УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
«Брестский государственный университет имени А.С. Пушкина»

СОВРЕМЕННЫЕ ОБЛАЧНЫЕ ТЕХНОЛОГИИ И СИСТЕМЫ ВИРТУАЛИЗАЦИИ

Электронный учебно-методический комплекс
для специальности высшего образования второй ступени
(магистратуры):
1-31 80 03 «Математика и компьютерные науки»
физико-математического факультета

Брест
БрГУ имени А.С. Пушкина
2020

Рецензенты:

Кафедра информатики и прикладной математики

Брестского государственного технического университета

доцент кафедры алгебры, геометрии и математического моделирования
учреждения образования
«Брестский государственный университет имени А.С. Пушкина»,
кандидат физико-математических наук, доцент

З.Н. Серая

Кондратюк, А.П.

Современные облачные технологии и системы виртуализации : электронный учебно-методический комплекс для студ. второй ступени (магистратуры) специальности 1-31 80 03 «Математика и компьютерные науки» физ.-мат. фак. / А.П. Кондратюк ; Брест. гос. ун-т им. А.С. Пушкина, каф. ПМИ. – Брест : электрон. издание БрГУ, 2020. – 448 с.

ЭУМК написан в соответствии с действующей базовой программой по дисциплине «Современные облачные технологии и системы виртуализации» и ставит своей целью облегчить самостоятельную работу студентов с теоретическим материалом при подготовке к лекциям и практическим занятиям.

Предназначено для студентов специальности 1-31 80 03 «Математика и компьютерные науки».

СОДЕРЖАНИЕ

Предисловие 7

Глава 1 **Общая организация облачных вычислений** **8**

§1. Введение. Основные определения 8

1.1 Введение. Основные возможности и определения. . 8

Глава 2 **Аппаратная основа облачных технологий** **13**

§2. Развитие аппаратного обеспечения 13

2.1 Развитие аппаратного обеспечения 13

2.2 Появление Blade-систем 19

2.3 Появление систем и сетей хранения данных 26

2.4 Топологии SAN 32

2.5 Консолидация ИТ инфраструктуры 35

Глава 3 **Технологии виртуализации и облачные технологии** **42**

§3. Технологии виртуализации 42

3.1 Преимущества виртуализации 47

3.2 Виртуальная машина 50

3.3 Виртуализация серверов 54

3.4 Полная виртуализация 59

3.5 Паравиртуализация 60

3.6 Виртуализация на уровне ядра ОС 61

Кафедра ПМИ

3.7 Виртуализация приложений 62
3.8 Виртуализация представлений (рабочих мест) . . . 64
3.9 Краткий обзор платформ виртуализации 69
§4. Облачные вычисления 96
4.1 Понятие облачных вычислений. История появления. 96
4.2 Виды облачных вычислений 98
4.3 Инфраструктура как сервис (IaaS) 99
4.4 Платформа как сервис (PaaS) 102
4.5 Программное обеспечение как сервис (SaaS). 105
4.6 Частное облако (private cloud) 112
4.7 Публичное облако 113
4.8 Смешанное (гибридное) облако 113
4.9 Достоинства облачных вычислений 115
4.10 Недостатки и проблемы облачных вычислений . . . 120
4.11 Препятствия развитию облачных технологий 123
4.12 Распределенные вычисления (grid computing) 124
§5. Веб-службы в облачных вычислениях 128
5.1 Инфраструктура как Сервис (IaaS) 128
5.2 Amazon 132
5.3 Платформа как Сервис (PaaS) 139
5.4 Microsoft Azure 143
5.5 Программное обеспечение как Сервис (SaaS) 158
5.6 Коммуникация как Сервис (CaaS) 165

5.7 Мониторинг как Сервис (MaaS) 170
§6. Комплект средств разработки Windows Azure SDK 174
§7. Архитектура Windows Azure Platform 187
7.1 Windows Azure Storage 192
7.2 Azure Table Services 193
7.3 Пример 204
§8. Архитектура Windows Azure Platform. 225
8.1 Azure Blob Services 225
8.2 Azure Queue Services 245
8.3 Пример 257
§9. Технологии Microsoft .NET Services. 266
§10. Предоставление и использование облачных услуг 292
10.1 Office Live Workspace 292
10.2 Web Apps.Microsoft Word Web App 309
10.3 Microsoft Excel Web App 315
10.4 OneDrive 322
10.5 Office 365 329
§11. GoogleApps.App Engine 334
11.1 GoogleApps 334
11.2 App Engine 344

Лабораторный практикум **358**
Лабораторная работа № 1.
Установка и настройка Hyper-V 358
Лабораторная работа № 2.
Установка и настройка VMWare Workstation 363
Лабораторная работа № 3.
Создание первого Windows Azure приложения 368
Лабораторная работа № 4.
Развертывание приложения Windows Azure 389
Лабораторная работа № 5.
Работа с Blob 404
Лабораторная работа № 6.
Работа с Tables 427
Лабораторная работа № 7.
Работа в Windows Live 438
Лабораторная работа № 8.
Работа в Office Live 440
Вопросы к зачету 442

Итоговый тест 446
Литература 447

ПРЕДИСЛОВИЕ

Настоящий электронный учебно-методический комплекс предназначен для студентов специальности 1-31 80 03 «Математика и компьютерные науки» физико-математического факультета. Он написан в соответствии с действующей базовой программой по дисциплине «Современные облачные технологии и системы виртуализации».

В электронном издании излагается теоретический материал, содержащий вопросы: общая организация и аппаратная основа облачных технологий, технологии виртуализации, облачные технологии и веб-службы в облачных вычислениях, комплект средств разработки Windows Azure SDK, архитектура Windows Azure Platform, технологии Microsoft .NET Services, предоставление и использование облачных услуг, GoogleApps.App Engine. Теоретический материал иллюстрируется примерами.

Практический раздел представлен лабораторным практикумом.

Раздел контроля знаний представлен вопросами и тестовыми заданиями.

Учебно-методический комплекс ставит своей целью облегчить самостоятельную работу студентов с теоретическим и практическим материалом при подготовке к лекциям, лабораторным занятиям и зачету.

Автор.

Учебную программу можно посмотреть по ссылке.

ГЛАВА 1
Общая организация облачных вычислений

§1. Введение. Основные определения

1.1 Введение. Основные возможности и определения.

В настоящее время технологии "облачных"вычислений приобретают все большую популярность, а концепция *Cloud Computing* является одной из самых модных тенденций развития информационных технологий. По оценкам Gartner, "облака"– один из главных приоритетов бизнеса на 2010 год. Крупнейшие мировые ИТ вендоры (Microsoft, Amazon, Google и прочие) так или иначе внедряют сервисы "облачных"вычислений.

Сегодня под облачными вычислениями обычно понимают возможность получения необходимых вычислительных мощностей по запросу из сети, причем пользователю не важны детали реализации этого механизма и он получает из этого "облака"все необходимое. Ярким примером могут служить поисковые системы, *интерфейс* которых очень прост, но в то же время они предоставляют пользователям огромные вычислительные ресурсы для поиска нужной информации. Сегодня крупные вычислительные центры не только позволяют хранить и обрабатывать внутри себя определенные данные, но и дают возможности для создания собственных виртуальных дата-центров, позволяя молодым компаниям не тратить силы на создание всей инфраструктуры с нуля. На сего-

дняшний день существует множество определений "облачных вычислений". Зачастую они расходятся в своем значении и акцентах. Рассмотрим некоторые из этих определений для того чтобы понять что такое "облачные"вычисления с разных точек зрения.

Облачные вычисления представляют собой динамически масштабируемый способ доступа к внешним вычислительным ресурсам в виде сервиса, предоставляемого посредством Интернета, при этом пользователю не требуется никаких особых знаний об инфраструктуре "облака"или навыков управления этой "облачной"технологией.

Cloud computing – это программно-аппаратное обеспечение, доступное пользователю через Интернет или локальную сеть в виде сервиса, позволяющего использовать удобный интерфейс для удаленного доступа к выделенным ресурсам (вычислительным ресурсам, программам и данным). *Компьютер* пользователя выступает при этом рядовым терминалом, подключенным к Сети. Компьютеры, осуществляющие *cloud computing*, называются "вычислительным облаком". При этом нагрузка между компьютерами, входящими в "вычислительное облако", распределяется автоматически.

Облачные вычисления – это новый подход, позволяющий снизить сложность ИТ-систем, благодаря применению широкого ряда эффективных технологий, управляемых самостоятельно и доступных по требованию в рамках виртуальной ин-

фраструктуры, а также потребляемых в качестве сервисов. Переходя на частные облака, заказчики могут получить множество преимуществ, среди которых снижение затрат на ИТ, повышение качества предоставления сервиса и динамичности бизнеса.

"Облако"является новой бизнес-моделью для предоставления и получения информационных услуг. Эта модель обещает снизить оперативные и капитальные *затраты*. Она позволяет ИТ департаментам сосредоточиться на стратегических проектах, а не на рутинных задачах управления собственным центром обработки данных.

Облачные вычисления – это не только технологическая *инновация* в ИТ, но и способ создания новых бизнес-моделей, когда у небольших производителей ИТ-продуктов, в том числе и в регионах, появляется возможность быстрого предложения рынку своих услуг и мало затратного способа воплощения своих бизнес-идей. *Поддержка* облачных вычислений в сочетании с инвестициями в молодые компании создают быстро развивающуюся экосистему инновационных производств.

Облачные вычисления являются рыночным ответом на систематическую специализацию и усиление роли аутсорсинга в ИТ. По сути, переход к облачным вычислениям означает *аутсорсинг* традиционных процессов управления ИТ-инфраструктурой профессиональными внешними поставщиками. Большинство современных поставщиков решений сферы облачных вычислений предоставляет возможность не только использовать существующие облачные платформы, но и создавать соб-

ственные, отвечающие технологическим и юридическим требованиям заказчиков. "Облачные вычисления"работают следующим образом: вместо приобретения, установки и управления собственными серверами для запуска приложений, происходит аренда сервера у Microsoft, Amazon, Google или другой компании. Далее *пользователь* управляет своими арендованными серверами через *Интернет*, оплачивая при этом только фактическое их использование для обработки и хранения данных. Вычислительные облака состоят из тысяч серверов, размещенных в датацентрах, обеспечивающих работу десятков тысяч приложений, которые одновременно используют миллионы пользователей. Непременным условием эффективного управления такой крупномасштабной инфраструктурой является максимально полная *автоматизация*. Кроме того, для обеспечения различным видам пользователей – облачным операторам, сервис-провайдерам, посредникам, ИТ-администраторам, пользователям приложений – защищенного доступа к вычислительным ресурсам облачная *инфраструктура* должна предусматривать возможность самоуправления и делегирования полномочий.

Концепция "облачных"вычислений появилась не на пустом месте, а явилась результатом эволюционного развития информационных технологий за последние несколько десятилетий и ответом на вызовы современного бизнеса. Аналитики Гартнер групп (Gartner *Group*) называют "Облачные"вычисления – самой перспективной стратегической технологией будущего, прогнозируя перемещение большей части информацион-

ных технологий в "облака"в течение 5-7 лет. По их оценкам, к 2015 году *объём* рынка облачных вычислений должен был достигнуть отметки 200 миллиардов долларов.

В России технологии "облачных"вычислений делают лишь первые шаги. Несмотря на существующие предложения со стороны крупнейших международных корпораций Microsoft, IBM, Intel, NEC, а также ряда отечественных ИТ-поставщиков спрос на облачные сервисы в России пока невелик. Однако, *по* прогнозу аналитической компании *IDC*, за ближайшие 5 лет рынок облачных услуг в России вырастет более чем на 500 % и составит 113 миллионов долларов.

Перспективы "облачных"вычислений неизбежны, поэтому *знание* об этих технологиях необходимо любому специалисту, который связывает свою текущую или будущую *деятельность* с современными информационными технологиями.

ГЛАВА 2
Аппаратная основа облачных технологий

§2. Развитие аппаратного обеспечения

Целью данной лекции является знакомство с основными этапами развития вычислительной техники. *Анализ* современных тенденций развития аппаратного обеспечения, приведших к появлению технологий облачных вычислений.

2.1 Развитие аппаратного обеспечения

Для того, чтобы понять, как появились "облачные"вычисления, необходимо представлять основные моменты процесса развития вычислений и вычислительной техники.

В наше время жизнь без компьютеров не представляется возможной. Внедрение вычислительной техники проникло почти во все жизненные аспекты, как личные, так и профессиональные. Развитие компьютеров было достаточно быстрым. Началом эволюционного развития компьютеров стал 1930 год, когда двоичная арифметика была разработана и стала основой компьютерных вычислений и языков программирования. В 1939 году были изобретены электронно-вычислительные машины, выполняющие *вычисление* в цифровом виде. Появление вычислительных устройств приходится на 1942 год, когда было изобретено устройство, ко-

торое могло механически добавлять числа. Вычисления производились с использованием электронных ламп.

Появившаяся в 1941 году модель Z3 Конрада Цузе в немецкой Лаборатории Авиации в Берлине была одним из наиболее значительных событий в развитии компьютеров, потому что эта машина поддерживала вычисления как с плавающей точкой, так и двоичную арифметику. Это устройство рассматривают как самый первый *компьютер*, который был полностью работоспособным. *Язык программирования* считают "Turing-complete", если он попадает в тот же самый вычислительный *класс*, как *машина Тьюринга*.

Первое поколение современных компьютеров появилось в 1943, когда были разработаны Марк I и машина Колосс. С финансовой поддержкой от *IBM (International* Business Machines Corporation) Марк был сконструирован и разработан в Гарвардском университете. Это был электромеханический программируемый *компьютер* общего назначения. Первое поколение компьютеров было построено с использованием соединенных проводов и электронных ламп (термоэлектронных ламп). Данные хранились на бумажных перфокартах. Колосс использовался во время Второй мировой войны, чтобы помочь расшифровать зашифрованные сообщения.

Чтобы выполнить его задачу расшифровки, Колосс сравнил два потока данных, прочитанных на высокой скорости с перфоленты. Колосс оценивал *поток* данных, считая каждое совпадение, которое было обна-

ружено, основываясь на программируемой Булевой функции. Для сравнения с другими данными был создан отдельный *поток*.

Другой *компьютер* общего назначения этой эры был ENIAC (Электронный Числовой *Интегратор* и *Компьютер*), который был построен в 1946. Он был первым компьютером, способным к перепрограммированию, чтобы решать полный спектр вычислительных проблем. ENIAC содержал 18 000 термоэлектронных ламп, он весил более чем 27 тонн, и потреблял электроэнергии 25 киловатт в час. ENIAC выполнял 100 000 вычислений в секунду. Изобретение транзистора означало, что неэффективные термоэлектронные лампы могли быть заменены более мелкими и надежными компонентами. Это было следующим главным шагом в истории вычислений.

Компьютеры *Transistorized* отметили появление второго поколения компьютеров, которые доминировали в конце 1950-ых и в начале 1960-ых. Несмотря на использование транзисторов и печатных схем, эти компьютеры были все еще большими и дорогостоящими. В основном они использовались университетами и правительством. *Интегральная схема* или чип были развиты Джеком Килби. Благодаря этому достижению он получил Нобелевскую премию по физике в 2000 году.

Изобретение Килби вызвало взрыв в развитии компьютеров третьего поколения. Даже при том, что первая *интегральная схема* была произведена в сентябре 1958, чипы не использовались в компьютерах до 1963. Историю мейнфреймов принято отсчитывать с появления в 1964 году

универсальной компьютерной системы IBM System/360, на разработку которой *корпорация IBM* затратила 5 млрд долларов.

Мейнфрейм – это главный компьютер вычислительного центра с большим объемом внутренней и внешней памяти. Он предназначен для задач, требующих сложных вычислительных операций. Сам термин *"мейнфрейм"* происходит от названия типовых процессорных стоек этой системы. В 1960-х – начале 1980-х годов System/360 была безоговорочным лидером на рынке. Её клоны выпускались во многих странах, в том числе – в СССР (серия ЕС ЭВМ). В то время такие мэйнфреймы, как IBM 360 увеличили способности хранения и обработки, интегральные схемы позволяли разрабатывать миникомпьютеры, что позволило большому количеству маленьких компаний производить вычисления. *Интеграция* высокого уровня диодных схем привела к развитию очень маленьких вычислительных единиц, что привело к следующему шагу развития вычислений.

В ноябре 1971 Intel выпустили первый в мире коммерческий *микропроцессор*, Intel 4004. Это был первый полный центральный *процессор* на одном чипе и стал первым коммерчески доступным микропроцессором. Это было возможно из-за развития новой технологии кремниевого управляющего электрода. Это позволило инженерам объединить на много большее число транзисторов на чипе, который выполнял бы вычисления на небольшой скорости. Эта разработка способствовала появлению компьютерных платформ четвертого поколения.

Компьютеры четвертого поколения, которые развивались в это время, использовали *микропроцессор*, который помещает способности компьютерной обработки на единственном чипе. Комбинируя *память* произвольного доступа (RAM), разработанную Intel, компьютеры четвертого поколения были быстрее, чем когда-либо прежде и занимали на много меньшую *площадь*. Процессоры Intel 4004 были способны выполнять всего 60 000 инструкций в секунду. Микропроцессоры, которые развились из Intel 4004 разрешенные изготовителями, стали базой для начала развития персональных компьютеров, маленьких достаточно дешевых, чтобы быть купленными широкой публикой. Первым коммерчески доступным персональным компьютером был MITS Altair 8800, выпущенный в конце 1974. В последствии были выпущены такие персональные компьютеры, как Apple I и II, Commodore PET, VIC-20, Commodore 64, и, в конечном счете, оригинальный IBM-PC в 1981. Эра PC началась всерьез к середине 1980-ых. В течение этого времени IBM-PC, Commodore Amiga и Atari ST были самыми распространенными платформами PC, доступными общественности. Даже при том, что микровычислительная *мощность* и *память* увеличились на много порядков, начиная с изобретения из Intel 4004 процессоров, технологии чипов интеграции высокого уровня *(LSI)* или *интеграция* сверхвысокого уровня *(VLSI)* сильно не изменились. Поэтому большинство сегодняшних компьютеров все еще попадает в категорию компьютеров четвертого поколения.

Одновременно с резким ростом производства персональных компью-

теров в начале 1990-х начался кризис рынка мейнфреймов, пик которого пришёлся на 1993 год. Многие аналитики заговорили о полном вымирании мейнфреймов, о переходе от централизованной обработки информации к распределённой (с помощью персональных компьютеров, объединённых двухуровневой архитектурой "клиент-*сервер*"). Многие стали воспринимать мейнфреймы как вчерашний день вычислительной техники, считая Unix- и PC-серверы более современными и перспективными.

С 1994 года вновь начался рост интереса к мейнфреймам. Дело в том, что, как показала практика, централизованная обработка на основе мейнфреймов решает многие задачи построения информационных систем масштаба предприятия проще и дешевле, чем распределённая. Многие из идей, заложенных в концепции облачных вычислений также "возвращают"нас к эпохе мэйнфреймов, разумеется с поправкой на время. Еще шесть лет назад в беседе с Джоном Мэнли, одним из ведущих научных сотрудников центра исследований и разработок HP в Бристоле, обсуждалась тема облачных вычислений, и Джон обратил внимание на то, что основные идеи *cloud computing* до боли напоминают мэйнфреймы, только на другом техническом уровне: "Все идет от мэйнфреймов. Мэйнфреймы научили нас тому, как в одной среде можно изолировать приложения, – умение, критически важное сегодня".

С каждым годом требования бизнеса к непрерывности предоставления сервисов возрастают, а на устаревшем оборудовании обеспечить бес-

перебойное функционирование практически невозможно. В связи с этим крупнейшие ИТ-вендоры производят и внедряют более функциональные и надежные аппаратные и программные решения. Рассмотрим основные тенденции развития инфраструктурных решений, которые, так или иначе, способствовали появлению концепции облачных вычислений.

- Рост производительности компьютеров. Появление многопроцессорных и многоядерных вычислительных систем, развитие Blade-систем
- Появление систем и сетей хранения данных
- Консолидация инфраструктуры

2.2 Появление Blade-систем

В процессе развития средств вычислительной техники всегда существовал большой класс задач, требующих высокой концентрации вычислительных средств. К ним можно отнести, например сложные ресурсоемкие вычисления (научные задачи, математическое моделирование), а так же задачи по обслуживанию большого числа пользователей (распределенные базы данных, Интернет-сервисы, хостинг).

Не так давно (порядка 5-ти лет назад) производители процессоров достигли разумного ограничения наращивания мощности процессора, при котором его производительность очень высока при относительно низкой

стоимости. При дальнейшем увеличении мощности процессора, необходимо было прибегать к нетрадиционным методам охлаждения процессоров, что достаточно неудобно и дорого. Оказалось, что для увеличения мощности вычислительного центра более эффективно увеличить количество отдельных вычислительных модулей, а не их производительность. Это привело к появлению многопроцессорных, а позднее и многоядерных вычислительных систем. Появляются многопроцессорные системы, которые насчитывают более 4 процессоров. На текущий момент существуют процессоры с количеством ядер 8 и более, каждое из которых эквивалентно по производительности. Увеличивается количество слотов для *подключения модулей* оперативной памяти, а также их емкость и скорость.

Увеличение числа вычислительных модулей в вычислительном центре требует новых подходов к размещению серверов, а также приводит к росту затрат на помещения для центров обработки данных, их электропитание, охлаждение и обслуживание.

Для решения этих проблем был создан новый тип серверов XXI века – модульные, чаще называемые Blade-серверами, или серверами-лезвиями (blade – лезвие). Преимущества Blade-серверов, первые модели которых были разработаны в 2001 г. изготовители описывают с помощью правила "1234". "По сравнению с обычными серверами при сравнимой производительности Blade-серверы занимают в два раза меньше места, потребляют в три раза меньше энергии и обходятся в четыре раза дешевле".

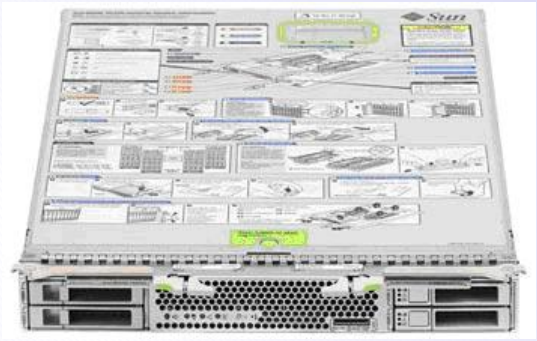


Рисунок 2.1: Типичный Blade-сервер (Sun Blade X6250)

Что представляет собой Blade-сервер? По определению, данному аналитической компании IDC *Blade-сервер или лезвие – это модульная одноплатная компьютерная система, включающая процессор и память*. Лезвия вставляются в специальное *шасси* с объединительной панелью (*backplane*), обеспечивающей им подключение к сети и подачу электропитания. **Это шасси с лезвиями, является Blade-системой**. Оно выполнено в конструктиве для установки в стандартную 19-дюймовую стойку и в зависимости от модели и производителя, занимает в ней 3U, 6U или 10U (один U – unit, или монтажная единица, равен 1,75 дюйма). За счет общего использования таких компонентов, как источники питания, сетевые карты и жесткие диски, Blade-серверы обеспечивают более высокую плотность размещения вычислительной мощности в стойке по сравнению с обычными тонкими серверами высотой 1U и 2U.

Рисунок 2.2: Типичное 10U шасси для 10 Blade-серверов (Sun Blade 6000) используемое в БрГУ

Технология Blade-систем заимствует некоторые черты мейнфреймов. В настоящее время лидером в производстве Blade-систем являются компании Hewlett-Packard, IBM, Dell, Fujitsu Siemens Computers, Sun.

Преимущества Blade-серверов

Рассмотрим основные преимущества Blade-систем:

Уникальная физическая конструкция. Архитектура Blade-систем основана на детально проработанной уникальной физической конструкции. Совместное использование таких ресурсов, как средства питания, охлаждения, коммутации и управления, снижает сложность

и ликвидирует проблемы, которые характерны для более традиционных стоечных серверных инфраструктур. Физическая конструкция Blade систем предполагает размещение Blade серверов в специальном *шасси* и основным ее конструктивным элементом является объединительная панель. Объединительная панель разработана таким образом, что она решает все задачи коммутации Blade серверов с внешним миром: с сетями Ethernet, сетями хранения данных Fiber Channel, а также обеспечивает взаимодействие по протоколу SAS (SCSI) с дисковыми подсистемами в том же *шасси*. *Шасси* для Blade-ов также позволяет размещать в нем необходимые коммутаторы Ethernet или Fiber Channel для связи с внешними сетями. Выход на эти коммутаторы из Blade серверов обеспечивают предустановленные или устанавливаемые дополнительно контроллеры. Средства коммутации во внешние сети, интегрированные в общую полку, значительно сокращают количество кабелей для подключения к ЛВС и SAN, чем традиционным стоечным серверам. Blade сервера имеют общие средства питания и охлаждения. Размещение систем питания и охлаждения в общей полке, а не в отдельных серверах, обеспечивает снижение энергопотребления и повышение надежности.

Лучшие возможности управления и гибкость. Blade-серверы принципиально отличаются от стоечных серверов тем, что серверная полка имеет интеллект в виде модулей управления, который отсутствует в стойках при размещении традиционных серверов. Для управления системой не требуется клавиатура, видео и мышь. Управление Blade си-

стемой осуществляется с помощью централизованного модуля управления и специального процессора удаленного управления на каждом Blade-сервере. Система управления *шасси* и серверами как правило имеют достаточно удобное программное обеспечение для управления. Появляются возможности удаленно управлять всей "Blade"-системой, в том числе управление электропитанием и сетью отдельных узлов.

Масштабируемость – при необходимости увеличение производительных мощностей, достаточно приобрести дополнительные лезвия и подключить к *шасси*. Серверы и инфраструктурные элементы в составе Blade-систем имеют меньший размер и занимают меньше места, чем аналогичные стоечные решения, что помогает экономить электроэнергию и пространство, выделенное для ИТ. Кроме того, благодаря модульной архитектуре, они являются более удобными во внедрении и модернизации.

Повышенная надежность. В традиционных стоечных средах для повышения надежности устанавливается дополнительное оборудование, средства коммутации и сетевые компоненты, обеспечивающие резервирование, что влечет за собой дополнительные расходы. Blade-системы имеют встроенные средства резервирования, например предполагается наличие нескольких блоков питания, что позволяет при выходе из строя одного блока питания, обеспечивать бесперебойную работу всех серверов, расположенных в *шасси*. Также дублируются и охлаждающие компоненты. Выход из строя одного из вентиляторов не приводит к критическим последствиям. При выходе одного сервера из строя системный

администратор просто заменяет лезвие на новое и затем в дистанционном режиме инсталлирует на него ОС и прикладное ПО.

Снижение эксплуатационных расходов. Применение Blade-архитектуры приводит к уменьшению энергопотребления и выделяемого тепла, а также к уменьшению занимаемого объема. Помимо уменьшения занимаемой площади в ЦОД, экономический эффект от перехода на лезвия имеет еще несколько составляющих. Поскольку в них входит меньше компонентов, чем в обычные стоечные серверы, и они часто используют низковольтные модели процессоров, что сокращаются требования к энергообеспечению и охлаждению машин. Инфраструктура Blade-систем является более простой в управлении, чем традиционные ИТ-инфраструктуры на стоечных серверах. В некоторых случаях Blade-системы позволили компаниям увеличить количество ресурсов под управлением одного администратора (серверы, коммутаторы и системы хранения) более чем в два раза. Управляющее программное обеспечение помогает ИТ-организациям экономить время благодаря возможности эффективного развертывания, мониторинга и контроля за инфраструктурой Blade-систем. Переход к серверной инфраструктуре, построенной из лезвий, позволяет реализовать интегрированное управление системы и отойти от прежней схемы работы Intel-серверов, когда каждому приложению выделялась отдельная машина. На практике это означает значительно более рациональное использование серверных ресурсов, уменьшение числа рутинных процедур (таких, как подключение

кабелей), которые должен выполнять системный администратор, и экономию его рабочего времени.

2.3 Появление систем и сетей хранения данных

Другой особенностью современной истории развития вычислительных систем, наряду с появлением Blade-серверов, стало появления специализированных систем и сетей хранения данных. Внутренние подсистемы хранения серверов часто уже не могли предоставить необходимый уровень масштабируемости и производительности в условиях лавинообразного наращивания объемов обрабатываемой информации. В итоге появились внешние системы хранения данных, ориентированные сугубо на решение задач хранения данных и предоставление интерфейса доступа к данным для их использования.

Система Хранения Данных (СХД) – это программно-аппаратное решение по организации надёжного хранения информационных ресурсов и предоставления к ним гарантированного доступа.

Системы хранения данных представляют собой надежные устройства хранения, выделенные в отдельный узел. Система хранения данных может подключаться к серверам многими способами. Наиболее производительным является подключение по оптическим каналам (Fiber Channel), что дает возможность получать доступ к системам хранения данных со

скоростями 4-8 Гбит/сек. Системы хранения данных так же имеют резервирование основных аппаратных компонент – несколько блоков питания, raid контроллеров, FC адаптеров и оптических патчкордов для подключения к FC коммутаторам.

Рисунок 2.3: Типичная Система хранения данных начального уровня (Sun StorageTek 6140)

Отметим основные преимущества использования СХД:

Высокая надёжность и отказоустойчивость – реализуется полным или частичным резервированием всех компонент системы (блоков питания, путей доступа, процессорных модулей, дисков, кэша и т.д.), а также мощной системой мониторинга и оповещения о возможных и существующих проблемах;

Высокая доступность данных – обеспечивается продуманными функциями сохранения целостности данных (использование технологии RAID, создание полных и мгновенных копий данных внутри дисковой

стойки, реплицирование данных на удаленную СХД и т.д.) и возможностью добавления (обновления) аппаратуры и программного обеспечения в беспрерывно работающую систему хранения данных без остановки комплекса;

Мощные средства управления и контроля – управление системой через web-интерфейс или командную строку, выбор нескольких вариантов оповещения администратора о неполадках, полный мониторинг системы, работающая на уровне "железа"технология диагностики производительности;

Высокая производительность – определяется числом жёстких дисков, объёмом кэш-памяти, вычислительной мощностью процессорной подсистемы, числом внутренних (для жёстких дисков) и внешних (для подключения хостов) интерфейсов, а также возможностью гибкой настройки и конфигурирования системы для работы с максимальной производительностью;

Беспроблемная масштабируемость – обычно существует возможность наращивания числа жёстких дисков, объёма кэш-памяти, аппаратной модернизации существующей системы хранения данных, наращивания функционала с помощью специального ПО, работающего на стойке, без значительного переконфигурирования или потерь какой-то функциональности СХД. Этот момент позволяет значительно экономить и более гибко проектировать свою сеть хранения данных.

Сегодня системы хранения данных являются одним из ключевых эле-

ментов, от которых зависит непрерывность бизнес-процессов компании. В современной корпоративной ИТ-инфраструктуре СХД, как правило, отделены от основных вычислительных серверов, адаптированы и настроены для различных специализированных задач. Системы хранения данных реализуют множество функций, они играют важную роль в построении систем оперативного резервного копирования и восстановления данных, отказоустойчивых кластеров, высоко доступных ферм виртуализации.

Сети хранения данных

SAN – это высокоскоростная коммутируемая сеть передачи данных, объединяющая серверы, рабочие станции, дисковые хранилища и ленточные библиотеки. Обмен данными происходит по протоколу Fibre Channel, оптимизированному для быстрой гарантированной передачи сообщений и позволяющему передавать информацию на расстояние от нескольких метров до сотен километров.

Движущей силой для развития сетей хранения данных стал взрывной рост объема деловой информации (такой как электронная почта, базы данных и высоконагруженные файловые сервера), требующей высокоскоростного доступа к дисковым устройствам на блочном уровне. Ранее на предприятии возникали "острова"высокопроизводительных дисковых массивов SCSI. Каждый такой массив был выделен для конкретного приложения и виден ему как некоторое количество "виртуальных жестких дисков". Сеть хранения данных (Storage Area Network или SAN)

позволяет объединить эти "острова"средствами высокоскоростной сети. Основу SAN составляет волоконно-оптическое соединение устройств по интерфейсу Fibre Chanel, обеспечивающее скорость передачи информации между объектами 1,2,4 или 8 Gbit/sec. Сети хранения помогают повысить эффективность использования ресурсов систем хранения, поскольку дают возможность выделить любой ресурс любому узлу сети. Рассмотрим основные преимущества SAN:

- **Производительность**. Технологии SAN позволяют обеспечить высокую производительность для задач хранения и передачи данных.
- **Масштабируемость**. Сети хранения данных обеспечивают удобство расширения подсистемы хранения, позволяют легко использовать приобретенные ранее устройства совместно с новыми устройствами хранения данных.
- **Гибкость**. Совместное использование систем хранения данных, как правило, упрощает администрирование и добавляет гибкость, поскольку кабели и дисковые массивы не нужно физически транспортировать и перекоммутировать от одного сервера к другому. SAN позволяет подключить новые серверы и дисковые массивы к сети без остановки системы.
- **Централизованная загрузка**. Другим преимуществом является возможность загружать сервера прямо из сети хранения. При такой конфигурации можно быстро и легко заменить сбойный сервер,

переконфигурировав SAN таким образом, что сервер-замена, будет загружаться с логического диска сбойного сервера.

- **Отказоустойчивость**. Сети хранения помогают более эффективно восстанавливать работоспособность после сбоя. В SAN может входить удаленный участок с вторичным устройством хранения. В таком случае можно использовать репликацию – реализованную на уровне контроллеров массивов, либо при помощи специальных аппаратных устройств. Спрос на такие решения значительно возрос после событий 11 сентября 2001 года в США.
- **Управление**. Технологии SAN позволяют обеспечить централизованное управление всей подсистемой хранения данных.

2.4 Топологии SAN

Рассмотрим некоторые топологии сетей хранения данных:

Однокоммутаторная структура (англ. single-switch fabric) состоит из одного коммутатора Fibre Channel, сервера и системы хранения данных. Обычно эта топология является базовой для всех стандартных решений – другие топологии создаются объединением однокоммутаторных ячеек.

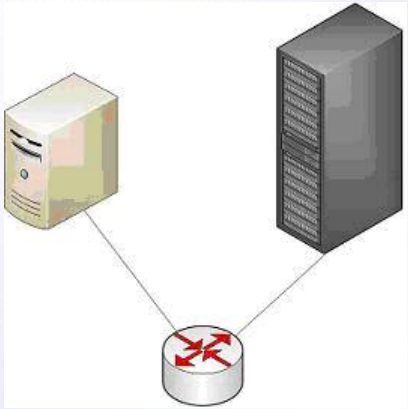


Рисунок 2.4: Однокоммутаторная структура SAN

Каскадная структура – набор ячеек, коммутаторы которых соединены в дерево с помощью межкоммутаторных соединений.

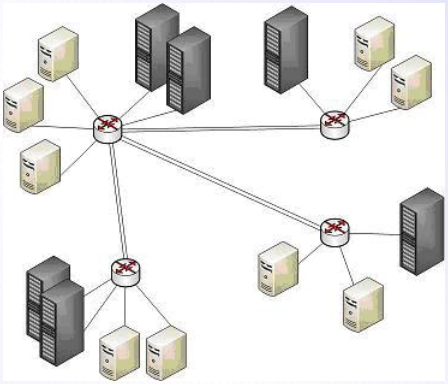


Рисунок 2.5: Каскадная структура SAN

Решетка – набор ячеек, коммутатор каждой из которых соединен со всеми другими. При отказе одного (а в ряде сочетаний – и более) соединения связность сети не нарушается. Недостаток – большая избыточность соединений.

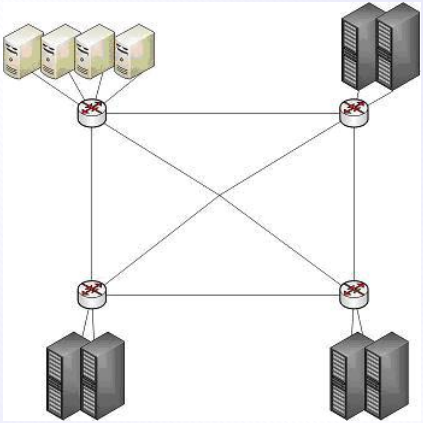


Рисунок 2.6: Структура Решетка

Кольцо – практически повторяет схему топологии решётка. Среди преимуществ – использование меньшего количества соединений.

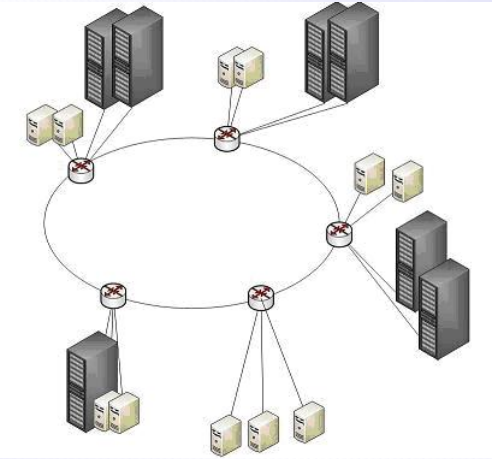


Рисунок 2.7: Структура Кольцо

2.5 Консолидация ИТ инфраструктуры

Консолидация – это объединение вычислительных ресурсов либо структур управления в едином центре.

Анализ международного опыта позволяет сегодня говорить о четкой тенденции к консолидации ИТ-ресурсов корпораций. Именно она способна существенно уменьшить затраты на ИТ. Сэкономленные же средства

можно направить на повышение качества имеющихся информационных услуг и внедрение новых. Кроме оптимизации расходов на ИТ, консолидация ИТ-ресурсов позволяет улучшить управляемость предприятий за счет более актуальной и полной информации об их функционировании. Обычно говорят о консолидации:

- **серверов** – перемещение децентрализованных, приложений, распределенных на различных серверах компании, в один кластер централизованных гомогенных серверов;
- **систем хранения** – совместное использование централизованной системы хранения данных несколькими гетерогенными узлами;
- **приложений** – размещение нескольких приложений на одном хосте.

При этом можно выделить два базовых типа консолидации – физическую и логическую. Физическая консолидация подразумевает географическое перемещение серверов на единую площадку (в центр данных), а логическая – централизацию управления.

Перемещение компьютеров в единый центр обработки данных позволяют обеспечить комфортные условия для оборудования и технического персонала, а также увеличить степень физической защиты серверов. Кроме того, в центре обработки данных можно использовать более производительное и высококачественное оборудование, которое экономически неэффективно устанавливать в каждом подразделении. Создавая

центры обработки данных, можно снизить расходы на техническую поддержку и управление самыми важными серверами предприятия. Удачным примером оборудования, которое может успешно решить задачи консолидации вычислительных ресурсов в организациях любого уровня являются Blade-системы, а также и системы и сети хранения данных.

Очевидное преимущество этого решения в том, что упрощается выделение персонала поддержки и его работа по развертыванию и управлению системами, снижается степень дублирования опытных кадров. Централизация также облегчает использование стандартизованных конфигураций и процессов управления, создание рентабельных систем резервного копирования для восстановления данных после сбоя и поддержания связности бизнеса. Упрощается и решение вопросов организации высококачественного контроля за состоянием окружающей среды и обеспечения физической защиты. Может быть улучшена и сетевая безопасность, поскольку серверы оказываются под защитой единого, централизованно управляемого межсетевого экрана.

Логический тип консолидации подразумевает перестройку системы управления ИТ-инфраструктуры. Это необходимо как для увеличения масштабируемости и управляемости сложной распределенной вычислительной системы, так и для объединения сегментов корпоративной сети. Логическая консолидация обеспечивает введение централизованного управления и унификацию работы с ресурсами компании на основе открытых стандартов. В результате появляется возможность создания гло-

бальных информационных служб предприятия – каталога LDAP, корпоративного портала или ERP-системы, что в конечном итоге позволит улучшить управляемость предприятия за счет более актуальной и полной информации об его функционировании.

Логическая консолидация приложений приводит к централизации управления критическими для бизнеса системами и приложениями. Преимущества логической консолидации очевидны: в первую очередь это высвобождение аппаратных ресурсов, которые можно использовать на других участках информационной системы. Во-вторых, более простая и логичная структура управления ИТ-инфраструктурой делает ее более гибкой и приспособленной для будущих изменений.

Сценарий гомогенной консолидации предусматривает перенос одного масштабного приложения, ранее выполнявшегося на нескольких серверах, на один, более мощный (рисунок 2.8). В качестве примера такой операции можно привести базы данных, которые зачастую наращивают экстенсивным путем по мере роста объема обрабатываемой информации. Объединение данных и приложений на одном сервере заметно ускоряет процессы обработки и поиска, а также повышает уровень целостности.

Гетерогенная консолидация по содержанию схожа с гомогенной, но в этом случае объединению подлежат разные приложения. Например, несколько экземпляров Exchange Server и SQL Server, ранее запускавшиеся на отдельных компьютерах, могут быть сведены на единой машине. Преимущества гетерогенной консолидации – возрастающая масштаби-

руемость сервисов и более полное задействование системных ресурсов.

Рисунок 2.8: Консолидация приложений

Как отмечают специалисты по облачным технологиям – консолидация ИТ-инфраструктуры – является первым шагом к "облаку". Чтобы перейти к использованию облачных технологий, компаниям необходимо сначала решить задачи неконсолидированной ИТ-инфраструктуры. "Без консолидации невозможно построить эффективное процессно-ориентированное управление, поскольку отсутствует единая точка предоставления сервисов".

Анализируя историю развития информационных технологий и современные тенденции можно сделать вывод, что эволюционный виток ИТ, начавшийся вместе с эпохой мэйнфреймов более пятидесяти лет назад, замкнулся – вместе с облаками мы вернулись к централизации ресурсов,

но на этот раз не на уровне мэйнфреймов с их зелеными терминалами на новом технологическом уровне.

Выступая на конференции, посвященной проблемам современных процессоров, профессор Массачусетского технологического института Ананд Агарвал сказал: "Процессор – это транзистор современности". Новый уровень отличается тем, что здесь также собираются мэйнфреймы, но виртуальные, и не из отдельных транзисторов, как полвека назад, а из целых процессоров или целиком из компьютеров. На заре ИТ многочисленные компании и организации "лепили"собственные компьютеры из дискретных компонентов, монтируя их на самодельных печатных платах – каждая организация делала свою машину, и ни о какой стандартизации или унификации и речи не могло быть. И вот на пороге второго десятилетия XXI века ситуация повторяется – точно так же из серверов-лезвий, компьютеров, разнообразного сетевого оборудования собираются внешние и частные облака. Одновременно наблюдается та же самая технологическая разобщенность и отсутствие унификации: Microsoft, Google, IBM, Aptana, Heroku, Rackspace, Ning, Salesforce строят глобальные мэйнфреймы, а кто-то под собственные нужды создает частные облака, которые являются теми же мэйнфреймами, но меньшего масштаба. Остается предположить, что впереди изобретение интегральной схемы и микропроцессора.

Краткие итоги:

В данной лекции мы ознакомились с основными моментами историче-

ского развития средств вычислительной техники. Рассмотрели тенденции современных инфраструктурных решений.

Ключевые термины:

Мейнфрейм – это главный *компьютер* вычислительного центра с большим объемом внутренней и внешней памяти.

Blade-сервер – компьютерный *сервер* с компонентами, вынесенными и обобщёнными в корзине для уменьшения занимаемого пространства.

Система Хранения Данных (СХД) – это программно-аппаратное решение по организации надёжного хранения информационных ресурсов и предоставления к ним гарантированного доступа.

SAN – это высокоскоростная коммутируемая *сеть передачи данных*, объединяющая серверы, рабочие станции, дисковые хранилища и ленточные библиотеки. *Обмен данными* происходит по протоколу Fibre *Channel*, оптимизированному для быстрой гарантированной передачи сообщений и позволяющему передавать информацию на *расстояние* от нескольких метров до сотен километров.

Консолидация – это *объединение* вычислительных ресурсов либо структур управления в едином центре.

ГЛАВА 3

Технологии виртуализации и облачные технологии

Цель данной лекции – получить сведения о технологиях виртуализации, терминологии, разновидностях и основных достоинствах виртуализации. Ознакомиться с основными решениями ведущих ИТ-вендоров. Рассмотреть особенности платформы виртуализации Microsoft.

§3. Технологии виртуализации

Согласно статистике средний уровень загрузки процессорных мощностей у серверов под управлением *Windows* не превышает 10 %, у Unix-систем этот показатель лучше, но тем не менее в среднем не превышает 20 %. Низкая эффективность использования серверов объясняется широко применяемым с начала 90-х годов подходом "одно *приложение* – один *сервер* т. е. каждый раз для развертывания нового приложения компания приобретает новый *сервер*. Очевидно, что на практике это означает быстрое увеличение серверного парка и как следствие – возрастание затрат на его *администрирование*, *энергопотребление* и *охлаждение*, а также потребность в дополнительных помещениях для установки всё новых серверов и приобретении лицензий на серверную ОС.

Виртуализация ресурсов физического сервера позволяет гибко рас-

пределять их между приложениями, каждое из которых при этом "видит"только предназначенные ему ресурсы и "считает что ему выделен отдельный *сервер*, т. е. в данном случае реализуется подход "один *сервер* – несколько приложений но без снижения производительности, доступности и безопасности серверных приложений. Кроме того, решения виртуализации дают возможность запускать в разделах разные ОС с помощью эмуляции их системных вызовов к аппаратным ресурсам сервера.

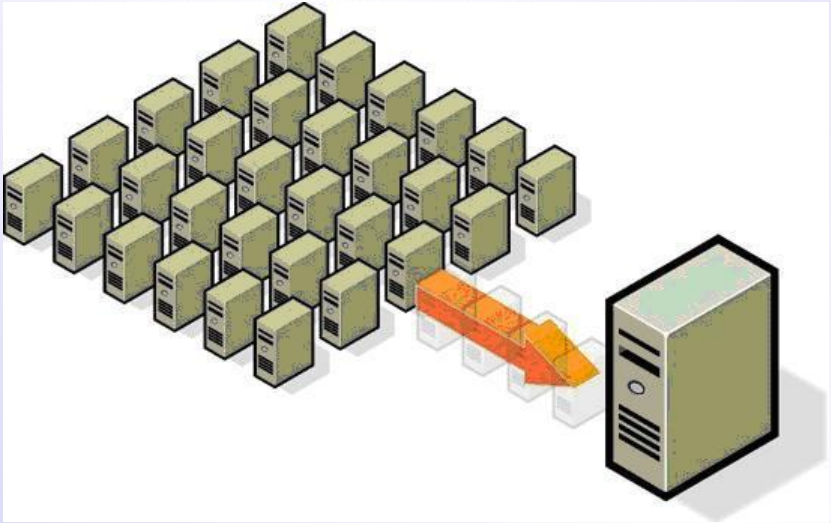


Рисунок 3.1: Виртуализация подразумевает запуск на одном физическом компьютере нескольких виртуальных компьютеров

В основе виртуализации лежит возможность одного компьютера выполнять работу нескольких компьютеров благодаря распределению его ресурсов *по* нескольким средам. С помощью виртуальных серверов и виртуальных настольных компьютеров можно разместить несколько ОС и несколько приложений в едином местоположении. Таким образом, физические и географические ограничения перестают иметь какое-либо *значение*. Помимо энергосбережения и сокращения расходов благодаря более эффективному использованию аппаратных ресурсов, виртуальная *инфраструктура* обеспечивает высокий уровень доступности ресурсов, более эффективную систему управления, повышенную *безопасность* и усовершенствованную систему восстановления в критических ситуациях.

В широком смысле понятие виртуализации представляет собой сокрытие настоящей реализации какого-либо процесса или объекта от истинного его представления для того, кто им пользуется. Продуктом виртуализации является нечто удобное для использования, на самом деле, имеющее более сложную или совсем иную структуру, отличную от той, которая воспринимается при работе с объектом. Иными словами, происходит отделение представления от реализации чего-либо. *Виртуализация* призвана абстрагировать *программное обеспечение* от аппаратной части.

В компьютерных технологиях под термином "виртуализация" обычно понимается абстракция вычислительных ресурсов и предостав-

ление пользователю системы, которая "инкапсулирует"(скрывает в себе) собственную реализацию. Проще говоря, *пользователь* работает с удобным для себя представлением объекта, и для него не имеет значения, как *объект* устроен в действительности.

Сейчас возможность запуска нескольких виртуальных машин на одной физической вызывает большой интерес среди компьютерных специалистов, не только потому, что это повышает гибкость ИТ-инфраструктуры, но и потому, что *виртуализация*, на самом деле, позволяет экономить деньги.

История развития технологий виртуализации насчитывает более сорока лет. Компания IBM была первой, кто задумался о создании виртуальных сред для различных пользовательских задач, тогда еще в мэйнфреймах. В 60-х годах прошлого века *виртуализация* представляла чисто научный интерес и была оригинальным решением для изоляции компьютерных систем в рамках одного физического компьютера. После появления персональных компьютеров интерес к виртуализации несколько ослаб ввиду бурного развития операционных систем, которые предъявляли адекватные требования к аппаратному обеспечению того времени. Однако бурный рост аппаратных мощностей компьютеров в конце девяностых годов прошлого века заставил ИТ-сообщество вновь вспомнить о технологиях виртуализации программных платформ.

В 1999 г. компания VMware представила технологию виртуализации систем на базе *x86* в качестве эффективного средства, способного пре-

образовать системы на базе x86 в единую аппаратную инфраструктуру общего пользования и назначения, обеспечивающую полную изоляцию, мобильность и широкий выбор ОС для прикладных сред. Компания VMware была одной из первых, кто сделал серьезную ставку исключительно на виртуализацию. Как показало время, это оказалось абсолютно оправданным. Сегодня WMware предлагает комплексную виртуализационную платформу четвертого поколения VMware vSphere 4, которая включает средства как для отдельного ПК, так и для центра обработки данных. Ключевым компонентом этого программного комплекса является гипервизор VMware ESX*Server*. Позднее в "битву"за *место* в этом модном направлении развития информационных технологий включились такие компании как Parallels (ранее SWsoft), *Oracle* (*Sun* Microsystems), Citrix Systems (XenSourse).

Корпорация Microsoft вышла на рынок средств виртуализации в 2003 г. с приобретением компании Connectix, выпустив свой первый продукт *Virtual PC* для настольных ПК. С тех пор она последовательно наращивала спектр предложений в этой области и на сегодня почти завершила формирование виртуализационной платформы, в состав которой входят такие решения как *Windows* 2008 *Server* R2 с компонентом Hyper-V, Microsoft *Application Virtualization* (App-v), Microsoft *Virtual Desktop Infrastructure* (VDI), *Remote* Desktop Services,*System Center Virtual Machine Manager*.

На сегодняшний день поставщики технологий виртуализации предла-

гают надежные и легкоуправляемые платформы, а рынок этих технологий переживает настоящий бум. По оценкам ведущих экспертов, сейчас *виртуализация* входит в тройку наиболее перспективных компьютерных технологий. Многие эксперты предсказывали, что к 2015 году около половины всех компьютерных систем будут виртуальными.

Повышенный интерес к технологиям виртуализации в настоящее время неслучаен. Вычислительная мощь нынешних процессоров быстро растет, и вопрос даже не в том, на что эту мощь расходовать, а в том, что современная "мода"на двухъядерные и многоядерные системы, проникшая уже и в персональные компьютеры (ноутбуки и десктопы), как нельзя лучше позволяет реализовать богатейший *потенциал* идей виртуализации операционных систем и приложений, выводя удобство пользования компьютером на новый качественный уровень. Технологии виртуализации становятся одним из ключевых компонентов (в том числе, и маркетинговых) в самых новых и будущих процессорах Intel и *AMD*, в операционных системах от Microsoft и ряда других компаний.

3.1 Преимущества виртуализации

Приведем основные достоинства технологий виртуализации:

1. **Эффективное использование вычислительных ресурсов**. Вместо 3-х, а то и 10-и серверов, загруженных на 5-20 % можно использовать один, используемый на 50-70 %. Кроме прочего, это еще и

экономия электроэнергии, а также значительное сокращение финансовых вложений: приобретается один высокотехнологичный сервер, выполняющий функции 5-10 серверов. С помощью виртуализации можно достичь значительно более эффективного использования ресурсов, поскольку она обеспечивает объединение стандартных ресурсов инфраструктуры в единый пул и преодолевает ограничения устаревшей модели "одно приложение на сервер".

2. **Сокращение расходов на инфраструктуру**: Виртуализация позволяет сократить количество серверов и связанного с ними ИТ-оборудования в информационном центре. В результате этого потребности в обслуживании, электропитании и охлаждении материальных ресурсов сокращаются, и на ИТ затрачивается гораздо меньше средств.

3. **Снижение затрат на программное обеспечение**. Некоторые производители программного обеспечения ввели отдельные схемы лицензирования специально для виртуальных сред. Так, например, покупая одну лицензию на Microsoft Windows Server 2008 Enterprise, вы получаете право одновременно её использовать на 1 физическом сервере и 4 виртуальных (в пределах одного сервера), а Windows Server 2008 Datacenter лицензируется только на количество процессоров и может использоваться одновременно на неограниченном количестве виртуальных серверов.

4. **Повышение гибкости и скорости реагирования системы**: Виртуализация предлагает новый метод управления ИТ-инфра-

структурой и помогает ИТ-администраторам затрачивать меньше времени на выполнение повторяющихся заданий – например, на инициацию, настройку, отслеживание и техническое обслуживание. Многие системные администраторы испытывали неприятности, когда "рушится"сервер. И нельзя, вытащив жесткий диск, переставив его в другой сервер, запустить все как прежде... А установка? поиск драйверов, настройка, запуск... и на все нужны время и ресурсы. При использовании виртуального сервера – возможен моментальный запуск на любом "железе", а если нет подобного сервера, то можно скачать готовую виртуальную машину с установленным и настроенным сервером, из библиотек, поддерживаемых компаниями разработчиками гипервизоров (программ для виртуализации).

5. **Несовместимые приложения могут работать на одном компьютере**. При использовании виртуализации на одном сервере возможна установка linux и windows серверов, шлюзов, баз данных и прочих абсолютно несовместимых в рамках одной не виртуализированной системы приложений.

6. **Повышение доступности приложений и обеспечение непрерывности работы предприятия**. Благодаря надежной системе резервного копирования и миграции виртуальных сред целиком без перерывов в обслуживании вы сможете сократить периоды планового простоя и обеспечить быстрое восстановление системы в критических ситуациях. "Падение"одного виртуального сервера не ведет к потере осталь-

ных виртуальных серверов. Кроме того, в случае отказа одного физического сервера возможно произвести автоматическую замену на резервный сервер. Причем это происходит не заметно для пользователей без перезагрузки. Тем самым обеспечивается непрерывность бизнеса.

7. **Возможности легкой архивации**. Поскольку жесткий диск виртуальной машины обычно представляется в виде файла определенного формата, расположенный на каком-либо физическом носителе, виртуализация дает возможность простого копирования этого файла на резервный носитель как средство архивирования и резервного копирования всей виртуальной машины целиком. Возможность поднять из архива сервер полностью еще одна замечательная особенность. А можно поднять сервер из архива, не уничтожая текущий сервер и посмотреть положение дел за прошлый период.

8. **Повышение управляемости инфраструктуры**: использование централизованного управления виртуальной инфраструктурой позволяет сократить время на администрирование серверов, обеспечивает балансировку нагрузки и "живую"миграцию виртуальных машин.

3.2 Виртуальная машина

Виртуальной машиной *будем называть программную или аппаратную среду, которая скрывает настоящую реализацию какого-либо процесса или объекта от его видимого представления.*

Виртуальная машина *– это полностью изолированный программный контейнер, который работает с собственной ОС и приложениями, подобно физическому компьютеру. Виртуальная машина действует так же, как физический компьютер, и содержит собственные виртуальные (т.е. программные) ОЗУ, жесткий диск и сетевой адаптер.*

ОС не может различить виртуальную и физическую машины. То же самое можно сказать о приложениях и других компьютерах в сети. Даже сама *виртуальная машина* считает себя "настоящим"компьютером. Но несмотря на это виртуальные машины состоят исключительно из программных компонентов и не включают оборудование. Это дает им ряд уникальных преимуществ над физическим оборудованием.

Рисунок 3.2: Виртуальная машина

Рассмотрим основные особенности виртуальных машин более детально:

1. **Совместимость**. Виртуальные машины, как правило, совместимы со всеми стандартными компьютерами. Как и физический компьютер, виртуальная машина работает под управлением собственной гостевой операционной системы и выполняет собственные приложения. Она также содержит все компоненты, стандартные для физического компьютера (материнскую плату, видеокарту, сетевой контроллер и т.д.). Поэтому виртуальные машины полностью совместимы со всеми стандартными операционными системами, приложениями и драйверами устройств. Виртуальную машину можно использовать для выполнения любого программного обеспечения, пригодного для соответствующего физического компьютера.

2. **Изолированность**. Виртуальные машины полностью изолированы друг от друга, как если бы они были физическими компьютерами Виртуальные машины могут использовать общие физические ресурсы одного компьютера и при этом оставаться полностью изолированными друг от друга, как если бы они были отдельными физическими машинами. Например, если на одном физическом сервере запущено четыре виртуальных машины, и одна из них дает сбой, это не влияет на доступность оставшихся трех машин. Изолированность – важная причина гораздо более высокой доступности и безопасности приложений, выполняемых в виртуальной среде, по сравнению с приложениями, выполняемыми в стандартной, невиртуализированной системе.

3. **Инкапсуляция**. Виртуальные машины полностью инкапсулируют

вычислительную среду. Виртуальная машина представляет собой программный контейнер, связывающий, или "инкапсулирующий"полный комплект виртуальных аппаратных ресурсов, а также ОС и все её приложения в программном пакете. Благодаря инкапсуляции виртуальные машины становятся невероятно мобильными и удобными в управлении. Например, виртуальную машину можно переместить или скопировать из одного местоположения в другое так же, как любой другой программный файл. Кроме того, виртуальную машину можно сохранить на любом стандартном носителе данных: от компактной карты Flash-памяти USB до корпоративных сетей хранения данных.

4. **Независимость от оборудования**. Виртуальные машины полностью независимы от базового физического оборудования, на котором они работают. Например, для виртуальной машины с виртуальными компонентами (ЦП, сетевой картой, контроллером SCSI) можно задать настройки, абсолютно не совпадающие с физическими характеристиками базового аппаратного обеспечения. Виртуальные машины могут даже выполнять разные операционные системы (Windows, Linux и др.) на одном и том же физическом сервере. В сочетании со свойствами инкапсуляции и совместимости, аппаратная независимость обеспечивает возможность свободно перемещать виртуальные машины с одного компьютера на базе x86 на другой, не меняя драйверы устройств, ОС или приложения. Независимость от оборудования также дает возможность запускать в сочетании абсолютно разные ОС и приложения на одном

физическом компьютере.

Рассмотрим основные разновидности виртуализации, такие как:

- виртуализация серверов (полная виртуализация и паравиртуализация)
- виртуализация на уровне операционных систем
- виртуализация приложений
- виртуализация представлений

3.3 Виртуализация серверов

Сегодня, говоря о технологиях виртуализации, как правило, подразумевают виртуализацию серверов, так как последняя становится наиболее популярным решением на рынке IT. Виртуализация серверов подразумевает запуск на одном физическом сервере нескольких виртуальных серверов. Виртуальные машины или сервера представляют собой приложения, запущенные на хостовой операционной системе, которые эмулируют физические устройства сервера. На каждой виртуальной машине может быть установлена операционная система, на которую могут быть установлены приложения и службы. Типичные представители это продукты VmWare (ESX, Server, Workstation) и Microsoft (Hyper-V, Virtual Server, Virtual PC).

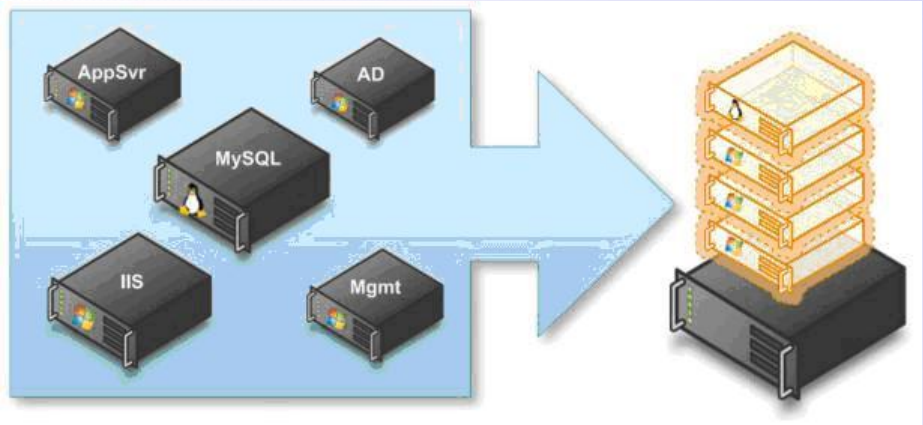


Рисунок 3.3: Виртуализация серверов

Центры обработки данных используют большое пространство и огромное количество энергии, особенно если прибавить к этому сопровождающие их системы охлаждения и инфраструктуру. Средствами технологий виртуализации выполняется консолидация серверов, расположенных на большом количестве физических серверов в виде виртуальных машин на одном высокопроизводительном сервере.

Число физических машин, необходимых для работы в качестве серверов уменьшается, что снижает количество энергии, необходимой для работы машин и пространство, требуемое для их размещения. Сокращение в количестве серверов и пространстве уменьшает количество энергии, необходимой для их охлаждения. При меньшем расходе энергии

вырабатывается меньшее количество углекислого газа. Данный показатель, например в Европе, имеет достаточно важную роль.

Немаловажным фактором является финансовая сторона. Виртуализация является важным моментом экономии. Виртуализация не только уменьшает потребность в приобретении дополнительных физических серверов, но и минимизирует требования к их размещению. Использование виртуального сервера предоставляет преимущества по быстроте внедрения, использования и управления, что позволяет уменьшить время ожидания развертывания какого-либо проекта.

Не так давно появились модели последнего поколения процессоров в архитектуре x86 корпораций AMD и Intel, где производители впервые добавили технологии аппаратной поддержки виртуализации. До этого виртуализация поддерживалась программно, что естественно приводила к большим накладным расходам производительности.

Для появившихся в восьмидесятых годах двадцатого века персональных компьютерах проблема виртуализации аппаратных ресурсов, казалось бы, не существовала по определению, поскольку каждый пользователь получал в свое распоряжение весь компьютер со своей ОС. Но по мере повышения мощности ПК и расширения сферы применения x86-систем ситуация быстро поменялась. "Диалектическая спираль"развития сделала свой очередной виток, и на рубеже веков начался очередной цикл усиления центростремительных сил по концентрации вычислительных ресурсов. В начале нынешнего десятилетия на фоне растущей за-

интересованности предприятий в повышении эффективности своих компьютерных средств стартовал новый этап развития технологий виртуализации, который сейчас преимущественно связывается именно с использованием архитектуры x86.

Отметим, что хотя в идеях x86-виртуализации в теоретическом плане вроде бы ничего неизвестного ранее не было, речь шла о качественно новом для ИТ-отрасли явлении по сравнению с ситуацией 20-летней давности. Дело в том, что в аппаратно-программной архитектуре мэйнфреймов и Unix-компьютеров вопросы виртуализации сразу решались на базовом уровне и аппаратном уровне. Система же x86 строилась совсем не в расчете на работу в режиме датацентров, и ее развитие в направлении виртуализации – это довольно сложный эволюционный процесс со множеством разных вариантов решения задачи.

Важный момент заключается также в качественно разных бизнес-моделях развития мэйнфреймов и x86. В первом случае речь идет фактически о моновендорном программно-аппаратном комплексе для поддержки довольно ограниченного круга прикладного ПО для достаточно узкого круга крупных заказчиков. Во втором - мы имеем дело с децентрализованным сообществом производителей техники, поставщиков базового ПО и огромной армией разработчиков прикладного программного обеспечения.

Использование средств x86-виртуализации началось в конце 90-х с рабочих станций: одновременно с увеличением числа версий клиентских

ОС постоянно росло и количество людей (разработчиков ПО, специалистов по технической поддержке, экспертов), которым нужно было на одном ПК иметь сразу несколько копий различных ОС.

Виртуализация для серверной инфраструктуры стала применяться немного позднее, и связано это было, прежде всего, с решением задач консолидации вычислительных ресурсов. Но тут сразу сформировалось два независимых направления:

- поддержка неоднородных операционных сред (в том числе, для работы унаследованных приложений). Этот случай наиболее часто встречается в рамках корпоративных информационных систем. Технически проблема решается путем одновременной работы на одном компьютере нескольких виртуальных машин, каждая из которых включает экземпляр операционной системы. Но реализация этого режима выполнялась с помощью двух принципиально разных подходов: **полной виртуализации и паравиртуализации** ;

- поддержка однородных вычислительных сред подразумевает изоляцию служб в рамках одного экземпляра ядра операционной системы (**виртуализация на уровне ОС**), что наиболее характерно для хостинга приложений провайдерами услуг. Конечно, тут можно использовать и вариант виртуальных машин, но гораздо эффективнее создание изолированных контейнеров на базе одного ядра одной ОС.

Следующий жизненный этап технологий x86-виртуализации стартовал в 2004-2006 гг. и был связан с началом их массового применения в корпоративных системах. Соответственно, если раньше разработчики в основном занимались созданием технологий исполнения виртуальных сред, то теперь на первый план стали выходить задачи управления этими решениями и их интеграции в общую корпоративную ИТ-инфраструктуру. Одновременно обозначилось заметное повышение спроса на виртуализацию со стороны персональных пользователей (но если в 90-х это были разработчики и тестеры, то сейчас речь уже идет о конечных пользователях как профессиональных, так и домашних).

Многие трудности и проблемы разработки технологий виртуализации связаны с преодолением унаследованных особенностей программно-аппаратной архитектуры x86. Для этого существует несколько базовых методов, речь о которых пойдет ниже.

3.4 Полная виртуализация

Полная виртуализация (Full, Native Virtualization). Используются не модифицированные экземпляры гостевых операционных систем, а для поддержки работы этих ОС служит общий слой эмуляции их исполнения поверх хостовой ОС, в роли которой выступает обычная операционная система. Такая технология применяется, в частности, в VMware Workstation, VMware Server (бывший GSX Server), Parallels Desktop,

Parallels Server, MS Virtual PC, MS Virtual *Server*, *Virtual* Iron. К достоинствам данного подхода можно причислить относительную простоту реализации, универсальность и надежность решения; все функции управления берет на себя хост-ОС. Недостатки – высокие дополнительные накладные расходы на используемые аппаратные ресурсы, отсутствие учета особенностей гостевых ОС, меньшая, чем нужно, гибкость в использовании аппаратных средств.

Рисунок 3.4: Полная виртуализация

3.5 Паравиртуализация

Паравиртуализация (paravirtualization). Модификация ядра гостевой ОС выполняется таким образом, что в нее включается новый набор API, через который она может напрямую работать с аппаратурой, не конфликтуя с другими виртуальными машинами. При этом нет необходимости задействовать полноценную ОС в качестве хостового ПО, функ-

ции которого в данном случае исполняет специальная система, получившая название гипервизора (*hypervisor*). Именно этот вариант является сегодня наиболее актуальным направлением развития серверных технологий виртуализации и применяется в VMware ESX Server, Xen (и решениях других поставщиков на базе этой технологии), Microsoft Hyper-V. Достоинства данной технологии заключаются в отсутствии потребности в хостовой ОС – ВМ, устанавливаются фактически на "голое железо а аппаратные ресурсы используются эффективно. Недостатки – в сложности реализации подхода и необходимости создания специализированной ОС-гипервизора.

Рисунок 3.5: Паравиртуализация

3.6 Виртуализация на уровне ядра ОС

Виртуализация на уровне ядра ОС (operating *system-level* virtualization). Этот вариант подразумевает использование одного ядра хостовой ОС для создания независимых параллельно работающих

операционных сред. Для гостевого ПО создается только собственное сетевое и аппаратное окружение. Такой вариант используется в Virtuozzo (для Linux и Windows), OpenVZ (бесплатный вариант Virtuozzo) и Solaris Containers. Достоинства – высокая эффективность использования аппаратных ресурсов, низкие накладные технические расходы, отличная управляемость, минимизация расходов на приобретение лицензий. Недостатки – реализация только однородных вычислительных сред.

Рисунок 3.6: Виртуализация на уровне ОС

3.7 Виртуализация приложений

Виртуализация приложений подразумевает применение модели сильной изоляции прикладных программ с управляемым взаимодействием с ОС, при которой виртуализируется каждый экземпляр приложений, все его основные компоненты: файлы (включая системные), реестр, шрифты, INI-файлы, COM-объекты, службы. Приложение исполняется

без процедуры инсталляции в традиционном ее понимании и может запускаться прямо с внешних носителей (например, с флэш-карт или из сетевых папок). С точки зрения ИТ-отдела, такой подход имеет очевидные преимущества: ускорение развертывания настольных систем и возможность управления ими, сведение к минимуму не только конфликтов между приложениями, но и потребности в тестировании приложений на совместимость. Данная технология позволяет использовать на одном компьютере, а точнее в одной и той же операционной системе несколько несовместимых между собой приложений одновременно. Виртуализация приложений позволяет пользователям запускать одно и то же заранее сконфигурированное приложение или группу приложений с сервера. При этом приложения будут работать независимо друг от друга, не внося никаких изменений в операционную систему. Фактически именно такой вариант виртуализации используется в Sun *Java Virtual Machine*, Microsoft *Application Virtualization* (ранее называлось Softgrid), Thinstall (в начале 2008 г. вошла в состав VMware), Symantec/Altiris.

Рисунок 3.7: Виртуализация приложений

3.8 Виртуализация представлений (рабочих мест)

Виртуализация представлений (рабочих мест) Виртуализация представлений подразумевает эмуляцию интерфейса пользователя. Т.е. пользователь видит приложение и работает с ним на своём терминале, хотя на самом деле приложение выполняется на удалённом сервере, а пользователю передаётся лишь картинка удалённого приложения. В зависимости от режима работы пользователь может видеть удалённый рабочий стол и запущенное на нём приложение, либо только само окно приложения.

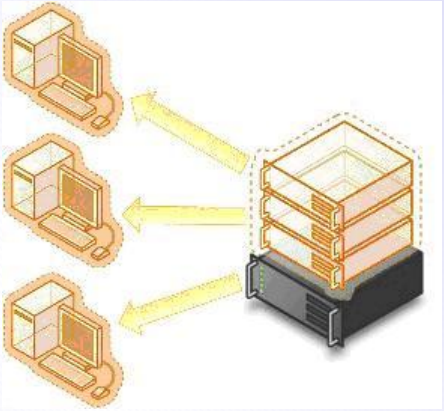


Рисунок 3.8: Виртуализация представлений

Потребности бизнеса меняют наши представления об организации рабочего процесса. Персональный компьютер, ставший за последние десятилетия неотъемлемым атрибутом офиса и средством выполнения большинства офисных задач, перестает успевать за растущими потребностями бизнеса. Реальным инструментом пользователя оказывается программное обеспечение, которое лишь привязано к ПК, делая его промежуточным звеном корпоративной информационной системы. В результате активное развитие получают "облачные"вычисления, когда пользователи имеют доступ к собственным данным, но не управляют и не задумываются об инфраструктуре, операционной системе и собственно программном обеспечении, с которым они работают.

Вместе с тем, с ростом масштабов организаций, использование в ИТ-

инфраструктуре пользовательских ПК вызывает ряд сложностей:

- большие операционные издержки на поддержку компьютерного парка;
- сложность, связанная с управлением настольными ПК;
- обеспечение пользователям безопасного и надежного доступа к ПО и приложениям, необходимым для работы;
- техническое сопровождение пользователей;
- установка и обновление лицензий на ПО и техническое обслуживание;
- резервное копирование и т.д.

Уйти от этих сложностей и сократить издержки, связанные с их решением, возможно благодаря применению технологии виртуализации рабочих мест сотрудников на базе инфраструктуры виртуальных ПК – *Virtual Desktop* Infrastructure (*VDI*). *VDI* позволяет отделить пользовательское ПО от аппаратной части – персонального компьютера, - и осуществлять доступ к клиентским приложениям через терминальные устройства.

VDI – комбинация соединений с удаленным рабочим столом и виртуализации. На обслуживающих серверах работает множество виртуальных машин, с такими клиентскими операционными системами, как Windows

7, Windows Vista и Windows XP или Linux операционными системами. Пользователи дистанционно подключаются к виртуальной машине своей настольной среды. На локальных компьютерах пользователей в качестве удаленного настольного клиента могут применяться терминальные клиенты, старое оборудование с Microsoft Windows Fundamentals или дистрибутив Linux.

VDI полностью изолирует виртуальную среду пользователей от других виртуальных сред, так как каждый пользователь подключается к отдельной виртуальной машине. Иногда используется статическая инфраструктура *VDI*, в которой пользователь всегда подключается к той же виртуальной машине, в других случаях динамическая *VDI*, в которой пользователи динамически подключаются к различным виртуальным машинам, и виртуальные машины создаются по мере необходимости. При использовании любой модели важно хранить данные пользователей вне виртуальных машин и быстро предоставлять приложения.

Наряду с централизованным управлением и простым предоставлением компьютеров, *VDI* обеспечивает доступ к настольной среде из любого места, если пользователи могут дистанционно подключиться к серверу.

Представим, что на клиентском компьютере возникла неполадка. Придется выполнить диагностику и, возможно, переустановить операционную систему. Благодаря *VDI* в случае неполадок можно просто удалить виртуальную машину и за несколько секунд создать новую среду, с помощью созданного заранее шаблона виртуальной машины. *VDI* обеспе-

чивает дополнительную безопасность, так как данные не хранятся локально на настольном компьютере или ноутбуке.

Как пример виртуализации представлений можно рассматривать и технологию тонких терминалов, которые фактически виртуализируют рабочие места пользователей настольных систем: пользователь не привязан к какому-то конкретному ПК, а может получить доступ к своим файлам и приложениям, которые располагаются на сервере, с любого *удаленного терминала* после выполнения процедуры авторизации. Все команды пользователя и изображение сеанса на мониторе эмулируются с помощью ПО управления тонкими клиентами. Применение этой технологии позволяет централизовать обслуживание клиентских рабочих мест и резко сократить расходы на их поддержку – например, для перехода на следующую версию клиентского приложения новое ПО нужно инсталлировать только один раз на сервере.

Рисунок 3.9: Пример тонкого клиента. Терминал Sun Ray.

Одним из наиболее известных тонких клиентов является терминал Sun Ray, для организации работы которого используется программное обеспечение Sun Ray *Server Software*. Для начала сеанса Sun Ray достаточно лишь вставить в это устройство идентификационную смарт-карту. Применение смарт-карты существенно повышает мобильность пользователя – он может переходить с одного Sun Ray на другой, переставляя между ними свою карточку и сразу продолжать работу со своими приложениями с того места, где он остановился на предыдущем терминале. А отказ от жесткого диска не только обеспечивает мобильность пользователей и повышает безопасность данных, но и существенно снижает энергопотребление по сравнению с обычными ПК, поэтому терминал Sun не имеет вентилятора и работает практически бесшумно. Кроме того, сокращение числа компонентов тонкого терминала уменьшает и риск выхода его из строя, а следовательно, экономит расходы на его обслуживание. Еще одно преимущество Sun Ray – это существенно расширенный по сравнению с обычными ПК *жизненный цикл продукта*, поскольку в нём нет компонентов, которые могут морально устареть.

3.9 Краткий обзор платформ виртуализации

VMware

Компания VMware – один из первых игроков на рынке платформ виртуализации. В 1998 году VMware запатентовала свои программные

техники виртуализации и с тех пор выпустила немало эффективных и профессиональных продуктов для виртуализации различного уровня: от VMware Workstation, предназначенного для настольных ПК, до VMware ESX Server, позволяющего консолидировать физические серверы предприятия в виртуальной инфраструктуре.

В отличие от ЭВМ (мэйнфрейм) устройства на базе x86 не поддерживают виртуализацию в полной мере. Поэтому компании VMware пришлось преодолеть немало проблем в процессе создания виртуальных машин для компьютеров на базе x86. Основные функции большинства ЦП (в ЭВМ и ПК) заключаются в выполнении последовательности сохраненных инструкций (т.е. программ). В процессорах на базе x86 содержатся 17 особых инструкций, создающих проблемы при виртуализации, из-за которых операционная система отображает предупреждающее сообщение, прерывает работу приложения или просто выдает общий сбой. Итак, эти 17 инструкций оказались значительным препятствием на начальном этапе внедрения виртуализации для компьютеров на базе x86.

Для преодоления этого препятствия компания VMware разработала адаптивную технологию виртуализации, которая "перехватывает"данные инструкции на этапе создания и преобразует их в безопасные инструкции, пригодные для виртуализации, не затрагивая при этом процессы выполнения всех остальных инструкций. В результате мы получаем высокопроизводительную виртуальную машину, соответствующую аппаратному обеспечению узла и поддерживающую полную программную

совместимость. Компания VMware первой разработала и внедрила данную инновационную технологию, поэтому на сегодняшний день она является неоспоримым лидером технологий виртуализации.

В весьма обширном списке продуктов VMware можно найти немало инструментов для повышения эффективности и оптимизации ИТ-инфраструктуры, управления виртуальными серверами, а также средства миграции с физических платформ на виртуальные. По результатам различных тестов производительности средства виртуализации VMware почти всегда по большинству параметров выигрывают у конкурентов. VMware имеет более 100 000 клиентов по всему миру, в списке ее клиентов 100 % организаций из Fortune 100. Сеть партнерств охватывает более 350 производителей оборудования и ПО и более 6000 реселлеров. На данный момент объем рынка, принадлежащий VMware, оценивается на 80 %. Между тем, среди платформ виртуализации у VMware есть из чего выбирать:

VMware Workstation – платформа, ориентированная на desktop-пользователей и предназначенная для использования разработчиками ПО, а также профессионалами в сфере ИТ. Новая версия популярного продукта VMware Workstation 7 стала доступна в 2009 г, в качестве хостовых операционных систем поддерживаются Windows и Linux. VMware Workstation 7 может использоваться совместно со средой разработки, что делает ее особенно популярной в среде разработчиков, преподавателей и специалистов технической поддержки. Выход VMware

Workstation 7 означает официальную поддержку Windows 7 как в качестве гостевой, так и хостовой операционной системы. Продукт включает поддержку Aero *Peek* и Flip 3D, что делает возможным наблюдать за работой виртуальной машины, подводя курсор к панели задач VMware или к соответствующей вкладке на рабочем столе хоста. Новая версия может работать на любой версии Windows 7, также как и любые версии Windows могут быть запущены в виртуальных машинах. Кроме того, виртуальные машины в VMware Workstation 7, полностью поддерживают Windows Display Driver Model (WDDM), что позволяет использовать интерфейс Windows Aero в гостевых машинах.

VMware Player – бесплатный "проигрыватель"виртуальных машин на основе виртуальной машины VMware Workstation, предназначенный для запуска уже готовых образов виртуальных машин, созданных в других продуктах VMware, а также в Microsoft VirtualPC и Symantec LiveState Recovery. Начиная с версии 3.0 VMware Player позволяет также создавать образы виртуальных машин. Ограничение функциональности теперь касается в основном функций, предназначенных для IT-специалистов и разработчиков ПО.

VMware Fusion – настольный продукт для виртуализации на платформе Mac от компании Apple.

VMware Server, Бесплатный продукт VMware Server является довольно мощной платформой виртуализации, которая может быть запущена на серверах под управлением хостовых операционных систем

Windows и Linux. Основное предназначение VMware Server – поддержка малых и средних виртуальных инфраструктур небольших предприятий. В связи с небольшой сложностью его освоения и установки, VMware Server может быть развернут в кратчайшие сроки, как на серверах организаций, так и на компьютерах домашних пользователей.

VMware Ace – продукт для создания защищенных политиками безопасности виртуальных машин, которые затем можно распространять по модели SaaS (Software-as-a-Service).

VMware vSphere – комплекс продуктов, представляющий надежную платформу для виртуализации ЦОД. Компания позиционирует данный комплекс также как мощную платформу виртуализации для создания и развертывания частного "облака". VMware vSphere поставляется в нескольких выпусках с возможностями, предназначенными специально для малых компаний и средних компаний и корпораций. VMware vSphere включает ряд компонентов, преобразующих стандартное оборудование в общую устойчивую среду, напоминающую мейнфрейм и включающую встроенные элементы управления уровнями обслуживания для всех приложений:

- Службы инфраструктуры – это компоненты, обеспечивающие всестороннюю виртуализацию ресурсов серверов, хранилищ и сетей, их объединение и точное выделение приложениям по требованию и в соответствии с приоритетами бизнеса.

- Службы приложений – это компоненты, предоставляющие встроенные элементы управления уровнями обслуживания для всех приложений на платформе платформы vSphere независимо от их типа или ОС.
- VMware vCenter Server предоставляет центральную консоль для управления виртуализацией, обеспечивающую администрирования служб инфраструктуры и приложений. Эта консоль поддерживает всестороннюю визуализацию всех аспектов виртуальной инфраструктуры, автоматизацию повседневной эксплуатации и масштабируемость для управления крупными средами ЦОД.

Рисунок 3.10: Структура платформы vShpere.

VMware ESX Server – это гипервизор, разбивающий физические серверы на множество виртуальных машин. VMware ESX является основой пакета VMware vSphere и входит во все выпуски VMware vSphere.

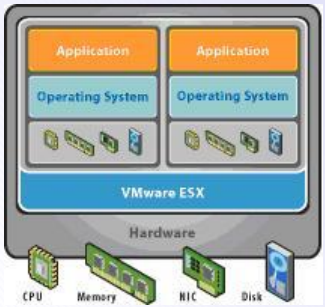


Рисунок 3.11: Гипервизор VMware ESX.

VMware vSphere Hypervisor (ранее VMware ESXi) – "облегчённая"платформа виртуализации корпоративного уровня, основанная на технологиях ESX. Продукт является бесплатным и доступен для загрузки с сайта VMware. VSphere VMware *Hypervisor* является простейшим способом для начала работы с виртуализацией.

VMware vCenter – предоставляет расширяемую и масштабируемую платформу для упреждающего управления виртуальной инфраструктурой и обеспечивает получение о ней всеобъемлющей информации. VMware vCenter Server обеспечивает централизованное управление средами vSphere и упрощает выполнение повседневных задач, значительно улучшая административное управление средой. Продукт обладает широкими возможностями по консолидации серверов, их настройке и управлению. VMware vCenter Server агрегирует в себе все аспекты управления виртуальной средой: от виртуальных машин до сбора ин-

формации о физических серверах для последующей их миграции в виртуальную инфраструктуру. Кроме центрального продукта управления виртуальной инфраструктурой vCenter Server существует также ряд дополнений, реализующих различные аспекты планирования, управления и интеграции распределенной виртуальной инфраструктуры (VMware vCenter Server *Heartbeat*, VMware vCenter Orchestrator, VMware vCenter Capacity IQ, VMware vCenter Site Recovery Manager, VMware vCenter Lab Manager, VMware vCenter Configuration Manager, VMware vCenter Converter). В частности, **vCenter Converter** предназначен для перевода в виртуальную среду физических серверов, позволяющий осуществлять "горячую"(без останова систем) и "холодную"миграцию. **vCenter Site Recovery Manager** – это ПО для создания территориально-удаленного резервного сегмента виртуальной инфраструктуры, который в случае отказа основного узла, берет на себя функции по запуску виртуальных машин в соответствии с планом восстановления после сбоев. **vCenter Lab Manager** – продукт для создания инфраструктуры хранения и доставки конфигураций виртуальных машин, позволяющий организовать эффективную схему тестирования в компаниях-разработчиках ПО.

VMware ThinApp – бывший продукт Thinstall Virtualization Suite, ПО для виртуализации приложений, позволяющее распространять предустановленные приложения на клиентские рабочие станции, сокращая время на стандартные операции по установке и конфигурации.

VMware View – комплекс продуктов, обеспечивающий централизацию пользовательских рабочих станций в виртуальных машинах на платформе vSphere. Это позволяет сократить затраты на стандартные ИТ-операции, связанные с развертыванием и обслуживанием пользовательских десктопов.

VMware Capacity Planner – средство централизованного сбора и анализа данных об аппаратном и программном обеспечении серверов, а также производительности оборудования. Эти данные используются авторизованными партнерами VMware для построения планов консолидации виртуальных машин на платформе VMware ESX Server.

VMware VMmark – продукт, доступный только производителям аппаратного обеспечения, предназначенный для тестирования производительности VMware ESX Server на серверных платформах.

Citrix (Xen)

Разработка некоммерческого гипервизора Xen начиналась как исследовательский проект компьютерной лаборатории Кембриджского университета. Основателем проекта и его лидером был Иан Пратт (Ian Pratt) сотрудник университета, который создал впоследствии компанию XenSource, занимающуюся разработкой коммерческих платформ виртуализации на основе гипервизора Xen, а также поддержкой Open Source сообщества некоммерческого продукта Xen. Изначально Xen представлял собой самую развитую платформу, поддерживающую технологию паравиртуализации. Эта технология позволяет гипервизору в хостовой

системе управлять гостевой ОС посредством гипервызовов VMI (Virtual Machine Interface), что требует модификации ядра гостевой системы. На данный момент бесплатная версия Xen включена в дистрибутивы нескольких ОС, таких как Red Hat, Novell SUSE, Debian, Fedora Core, Sun Solaris. В середине августа 2007 года компания XenSource была поглощена компанией Citrix Systems. Сумма проведенной сделки около 500 миллионов долларов (акциями и денежными средствами) говорит о серьезных намерениях Citrix в отношении виртуализации. Эксперты полагают, что не исключена и покупка Citrix компанией Microsoft, учитывая давнее ее сотрудничество с XenSource.

Бесплатный Xen. В настоящее время Open Source версия платформы Xen применяется в основном в образовательных и исследовательских целях. Некоторые удачные идеи, реализованные многочисленными разработчиками со всего мира, находят свое отражение в коммерческих версиях продуктов виртуализации компании Citrix. Сейчас бесплатные версии Xen включаются в дистрибутивы многих Linux-систем, что позволяет их пользователям применять виртуальные машины для изоляции программного обеспечения в гостевых ОС с целью его тестирования и изучения проблем безопасности, без необходимости установки платформы виртуализации. К тому же многие независимые разработчики ПО могут распространять его с помощью виртуальных шаблонов, в которых уже установлена и настроена гостевая система и предлагаемый продукт. Кроме того, Xen идеально подходит для поддержки старого программ-

ного обеспечения в виртуальной машине. Для более же серьезных целей в производственной среде предприятия необходимо использовать коммерческие платформы компании Citrix.

Citrix XenApp – предназначен для виртуализации и публикации приложений в целях оптимизации инфраструктуры доставки сервисов в крупных компаниях. XenApp имеет огромное количество пользователей по всему миру и во многих компаниях является ключевым компонентом ИТ-инфраструктуры.

Citrix XenServer – платформа для консолидации серверов предприятий среднего масштаба, включающая основные возможности для поддержания виртуальной инфраструктуры. Производитель позиционирует данный продукт как решение Enterprise-уровня для виртуализации серверов, поддерживающее работу в "облачном"окружении.

Citrix XenDesktop – решение по виртуализации десктопов предприятия, позволяющее централизованно хранить и доставлять рабочие окружения в виртуальных машинах пользователям. Продукт поддерживает несколько сценариев доставки приложений на настольные ПК, тонкие клиенты и мобильные ПК и совместим с серверными виртуализационными решениями конкурентов.

Microsoft

Для Microsoft все началось, когда в 2003 году она приобрела компанию Connectix, одну из немногих компаний производящую программное обеспечение для виртуализации под Windows. Вместе с Connectix,

компании Microsoft достался продукт Virtual PC, конкурировавший тогда с разработками компании VMware в отношении настольных систем виртуализации. По большому счету, Virtual PC предоставлял тогда такое количество функций, что и VMware Workstation, и при должном внимании мог бы быть в настоящее время полноценным конкурентом этой платформы. Однако с того времени, компания Microsoft выпускала по минорному релизу в год, не уделяя особого внимания продукту Virtual PC, в то время как VMware стремительно развивала свою систему виртуализации, превратив ее по-настоящему в профессиональный инструмент. Осознав свое технологическое отставание в сфере виртуализации серверных платформ, компания Microsoft выпустила продукт Virtual Server 2005, нацеленный на создание и консолидацию виртуальных серверов организаций. Однако было уже поздно. Компания VMware уже захватила лидерство в этом сегменте рынка, предлагая в тот момент две серверные платформы виртуализации VMware GSX Server и VMware ESX Server, каждая из которых по многим параметрам превосходила платформу Microsoft. Окончательный удар был нанесен в 2006 году, когда VMware фактически объявила продукт VMware GSX Server бесплатным, взявшись за разработку продукта VMware Server на его основе и сконцентрировав все усилия на продажах мощной корпоративной платформы VMware ESX Server в составе виртуальной инфраструктуры *Virtual Infrastructure 3*. У компании Microsoft был только единственный выход в этой ситуации: в апреле 2006 года она также объявила о бесплат-

ности продукта Microsoft Virtual Server 2005. Также существовавшие ранее два издания Standard Edition и Enterprise Edition были объединены в одно – Microsoft Virtual Server Enterprise Edition. С тех пор Microsoft существенно изменила стратегию в отношении виртуализации, и летом 2008 года был выпущен финальный релиз платформы виртуализации Microsoft Hyper-V, интегрированной в ОС Windows Server 2008. Теперь роль сервера виртуализации доступна всем пользователям новой серверной операционной системы Microsoft.

Microsoft Virtual Server. Серверная платформа виртуализации Microsoft Virtual Server может использоваться на сервере под управлением операционной системы Windows Server 2003 и предназначена для одновременного запуска нескольких виртуальных машин на одном физическом хосте. Платформа бесплатна и предоставляет только базовые функции.

Microsoft Virtual PC. Продукт Virtual PC был куплен корпорацией Microsoft вместе с компанией Connectix и впервые под маркой Microsoft был выпущен как Microsoft Virtual PC 2004. Приобретая Virtual PC и компанию Connectix, компания Microsoft строила далеко идущие планы по обеспечению пользователей инструментом для облегчения миграции на следующую версию операционной системы Windows. Теперь Virtual PC 2007 бесплатен и доступен для поддержки настольных ОС в виртуальных машинах.

Microsoft Hyper-V. Продукт Microsoft позиционируется как основ-

ной конкурент VMware ESX Server в области корпоративных платформ виртуализации. Microsoft Hyper-V представляет собой решение для виртуализации серверов на базе процессоров с архитектурой x64 в корпоративных средах. В отличие от продуктов Microsoft Virtual Server или Virtual PC, Hyper-V обеспечивает виртуализацию на аппаратном уровне, с использованием технологий виртуализации, встроенных в процессоры. Hyper-V обеспечивает высокую производительность, практически равную производительности одной операционной системы, работающей на выделенном сервере. Hyper-V распространяется двумя способами: как часть Windows Server 2008 или в составе независимого бесплатного продукта Microsoft Hyper-V Server.

В Windows Server 2008 технология Hyper-V может быть развернута как в полной установке, так и в режиме Server Core, Hyper-V Server работает только в режиме Core. Это позволяет в полной мере реализовать все преимущества "тонкой экономичной и управляемой платформы виртуализации.

Hyper-V является встроенным компонентом 64-разрядных версий Windows Server 2008 Standard, Windows Server 2008 Enterprise и Windows Server 2008 Datacenter. Эта технология недоступна в 32-разрядных версиях Windows Server 2008, в Windows Server 2008 Standard без Hyper-V, Windows Server 2008 Enterprise без Hyper-V, Windows Server 2008 Datacenter без Hyper-V, в Windows Web Server 2008 и Windows Server 2008 для систем на базе Itanium.

Рассмотрим кратко особенности архитектуры Hyper-V. Hyper-V представляет собой гипервизор, т.е. прослойку между оборудованием и виртуальными машинами уровнем ниже операционной системы. Эта архитектура была первоначально разработана IBM в 1960-е годы для мэйнфреймов и недавно стала доступной на платформах x86/x64, как часть ряда решений, включая Windows Server 2008 Hyper-V и Vmware ESX.

Рисунок 3.12: Архитектура виртуализации с гипервизором

Виртуализация на базе гипервизора основана на том, что между оборудованием и виртуальными машинами появляется прослойка, перехватывающая обращения операционных систем к процессору, памяти и дру-

гим устройствам. При этом доступ к периферийным устройствам в разных реализациях гипервизоров может быть организован по-разному. С точки зрения существующих решений для реализации менеджера виртуальных машин можно выделить два основных вида архитектуры гипервизора: **микроядерную и монолитную**.

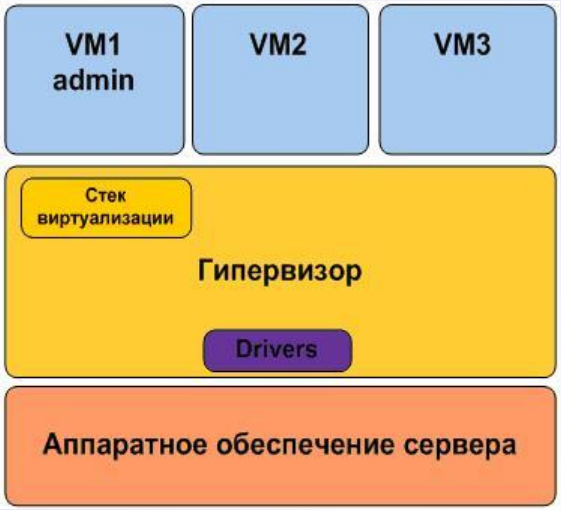


Рисунок 3.13: Архитектура монолитного гипервизора

Монолитный подход размещает гипервизор в едином уровне, который также включает большинство требуемых компонентов, таких как ядро, драйверы устройств и стек ввода/вывода. Это подход, использу-

емый такими решениями, как VMware ESX и традиционные системы мэйнфреймов.

Монолитный подход подразумевает, что все драйвера устройств помещены в гипервизор. В монолитной модели – гипервизор для доступа к оборудованию использует собственные драйверы. Гостевые ОС работают на виртуальных машинах поверх гипервизора. Когда гостевой системе нужен доступ к оборудованию, она должна пройти через гипервизор и его модель драйверов. Обычно одна из гостевых ОС играет роль администратора или консоли, в которой запускаются компоненты для предоставления ресурсов, управления и мониторинга всех гостевых ОС, работающих на сервере.

Модель монолитного гипервизора обеспечивает прекрасную производительность, но имеет ряд недостатков, таких как:

- Устойчивость – если в обновленную версию драйвера затесалась ошибка, в результате сбои начнутся во всей системе, во всех ее виртуальных машинах.
- Проблемы обновления драйверов – при необходимости обновления драйвера какого-либо устройства (например сетевого адаптера) обновить драйвер возможно только вместе с выходом новой версии гипервизора, в которую будет интегрирован новый драйвер для данного устройства.
- Трудности с использованием неподдерживаемого оборудования. На-

пример, вы собрались использовать оборудование "Сервер" достаточно мощный и надежный, но при этом в гипервизоре не оказалось нужного драйвера для RAID-контроллера или сетевого адаптера. Это сделает невозможным использование соответствующего оборудования, а, значит, и сервера.

Микроядерный подход использует очень тонкий, специализированный гипервизор, выполняющий лишь основные задачи обеспечения изоляции разделов и управления памятью. Этот уровень не включает стека ввода/вывода или драйверов устройств. Это подход, используемый Hyper-V. В этой архитектуре стек виртуализации и драйверы конкретных устройств расположены в специальном разделе ОС, именуемом родительским разделом.

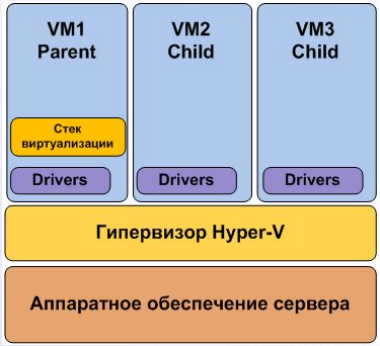


Рисунок 3.14: Архитектура микроядерного гипервизора

В микроядерной реализации можно говорить о "тонком гипервизоре в этом случае в нем совсем нет драйверов. Вместо этого драйверы работают в каждом индивидуальном разделе, чтобы любая гостевая ОС имела возможность получить через гипервизор доступ к оборудованию. При такой расстановке сил каждая виртуальная машина занимает совершенно обособленный раздел, что положительно сказывается на защищенности и надежности. В микроядерной модели гипервизора (в виртуализации Windows Server 2008 R2 используется именно она) один раздел является родительским (parent), остальные – дочерними (child). Раздел – это наименьшая изолированная единица, поддерживаемая гипервизором. Размер гипервизора Hyper-V менее 1,5 Мб , он может поместиться на одну 3.5-дюймовую дискету.

Каждому разделу назначаются конкретные аппаратные ресурсы – долю процессорного времени, объем памяти, устройства и пр. Родительский раздел создает дочерние разделы и управляет ими, а также содержит стек виртуализации (virtualization stack), используемый для управления дочерними разделами. Родительский раздел создается первым и владеет всеми ресурсами, не принадлежащими гипервизору. Обладание всеми аппаратными ресурсами означает, что именно корневой (то есть, родительский) раздел управляет питанием, подключением самонастраивающихся устройств, ведает вопросами аппаратных сбоев и даже управляет загрузкой гипервизора.

В родительском разделе содержится стек виртуализации – набор про-

граммных компонентов, расположенных поверх гипервизора и совместно с ним обеспечивающих работу виртуальных машин. Стек виртуализации обменивается данными с гипервизором и выполняет все функции по виртуализации, не поддерживаемые непосредственно гипервизором. Большая часть этих функций связана с созданием дочерних разделов и управлением ими и необходимыми им ресурсами (ЦП, память, устройства).

Преимущество микроядерного подхода, примененного в Windows Server 2008 R2, по сравнению с монолитным подходом состоит в том, что драйверы, которые должны располагаться между родительским разделом и физическим сервером, не требуют внесения никаких изменений в модель драйверов. Иными словами, в системе можно просто применять существующие драйверы. В Microsoft этот подход избрали, поскольку необходимость разработки новых драйверов сильно затормозила бы развитие системы. Что же касается гостевых ОС, они будут работать с эмуляторами или синтетическими устройствами.

С другой стороны, микроядерная модель может несколько проигрывать монолитной модели в производительности. Однако в наши дни главным приоритетом стала безопасность, поэтому для большинства компаний вполне приемлема будет потеря пары процентов в производительности ради сокращения фронта нападения и повышения устойчивости.

Рисунок 3.15: Архитектура Hyper-V

Все версии Hyper-V имеют один родительский раздел. Этот раздел управляет функциями Hyper-V. Из родительского раздела запускается консоль Windows *Server Virtualization*. Кроме того, родительский раздел используется для запуска виртуальных машин (VM), поддерживающих потоковую эмуляцию старых аппаратных средств. Такие VM, построенные на готовых шаблонах, эмулирующих аппаратные средства, являются аналогами VM, работающих в продуктах с виртуализацией на базе хоста, например Virtual Server.

Гостевые VM запускаются из дочерних разделов Hyper-V. Дочерние разделы поддерживают два типа VM: высокопроизводительные VM на

основе архитектуры VMBus и VM, управляемые системой-хостом. В первую группу входят VM с системами Windows Server 2003, Windows Vista, Server 2008 и Linux (поддерживающими Xen). Новую архитектуру VMBus отличает высокопроизводительный конвейер, функционирующий в оперативной памяти, соединяющий клиентов Virtualization Service Clients (VSC) на гостевых VM с провайдером Virtual Service Provider (*VSP*) хоста. VM, управляемые хостом, запускают платформы, не поддерживающие новую архитектуру VMBus: Windows NT, Windows 2000 и Linux (без поддержки технологии Xen, например SUSE Linux Server Enterprise 10).

Microsoft System Center Virtual Machine Manager (SCVMM) – отдельный продукт семейства System Center для управления виртуальной инфраструктурой, эффективного использованием ресурсов физических узлов, а также упрощение подготовки и создания новых гостевых систем для администраторов и пользователей. Продукт обеспечивает всестороннюю поддержку консолидации физических серверов в виртуальной инфраструктуре, быстрое и надежное преобразование физических машин в виртуальные, разумное размещение виртуальных нагрузок на подходящих физических узлах, а также единую консоль для управления ресурсами и их оптимизации. SCVMM обеспечивает следующие возможности:

- Централизованное управление серверами виртуальных машин в мас-

штабах предприятия. SCVMM поддерживает управление серверами Microsoft Hyper-V, Microsoft Virtual Server, VMware ESX и в будущем будет реализована поддержка Xen.

- Создание библиотеки шаблонов виртуальных машин. Шаблоны виртуальных машин представляют собой наборы образов предустановленных операционных систем, которые могут быть развёрнуты за считанные минуты.
- Мониторинг и размещение виртуальных машин в соответствие с загруженностью физических серверов.
- Миграция (конвертирование) физических серверов в виртуальные машины – технология *P2V*. Технология *P2V* позволяет произвести перенос физического сервера на виртуальный без остановки работы. Таким образом, появляется возможность онлайнового резервирования целого сервера, и в случае выхода его из строя, можно в течение минуты запустить виртуальный сервер и продолжить работу.
- Миграция (конвертирование) виртуальных машин других форматов в виртуальные машины Hyper-V – технология *V2V*. Данная технология аналогична *P2V*, но при этом позволяет переносить виртуальные машины Microsoft Virtual Server или VMware ESX в Hyper-V.
- Управление кластерами Hyper-V.

Краткие итоги:

В ходе данной лекции мы ознакомились с технологиями виртуализации, рассмотрели основные типы виртуализации. Также рассмотрели набор программных продуктов крупнейших компаний виртуализации.

Ключевые термины:

Виртуализация – процесс представления набора вычислительных ресурсов или их логического объединения, который даёт какие-либо преимущества перед оригинальной конфигурацией.

Виртуальная машина – программная или аппаратная среда, которая скрывает настоящую реализацию какого-либо процесса или объекта от его видимого представления.

Полная виртуализация – *Виртуализация*, при которой используются не модифицированные экземпляры гостевых операционных систем, а для поддержки работы этих ОС служит общий слой эмуляции их исполнения поверх хостовой ОС, в роли которой выступает обычная *операционная система*.

Паравиртуализация – *Виртуализация*, при которой производится модификация ядра гостевой ОС, выполняется таким образом, что в нее включается новый набор *API*, через который она может напрямую работать с аппаратурой, не конфликтуя с другими виртуальными машинами.

Виртуализация на уровне ОС – вид виртуализации, который подразумевает использование одного ядра хостовой ОС для создания неза-

висимых параллельно работающих операционных сред.

Виртуализация серверов – это *запуск* на одном физическом сервере нескольких виртуальных серверов. Виртуальные машины или сервера представляют собой приложения, запущенные на хостовой операционной системе, которые эмулируют физические устройства сервера. На каждой виртуальной машине может быть установлена *операционная система*, на которую могут быть установлены приложения и службы.

Виртуализация приложений – вид виртуализации, которая подразумевает применение модели сильной изоляции прикладных программ с управляемым взаимодействием с ОС, при которой виртуализируется каждый экземпляр приложений, все его основные компоненты: файлы (включая системные), реестр, шрифты, INI-файлы, COM-объекты, службы. *Приложение* исполняется без процедуры инсталляции в традиционном ее понимании и может запускаться прямо с внешних носителей.

Виртуализация представлений (рабочих мест) – *Виртуализация* представлений имеет *место*, когда *сервер* предоставляет свои ресурсы клиентам, причем клиентское *приложение* выполняется на этом сервере, а клиент получает только *представление*.

Монолитная архитектура гипервизора – *архитектура* гипервизора, при которой гипервизор размещается в едином уровне, который также включает большинство требуемых компонентов, таких как *ядро*, драйверы устройств и *стек* ввода/вывода.

Микроядерная архитектура гипервизора – подход, при котором

используется очень тонкий, специализированный гипервизор, выполняющий лишь основные задачи обеспечения изоляции разделов и управления памятью. Этот уровень не включает стека ввода/вывода или драйверов устройств.

§4. Облачные вычисления

4.1 Понятие облачных вычислений. История появления.

Цель данной лекции – получить сведения о появлении облачных вычислений, их преимуществах и недостатках.

Рисунок 4.1: Примеры применения концепции SaaS

Первые идеи об использовании вычислений как публичной услуги были предложены еще в 1960-х известным ученым в области информационных технологий, изобретателем языка *Lisp*, профессором *MIT* и Стэнфордского университета Джоном Маккарти (John McCarthy). Реализация первого реального проекта приписывается компании Salesforce.*com*,

основанной в 1999 году. Именно тогда и появилось первое предложение нового вида b2b продукта "*Программное обеспечение* как сервис" ("*Software as a Service* "SaaS"). Определенный успех Salesforce в этой области возбудил интерес у гигантов ИТ индустрии, которые спешно сообщили о своих исследованиях в области облачных технологий. И вот уже первое бизнес-решение под названием "Amazon *Web* Services"было запущено в 2005 году компанией Amazon.*com*, которая со времен кризиса доткомов активно занималась модернизацией своих датацентров. Следующим свою технологию постепенно ввела Google, начав с 2006 года b2b предложение SaaS сервисов под названием "Google Apps". И, наконец, свое предложение анонсировала компания Microsoft, презентовав ее на конференции PDC 2008 под названием "Azure Services *Platform*".

Рисунок 4.2: SaaS сервисы Google

Сам факт высокой заинтересованности крупнейших игроков рынка ИТ демонстрирует определенный статус облачных вычислений как тренда 2009-2010 годов. Кроме того, с релизом Microsoft Azure *Service Platform* множество экспертов связывает новый виток развития веб-технологий и *выход* всей сферы облачных вычислений на новый уровень.

Напомним, что под *облачными вычислениями мы понимаем программно-аппаратное обеспечение, доступное пользователю через Интернет или локальную сеть в виде сервиса, позволяющего использовать удобный интерфейс для удаленного доступа к выделенным ресурсам (вычислительным ресурсам, программам и данным).*

На данный момент большинство облачных инфраструктур развернуто на серверах датацентров, используя технологии виртуализации, что фактически позволяет любому пользовательскому приложению использовать вычислительные мощности, совершенно не задумываясь о технологических аспектах. Тогда можно понимать "облако" как единый *доступ* к вычислениям со стороны пользователя.

4.2 Виды облачных вычислений

С понятием облачных вычислений часто связывают такие сервис-предоставляющие (Everything as a service) технологии, как:

- **"Инфраструктура как сервис"** ("Infrastructure as a Service"или "IaaS")

- **"Платформа как сервис"**("Platform as a Service "PaaS")
- **"Программное обеспечение как сервис"** ("Software as a Service" или "SaaS").

Рассмотрим каждую из этих технологий подробнее.

4.3 Инфраструктура как сервис (IaaS)

IaaS - это предоставление компьютерной инфраструктуры как услуги на основе концепции облачных вычислений.

IaaS состоит из трех основных компонентов:

1. Аппаратные средства (серверы, системы хранения данных, клиентские системы, сетевое оборудование).

2. Операционные системы и системное ПО (средства виртуализации, автоматизации, основные средства управления ресурсами).

3. Связующее ПО (например, для управления системами).

Рисунок 4.3: Компоненты облачной инфраструктуры

IaaS основана на технологии виртуализации, позволяющей пользователю оборудования делить его на части, которые соответствуют текущим потребностям бизнеса, тем самым увеличивая эффективность использования имеющихся вычислительных мощностей. Пользователь (компания или разработчик ПО) должен будет оплачивать всего лишь реально необходимые ему для работы серверное время, дисковое пространство, сетевую пропускную способность и другие ресурсы. Кроме того, IaaS предоставляет в распоряжение клиента весь набор функций управления в одной интегрированной платформе.

IaaS избавляет предприятия от необходимости поддержки сложных инфраструктур центров обработки данных, клиентских и сетевых инфраструктур, а также позволяет уменьшить связанные с этим капиталь-

ные затраты и текущие расходы. Кроме того, можно получить дополнительную экономию, при предоставлении услуги в рамках инфраструктуры совместного использования.

Первопроходцами в IaaS считается компания Amazon, которые на сегодняшний день предлагают два основных IaaS-продукта: EC2 (**Elastic Compute Cloud**) и S3 (**Simple Storage Service**). EC2 представляет собой Xen-хостинг со статическими VPS-характеристиками, которые не расширяются на лету (хотя многие подобные сервисы уже предоставляют т.н. auto scaling). Хранилище S3 имеет интерфейс *WebDAV* и поддерживает работу со многими известными языками программирования.

Среди других инфра-сервисных компаний можно отметить:

GoGrid имеет очень удобный интерфейс для управления VPS, а также *cloud storage* с поддержкой протоколов SCP, FTP, SAMBA/CIFS, RSYNC, причем размер хранилища масштабируется на лету. В скором времени разработчики обещают добавить управление посредством API.

Enomaly представляет собой решение для развертывания и управления виртуальными приложениями в облаке, при этом управление услугами осуществляется через браузер. Приятным дополнением является автоматическое масштабирование виртуальных машин под текущую нагрузку, а также автобалансировка нагрузки. Среди поддерживаемых виртуальных архитектур поддерживаются Linux, Windows, Solaris и BSD Guests. Для виртуализации применяют не только Xen, но и *KVM*, а также VMware.

Eucalyptus представляет собой программный комплекс с открытым кодом для реализации *cloud computing* на кластерных системах. В настоящее время интерфейс совместим с Amazon EC2, но заявлена поддержка и других.

4.4 Платформа как сервис (PaaS)

PaaS - это предоставление интегрированной платформы для разработки, тестирования, развертывания и поддержки веб-приложений как услуги.

Для разворачивания веб-приложений разработчику не нужно приобретать оборудование и программное обеспечение, нет необходимости организовывать их поддержку. Доступ для клиента может быть организован на условиях аренды.

Такой подход имеет следующие достоинства:

- масштабируемость;
- отказоустойчивость;
- виртуализация;
- безопасность.

Масштабируемость *PaaS* предполагает автоматическое выделение и освобождение необходимых ресурсов в зависимости от количества обслуживаемых приложением пользователей.

PaaS как интегрированная платформа для разработки, тестирования, разворачивания и поддержки веб-приложений позволит весь перечень операций по разработке, тестированию и разворачиванию веб-приложений выполнять в одной интегрированной среде, исключая тем самым затраты на поддержку отдельных сред для отдельных этапов.

Способность создавать исходный код и предоставлять его в общий доступ внутри команды разработки значительно повышает производительность по созданию приложений на основе *PaaS*.

Самым известным примером такой платформы является AppEngine от Google, которая предлагает хостинг для веб-приложений с возможностью покупать дополнительные вычислительные ресурсы (например, для тестирования высоких нагрузок). Для запуска приложений Google AppEngine на виртуальных кластерных системах была разработана платформа AppScale, не имеющая, тем не менее, никакого отношения к Google.

В системах веб-поиска и контекстной рекламы компании Yahoo используется платформа Hadoop, ориентированная на передачу больших объемов данных между сетевыми серверами. На базе Hadoop построены HBase (аналог базы данных Google BigTable), а также *HDFS* (Hadoop Distributed File System, аналог Google File System).

Еще одним ярким представителем *PaaS* являются продукты компании Mosso:

- Cloud Sites – веб-хостинг (Linux, Windows, Mail) для нагрузочных

веб-проектов с возможностью расширять базовые бесплатные – возможности за дополнительную плату (трафик, хранилище данных, вычислительная мощность).

- Cloud Files – файловый cloud-хостинг с ежемесячной погигабайтной оплатой за объем хранимых файлов. Управление осуществляется через браузер, либо посредством API (PHP, Python, Java, .NET, Ruby).

- Cloud Servers – почасовая аренда серверов (RAM в час), с возможностью выбора серверной ОС. Можно изменять характеристики сервера, но не в режиме реального времени. В скором времени разработчики обещают сделать API для управления серверами.

Ну а в центре всей облачной инфраструктуры Microsoft – операционная система Windows Azure. Windows Azure создает единую среду, включающую облачные аналоги серверных продуктов Microsoft (реляционная база данных SQL Azure, являющаяся аналогом SQL Server, а также Exchange Online, SharePoint Online и Microsoft Dynamics CRM Online) и инструменты разработки (.NET Framework и Visual Studio, оснащенная в версии 2010 года набором Windows Azure Tools). Так, например, программист, создающий сайт в Visual Studio 2010, может не выходя из приложения разместить свой сайт в Windows Azure.

4.5 Программное обеспечение как сервис (SaaS).

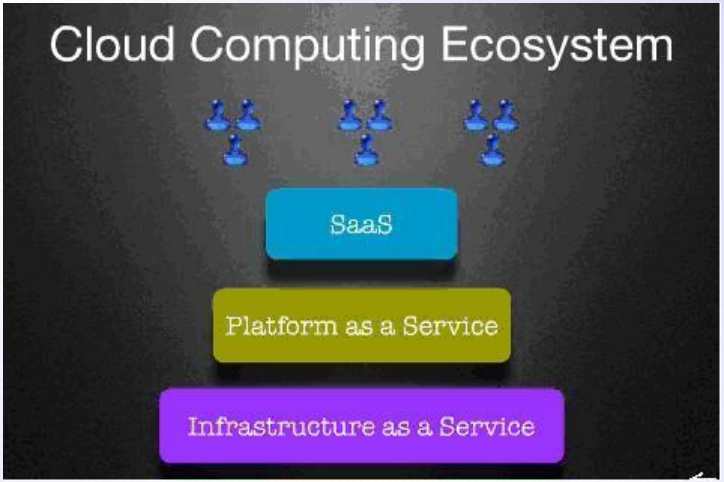


Рисунок 4.4: Вершина айсберга облачных технологий представлена сервисами SaaS

SaaS – модель развертывания приложения, которая подразумевает предоставление приложения конечному пользователю как услуги по требованию (on demand). Доступ к такому приложению осуществляется посредством сети, а чаще всего посредством Интернет-браузера. В данном случае, основное преимущество модели SaaS для клиента состоит в отсутствии затрат, связанных с установкой, обновлением и поддержкой работоспособности оборудования и программного обеспечения, работающего на нём. Целевая аудитория – конечные потребители.

В модели SaaS:

- приложение приспособлено для удаленного использования;
- одним приложением могут пользоваться несколько клиентов;
- оплата за услугу взимается либо как ежемесячная абонентская плата, либо на основе суммарного объема транзакций;
- поддержка приложения входит уже в состав оплаты;
- модернизация приложения может производиться обслуживающим персоналом плавно и прозрачно для клиентов.

С точки зрения разработчиков программного обеспечения, модель SaaS позволит эффективно бороться с нелицензионным использованием программного обеспечения, благодаря тому, что клиент не может хранить, копировать и устанавливать программное обеспечение.

По-сути, программное обеспечение в рамках SaaS можно рассматривать в качестве более удобной и выгодной альтернативы внутренним информационным системам.

Развитием логики SaaS является концепция *WaaS* (*Workplace* as a Service – рабочее место как услуга). То есть клиент получает в свое распоряжение полностью оснащенное всем необходимым для работы ПО виртуальное рабочее место.

По недавно опубликованным данным SoftCloud спросом пользуются следующие SaaS приложения (в порядке убывания популярности):

- Почта
- Коммуникации (VoIP)
- Антиспам и антивирус
- Helpdesk
- Управление проектами
- Дистанционное обучение
- CRM
- Хранение и резервирование данных

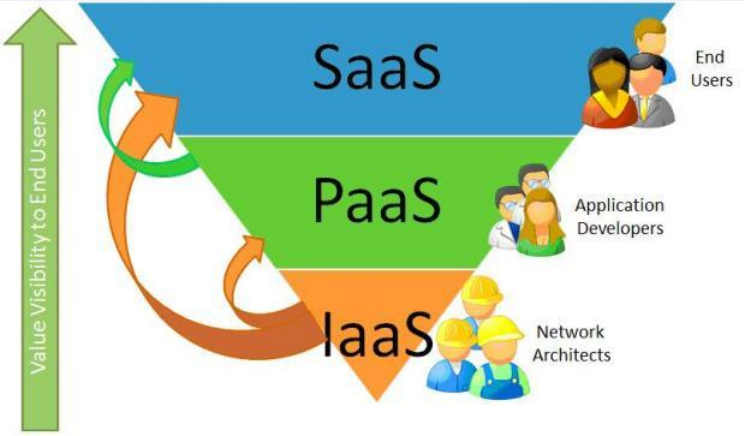


Рисунок 4.5: Сервисы SaaS имеют наибольшую потребительскую базу

Весьма схожими являются продукты MobileMe (Apple), Azure (Microsoft) и LotusLive (IBM). Суть данных сервисов в том, что они предоставляют пользователям доступ к хранению своих данных (контакты, почта, файлы), а также для совместной работы нескольких пользователей с документами.

Вопросами хранения пользовательских данных в Интернет озадачена и компания Google, которая разрабатывает проект GDrive, который будет представлять собой виртуальный жесткий диск, который будет определяться ОС как локальный. Также заявлено, что можно будет хранить неограниченное количество данных, что звучит весьма заманчиво.

Хранение файлов без ограничений также предлагает MediaFire.com. Имеется как полностью бесплатное использование (правда, с некоторыми ограничениями, например, на максимальный размер загружаемого файла), так и покупка премиум-аккаунта, расширяющего возможности (например, шифрование файлов, получение прямых ссылок на скачивание).

Еще одним интересным представителем вида SaaS является продукт iCloud, представляющий собой операционную систему, работать с которой можно непосредственно через браузер. Интерфейс операционной системы выполнен в стиле Windows Vista/XP. На сегодняшний день проект находится в стадии беты и в самой ОС реализован минимум приложений.

Также к SaaS относятся услуги *Online backup*, или, проще говоря –

резервному копированию данных. Пользователь просто платит абонентскую плату, а сервисы сами автоматически в определенное время шифруют данные с компьютера или другого устройства и отправляют их на удаленный сервер, тем самым данные могут быть доступны из любой точки земного шара. Данную услугу сейчас предоставляют множество компаний, в том числе, такие как Nero и Symantec.

Интересное применение cloud-технологиям нашли и разработчики компьютерных игр: теперь современным компьютерам и игровым приставкам не будут нужны мощные графические адаптеры (видеокарты), ведь вся обработка данных и рендеринг будут производиться cloud-серверами, а игроки будут получать уже обработанное видео. Одним из первых заявил о себе сервис OnLive, и совсем недавно об этом заговорила и компания Sony, которая собирается внедрить данную идею в Playstation 3.

Согласно SaaS-концепции пользователь платит не единовременно, покупая продукт, а как бы берет его в аренду. Причем, использует ровно те функции, которые ему нужны. Например, раз в год вам нужна некая программа. И чаще вы ее использовать не собираетесь. Так зачем же покупать продукт, который будет у вас лежать без дела? И зачем тратить на него место (в квартире, если это коробка с диском, на винчестере, если это файл)?

Конкуренция в облачной сфере привела к появлению бесплатных сервисов. Именно по такому пути пошли два конкурента – Microsoft и Google. Обе компании выпустили наборы сервисов, позволяющих рабо-

тать с документами. У Google это Google Docs, у Microsoft – Office Web Apps.

При этом оба сервиса тесно взаимосвязаны с почтой (Gmail в первом случае и Hotmail во втором) и файловыми хранилищами. Таким образом, пользователя как бы переводят из привычной ему оффлайн-среды в онлайн. Важно, что и Google, и Microsoft интегрируют поддержку своих онлайн-сервисов во все программные среды – как настольные, так и мобильные (напомним, что Google создала ОС *Android*, а Microsoft – Windows Phone 7).

Аналогичную концепцию (но с несколько другими акцентами) продвигает и главный конкурент обеих компаний – Apple. Речь идет об очень любопытном сервисе под названием MobileMe. Сервис включает в себя почтовый клиент, календарь, адресную книгу, файловое хранилище, альбом фотографий и инструмент для обнаружения утерянного iPhone. За возможность пользоваться всем этим Apple берет примерно 65 евро (или 100 долларов) в год. При этом Apple обеспечивает такой уровень взаимодействия своего набора интернет-сервисов и приложений на компьютере (под управлением Mac OS X), телефоне, плеере и iPad, что необходимость в использовании браузера пропадает. Вы пользуетесь привычными программами на своем Mac, iPhone и iPad, однако, все данные хранятся не на них, а в облаке, что позволяет забыть о необходимости синхронизации, а также – о их доступности.

Если Apple интегрирует веб-сервисы в привычные приложения опера-

ционной системы, то Google заходит с противоположной стороны: разрабатываемая интернет-гигантом операционная система Chrome OS представляет собой, фактически, один браузер, через который пользователь взаимодействует с разветвленной сетью веб-сервисов. ОС ориентирована на нетбуки, отмечаются очень низкие системные требования и отсутствие необходимости самостоятельной установки программ (так как все программы работают непосредственно в вебе). То есть Google предоставляет преимущества облачной концепции, обычно декламируемые при работе с корпоративными клиентами, обычным пользователям. Вместе с тем, очевидна невозможность использования таких нетбуков в странах с недостаточно широким проникновением широкополосного интернета. Потому что без интернета нетбук на базе Chrome OS будет совершенно бесполезен.

Все три типа облачных сервисов взаимосвязаны, и представляют вложенную структуру.

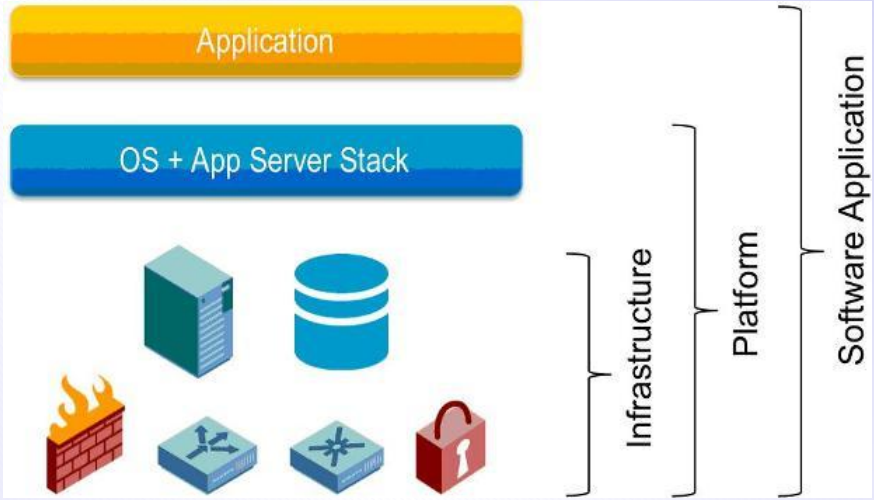


Рисунок 4.6: Взаимосвязь облачных сервисов

Помимо различных способов предоставления сервисов различают несколько вариантов развёртывания облачных систем:

4.6 Частное облако (private cloud)

Частное облако (private cloud) – используется для предоставления сервисов внутри одной компании, которая является одновременно и заказчиком и поставщиком услуг. Это вариант реализации "облачной концепции когда компания создает ее для себя самой, в рамках организации. В первую очередь реализация *private cloud* снимает один из важных

вопросов, который непременно возникает у заказчиков при ознакомлении с этой концепцией – вопрос о защите данных с точки зрения информационной безопасности. Поскольку "облако"ограничено рамками самой компании, этот вопрос решается стандартными существующими методами. Для *private cloud* характерно снижение стоимости оборудования за счет использования простаивающих или неэффективно используемых ресурсов. А также, снижение затрат на закупки оборудования за счет сокращения логистики (не думаем, какие сервера закупать, в каких конфигурациях, какие производительные мощности, сколько места каждый раз резервировать и т.д.

В сущности, мощность наращивается пропорционально растущей в целом нагрузке, не в зависимости от каждой возникающей задачи – а, так сказать, в среднем. И становится легче и планировать, и закупать и реализовывать – запускать новые задачи в производство.

4.7 Публичное облако

Публичное облако – используется облачными провайдерами для предоставления сервисов внешним заказчикам.

4.8 Смешанное (гибридное) облако

Смешанное (гибридное) облако – совместное использование двух вышеперечисленных моделей развёртывания.

Вообще одна из ключевых идей Cloud заключается как раз в том, чтобы с технологической точки зрения разницы между внутренними и внешними облаками не было и заказчик мог гибко перемещать свои задания между собственной и арендуемой ИТ-инфраструктурой, не задумываясь, где конкретно они выполняются.

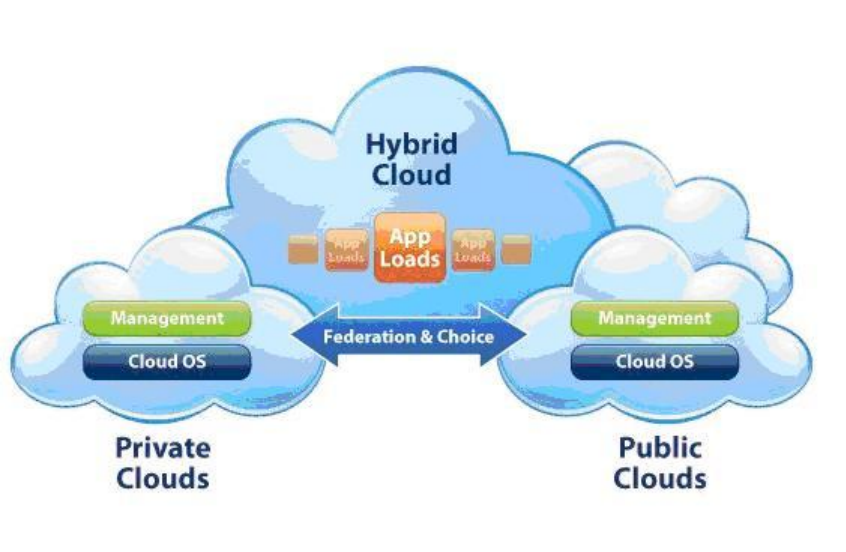


Рисунок 4.7: Взаимосвязь облаков разных типов

Таким образом, эти технологии при совместном использовании позволяют пользователям облачных вычислений воспользоваться вычислительными мощностями и хранилищами данных, которые посредством

определенных технологий виртуализации и высокого уровня абстракции предоставляются им как услуги.

4.9 Достоинства облачных вычислений

Рассмотрим основные преимущества и достоинства технологий облачных вычислений:

Доступность и отказоустойчивость – всем пользователям, из любой точки где есть Интернет, с любого компьютера, где есть браузер.

Клиентские компьютеры. Пользователям нет необходимости покупать дорогие компьютеры, с большим объемом памяти и дисков, чтобы использовать программы через веб-интерфейс. Также нет необходимости в CD и DVD приводах, так как вся информация и программы остаются в "облаке". Пользователи могут перейти с обычных компьютеров и ноутбуков на более компактные и удобные нетбуки.

Доступ к документам. Если документы хранятся в "облаке они могут быть доступны пользователям в любое время и в любом месте. Больше нет такого понятия как забытые файлы: если есть Интернет - они всегда рядом.

Устойчивость к потере данных или краже оборудования. Если данные хранятся в "облаке их копии автоматически распределяются по нескольким серверам, возможно находящимся на разных континентах. При краже или поломке персональных компьютеров пользователь не

теряет ценную информацию, которую он к тому же может получить с любого другого компьютера.

Надежность. Дата центры управляются профессиональными специалистами, обеспечивающими круглосуточную поддержку функционирования виртуальных машин. И даже если физическая машина "рухнет благодаря распределению приложения на множество копий оно все равно продолжит свою работу. Это создает определенный высокий уровень надежности и отказоустойчивости функционирования системы.

Экономичность и эффективность – плати столько, сколько используешь, позволь себе дорогие, мощные компьютеры и программы. "Облако"позволяет учитывать и оплачивать только фактически потребленные ресурсы строго по факту их использования;

Аренда ресурсов. Обычные сервера средней компании загружены на 10-15 %. В одни периоды времени есть потребность в дополнительных вычислительных ресурсах, в других эти дорогостоящие ресурсы простаивают. Используя необходимое количество вычислительных ресурсов в "облаке"в любой момент времени, компании сокращают затраты на оборудование и его обслуживание. Это дает возможность заказчику отказаться от закупок дорогостоящих ИТ-активов в пользу их даже не аренды, а операционного потребления по мере надобности, при сокращении затрат на обслуживание своих систем и получении от поставщика гарантий уровня сервиса.

Аренда ПО. Вместо приобретения пакетов программ для каждого ло-

кального пользователя, компании покупают нужные программы в "облаке". Данные программы будут использоваться только теми пользователями, которым эти программы необходимы в работе. Более того, стоимость программ, ориентированных на доступ через Интернет, значительно ниже, чем их аналогов для персональных компьютеров. Если программы используются не часто, то их можно просто арендовать с почасовой оплатой. Затраты на обновление программ и поддержку в работоспособном состоянии на всех рабочих местах вовсе сведены к нулю.

Для поставщика ИТ-услуг экономический смысл облака состоит в *эффекте масштаба* (обслуживать большой однородный центр обработки дешевле, чем множество маленьких разнородных) и сглаживания нагрузки (когда потребителей много, маловероятно, что пиковые мощности понадобятся всем им одновременно).

Разработчики ПО тоже получают выгоду от перехода в облака: теперь им стало проще, быстрее и дешевле разрабатывать, тестировать под нагрузкой и предлагать клиентам свои решения – это можно делать прямо в облаке с минимальными затратами. Кроме того, Облачные вычисления - это эффективный инструмент повышения прибыли и расширения каналов продаж для независимых производителей программного обеспечения в форме SaaS. Этот 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.

Простота – не требуется покупка и настройка программ и оборудования, их обновление.

Обслуживание. Так как физических серверов с внедрением *Cloud Computing* становится меньше, их становится легче и быстрее обслуживать. Что касается программного обеспечения, то последнее установлено, настроено и обновляется в "облаке". В любое время, когда пользователь запускает удаленную программу, он может быть уверен, что эта программа имеет последнюю версию - без необходимости что-то переустанавливать или платить за обновления.

Совместная работа. При работе с документами в "облаке"нет необходимости пересылать друг другу их версии или последовательно редактировать их. Теперь пользователи могут быть уверенными, что перед ними последняя версия документа и любое изменение, внесенное одним пользователем, мгновенно отражается у другого.

Открытые интерфейсы. "Облако"как правило, имеет стандартные открытые API (интерфейсы прикладного программирования) для связи с существующими приложениями и разработки новых – специально для облачной архитектуры.

Гибкость и масштабируемость - неограниченность вычислительных ресурсов (память, процессор, диски). "Облако"масштабируемо и эластично – ресурсы выделяются и освобождаются по мере надобности;

Производительные вычисления. По сравнению с персональным ком-

пьютером вычислительная мощь, доступная пользователю "облачных" компьютеров, практически ограничена лишь размером "облака то есть общим количеством удаленных серверов. Пользователи могут запускать более сложные задачи, с большим количеством необходимой памяти, места для хранения данных, тогда, когда это необходимо. Иными словами, пользователи могут при желании легко и дешево поработать с суперкомпьютером без каких-либо фактических приобретений. Возможность запуска множество копий приложения на многих виртуальных машинах представляет преимущества масштабируемости: количество экземпляров приложения способно практически мгновенно увеличиваться по требованию, в зависимости от нагрузок.

Хранение данных. По сравнению с доступным местом для хранения информации на персональных компьютерах объем хранилища в "облаке"может гибко и автоматически подстраиваться под нужды пользователя. При хранении информации в "облаке"пользователи могут забыть об ограничениях, накладываемых обычными дисками, - "облачные"размеры исчисляются миллиардами гигабайт доступного места.

Инструмент для стартапов. В глазах таких потребителей сервиса облачных вычислений как компании, начинающие свой бизнес основным преимуществом данной технологии является, отсутствие необходимости закупать все соответствующее оборудование и ПО, а затем поддерживать их работу.

4.10 Недостатки и проблемы облачных вычислений

Есть ли минусы у "облачных"вычислений? Почему "облачные" технологии в России только набирают обороты, а директора некоторых крупных компаний не спешат переводить ИТ-инфраструктуру своих предприятий в "облака"? Итак, отметим основные недостатки и трудности использования *cloud computing*:

Постоянное соединение с сетью. *Cloud Computing* почти всегда требует соединения с сетью (Интернет). Если нет доступа в сеть - нет работы, программ, документов. Многие "облачные"программы требуют хорошего Интернет-соединения с большой пропускной способностью. Соответственно программы могут работать медленнее чем на локальном компьютере. По мнению ведущих российских ИТ-компаний, основным препятствием широкому развитию облаков, является отсутствие широкополосного доступа в Интернет (ШПД) – прежде всего в регионах.

Безопасность.

Безопасность данных теоретически может быть под угрозой. Не все данные можно доверить стороннему провайдеру в интернете, тем более, не только для хранения, но ещё и для обработки. Все зависит от того, кто предоставляет "облачные"услуги. Если этот кто-то надежно шифрует Ваши данные, постоянно делает их резервные копии, уже не один год работает на рынке подобных услуг и имеет хорошую репутацию, то угрозы безопасности данных может никогда не случиться. У

пользователя "облачных"бизнес приложений могут также возникнуть и юридические проблемы, например связанные с выполнением требований защиты персональных данных.

Государство, на территории которого размещен датацентр, может получить доступ к любой информации, которая в нем хранится. Например, по законам США, где находится самое большое количество датацентров, в этом случае компания-провайдер даже не имеет права разглашать факт передачи конфиденциальной информации кому-либо, кроме своих адвокатов.

Эта проблема является, наверное, одной из самых существенных в вопросе вывода конфиденциальной информации в облако. Путей ее решения может быть несколько. Во-первых, можно шифровать всю информацию, помещаемую на облако. Во-вторых, можно просто ее туда не помещать. Однако, во всяком случае, у компаний, пользующихся облачными вычислениями, это должно быть определенным пунктом в списке вопросов информационной безопасности. Кроме того, сами провайдеры должны улучшать свои технологии, предоставляя некоторые услуги по шифрованию.

Функциональность "облачных"приложений. Не все программы или их свойства доступны удаленно. Если сравнивать программы для локального использования и их "облачные" аналоги, последние пока проигрывают в функциональности. Например, таблицы Google Docs или приложения Office web application имеют гораздо меньше функций

и возможностей, чем Microsoft Excel.

Зависимость от "облачного"провайдера.

Всегда остаётся риск, что провайдер онлайновых сервисов однажды не сделает резервную копию данных – как раз перед крушением сервера. Риск этот, впрочем, вряд ли превышает опасность того, что пользователь сам упустит свои данные – потеряв или разбив мобильник или ноутбук, не создав на домашнем ПК резервную копию. Кроме того, привязавшись к той или иной услуге, мы в какой-то степени также ограничиваем свою свободу – свободу перехода на старую версию софта, выбора способов обработки информации и так далее.

Некоторые эксперты, например Г. Маклеод (Hugh Macleod) в статье "Самый хорошо охраняемый секрет Облаков утверждают, что облачные вычисления ведут к созданию огромной, невиданной ранее монополии. Возможно ли это? Конечно, на рынке облачных вычислений для помещения в облако какой-либо информации, в отношении которой существуют правила информационной безопасности, компании будут скорее использовать таких вендоров, чье имя "на слуху"и кому они доверяют. Таким образом, существует определенная опасность того, что все вычисления и данные будут агрегированы в руках одной сверхмонополии. Однако на данный момент на рынке уже существуют несколько компаний с примерно одинаковым высоким уровнем доверия со стороны клиентов (Microsoft, Google, Amazon), и нет никаких фактов, которые бы указывали на возможность доминирования одной компанией всех остальных.

Поэтому в ближайшем будущем появление глобальной сверхкомпании, которая будет координировать и контролировать все вычисления в мире, очень маловероятно, хотя одна лишь возможность такого события отпугивает некоторых клиентов.

4.11 Препятствия развитию облачных технологий

Недостаточное доверие потребителей облачных услуг. Нередко бизнес относится к облачным услугам несколько настороженно. "Причин же недоверчивого отношения малого и среднего бизнеса к облачным дата-центрам может быть несколько. Скорее всего, это боязнь лишиться контроля над ИТ-ресурсами, опасения насчет гарантии сохранности и защиты переданной информации и представление дата-центра лишь как площадки для размещения оборудования".

Каналы связи в большинстве регионов страны характеризуются отсутствием SLA по качеству предоставляемого сервиса (QoS), что особенно относится к последним милям. Что толку от того, что ваш основной трафик идет по магистрали с гарантированным QoS (со своими ограничениями), если конечные условия подключены к ней через местного оператора, даже не слышавшего о такой проблеме. При этом стоимость связи для крупных организаций может составлять до 50 % от ИТ-бюджета. Соответственно переход к облачной модели существенно влияет на сетевую топологию ваших потоков данных и, скорее всего, QoS будет хуже

чем во внутренней сети. Или чтобы получить качество обслуживания на приемлемом уровне придется заплатить столько денег, что вся экономия от централизации инфраструктуры или приложений будет перечеркнута ростом коммуникационных затрат.

Безопасность. Проблема безопасности является серьезным сдерживающим фактором. Нередко Службы Безопасности создают довольно высокий заградительный барьер для идеи вынести какие-либо данные за периметр своей сети. Часто без какой-либо вменяемой аргументации.

Отсутствие надежных ЦОДов. По поводу центров обработки данных (ЦОД) достаточно вспомнить, что в стране, кажется, еще нет ни одного Tier III ЦОДа по классификации Uptime Institute. Совершенно понятно, что их появление – это вопрос времени. Из-за кризиса большинство строек было заморожено или отложено. Тем не менее, пока достаточной инфраструктуры в стране просто нет.

4.12 Распределенные вычисления (grid computing)

Отметим в заключение еще одну технологию, которая с одной стороны также оказала влияние на появление концепции облачных вычислений, а с другой стороны имеет ряд существенных отличий. Речь идет о коллективных, или *распределённых вычислениях (grid computing) – когда большая ресурсоёмкая вычислительная задача распределяется для выполнения между множеством компьютеров, объединённых в мощ-*

ный вычислительный кластер сетью в общем случае или интернетом в частности.

Установление общего протокола в сети Интернет непосредственно привело к быстрому росту онлайн пользователей. Это привело к необходимости выполнять больше изменений в текущих протоколах и к созданию новых. На текущий момент обширно используется протокол Ipv4 (четвёртая версия IP протокола), но ограничение адресного пространства, заданного ipv4, неизбежно приведет к использованию протокола ipv6. В течение долгого времени усовершенствовалось аппаратное и программное обеспечение, в результате чего удалось построить общий интерфейс в Интернет. Использование веб-браузеров привело к использованию модели "Облака взамен традиционной модели информационного центра.

В начале 1990-ых, Иэн Фостер и Карл Кесселмен представили их понятие *Грид* вычислений. Они использовали аналогию с электрической сетью, где пользователи могли подключаться и использовать услугу. *Грид* вычисления во многом опираются на методах, используемых в кластерных вычислительных моделях, где многократные независимые группы, действуют, как сеть просто потому, что они не все расположены в пределах той же области.

В частности, развитие Грид технологий позволило создать так называемые GRID-сети, в которых группа участников могла общими усилиями решать сложные задачи. Так, сотрудники IBM создали интернациональную команду grid-вычислений, позволившую существенно про-

двинуться в области борьбы с вирусом иммунного дефицита. Целые команды из разных стран присоединяли свои вычислительные мощности и помогли "обсчитать"и смоделировать наиболее перспективные формы для создания лекарства от СПИДа..."

На практике границы между этими (grid и cloud) типами вычислений достаточно размыты. Сегодня с успехом можно встретить "облачные"системы на базе модели распределённых вычислений, и наоборот. Однако будущее облачных вычислений всё же значительно масштабнее распределённых систем, к тому же не каждый "облачный сервис"требует больших вычислительных мощностей с единой управляющей инфраструктурой или централизованным пунктом обработки платежей.

Краткие итоги:

Мы рассмотрели основные понятия облачных вычислений, примеры, особенности, основные разновидности облачных технологий, их достоинства и недостатки.

Ключевые термины:

Облачные вычисления – технология обработки данных, в которой компьютерные ресурсы и мощности предоставляются пользователю как Интернет-сервис.

Инфраструктура как сервис – это предоставление компьютерной инфраструктуры как услуги на основе концепции облачных вычислений.

Платформа как сервис – это предоставление интегрированной плат-

формы для разработки, тестирования, развертывания и поддержки веб-приложений как услуги.

Программное обеспечение как сервис – модель развертывания приложения, которая подразумевает предоставление приложения конечному пользователю как услуги по требованию. Доступ к такому приложению осуществляется посредством сети, а чаще всего посредством Интернет-браузера.

Частное облако – это вариант локальной реализации "облачной концепции когда компания создает ее для себя самой, в рамках одной организации.

Публичное облако – используется облачными провайдерами для предоставления сервисов внешним заказчикам.

Распределенные вычисления – Технология когда большая ресурсоёмкая вычислительная задача распределяется для выполнения между множеством компьютеров, объединённых в мощный вычислительный кластер сетью или интернетом.

§5. Веб-службы в облачных вычислениях

Целью данной лекции является обзор веб-служб, предоставляемые концепцией облачных вычислений. Особое внимание уделяется типу "*Инфраструктура* как Сервис".

Поскольку технологии мигрируют от традиционной локальной модели в новую модель облака, сервисные предложения развиваются практически ежедневно. Предложения веб-служб часто имеют много общих характеристик. Часто от клиента требуются лишь минимальные *затраты* для получения услуги. Масштабируемость предполагается для каждого из типов предложений, но это не всегда необходимо. Многие из "облачных"вендоров еще работают над использованием масштабируемости, потому что их пользователи пока не нуждаются в данном виде услуг. Наконец, устройство и независимость местоположения позволяет пользователям получить доступ к системам независимо от того, где они находятся или какие устройства используют.

5.1 Инфраструктура как Сервис (IaaS)

Инфраструктура как Сервис (*Infrastructure-as-a-Service, IaaS*) – предоставление компьютерной инфраструктуры (как правило, это платформы виртуализации) как сервиса. IaaS существено усиливает технологию, услуги и вложения в центры обработки данных, чтобы предоставить это

как услугу клиентам. В отличие от традиционного аутсорсинга, который требует должного усердия, бесконечных переговоров и сложных, длинных контрактов, IaaS сосредоточенна вокруг модели предоставления услуг, которая обеспечивает предопределенную, стандартизированную инфраструктуру, определенно оптимизированную под потребности клиента. Упрощенные предложения работы и выбор уровня сервисного обслуживания облегчает клиенту выбор решения с определенным набором основных эксплуатационных характеристик. Как правило, поставщики предоставляют компоненты следующих уровней:

- Аппаратное обеспечение (как правило, Грид с массивной горизонтальной масштабируемостью);
- Компьютерная сеть (включая маршрутизаторы, брандмауэры, балансировку нагрузки и т.д.);
- Подключение Интернет;
- Платформа виртуализации для того, чтобы запускать виртуальные машины;
- Соглашения сервисного обслуживания;
- Инструменты учета вычислений.

Вместо покупки пространства в центрах обработки данных, серверов, программного обеспечения, сетевого оборудования, и т.д., клиенты IaaS

по существу арендуют ресурсы, которые находятся на стороне обслуживающих поставщиков услуг IaaS. Оплата за предоставление услуг обычно производится ежемесячно. Клиент платит только за потребленные ресурсы. Основные преимущества данного типа услуг включает:

- Свободный доступ к предварительно сконфигурированной окружающей среде;
- Использование инфраструктуры последнего поколения;
- Защищенные и изолированные вычислительные платформы;
- Уменьшение риска, за счет использования сторонних ресурсов, поддерживаемых третьими лицами;
- Способность управлять пиковыми нагрузками;
- Более низкие затраты;
- Меньшее время, стоимость и сложность при добавлении или расширении функциональности.

Вычисления по требованию приобретают все большую популярность среди предприятий. Вычислительные ресурсы, которые обслуживают пользовательские веб сайты, становятся все меньше и меньше, в то время, как доступные ресурсы поставщиков услуг постоянно возрастают. Модель по требованию развилась, чтобы преодолеть проблему того, как эффективно удовлетворить колеблющимся требованиям системы к ресурсам. Спрос на вычислительные ресурсы может существенно меняться за

достаточно короткие промежутки времени, и поддержка ресурсов достаточных, чтобы удовлетворить пиковым требованиям может быть дорогостоящей. Технически переусложненное решение может быть столь же неблагоприятной, как ситуация, когда предприятие сокращает издержки, поддерживая только минимальные вычислительные ресурсы. Такие понятия как кластерные вычисления, Грид вычисления и т.д., могут казаться очень подобными понятию вычислений по требованию, но лучше их понять можно, если думать о них, как стандартных блоках, которые развивались в течение долгого времени, чтобы достигнуть современной модели облачных вычислений, которую мы используем сегодня.

Рисунок 5.1: Грид вычисления

5.2 Amazon

Рассмотрим один из примеров – Amazon's Elastic Compute Cloud (Amazon EC2). *Amazon EC2* – веб-служба, которая обеспечивает вычислительные мощности порядочного размера в облаке. Это разработано, чтобы сделать веб-вычисления доступнее для разработчиков и чтобы предложить множество преимуществ для клиентов:

- Интерфейс веб-службы, позволяет клиентам получать и формировать пространство с минимальным усилием;
- Предоставляет пользователям полный контроль над их (арендованными) вычислительными ресурсами и позволяют им работать в проверенной вычислительной окружающей среде;
- Уменьшает время, требуемое для получения и загрузки нового сервера до минут, разрешая клиентам быстро изменять конфигурацию, согласно их вычислительным требованиям;
- Изменяет экономику вычислений, позволяя клиентам платить только за используемые ресурсы;
- Предоставляет разработчикам инструменты, которые необходимы для построения отказоустойчивых приложений и изолирования себя от общих сценариев отказа.

Amazon EC2 представляет вычислительную окружающую среду, разрешая клиентам использовать веб интерфейс, для получения и управле-

ния услугами, необходимыми для запуска одного или более экземпляров операционной системы. Клиенты могут загрузить окружающую среду с их настроенными приложениями. Они могут управлять сетевыми правами доступа и так много систем, сколько им нужно. Для использования *Amazon EC2*, клиентам сначала необходимо создать Amazon Machine Image (*AMI*). Этот образ содержит приложения, библиотеки, данные и связанные параметры конфигурации, используемые в виртуальной вычислительной среде. *Amazon EC2* предлагает использование предварительно сконфигурированные образы, созданные с шаблонами, необходимыми для немедленного запуска. Когда пользователи определили и сформировали их *AMI*, они используют инструменты *Amazon EC2* для загрузки образа в Amazon S3. *Amazon S3* – склад, который обеспечивает безопасный, надежный и быстрый доступ к клиенту *AMI*. Прежде, чем клиенты смогут использовать *AMI*, они должны использовать веб интерфейс *Amazon EC2* для настройки безопасности и сетевой доступ.

Во время конфигурации пользователи выбирают, какой тип категории и какую операционную систему они хотят использовать. Доступные типы категорий составляют две различные категории: Стандартный процессор или High-CPU процессор. Большинству приложений лучше всего удовлетворяет Стандартный случай, в который входят маленький, большой и очень большой экземпляры платформы. В случае High-CPU используется пропорционально больше ресурсов центрального процессора, чем памяти произвольного доступа (RAM), что более подходит

высоконагруженным приложениям. В случае High-CPU процессора для выбора есть средняя и очень большая платформы. После определения, какую сущность использовать, клиенты могут запустить, завершить, и контролировать так много экземпляров их *AMI*, сколько необходимо при использовании интерфейсов прикладного программирования веб-службы (API) или большое разнообразие других инструментов управления, которыми производится обслуживание. Пользователи могут выбрать, хотят ли они запускать приложения в разных центрах обработки данных, использовать статические IP адреса конечных точек, при этом они платят только за фактически потребляемые ресурсы. Пользователи также могут выбрать доступные *AMI* из библиотеки. Например, если необходим обычный Linux сервер, то клиентами может быть один из стандартных Linux сборок AMIs.

Есть довольно много особенностей сервиса EC2, которые обеспечивают существенные льготы для предприятия. Прежде всего, *Amazon EC2* обеспечивает финансовую выгоду. Из-за крупного масштаба компании Amazon и большой клиентской базы, это недорогая альтернатива многим другим возможным решениям. Затраты понесенные на запуск и управление разделены между большим количеством клиентов, делая стоимость для любого клиента намного ниже, чем любая другая альтернатива. Клиенты платят очень низкий процент за вычислительные мощности, которые они фактически потребляют. Безопасность также обеспечена через интерфейсы веб-службы *Amazon EC2*. Эти интерфей-

сы позволяют пользователям формировать параметры настройки брандмауэра, которые контролируют сетевой доступ к им между группами экземпляров служб. *Amazon EC2* предлагает очень надежную среду, где случаи замены могут быть быстро обеспечены.

- Динамическая Масштабируемость.

Amazon EC2 позволяет пользователям увеличивать или уменьшать производительность за несколько минут. Пользователи могут запускать единственный экземпляр, сотни экземпляров или даже тысячи экземпляров служб одновременно. Всем этим управляют с помощью API веб-службы, приложение может автоматически масштабировать себя вверх или вниз, в зависимости от его потребностей. Данный тип динамической масштабируемости очень привлекателен для клиентов предприятий, потому что это позволяет им соответствовать требованиям своих клиентов, не имея необходимость достраивать их инфраструктуру.

- Полный контроль над экземплярами.

У пользователей есть полный контроль над их экземплярами. У них есть полный доступ к каждому экземпляру, и они могут взаимодействовать с ними с любой машины. Экземпляры могут быть перезагружены, удаленно используя API веб-службы. Пользователи также имеют доступ к консоли своих экземпляров. Как только пользователи настроили их аккаунт и загрузили их *AMI* в сервис *Amazon S3*, им необходимо только запустить экземпляр. Возможно запустить *AMI* на любом числе

экземпляров (или для любого типа), вызвав RunInstances API, который поддерживается Amazon.

- Гибкость конфигурации.

Параметры настройки конфигурации могут значительно различаться среди пользователей. У них есть выбор из разных типов экземпляров, операционных систем, и пакетов программного обеспечения. *Amazon EC2* позволяет им выбирать конфигурацию памяти, центрального процессора, и системы хранения, которая оптимально подходит для их выбора операционной системы и приложения. Например, выбор пользователя операционной системы может также включать многочисленные сборки Linux, Microsoft Windows Server, и даже OpenSolaris, все запущенные на действительных серверах.

- Интеграция с Другими Веб-службами Amazon.

Amazon EC2 работает в соединении со множеством других веб-служб Amazon. Например, *Amazon Simple Storage Service (Amazon S3)*, *Amazon SimpleDB*, *Amazon Simple Queue Service (Amazon SQS)* и Amazon *CloudFront* все интегрированы, чтобы обеспечить полное решение для вычислений, обработки запросов и хранение между широким диапазоном приложений.

Amazon S3 обеспечивает интерфейс веб-служб, который позволяет пользователям хранить и восстанавливать любой объем данных через Интернет в любое время, где угодно. Это предоставляет разработчикам

прямой доступ к тому же самому, хорошо масштабируемому, надежному, быстрому, недорогому использованию инфраструктуры хранения данных Amazon, чтобы управлять их собственной глобальной сетью веб сайтов. Служба S3 стремится максимизировать преимущества масштабируемости и передать эти преимущества разработчикам.

Amazon SimpleDB - другой веб сервис, разработанный для того, чтобы выполнять запросы на структурированных данных *Amazon Simple Storage Service (Amazon S3)* в режиме реального времени. Этот сервис работает в соединении с *Amazon EC2*, чтобы предоставить пользователям возможность хранения, обработки и запросов наборов данных в пределах окружающей среды облака. Эти сервисы разработаны, чтобы сделать веб масштабируемые вычисления легче и более прибыльными для разработчиков. Традиционно данный тип функциональности был обеспечен использованием кластерной реляционной базы данных, которая требует значительных инвестиций. Внедрение данных технологий вызывало больше сложности и часто требовало услуг администрирования и поддержки базы данных.

В сравнении с традиционным подходом, *Amazon SimpleDB* легка в использовании и обеспечивает основную функциональность баз данных (например, поиск в реальном времени и запросы структурированных данных), не наследуя сложности эксплуатации, возникающие при традиционном выполнении. *Amazon SimpleDB* не требует схемы, данные индексируются автоматически, обеспечивает простой интерфейс API для

хранения и доступа к данным. Это избавляет клиентов от необходимости выполнить задачи, такие как моделирование данных, обслуживание индексов и настройка производительности.

Amazon Simple Queue Service (Amazon SQS) – сервис принимает очереди сообщений для хранения. При использовании Amazon SQS, разработчики могут просто переместить данные, распределённые между компонентами своих приложений, которые выполняют различные задачи, не теряя при этом сообщения. При этом достигается высокая масштабируемость и надёжность. Amazon SQS работает как демонстрация масштабируемой передачи сообщений Amazon, инфраструктуры как сервиса. Любой компьютер, связанный с Интернетом, может добавить или прочитать сообщения без необходимости в установке какого-либо программного обеспечения или специальный конфигурации брандмауэра. Компоненты приложений, использующие Amazon SQS, могут запускаться независимо и не должны обязательно размещаться в той же самой сети, использующей те же самые технологии или работающие в то же самое время.

Amazon CloudFront – веб-сервис для доставки контента (содержания). *Amazon CloudFront* интегрируется с другими Amazon Web Services. Цель сервиса – дать разработчикам и предприятиям простой способ распространять контент для конечных пользователей с низкой задержкой, высокой скоростью передачи данных, при этом не требуя никаких обязательств. Запросы объектов автоматически маршрутизируются на самый

близкий граничный сервер. Таким образом, содержание доставлено с лучшей возможной производительностью. Граничный сервер получает запрос от компьютера пользователя и соединяется с другим компьютером, вызывая оригинальный сервер, где расположено приложение. Когда оригинальный сервер выполняет запрос, он отправляет данные приложения назад на граничный сервер, который передает данные обратно компьютеру клиента, который выполнял запрос. Сервис не является свободным для пользования.

5.3 Платформа как Сервис (PaaS)

Развитие "облачных"вычислений привело к появлению платформ, которые позволяют создавать и запускать веб-приложения. *Платформа как сервис (Platform as a Service, PaaS)* – это предоставление интегрированной платформы для разработки, тестирования, развертывания и поддержки веб-приложений как услуги, организованная на основе концепции облачных вычислений.

Модель *PaaS* создает все условия требуемые для поддержки полного жизненного цикла создания и доставки веб-приложений и услуг доступных из сети Интернет, не требующих загрузки или установки программного обеспечения для разработчиков, ИТ менеджеров или конечных пользователей. В отличие от модели IaaS, где разработчики могут создавать определенные экземпляры операционных систем с домо-

рощенными приложениями, разработчики *PaaS* заинтересованы только веб разработкой и не заботятся о том, какая операционная система используется. *PaaS* сервисы позволяют пользователям сосредотачиваться на инновациях, а не на сложной инфраструктуре. Организации могут направить существенную часть их бюджета на создание приложений, которые обеспечивают реальную ценность, вместо затрат на поддержку инфраструктуры. Модель *PaaS* таким образом открывает новую эру массовых инноваций. Теперь разработчики во всем мире могут получить доступ к неограниченной вычислительной мощности. Любой человек, имеющий доступ в Интернет, может создавать приложения и легко разворачивать.

Традиционный подход создания и запуска локальных (On-*Premises*) приложений всегда был сложен, дорог и рискован. Строительство Вашего собственного решения никогда не предоставляло гарантии успеха. Каждое приложение было разработано, чтобы удовлетворить определенным деловым требованиям. Каждое решение потребовало определенной конфигурации аппаратных средств, операционной системы, базы данных, электронную почту, веб-серверы, и т.д. Когда была создана окружающая среда аппаратного и программного обеспечения, команда разработчиков должна была выбрать комплекс платформ для разработки, чтобы создавать приложения. Неизбежно бизнес требует от разработчиков производить изменения в приложении. Измененное приложение требует новых циклов испытательных работ, прежде чем быть распростра-

ненным. Крупные компании часто нуждаются в специализированных средствах, чтобы разместить их в центрах обработки данных. Огромное количество электричества необходимо для работы серверов и поддержки системы кондиционирования. Наконец, все это требует использования отказоустойчивых площадок для центров обработки данных так, чтобы информация могла копироваться в случае сбоя.

PaaS предлагает более быструю, более экономически выгодную модель для разработки и доставки приложений. *PaaS* обеспечивает всю инфраструктуру для запуска приложений через Интернет. Аналогичные сервисы предоставляют большое количество компаний, таких как Microsoft, Amazon.com, Google. *PaaS* основан на модели учета лицензий или модели подписки, таким образом, пользователи платят только за то, что они используют. Предложения *PaaS* включают рабочие процессы для создания приложений, разработки приложений, тестирования, развертывания и размещения. Также сервисы приложений, виртуальные офисы, командное сотрудничество, интеграцию баз данных, безопасность, масштабируемость, хранение, работоспособность, управление состоянием, инструментарий приборных панелей и много другое.

Главные особенности *PaaS* включают сервисы для разработки, тестирования, развертывания, размещения и управления приложениями для поддержки жизненного цикла разработки приложений. Веб интерфейсы инструментов создания, как правило, обеспечивают некоторый уровень поддержки чтобы упростить создание пользовательских интерфей-

сов, основанных на таких технологиях как HTML, JavaScript и других технологиях. Поддержка многопользовательской архитектуры помогает избежать проблем при разработке относительно использования приложений многими пользователями одновременно. Провайдеры *PaaS* часто включают услуги для управления параллельной обработкой, масштабируемостью, отказоустойчивостью и безопасностью. Другая особенность – это интеграция с веб-службами и базами данных. Поддержка протокола обмена структурированными сообщениями в распределённой вычислительной среде (Simple Object Access Protocol, SOAP) и других интерфейсов позволяют приложениям *PaaS* создавать комбинации веб-сервисов (которые называют mashup) так же легко, как наличие доступа к базам данных и повторному использованию услуг внутри частных сетей. Способность формировать и распространять код между специализированными, предопределенными или распределенными командами очень увеличивают производительность предложений вендоров *PaaS*. Интегрированные предложения *PaaS* обеспечивают возможность для разработчиков, чтобы наиболее хорошо понимать внутреннюю работу их приложений и поведение пользователей при использовании инструментов, подобных приборной панели, чтобы рассмотреть *внутренние параметры*, основанные на измерениях количества параллельных соединений и т.д. Некоторые предложения *PaaS* расширяют этот инструментарий, что позволяет составлять счета оплаты за использование.

5.4 Microsoft Azure

Платформа корпорации Майкрософт Windows Azure (первоначально известная под названием Azure Services Platform) – это группа "облачных"технологий, каждая из которых предоставляет определенный набор служб для разработчиков приложений. На рисунке 5.2 показано, что платформа Windows Azure может быть использована как приложениями, выполняющимися в "облаке так приложениями, работающими на локальных компьютерах.

Платформа Windows Azure состоит из следующих компонентов:

- Windows Azure. Обеспечивает среду на базе Windows для выполнения приложений и хранения данных на серверах в центрах обработки данных корпорации Майкрософт.
- SQL Azure. Обеспечивает службы данных в "облаке"на базе SQL Server.
- .NET Services. Обеспечивают распределенную инфраструктуру для "облачных"приложений и локальных приложений.

Каждый компонент платформы Windows Azure играет собственную роль. На высоком уровне понять Windows Azure очень легко. Это платформа для выполнения приложений Windows и хранения их данных в Интернете ("облаке"). На рисунке 5.3 показаны ее основные компоненты.

Рисунок 5.2: Платформа Windows Azure поддерживает приложения, данные и инфраструктуру, находящиеся в "облаке"

Как показано на рисунке, Windows Azure выполняется на большом количестве компьютеров, расположенных в центрах обработки данных корпорации Майкрософт, и доступна через Интернет. Общая структура подключения Fabric Windows Azure соединяет множество вычислительных мощностей в единое целое. Службы Windows Azure для вычисления и хранения построены на основе этой структуры.

Вычислительная служба Windows Azure, естественно, работает на базе Windows. Для обеспечения первоначальной доступности этой службы

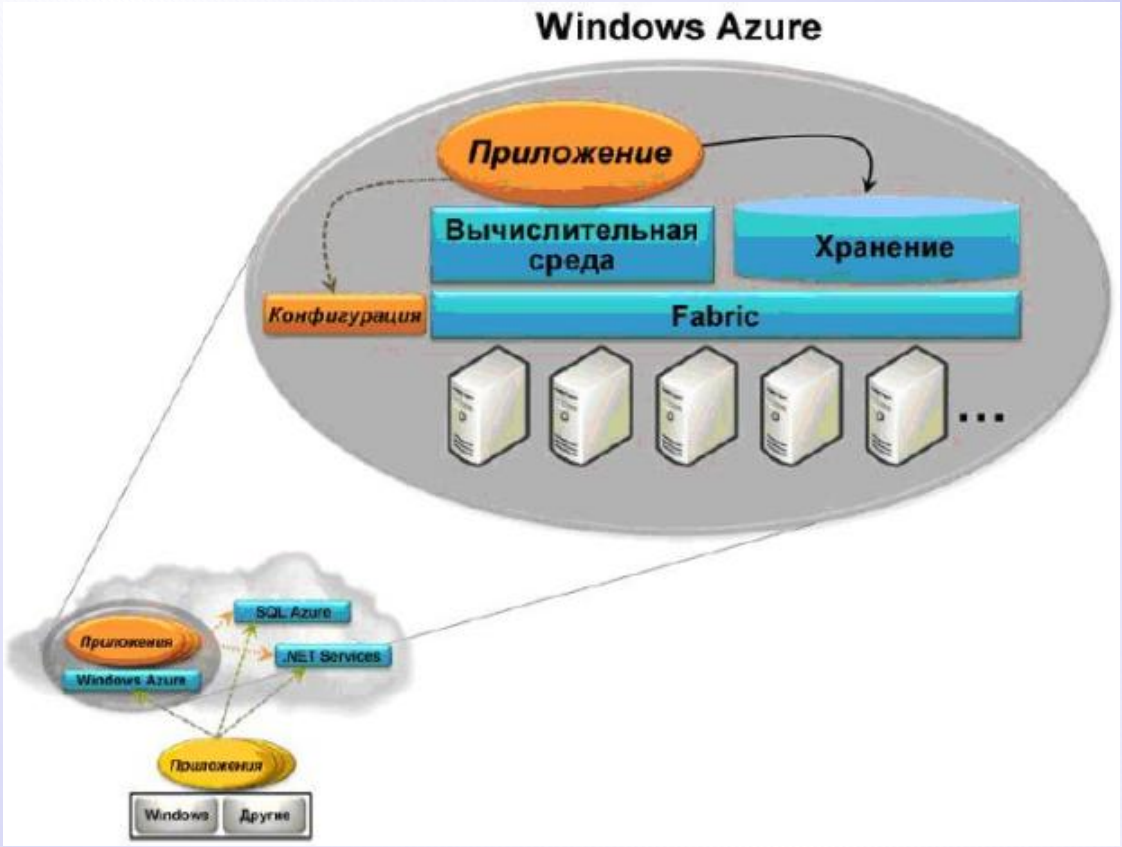


Рисунок 5.3: Windows Azure предоставляет "облачные"приложениям службы для вычисления и хранения на базе Windows

осенью 2008 г. была открыта для широкой публики *CTP*-версия. Корпорация Майкрософт разрешила выполнять на Windows Azure только

приложения, разработанные на платформе .NET Framework. Сегодня, однако, Windows Azure также поддерживает неуправляемый код, позволяя разработчикам выполнять приложения, которые разработаны не на базе .NET Framework. В любом случае такие приложения написаны на обычных языках Windows – C#, Visual Basic, C++ и других – с помощью Visual Studio 2008 или других средств разработки. Разработчики могут создавать веб-приложения с помощью таких технологий, как ASP.NET и Windows Communication Foundation (WCF), приложения, которые выполняются как независимые фоновые процессы, или приложения, сочетающие и то и другое.

Как приложения Windows Azure, так и локальные приложения могут получать доступ к службе хранилища Windows Azure, делая это одним и тем же способом: с помощью подхода RESTful. Однако Microsoft SQL Server не является базовым хранилищем данных. Фактически хранилище Windows Azure не относится к реляционным системам, и язык его запросов не SQL. Поскольку оно изначально предназначено для поддержки приложений на базе Windows Azure, то обеспечивает более простые и масштабируемые способы хранения. Следовательно, оно позволяет хранить большие двоичные объекты (binary large object – blob), обеспечивает создание очередей для взаимодействия между компонентами приложений и даже что-то вроде таблиц с простым языком запросов. (Для тех приложений Windows Azure, которым требуется обычное реляционное хранилище, платформа Windows Azure предоставляет базу

данных SQL Azure, описанную далее.)

Выполнение приложений и хранение их данных в Интернете имеет очевидные преимущества. Например, вместо того, чтобы покупать, устанавливать и эксплуатировать собственные компьютеры, организация может доверить все это поставщику услуг Интернета. При этом заказчики платят только за вычислительные мощности и хранилище, которое они используют, и не связаны с обслуживанием большого количества серверов, предназначенных только для пиковых нагрузок. Если приложения правильно написаны, их можно легко масштабировать, воспользовавшись преимуществами огромных центров обработки данных, которые могут предложить поставщики.

И все же для получения этих преимуществ требуется эффективное управление. В Windows Azure каждое приложение имеет файл конфигурации, как показано на рис. 2. Изменяя информацию в этом файле вручную или с помощью программы, владелец приложения может контролировать различные аспекты его поведения, такие как настройка количества экземпляров, которые должны выполняться на платформе Windows Azure. Структура Fabric платформы Windows Azure наблюдает за тем, чтобы приложение поддерживалось в требуемом состоянии.

Чтобы позволить своим заказчикам создавать, настраивать приложения и наблюдать за ними, Windows Azure предоставляет портал, доступный с помощью браузера. Заказчик предоставляет Windows Live ID, а затем решает, создавать ему учетную запись размещения для выполне-

ния приложений, учетную запись хранения для хранения данных или и ту и другую. Оплата использования приложения заказчиками может производиться любым удобным способом: с помощью подписки, повременно или как-нибудь иначе.

Windows Azure – это общая платформа, которую можно использовать в различных сценариях. Приведем несколько примеров, все они описываются с учетом возможностей *CTP*-версии.

- При создании нового веб-сайта, скажем, такого как Facebook, можно разрабатывать приложения на платформе Windows Azure. Благодаря тому, что данная платформа поддерживает как веб-службы, так и фоновые процессы, приложение может предоставлять интерактивный интерфейс пользователя и асинхронно выполнять работу для пользователей. Вместо того, чтобы тратить время и деньги, думая об инфраструктуре, пусковая группа может полностью сосредоточить свое внимание на разработке кода, который будет приносить пользу пользователям и инвесторам. Компания может также запустить небольшой веб-сайт, требующий незначительных затрат, если у ее приложения очень мало пользователей. Если приложение приобретает популярность и количество пользователей растет, Windows Azure при необходимости позволяет масштабировать его.

- Независимые поставщики программного обеспечения, создающие версию программы как службы имеющегося локального приложе-

ния Windows, могут разработать его на базе Windows Azure. Благодаря тому, что Windows Azure главным образом обеспечивает стандартную среду Windows, перемещение бизнес-логики приложения на эту "облачную"платформу не должно создавать особых проблем. Еще раз подчеркнем: разработка на базе имеющейся платформы позволяет независимым поставщикам ПО сосредоточить внимание на бизнес-логике, то есть на том, что позволяет им делать деньги, а не тратить время на инфраструктуру.

- Компания, создающая приложение для своих заказчиков, может выбрать для его разработки платформу Windows Azure. В силу того что Windows Azure поддерживает .NET, не представляет труда найти разработчиков с соответствующими навыками к тому же за разумную плату. Выполнение приложения в центрах обработки данных Microsoft освобождает предприятия от ответственности и расходов на поддержку собственных серверов, превращая капитальные затраты в эксплуатационные расходы. В особенности если у приложения имеются периоды пиковой нагрузки (если это, например, сетевой магазин цветов, которые необходимо вручить во время всеобщего ажиотажа 8 марта), предоставление корпорации Microsoft функции поддержки большой серверной базы, необходимой для этого, может оказаться экономически выгодным.

Выполнение приложений в "облаке"– один из самых важных аспектов

"облачных"вычислений. С помощью Windows Azure корпорация Microsoft обеспечивает как платформу для выполнения приложений, так и способ хранения данных. По мере того, как растет интерес к "облачным"вычислениям, ожидается создание еще большего количества приложений Windows для этой новой области.

Один из наиболее привлекательных способов использования серверов, доступных через Интернет, – это обработка данных. Цель SQL Azure – решить эту проблему, предлагая набор веб-служб для хранения самой разной информации и работы с ней. В то время, как представители Microsoft заявляют, что постепенно SQL Azure будет содержать целый ряд возможностей, ориентированных на данные, включая создание отчетов, анализ данных и многое другое, первыми компонентами SQL Azure станут база данных SQL Azure Database и средство синхронизации данных Huron. Это наглядно продемонстрировано на рисунке 5.4.

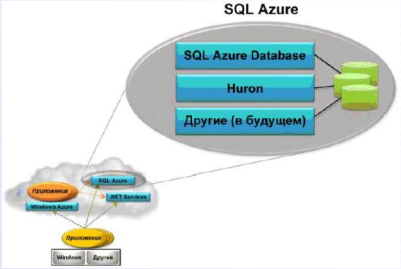


Рисунок 5.4: SQL Azure обеспечивает средства в Интернете, ориентированные на работу с данными

База данных SQL Azure Database (ранее известная под названием SQL Data Services) обеспечивает систему управления базами данных (СУБД) в Интернете. Эта технология позволяет локальным и веб-приложениям хранить реляционные и другие типы данных на серверах Microsoft в центрах обработки данных Microsoft. Так же как при работе с другими веб-технологиями, компания платит только за то, что использует, увеличивая и уменьшая объем использования (и затраты) по мере возникновения необходимости в изменениях. Использование базы данных в "облаке"также меняет характер капитальных затрат: на место инвестиций в жесткие диски и ПО для СУБД приходят эксплуатационные затраты.

В отличие от службы хранилища Windows Azure база данных SQL Azure разработана на основе Microsoft SQL Server. Тем не менее в первоначальной *CTP*-версии 2008 г. база данных SQL Azure Database не предоставляла традиционный реляционный подход к данным. Учитывая отзывы заказчиков, корпорация Microsoft решила внести соответствующие изменения. В дальнейшем база данных SQL Azure Database будет поддерживать реляционные данные, обеспечивая среду SQL Server в "облаке"с индексами, представлениями, хранимыми процедурами, триггерами и многим другим. Доступ к этим данным можно получить с помощью ADO.NET и других интерфейсов доступа к данным Windows. Фактически приложения, которые сегодня получают доступ к SQL Server локально, будут работать почти точно так же с данными в SQL Azure

Database. Для работы с этой информацией в "облаке"заказчики могут также использовать локальное ПО, такое как службы отчетов SQL Server.

В то время, как приложения могут использовать базу данных SQL Azure Database в значительной степени также, как локальную СУБД, требования к управлению существенно сокращены. Вместо того, чтобы беспокоиться о технике, например, обеспечивать мониторинг использования диска и обслуживание файлов журнала, заказчик SQL Azure Database может сосредоточить внимание на том, что действительно важно, на данных. Корпорация Microsoft будет отвечать за вопросы эксплуатации. Кроме того, так же как в случае с другими компонентами платформы Windows Azure, использование SQL Azure Database не составляет труда. Нужно просто зайти на веб-портал и предоставить необходимую информацию.

Второй компонент SQL Azure был заявлен под названием Huron Data Sync. Эта технология, разработанная на основе Microsoft Sync Framework и SQL Azure Database, позволяет синхронизировать реляционные данные в разных локальных СУБД. Владельцы данных могут определять, что именно должно синхронизироваться, как должны разрешаться конфликты и многое другое.

Приложения могут использовать SQL Azure самыми разными способами. Приведем несколько примеров.

- Приложение Windows Azure может хранить данные в SQL Azure Database. Несмотря на то, что Windows Azure обеспечивает собственное хранилище, реляционные таблицы не входят в число предлагаемых вариантов. Учитывая то, что многие имеющиеся приложения используют реляционное хранилище, а многие разработчики знают, как с ним работать, значительное количество приложений Windows Azure, скорее всего, будет работать с данными привычным способом, то есть с опорой на SQL Azure Database. Для повышения производительности заказчики могут указать, что определенное приложение Windows Azure должно выполняться в том же центре обработки данных, где SQL Azure Database хранит информацию этого приложения.

- В небольшой компании или подразделении большой организации приложение может использовать SQL Azure Database. Вместо того, чтобы хранить данные в базе данных SQL Server или Access, работающей на компьютере под чьим-то столом, приложение может использовать преимущества надежного и отказоустойчивого "облачного"хранилища.

- Предположим, производитель хочет сделать информацию о продукте доступной для своей дилерской сети и непосредственно для заказчиков. Размещение данных в SQL Azure Database позволит сделать их доступными для приложений, выполняемых на стороне

дилеров, и для ориентированных на заказчиков веб-приложений, которые выполняются на стороне производителя.

- Компания с клиентской базой данных, реплицированной в географически удаленных местах, должна использовать компонент Huron для синхронизации этих реплик. Возможно, в каждой из географических точек требуется собственная копия данных для повышения производительности, обеспечения доступности или по каким-то иным причинам. Автоматическая синхронизация может сделать такое обязательное распределение менее проблематичным.

Идет ли речь о приложении Windows Azure, обеспечении большей доступности данных, синхронизации этих данных или о чем-то еще, службы данных в Интернете могут оказаться очень полезными. По мере появления новых технологий в рамках SQL Azure организации будут получать возможность использования Интернета для выполнения все большего количества задач, ориентированных на работу с данными.

Выполнение приложений и хранение данных в Интернете относятся к важным аспектам вычислительной сетевой среды. Однако они далеко не исчерпывают ее возможности. Другая возможность заключается в обеспечении инфраструктуры служб на базе "облака которые могут использоваться локальными приложениями или веб-приложениями. Заполнить этот пробел и призваны службы .NET Services.

Первоначально известные как BizTalk Services, службы .NET Services

предлагают функции для решения общих проблем инфраструктуры при создании распределенных приложений. На рисунке 5.5 показаны их основные компоненты.

Рисунок 5.5: Службы .NET Services обеспечивают инфраструктуру в "облаке которая может быть использована для веб-приложений и локальных приложений

Службы .NET Services состоят из следующих компонентов.

- Управление доступом. Получающий все большее распространение подход к удостоверениям заключается в том, что каждый пользователь должен предоставить приложению маркер, содержащий некоторый набор утверждений. На основании этих утверждений приложение решает, что разрешено делать пользователю. Эффективное осуществление этой процедуры в масштабах компании требует федерации удостоверений, которая позволяет принимать утверждения, сделанные в одной области удостоверений, в другой области. Может также потребоваться преобразование утверждений, изменяющее их при передаче из одной области удостоверений в другую. Служба управления доступом обеспечивает реализацию обеих функций на основе "облака".

- Шина служб. Предоставление служб приложений в Интернете гораздо труднее, чем может показаться. Задача шины служб – упростить эту процедуру, позволяя приложениям предоставлять конечные точки веб-служб, доступ к которым может быть получен другими приложениями – локальными или работающими в "облаке". Каждой предоставленной конечной точке присваивается URI, который клиенты могут использовать для поиска службы и получения доступа к ней. Шина служб также решает проблему преобразования сетевых адресов и прохождения через межсетевые экраны без открытия новых портов для предоставленных приложений.

Приведем несколько примеров использования службы .NET Services.

- Независимый поставщик ПО, который поставляет приложение, необходимое заказчикам из разных организаций, может использовать службу управления доступом для упрощения разработки и эксплуатации приложения. Например, этот компонент .NET Services может преобразовывать различные утверждения, применяемые в разных организациях с различными технологиями идентификации в согласованный набор, подходящий для приложения независимого поставщика ПО. Такое преобразование позволяет также разгрузить механизм федерации удостоверений за счет службы управления доступом, освобождая независимых поставщиков ПО от необходимости выполнения собственной локальной программы федерации.

- Предположим, предприятие хочет открыть доступ к одному из своих приложений торговым партнерам. Оно может распределить функции приложения с помощью веб-служб SOAP или RESTful и зарегистрировать их конечные точки с помощью шины служб. Затем торговые партнеры могут использовать шину для поиска конечных точек и доступа к службам. Это позволяет снизить риски, связанные с предоставлением приложения, поскольку не требует открытия новых портов в межсетевом экране компании. Организация может также использовать службу управления доступом, предназначенную для работы с шиной служб, для рационализации сведений о

проверке подлинности, отправленной приложению ее партнерами.

Так же как в случае Windows Azure, предоставляется портал, доступный с помощью браузера, чтобы дать заказчикам возможность использовать службы .NET Services с помощью Windows Live ID. Цель корпорации Microsoft, достигаемая с помощью .NET Services, совершенно очевидна: обеспечить полезную "облачную" инфраструктуру для распределенных приложений.

5.5 Программное обеспечение как Сервис (SaaS)

Программное обеспечение как сервис (Software as a service, SaaS) или программное обеспечение по требованию (Software on Demand,*SoD*) – бизнес-модель продажи программного обеспечения, при которой поставщик разрабатывает веб-приложение и самостоятельно управляет им, предоставляя заказчикам доступ к программному обеспечению через Интернет. Основное преимущество модели SaaS для потребителя состоит в отсутствии затрат, связанных с установкой, обновлением и поддержкой работоспособности оборудования и работающего на нём программного обеспечения. Программное обеспечение как сервис является моделью *распространения программного обеспечения*, в которой приложения размещены у вендора SaaS или поставщика услуг и доступны для клиентов по сети, как правило, Интернет. Модель SaaS доставки приложений становится все более и более распространенной технологией,

которая поддерживает веб-службы и сервис-ориентированную архитектуру (SOA). SaaS также часто ассоциирована с моделью лицензирования, когда оплата происходит по мере получения услуг. Тем временем, услуги широкополосных сетей стали все более и более доступными, для поддержки доступа пользователей из большего количества мест по всему миру.

Огромные успехи, достигнутые поставщиками услуг интернет (ISP), чтобы увеличить полосу пропускания и сохранить возможность использования более мощных микропроцессоров вместе с недорогими устройствами хранения данных. Это обеспечивает огромную платформу для того, чтобы проектировать, разворачивать и использовать программное обеспечение через все области бизнес- и частных вычислений. Приложения SaaS также должны быть в состоянии взаимодействовать с другими данными и другими приложениями среди большого разнообразия окружающих сред и платформ. Компания IDC описывает две немного отличающихся модели поставки SaaS.

SaaS чаще всего предназначен для обеспечения бизнес функциональности программного обеспечения для корпоративных клиентов по низкой цене, что позволяет избавиться от установки, управления, поддержки, лицензирования и высоких затрат в компании. Большинству клиентов неинтересно знать, как или почему программное обеспечение реализовано, развернуто и т.д., но все они, в тоже время, имеют потребность в использовании программного обеспечения в их работе. Многие типы

программного обеспечения хорошо удовлетворяют модели SaaS (например, бухгалтерский учет, работа с клиентами, электронная почта, учет трудовых ресурсов, ИТ безопасность, управление ИТ, видеоконференц-связь, *веб-аналитика*, управление веб-контентом). Различие между SaaS и более ранними способами доставки приложений через Интернет в том, что решения SaaS были разработаны специально, чтобы работать с веб браузерами. Архитектура приложений на основе SaaS специально предназначена для поддержки обработки запросов от большого количества пользователей. В этом и заключается большая разница между традиционным клиент-серверным приложением решением, расположенным у поставщиков услуг. С другой стороны, поставщики услуг SaaS увеличивают экономию масштабирования при развертывании, управлении, поддержке и обслуживании их предложений.

Много типов компонентов программного обеспечения и Фреймворков могут быть использованы при разработке приложений SaaS. Используя новые технологии в этих современных компонентах и средах разработки приложений, можно значительно уменьшить время разработки и стоимости преобразования традиционного продукта в решение SaaS. Согласно Microsoft, SaaS архитектура может быть классифицирована в один из четырех уровней, с ключевыми признаками: простота конфигурации, эффективность при многопользовательском доступе и масштабируемость. Каждый уровень отличается от предыдущего добавлением одного из этих признаков. Рассмотрим уровни, описанные Microsoft:

- **Архитектурный Уровень 1** – Специальный/Настраиваемый.

Первый уровень является фактически самым низким. Каждый клиент имеет уникальную, настроенную версию размещаемого приложения. Приложение запускает свои собственные экземпляры на серверах. Миграция традиционных несетевых или клиент-серверных приложений на этот уровень SaaS, как правило, требует незначительных усилий при разработке и уменьшает эксплуатационные расходы, благодаря объединению серверного аппаратного обеспечения и администрирования.

- **Архитектурный Уровень 2** – Конфигурируемость.

Второй уровень SaaS обеспечивает большую гибкость программы благодаря метаданным конфигурации. На данном уровне клиенты могут использовать много отдельных экземпляров одного приложения. Это позволяет вендорам удовлетворять переменные потребностям каждого клиента при использовании детализированной конфигурации. Также облегчается обслуживание, появляется возможность обновить общую кодовую базу.

- **Архитектурный Уровень 3** – Эффективность Мультиарендатора.

Третий уровень отличается от второго наличием поддержки многопользовательского доступа. Единственный экземпляр программы способен обслужить всех пользователей. Данный подход позволяет более эф-

фективно использовать ресурсы сервера незаметно для конечного пользователя, но, в конечном счете, этот уровень не позволяет выполнять масштабирование системы.

- **Архитектурный Уровень 4** – Масштабируемость.

В четвертом Уровень SaaS, масштабируемость добавлена благодаря использованию многоуровневой архитектуры. Эта архитектура способна поддерживать распределение нагрузки фермы идентичных экземпляров приложений, запущенных на переменном количестве серверов, которое достигает сотен и даже тысяч. Мощность системы может быть динамически увеличена или уменьшена в соответствии с требованиями. Это осуществляется путем добавления или удаления серверов без необходимости для дальнейшего изменения прикладной архитектуры программного обеспечения.

Развертывание приложений в сервис-ориентированной архитектуре является более сложной проблемой, чем развертывание программного обеспечения в традиционных моделях. В результате стоимость использования приложения SaaS основывается на числе пользователей, которые осуществляют доступ к сервису. Довольно часто возникают дополнительные расходы, связанные с использованием услуг сервисной службы, дополнительной полосы пропускания, и дополнительного дискового пространства. Доходы поставщиков услуг SaaS обычно первоначально ниже, чем традиционный расходы за лицензии на программное обеспе-

чение. Однако компромисс для более низких затрат лицензии – ежемесячно возвращающий доход, который рассматривается финансовым директором компании, как более предсказуемый *критерий существования* бизнеса. К ключевым особенностям программного обеспечения SaaS относятся:

- Управление по сети и сетевой доступ к коммерческому программному обеспечению в централизованных центрах обработки данных, а не на сайтах клиентов, предоставление возможности клиентам получить доступ к приложениям удаленно через Интернет.
- Доставка приложений по модели "один ко многим в противоположность традиционной модели "один к одному".
- Централизованная модернизация и обновления, что позволяет избежать необходимости в загрузке и установке приложений пользователем. SaaS часто используется в крупных сетях коммуникаций и программного обеспечения для совместной работы, иногда как программное расширение к архитектуре *PaaS*.

Циклы разработки программ в компаниях могут занимать достаточно долгое время, потребляя большие ресурсы и приводя к неудовлетворительным результатам. Хотя решение уступить контроль является трудным, это может привести к улучшению эффективности, снижению рисков и сокращению расходов. Постоянно увеличивается число компаний, которые хотят использовать модель SaaS для корпоративных приложе-

ний, таких как работа с клиентами, финансовые расходы, управление персоналом. Модель SaaS гарантирует предприятиям, что все пользователи системы используют правильную версию приложения и поэтому формат зарегистрированных и переданных данных корректен, совместим и точен. Возлагая ответственность за приложения на поставщика SaaS, предприятия могут уменьшить затраты на администрирование и управление, которые необходимы для поддержки собственного корпоративного приложения. SaaS увеличивает доступность приложений в сети Интернет. SaaS гарантирует, что все транзакции приложения зарегистрированы. Преимущества SaaS для клиентов достаточно понятны:

- Рациональное управление;
- Автоматизированное обновление и исправление;
- Целостность данных в рамках предприятия;
- Совместная работа сотрудников предприятия;
- Глобальная доступность.

Серверная виртуализация может использоваться в архитектуре SaaS вместо или в дополнение к поддержке многопользовательского режима. Главное преимущество платформы виртуализации – увеличение производительности системы без необходимости в дополнительном программировании. Эффект объединения совместного использования ресурсов

и платформы виртуализации в решение SaaS обеспечивает большую гибкость и производительность для конечного пользователя.

5.6 Коммуникация как Сервис (CaaS)

Коммуникация как Сервис (CaaS) - построенное в облаке коммуникационное решение для предприятия. Поставщики этого типа облачного решения отвечают за управление аппаратным и программным обеспечением, требуемым для того, чтобы предоставить:

- систему связи, обеспечивающую передачу речевого сигнала по сети Интернет или по любым другим IP-сетям (VoIP),
- обмен мгновенными сообщениями (IM),
- видеоконференц-связь.

Эта модель начала свой эволюционный процесс в индустрии телекоммуникаций, не сильно отличаясь от модели SaaS, стала результатом сектора служб доставки программного обеспечения. Вендоры *CaaS* ответственные за управление аппаратным и программным обеспечением их пользователей. Вендоры *CaaS*, как правило, предоставляют гарантируемое качество обслуживания (QoS) в соответствии с соглашением сервисного обслуживания (SLA).

Модель *CaaS* позволяет деловым клиентам выборочно разворачивать средства коммуникаций и услуг на оснований оплаты услуг в срок для

используемых сервисов. *CaaS* разработан на ценовой политике общего назначения, которая предоставляет пользователям всесторонний, гибкий и легкий в понимании сервисный план. Согласно Gartner, рынок *CaaS*, как ожидается, будет насчитывать $2,3 миллиарда в 2011 году, с ежегодным темпом роста более 105 %.

Сервисные предложения *CaaS* часто связаны и включают интегрированный доступ к традиционному голосу (или VoIP) и данным, дополнительная функциональность объединенных коммуникаций, такие как видео вызовы, совместная работа, беседы, присутствие в реальном времени и передача сообщений, телефонная сеть, местная и распределенная голосовые услуги, голосовая почта. *CaaS* решение включает избыточное переключение, сеть, избыточность оборудования, WAN *failover* – что определенно подходит к потребностям клиентов. Все транспортные компоненты VoIP расположены в географически распределенных, безопасных информационных центрах для высокой доступности и жизнеспособность. *CaaS* предполагает гибкость и масштабируемость для мелкого и среднего бизнеса, чего зачастую сами компании не могут обеспечить. Поставщики услуг *CaaS* подготовлены к пиковым нагрузкам, оказывают услуги по расширению емкости устройств, состояний или области покрытия по требованию заказчика. Пропускная способность сети и наборы средств могут быть изменены динамически, таким образом, функциональность идет в ногу с потребительским спросом и ресурсы, находящиеся в собственности поставщика не используются впустую. В отличие

от поставщика услуг, перспектива клиента фактически не приводит к риску обслуживания устаревшего оборудования, так как обязательства поставщика услуг *CaaS* заключается в том, чтобы периодически модернизировать или заменять аппаратное и программное обеспечение, чтобы подерживать платформу в технологически актуальном состоянии.

CaaS не требует контроля от клиентов. Это избавляет от необходимости клиентов совершать какие-либо капиталовложения в инфраструктуру, и это устраняет накладные расходы для инфраструктуры. С решением *CaaS* клиенты в состоянии усиливать коммуникационные услуги класса предприятия, не имея необходимости к построению собственное решение внутри своей организации. Это позволяет клиентам перераспределять бюджет и трудозатраты персонала, использовать их в тех местах, где это наиболее необходимо.

От телефонной трубки, которую можно найти на столе каждого сотрудника до клиентского программного обеспечения на ноутбуке сотрудника, VoIP частная основа, и все необходимые действия между каждым из компонентов в решении *CaaS* поддерживаются в режиме 24/7 поставщиком услуг *CaaS*.

- Решения размещения и управления

Удаленное управление услугами инфраструктуры обеспечивается третьими лицами, казалось недопустимой ситуацией для большинства компаний. Однако за прошлое десятилетие с развитием технологий, органи-

зацией сети и программным обеспечением отношение изменилось. Это частично связано со снижением издержек при использовании выбранных услуг. Однако в отличие от единичных услуг предложение поставщиков услуг *CaaS* предоставляет полное коммуникационное решение, которое является полностью управляемый одним вендором. Наряду с особенностями, такими как VoIP и объединенные коммуникации, интеграция офисной автоматической телефонной станции с дополнительной функциональностью управляется одним вендором, который ответственен за всю интеграцию и доставку услуг пользователям.

- Удобство управления и функциональность

Когда клиенты пользуются услугами связи на стороне поставщика услуг *CaaS*, они платят только за необходимую функциональность. Поставщик услуг может распределять стоимость услуг. Как отмечалось ранее, это способствует более экономичному внедрению и использованию общей необходимой функциональности для клиентов. Экономия за счет роста производства позволяет поставщикам услуг производить обслуживание достаточно гибко, они не привязаны к единственному поставщику инвестиций. Поставщики услуг в состоянии усилить решения лучших среди аналогичных поставщиков, таких как Microsoft, Google, Amazon, Cisco, Nortel более экономично.

- Нет затрат средств на оборудования

Все оборудование расположено у поставщиков услуг *CaaS*, это фактически избавляет от необходимости клиентов поддерживать собственные информационные центры и оборудование. Отсутствуют расходы средств на электропотребление, охлаждение, аренду помещений. Клиенты получают многократную выгоду, используя центры обработки данных масштаба крупных компаний с полным резервированием – и это все включено в ежемесячную оплату.

- Гарантируемая непрерывность бизнеса

Позволяет ли Ваш план аварийного восстановления после катастрофических событий в центре обработки данных продолжать непрерывно работать Вашему бизнесу? Как долго Ваша компания может работать при отключении электроэнергии? Для большинства компаний эти события неизбежно означают ощутимые финансовые потери, связанные с простоем бизнеса. Распределение информационной системы компании между географически распределенными центрами обработки данных становятся нормой для все большего числа компаний. Это смягчает риск финансовых потерь и позволяет компаниям расположенным в месте, где произошли какие либо катастрофические события, восстанавливать инфраструктуру так скоро, насколько это возможно. Этот процесс осуществлен поставщиками услуг *CaaS*. Для большого количества компаний, работающий с голосовой передачей данных, перебои в работе системы являются катастрофическими. В отличие от целостно-

сти данных, устранение единственных точек отказа для голосовой сети является обычно достаточно дорогостоящим из-за крупного масштаба и сложности управления проектом. Решения *CaaS* обладают многократными уровнями избыточности системы, что исключает из системы единые точки отказа.

5.7 Мониторинг как Сервис (MaaS)

Мониторинг как Сервис (Monitoring-as-a-Service, MaaS) является обслуживаемым в облаке обеспечением безопасности, прежде всего на бизнес платформах. За прошлое десятилетие *MaaS* стал все более и более популярным. С появлением облачных вычислений, популярность *MaaS* стала больше. *Контроль безопасности* затрагивает защиту клиентов – предприятий или правительства от кибер угроз. Служба безопасности играет важную роль в обеспечении и поддержании конфиденциальность, целостность, и доступность средств ИТ. Однако время и ограниченные ресурсы ограничивают мероприятия безопасности и их эффективность для большинство компаний. Это требует постоянной бдительности безопасности инфраструктуры и критических информационных средств. Много промышленных правил требуют, чтобы организации контролировали свою среду безопасности, журналы серверов, и другие информационные средства, чтобы гарантировать целостность этих систем. Однако обеспечение эффективного контроля состояния безопасности может

быть пугающей задачей, потому что она требует передовых технологий, квалифицированных экспертов по безопасности, и масштабируемые процесс, ни один из которых не является дешевым. Сервисы контроля состояния безопасности *MaaS* предлагает контроль в реальном времени, в режиме 24/7 и практически немедленные реагирование по инцидентам через инфраструктуру безопасности. Эти сервисы помогают защитить критические информационные активы клиентов. До появления электронных систем обеспечения безопасности, контроль состояния безопасности и реагирование зависели в большой степени от человеческих ресурсов и человеческих способностей, которые ограничивали правильность и эффективность контролирующих усилий. За прошедшие два десятилетия, были разработаны информационные технологии в системах обеспечения безопасности, которые способны взаимодействовать с центрами операционной безопасности *(SOC)* через корпоративные сети, что значительно изменило картину. Данные средства включают две важных вещи:

1. Общая стоимость владения центром операционной безопасности намного выше, чем для современной технологии *SOC*;

2. Достижение более низких операционных затрат безопасности и более высокая эффективность средств безопасности.

SOC услуги контроля состояния безопасности могут улучшить эффективность *инфраструктура безопасности* клиента, активно анализируя журналы и оповещения от устройств инфраструктуры круглосуточ-

но и в режиме реального времени. Контроль команд соотносит информацию с различных устройств безопасности, чтобы предоставить аналитикам по безопасности данные, необходимые им для устранения ложный угроз и для реагирования на истинный угрозы предприятия. Служба информационной безопасности может оценить производительность системы на периодически повторяющейся основе и обеспечить рекомендации для усовершенствований если необходимо.

Сервис раннего обнаружения сообщает о новых слабых местах в безопасности вскоре после того, как они появляются. Вообще, угрозы взаимосвязаны с источниками, имеющими отношение к третьей стороне. Отчет обычно посылается по электронной почте ответственному человеку, назначенному компанией. Отчеты об уязвимости безопасности, кроме содержания подробного описания уязвимости, также включает информацию о влиянии данной уязвимости на систему или приложение. Наиболее часто отчет также указывает на определенные действия, которые нужно выполнить, чтобы минимизировать эффект уязвимости.

Платформа, управление и мониторинг сервиса часто предоставляются как приборная панель, что позволяет в любое время узнать рабочее состояние системы. Доступ можно получить через веб-интерфейсы, что позволяет работать удаленно. Каждый рабочий элемент, который проверяется обычно содержит рабочий индикатор статуса, всегда принимая во внимание критическое воздействие каждого элемента. Данные сервиса позволяют определить, какие элементы находятся в рабочем состоянии,

каким не хватает мощности, а какие находятся за пределами установленных параметров. Обнаруживая и идентифицируя такие проблемы, можно принимать профилактические меры, для предотвращения потери работоспособности сервиса.

Краткие итоги:

В данной лекции была рассмотрена технология предоставления веб-сервисов "*инфраструктура* как сервис". Был получен базовый материал о технологиях ведущих вендоров Amazon и Microsoft.

Ключевые термины:

Инфраструктура как Сервис, IaaS - предоставление компьютерной инфраструктуры (как правило, это платформы виртуализации) как сервиса.

Платформа как сервис, PaaS – предоставление интегрированной платформы для разработки, тестирования, развертывания и поддержки веб-приложений как услуги, организованная на основе концепции облачных вычислений.

Программное обеспечение как сервис, SaaS – *бизнес-модель* продажи программного обеспечения, при которой поставщик разрабатывает *веб-приложение* и самостоятельно управляет им, предоставляя заказчикам *доступ* к программному обеспечению через *Интернет*.

Коммуникация как Сервис, CaaS – построенное в облаке коммуникационное решение для предприятия *MaaS*.

§6. Комплект средств разработки Windows Azure SDK

Цель данной лекции – ознакомиться с комплектом средств разработки *Windows* Azure SDK.

Windows Azure SDK предоставляет разработчикам *интерфейс* программирования приложений, необходимый для разработки, развертывания и управления масштабируемых сервисов в *Windows* Azure.

Azure *Cloud Fabric* и службы Azure *Storage* не поддерживают разработку или отладочные *операции* в облаке, поэтому Azure SDK позволяет делать это локально в виде приложений *Development Fabric* (DF) и *Development Storage* (DS), которые устанавливает *Windows* Azure *SDK*. Вместе с *SDK* также устанавливаются коллекция приложений примеров и библиотеки упакованных классов для облегчения программирования приложений.

Должны быть установлены *.NET* Framework 3.5 SP1 и *SQL* Express 2005 или 2008, также необходимо включить *ASP.NET* и WCF *HTTP*Activation для *IIS* 7.0 для *Windows Server* 2008, *Windows Vista* SP2, или *Windows* 7 RC или позже для установки и запуска *SDK*. Заметки к выпуску включают инструкции для настройки этих опций. Использование *SDK* не является обязательным, потому что есть возможность пользоваться любыми операционными системами и языками программирования, которые поддерживают *HTTP* запросы и ответы. Однако, вы увидите что использование *SDK* интерфейсов прикладного програм-

мирования *.NET* и библиотек для приложений и хранилищ позволяет наиболее просто работать с *HTTP* напрямую.

После того, как вы установили Azure *SDK*, Вы должны скачать и установить инструменты *Windows* Azure для *Visual Studio* для добавления шаблонов проектов *Web Cloud Sevice, Worker Cloud Service, Web and Worker Cloud Service Workflow Service*. Вы можете скачать текущую версию *Windows* Azure *SDK* и *Windows* Azure *Tools* для *Visual Studio* с главной страницы *Windows* Azure по ссылке www.microsoft.com/azure/windowsazure.mspx.

После установки *Windows* Azure *Tools* в *Visual Studio* появляются шаблоны *Cloud Service* в диалоге создания нового проекта. При выборе узла *Cloud Service* открываются *New Cloud Service*, который позволяет добавить *ASP.NET Web Roles*, *Worker Roles или CGI Web Roles* для нового проекта. *Windows* Azure *SDK* позволяет добавить более чем одну роль для каждого типа *Cloud Service*. Каждая роль использует отдельный экземпляр *Windows* Azure *CPU*, так минимальная *стоимость* запуска проекта в облаке будет примерно 4 * \$0.12 = \$0.48 в час.

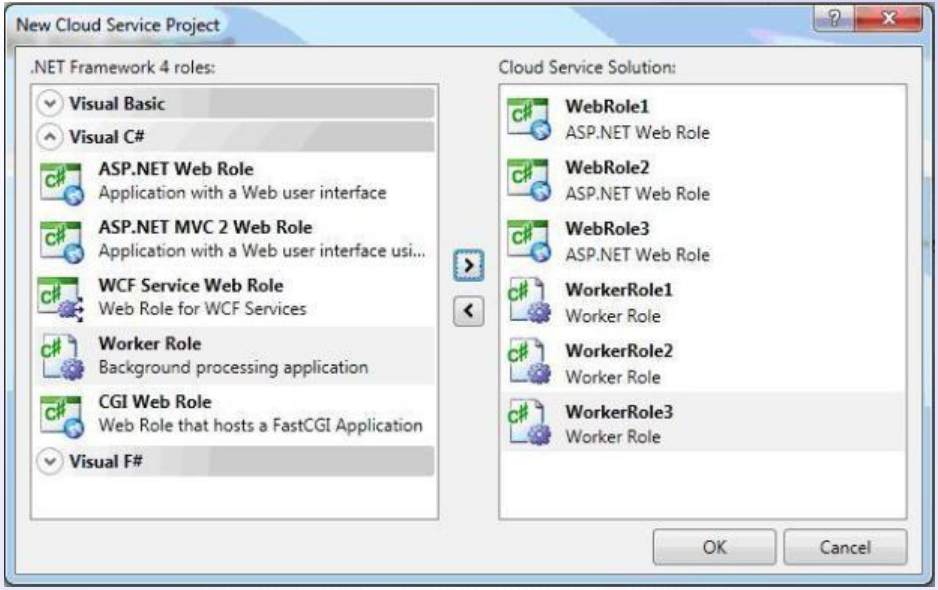


Рисунок 6.1: Создание нового проекта Cloud Service в Visual Studio

Добавив указанные роли и нажав OK, откроется новое решение с проектами *WebRole* и *Worker-Role* в *Solution Explorer* как показано на рисунке 6.2.

Узел *Roles* содержит элементы *WebRole*, которые указывают на каждую *WebRole*, которая обеспечивает пользовательский *интерфейс ASP.NET* для приложения и каждый *WorkerRole* для вычислительный операций, которые не требуют пользовательского интерфейса или используют страниц *ASP.NET WebRole* вместо этого.

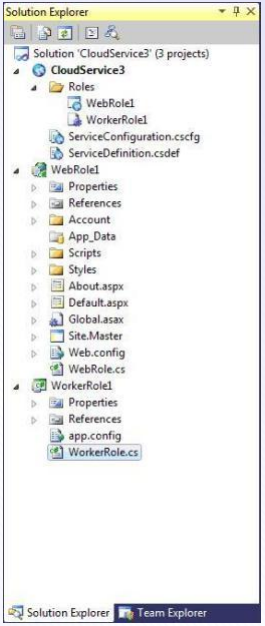


Рисунок 6.2: Solution Explorer Cloud Service

В зависимости от типа *Cloud Service* проекты включают *пространство* имен *Microsoft.ServiceHosting.ServiceRuntime*, которое содержит классы, указанные в таблице ниже.

Класс	Описание
RoleEntryPoint	Обеспечивает методы для управления инициализацией, запуском и остановкой методов сервиса, так же используется для мониторинга состояния сервиса.
RoleException	Сообщает об ошибках когда происходят недопустимые операции внутри роли.
RoleManager	Обеспечивает методы для журналирования сообщений и поступающих предупреждений, извлекает настройки конфигурации сервиса и возвращает местоположение ресурса.
RoleStatus	Информирует о текущем статусе роли: Healthy, NonExistent, Started, Starting, Stopped, Stopping или Unhealthy.

Проекты, которые используют *шаблон WebRole* определяют веб страницу *ASP.NET Default*.aspx как начальную точку для пользовательского интерфейса приложения в облаке.

Это сервис объединяет библиотеку класса *Common* из приложения-образца *HelloFabric* для содействия в журналировании проблем приложения. Журналы приложения – это практичные средства откладки приложений, запущенных в *Cloud Fabric*. Для чтения журналов, вы должны скопировать их в *Blob массив* использую *инструментарий* портала.

Образец проекта *StorageClient* включает библиотеку класса *StorageClient*, которая обеспечивает в объединении с библиотекой

.NETClient для сервиса данных *ADO.NET*, интерфейсный *класс Microsoft .NET* для *HTTP* операций над *Azure Blob, Queue и Table Storage* сервисами. Этот проект также включает консольное *приложение*, которые позволяет Вам тестировать возможности библиотеки. Консольное *приложение* C# запускается в *Development Fabric с Development Storage*.

При установке *Windows* Azure *SDK* не устанавливаются образцы приложений, которые включены в *Program Files\Microsoft Windows Azure SDK\v1.0\samples.zip*. Установите образцы, разархивировав *samples.zip* в директорию, где Вы имеете *права* на *запись*. В таблице ниже можно найти описание некоторых образцов приложений.

Для запуска примера *CloudDrive* необходим PowerShell.

Директория, в которую было извлечено содержимое архива *samples.zip* также содержит следующие три пакетных файла (cmd), которые можно запустить из командной строки:

- *buildall.cmd* строит все образцы проектов без использования Visual Studio.
- *createtables.cmd* вызывает *buildall.cmd* и создает базу данных и таблицы, необходимые для образцов, которые используют Table Storage.
- *rundevstore.cmd* вызывает *createtables.cmd* и запускает разработку хранилища, размещая его в базе данных, созданной *createtables.cmd*.

В состав *Development Fabric* входят следующие исполняемые файлы: *DFAgent.exe , DFLoadBalancer.exe, DFMonitor.exe и DFService.exe*, кото-

рые по умолчанию устанавливаются программой установки Azure *SDK* в каталог \Program Files\Windows Azure *SDK*\v1.0\bin\devfabric. После запуска *Development Fabric* в диспетчере задач вы можете увидеть эти четыре процесса. Сделать это можно выполнив:

- Выберите *Программы*|*Windows Azure SDK*|*Development Fabric* для запуска службы
  Development Fabric и его пользовательского интерфейса *DFUI.exe*.
- Правый щелчок мыши по значку Development Fabric в области уведомлений панели задач и выбрать запуск службы Development Fabric рисунок 6.3.
- Скомпилировать и запустить приложение Azure в Visual Studio.

Рисунок 6.3: Сообщения, отображаемые нажатии правой кнопкой мыши по значку Development Fabric в области уведомлений панели задач

Рисунок 6.4 показывает пользовательский *интерфейс* DFUI. Когда вы запускаете или останавливаете отладку, соответствующие приложения появляются или пропадают из пользовательского интерфейса DFUI.

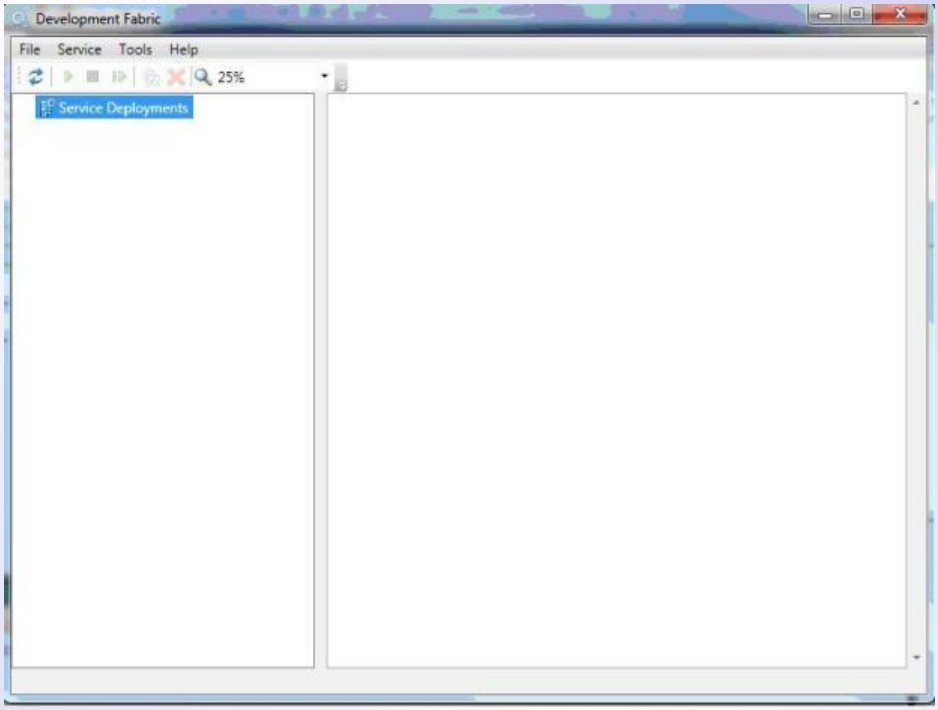


Рисунок 6.4: Пользовательский интерфейс приложения Development Fabric

Платформа *Windows* Azure поддерживает три типа масштабируемых хранилищ:

- Неструктурированные данные (blob)
- Структурированные данные (таблицы)
- Сообщение между приложениями и сервисами (очереди)

Запуская *rundevstore.exe* или собирая и запуская пользовательский код Azure в *Visual Studio*, запускаются все три сервиса, даже если Ваш проект требует только один сервис и отображается в пользовательском интерфейсе *Development Storage*.

Для защиты от потери данных, облако Azure хранит блобы, таблицы и очереди в *минимум* трех раздельных контейнерах в одном центре обработки данных. Инструмент геолокации Azure позволяет дублировать данные в нескольких центрах обработки данных *Microsoft* для уменьшения последствий восстановления после катастроф и для повышения производительности в специфичных географических регионах.

Приложение Azure, которые Вы запускаете в *Development* Framework, могут иметь *доступ* к локальным данным в *Development Storage* или к данным, загруженным в облако Azure. *Приложение* обращается к определенному порту и данным, расположенным в определенных местах в файле конфигурации проекта *ServiceConfiguration.cscfg*.

Файл конфигурации проекта Azure *ServiceDefinition.csdef* определяет стандартные точки входа и настройки конфигурации, которые хранятся в файле ServiceConfiguration.cscfg. В распечатке 5.1 показано содержимое файла ServiceDefinition.csdef по умолчанию, когда Вы создаете проект Azure, используя один из стандартных шаблонов *Windows* Azure *Tools* для *Visual Studio* (отмечены важные значения).

<ServiceDefinition name="SampleWebCloudService"
xmlns="http://schemas.microsoft.com/ServiceHosting/2008/10/

```
ServiceDefinition">
<WebRole name="WebRole">
<InputEndpoints>
<!– Must use port 80 for http and port 443 for https when running
in the cloud –>
<InputEndpoint name="HttpIn"protocol="http"port="80"/>
</InputEndpoints>
<ConfigurationSettings>
<Setting name="AccountName"/>
<Setting name="AccountSharedKey"/>
<Setting name="BlobStorageEndpoint"/>
<Setting name="QueueStorageEndpoint"/>
<Setting name="TableStorageEndpoint"/>
</ConfigurationSettings>
</WebRole>
</ServiceDefinition>
```

Листинг 5.1. Содержимое файла ServiceDefinition.csdef

Значение InputEndpoint применяется только для хранилищ в облаке.

Распечатка 5.2 показывает содержимое файла *ServiceConfiguration.cscfg* для веб приложения *SampleWebCloudService* с конфигураций по умолчанию для *Development Storage* (выделено):

```
<?xml version="1.0"?>
```

```
<ServiceConfiguration serviceName="SampleWebCloudService"
xmlns="http://schemas.microsoft.com/ServiceHosting/2008/10/
ServiceConfiguration">
<Role name="WebRole">
<Instances count="1"/>
<ConfigurationSettings> <Setting name="AccountName"
value="devstoreaccount1"/>
<Setting name="AccountSharedKey"
value="Eby8vdM02xNOcqFlqUwJPLlmEtlCDXJ1OUzFT50uSRZ6IF
suFq2UVErCz4I6tq/K1SZFPTOtr/KBHBeksoGMGw==/>
<Setting name="BlobStorageEndpoint"
value="http://127.0.0.1:10000/"/>
<Setting name="QueueStorageEndpoint"
value="http://127.0.0.1:10001/"/>
<Setting name="TableStorageEndpoint"
value="http://127.0.0.1:10002/"/>
<!-<Setting name="AccountName"value="oakleaf"/>
<Setting name="AccountSharedKey"
value="3elV1ndd . . . Coc0AMQA==/>
<Setting name="BlobStorageEndpoint"
value="http://blob.core.windows.net"/>
<Setting name="QueueStorageEndpoint"
value="http://queue.core.windows.net"/>
```

```
<Setting name="TableStorageEndpoint"
value="http://table.core.windows.net"/>
</ConfigurationSettings>
</Role>
</ServiceConfiguration>
```

Листинг 5.2. Содержимое файла ServiceConfiguration.cscfg

Описания элементов файла конфигурации ServiceConfiguration.csfg:

- Instances count – количество экземпляров вашего приложения, которое будет создано в облаке, когда вы развернете его.
- AccountName – имя, ассоциированное с Вашим *Hosted Service*, с которым вы создавали учетную запись, для Development Storage это devstoreaccount1.
- AccountSharedKey шифрует несколько элементов в HTTP запросе.
- BlobStorageEndpoint – это публичный постоянный Universal Resource Identifier (URI). Для Developer Storage это адрес интерфейса компьютера loopback (localhost = 127.0.0.1) с TCP портом по умолчанию 10000.
- QueueStorageEndpoint для хранилища в облаке это публичный постоянный URI. Для Developer Storage это адрес интерфейса компьютера loopback с TCP портом по умолчанию 10001.

- TableStorageEndpoint публичный постоянный Universal Resource Identifier (URI). Для Developer Storage это адрес интерфейса компьютера loopback с TCP портом по умолчанию 10002.

Вы можете настроить собственные номера *TCP* портов если при использовании значений по умолчанию возникает конфликт с текущей конфигурацией.

Краткие итоги:

В данной лекции мы получили первоначальные сведения о работе с *Windows* Azure *SDK*. Рассмотрели процедуру создание *Cloud Service*, пользовательский *интерфейс Development Fabric*.

§7. Архитектура Windows Azure Platform

Цель данной лекции – получить *представление* об архитектуре linebreak *Windows* Azure.

Архитектура Windows Azure Platform

Платформа Windows Azure – это модель Платформа как Сервис, которая предполагает *запуск* приложений на серверах и связанной сетевой инфраструктуре, размещенной в центрах обработки данных *Microsoft* и имеющей *доступ в Интернет*. Платформа состоит из мастшабируемой "облачной"операционной системы, фабрики хранения данных и связанных сервисов доставки через физические или логические (*виртуализация*) экземпляры *Windows Server 2008*. Комплект средств разработки *Windows* Azure (*SDK*) обеспечивает разработку версии "облачных сервисов также хорошо, как инструменты и интерфейсы прикладного программирования (*API*), необходимые для разработки, разворачивания и управления масштабируемых сервисов в *Windows* Azure, включая шаблоны приложений Azure для *Visual Studio* 2008 и 2010. На рисунке 7.1 изображены компоненты основа "облачной"платформы и компоненты разработчика.

Согласно *Microsoft*, при использовании Azure Вы получаете:

- Адаптация существующих приложений для работы с веб сервисами;
- Построение, изменение и распределение приложений в Сети с минимальными локальными ресурсами;

- Выполняют услуги, таких как хранение больших объемов данных, пакетная обработка данных, вычисления больших объемов данных, и так далее;
- Создание, тестирование, отладка и распределение веб сервисов быстро и недорого;
- Снижение стоимости и рисков построения и распространения местных ресурсов;
- Снижение затрат и усилий на ИТ управление;

Экономическая обстановка *Microsoft* во время выпуска платформы Azure акцентировала внимание на снижении затрат, что является основным поводом для использования Azure мелкими, средними и крупными ИТ отделами.

Microsoft разработали платформу Azure, позволив *.NET* разработчикам усилить их *опыт* создания в *Visual Studio* 2008 (и выше) *ASP.NET* веб приложений и *Windows Communication* Framework (WCF) сервисов. Проекты веб приложений запускаются в изолированной версии *Internet Information* Services (*IIS*) 7. Веб приложения и веб сервисы запускаются в частично доверенном механизме защиты, позволяющем ограничивать *доступ* коду к ресурсам компьютера (Code Access *Security*), который приблизительно соответсвует среднему уровню доверия *ASP.NET* и ограничивает *доступ* к некоторым ресурсам операционной системы.

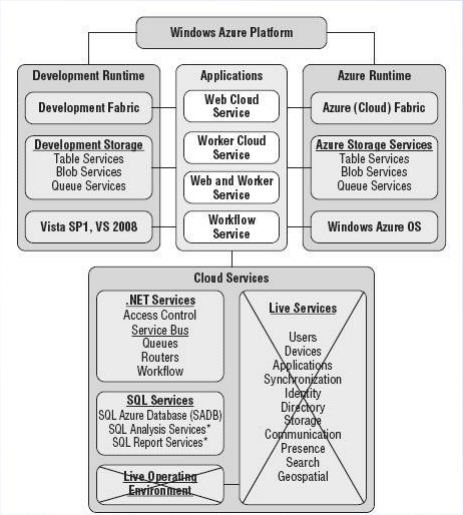


Рисунок 7.1: Компоненты платформы Windows Azure и Комплекта средств разработки

Комплект средств разработки *Windows* Azure (Март 2009) позволяет использоваться полный *доступ* к ресурсам компьютера для запуска не *.NET* кода, использования *.NET* библиотек, которые требуют полного доверия и процесс обработки взаимодействия, используя программные каналы (*Pipe*). *Microsoft* обещает поддержку запуска программного кода *Ruby, PHP* и *Python* в "облачной"платформе. Исходный вариант платформы разработки был ограничен средой программирования *Visual Studio* 2008 и выше с планом на поддержку инструментов Eclipse. Платформа *Windows* Azure поддерживает веб стандарты и протоколы вклю-

чая *SOAP, HTTP, XML, Atom* и AtomPub.

Исходной точкой входа для разработчиков Azure для размещения *ASP.NET* приложений в облако является портал *Windows* Azure по адресу https://windows.azure.com/Cloud/Provisioning/Default.aspx. Портал требует входа с использованием *Windows Live ID*. Предварительная версия Azure (Community *Technical* Previews , *CTPs*) требует разных токенов для:

Windows Azure, которая включает:

- Azure Hosted service;
- Storage Accounts;

SQL Azure *Live* службы, которые включают:

- Live Framework: Предварительная версия;
- Live Services: существующие интерфейсы прикладного программирования.

Live сервисы: существующие интерфейсы прикладного программирования не являются частью предварительной версии и не требуют токена. С начала 2009 года *Windows* Azure *токен* дает право на одну учетную *запись* Hosted *Service*, две учетные записи для *Storage*. Вы запрашиваете токены Azure через страницу *Microsoft Connect*, на которую можно попасть со страницы портала.

Страница учетной записи содержит ссылки на страницы настроек и управления *SQL* Azure, сервисами *.NET* и *Live* сервисами. Эта страница также позволяет вам получать Azure и *Live* Framework *CTP* токены. Страница *Live* Alerts позволяет настроить, как и когда получать сообщения, содержащие критические сигналы приложения, информационные бюллетени и обновления портала.

Предварительная версия марта 2009 года включает геолокацию, которая позволяет владельцам учетных записей выбрать центры обработки данных для расположения *Hosted Services* и *Storage* Accounts. Например, USA-Northwest (Quincy, WA) и USA-Southeast (*San*Antonio, TX.). Вы можете добавить наборы *Hosted Services* и *Storage* Accounts в группу, чтобы гарантировать, что сервисы и хранилище располагаются в одном и том же центре обработки данных, для того, чтобы увеличить *производительность*.

Вы можете вставить *GUID* полученные в сообщении от члена команды Azure в текстовое окно *Resource Token ID* и нажать Claim *Token* для добавление еще одной или больше точки входа для подходящего объекта или нескольких объектов в *список*.

Если вы хотите разрабатывать веб сайты с поддержкой *Live* Framework или Mesh (программный комплекс для синхронизации данных в кроссплатформенных средах разработанный компанией *Microsoft*) веб приложений, необходимо запросить *токен Live* Framework по *email* meshctpe@*microsoft.com*. После того, как вы получите *Live* Framework

токен, вы можете скачать и установить текущие версии *Live* Framework *SDK* и *Live* Framework *Tools* для *Visual Studio* по ссылкам указанным на странице http://dev.live.com/liveframework/sdk/. Вы должны оплатить *Live* Framework *токен* для того, чтобы скачать *Live SDK* и дополнительные инструменты. Нет необходимости использовать учетную *запись Windows* Azure для тестирования Azure *Hosted Services and Storage* Services, потому что платформа разработки Azure эмулирует "облачные"сервисы Azure на вашем компьютере.

7.1 Windows Azure Storage

Хранилище Windows Azure Storage обеспечивает разработчикам возможность хранения данных в облаке. *Приложение* может выполнять *доступ* к своим данным в любой момент времени из любой точки планеты, хранить любой объем данных и как угодно долго. При этом данные гарантированно не будут повреждены и утеряны. *Windows* Azure *Storage* предлагает богатый набор абстракций данных:

- Windows Azure Table – обеспечивает структурированное хранилище состояний сервиса.
- Windows Azure Blob – обеспечивает хранилище больших элементов данных.
- Windows Azure Queue – обеспечивает диспетчеризацию асинхрон-

ных заданий для реализации обмена данными между сервисами.

7.2 Azure Table Services

Windows Azure Table – структурированное хранилище, которое поддерживает высокомасштабируемые таблицы в облаке, которые могут содержать миллиарды сущностей и терабайты данных. По мере увеличения трафика, система будет эффективно масштабироваться, автоматически подключая тысячи серверов. Структурированное хранилище реализовано в виде таблиц (Tables), в которых располагаются сущности (Entities), содержащие ряд именованных свойств (Properties). Вот некоторые из основных характеристик Windows Azure Table:

- Поддержка LINQ, ADO .NET Data Services и REST.
- Контроль типов во время компиляции при использовании клиентской библиотеки ADO .NET Data Services.
- Богатый набор типов данных для значений свойств.
- Поддержка неограниченного количества таблиц и сущностей без ограничения размеров таблиц.
- Поддержка целостности для каждой сущности.
- Нежесткая блокировка при обновлениях и удалениях.

- Для запросов, выполнение которых требует длительного периода времени, или запросов, прерванных по завершению времени ожидания, возвращаются частичные результаты и маркер продолжения.

Рассмотрим модель данных таблицы Windows Azure Table:

- **Учетная запись хранилища (Storage Account)** – для доступа к Windows Azure Storage приложение должно использовать действительную учетную запись. Новую учетную запись можно создать через веб-интерфейс портала Windows Azure. Как только учетная запись создана, пользователь получает 256-разрядный секретный ключ, который впоследствии используется для аутентификации запросов этого пользователя к системе хранения. В частности, с помощью этого секретного ключа создается подпись HMAC SHA256 для запроса. Эта подпись передается с каждым запросом данного пользователя для обеспечения аутентификации. Имя учетной записи входит в состав имени хоста в URL. Для доступа к таблицам используется следующий формат имени хоста: <имяУчетнойЗаписи>.table.core.windows.net.
- **Таблица (Table)** – содержит набор сущностей. Область действия имен таблиц ограничена учетной записью. Приложение может создавать множество таблиц в рамках учетной записи хранилища.
- **Сущность (строка) (Entity (Row))** – Сущности (сущность является аналогом "строки") – это основные элементы данных, хра-

нящиеся в таблице. Сущность включает набор свойств. В каждой таблице имеется два свойства, которые образуют уникальный ключ для сущности.

- **Свойство (столбец) (Property (Column))** – Представляет отдельное значение сущности. Имена свойств чувствительны к регистру. Для значений свойств поддерживается богатый набор типов.
- **Ключ секции (PartitionKey)** – Первое свойство ключа каждой таблицы. Эта система использует данный ключ для автоматического распределения сущностей таблицы по множеству узлов хранения.
- **Ключ строки (RowKey)** – Второе свойство ключа таблицы. Это уникальный ID сущности в рамках секции. PartitionKey в сочетании с RowKey уникально идентифицирует сущность в таблице.
- **Временная метка (Timestamp)** – Каждая сущность имеет версию, сохраняемую системой.
- **Секция (Partition)** – Набор сущностей в таблице с одинаковым значением ключа секции.
- **Порядок сортировки (Sort Order)** – Для *CTP*-версии предоставляется всего один индекс, в котором все сущности сортированы по PartitionKey и затем по RowKey. Это означает, что запросы с указанием этих ключей будут более эффективными, и все возвращаемые результаты будут сортированы по PartitionKey.

Таблица имеет гибкую схему. Windows Azure Table отслеживает имя и типизированное значение каждого свойства каждой сущности. Приложение может моделировать фиксированную схему на стороне клиента, обеспечивая одинаковый набор свойств для всех создаваемых сущностей.

Рассмотрим некоторые дополнительные сведения о сущностях:

- Сущность может иметь до 255 свойств, включая обязательные системные свойства: PartitionKey, RowKey и Timestamp. Имена всех остальных свойств сущностей определяются приложением.
- Свойства PartitionKey и RowKey строкового типа.
- Свойство Timestamp является доступным только для чтения обслуживаемым системой свойством, которое должно рассматриваться как непрозрачное свойство.
- Отсутствие фиксированной схемы – Windows Azure Table не сохраняет никакой схемы, поэтому все свойства хранятся как пары <имя, типизированное значение>. Это означает, что свойства сущностей одной таблицы могут сильно отличаться. В таблице даже может быть две сущности, свойства которых имеют одинаковые имена, но разные типы значений.
- Суммарный объем всех данных сущности не может превышать 1 МБ. Сюда входит размер имен свойств, а также размер значений

свойств или их типов, включая и два обязательных свойства ключей (PartitionKey и RowKey).

- Поддерживаются типы Binary, Bool, DateTime, Double, GUID, Int, Int64, String. Ограничения представлены в таблице ниже.

Таблица 3.1

Тип свойства	**Описание**
Binary	Массив байтов размером до 64 КБ.
Bool	значение.
DateTime	64-разрядное значение, представляющее время в формате UTC. Поддерживаемый диапазон значений: от 1/1/1600 до 12/31/9999.
Double	64-разрядное значение с плавающей точкой.
GUID	128-разрядный глобально уникальный идентификатор.
Int	32-разрядное целое значение.
Int64	64-разрядное целое значение.
String	16-разрядное UTF-кодированное значение. Размер строковых значений может быть до 64 КБ.

Windows Azure Table обеспечивает возможность масштабирования таблиц до тысяч узлов хранения через распределение сущностей в таб-

лице. При распределении сущностей желательно обеспечить, чтобы сущности, входящие в одно множество, располагались в одном узле хранения. Приложение формирует эти множества соответственно значениям свойства PartitionKey сущностей.

Приложениям должна быть известна рабочая нагрузка каждой отдельно взятой секции. Для обеспечения желаемых результатов тестирование должно моделировать максимальную рабочую нагрузку.

Partition Key Document Name	Row Key Version	Property 3 Modification Time	.....	Property N Description	
Examples Doc	V1.0	8/2/2007	.....	Committed version	Partition 1
Examples Doc	V2.0.1	9/28/2007		Alice's working version	
FAQ Doc	V1.0	5/2/2007		Committed version	
FAQ Doc	V1.0.1	7/6/2007		Alice's working version	Partition 2
FAQ Doc	V1.0.2	8/1/2007		Sally's working version	

Рисунок 7.2: Примеры секций

На рисунке выше представлена таблица, содержащая множество версий документов. Каждая сущность данной таблицы соответствует определенной версии определенного документа. В этом примере ключом *секции таблицы* является имя документа, и ключом строки – номер версии. Имя документа и версия уникально идентифицируют каждую сущность

таблицы. В данном примере секцию образуют все версии одного документа.

Хорошая масштабируемость системы хранения достигается за счет распределения секций по множеству узлов хранения.

Система отслеживает характер использования секций и автоматически равномерно распределяет эти секции по всем узлам хранения. Это позволяет системе и приложению масштабироваться соответственно количеству запросов к таблице. То есть если некоторые секции запрашиваются больше других, система автоматически разнесет их на несколько узлов хранения, таким образом, распределяя трафик между множеством серверов. Однако секция, т.е. все сущности, имеющие одинаковый ключ секции, будут обслуживаться как один узел. Но даже несмотря на это, объем данных в рамках секции не ограничен емкостью хранилища одного узла хранения.

Сущности одной секции хранятся вместе. Это обеспечивает наиболее эффективную обработку запросов к секции. Более того, в этом случае приложение может использовать все преимущества эффективного кэширования и других оптимизаций производительности, обеспечиваемых расположением данных в секции.

В примере выше секцию образуют все версии одного документа. Таким образом, для извлечения всех версий данного документа необходимо выполнить доступ всего к одной секции. Чтобы получить все версии документов, измененные до 5/30/2007, придется запрашивать несколько

секций, что будет не так эффективно и более ресурсоемко, поскольку по запросу должны будут проверяться все секции, которые к тому же могут располагаться на разных узлах хранения.

Выбор ключа секции важен с точки зрения обеспечения эффективного масштабирования приложения. При этом необходимо найти компромисс между размещением сущностей в одной секции, что обеспечивает большую эффективность запросов, и масштабируемостью таблицы, поскольку, чем больше секций в таблице, тем проще для Windows Azure Table распределить нагрузку между множеством серверов.

Для наиболее частых и критичных по времени ожидания запросов PartitionKey должен быть включен как часть выражения запроса. Запрос, в котором указан PartitionKey, будет намного эффективнее, поскольку в этом случае просматриваются сущности только одной секции. Если при выполнении запроса PartitionKey не указан, в поисках необходимых сущностей просматриваются все *секции таблицы*, что значительно снижает эффективность.

Далее представлены некоторые советы и рекомендации по выбору PartitionKey для таблицы:

1. Прежде всего, выявите важные свойства таблицы. Это свойства, используемые в условиях запросов.

2. Из этих важных свойств выберите *потенциальные ключи.*

3. Из преобладающего запроса выберите свойства, используемые в условиях. Важно понять, какой запрос будет преобладающим для при-

ложения.

4. Это будет исходный набор свойств ключей.

5. Расставьте свойства ключей в порядке их значимости в запросе.

6. Проверьте, обеспечивают ли свойства ключей уникальную идентификацию сущности? Если нет, включите в набор ключей уникальный идентификатор.

7. Если имеется только одно свойство ключа, используйте его в качестве PartitionKey.

8. Если имеется только два свойства ключей, первое используйте как ParitionKey и второе – как RowKey.

9. При наличии более двух свойств ключей можно попытаться распределить их в две группы: первая группа будет PartitionKey, и вторая – RowKey. что PartitionKey, например, состоит из двух ключей, разделенных '-'.

Теперь, когда приложение имеет набор *потенциальных ключей*, необходимо убедиться, что выбранная *схема секционирования* является масштабируемой:

1. Исходя из статистических данных интенсивности использования приложения, определите, не приведет ли секционирование по выбранному выше PartitionKey к созданию слишком загруженных секций, которые не смогут эффективно обслуживаться одним сервером? Проверить это можно, применив *нагрузочное тестирование секции таблицы*. Для этого в тестовой таблице создается секция соответственно выбран-

ным ключам. Она подвергается пиковой нагрузке, полученной исходя из предполагаемых полезной нагрузки и запросов. Это позволяет проверить, может ли *секция таблицы* обеспечить необходимую производительность приложения.

2. Если *секция таблицы* проходит *нагрузочное тестирование*, ключи выбраны правильно.

3. Если *секция таблицы* не проходит *нагрузочного тестирования*, найдите ключ секции, который обеспечил бы более узкое подразделение сущностей. Это можно сделать через объединение выбранного ключа секции со следующим свойством ключа, или выбрав в качестве ключа секции другое важное свойство. Целью этой операции должно быть создание большего количества секций, чтобы не возникало одной слишком большой или слишком загруженной секции.

4. Система спроектирована так, что обеспечивает необходимое масштабирование и обработку большого количества запросов. Но при чрезвычайно высокой интенсивности запросов ей приходится выполнять балансировку нагрузки, в результате чего некоторые из запросов могут завершаться ошибкой превышения времени ожидания. Сократить или устранить ошибки такого рода может снижение интенсивности запросов. Вообще говоря, такие ошибки возникают редко; однако если вы столкнулись с частыми или неожиданными ошибками превышения времени ожидания, свяжитесь с нами через сайт MSDN, мы обсудим, как оптимизировать использование Windows Azure Table и предотвратить

возникновение таких ошибок в вашем приложении.

Также можно проанализировать расширяемость выбранных ключей, особенно если на момент их выбора нет точных сведений о характеристиках пользовательского трафика. В этом случае важно выбирать ключи, которые можно легко расширять для обеспечения более тонкого секционирования. Далее в данном документе приводится подробный пример этого.

Для таблиц и сущностей поддерживаются следующие базовые операции:

- Создание таблицы или сущности.
- Извлечение таблицы или сущности с применением фильтров.
- Обновление сущности (но не таблицы).
- Удаление таблицы или сущности.

Для работы с таблицами в .NET-приложении можно просто использовать ADO.NET Data Services.

В следующей таблице приведен список предлагаемых API. Поскольку применение ADO.NET Data Services в итоге сводится к передаче REST-пакетов, приложения могут использовать REST напрямую. Кроме того, что REST обеспечивает возможность доступа к хранилищу посредством не-.NET языков, он также позволяет реализовывать более тонкое управление сериализацией/десериализацией сущностей, что пригодится при

работе с такими сценариями, как наличие разных типов сущностей или более чем 255 свойств в таблице и т.д.

7.3 Пример

В приведенных ниже примерах описываются операции с таблицей "Blogs". В этой таблице хранятся блоги для приложения MicroBlogging.

В приложении MicroBlogging есть две таблицы: Channels (Каналы) и Blogs (Блоги). Имеется список каналов, блоги публикуются в определенном канале. Пользователи подписываются на каналы и ежедневно получают новые блоги этих каналов.

В данном примере рассмотрим только таблицу Blogs и приведем примеры следующих операций с ней:

1. Описание схемы таблицы.
2. Создание таблицы.
3. Вставка блога в таблицу.
4. Получение списка блогов из таблицы.
5. Обновление блога в таблице.
6. Удаление блога из таблицы.

Схема таблицы описывается как C#-класс. Такую модель использует ADO.NET Data Services. Схема известна только клиентскому приложению и упрощает доступ к данным. Сервер схему не применяет.

Рассмотрим описание сущностей Blog, хранящихся в таблице Blogs.

Каждая сущность блога содержит следующие данные:

1. Имя канала (ChannelName) – канал, в котором размещается блог.
2. Дата размещения.
3. Текст (Text) – содержимое тела блога.
4. Рейтинг (Rating) – популярность этого блога.

Во-первых, обратите внимание, что для таблицы определен PartitionKey, представляющий имя канала, частью которого является блог, и в качестве RowKey используется дата размещения блога. PartitionKey и RowKey – ключи таблицы Blogs, они объявляются посредством атрибута класса DataServiceKey (Ключ сервиса данных). То есть таблица Blogs секционирована по именам каналов (ChannelName). Это позволяет приложению эффективно извлекать самые недавние блоги канала, на который подписан пользователь. Кроме ключей, в качестве свойств объявлены характерные для пользователя атрибуты. Все свойства имеют открытые (public) методы считывания и присвоения значения и хранятся в таблице Windows Azure Table. Итак, в примере ниже:

- Text и Rating хранятся для экземпляра сущности в таблице Azure.
- RatingAsString нет, потому что для него не определен метод присвоения значения.
- Id не хранится, потому что методы доступа не public.

```
[DataServiceKey("PartitionKey "RowKey")]
public class Blog
```

```
{
// ChannelName
public string PartitionKey { get; set; }
// PostedDate
public string RowKey { get; set; }
// Определяемые пользователем свойства
public string Text { get; set; }
public int Rating { get; set; }
public string RatingAsString { get; }
protected string Id { get; set; }
}
```

Далее рассмотрим, как создать таблицу Blogs для учетной записи хранилища. Создание таблицы аналогично созданию сущности в основной таблице "Tables". Эта основная таблица определена для каждой учетной записи хранилища, и имя каждой таблицы, используемой учетной записью хранения, должно быть зарегистрировано в основной таблице. Описание класса основной таблицы приведено ниже, где свойство TableName (Имя таблицы) представляет имя создаваемой таблицы.

```
[DataServiceKey("TableName")]
public class TableStorageTable
{
public string TableName { get; set; }
}
```

Фактическое создание таблицы происходит следующим образом:

```
// Uri сервиса: "http://<Account>.table.core.windows.net/"
DataServiceContext context = new DataServiceContext(serviceUri);
TableStorageTable table = new TableStorageTable("Blogs");
// Создаем новую таблицу, добавляя новую сущность
// в основную таблицу "Tables"
context.AddObject("Tables", table);
// результатом вызова SaveChanges является отклик сервера
DataServiceResponse response = context.SaveChanges();
```

serviceUri – это uri сервиса таблицы, http://<Здесь указывается имя учетной записи>.table.core.windows.net/. DataServiceContext (Контекст сервиса данных) – один из основных классов сервиса данных ADO.NET, представляющий контекст времени выполнения для сервиса. Он обеспечивает API для вставки, обновления, удаления и запроса сущностей с помощью либо LINQ, либо RESTful URI и сохраняет состояние на стороне клиента.

Рассмотрим вставку элемента Blog. Чтобы вставить сущность, приложение должно выполнить следующее.

1. Создать новый C#-объект и задать все свойства.

2. Создать экземпляр DataServiceContext, который представляет подключение к серверу в сервисе данных ADO .NET для вашей учетной записи хранилища.

3. Добавить C#-объект в контекст.

4. Вызвать метод SaveChanges (Сохранить изменения) объекта DataServiceContext для отправки запроса серверу. Это обеспечивает отправку на сервер HTTP-запроса с сущностью в XML-формате ATOM.

Далее представлены примеры кода для перечисленных выше операций:

```
Blog blog = new Blog {
PartitionKey = "Channel9", // ChannelName
RowKey = DateTime.UtcNow.ToString(), // PostedDate
Text = "Hello",
Rating = 3
};
serviceUri = new Uri("http://<account>.table.core.windows.net");
var context = new DataServiceContext(serviceUri);
context.AddObject("Blogs blog); DataServiceContext response = context.SaveChanges();
```

Запрос сущностей выполняется с помощью встроенного в C# языка запросов LINQ (Language Integrated Query). В данном примере извлечем все блоги, рейтинг которых равен 3.

При обработке запроса (например, с помощью выражение foreach), он передается на сервер. Сервер отправляет результаты в XML-формате ATOM. Клиентская библиотека ADO .NET Data Services десериализует результаты в C#-объекты, после чего они могут использоваться приложением.

```
var serviceUri = new Uri("http://<account>.table.core.windows.net");
DataServiceContext context = new DataServiceContext(serviceUri);
// LINQ-запрос с использованием DataServiceContext для выбора
// из таблицы Blogs всех сущностей блогов, для которых rating = 3
var blogs =
from blog in context.CreateQuery<blog>("Blogs")
where blogs.Rating == 3
select blog;
// запрос отправляется на сервер и выполняется
foreach (Blog blog in blogs) { }
```

Обновление сущности выполняется следующим образом.

1. Создается DataContext (Контекст данных), свойству MergeOption (Вариант объединения) которого задается значение OverwriteChanges (Перезапись изменений) или PreserveChanges (Сохранение изменений), как описывается в разделе 4.8. Это обеспечивает правильную обработку ETag для каждого извлекаемого объекта.

2. С помощью LINQ DataContext получает сущность, которая будет обновляться. Извлечение ее с сервера гарантирует обновление ETag в сущностях, отслеживаемых контекстом, и то, что при последующих обновлениях и удалениях в заголовке if-match будет использоваться обновленный ETag. Меняем C#-объект, представляющий сущность.

3. Возвращаем C#-объект в тот же DataContext для обновления. Использование того же DataContext гарантирует автоматическое повтор-

ное использование ETag, полученного ранее для этого объекта.

4. Вызываем метод SaveChanges для отправки запроса на сервер.

```
Blog blog =
(from blog in context.CreateQuery<blog>("Blogs")
where blog.PartitionKey == "Channel9"
&& blog.RowKey == "Oct-29"
select blog).FirstOrDefault();
blog.Text = "Hi there";
context.UpdateObject(blog);
DataServiceResponse response = context.SaveChanges();
```

Удаление Blog

Удаление сущности аналогично ее обновлению. Для этого извлекаем сущность с помощью DataServiceContext и вызываем для содержимого вместо метода UpdateObject метод DeleteObject (Удалить объект).

```
// Получаем объект Blog для ("Channel9 "Oct-29")
context.DeleteObject(blog);
DataServiceResponse response = context.SaveChanges();
```

Рассмотрим рекомендации по работе с DataServiceContext:

- Объект DataServiceContext не обеспечивает безопасность потоков, поэтому он не может использоваться совместно разными потоками, а также имеет непродолжительное время существования.
- DataServiceContext не является объектом с длительным временем

жизни. Вместо того, чтобы использовать один DataServiceContext в течение всей жизни потока, рекомендуется создавать объект DataServiceContext каждый раз, когда возникает необходимость выполнить ряд транзакций с WindowsAzureTable, и затем удалять этот объект.

- Если для всех вставок/обновлений/удалений используется один экземпляр DataServiceContext и возникает сбой при выполнении SaveChanges, сведения об операции, давшей сбой, сохраняются в DataServiceContext. При последующем вызове SaveChanges попытка выполнить эту операцию повторяется.
- DataServiceContext имеет свойство MergeOption, которое используется для управления тем, как DataServiceContext обрабатывает отслеживаемые сущности. Возможные значения:
  - AppendOnly (Только добавление): Это значение по умолчанию, при использовании которого DataServiceContext не загружает экземпляр сущности с сервера, если он уже имеется в его кэше.
  - OverwriteChanges: DataServiceContext всегда загружает экземпляр сущности с сервера и перезаписывает предыдущий вариант сущности, т.е. обеспечивает соответствие экземпляра сущности ее текущему состоянию.
  - PreserveChanges: Если экземпляр сущности существует в DataServiceContext, он не загружается из постоянного храни-

лица. Все изменения, вносимые в свойства объектов в DataServiceContext, сохраняются, но ETag обновляется, поэтому данную опцию следует использовать при необходимости восстановления после ошибок совместного доступа с нежесткой блокировкой.

– NoTracking (Без отслеживания): DataServiceContext не отслеживает экземпляры сущностей. Обновление сущности в контексте без отслеживания реализуется с помощью Etag, который обновляется посредством AttachTo. Этот вариант не рекомендуется к применению.

```
context.AttachTo("Blogs", blog, "etag to use");
context.UpdateObject(blog);
context.SaveChanges();
```

Когда MergeOption контекста задано значение AppendOnly и объект DataServiceContext уже отслеживает сущность в результате предыдущей операции извлечения или добавления, повторное извлечение сущности с сервера не приведет к обновлению отслеживаемой сущности в контексте. Таким образом, если сущность на сервере была изменена, последующие обновления/удаления приведут к сбою необходимых условий (PreCondition). В примере кода раздела 5 MergeOption задано значение PreserveChanges, которое обеспечивает, что сущность будет загружаться с сервера всегда.

Результатом всех рассматриваемых выше операций является передача HTTP-сообщений на и с сервера. Приложение может отказаться от использования клиентской библиотеки .NET и работать на уровне HTTP/REST.

Рассмотрим параллельные обновления. Для обновления сущности необходимо выполнить следующие операции.

1. Получить сущность с сервера.

2. Обновить объект локально и вернуть его на сервер.

Предположим, два процесса, выполняющихся параллельно, пытаются обновить одну и ту же сущность. Поскольку шаги 1 и 2 не являются неделимыми, на любом из них может возникнуть ситуация внесения изменений в уже устаревшую версию сущности. Для решения этой проблемы Windows Azure Table использует нежесткую блокировку.

1. Для каждой сущности система сохраняет версию, которая изменяется сервером при каждом обновлении.

2. При извлечении сущности, сервер отправляет эту версию клиенту в виде ETag HTTP.

3. Когда клиент передает запрос UPDATE (обновить) на сервер, он отправляет на него этот ETag в виде заголовка If-Match.

4. Если версия сущности, хранящаяся на сервере, аналогична ETag в заголовке If-Match, изменение принимается, и хранящаяся на сервере сущность получает новую версию. Эта новая версия возвращается клиенту как заголовок ETag.

5. Если версия сущности на сервере отличается от ETag в заголовке If-Match, изменение отклоняется, и клиенту возвращается HTTP-ошибка 'precondition failed' (необходимое условие не выполнено).

При получении ошибки 'precondition failed' типовым поведением клиентского приложения будет повторение всей операции, как показано в фрагменте кода ниже.

1. Приложение должно извлечь этот объект снова, т.е. получить его последнюю версию.

2. Обновить объект локально и вернуть его на сервер.

При использовании клиентской библиотеки .NET приложение получает HTTP-код ошибки в виде исключения DataServiceRequestException.

В примере ниже два разных клиента выполняют один и тот же код для изменения текста. Эти два клиента пытаются задать Text разные значения.

1. Они извлекают сущность. При этом для каждой сущности извлекается ETag, например, 'v1'. Оба клиента полагают, что предыдущая версия сущности – 'v1'.

2. Каждый клиент локально обновляет свойство Text.

3. Каждый клиент вызывает методы UpdateObject и SaveChanges.

4. Каждый клиент отправляет на сервер HTTP-запрос с заголовком 'If-Match: v1'.

5. Запрос одного из клиентов попадает на сервер первым.

- Сервер сравнивает заголовок If-Match с версией сущности. Они совпадают.
- Сервер применяет изменение.
- Версия сущности на сервере обновляется и становится 'v2'.
- В качестве ответа клиенту отправляется новый заголовок 'ETag:v2'.

6. Далее на сервер поступает запрос другого клиента. На этот момент изменения первого клиента уже применены.

- Сервер сравнивает заголовок If-Match с версией сущности. Они не совпадают, поскольку версия сущности уже изменена на 'v2', тогда как в запросе указывается версия 'v1'.
- Сервер отклоняет запрос.

```
// Задаем такой вариант объединения, который обеспечивает
// сохранение обновлений, но позволяет обновление etag.
// По умолчанию применяется значение AppendOnly, при котором
// уже отслеживаемая сущность не перезаписывается значениями,
// полученными с сервера, в результате чего в случае изменения
// сущности на сервере используется недействительный etag.
context.MergeOption = MergeOption.PreserveChanges;
Blog blog =
(from blog in context.CreateQuery<blog>(& quot;Blogs& quot;)
where blog.PartitionKey == & quot;Channel9& quot;
```

```
&& blog.RowKey == & quot;Oct-29 & quot;
select blog).FirstOrDefault();
blog.Text = & quot;Hi there again& quot;;
try
{
context.UpdateObject(blog);
DataServiceResponse response = context.SaveChanges();
}
catch (DataServiceRequestException e)
{
OperationResponse response = e.Response.First();
if (response.StatusCode == (int)HttpStatusCode.PreconditionFailed)
{
// выполняем запрос объекта повторно, чтобы получить
// последний etag, и проводим обновление
}
}
```

Для безусловного обновления сущности приложение выполняет следующее:

1. Создает новый объект DataServiceContext или, в случае использования существующего контекста, отсоединяет объект, как демонстрирует пример ниже.

2. Присоединяем сущность к контексту и используем ” ∗ ” как новое значение ETag.

3. Обновляем сущность.

4. Вызываем SaveChanges.

```
// задаем опцию объединения, разрешающую перезапись,
// чтобы обеспечить возможность обновления отслеживаемой сущности
context.Detach(blog);
// Присоединяем сущность к контексту, используя имя таблицы, сущность,
// которая должна быть обновлена, и "*"как значение etag.
context.AttachTo("Blogs", blog, "*");
blog.Text = "Hi there again";
try
{
context.UpdateObject(blog);
DataServiceResponse response = context.SaveChanges();
}
catch (DataServiceRequestException e)
{
```

```
// Обработка ошибки, но в данном случае формировнаие ошибки PreCondition невозможно
}
```

Для запросов, которые могут возвращать большое количество результатов, система обеспечивает два механизма:

1. Возможность получать первые N сущностей, используя LINQ-функцию Take(N).

2. Маркер продолжения, который обозначает место начала следующего множества результатов.

Система поддерживает функцию возвращения первых N соответствующих запросу сущностей. Например, если программа разрабатывается на .NET, для извлечения первых N сущностей (в данном примере это первые 100 сущностей) можно использовать LINQ-функцию Take(N).

```
serviceUri = new Uri("http://<account>.table.core.windows.net");
DataServiceContext svc = new DataServiceContext(serviceUri);
var allBlogs = context.CreateQuery<blog>("Blogs");
foreach (Blog blog in allBlogs.Take(100))
{
// выполняем некоторые операции с каждым блогом
}
```

Аналогичная функциональность поддерживается в интерфейсе REST

через опцию строки запроса $top=N. Например, запрос "GET http://<UriСервиса>/Blogs?$top=100" обеспечил бы возвращение первых 100 сущностей, соответствующих запросу. Фильтрация выполняется на сервере, поэтому в ответе клиенту может быть передано максимум 100 сущностей.

1. В запросе указывается максимальное число сущностей, которое должно быть возвращено.

2. Количество сущностей превышает максимально разрешенное сервером число сущностей в ответе (в настоящее время это 1000 сущностей).

3. Общий размер сущностей в ответе превышает максимально допустимый размер ответа (в настоящее время это 4МБ, включая имена свойств, но исключая xml-теги, используемые для REST).

4. На выполнение запроса требуется больше времени, чем заданный период ожидания сервера (в настоящее время это 60 секунд).

В любом из этих случаев ответ будет включать маркер продолжения в виде специального заголовка. Для запроса к вашим сущностям используются такие заголовки:

- x-ms-continuation-NextPartitionKey
- x-ms-continuation-NextRowKey

Если клиент получил эти значения, он должен передать их со следующим запросом в виде опций HTTP-запроса; во всем остальном запрос остается неизменным. Это обеспечит возвращение следующего набора

сущностей, начинающегося с места, обозначенного маркером продолжения.

Последующий запрос выглядит следующим образом:
http://<UriСервиса>/Blogs?<исходныйЗапрос>& NextPartitonKey=<некотороеЗначение>& NextRowKey=<другоеЗначение>

Это повторяется до тех пор, пока клиентом не будет получен ответ без маркера продолжения, что свидетельствует об извлечении всех соответствующих запросу результатов.

Маркер продолжения должен рассматриваться как непрозрачное значение. Оно указывает на точку начала следующего запроса и может не соответствовать фактической сущности в таблице. Если в таблицу добавляется новая сущность, так что Key (новая сущность) > Key (последняя сущность, извлеченная запросом), но Key (новая сущность) < "Маркера продолжения", тогда эта новая сущность не будет возвращена повторным запросом, использующим маркер продолжения. Но новые сущности, добавленные так, что Key(новая сущность) > "Маркера продолжения", войдут в результаты, возвращаемые последующими использующими маркер продолжения запросами.

Теперь рассмотрим модель согласованности, обеспечиваемую Windows Azure Table.

В рамках одной таблицы система обеспечивает гарантии транзакции ACID для всех операций вставки/обновления/удаления для одной сущности.

Для запросов в рамках отдельной секции выполняется изоляция моментального снимка. Запрос обеспечивается согласованным представлением секции с момента его начала и в течение всей транзакции. Моментальный снимок обеспечивает следующее:

1. Отсутствие "грязного считывания". Транзакция не будет видеть незафиксированные изменения, вносимые другими транзакциями, которые выполняются параллельно. Будут представлены только те изменения, которые были завершены до начала выполнения запроса на сервере.

2. Механизм изоляции моментального снимка позволяет производить чтение параллельно с обновлением секции без блокирования этого обновления.

Изоляция моментального снимка поддерживается только в рамках секции и в рамках одного запроса. Система не поддерживает изоляцию моментального снимка для нескольких секций таблицы или других фаз запроса.

Приложения отвечают за сохранение согласованности между множеством таблиц.

В примере MicroBlogging использовалось две таблицы: Channels и Blogs. Приложение выполняет согласование таблиц Channels и Blogs. Например, когда канал удаляется из таблицы Channels, приложение должно удалить соответствующие блоги из таблицы Blogs.

Сбои могут возникать в моменты синхронизации состояния множества таблиц. Приложение должно уметь обрабатывать такие сбои и иметь

возможность возобновлять работу с момента, на котором она была прервана.

В предыдущем примере, когда канал удаляется из таблицы каналов, приложение должно также удалить все блоги этого канала из таблицы Blogs. В ходе этого процесса могут возникать сбои приложения. Для обработки таких сбоев приложение может сохранять транзакцию в Windows Azure Queues, что позволяет пользователю возобновить операцию удаления канала и всех его блогов даже в случае сбоя.

Вернемся к примеру с таблицами Channels и Blogs. Channels имеет следующие свойства: Name (Имя) как PartitionKey,пустая строка как RowKey, Owner (Владелец), CreatedOn (Дата создания). И Blogs имеет свойства Channel Name (Имя канала) как PartitionKey, CreatedOn как RowKey, Title (Название), Blog, UserId. Теперь, когда канал удален, необходимо обеспечить, чтобы все ассоциированные с ним блоги также были удалены. Для этого выполняем следующие шаги:

1. Создаем очередь для обеспечения согласованности таблиц, назовем ее "DeleteChannelAndBlogs"(Удаление каналов и блогов).

2. При поступлении запроса на удаление канала от роли веб-интерфейса, ставим в созданную выше очередь элемент, определяющий имя канала.

3. Создаем рабочие роль, которые будут ожидать событие добавления элемента в очередь "DeleteChannelAndBlogs".

4. Рабочая роль изымает элемент из очереди DeleteChannelAndBlogs, задавая для извлеченного элемента очереди время невидимости в тече-

ние N секунд. При этом элемент, определяющий имя канала, который должен быть удален, изымается. Если роль работника удаляет элемент очереди в течение этих N секунд, данный элемент будет удален из очереди. Если нет, элемент станет вновь видимым и доступным для использования рабочей ролью. При извлечении элемента рабочая роль делает следующее:

- В таблице Channels помечает канал как недействительный, чтобы с этого момента никто не мог выполнять чтение из него.
- Удаляет из таблицы Blogs все записи, для которых PartitionKey = "имени канала" указанному в элементе очереди.
- Удаляет канал из таблицы Channels.
- Удаляет элемент из очереди.
- Возвращается.

Если в ходе выполнения, например, шага 4, возникает какой-либо сбой, производится аварийное завершение рабочего процесса, при этом элемент очереди не удаляется из нее. Таким образом, как только соответствующий элемент очереди станет снова видимым (т.е. когда истечет время, заданное как время ожидания видимости), это сообщение будет вновь извлечено из очереди рабочим процессом, и процесс удаления возобновится с шага 4. Более подробно обработка очередей рассматривается в документации Windows Azure Queue.

Краткие итоги:

В данной лекции мы ознакомились с одной из абстракций данных *Windows* Azure *Storage* – *Windows* Azure *Table*. Это технология, которая обеспечивает структурированное хранилище состояний сервиса.

Ключевые термины

Windows Azure Table – *абстракция* данных, которая обеспечивает структурированное хранилище состояний сервиса.

Windows Azure Blob – *абстракция* данных, которая обеспечивает хранилище больших элементов данных.

Windows Azure Queue – *абстракция* данных, которая обеспечивает диспетчеризацию асинхронных заданий для реализации обмена данными между сервисами.

§8. Архитектура Windows Azure Platform.

Цель данной лекции – получить *представление* об архитектуре *Windows* Azure.

8.1 Azure Blob Services

Для работы с *Windows Azure Storage пользователь* должен создать учетную *запись* хранилища. Выполняется это через веб-*интерфейс* портала *Windows* Azure *Portal*. При создании учетной записи *пользователь* получает 256-разрядный *секретный ключ*, который впоследствии используется для аутентификации запросов этого пользователя к системе хранения. В частности, с помощью этого секретного ключа создается подпись HMAC SHA256 для запроса. Эта подпись передается с каждым запросом данного пользователя для обеспечения аутентификации через проверку достоверности подписи HMAC.

Благодаря *Windows* Azure *Blob* приложения получают возможность хранения в облаке больших объектов, до 50 ГБ каждый. Он поддерживает высоко масштабируемую систему больших двоичных объектов (*blob*), в которой наиболее часто используемые *blob* распределяются среди *множества* серверов для обслуживания необходимых объемов трафика. Более того, эта система характеризуется высокой надежностью и длительностью хранения. Данные доступны в любой момент времени

из любой точки планеты и продублированы, *по* крайней мере, трижды для повышения надежности. Кроме того, обеспечивается строгая согласованность, что гарантирует немедленную доступность объекта при его добавлении или обновлении: все изменения, внесенные в предыдущей *операции* записи, немедленно видны при последующем чтении.

Рассмотрим *модель данных Azure Blob*. На рисунке ниже представлено *пространство имен Windows* Azure *Blob*.

- Учетная запись хранилища – любой доступ к Windows Azure Storage осуществляется через учетную запись хранилища.
  - Это самый высокий уровень пространства имен для доступа к объектам blob.
  - Учетная запись может иметь множество контейнеров Blob.

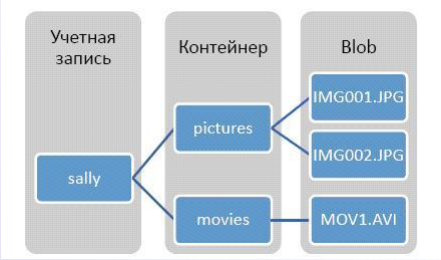


Рисунок 8.1: Общее представление хранилища Blob

- Контейнер Blob – Контейнер обеспечивает группировку набора объектов blob. Область действия имени контейнера ограничена учетной записью.
  - Политики совместного использования задаются на уровне контейнера. В настоящее время поддерживаются "Public READ" (Открытое чтение) и "Private"(Закрытый). Если для контейнера определена политика "Public READ" , все его содержимое открыто доступно для чтения без необходимости аутентификации. Политика "Private"означает доступ с аутентификацией, т.е. только владелец соответствующей учетной записи имеет доступ к объектам blob этого контейнера.
  - С контейнером могут быть ассоциированы метаданные, которые задаются в виде пар <имя, значение>. Максимальный размер метаданных контейнера – 8КБ.
  - Существует возможность получения списка всех объектов blob контейнера.
- Blob – Объекты blob хранятся в контейнерах Blob Container и их область действия ограничена этими контейнерами. Каждый blob может быть размером до 50ГБ и имеет уникальное в рамках контейнера строковое имя. С blob могут быть ассоциированы метаданные, которые задаются в виде пар <имя, значение> и могут достигать

размера 8КБ для blob. Метаданные blob могут быть получены и заданы отдельно от данных blob.

Для доступа к *Windows* Azure *Blob* используется приведенные выше подходы. *URI* конкретного *blob* структурирован следующим образом:

http://<учетнаязапись>.blob.core.windows.net /<контейнер>/<имяblob>

Первая часть имени хоста образована именем учетной записи хранилища, за которым следует ключевое *слово "blob"*. Это обеспечивает направление запроса в часть *Windows* Azure *Storage*, которая обрабатывает запросы *blob*. За именем хоста идет имя контейнера, "/"и затем имя *blob*. Существуют ограничения именования учетных записей и контейнеров (подробнее об этом рассказывается в документе *Windows* Azure *SDK*). Например, имя контейнера не может включать символ "/".

Еще несколько замечаний *по* поводу контейнеров:

- Как говорилось выше, область действия контейнеров ограничена учетной записью, которой они принадлежат. Контейнеры хранятся распределено, что устраняет вероятность возникновения "узких мест"для трафика при работе с ними.
- Возможна задержка при воссоздании контейнера, который был недавно удален, особенно если в этом контейнере находилось большое количество объектов blob. Система должна очистить объекты blob этого контейнера, прежде чем контейнер с таким же именем

сможет быть создан вновь. Пока сервер удаляет все объекты blob, попытки создания контейнера с таким же именем будут оканчиваться неудачей с формированием ошибки, указывающей на то, что контейнер находится в процессе удаления.

- Команды на удаление или создание совершенно нового контейнера быстро передаются на сервер, и приложению возвращается подтверждение об их выполнении, даже если операция удаления занимает некоторое время.

Рассмотрим *интерфейс* REST объектов *Blob*. Любой *доступ* к *Windows* Azure *Blob* выполняется через стандартные *HTTP*-команды PUT/*GET*/*DELETE* интерфейса REST.

К командам *HTTP*/REST, поддерживаемым для реализации операций с *blob*, относятся:

- PUT Blob – Вставить новый или перезаписать существующий blob с заданным именем.
- GET Blob – Получить весь blob или диапазон байтов blob, используя стандартную HTTP-операцию для возвращения диапазона GET.
- DELETE Blob – Удалить существующий blob.

Все эти *операции* с *blob* – *Put, Get* и *Delete,* – могут быть выполнены с использованием следующего URL:

http://<учетнаязапись>.blob.core.windows.net/<контейнер>/<имяblob>

Один *запрос* PUT обеспечивает возможность загрузки в облако *blob* размером до 64 МБ. Для загрузки *blob*, размер которого превышает 64 МБ, используется технология загрузки блоками, описываемый в следующем разделе.

Полное описание всех *API* REST можно найти в документе *Windows* Azure *SDK*.

Один из целевых сценариев *Windows* Azure *Blob* – эффективная *загрузка* объектов *blob*, размером десятки *гигабайт*. *Windows* Azure *Blob* обеспечивает это следующим образом:

Загружаемый *Blob* (например, Movie.*avi*) разбивается на последовательные блоки. Например, ролик размером 10ГБ может быть разбит на 2500 блоков *по* 4МБ, при этом первый блок будет представлять *диапазон* байтов данных от 1 до 4194304, второй блок – от 4194305 до 8388608 и т.д.

- Каждому блоку присваивается уникальный ID/имя. Область действия этого уникального ID ограничена именем загружаемого blob. Например, первый блок может быть назван "Block 0001 второй – "Block 0002"и т.д.
- Каждый блок размещается в облаке. Для этого используется операция PUT с указанием представленного выше URL с запросом,

определяющим, что это операция размещения блока, и ID блока. Продолжая пример, при размещении первого блока будет указано имя blob "Movie.avi"и ID блока "Block 0001".

- Когда все блоки размещены в Windows Azure Storage, передается список загруженных блоков, чтобы представить blob, с которым они ассоциированы. Выполняется это с помощью операции PUT с указанием представленного выше URL с запросом, определяющим, что это команда blocklist. После этого HTTP-заголовок включает список блоков, которые должны использоваться в этом blob. В случае успешного завершения этой операции получаем список блоков, представляющий пригодную для чтения версию blob. Теперь blob может быть считан с помощью описанной выше команды GET Blob.

На следующем рисунке представлено *место* блоков в концепции данных *Windows* Azure *Blob*.

Как описывалось ранее, *доступ* к объектам *blob* может осуществляться посредством операций PUT и *GET* с использованием следующего *URL*:

http://<учетнаязапись>.blob.core.windows.net/<контейнер>/<имяblob>

В примере, представленном на рисунке 8.2, рисунки со следующими *URL* могут быть размещены одной операцией PUT:

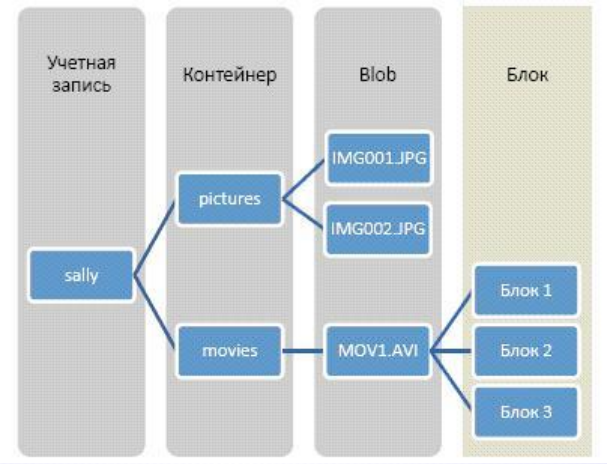


Рисунок 8.2: Общее представление хранилища Blob, добавление блоков

http://sally.blob.core.windows.net/pictures/IMG001.JPG
http://sally.blob.core.windows.net/pictures/IMG002.JPG

Те же *URL* могут использоваться для возвращения объектов *blob*. Одна операция PUT может обеспечить *размещение* в хранилище объектов *blob* размером до 64МБ. Для сохранения объектов *blob* размером больше 64МБ и вплоть до 50 ГБ необходимо сначала разместить все блоки посредством соответствующего количества операций PUT и затем, с помощью все той же *операции* PUT, передать *список* блоков, чтобы обеспечить пригодную для чтения версию *blob*. В примере, который иллюстрирует рис. 2, только после размещения всех блоков и подтверждения их принадлежности *blob* посредством списка блоков *blob* может

быть считан с использованием следующего *URL*:

http://sally.blob.core.windows.net/pictures/MOV1.AVI

Операции GET всегда выполняются на уровне *blob* и не предполагают использования блоков.

Рассмотрим абстракции данных блоков. Каждый блок идентифицирует *ID* блока размером до 64 *байт*. *Область действия ID* блока ограничена именем *blob*, поэтому разные объекты *blob* могут иметь блоки с одинаковыми *ID*. Блоки неизменны. Каждый блок может быть размером до 4МБ, и один *blob* может включать блоки разного размера. *Windows* Azure *Blob* обеспечивает следующие *операции* уровня блока:

- PUT block – загрузить блок blob. Обратите внимание, что успешно загруженный посредством операции PUT block блок не является частью blob до тех пор, пока это не будет подтверждено списком блоков, загружаемым операцией PUT blocklist.
- PUT blocklist – подтвердить blob через предоставление списка ID блоков, его составляющих. Указанные в этой операции блоки должны быть уже успешно загружены через вызовы PUT. Порядок блоков в операции PUT blocklist обеспечит пригодную для чтения версию blob.
- GET blocklist – извлечь список блоков, переданный ранее для blob операцией PUT blocklist. В возвращаемом списке блоков указываются ID и размер каждого блока.

Во всех рассматриваемых далее примерах используется *blob "MOV1.avi"*, располагающийся в контейнере "*movies*"(ролики) под учетной записью "sally".

Ниже представлен пример REST-запроса для размещения блока размером 4МБ посредством *операции* PUT *block*. Обратите внимание, что используется *HTTP-команда* PUT. *"?comp=block"* указывает на то, что это операция PUT *block*. Затем задается BlockID. *Параметр* Content-*MD5* может быть задан для защиты от ошибок передачи *по* сети и обеспечения целостности. В данном случае, Content-*MD5* – это *контрольная сумма MD5* данных блока в запросе. *Контрольная сумма* проверяется на сервере, в случае несовпадения возвращается ошибка. *Параметр Content-Length* (*Длина* содержимого) определяет размер содержимого блока. Также в заголовке *HTTP*-запроса имеется заголовок авторизации, как показано ниже.

```
PUT http://sally.blob.core.windows.net/movies/MOV1.avi
?comp=block & blockid=BlockId1 & timeout=60
HTTP/1.1 Content-Length: 4194304
Content-MD5: HUXZLQLMuI/KZ5KDcJPcOA==
Authorization: SharedKey sally:
F5a+dUDvef+PfMb4T8Rc2jHcwfK58KecSZY+l2naIao=
x-ms-date: Mon, 27 Oct 2008 17:00:25 GMT
..... Block Data Contents .....
```

Ниже представлен пример REST-запроса для *операции* PUT blocklist.

Обратите внимание, что используется *HTTP-команда* PUT. *"?comp=blocklist"* указывает на то, что это операция PUT blocklist. *Список* блоков задается в теле *HTTP*-запроса в формате *XML*, как показано в примере ниже. Обратите внимание, что *значение* поля *Content-Length* в заголовке запроса соответствует размеру тела запроса, а не размеру создаваемого *blob*. Также в заголовке *HTTP*-запроса имеется заголовок авторизации, как показано ниже.

```
PUT http://sally.blob.core.windows.net/movies/MOV1.avi
?comp=blocklist & timeout=120
HTTP/1.1 Content-Length: 161213
Authorization: SharedKey sally:
QrmowAF72IsFEs0GaNCtRU143JpkflIgRTcOdKZaYxw=
x-ms-date: Mon, 27 Oct 2008 17:00:25 GMT
<?xml version="1.0"encoding="utf-8"?>
<BlockList>
<Block>BlockId1</Block>
<Block>BlockId2</Block>
. . . . . .
</BlockList>
```

Ниже представлен пример REST-запроса для *операции GET blob*. В данном случае используется *HTTP-команда GET*. Этот *запрос* обеспечит извлечение всего содержимого заданного *blob*. Если для контейнера, которому принадлежит *blob* (в данном примере "movies"), задана поли-

тика совместного использования *"Private"*, для получения *blob* необходимо пройти аутентификацию. Если задана политика совместного использования "Public-Read", *аутентификация* не требуется, и заголовок аутентификации в заголовке запроса не нужен.

GET http://sally.blob.core.windows.net/movies/MOV1.avi
HTTP/1.1
Authorization: SharedKey sally:
RGllHMtzKMi4y/nedSk5Vn74IU6/fRMwiPsL+uYSDjY=
X-ms-date: Mon, 27 Oct 2008 17:00:25 GMT

Как показано в примере ниже, также поддерживается операция GET для извлечения
диапазона байт заданного blob.

GET http://sally.blob.core.windows.net/movies/MOV1.avi
HTTP/1.1
Range: bytes=1024000-2048000

Загрузка blob в виде списка блоков обладает следующими преимуществами:

- Возможность продолжения – для каждого блока в отдельности можно проверить успешность загрузки, в случае сбоя повторить попытку загрузки и продолжить выполнение с этого момента.
- Параллельная загрузка – загрузка блоков может выполняться параллельно, что обеспечивает сокращение времени загрузки очень

больших объектов blob.

- Загрузка не по порядку – Блоки могут загружаться в произвольном порядке. Значение имеет лишь порядок блоков в списке операции PUT blocklist. Список блоков в операции PUT blocklist определяет пригодную для чтения версию blob.

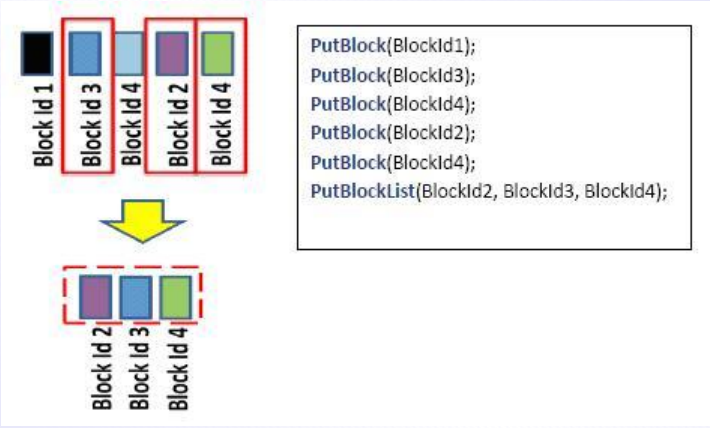


Рисунок 8.3: Сценарий загрузки блоков

- Используя представленный на рисунке 8.3 пример, опишем различные сценарии, возможные при использовании блоков для загрузки объектов blob:
- Загрузка блоков с одинаковыми ID блока – когда для одного blob загружаются блоки с одинаковыми ID блока, при формировании окончательно версии blob в операции PUT blocklist из всех блоков с одинаковым ID будет использоваться только загруженный самым последним. В примере выше загружаются два блока с BlockId4, и только второй из них будет использоваться в окончательном списке блоков blob.
- Загрузка блоков в произвольном порядке – блоки могут загружаться в порядке, отличном от указанного в окончательном списке блоков blob. В примере выше в окончательном списке блоки располагаются в порядке BlockId2, BlockId3 и BlockId4, но загружались они в другой последовательности. *Упорядочивание данных* blob (через операцию GET) в пригодную для чтения версию выполняется соответственно списку, указанному в операции PUT blocklist.
- Неиспользуемые блоки – более того, некоторые блоки могут никогда не войти в окончательный список блоков blob. Эти блоки будут удалены системой в процессе сборки мусора. В рассматриваемом примере такими блоками являются BlockId1 и первый блок с ID BlockId4. Точнее говоря, как только blob создан посредством опе-

рации PUT blocklist, все загруженные, но не вошедшие в список блоков операции PUT blocklist блоки будут удалены путем сборки мусора.

Загрузка большого *blob* может занимать довольно длительное время. При этом загруженные, но не использованные блоки занимают *место* в хранилище. Многие загруженные блоки могут никогда не войти в *список* PUT blocklist. В случае отсутствия активности для данного *blob* в течение длительного периода времени (в настоящее время этот период составляет неделю), эти неиспользованные блоки будут удалены системой в процессе сборки мусора.

Любопытным является *сценарий* параллельной загрузки блоков для одного *blob*. В этом случае заслуживают внимания два вопроса:

- ID блоков – если для загрузки блоков в один blob приложение использует множество клиентских модулей записи, во избежание конфликтов ID блоков должны быть уникальными среди всех этих модулей записи, или они должны представлять содержимое записываемого блока (таким образом, если один и тот же блок записывается несколькими клиентами, ID блока во всех клиентах будет одинаковым, поскольку он представляет одни и те же данные). Во избежание ошибок при потенциальной возможности записи одного Blob несколькими модулями записи в качестве ID блока рекомендуется использовать хеш (например, MD5-хеш) содержимого блока. Таким

образом, ID блока будет представлять его содержимое.

- Приоритет имеет первая фиксация – в ситуации, когда множество клиентов выполняют загрузку блоков для одного blob параллельно, приоритет имеет первая фиксация blob посредством операции PUT blocklist (или модуль записи, вызвавший PUT blob первым). Все остальные неиспользованные блоки, загруженные другими модулями записи для blob с этим именем, будут удалены в процессе сборки мусора. Следовательно, для эффективного параллельного обновления blob необходима координация всех клиентов, ведущих запись параллельно.

Windows Azure *Blob* поддерживает условные PUT и *GET*, которые обеспечивают реализацию эффективной обработки параллелизма и клиентского кэширования.

Условный PUT может использоваться в ситуациях, когда один *blob* обновляется несколькими пользователями. Например, *загрузка blob* может выполняться на основании времени последнего изменения; это гарантирует, что версия изменяемого *blob* аналогична версии, которую изменяет клиент. Так может быть реализован *совместный доступ* с нежесткой блокировкой. Скажем, два клиента, A и B, обновляют один и тот же *blob*. Они параллельно выполняют чтение версии *blob*, вносят в нее какие-то изменения и вновь загружают в хранилище. В этом сценарии каждый из клиентов записывает *время последнего изменения*

извлеченного из хранилища *blob* (пусть *время последнего изменения* будет X). Когда они готовы загрузить обновленную версию *blob* назад в хранилище, они делают это с помощью условного PUT на основании сохраненного при извлечении *blob* времени последнего изменени я. В *операции* должно быть определено, что условием выполнения PUT является "если не изменялся с момента X". Таким образом, если *blob* был изменен другим клиентом в промежуток времени с момента X, операция обновления даст сбой, и клиент получит уведомление об этом.

Условный *GET* может использоваться для эффективной обработки вопросов соответствия содержимого кэшей. Например, клиент имеет *локальный кэш* объектов *blob*, в котором кэшируются чаще всего извлекаемые из хранилища *blob*. Для каждого кэшированного *blob* записывается время его последнего изменения. Когда клиентский *кэш* принимает решение обновить объекты *blob* из хранилища, он может использовать условный *GET* на основании времени изменения (с условием "если изменен с момента X"). Таким образом, из хранилища будут загружаться только те объекты *blob*, которые были изменены в период времени, прошедший с момента X, и отличаются от своей кэшированной копии.

Система *Blob* обеспечивает *интерфейс* для перечисления объектов *blob* контейнера. Поддерживается иерархический перечень объектов *blob* контейнера и механизм продолжения, что позволяет выполнять *перечисление* большого количества объектов *blob*.

Интерфейс ListBlobs поддерживает параметры *"prefix"(префикс)* и

"delimiter" (разделитель), которые обеспечивают возможность построения иерархического перечня объектов *blob*. Например, пусть в учетной записи *"sally"* имеется *контейнер "movies"* объектов *blob* с такими именами:

Action/Rocky1.wmv
Action/Rocky2.wmv
Action/Rocky3.wmv
Action/Rocky4.wmv
Action/Rocky5.wmv
Drama/Crime/GodFather1.wmv
Drama/Crime/GodFather2.wmv
Drama/Memento.wmv
Horror/TheBlob.wmv

Как видите, "/"используется в качестве разделителя для создания подобной каталогу иерархии имен *blob*. Чтобы получить *список* всех "папок", задаем в запросе *ListBlobs "delimiter=/"*. И вот как будет выглядеть *запрос* и часть ответа:

Запрос:

GET http://sally.blob.windows.net/movies?comp=list& delimiter=/

Ответ:

<BlobPrefix>Action</BlobPrefix>
<BlobPrefix>Drama</BlobPrefix>
<BlobPrefix>Horror</BlobPrefix>

Обратите внимание, *тег* "BlobPrefix"указывает на то, что соответствующая *запись* является префиксом имени *blob*, а не полным именем *blob*. Также следует заметить, что один и тот же *префикс* возвращается в результате только один раз.

Следующим этапом можно сочетать *префикс* и разделитель для получения списка содержимого "подпапки". Например, задавая "prefix=Drama/" и "delimiter=/", получаем *список* всех подпапок и файлов каталога *"Drama"*:

Запрос:

GET http://sally.blob.windows.net/movies?comp=list & prefix=Drama/ & delimiter=/

Ответ:

<BlobPrefix>Drama/Crime</BlobPrefix>

<Blob>Drama/Memento.wmv</Blob>

Обратите внимание, что *"Drama/Memento.wmv"* – это *полное имя blob*, следовательно, оно так и обозначено.

Интерфейс ListBlobs обеспечивает возможность задавать *"maxresults"*, т.е. максимальное число результатов, которое должно быть возвращено в этом вызове. Более того, система определяет верхний *предел* для максимального числа результатов, которые могут быть возвращены одним вызовом (более подробную информацию об этом можно найти в документации *по SDK*). *По* достижении меньшего из этих двух предельных значений вызов возвращается с соответствующим количеством результа-

тов и непрозрачным *"NextMarker"*(маркер следующего). Наличие этого маркера свидетельствует о том, что данный запрос не обеспечил возвращения всех возможных результатов. *"NextMarker"* может использоваться для продолжения составления списка для следующей страницы результатов.

В предыдущем примере предположим, что требуется составить *список* всех объектов *blob* каталога *"Action"*, возвращая каждый раз *максимум по* 3 результата. В этом случае первый набор результатов был бы таким:

Запрос:

GET http://sally.blob.windows.net/movies?comp=list & prefix =Action & maxresults=3

Ответ:

<Blob>Action/Rocky1.wmv</Blob>
<Blob>Action/Rocky2.wmv</Blob>
<Blob>Action/Rocky3.wmv</Blob>
</NextMarker> OpaqueMarker1</NextMarker>

С первым набором объектов *blob* возвращается и непрозрачный маркер, который может быть передан во второй вызов ListBlobs. Тогда этот вызов обеспечит возвращение следующих результатов:

Запрос:

GET http://sally.blob.windows.net/movies?comp=list
& prefix=Action & maxresults=3 & marker=OpaqueMarker1

Ответ:

<Blob>Action/Rocky4.wmv</Blob>
<Blob>Action/Rocky5.wmv</Blob>
</NextMarker></NextMarker>

Как показано выше, возвращены оставшиеся объекты *blob* каталога; *"NextMarker"* пуст, это свидетельствует о том, что получены все результаты.

8.2 Azure Queue Services

Windows Azure Queue предоставляет надежный механизм доставки сообщений. Она предлагает простой *алгоритм* диспетчеризации асинхронных заданий, который обеспечивает возможность подключения к разным компонентам приложения в облаке. Очереди *Windows* Azure Queue характеризуются высокой надежностью, постоянством и производительностью. Текущая реализация гарантирует, *по* крайней мере, однократную обработку сообщения. Более того, *Windows* Azure Queue имеет REST-*интерфейс*, таким образом, приложения могут создаваться на любом языке программирования и выполнять *доступ* к очереди через веб в любое время из любой точки Интернета.

Рассмотрим создание приложений в облаке с использованием Azure Queue. *Windows* Azure Queue позволяет разделить разные части приложения в облаке, что делает возможным использование разных техно-

логий для создания этих приложений, и их масштабирование соответственно нуждам трафика.

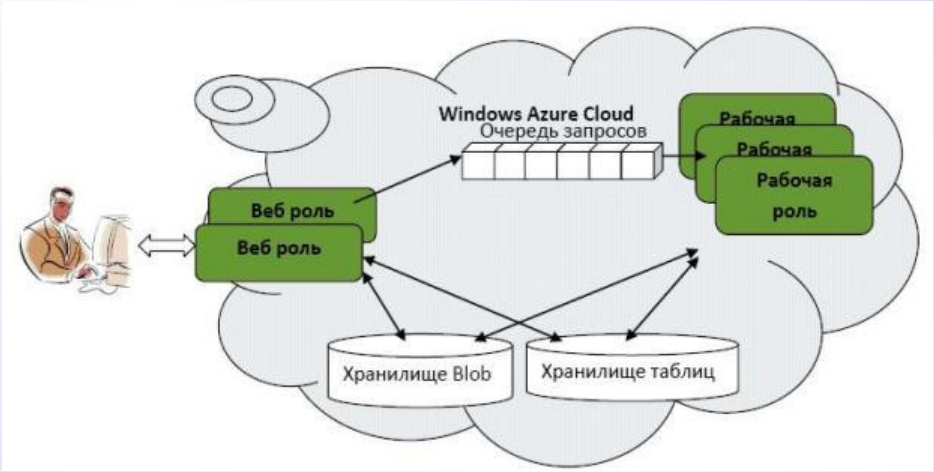


Рисунок 8.4: Построение приложений для облака с использованием Azure Queue

Приведенный выше рисунок иллюстрирует простой и распространенный *сценарий* для приложений в облаке. Имеется ряд веб ролей, на которых размещается интерфейсная логика обработки веб-запросов. Также существует ряд рабочих ролей, реализующих бизнес-логику приложения. Веб роли обмениваются информацией с рабочими ролями посредством наборов запросов. Устойчивое состояние приложения может сохраняться в хранилищах *Windows* Azure *Blob* и *Windows* Azure *Table*.

Рассмотрим в качестве примера *приложение онлайн* сервиса видеохостинга. Пользователи могут загружать видео в это *приложение*; после этого *приложение* может автоматически преобразовывать и сохранять этот видеофайл в различных форматах *мультимедиа*; кроме того, *приложение* автоматически индексирует данные описания видео для упрощения поиска (например, *по* ключевым словам описания, именам актеров, режиссеров, названию и т.д.).

Такое *приложение* может использовать описанную ранее архитектуру. Веб-роли реализуют *уровень представления* и обрабатывают вебзапросы пользователей. Пользователи могут загружать видео через вебинтерфейсы. Видео-файлы могут храниться как большие двоичные объекты в хранилище Azure *Blob*. *Приложение* может также обслуживать ряд таблиц для учета имеющихся видеофайлов и хранения индексов, используемых для поиска. Рабочие роли выполняют преобразование входящих видеофайлов в разные форматы и сохранение их в хранилище Azure *Blob*. Рабочие роли также отвечают за обновление таблиц этого приложения в Azure *Table*. При получении запроса пользователя (например, запроса на загрузку видео) веб-роли создают рабочий элемент и помещают его в *очередь* запросов. Рабочие роли извлекают эти рабочие элементы из очереди и обрабатывают соответствующим образом. В случае успешной обработки рабочая роль должна удалить рабочий элемент из очереди во избежание повторной его обработки другой рабочей ролью.

Благодаря применению *Windows* Azure Queue такая *архитектура* обладает рядом преимуществ:

1. Масштабируемость – *Приложение* может легче масштабироваться соответственно нуждам трафика. Вот преимущества, связанные с масштабированием: Во-первых, *длина* очереди напрямую отражает насколько хорошо рабочие роли справляются с общей рабочей нагрузкой. Увеличение очереди свидетельствует о том, что рабочие роли не могут обрабатывать данные с необходимой скоростью. В этом случае приложению может потребоваться увеличить количество рабочих ролей, чтобы обеспечить более быструю обработку. Если *длина* очереди неизменно остается близкой к нулю, это означает, что серверная часть приложения обладает большими вычислительными мощностями, чем требуется. В этом случае *приложение* может сократить количество рабочих ролей для обеспечения рационального расходования ресурсов. Отслеживая размеры очереди и настраивая количество внутренних узлов, приложения могут эффективно масштабироваться соответственно объемам трафика. Во-вторых, применение очередей позволяет разделить части приложения и выполнять их масштабирование независимо друг от друга, что намного упрощает эту задачу. В данном примере веб-роли и рабочие роли отделены друг от друга и обмениваются информацией через очереди. Это позволяет настраивать количество веб-ролей и рабочих ролей независимо друг от друга без влияния на логику приложения. Таким образом, *приложение* может свободно масштабировать критически важ-

ные компоненты, добавляя для них больше ресурсов/компьютеров. В-третьих, применение очередей обеспечивает гибкость для эффективного использования ресурсов в приложении, что повышает эффективность масштабирования. То есть для рабочих элементов с разными приоритетами и/или разных размеров могут использоваться разные очереди, которые будут обрабатываться разными пулами рабочих ролей. В этом случае *приложение* может распределять соответствующие ресурсы (например, с точки зрения количества серверов) для каждого пула, таким образом, обеспечивая эффективное использование доступных ресурсов при разных характеристиках трафика. Например, рабочие элементы, имеющие критически важное *значение* для всей задачи, могут быть помещены в отдельную *очередь*. Это обеспечит их обработку независимо от всех остальных задач. Кроме того, отдельная *очередь* может использоваться для рабочих элементов, обработка которых требует привлечения большого количества ресурсов (например, преобразование видео). Для каждой из этих очередей могут использоваться разные пулы рабочих ролей. *Приложение* может настраивать размер каждого из пулов независимо, соответственно поступающему трафику.

2. Разделение ролей – Использование очередей позволяет разделить разные части приложения, что обеспечивает существенную гибкость и *расширяемость* с точки зрения построения приложения. Сообщения в очереди могут быть в стандартном или расширяемом формате, таком как *XML*, что обеспечивает независимость компонентов на обоих кон-

цах очереди друг от друга, поскольку они могут понимать сообщения в очереди. Разные части системы могут быть реализованы с использованием разных технологий и языков программирования. Например, *компонент* на стороне очереди может быть написан на *.NET* Framework, а другой *компонент* – на *Python*. Более того, изменения, происходящие внутри компонента, не имеют никакого влияния на остальную систему. Например, *компонент* может быть изменен с применением совершенно другой технологии или языка программирования, при этом система будет продолжать работать точно так же, и для этого не придется изменять другие компоненты, поскольку использование очередей обеспечивает разделение компонентов. Кроме того, использование очередей делает возможным сосуществование в системе разных реализаций одного и того же компонента, т.е. *приложение* может свободно переходить к новым технологиям. В рассматриваемом выше примере можно постепенно уходить от компонентов, созданных с применением устаревших технологий, и заменять их новыми реализациями. Старая и новая реализации могут выполняться одновременно на разных серверах и обрабатывать рабочие элементы одной очереди. При этом другие компоненты приложения никак не будут затронуты.

3. Всплески трафика – Очереди обеспечивают буферизацию, что компенсирует всплески трафика и сокращает влияние, оказываемое сбоями отдельных компонентов. В рассматриваемом ранее примере возможны случаи поступления большого числа запросов в короткий промежуток

времени. Рабочие роли не могут быстро обработать все запросы. В этом случае запросы не отклоняются, а буферизуются в *очередь*, и рабочие роли получают возможность обрабатывать их в собственном нормальном темпе, постепенно возвращаясь к обычному режиму работы. Это позволяет приложению обрабатывать неравномерный трафик без снижения надежности. Более того, использование очередей также снижает влияние, оказываемое сбоями отдельных компонентов. В рассматриваемом выше примере в случае сбоя нескольких рабочих ролей *очередь* обеспечит буферизацию всех рабочих элементов, поступивших во время простоя рабочих ролей, таким образом, эти элементы не будут утрачены. Когда рабочие роли вернуться в работу, они смогут продолжить обработку рабочих элементов из очереди и, в конце концов, вернуться к нормальному режиму обработки данных *по* мере их поступления. Такие сбои не оказывают никакого влияния на другие компоненты. Обратите внимание, что рабочие элементы, обрабатываемые рабочими ролями на момент их сбоя, также не будут утеряны, поскольку возвратятся в *очередь по* истечении времени ожидания видимости (VisibilityTimeout), что обеспечивает *сохранность данных* при сбоях компонентов. Таким образом, *приложение* может переживать сбои без потери надежности.

Итак, благодаря модели очереди приложения застрахованы от потери данных и снижения надежности даже в условиях систематических сбоев компонентов приложения. Для обеспечения корректной работы этой модели разработчик приложения должен обеспечить идемпотентность

обработки рабочих элементов очереди рабочими ролями. Благодаря этому, прежде чем рабочий элемент будет полностью обработан и удален из очереди, допускаются его многократные частичные обработки, прерывающиеся в результате сбоев.

Windows Azure Queue имеет следующую *модель данных*.

- **Учетная запись хранилища** – Любой доступ к Windows Azure Storage осуществляется через учетную запись хранилища.
  - Это самый высокий уровень пространства имен для доступа к очередям и их сообщениям. Для использования Windows Azure Storage пользователь должен создать учетную запись хранилища. Выполняется это через веб-интерфейс портала Windows Azure Portal. При создании учетной записи пользователь получает 256-разрядный секретный ключ, который впоследствии используется для аутентификации запросов этого пользователя к системе хранения. В частности, с помощью этого секретного ключа создается подпись HMAC SHA256 для запроса.

    Эта подпись передается с каждым запросом данного пользователя для обеспечения аутентификации через проверку достоверности подписи HMAC.
  - Учетная запись может иметь множество очередей.
- **Очередь** – Очередь содержит множество сообщений. Область действия имени очереди ограничена учетной записью.

1. Количество сообщений в очереди не ограничено.

2. Сообщение хранится максимум неделю. Система удаляет сообщения, поступившие более недели назад, в процессе сборки мусора.

3. С очередями могут быть ассоциированы метаданные. Метаданные представляются в форме пар <имя, значение>, их размер может составлять максимум 8КБ на очередь.

- **Сообщения** – Сообщения хранятся в очередях. Каждое сообщение может быть размером не более 8КБ. Для хранения данных большего размера используются хранилища Azure Blob или Azure Table, а в сообщении указывается имя большого двоичного объекта/сущности. Обратите внимание, что когда сообщение помещается в хранилище, его данные могут быть двоичными. Но при извлечении сообщений из хранилища ответ формируется в формате XML, и данные сообщения возвращаются base64-кодированными. Сообщения могут возвращаться из очереди в любом порядке, и сообщение может быть возвращено несколько раз. Рассмотрим некоторые параметры, используемые Azure Queue Service:

  0. *MessageID*: Значение GUID, которое идентифицирует сообщение в очереди.

  1. *VisibilityTimeout*: Целое значение, определяющее время ожидания видимости сообщения в секундах. Максимальное значение – 2 часа. Значение по умолчанию – 30 секунд.

2. *PopReceipt*: Строка, возвращаемая для каждого извлеченного сообщения. Эта строка, вместе с MessageID, необходима для удаления сообщения из очереди (Queue). Данный параметр следует рассматривать как непрозрачный, поскольку в будущем его формат и содержимое могут быть изменены.

3. *MessageTTL*: Определяет срок жизни (time-to-live, TTL) сообщения в секундах. Максимально допустимый срок жизни – 7 дней. Если этот параметр опущен, срок жизни по умолчанию – 7 дней. Если в течение срока жизни сообщение не будет удалено из очереди, оно будет удалено системой хранения в процессе сборки мусора.

URI конкретной очереди структурировано следующим образом:
http://<учетнаязапись>.queue.core.windows.net/<ИмяОчереди>

Первая часть имени хоста образована именем учетной записи хранилища, за которым следует ключевое слово "queue". Это обеспечивает направление запроса в часть *Windows* Azure *Storage*, которая обрабатывает запросы очереди. За именем хоста идет имя очереди. Существуют ограничения именования учетных записей и очередей (подробнее об этом рассказывается в документе *Windows* Azure *SDK*). Например, имя очереди не может включать символ "/".

Любой *доступ* к *Windows* Azure Queue выполняется через *HTTP-интерфейс* REST. Поддерживаются как *HTTP*, так и *HTTPS* протоколы.

К командам *HTTP*/REST на уровне учетной записи относятся:

- *List Queues* – Представить список всех очередей данной учетной записи.

К командам *HTTP*/REST на уровне очереди относятся:

- *Create Queue* – Создать очередь под данной учетной записью.
- *Delete Queue* – Удалить указанную очередь и ее содержимое без возможности восстановления.
- *Set Queue Metadata* – Задать/обновить определяемые пользователем метаданные очереди. Метаданные ассоциированы с очередью в виде пар имя/значение. Эта команда обеспечит перезапись всех существующих метаданных новыми.
- *Get Queue Metadata* – Извлечь определяемые пользователем метаданные очереди, а также примерное число сообщений в заданной очереди.

Операции уровня очереди могут выполняться с использованием следующего *URL*:

http://<учетнаязапись>.queue.core.windows.net/<ИмяОчереди>

К командам *HTTP*/REST, поддерживаемым для реализации операций на уровне сообщения, относятся:

- *PutMessage* (QueueName, Message, MessageTTL) – Добавить новое сообщение в конец очереди. MessageTTL определяет срок жизни данного сообщения. Сообщение может храниться в текстовом или двоичном формате, но возвращается base64-кодированным.
- *GetMessages* (QueueName, NumOfMessages N, VisibilityTimeout T) – Извлечь N сообщений из начала очереди и сделать их невидимыми в течение заданного VisibilityTimeout промежутка времени T. Эта операция возвратит ID сообщения, возвращенного вместе с PopReceipt. Сообщения могут возвращаться из очереди в любом порядке, и сообщение может быть возвращено несколько раз.
- *DeleteMessage* (QueueName, MessageID, PopReceipt) – Удалить сообщение, ассоциированное с данным PopReceipt, возвращенным ранее вызовом GetMessage. Обратите внимание, что если сообщение не будет удалено, оно повторно появится в очереди по истечении VisibilityTimeout.
- *PeekMessage* (QueueName, NumOfMessages N) – Извлечь N сообщений из начала очереди, не делая сообщения невидимыми. Эта операция возвратит ID сообщения для каждого возвращенного сообщения.
- *ClearQueue* – Удалить все сообщения из заданной очереди. Заметьте, что вызывающая сторона должна повторять эту операцию до

тех пор, пока получает сообщения об успешном ее выполнении, это обеспечит удаление всех сообщений очереди.

Операции уровня сообщения могут быть выполнены с использованием следующего *URL*:

http://<учетнаязапись>.queue.core.windows.net/<ИмяОчереди>/messages

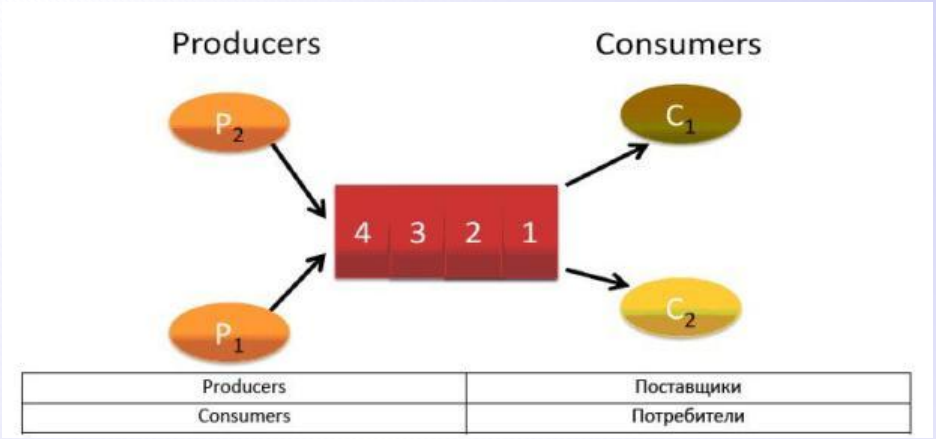
Полное описание *API* REST можно найти в документе *Windows* Azure *SDK*.

8.3 Пример

На рисунке 8.5 представлен пример, иллюстрирующий семантику *Windows* Azure Queue.

В этом примере поставщики (P1 и P2) и потребители (C1 и C2) обмениваются информацией через представленную выше *очередь*. Поставщики формируют рабочие элементы и помещают их в виде сообщений в *очередь*. Потребители изымают сообщения/рабочие элементы из очереди и обрабатывают их. Может существовать множество поставщиков и множество потребителей. Рассмотрим последовательность действий:

1. C1 извлекает сообщение из очереди. Эта операция возвращает сообщение 1 и делает его невидимым в очереди на 30 секунд (принимаем в данном примере, что используется VisibilityTimeout по умолчанию, что составляет 30 секунд).



Producers	Поставщики
Consumers	Потребители

Рисунок 8.5: Пример использования очереди

2. Затем появляется C2 и извлекает из очереди другое сообщение. Поскольку сообщение 1 по-прежнему невидимое, эта операция не увидит сообщения 1 и возвратит C2 сообщение 2.

3. По завершении обработки сообщения 2 C2 вызывает Delete, чтобы удалить сообщение 2 из очереди.

4. Теперь представим, что C1 дает сбой в ходе обработки сообщения 1, таким образом, C1 не удаляет это сообщение из очереди.

5. По истечении времени VisibilityTimeout для сообщения 1, оно опять появляется в очереди.

6. После того, как сообщение 1 повторно появилось в очереди, оно может быть извлечено последующим вызовом от C2. Тогда C2 полностью обработает сообщение 1 и удалит его из очереди.

Как показано в этом примере, *семантика API* очереди гарантирует каждому сообщению в очереди шанс быть обработанным полностью, *по* крайней мере, один раз. То есть, если возникает сбой потребителя в период после извлечения им сообщения из очереди и до его удаления, сообщение опять появится в очереди *по* истечении времени *VisibilityTimeout*. Это обеспечивает возможность этому сообщению быть обработанным полностью другим потребителем.

Рассмотрим REST-запросы, используемые *Windows* Azure Queue.

Ниже показан пример REST-запроса для *операции* постановки в *очередь*. Заметьте, что используется *HTTP-команда* PUT. Задается необязательная *опция "messagettl"*, определяющая срок жизни сообщения в секундах. Максимальный срок жизни – 7 дней. Если этот *параметр* опущен, *значение по* умолчанию – 7 дней. *По* истечении срока жизни сообщение будет удалено системой в процессе сборки мусора. *Параметр* Content-*MD5* может быть задан для защиты от ошибок передачи *по* сети и обеспечения целостности. В данном случае, Content-*MD5* – это *контрольная сумма MD5* данных сообщения в запросе. *Параметр* Content-Length (*Длина* содержимого) определяет размер содержимого сообщения. Также в заголовке *HTTP*-запроса имеется заголовок авторизации. Обратите внимание, что данные сообщения располагаются в теле *HTTP*-запроса. Сообщение может храниться в текстовом или двоичном формате, но при извлечении оно возвращается base64-кодированным.

PUT http://sally.queue.core.windows.net/myqueue/messages

? messagettl=3600
HTTP/1.1 Content-Length: 3900
Content-MD5: HUXZLQLMuI/KZ5KDcJPcOA==
Authorization: SharedKey sally:
F5a+dUDvef+PfMb4T8Rc2jHcwfK58KecSZY+l2naIao=
x-ms-date: Mon, 27 Oct 2008 17:00:25 GMT
Message Data Contents . . . . . .

Ниже показан пример REST-запроса для *операции* извлечения из очереди. Заметьте, что используется *HTTP-команда GET*. Заданы два необязательных параметра. *"numofmessages"* определяет, сколько сообщений должно быть изъято из очереди; максимальное число – 32. По умолчанию извлекается одно сообщение. В примере ниже будет извлекаться *по* 2 сообщения. *Параметр* "visibilitytimeout"определяет *время ожидания* видимости; сообщение будет оставаться невидимым в очереди, в течение этого промежутка времени, в секундах, и вновь появится в очереди, если не будет удалено до завершения периода ожидания видимости. Максимальное *значение* этого времени ожидания – 2 часа, и *значение по* умолчанию – 30 секунд. В примере ниже *время ожидания* видимости задано равным 60 секундам. Также в заголовке *HTTP*-запроса имеется элемент авторизации. Обратите внимание, что ответ поступает в *XML*-формате, и данные сообщения в ответе будут base64-кодированными (в примере ниже располагаются между тегами <MessageText> </MessageText>).

```
GET http://sally.queue.core.windows.net/myqueue/messages
?numofmessages=2 & visibilitytimeout=60
HTTP/1.1
Authorization: SharedKey sally:
QrmowAF72IsFEs0GaNCtRU143JpkflIgRTcOdKZaYxw=
x-ms-date: Thu, 13 Nov 2008 21:37:56 GMT
```

Ответ на этот вызов будет аналогичен получаемому в следующем примере:

```
HTTP/1.1 200 OK
Transfer-Encoding: chunked
Content-Type: application/xml
Server: Queue Service Version 1.0 Microsoft-HTTPAPI/2.0
x-ms-request-id: 22fd6f9b-d638-4c30-b686-519af9c3d33d
Date: Thu, 13 Nov 2008 21:37:56 GMT
<?xml version="1.0"encoding="utf-8"?>
<QueueMessagesList>
<QueueMessage>
<MessageId>6012a834-f3cf-410f-bddd-dc29ee36de2a</MessageId>
<InsertionTime>Thu, 13 Nov 2008 21:38:26 GMT</InsertionTime>
<ExpirationTime>Thu, 20 Nov 2008 21:36:40 GMT</ExpirationTime>
<PopReceipt>AAEAAAD////
/AQAAAAAAAAAMAgAAAFxOZXBob3MuUXVldWU
uU2VydmljZS5RdWV1ZU1hbmFnZ
```

XIuWEFDLCBWZXJzaW9uPTYuMC4wLjAsIEN
1bHR1cmU9bmV1dHJhbCwgUHVibGljS2VG9rZW49bnV
sbAUBAAAAVU1p Y3Jvc29mdC5DaXMuU2VydmljZXMuTm
VwaG9zLlF1Z XVlLlNlcnZpY2U
uUXVldWVNYW5hZ2VyLlhBQy5SZWFsUXVld
WVNYW5hZ2VyK1JYVpcHQCAAAAFjxNc2dJZD5rX19CYWN
raW5nRmllbGQgPFZpc2liaWxpdHlTdGFydD5rX19CYWNr
aW5nRmllbGQDAAtTeXN0ZW0uR3VpZA0CAAAABP3///8L
U3lzdGVtLkd1aWQLAAAAAl9hAl9iAl9jAl9kAl9lAl9mA
l9nAl9oAl9pAl9qAl9rAAAAAAAAAAAAAAAAIBwcCAgICAgI
CAjSoEmDP8w9Bvd3cKe
423ipfNapL7xPLSAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAA AAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAA=</PopReceipt>
<TimeNextVisible>Thu, 13 Nov 2008 21:38:26 GMT</TimeNextVisible>
<MessageText>.....</MessageText>
</QueueMessage>
<QueueMessage>
<MessageId>2ab3f8e5-b0f1-4641-be26-83514a4ef0a3</MessageId>

<InsertionTime>Thu, 13 Nov 2008 21:38:26 GMT</InsertionTime>
<ExpirationTime>Thu, 20 Nov 2008 21:36:40 GMT</ExpirationTime>
<PopReceipt>AAEAAAD/////
AQAAAAAAAAAMAgAAAFxOZXBob3MuUXVl
dWUuU2VydmljZS5RdWV1ZU1hbmFn
ZXIuWEFDLCBWZXJzaW9uPTYuMC4wLjAsIEN1bHR1cmU
9bmV1dHJhbCwgUHVibGljS2VG9rZW49bnVsbAUBAAAAVU1
pY3Jvc29mdC5DaXMuU2VydmljZXMuTmVwaG9zLlF1ZXVlLlNl
cnZpY2UuUXVldWVNYW5hZ2VyLlhBQy5SZWFsUXVl
dWVNYW5hZ2VyK1JYVpcHQCAAAAFjxNc2dJZD5rX19C
YWNraW5nRmllbGQgPFZpc2liaWxpdHlTdGFydD5rX19CYWN
raW5nRmllbGQDAAtTeXN0ZW0uR3VpZA0CAAAABP3///8L
U3lzdGVtLkd1aWQLAAAAAl9hAl9iAl9jAl9kAl9lAl9m
Al9nAl9oAl9pAl9qAl9rAAAAAAAAAAAAAAAIB
wcCAgICAgICAuX4syrxsEFGviaDUUpO8KNfNapL7xPLSAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAA=</PopReceipt>
<TimeNextVisible>Thu, 13 Nov 2008 21:38:26 GMT</TimeNextVisible>

<MessageText>.....</MessageText>
</QueueMessage>
</QueueMessagesList>

Ниже представлен пример REST-запроса для *операции* удаления сообщения. В этом случае используется *HTTP-команда DELETE.Параметр "popreceipt"* определяет сообщение, которое должно быть удалено. *"popreceipt"* получается в результате предыдущей *операции* извлечения из очереди, как показано в примере ранее.

DELETE /sally/myqueue/messages/
6012a834-f3cf-410f-bddd-dc29ee36de2a?
popreceipt=AAEAAAD%2f%2
f%2f%2f %2fAQAAAAAAAAAMAgAAAFxOZXBob3MuUXVld
WUuU2VydmljZS5RdWV1ZU1hbmFnZXIuWEFDLCBWZXJzaW9
uPTYuMC4wLjAsIEN1bHR1cmU9bmV1dHJhbCwg
UHVibGljSV5VG9rZW49bnVsbAUBAAAAVU1pY3Jvc29mdC5DaXMuU
2VydmljZXMuTmVwaG9zLlF1ZXVlLlNlcnZpY2
UuUXVldWVNYW5hZ2VyLlhBQy5SZWFsUXVldWVNYW5hZ2Vy
KJlY2V
pcHQCAAAAFjxNc2dJZD5rX19CYWNraW5nRmllbG
QgPFZpc2liaWxpdHlTdGFydD5rX19CYWNraW5nRmllbGQDAAtT
eXN0ZW0uR3VpZA0CAAAABP3%2f%2f
%2f8LU3lzdGVtLkd1aWQLAAAAAl9hAl9iAl9jAl9k
Al9lAl9mAl9nAl9oAl9

pAl9qAl9rAAAAAAAAAAAAAAAAIBwcCAgICAgICAjSo
EmDP8w9Bvd3cKe423ipfNapL7xPSAsAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAA%3d& timeout=30
HTTP/1.1
Content-Type: binary/octet-stream
x-ms-date: Thu, 13 Nov 2008 21:37:56 GMT
Authorization: SharedKey sally:
M/N65zg/5hjEuUS1YGCbVDHfGnI7aCAudkuTHpCDvZY=

Краткие итоги: В данной лекции мы ознакомились с двумя абстракциями данных Windows Azure: Windows Azure Blob – абстракция данных, которая обеспечивает хранилище больших элементов данных.

Windows Azure Queue – абстракция данных, которая обеспечивает диспетчеризацию асинхронных заданий.

Ключевые термины: Windows Azure Blob – *абстракция* данных, которая обеспечивает хранилище больших элементов данных.

Windows Azure Queue – *абстракция* данных, которая обеспечивает диспетчеризацию асинхронных заданий для реализации обмена данными между сервисами.

§9. Технологии Microsoft .NET Services.

Цель данной лекции – ознакомиться с технологиями Microsoft *.NET* Services.

.NET Services предоставляет основные стандартные блоки, которые понадобятся при построении приложений в облаке и работающих с облаком для Azure™ Services *Platform*.

Сервисы, собранные под именем *.NET* Services, обеспечивают инфраструктуру облака, которая, в конечном счете, упрощает построение работающих в облаке приложений.

Сегодня *.NET* Services обеспечивают основную функциональность, связанную с возможностями подключения приложений, управления доступом и взаимодействия посредством сообщений на базе рабочего процесса. Со временем они будут предоставлять больший набор функций и среду на базе облака. На данный момент под именем *.NET* Services объединены следующие основные блоки сервисов:

- **Microsoft®.NET Service Bus**: предоставляет сетевую инфраструктуру для соединения приложений через Интернет с использованием разнообразных шаблонов обмена сообщениями способом, обеспечивающим возможность прохождения межсетевых экранов и NAT-устройств без нарушения безопасности, предоставляемой этими устройствами.
- **Microsoft®.NET Access Control Service**: обеспечивает управ-

ление доступом в облаке на основании утверждений. Он включает механизм преобразования утверждений, который объединяется с поставщиками удостоверений, такими как Active Directory и Windows Live ID (WLID). В будущих версиях будет реализована интеграция с любыми поставщиками удостоверений.

- **Microsoft®.NET Workflow Services**: предоставляет инфраструктуру для размещения и управления рабочими процессами (WF), уделяя основное внимание взаимодействию через сообщения посредством .NET Service Bus. Поставляется с новыми действиями WF и инструментами для размещения и управления экземплярами рабочего процесса.

Данные новые сервисы можно рассматривать как *.NET*-инфраструктуру сервисов для облака. Все они доступны через открытые протоколы и стандарты, включая REST, *SOAP*, Atom/AtomPub и WS. Это означает, что разработчики на любой платформе могут интегрироваться с этими сервисами.

Однако, в попытке сделать все максимально привычным для *.NET*-разработчиков, Microsoft также предоставляет *.NET* Services *SDK*, который обеспечивает первоклассные условия для *.NET*-разработчика и скрывает многие сложные моменты работы с сервисами.

.NET Services *SDK* позволяет разработчикам использовать имеющийся *опыт .NET*-разработки, в частности в областях WCF и WF, че-

рез применение новых расширений инфраструктуры *SDK* (например, новых привязок, каналов и действия). *SDK* также включает поддержку инструментов *Visual Studio* для интеграции с порталом Azure™ Services *Portal*. Кроме *.NET* Services *SDK*, сегодня партнеры Microsoft предлагают *Java* и *Ruby SDK* (ссылки можно найти в разделе "*Дополнительные ресурсы* ").

Чтобы начать работу с *.NET* Services, перейдите на портал Azure™ Services *Platform* по адресу http://azure.*com* и щелкните ссылку "*Try* It Now"(Попробуйте сейчас). Вы перейдете на страницу "*Register for Azure Services* "(*Регистрация* для сервисов Azure), представленную на рисунке 9.1. На этой странице даются важные ссылки для скачивания различных *SDK*, доступа к дополнительным ресурсам и перехода на *сайт* Microsoft *Connect*, где можно зарегистрироваться для получения кода приглашения.

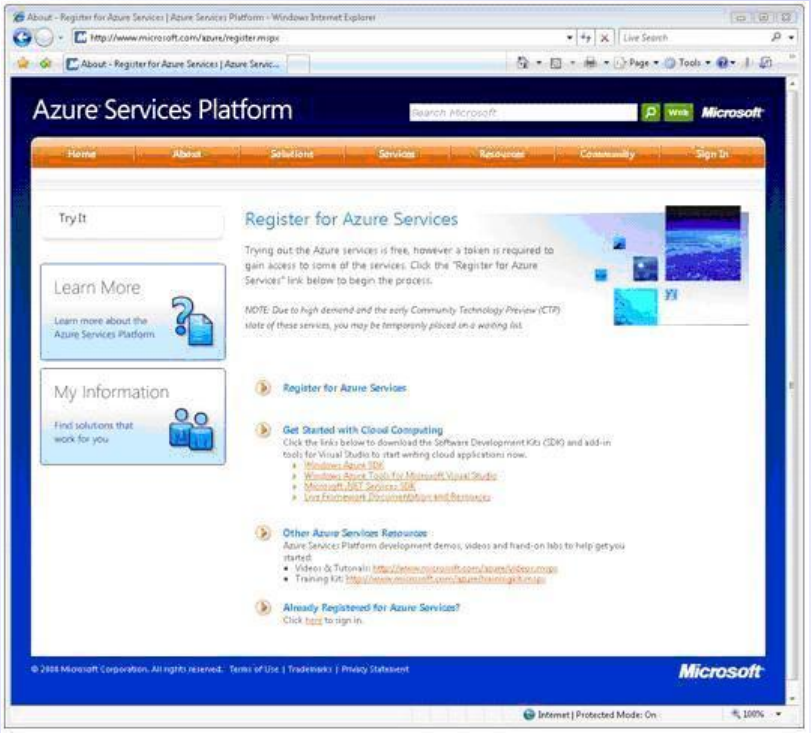


Рисунок 9.1: Портал Azure™ Services Platform

Далее потребуется загрузить *.NET* Services *SDK*. Обратите внимание, что имеется несколько *SDK*: один – специально предназначенный для разработки *Windows*® Azure™; другой – для разработки *.NET* Services; и остальные – для *SQL Data* Services и *Live* Framework. Для воспроизведения примеров, предлагаемых в данной серии документов, понадобится

скачать и установить *.NET* Services *SDK*.

Скачав *.NET* Services *SDK*, просто запустите программу установки, как показано на рисунке 9.2. Тем самым вам будут доступны новые *.NET*-сборки, которые вместе с некоторыми надстройками *Visual* Studio помогут начать использование различных функций *.NET* Services. Приступая к работе с *.NET* Services, обязательно ознакомьтесь с остальными ресурсами, доступными с этой страницы (демонстрации, видео, практические лабораторные и т.д.), цель которых – сделать процесс обучения более насыщенным и разнообразным. Скачать *SDK* можно, не создавая учетной записи, но, чтобы использовать сервисы, необходимо зарегистрироваться.

Рисунок 9.2: Запуск установки NET Services SDK

Чтобы зарегистрироваться на получения учетной записи Azure Services, щелкните показанную выше ссылку "*Register for Services* "(*Регистрация* для сервисов). Вам будет предложено зарегистрироваться, используя *Windows Live ID* (WLID). После этого вы перейдете на *сайт* Microsoft *Connect*, где потребуется заполнить регистрационную форму Azure Services *CTP*. После успешной регистрации на Azure Services *CTP*, на экране появится страница, представленная на рисунке 9.3.

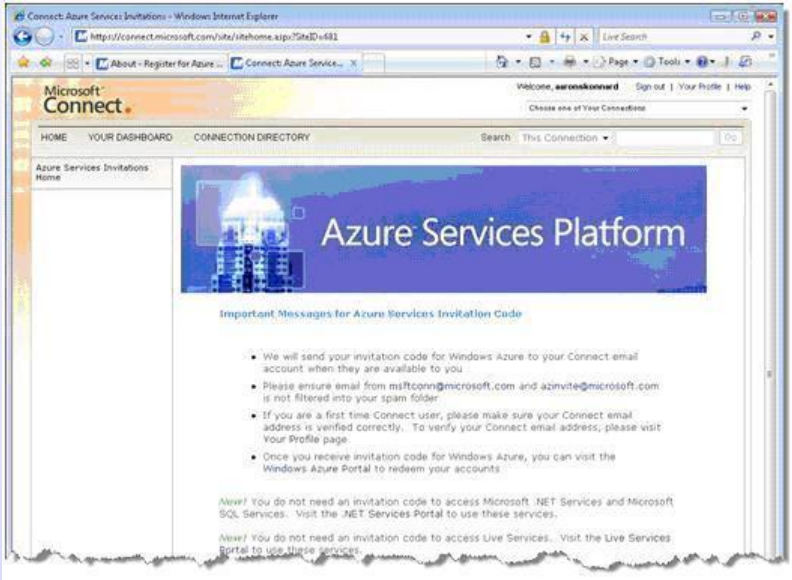


Рисунок 9.3: Регистрация для Azure Services Platform на сайте Microsoft Connect

Теперь можно вернуться на страницу входа *.NET* Services Эту страницу можно увидеть на рисунке 9.4.

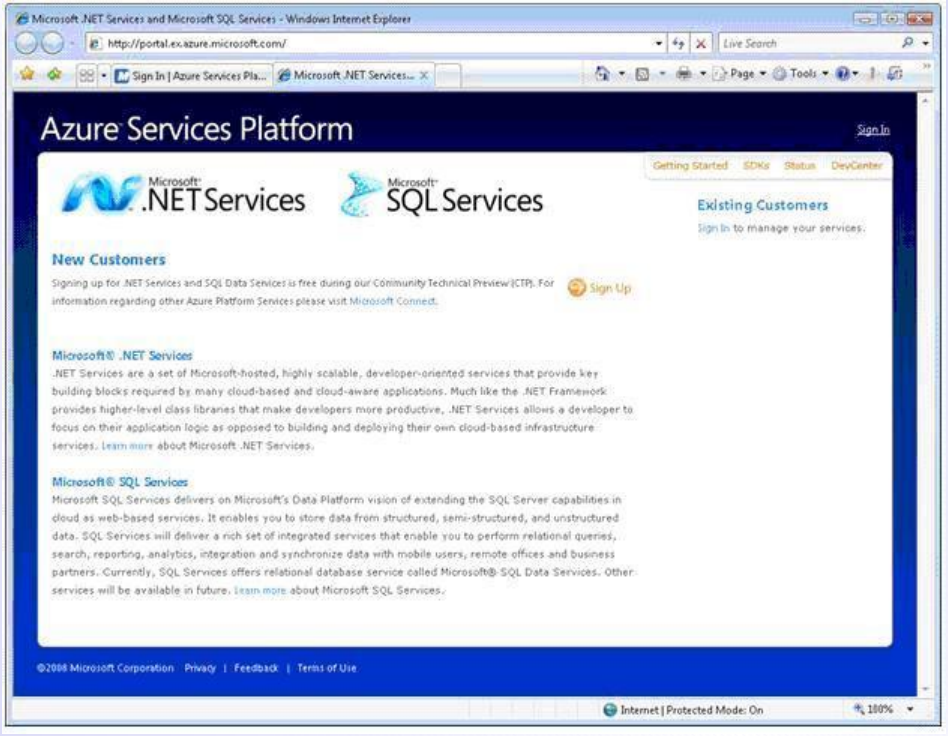


Рисунок 9.4: Страница входа .NET Services

Теперь, щелкнув "*Sign Up*"(Войти), вы получаете возможность создавать решение *.NET* Services. Примечание: в *CTP*-версии, вышедшей

в марте 2009, для создания решения *.NET* Services больше не требуется код приглашения.

Для создания решения необходимо просто ввести уникальное имя решения, принять условия использования и нажать "*Create Solution*"(Создать решение). После этого новое решение будет подготовлено и ассоциировано с вашим WLID. Теперь, в любой момент, зарегистрировавшись на портале Azure™ Services *Platform*, вы имеете возможность управлять решениями, ассоциированными с вашим WLID.

Имя решения должно быть не менее 6 символов длиной и глобально уникальным среди всех пользователей *.NET*-сервисов. Возможно, придется проявить смекалку, чтобы придумать такое имя для решения, которое еще не используется никем другим.

После того, как новое решение создано, на экран выводится страница (рис. 9.5), предлагающая *пароль* решения, который желательно сохранить для использования в будущем. Имя решения и *пароль* выступают в роли учетных данных для доступа к различным сервисам *.NET* Services.

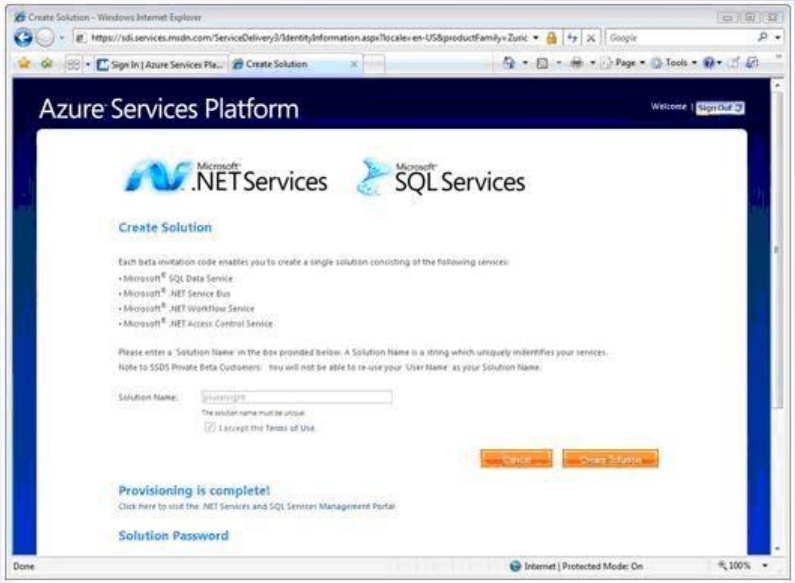


Рисунок 9.5: Завершение процесса подготовки решения

После успешного создания решения можно приступать к работе с ним на портале Azure™ Services *Platform*. Зарегистрировавшись под собственным WLID, на странице портала справа вы увидите *меню "My Solutions"* (Мои решения) (рис. 9.6). Для работы с конкретным решением необходимо просто выбрать его в *меню "My Solutions"*, после чего вам будет представлена страница, показанная на рисунке 9.7.

По сути, *решение* – это *контейнер* верхнего уровня для организации различных ресурсов *.NET* Services.Например, в нем размещаются

конечные точки *.NET Service Bus*, типы и экземпляры рабочих процессов *.NET* Workflow *Service*, а также ваши удостоверения *.NET* Access *Control Service* и правила преобразования утверждений. Но одним из самых важных аспектов, которым вы захотите управлять после создания собственного решения, являются учетные данные решения.

Именно поэтому имя решения должно быть уникальным среди всех пользователей.

Рисунок 9.6: Управление своими решениями посредством меню "My Solutions"

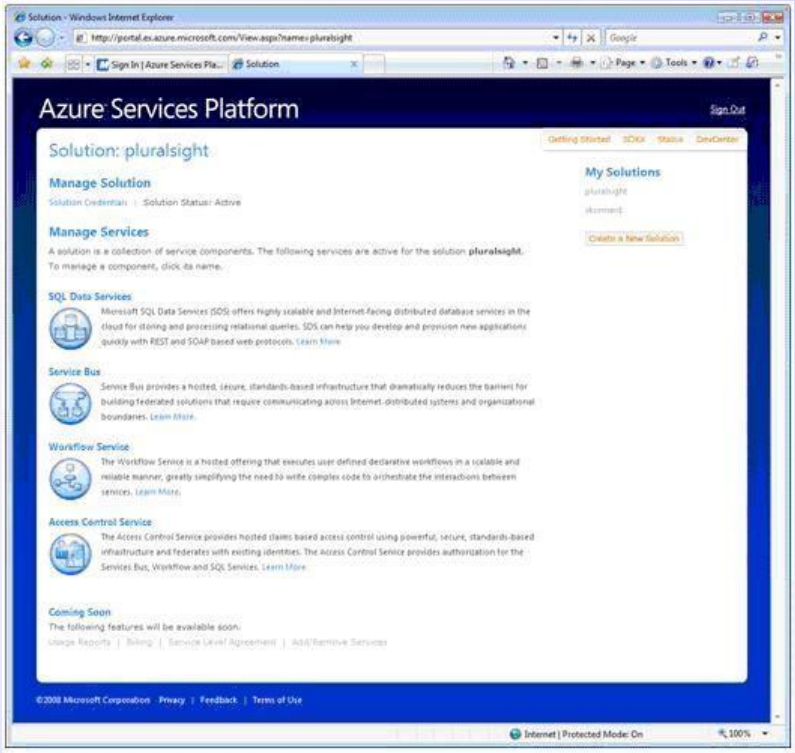


Рисунок 9.7: Управление отдельным решением

Пароль решения, предоставленный в процессе подготовки, можно изменить на странице "*Credential Management*"(Управление учетными данными) (просто щелкните ссылку "*Solution Credentials*"(Учетные данные решения), которую можно видеть на рис. 9.8). С этой страницы можно также конфигурировать информационные карточки *Windows*

CardSpace, ассоциированные с данным решением, и также любые сертификаты, которые необходимо ассоциировать с решением (рис. 9.8).

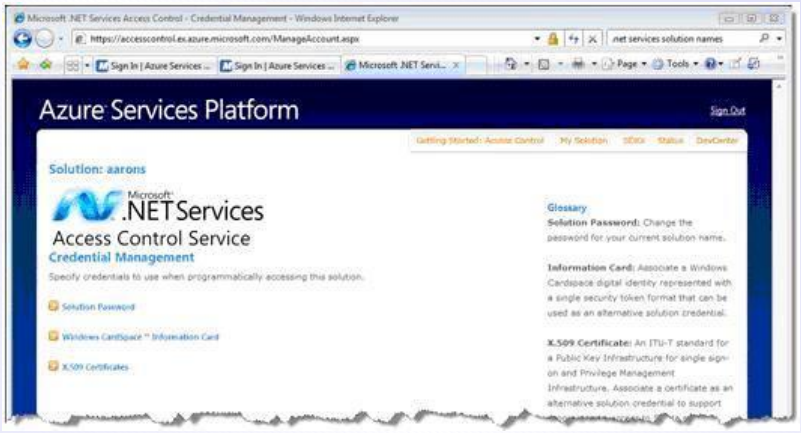


Рисунок 9.8: Управление учетными данными своего решения

Для *Windows CardSpace* и сертификатов вам предложат выбрать необходимую карточку/сертификат, после чего соответствующая *информация* будет передана в учетную *запись* вашего решения. С этого момента в сочетании со своей учетной записью вы можете использовать учетные данные указанной карточки/сертификата.

Самым распространенным требованием в распределенных приложениях с высоким уровнем масштабируемости является возможность подключения приложений. Обычно *интеграция* приложений – одна из са-

мых дорогостоящих и хлопотных областей ИТ. Сегодня для этих задач многие организации используют решение *Enterprise Service Bus*. *Enterprise Service Bus (сервисная шина предприятия, ESB)* – подход к построению распределённых корпоративных информационных систем. Обычно включает в себя промежуточное *ПО*, которое обеспечивает взаимосвязь между различными приложениями *по* различным протоколам взаимодействия.

Архитектура ESB заключается в взаимодействии всех приложений через единую точку, которая, при необходимости, обеспечивает транзакции, *преобразование данных*, сохранность обращений. Данный подход обеспечивает большую гибкость, простоту масштабирования и переноса: при замене одного приложения подключенного к шине нет необходимости перенастраивать остальные.

Одним из стандартов взаимодействия являются веб-сервисы. В популярных реализациях *ESB* добавляются шлюзы для обмена данными с корпоративным *ПО*. С использованием *ESB* может быть реализована сервисно-ориентированная *архитектура*. Существует некоторое разногласие, что именно считать *ESB* – архитектуру или *программное обеспечение*. Обе точки зрения имеют право на существование.

NET Service Bus является основной частью предложения *.NET* Services. Ее основная задача – сделать *шаблон ESB* реальностью в Интернете в рамках платформы Azure™ Services *Platform*. Предоставляемые *.NET Service Bus* архитектурные характеристики во многом аналогичны пред-

лагаемым типовыми решениями *ESB*, включая идентификацию и *управление доступом, присваивание* имен, реестр сервиса и общую среду обмена сообщениями. Основное отличие в области применения. В случае с *.NET Service Bus* компоненты должны разрабатываться для работы в облаке, в глобальной области Интернета, с обеспечением высокого уровня масштабируемости и интегрируемости. Именно поэтому в прошлом этот предлагаемый сервис назывался Microsoft *Internet Service Bus* (рис. 9.9).

Internet Service Bus позволила бы интегрировать ваш локальный *ESB*-продукт с вашими собственными выполняющимися в облаке сервисами, с различными сторонними сервисами, предоставляемыми Microsoft или другими производителями (такими как предлагаются в рамках платформы Azure™ *Service Platform*) и с различными настольными, RIA[3] и веб-приложениями, которые могут выполняться на вспомогательных площадках вне межсетевого экрана корпорации.

Чтобы это стало реальностью, реализация должна обеспечивать интегрированные решения, основанные на открытых *Интернет*-стандартах, и насыщенную среду обмена сообщениями с возможностью двусторонней связи в *Интернет*.

1. Сервисная шина предприятия.

2. Эта терминология применялась в документации BizTalk Services, но более не является официальным наименованием, используемым Microsoft.

3. RIA = Rich Internet Application (Насыщенное Интернет-приложение).

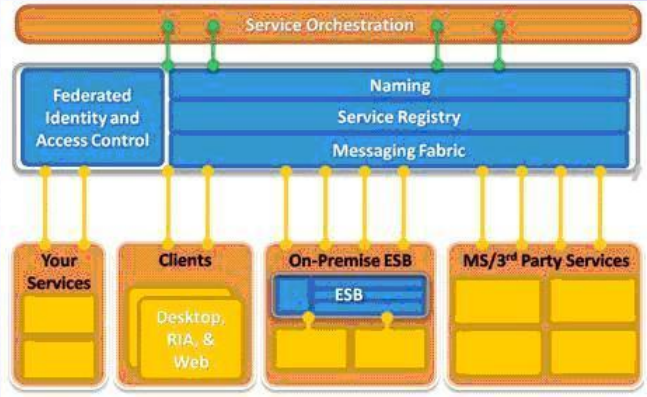


Рисунок 9.9: Сервисная шина Интернет

Реализация двусторонней связи в Интернете не такая уж тривиальная задача из-за некоторых реалий организации современных сетей. Преимущественно, барьеры в сети создают межсетевые экраны и устройства, работающие *по* протоколу *NAT*, которые усложняют *обмен информацией* с узлами, располагающимися за ними. Представьте ситуацию: торговый *агент* использует ваше *приложение по* беспроводной сети в случайной гостинице в некоторой точке земного шара. Как при таком сценарии определить его местоположение и инициировать *связь*?

Часто компании решают эти проблемы связи, открывая входящие порты межсетевых экранов (что доставляет немало хлопот системным администраторам) или используя различных обходные приемы, такие как *динамическая DNS*, сопоставление портов *NAT* или технологию *UpnP*.

Все эти методы неустойчивы, трудно управляемы и восприимчивы к угрозам безопасности. Число приложений, для которых требуется такой тип двусторонней связи, постоянно растет. *.NET Service Bus* призвана удовлетворить эту потребность.

Network address translation – Преобразование сетевых адресов. *Domain Name System* – Служба доменных имен.

Universal Plug and Play – Универсальная автоматическая настройка сетевых устройств.

Решение идентификации, реализацией которой Microsoft занимается последние несколько лет, основано на концепции утверждений. Модель идентификации на базе утверждений позволяет выносить общие функции аутентификации и авторизации из приложений и осуществлять их централизованно во внешних сервисах, написанных и обслуживаемых экспертами безопасности и идентификации, что выгодно всем, кто участвует в этом процессе.

Microsoft® .NET Access Control Service – это сервис в облаке, выполняющий именно эту функцию. Вместо того чтобы создавать собственную базу данных пользовательских учетных записей и ролей, можно предоставить возможность *.NET Access Control Service* управлять аутентификацией и авторизацией ваших пользователей. *.NET Access Control Service* использует существующие хранилища учетных записей пользователей, такие как *Windows Live ID* и *Active Directory*, а также любые другие хранилища, поддерживающие стандартные протоколы интегри-

рования. Таким образом, использование единой регистрации для доступа ко всем приложениям становится вполне естественным. Также это обеспечивает централизацию логики аутентификации и управления доступом, что упрощает ваши приложения.

В поддерживающих утверждения приложениях *пользователь* представляет свое удостоверение как набор утверждений. Одним утверждением может быть *имя пользователя*; другим – его *адрес* электронной почты. Эти утверждения предоставляются организацией, выдающей удостоверения, которая знает, как аутентифицировать пользователя и где найти его атрибуты. Клиентское *приложение*, в роли которого может выступать *браузер* или насыщенный клиент, напрямую получает утверждения от этой организации и передает их в ваше *приложение* (рис. 9.10).

Рисунок 9.10: Использование идентификации на базе утверждений для веб-сервисов

В итоге, *приложение* получает все сведения, необходимые для идентификации пользователя, в виде набора утверждений. Эти утверждения подписываются, что обеспечивает криптографическое подтверждение их происхождения.

Модель идентификации на базе утверждений упрощает реализацию единой регистрации, при этом *приложение больше не отвечает* ни за один из перечисленных ниже аспектов безопасности:

- Аутентификация пользователей
- Хранение учетных записей пользователей и паролей
- Обращение к каталогам предприятия в поисках данных удостоверения пользователя
- Интеграция с системами удостоверений других платформ или компаний

Используя такую модель, *приложение* может принимать решения об идентификации на основании предоставленных пользователем утверждений. И *диапазон* таких решений велик: от простой персонализации приложения *по* имени пользователя до авторизации пользователя для доступа к особо важным функциям и ресурсам приложения.

.NET Access *Control Service* реализовывает идентификацию на базе утверждений в рамках платформы Azure™ Services *Platform*. Система администрирования является важной частью *.NET* Access *Control Service*.

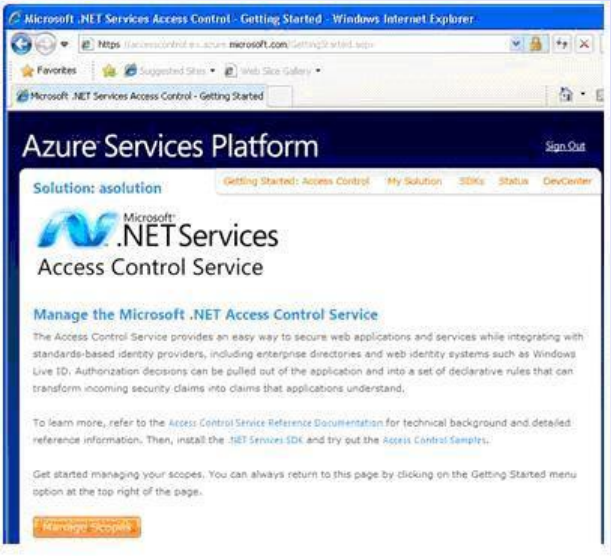


Рисунок 9.11: Портал ACS

.NET Access Control Service предоставляет портал администрирования (рис. 9.11) в рамках портала Azure™ Services *Portal*. Здесь вы выполняете настройку правил, которые определяют схему выпуска утверждений для различных пользователей.

Портал *Access Control Service* – замечательное средство для исследования, изучения и начала работы с *ACS*. И для относительно простых приложений он может быть единственным необходимым инструментом. Но для нетривиальных систем с сотнями или тысячами пользователей

и, возможно, таким же количеством правил, использование портала становится громоздким. В таких случаях программный *интерфейс* – более предпочтительный вариант, поэтому *ACS* также предоставляет *интерфейс AtomPub* для программного администрирования. *AtomPub* – это протокол *RESTful*, который стандартизует базовые *операции CRUD (Create, Retrieve, Update и Delete)* для управления удаленными ресурсами. Это открывает совершенно новые возможности.

Сервис *.NET* Access *Control Service* также включает *конечные* точки *SOAP* и REST для программного администрирования, а также ряд *.NET*-классов, которые упрощают вызов этих конечных точек. Итак, если вам не нравится портал, предоставляемый *ACS*, или вы желаете реализовать настройки, характерные для определенной *предметной области*, можно создать собственную *консоль* администрирования.

Самой большой проблемой в построении крупномасштабных распределенных приложений является принятие решения о моделировании сложных схем взаимодействия через *обмен сообщениями*. Microsoft *.NET* Workflow *Service* позволяет разрабатывать логику взаимодействия сообщений с помощью WF и обеспечивает размещенную масштабируемую среду для выполнения и управления экземплярами рабочего процесса WF в облаке, освобождая разработчика от необходимости создания собственного хоста для WF.

.NET Workflow Service является частью Azure™ Services *Platform* и интегрируется с сервисами *.NET Service Bus* и *.NET* Access *Control*

Service для безопасного координирования взаимодействия посредством сообщений. *.NET* Workflow *Service* также обеспечивает инструменты управления для создания и управления типами и экземплярами рабочих потоков и *API* веб-сервисов для ситуаций, когда требуется создать собственные инструменты.

Поскольку управляющая среда построена на платформе *Windows*® Azure™, она может масштабироваться *по* требованию и в значительной степени, при этом организации или разработчику не приходится беспокоиться о планировании большего количества оборудования или программного обеспечения. Благодаря использованию среды выполнения WF экземпляры рабочего потока могут выполняться в *пуле серверов* и перемещаться с одного сервера на другой в каждом эпизоде выполнения. Управляющая среда включает сервис хранения, который использует безопасные тиражированные сервисы Microsoft *SQL Service* для сохранения состояния выполняющихся рабочих процессов и для обеспечения возможности восстановления.

На период перехода к обработке данных в облаке *.NET* Workflow Services предоставляет упрощенный подход для управления сложными взаимодействиями *.NET Service Bus* в создаваемых вами составных решениях "в облаке".

Построение хоста для рабочих процессов WF означает принятие решений о том, какие возможности будет поддерживать среда и как лучше сделать ее безопасной, масштабируемой и стабильной. Сегодня *.NET*

Workflow *Service* построен на *.NET* Framework 3.5 и действиях и компонентах WF, входящих в данную версию инфраструктуры. Однако для обеспечения наилучших условий Microsoft были добавлены несколько специальных действий и сервисов, которые наложили некоторые ограничения на рабочие процессы, выполняющиеся в облаке.

В облаке не используется сервис хранения *SqlWorkflowPersistenceProvider*, получивший наибольшую популярность среди разработчиков, применяющих WF. Чтобы использовать операционную среду Azure и обеспечить наилучшие возможности масштабирования и стабильности, в инфраструктуре облака имеется специальный *провайдер* услуг хранения, который реализует сохранение состояния выполняющихся рабочих процессов посредством возможностей хранения Microsoft *SQL Services*. Кроме всего прочего, для *Интернет*-сервиса необходима *Интернет*-технология хранения и извлечения данных. Но поскольку механизм WF един – как в облаке, так и в ваших локальных решениях, – применение специального провайдера услуг хранения прозрачно для разработчиков рабочих процессов. Все делается так же, как в любой другое среде WF.

При построении рабочих процессов для облака разработчики используют привычные инструменты *Visual* Studio, включая тот же дизайнер рабочего процесса для создания XAML-файлов рабочих процессов и файлов правил. Затем эти XML-файлы загружаются на *сервер* в облаке, где они могут использоваться для создания экземпляров рабо-

чего процесса. *.NET* Services *SDK* включает *шаблон* проекта для создания *SequentialCloudWorkflow* (Последовательный *рабочий процесс* в облаке), который является специальной версией стандартного шаблона *SequentialWorkflow* (Последовательный *рабочий процесс*). Одним из ограничений текущей инфраструктуры является то, что при определении рабочих процессов, которые будут выполняться в облаке, можно использовать только *подмножество* действий базовой библиотеки действий, а также комплект специальных действий, предоставляемый как часть *.NET* Services *SDK*.

Набор действий требует, чтобы рабочие процессы были полностью декларативными и ограничивающими. Это предотвращает введение пользовательского кода, т.е. позволяет гарантировать *стабильность* среды. При построении управляющей среды для рабочих процессов, написанных любым количеством разработчиков, разбросанных *по* всему миру, такой уровень контроля является обязательным. Поскольку сегодня для выполнения WF необходимо полное *доверие*, мы не можем обеспечить ограниченный набор функциональности, просто выделив пользовательский код в безопасную изолированную программную среду на серверах. Следует отметить, что со временем доступный сегодня ограниченный набор действий будет расширен для увеличения возможностей рабочего процесса в облаке. *По* мере выхода новых версий *.NET* Framework и *.NET* Workflow *Service* также будет поддерживать их. Кроме того, если понадобятся специальные этапы, возможности локальных рабочих

процессов могут комбинироваться с рабочими процессами в облаке с помощью *.NET Service Bus*.

.NET Services *SDK* содержит новый *шаблон* проекта для построения рабочих процессов в облаке, набором новых действий в облаке и клиентским *API* для удаленного развертывания и управления рабочими процессами, размещаемыми в облаке.

При написании рабочих процессов в облаке необходимо быть аккуратным с используемыми действиями (дизайнер предложит только допустимые действия). Разрешенными являются некоторые основные действия потока управления WF, включая *IfElse* (Если...то), *While*(Пока), *Sequence* (Последовательность), *Suspend* (Приостановить), *Terminate* (Завершить) и *FaultHandler* (Обработчик сбоев). Кроме базовых действий потока управления, в рабочих процесса в облаке могут также использоваться *CancellationHandlerActivity* и *FaultHandlersActivity* для моделирования обработки исключений и логики отмены. Обратите внимание, что эти действия обычно не добавляются в модель напрямую; для этого используется дизайнер составных действий, который вводит их автоматически, когда *представление* переходит к этому действию.

Ни одно другое действие WF или пользовательские действия не могут использоваться. Разрешены к применению только новые специальные действия в облаке, включенные в *.NET* Services *SDK*. Ниже описаны новые специальные действия в облаке, которые поставляются с *.NET* Services *SDK*.

Поскольку основной задачей *.NET* Workflow *Service* является упрощение взаимодействия сообщений, эти действия, главным образом, касаются отправки, получения и обработки сообщений. Отправлять/принимать сообщения можно посредством традиционных HTTP-запросов или через *.NET Service Bus*. Эти действия можно будет найти на панели инструментов *Visual* Studio при использовании шаблона проекта последовательного рабочего процесса в облаке.

Краткие итоги:

Платформа Azure™ Services *Platform* представляет комплексную стратегию, разработанную Microsoft для облегчения разработчикам задач *по* реализации возможностей обработки данных в облаке. Microsoft® *.NET* Services – ключевая составляющая этой платформы, созданная специально, чтобы помочь *.NET*-разработчикам сделать первый шаг. *.NET* Services предлагает ориентированные на работу в облаке стандартные блоки и инфраструктуру для обеспечения возможности подключения приложений, управления доступом, размещения и управления рабочим процессом. Эти стандартные блоки станут основными средствами организации работы "с облаком"для *.NET*-разработчиков на годы вперед. Больше информации о *.NET Service Bus*, *.NET* Access *Control Service* и *.NET* Workflow *Service* можно найти в документах данной серии, посвященных каждой из этих тем в отдельности.

Ключевые термины:

Microsoft® *.NET* **Service Bus** – блок сервисов, который предоставляет сетевую инфраструктуру для соединения приложений через *Интернет* с использованием разнообразных шаблонов обмена сообщениями способом, обеспечивающим возможность прохождения межсетевых экранов и *NAT*-устройств без нарушения безопасности, предоставляемой этими устройствами.

Microsoft® *.NET* **Access Control Service** – блок сервисов, который обеспечивает *управление доступом* в облаке на основании утверждений. Он включает механизм преобразования утверждений, который объединяется с поставщиками удостоверений, такими как *Active Directory* и *Windows Live ID* (WLID). В будущих версиях будет реализована *интеграция* с любыми поставщиками удостоверений.

Microsoft® *.NET* **Workflow Services** – блок сервисов, который предоставляет инфраструктуру для размещения и управления рабочими процессами WF, уделяя основной внимание взаимодействию через сообщения посредством.

§10. Предоставление и использование облачных услуг

Ознакомиться с основными решениями "облачных"сервисов. Понять принципы предоставления и использования "облачных"услуг.

10.1 Office Live Workspace

Ресурсы интернета стали так же доступны, как и *разделы* жесткого диска. *Исключение* составляет, пожалуй, лишь скорость обмена данными. В остальном, сегодня можно смело хранить документы и запускать приложения из интернета. Все упирается не в доступность, а в более осязаемые качества: удобство, *безопасность*, функциональность. Они уже поддаются объективной оценке, чем мы и воспользуемся в данной статье, на примере *Microsoft* Office *Live* Workspace, представленного широкой публике 10 сентября 2008 года.

Данный сервис предназначается, в первую *очередь*, для тех пользователей, кто работает за несколькими компьютерами. Наиболее очевидный способ обмена документами - это использование какого-то компактного носителя, например, flash-карты. Так многие и поступают, если есть домашний и рабочий компьютеры, и ноутбук в придачу. Ситуация усугубляется, если ведется *совместная работа* над проектом. Два студента пишут одну курсовую работу, фотограф и дизайнер готовят презентацию, несколько менеджеров составляют прайс-лист фирмы. Общая *пап-*

ка в локальной сети или пинг-понг по электронной почте – не самое рациональное решение проблемы. Во всех описанных случаях, более оптимальный или, как еще можно выразиться, технологичный вариант – это использование специального сетевого сервиса, который как раз и предназначен, в первую *очередь*, для организации совместного доступа к документам. Главное, иметь *выход* в *интернет*.

Перейдем к непосредственному знакомству с ресурсом.

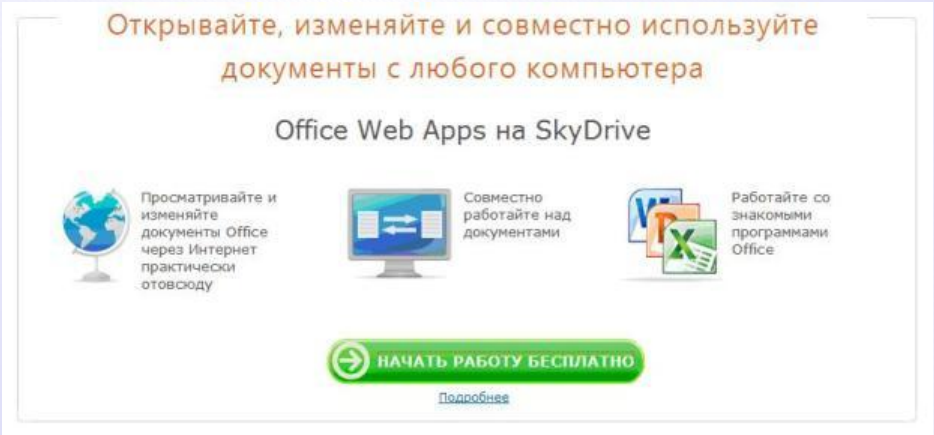


Рисунок 10.1: Стартовая страница Microsoft Office Live Workspace

Первое, с чего стоит начать рассказ о новом сервисе *Microsoft*, – он бесплатен, его использование абсолютно свободно. Впрочем, компания, подарившая миру *Windows*, разумеется, альтруизмом не страдает, и это

нетрудно заметить, если проанализировать остальные свойства сетевого продукта. Начнем с поддержки браузеров. Чтобы можно было использовать бета-версию *Microsoft* Office *Live* Workspace, *компьютер* должен удовлетворять одному из следующих условий:

- Установлен веб-браузер Microsoft Internet Explorer 6, 7 или 8 в операционной системе Microsoft Windows XP, Windows Server 2003 или Windows Vista. Веб-браузер Internet Explorer можно загрузить с веб-страницы Internet Explorer.
- Установлен веб-браузер Mozilla Firefox в операционной системе Windows XP, Windows Server 2003, Windows Vista или Mac OS X 10.2.x (или более поздней версии). Веб-браузер Firefox можно загрузить с веб-страницы загрузки Firefox.
- Установлен веб-браузер Safari 3 или 4 в операционной системе Mac OS X 10.2.x (или более поздней версии).

При использовании других конфигураций, выводится сообщение о не поддерживаемом продукте и рекомендуется загрузить *Internet* Explorer. И еще одно требование – *запуск* сервиса исключительно из *Windows* XP или более поздних версий системы. Здесь картина уже более-менее проясняется. Чтобы воспользоваться преимуществами, даримыми *Microsoft* Office *Live* Workspace, надо, как *минимум*, купить операционную систему от того же производителя. При использовании других операционных

систем и веб-браузеров, возможны проблемы с отображением и функциональностью данного сервиса.

После первого старта сервиса открывается окно приглашения с кратким описанием его возможностей. Перед началом работы можно ознакомиться со списком наиболее часто задаваемых вопросов. Далее можно приступить к работе. Для этого необходимо нажать на зеленую кнопку со стрелкой вправо.

Вход в систему осуществляется через *Windows Live ID*. При создании учетных данных для идентификатора *Windows Live ID* можно использовать любой существующий *адрес* электронной почты любого поставщика услуг электронной почты. Затем эти учетные данные можно использовать для входа на любой веб-*сайт Windows Live ID*. Дополнительную информацию *по Live ID* можно получить на сайте http://www.*microsoft*.com/rus/liveid.

После того, как *регистрация* завершена, на электронный ящик должно прийти *письмо* с подтверждением о регистрации учетной записи.

Перед началом работы с сервисом предлагается установить *дополнение* для *Microsoft* Office версий XP, 2003 или 2007. *Дополнение* поставляется в виде исполняемого модуля, размером около 800КБ. При запуске дополнения, установка проходит автоматическом режиме. Отказ от использования дополнения для офисного пакета означает потерю части интеграции с сервисом. После установки дополнения, в главном *меню* офисного пакета появятся два новых пункта. Один из них позво-

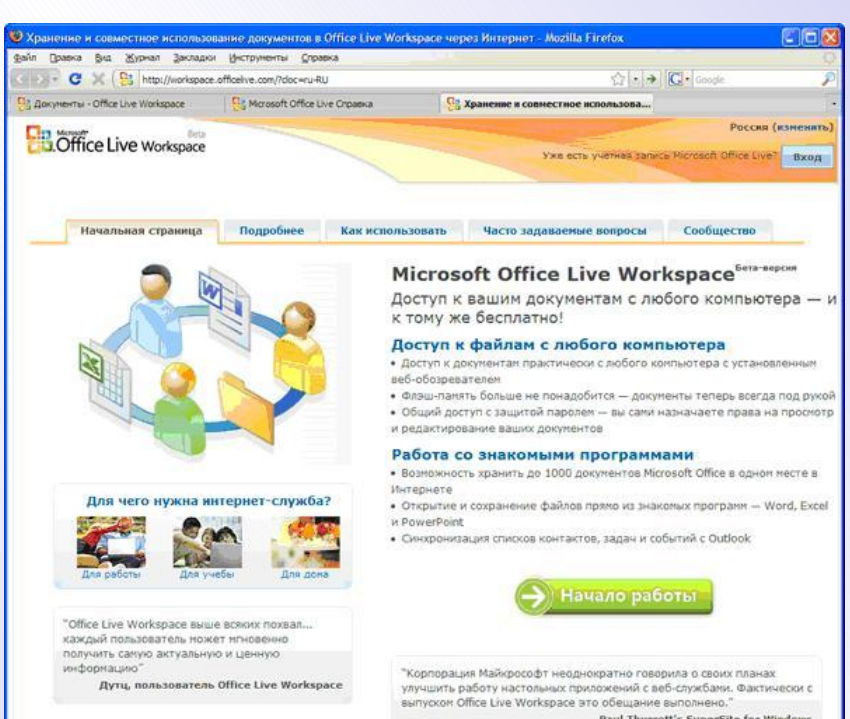


Рисунок 10.2: Вход в Live Workspace

ляет загружать документы с сервиса, а другой – напротив, сохранять

их туда. Если ранее не была произведена *авторизация*, то сначала будет предложено войти в *Microsoft* Office *Live* Workspace. Перейдем к непосредственному знакомству с его главной страницей.

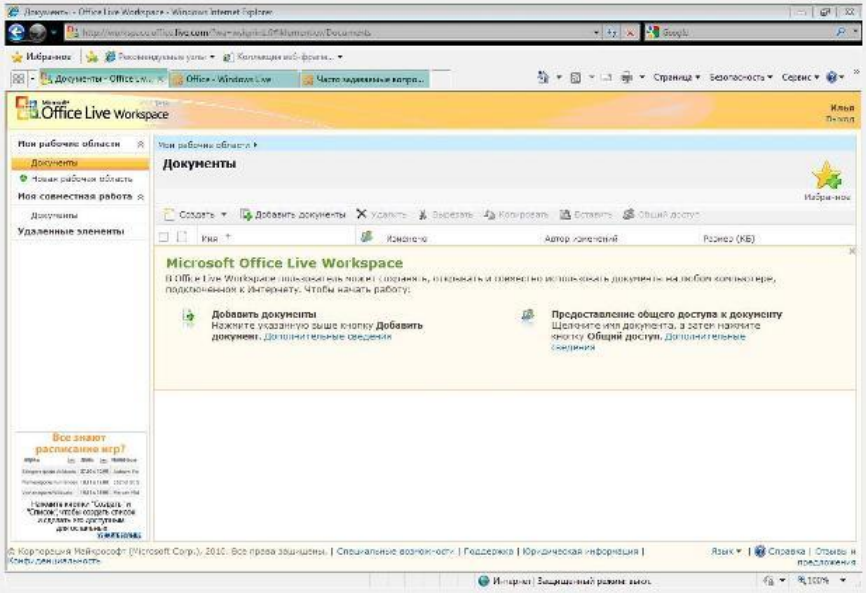


Рисунок 10.3: Главная страница Microsoft Office Live Workspace

Загрузка страницы происходит относительно быстро. В левой боковой панели отображается *список* рабочих областей, совместных и удаленных документов. В нижней части панели всегда находится полезный совет *по* работе с сервисом. Если навести *указатель* мыши на любую строку рабочей области, то через секунду появляется всплывающее окно с выбором доступных операций.

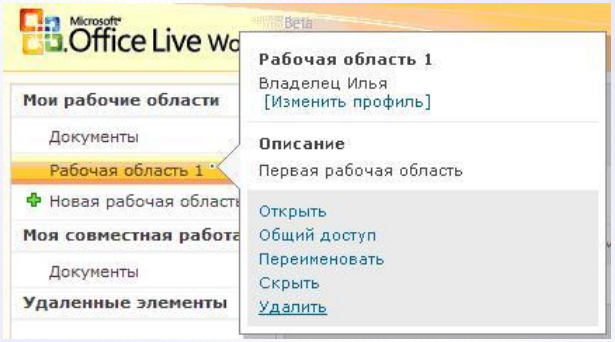


Рисунок 10.4: Высплывающее меню рабочей области

По каждому проекту выводится имя, владелец, текстовое описание и *список* действий, наподобие открытия или переименования.

Основная часть страницы сервиса отводится под *список* документов текущего проекта. Вы видите их названия, имя автора, дату последнего изменения, а также размер. Значок в левой части списка говорит о типе файла.

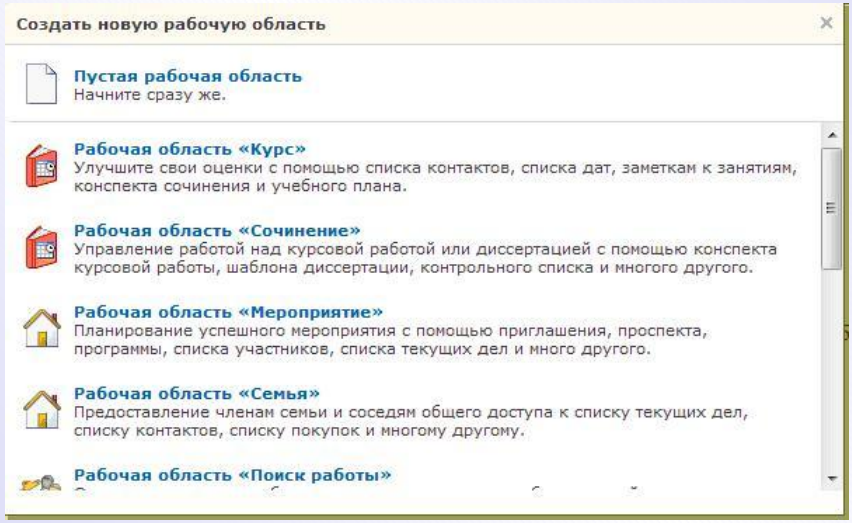


Рисунок 10.5: Создание нового проекта в Microsoft Office Live Workspace

Первый этап работы с сервисом – создание собственного проекта. Он может состоять из нескольких документов, которым, в свою *очередь*,

не обязательно быть однотипными. Новая рабочая область может быть пустой, а также иметь вид некого тематического шаблона.

Предлагается большое число готовых шаблонов. Здесь разработчики постарались учесть большинство потребностей потенциальных пользователей и, как *минимум*, упростить начало работы над проектами.

После создания новой рабочей области, на главной странице отображается некий *список* документов. Авторство пока целиком и полностью принадлежит вам. Вы можете изменить имя и добавить текстовое описание проекту. Сделать это очень просто. Поля свободно редактируются, если щелкнуть *по* ним мышью. Можно создавать новый документ или загрузить *файл* с локального диска.

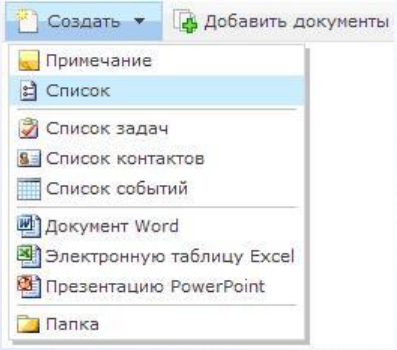


Рисунок 10.6: Диалог создания объектов

Теперь можно перейти к непосредственной работе с документами.

Как было сказано ранее, используя *Internet* Explorer, вы можете загружать файлы с локального диска, просматривая их средствами online-сервиса.

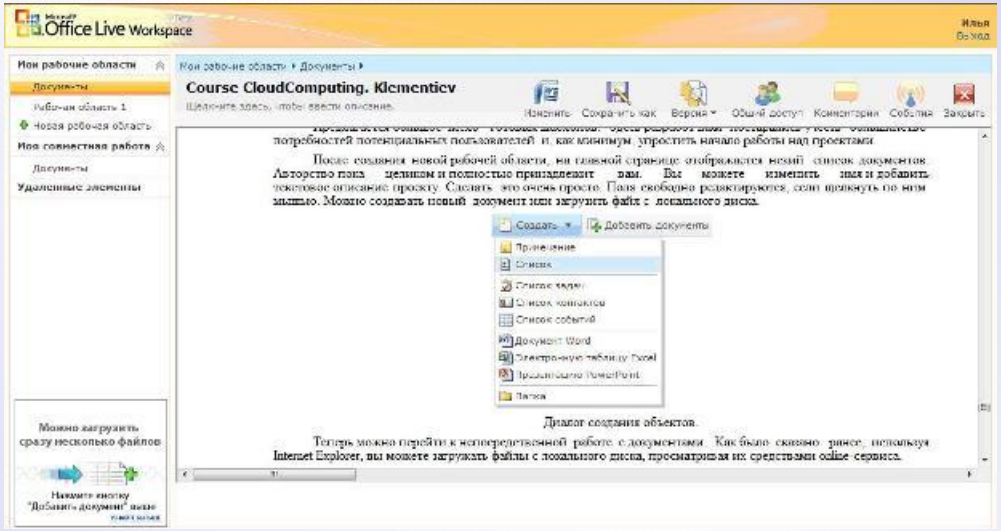


Рисунок 10.7: Открытие документа в Microsoft Office Live Workspace

Вы можете свободно просматривать офисные документы, используя *Microsoft* Office *Live* Workspace, однако их редактирование из браузера невозможно. Редактирование выполняется в приложениях *Microsoft* Office, установленного на локальной машине.

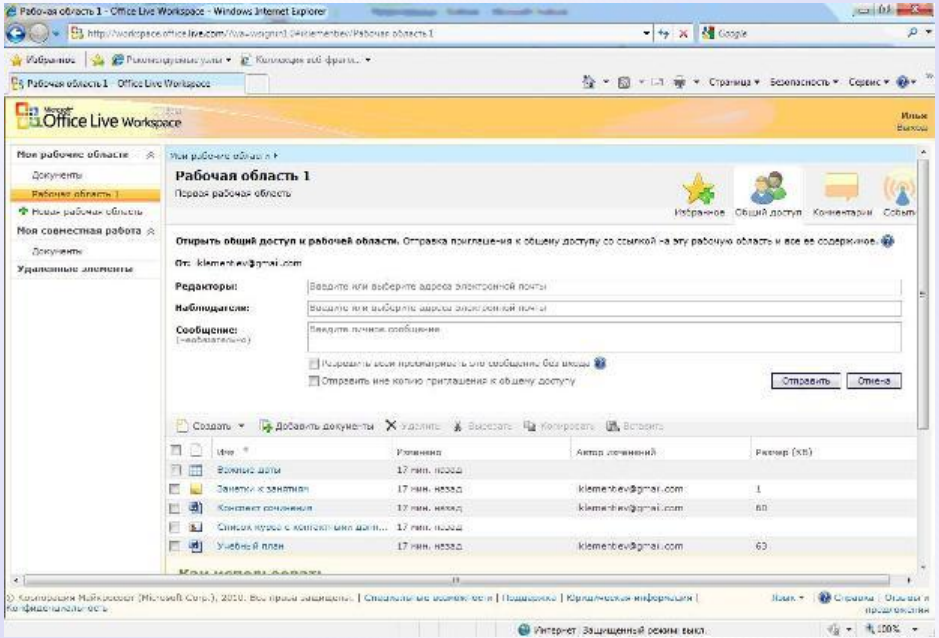


Рисунок 10.8: Управление общим доступом к документам Microsoft Office Live Workspace

Управление общим доступом осуществляется с помощью одноименной кнопки, расположенной на панели инструментов справа от имени

рабочей области и ее описания. Вы можете добавлять пользователей в качестве наблюдателей и редакторов. В первом случае открывается *доступ* только на чтение, во втором - имеется возможность редактирования проектов.

Пользователи выбираются из адресной книги *Windows Live*. Также, Вы можете вводить их электронные адреса вручную. Далее вводится текст сообщения, который будет разослан адресатам. Всем участникам приходит уведомление, в том числе и Вам, как администратору, приходит копия письма. Не только разрешение, но и запрет доступа производится очень просто. Достаточно нажать на крестики напротив нежелательных пользователей или просто воспользоваться кнопкой отмены общего доступа.

Каждый *пользователь* может оставлять свои комментарии к рабочим областям. Комментарии можно создавать на правой боковой панели, либо в развернуто виде (см. рисунок 10.9) Напротив каждого комментария отображается имя автора, дата/время создания и, непосредственно, текст.

Вы можете создавать несколько версий одного документа для обеспечения возможности быстрого возврата к предыдущим версиям. Версионность особенно проявляется при многопользовательской работе. Работа с версиями не поддерживается при работе со списками - специальным типом документов *Microsoft* Office *Live* Workspace.

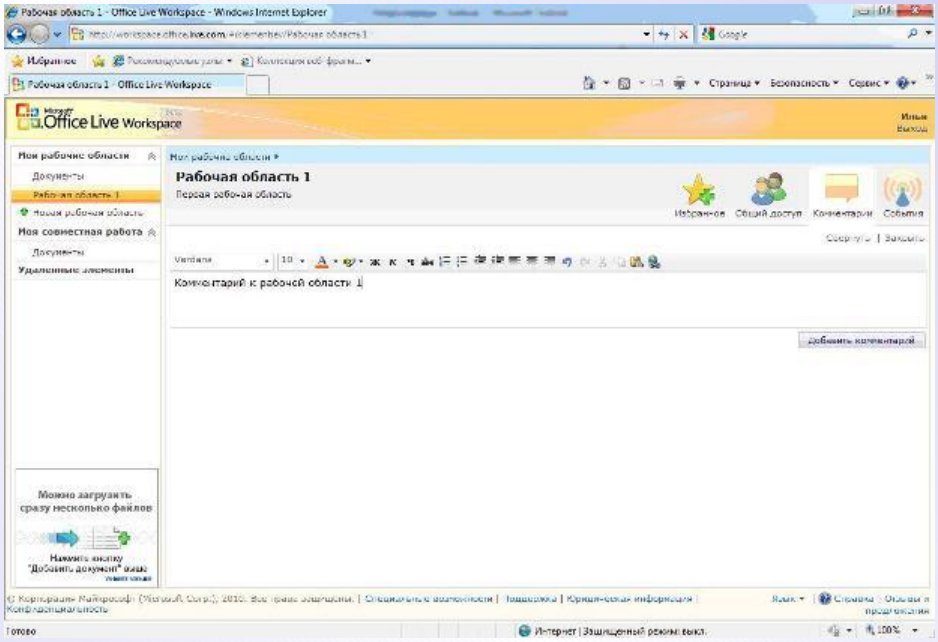


Рисунок 10.9: Добавление пользователями заметок к рабочим областям Microsoft Office Live Workspace

Правая боковая панель может использоваться также для отображения журнала действий пользователей, работающих над одним проектом в *Microsoft* Office *Live* Workspace. *Список* можно разворачивать на всю web-страницу. Ссылки на документы внутри списка обладают интерактивностью. Остальные пункты носят исключительно информационный смысл. Выполнить *откат* на какое-либо действие нельзя.

Рисунок 10.10: Управление версиями документа в Microsoft Office Live Workspace

Как было сказано ранее, online-сервис тесно интегрирован с пакетом *Microsoft* Office. Подразумевается, что *пользователь* будет работать над документом с помощью локального приложения. Однако сервис все же имеет некоторые средства создания и редактирования файлов, не выходя за его пределы, не пользуясь сторонними инструментами. Предлагается два типа документов, которые можно редактировать прямо в браузере. Первый их них основывается на списках.

Списки используются как основа для нескольких типов документов. Их в будущем можно экспортировать в *Outlook* или в *Excel* в зависимости от конкретного содержания. Вы создаете таблицу, для которой добавляются строки и столбцы. Столбцы имеют один из пяти возможных типов: строка, текст из нескольких строк, число, да/нет и дата. В зависимости от выбранного варианта, меняется метод ввода информации, а также внешний вид ячейки. Например, дата указывается с по-

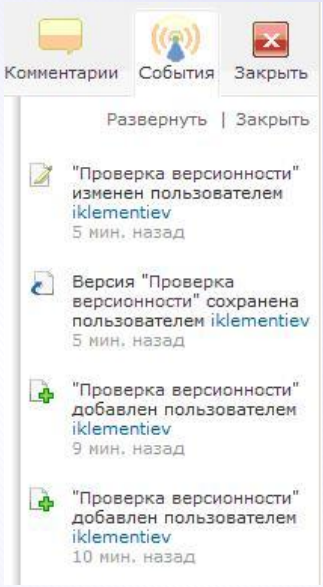


Рисунок 10.11: Список действий пользователей, работающих над одним проектом в Microsoft Office Live Workspace

мощью календаря. Если вы попробуете ввести буквы в числовое *поле*, будет выведено *сообщение об ошибке*. Вычислений между ячейками не предусмотрено. Большинство шаблонов содержат несколько списков, например, *расписание проекта, список* участников, важные даты и многое другое.

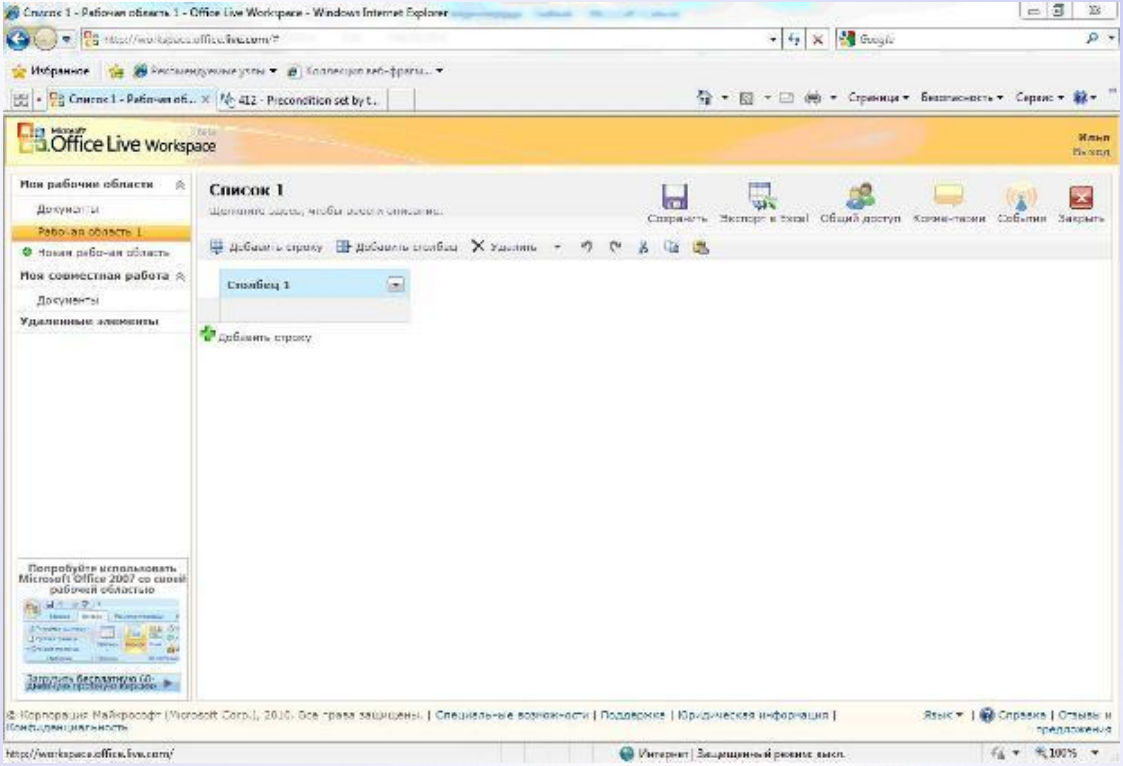


Рисунок 10.12: Работа со списками в Microsoft Office Live Workspace

Второй инструмент предназначается для ведения заметок. Редактор поддерживает сложное *форматирование* текста. Вы можете менять *шрифт*, его стиль, размер. Имеется возможность выделения текста и фона разными цветами. Допускается добавление ссылок в документы.

В заметках можно применять нумерованные и маркированные списки. Имеется возможность форматирования текста по левому и правому краю, а также по центру.

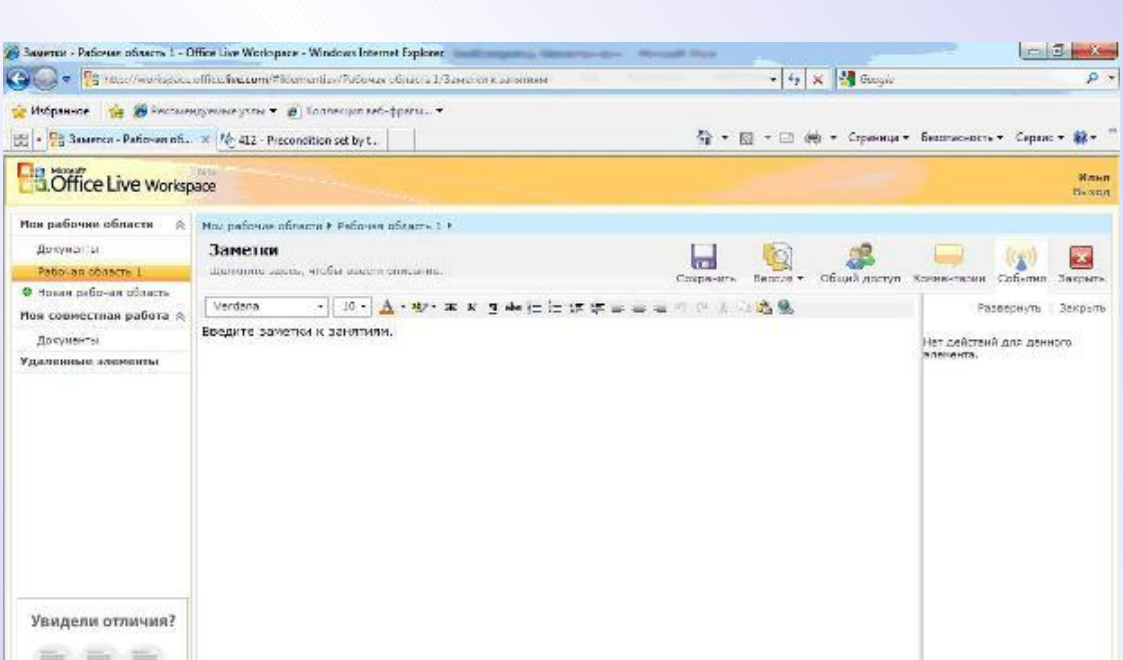


Рисунок 10.13: Работа с заметками в Microsoft Office Live Workspace

На этом возможности встроенного в *Microsoft* Office *Live* Workspace текстового редактора заканчиваются. Отсутствует *поддержка* таблиц, проверки правописания, прочих сервисных функций.

Microsoft Office *Live* Workspace стоит воспринимать как *дополнение* функциональных возможностей *Microsoft* Office, связанное с обеспечением совместной работы над документами.

10.2 Web Apps.Microsoft Word Web App

Благодаря приложению Microsoft Word Web App функции Microsoft Word теперь доступны в браузере, что позволяет работать с документами прямо на веб-сайте, на котором они хранятся. Приложение Word Web App входит в состав набора Office Web Apps, который доступен через службу Windows Live, а также в организациях, в которых приложения Office Web Apps настроены на сервере SharePoint 2010.

Предположим, вы создали документ в приложении Word и хотите опубликовать его на веб-сайте, чтобы другие пользователи могли просматривать его или даже выводить на печать. Кроме того, после публикации может выясниться, что в тексте есть опечатка, которую требуется исправить. Было бы здорово, если бы это можно было сделать непосредственно на веб-сайте. Все эти возможности теперь доступны благодаря приложению Word Web App .

Чтобы приступить к использованию приложения Word Web App из

Microsoft Word 2010, нужно просто сохранить документ в службе *Windows Live OneDrive* или в библиотеке SharePoint. Для этого откройте вкладку Файл и выберите команду *Сохранить и отправить*, а затем – Сохранить на веб-сайте или *Сохранить в SharePoint*.

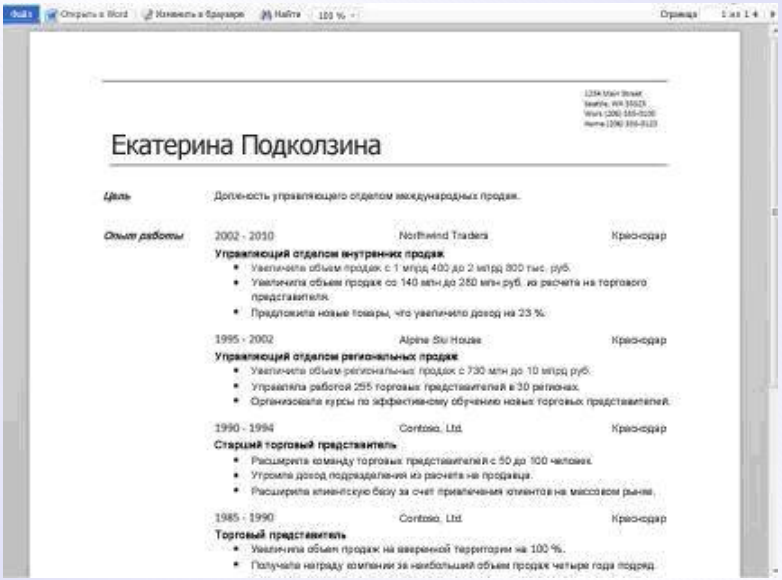


Рисунок 10.14: Microsoft Word Web App

Теперь документ можно быстро просматривать и выводить на печать, редактировать в браузере и открывать в приложении Word Приложение Word Web App позволяет работать с документами, созданными в более

ранних версиях Word. Для достижения наилучшей совместимости загрузите пакет обеспечения совместимости для форматов файлов Microsoft Office Word, Excel и PowerPoint 2007, а затем сохраните необходимые файлы в формате Office Open XML. Сохраните или отправьте документ в библиотеку SharePoint 2010 либо в службу OneDrive.

Документы, хранящиеся в службе Windows Live или на сервере SharePoint, открываются приложением Word Web App непосредственно в браузере. Их структура и формат выглядят так же, как в режиме разметки в версии Word для настольных компьютеров.

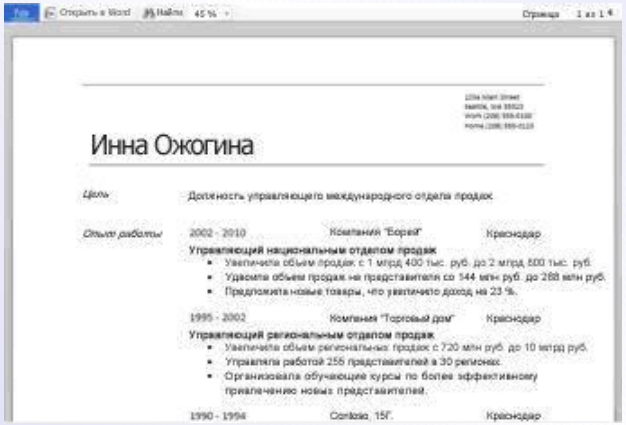


Рисунок 10.15: Microsoft Word Web App

Если на компьютере не установлены компоненты Microsoft Silverlight 2 или более поздней версии, приложение Word Web App отображает

строку со ссылкой, позволяющей загрузить и установить Silverlight. Наличие компонентов Silverlight не является обязательным условием, однако если на компьютере установлена последняя версия платформы Silverlight, документ отображается быстрее, а при увеличении выглядит более качественно.

В режиме чтения можно просматривать документ постранично и переходить к нужным страницам. Для этого введите номер страницы или воспользуйтесь кнопками *Предыдущая страница* и *Следующая страница*.

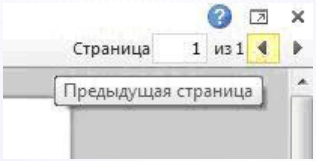


Рисунок 10.16: Microsoft Word Web App в режиме чтения

Чтобы приблизить документ или просмотреть на экране всю страницу, воспользуйтесь командой *Масштаб*, которая позволяет увеличить или уменьшить представление.

Кроме того, с помощью команды *Найти* можно искать в документах слова и фразы. Результаты поиска при этом выделяются.

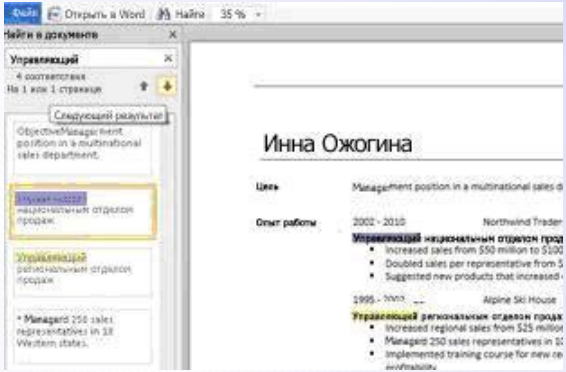


Рисунок 10.17: Поиск в Microsoft Word Web App

Как и при работе с веб-страницей, текст в документе можно выделить, а затем скопировать и вставить в другое приложение.

Приложение Word Web App выводит документ на печать в том виде, какой он имеет в режиме чтения. Чтобы напечатать документ, в режиме чтения откройте вкладку *Файл* и выберите команду *Печать*. Для вывода данных на печать из приложения Word Web App необходимо средство просмотра PDF-файлов. Если оно не установлено, будет предложено загрузить его.

Чтобы внести в документ изменения, нажмите кнопку *Изменить в браузере*. В режиме правки можно добавлять и удалять содержимое, а также форматировать текст. Режим правки оптимизирован для редактирования содержимого, а не для его просмотра. В этом режиме разметка упрощена, а элементы, которые нельзя отобразить, показываются

в виде заполнителей. Заполнители помогают избежать случайного удаления содержимого документа, которое можно отобразить, но нельзя изменять в приложении Word Web App .

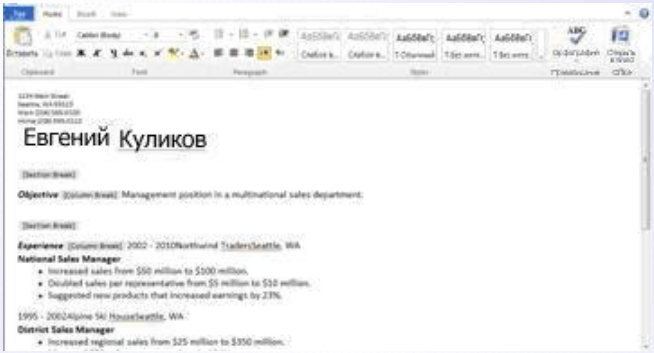


Рисунок 10.18: Правка текста в Microsoft Word Web App

В режиме правки текст можно вводить и форматировать обычным образом, в том числе с помощью команд копирования, вставки, отмены и повтора. Отформатировать текст можно с помощью стилей и команд форматирования, доступных на вкладке *Главная*.

Кроме того, в документ можно добавлять изображения, картинки, таблицы и гиперссылки. Картинки вставляются из коллекции изображений, доступных на веб-сайте Office.com. Все эти объекты можно добавить на вкладке *Вставка*.

Чтобы увидеть, как будут выглядеть внесенные в документ изменения, откройте вкладку Файл и нажмите кнопку Сохранить, после чего вернитесь в режим чтения (для этого на вкладке Вид нажмите кнопку *Режим чтения*).

Возможности редактирования в Word Web App идеально подходят для внесения быстрых изменений, например исправления ошибок, добавления изображений и текста. Чтобы получить доступ ко всем возможностям Word, откройте вкладку *Файл* и нажмите кнопку *Открыть в Word*.

Word Web App откроет документ непосредственно в приложении Word для настольных компьютеров, в котором доступны дополнительные функции (например, возможности настроить стили документа, изменить объект SmartArt или добавить колонтитулы). При нажатии кнопки *Сохранить* в приложении Word документ будет снова сохранен на веб-сервере. Чтобы можно было использовать функцию "Открыть в Word"приложения Word Web App , на компьютере должна быть установлена программа Word 2003 или более поздней версии (при работе с браузером Internet Explorer) либо Word 2010 (при работе с браузером Firefox).

10.3 Microsoft Excel Web App

Благодаря приложениям Microsoft Excel Web App функции Microsoft Excel теперь доступны в браузере, что позволяет работать с книгами

на том веб-сайте, на котором они хранятся. Приложение Excel Web App входит в состав набора Office Web Apps, который доступен через службу Windows Live, а также в организациях, в которых приложения Office Web Apps настроены на сервере SharePoint 2010.

Предположим, вы создали книгу в приложении Excel и хотите опубликовать ее на веб-сайте, чтобы другие пользователи могли поработать с данными в ней или даже внести в них свои изменения. Возможно, вам потребуется работать с ней совместно с пользователем, на компьютере которого установлена другая версия Excel. Было бы неплохо, если бы можно было все сделать непосредственно на веб-сайте. Теперь это возможно благодаря приложению Excel Web App.

Чтобы приступить к использованию приложения Excel Web App из Microsoft Excel 2010, нужно просто сохранить книгу в службе *Windows Live OneDrive* или библиотеке SharePoint. Для этого откройте вкладку **Файл** и выберите команду **Сохранить и отправить**, а затем – **Сохранить на веб-сайте** или **Сохранить в SharePoint**.

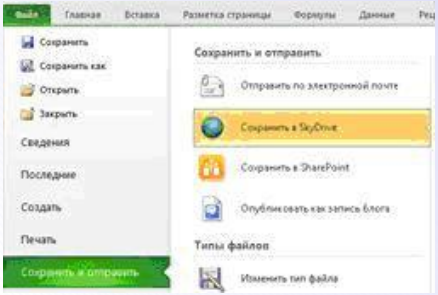


Рисунок 10.19: Меню Файл Microsoft Excel Web App

Теперь книгу можно просматривать и редактировать в браузере или открывать в приложении Excel. Приложение Excel Web App позволяет работать с книгами, созданными в более ранних версиях Excel. Для достижения наилучшей совместимости загрузите пакет обеспечения совместимости для форматов файлов Microsoft Office Word, Excel и PowerPoint 2007, а затем сохраните необходимые файлы в формате Office Open XML. Сохраните или отправьте книгу в библиотеку SharePoint 2010 либо в службу OneDrive.

Книги, хранящиеся в службе Windows Live или на сервере SharePoint, открываются приложением Excel Web App в браузере, в котором можно просматривать их, сортировать и *фильтровать данные*, разворачивать и сворачивать сводные таблицы и даже выполнять перерасчет значений.

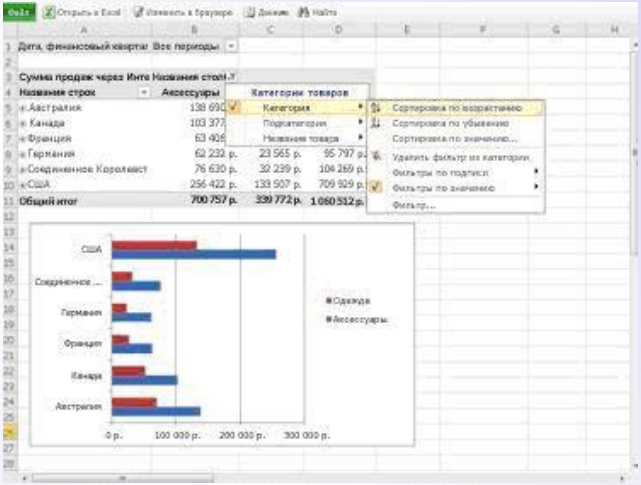


Рисунок 10.20: Microsoft Excel Web App.

Кроме того, с помощью команды **Найти** можно искать в книгах слова и фразы. Как и при работе с веб-страницей, содержимое листа можно выделить, а затем скопировать и вставить в другое приложение.

Чтобы внести в книгу изменения, нажмите кнопку **Изменить в браузере**. После этого можно вводить в книгу данные, добавлять и изменять формулы и применять основные виды форматирования.

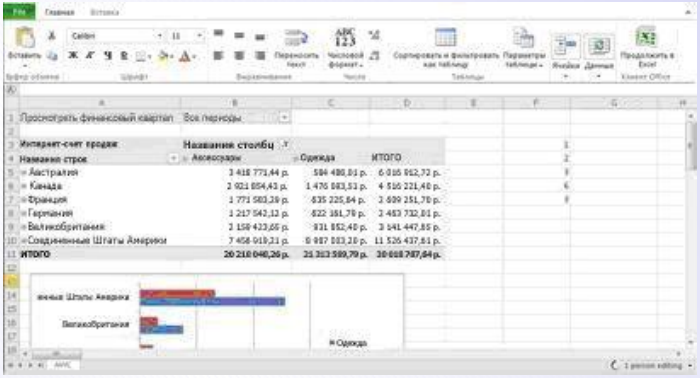


Рисунок 10.21: Меню Файл Microsoft Excel Web App

В режиме правки текст можно вводить и форматировать обычным образом, в том числе с помощью команд вырезания, копирования, вставки, отмены и повтора. Чтобы добавить формулу, перейдите в соответствующую ячейку и введите знак равенства (=), а затем – нужную формулу или функцию.

Кроме того, с помощью вкладки *Вставка* в книгу можно добавлять таблицы и гиперссылки.

Приложение Excel Web App автоматически сохраняет книгу по мере работы с ней, поэтому сохранять изменения вручную не требуется. *Отменить* нежелательные изменения можно с помощью команды Отменить или клавиш CTRL+Z.

Предоставив к книге доступ для редактирования другим пользовате-

лям, над ней можно будет работать одновременно с друзьями и коллегами. Эта возможность пригодится для сбора сведений от группы пользователей, например при составлении списка сведений или разработке группового проекта. Благодаря этому больше не нужно пересылать книгу друг другу по электронной почте или ждать, пока коллега вернет ее на сервер.

В режиме редактирования в приложении Excel Web App отображаются сведения о других пользователях, работающих с этой книгой. Сохраните книгу на веб-сайте, доступном для других пользователей, например в библиотеке SharePoint или папке в службе OneDrive.

В SharePoint скопируйте веб-адрес книги из браузера (выделите его и нажмите клавиши CTRL+C) и вставьте его в сообщение с помощью клавиш CTRL+V.

В службе Windows Live в приложении *Excel Web App* откройте вкладку *Файл* и выберите пункт *Общий доступ*. Добавьте пользователей, которым необходимо предоставить доступ к этой книге, нажмите кнопку *Сохранить* и создайте сообщение.

Откройте книгу для редактирования в приложении Excel Web App. Сведения о других пользователях, работающих с ней, появятся в строке состояния.

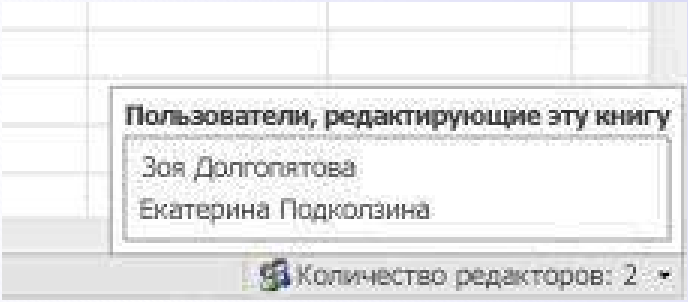


Рисунок 10.22: Сведения о пользователях Microsoft Excel Web App

Возможности редактирования Excel Web App идеально подходят для быстрого внесения изменений и совместной работы. Чтобы получить доступ ко всем возможностям Excel, откройте вкладку Файл и нажмите кнопку *Открыть в Excel*.

Excel Web App откроет книгу непосредственно в приложении Excel для настольных компьютеров, в котором доступны дополнительные функции (например, возможность настроить параметры диаграммы или сводной таблицы). При нажатии кнопки Сохранить в Excel книга будет снова сохранена на веб-сервере. Чтобы можно было использовать функцию "*Открыть в Excel*"приложения Excel Web App, на компьютере должна быть установлена программа Excel 2003 или более поздней версии (при работе с браузером Internet Explorer) либо Excel 2010 (при работе с браузером Firefox).

10.4 OneDrive

OneDrive – это бесплатное интернет-хранилище для файлов, к которому можно получать доступ отовсюду. Сохраните файлы в OneDrive – и они всегда будут с вами. А с приложением для компьютера OneDrive можно автоматически синхронизировать файлы с устройствами.

Возможности

Доступ к файлам на всех устройствах

С OneDrive фотографии, документы и другие важные файлы всегда доступны, где бы вы ни были. Доступны приложения для iPhone, iPad, Android и Windows Phone.

OneDrive и Office – работа в тандеме

Со OneDrive и Office 2013 можно открывать документы Word, Excel и других программ Office, а также работать с ними из OneDrive. Вы можете с легкостью делиться документами со своими друзьями и совместно над ними работать. Завершив работу, сохраните их в OneDrive, и они будут под рукой, где бы вы ни находились.

Если у вас не установлен пакет Office 2013, вы все равно сможете получать доступ к документам Office, а также работать над ними и делиться ими на сайте OneDrive.com прямо в своем браузере, используя Office Web Apps.

Защитите свои файлы с помощью OneDrive

Если ваши файлы сохранены в OneDrive, вы сможете получить к ним

доступ, даже если что-то случится с устройством. Кроме того, вы можете предоставлять доступ к определенным файлам определенным людям.

Приложения, совместимые с OneDrive

OneDrive совместим с отличными приложениями, которыми, возможно, вы уже пользуетесь. Например, вы можете сканировать файлы с помощью телефона, делать заметки в OneNote, сохранять статьи как звуковые файлы и выполнять другие задачи, а затем сохранять файлы прямо в OneDrive.

Обзор OneDrive

Доступ к фотографиям и файлам практически отовсюду

OneDrive предоставляет вам бесплатное хранилище в облаке объемом 7 ГБ для хранения файлов и фотографий, их синхронизации между всеми компьютерами и даже редактирования и совместной работы над документами Office. Вы можете получать доступ к последней версии файлов с любого компьютера или телефона. Никакие кабели или USB-устройства флэш-памяти теперь не нужны.

Делитесь фотографиями и файлами, отправляя их в виде сообщения с короткой ссылкой или размещая в социальных сетях, на своем сайте или в блоге, прямо из OneDrive. А с помощью классического приложения OneDrive, которое создает на компьютере с Windows или Mac OS папку, синхронизируемую с вашей папкой в облаке, вы сможете получать удаленный доступ к файлам на своем компьютере. (Если на компьютере установлена Windows 8.1, вам не потребуется классическое приложение

OneDrive.)

OneDrive в Интернете

Свободу файлам! Храните и просматривайте свои фотографии и файлы, а также делитесь ими с любого устройства на сайте OneDrive.com. Настраивайте внешний вид своего хранилища OneDrive в Интернете. Просматривайте файлы подробным списком или в представлении эскизов, как показано ниже. Переключайте режимы отображения одним нажатием в верхнем правом углу страницы.

Файлы OneDrive упорядочены так, чтобы переходить к ним одним щелчком:

- Чтобы просмотреть все свои файлы и папки OneDrive, нажмите Файлы.
- Чтобы просмотреть недавно просмотренные документы Office, нажмите Последние документы.
- Чтобы просмотреть все свои фотографии, нажмите Все фотографии. Фотографии сортируются по дате, поэтому для их просмотра не нужно переходить в отдельные папки. Это представление подобно фотоальбому со снимками ваших друзей и путешествий.
- Чтобы просмотреть элементы, которыми вы поделились и которыми поделились с вами, нажмите Общие.

OneDrive на компьютере

Когда вы устанавливаете классическое приложение OneDrive на компьютере с Windows или Mac, на нем автоматически создается папка OneDrive. Все файлы или фотографии, сохраненные вами в этой папке, будут автоматически синхронизироваться с другими вашими компьютерами, на которых установлена OneDrive. Вы также можете легко получать доступ к этим файлам на своем телефоне. Кроме того, вы можете просматривать файлы, сохраненные на компьютере, даже если они не добавлены в вашу папку OneDrive. Если на компьютере установлена Windows 8.1, вам не потребуется классическое приложение OneDrive.

Начать – просто:

1. Перетащите файлы в свою папку OneDrive, и они автоматически будут переданы в ваше облачное хранилище. После завершения передачи файлов возле них появится зеленый флажок. Если вы внесете изменения в файл и сохраните его обновленную версию, OneDriveавтоматически обновит файл в вашей папке.

2. С помощью классического приложения вы можете синхронизировать все файлы в OneDrive с файлами на компьютере или выбрать отдельные папки для синхронизации. Чтобы изменить список синхронизированных папок, щелкните правой кнопкой мыши значок рабочего стола OneDrive в области уведомлений справа на панели задач. (Возможно, для отображения значка вначале потребуется нажать кнопку Отображать скрытые значки). Нажмите Параметры, а затем выберите

вкладку Выбор папок.

3. Извлекайте свои файлы, сохраненные на компьютере под управлением Windows (с другого компьютера с установленной Windows), подключенном к Интернету, даже если они не добавлены в ваше хранилище OneDrive. Чтобы активировать функцию удаленного доступа, щелкните правой кнопкой мыши значок рабочего стола OneDrive в области уведомлений справа на панели задач. (Возможно, для отображения значка вначале потребуется нажать кнопку Отображать скрытые значки). Нажмите Параметры и установите флажок Получать удаленный доступ ко всем файлам на этом компьютере через OneDrive.

4. И конечно же, вы всегда сможете получить доступ к своим файлам с любого компьютера на сайте onedrive.live.com.

Обмен файлами и фотографиями

Благодаря OneDrive вы можете легко делиться файлами с определенными людьми. Им не придется устанавливать специальные программы или создавать новые учетные записи. Просмотреть отправленные вами файлы они смогут в любом веб-браузере.

Делитесь файлами любого размера

Зачем ограничиваться размерами вложений в электронной почте? Отправляйте ссылки на файлы, папки или фотоальбомы в OneDrive. **Вот как это делается:**

1. Передайте файлы в OneDrive. (В зависимости от способа отправки, размер передаваемых файлов может достигать 2 ГБ.)

2. Выберите элемент, которым хотите поделиться.

3. Нажмите Поделиться, а затем выберите способ предоставления доступа.

- способом, становятся доступными на веб-сайте OneDrive.com в разделе Общие. Если понадобится, вы сможете отозвать разрешения для отдельных людей в любое время.
- В социальной сети, к которой вы подключены с помощью своей учетной записи OneDrive.
- С помощью ссылки (это удобно, если вы хотите скопировать и вставить ее в мгновенное сообщение или на веб-страницу).

Открыв доступ к своим файлам или папкам, вы можете изменить разрешения для них. Выделите файл или папку и нажмите Поделиться. Вы можете удалить разрешения, чтобы отменить общий доступ к элементу, или, в зависимости от того, как был открыт доступ, изменить возможность редактирования элемента.

Внедряйте файлы непосредственно в свой веб-сайт или блог

Если у вас есть веб-сайт или блог, вы можете легко внедрять в него фотографии, видеозаписи, документы Office и другие файлы из OneDrive. Кроме того, вы можете настроить вид внедренных книг Excel .

1. Выберите файл, который вы хотите внедрить.
2. Нажмите Внедрить, а затем выберите Создать.
3. Скопируйте и вставьте код в код вашего веб-сайта или блога.

OneDrive и Office – работа в тандеме

OneDrive предоставляет вам автоматический доступ к своим документам с любого компьютера с Windows или Mac, планшета iPad или телефона. Кроме того, вы можете делиться файлами и работать с ними одновременно с другими пользователями, независимо от используемого устройства. Каждый может просматривать и редактировать документы с помощью веб-браузера и бесплатных веб-приложений Office Web Apps.

Совместная работа в Интернете без ограничений

OneDrive идеально подходит для предоставления общего доступа и совместной работы с другими людьми. Чтобы поделиться файлами с кем угодно, достаточно перейти на веб-сайт onedrive.live.com. С помощью бесплатных веб-приложений Office Web Apps ваши друзья могут просматривать и редактировать документы, даже если у них не установлен пакет Office.

Кроме того, Office Web Apps сохраняют форматирование документов. А это значит, что ваши файлы останутся такими, какими вы их создали. Кроме того, пользователи, просматривающие документ, смогут вносить изменения, не нарушая форматирования.

Использование всех возможностей Office

Office 2013 подключается к вашей учетной записи Майкрософт (это адрес электронной почты и пароль, используемые для входа в OneDrive), поэтому просматривать, хранить файлы и делиться ими – проще, чем когда либо. Вот как вы можете начать:

- После установки Office 2013 войдите в OneDrive или в любую другую программу Office (Word, Excel, PowerPoint или OneNote). Выберите Войти в правом верхнем углу окна программы, а затем введите адрес электронной почты и пароль учетной записи Майкрософт.
- Сохраняйте свои документы Office прямо в папке OneDrive. Не закрывая документ Office, в меню Файл выберите Сохранить как, а затем – OneDrive, и укажите папку, в которой вы хотите его сохранить.
- Перетащите файлы прямо в папку OneDrive, чтобы сохранить их локально на компьютере. Эта папка не отличается от других папок. Чтобы перейти к ней, откройте проводник Windows и выберите OneDrive.

Вы можете работать с документами, сохранять их делиться ими на различных устройствах с помощью OneDrive в Интернете, на компьютере, а также на телефоне или планшете.

10.5 Office 365

19 октября 2010 года (Редмонд, США) Microsoft анонсировала **Office 365** – пакет услуг нового поколения, объединяющий самые современные решения для эффективной работы и взаимодействия в сфере облачных вычислений: Office, SharePoint, Exchange и Lync. Благодаря Office 365

многие организации во всём мире получат удобный и экономичный доступ к передовым технологиям, автоматически обновляемым по мере выхода новых продуктов и их версий. Кроме того, с сегодняшнего дня 1000 компаний в 13 странах начинают бета-тестирование сервиса Office 365.

Office 365 – это новый этап в реализации облачной стратегии Microsoft. Он заменит сервисы *Business Productivity Online Suite, Office Live Small Business и Live@edu*. При этом переход к Office 365 никаким образом не отразится на стиле работы пользователей, так как облачный сервис обеспечит доступ к нужным программам и файлам из любого места и с любого устройства, имеющего выход в Интернет. Кроме того, он совместим со многими приложениями и браузерами.

"Office 365 – это, пожалуй, лучшее решение для повышения производительности труда, – отметил Павел Кузьменко, руководитель отдела по продвижению информационных офисных систем Microsoft в России. Подобно тому, как Office для многих пользователей в мире определяет стандарты работы за компьютером, Office 365 определит направление развития облачных вычислений в будущем. Новый пакет услуг действительно подходит всем: и маленькой компании, которая приобретает свою первую бизнес-платформу, и международной корпорации, которая стремится к сокращению расходов не в ущерб качеству и производительности. С Office 365 люди смогут полностью сконцентрироваться на своём бизнесе, так как о технологиях позаботимся мы".

Microsoft активно привлекала пользователей по всему миру к процессу разработки Office 365. Именно их отзывы и пожелания помогли создать пакет услуг, который обладает возможностями и функциями, удовлетворяющими потребности самых разных клиентов: от индивидуальных профессионалов до небольших, средних и крупных организаций.

Сегодня организации могут выбирать между стационарным и виртуальным сервером в зависимости от своих потребностей и имеющихся ресурсов. Но уже сейчас Microsoft прикладывает большие усилия по разработке облачных технологий, исходя из понимания, что в долгосрочной перспективе ИТ-инфраструктуры и бизнес-платформы предприятий будут разворачиваться в облаке.

В России ряд пользователей уже успел оценить преимущества размещения бизнес-платформы в облачной среде. Для повышения эффективности, снижения расходов и обеспечения совместной работы российские компании выбирают такие решения Microsoft, как Hosted Exchange и SharePoint, реализуемые на базе инфраструктуры партнеров.

Так, санкт-петербургская стоматологическая клиника SmileClinic с помощью функционала Microsoft Exchange Server и набора прикладных решений на его основе, смогла расширить возможности использования программного обеспечения практически без дополнительных затрат. Не вкладывая финансов в приобретение ПО, оборудования и найм ИТ-специалистов, компания получила в свое распоряжение отлично работающий сервис, включающий электронную почту, совместные календари,

контакты и задачи, лицензионную версию программы Microsoft Outlook 2010.

"Заказчик точно представлял, как должна работать поставленная им бизнес-задача. Нам оставалось только предложить техническое решение, которое бы удовлетворяло клинику по всем параметрам: уровню сервиса, качеству предоставления услуги, ценам. Мы выбрали решение на базе Microsoft Exchange Server – прокомментировал Алексей Бахтиаров, генеральный директор, Infobox.

Более того, решив все вопросы с организацией обмена электронными сообщениями внутри компании, медицинское учреждение заинтересовалось возможностью использовать корпоративный портал SharePoint. "Мы организовали на портале систему информирования сотрудников через корпоративный блог, внутреннюю базу знаний для обучения молодых сотрудников, открыв обсуждение проектов и внутренних вопросов на форуме и многое другое. Информация, размещаемая на портале, теперь доступна нам с любого компьютера, в том числе с домашнего - рассказал Илья Скляров, директор клиники Smile Clinic.

В рамках пакета услуг Office 365 впервые предоставляется возможность использовать набор приложений Office Профессиональный плюс 2010 по подписке. За 24$ в месяц организации получат не только Office Профессиональный плюс 2010, но и электронную почту, голосовую почту, внутренние социальные сети, программы для обмена мгновенными сообщениями, веб-порталы, экстранет, голосовые и видео-конференции и

многое другое. Кроме того, пользователи получат круглосуточную техническую поддержку и лицензии на продукты, которые могут использоваться не только в "облаке и многое другое.

Пакет услуг Microsoft Office 365 будет доступен по всему миру в следующем году. Со временем к этому процессу подключится ещё больше компаний. Кроме этого, в следующем году Microsoft планирует добавить к пакету услуг Office 365 решение Microsoft Dynamics CRM Online.

Небольшие компании со штатом менее 25 человек, могут развивать свой бизнес с помощью мощных инструментов веб-приложений Office Web Apps, таких серверов, как Exchange, SharePoint, Lync и фирменного веб-сайта, который можно создать за 15 минут, всего за 6$ в месяц за пользователя.

Средние и крупные предприятия, а также правительственные организации могут подобрать для себя наиболее подходящие инструменты из всего богатства технологий, представленных в Office 365. При этом, например, электронная почта им обойдётся всего за 2$ в месяц за пользователя.

Краткие итоги

В ходе данной лекции мы рассмотрели несколько наиболее ярких примеров облачных сервисов. Количество данных сервисов увеличивается постоянно. Все больше идей и стартапов реализуется именно в "облаках". Все это свидетельствует о популярности данных технологий.

§11. GoogleApps.App Engine

Ознакомиться с основными решениями "облачных"сервисов. Понять принципы предоставления и использования "облачных"услуг.

11.1 GoogleApps

Google Apps – это среда, которая предоставляет следующие средства совместной работы: уже ставший популярным почтовый сервис GMail, клиент обмена мгновенными сообщениями Google Talk (фактически сервис полностью пригоден для общения с любым *jabber*-пользователем), календарь Google *Calendar*, средства для работы с документами и электронными таблицами Google Docs & *Spreadsheets*, "центральную страницу"– *место* для удобного размещения той информации, которая будет общей для всех пользователей, редактор страниц от Google, который позволяет быстро создать и опубликовать нужную информацию.

Часть 1. Функции, доступные пользователю

Описывать все функции, которые доступны каждому пользователю системы Google Apps, – довольно длительное занятие, так как система сочетает в себе несколько довольно гибких продуктов от Google, каждый из которых уже успешно зарекомендовал себя. Кроме того, большинство из этих сервисов довольно популярны по отдельности.

Для начала отметим, что Google Apps – это серьезный сервис, кото-

рый имеет несколько пакетов, каждый из которых отличается разным количеством услуг, предоставляемых Google. Все пакеты предоставляют: полный набор сервисов (почта, календарь, работа с документами, создание страниц, клиент обмена быстрыми сообщениями и так далее), отсутствие ограничений на количество пользовательских учетных записей, доступ с мобильных устройств, панель управления администратора. Таким образом, базовый набор, который предоставляется всем пользователям, приблизительно одинаков, кроме того, существует система миграции с одного пакета на другой, которая позволяет начать с более простого и бесплатного пакета, а позже по необходимости мигрировать на подходящий пакет с нужными дополнениями. Основные различия в следующем.

Начинать лучше со стандартного пакета Google Apps. Он бесплатный.

- Стандартный пакет (Standard Edition) – 2 гигабайта свободного места под почту, помощь через онлайн-ресурсы (но не онлайновая телефонная помощь), присутствие контекстной рекламы на страницах сервисов.

- Премьер-пакет (Premier Edition) – 10 гигабайт под почту, 99,9% гарантированный *uptime* почтовой службы, возможность управления ресурсами, онлайн-помощь в режиме 24/7, которая включает телефонные консультации, API для того, чтобы наилучшим образом интегрировать Google Apps в уже существующую инфраструктуру.

Единственный из всех пакетов, который не является бесплатным. Стоимость данного пакета зависит от количества пользовательских учетных записей. Пользователи пакета получают доступ ко всем новым функциям и сервисам системы по мере их выхода, к примеру, в ближайших планах сервис миграции с других почтовых клиентов, который позволит произвести миграцию с наименьшими усилиями.

- Пакет для образовательных учреждений (Education Edition) – все то же самое, что и в предыдущем случае. С небольшой лишь разницей: всего 2 гигабайта на одну учетную запись, отсутствие гарантии 99,9% времени бесперебойной работы. Пакет предоставляется по отдельной лицензии для некоммерческих образовательных организаций.

После того как администратор, который работает с Google Apps, добавил пользователя в систему (домен Google Apps), пользователь может работать со всеми сервисами, которые включены в системе. Одной из основных идей Google Apps является глобальная интеграция всех сервисов и организация удобной работы людей, которые объединены общей системой Google Apps. Это позволяет серьезно экономить время на организации совместной работы, так как среда для работы оказывается готовой к использованию уже через считанные минуты после полной активации.

Почта и обмен сообщениями

После того как почтовый сервис, относящийся к домену, активизирован, он будет доступен пользователям на специальной странице, имя которой определяется системным администратором. Если настройки домена позволяют это, то все пользователи домена автоматически будут добавлены в контакты каждого нового пользователя системы.

Рисунок 11.1: Список контактов

Все пользователи домена автоматически присутствуют в адресной книге.

Аналогично и с клиентом обмена быстрыми сообщениями: новые пользователи домена автоматически будут добавлены в контакт-лист клиента. Это обеспечивает мгновенный старт работы пользователя после регистрации его в системе. Уже нет необходимости добавлять в контакт-лист каждого коллегу отдельно.

Очевидно, что пользователям достаются и все стандартные возможности почтового сервиса от Google: архивация почты, фильтрация спама, возможность поиска по всем почтовым сообщениям, создание филь-

тров, доступ через POP, пересылка почты и многое другое. То же самое можно сказать и о Google Talk – все функции доступны полностью, причем администратор может ограничить возможность пользователей добавлять в контакт-лист пользователей из других доменов или с других *jabber*-серверов, что позволяет ограничить круг общения только необходимыми контактами.

Календарь

Очень удобное средство планирования личного рабочего времени, которое в условиях глобальной интеграции позволяет планировать не только свое рабочее время, но и учитывать рабочее время и задачи коллег. Основные возможности календаря в Google Apps: создание событий, для каждого из которых можно определить название события, время и продолжительность, определение состава участников и проверка их занятости во время события, установка напоминаний о событиях, просмотр чужих календарей, работа с календарем на мобильных устройствах, управление доступом к календарям и так далее.

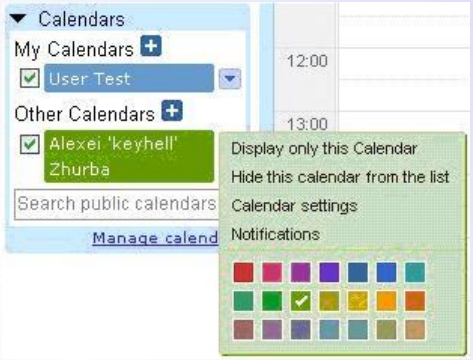


Рисунок 11.2: Просмотр событий в календаре другого пользователя

Пользователь домена может добавить календарь конкретного сотрудника, используя специальную форму, которая позволяет искать календари, используя как ключевые слова, так и адрес электронной почты сотрудника. Таким образом, можно всегда иметь актуальную информацию о задачах и работе определенного сотрудника (очевидно, что только в той мере, в которой это позволяет определить его календарь).

Работа с документами

Поддерживаются все популярные форматы документов: Word, Excel, OpenOffice.

В настоящее время доступна возможность работы со следующими типами файлов: документы Word и Excel, документы OpenOffice, RTF, HTML и текстовые документы. Такой набор поддерживаемых форматов обеспечивает пригодность сервиса для широкого круга пользователей.

Результаты работы могут быть сохранены как на локальный компьютер, так и оставлены на хранение на сервере. Таким образом, для полноценной работы с документами достаточно просто иметь доступ к сервису Google Apps из любой точки мира, с любого компьютера.

Сравнение возможностей самих редакторов Google Docs & *Spreadsheets* с Word и Excel или OpenOffice является темой для отдельной дискуссии. Практика использования показывает, что функций достаточно для того, чтобы подготовить нормальный документ, который содержит обычно используемые элементы оформления: списки, форматирование и различные стили, таблицы, изображения и гиперссылки и так далее.

Google Docs & *Spreadsheets* предоставляет широкие возможности для совместной работы над документами. Для того чтобы сделать работу с документами как можно более удобной и продуктивной для группы сотрудников, доступны следующие возможности.

- Управление версиями документа. Промежуточные версии документа создаются системой автоматически довольно часто и, кроме того, каждый раз, когда пользователь сохраняет документ. Доступна функция сравнения двух выбранных версий, что позволяет легко отслеживать изменения, которые были внесены очередным редактированием документа.

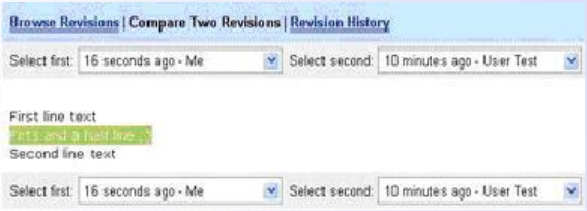


Рисунок 11.3: Сравнение версий

Зеленым цветом выделены изменения между версиями документа.

- Управление доступом к документу. Можно приглашать пользователей системы к совместной работе с документом, указывая, какие права даются пользователю: только просмотр документа или редактирование. Дополнительные возможности позволяют пользователю, приглашенному работать с документом, в свою очередь приглашать других пользователей. Люди, которые одновременно работают с документом, могут устраивать чат для обсуждения изменений в документе, которые будут видны всем участникам обсуждения. Доступна возможность публикации документа с постоянным адресом, что дает возможность любому сотруднику домена получать доступ к документу (например, после того, как все изменения сделаны и одобрены, документ публикуется для всеобщего ознакомления). Кроме того, существует возможность архивирования документов, которые уже приведены в их финальное состояние, но еще могут понадобиться.

Рисунок 11.4: Приглашение пользователю на прочтение документа

Совместная работа с документами в Google Apps организована на высоком уровне, достаточном для того, чтобы работать с документами и быстро и эффективно получать доступ к общей информации. А пользовательские функции редактирования содержимого документов лишь в отдельных аспектах уступают привычным Word, Excel и OpenOffice.

Стартовая страница и редактор страниц

Стартовая страница – первое, что увидят пользователи после входа в Google Apps.

Стартовая страница – это то место, которое предназначено для того, чтобы быть первым, что увидят пользователи после входа в систему Google Apps. Эта страница схожа с персональной страницей Google (*www.igoogle.com*), точно так же она предназначена быть первой страницей, с которой пользователь сталкивается, начиная работу. На ней

могут быть расположены гаджеты от Google и сторонних разработчиков, а также информация, необходимая для начала работы сотрудников. Далеко не полный список полезных вещей, которые можно разместить на стартовой странице Google Apps: гаджеты для предпросмотра личных почтовых ящиков и событий в календаре, поиск, в том числе можно и специальный Google Custom Search, который позволяет искать только то, что реально необходимо, просмотр RSS, закладки, Google *Notebook* и еще большое количество полезных элементов.

Редактор веб-страниц позволяет легко и быстро создавать собственные страницы, которые потом удобно публиковать с помощью сервиса Google. Для создания можно использовать большое количество уже готовых дизайнов (расположение элементов на странице, цветовая гамма и так далее) и довольно удобный редактор, который практически не требует от пользователя знания HTML, CSS и прочих языков. Публикация новой страницы происходит мгновенно, что существенно сокращает время, которое тратится на то, чтобы донести информацию до остальных пользователей или клиентов.

Интеграция в Google Apps достигла того, что документы, которые пришли вам по почте, уже сразу из почтового ящика можно открыть для работы Google Docs & *Spreadsheets*. Переключение между различными сервисами осуществляется простым кликом мышки. Впрочем, нет проблем в том, чтобы держать их все открытыми и готовыми для работы в разных окнах браузера.

Таким образом, Google Apps создает собой среду, которая удобна не только для совместной работы нескольких человек, которые находятся в разных местах, но и для организации совместной работы большого числа сотрудников, которые могут быть раскиданы по различным странам. Учитывая то, что компания Google довольно часто предоставляет API к своим продуктам, тем самым позволяя создавать собственные приложения, которые еще более интегрируют приложения Google в конкретную среду работы, можно сказать, что Google Apps может стать очень удобной средой для совместной работы. Кроме того, компания Google постоянно ведет работу над улучшением существующих сервисов и добавлением новых. В ближайших планах добавление сервиса для миграции с различных почтовых клиентов на GMail и возможность создавать и редактировать презентации, то есть сервис, аналогичный программе PowerPoint.

11.2 App Engine

Google App Engine позволяет выполнять ваши веб-приложения в инфраструктуре Google. Приложения *App Engine* легко создавать, поддерживать и усовершенствовать по мере увеличения трафика и хранилища данных. При работе с *App Engine* вам не понадобится поддерживать *сервер*: просто загрузите свое *приложение*, и пользователи смогут работать с ним.

Приложение можно опубликовать в собственном домене (например, *http://www.example.com*) с помощью Служб Google. Или воспользоваться бесплатным именем в домене appspot.*com*. *Приложение* можно сделать доступным для всех или предоставить *доступ* только участникам вашего коллектива.

Google *App Engine* поддерживает приложения, написанные на нескольких языках программирования. Благодаря среде выполнения *JavaApp Engine* можно создавать приложения с помощью стандартных технологий *Java*, в том числе *JVM*, *сервлетов Java* и языка программирования *Java*, или другого языка, использующего *интерпретатор* или *компилятор* на *JVM*, например JavaScript или *Ruby*. Кроме того, *App Engine* предоставляет специальную среду выполнения *Python*, которая включает быстрый *интерпретатор* и стандартную библиотеку *Python*. Среды выполнения *Java* и *Python* разработаны специально для того, чтобы приложения могли быстро и безопасно выполняться без взаимодействия с другими приложениями в системе.

С *App Engine* нужно платить, только за то, что используете. Не требуется платить за установку или вносить периодические платежи. *Оплата* за использованные приложением ресурсы, такие как объем хранения и трафик, измеряемые в гигабайтах, взимается по обоснованным ставкам. Можно управлять максимальным количеством ресурсов, которые *приложение* может использовать, что позволит всегда оставаться в пределах бюджета.

Начать использовать *App Engine* можно бесплатно. Вам не придется платить за приложения, использующие менее 500 МБ хранилища, а также ресурсы ЦП и трафик, достаточные для эффективного приложения, обслуживающего до пяти миллионов просмотров страниц в месяц. Включив оплату для приложения, эти ограничения повышаются, а *оплата* взимается только за ресурсы, использованные свыше бесплатных уровней.

Среда приложений

Google *App Engine* позволяет легко создавать приложения, надежно работающие даже при большой нагрузке и с большими объемами данных. *App Engine* включает следующие функции:

- динамическую работу в Интернете с полной поддержкой основных веб-технологий;
- постоянное хранилище с запросами, сортировкой и транзакциями;
- автоматическое масштабирование и регулировку нагрузки;
- API для аутентификации пользователей и отправку электронной почты с помощью аккаунтов Google;
- полнофункциональную локальную среду разработки, имитирующую Google App Engine на вашем компьютере.
- запланированные задачи для отслеживания событий в определенное время или через регулярные интервалы.

Приложение может выполняться в одной из двух сред выполнения: *Java* и *Python*. Каждая среда предоставляет стандартные протоколы и основные технологии для разработки веб-приложений.

Приложения работают в безопасной среде, обеспечивающей ограниченный *доступ* к прилагаемой операционной системе. Ограничения позволяют *App Engine* распространять веб-запросы для приложения на несколько серверов, а также запускать и останавливать серверы в зависимости от трафика. Тестовая среда изолирует ваше *приложение* в собственной безопасной и надежной среде, независимой от оборудования, операционной системы и физического расположения веб-сервера.

Ниже приведены примеры ограничений надежной тестовой среды.

Приложение может получать *доступ* к другим компьютерам в Интернете только через предоставленные *API*, службу получения данных по *URL* и службу электронной почты. Другие компьютеры могут подключаться к приложению только путем *HTTP*-запросов (или HTTPS-запросов) на стандартных портах.

Приложение не может выполнять *запись* в файловую систему. *Приложение* может считывать файлы, но только те, которые были загружены вместе с кодом приложения. *Приложение* должно использовать *хранилище данных App Engine, кэш* памяти и другие службы для всех данных, сохраняющихся между запросами.

Код приложения выполняется только в ответ на веб-*запрос* или задачу Cron и в любом случае должен возвращать данные ответа в течение

30 секунд. Обработчик запросов не может создать *подпроцесс* или выполнить код после отправки ответа.

Для среды выполнения *Java* можно разработать *приложение* с помощью стандартных инструментов веб-разработки *Java* и стандартных *API*. *Приложение* взаимодействует со средой с помощью стандарта *Java Servlet* и может использовать стандартные технологии веб-приложения, такие как страницы JavaServer Pages (*JSP*).

Среда выполнения *Java* использует *Java 6*. *SDK Java App Engine* поддерживает разработку приложений с помощью *Java 5* и 6.

Среда включает платформу *Java* SE Runtime *Environment (JRE)* 6 и библиотеки. Ограничения на тестовую среду реализованы в *JVM*. *Приложение* может использовать байтовый код *JVM* или библиотеки в пределах ограничений тестовой среды. Например, при попытке байтового кода открыть *сокет* или записать *файл* возникнет *исключение* среды выполнения.

Доступ к большинству служб *App Engine* можно получить через стандартные *API Java*. Для хранилища данных *App Engine SDK Java* содержит реализации интерфейсов объектов данных *Java (JDO)* и *Java Persistence API* (JPA). Чтобы отправлять сообщения по электронной почте с помощью службы *Mail App Engine*, можно использовать *API JavaMail*. У *API HTTP java.net* есть *доступ* к службе получения данных по *URL App Engine*. Кроме того, для своих служб *App Engine* включает низкоуровневые *API*, которые реализуют дополнительные адаптеры и

позволяют использовать службы напрямую из приложения. Ознакомьтесь с документацией по *API* хранилища данных, кэша памяти, получения данных по *URL*, почты, изображений и аккаунтов Google.

Обычно, чтобы разработать веб-приложения для *JVM,* *Java*-разработчики используют *язык программирования Java* и *API*. Используя совместимые с *JVM* компиляторы и интерпретаторы, можно разрабатывать веб-приложения на других языках, таких как JavaScript, *Ruby* и Scala.

Благодаря среде выполнения *Python App Engine* можно создавать приложения с помощью языка программирования *Python* и выполнять их с помощью оптимизированного интерпретатора *Python*. *App Engine* включает разнообразные *API* и инструменты для разработки веб-приложений *Python*, в том числе *API* моделирования обогащенных данных, простую в использовании инфраструктуру веб-приложений и инструменты для управления и доступа к данным приложения. Для разработки веб-приложений *Python* можно воспользоваться преимуществами широкого набора библиотек и инфраструктур, например Django.

Среда выполнения *Python* использует *Python* версии 2.5.2. Для будущего выпуска рассматривается возможность поддержки *Python 3*.

Среда Python содержит стандартную библиотеку *Python*. Естественно, не все функции библиотеки можно использовать в тестовой среде. Например, при вызове метода, открывающего *сокет* или записывающего в *файл*, возникнет *исключение*. Для удобства отключены несколько

модулей стандартной библиотеки, ключевые функции которых не поддерживаются средой выполнения. Выполнение импортирующего их кода приводит к ошибке.

Код приложения, созданный для среды *Python*, должен быть написан исключительно на *Python*. Расширения, написанные на языке C, не поддерживаются.

Среда *Python* предоставляет мощные *API Python* для служб хранилища данных, аккаунтов Google, получения *URL* и электронной почты. *App Engine* также предоставляет простую инфраструктуру веб-приложения *Python* под названием webapp, которая облегчает создание приложений.

Вместе с приложением можно загружать сторонние библиотеки, но они должны быть реализованы на чистом *Python* и не должны требовать неподдерживаемых модулей стандартной библиотеки.

App Engine предоставляет мощную службу распределенного хранения данных, включающую механизм запросов и транзакции. Расширение распределенной *базы данных* за счет данных аналогично расширению распределенного веб-сервера за счет трафика.

Хранилище данных App Engine не похоже на обычную реляционную базу данных. Объекты данных, или "записи имеют вид и обладают набором свойств. С помощью запросов можно получать записи определенного вида, отфильтрованные и отсортированные по значениям свойств. Значения свойств могут быть любыми из поддерживаемых типов значений свойств.

Для объектов хранилища данных не требуется схема. Структура объектов данных определяется в коде приложения. Интерфейсы *JDO*/JPA *Java* и хранилища данных *Python* включают функции для применения структуры в приложении. *Приложение* может получить *прямой доступ* к хранилищу данных, чтобы реализовать нужную часть структуры.

Хранилище данных согласованно и использует оптимистическое управление параллельными транзакциями. Обновление записи происходит в транзакции, которая выполняется повторно определенное количество раз, если другие процессы одновременно пытаются обновить ту же *запись. Приложение* может выполнять несколько операций с хранилищем данных в одной транзакции. Эти *операции* либо все будут успешны, либо все будут неудачны, что обеспечивает *целостность данных.*

Хранилище данных реализует транзакции в своей распределенной сети с помощью "групп записей". *Транзакция* осуществляет действия над записями в одной группе. Записи каждой из групп хранятся вместе для эффективного выполнения транзакций. При создании записей *приложение* может присоединять их к группам .

App Engine поддерживает интеграцию приложения с аккаунтами Google для аутентификации пользователей. Ваше *приложение* может позволить пользователю войти в свой аккаунт Google и получить *доступ* к адресу электронной почты и отображаемому имени, связанным с аккаунтом. Использование аккаунтов Google дает пользователю возможность быстрее начать использовать ваше *приложение*, поскольку

ему не придется создавать новый аккаунт. Это также снимает с вас необходимость реализовывать систему аккаунтов пользователей только для своего приложения.

Если *приложение* работает в Службах Google, оно может использовать те же функции для участников вашей организации и аккаунтов Служб Google.

API пользователей может также сказать приложению, является ли текущий *пользователь* зарегистрированным администратором приложения. Это упрощает реализацию административных зон сайта.

App Engine предоставляет набор служб, позволяющих выполнять рядовые *операции* при управлении приложением. Для доступа к этим службам предоставлены следующие *API*.

Приложения могут получать *доступ* к ресурсам в Интернете, например к веб-службам или другим данным, с помощью службы получения *URL App Engine*. Служба получения данных по *URL* обеспечивает получение веб-ресурсов посредством той же высокоскоростной инфраструктуры Google, которая получает веб-страницы для многих других продуктов Google.

- **Электронная почта**

  Приложения могут отправлять сообщения электронной почты с помощью почтовой службы App Engine. Для отправки электронных сообщений эта служба использует инфраструктуру Google.

- **Memcache**

  Служба Memcache предоставляет вашему приложению высокопроизводительный кэш памяти, использующий структуру ключ-значение, к которому может получать доступ несколько экземпляров приложения. Кэш памяти пригодится для данных, не требующих постоянного хранения и функции работы с транзакциями, которые предоставляет хранилище данных, например для временных данных или данных, копируемых из хранилища в кэш для ускорения доступа.

- **Работа с изображениями**

  Служба изображений позволяет приложению работать с изображениями. С помощью этого API можно изменять размер, обрезать, поворачивать и отражать изображения в форматах JPEG и PNG.

- **Запланированные задачи**

  Служба Cron позволяет планировать задачи для выполнения через определенные интервалы. Подробнее о ней можно узнать в документации по службе Cron Python и Java.

- **Процесс разработки**

  Инструментарий разработки App Engine (SDK) для Java и Python включает приложение на веб-сервере, которое имитирует службы App Engine на локальном компьютере. Каждый SDK включает все

API и библиотеки, доступные в App Engine. Кроме того, веб-сервер имитирует безопасную тестовую среду, включающую проверку на доступ к системным ресурсам, запрещенную в App Engine.

Каждый SDK также включает инструмент для добавления приложения в App Engine. После создания кода приложения, статических файлов и файлов конфигурации запустите этот инструмент, чтобы загрузить данные. Инструмент запросит адрес электронной почты и пароль вашего аккаунта Google.

При создании нового выпуска приложения, уже работающего в App Engine, вы сможете загрузить его как новую версию. Старая версия будет работать для пользователей до тех пор, пока вы не перейдете на новую. Вы можете тестировать новую версию в App Engine, пока работает старая.

SDK Java выполняется на любой платформе с Java 5 или Java 6. SDK доступен в виде ZIP-файла. При использовании среды разработки Eclipse, чтобы создать, проверить и добавить приложения App Engine, можно использовать плагин Google для Eclipse. SDK также содержит инструменты, работающие из командной строки, позволяющие запускать сервер разработки и добавлять приложения.

SDK Python реализован на чистом Python и выполняется на любой платформе с Python 2.5, в том числе Windows, Mac OS X и

Linux. SDK доступен в виде Zip-файла, а для Windows и Mac OS X доступны программы установки.

Консоль администрирования – это веб-интерфейс для управления приложениями, работающими в App Engine. Ее можно использовать для создания новых приложений, настройки доменных имен, изменения рабочей версии приложения, изучения доступа и журналов ошибок и просмотра хранилища данных приложения.

- **Квоты и ограничения**

  Создать приложение в App Engine не только просто, но и бесплатно! Вы можете создать аккаунт и опубликовать приложение, которое можно будет использовать сразу же, бесплатно и без дополнительных требований. Приложение с бесплатным аккаунтом может использовать до 500 МБ хранилища данных и до пяти миллионов просмотров страниц в месяц. Если нужно больше, включите оплату, установите максимальный дневной бюджет и распределите его между ресурсами в соответствии со своими потребностями.

  Для аккаунта разработчика можно зарегистрировать до 10 приложений.

  Каждому приложению предоставляются ресурсы в пределах ограничений или "квот". Квота определяет объем определенного ресурса, который можно использовать в течение календарного дня. В ближайшее время будет возможно настроить некоторые из этих

квот, оплатив дополнительные ресурсы.

Для некоторых функций ограничения не связаны с квотами, а предназначены для сохранения стабильности системы. Например, если приложение вызывается для выполнения веб-запроса, оно должно создать ответ в течение 30 секунд. Если этот процесс длится слишком долго, то он прекращается, а сервер возвращает пользователю код ошибки. Таймаут запроса динамичен и может уменьшаться для экономии ресурсов, если обработчик запросов достигает его слишком часто.

Другой пример ограничения обслуживания – количество возвращаемых запросом результатов. Запрос может вернуть не более 1000 результатов. Запросы, которые могли бы вернуть больше, возвращают только максимально допустимое количество. В этом случае такой запрос скорее всего не вернет результаты до наступления таймаута, но благодаря ограничению ресурсы хранилища данных будут сэкономлены.

Попытки обойти или превысить квоты, например, выполняя приложения в нескольких совместно работающих аккаунтах, нарушают Условия предоставления услуг и могут привести к отключению приложений или закрытию аккаунтов.

Список квот и объяснение системы квот, включая квоты, которые можно увеличить, включив оплату, можно посмотреть в статье Кво-

ты.

Краткие итоги:

В ходе данной лекции мы рассмотрели несколько наиболее ярких примеров облачных сервисов. Количество данных сервисов увеличивается постоянно. Все больше идей и стартапов реализуется именно в "облаках". Все это свидетельствует о популярности данных технологий.

Лабораторный практикум

Лабораторная работа № 1.

«Установка и настройка Hyper-V»

Целью лабораторной работы является практическое освоение технологий виртуализации Hyper-V.

Аппаратура и программные инструменты, необходимые для лабораторной работы:

Компьютер или *сервер*, поддерживающий виртуализацию, *операционная система* Microsoft *Windows Server* 2008 (R2).

Продолжительность лабораторной работы:

6 академических часов.

Ход работы:

1. Установите роль Hyper-V на сервере Windows 2008 (или выше).

Включите компьютер, дождитесь загрузки. Войдите в систему, используя учетную запись администратора. Откройте **Пуск – Диспетчер сервера – Роли – Добавить роли.** В открывшемся окне мастера установки выберите роль сервера Hyper-V и следуя инструкциям мастера произведите установку.

2.Произведите сетевые настройки.

После успешной установки роли Hyper-V, произведите настройку виртуальной сети. Откройте **Пуск – Диспетчер сервера – Роли – Hyper-V.** В панели **Действия** выберите **Диспетчер виртуальной сети.** Создайте **Внешнюю** и **Внутреннюю** виртуальные сети.

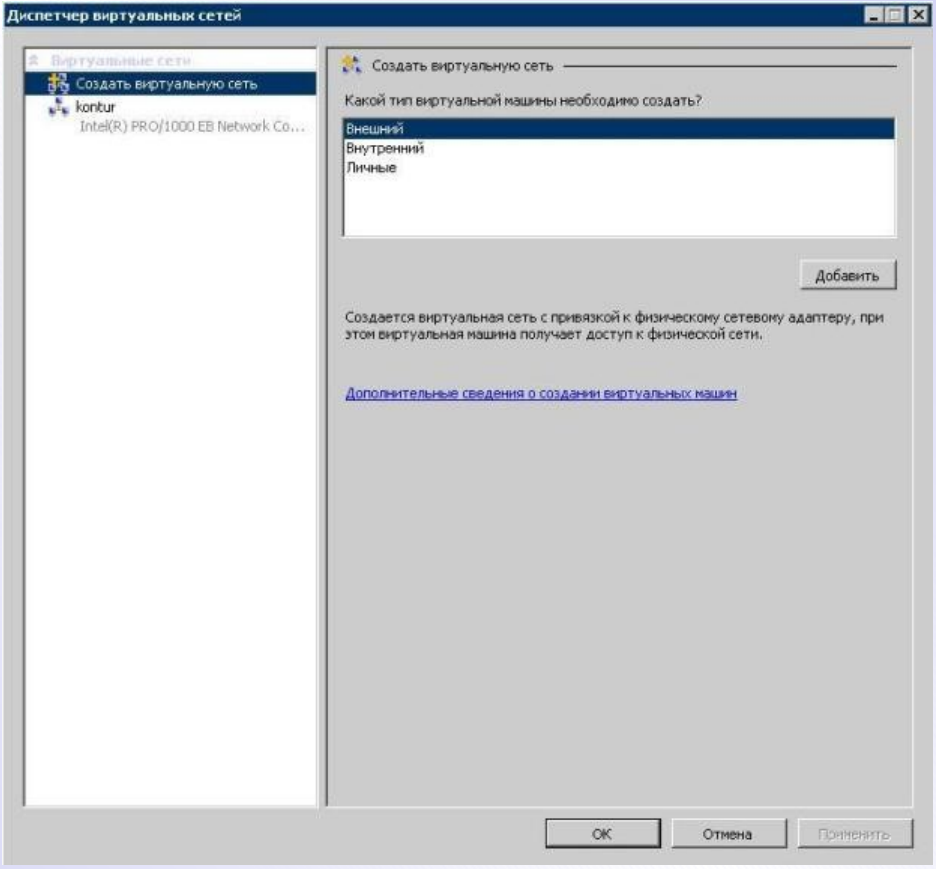


Рисунок 11.1: Диспетчер виртуальной сети.

3. Создание виртуальных машин

Откройте **Пуск – Диспетчер сервера – Роли – Hyper-V.**

1. Создайте виртуальную машину с фиксированным виртуальным жестким диском.
2. Создайте виртуальную машину с динамически расширяющимся виртуальным жестким диском.
3. Создайте виртуальную машину для гостевой операционной системы Windows 7.
4. Измените количество оперативной памяти, процессоров, тип сетевого подключения в конфигурации виртуальной машины.
5. Установите Integration Services для гостевой операционной системы.
6. Создайте снимок виртуальной машины.
7. Произведите изменения в гостевой операционной системе.
8. Отмените изменения, используйте возврат к предыдущему снимку.
9. Выполните экспорт виртуальной машины.
10. Измените конфигурацию виртуальной машины, измените размер диска виртуальной машины.

4. Обзор System Center Virtual Machine Manager

1. Запустите Configuration Analyzer для тестирования системы.
2. Установите SCVMM сервер.
3. Установите консоль администратора SCVMM.

4. Добавьте пользователей в группу IT Admin Support.
5. Создайте новую группу узлов в *VMM*.
6. Добавьте необходимые сервера Hyper-V в *VMM*.
7. Измените конфигурацию виртуальной машины.
8. Создайте шаблон виртуальной машины.
9. Создайте несколько экземпляров виртуальных машины из полученного шаблона.
10. Обзор библиотеки *VMM*.
11. Произведите конвертирование физического сервера в виртуальное окружение.

Лабораторная работа № 2.

«Установка и настройка VMWare Workstation»

Целью лабораторной работы является практическое освоение технологий виртуализации VMWare на примере VMWare *Workstation*.

Аппаратура и программные инструменты, необходимые для лабораторной работы:

Настольный или портативный *компьютер*, поддерживающий виртуализацию, *операционная система* Microsoft *Windows* XP, Vista, *Windows* 7.

Продолжительность лабораторной работы:

2 академических часа.

Ход работы:

1. **Установите VMWare Workstation.** Используя установочный дистрибутив VMWare Workstation, установите продукт на компьютер. Запустите программу, открыв **Пуск – Все программы – VMWare – VMWare Workstation**.

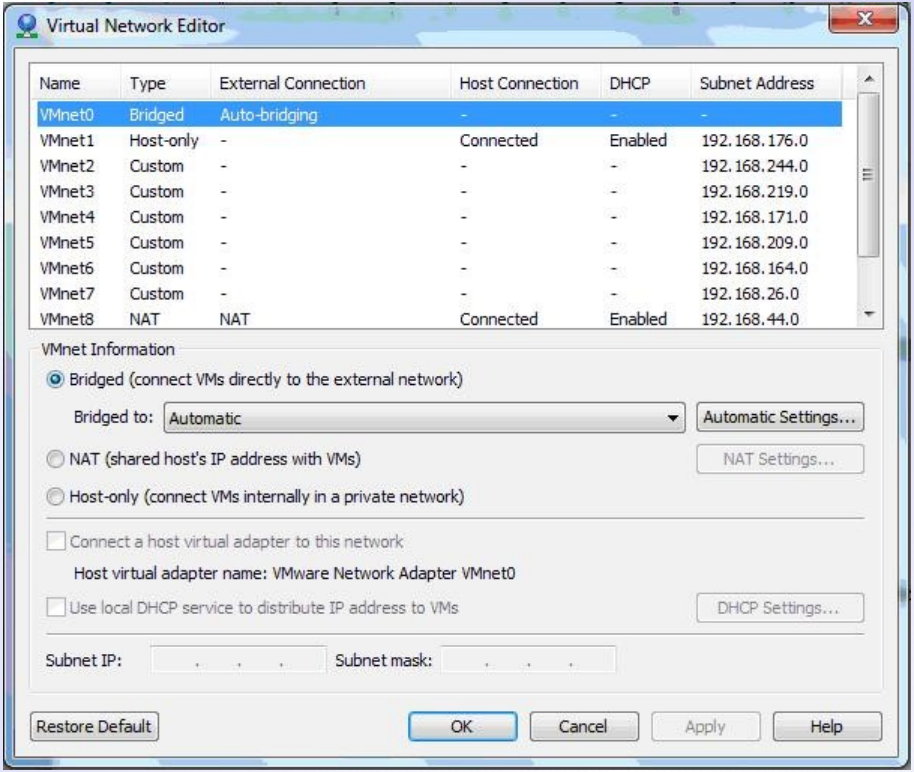


Рисунок 11.2

2. **Произведите сетевые настройки.** Откройте **Пуск – Все программы – VMWare – Virual Network Editor**. Произведите конфигурацию виртуальной сети.

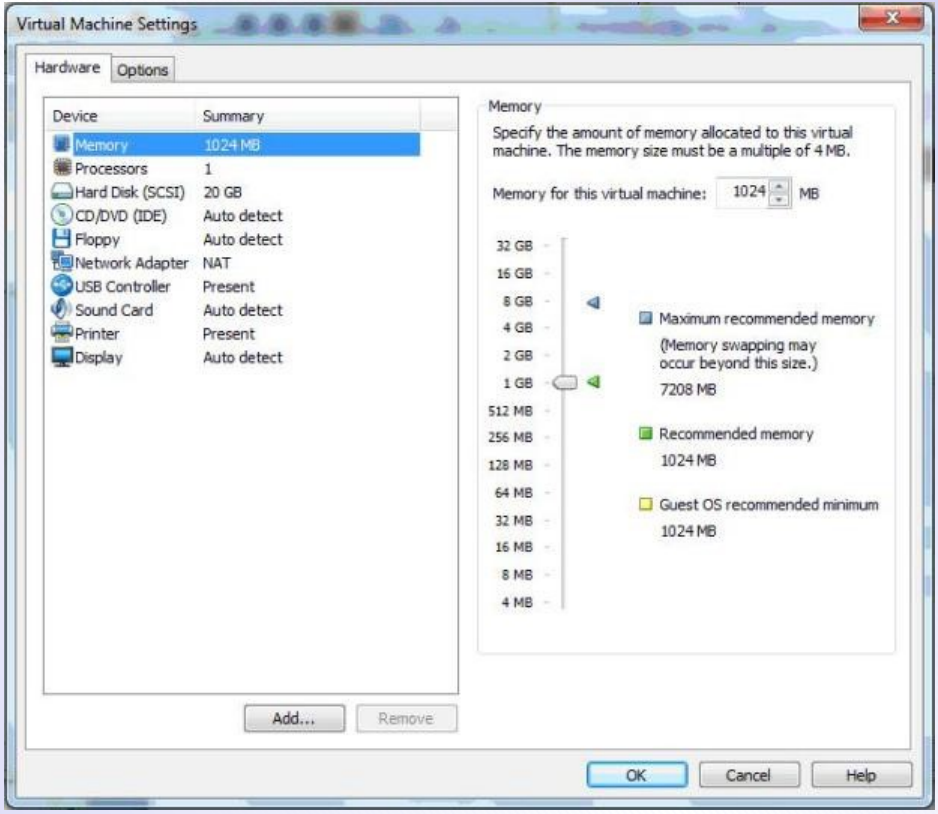


Рисунок 11.3

3. **Создайте виртуальную машину для гостевой операционной системы Windows 7.** В меню **File – New –Virtual Machine** создайте новую виртуальную машину. Установите операционную

систему Windows 7 в виртуальной машине.

4. **Установите VMWare Tools.** После установки операционной системы Windows 7, установите инструменты VMWare в меню **VM – Install VMWare Tools**.

5. **Создайте снимок виртуальной машины.** Создайте снимок виртуальной машины в меню **VM – Snapshot – Take Snapshot.**

6. **Произведите изменения в гостевой операционной системе.** Выполните произвольные изменения в виртуальной машине, скопируйте на рабочий стол несколько ярлыков, создайте несколько папок.

7. **Отмените изменения, использовав возврат к предыдущему снимку.** Выполните возврат к предыдущему снимку виртуальной машины в меню **VM – Snapshot – Revert to Snapshot** и выберите предыдущий снимок.

8. **Измените конфигурацию виртуальной машины.** Произведите изменение конфигурации виртуальной машины в меню **VM – Settings**. Увеличьте количество оперативной памяти, количество процессоров. Увеличьте размер жесткого диска. Создайте дополнительный жесткий диск.

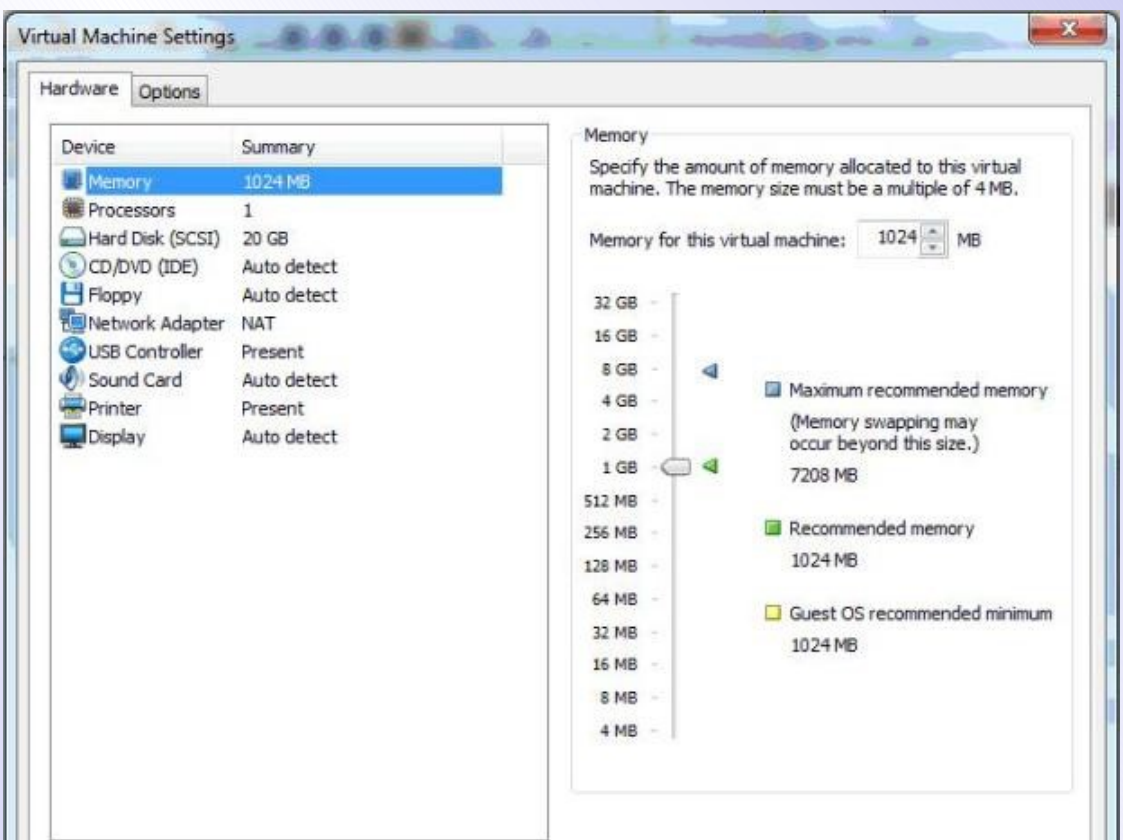

Virtual Machine Settings
Hardware
Options
Device
Summary
Memory
1024 MB
Processors
1
Hard Disk (SCSI)
20 GB
CD/DVD (IDE)
Auto detect
Floppy
Auto detect
Network Adapter
NAT
USB Controller
Present
Sound Card
Auto detect
Printer
Present
Display
Auto detect
Add...
Remove
Memory
Specify the amount of memory allocated to this virtual machine. The memory size must be a multiple of 4 MB.
Memory for this virtual machine:
1024
MB
32 GB
16 GB
8 GB
4 GB
2 GB
1 GB
512 MB
256 MB
128 MB
64 MB
32 MB
16 MB
8 MB
4 MB
Maximum recommended memory
(Memory swapping may occur beyond this size.)
7208 MB
Recommended memory
1024 MB
Guest OS recommended minimum
1024 MB
OK
Cancel
Help

Рисунок 11.4

Лабораторная работа № 3

«Создание первого Windows Azure приложения»

Целью лабораторной работы является практическое освоение создания приложений *Windows* Azure.

Аппаратура и программные инструменты, необходимые для лабораторной работы:

1. Настольный или портативный компьютер, поддерживающий виртуализацию, операционная система Microsoft Windows XP, Vista, Windows 7.
2. Доступ к сети Интернет.
3. Наличие аккаунта Windows Azure.

Продолжительность лабораторной работы:

2 академических часа.

Ход работы:

Создание проекта в Visual Studio

1. Откройте меню **Пуск - Все программы - Microsoft Visual Studio 2010 - Microsoft Visual Studio 2010**.
2. В меню **File** выберите **New** и затем **Project**.
3. В диалоговом окне **New Project** разверните узел **Visual C#** и в списке **Installed Templates** выберите **Cloud**.

4. В списке **Templates** выберите **Windows AzureCloud Service**. Введите Name **"GuestBook"**, имя solution **"Begin"**. Затем выберите расположение внутри папки **Ex1-BuildingYourFirstWindowsAzureApp**. Убедитесь что опция **Create directory for solution** выбрана и нажмите **OK** чтобы создать проект.

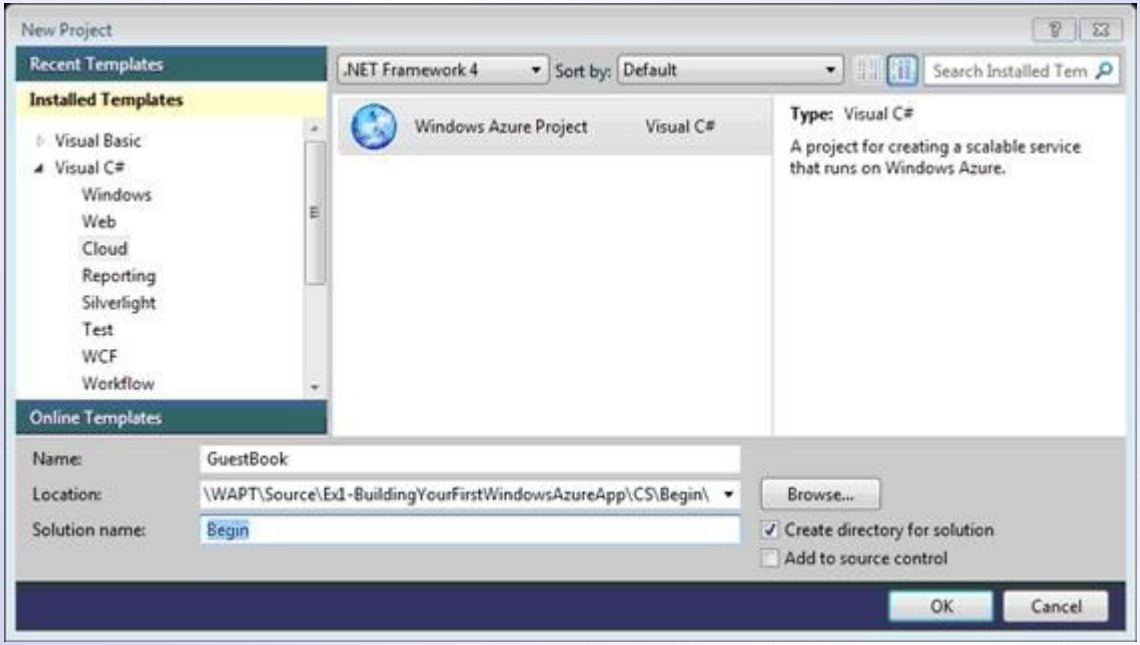


Рисунок 11.5: Создание нового проектаWindows Azure Cloud Service

5. В диалоге **New Cloud Service Project** разверните узел **Visual C#** и выберите **ASP.NET Web Role**. Переместите выбранную роль в проект. Выберите роль в проекте, нажмите на изображение "карандаш"и введите имя **GuestBook_WebRole**. Нажмите **OK** для завершения.

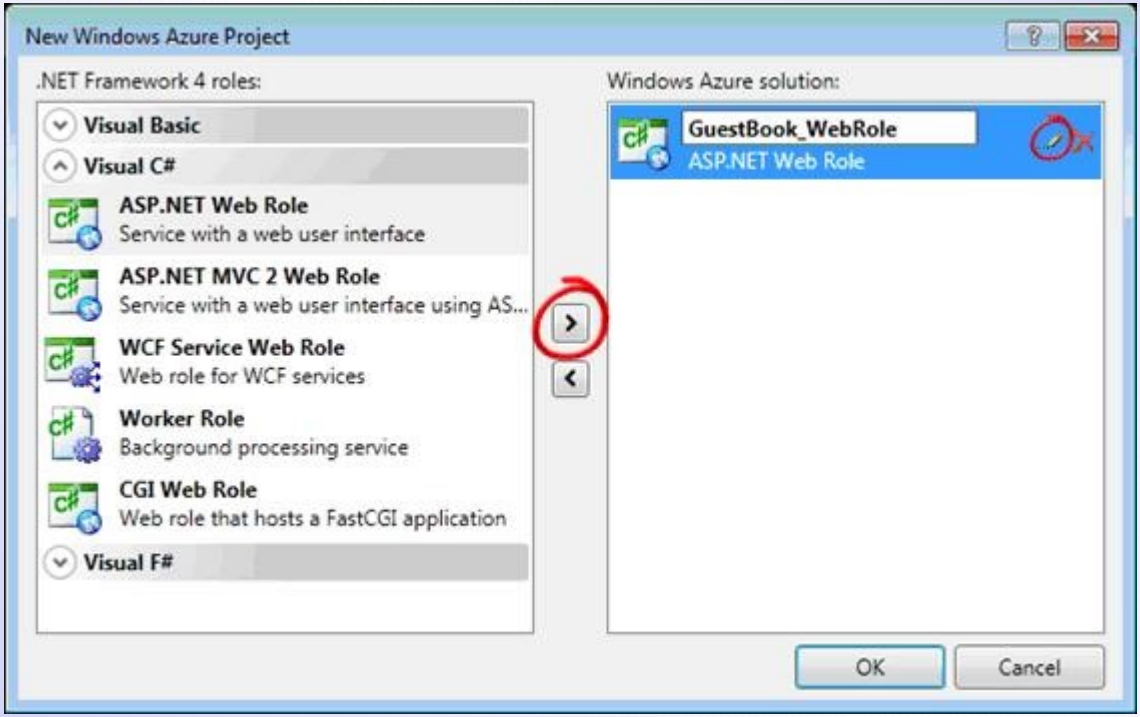


Рисунок 11.6: Добавление ролей в проект

6. Обратите внимание на структуру проекта в **Solution Explorer**.

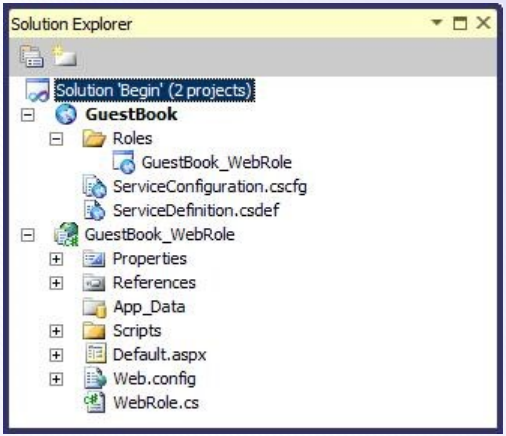


Рисунок 11.7: Структура проекта в Solution Explorer

Создание модели данных для элементов в Table Storage

1. В **Solution Explorer** нажмите правой кнопкой мыши по **Begin**, выберите **Add - New Project**.
2. В диалоге **Add New Project**, разверните узел **Visual C#** в списке **Installed Templates**, выберите категорию **Windows** и выделите **Class Library** в списке шаблонов. Убедитесь что выбрат **. NET Framework 3.5**. Введите имя **GuestBook_Data** и нажмите **OK**.

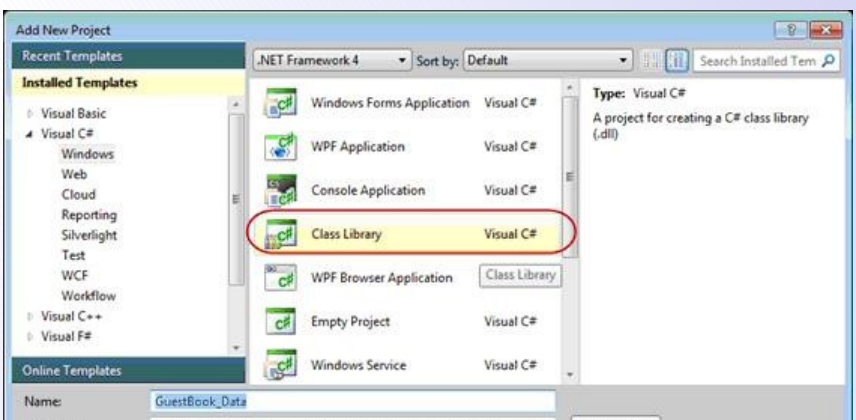


Рисунок 11.8: Создание библиотеки классов

3. Удалите файл класса по умолчанию. Нажмите правой кнопкой по **Class1.cs** и выберите **Delete**. Нажмите **OK**.
4. Добавьте ссылку на библиотеку **.NET** для **ADO.NET** в проект **GuestBook_Data**. В **Solution Explorer** нажмите правой кнопкой по проекту **GuestBook_Data**, выберите **Add Reference**, затем выберите закладку **.NET**, выделите компонент **System.Data.Service.Client** и нажмите **OK**.
5. Выполните пункт 4, добавив ссылку библиотеку **Microsoft.WindowsAzure.StorageClient**

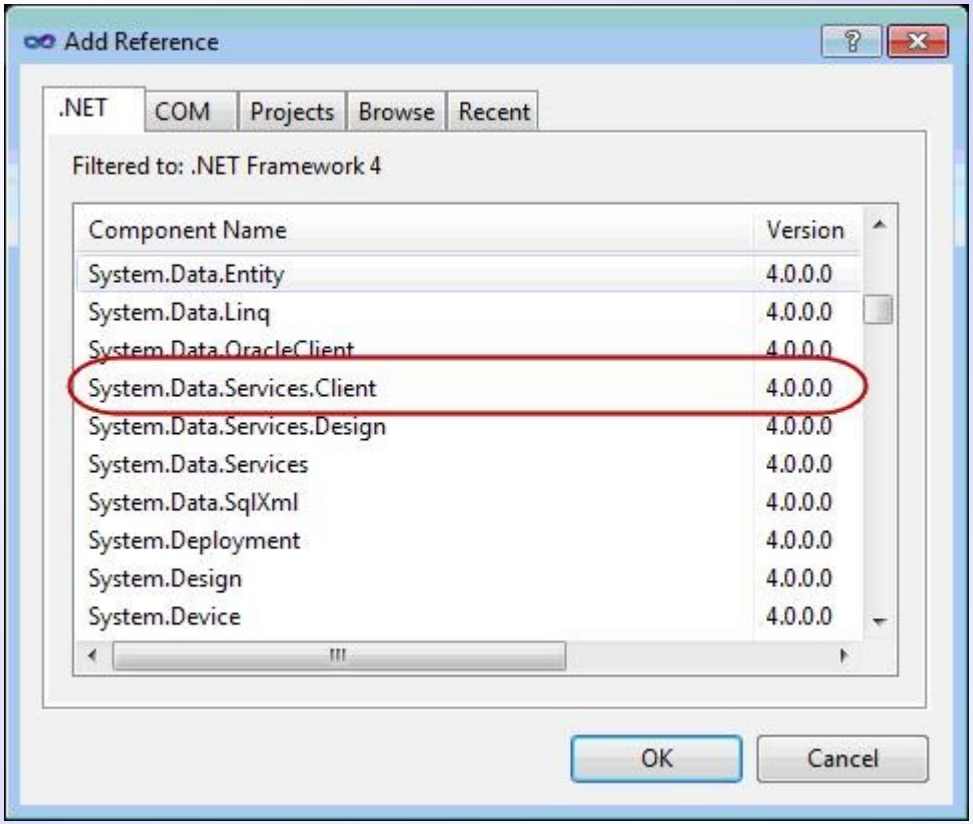


Рисунок 11.9

6. Нажмите правой кнопкой мыши по **GuestBook_Data** в **Solution Explorer**, выберите **Add**, затем **Class**. В диалоге **Add New Item** введите имя **GuestBookEntry.cs** и нажмите **Add**.

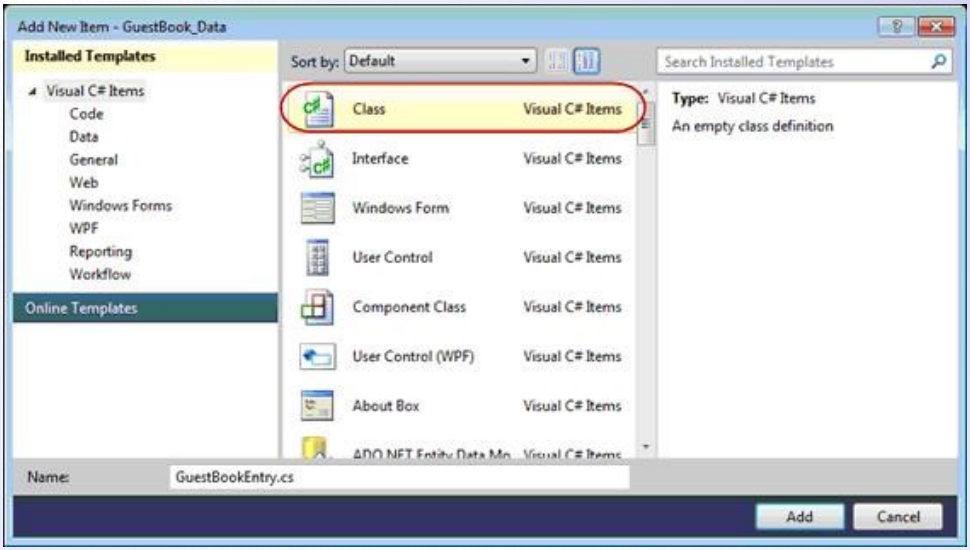


Рисунок 11.10

7. Откройте файл **GuestBookEntry.cs**, добавьте в начало файла *using Microsoft.WindowsAzure.StorageClient*;
8. Измените объявление класса **GuestBookEntry**

```
public class GuestBookEntry :
    Microsoft.WindowsAzure.StorageClient.TableServiceEntity
        {
        }
```

9. Добавьте конструктор по умолчанию

```
public GuestBookEntry()
```

```
        {
    PartitionKey = DateTime.UtcNow.ToString("MMddyyyy");
// Row key allows sorting, so we make sure
    the rows come back in time order.
    RowKey = string.Format("{0:10}_{1}", DateTime.MaxValue.Ticks -
    DateTime.Now.Ticks, Guid.NewGuid());
        }
```

10. Добавьте свойства

```
public string Message { get; set; }
public string GuestName { get; set; }
public string PhotoUrl { get; set; }
public string ThumbnailUrl { get; set; }
```

11. Сохраните файл **GuestBookEntry.cs**.

12. Нажмите правой кнопкой мыши по **GuestBook_Data** в **Solution Explorer**, выберите **Add**, затем **Class**. В диалоге **Add New Item** введите имя **GuestBookDataContext.cs** и нажмите **Add**.

13. Откройте файл **GuestBookDataContext.cs**, добавьте в начало файла

```
using Microsoft.WindowsAzure;
using Microsoft.WindowsAzure.StorageClient;
```

14. Измените объявление класса **GuestBookDataContext** и добавьте конструктор

```
public class GuestBookDataContext : TableServiceContext
 {
```

```
    public GuestBookDataContext(string baseAddress,
    StorageCredentials credentials) : base(baseAddress, credentials)
      { }
    }
```

15. Добавьте свойство

```
public class GuestBookDataContext : TableServiceContext
  {
            ...
   public IQueryable<GuestBookEntry> GuestBookEntry
        {
          get
          {
      return this.CreateQuery<GuestBookEntry>("GuestBookEntry");
      }
        }
   }
```

16. Нажмите правой кнопкой мыши по **GuestBook_Data** в **Solution Explorer**, выберите **Add**, затем **Class**. В диалоге **Add New Item** введите имя **GuestBookDataContext.cs** и нажмите **Add**.
17. Откройте файл **GuestBookEntryDataSource.cs**, добавьте в начало файла

```
using Microsoft.WindowsAzure;
using Microsoft.WindowsAzure.StorageClient;
```

18. Далее измените класс

```
public class GuestBookEntryDataSource
  {
   private static CloudStorageAccount storageAccount;
   private GuestBookDataContext context;
  }
```

19. Добавьте конструктор

```
public class GuestBookEntryDataSource
 {
   private static CloudStorageAccount storageAccount;
   private GuestBookDataContext context;

   static GuestBookEntryDataSource()
   {
    storageAccount = CloudStorageAccount.FromConfigurationSetting(
    "DataConnectionString");

     CloudTableClient.CreateTablesFromModel(
     typeof(GuestBookDataContext),
     storageAccount.TableEndpoint.AbsoluteUri,
     storageAccount.Credentials);
   }
 }
```

20. Добавьте конструктор для класса **GuestBookDataEntrySource**

```
public GuestBookEntryDataSource()
  {
  this.context = new GuestBookDataContext(
  storageAccount.TableEndpoint.AbsoluteUri,
```

```
storageAccount.Credentials);
this.context.RetryPolicy =RetryPolicies.Retry(
3, TimeSpan.FromSeconds(1));
}
```

21. Добавьте методы

```
public IEnumerable<GuestBookEntry> Select()
{
 var results = from g in this.context.GuestBookEntry
 where g.PartitionKey == DateTime.UtcNow.ToString("MMddyyyy")
 select g;
 return results;
}

public void UpdateImageThumbnail(string partitionKey,
string rowKey, string thumbUrl)
{
 var results = from g in this.context.GuestBookEntry
 where g.PartitionKey == partitionKey && g.RowKey == rowKey
 select g;

 var entry = results.FirstOrDefault<GuestBookEntry>();
 entry.ThumbnailUrl = thumbUrl;
 this.context.UpdateObject(entry);
 this.context.SaveChanges();
}
```

22. Сохраните файл **GuestBookEntryDataSource.cs**

Создание Веб роли для отображения гостевой книги

1. В **Solution Explorer** нажмите правой кнопкой по проекту **GuestBook_WebRole**, выберите **Add Reference**, затем выберите закладку **Project**, выделите **GuestBook_Data** и нажмите **OK**.
2. Нажмите правой кнопкой по **Default.aspx** и выберите **Delete**. Нажмите **OK**.
3. В **Solution Explorer** нажмите правой кнопкой по проекту **GuestBook_WebRole**, выберите **Add**, выделите **Existing Item**.
4. В диалоге **Add Existing Item** выберите директорию **Source - Ex1-BuildingYourFirstWindowsAzureApp - CS - Assets**, выберите **Add**.
5. В **Solution Explorer** нажмите правой кнопкой по **Default.aspx**, выберите **View Code**, объявите следующие пространства имен

```
        using System.IO;
        using System.Net;
        using Microsoft.WindowsAzure;
        using Microsoft.WindowsAzure.ServiceRuntime;
        using Microsoft.WindowsAzure.StorageClient;
using GuestBook_Data;
```

6. В классе укажите

```
        private static bool storageInitialized = false;
        private static object gate = new Object();
private static CloudBlobClient blobStorage;
```

7. Найдите событие **SignButton_Click** и добавьте следующий код

```
protected void SignButton_Click(object sender, EventArgs e)
 {
  if (FileUpload1.HasFile)
   {
    InitializeStorage();

    // upload the image to blob storage
    CloudBlobContainer container =
    blobStorage.GetContainerReference("guestbookpics");
    string uniqueBlobName = string.Format(
    "image_{0}.jpg", Guid.NewGuid().ToString());
    CloudBlockBlob blob =
    container.GetBlockBlobReference(uniqueBlobName);
    blob.Properties.ContentType =
    FileUpload1.PostedFile.ContentType;
    blob.UploadFromStream(FileUpload1.FileContent);
    System.Diagnostics.Trace.TraceInformation(
    "Uploaded image '{0}' to blob storage as '{1}'",
    FileUpload1.FileName, uniqueBlobName);

    // create a new entry in table storage
    GuestBookEntry entry = new GuestBookEntry()
    {
     GuestName = NameTextBox.Text,
     Message = MessageTextBox.Text,
     PhotoUrl = blob.Uri.ToString(),
     ThumbnailUrl = blob.Uri.ToString() };
     GuestBookEntryDataSource ds = new GuestBookEntryDataSource();
     ds.AddGuestBookEntry(entry);
     System.Diagnostics.Trace.TraceInformation("Added entry
```

```
        {0}-{1} in table storage for guest '{2}'",
        entry.PartitionKey, entry.RowKey, entry.GuestName);
      }

    NameTextBox.Text = "";
    MessageTextBox.Text = "";

    DataList1.DataBind();
  }
```

8. Обновите метод **Timer1_Tick**

```
protected void Timer1_Tick(object sender, EventArgs e)
 {
          DataList1.DataBind();
 }
```

9. Обновите событие **Page_Load**

```
protected void Page_Load(object sender, EventArgs e)
 {
   if (!Page.IsPostBack)
     {
      Timer1.Enabled = true;
     }
 }
```

10. Произведите изменения в методе **InitializeStorage**

```
private void InitializeStorage()
 {
   if (storageInitialized)
```

```
    {
      return;
    }
    lock (gate)
    {
      if (storageInitialized)
      {
        return;
      }
       try
       {
       // read account configuration settings
var storageAccount =
CloudStorageAccount.FromConfigurationSetting(
"DataConnectionString");

       // create blob container for images
       blobStorage = storageAccount.CreateCloudBlobClient();
       CloudBlobContainer container =
       blobStorage.GetContainerReference("guestbookpics");
       container.CreateIfNotExist();

       // configure container for public access
       var permissions = container.GetPermissions();
       permissions.PublicAccess =
       BlobContainerPublicAccessType.Container;
       container.SetPermissions(permissions);
       }
            catch (WebException)
             {
```

```
                throw new WebException("Storage services
                initialization failure. " + "Check your storage
                account configuration settings. If running locally, "
                + "ensure that the Development Storage service is
                running.");
              }
              storageInitialized = true;
            }
        }
```

11. В **Solution Explorer** разверните узел **Roles** в проекте **GuestBook**. Нажмите два раза по **GuestBook_WebRole**, откроется свойства данной роли, выберите закладку **Setting**. Нажмите **Add Setting**, наберите **"DataConnectionString"** в колонке **Name**, измените **Type** на **ConnectionString** и нажмите **Add Setting**.

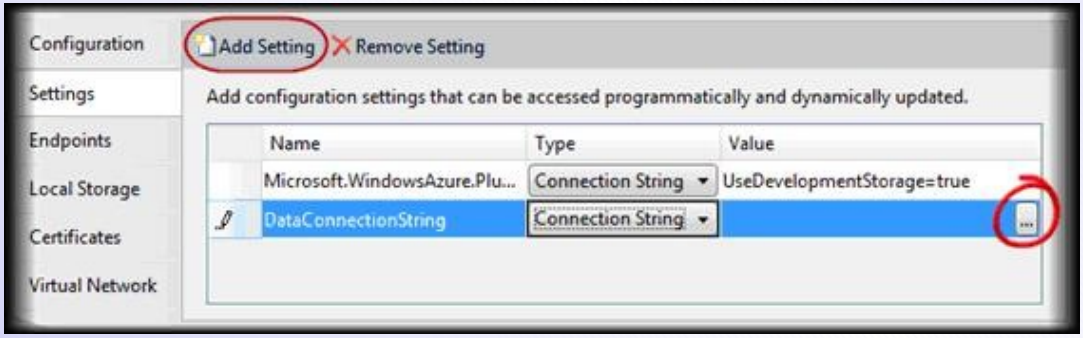


Рисунок 11.11

12. В диалоге **Storage Connection String** выберите **Use development storage** и нажмите **OK**.

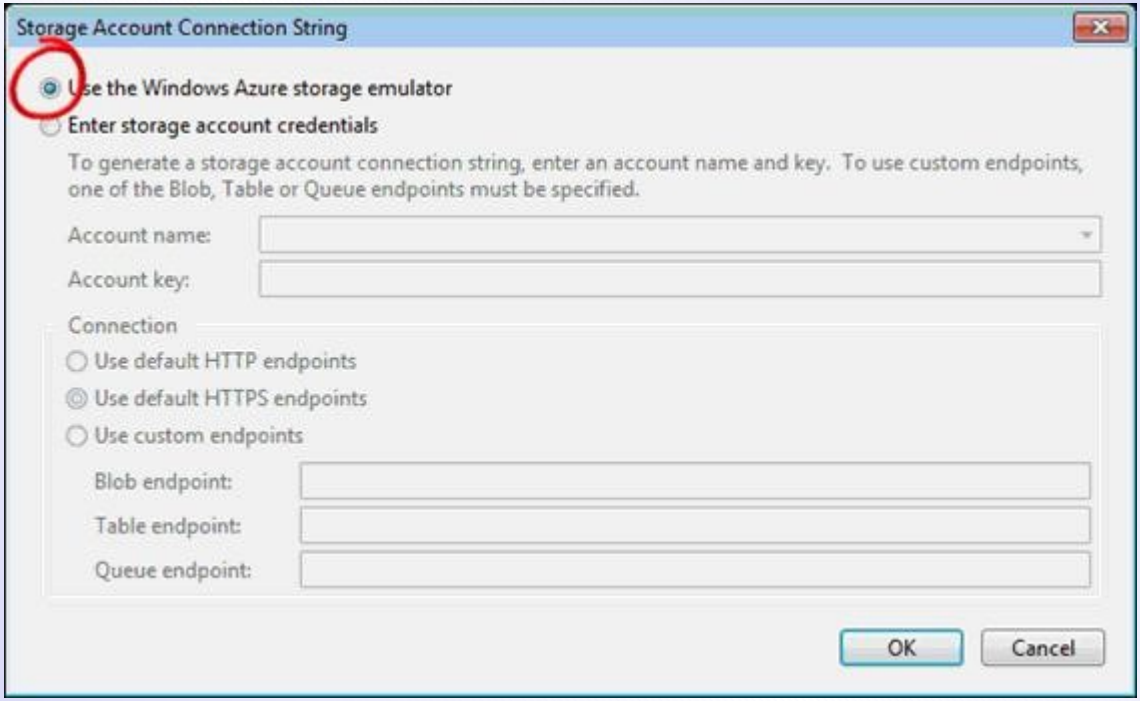


Рисунок 11.12

13. Сохраните изменения.
14. В проекте **GuestBook_WebRole**, откройте файл **Global.asax.cs**
15. Объявите пространства имен

```
using Microsoft.WindowsAzure;
using Microsoft.WindowsAzure.ServiceRuntime;
```

16. Вставьте следующий код внутрь метода Application_Start, заменим содержимое по умолчанию

```
void Application_Start(object sender, EventArgs e)
  {
  Microsoft.WindowsAzure.CloudStorageAccount.
  SetConfigurationSettingPublisher((configName, configSetter) =>
   {
   configSetter(RoleEnvironment.GetConfigurationSettingValue(
   configName));
   });
  }
```

Организация очереди рабочих элементов для обработки в фоне

1. В **Solution Explorer** нажмите правой кнопкой по **Default.aspx**, выберите **View Code**, объявите элемент клиента очереди

```
public partial class _Default : System.Web.UI.Page
    {
    private static bool storageInitialized = false;
    private static object gate = new Object();
    private static CloudBlobClient blobStorage;
    private static CloudQueueClient queueStorage;
```

```
            ...
    }
```

2. Найдите метод **InitializeStorage** и вставьте следующий код внутрь данного метода

```
public partial class Default : System.Web.UI.Page
 {
            ...
  private void InitializeStorage()
  {
     ...
   try
   {
       ...
 // configure container for public access
 var permissions = container.GetPermissions();
 permissions.PublicAccess =
 BlobContainerPublicAccessType.Container;
 container.SetPermissions(permissions);

 // create queue to communicate with worker role
 queueStorage = storageAccount.CreateCloudQueueClient();
 CloudQueue queue = queueStorage.GetQueueReference("guestthumbs");
 queue.CreateIfNotExist();
   }
 catch (WebException)
    {
         ...
 }
```

Проверка

1. Нажмите **F5** для запуска сервиса. Сервис запустится в **development fabric**. Для открытия пользовательского интерфейса необходимо нажать правой кнопкой мыши на значке в области уведомления панели задач и выбрать **Show Development Fabric UI**

Рисунок 11.13

2. Переключитесь на **Internet Explorer** для просмотра приложения **GuestBook**
3. Добавьте новую запись в гостевой книг

Рисунок 11.14

Лабораторная работа № 4.

«Развертывание приложения Windows Azure»

Целью лабораторной работы является практическое освоение процесса развертывания приложений *Windows* Azure.

Аппаратура и программные инструменты, необходимые для лабораторной работы:

1. Настольный или портативный компьютер, поддерживающий виртуализацию, операционная система Microsoft Windows XP, Vista, Windows 7.
2. Доступ к сети Интернет.
3. Наличие аккаунта Windows Azure.

Продолжительность лабораторной работы:

2 академических часа

Ход работы:

Создание Storage Account

1. Откройте в веб браузере адрес http://windows.azure.com и войдите используя **Windows Live ID**, ассоциированный с учетной записью **Windows Azure**.

Рисунок 11.15

2. На панели инструментов **Windows Azure** выберите **New Storage Account**.

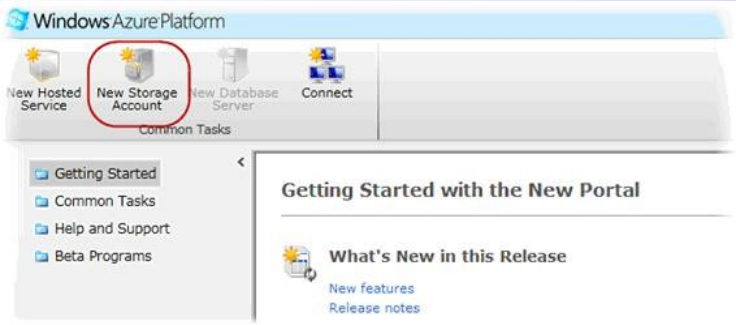


Рисунок 11.16

3. В диалоге **Create a New Storage Account** выберите Вашу подписку из списка.

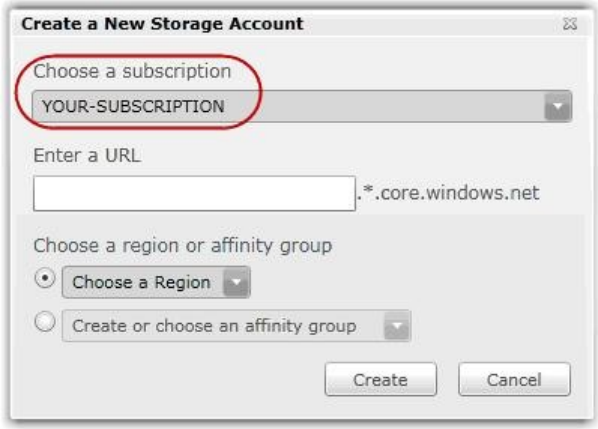


Рисунок 11.17

4. Введите имя для Вашего **Storage Account**.

Рисунок 11.18

5. Выберите настройку **Create or choose an affinity group** и затем выберите из списка **Create a new affinity group**.

Рисунок 11.19

6. В диалоге **Create a New Affinity Group** введите **Affinity Group Name**, выберите размещение и нажмите OK.

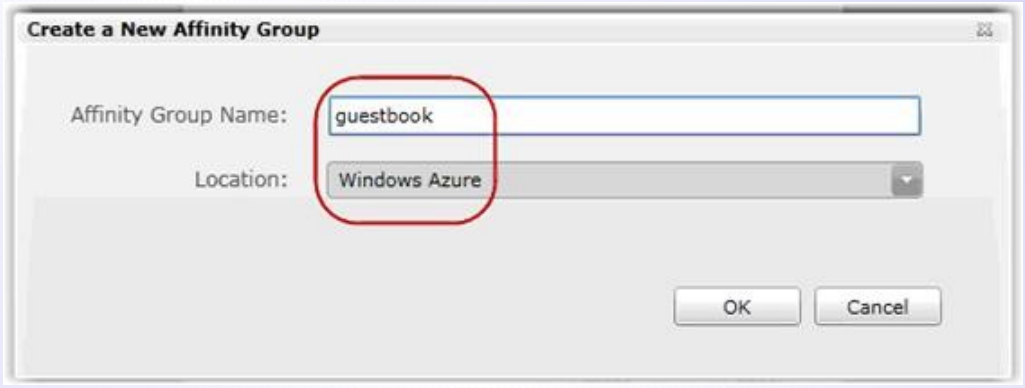


Рисунок 11.20

7. Вернитесь в диалог **Create a New Storage Account** и нажмите **Create** для создания нового **Storage Account**.

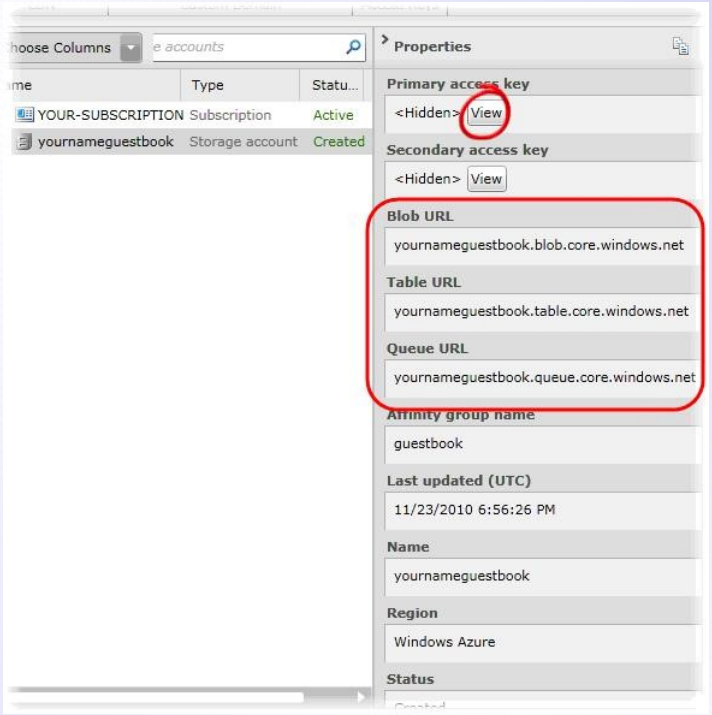


Рисунок 11.21

8. Нажмите кнопку **View**. В диалоге **View Storage Access Keys** нажмите кнопку **Copy to Clipboard** следующую за **Primary Access Key**.

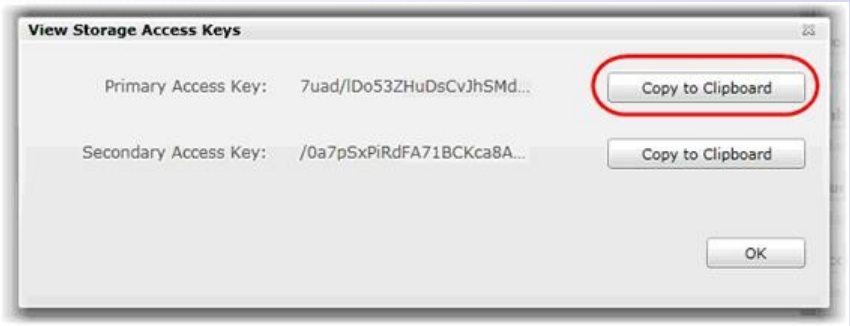


Рисунок 11.22

9. Нажмите **Hosted Services** на левой панели. Нажмите кнопку **New Hosted Service** на панели инструментов.

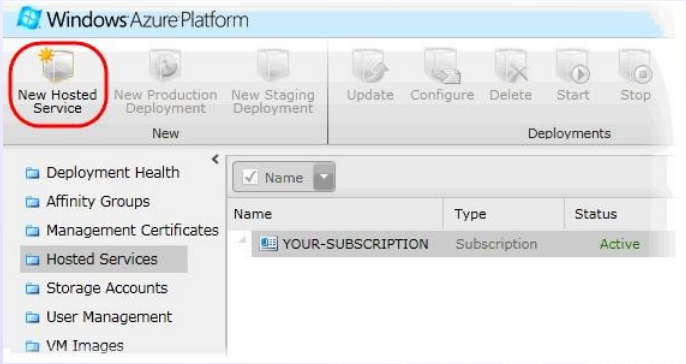


Рисунок 11.23

10. В диалоге **Create a new Hosted Service** выберите подписку и

списка **Choose a subscription**.

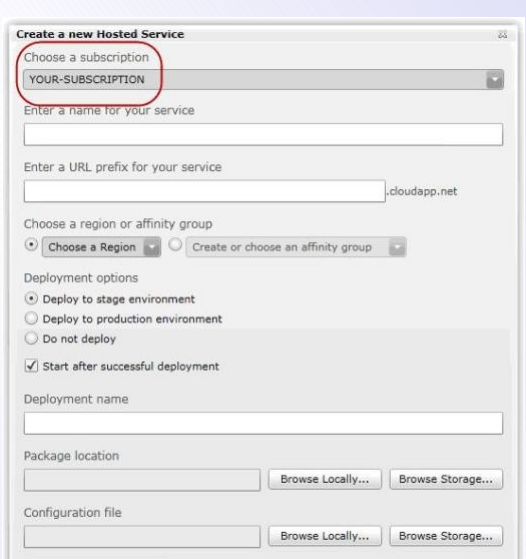


Рисунок 11.24

11. Введите имя сервиса **Enter a name for your service** и выберите префикс адреса в **Enter a URL prefix for your service**.

Рисунок 11.25

12. Выберите настройку **Create or choose an affinity group** и затем выберите из списка **guestbook**.

Рисунок 11.26

13. Выберите настройку **Do not Deploy**.
14. Нажмите **OK** для создания сервиса и дождитесь завершения процесса инициализации.

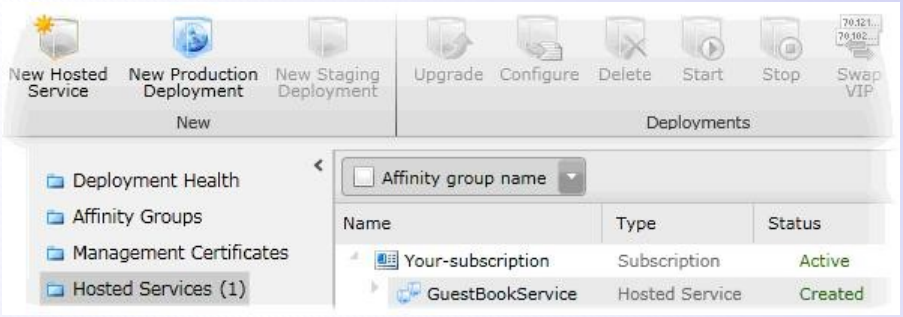


Рисунок 11.27

Развертывание приложения на портале Windows Azure Platform

1. Откройте меню **Пуск - Все программы - Microsoft Visual Studio 2010** и запустите **Microsoft Visual Studio 2010** от имени администратора.
2. В меню **File** выберите **Open** и затем **Project/Solution**. Откройте файл **Begin.sln** проекта **Ex3-WindowsAzureDeployment**.
3. Для изменения конфигурации хранилища перед развертыванием сервиса, откройте файл **ServiceConfiguration.cscfg** расположенный в сервисе **GuestBook**. Замените значение **[YOUR_ACCOUNT_NAME]** на значение **Storage Account Name**, которое Вы выбрали когда настраивали **Storage account** в Задании 1.
4. Далее замените значение **[YOUR_ACCOUNT_KEY]** на значе-

ние **Primary Access Key** которое Вы получили раннее, при создании **Storage account** в Задании 1

```
<ServiceConfiguration serviceName="GuestBook" xmlns="http://schemas.microsoft.c
 <Role name="GuestBook_WebRole">
  <Instances count="1" />
  <ConfigurationSettings>
   <Setting name="Microsoft.WindowsAzure.Plugins.Diagnostics.ConnectionString"
            value="DefaultEndpointsProtocol=https;
                   AccountName=[YOUR_ACCOUNT_NAME];
                   AccountKey=[YOUR_ACCOUNT_KEY]" />
   <Setting name="DataConnectionString"
            value="DefaultEndpointsProtocol=https;
                   AccountName=[YOUR_ACCOUNT_NAME];
                   AccountKey=[YOUR_ACCOUNT_KEY]" />
 </ConfigurationSettings>
 </Role>
  <Role name="GuestBook_WorkerRole">
   <Instances count="1" />
   <ConfigurationSettings>
    <Setting name="Microsoft.WindowsAzure.Plugins.Diagnostics.ConnectionString"
             value="DefaultEndpointsProtocol=https;
                    AccountName=[YOUR_ACCOUNT_NAME];
                    AccountKey=[YOUR_ACCOUNT_KEY]" />
    <Setting name="DataConnectionString"
             value="DefaultEndpointsProtocol=https;
                    AccountName=[YOUR_ACCOUNT_NAME];
                    AccountKey=[YOUR_ACCOUNT_KEY]" />
  </ConfigurationSettings>
 </Role>
</ServiceConfiguration>
```

Рисунок 11.28

5. Необходимо создать пакет для развертывания в облаке. Для этого откройте меню правой кнопкой мыши в проекте **GuestBook** и выберите **Publish**. В диалоге **Deploy Windows Azure project**, выберите настройку **Create Service Package Only** и затем нажмите **OK**.

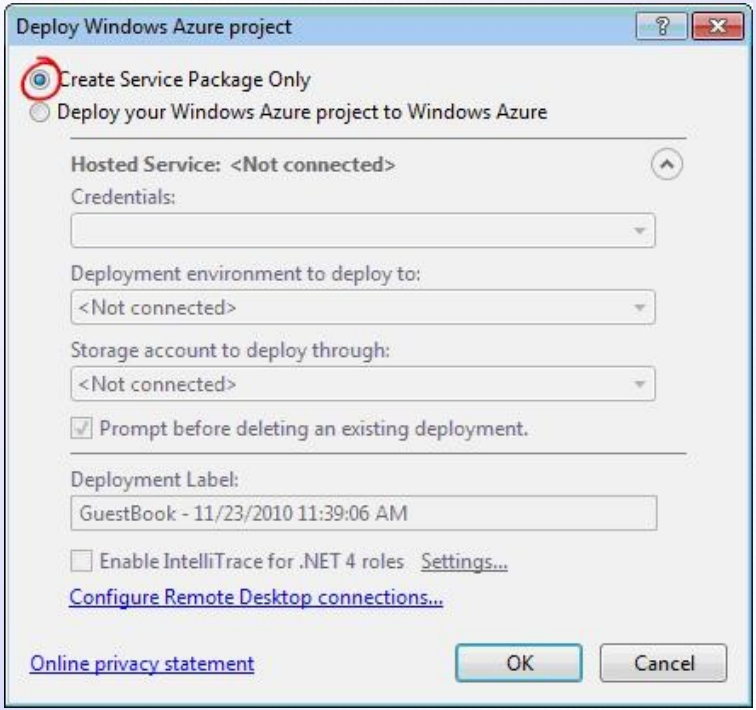


Рисунок 11.29

6. Переключитесь обратно в окно браузера, где открыт портал управления **Windows Azure**.
7. На портале выберите сервис, который Вы создали в предыдущем упражнении и нажмите **New Staging Deployment** на панели инструментов.

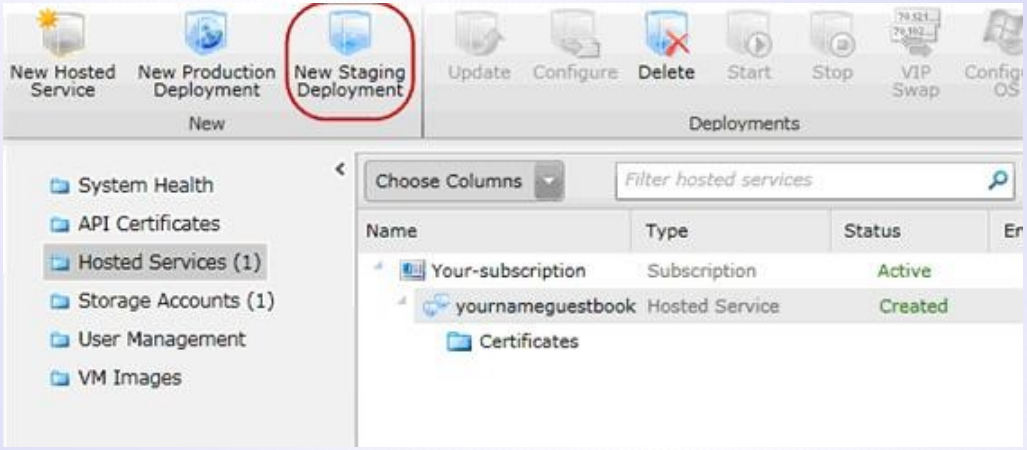


Рисунок 11.30

8. В диалоге **Create a new Deployment** выберите **Package location**, нажмите **Browse Locally**, выберите папку где был создан пакет в пункте 4 и затем выберите файл **GuestBook.cspkg**.
9. Далее выберите **Configuration File**, нажмите **Browse Locally** и выберите **ServiceConfiguration.cscfg** из той же папки (пункт 8).

10. Введите имя **Deployment name** и нажмите **OK**. В диалоге с предупреждением выберите **Yes**

Рисунок 11.31

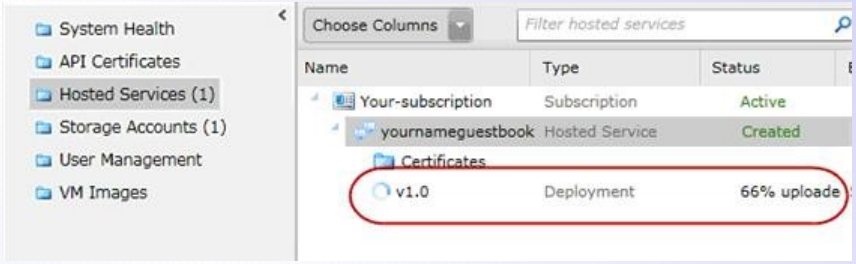


Рисунок 11.32

11. Дождитесь завершения процесса развертывания, это можно занять несколько минут

Лабораторная работа № 5.

«Работа с Blob»

Целью лабораторной работы является практическое освоение процесса работы с *Blob* в *Windows* Azure.

Аппаратура и программные инструменты, необходимые для лабораторной работы:

1. Настольный или портативный компьютер, поддерживающий виртуализацию, операционная система Microsoft Windows XP, Vista, Windows 7.
2. Доступ к сети Интернет.
3. Наличие аккаунта Windows Azure.

Продолжительность лабораторной работы:

2 академических часа

Ход работы:

Получение Blob данных из хранилища

1. Откройте меню **Пуск - Все программы - Microsoft Visual Studio 2010 - Microsoft Visual Studio 2010**.
2. В меню **File** выберите **Open** и затем **Project/Solution**. Откройте файл проекта **ExploringWindowsAzureStorageVS2010/Source/Ex02-WorkingWithBlobs/begin/CS/begin.sln**

3. В **Solution Explorer**, в проекте **RDImageGallery** нажмите правой кнопкой по узлу **RDImageGallery_WebRole** и выберите **Properties**

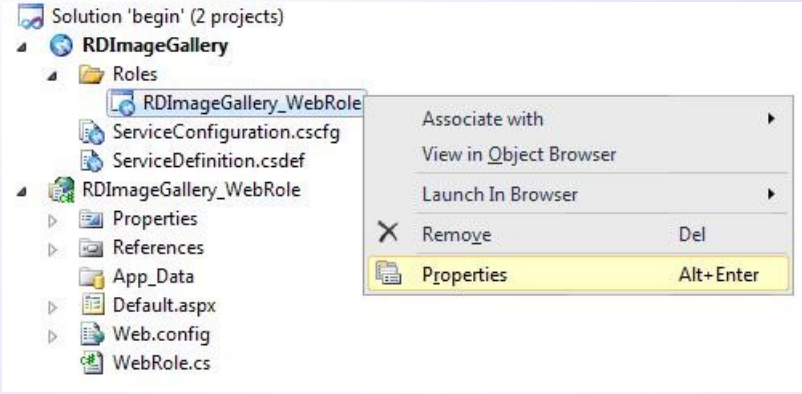


Рисунок 11.33

4. На закладке **Settings** создайте **ConnectionString** с именем **DataConnectionStrin**. Выберите **Use development storage**.

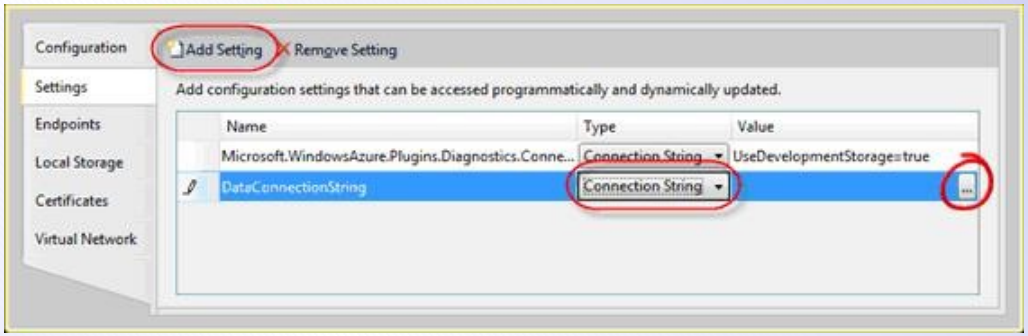


Рисунок 11.34

Рисунок 11.35

5. Создайте **String** с именем **ContainerName** и значением **gallery**

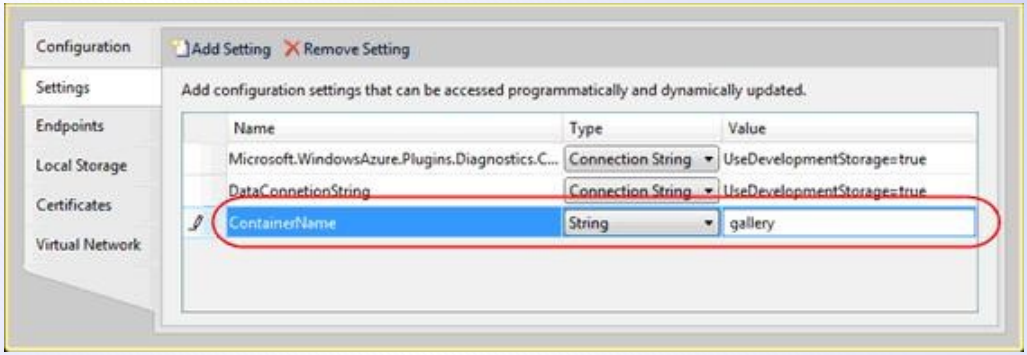


Рисунок 11.36

6. В **Solution Explorer** нажмите правой кнопкой по **Default.aspx** в проекте **RDImageGallery_WebRole**, выберите **View Code**, объявите следующие пространства имен
7. Убедитесь что в начале файла объявлены пространства имен

```
using Microsoft.WindowsAzure;
using Microsoft.WindowsAzure.StorageClient;
using Microsoft.WindowsAzure.ServiceRuntime;
```

8. В конец класса **_Default** добавьте метод

```
private void EnsureContainerExists()
 {
   var container = GetContainer();
   container.CreateIfNotExist();
```

```
    var permissions = container.GetPermissions();
    permissions.PublicAccess =
    BlobContainerPublicAccessType.Container;
    container.SetPermissions(permissions);
  }
```

9. В конец класса **_Default** добавьте метод

```
private CloudBlobContainer GetContainer()
 {
   // Get a handle on account, create a blob storage
   client and get container proxy
   var account = CloudStorageAccount.FromConfigurationSetting(
   "DataConnectionString");
   var client = account.CreateCloudBlobClient();

   return client.GetContainerReference(
   RoleEnvironment.GetConfigurationSettingValue("ContainerName"));
 }
```

10. Добавьте следующий код в метод **Page_Load**

```
protected void Page_Load(object sender, EventArgs e)
 {
  try
   {
    if (!IsPostBack)
    {
     this.EnsureContainerExists();
    }
```

```
        this.RefreshGallery();
      }
      catch (System.Net.WebException we)
      {
        status.Text = "Network error: " + we.Message;
        if (we.Status == System.Net.WebExceptionStatus.ConnectFailure)
        {
          status.Text += "<br />Please check if the blob storage
          service is running at " +
          ConfigurationManager.AppSettings["storageEndpoint"];
        }
      }
      catch (StorageException se)
      {
        Console.WriteLine("Storage service error: " + se.Message);
      }
    }
```

11. В конец класса **_Default** добавьте метод

```
    private void RefreshGallery()
    {
      images.DataSource = this.GetContainer().ListBlobs(
      new BlobRequestOptions()
      {
        UseFlatBlobListing = true,
        BlobListingDetails = BlobListingDetails.All
      });
      images.DataBind();
    }
```

12. Нажмите **F5** для запустка приложения. Запустится браузер с приложением.

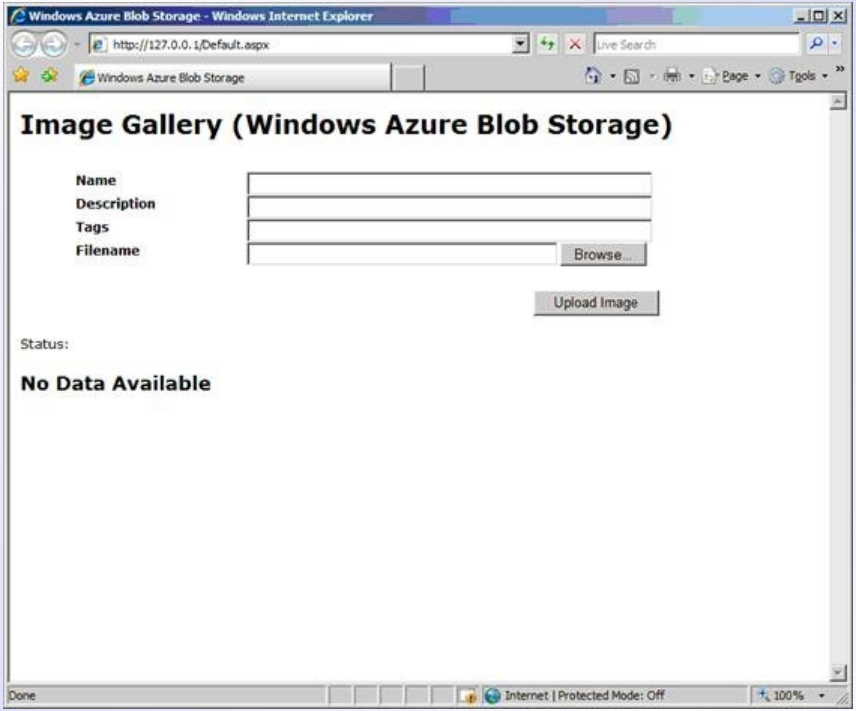


Рисунок 11.37

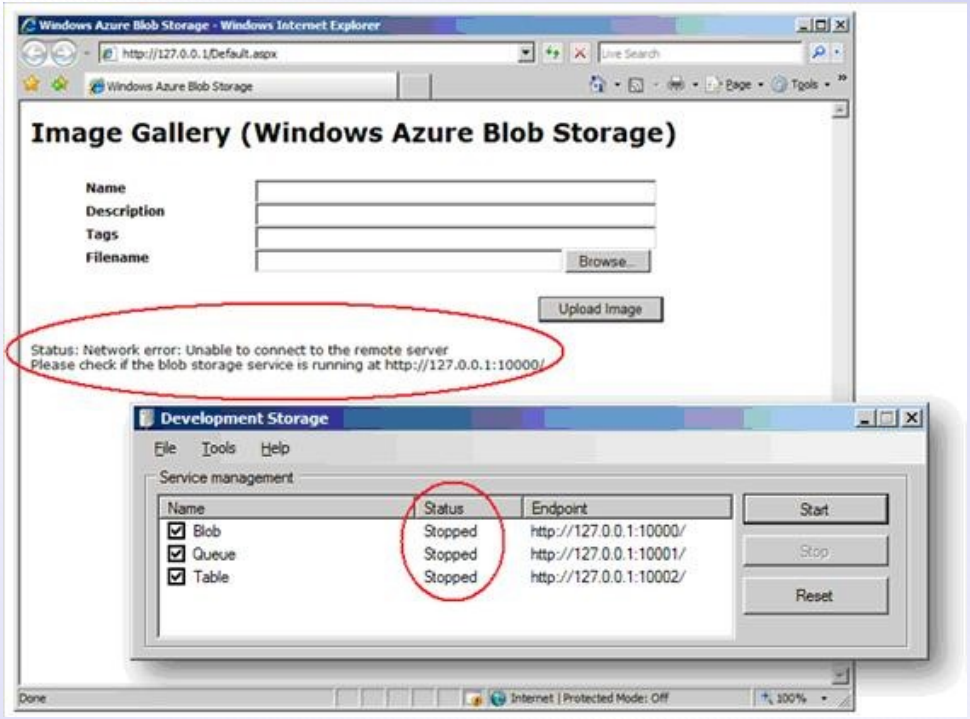


Рисунок 11.38

Загрузка Blob данных в хранилище.

1. Откройте **Default.aspx.cs**
2. Добавьте метод в конец страницы.

```
private void SaveImage(string id, string name, string description,
string tags, string fileName, string contentType, byte[] data)
```

```
{
 // Create a blob in container and upload image bytes to it
 var blob = this.GetContainer().GetBlobReference(name);

 blob.Properties.ContentType = contentType;

 // Create some metadata for this image
 var metadata = new NameValueCollection();
 metadata["Id"] = id;
 metadata["Filename"] = fileName;
 metadata["ImageName"] = String.IsNullOrEmpty(name)
 ? "unknown" : name;
 metadata["Description"] = String.IsNullOrEmpty(description)
 ? "unknown" : description;
 metadata["Tags"] = String.IsNullOrEmpty(tags)
 ? "unknown" : tags;

 // Add and commit metadata to blob
 blob.Metadata.Add(metadata);
 blob.UploadByteArray(data);
}
```

3. Измените метод **upload_Click**.

```
protected void upload_Click(object sender, EventArgs e)
 {
  if (imageFile.HasFile)
   {
    status.Text = "Inserted [" + imageFile.FileName + "] -
    Content Type [" + imageFile.PostedFile.ContentType + "] -
    Length [" + imageFile.PostedFile.ContentLength + "]";
```

```
        this.SaveImage(Guid.NewGuid().ToString(), imageName.Text,
        imageDescription.Text, imageTags.Text, imageFile.FileName,
        imageFile.PostedFile.ContentType, imageFile.FileBytes);

        RefreshGallery();
      }
        else
        status.Text = "No image file";
    }
```

4. Нажмите **F5** для запуска приложения.
5. Введете метаданные **Name**, **Description** и **Tags** . Для выбора изображения нажмите **Browse**.

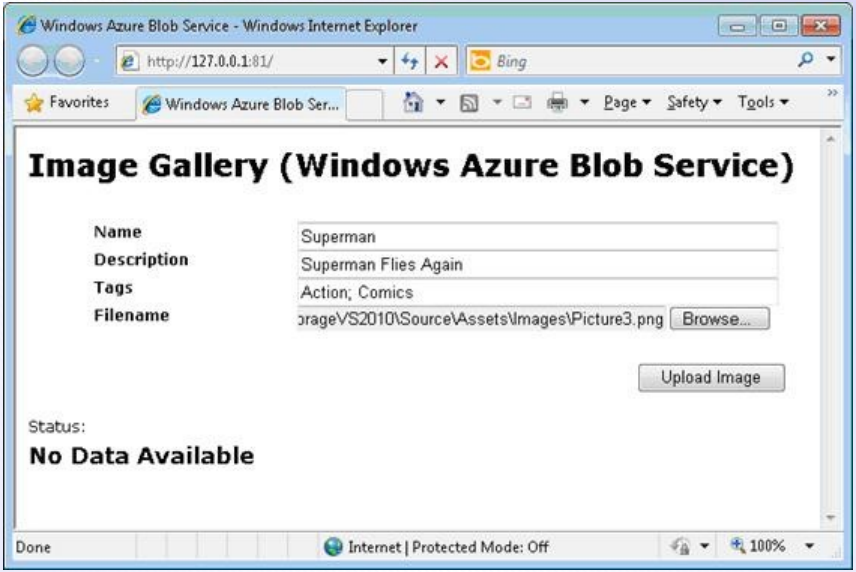


Рисунок 11.39

6. Нажмите **Upload Image** для публикации изображения в веб приложении.

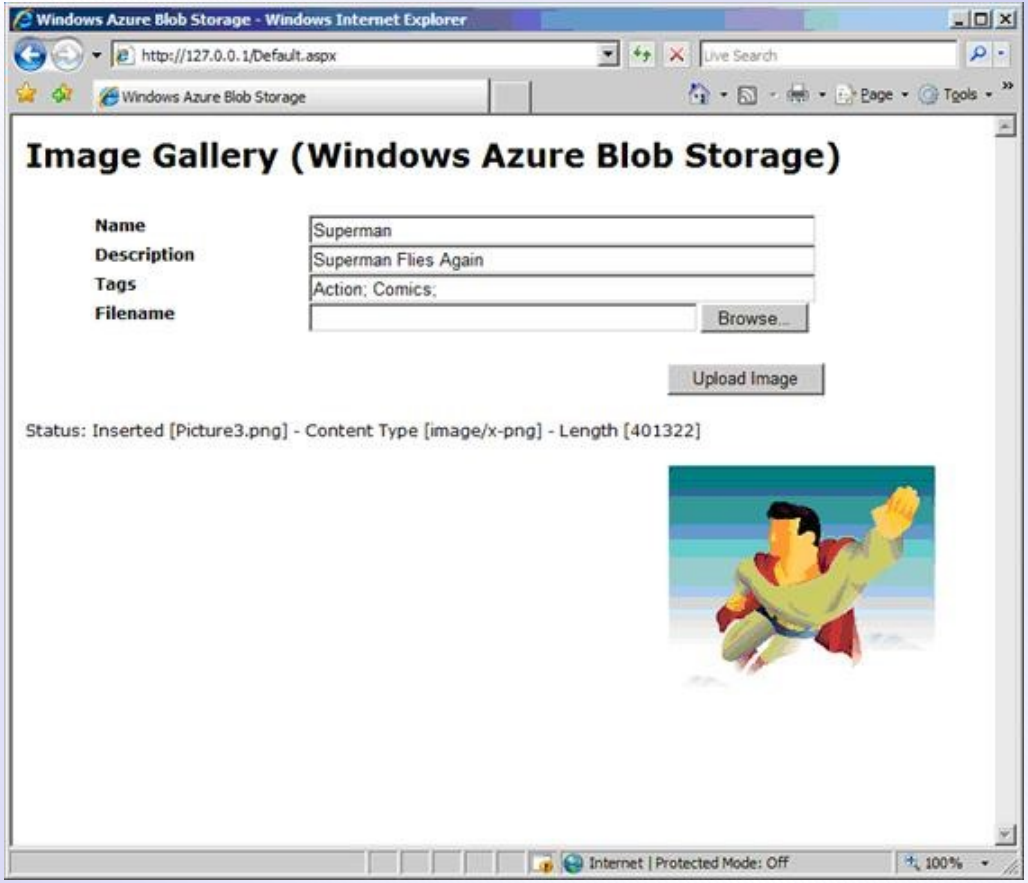


Рисунок 11.40

Извлечение метаданных для Blob в хранилище

1. Откройте **Default.aspx** в режиме **Design**, выберите контрол

imagesListView и в окне свойств нажмите кнопку **Events**.

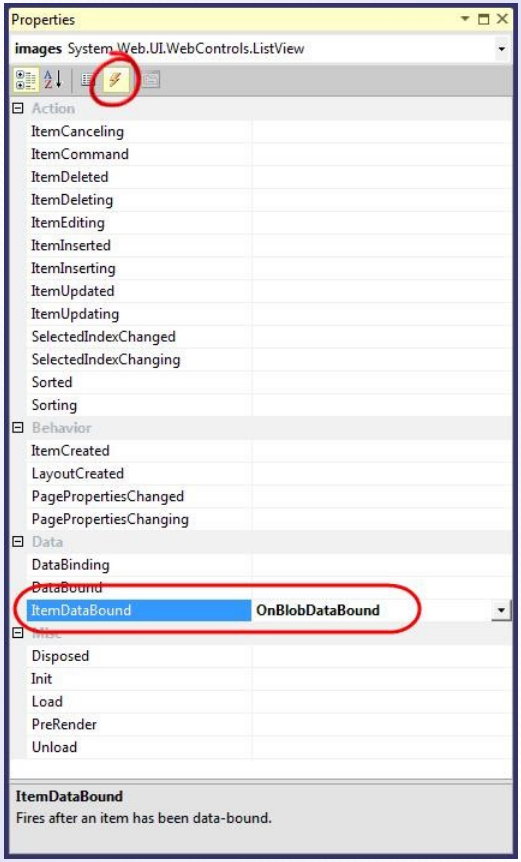


Рисунок 11.41

2. Найдите метод **OnBlobDataBound** и вставьте следующий код

```
protected void OnBlobDataBound(object sender,
          ListViewItemEventArgs e)
 {
  if (e.Item.ItemType == ListViewItemType.DataItem)
   {
     var metadataRepeater = e.Item.FindControl("blobMetadata")
     as Repeater;
     var blob = ((ListViewDataItem)(e.Item)).DataItem
     as CloudBlob;

     // If this blob is a snapshot, rename button to "Delete
     Snapshot"
     if (blob != null)
     {
      if(blob.SnapshotTime.HasValue)
      {
       var delBtn = e.Item.FindControl("deleteBlob")
       as LinkButton;
       if (delBtn != null) delBtn.Text = "Delete Snapshot";

       var snapshotBtn = e.Item.FindControl("SnapshotBlob")
       as LinkButton;
       if (snapshotBtn != null) snapshotBtn.Visible = false;
      }

      if (metadataRepeater != null)
       {
     //bind to metadata
     metadataRepeater.DataSource = from key
```

```
        in blob.Metadata.AllKeys
        select new
            {
             Name = key, Value = blob.Metadata[key]
            };
        metadataRepeater.DataBind();
          }
        }
      }
    }
```

3. Нажмите **F5** для запуска приложения. Убедитесь что отображаются метаданные для изображения, загруженного ранее

Удаление Blob из хранилища

1. Откройте **Default.aspx** в режиме **Source**, найдте **ItemTemplate** для контрола **asp:ListView**. Раскомментируйте код , следующий за контролом **blobMetadata**

```
<div class="item">
<ul style="width:40em;float:left;clear:left" >
<asp:Repeater ID="blobMetadata" runat="server">
<ItemTemplate>
<%# Eval("Name") %><span><%# Eval("Value") %></span>
</ItemTemplate>
</asp:Repeater>

<asp:LinkButton ID="deleteBlob"
OnClientClick="return confirm('Delete image?');"
```

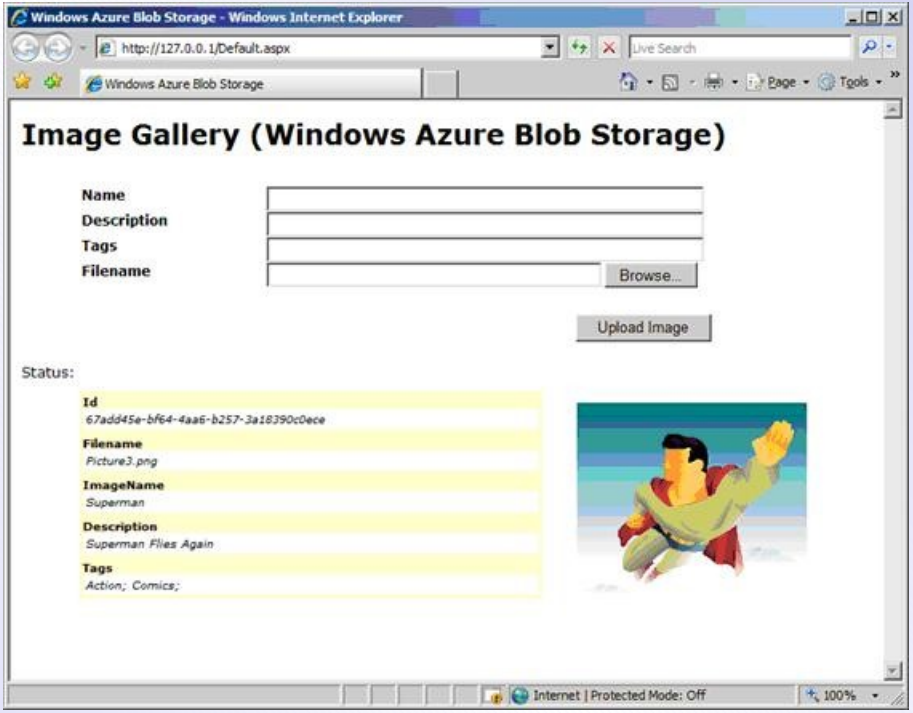


Рисунок 11.42

```
CommandName="Delete"
CommandArgument='<%# Eval("Uri")%>'
runat="server" Text="Delete" oncommand="OnDeleteImage" />

 </ul>
 <img src="<%# Eval("Uri") %>" alt="<%# Eval("Uri") %>"
 style="float:left"/>
```

```
</div>
```

2. Добавьте следующий код в файл **Default.aspx.cs**:

```
protected void OnDeleteImage(object sender, CommandEventArgs e)
 {
  try
  {
   if (e.CommandName == "Delete")
    {
     var blobUri = (string)e.CommandArgument;
     var blob = this.GetContainer().GetBlobReference(blobUri);

     blob.DeleteIfExists();

     RefreshGallery();
    }
  }
 catch (StorageClientException se)
  {
   status.Text = "Storage client error: " + se.Message;
  }
    catch (Exception) { }
 }
```

3. Нажмите **F5** для запуска приложения
4. Добавьте еще несколько изображений и нажмите **Delete** на любом из изображений

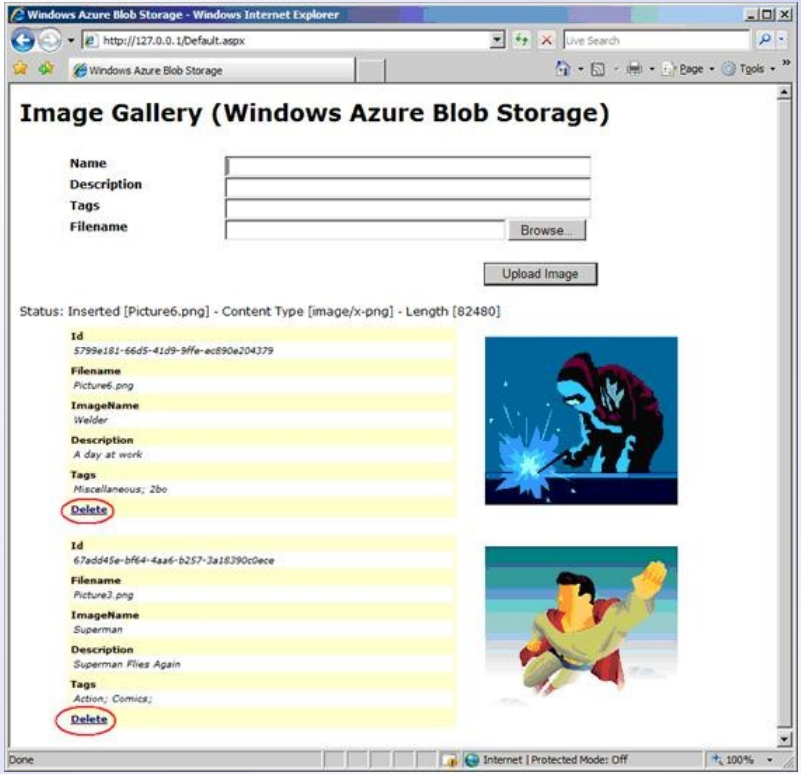


Рисунок 11.43

Копирование Blob

1. Откройте **Default.aspx** в режиме **Source**, найдте **ItemTemplate** для контрола **asp:ListView**. Раскомментируйте следующий код

```
<div class="item">
<ul style="width:40em;float:left;clear:left" >
```

```
<asp:Repeater ID="blobMetadata" runat="server">
<ItemTemplate>
<%# Eval("Name") %><span><%# Eval("Value") %></span>
</ItemTemplate>
</asp:Repeater>

<asp:LinkButton ID="deleteBlob"
    OnClientClick="return confirm('Delete image?');"
    CommandName="Delete"
    CommandArgument='<%# Eval("Uri")%>'
    runat="server" Text="Delete" oncommand="OnDeleteImage" />
<asp:LinkButton ID="CopyBlob"
    OnClientClick="return confirm('Copy image?');"
    CommandName="Copy"
    CommandArgument='<%# Eval("Uri")%>'
    runat="server" Text="Copy" oncommand="OnCopyImage" />

</ul>
<img src="<%# Eval("Uri") %>" alt="<%# Eval("Uri") %>"
 style="float:left"/>
</div>
```

2. Добавьте в файл **Default.aspx.cs**

```
protected void OnCopyImage(object sender, CommandEventArgs e)
 {
  if (e.CommandName == "Copy")
   {
   // Prepare an Id for the copied blob
   var newId = Guid.NewGuid();
```

```
        // Get source blob
        var blobUri = (string)e.CommandArgument;
        var srcBlob = this.GetContainer().GetBlobReference(blobUri);

        // Create new blob
        var newBlob = this.GetContainer().GetBlobReference(
        newId.ToString());

        // Copy content from source blob
        newBlob.CopyFromBlob(srcBlob);

        // Explicitly get metadata for new blob
        newBlob.FetchAttributes(new BlobRequestOptions{
        BlobListingDetails = BlobListingDetails.Metadata});

      // Change metadata on the new blob to reflect this is a
      copy via UI
      newBlob.Metadata["ImageName"] = "Copy of \"" +
      newBlob.Metadata["ImageName"] + "\"";
      newBlob.Metadata["Id"] = newId.ToString();
      newBlob.SetMetadata();

      // Render all blobs
      RefreshGallery();
      }
    }
```

3. Нажмите **F5** для запуска приложения

4. Добавьте еще несколько изображений и нажмите **Copy** на любом

из изображений

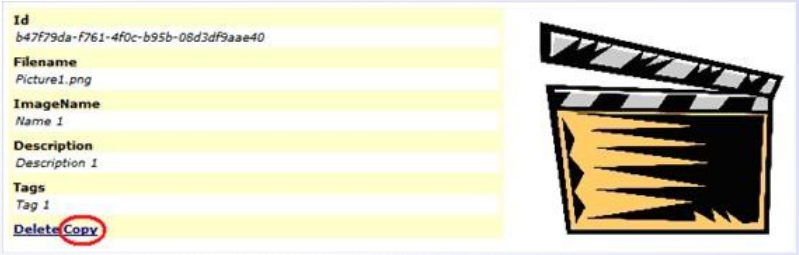


Рисунок 11.44

Получение снимков Blob

1. Откройте **Default.aspx** в режиме **Source**, найдте **ItemTemplate** для контрола **asp:ListView**. Раскомментируйте следующий код

```
<div class="item">
<ul style="width:40em;float:left;clear:left" >
<asp:Repeater ID="blobMetadata" runat="server">
<ItemTemplate>
<%# Eval("Name") %><span><%# Eval("Value") %></span>
</ItemTemplate>
</asp:Repeater>

<asp:LinkButton ID="deleteBlob"
      OnClientClick="return confirm('Delete image?');"
      CommandName="Delete"
      CommandArgument='<%# Eval("Uri")%>'
      runat="server" Text="Delete" oncommand="OnDeleteImage" />
```

```
<asp:LinkButton ID="CopyBlob"
      OnClientClick="return confirm('Copy image?');"
      CommandName="Copy"
      CommandArgument='<%# Eval("Uri")%>'
      runat="server" Text="Copy" oncommand="OnCopyImage" />
<asp:LinkButton ID="SnapshotBlob"
      OnClientClick="return confirm('Snapshot image?');"
      CommandName="Snapshot"
      CommandArgument='<%# Eval("Uri")%>'
      runat="server" Text="Snapshot" oncommand="OnSnapshotImage" />

</ul>
<img src="<%# Eval("Uri") %>" alt="<%# Eval("Uri") %>"
style="float:left"/>
</div>
```

2. В файл **Default.aspx.cs** добавьте

```
protected void OnSnapshotImage(object sender, CommandEventArgs e)
 {
    if (e.CommandName == "Snapshot")
    {
    // Get source blob
    var blobUri = (string) e.CommandArgument;
    var srcBlob = this.GetContainer().GetBlobReference(blobUri);

    // Create a snapshot
    var snapshot = srcBlob.CreateSnapshot();

    status.Text = "A snapshot has been taken for image blob:"
    + srcBlob.Uri + " at " + snapshot.SnapshotTime;
```

```
        RefreshGallery();
        }
    }
```

3. Нажмите **F5** для запуска приложения.
4. Нажмите **Snapshot** на любом из изображений.

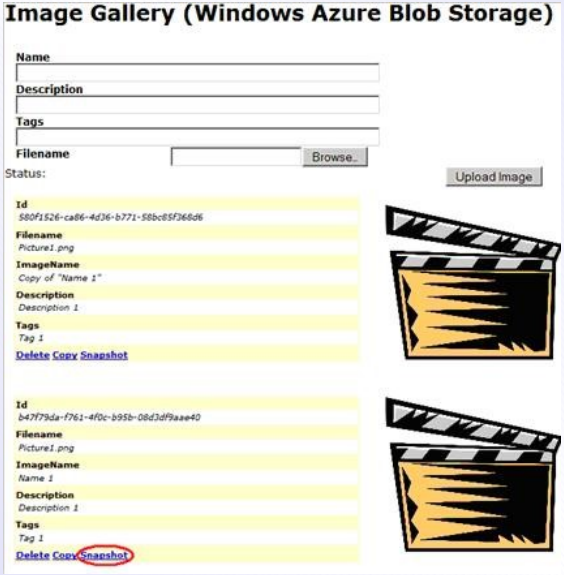


Рисунок 11.45

Лабораторная работа № 6.

«Работа с Tables»

Целью лабораторной работы является практическое освоение процесса работы с *Blob* в *Windows* Azure.

Аппаратура и программные инструменты, необходимые для лабораторной работы:

1. Настольный или портативный компьютер, поддерживающий виртуализацию, операционная система Microsoft Windows XP, Vista, Windows 7.
2. Доступ к сети Интернет.
3. Наличие аккаунта Windows Azure.

Продолжительность лабораторной работы:
2 академических часа

Ход работы:

Настройка Storage Account Settings

1. Откройте меню **Пуск - Все программы - Microsoft Visual Studio 2010 - Microsoft Visual Studio 2010**.
2. В меню **File** выберите **Open** и затем **Project/Solution**. Откройте файл проекта **ExploringWindowsAzureStorageVS2010/Source/Ex01-WorkingWithTables/begin/C#/begin.sln**

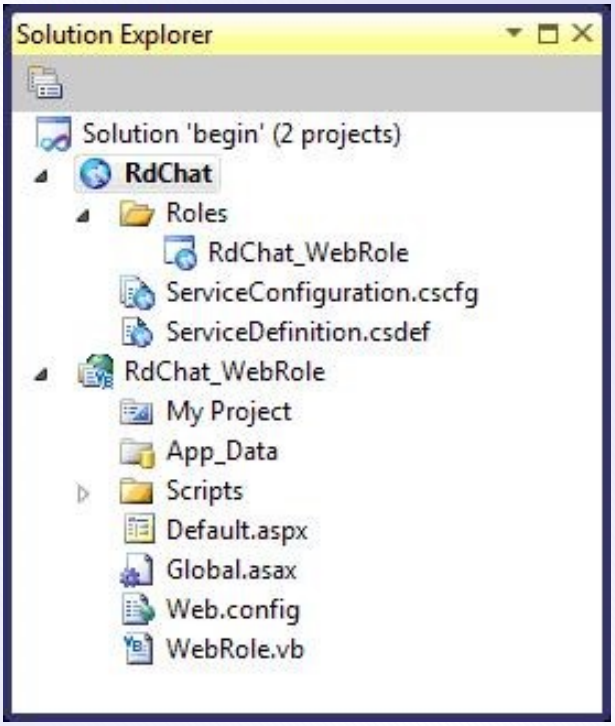


Рисунок 11.46

3. Убедитесь что проект **RdChat_WebRole** запускается по умолчанию.

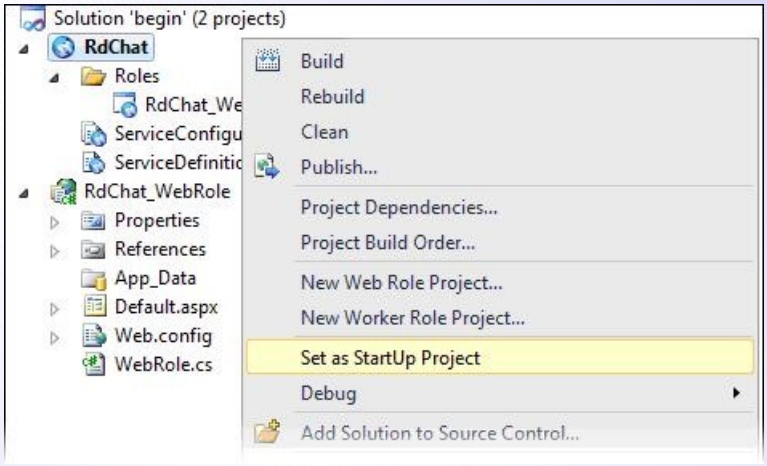


Рисунок 11.47

4. В **Solution Explorer**, в проекте **RdChat** нажмите правой кнопкой по узлу **RdChat_WebRole** и выберите **Properties**

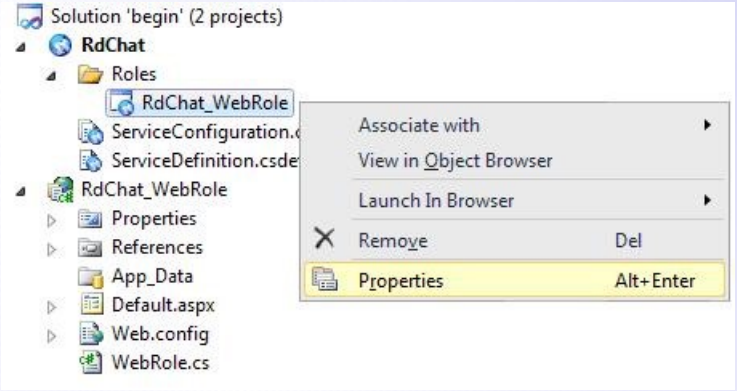


Рисунок 11.48

5. На закладке **Settings** создайте **ConnectionString** с именем **DataConnectionString**. Выберите **Use development storage**

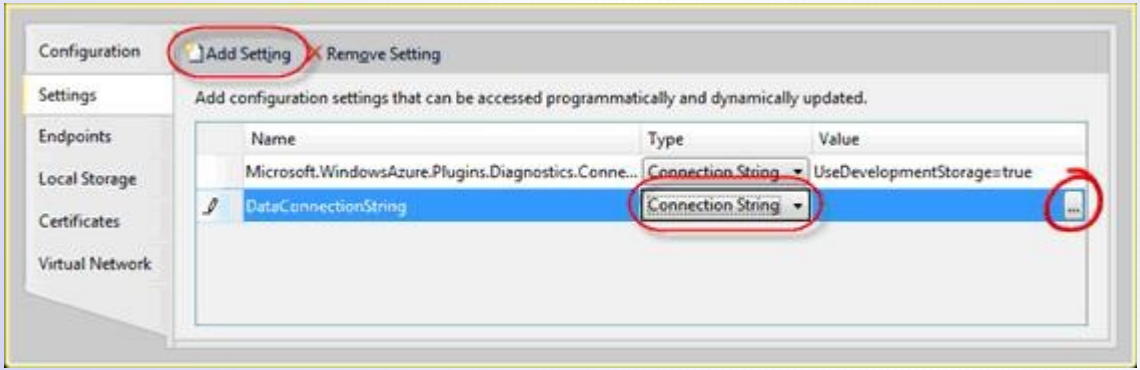


Рисунок 11.49

6. Сохраните изменения

Создание классов для модели Model the Table Schema

1. В **Solution Explorer** нажмите правой кнопкой по проекту **RdChat_WebRole**, выберите **Add Reference**, затем выберите закладку **.NET**, выделите компонент **System.Data.Service.Client** и нажмите **OK**.

2. Нажмите правой кнопкой мыши по **RdChat_WebRole** в **Solution Explorer**, выберите **Add**, затем **Class**. В диалоге **Add New Item** введите имя **Message.cs** и нажмите **Add**.

3. Обновите класс

```
public class Message :
 Microsoft.WindowsAzure.StorageClient.TableServiceEntity
 {
 }
```

4. Добавьте конструктор

```
public Message()
 {
    PartitionKey = "a";
    RowKey = string.Format("{0:10}_{1}", DateTime.MaxValue.Ticks -
    DateTime.Now.Ticks, Guid.NewGuid());
 }
```

5. Добавьте два свойства

```
public string Name { get; set; }
public string Body { get; set; }
```

6. Сохраните изменения в **Message.cs**
7. Нажмите правой кнопкой мыши по **RdChat_WebRole** в **Solution Explorer**, выберите **Add**, затем **Class**. В диалоге **Add New Item** введите имя **MessageDataServiceContext.cs** и нажмите **Add**.
8. Добавьте пространства имен в начало файла

```
using Microsoft.WindowsAzure.StorageClient;
using Microsoft.WindowsAzure;
```

9. Замените объявление класса

```
namespace RdChat_WebRole
 {
  public class MessageDataServiceContext : TableServiceContext
   {
     public MessageDataServiceContext(string baseAddress,
     StorageCredentials credentials) :
     base(baseAddress, credentials)
     {
     }
   }
 }
```

10. Добавьте свойство

```
public IQueryable<Message> Messages
 {
```

```
        get
        {
            return this.CreateQuery<Message>("Messages");
        }
    }

    public void AddMessage(string name, string body)
    {
        this.AddObject("Messages", new Message {
        Name = name, Body = body });
        this.SaveChanges();
    }
```

11. В меню **Build** выберите **Build Solution**.

Создание пользовательского интерфейса Chat

1. В файле **Global.asax.cs** найдите метод **Application_Start** и вставьте следующий код.

```
protected void Application_Start()
{
            ...

    /// Create data table from MessageDataServiceContext
    /// It is recommended the data tables should be only
    created once. It is typically done as a
    /// provisioning step and rarely in application code.
    var account = CloudStorageAccount.FromConfigurationSetting(
    "DataConnectionString");
```

```
        // dynamically create the tables
        CloudTableClient.CreateTablesFromModel(typeof(
        MessageDataServiceContext), account.TableEndpoint.AbsoluteUri,
        account.Credentials);

    }
```

2. Убедитесь что в файле **Global.asax.cs** объявлены пространства имен

```
using Microsoft.WindowsAzure;
using Microsoft.WindowsAzure.ServiceRuntime;
using Microsoft.WindowsAzure.StorageClient;
```

3. Разверните узел **RDChat_WebRole** в **Solution Explorer**, откройте меню правой кнопкой мыши на **Default.aspx** и выберите **View Code**. Убедитесь что следующие пространства имен есть в файле **Default.aspx.cs/Default.aspx.vb** using System.Data.Services.Client; using Microsoft.WindowsAzure;
4. В файле **Default.aspx.cs** найдите событие **SubmitButton_Click** и вставьте следующий код

```
protected void SubmitButton_Click(object sender, EventArgs e)
 {
   var statusMessage = String.Empty;

    try
     {
      var account = CloudStorageAccount.FromConfigurationSetting(
      "DataConnectionString");
```

```
            var context = new MessageDataServiceContext(
            account.TableEndpoint.ToString(), account.Credentials);

            context.AddMessage(this.nameBox.Text, this.messageBox.Text);

            this.messageList.DataSource = context.Messages;
            this.messageList.DataBind();
            }
        catch (DataServiceRequestException ex)
            {
            statusMessage = "Unable to connect to the table storage
            server. Please check that the service is running.<br>"
            + ex.Message;
            }

        status.Text = statusMessage;
    }
```

5. Сохраните все изменения. В меню **Build** выберите **Build Solution**.

Проверка:

1. Для запуска приложения нажмите **F5**. Приложение откроется в веб браузере

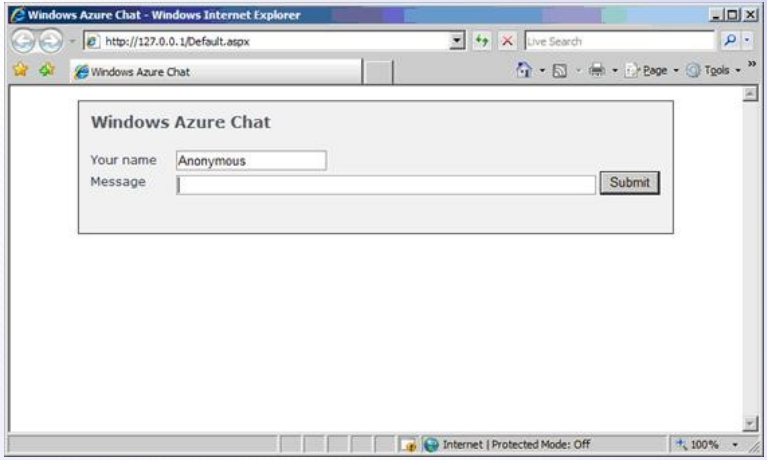


Рисунок 11.50

2. В области уведомлений панели задач нажмите правой кнопкой по значку и выберите **Show Development Storage UI**

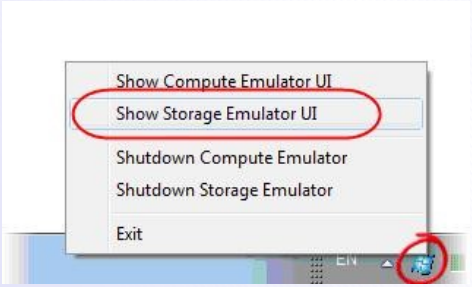


Рисунок 11.51

3. Наберите сообщение и нажмите кнопку **Submit**. Через несколько секунд страница обновится и появится сообщение.

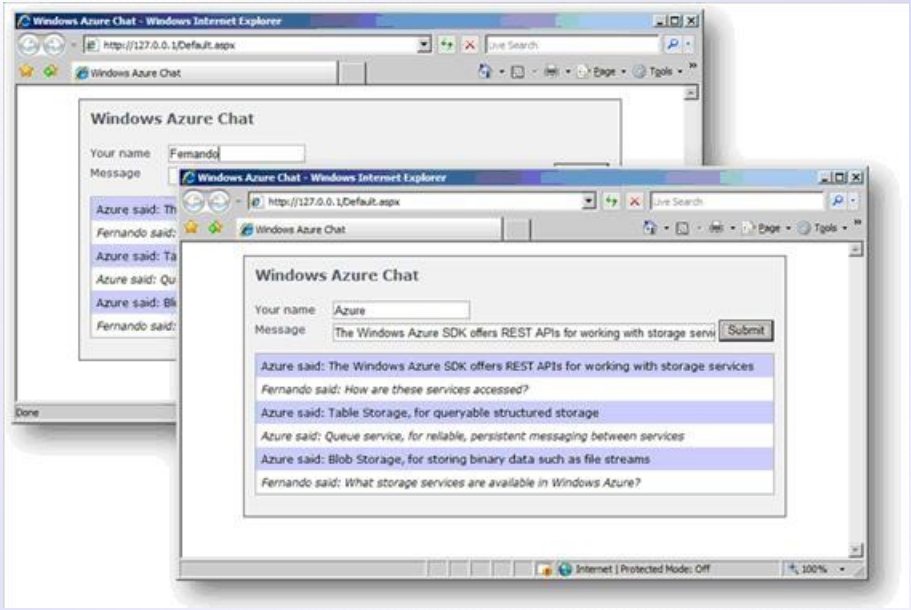


Рисунок 11.52

Лабораторная работа № 7.

«Работа в Windows Live»

Ход работы:

1. Откройте в **Internet Explorer** страницу http://live.com/
2. Пройдите аутентификацию использую **Live ID**, при необходимости зарегистрируйтесь.
3. В верхнем меню наведите выберите **Hotmail**.
4. Создайте учетную запись **Hotmail**.
5. Подключите имеющийся у Вас почтовый аккаунт.
6. В верхнем меню наведите курсор на Hotmail и выберите **Calendar**.
7. Укажите часовой пояс.
8. Создайте новый календарь в меню **New | Calendar**.
9. Создайте новое событие в меню **New | Event**.
10. В верхнем меню наведите курсор на **Messenger**, выберите **Contacts**.
11. Создайте новый контакт в меню **New**.
12. В меню **Office** создайте документы **Word, Excel, PowerPoint**.
13. Дважды выполните изменения в файле.

14. Откройте историю версий, выполните возврат к предыдущей версии документа.
15. Измените настройки общего доступа к файлу.
16. В меню **Photos | Your albums** создайте новый альбом, нажав **Create album**.
17. Загрузите несколько фотографий в созданный альбом.
18. Откройте общий доступ к альбому в меню **Photos | Share photos**.
19. В меню **Windows Live** выберите **SkyDrive**.
20. Создайте учетную запись в **SkyDrive**.
21. Создайте новую папку в **SkyDrive** через меню **New**.
22. Загрузите несколько файлов в созданную папку.
23. Откройте общий доступ к папке, выбрав ее и в меню **Share** выберите **Edit Permissions**.
24. В меню **Profile** выберите **connect**.
25. При наличие, подключите имеющиеся учетные записи **Facebook, YouTube, MySpace, LinkedIn**.

Лабораторная работа № 8.

«Работа в Office Live»

Ход работы:

1. Откройте в **Internet Explorer** страницу http://workspace.officelive.com/
2. Нажмите **Вход** (справа вверху), используйте **Live ID** для входа.
3. Выберите **Новая рабочая область** для создания новой рабочей области.
4. Перейдите в созданную рабочую область.
5. В меню **Создать** выберите **Примечание**, заполните необходимую информацию.
6. В меню **Создать** выберите **Список задач**, заполните необходимую информацию.
7. Добавьте новую строку и новый столбец.
8. В меню **Создать** выберите **Список контактов**, заполните необходимую информацию.
9. Добавьте несколько контактов.
10. В меню **Создать** выберите **Список событий**, заполните необходимую информацию.

11. Добавьте несколько событий.
12. Подключите список в **Microsoft Outlook**, нажав в меню на кнопку **Подключение к Outlook**.
13. Произведите экспорт в **Microsoft Excel** нажав в меню на кнопку **Экспорт в Excel**.
14. Измените настройки общего доступа в меню **Общий доступ**.
15. Создайте несколько комментариев.

Вопросы к зачету

1. Тенденции, разновидности и направления развития облачных вычислений.
2. Аппаратная основа облачных технологий. Развитие вычислительной техники. Современные инфраструктурные решения. Появление блэйд-систем.
3. Аппаратная основа облачных технологий. Появление систем и сетей хранения данных. Топологии SAN. Консолидация ИТ инфраструктуры.
4. Технологии виртуализации. Виртуализация. Ее преимущества.
5. Технологии виртуализации. Виртуальная машина. Виртуализация серверов.
6. Технологии виртуализации. Полная виртуализация. Паравиртуализация.
7. Технологии виртуализации. Виртуализация на уровне ядра ОС. Виртуализация приложений, рабочих мест.
8. Технологии виртуализации. Обзор платформ виртуализации.
9. Понятие облачных вычислений. История появления. Виды облачных вычислений.

10. Инфраструктура как сервис (IaaS). Amazon.
11. Платформа как сервис (PaaS). Microsoft Azure.
12. Программное обеспечение как сервис (SaaS). Архитектурные уровни.
13. Частное облако. Публичное облако. Смешанное (гибридное) облако.
14. Достоинства облачных вычислений. Недостатки и проблемы облачных вычислений.
15. Препятствия развитию облачных технологий.
16. Распределенные вычисления.
17. Коммуникация как Сервис (CaaS). Мониторинг как Сервис (MaaS).
18. Описание и состав Windows Azure SDK.
19. Windows Azure SDK. Роли и решения. Описание классов.
20. Windows Azure SDK. Конфигурация проекта.
21. Компоненты платформы Windows Azure и комплекта средств разработки.
22. Хранилище Windows Azure Storage . Azure Table Services. Модель данных таблицы Windows Azure Table.
23. Хранилище Windows Azure Storage . Windows Azure Blob. Windows Azure Queue. Примеры.

24. Azure Blob Services. Общее представление хранилища Blob, добавление блоков. Сценарий загрузки блоков.
25. Azure Queue Services. Построение приложений для облака с использованием Azure Queue. Модель данных. Пример.
26. Технологии Microsoft .NET Services. Microsoft® .NET Service Bus.
27. Технологии Microsoft .NET Services. Microsoft® .NET Access Control Service.
28. Технологии Microsoft .NET Services. Microsoft® .NET Workflow Services.
29. Технологии Microsoft .NET Services. Создание решения.
30. Office Live Workspace. Управление ресурсами и документами.
31. Приложения Microsoft Word Web App и Microsoft Excel Web App.
32. Интернет-сервис SkyDrive.
33. Пакет услуг Office 365.
34. GoogleApps. Функции, доступные пользователю. Почта и обмен сообщениями.
35. GoogleApps. Календарь. Работа с документами. Стартовая страница и редактор страниц.
36. App Engine. Среда приложений.

37. App Engine. Электронная почта. Memcache. Работа с изображениями. Запланированные задачи.

38. App Engine. Процесс разработки. Квоты и ограничения.

Итоговый тест

Выполните, пожалуйста, этот тест!

Литература

Основная литература

1. Монахов, Д.Н. Облачные технологии. Теория и практика. / Д.Н. Монахов, Н.В. Монахов, Г.Б. Прончев, Д.А. Кузьменков. – МАКС Пресс Москва, МГУ, 2013. – 128 с.
2. Дэвис, К. Шаблоны проектирования для облачной среды. / К. Дэвис. – ДМК, 2020. – 388 с.
3. Бастани Кеннет. Java в облаке. / Бастани Кеннет, Джош Лонг. – Питер, 2019. – 624 с.
4. Сенько, А.В. Работа с BigData в облаках. / А.В. Сенько. – Питер, 2018. – 448 с.
5. Облачное хранилище данных [Электронный ресурс]. 2020. – Режим доступа: http://goo.gl/fhoga. – Дата доступа: 18.02.2020.
6. Чинков, М. Создание инфраструктуры веб-приложений на базе облачных технологий. / М. Чинков. – LAP LAMBERT Academic Publishing, 2017. – 52 с.
7. Лэнгоун, Д. Виртуализация настольных компьютеров с помощью VMware View 5. / Д. Лэнгоун, А. Лейбовичи. – ДМК Пресс, 2017. – 280 с.

8. Сафонов, В.О. Платформа облачных вычислений Microsoft Windows Azure / В.О. Сафонов. – Интернет-университет информационных технологий. Бином. Лаборатория знаний, 2013. – 240 с.

9. Леонов, В. Google Docs, Windows Live и другие облачные технологии / В. Леонов. – Эксмо, 2012. – 304 с.

Дополнительная литература

10. Web 2.0 and Cloud Computing. Веб ресурс. Режим доступа: http://radar.oreilly.com/2008/10/web-20-and-cloud-computing.html. Дата доступа: 18.02.2020.

11. Минаси, М. Windows Server 2012 R2. Полное руководство. Том 2. Дистанционное администрирование, установка среды с несколькими доменами, виртуализация, мониторинг и обслуживание сервера. / М. Минаси, К. Грин, К. Бус, Р. Батлер и др. – Вильямс, 2015. – 864 с.

12. Форум по продуктам Google. Веб ресурс. Режим доступа: https://productforums.google.com/forum/# !forum/ru. Дата доступа: 18.02.2020.