ОБЛАЧНЫЕ ВЫЧИСЛЕНИЯ

O'REILLY®

Джордж Риз

bhv®

Cloud Application Architectures

George Reese

O'REILLY®
Beijing • Cambridge • Farnham • Köln • Sebastopol • Taipei • Tokyo

Джордж Риз

ОБЛАЧНЫЕ ВЫЧИСЛЕНИЯ

Санкт-Петербург
«БХВ-Петербург»
2011

УДК 681.3.06
ББК 32.973.26-018.2
Р49

Риз Дж.

Р49 Облачные вычисления: Пер. с англ. — СПб.: БХВ-Петербург, 2011. — 288 с.: ил.

ISBN 978-5-9775-0630-4

Рассмотрена современная бизнес-модель, в которой вычислительные ресурсы предоставляются пользователям как услуги в сети Интернет. Дана практика разработки Web-приложений для развертывания в облачной среде и переноса в нее уже существующих приложений. Рассказано о приемах программирования и о навыках системного администрирования, которыми необходимо овладеть, чтобы успешно разрабатывать и сопровождать приложения, развертываемые в облаке.

Оценена целесообразность переноса существующих приложений в облачную среду как с технической, так и с экономической точек зрения. Описаны построение транзакционных Web-приложений и установка виртуальных серверов для их поддержки. Рассмотрены особенности подготовки плана аварийного восстановления в облачной среде. Показаны преимущества облачной инфраструктуры в области масштабирования приложений.

Для разработчиков

УДК 681.3.06
ББК 32.973.26-018.2

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Евгений Рыбаков*
Зав. редакцией	*Григорий Добин*
Перевод с англ.	*Ольги Кокоревой*
Редактор	*Екатерина Капалыгина*
Компьютерная верстка	*Натальи Смирновой*
Корректор	*Наталия Першакова*
Оформление обложки	*Елены Беляевой*
Зав. производством	*Николай Тверских*

Authorized translation of the English edition of Cloud Application Architectures, ISBN: 978-0-596-15636-7, Copyright © 2009 George Reese. Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472. All rights reserved. Russian edition copyright © 2011 year by BHV – St.Petersburg. All rights reserved. This translation is published and sold by permission of O'Reilly Media, Inc., the owner of all rights to publish and sell the same.

Авторизованный перевод английской редакции книги Cloud Application Architectures, ISBN: 978-0-596-15636-7, Copyright © 2009 George Reese. Издание O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472. Все права защищены. Русская редакция издания выпущена издательством БХВ – Петербург в 2011 году. Все права защищены. Перевод опубликован и продается с разрешения O'Reilly Media, Inc., собственника всех прав на публикацию и продажу издания.

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 30.12.10.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 23,22.
Тираж 1500 экз. Заказ №
"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

Санитарно-эпидемиологическое заключение на продукцию № 77.99.60.953.Д.005770.05.09
от 26.05.2009 г. выдано Федеральной службой по надзору
в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 978-0-596-15636-7 (англ.)
ISBN 978-5-9775-0630-4 (рус.)

© 2009 George Reese
© Перевод на русский язык "БХВ-Петербург", 2011

Оглавление

Об авторе

Джордж Риз (George Reese) является основателем двух компаний, расположенных в Миннеаполисе — enStratus Networks LLC (поставщик элитного инструментария, предназначенного для управления облачной инфраструктурой) и Valtira LLC (поставщик одноименной онлайновой платформы управления маркетингом). В течение последних 15 лет Джордж написал целый ряд книг об информационных технологиях, выпущенных издательством O'Reilly, в том числе: "MySQL Pocket Reference", "Database Programming with JDBC and Java" и "Java Database Best Practices".

На протяжении эры господства Интернета Джордж занимался созданием корпоративного инструментария для разработчиков и построением решений для маркетинга. Он являлся влиятельным участником процесса развития многопользовательских компьютерных игр из разряда так называемых "многопользовательских миров". Он является автором ряда библиотек Open Source MUD[1], и в 1996 году разработал первый драйвер JDBC — Open Source mSQL-JDBC. В последнее время Джордж занимается развертыванием транзакционных Web-приложений в облачной среде. До того как заняться разработкой Web-приложений, Джордж работал над моделированием крэш-тестов для Национального управления по безопасности автодорожного движения США (National Highway Traffic Safety Administration, NHTSA).

Джордж имеет степень бакалавра прикладных наук (философия), полученную в колледже Бейтса (Bates College), г. Льюистон, штат Мэн (**http://www.bates.edu/**), и степень магистра бизнеса (M.B.A.), полученную в Школе Менеджмента Келлог (Kellogg School of Management), г. Эванстон, штат Иллинойс (**http://www.kellogg.northwestern.edu/**). В настоящее время он проживает в штате Миннесота с женой Моникой и дочерьми Кирой и Линдси.

В процессе работы над книгой автору помогали его друзья и коллеги Рэнди Байас (Randy Bias) из компании GoGrid и Эрик "E. J". Джонсон (Eric "E. J" Johnson) из компании Rackspace. Во-первых, они рецензировали авторский текст, посвященный работе с Web-сервисами Amazon (Amazon Web Services), и, кроме того, они написали приложения, в которых рассказали о предложениях компаний GoGrid и Rackspace соответственно.

[1] MUD (Multi User Dungeon, Dimension или Domain, русский вариант — "Многопользовательский мир", МПМ). — *Прим. перев.*

- Приложение 2 "Облачный хостинг GoGrid" написано Рэнди Байесом.

  Рэнди Байес — вице-президент по технологическим стратегиям GoGrid, подразделения ServePath (**http://www.servepath.com/**). Он является признанным экспертом в области облачных вычислений и инфраструктур центров обработки данных, обладающим более чем 20-летним опытом в области предоставления услуг Интернета (ISP), предоставления услуг хостинга и разработки крупномасштабных инфраструктур. На все эти темы он часто пишет в своем личном блоге: **http://neotactics.com/blog**.

- Приложение 3 "Компания Rackspace" написано Эриком Джонсоном.

  Эрик Джонсон (Eric Johnson, или просто "E. J." — псевдоним, под которым он известен еще шире) является менеджером по развитию и разработке системы хранения данных Rackspace Cloud Files. К его дополнительной работе относятся исследования, проводимые в Racklabs, исследовательско-инженерном подразделении Rackspace.

  В течение последних 15 лет Эрик работает исключительно с использованием технологий Open Source, в том числе таких, как системное администрирование UNIX, сетевые технологии, администрирование баз данных и разработка программного обеспечения. Он выполняет исследовательские работы в области воздушного сообщения и аэрокосмических технологий. В течение целого ряда лет он вносит свой вклад в сообщество Open Source, разрабатывая патчи для SSH, поддерживая пакеты для Arch Linux и разрабатывая технические руководства для DNS/Bind.

  Эрик имеет степени бакалавра естественных наук (Bachelor of Science, B. S.) в области электротехники, полученную в Университете им. Дрекселя (Drexel University, Филадельфия, штат Пенсильвания, см. **http://www.drexel.edu/**), и магистра естественных наук (Master of Science, M. S.) в области компьютерных технологий, полученную в Политехническом институте Ренсселира (Rensselaer Polytechnic Institute, Трой, Нью-Йорк, см. **http://rpi.edu/**, **http://en.wikipedia.org/wiki/Rensselaer_Polytechnic_Institute**). В настоящее время вместе с семьей проживает в Сан-Антонио, штат Техас.

Предисловие переводчика

Облачные вычисления (cloud computing) — довольно известное на сегодняшний день явление, о котором говорится очень и очень много. Фактически это то, чем почти каждый из нас пользуется ежедневно. Это и почтовые сервисы, например, такие как Gmail, Yahoomail, Webmail, Hotmail и др. Это и онлайновые текстовые редакторы — скажем, Zoho Writer или Документы Google. И даже онлайновые сервисы по работе с графикой — например, Lunapic или всем известное онлайновое приложение Google Picasa, не говоря уже о набравших популярность музыкальных и видеосервисах. Их существует великое множество — всевозможных развлекательных и офисных либо творческих приложений на базе облачных технологий. Но есть и рынок облачных вычислений, где действительно крутятся огромные деньги!

И именно этому рынку как раз и посвящена книга, которую вы держите в руках. В ней рассматривается современная бизнес-модель, в которой вычислительные ресурсы предоставляются пользователям как услуги в сети Интернет. На простом и понятном примере самого известного "облака" компании Amazon объяснены как основные концепции, так и нюансы организации облачных вычислений, дано большое количество практических советов и полезных идей. Эта книга является признанным на Западе бестселлером, и одновременно с этим — первой книгой на русском языке, посвященной облачным вычислениям.

Несколько слов о том, что вы найдете в этой книге, и чего вы в ней не найдете. Основное внимание в книге уделено рассмотрению реальных решений, позволяющих сэкономить капиталовложения и трудозатраты, повысить надежность вычислительной инфраструктуры — поэтому ее можно рекомендовать, в первую очередь, системным и сетевым архитекторам, администраторам, разработчикам ПО, работающим в предприятиях малого и среднего бизнеса, государственных учреждениях. Сотрудников отделов IT крупных предприятий должны заинтересовать описанные способы повышения производительности и отказоустойчивости их вычислительных мощностей.

Большая часть материалов книги излагается на примере облачных сервисов компании Amazon — одного из ведущих представителей облачных провайдеров. Но при этом рассказывается и об универсальных подходах, используемых, например, такими провайдерами, как GoGrid и Rackspace. К сожалению, объять необъятное невозможно, поэтому в книге не описываются, например, такие платформы,

как Microsoft Azure (**http://www.microsoft.com/windowsazure/**)[1] или Eucalyptus (Elastic Utility Computing Architecture Linking Your Programs To Useful Systems, открытый проект, реализующий облачные вычисления в пределах центра управления данными, на котором концентрируются разработчики Ubuntu Server[2]). Чтобы отчасти скомпенсировать этот недостаток, группа подготовки издания дополнила книгу списком литературы, в котором освещаются различные аспекты облачных вычислений, а также списком русскоязычных ресурсов Интернета, которые помогут читателям сориентироваться в различных аспектах облачных вычислений в соответствии со стоящими перед ними конкретными задачами. Эта информация поможет читателям узнать, как обстоят дела с развитием облачных вычислений в России, и подскажет им, как максимально эффективно применять облачные сервисы в практической повседневной работе.

Зато огромным достоинством этой книги является концентрация внимания на таких вопросах, которые, безусловно, волнуют всех — и рядовых пользователей, и клиентов интернет-магазинов, и системных архитекторов, и разработчиков ПО. Как несложно догадаться, это вопросы по поводу конфиденциальности информации, находящейся за пределами ваших компьютеров, будь то домашний компьютер рядового пользователя Интернета или сервер компании. Ведь если вы перемещаете свою IT-деятельность в чужой центр обработки данных, вы кладете ее в "черный ящик". Применительно к облачным сервисам — вам не просто непонятно, что внутри, как оно работает, но и где ваши данные расположены физически. Наконец, еще один важный вопрос — а будет ли этот облачный сервис продолжать работать и завтра, и после завтра, и в более отдаленном будущем. Сможете ли вы вообще извлечь свои данные в случае наступления каких-либо форс-мажорных обстоятельств. Все эти вопросы изложены достаточно подробно, и это одно из важных достоинств книги, которые делают ее интересной для всех, кто интересуется облачными вычислениями.

---

[1] Подробнее о Microsoft Azure см. **http://www.osp.ru/pcworld/2010/09/13004178/**, **http://www.azuresupport.com/**. — *Прим. перев.*

[2] См. интервью с Сореном Хансеном (Søren Hansen) из команды виртуализации Ubuntu в журнале Linux Format (апрельский номер 2010 года), подробнее см. **https://help.ubuntu.com/community/Eucalyptus**, **http://packages.ubuntu.com/ru/lucid/eucalyptus-common**. — *Прим. перев.*

Введение

Облачными вычислениями (Cloud computing) на сегодняшний день интересуются все, и это действительно перспективное направление! Облачные вычисления представляют собой высокоэффективный инструмент повышения прибыли и расширения каналов продаж для независимых производителей программного обеспечения (Independent Software Vendors), операторов связи и VAR-посредников, расширяющих возможности существующих продуктов с целью их перепродажи конечным пользователям. Облачный подход позволяет организовать динамическое предоставление услуг, когда пользователи могут производить оплату по факту и регулировать объем своих ресурсов в зависимости от реальных потребностей без долгосрочных обязательств.

Благодаря облачным вычислениям компании могут предоставлять конечным пользователям удаленный динамический доступ к услугам, вычислительным ресурсам и приложениям (включая операционные системы и инфраструктуру) через Интернет. Вычислительные облака состоят из тысяч серверов, размещенных в физических и виртуальных центрах обработки данных, обеспечивающих работу десятков тысяч приложений, которые одновременно используют миллионы пользователей. По сравнению с традиционным подходом, облачные вычисления позволяют управлять более крупными инфраструктурами, обслуживать различные группы пользователей в пределах одного облака. Они обеспечивают огромный потенциал роста, но при этом дают возможность добиваться существенной экономии на масштабах.

Индустрия облачных вычислений стремительно развивается, и, по прогнозам аналитиков, к 2012 году на ее долю будет приходиться 9 % всех расходов на информационные инфраструктуры и услуги. Каким бы ни был бизнес вашей компании — разработка программного обеспечения, предоставление хостинга (hosting) или услуг связи, маркетинг или реклама — ваши клиенты наверняка тоже ожидают от вас движения в этом направлении.

Целевая аудитория этой книги

Я написал эту книгу, ориентируясь на технологов различных уровней подготовки и находящихся на всех ступенях "карьерной лестницы". Книга будет полезна

разработчикам, которым требуется писать код для приложений, входящих в облачные инфраструктуры, системным архитекторам, занятым проектированием систем для работы в облачных инфраструктурах, IT-менеджерам, отвечающим за перенос систем в облака.

В книге нет больших фрагментов кода, но время от времени я все же привожу примеры кодирования, демонстрирующие мой подход к решению различных задач. В основном я программирую на Java и Python и использую базы данных MySQL, хотя иногда прибегаю и к таким базам данных, как SQL Server или Oracle. Вместо того, чтобы приводить большие фрагменты кода на Java, я предпочел излагать методики, подходящие для любого языка программирования.

Если вы проектируете, строите или поддерживаете Web-приложения для развертывания в облаке, значит, эта книга — для вас.

Структура книги

Глава 1 универсальна и предназначена для широкого круга читателей. В ней описывается то, что я понимаю под "облаком", и поясняю, почему облако имеет такую ценность для предприятий и организаций. Эта глава дает материал на таком уровне и таким языком, чтобы финансовые директора ваших компаний (Chief Financial Officers, CFO) без проблем могли ее прочесть и понять, в чем ценность и важность облаков.

В *главе 2* приводится небольшой ознакомительный курс об облаке Amazon.

Цель данной книги заключается в предоставлении наилучших примеров, иллюстрирующих использование облачной инфраструктуры, вне зависимости от того, конкретно какую реализацию вы используете. На основании собственного опыта я могу сказать, что на данный момент именно облако Amazon и Amazon Web Services занимают самую внушительную рыночную нишу. По этой причине я считаю, что необходимо дать клиентам представление о том, как быстрее всего начать работу с облаком Amazon, а также предоставить наиболее общие сведения, необходимые для обсуждения дальнейших тем, освещаемых в данной книге, плюс определения базовых терминов, которыми мы впоследствии будем оперировать.

Если вас интересуют другие реализации облачной инфраструктуры, то и эту информацию я тоже предоставляю. Мне оказали помощь и содействие некоторые друзья и коллеги из таких компаний, как Rackspace и GoGrid. Эрик "E. J". Джонсон (Eric Johnson) из Rackspace выступил в качестве научного редактора книги и консультанта по вопросам, касающимся совместимости описываемого ПО с предложениями от компании Rackspace, а Ренди Байас (Randy Bias) из GoGrid сделал то же самое для предложений от своей компании.

Глава 3 содержит предварительную информацию, необходимую для того, чтобы разработать план вашей компании по переходу на облачную инфраструктуру, проанализировать этот план и приступить к его осуществлению.

Главы 4—7 предоставляют подробную информацию о построении Web-приложений для работы в облаке. *Глава 4* описывает архитектуры, основанные на транзакциях Web-приложений, и изменения, которые они должны претерпеть в облаке. *Глава 5* описывает основные проблемы, связанные с концепциями безопасности облачных вычислений. В *главе 6* рассказывается о том, как наилучшим образом спланировать и подготовить процедуры аварийного восстановления и обеспечить аварийное восстановление в кратчайшие сроки. Наконец, в *главе 7* мы обсудим, как облачные вычисления влияют на перспективы масштабирования приложений, включая автоматическое масштабирование Web-приложений.

Соглашения, использованные в этой книге

В книге используются следующие соглашения по оформлению текста:

- *курсив* — так выделяются новые термины, впервые введенные в книге;
- **полужирный шрифт** — так выделяются Web-адреса и адреса электронной почты;
- `моноширинный шрифт с засечками` — так выделяются имена файлов и папок, их содержимое, вывод команд, и, в общем случае, — все, что может содержаться в программах;
- **`полужирный моноширинный шрифт с засечками`** — так выделены команды и любой другой текст, который вводится с клавиатуры пользователем, а также фрагменты кода или содержимого файлов, которые обсуждаются в данный момент;
- ***`полужирный моноширинный шрифт курсивного начертания`*** — так выделяется текст, который должен быть заменен на информацию, предоставляемую пользователем.

Использование примеров кода

Основное предназначение этой книги — помочь вам выполнить вашу работу. В общем случае, вы можете применять код, приведенный в этой книге, в ваших программах и документации. За исключением тех случаев, когда вы копируете большие объемы кода, вам не обязательно обращаться в издательство O'Reilly за разрешением на копирование. Например, для использования небольших фрагментов приведенного в книге кода получения разрешения не требуется. С другой стороны, распространение примеров из книг O'Reilly на CD-ROM разрешения требует. Если вам нужно ответить на вопрос, и в вашем ответе вы цитируете данную книгу, то на это разрешения не требуется. Включение больших объемов текста книги в сопроводительную документацию к вашим программным продуктам разрешения требует.

Мы очень ценим, но не требуем в обязательном порядке указания полной информации об этой книге. Как правило, при ссылке на книгу указывается ее название, автор, издатель и ISBN, например: *"Cloud Application Architectures by George Reese"*. Copyright 2009 George Reese, 978-0-596-15636-7.

Если вы чувствуете, что объем цитирования выходит за рамки правомерного использования, обозначенного в этом разделе, вы можете обращаться за разрешением на использование материалов по следующему адресу: **permissions@oreilly.com**.

Safari® Books Online

Если вы видите на обложке вашей любимой книги, посвященной информационным технологиям, значок Safari® Books Online, это означает, что данная книга доступна через сервис O'Reilly Network Safari Bookshelf.

Safari предлагает даже лучшее решение, чем электронные книги (e-books). Safari® Books Online — это виртуальная библиотека, которая позволяет с легкостью искать нужную информацию по тысячам книг об информационных технологиях, копировать и вставлять в ваши тексты образцы кода, скачивать главы из книг и быстро находить ответы на интересующие вас вопросы, получая наиболее точную и актуальную информацию. Попробуйте бесплатно протестировать этот сервис по следующему адресу: **http://my.safaribooksonline.com**.

Мы ждем ваших отзывов!

Редакция O'Reilly проверила информацию, приведенную в этой книге, и протестировала образцы кода, но никто из нас не безупречен, поэтому ошибки и недосмотры все же случаются. Если вы обнаружите какие-нибудь ошибки или неточности, а также захотите поделиться с нами своими пожеланиями на будущее, пожалуйста, пишите нам:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (в США или Канаде)
707-829-0515 (международное или местное отделение)
707-829-0104 (факс)

У нас есть Web-страница, заведенная для этой книги, на которой мы публикуем списки опечаток, примеры и дополнительную информацию. Эта страница имеет следующий адрес:

http://www.oreilly.com/catalog/9780596156367

Если вы хотите задать технический вопрос или что-либо прокомментировать, пишите по следующему адресу:

bookquestions@oreilly.com

Дополнительную информацию о наших книгах, конференциях, программному обеспечению, центрах ресурсов (Resource Centers) и сети O'Reilly (O'Reilly Network) можно получить на нашем Web-сайте:

http://www.oreilly.com

Благодарности

В этой книге рассматривается такое количество научных дисциплин и такое множество технологий, что я просто не смог бы написать ее полностью самостоятельно.

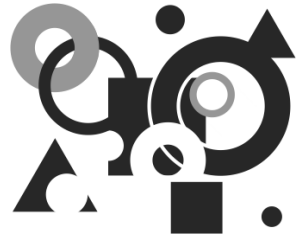
В первую очередь я хотел бы выразить безмерную благодарность за помощь, которую оказали мне Рэнди Байас (Randy Bias) из компании GoGrid и Эрик "E. J". Джонсон (Eric "E. J" Johnson) из компании Rackspace. Мой опыт работы с облачной инфраструктурой ограничивался работой с Web-сервисами Amazon (Amazon Web Services), поэтому Рэнди и Эрик потратили довольно долгое время на рецензирование и проверку фрагментов книги, посвященных AWS. Также они написали приложения, в которых рассказали о предложениях компаний GoGrid и Rackspace соответственно.

Далее, я крайне признателен всем, кто вычитывал главы книги и предоставлял подробные комментарии: Джону Оллспо (John Allspaw), Джеффу Барру (Jeff Barr), Кристоферу Хоффу (Christofer Hoff), Тео Шлосснейглу (Theo Schlossnagle) и Джеймсу Уркхарту (James Urquhart). Каждый из них обладает уникальным опытом и привнес в книгу ценные технические материалы. Благодаря их труду и критическим замечаниям книга стала намного полезнее, чем могла бы оказаться.

В дополнение к этому, еще целый ряд специалистов предоставил свои замечания по избранным частям этой книги, в том числе: Дэвид Бэгли (David Bagley), Морган Кэтлин (Morgan Catlin), Майк Хоруот (Mike Horwath), Моника Риз (Monique Reese), Стейси Релофс (Stacey Roelofs) и Джон Вьега (John Viega).

Наконец, я выражаю отдельную благодарность сотрудникам O'Reilly: Энди Орэму (Andy Oram) и Изабель Канкл (Isabel Kunkle). Я неоднократно говорил и писал это, и не только в данной книге, но я должен упомянуть это и здесь: благодаря их редактированию я стал гораздо лучшим писателем.

Глава 1



Облачные вычисления

Отличительным признаком любого "модного" слова, которое у всех на слуху, является то, что оно может передавать не реальное понимание его точного значения, а лишь видимость этого понимания. Термин "облако" (cloud) является как раз таким словом — многие люди просто произносят его, не понимая его истинного смысла.

Оно используется в самых разных, не согласующихся между собой контекстах, и часто им обозначают совершенно разные вещи и понятия. В одном из обсуждений люди могут говорить о Google Gmail; в другом — о Web-сервисе Amazon Elastic Compute Cloud (Amazon EC2) — но здесь, по крайней мере, термин "облако" все же присутствует в названии сервиса.

С другой стороны, термин "облачные вычисления" (cloud computing) является "модным словечком" ничуть не более, чем им является термин Web. Облачные вычисления представляют собой результат эволюции множества различных технологий, которые в комбинации изменили организационный подход к построению информационной инфраструктуры предприятия. Как и в случае с Web (а это произошло чуть более десятилетия тому назад), в облачных вычислениях нет абсолютно ничего фундаментально нового. Многие из технологий, вошедших в состав Web, существовали десятилетиями до появления компании Netscape Communications, которая объединила их и сделала универсально доступными. По аналогии с этим событием, большинство технологий, которые в совокупности и формируют облачные вычисления, существовали уже в течение десятков лет. Чтобы появился термин "облачные вычисления", потребовалось, чтобы компания Amazon объединила их и сделала их комбинацию массово доступной.

Цель данной книги заключается в том, чтобы дать возможность разработчикам Web-приложений на базе транзакций использовать облачную инфраструктуру в качестве средства для развертывания их приложений. Поэтому в книге основное внимание уделяется таким облакам, как Amazon EC2, и в меньшей степени — другим, таким как Google Gmail. Тем не менее, обсуждение мы начнем с описания общей инфраструктуры для облачных вычислений.

Облако

Облако — это не просто новейший модный термин, применяющийся для описания интернет-технологий удаленного хранения данных. Хотя Интернет представляет собой основной фундамент, необходимый для облака, облако — это нечто большее, чем просто Интернет. *Облако* — это то, куда вы обращаетесь, когда вам требуется использовать ту или иную технологию, и пользуетесь ею до тех пор, пока она вам нужна, и ни минутой дольше. Для этого вам не требуется ничего устанавливать на вашем компьютере и вы не платите за технологию, когда вы не пользуетесь ею.

Облако может означать как программное обеспечение, так и инфраструктуру. Оно может быть приложением, доступ к которому предоставляется через Web, или сервером, к которому вы обращаетесь в точности тогда, когда вам это требуется. Вне зависимости от того, является ли сервис программным или аппаратным, вам нужен критерий, позволяющий определить, является ли этот сервис облачным. Этот критерий формулируется так:

"Если для доступа к сервису вы можете зайти в любую библиотеку или интернет-кафе, сесть за любой компьютер, не предъявляя никаких особых требований ни к операционной системе, ни к браузеру, и получить доступ к сервису, то этот сервис является облачным".

Я сформулировал три условия, по которым в дальнейшем обсуждении мы будем определять, является ли тот или иной сервис облачным:

- сервис доступен через Web-браузер (непатентованный и не являющийся собственностью той или иной фирмы) или при помощи специального интерфейса прикладного программирования (API) для доступа к Web-сервисам;
- чтобы начать пользоваться сервисом не требуется никаких затрат капитала;
- вы платите только за то, чем вы пользуетесь, и оплачиваете только то время, в течение которого вы этим пользуетесь.

Я не считаю, что три перечисленных условия полностью закрывают вопрос, но они представляют собой твердое основание, позволяющее вести дальнейшее обсуждение, и полностью отражают мое понимание облачных сервисов, которые рассматриваются в данной книге.

Если вам не нравится мое сжатое и концентрированное определение облачных сервисов, вы можете обратиться к определению, которое Джеймс Гавернор (James Governor) дал в своем блоге: запись называется "15 Ways to Tell It's Not Cloud Computing", а найти ее можно по следующему адресу: **http://www.redmonk.com/jgovernor/2008/03/13/15-ways-to-tell-its-not-cloud-computing**.

Программное обеспечение

Как я уже упоминал ранее, облачные сервисы подразделяются на программные сервисы и сервисы инфраструктуры. С точки зрения зрелости технологии, про-

граммное обеспечение в облаке развито значительно лучше, чем аппаратная составляющая.

Программное обеспечение как услуга (Software as a Service, SaaS), в сущности, представляет собой термин, призванный обозначать программную составляющую в облаке. Большинство систем SaaS, хотя и не все, являются облачными системами.

SaaS представляет собой модель развертывания ПО на основе Web, благодаря которой программное обеспечение оказывается полностью доступным через Web-браузер. Для пользователей систем SaaS не имеет значения, где установлено программное обеспечение, какую операционную систему оно использует и на каком языке оно написано — PHP, Java или .NET. И, самое главное — вам самим нет необходимости устанавливать что-либо и где-либо.

Например, Gmail представляет собой не что иное, как программу электронной почты, доступную через браузер. Она предоставляет все те же функциональные возможности, что и Apple Mail или Outlook, но при этом для пользования ею не нужен так называемый "толстый" клиент[1]. Даже если ваш домен не получает почту через Gmail, вы все равно можете пользоваться Gmail для доступа к вашей почте.

SalesForce.com — это еще один вариант SaaS. SalesForce.com представляет собой систему управления отношениями с клиентами (CRM), предназначенную для крупного предприятия. Она предназначена для специалистов из отделов продаж и позволяет им отслеживать потенциальных заказчиков и менеджеров по закупкам, определять их место в организации процесса продаж, управлять всем процессом заключения сделки — от первого контакта до заключения договора и проведения расчета, и т. д. Как и в случае с Gmail, вам не требуется устанавливать никакое дополнительное программное обеспечение для получения доступа к SalesForce.com: достаточно только ввести адрес SalesForce.com в браузере, завести учетную запись, и можно приступать к работе.

Системы SaaS обладают некоторыми определяющими характеристиками, в том числе:

- *Доступность через Web-браузер.* Программное обеспечение типа SaaS никогда не требует установки каких-либо дополнительных программ на ваш ноутбук или настольный компьютер. Доступ к системам SaaS осуществляется через Web-браузер с использованием открытых стандартов или универсальный плагин браузера. Облачные вычисления и патентованное ПО, являющееся собственностью какой-либо компании, просто не сочетаются между собой.
- *Доступность по требованию.* Чтобы получить доступ к системе SaaS, вам не требуется полностью проходить процесс сделки в классическом понимании этого слова. Как только вы заведете учетную запись, вы сможете получать дос-

[1] "Толстый" клиент (fat client, thick client, иногда rich client) — в архитектуре "клиент-сервер" представляет собой приложение, предоставляющее (в противовес "тонкому" клиенту) расширенные функциональные возможности, вне зависимости от центрального сервера. При таком подходе сервер выполняет роль хранилища данных, а вся работа по обработке и представлению этих данных переносится на клиентский компьютер. — *Прим. перев.*

туп к нужному вам программному обеспечению в любой момент и из любой географической точки.

- *Условия оплаты зависят от использования.* SaaS не требует никаких вложений в инфраструктуру или какой-то особой настройки, поэтому вам не потребуется начинать работу с серьезных капиталовложений. Платить вы будете только за те виды сервиса, которыми вы пользуетесь, и только за фактическое время использования. Если сервис вам больше не нужен, вы просто прекращаете за него платить.
- *Минимальные требования к инфраструктуре IT.* Если вам не требуется покупать никаких серверов и нет необходимости строить сеть, то зачем вам вообще нужна какая-то инфраструктура IT? Хотя для конфигурирования систем SaaS и может иногда требоваться некоторый минимальный уровень технических знаний (например, для управления DNS в Google Apps), но этот уровень не выходит за рамки, характерные для обычного опытного пользователя. Высококвалифицированный IT-администратор для этого не требуется.

Одной из особенностей некоторых развернутых систем SaaS, которую я нарочно пока пропустил, является так называемая "многоарендная архитектура" (multitenancy)[1]. Многие поставщики SaaS хвастаются тем, что их системы предоставляют "многоарендные услуги", а некоторые даже утверждают, что "многоарендная" архитектура является основным требованием к системе SaaS.

Многоарендное приложение представляет собой программу-сервер, которая поддерживает развертывание множества клиентов, одновременно пользующихся одним и тем же экземпляром программы-сервера. Эта модель предлагает поставщику SaaS очевидные преимущества в том, что позволяет обеспечить для конечных пользователей следующие возможности:

- поддержку большего количества клиентов на меньшем количестве аппаратных компонентов;
- ускоренную и упрощенную процедуру развертывания обновлений для приложений и обновления для системы безопасности;
- в целом более надежную архитектуру.

[1] "Многоарендная архитектура" (multitenancy) — это специальный прием программирования или архитектурное решение, поддерживающее одновременное использование одного и того же экземпляра программы несколькими клиентами. В основе этого термина лежит слово "tenant", которое буквально означает "жилец" или "арендатор". Вообще-то при описании multitenancy сразу же возникает ассоциация с таким бытовым явлением, как "коммунальная квартира", и, наверное, можно было бы назвать такую архитектуру действительно "коммунальной". Однако это слово уже "занято" — оно уже использовано для перевода термина utility computing (принцип предоставления информационных сервисов как коммунальных услуг или "коммунальные вычисления"). Впрочем, в отличие от физической коммунальной квартиры, виртуальная облачная "квартира" имеет раздвижные перегородки, и ее конфигурацию можно менять по мере необходимости. Таким образом, термин multitenancy можно интерпретировать как технологическое решение, позволяющее нескольким пользователям независимо друг от друга разделять один и тот же ресурс, не нарушая при этом конфиденциальности и защиты принадлежащих им данных. — *Прим. перев.*

Максимальный выигрыш для конечного пользователя заключается в косвенном уменьшении платы за сервис, более быстром доступе к новым функциональным возможностям и (иногда) ускоренном исправлении слабых мест в системе безопасности. Однако за счет того, что базовым принципом облачных вычислений является незаинтересованность конечного пользователя в архитектуре приложения, лежащего в основе используемого сервиса, значимость многоарендной архитектуры в перспективе будет уменьшаться.

Как будет показано в следующем разделе, технологии виртуализации в принципе делают архитектурные преимущества многоарендности неактуальными.

Аппаратные средства

В общем случае, большинству людей концептуально гораздо сложнее понять и принять аппаратные средства в облаке, нежели программное обеспечение в облаке. Аппаратные средства — это нечто материальное, что вы можете потрогать: это ваша собственность, и вы не сдаете ее напрокат. Если ваш сервер пострадает в результате пожара — это бедствие касается только вас. Многим людям тяжело смириться с тем, что их аппаратные средства принадлежат не только им, и отказаться от возможности увидеть и физически потрогать свое собственное оборудование.

Если аппаратные средства находятся в облаке, вы запрашиваете новый "сервер" по мере необходимости. Обычно он будет вам доступен менее, чем через 10 минут. Когда вы заканчиваете с ним работать, вы освобождаете ресурс, и он возвращается в облако. При этом вы не имеете ни малейшего представления, какой физический сервер предоставляет вам облачный сервис, и, скорее всего, вы не знаете даже его конкретного физического местоположения.

Барьер старых ожиданий

Для меня, как для поставщика облачных сервисов, труднее всего ответить на вопрос: "Так где же находятся наши серверы?. Фактически честный ответ на этот вопрос будет таким: "Я не знаю — может быть, где-нибудь на восточном побережье США или где-нибудь в Западной Европе", но многие клиенты, получив такой ответ, почувствуют себя крайне неуютно. Однако это отсутствие информации о физическом местоположении ваших серверов позволяет почувствовать интересное ощущение физической безопасности, поскольку для мотивированных злоумышленников это делает почти невозможной физическую атаку на ваше оборудование с целью его компрометации.

Преимущества облачной инфраструктуры

Задумайтесь о таких факторах, о которых вы должны заботиться, если вы имеете аппаратные средства в собственности и должны обеспечивать их работу.

- Мощность серверов стала недостаточной?

  Планирование необходимой мощности и обеспечение ресурсами всегда играет важную роль. Облачные вычисления упрощают для вас решение следующих двух

проблем, которые вам необходимо решать в том случае, если аппаратные средства находятся в вашей собственности: что делать, если вы неправильно оценили возможности вашего оборудования (чересчур оптимистично или слишком пессимистично)? Что делать, если у вас нет финансовых средств для расширения бизнеса на тот момент, когда встает проблема покупки нового оборудования? Если вы управляете собственной инфраструктурой, вам потребуется собрать большие суммы наличными на каждую новую сеть устройств хранения данных (Storage Area Network, SAN) или каждый приобретаемый сервер. Кроме того, вам потребуется довольно значительное время на ввод в эксплуатацию — от момента принятия решения до размещения заказа, его оплаты, физической доставки, приемки, монтажа, инсталляции ПО, тестирования и, наконец, ввода в эксплуатацию.

- Что произойдет в случае возникновения проблемы?

  Естественно, любой высококачественный сервер имеет некоторый резерв, позволяющий безболезненно пережить некоторые типичные аппаратные проблемы. Но, даже если у вас есть, например, резервный жесткий диск, предназначенный на замену отказавшего диска в составе массива RAID, все равно кто-то должен заменить отказавший диск, управлять процедурой RMA[1], а затем установить новый диск на сервер. Для этого требуется не только время, но и высокая квалификация, и при этом вам необходимо провести все эти работы в сжатые сроки, чтобы избежать полного выхода сервера из строя.

- Что произойдет в случае чрезвычайной ситуации?

  Если сервер окончательно выйдет из строя, то, если только вы не имеете инфраструктуры с высокой степенью отказоустойчивости, вы столкнетесь с бедствием, и все ваши сотрудники должны будут действовать в условиях форс-мажорных обстоятельств, предпринимая все усилия для устранения последствий возникшей аварийной ситуации. В таких случаях вам остается только надеяться на то, что у вас есть качественная и актуальная резервная копия, а ваш план аварийного восстановления проработан достаточно хорошо, чтобы провести восстановление в кратчайшие возможные сроки. Восстановление — это почти всегда ручной процесс.

- Что делать, если сервер вам больше не нужен?

  Вполне возможно, что ваши потребности в мощности используемого оборудования со временем изменятся. Кроме того, вполне возможно, что вам необходимо будет вывести из эксплуатации устаревший и обесцененный сервер. Что делать с этим старым сервером? Даже если вы хотите его полностью списать, все равно кто-то должен будет этим заниматься. Если же сервер обесценился не полностью, ваша компания понесет убытки из-за машины, которая совершенно бесполезна для вашего бизнеса.

[1] RMA (Return Merchandise Authorization) — процедура возврата неисправного или некачественного изделия поставщику или производителю. Когда вам нужно заменить дефектный компонент, вы обычно должны выполнить определенную последовательность действий по взаимоотношениям с поставщиком (составить заявку на возврат и замену, переслать поставщику, затем провести переговоры о ремонте/замене и, наконец, получить в качестве замены исправное изделие).

- Как решить вопросы с недвижимостью и электроснабжением?

  Если вы управляете собственной инфраструктурой (или даже имеете собственную стойку у вашего интернет-провайдера), вам может потребоваться оплачивать недвижимость или электроэнергию, не используя при этом большую часть оплачиваемых ресурсов. Это мало того, что неэкологично, но и представляет собой абсолютно непродуктивную трату денег.

При использовании облачной инфраструктуры ни один из следующих аспектов не представляет проблемы, потому что:

- вы добавляете мощностей в облачную инфраструктуру только на тот момент, когда она вам нужна, и ни минутой раньше. Вы не несете никаких затрат, ассоциированных с выделением ресурсов, поэтому вам нет необходимости беспокоиться о синхронизации потребностей в мощностях и бюджетными нуждами. Наконец, вы можете нарастить мощность за считанные минуты, и это позволит вам не потерять лицо даже в сложных ситуациях;
- вам вообще нет необходимости беспокоиться об аппаратных средствах, на которых все это работает. Вы можете никогда не узнать о том, что физический сервер, на котором фактически выполнялась ваша работа, полностью отказал. Если вы правильно подберете инструментарий, вы можете добиться автоматического восстановления после сложнейших аварийных ситуаций, а ваша команда в это время может попросту спать;
- если ваши потребности в мощностях изменяются, вам может потребоваться другая виртуальная аппаратная конфигурация. В этом случае вы просто отказываетесь от вашего виртуального сервера и получаете другой. При этом вам нет необходимости задумываться о перепродаже, утилизации или беспокоиться о вреде для окружающей среды;
- вам нет необходимости платить за недвижимость и электроэнергию, которыми вы не пользуетесь. Поскольку вы используете лишь часть гораздо больших аппаратных мощностей, нежели вам нужно, вы повышаете эффективность использования физического пространства, необходимого для обеспечения ваших потребностей в обработке информации. Более того, вы не платите за всю стойку серверов, потребляющих электроэнергию, где большинство циклов центрального процессора являются холостыми.

Аппаратная виртуализация

Аппаратная виртуализация (hardware virtualization) представляет собой технологию, которая позволяет большинству поставщиков услуг облачных вычислений, в том числе и Amazon Web Services (AWS), предлагать свои услуги[1]. Если у вас

[1] Существуют и другие подходы к реализации облачной инфраструктуры, включая предоставление аппаратных средств по требованию через такие компании, как AppNexus и NewClouds. Кроме того, некоторые поставщики услуг, скажем, такие, как GoGrid (*см. приложение 2*), предлагают гибридные решения.

есть компьютер Mac, на котором вы запускаете Windows или Linux в таких эмуляторах, как Parallels или Fusion, то вы используете технологию виртуализации, аналогичную той, что применяется для реализации облачных вычислений. Благодаря виртуализации администратор IT может подразделить физический сервер на любое количество виртуальных серверов, каждый из которых работает под управлением собственной операционной системы и каждому из которых выделяются такие ресурсы, как память, CPU, участки дискового пространства. Некоторые технологии виртуализации даже позволяют вам перемещать работающие экземпляры виртуальных серверов с одного физического сервера на другой. С точки зрения пользователя или приложения, которые работают на виртуальном сервере, не существует никаких возможностей определить, является ли сервер, на котором они работают, виртуальным или физическим.

Ряд виртуализационных технологий, доступных на рынке, используют различные подходы к проблеме виртуализации. Решение Amazon представляет собой расширение популярной системы виртуализации на основе открытого кода, которая называется Xen[1]. Xen предоставляет компонент, называемый *гипервизором* (hypervisor), на котором могут работать одна или несколько гостевых операционных систем. Гипервизор создает уровень аппаратных абстракций, который позволяет операционным системам совместно использовать ресурсы физического сервера, при этом гостевые операционные системы не имеют физического доступа к этим ресурсам, как "своим", так и "чужим".

Общеизвестное возражение против виртуализации — особенно со стороны тех, кто работал с эмуляторами на настольных компьютерах под управлением настольных операционных систем — заключается в том, что виртуализированные системы существенно теряют в производительности. Но в облаке это общепринятое предубеждение против виртуализации не имеет под собой оснований по следующим причинам:

- даже пониженная производительность оборудования вашего поставщика облачных услуг, по всей вероятности, окажется лучше, чем максимальная производительность вашего потребительского сервера, и это при том условии, что он оптимальным образом настроен;
- технологии виртуализации, предназначенные для крупных предприятий, например, такие как Xen и VMware (**http://www.vmware.com/**), используют так называемую *паравиртуализацию* (paravirtualization), а также возможности *аппаратной виртуализации* (hardware-assisted virtualization capabilities), разработанные различными производителями процессоров. Это позволяет им добиться почти такой же производительности, как и при работе приложений на "живом железе".

[1] Официальный сайт проекта: **http://www.xen.org/**, подробные описания см. по адресам: **http://wiki.xensource.com/xenwiki/Xen_on_4_app_servers**, **http://ru.wikipedia.org/wiki/Xen**, **http://www.vmgu.ru/citrix-xen**. — *Прим. перев.*

Облачное хранилище

Абстрагирование от аппаратных средств в облаке осуществляется не только благодаря замене физических серверов виртуальными. Виртуализации подлежат и системы физического хранения данных.

Облачное хранилище позволяет вам перебрасывать данные в облако, и при этом не беспокоиться о том, как именно они хранятся, и не задумываться об их резервном копировании. Когда данные, перемещенные в облако, понадобятся вам снова, вам достаточно будет просто обратиться в облако и получить свои данные. При этом вы можете не знать, как хранятся эти данные, где они хранятся, или того, что происходит с тем или иным оборудованием, когда вы периодически перемещаете данные в облако и извлекаете их оттуда.

Как и в случае с другими элементами облачных вычислений, на рынке предлагается несколько подходов к облачному хранилищу. Говоря общими словами, они связаны с разбиением ваших данных на небольшие цепочки и хранение их на множестве серверов. Цепочки данных снабжаются индивидуально вычисленными контрольными суммами, так, чтобы данные можно было быстро восстановить, вне зависимости от того, что могло бы произойти в течение времени хранения с накопителями, физически хранящими данные, и скомпрометировать облако.

Я не раз был свидетелем того, как люди, начинающие работать с облаком, пытались обращаться с облачным хранилищем так, как если бы оно представляло собой сетевой накопитель. С точки зрения принципа работы облачное хранилище принципиально отличается от традиционных сетевых накопителей, и служит принципиально другим целям. Облачное хранилище имеет тенденцию работать гораздо медленнее и имеет гораздо бóльшую степень структурированности, вследствие чего его использование в качестве оперативного хранилища данных непрактично, вне зависимости от того, работает ли приложение, использующее эти данные, в облаке или где-то еще.

Вообще говоря, облачное хранилище не подходит для оперативного использования облачными приложениями на базе транзакций. В последующих главах мы подробнее обсудим роль облачного хранилища в инфраструктуре приложений, работающих на базе транзакций. Пока же об облачном хранилище можно думать примерно так же, как об аналоге резервной копии на ленточном носителе, но, в отличие от системы резервного копирования с ленточным приводом, при работе в облаке вам не нужны ни привод, ни ленты.

Примечание

Компания Amazon недавно выступила с новым коммерческим предложением, которое называется Amazon CloudFront. Amazon CloudFront использует Amazon S3 в качестве средства для реализации сети, предназначенной для распределения информационного содержимого. Идея, на которой основывается Amazon CloudFront, заключается в репликации вашего информационного содержимого по периферии сети. Хотя облачное хранилище Amazon S3 не подходит для использования с Web-приложениями на базе транзакций в оперативном режиме, в будущем оно обещает стать ключевым компонентом систем быстрого распределения статического контента.

Архитектуры облачных приложений

Мы могли бы еще долго обсуждать технологии виртуализации (кстати, знаете ли вы, что вы можете комбинировать и совместно использовать не менее пяти различных типов виртуализации?), но центральной темой этой книги являются не эти вопросы, а написание приложений, которые наилучшим образом используют преимущества облачных вычислений. Именно к этой теме мы сейчас и перейдем.

Концепция Grid Computing

Концепция Grid Computing[1] представляет собой архитектуру приложений, предоставляющую самый простой метод перехода к облачной архитектуре. Приложения, использующие *грид-технологии*, представляют собой ПО, интенсивно потребляющее ресурсы процессора. *Грид-приложения* разбивают операции по обработке данных на небольшие последовательности элементарных операций, которые могут выполняться изолированно.

В общем, если вы когда-либо принимали участие в проекте **SETI@home**, то вы уже принимали участие в грид-вычислениях. Проект по поиску внеземных цивилизаций SETI (Search for Extra-Terrestrial Intelligence) использует радиотелескопы, ведущие постоянное наблюдение за активностью в космосе. Эти радиотелескопы собирают гигантские объемы данных, которые затем нуждаются в обработке с целью поиска сигналов, отличающихся от естественных — обнаружение сигнала, который может иметь искусственное происхождение, может служить признаком деятельности внеземной цивилизации. Обработка такого объема информации на одном компьютере потребует такого длительного времени, что вполне может статься, человечество к завершению этих вычислений действительно уже освоит межзвездные перелеты. Но вот множество компьютеров, каждый из которых использует для решения этой проблемы свои холостые циклы CPU, могут справиться с задачей намного быстрее.

Эти компьютеры, на которых работает приложение **SETI@home** (возможно, включая и ваш собственный компьютер), формируют сеть или "решетку" (grid). Каждый раз, когда у них есть свободные такты процессора, компьютеры направляют запросы серверам проекта SETI на получение наборов данных для обработки. Они обрабатывают наборы данных и передают результаты обратно серверам проекта SETI. Результаты, полученные одним из участников проекта, перепроверяются повторной обработкой на компьютерах других участников, а затем производится дальнейшая проверка наиболее интересных результатов [1].

---

[1] В русскоязычных ресурсах встречается и такое написание, как "грид" (англ. *grid* — решетка, сеть) — согласованная, открытая и стандартизованная компьютерная среда, которая обеспечивает гибкое, безопасное, скоординированное разделение вычислительных ресурсов и ресурсов хранения информации, которые являются частью этой среды, в рамках одной виртуальной организации. См., например**: http://tinyurl.com/37mx4jn**, **http://gridclub.ru/**. — *Прим. перев.*

В 1999 г. участники программы SETI предложили использовать холостые циклы настольных компьютеров обычных пользователей для обработки данных проекта. Коммерческие и правительственные организации сформировали сеть из суперкомпьютеров для выполнения той же задачи. Впоследствии были созданы пулы серверов (так называемые "серверные фермы", server farms) для формирования сети, выполняющей такие задачи, как рендеринг видео (визуализация изображений и обсчет последовательностей кадров). Как суперкомпьютеры, так и серверные фермы оказались очень дорогими подходами к грид-вычислениям и требовали крупных капиталовложений.

Облачная инфраструктура серьезно упрощает и удешевляет создание грид-приложений. Если у вас есть данные, нуждающиеся в обработке, вы просто запрашиваете сервер для обработки этих данных. После завершения обработки этот сервер может быть остановлен или может запросить для обработки новую порцию данных.

На рис. 1.1 проиллюстрировано, как работает грид-приложение. Сначала сервер или кластер серверов получает набор данных, которые нуждаются в обработке. На первом шаге эти данные передаются в очередь сообщений (1). Другие узлы, которые часто называются серверам обработки (или, как в случае **SETI@home** — другими настольными компьютерами), наблюдают за очередью сообщений (2) и ждут появления новых наборов данных. Когда набор данных появляется в очереди сообщений, он обрабатывается первым же компьютером, который его обнаружил, и результаты отсылаются обратно в очередь сообщений (3), откуда они считываются сервером или кластером серверов (4). Оба компонента могут работать независимо друг от друга, и каждый из них может работать даже в том случае, если второй компонент не работает ни на одном компьютере.

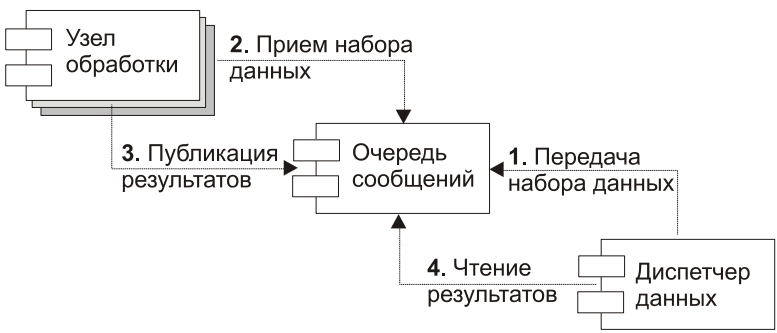


Рис. 1.1. Архитектура грид-приложений отделяет основное приложение от узлов, осуществляющих обработку данных

Здесь приходят на помощь облачные вычисления, потому что при их использовании вы не должны иметь никаких серверов в личной собственности, и когда у вас нет данных для обработки, вам не требуется никаких серверов вообще. Таким образом, вы можете масштабировать выделяемые вам вычислительные мощности,

запрашивая такие количества серверов, которые необходимы для переработки наборов данных, передаваемых вашему приложению. Иными словами, вместо того, чтобы заставлять компьютеры, работающие вхолостую, заниматься обработкой данных по мере их поступления, вы можете сделать так, чтобы серверы включались, когда поток данных становится более интенсивным, и выключались по мере того, как интенсивность потока данных ослабевает.

Поскольку на сегодняшний день грид-приложения имеют очень ограниченную область применения (переработка больших объемов научных и финансовых данных), мы не будем рассматривать их в этой книге слишком подробно. Тем не менее, многие из принципов, излагаемых в этой книге, применимы и к Grid.

Транзакционные вычисления

Большая часть бизнес-приложений использует транзакционные вычисления, которые находятся в центре внимания данной книги. *Транзакционная система* — это система, в которой один или большее количество поступающих наборов данных обрабатываются совместно в рамках единой транзакции и устанавливают взаимосвязи с другими данными, уже введенными в систему. Основой транзакционной системы обычно является реляционная база данных, которая управляет взаимоотношениями между всеми данными, образующими систему.

На рис. 1.2 показана логическая структура транзакционной системы высокой отказоустойчивости. При использовании архитектуры этого типа сервер приложений обычно обрабатывает данные, хранящиеся в базе данных, и представляет их через Web-интерфейс. За счет этого пользователь может работать с данными. Большинство Web-сайтов и Web-приложений, которые вы используете в своей повседневной работе, представляют собой ту или иную форму транзакционной системы. Для обеспечения высокой отказоустойчивости все эти компоненты могут формировать кластеры, а бизнес-логика и логика представления данных могут прятаться за балансировщиком нагрузки[1].

Развертывание транзакционной системы в облаке несколько сложнее и не настолько самоочевидно, как развертывание грид-системы. В то время как в грид-системе узлы должны быть короткоживущими, в транзакционной системе они должны быть долгоживущими.

Ключевой проблемой для любой системы, нуждающейся в долгоживущих узлах в облачной инфраструктуре, является тот основополагающий факт, что среднее время наработки на отказ (MTBF) для виртуального сервера обязательно меньше,

[1] Балансировщик нагрузки (load balancer) — это программный или аппаратный компонент сети, который распределяет процесс выполнения заданий между несколькими серверами сети с тем, чтобы оптимизировать использование ресурсов и сократить время вычисления.
См., например: **http://citforum.ru/internet/webservers/websbal.shtml**, **http://www.cisco.com/en/US/tech/tk365/technologies_tech_note09186a0080094820.shtml**. — *Прим. перев.*

чем для поддерживающих его аппаратных средств. В предельно упрощенной формулировке эта проблема выражается так: если у вас есть два физических сервера, среднее время наработки на отказ для которых составляет три года, то вероятность сбоя во всей системе у вас будет меньше, чем если бы вы имели только один физический сервер, на котором работают два виртуальных узла. Фактически количество физических узлов управляет показателем MTBF, а так как физических узлов меньше, в вашей облачной транзакционной системе для физических узлов MTBF будет выше, чем для виртуальных.

Рис. 1.2. В транзакционных приложениях функции подразделяются на представление данных, реализацию бизнес-логики и хранилище данных

Однако облачные вычисления предлагают ряд подходов, которые помогают не только сгладить эту проблему, но и потенциально увеличить общий показатель MTBF для транзакционной системы в целом. Далее в этой книге мы рассмотрим некоторые приемы, которые позволят добиться уровня готовности, более высокого, чем можно добиться в рамках вашего бюджета, и при этом поддерживать транзакционную целостность вашего облачного приложения.

Значимость облачных вычислений

Как далеко вы можете зайти в этой области?

Если вы можете развернуть все ваше индивидуально разработанное ПО в облачной инфраструктуре и воспользоваться системами SaaS в качестве замены традиционному потребительскому ПО, вы можете добиться полностью облачной инфраструктуры IT. В табл. 1.1 перечислены все типичные программные компоненты, которые обычно применяются в малых и средних предприятиях, и их эквиваленты в облачной инфраструктуре.

Таблица 1.1. *Традиционные компоненты типичной инфраструктуры IT малых и средних предприятий и их эквиваленты в облачной инфраструктуре*

Традиционная инфраструктура	Облачная инфраструктура
Файл-сервер	Google Docs
MS Outlook, Apple Mail	Gmail, Yahoo!, MSN
SAP CRM[1]/Oracle Siebel CRM[2]	SalesForce.com
Quicken[3]/Oracle Financials[4]	Intacct/NetSuite
Microsoft Office/Lotus Notes	Google Apps
Stellent[5]	Valtira
Дистанционное резервное копирование	Amazon S3
Сервер, стойки и брандмауэр	Amazon EC2, GoGrid, Mosso

Потенциальное влияние облачной инфраструктуры очень значительно. Для некоторых организаций — особенно малых и средних предприятий — это позволяет никогда больше не приобретать ни серверные платформы, ни какие бы то ни было лицензии на программное обеспечение. Иначе говоря, следующие вопросы либо сильно теряют свою значимость, либо вообще больше никогда не возникают.

- Насколько актуальны все наши лицензии на ПО? Системы SaaS и программное обеспечение с системой лицензирования, ориентированной на облачные вы-

[1] Приложение для управления взаимоотношениями с клиентами от компании SAP AG — это решение, которое позволяет объединять сотрудников, партнеров, процессы и технологии в рамках полного замкнутого цикла взаимодействия с клиентами. Оно предоставляет средства для выполнения проверки доступности ресурсов в режиме реального времени, управления договорами и т. д. С его помощью достигается высокая степень прозрачности выполнения заказов и отслеживания их статуса. SAP CRM предлагает автоматизированные бизнес-процессы для таких областей, как маркетинг, продажи, сервис, аналитика, центр взаимодействия, мобильные приложения, электронная коммерция, управление каналами продаж. См. **http://www.sap.com/germany/index.epx**, **http://www.sap.com/cis/index.epx**, **http://www.sap.com/cis/solutions/business-suite/crm/index.epx**. — *Прим. перев.*

[2] Дополнительную информацию по решению Oracle Siebel CRM см. по адресу: **http://www.oracle.com/global/ru/applications/siebel/index.html** — *Прим. перев.*

[3] Quicken — это программа для управления финансами, разработанная компанией Intuit, Inc. (**http://en.wikipedia.org/wiki/Intuit,_Inc**) — *Прим. перев.*

[4] См. **http://www.oracle.com/us/products/applications/ebusiness/index.htm**, **http://www.oracle.com/global/ru/applications/ebs/finance.html**. — *Прим. перев.*

[5] Stellent IBPM (Imaging and Business Process Management) — это информационная система, служащая для управления электронным архивом документов, управления электронным оборотом документов и управлением актами, согласно юридическим требованиям. IBPM — это решение, созданное для средних и больших организаций. См. **http://www.oracle.com/stellent/index.html**, **http://www.acsys.com.pl/index_ru.php?action=IBPM**, **http://www.ibs.ru/content/rus/614/6143-article.asp**. — *Прим. перев.*

числения, обойдутся вам ровно в ту сумму, которую стоит фактическое их использование, и рассчитаться за это можно кредитной картой.

- Когда следует спланировать следующее обновление ПО? Поставщики SaaS выполняют для вас все необходимые обновления, причем часто вы можете даже не знать, какую версию вы используете.
- Что необходимо делать, если один из аппаратных компонентов отказывает в 3 часа ночи? Средства управления облачной инфраструктурой могут автоматизировать операцию аварийного восстановления даже для самых тяжелых случаев.
- Как управлять нашими технологическими активами? Когда вы работаете в облачной инфраструктуре, то технологических активов (например, компьютеров, принтеров), которыми необходимо управлять, у вас гораздо меньше.
- Что делать со старым оборудованием? Оборудование не находится в вашей собственности, поэтому вам нет необходимости заботиться о его утилизации.
- Как управлять амортизацией моих активов IT? Цены, которые вы платите, вы платите только за время фактического использования, поэтому вы не несете никаких затрат на амортизацию.
- Когда я смогут нарастить мощность своей информационной инфраструктуры? Если вы работаете в облаке, то наращивать мощность можно пошагово и по мере необходимости.

Поставщики SaaS (я рассматриваю SaaS как часть облачной инфраструктуры) могут запускать все свои сервисы на оборудовании, поставленном в облако другим поставщиком, и при этом предлагать своим клиентам надежную облачную инфраструктуру без необходимости иметь в собственности аппаратные средства. Фактически мой собственный бизнес именно так и работает.

Возможности выбора для инфраструктуры IT

Облачные вычисления конкурируют с двумя подходами к инфраструктуре IT:

- внутренняя инфраструктура IT и собственная поддержка;
- аутсорсинг (передача функций сервисам, управляемым сторонними организациями).

Если оборудование находится у вас в собственности — это значит, что вы имеете инфраструктуру IT с внутренним управлением, даже если оборудование смонтировано в стойку в центре управления данными, принадлежащем другому владельцу. Потенциальным ключевым преимуществом облачных вычислений (с финансовой точки зрения) по сравнению с внутренней инфраструктурой IT с собственным управлением является отсутствие необходимости капиталовложений, необходимых для ее использования.

Внутренняя инфраструктура IT и собственная поддержка — это тот случай, когда оборудование находится в вашей собственности, и вы должны платить людям, которые его обслуживают, неважно, являются ли они штатными или внештатными сотрудниками вашего предприятия. Когда один из серверов отказывает, вы несете

расходы по ремонту и замене оборудования, за исключением расходов на холодный резерв, находящийся в вашей собственности.

Передача функций сервисам, управляемым сторонними организациями, подразумевает, что вы перечисляете фиксированную плату тому, кто является владельцем серверов, на которых работает ваше ПО, за аренду оборудования и его обслуживание. Если сервер отказывает, то именно компания, предоставляющая сервис по управлению, должна заботиться о его немедленной замене (или предоставлению услуг, предусмотренных вашим контрактом об уровне предоставляемого сервиса). Эта же компания заботится о том, чтобы на серверах были установлены необходимые операционные системы со всеми требуемыми обновлениями, и управляет сетевой инфраструктурой, в которой работают эти серверы.

В табл. 1.2 проведено сравнение характеристик внутренней структуры IT, управляемых сервисов и облачной инфраструктуры IT с учетом различных аспектов разработки инфраструктуры IT.

Таблица 1.2. *Сравнение характеристик инфраструктуры IT различных типов*

<table>
<tr><th></th><th>Внутренняя инфраструктура IT</th><th>Управляемые сервисы</th><th>Облачная инфраструктура</th></tr>
<tr><td rowspan="2">Капиталовложения</td><td>Высокие</td><td>Умеренные</td><td>Пренебрежимо малые</td></tr>
<tr><td colspan="3">Какие капиталовложения вам потребуются, чтобы развернуть вашу инфраструктуру IT "с нуля" или внести в нее изменения?
При внутренней инфраструктуре IT вы должны заплатить за оборудование прежде, чем сможете его использовать (при этом источник финансирования роли не играет)[1]. При передаче функций сторонней компании вам обычно требуется внести начальную плату (более умеренную, нежели в первом случае). При использовании облачной инфраструктуры вам обычно не требуется нести никаких начальных затрат и никаких обязательных платежей</td></tr>
<tr><td rowspan="2">Текущие эксплуатационные издержки</td><td>Умеренные</td><td>Высокие</td><td>Зависят от интенсивности пользования сервисом</td></tr>
<tr><td colspan="3">Ваши текущие эксплуатационные затраты на внутреннюю инфраструктуру IT складываются из затрат на заработную плату обслуживающего персонала и/или субподрядчиков, которые управляют работой оборудования, а также затрат на площади, предоставляемые вам хостинг-провайдером, и затрат на недвижимость, платы за энергию и коммунальные услуги.</td></tr>
</table>

[1] С финансовой точки зрения разница между оплатой из собственных средств или получением кредита в банке не имеет значения. В любом случае, если вы берете в банке кредит, вы выплачиваете по нему проценты. Если же вы снимаете деньги со счета в банке, вы теряете потенциальные суммы, которые могли бы быть вами получены в качестве процентов, и лишаетесь возможности потратить деньги на что-то другое (стоимость капитала).

Таблица 1.2 *(продолжение)*

	Внутренняя инфраструктура IT	**Управляемые сервисы**	**Облачная инфраструктура**
	Текущие эксплуатационные расходы могут сильно варьироваться в зависимости от условий вашего контракта и в случаях возникновения чрезвычайных ситуаций. С другой стороны, услуги за пользование сторонним сервисом тоже могут быть достаточно дорогими, но в этом случае вы обычно знаете, какую сумму вам требуется выплачивать ежемесячно, и эта сумма меняется редко. Что касается облачных вычислений, они могут быть как дешевыми, так и дорогими — здесь все зависит от ваших потребностей. Ключевым преимуществом облачной инфраструктуры является то, что вы платите только за фактически потребленный сервис, и ни за что другое. Затраты на заработную плату персонала в этом случае выше, чем при пользовании услугами стороннего сервис-провайдера, но ниже, чем в случае, когда вы поддерживаете собственную инфраструктуру IT		
Время ввода в эксплуатацию	Высокое	Умеренное	Мгновенная доступность
	Какое время вам потребуется на включение нового компонента в вашу инфраструктуру? При использовании внутренней инфраструктуры IT и модели управляемых сервисов вам необходимо заранее спланировать эту работу, разместить заказ, дождаться доставки компонента и включить его в центр обработки данных. В случае использования услуг стороннего сервис-провайдера время ввода в эксплуатацию обычно существенно короче, потому что такие компании обычно делают массовые закупки оборудования заранее. При использовании облачной инфраструктуры вы можете запустить новый "сервер" за считанные минуты, после того как вы решите, что он вам действительно нужен		
Гибкость	Низкая	Умеренная	Высокая
	Насколько легко может ваша инфраструктура адаптироваться к неожиданным пиковым нагрузкам при повышении потребностей в ресурсах? Например, есть ли у вас ограничения по дисковому пространству? Что происходит, если вы неожиданно приближаетесь к этому пределу? Внутренняя структура IT имеет очень жесткие ограничения и может быть приведена в соответствие с возросшей потребностью в ресурсах только за счет дальнейших капиталовложений. Сервис-провайдер, предоставляющий вам услуги, обычно может предложить временное повышение дисковой емкости за счет расширения полосы пропускания и предоставления вам временного доступа к альтернативным носителям и т. д. Однако облачная инфраструктура может настраиваться так, чтобы предоставлять вам в пользование ресурсы по мере необходимости и вы можете их освобождать, когда потребность в этих ресурсах отпадет		

Таблица 1.2 *(окончание)*

	Внутренняя инфраструктура IT	**Управляемые сервисы**	**Облачная инфраструктура**
Требования к квалификации персонала	Высокие	Низкие	Умеренные
	Каковы должны быть требования к квалификации персонала, занимающегося поддержкой вашей информационной среды? При внутренней инфраструктуре IT вам, очевидно, нужны штатные или внештатные сотрудники, которые досконально знают всю вашу инфраструктуру, могут поддерживать ваши компьютеры, осуществлять замену аппаратных устройств, поддерживать в актуальном состоянии операционные системы и устанавливать на них последние обновления. С другой стороны, использование услуг сервис-провайдера дает вам то преимущество, что ваш персонал может быть почти полностью невежественным в области IT — вы платите сервис-провайдеру, и он решает за вас ваши проблемы. Наконец, облачная инфраструктура предъявляет требования, которые могут варьироваться, в зависимости от того, как и чем вы пользуетесь. Часто можно найти такого менеджера облачной инфраструктуры (например, enStratus или RightScale), который будет управлять вашей средой, но при этом вам все равно необходимы навыки, достаточные для того, чтобы настраивать образы ваших машин		
Надежность	Может варьироваться в зависимости от обстоятельств	Высокая	От умеренной до высокой
	Насколько уверенными вы можете быть в том, что ваши сервисы доступны клиентам постоянно и круглосуточно? Возможность создать информационную среду с высокой отказоустойчивостью зависит от квалификации вашего персонала и от того, какие капиталовложения вы готовы сделать в вашу инфраструктуру IT. Прибегнуть к услугам сервис-провайдера в данном случае безопаснее, но этому варианту не хватает избыточности облачной инфраструктуры. В облачной инфраструктуре есть достаточно избыточных компонентов, но ей недостает стабильности и уровня обслуживания, обеспечиваемых сервис-провайдером		

Наиболее очевидный вывод, который можно сделать на основе проведенного сравнительного анализа, состоит в том, что построение собственной инфраструктуры IT "с нуля" больше не имеет смысла. Выиграть на этом могут только те предприятия и организации, которые к данному моменту уже сделали существенные капиталовложения в собственную инфраструктуру IT или те, чьи внутренние требования к безопасности и конфиденциальности запрещают хранить данные "на стороне".

Всем остальным можно порекомендовать либо использование услуг сервис-провайдеров, либо переход к использованию облачной инфраструктуры.

Экономические соображения

Возможно, наибольший выигрыш от использования облачной инфраструктуры вместо поддержания собственной инфраструктуры IT — даже не технологический, а финансовый. Модель оплаты только фактически потребленных услуг, принятая в облачной инфраструктуре, делает этот подход намного более дешевым, чем модель "предоплата за все", характерная для внутренней инфраструктуры IT.

Капитальные затраты

Основной финансовой проблемой, связанной с поддержанием внутренней инфраструктуры IT, являются капитальные затраты. *Капитальные затраты* — это те денежные средства, которые вы платите за ваши активы перед тем, как ввести их в эксплуатацию. Если вы покупаете сервер, то это приобретение представляет собой капитальные затраты, потому что вы вносите за него предоплату, и только потом начинаете его использовать и получать от него прибыль (как правило, он должен окупиться за 2—3 года).

Рассмотрим этот случай на примере компьютера, стоимость которого составляет $5000, плюс еще $2000 на настройку и конфигурирование. В этом случае $5000 представляют собой капитальные затраты, а $2000 — текущие эксплуатационные расходы. С точки зрения бухгалтерского учета, $5000 представляют собой перенос денег с одного счета на другой (например, сумма $5000 снимается с вашего банковского счета и переносится на ваш счет основных фондов предприятия. С другой стороны, сумма $2000 представляет собой реальные затраты, которые ведут к отсрочиванию получения прибыли.

Сервер представляет собой то, что с точки зрения бухгалтерского учета представляет собой так называемый "амортизируемый актив" или изнашиваемое имущество. По мере того, как он эксплуатируется, стоимость сервера падает в соответствии с тем, как долго он находится в эксплуатации. Иными словами, стоимость сервера для компании убывает с каждым месяцем, в течение которого он находится в эксплуатации, до тех пор, пока он не окажется полностью изношенным и будет подлежать замене. Каждое уменьшение стоимости сервера считается затратой, которая отсрочивает получение компанией прибыли.

Финансисты не любят капитальные затраты по целому ряду причин. На самом деле, они не любят любые затраты, которые не имеют отношения к текущей работе компании. Основной причиной этой нелюбви является то, что деньги затрачиваются сегодня, а выгода, которую вы от этого ожидаете получить, может быть получена в отдаленном будущем (с технической точки зрения после амортизации сервера). Любой владелец предприятия или наемный управляющий хочет сконцентрировать финансовые средства компании на таких вещах, которые окупаются быстро. Этот вопрос особенно актуален для малых и средних предприятий, которые могут переживать нелегкие времена, обращаясь в банки за кредитами.

Основной проблемой этой отложенной окупаемости является то, что на сегодняшний день деньги тоже стоят денег. Компании часто покрывают свои текущие расходы на функционирование от полученных прибылей и платят за капитальные

затраты, обращаясь в банки за кредитами. Вы выигрываете, если компания растет быстрее, чем повышается ценность денег. Если вы не можете расти с такой же скоростью или — хуже того, не можете получить кредит, то капитальные затраты могут пробить серьезную брешь в бюджете вашей организации.

Сравнение затрат

Инфраструктуры управляемых услуг, предоставляемые сервис-провайдерами удаленных услуг (MSP), а также облачные инфраструктуры существенно снижают долю капитальных затрат и других форм предоплаты. Облачная инфраструктура добавляет еще и то преимущество, что ваши затраты связаны только с фактически потребляемыми услугами, а это значит, что вы часто можете связать ваши расходы на IT с выручкой вместо того, чтобы рассматривать их как накладные расходы.

В табл. 1.3 приведен пример, иллюстрирующий сравнение затрат на инфраструктуры IT в целью поддержания единственного транзакционного Web-приложения умеренно высокой отказоустойчивости с балансировщиком нагрузки, двумя серверами приложений и двумя серверами баз данных. В этом примере я использовал цены, типичные для октября 2008 г.

Таблица 1.3. *Сравнительный анализ затрат на организацию IT-инфраструктур различных типов*

	Внутренняя инфраструктура IT	**Управляемый сервис**	**Облачная инфраструктура**
Капитальные затраты	$40 000	$0	$0
Затраты на установку	$10 000	$5000	$1000
Ежемесячная плата за обслуживание	$0	$4000	$2400
Ежемесячная зарплата персоналу	$3200	$0	$1000
Чистая себестоимость за три года	$149 000	$129 000	$106 000

Данные табл. 1.3 приведены с учетом следующих предположений[1].

- Используются совершенно стандартные серверные системы 1u[1], например, такие как Dell 2950 или элитные экземпляры Amazon.

[1] В *главе 3* о методике расчетов будет рассказано чуть более подробно. — *Прим. перев.*

- Во внутренней инфраструктуре IT и среде управляемых сервисов используются аппаратные балансировщики нагрузки, а в облачной инфраструктуре применяются программные балансировщики нагрузки.
- Существенных требований по хранению данных или ширине полосы пропускания не предъявляется (различие требований к полосе пропускания или пространству для хранения данных может оказывать серьезное влияние на этот расчет).
- Цифры взяты по нижней границе диапазона ценового спектра (например, некоторые провайдеры управляемых услуг могут запрашивать цены, превышающие указанные почти в три раза).
- Чистая себестоимость деноминирована в современные доллары (иными словами, не следует беспокоиться об инфляции).
- В расчеты заложены проценты на капитал 10 % (проценты на капитал — это то, что мог бы принести вам вложенный капитал, если бы вы не потратили его на покупку сервера и установочные затраты — в основном, это значение складывается из процентной ставки плюс издержки неиспользованных возможностей).
- Использование инструментов управления облачной инфраструктурой от сторонних поставщиков, таких как enStratus или RightScale, включено в затраты на облачные вычисления.
- Затраты на зарплату обслуживающему персоналу указаны для внештатных работников (рассматриваемая инфраструктура не требует найма штатного сотрудника, работающего полный рабочий день).

Возможно, наиболее противоречивой частью этого анализа является то, что на первый взгляд он может показаться "сравнением яблок с апельсинами" в том, что касается затрат на балансировщик нагрузки. Реальность же заключается в том, что при такой архитектуре балансировщик нагрузки практически не нужен, если только вы не имеете дела с исключительно объемными Web-сайтами. По этой причине в данной ситуации при всех трех вариантах архитектуры вы могли бы обойтись программным балансировщиком нагрузки.

Однако использование программного балансировщика нагрузки весьма проблематично как во внутренней инфраструктуре IT, так и в среде, управляемой внешним сервис-провайдером. Происходит это по двум причинам.

- Вероятность отказа обычного сервера гораздо выше, чем вероятность отказа аппаратного балансировщика нагрузки. Поскольку при сценариях с использованием внутренней IT-инфраструктуры и среды, управляемой внешним сервис-провайдером, заменить сервер гораздо сложнее, потеря программного балансировщика загрузки будет неприемлемой. В то же время при использовании облачной инфраструктуры потеря программного балансировщика нагрузки пройдет незамеченной.

---

[1] Серверы 19" монтируются в специально предназначенные для этого шкафы и стойки. Высота таких серверов измеряется в юнитах (1 unit = 1u = 43,5 мм). Минимальная высота для сервера — 1U (юнит), максимальная — 8U. Подробнее см. **http://en.wikipedia.org/wiki/Rack_unit, http://tinyurl.com/325zrnx** — *Прим. перев.*

- Если вы инвестируете в реальные аппаратные средства, вам захочется иметь балансировщик нагрузки, который реально будет расти по мере роста ваших требований к инфраструктуре IT. Аппаратный балансировщик нагрузки в этой области предоставляет гораздо больше возможностей, нежели программный. Однако в облачной инфраструктуре вы можете при минимуме затрат добавить выделенный программный балансировщик нагрузки, и это не представит большой проблемы.

Кроме того, некоторые провайдеры, предоставляющие сервис облачных вычислений (например, GoGrid), бесплатно включают аппаратный балансировщик нагрузки, что делает всю дискуссию о том, какой балансировщик нагрузки следует предпочесть (программный или аппаратный) попросту неактуальной. Далее, Amazon планирует начать предоставление собственного решения по балансировке нагрузки в 2009 г[1]. Тем не менее, если вы не доверяете моим рассуждениям по сравнению аппаратных балансировщиков нагрузки с программными, вот стоимость использования программных балансировщиков для всех вариантов инфраструктуры: $134K для внутренней IT-инфраструктуры, $92K — для среды, управляемой сторонним сервис-провайдером, и $106K — для облачной инфраструктуры.

Нижняя граница

Если мы отбросим пониженные затраты, то управляемый сервис от правильно выбранного удаленного сервис-провайдера и облачные вычисления всегда будут финансово более привлекательны, чем поддержание собственной инфраструктуры IT. По всем финансовым показателям — капитальным затратам, совокупной стоимости владения (Total Cost of Ownership, TCO), сопутствующим затратам — внутренняя инфраструктура IT всегда оказывается аутсайдером.

По мере того, как ваша инфраструктура усложняется, определение того, какое решение будет более экономичным — удаленно управляемая среда, смешанная инфраструктура или облачные вычисления, становится все более сложным.

Если у вас есть приложение, о котором вы твердо знаете, что оно должно быть доступно постоянно и круглосуточно, и даже одна минута простоя обернется серьезными убытками и потому неприемлема, вы почти наверняка предпочтете прибегнуть к услугам сервис-провайдера управляемых услуг, и вы не будете слишком серьезно задумываться о разнице в ценах (в таких ситуациях они могут даже оказаться более выгодными при сценарии использования управляемой среды).

С другой стороны, если вы хотите добиться высокой отказоустойчивости по доступным ценам и 99,995 % надежности — это достаточно хороший с вашей точки зрения показатель, то ничего более выгодного, чем облачная инфраструктура, вы не найдете.

---

[1] Фактически к моменту подготовки русского издания этой книги данный сервис уже предоставляется, см. **http://blog.rightscale.com/2009/05/18/amazon-load-balancing-monitoring-auto-scaling/**. — *Прим. перев.*

Уркхарт о препятствиях для отказа от собственной инфраструктуры IT

В ноябре 2008 года Джеймс Уркхарт (James Urquhart) и я начали дискуссию в Twitter[1], где обсуждали вопрос совокупной стоимости владения облачной инфраструктурой (Джеймс является рыночным аналитиком, определяющим стратегию Data Center 3 для Cisco Systems и членом сообщества блогеров CNET). В ходе этой дискуссии мы поняли, что я смотрел на проблему с точки зрения того, как начать новый бизнес "с нуля", в то время как Джеймс имел другой взгляд на проблему: с позиций реальности и входя в положение организации, которая уже сделала крупные капиталовложения в существующую инфраструктуру IT. В этом примечании приводится краткое обобщение результатов нашей дискуссии, которые Джеймс собрал вместе и кратко пересказал специально для этой книги.

Хотя при оценке перспектив создания нового предприятия "с нуля" действительно очень просто заразиться энтузиазмом по поводу экономических перспектив облачной инфраструктуры, большинство современных средних и крупных предприятий уже имеют собственную инфраструктуру IT, в которую они уже вложили достаточно крупные капиталы, и это надо учитывать при планировании и подсчетах расходов перехода на облачные вычисления.

Такие организации обычно уже имеют стойки, системы охлаждения и силовые инфраструктуры для поддержания новых приложений, и они не захотят нести эти капитальные расходы заново. Следовательно, стоимость добавления новых серверов в существующую инфраструктуру и управления ими окажутся значительно ниже, чем в приведенных примерах.

В этом случае уже сделанные капиталовложения часто оказываются решающим фактором, потому что расчеты показывают, что гораздо дешевле будет расширить существующую инфраструктуру (возможно, дополнив ее некоторыми функциями по автоматизации) и получить при этом достаточно стабильную загрузку мощностей. Таким образом, уже сделанные капиталовложения в существующую инфраструктуру часто действуют как препятствие к переходу на облачные вычисления.

Конечно, существуют определенные классы приложений, которые даже крупным предприятиям будет выгоднее перевести на облачную инфраструктуру. К числу таких приложений относятся:

- приложения с нагрузками, варьирующимися в широком диапазоне, для которых пиковую нагрузку и, соответственно, потребность в ресурсах предсказать сложно. Закупать для таких приложений оборудование "с запасом" на случай пиковых нагрузок будет экономически невыгодно, потому что бóльшую часть времени эти мощности использоваться не будут;

- приложения, при работе которых от случая к случаю или периодически возникают пиковые нагрузки. Например, к такому классу приложений относятся приложения для расчета налогов или системы расчета прибылей перед длительными праздниками. Облачная обработка данных может обеспечить поддержку работы приложения, динамически предоставив дополнительные мощности в момент пиковых нагрузок, с помощью технологии, которая называется "cloudbursting"[2];

---

[1] **http://blog.jamesurquhart.com/2008/12/enterprise-barrier-to-exit-to-cloud.html**.

[2] Cloud bursting (англ. "ливень, проливной дождь") — в облачных вычислениях этот термин может использоваться двояко, причем значение его может быть как положительным, так и отрицательным. В отрицательном смысле это означает крах всей облачной среды в момент пиковой нагрузки вследствие неспособности сбалансировать нагрузку между различными компаниями, пользующимися облаком. Описание такой ситуации см. здесь: **http://www.roughtype.com/archives/2007/04/intuits_cloudbu.php**. В положительном же смысле этот термин обозначает технологию динамического развертывания ПО, которое работает, используя внутренние вычислительные ресурсы организации, в общественную облачную инфраструктуру в момент возникновения пиковых нагрузок. Источник: **http://sites.google.com/site/cloudcomputingwiki/Home/cloud-computing-vocabulary**. — *Прим. перев.*

- новые приложения, описываемые в данной книге, которые потребуют дополнительно пространства в центре обработки данных или капиталовложений в существующую инфраструктуру, например, таких как новые системы охлаждения или электропитания.

Мне представляется очень забавным — а может быть, уникальным и знаменательным — тот факт, что движущей силой, стоящей за развитием определенных аспектов рынка облачной обработки данных, будут не крупные организации, владеющие множеством центров обработки данных и сотнями тысяч серверов, а именно малые предприятия, которые имели несколько серверов и, в конце концов, отказались от использования собственной инфраструктуры и перешли на Amazon.

Модели облачных инфраструктур

Мы обсудили некоторые технологии, являющиеся составной частью облачной обработки данных, и общее значение облачной инфраструктуры. Прежде, чем перейти к построению облачных систем, необходимо поговорить о различных моделях облачной инфраструктуры. Основное внимание будет уделено наиболее популярной модели, с которой и будет работать большинство пользователей — Web-сервисам Amazon (Amazon Web Services). Но, тем не менее, будет рассмотрен и ряд других опций.

Классифицировать эти сервисы было бы очень легко, если бы между ними существовали четкие различия. Но таких различий нет, и сервисы представляют собой, скорее, непрерывный спектр вариаций, плавно переходящих в друг друга — от управляемых сервисов (managed services) до таких разновидностей, как "Инфраструктура как сервис" (Infrastructure as a Service, IaaS) или "Платформа как сервис" (Platform as a Service, PaaS).

Платформа как сервис

Среды PaaS (Platform as a Service) предоставляют вам инфраструктуру, а также функционально полные среды обслуживания и разработки приложений для развертывания вашего Web-приложения. Вы программируете, используя платформу разработки приложений, предоставленную вам поставщиком, и предоставляете ему заботиться обо всех деталях развертывания готового приложения.

Наиболее широко используемым образцом PaaS является Google App Engine. Чтобы воспользоваться Google App Engine, вы пишете ваши приложения на Python, используя инфраструктуры разработки, предоставляемые Google, и инструменты, предназначенные для работы с файловой системой и хранилищами данных Google. Этот подход хорошо работает для приложений, которые необходимо быстро развертывать, и которые не предъявляют существенных требований к интеграции.

Отрицательной стороной подхода PaaS является его привязка к конкретному поставщику. Например, если вы работаете с Google, вы должны писать свои приложения на языке программирования Python, используя при этом специфический интерфейс прикладного программирования (API) от Google.

Python — это замечательный язык программирования! По правде сказать, это мой любимый язык — но он не является центральным в большинстве команд разработчиков, и не все его хорошо знают. Даже если у вас есть сотрудники, которые хорошо владеют Python, вам придется смириться с мыслью о том, что приложение Google App Engine, возможно, будет хорошо работать только в инфраструктуре Google.

Инфраструктура как сервис

Идея IaaS (Infrastructure as a Service) в данной книге является центральной темой. В ней приведено немало примеров, иллюстрирующих этот подход на примере одного из ключевых игроков в этой области — Amazon Web Services (AWS). У AWS есть ряд конкурентов, которые предлагают иные подходы к решению проблемы IaaS. Эти альтернативные подходы позиционируются как имеющие ключевое значение для различных типов клиентов, использующих облачные инфраструктуры.

AWS основывается на чистой виртуализации. Amazon имеет в собственности все оборудование и управляет всей сетевой инфраструктурой, а клиентам принадлежит все, что содержится в гостевых операционных системах (guest operating system), работающих на виртуальных машинах. Вы запрашиваете виртуальные экземпляры по требованию и освобождаете их по мере того, как они перестают быть вам нужными. Одно из своих ключевых преимуществ Amazon видит в том, что они обязуются не выделять на виртуализацию больше ресурсов, нежели реально могут.

AppNexus — это другой подход к описываемой проблеме. Как и в случае с AWS, AppNexus позволяет получить доступ к серверам по мере необходимости ("по требованию"). Однако AppNexus предлагает выделенные серверы, с виртуализацией, реализованной "поверх". При этом вы можете быть уверены в том, что ваши приложения не конкурируют за ресурсы ни с какими другими приложениями, и что вы можете обеспечить выполнение любых требований, которые предписывают необходимость обладать полным контролем над физическими ресурсами сервера.

Гибридная обработка данных (Hybrid computing) использует преимущества этих моделей, предлагая виртуализацию там, где это уместно, и выделенное оборудование там, где это требуется. Кроме того, большинство поставщиков гибридной модели обработки данных (например, Rackspace или GoGrid) основывают свои модели на той идее, что клиенты центров обработки данных в традиционном понимании этого термина хотят пользоваться услугами традиционного центра обработки данных — только в данном случае этот центр находится в облаке.

Как будет показано далее в этой книге, существует множество причин, по которым вам может не подойти решение, основанное на модели "чистой виртуализации":

- нормативные требования, которые предписывают, чтобы определенные функции работали только на выделенном оборудовании;
- требования по производительности — особенно в области ввода/вывода, которые не поддерживают часть ваших приложений;

- необходимость интеграции с некоторыми наследуемыми системами, в процессе разработки которых не учитывались какие бы то ни было стратегии интеграции с Web.

Во всех перечисленных случаях ваши требования может удовлетворить облачный подход, в большей степени ориентированный на выделение физического оборудования.

Частные облачные вычислительные среды

Я не приверженец термина "частные облачные вычислительные среды" или "частные облака" (private clouds), но это — именно тот термин, который вы чаще всего будете слышать, когда речь зайдет о предоставлении виртуальных сред по требованию в центрах обработки данных с внутренним управлением. В *частном облаке* организация создает среду виртуализации на своих собственных серверах, которые расположены в центрах обработки данных, принадлежащих либо самой этой организации, либо сервис-провайдеру управляемых услуг. Эта структура полезна для компаний, которые либо сделали внушительные капиталовложения в развитие собственной IT-инфраструктуры, либо чувствуют, что им абсолютно необходимо иметь полный контроль над всеми аспектами их инфраструктуры.

Ключевым преимуществом частных облаков является именно контроль. Вы сохраняете полный контроль над вашей инфраструктурой, но также получаете и все преимущества, которые дает виртуализация. Причина, по которой я не являюсь фанатом термина "частное облако", заключается в том, что, если основываться на критериях, сформулированных ранее в этой главе, я не вижу в частном облаке признаков настоящего облачного сервиса. В частности, здесь вы не свободны от капитальных затрат и не получаете практически неограниченной свободы облачных вычислений. Как отметил Джеймс Уркхарт (*см. примечание "Уркхарт о препятствиях для отказа от собственной инфраструктуры IT"*), и я с ним согласен, использование частного облака может оправдать нежелание действительно переходить на облачные вычисления. Я также считаю, что в долгосрочной перспективе это может поставить под угрозу конкурентоспособность вашей компании.

Комбинация моделей

И наконец, не следует упускать из виду Microsoft Azure. Microsoft Azure[1] сочетает все аспекты облачной обработки данных, от частных облаков до PaaS. При использовании Microsoft Azure вы пишете приложения, пользуясь технологиями

[1] Microsoft Azure — технология, позволяющая объединить имеющиеся приложения в "вычислительное облако", повышая их производительность.
Подробнее о Microsoft Azure см. **https://www.microsoft.com/windowsazure/products/**, **https://secure.wikimedia.org/wikipedia/en/wiki/Azure_Services_Platform**, **http://www.osp.ru/pcworld/2010/09/13004178/** — *Прим. перев.*

Microsoft, и сначала можете развернуть их в частном облаке, а затем — перенести их в публичное облако. Как и в случае с Google App Engine, вы пишете приложения для патентованной инфраструктуры разработки приложений. В случае с Azure инфраструктура основана на более "вездесущей" платформе .NET и, таким образом, легче переносится из одной среды Microsoft в другую.

Обзорная информация об Amazon Web Services

Моя цель в этой книге заключается в том, чтобы следовать наиболее общим принципам которые вы можете применять в любой облачной среде. Однако в реальности большинство из вас примут решение воспользоваться именно средой AWS. Игнорировать этот факт, по меньшей мере, неразумно; поэтому я в дальнейших главах этой книги приведу большое количество примеров, которые используют среду AWS.

AWS — это собирательное описание Amazon для всех предоставляемых компанией сервисов, основанных на Web-технологиях. Он охватывает широкий диапазон сервисов, каждый из которых подпадает под концепцию облачной обработки данных (хотя, честно говоря, я не очень понимаю, как классифицировать Amazon Mechanical Turk[1]). В данной книге основное внимание будет сосредоточено на технологиях, используемых следующими сервисами облачной инфраструктуры Amazon:

- Amazon Elastic Cloud Compute (Amazon EC2);
- Amazon Simple Storage Service (Amazon S3);
- Amazon Simple Queue Service (Amazon SQS);
- Amazon CloudFront;
- Amazon SimpleDB.

В контексте транзакционных систем особый интерес представляют две технологии — Amazon EC2 и Amazon S3.

[1] Сервис Amazon Mechanical Turk (MTurk) — это один из сервисов семейства AWS, который получил свое название от первого шахматного автомата, известного под названием "Турок" (потому что он представлял собой выполненную в натуральную величину восковую фигуру мужчины, одетого в турецкий костюм и сидящего перед ящиком с шахматной доской). Этот шахматный автомат был сконструирован в XVIII веке Вольфгангом фон Кемпеленом (Wolfgang Ritter von Kempelen). Впоследствии выяснилось, что в действительности это устройство не было автоматом, потому что в специальном отделении ящика сидел живой (и очень сильный) шахматист, который и управлял всеми действиями аппарата. Аналогичным образом действует и сервис Amazon Mechanical Turk: с его помощью компании (пока только расположенные на территории США) — так называемые заказчики (Requesters) — размещают на сайте несложные задания, которые, тем не менее, человек заведомо выполнит лучше машины. Зарегистрировавшиеся пользователи (workers) выполняют эти задания и получают за это деньги. Подробнее см.: **http://en.wikipedia.org/wiki/Amazon_Mechanical_Turk**, **http://moneynews.ru/9206/**. — *Прим. перев.*

Как я упоминал ранее, очереди сообщений имеют критическое значение в грид-системах, а также играют важную роль в большом количестве иных транзакционных систем. Однако они не типичны среди Web-приложений, поэтому в этой книге не будет уделяться особо пристального внимания таким сервисам, как, например, Amazon SQS.

Учитывая, что "сердцем" транзакционной системы является база данных, вы могли бы подумать, что сервис Amazon SimpleDB мог бы оказаться критически важным компонентом транзакционного приложения в облаке Amazon. Однако в действительности Amazon SimpleDB, как и следует из названия, представляет собой очень простую базу данных, и, следовательно, не подходит для крупномасштабных Web-приложений. Более того, это патентованная (proprietary) СУБД, а это значит, что приложение, слишком тесно интегрированное с Amazon SimpleDB, будет "привязано" к облаку Amazon.

Amazon Elastic Cloud Compute (EC2)

Amazon EC2 — это и есть настоящее "сердце" облака Amazon. Этот сервис предоставляет API Web-сервисов для выделения виртуальных серверов, управления ими и освобождения после того, как они стали ненужными с возвращением освобожденных ресурсов в облако Amazon. Иными словами, любое приложение, работающее где угодно в Интернете, может запустить виртуальный сервер в облаке Amazon, осуществив единственный вызов к Web-сервисам Amazon.

На момент написания этой книги основание Amazon EC2 в США состояло из трех центров обработки данных на восточном побережье США и двух — в Западной Европе. Вы можете отдельно создать учетную запись для европейского центра обработки данных Amazon, но вы не можете смешивать и комбинировать американскую и европейскую среды. На серверах в этих средах работает в высшей степени индивидуализированная версия Xen гипервизора (hypervisor) на основе открытого кода (Open Source). Эта виртуальная среда Xen позволяет динамически выделять и освобождать серверы, а также дает возможности, необходимые для предоставления изолированных компьютерных сред гостевым серверам.

Если вы хотите запустить виртуальный сервер в среде Amazon, вам необходимо создать новый узел на предопределенном образе машины Amazon (Amazon Machine Image, AMI). В состав AMI входят ваша операционная система и другое предварительно подготовленное (prebuilt) программное обеспечение. Большинство людей начинают со стандартного AMI на основе их предпочтительной операционной системы, настраивают его, создают новый образ, а затем запускают свои серверы на основе собственных индивидуально настроенных образов.

Сам по себе, сервис EC2 предоставляет два вида пространства для хранения:

- "краткосрочное" или "эфемерное" пространство (Ephemeral storage), привязанное к узлу, срок жизни которого истекает с истечением срока жизни узла;
- блочное пространство (Block storage), которое работает аналогично сети хранения данных (Storage Area Network, SAN) и не исчезает с течением времени.

Многие конкуренты Amazon тоже предоставляют узлам постоянное пространство для хранения данных, чтобы все выглядело более похожим на привычный центр обработки данных.

В дополнение к этому, серверы в EC2, как и любые другие серверы в Интернете, могут получать доступ к постоянному облачному пространству для хранения данных — Amazon S3. Использование этой комбинации позволяет добиться как финансовой экономии, так и повышения общей эффективности работы.

Чтобы обезопасить вашу сеть в пределах облака, вы можете управлять правилами виртуального брандмауэра, который будет фильтровать трафик, поступающий на ваши виртуальные узлы. Вы должны будете определить правила маршрутизации путем создания групп безопасности и ассоциирования правил фильтрации трафика с этими группами. Например, вы можете создать группу DMZ[1], которая разрешает входящий трафик на ваши серверы через порты 80 и 443 из Интернета, но запрещает любой другой входящий трафик.

Amazon Simple Storage Service (S3)

Amazon S3 представляет собой облачное хранилище данных, доступное в реальном времени через API Web-сервисов из любой точки Интернета. Используя этот API, вы можете хранить любое количество объектов, размер которых варьируется от 1 байта до 5 Гбайт, в более или менее плоском пространстве имен.

Важно *не* думать об Amazon S3 как о файловой системе. Я видел много случаев, когда люди наживали себе неприятности, когда воспринимали Amazon S3 таким образом. В первую очередь, Amazon S3 имеет двухуровневое пространство имен. На первом уровне вы имеете так называемые "корзины" или "ведра" (buckets). Их можно считать аналогом каталогов в файловой системе, если вам так нравится, потому что они содержат данные, которые вы помещаете в S3. Однако, в отличие от традиционных каталогов, вы не можете организовать их иерархически — корзины не допускают вложенности (иначе говоря, вы не можете создать корзину внутри корзины). Возможно, еще более важное значение имеет тот факт, что пространство имен корзин предоставлено в общий доступ всем клиентам Amazon. Поэтому вам необходимо особо позаботиться о том, чтобы имена ваших корзин не пересекались с именами других корзин. Проще говоря, не следует создавать корзины с именами наподобие "Documents".

Еще один важный момент, который следует иметь в виду, заключается в том, что служба Amazon S3 работает относительно медленно. В действительности эта служба работает очень быстро для сервиса, развернутого в Интернете, но если вы

[1] DMZ (DeMilitarized Zone) — демилитаризованная зона. Технология обеспечения защиты информационного периметра, при которой серверы, отвечающие на запросы из внешней сети, находятся в особом сегменте сети (который и называется DMZ) и ограничены в доступе к основным сегментам сети с помощью брандмауэра (firewall) с целью минимизировать ущерб при взломе одного из общедоступных сервисов, находящихся в DMZ. Подробнее см. **http://en.wikipedia.org/wiki/Demilitarized_zone_(computing)**. — *Прим. перев.*

ожидаете, что она будет отвечать на ваши запросы так же быстро, как локальный жесткий диск или SAN, то вы будете глубоко разочарованы. Таким образом, использование Amazon S3 в качестве оперативного хранилища не слишком благоразумно.

Наконец, доступ к S3 осуществляется через Web-службы, а не через файловую систему или WebDAV[1]. В результате этого приложения должны разрабатываться с учетом того, что они должны хранить данные в Amazon S3. Вероятно, еще важнее то, что вы не можете просто воспользоваться командой `rsync` для синхронизации каталога с S3 без специальных инструментов, которые применяют Amazon API и, таким образом, обходят ограничения S3.

Итак, мы разобрались с тем, чего не следует ожидать от Amazon S3 — а теперь рассмотрим, что мы можем ожидать от этой службы.

Amazon S3 позволяет поместить хранимые данные в облачную инфраструктуру и впоследствии при необходимости их извлечь. При этом вы можете быть практически полностью уверены в том, что извлеченные данные будут представлять собой единый и непротиворечивый блок информации. Ключевым преимуществом Amazon S3 является то, что вы можете сбрасывать и сбрасывать данные в Amazon S3, совершенно не беспокоясь о проблеме нехватки пространства для их хранения. Иначе говоря, для большинства пользователей S3 служит в качестве краткосрочного или долгосрочного хранилища резервных копий.

ХРАНИЛИЩЕ CLEVERSAFE[2]

Облачные системы хранения данных призваны решить уникальные проблемы, которые невозможно решить с помощью более старых технологий хранения данных. Технологии хранения данных на основе RAID и репликации не слишком хорошо подходят для использования в облачной инфраструктуре, поскольку не существует удобных путей их масштабирования до эксабайтного[3] уровня. Старые технологии хранения данных, основывающиеся на создании избыточных копий с целью повышения надежности, приводят к тому, что вы получаете системы, которыми сложно управлять. К тому же эти системы требовательны к полосе пропускания и дороги.

---

[1] WebDAV (Web-based Distributed Authoring and Versioning) — защищенный сетевой протокол высокого уровня, работающий поверх HTTP и обеспечивающий доступ к объектам и коллекциям. Подробнее см. **http://www.webdav.org/**, **http://plone.org.ru/docs/howto/WebDAV**. — *Прим. перев.*

[2] Cleversafe, Inc. — это частная компания, зарегистрированная в США, которая является пионером в области разработки технологий распределенного хранения данных. Распределенное хранение (Dispersed storage) представляет собой специальную технологию хранения данных, при которой сохраняемые данные разбиваются на "секции" (slices), снабжаемые так называемыми кодами восстановления (erasure codes), после чего каждая "секция" сохраняется независимо от остальных на независимых носителях информации, доступ к которым осуществляется через TCP/IP. Благодаря свойствам кодов восстановления распределенное хранилище может выдерживать множественные отказы индивидуальных носителей без потери возможности восстановления данных. Подробнее см. **http://en.wikipedia.org/wiki/Cleversafe**. — *Прим. перев.*

[3] 1 Эбайт = 2^{60} байт или 1 048 576 Тбайт, см. **http://en.wikipedia.org/wiki/Exabyte**. — *Прим. перев.*

Облачная платформа Cleversafe основана на уникальной технологической разработке компании — рассредоточенном хранении (зарегистрированная торговая марка Dispersed Storage). Технология рассредоточенного хранения заключается в том, что данные разбиваются на "секции" (slices) и сохраняются на разных физических носителях, расположенных в различных географических точках. Алгоритмы, использованные для рассредоточения данных, основаны на концепции контроля четности (parity), но по усложненной процедуре, поскольку эти алгоритмы позволяют из подмножества данных восстановить весь набор данных целиком. Например, ваши данные могут быть рассредоточены по 12 различным географическим точкам, и при этом для восстановления полного набора данных достаточно, чтобы сохранились любые 8 подмножеств данных. Эта технология, известная как *рассредоточение информации по дисперсионным алгоритмам* (information dispersal), позволяет добиться географической избыточности и высокой отказоустойчивости, избежав при этом затрат на репликацию данных.

В апреле 2008 года компания Cleversafe реализовала свои технологии рассредоточения данных на аппаратном уровне, встроив их в устройства, предоставляющие внешний интерфейс пользователям с помощью стандартных протоколов, таких как REST API или iSCSI. Эти устройства выполняют задачу по разбиению информации и распределению ее по узлам хранения, при этом размеры исходного файла увеличиваются примерно в 1,3 —1,6 раз (для сравнения — при использовании традиционной репликации этот коэффициент составляет 3).

Компании используют устройства Cleversafe Dispersed Storage для построения публичных и частных облаков в качестве основы для инфраструктуры Software as a Service. Рассредоточенное хранилище соответствует всем требованиям облачной инфраструктуры, так как оно предоставляет пространство для хранения по требованию, и выделенное пространство будет доступно из любой географической точки.

Кроме того, рассредоточение позволяет повысить уровни безопасности в облаке, не обязательно требуя шифрования, поскольку каждая "секция" (slice) содержит слишком мало информации, чтобы быть полезной. Данная уникальная архитектура помогает преодолеть обоснованные опасения пользователей за данные, хранящиеся вне их непосредственного контроля. Такие опасения часто становятся препятствием к принятию решения о хранении данных за пределами предприятия. В то время как, например, ленточный накопитель с резервной копией содержит полную копию данных, доступ к единственному устройству, содержащему отдельный фрагмент рассредоточенной копии, не даст злоумышленнику ничего.

В дополнение к сказанному, рассредоточенное хранилище может масштабироваться в широких пределах и дает возможность хранения петабайтов[1] данных. Путем добавления серверов в облачную инфраструктуру с автоматическим обнаружением пространства для хранения общее пространство, доступное для хранения данных, можно легко нарастить, а производительность можно масштабировать за счет добавления дополнительных накопителей. Средства виртуализации обеспечивают быстрое развертывание и предоставление пространства для хранения данных по требованию. Все перечисленные возможности позволяют оптимизировать усилия по администрированию рассредоточенного хранилища.

Рассредоточенное хранилище подходит для хранения и распространения крупных объектов. Эта возможность представляет собой краеугольный камень информационных технологий в современном обществе, где видео, изображения и другая мультимедийная информация всесторонне влияют на каждый аспект жизни. Рассредоточение как будто специально было разработано для естественной балансировки нагрузки и распределенной обработки данных за счет множества "секций" (Slices), разбросанных по всей сети и

[1] 1 Пбайт = 2^{50} байт = 1024 Тбайт. См. **http://en.wikipedia.org/wiki/Petabyte**. — *Прим. перев.*

используемых для реконструкции исходного файла. Это означает, что предприятия не должны отдельно платить за создание сети, предназначенной для распространения информационного содержимого.

Рассредоточенное хранение представляет собой инновационный подход к облачному хранению данных, который сыграет важную роль в ходе совершенствования облачных технологий и заменит традиционные методы хранения данных.

Amazon Simple Queue Service (SQS)

Amazon SQS — это краеугольный камень во всех проектах Amazon в области грид-обработки данных. Как и в случае с сервисом очереди сообщений, Amazon SQS принимает сообщения и передает их серверам, "подписанным" на очередь сообщений.

Как правило, система обмена сообщениями позволяет множеству компьютеров обмениваться информацией, не имея никаких сведений друг о друге. Отправитель просто передает короткое сообщение (в Amazon SQS его длина не превышает 8 Кбайт) в очередь сообщений и продолжает заниматься своим делом. Получатель принимает сообщение из очереди и оперирует с его содержимым.

Например, сообщение может содержать следующие инструкции: "Обработать набор данных `123.csv` в корзине S3 `s3://fancy-bucket` и передать результаты в очередь сообщений Y". Одним из преимуществ, предоставляемых системой на базе очереди сообщений, является то, что отправитель не должен идентифицировать получателя или выполнять обработку ошибок в случае перебоев со связью. Получатель даже не обязательно должен быть активным в момент отправки сообщения.

Система Amazon SQS хорошо вписывается в концепцию облачной обработки данных именно из-за своей простоты. Большинству систем, основанных на очереди сообщений, требуется только очень простой API, позволяющий передать сообщение в очередь, извлечь его, полагаясь на целостность сообщения в очереди. Написание и поддержание такой простой программы может быть сложной задачей, но и приобретение коммерческого пакета может быть не менее сложным и дорогим решением.

Amazon CloudFront

Amazon CloudFront — это облачная сеть распространения информационного содержимого (content distribution network, CDN), которая на момент написания этих строк представляла собой новое предложение от Amazon Web Services. Этот сервис позволяет распространять ваше информационное наполнение (интернет-контент) по периферии сети с учетом того, что информация будет доставлена из точки, близкой к местоположению запросившего ее пользователя. Иными словами, если посетитель сайта находится в Лос-Анджелесе, то он получит информацию с сайта Amazon, расположенного в Лос-Анджелесе, а посетитель из Нью-Йорка получит информацию с сервера, расположенного в Нью-Йорке. Вам достаточно поместить

информацию в сеть S3, и она быстро распространится по конечным точкам сети Amazon, где станет доступна для быстрой доставки потребителям.

Amazon SimpleDB

Amazon SimpleDB — это необычная комбинация структурированного хранилища данных с более высокой отказоустойчивостью, чем типичные экземпляры MySQL или Oracle, и очень скромными базовыми потребностями в реляционном хранилище. Это очень мощный сервис, предназначенный для тех, кого больше волнуют доступность реляционных данных и в меньшей степени — сложность реляционной модели или управление транзакциями. На основании моего личного опыта я могу сказать, что эта целевая аудитория составляет очень небольшое подмножество пользователей реляционных приложений. Впрочем, такое приложение, как Amazon SimpleDB, может быть чрезвычайно полезным в информационных средах с высокой степенью нагрузки, например, таких как системы управления Web-контентом.

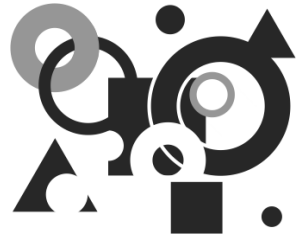
К числу преимуществ Amazon SimpleDB относятся:

- отсутствие необходимости в администраторе базы данных (database administrator, DBA);
- очень простой интерфейс прикладного программирования (API) для запросов данных через Web-сервисы;
- доступность кластерной системы управления базами данных (database management system, DBMS);
- высокая масштабируемость с точки зрения потребностей в пространстве для хранения данных.

Если вам требуется вся мощь реляционной базы данных, то инструмент Amazon SimpleDB вам не подойдет. Но, с другой стороны, если в качестве приложения для работы с базой данных вы выбрали bdb[1], то Amazon SimpleDB будет для вас идеальным выбором.

[1] Berkeley DB (BDB) — это высокопроизводительная встраиваемая база данных, реализованная в виде библиотеки. См. **http://ru.wikipedia.org/wiki/Berkeley_DB/** — *Прим. перев.*

Глава 2



Облачная обработка данных Amazon

Как я уже упомянул в предыдущей главе, эта книга представляет собой общее руководство, предназначенное для разработчиков ПО и системных администраторов, которые ставят перед собой цели разработки транзакционных Web-приложений и их развертывание в облачной инфраструктуре любого типа. Однако термин "облачная инфраструктура" многие люди, работающие с облачными вычислениями, воспринимают как синоним терминов Amazon EC2 и Amazon S3, поэтому необходимо привести обзор облачных вычислений применительно к реализации Amazon.

Amazon S3

Amazon Simple Storage Service (S3) — это долгосрочное хранилище данных на основе облачной обработки. Этот сервис работает независимо от остальных сервисов, предоставляемых Amazon. Фактически приложения, которые вы разрабатываете для хостинга ваших собственных серверов, могут использовать Amazon S3, без необходимости перемещения в облако иным образом. Когда компания Amazon называет S3 "простым хранилищем" (simple storage), то она имеет в виду набор функций, предоставляемых сервисом, а не простоту использования. Amazon S3 позволяет вам переместить данные в облако и извлечь их обратно. При этом вам нет необходимости знать что-либо о том, как именно и где именно в облаке хранятся ваши данные. Если вы воспринимаете Amazon S3 как удаленную файловую систему, то вы глубоко заблуждаетесь. Amazon S3 во многих отношениях гораздо примитивнее файловой системы. В действительности вы не храните там нечто, называемое "файлами", вы храните там объекты. Более того, объекты (objects) хранятся в "корзинах" (buckets), а не в каталогах (directories), как в файловой системе. Хотя эти различия могут казаться чисто семантическими, вы должны учитывать их, принимая во внимание следующие важные факты:

- объекты, хранимые в S3, не могут иметь размер, превышающий 5 Гбайт;

- "корзины" (Buckets) существуют в плоском пространстве имен, предоставленном в общий доступ всем пользователям Amazon S3. Вы не имеете возможности создавать вложенные корзины и должны быть предельно внимательны к пересечениям в пространстве имен;
- вы можете сделать свои корзины и объекты доступными для публичного просмотра;
- без помощи инструментов от сторонних разработчиков вы не можете "монтировать" (mount) хранилище S3. На самом деле, я не слишком одобрительно отношусь к самой идее "монтирования" S3, как раз из-за столь существенной концептуальной разницы между S3 и файловой системой. Это различие делает неправильной саму мысль о рассмотрении S3 в качестве файловой системы.

Доступ к S3

Прежде чем получить доступ к S3, вам необходимо создать учетную запись Amazon Web Services. При этом вы можете запросить пространство для хранения данных на серверах, расположенных либо в США, либо в Европе, либо даже в Азиатско-Тихоокеанском регионе[1]. Когда речь заходит о хранении данных, выбор местоположения серверов — это вопрос не только вашего места жительства. Как будет рассказано далее в этой книге, на решение о месте хранения ваших облачных данных влияют административные вопросы и различные аспекты, касающиеся конфиденциальности информации. В этой главе я рассматриваю вопрос доступа к S3, исходя из предположения, что вы выбрали хранилище, ближайшее к вам.

Web-сервисы

Amazon предоставляет доступ к S3 как через SOAP[2] API, так и через REST[3] API. Хотя разработчики обычно лучше знакомы с разработкой Web-сервисов с помощью SOAP, REST представляет собой предпочтительный механизм для доступа к S3, вследствие того, что при обработке больших двоичных объектов (Blinary

[1] В апреле 2010 года компания Amazon запустила первый регион EC2 в Азиатско-Тихоокеанском регионе (Сингапур). Нет сомнений в том, что в будущем количество регионов будет расти. — *Прим. перев.*

[2] SOAP — протокол обмена структурированными сообщениями в распределенной вычислительной среде. Первоначально SOAP предназначался в основном для реализации удаленного вызова процедур (Remote Procedure Calls, RPC), а название представляло собой аббревиатуру от Simple Object Access Protocol ("простой протокол доступа к объектам"). Сейчас протокол используется для обмена произвольными сообщениями в формате XML, а не только для вызова процедур. Подробнее см. **http://ru.wikipedia.org/wiki/SOAP**, **http://www.w3.org/2002/07/soap-translation/russian/part0.html**. — *Прим. перев.*

[3] REST (Representational State Transfer, "передача состояния представления") — подход к архитектуре сетевых протоколов, обеспечивающих доступ к информационным ресурсам. Был описан в 2000 году Роем Филдингом (Roy Fielding), одним из создателей протокола HTTP. Подробнее см. **http://en.wikipedia.org/wiki/Representational_State_Transfer**. — *Прим. перев.*

Large Objects, BLOBs) через SOAP API возникают сложности. В частности, SOAP ограничивает размер объекта, которым можно манипулировать в S3, а также ограничивает любую обработку (например, индикатор статуса передачи), которую вам может потребоваться осуществлять над потоками данных по мере того, как они передаются в S3 и обратно.

Интерфейс прикладного программирования Amazon Web Services поддерживает следующие возможности:

- поиск корзин (buckets) и объектов;
- обнаружение их метаданных;
- создание новых корзин;
- загрузка новых объектов;
- удаление существующих корзин и объектов.

При манипулировании вашими корзинами вы при желании можете указать местоположение для хранения содержимого корзины. За исключением случаев, когда вам требуется по-настоящему тонкоструктурированный контроль над взаимодействиями с S3, я рекомендовал бы вам использовать "обертку" API (API wrapper) для вашего предпочитаемого языка программирования, которая будет абстрагировать вас от S3 REST API. Например, мои разработчики программируют на Java с применением Jets3t[1]. Чтобы начать работу с Amazon S3, вам наверняка пригодится клиентское приложение командной строки `s3cmd` для Amazon S3 (**http://s3tools.logix.cz/s3cmd**). Это приложение представляет собой "обертку" с интерфейсом командной строки для доступа к Web-сервисам S3. Данный инструмент написан на Python, а это значит, что вы можете изучить его исходный код — он послужит отличным образцом, иллюстрирующим хороший стиль написания приложений на Python для S3.

BitTorrent

Кроме того, Amazon предоставляет доступ к Amazon S3 по протоколу BitTorrent. BitTorrent представляет собой пиринговый (peer-to-peer, P2P)[2] протокол для кооперативного доступа к файлам. Поскольку BitTorrent является стандартным протоколом для совместного доступа к большим двоичным объектам, на рынке предлагается целый ряд клиентов и приложений, предназначенных для потребления и публикации данных с помощью BitTorrent. Если ваше приложение может использовать эту встроенную инфраструктуру, то возможно, имеет смысл воспользоваться преимуществами, предоставляемыми поддержкой протокола BitTorrent со

[1] См. **http://bitbucket.org/jmurty/jets3t/wiki/Home**. — *Прим. перев.*

[2] Одноранговые, децентрализованные или пиринговые (peer-to-peer, P2P) сети — это компьютерные сети, основанные на равноправии участников. В таких сетях отсутствуют выделенные серверы, а каждый узел (peer) является как клиентом, так и сервером. В отличие от архитектуры "клиент-сервер", такая организация позволяет сохранять работоспособность сети при любом количестве и любом сочетании доступных узлов. Подробнее см. **http://ru.wikipedia.org/wiki/P2P**. — *Прим. перев.*

стороны Amazon S3. В общем случае, однако, транзакционные Web-приложения не используют BitTorrent для взаимодействия с S3.

S3 в действии

Чтобы проиллюстрировать практическое использование S3, воспользуемся утилитой `s3cmd` для передачи файлов в хранилище S3 и обратно. Команды, поддерживаемые этой утилитой, отражают функции API Web-сервисов, на которых она основывается. После того как вы скачаете эту утилиту, вам потребуется сконфигурировать ее, указав ваши ключ доступа к S3 и секретный ключ S3. Вне зависимости от того, какой инструментарий вы используете — эту утилиту или какое-нибудь другое средство, вам все равно будут нужны эти ключи для доступа к вашим частным корзинам в S3.

Первое, что вам потребуется сделать в S3 — это создать корзину, в которой вы будете хранить объекты. Это делается следующей командой:

```
s3cmd mb s3://BUCKET
```

Данная команда создает корзину и присваивает ей указанное вами имя. Как я уже заметил ранее в этой главе, пространство имен для вашей корзины доступно всем клиентам Amazon. Если только вы не являетесь первым читателем этой книги, крайне маловероятно, что только что приведенная команда будет выполнена успешно. Чтобы команда выполнилась успешно, вам необходимо заменить имя *BUCKET* на любое другое, уникальное для вас. Многие пользователи с тем, чтобы сделать имена корзин уникальными для себя, примерно так, как это делается для доменных имен, предваряют имена корзин префиксами. Тем не менее, имейте в виду, что какой бы стандарт именования вы ни приняли, ничто не может помешать другим пользователям нарушать ваше соглашение об именовании.

Именование корзин

Я уже несколько раз предупреждал вас о ситуациях с пространством имен корзин. Возможно, к этому моменту вы уже серьезно задумались о том, как же следует работать в условиях таких ограничений, касающихся именования.

В первую очередь, вам следует иметь в виду следующие правила именования.

- Имена могут состоять только из строчных букв (символов в нижнем регистре клавиатуры), цифр, точек, символов подчеркивания и дефисов. Начальным символом имени должна быть буква или цифра.
- Имя не должно составляться в стиле IP-адреса (иными словами, не допускаются имена наподобие 10.0.0.1).
- Длина имени должна составлять не менее 3 и не более 255 символов.

Возможно, вам захочется называть свои корзины таким образом, чтобы они образовали ваше собственное виртуальное пространство имен корзин. Например, мне вполне подойдет такое имя, как `com.imaginary.mybucketis`, потому что мне принадлежит домен `imaginary.com`. Важно отметить, что ничто не гарантирует вам того, что никакой другой пользователь не станет использовать ваш домен (или любое другое придуманное вами соглашение об именовании) в качестве префикса имен своих корзин. Поэтому

ваши приложения должны использовать достаточно интеллектуальный подход к созданию новых корзин и не должны быть привязаны к конкретной схеме именования.

Amazon предлагает следующие правила именования корзин, которые позволят вам создавать действительные и корректные URL для ваших объектов S3.

- Не следует создавать имена корзин, в состав которых входят символы подчеркивания (несмотря на то, что официально это разрешено).
- В идеальном варианте лучше всего ограничить длину имени 63 символами.
- Не следует создавать имена корзин, завершающиеся дефисом, а также следует избегать ситуаций, когда в составе имени за дефисом следует точка.

После того как корзина будет создана, ее можно начинать заполнять объектами:

```
s3cmd put LOCAL_FILE s3://BUCKET/S3FILE
```

Например:

```
s3cmd put home_movie.mp4 s3://com.imaginary.movies/home_movie.mp4
```

Утилита `s3cmd` ограничивает размеры ваших файлов до 5 Гбайт, что является следствием ранее упомянутого ограничения S3. Далее в этой книге мы обсудим стратегии обхода этого ограничения, методы обеспечения более высокого уровня безопасности, а также контроля целостности ваших объектов S3.

Когда объект вам потребуется, вы можете извлечь его из облака:

```
s3cmd get s3://BUCKET/S3FILE LOCAL_FILE
```

Например:

```
s3cmd get s3://com.imaginary.movies/home_movie.mp4 home_movies3.mp4
```

В приведенном примере файл с вашим домашним видео извлечен обратно на ваш настольный компьютер. Приведу еще несколько популярных команд, которые вы можете использовать.

- Вывод списка всех ваших корзин: `s3cmd ls`
- Вывод списка содержимого конкретной корзины: `s3cmd ls s3://BUCKET`
- Удаление объекта из корзины: `s3cmd del s3://BUCKET/S3FILE`
- Удаление корзины: `s3cmd rb s3://BUCKET`

Удалить корзину можно только в том случае, если она пуста. Прежде чем удалить корзину, вы должны пошагово, один за другим удалить из нее все объекты. Скоро в составе утилиты `s3cmd` появится опция `--recursive`, предназначенная для использования с командой `del`, но вы должны иметь в виду, что она всего лишь будет перечислять содержимое корзины и удалять объекты один за другим. Если вы используете API Web-сервисов, вам потребуется написать собственную подпрограмму рекурсивного удаления корзины, которая содержит объекты.

Amazon EC2

Когда речь заходит об облаке Amazon, большинство пользователей сразу же ассоциируют его с Amazon EC2. EC2 представляет собой вашу виртуальную сеть,

в составе которой работают все ваши виртуальные серверы. Однако эта сеть не является независимой. Если вы пользуетесь EC2, то вы должны пользоваться и S3 для хранения ваших образов машин (machine images), о которых речь пойдет совсем скоро. Помимо этого, S3 может вам потребоваться и для хранения другой информации. Поэтому, если вы пропустили предыдущий раздел, считая, что вы можете пользоваться только EC2, а S3 вам не требуется, вернитесь назад и перечитайте его!

Концепции EC2

EC2 — это сервис несколько более сложный, чем S3. Основные компоненты, формирующие Amazon EC2, и их взаимоотношения схематически изображены на рис. 2.1.

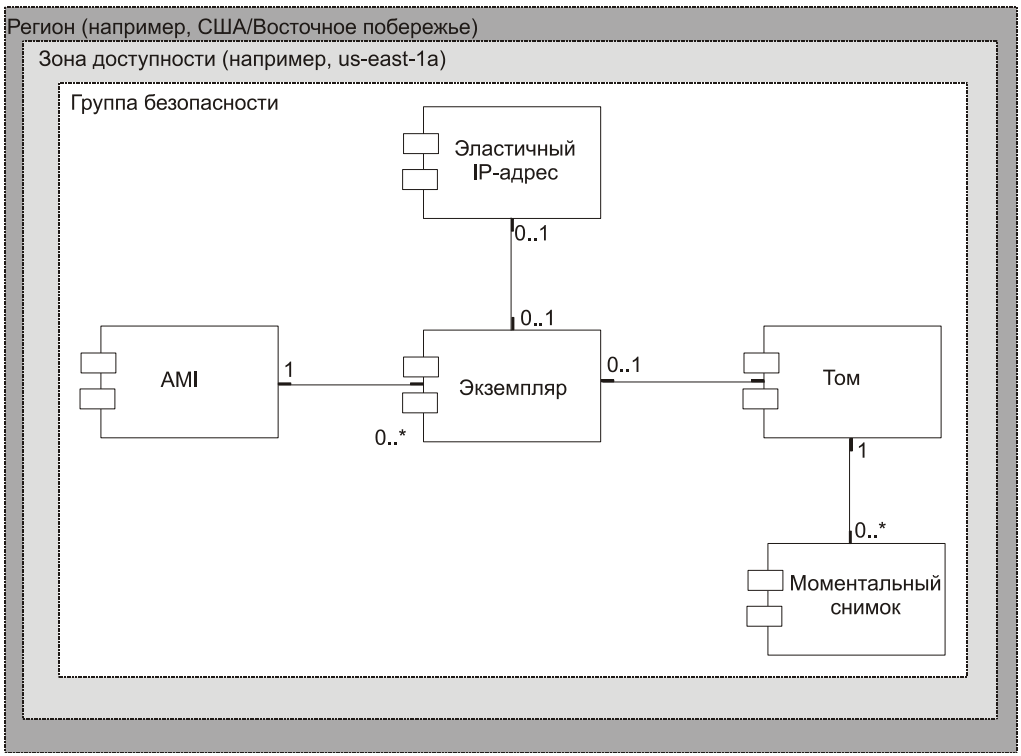


Рис. 2.1. Компоненты, поддерживающие Amazon EC2

Перечислим основные концепции Amazon EC2.

- *Экземпляр* (Instance) — экземпляром EC2 называется виртуальный сервер, на котором работает выбранная вами гостевая операционная система (guest operating system), созданный на основании образа машины (machine image), который использовался для клонирования экземпляра.

- *Образ машины Amazon* (Amazon Machine Image, AMI) — изначальная (pristine) копия вашего сервера, которая используется для запуска любого количества экземпляров. Если вы знакомы с концепцией клонирования (ghosting), то вы быстро поймете, что образ машины представляет собой клонированный образ, на основе которого вы можете создать любое количество серверов. Как минимум, образ машины содержит основную операционную систему плюс типовой набор преинсталлированных утилит. В зависимости от выбранной вами стратегии развертывания он может содержать и ваше предварительно построенное Web-приложение. Amazon предлагает набор предварительно построенных AMI, с помощью которых вы можете быстро начать работу. В дополнение к этому, существуют еще AMI от сторонних разработчиков, и, наконец, при желании вы можете собрать и свой собственный AMI с учетом ваших индивидуальных требований.
- *Эластичный IP-адрес* (Elastic IP address) — это просто статический IP-адрес, присвоенный вам. Термин "эластичный" звучит несколько странно и может дезориентировать: будьте уверены в том, что он означает именно статический IP-адрес, а не динамический. По умолчанию каждый экземпляр Amazon получает динамически назначаемый IP-адрес, который после завершения работы вашего экземпляра может быть присвоен другому пользователю. Эластичные IP-адреса зарезервированы для вас, что очень полезно для экземпляров, доступ к которым должен осуществляться по одному и тому же статическому IP-адресу.
- *Регион* (Region) — группа зон доступности, которые формируют единый географический кластер. На момент написания этой книги соглашение об уровне сервиса (service level agreement, SLA) компании Amazon для сервиса EC2 гарантировало доступность как минимум двух зон доступности в пределах региона на 99,95 % в течение 12-месячного периода.
- *Зона доступности* (Availability zone) — примерный аналог центра обработки данных. SLA гарантирует, что в двух зонах доступности никогда не будет общих точек отказа (points of failure). На настоящий момент Amazon имеет три зоны доступности в США (все — на восточном побережье) и две — в Западной Европе. Вы можете указать зону доступности, в которой вы собираетесь запускать свои экземпляры виртуальных серверов, чтобы создать определенный уровень географической избыточности для ваших приложений.
- *Группа безопасности* (Security group) — в очень грубом приближении группа безопасности может рассматриваться как аналог сетевого сегмента, доступ к которому контролируется брандмауэром. Вы запускаете экземпляры своих серверов в группах безопасности и, в свою очередь, группы безопасности определяют, кто и что имеет право получать доступ к вашим новым экземплярам.
- *Том блочной записи* (Block storage volume) — на концептуальном уровне представляет собой аналог SAN. Том блочной записи предоставляет хранилище для блочной записи, которое вы можете примонтировать на основе ваших экземпляров EC2. После этого том можно отформатировать по вашему усмотрению,

или вести на него запись неструктурированных данных (raw-формат). Можно даже создать на основе нескольких томов блочной записи виртуальный RAID.

- *Моментальный снимок* (Snapshot) — вы в любой момент можете создавать "моментальные снимки" ваших томов блочного хранения для целей резервного копирования или репликации. Эти моментальные снимки хранятся в Amazon S3, где их можно использовать для создания новых томов.

ОБРАЗЫ МАШИН ПО СРАВНЕНИЮ С ЭКЗЕМПЛЯРАМИ

Когда вы только начинаете работать с Amazon EC2, вы должны уяснить себе важное различие между образами машин (AMI) и экземплярами. Не расстраивайтесь, если сначала это отличие не вполне вам ясно, хотя в будущем эту разницу нужно прочувствовать, потому что ваша стратегия управления образами машин может оказать критическое влияние на ваши долгосрочные успехи в работе с облачными вычислениями.

Образ машины представляет собой прототип, по образу и подобию которого создаются ваши виртуальные серверы. Возможно, проще всего уяснить себе этот вопрос на концептуальном уровне, если воспринимать образы машин как копии жестких дисков вашего виртуального сервера, которые клонируются перед созданием и запуском нового сервера. Этот "новый сервер" как раз и будет представлять собой ваш экземпляр. Каждый раз, когда вы запускаете экземпляр, EC2 копирует образ машины на новый виртуальный жесткий диск, с которого и происходит затем загрузка экземпляра. На основе одного образа машины можно создать множество экземпляров.

Если вы программист, возможно, вам будет удобнее использовать такую аналогию: образ машины представляет собой класс, написанный на одном из языков программирования, а экземпляр EC2 — это экземпляр объекта.

Доступ к EC2

Как и в случае с Amazon S3, основным средством доступа к Amazon EC2 является интерфейс прикладного программирования (API) Web-сервисов. Amazon предоставляет ряд интерактивных инструментов, работающих "поверх" API Web-сервисов, в том числе:

- консоль Web-сервисов Amazon (Amazon Web Services Console), показанную на рис. 2.2 и доступную по адресу **http://console.aws.amazon.com**;
- плагин для Firefox ElasticFox;
- набор инструментов командной строки Amazon (Amazon Command Line tools).

Когда вы приобретете некоторый опыт работы с Web-сервисами Amazon, вы, скорее всего, начнете пользоваться надежными средствами управления инфраструктурой, например, такими как enStratus (**http://enstratus.com/**) и RightScale (**http://www.rightscale.com/**). Но в самом начале освоения облачных сервисов Amazon вы, по всей вероятности, будете работать с комбинацией инструментов, предоставляемых Amazon. В примерах, которые приводятся в оставшихся разделах этой главы, основное внимание уделяется инструментам командной строки, которые доступны по следующему адресу: **http://developer.amazonwebservices.com/connect/entry.jspa?externalID=351**. После того как вы скачаете инструменты командной

строки и освоите их, все остальное будет очень просто. Вне зависимости от того, какие средства вы используете в основном, необходимость в командной строке оспариванию не подлежит.

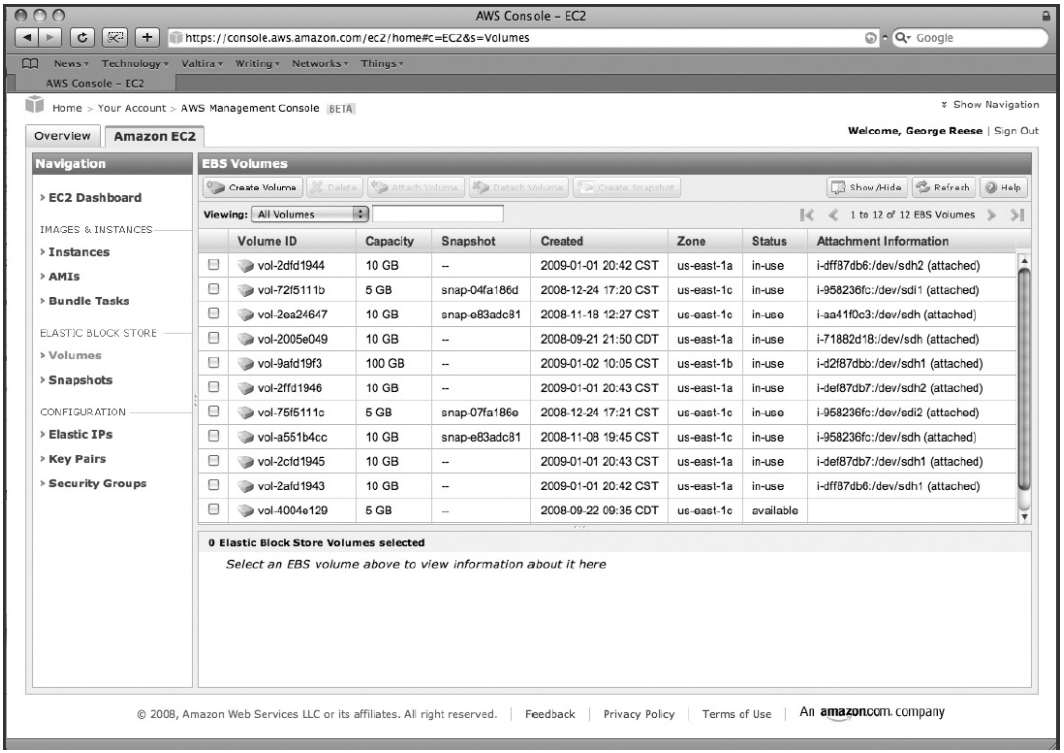


Рис. 2.2. Консоль Web-сервисов Amazon

Установка экземпляра

Самое простое, что вы можете сделать с помощью EC2 — это запустить экземпляр. В результате этой операции EC2 динамически создает и настраивает сервер, после чего он становится вам доступен.

Экземпляр запускается из AMI, хранящегося в S3. Amazon предоставляет ряд готовых к использованию образов машин, помимо этого, вы можете купить и коммерческие образы. Когда вы достигнете определенного уровня уверенности в работе с EC2, вы сможете создавать и регистрировать ваши собственные образы машин.

Внимание!

Поскольку ваши образы машин хранятся в Amazon S3, вы не можете запустить новый экземпляр EC2, если недоступен сервис Amazon S3.

Чтобы просмотреть, какие образы машин вам доступны, дайте следующую команду:

```
ec2-describe-images -o amazon
```

Опция `-o amazon` дает указание команде искать все образы машин, предоставляемые Amazon своим клиентам. Когда вы только начинаете работать, самый простой вариант для начала заключается в использовании одного из этих стандартных образов или же стандартного образа от другого поставщика. После того как вы выполните индивидуальную настройку этих образов, вы сможете построить собственные образы и зарегистрировать их. Чтобы просмотреть список зарегистрированных вами образов, дайте следующую команду:

```
ec2-describe-images
```

Вывод этой команды может выглядеть примерно следующим образом:

```
IMAGE        ami-225fba4b        ec2-public-images/fedora-core4-apache-
mysqlv1.07.manifest.xml amazon available public i386 machine
```

Вторым элементом вывода команды является идентификатор образа (image ID). Это значение потребуется вам, чтобы ссылаться на образ во всех остальных контекстах, вне зависимости от того, чем вы пользуетесь — утилитами командной строки или API Web-сервисов.

Третий элемент вывода команды — это объект S3, который представляет собой файл манифеста образа. Файл манифеста (manifest file) описывает образ машины для EC2. В рассматриваемом примере `ec2-public-images` — это имя корзины, в которой хранится образ, а `fedora-core4-apache-mysql-v1.07.manifest.xml` — это объект, представляющий файл манифеста образа.

Четвертый элемент — это владелец образа машины, в данном случае — Amazon. Пятый элемент описывает состояние образа. Шестой элемент указывает, является ли образ публичным и общедоступным, или же частным.

Седьмой элемент, `i386`, идентифицирует целевую процессорную архитектуру образа. На данный момент Amazon поддерживает две архитектуры: 32-разрядную архитектуру Intel (`i386`) и 64-разрядную архитектуру Intel (`x86_64`).

Наконец, последний элемент идентифицирует тип образа: машина, электронный диск (ramdisk) или ядро (kernel). На текущий момент сконцентрируем основное внимание на образах машин.

Выбор AMI

Изначально выбор AMI в EC2 представлял собой очень простую задачу. Но сегодня появилось довольно большое количество образов, как предоставляемых Amazon, так и построенных сторонними поставщиками, поэтому выбор нужного образа, особенно для новичка, стал делом не таким уж простым. Часто сложно сказать, какие из AMI являются достойными внимания, а какие представляют собой откровенный мусор, не говоря уже о том, что некоторые из них содержат вредоносные "трояны" или даже заранее предусмотренные "лазейки" (backdoors). К счастью, Amazon предоставляет каталог, в котором известные образы снабжаются аннотациями и рейтингами. Каталог этот доступен по следующему адресу: **http://aws.amazon.com/amis**. Обычно лучше всего начать с вашего любимого дистрибутива Linux (хотя на сегодняшний день доступно большое количество других вариаций UNIX и даже Windows). Просмотрите каталог и выберите какого-нибудь уважаемого пользователя, собравшего надежный, качественный дистрибутив в духе минимализма, собранный для предпочитаемой вами архитектуры, и воспользуйтесь этим

дистрибутивом в качестве базы. Лично я предпочитаю сборки Ubuntu, созданные Эриком Хэммондом (Eric Hammond).

Каким бы ни был ваш выбор, я рекомендую начать с чего-нибудь максимально простого, а затем по мере необходимости устанавливать только те утилиты и сервисы, которые вам абсолютно необходимы для поддержки вашего Web-приложения. Прежде, чем возиться с инсталляциями, обезопасьте ваш сервер с помощью какого-нибудь средства защиты наподобие Bastille (**http://www.bastille-unix.org**), а затем, закончив с инсталляциями приложений, вернитесь еще раз к настройкам безопасности. Более подробно о защите ваших экземпляров в облаке Amazon будет рассказано в *главе 5*.

После того как вы выберете подходящий образ машины, его нужно будет использовать для запуска экземпляра виртуального сервера. Это делается с помощью следующей команды:

```
ec2-run-instances AMI_ID
```

Например:

```
$ ec2-run-instances ami-1fd73376
```

В результате выполнения этой команды вы получите либо сообщение об ошибке, либо информацию о резервации ресурсов (reservation) для вашего нового экземпляра:

```
RESERVATION r-3d01de54 1234567890123 default
INSTANCE i-b1a21bd8 ami-1fd73376 pending 0
     m1.small 2008-10-22T16:10:38+0000 us-east-1a aki-a72cf9ce
     ari-a52cf9cc
```

Различие между резервацией (reservation) и экземпляром (instance) не столь очевидно, как в этом простейшем примере, где я демонстрирую запуск единственного нового экземпляра. Когда вы попытаетесь запускать экземпляры, вы можете указать минимальное и максимальное количество экземпляров для запуска (иначе говоря, вы можете запустить не один, а множество экземпляров). Вывод команды сообщит вам информацию о резервации. В выводе команды будет указано, какое количество экземпляров может быть запущено с помощью этого запроса.

Например, вы можете запускать новые серверы приложений, чтобы распределить нагрузку на Web-сайт в часы максимальных пиковых нагрузок. Если вы хотите запустить 10 экземпляров, чтобы поддержать возросшие требования к ресурсам, вы можете столкнуться с ситуацией, когда ресурсы для всех 10 экземпляров не будут доступны одновременно в пределах одной и той же зоны доступности. Если вы запросили 10 экземпляров, но ресурсы для всех 10 виртуальных серверов выделить невозможно, команда завершится неудачей. Однако вы можете запросить ресурсы для 5 виртуальных серверов, указав в качестве максимального количества экземпляров число 10. Эта команда будет успешно выполняться до тех пор, пока доступны ресурсы как минимум для 5 экземпляров. Таким образом, резервация отображает успешно запущенные экземпляры.

К числу важных сведений об экземпляре, на которые следует обращать внимание, относятся:

- `i-b1a21bd8` — идентификатор экземпляра (instance ID). Другие команды и API Web-сервисов используют этот идентификатор для ссылок на данный экземпляр;

- `ami-1fd73376` — идентификатор образа машины того AMI, который использовался для запуска экземпляра;
- `pending` — текущий статус экземпляра. Статус экземпляра, кроме значения `pending` (ожидает) может иметь следующие значения: `running` (работает), `shutting down` (останавливается) и `terminated` (завершен);
- `m1.small` — тип экземпляра EC2, который вы запустили. Типы экземпляра указывают объем RAM, дискового пространства и ресурсы CPU, которые выделены вашему экземпляру. Тип `m1.small` соответствует самому простому экземпляру, потребляющему минимум ресурсов;
- `us-east-1a` — зона доступности, в которой запущен экземпляр. Поскольку в командной строке зона доступности указана не была, сервис EC2 по умолчанию выбирает наилучшую зону доступности для запуска вашего экземпляра, осуществляя выбор на основании текущей загрузки. Чуть далее в этой главе я более подробно остановлюсь на зонах доступности.

На данном этапе вам следует немного подождать. Время ожидания того момента, когда ваш экземпляр станет доступным, может составить до 10 минут. В течение этого промежутка ожидания Amazon выделяет вашему экземпляру ресурсы и загружает его. Проверить статус экземпляра можно с помощью команды `ec2-describe-instances`:

```
ec2-describe-instances [INSTANCE_ID]
```

Вот пример вывода этой команды:

```
$ ec2-describe-instances i-b1a21bd8
RESERVATION r-3d01de54 1234567890123 default
INSTANCE i-b1a21bd8 ami-1fd73376 ec2-75-101-201-11.compute-1.amazonaws.com
     domU-12-31-38-00-9D-44.compute-1.internal running 0
     m1.small 2008-08-11T14:39:09+0000 us-east-1c aki-a72cf9ce
     ari-a52cf9cc
```

Приведенный пример вывода показывает, что ваш экземпляр уже запустился и теперь имеет статус `running`. Иначе говоря, теперь он доступен, но остается одна небольшая проблема: у вас нет учетной пользовательской записи на новой виртуальной машине.

Прежде чем я расскажу вам о том, как получить на запущенной машине учетную запись с доступом к оболочке (shell account)[1], давайте посмотрим, какую еще информацию вы можете получить о работающем экземпляре.

Вывод команды `ec2-describe-instances` практически идентичен выводу команды `ec2-run-instances`, с тем исключением, что `ec2-describe-instances` предоставляет вам публичный IP-адрес и частный IP-адрес экземпляра, потому что

[1] Shell account — в UNIX и Linux это учетная запись пользователя с возможностью работать через оболочку (shell). Чтобы избежать излишнего многословия, здесь и далее предлагается использовать термин shell-аккаунт или даже просто "шелл". Хотя некоторые ревнители чистоты русского языка и морщат нос, услышав этот "жаргон", но данный термин отлично прижился. Все, кто работает с UNIX и Linux, отлично знают, что это такое, и сам термин намного лаконичнее, чем "учетная запись с возможностью работы через оболочку". — *Прим. перев.*

теперь экземпляр находится в работающем состоянии (`running`). Когда экземпляр запускается, Amazon динамически присваивает ему один публичный и один частный IP-адрес. Для доступа к экземпляру из точки, находящейся вне облака, следует использовать публичный IP-адрес, а частный IP-адрес используется для доступа к экземпляру изнутри облака.

Доступ к экземпляру

Как я уже упомянул, вы не сможете получить доступ к вновь запущенному экземпляру EC2 до тех пор, пока не создадите на нем учетную запись для себя. Поэтому остановите экземпляр и запустите другой экземпляр, к которому у вас будет доступ:

```
$ ec2-terminate-instances i-b1a21bd8
```

Поскольку вы используете предварительно подготовленный экземпляр машины, то у вас, очевидно, не будет никаких пользовательских учетных записей на этой машине на тот момент, когда запускается ее образ. Более того, для Amazon (да и для кого угодно другого) это была бы страшнейшая брешь в системе безопасности, если бы в образах машин имелись стандартные учетные записи с легко угадываемыми паролями наподобие "scott"/"tiger".

Фокус с получением доступа заключается в генерации пары ключей (keypair)[1] SSH[2]. Секретный ключ (private key) будет находиться на вашем локальном жестком диске, в то время как открытый ключ (public key) передается экземпляру при его запуске. EC2 сконфигурирует экземпляр таким образом, чтобы к учетной записи `root` можно было получить доступ с помощью вашего секретного ключа.

Прежде чем запускать новый экземпляр, вам необходимо сгенерировать пару ключей:

```
ec2-add-keypair KEYPAIR_NAME
```

Например:

```
$ ec2-add-keypair mykeypair
KEYPAIR mykeypair 1f:51:ae:28:bf:89:e9:d8:1f:25:5d:37:2d:7d:b8:ca:9f:f5:f1:6f
-----BEGIN RSA PRIVATE KEY----MIIEoQIBAAKCAQBuLFg5ujHrtm1jnutSuoO8Xe56LlT+
HM8v/xkaa39EstM3/aFxTHgElQiJLChp
HungXQ29VTc8rc1bW0lkdi23OH5eqkMHGhvEwqa0HWASUMll4o3o/IX+0f2UcPoKCOVUR+jx71Sg
5AU52EQfanIn3ZQ8lFW7Edp5a3q4DhjGlUKToHVbicL5E+g45zfB95wIyywWZfeW/UUF3LpGZyq/
ebIUlq1qTbHkLbCC2r7RTn8vpQWp47BGVYGtGSBMpTRP5hnbzzuqj3itkiLHjU39S2sJCJ0TrJx5
```

[1] Подробнее о криптографических системах с открытым ключом, подобных применяемой здесь, можно прочитать по следующим адресам: **http://tinyurl.com/dbhe3b**, **http://en.wikipedia.org/wiki/Public-key_cryptography**. — *Прим. перев.*

[2] SSH (Secure SHell) — "безопасная оболочка", сетевой протокол сеансового уровня, позволяющий осуществлять удаленное управление операционной системой и туннелирование TCP-соединений. SSH позволяет безопасно передавать в незащищенной среде практически любой другой сетевой протокол. Подробнее см. **http://en.wikipedia.org/wiki/Secure_Shell**, **http://www.openssh.ru/base/sec/ssh_intro.txt.html**. — *Прим. перев.*

```
i8BygR4s3mHKBj8l+ePQxG1kGbF6R4yg6sECmXn17MRQVXODNHZbAgMBAAECggEAY1tsiUsIwDl5
91CXirkYGuVfLyLflXenxfI50mDFms/mumTqloHO7tr0oriHDR5K7wMcY/YY5YkcXNo7mvUVD1pM
ZNUJs7rw9gZRTrf7LylaJ58kOcyajw8TsC4e4LPbFaHwS1d6K8rXh64o6WgW4SrsB6ICmr1kGQI7
3wcfgt5ecIu4TZf0OE9IHjn+2eRlsrjBdeORi7KiUNC/pAG23I6MdDOFEQRcCSigCj+4/mciFUSA
SWS4dMbrpb9FNSIcf9dcLxVM7/6KxgJNfZc9XWzUw77Jg8x92Zd0fVhHOux5IZC+UvSKWB4dyfcI
tE8C3p9bbU9VGyY5vLCAiIb4qQKBgQDLiO24GXrIkswF32YtBBMuVgLGCwU9h9HlO9mKAc2m8Cm1
jUE5IpzRjTedc9I2qiIMUTwtgnw42auSCzbUeYMURPtDqyQ7p6AjMujp9EPemcSVOK9vXYL0Ptco
xW9MC0dtV6iPkCN7gOqiZXPRKaFbWADp16p8UAIvS/a5XXk5jwKBgQCKkpHi2EISh1uRkhxljyWC
iDCiK6JBRsMvpLbc0v5dKwP5alo1fmdR5PJaV2qvZSj5CYNpMAy1/EDNTY5OSIJU+0KFmQbyhsbm
rdLNLDL4+TcnT7c62/aH01ohYaf/VCbRhtLlBfqGoQc7+sAc8vmKkesnF7CqCEKDyF/dhrxYdQKB
gC0iZzzNAapayz1+JcVTwwEid6j9JqNXbBc+Z2YwMi+T0Fv/P/hwkX/ypeOXnIUcw0Ih/YtGBVAC
DQbsz7LcY1HqXiHKYNWNvXgwwO+oiChjxvEkSdsTTIfnK4VSCvU9BxDbQHjdiNDJbL6oar92UN7V
rBYvChJZF7LvUH4YmVpHAoGAbZ2X7XvoeEO+uZ58/BGKOIGHByHBDiXtzMhdJr15HTYjxK7OgTZm
gK+8zp4L9IbvLGDMJO8vft32XPEWuvI8twCzFH+CsWLQADZMZKSsBasOZ/h1FwhdMgCMcY+Qlzd4
JZKjTSu3i7vhvx6RzdSedXEMNTZWN4qlIx3kR5aHcukCgYA9T+Zrvm1F0seQPbLknn7EqhXIjBaT
P8TTvW/6bdPi23ExzxZn7KOdrfclYRph1LHMpAONv/x2xALIf91UB+v5ohy1oDoasL0gij1houRe
2ERKKdwz0ZL9SWq6VTdhr/5G994CK72fy5WhyERbDjUIdHaK3M849JJuf8cSrvSb4g==
-----END RSA PRIVATE KEY----
```

Выполнив эту команду, вы получите только ваш секретный ключ. Открытого ключа, который будет добавлен к вашей учетной записи Amazon Web Services, чтобы использоваться всякий раз, когда вы запускаете экземпляры с применением этой пары ключей, вы никогда не увидите.

Скопируйте секретный ключ и сохраните все, что начинается со строки `-----BEGIN RSA PRIVATE KEY-----` и заканчивается строкой `-----END RSA PRIVATE KEY----`, в отдельный файл. Как вы назовете этот файл и куда вы его поместите, существенной роли не играет, но часто бывает очень полезно включать имя пары ключей в имя файла, например: `id-rsa_mykeypair`. Кроме того, вам потребуется указать права доступа к этому файлу таким образом, чтобы прочесть его могли только вы. В UNIX-подобных системах, например, таких как Linux и Mac OS X, это можно сделать так:

```
$ chmod 400 id_rsa-mykeypair
```

Теперь вы можете запустить новый экземпляр, который ссылается на только что созданную пару ключей:

```
$ ec2-run-instances -k mykeypair ami-1fd73376
RESERVATION r-8a01de64 1234567890123 default
INSTANCE i-a7d32cc3 ami-1fd73376 pending
mykeypair m1.small 2008-10-22T16:40:38+0000
us-east-1a aki-a72cf9ce ari-a52cf9cc
```

Обратите внимание, что опция командной строки `-k` ссылается на имя вашей пары ключей, а не на имя файла, в котором вы сохранили свой секретный ключ.

Вывод этой команды выглядит практически так же, как и в предыдущем случае, за исключением того, что теперь вы получаете новый идентификатор резервации (reservation ID) и новый идентификатор экземпляра (instance ID), а `0` из вывода предыдущей команды замещен строкой `mykeypair`, что указывает на то, что экзем-

пляр был запущен с открытым ключом, входящим в состав созданной вами пары ключей.

Открытый ключ присваивается учетной записи `root` нового экземпляра. Иными словами, вы теоретически можете воспользоваться SSH, чтобы зарегистрироваться в системе как `root`, не вводя никакой регистрационной информации. Это безопасно, потому что ваш секретный ключ хранится на вашем локальном компьютере, который соответствует открытому ключу. Однако на практике вы пока все равно не сможете получить доступ к экземпляру, потому что правила, установленные по умолчанию для любой из групп безопасности EC2, запрещают доступ через сеть к экземплярам, запущенным в этой группе безопасности.

Примечание

На данном этапе те, кто привык к проверенной временем практике защиты операционной системы путем блокировки учетной записи `root`, могут быть готовы просто выпрыгнуть из окна от негодования. Не беспокойтесь, механизмы устранения необходимости хранения каких-либо учетных записей в вашем AMI, которые позволят вам блокировать регистрацию от имени `root`, будут описаны в *главе 5*.

Группы безопасности

Последний элемент вывода команды `ec2-describe-instances`, которая описывает резервацию ресурсов и которая по умолчанию до сих пор использовалась во всех примерах этой главы, указывает группу безопасности, в которой был запущен ваш экземпляр.

Если посмотреть на группу безопасности с практической точки зрения, то ее ближайшим аналогом может послужить сетевой сегмент, защищенный брандмауэром. При создании брандмауэр запускается, имея всего лишь одно установленное правило: блокировать весь входящий трафик из всех источников во все пункты назначения через все порты.

По целому ряду причин эта аналогия воистину ужасна. Однако для целей данной главы это единственная работающая аналогия, позволяющая дать простое и понятное объяснение, поэтому именно ею мы и будем пока пользоваться.

Почему группа безопасности не является сетевым сегментом

В этой главе при описании групп безопасности я буду использовать аналогию между группой безопасности и сетевым сегментом, который защищен брандмауэром. Хотя мы и будем пока пользоваться этой аналогией, вы должны сразу же уяснить себе, что не стоит путать группу безопасности с сетевым сегментом. Приведу некоторые из отличительных особенностей группы безопасности, из-за которых ее не следует считать сетевым сегментом.

- Экземпляры, запущенные в одной и той же группе безопасности, не используют совместно один и тот же блок IP-адресов, ни публичных, ни частных. Фактически они могут даже существовать в разных зонах доступности. В результате этого вы не мо-

жете работать с приложениями, которые полагаются на многоадресный (multicast) или широковещательный (broadcast) трафик.

- Хотя вы можете перевести свой сервер в так называемый "неразборчивый" режим (promiscuous mode)[1], вы сможете наблюдать только тот сетевой трафик, где пунктом назначения является ваш сервер. Эта функция существенно повышает защищенность EC2, но одновременно она и затрудняет работу систем обнаружения вторжений в сеть (network intrusion detection systems).
- Правила вашего "брандмауэра" просто указывают, каким образом разрешено маршрутизировать трафик в вашу группу. Эти правила не обеспечивают никакой антивирусной защиты или фильтрации на базе контента.
- Экземпляры, расположенные в той же группе безопасности, не могут поддерживать коммуникации друг с другом до тех пор, пока вы не создадите правило, которое это разрешает.
- Как только экземпляр будет запущен, вы не сможете изменить его членство в группе безопасности.

Практический смысл сказанного заключается в том, что вы не можете получить доступ к экземпляру в новой группе безопасности до тех пор, пока вы явно не предоставите доступ с помощью следующей команды `ec2-authorize`:

```
ec2-authorize GROUP_NAME [OPTIONS]
```

Чтобы получить доступ к вашему новому экземпляру через командную оболочку, вам потребуется открыть порт SSH:

```
$ ec2-authorize default -p 22
PERMISSION default ALLOWS tcp 22 22 FROM CIDR 0.0.0.0/0
```

Если вы хотите создать Web-сервер, вы, естественно, захотите предоставить к нему доступ через HTTP и HTTPS:

```
$ ec2-authorize default -p 80
PERMISSION default ALLOWS tcp 80 80 FROM CIDR 0.0.0.0/0
$ ec2-authorize default -p 443
PERMISSION default ALLOWS tcp 443 443 FROM CIDR 0.0.0.0/0
```

Вывод этой команды показывает, что для группы безопасности по умолчанию теперь определен доступ через указанные порты из любой точки в Интернете (запись `FROM CIDR 0.0.0.0/0` означает "любой IP-адрес").

Внимание!

В приведенном примере я открыл доступ через SSH с любого IP-адреса, но это вовсе не означает, что я считаю это очень хорошей идеей. В общем случае, если это возможно, все правила должны определять конкретный источник. Очевидно, HTTP и HTTPS открыты для всего мира, но практически все остальное должно быть полностью закрыто или предоставлено в доступ избирательно.

[1] Promiscuous mode или promisc mode — так называемый "неразборчивый" режим, в котором сетевая карта позволяет принимать все пакеты независимо от того, кому они адресованы. Подробнее см. **http://en.wikipedia.org/wiki/Promiscuous_mode**. — *Прим. перев.*

Чтобы разрешить доступ к конкретной подсети для входящего трафика, вам необходимо указать исходный IP-адрес (с помощью опции `-s`) или адрес подсети, из которой вы и ваши разработчики смогут получать доступ к работающему экземпляру, например:

```
$ ec2-authorize default -P tcp -p 22 -s 10.0.0.1/32
```

Эта команда разрешит доступ по протоколу TCP через порт 22 только из 10.0.0.1. Теперь вы сможете получить доступ к серверу через SSH:

```
ssh -i PRIVATE_KEY_FILE root@PUBLIC_IP
```

В данном случае вы можете ввести:

```
$ ssh -i id_rsa-mykeypair root@ec2-75-101-201-11.compute-1.
amazonaws.com
```

После этого вы зарегистрируетесь в системе вашего нового экземпляра от имени `root`.

Вы совершенно не обязательно должны быть ограничены только одной группой безопасности. Например, вы можете создать дополнительные группы безопасности с помощью команды `ec2-add-group`:

```
ec2-add-group GROUP -d DESCRIPTION
```

Например:

```
$ ec2-add-group mygroup -d MyGroup
GROUP mygroup MyGroup
```

За счет использования множества групп вы можете определять разные наборы правил для различных типов экземпляров — например открыть доступ через HTTP/HTTPS к группе балансировщика нагрузки, одновременно закрыв любой доступ к группе серверов приложений, за исключением доступа к Tomcat *mod_jk*[1] из группы балансировщика нагрузки.

Настройку групп безопасности с целью обеспечения надежной защиты Web-приложений мы будем рассматривать в *главе 5*.

Зоны доступности

Одна из концепций, а именно, зоны доступности (availability zones), до сих пор пока была обойдена молчанием. Зона доступности примерно представляет собой аналог физического центра обработки данных. На момент написания данных строк компания Amazon предоставляла три зоны доступности в США и две — в Западной

[1] *mod_jk* — это модуль-коннектор, используемый для подключения контейнера сервлетов Tomcat к Web-серверам, таким как Apache, с помощью протокола Apache JServ Protocol (AJP), который проводит входящие запросы с Web-сервера до сервера приложений, который в сетевой архитектуре находится за Web-сервером.
Подробнее см. **http://en.wikipedia.org/wiki/Apache_JServ_Protocol**, **http://tomcat.apache.org/connectors-doc/ajp/ajpv13a.html**, **http://en.wikipedia.org/wiki/Mod_jk**, **http://tomcat.apache.org/connectors-doc/**. — *Прим. перев.*

Европе[1]. Наиболее важной чертой зоны доступности является то, что любые две зоны доступности имеют различные физические инфраструктуры. Таким образом, сбой зоны доступности или некоторой ее части не влияет на остальные зоны доступности, по крайней мере, до тех пор, пока другие зоны доступности включаются в поддержку систем, ранее работавших в зонах доступности, которые в данный момент испытывают сложности. Соглашение об уровне сервиса Amazon гарантирует доступность, по крайней мере, двух зон доступности в пределах заданного региона с вероятностью 99,95 %.

В ранее приведенных примерах использовалась зона доступности `us-east-1a`. Этот идентификатор не означает конкретного центра обработки данных; `us-east-1a` для вашей учетной записи Amazon, вероятно, будет иным, нежели для моей.

Понимание концепции зон доступности важно по двум причинам.

- Когда вы запускаете один экземпляр в одной зоне доступности, а второй — в другой, вы получаете определенный "запас прочности" за счет инфраструктурной избыточности. Это означает, что можно быть практически полностью уверенным в том, что, по крайней мере, один экземпляр останется в рабочем состоянии в случае отказа другого, причем независимо от причины сбоя.
- Вы оплачиваете трафик между двумя зонами доступности. Иначе говоря, если у вас главный сервер MySQL работает в одной зоне доступности, а подчиненный — в другой, то вы оплачиваете затраты на полосу пропускания для всех передач данных между главным и подчиненным серверами. Если же оба сервера находятся в одной зоне доступности, то за этот трафик вы не платите. С другой стороны, если главный и подчиненный серверы находятся в одной зоне доступности, то вы теряете преимущества повышенной отказоустойчивости от варианта настройки "главный/подчиненный".

Когда вы запускаете экземпляр, не указывая зоны доступности, Amazon выберет ее для вас самостоятельно. В общем случае, при запуске первого экземпляра лучше, если Amazon сделает этот выбор за вас, потому что выбор будет сделан в пользу зоны доступности с наибольшим объемом ресурсов. По мере того, как вы будете добавлять экземпляры в свою инфраструктуру, вы, скорее всего, захотите самостоятельно указывать зоны доступности с тем, чтобы повысить избыточность там, где она критична, и снизить затраты на трафик в тех ситуациях, когда избыточность не имеет такого уж важного значения.

Статические IP-адреса

Каждый раз, когда вы запускаете в EC2 новый экземпляр, Amazon динамически присваивает ему публичный и частный IP-адреса. Далее в этой книге я опишу методы, позволяющие снизить потребность в статических IP-адресах, однако полно-

[1] Впрочем, 29 апреля 2010 года компания Amazon открыла еще одну зону доступности — в Сингапуре. См. **http://www.networkworld.com/news/2010/042910-amazon-web-services-opens-first.html**. — *Прим. перев.*

стью ликвидировать эту потребность невозможно. Например, вам практически наверняка потребуется, чтобы Web-сайт имел фиксированный IP-адрес, поставленный в соответствие вашему DNS-серверу. Amazon обеспечивает эту потребность за счет эластичных IP-адресов (elastic IP addresses).

При создании новой учетной записи Amazon вы получаете возможность создать пять эластичных (статических) IP-адресов. Плата за эти IP-адреса с вас взиматься не будет до тех пор, пока вы фактически их не создадите. Даже и в этом случае вы платите только за их создание без назначения их работающему экземпляру EC2 или постоянного их переназначения. Эластичные IP-адреса — это одна из немногих вещей, за которые вы должны платить даже в течение того времени, когда вы их не используете.

Примечание

Будьте добропорядочным интернет-гражданином. Даже несмотря на то, что плата за IP-адреса не опустошит ваш банковский счет, назначайте статический IP-адрес только тогда, когда вы точно уверены в том, что он вам действительно нужен. Запас доступных IP-адресов в Интернете недостаточен, и эта глобальная проблема обостряется на локальном уровне, потому что каждой организации выделяется фиксированное количество IP-адресов. Поэтому каждый раз, когда пользователи получают выделенный IP-адрес, который не ассоциирован с активным экземпляром, это налагает на систему дополнительную нагрузку. И не надо считать, что IPv6 решит эту проблему, если только у вас нет желания затратить достаточно продолжительное время на изучение новой парадигмы, переустановку новых версий программного обеспечения и их конфигурирование "с нуля". Причем на данный момент Amazon не поддерживает внешней маршрутизации адресов IPv6.

Чтобы выделить новый эластичный IP-адрес, применяйте следующую команду:

```
$ ec2-allocate-address
ADDRESS 67.202.55.255
```

Теперь этот IP-адрес станет "вашим", и вы можете присвоить его экземпляру EC2:

```
$ ec2-associate-address -i i-a7d32cc3 67.202.55.255
ADDRESS 67.202.55.255 i-a7d32cc3
```

Наконец, вы можете вывести список всех ваших выделенных IP-адресов:

```
$ ec2-describe-addresses
ADDRESS 67.202.55.255 i-a7d32cc3
ADDRESS 75.101.133.255 i-ba844bc3
```

Когда вы присваиваете эластичный IP-адрес экземпляру, старый публичный IP-адрес освобождается и заменяется вашим выделенным эластичным IP-адресом. Если этот экземпляр по какой-либо причине теряется, вы можете запустить новый экземпляр и присвоить ему эластичный IP-адрес старого экземпляра. За счет этого вы можете восстановить публичный доступ к вашей системе посредством единственного статического IP-адреса.

Частный IP-адрес для экземпляра всегда остается динамическим адресом, присвоенным ему при запуске.

Хранение данных в EC2

Amazon предоставляет три различных метода хранения данных в своей облачной инфраструктуре, а именно:

- постоянное облачное хранилище (Persistent cloud storage);
- эфемерное хранилище экземпляра (Ephemeral instance storage);
- эластичное блочное хранилище (Elastic block storage).

Amazon S3 — это механизм, посредством которого Amazon предоставляет *постоянное облачное хранилище*.

Когда вы запускаете новый экземпляр EC2, он получает свое *эфемерное хранилище экземпляра*, срок жизни которого соответствует сроку жизни экземпляра, который оно поддерживает. Иными словами, если вы теряете этот экземпляр или завершаете его работу, вы теряете и все, что содержалось в эфемерном хранилище этого экземпляра.

Еще один вариант облачного хранилища Amazon — *эластичное блочное хранилище* (elastic block storage, EBS). EBS в принципе представляет собой сетевое устройство блочного хранения данных, аналогом которого в физической инфраструктуре является SAN. Вы имеете возможность создавать тома различных размеров, варьирующихся от 1 Гбайт до 1 Ебайт и монтировать любое количество томов к экземпляру EC2. Однако вы не можете одновременно использовать один и тот же том, разделяя его между двумя экземплярами EC2.

Скорее всего ваша немедленная, "инстинктивная" реакция будет такой: "Я хочу воспользоваться третьим вариантом! Мне нужно блочное хранилище!" И да, блочное хранилище, совершенно определенно, представляет собой очень хороший вариант. Но в хорошо проработанной облачной инфраструктуре вы вполне можете организовать интеллектуальное использование всех видов хранилищ, каждое из которых будет наилучшим образом приспособлено к конкретным обстоятельствам и требованиям. В табл. 2.1 перечислены различные возможности и характеристики всех видов хранилищ, их преимущества и недостатки.

Таблица 2.1. *Сравнительные характеристики опций хранилищ данных в EC2*

	Amazon S3	**Экземпляр**	**Блочное хранилище**
Скорость	Низкая	Непредсказуемая	Высокая
Надежность	Умеренная	Высокая	Высокая
Долговечность	Сверхвысокая	Сверхнизкая	Высокая
Гибкость	Низкая	Умеренная	Высокая
Сложность	Высокая	Низкая	Высокая
Стоимость	Умеренная	Низкая	Высокая

Таблица 2.1 *(окончание)*

	Amazon S3	**Экземпляр**	**Блочное хранилище**
Сильная сторона	Управление аварийным восстановлением данных	Динамические данные	Оперативные данные
Слабая сторона	Оперативные данные	Нединамические данные	Большое количество мелких операций ввода/вывода

Надежность и долговечность S3

S3 имеет умеренную надежность, но супервысокую долговечность, потому что представляет собой одновременно наиболее долговечный, но наименее надежный из всех вариантов. Когда вы помещаете данные в S3, вы знаете, что они там останутся и завтра, и послезавтра, и еще через день. Никаких потерь данных, никакой порчи данных. На самом деле, то, что происходит в облаке, даже не имеет значения. Но проблема состоит в том, что S3 имеет очень слабую "кривую показателей", когда речь заходит о доступности. В прошлом году мне пришлось пережить целый день, не имея доступа к S3, а также столкнуться с рядом многочасовых периодов недоступности сервиса. Однако, насколько мне известно, в S3 никогда не теряются никакие данные, и общая архитектура этого сервиса такова, что вероятность потери данных крайне мала. Ирония ситуации в том, что до последнего времени S3 был единственным сервисом, который компания Amazon предлагала с гарантированным уровнем обслуживания.

Еще одна проблема заключается в непредсказуемости уровня производительности для хранилища экземпляра. Вы можете думать, что этот вариант окажется быстрее других, и в некоторых случаях это действительно будет так. Однако иногда данная опция оказывается невероятно медленной — еще даже медленнее, чем монтирование NFS через медленное соединение Ethernet. С другой стороны, EBS постоянно предоставляет производительность SAN через гигабитный Ethernet.

Настройка тома EBS

Чтобы начать работу с эластичным блочным хранилищем, необходимо в первую очередь создать том EBS:

```
ec2-create-volume --size SIZE_GB -z ZONE
```

Эта команда создает блочный том указанного размера в указанной зоне доступности.

Одним из ключевых требований к тому блочного хранения является то, что он должен располагаться в той же самой зоне доступности, что и ваш экземпляр EC2 — монтировать тома, расположенные в других зонах доступности, невозможно. Следовательно, вам всегда необходимо указывать целевую зону доступности при создании блочного тома, например:

```
$ ec2-create-volume --size 10 -z us-east-1a
VOLUME vol-9a773124 800 creating 2008-10-20T18:21:03+0000
```

Затем вам потребуется некоторое время подождать, пока том станет доступным, после чего вы сможете назначить его экземпляру EC2. Проверить доступность томов можно с помощью команды `ec2-describe-volumes`, например так:

```
$ ec2-describe-volumes vol-9a773124
VOLUME vol-9a773124 800 available 2008-10-20T18:21:03+0000
```

С этого момента Amazon начнет взимать с вас плату за том, хотя он еще не используется ни одним экземпляром. Чтобы дать экземпляру возможность использования тома, его нужно назначить конкретному экземпляру, находящемуся в пределах той же зоны доступности:

```
$ ec2-attach-volume vol-9a773124 -i i-a7d32cc3 -d /dev/sdh
ATTACHMENT vol-9a773124 i-a7d32cc3 /dev/sdh attaching 2008-10-20T18:23:27+0000
```

Эта команда сообщает EC2 о необходимости подключить ваш вновь созданный том к указанному экземпляру и назначить тому устройство `/dev/sdh` (имя устройства, выбранное мною, соответствует общей схеме именования устройств в UNIX/Linux, а сокращение `sdh` представляет собой имя, которое в Linux обычно назначается устройствам SCSI). Если экземпляр находится в другой зоне доступности или уже используется другим экземпляром, вы получите сообщение об ошибке. В противном случае на данном этапе вы получите Raw-устройство[1], подключенное к вашему экземпляру, которое вы можете примонтировать и отформатировать по вашему усмотрению.

Наиболее распространенный метод сделать устройство пригодным для использования в Linux заключается в том, чтобы отформатировать новый том для использования файловой системы ext3 (я предпочитают XFS[2], поскольку в этом случае вы можете "заморозить" файловую систему), а затем примонтировать его. Чтобы выполнить эти базовые задачи, зарегистрируйтесь на вашем экземпляре через SSH и выполните следующие команды:

```
$ mkdir /mnt/myvolume
$ yes | mkfs -t ext3 /dev/sdh
$ mount /dev/sdh /mnt/myvolume
```

Теперь в вашем распоряжении окажется том объемом 10 Гбайт, доступный для использования экземпляром, которому вы его назначили. Вы сможете использовать его точно так же, как пользуетесь любым другим томом, отформатированным под ext3.

---

[1] Raw-устройства (Raw devices, "сырые устройства") — это специальный тип устройств, предназначенных для организации доступа к блочным устройствам без кэширования. Обычно используются системами, которые сами "знают", как им кэшировать данные, и не хотят полагаться в этом на операционную систему.
Подробнее см. **http://tinyurl.com/247tzyy**, **http://en.wikipedia.org/wiki/Raw_device**. — *Прим. перев.*

[2] XFS — высокопроизводительная файловая система с ведением журнала, созданная компанией Silicon Graphics (**http://www.sgi.com/**) для их собственной операционной системы IRIX. Подробнее см. **http://en.wikipedia.org/wiki/XFS**, **http://xfs.org/index.php/Main_Page**. — *Прим. перев.*

Управление томами

Как и в случае с любым другим типом устройства, том EBS может оказаться поврежденным, если его отключение произведено некорректно.

ВНИМАНИЕ!

Перед тем как отключить том EBS, его всегда следует предварительно отмонтировать. Если на этом томе работает база данных, никогда не следует забывать предварительно остановить сервер базы данных прежде, чем отмонтировать том.

В процессе останова операционной системы ваш экземпляр всегда должен корректно отмонтировать тома. Однако если вы намерены просто отключить том, не останавливая операционной системы, вам необходимо сначала вручную отмонтировать его (и синхронизация тома тоже не повредит):

```
$ umount /mnt/myvolume
```

После этого вы сможете безопасно отключить том от экземпляра с тем, чтобы подключить к другому экземпляру:

```
$ ec2-detach-volume vol-9a773124 -i i-a7d32cc3
ATTACHMENT vol-9a773124 i-a7d32cc3 /dev/sdh detaching 2008-10-20T18:55:17+0000
```

После того как это будет выполнено, вы сможете подключить том к другому экземпляру.

Моментальные снимки

Способность создания моментального снимка (snapshot) тома — это особенно привлекательная черта блочного хранилища Amazon. Вы можете создавать моментальные снимки ваших томов так часто, как вам это требуется. EC2 автоматически сохраняет эти моментальные снимки в S3, благодаря чему вы получаете очень мощную и быструю систему резервного копирования.

ВНИМАНИЕ!

Хотя моментальные снимки EBS представляют собой особенно мощный механизм резервного копирования, имейте в виду, что они абсолютно не допускают переноса. Вы не имеете возможности "вывести" свои моментальные снимки EBS за пределы облака Amazon. И даже если бы вы смогли это сделать, они были бы для вас практически бесполезны. В *главе 6* я подробнее расскажу о подходах к использованию функции снятия моментальных снимков EBS при разработке стратегии создания переносимых резервных копий.

Моментальные снимки создаются с помощью команды `ec2-create-snapshot`. Однако прежде, чем вы создадите моментальный снимок, вам необходимо убедиться в том, что тот том, с которого вы будете его создавать, находится в непротиворечивом состоянии. Иными словами, вам потребуется остановить все операции записи на этот том. Каким образом вы будете это делать, зависит от того, какие приложения выполняют запись на данный том.

С точки зрения непротиворечивости файловой системы тома самую большую проблему представляют собой любые базы данных, хранящиеся на этом томе. Перед тем, как создавать моментальный снимок, абсолютно необходимо остановить все операции, производимые с базами данных.

Как заблокировать MySQL перед созданием моментального снимка

В случае, если вы работаете с MySQL, вы можете либо установить блокировку на весь движок базы данных или перевести базу данных в режим только для чтения. Самый безопасный подход заключается в установке блокировки. Из командной строки MySQL можно выполнить следующую команду:

```
FLUSH TABLES WITH READ LOCK
```

Оставьте командную строку открытой, заблокируйте файловую систему и затем создайте моментальный снимок. Если вы пользуетесь Linux, то лучшей файловой системой для использования на томах EBS является XFS, благодаря тому, что существует команда `xfs_freeze`.

Как только моментальный снимок будет создан, вы можете закрыть клиентское приложение и снять блокировку. Вам потребуется дождаться только выхода команды `ec2-create-snapshot` — дожидаться завершения создания моментального снимка нет необходимости.

Практически любая база данных, поддерживающая "теплое" резервирование (warm backups), может быть хорошо приспособлена к стратегии создания моментальных снимков EBS.

Чтобы создать моментальный снимок, дайте следующую команду:

```
$ ec2-create-snapshot vol-9a773124
SNAPSHOT snap-21ab4c89b vol-9a773124 pending 2008-10-20T19:02:18+0000
```

Как только вы создадите моментальный снимок, вы можете сразу же возобновить запись на том; вам не нужно дожидаться момента, когда завершится создание моментального снимка. Полный процесс создания моментального снимка потребует некоторого времени и завершится в фоновом режиме, поэтому новый моментальный снимок не станет доступным для использования сразу же. Чтобы определить, когда моментальный снимок станет доступным для использования, дайте команду `ec2-describe-snapshots`:

```
$ ec2-describe-snapshots snap-21ab4c89b
SNAPSHOT snap-21ab4c89b vol-9a773124 pending 2008-10-20T19:02:33+
0000 20%
```

Когда процесс создания моментального снимка завершится, поле статуса изменит свое значение с `pending` (ожидание) на `completed` (завершено), а в поле процента завершенности появится значение `100%`.

Моментальный снимок сам по себе не является пригодным к использованию томом, но вы можете получить доступ к его данным для создания на его основе новых томов:

```
$ ec2-create-volume --snapshot snap-21ab4c89b -z us-east-1a
VOLUME vol-13b692332a 800 creating 2008-02-15T19:11:36+0000
```

После этого вы получите предварительно отформатированный том, на который будут перенесены данные со старого тома, и этот новый том будет готов к использованию с новым экземпляром EC2. На основе одного и того же моментального снимка можно создать любое количество томов.

Функция создания моментальных снимков предоставляет вам множество удобных и мощных возможностей, в том числе:

- возможность быстрого создания дубликатов вашей среды для тестирования системы, тестирования нагрузки или для других ситуаций, когда вам требуется точная копия вашей производственной среды;
- возможность резервного копирования с минимальным влиянием на вашу производственную среду.

Еще одной важной особенностью моментальных снимков является то, что они инкрементны. Иначе говоря, если вы создаете моментальный снимок тома объемом 20 Гбайт в 18:00, а затем повторяете этот процесс в 18:10, то вы получите две резервные копии, сохраненных в S3: первая из них будет содержать исходные 20 Гбайт, а вторая — изменения, записанные на том в интервале между 18:00 и 18:10. Инкрементные моментальные снимки позволяют постоянно резервировать ваши тома.

Управление AMI

Когда вы только приступаете к работе с AWS, вы начинаете с использования предварительно созданных и готовых к употреблению AMI, с которых загружаете новые экземпляры. Прежде чем создать реально рабочую среду, вам обычно требуется создать собственную библиотеку машинных образов (machine images) и научиться управлять ими. В последующих главах мы подробно обсудим различные стратегии управления образами машин, как для Amazon, так и для других облачных инфраструктур. Здесь же мы рассмотрим базовую процедуру, которую необходимо проделать, чтобы создать и зарегистрировать новый образ машины.

Образ машины Amazon (Amazon machine image, AMI) содержит корневую файловую систему для вашего экземпляра. В его эфемерном хранилище (в большинстве случаев это файлы, содержащиеся в каталоге `/mnt`) ничего нет. Чтобы построить AMI, вам необходимо скопировать ваш сертификат Amazon EC2 и секретный ключ (это два файла с расширением .pem, которые вы получили при создании и настройке вашей учетной записи Amazon). Поместите эти файлы в каталог `/mnt`, потому что вряд ли вы захотите, чтобы ваши ключи Amazon были встроены в ваш AMI. Возможно, вы предпочтете также очистить каталоги `/tmp` и `/var/tmp`, чтобы не тратить лишнего пространства в S3 на временные файлы. Наконец, если вы запускаете экземпляр базы данных в корневом каталоге вашего экземпляра, остановите базу данных.

Первая ваша задача заключается в комплектации вашего образа:

```
$ cd /mnt
$ sudo mkdir ami
$ sudo ec2-bundle-vol -d /mnt/ami -k /mnt/pk-ZZZ.pem \
       -c /mnt/cert-ZZZ.pem -u 1234567890123 -r i386 -p myami
```

Этот процесс довольно длителен, так что в течение некоторого времени вам будет казаться, что ничего не происходит. Команда скомплектует вашу корневую файловую систему, разобьет ее на небольшие фрагменты и сохранит в каталоге `/mnt/ami/myami`.

Как только этот процесс завершится, вы получите множество фрагментов вашего AMI, а также файл манифеста, который называется `/mnt/ami/myami/myami.manifest.xml`.

На этом шаге вам необходимо будет закачать ваш комплект, образующий AMI, в Amazon S3. Делается это так:

```
$ s3cmd mb s3://myami
$ sudo ec2-upload-bundle -b myami -m /mnt/ami/myami.manifest.xml \
       -a ACCESS_KEY -s SECRET_KEY
```

Примечание

Ключ доступа и секретный ключ, используемые для закачивания вашего комплекта, представляют собой ключ доступа и секретный ключ S3, а не сертификаты EC2, которыми вы пользовались для создания комплекта.

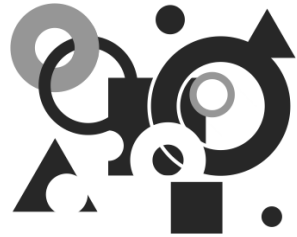
Эта команда тоже потребует некоторого времени на закачивание всех фрагментов вашего комплекта в S3. На основании личного опыта могу сказать, что здесь очень высока вероятность неудачи, поскольку в ходе ее исполнения выполняется большое количество команд S3 `PUT`. В случае сбоя не расстраивайтесь, а повторите попытку. Рано или поздно, но команда выполнится успешно.

Теперь остается один последний шаг, после которого вы сможете использовать образ. Этот шаг заключается в регистрации AMI в EC2. Чтобы зарегистрировать AMI, дайте следующую команду:

```
$ ec2-register myami/myami.manifest.xml
IMAGE ami-33a2d51c
```

При запуске новых экземпляров вы сможете использовать полученный идентификатор AMI (AMI ID).

Глава 3



Планирование перехода на облачную обработку данных

Большинство читателей этой книги наверняка уже строили Web-приложения, предназначенные для развертывания в традиционных центрах обработки данных. Теперь, когда вы получили общее представление о том, что представляет собой облачная инфраструктура, и как Amazon реализует облачные вычисления, настал момент рассмотреть проблемы, которые вам придется решать при переходе на облачную инфраструктуру.

В данной главе рассматривается широкий спектр таких проблем и соображений по их решению, хотя мы на данном этапе рассмотрим лишь общие подходы к переходу на использование облачных вычислений. В последующих главах каждая из проблем и подходы к ее решению будут рассматриваться более углубленно.

Вопросы лицензирования программного обеспечения

Когда заходит разговор о переходе на использование облачных вычислений, я всегда начинаю с вопросов лицензирования ПО, потому что это нетехническая проблема, которую очень легко недооценить и упустить из виду. То, что вы уже имеете схему лицензирования ПО, которая хорошо работает в традиционном центре обработки данных, еще не значит, что вы сможете перенести эти лицензии в облачную инфраструктуру.

Во многих современных облачных средах вы платите за ресурсы, используя такие единицы измерения ресурсов, как CPU-час. Например, самая дешевая из виртуальных машин в облаке Amazon стоит $0,10 за каждый час пользования экземпляром. Если эта виртуальная машина проработает 10 часов, а затем будет остановлена, вам потребуется заплатить за ее использование всего $1,00 — даже если это будет единственным фактом вашего пользования облаком Amazon в течение месяца.

В реальной жизни вы можете пользоваться, например, таким рабочим сценарием:

- начиная с полуночи и до 9 часов утра ваше приложение работает на двух серверах приложений (два сервера приложений используются для обеспечения избыточности);
- с 9 часов утра и до 17:00 вы запускаете еще 6 дополнительных серверов приложений, чтобы удовлетворить требованиям по нагрузке в рабочие часы;
- в вечернее время, начиная с 17:00 и до полуночи вы используете четыре сервера приложений, чтобы сэкономить на выплатах, но, тем не менее, удовлетворять потребности в ресурсах.

Если все это просуммировать, вам понадобится оплачивать 110 часов машинного времени в сутки. Если бы вы использовали традиционную инфраструктуру, вам бы потребовалось купить восемь серверов, которые работали бы все время.

К сожалению, не все поставщики ПО предоставляют лицензионные условия, которые соответствовали бы вашей оплате за пользование облачной инфраструктурой. Традиционные лицензии на ПО чаще всего основываются на количестве процессоров. Организация, которая использует 10 серверов приложений, должна оплатить 10 лицензий на серверное ПО — даже если 5 из этих 10 серверов выключаются на ночь.

Таким образом, при переходе на облачные вычисления вам необходимо прояснить условия лицензирования вашего ПО, в частности:

- поддерживает ли ваша лицензия расчет затрат на основе времени использования (CPU-час, количество пользователей и т. д.)?
- поддерживает ли само ПО работу в виртуальных средах?

Поскольку в облачной среде очень просто запускать новые экземпляры, вы очень легко можете попасть в такую ситуацию, когда один из технических сотрудников нижнего звена запустит такое количество экземпляров программного обеспечения, на которое у вас нет лицензии. В результате вы нарушите лицензионное соглашение с поставщиком ПО.

С точки зрения модели лицензирования, идеальным для использования в облачной среде является ПО на основе открытого кода (Open Source). Фактически именно гибкость модели лицензирования Open Source и сделала возможной реализацию облака Amazon. Если вы сможете полностью ликвидировать вопросы лицензирования при развертывании ваших приложений в облачной инфраструктуре, вы можете сконцентрироваться на других вопросах перехода на использование облачной обработки данных. Хотя большинство решений на основе открытого кода (например, Apache и большинство дистрибутивов Linux) предоставляют вам полную свободу действий, с вопросами лицензирования вам все же придется столкнуться, если вы приобретаете поддерживаемые версии программного обеспечения Open Source, например Red Hat Enterprise Linux или MySQL Enterprise. К счастью, схемы лицензирования такого ПО вполне дружественны по отношению к использованию в облачной инфраструктуре.

Если не брать в расчет модель лицензирования Open Source, которая наилучшим образом подходит для облачных вычислений, то второй будет модель, в соответствии с которой плата взимается за CPU в час. По мере того как облачная модель входит в обиход, все большее и большее количество поставщиков предлагают услуги с почасовой оплатой. Например, Microsoft, Valtira, Red Hat, Vertica, Sun, а также многие другие компании уже приняли условия почасовой оплаты за CPU и довольно неплохо поддерживают облачную обработку данных. Oracle тоже рекламирует свою доступность в облачных вычислениях, но вот, к сожалению, они все равно придерживаются своей устаревшей модели лицензирования, которая направлена на поддержание традиционных условий.

ПО, схема лицензирования которого основывается на количестве пользователей, тоже может адекватно работать в облачной среде. Основная сложность с таким ПО часто заключается в том, как оно производит проверку условий лицензирования. Вы можете подвергаться риску нарушения лицензионного соглашения, лицензия может быть "привязана" к конкретным MAC- или IP-адресам или система управления лицензиями может оказаться недостаточно интеллектуальной для того, чтобы поддерживать облачную среду, или необоснованно лишать вас возможности масштабирования вашей системы в облаке.

Наихудший сценарий[1] с точки зрения облачной инфраструктуры представляет собой ПО, условия лицензирования которого привязаны к количеству процессоров. Как и в некоторых системах, где схема лицензирования основывается на количестве пользователей, в составе такого ПО поставляются средства управления лицензиями, которые могут серьезно осложнить жизнь. Например, вам может понадобиться создать индивидуальную установку для каждого экземпляра ПО. Необходимость делать это оказывает отрицательный эффект на гибкость, которую предоставляет облачная инфраструктура.

Некоторые программы, схемы лицензирования которых основаны на количестве процессоров, требуют проверки (валидизации) на специальном сервере лицензирования. Любое ПО, в котором налагается такое требование, в облачной среде может оказаться неработоспособным, если оно не является достаточно интеллектуальным, чтобы "на лету" распознать замену физического сервера лицензирования на виртуальный. Даже если такое ПО и может распознать замену, вам все равно потребуется убедиться в том, что сервер лицензирования позволит вам запустить то количество серверов, которое вам требуется.

Однако за исключением тех случаев, когда сервер лицензирования превращается в помеху, результаты, достигаемые с помощью лицензионного ПО, в облачной среде не хуже, чем те, которых можно добиться в физической инфраструктуре. Если все ваше ПО использует те или иные схемы лицензирования, осложняющие жизнь, то преимущества облачной инфраструктуры могут оказаться незначительными или же вовсе сведены на нет.

---

[1] Впрочем, этот сценарий еще не самый худший. В наихудшем случае вы имеете дело с ПО, которое явным образом запрещается использовать в облаке или в виртуализованных средах.

Переход к модели затрат на облачную среду

Как я уже заметил в начале этой главы, плата за облачные ресурсы взимается по факту их использования. Для Amazon эта модель основана на такой единице, как CPU-час. Для некоторых других облаков, например, GoGrid, применяется оценка в RAM-час. Рассмотрим пример оценки затрат, которые придется понести с учетом потребности в ресурсах, описанной чуть ранее (два сервера приложений в течение интервала от полуночи до 9:00, восемь серверов в течение интервала от 9:00 до 17:00 и четыре — с 17:00 до полуночи).

Предположим, что вы имеете такую основную инфраструктуру:

$0,10/CPU-час: один балансировщик нагрузки

$0,40/CPU-час: два сервера приложений

$0,80/CPU-час: два сервера баз данных

В этом случае ваша ежедневная плата составит:

$2,40 + $44,00 + $38,40 = $84,80.

Ваши ежегодные затраты на хостинг составят $30 952,00 — без учета лицензионных отчислений за ПО, инструменты управления облачной инфраструктурой и заработной платы сотрудникам.

Подходы к сравнению расценок

Наилучший способ сравнить расценки в различных облачных моделях заключается в определении полной стоимости владения (Total Cost of Ownership, TCO) в течение периода амортизации аппаратных средств. В зависимости от организации, период амортизации аппаратных средств обычно составляет от двух до трех лет. Чтобы получить точную информацию о ваших полных затратах в облачной среде, вам необходимо принять во внимание следующие четыре статьи затрат:

- оценочные затраты на использование виртуального сервера в течение трех лет;
- оценочные затраты на лицензионные выплаты на поддержку использования виртуального сервера в течение трех лет;
- оценочные затраты на инструменты управления облачной инфраструктурой (в случае их применения) в течение трех лет;
- оценочные затраты на оплату труда по созданию образов машин, управления инфраструктурой и других работ в течение трех лет;
- затраты на использование любых сторонних инструментов для работы с облачной средой.

Если вам требуется по-настоящему точная оценка ваших затрат в течение трех лет, вам необходимо учесть времена платежей в течение трех лет и отрегулировать использование стоимости капитала в вашей организации. Все эти финансовые

ухищрения необходимы только в том случае, когда вы сравниваете затраты на облачную среду с затратами, которые вам придется понести с созданием новой собственной физической инфраструктуры. Впрочем, даже если вы понимаете эти финансовые механизмы, такой расчет все равно будет полезен.

Сниженные цены и существующая инфраструктура

В продолжение дискуссии, начатой в *главе 1* (*см. примечание "Уркхарт о препятствиях для отказа от собственной инфраструктуры IT"*), при таком анализе не учитываются невосполнимые издержки (sunk costs). Если вы можете использовать существующую инфраструктуру без дополнительных затрат, вы можете считать, что по этой статье у вас будут нулевые затраты. Если у вас есть дополнительные серверы и IT-ресурсы, которые обеспечивают избыточную отказоустойчивость, вы можете обнаружить, что ваша существующая инфраструктура требует меньших общих затрат. Принимая во внимание это соображение, вам, тем не менее, следует решить, не повлечет ли откладывание перехода на облачные вычисления каких-либо долгосрочных затрат, которые могут уменьшить выигрыш от использования существующей инфраструктуры.

При выполнении сравнения вам необходимо исследовать следующие аспекты имеющихся альтернативных вариантов.

- Каковы будут ваши затраты на предоплату (плата за установку, капиталовложения в физическое пространство, затраты на аппаратные средства, покупку лицензий)?
- Каковы предполагаемые трудозатраты на установку инфраструктуры?
- Каковы затраты, ассоциированные с работой инфраструктуры (хостинг, арендная плата за помещение, затраты на электроэнергию, страхование)?
- Каковы трудозатраты, ассоциированные с поддержкой аппаратных средств и сетевой инфраструктуры?
- Каковы предстоящие затраты на лицензирование и/или апгрейд? Каковы будут затраты на поддержку программного обеспечения?

В *главе 1* я представил пример, иллюстрирующий сравнение затрат на инфраструктуру простого транзакционного приложения в физической внутренней инфраструктуре IP предприятия, в среде управляемых сервисов и в облачной среде. Хотя этот пример и дает очень хорошее представление о том, насколько выгодной может быть облачная среда, вы должны в обязательном порядке провести собственный анализ, используя ваши собственные экономические показатели и планируемые ресурсы. Выполняя такие расчеты, вы должны убедиться в том, что ваша планируемая облачная инфраструктура будет надлежащим образом поддерживать ваши производственные требования, затраты на вашу планируемую внутреннюю инфраструктуру учитывают такие аспекты, как затраты на помещения (плату за электроэнергию и другие коммунальные услуги), а затраты на управляемый хостинг включают стоимость услуг, не включенных в соглашение на предоставление управляемых сервисов.

Пример анализа прибыли на инвестированный капитал

В *главе 1* я приводил теоретический анализ на инфраструктуру без детального объяснения источников той или иной статьи расходов. Теперь, после более подробного обсуждения и зная материал, изложенный до сих пор, можно выполнить более конкретный анализ прибыли на инвестированный капитал применительно к конкретной инфраструктуре, сравнив затраты на развертывание внутренней инфраструктуры предприятия и реализации ее в облачной среде.

Примечание

Не следует слишком въедливо относиться к конкретным статьям затрат и вариантам выбора при покупке оборудования или опциям выбора облачных сервисов, приведенным в этом примере. Его цель не заключается в предоставлении подробного анализа прибыли на инвестированный капитал (ROI) в расчете на внутренний центр обработки данных или облачную инфраструктуру. Напротив, основная задача этого примера анализа экономических показателей облачной инфраструктуры в сравнении с внутренней физической IT-инфраструктурой предприятия или организации заключается в том, чтобы продемонстрировать основные вопросы, которые вам надо будет принять во внимание, выполняя такой анализ для своего собственного предприятия. Ваш анализ должен учитывать вопросы затрат на организацию инфраструктуры и стоимость этих решений для вашего предприятия.

Данный конкретный пример подразумевает, что два сервера приложений без особых проблем поддерживают стандартные требования к ресурсам и производительности. Кроме того, в рассматриваемом примере предполагается, что период пиковых нагрузок наступает на 15-й день каждого месяца и продолжается 24 часа. Обслуживание этой пиковой нагрузки при обеспечении определенного уровня производительности в соответствии с требованиями стандарта требует подключения четырех дополнительных серверов. Если ограничиться всего двумя дополнительными серверами приложений, то рассматриваемая система все же может функционировать при пиковых нагрузках, но ее производительность существенно упадет.

Если вы начинаете вести бизнес "с нуля", то для организации минимально необходимой инфраструктуры вам понадобится приобрести следующее оборудование:

- полустойку у надежного ISP с полосой пропускания, достаточной для поддержания ваших требований;
- два надежных брандмауэра (firewall);
- один надежный аппаратный балансировщик нагрузки;
- два хороших коммутатора гигабитного Ethernet;
- шесть надежных серверов бизнес-класса (для работы в условиях пиковых нагрузок на 15-й день).

Если вы сделаете выбор в пользу облачной инфраструктуры, вам потребуются несколько виртуальных экземпляров:

- один 32-битный экземпляр умеренной производительности (модель `medium`);

- четыре 64-битных экземпляра (модель `large`) для работы в стандартных условиях, с увеличением их количества до 8 для работы в условиях пиковой нагрузки.

В дополнение к этому вам потребуются программное обеспечение и сервисы. В предположении того, что вы будете использовать только открытое ПО, ваши затраты на программное обеспечение и сервисы составят только повременную оплату за установку рабочей среды, сервисов мониторинга, контрактов на поддержку и трудозатрат на управление вашей средой. В табл. 3.1 представлены ваши ожидаемые начальные затраты и повременные затраты на поддержание вашей инфраструктуры.

Таблица 3.1. *Затраты, ассоциированные с физической инфраструктурой и облачной инфраструктурой*

	Внутренняя инфраструктура (начальные единовременные затраты)	**Облачная инфраструктура (начальные единовременные затраты)**	**Внутренняя инфраструктура (ежемесячные затраты)**	**Облачная инфраструктура (ежемесячные затраты)**
Стойка	$3000	$0	$500	$0
Коммутаторы	$2000	$0	$0	$0
Балансировщик нагрузки	$20 000	$0	$0	$73
Серверы	$24 000	$0	$0	$1206
Брандмауэры	$3000	$0	$0	$0
Круглосуточная техподдержка	$0	$0	$0	$400
Управляющее ПО	$0	$0	$100	$730
Ожидаемые трудозатраты	$1200	$1200	$1200	$600
Снижение производительности[1]	$0	$0	$100	$0
Totals	$53 200	$1200	$1900	$3009

[1] Не следует забывать, что если мы приобрели только четыре сервера приложений для внутренней инфраструктуры IT вместо шести, то нам придется мириться с пониженной производительностью на 15-й день каждого месяца. Это снижение производительности приведет к определенным убыткам. Как правило, это будут либо упущенная выгода, либо задержки в получении денег. Цифра $100 взята мною "с потолка" — просто потому, что любая цифра свыше $110, на мой взгляд, уже оправдывает добавление новых серверов, а любые суммы меньше этой делают вопрос о поддержке пиковых нагрузок спорным.

Чтобы ваш анализ был полным, вам необходимо изучить график амортизации оборудования в вашей организации и ее механизм формирования процентов на капитал. Как правило, срок амортизации аппаратных средств составляет от двух до трех лет. В рассматриваемом примере я использую более консервативную схему, в соответствии с которой срок амортизации составляет три года. Эта схема определяет ожидаемый срок службы аппаратных средств и позволяет определить, как ежемесячные затраты комбинируются с единовременными.

Проценты на капитал для большинства организаций лежат в диапазоне от 10 до 20 %. Теоретически проценты на капитал — это то, во что вам обойдутся ваши деньги, если вы их инвестируете. Например, предположим, что у вас есть $10 000, а показатель процентов на капитал, равный 10 %, говорит о том, что вы можете увеличить свой капитал с $10 000 до $11 046,69 (с ежемесячной капитализацией процентов) в течение года с использованием одного или нескольких стандартных инвестиционных механизмов. Еще один вариант рассмотрения этой ситуации заключается в том, что вы можете считать, что вы одолжили $10 000, то через год должны будете вернуть $11 046,69. В любом случае проценты на капитал $10 000 составляют 10 % ежемесячно.

Если не учитывать затраты на капитал, то построение внутренней инфраструктуры IT обойдется вам в $121 600, а облачная инфраструктура будет стоить $109 254. Эти цифры получаются за счет сложения сумм начальных единовременных инвестиций и месячных затрат после амортизационного периода (в рассматриваемом примере он составляет три года). В приведенном примере очевидно, что облачная инфраструктура экономически выгоднее, даже если не принимать во внимание проценты на капитал. Проценты на капитал всегда будут выше для того варианта, который требует бóльших капиталовложений на начальном этапе. В результате, даже если обе цифры окажутся приблизительно равными $100 000, из графика капиталовложений будет понятно, что облачная инфраструктура с финансовой точки зрения более привлекательна.

Если вы хотите рассмотреть сценарии, в которых облачная инфраструктура будет чуть дороже, или если вы хотите узнать, какого уровня развития должен достигнуть ваш бизнес, чтобы оправдать созданную вами инфраструктуру, вам следует выполнить финансовый анализ с учетом процентов на капитал. Это позволит вам получить возможность оценить истинную стоимость каждого из вариантов для вашего предприятия. В финансовых терминах это означает, что вам требуется вычислить так называемую текущую ценность (present value), также известную как дисконтированная или приведенная стоимость[1] (сумма ожидаемого в будущем дохода или платежа минус процент на капитал как "компенсация за ожидание") всех ваших ежемесячных выплат за период амортизации.

[1] Дисконтированная стоимость выражает стоимость будущих потоков платежей в значении текущих потоков платежей. Определение дисконтированной стоимости широко используется в экономике и финансах как инструмент сравнения потоков платежей, получаемых в разные сроки. Подробнее см. **http://en.wikipedia.org/wiki/Present_value**, **http://tinyurl.com/3ybqc93**. — *Прим. перев.*

Как правило, вычислению приведенной стоимости (present value) обычно посвящается целая глава в учебниках по финансовым вычислениям, и эта тема, безусловно, выходит далеко за рамки круга вопросов, обсуждаемых в этой книге. К счастью, в большинстве финансовых калькуляторов[1], а также в таких программах, как Microsoft Excel[2] или Apple Numbers[3], уже имеются готовые функции, с помощью которых вы легко сможете выполнить данные расчеты.

Для каждого варианта вам следует вычислить приведенную стоимость всех ваших ежемесячных платежей и сложить полученное значение с первоначальными единоразовыми капиталовложениями:

- стоимость внутренней инфраструктуры:

  = (–PV(10 % / 12,36,1900,0)) + 53 200 = $112 083,34;

- стоимость облачной инфраструктуры:

  = (–PV(10 % / 12,36,3900,0)) + 1200 = $94 452,63.

Теперь вы видите, что облачная инфраструктура не просто стоит дешевле, но и сама ее схема выплат позволяет сэкономить несколько тысяч долларов по сравнению с физической внутренней инфраструктурой IT.

Наконец, полученные цифры ($112 083,34 и $94 452,63) позволяют вам оценить, какие деньги вы должны заработать в течение трех лет для того, чтобы ваш бизнес начал приносить прибыль (разумеется, вам необходимо учесть еще и инфляцию).

На чем еще позволяет экономить облачная инфраструктура

В процессе проведения финансового анализа вы обнаружите, что ваша экономия будет умеренной, если ваше приложение имеет статическую потребность в ресурсах. Иными словами, если вы постоянно потребляете один и тот же объем ресурсов, то вы не сможете полностью прочувствовать одно из ключевых преимуществ от использования облачной инфраструктуры. Выбор облачной инфраструктуры даст вам некоторое преимущество над физической внутренней IT-инфраструктурой за счет экономии трудозатрат, но в некоторых ситуациях облако может обойтись даже дороже, чем управляемые сервисы, особенно если вам требуется среда, обладающая по-настоящему высокой отказоустойчивостью и высокой степенью доступности.

Экономия за счет использования облачной среды становится значительной, а иногда достигает и просто немыслимых уровней, если ваша среда динамична и между пиковыми нагрузками и средним уровнем нагрузки наблюдается существенная разница. Например, у моей компании есть один заказчик, чей пример может быть очень показательным. Обычно в течение целого года сайт этой компании посещает

---

[1] См., например: **http://www.calculatorsoup.com/calculators/financial/present-value.php**. — *Прим. перев.*

[2] См. **http://castle.eiu.edu/~dmcgrady/bus3710/assign/excel_functions.html**. — *Прим. перев.*

[3] См. **http://docs.info.apple.com/article.html?path=Numbers/1.0/en/c12la125.html/** — *Прим. перев.*

всего лишь несколько пользователей за день. Однако в конце квартала картина резко меняется и за 15 минут их сайт может посетить более 10 миллионов разных пользователей. Очевидно, что приобретение аппаратных средств для поддержания этих пиковых нагрузок — это чудовищное расточительство (фактически с полной нагрузкой это оборудование будет работать всего лишь 1 час в году). Тем не менее, они не могут позволить себе снижения производительности даже на 1 час в году. Для них облачная инфраструктура будет идеальным решением, потому что бóльшую часть времени они смогут работать с минимальным потреблением ресурсов, и мгновенно наращивать вычислительные мощности в те минуты, когда им необходимо поддерживать пиковые нагрузки.

Распространенные статьи экономии заключаются в управлении непроизводственными средами — перемещение данных, тестирование, разработка и т. д. Обычно такие среды требуются предприятию в определенные моменты времени на различных этапах разработки приложений, после чего необходимость в них отпадает. Более того, при тестировании обычно требуется полное дублирование производственной среды. Если вы пользуетесь облачными сервисами, вы на короткий срок можете развернуть полный дубликат вашей производственной среды, выполнить тестирование, а затем, когда ваши задачи будут выполнены, свернуть эту среду и освободить ресурсы.

Уровни сервиса для облачных приложений

Когда предприятие предоставляет клиентам некий сервис — не важно, какой именно, облачный или традиционный на основе собственного центра обработки данных — то эта компания, как правило, предоставляет клиентам соглашение об уровне сервиса (service level agreement, SLA), где указываются ключевые параметры (уровни сервиса), которых клиент вправе ожидать от данного сервиса. Прежде, чем вы примете решение о переходе на облачные технологии, вам необходимо понять на концептуальном уровне, что представляют собой доступность (availability), надежность (reliability) и производительность (performance) облачных сервисов.

Доступность

Доступность — это показатель, который указывает, насколько часто может использоваться сервис в течение предопределенного периода времени. Например, если Web-сайт доступен широкой публике в течение 710 часов из 720-часового периода (1 месяц), то можно сказать, что в течение этого месяца доступность сервера составляла 98,6 %.

Хотя показатель 98,6 % на первый взгляд кажется очень хорошим, приемлемость этого значения в действительности сильно зависит от того, для какого приложения замеряется этот показатель, а иногда и от того, какие отдельные функции

приложения были доступными. Например, если Google spider[1] не будет работать в течение 24 часов, но при этом вы все равно сможете выполнять поиск и получать результаты, сочтете ли вы, что "Google не работает"?

Большинство людей считают, что система обладает высокой доступностью, если объявленное ожидаемое значение показателя доступности составляет от 99,99 до 99,999 %. При доступности 99,999 % система может оказаться недоступной только 5 минут 15 секунд за год.

Удача — это еще не высокая доступность

Когда о системе говорится, что она является системой высокой доступности, имеется в виду то, на что вы можете рассчитывать в будущем, а не то, что произошло в течение предыдущего года, который прошел удачно для вас. Например, то, что ваш Web-сайт, размещенный на старом сервере на базе процессора Intel 486, в течение прошлого года работал без перебоев, еще не значит, что вы имеете систему высокой доступности. Если вероятность отказа этого старого сервера в течение календарного года составляет примерно 40 %, у вас нет никакого основания считать его системой высокой доступности, несмотря на его производительность в течение прошлых лет.

Как оценить доступность вашей системы

Большинство перебоев в работе сервисов представляют собой результат неполадок в работе оборудования. Эти перерывы в работе могут оказаться длительными, если администраторы сервиса неправильно диагностируют причину сбоя или допустят другие ошибки в устранении возникшей проблемы. Таким образом, для оценки ожидаемой доступности должны использоваться две переменные:

- вероятность возникновения сбоев или неполадок системы в течение оценочного периода;
- ожидаемое время простоя в случае возникновения сбоев и неполадок.

Ожидаемая доступность компонента выражается следующей математической формулой:

$$a = (p - (c \times d)) / p.$$

Здесь:

- a — ожидаемая доступность;
- c — вероятность (в %) отказа сервера в течение заданного периода;
- d — ожидаемое время простоя вследствие отказа сервера;
- p — оцениваемый период.

Таким образом, если ваш старый компьютер на базе процессора i486 имеет вероятность отказа, которая составляет 40 %, и в этом случае вы предполагаете, что

[1] Google spider — поисковый робот Google, программа, представляющая собой часть поисковой системы Google и предназначенная для сканирования интернет-страниц с целью занесения информации о них в базу данных поисковика. Такие программы есть во всех поисковых системах, называться они могут по-разному ("веб-пауки", краулеры). Подробнее см. **http://en.wikipedia.org/wiki/Web_crawler**, **http://www.dmoz.org/Computers/Internet/Searching/Search_Engines/**. — *Прим. перев.*

время простоя составит 24 часа, то его ожидаемая доступность составит (8760 – (40 % × 24)) / 8760, или 99,9 %.

Показатель доступности 99,9 % выглядит довольно неплохо, особенно для старого компьютера на базе процессора i486, не так ли? Ладно, в данном случае я всего лишь чрезмерно упростил ситуацию. Вы что, действительно верите, что ваше кабельное или DSL-соединение будет работать бесперебойно и никогда не будет разрываться? Вы действительно верите, что за 24 часа вы успеете заменить сервер, сконфигурировать его и восстановить данные с резервной копии? Как обстоят дела с вашим сетевым оборудованием? Насколько надежны ваши резервные копии?

Чтобы получить надежную оценку доступности, вам необходимо оценить все возможные компоненты, отказ которых может привести к перебоям в работе, и просуммировать их. Доступность системы оценивается как разность общей продолжительности оценочного периода минус сумма продолжительности всех простоев в течение этого периода, поделенная на общую продолжительность оценочного периода:

$$a = (p - \mathrm{SUM}(c1 \times d1 \,/\, cn \times dn)) \,/\, p.$$

Например, если ваш провайдер обычно испытывает технические проблемы два раза в год, причем время простоя обычно составляет два часа, то доступность интернет-соединения оценивается так:

$$(8760 - (200\ \% \times 2)) \,/\, 8760 = 99{,}95\ \%.$$

Таким образом, общий показатель доступности вашей системы будет таким:

$$(8760 - ((40\ \% \times 24) + (200\ \% \times 2))) \,/\, 8760 = 99{,}84\ \%.$$

Этот пример призван продемонстрировать горькую правду о доступности программных систем: чем больше точек отказа (points of failure), которые представляют собой компоненты, отказ которых приведет к простою системы, тем ниже ее рейтинг доступности. Далее, продолжительность времени простоя оказывает еще более сильное влияние на вероятность того, что ваш сервис будет недоступен.

Смягчить проблему помогает избыточность. Если у вас есть два или более физических компонентов, представляющих логические компоненты, то ожидаемое время простоя логического компонента представляет собой ожидаемую продолжительность периода времени в случае события, когда все эти физические компоненты откажут одновременно. Иными словами, формула $c \times d$, используемая для вычисления времени простоя, несколько усложняется и принимает следующий вид:

$$(c \times d^n) \,/\, (p^{(n-1)}).$$

В этой формуле n представляет собой уровень избыточности системы. В случае, когда $n = 1$, формула, как и ожидалось, упрощается:

$$(c \times d^n) \,/\, (p^{(n-1)}) = (c \times d) \,/\, (p^0) = c \times d.$$

Если в рассматриваемом примере в систему добавить еще один избыточный компонент — еще один сервер с процессором i486, то это позволит быстро переходить на другой ресурс при сбое, и в данном случае оценка доступности вашего Web-сервера будет намного улучшена:

$$(8760 - ((40\ \% \times 24^2)) \,/\, (8760^{(2-1)})) \,/\, 8760 = 99{,}999\ \%.$$

Из чего складывается доступность?

На определенном уровне высокая доступность — это вопрос субъективного восприятия. Если доступность — это одно из требований, предъявляемых к вашей системе, то вы должны не только определить процент времени, в течение которого она будет доступна, но и определить само понятие доступности (т. е. что означает доступность вашей системы для пользователя). В частности, вам потребуется определить следующие критерии доступности.

- Какие функциональные возможности должны быть доступны потребителю, чтобы ваша система воспринималась как доступная? Например, ранее в этой главе мы рассматривали пример с поисковой системой Google — до тех пор, пока пользователи могут выполнять поиск и получать результаты, Google можно считать доступным, вне зависимости от того, каков на данный момент статус поискового робота Google (Google spider).
- Следует ли вам включить в анализ периоды планового простоя? Включение планового простоя в чем-то противоречиво, но оно может вам потребоваться для некоторых видов управления доступностью. Например, возможна такая ситуация, когда архитектура вашей системы позволяет вам обеспечить доступность 99,999 %, но вы, тем не менее, предпочитаете запланировать для своей системы 1 час простоя в неделю для проведения плановых профилактических работ (вследствие повышенных требований к безопасности или любого другого критерия, который не имеет отношения к теоретической оценке доступности системы как таковой). Если вы производите оценку для внутренних целей, вы можете считать, что оценка 99,999 % объективна. Но если вы пытаетесь рекламировать систему для конечных пользователей и сообщаете им, какого уровня доступности им следует ожидать, то оценка 99,999 % введет их в заблуждение.
- В течение какого процента времени ваша среда должна быть доступна? Например, для Web-сайта можно задать требование, в соответствии с которым доступность его домашней и всех дочерних страниц из-за пределов корпоративной сети должна составлять 99,999 % ежедневно от 07:00 до 21:00.

Доступность облачных сервисов

Поскольку мы планируем архитектуру для развертывания Web-приложений в облаке, нам необходимо определить, что мы можем ожидать от облачной инфраструктуры. Хотя при оценке облачной инфраструктуры действуют те же концепции высокой доступности, что и при оценке традиционных центров обработки данных, при использовании этих двух парадигм понятие надежного компонента сильно отличается.

Возможно, наиболее странной и тревожащей может показаться идея о том, что большинство отказов, которые в традиционных центрах обработки происходят редко, в облачной инфраструктуре случаются очень часто. Этот кажущийся недостаток надежности компенсируется тем фактом, что многие из отказов, которые в

обычном центре обработки данных считаются катастрофическими, в облаке являются обыденностью.

Например, экземпляр EC2 считается абсолютно ненадежным по сравнению с ожидаемой доступностью обычного сервера с избыточными компонентами. Такие события, как утрата физического сервера без всяких предупреждений, случаются крайне редко. Напротив, когда один из компонентов отказывает (или выдает предупреждение о высокой вероятности отказа), он замещается избыточным компонентом, что позволяет вам избежать простоя системы. Во многих облачных системах, например Rackspace и GoGrid, вы также можете ожидать аналогичных уровней надежности. Но вот в облаке Amazon ваши экземпляры периодически будут отказывать без всяких предупреждений, и это можно утверждать с полной стопроцентной гарантией.

Более того, отказ виртуального экземпляра можно сравнить с попаданием гранаты в ваш сервер без причинения побочного вреда (кроме уничтожения этого сервера). Иначе говоря, сервер будет потерян бесповоротно, восстановить вы его не сможете.

Вы, наверное, думаете, что я пытаюсь вас запугать? Эта отличительная особенность — один из тех подводных камней, которые заставляют технических специалистов скептически относиться к облаку. Это действительно пугает, потому что в физической инфраструктуре потеря сервера — это небольшая катастрофа. Но в виртуальном мире это практически ничего не значащее событие. Фактически в облачной среде вы можете потерять целую зону доступности и считать это мелким инцидентом. Напротив, если вы работаете в физической среде — задумайтесь, как вы воспримете такое событие, как внезапная потеря всего вашего центра обработки данных?

В *главах 4 и 6* будут описаны приемы, которые позволят не просто смягчить потерю и ликвидировать все ваши страхи, но и почувствовать себя в облачной среде более комфортно и уверенно, нежели в физической инфраструктуре.

Уровни сервиса в Amazon Web Services

Одной из областей, где другие компании конкурируют с Amazon, является область уровней сервиса. Большинство конкурентов предлагают в своих облачных инфраструктурах более высокие уровни сервиса, нежели Amazon. Хотя Amazon уже в течение нескольких лет предоставляет своим клиентам SLA для S3, для EC2 формальное соглашение об уровне сервиса было добавлено относительно недавно. SLA для сервиса S3 обещает, что S3 будет иметь уровень доступности 99,5 % в течение каждого календарного месяца. С другой стороны, EC2 определяет более сложный уровень доступности сервиса. В частности, EC2 декларирует доступность 99,95 % как минимум для двух зон доступности в пределах региона.

Эти уровни сервиса не обеспечивают непосредственного соответствия для уровней сервиса, которые вы могли бы обещать своим клиентам, выполняя развертывание своей инфраструктуры в облаке.

В частности:

- Чтобы запустить экземпляр EC2, вам нужен сервис S3. Если сервис S3 характеризуется доступностью 99,5 %, вы сможете запускать новые экземпляры EC2 только в течение 99,5 % времени в течение календарного месяца, вне зависимости от того, насколько хорошо EC2 выполняет или даже перевыполняет обещанные показатели. Этот недостаток применим и к моментальным снимкам, и к созданию томов, потому что вы не можете ни создать моментальный снимок, ни новый том, пока сервис S3 не станет вновь доступен.
- EC2 обеспечивает заявленный уровень сервиса до тех пор, пока в пределах одного региона доступны две зоны доступности (с вероятностью 99,95 %). EC2 технически может обеспечить заявленный уровень сервиса, даже если целая зона доступности внезапно исчезнет.
- Вы должны самостоятельно разрабатывать архитектуру вашего приложения, чтобы она с достаточным уровнем надежности поддерживала предъявляемые к ней требования.

Можно ли вообще доверять AWS?

Важно не то, что Amazon обещает определенный уровень доступности, а то, что они выполняют это обещание.

Очень важно здесь понимать отличия между EC2 и S3. EC2 основывается на хорошо известной технологии (Xen), модифицированной таким образом, чтобы решить понятную проблему: проблему виртуализации. С другой стороны, S3 является в значительной мере "доморощенной" системой, выполняющей вещи довольно уникальные. На основании собственного опыта я могу сказать, что проблемы с EC2 относятся к довольно обыденным вопросам, связанным с центрами обработки данных, в то время как проблемы с S3 коренятся в довольно загадочных элементах патентованной технологии Amazon, использованной для реализации облачного хранения данных.

"Сухой остаток" заключается в следующем: сервис EC2 организован понятнее и лучше изучен, чем S3. Таким образом, обещания Amazon в отношении EC2 более объективны и справедливы, нежели их обещания в отношении S3. Реальность же такова: в прошлом году сервис S3 имел значительные простои[1], и доступность на уровне 99,5 % обеспечена не была. В течение того же самого периода в работе EC2 проблем не наблюдалось, и уровень доступности сервиса 99,95 % соблюдался, несмотря даже на то, что на тот момент компания Amazon еще не декларировала SLA для этого сервиса и позиционировала его как "бета"-сервис.

Каков же основной вывод, который можно сделать на основе только что сказанного? Вам следует разработать такую инфраструктуру приложений, которая не полагается на сервис S3 для того, чтобы обеспечить соблюдение ваших собственных целей по доступности вашего сервиса. В последующих главах этой книги я поделюсь с вами своими соображениями о том, как этого добиться.

---

[1] Я не хочу "раздувать" шумиху вокруг этих простоев и придавать чересчур важное значение. На основании собственного опыта могу сказать, что коммуникации с Amazon и "прозрачность" их взаимодействия с клиентами во время этих простоев были просто выдающимися, и они действительно выполняют очень большой объем работ по повышению надежности сервиса, в разработке которого они движутся методом проб и ошибок. Возможно, что к тому моменту, когда вы будете читать эту книгу, проблемы простоев S3 давно уже станут пережитком прошлого.

Ожидаемая доступность в облачной среде

Одним из ключевых отличий в оценке простоев в облачной и физической средах является то, насколько в каждой из этих сред просто создать такую инфраструктуру, которая позволит осуществить быстрое восстановление в случае критических сбоев. Иными словами, даже несмотря на то, что физический сервер более надежен, нежели виртуальный сервер, расположенный в облачной среде, облачная инфраструктура позволяет вам с минимальными затратами создавать такой уровень избыточности, который значительно превысит уровень избыточности физического центра обработки данных. Это, в свою очередь, означает, что в облачной среде вы сможете быстрее восстановиться после критического сбоя и снизить время простоя.

Давайте рассмотрим очень простую инфраструктуру, состоящую из одного балансировщика нагрузки, двух серверов приложений и сервера базы данных, и сравним уровень доступности этих инфраструктур в физической и облачной средах.

При реализации этой архитектуры в физическом центре обработки данных вы, как правило, получите единую стойку у вашего хостинг-провайдера, обычные потребительские серверы с избыточными компонентами и аппаратным балансировщиком нагрузки. Если не учитывать частоту сетевых сбоев и исключить возможность катастрофы в физическом центре обработки данных, то вы, скорее всего, сможете добиться уровня доступности, определяемого следующими расчетами:

- балансировщик нагрузки:
  99,999 %
  (Большинство организаций и предприятий предпочитают использовать аппаратные балансировщики нагрузки, чтобы снизить вероятность отказа компонента.);
- сервер приложений:
  $(8760 – ((30\ \% \times (24^2)) / 8760)) / 8760 = 99{,}999\ \%$;
- сервер базы данных:
  $(8760 – (24 \times 30\ \%)) / 8760 = 99{,}92\ \%$;
- система в целом:
  $(8760 – ((24 \times 30\ \%) + (24 \times (((30\ \% \times (24^2)) / 8760)) + (24 \times 30\ \%))) / 8760 = 99{,}84\%$.

В данном расчете я исходил из предположения, что балансировщик нагрузки практически никогда не выйдет из строя в течение его амортизационного периода, что серверы периодически будут выходить из строя и что потеря любого из компонентов системы вызовет простой продолжительностью 24 часа.

Наилучший вариант усовершенствования этой инфраструктуры — иметь запасные избыточные компоненты под рукой. За счет снижения времени простоя, ассоциированного с любым из компонентов, вы можете существенно повысить свой рейтинг доступности. Фактически, если вы пользуетесь услугами управляемого хостинга, то обычно будете иметь дополнительные компоненты и дополнительные серверы, которые позволят вам снизить время простоя за счет быстрой замены от-

казавших компонентов и их конфигурирования. С другой стороны, если вы сами управляете своими серверами и не имеете гарантии от поставщика вашего оборудования на замену отказавшего компонента в течение 24 часов, то вы можете получить недопустимо низкий рейтинг доступности.

Что в действительности означают эти цифры?

Рассудительные пользователи могут не согласиться с цифрами, приведенными мною в этой главе, и обязательно это сделают. Цель приведенных примеров вычислений не заключается в заявлении, что традиционный центр обработки данных обеспечит для описываемой архитектуры доступность 99,84 %, а облачная среда — 99,994 %. Если вы, прочитав этот раздел, запомните эти цифры и будете повсюду повторять, что такую вещь сказал вам я, то знайте, что вы меня неправильно поняли!

А в чем тогда цель приведенных вычислений? Просто в том, чтобы вы поняли сам подход к обеспечению уровня доступности и поняли, насколько понятие стабильности в облачной среде отличается от понятия стабильности в физической инфраструктуре. Вот эти отличия:

- экземпляры EC2 намного менее стабильны, чем физические серверы;
- использование нескольких зон доступности может существенно смягчить проблему стабильности экземпляров EC2;
- недостаточная стабильность программного балансировщика нагрузки в значительной мере не имеет отношения к стабильности благодаря возможности его быстрой и автоматической замены.

В облачной среде вычисления производятся другим способом. Балансировщик нагрузки представляет собой просто еще один экземпляр, а отдельные экземпляры серверов характеризуются гораздо более низкой надежностью. С другой стороны, возможность пересечения зон доступности увеличивает доступность кластера. Далее, время простоя для узлов значительно уменьшается[1]:

- балансировщик нагрузки:
  $(8760 – (0{,}17 \times 80\ \%)) / 8760 = 99{,}998\ \%$;
- сервер приложений:
  $(8760 – (17\ \% \times (0{,}17^2 / 8760))) / 8760 = 99{,}9999\ \%$;
- сервер базы данных:
  $(8760 – (0{,}5 \times 80\ \%)) / 8760 = 99{,}995\ \%$;
- система в целом:
  $(8760 – ((0{,}17 \times 80\ \%) + (17\ \% \times (0{,}17^2 / 8760)) + (0{,}5 \times 80\ \%))) / 8760 = 99{,}994\ \%$.

В этом примере подразумевается, что вы пользуетесь специальным инструментарием для автоматического восстановления вашей облачной инфраструктуры. Если вы все выполняете вручную, то вам необходимо принять в расчет повышенное время простоя, ассоциированное с ручным управлением облачной средой. Наконец, в этом расчете не учитывается влияние сервиса S3 на надежность. В реальности

[1] Я не учитываю влияние потери отдельного узла на производительность и основное внимание уделяю именно доступности. Если ваши избыточные серверы приложений работают на полную мощность, то потеря одного из них может привести к снижению производительности или даже потере доступности.

всегда существует небольшой шанс на то, что когда вы попытаетесь запустить новый экземпляр, предназначенный на замену утраченного (что сводит время простоя к минимуму), вы можете столкнуться с невозможностью это сделать вследствие недоступности S3.

Надежность

Надежность часто связана с доступностью, хотя в действительности надежность представляет собой несколько иную концепцию. В частности, надежность относится к тому, насколько вы можете полагаться на способность системы защищать целостность данных и выполнять свои транзакции. Нестабильность, ассоциированная с низкой доступностью очень часто имеет побочный эффект, в результате которого люди не испытывают уверенности в том, что их последний запрос действительно был выполнен, что часто может привести к повреждению данных в системах управления реляционными базами данных.

Примечание

Очевидно, что система, которая часто недоступна, не может считаться и надежной. Система с высоким уровнем доступности, тем не менее, тоже может быть ненадежной, если вы не доверяете тем данным, которые она предоставляет. Такое может произойти, например, в случае, когда один из процессов или компонентов неожиданно останавливается, никак об этом не сообщая.

В значительной степени надежность вашей системы зависит от того, как написан код, управляющий ее работой. Облачная структура имеет еще ряд аспектов, выходящих за рамки написания кода вашего приложения, которые, тем не менее, тоже могут повлиять на надежность вашей системы. В облачной среде наиболее серьезным среди этих аспектов является то, как вы управляете постоянными данными.

Поскольку виртуальные экземпляры имеют тенденцию иметь более низкую доступность по сравнению с их физическими аналогами, шанс на то, что данные окажутся поврежденными, в облачной среде повышается по сравнению с физическим центром обработки данных. В частности, каждый раз, когда происходит потеря сервера, существенное влияние могут оказать следующие факторы.

- Вы теряете любые данные, которые хранятся в этом экземпляре и никогда не резервировались.
- Ваши устройства блочного хранения подвергаются риску повреждения (в точности так же, как это может случиться и в традиционном центре обработки данных).

Вопросы выработки стратегии борьбы с этими негативными факторами будут рассматриваться в *главе 4*. На данный момент запомните два эмпирических правила, которые позволят вам повысить надежность приложений, развернутых в облачной среде.

- Не храните постоянные данные EC2 в эфемерных хранилищах экземпляра (не обязательно применимо к другим облакам, отличным от Amazon).
- Регулярно создавайте моментальные снимки ваших блочных томов.

Производительность

Большей частью все вопросы, о которых вы беспокоитесь при развертывании высокопроизводительного транзакционного приложения в физическом центре данных, относятся и к развертыванию приложений в облачной среде. Стандартные рекомендации сводятся к следующему.

- Разрабатывайте ваши приложения таким образом, чтобы логические операции могли быть распределены по множеству серверов.
- Если вы не создаете кластеры серверов баз данных, сегментируйте доступ к базе данных таким образом, чтобы операции чтения могли выполняться на подчиненных серверах (slaves), а операции записи — на главном (master).
- Применяйте такие механизмы, как многопоточность и/или ветвление процессов, чтобы как можно эффективнее реализовать потенциал каждого отдельного процессорного ядра.

Организация кластеров или независимые узлы

В зависимости от природы вашего приложения, ваши "стратегические точки" могут располагаться на уровне сервера приложений или на уровне базы данных. На уровне приложений у вас, в принципе, есть два возможных варианта для распределения обработки по множеству серверов приложений.

- Использовать балансировщик нагрузки для автоматического разбиения сеансов на неопределенное количество независимых узлов. В случае применения этого подхода каждый экземпляр виртуального сервера полностью лишен информации о других экземплярах. Следовательно, каждый экземпляр не может содержать больше информации о критическом состоянии, чем ему требуется для успешной обработки запросов, с которыми он работает в данный момент.
- Использовать балансировщик нагрузки для маршрутизации трафика на узлы, собранные в кластер серверов приложений. Путем организации кластеров узлы серверов приложений могут взаимодействовать друг с другом и совместно использовать информацию о статусе. Организация кластеров дает то преимущество, что вы можете организовать обмен информацией о статусе на уровне сервера приложений; наряду с этим, недостатком этого подхода является то, что он сложнее и ограничивает долгосрочную масштабируемость. Еще одна проблема с настоящей кластеризацией состоит в том, что многие кластерные архитектуры основываются на многоадресной рассылке (multicasting) — а в Amazon EC2 она недоступна (но доступна в GoGrid и Rackspace).

Я являюсь убежденным сторонником подхода с использованием независимых узлов. Хотя для построения такой архитектуры приложения, которая будет рабо-

тать с использованием независимых узлов, потребуется определенный уровень знаний и навыков, основным преимуществом данной архитектуры является ее высокая масштабируемость. Весь "фокус" заключается в том, что ваше приложение управляется через совместно используемую базу данных, очередь сообщений или централизованное хранилище данных, из которого все узлы могут получать сведения о модификации состояния в режиме, приближающемся к режиму реального времени.

Ограничения по производительности в EC2

Системы EC2, в общем случае, демонстрируют неплохую производительность. Когда вы выбираете конкретную машину EC2, то CPU и RAM работают более или менее так же, как вы могли бы ожидать и от физического сервера. Сетевые скорости — вообще выдающиеся. Различия проявляются в производительности сети и скоростей ввода/вывода дисковой подсистемы.

> ***Внимание!***
>
> 32-битные экземпляры EC2 невероятно медленны. Эта низкая скорость является артефактом не столько сервиса EC2, как такового, сколько отражением объема ресурсов CPU, который вы получаете за сумму $0,10/CPU-час. Тем не менее, они полезны для таких сценариев, в которых производительность имеет меньшее значение, нежели затраты, а именно: создание прототипов, быстрая разработка, работа в условиях низких требований к CPU (например, балансировка нагрузки) и т. д.

Три типа хранилищ данных имеют три очень разнящихся профиля производительности.

- Блочное хранилище имеет в точности тот тип производительности, который вы могли бы ожидать от SAN или других приложений, работающих через соединение по гигабитному Ethernet. Это наиболее надежная модель производительности из всех доступных, и ее производительность действительно устойчиво высока.
- Хранилище S3 очень медленно (говоря относительно). Приложения, работающие в режиме реального времени, полагаться на него не должны.
- Производительность локального хранилища абсолютно непредсказуема. В общем случае первые записи в блок локального хранилища происходят медленней, чем последующие. Я в своей практике встречал как выдающиеся скорости чтения/записи на диск, так и скорости, даже худшие, чем наблюдаемые при монтировании диска WebDAV при соединении через 56K-модем[1]. Мне нигде не встретилось ни одного примера, который помог бы мне объяснить, как же оптимизировать производительность локального хранилища. Если вы наблюдаете чудовищно медленную производительность, то иногда проблему может

[1] См. **http://en.wikipedia.org/wiki/56_kbit/s_modem**. — *Прим. перев.*

решить перезагрузка экземпляра[1] (и этот факт следует принять во внимание при оценке времени работы и времени простоя).

Производительность дисковых операций в EC2 обслуживает нужды большинства транзакционных Web-приложений. Однако, если вы имеете дело с приложением, для которого скорость дискового ввода/вывода действительно имеет значение, вам лучше всего произвести оценку производительности вашего приложения в EC2 прежде, чем принимать решение и действительно начинать развертывание этого приложения в EC2.

Безопасность

Безопасность в облачной среде — это один из основных аспектов, на которые особенно "напирают" критики облачной инфраструктуры. Для многих людей перенос их данных в облачную среду на компьютеры, где они никак не могут их контролировать, кажется эмоционально дискомфортным. Кроме того, этот подход действительно требует рассмотрения вопросов соответствия стандартам и учета различных нормативно-правовых аспектов. Впрочем, в действительности облачная среда может быть защищена ничуть не хуже, а может быть, даже и лучше, чем традиционный центр обработки данных. Однако в случае работы с облачной средой ваш подход к информационной безопасности тоже будет радикально отличаться от того, к чему вы привыкли в традиционной среде.

Принимая решение о переходе на облачную обработку данных, вам следует рассмотреть целый ряд критических аспектов, касающихся информационной безопасности, в том числе:

- организационно-правовые аспекты, вопросы соответствия местному законодательству и стандартам в облачной инфраструктуре будут принципиально иными;
- в облаке Amazon нет такого понятия, как "периметр". Следовательно, политика безопасности, разработанная таким образом, чтобы держать в центре внимания защиту периметра, в облаке Amazon работать не будет. Вам не следует ориентироваться на такую политику даже в тех облачных инфраструктурах, которые поддерживают традиционную модель защиты периметра;
- хотя нигде не было опубликовано эксплойтов, направленных на компрометацию системы безопасности облачной инфраструктуры, при организации облачного хранилища данных следует придерживаться профиля безопасности с высокой степенью риска;
- технологии виртуализации, такие как Xen, могут иметь собственные уязвимости, что может привести к появлению новых направлений атаки.

[1] Экземпляр EC2 сохраняет свое состояние при перезагрузках. Перезагрузка — это не то же самое, что потеря экземпляра.

Сложности, связанные с законодательными, организационно-правовыми аспектами и вопросами стандартизации

К сожалению, законодательство и организации, отвечающие за разработку стандартов, несколько "отстали от жизни" в тех вопросах, которые касаются применения виртуализации. Многие законодательные акты и стандарты подразумевают, что любой сервер представляет собой некую физическую сущность. В большинстве случаев разница между физическим и виртуальным серверами не имеет особо важного влияния на дух того или иного закона или стандарта, но закон или стандарт, несмотря ни на что, указывает именно на физический сервер, поскольку на момент его разработки концепция виртуальных серверов не была широко распространена.

Прежде чем переходить к использованию облачных вычислений, вам необходимо четко уяснить себе все законы и стандарты, к которым "привязаны" ваши приложения и инфраструктура. По всей вероятности, облачный эквивалент вашей физической инфраструктуры будет поддерживать дух этих законов, стандартов и спецификаций, но в сомнительных случаях вам необходимо проконсультироваться с экспертами и получить однозначный ответ на вопрос о том, будет ли обеспечено соответствие всем этим нормам в облачной среде. В случаях, когда "чистая" облачная инфраструктура не соответствует вашим потребностям, вы почти всегда можете создать смешанную среду, которая позволит вам получать выгоду от использования облачных технологий, при одновременном соблюдении всех требований законов, стандартов и спецификаций.

Внимание!

Если вы будете искать в этой книге ответ на вопрос наподобие "Может ли облачная среда обеспечить соответствие спецификации X?", то вы будете разочарованы. Amazon работает над обеспечением соответствия SAS 70[1], а некоторые другие провайдеры услуг в области облачной обработки данных предоставляют среды, обеспечивающие поддержку некоторых других спецификаций. Вам все же потребуется управлять собственными экземплярами, стремясь обеспечить соответствие определенной целевой спецификации. В этой книге я не имею возможности ответить на все вопросы подобной направленности. Ответить на них вам сможет только сертифицированный юрист или выполнение формальной процедуры сертификации вашей конкретной инфраструктуры.

---

[1] SAS (Statement on Auditing Standards (SAS) No. 70 — широко известный американский стандарт операционного аудита, разработанный Американским институтом сертифицированных присяжных бухгалтеров (American Institute of Certified Public Accountants, AICPA). В России этот стандарт вызывает все возрастающий интерес в связи с интеграцией в мировую экономику. Проведение сертификации по SAS 70 стало своеобразной "визитной карточкой" на глобальном финансовом рынке и в целом на рынке услуг — для акционеров, а также клиентов и партнеров. Более подробную информацию см. по следующим адресам: **http://www.ndc.ru/common/img/uploaded/files/depo/42/08_10_ndc_naumov.pdf**, **http://www.sas70guide.com/**. — *Прим. перев.*

Помимо аспектов соответствия законам, стандартам и спецификациям, облачная среда поднимает и еще один вопрос: юридические проблемы, связанные с тем, где именно хранятся ваши данные.

- Ваши данные могут предоставляться для рассмотрения в судебные инстанции по повесткам из судов или быть использованы в других судебных действиях, в которых принимает участие ваш провайдер (а не вы) и которые могут требовать других процедур, в которые вы можете быть непосредственно вовлечены.
- В некоторых странах (в частности, в странах Евросоюза) законодательство предъявляет достаточно жесткие требования, указывающие, где и как должны храниться конфиденциальные данные.

В облачной среде нет периметра

Облачная среда Amazon не имеет такой концепции, как сетевой сегмент, контролируемый брандмауэрами, и предоставляет вам минимальные возможности мониторинга сетевого трафика. Каждый экземпляр "видит" только "свой" входящий и исходящий трафик, а изначальная политика безопасности, принятая по умолчанию для группы безопасности в EC2, блокирует весь входящий сетевой трафик.

Примечание

Другие облачные среды поддерживают концепции виртуальных локальных сетей и предоставляют возможности по традиционной защите периметра. Однако наиболее правильным подходом будет не слишком полагаться на защиту периметра и использовать ее как основной подход к безопасности системы, вне зависимости от того, какую облачную среду вы используете.

Если говорить совсем кратко, то для защиты облачной структуры вы не устанавливаете брандмауэров и не используете сетевую систему обнаружения вторжений (Intrusion Detection System, IDS) — да, я знаю, что в физической инфраструктуре защита периметра не сводится только к упомянутым мерам. В *главе 5* будет подробно рассказано, как эффективно защитить входящий и исходящий сетевой трафик для ваших виртуальных серверов без использования традиционных брандмауэров и других средств защиты сетевого периметра.

Профиль риска для S3 и других облачных хранилищ данных не апробирован

Хотя отсутствие традиционных средств защиты сетевого периметра и заставляет многих экспертов в области безопасности скептически относиться к облачным средам, размещение критических данных в центре обработки данных, находящемся под контролем сторонней фирмы, выглядит нисколько не лучше.

Когда вы пользуетесь облачной средой, вы никогда не можете с уверенностью сказать, где именно в облачной инфраструктуре располагаются ваши данные.

Тем не менее, вы обладаете информацией о следующих базовых параметрах.

- Ваши данные находятся в гостевой операционной системе, работающей на виртуальной машине Xen, или хранятся на томе EBS, и в вашем распоряжении имеются механизмы, с помощью которых вы можете получить доступ к этим данным.
- Сетевой трафик, которым обмениваются ваши виртуальные экземпляры, невидим для других виртуальных хостов.
- Хранилище S3 лежит в публичном пространстве имен, но доступ к нему осуществляется через объекты, которые по умолчанию являются частными.
- Amazon обнуляет эфемерное хранилище в промежутках времени, когда оно не используется.

Поэтому вы можете исходить из следующих предположений, касающихся защищенности ваших данных.

- За исключением возможности специфических эксплойтов, использующих специфические уязвимости виртуализации, данные, находящиеся внутри виртуальной машины, защищены ничуть не хуже, чем данные, хранящиеся на физической машине. Если вас действительно серьезно заботит безопасность ваших данных, вы можете воспользоваться зашифрованными файловыми системами.
- Ваши объекты S3 не обладают внутренне присущей защищенностью, поэтому все данные, которые действительно являются конфиденциальными, перед помещением их в хранилище S3 должны шифроваться с использованием криптостойких алгоритмов.
- Ваши блочные хранилища и моментальные снимки, возможно, являются вполне защищенными. Хотя они и хранятся в S3, доступ к ним через обычные протоколы S3 невозможен. Тем не менее, если проблема безопасности вас серьезно заботит, не помешает зашифровать ваши блочные тома.
- Вы должны убедиться в том, что имеете возможность восстановления контроля над данными средствами, отличными от средств вашего облачного провайдера на случай его банкротства или возникновения других форс-мажорных обстоятельств.

Ликвидация последствий чрезвычайных происшествий

Аварийное восстановление представляет собой возможность и умение возобновить нормальную работу системы после аварии, катастрофы или возникновения других форс-мажорных обстоятельств. Что считать катастрофой зависит от ваших конкретных обстоятельств. В общем случае я определяю катастрофу как любое аномальное событие, в результате которого нормальная работа вашей системы ос-

танавливается. В традиционном центре обработки данных, например, утрата жесткого диска катастрофой не является, потому что, скорее всего, это событие более или менее ожидаемое, и ликвидация его последствий представляет собой более или менее рутинную операцию. С другой стороны, пожар в центре обработки данных — это уже катастрофическое событие, потому что оно, скорее всего, вызовет не просто перебои в работе, а приведет к приостановке нормального функционирования всего центра обработки данных.

Внезапный и полный отказ целого сервера, который в физическом центре обработки данных вы могли бы счесть катастрофой, в облачной среде представляет собой обыденное событие. Хотя значимость такого события в облачной среде и "понижена в ранге", вам все же следует разработать процедуры аварийного восстановления на тот случай, если подобное событие произойдет. Как результат, аварийное восстановление — это не просто "хорошая идея", а одно из приоритетных направлений, которому нужно уделить должное внимание.

Основной фактор, который осложняет аварийное восстановление в физической среде — это объем ручных операций и трудозатраты, связанные с подготовкой и реализацией плана аварийного восстановления. Далее, следует учесть, что подготовка плана аварийного восстановления и его последующее тестирование — это довольно трудная и ответственная задача. Важность ее совершенно очевидна, но, к сожалению, слишком многие организации пострадали из-за того, что их план аварийного восстановления никогда не тестировался в средах, максимально приближенных к реальным условиям (что абсолютно необходимо для того, чтобы иметь полную уверенность, что ваш с таким трудом составленный план действительно будет работать, а не окажется совершенно бесполезным в случае, когда критическая ситуация все-таки наступит).

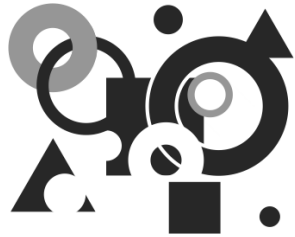
План аварийного восстановления в облачной среде можно в значительной мере автоматизировать. Фактически некоторые инструменты управления облачной инфраструктурой могут осуществить план аварийного восстановления, обойдясь даже без минимального вмешательства человека.

Что произойдет в традиционной физической инфраструктуре IT в случае утери всего центра обработки данных? Останется лишь надеяться на то, что у вас есть тщательно проработанная стратегия резервного копирования и хранения резервных копий за пределами территории предприятия. Если это так, тогда за несколько недель вы сможете запустить новый центр обработки данных. В облачной среде вектор развития таков, что скоро вы сможете в случае аварийной ситуации полностью перенести всю инфраструктуру от одного облачного провайдера к другому, и даже сделать это в полностью автоматическом режиме (хотя пока, конечно, этот уровень еще не достигнут).

Еще одно преимущество облачной обработки данных состоит в стоимости мероприятий по быстрому реагированию на катастрофические события. Ваши затраты на аварийное восстановление облачной инфраструктуры окажутся пренебрежимо малыми по сравнению с затратами, которые вам пришлось бы понести в

физической инфраструктуре. В сущности, они лишь немного превысят затраты, которые вы несете при нормальном функционировании вашей среды. В случае же с традиционным центром обработки данных вам придется понести новые капитальные затраты на создание новой инфраструктуры и надеяться на получение компенсационных выплат по страховке. Наконец, в облачной среде у вас есть все возможности реально протестировать различные сценарии аварийного восстановления в условиях, "максимально приближенных к боевым", что в традиционной инфраструктуре вряд ли возможно.

Итак, мы рассмотрели основные моменты, влияющие на архитектуру ваших приложений и серверные конфигурации, которые необходимо было принять во внимание при планировании перехода на облачные вычисления. В остальных главах этой книги мы рассмотрим практические аспекты этого перехода.



Глава 4

Подготовка к переходу на облачные вычисления

До известной степени всего один-единственный секрет поможет вам развернуть в облачной среде такое приложение, которое сможет в полной мере использовать все ее преимущества:

Действуйте так, как если бы вам требовалось построить в высшей степени масштабируемое[1] *Web-приложение.*

В случае отсутствия особых требований к соответствию конкретным стандартам или спецификациям, а также другим организационно-правовым нормативам, если ваше приложение способно работать за балансировщиком нагрузки на множестве серверов приложений (узлов), не испытывая никаких серьезных проблем, то вы уже почти у цели. В этой главе будет показано, как переместить такое приложение в облачную среду.

С другой стороны, многие Web-приложения были построены с учетом того, что они будут работать не в распределенной среде, а на одном сервере, и разработчики таких приложений не могут быть полностью уверены в том, что их продукт будет хорошо работать в кластере. Если ваше приложение подпадает под эту категорию (или если вы просто знаете, что конкретное нужное вам приложение не будет работать в кластерной среде), то в этой главе будет рассказано и о том, как переработать ваше приложение таким образом, чтобы его без опасений можно было переместить в облачную инфраструктуру.

Разработка Web-приложений

Возможно, я не смогу описать все сложности, связанные с использованием каждой из популярных среди разработчиков платформ разработки Web-приложений,

---

[1] Масштабируемость (scalability) — это способность информационной системы увеличивать свою производительность при добавлении ресурсов (обычно аппаратных). Масштабируемость — один из главных аспектов электронных систем, программных комплексов, баз данных, маршрутизаторов, сетей и т. п., если для них требуется возможность работать под большой нагрузкой. Подробнее см. **http://en.wikipedia.org/wiki/Scalability**, **http://www.insight-it.ru/net/scalability/masshtabiruemye-veb-arkhitektury/**. — *Прим. перев.*

но большинство проблем, с которыми вы столкнетесь, не имеют ничего общего с выбором платформы разработки. Не имеет абсолютно никакого значения, на чем вы будете писать свое приложение — .NET, Ruby, Java, PHP, или воспользуетесь чем-то еще, но общая архитектура для разрабатываемых Web-приложений будет примерно одинакова.

Типовая архитектура, которую использует большинство Web-приложений, представлена на рис. 4.1.

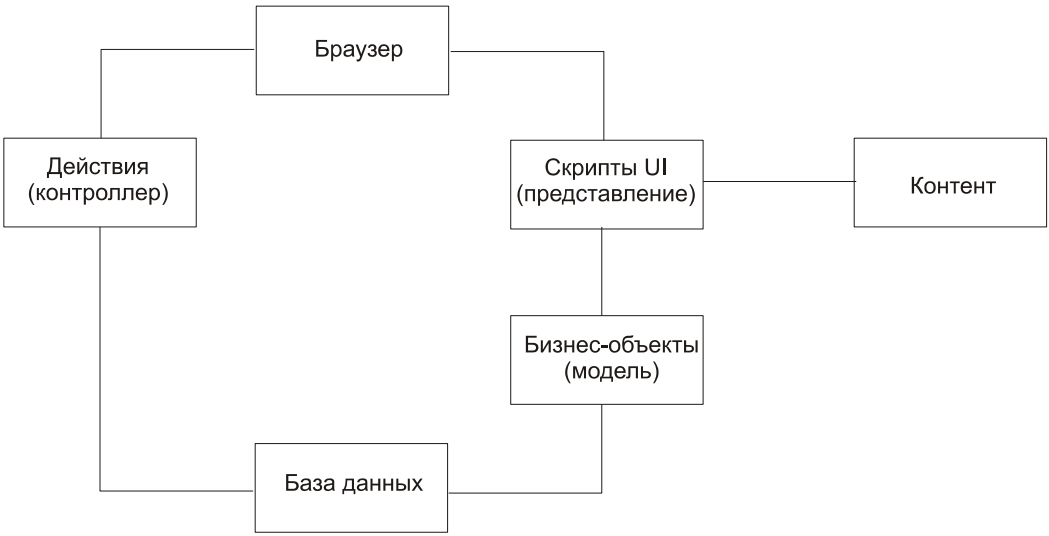


Рис. 4.1. Большинство Web-приложений использует примерно одинаковую базовую архитектуру

Возможно, элементы блок-схемы будут расположены несколько иначе и в других сочетаниях, но вы можете быть уверены, что в большинстве случаев вы должны будете использовать тот или иной язык (чаще всего язык скриптов), который будет генерировать контент на основе комбинации шаблонов и данных, извлеченных из модели, реализованной с помощью базы данных. Обновление системы будет осуществляться на основе модели путем действий, которые реализуют транзакции.

Состояние системы и защита транзакций

Определяющим фактором при перемещении приложения в облачную среду является способ, которым ваше приложение управляет своим состоянием. Рассмотрим эту концепцию на примере программы, используемой для бронирования номеров в отеле.

Архитектура, представленная на рис. 4.1, предполагает, что вы создали модель, описывающую отель и номера. Для целей нашего примера не имеет существенного

значения, представили ли вы модель, представление и данные отдельно или до некоторой степени допускаете их смешивание. Ключевым моментом здесь является то, что данные об отеле и номерах представлены в пространстве вашего приложения, которое отражает их соответствующие статусы в базе данных.

Каким образом ваше приложение реализует связь между изменениями, такими как время бронирования пользователем номера и осуществлением транзакции в базе данных?

Весь процесс может выглядеть примерно как следующая цепочка шагов:

1. Блокировка данных, ассоциированных с номером отеля.
2. Проверка статуса номера — свободен ли он на данный момент и может ли быть забронирован.
3. Если номер на данный момент доступен, необходимо пометить его как забронированный (booked) и, следовательно, более недоступный для заказа.
4. Снятие блокировки.

Проблема блокировок в памяти

Эту логическую последовательность можно реализовать множеством различных способов, не все из которых пригодны для использования в облачной среде. Типовой подход с использованием Java, который хорошо работает в односерверной среде, но не будет работать в многосерверном контексте, представлен в листинге 4.1.

Листинг 4.1. Типовой подход Java, реализующий последовательность действий по бронированию номера в отеле, но не подходящий для использования в облачной среде

```
public void book(Customer customer, Room room, Date[] days)
    throws BookingException {
    synchronized( room ) { // синхронизация блокировок объекта room
        if( !room.isAvailable(days) ) {
            throw new BookingException("Room unavailable.");
        }
        room.book(customer, days);
    }
}
```

Поскольку код, приведенный в листинге 4.1, использует ключевое слово блокировок Java `synchronized`, никакие другие потоки (threads) в текущем процессе не могут вносить изменения в объект `room`[1]. Если у вас — односерверная среда, то такой код будет без проблем работать под любой нагрузкой, которую может выдер-

[1] Я прошу программистов на Java простить мне это чрезмерное упрощение и технологически не совсем корректное описание ключевого слова `synchronized`. Это объяснение приводится здесь просто для того, чтобы помочь читателям понять в общих чертах все, что происходит в данном контексте, а не для того, чтобы подробно рассказывать им о тонкостях многопоточного программирования на Java.

живать данный сервер. К сожалению, в многосерверном контексте этот код работать не будет.

Проблема с этим примером заключается в блокировке в памяти, которую захватывает приложение. Если у вас два клиента, одновременно пытающихся сделать запрос на бронирование номера к одному и тому же серверу, то Java позволит выполнить синхронизирующую блокировку только одному из них, потому что в каждый конкретный момент времени может быть установлена только одна такая блокировка. За счет этого удается избежать таких ситуаций, когда один и тот же номер может оказаться забронирован сразу двумя клиентами.

С другой стороны, если ваши клиенты делают запросы на бронирование номера через разные серверы приложений (или даже через разные процессы на одном и том же сервере), то синхронизирующие блокировки на каждом из серверов могут быть выполнены параллельно. В результате этого первый клиент, который сделал вызов `room.book()`, потеряет свою бронь, потому что она будет перезаписана вторым клиентом. Проблему двойного бронирования иллюстрирует рис. 4.2.

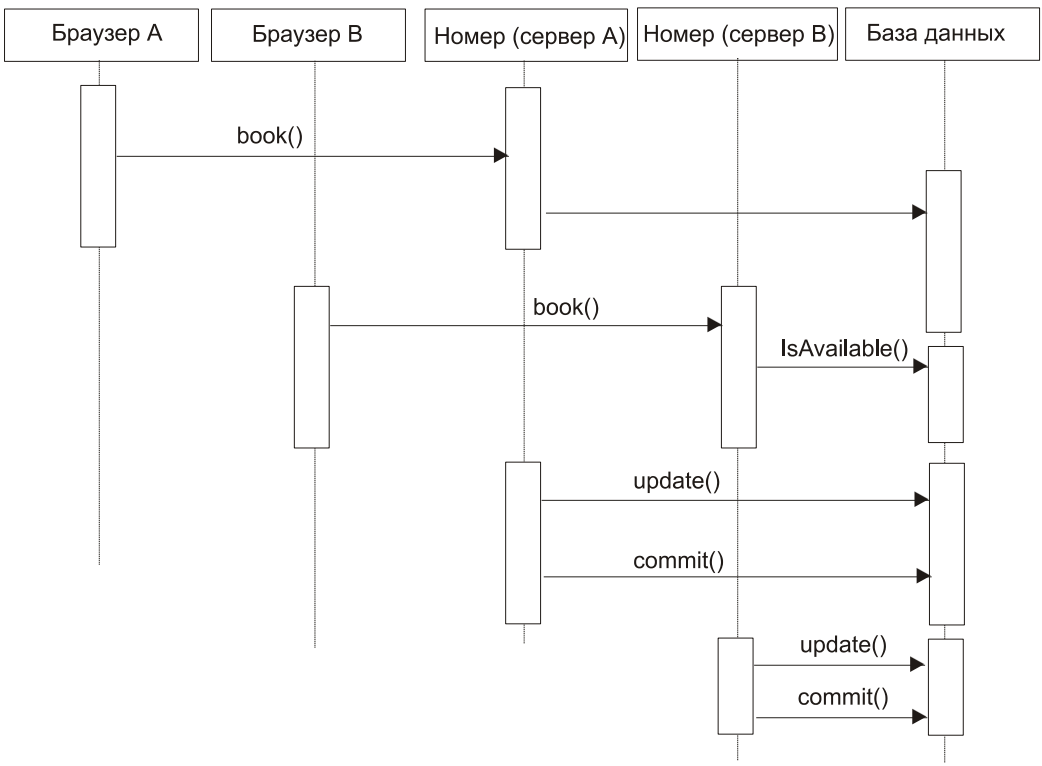


Рис. 4.2. Бронь первого клиента, сделавшего запрос на бронирование номера, будет перезаписана вторым клиентом (проблема двойного бронирования)

Если не применять при объяснении проблемы двойного бронирования подхода Java, то суть проблемы заключается в том, что если ваша логика реализации тран-

закций использует блокировку памяти для защиты целостности транзакций, то в многосерверной среде транзакция завершится неудачно. Из-за этого ваше приложение не сможет воспользоваться преимуществами облачной среды для динамического масштабирования приложения.

Одним из способов обойти эту проблему является использование кластерных технологий или системы, позволяющих серверам совместно использовать память. Второй подход к проблеме заключается в том, чтобы рассматривать базу данных как арбитра, управляющего статусом вашей системы.

Что делать, если мое приложение использует блокировки памяти?

Я уже могу представить себе, что сейчас многие из вас начнут обо мне говорить (особенно те, чьи приложения в широком масштабе используют многопоточность). Но не спешите возмущаться. Если вы тоже оказались в такой ситуации, когда ваши приложения используют блокировки в памяти, и переработка такого приложения не представляется возможной, вы все равно можете мигрировать в облачную инфраструктуру. Правда, вы не сможете масштабировать ваше приложение, распределив его по множеству серверов.

Выход из этого положения заключается в том, чтобы блокировать абсолютно все с помощью разделяемого механизма блокировки — как правило, это движок вашей базы данных (database engine). Еще один вариант состоит в самостоятельном написании распределенной системы управления транзакциями. Правда, зачем это делать, если в вашей базе данных такая система уже имеется?

Защита целостности транзакций с помощью хранимых процедур

Я не фанат хранимых процедур. Однако ключевым преимуществом хранимых процедур является то, что они позволяют вам использовать базу данных для управления целостностью ваших транзакций. В конце концов, ведь целостность данных и есть основная задача, ставящаяся перед движком вашей базы данных!

Вместо реализации всей логики бронирования на Java вы можете воспользоваться хранимой процедурой MySQL, как показано в листинге 4.2.

Листинг 4.2. Пример использование хранимой процедуры MySQL для защиты целостности транзакций

```
DELIMITER |

CREATE PROCEDURE book

(
    IN customerId BIGINT,
    IN roomId BIGINT,
    IN startDate DATE,
    IN endDate DATE,
```

```
    OUT success CHAR(1)
)

BEGIN
    DECLARE n DATE;
    DECLARE cust BIGINT;
    SET success = 'Y';
    SET n = startDate;

    bookingAttempt:

    REPEAT
        SELECT customer INTO cust FROM booking
        WHERE room_id = roomId AND booking_date = n;
        IF cust IS NOT NULL AND cust <> customerId
        THEN

            SET success = 'N';
            LEAVE bookingAttempt;
        END IF;
        UPDATE booking SET customer = customerId
        WHERE room_id = roomId AND booking_date = n;
        SET n = DATE_ADD(n, INTERVAL 1 DAY);
    UNTIL n > endDate
    END REPEAT;
    IF success = 'Y' THEN
       COMMIT;
    ELSE
        ROLLBACK;
    END IF;
END
|
```

Этот метод просматривает каждую запись в таблице бронирований `booking` вашей базы данных на MySQL и выбирает номер, который помечен как забронированный конкретным клиентом. Далее производится проверка дат бронирования номера — если процедура встречает дату, на момент наступления которой номер уже был ранее забронирован другим клиентом, то транзакция заканчивается неудачей, и производится ее откат.

Пример использования хранимой процедуры с помощью Python приведен в листинге 4.3.

Листинг 4.3. Пример использования хранимой процедуры с помощью Python

```
def book(customerId, roomId, startDate, endDate):
    conn = getConnection();
```

```
c = conn.cursor();
c.execute("CALL book(%s, %s, %s, %s, @success)", \
         (customerId, roomId, startDate, endDate));
c.execute("SELECT @success");
row = c.fetchone();
success = row[0];
if success == "Y":
   return 1
else:
   report 0
```

Даже если вы используете два сервера приложений, на каждом из которых работает по экземпляру вашего приложения, написанного на Python, то теперь эта транзакция, как и должно быть, завершится неудачей для второго клиента, независимо от того временного момента, когда началась эта вторая транзакция.

Две альтернативы хранимым процедурам

Как я уже замечал ранее, я отнюдь не являюсь фанатом использования хранимых процедур. Конечно, они обладают тем преимуществом, что исполняются быстрее, нежели та же самая программная логика, реализованная на том языке программирования, на котором написано приложение. Далее, управление транзакциями в многосерверной среде посредством хранимых процедур представляет собой очень элегантное решение. Но у меня есть три ключевых возражения против этого подхода.

- Хранимые процедуры нельзя переносить из одной базы данных в другую (иначе говоря, невозможно их портирование).
- Чтобы грамотно пользоваться хранимыми процедурами, необходимо обладать глубоким пониманием принципов программирования баз данных — не во всех коллективах можно найти специалиста с такой квалификацией.
- Хранимые процедуры не полностью решают проблему масштабирования транзакций по серверам приложений при различных сценариях. При написании приложений вам все равно придется заботиться о том, чтобы они грамотно использовали хранимые процедуры, а в результате этого код вашего приложения может чрезмерно усложниться.

В дополнение к этим базовым возражениям я лично предпочитаю четкое разделение между представлением (presentation), бизнес-моделированием (business modeling), реализацией бизнес-логики (business logic) и собственно данными (data).

Последнее из моих возражений субъективно, и я допускаю, что некоторые могут посчитать его просто "дурацким личным заскоком". Но первые два возражения представляют реальные проблемы. Подумайте сами — многие ли из вас вынуждены использовать приложения, работающие с Oracle, которые легко можно было бы перевести на работу с MySQL, если бы не хранимые процедуры? Вы будете вынуждены платить Oracle немалые деньги — а все из-за того, что ваше приложение использует хранимые процедуры!

Второе возражение несколько более спорно. Конечно, если в вашей компании работает большая команда квалифицированных разработчиков разнообразной специализации, то этот фактор не будет представлять большой проблемы — вы можете ее даже не заметить. Но ваше предприятие — небольшая компания, где каждый сотрудник должен быть универсалом и выполнять различные функции, разработка такой архитектуры приложения, при которой в составе команды разработчиков вам не требуется эксперт в области программирования баз данных, может оказаться преимуществом.

Чтобы сохранить программную логику на уровне сервера приложений, но при этом одновременно обеспечить защиту целостности транзакций в многосерверной среде, вам необходимо либо реализовать защиту от *феномена "грязной записи"* (dirty writes)[1], либо создать блокировку в базе данных.

В главе 3 книги [2] я подробно описал политику управления транзакциями для систем, написанных на Java и управляющих логикой транзакций на уровне сервера приложений. Одним из приемов, которые я описал в упомянутой книге и который в общем случае настоятельно рекомендую применять, независимо от общей архитектуры вашего приложения, благодаря скоростным преимуществам этого метода, является использование *временной метки последнего обновления* (last update timestamp) и *агента модификации* (modifying agent) при выполнении обновлений.

Логика бронирования из хранимой процедуры, в сущности, представляет собой обновление таблицы бронирований (`booking`):

```
UPDATE booking SET customer = ? WHERE booking_id = ?;
```

Если вы добавите поля `last_update_timestamp` и `last_update_user`, SQL будет эффективнее работать в многосерверной среде. Пример использования этого приема показан в листинге 4.4.

Листинг 4.4. Добавление полей `last_update_timestamp` и `last_update_user` с целью повышения эффективности работы SQL в многосерверной среде

```
UPDATE booking
SET customer = ?, last_update_timestamp = ?, last_update_user = ?
WHERE booking_id = ? AND last_update_timestamp = ? AND last_update_user = ?;
```

[1] В идеальном случае результат выполнения транзакции не должен зависеть от остальных транзакций, сколько бы транзакций ни выполнялось параллельно. Но, к сожалению, этот идеальный случай накладывает сильные ограничения на параллельную обработку, практически выстраивая транзакции в очередь. Однако в большинстве случаев такая строгость не нужна, и поэтому были введены так называемые "уровни изоляции" (Isolation Level), которые определяют степень параллелизма выполнения транзакций. Чем ниже уровень изоляции, тем выше степень параллелизма и тем больше риск "неправильного" выполнения транзакции. В стандарте ANSI SQL уровни изоляции привязаны к четырем "феноменам" — нарушениям изолированности транзакций: грязная запись (dirty write), грязные чтения (dirty reads), неповторяющиеся чтения (Non-repeatable Read или Fuzzy Read) и фантомы (Phantom). В частности, феномен грязной записи можно описать так: представьте себе, что некоторая транзакция `T1` модифицирует некий элемент данных. После этого другая транзакция `T2` тоже модифицирует этот элемент данных, и делает это до того, как транзакция `T1` выполнит `COMMIT` или `ROLLBACK`. Если `T1` или `T2` после этого выполнит `ROLLBACK`, то становится непонятным, каким должно быть корректное значение данных. Подробнее об этом см. **http://www.rsdn.ru/article/db/deadlocks.xml**. — *Прим. перев.*

В данной ситуации первый клиент попытается забронировать номер в отеле на указанную дату, и его транзакция будет успешной. Теперь представим себе, что второй клиент пытается обновить эту запись. Этот второй клиент не получит совпадений, потому что временная метка (timestamp), которую он прочитает, как и идентификатор пользователя (user ID), не совпадут со значениями, обновленными первым клиентом. Его транзакция завершится неудачей, потому что он увидит, что не обновлено ни одной записи и появилось сообщение об ошибке. И никакого больше двойного бронирования!

Этот подход хорошо работает до тех пор, пока вы не сталкиваетесь со структурирующими транзакциями таким образом, что это вызывает появление *взаимоблокировок* (deadlocks). Взаимоблокировка происходит между двумя транзакциями, когда каждая из них ждет, чтобы другая сняла блокировку. Наша система бронирования номеров в отелях представляет собой приложение, в котором взаимоблокировки вполне возможны.

Поскольку номер бронируется на определенный период времени (диапазон дат), в случае плохо структурированной логики приложения возможно возникновение такой ситуации, когда два клиента будут бесконечно ожидать друг друга (например, один из них пытается забронировать номер на дату, на которую номер уже забронирован другим клиентом, и наоборот). Представьте себе ситуацию, когда два клиента (допустим, вы и я) пытаются забронировать номер на вторник и среду, но, по непонятной причине, ваш клиент сначала пытается забронировать сначала на вторник, а мой — сначала пробует среду. У нас получится взаимоблокировка, когда я буду ждать, чтобы вы подтвердили свое бронирование на среду, а вы, соответственно, будете ждать, пока я дам подтверждение своего бронирования на вторник.

Проблему с этим несколько надуманным сценарием легко устранить, гарантировав, что проверка дат должна вестись последовательно по возрастанию дат. Но возможны и другие ситуации, для которых уже нельзя будет найти такое простое решение.

Еще один вариант заключается в создании поля, которое позволит управлять вашими блокировками. Например, в таблицу `room` можно добавить два дополнительных столбца для целей бронирования номеров: `locked_by` и `locked_timestamp`. Прежде чем начинать транзакцию для бронирования номера, необходимо обновить таблицу `room` и подтвердить изменение (`commit`). После того как транзакция бронирования завершится, снимите блокировку, обнулив эти поля перед подтверждением этой транзакции.

Поскольку этот подход требует проведения двух транзакций над базой данных, вы более не будете выполнять бронирование номера как единую и неделимую транзакцию (atomic transaction). Следовательно, вы рискуете оставить открытую блокировку, которая не позволит никому бронировать номера ни на какие даты. Существуют два способа решения этой проблемы.

- Номер считается разблокированным не только когда поля `locked_by` и `locked_timestamp` имеют значение `NULL`, но и когда поле `locked_timestamp` слишком долго остается открытым.

- При обновлении блокировки в конце транзакции бронирования номера использовать поля `locked_by` и `locked_timestamp` в составе предложения `WHERE`. Таким образом, если кто-то другой "украдет" у вас блокировку, вы в результате всего лишь выполните откат вашей транзакции.

Очевидно, что оба этих подхода сложнее, чем использование хранимых процедур. Вне зависимости от того, какой подход вы используете, ключевым моментом при переходе на облачные вычисления является гарантирование того, что для поддержания целостности статуса вашего приложения вы не полагаетесь на блокировки в памяти.

Что происходит при сбоях серверов

Основная цель облачной архитектуры заключается в создании и поддержании такой среды, в которой "падение" любого из серверов приложений, в принципе, большого значения не имеет. Если у вас есть только один сервер, то его сбой, очевидно, будет иметь определенное значение, но это событие все равно будет менее значимым, нежели утрата физического сервера.

Чтобы обойти проблемы, описанные в предыдущем разделе, иногда применяют такой прием, как *сегментация данных* (data segmentation), известный также как разделение данных на уровне ресурсов или *шардинг* (sharding)[1]. На рис. 4.3 показан пример сегментации данных с целью распределения обработки по множеству серверов приложений.

Иными словами, каждый сервер приложений будет управлять подмножеством данных. В результате этого вы устраняете риск того, что какой-нибудь другой сервер выполнит перезапись данных. Хотя сегментация и нашла применение в масштабируемых приложениях, но она не должна применяться на серверах приложений в облачном кластере. Дело в том, что сегментированный сервер приложений получит очень низкий рейтинг доступности, потому что сбой любого отдельного сервера приложений имеет значение.

Наконец, заключительный вывод ко всему сказанному о статусе приложения и о сбоях серверов будет звучать так: серверы приложений в облачной среде не могут хранить никаких данных о статусе кроме кэшированных данных. Иными словами, если вы должны выполнять резервное копирование сервера приложений, это значит, что вам не удалось разработать надежную архитектуру приложения для работы в облачной инфраструктуре.

---

[1] Шардинг (sharding) — это разделение данных на уровне ресурсов. Концепция шардинга заключается в логическом разделении данных по различным ресурсам исходя из требований к нагрузке. Подробнее см. **http://www.codefutures.com/database-sharding/**, **http://tinyurl.com/yanw837**, **http://tinyurl.com/2vbkhhv**. — *Прим. перев.*

Вся информация о статусе, включая двоичные данные, принадлежит базе данных, которая должна находиться в постоянном хранилище[1].

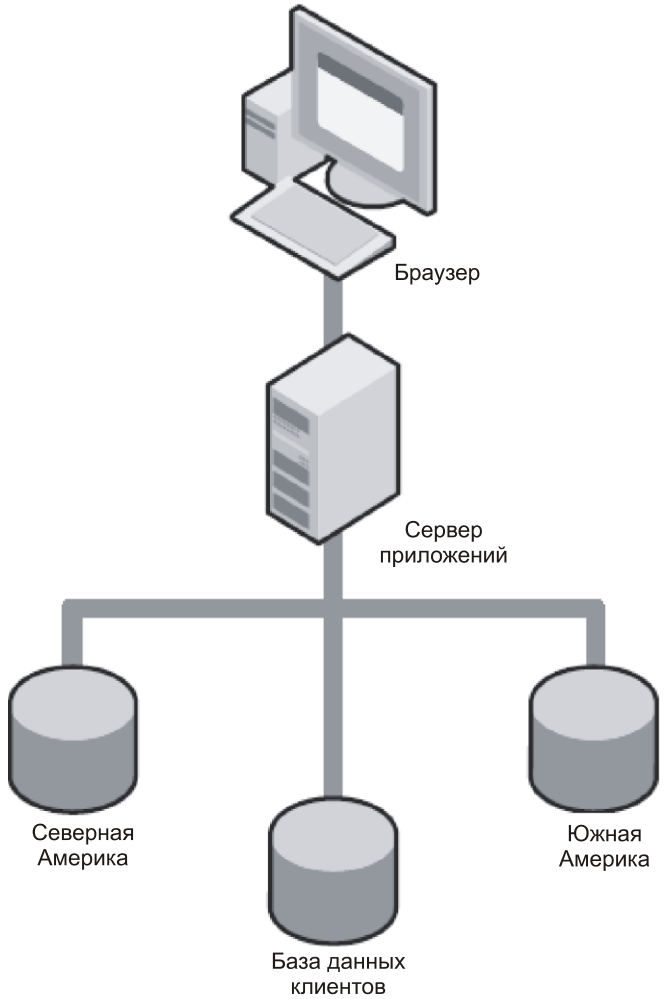


Рис. 4.3. Поддержание различных отелей на различных серверах позволяет избежать проблемы двойного бронирования

[1] Если вы читали мою статью *"Ten MySQL Best Practices"* (**http://www.onlamp.com/pub/a/onlamp/2002/07/11/MySQLtips.html**), то вы наверняка заметите противоречие между тем, что я не советовал хранить двоичные данные в MySQL, и тем, что я рекомендую здесь. Вы действительно должны кэшировать двоичные данные на сервере приложений, чтобы вам не требовалось извлекать (pull) их с сервера базы данных в режиме реального времени. Поступая так, вы обходите мои возражения против хранения двоичных данных в MySQL.

Построение образов машин

Двумя косвенными преимуществами облачных вычислений являются:

- переход на облачную инфраструктуру требует и, можно сказать, в принудительном порядке устанавливает дисциплину в планировании развертываний;
- облачные вычисления также принудительным образом требуют дисциплину в устранении последствий катастрофических сбоев системы.

Благодаря тому способу, с помощью которого виртуализованные серверы запускаются с образов машин, вашим первым шагом при подготовке к переходу на облачную модель обработки данных является создание повторяемого процесса развертывания, который управляет всеми вопросами, которые могут встать перед вами в процессе запуска системы. Чтобы гарантировать успешный запуск, вам необходимо выполнить некоторое предварительное планирование развертывания.

Образ машины (в случае использования сервисов Amazon образ машины называется Amazon Machine Image, AMI) представляет собой так называемую "сырую" (raw) копию вашей операционной системы и основного программного обеспечения, предназначенную для работы в конкретном окружении на конкретной платформе. Когда вы запускаете виртуальный сервер, он копирует вашу рабочую среду с образа машины и загружает ее. Если ваш образ машины содержит ваше установленное приложение, то развертывание представляет собой не что иное, как процедуру запуска нового виртуального экземпляра.

Безопасность данных образа машины Amazon

Когда вы создаете образ машины Amazon, он зашифровывается и сохраняется в комплекте Amazon S3. Для последующей расшифровки вашего AMI может использоваться один из следующих двух ключей:

- ваш ключ Amazon;
- ключ, держателем которого является Amazon.

Доступ к вашему AMI можно получить только с помощью ваших учетных данных пользователя. Amazon требуется способность расшифровки AMI с тем, чтобы запустить виртуальный экземпляр с образа машины.

Не храните в составе AMI никакой конфиденциальной информации

Даже несмотря на то, что ваш AMI зашифрован, я настоятельно не рекомендую хранить в его составе никакой конфиденциальной информации. Дело не только в том, что Amazon имеет теоретическую возможность расшифровать AMI, но и в том, что существуют механизмы, позволяющие сделать ваш AMI общедоступным. Вследствие этого вы, неосознанно и сами того не желая, можете по ошибке сделать общедоступными любые конфиденциальные данные, которые включены в состав AMI.

Представьте себе такую ситуацию: одна компания, пользующаяся сервисами Amazon, подает в суд на другую компанию, которая тоже является клиентом Amazon. Суд может затребовать предоставления данных клиента Amazon для изучения. К сожалению, не-

редко суды выходят за рамки благоразумия и здравого смысла, требуя от таких компаний, как Amazon, предоставления данных всех клиентов. Если вы хотите, чтобы ваши данные никогда не были раскрыты в процессе судебного разбирательства между двумя сторонними компаниями, ни одна из которых никак не связана с вами, никогда не храните свою информацию в Amazon AMI.

Вместо этого вам следует отдельно зашифровать эти данные и отдельно загружать их в ваш экземпляр во время его запуска. При таком подходе у Amazon не будет ключей для расшифровки, и доступа к данным не будет ни у кого (за исключением того случая, когда вы сами являетесь одной из сторон, вовлеченных в судебный процесс).

Что входит в состав образа машины?

Образ машины должен включать в свой состав все программное обеспечение, необходимое для работы в среде времени исполнения виртуального экземпляра, основанного на этом образе, и ничего сверх этого. Начальной точкой, очевидно, является операционная система, но выбор компонентов является абсолютно критическим. Весь процесс формирования образа машины состоит из следующих шагов:

1. Создание компонентной модели, которая идентифицирует, какие компоненты и версии необходимы для работы сервиса, который будет поддерживать новый образ машины.
2. Выделение из компонентной модели данных о статусе. Эти данные должны храниться за пределами вашего образа машины.
3. Идентификация операционной системы, которую вы собираетесь развертывать.
4. Поиск существующего, опубликованного в открытом доступе и заслуживающего доверия образа машины для выбранной операционной системы.
5. Усиление защищенности вашей системы с помощью инструмента наподобие Bastille.
6. Установка всех компонентов вашей компонентной модели.
7. Проверка работоспособности виртуального экземпляра с использованием образа машины.
8. Построение и сохранение образа машины.

Начальной точкой в процессе должно стать выяснение точного списка компонентов, необходимых вам для работы вашего сервиса. На рис. 4.4 показан пример модели, описывающей компоненты времени исполнения для сервера базы данных MySQL.

В данном случае данные о статусе существуют в каталоге MySQL, который монтируется как внешнее устройство блочного хранения данных. Следовательно, вам понадобится убедиться в том, что скрипты, исполняемые при запуске, монтируют устройство блочного хранения данных до запуска MySQL.

Так как подразумевается, что данные о статусе находятся на устройстве блочного хранения данных, этот образ машины полезен для запуска любых баз данных MySQL, а не только конкретного их набора.

Сервисы, которые вам нужно будет запускать на экземпляре, обычно диктуют и выбор операционной системы, на которой будет базироваться ваш образ машины.

Если вы осуществляете развертывание приложения .NET, вы, вероятнее всего, предпочтете Windows-образы Amazon. С другой стороны, для приложения PHP лучше подойдет среда Linux. В любом случае я рекомендую начать с наиболее надежных базовых AMI, готовых к употреблению, рассчитанных на выбранную вами операционную систему, и строить свои собственные образы, модернизируя эти предварительно собранные AMI. Более подробное обсуждение технических деталей создания AMI приводилось в *главе 2*.

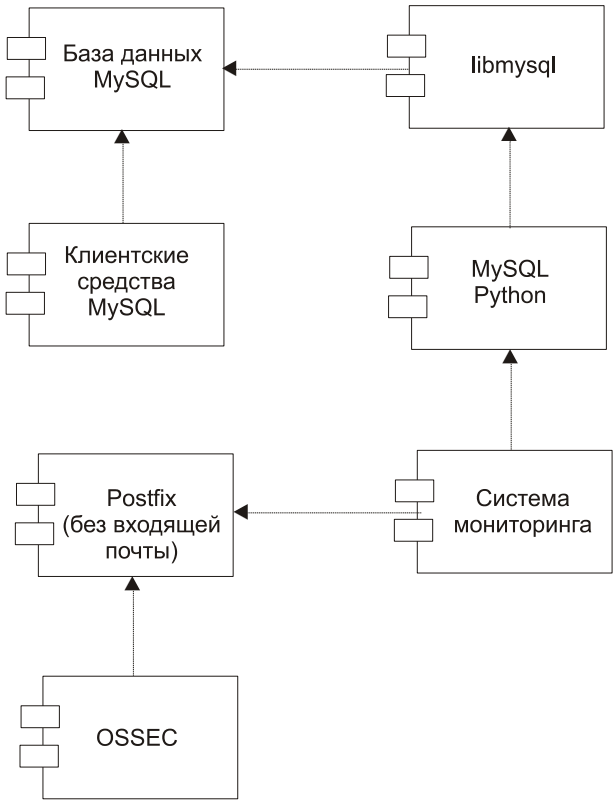


Рис. 4.4. Программное обеспечение, необходимое для поддержания работы сервера базы данных MySQL

Внимание!

Избегайте использования дистрибутивов Linux, содержащих "почти все, что надо", но плохо защищенных. Любой из используемых образов должен представлять собой операционную систему с усиленной безопасностью и содержащий только тот инструментарий, который абсолютно необходим для выполнения системой своих функций.

Усиление безопасности операционной системы направлено на минимизацию векторов атаки на ваш сервер.

Помимо прочего, усиление защищенности операционной системы включает следующие действия:

- удаление ненужных сервисов;
- удаление ненужных учетных записей;
- запуск всех сервисов в рамках учетной записи, выполняющей определенную роль и отличной от `root` (по возможности);
- запуск всех сервисов в "тюрьме строгого режима" (restricted jail)[1] всегда, когда это только возможно;
- верификацию корректности полномочий для необходимых системных сервисов.

Наилучший способ укрепления безопасности системы Linux заключается в использовании надежного и хорошо апробированного средства повышения безопасности, например Bastille. Более детально защита и укрепление безопасности в облачных средах будут обсуждаться в *главе 5*.

Теперь, когда у вас уже есть защищенная база, с которой можно начинать работу, наступает время фактически приступить к установке программного обеспечения, которое будет поддерживать эта система. В рассматриваемом примере наступает момент для установки MySQL.

При установке сервисов, специфичных для конкретного сервера, вам может понадобиться изменить свой подход к развертыванию вследствие того, что данные о статусе должны храниться за пределами образа машины. Для сервера MySQL вы, вероятно, будете хранить данные о статусе на устройстве блочного хранения данных, и монтировать его при запуске системы. Web-сервер, с другой стороны, может хранить данные о статуе мультимедийных ресурсов в облачном хранилище, таком как Amazon S3, и извлекать их оттуда при запуске экземпляра для работы с ними во время исполнения.

Различные приложения, определенно, требуют различных подходов, выбор которых осуществляется на основе их уникальных требований. Какой бы ни была ситуация, вам необходимо структурировать свою процедуру развертывания таким образом, чтобы образ машины вел себя достаточно интеллектуально для того, чтобы искать и находить необходимые данные о статусе при запуске и предоставлять эти данные компонентам образа машины в тот момент, когда они нуждаются в этой информации.

Как только вы структурируете процедуру развертывания нужным образом, вам необходимо будет ее протестировать. Это означает, что необходимо протестировать запуск системы, а также ее останов и аварийное восстановление. Следовательно, вам необходимо будет выполнить следующие шаги:

1. Построить временный образ на основе вашего разрабатываемого экземпляра.
2. Запустить новый экземпляр на основе временного образа.
3. Убедиться в том, что он функционирует так, как и было задумано.

---

[1] "Тюрьма" (jail) представляет собой набор ограничений на ресурсы, который налагает на программы ядро операционной системы. Это могут быть ограничения по полосе ввода/вывода, ограничения сетевого доступа и ограниченное пространство имен файловой системы. "Тюрьмы" обычно используются в виртуальном хостинге. Подробнее см. **http://en.wikipedia.org/wiki/Sandbox_(computer_security)**, **http://tinyurl.com/3yb387g**. — *Прим. перев.*

4. Устранить любые проблемы при их возникновении.
5. Повторять перечисленные шаги до тех пор, пока процесс не станет устойчивым и надежным.

На определенном этапе вы получите работающий экземпляр на основе хорошо структурированного образа машины. После этого вы можете построить окончательный экземпляр и пойти вознаградить себя бокалом пива (ну, или чашечкой кофе, в зависимости от вкусов).

Пример образа машины MySQL

Основная сложность в создании образа машины, который поддерживает серверы базы данных, заключается в том, что для этого необходимо знать, как выбранный вами движок базы данных (database engine) хранит данные. В случае с MySQL движок базы данных имеет каталог данных для хранения информации о статусе. Этот каталог данных в действительности может называться по-разному (например, `/usr/local/mysql/data`, `/var/lib/mysql` и т. д.), но это единственное, что, кроме конфигурационного файла, должно быть отделено от образа машины. В типичной индивидуализированной сборке каталог данных располагается в каталоге `/usr/local/mysql/data`.

Примечание

Если вы собираетесь поддерживать большое количество образов машин, часто бывает полезным сначала построить образ машины с усиленной безопасностью, без сервисов, а затем построить на основе этого образа другие, каждый из которых будет ориентирован на конкретный сервис.

Как только вы запустите экземпляр со стандартного образа и укрепите его безопасность, вам потребуется создать том для эластичного блочного хранения и примонтировать его[1]. Стандартный подход Amazon заключается в монтировании тома от `/mnt` (например, `/mnt/database`). В принципе, с технической точки зрения точка монтирования значения не имеет, но если для каждого образа монтирование производится к тому же самому каталогу, то это может помочь уменьшить путаницу.

После этого вы можете установить MySQL, причем необходимо делать это в пределах корневой файловой системы экземпляра (т. е. `/usr/local/mysql`). На данном этапе переместите данные через блочное устройство, выполнив следующие шаги:

1. Остановите MySQL, если процесс инсталляции автоматически запустил сервер базы данных.
2. Переместите ваш каталог данных в точку монтирования и дайте ему имя, более подходящее для монтирования отдельного устройств (например, `/mnt/database/mysql`).
3. Отредактируйте ваш файл `my.cnf` таким образом, чтобы он указывал на новый каталог данных.

[1] Этот процесс я описал на примере Amazon EC2 в *главе 2*.

Теперь вам требуется решить любопытную задачку: MySQL не сможет запуститься до тех пор, пока не будет примонтировано блочное устройство, но блочное устройство в Amazon EC2 не может быть подключено к экземпляру виртуальной машины до тех пор, пока этот экземпляр работает. В результате вы не сможете запустить MySQL, следуя нормальным процедурам загрузки. Однако вы можете оказаться в такой ситуации, когда вы хотите принудительным образом установить необходимый порядок событий: загрузить виртуальную машину, примонтировать устройство, и, наконец, запустить MySQL. Поэтому вам необходимо осторожно изменить ваши пусковые скрипты MySQL, таким образом, чтобы система больше не запускала MySQL в процессе загрузки системы, но, тем не менее, останавливала движок MySQL при останове системы.

Внимание!

Не следует просто вырезать MySQL из скриптов, управляющих запуском. Если вы это сделаете, MySQL не сможет быть корректно остановлен при останове вашего экземпляра сервиса. Таким образом, вы столкнетесь с проблемой поврежденной базы данных на вашем устройстве блочного хранения.

Наилучший подход к внесению этого изменения заключается в редактировании скрипта, управляющего запуском MySQL, таким образом, чтобы он дожидался того момента, когда станет доступным каталог данных MySQL, и только после этого запускал исполняемый файл MySQL.

Философия Amazon AMI

Выбирая подход к конструированию AMI, вы можете выбрать один из двух основных подходов:

- минималистский подход, при котором вы строите несколько многоцелевых образов машин;
- детальный подход, при котором вы строите множество специализированных образов машин.

Я — убежденный сторонник минималистического подхода. Минималистический подход обладает тем преимуществом, что он упрощает развертывание патчей безопасности и других модификаций системного уровня. Но его оборотной стороной является то, что он требует более тщательного планирования и более фундаментальных навыков в области EC2. Структурирование многоцелевого AMI, способного определять свою функцию после запуска и самоконфигурироваться для поддержки именно этой функции представляет собой более сложную задачу. Если вы только начинаете работать с EC2, то, возможно, сначала следует попробовать детальный подход и воспользоваться инструментами управления облачной инфраструктурой, и только впоследствии, приобретя необходимый опыт, приступать к разработке библиотеки минималистических образов машин.

Если вы начинаете с инсталляции единственного приложения, вам, скорее всего, не потребуется иметь большое количество машинных образов, и поэтому разница между детальным и минималистичным подходами будет пренебрежимо малой. Приложения SaaS, особенно те, которые не являются многоарендными (multitenant), требуют развертывания приложений во время исполнения.

Развертывание во время исполнения означает закачку прикладного программного обеспечения, например, такого как исполняемый файл MySQL, обсуждавшийся в предыдущем разделе, на только что запущенный виртуальный экземпляр после его запуска, вместо встраивания его в образ машины. Развертывание приложений во время исполнения является более сложной процедурой (и, следовательно, требующей применения инструментов управления облачной средой), нежели простое встраивание приложения в машинный образ, но зато этот подход обладает и рядом преимуществ.

- Вы можете развертывать и удалять приложения с виртуального экземпляра в процессе его работы. В результате в среде, где развернуто множество приложений, вы с легкостью сможете перемещать приложения из кластера в кластер.
- Вы сможете добиться автоматического восстановления приложений. Обычно развертывание приложения происходит во время исполнения с использованием образа последней резервной копии. С другой стороны, если вы встраиваете приложение в образ, то успех или неудача его запуска зависит от того, насколько хороша была последняя сборка образа машины.
- Вы можете избежать хранения верификационной информации для аутентификации одного сервиса другим в составе образа машины. Вместо этого вы можете переместить эту информацию в состав зашифрованной резервной копии, с которой производится развертывание приложения.

Проектирование системы защиты конфиденциальной информации

В *главе 5* я поговорю обо всех аспектах безопасности и облачной среды. Поскольку в этой главе мы обсуждаем общую архитектуру приложения, важно рассмотреть, какой подход к разработке архитектуры приложения нужно принять для систем, в которых имеется особый сегмент, предназначенный для хранения конфиденциальных данных, в частности системах электронной коммерции, в которых должна храниться информация о кредитных карточках и медицинских системах, в которых хранится информация, представляющая собой врачебную тайну. Здесь мы проведем краткий обзор подходов к организации хранения конфиденциальной информации, а более полное и детальное обсуждение нас еще ждет.

Конфиденциальность в облачной среде

Ключом к конфиденциальности в облачной среде, как, впрочем, и в любой другой, является четкое разделение между конфиденциальными данными и неконфиденциальными данными, с последующим шифрованием тех элементов данных, которые содержат конфиденциальную информацию. Характерным примером является система электронной коммерции, в которой должна храниться информация о кредитных картах клиента. Вы можете иметь довольно сложное по своей структуре приложение, реализующее систему электронной коммерции, хранящую в своем составе множество сведений об отношениях между различными элементами данных, но вам необходимо отделить информацию о кредитных картах клиентов от всей остальной информации, потому что это необходимый шаг, позволяющий приступить к построению безопасной инфраструктуры для построения системы электронной коммерции.

Примечание

Когда я говорю о том, что вам необходимо разделить данные, я имею в виду то, что доступ к любому из двух сегментов данных не может скомпрометировать безопасность системы и нарушить конфиденциальность данных. В случае с системой обработки кредитных карт вам необходимо хранить номера кредитных карт на другом виртуальном сервере в другом сетевом сегменте, причем номера кредитных карт должны быть зашифрованы. Доступ к первому набору данных позволяет получить только контактную информацию клиента; доступ к набору данных с номерами кредитных карт позволит получить только зашифрованные их номера.

На рис. 4.5 проиллюстрирована архитектура приложения, позволяющего осуществлять безопасную работу с данными кредитных карт.

Эта архитектура очень проста, но ее, тем не менее, очень сложно скомпрометировать при том условии, что вы будете соблюдать следующие меры предосторожности.

- Сервер приложения и сервер обработки кредитных карт располагаются в двух разных зонах безопасности, причем между этими зонами допускается только трафик Web-сервисов от сервера приложений к зоне процессора кредитных карт.
- Номера кредитных карт шифруются с использованием индивидуальных клиентских ключей шифрования.
- Процессор кредитных карт не имеет доступа к ключу шифрования, за исключением краткого периода времени, когда он (в памяти) осуществляет обработку транзакции по конкретной кредитной карте.
- Сервер приложений никогда не имеет возможности прочесть информацию о номере кредитной карты с сервера обработки кредитных карт.
- Никто не имеет административного доступа к обоим серверам.

При использовании этой архитектуры злоумышленник не может извлечь никакой пользы из информации, полученной с отдельного сервера; чтобы получить доступ к информации кредитных карт, ему необходимо взломать оба сервера. Разуме-

ется, если ваше Web-приложение написано плохо, то никакое структурирование не защитит ваши данные от компрометации.

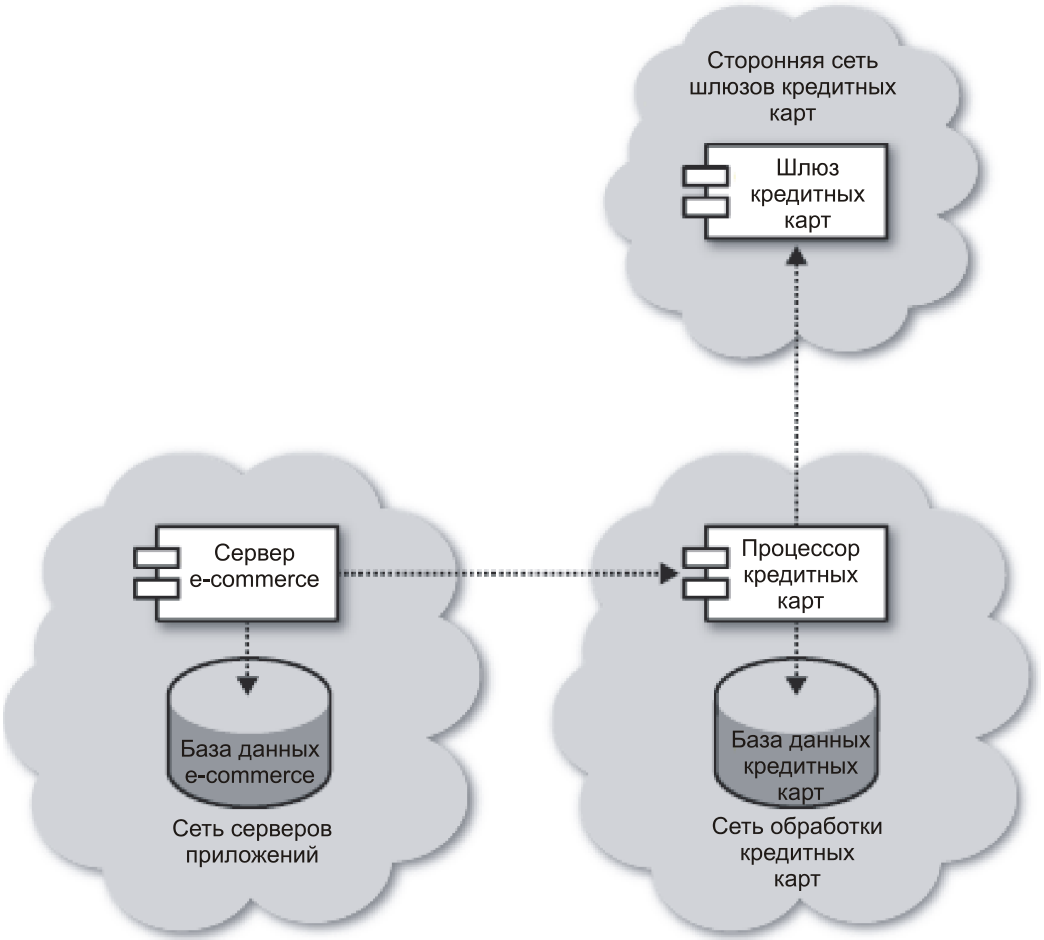


Рис. 4.5. Хранение данных кредитных карт за Web-сервисом, который шифрует данные кредитных карт

Следовательно, вам необходимо минимизировать риск того, что взломщик сможет использовать информацию с одного сервера для взлома другого. Поскольку эта проблема применима и к облачной инфраструктуре в целом, я детально рассмотрю ее в *главе 5*. На текущий момент я ограничусь формулировкой двух эмпирических правил.

- Убедитесь в том, что оба сервера имеют два различных вектора атаки. Иными словами, на них не должно работать одно и то же программное обеспечение. Если вы будете следовать этому руководящему указанию, вы сможете гарантировать, что какой бы эксплойт не скомпрометировал первый сервер, он не сможет добиться этой цели на втором.

- Убедитесь в том, что ни один из серверов не хранит учетные данные или другую информацию, с помощью которой можно скомпрометировать другой сервер. Иными словами, не используйте пароли для регистрации пользователей и не храните никаких конфиденциальных ключей SSH ни на одном из серверов.

Управление шифрованием информации о кредитных картах

Чтобы снять деньги с кредитной карты, необходимо предоставить номер этой кредитной карты, дату истечения срока ее годности, а также переменное количество других элементов данных, описывающих владельца этой кредитной карты. Кроме того, вам может потребоваться ввести секретный код.

Описываемая архитектура разделяет базовый захват данных от фактического списания денег с карточного счета. Когда клиент сначала вводит свою информацию, система сохраняет контактную информацию и некоторую базовую информацию профиля кредитной карты с помощью приложения электронной коммерции и отправляет номер кредитной карты на процессор кредитных карт для шифрования и дальнейшего сохранения.

Первая сложность заключается в создании пароля на сервере электронной коммерции и сохранения его вместе с записью клиента. Это не тот пароль, который сможет увидеть или использовать каждый, поэтому вам необходимо автоматически сгенерировать нечто сложное, следуя руководящим указаниям по созданию криптографически стойких паролей. Кроме того, вам также потребуется создать на сервере электронной коммерции запись для кредитной карты, в которой будет храниться все, за исключением номера кредитной карты. Пример такой модели данных для системы электронной коммерции показан на рис. 4.6.

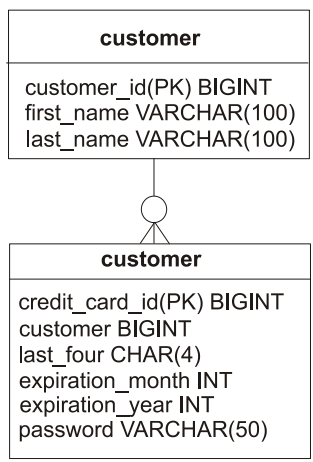


Рис. 4.6. Система электронной коммерции сохраняет все данные, за исключением номера кредитной карты и секретного кода

Сохранив эту информацию в базе данных системы электронной коммерции, система передает номер кредитной карты, ее пароль и уникальный идентификатор кредитной карты из системы электронной коммерции на процессор кредитных карт.

Процессор кредитных карт не сохраняет пароль. Вместо этого он использует пароль в качестве "привязки" (SALT)[1], чтобы зашифровать номер кредитной карты, сохраняет зашифрованный номер кредитной карты и ассоциирует его с идентификатором кредитной карты. Модель данных, используемая процессором кредитных карт, показана на рис. 4.7.

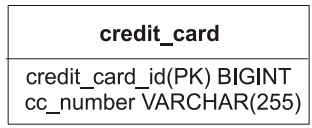


Рис. 4.7. Процессор кредитных карт хранит зашифрованный номер кредитной карты и ассоциирует его с идентификатором кредитной карты с сервера электронной коммерции

Ни одна из систем не хранит секретного кода клиента, потому что компании, поддерживающие платежные системы на основе кредитных карт, это явно запрещают.

Обработка транзакции с использованием кредитной карты

Когда дело доходит до списания денег с кредитной карты, сервис системы электронной коммерции отправляет запрос на процессор кредитных карт, чтобы снять с карточного счета конкретную денежную сумму. Система электронной коммерции ссылается на кредитную карту на процессоре кредитных карт, используя уникальный идентификатор, который был сгенерирован при первом вводе кредитной карты. Он передает пароль кредитной карты, секретный код и сумму, подлежащую списанию. Процессор кредитных карт затем расшифровывает номер кредитной карты, используя указанный пароль. Незашифрованный номер кредитной карты, секретный код и сумма, подлежащая списанию, передаются в банк для завершения транзакции.

---

[1] В криптографии SALT состоит из случайных битов, которые используются в качестве одного из элементов ввода для функции генерации ключа. Вторым элементом ввода обычно является пароль (password) или парольная фраза (passphrase). Вывод функции генерации ключа сохраняется как зашифрованная версия пароля. Подробнее см. **http://en.wikipedia.org/wiki/Salt_(cryptography)**, **http://faqs.org.ru/progr/common/crypt_faq2.htm**, **http://www.cybersecurity.ru/manuals/crypto/ssl/2088.html**. — *Прим. перев.*

Что происходит при компрометации системы электронной коммерции

Если приложение, реализующее систему электронной коммерции, будет скомпрометировано, атакующий получит доступ только к неконфиденциальной контактной информации клиента. Механизма, который позволил бы ему скачать эту базу данных и получить с ее помощью доступ к информации о кредитных картах или как-либо иначе использовать ее для хищения персональной информации (identity theft), не существует. Чтобы добиться этой цели, злоумышленнику потребуется отдельно взломать процессор кредитных карт.

С учетом всего сказанного следует отметить, что если ваше приложение, реализующее систему электронной коммерции, небезопасно, атакующий все же может вычислить идентификационную информацию существующих пользователей и размещать заказы от их имени, но с доставкой на свой адрес. Иными словами, вам все равно в любом случае следует серьезно задуматься о планировании архитектуры каждого из компонентов вашей системы.

Примечание

Очевидно, что вам также захочется предотвратить доступ злоумышленников и к контактным данным ваших клиентов. В контексте этого раздела ссылка на контактную информацию о клиентах как "не являющуюся конфиденциальной" тоже является весьма относительной. Ваша цель должна заключаться в том, чтобы злоумышленник не мог получить из вашей системы ни одного бита данных.

Что происходит при компрометации процессора кредитных карт

Компрометация процессора кредитных карт еще бесполезнее, чем компрометация приложения, реализующего систему электронной коммерции. Если атакующий получит доступ к базе данных кредитных карт, все, что он получит, будет представлять собой набор данных, состоящий из случайным образом сгенерированных уникальных идентификаторов и номеров кредитных карт, зашифрованных сильным криптографическим алгоритмом (потому что каждая из записей зашифрована с использованием уникального ключа шифрования). В результате атакующий сможет скопировать базу данных и в автономном (offline) режиме предпринять на нее атаку по методу грубой силы (brute-force), чтобы расшифровать номера кредитных карт. Во-первых, расшифровка каждого номера потребует длительного времени, а во-вторых, даже в случае успеха ее нельзя будет использовать для хищения конфиденциальной информации, естественно, при условии, что хакер, раздобывший номер кредитной карты, не имеет никакой индивидуальной информации клиента, которую можно было бы использовать для размещения заказа.

Еще один вектор атаки заключается в том, чтобы догадаться, как встроить приложение типа "троянский конь" (Trojan application) на скомпрометированный сервер с тем, чтобы прослушивать расшифровку паролей. Однако если у вас применя-

ется система обнаружения вторжений (*см. главу 5*), то и этот вектор атаки сможет быть обезврежен.

Что происходит, если облачные сервисы Amazon не могут удовлетворить ваши потребности

Архитектура, которую я описал в предыдущем разделе, довольно близко соответствует обычным, необлачным приложениям. Однако вы, при развертывании в облаке Amazon, можете столкнуться с проблемами, включая критические вопросы обработки конфиденциальных данных. Вот две наиболее важных:

- Некоторые законы и спецификации устанавливают ограничения политического и юридического характера на местоположение хранилищ данных. В частности, компании, ведущие бизнес в Евросоюзе, не могут хранить конфиденциальные данные граждан ЕС на серверах, расположенных в США (или любой другой стране, в которой не применяются стандарты конфиденциальности, принятые в ЕС).
- Некоторые законы и спецификации разрабатывались без учета виртуализации. Иными словами, в них рассматриваются физические серверы для тех случаев, когда и виртуальные серверы могут справляться с задачей не хуже, просто потому что под словом "сервер" на момент разработки законопроекта или стандарта подразумевался именно физический сервер.

Первая проблема имеет достаточно очевидное решение: если вы ведете бизнес в одной из стран Евросоюза и управляете конфиденциальными данными граждан Евросоюза, эти данные должны обрабатываться на серверах, физически расположенных в ЕС, храниться на устройствах, физически расположенных в ЕС, и никогда не должны пересекать границ инфраструктуры, выходящей за пределы ЕС.

Amazon обеспечивает присутствие как в США, так и в ЕС. В результате вы можете решить первую проблему, тщательно планируя инфраструктуру вашего облачного решения на базе сервисов Amazon. Однако для этого требуется, чтобы вы четко сформулировали свои требования к управлению данными и ассоциировали их с местоположением запускаемых экземпляров.

Вторая проблема особенно сложна, особенно для тех, кто выбирает решение на базе сервисов Amazon, поскольку все такие решения полностью полагаются на виртуализацию. В данном случае, однако, причина возникновения проблемы довольно глупа. Ведь то, что вы делаете, может полностью соблюдать дух закона, но противоречить его букве, только потому, что концепция виртуализации не была распространена на момент принятия закона или спецификации. Но и для этой проблемы есть решение, причем оно даже аналогично решению первой.

В обоих случаях вам требуется сделать все, чтобы воспользоваться, насколько это возможно, преимуществами облачной инфраструктуры, не нарушая законодательства о хранении персональных данных и избегая чрезмерного усложнения всей

системы до такого уровня, когда она просто перестанет окупать себя. Другие провайдеры облачных сервисов, такие как Rackspace и GoGrid, предлагают свои услуги, более простые, нежели гибридные решения на базе Amazon или какой-либо другой компании.

Что считать конфиденциальной информацией?

Когда заходит речь о законах и стандартах, то, что вы считаете конфиденциальной информацией может ею и не являться с точки зрения законодательства. Например, в некоторых случаях действующее законодательство ЕС о конфиденциальной информации считает IP-адреса информацией, идентифицирующей личность. Я не являюсь экспертом в области законодательства, и поэтому даже не буду делать попыток дать определение конфиденциальной информации. Но, как только вы и ваши эксперты-юристы определите, что именно должно считаться конфиденциальной информацией, вы должны будете следовать общим правилам, определенным в данном разделе: конфиденциальные данные должны храниться на защищенном сервере, а данные, которые конфиденциальной информацией не считаются, могут размещаться и в облачной среде.

Чтобы решить эту задачу, вы должны маршрутизировать и хранить всю конфиденциальную информацию за пределами облака, но при этом стремиться к тому, чтобы как можно большее количество операций, реализующих логику приложения, выполнялись в облачной среде. Реализовать эту цель можно, следуя общему подходу, который я сформулировал для обработки кредитных карт. В общем виде концепции сервера обработки конфиденциальной информации и сервера, на котором работают Web-приложения, выглядят так:

- сервер, предназначенный для обработки конфиденциальной информации, находится вне облачной среды и обеспечивает минимальную поддержку структур, предназначенных для обработки конфиденциальных данных;
- сервер Web-приложения находится в облачной среде и реализует основную логику работы вашего приложения.

Вследствие того, что цель создания сервера обработки конфиденциальной информации заключается в физическом выделении секретных данных, вам при использовании этого подхода уже не обязательно шифровать всю информацию на сервере обработки секретных данных. На рис. 4.8 показано, как можно трансформировать архитектуру приложения за счет вывода сети обработки секретных данных за пределы облачной среды.

Как и в случае с системой электронной коммерции, полностью развернутой в облачной среде, информация о кредитных картах хранится на сервере, расположенном в собственном сетевом сегменте. Единственным отличием является то, что теперь процессор кредитных карт находится за пределами облака.

Еще один "фрагмент головоломки" заключается в обработке информации, персонально идентифицирующей клиентов. Эти данные теперь хранятся на собственном сервере за пределами облака, но отдельно от информации кредитных карт. При сохранении информации пользовательских профилей эти функции исполняются на сервере обработки секретной информации, а не на сервере основного Web-приложения. Основное Web-приложение ни при каких обстоятельствах не должно

получать доступ к информации персональной идентификации, за исключением случаев, когда эти данные аггрегируются перед представлением Web-приложению.

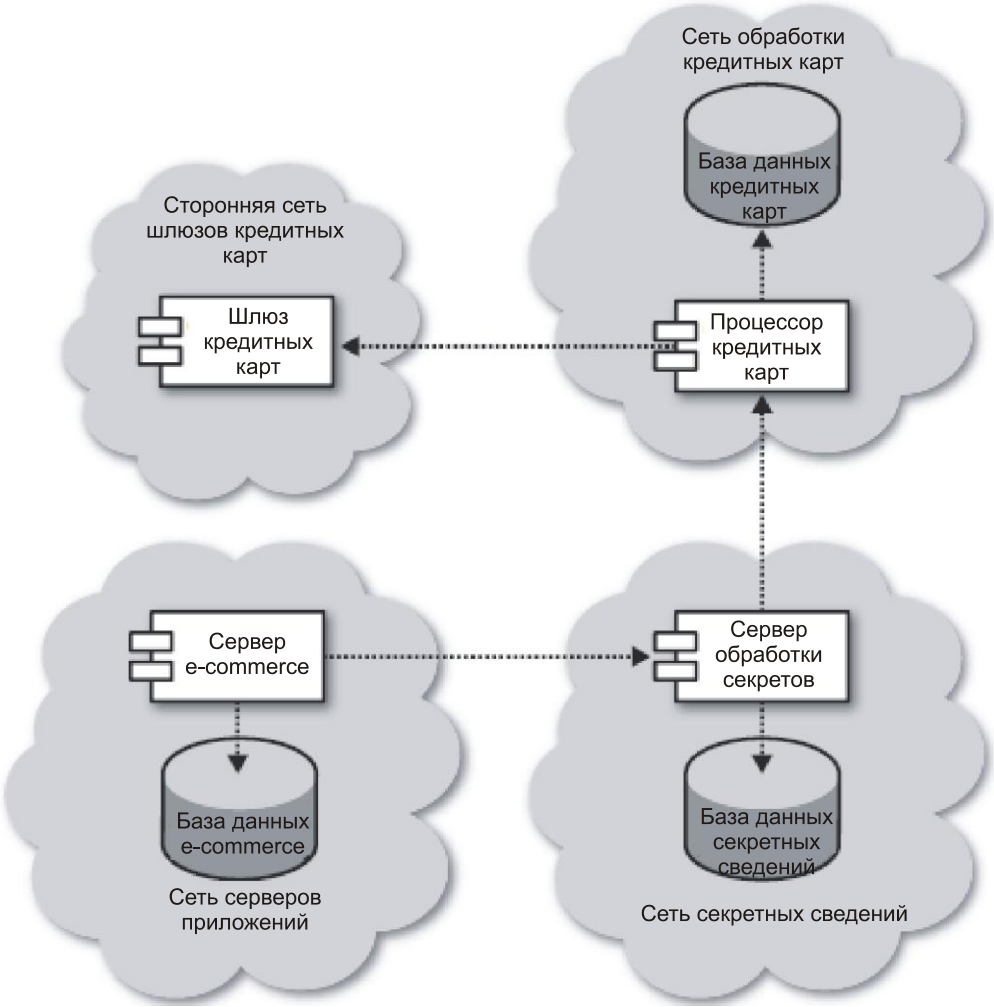


Рис. 4.8. Вывод сети обработки секретных данных за пределы облака приводит к возникновению трех компонентов приложения

Эффективность этой архитектуры сильно зависит от количества операций по обработке данных, не имеющих отношения к конфиденциальной информации. Если все ваши транзакции связаны с чтением и записью конфиденциальных данных, то вполне возможно, что вы просто усложните архитектуру и ничего не выиграете взамен. С другой стороны, если управление конфиденциальной информацией представляет собой лишь малую часть вашего приложения, вы можете использовать преимущества облачной обработки данных для всех остальных компонентов вашего приложения, не имеющих отношения к секретности, наряду с этим обеспечивая

полное соответствие действующему законодательству за счет физического выделения конфиденциальных данных.

Управление базами данных

Наиболее сложным аспектом управления облачной инфраструктурой является обработка постоянных данных (persistent data). *Постоянными данными* считается вся информация, которая в обязательном порядке должна сохраниться при разрушении вашей облачной среды. Поскольку вы с легкостью можете восстановить операционную систему, приложения и простые конфигурационные файлы, вся эта информация постоянными данными не считается. Таким образом, к постоянным данным относятся те данные, которые не могут быть восстановлены так просто. Если вы следуете моим рекомендациям, то эти данные должны "жить" в движке вашей базы данных.

Проблема с поддержанием целостности базы данных характерна не только для облачной инфраструктуры. Переход на облачные вычисления просто обостряет эту старую проблему, потому что ваш сервер базы данных в облачной среде будет гораздо менее надежен, чем сервер базы данных в физической инфраструктуре. Виртуальный сервер, на котором работает ваша база данных, может отказать полностью и без предупреждений. Имейте это в виду.

Когда отказывает сервер базы данных, неважно, физический или виртуальный, существует явная вероятность того, что база данных окажется поврежденной. Вероятность этого события зависит от того, какой движок базы данных вы используете, но случиться это может с любым из существующих движков.

Если абстрагироваться от вопросов, связанных с повреждением данных, все остальные аспекты управления сервером базы данных в облачной среде очень просты. На практике восстановление сервера в виртуальной среде даже проще, чем в физической инфраструктуре: достаточно просто запустить новый экземпляр вашего образа сервера базы данных, примонтировать старое устройство блочного хранения данных, и работоспособность вашего сервера будет восстановлена.

Примечание

Для хранения баз данных используйте блочные устройства хранения. Устройства блочного хранения являются лучшим выбором с точки зрения производительности (это лучше, чем локальное хранение) и предоставляют больше гибкости при реализации стратегий резервного копирования баз данных.

Объединение в кластеры или репликация?

Наиболее эффективным механизмом, позволяющим избежать повреждения баз данных, является применение возможностей движка базы данных, поддерживаю-

щего возможности *истинной кластеризации* (true clustering). В кластерной среде управления базами данных множество серверов баз данных действуют совместно как единый сервер базы данных. Механизмы реализации этого процесса зависят от движка базы данных и для разных движков разнятся, но результат заключается в том, что транзакция, подтвержденная в кластере, "переживет" отказ любого из одиночных узлов, и целостность данных будет сохранена в полном объеме. На практике клиенты базы данных никогда даже не узнают о том, что какой-то из узлов в какой-то момент времени отказал, и смогут продолжать работу, как ни в чем не бывало.

К сожалению, кластеризация баз данных представляет собой очень сложное и дорогое решение. При принятии решений за или против кластерной обработки баз данных руководствуйтесь следующими принципами.

- Если только в вашей команде нет опытных специалистов-экспертов в области баз данных, вам не следует даже рассматривать развертывание кластерной базы данных.
- Поставщики кластеризованных баз данных часто требуют от клиентов оплаты более дорогих лицензий, чем при покупке обычных систем управления базами данных (DBMS), не поддерживающих кластерную обработку. Даже если вы используете возможности по кластеризации MySQL, вам уже потребуется платить за то, чтобы пять экземпляров эффективно работали в качестве кластера.
- Кластеризация связана с серьезными проблемами производительности. Если вы попытаетесь создать кластер, пересекающий границы физических инфраструктур (в облачной среде — зон доступности), то за это вам придется расплачиваться существенно возросшими задержками в работе сети.

Внимание!

Хотя проблем с кластеризацией базы данных в облачной инфраструктуре немного, одна из них является существенной: проблемы с вводом/выводом, присущие виртуализованным системам. В частности, операции записи в кластеризованной системе интенсивно потребляют сетевой трафик. В результате приложения, интенсивно осуществляющие запись, в виртуальной кластерной среде будут демонстрировать существенно более низкую производительность, нежели в стандартном центре обработки данных.

Альтернативой кластерной обработке является *репликация*. Инфраструктура системы управления базами данных, основанная на репликации, обычно подразумевает наличие главного сервера баз данных (database master). Клиентские приложения выполняют транзакции записи на сервере главной базы данных. Затем успешные транзакции реплицируются на подчиненные серверы баз данных (database slaves).

По сравнению с кластеризацией репликация обладает двумя основными преимуществами.

- Обычно репликация намного проще в реализации.
- Репликация не требует очень большого количества серверов и дорогих лицензий.

Внимание!

Репликация MySQL не является приемлемым решением для тех, кто ни при каких обстоятельствах не может позволить себе потерять ни единого байта данных в результате сбоя сервера. Но компании, занимающиеся таким бизнесом, как правило, могут позволить себе кластеризацию.

К сожалению, надежность репликации даже не приближается к той надежности, которую обеспечивает кластеризация. Главная база данных теоретически может отказать после того, как транзакция была записана локально, но еще не распространена на подчиненные серверы. В этом случае вы получите подчиненную базу данных, в которой будут отсутствовать данные об этой транзакции. На практике, когда сервер главной базы данных работает в условиях пиковой нагрузки, обновление подчиненных баз данных может существенно запаздывать. Если при этом случится сбой главного сервера баз данных, то вдобавок к этому он может начать распространение поврежденных данных.

Кроме проблем с надежностью, среды с репликацией не обеспечивают такого легкого восстановления после сбоев, как кластерные решения. Когда отказывает главный сервер базы данных, клиенты, ведущие туда запись транзакций, не смогут нормально функционировать до тех пор, пока сервер главной базы данных не будет восстановлен. С другой стороны, если отказывает один из узлов в составе кластера, то клиенты этого сбоя могут даже не заметить, поскольку кластер продолжит обработку транзакций как ни в чем не бывало.

Применение кластеризации баз данных в облачной среде

В целом, хорошая новость заключается в том, что облачная среда создает очень мало специфических проблем с кластеризацией баз данных. Плохая же новость состоит в том, что каждый конкретный движок базы данных использует свой механизм кластерной обработки (или даже несколько различных подходов к кластеризации), и по этой причине детальное рассмотрение кластеризации баз данных в облачной инфраструктуре выходит далеко за рамки этой книги. Однако я могу предоставить несколько руководящих указаний.

- Некоторые кластерные архитектуры создавались с целью обеспечения высокой производительности, а высокая доступность во главу угла не ставилась. При использовании этих архитектур все же могут существовать отдельные точки сбоя. Фактически сложность кластеризации может все-таки приводить к существованию дополнительных точек отказа.
- Кластеры, разрабатывавшиеся специально для обеспечения высокой надежности, обычно работают медленнее при обработке отдельных транзакций записи,

но они могут работать под значительно более высокой нагрузкой, чем одиночные серверы баз данных. В частности, они могут масштабироваться в соответствии с вашими потребностями на объем операций чтения.

- Некоторые решения, например, такие как MySQL, для эффективной работы требуют большего количества серверов. Даже если затраты на лицензирование такой конфигурации пренебрежимо малы, но к ним добавятся увеличенные расходы на облачную инфраструктуру.
- Динамическая природа назначения IP-адресов в облачной среде может добавить проблем с конфигурированием кластеров и их правил восстановления после сбоев.

Использование репликации баз данных в облачной среде

Для большинства приложений баз данных, не предназначенных для решения критически важных задач, репликация является неплохим решением, которое позволит вам сэкономить финансовые средства и, вполне возможно, даже предоставит возможности по оптимизации производительности. На практике система репликации MySQL в облачной среде может предоставить вам безупречные системы резервного копирования и аварийного восстановления, а также обеспечить высокую доступность, сравнимую с обеспечиваемой кластерными решениями. Поскольку репликация в облачной инфраструктуре может иметь такое существенное значение по сравнению с репликацией в традиционном центре обработки данных, мы рассмотрим ее более подробно, чем кластеризацию.

Пример простой среды, использующей репликацию, показан на рис. 4.9.

В этой структуре имеется единственный сервер главной базы данных, работающий на запись (master), который распространяет копии главной базы данных на один или несколько подчиненных серверов (slaves). В общем случае[1] процесс, осуществляющий репликацию данных с главной базы данных в подчиненные, не является атомарным в отношении оригинальной транзакции. Иными словами, сам факт успешного подтверждения транзакции на главном сервере еще не значит, что она будет успешно реплицирована в подчиненные базы данных. Системы, которые могут это гарантировать, обычно обеспечивают атомарность. В результате подчиненные базы данных могут быть рассинхронизированы с главной, потому что, хотя подчиненная база данных и может быть рассинхронизирована с главной, она все же должна сохранять внутренне непротиворечивое состояние (иными словами, оставаться неповрежденной).

[1] Эта ситуация может и не быть применимой ко всем движкам баз данных в среде, использующей архитектуру, основанную на репликации. Однако за исключением случаев, когда вы точно знаете, что ваш механизм репликации атомарен в отношении оригинальных транзакций, вы должны подразумевать, что в определенные моменты времени ваша подчиненная база данных может быть рассинхронизирована с главной.

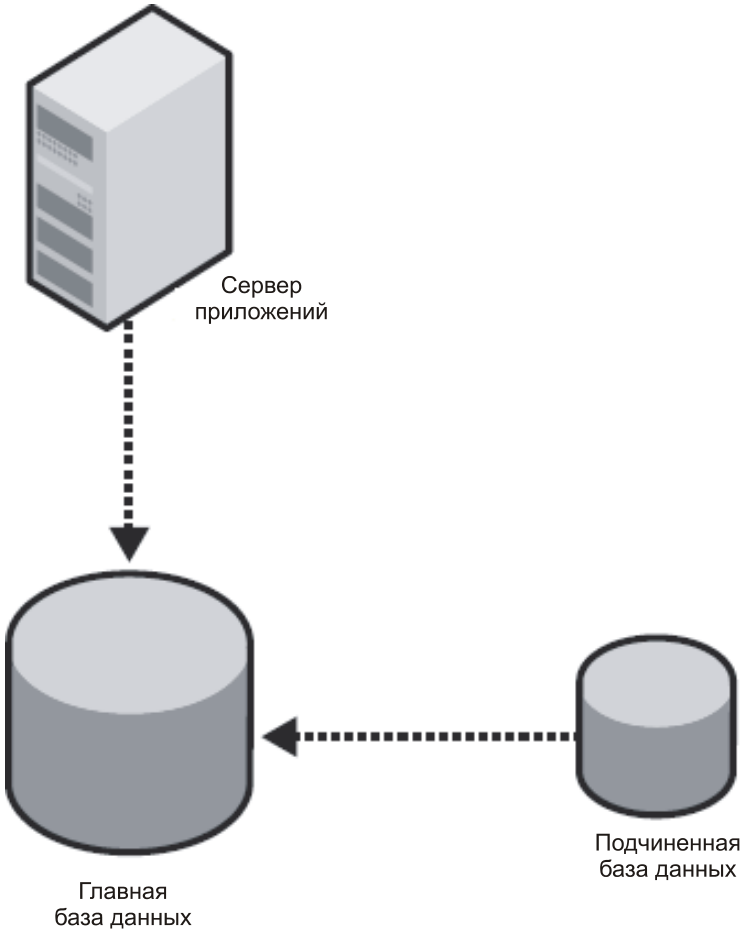


Рис. 4.9. Схема простой среды на базе репликации (направление стрелки указывает на зависимость)

В случае простой конфигурации ваши Web-приложения указывают на главную базу данных. По этой причине сбой вашей подчиненной базы данных может не повлиять на работу вашего Web-приложения. Чтобы выполнить восстановление, запустите новый подчиненный сервер базы данных и направьте его на главный.

Восстановление после сбоя сервера главной базы данных окажется более сложным. Если вы пользуетесь облачными сервисами Amazon, вы столкнетесь с дополнительными препятствиями, которых вы не встретите в стандартной среде с репликацией.

В идеальном случае вы сможете восстановить ваш главный сервер баз данных, запустив новый виртуальный сервер с помощью образа вашей машины базы данных, и затем примонтировав к нему том, который изначально был примонтирован к отказавшему серверу. Но отказ вашего основного сервера баз данных теоретически может привести к повреждению файлов на этом томе. На этот момент вам придется переключиться на подчиненный сервер баз данных.

Дальнейшее восстановление базы данных с использованием подчиненного сервера может пройти одним из двух способов.

- Выдвижение подчиненного сервера базы данных на роль главного сервера баз данных (для этой цели вам потребуется запустить еще один сервер, который заменит подчиненный сервер, выдвинутый на роль главного).
- Построение нового главного сервера баз данных и экспортирование базы данных в ее текущем состоянии с подчиненного сервера на новый главный сервер.

Выдвижение подчиненного сервера на роль главного является самым быстрым механизмом восстановления. Вероятнее всего, вы примете именно этот подход, если только вы не хотите иметь отдельные образы для главной и подчиненной баз данных. В этом последнем случае вам потребуется использовать более сложный подход к восстановлению.

Внимание!

Чтобы разработать надежную, "пуленепробиваемую" архитектуру репликации, вам необходимо заглянуть чуть дальше, чем восстановление с подчиненной базы данных. Вполне возможна и такая ситуация, когда ваш подчиненный процесс остановился задолго до отказа главного сервера баз данных, или, что еще хуже, подчиненный сервер отказал вместе с главным. Таким образом, вам необходимо продумать возможности восстановления с моментального снимка тома и, в самом худшем случае, с дампа базы данных, хранящегося в вашей облачной системе хранения. Кроме того, вы должны предусмотреть ведение мониторинга статуса подчиненного сервера, и это должно стать ключевой частью вашей системы мониторинга облачной инфраструктуры.

Как и в случае с любыми другими архитектурными компонентами вашего Web-приложения, помещение базы данных в архитектуру, основанную на репликации, дает возможность быстрого восстановления узла после сбоя и, в результате, существенно повышает общий рейтинг доступности системы.

Подробную информацию о репликации MySQL можно найти в книге [3].

Использование репликации для повышения производительности

Еще одним доводом в пользу применения репликации является повышение производительности. Без сегментации данных большинство движков базы данных позволяет вести запись только в главную базу данных, но читать данные вы можете как из главной базы данных, так и из любой подчиненной. Приложение, интенсивно читающее данные, может существенно выиграть по производительности за счет распределения операций чтения по подчиненным серверам базы данных. Архитектура приложения, использующего репликацию в целях повышения производительности, показана на рис. 4.10.

Преимущества, выигрываемые за счет применения репликации, огромны, но существуют и определенные риски. Основным из них является возможность случайного исполнения операции записи в одну из подчиненных баз данных. Если

это случится, вся схема репликации "развалится", потому что главная база и ее подчиненные окажутся рассогласованными, и возникнет неустранимое противоречие.

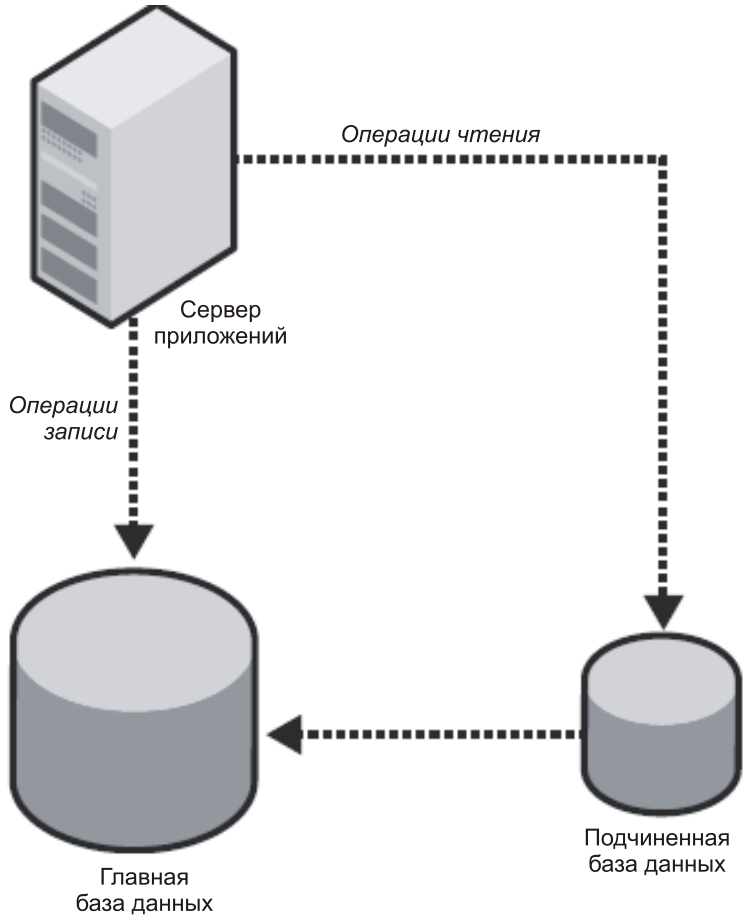


Рис. 4.10. За счет распределения операций чтения по подчиненным серверам ваше приложение может масштабироваться без кластеризации

Чтобы решить эту проблему, применяются два подхода.

- Четкое отделение логики чтения от логики записи на уровне кода и централизация установления соединений с базой данных.
- Защита подчиненных узлов от записи, т. е. присвоение им прав доступа "только для чтения" (read-only).

Второй метод более надежен, но зато он осложняет процесс выдвижения подчиненного сервера на роль главного, поскольку вам придется переконфигурировать сервер, сняв режим "только для чтения" перед его выдвижением на новую роль.

Управление главными ключами

В случаях, когда Web-приложение работает за балансировщиком нагрузки и отдельные узлы Web-приложения не делятся друг с другом информацией о статусе, межбазовая генерация главных ключей (primary key) превращается в проблему. Функциональные возможности движка базы данных по автоматическому инкрементированию зависят от типа используемой вами СУБД, и возможности по их настройке не отличаются гибкостью. Довольно часто можно встретить ситуации, когда гарантировать уникальность главного ключа возможно только в пределах одного сервера.

В своей книге "Java Database Best Practices" (O'Reilly; **http://oreilly.com/catalog/9780596005221/index.html**) я детально описал механизм генерации ключей в памяти на сервере приложений, который позволяет обеспечить уникальность первичных ключей при распределенной работе на множестве серверных узлов — даже для множества приложений, написанных на разных языках программирования. Здесь я опишу этот механизм более подробно, причем добавлю еще полезный прием: генерацию случайных идентификаторов, уникальность которых гарантируется при распределенной работе на множестве серверных узлов.

Генерация глобально уникальных первичных ключей

Во-первых, вы можете пользоваться стандартом UUID[1] для генерации идентификаторов, используя его как ваш основной механизм генерации первичных ключей. Это позволит вам практически свести к нулю шанс генерации конфликтующих идентификаторов, причем большинство языков программирования содержат встроенные функции для генерации таких идентификаторов. Правда, я их не использую, в основном, по трем причинам:

- генерируемые идентификаторы представляют собой 128-битные значения, которые требуют больше пространства для хранения и больше времени на поиск, чем предпочитаемые мною 64-битные первичные ключи;
- четкое представление 128-битных значений в Java и некоторых других языках программирования довольно сложно. Фактически наилучшим методом пред-

[1] UUID (Universally Unique Identifier) — унивесально уникальный идентификатор. Представляет собой стандарт идентификации, используемый в создании ПО, стандартизированный Open Software Foundation (OSF) как часть распределенной компьютерной среды (Distrubuted Computing Environment, DCE). Основное назначение UUID заключается в том, чтобы дать распределенным системам возможность уникально идентифицировать информацию без центра координации. Таким образом, любой может создать UUID и использовать его для идентификации чего-либо с приемлемым уровнем уверенности, что данный идентификатор непреднамеренно никогда не будет использован для чего-то еще. Информация, помеченная с помощью UUID, может, поэтому, позже быть помещена в общую базу данных без необходимости разрешения конфликта имен. Подробнее см. **http://ru.wikipedia.org/wiki/UUID**, **http://rus-linux.net/nlib.php?name=/MyLDP/consol/HuMan/blkid-ru.html**. — *Прим. перев.*

ставления таких значений заключается в использовании двух отдельных значений, представляющих старшие 64 бит и младшие 64 бит соответственно;

- возможность конфликтов, хотя и пренебрежимо малая, все же существует.

Чтобы генерировать идентификаторы на уровне сервера приложений, уникальность которых гарантируется в целевой базе данных, я обычно полагаюсь на функциональные возможности генерации ключей, встроенных в эту базу. Я решаю эту проблему за счет создания таблицы `sequencer`, которая выдает ключ с безопасным пространством ключей. Сервер приложений после этого свободно может создавать ключи в этом пространстве до тех пор, пока оно не будет полностью исчерпано.

Примечание

Я предпочитаю использовать в базах данных первичные ключи, представляющие собой 64-битные целые числа. Длина в 64 бита — это достаточно широкий диапазон для ключей, одновременно позволяющий добиться быстрого поиска. Метод, о котором я говорю, работает и при создании алфавитно-цифровых ключей.

Таблица `sequencer` выглядит так, как показано в листинге 4.5.

Листинг 4.5. Создание таблицы `sequencer`

```
CREATE TABLE sequencer (
        name VARCHAR(20) NOT NULL,
        next_key BIGINT UNSIGNED NOT NULL,
        last_update BIGINT UNSIGNED NOT NULL,
        spacing INT UNSIGNED NOT NULL;
        PRIMARY KEY ( name, last_update ),
        UNIQUE INDEX ( name )
);
```

Первое, на что здесь следует обратить внимание — это то, что структура данной таблицы не имеет никаких особенностей, специфичных для конкретной базы данных, и ваши ключи не "привязаны" ни к какой конкретной таблице. При необходимости пространство первичных ключей может совместно использоваться многими таблицами. Аналогично, вы можете создавать уникальные идентификаторы, не имеющие ничего общего ни с одной из таблиц в вашей базе данных.

Чтобы сгенерировать уникальный идентификатор клиента (`person_id`) для вашей таблицы `person`, действуйте следующим образом:

1. Создайте в памяти значение `next_key` и инициализируйте его значением 0.
2. Захватите из записи `sequencer` значения `spacing` и `last_update` с `name = 'person.person_id'`.
3. Добавьте 1 к извлеченному значению `next_key` и обновите таблицу `sequencer` значениями `name` и `last_update` в предложении `WHERE`.
4. Если ни одна из строк не обновлена (потому что вас опередил другой сервер), повторите шаги 2 и 3.

5. Установите следующий идентификатор клиента (`person_id`) на значение `next_key`.
6. Увеличьте значение `next_key` на 1.
7. Когда вам в следующий раз потребуется создать уникальный идентификатор клиента, выполняйте шаги 5 и 6 до тех пор, пока `next_key < next_key + spacing`. В противном случае снова установите значение `next_key` на 0 и повторите всю процедуру с начала и до конца.

В пределах сервера приложений весь этот процесс должен быть заблокирован от многопоточного доступа.

Поддержка глобально уникальных случайных ключей

Прием генерации уникальных ключей, описанный в предыдущем разделе, генерирует идентификаторы, образующие более или менее упорядоченные последовательности. В некоторых случаях важно устранить предсказуемость при генерации идентификаторов. Для этой цели вам необходимо привнести в этот алгоритм некоторый элемент случайности.

Внимание!

Этот метод не является истинно случайным. Напротив, он псевдослучаен (pseudorandom). Здесь для чисел, генерируемых случайным образом, нет источника энтропии, и диапазон случайных чисел достаточно мал для того, чтобы человек, целенаправленно пытающийся осуществить взлом, мог добиться успеха.

Чтобы получить случайный идентификатор, вам требуется умножить значение `next_key` на одну из степеней 10 и добавить случайным образом полученное число через генератор случайных чисел, встроенный в выбранный вами язык программирования. Чем выше вероятность случайного числа, тем меньше будет ваше общее пространство ключей. С другой стороны, чем меньше вероятность случайного числа, тем проще будет вычислить ваш алгоритм генерации ключей.

В листинге 4.6 приведен пример кода на Python, иллюстрирующий генерацию псевдослучайного уникального личного идентификатора (person ID).

Листинг 4.6. Код на Python, иллюстрирующий генерацию псеводслучайного личного идентификатора

```
import thread
import random

nextKey = .1;
spacing = 100;
lock = thread.allocate_lock();

def next():
    try:
        lock.acquire(); # в каждый момент времени доступ может иметь только один поток
```

```
        if nextKey == .1 or nextKey > spacing:
            loadKey();
        nextId = (nextKey * 100000);
        nextKey = nextKey + 1;
    finally:
        lock.release();
    rnd = random.randint(0,99999);
    nextId = nextId + rnd;
    return nextId;
```

Вы можете минимизировать непроизводительные затраты пространства ключей, отслеживая выделение случайных чисел и увеличивая значение `nextKey` только после того, как случайное пространство будет в достаточной степени израсходовано. Однако чем дальше вы будете двигаться по этому пути, тем выше вероятность того, что вы столкнетесь со следующими проблемами:

- генерация уникальных ключей будет требовать больше времени;
- ваше приложение будет потреблять все больше памяти;
- случайность генерации идентификаторов будет убывать.

Резервное копирование баз данных

На протяжении всего предшествующего изложения я все время упоминал о сложностях управления резервным копированием баз данных и взаимосвязи этого процесса с аварийным восстановлением. В полном объеме я буду обсуждать аварийное восстановление в облачной среде в *главе 6*, пока же необходимо рассмотреть специфическую проблему выполнения безопасного резервного копирования баз данных в облачной инфраструктуре.

Выработать хорошую стратегию резервного копирования баз данных довольно сложно в любом случае, вне зависимости от того, работаете вы в облачной среде или в традиционной физической инфраструктуре. Однако в облаке иметь работоспособную и протестированную стратегию резервного копирования еще важнее.

Типы резервных копий баз данных

Большинство движков баз данных предоставляют множество механизмов выполнения резервного копирования баз данных. Объяснение существования различных стратегий резервного копирования заключается в том, что необходимо обеспечить разумный компромисс между выполнением резервного копирования в производственной среде и целостностью данных в резервной копии. Как правило, ваш движок базы данных предлагает, как минимум, три варианта резервного копирования (перечисляются в порядке убывания их надежности):

- экспорт/дамп базы данных;
- резервное копирование файловой системы;
- резервное копирование журнала транзакций.

Примечание

Движок вашей базы данных почти наверняка предложит и другие опции. Настоятельно рекомендуется изучить их и настроить ваш процесс резервного копирования таким образом, чтобы воспользоваться всеми преимуществами дополнительных возможностей.

Наиболее надежная резервная копия получается, когда вы выполняете *экспорт* (сбрасываете дамп) *всей базы данных*. Когда вы осуществляете экспорт базы данных, вы сбрасываете дамп всей схемы базы данных и всех хранящихся в ней данных в один или несколько экспортированных файлов. Затем вы можете создать резервные копии экспортированных файлов. В процессе восстановления вы можете применить экспортированные файлы для восстановления изначальной инсталляции (pristine install) вашего движка базы данных.

Чтобы экспортировать базу данных, например, в SQL Server, выполните следующую команду:

```
BACKUP DATABASE website to disk = 'D:\db\website.dump'
```

В результате вы получите экспортированный файл, который сможете переносить из одной среды SQL Server в другую.

"Оборотной стороной медали" при использовании метода экспорта базы данных является то, что на время его выполнения сервер должен быть заблокирован от записи, чтобы полученный дамп гарантированно находился во внутренне непротиворечивом состоянии. К несчастью, экспорт большой базы данных для завершения требует длительного времени. На все это время сервер выпадает из производственного процесса, поэтому обычно полный экспорт базы данных в производственной среде не практикуется.

Большинство баз данных предлагают опцию экспорта частей баз данных на индивидуальной основе. Например, в MySQL можно сбрасывать дамп таблицы `access_log` по ночам:

```
$ mysqldump website access_log > /backups/db/website.dump
```

Однако если таблица имеет зависимости от других таблиц системы, вы в результате можете получить набор внутренне противоречивых данных, если будете экспортировать таблицы по отдельности. Частичные операции экспорта обычно полезны для резервного копирования хранилищ данных (data warehouse).

Резервное копирование файловой системы подразумевает резервирование всех файлов, на которых основывается база данных. Некоторые движки баз данных подразумевают, что вся база данных хранится в одном большом файле, в то время как в других системах таблицы и их схемы хранятся во множестве файлов. В любом случае процесс резервного копирования заключается в копировании файлов базы данных на резервные носители.

Хотя резервное копирование на уровне файловой системы требует, чтобы база данных была заблокирована от обновлений, время блокировки обычно бывает непродолжительным. На практике возможности по созданию моментальных снимков томов блочного хранения обычно уменьшают время блокировки до секунды, при этом размер базы данных никакой роли не играет.

Следующее предложение SQL "заморозит" MySQL и даст вам возможность снять моментальный снимок файловой системы, на которой хранится база данных:

```
FLUSH TABLES WITH READ LOCK
```

Когда база данных будет заблокирована, снимите моментальный снимок тома, а затем снимите блокировку.

Наименьшее количество помех вызывает такой вид резервного копирования, как *резервирование журнала транзакций*. По мере того, как база данных подтверждает транзакции, эти транзакции записываются в файл журнала транзакций. Поскольку в журнале транзакций фиксируются только подтвержденные транзакции, эти журналы можно копировать, не блокируя базу данных и не останавливая сервис. Кроме того, файлы журнала меньше по размеру, чем дампы или файлы базы данных, и поэтому процесс резервного копирования проходит быстрее. Применяя эту стратегию, вы можете создавать полную резервную копию базы данных каждый день в ночное время или раз в неделю, а резервное копирование журналов базы данных — на более регулярной основе.

Восстановление по резервной копии журнала транзакций подразумевает сначала восстановление наиболее свежей полной резервной копии с последующим наложением журналов транзакций. Этот подход представляет собой более сложную схему резервного копирования по сравнению с двумя предыдущими, потому что в вашем распоряжении окажется набор файлов, созданных в разное время, и вам потребуется управлять этими файлами совместно. Далее, восстановление по журналам транзакций представляет собой наиболее длительный процесс из всех трех перечисленных.

Применение стратегии резервного копирования в облачной среде

Наилучшей стратегией резервного копирования баз данных в облачной среде будет решение, основанное на пофайловом резервном копировании. Вы можете заблокировать базу данных от записи, снять моментальный снимок, а затем разблокировать ее. Этот метод элегантен, быстр и надежен. Ключевой особенностью облачной среды является возможность снятия моментальных снимков с томов блочного хранения. Без возможности получения моментальных снимков процесс резервного копирования занимал бы слишком длительное время.

Однако ваша стратегия резервного копирования не может ограничиваться только пофайловым резервным копированием. Моментальные снимки отлично работают в единственном облаке, но они не могут применяться за пределами инфраструктуры вашего провайдера облачных услуг. Иными словами, в Amazon EC2 моментальный снимок эластичного блочного тома не может быть применен в развертывании облачной среды. Чтобы добиться возможности переноса вашего приложения из одного облака в другое, вам необходимо регулярно выполнять полный экспорт базы данных.

То, насколько часто вам требуется выполнять операции экспорта вашей базы данных, зависит от объема используемых вами данных. Основной вопрос, которым

вы должны задаться, формулируется так: "Если мой облачный провайдер внезапно приостановит обслуживание на длительный период времени, то какой объем данных я могу позволить себе потерять при развертывании новой инфраструктуры?"

Для системы управления контентом в такой экстремальной ситуации приемлемым вариантом может оказаться потеря данных, наработанных за неделю. Но для системы электронной коммерции этот вариант неприемлем, такие системы не могут позволить себе никаких потерь данных ни при каких обстоятельствах, даже самых экстремальных.

Я рекомендую подход к регулярному резервному копированию базы данных с подчиненного сервера MySQL, проиллюстрированный на рис. 4.11.

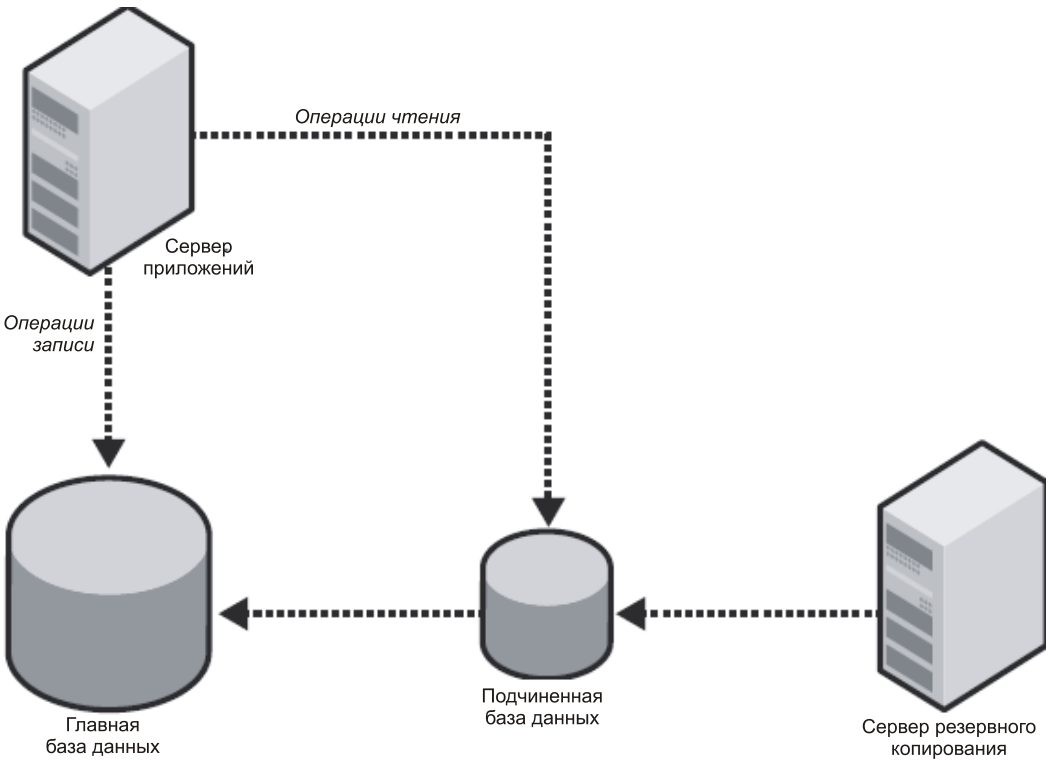


Рис. 4.11. Выполнение регулярного полного экспорта базы данных на подчиненном сервере в репликационной среде

Для целей резервного копирования не имеет большого значения то, что база данных на подчиненном сервере чуть отстает от состояния, в котором она находится на главном сервере. Особое значение имеет тот факт, что на подчиненном сервере вся база данных находится во внутренне непротиворечивом состоянии в относительно разумный момент времени. Таким образом, вы можете выполнить очень длительную процедуру резервного копирования и не беспокоиться о ее влиянии на производительность вашей системы в производственной среде. Поскольку вы мо-

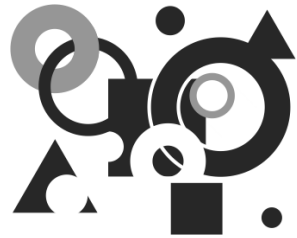
жете выполнять длительные процедуры резервного копирования, частота их выполнения зависит в основном от ваших "аппетитов".

Если ваши процедуры резервного копирования занимают уж совсем длительные временные интервалы, в течение которых ваш подчиненный сервер может сильно "отстать" от главного, имеет смысл сконфигурировать несколько подчиненных серверов и выполнять на них резервное копирование поочередно, по принципу "ротации". Такая политика "ротации" даст подчиненным серверам достаточно времени, чтобы синхронизироваться с главным сервером в промежутках между их "дежурствами" по выполнению резервного копирования.

Как только процедура резервного копирования завершится, вам следует переместить резервную копию в S3 и не забывать регулярно копировать эти резервные копии из S3 к другому облачному провайдеру или на ваш собственный внутренний файл-сервер.

К настоящему моменту архитектура вашего приложения должна быть структурирована достаточно хорошо, чтобы успешно работать не только в облачной среде Amazon, но и в других облачных инфраструктурах.

Глава 5



Безопасность

Если облачная среда и заставляет вас полностью пересмотреть все свои представления о какой-либо части вашей инфраструктуры, то, вероятнее всего, этим компонентом инфраструктуры окажется безопасность. Первый вопрос, который я слышу от большинства руководителей компании-клиента, звучит так: "Насколько серьезно стоит для меня вопрос утраты контроля над тем, где и как хранятся мои данные?" Несмотря на то, что этот вопрос волнует всех без исключения людей, только начинающих знакомиться с облачной обработкой данных и рассматривающих возможности перехода на облачную инфраструктуру, на самом деле в ней существуют другие, и намного более серьезные вопросы, связанные с безопасностью.

- Иногда судебные процессы, в которые вы не вовлечены, тем не менее, представляют угрозу для вашей безопасности.
- Многие законы и стандарты, управляющие вашей инфраструктурой IT, разрабатывались без учета виртуализации.
- Сама идея защиты периметра в сети в облачной среде утрачивает смысл.
- Управление регистрационной информацией пользователей выходит за рамки стандартных процедур управления идентификационной информацией (identity management).

Как и во многих других аспектах облачной среды, безопасность здесь тоже может быть обеспечена, и даже на более высоком уровне, чем в обычном внутреннем центре обработки данных. Эфемерная природа виртуальных экземпляров требует, чтобы вы приняли на вооружение надежные процедуры обеспечения безопасности, без которых традиционные среды хостинга спокойно обходятся. Таким образом, переход на облачные вычисления может привести к созданию компьютерной инфраструктуры повышенной защищенности.

Безопасность данных

Физическая безопасность определяет, каким образом осуществляется управление физическим доступом к серверам, поддерживающим вашу инфраструктуру. Облачная среда все еще может иметь ограничения по физической безопасности.

В конце концов, физические серверы все-таки существуют, и они где-то стоят и работают. Когда вы выбираете провайдера облачных сервисов, вам необходимо ознакомиться с их правилами обеспечения физической безопасности, и понять, какие меры вы должны принять со своей стороны, чтобы защитить свои системы от физических уязвимостей.

Управление данными

Самое глубокое и принципиальное различие между традиционными центрами обработки данных и облачной средой кроется в том, что ваши данные физически располагаются на серверах, принадлежащих не вам, а сторонней компании. Те предприятия и организации, которые прибегли к услугам аутсорсинга и доверили свои данные провайдерам управляемых услуг (MCP), частично преодолели эту пропасть между собственной физической инфраструктурой IT и облачной средой. Облачные же сервисы добавляют к этому физическую невозможность даже увидеть те серверы, на которых размещены ваши данные. Хотя этот вопрос и лежит больше в эмоциональной сфере, но, тем не менее, с ним связаны и некоторые реальные бизнес-задачи.

Основной практической проблемой является то, что некоторые посторонние факторы, не имеющие никакого отношения к вашему бизнесу, могут, тем не менее, поставить под вопрос безопасность ваших данных. Например, создать проблемы в вашей инфраструктуре может любое из следующих событий.

- Фирма, предоставляющая вам облачные сервисы, объявляется банкротом, ее имущество (в том числе серверы) конфискуется, и провайдер перестает предоставлять услуги.
- Некая третья сторона, не имеющая к вам никакого отношения (или, что еще хуже — ваш конкурент), подает в суд на вашего облачного провайдера и получает через суд полный доступ ко всем серверам, принадлежащим облачному провайдеру.
- Неспособность вашего облачного провайдера надлежащим образом защитить компоненты своей инфраструктуры — особенно в том, что касается управления физическим доступом, что может скомпрометировать ваши системы.

Решить эти проблемы позволяют следующие меры, которые вам необходимо предпринять в любом случае, но которые зачастую недооцениваются большинством пользователей: зашифруйте всю информацию и выполняйте удаленное резервное копирование.

- Зашифруйте все конфиденциальные данные в вашей базе данных и в памяти. Расшифровывайте эту информацию только в памяти и только на то время, пока существует потребность в этих данных. Кроме того, шифруйте резервные копии и все сетевые коммуникации.
- Подберите себе еще одного облачного провайдера и регулярно осуществляйте автоматизированные процедуры резервного копирования (для этой цели существуют как коммерческие решения, так и решения на основе открытого кода).

Это позволит гарантировать возможность восстановления как текущих данных, так и исторической информации, причем восстановление будет возможно даже в том случае, если ваш облачный провайдер вообще исчезнет с лица земли.

Рассмотрим применение перечисленных мер для всех возможных сценариев развития событий.

Что происходит в случае остановки деятельности облачного провайдера

Возможны различные варианты развития событий по этому сценарию: банкротство, сворачивание бизнеса для переориентирования на другую область деятельности или длительный простой вследствие масштабного и продолжительного сбоя в подаче электроэнергии. Что бы ни происходило, вы рискуете потерять доступ к вашим производственным системам вследствие действий другой компании. Кроме того, вы рискуете и тем, что организация, управляющая вашими данными, может не защитить их в соответствии с ранее заключенными соглашениями об уровне сервиса.

В *главе 6* мы еще вернемся к теме аварийного восстановления системы с резервных копий при развитии событий по такому сценарию. Наиболее важный аспект в данном случае — это регулярное осуществление удаленного резервного копирования и хранение резервных копий за пределами вашей производственной среды. Эта мера должна вас защитить от негативных последствий, вызванных приостановкой работы вашего облачного провайдера. Еще лучше пользоваться услугами еще одного облачного провайдера, с помощью которого вы можете запустить замещающую инфраструктуру.

Что происходит, когда судебная повестка вынуждает вашего облачного провайдера раскрыть ваши данные

Естественно, если вы вовлечены в судебный процесс и повестка адресована непосредственно вам, то вы обязаны предоставить данные судам, вне зависимости от того, какие предосторожности вы предпринимаете, и вне зависимости от того, где хранятся ваши данные — в облачной среде или в вашей собственной внутренней IT-инфраструктуре. Но здесь мы рассматриваем другой вариант — случай, когда повестка из суда направлена вашему облачному провайдеру, а не вам, вы не вовлечены в судебный процесс и не имеете к нему никакого отношения.

С чисто технической точки зрения, судебная повестка должна покрывать достаточно узкий круг вопросов и не должна затрагивать вас. Однако вы не можете быть полностью уверены в том, что судебные повестки, касающиеся споров между конкурирующими фирмами по поводу приоритетов в области новейших технологий, окажутся сформулированы корректно — а именно так, чтобы вовлекать в расследование достаточно узкий круг заинтересованных сторон. Кроме того, вы не може-

те быть уверены даже в том, что вы вообще узнаете о том, что повестка была выпущена.

От этого сценария вас защитит только полное шифрование ваших данных. Повестка может требовать, чтобы ваш облачный провайдер предоставил суду ваши данные и доступ к ним, но у провайдера не будет вашего ключа доступа и ключа для расшифровки. Чтобы их получить, суд должен будет обратиться к вам и отправить повестку вам. В результате вы будете иметь тот же уровень контроля над своими данными в облачной среде, как и в вашем собственном частном центре обработки данных.

Что происходит, если облачный провайдер не может обеспечить адекватную защиту собственной сети

Когда вы выбираете облачного провайдера, вы должны ознакомиться с тем, как они обеспечивают физическую защиту и как у них организована защита сети и отдельных хостов. Как бы парадоксально это ни звучало, но самым защищенным облачным провайдером окажется тот, о котором вы никогда не сможете получить подобной информации и узнать, где физически находится сервер, на котором работает ваш виртуальный экземпляр. Вероятнее всего, что если вы не можете даже приблизительно догадаться об этом, то и мотивированный взломщик, целенаправленно собирающий информацию о вашей организации, столкнется с теми же сложностями, пытаясь определить физическое расположение среды, предоставляющей вам хостинг.

Примечание

Компания Amazon не раскрывает информации о том, где физически располагаются их центры обработки данных; они просто декларируют, что каждый из центров размещается в ничем не примечательном здании с охраной периметра по типу армейской. Даже если вы узнаете, что мой сервер базы данных находится в зоне доступности us-east-1a, вы все равно не узнаете, ни где располагаются центры обработки данных, формирующие эту зону, ни даже какую из трех зон доступности на восточном побережье США представляет зона доступности us-east-1a.

Amazon публикует свои стандарты безопасности и процедуры по ее обеспечению по следующему адресу: **http://aws.amazon.com**. Услугами какого бы облачного провайдера вы ни пользовались, вы должны понимать, каким стандартам безопасности они следуют, и ожидать, что они даже превышают все ваши требования.

Впрочем, на практике ничто не может гарантировать, что ваш облачный провайдер на самом деле обеспечивает те стандарты и процедуры, которые они декларативно поддерживают. Однако если вы будете следовать рекомендациям, которые я даю в данной главе, конфиденциальность вашей информации будет надежно защищена даже от полной некомпетентности некоторых сотрудников вашего облачного провайдера.

Шифруйте все!

В облачной среде ваши данные, разумеется, где-то хранятся; просто вы не имеете точной информации о том, где именно они располагаются. Однако вы обладаете некоторыми базовыми знаниями, в том числе:

- ваши данные находятся внутри гостевой операционной системы, работающей на виртуальной машине, и у вас есть механизмы контроля за доступом к этим данным;
- сетевой трафик, которым обмениваются ваши виртуальные экземпляры, невидим для других виртуальных хостов;
- для большинства облачных сервисов хранения данных доступ к данным является по умолчанию частными (защищенным). Тем не менее, многие из них, включая и Amazon S3, дают вам возможность открыть к этим данным публичный доступ.

Шифрование сетевого трафика

Не имеет значения, насколько слабы правила безопасности, которые вы практикуете на данный момент, но в любом случае вы, скорее всего, как минимум, шифруете свой сетевой трафик — по крайней мере, в большинстве своем. Привлекательной особенностью облачной среды Amazon является то, что виртуальные серверы не могут перехватывать сетевой трафик других виртуальных серверов. Тем не менее, я все же не рекомендую полностью полагаться на эту особенность, потому что другие провайдеры могут ее и не обеспечивать. Более того, и Amazon в будущем может ввести какую-нибудь еще новую функцию, из-за которой данная возможность может оказаться устаревшей. Поэтому я рекомендую вам шифровать весь сетевой трафик, а не только Web-трафик.

Шифрование резервных копий

Когда вы формируете наборы данных для резервного копирования, вам следует шифровать их с использованием криптостойкого алгоритма, например такого, как Pretty Good Privacy (PGP)[1]. После этого вы можете хранить их в умеренно защищенной облачной среде наподобие Amazon S3, или даже в полностью незащищенной среде.

Шифрование потребляет ресурсы CPU. Поэтому я сначала рекомендую скопировать ваши файлы в незашифрованном виде на временный сервер резервного копирования, чья задача заключается в осуществлении шифрования, а затем закачать резервные копии в вашу облачную систему хранения данных. Использование промежуточного сервера резервного копирования позволит вам не только избежать

[1] См. **https://secure.wikimedia.org/wikipedia/ru/wiki/PGP**, **https://secure.wikimedia.org/wikipedia/en/wiki/Pretty_Good_Privacy**, **http://www.pgp.com/**, **http://pgpru.com/**, **http://www.pgpi.org/products/pgp/versions/freeware/**. — *Прим. перев.*

переплаты за использование ресурсов CPU на ваших серверах приложений и баз данных, но и позволит вам создать единую систему с высокой степенью защиты, хранящую ваши учетные данные для доступа к облачному хранилищу, вместо того, чтобы передавать учетные данные на каждую систему, которой требуется выполнить резервное копирование.

Шифрование файловых систем

Каждый используемый вами виртуальный сервер будет монтировать эфемерные устройства хранения данных (например, такие как раздел `/mnt` на экземплярах UNIX в EC2) или устройства блочного хранения данных. В среде EC2 отсутствие шифрования эфемерных устройств представляет умеренный риск, потому что система Xen в EC2 затирает эфемерные устройства нулями при завершении работы экземпляра. Но моментальные снимки блочных устройств хранения данных располагаются в Amazon S3 и остаются незашифрованными, если только вы не предпримете специальных мер по их шифрованию.

Примечание

Шифрование во всех его проявлениях — мера дорогостоящая. Но самой дорогой из форм шифрования является шифрование на уровне файловой системы. Моя стандартная рекомендация — зашифровать всю файловую систему, но эта мера может оказаться не слишком практичной для некоторых приложений. В конце концов, вам необходимо сбалансировать ваши требования к производительности конкретных приложений и ваши требования к защите данных. К сожалению, высока вероятность того, что на определенном уровне эти требования начнут конфликтовать, и вам придется искать компромисс, позволяющий использовать облачную инфраструктуру.

Наиболее безопасный подход к обоим сценариям заключается в монтировании эфемерных устройств и устройств блочного хранения с использованием зашифрованной файловой системы (encrypted filesystem). В облачной среде управление запуском виртуального сервера с использованием зашифрованных файловых систем оказывается проще и предлагает более высокий уровень защищенности.

Основная проблема, связанная с использованием зашифрованных файловых систем на серверах, заключается в том, каким образом вы будете управлять паролем для расшифровки. Конкретному серверу необходимо получить пароль для расшифровки прежде, чем он сможет примонтировать зашифрованную файловую систему. Наиболее общий подход к решению этой проблемы заключается в хранении пароля в незашифрованной корневой файловой системе. Так как целью шифрования файловой системы является защита от физического доступа к образу диска, хранение пароля на отдельной, незашифрованной файловой системе не настолько проблематично, как это кажется на первый взгляд, но, тем не менее, оно все-таки представляет проблему.

В облачной среде вам не требуется хранить в облаке пароль для расшифровки. Вместо этого вы можете передать пароль для расшифровки вашему новому виртуальному экземпляру при его запуске. Сервер может захватить ключ для расшифровки из одного из параметров запуска сервера, а затем примонтировать

любое эфемерное или блочное устройство, используя зашифрованную файловую систему.

Вы можете добавить дополнительный уровень безопасности, зашифровав пароль и сохранив ключ для его расшифровки в образе машины. Процесс запуска виртуального сервера, который монтирует зашифрованную файловую систему с использованием зашифрованного пароля, проиллюстрирован на рис. 5.1.

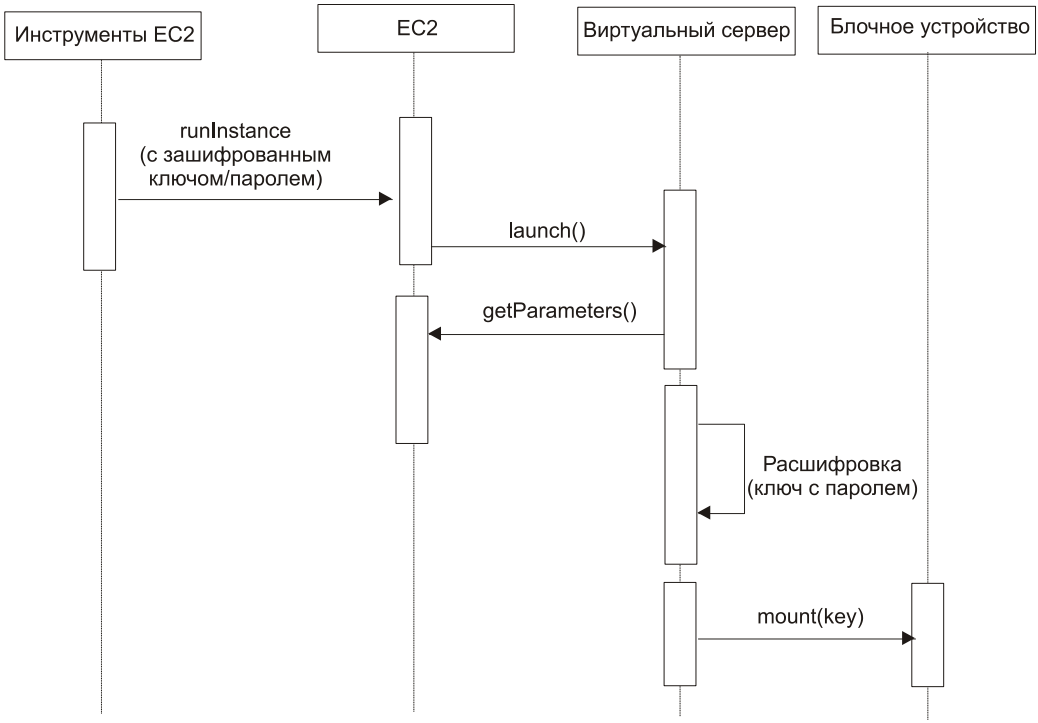


Рис. 5.1. Процесс запуска виртуального сервера, монтирующего зашифрованные файловые системы

Соответствие стандартам и нормативно-законодательным актам

Большинство проблем, вызванных соответствием или несоответствием стандартам и нормативно-законодательным актам, лежит не в самой облачной среде, а в том факте, что эти стандарты и законодательные акты были написаны для интернет-приложений до того, как виртуализационные технологии получили широкое распространение. Иными словами, существует вероятность того, что, развертывая свои приложения в облачной инфраструктуре, вы можете обеспечить соответствие духу законов и спецификаций, но не их букве.

Например, если стандарт, соответствие требованиям которого вы должны гарантировать, требует, чтобы определенные данные хранились на другом сервере, отличном от сервера, реализующего логику работы системы, то может ли виртуальный сервер обеспечить такое соответствие? Я сам с уверенностью могу сказать, что это возможно, но вот интерпретация факта соответствия или несоответствия вашей среды требованиям законов и стандартов может быть отдана на откуп юристов (адвокатов и судей) или других людей, не являющихся техническими специалистами и не понимающих природу виртуализации. Даже если некоторые законы, например закон Сарбейнза—Оксли (Sarbanes—Oxley, SOX)[1], и не предъявляют никаких специфических требований к защите информации, они все равно наводят ужас на топ-менеджеров.

Список сокращений

- Директива 95/46/EC (Directive 95/46/EC)[2] — Директива Европейского парламента и Совета Европейского Союза от 15 декабря 1997 года, касающаяся использования персональных данных и защиты неприкосновенности частной жизни в сфере телекоммуникаций, дающая определение "личным данным" и регламентирующая их хранение.

- Закон об отчетности и безопасности медицинского страхования (Health Insurance Portability and Accountability Act, HIPAA)[3] — обширный закон, регламентирующий вопросы, относящиеся к медицинскому страхованию, включая правила обеспечения защиты личных данных и безопасности их хранения.

- Стандарт защиты информации в индустрии платежных карт (Payment Card Industry Data Security Standard, PCI или PCI DSS)[4] — стандарт, разработанный международными платежными системами Visa и MasterCard. Объединяет в себе требования ряда программ по защите информации при обработке транзакций с платежными картами.

- Закон Сарбейнза—Оксли (Sarbanes—Oxley, SOX) — устанавливает законодательные требования, обязательные для соблюдения открытыми акционерными обществами при информировании своих акционеров.

---

[1] Закон Сарбейнза—Оксли (Sarbanes—Oxley Act, SOX) был подписан 30 июля 2002 года и представляет собой одно из самых значительных событий по изменению федерального законодательства США по ценным бумагам за последние 60 лет. Значительно ужесточает требования к финансовой отчетности и к процессу ее подготовки — результат многочисленных корпоративных скандалов, связанных с недобросовестными менеджерами крупных корпораций. Подробнее см. **http://tinyurl.com/35efa7r**, **http://tinyurl.com/3xe5qu3.** — *Прим. перев.*

[2] Подробнее см. **https://secure.wikimedia.org/wikipedia/en/wiki/Data_Protection_Directive**, **http://03.rsoc.ru/docs/archive/03/documents/direktiva_97_66_ES_15.12.1997.doc**. — *Прим. перев.*

[3] Подробнее см. **http://en.wikipedia.org/wiki/Health_Insurance_Portability_and_Accountability_Act**, **https://secure.wikimedia.org/wikipedia/en/wiki/Health_Insurance_Portability_and_Accountability_Act**. — *Прим. перев.*

[4] Подробнее см. **https://secure.wikimedia.org/wikipedia/ru/wiki/PCI_DSS**, **http://www.osp.ru/news/articles/2010/33/13003470/** — *Прим. перев.*

- Стандарт 21 CFR 11 (Title 21 CFR Part 11 of the Federal Code of Regulations, 21CFR11)[1] — нормативный документ американского органа по контролю за управлением электронными подписями и электронными данными в области выпуска пищевых продуктов, медицинской техники и лекарственных препаратов.

С точки зрения безопасности, при обеспечении соответствия требованиям стандартов и законодательных актов[2] вы можете встретить следующие три основных проблемы.

- "Как" — вопросы этого типа касаются таких стандартов, как PCI DSS или законодательных актов наподобие HIPAA или SOX, регламентирующих, каким образом приложения определенного типа должны работать с целью защитить информацию, специфическую для конкретной области. Например, HIPAA определяет, каким образом должна осуществляться обработка персональных идентификационных данных в области здравоохранения и медицинского страхования.
- "Где" — эти вопросы проистекают из требований таких законодательных актов, как Директива 95/46/ЕС, которая определяет требования к местам хранения определенной информации. Ключевым фактором влияния этой конкретной директивы является то, что персональные данные граждан Евросоюза не могут храниться в США (или любой другой стране, где требования к хранению персональной информации не соответствуют требованиям к ее хранению, действующим в ЕС).
- "Что" — эти вопросы задаются на основе стандартов, предписывающих иметь в системе вполне конкретные компоненты. Например, стандарт защиты информации в индустрии платежных карт (PCI DSS) предписывает использование антивирусного ПО на всех серверах, где производится обработка данных по кредитным картам.

На сегодняшний день наиболее важным аспектом является то, что система, развернутая в облачной среде, может, соблюдая дух закона, соблюдать или не соблюдать его букву. Для некоторых спецификаций можно обеспечить соблюдение буквы закона за счет реализации смешанной архитектуры, в состав которой будут входить некоторые физические элементы и некоторые виртуальные. Облачные инфраструктуры, которые специализируются на гибридных решениях, могут оказаться наилучшим вариантом. С другой стороны, возможно, имеет смысл присмотреться к тем поставщикам, которые предлагают в качестве сервиса некую часть вашей системы, для которой вы обязаны соблюдать некоторые специфические требова-

---

[1] Подробнее см. **https://secure.wikimedia.org/wikipedia/en/wiki/Title_21_CFR_Part_11**. — *Прим. перев.*

[2] В свете предстоящего вступления России в ВТО законодательство РФ тоже должно будет подвергнуться доработке с тем, чтобы соблюдать международные правовые нормы. Во всяком случае уже сейчас предприятия, выходящие на международный рынок, должны соблюдать перечисленные в этой книге законодательные акты. О существующих проблемах можно прочесть, например, здесь: **http://www.gdm.ru/meropr/18.10.2004/2839/book/hramtsovskaya1/**. Информационные порталы, посвященные законам и стандартам в области информационной безопасности: **http://www.securit.ru/solutions/laws/**, **http://www.iso27000.ru/**. — *Прим. перев.*

ния. Например, если один из компонентов вашего сайта реализует электронную коммерцию, вы можете воспользоваться услугами одного из поставщиков, предоставляющих сервисы электронной коммерции, которые гарантированно обеспечивают выполнение требований стандарта защиты информации в индустрии платежных карт (PCI DSS)[1].

В смешанной среде вы не храните в облачной инфраструктуре никаких конфиденциальных данных. Вместо этого вы переводите обработку конфиденциальной информации на предназначенные для этой цели серверы в физическом центре обработки данных, который находится полностью под вашим контролем. Например, вы можете держать сервер обработки данных о кредитных картах у сервис-провайдера услуг внешнего управления, а запросы к этому серверу (например, на сохранение номеров кредитных карт или запросы на списание денег со счета) могут поступать из облачной среды.

Что касается места хранения конфиденциальных данных, то Amazon предоставляет хранилища S3, расположенные на территории Евросоюза. Поэтому комбинация облачных сервисов Amazon и хранилища S3 позволяет обеспечить соблюдение требований Директивы 95/46/EC (Directive 95/46/EC).

Сетевая безопасность

Как уже говорилось ранее, облако Amazon не имеет периметра. Вместо этого EC2 предоставляет группы безопасности, которые обеспечивают правила пропускания сетевого трафика, аналогичные правилам, задаваемым при конфигурировании брандмауэра (firewall). Группы безопасности позволяют управлять трафиком, который достигает виртуальных серверов в этой группе. Хотя я часто говорю о группах безопасности так же, как если бы они представляли собой сетевые сегменты, защищенные брандмауэром, в действительности они не являются виртуальными сетевыми сегментами, потому что:

- два сервера в двух различных зонах доступности Amazon EC2 могут работать в составе одной группы безопасности;
- сервер может принадлежать более, чем к одной группе безопасности;
- серверы, находящиеся в одной группе безопасности, могут вообще не иметь возможности поддерживать между собой какие бы то ни было коммуникации;
- серверы, находящиеся в одном и том же сетевом сегменте, могут иметь разные характеристики IP — они даже могут находиться в разных адресных пространствах;

[1] О стандарте PCI DSS в России см. следующие ресурсы: **http://www.pcisecurity.ru/**, **http://www.dsec.ru/consult/pcidss/**, **http://www.pcidss.ru/download/**, **http://www.iso27000.ru/standarty/pci-dss-standart-zaschity-informacii-v-industrii-platezhnyh-kart-1**. — *Прим. перев.*

- ни один сервер в EC2 не может "видеть" входящий и исходящий сетевой трафик других серверов (для других облачных систем это не обязательно должно быть так). Если вы попытаетесь разместить ваш виртуальный сервер Linux в неизбирательном режиме, т. е. в режиме приема всех сетевых пакетов (promiscuous mode), то единственный сетевой трафик, который вы сможете увидеть, будет ваш собственный входящий и исходящий трафик.

Правила брандмауэра

Как правило, брандмауэр (firewall) защищает периметр одного или нескольких сетевых сегментов. Защита периметра сети с помощью брандмауэра показана на рис. 5.2.

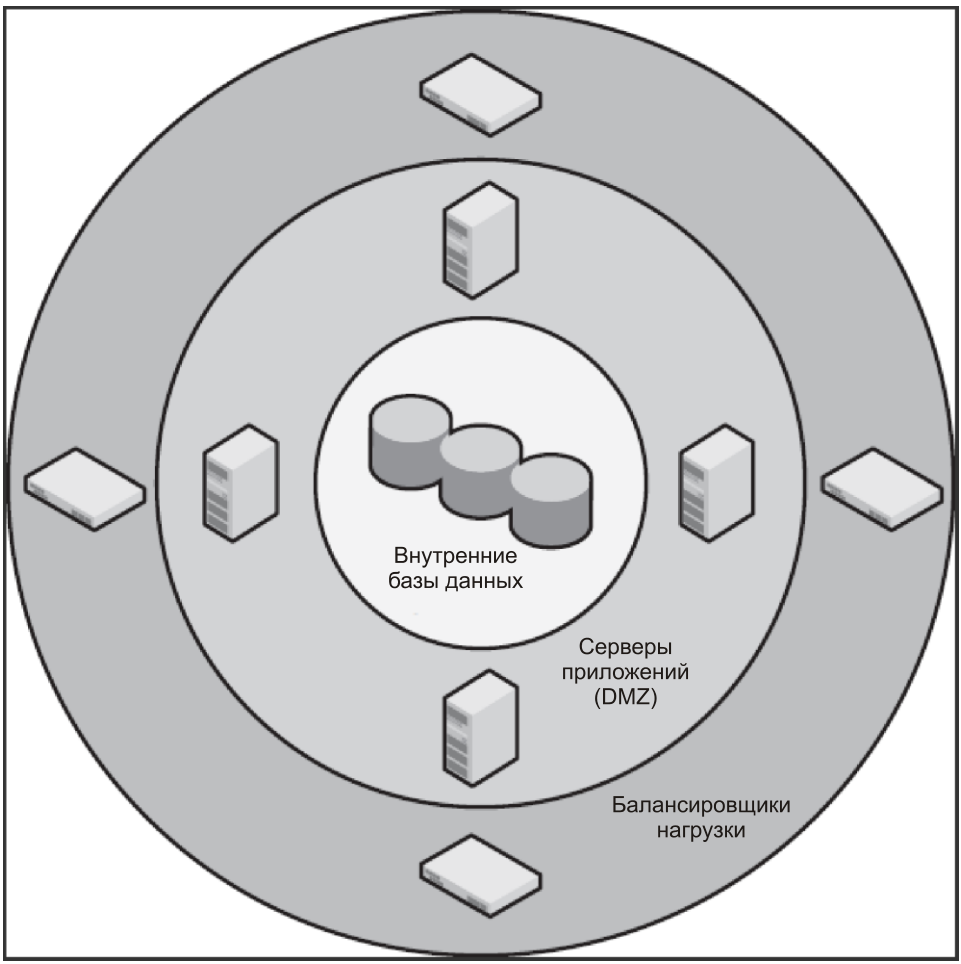


Рис. 5.2. Брандмауэры являются основным инструментом защиты периметра

Основной брандмауэр защищает наружный периметр, пропуская только трафик HTTP, HTTPS и (иногда) FTP[1]. Между защищаемым сетевым сегментом и наружным периметром находятся пограничные системы, например балансировщики нагрузки, которые направляют трафик в так называемую "демилитаризованную зону" (DMZ), защищенную другим брандмауэром. В демилитаризованной зоне располагаются серверы приложений, которые направляют запросы к базам данных через третий брандмауэр во внутреннюю защищенную сеть, где находятся внутренние базы данных, хранящие конфиденциальную информацию.

Такая структура подразумевает, что для получения доступа к данным по возрастанию уровня секретности организуются несколько уровней (или периметров) сетевой защиты с помощью брандмауэров. Основным преимуществом такой архитектуры является то, что даже если правила брандмауэра, защищающего внутреннюю сеть, сформулированы плохо, они не обязательно открывают ее для доступа извне, за исключением тех случаев, когда демилитаризованная зона тоже уже скомпрометирована. В дополнение, общая тенденция заключается в том, что внешние сервисы сильнее защищены от уязвимостей Интернета, в то время как внутренние сервисы менее ориентированы на Интернет. Слабость же этой инфраструктуры состоит в том, что компрометация любого из внутренних серверов в пределах конкретного сегмента автоматически предоставляет полный доступ и к другим серверам в этом сетевом сегменте.

На рис. 5.3 приведено визуальное представление концепции правил брандмауэров в облачной среде Amazon. Как видите, эта концепция в облачной инфраструктуре сильно отличается от концепции, принятой в традиционных центрах обработки данных.

Все виртуальные серверы находятся в сети на одном уровне, а управление трафиком осуществляется посредством определения *групп безопасности* (security group). Нет ни сетевых сегментов, ни периметра. Членство в одной и той же группе безопасности не предоставляет привилегированного доступа к другим серверам, принадлежащим к той же группе безопасности, за исключением того случая, когда вы явно определите правила, предоставляющие привилегированный доступ. Наконец, отдельный сервер может быть членом нескольких различных групп безопасности. Правила, определенные для конкретного сервера, представляют собой объединение (union) правил для всех групп, к которым этот сервер принадлежит.

Вы можете определить группы безопасности таким образом, чтобы имитировать традиционную защиту сетевого периметра. Например, вы можете создать следующую конфигурацию:

- создать пограничную группу безопасности (border security group), которая бы прослушивала весь трафик через порты 80 и 443;

[1] Не следует пропускать в свою сеть трафик FTP. FTP — это небезопасный протокол, и на протяжении всей его истории в различных реализациях было найдено множество уязвимостей. Вместо FTP пользуйтесь SCP (secure copy) — протоколом копирования файлов, использующим в качестве транспорта не RSH (Remote Shell), а SSH (Secure Shell).

- создать группу безопасности DMZ, которая бы прослушивала трафик, исходящий из пограничной группы, через порты 80 и 443;
- создать внутреннюю группу безопасности, которая бы прослушивала трафик, исходящий из группы DMZ, через порт 3306.

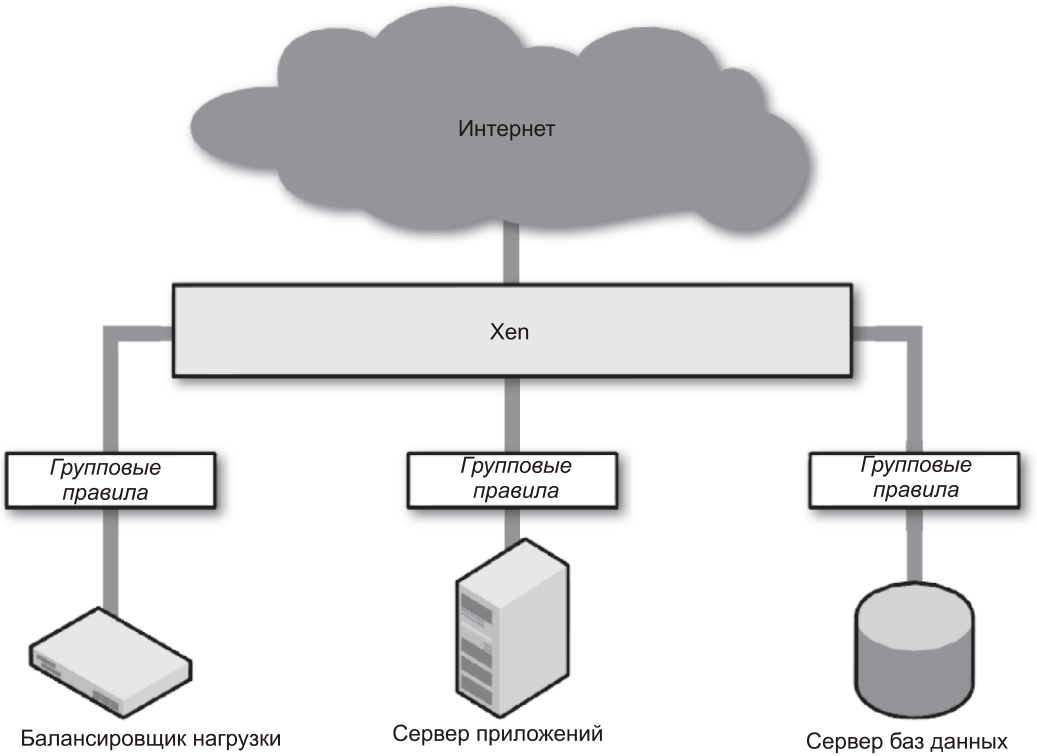


Рис. 5.3. В облачной среде нет периметра и сетевых сегментов

Внимание!

Система безопасности Amazon EC2 пока не позволяет ограничивать доступ через порты при определении правил доступа из одной группы безопасности в другую. Вероятно, эта возможность появится в будущем. Тем не менее, вы можете имитировать эту возможность за счет определения правил на основе исходного IP-адреса для каждого сервера в исходной группе.

Как и в случае с традиционной защитой периметра, доступ к серверам, принадлежащим к вашей внутренней группе безопасности, можно получить лишь тогда, когда предварительно будет скомпрометирована сначала пограничная группа, затем — DMZ, и, наконец — один из внутренних серверов. В отличие от традиционной защиты периметра, здесь существует возможность того, что вы случайно предоставите глобальный доступ к внутренней зоне и, таким образом, откроете ее для

вторжений. Но в облачной среде, если атакующий и скомпрометирует один из серверов внутренней зоны, не получит автоматического доступа к другим серверам из этой зоны, если только не воспользуется оригинальным эксплоитом. Иными словами, доступ к одному из серверов во внутренней зоне не обязательно означает доступ к другим серверам в пределах этой зоны.

Подход Amazon позволяет использовать и такие возможности, которые были недоступны в традиционной инфраструктуре. Например, вы можете с легкостью обеспечить прямой доступ через SSH к любому виртуальному серверу в вашей облачной инфраструктуре из вашей корпоративной сети, не используя для этого виртуальную частную сеть (Virtual Private Network, VPN). При этом вы сохраняете возможность использования преимуществ, даваемых традиционным подходом к защите сети по периметру, когда речь идет об открытом доступе в Интернет, и можете получать быстрый доступ к вашим серверам для управления ими из критических точек.

Такая архитектура системы безопасности предоставляет два основных преимущества.

- Поскольку вы удаленно управляете правилами брандмауэра, атакующий не имеет единой мишени для своей атаки, как в случае с физическим брандмауэром.
- У вас нет возможности случайно разрушить правила защиты сети и таким образом навсегда блокировать любой доступ в данный сетевой сегмент.

Я рекомендую воспользоваться подходом, который имитирует традиционную защиту сетевого периметра, потому что этот подход к управлению сетевым трафиком хорошо изучен и прост для понимания. Если вы воспользуетесь этим подходом, важно понимать, что вы создаете только виртуальные эквиваленты физических сетевых сегментов традиционной физической инфраструктуры. Настоящих уровней сетевой безопасности, которые имеются в традиционной конфигурации, у вас нет.

Рекомендации по наиболее эффективной организации сетевой системы безопасности в облаке сводятся к следующему.

- *На каждом виртуальном сервере запускайте только один сетевой сервис (плюс все сервисы, необходимые для администрирования)*. Каждый новый сетевой сервис, присутствующий в системе, представляет собой вектор атаки. Если вы сосредоточите на одном сервере множество сервисов, вы создадите множество векторов атаки, потенциально позволяющих получить доступ к данным, хранящимся на этом сервере или для использования этого сервера для получения прав доступа к остальной сети.
- *Не предоставляйте открытого доступа к данным, имеющим высший уровень секретности*. Если получение несанкционированного доступа к вашей клиентской базе данных требует компрометации балансировщика нагрузки, сервера приложений и сервера базы данных (и при этом вы следуете рекомендации запускать только один сервис на каждом из серверов), злоумышленнику потребуется реализовать целых три различных вектора атаки прежде, чем он сможет добраться до этих данных.

- *Открывайте только те порты, которые абсолютно необходимы для поддержания сервиса, предоставляемого конкретным сервером, и не более того*. Разумеется, защита каждого из серверов должна быть усилена таким образом, чтобы на нем работал только один сервис — тот, который изначально был предназначен для работы на нем. Иногда бывает и так, что вы непреднамеренно запускаете на сервере те сервисы, которые изначально не предназначались для работы на данном сервере, или же вы оказываетесь в ситуации, когда в составе сервиса обнаруживается эксплоит, не требующий доступа от имени root (nonroot exploit), который позволяет атакующему запустить еще один сервис с помощью эксплоита, требующего доступа от имени root (root exploit). Блокируя доступ ко всему, за исключением вашего целевого сервиса, вы можете предотвратить использование этих типов эксплоитов.
- *Ограничьте доступ к вашим сервисам, предоставляя его только тем клиентам, которые действительно в них нуждаются*. Естественно, ваши балансировщики нагрузки должны открывать Web-порты 80 и 443 для всего трафика. В открытом доступе нуждаются только эти два протокола и конкретный сервер. Для любого другого сервиса трафик должен быть ограничен конкретными исходными адресами или группами безопасности.
- *Даже если вы не осуществляете балансировку нагрузки, используйте обратный прокси (reverse proxy)*[1]. Обратный прокси представляет собой Web-сервер, например Apache, который маршрутизирует трафик от клиента к серверу. За счет использования прокси-сервера вы можете усложнить для злоумышленника атаку на вашу инфраструктуру. Во-первых, Apache и IIS намного лучше справляются с "боевыми задачами" по отражению сетевых атак, чем любой из серверов приложений, которые вы можете использовать. В результате вероятность проникновения эксплоита будет значительно снижена, а вероятность его обезвреживания и скорость выпуска "заплатки" (patch) существенно повысятся. Во-вторых, использование эксплоита на прокси-сервере оставит атакующего буквально "ни с чем" — никакого доступа он все равно не получит. Атакующим в любом случае придется искать дополнительную уязвимость на самом вашем сервере приложений.
- *Используйте динамическую природу облачной среды для автоматизации устранения проблем с сетевой безопасностью*. Признайте это. Вам все равно требуется открывать порты на вашем брандмауэре, чтобы решить некоторые серьезные бизнес-задачи. Возможно, вы открыли порт FTP на Web-сервере, потому что у одного из ваших ключевых заказчиков появилась настоятельная необходимость использовать анонимный доступ через FTP для пакетной закачки фай-

[1] Обратный прокси — это прокси-сервер, который, в отличие от прямого, ретранслирует запросы клиентов из внешней сети на один или несколько серверов, логически расположенных во внутренней сети. Обычно обратные прокси-серверы устанавливаются перед Web-серверами. Часто используется для балансировки сетевой нагрузки между несколькими Web-серверами и повышения их безопасности, играя при этом роль брандмауэра (межсетевого экрана) на прикладном уровне. Подробнее см. **http://tinyurl.com/36smmyu**, **http://en.wikipedia.org/wiki/Reverse_proxy**. — *Прим. перев.*

лов. Вместо того, чтобы держать этот порт постоянно открытым, вы можете открывать его в заранее определенное время суток и держать открытым в течение оговоренного с клиентом промежутка времени, а затем вновь закрывать. Возможно, вы даже решите создавать временный сервер, выполняющий роль FTP-сервера для пакетной загрузки, обрабатывать закачанный файл, а затем вновь останавливать этот сервер.

Рекомендации из приведенного списка не являются чем-то новым — это стандартные меры безопасности. Облачная среда предоставляет относительно простой путь для их реализации, и указанные меры играют важную роль в обеспечении безопасности вашей облачной инфраструктуры.

Системы обнаружения сетевых вторжений

Защита сети по периметру часто включает в свой состав системы обнаружения вторжений из сети (network intrusion detection systems, NIDS), например такие, как Snort[1], которые осуществляют мониторинг локального трафика с целью выявления всего, что выглядит нестандартным или просто необычным. В качестве примеров нестандартного трафика можно привести:

- попытки сканирования портов (Port scans);
- атаки по типу Denial-of-Service (DoS-атаки);
- попытки использования эксплойтов, эксплуатирующих известные уязвимости.

Обнаружение сетевых вторжений осуществляется либо путем маршрутизации всего трафика через систему, выполняющую его анализ, либо же путем пассивного мониторинга трафика с одного из компьютеров в вашей локальной сети. В облачной среде Amazon возможен только первый из этих двух вариантов; использование же второго варианта не имеет смысла, так как любой из экземпляров EC2 способен "видеть" только собственный трафик.

Предназначение сетевых систем обнаружения вторжений

Сетевые системы обнаружения вторжений предназначены для того, чтобы предупредить вас о возможности атак до их начала и, в некоторых случаях, для отражения уже начавшихся атак. Однако вследствие структуры облачной среды Amazon многие из задач, выполняемых с помощью NIDS, просто теряют смысл. Например, NIDS обычно предупреждает администратора сети о попытках сканирования портов, считая это свидетельством того, что злоумышленник готовится к атаке. Однако в облаке Amazon вы, скорее всего, просто не заметите попытки сканирования портов, потому что ваша NIDS будет осведомлена только о запросах к

[1] Snort — это свободная сетевая система предотвращения вторжений (Intrusion Prevention System, IPS) и сетевая система обнаружения вторжений (Intrusion Detection System, IDS) на основе открытого исходного кода (Open Source), способная выполнять регистрацию пакетов и в реальном времени осуществлять анализ трафика в IP-сетях.
Подробнее см. **http://www.snort.org/**, **http://ru.wikipedia.org/wiki/Snort**. — *Прим. перев.*

портам, разрешенных правилами вашей группы безопасности. Весь остальной трафик будет для NIDS невидимым, и, следовательно, не будет восприниматься как попытка сканирования портов.

Сканирование портов и облачная среда Amazon

Когда злоумышленники ищут уязвимости в целевой сети, избранной мишенью для атаки, одной из первых задач, которые они пытаются выполнить, является сканирование портов известного сервера с последующим исследованием серверов, имеющих "соседние" IP-адреса. Этот подход не позволяет получить особо полезной информации, если применить его в облачной инфраструктуре, что является следствием целого ряда причин.

- Узлы, имеющие близкие IP-адреса, почти всегда не имеют друг к другу никакого отношения. В результате вы не сможете узнать ничего об архитектуре сети конкретной организации только за счет сканирования портов.
- Группы безопасности Amazon по умолчанию перекрывают весь входящий трафик, и запросы к портам, которые не открыты, попросту останутся без ответа. В результате лишь немногие порты на конкретном сервере окажутся действительно открытыми. Далее, сканирование всех портов — это очень медленный процесс, потому что все закрытые порты будут давать тайм-аут вместо активного отказа пропустить трафик.
- Amazon имеет собственные системы обнаружения вторжений и не позволяет клиентам осуществлять сканирование портов на серверах, принадлежащих Amazon. В результате сканирование активных портов, вероятнее всего, будет блокировано еще до того, как будет собрана хоть какая-то информация.

Как и в случае со сканированием портов, сетевые системы обнаружения вторжений Amazon активно выявляют атаки по типу "отказ в обслуживании " (Denial-of-Service, DoS) и, скорее всего, выявят попытки осуществления такой атаки намного раньше, чем ваша собственная программная система обнаружения вторжений.

Одной из сторон, в отношении которых дополнительная система обнаружения атак может оказаться полезной и в облачной среде, является ее способность обнаруживать вредоносное содержимое сетевых пакетов, поступающих в вашу сеть. Когда NIDS обнаруживает, что трафик содержит вредоносное содержимое, она может блокировать этот трафик или отправить вам предупреждение, позволяющее вам своевременно отреагировать на попытку атаки. Даже если вредоносное содержимое будет доставлено по адресу и сможет скомпрометировать сервер, вы сможете быстро отреагировать и предотвратить нанесение вам ущерба.

Внимание!

Не следует осуществлять сканирование портов на ваших собственных серверах, расположенных в облачной инфраструктуре. Как правило, когда вы усиливаете защиту вашей сетевой инфраструктуры, вы воспользуетесь такими средствами, как NESSUS[1], с тем, чтобы выявить уязвимости, имеющиеся в вашей сети. Эти инструменты в качестве одного из этапов тестирования выполняют сканирование портов. Amazon и некоторые другие

[1] NESSUS — программа для автоматического поиска известных уязвимостей в защите информационных систем. Она способна обнаружить наиболее часто встречающиеся виды уязвимостей, включая наличие уязвимых версий сервисов, ошибки в конфигурации, наличие паролей по умолчанию, пустых или слабых паролей и т. д. Подробнее см. **http://www.nessus.org/nessus/**, **http://blog.tenablesecurity.com/**, **http://www.securitylab.ru/analytics/216317.php**. — *Прим. перев.*

провайдеры облачных сервисов явным образом запрещают осуществление такой деятельности, и использование такого инструментария может привести к нарушению условий обслуживания с вашей стороны.

Реализация системы обнаружения сетевых вторжений в облачной среде

Как я уже упоминал в предыдущем разделе, в облачной среде Amazon (как и в других облачных инфраструктурах, скрывающих трафик локальной сети) вы просто не сможете развернуть сетевую систему обнаружения вторжений, которая пассивно прослушивает сетевой трафик. Вместо этого вам следует запускать NIDS на ваших балансировщиках нагрузки или на каждом из серверов в облачной инфраструктуре. Каждый из этих двух подходов имеет свои преимущества и недостатки. Но вот я, например, не являюсь сторонником применения NIDS в облачной среде за исключением тех случаев, когда это явно предписывается стандартом или законодательным актом, которые вы обязаны соблюдать.

Простейшим подходом к использованию выделенного сервера NIDS является его установка на выделенный сервер перед сетью, который осуществляет мониторинг всего входящего трафика и, в случае обнаружения потенциально вредоносной активности, действует соответствующим образом. Пример такой архитектуры приведен на рис. 5.4.

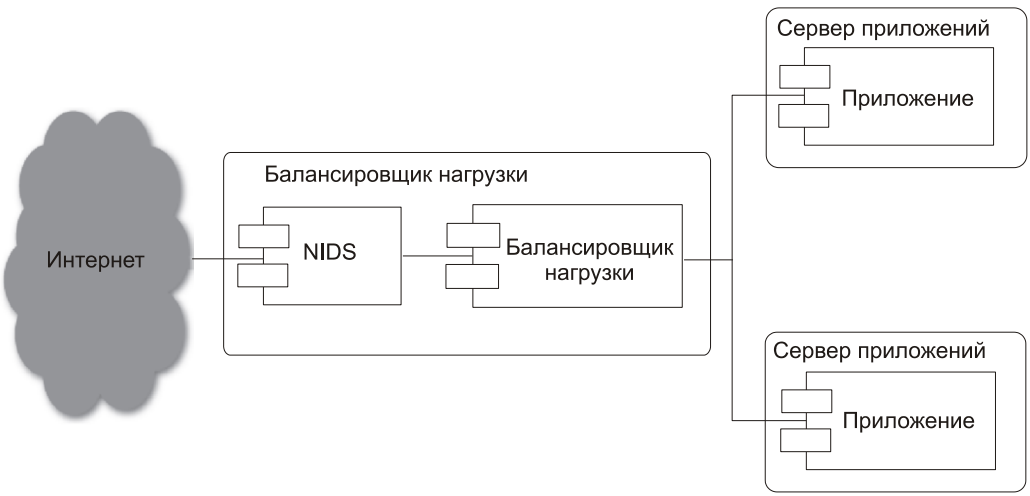


Рис. 5.4. Сетевая система обнаружения вторжений, осуществляющая мониторинг трафика на балансировщике нагрузки

Поскольку на балансировщике нагрузки работают только программная система NIDS и сервер Apache, профиль атаки на балансировщике нагрузки будет очень низок. Компрометация сервера NIDS требует наличия уязвимости в программном

обеспечении NIDS или сервера Apache (в предположении того, что защита всех остальных компонентов системы надлежащим образом усилена, и ни один из реальных сервисов не выполняет прослушивания других портов, открытых для Web в целом).

В этом случае балансировщик нагрузки является единственной точкой отказа вашей сетевой системы обнаружения вторжений, потому что в общем случае балансировщики нагрузки представляют собой системные компоненты, наиболее открытые для атак. Обнаружив способ компрометации вашего балансировщика нагрузки, атакующий не только может захватить контроль над балансировщиком нагрузки, но и получить возможность остановить процесс обнаружения вторжений.

Альтернативный вариант заключается в реализации системы обнаружения вторжения на сервере за балансировщиком нагрузки, который действует как промежуточная точка между балансировщиком нагрузки и остальными компонентами системы. В общем случае такой подход лучше, чем предыдущий, за исключением того, что он оставляет открытым балансировщик нагрузки (исследуется только трафик, который прошел через балансировщик нагрузки) и понижает общую доступность системы.

Еще один подход к реализации системы обнаружения вторжений это ее установка на всех серверах в сети. Этот подход немного повышает общий профиль атаки на систему в целом, потому что вы приходите к тому, что на всех серверах работает распространенное программное обеспечение. Уязвимость в вашей NIDS приведет к тому, что эта уязвимость будет присутствовать на каждом из серверов в вашей облачной архитектуре. В качестве позитивной особенности такой архитектуры можно заметить, что она серьезно мешает атакующим "заметать следы" их вредоносной деятельности.

Как я уже упоминал ранее, я не являюсь сторонником использования сетевой системы обнаружения вторжений в облачной среде Amazon[1]. В отличие от традиционной инфраструктуры, здесь нет особо значимых поводов для использования NIDS. Вы просто не сможете разработать такую архитектуру NIDS, которая обеспечивала бы вашей системе NIDS возможность "видеть" весь трафик, направляющийся к вашим экземплярам. Лучшее из того, что вы можете предпринять, заключается в создании такой реализации, когда NIDS развертывается на каждом из серверов вашей инфраструктуры, чтобы иметь возможность "видеть" весь трафик, который Amazon пропускает в группу безопасности, к которой принадлежит конкретный экземпляр. У вас будут минимальные возможности, доступные для превентивного уведомления, а основным преимуществом будет защита от вредоносного содержимого сетевых пакетов. Но, если вы шифруете весь трафик, даже это преимущество будет сведено к минимуму. С другой стороны, присутствие NIDS

[1] Я подчеркиваю — именно в облачной среде Amazon. В вашем традиционном центре обработки данных или в облачных инфраструктурах, предоставляемых другими поставщиками облачных сервисов, вы можете получить серьезные преимущества за счет установки сетевых систем обнаружения вторжений.

серьезно снизит производительность этих серверов и создаст единый вектор атаки на все хосты в вашей инфраструктуре.

Защита хостов

Защита хостов описывает, каким образом ваш сервер сконфигурирован для выполнения следующих задач:

- предотвращение атак;
- минимизация влияния успешно проведенной атаки на всю вашу систему в целом;
- реакция на начавшиеся атаки.

В любом случае самый выигрышный подход заключается в использовании такого программного обеспечения, в котором нет изъянов в системе безопасности. Но ведь это просто вопрос везения! В реальной жизни наилучшим подходом к предотвращению атак будет предположение о том, что ваше программное обеспечение все-таки не свободно от уязвимостей. Как я уже отмечал ранее в этой главе, каждый сервис, запускаемый вами на конкретном хосте, представляет собой отдельный вектор атаки на данный хост. Чем больше векторов атаки, тем выше вероятность, что атакующий обнаружит один из них с помощью эксплойта системы безопасности. Таким образом, вам следует стремиться к тому, чтобы на каждом отдельном хосте работал лишь минимально необходимый набор программного обеспечения.

Исходя из предположения о том, что ваши сервисы уязвимы, наилучшим средством предотвращения атак на уязвимости ваших серверов является быстрое развертывание "заплат" (patches), закрывающих известные уязвимости в системе безопасности. Именно здесь динамическая природа облачной среды радикально меняет подходы к обеспечению безопасности. В традиционном центре обработки данных развертывание обновлений, закрывающих бреши в системе безопасности, по всей инфраструктуре является длительным и рискованным процессом. В облачной среде развертывание "заплат", латающих "дыры" в системе безопасности, по всей вашей облачной инфраструктуре состоит из следующих трех очень простых шагов:

1. Установка обновлений безопасности во все ваши AMI.
2. Тестирование результатов.
3. Перезапуск всех ваших виртуальных серверов.

Здесь инструмент для управления вашей инфраструктурой, наподобие enStratus или RightScale, приобретает важнейшее значение. Если вам требуется выполнять все эти три шага вручную, облачная среда превращается в источник постоянной головной боли. Но управляющие средства позволяют автоматизировать развертывание обновлений системы безопасности, минимизируя ручные операции, время простоя и потенциальные возможности простоя вследствие человеческой ошибки.

Усиление защиты системы

Предотвращение вторжений начинается с настройки вашего образа машины. По мере приобретения опыта вы сможете экспериментировать с различными конфигурациями и постоянно перестраивать ваши образы. Как только вы обнаружите конфигурацию, хорошо работающую для конкретного сервисного профиля, вы сможете предварительно усилить защиту системы перед развертыванием этого образа.

Процесс усиления защиты сервера заключается в блокировании ненужных сервисов и исключении пользовательских учетных записей, не имеющих важного значения. Намного повысить эффективность этого процесса могут инструменты наподобие Bastille Linux. Как только вы установите Bastille Linux, вы сможете выполнить интерактивные скрипты, задающие вам вопросы о вашем сервере. После того как вы ответите на все вопросы, скрипт блокирует сервисы и учетные записи, которые не являются абсолютно необходимыми для того, чтобы ваш сервер мог выполнять поставленные перед ним задачи. В частности, скрипт убедится в том, чтобы ваша система с усиленной безопасностью соответствовала следующим критериям.

- На сервере не работают никакие сетевые сервисы, кроме тех, которые абсолютно необходимы для поддержания функциональных возможностей, предоставляемых этим сервером.
- В системе заблокированы все пользовательские учетные записи, за исключением тех, которые нужны для предоставления доступа к серверу тем пользователям, которым он требуется, и более никому.
- Все конфигурационные файлы для всех программ, работающих на сервере, настроены так, чтобы обеспечивать наивысший уровень защищенности.
- Все необходимые сервисы работают от имени непривилегированных пользовательских учетных записей (например, MySQL должен запускаться от имени пользователя mysql, а не root).
- Всегда, когда это только возможно, сервисы должны работать в закрытой файловой системе (restricted filesystem), например в "тюрьме" chroot (chroot jail).

Прежде чем приступать к комплектации вашего машинного образа, вам необходимо удалить все интерактивные учетные записи пользователей и пароли, хранящиеся в конфигурационных файлах. Хотя машинный образ и хранится в зашифрованном формате, Amazon держит ключи шифрования, которые он может быть вынужден передать третьим сторонам по повестке из суда.

Антивирусная защита

Некоторые законодательные акты и стандарты требуют, чтобы на ваших серверах была реализована система антивирусной защиты. Это действительно проблемное требование, потому что система антивирусной защиты с эксплойтом и сама яв-

ляется вектором атаки, и, в некоторых операционных системах, процент эксплойтов для антивирусного ПО относительно высок для известных вирусов.

Лично я испытываю по отношению к антивирусным системам смешанные чувства. При определенных обстоятельствах они, безусловно, необходимы, тогда как в некоторых случаях могут представлять собой дополнительный фактор риска. Например, если вы допускаете закачку на ваши серверы фотографий или других файлов, которые могут содержать в своем составе вирусы, то эти вирусы впоследствии могут быть доставлены пользователям, значит, вы просто обязаны использовать какую-нибудь из систем антивирусной защиты, чтобы исключить возможность использования вашего сайта в качестве средства распространения вируса.

К сожалению, не все антивирусные системы разработаны одинаково. Некоторые из них написаны гораздо лучше других и защищают вас надежнее других аналогичных продуктов. Наконец, некоторые серверы просто не имеют рабочего профиля, который делает вирусы (viruses), черви (worms) и "троянские кони" (Trojans) векторами атаки. Поэтому меня раздражают стандарты, законодательные акты и другие требования, которые требуют полной защиты всей системы с помощью антивирусного ПО.

При рассмотрении вопроса антивирусной защиты вам в первую очередь необходимо определить, каковы ваши собственные требования. Если вы обязаны реализовать систему антивирусной защиты, то вы, безусловно, должны выполнить это требование. При этом вам следует обратить особое внимание на следующие два аспекта, касающиеся выбора антивирусного ПО.

- Насколько широк диапазон защитных мер, предлагаемых выбранным антивирусным пакетом? Иными словами, каков процент известных эксплоитов, которые покрываются выбранным антивирусным пакетом[1]?
- Каков средний временной интервал между моментом выпуска вируса "в диком состоянии" и моментом, начиная с которого используемый вами антивирусный продукт начинает предоставлять от него защиту?

После того как вы выберете поставщика антивирусного ПО и развернете продукт на ваших серверах, вы обязаны будете поддерживать вашу базу данных вирусных сигнатур в актуальном состоянии. Вполне возможно, что вы окажетесь в большей безопасности, не имея антивирусной системы вообще, чем имея устаревшие версии антивирусного ПО и используемых им баз данных вирусных сигнатур.

Системы обнаружения вторжений на уровне хоста

В то время как сетевые системы обнаружения вторжений осуществляют мониторинг сетевого трафика, пытаясь обнаружить аномальный трафик и подозрительную активность, *системы обнаружения вторжений на уровне хоста* (host intrusion

[1] На сегодняшний день количество вирусов настолько велико, что большинство разработчиков антивирусного ПО не в состоянии обеспечить защиту от каждого из них. Они знают об этом и полагаются на то, что и вы понимаете, что их продукт обеспечит защиту только от известных (опубликованных) вирусов.

detection system, HIDS), например такие, как OSSEC[1], наблюдают за состоянием вашего сервера с целью обнаружения чего бы то ни было необычного. Системы HIDS во многих отношениях работают по аналогии с антивирусными системами, за исключением того, что они исследуют систему на все признаки компрометации и уведомляют вас обо всех случаях изменения файлов операционной системы или системных сервисов.

ВНИМАНИЕ!

Я могу неоднозначно относиться к развертыванию в облачной среде сетевых систем обнаружения вторжений (NIDS) и антивирусных систем (AV), но я — убежденный сторонник систем обнаружения вторжений на уровне хоста (HIDS), и особенно в облачной инфраструктуре. Я считаю, что без HIDS вы не должны выполнять развертывание серверов в облачной среде.

В моих системах Linux, которые я развертываю в облачной среде, я в основном пользуюсь OSSEC (**http://www.ossec.net**). OSSEC имеет два конфигурационных профиля:

- *независимый* (standalone), при котором каждый сервер сканирует себя самостоятельно и высылает вам предупреждения;
- *централизованный* (centralized), при котором вы создаете централизованный сервер HIDS, на который каждый из остальных серверов направляет отчеты.

В облачной инфраструктуре вы всегда должны выбирать централизованную конфигурацию. Она централизует все ваши правила и выполнит анализ таким образом, чтобы серьезно упростить для вас поддержание инфраструктуры HIDS в актуальном состоянии. Более того, централизованная конфигурация позволит вам разработать профиль безопасности повышенного уровня, обеспечивающий большую степень защиты отдельных сервисов, чем независимые профили. Облачная инфраструктура, использующая централизованную систему HIDS, представлена на рис. 5.5.

Как и в случае с решением, использующим антивирусное программное обеспечение, вы должны постоянно заботиться о поддержании ваших серверов HIDS в актуальном состоянии, но при этом у вас нет необходимости в таком частом обновлении отдельных серверов, как в случае с использованием антивирусного ПО.

"Оборотной стороной медали" в случае применения HIDS является то, что такие системы очень требовательны к ресурсам CPU, и, таким образом, могут потреблять значительную часть вычислительных мощностей на вашем сервере. При использовании модели централизованного сервера вы можете передать на центральный сервер большую часть задач, выполняемых специализированным сервером обнаружения вторжений.

---

[1] OSSEC (Open Source Host-based Intrusion Detection System) — это бесплатная система обнаружения вторжений на уровне хоста, основанная на открытом коде (Open Source). Она предоставляет такие возможности, как анализ файлов журнала, проверку целостности, выявление руткитов (rootkit), превентивного оповещения и активной реакции на начавшуюся атаку. Существуют версии OSSEC для большинства популярных операционных систем, включая Linux, OpenBSD, FreeBSD, Mac OS X, Solaris и Windows. Подробнее см. **http://en.wikipedia.org/wiki/OSSEC**, **http://www.ossec.net/**, **http://tinyurl.com/2w3qq79**. — *Прим. перев.*

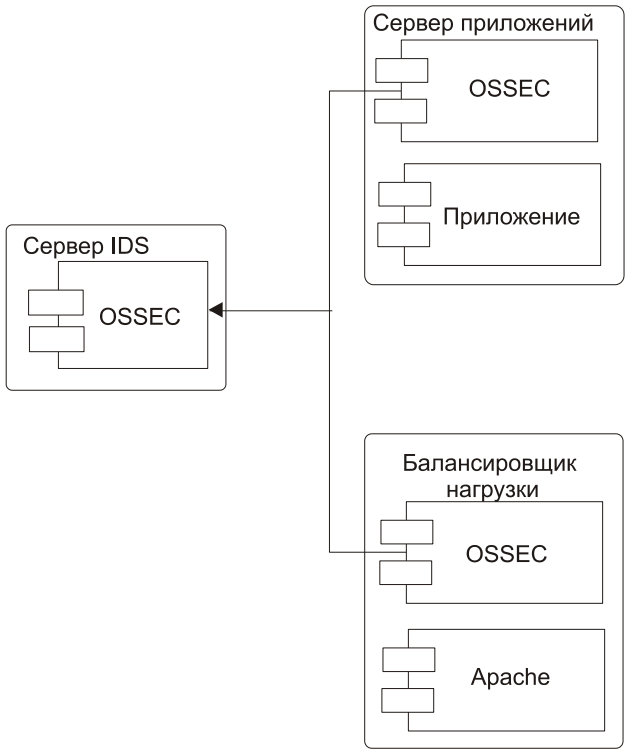


Рис. 5.5. Централизованная система HIDS в облачной инфраструктуре

Сегментация данных

В дополнение к предположению о том, что сервисы, запущенные на ваших серверах, не свободны от уязвимостей, вы должны исходить из того, что рано или поздно один из них будет скомпрометирован. Естественно, вы не хотите возникновения такой ситуации. Однако наилучшей инфраструктурой будет та, которая "терпимо" относится к компрометации любого из индивидуальных узлов и, фактически, подразумевает такую возможность. В данном случае под "терпимостью" мы не подразумеваем слабую защищенность отдельных серверов, а, напротив, минимизацию влияния такого события, как компрометация одного из узлов, на общую работоспособность всей системы. Такой подход позволит вам разработать систему, обладающую следующими преимуществами:

- доступ к данным, обладающим наивысшим уровнем секретности, требует полного взлома всей системы;
- для компрометации всей системы требуются множественные векторы атаки и радикально различные уровни квалификации;

- время простоя, ассоциированное с компрометацией отдельного узла, либо пренебрежимо мало, либо вообще равно нулю.

Сегментация данных в соответствии с различными уровнями конфиденциальности является основным средством минимизации влияния успешно проведенной атаки на работоспособность системы в целом. В *главе 4* мы изучили сегментацию данных на примере разделения данных о кредитных картах и персональной информации пользователей. В этом примере атакующий, получающий доступ к клиентской базе данных, получит часть ценной информации, но для того, чтобы воспользоваться этими сведениями, ему все равно потребуется доступ к данным о кредитных картах. Чтобы получить возможность доступа к данным обработки кредитных карт, расшифровать эту информацию и ассоциировать ее с персональными данными конкретного клиента, атакующему необходимо будет дополнительно взломать приложение, реализующее систему электронной коммерции, а также процессор кредитных карт.

В реализации такой системы вам вновь окажет помощь подход "один сервер — один сервис". Поскольку каждый из серверов в этой цепочке имеет единственный вектор атаки, отличный от векторов атаки других серверов, атакующему для достижения успеха потребуется воспользоваться множеством эксплоитов, что серьезно усложнит его задачу.

Управление учетной информацией

Профили OSSEC ваших машинных образов не должны содержать никаких встроенных пользовательских учетных записей. В реальности вы никогда не должны предоставлять возможности доступа к оболочке с применением пароля ни на одном из ваших виртуальных серверов. Наиболее безопасный подход к предоставлению доступа к вашим виртуальным серверам заключается в динамической доставке открытых ключей SSH на целевые серверы. Иными словами, если кому-то требуется доступ к серверу, то вы должны передавать соответствующую информацию серверу в момент его запуска или через административный интерфейс, а не за счет хранения учетной информации в составе образа машины.

Примечание

Лучшие поставщики систем управления учетной информацией предоставляют доступ к серверу только для тех пользователей, которым действительно необходимо по долгу службы обращаться к серверу, и только на то время, в течение которого этот доступ им необходим. Я настоятельно рекомендую вам воспользоваться преимуществом динамической природы облачной инфраструктуры для ограничения необходимости поддержания учетной информации для пользователей, которые либо имеют ограниченные потребности в доступе к серверу, либо не имеют их вообще.

Разумеется, встраивание открытых ключей SSH в образ машины абсолютно безопасно, и это серьезно упрощает жизнь. К сожалению, это усложняет построение машинных образов общего назначения, которые я описывал в *главе 4*. В частности, если вы встроите учетную информацию открытого ключа в образ

машины, пользователь, к которому относится эта информация, сможет получить доступ к каждому виртуальному серверу, построенному на основе этого образа. Чтобы запретить доступ этому пользователю или разрешить доступ другому, вам потребуется построить новый машинный образ, отражающий необходимое вам изменение.

Следовательно, вы должны поддерживать конфигурацию настолько простой и удобной в поддержке, насколько это возможно, за счет передачи учетной информации в процессе запуска вашего виртуального сервера. Во время загрузки виртуальный сервер имеет доступ ко всем параметрам, которые вы ему передаете, и он может создать учетные записи для всех указанных вами пользователей. Это просто, потому что для решения данной задачи не требуется никаких дополнительных инструментов, кроме тех, которые уже предоставлены Amazon. С другой стороны, блокировка доступа и его предоставление во время загрузки системы превращается в ручную задачу.

Еще один подход заключается в использовании существующих средств управления облачной инфраструктурой или разработке вашего собственного инструментария, который позволил бы вам хранить учетную информацию пользователей за пределами облачной инфраструктуры и динамически добавлять или удалять пользовательские учетные записи на облачные серверы во время исполнения. Однако этот подход требует запуска административного сервиса на каждом из хостов и, таким образом, представляет собой дополнительный вектор атаки на ваш сервер.

Реакция на компрометацию

Поскольку вы должны иметь систему обнаружения вторжений, вы должны очень быстро узнавать о случаях, когда имеет место фактическая компрометация системы. Если вы реагируете на такие ситуации быстро, вы можете воспользоваться преимуществом снижения времени простоя, вызванного компрометацией системы вследствие применения эксплоита.

Когда вы обнаруживаете, что скомпрометирован ваш физический сервер, процедура реагирования представляет собой долгий и болезненный процесс:

1. В первую очередь необходимо перекрыть доступ злоумышленнику, вторгшемуся в систему. Как правило, это достигается отключением сервера от всей остальной сети.
2. Затем следует определить вектор атаки. Вам требуется не просто остановить сервер и перезапустить его, поскольку уязвимость, о которой идет речь, может затрагивать любое количество серверов. Далее, вполне возможно, что атако-

вавший вас злоумышленник уже оставил в системе руткит (rootkit)[1] или какое-либо еще программное обеспечение, дающее возможность повторного вторжения после того, как вы ликвидируете изначальную уязвимость, которая и позволила ему осуществить вторжение. Поэтому крайне важно идентифицировать способ, которым атакующий сумел скомпрометировать вашу систему, а также выяснить, дало ли это ему возможность скомпрометировать и другие системы в вашей сети, а также определить, нет ли в остальных системах такой же уязвимости, которая дала злоумышленнику доступ в вашу сеть.

3. Вычистить сервер и снова запустить его. На этом шаге вам потребуется "залатать" дыру в системе безопасности и перестроить систему с помощью новейшей версии нескомпрометированной резервной копии.
4. Запустить сервер в рабочее состояние и повторить перечисленные шаги для всех серверов, имевших точно такой же вектор атаки.

Этот процесс очень трудоемок и может потребовать длительного времени. В облачной среде он протекает намного проще.

Во-первых, вы можете провести некоторые процедуры по криминалистическому анализу. Вы можете скопировать корневую файловую систему на один из блочных томов, снять моментальный снимок блочных томов, остановить сервер и осуществить его замену.

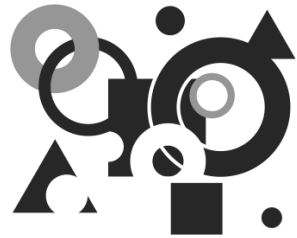
После того как предназначенный на замену сервер начнет работу (определенно, он будет иметь прежнюю уязвимость, но, по крайней мере, он не будет скомпрометирован), вы сможете запустить сервер в выделенной группе безопасности и смонтировать скомпрометированные тома. Поскольку этот новый сервер будет иметь другую корневую файловую систему, и на нем не будет работать никаких сервисов, он не будет скомпрометирован. Тем не менее, вы все же будете иметь полный доступ к скомпрометированной информации, так что вы будете иметь возможность и для идентификации вектора атаки.

После того как вы идентифицируете вектор атаки, вы сможете применить обновления безопасности, устраняющие уязвимость, к образам машин. Как только уязвимость будет устранена, вы сможете перезапустить другие экземпляры. Таким образом, в результате вы сможете гораздо быстрее реагировать на уязвимости и компрометацию серверов, сведя к минимуму (или даже к нулю) время простоя.

---

[1] Руткит (*rootkit*, т. е. "набор root'a") — программа или набор программ для скрытия следов присутствия злоумышленника или вредоносной программы в системе. Термин "rootkit" исторически пришел из мира UNIX, и под этим термином понимается набор утилит или специальный модуль ядра, которые взломщик устанавливает на взломанной им компьютерной системе сразу после получения прав суперпользователя. Подробнее см. **http://tinyurl.com/ye4jkb6**, **http://en.wikipedia.org/wiki/Rootkit**, **http://www.z-oleg.com/secur/articles/rootkit.php**, **http://www.rootkit.com/**. — *Прим. перев.*

Глава 6



Аварийное восстановление

Насколько хорош ваш план аварийного восстановления? Насколько полно и хорошо он документирован? Выполняете ли вы регулярное тестирование этого плана, осуществляя тренировочные работы по аварийному восстановлению (учения на случай аварий, катастроф, стихийных бедствий)?

До сих пор на всем протяжении обсуждения мы говорили только о рутинных, ожидаемых событиях и отказах, не представляющих серьезной угрозы. *Аварийное восстановление* (Disaster recovery) — это комплекс мер, разрабатываемых на случай непредвиденных аварий, катастроф, стихийных бедствий и призванных обеспечить "выживание" вашей системы в случае таких экстраординарных событий. План аварийного восстановления, к примеру, должен обеспечить выживание вашей инфраструктуры IT даже в таких случаях, как, например, пожар в центре обработки данных, в результате которого окажутся потеряны все ваши серверы и системы, которые на них работают.

Любое предприятие или организация должны иметь тщательно спланированный и хорошо документированный план аварийного восстановления и тестировать работоспособность этого плана не реже, чем два раза в год. В действительности же даже те компании, где существуют достаточно жесткие требования к организационной дисциплине, часто не уделяют планированию процесса аварийного восстановления должного внимания. Например, практика показала, что в только что описанном мною сценарии с пожаром в центре восстановления данных слишком многие малые и средние предприятия будут вынуждены просто закрыться.

Одним из преимуществ виртуализации, которое я особо высоко ценю, является автоматизация процесса аварийного восстановления. Восстановление после тривиальных сбоев и процесс аварийного восстановления в форс-мажорных обстоятельствах в облачной инфраструктуре почти неотличимы друг от друга. В результате этого, даже если вся ваша облачная инфраструктура будет разрушена, при правильном подходе вы сможете восстановить ее в полном объеме либо на внутренних серверах, либо в центре обработки данных провайдера управляемых услуг, либо прибегнув к услугам другого облачного провайдера. При этом сам процесс восстановления можно провести за считанные часы или даже минуты.

Планирование процесса аварийного восстановления

Аварийное восстановление планируется[1] в расчете на катастрофические события, наступление которых маловероятно в течение жизненного цикла системы. Если отказы, восстановление после которых вы планируете, являются событиями не столь уж невероятными, и наступления их можно ожидать, то процедуры восстановления подпадают под категорию традиционного планирования по восстановлению доступности системы. Хотя наступление любого конкретного катастрофического события в течение жизненного цикла системы и является маловероятным, тем не менее, вероятность того, что какое-нибудь бедствие все же произойдет, является ненулевой.

В процессе планирования аварийного восстановления вам необходимо определить так называемое *приемлемое восстанавливаемое состояние* (acceptable recovery state), до которого необходимо восстановить систему, а затем разработать пошаговые процедуры и способы восстановления этого состояния при ликвидации последствий катастрофы. Говоря о приемлемом восстанавливаемом состоянии, я имею в виду объем данных, который вы можете позволить себе потерять в случае наступления катастрофического события.

При выполнении планирования аварийного восстановления после катастрофических событий необходимо определить следующие два ключевых показателя.

- *Целевая точка восстановления* (Recovery Point Objective, RPO)[2] — этот показатель определяет, какой объем данных вы можете себе позволить потерять в случае наступления катастрофического события. Обычно это значение выражается в количестве часов или рабочих дней, в течение которых шли накопление и обработка данных. Например, если вы сочтете, что можете позволить себе потерять данные, наработанные в течение 24 часов, вам необходимо будет убедиться в том, что резервные копии, которые вы будете использовать при реализации вашего плана аварийного восстановления, не "старше" 24 часов.

---

[1] Планирование и подготовка процессов аварийного восстановления после катастрофических событий является одной из самых горячих тем в отрасли IT. Читателям, желающим углубленно ознакомиться с данной тематикой и не путаться в различиях между аварийным восстановлением (Disaster Recovery, DR), планированием непрерывности производственного процесса (Business Continuity Planning, BCP/BC) и такими вещами, как высокая доступность (High availability) и резервное копирование (Backups), можно порекомендовать для начала ознакомиться со следующими материалами:
http://en.wikipedia.org/wiki/Disaster_recovery, **http://www.continuitycentral.com/itdr.htm**, **http://www.disaster-recovery-guide.com/**, **http://www.drj.com/**, **http://msft.ineta.ru/blogs/blogEntrySearch.aspx?tags=failover**. — *Прим. перев.*

[2] Фактически представляет собой временную точку, на момент которой будет восстановлено состояние системы после приведения в действие плана аварийного восстановления. Подробнее см.: **http://en.wikipedia.org/wiki/Recovery_point_objective**, **http://www.it.ru/press_center/expert/Est_li_u_vas_plan**. — *Прим. перев.*

- *Допустимое время восстановления* (Recovery Time Objective, RTO)[1] — этот показатель определяет допустимое время простоя в случае наступления катастрофического события. Если показатель RTO составляет 24 часа, это означает, что промежуток между выходом вашей системы из строя и моментом, когда система должна будет возвращена в полнофункциональное состояние, должен составлять не более 24 часов.

В дополнение к только что упомянутым ключевым показателям, специалисты, занимающиеся разработкой плана аварийного восстановления, должны определить критерии, "запускающие" претворение в жизнь плана аварийного восстановления. В общем случае начало реализации любого плана, который допускает потерю данных, требует санкции менеджеров, возглавляющих предприятие или организацию, даже в том случае, когда реализация этого плана полностью автоматизирована, как я уже говорил в начале этой главы.

Естественно, что абсолютно все предпочтут такой сценарий восстановления работоспособности, при котором не наблюдается никакого простоя и не происходит никаких потерь данных, причем не имеет никакого значения тип произошедшей катастрофы. Однако в реальной жизни природа катастрофического события обычно вынуждает нас согласиться с некоторым уровнем потерь; что-либо другое окажется весьма дорогостоящим. Например, в случае такого бедствия в масштабах целого города, как ураган "Катрина" (Hurricane Katrina)[2], стоимость выживания IT-компании, расположенной в Новом Орлеане (80 % площади которого было затоплено), без потери данных и с нулевым временем простоя включала бы стоимость множества физических центров обработки данных, расположенных в разных географических точках и постоянно синхронизирующих информацию. Иными словами, такая компания должна была обладать возможностью иметь не менее двух дублирующих центров обработки данных, соединенных выделенным широкополосным соединением.

Обеспечение такого уровня избыточности — это очень дорогой подход. К тому же, несмотря на избыточность, такая компания неизбежно столкнулась бы с нетривиальным падением производительности. Суровая реальность для большинства компаний заключается в том, что стоимость утраты данных, наработанных за последние 24 рабочих часа, все же меньше, чем затраты на поддержание инфраструктуры, обеспечивающей нулевое время простоя без каких бы то ни было потерь данных.

Определение оптимальных для компании показателей RPO и RTO, в таком случае, представляет собой именно финансовый расчет: необходимо определить, в какой именно момент стоимость потери данных и простоя превысит стоимость поддержания стратегии резервного копирования, которая предотвратит такие потери и

---

[1] Подробнее см. **http://en.wikipedia.org/wiki/Recovery_Time_Objective**, **http://www.trinitygroup.ru/solution/infrastucture/disastrous_accident**. — *Прим. перев.*

[2] Ураган "Катрина" — самый разрушительный ураган в истории США, произошедший относительно недавно (в конце августа 2005 года). См. **http://hurricane-katrina.org/** — *Прим. перев.*

такой срок восстановления. Для различных предприятий и организаций правильный ответ на этот вопрос может существенно отличаться. Если вы являетесь топ-менеджером компании по управлению ее ресурсами IT, то вы наверняка знаете ответ на этот вопрос применительно именно к вашему бизнесу.

Наконец, важным элементом планирования катастрофоустойчивой инфраструктуры предприятия является понимание того, как именно будут развиваться события в случае катастрофы. Совершенно определенно можно сказать, что существует и вероятность наступления катастрофы такого масштаба, при котором ваша инфраструктура IT не выживет, не важно, как бы тщательно вы ни продумывали план ее восстановления и сколько бы времени вы на это ни потратили. В хорошо разработанном плане аварийного восстановления отдельным пунктом должна быть прописана возможность такого сценария, с тем, чтобы все акционеры понимали это и были готовы пойти на такой риск.

Целевая точка восстановления (RPO)

Проще всего начать планирование с определения целевой точки восстановления (RPO). Сценарий развития катастрофических событий по типу "Армагеддон" ведет к полной потере всех системных данных и сборок (binaries) всех приложений, необходимых для работы системы. Ваш показатель RPO лежит где-то в интервале между состоянием вашего приложения на момент его первого развертывания и состоянием, которое оно имело в момент наступления катастрофы. Вы можете также спланировать несколько уровней угрозы катастроф, и для каждого из них определить свой показатель RPO[1].

Практически любая программная система должна обладать способностью поддержания показателя RPO между 24 часами (для менее масштабных катастроф) до одной рабочей недели (для более масштабных катастроф), не требуя при этом абсурдных затрат. Разумеется, потеря 24 часов рабочего времени в системе обработки банковских транзакций неприемлема, не говоря уже о недельном сроке.

Обычно ваш показатель RPO зависит от того, как вы сохраняете данные и выполняете их резервное копирование.

- Инфраструктуры, в которых выполняется удаленное резервное копирование (т. е. такое копирование, при котором резервная копия хранится за пределами здания, занимаемого вашим центром обработки данных), сможет пережить потерю вашего центра обработки данных с потерей данных, наработанных за неделю. Для обеспечения живучести гораздо лучше выбирать стратегию, когда удаленное резервное копирование выполняется ежедневно.
- Инфраструктура, в которых выбрана стратегия ежедневного создания удаленной резервной копии, сможет пережить потерю вашей производственной среды с потерей данных, наработанных за один рабочий день, плюс время репликации

[1] Например, вы можете определить катастрофу первого уровня как потерю одного центра обработки данных, а катастрофу уровня 2 — как потерю двух или большего количества центров обработки данных.

транзакций за восстановительный период после полной потери системы. Если это для вас неприемлемо, лучше выполнять удаленное резервное копирование каждый час.

- Сетевые системы хранения данных (Network Attached Storage, NAS) и сети хранения данных (Storage Area Networks, SAN)[1] могут без потерь пережить потерю любого отдельного сервера, за исключением случаев, когда имеет место повреждение данных.
- Кластеризованная база данных переживет потерю любого отдельного устройства хранения данных или узла базы данных без потери данных.
- Кластеризованная база данных, распределенная по множеству центров обработки данных, переживает потерю любого отдельного центра обработки данных без потери данных.

Далее в этой главе мы поговорим о том, как облачные технологии позволяют минимизировать показатель RPO.

Допустимое время восстановления (RTO)

Даже если вы будете выполнять удаленное резервное копирование вплоть до секунд, это не поможет вам, если у вас нет среды, в которой вы сможете восстановить информацию в случае отказа. Способность создать замещающую инфраструктуру для восстановления вашей рабочей среды в случае катастрофы, включая время, необходимое на восстановление данных, управляет другим показателем — допустимым (целевым) временем восстановления (RTO).

Что произойдет, если ваш сервис-провайдер, предоставляющий услуги внешнего управления, вдруг закроется завтра и без предупреждения? Если у вас есть ряд выделенных серверов, вам потребуется время (от нескольких дней до нескольких недель) для того, чтобы ваша производственная среда снова была приведена в рабочее состояние, если только у вас нет заранее заключенного соглашения о предоставлении вам замещающей инфраструктуры[2].

Если вы имеете традиционную инфраструктуру IT, то обеспечение быстрого целевого времени восстановления окажется чрезвычайно дорогим мероприятием. Как я уже отмечал, на случай непредвиденных событий вам необходимо иметь заранее заключенное соглашение с другим провайдером услуг внешнего управления (MSP), который предоставит вам замещающую инфраструктуру, или соглашение об уровне обслуживания (SLA), предусматривающее предоставление вам замещающей инфраструктуры в случае прекращения деятельности вашего текущего

---

[1] Подробнее см. **http://en.wikipedia.org/wiki/Network-attached_storage**, **http://www.ixbt.com/storage/san.shtml**, **http://www.nas-central.org/**. — *Прим. перев.*

[2] Этот сценарий наглядно показывает, что просто выполнения резервного копирования с хранением резервных копий вне ваших рабочих площадей еще недостаточно для выживания в случае катастроф. Резервные копии должны быть неподконтрольны вашему провайдеру услуг внешнего управления, в противном случае вы рискуете лишиться их в случае банкротства вашего MSP.

MSP. В зависимости от типа заключенного соглашения, это может практически удвоить ваши расходы на инфраструктуру IT.

Облачная среда, даже при использовании виртуальных центров обработки данных, меняет и взгляды на показатель RTO. Этот вопрос мы тоже обсудим далее в этой главе.

Катастрофические события в облачной среде

В предположении того, что вы имеете неограниченные возможности и неограниченный же бюджет, я выделяю три ключевых аспекта в отношении планирования аварийного восстановления:

1. Резервное копирование и хранение данных.
2. Географическая избыточность.
3. Организационная избыточность.

Если вы позаботитесь обо всех этих аспектах, вы практически с полной уверенностью сможете обеспечить соответствие вашим потребностям в отношении целевой точки восстановления (RPO) и целевого времени восстановления (RTO). Впрочем, я сам никогда не бывал в таких ситуациях, когда бы я располагал неограниченным бюджетом и неограниченными возможностями, поэтому мне всегда приходилось искать компромисс. В результате этого для меня всегда имел значение порядок, в котором были перечислены упомянутые аспекты. В дополнение к этому, если ваш хостинг-провайдер представляет собой организацию, которая еще не успела хорошо себя зарекомендовать, то возможно, организационная избыточность может оказаться для вас даже важнее, чем географическая.

К счастью, облачная структура Amazon существенно упрощает решение первого и второго вопросов. Кроме того, сама природа облачной обработки данных существенно упрощает и решение третьей задачи.

Примечание

В этом разделе приводится большое количество информации, специфичной именно для облачной среды Amazon. Большинство облачных инфраструктур на сегодняшний день развиваются на основе концепций, аналогичных устройствам блочного хранения EC2 и моментальным снимкам (snapshots). Для таких облачных инфраструктур будут применимы и многие из концепций, обсуждающихся в данной главе. Вопросы, касающиеся географической избыточности, однако, специфичны для выбираемых вами облачных провайдеров, но то немногое, что я пишу по данному вопросу в этой главе, применимо и к другим облачным провайдерам.

Управление резервным копированием

В *главе 4* рассматривались технические детали, касавшиеся управления образами машин Amazon (AMI) и выполнения резервного копирования в облачной среде. Сейчас самое время немного отступить от технических деталей и изучить типы

данных, резервное копирование которых вам необходимо спланировать, а также рассмотреть, какова роль всех этих операций в вашем общем плане аварийного восстановления после катастрофических событий.

Ваши возможности по восстановлению после катастроф ограничиваются частотой резервного копирования и качеством получаемых вами резервных копий. В традиционной инфраструктуре IT многие организации придерживаются стратегии резервного копирования, при которой полное резервное копирование на ленточные носители выполняется раз в неделю, ежесуточно по ночам создаются дифференциальные резервные копии, после чего недельные резервные копии сохраняются на носители, хранящиеся за пределами помещений, в которых располагаются их центры обработки данных. В облачной среде вы можете выработать намного лучшую стратегию, которая к тому же обойдется дешевле. Это достигается за счет применения поуровневого подхода.

Резервное копирование, непрерывность бизнеса и Amazon Web Services

В данном разделе будет рассматриваться ряд технологий, предоставляемых Amazon Web Services (AWS) с тем, чтобы помочь вам эффективно управлять процессом резервного копирования. Если вы пользуетесь услугами другого облачного провайдера, то эта компания, вероятнее всего, предложит вам как средства, аналогичные описанным здесь, так и радикально отличающиеся. Однако концепция удаленного резервного копирования и хранения резервной копии за пределами вашего центра обработки данных играет ключевую роль в любой стратегии резервного копирования. Какой бы ни была ваша стратегия резервного копирования, вы должны выработать не только механизм выведения всех данных, имеющих критическое влияние на ваш показатель целевой точки восстановления (RPO), за пределы облачной инфраструктуры, но и разработать способ хранения этих данных в переносимом формате с тем, чтобы получить возможность их восстановления в любой среде, которая может радикально отличаться от среды, предоставляющейся вашим текущим облачным провайдером.

Различные типы данных, которыми могут управлять ваши Web-приложения, перечислены в табл. 6.1.

Таблица 6.1. *Требования к резервному копированию в зависимости от типов данных*

Тип данных	Описание
Неизменные данные (Fixed data)	Неизменные данные, такие как ваша операционная система и утилиты общего назначения, принадлежащие вашему AMI. В облачной среде вам нет необходимости выполнять резервное копирование AMI, потому что они не имеют никакой ценности за пределами облачной среды[1]

[1] Имейте в виду, что даже в случае полного выхода из строя Amazon S3, в результате которого вы теряете все ваши AMI, до тех пор, пока доступен сервис EC2 и в вашем распоряжении остаются экземпляры EC2, запущенные из утраченных AMI, вы все равно будете иметь возможность быстрой перестройки утраченных AMI. С другой стороны, в случае одновременного отказа EC2, S3 и полной потери всех данных S3 вам потребуется выполнять восстановление в другой облачной среде, которая в любом случае не сможет распознавать ваши AMI!

Таблица 6.1 *(окончание)*

Тип данных	Описание
Текущие данные (Transient data)	Файловые кэши и другие данные, которые могут быть полностью потеряны без оказания влияния на целостность вашей системы. Так как состояние вашего приложения не зависит от этих данных, вам нет необходимости осуществлять их резервное копирование
Конфигурационные данные (Configuration data)	Конфигурационные данные времени исполнения (Runtime configuration data), необходимые для обеспечения правильной работы вашей системы в специфическом контексте. Эти данные не являются текущими, поскольку они должны "переживать" такие события, как перезапуск машины. С другой стороны, эти данные должны предоставлять возможность быстрого переконфигурирования при "чистой" установке приложения. Такие данные требуют более или менее регулярного резервного копирования
Сохраняемые данные (Persistent data)	К сохраняемым данным относится состояние вашего приложения, в том числе критически важная клиентская информация (например, заказы на покупки). Эти данные постоянно изменяются, и наилучшим средством для управления ими является "движок" базы данных. Ваш "движок" базы данных должен сохранять информацию о своем состоянии на устройстве блочного хранения, и вы должны регулярно выполнять резервное копирование этих данных. Наиболее важными инструментами управления базой данных являются кластеризация и/или репликация

Сохраняемые данные играют ключевую роль при планировании аварийного восстановления после катастрофических событий. Операционную систему можно перестроить всегда, установить все необходимое программное обеспечение и переконфигурировать его тоже можно всегда, но вот возможности ручного восстановления сохраняемых данных просто не существует.

Стратегия в отношении неизменных данных

Если идея резервного копирования ваших образов машин является для вас самоценной, то вы можете скачать образы из S3 и хранить их за пределами облачной среды Amazon. В случае отказа S3, повлекшего повреждение или потерю данных, влияющих на ваши AMI, впоследствии вы сможете заказать их в облачную инфраструктуру со своих резервных копий и повторно зарегистрировать их. В принципе, это неплохая и несложная в реализации идея, но такой подход ограничивается типом сценария отказа, в результате которого вы можете прибегнуть к такого рода резервным копиям.

Стратегия в отношении конфигурационных данных

Хорошо продуманная стратегия резервного копирования для конфигурационной информации предусматривает двухуровневый подход. При этом первый уро-

вень представляет собой регулярный сброс дампа вашей файловой системы в ваше облачное хранилище или "моментальный снимок" вашей файловой системы. Для большинства приложений вы можете выполнять резервное копирование ваших данных ежедневно или раз в неделю (даже это будет уже неплохо). Но при этом вам необходимо иметь в виду ваш показатель целевой точки восстановления (Recovery Point Objective). Если ваши конфигурационные данные изменяются дважды в день, и при этом вы установили для себя значение RPO, равное 2 часам, вам потребуется выполнять резервное копирование ваших конфигурационных данных дважды в день. Если конфигурационные данные изменяются нерегулярно, возможно, вам потребуется выполнять их резервное копирование раз в час или "привязать" ваше расписание резервного копирования к изменениям в конфигурации приложения.

Альтернативный подход заключается в проверке конфигурации приложения в репозитории исходного кода, находящегося за пределами облачной среды, и применении этого репозитория для восстановления даже при минимальных потерях.

Не имеет значения, что вы предпочтете — выполнять моментальные снимки вашей файловой системы или просто держать запакованные архивы вашей файловой системы — эти данные будут находиться в режиме гибернации в недрах S3. Общая тенденция такова, что "моментальные снимки" представляют собой более эффективный и менее "назойливый" механизм выполнения резервного копирования, но зато он обеспечивает и гораздо меньший уровень переносимости. У вас нет прямого доступа к моментальным снимкам EC2, и даже если бы вы его имели, этот формат не может быть использован за пределами облачной среды Amazon. На определенном этапе вам все же понадобится вывести эти данные за пределы облачной инфраструктуры, поэтому вам понадобятся хранящиеся на стороне резервные копии в переносимом формате. Мои рекомендации сводятся к следующему.

- Регулярно (как минимум, ежедневно) создавайте моментальные снимки ваших конфигурационных данных.
- Периодически создавайте архивы вашей файловой системы (по крайней мере, периодичность должна быть меньше, чем ваш показатель RPO), упаковывайте его в форматы ZIP или TAR, и помещайте эти архивы на хранение в Amazon S3.
- Периодически (как и в предыдущем случае, этот период должен быть меньше, чем ваш показатель RPO) копируйте ваши архивы файловой системы за пределы облака Amazon в другую облачную или физическую среду.

Предположим, что у меня имеется приложение, для которого RPO составляет один день (24 часа), объем данных, которыми управляет это приложение, составляет менее 10 Гбайт, а конфигурационные данные этого приложения могут измениться в любой момент. Для такой ситуации я выбрал бы следующую стратегию.

- Создавать моментальные снимки файловой системы раз в час, а всю файловую систему архивировать в переносимом формате и, как минимум, один раз в день копировать в Amazon S3.
- Я бы хранил недельный архив полных резервных копий в S3 на случай возникновения вопросов с повреждениями данных. В дополнение к этому я бы копировал каждую недельную резервную копию из S3 в другую облачную среду или

на собственные физические файл-серверы внутренней инфраструктуры IT в заранее оговоренный момент времени сразу же после создания резервных копий.

- Наконец, я бы хранил набор резервных копий за последнюю неделю за пределами рабочей площадки вместе с недельной резервной копией за предшествующий месяц и одной месячной резервной копией за прошедший год. В зависимости от объема данных, о которых идет речь, и требований к хранению данных я и рекомендовал бы выстраивать стратегию по архивации старых резервных копий.

Различные требования к показателю RPO и поведению приложений диктуют выбор различных стратегий. Подчеркнем еще раз: вам необходимо очень четко оценить ваш показатель целевой точки восстановления (RPO), потому что от него зависит выбор подходящей для вас стратегии резервного копирования. Если взять только что приведенный пример, то описанная в нем стратегия позволит создать рабочий план по развертыванию среды приложения в любом центре обработки данных в любой географической точке с потерей данных, не превышающей наработки за последние 24 часа, за исключением ситуации полного повреждения данных на всех резервных копиях и одновременной полной потери всех хранилищ (как Amazon, так и всех резервных копий, хранящихся за пределами рабочей площадки).

Стратегия в отношении сохраняемых данных (резервное копирование баз данных)

Для хранения клиентской информации и других сохраняемых данных я настоятельно рекомендую применять реляционные базы данных. Реляционные базы данных в любом случае нацелены на поддержание целостности данных по обработке транзакций. Сложность в организации резервного копирования баз данных заключается в том, чтобы эти операции производились на регулярной основе и при этом не оказывали влияние на общую работоспособность системы при одновременном поддержании целостности баз данных.

MySQL, как и любой другой движок базы данных, обеспечивает ряд удобных инструментов для осуществления резервного копирования. Однако вы должны использовать их продуманно, чтобы избежать случайного повреждения данных. Методы резервного копирования хорошо документированы в многочисленных книгах по работе с базами данных, но при этом они настолько важны, что в данной книге я приведу их общий обзор и опишу работу с ними в среде Amazon EC2.

Если вы не слишком хорошо знакомы с принципами работы баз данных, возможно, у вас возникнет вопрос: "Что в этом такого уж сложного? Почему при работе с базами данных я не могу снять моментальный снимок или архивировать их, точно так же, как и в случае с конфигурационными данными?" Дело в том, что при работе с конфигурационными данными крайне маловероятно, что вы будете выполнять резервное копирование в процессе записи двух различных файлов, которые должны быть взаимно непротиворечивыми, или в процессе записи файла в файловую систему. Напротив, при работе с базами данных вероятность того, что

вы попытаетесь выполнить копирование файлов, движок базы данных будет выполнять над ними какую-либо транзакцию. Поэтому разработка стратегии резервного копирования баз данных требует большей тщательности.

Первой "линией обороны", позволяющей избежать нарушения целостности данных, является применение или *репликации с несколькими ведущими узлами* (multimaster replication), или *кластеризации* (clustering). База данных с несколькими ведущими узлами представляет собой базу данных, в которой два ведущих сервера баз данных независимо друг от друга ведут запись транзакций, а затем реплицируют их на другой ведущий сервер. В кластерной среде обработки баз данных имеется множество серверов, которые действуют как единый логический сервер. При использовании обоих сценариев система остается работоспособной и внутренне непротиворечивой.

Кроме того, вы можете выполнять *репликацию по сценарию "главный—подчиненный"* (master-slave replication). Репликация по сценарию "главный—подчиненный" подразумевает настройку главного сервера, который управляет операциями записи и затем тиражирует транзакции на подчиненный сервер (slave). Каждый раз, когда на главном сервере что-то происходит, он реплицирует изменение на подчиненный сервер.

Репликация сама по себе не является "первой линией обороны", потому что репликация не обладает атомарностью в отношении транзакций, в отличие от записи транзакций на главный сервер. Иначе говоря, возможна такая ситуация, когда главный сервер откажет после подтверждения транзакции, но до того, как он успеет реплицировать ее на подчиненный сервер. Чтобы обойти эту проблему, я обычно поступаю так:

- устанавливаю и настраиваю главный сервер так, чтобы его данные хранились на устройстве блочного хранения;
- устанавливаю и настраиваю подчиненный сервер так, чтобы его файлы данных располагались на устройстве блочного хранения;
- регулярно создаю моментальные снимки устройства блочного хранения главного сервера, основываясь на выбранном значении показателя целевой точки восстановления (RPO);
- регулярно создаю дампы базы данных на подчиненном сервере и сохраняю их в S3;
- время от времени копирую дампы базы данных из S3 в хранилище, расположенное за пределами облака Amazon.

Примечание

При сбрасывании дампа состояния вашей базы данных MySQL на автономный носитель (offline media) я настоятельно рекомендую использовать для этой цели подчиненный сервер, который не поддерживает запросы только на чтение от приложения. За счет использования подчиненного сервера вы минимизируете влияние резервного копирования на производительность производственной системы.

Эластичное блочное устройство Amazon (Elastic Block Storage, EBS) предлагается еще недостаточно долго для того, чтобы иметь полную уверенность в том, что

вы сможете обнаружить повреждение базы данных в случае отказа сервера MySQL. Я исхожу из предположения, что повреждения имеют высокую вероятность, как и в случае с любой файловой системой, и соответствующим образом выстраиваю свою стратегию резервного копирования. В этом отношении я могу быть и неправ, но на данный момент я предпочитаю соблюдать осторожность.

В действительности создание моментальных снимков или сброс дампов базы данных для некоторых движков в среде времени выполнения довольно рискованно, особенно если вы хотите делать это раз в час или еще чаще. Сложность заключается в том, что на то время, пока выполняется резервное копирование, вы должны остановить обработку всех транзакций. Еще более осложняет ситуацию тот факт, что для завершения сброса дампа базы данных может потребоваться довольно длительное время. В результате работа всех ваших приложений застопорится до тех пор, пока не завершится сброс базы данных.

Моментальные снимки можно создавать в большинстве облачных сред, и они представляют собой важный метод поддержания целостности базы данных без необходимости полной приостановки работы всех приложений — даже при обработке больших объемов данных, как в MySQL.

"Заморозить" базу данных нужно только на момент создания моментального снимка. Пошаговая процедура выглядит следующим образом:

1. Заблокируйте базу данных.
2. Синхронизируйте файловую систему (эта процедура зависит от файловой системы).
3. Создайте моментальный снимок.
4. Разблокируйте базу данных.

Весь процесс должен занимать около секунды.

В Amazon EC2 вы будете хранить ваши моментальные снимки непосредственно на блочном устройстве. К сожалению, моментальные снимки не являются переносимыми, так что вы не сможете использовать их за пределами облака Amazon. Поэтому вам необходимы дампы базы данных, как бы вам ни хотелось избежать этой процедуры. Поэтому я предпочитаю выполнять эту процедуру на подчиненном сервере базы данных. Подчиненный сервер можно заблокировать на тот период времени, который необходим для завершения процесса сброса дампа базы данных. Для целей резервного копирования не имеет большого значения тот факт, что состояние подчиненного сервера базы данных может на одну-две транзакции отставать от главного сервера. Значение имеет то, что вы можете создать дамп, не оказывая существенного влияния на ваши приложения.

Примечание

Если базы данных очень велики, сброс дампов может оказаться настолько длительным, что перестанет быть разумным и оправданным. Альтернативный подход заключается в резервном копировании дампов баз данных вместе с файлами журналов баз данных таким образом, чтобы и они могли быть использованы для восстановления данных после сбоя или ошибочного изменения. Как вариант, при работе с MySQL можно просто выполнить резервное копирование файлов баз данных. Оба этих подхода несколько сложнее, чем обычный сброс дампа базы данных.

Для создания дампа базы данных необходимо проделать следующие шаги:

1. Осуществите сброс дампа базы данных.
2. После завершения сброса дампа зашифруйте дамп и разбейте его на небольшие управляемые "цепочки".
3. Переместите дамп в S3.

Amazon S3 имеет ограничение по размеру файлов, который не должен превышать 5 Гбайт. В результате этого большие базы данных необходимо разбивать на "цепочки", а в целях обеспечения безопасности вы обязательно должны шифровать дамп, как и все, что вы переносите в Amazon S3.

Теперь, когда у вас есть переносимая резервная копия вашего сервера базы данных, вы можете скопировать ее за пределы облака Amazon и тем самым защитить данные на случай потери резервных копий, хранящихся в S3.

Безопасность резервного копирования

Если вы следуете рекомендациям по обеспечению безопасности, которые обсуждались ранее в этой книге, то ваши файловые системы должны быть зашифрованы с тем, чтобы защитить создаваемые вами моментальные снимки от посторонних глаз. Несколько более сложной задачей является защита ваших переносимых резервных копий в ходе того, как вы сохраняете их в S3 и копируете их за пределы вашей рабочей среды.

Обычно для защиты моих переносимых резервных копий я использую одну из разновидностей алгоритма шифрования, совместимого с PGP. Особого внимания заслуживают следующие два аспекта.

- Хранение вашего закрытого ключа расшифровки за пределами облачной среды.
- Хранение вашего закрытого ключа расшифровки там, где он никогда и ни при каких обстоятельствах не будет потерян.

Процедуры подготовки к аварийному восстановлению, которые я описал в этой главе, не требуют, чтобы вы передавали ваши закрытые ключи экземпляру, работающему в облачной среде, за исключением тех случаев, когда вы хотите автоматизировать процесс переключения при отказе на другую облачную инфраструктуру. Облачная среда требует только ваш открытый ключ шифрования, чтобы иметь возможность зашифровать ваши переносимые резервные копии.

Нельзя хранить ваш ключ расшифровки в составе ваших резервных копий. Во-первых, это обессмыслит всю процедуру шифрования. Поскольку вы будете хранить ваш ключ расшифровки отдельно от ваших резервных копий, вы рискуете потерять его. С другой стороны, хранение пакета копий ваших ключей расшифровки повышает вероятность того, что он может попасть не в те руки.

Каким же должен быть наилучший подход? Сохраните две копии:

- одна из копий должна храниться на надежно защищенном сервере в вашей внутренней сети;
- вторую копию следует распечатать на бумаге и хранить ее в сейфе надежного банка, расположенного достаточно далеко от здания, где находится ваш защищенный сервер.

Знать о том, где хранятся копии вашего ключа расшифровки, должны несколько человек. Ведь настоящая катастрофа, к сожалению, неизбежно приводит к потерям, в том числе и среди персонала. Поэтому взаимозаменяемость сотрудников тоже является важным аспектом плана аварийного восстановления.

Если вы автоматизируете восстановление с переносимых резервных копий, вам тоже необходимо иметь копию секретного ключа на защищенном сервере, который будет управлять процессом аварийного восстановления.

Географическая избыточность

Все, что до сих пор обсуждалось в данной главе, относилось к целевой точке восстановления (RPO). В сущности, это самая простая часть аварийного восстановления. Технологии виртуализации, на которых построена облачная инфраструктура, существенно упрощают автоматизацию этих процессов и предоставляют относительно недорогой механизм создания резервных копий за пределами производственной среды.

Теперь перейдем к обсуждению целевого времени восстановления (Recovery Time Objective, RTO), показателя, для обеспечения которого ключевую роль играет избыточность инфраструктуры. Если вы можете обеспечить географическую избыточность, то вы сможете пережить почти любую катастрофу, которая может случиться. Для физической инфраструктуры обеспечение географической избыточности обходится очень дорого. В отличие от этой ситуации, в облачной среде эта мера относительно дешева.

Вам нет необходимости обеспечивать активную работу вашего приложения во всех географических точках, но вы должны иметь возможность быстро развернуть ваше приложение с резервной копии, соответствующей вашим требованиям к показателю целевой точки восстановления (RPO) в дополнительном подразделении, соблюдая временные рамки, устанавливаемые вашим показателем целевого времени восстановления (Recovery Time Objective, RTO). Например, представьте себе, что ваш показатель RTO составляет 2 часа, а RPO — 24 часа. В этом случае географическая избыточность подразумевает, что у вас имеется замещающий "филиал", в котором вы через 2 часа восстановите свою рабочую среду с резервной копии, "возраст" которой составляет не более 24 часов. При этом вы потеряете только те данные, которые были наработаны за прошедшие с момента катастрофы 24 часа.

Amazon предоставляет встроенную географическую избыточность за счет регионов (regions) и зон доступности (availability zones). Если ваши экземпляры работают в одной зоне доступности, вы сможете без особых усилий заново запустить их в другой зоне доступности того же самого региона. Если вы обязаны соблюдать специфические требования, касающиеся географической избыточности[1], то зоны доступности, предоставляемые Amazon, могут оказаться недостаточной мерой. В этом случае вам потребуется пересекать границы регионов.

[1] Некоторые правительственные агентства и другие специфические организации требуют, чтобы замещающие инфраструктуры отстояли не менее чем на 50 миль от основной.

Пересечение границ зон доступности

Почти все компоненты, образующие инфраструктуру Amazon, за исключением блочных устройств, могут располагаться в разных зонах доступности в пределах заданного региона. Хотя сетевой трафик, пересекающий границы зон доступности, и представляет собой платную услугу, цена, по которой она предоставляется, обычно стоит того, чтобы ее заплатить за возможность создания географической избыточности в нескольких зонах доступности.

На рис. 6.1 представлена производственная среда, которая с легкостью переживет потерю целой зоны доступности.

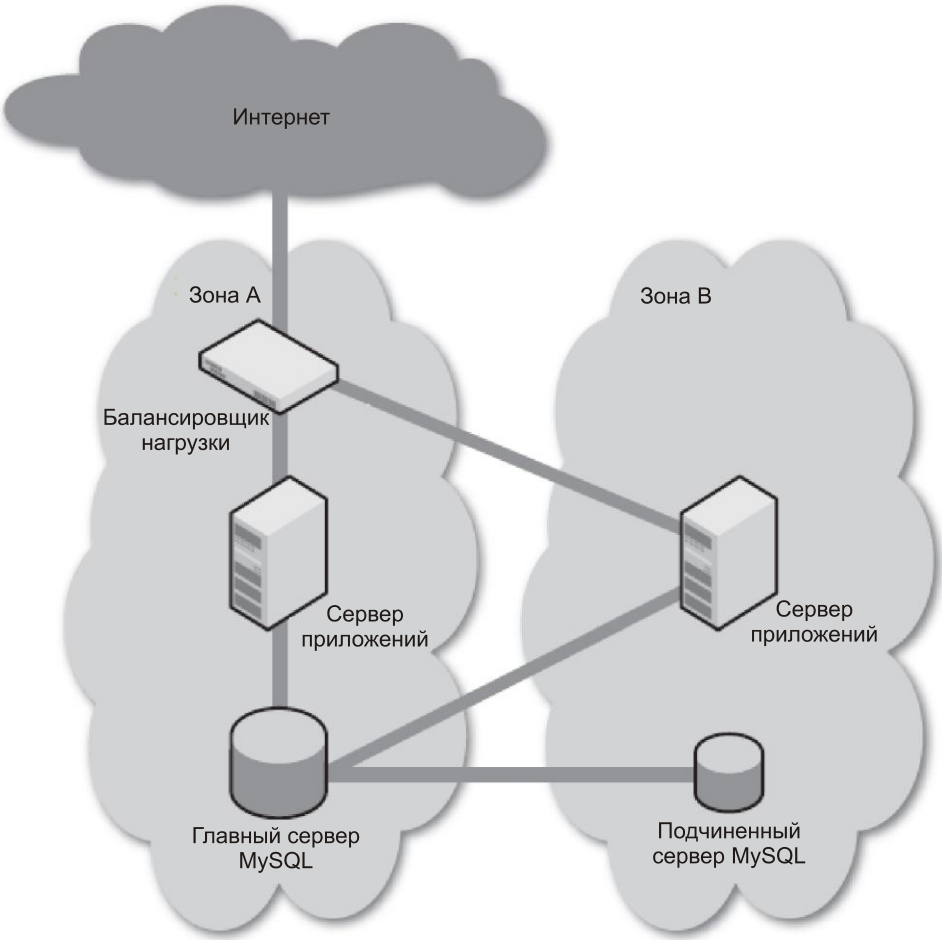


Рис. 6.1. Географическую избыточность в облачной среде можно обеспечить за счет пересечения границ зон доступности

Если вы потеряете всю зону доступности B, то с вашей операционной средой не произойдет вообще ничего. Приложение продолжит работу, как ни в чем не бывало, хотя, возможно, будет наблюдаться некоторое падение производительности.

Если вы потеряете целиком всю зону доступности A, вам потребуется запустить в зоне доступности B новый балансировщик нагрузки и выдвинуть подчиненный сервер базы данных в зоне B на роль главного. Система будет возвращена в рабочее состояние в течение нескольких минут, причем либо с минимальными потерями данных, либо вообще без каких бы то ни было потерь. Если вы используете кластеризованный сервер базы данных, а в зоне B имеется дополнительный балансировщик нагрузки, работающий в фоновом режиме, вы сможете обойтись простым переназначением IP-адреса старого балансировщика нагрузки дополнительному балансировщику. В этом случае вы сможете избежать потери данных, а время простоя составит несколько секунд.

Соглашение об уровне обслуживания Amazon (Amazon SLA) гарантирует доступность на уровне 99,95 % как минимум для двух зон доступности в пределах каждого региона. Если вы можете себе позволить пересекать границы нескольких зон доступности, вы можете даже превысить этот уровень для тех регионов, в которых существует более двух зон доступности. Например, на восточном побережье США есть три зоны доступности[1]. В результате вероятность полного отказа при использовании двух зон доступности составляет только 33 %.

Даже если вам настолько не повезет, что ваша инфраструктура окажется расположенной в двух зонах доступности, и обе они откажут, вы, тем не менее, сможете продолжить работу, если в том регионе, где вы работаете, существует более двух зон доступности. Путь решения проблемы заключается в исполнении вашего плана аварийного восстановления и переносе вашей инфраструктуры в зону доступности, сохранившую работоспособность. Таким образом, вы сможете возобновить работу, даже если ваши исходные зоны доступности вышли из строя.

Пересечение границ регионов

На тот момент, когда я начинал написание этой главы, компания Amazon поддерживала два региона[2]: us-east-1 (восточное побережье США) и eu-west-1 (Западная Европа). Эти регионы не разделяют никакой осмысленной инфраструктуры. Преимущество этой структуры заключается в том, что если вы можете себе позволить пересекать границы регионов, то ваше приложение фактически может пережить даже ядерную атаку на США, Евросоюз или Азиатско-Тихоокеанский регион (но не все регионы!). С другой стороны, отсутствие общей инфраструктуры усложняет задачу репликации вашей производственной среды через границы регионов.

---

[1] В Европейском регионе имеются две зоны доступности. Как результат, в этом регионе вам гарантируется доступность на уровне 99,95 %.

[2] На момент подготовки русского издания Amazon поддерживает уже четыре региона: US East (Северная Вирджиния), US West (Северная Калифорния), EU (Ирландия) и Asia Pacific (Сингапур), см. **http://aws.amazon.com/ec2/**. — *Прим. перев.*

В каждом регионе EC2 имеется ассоциированный регион Amazon S3. При этом вы не можете запускать экземпляры EC2 в ЕС, используя AMI из США, и не можете использовать в США IP-адреса, которые ранее были ассоциированы с балансировщиком нагрузки из Евросоюза.

Примечание

На момент написания этой книги сама идея поддержания множества регионов была для AWS новшеством. Вполне возможно, что некоторые из перечисленных мною здесь ограничений уже утратят свою силу к тому моменту, когда вы будете читать эту книгу. Далее, по мере того, как клиенты набирают опыт работы с пересечением границ регионов, они, скорее всего, и сами выработают свои приемы и рекомендации, которые в данном разделе не рассматриваются.

То, каким образом вы будете управлять вашей инфраструктурой при пересечении границ регионов, будет зависеть в основном от природы вашего Web-приложения и от ваших потребностей в избыточности. Вполне возможно, что значение для вас будет иметь только возможность быстрого запуска в другом регионе, без разработки разветвленной инфраструктуры, компоненты которой одновременно работают в нескольких регионах.

Если вам требуется именно одновременная работа экземпляров в нескольких регионах, то для этого вам необходимо рассмотреть следующие вопросы.

- Управление DNS. Чтобы обойти тот факт, что перенос IP-адресов через границы регионов невозможен, вы можете использовать круговую систему DNS (round-robin DNS)[1], но в конечном итоге вы придете к тому, что будете посылать посетителей из Европы в США и, соответственно, наоборот (крайне неэффективный подход к управлению сетевым трафиком) и потеряете половину вашего трафика, когда окажется потерянным один из регионов. Лучше применить динамическую систему DNS, например такую, как UltraDNS (**http://www.ultradns.com/**), которая предложит вам правильное разрешение имен DNS на основании источника и доступности.

[1] Круговая система DNS (Round-robin DNS) — это один из методов распределения нагрузки и обеспечения отказоустойчивости за счет избыточности количества серверов с помощью управления ответами DNS-сервера. Обычно применяется к Web-серверам, FTP-серверам и т. п. В простейшем случае круговая система DNS (Round robin DNS) работает, отвечая на запросы не одним IP-адресом, а списком из нескольких адресов серверов, предоставляющих идентичный сервис. Порядок, в котором возвращаются IP-адреса из списка, основан на алгоритме кругового обслуживания (round-robin algorithm). С каждым ответом последовательность IP-адресов меняется. Как правило, простые клиенты пытаются устанавливать соединения с первым адресом из списка, таким образом разным клиентам будут выданы адреса разных серверов, что распределит общую нагрузку между серверами. Например, на DNS-сервере создается несколько записей типа Host(A) с одинаковым именем (например, terminal.domain.local) и разными IP-адресами. Устанавливая соединение, клиенты обращаются к terminal.domain.local, а DNS-сервер на первый запрос выдает IP-адрес первого сервера, на второй — второго и так далее по кругу.
Подробнее см. **https://secure.wikimedia.org/wikipedia/en/wiki/Round_robin_DNS**, **http://www.wisegeek.com/what-is-round-robin-dns.htm**. — *Прим. перев.*

- Управление базами данных. Кластеризация баз данных при пересечении границ регионов, вероятно, окажется непрактичным решением (хотя попробовать его можно). Кроме того, вы можете установить главный сервер в одном регионе, а подчиненный — в другом. После этого операции записи будут выполняться на главный сервер, а операции чтения для трафика, исходящего из одного и того же региона с подчиненным сервером, — с подчиненного сервера. Еще один вариант заключается в сегментировании вашей базы данных таким образом, чтобы, например, в европейском регионе хранились "Европейские данные", а в американских — "Американские данные". Кроме того, каждый из регионов также имеет подчиненный сервер в другом регионе, который будет использован в качестве "точки восстановления" в случае полной потери одного из регионов.
- Вопросы нормативно-правового регулирования. Законодательство Евросоюза не разрешает хранить данные определенных типов за пределами Евросоюза. В результате этого законодательные ограничения могут помешать вам пересекать границы регионов, какие бы грамотные технические решения вы бы ни предлагали. В реальности подход к избыточности за счет использования комбинации облачных провайдеров например Amazon+GoGrid, или Amazon+Rackspace, может оказаться более эффективным, чем использование услуг только Amazon с пересечением границ регионов.

Для большинства целей я рекомендую регулярно копировать инфраструктурные элементы (AMI, резервные копии и конфигурационные данные) через границы регионов с тем, чтобы получить возможность быстрого развертывания вашей инфраструктуры даже в случае полного отказа вашей основной зоны и ее длительного простоя.

Организационная избыточность

Если у вас имеется инфраструктура, при построении которой соблюдены все рекомендации, данные в этой главе, то вы очень хорошо защищены от любого физического ущерба, который вы можете понести. Но, тем не менее, на уровне бизнес-процессов определенный риск все же существует. В частности, если облачный провайдер, услугами которого вы пользуетесь, будь то Amazon, Rackspace, GoGrid или какая-нибудь другая компания, вдруг решит свернуть бизнес или переориентироваться с облачной обработки данных на что-либо другое, у вас могут возникнуть серьезные неприятности.

Стихийные бедствия или другие катастрофические события, влекущие за собой физический ущерб, все же происходят относительно редко, зато экономические проблемы в наше время возникают часто, и практически по всему миру предприятия бывают вынуждены сворачивать бизнес по экономическим причинам — даже такие крупные, как Amazon или Rackspace. Даже если компания банкротится или в ней происходит процесс реструктуризации, никогда нельзя точно предсказать, что произойдет с ее аппаратными фондами, на которых развернуты их облачные ин-

фраструктуры. Поэтому ваш план аварийного восстановления должен включать в свой состав меры на такие непредвиденные ситуации и форс-мажорные обстоятельства, которые предусматривают полное исчезновение вашего облачного провайдера с лица Земли.

Возможно, вы и не будете поддерживать параллельно работающие инфраструктуры в различных облачных средах (за исключением тех ситуаций, когда это предоставляет определенные преимущества в отношении, скажем, географического положения). Однако даже в этом случае ваши среды вряд ли будут настолько избыточными, как сегментированные, для географических регионов, которые вы обслуживаете. Напротив, наилучшим подходом к организационной избыточности будет выбор еще одного облачного провайдера и настройка с его помощью среды резервного копирования, на случай, если что-то произойдет с вашим первым провайдером.

Внимание!

При выборе альтернативного облачного провайдера для организации среды резервного копирования вам необходимо в первую очередь убедиться в том, что этот второй провайдер никак не полагается на вашего первого и основного провайдера и ни в чем от него не зависит. Например, если ваш второй облачный провайдер использует центр данных вашего первого провайдера и пользуется его физической инфраструктурой для хостинга своей облачной среды, то в таком случае организационно вы никак не будете защищены от ситуаций, когда ваш первый провайдер будет вынужден приостановить обслуживание или просто прекратит свое существование.

Вопросы, ассоциированные с организационной избыточностью, аналогичны вопросам, обсуждавшимся ранее в этой главе при рассмотрении работы с пересечением границ регионов Amazon EC2. В частности, вы должны учитывать следующие факторы.

- Хранение переносимых (portable) резервных копий на хостинге другого облачного провайдера.
- Создание машинных образов, которые могут запускать ваши приложения в виртуализованной среде другого облачного провайдера.
- Поддержание машинных образов в актуальном состоянии с учетом их аналогов в среде вашего основного провайдера.
- Не все облачные провайдеры и провайдеры управляемых услуг (MSP) поддерживают одни и те же операционные системы и файловые системы. Если ваши приложения зависят от операционных систем или файловых систем, то при выборе альтернативного облачного провайдера необходимо убедиться в том, что он сможет удовлетворить ваши потребности.

Управление нештатными ситуациями

Итак, вы разработали стратегию резервного копирования и выполняете его, а также создали резервную инфраструктуру с необходимыми уровнями избыточности

на случай непредвиденных обстоятельств. Теперь для завершения планирования аварийного восстановления в форс-мажорных обстоятельствах вам нужны средства, позволяющие быстро определить, что наступила нештатная ситуация, и оперативно отреагировать на нее. Для этого вам нужны инструментарий и алгоритмы действий по проведению в жизнь вашего плана аварийного восстановления. Одной из наиболее привлекательных особенностей облачной среды является то, что все эти процессы можно автоматизировать. Вы можете создать такую среду, которая автоматически обнаружит признаки бедствия и самовосстановится в случае потери центров обработки данных Amazon на территории США, без всякого вмешательства с вашей стороны (фактически вы можете даже просто проспать это событие)[1].

Мониторинг

Особо важную роль играет мониторинг вашей облачной инфраструктуры. Вы не сможете выполнить замену отказавшего сервера или выполнить план аварийного восстановления, если вы не будете знать о произошедшем отказе. Однако в данной ситуации "фокус" заключается в том, что ваши системы мониторинга не могут находиться в инфраструктуре ни вашего основного, ни вашего резервного облачных провайдеров. Они должны быть независимыми от вашей облачной среды. Если вы хотите получить возможность автоматического аварийного восстановления, то ваши системы мониторинга должны иметь возможность управлять вашей инфраструктурой EC2 с сайта, выделенного для мониторинга.

Примечание

Существует ряд инструментов, таких как enStratus и RightScale, которые могут осуществлять для вас мониторинг вашей инфраструктуры. Некоторые из них даже способны автоматизировать для вас процесс аварийного восстановления таким образом, что вам не потребуется писать никакого индивидуального кода.

Основной целью мониторинга является получение возможности определять, что какой-то из компонентов может отказать еще до того, как это действительно произойдет. Наиболее общей проблемой, которую я чаще всего встречал в EC2, является постепенное снижение локального ввода/вывода на серверах до тех пор, пока их использование не становится полностью невозможным. Эта проблема представляет собой довольно простой случай, за ее развитием можно наблюдать, и ее можно исправить еще до того, как пользователи ее хотя бы заметят. С другой стороны, если вы будете ждать до тех пор, пока ваше приложение не откажет, весьма вероятно, что пользователям придется в течение некоторого времени мириться с низкой производительностью. Кроме того, такая ситуация может служить предвестником более серьезного отказа в облачной среде.

---

[1] Впрочем, возможно, идея автоматического запуска процессов аварийного восстановления без всякого вмешательства со стороны человека — это и не слишком хорошая идея, потому что потеря данных может произойти в течение запуска этих процессов.

Есть еще множество обыденных и рутинных событий, за наступлением которых надо следить в типичной облачной среде. В частности, вам нужно постоянно следить за вопросами вычислительных мощностей, включая такие ресурсы, как используемое дисковое пространство, ресурсы RAM и CPU. Более подробно эти вопросы будут обсуждаться в *главе 7*.

В конечном итоге вам необходимо наблюдать за признаками отказов на трех уровнях:

- через предоставляемый вам API (в случае с Amazon это будет API Web-сервисов EC2);
- с помощью ваших собственных средств мониторинга за состоянием запущенных вами экземпляров;
- с помощью средств мониторинга состояния, встроенных в ваше приложение.

API, предоставляемый вашим облачным провайдером, будет сообщать вам о статусе ваших экземпляров, любых монтируемых вами томов и центров обработки данных, в которых они работают. Когда вы обнаружите отказ на этом уровне, это, вероятно, будет говорить о том, что что-то случилось с самой облачной инфраструктурой. Прежде, чем начинать процедуры аварийного восстановления, вам потребуется определить, затрагивает ли сбой только один из серверов, или же в этот процесс вовлечено неопределенное количество серверов, что затрагивает всю зону доступности или даже целый регион.

Мониторинг не просто следит за наступлением бедственных событий, в основном он следит за обыденными событиями. При помощи enStratus я помещаю на каждый сервер сервис Python, который наблюдает за различными показателями "здоровья" сервера (server health indicators) — в основном относящимися к управлению мощностями. Этот сервис уведомляет систему мониторинга о возникновении проблем с сервером или его конфигурацией и позволяет системе мониторинга предпринять соответствующие меры. Кроме того, этот сервис проверяет и показатели, относящиеся к приложениям, работающим на экземпляре.

Восстановление балансировщика нагрузки

Одной из причин, по которым компании часто платят абсурдные суммы денег за физические балансировщики нагрузки, является существенное снижение вероятности отказа балансировщика нагрузки. Благодаря облачным провайдерам, таким как GoGrid и в будущем — Amazon, вы можете оценить преимущества аппаратных балансировщиков нагрузки без необходимости нести такие серьезные расходы. В текущем состоянии AWS вы должны использовать менее надежные экземпляры EC2. Восстановление балансировщика нагрузки в облачной среде, тем не менее, происходит молниеносно. Поэтому влияние сбоя балансировщика нагрузки в облачной среде сведено к минимуму.

Восстановление балансировщика нагрузки в облачной среде — это всего лишь вопрос запуска нового экземпляра балансировщика нагрузки с AMI и извещения его об IP-адресах обслуживаемых им серверов приложений. Далее вы можете дополнительно снизить время простоя за счет поддержания работающего балансировщика

ровщика нагрузки в альтернативной зоне доступности и переназначении вашего статического IP-адреса при отказе основного балансировщика нагрузки.

Восстановление сервера приложений

Если вы имеете множество серверов приложений, работающих в нескольких зонах доступности, ваша система в целом сможет пережить отказ любого из экземпляров или даже целой зоны доступности. Впрочем, вы все равно должны будете восстановить отказавший сервер, чтобы последующие отказы не сказались на вашей инфраструктуре.

Восстановление отказавшего сервера приложений представляет собой процедуру, лишь ненамного более сложную, чем восстановление отказавшего балансировщика нагрузки. Как и в случае с отказавшим балансировщиком нагрузки, вам необходимо запустить новый экземпляр с образа машины, использующегося для запуска сервера приложений. Затем запущенному экземпляру необходимо передать конфигурационную информацию, в том числе — сведения о местоположении базы данных. Когда сервер окажется в работоспособном состоянии, вам потребуется уведомить балансировщик нагрузки о появлении нового сервера (а также деактивировать информацию о старом) так, чтобы новый сервер включился в процесс ротации по балансировке нагрузки.

Восстановление базы данных

Восстановление базы данных — это наиболее сложная часть процесса аварийного восстановления в облачной среде. Ваш алгоритм аварийного восстановления должен определить, где имеется неповрежденная копия базы данных. Этот процесс может потребовать выдвижения подчиненных серверов на роль главных, перестройки управления процессом резервного копирования и переконфигурирования серверов приложений.

Наилучшим решением будет кластеризованная база данных, которая может пережить потерю любого из серверов баз данных без необходимости выполнения сложной процедуры восстановления. Если вы не можете позволить себе кластеризацию, то наилучшим планом восстановления будет тот, при котором запускается новый экземпляр сервера базы данных, к нему монтируется функциональный том EC2, который ранее использовался отказавшим экземпляром. Впрочем, когда экземпляр отказывает, может возникнуть целый ряд проблем, которые могут повлиять на эту стратегию.

- База данных может оказаться непоправимо поврежденной по той же причине, которая вызвала и сам отказ экземпляра.
- Том может оказаться поврежденным точно так же, как и экземпляр.
- Зона доступности, в которой работал экземпляр, а следовательно, и этот том, могут оказаться недоступными.

- Может случиться так, что вы не сможете запускать новые экземпляры в зоне доступности тома.

На первый взгляд может показаться, что вероятность такого развития событий мала. Но, тем не менее, такое случается. В результате этого вам необходимо разработать резервный план для процедур аварийного восстановления. Как правило, следующие меры позволяют решить проблемы с отказом базы данных на всех уровнях:

1. Запустите замещающий экземпляр в зоне доступности, где работал старый экземпляр, и попробуйте примонтировать старый том.
2. Если попытка запуска окажется неудачной, но том все еще доступен, создайте моментальный снимок этого тома и запустите новый экземпляр в любой зоне доступности, а затем создайте в этой зоне новый том на основе моментального снимка.
3. Если том, использовавшийся на шаге 1, или моментальный снимок, полученный на шаге 2, окажутся поврежденными, вам потребуется выдвинуть подчиненный сервер на роль главного.
4. Если подчиненный сервер базы данных не работает или окажется поврежденным, то следующим шагом должна стать попытка запустить замещающий том на основе последнего моментального снимка базы данных.
5. Если моментальный снимок окажется поврежденным, пробуйте запускать более ранние моментальные снимки до тех пор, пока не обнаружите неповрежденную резервную копию.

Как правило, шаг 4 представляет собой самый худший сценарий. Если вам требуется прибегать к шагу 5, это говорит о том, что в вашей стратегии резервного копирования есть что-то неправильное.

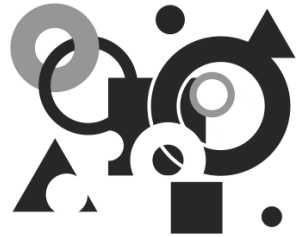
Как определить, работоспособны ли резервные копии?

Чтобы протестировать ваш план аварийного восстановления, вы должны убедиться в том, что ваши резервные копии пригодны к использованию, а выработанные вами процедуры работают. Я рекомендую вам тестировать выработанные вами алгоритмы и резервные копии раз в квартал. В любом случае такое тестирование необходимо производить не реже, чем один раз в течение каждого года.

Наиболее значительной проблемой, с которой вы можете столкнуться в реализации вашего плана аварийного восстановления, является неработоспособность резервных копий базы данных. В частности, если вы неправильно выполняете резервное копирование ваших баз данных, может сложиться такая ситуация, когда на первый взгляд вам будет казаться, что процессы резервного копирования прошли успешно, но при этом в результате вы будете получать резервные копии поврежденной базы данных. Нет ничего хуже, чем сбои, оставшиеся незамеченными.

Простейший способ тестирования резервных копий ваших баз данных заключается в написании и установке некоторых автоматических программ-"ботов" (bots), создающих высокую нагрузку по записи транзакций в вашу базу данных. После этого запустите скрипт, выполняющий процедуру резервного копирования при работающих "ботах", а затем попробуйте восстановить вашу базу данных по полученной резервной копии. Если ваш процесс резервного копирования не совсем корректен, то в результате этого теста вы должны будете получить поврежденную базу данных.

Глава 7



Масштабирование облачной инфраструктуры

Одной из наиболее полезных особенностей облачных инфраструктур является их способность автоматически масштабировать инфраструктуру вертикально и горизонтально, оказывая минимальное влияние или вообще не влияя на приложения, работающие в этой инфраструктуре. По правде говоря, наиболее полезным подходом будет занижение оценки. Эта функция фундаментально меняет отношение IT-менеджеров к их инфраструктурам и изменяет подход финансовых менеджеров к затратам на финансирование IT. Но, тем не менее, эта возможность представляет собой "обоюдоострый меч".

Очевидное преимущество масштабирования облачной инфраструктуры заключается в том, что вы платите только за те ресурсы, которые вы фактически потребляете. Необлачный подход заключается в приобретении такой инфраструктуры, которая может покрыть пиковые нагрузки, что является расточительством, и при этом вам остается лишь надеяться на то, что ваши прогнозы, касающиеся пиковых нагрузок, в самом деле совпадут с реальностью. Оборотной стороной медали, однако, является то, что масштабирование облачной инфраструктуры может послужить для ленивых системных архитекторов отговоркой, которую они используют, чтобы оправдать свое нежелание планировать потребности в вычислительных мощностях. В дополнение к этому, чрезмерное доверие к масштабированию в облачной среде может привести компанию к тому, что она начнет запускать новые экземпляры для удовлетворения таких потребностей, которые просто не принесут никаких деловых выгод.

В данной главе я рассмотрю все вопросы масштабирования инфраструктуры в облачной среде, попытаюсь научить вас грамотно оценивать все имеющиеся возможности, и при этом избежать лишнего риска. Путь к успеху начинается с осмотрительного планирования мощностей.

Планирование мощностей

Планирование мощностей в принципе представляет собой разработку такой стратегии, которая гарантирует, что ваша инфраструктура сможет поддерживать те

потребности в ресурсах, которые нужны для ее нормальной работы. При этом для того, чтобы в полной мере рассмотреть все сложности планирования мощностей, потребуется написать целую книгу. Я настоятельно рекомендую вам книгу [4], поскольку планирование мощностей играет определяющую роль в успехе развертывания приложений в облачной среде.

В этой главе мы рассмотрим основные вопросы, относящиеся к масштабированию в облачной среде.

- Изучение статистики использования, меняющейся в зависимости от времени суток, дней недели, праздничных дней и сезонных колебаний в вашем бизнесе.
- Изучение реакции вашего приложения на нагрузку, с тем, чтобы выяснить, когда и какие дополнительные мощности вам могут потребоваться.
- Изучение ценности ваших систем для бизнеса. Это важно для того, чтобы понять, когда наращивание дополнительных мощностей идет бизнесу на пользу, а когда — нет.

Некоторые пользователи считают, что возможности облачных сред по автоматическому наращиванию мощностей по требованию, и возможности по сокращению мощностей, когда необходимость в них отпадает, делают планирование мощностей ненужным. Другие же считают, что планирование мощностей связано с расходами в десятки и сотни тысяч долларов, идущих на оплату консалтинговых услуг, и с радостью отказываются от него при переходе в облачную среду. Но обе эти мысли представляют собой не что иное, как опасные и вредные заблуждения.

Планирование мощностей в облачной инфраструктуре в действительности играет не менее важную роль, чем в физической инфраструктуре. И для разработки надлежащего плана вам вовсе не нужно затевать никаких экзотических проектов. В действительности ваша цель должна заключаться в том, чтобы убедиться, что дополнительные затраты, которые вы несете на масштабирование вашей инфраструктуры, действительно оправдывают ваши потребности и ваши цели, для которых эта инфраструктура и создавалась.

Ожидаемые потребности

Вам абсолютно необходимо знать, какие требования к ресурсам будет предъявлять ваше приложение. При этом я не утверждаю, что вы должны быть выдающимся предсказателем и точно определять, например, количество просмотров вашего Web-сайта на каждый день. Вам нужно просто создать приблизительный численный прогноз, который позволит вам добиться следующих целей:

- спланировать инфраструктуру, которая будет поддерживать ожидаемые нагрузки;
- определить, когда фактическая нагрузка начинает значительно отличаться от ожидаемой;
- понять, как изменяющиеся потребности вашего приложения влияют на вашу инфраструктуру.

Оценив ожидаемую нагрузку и поняв, как ваше приложение реагирует на возрастание нагрузки, вы сможете определить, какие типы серверов вам потребуются, каким должно быть их количество и какой будет их потребность в ресурсах. Это даст вам очевидное преимущество. Если вы не имеете представления о том, какое количество людей будет посещать ваш Web-сайт и использовать ваше приложение, вы не сможете заранее определить, будет ли созданная вами инфраструктура работать хорошо. Вполне возможно, что она не справится с нагрузкой уже после часа работы, и, наоборот, что вы заплатите неоправданно крупную сумму за вычислительные мощности, которые фактически будут простаивать.

Естественно, вы — не пророк и не можете предсказывать будущее. Целью планирования мощностей является совсем не исключение неожиданных пиковых нагрузок, потому что они неизбежны. Напротив, смысл планирования мощностей заключается в прогнозировании именно того, что вы ожидаете увидеть (и что будет для вас нормой), определении того, что должно считаться аномалией (неожиданным отклонением от этой нормы), и выработке правильной реакции на такие отклонения.

Рассмотрим, к примеру, сценарий, при котором вам необходимо создать такую инфраструктуру, которая обеспечивает поддержку 10 миллионов транзакций в секунду, и вы ожидаете, что при пиковых нагрузках вам необходимо будет поддерживать увеличение нагрузки в диапазоне от 1 до 5 миллионов транзакций в секунду по сравнению с этим средним значением. Если вы правильно оцените нагрузку, вы сможете определять, является ли внезапный выброс ожидаемым (и в этом случае беспокоиться не о чем) или неожиданным (в таком случае за ним нужно понаблюдать и принять меры, чтобы избежать потенциальных проблем с производительностью). Без надлежащей оценки ожидаемой нагрузки вы не сможете понять, что следует делать в случае резких колебаний нагрузки.

Определение ожидаемых потребностей

На рис. 7.1 и 7.2 представлены графики, отражающие ожидаемый трафик сайта системы электронной коммерции в течение типичного рабочего дня и предполагаемые пиковые нагрузки на ближайшие 12 месяцев.

График дневного распределения нагрузки показывает, что пиковые нагрузки наблюдаются по утрам, в обеденное время и в ранние вечерние часы. Основное затишье, когда трафика почти нет, сглаживает эти пики в ранние утренние часы.

Поскольку вы являетесь растущей компанией, в связи с расширением деловой активности следует ожидать постепенного роста потребностей в течение ближайшего года, причем пики активности планируются на май и сентябрь — как раз те месяцы, на которые у вас запланирован запуск новых продуктов. Наконец, в конце года традиционно ожидается рост продаж, типичный для конца года.

Каким образом мы получаем эти цифры? Как и во многих других случаях это зависит от вашего приложения. Например, для графика суточных колебаний нагрузки за основу прогноза берутся типичные образцы, наблюдавшиеся на практике уже в течение ряда лет.

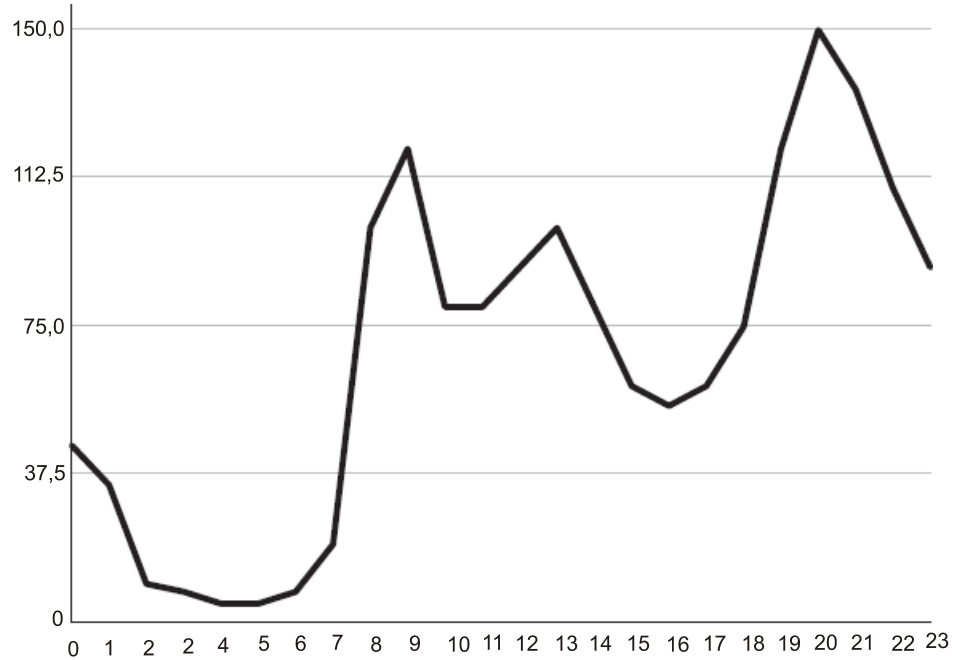


Рис. 7.1. Предполагаемое распределение дневной нагрузки на сайт системы электронной коммерции

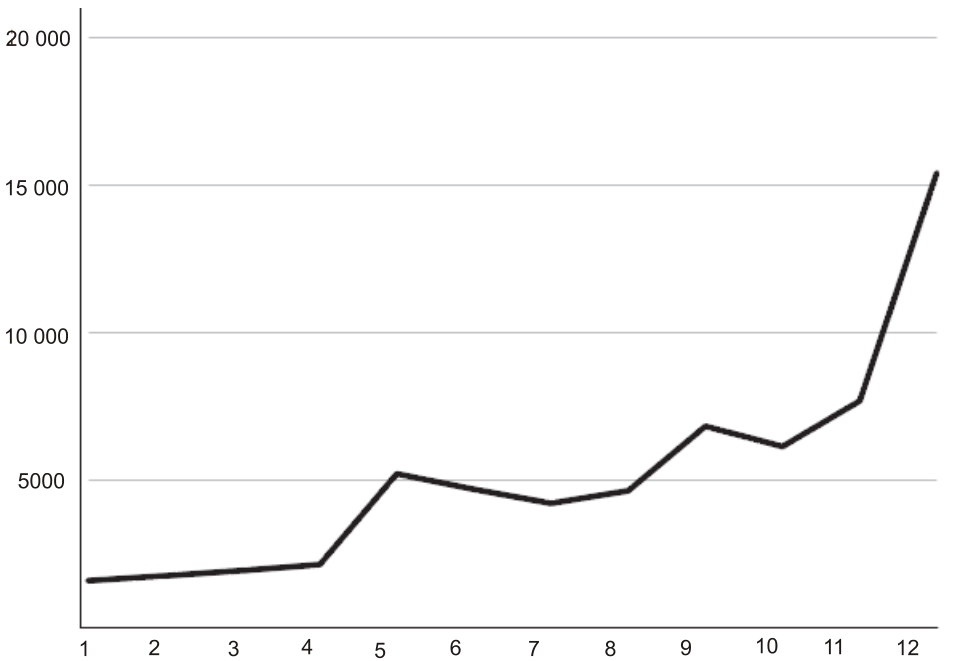


Рис. 7.2. Ожидаемое распределение нагрузки на сайт системы электронной коммерции на ближайшие 12 месяцев

Если ваша компания ведет устойчивый бизнес в течение нескольких лет, то "исторические шаблоны" будут надежной основой для вашего прогноза. Расширяющиеся компании, однако, могут увидеть изменения в этом типовом поведении по мере расширения и изменения рынка.

Более сложная задача стоит перед вами в том случае, если вы еще не имеете никаких исторических данных. Если вы хорошо понимаете рынок, то следует использовать оптимистический прогноз до тех пор, пока вы не накопите собственных исторических данных.

При оценке ежегодного уровня ваши прогнозы должны основываться и на других аспектах вашего бизнеса. Если вы, например, продаете потребительские товары, то высока вероятность сезонных колебаний спроса, особенно в конце года. Помимо общей сезонной доступности, вы во многом зависите от того, что делает ваше предприятие для привлечения трафика на ваш Web-сайт. Для этой цели вам следует заручиться поддержкой отделов продаж, маркетинга, рекламы и всех, кто способствует привлечению посетителей на ваш сайт. Например, как показано на рис. 7.2, вы должны обладать полной информацией об ожидаемых запусках новых продуктов в мае и об ожиданиях компании, связанных с этим продуктом, для того чтобы правильно учесть эти факторы в своем прогнозе.

Анализ неожиданностей

Вы непременно столкнетесь и с неожиданным (аномальным) трафиком. Иногда бывает так, что когда компания запускает новый продукт, популярность этого продукта превосходит ее ожидания; иногда оказывается так, что продукт получает незапланированное широкое освещение в прессе; а иногда случается и так, что вы ошибаетесь в прогнозах. Каждый раз, когда вы видите, что трафик вашего сайта растет неожиданно и резко, вы должны в первую очередь понять, почему он отличается от ожидаемого. Как будет рассказано далее в этой главе, неожиданный трафик может быть как хорошим, так и плохим признаком. Он может потребовать изменения всех ваших прогнозов или оказаться просто случайным "выбросом".

Влияние нагрузки

Способность Web-приложения или Web-сайта к масштабированию непосредственно связана с пониманием того, в чем заключаются ограничения по ресурсам, и какое влияние окажет добавление различных ресурсов на ваше приложение. К сожалению, системные архитекторы чаще всего считают, что простое добавление нового сервера в расширяемую архитектуру поможет решить любые проблемы с производительностью приложения. В действительности же, например, добавление еще одного сервера приложений в инфраструктуру, где работает сервер базы данных, предъявляющий повышенные потребности к дисковым операциям чтения/записи, только усугубит возникшую проблему. Поэтому системным архитекторам необходимо четко понимать "шаблоны" использования ресурсов приложениями, а

также выполнять тестирование с целью определения, какие сценарии создают точки повышенной нагрузки на отдельные узлы производственной инфраструктуры.

В предположении того, что вы имеете грамотно разработанное Web-приложение и эффективный движок базы данных, можно перечислить следующие потенциальные ограничения по производительности при развертывании этого Web-приложения в облачной среде.

- Полоса пропускания канала связи к балансировщику нагрузки.
- Ресурсы CPU и RAM на балансировщике нагрузки.
- Способность балансировщика нагрузки правильно распределять нагрузку по серверам приложений.
- Полоса пропускания каналов связи, связывающих балансировщик нагрузки и серверы приложений.
- Ресурсы CPU и RAM на сервере приложений.
- Производительность дисковой подсистемы для операций чтения на сервере приложений.
- Производительность дисковой подсистемы для операций на сервере приложений, вторичное ограничение дисковой подсистемы, если приложение кэширует данные на диск.
- Ширина полосы пропускания канала связи между сервером приложений и сетевыми устройствами хранения данных (например такими, как эластичные блочные устройства Amazon (Amazon EBS).
- Ширина полосы пропускания канала связи между сервером приложений и сервером базы данных.
- Производительность дисковой подсистемы для операций записи не сервере базы данных.
- Производительность дисковой подсистемы для операций записи на сервере главной базы данных.
- Свободное дисковое пространство для поддержания потребностей в хранении данных.

Существует множество аспектов, по которым приложение может начать испытывать нехватку ресурсов. Впрочем, в действительности многие из них вряд ли создадут реальную проблему. Настоящие проблемы может создать как раз архитектура вашего Web-приложения, а не эти факторы.

К вопросу об архитектуре приложений и баз данных

В *главе 4* я рассматривал способы правильного проектирования приложений для работы с базами данных в расчете на их масштабирование в облачной среде. Подводя итоги, можно сказать, что вам необходимо следовать тем же правилам, что и для горизонтального масштабирования вне облачной среды. Вот рекомендации, которые необходимо соблюдать:

- используйте как можно более скоростные устройства хранения данных для ускорения доступа к базе данных;

- избегайте хранения транзакционных данных на уровне сервера приложений;
- создайте несколько копий сервера приложений, работающих с одной и той же базой данных, причем эти серверы приложений не должны взаимодействовать друг с другом;
- правильно индексируйте свою базу данных;
- по возможности используйте архитектуру "главный/подчиненный" и перенаправляйте операции чтения на подчиненные серверы;
- работая в Amazon EC2, планируйте избыточность таким образом, чтобы минимизировать трафик, пересекающий границы зон доступности.

Шкалы масштабирования

В зависимости от вашего приложения наиболее вероятными участками напряженности окажутся следующие три компонента:

- CPU на сервере приложений;
- объем RAM на сервере приложений;
- дисковая подсистема (ввод/вывод) на сервере базы данных.

Каждое приложение имеет свои "узкие места". Если бы это было не так, то любое приложение могло бы работать на старом компьютере Intel 386 под неограниченно высокой нагрузкой. Например, для приложения Valtira (Web-приложение, написанное на Java, которое я проектировал для одноименной компании) первым "узким местом" всегда является CPU на сервере приложений. Поскольку приложение масштабируется горизонтально, CPU перестает быть слишком серьезным фактором, но (в зависимости от информационного наполнения, с которым работает Valtira) мы сталкиваемся с еще одним "узким местом" — это будет либо полоса пропускания сети, либо производительность подсистемы дискового ввода/вывода. В облачной среде этим "узким местом" обычно становится производительность дискового ввода/вывода.

Поэтому нашим следующим шагом по масштабированию приложения становится распределение операций чтения по нескольким подчиненным серверам базы данных и использование главного сервера базы данных исключительно для операций записи[1]. Общая тенденция такова, что при дальнейшем росте нагрузки следующим "узким местом" становится производительность дисковой подсистемы на главном сервере базы данных (операции записи). При достижении этого момента нам необходимо сегментировать базу данных или задуматься о применении более дорогого решения в области управления базами данных.

Иными словами, необходимо четко понимать принципы работы приложения с тем, чтобы, столкнувшись с ограничениями производительности, эффективно решить проблему за счет снятия этих ограничений. Если такого понимания не выра-

[1] Чтобы иметь возможность отделить операции чтения от операций записей, вам необходимо обеспечить в вашем приложении защиту от "грязной записи", о чем подробнее рассказывалось в *главе 4*. О феномене "грязной записи" (dirty writes) подробнее см. здесь: **http://www.rsdn.ru/article/db/deadlocks.xml**.

ботать, то общая тенденция такова, что большинство архитекторов будут думать о том, что решением проблемы будет добавление новых серверов приложений, хотя, как уже было сказано, это не всегда так.

Конечно, вы можете провести очень сложный и дорогой анализ с тем, чтобы определить точное количество пользователей, при достижении которого необходимо предпринимать шаги по масштабированию приложения, и в чем должны заключаться эти шаги. Однако это не всегда является необходимостью. Достаточно запустить ваше приложение в среде с ожидаемым уровнем масштаба, провести реалистичные тесты поведения приложения при повышении нагрузки, а затем выполнить необходимое масштабирование. Времени это займет немного, обойдется в незначительную сумму, а полученная вами информация будет достаточно точна (за исключением сложных ситуаций, когда вам действительно потребуются услуги экспертов в области планирования мощностей).

Ценность ваших вычислительных мощностей

Если вы имеете дело с Web-приложениями, не стоит добавлять в систему дополнительные мощности только потому, что процессоры в вашей инфраструктуре достигли порога, когда их мощности используются на 90 %. Когда это происходит, важно ответить на следующий вопрос: "А что мне даст поддержка дополнительных нагрузок?" Ответ на этот вопрос можно получить, оценив стоимость поддержки потребностей вашей системы.

В вычислительных грид-системах оценить стоимость поддержки дополнительных нагрузок обычно бывает довольно просто. Например, пул серверов масштабируется специально для того, чтобы визуализировать большие объемы видео. Если в этом и заключается ваш бизнес, вы должны иметь возможность определять, сколько стоит визуализация каждого видеофайла для вас, и соответствующим образом провести анализ затрат и прибылей. Понимание того, следует ли затребовать дополнительные мощности для обработки нового видео или же стоит подождать с его визуализацией пока не освободятся имеющиеся мощности, поможет вам эффективнее планировать свои затраты на видеорендеринг. Но в большинстве случаев принятие решений по добавлению мощностей для поддержки систем, не основанных на Web, представляет собой довольно простой процесс.

Web-приложения не настолько стандартны и шаблонны. Обычно Web-сайт поддерживает десятки или даже сотни сценариев использования, каждый из которых имеет собственную ценность для бизнеса. Ценность сайта, поддерживающего систему электронной коммерции, сильно отличается от ценности блога директора по маркетингу. Поэтому 100 % всплеск активности, относящийся к успешному запуску продукта, гораздо важнее, чем такой же всплеск активности, вызванный записью в блоге директора и ссылками на нее из Twitter.

Простой мысленный эксперимент

Рассмотрим простейший пример, объясняющий смысл обсуждаемых вопросов. Представьте себе, что у нас есть простое Web-приложение, работающее с корпоративным контентом, который служит основным инструментом продаж. Мы пользуемся модулем SalesForce.com Web2Lead[1] для получения руководящих указаний с Web-сайта и переноса этой информации в SalesForce.com. Интранет-компонент, поддерживающий этот Web-сайт, позволяет коллективу отдела маркетинга планировать рекламные кампании, формировать целевые страницы (landing pages) и вести базовую отчетность.

Допустим, что на Web-сайте наблюдается внезапный и неожиданный всплеск активности. Поскольку уровень нагрузки приближается к предельному, поддерживаемому имеющимися вычислительными мощностями, нам необходимо решить, что следует делать дальше: наращивать мощность или просто переждать данный всплеск активности?

Чтобы сделать запрос на наращивание мощностей, нам необходимо определить, сколько это будет стоить, и какова будет ценность этой добавленной мощности для предприятия. Для этого нам необходимо ответить на следующие вопросы:

- Как нехватка мощностей повлияет на посетителей сайта?
- Является ли данный всплеск активности нормальной реакцией на повышение интереса пользователей к системе (т. е. является ли он результатом нормальной деятельности)? Нет ли еще более серьезной проблемы, о которой необходимо задуматься (не есть ли это признак производимой на сайт DoS-атаки)?
- Следует ли ожидать дальнейшего повышения активности? Приблизились ли мы к пиковой мощности?
- В какие суммы обойдется нам бездействие (т. е. что произойдет, если мы пустим ситуацию "на самотек"?). Что обойдется нам дороже: добавление дополнительных мощностей или тактика выжидания?

В рассматриваемом примере предположим, что все операции по масштабированию выполняются вручную. Теперь нам необходимо провести некоторые исследования. Представьте себе, что на поставленные вопросы мы дали следующие ответы:

- *Как нехватка мощностей скажется на посетителях сайта?* Поскольку мы приближаемся к предельной мощности, работа нашего Web-сайта замедлится, и будет замедляться до тех пор, пока система не прекратит обрабатывать вводимые пользователями регистрационные данные (входные имена и пароли). Если

[1] Формы Web-to-Lead (Web2Lead) представляют собой основной компонент автоматизации деятельности отделов маркетинга и продаж. Их цель заключается в захвате данных, предоставленных посетителями Web-сайта (например, таких как контактная информация и сведения о заинтересованности в тех или иных продуктах) и сохранении их в виде записей (Lead) в базе данных продуктов CRM (в данном случае Salesforce.com). Подробнее см. **http://www3.formassembly.com/blog/tutorial-how-to-create-a-salesforce-web-to-lead-form/**. — *Прим. перев.*

система слишком перегружена, скорость работы сайта замедлится до такой степени, что пользоваться им станет невозможно.

- *Считаем ли мы этот всплеск нормальным и законным?* Анализ трафика показывает, что запись в блоге директора по маркетингу содержит нечто, вызвавшее повышенный интерес пользователей и средств массовой информации. Блог находится под мощным "обстрелом", но весь остальной трафик сайта выглядит абсолютно нормальным.
- *Стоит ли ожидать ухудшения ситуации?* Это маловероятно, и трафик пошел на спад. Но, может быть, это просто временное "плато"?
- *В какую сумму нам обойдется выжидательная тактика?* В значительной степени это не приведет к существенным затратам, потому что постоянные пользователи сайта работать все-таки могут. Наиболее сильно от этой повышенной активности страдают служащие маркетингового отдела, которые в течение нескольких часов вынуждены мириться с проблемами с доступом или полным его отсутствием. С другой стороны, мы работаем в облачной среде, а результаты тестирования показали, что добавление одного сервера приложений позволит вернуть работу системы к норме. Иными словами, наращивание мощностей обойдется нам в несколько долларов.
- *Ответ.* Мы выбираем наращивание мощностей. Это не принесет нам большой выгоды, но и стоит совсем недорого.

Насколько различными могут оказаться исходы?

Наиболее важный урок, извлеченный из рассмотрения описанного примера, заключается в понимании того, насколько разными могли бы быть результаты тех или иных действий, если бы сайт не был развернут в облачной среде. Затраты на наращивание мощности могли бы оказаться огромными, а к моменту получения нового сервера необходимость в нем могла бы уже отпасть.

Еще один вариант, заслуживающий рассмотрения, заключается в том, как развивалась бы ситуация, если бы в трафике были выявлены аномалии — например такие, как деятельность неподконтрольной программы-робота, пытающейся обнаружить уязвимости сайта. При таком сценарии принятие решений усложняется, поскольку в этом случае нет полной ясности в вопросе о том, поможет ли наращивание мощностей. На практике возможно такое развитие ситуации, когда в ответ на предпринятое вами наращивание мощностей программа-робот или даже бот-сеть интенсифицируют свою деятельность, что вынудит вас предпринять дополнительное наращивание мощностей. В итоге ситуация начнет "раскручиваться" по спирали, и вы все равно окажетесь жертвой бот-сети. При нормальном трафике стоимость наращивания мощностей пренебрежимо мала, но при атаке бот-сети вы будете вынуждены тратить все большие и большие суммы, подвергаясь при этом внешней угрозе и не получая никаких деловых выгод.

Ключевой вывод, который вы должны сделать из анализа такого развития событий, заключается в том, что внезапное возрастание потребности в ресурсах еще не означает, что вы должны моментально наращивать мощности.

Масштабирование в облачной среде

Облачная инфраструктура дает вам возможность менять находящиеся в вашем распоряжении ресурсы и вычислительные мощности таким образом, чтобы они соответствовали вашим потребностям, диктуемым рабочей нагрузкой. Регулировать вычислительные мощности можно как вручную (вводя команды из командной строки или через Web-интерфейс), так и программно (внося предопределенные изменения в настройку мощностей или с помощью специализированных программ, автоматически регулирующих вычислительные мощности в соответствии с текущими потребностями).

Способность ручного регулирования вычислительных мощностей предоставляет облачной среде огромное преимущество по сравнению с традиционной инфраструктурой IT. Но истинную мощь масштабирования в облаке можно оценить, только прибегнув к динамическому масштабированию.

- *Динамическое масштабирование* (Dynamic scaling). Этот термин, который я считаю взаимозаменяемым с другим термином — облачное масштабирование (cloud scaling) — позволяет программно регулировать ресурсы, доступные в вашей инфраструктуре, не требуя интерактивного вмешательства пользователей. Динамическое масштабирование может принимать формы превентивного масштабирования (proactive scaling) или реактивного масштабирования (reactive scaling).
- *Превентивное масштабирование* (Proactive scaling) — эта тактика предполагает составление расписания, в соответствии с которым производится масштабирование вашей инфраструктуры в соответствии с прогнозируемыми запросами. Если мы вернемся к рассмотрению приложения, обсуждавшегося в *разд. "Определение ожидаемых потребностей"* (см. рис. 7.1), то нам необходимо сконфигурировать наши инструменты управления облачной инфраструктурой таким образом, чтобы запускать минимальную инфраструктуру, поддерживающую наши потребности, в ранние утренние часы, наращивать мощности с поздних утренних часов, снова снижать мощности до базового уровня до обеденного времени и, таким образом, менять мощности в соответствии с суточными колебаниями нагрузки. При использовании этой стратегии мы не ждем повышения наших потребностей в наращивании мощностей, а заранее, превентивно, наращиваем мощности в соответствии с разработанным планом.
- *Реактивное масштабирование* (Reactive scaling) — при использовании этой стратегии ваша инфраструктура реагирует на изменение потребности за счет добавления или отключения мощностей, полагаясь на собственный анализ ситуации. При экспериментальной оценке мощностей среда, в которой принята модель

реактивного масштабирования, может автоматически наращивать мощности, обнаружив неожиданные всплески активности (как в примере, где всплеск активности был вызван записью в блоге директора фирмы по маркетингу).

Примечание

Если вы читаете мой блог (**http://georgereese.tumblr.com/**), вы, наверное, заметили, что в прошлом я использовал термин "динамическое масштабирование" (dynamic scaling), говоря о превентивном масштабировании, и "автоматическое масштабирование" (autoscaling), когда речь заходила о реактивном масштабировании. Теперь я изменил терминологию, поскольку термины "динамическое масштабирование" и "автоматическое масштабирование" оказались не слишком удачными и создавали много путаницы, в то время как термины "превентивное масштабирование" (т. е. масштабирование по заранее разработанному плану) и "реактивное масштабирование" (т. е. масштабирование как реакцию на неожиданное повышение нагрузки) достаточно четко дают представление о том, что имеется в виду, и о разнице между этими двумя подходами.

Средства и системы мониторинга

На протяжении всей этой книги я говорил о средствах управления облачной инфраструктурой и системах ее мониторинга как о критически важных инструментах для нормального функционирования инфраструктуры компании в облачной среде. Я управляю одной такой компанией, enStratus **(http://www.enstratus.com/)**, но есть и множество других систем, предназначенных для той же цели, в том числе — RightScale (**http://www.rightscale.com/**) и Morph (**http://mor.ph/**). Какая из этих компаний лучше всего подойдет именно вам, зависит от вашего бюджета, от типов приложений, которыми вы управляете, и какие компоненты вашей инфраструктуры для вас наиболее важны.

Какой бы инструмент вы ни подобрали для себя, он, как минимум, должен обладать следующими возможностями (все три упомянутых мной компании их предоставляют):

- планирование изменений в объемах мощностей при развертывании ваших приложений;
- мониторинг процесса развертывания с целью выявления избыточных потребностей и случаев падения потребностей ниже нормального уровня;
- автоматическая настройка доступных мощностей на основании неожиданных всплесков и падений активности в зависимости от текущих потребностей.

Мониторинг подразумевает много большее, чем простое наблюдение за потребляемыми мощностями и подключение/отключение серверов. В *главе 6* рассматривалась роль, которую средства мониторинга играют в обнаружении случаев отказа в облачной среде и восстановлении после сбоев. Наиболее важным компонентом системы мониторинга является система оповещений. Вам необходима полнофункциональная система ведения журналов любых изменений в потребляемых мощностях (не имеет значения, запланированных или неожиданных), а также средство рассылки уведомлений об аномалиях по электронной почте.

Если у вас нет возможностей или желания платить за программное обеспечение от сторонних производителей, вы можете разработать и развернуть собственную систему мониторинга. Архитектуру системы мониторинга иллюстрирует рис. 7.3.

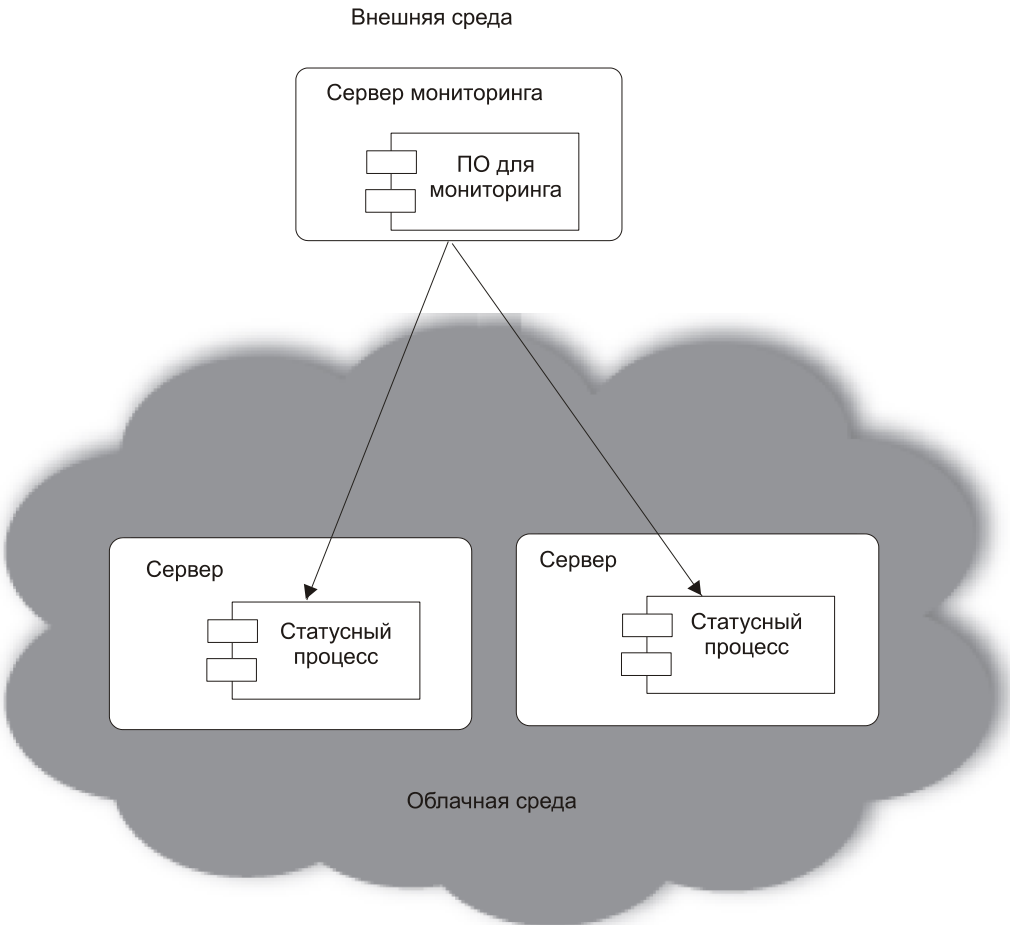


Рис. 7.3. Общая архитектура системы мониторинга состояния облачной инфраструктуры

По сравнению с процессом аварийного восстановления, для целей планирования мощностей не настолько важно расположение сервера мониторинга — в облачной среде или за ее пределами. Тем не менее, расположение сервера мониторинга вне облачной среды — это очень хорошая идея, и она очень важна, если управление полосой пропускания является частью вашего профиля мониторинга.

Система мониторинга ведет наблюдение за каждым отдельным сервером, чтобы собрать единую картину всех текущих ограничений по ресурсам. Каждый облачный экземпляр имеет процесс, способный собрать основную информацию о данном экземпляре и отослать отчет на сервер мониторинга. Большинство современных операционных систем обладает возможностью запуска статусного процес-

са для сбора основных системных данных о ресурсах, включая CPU, RAM, а также остальные данные, связанные с протоколом SNMP. В дополнение, сервер приложений Java поддерживает интерфейсы JMX (Java Management eXtensions)[1], которые позволяют запрашивать информацию о производительности ваших виртуальных машин Java.

В целях обеспечения безопасности я предпочитаю, чтобы сервер мониторинга осуществлял опрос облачных экземпляров, а не создавать такую конфигурацию, при которой сами облачные экземпляры регулярно отсылали бы отчеты на сервер мониторинга. За счет осуществления опроса вы можете разместить ваш сервер мониторинга за брандмауэром и не допускать на сервер никакого входящего трафика. Кроме того, необходимо иметь в виду, что вы должны иметь возможность масштабировать и сам сервер мониторинга по мере роста количества узлов, за которыми он ведет наблюдение.

Процесс, проверяющий показатели производительности каждого из экземпляров, должен меняться в зависимости от функции, выполняемой этим экземпляром. Балансировщик нагрузки может быть достаточно примитивным, поэтому все, что заслуживает мониторинга — это уровни загрузки RAM и CPU. Сервер базы данных нуждается в несколько более интеллектуальном мониторинге: в список жизненно важных показателей производительности должна входить производительность операций дискового ввода/вывода, так как это позволит быстро выявить признаки надвигающейся проблемы. Наиболее сложен процесс мониторинга ваших серверов приложений. Он должен обладать возможностью не только сообщать об уровне использования ресурсов экземпляра, но и отображать образцы активности на соответствующем экземпляре.

Сервер мониторинга обрабатывает все полученные данные и анализирует собранную информацию. За счет этого он определяет, когда наступает момент, в который необходимо превентивно добавлять или отключать ресурсы, распознает аномальную активность и запускает предопределенные операции в ответ на обнаруженную аномальную активность.

Процесс выделения ресурсов в облачной среде

Вне зависимости от того, выполняете ли вы динамическое масштабирование или вручную изменяете уровни потребления, ваши представления о денежных затратах в облачной инфраструктуре сильно отличаются от взглядов на денежные траты в традиционной инфраструктуре. Когда вы добавляете новые ресурсы в ваш внутренний центр обработки данных или в центр обработки данных провайдера

[1] Технология JMX была разработана для упрощения разработки систем мониторинга и управления. Причем управлять можно фактически чем угодно — лишь бы это было написано на Java. Это может быть микроустройство типа считывателя отпечатка или система, включающая тысячи машин, каждая из которых предоставляет определенные сервисы.
Подробнее см. **http://java.sun.com/javase/technologies/core/mntr-mgmt/javamanagement/**, **http://www.java-course.ru/articles/jmx.html**, **https://secure.wikimedia.org/wikipedia/en/wiki/Java_Management_Extensions**. — *Прим. перев.*

услуг внешнего управления (MSP), вам обычно необходимо сначала оформить заказ на покупку, который должен быть одобрен финансовым отделом или бухгалтерией вашей компании. Финансовый отдел одобряет заказ, основываясь на бюджете вашей компании, после чего ваш заказ отправляется поставщику. Вы не можете, например, потратить на новый сервер сумму $3000 до тех пор, пока эта трата не будет одобрена финансовым отделом вашей компании.

С другой стороны, в инфраструктуре AWS ничто не мешает вам запускать команду `ec2-run-instances` каждый раз, когда вам требуется новый экземпляр EC2 модели `extra-large`, и тратить на это примерно $7000 в течение года. Каждый, кто имеет возможность запускать новые экземпляры или менять критерии масштабирования ваших средств управления облачной средой, имеет полные права поставки облачных ресурсов. IT-менеджер не должен получать никаких одобрений от финансового отдела, он просто меняет конфигурационные параметры программы, доступа к которой финансовый отдел может и не иметь.

Поэтому финансовый отдел вашей компании должен привлекаться к планированию и одобрению вашего месячного бюджета на работу облачной инфраструктуры. Более того, вам необходимы соответствующие инструменты, которые позволят гарантировать, что любые изменения, вносимые в облачную инфраструктуру, не будут превышать согласованный и одобренный бюджет. Если вы не используете интерактивный подход, когда в процессе масштабирования участвуют люди, вы рискуете тем, что финансовый отдел из сторонников ваших планов перехода на облачную среду станет одним из ваших наиболее ожесточенных оппонентов.

Управление превентивным масштабированием

Хорошо разработанная система с превентивным масштабированием позволяет вам составить расписание изменений потребляемых мощностей в зависимости от прогнозируемых изменений в потребностях приложения. При использовании превентивного масштабирования вы должны будете оперировать со среднеквадратичными отклонениями от ваших прогнозов. Для этого вам не требуется зарываться в учебники по статистике или отказываться от ваших планов только потому, что я упомянул термин "статистика". Я имею в виду, что вы должны в общих чертах понимать, насколько нормальный трафик отличается от ваших ожиданий. Если вы ожидаете, что статистика использования вашего сайта будет предусматривать 1000 просмотров страниц за час в полуденное время, а в действительности вы видите, что этот показатель составляет 1100 просмотров страниц за час, то соответствует ли это вашим ожиданиям? Вероятно, нет.

Ваши потребляемые мощности на любой момент времени должны поэтому обладать возможностью покрыть эти повышенные потребности с некоторым запасом. Наиболее эффективное использование ваших ресурсов заключается не столько в их объемах, сколько в планировании суточных колебаний, и именно они создадут вам проблемы, если вы ошибетесь в своих прогнозах — даже если вы будете пользоваться реактивным масштабированием. Наиболее сложным для понимания в пла-

нировании мощностей является как раз то, что следует понимать под словами "с некоторым запасом".

Управление реактивным масштабированием

Реактивное масштабирование — это очень хорошее и мощное средство, но прочный канат ведь можно использовать не только для страховки, но и для того, чтобы, скажем, взять и повеситься. Реактивное масштабирование позволяет вам очень быстро реагировать на неожиданное возрастание потребностей в ресурсах. Но если вы не будете выполнять никакого планирования мощностей, а вместо этого целиком и полностью положитесь на реактивное масштабирование вашего Web-приложения, вы очень быстро обнаружите, что вместо ожидаемой пользы оно может нанести вам серьезный ущерб.

Самая грубая форма реактивного масштабирования — это реактивное масштабирование, основывающееся на фактическом потреблении. Иными словами, в тот момент, когда потребление ресурсов CPU, RAM или каких-либо других ресурсов достигает определенного уровня, вы просто добавляете дополнительные объемы нужных ресурсов в вашу среду. Это очень серьезно упрощает логику системы мониторинга, но в реальной жизни вам все-таки требуется некоторое время на размышления. Мы уже видели несколько примеров, когда такой подход терпит неудачу.

- Добавление сервера приложений, когда проблема вызвана снижением производительности операций дискового ввода/вывода на сервере базы данных. В результате нагрузка на сервер базы данных от этого еще более возрастет, и это только усугубит ситуацию.
- Атака, на преодоление которой вы начинаете наращивание ресурсов. В результате ваше потребление начинает расти, "раскручиваясь по спирали", в результате чего вы ничего не добьетесь, а только резко повысите ваши расходы на ресурсы.
- Неожиданный "всплеск" активности в Web, который начинает влиять на ваши затраты на инфраструктуру, но слабо влияет на работу конечных пользователей. Вы знаете, что эта активность скоро спадет, но ваша система мониторинга "тупо" наращивает мощности, просто потому, что имело место резкое возрастание нагрузки.

Хорошая система мониторинга должна предоставлять такие инструменты, которые позволяют сглаживать эти потенциальные проблемы, вызванные реактивным масштабированием. Однако я еще не видел такой системы, которая бы при таком сценарии действовала идеально. Для этого требуется логика, позволяющая выделить проблемную область и проанализировать активность с тем, чтобы определить ситуации, в которых имеет смысл запускать новые экземпляры, а в каких — нет. Здесь, к сожалению, никакой алгоритм пока не может сравниться с человеческой интуицией.

Тем не менее, ваша система мониторинга может определять правила для реактивного масштабирования, и вы всегда должны иметь в ней регулятор, который может взять управление на себя. Этот регулятор устанавливает пределы, ограничивающие количество ресурсов, которые может автоматически задействовать система мониторинга. За счет этого вы имеете некоторую возможность ограничить финансовые затраты, которые мониторинговая система может предпринять от вашего имени. Если на вашу инфраструктуру предпринята атака, то система мониторинга может со временем достигнуть предела, установленного вашим ограничителем, но от этого вы не перестанете тратить деньги на безумное количество экземпляров, запущенное в вашей облачной среде.

Наконец, есть еще один аспект, влияющий как на превентивное, так и на реактивное масштабирование (но в большей степени — на реактивное). Этой проблемой является подверженность ошибкам интерфейсов прикладного программирования Amazon S3 и AWS. Если вы рассчитываете на реактивное масштабирование в надежде на то, что у вас всегда будет достаточно ресурсов, то проблемы с Amazon S3 ослабляют ваши планы. Ваша система рано или поздно обманет ваши ожидания.

Рекомендуемый подход

Я не являюсь ярым сторонником реактивного масштабирования, но оно находит свои области применения. При управлении своей инфраструктурой я предпочитаю полагаться на превентивное масштабирование на основе прогнозируемой потребности, включая избыточные мощности, в два раза превышающие ожидаемую нагрузку и верхние пределы ожидаемых нагрузок (от трех до пяти среднеквадратичных отклонений от ожидаемого уровня). При такой настройке реактивное масштабирование обычно почти никогда и не требуется.

Неожиданный рост потребностей действительно существует. Вместо использования реактивного масштабирования в ответ на непредвиденное возрастание нагрузки я обычно даю время персоналу проанализировать ситуацию и отреагировать соответственно. Мои ограничители обычно установлены на значения от 150 до 200 % от базовой конфигурации. Иными словами, если моя базовая конфигурация состоит из двух серверов приложений, и мое приложение может масштабироваться до 200 % от базовой конфигурации за счет добавления двух дополнительных серверов приложений, я устанавливаю лимит масштабирования и конфигурирую мониторинговую систему таким образом, чтобы она заранее выдавала предупреждение при необходимости добавить хотя бы один.

В результате такого применения реактивного масштабирования я получаю возможность управляемого автоматического масштабирования моей инфраструктуры в ответ на неожиданный рост нагрузки, но только до определенного предела. Кроме того, я также должен иметь время на анализ аномальной активности с тем, чтобы определить, должен ли я поднять ограничения по масштабированию, изменить

мою базовую конфигурацию или же просто проигнорировать происходящее. Поскольку моя инфраструктура не работает на пределе мощности, и я не полагаюсь на реактивное масштабирование, сбои Amazon S3 вряд ли окажут на мою инфраструктуру негативное влияние.

Хотя приведенные мною показатели могут не иметь никакого смысла для вашего Web-приложения, но общая политика должна быть вам понятна, и, как я надеюсь, она и вам будет служить так же хорошо, как и мне. Какими бы ни были абсолютные показатели для вашего приложения, вы в любом случае должны определить бюджет, который позволит вам работать на пределе мощности, по крайней мере, до тех пор, пока не будет согласовано и одобрено расширение бюджета. Если ваш бюджет одобрен для ожидаемых мощностей, а вы обнаруживаете, что начали работать не пределе, установленном вашими ограничителями, вряд ли это обрадует финансовый отдел вашей компании.

Вертикальное масштабирование

До сих пор все обсуждение было сосредоточено на горизонтальном масштабировании, которое сводится к добавлению новых серверов. Вертикальное масштабирование, в отличие от горизонтального, сводится к замене существующих серверов более мощными или более специализированными. Все виртуализованные среды — и, в частности, облачные среды, очень хорошо масштабируются горизонтально. Но когда речь заходит о вертикальном масштабировании, то здесь облачные среды имеют определенные сильные стороны и некоторые значительные недостатки.

Сильной стороной облачной инфраструктуры в отношении вертикального масштабирования является простота, с которой вы можете протестировать менее мощные и менее специализированные конфигурации и оценить потребности системы в более мощных или более специализированных конфигурациях. Облачные среды (а к облачной среде Amazon это относится даже в большей степени, чем к ее конкурентам) гораздо хуже показали себя при работе со специализированными системными конфигурациями.

На настоящий момент Amazon предоставляет пять вариантов системных конфигураций. Если какой-нибудь из компонентов вашего приложения требует больших объемов RAM, чем поддерживается экземплярами Amazon, то считайте, что вам не повезло. По контрасту с этой ситуацией GoGrid предоставляет гораздо больше возможностей по индивидуальной настройке, включая индивидуальную работу по определению конфигураций с высокопроизводительными системами дискового ввода/вывода. Но, в конечном счете, ни один из этих вариантов не даст вам таких возможностей по созданию высокоиндивидуализированных конфигураций, как в случае применения конфигуратора Dell.

Хотя я привел в этой книге достаточно много информации о горизонтальном масштабировании, сам я всегда сначала пытаюсь выполнить вертикальное.

НАЧИНАЙТЕ С МАЛОГО

Когда мы строим приложения для Valtira, мы всегда начинаем с экземпляров Amazon модели `medium` (средний уровень) и тестируем приложение, чтобы убедиться в том, что оно и в самом деле требует использования моделей `large` или `extra-large`. И этот подход оправдан, потому что экземпляры на основе модели `large` стоят в четыре раза дороже, чем экземпляры на основе модели `medium`. Если вы можете добиться такой же производительности, разворачивая ваши приложения на четырех серверах приложений модели `medium`, как и на одном экземпляре на основе модели `large`, то вам лучше использовать экземпляры, основанные на модели `medium`, потому что таким образом вы одновременно добиваетесь как нужной производительности, так и высокой доступности. В случае же с экземпляром на основе модели `large` вы обеспечиваете только производительность.

Вертикальное масштабирование в облачной среде Amazon наиболее эффективно в случаях, когда вам требуются большие объемы RAM. Отличным примером такого приложения является ранее упомянутое приложение Valtira. Первую точку масштабирования для Valtira (в основном — RAM) я здесь не рассматриваю. Большинство систем, развертываемых на платформе Valtira, не требуют больших объемов RAM — как правило, от 1 до 2 Гбайт бывает достаточно. Некоторые приложения, использующие определенные компоненты платформы Valtira, требуют намного больших объемов RAM. Так как Valtira фактически требует таких же объемов памяти на всех серверах кластера, добавление новых серверов совсем не помогает. Переход к использованию сервера с большими объемами RAM, однако, меняет дело.

Вертикальное масштабирование может помочь и при устранении других ограничений по мощностям. В табл. 7.1 описана предполагаемая реакция теоретического приложения на различные виды масштабирования.

Таблица 7.1. *Примеры выбора мощностей CPU для сервера Amazon*

Конфигурация	**Мощность**	**Стоимость**
Восемь серверов Amazon medium	8000 просмотров страниц в минуту	$0,80/час
Два сервера Amazon large	10 000 просмотров страниц в минуту	$0,80/час
Один сервер Amazon extra-large	10 000 просмотров страниц в минуту	$0,80/час

Если вы предполагаете линейное горизонтальное масштабирование, вы захотите перестроить инфраструктуру так, чтобы вместо восьми экземпляров `medium` использовать два экземпляра `large`. Понятно, что этот подход до абсурдности прост. Смысл, однако, заключается в том, что вертикальное масштабирование более разумно с финансовой точки зрения.

Вертикальное динамическое масштабирование сложнее, чем горизонтальное. Если быть более точным, то вертикальное масштабирование представляет собой

просто частный случай горизонтального. Оно требует следующей комбинации действий:

1. Добавление в облачную среду более мощной системы, используя точно такой же подход, как при горизонтальном масштабировании. Единственным отличием здесь будет то, что вы будете использовать более мощные экземпляры машин вместо дублирования существующих экземпляров.
2. Затем следует дождаться, когда новый экземпляр начнет отвечать на запросы.
3. После этого из системы можно будет удалить один или большее количество маломощных экземпляров.

Если вы будете комбинировать вертикальный и горизонтальный подходы к масштабированию, вы сможете получить инфраструктуру, наиболее эффективным образом потребляющую вычислительные ресурсы.

Заключение

В 2003 году я, если можно так выразиться, "прыгнул с утеса в холодные воды предпринимательства" — основал новую компанию, которую назвал Valtira (**http://www.valtira.com/**). Если говорить о функциях и задачах этой компании упрощенно (может быть, даже чересчур упрощенно), то Valtira занимается обслуживанием отделов маркетинга различных компаний примерно так же, как, например, SalesForce.com обслуживает отделы продаж. Valtira осуществляет управление проводимыми рекламными акциями (Campaign Management), проводит обслуживание команд CRM (Customer Relationship Management) в оперативном режиме (online), позволяя объединить маркетинговые программы с персонализированным информационным наполнением Web, а также предоставляет маркетологам множество других возможностей. Бизнес-модель Valtira отличается от аналогичной модели SalesForce.com одной ключевой особенностью: платформой, которая необходима вам, чтобы построить ваш Web-сайт, используя в качестве ядра уже имеющуюся у вас систему управления контентом (Content Management System, CMS).

Это требование к CMS сделало Valtira намного более мощным инструментом, нежели конкурирующая система управления маркетингом, известная под названием "программное обеспечение как услуга" (Software as a Service, SaaS). К сожалению, эта же особенность воздвигла и мощный барьер на пути тех, кто хотел бы воспользоваться решениями Valtira. Хотя многие компании приходят к выводу о том, что им необходимо прибегнуть к дорогостоящим сервисам интеграции CRM, предлагаемым SalesForce.com, вы можете начать развитие на их платформе, не начиная крупного проекта по интеграции. В отличие от этой позиции, Valtira, напротив, требовала начать крупный проект по Web-разработке для каждого из клиентов.

С 2007 года мы решили изменить наш подход и начали делать компоненты платформы Valtira доступными по требованию (on-demand). Иначе говоря, мы модифицировали наше программное обеспечение таким образом, чтобы сотрудники маркетинговых отделов компаний могли регистрироваться на Web-сайте Valtira и немедленно начинать создание окончательных страниц или персонализированных виджетов (widgets), которые подходили бы для их Web-сайтов.

Наше поставляемое по требованию приложение имело другой профиль риска (risk profile), отличный от других проектов развертывания, которыми мы управляли. Когда клиенты приступали к построению своих Web-сайтов на основе платформы Valtira Online Marketing, то они выбирали такую инфраструктуру, которая

соответствовала их требованиям к доступности их сервиса, и оплачивали эту инфраструктуру.

Если им требовалась инфраструктура с высоким уровнем доступности, они платили за управляемые сервисы высокой доступности ipHouse (**http://www.iphouse.com/**) или Rackspace и производили развертывание собственного ПО в рамках этой инфраструктуры. Если повышенный уровень доступности им не требовался, то мы предоставляли им инфраструктуру серверов общего доступа, которую они могли настраивать.

Профиль "по требованию" (on-demand profile) отличается в том, что любой клиент может ожидать, что запрашиваемый сервис "по требованию" (on-demand service) всегда будет доступен, вне зависимости от того, сколько он платит. Я составил расценки на приобретение стартовой среды высокой степени доступности для платформы Valtira, в состав которой входили следующие компоненты:

- балансировщик нагрузки элитного уровня (high-end);
- два сервера приложений с большим объемом RAM;
- два сервера баз данных с большим объемом дискового пространства;
- различные брандмауэры и коммутаторы (switches);
- дополнительный сервер форм-фактора "полустойка" (half-rack) у нашего интернет-провайдера (ISP).

Упомянул ли я, что Valtira является полностью самофинансируемым проектом, основанным на собственных фондах? Банковские кредиты, собственное управление компанией и стартовый капитал — все это собственный вклад семьи, и никаких других сторонних вложений для поддержки проекта не привлекалось. Все остальные средства, привлеченные в проект, брались из доходов, полученных в результате деятельности компании. Мы использовали прибыль для расширения бизнеса и при этом избегали каких бы то ни было излишеств. Мы всегда с большой осторожностью управляли нашими финансовыми потоками.

Я начал искать альтернативные варианты для построения собственной инфраструктуры и рассмотрел политику ценообразования нескольких поставщиков услуг. Хотя начальные расценки (up-front costs) были достаточно скромны, последующие затраты были достаточно высоки (по крайней мере, до тех пор, пока не будет достигнут определенный уровень продаж). Именно тогда я и начал экспериментировать с Web-сервисами Amazon (Amazon Web Services, AWS).

Со стороны AWS в наш адрес последовало предложение обеспечивать среду с относительно высоким уровнем доступности, которая, по грубым прикидкам, примерно соответствовала желаемой нами конфигурации, без начального взноса (up-front cash) и примерным уровнем ежемесячных затрат не выше $1000. Изначально я отнесся к этому весьма скептически. В принципе, это выглядело слишком хорошо, чтобы быть правдой. Но, тем не менее, я начал проводить исследовательскую работу....

Если бы вам захотелось просто убедиться в том, что ваше приложение будет надлежащим образом работать за элитным балансировщиком нагрузки (уровня high-end) на двух серверах приложений, то, положа руку на сердце, стали бы вы

просто покупать их, только для того, чтобы убедиться, что это действительно так? Я бы предположил, что ваш ответ будет отрицательным. Иными словами, даже если бы в результате я убедился в том, что облачные вычисления не подходят для бизнес-планов Valtira, ценность облачных технологий заключена именно в этой фразе: "Я начал исследования".

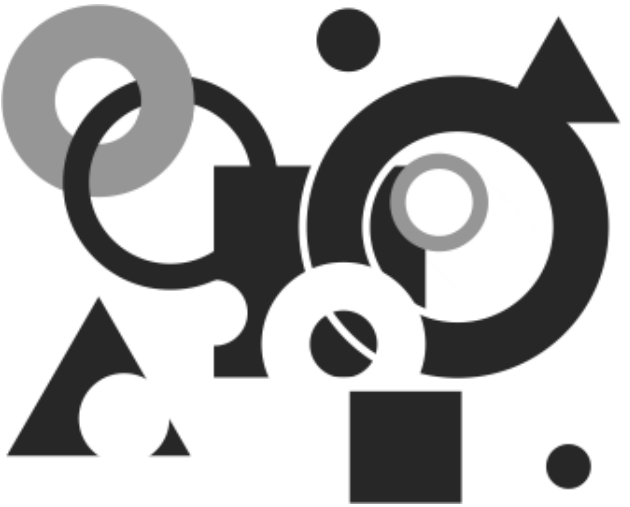
И я действительно столкнулся с проблемами. Сначала я обнаружил, как облачная инфраструктура Amazon управляет IP-адресами. Amazon присваивает все адреса динамически, вы не получаете никаких сетевых блоков (netblocks), и в те времена не было никаких опций для присваивания статических IP-адресов. Мы затратили некоторое время на поиски решения этой проблемы и выяснили, что имеем возможность предоставить для нее автоматическое решение. После этого наша команда начала работать над решением следующей проблемы.

А следующей проблемой было то, что сервис Amazon не предоставлял постоянного пространства на диске для хранения информации. Как и в случае со статическими IP-адресами, эта проблема тоже уже решена. Но до того, как в решение от Amazon были введены сервисы гибкого блочного хранилища (Elastic Block Storage), при закрытии вашего экземпляра EC2 вы теряли все данные. Если бы Valtira была крупной компанией с большим капиталом, мы бы сочли этот фактор решающим и начали бы искать другого партнера.

Мы даже почти решили так и поступить. В конце концов, платформа Valtira — это приложение, основывающееся на базах данных, а это значит, что никакие потери данных для нас недопустимы. Но мы разработали собственное решение, которое состояло из "ведомой" (slave) базы данных MySQL, которая синхронизировалась с Amazon S3. Для данного конкретного случая использования платформы Valtira это решение оказалось достаточно удачным. Мы обнаружили, что разработанное нами решение обладает возможностями автоматического аварийного восстановления.

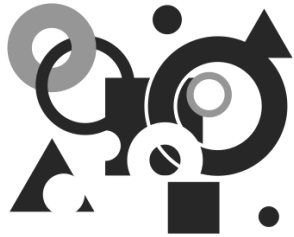
Наша экспериментальная работа продолжалась. Мы столкнулись еще с некоторыми проблемами, которые могли бы рассматриваться как основание для прекращения сотрудничества, но каждый раз выяснялось, что мы либо можем найти обходные варианты, либо создать собственное решение, лучшее, нежели предлагаемое. В конце концов, мы пришли к выводу, что все наши решения могут работать с облаком Amazon. Кроме того, в итоге мы выделили все решения, разработанные нами в ходе этих изысканий, в ведение отдельной компании, enStratus (**http://www.enstratus.com/**).

На сегодняшний день большая часть моего рабочего времени посвящена тому, чтобы перемещать другие компании в облако, построенное на основе программного обеспечения enStratus. Мои клиенты обычно гораздо больше озабочены аспектами конфиденциальности информации и вопросами безопасности, чем среднестатистический клиент, начинающий использовать облачные вычисления. Поэтому цель данной книги — помочь вам безболезненно осуществить переход на облачные вычисления и подготовить ваши Web-приложения к успешной работе в облачной инфраструктуре.



ПРИЛОЖЕНИЯ

Приложение 1



Справочная информация по Amazon Web Services

Доступ к Web-сервисам Amazon (Amazon Web Services, AWS) в основном осуществляется через Web-сервисы с помощью высокоуровневых командных строк и интерфейсов прикладного программирования (API), специфичных для выбранного языка программирования. В данном приложении описываются команды-"обертки" (wrappers) для упомянутых интерфейсов прикладного программирования.

Существует множество высокоуровневых абстракций, реализующих SOAP и API на различных языках программирования. Если вы создаете инструментарий для управления Amazon EC2 или Amazon S3, вам следует идентифицировать библиотеку, которая наилучшим образом подходит для использования с выбранным вами языком программирования и максимально удовлетворяет ваши функциональные потребности.

Справочная информация по командной строке Amazon EC2

Инструменты командной строки для управления Amazon EC2 представляют собой "обертки" вокруг функций API Web-сервисов. Фактически они устанавливают соответствие "один-к-одному" между командной строкой, ее аргументами и одноименным вызовом API с аналогичными параметрами[1].

Каждая команда имеет следующую общую форму:

```
command [GENERAL OPTIONS] [COMMAND OPTIONS]
```

[1] Приведенная в данном разделе информация представляет собой сокращенную версию полной документации по Web-сервисам Amazon. Полную версию документации, которую в любом случае необходимо прочесть, можно найти здесь: **http://aws.amazon.com/documentation/ec2/**, **http://docs.amazonwebservices.com/AWSEC2/latest/CommandLineReference/**, **http://developer.amazonwebservices.com/connect/forumindex.jspa**, **http://xgu.ru/wiki/Amazon_EC2**, **http://www.livebusiness.ru/news/8331/**. — *Прим. перев.*

Например, команда, запускающая экземпляр EC2, выглядит следующим образом:

```
ec2-run-instances -v ami-123456 -g dmz
```

В приведенном примере ключ `-v` представляет собой опцию, указывающую команде на необходимость подробного (`verbose`) вывода, а ключи `ami-1234566 -g dmz` являются опциями, специфичными для этой команды.

Для опций общего назначения или общих опций (`[GENERAL OPTIONS]`) допускаются следующие значения:

- `-`

  Дает указание команде извлекать параметры из потока стандартного пользовательского ввода (`stdin`).

- `-C certificate`

  Указывает цифровой сертификат (certificate)[1] для аутентификации ваших Web-сервисов в Amazon. Это значение имеет приоритет перед переменной окружения `EC2_CERT`.

- `--connection-timeout`

  Указывает значение тайм-аута в секундах для альтернативного соединения SOAP.

- `--debug`

  Выводит отладочную информацию.

- `--headers`

  Отображает информацию о заголовках столбцов.

- `--help`

  Выводит справочную информацию о той команде, которая была дана из командной строки.

- `-K privatekey`

  Указывает секретный ключ (private key) для аутентификации запроса ваших Web-сервисов к Amazon. Указанное вами значение имеет приоритет перед переменной окружения `EC2_PRIVATE_KEY`.

- `--region region`

  Указывает регион, применительно к которому действует команда.

- `--request-timeout`

  Указывает тайм-аут в секундах для альтернативного запроса SOAP.

- `--show-empty-fields`

  Отображает список пустых столбцов (`nil`) в ответе на запрос команды.

- `-U url`

---

[1] Цифровой сертификат (digital certificate) — выпущенный удостоверяющим центром сертификации электронный или печатный документ, подтверждающий принадлежность владельцу открытого ключа или каких-либо атрибутов. Сертификат открытого ключа содержит имя субъекта, открытый ключ, имя удостоверяющего центра, политику использования соответствующего удостоверяемому открытому ключу закрытого ключа и другие параметры, заверенные подписью удостоверяющего центра. Подробнее см. **http://en.wikipedia.org/wiki/Public_key_certificate**. — *Прим. перев.*

Указывает URL Web-сервисов Amazon, по которому необходимо обратиться с вызовом. Эта опция имеет приоритет перед переменной окружения `EC2_URL`.

- `-v`

  Указывает, что вы предпочитаете подробный (`verbose`) вывод. Подробный вывод отображает запросы и ответы SOAP.

Команда *ec2-add-group*

Эта команда использует следующий синтаксис:

```
ec2-add-group groupname -d description
```

Добавляет новую группу безопасности в вашу среду Amazon EC2. Указанное вами имя группы (*`groupname`*) будет идентифицировать данную группу для использования в других командах. Описание группы (*`description`*) — это простое описание группы в удобочитаемом формате, которое призвано помочь вам впоследствии быстро вспомнить предназначение группы.

Новая группа по умолчанию запрещает внешний доступ к экземплярам, запущенным в ее составе. Чтобы разрешить перенаправление трафика в эту группу, необходимо дать команду `ec2-authorize`.

Пример

```
$ ec2-add-group mydmz -d DMZ GROUP mydmz DMZ
Group mydmz DMZ
```

Команда *ec2-add-keypair*

Данная команда использует следующий синтаксис:

```
ec2-add-keypair keyname
```

Команда создает новую пару 2048-битных ключей шифрования RSA[1]. Имя, указанное вами при создании ключа (*`keyname`*), будет применяться для ссылок на этот ключ при использовании других команд или вызовов API Web-сервисов EC2. Команда сохраняет открытый ключ в Amazon и передает секретный ключ на поток стандартного вывода (`stdout`). После этого вы должны будете надежно сохранить секретный ключ для дальнейшего его применения для доступа к экземплярам, которые вы запускаете с помощью этой пары ключей.

---

[1] RSA (буквенное сокращение от фамилий Rivest, Shamir и Adleman) — криптографический алгоритм с открытым ключом.
Подробнее см. **http://ru.wikipedia.org/wiki/RSA**, **http://en.wikipedia.org/wiki/RSA**, **http://www.di-mgt.com.au/rsa_alg.html**, **http://www.muppetlabs.com/~breadbox/txt/rsa.html**, **http://cryptool.org/download/RSA/RSA-Flash-en/player.html**. — *Прим. перев.*

Пример

```
$ ec2-add-keypair georgekey
KEYPAIR georgekey 2e:82:bb:91:ca:51:22:e1:1a:84:c8:19:db:7c:8b:ad:f9:5e:27:3e
-----BEGIN RSA PRIVATE KEY----MIIEowIBAAKCAQEAkot9/
09tIy5FJSh0X5vLGCu3so5Q4qG7cU/MBm45c4EVFtMDpU1VpAQi1vn9
r7hr5kLr+ido1d1eBmCkRkHuyhfviJmH1FTOWm6JBhfOsOgDU0pInyQOP0nRFLx4eyJfYsiK/mUm
hiYC9Q6VnePjMUiHSahOL95C8ndAFBlUAuMDDrXMhLypOGRuWkJo+xtlVdisKjlOT0l33q3VSeT6
NBmZwymWOguGWgKWMpzpDLhV9jhDhZgaZmGUKP0wPQqdV6psA9PuStN1LJkhWVYuQTqH9UUolvJn
ZXx5yE2CSpPW+8zMb4/xUuweBQ6grw8O3IxhKWbFCpGGhkpk5BB+MQIDAQABAoIBAQCIs6U6mA4X
5l7MFdvRIFSpXIbFAutDLlnbjvOlAAeJzt0saHWbKvP7x3v0jElxNRk6OC1HMqIh9plyW46Cl5i4
XvGsvIOvt9izFS+vRmAiOJx5gu8RvSGpOiPXMyU0wFC4ppi6TQNN2oGhthQtsFrMK3tAY8dj8fMD
mehll2b+NPZRWPp9frm3QtwLIOMeWm1ntknCVSjBqj21XRg3UPbE8r8ISlSGryqJBA0KjnOj+cMN
2SBx8iC+BHxD9xSUvXs4hVjUpQofzd+8BAZbsXswj+/ybuq1GlNwzpUKKEfH1rN3TZztywN5Z9Hb
EbkOtgRYi/2htSpbuDq5b/cTaxIRAoGBAOLRgfZhEwnGQvveMOhRLLko1D8kGVHP6KCwlYiNow07
G8FkP6U3wcUrsCTtvOFB/79FeWVT+o7v25D34cYFtGbfnp3Zh9bdTBi18PbIHQHvD4tIAIF+4PcO
XMRsJCrzhChOLY1G/laMi5EKFcx6RU8Pjup92YbEbi/fkybcrmS9AoGBAKVmGI5PV00A10LQkTov
CnLuyfAPL9s8w6eOy+9WMRd8+27tI6H8yGuEdsF9X9bOJQsnTM3+A+RC8ylVXWPgiflbcpbPsZ8a
HVuTg37D/iFpl42RzrMtzhgLCahvNotirNyAYnklBsOlmtsQdJSJ0GPpv4SOloSoPT+jbP4ONUiF
AoGAWU48aZHXOSYDAcB+aTps7YqR5zqDbZ767SoZ9mYuKOt5BjA+jwLhHI0TEbc5g0fFNr5YCfmC
0fzG6tFu59UfLtIlVelsfsErUR9x/PjV0wkZibGT4Wjfkubox738j5zKEESX0uR9B/7WhQj/hD8w
QuzRTKq4l0OITvksq0SAtdECgYAqpr1GVWdp0AGylR4eJutG4BTq9r+chXrexpAIU+2s5OnhnP1H
VGxKbYpCMxZ3ygj7a1L++7X9MtaJnh3LF6f8yXwvL7faE13ms4+BLQFnlFckhqkKw5EV2iLPcH5c
S0HQSrsaClZINXhNbVziwPcgDLL6d9qQsuG4e2gry3YqEQKBgFHqE4UJCOd5WiAG0N0cYDTF/wh6
iujW5tY90F63xAn2B236DGE+8o2wGwU77u59LO7jyx4WyR8TpcorL79zZuzm0Vjn9nslAu7tkS6O
wmdEM0O2LrGnKGydSdRF50NH8Tgb060txh+hWyWtvkP0tSOZyGu1z7S7JS0ZPX42Arm8
-----END RSA PRIVATE KEY----
```

Команда *ec2-allocate-address*

Данная команда использует следующий синтаксис:

```
ec2-allocate-address
```

Команда выделяет новый эластичный IP-адрес Amazon и печатает выделенный IP-адрес. После этого данный IP-адрес назначается вам и становится доступным только вам. Вы можете по собственному усмотрению назначать этот адрес своим экземплярам. Amazon взимает плату за неиспользование эластичных IP-адресов, точно так же, как и за избыточное переназначение адресов.

Пример

```
$ ec2-allocate-address
ADDRESS 67.202.55.255
```

Команда *ec2-associate-address*

Данная команда использует следующий синтаксис:

```
ec2-associate-address -i instanceid ipaddress
```

Команда ассоциирует адрес (*ipaddress*), полученный с помощью команды `ec2-allocate-address`, с работающим экземпляром EC2 (*instanceid*). Эта же команда используется и для снятия любых действующих ассоциаций IP-адреса, которые он может иметь на текущий момент. Поэтому очень важно, чтобы вы помнили назначение IP-адресов, чтобы случайно не разрушить установленные ассоциации (за исключением тех случаев, когда вы действительно намерены выполнить переназначение IP-адресов).

Пример

```
$ ec2-associate-address -i i-12b3ff6a 67.202.55.255
ADDRESS 67.202.55.255 i-12b3ff6a
```

Команда *ec2-attach-volume*

Эта команда использует следующий синтаксис:

```
ec2-attach-volume volumeid -i instanceid -d device
```

Команда прикрепляет существующий блочный том (*volumeid*) к работающему экземпляру EC2 (*instanceid*), предоставляя экземпляру доступ к нему как к указанному устройству (*device*). Корректное имя устройства зависит от платформы, при этом различные варианты Linux принимают имена устройств наподобие `/dev/sdh`, а операционные системы из семейства Windows ожидают имена устройств, такие как `xvdh`.

Когда том блочного хранения будет присоединен к вашему экземпляру, вы сможете примонтировать его и отформатировать с помощью утилит управления дисками, встроенных в выбранные вами операционные системы.

Вывод команды описывает состояние прикрепленного тома и выводит сопутствующую информацию.

Пример

```
$ ec2-attach-volume vol-81aeb37f -i i-12b3ff6a -d /dev/sdf
ATTACHMENT vol-81aeb37f i-12b3ff6a /dev/sdf attaching 2008-12-17T22:36:00+0000
```

Команда *ec2-authorize*

Данная команда использует следующий синтаксис:

```
ec2-authorize groupname -P protocol (-p portrange
  | -t icmptypecode) [-u sourceuser ...] [-o sourcegroup ...]
  [-s sourceaddress]
```

Авторизует входящий сетевой трафик к экземплярам EC2, запущенным под указанным именем группы (*groupname*).

Вы можете авторизовать входящий трафик, руководствуясь различными критериями:

- на основании подсети, из которой исходит этот трафик;
- исходя из членства в группах того экземпляра EC2, из которого исходит трафик (если трафик исходит из экземпляра EC2);
- исходя из сетевого протокола (TCP, UDP, ICMP), используемого трафиком (*protocol*);
- исходя из целевого порта данного трафика (*portrange*).

По умолчанию группа не допускает никакого входящего трафика, поступающего на экземпляры EC2, входящие в состав этой группы (хотя члены этой группы и могут принадлежать не только к ней, но и к другим группам, членство в которых может допускать входящий трафик). Чтобы допустить входящий трафик, вы должны отдельно авторизовать его прохождение.

За исключением тех случаев, когда вы хотите разрешить входящий трафик из одной группы EC2 в другую, вы можете контролировать трафик вплоть до уровней протокола и порта. Если вы хотите разрешить прохождение трафика из одной группы в другую, то здесь вы имеете возможность, либо разрешить все, либо не разрешить ничего (all-or-nothing proposition).

Примеры

```
# Предоставляет доступ через порт 80 для всего входящего трафика,
# независимо от источника
$ ec2-authorize mydmz -P tcp -p 80 -s 0.0.0.0/0
GROUP mydmz
PERMISSION mydmz ALLOWS tcp 80 80 FROM CIDR 0.0.0.0/0

# Предоставляет доступ к группе серверов приложений из группы DMZ
$ ec2-authorize myapp -u 999999999999 -o mydmz
GROUP myapp
PERMISSION myapp ALLOWS all FROM USER 999999999999 GRPNAME mydmz

# Предоставляет доступ к диапазону портов с указанного IP-адреса
$ ec2-authorize mydmz -P udp -p 3000-4000 -s 67.202.55.255/32
GROUP mydmz
PERMISSION mydmz ALLOWS udp 3000 4000 FROM CIDR 67.202.55.255/32
```

Команда *ec2-bundle-instance*

Данная команда использует следующий синтаксис:

```
ec2-bundle-instance instanceid -b s3bucket -p prefix -o accesskey (-c
    policy | -w secretkey)
```

Эта команда действует только применительно к экземплярам Windows. Комплектует и упаковывает ваш экземпляр Windows и сохраняет его в Amazon S3 для регистрации с помощью команды `ec2-register`.

Пример

```
$ ec2-bundle-instance i-12b3ff6a -b mybucket -p myami -o 999999999999 -w
lY1zp/1iKzSAg9B04lQ0T3gMxje7IfnXtN5asrM/dy==
BUNDLE bun-abd5209d8 i-12b3ff6a mybucket myami pending 2008-1218T13:
08:18+0000 2008-12-18T13:08:18+0000
```

Команда *ec2-cancel-bundle-task*

Эта команда использует следующий синтаксис:

```
ec2-cancel-bundle-task bundleid
```

Данная команда действует только применительно к экземплярам Windows. Она отменяет текущий незавершенный процесс комплектации и упаковки.

Пример

```
$ ec2-cancel-bundle-task bun-abd5209d8
BUNDLE bun-abd5209d8 i-12b3ff6a mybucket myami canceling 2008-1218T13:
13:29+0000 2008-23-18T13:13:29+0000
```

Команда *ec2-confirm-product-instance*

Данная команда использует следующий синтаксис:

```
ec2-confirm-product-instance productcode -i instanceid
```

Позволяет владельцу образа AMI проверить, присоединен ли к указанному экземпляру (*instanceid*) именованный код продукта (*productcode*).

Пример

```
$ ec2-confirm-product-instance zt1 -i i-12b3ff6a
ztl i-12b3ff6a false
```

Команда *ec2-create-snapshot*

Данная команда использует следующий синтаксис:

```
ec2-create-snapshot volumeid
```

Создает дифференциальный моментальный снимок (snapshot) указанного тома (volume ID) и сохраняет этот моментальный снимок в Amazon S3. Наилучший под-

ход заключается в создании моментальных снимков "замороженных" (frozen) файловых систем, что позволяет избежать проблем с нарушением целостности данных. После того как данная команда вернет код успешного завершения, на этот том снова можно будет вести запись.

Пример

```
$ ec2-create-snapshot vol-12345678
SNAPSHOT snap-a5d8ef77 vol-12345678 pending 2008-12-20T20:47:23+0000
```

Команда *ec2-create-volume*

Данная команда использует следующий синтаксис:

```
ec2-create-volume (-s size | --snapshot snapshotid) -z zone
```

Создает в указанной зоне доступности (*zone*) новый том указанного размера (*size*) или же новый том на основании существующего моментального снимка (*snapshotid*). Вам необходимо указать зону доступности, в которой требуется создать новый том, а также задать его размер или имя существующего моментального снимка. Размер тома должен указываться в гигабайтах.

Примеры

```
# Создать новый том размером 10 Гбайт
$ ec2-create-volume -s 10 -z eu-west-1a
VOLUME vol-12345678 10 eu-west-1a creating 2008-12-20T20:47:23+0000

# Создать новый том на основании сохраненного моментального снимка
$ ec2-create-volume --snapshot snap-a5d8ef77 -z eu-west-1a
VOLUME vol-12345678 10 eu-west-1a creating 2008-12-20T20:47:23+0000
```

Команда *ec2-delete-group*

Команда использует следующий синтаксис:

```
ec2-delete-group group
```

Команда удаляет указанную группу безопасности (*group*) из состава вашей учетной записи. Вы не сможете удалить группу до тех пор, пока не будут удалены все ссылки на нее (экземпляры в составе группы и другие группы, в составе которых имеются правила, разрешающие доступ к удаляемой группе).

Пример

```
$ ec2-delete-group mydmz
GROUP mydmz
```

Команда *ec2-delete-keypair*

Команда использует следующий синтаксис:

```
ec2-delete-keypair keypair
```

Команда удаляет указанный открытый ключ, ассоциированный с именованной парой ключей (*keypair*), из вашей учетной записи Amazon.

Пример

```
$ ec2-delete-keypair georgekey
KEYPAIR georgekey
```

Команда *ec2-delete-snapshot*

Команда использует следующий синтаксис:

```
ec2-delete-snapshot snapshotid
```

Удаляет из вашей учетной записи указанный моментальный снимок (*snapshotid*).

Пример

```
$ ec2-delete-snapshot snap-a5d8ef77
SNAPSHOT snap-a5d8ef77
```

Команда *ec2-delete-volume*

Данная команда использует следующий синтаксис:

```
ec2-delete-volume volumeid
```

Команда удаляет указанный том (*volumeid*) из вашей учетной записи.

Пример

```
$ ec2-delete-volume vol-12345678
VOLUME vol-12345678
```

Команда *ec2-deregister*

Команда использует следующий синтаксис:

```
ec2-deregister imageid
```

Отменяет регистрацию образа машины (*imageid*). После аннулирования регистрации вы больше не сможете запускать с него экземпляры. Чтобы освободить пространство для хранения данных, вам нужно будет отдельно удалить AMI из хранилища S3.

Пример

```
$ ec2-deregister ami-f822a39b
IMAGE ami-f822a39b
```

Команда *ec2-describe-addresses*

Эта команда использует следующий синтаксис:

```
ec2-describe-addresses [ipaddres1 [...ipaddressN]]
```

Команда выводит сведения, ассоциированные с указанными эластичными IP-адресами (*[ipaddres1 [...ipaddressN]]*). Если вы не укажете никаких конкретных адресов, команда выведет все IP-адреса, которые выделены вам в текущем регионе EC2.

Примеры

```
# ПОКАЗАТЬ ВСЕ ВЫДЕЛЕННЫЕ АДРЕСА
$ ec2-describe-addresses
ADDRESS 67.202.55.255 i-12b3ff6a
ADDRESS 67.203.55.255

# ВЫВЕСТИ ИНФОРМАЦИЮ О КОНКРЕТНОМ ВЫДЕЛЕННОМ АДРЕСЕ
$ ec2-describe-addresses 67.202.55.255
ADDRESS 67.202.55.255 i-12b3ff6a
```

Команда *ec2-describe-availability-zones*

Данная команда использует следующий синтаксис:

```
ec2-describe-availability-zones [zone1 [...zoneN]]
```

Команда выводит информацию, ассоциированную с конкретными зонами доступности EC2 (*[zone1 [...zoneN]]*). Если не указано никаких конкретных зон, команда выведет информацию обо всех зонах доступности, ассоциированных с текущим регионом EC2.

Примеры

```
# ВЫВЕСТИ ИНФОРМАЦИЮ ОБО ВСЕХ ЗОНАХ ДОСТУПНОСТИ
$ ec2-describe-availability-zones
AVAILABILITYZONE us-east-1a available
AVAILABILITYZONE us-east-1b available
AVAILABILITYZONE us-east-1c available
```

```
# ВЫВЕСТИ ИНФОРМАЦИЮ О КОНКРЕТНОЙ ЗОНЕ ДОСТУПНОСТИ
$ ec2-describe-availabilty-zones us-east-1a
AVAILABILITYZONE us-east-1a available

# ВЫВЕСТИ ИНФОРМАЦИЮ ОБО ВСЕХ ЗОНАХ ДОСТУПНОСТИ НА ТЕРРИТОРИИ ЕВРОСОЮЗА
$ ec2-describe-availability-zones --region eu-west-1
AVAILABILITYZONE eu-west-1a available
AVAILABILITYZONE eu-west-1b available
```

Команда *ec2-describe-bundle-tasks*

Данная команда использует следующий синтаксис:

```
ec2-describe-bundle-tasks [bundle1 [...bundleN]]
```

Эта команда действует только применительно к экземплярам Windows. Она выводит информацию, ассоциированную с задачами указанного пакета (bundle). Если не указано никакого конкретного имени задачи, команда выведет все пакетные задачи, ассоциированные с данной учетной записью.

Примеры

```
# ВЫВЕСТИ ИНФОРМАЦИЮ ОБО ВСЕХ ЗАДАЧАХ
$ ec2-describe-bundle-tasks
BUNDLE bun-abd5209d8 i-12b3ff6a mybucket myami pending 2008-1218T13:
08:18+0000 2008-12-18T13:08:18+0000
BUNDLE bun-abd5209d9 i-12b3ff7a mybucket myami pending 2008-1218T13:
08:18+0000 2008-12-18T13:08:18+0000

# ВЫВЕСТИ ИНФОРМАЦИЮ О КОНКРЕТНОЙ ЗАДАЧЕ
$ ec2-describe-bundle-tasks bun-abd5209d8
BUNDLE bun-abd5209d8 i-12b3ff6a mybucket myami pending 2008-1218T13:
08:18+0000 2008-12-18T13:08:18+0000
```

Команда ec2-*describe-group*

Данная команда использует следующий синтаксис:

```
ec2-describe-group [group1 [...groupN]]
```

Команда выводит информацию, ассоциированную с указанными группами безопасности (`[group1 [...groupN]]`). Если не указаны имена конкретных групп, команда выведет все группы, ассоциированные с данной учетной записью.

Примеры

```
# Вывести информацию обо всех группах безопасности
$ ec2-describe-group
GROUP mydmz DMZ
PERMISSION mydmz ALLOWS tcp 80 80 FROM CIDR 0.0.0.0/0
GROUP myapp App
PERMISSION myapp ALLOWS all FROM USER 999999999999 GRPNAME mydmz

# Вывести информацию о конкретной группе безопасности
$ ec2-describe-group mydmz
PERMISSION mydmz ALLOWS tcp 80 80 FROM CIDR 0.0.0.0/0
```

Команда *ec2-describe-image-attribute*

Данная команда использует следующий синтаксис:

```
ec2-describe-image-attribute imageid (-l | -p)
```

Описывает атрибуты указанного AMI (*imageid*). Вы можете указать, хотите ли вы просмотреть полномочия запуска (launch permissions, опция `-l`) или коды продукта (product codes, опция `-p`).

Примеры

```
# Вывести полномочия запуска
$ ec2-describe-image-attribute ami-f822a39b -l
launchPermission ami-f822a39b userId 999999999999

# Вывести код продукта
$ ec2-describe-image-attribute ami-f822a39b -p
productCodes ami-f822a39b productCode zz95xy
```

Команда *ec2-describe-images*

Данная команда использует следующий синтаксис:

```
ec2-describe-images [imageid1 [...imageidN]] [-a] [-o ownerid] [-x ownerid]
```

Команда выводит информацию, ассоциированную с конкретным идентификатором образа (image ID) или любыми образами, которые имеют совпадения с указанными параметрами ([*imageid1* [...*imageidN*]]). Если вы не укажете никаких конкретных параметров, будут перечислены все образы, которые принадлежат вашей учетной записи.

Команда может использоваться со следующими опциями командной строки:

- `-a`

  Выводит список всех AMI, на которые запустивший ее пользователь имеет право запуска.

- `-o` *`ownerid`*

  Выводит список всех AMI, которые принадлежат указанному владельцу (*`ownerid`*) или указанным владельцам. Кроме того, вы можете использовать специальные идентификаторы владельцев: `amazon` (для общедоступных образов), `self` (для образов, принадлежащих вам) и `explicit` (относится ко всем образам, на которые у вас есть права запуска).

- `-x` *`ownerid`*

  Выводит список AMI, на которые указанный владелец (*`ownerid`*) имеет права запуска. В дополнение к стандартному идентификатору владельца (owner ID) вы можете указать себя (`self`), чтобы получить доступ к тем образам, которые вы имеете право запускать, или всех (`all`), чтобы вывести список всех AMI, которые могут быть запущены любым пользователем.

  Особенно полезен следующий вариант команды, который позволяет быстро начать работу:

```
ec2-describe-images -o amazon
```

Примеры

```
# Вывести список всех образов
$ ec2-describe-images
IMAGE ami-f822a39b myami/myami.manifest.xml 999999999999
zz95xy i386 machine aki-a71cf9ce ari-a51cf9cc available private

# Вывести список образов для конкретного пользователя
$ ec2-describe-images -o 063491364108
IMAGE ami-48de3b21 level22-ec2-images-64/ubuntu-7.10-gutsy-base-
64-20071203a.manifest.xml 063491364108 available public x86_64 machine
IMAGE ami-dd22c7b4 level22-ec2-images-64/ubuntu-7.10-gutsy-
64-20071227a.manifest.xml 063491364108 available public x86_64 machine
```

Команда *ec2-describe-instances*

Данная команда использует следующий синтаксис

```
ec2-describe-instances [instanceid1 [...instanceidN]]
```

Команда выводит информацию, ассоциированную с указанными экземплярами ([*`instanceid1`* [*`...instanceidN`*]]). Если вы не указали ни одного конкретного экземпляра, то команда выведет информацию обо всех экземплярах, ассоциированных с учетной записью.

Примеры

```
# Вывести информацию обо всех экземплярах
$ ec2-describe-instances
RESERVATION r-3d01de54 999999999999 default
```

```
INSTANCE i-b1a21bd8 ami-1fd73376 pending 0 m1.small 2008-1022T16:
10:38+0000 us-east-1a aki-a72cf9ce ari-a52cf9cc
RESERVATION r-3d01cc99 999999999999 default
INSTANCE i-ccdd1b22 ami-1fd73376 pending 0 m1.small 2008-1022T16:
10:38+0000 us-east-1a aki-a72cf9ce ari-a52cf9cc

# Вывести информацию о конкретном экземпляре
$ ec2-describe-instances i-b1a21bd8
RESERVATION r-3d01de54 999999999999 default
INSTANCE i-b1a21bd8 ami-1fd73376 pending 0 m1.small 2008-1022T16:
10:38+0000 us-east-1a aki-a72cf9ce ari-a52cf9cc
```

Команда *ec2-describe-keypairs*

Данная команда использует следующий синтаксис:

```
ec2-describe-keypairs [keypairid1 [...keypairidN]]
```

Выводит информацию, ассоциированную с указанной парой ключей (*keypairid*). Если никакой конкретной пары ключей не указано, команда перечислит все ключи, которыми вы владеете.

Пример

```
$ ec2-describe-keypairs
KEYPAIR georgekey 98:21:ff:2a:6b:35:71:6e:1f:36:d9:f2:2f:d7:aa:e4:14:bb:1d:1a
```

Команда ec2-describe-regions

Данная команда использует следующий синтаксис:

```
ec2-describe-regions [region1 [...regionN]]
```

Команда выводит информацию об указанных регионах ([*region1* [...*regionN*]]). Если не указано ни одного конкретного региона, то команда выведет список всех регионов.

Пример

```
$ ec2-describe-regions
REGION eu-west-1 eu-west-1.ec2.amazonaws.com
REGION us-east-1 us-east-1.ec2.amazonaws.com
```

Команда *ec2-describe-snapshots*

Данная команда использует следующий синтаксис:

```
ec2-describe-snapshots [snapshotid1 [...snapshotidN]]
```

Команда выводит информацию, ассоциированную с конкретным моментальным снимком или моментальными снимками (`[snapshotid1 [...snapshotidN]]`). Если никакого конкретного моментального снимка не указано, команда выведет все моментальные снимки, ассоциированные с учетной записью.

Пример

```
$ ec2-describe-snapshots
SNAPSHOT snap-a5d8ef77 vol-12345678 pending 2008-12-20T20:47:23+0000 50%
```

Команда *ec2-describe-volumes*

Данная команда использует следующий синтаксис:

```
ec2-describe-volumes [volumeid1 [...volumeidN]]
```

Команда выводит информацию, ассоциированную с указанным томом или указанными томами (`[volumeid1 [...volumeidN]]`). Если не указано ни одного конкретного тома, то команда отобразит все тома, ассоциированные с учетной записью.

Пример

```
$ ec2-describe-volumes
VOLUME vol-81aeb37f 5 snapa5d8ef77 us-east-1a in-use 2008-12-17T22:36:00+0000
ATTACHMENT vol-81aeb37f i-12b3ff6a /dev/sdf attached 2008-12-17T22:36:00+0000
```

Команда *ec2-detach-volume*

Эта команда использует следующий синтаксис:

```
ec2-detach-volume volumeid [-i instanceid] [-d device] --force
```

Открепляет указанный том (*volumeid*) от экземпляра (*instanceid*), к которому он прикреплен на данный момент. Перед тем как открепять том от экземпляра, необходимо убедиться в том, что вы предварительно отмонтировали от экземпляра файловую систему открепляемого тома; в противном случае ваши данные, скорее всего, окажутся поврежденными. Если процедура открепления завершится неудачей, вы можете попытаться повторить ее, дав команду с опцией `--force`, чтобы выполнить открепление в принудительном порядке.

Пример

```
$ ec2-detach-volume
ATTACHMENT vol-81aeb37f i-12b3ff6a /dev/sdf detaching 2008-12-17T22:36:00+0000
```

Команда *ec2-disassociate-address*

Данная команда использует следующий синтаксис:

```
ec2-disassociate-address idaddress
```

Команда снимает ассоциацию указанного эластичного IP-адреса (*idaddress*) с любого экземпляра, с которым этот адрес может быть ассоциирован на текущий момент.

Пример

```
$ ec2-disassociate-address 67.202.55.255
ADDRESS 67.202.55.255
```

Команда *ec2-get-console-output*

Данная команда использует следующий синтаксис:

```
ec2-get-console-output instanceid [-r]
```

Команда отображает консольный вывод при запуске экземпляра (*instanceid*). Если команда дается с опцией `-r` (`raw`), то вывод будет отображаться без форматирования.

Пример

```
$ ec2-get-console-output i-b1a21bd8
i-b1a21bd8
2008-12-23T20:03:07+0000
Linux version 2.6.21.7-2.fc8xen (mockbuild@xenbuilder1.fedora.redhat.com) (gcc
version 4.1.2 20070925 (Red Hat 4.1.2-33)) #1 SMP Fri Feb 15 12:39:36 EST 2008
BIOS-provided physical RAM map:
sanitize start
sanitize bail 0
copy_e820_map() start: 0000000000000000 size: 000000006ac00000 end:
000000006ac00000 type: 1
 Xen: 0000000000000000 - 000000006ac00000 (usable)
980MB HIGHMEM available.
727MB LOWMEM available.
NX (Execute Disable) protection: active
Zone PFN ranges:
  DMA             0         -> 186366
  Normal     186366         -> 186366
  HighMem    186366         -> 437248
early_node_map[1] active       PFN ranges
     0:        0             -> 437248
```

```
ACPI in unprivileged domain disabled
Detected 2600.043 MHz processor.
Built 1 zonelists. Total pages: 433833
Kernel command line: root=/dev/sda1 ro 4
Enabling fast FPU save and restore... done.
Enabling unmasked SIMD FPU exception support... done.
Initializing CPU#0
CPU 0 irqstacks, hard=c136c000 soft=c134c000
PID hash table entries: 4096 (order: 12, 16384 bytes)
Xen reported: 2600.000 MHz processor.
Console: colour dummy device 80x25
Dentry cache hash table entries: 131072 (order: 7, 524288 bytes)
Inode-cache hash table entries: 65536 (order: 6, 262144 bytes)
Software IO TLB disabled
vmalloc area: ee000000-f4ffe000, maxmem 2d7fe000
Memory: 1711020k/1748992k available (2071k kernel code, 28636k reserved, 1080k
data, 188k init, 1003528k highmem)
```

Команда *ec2-get-password*

Данная команда использует следующий синтаксис:

```
ec2-get-password instanceid -k keypair
```

Эта команда действует только применительно к экземплярам Windows. Предоставляет пароль администратора для запущенного Windows-экземпляра (*instanceid*) на основе пары ключей (*keypair*), использованных для запуска этого экземпляра. SOAP-версии этой команды не существует.

Пример

```
$ ec2-get-password i-b1a21bd8 -k georgekey
sZn7h4Dp8
```

Команда *ec2-modify-image-attribute*

Данная команда использует следующие варианты синтаксиса:

```
ec2-modify-image-attribute imageid -l -a value
ec2-modify-image-attribute imageid -l -r value
ec2-modify-image-attribute imageid -p productcode [-p productcode]
```

Команда модифицирует атрибуты для указанного образа (*imageid*). Опция `-l` указывает атрибуты прав на запуск экземпляра, а опция `-p` указывает коды продукта.

Примеры

```
# Добавить права доступа
$ ec2-modify-image-attribute ami-f822a39b -l -a 123456789
launchPermission ami-f822a39b ADD userId 123456789

# Отобрать права доступа
$ ec2-modify-image-attribute ami-f822a39b -l -r 123456789
launchPermission ami-f822a39b REMOVE userId 123456789

# Добавить код продукта
$ ec2-modify-image-attribute ami-f822a39b -p crm114
productCodes ami-f822a39b productCode crm114
```

Команда *ec2-reboot-instances*

Эта команда использует следующий синтаксис:

```
ec2-reboot-instances instanceid1 [...instanceidN]
```

Команда перезагружает экземпляры, указанные в командной строке (*instanceid1* [...*instanceidN*]). Она ничего не выводит, за исключением кодов ошибок, если ошибки имеют место.

Команда *ec2-release-address*

Данная команда использует следующий синтаксис:

```
ec2-release-address ipaddress
```

Команда освобождает IP-адрес (*ipaddress*), выделенный вам на текущий момент. После исполнения этой команды вы уже не сможете вернуть себе освобожденный адрес.

Пример

```
$ ec2-release-address 67.202.55.255
ADDRESS 67.202.55.255
```

Команда *ec2-register*

Данная команда использует следующий синтаксис:

```
ec2-register s3manifest
```

Команда регистрирует образ машины, файл манифеста которого находится по указанному пути (*s3manifest*).

Пример

```
$ ec2-register myami/myami.manifest.xml
IMAGE ami-f822a39b
```

Команда *ec2-reset-image-attribute*

Данная команда использует следующий синтаксис:

```
ec2-reset-image-attribute imageid -l
```

Сбрасывает атрибут прав на запуск образа для указанного образа машины (*imageid*).

Пример

```
$ ec2-reset-image-attribute ami-f822a39b -l
launchPermission ami-f822a39b RESET
```

Команда *ec2-revoke*

Эта команда использует следующий синтаксис:

```
ec2-revoke groupname [-P protocol] (-p portrange | -t icmptypecode)
[-u sourceuser ...] [-o sourcegroup ...] [-s sourceaddress]
```

Команда отзывает первоначальную авторизацию для указанной группы безопасности (*groupname*). Опции командной строки представляют собой те самые опции, которые вы использовали при установке полномочий с помощью команды `ec2-authorize`. Для получения более подробной информации см. информацию о команде `ec2-authorize`.

Пример

```
$ ec2-revoke -P tcp -p 80 -s 0.0.0.0/0
GROUP mydmz
PERMISSION mydmz ALLOWS tcp 80 80 FROM CIDR 0.0.0.0/0
```

Команда *ec2-run-instances*

Данная команда использует следующий синтаксис:

```
ec2-run-instances imageid [-n count] [-g groupname1 [... -g groupnameN]]
[-k keypair] -d customdata | -f customfile] [ -t type] [ -z zone]
[ --kernel kernelid] [ --ramdisk ramdiskid] [ -B devicemapping]
```

Команда предпринимает попытку запустить один или несколько экземпляров EC2 на основе выбранного AMI и опций, указанных в командной строке.

Команда использует следующие опции:

- `-B` *`devicemapping`*

  Определяет, как блочные устройства представляются запускаемым экземплярам. Вы можете указать ряд различных виртуальных имен:

  - `ami` — корневая файловая система в том виде, в котором ее "видит" экземпляр;
  - `root` — корневая файловая система, как ее "видит" ядро;
  - `swap` — устройство своппинга (swap device), как его "видит" экземпляр;
  - `ephemeral`*`N`* — *N*-е эфемерное устройство хранения.

- `-d` *`customdata`*

  Данные, которые должны быть доступны вашему экземпляру во время исполнения. Если вам требуется предоставить экземпляру большой объем данных, сохраните эти данные в файле и воспользуйтесь опцией командной строки `-f`.

- `-f` *`customfile`*

  Имя файла, содержащего данные времени исполнения, которые должны быть доступны вашим экземплярам после запуска.

- `-g` *`groupname`*

  Имя группы безопасности (*`groupname`*), правила которой управляют запускаемыми экземплярами. Вы можете указать несколько групп безопасности, тогда доступом к экземпляру (экземплярам) будет управлять объединение (union) прав и привилегий, ассоциированных с указанными группами.

- `-k` *`keypair`*

  Открытый ключ, передаваемый экземпляру EC2 при запуске.

- `--kernel` *`kernelid`*

  Идентификатор ядра (kernel ID), с которым запускаются экземпляры.

- `-n` *`count`*

  Минимальное количество экземпляров (*`count`*), запускаемое командой. Если EC2 не может запустить указанное минимальное количество экземпляров, то команда не запустит ни одного.

- `--ramdisk` *`ramdiskid`*

  Идентификатор RAM-диска (RAM disk ID), используемый для запуска экземпляра.

- `-t` *`type`*

  Тип экземпляра Amazon, который определяет ресурсы (CPU, RAM и другие атрибуты), выделяемые запускаемому экземпляру или экземплярам. На момент написания этой книги допустимыми значениями были: `m1.small`, `m1.large`, `m1.xlarge`, `c1.medium` и `c1.xlarge`.

- `-z` *`zone`*

  Зона доступности (*`zone`*), в которой должны запускаться ваши экземпляры. Если вы не укажете никакого конкретного значения, то EC2 запустит их в той зоне доступности, которая на момент запуска считается наилучшей.

Примеры

```
# Запустить в точности один экземпляр в любой зоне доступности
$ ec2-run-instances ami-f822a39b
RESERVATION r-a882e29b7 999999999999 default
INSTANCE i-b1a21bd8 ami-f822a39b pending 0 m1.small 2008-12-23T21:37:13+0000 useast-
1c

# Запустить не менее двух экземпляров в зоне доступности us-east-1b
$ ec2-run-instances ami-f822a39b -n 2 -z us-east-1b
RESERVATION r-ac82e29b8 999999999999 default
INSTANCE i-b1a21be9 ami-f822a39b pending 0 m1.small 2008-12-23T21:37:13+0000 useast-
1b
INSTANCE i-b1a21bf0 ami-f822a39b pending 0 m1.small 2008-12-23T21:37:13+0000 useast-
1b

# Запустить в точности один экземпляр с указанной парой ключей в группе
# безопасности myapp
$ ec2-run-instances ami-f822a39b -g myapp -k georgekey
RESERVATION r-a882e29b7 999999999999 default
INSTANCE i-b1a21bd8 ami-f822a39b pending georgekey 0 m1.small 2008-1223T21:
37:13+0000 us-east-1c
```

Команда *ec2-terminate-instances*

Данная команда использует следующий синтаксис:

```
ec2-terminate-instances
```

Команда завершает работу указанного экземпляра или экземпляров.

Пример

```
$ ec2-terminate-instances i-b1a21bd8
INSTANCE i-b1a21bd8 running shutting-down
```

Рекомендации по использованию сервиса Amazon EC2

В этой книге мы обсудили ряд концепций, но до сих пор оставался открытым вопрос о том, как именно вы лично подойдете к реализации этих концепций. В данном разделе я попытаюсь свести воедино некоторые рекомендации, которые помогут вам эффективно настроить ваши облачные инфраструктуры Amazon EC2 и управлять ими. Эти рекомендации, естественно, не представляют собой единствен-

но возможный путь решения поставленной перед вами задачи, так что вам можно дать совет поискать и другие возможности, которые, вероятно, смогут лучше удовлетворить ваши потребности[1].

Шифрование на уровне файловой системы

Я на протяжении всей книги рекомендовал вам прибегнуть к шифрованию ваших файловых систем в облачных инфраструктурах Amazon. Прежде чем вы примете решение применять шифрование, вам необходимо взвесить, что для вас имеет высший приоритет — шифрование на уровне файловой системы в целях обеспечения безопасности или производительность работы вашей файловой системы. Зашифрованные файловые системы всегда будут работать медленнее, чем незашифрованные. Реальное замедление работы будет зависеть от того, что за файловую систему вы используете как основу, а также от того, применяете ли вы RAID. Лично я обычно использую файловую систему XFS на зашифрованных томах RAID0[2].

Чтобы воспользоваться данным советом, вам необходимо установить пакет `cryptsetup`[3]. Если вы хотите обеспечивать поддержку XFS, вам дополнительно потребуется пакет `xfsprogs`[4]. Если в качестве операционной системы вы выбрали Debian, вам нужно выполнить следующие команды от имени пользователя root:

```
apt-get install -y cryptsetup
apt-get install -y xfsprogs
echo sha256 >> /etc/modules
echo dm_crypt >> /etc/modules
```

Командный сценарий (скрипт) UNIX, приведенный в листинге П1.1, если его запустить при старте операционной системы, создаст зашифрованный том XFS, предназначенный для использования эфемерного тома для экземпляра Amazon, использующего модель `m1.small`.

---

[1] Вот только несколько источников, которые содержат полезную информацию и ценные советы по использованию Amazon Web Services:
http://jesperatwork.blogspot.com/2008/02/installing-aqualogic-bpm-enterprise.html,
http://www.royans.net/arch/architecting-for-the-cloud-best-practices/,
http://support.rightscale.com/12-Guides/EC2_Best_Practices/EC2_Site_Architecture_Diagrams,
http://developer.amazonwebservices.com/connect/entry.jspa?externalID=1633&categoryID=100,
http://blogs.sun.com/VirtualGuru/entry/warm_welcome_opensolaris_on_amazon. — *Прим. перев.*

[2] Подробнее о RAID различных уровней см.
http://ru.wikipedia.org/wiki/RAID, **http://en.wikipedia.org/wiki/RAID**,
http://www.parallel.ru/computers/reviews/raid-technology.html, **http://beget.ru/term_Raid**. — *Прим. перев.*

[3] См. **http://tinyurl.com/398hbgp**, **http://code.google.com/p/cryptsetup/**,
http://www.opennet.ru/base/sys/crypt_disk.txt.html, **http://tinyurl.com/394599z**,
http://tinyurl.com/dbv8sk. — *Прим. перев.*

[4] См. **http://linux.about.com/cs/linux101/g/xfsprogs.htm**, **http://oss.sgi.com/projects/xfs/**. — *Прим. перев.*

Листинг П1.1. Скрипт UNIX, запускаемый при старте операционной системы и создающий зашифрованный том XFS для эфемерного тома, предназначенного для экземпляра Amazon EC2 на базе модели `m1.small`

```
# enStratus передает ключ шифрования через Web-сервис при запуске.
# Вы можете получить ключ шифрования из параметров запуска или, для
# эфемерного тома, даже создавать его по требованию, до тех пор пока вы
# не ожидаете возникновения необходимости в поддержке перезагрузки.
# В любом случае ключ шифрования временно помещается в файл /var/tmp/keyfile
# или любой другой по вашему усмотрению.

KEYFILE=/var/tmp/keyfile
# Переход к желаемой файловой системе; если она не поддерживается,
# возврат к ext3
FS=${1}
if [ ! -x /sbin/mkfs.${FS} ] ; then
    FS=ext3
    if [ ! -x /sbin/mkfs.${FS} ] ; then
        echo "Unable to identify a filesystem, aborting..."
        exit 9
    fi
fi
echo "Using ${FS} as the filesystem... "

if [ -f ${KFILE} ] ; then
  if [[ -x /sbin/cryptsetup || -x /usr/sbin/cryptsetup ]]; then
    # Отмонтировать точку монтирования /mnt, которая обычно
    # монтируется к экземплярам заранее
    sudo umount /mnt
    # Настроить шифрование на целевом устройстве
    # (в данном случае на /dev/sda2)
    sudo cryptsetup -q luksFormat --cipher aes-cbc-essiv:sha256 /dev/sda2 ${KFILE}
    if [ $? != 0 ] ; then
        echo "luksFormat failed with exit code $?"
        exit 10
    fi
    # Открыть устройство для использования системой
    sudo cryptsetup --key-file ${KFILE} -q luksOpen /dev/sda2 essda2
    if [ $? != 0 ] ; then
        echo "luksOpen failed with exit code $?"
        exit 11
    fi
    # Форматировать с использованием ранее указанной файловой системы
    sudo mkfs.${FS} /dev/mapper/essda2
    if [ $? != 0 ] ; then
        echo "mkfs failed with exit code $?"
```

```
        exit 12
    fi
    # Удалить текущую запись в /etc/fstab для /dev/sda2
    sudo perl -i -ane 'print unless /sda2/' /etc/fstab
    # Создать новую запись в /etc/fstab (не монтировать автоматически!)
    echo "/dev/mapper/essda2 /mnt ${FS} noauto 0 0" | sudo tee -a /etc/fstab
    # Примонтировать диск
    sudo mount /mnt
    if [ $? != 0 ] ; then
        echo "Remount of /mnt failed with exit code $?"
        exit 13
    fi
  fi
fi
```

При использовании этой модели вам потребуется держать наготове файл с ключом в незашифрованном виде. Сервис enStratus (**http://www.enstratus.com/**) позволяет вам удалить этот файл после завершения работы скрипта, так как ключ будет передан ядру в тот момент, когда он понадобится.

Единственным слабым местом любого из перечисленных подходов является то, что незашифрованная версия ключа хранится в незашифрованном корневом разделе, пусть даже в течение очень краткого промежутка времени, как в модели enStratus.

Чтобы обойти эту проблему, можно воспользоваться следующим простым приемом:

1. Автоматически сгенерировать случайный ключ и сохранить его в каталоге `/var/tmp`.
2. Примонтировать эфемерное хранилище к каталогу `/mnt` как зашифрованное устройство.
3. Стереть автоматически сгенерированный ключ.
4. Передать постоянный ключ из enStratus (или любого другого инструмента).
5. Сохранить его в `/mnt/tmp`.
6. Примонтировать том EBS, используя постоянный ключ шифрования.
7. Удалить постоянный ключ шифрования.
8. Использовать том EBS для хранения всех критически важных данных.

При использовании этого подхода никакие фрагменты критически важных данных не хранятся на томе, ключ которого хоть на какое-то время оказывается незашифрованным, за исключением работы на уровне локальной операционной системы. Постоянный ключ хранится на зашифрованном томе, для расшифровки которого применяется временный ключ шифрования, генерирующийся случайным образом.

Если вы действительно хотите использовать сильное шифрование, заполните носитель случайным образом сгенерированными данными в формате `raw` перед тем, как начать использование этого носителя.

Настройка RAID для использования множества томов EBS

Я убедился в том, что за счет объединения нескольких томов EBS в массив RAID0 (чередование, striping) производительность дискового ввода/вывода серьезно повышается. С другой стороны, я бы не стал ожидать существенных преимуществ за счет организации массива RAID1 (зеркальное копирование, mirroring), так как все тома EBS должны находиться в одной и той же зоне доступности, что и экземпляр, к которому они прикрепляются. К сожалению, мои собственные эксперименты, которые примерно соответствуют ожиданиям Amazon, показали, что зеркальное копирование (mirroring) томов EBS почти полностью сводит на нет выигрыш в производительности, получаемый за счет организации массивов с чередованием (striping), так что создание массивов по типу RAID5 или RAID10 вообще не дает никакого положительного результата.

Чтобы обеспечить честность и беспристрастность оценки, я не только провел собственные эксперименты по тестированию на производительность всех доступных опций с применением всех популярных (или хотя бы известных) файловых систем. В настоящий момент я предпочитаю применять один зашифрованный том EBS для каждого отдельного экземпляра, за исключением тех случаев, когда производительность представляет собой абсолютный приоритет. В таких случаях, когда производительность ставится во главу угла, я предпочитаю объединять тома в общий массив RAID0.

Я могу шифровать или не шифровать этот массив RAID0, в зависимости от результатов лично проведенного анализа приоритетов между безопасностью данных и производительностью работы дисковой подсистемы.

При планировании конфигурации, использующей RAID0, вам следует иметь в виду, что параллельно выполняемые операции записи на диск, которые и позволяют добиться максимальной скорости обмена данными при использовании RAID0, имеют верхний предел по скорости обмена данными через соединение, подключающее ваш экземпляр к вашим томам EBS. По этой причине оптимальное количество носителей и их объемы будут зависеть от типа вашего приложения.

Для создания и настройки массивов RAID я использую пакет `mdadm`:

```
sudo apt-get install -y mdadm
```

Командный сценарий (скрипт) UNIX, приведенный в листинге П1.2, позволит создать массив RAID0 из двух дисков.

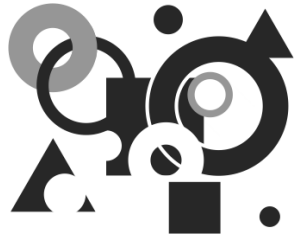
Листинг П1.2. Скрипт UNIX, позволяющий создавать массив RAID0 из двух дисков

```
SVCMOUNT=/mnt/svc
# Переход к желаемой файловой системе; если она не поддерживается,
# возврат к ext3
FS=${1}
if [ ! -x /sbin/mkfs.${FS} ] ; then
    FS=ext3
```

```
    if [ ! -x /sbin/mkfs.${FS} ] ; then
        echo "Unable to identify a filesystem, aborting..."
        exit 9
    fi
fi
echo "Using ${FS} as the filesystem... "

ISNEW=${2}

ARGS=($@)
DEVICELIST=${ARGS[@]:2}
DEVARR=($DEVICELIST)
DEVCOUNT=${#DEVARR[*]}

if [ ! -d ${SVCMOUNT} ] ; then
    sudo mkdir ${SVCMOUNT}
    sudo chmod 775 ${SVCMOUNT}
fi

# Убедиться в наличии устройств для монтирования
if [ ${DEVCOUNT} -gt 0 ] ; then
 # Если устройств больше одного, создадим RAID0
 if [ ${DEVCOUNT} -gt 1 ] ; then
  map=""
  for d in ${DEVICELIST}; do
    map="${map} /dev/${d}"
  done
  # Создать устройство RAID0 /dev/md0
  yes | sudo mdadm --create /dev/md0 --level 0 --metadata=1.1 --raid-devices ${DEVCOUNT} $map
  if [ $? != 0 ] ; then
     exit 20
  fi
  # Сконфигурировать RAID на случай перезагрузки
  echo "DEVICE ${DEVICELIST}" | sudo tee /etc/mdadm.conf
  sudo mdadm --detail --scan | sudo tee -a /etc/mdadm.conf
  # Есть ли вновь созданные тома или перемонтированные
  # старые тома/моментальные снимки?
  if [ ${ISNEW} == "true" ] ; then
     # Создать новую файловую систему
     sudo mkfs.${FS} /dev/md0
     if [ $? != 0 ] ; then
        exit 24
     fi
     echo "/dev/md0 ${SVCMOUNT} ${FS} noatime 0 0" | sudo tee -a /etc/fstab
```

```
  else
     echo "/dev/md0 ${SVCMOUNT} ${FS} noatime 0 0" | sudo tee -a /etc/fstab
  fi
else
  # Только один том, это не RAID
  if [ ${ISNEW} == "true" ] ; then
    # Новый том, создать файловую систему
    sudo mkfs.${FS} /dev/${DEVICELIST}
      if [ $? != 0 ] ; then
        exit 29
      fi
      echo "/dev/${DEVICELIST} ${SVCMOUNT} ${FS} noauto 0 0" | sudo tee -a /etc/fstab
  else
    echo "/dev/${DEVICELIST} ${SVCMOUNT} ${FS} noauto 0 0" | sudo tee -a /etc/fstab
  fi
 fi
    sudo mount ${SVCMOUNT}
fi
```

Приложение 2



Облачный хостинг GoGrid

Рэнди Байес (Randy Bias)

Существует множество методов построения облачной инфраструктуры. Amazon Web Services (AWS) использует для этого подход, известный как "сервисная инфраструктура" (service infrastructure), и предлагает для этой цели заказчикам ряд индивидуально настраиваемых, нестандартных и в высшей степени масштабируемых сервисов, позволяющих им перестроить существующую у них инфраструктуру для работы в полностью виртуальной среде. В отличие от AWS, GoGrid[1] предоставляет своим клиентам подход, более ориентированный на существующие промышленные стандарты, предлагающий им знакомую среду, напоминающую типичный центр обработки данных, но разворачиваемый в облачной инфраструктуре.

Есть большое количество клиентов, для которых подход GoGrid окажется более "близким по духу", более привычным и комфортным, поскольку в нем находят применение более привычная терминология и более знакомые технологии, например виртуальные локальные сети (VLANs, virtual LANs), сетевые блоки (network block), аппаратные устройства (hardware appliances), такие как балансировщики нагрузки (load balancers), и межсетевые экраны или брандмауэры (firewalls) и сетевые хранилища данных, в том числе — SAN (storage area networks) или NAS (network attached storage). Эти решения определяют используемую нами концепцию "облачного центра обработки данных" (cloudcenter, или "центр обработки данных в облачной среде").

Типы облаков

Инфраструктурные облака (также известные под названием "инфраструктура как сервис", "Infrastructure-as-a-Service" или IaaS) можно строить двумя путями:

---

[1] Подробнее о GoGrid см. **http://www.gogrid.com/**, **http://wiki.gogrid.com/wiki/index.php/Main_Page**, **http://gogrid.ru/**. — *Прим. перев.*

построением сервисных инфраструктур (service infrastructures) или построением "облачных центров" (cloudcenters).

Оба варианта позволяют представлять клиентам те возможности, которых они вправе ожидать от поставщика услуг IaaS:

- масштабирование по требованию;
- оплата по факту потребления услуги;
- возможность преобразования капитальных расходов (capital expenditures, CapEx) в производственные затраты (operational expenditures, OpEx);
- программные (API) и графические пользовательские (GUI) интерфейсы;
- базовая инфраструктура: пространство для хранения данных, серверы, сеть, электропитание, охлаждение.

Хотя оба подхода и предоставляют тот же самый базовый набор услуг, они существенно отличаются.

- *Сервисные инфраструктуры* — этот подход похож на подход, применяемый AWS и широко освещенный в данной книге. Сервисные инфраструктуры в принципе представляют собой индивидуально настраиваемые Web-сервисы в облачной среде. Они могут использоваться как по отдельности, так и в комбинации, с тем, чтобы предоставить клиенту возможность развертывания Web-приложения или выполнения пакетной обработки данных. Например, Amazon предоставляет серверы, пространство для хранения данных, базы данных, обслуживание очередей (queuing)/систем обмена сообщениями (messaging), обработки электронных платежей (payment processing) и многое другое. Каждый отдельный Web-сервис из числа перечисленных представляет собой уникальное и индивидуально настраиваемое решение. Хранилище на базе S3 использует протокол S3 и механизмы организации пространства для хранения данных, специфичные для S3. Сервис обслуживания очередей AWS SQS использует собственный нестандартный патентованный протокол и собственный формат сообщений. Точно так же организован и их сервис управления базами данных SimpleDB. Все эти сервисы компания Amazon разработала так, чтобы обеспечить возможности масштабирования до 50 000 серверов и более, на которых работают тысячи разнообразных программных продуктов. Все предоставляемые сервисы являются общедоступными и могут перенастраиваться так, чтобы клиенты AWS могли применять их в своих собственных целях, в рамках собственных бизнес-моделей.
- *Облачные центры* — это подход, применяемый большинством компаний, конкурирующих с AWS. Эта методология нацелена на предоставление стандартных услуг, типичных для обычных центров обработки данных с использованием стандартных технологий и протоколов, но только в облачной среде. Пространство для хранения данных предоставляется через известные протоко-

лы, такие как SMB[1]/CIFS[2] и NFS[3]. Доступ к базам данных обеспечивается посредством стандартного языка SQL и реляционных систем управления базами данных RDBMS). Межсетевые экраны (Firewalls) и балансировщики нагрузки представляют собой специализированные аппаратные устройства, а не распределенное и индивидуально конфигурируемое программное обеспечение.

В результате клиенты делают выбор между индивидуально настраиваемой инфраструктурой с патентованными протоколами и стандартами, которые клиент обязан соблюдать, и традиционным сервисом, более похожим на услуги, предоставляемые обычными центрами обработки данных. Такие услуги обычно предоставляют все те же преимущества, но более ориентированы на соответствие общепринятым промышленным стандартам, чем сервисные инфраструктуры.

Облачные центры

GoGrid — первый и самый крупный из облачных центов в США, популяризирующий и пропагандирующий этот подход. В качестве примера других компаний, применяющих подход "облачного центра" (cloudcenter), можно привести FlexiScale[4], ElasticHosts[5] и AppNexus[6]. К числу преимуществ данного подхода можно отнести то, что при его использовании вам не потребуется переучиваться, приобретая новые знания и навыки, возможность использования уже существующей инфраструктуры, а также более гибкую облачную среду. Подход компании GoGrid

---

[1] SMB (Server Message Block) — сетевой протокол прикладного уровня для удаленного доступа к файлам, принтерам и другим сетевым ресурсам, а также для межпроцессного взаимодействия. Первая версия протокола была разработана компаниями IBM, Microsoft, Intel и 3Com в 1980-х годах; вторая (SMB 2.0) была создана Microsoft и впервые была введена в Windows Vista. В настоящее время SMB связан, главным образом, с операционными системами Microsoft Windows, где используется для реализации "Сети Microsoft Windows" (Microsoft Windows Network) и "Совместного использования файлов и принтеров" (File and Printer Sharing). Подробнее см. **http://en.wikipedia.org/wiki/Server_Message_Block**. — *Прим. перев.*

[2] Common Internet File System,
см. **http://ru.wikipedia.org/wiki/SMB**, **http://www.samba.org/cifs/**. — *Прим. перев.*

[3] Network File System (NFS) — протокол сетевого доступа к файловым системам, первоначально разработанный Sun Microsystems в 1984 году. Основан на протоколе вызова удаленных процедур, позволяет подключать (монтировать) удаленные файловые системы через сеть. Подробнее см. **http://ru.wikipedia.org/wiki/Network_File_System**, **http://datatracker.ietf.org/wg/nfsv4/charter/**, **http://wiki.linux-nfs.org/wiki/index.php/Main_Page**, **http://solarisblog.ru/networks/osnovy-nfs-v-os-solaris**. — *Прим. перев.*

[4] FlexiScale — это платформа, работающая по принципу предоставления информационных сервисов как коммунальных услуг (utility computing), созданная компанией XCalibre летом 2007 года, а затем приобретенная компанией Flexiant. Подробнее см.
http://www.flexiant.com/products/flexiscale/, **http://www.flexiant.com/**. — *Прим. перев.*

[5] ElasticHosts — это международный облачный сервис-провайдер, обладающий двумя центрами обработки данных (под Лондоном в Великобритании и в Сан-Антонио, США). Подробнее см. **http://www.elastichosts.com/**. — *Прим. перев.*

[6] AppNexus — это компания, которая предлагает аукцион рекламы в реальном времени. Служба AppNexus обслуживает 120 рекламных сетей в 12 странах. См. **http://appnexus.com/**. — *Прим. перев.*

предлагает и еще одно преимущество — простоту и удобство интеграции вашего существующего центра обработки данных с внешними облачными средами.

Центры обработки данных в облачных средах

Традиционные центры обработки данных включают в свой состав следующие компоненты:

- защиту периметра сети с помощью аппаратного брандмауэра и сетевой системы обнаружения вторжений (NIDS);
- балансировку нагрузки с помощью аппаратного балансировщика нагрузки;
- сегментацию сети с помощью различных сетевых блоков (network blocks) и виртуальных локальных сетей (VLAN);
- комбинацию физических аппаратных средств и виртуальных гостевых операционных систем;
- совместный доступ к файлам с использованием NAS;
- блочные устройства хранения (SAN);
- поддержку сервисов центра обработки данных: DNS, DHCP, создание образов серверов (server imaging), управление инвентаризацией (inventory management), управление активами и их мониторинг (asset management and monitoring);
- электропитание, охлаждение, полосу пропускания и резервное копирование для всех перечисленных сервисов;
- круглосуточную непрерывную техническую поддержку и квалифицированный персонал.

Облачные центры очень похожи на традиционные центры обработки данных. Они предлагают большинство из перечисленных услуг с лишь незначительными вариациями, предоставляя их на многоарендной основе. В дополнение, облачные центры, в отличие от обычных центров обработки данных, посредством GUI и API предоставляют выигрыш по ценам и по индивидуальной производительности работы персонала.

GoGrid и традиционные центры обработки данных

Основным недостатком традиционных центров обработки данных является необходимость их построения в расчете на максимальную вычислительную мощность. Вы, как владелец инфраструктуры IT, должны уметь правильно спрогнозировать необходимую вам вычислительную мощность и грамотно управлять вашей готовой внутренней физической инфраструктурой.

Облачные центры предоставляют вам возможность повторно использовать весь опыт, накопленный в области управления собственной физической инфраструктурой IT при пользовании услугами внешних облачных провайдеров. Повторное использование знаний и накопленного опыта означает, что вам не потребуется тратить много времени на изучение новых подходов и парадигм. Это, в свою очередь,

значит, что у вас будет больше времени на ведение реального бизнеса и достижение целей, поставленных вашим бизнес-планом. Вы сможете сохранить полный контроль над своими данными, но при этом в полной мере использовать преимущества, предоставляемые облачной средой, в том числе: лицензирование операционных систем, возможность избежать капитальных затрат, прогнозируемость, управление вычислительными мощностями.

Кроме того, вам будут доступны и такие преимущества облачной среды, как наращивание доступных мощностей по требованию, автоматизация распределения рабочих нагрузок, и оплата только за фактически полученные услуги. Все это позволяет техническому персоналу и менеджерам IT оптимизировать затраты на инфраструктуру, как внутреннюю, так и облачную, за счет динамического управления потребляемыми ресурсами и расширения или уменьшения потребляемых мощностей в зависимости от текущих потребностей.

Горизонтальное и вертикальное масштабирование

Развертывание приложений в GoGrid напоминает аналогичный процесс, выполняемый в собственной физической IT-инфраструктуре (центре обработки данных) предприятия, но с возможностью использования множества дополнительных средств, помогающих ускорить и упростить эти процессы. Как и в случае с другими облачными средами, вы имеете возможность расширения, известную под названием "горизонтального масштабирования" (horizontal scaling). Но, в отличие от многих других облачных сред, вы имеете возможность и "вертикального роста" (scale up), известную под названием "вертикального масштабирования" (vertical scaling), при которой используются не только возможности виртуализованной облачной среды, но и применения реальных, выделенных физических аппаратных устройств. В этом услуги GoGrid напоминают услуги традиционного физического центра обработки данных, где клиенты могут использовать комбинацию виртуальных и физических серверов. Иначе говоря, вы имеете возможность получения прямого доступа к физическому оборудованию с большими объемами RAM, высокоскоростные устройства хранения данных с непосредственным доступом (high-speed direct-attached-storage, DAS), а также большое количество независимых сервисов, основанных на виртуальных частных сетях (virtual private network, VPN).

Используйте правильно подобранный инструмент для решения правильно сформулированной задачи: виртуальные серверы для бесстатусных (stateless) рабочих нагрузок (таких как централизованная обработка, не связанная с сохранением данных), которые вы легко можете масштабировать горизонтально, и физические серверы — для рабочих нагрузок с сохранением статуса (stateful workloads), т. е. таких как манипулирование сохраняемыми данными, например операции над базами данных, которые проще масштабировать вертикально.

Рассмотрим основные факторы, которые обосновывают сказанное.

- *Горизонтальное масштабирование* (scaling out). Горизонтальному масштабированию проще всего поддаются серверы и такие приложения, которые сохраняют относительно мало информации о статусе, например Web-серверы,

серверы приложений, а также все задания по пакетной обработке (batch processing). При этих типах рабочих нагрузок добавление дополнительного сервера обычно требует небольшого объема работ по планированию архитектуры, настройке и конфигурированию, или не требует их вообще. Таким образом, добавление дополнительных серверов предоставляет простой путь наращивания мощностей.

- *Вертикальное масштабирование* (Scaling up). В отличие от предыдущего случая, вертикальное масштабирование лучше всего подходит для таких приложений и типов рабочих нагрузок, которые связаны с хранением информации о состоянии — например, для баз данных и файл-серверов. В таких случаях простое добавление дополнительного сервера не приводит к прямому наращиванию мощностей. Обычно в таких ситуациях требуется выполнять большие работы по переконфигурированию, менять архитектуру или, по крайней мере, выполнять автоматическую балансировку данных о состоянии по новым серверам. Большинство ваших статусных данных хранятся именно на таких серверах, поэтому добавление нового сервера потребует динамической синхронизации терабайтов данных, что само по себе является нетривиальной задачей. Именно по этой причине обычно для таких задач желательно использовать более мощные серверы, нежели большее количество менее мощных. Кроме того, такие серверы имеют тенденцию быть не динамическими по своей природе, даже в облачной среде.

По только что описанным причинам GoGrid поддерживает обе шкалы масштабирования — и горизонтальную, и вертикальную. Горизонтальное масштабирование не всегда достаточно, а вертикальное требует выработки стратегии масштабирования. Компания Amazon тоже это понимает, поэтому для развертывания приложений в своей облачной среде AWS она предлагает различные масштабы серверов. Однако виртуализации внутренне присуща стратегия, известная под названием многоарендности (multitenancy), при которой подразумевается, что каждый отдельно взятый сервер предоставляет хостинг многим клиентам и многим приложениям. Если некоторый из компонентов вашего приложения может полностью использовать все вычислительные мощности современного физического сервера, то не имеет смысла запускать это приложение на сервере виртуальном. В данном случае виртуализация будет приводить к неоправданным накладным расходам, и, если вам требуются бóльшие вычислительные мощности или объемы оперативной памяти (RAM), чем может предложить облачная среда, вам потребуется пересмотреть вашу архитектуру с тем, чтобы прибегнуть к горизонтальному масштабированию.

С другой стороны, большинство приложений на определенном этапе достигнут "потолка", при котором еще возможно вертикальное масштабирование, и тогда вы будете уже просто вынуждены прибегнуть к горизонтальному масштабированию. Вам потребуется соотнести потребность в ресурсах и доступные мощности, чтобы получить возможность применения модели горизонтального масштабирования. Но,

поскольку закон Мура (Moore's Law)[1] все-таки никто не отменял, пройдет немало времени, прежде чем физическое оборудование перестанет удовлетворять требованиям стратегии вертикального масштабирования для большинства приложений.

Архитектуры для развертывания GoGrid

Типичная среда развертывания приложений GoGrid выглядит примерно так, как ваш собственный внутренний центр обработки данных (рис. П2.1).

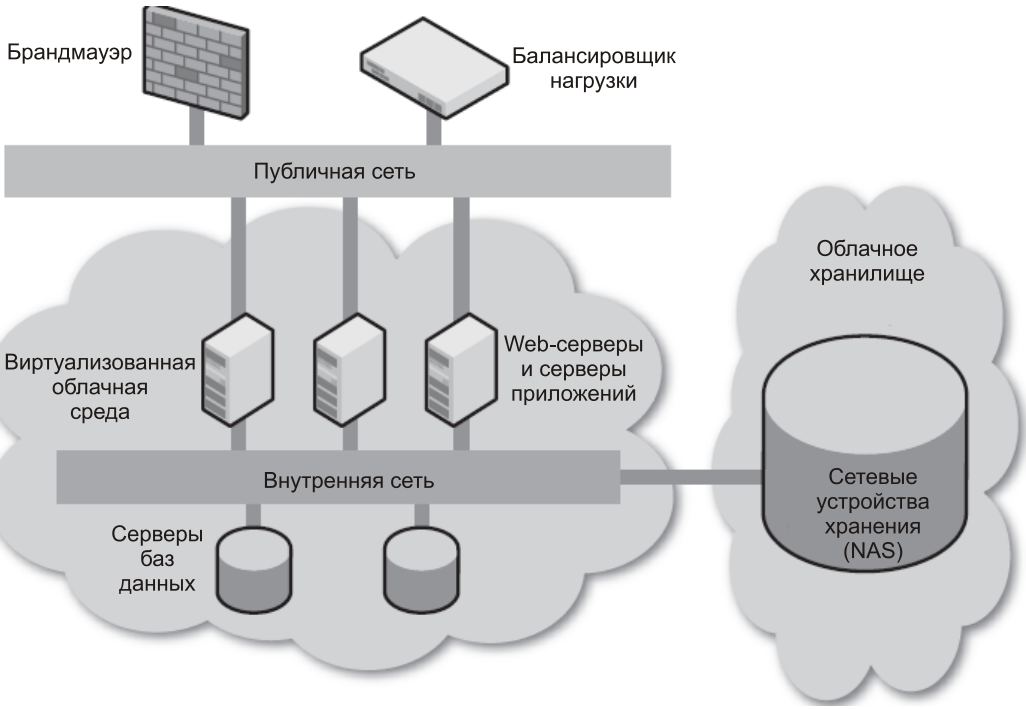


Рис. П2.1. Роль виртуализации и облачной среды в GoGrid

В типичных физических центрах обработки данных серверы приложений взаимодействуют с Интернетом, чтобы дать пользователям возможности доступа к приложениям, в то время как внутренние серверы обработки данных защищены дополнительным уровнем, подобным демилитаризованной зоне (DMZ), и все системы имеют доступ по защищенным каналам к сетевым средствам хранения данных (NAS). Как и в традиционном центре обработки данных, в GoGrid имеются два

[1] Закон Мура — эмпирический закон, выведенный в 1965 году Гордоном Муром (Gordon Earle Moore). Он высказал предположение о том, что число транзисторов на кристалле будет удваиваться каждые 2 года.
Подробнее см. **http://en.wikipedia.org/wiki/Moore's_law**, **http://download.intel.com/museum/Moores_Law/Printed_Materials/Moores_Law_Backgrounder.pdf**, **http://www.ixbt.com/editorial/moorelaw40th.shtml**. — *Прим. перев.*

сетевых сегмента (VLANs), один — для публичного доступа (frontend), другой — для внутреннего доступа (backend), использующий частные IP-адреса (RFC1918). Как и в традиционном центре обработки данных, здесь имеется NAS (GoGrid Cloud Storage) для использования под домашние каталоги, резервное копирование, архивы и другие потребности.

Вертикальное масштабирование (scaling up) в GoGrid не слишком сильно отличается от горизонтального (scaling out), за тем исключением, что высокопроизводительные базы данных, имеющие критически важное значение, работают на выделенных физических серверах, как показано на рис. П2.2.

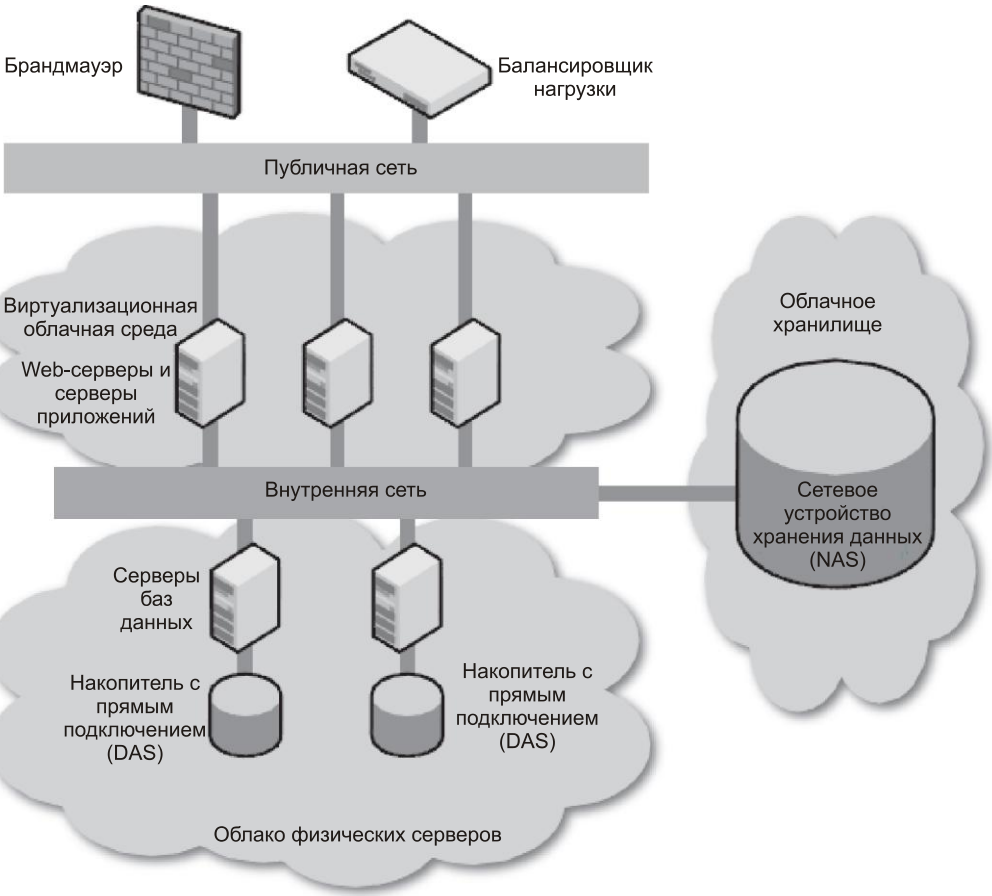


Рис. П2.2. Роль физического хостинга в GoGrid

Web-приложения — в центре внимания

Архитектура облачного центра более "дружелюбна" по отношению к Web-приложениям, нежели к заданиям по пакетной обработке данных, которым не обязательно требуется вся инфраструктура традиционного центра обработки данных, вклю-

чая брандмауэры (firewalls), балансировщики нагрузки и VLAN. Большинство приложений, предназначенных для пакетной обработки данных, довольно хорошо работают в средах, предоставляющих информационные сервисы как коммунальные услуги (utility computing), и грид-средах (grid computing). GoGrid тоже может применяться для пакетной обработки данных, и, хотя пакетная обработка является важным компонентом многих Web-приложений, мы оптимизировали нашу среду под основные транзакционные Web-приложения, которые мы считаем намного более важными.

Сравнение подходов

При сравнении облачных центров (GoGrid) с сервисными инфраструктурами (AWS)[1] важно учитывать как практику развертывания приложений в традиционных центрах обработки данных, так и природу развертываемых приложений.

Непосредственное сравнение

Весьма полезным будет сравнительный анализ традиционных центров обработки данных, облачных центров (cloudcenters) и сервисных инфраструктур. В табл. П2.1 перечислены основные области применения для каждого типа инфраструктур IT и приведен их сравнительный анализ.

Таблица П2.1. *Сравнение функциональных возможностей и областей применения различных инфраструктур IT*

Функциональные возможности	Традиционный центр обработки данных	GoGrid (облачный центр обработки данных)	Amazon (сервисная инфраструктура)
Межсетевой экран (брандмауэр)	Аппаратный брандмауэр для защиты периметра	Аппаратный брандмауэр для защиты периметра (с начала 2009 года)	Индивидуально настроенный распределенный программный брандмауэр
Балансировщик нагрузки	Аппаратный балансировщик нагрузки	Аппаратный балансировщик нагрузки	Индивидуально развертываемый программный балансировщик нагрузки (возможно использование релиза от 2009 года для индивидуального сервиса балансировщика нагрузки)

[1] См. также **http://gogrid.ru/pricing/gogrid_vs_ec2.php** и **http://www.cloudtweaks.com/2010/09/cloud-computing-comparisons-between-aws-rackspace-and-gogrid/** (более полный сравнительный анализ, учитывающий также и возможности сервиса Rackspace, который будет рассматриваться в *Приложении 3*). *— Прим. перев.*

Таблица П2.1 *(окончание)*

Функциональные возможности	**Традиционный центр обработки данных**	**GoGrid (облачный центр обработки данных)**	**Amazon (сервисная инфраструктура)**
Изоляция сетей	VLAN	VLAN	Искусственное разделение по типу "VLAN" с использованием распределенного программного брандмауэра
Частные сети	Есть (VLAN)	Есть (VLAN)	Нет
Сетевые протоколы	Без ограничений	Без ограничений	Ограниченные возможности, нет многоадресной рассылки (multicast), нет широковещания (broadcast), GRE[1] и все, что относится к туннелированию, не работает
Выбор операционных систем	Без ограничений	С некоторыми ограничениями	С некоторыми ограничениями
DNS	Есть; внутрифирменное управление	Есть; управляется GoGrid	Нет
Постоянное локальное хранилище	Есть	Есть	Нет
Постоянное сетевое хранилище	Есть	Есть	Есть
Комбинирование виртуальных и физических серверов	Есть	Есть	Нет

Как видите, облачные архитектуры в стиле облачных центров во многом очень похожи на архитектуры традиционных центров обработки данных.

[1] GRE (Generic Routing Encapsulation — общая инкапсуляция маршрутов) — протокол туннелирования сетевых пакетов, разработанный компанией Cisco Systems. Его основное назначение — инкапсуляция пакетов сетевого уровня сетевой модели OSI (Open Systems Interconnection) в IP-пакеты. Подробнее см. **http://en.wikipedia.org/wiki/Generic_Routing_Encapsulation**. — *Прим. перев.*

Практическое использование

Различия между облачными центрами и сервисными инфраструктурами станут более очевидны, если вы попробуете параллельно пользоваться GoGrid и AWS.

При использовании AWS (модель сервисной инфраструктуры) полезной и относящейся к делу окажется лишь небольшая часть накопленного вашим персоналом опыта работы в обслуживании сетей и накопителей. Вам потребуется приобретать новые знания и навыки для управление Amazon S3, а также расширять имеющиеся навыки системного администрирования, изучая новые парадигмы управления серверами Amazon EC2, включая метаданные времени исполнения, отсутствие возможностей многоадресной доставки сетевых пакетов и широковещательного сетевого трафика, группы серверов, а также концепцию индивидуально настраиваемого распределенного программного брандмауэра.

Напротив, подход "облачного центра обработки данных" (cloudcenter), практикуемый GoGrid, очень похож на использование консоли VMware VirtualCenter[1] или любой другой системы управления виртуализацией. В дополнение к серверам, в GoGrid вы имеете возможность управления вашей сетью, DNS, накопителями, балансировщиками нагрузки, брандмауэрами[2].

При использовании любой из систем вы сможете воспользоваться стандартными преимуществами, предоставляемыми облачной средой за счет масштабирования по требованию и оплаты только за фактически потребляемые ресурсы.

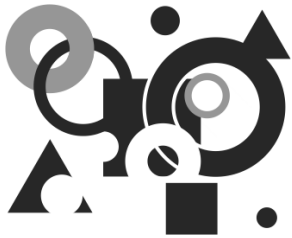
Что лучше подойдет вам?

В сущности, правильный выбор в вашем случае будет зависеть от природы вашего приложения, потребностей вашего предприятия, а также от вашего подхода к традиционному вопросу "построить самостоятельно или купить готовое". При выборе различных возможностей по приобретению и сравнении их с решениями на базе облачных инфраструктур вам следует оценить, что для вас важнее — сконцентрировать усилия на адаптации индивидуально настроенных решений в различных сервисных инфраструктурах или же лучше воспользоваться облачной средой, больше напоминающей ваш традиционный центр обработки данных.

[1] Подробнее см. **http://www.vmware.com/ru/products/vi/vc/**. — *Прим. перев.*

[2] Поддержка аппаратных брандмауэров в GoGrid появилась с 2009 года. Подробнее см. **http://wiki.gogrid.com/wiki/index.php/Hardware_Firewalls**.

Приложение 3



Компания Rackspace

Эрик Джонсон (Eric Johnson)

Rackspace — это, по всей видимости, самая известная из компаний, предоставляющих корпорациям услуги хостинга в традиционном понимании этого термина. Свою деятельность компания начала с предоставления услуг управляемого хостинга физических серверов и предложения услуг по технической поддержке под слоганом Fanatical Support[1], позволяя большинству клиентов полностью положиться на услуги технических экспертов компании, которые обеспечивали им высокую стабильность и отказоустойчивость. Среди клиентов Rackspace можно найти самые разные компании — от малых предприятий, имеющих всего один сервер, до фирм, поддерживающих сложные сетевые конфигурации, насчитывающие сотни серверов, и имеющих мощную высокопроизводительную сетевую инфраструктуру.

Но на настоящий момент Rackspace находится в курсе характерных тенденций, ярче всего проявившихся на примере Amazon EC2 и S3. Теперь компания практически готова полностью (или частично) перейти от предоставления в аренду физических серверов на использование облачной инфраструктуры, в которой некоторые клиенты чувствуют себя комфортнее, работая в разделяемой среде.

Rackspace ищет возможности для предоставления клиентам полного набора опций, варьирующихся в самом широком диапазоне — от выделенных физических серверов до полностью виртуальных серверов, не пренебрегая и гибридными средами, сочетающими возможности полностью физических и полностью виртуальных. Облачные сервисы от Rackspace (Cloud Services), предоставляемые подразделением Rackspace Cloud Division[2], являются своего рода "зонтичной" группой сервисов, предоставляющих клиентам продукты на основе облачных технологий от

---

[1] Fanatical Support ("фанатическая поддержка") — этот лозунг основатели фирмы Rackspace ввели в оборот, как только увидели, что все существующие компании в этом бизнесе в основном были сосредоточены только на технических аспектах, а не на поддержке пользователей и услугах, и ни одна компания в то время не могла удовлетворить предъявленные ими уникальные требования. Подробнее см. **http://www.smart-cloud.org/sorted-articles/44-for-all/159-own-cloud-server-made-easy-5**, и здесь же — довольно подробный и интересный рассказ о самой компании Rackspace, истории ее основания и о том, как стать клиентом. — *Прим. перев.*

[2] Точный адрес подразделения: **http://www.rackspacecloud.com/?id=736**. — *Прим. перев.*

Rackspace. За счет комбинирования уже существующих предложений по управляемому хостингу и новых вариантов услуг, разработанных Rackspace и некоторыми из компаний, приобретенных Rackspace в последнее время, компания планирует предлагать клиентам самый широкий спектр решений, удовлетворяющих любые потребности. Контингент клиентов может варьироваться — от небольших предприятий типа "стартап"[1], которые начинают свою деятельность в облачной и виртуальной среде, и до крупных предприятий, владеющих мощной инфраструктурой (не исключая, разумеется, и множества промежуточных вариантов).

Многие предприятия, которые на текущий момент являются клиентами Rackspace, начали чувствовать преимущества использования облачных серверов. За счет того, что Rackspace разрабатывает и предлагает заказчикам облачные сервисы, клиенты компании могут пользоваться преимуществом взаимоотношений с единым поставщиком услуг, применять технологии, полностью интегрированные в общую инфраструктуру, а также во многих случаях могут получить и дополнительный выигрыш по производительности за счет того, что физические серверы располагаются в тех же центрах обработки данных, где получают хостинг и их облачные сервисы.

Облачные серверы Rackspace

В октябре 2008 года компания Rackspace приобрела фирму Slicehost (**http://www.slicehost.com**), которая являлась лидером рынка в области предоставления хостинга виртуальным серверам Linux. Mosso, подразделение Rackspace, начало заниматься разработкой облачных серверов, которые должны стать аналогом того, что предлагается Amazon EC2.

Как показывалось на протяжении всей этой книги, привлекательность использования EC2 заключается в том, что ваши физические серверы замещаются виртуальными экземплярами EC2, расположенными в облачной инфраструктуре. Очевидно, что использование EC2 особенно выгодно тем клиентам, которые испытывают временную потребность в вычислительных мощностях и желают увеличивать или уменьшать ресурсы в соответствии с требованиями, испытываемыми в конкретный момент времени. В отличие от этой схемы, Slicehost строится в соответствии с нуждами клиентов, которым требуется дешевый хостинг, и такой хостинг им предоставляется на основе выделенных виртуальных серверов. Предложение Slicehost разработано в соответствии со схемой 24/7/365 (круглосуточно, без перерывов, все 365 дней в году). Slicehost использует физические хост-серверы с дисковыми массивами RAID(1+0), избыточные резервные источники энергии и пользуется услугами магистральных интернет-провайдеров.

[1] Стартап (от англ. *start-up* — запускать) — компания с короткой историей операционной деятельности. Как правило, такие компании созданы недавно, находятся в стадии развития или исследования перспективных рынков. Подробнее см. **http://en.wikipedia.org/wiki/Startup_company**. — *Прим. перев.*

В ближайшем будущем основная технология Slicehost будет комбинироваться с другими функциями, направленными на то, чтобы предоставлять клиентам динамические возможности, аналогичные тем, которые обеспечиваются Amazon EC2. Slicehost — это компания, которая планирует сохранить независимость за счет расширения спектра своих услуг и повышения их качества. Продукт Cloud Servers, разработанный на основе технологий Slicehost, будет включать в свой состав следующие компоненты:

- индивидуально настроенные образы и репозитории образов;
- интерфейсы прикладного программирования (API);
- статические IP-адреса;
- расценки по факту использования;
- поддержку Microsoft Windows;
- традиционную круглосуточную беспрерывную техническую поддержку (24/7 Fanatical Support).

Как и в случае с управляемым хостингом, продукт Cloud Servers будет динамически предоставлять клиентам ресурсы в зависимости от их потребностей в вычислительных ресурсах. Эти "гибридные" возможности особенно удобны для всех типов предприятий, поскольку они позволяют быстро создавать временные виртуальные серверы на момент ударных нагрузок, мгновенно удовлетворяя внезапно возросшие требования.

Выгодой, получаемой клиентами от Rackspace Cloud Servers, является возможность мгновенного создания операционных сред, полностью дублирующих их физические конфигурации. Эти дублирующие среды могут применяться для тестирования, анализа качества, разработки, исследования модификаций и т. д. Их можно создавать на ограниченные периоды, а затем останавливать до того момента, когда необходимость в них возникнет вновь.

Сервис Cloud Files

Cloud Files — это сервис хранения файлов, очень похожий на систему Amazon S3, которая описывалась ранее в этой книге. Раннее (закрытое) бета-тестирование Cloud Files началось в первые месяцы 2008 года, а в октябре того же года продукт вступил в стадию публичного бета-тестирования. Как и S3, Cloud Files не предназначается для прямой замены традиционных решений по резервному копированию. Однако этот сервис предлагает очень интересные возможности, в том числе:

- организация данных (объектов) по "отделениям" хранилища, которые называются контейнерами (Containers). Контейнеры не допускают вложенности (т. е. не могут содержать других контейнеров). Это значит, что хранилище не является иерархическим;
- размер объектов может варьироваться в пределах от нуля до 5 Гбайт;

- с объектами можно ассоциировать индивидуально настроенные наборы метаданных;
- каждый клиент может создавать неограниченное количество контейнеров и объектов;
- все функции и возможности хранилища доступны через Web-интерфейс и программно — через различные API (ReST, PHP, Python, Ruby, Java, C#/.NET);
- сервисы могут предоставляться на условиях оплаты по факту использования, что избавляет вас от переплаты за ресурсы, которые вы оплатили, но потребили в меньшем количестве, чем было заказано (в частности, за недоиспользованное пространство для хранения данных).

Хорошим примером, иллюстрирующим выгодность такой системы хранения данных, является хранилище Web-контента (изображения, видео, JavaScript, CSS и т. п.). Cloud Files предоставляет и дополнительное преимущество публикации этого контента посредством проверенной сети распределения контента (Content Distribution Network, CDN) через инфраструктуру Limelight Networks[1]. Этот подход позволяет новичкам быстро освоиться с CDN и делает сети распределения контента доступными каждому. Пользователи должны просто создать контейнер, закачать в него свои данные, пометить этот контейнер как общедоступный (public) и скомбинировать CDN-URI контейнера с именем объекта, чтобы поставлять информацию в CDN.

Cloud Files прекрасно зарекомендовала себя как система, предоставляющая неограниченное пространство для хранения резервных копий и архивов данных. Возможно, вы один из тех немногих ответственных и аккуратных пользователей, которые создают полные резервные копии всего содержимого своих персональных компьютеров. Даже если это так, то скорее всего, вы выполняете это резервное копирование на внешний жесткий диск или на DVD. Обычно такие носители с резервными копиями хранятся недалеко от самого компьютера и будут утрачены в случае пожара или какого-либо иного катастрофического события. Если же вы скопируете данные в Cloud Files, то вы получите дешевое хранилище, доступ к которому вы сможете получить из любой точки, где есть интернет-соединение.

Как и Amazon S3, Cloud Files представляет собой среду, максимально учитывающую интересы разработчиков. Cloud Files предоставляет Web-сервисы RESTful вместе с API для целого ряда популярных языков программирования. Данные, хранящиеся в системе (объекты), могут включать в свой состав разнообразные индивидуально подобранные метаданные, которые задаются атрибутами, выбираемыми пользователями. Например, фотографии, которые закачивает пользователь, могут быть снабжены метаданными, указывающими, к примеру, к какому фотоальбому эти фотографии принадлежат.

---

[1] Limelight Networks — известный провайдер услуг распределения (доставки) контента (CDN-провайдер). Подробнее см.
http://www.denivip.ru/index.php/cdn_services, **http://uk.limelightnetworks.com/index.php**, **http://en.wikipedia.org/wiki/Limelight_Networks**. — *Прим. перев.*

Сервис Cloud Sites

Хотя в данной книге и не рассматривается хостинг приложений, список услуг в области облачной обработки данных, предлагаемых Rackspace, будет неполным, если не упомянуть такой сервис, как Cloud Sites. Исходный вариант продукта, разработанный Mosso, одним из подразделений, входящих в группу Rackspace, со временем развился в автоматически масштабируемую Web-платформу хостинга, известную как LAMP (Linux, Apache, MySQL, PHP/Perl/Python), которая также предлагает решения и для Microsoft Windows (.NET, ASP, SQL Server). Клиент имеет возможность потреблять каждый из предложенных стеков (Linux, Apache, MySQL, PHP/Perl/Python) или (.NET, ASP, SQL Server) по отдельности, или оба — в комбинации.

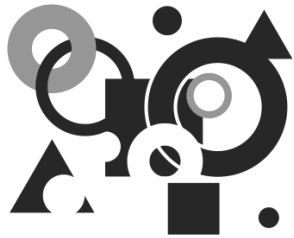
И Amazon EC2, и Slicehost, и Cloud Servers требуют от пользователей, чтобы они самостоятельно планировали и осуществляли конфигурирование своих серверов и приложений, а также управляли их работой. Если все, что вы планируете делать, заключается в поддержании работы Web-сайта на традиционном стеке Web, то сервис Cloud Sites управляет серверной платформой таким образом, чтобы вы могли полностью сконцентрировать свои усилия на вашем Web-сайте и всем, что с ним связано. Технология Cloud Sites даже автоматически масштабирует ваш сайт в зависимости от возрастания или убывания потребностей в ресурсах.

Полная интеграция с обеспечением круглосуточной непрерывной технической поддержки

В 2009 году эти сервисы были интегрированы, и теперь клиентам Rackspace предоставляются надежные и стабильные облачные сервисы. Каждый клиент компании имеет возможность использовать комбинацию физических и виртуальных серверов, "неограниченного" пространства для хранения данных, а также быстрого создания Web-сайтов с возможностями CDN и автоматического масштабирования в зависимости от изменяющихся потребностей.

Rackspace является сервисной компанией, в штате которой работают специалисты-эксперты, предоставляющие круглосуточную техническую поддержку, касающуюся новых технологий, используемых компанией.

Приложение 4



Терминологический глоссарий

A

Acceptable recovery state — приемлемое восстанавливаемое состояние — это состояние (временная точка), до которого необходимо восстановить систему, а затем разработать пошаговые процедуры и способы восстановления этого состояния при ликвидации последствий катастрофы. Определяет объем данных, который вы можете позволить себе потерять в случае наступления катастрофического события.

Amazon EBS (Elastic Block Store) — тип системы хранения данных (storage), позволяющей создавать тома (volumes), которые могут монтироваться к экземплярам Amazon EC2 как устройства (devices). Тома Amazon EBS ведут себя как внешние блочные устройства, не размеченные как дисковые разделы (raw unformatted external block devices). Они имеют имена, указанные пользователями, и предоставляют интерфейс блочных устройств. Вы можете загрузить файловую систему поверх томов Amazon EBS или использовать их точно так же, как использовали бы блочное устройство.

Amazon EBS-backed AMI — экземпляр, запущенный с AMI, использующего том Amazon EBS в качестве корневого устройства. *См. Amazon EBS.*

Amazon Elastic Compute Cloud (Amazon EC2) — Web-сервис, который предоставляет вычислительные мощности в облачной среде Amazon. Сервис входит в инфраструктуру Amazon Web Services (AWS).

Amazon Simple Queue Service (Amazon SQS) — сервис принимает очереди сообщений для хранения. При использовании Amazon SQS разработчики могут просто переместить данные, распределенные между компонентами своих приложений, которые выполняют различные задачи, не теряя при этом сообщения. При этом достигается высокая масштабируемость и надежность.

Amazon machine image (AMI) — образ машины Amazon. Представляет собой зашифрованный образ машины, хранящийся в Amazon S3. Содержит всю информацию, необходимую для того, чтобы запустить экземпляр выбранной вами операционной системы с установленным в ней вашим программным обеспечением в облачной среде Amazon.

Amazon Simple Storage Service (Amazon S3) — онлайновая Web-служба, предлагаемая AWS, предоставляющая возможность для хранения и получения любого

объема данных в любое время из любой точки сети, так называемый файловый хостинг. С помощью Amazon S3 достигается высокая масштабируемость, надежность, высокая скорость и недорогая инфраструктура хранения данных. Впервые появилась в марте 2006 года в США и в ноябре 2007 года в Европе. Amazon S3 используется многими другими сервисами для хранения и хостинга файлов.

Amazon S3-backed AMI — экземпляр, запущенный с AMI, использующего том Amazon S3 в качестве корневого устройства. *См. Instance store.*

Availability Zone — зона доступности. Представляет собой конкретную географическую зону в пределах региона (Region), которая должна быть изолирована от других зон доступности и предоставлять недорогие средства сетевых коммуникаций с низким запаздыванием (latency) к другим зонам доступности в пределах того же самого региона.

B

BitTórrent (букв. "битовый поток") — пиринговый сетевой протокол для кооперативного обмена файлами через Интернет. Файлы передаются частями, каждый torrent-клиент, получая (скачивая) эти части, в то же время отдает (закачивает) их другим клиентам, что снижает нагрузку и зависимость от каждого клиента-источника и обеспечивает избыточность данных.

C

Campaign Management — управление проводимыми рекламными и маркетинговыми акциями.

Chief Financial Officer (CFO) — финансовый директор, один из топ-менеджеров компании, ответственный за управление финансовыми потоками бизнеса, за финансовое планирование и отчетность.

Cloud bursting (англ. "ливень, проливной дождь") — в облачных вычислениях этот термин может использоваться двояко, причем значение его может быть как положительным, так и отрицательным. В отрицательном смысле это означает крах всей облачной среды в момент пиковой нагрузки вследствие неспособности сбалансировать нагрузку между различными компаниями, пользующимися облаком. Описание такой ситуации см. здесь: **http://www.roughtype.com/archives/2007/04/intuits_cloudbu.php**. В положительном же смысле этот термин обозначает технологию динамического развертывания ПО, которое работает, используя внутренние вычислительные ресурсы организации, в общественную облачную инфраструктуру в момент возникновения пиковых нагрузок. Источник: **http://sites.google.com/site/cloudcomputingwiki/Home/cloud-computing-vocabulary**.

Cloud portability — облачная переносимость. Возможность переноса приложений и данных из среды, предоставляемой одним облачным провайдером, в облачную инфраструктуру другого облачного провайдера.

Cloudware — программное обеспечение, позволяющее разрабатывать, развертывать, запускать приложения в облачной среде и управлять ими.

Cluster — кластер. Группа взаимосвязанных компьютеров, взаимодействующих так, как если бы они представляли собой единый компьютер, за счет чего обеспечивается высокая доступность и/или балансировка нагрузки.

Content Delivery Network (CDN) — сеть распределения контента. Система, состоящая из множества компьютеров, содержащих распределенные копии данных, которые клиенты могут копировать с ближайшего к ним компьютера. Использование контент-провайдерами CDN способствует увеличению скорости загрузки интернет-пользователями аудио-, видео-, программного, игрового и других видов цифрового контента в точках присутствия сети CDN.

Content Management System (CMS) — система управления информационным наполнением, система управления контентом.

CRM (Customer Relationship Management) — управление взаимоотношениями с клиентами. CRM-системы представляют собой корпоративные информационные системы, предназначенные для автоматизации стратегии компании, направленной на управление взаимоотношениями с клиентами, в частности для повышения уровня продаж, оптимизации маркетинга и улучшения обслуживания клиентов путем сохранения информации о клиентах/контрагентах и истории взаимоотношений с ними, установления и улучшения бизнес-процедур и последующего анализа результатов.

D

Database engine — движок базы данных. Представляет собой компонент СУБД, управляющий базами данных, или библиотеку, подключаемую к программам и дающую им функции СУБД.

Desktop as a Service (DaaS) — модель распространения и эксплуатации программного обеспечения, получившая известность в начале 2000-х годов и являющаяся логическим продолжением SaaS. *См. SaaS.*

E

Elastic IP address — эластичный IP-адрес. Статический IP-адрес, разработанный для динамических облачных вычислений. Эластичные IP-адреса ассоциируются с вашей учетной записью, а не с конкретными экземплярами. Любой эластичный IP-адрес, который ассоциирован с вашей учетной записью до тех пор, пока вы явно не освободите его. В отличие от традиционных статических IP-адресов, эластичные IP-адреса позволяют вам маскировать сбои экземпляров или даже целых зон доступности за счет быстрого переназначения ваших публичных IP-адресов любому экземпляру, принадлежащему к вашей учетной записи.

enStratus — платформа для управления облачными инфраструктурами, предоставляемая компанией enStratus Networks LLC, дочерней компании Valtira. Компания расположена в Миннеаполисе, штат Миннесота, США.

Ephemeral store (instance store) — эфемерное хранилище экземпляра. При запуске каждого экземпляра он получает свое эфемерное хранилище, срок жизни которого соответствует сроку жизни экземпляра, который оно поддерживает. При завершении работы экземпляра или при потере экземпляра все, что содержалось в эфемерном хранилище этого экземпляра, теряется.

Explicit launch permission — явные полномочия запуска. Полномочия запуска, присвоенные конкретной учетной записи AWS.

F

Fair use — правомерное использование, допустимое, добросовестное использование.

H

Hardware virtualization — аппаратная виртуализация, т. е. виртуализация компьютеров или операционных систем. Она скрывает от пользователей физические особенности вычислительной платформы, представляя вместо них еще одну абстрактную компьютерную платформу. Программное обеспечение, управляющее аппаратной виртуализацией, называется гипервизором (hypervisor) или монитором виртуальных машин (virtual machine monitor).

Hosting — хостинг. Услуга по предоставлению вычислительных мощностей для физического размещения информации на серверах, постоянно находящихся в сети (обычно Интернет). Хостингом также называется услуга по размещению оборудования клиента на территории провайдера с обеспечением подключения его к каналам связи с высокой пропускной способностью. Как правило, услуги хостинга подразумевают, как минимум, услугу размещения файлов сайта на сервере, на котором запущено ПО, необходимое для обработки запросов к этим файлам, включая предоставление места для почтовой корреспонденции, баз данных, DNS, файлового хранилища и т. п., а также поддержка функционирования соответствующих сервисов.

Host-based intrusion detection system (HIDS) — это система обнаружения вторжений, ведущая наблюдение и анализ событий, происходящих внутри системы (в отличие от сетевой системы обнаружения вторжений, которая отслеживает в первую очередь сетевой трафик).

I

IaaS (Infrastructure as a Service) — "инфраструктура как сервис". Предоставление компьютерной инфраструктуры (как правило, виртуализованной формы) как

услуги на основе концепции облачной обработки данных. IaaS состоит из трех основных компонентов:

- аппаратные средства (серверы, системы хранения данных, клиентские системы, сетевое оборудование);
- операционные системы и системное ПО (средства виртуализации, автоматизации, основные средства управления ресурсами);
- связующее ПО (например, для управления системами).

Identity Management — управление идентификационной информацией. Представляет собой область административного управления, которая занимается идентификацией личных данных клиентов и управляет их доступом к ресурсам в пределах системы, ассоциируя пользовательские права и ограничения доступа с помощью личных идентификаторов.

Independent Software Vendors — независимые производители программного обеспечения.

Instance — экземпляр. Как только AMI будет запущен, полученная работающая операционная система будет называться экземпляром. Все экземпляры, основанные на одном и том же образе AMI, идентичны. При останове или сбое экземпляра вся информация, хранящаяся внутри экземпляра, теряется.

Instance store — хранилище экземпляра. Каждый экземпляр имеет фиксированный объем дискового пространства, на котором он может хранить данные. Этот тип хранилища не является постоянным (permanent). Если вам требуется постоянное хранилище, используйте Amazon EBS.

Instance type — тип экземпляра. Представляет собой спецификацию, которая определяет объем памяти, CPU, объем пространства для хранения данных, а также стоимость часа использования экземпляра этого типа. Некоторые из типов экземпляров разработаны для стандартных приложений, в то время, как другие — для приложений, предъявляющих повышенные требования к ресурсам CPU или памяти.

Intrusion Detection System (IDS) — система обнаружения вторжений, программное или аппаратное средство, предназначенное для выявления фактов неавторизованного доступа в компьютерную систему или сеть, либо несанкционированного управления ими в основном через Интернет.

L

Launch permission — полномочие запуска. Атрибут AMI, указывающий, какие учетные записи AWS имеют право запускать конкретный AMI.

M

MSP (Managed Service Providers) — провайдеры управляемых услуг, сервис-провайдеры, предоставляющие услуги внешнего управления инфраструктурой IT

предприятия (от доставки ПО и управления обновлением операционных систем до дистанционного мониторинга и управления всей IT-инфраструктурой компании).

MTBF (Mean Time Between Failures) — среднее время наработки на отказ, среднее время безотказной работы.

Multitenancy — многоарендная архитектура.

N

NAS (Network Attached Storage) — сетевое устройство хранения данных (накопитель, подключенный к сети).

Network intrusion detection system (NIDS) — сетевая система обнаружения вторжений, которая отслеживает такие виды вредоносной деятельности, как DoS-атаки, сканирование портов и попытки проникновения в сеть.

O

OSSEC — бесплатная система обнаружения вторжений (IDS) на уровне хоста, основанная на Open Source. OSSEC обрабатывает информацию о событиях, которая приходит от других систем, от узлов с установленными агентами OSSEC, от межсетевых экранов. На основании правил в формате XML выдаются оповещения о подозрительных действиях.

P

PaaS (Platform as a Service) — "платформа как сервис". Предоставление интегрированной платформы для разработки, тестирования, развертывания и поддержки веб-приложений как услуги, организованная на основе концепции облачной обработки данных.

Paid AMI — платный AMI. Представляет собой образ AMI, который вы продаете другим пользователям Amazon EC2.

Paravirtualization — паравиртуализация, техника виртуализации, при которой гостевые операционные системы подготавливаются для исполнения в виртуализированной среде, для чего их ядро незначительно модифицируется. Операционная система взаимодействует с программой гипервизора, который предоставляет ей гостевой API, вместо использования напрямую таких ресурсов, как таблица страниц памяти.

Pay as you go — оплата по факту потребления. Модель расценок на облачные сервисы, учитывающая модели оплаты по подписке (subscription-based) и по факту потребления (consumption-based), принципиально отличающаяся от модели затрат

на традиционную инфраструктуру, при которой требуются первоначальные капитальные вложения как на аппаратные средства, так и на программное обеспечение.

Private IP address — частный IP-адрес. Все экземпляры Amazon EC2 при запуске получают два IP-адреса: частный и публичный, между ними установлено непосредственное соответствие с помощью протокола трансляции сетевых адресов (Network Address Translation, NAT).

Public AMI — публичный AMI. Тот AMI, права запуска на который имеют все учетные записи AWS.

Public data sets — публичные наборы данных представляют собой такие наборы данных, которые без проблем можно интегрировать в состав любого облачного приложения AWS. Amazon бесплатно хранит наборы данных, представляющие собой достояние сообщества, и при работе с такими наборами вы платите по факту за потребляемые вычислительные мощности и пространство для хранения данных (как, впрочем, и при работе с любыми сервисами AWS). За пользование самими данными, представляющими собой общественное достояние, вы не платите ничего. К таким данным относятся, например, информация, накопленная в ходе работы над проектом по расшифровке генома человека (Human Genome Project), Wikipedia и т. д.

Public IP address — публичный IP-адрес. Все экземпляры Amazon EC2 при запуске получают два IP-адреса: частный и публичный, между ними установлено непосредственное соответствие с помощью протокола трансляции сетевых адресов (Network Address Translation, NAT).

R

Recovery Point Objective, RPO) — момент времени, на который будет восстановлено состояние системы после приведения в действие плана аварийного восстановления *(см. Acceptable Recovery State)*. Этот показатель определяет, какой объем данных вы можете себе позволить потерять в случае наступления катастрофического события. Обычно это значение выражается в количестве часов или рабочих дней, в течение которых шли накопление и обработка данных.

Recovery Time Objective (RTO) — допустимое время восстановления — показатель, определяющий допустимое время простоя в случае наступления катастрофического события.

Region — регион. Географическая область, в которой клиенты могут запускать свои экземпляры (например, US, EU).

REST (Representational State Transfer, "передача состояния представления") — подход к архитектуре сетевых протоколов, обеспечивающих доступ к информационным ресурсам.

RMA (Return Merchandise Authorisation) — разрешение на возврат товара, возврат неисправных или некачественных изделий.

Risk profile — профиль риска, набор характеристик, отражающий вероятность того или иного события и возможные потери, связанные с этим.

S

SaaS (Software as a Service) — программное обеспечение как услуга.

SAN (Storage Area Network) — сеть хранения данных.

SAP AG — немецкая компания, крупнейший в Европе производитель программного обеспечения. Компания занимается разработкой автоматизированных систем управления внутренними процессами предприятия, такими как бухгалтерский учет, торговля, производство, финансы, управление персоналом, управление складами и т. д. Приложения обычно можно адаптировать под правовой контекст определенной страны.

SAS (Statement on Auditing Standards (SAS) No. 70 — широко известный американский аудиторский стандарт, разработанный Американским институтом сертифицированных присяжных бухгалтеров (American Institute of Certified Public Accountants, AICPA).

Scalability — масштабируемость. В информационных технологиях под масштабируемостью понимается способность системы увеличивать свою производительность при добавлении в нее новых ресурсов (обычно аппаратных). Масштабируемость играет важную роль для программных комплексов, баз данных, маршрутизаторов, сетей и т. п., если для них требуется возможность работать под большой нагрузкой.

Security group — группа безопасности. Представляет собой именованную коллекцию прав доступа. Эти права доступа указывают, какие типы входящего трафика могут достигать ваших экземпляров в составе группы. Весь остальной входящий трафик будет заблокирован.

Service level agreement (SLA) — соглашение об уровне предоставления услуги (Service Level Agreement (SLA)) — это формальный договор между заказчиком услуги и ее поставщиком, содержащий описание услуги, права и обязанности сторон и, самое главное, согласованный уровень качества предоставления данной услуги.

Shared AMI — разделяемые AMI. Образы машин Amazon, построенные клиентами и предоставленные в общий доступ другим разработчикам AWS.

Snapshot — Моментальный снимок. Amazon EBS предоставляет возможность создания моментальных снимков или резервных копий ваших томов Amazon EBS и их сохранения в хранилище Amazon S3. Эти моментальные снимки можно использовать в качестве "начальных точек" для создания новых томов Amazon EBS и длительного хранения ваших данных.

SOAP (Simple Object Access Protocol) — "простой протокол доступа к объектам". Протокол обмена структурированными сообщениями в распределенной вычислительной среде.

T

Total Cost of Ownership (TCO) — совокупная стоимость владения.

U

Utility computing — коммунальные вычисления (вычисления как коммерческая услуга), принцип предоставления информационных сервисов как коммунальных услуг.

V

Valtira — компания, основанная автором данной книги, предоставляющая одноименную маркетинговую Web-платформу, интегрированную с инструментами Web-персонализации, системами электронной почты и решениями по управлению взаимоотношениями с клиентами (в том числе такими, как Salesforce.com).

Value-added reseller (VAR) — компания, которая модифицирует или расширяет возможности уже существующего продукта, а затем перепродает его (обычно — конечным пользователям) как новый продукт.

Virtual private cloud (VPC) — виртуальное частное облако, существующее в разделяемой или публичной облачной среде. Например, виртуальное частное облако Amazon (Amazon VPC) позволяет соединять Amazon EC2 с наследуемыми инфраструктурами через протокол IPsec.

W

WebDAV (Web-based Distributed Authoring and Versioning) — защищенный сетевой протокол высокого уровня, работающий поверх HTTP и обеспечивающий доступ к объектам и коллекциям.

Список литературы и ресурсов Интернета

1. Brian McConnell. "Beyond Contact. A Guide to SETI and Communicating with Alien Civilizations" — O'Reilly Media, 2001, ISBN: 978-0-596-00037-0 | ISBN 10: 0-596-00037-5, см. **http://oreilly.com/catalog/9780596000370**.
2. George Reese. "Java Database Best Practices" — O'Reilly Media. 2003. ISBN: 978-0-596-00522-1 | ISBN 10: 0-596-00522-9, см. **http://oreilly.com/catalog/ 9780596005221/ index.html**.
3. Baron Schwartz, Peter Zaitsev, Vadim Tkachenko et.al. "High Performance MySQL, Second Edition. Optimization, Backups, Replication, and More". — O'Reilly Media, 2008. ISBN: 978-0-596-10171-8 | ISBN 10: 0-596-10171-6, см. **http://oreilly.com/catalog/ 9780596101718/index.html**.
4. John Allspaw. "The Art of Capacity Planning: Scaling Web Resources". — O'Reilly Media, 2008. **ISBN-10:** 0596518579 | **ISBN-13:** 978-0596518578, см. **http://tinyurl.com/34mbslw**.
5. John Rittinghouse, James Ransome. "Cloud Computing: Cloud Computing: Implementation, Management, and Security". — CRC Press, 2009. ISBN-10: 1439806802 ISBN-13: 978-1439806807. См. **http://tinyurl.com/25qye27**.
6. James Murty. "Programming Amazon Web Services: S3, EC2, SQS, FPS, and SimpleDB". O'Reilly Media, 2008. ISBN-10: 0596515812 | ISBN-13: 978-0596515812. См. **http://tinyurl.com/333p6a7**.
7. Nancy Conner. "Google Apps: The Missing Manual" — Pogue Press, 2008. ISBN-10: 0596515790 | ISBN-13: 978-0596515799. См. **http://tinyurl.com/36tbj2j**.
8. Edvard L. Haletky. "VMware vSphere and Virtual Infrastructure Security: Securing the Virtual Environment". — Prentice Hall, 2009, ISBN-10: 0137158009 | ISBN-13: 978-0137158003. **См. http://tinyurl.com/38twt4g**.
9. Ronald L. Krutz, Russel Dean Vines. "Cloud Security: A Comprehensive Guide to Secure Cloud Computing". Wiley, 2010. **ISBN-10:** 0470589876 | **ISBN-13:** 978-0470589878. См. **http://tinyurl.com/2v9aphr**.
10. John Rhoton. "Cloud Computing Explained: Implementation Handbook for Enterprises". Recursive Press, 2009. **ISBN-10:** 0956355609 | **ISBN-13:** 978-0956355607. См. **http://tinyurl.com/2ufauke**.

11. Р. Яргер, Дж. Риз, Т. Янг. MySQL и mSQL. Базы данных для небольших предприятий и Интернета. — СПб.: Символ-Плюс, 2000.

Как видите, в этом списке практически нет литературы на русском языке — ни оригинальной, ни переводной. Чтобы восполнить этот недостаток, приведем наиболее интересные и информативные из русскоязычных интернет-ресурсов.

Русскоязычные ресурсы Интернета

- **http://www.smart-cloud.org/** — один из лучших отечественных ресурсов, посвященных облачным вычислениям, где вы найдете самую свежую информацию о Cloud Computing и виртуализации — последние новости, статьи, обсуждения и многое другое.
- **http://www.cloudzone.ru/** — еще один специализированный русскоязычный сайт, посвященный облачным технологиям.
- **http://bokov.net/blog/** — блог Алексея Бокова, содержащий много информации по теме данной книги — Amazon Web Services, с иллюстрациями и практическими примерами работы с AWS.
- **http://grigorkin.ru/2010/03/probuem-elastic-compute-cloud-ec2/** — еще один профессиональный блог, содержащий большое количество информации об облачных вычислениях.
- **http://gogrid.ru/** — русскоязычная версия портала GoGrid. Крайне полезное дополнение к *приложению 2*, в котором как раз и рассказывается о компании GoGrid.
- **http://tinyurl.com/32p27pp** — статья "Технология облачных вычислений как эффективный инструмент поддержки вашего бизнеса" с русскоязычного Web-сайта IBM.
- **http://www.vm4.ru/** — информационный ресурс, посвященный виртуализации с использованием VMware, Xen, MS Hyper-V и других продуктов и технологий.
- **http://tinyurl.com/39v92jg** — раздел журнала BYTE-Россия, посвященный виртуализации и облачным технологиям.
- **http://www.pcidss.ru/about/pcidss/** — интернет-портал, посвященный стандартам безопасности в области платежных карт. Содержит всю необходимую информацию, которую только можно порекомендовать для дополнительного прочтения к *главам 4* и *5*, которая была посвящена безопасности в облачной среде.
- **http://www.pcisecurity.ru/** — еще один портал, посвященный стандартам безопасности, в том числе — и в облачной среде.
- **http://www.iso27000.ru** — портал, посвященный информационной безопасности, ориентированный на менеджеров и экспертов, которые способны видеть и управлять взаимосвязями между бизнесом, информационными технологиями и информационной безопасностью.
- **http://www.securit.ru/solutions/laws/** — сайт компании SecureIT, занимающейся законодательством и стандартами информационной безопасности.

Предметный указатель

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z

А

Б

В

Г

Д

З

И

К

М

Н

О

П

Р

С

Т

У

Ф

Х

Ц

Ч

Ш

Э

Я