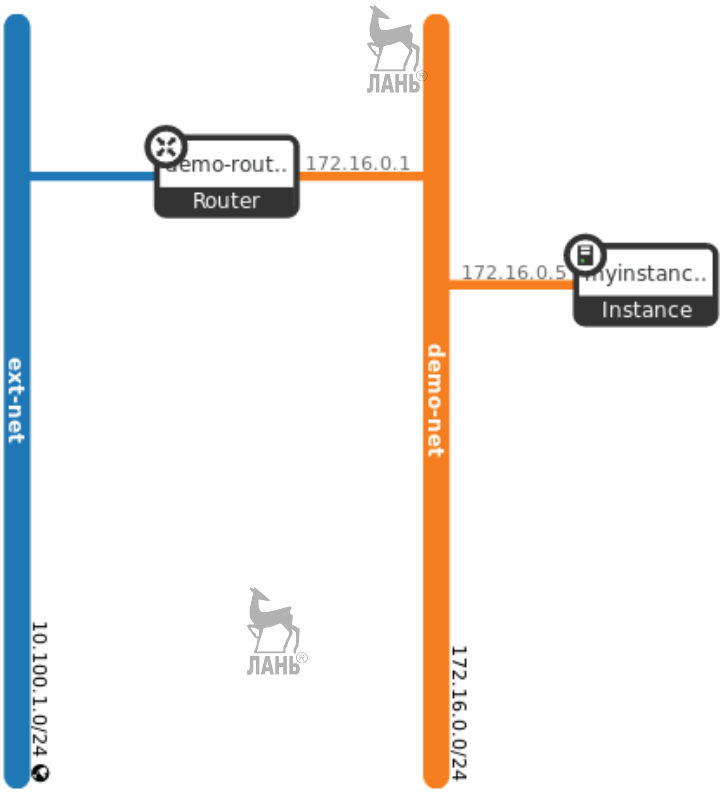


Рис. 10.2. Рассматриваемая топология сети в интерфейсе Horizon

Выведем список всех портов и после посмотрим на конфигурацию OVS:

```
$ openstack port list
+--------------------------------------+------+-------------------+---------------
-------------------------------------------------------------+--------+
| ID                                   | Name | MAC Address       | Fixed IP
Addresses                                                    | Status |
+--------------------------------------+------+-------------------+---------------
-------------------------------------------------------------+--------+
| 118776a5-cb5a-4b07-a340-a0c33a4d3b57 |      | fa:16:3e:79:ea:e4 | ip_
address='10.100.1.108', subnet_id='18b64199-932b-46ee-9537-06bf83f6d4d7' | DOWN   |
| 4194b0a8-77ff-4b72-b744-5d3cd86c5c7f |      | fa:16:3e:95:67:8b | ip_
address='172.16.0.5', subnet_id='e3729508-98d6-4fd1-856d-03602c6b178e'   | ACTIVE |
| 656bbfee-1642-4837-8af0-f11ae7fa5184 |      | fa:16:3e:75:bb:20 | ip_
address='10.100.1.109', subnet_id='18b64199-932b-46ee-9537-06bf83f6d4d7' | N/A    |
| 866cd6d8-7e1f-43a8-a25b-cffb29347ec3 |      | fa:16:3e:8d:ea:4b | ip_
address='172.16.0.1', subnet_id='e3729508-98d6-4fd1-856d-03602c6b178e'   | ACTIVE |
| cf666fec-a9b3-4e28-840e-01008e03cb6d |      | fa:16:3e:c6:79:ac | ip_
address='172.16.0.2', subnet_id='e3729508-98d6-4fd1-856d-03602c6b178e'   | ACTIVE |
+--------------------------------------+------+-------------------+---------------
-------------------------------------------------------------+--------+
```

Посмотрим на конфигурацию Open vSwitch на сетевом узле:

```
[root@network ~]# ovs-vsctl show
d9eb006a-b746-4875-947a-ef9dc0190975
    Manager "ptcp:6640:127.0.0.1"
        is_connected: true
    Bridge br-ex
        Controller "tcp:127.0.0.1:6633"
            is_connected: true
        fail_mode: secure
        Port "qg-118776a5-cb"
            Interface "qg-118776a5-cb"
                type: internal
        Port "eth1"
            Interface "eth1"
        Port phy-br-ex
            Interface phy-br-ex
                type: patch
                options: {peer=int-br-ex}
        Port br-ex
            Interface br-ex
                type: internal
    Bridge br-tun
        Controller "tcp:127.0.0.1:6633"
            is_connected: true
        fail_mode: secure
        Port br-tun
            Interface br-tun
                type: internal
        Port patch-int
            Interface patch-int
```

```
                type: patch
                options: {peer=patch-tun}
        Port "gre-c0a87ad7"
            Interface "gre-c0a87ad7"
                type: gre
                options: {df_default="true", in_key=flow, local_
ip="192.168.122.220", out_key=flow, remote_ip="192.168.122.215"}
        Port "gre-c0a87ad2"
            Interface "gre-c0a87ad2"
                type: gre
                options: {df_default="true", in_key=flow, local_
ip="192.168.122.220", out_key=flow, remote_ip="192.168.122.210"}
    Bridge br-int
        Controller "tcp:127.0.0.1:6633"
            is_connected: true
        fail_mode: secure
        Port "qr-866cd6d8-7e"
            tag: 1
            Interface "qr-866cd6d8-7e"
                type: internal
        Port "tapcf666fec-a9"
            tag: 1
            Interface "tapcf666fec-a9"
                type: internal
        Port int-br-ex
            Interface int-br-ex
                type: patch
                options: {peer=phy-br-ex}
        Port br-int
            Interface br-int
                type: internal
        Port patch-tun
            Interface patch-tun
                type: patch
                options: {peer=patch-int}
    ovs_version: "2.8.2"
```

И на узле, где запущен экземпляр виртуальной машины. В данном случае это compute-opt:

```
[root@compute-opt ~]# ovs-vsctl show
9a15c9b2-2347-4263-9105-0e95c300090c
    Manager "ptcp:6640:127.0.0.1"
        is_connected: true
    Bridge br-int
        Controller "tcp:127.0.0.1:6633"
            is_connected: true
        fail_mode: secure
        Port patch-tun
            Interface patch-tun
                type: patch
```

```
                options: {peer=patch-int}
        Port br-int
            Interface br-int
                type: internal
        Port "qvo4194b0a8-77"
            tag: 1
            Interface "qvo4194b0a8-77"
    Bridge br-tun
        Controller "tcp:127.0.0.1:6633"
            is_connected: true
        fail_mode: secure
        Port patch-int
            Interface patch-int
                type: patch
                options: {peer=patch-tun}
        Port "gre-c0a87ad2"
            Interface "gre-c0a87ad2"
                type: gre
                options: {df_default="true", in_key=flow, local_
ip="192.168.122.215", out_key=flow, remote_ip="192.168.122.210"}
        Port "gre-c0a87adc"
            Interface "gre-c0a87adc"
                type: gre
                options: {df_default="true", in_key=flow, local_
ip="192.168.122.215", out_key=flow, remote_ip="192.168.122.220"}
        Port br-tun
            Interface br-tun
                type: internal
    ovs_version: "2.8.2"
```

В дополнение к этой распечатке автор привел на рис. 10.3 и 10.4 результат работы команды `plotnetcfg` для сетевого и вычислительного узлов. Эти диаграммы также помогут нам при разборе упражнений данного раздела. Информация про установку и использование `plotnetcfg` будет приведена далее в этой главе. Для использования диаграмм она пока не нужна.

Возвращаемся к выводу ovs-vsctl show и иллюстрациям. Как мы видим, на узле network созданы три моста: br-int, br-tun и br-ex. На compute-opt один из них, br-ext, отсутствует.

Br-int – интеграционный мост, предназначенный для подключения экземпляров виртуальных машин. Он осуществляет VLAN-тегирование и снятие VLAN-тегов для трафика приходящего с/на вычислительные узлы. Br-int существует на вычислительных и сетевых узлах и создается автоматически при первом старте агента Neutron для Open vSwitch. На узле network к нему в настоящий момент подключены пять портов. Один из них соответствует са-

мому мосту, один – порт с IP 172.16.0.2. Его имя tapcf666fec-a9. То, что он соответствует именно этому порту, мы можем определить по идентификатору подсети – см. вывод команды `openstack port list` ранее. Порт patch-tun – соединение с мостом tun. Порт qr-866cd6d8-7e – подключение интеграционного моста к единственному маршрутизатору. И наконец, int-br-ex – подключение к мосту br-ex.

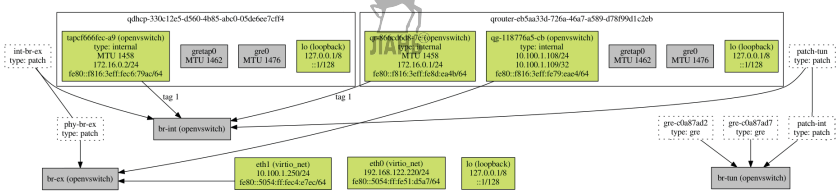


Рис. 10.3. Упрощенный вывод диаграммы plotnetcfg для сетевого узла

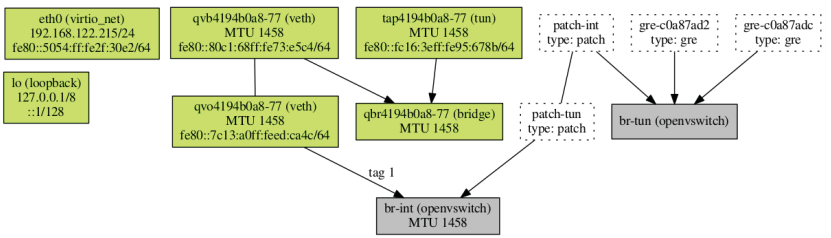


Рис. 10.4. Упрощенный вывод диаграммы plotnetcfg для вычислительного узла

На compute-opt у нас отсутствуют интерфейсы tapcf666fec-a9 и qr-866cd6d8-7e, зато присутствует qvo4194b0a8-77 для единственной виртуальной машины с IP-адресом 172.16.0.5. При этом поле tag идентифицирует VLAN. Обратите внимание, что эти теги уникальны только в пределах одного узла. Они не передаются при помощи туннелей.

Tap-устройства эмулируют устройства 2-го уровня стека, оперирующие фреймами. Они часто используются в системах, связанных с сетевой безопасностью или мониторингом для перехвата трафика. В OpenStack они реализуют виртуальные сетевые карты виртуальных машин (VIF или vNIC) и порты на мостах, реализованных Linux Bridge. Virtual Ethernet (vEth) устройства, в нашем случае qvo (в сторону Open vSwitch) и qvb (в сторону моста), всегда идут парами и позволяют связывать сетевые пространства имен с «внешним миром».

Мост br-tun – в нашем случае это GRE-туннель. Связывает сетевые и вычислительные узлы, передавая тегированный трафик с интеграционного моста, используя правила OpenFlow. В выводе команды мы видим связанность сетевого узла (192.168.122.210) и обоих вычислительных (192.168.122.210 и 192.168.122.215).

Br-ex – мост, осуществляющий взаимодействие с внешним миром. Существует только на сетевых узлах. В нашем примере видно, что среди портов присутствуют физический интерфейс eth1 и порт qg-118776a5-cb, который обслуживает «внешние» IP-адреса маршрутизатора. В этом можно убедиться, введя команду, позволяющую посмотреть IP-адреса пространства имен маршрутизатора:

```
[root@network ~]# ip netns exec qrouter-eb5aa33d-726a-46a7-a589-
d78f99d1c2eb ip a
...
14: qg-118776a5-cb: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc
noqueue state UNKNOWN qlen 1000
    link/ether fa:16:3e:79:ea:e4 brd ff:ff:ff:ff:ff:ff
    inet 10.100.1.108/24 brd 10.100.1.255 scope global qg-118776a5-cb
       valid_lft forever preferred_lft forever
    inet 10.100.1.109/32 brd 10.100.1.109 scope global qg-118776a5-cb
       valid_lft forever preferred_lft forever
    inet6 fe80::f816:3eff:fe79:eae4/64 scope link
       valid_lft forever preferred_lft forever
...
```

Суммарно информация по конфигурации Open vSwitch приведена в табл. 5.

Таблица 5. Конфигурация Open vSwitch

Интерфейс	Описание
br-ex	Мост, осуществляющий взаимодействие с внешним миром. Через tap-интерфейс подключается к порту маршрутизатора qg
br-int	Интеграционный мост. Предназначен для передачи входящего и исходящего трафиков виртуальных машин. Отвечает за создание и удаление VLAN-тегов
br-tun	Связывает сетевые и вычислительные узлы, передавая тегированный трафик с интеграционного моста, используя правила OpenFlow
qvo	Подключение в сторону Open vSwitch (veth-пара)
qvb	Подключение в сторону моста qbr (veth-пара)

Интерфейс	Описание
qbr	Мост Linux bridge, предназначенный для создания групп безопасности. Фактически это – сетевой интерфейс виртуальной машины
qg	Подключение маршрутизатора к шлюзу
qr	Подключение маршрутизатора к интеграционному мосту

Напоследок приведем пример части вывода команды `ovs-vsctl show` в том случае, если бы мы использовали VXLAN-тунелирование:

```
[root@compute-opt ~]# ovs-vsctl show
...
    Bridge br-tun
        fail_mode: secure
        Port patch-int
            Interface patch-int
                type: patch
                options: {peer=patch-tun}
        Port br-tun
            Interface br-tun
                type: internal
        Port "vxlan-ac19fc0a"
            Interface "vxlan-ac19fc0a"
                type: vxlan
                options: {df_default="true", in_key=flow, local_
ip="192.168.122.215", out_key=flow, remote_ip="192.168.122.210"}
...
```

Группы безопасности

Правила группы безопасности преобразуются в цепочку Netfilter и применяются к мосту qbrfID, который управляется не Open vSwitch, а Linux Bridge на вычислительном узле.

Собственно, единственная причина появления Linux bridge – это то, что Open vSwitch не может работать с правилами iptables, которые применяются на виртуальный интерфейс, непосредственно подключенный к порту коммутатора. Отсюда и появилась «прокладка» в виде моста qbr. Посмотреть, как преобразовались правила группы безопасности в правила iptables, можно командой `iptables -S`. Пример вывода для рассматриваемой конфигурации приведен в приложении 1 в конце книги. Важно отметить, что в новых версиях OpenStack появилась возможность избавиться от qbr за счет реализации правил iptables в правилах потоков OVS.

Посмотрим, как это работает. Выведем конфигурацию Linux bridge:

```
[root@compute-opt ~]# brctl show
bridge name      bridge id           STP enabled   interfaces
qbr4194b0a8-77   8000.82c16873e5c4   no            qvb4194b0a8-77
                                                   tap4194b0a8-77
```

В нашем случае искомый интерфейс – это tap4194b0a8-77. Данное TAP-устройство – фактически сетевой интерфейс виртуальной машины. Посмотрим на правила брандмауэра, связанные с этим интерфейсом:

```
[root@compute-opt ~]# iptables -S | grep tap4194b0a8-77
-A neutron-openvswi-FORWARD -m physdev --physdev-out tap4194b0a8-77
--physdev-is-bridged -m comment --comment "Direct traffic from the VM
interface to the security group chain." -j neutron-openvswi-sg-chain
-A neutron-openvswi-FORWARD -m physdev --physdev-in tap4194b0a8-77
--physdev-is-bridged -m comment --comment "Direct traffic from the VM
interface to the security group chain." -j neutron-openvswi-sg-chain
-A neutron-openvswi-INPUT -m physdev --physdev-in tap4194b0a8-77
--physdev-is-bridged -m comment --comment "Direct incoming traffic
from VM to the security group chain." -j neutron-openvswi-o4194b0a8-7
-A neutron-openvswi-sg-chain -m physdev --physdev-out tap4194b0a8-77
--physdev-is-bridged -m comment --comment "Jump to the VM specific
chain." -j neutron-openvswi-i4194b0a8-7
-A neutron-openvswi-sg-chain -m physdev --physdev-in tap4194b0a8-77
--physdev-is-bridged -m comment --comment "Jump to the VM specific
chain." -j neutron-openvswi-o4194b0a8-7
```

В цепочке neutron-openvswi-sg-chain применяются группы безопасности. Цепочка neutron-openvswi-o4194b0a8-7 относится к исходящему трафику из виртуальной машины, а цепочка neutron-openvswi-i4194b0a8-7 отвечает за трафик, отправляемый в виртуальную машину. Приведем пример вывода iptables -L для этой цепочки:

```
Chain neutron-openvswi-i4194b0a8-7 (1 references)
target     prot opt source               destination
RETURN     all  --  anywhere             anywhere             state RELATED,ESTABLISHED
/* Direct packets associated with a known session to the RETURN chain. */
RETURN     udp  --  anywhere             172.16.0.5           udp spt:bootps dpt:bootpc
RETURN     udp  --  anywhere             255.255.255.255      udp spt:bootps dpt:bootpc
RETURN     tcp  --  anywhere             anywhere             tcp dpt:ssh
RETURN     icmp --  anywhere             anywhere
DROP       all  --  anywhere             anywhere             state INVALID /* Drop
packets that appear related to an existing connection (e.g. TCP ACK/FIN) but do not
have an entry in conntrack. */
```

```
neutron-openvswi-sg-fallback  all  --  anywhere             anywhere             /*
Send unmatched traffic to the fallback chain. */
```

Как вы помните, единственными разрешающими правилами для созданной нами группы безопасности было разрешение ssh-трафика и протокола ICMP (выделено в листинге).

Другой способ ограничения трафика в OpenStack – это использование Firewall-as-a-Service (FWaaS). В отличие от групп безопасности, которые работают с трафиком на уровне интерфейсов виртуальных машин, FWaaS позволяет защитить сразу несколько виртуальных машин.

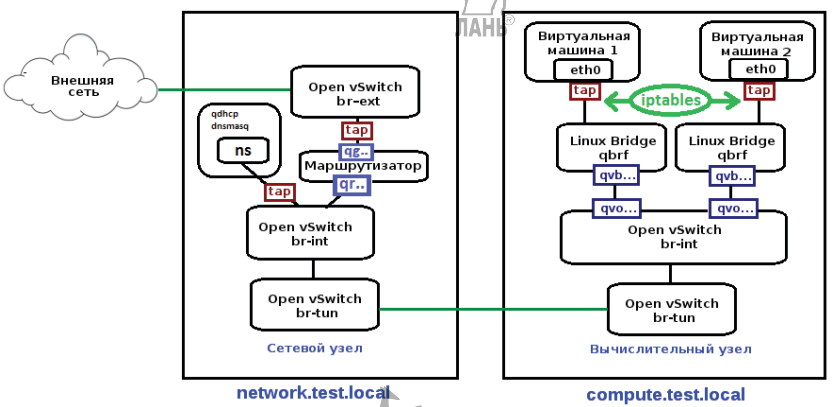


Рис. 10.5. Диаграмма сетевых подключений Neutron

В заключение раздела приведем рис. 10.5 с упрощенной диаграммой сетевых подключений Neutron. Как получить реальную диаграмму с реально работающей системой, мы рассмотрим далее в этой главе.

Утилита для визуализации сети plotnetcfg

Одним из полезных инструментов для изучения и отладки сетевой подсистемы OpenStack является plotnetcfg. Этот инструмент позволяет просканировать текущую сетевую конфигурацию и представить ее в виде диаграммы. Примеры результата работы plotnetcfg приведены ранее на рис. 10.3 и 10.4.

Установить утилиту в учебную среду можно командой

```
# yum -y install plotnetcfg graphviz
```

Первый пакет содержит непосредственно утилиту plotnetcfg, а второй добавит в систему ряд утилит, включая dot. Вы, возможно, уже использовали утилиту визуализации графов dot, если анализировали зависимости модулей systemd при помощи systemd-analyze. Запуск производим следующим образом:

```
$ plotnetcfg | dot -Tpdf > config.pdf
```

Одна из задач, поставленных при создании plotnetcfg, – легковесность. В «боевой» среде рекомендуется просто скопировать бинарный файл на исследуемый узел. Исходный код и инструкции по сборке находятся на сайте проекта: https://github.com/jbenc/plotnetcfg.

Зеркалирование трафика на Open vSwitch для мониторинга сети в OpenStack

Зачастую перед администратором встает задача отладки приложений в виртуальной сети OpenStack, и тогда возникает необходимость воспользоваться привычными инструментами наподобие tcpdump и Wireshark.

Запустим один экземпляр виртуальной машины:


```
$ nova boot --flavor m2.tiny --image cirros-raw --key-name demokey1
--security-groups demo-sgroup test-vm
```

Далее, определив, на каком из вычислительных узлов запустилась виртуальная машина test-vm, посмотрим топологию Open vSwitch:

```
[root@compute-opt ~]# ovs-vsctl show
20eab69c-e759-41b0-a480-97688ec0b4b8
    Bridge br-int
        fail_mode: secure
        Port "qvobee51cf7-fb"
            tag: 1
            Interface "qvobee51cf7-fb"
        Port patch-tun
            Interface patch-tun
                type: patch
                options: {peer=patch-int}
        Port br-int
            Interface br-int
                type: internal
```

```
    Bridge br-tun
        fail_mode: secure
        Port "gre-c0a87adc"
            Interface "gre-c0a87adc"
                type: gre
                options: {df_default="true", in_key=flow, local_
ip="192.168.122.215", out_key=flow, remote_ip="192.168.122.220"}
        Port br-tun
            Interface br-tun
                type: internal
        Port "gre-c0a87ad2"
            Interface "gre-c0a87ad2"
                type: gre
                options: {df_default="true", in_key=flow, local_
ip="192.168.122.215", out_key=flow, remote_ip="192.168.122.210"}
        Port patch-int
            Interface patch-int
                type: patch
                options: {peer=patch-tun}
    ovs_version: "2.4.0"
```

Для наглядности на рис. 10.6 приведена релевантная часть диаграммы, сформированной на этом узле при помощи plotnetcfg. Пока не обращайте внимания на интерфейс br-int-tcpdump, который сейчас отсутствует в топологии.

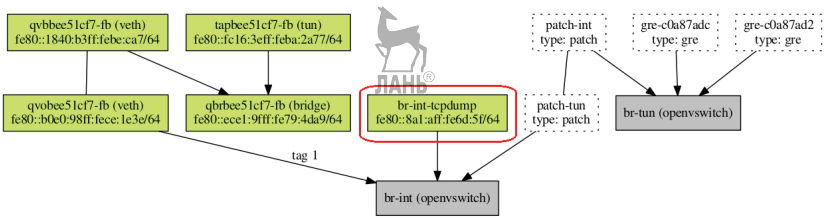


Рис. 10.6. Сетевая диаграмма вычислительного узла при зеркалировании порта OVS

Теперь из виртуальной машины test-vm попробуем «достучаться» до виртуального маршрутизатора:

```
$ ping 172.16.0.1
PING 172.16.0.1 (172.16.0.1) 56 data bytes
64 bytes from 172.16.0.1: icmp_seq=0 ttl=64 time=1.766 ms
64 bytes from 172.16.0.1: icmp_seq=1 ttl=64 time=0.617 ms
...
```

Эти ICMP-пакеты мы и хотим перехватить. Попробуем с вычислительного узла захватить пакеты при помощи tcpdump. Однако, как мы видим, все попытки приведут к ошибкам:

```
[root@compute-opt ~]# tcpdump -npi patch-tun -vvvs0 -w /tmp/dump.cap
tcpdump: patch-tun: No such device exists
(SIOCGIFHWADDR: No such device)
[root@compute-opt ~]# tcpdump -npi br-int -vvvs0 -w /tmp/dump.cap
tcpdump: br-int: That device is not up
```

Это связано с тем, что внутренние устройства Open vSwitch не видимы для большинства утилит «извне» OVS. В частности, именно из-за этого для реализации групп безопасности в Neutron используется Linux Bridge. Open vSwitch не может работать с правилами iptables, которые применяются на виртуальный интерфейс, непосредственно подключенный к порту коммутатора.

Мы бы могли снимать трафик с интерфейса qvobee51cf7-fb, но в случае рестарта виртуальной машины или если бы нам надо было бы зеркалировать весь трафик, это бы решение нам не подошло бы. В качестве выхода мы создадим dummy-интерфейс:

```
[root@compute-opt ~]# ip link add name br-int-tcpdump type dummy
[root@compute-opt ~]# ip link set dev br-int-tcpdump up
```

Затем добавим br-int-tcpdump к мосту br-int, трафик с которого мы хотим перехватывать:

```
[root@compute-opt ~]# ovs-vsctl add-port br-int br-int-tcpdump
```

Проверяем:

```
[root@compute-opt ~]# ovs-vsctl show
20eab69c-e759-41b0-a480-97688ec0b4b8
    Bridge br-int
        fail_mode: secure
        Port "qvobee51cf7-fb"
            tag: 1
            Interface "qvobee51cf7-fb"
        Port patch-tun
            Interface patch-tun
                type: patch
                options: {peer=patch-int}
        Port br-int
            Interface br-int
                type: internal
        Port br-int-tcpdump
            Interface br-int-tcpdump
...
```

Именно текущей конфигурации соответствует рис. 10.5. Осталось открыть man ovs-vsctl и поискать фразу «Port Mirroring». В результате поиска по инструкции из man-страницы задаем сле-

дующую команду, при помощи которой мы будем зеркалировать трафик со внутреннего порта на br-int-tcpdump:

```
[root@compute-opt ~]# ovs-vsctl -- set Bridge br-int mirrors=@m
-- --id=@br-int-tcpdump get Port br-int-tcpdump -- --id=@br-int
get Port br-int -- --id=@m create Mirror name=mirrortest select-
dst-port=@br-int select-src-port=@br-int output-port=@br-int-
tcpdump select_all=1
```

Наконец, можно начать перехватывать трафик:

```
[root@compute-opt ~]# tcpdump -npi br-int-tcpdump -vvvs0 -w /tmp/dump.cap
tcpdump: WARNING: br-int-tcpdump: no IPv4 address assigned
tcpdump: listening on br-int-tcpdump, link-type EN10MB (Ethernet), capture
size 65535 bytes
^C23 packets captured
23 packets received by filter
0 packets dropped by kernel
```

После чего копируем дамп на машину с установленным Wireshark для удобного анализа:

```
andrey@elx:~$ scp root@192.168.122.215:/tmp/dump.cap .
root@192.168.122.215's password:
dump.cap                                      100% 2626     2.6KB/s   00:00
```

И наконец, наша задача выполнена. Открываем дамп – рис. 10.7.

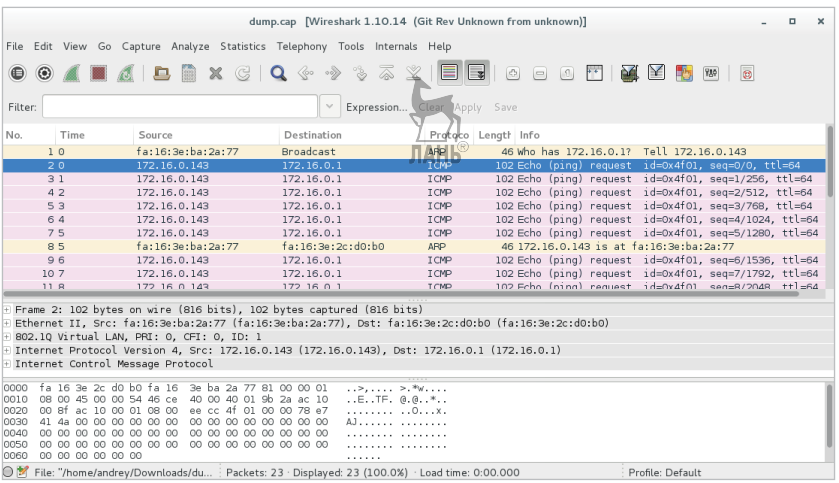


Рис. 10.7. Перехваченный трафик в Wireshark

Балансировщик нагрузки как сервис (LBaaS)

Рассмотрим реализацию LBaaS в OpenStack при помощи подключаемого модуля Neutron LBaaS и HAProxy. По плану сообщества в дальнейшем, стандартной реализацией балансировщика нагрузки как сервиса должен стать проект Octavia (https://github.com/openstack/octavia).

При помощи подключаемого модуля Neutron LBaaS становится возможным средствами самого облака производить балансировку входящих сетевых подключений между экземплярами виртуальных машин, входящих в один кластер. Подключаемый модуль LBaaS по умолчанию использует HAProxy (http://www.haproxy.org/) – быстрый и широко известный в мире GNU/Linux балансировщик нагрузки, написанный на языке программирования C.

По умолчанию HAProxy использует тот же порт, что и сервис Keystone, – 5000. Поэтому если вы устанавливаете сервис на тот же узел, что и сервис идентификации, то вам необходимо поменять в конфигурационном файле HAProxy на какой-либо другой незанятый порт. В нашем случае сетевой узел и управляющий разнесены, поэтому в подобной настройке необходимость отсутствует. Ставим прокси:

```
[root@network ~]# yum -y install haproxy
```

Укажем в качестве сервисного подключаемого модуля LBaaS:

```
[root@controller ~]# crudini --set /etc/neutron/neutron.conf
DEFAULT service_plugins neutron.services.loadbalancer.plugin.
LoadBalancerPlugin
```

В качестве service_provider мы укажем HAProxy:

```
[root@controller ~]# crudini --set /etc/neutron/neutron.conf
service_provider service_provider LOADBALANCER:Haproxy:neutron.
services.loadbalancer.drivers.haproxy.plugin_driver.HaproxyOnHos
tPluginDriver:default
```

Если бы мы использовали какой-либо аппаратный балансировщик нагрузки, его драйвер как раз бы стоило указывать в этом параметре. Проверить список доступных сервисных провайдеров по окончании настройки можно командой `neutron service-provider-list`:

```
$ neutron service-provider-list
+--------------+----------+---------+
| service_type | name     | default |
+--------------+----------+---------+
| VPN          | openswan | True    |
| LOADBALANCER | haproxy  | True    |
+--------------+----------+---------+
```

Настраиваем агент LBaaS на сетевом узле:

```
[root@network ~]# crudini --set /etc/neutron/lbaas_agent.ini DEFAULT
interface_driver neutron.agent.linux.interface.OVSInterfaceDriver
[root@network ~]# crudini --set/etc/neutron/lbaas_agent.ini DEFAULT
device_driver neutron.services.loadbalancer.drivers.haproxy.
namespace_driver.HaproxyNSDriver
[root@network ~]# crudini --set /etc/neutron/lbaas_agent.ini haproxy
user_group haproxy
```

Запускаем сервисы:

```
[root@controller ~]# systemctl restart neutron-server.service
[root@network ~]# systemctl start neutron-lbaas-agent.service
[root@network ~]# systemctl enable neutron-lbaas-agent.service
```

Теперь осталось проверить секцию OPENSTACK_NEUTRON_NETWORK в конфигурационном файле /etc/openstack-dashboard/local_settings, если вы уже установили веб-интерфейс Horizon, который рассматривается в следующей главе. Соответствующая секция должна выглядеть примерно так:

```
OPENSTACK_NEUTRON_NETWORK = {
    'enable_lb': True,
    'enable_firewall': True,
    'enable_quotas': True,
    'enable_security_group': True,
    'enable_vpn': False,
}
```

Нас в данном случае интересует параметр `'enable_lb': True`, который отвечает за появление соответствующей вкладки для работы с LBaaS.

Балансировщик нагрузки поддерживает три типа политики: подключения принимаются по очереди всеми виртуальными машинами, с одного IP-адреса подключения принимает одна и та же виртуальная машина, и, наконец, подключение принимает машина с наименьшим числом подключений. Реализован LBaaS по аналогии с пространствами имен DHCP. По команде `ip netns list` вы можете найти пространства имен вида `qlbaas-*`.

Попробуем создать сервис при помощи LbaaS. Запускаем несколько экземпляров виртуальных машин, которые составят кластер, с которым будет работать балансировщик.

1. Создаем пул виртуальных машин командой:

   ```
   neutron lb-pool-create --lb-method ROUND_ROBIN --protocol
   HTTP --name pool1 --subnet privsub.
   ```

2. Создаем членов пула, используя IP виртуальных машин:

   ```
   neutron lb-member-create --subnet privsubnet --address
   <ip_сервера> --protocol-port 80 pool1.
   ```

3. Создаем сервис мониторинга доступности сервиса, обеспечиваемого виртуальными машинами:

   ```
   neutron lb-healthmonitor-create --delay 3 --type HTTP
   --max-retries 3 --timeout 3.
   ```

4. Создаем виртуальный IP-адрес кластера:

   ```
   neutron lb-vip-create --name vip1 -protocol-port 80 --protocol
   HTTP --subnet subpriv pool1.
   ```

5. Создаем новый плавающий IP:

   ```
   neutron floatingip-create ext.
   ```

Глава 11

Веб-панель управления Horizon и работа пользователя из графического интерфейса

```
Название: OpenStack Dashboard (Horizon)
Назначение: веб-интерфейс управления OpenStack
Пакет: openstack-dashboard
Имена сервисов:  httpd.service, memcached.service
Порты: 80, 443
Конфигурационные файлы: /etc/openstack-dashboard/local_settings
Файлы журнала: /var/log/httpd/error_log, /var/log/horizon/horizon.log
```

В этой главе мы рассмотрим, как установить и работать с графическим интерфейсом OpenStack Dashboard (Horizon). Horizon представляет собой веб-приложение, написанное с использованием Django Python. Для работы с OpenStack Dashboard нужен браузер, поддерживающий HTML5 со включенными cookies и JavaScript. Обратите внимание, что через Horizon доступны только около 70% функционала управления облаком. Все 100% доступны лишь из командной строки.

Установка веб-интерфейса

Запускать сервис мы будем на узел контроллера controller, но с таким же успехом мы могли бы вынести Horizon на отдельную виртуальную машину. Устанавливаем необходимые пакеты:

```
[root@controller ~]# yum -y install httpd mod_ssl mod_wsgi
memcached python-memcached openstack-dashboard
```

Далее необходимо отредактировать конфигурационный файл /etc/openstack-dashboard/local_settings. Нужно изменить следующие опции:

```
ALLOWED_HOSTS = ['*']
OPENSTACK_HOST = "192.168.122.200"
```

Опция OPENSTACK_HOST – это самая главная опция при настройке Horizon. Это IP-адрес сервиса, где запущен Keystone. Желательно при промышленной эксплуатации ограничить доступность сервиса для конкретных IP-адресов и имен узлов. Их можно перечислить через запятую в опции ALLOWED_HOSTS. «Звездочка» означает любое имя узла. В ALLOWED_HOSTS должны быть перечислены имена хостов, которые будут приниматься сервером от браузера в качестве директивы host. Там можно указать, например, IP-адрес сетевой карты узла и FQDN имя сервера.

Параметр OPENSTACK_HOST указывает на местоположение сервиса Keystone.

Существует достаточно много опциональных параметров, которые также можно указать в конфигурационном файле /etc/openstack-dashboard/local_settings. Например, время жизни сессии в Horizon:

```
SESSION_TIMEOUT=3600
```

Время задается в секундах. По умолчанию – 30 минут. При использовании медленного канала бывает полезным увеличить это время, если ваши пользователи загружают образы виртуальных машин большего размера или запускаются шаблоны Heat с долгим временем исполнения. Этот параметр не может быть больше, чем время expiration=3600 секции [Token] конфигурационного файла keystone.conf.

Также полезно увеличить параметр в keystone.conf, если пользователи работают даже без применения Horizon, но с виртуальными машинами с большими образами дисков, например 150 Гб и больше.

Вернемся к настройке. Надо найти секцию CACHES, удалить или закомментировать второй блок и убрать комментарии с первого, чтобы он выглядел следующим образом:

```
CACHES = {
    'default': {
        'BACKEND': 'django.core.cache.backends.memcached.
MemcachedCache',
        'LOCATION': '127.0.0.1:11211',
    }
}
```

Включаем третью версию Keystone API:

```
OPENSTACK_KEYSTONE_URL = "http://%s:5000/v3" % OPENSTACK_HOST
```

и поддержку доменов:

```
OPENSTACK_KEYSTONE_MULTIDOMAIN_SUPPORT = True
```

Зададим поддерживаемые версии API сервисов:

```
OPENSTACK_API_VERSIONS = {
    "identity": 3,
    "image": 2,
    "volume": 2,
}
```

Зададим имя домена по умолчанию, которое будет автоматически подставляться на странице регистрации:

```
OPENSTACK_KEYSTONE_DEFAULT_DOMAIN = "default"
```

На момент написания книги существовала ошибка, связанная с некорректными разрешениями на директорию в CentOS 7. Необходимо исправить их командой:

```
[root@controller ~]# chown -R apache:apache /usr/share/openstack-
dashboard/static
```

Если вы не отключали SELinux, то необходимо выставить переключатель:

```
[root@controller ~]# setsebool -P httpd_can_network_connect on
```

Для исправления одной из возможных в некоторых конфигурациях проблем необходимо добавить строку WSGIApplicationGroup %{GLOBAL} в конфигурационный файл /etc/httpd/conf.d/openstack-dashboard.conf. Наконец, включаем сервисы и перезапускаем:

```
[root@controller ~]# systemctl enable memcached.service
[root@controller ~]# systemctl restart httpd.service memcached.service
```

Теперь можно открыть в браузере адрес http://controller.test.local/dashboard и подключиться одним из существующих пользователей, указав в качестве домена Default.

Работа с OpenStack в интерфейсе Horizon

Кратко рассмотрим работу с веб-интерфейсом Horizon. Обратите внимание, что число аккордеонов (accordion) – сворачивающихся элементов интерфейса в левой части экрана – зависит от того, административным или простым пользователем вы зашли в Horizon. Также не отображаются аккордеоны, отвечающие за службы, которые у вас не установлены, например если вы выполняете упражнения последовательно глава за главой, то Orchestration внутри Project у вас будет отсутствовать.

Начнем с простого пользователя demo и аккордеона Project, внутри которого скрывается большинство необходимых пользователю инструментов работы с облаком:

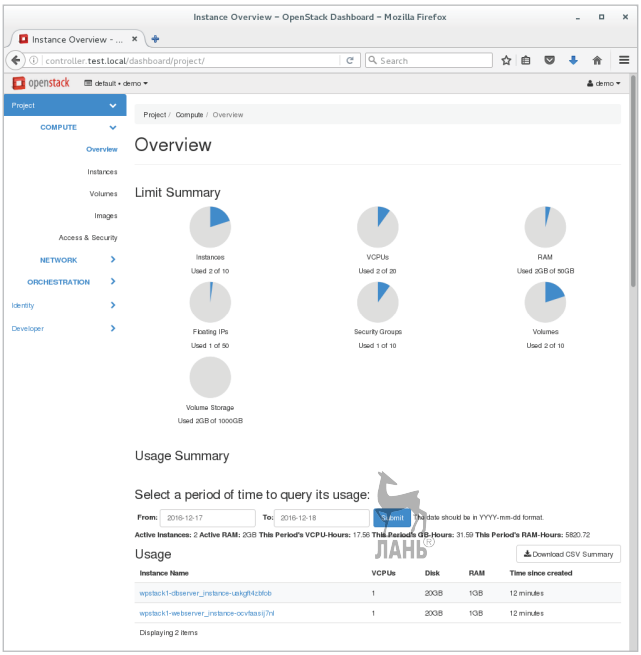


Рис. 11.1. Общий обзор ресурсов пользователя

- Compute
  - **Overview** – общий обзор ресурсов пользователя. Вкладка приведена на рис. 11.1. Помимо гистограмм, отображающих текущее потребление ресурсов, можно задать период и получить краткий отчет за это время.
  - **Instances** – на этой вкладке приведен список виртуальных машин для текущего проекта, а также имеется возможность производить с виртуальными машинами различные действия. С этой же вкладки открывается диалог создания новой виртуальной машины. Его внешний вид приведен на рис. 11.2. Обратите внимание, что в этом диалоге вкладки, помеченные звездочкой, содержат обязательные для заполнения параметры. Возможность подключиться к консоли и просмотреть вывод терминала виртуальной машины – также на вкладке **Instances**.
  - **Volumes** – вкладка предназначена для управления томами и снимками Cinder. Соответственно, состоит из двух вкладок – **Volumes** и **Volume Snapshots**.
  - **Images** – управление образами Glance. Снимок с экрана приведен на рис. 14.1. В верхней части экрана имеется возможность отсортировать образы по общедоступным образам, к которым вам дали доступ, и частным образам. Из образов тут же можно стартовать виртуальную машину и создать том.
  - **Access&Security** – на этой заключительной части аккордеона Compute собраны настройки, относящиеся к безопасности и контролю доступа. Разделена на четыре вкладки: **Security Groups** (группы безопасности), **Key Pairs** (ключи доступа по SSH), **Floating IPs** («плавающие» IP-адреса), **API Access** (URL, по которым доступны API соответствующих служб облака).
- Network
  - **Network Topology** – топология сети проекта. Пример снимка с экрана приведен на рисунке в главе, посвященной работе с сетью.
  - **Networks** – список сетей проекта. Тут же создаются подсети, а также находится список портов и возможность их редактирования.
  - **Routers** – список маршрутизаторов проекта и управление ими.

- Object Store
  - **Containers** – тут вы можете создавать контейнеры и псевдопапки, а также загружать в объектное хранилище файлы и скачивать их. Пример снимка с экрана приведен на рис. 6.2 в главе, рассказывающей о Swift.
- Orchestration
  - **Stacks** – снимки с экрана представлены на рис. 13.1 и 13.2. На этой вкладке располагается единая точка управления стеками, включая их запуск и мониторинг.

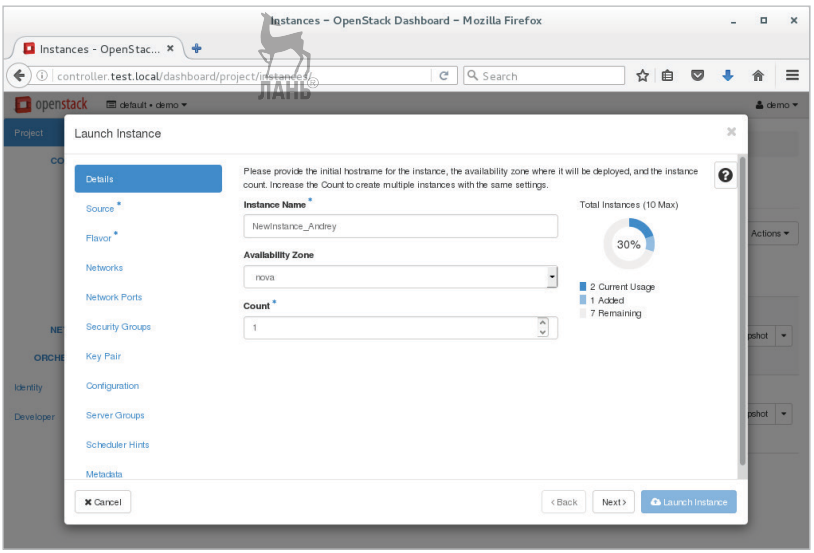


Рис. 11.2. Создание новой виртуальной машины

Для непривилегированного пользователя второй доступный аккордеон Identity предоставляет только справочную информацию, поэтому перейдем к рассмотрению веб-клиента Horizon для пользователя, прошедшего аутентификацию как администратор.

Первое, что мы увидим, – это дополнительный аккордеон Admin. С его описания мы и начнем:

- **Overview** – тут располагается статистика по всем проектам за заданный период времени;
- **Resource Usage** – эта вкладка отвечает за вывод информации от OpenStack Telemetry (Ceilometer). Пример снимка экрана приведен на рис. 12.1;

- **Hypervisors** – информация по всем гипервизорам, управляемым OpenStack. Кликнув по имени выбранного узла, можно посмотреть, какие виртуальные машины он обслуживает;
- **Host Aggregates** – на этой вкладке администратор может настроить группировку узлов как на логические группы, так и сгруппировать их по физическому месторасположению;
- **Instances** – вкладка соответствует аналогичной вкладке **Projects**, только отображает все виртуальные машины;
- **Volumes** – аналогична вкладке **Projects**, однако включает дополнительную подвкладку **Volume Types**, на которой можно определить типы томов и соответствующие им параметры качества обслуживания (QOS);
- **Flavors** – управление типами экземпляров виртуальных машин;
- **Images** – управление образами Glance. Отображаются все образы;
- **Networks** – аналогична вкладке **Projects**, но включает в себя сети всех проектов. Тут администратор также может задать внешнюю сеть;
- **Routers** – опять же как и в аккордеоне **Projects**, но включает в себя все маршрутизаторы;
- **Defaults** – тут администратор может задать квоты для проектов, действующие по умолчанию;
- **System Information** – на этой вкладке представлена информация о сервисах облака и их текущем состоянии. Пример снимка с экрана вкладки **System Information** приведен на рис. 11.3.

Самый последний аккордеон **Identity** состоит всего из двух разделов: **Projects** и **Users**. На вкладке **Projects** можно создавать и управлять проектами. Вкладка **Users** предназначена для аналогичных действий с пользователями.

Существует возможность переключить язык интерфейса Heat на русский язык. Если щелкнуть по имени пользователя в правом верхнем углу и выбрать из выпадающего меню пункт **Settings**, то откроется экран, показанный на рис. 11.4, где вы можете поменять выбор языка интерфейса и локальную зону времени. Кроме того, на этом рисунке продемонстрирована вторая, поставляющаяся в комплекте с Horizon тема оформления – material.

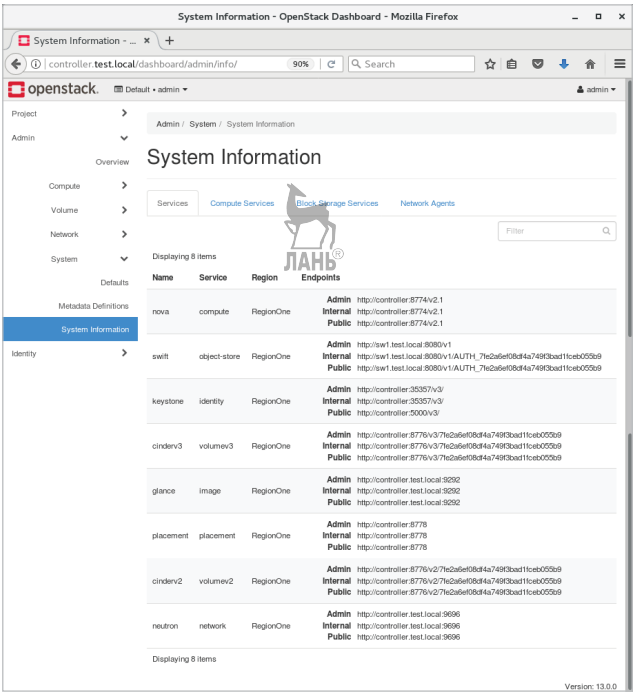


Рис. 11.3. Информация о системе в веб-клиенте Horizon

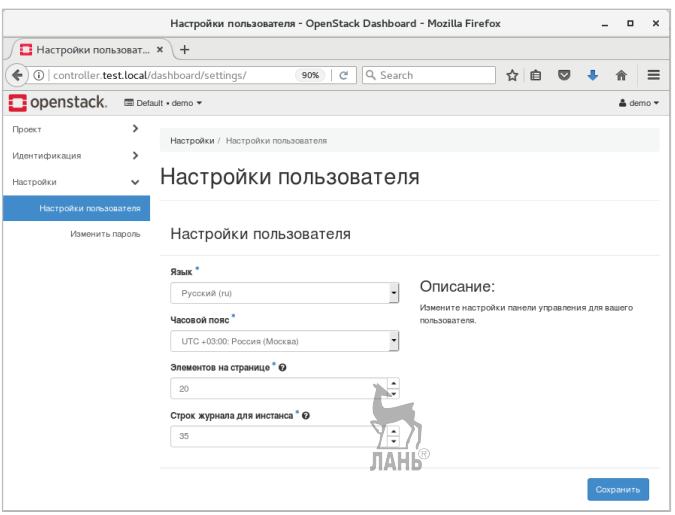


Рис. 11.4. Настройки интерфейса пользователя

Хотя в настоящее время «ванильный» OpenStack поставляется с двумя темами оформления Horizon (default и material), некоторые из дистрибутивов OpenStack поставляются со своей «корпоративной» темой оформления. Например, на рис. 11.5 приведено оформление дистрибутива Red Hat Enterprise Linux OpenStack Platform, а на рис. 11.6 – внешний вид веб-клиента Ericsson Cloud Execution Environment.

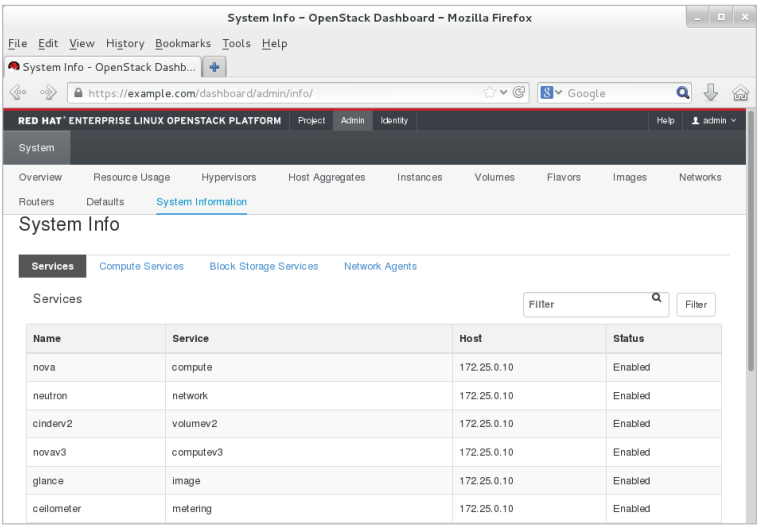


Рис. 11.5. Тема оформления Horizon в Red Hat OpenStack Platform

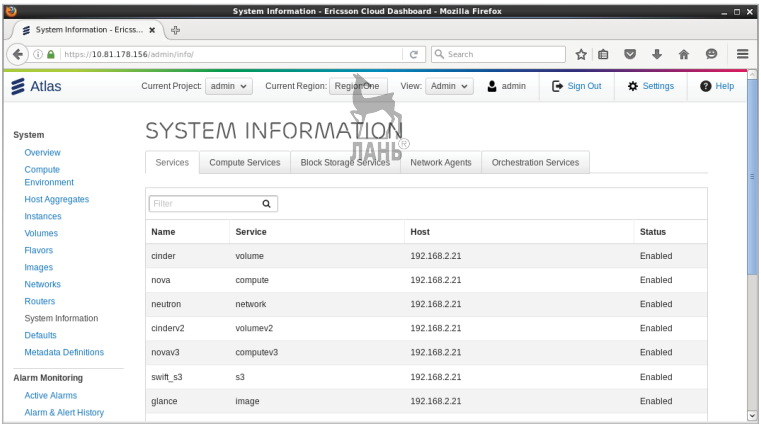


Рис. 11.6. Внешний вид веб-клиента Ericsson Cloud Execution

Environment (Atlas)

Привести оформление к стандартному в таких случаях можно, просто удалив соответствующий пакет. Для Red Hat OpenStack Platform это можно сделать командой

```
# rpm -e openstack-dashboard-theme --nodeps
```

Canonical также поставляет свою тему оформления. В случае с Ubuntu:

```
# apt-get remove --auto-remove openstack-dashboard-ubuntu-theme
```

Вообще, ряд изменений интерфейса можно сделать и через конфигурационный файл Horizon. Например, можно добавить в заголовок окна браузера название своей компании или имя облачного сервиса:

```
SITE_BRANDING = "Мое облако"
```

Подробнее про настройку интерфейса и темы оформления можно почитать в официальной документации: http://docs.openstack.org/newton/config-reference/dashboard/config-options.html

На рис. 11.7 в качестве справочной информации приведены все основные меню **Horizon**.

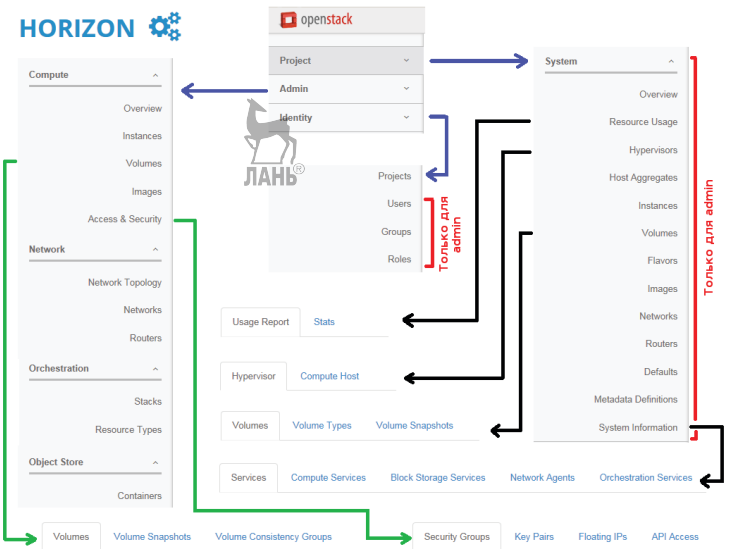


Рис. 11.7. Диаграмма меню **Horizon**

Глава 12

Сервис сбора телеметрии

```
Название: OpenStack Telemetry
Назначение: сбор информации об использовании ресурсов облака
Пакеты: openstack-ceilometer-*, openstack-aodh-*
Имена сервисов: openstack-ceilometer-*, openstack-aodh-*,
openstack-gnocchi-*
Порты: 8041,8042
Конфигурационные файлы: /etc/ceilometer/ceilometer.conf, /etc/
aodh/aodh.conf, /etc/gnocchi/gnocchi.conf
Файлы журнала: /var/log/ceilometer/*, /var/log/gnocchi/
```

Сервис OpenStack Telemetry – компонент облака OpenStack, отвечающий за сбор, хранение метрик (как самостоятельно в прошлом, так и используя новый бэкэнд Gnocchi в настоящее время) и мониторинг использования ресурсов в первую очередь для целей биллинга. Помимо сбора метрик работы облака, Ceilometer также собирает информацию о событиях, происходящих в работающей системе. Название сервиса Ceilometer происходит от названия прибора, используемого метеорологами для измерения высоты облаков над поверхностью земли.

Сервис телеметрии в настоящий момент состоит из нескольких отдельных проектов:

- **Ceilometer** – отвечает за мониторинг и сбор данных;
- **Aodh** – отвечает за обработку триггеров (alarm). Ранее данный функционал входил в обязанности Ceilometr, соответственно, в первом и втором изданиях книги он не рассматривался;
- **Gnocchi** – бэкэнд для хранения метрик, собранных сервисом телеметрии. Причиной его появления стало желание снизить накладные расходы и обеспечить возможность хранения данных на облачных масштабируемых файловых системах, таких как Ceph. Ceilometer может хранить данные как в Gnocchi, так и в MongoDB (устаревший функционал).

Сервис спроектирован как расширяемый за счет подключаемых агентов сбора информации и легко масштабируемый горизонтально. Ceilometer поддерживает два способа сбора данных. Предпочтительный метод сбора – при помощи очереди сообщений. Реализуется сервисом ceilometer-collector. Данный сервис запускается на одном или более управляющих узлах и отслеживает очередь сообщений. Сборщик получает уведомления от инфраструктурных сервисов (Nova, Glance, Cinder, Neutron, Swift, Keystone, Heat), затем преобразует их в сообщения телеметрии и отправляет обратно в очередь сообщений. Сообщения телеметрии записываются в хранилище без преобразований. Второй, менее предпочтительный способ – через опрашивающие инфраструктуру агенты. При помощи вызовов API или других инструментов агенты периодически запрашивают у сервисов необходимую информацию.

В релизе Havana также появилась возможность запуска событий по срабатываниям триггеров при достижении метрикой заданного значения. В качестве действия может быть задано обращение по HTTP на определенный адрес или запись события в журнал. В цикле разработки Liberty данный сервис был вынесен в отдельный проект Aodh.

Сервис состоит из ряда компонентов, часть из которых запускается на управляющих узлах, а часть – на вычислительных. Перечислим их:

- Gnocchi (**openstack-gnocchi-api** и **openstack-gnocchi-metricd**) – используется для хранения данных и вычисления в реальном времени статистики. Данные индексируются для возможности быстрого их получения. Индексы хранятся в реляционной базе данных. Мы будем использовать MariaDB;
- **openstack-ceilometer-notification** – агент, отправляющий по протоколу AMQP метрики сборщику от различных сервисов;
- **openstack-ceilometer-central** – агент, запускаемый на центральном сервере для запроса статистики по загрузке, не связанной с экземплярами виртуальных машин или вычислительными узлами. В целях горизонтального масштабирования допускается запуск нескольких агентов;
- **openstack-ceilometer-compute** – агент, запускаемый на всех вычислительных узлах для сбора статистики по узлам и экземплярам виртуальных машин;

- API-сервер **openstack-aodh-api** – запускается на одном или более узлах. Служит для предоставления доступа к информации о сработавших триггерах (alarms);
- **openstack-aodh-evaluator** – сервис, определяющий, сработал ли триггер при достижении метриками заданных значений в течение определенного измеряемого периода;
- **openstack-aodh-notifier** – сервис, запускающий те или иные действия при срабатывании триггера;
- **openstack-aodh-listener** – сервис, определяющий, когда триггер сработает. Срабатывание определяется сравнением правил триггера и событий, полученных агентами сбора телеметрии.

Список собираемых метрик с описаниями можно взять с официального сайта в руководстве администратора облака OpenStack – http://docs.openstack.org/admin-guide/telemetry.html. Метрики могут быть трех типов:

- накопительные счетчики (cumulative) – постоянно увеличивающиеся со временем значения;
- индикаторы (gauge) – дискретные и плавающие значения, например ввод/вывод диска или присвоенные «плавающие IP»;
- дельта (delta) – изменение со временем, например пропускная способность сети.

Некоторые метрики собираются раз в тридцать минут, для иных требуется более частая периодичность, например раз в минуту.

Установка служб Gnocchi и Ceilometr управляющего узла

Вы можете установить сервер API, сборщик сообщений, центральный агент и базу на разные узлы. Мы все эти сервисы будем запускать на управляющем узле, а на вычислительном узле установим соответствующий агент вычислительного узла. Также нам понадобится внести изменения в конфигурационные файлы других служб, например Glance, для того чтобы они начали передавать информацию о событиях в шину брокера сообщений.

Устанавливаем пакеты openstack-gnocchi-* и клиента командной строки:

```
[root@controller ~]# yum -y install openstack-gnocchi-api
openstack-gnocchi-metricd python-gnocchiclient
```

Продолжаем установку уже привычными командами из прошлых глав. Создаем пользователей ceilometer и gnocchi:

```
$ source keystonerc_admin
$ openstack user create --domain default --password openstack
ceilometer
+---------------------+----------------------------------+
| Field               | Value                            |
+---------------------+----------------------------------+
| domain_id           | default                          |
| enabled             | True                             |
| id                  | e7fbdb6eaecc493e9f1f60a37c52e518 |
| name                | ceilometer                       |
| options             | {}                               |
| password_expires_at | None                             |
+---------------------+----------------------------------+
$ openstack user create --domain default --password openstack
gnocchi
+---------------------+----------------------------------+
| Field               | Value                            |
+---------------------+----------------------------------+
| domain_id           | default                          |
| enabled             | True                             |
| id                  | 4e27f716410d45848c0e84a1f2deb3f8 |
| name                | gnocchi                          |
| options             | {}                               |
| password_expires_at | None                             |
+---------------------+----------------------------------+
```

Затем добавляем пользователей ceilometer и gnocchi в проект service с ролью admin:

```
$ openstack role add --project service --user ceilometer admin
$ openstack role add --project service --user gnocchi admin
```

Создаем сервис gnocchi:

```
$ openstack service create --name gnocchi --description "Metric
Service" metric
+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description | Metric Service                   |
| enabled     | True                             |
| id          | 27c82194ea0844ed8fbd59e9798e3337 |
| name        | gnocchi                          |
| type        | metric                           |
+-------------+----------------------------------+
```

Создаем точки входа в сервис:

```
$ openstack endpoint create --region RegionOne metric public
http://controller.test.local:8041
+--------------+----------------------------------+
| Field        | Value                            |
+--------------+----------------------------------+
| enabled      | True                             |
| id           | 52e5ffb0065d4b67be8692446a2df32b |
| interface    | public                           |
| region       | RegionOne                        |
| region_id    | RegionOne                        |
| service_id   | 27c82194ea0844ed8fbd59e9798e3337 |
| service_name | gnocchi                          |
| service_type | metric                           |
| url          | http://controller.test.local:8041 |
+--------------+----------------------------------+
$ openstack endpoint create --region RegionOne metric internal
http://controller.test.local:8041
+--------------+----------------------------------+
| Field        | Value                            |
+--------------+----------------------------------+
| enabled      | True                             |
| id           | 8eb125cc5b3d412aafa74e308ad9a53a |
| interface    | internal                         |
| region       | RegionOne                        |
| region_id    | RegionOne                        |
| service_id   | 27c82194ea0844ed8fbd59e9798e3337 |
| service_name | gnocchi                          |
| service_type | metric                           |
| url          | http://controller.test.local:8041 |
+--------------+----------------------------------+
$ openstack endpoint create --region RegionOne metric admin
http://controller.test.local:8041
+--------------+----------------------------------+
| Field        | Value                            |
+--------------+----------------------------------+
| enabled      | True                             |
| id           | cd65af6592764986ab18691cb276cacd |
| interface    | admin                            |
| region       | RegionOne                        |
| region_id    | RegionOne                        |
| service_id   | 27c82194ea0844ed8fbd59e9798e3337 |
| service_name | gnocchi                          |
| service_type | metric                           |
| url          | http://controller.test.local:8041 |
+--------------+----------------------------------+
```

Создадим базу для службы индекса Gnocchi:

```
[root@controller ~]# mysql -u root -p
MariaDB [(none)]> CREATE DATABASE gnocchi;
MariaDB [(none)]> GRANT ALL PRIVILEGES ON gnocchi.* TO
'gnocchi'@'localhost' IDENTIFIED BY 'openstack';
MariaDB [(none)]> GRANT ALL PRIVILEGES ON gnocchi.* TO
'gnocchi'@'%' IDENTIFIED BY 'openstack'
MariaDB [(none)]> exit
```

Указываем параметры службы идентификации Keystone:

```
[root@controller ~]# crudini --set /etc/gnocchi/gnocchi.conf api
auth_mode keystone
[root@controller ~]# crudini --set /etc/gnocchi/gnocchi.conf
keystone_authtoken auth_url http://controller.test.local:5000/v3
[root@controller ~]# crudini --set /etc/gnocchi/gnocchi.conf
keystone_authtoken auth_type password
[root@controller ~]# crudini --set /etc/gnocchi/gnocchi.conf
keystone_authtoken project_domain_name Default
[root@controller ~]# crudini --set /etc/gnocchi/gnocchi.conf
keystone_authtoken user_domain_name Default
[root@controller ~]# crudini --set /etc/gnocchi/gnocchi.conf
keystone_authtoken project_name service
[root@controller ~]# crudini --set /etc/gnocchi/gnocchi.conf
keystone_authtoken username gnocchi
[root@controller ~]# crudini --set /etc/gnocchi/gnocchi.conf
keystone_authtoken password openstack
[root@controller ~]# crudini --set /etc/gnocchi/gnocchi.conf
keystone_authtoken interface internalURL
[root@controller ~]# crudini --set /etc/gnocchi/gnocchi.conf
keystone_authtoken region_name RegionOne
```

Укажем параметры подключения к базе данных индексатора:

```
[root@controller ~]# crudini --set /etc/gnocchi/gnocchi.conf indexer
url mysql+pymysql://gnocchi:openstack@controller.test.local/gnocchi
```

Также укажем место на файловой системе, где будут храниться данные метрик. Локальная файловая система – самый простой для демонстрационного стенда способ. В документации приведены другие, более подходящие для промышленного применения варианты.

```
[root@controller ~]# crudini --set /etc/gnocchi/gnocchi.conf storage
file_basepath /var/lib/gnocchi
[root@controller ~]# crudini --set /etc/gnocchi/gnocchi.conf storage
driver file
```

Инициализируем gnocchi:

```
[root@controller ~]#  gnocchi-upgrade
```

Последним шагом перед переходом к установке пакетов Ceilometer будут включение и запуск установленных служб Gnocchi:

```
[root@controller ~]# systemctl enable openstack-gnocchi-api.service
openstack-gnocchi-metricd.service
[root@controller ~]# systemctl start openstack-gnocchi-api.service
openstack-gnocchi-metricd.service
```

На момент написания этой главы существовала не исправленная в пакетах RDO проблема – https://goo.gl/u7bbR6. Возможно, вам придется внести изменения в код.

Устанавливаем пакеты Ceilometer:

```
[root@controller ~]# yum -y install openstack-ceilometer-notification
openstack-ceilometer-central
```

Добавляем в конфигурационные файлы информацию по сервису RabbitMQ и реквизиты сервиса:

```
[root@controller ~]# crudini --set /etc/ceilometer/ceilometer.conf
DEFAULT transport_url rabbit://openstack:openstack@controller.test.
local
[root@controller ~]# crudini --set /etc/ceilometer/ceilometer.conf
service_credentials auth_type password
[root@controller ~]# crudini --set /etc/ceilometer/ceilometer.conf
service_credentials auth_url http://controller.test.local:5000/v3
[root@controller ~]# crudini --set /etc/ceilometer/ceilometer.conf
service_credentials project_domain_id default
[root@controller ~]# crudini --set /etc/ceilometer/ceilometer.conf
service_credentials user_domain_id default
[root@controller ~]# crudini --set /etc/ceilometer/ceilometer.conf
service_credentials project_name service
[root@controller ~]# crudini --set /etc/ceilometer/ceilometer.conf
service_credentials username ceilometer
[root@controller ~]# crudini --set /etc/ceilometer/ceilometer.conf
service_credentials password openstack
[root@controller ~]# crudini --set /etc/ceilometer/ceilometer.conf
service_credentials interface internalURL
[root@controller ~]# crudini --set /etc/ceilometer/ceilometer.conf
service_credentials region_name RegionOne
```

Создаем ресурсы в Gnocchi:

```
[root@controller ~]# ceilometer-upgrade
```

Включаем и стартуем сервисы:

```
[root@controller ~]# systemctl enable openstack-ceilometer-
notification.service openstack-ceilometer-central.service
[root@controller ~]# systemctl start openstack-ceilometer-
notification.service openstack-ceilometer-central.service
```

Установка службы триггеров Aodh

Создадим базу для хранения информации о триггерах:

```
[root@controller ~]# mysql -u root -p
MariaDB [(none)]> CREATE DATABASE aodh;
MariaDB [(none)]> GRANT ALL PRIVILEGES ON aodh.* TO
'aodh'@'localhost' IDENTIFIED BY 'openstack';
MariaDB [(none)]> GRANT ALL PRIVILEGES ON aodh.* TO 'aodh'@'%'
IDENTIFIED BY 'openstack';
MariaDB [(none)]> exit
```

Как всегда, создаем пользователя, сервис и конечные точки сервиса:

```
$ openstack user create --domain default --password openstack aodh
+---------------------+----------------------------------+
| Field               | Value                            |
+---------------------+----------------------------------+
| domain_id           | default                          |
| enabled             | True                             |
| id                  | a1b44510220145da83c70f4a9e9706d8 |
| name                | aodh                             |
| options             | {}                               |
| password_expires_at | None                             |
+---------------------+----------------------------------+
$ openstack role add --project service --user aodh admin
$ openstack service create --name aodh --description "Telemetry"
alarming
+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description | Telemetry                        |
| enabled     | True                             |
| id          | 674b422d44fa435c9c05f5bc79293d67 |
| name        | aodh                             |
| type        | alarming                         |
+-------------+----------------------------------+
$ openstack endpoint create --region RegionOne alarming public
http://controller.test.local:8042
+--------------+----------------------------------+
| Field        | Value                            |
+--------------+----------------------------------+
| enabled      | True                             |
```

```
| id           | 07cfee43478541918f07b0b6a10c7a34  |
| interface    | public                            |
| region       | RegionOne                         |
| region_id    | RegionOne                         |
| service_id   | 674b422d44fa435c9c05f5bc79293d67  |
| service_name | aodh                              |
| service_type | alarming                          |
| url          | http://controller.test.local:8042 |
+--------------+-----------------------------------+
$ openstack endpoint create --region RegionOne alarming internal
http://controller.test.local:8042
+--------------+-----------------------------------+
| Field        | Value                             |
+--------------+-----------------------------------+
| enabled      | True                              |
| id           | 5613210263ae43fc8716520b1a645905  |
| interface    | internal                          |
| region       | RegionOne                         |
| region_id    | RegionOne                         |
| service_id   | 674b422d44fa435c9c05f5bc79293d67  |
| service_name | aodh                              |
| service_type | alarming                          |
| url          | http://controller.test.local:8042 |
+--------------+-----------------------------------+
$ openstack endpoint create --region RegionOne alarming admin
http://controller.test.local:8042
+--------------+-----------------------------------+
| Field        | Value                             |
+--------------+-----------------------------------+
| enabled      | True                              |
| id           | 6ebc9226ba144a058b51090d2418d369  |
| interface    | admin                             |
| region       | RegionOne                         |
| region_id    | RegionOne                         |
| service_id   | 674b422d44fa435c9c05f5bc79293d67  |
| service_name | aodh                              |
| service_type | alarming                          |
| url          | http://controller.test.local:8042 |
+--------------+-----------------------------------+
```

Устанавливаем необходимые пакеты:

```
# yum -y install openstack-aodh-api openstack-aodh-evaluator
openstack-aodh-notifier openstack-aodh-listener openstack-aodh-expirer
python-aodhclient
```

Далее необходимо добавить в файл /etc/aodh/aodh.conf настройки сервиса идентификации Keystone и указать, с какими реквизитами подключается пользователь aodh.

```
[root@controller ~]# crudini --set /etc/aodh/aodh.conf DEFAULT
auth_strategy keystone
[root@controller ~]# crudini --set /etc/aodh/aodh.conf keystone_
authtoken auth_uri http://controller.test.local:5000
[root@controller ~]# crudini --set /etc/aodh/aodh.conf keystone_
authtoken auth_url http://controller.test.local:35357
[root@controller ~]# crudini --set /etc/aodh/aodh.conf keystone_
authtoken auth_type password
[root@controller ~]# crudini --set /etc/aodh/aodh.conf keystone_
authtoken project_domain_id default
[root@controller ~]# crudini --set /etc/aodh/aodh.conf keystone_
authtoken user_domain_id default
[root@controller ~]# crudini --set /etc/aodh/aodh.conf keystone_
authtoken project_name service
[root@controller ~]# crudini --set /etc/aodh/aodh.conf keystone_
authtoken username aodh
[root@controller ~]# crudini --set /etc/aodh/aodh.conf keystone_
authtoken password openstack
```

Те же действия для самого сервиса:

```
[root@controller ~]# crudini --set /etc/aodh/aodh.conf service_
credentials auth_type password
[root@controller ~]# crudini --set /etc/aodh/aodh.conf service_
credentials auth_url http://controller.test.local:5000/v3
[root@controller ~]# crudini --set /etc/aodh/aodh.conf service_
credentials project_domain_id default
[root@controller ~]# crudini --set /etc/aodh/aodh.conf service_
credentials user_domain_id default
[root@controller ~]# crudini --set /etc/aodh/aodh.conf service_
credentials project_name service
[root@controller ~]# crudini --set /etc/aodh/aodh.conf service_
credentials username aodh
[root@controller ~]# crudini --set /etc/aodh/aodh.conf service_
credentials password openstack
[root@controller ~]# crudini --set /etc/aodh/aodh.conf service_
credentials interface internalURL
[root@controller ~]# crudini --set /etc/aodh/aodh.conf service_
credentials region_name RegionOne
```

Помимо этого, укажем параметры подключения к базе данных:

```
[root@controller ~]# crudini --set /etc/aodh/aodh.conf database
connection mysql+pymysql://aodh:openstack@controller.test.local/aodh
```

И добавим в конфигурационные файлы информацию по сервису RabbitMQ:

```
[root@controller ~]# crudini --set /etc/aodh/aodh.conf DEFAULT
transport_url rabbit://openstack:openstack@controller.test.local
```

Для инициализации базы данных запускаем скрипт:

```
[root@controller ~]# aodh-dbsync
```

Теперь можно включить и стартовать сервисы:

```
[root@controller ~]# systemctl enable openstack-aodh-api.service
openstack-aodh-evaluator.service openstack-aodh-notifier.service
openstack-aodh-listener.service
[root@controller ~]# systemctl start openstack-aodh-api.service
openstack-aodh-evaluator.service openstack-aodh-notifier.service
openstack-aodh-listener.service
```

Установка служб вычислительного узла для отправки сообщений телеметрии

Переходим к вычислительным узлам. Дальнейшие действия необходимо выполнить на обоих узлах: compute и compute-opt. Тут количество устанавливаемых пакетов значительно меньше:

```
[root@compute ~]# yum -y install openstack-ceilometer-compute
```

Затем добавляем в конфигурационные файлы информацию по сервису RabbitMQ и реквизиты сервиса:

```
[root@compute ~]# crudini --set /etc/ceilometer/ceilometer.conf DEFAULT
transport_url rabbit://openstack:openstack@controller.test.local
[root@compute ~]# crudini --set /etc/ceilometer/ceilometer.conf
service_credentials auth_type password
[root@compute ~]# crudini --set /etc/ceilometer/ceilometer.conf
service_credentials auth_url http://controller.test.local:5000/v3
[root@compute ~]# crudini --set /etc/ceilometer/ceilometer.conf
service_credentials project_domain_id default
[root@compute ~]# crudini --set /etc/ceilometer/ceilometer.conf
service_credentials user_domain_id default
[root@compute ~]# crudini --set /etc/ceilometer/ceilometer.conf
service_credentials project_name service
[root@compute ~]# crudini --set /etc/ceilometer/ceilometer.conf
service_credentials username ceilometer
[root@compute ~]# crudini --set /etc/ceilometer/ceilometer.conf
service_credentials password openstack
[root@compute ~]# crudini --set /etc/ceilometer/ceilometer.conf
service_credentials interface internalURL
[root@compute ~]# crudini --set /etc/ceilometer/ceilometer.conf
service_credentials region_name RegionOne
```

Отредактируем файл настроек сервиса Nova, для того чтобы сервис начал отправлять сообщения через брокер сообщений:

```
[root@compute ~]# crudini --set /etc/nova/nova.conf DEFAULT
instance_usage_audit_period hour
[root@compute ~]# crudini --set /etc/nova/nova.conf DEFAULT
notify_on_state_change vm_and_task_state
[root@compute ~]# crudini --set /etc/nova/nova.conf oslo_
messaging_notifications driver messagingv2
```

Для сбора статистики по «memory.usage», «disk.usage» и «disk.device.usage» необходимо включить параметр `instance_usage_audit`:

```
[root@compute ~]# crudini --set /etc/nova/nova.conf DEFAULT
instance_usage_audit True
```

Теперь нужно включить и стартовать сервис Ceilometer, а также перезапустить Nova:

```
[root@compute ~]# systemctl enable openstack-ceilometer-compute.
service
[root@compute ~]# systemctl start openstack-ceilometer-compute.
service
[root@compute ~]# systemctl restart openstack-nova-compute.
service
```

Интеграция с сервисами Glance и Cinder

В качестве примера настроим отправку сообщений в сервис телеметрии для сервисов Glance и Cinder. Для информации о настройке других служб автор переадресует читателя к официальной документации.

На узле, где запущены сервисы glance-api и glance-registry, необходимо отредактировать конфигурационные файлы, настроив службу на отправку сообщений для Ceilometer по AMQP:

```
[root@controller ~]# crudini --set /etc/glance/glance-api.conf
oslo_messaging_notifications driver messagingv2
[root@controller ~]# crudini --set /etc/glance/glance-api.conf.
conf oslo_messaging_notifications driver messagingv2
[root@controller ~]# crudini --set /etc/glance/glance-api.conf
DEFAULT transport_url rabbit://openstack:openstack@controller.
test.local
[root@controller ~]# crudini --set /etc/glance/glance-registry.conf
DEFAULT transport_url rabbit://openstack:openstack@controller.test.
local
```

Затем рестартуйте обе службы: openstack-glance-api и openstack-glance-registry:

```
# systemctl restart openstack-glance-api.service openstack-glance-
registry.service
```

Завершим установку настройкой блочного хранилища Cinder. Изменения необходимо произвести как на узлах хранения, так и на управляющем узле. В нашем лабораторном окружении обе роли исполняет узел controller.test.local:

```
[root@controller ~]# crudini --set /etc/cinder/cinder.conf oslo_
messaging_notifications driver messagingv2
```

Кроме того, необходимо создать периодически запускаемую работу в cron для сбора информации. Например, раз в пять минут:

```
*/5 * * * * /path/to/cinder-volume-usage-audit --send_actions
```

Для завершения настройки следует рестартовать сервисы Cinder:

```
[root@controller ~]# systemctl restart openstack-cinder-api.service
openstack-cinder-scheduler.service openstack-cinder-volume.service
```

Теперь перейдем непосредственно к работе с сервисом.

Работа со службой телеметрии в современных версиях OpenStack

Посмотрим, какие типы ресурсов доступны:

```
$ openstack metric resource-type list -c name
+----------------------------+
| name                       |
+----------------------------+
| ceph_account               |
| generic                    |
| host                       |
| host_disk                  |
| host_network_interface     |
| identity                   |
| image                      |
| instance                   |
| instance_disk              |
| instance_network_interface |
| ipmi                       |
| manila_share               |
```

```
| network                    |
| nova_compute               |
| port                       |
| stack                      |
| swift_account              |
| switch                     |
| switch_port                |
| switch_table               |
| volume                     |
| volume_provider            |
| volume_provider_pool       |
+----------------------------+
```

Далее посмотрим, какие ресурсы доступны текущему пользователю:

```
$ openstack metric resource list -c type -c id
+--------------------------------------+---------------------------+
| id                                   | type                      |
+--------------------------------------+---------------------------+
| 484ee4ef-84bd-4f12-a093-3b62c56c29f0 | instance                  |
| 29ddb3c7-b418-557f-90e2-f7a3f90d20ff | instance_disk             |
...
| 671a756b-0423-4300-a0c7-5b03b621a1e8 | instance                  |
| ae6ad5eb-58b6-50f9-9149-a4b603110b67 | instance_network_interface |
| c10229e0-1dc2-543b-b272-26f5c9708310 | instance_disk             |
| 35923211-ec9d-4f0a-8d5c-2cec96a427d8 | instance                  |
| e36d78b3-b767-57a5-a839-8f5d7cd752ce | instance_disk             |
| bead1067-20dc-58db-a785-bc110bc6e0fb | instance_network_interface |
+--------------------------------------+---------------------------+
```

Как мы видим, присутствует несколько ресурсов типа instance. Можно сравнить и убедиться, что их ID совпадают с ID двух имеющихся в наличии экземпляров виртуальных машин:

```
$ openstack server list -c ID -c Name
+--------------------------------------+-------+
| ID                                   | Name  |
+--------------------------------------+-------+
| 35923211-ec9d-4f0a-8d5c-2cec96a427d8 | myvm2 |
| 671a756b-0423-4300-a0c7-5b03b621a1e8 | myvm1 |
+--------------------------------------+-------+
```

Можно посмотреть все метрики, ассоциированные, например, с экземпляром myvm1:

```
$ openstack metric resource show 35923211-ec9d-4f0a-8d5c-2cec96a427d8
+----------------------+-------------------------------------------------+
| Field                | Value                                           |
+----------------------+-------------------------------------------------+
```

```
| created_by_project_id | c19f44553ab640779672e2321364e6ce                  |
| created_by_user_id    | e7fbdb6eaecc493e9f1f60a37c52e518                  |
| creator               | e7fbdb6eaecc493e9f1f60a37c52e518:c19f44553ab64.. |
| ended_at              | None                                              |
| id                    | c418c348-94f9-4d99-981b-e41e7d34d286              |
| metrics               | compute.instance.booting.time: 81da677f-1e28-4.. |
|                       | cpu.delta: d35aab34-3347-4701-9aaa-3fce10765d93   |
|                       | cpu: 2580fa7a-8e78-4514-b445-f118d419970a         |
|                       | cpu_l3_cache: 7dd54beb-4ee0-4ea3-99a3-45e3e33a8   |
|                       | cpu_util: df31a8d0-ce7a-4f71-8506-fd1acb30431b    |
|                       | disk.allocation: 793c5989-1448-4bff-9aef-9431e.. |
|                       | disk.capacity: 8b7a4034-52a1-4340-9775-258bab3.. |
|                       | disk.ephemeral.size: b27317f0-3077-4f92-aad3-6.. |
|                       | disk.iops: f065823f-bc19-4c46-a1fd-f96426ee793d   |
|                       | disk.latency: 8b1d2adb-3d4e-47b8-847b-61f7026b.. |
|                       | disk.read.bytes.rate: df2dd949-bc30-4316-8872-.. |
|                       | disk.read.bytes: ff0d9068-cc01-4e39-999e-a070b.. |
|                       | disk.read.requests.rate: 1a38d66f-81ae-445e-b7.. |
|                       | disk.read.requests: 627bc618-dbea-494b-a57c-12.. |
|                       | disk.root.size: 0fa5905f-18de-4408-828b-5dea0c.. |
|                       | disk.usage: d279c119-42d3-4cea-8a0e-3df2723b34.. |
|                       | disk.write.bytes.rate: 53c9e49e-15f2-4fe2-8613.. |
|                       | disk.write.bytes: 33d782b3-bda9-44b3-aca0-7ffa.. |
|                       | disk.write.requests.rate: bc9406c9-e677-48e1-8.. |
|                       | disk.write.requests: c482e1f1-b334-495b-872c-1.. |
|                       | memory.bandwidth.local: e2ebd189-242c-4e8c-b2a.. |
|                       | memory.bandwidth.total: 8c8e5bf3-2656-4505-b4b.. |
|                       | memory.resident: 7f497d55-e3f9-4546-83fb-44093.. |
|                       | memory.swap.in: 41935ec0-6592-4eac-a364-60b598.. |
|                       | memory.swap.out: d52a0774-4c57-447a-b31d-d20f8.. |
|                       | memory.usage: 7feb9c6f-1e95-4ec8-9b58-3f665bcf.. |
|                       | memory: 9cc65602-e775-44f4-a0de-51ff1ca492b4      |
|                       | perf.cache.misses: 78c07416-b67b-4292-a313-83f.. |
|                       | perf.cache.references: 88b5e885-68c9-4158-bec5.. |
|                       | perf.cpu.cycles: bbd56f3d-2fda-4484-b358-e269f.. |
|                       | perf.instructions: 3e3e2709-e487-4193-bed9-ab3.. |
|                       | vcpus: 7bb09bfd-394e-4b3c-b887-322002d14292       |
| original_resource_id  | 35923211-ec9d-4f0a-8d5c-2cec96a427d8              |
| project_id            | bc10ac4b71164550a363b8098e8ad270                  |
| revision_end          | None                                              |
| revision_start        | 2018-03-18T13:41:32.690577+00:00                  |
| started_at            | 2018-03-18T13:41:32.690550+00:00                  |
| type                  | instance                                          |
| user_id               | 3b76dece42b140e092dc1a76a85c1879                  |
+-----------------------+---------------------------------------------------+
```

Далее, используя идентификатор ресурса, можно смотреть конкретную метрику, ассоциированную с экземпляром виртуальной машины:

```
$ openstack metric measures show --resource-id 35923211-ec9d-4f0a-
8d5c-2cec96a427d8 cpu_util
+---------------------------+-------------+--------------+
| timestamp                 | granularity |        value |
+---------------------------+-------------+--------------+
| 2018-03-18T14:50:00+01:00 |       300.0 | 0.3566421273 |
| 2018-03-18T14:55:00+01:00 |       300.0 | 0.4333532502 |
| 2018-03-18T15:00:00+01:00 |       300.0 | 0.4066271235 |
| 2018-03-18T15:05:00+01:00 |       300.0 | 0.4033385404 |
| 2018-03-18T15:10:00+01:00 |       300.0 | 0.3933495747 |
+---------------------------+-------------+--------------+
```

Попробуем познакомиться с триггерами или сигналами оповещения (alarms). Триггер может быть в трех состояниях: «Ok», «Тревога» и «Недостаточно данных». Правила триггеров могут объединяться при помощи операторов AND и OR. К триггеру можно привязать определенное правило, которое чаще всего HTTP-команда POST использует для заданного URL. Можно также в качестве действия выбрать запись в файл журнала, но это применяется только для отладки, поскольку необходим доступ с правами администратора облака.

Создадим триггер для виртуальной машины с ID 35923211-ec9d-4f0a-8d5c-2cec96a427d8:

```
$ openstack alarm create --type gnocchi_aggregation_by_resources_threshold
--name myalarm1 --metric cpu_util --aggregation-method mean --comparison-
operator 'ge' --threshold 60 --evaluation-periods 2 --granularity 300 --alarm-
action 'log://' --resource-type instance --query '{"=": {"id": "35923211-ec9d-
4f0a-8d5c-2cec96a427d8"}}'
+---------------------------+------------------------------------------------+
| Field                     | Value                                          |
+---------------------------+------------------------------------------------+
| aggregation_method        | mean                                           |
| alarm_actions             | [u'log://']                                    |
| alarm_id                  | 5db57222-97ec-4dde-baa3-a2ca3fac472e           |
| comparison_operator       | ge                                             |
| description               | gnocchi_aggregation_by_resources_threshold alarm rule |
| enabled                   | True                                           |
| evaluation_periods        | 2                                              |
| granularity               | 300                                            |
| insufficient_data_actions | []                                             |
| metric                    | cpu_util                                       |
| name                      | myalarm1                                       |
| ok_actions                | []                                             |
| project_id                | 7fe2a6ef08df4a749f3bad1fceb055b9               |
| query                     | {"=": {"id": "35923211-ec9d-4f0a-8d5c-2cec96a427d8"}} |
| repeat_actions            | False                                          |
| resource_type             | instance                                       |
```

```
| severity                | low                                                  |
| state                   | insufficient data                                    |
| state_reason            | Not evaluated yet                                    |
| state_timestamp         | 2018-03-18T15:19:28.306432                           |
| threshold               | 60.0                                                 |
| time_constraints        | []                                                   |
| timestamp               | 2018-03-18T15:19:28.306432                           |
| type                    | gnocchi_aggregation_by_resources_threshold           |
| user_id                 | a37a67fdf3dc4e9c9e5251754b26770d                     |
+-------------------------+------------------------------------------------------+
```

Данный триггер сработает, если средняя загрузка виртуального процессора экземпляра превысит 60% в течение двух промежутков измерений по 300 секунд. Проверим, как последовательно статус сменится с «Недостаточно данных» на «Ok»:

```
$ openstack alarm list -c name -c state -c enabled
+----------+------------------+---------+
| name     | state            | enabled |
+----------+------------------+---------+
| myalarm1 | insufficient data  | True    |
+----------+------------------+---------+
$ openstack alarm list -c name -c state -c enabled
+----------+-------+---------+
| name     | state | enabled |
+----------+-------+---------+
| myalarm1 | ok    | True    |
+----------+-------+---------+
```

Теперь можно нагрузить процессор виртуальной машины, подключившись к ней и запустив команду

```
$ md5sum /dev/zero
```

Через некоторое время проверим загрузку и убедимся, что триггер сработал:

```
$ openstack metric measures show --resource-id 35923211-ec9d-
4f0a-8d5c-2cec96a427d8 cpu_util
+---------------------------+-------------+---------------+
| timestamp                 | granularity |         value |
+---------------------------+-------------+---------------+
| 2018-03-18T14:50:00+01:00 |       300.0 |  0.3566421273 |
...
| 2018-03-18T16:25:00+01:00 |       300.0 |  0.3933242633 |
| 2018-03-18T16:30:00+01:00 |       300.0 |  0.3866072657 |
| 2018-03-18T16:35:00+01:00 |       300.0 | 94.2390209724 |
| 2018-03-18T16:40:00+01:00 |       300.0 | 99.6113294916 |
+---------------------------+-------------+---------------+
```

```
$ openstack alarm list -c name -c state -c enabled
+----------+-------+---------+
| name     | state | enabled |
+----------+-------+---------+
| myalarm1 | alarm | True    |
+----------+-------+---------+
```

Работа со службой телеметрии Ceilometer в версиях Newton и ранее

Если вы по каким-то причинам вынуждены работать с более старыми версиями OpenStack, в этом разделе приведены примеры работы с использованием устаревшего клиента ceilometer.

В веб-интерфейсе OpenStack за отображение информации Ceilometer отвечают две вкладки на странице **Admin → System → Resource Usage**. Намного больше возможностей предоставляет клиент командной строки ceilometer, который использует RESTful API.

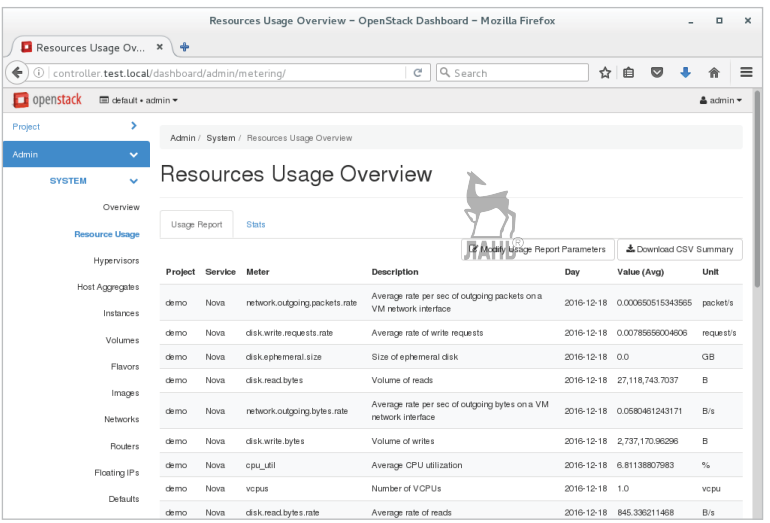


Рис. 12.1. Пример данных Ceilometer в веб-интерфейсе (OpenStack Newton)

Попробуем запросить метрики телеметрии, для которых уже есть записи в базе данных:

```
$ ceilometer meter-list
+------------------------------+------------+----------+-------------+---------+------------+
```

```
| Name                          | Type       | Unit     | Resource ID | User ID | Project ID |
+-------------------------------+------------+----------+-------------+---------+------------+
| compute.instance.booting.time | gauge      | sec      | 877b0a5e-.. | 372dd.. | 9c1258e1.. |
| cpu                           | cumulative | ns       | 877b0a5e-.. | 372dd.. | 9c1258e1.. |
| disk.allocation               | gauge      | B        | 877b0a5e-.. | 372dd.. | 9c1258e1.. |
| disk.capacity                 | gauge      | B        | 877b0a5e-.. | 372dd.. | 9c1258e1.. |
| disk.device.allocation        | gauge      | B        | 877b0a5e-.. | 372dd.. | 9c1258e1.. |
| instance                      | gauge      | instance | 877b0a5e-.. | 372dd.. | 9c1258e1.. |
| memory                        | gauge      | MB       | 877b0a5e-.. | 372dd.. | 9c1258e1.. |
| memory.resident               | gauge      | MB       | 877b0a5e-.. | 372dd.. | 9c1258e1.. |
| network.incoming.bytes        | cumulative | B        | instance-5  | 372dd.. | 9c1258e1.. |
...
+-------------------------------+------------+----------+-------------+---------+------------+
```

Нужно обратить внимание, что данная команда выведет данные, только если они были предварительно собраны. Поэтому если вы запускаете команду `ceilometer` первый раз после установки сервиса, запустите при помощи nova boot хотя бы одну виртуальную машину.

Ограничить вывод по конкретному ресурсу, например по определенному экземпляру виртуальной машины, можно, добавив опции --query resource=<ID виртуальной машины>.

Можно также попробовать вывести все измерения для заданной метрики:

```
$ceilometer sample-list -m memory
+--------------+--------+-------+--------+------+---------------------------+
| Resource ID  | Name   | Type  | Volume | Unit | Timestamp                 |
+--------------+--------+-------+--------+------+---------------------------+
| 877b0a5e-e.. | memory | gauge | 300.0  | MB   | 2016-12-17T19:36:48.956000 |
| 877b0a5e-e.. | memory | gauge | 300.0  | MB   | 2016-12-17T19:36:19.067000 |
+--------------+--------+-------+--------+------+---------------------------+
```

Или получить статистику, опять же в разрезе заданной метрики:

```
$ ceilometer statistics -m memory
+--------+--------------+--------------+------+------+------+-------+------+----------+----------------+--------------+
| Period | Period Start | Period End   | Max  | Min  | Avg  | Sum   | Count | Duration | Duration Start | Duration End |
+--------+--------------+--------------+------+------+------+-------+------+----------+----------------+--------------+
| 0      | 2016-12-17.. | 2016-12-17.. | 300.0 | 300.0 | 300.0 | 1800.0 | 6    | 145.624  | 2016-12-17T0.. | 2016-12-17T.. |
+--------+--------------+--------------+------+------+------+-------+------+----------+----------------+--------------+
```

Более сложные запросы можно создавать при помощи команды `ceilometer query-samples`, используя опции `--filter`, `--orderby` и `--limit`.

Данные, собранные сервисом ceilometer-collector, также можно отправлять в различные внешние сервисы и системы при помощи

механизма публикаций. За эти настройки отвечает файл pipeline.yaml, находящийся в директории /etc/ceilometer. Для дальнейших экспериментов поправим в этом файле периодичность снятия метрик по процессору с десяти минут до одной минуты. Соответствующая часть файла должна выглядеть так:

```
  - name: cpu_source
    interval: 60
```

Попробуем создать простой триггер для виртуальной машины. Для начала запустим экземпляр:

```
$ source keystonerc_demo
$ nova boot --flavor m2.tiny --image cirros-0.3.4-x86_64 --key-name demokey1
--security-groups demo-sgroup mytest1
$ nova add-floating-ip mytest1 10.100.1.110
$ nova list
+-------+---------+--------+------------+-------------+----------------------+
| ID    | Name    | Status | Task State | Power State | Networks             |
+-------+---------+--------+------------+-------------+----------------------+
| b99.. | mytest1 | ACTIVE | -          | Running     | demo-net=172.16.0.8, |
|       |         |        |            |             |          10.100.1.110 |
+-------+---------+--------+------------+-------------+----------------------+
```

В выводе последней команды нас интересует ID экземпляра (в таблице для экономии места первый столбец сокращен). Создадим триггер, используя этот ID:

```
$ source keystonerc_admin
$ ceilometer alarm-threshold-create --name myalarm1 --meter-name
cpu_util --threshold 60.0 --comparison-operator gt --statistic avg
--period 60 --evaluation-periods 3 --alarm-action 'log://' --query
resource_id=b99e45af-95d2-436f-bfc6-64e5bfa999de
+---------------------------+--------------------------------------+
| Property                  | Value                                |
+---------------------------+--------------------------------------+
| alarm_actions             | [u'log://']                          |
| alarm_id                  | c33000db-a4eb-41ea-9742-96e5d7b4d034 |
| comparison_operator       | gt                                   |
| description               | Alarm when cpu_util is gt a avg of   |
|                           | 60.0 over 60 seconds                 |
| enabled                   | True                                 |
| evaluation_periods        | 3                                    |
| exclude_outliers          | False                                |
| insufficient_data_actions | []                                   |
| meter_name                | cpu_util                             |
| name                      | myalarm1                             |
| ok_actions                | []                                   |
| period                    | 60                                   |
```

```
| project_id                | |                                      |
| query                     | resource_id == b99e45af-95d2-436f-.. |
| repeat_actions            | False                                |
| state                     | insufficient data                    |
| statistic                 | avg                                  |
| threshold                 | 90.0                                 |
| type                      | threshold                            |
| user_id                   | 595c7da34b8e41bb812a0f3ecd6e7260     |
+---------------------------+--------------------------------------+
```

Мы создали триггер с именем myalarm1, который сработает, если средняя загрузка процессора виртуальной машины превысит 60% для трех измерений подряд через каждые 60 секунд. Как видно из вывода команды, сразу после создания триггер находится в состоянии «Недостаточно данных» (insufficient data). Подождем несколько минут и выполним команду:

```
$ ceilometer alarm-list
+----------+---------+-------+---------+------------+-----------------------------+-----------------+
| Alarm ID | Name    | State | Enabled | Continuous | Alarm condition             | Time constraints |
+----------+---------+-------+---------+------------+-----------------------------+-----------------+
| c33000d.. | myalarm1 | ok    | True    | False      | cpu_util > 60.0 during 3 x 60s | None            |
+----------+---------+-------+---------+------------+-----------------------------+-----------------+
```

Состояние триггера изменилось на «Ok». Это значит, что данные собираются, но заданное условие не наступило. Проверим загрузку процессора:

```
$ ceilometer sample-list --meter cpu_util -q 'resource_id=b99e45af-95d2-436f-bfc6-
64e5bfa999de'
+-------------+-------- --+----- -+---------------+------+---------------------+
| Resource ID | Name      | Type  | Volume        | Unit | Timestamp           |
+-------------+----------+-------+---------------+------+---------------------+
| b99e45af-.. | cpu_util | gauge | 10.868852459  | %    | 2015-06-02T19:05:09 |
| b99e45af-.. | cpu_util | gauge | 9.81632653061 | %    | 2015-06-02T19:04:08 |
| b99e45af-.. | cpu_util | gauge | 6.875         | %    | 2015-06-02T19:03:19 |
+-------------+----------+-------+---------------+------+---------------------+
```

или суммарную статистику:

```
$ ceilometer statistics -m cpu_util -q 'resource_id=b99e45af-95d2-436f-bfc6-64e5bfa999de'
+--------+------------+------------+------+------+------+-------+-------+----------+--------------+--------------+
| Period | Period Start| Period End | Max  | Min  | Avg  | Sum   | Count | Duration | Duration Start| Duration End |
+--------+------------+------------+------+------+------+-------+-------+----------+--------------+--------------+
| 0      | 2015-06-02T1| 2015-06-02T| 14.35 | 6.87 | 11.9 | 250.6 | 21    | 1189.0   | 2015-06-02T19 | 2015-06-02T1 |
+--------+------------+------------+------+------+------+-------+-------+----------+--------------+--------------+
```

Действительно, значение около десяти процентов. «Исправим» это, подключившись к виртуальной машине:

```
$ ssh -i demokey1 cirros@10.100.1.110
$ md5sum /dev/zero
```

Проверим, что команда `md5sum` действительно загрузила процессор более чем на шестьдесят процентов:

```
$ ceilometer sample-list --meter cpu_util -q 'resource_id=b99e45af-95d2-436f-bfc6-
64e5bfa999de'
+-------------+----------+-------+---------------+------+---------------------+
| Resource ID | Name     | Type  | Volume        | Unit | Timestamp           |
+-------------+----------+-------+---------------+------+---------------------+
| b99e45af-.. | cpu_util | gauge | 69.7666666667 | %    | 2015-06-02T20:20:37 |
| b99e45af-.. | cpu_util | gauge | 88.7627118644 | %    | 2015-06-02T20:19:37 |
| b99e45af-.. | cpu_util | gauge | 82.6          | %    | 2015-06-02T20:18:38 |
+-------------+----------+-------+---------------+------+---------------------+
```

и триггер сработал:

```
$ ceilometer alarm-list
+----------+---------+-------+---------+------------+----------------------------+-----------------+
| Alarm ID | Name    | State | Enabled | Continuous | Alarm condition            | Time constraints |
+----------+---------+-------+---------+------------+----------------------------+-----------------+
| c33000d.. | myalarm1 | alarm | True    | False      | cpu_util > 60.0 during 3 x 60s | None            |
+----------+---------+-------+---------+------------+----------------------------+-----------------+
```

При необходимости можно также обновить триггер, например задать границу срабатывания в 75 процентов:

```
$ ceilometer alarm-threshold-update --threshold 75.0 c33000db-
a4eb-41ea-9742-96e5d7b4d034
```

Наконец, можно просмотреть историю по триггеру:

```
$  ceilometer alarm-history -a c33000db-a4eb-41ea-9742-96e5d7b4d034
+------------------+----------------------------+----------------------+
| Type             | Timestamp                  | Detail               |
+------------------+----------------------------+----------------------+
| rule change      | 2015-06-.. | rule: cpu_util > 75.0 during 3 x 60s |
| state transition | 2015-06-.. | state: ok                            |
| state transition | 2015-06-.. | state: alarm                         |
| creation         | 2015-06-.. | name: myalarm1                       |
|                  |            | description: Alarm when cpu_util is  |
|                  |            |     gt a avg of 60.0 over 60 seconds |
|                  |            | type: threshold                      |
|                  |            | rule: cpu_util > 60.0 during 3 x 60s |
|                  |            | time_constraints: None               |
+------------------+----------------------------+----------------------+
```

Когда триггер нам больше не нужен, его можно удалить командой `ceilometer alarm-delete`.

Глава 13

Сервис оркестрации Heat

```
Название: OpenStack Orchestration
Назначение: управление жизненным циклом инфраструктуры и приложений
Пакет: openstack-heat-*, python-heatclient
Имена сервисов: openstack-heat-api, openstack-heat-api-cfn,
openstack-heat-engine
Порт: 8000, 8004
Конфигурационный файл: /etc/heat/heat.conf
Файлы журнала: /var/log/heat/heat-*
```

Последний из основных проектов OpenStack, с которым мы познакомимся в книге, – это сервис оркестрации OpenStack Orchestration или Heat. Службы Heat позволяют автоматизировать управление жизненным циклом наборов облачных сервисов (виртуальными машинами, сетями, томами, группами безопасности и т. д.) как единым целым, объединяя их в так называемые стеки (stack). Задачи могут быть как простыми, например развертывание виртуальной машины, так и более сложными, например старт комплексного приложения из многих машин и его масштабирование в зависимости от информации, передаваемой модулем телеметрии. Для описания стеков используются специальные, одновременно легко читаемые человеком и дружественные к обработке машиной форматы описания ресурсов, их ограничений, зависимостей и параметров:

- HOT (Heat Orchestration Template) – формат, предназначенный исключительно для OpenStack. Представляет собой документ формата YAML. Данный формат появился начиная с версии Icehouse и считается стандартным в Heat. Именно с ним мы и будем работать;
- CFT (AWS CloudFormation) – документ формата JSON в формате, совместимом с шаблонами сервиса CloudFormation (http://aws.amazon.com/ru/cloudformation/). Наличие возможности работать с этим типом форматов позволяет использовать множество уже существующих для AWS шаб-

лонов. В качестве стартовой точки можно рекомендовать https://aws.amazon.com/cloudformation/aws-cloudformation-templates/.

Архитектура сервиса

К числу основных компонентов службы оркестрации относятся:

- **openstack-heat-engine** – основной сервис, обеспечивающий обработку шаблонов и отправляющий события пользователям API;
- **openstack-heat-api** – сервис, отвечающий за предоставление основного REST API Heat. Сервис взаимодействует с openstack-heat-engine через вызовы RPC;
- **openstack-heat-api-cfn** – аналогичен предыдущему сервису, но обеспечивает работу с API, совместимым с AWS CloudFormation. Также взаимодействует с openstack-heat-engine;
- **клиент командной строки heat** – интерфейс взаимодействия с Heat API. Помимо командной строки, разработчики могут напрямую вызывать REST API, а пользователи облака могут запускать стеки через веб-интерфейс Horizon (**Project → Orchestration → Stacks**). Снимок экрана одной из вкладок веб-интерфейса представлен на рис. 13.1.

Установка сервисов Heat

Мы установим сервисы Heat на узел controller.test.local. Для начала установим необходимые пакеты:

```
[root@controller ~]# yum -y install openstack-heat-api openstack-heat-
api-cfn openstack-heat-engine python2-heatclient openstack-heat-ui
```

Затем нам необходимо создать базу данных MariaDB с необходимыми привилегиями. По соглашению, принятому в книге, в качестве пароля используем имя сервиса «heat»:

```
[root@controller ~]# mysql -u root -p
MariaDB [(none)]> CREATE DATABASE heat;
MariaDB [(none)]> GRANT ALL PRIVILEGES ON heat.* TO
'heat'@'localhost' IDENTIFIED BY 'heat';
MariaDB [(none)]> GRANT ALL PRIVILEGES ON heat.* TO 'heat'@'%'
IDENTIFIED BY 'heat';
MariaDB [(none)]> exit
```

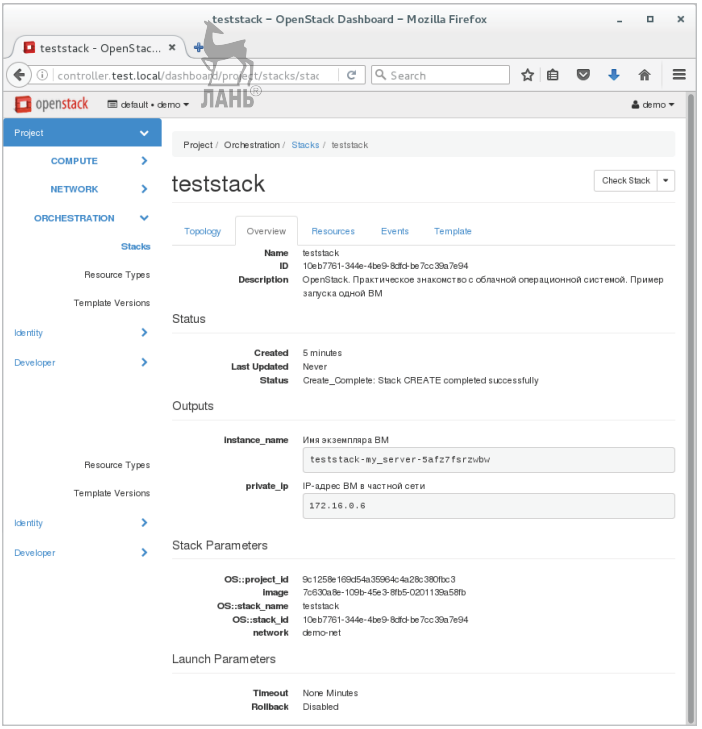


Рис. 13.1. Внешний вид интерфейса Horizon для работы со стеками

Затем создаем пользователя heat в сервисе Keystone и добавляем ему роль admin в проекте service:

```
$ source keystonerc_admin
$ openstack user create --domain default --password openstack heat
$ openstack role add --project service --user heat admin
```

Для сервиса оркестрации также понадобится домен, который будет содержать проекты и пользователей стека:

```
$ openstack domain create --description "Stack projects and users" heat
```

В новом домене `heat` нам нужен пользователь `heat_domain_admin`, который будет управлять проектами и пользователями:

```
$ openstack user create --domain heat --password openstack heat_
domain_admin
$ openstack role add --domain heat --user-domain heat --user
heat_domain_admin admin
```

Теперь создадим роль владельца стека и добавим эту роль в проект и пользователю demo, тем самым дав ему возможность управлять стеками:

```
$ openstack role create heat_stack_owner
$ openstack role add --project demo --user demo heat_stack_owner
```

Также нам понадобится роль пользователя стека, который будет получать данные о прогрессе выполнения операций. Сервис оркестрации будет автоматически добавлять роль `heat_stack_user` пользователям, запускающим стек во время его запуска.

```
$ openstack role create heat_stack_user
```

Как это делали не раз, создадим сервисы. В данном случае два: для основного сервиса heat-api и для совместимого с CloudFormation heat-api-cfn:

```
$ openstack service create --name heat --description "Orchestration"
orchestration
$ openstack service create --name heat-cfn --description "Orchestration"
cloudformation
```

Создаем точки входа в сервисы Heat:

```
$ openstack endpoint create --region RegionOne orchestration
public http://controller.test.local:8004/v1/%\(tenant_id\)s
$ openstack endpoint create --region RegionOne orchestration
internal http://controller.test.local:8004/v1/%\(tenant_id\)s
$ openstack endpoint create --region RegionOne orchestration
admin http://controller.test.local:8004/v1/%\(tenant_id\)s
$ openstack endpoint create --region RegionOne cloudformation
public http://controller.test.local:8000/v1
$ openstack endpoint create --region RegionOne cloudformation
internal http://controller.test.local:8000/v1
$ openstack endpoint create --region RegionOne cloudformation
admin http://controller.test.local:8000/v1
```

Затем приступаем к редактированию конфигурационного файла /etc/heat/heat.conf. Первое, что необходимо указать, – это параметры подключения к базе данных MariaDB:

```
[root@controller ~]# crudini --set /etc/heat/heat.conf database
connection mysql+pymysql://heat:heat@controller.test.local/heat
```

Дальше добавляем в конфигурационные файлы информацию о сервисе RabbitMQ:

```
[root@controller ~]# crudini --set /etc/heat/heat.conf DEFAULT
transport_url rabbit://openstack:openstack@controller.test.local
```

Производим настройки сервиса идентификации Keystone. Эти параметры мы уже задавали в конфигурационных файлах других, уже рассмотренных сервисов. Также добавляются секции `[trustee]` и `[clients_keystone]`:

```
[root@controller ~]# crudini --set /etc/heat/heat.conf keystone_
authtoken auth_uri http://controller.test.local:5000
[root@controller ~]# crudini --set /etc/heat/heat.conf keystone_
authtoken auth_url http://controller.test.local:35357
[root@controller ~]# crudini --set /etc/heat/heat.conf keystone_
authtoken auth_type password
[root@controller ~]# crudini --set /etc/heat/heat.conf keystone_
authtoken project_domain_name default
[root@controller ~]# crudini --set /etc/heat/heat.conf keystone_
authtoken user_domain_name default
[root@controller ~]# crudini --set /etc/heat/heat.conf keystone_
authtoken project_name service
[root@controller ~]# crudini --set /etc/heat/heat.conf keystone_
authtoken username heat
[root@controller ~]# crudini --set /etc/heat/heat.conf keystone_
authtoken password openstack
[root@controller ~]# crudini --set /etc/heat/heat.conf trustee
auth_type password
[root@controller ~]# crudini --set /etc/heat/heat.conf trustee
auth_url http://controller.test.local:35357
[root@controller ~]# crudini --set /etc/heat/heat.conf trustee
username heat
[root@controller ~]# crudini --set /etc/heat/heat.conf trustee
password openstack
[root@controller ~]# crudini --set /etc/heat/heat.conf trustee
user_domain_name default
[root@controller ~]# crudini --set /etc/heat/heat.conf clients_
keystone auth_uri http://controller.test.local:5000
```

Задаем URL сервера метаданных и сервера, сообщающего о доступности ресурсов при создании шаблонов, где один ресурс создается после того, как стал доступен ресурс, от которого он зависит:

```
[root@controller ~]# crudini --set /etc/heat/heat.conf DEFAULT
heat_metadata_server_url http://controller.test.local:8000
[root@controller ~]# crudini --set /etc/heat/heat.conf DEFAULT
heat_waitcondition_server_url = http://controller.test.
local:8000/v1/waitcondition
```

Для ряда операций, используемых в шаблонах, требуется создание отдельного домена Keystone для сервиса оркестрации. При-

менение отдельного домена позволяет разделить виртуальные машины и пользователей, запускающих стеки. Это дает возможность простым пользователям без административных прав работать со стеками. Как была выбрана данная модель и каковы были соображения разработчиков, можно почитать в блоге одного из инженеров, реализовавшего данный функционал, – hardysteven.blogspot.com. Указываем выделенный под запуск стеков домен и учетные данные:

```
[root@controller ~]# crudini --set /etc/heat/heat.conf DEFAULT
stack_domain_admin heat_domain_admin
[root@controller ~]# crudini --set /etc/heat/heat.conf DEFAULT
stack_domain_admin_password openstack
[root@controller ~]# crudini --set /etc/heat/heat.conf DEFAULT
stack_user_domain_name heat
```

Последним шагом в настройке заполняем базу данных:

```
[root@controller ~]# su -s /bin/sh -c "heat-manage db_sync" heat
```

Включаем и стартуем сервисы:

```
[root@controller ~]# systemctl enable openstack-heat-api.service
openstack-heat-api-cfn.service openstack-heat-engine.service
[root@controller ~]# systemctl start openstack-heat-api.service
openstack-heat-api-cfn.service openstack-heat-engine.service
```

Проверим, что сервисы стартовали и доступны:

```
$ openstack orchestration service list
+----------------------+-------------+-----------+----------------------+--------+----------------+--------+
| hostname             | binary      | engine_id | host                 | topic  | updated_at     | status |
+----------------------+-------------+-----------+----------------------+--------+----------------+--------+
| controller.test.local | heat-engine | edcc580.. | controller.test.local | engine | 2018-03-18T17.. | up     |
| controller.test.local | heat-engine | e67b25c.. | controller.test.local | engine | 2018-03-18T17.. | up     |
| controller.test.local | heat-engine | 6737c1d.. | controller.test.local | engine | 2018-03-18T17.. | up     |
| controller.test.local | heat-engine | dc9e7fe.. | controller.test.local | engine | 2018-03-18T17.. | up     |
+----------------------+-------------+-----------+----------------------+--------+----------------+--------+
```

Теперь можем начать разбираться с шаблонами и работой сервиса оркестрации.

Запуск простого стека

Начнем с того, что со страницы блога автора книги http://markelov.blogspot.ru/p/openstack.html скачаем архив с примерами конфигурационных файлов и шаблонов. Нам понадобится файл

/config_files/HOT/test-server.yml. Также его текст приведен в приложении 2.

Данный шаблон является одной из многих вариаций «Hello, World!» для Heat. Создается стек, состоящий из одной виртуальной машины, которой во время старта передается скрипт, выводящий сообщение «Instance STARTED!» на стандартный вывод. Прежде чем запустить стек, рассмотрим листинг.

Строки с первой по четвертую – это заголовок шаблона и описание. Дата в версии heat_template_version выбирается не произвольно, а задается одним из следующих вариантов, соответствующих релизу OpenStack:

- 2013-05-23 – Icehouse;
- 2014-10-16 – Juno;
- 2015-03-30 – Kilo;
- 2015-10-15 – Liberty;
- 2016-04-08 – Mitaka;
- 2016-10-14 – Newton;
- 2017-02-24 – Ocata;
- 2017-09-01 – Pike;
- 2018-03-02 – Queens.

Описание опционально, и если оно не помещается в одну строку, то, как в примере, разбивается на несколько строк в соответствии со спецификацией YAML.

Далее следуют три секции, первая из которых – parameters – начинается с шестой строки. В данной части шаблона определяются параметры network и image, которые можно задать во время запуска стека. Оба параметра имеют тип – строка. У каждого из параметров есть значения по умолчанию, используемые, если во время запуска стека их не задали, – это строки десятая и четырнадцатая. В шаблоне также могла быть секция parameter_groups, в которой описывалось бы, как параметры должны быть сгруппированы, и их порядок. В данном шаблоне секция parameter_groups отсутствует.

Следующая секция – resources, которая описывает ресурсы шаблона. В этой секции должен быть описан как минимум один ресурс. В данном случае как раз и описан один ресурс с именем my_server типа OS::Nova::Server. В подсекции properties определены параметры сервера, которые мы обычно задаем командой nova boot. Три из них, а именно размер виртуальной машины (flavor), имя открытого SSH-ключа и скрипт, который будет исполнен при

старте при помощи cloud-init, жестко заданы в теле шаблона. Еще два параметра, которые ранее были описаны в секции parameters, подставляются при помощи функции *get_param*.

Наконец, третья секция – outputs – задает параметры, которые будут выводиться пользователю, когда шаблон отработает в выводе heat stack-show, в интерфейсе Horizon или при запросе через API. Секция outputs является необязательной.

Теперь, когда мы разобрали шаблон, попробуем его запустить командой `openstack stack create`. Нам нужно указать путь к файлу шаблона и, по желанию, после опции `--parameter` параметры, которые были определены в секции parameters. По желанию, потому что в шаблоне для них заданы значения по умолчанию.

```
$ source keystonerc_demo
$ openstack stack create --parameter network=demo-net --parameter
image=c8ccc9b3-29bb-4220-be38-8f261ac8b99a -t test-server.yml
teststack
+---------------------+--------------------------------------+
| Field               | Value                                |
+---------------------+--------------------------------------+
| id                  | 15cd8428-3787-4fad-95ea-aec1ec2eec81 |
| stack_name          | teststack                            |
| description         | OpenStack. Практическое знакомство   |
|                     |  с облачной операционной системой.   |
|                     |  Пример запуска одной ВМ             |
|                     |                                      |
| creation_time       | 2018-03-20T16:44:35Z                 |
| updated_time        | None                                 |
| stack_status        | CREATE_IN_PROGRESS                   |
| stack_status_reason | Stack CREATE started                 |
+---------------------+--------------------------------------+
```

При помощи следующей команды мы можем следить за процессом отработки стека, дождавшись, пока статус не поменяется на CREATE_COMPLETE:

```
$ openstack stack list
+------+------------+-----------------+----------------------+--------------+
| ID   | Stack Name | Stack Status    | Creation Time        | Updated Time |
+------+------------+-----------------+----------------------+--------------+
| 10.. | tteststack | CREATE_COMPLETE | 2018-03-20T16:44:35Z | None         |
+------+------------+-----------------+----------------------+--------------+
```

Проверим, что у нас действительно была создана виртуальная машина. При этом обратите внимание, что ее имя было сформировано из имени стека и имени ресурса:

```
$ openstack server list
+--------------------------------------+-------------------------------+--------+-------------------+------------------+--------+
| ID                                   | Name                          | Status | Networks          | Image            | Flavor |
+--------------------------------------+-------------------------------+--------+-------------------+------------------+--------+
| e1a98861-27d9-4640-aafb-24e159f5d7d2 | teststack-my_server-g6m7qyi7jsn4 | ACTIVE | demo-net=172.16.0.12 | cirros-0.4.0-x86_64 | m2.tiny |
+--------------------------------------+-------------------------------+--------+-------------------+------------------+--------+
```

Команда `openstack stack show` вместе с именем стека покажет его детали, включая параметры, которые попросили вывести в секции outputs. Будут показаны имя экземпляра виртуальной машины и IP-адрес виртуальной машины в сети demo-net:

```
$ openstack stack show teststack
+-----------------------+-----------------------------------------------------+
| Field                 | Value                                               |
+-----------------------+-----------------------------------------------------+
| id                    | 15cd8428-3787-4fad-95ea-aec1ec2eec81                |
| stack_name            | teststack                                           |
| description           | OpenStack. Практическое знакомство с облачной       |
|                       | операционной системой. Пример запуска одной ВМ      |
| creation_time         | 2018-03-20T16:44:35Z                                |
| updated_time          | None                                                |
| stack_status          | CREATE_COMPLETE                                     |
| stack_status_reason   | Stack CREATE completed successfully                 |
| parameters            | OS::project_id: bc10ac4b71164550a363b8098e8ad270    |
|                       | OS::stack_id: 15cd8428-3787-4fad-95ea-aec1ec2..     |
|                       | OS::stack_name: teststack                           |
|                       | image: c8ccc9b3-29bb-4220-be38-8f261ac8b99a         |
|                       | network: demo-net                                   |
|                       |                                                     |
| outputs               | - description: "Имя экземпляра ВМ"                  |
|                       |   output_key: instance_name                         |
|                       |   output_value: teststack-my_server-g6m7qyi7jsn4     |
|                       | - description: "IP-адрес ВМ в частной сети"         |
|                       |   output_key: private_ip                            |
|                       |   output_value: 172.16.0.12                         |
|                       |                                                     |
| links                 | - href: http://controller.test.local:8004/v1/9c..   |
| parent                | None                                                |
| disable_rollback      | True                                                |
| deletion_time         | None                                                |
| stack_user_project_id | 10fd4f2d320b4c94b79271219e560384                    |
| capabilities          | []                                                  |
| notification_topics   | []                                                  |
| stack_owner           | None                                                |
| timeout_mins          | None                                                |
| tags                  | null                                                |
+-----------------------+-----------------------------------------------------+
```

Для того чтобы убедиться, что скрипт, выводящий сообщение через cloud-init, сработал, можно либо подключиться к консоли виртуальной машины в Horizon, либо ввести команду nova, которая показывает вывод консоли:

```
$ openstack console log show teststack-my_server-g6m7qyi7jsn4 |
grep STARTED
Instance STARTED!
```

Можно проверить список событий при создании стека:

```
$ openstack stack event list teststack
2018-03-20 16:44:36Z [teststack]: CREATE_IN_PROGRESS  Stack CREATE started
2018-03-20 16:44:37Z [teststack.my_server]: CREATE_IN_PROGRESS  state changed
2018-03-20 16:44:59Z [teststack.my_server]: CREATE_COMPLETE  state changed
2018-03-20 16:44:59Z [teststack]: CREATE_COMPLETE  Stack CREATE completed
successfully
```

Список ресурсов включает в себя только один ресурс – сервер my_server:

```
$ openstack stack resource list teststack
+---------------+----------------------+-----------------+-----------------+--------------+
| resource_name | physical_resource_id | resource_type   | resource_status | updated_time |
+---------------+----------------------+-----------------+-----------------+--------------+
| my_server     | e1a98861-27d9-464..  | OS::Nova::Server | CREATE_COMPLETE | 2018-03-20.. |
+---------------+----------------------+-----------------+-----------------+--------------+
```

Команда openstack stack template show teststack покажет шаблон, из которого был создан стек. В конце работы удаляем стек:

```
$ openstack stack delete teststack
Are you sure you want to delete this stack(s) [y/N]? y
```

На рис. 13.2 представлена топология более сложного стека, как она отображается в веб-интерфейсе. Эта вкладка позволяет наглядно представить связи между ресурсами.

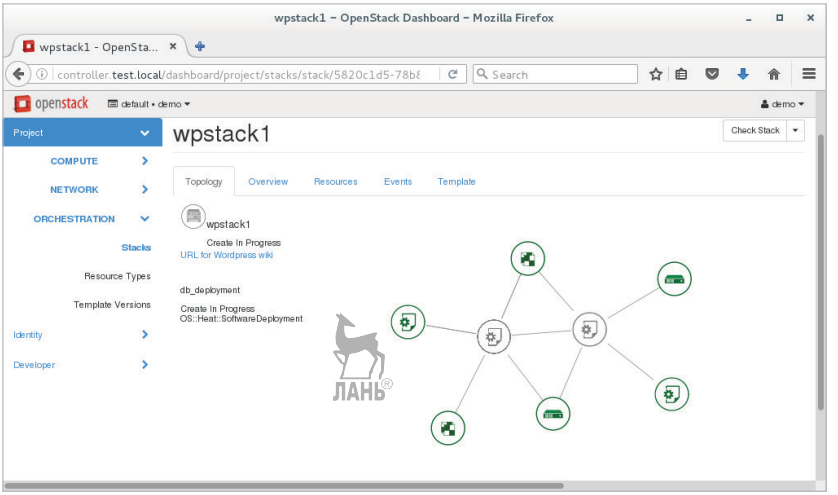


Рис. 13.2. Топология шаблона Heat

Глава 14

Контейнеры и OpenStack

Контейнеры Docker, как и OpenStack, в последнее время являются «горячей темой» в мире DevOps и виртуализации. Неудивительно, что OpenStack также может интегрироваться с контейнерами. Для начала кратко рассмотрим, что же представляет из себя Docker.

Краткое знакомство с Docker

В отличие от «вертикального» абстрагирования в случае виртуализации, контейнеры, в частности Docker, обеспечивают горизонтальное разбиение операционной системы на отдельные изолированные окружения. За счет того, что в каждом контейнере, в отличие от виртуализации, обходятся без использования отдельного экземпляра операционной системы, значительно ниже накладные расходы. С другой стороны, вы не сможете на одном узле в контейнерах запускать разные операционные системы, например Windows и GNU/Linux.

Контейнеры используются для замены виртуализации там, где они справляются лучше:

- горизонтальная изоляция приложений;
- делегирование окружений;
- «виртуализация приложений»;
- максимальная плотность.

Также необходимо отметить, что зачастую контейнеры используются поверх виртуальных машин.

Проще всего дать определение контейнеру Docker как запущенному из образа приложению. Кстати, именно этим идеологически и отличается Docker, например, от LXC (Linux Containers), хотя они используют одни и те же технологии ядра Linux. Docker исповедует принцип: один контейнер – это одно приложение. Образ Docker –

статический снимок конфигурации контейнера. Образы могут зависеть от других образов. Образ всегда находится в режиме «только чтение», а изменения сохраняются созданием образа поверх образа.

Для обеспечения изоляции приложений в контейнерах используются стандартные Linux-технологии: пространства имен (с ними мы уже сталкивались в главе, посвященной Neutron), Cgroups – средство распределения ресурсов и технологии мандатного контроля доступа SELinux. Для разделения различных уровней контейнера на уровне файловой системы могут использоваться AUFS, btrfs, vfs и Device Mapper.

Готовые образы в формате Docker можно скачивать как из публичных репозиториев, так и создавать свои, приватные.

Более подробно с использованием Docker можно ознакомиться при помощи документации на сайте https://www.docker.com/.

Важно отметить, что в июле 2015 года Linux Foundation анонсировала запуск нового проекта Open Container Project (OCP), который призван установить общие стандарты и обеспечить фундамент совместимости для продолжения развития контейнерных решений без дальнейшей фрагментации этого направления. Инициативу OCP поддержали многие крупные компании и организации, среди которых можно упомянуть Amazon Web Services, Apcera (компания принадлежит Ericsson), Cisco, CoreOS, EMC, Google, HP, IBM, Intel, Microsoft, Red Hat, Vmware и др. В качестве основы Open Container Project будут выступать в значительной степени наработки Docker.

Второе событие июля, которое хотелось бы отметить, – это вхождение Google в состав OpenStack Foundation в качестве корпоративного спонсора. Google планирует сконцентрировать свои усилия на интеграции системы управления контейнерами Kubernetes с OpenStack.

Совместное использование Docker и OpenStack

Существует несколько проектов, направленных на организацию совместной работы Docker и OpenStack, и все они находятся в активной разработке.

Исторически первым появился драйвер Docker для OpenStack Nova – https://github.com/stackforge/nova-docker. Для облака контейнеры в таком случае выглядят как другие экземпляры вирту-

альных машин, а образы Docker загружаются из сервиса Glance. Это наиболее простой способ совместного использования контейнеров и OpenStack, и далее в главе мы рассмотрим именно его.

Следующий проект – Kolla (https://github.com/stackforge/kolla). Его задачей является облегчение работы по обслуживанию OpenStack. Kolla представляет собой сервис для контейнеризации служб OpenStack как отдельных микросервисов. Каждый микросервис представляет собой атомарный объект для развертывания, обновлений и т. д. Kolla выступает как связывающее звено, позволяя осуществлять упрощенное управление облаком при помощи таких инструментов, как TripleO, Heat, Ansible и т. д. В качестве источника готовых контейнеров с микросервисами OpenStack можно использовать готовый репозиторий Kollaglue. На момент написания этого текста в репозитории было около тридцати пяти сервисов. Как многие другие идеи и проекты в мире OpenSource, этот проект используется и в других разработках. Так, например, разработчики oVirt, открытой системы управления виртуализацией, планируют в версии 3.6 при помощи Kolla осуществлять интеграцию oVirt Manager со службами OpenStack.

Наконец, еще один проект – Magnum, который можно было бы назвать CaaS – «контейнеры как сервис». Страница проекта – https://github.com/openstack/magnum. В отличие от использования Docker при помощи драйвера Nova compute, где контейнеры заменяют собой экземпляры виртуальных машин, при помощи Magnum контейнеры работают поверх виртуальных машин, создаваемых Heat.

Magnum управляет контейнерами не напрямую, а через один из движков управления контейнерами (container orchestration engines). Поддерживаются три варианта: Docker Swarm, Kubernetes и Apache Mesos. Из них наиболее популярный – Kubernetes – система оркестрации контейнеров, разработка которой была начата компанией Google. Именно Kubernetes используется в Magnum по умолчанию.

В качестве операционной системы предполагается использование одной из ориентированных на контейнеры сборок: Atomic, CoreOS, Snappy и им подобных. Можно сказать, что кластер контейнеров Magnum запускается в кластере виртуальных машин Nova.

Еще один способ совместного использования OpenStack и Docker – применение каталога Murano, для которого существует пакет, опубликованный в каталоге приложений сообщества:

http://apps.openstack.org/#tab=murano-apps&asset=Kubernetes%20Cluster. Пакет Murano может развернуть и настроить кластер Kubernetes, руководствуясь параметрами, которые пользователь передает через API или графический интерфейс Murano. Предварительным требованием является развернутый каталог приложений Murano, который не рассматривается в книге.

Настройка работы драйвера Docker для OpenStack Nova

Первое, с чего мы начнем, – это с установки Docker и необходимых пакетов. Обращаю внимание, что этот вариант установки подходит только для тестовых сред. Для «боевой установки» следует собрать rpm-пакет с драйвером Docker и не устанавливать на вычислительные узлы средства, применяемые при разработке. Действия выполняем на вычислительном узле:

```
[root@compute ~]# yum -y install net-tools docker-io python-pip git
```

Также обратите внимание на то, что мы устанавливаем пакет net-tools, который в современных дистрибутивах при установке по умолчанию заменен на iproute, а утилиты из его состава имеют статус устаревших. Нам он понадобится ради команды `ifconfig`, которая используется драйвером Docker.

Теперь забираем с Github исходный код драйвера и устанавливаем:

```
[root@compute ~]# git clone https://github.com/stackforge/nova-
docker.git
[root@compute ~]# cd nova-docker
[root@compute nova-docker]# python setup.py install
[root@compute nova-docker]# pip install docker-py
```

Следующим шагом запускаем и включаем сервис Docker и выполняем не очень «чистый» обходной прием с правами, для того чтобы в CentOS драйвер получил доступ к Docker:

```
[root@compute ~]# systemctl start docker
[root@compute ~]# systemctl enable docker
[root@compute ~]# chmod 666  /var/run/docker.sock
```

Нам необходимо в соответствии с инструкцией на Github настроить драйвер. Создаем файл с инструкциями по настройке сети для Docker:

```
[root@compute ~]# mkdir /etc/nova/rootwrap.d
[root@compute ~]# vi /etc/nova/rootwrap.d/docker.filters
```

и в файл docker.filters копируем следующее содержимое:

```
# nova-rootwrap command filters for setting up network in the docker
driver
# This file should be owned by (and only-writeable by) the root user
[Filters]
# nova/virt/docker/driver.py: 'ln', '-sf', '/var/run/netns/.*'
ln: CommandFilter, /bin/ln, root
```

Наконец, на узле Glance (это в нашем стенде controller) добавляем к форматам контейнеров docker:

```
[root@controller ~]# crudini --set /etc/glance/glance-api.conf
DEFAULT container_formats ami,ari,aki,bare,ovf,ova,docker
[root@controller ~]# systemctl restart openstack-glance-api
```

А на вычислительном узле compute в качестве драйвера указываем драйвер Docker. Других изменений производить не надо:

```
[root@compute ]# crudini --set /etc/nova/nova.conf DEFAULT
compute_driver novadocker.virt.docker.DockerDriver
[root@compute ]# systemctl restart openstack-nova-compute
```

Теперь тестируем нашу конфигурацию. Для начала ограничимся работоспособностью контейнеров Docker. Попробуем запустить контейнер с дистрибутивом Fedora:

```
[root@compute ~]# docker run -i -t fedora /bin/bash
Unable to find image 'fedora:latest' locally
Trying to pull repository docker.io/fedora ...
...
Status: Downloaded newer image for docker.io/fedora:latest
[root@4f3f76431725 /]# cat /etc/redhat-release
Fedora release 23 (Twenty Three)
```

Как мы видим, из репозитория docker.io был скачан последний образ Fedora и запущен. Если посмотреть на список контейнеров, мы также это увидим:

```
[root@compute ~]# docker ps
CONTAINER ID  IMAGE  COMMAND      CREATED        STATUS    PORTS   NAMES
19e4bb1d36ab  fedora "/bin/bash"  3 minutes ago  Up 3 minutes  prickly_ardinghelli
```

Теперь скачаем образ с минимальным http-сервером thttpd, который будем использовать для проверки работы:

```
[root@compute ~]# docker pull larsks/thttpd
Trying to pull repository docker.io/larsks/thttpd ...
...
Status: Downloaded newer image for docker.io/larsks/thttpd:latest
```

После этого загрузим его в Glance:

```
[root@compute ~]# source keystonerc_admin
[root@compute ~]# docker save larsks/thttpd | glance image-create
--visibility public --container-format docker --disk-format raw
--name larsks/thttpd
+------------------+--------------------------------------+
| Property         | Value                                |
+------------------+--------------------------------------+
| checksum         | 01453bf647d22c7824e3b5eb1534812c     |
| container_format | docker                               |
| created_at       | 2015-12-22T11:42:47Z                 |
| direct_url       | rbd://1d618cdf-c648-4e9c-8aed-c170.. |
|                  | b6b3520ba0fc/snap                    |
| disk_format      | raw                                  |
| id               | b3194d6e-2886-430f-bffc-b6b3520ba0fc |
| min_disk         | 0                                    |
| min_ram          | 0                                    |
| name             | larsks/thttpd                        |
| owner            | 8cc74ebe8da94fe0a7ac6cf54f31b420     |
| protected        | False                                |
| size             | 1081856                              |
| status           | active                               |
| tags             | []                                   |
| updated_at       | 2015-12-22T11:42:50Z                 |
| virtual_size     | None                                 |
| visibility       | public                               |
+------------------+--------------------------------------+
```

Проверяем список образов:

```
$  openstack image list
+--------------------------------------+
| ID   | Name                          |
+------+-------------------------------+
| b3.. | larsks/thttpd                 |
| 15.. | cirros-raw                    |
| 1a.. | apcera-trusty-orchestrator_.. |
| 08.. | apcera-trusty-deploy_143717.. |
| f3.. | ManageIQ-devel                |
| a9.. | fedora-20.x86_64              |
| 42.. | cirros-0.3.4-x86_64           |
+------+-------------------------------+
```

И в графическом интерфейсе Horizon:

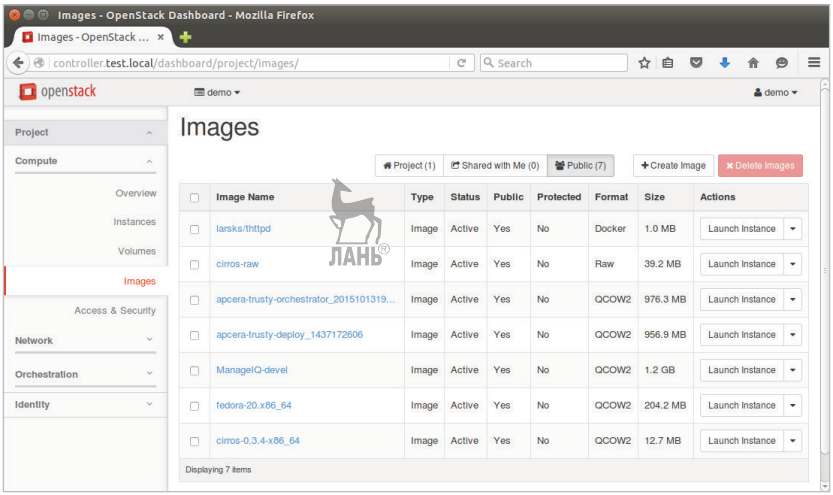


Рис. 14.1. Список образов

Наконец, можно попробовать создать экземпляр контейнера:

```
$ source keystonerc_demo
$ nova boot --image larsks/thttpd --flavor m1.small test1
+--------------------------------------+--------------------------------------+
| Property                             | Value                                |
+--------------------------------------+--------------------------------------+
| OS-DCF:diskConfig                    | MANUAL                               |
| OS-EXT-AZ:availability_zone          |                                      |
| OS-EXT-STS:power_state               | 0                                    |
| OS-EXT-STS:task_state                | scheduling                           |
| OS-EXT-STS:vm_state                  | building                             |
| OS-SRV-USG:launched_at               | -                                    |
| OS-SRV-USG:terminated_at             | -                                    |
| accessIPv4                           |                                      |
| accessIPv6                           |                                      |
| adminPass                            | zgcbrCvq7v9v                         |
| config_drive                         |                                      |
| created                              | 2015-12-22T13:47:25Z                 |
| flavor                               | m1.small (2)                         |
| hostId                               |                                      |
| id                                   | ee032b15-c2d1-4523-a3bd-810cd9a75d01 |
| image                                | larsks/thttpd (b3194d6e-2886-430f..  |
| key_name                             | -                                    |
| metadata                             | {}                                   |
| name                                 | test1                                |
| os-extended-volumes:volumes_attached | []                                   |
| progress                             | 0                                    |
| security_groups                      | default                              |
```

```
| status                               | BUILD                                |
| tenant_id                            | eca00feab38e4aa5b462bd31af0b9dca     |
| updated                              | 2015-12-22T13:47:25Z                 |
| user_id                              | 924c18c923654d7c930bcb1044580d8b     |
+--------------------------------------+--------------------------------------+
```

Проверяем, что контейнер запущен:

```
[root@compute ~]# docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED
STATUS              PORTS               NAMES
7bfa3c489c16        larsks/thttpd       "/thttpd -D -l /dev/s"   3 minutes ago
Up 3 minutes                            nova-ee032b15-c2d1-4523-a3bd-810cd9a75d01
nova-65de57d0-f033-4818-a522-2c3291dc516b
```

В графическом интерфейсе это выглядит следующим образом:

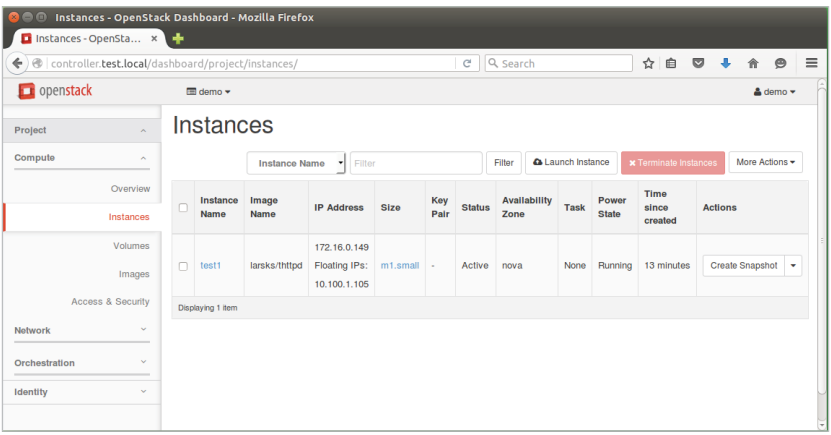


Рис. 14.2. Контейнер Docker в OpenStack

Для доступа к http-серверу присвоим экземпляру внешний floating IP:

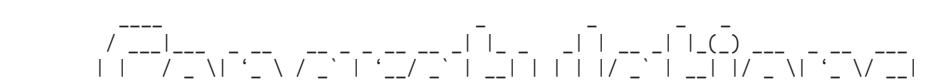
```
$ nova floating-ip-associate test1 10.100.1.105
```

Проверяем:

```
$ nova list
+------+-------+--------+------------+-------------+-----------------------+
| ID   | Name  | Status | Task State | Power State | Networks              |
+------+-------+--------+------------+-------------+-----------------------+
| ee.. | test1 | ACTIVE | -          | Running     | demo-net=172.16.0.149, |
|      |       |        |            |             |          10.100.1.105  |
+------+-------+--------+------------+-------------+-----------------------+
```

И из тестовой машины со внешней сети пробуем подключиться к серверу:

```
$ curl http://10.100.1.105
```



Рассмотренному варианту интеграции недостает ряда функций, которые реализованы для экземпляров виртуальных машин. Например, не поддерживается подключение блочных устройств Cinder. Также при обновлении образа в Glance приходится вручную удалять образ в локальном репозитории Docker на гипервизоре.

Глава 15

Программно-определяемая система хранения данных Ceph

```
Название: Ceph
Назначение: Программно-определяемая система хранения данных
Пакеты: ceph, ceph-radosgw
Имя сервиса: ceph.service
Конфигурационный файл: /etc/ceph/ceph.conf
Файлы журнала: /var/log/ceph/*
Сайт проекта: ceph.com
```

Согласно результатам опроса OpenStack User Survey от апреля 2017 года, проводившегося перед очередным саммитом OpenStack, значительная часть внедрений OpenStack в качестве блочной системы хранения данных использует Ceph. На тот момент эта доля составляла для всех типов внедрений 65%. Ceph – отдельный проект с открытым исходным кодом, не входящий в число проектов OpenStack. Однако, рассматривая службы OpenStack, нельзя не упомянуть проект, чей код входит практически во все основные дистрибутивы OpenStack.

Что же представляет из себя Ceph? Ceph – это проект с открытым исходным кодом по построению унифицированного программно-определяемого хранилища данных. Отличительными особенностями Ceph являются высокая масштабируемость, производительность и отсутствие единой точки отказа. В качестве аппаратного обеспечения предполагается использование серверов стандартной x86 архитектуры. Под унификацией понимается возможность хранилища предоставлять объектный, блочный и файловый доступы.

Основные принципы, которыми руководствовались разработчики при создании архитектуры Ceph:

- отказ компонента системы – это обычное поведение системы, а не исключительная ситуация. Соответственно, не должно быть единой точки отказа;
- каждый компонент системы должен масштабироваться горизонтально;
- по возможности, все компоненты для своего обслуживания и управления должны требовать минимального вмешательства администратора;
- решение должно работать на серверах стандартной архитектуры (сейчас есть коммерческие решения и на ARM-архитектуре);
- решение должно быть с открытым исходным кодом.

Изначально Ceph был разработан в 2003 году в Калифорнийском университете (University of California, Santa Cruz). В 2006-м он был выпущен как продукт с открытым исходным кодом. В промышленной эксплуатации проект использовался с 2007 года хостинговой компанией DreamHost. А в 2012 году первоначальный разработчик проекта Sage Wail при поддержке DreamHost основал компанию Inktank, которая начала оказывать коммерческую поддержку для Ceph. Кстати, Ceph – это сокращение от «cephalopod». Отсюда становится понятным выбор названия для компании – Inktank. В середине 2014 года Inktank купила компания Red Hat. Для Red Hat на тот момент это было второе поглощение компании, создающей программно-определяемую систему хранения данных. Предыдущей покупкой была GlusterFS. На настоящий момент поддержку Ceph в составе OpenStack, помимо Red Hat, оказывает большинство компаний, имеющих свои дистрибутивы облачного программного обеспечения: SUSE, Mirantis, Canonical и др.

OpenStack и Ceph совместно используются достаточно давно, начиная с релиза Grizzly, и эта связка является проработанной и надежной. Со стороны OpenStack программно-определяемую систему хранения данных могут использовать компоненты Glance, Cinder и Nova. Также Ceph может использоваться вместо объектного хранилища Swift.

Фактически вышесказанное означает, что Ceph может заменить все другие типы хранилищ при использовании совместно с OpenStack. В целом это так, однако конкретная архитектура

определяется в зависимости от требований, предъявляемых к облаку. Например, некоторые аргументы в сторону Swift или Ceph при выборе объектного хранилища приведены в блоге компании Mirantis (https://www.mirantis.com/blog/ceph-vs-swift-architects-perspective/).

Проект Ceph именует каждую версию по аналогии с OpenStack именем, начинающимся с буквы латинского алфавита: A, B, C, D... Начиная с версии Infernalis каждый релиз имеет основную версию и минорную, после точки. Минорная версия 0 соответствует релизу, находящемуся в разработке, 1 – релиз-кандидат и 2 – стабильная версия. Например, стабильный релиз Jewel (версия 10):

```
# ceph -v
ceph version 10.2.3-17.el7cp (ca9d57c0b140eb5cea9de7f7133260271e57490e)
```

Архитектура Ceph

В «крупную клетку» основные компоненты Ceph приведены на рис. 15.1.

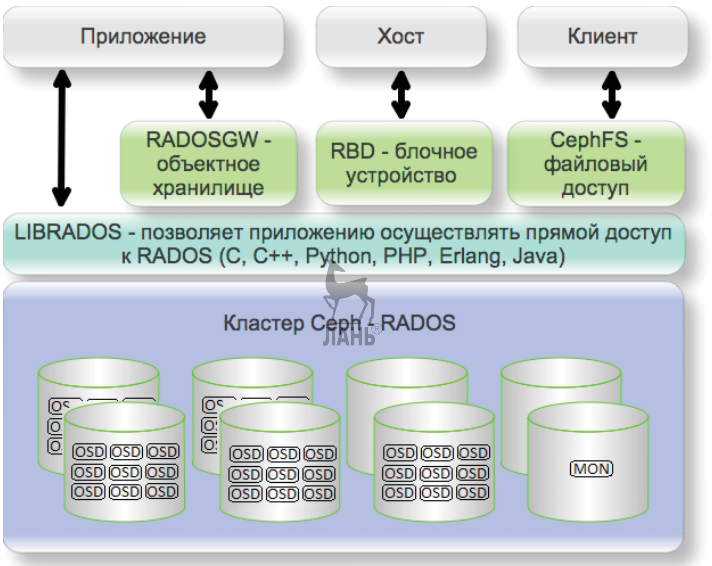


Рис. 15.1. Архитектура Ceph

Давайте рассмотрим эти компоненты. Основой Ceph является **кластер RADOS** (Reliable Autonomic Distributed Object Store). Да,

в основе Ceph лежит именно объектное хранилище. Данные при этом хранятся без использования какой-либо иерархии, в «плоском» пространстве имен. У каждого объекта имеется уникальный во всем кластере идентификатор. Объект состоит из непосредственно самих бинарных данных и метаданных. Метаданные представляют собой пары имя/значение.

Кластер RADOS состоит из узлов, на которых работают два типа демонов:

- **Object Storage Device** (OSD) – отвечает непосредственно за хранение данных. Обычно один демон OSD связан с одним физическим диском. Таким образом, на одном узле хранения может быть одновременно несколько OSD, и число демонов OSD в кластере обычно соответствует общему числу физических дисков. OSD сами непосредственно выполняют операции по репликации, ребалансировке и восстановлению данных. Информацию о состоянии данных они передают мониторам. Для объектов OSD могут выполнять главную (primary) или вспомогательную (secondary) роль. Именно OSD с первичной ролью выполняют запросы на ввод/вывод, отвечают за репликацию и целостность данных, ребалансировку данных и восстановление;
- **Ceph monitors** (MON) – поддерживает мастер-копию карты состояния кластера и информацию о его текущем состоянии. Карта состояния кластера состоит из пяти карт, хранящих информацию о кластере и конфигурации. Все узлы кластера отправляют информацию мониторам о каждом изменении в их состоянии. В кластере должно быть нечетное число MON. Минимальное число мониторов для отказоустойчивой инфраструктуры – три. Для тестов и в лабораторном окружении можно использовать один. Кластер будет доступным до тех пор, пока доступно более половины MON.

Нужно также отметить важную роль файловой системы, с которой работают OSD. Файловая система, созданная на диске для хранения данных, должна поддерживать расширенные атрибуты файлов. В расширенных атрибутах файлов хранится информация о состояниях объектов, моментальных снимков и списков контроля доступа. Для промышленной эксплуатации на настоящий момент рекомендуются в первую очередь XFS. Также можно выбрать ext4, но она не настолько популярна при работе с Ceph. Лучших

результатов при тестировании производительности кластера Ceph позволяет добиться Btrfs. Однако по ряду мнений она на настоящий момент недостаточно стабильна для промышленного применения. В любом случае, решение о выборе файловой системы как множества других параметров внедрения стоит принимать в зависимости от конкретного проекта. Для тестовых внедрений Ceph файловую систему Btrfs с большой долей уверенности можно назвать хорошим кандидатом. В версии Jewel также появилась технология под названием BlueStore. BlueStore использует для хранения данных «сырые» диски без файловой системы. В этой книге BlueStore не рассматривается.

Также, говоря о хранении данных, нужно отметить, что OSD использует отдельный небольшой раздел или диск, а иногда и файл в качестве журнала, по аналогии с журналами файловых систем. При этом Ceph сначала пишет все данные в журнал, а потом переносит их в место постоянного хранения. Данные считаются записанными на кластер Ceph, когда из журнала скопировано минимальное число реплик. Рекомендуется размещать журналы на SSD-дисках.

Для определения местоположения данных кластер использует псевдослучайный алгоритм распределения данных **CRUSH (Controlled Replication Under Scalable Hashing)**, который эффективно распределяет реплики объектов по узлам кластера. Описание CRUSH доступно по ссылке http://ceph.com/papers/weil-crush-sc06.pdf. Вместо того чтобы обращаться к некой центральной точке хранения информации о расположении объектов для каждого запроса, CRUSH вычисляет положение данных. Алгоритм CRUSH при помощи созданных под конкретное внедрение карт (CRUSH map) учитывает физическую инфраструктуру кластера в виде иерархии (диски, пулы, узлы и т. д.) и политику распределения данных. Это отличает CRUSH, например, от подобного же принципа определения места положения данных в другом SDS с открытым исходным кодом GlusterFS.

Объекты хранятся в логических группах – пулах. Пул можно рассматривать как тег, привязанный к ряду объектов. На пулы также можно привязать, например, права пользователей. Разные пулы могут иметь разное число реплик или использовать технологию erasure coding (EC). EC в настоящее время рекомендуется только при работе Ceph в качестве объектного хранилища. При использовании пула erasure coding в Ceph как блочного или файлового хра-

нилища значительно падает производительность. Таким образом, вы также при помощи пулов можете разделять данные по методу доступа.

Пулы делятся на группы (placement groups, PG). PG – это набор объектов, которые реплицируются OSD для обеспечения сохранности данных. Объект принадлежит одной PG, и все объекты, принадлежащие одной PG, имеют один хэш, используемый алгоритмом CRUSH. Рекомендуется от пятидесяти до ста PG в расчете на один OSD. PG решают проблему масштабирования пулов при значениях в миллионы объектов.

Вкратце поиск объекта клиентом происходит следующим образом. Клиент обращается к монитору и получает последнюю карту кластера, которая помогает клиенту понять состояние и конфигурацию кластера. Далее, обладая картой, на основании имени объекта и имени пула CRUSH может вычислить PG и главный OSD для записи или чтения данных. При этом клиент напрямую обращается к OSD.

Следующий уровень на нашем рисунке, описывающем архитектуру Ceph, – библиотека **librados**. Эта библиотека написана на языке программирования C и позволяет приложениям работать напрямую с RADOS, не используя каких-либо иных интерфейсов. Другие интерфейсы, такие как RBD, RADOSGW и CephFS, работают поверх librados. Подобные librados библиотеки также позволяют разработчикам на других языках программирования получать напрямую доступ к кластеру RADOS. В число языков программирования входят: C++, Java, Python, Ruby и PHP. Возможность работы напрямую с кластером RADOS может быть дополнительным преимуществом хранилища данных при построении своих сервисов поверх IaaS.

Одним из наиболее востребованных методов доступа к кластеру Ceph является блочный доступ, осуществляемый при помощи **RBD (RADOS block device)**. Чаще всего блочное устройство Ceph используется библиотекой librbd совместно с QEMU и libvirt. Также имеется драйвер RBD, интегрированный в ядро Linux (Kernel RDB, или сокращенно KRDB). Вместе с сервисами OpenStack Glance и Cinder в качестве хранилища Ceph RDB представляет собой дешевую альтернативу проприетарным SAN. Поддерживаются copy-on-write и мгновенное клонирование виртуальных машин, мгновенные снимки и загрузка виртуальных машин с образов, хранящихся в кластере Ceph.

В качестве примера ниже приведен вывод с одного из вычислительных узлов OpenStack, где в качестве системы хранения данных используется Ceph. Приведена только часть командной строки запуска виртуальной машины, а именно опция «-drive». Это лучшим образом иллюстрирует интеграцию Ceph с QEMU/KVM:

```
# ps aux | grep qemu-kvm
5636 ? Sl 1:49 /usr/libexec/qemu-kvm -name instance-...
-drive file=rbd:vms/73f..._disk:id=openstack:key=BF3...==:auth_
supported=cephx\;none:mon_host=192.168.0.24\:6789\;192.16
8.0.25\:6789\;192.168.0.26\:6789,if=none,id=drive-virtio-
disk0,format=raw,cache=none...
```

Минусом для других применений Ceph RBD является отсутствие поддержки в иных операционных системах, кроме Linux, и других распространенных гипервизорах, например VMware ESXi или Hyper-V.

Следующим компонентом является **RADOSGW (Ceph Object Gateway)**. Данный компонент проксирует HTTP-запросы от и к RADOS, предоставляя объектное хранилище, совместимое с OpenStack Swift и Amazon S3. Третьим поддерживаемым является Admin API для управления кластером через RESTful-интерфейс. S3 и Swift API можно использовать одновременно и для общего пространства имен.

Еще один тип доступа – файловый при помощи **CephFS (Ceph file system)**. Файловая система требует не показанного на рисунке компонента – сервиса метаданных MDS. В настоящий момент CephFS не готова для промышленного применения и далее в книге не рассматривается.

Наконец, последний опциональный компонент, не показанный на рисунке, – это **Calamari**, который является веб-инструментом мониторинга и управления кластером Ceph. Изначально этот инструмент был частью проприетарного продукта Inktank Ceph Enterprise. В 2014 году исходный код Calamari был открыт (https://github.com/ceph/calamari).

Начнем с установки и подготовки операционной системы для узлов кластера. Как и в предыдущих главах, воспользуемся дистрибутивом CentOS 7. Нам необходимы минимум три виртуальные машины. Каждая виртуальная машина должна иметь два сетевых адаптера и минимум 1 Гб оперативной памяти. К каждой виртуальной машине подключим по четыре виртуальных диска. Один будет использоваться под операционную систему, и три – под хранение данных кластером Ceph.

Быстрая установка кластера Ceph при помощи ceph-deploy

Покажем два варианта установки кластера: при помощи утилиты ceph-deploy и вручную. Также распространенным вариантом является установка при помощи Ansible. Для первого варианта нам нужна «машина администратора», где запускается ceph-deploy. Мы используем для этих целей узел ceph1 для экономии ресурсов. На практике, как правило, это отдельный узел, не входящий в кластер.

Первое, что необходимо, – установить саму операционную систему. В качестве варианта установки можно выбрать Minimal или Server with GUI.

После установки операционной системы обновите все установленные пакеты командой

```
# yum -y update
```

Следующее – это добавление репозиториев с пакетами OpenStack и дополнительных пакетов. Начнем с необходимого репозитория Extra Packages for Enterprise Linux (EPEL). Данный репозиторий содержит пакеты, предназначенные для RHEL и его производных, коим является CentOS. Как правило, там содержатся пакеты, которые присутствуют в Fedora, но которые компания Red Hat не включила в свой промышленный дистрибутив. В частности, пакеты Ceph мы будем ставить именно оттуда:

```
# yum -y install epel-release
```

Далее для упрощения отладки мы отключим сервис брандмауэра:

```
# systemctl stop firewalld.service
# systemctl disable firewalld.service
```

Для версий Ceph вплоть до Infernalis необходимо отключить или перевести в режим Permissive SELinux.

Также убедитесь, что все взаимодействующие виртуальные машины могут разрешать имена друг друга.

В первом варианте для установки кластера мы будем использовать стандартную для Ceph утилиту ceph-install. Безусловно, это только один из возможных методов. Зачастую кластер Ceph устанавливается теми же инструментами, что и компоненты OpenStack.

Одну из виртуальных машин мы будем использовать в качестве узла, с которого произведем установку при помощи утилиты ceph-install. В качестве предварительного требования для ceph-install необходимо на всех трех машинах создать пользователя, которым будет осуществляться установка. Этот пользователь на всех трех узлах должен иметь право выполнять команды при помощи sudo с привилегиями администратора и без ввода пароля. В качестве имени пользователя нельзя использовать «ceph», поскольку это имя зарезервировано для демонов. Кроме того, пользователь должен иметь право подключаться по ssh с узла, где выполняется ceph-install, на все остальные узлы без ввода пароля.

Создадим на всех трех виртуальных машинах пользователя cephinstall и настроим sudo:

```
[root@ceph1 ~]# useradd cephinstall
[root@ceph1 ~]# passwd cephinstall
[root@ceph1 ~]# cat << EOF >/etc/sudoers.d/cephinstall
> cephinstall ALL = (root) NOPASSWD:ALL
> Defaults:cephinstall !requiretty
> EOF
[root@ceph1 ~]# chmod 0440 /etc/sudoers.d/cephinstall
```

Обратите внимание, что также необходимо отключить в sudo параметр `requiretty`.

Теперь разрешаем с узла ceph1 подключаться на все три виртуальные машины по ssh без пароля:

```
[root@ceph1 ~]# su - cephinstall
[cephinstall@ceph1 ~]$ ssh-keygen
[cephinstall@ceph1 ~]$ ssh-copy-id cephinstall@ceph1
[cephinstall@ceph1 ~]$ ssh-copy-id cephinstall@ceph2
[cephinstall@ceph1 ~]$ ssh-copy-id cephinstall@ceph3
```

Рекомендуется создать конфигурационный файл ~/.ssh/config, указав, каким пользователем мы подключаемся к каждому из узлов:

```
Host ceph1
   Hostname ceph1
   User cephinstall
Host ceph2
   Hostname ceph2
   User cephinstall
Host ceph3
   Hostname ceph3
   User cephinstall
```

Не забудьте про правильные разрешения, которые более строгие, чем задающиеся по умолчанию:

```
[cephinstall@ceph1 test-cluster]$ chmod 600 /home/cephinstall/.ssh/config
```

Утилита ceph-deploy сохраняет конфигурационные файлы и ключи в текущей рабочей директории. Создадим директорию для файлов нашего тестового кластера:

```
[cephinstall@ceph1 ~]$ mkdir test-cluster
[cephinstall@ceph1 ~]$ cd test-cluster
```

Первым шагом создадим новый кластер. По окончании выполнения команды убедимся, что в рабочей директории появились три файла: конфигурационный файл, файл связки ключей и журнал установки:

```
[cephinstall@ceph1 test-cluster]$ ceph-deploy new ceph1
[cephinstall@ceph1 test-cluster]$ ls
ceph.conf  ceph.log  ceph.mon.keyring
```

Теперь можно начать править конфигурационный файл. Мы создавали виртуальные машины с двумя сетевыми интерфейсами. Один из них предназначен для внешних клиентов, другой – для внутренней сети кластера. Добавим в ceph.conf соответствующие опции:

```
public_network = 192.168.122.11/24
cluster_network = 192.168.222.11/24
```

По умолчанию число реплик равняется трем, и минимальное число реплик, для того чтобы кластер оставался в работоспособном состоянии, – две. Для нашего тестового окружения уменьшим эти числа до двух и одного соответственно:

```
osd_pool_default_size = 2
osd_pool_default_min_size = 1
```

Необходимо заметить, что вместо символов подчеркивания можно использовать пробелы.

Установим на все три узла пакеты. Можно открыть на узле ceph2 или ceph3 файл системного журнала для мониторинга происходящего:

```
[root@ceph2 ~]# tail -f /var/log/messages
[cephinstall@ceph1 test-cluster]$ ceph-deploy install ceph1 ceph2
ceph3
```

```
...
[ceph3][DEBUG ] Complete!
[ceph3][INFO  ] Running command: sudo ceph --version
[ceph3][DEBUG ] ceph version 0.94.5 (9764da52395923e0b32908d83a9
f7304401fee43)
```

Вывод говорит о том, что пакеты версии Hammer (0.94) установлены. Если по каким-то причинам, например в случае сбоя, вам необходимо будет начать все заново, то это можно сделать, отдав команды

```
[cephinstall@ceph1 test-cluster]$ ceph-deploy purgedata ceph1
ceph2 ceph3
[cephinstall@ceph1 test-cluster]$ ceph-deploy forgetkeys
```

Создадим наш первый монитор на узле ceph1:

```
[cephinstall@ceph1 test-cluster]$ ceph-deploy mon create-initial
```

После добавим OSD сначала на узле ceph2. На каждом из узлов ceph2 и ceph3 у нас по четыре диска. Первый занят операционной системой. Три оставшихся мы используем под данные кластера. Первым делом рекомендуется дать команду `ceph-deploy disk zap`, которая удалит разделы с дисков. Перед запуском команды уточните имена дисков в вашей тестовой среде. Далее приведен пример для VirtIO дисков в KVM/QEMU, где они имеют имена vda, vdb и т. д. Помним, что первый диск занимает операционная система:

```
[cephinstall@ceph1 test-cluster]$ ceph-deploy disk zap ceph2:vdb
ceph2:vdc ceph2:vdd
```

Теперь подготовим диски на узле ceph2. На каждом из них будет создано по два раздела, второй из них будет использован под журнал. Мы также могли бы задать конкретное расположение журнала, указав имя раздела после каждого диска через двоеточие. Зачастую несколько журналов одного сервера помещают на выделенный ssd-диск.

```
[cephinstall@ceph1 test-cluster]$ ceph-deploy osd prepare
ceph2:vdb ceph2:vdc ceph2:vdd
...
[ceph_deploy.osd][DEBUG ] Host ceph2 is now ready for osd use.
```

Можно проверить для одного из дисков непосредственно на узле ceph2:

```
[root@ceph2 ~]# fdisk -l /dev/vdb
...
Disk label type: gpt
#         Start          End    Size  Type            Name
 1     10487808    104857566     45G  unknown         ceph data
 2         2048     10485760      5G  unknown         ceph journal
```

Шага активации OSD, как это описано в документации на сайте проекта, больше не требуется. Теперь те же действия можно повторить для узла ceph3. Способ с использованием ceph-deploy подходит, когда нужно быстро развернуть кластер ceph. Для того чтобы лучше разобраться с настройками, установим кластер вручную.

Установка кластера ceph вручную

Установим на всех узлах пакеты ceph:

```
# yum -y install ceph ceph-radosgw
```

Теперь начнем на узле ceph1, который играет роль монитора, создавать конфигурационный файл /etc/ceph/ceph.conf. Во-первых, нам понадобится уникальный идентификатор кластера UUID. Проще всего сгенерировать случайный UUID при помощи стандартной утилиты из пакета util-linux:

```
$ uuidgen
1d618cdf-c648-4e9c-8aed-c170577a5d83
```

Добавим в /etc/ceph/ceph.conf первую строчку:

```
fsid = 1d618cdf-c648-4e9c-8aed-c170577a5d83
```

Дальше пропишем имя нашего первого MON-узла и его IP-адрес. Можно было бы указать их не один, а несколько, но в нашем случае это только узел ceph1:

```
mon initial members = ceph1
mon host = 192.168.122.11
```

Нам нужно создать два набора ключей: monitor keyring и administrator keyring, которые будут использоваться в качестве общего секрета для взаимодействия мониторов и для доступа администратора к управлению кластером при помощи утилиты ceph:

```
[root@ceph1 ~]# ceph-authtool --create-keyring /tmp/ceph.mon.keyring
--gen-key -n mon. --cap mon 'allow *'
```

```
creating /tmp/ceph.mon.keyring
[root@ceph1 ~]# ceph-authtool --create-keyring /etc/ceph/ceph.client.
admin.keyring --gen-key -n client.admin --set-uid=0 --cap mon 'allow *'
--cap osd 'allow *' --cap mds 'allow'
creating /etc/ceph/ceph.client.admin.keyring
```

Теперь необходимо добавить ключ client.admin в связку ключей ceph.mon.keyring:

```
[root@ceph1 ~]# ceph-authtool /tmp/ceph.mon.keyring --import-keyring
/etc/ceph/ceph.client.admin.keyring
importing contents of /etc/ceph/ceph.client.admin.keyring into /tmp/
ceph.mon.keyring
```

Утилитой monmaptool создадим карту монитора:

```
[root@ceph1 ~]# monmaptool --create --add ceph1 192.168.122.11
--fsid 1d618cdf-c648-4e9c-8aed-c170577a5d83 /tmp/monmap
monmaptool: monmap file /tmp/monmap
monmaptool: set fsid to 1d618cdf-c648-4e9c-8aed-c170577a5d83
monmaptool: writing epoch 0 to /tmp/monmap (1 monitors)
```

После создания можно просмотреть содержимое карты, которая задает фиксированные адреса узлов:

```
[root@ceph1 ~]# monmaptool --print /tmp/monmap
monmaptool: monmap file /tmp/monmap
epoch 0
fsid 1d618cdf-c648-4e9c-8aed-c170577a5d83
last_changed 2015-11-13 07:15:04.636212
created 2015-11-13 07:15:04.636212
0: 192.168.122.11:6789/0 mon.ceph1
```

Создадим директорию вида /var/lib/ceph/mon/имя_кластера-имя_монитора, где монитор будет хранить свои данные:

```
[root@ceph1 ~]# mkdir /var/lib/ceph/mon/ceph-ceph1/
```

Заполним директорию данными демона монитора (карта и связка ключей):

```
[root@ceph1 ~]# ceph-mon --mkfs -i ceph1 --monmap /tmp/monmap --keyring
/tmp/ceph.mon.keyring
ceph-mon: set fsid to 1d618cdf-c648-4e9c-8aed-c170577a5d83
ceph-mon: created monfs at /var/lib/ceph/mon/ceph-ceph1 for mon.ceph1
[root@ceph1 ~]# ls /var/lib/ceph/mon/ceph-ceph1/
keyring  store.db
```

Добавим общие настройки в файл конфигурации /etc/ceph/ceph.conf:

```
 1  [global]
 2  fsid = 1d618cdf-c648-4e9c-8aed-c170577a5d83
 3  mon initial members = ceph1
 4  mon host = 192.168.122.11
 5  public network = 192.168.122.0/24
 6  auth cluster required = cephx
 7  auth service required = cephx
 8  auth client required = cephx
 9  osd journal size = 1024
10  osd pool default size = 2
11  osd pool default min size = 1
12  osd pool default pg num = 128
13  osd pool default pgp num = 128
```

Номера строк приведены для удобства. В реальный конфигурационный файл их добавлять не надо. Прокомментируем настройки. Строки с шестой по восьмую указывают на механизм аутентификации. Далее – размер журнала в килобайтах. В двенадцатой и тринадцатой строках задано число групп (placement groups, PG) для пула по умолчанию. Как обсуждалось ранее, PG – это набор объектов, которые реплицируются OSD для обеспечения сохранности данных. Pg_num и pgp_num должны совпадать. PGP (placment group for placment) влияет на ребалансировку кластера при увеличении числа PG. В конце добавим в конфигурационный файл секцию, описывающую наш единственный монитор:

```
[mon.ceph1]
host = ceph1
mon addr = 192.168.122.11:6789
```

Создадим файл, указывающий, что настройка монитора закончена:

```
[root@ceph1 ~]# touch /var/lib/ceph/mon/ceph-ceph1/done
```

Теперь можно попытаться стартовать монитор:

```
[root@ceph1 ~]# /etc/init.d/ceph start mon.ceph1
=== mon.ceph1 ===
Starting Ceph mon.ceph1 on ceph1...
Running as unit run-1045.service.
Starting ceph-create-keys on ceph1...
```

Проверим, что создан пул по умолчанию rbd:

```
[root@ceph1 ~]# ceph osd lspools
0 data,1 metadata,2 rbd,
```

и что монитор запущен:

```
[root@ceph1 ~]# ceph -s | grep mon
     monmap e1: 1 mons at {ceph1=192.168.122.11:6789/0}, election
epoch 1, quorum 0 ceph1
```

Перейдем к установке OSD на узлы ceph2 и ceph3. Предполагаем, что на каждом из них по четыре диска, три из которых мы будем использовать под хранение данных. На дисках, которые предполагается использовать под данные, создайте по разделу и файловую систему XFS на каждом. Также добавьте все три диска в /etc/fstab. По умолчанию предполагается, что раздел монтируется в директорию /var/lib/ceph/osd/имя_кластера-идентификатор_OSD. Продемонстрируем на примере одного из разделов:

```
[root@ceph2 ~]# mkfs.xfs /dev/vdb1
[root@ceph2 ~]# mkdir /var/lib/ceph/osd/ceph-0
[root@ceph2 ~]# echo "/dev/vdb1 /var/lib/ceph/osd/ceph-0 xfs
defaults 0 1" >> /etc/fstab
[root@ceph2 ~]# mount -a
```

Добавим в конфигурационный файл на узле ceph1 описание шести OSD, по три на узел. OSD с номерами от нуля до двух будут на узле ceph2, а с трех до пяти – на узле ceph3:

```
[osd.0]
host = ceph2
...
[osd.5]
host = ceph3
```

Скопируем отредактированный конфигурационный файл с первого узла на оба оставшихся. То же самое проделаем с файлом /etc/ceph/ceph.client.admin.keyring.

По числу OSD на каждом из узлов запускаем команду

```
[root@ceph3 ~]# ceph osd create
5
```

При этом будет выдаваться номер идентификатора OSD. Поскольку у нас по три диска и соответственно три OSD на каждом узле, то всего мы выдаем эту команду шесть раз, по три раза на каждом из двух узлов. В примере выше это – вывод последнего выполнения команды из шести (OSD нумеруются начиная с нуля).

Для каждого из дисков/OSD выполняем команды вида:

```
[root@ceph3 ~]# ceph-osd -i 5 --mkfs --mkkey
[root@ceph3 ~]# ceph auth add osd.5 osd 'allow *' mon 'allow
profile osd' -i /var/lib/ceph/osd/ceph-5/keyring
```

Первой командой мы инициализируем директорию, в которую смонтирована каждая из файловых систем. Второй командой мы регистрируем ключ аутентификации этого конкретного OSD. Ключ создавался во время инициализации предыдущей командой. Просмотреть список ключей кластера можно командой `ceph auth list`.

Для каждого из двух узлов выполняем команды:

```
[root@ceph2 ~]# ceph --cluster ceph osd crush add-bucket ceph2 host
added bucket ceph3 type host to crush map
[root@ceph2 ~]#  ceph osd crush move ceph2 root=default
moved item id -2 name 'ceph2' to location {root=default} in crush map
```

Тем самым мы добавили узел в карту алгоритма CRUSH и поместили его в корень default.

По окончании настройки просмотреть карту и веса OSD можно будет командой

```
[root@ceph1 ~]# ceph osd tree
# id    weight    type name     up/down        reweight
-1      6         root default
-2      3                 host ceph2
0       1                       osd.0  up      1
1       1                       osd.1  up      1
2       1                       osd.2  up      1
-3      3                 host ceph3
3       1                       osd.3  up      1
4       1                       osd.4  up      1
5       1                       osd.5  up      1
```

Для каждого из шести OSD выполняем следующую команду, добавляя их в CRUSH-карту:

```
[root@ceph3 ~]# ceph osd crush add osd.5 1.0 host=ceph3
add item id 5 name 'osd.5' weight 1 at location {host=ceph3} to
crush map
```

После этого можно запустить скрипт /etc/init.d/ceph start и проверить состояние нашего кластера:

```
[root@ceph1 ~]# ceph -s
    cluster 1d618cdf-c648-4e9c-8aed-c170577a5d83
     health HEALTH_OK
     monmap e1: 1 mons at {ceph1=192.168.122.11:6789/0}, election
epoch 1, quorum 0 ceph1
```

```
     osdmap e38: 6 osds: 6 up, 6 in
      pgmap v94: 192 pgs, 3 pools, 0 bytes data, 0 objects
            3275 MB used, 593 GB / 596 GB avail
                 192 active+clean
```

Также нужно знать команду ceph –w, которая аналогична предыдущей, но позволяет отслеживать состояние кластера в реальном времени. Приведем еще несколько полезных команд. Ceph df покажет свободное место в кластере и распределение его по пулам:

```
[root@ceph1 ~]# ceph df
GLOBAL:
    SIZE      AVAIL      RAW USED     %RAW USED
    596G       593G         3275M          0.54
POOLS:
    NAME          ID     USED      %USED     MAX AVAIL     OBJECTS
    data          0         0         0          296G           0
    metadata      1         0         0          296G           0
    rbd           2         0         0          296G           0
```

Естественно, что у нас доступно все пространство на дисках кластера.

Интеграция Ceph с сервисами OpenStack

Начнем с интеграции Ceph с Cinder. Первым делом установим на все вычислительные узлы и контроллер OpenStack пакеты Ceph:

```
# yum -y install python-rbd ceph
```

Далее нужно на эти же узлы скопировать с одной из машин кластера Ceph конфигурационный файл /etc/ceph/ceph.conf.

Создадим для Cinder новый пул, как мы обсуждали это ранее. Выполнять команду необходимо на узле, где присутствует связка ключей /etc/ceph/ceph.client.admin.keyring:

```
[root@ceph1 ~]# ceph osd pool create cinder-volumes 128
pool 'cinder-volumes' created
```

Поскольку мы используем аутентификацию cephx, то создаем пользователя для сервиса Cinder и даем ему права на пул cinder-volumes:

```
[root@ceph1 ~]# ceph auth get-or-create client.cinder mon 'allow r'
osd 'allow class-read object_prefix rbd_children, allow rwx pool=cinder-
volumes'
[client.cinder]
    key = AQAIsW1WMIoLCBAA9xvL26nqMFVeIURAOyqBTA==
```

Создаем и копируем связку ключей для cinder на управляющий узел и на все вычислительные узлы:

```
[root@ceph1 ~]# ceph auth get-or-create client.cinder | ssh root@controller
tee /etc/ceph/ceph.client.cinder.keyring
[root@ceph1 ~]# ssh root@controller chown cinder:cinder /etc/ceph/ceph.client.
cinder.keyring
[root@ceph1 ~]# ceph auth get-or-create client.cinder | ssh root@compute tee
/etc/ceph/ceph.client.cinder.keyring
[root@ceph1 ~]# ceph auth get-key client.cinder | ssh root@compute tee client.
cinder.key
```


На вычислительных узлах нам необходимо добавить в libvirt общий секрет. Сгенерируем случайный uuid, например при помощи команды uuidgen:

```
$ uuidgen
7651c383-1aef-4644-b7fe-d9cbbd9fe116
```

Далее, используя этот uuid, вы должны создать файл такого содержания:

```
<secret ephemeral='no' private='no'>
  <uuid>7651c383-1aef-4644-b7fe-d9cbbd9fe116</uuid>
  <usage type='ceph'>
    <name>client.cinder secret</name>
  </usage>
</secret>
```

Про формат файла секрета можно почитать по ссылке http://libvirt.org/formatsecret.html. На всех вычислительных узлах выполним команды:


```
[root@compute ~]#  virsh secret-define --file имя_файла.xml
Secret 7651c383-1aef-4644-b7fe-d9cbbd9fe116 created
[root@compute ~]# virsh secret-set-value --secret 7651c383-1aef-
4644-b7fe-d9cbbd9fe116 --base64 $(cat client.cinder.key) && rm
client.cinder.key secret.xml
```

Первой мы создали секрет согласно описанию из XML-файла, а второй командой мы присвоили ему в качестве содержимого client.cinder.key, предварительно переведя в кодировку Base64.

Теперь перейдем на управляющий узел OpenStack. Нам нужно внести дополнения в конфигурационный файл Cinder /etc/cinder/cinder.conf:


```
[DEFAULT]
enabled_backends = rbd
[rbd]
volume_driver = cinder.volume.drivers.rbd.RBDDriver
rbd_pool = cinder-volumes
rbd_ceph_conf = /etc/ceph/ceph.conf
rbd_flatten_volume_from_snapshot = false
rbd_max_clone_depth = 5
rbd_store_chunk_size = 4
rados_connect_timeout = -1
glance_api_version = 2
rbd_user = cinder
rbd_secret_uuid = 7651c383-1aef-4644-b7fe-d9cbbd9fe116
```

После чего осталось отредактировать только /etc/nova/nova.conf на вычислительных узлах. Нам необходимо добавить секцию libvirt:

```
[libvirt]
rbd_user = cinder
rbd_secret_uuid = 7651c383-1aef-4644-b7fe-d9cbbd9fe116
```

Потом рестартуем сервисы nova-compute и cinder. Теперь можно попробовать создать том блочного устройства:

```
$ cinder create --display-name chephtest1 1
+---------------------------------------+--------------------------------------+
|                Property               |                Value                 |
+---------------------------------------+--------------------------------------+
|              attachments              |                  []                  |
|           availability_zone           |                 nova                 |
|                bootable               |                false                 |
|          consistencygroup_id          |                 None                 |
|               created_at              |      2015-12-13T19:18:34.000000      |
|              description              |                 None                 |
|               encrypted               |                False                 |
|                   id                  | 417d122b-55a3-4667-95be-7beb34a5a93d |
|               metadata                |                  {}                  |
|            migration_status           |                 None                 |
|              multiattach              |                False                 |
|                  name                 |               chephtest1             |
|         os-vol-host-attr:host         |                 None                 |
|     os-vol-mig-status-attr:migstat    |                 None                 |
|     os-vol-mig-status-attr:name_id    |                 None                 |
|      os-vol-tenant-attr:tenant_id     |   8cc74ebe8da94fe0a7ac6cf54f31b420   |
|   os-volume-replication:driver_data   |                 None                 |
```

```
| os-volume-replication:extended_status |                  None                  |
|           replication_status          |                disabled                |
|                  size                 |                   1                    |
|              snapshot_id              |                  None                  |
|              source_volid             |                  None                  |
|                 status                |                creating                |
|               user_id                 |   6310882678344e8183f2d7e886088293   |
|              volume_type              |                  None                  |
+---------------------------------------+----------------------------------------+
```

Проверим список томов и убедимся, что chephtest1 доступен:

```
$ cinder list
+------+-----------+-----------------+------------+------+-------------+----------+-------------+-------------+
| ID   |   Status  | Migration Status |    Name    | Size | Volume Type | Bootable | Multiattach | Attached to |
+------+-----------+-----------------+------------+------+-------------+----------+-------------+-------------+
| 41.. | available |        -        | chephtest1 |  1   |      -      |  false   |    False    |             |
+------+-----------+-----------------+------------+------+-------------+----------+-------------+-------------+
```

Проверим на узле Ceph, что том действительно занял место и хранится в пуле cinder-volumes:

```
[root@ceph1 ~]# ceph df
GLOBAL:
    SIZE      AVAIL      RAW USED      %RAW USED
    596G       593G         3276M           0.54
POOLS:
    NAME               ID     USED      %USED     MAX AVAIL     OBJECTS
    data               0         0         0          296G           0
    metadata           1         0         0          296G           0
    rbd                2         0         0          296G           0
    cinder-volumes     3        16         0          296G           3
[root@ceph1 ~]# rados -p cinder-volumes ls
rbd_id.volume-417d122b-55a3-4667-95be-7beb34a5a93d
rbd_directory
rbd_header.121a4262753d
```

Следующим шагом настроим интеграцию с Glance. Сразу отметим, что с Ceph рекомендуется использовать RAW-образы дисков, а не QCOW2, как мы это делали ранее. Это связано с тем, что при загрузке с RAW-диска Ceph делает мгновенную копию образа (zero byte). При этом мгновенные копии делаются даже не на уровне RADOS, а на уровне RBD, так что тип локальной файловой системы на узлах хранения в этом случае не важен. Если используется формат QCOW2, то Ceph придется полностью копировать весь образ, поскольку «родной» функционал copy-on-write образов QCOW2 в данном случае не поддерживается.

В случае если вы все-таки используете QCOW2-формат, Nova и Cinder при запуске виртуальной машины должны конвертировать образ в RAW, перед тем как передать его драйверу RBD в эмуляторе QEMU. Это значит, что сперва QCOW2-образ выкачивается на вычислительный узел, конвертируется и потом закачивается в виде RAW обратно в Ceph.

Как и в случае с Cinder, первым делом создадим пул для хранения образов:

```
[root@ceph1 ~]# ceph osd pool create images 128
```

Затем знакомая операция по созданию пользователя и выдаче ему права на пул images:

```
[root@ceph1 ~]# ceph auth get-or-create client.glance mon 'allow r' osd
'allow class-read object_prefix rbd_children, allow rwx pool=images'
[client.glance]
    key = AQDoxW1WiGBrLhAA3ydL2Lr6eHNjBh6u4w0AMw==
```


Создаем и копируем связку ключей на управляющий узел и выставим соответствующие права доступа к файлу:

```
[root@ceph1 ~]# ceph auth get-or-create client.glance | ssh root@
controller tee /etc/ceph/ceph.client.glance.keyring
[root@ceph1 ~]# ssh root@controller chown glance:glance /etc/
ceph/ceph.client.glance.keyring
```


Теперь добавим в /etc/glance/glance-api.conf секцию glance_store и в default – show_image_direct_url. Последняя опция включает клонирование «copy-on-write»:

```
[glance_store]
default_store = rbd
stores = rbd
rbd_store_pool = images
rbd_store_user = glance
rbd_store_ceph_conf = /etc/ceph/ceph.conf
rbd_store_chunk_size = 8
[DEFAULT]
show_image_direct_url = True
```

Важное замечание: в версиях OpenStack Juno или младше параметр default_store = rbd должен быть в секции default. Последним шагом настройки рестартуем сервисы glance-api и glance-registry.

Можно попробовать загрузить образ в сервис Glance. Для этого скачаем образ cirros и сконвертируем его из QCOW2 в формат RAW:

```
$ wget -P /tmp http://download.cirros-cloud.net/0.3.4/cirros-0.3.4-x86_64-
disk.img
$ qemu-img convert -O raw /tmp/cirros-0.3.4-x86_64-disk.img /tmp/cirros-raw.
img
```

Загружаем образ в Glance:

```
$ glance image-create --name "cirros-raw" --visibility public
--disk-format raw --container-format=bare --file /tmp/cirros-raw.img
+------------------+--------------------------------------+
| Property         | Value                                |
+------------------+--------------------------------------+
| checksum         | 56730d3091a764d5f8b38feeef0bfcef     |
| container_format | bare                                 |
| created_at       | 2015-12-13T19:47:05Z                 |
| disk_format      | raw                                  |
| id               | 153618d8-c4e2-4983-8b94-756285194489 |
| min_disk         | 0                                    |
| min_ram          | 0                                    |
| name             | cirros-raw                           |
| owner            | 8cc74ebe8da94fe0a7ac6cf54f31b420     |
| protected        | False                                |
| size             | 41126400                             |
| status           | active                               |
| tags             | []                                   |
| updated_at       | 2015-12-13T19:47:11Z                 |
| virtual_size     | None                                 |
| visibility       | public                               |
+------------------+--------------------------------------+
```

И по аналогии с Cinder, командами `rados -p images ls` и `ceph df` убеждаемся, что Ceph используется для хранения образов.

Последним шагом настраиваем сервис Nova. Создадим пул vms для виртуальных машин:

```
[root@ceph1 ~]# ceph osd pool create vms 128
```

В конфигурационный файл ceph.conf на всех узлах добавляем настройки клиента:

```
[client]
    rbd cache = true
    rbd cache writethrough until flush = true
    admin socket = /var/run/ceph/guests/$cluster-$type.$id.$pid.$cctid.asok
    log file = /var/log/qemu/qemu-guest-$pid.log
    rbd concurrent management ops = 20
```

Создаем объявленные в конфигурационном файле директории и выставляем права файловой системы:

```
[root@compute ~]# mkdir -p /var/run/ceph/guests/ /var/log/qemu/
[root@compute ~]# chown qemu.qemu /var/run/ceph/guests /var/log/qemu/
```

Теперь обновляем nova.conf на вычислительных узлах:

```
[libvirt]
images_type = rbd
images_rbd_pool = vms
images_rbd_ceph_conf = /etc/ceph/ceph.conf
disk_cachemodes="network=writeback"
hw_disk_discard = unmap
```

После чего рестартуем сервис nova-compute. Последнее, что нам нужно, – это обновить права клиента client.cinder, которого мы будем использовать при запуске экземпляров виртуальных машин. Для корректной работы требуется доступ rwx для пулов cinder-volumes и vms. Для images достаточно rx:

```
[root@ceph1 ~]# ceph auth caps client.cinder mon 'allow r'
osd 'allow class-read object_prefix rbd_children, allow rwx
pool=cinder-volumes, allow rwx pool=vms, allow rx pool=images'
updated caps for client.cinder
```

Проверяем разрешения:

```
[root@ceph1 ~]# ceph auth list
...
client.cinder
    key: AQAIsW1WMIoLCBAA9xvL26nqMFVeIURAOyqBTA==
    caps: [mon] allow r
    caps: [osd] allow class-read object_prefix rbd_children, allow
rwx pool=cinder-volumes, allow rwx pool=vms, allow rx pool=images
client.glance
    key: AQDoxW1WiGBrLhAA3ydL2Lr6eHNjBh6u4w0AMw==
    caps: [mon] allow r
    caps: [osd] allow class-read object_prefix rbd_children, allow
rwx pool=images
```

Теперь можно попробовать запустить виртуальную машину:

```
$ nova boot --flavor m2.tiny --image cirros-raw --key-name demokey1 --security-
groups demo-sgroup test-vm
+--------------------------------------+-----------------------------------+
| Property                             | Value                             |
+--------------------------------------+-----------------------------------+
| OS-DCF:diskConfig                    | MANUAL                            |
| OS-EXT-AZ:availability_zone          |                                   |
| OS-EXT-STS:power_state               | 0                                 |
| OS-EXT-STS:task_state                | scheduling                        |
```

```
| OS-EXT-STS:vm_state                  | building                             |
| OS-SRV-USG:launched_at               | -                                    |
| OS-SRV-USG:terminated_at             | -                                    |
| accessIPv4                           |                                      |
| accessIPv6                           |                                      |
| adminPass                            | ZcEMr8RVHZ9q                         |
| config_drive                         |                                      |
| created                              | 2015-12-14T12:52:16Z                 |
| flavor                               | m2.tiny (98cb36fb-3541-415b-9835-b..) |
| hostId                               |                                      |
| id                                   | eec707f7-55f4-4ac3-9df8-7a684048aac1 |
| image                                | cirros-raw (153618d8-c4e2-4983-8b9..) |
| key_name                             | demokey1                             |
| metadata                             | {}                                   |
| name                                 | test-vm                              |
| os-extended-volumes:volumes_attached | []                                   |
| progress                             | 0                                    |
| security_groups                      | demo-sgroup                          |
| status                               | BUILD                                |
| tenant_id                            | eca00feab38e4aa5b462bd31af0b9dca     |
| updated                              | 2015-12-14T12:52:16Z                 |
| user_id                              | 924c18c923654d7c930bcb1044580d8b     |
+--------------------------------------+--------------------------------------+
```

Перейдем на тот вычислительный узел, где запущена виртуальная машина, и посмотрим параметры запуска процесса, для того чтобы убедиться, что виртуальная машина стартовала с Ceph. В выводе интересующая нас опция выделена жирным:

```
[root@compute-opt ~]# ps aux | grep qemu-kvm
qemu      2751  8.0  2.9 852144 118200 ?       Sl   15:52   0:20 /usr/
libexec/qemu-kvm -name instance-00000100 -S -machine pc-i440fx-… -drive
file=rbd:vms/eec707f7-55f4-4ac3-9df8-7a684048aac1_disk:id=cinder:key=AQ
AIsW1WMIoLCBAA9xvL26nqMFVeIURAOyqBTA==:auth_supported=cephx\;none:mon_
host=192.168.122.11\:6789,...
```

Ну и посмотрим на пул vms:

```
[root@ceph1 ~]# ceph df
GLOBAL:
    SIZE     AVAIL     RAW USED     %RAW USED
    596G      593G        3425M          0.56
POOLS:
    NAME               ID     USED       %USED     MAX AVAIL     OBJECTS
...
    vms                5      32768k         0          296G          11
```

На этом базовую интеграцию Ceph и OpenStack можно считать законченной.

Глава 16

Отказоустойчивость и производительность OpenStack

В данной главе кратко рассматриваются лишь некоторые из вопросов, которые следует иметь в виду при промышленном использовании OpenStack.

Обзор способов обеспечения высокой доступности сервисов облака

Под высокой доступностью понимается минимизация времени недоступности системы и потери данных.

Большинство служб OpenStack является сервисами без сохранения состояния (stateless), предоставляющими свой API клиентам этих служб. В противоположность сервисам с сохранением состояния (stateful), для них легче обеспечить отказоустойчивость и масштабирование. Проще всего описать работу такой службы на бытовом уровне «сделал и забыл». Отказоустойчивость в подобном случае достигается введением избыточного числа взаимозаменяемых узлов, на которых запускаются экземпляры сервиса и балансировщика нагрузки. В этом случае, если один из узлов выйдет из строя, балансировщик нагрузки просто перенаправит запросы на оставшиеся серверы.

Обслуживают сервисы OpenStack служебные инфраструктурные компоненты, например база данных и брокер сообщений. Их назначение – хранение постоянной информации и межсервисное взаимодействие. Большинство инфраструктурных служб может

работать в режиме «active-active», когда все узлы кластера могут обрабатывать запросы, в этом случае за операции запуска, остановки и мониторинга сервисов отвечает systemd. Однако часть из сервисов поддерживает только «active-passive». В случае «active-passive» активным является лишь один из узлов, а остальные работают в так называемом «холодном резерве». При этом обычно запросы идут на виртуальный IP-адрес, который в конкретный момент времени принадлежит одному, активному узлу. Мониторингом сервиса и переключением ресурсов поверх systemd в этом случае занимается ПО кластера, как правило, Pacemaker (http://clusterlabs.org/).

Отдельно автор хочет отметить наличие руководства по высокой доступности сервисов среди документации на сайте OpenStack – https://docs.openstack.org/ha-guide/index.html. Для целей обучения данное руководство, скорее всего, будет полезнее документации, которую можно найти в комплекте конкретных дистрибутивов OpenStack, поскольку их разработчики при развертывании отказоустойчивой конфигурации, как правило, полагаются на свою систему инсталляции (Fuel, Crowbar, TripleO и др.). Также в качестве полезного ресурса можно назвать руководство Understanding Red Hat OpenStack Platform High Availability из комплекта справочной документации RHOSP – https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/.

На рис. 16.1 приведен пример того, как обеспечивается отказоустойчивость сервисов OpenStack, распределенных по трем управляющим узлам после установки RHOSP 10 (Newton). Настройка и установка производились автоматически с использованием заранее подготовленных шаблонов. Рассмотрение промышленной установки OpenStack не входит в задачи этой книги. Однако данная диаграмма может дать представление, как сервисы распределяются в реальной жизни в «боевых» внедрениях. В верхней части изображены сервисы в кластере Pacemaker плюс ресурсы кластера, в частности виртуальные IP-адреса. В нижней части автор изобразил сервисы, доступ к которым осуществляет HAProxy. Справа приведен вывод команды netstat для публичного виртуального IP-ареса (ресурса кластера Pacemaker), который на данном развертывании был 172.17.2.19. Как вы видите, перечисленные порты совпадают с портами служб, работающими с HAProxy.

В двух словах перечислим, как организуется отказоустойчивость для ряда основных сервисов.

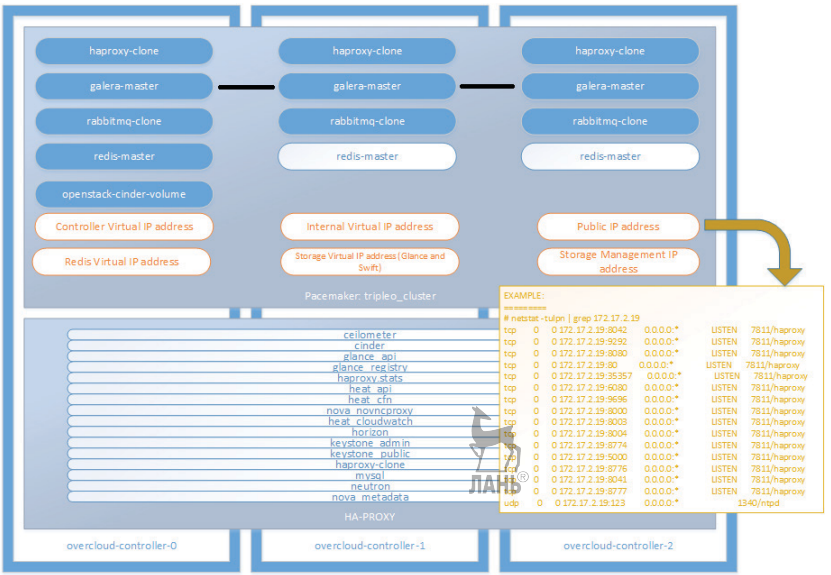


Рис. 16.1. Пример отказоустойчивости сервисов overcloud в RHOSP 10

База данных – один из важнейших компонентов, для которого необходимо обеспечить высокую доступность. В случае MySQL/MariaDB доступны два варианта обеспечения отказоустойчивости: при помощи кластера Pacemaker и общей системы хранения данных. При ее отсутствии можно использовать, например, систему репликации DRBD (http://drbd.linbit.com/). Второй вариант – это при помощи кластера Galera, который является подключаемым модулем репликации для MySQL. На момент написания этой главы вариант при помощи Galera являлся предпочтительным. Нужно отметить, что Galera является системой, работающей на основе кворума, что значит – у вас должно быть минимум три узла в кластере.

Большинство высокоуровневых компонентов OpenStack общается друг с другом посредством шины сообщений. Выход из строя брокера сообщений повлечет за собой нарушение работоспособности сервисов облака. Согласно информации почтовой рассылки openstack-operators, в большинстве внедрений вместо балансировщика нагрузки предпочитают использовать режим «active-active» при помощи нескольких экземпляров RabbitMQ. Три возможных варианта построения распределенной архитектуры рассмотрены

на странице https://www.rabbitmq.com/distributed.html. Самый распространенный в настоящее время – это зеркалирование очередей.

Как правило, для обеспечения работы в режиме «active-active» для сервисов без сохранения состояния из свободных балансировщиков нагрузки выбирают HAProxy (http://www.haproxy.org/). Такие сервисы, как nova-api, glance-api, keystone-api, neutron-api, cinder-api и nova-scheduler, не требуют особого внимания. Однако, как и для многих веб-приложений, для Horizon необходима привязка сессии к серверу, которая осуществляется средствами балансировщика нагрузки.

К числу сервисов, как правило, работающих в режиме «active-passive», относят в первую очередь Cinder-volume, ceilometer-central и агенты Neutron. Их отказоустойчивость обеспечивается при помощи Pacemaker. При этом обеспечить отказоустойчивость L3 агента (в режиме «active-passive») можно также при помощи протокола Virtual Router Redundancy Protocol (VRRP, RFC 3768). В последнем случае виртуальный маршрутизатор выбирается случайным образом из нескольких сетевых узлов, остальные при этом работают в «горячем резерве». Второй способ, появившийся в OpenStack, позднее заключается в использовании распределенного виртуального маршрутизатора (DVR). В случае DVR не только сетевые, но и каждый из вычислительных узлов обладает функционалом маршрутизации.

Выделение вычислительных ресурсов

Говоря о ресурсах, нужно заметить, что OpenStack позволяет подтверждать виртуальным машинам больше физической памяти и вычислительных ресурсов, чем имеется в наличии. По умолчанию планировщик ассоциирует с одним физическим или Hyper-threading-ядром 16 виртуальных процессоров (vCPU). За это значение в конфигурационном файле /etc/nova/nova.conf отвечает параметр

```
cpu_allocation_ratio=16.0
```

К примеру, по умолчанию один восьмиядерный процессор со включенной технологией Hyper-threading даст нам 8*2*16 = 256 vCPU.

Если запускается много требовательных к процессору приложений типа Hadoop или виртуальных машин с сетевыми функциями (NFV), то коэффициент выставляется ближе к 2, а иногда и к 1. Если основная нагрузка веб-сервера, то число можно увеличить вплоть до 16, заданного по умолчанию. Если вы не можете разделить типы нагрузки, то можно попробовать использовать коэффициент от 2 до 5.

Рекомендуется, по возможности, ограничить в своем облаке использование flavor (типы виртуальных машин) с числом vCPU более одного. Для гипервизоров первого типа намного проще выделить ресурсы виртуальной машине с 1 vCPU. Например, для выделения вычислительных ресурсов машине типа m1.xlarge планировщику гипервизора необходимо будет ждать, пока не освободятся восемь физических ядер центрального процессора. Также использование виртуальных машин с меньшим числом vCPU снижает риск того, что эти vCPU будут выделены на разных физических процессорах.

В современных системах рекомендуется включать технологию Hyper-threading. Производительность Hyper-threading, по сравнению с физическим ядром, отличается от приложения к приложению. Потенциально могут существовать достаточно редкие приложения, которые выиграют от отключения Hyper-threading. Вместо отключения Hyper-threading в таких случаях рекомендуется добавить виртуальной машине с таким приложением дополнительные vCPU.

Выделение оперативной памяти

Объем памяти, выделяемой виртуальным машинам по умолчанию, в полтора раза больше, чем имеющийся физический. За это значение в конфигурационном файле /etc/nova/nova.conf отвечает параметр

```
ram_allocation_ratio=1.5
```

В целом по рекомендациям в списках рассылки OpenStack для памяти выбирают значение 0,9. Также рекомендуется задать резервирование оперативной памяти при помощи параметра reserved_host_memory_mb в nova.conf. Обычно в расчетах можно руководствоваться закладыванием на накладные расходы порядка 100 Мб на одну виртуальную машину. Обязательно нужно предусмотреть swap, как минимум вдвое больший, чем этот параметр. Для процессора коэффициент сильно зависит от нагрузки. Обычно

память становится раньше «бутылочным горлышком», чем ресурсы центрального процессора.

Параметры ram_allocation_ratio и cpu_allocation_ratio необходимо менять на всех управляющих узлах, если у вас их более одного. После изменения параметров необходимо рестартовать сервис планировщика nova-scheduler.

Полезной командой для сбора статистики по использованию оперативной памяти будет `nova diagnostics`. По умолчанию ее выполнять может только пользователь admin. В качестве аргумента команде необходим идентификатор экземпляра виртуальной машины:

```
$ nova diagnostics 6aec269e-2633-4c56-9a61-7b8cb084995e
+--------------------------+-------------+
| Property                 | Value       |
+--------------------------+-------------+
| cpu0_time                | 34150000000 |
| memory                   | 307200      |
| memory-actual            | 307200      |
| memory-rss               | 127812      |
| tap6fc6b1e7-e7_rx        | 10177       |
| tap6fc6b1e7-e7_rx_drop   | 0           |
| tap6fc6b1e7-e7_rx_errors | 0           |
| tap6fc6b1e7-e7_rx_packets | 97          |
| tap6fc6b1e7-e7_tx        | 11576       |
| tap6fc6b1e7-e7_tx_drop   | 0           |
| tap6fc6b1e7-e7_tx_errors | 0           |
| tap6fc6b1e7-e7_tx_packets | 116         |
| vda_errors               | -1          |
| vda_read                 | 17636864    |
| vda_read_req             | 731         |
| vda_write                | 324608      |
| vda_write_req            | 98          |
+--------------------------+-------------+
```

Суммарную статистику по гипервизорам можно посмотреть при помощи команды `nova hypervisor-stats`:

```
$ nova hypervisor-stats
+----------------------+-------+
| Property             | Value |
+----------------------+-------+
| count                | 2     |
| current_workload     | 0     |
| disk_available_least | 95    |
| free_disk_gb         | 97    |
| free_ram_mb          | 6580  |
```

```
| local_gb              | 98    |
| local_gb_used         | 1     |
| memory_mb             | 7904  |
| memory_mb_used        | 1324  |
| running_vms           | 1     |
| vcpus                 | 8     |
| vcpus_used            | 1     |
+-----------------------+-------+
```

Еще одна полезная подкоманда – `virsh dommemstat`, которую необходимо использовать непосредственно на вычислительном узле. Для начала следует узнать имя виртуальной машины:

```
[root@compute ~]# virsh list
 Id    Name                           State
----------------------------------------------------
 2     instance-00000007              running
```

После чего можно отдать непосредственно команду `virsh dommemstat`:

```
[root@compute ~]# virsh dommemstat instance-00000007
actual 307200
rss 129636
```

Также автор может порекомендовать запись в личном блоге о структурах памяти Linux: http://markelov.blogspot.ru/2009/01/linux-procmeminfo.html.

Повышение производительности виртуальных машин

Поскольку виртуальные машины в случае использования гипервизора KVM – это процессы GNU/Linux, к ним применимы многие «общие» технологии настройки производительности. Рассмотрим одну из технологий – закрепление за виртуальными машинами определенных процессоров (CPU pinning). При закреплении процессоров за виртуальными машинами обеспечивается сопоставление ядер процессора и vCPU виртуальной машины. Это делается, во-первых, для того чтобы убедиться, что виртуальные машины запускаются на определенных ядрах при планировании нагрузки, во-вторых, что ресурсы этих ядер не используются процессами операционной системы гипервизора. Продемонстрируем это на практике. Начнем с изменения коэфициента сопоставления физических и виртуальных процессоров на единицу:

```
[root@compute-opt ~]# crudini --set /etc/nova/nova.conf DEFAULT
cpu_allocation_ratio 1
```

Просмотрим конфигурацию узлов NUMA, которая необходима нам для дальнейшего планирования:

```
[root@compute-opt ~]# yum -y install numactl
[root@compute-opt ~]# numactl --hardware
available: 1 nodes (0)
node 0 cpus: 0 1 2 3
node 0 size: 8095 MB
node 0 free: 6828 MB
node distances:
node   0
  0:  10
```

В нашем случае у нас только одна зона, и вся память с точки зрения доступа стоит одинаково. Кроме того, мы видим, что у нас четыре ядра. Зададим в настройках Nova, что виртуальные машины будут изолированы на ядрах с первого по второе:

```
[root@compute-opt ~]# crudini --set /etc/nova/nova.conf DEFAULT
vcpu_pin_set 2-3
```

Зарезервируем память под процессы операционной системы гипервизора:

```
[root@compute-opt ~]# crudini --set /etc/nova/nova.conf DEFAULT
reserved_host_memory_mb 512
```

Теперь необходимо убедиться, что процессы операционной системы не будут запускаться на ядрах, зарезервированных для Nova. Добавим в конце командной строки загрузки ядра параметр isolcpus:

```
[root@compute-opt ~]# cat /etc/default/grub
GRUB_TIMEOUT=5
GRUB_DISTRIBUTOR="$(sed 's, release .*$,,g' /etc/system-release)"
GRUB_DEFAULT=saved
GRUB_DISABLE_SUBMENU=true
GRUB_TERMINAL_OUTPUT="console"
GRUB_CMDLINE_LINUX="crashkernel=auto rd.lvm.lv=centos/root rd.lvm.
lv=centos/swap rhgb quiet isolcpus=2-3"
GRUB_DISABLE_RECOVERY="true"
[root@compute-opt ~]# grub2-mkconfig > /boot/grub2/grub.cfg
Generating grub configuration file ...
...
done
```

После чего необходимо перезагрузить сервер.

В девятой главе мы уже рассматривали агрегаторы узлов (Host Aggregates). Для запуска виртуальных машин, чьи vCPU должны быть изолированы на выделенных ядрах гипервизора, мы и воспользуемся агрегаторами совместно со специально созданным типом виртуальных машин (flavor). Создаем агрегатор:

```
$ source keystonerc_admin
$ openstack aggregate create perfvnf
+-------------------+----------------------------+
| Field             | Value                      |
+-------------------+----------------------------+
| availability_zone | None                       |
| created_at        | 2018-03-12T10:45:08.142708 |
| deleted           | False                      |
| deleted_at        | None                       |
| id                | 1                          |
| name              | perfvnf                    |
| updated_at        | None                       |
+-------------------+----------------------------+
```

Добавляем свойство, при помощи которого будем сопоставлять агрегатор с типом виртуальной машины:

```
$ openstack aggregate set --property pinned=true perfvnf
```

Включаем хост compute-opt.test.local в созданный агрегатор:

```
$ openstack aggregate add host perfvnf compute-opt.test.local
+-------------------+----------------------------+
| Field             | Value                      |
+-------------------+----------------------------+
| availability_zone | None                       |
| created_at        | 2018-03-12T10:45:08.000000 |
| deleted           | False                      |
| deleted_at        | None                       |
| hosts             | [u'compute-opt.test.local'] |
| id                | 1                          |
| metadata          | {u'pinned': u'true'}       |
| name              | perfvnf                    |
| updated_at        | None                       |
+-------------------+----------------------------+
```

Создадим второй агрегатор со свойством `pinned=false`, в который включим оставшийся гипервизор. На втором гипервизоре будут запускаться виртуальные машины, не требующие привязки к ядрам:

```
$ openstack aggregate create perfitapp
+-------------------+----------------------------+
| Field             | Value                      |
+-------------------+----------------------------+
| availability_zone | None                       |
```

```
| created_at          | 2018-03-12T11:04:10.859056 |
| deleted             | False                      |
| deleted_at          | None                       |
| id                  | 2                          |
| name                | perfitapp                  |
| updated_at          | None                       |
+---------------------+----------------------------+
$ openstack aggregate set --property pinned=false perfitapp
$ openstack aggregate add host perfitapp compute.test.local
+---------------------+----------------------------+
| Field               | Value                      |
+---------------------+----------------------------+
| availability_zone   | None                       |
| created_at          | 2018-03-12T11:04:10.000000 |
| deleted             | False                      |
| deleted_at          | None                       |
| hosts               | [u'compute.test.local']    |
| id                  | 2                          |
| metadata            | {u'pinned': u'false'}      |
| name                | perfitapp                  |
| updated_at          | None                       |
+---------------------+----------------------------+
```

Создадим новый тип виртуальной машины с соответствующими свойствами:

```
$ source keystonerc_admin
$ openstack flavor create --ram 2048 --disk 1 --vcpus 2 vnf
+----------------------------+--------------------------------------+
| Field                      | Value                                |
+----------------------------+--------------------------------------+
| OS-FLV-DISABLED:disabled   | False                                |
| OS-FLV-EXT-DATA:ephemeral  | 0                                    |
| disk                       | 1                                    |
| id                         | 9ef16daa-08de-4d64-8f70-5df9dda34a85 |
| name                       | vnf                                  |
| os-flavor-access:is_public | True                                 |
| properties                 |                                      |
| ram                        | 2048                                 |
| rxtx_factor                | 1.0                                  |
| swap                       |                                      |
| vcpus                      | 2                                    |
+----------------------------+--------------------------------------+
$ openstack flavor set --property hw:cpu_policy=dedicated vnf
$ openstack flavor set --property hw:cpu_thread_policy=prefer vnf
$ openstack flavor set --property aggregate_instance_extra_
specs:pinned=true vnf
```

Осталось добавить два фильтра, ServerGroupAntiAffinityFilter и ServerGroupAffinityFilter, в число включенных фильтров планировщика на управляющем узле:

```
[root@controller ~]# crudini --set /etc/nova/nova.conf enabled_filters
RetryFilter,AvailabilityZoneFilter,ComputeFilter,ComputeCapabilitiesFilter,
ImagePropertiesFilter,ServerGroupAntiAffinityFilter,ServerGroupAffinityFilter
[root@controller ~]# systemctl restart openstack-nova-scheduler
```

Теперь можно попробовать создать виртуальную машину:

```
$ source keystonerc_demo
$ openstack server create --image cirros-0.4.0-x86_64 --flavor vnf --key-name
demokey1 --nic net-id=demo-net myvnf1
+-----------------------------+------------------------------------------------------+
| Field                       | Value                                                |
+-----------------------------+------------------------------------------------------+
| OS-DCF:diskConfig           | MANUAL                                               |
| OS-EXT-AZ:availability_zone |                                                      |
| OS-EXT-STS:power_state      | NOSTATE                                              |
| OS-EXT-STS:task_state       | scheduling                                           |
| OS-EXT-STS:vm_state         | building                                             |
| OS-SRV-USG:launched_at      | None                                                 |
| OS-SRV-USG:terminated_at    | None                                                 |
| accessIPv4                  |                                                      |
| accessIPv6                  |                                                      |
| addresses                   |                                                      |
| adminPass                   | mnYzp4UVtbNp                                         |
| config_drive                |                                                      |
| created                     | 2018-03-12T12:45:20Z                                 |
| flavor                      | vnf (9ef16daa-08de-4d64-8f70-5df9dda34a85)           |
| hostId                      |                                                      |
| id                          | ed1e84ed-6194-437d-8b72-36b0178fe9c9                 |
| image                       | cirros-0.4.0-x86_64 (c8ccc9b3-29bb-4220-be38-8f261ac8b99a) |
| key_name                    | demokey1                                             |
| name                        | myvnf1                                               |
| progress                    | 0                                                    |
| project_id                  | bc10ac4b71164550a363b8098e8ad270                     |
| properties                  |                                                      |
| security_groups             | name='default'                                       |
| status                      | BUILD                                                |
| updated                     | 2018-03-12T12:45:20Z                                 |
| user_id                     | 3b76dece42b140e092dc1a76a85c1879                     |
| volumes_attached            |                                                      |
+-----------------------------+------------------------------------------------------+
```

Убеждаемся, что она создана на узле compute-opt и используется привязка к конкретным ядрам:

```
[root@compute-opt ~]# virsh list
 Id    Name                           State
----------------------------------------------------
 1     instance-00000013              running
[root@compute-opt ~]# virsh dumpxml instance-00000013
<domain type='kvm' id='1'>
```

```
  <name>instance-00000013</name>
  <uuid>ed1e84ed-6194-437d-8b72-36b0178fe9c9</uuid>
  <metadata>
    <nova:instance xmlns:nova="http://openstack.org/xmlns/libvirt/
nova/1.0">
      <nova:package version="17.0.0-1.el7"/>
...
  <cputune>
    <shares>2048</shares>
    <vcpupin vcpu='0' cpuset='2'/>
    <vcpupin vcpu='1' cpuset='3'/>
    <emulatorpin cpuset='2-3'/>
  </cputune>
...
```

Если запустить несколько новых виртуальных машин с типом, отличным от vnf, мы увидим, что все они запускаются на гипервизоре compute.test.local.

Повышение производительности сети

В порядке снижения производительности сети виртуальных машин можно расположить технологии, используемые в OpenStack, следующим образом: SR-IOV, коммутатор Open vSwitch с поддержкой Intel DPDK и, наконец, стандартный Open vSwitch.

Сетевые карты с поддержкой расширения PIC Single Root I/O Virtualization (SR-IOV) позволяют предоставлять как изолированный, так и общий доступ к аппаратным возможностям функций PCIe «железа» (физическим и виртуальным). Сетевую карту может между собой делить несколько виртуальных машин. Трафик, минуя гипервизор, поступает от виртуальной функции сетевой карты к виртуальной машине. Благодаря этому достигается производительность, близкая к производительности самой физической карты, и практически исключаются накладные расходы на виртуализацию. Как минус технологии можно отметить то, что при этом не поддерживается живая миграция виртуальных машин и экземпляр виртуальной машины становится «привязан» к физическому хосту.

Intel Data Plane Development Kit – это набор библиотек, обеспечивающий быструю обработку пакетов в пространстве пользователя. Приложения могут обрабатывать трафик напрямую с сетевой карты. В RDO Open vSwitch с поддержкой DPDK поставляется в пакете openvswitch-dpdk. Это медленнее, чем использование SR-IOV, но быстрее, чем стандартный OVS. Для работы

Open vSwitch с поддержкой DPDK требуется одна из совместимых сетевых карт. Пример вывода скрипта из состава DPDK, определяющего наличие совместимой карты:

```
# dpdk_nic_bind.py --status

Network devices using DPDK-compatible driver
============================================
0000:08:00.0 '82599 10 Gigabit Dual Port Backplane Connection'
drv=igb_uio unused=ixgbe
0000:08:00.1 '82599 10 Gigabit Dual Port Backplane Connection'
drv=igb_uio unused=ixgbe

Network devices using kernel driver
===================================
0000:06:00.0 'BCM57840 NetXtreme II 10/20-Gigabit Ethernet'
if=eth0 drv=bnx2x unused=igb_uio
0000:06:00.1 'BCM57840 NetXtreme II 10/20-Gigabit Ethernet'
if=eth1 drv=bnx2x unused=igb_uio
0000:87:00.0 '82599 10 Gigabit Dual Port Backplane Connection'
if=eth4 drv=ixgbe unused=igb_uio
0000:87:00.1 '82599 10 Gigabit Dual Port Backplane Connection'
if=eth5 drv=ixgbe unused=igb_uio

Other network devices
=====================
<none>
```

В случае использования DPDK ускорение осуществляется за счет того, что ядро не участвует в обработке пакетов.

Нужно отметить, что каждая сетевая карта создает дополнительную нагрузку на процессор, таким образом, рекомендуется использовать в виртуальных машинах меньшее число сетевых карт, насколько это возможно.

Определение аппаратных требований к оборудованию

Для целей приблизительного «сайзинга» (определения достаточных аппаратных требований) можно воспользоваться калькулятором от Mirantis – https://www.mirantis.com/openstack-services/bom-calculator/.

Еще один калькулятор приводится в руководстве по планированию архитектуры OpenStack на официальном сайте OpenStack: https://github.com/noslzzp/cloud-resource-calculator/blob/master/cloud-resource-calculator.ods.

Заключение

Если вы добрались до заключения, пройдя вместе с автором путь по созданию своего первого демонстрационного стенда OpenStack, то цель, которая ставилась при написании книги, достигнута. Однако при этом вы только в начале пути по изучению и работе с проектом OpenStack. Не забывайте, что название книги содержит слово «введение». Следующим шагом автор рекомендует изучение документации с сайта http://docs.openstack.org. Также небесполезным будет подписаться на списки рассылки – https://www.openstack.org/community/. В первую очередь будет полезен список Operators и Announcements. Во время написания книги появился официальный список рассылки на русском языке, однако за последние полгода там не появилось ни одного сообщения. Из полезных ресурсов еще можно порекомендовать подписаться на рассылки статей http://superuser.openstack.org/articles и читать блоги сообщества: http://planet.openstack.org. К сожалению, русскоязычных материалов в Сети не так уж и много, но надеюсь, что с популяризацией OpenStack их число будет возрастать.

Ну и в конце хотелось бы поблагодарить читателя за то, что проделал этот путь с автором.

Приложение 1

Пример правил брандмауэра, реализующих группы безопасности на вычислительном узле

```
  1  -P INPUT ACCEPT
  2  -P FORWARD ACCEPT
  3  -P OUTPUT ACCEPT
  4  -N neutron-filter-top
  5  -N neutron-openvswi-FORWARD
  6  -N neutron-openvswi-INPUT
  7  -N neutron-openvswi-OUTPUT
  8  -N neutron-openvswi-i4194b0a8-7
  9  -N neutron-openvswi-local
 10  -N neutron-openvswi-o4194b0a8-7
 11  -N neutron-openvswi-s4194b0a8-7
 12  -N neutron-openvswi-sg-chain
 13  -N neutron-openvswi-sg-fallback
 14  -A INPUT -j neutron-openvswi-INPUT
 15  -A INPUT -i virbr0 -p udp -m udp --dport 53 -j ACCEPT
 16  -A INPUT -i virbr0 -p tcp -m tcp --dport 53 -j ACCEPT
 17  -A INPUT -i virbr0 -p udp -m udp --dport 67 -j ACCEPT
 18  -A INPUT -i virbr0 -p tcp -m tcp --dport 67 -j ACCEPT
 19  -A FORWARD -j neutron-filter-top
 20  -A FORWARD -j neutron-openvswi-FORWARD
 21  -A FORWARD -d 192.168.124.0/24 -o virbr0 -m conntrack
--ctstate RELATED,ESTABLISHED -j ACCEPT
 22  -A FORWARD -s 192.168.124.0/24 -i virbr0 -j ACCEPT
 23  -A FORWARD -i virbr0 -o virbr0 -j ACCEPT
```

```
    24  -A FORWARD -o virbr0 -j REJECT --reject-with icmp-port-
unreachable
    25  -A FORWARD -i virbr0 -j REJECT --reject-with icmp-port-
unreachable
    26  -A OUTPUT -j neutron-filter-top
    27  -A OUTPUT -j neutron-openvswi-OUTPUT
    28  -A OUTPUT -o virbr0 -p udp -m udp --dport 68 -j ACCEPT
    29  -A neutron-filter-top -j neutron-openvswi-local
    30  -A neutron-openvswi-FORWARD -m physdev --physdev-out
tap4194b0a8-77 --physdev-is-bridged -m comment --comment "Direct
traffic from the VM interface to the security group chain." -j
neutron-openvswi-sg-chain
    31  -A neutron-openvswi-FORWARD -m physdev --physdev-in
tap4194b0a8-77 --physdev-is-bridged -m comment --comment "Direct
traffic from the VM interface to the security group chain." -j
neutron-openvswi-sg-chain
    32  -A neutron-openvswi-INPUT -m physdev --physdev-in
tap4194b0a8-77 --physdev-is-bridged -m comment --comment "Direct
incoming traffic from VM to the security group chain." -j neutron-
openvswi-o4194b0a8-7
    33  -A neutron-openvswi-i4194b0a8-7 -m state --state
RELATED,ESTABLISHED -m comment --comment "Direct packets
associated with a known session to the RETURN chain." -j RETURN
    34  -A neutron-openvswi-i4194b0a8-7 -d 172.16.0.5/32 -p udp
-m udp --sport 67 --dport 68 -j RETURN
    35  -A neutron-openvswi-i4194b0a8-7 -d 255.255.255.255/32 -p
udp -m udp --sport 67 --dport 68 -j RETURN
    36  -A neutron-openvswi-i4194b0a8-7 -p tcp -m tcp --dport 22
-j RETURN
    37  -A neutron-openvswi-i4194b0a8-7 -p icmp -j RETURN
    38  -A neutron-openvswi-i4194b0a8-7 -m state --state INVALID
-m comment --comment "Drop packets that appear related to an
existing connection (e.g. TCP ACK/FIN) but do not have an entry
in conntrack." -j DROP
    39  -A neutron-openvswi-i4194b0a8-7 -m comment --comment
"Send unmatched traffic to the fallback chain." -j neutron-
openvswi-sg-fallback
    40  -A neutron-openvswi-o4194b0a8-7 -s 0.0.0.0/32 -d
255.255.255.255/32 -p udp -m udp --sport 68 --dport 67 -m comment
--comment "Allow DHCP client traffic." -j RETURN
    41  -A neutron-openvswi-o4194b0a8-7 -j neutron-openvswi-
s4194b0a8-7
    42  -A neutron-openvswi-o4194b0a8-7 -p udp -m udp --sport 68
--dport 67 -m comment --comment "Allow DHCP client traffic." -j
RETURN
    43  -A neutron-openvswi-o4194b0a8-7 -p udp -m udp --sport 67
--dport 68 -m comment --comment "Prevent DHCP Spoofing by VM." -j DROP
    44  -A neutron-openvswi-o4194b0a8-7 -m state --state
RELATED,ESTABLISHED -m comment --comment "Direct packets
associated with a known session to the RETURN chain." -j RETURN
```

```
    45  -A neutron-openvswi-o4194b0a8-7 -j RETURN
    46  -A neutron-openvswi-o4194b0a8-7 -m state --state INVALID
-m comment --comment "Drop packets that appear related to an
existing connection (e.g. TCP ACK/FIN) but do not have an entry
in conntrack." -j DROP
    47  -A neutron-openvswi-o4194b0a8-7 -m comment --comment
"Send unmatched traffic to the fallback chain." -j neutron-
openvswi-sg-fallback
    48  -A neutron-openvswi-s4194b0a8-7 -s 172.16.0.5/32 -m mac
--mac-source FA:16:3E:95:67:8B -m comment --comment "Allow traffic
from defined IP/MAC pairs." -j RETURN
    49  -A neutron-openvswi-s4194b0a8-7 -m comment --comment
"Drop traffic without an IP/MAC allow rule." -j DROP
    50  -A neutron-openvswi-sg-chain -m physdev --physdev-out
tap4194b0a8-77 --physdev-is-bridged -m comment --comment "Jump
to the VM specific chain." -j neutron-openvswi-i4194b0a8-7
    51  -A neutron-openvswi-sg-chain -m physdev --physdev-in
tap4194b0a8-77 --physdev-is-bridged -m comment --comment "Jump
to the VM specific chain." -j neutron-openvswi-o4194b0a8-7
    52  -A neutron-openvswi-sg-chain -j ACCEPT
    53  -A neutron-openvswi-sg-fallback -m comment --comment
"Default drop rule for unmatched traffic." -j DROP
```

Приложение 2

Листинг шаблона Heat

Запуск одной виртуальной машины – test-server.yml

```
heat_template_version: 2014-10-16
description: >
  OpenStack. Практическое знакомство с облачной операционной системой.
  Пример запуска одной ВМ

parameters:
  network:
    type: string
    description: Сеть экземпляра ВМ
    default: demo-net
  image:
    type: string
    description: Образ для запуска ВМ
    default: cirros-0.3.4-x86_64

resources:
  my_server:
    type: OS::Nova::Server
    properties:
      flavor: m2.tiny
      key_name: demokey1
      networks:
      - network: { get_param: network }
      image: { get_param: image }
      user_data: |
        #!/bin/sh
        echo "Instance STARTED!"
      user_data_format: RAW

outputs:
  instance_name:
    description: Имя экземпляра ВМ
    value: { get_attr: [my_server, name] }
  private_ip:
    description: IP-адрес ВМ в частной сети
    value: { get_attr: [ my_server, first_address ] }
```

Приложение 3

Список основных используемых службами OpenStack сетевых портов

Служба	Номер сетевого порта
Keystone – точка входа API администратора	35357
Keystone – точка входа публичного API	5000
Сервис Glance	9292
Glance реестр образов	9191
Блочное хранилище Cinder и iSCSI target	8776, 3260
Nova	8774
Nova API	8773, 8775
Доступ к консолям ВМ по протоколу VNC	5900–5999
VNC прокси для доступа к консоли ВМ браузером	6080
Прокси для клиента HTML5 при доступе к консоли ВМ	6082
Swift объектное хранилище и rsync	8080, 6000, 6001, 6002, 873
Сервис оркестрации Heat	8004
Сервис сети Neutron	9696
Сервис телеметрии	8041, 8042
Брокер сообщений AMQP	5672
База данных MySQL	3306

Книги издательства «ДМК Пресс» можно заказать в торгово-издательском холдинге «Планета Альянс» наложенным платежом, выслав открытку или письмо по почтовому адресу: **115487, г. Москва, 2-й Нагатинский пр-д, д. 6А**.

При оформлении заказа следует указать адрес (полностью), по которому должны быть высланы книги; фамилию, имя и отчество получателя. Желательно также указать свой телефон и электронный адрес.

Эти книги вы можете заказать и в интернет-магазине: **www.alians-kniga.ru**.

Оптовые закупки: **тел. +7 (499) 782-38-89**.

Электронный адрес: **books@alians-kniga.ru**.

Маркелов Андрей Александрович

OpenStack. Практическое знакомство с облачной операционной системой

Главный редактор *Мовчан Д. А.*
dmkpress@gmail.com

Корректор *Синяева Г. И.*
Верстка *Паранская Н. В.*
Дизайн обложки *Мовчан А. Г.*

Формат 60×90 1/16.
Печать офсетная. Усл. печ. л. 18,74.
Тираж 200 экз.

Веб-сайт издательства: www.дмк.рф