Министерство образования Республики Беларусь

Учреждение образования
«Гомельский государственный технический
университет имени П. О. Сухого»

Кафедра «Информационные технологии»

И. Л. Стефановский

ВВЕДЕНИЕ В ОБЛАЧНЫЕ ВЫЧИСЛЕНИЯ

ПОСОБИЕ

Электронный аналог печатного издания

Гомель 2019

УДК 004.75(075.8)
ББК 32.973.4я73
С79

Рецензент: зав. каф. информационно-вычислительных систем Белорусского торгово-экономического университета потребительской кооперации
д-р техн. наук, проф. *А. Н. Семенюта*;
зав. каф. «Автоматизированные системы обработки информации» Гомельского государственного университета имени Ф. Скорины
канд. техн. наук, доц. *В. Д. Левчук*

Стефановский, И. Л.

С79 Введение в облачные вычисления : пособие / И. Л. Стефановский ; М-во образования Респ. Беларусь, Гомел. гос. техн. ун-т им. П. О. Сухого. – Гомель : ГГТУ им. П. О. Сухого, 2019. – 113 с. – Систем. требования: PC не ниже Intel Celeron 300 МГц ; 32 Mb RAM ; свободное место на HDD 16 Mb ; Windows 98 и выше ; Adobe Acrobat Reader. – Режим доступа: https://elib.gstu.by. – Загл. с титул. экрана.

ISBN 978-985-535-425-4.

Изложены основные темы, изучаемые в курсе «Введение в облачные вычисления». Рассмотрены основные концепции, технологии разработки облачных информационных систем уровня предприятия.

Для студентов специальности 1-40 05 01 «Информационные системы и технологии (по направлениям)» дневной формы обучения.

УДК 004.75(075.8)
ББК 32.973.4я73

ISBN 978-985-535-425-4

Оглавление

Предисловие 5
Глава 1. Введение в облачные информационные системы 7
1.1. Размещение облачных приложений 8
1.2. Основные характеристики облачных вычислений 9
1.3. Облачные вычисления и предоставляемые ими сервисы 11
1.4. Облачные сервисы и границы управляемости 14
1.5. Существующие облачные платформы 14
Глава 2. Платформа *Windows Azure* 17
2.1. Обзор платформы *Windows Azure* 17
2.2. Компоненты облачной платформы 18
2.3. Развитие платформы 27
2.4. Магазин *Windows Azure* 27
Глава 3. Разработка приложений с *Windows Azure Cloud Services* 29
3.1. Создание проекта в *Visual Studio* 30
3.2. Создание модели данных для элементов в *Table Storage* 31
3.3. Создание *web*-роли для отображения гостевой книги 36
3.4. Организация очереди рабочих элементов 40
Глава 4. Авторизация и безопасность с *Windows Azure Active Directory* 43
4.1. Аутентификация на базе утверждений 43
4.2. Федеративная аутентификация в *Windows Azure* 44
4.3. Программная реализация проверки токенов безопасности на стороне клиента 47
4.4. Программная модель *Windows Identity Foundation* 48
4.5. *Windows Azure Access Control Service* 49
4.6. *Active Directory Federation Services 2.0* 52
4.7. Многофакторная проверка подлинности *Windows Azure* 52
Глава 5. Хранение и обработка данных с *Windows Azure Storage* и *Windows Azure SQL Databases* 54
5.1. Блобы 55
5.2. Таблицы 58
5.3. Очереди 61
5.4. Партиции 64
5.5. Избыточность хранилища *Windows Azure Storage* 66
5.6. Диагностика хранилища 68
5.7. Обеспечение безопасности с использованием *Shared Access Signatures* 70
5.8. *Windows Azure SQL Databases* 73

Глава 6. Бизнес-аналитика и анализ данных с *SQL Reporting* и *Hadoop* 80
6.1. Обзор *Business Intelligence* 80
6.2. Создание отчетов в *Report Builder* 82
6.3. Анализ данных с *Hadoop* 86
Глава 7 Доступ к сервисам предприятия с *Windows Azure Service Bus* 94
7.1. Сценарии использования *Windows Azure Service Bus* 98
7.2. *Windows Azure Notification Hubs* 101
7.3. Определение дубликатов сообщений 102
7.4. Шаблон проектирования каналов и фильтров 103
Вопросы для самопроверки 111
Литература 113

Предисловие

Современный специалист в области разработки программного обеспечения (ПО) должен уметь пользоваться технологиями облачных сервисов и владеть основами создания облачных приложений различного уровня сложности.

Дисциплина «Введение в облачные вычисления» знакомит студентов с основными концепциями, технологиями разработки облачных информационных систем (ИС) уровня предприятия.

Целью дисциплины «Введение в облачные вычисления» является подготовка специалиста, владеющего знаниями и практическими навыками по архитектуре, разработке и использованию облачных ИС уровня предприятия.

Основными задачами дисциплины являются следующие:

– изучение студентами теоретических основ и технологий разработки облачных ИС уровня предприятия;

– приобретение студентами практических навыков по проектированию и особенностям использования облачных ИС в сетях на основе технологии *Windows Azure*;

– освоение студентами технологий: применения средств проектирования облачных ИС на основе технологии *Windows Azure*; доступа к БД с использованием *Windows Azure*; разработки *web*-систем на основе технологии *Windows Azure*; технологий удаленного доступа к облачным ИС.

После изучения дисциплины студенты должны иметь представление о: принципах программирования облачных ИС; принципах функционального и логического программирования; различных технологиях создания программных комплексов; перспективах развития технологий программирования облачных ИС.

В результате изучения учебной дисциплины студент должен:

– знать теоретические основы и базовые технологии проектирования и использования современных облачных ИС; технологии организации взаимодействия серверной и клиентской частей облачных ИС на основе технологии *Windows Azure*; технологии организации работы облачных ИС;

– уметь разработать клиент-серверную архитектуру облачных ИС согласно требованиям заданной предметной области; разработать клиентскую и серверную части облачных ИС на основе технологии *Windows Azure*, используя инструментальные средства создания

внешних приложений и реляционную СУБД; организовать работу созданной облачной ИС в многопользовательском режиме;

– владеть методами и технологиями разработки современных облачных ИС на основе технологии *Windows Azure*; технологией и методами конструирования программ на основе поставляемых библиотек и инструментальных средств разработки выбранной платформы; техникой, методами и средствами организации взаимодействия и обработки данных с использованием современных СУБД на основе технологии *Windows Azure*; методами разработки программных приложений в облачной среде; приемами и средствами отладки разрабатываемых программ и систем.

Знания и умения, приобретенные студентами при изучении указанной дисциплины, могут быть использованы при решении различных практических задач.

Глава 1. Введение в облачные информационные системы

Облачные вычисления и технологии являются сегодня одним из ведущих трендов мирового ИТ-рынка. Облачные вычисления представляют собой высокоэффективный инструмент повышения прибыли и расширения каналов продаж для независимых производителей программного обеспечения (*Independent Software Vendors*, *ISV*), операторов связи и *VAR*-посредников, расширяющих возможности существующих продуктов с целью их перепродажи конечным пользователям. Облачный подход позволяет организовать динамическое предоставление услуг, когда пользователи могут производить оплату по факту и регулировать объем своих ресурсов в зависимости от реальных потребностей без долгосрочных обязательств.

Существует большое количество вариантов определения для терминов «облачные вычисления» или «облачная платформа». Это связано с тем, что различные поставщики стараются подчеркнуть уникальность своих предложений и выбирают разные названия, которые зачастую не совсем верно отражают реальную суть предлагаемых сервисов. Когда говорят про облачную платформу, обычно используют такие термины, как «инфраструктура как сервис» (*IaaS*), «платформа как сервис» (*PaaS*) или «приложения как сервис» (*SaaS*).

Облачные вычисления обладают многими преимуществами по сравнению с традиционными решениями для построения инфраструктур предприятий, предложению сервисов и услуг и др. Среди таких преимуществ выделяются:

– гибкость;
– масштабируемость;
– оплата за фактически использованные ресурсы;
– высокая надежность и отказоустойчивость.

Предлагаемые облачные платформы и сервисы сегодня отличаются как по функционалу, так и стоимости. В зависимости от поставленных задач необходимо правильно выбрать поставщика и определить оптимальный план использования.

Корпорация *Microsoft* предлагает свою платформу *Windows Azure*, которая содержит множество сервисов, имеет гибкие планы подписок, поддерживает различные средства и языки разработки приложений. Платформа быстро развивается, и на сегодня она включает в себя более пяти основных видов услуг, от облачного хостинга

web-сайтов до полноценной архитектуры предприятия со множеством сервисов, виртуальными машинами, хранилищами данных и др.

При рассмотрении темы облачных вычислений необходимо рассмотреть следующие концепции [1]:

– размещение облачных приложений;

– основные характеристики облачных вычислений;

– предоставляемые сервисы;

– границы управляемости.

Получив ответы на эти вопросы, можно перейти к рассмотрению существующих платформ и бизнес-моделей, которые они предлагают.

1.1. Размещение облачных приложений

Обсуждая облачные вычисления, следует обращать внимание на то, где располагаются приложения. В настоящее время существует три основных модели расположения приложений:

– в инфраструктуре заказчика;

– у компании-хостера;

– в облаке.

Расположение в инфраструктуре заказчика (*on premises*). Это наиболее традиционная модель развертывания приложений, существующая уже десятки лет. Размещение приложений в локальной инфраструктуре предполагает существенные начальные инвестиции в аппаратные ресурсы, программное обеспечение, сетевую инфраструктуру и персонал.

Такая модель – оплата, приобретение, владение – напрямую связана с высокими капитальными затратами, но в то же время она обеспечивает полный контроль за инфраструктурой, аппаратным и программным обеспечением.

Расположение у компании-хостера (*hosting*). Такая модель развертывания приложений, называвшаяся ранее *Application Services Prodiver* (*ASP*), а затем – *SaaS* или просто «хостинг», получила свое развитие несколько лет назад и является одним из наиболее популярных способов снижения расходов на информационные технологии. Она основана на аренде аппаратной платформы, программного обеспечения, соответствующей инфраструктуры и персонала, выполняющего ее обслуживание. Такая модель отличается меньшим контролем за инфраструктурой, аппаратным и программным обеспечением и ба-

зируется на оплате фиксированного числа ресурсов, что обычно предполагает оплату даже в тех случаях, когда арендуемые ресурсы не используются.

Расположение в облаке (*cloud*). Данная модель появилась совсем недавно. Она предполагает оплату по факту использования арендуемых аппаратных и программных ресурсов, что приводит к существенному снижению начальных расходов и переходу от капитальных инвестиций к операционным расходам. Такая модель отличается практически отсутствием контроля за инфраструктурой и аппаратным обеспечением, а при аренде программного обеспечения – еще и отсутствием контроля за ним.

Каждый подход имеет свои достоинства и недостатки, но, с точки зрения экономики, самой важной характеристикой является оплата по факту использования, реализуемая именно облачными вычислениями.

Облачные вычисления – это такой подход к размещению, предоставлению и потреблению приложений и компьютерных ресурсов, при котором приложения и ресурсы становятся доступны через Интернет в виде сервисов, потребляемых на различных платформах и устройствах. Оплата таких сервисов осуществляется по их фактическому использованию.

1.2. Основные характеристики облачных вычислений

К основным характеристикам облачных вычислений относится масштабируемость, эластичность, мультитенантность, оплата за использование и самообслуживание.

Масштабируемость. Ввод новых продуктов и сервисов, расширение канала продаж и количества заказчиков требуют от информационных систем организации выдерживать растущие нагрузки и обрабатывать большие объемы данных. Быстрая и надежная работа, исключающая отказы в обслуживании, задержки в ответах от системы и сбои, позволяют повысить лояльность и удовлетворенность заказчиков. Масштабируемое приложение позволяет выдерживать большую нагрузку за счет увеличения количества одновременно запущенных экземпляров. Как правило, для одновременного запуска множества экземпляров используется типовое оборудование, что снижает общую стоимость владения и упрощает сопровождение инфраструктуры.

Эластичность. Гибкая реакция на изменяющиеся условия ведения бизнеса является одной из характеристик успешного бизнеса. На-

пример, сложившаяся рыночная конъюнктура и действия конкурентов могут потребовать быстро внедрить новый продукт или услугу, проведя при этом полный цикл планирования, проектирования и разработки ИС. Эластичность позволяет быстро нарастить мощность инфраструктуры, без необходимости проведения начальных инвестиций в оборудование и ПО. Эластичность связана с масштабируемостью приложений, так как решает задачу моментального изменения количества вычислительных ресурсов, выделяемых для работы ИС.

Мультитенантность – это один из способов снижения расходов за счет максимального использования общих ресурсов для обслуживания различных групп пользователей, разных организаций, разных категорий потребителей и др. Мультитенантность может быть особенно привлекательна для компаний-разработчиков приложений, так как позволяет снизить собственные расходы на оплату ресурсов облачной платформы и максимально использовать доступные вычислительные ресурсы.

Оплата за использование. Оплата использованных ресурсов – это еще один атрибут облачных вычислений, позволяющий перевести часть капитальных издержек в операционные. Приобретая только необходимый объем ресурсов, можно оптимизировать расходы, связанные с работой ИС организации. А в сочетании с мультитенантностью, разделяя ресурсы между различными потребителями, можно снизить расходы еще больше. Эластичность позволит быстро изменить объем ресурсов в сторону увеличения или уменьшения, тем самым приведя расходы на ИТ в соответствие с фактическими потребностям организации.

Самообслуживание. Быстрый вывод на рынок нового продукта или услуги в современных условиях сопровождается развертыванием или модификацией ИС. Традиционно развертывание ИС предваряется определением спецификации оборудования, его закупкой и настройкой. В зависимости от того, кем производится процесс разработки приложения (контрактором или внутренними силами), он может потребовать выделения аппаратных ресурсов и установку ПО. Все это может занять длительное время: месяцы и даже годы. Самообслуживание позволяет потребителям запросить и получить требуемые ресурсы за считанные минуты.

Как можно заметить, только сочетание нескольких атрибутов облачных вычислений приводит к достижению задачи повышения доходов и снижения расходов. Так, оплата только использованных

ресурсов максимально эффективна в сочетании с эластичностью инфраструктуры.

Эластичность, в свою очередь, предполагает, что приложения масштабируются, в противном случае быстрое выделение ресурсов не приведет к повышению производительности.

Выше рассмотрели, как основные атрибуты облачных вычислений могут влиять на решение задач повышения доходов и снижения расходов организации. Нужно также понимать, что переход в облако не является тривиальной задачей и часто требует пересмотра и изменения архитектуры существующих решений, а иногда – полного отказа от них в пользу создания новых, реализованных с учетом возможностей, предоставляемых облачными платформами. В зависимости от архитектуры существующих приложений и технологий, на которых они реализованы, их перенос на облачную платформу может привести к получению ряда преимуществ, а может – к появлению дополнительных проблем, связанных, например, с обеспечением совместимости или ограничениями реализации серверной платформы на уровне облака. Как один из шагов по адаптации облачных вычислений, можно рассмотреть переход к архитектуре, ориентированной на сервисы.

1.3. Облачные вычисления и предоставляемые ими сервисы

Облачные вычисления и предоставляемые ими сервисы (например, вычислительные мощности или хранилища) можно сравнить с коммунальными услугами. Так же как в жару или холод меняется потребление воды и электричества, так и потребление сервисов, предоставляемых «облачными» платформами, может возрастать или уменьшаться в зависимости от повышения или понижения нагрузок.

Схожесть сервисов и коммунальных услуг заключается в нескольких аспектах. Во-первых, и в том и в другом случае потребители платят только за реальную утилизацию. Во-вторых, и те и другие ресурсы берутся в аренду – т. е. в большинстве случаев не нужно подключаться к колодцу для получения воды или непосредственно к электростанции для получения электричества – поставщики таких сервисов обеспечивают их доступность в виде арендуемых «ресурсов», оставляя за собой вопросы создания и поддержания инфраструктуры. В-третьих, заключая договор с соответствующей организацией, подразумевается доступность тех или иных ресурсов, а организация – своевременную

оплату их аренды. Можно выделить следующие основные сервисы, предоставляемые облачными платформами.

Программное обеспечение как сервис (*SaaS*). Модель предоставления программного обеспечения как сервиса (*Software as a Service*, *SaaS*) обеспечивает возможность аренды приложений. Программное обеспечение как сервис включает платформу как сервис и инфраструктуру как сервис. Примером приложения как сервиса может быть *Business Productivity Online Suite*.

Модель предоставления программного обеспечения как сервиса является моделью обеспечения доступа к приложениям через Интернет с оплатой по факту их использования. Данная модель является наиболее распространенной на сегодняшний день моделью предоставления облачных сервисов. Организации могут реализовывать подобную модель предоставления сервиса из частных облаков, используя внутренние сетевые каналы, дополнительно защищенные и не связанные с Интернетом.

Потребителями данного типа сервисов являются конечные пользователи, которые работают с приложениями, предоставляемыми в «облаке». Соглашение о предоставлении сервисов (*SLA*) обычно покрывает такие характеристики сервисов, как их доступность (*uptime*) и производительность. Возможности настройки приложений под нужды потребителей минимальны или вообще отсутствуют, их уровень диктуется требованиями рынка или возможностями поставщиков таких приложений.

Оплата конечного сервиса, как правило, производится ежемесячно и рассчитывается на основе количества пользователей приложения.

Платформа как сервис (*PaaS*). Модель предоставления платформы как сервиса (*Platform as a Service*, *PaaS*) обеспечивает возможность аренды платформы, которая обычно включает ОС и прикладные сервисы. Платформа как сервис облегчает разработку, тестирование, развертывание и сопровождение приложений без необходимости инвестиций в инфраструктуру и программную среду. Платформа как сервис также включает и инфраструктуру как сервис. Примером платформы как сервис может служить *Windows Azure*, *Amazon Web Services* (*AWS*).

Здесь потребителями являются сами компании, разработавшие приложения. Платформа обеспечивает среду для выполнения приложений, сервисы по хранению данных и ряд дополнительных сервисов, например, интеграционные или коммуникационные. Соглашение о

предоставлении сервисов (*SLA*) обычно покрывает такие характеристики сервисов, как доступность среды выполнения приложений и ее производительность. Возможности настройки приложений под нужды потребителей практически не ограничены. Ограничением может послужить лишь функциональность сервисов, предоставляемых на уровне платформы. При этом необходимо понимать: для того чтобы воспользоваться возможностями облачной платформы, необходимо значительно модернизировать или вообще написать заново существующие приложения.

Оплата облачной платформы рассчитывается исходя из объема использованных вычислительных ресурсов, таких как:

– время работы приложения;

– объем данных и количество операций с данными (транзакций);

– сетевой трафик.

Провайдер облачной платформы может предоставлять существенные скидки при приобретении определенного объема ресурсов.

Инфраструктура как сервис. Модель предоставления инфраструктуры (аппаратных ресурсов) как сервиса (*Infrastructure as a Service*, *IaaS*) обеспечивает возможность аренды таких инфраструктурных ресурсов, как серверы, устройства хранения данных и сетевое оборудование. Управление всей инфраструктурой осуществляется поставщиком сервисов, а потребитель управляет только ОС и установленными приложениями. Такие сервисы обычно оплачиваются по их фактическому использованию и позволяют пользователю увеличивать или уменьшать объем используемой инфраструктуры через специальные порталы, предоставляемые поставщиками сервисов.

Здесь потребителями являются владельцы приложений, ИТ-специалисты, подготавливающие образы ОС для их запуска в сервисной инфраструктуре. Облачная платформа предоставляет сервисы для запуска виртуальных машин и сервисы хранения данных. Соглашение о предоставлении сервисов (*SLA*) обычно покрывает такие характеристики сервисов, как доступность виртуального сервера, время развертывания образа ОС. В данной сервисной модели могут быть запущены практически любые приложения, установленные на стандартные образы ОС.

Как и в случае с *PaaS*, оплата инфраструктуры как сервиса обычно производится исходя из объема использованных ресурсов.

1.4. Облачные сервисы и границы управляемости

Обсуждая различные типы облачных сервисов – программное обеспечение, платформу и инфраструктуру как сервис, следует обращать внимание на так называемые границы управляемости – т. е. на то, чем, в сравнении с традиционными моделями развертывания в собственной инфраструктуре, можно управлять при переходе на облачную платформу. По понятным причинам инфраструктура как сервис предоставляет большие возможности по настройке отдельных компонентов, тогда как платформа как сервис и ПО как сервис практически минимизируют эти возможности.

При развертывании собственной инфраструктуры необходимо управлять всеми ее компонентами – от сетевых ресурсов до выполняющихся приложений. Тогда как при использовании модели *IaaS* можно контролировать такие компоненты, как среда исполнения кода, безопасность и интеграция, базы данных и др. При переходе к модели *PaaS* все компоненты платформы предоставляются как сервисы с ограниченными возможностями для управления ими. Это сделано, чтобы предоставить в распоряжение потребителей оптимально сконфигурированную платформу, не требующую дополнительных настроек.

1.5. Существующие облачные платформы

На рынке сегодня существует множество платформ для организации облачных вычислений. Существуют как проприетарные (коммерческие), так и открытые (свободные). На основе открытых платформ, таких как *OpenStack* [2], *Cloud Foundry* [3], многие компании создают свои инфраструктуры и предлагают средства для их управления, в частности, предоставляют комплексы для превращения имеющихся ресурсов в облака.

Для того чтобы выбрать наиболее подходящую платформу и провайдера, необходимо четко сформулировать требования, предъявляемые к облаку, а также произвести пробное тестирование всех возможных платформ.

Из наиболее активных и серьезных игроков рынка облачных вычислений следует отметить следующие платформы и компании [4]:

Amazon Web Services [5]

Amazon является пионером рынка облачных платформ, и на сегодняшний момент – это безусловный лидер рынка. Особенность *AWS*

в том, что это инфраструктурный сервис (*IaaS*), который предоставляет максимум свободы разработчикам в выборе платформы и среды разработки. *AWS* подходит как для хостинга корпоративных приложений и контента, так и для построения *SaaS* сервисов.

Rackspace [6]

Rackspace является наиболее близким к *Amazon* (это тоже *IaaS* платформа) и в части стоимости и простоты администрирования – даже обходит своего конкурента. В отличии от *Amazon*, которая концентрирует усилия на развитии инструментов для развертывания и управления облачной инфраструктурой, *Rackspace* стремится быть ближе к прикладным приложениям [7]. Кроме того, *Rackspace* предоставляет базовые сервисы для совместной работы: почтовый сервер (*Rackspace Email*) и файловый сервер (*Rackspace Cloud Drive*), которые можно будет интегрировать в свои облачные приложения.

Windows Azure [8]

Это идеальная облачная платформа для *Microsoft*-ориентированных разработчиков и компаний. Впрочем, *Windows Azure* также поддерживает *PHP*, *MySQL*, *Ruby on Rails*, *Python*, *Java*, *Eclipse* и *Zend*. Главным преимуществом *Azure* перед *Amazon Web Services* и *Rackspace Cloud* является высокий уровень автоматизации. Кроме того, эта платформа позволяет легко интегрировать размещаемые на ней приложения с локальной ИТ-инфраструктурой компании с помощью стандартов *SOAP*, *REST* и *XML* (таким образом, поддерживает схему *S* + *S*).

Данная платформа будет рассмотрена подробнее в следующем разделе.

Google App Engine* + *Google Apps [9]

Платформа *Google App Engine* отличается гуманным отношением к стартапам – предоставляет ограниченные бесплатные ресурсы (дисковое пространство и трафик), которые весьма кстати для начинающих *SaaS*-сервисов. *GAE* поддерживает пока только два языка программирования – *Python* и *Java*. *GAE* в основном ориентирован на создание *SaaS*-сервисов для малого бизнеса. Кроме инфраструктурной платформы, *Google* предоставляет набор *API* для интеграции сервиса с популярными приложениями *Google Apps* и супермаркет приложений *Google Apps Marketplace* для вывода вашего сервиса на рынок.

Force.com [10]

Платформа компании *Salesforce* [11] – *Force.com* – претендует на роль монополиста на рынке корпоративных *SaaS*-приложений. Платформа построена вокруг самой успешной корпоративной *SaaS*-

системы – *Salesforce* и позволяет создавать дополнения к этой системе или независимые приложения. *Force.com* предоставляет широкий выбор инструментов разработки (*Apex*, *Flash*, *Java*), конструктор интерфейсов, готовые модули (аутентификация, социальные инструменты, бизнес-процессы, аналитика) и супермаркет приложений с огромной пользовательской базой.

VMWare vCloud [12]

VMWare vCloud – не является самодостаточной облачной платформой. Это промежуточный слой, который несколько партнеров *VMWare* предоставляют поверх своей серверной инфраструктуры (последней к списку партнеров присоединилась *Salesforce*). *VMWare* – это мировой лидер на рынке систем виртуализации, поэтому главным преимуществом этой платформы является поддержка виртуальных образов приложений. В частности, это позволяет быстро и просто переносить локальные бизнес-приложения на облачную платформу без проблем, связанных с переносом сопутствующей ИТ-инфраструктуры.

IBM Cloud [13]

IBM Cloud в основном ориентирована на крупные компании и ресурсоемкие процессы: разработка и тестирование ПО, хранение и аналитическая обработка огромных массивов данных. Очевидно, после недавнего приобретения сервиса *OmniConnect* еще одной функцией этого облака станет интеграция разрозненных облачных систем и платформ.

Таким образом, на рынке представлено достаточно платформ, чтобы был выбор.

Глава 2. Платформа *Windows Azure*

2.1. Обзор платформы *Windows Azure*

Windows Azure – это открытая и гибкая облачная платформа, которая позволяет быстро выполнять построение приложений, развертывать их и управлять ими в рамках глобальной сети из центров данных, управляемых корпорацией *Майкрософт*. Можно осуществлять построение приложений с помощью любого языка, средства или любой платформы, а также интегрировать общедоступные облачные приложения с существующей ИТ-средой.

Высокая готовность. *Windows Azure* предлагает ежемесячное соглашение об уровне обслуживания на уровне 99,95 %, что позволяет создавать и запускать высокодоступные приложения, не сосредоточивая внимание на инфраструктуре. Эта платформа обладает возможностью автоматического применения исправлений для ОС и служб, встроенной балансировкой сетевой нагрузки и устойчивостью к аппаратным сбоям. Она поддерживает модель развертывания, которая позволяет обновлять приложение с нулевым временем простоя.

Открытость. *Windows Azure* позволяет использовать для построения приложений любой язык, любое средство или любую платформу. Компоненты и службы предоставляются с помощью открытых протоколов *REST*. Клиентские библиотеки *Windows Azure* доступны для нескольких языков программирования, выпускаются по лицензии с открытым исходным кодом и размещаются на сайте *GitHub*.

Неограниченные серверные ресурсы. Неограниченное хранилище. *Windows Azure* позволяет легко масштабировать приложения до любого размера. Это полностью автоматизированная платформа самообслуживания, которая позволяет подготавливать ресурсы к работе за считанные минуты. Гибко расширяйте или сокращайте использование ресурсов в соответствии со своими потребностями. Оплата производится только за ресурсы, используемые вашим приложением. Платформа *Windows Azure* доступна в нескольких центрах обработки данных по всему миру, что позволяет развертывать приложения ближе к клиентам. Сегодня это шесть датацентров, по два на регион (Северная Америка, Европа, Азия).

Расширенные возможности. Платформа *Windows Azure* является гибкой облачной платформой, которая способна удовлетворить любые потребности приложений. Она обеспечивает надежное размещение и масштабирование кода в ролях выполнения приложений. Для хранения данных можно использовать реляционные БД *SQL*, хранилища таб-

лиц *NoSQL*, неструктурированные хранилища больших двоичных объектов, а при необходимости использовать компоненты *Hadoop* и службы бизнес-аналитики для интеллектуального анализа данных. Возможности безопасного обмена сообщениями платформы *Windows Azure* позволяют развертывать распределенные приложения и гибридные решения, работающие в смешанной облачной и локальной среде предприятия. Использование распределенного кэширования или сети кэширующих серверов (*CDN*) позволяет сократить задержку и улучшить временные характеристики приложения во всех точках земного шара.

Платформа *Windows Azure* предоставляет набор сервисов, которые в основной массе схожи с сервисами, используемыми разработчиками «традиционных» приложений.

Вычислительные сервисы. Представляют собой контейнеры для приложений с поддержкой современных технологий разработки, включая *.NET*, *Java*, *PHP*, *Python*, *Ruby on Rails* и нативный код.

Сервисы хранения данных. Масштабируемая распределенная система хранения данных, поддерживающая ряд моделей хранения, включая табличные структуры, бинарные объекты, асинхронные очереди сообщений, традиционные файловые системы и сети распределения контента (*CDN*, *content distribution networks*).

Коммуникационные сервисы. Доступны через облачную сервисную шину и могут использоваться как средство обмена сообщениями или брокер соединений с другими облачными сервисами или сервисами, находящимися у заказчиков.

Сервисы обеспечения безопасности. Сервисы управления доступом, основанные на политиках, которые поддерживают механизмы федерации и позволяют интегрироваться с существующими системами управления идентификацией.

Прикладные сервисы. Компоненты и сервисы, которые могут использоваться для разработки облачных приложений и прикладных сервисов.

2.2. Компоненты облачной платформы

Платформа *Windows Azure* состоит из следующих основных компонентов:

– *web*-сайты;
– виртуальные машины;
– мобильные службы;
– облачные службы;

– большие объемы данных (хранилища);

– мультимедиа.

Для каждого компонента возможны свои сценарии использования, причем они могут включать в себя несколько компонентов.

2.2.1. Web-сайты

Web-разработка является одним из самых быстрорастущих трендов. Развитие Интернета и технологий, обеспечивающих доступ к нему, требует новых средств и моделей для развертывания сайтов и обеспечения их высокой доступности и надежности. Традиционные хостинги остаются популярными и постоянно обновляются, при этом предоставляют самые последние версии средств для поддержания сайтов.

Облачные платформы позволяют расширить возможности разработки и предоставляют высокую степень масштабируемости. Они предоставляют качественно новые услуги, которые отличаются большей гибкостью, управляемостью и др. Это в свою очередь позволяет управлять своими затратами и платить лишь за реально необходимые и использованные ресурсы, сокращая издержки. В начале можно начать с небольшого сайта с настройками по умолчанию. Далее, при необходимости, можно подобрать подходящую виртуальную машину под высоконагруженный сайт, увеличить трафик, добавить другие сервисы, такие как кэширование, *CDN*, базы данных *SQL*, хранилище и др.

Для создания сайтов можно использовать языки и приложения с открытым исходным кодом по своему усмотрению, а затем выполнить развертывание с помощью *FTP*, *Git* и *TFS*. Использование *Git* и *TFS* дает возможность настроить автоматическую публикацию сайта после того, как его последняя версия обновляется в системе управления версиями (СУВ). Настройка непрерывной интеграции и развертывания снимает необходимость в ручной сборке, тестировании и размещении. Все это будет выполняться автоматически.

Для создания *web*-сайта можно выбрать два пути:

– выбрать шаблон сайта (из представленных в галерее);

– создать свой сайт (*Quick Create* или *Create With Database*).

В галерее доступно множество видов сайтов и платформ, таких как *WordPress*, *KentikoCMS*, *Orchard CMS* и др. Во многих случаях выбор приложений из существующих обеспечит более быстрое создание необходимого портала, а также предоставит возможности по управлению им.

При создании сайта самостоятельно в панели управления необходимо подготовить виртуальную машину для него, создать БД (в случае

необходимости) и выделить место под хранение. При таком сценарии будет предоставлен экземпляр виртуальной машины, на котором будет развернут сайт. Далее необходимо выбрать способ развертывания или публикации. Среди вариантов можно использовать не только *Git* и *TFS*, но и *Web Deploy* и *FTP Deploy*, доступные в *IDE* после выбора настроек публикации, которые могут быть загружены с портала.

После того как файлы сайта будут загружены, к нему можно получить доступ по адресу, который выдается автоматически в домене третьего уровня (<*your_name*>.*azurewebsites*.*net*) и имеет название вашего сайта (который был задан при создании). В случае необходимости *DNS* имя можно сменить на свое (это доступно для режимов работы, отличных от *Free*, а также требует фиксированной оплаты).

В случае необходимости повышения производительности сайта, увеличения размера БД необходимо выбрать более мощную виртуальную машину, а также БД.

2.2.2. Виртуальные машины

В *Windows Azure* можно легко использовать собственные образы *Windows Server* или *Linux*, а также выбрать образы из коллекции. Это позволяет сохранять полный контроль над образами и поддерживать их в соответствии с бизнес-требованиями. *Windows Azure* также помогает переносить приложения и инфраструктуру, не меняя существующий код, что ускоряет переход *SharePoint*, *SQL Server* и *Active Directory* в облако и экономит время и деньги.

Виртуальные машины следует использовать для получения гибкости, выполнения приложений в облаке и удаленного управления.

Получение гибкости. Виртуальные машины дают приложению мобильность, позволяя перемещать виртуальные жесткие диски (*VHD*) между локальной и облачной средой.

Выполнение приложений в облаке. Если компания использует популярные серверные приложения *Майкрософт*, виртуальные машины помогут применять те же локальные корпоративные приложения и инфраструктуру в облаке. Легко работайте с приложениями, такими как *Microsoft SQL Server*, *Active Directory* и *Microsoft SharePoint Server*.

Удаленное управление. С полным административным доступом можно удаленно подключаться к виртуальным машинам и управлять установленными на них приложениями.

Все виртуальные машины управляются расширенной версией гипервизора (*Hyper-V*) и располагаются в глобальных центрах обра-

ботки данных. Каждая виртуальная машина может иметь различные характеристики – число процессоров, объем памяти, объем хранилища (жесткого диска).

2.2.3. Облачные службы

Предоставляют возможность создания приложений и интерфейсов *API* с высокой доступностью и бесконечной масштабируемостью.

Облачные службы *Windows Azure* позволяют создавать приложения, которые остаются доступными даже во время обновления программного обеспечения и сбоев оборудования.

Соглашение об уровне обслуживания гарантирует степень доступности 99,95 % для приложения.

Каждому новому облачному приложению требуется мощный набор серверных служб. Облачные службы *Windows Azure* предоставляют все, что нужно для создания самых надежных и масштабируемых интерфейсов *API*.

Облачные службы *Windows Azure* предоставляют наиболее эффективную среду для создания самых современных распределенных вычислительных приложений на планете.

2.2.4. Мобильные службы

Windows Azure Mobile Services – набор сервисов, которые призваны облегчить разработчикам мобильных приложений создание и использование серверного бэкенда. Использование облака *Windows Azure* в качестве такого бэкенда позволит получить готовый функционал *push*-уведомлений, сохранения данных в облачное хранилище, аутентификации и авторизации пользователей без необходимости разворачивать собственную инфраструктуру.

Доступ к сервисам доступен из *C#* и *JavaScript*. Команда разработчиков работает над публичным *REST API*, который позволит получать данные и работать с сервисами из любого языка. На сегодня обеспечена официальная инструментальная поддержка *Windows Phone*, *iOS*, *Windows* 10. Также планируется добавление поддержки *Android*.

Сегодня *Windows Azure Mobile Services* предлагает следующий функционал:

– хранение пользовательских данных в облаке;

– аутентификация и авторизация пользователей в облаке;

– прием *push*-уведомлений от облачного сервиса.

Особенности:

– *REST API*, доступ с любого мобильного клиента;

– масштабирование по требованию;

– мониторинг потребления ресурсов и числа запросов в реальном времени;

– реляционное хранилище, поддержка *SQL*-запросов, индексов;

– автоматическое обновление схемы данных;

– разрешения, обработка запросов перед операциями *CRUD*;

– функциональная единая панель управления;

– бесплатно предоставляется 10 экземпляров.

Последнее обновление сервиса предоставило следующие возможности:

Поддержка платформы *iOS* и выпуск отдельного *iOS SDK.* Добавлены новые инструментальные средства для разработки *iOS*-приложений для *iPhone* и *iPad*. Эти инструменты выпущены с открытым исходным кодом под свободной лицензией *Apache* 2.0.

Для разработчиков *iOS*-приложений благодаря новому *SDK* упрощается доступ к сервисам хранения информации и авторизации через сторонние сервисы и сервис *Microsoft Account*. Поддержка *push*-уведомлений пока не доступна в новом *iOS SDK* и появится в скором будущем.

Поддержка сторонних сервисов авторизации: *Facebook*, *Twitter*, *Google.* В дополнение к уже предложенному сервису авторизации *Microsoft Account*, который можно было использовать для своих приложений ранее, в обновлении представлена поддержка сторонних сервисов авторизации: *Facebook*, *Twitter* и *Google*.

Использование *Windows Azure Tables*, *Blobs* и *Service Bus* внутри *Mobile services.* С обновлением сервиса у разработчиков появилась возможность использовать внутри скриптов *Mobile Services* вызовы к другим сервисам облачной платформы: средствам хранения информации *Tables* и *Blobs* и средству интеграции *Service Bus*.

Отправка почтовых и *SMS*-сообщений. В дополнение к использованию облачных сервисов самой платформы из серверных скриптов *Mobile Services* в обновлении добавлен функционал, позволяющий отправлять почтовые сообщения (используя *SendGrid*) и *SMS*-сообщения. Можно бесплатно отправлять до 25000 почтовых сообщений в месяц.

Аналогично отправке почтовых сообщений в обновлении появилась возможность отправлять *SMS*-уведомления. Для этого используется сервис *Twilio*, который предлагает разработчикам *Windows Azure* 1000 бесплатных сообщений.

2.2.5. Данные большого объема

Windows Azure предоставляет множество служб, помогающих управлять данными в облаке, которые называются *Windows Azure Storage*. Каждый сервис подходит для хранения определенного типа данных.

Таблицы – представляют собой структурированное хранилище. Каждая таблица состоит из набора объектов, каждый из которых имеет набор названий свойств и их значений. Один объект может иметь до 256 свойств. Таблицы распределены таким образом, чтобы максимально поддерживать балансировку нагрузок.

Бинарные объекты – используются для хранения больших бинарных объектов (файлов). Предоставляется простой интерфейс для хранения именованных файлов вместе с метаданными и обеспечивается поддержка сети распределения контента. Бинарные объекты располагаются в контейнерах, каждый из которых содержит набор объектов.

Бинарные объекты могут быть двух видов – блочные, оптимизированные для потокового обмена данными, и страничные, оптимизированные для случайных операций ввода/вывода. Размер блочного бинарного объекта не может превышать 200 Гб, а размер страничного бинарного объекта – 1 Тб.

***BLOB*-объекты** – это простейший способ хранения больших объемов неструктурированных текстовых или двоичных данных, таких как видео, музыкальный файлы и изображения. *BLOB*-объекты – это управляемая служба, сертифицированная по стандарту *ISO* 27001, которая может автоматически масштабироваться до объема в 100 ТБ и доступ к которой можно получить практически из любого места с помощью интерфейса *REST* и управляемых *API*.

Очереди – надежное хранилище сообщений. Обычно используется для обеспечения коммуникаций между ролями. Данный сервис оперирует очередями, в которых располагаются сообщения. Допускается использование неограниченного числа очередей, а очереди могут содержать неограниченное число сообщений. Размер же сообщения ограничен 8 Кб.

Диски – тома *NTFS*, доступные для приложений, выполняющихся в инфраструктуре *Windows Azure*. Диски (*Windows Azure Drives*) хранятся как отформатированные под *NTFS* виртуальные диски (*Virtual Hard Drives*, *VHDs*) в страничных бинарных объектах. Так как диски поддерживают сохранение информации, они могут использоваться приложениями, которым необходимо сохранять состояния.

После того как диск *Windows Azure* смонтирован, он доступен программно через стандартные интерфейсы *NTFS*. Использование дисков *Windows Azure* может существенно упростить миграцию существующих приложений на платформу *Windows Azure*.

Рассмотрим несколько примеров, иллюстрирующих сценарии использования некоторых сервисов хранения данных.

Хранилище бинарных объектов. Возможность хранения резервных копий, отчетов и прочего для их быстрого получения в случае необходимости.

Табличное хранилище. Возможность хранения состояний *web*-приложений, например, в случае электронной коммерции – хранение покупательской корзины или текущего состояния заказа.

Очереди. *Web*-приложение может вызывать сервисы, располагаемые на платформе *Windows Azure*, и осуществлять коммуникации между *web*-ролями и прикладными ролями в рамках одного или нескольких приложений.

Диски. За счет поддержки файловой системы *NTFS* могут использоваться сервисами для обеспечения поддержки традиционных файловых операций – чтение/запись, например, для протоколирования операций или сохранения временных данных.

Для хранения реляционных данных, например, при переносе локальной базы данных в облако следует использовать компонент платформы *Windows Azure – SQL Azure*.

SQL Azure – это способ предоставления реляционной базы данных *Microsoft* как сервиса. Данный сервер базируется на технологиях *Microsoft SQL Server* и обеспечивает устойчивую к ошибкам, масштабируемую и мультитенантную базу данных, доступную как сервис. Как и в случае с *Windows Azure*, *SQL Azure* – это не просто хостинг *Microsoft SQL Server*.

Работа *SQL Azure* базируется на компоненте *Cloud Fabric*, который управляет экземплярами базы данных и обеспечивает их развертывание, администрирование, обновление, мониторинг и поддерживает весь жизненный цикл работы с данными. От пользователей требуется только выполнение таких задач, как создание схемы и ее поддержание, оптимизация запросов и управление безопасностью.

Основные компоненты *SQL Azure* показаны на рис. 2.1.

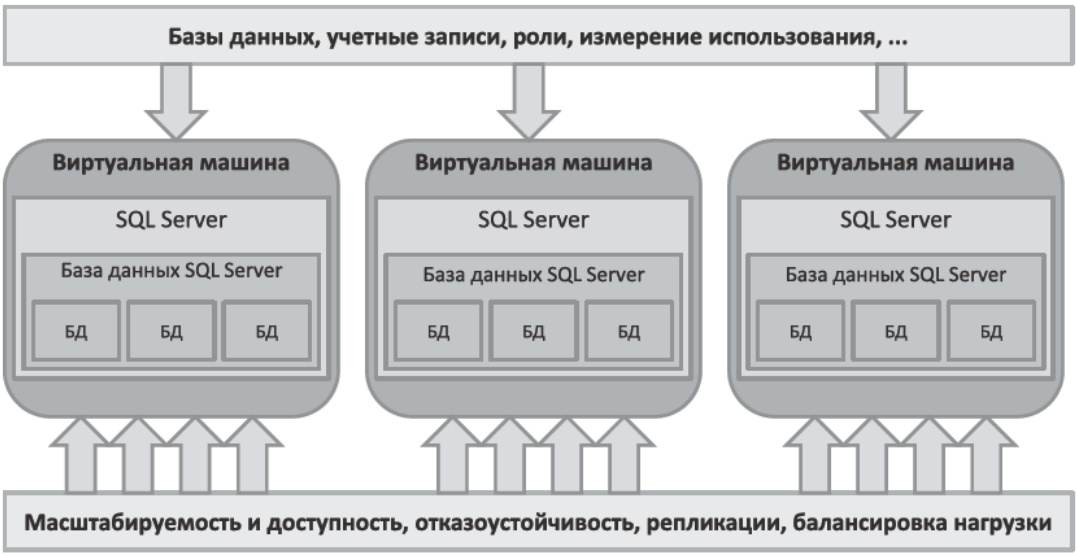

Рис. 2.1. Компоненты *SQL Azure*

Экземпляр базы *SQL Azure* реализован как три реплики в рамках серверной инфраструктуры, поддерживаемой *Cloud Fabric*. Этот компонент обеспечивает высокую надежность, доступность и масштабируемость с помощью автоматической и прозрачной для пользователей репликации и поддержки отказоустойчивости. Также поддерживается балансировка нагрузки и синхронизация инкрементальных изменений во всех репликах данных. *Cloud Fabric* отслеживает все конфликты при изменениях/обновлениях данных, используя двухнаправленную синхронизацию данных между репликами на основе встроенных или задаваемых пользователями политик.

Так как *SQL Azure* построена на основе *SQL Server*, пользователи получают знакомую реляционную модель данных, которая практически симметрична с серверами *SQL Server*, развернутыми у заказчиков. Поддерживаются многие возможности ядра *SQL Server*, хотя в текущей реализации облачной базы данных *SQL Azure* существует ряд ограничений.

Можно выделить четыре основных, высокоуровневых сценария использования *SQL Azure*:

• Использование *SQL Azure* приложениями, которым требуется обеспечение совместной работы пользователей, находящихся внутри и вне «границ» организации.

• Использование *SQL Azure* приложениями, расположенными в инфраструктуре *Windows Azure*.

• Использование *SQL Azure* как основы для создания средств консолидации данных из различных источников и предоставление

этих данных пользователям, разделенным географически и работающим с различными устройствами.

• Использование *SQL Azure* совместно с *web*-приложениями с высокой нагрузкой, использующимися для хранения данных реляционные структуры.

Все эти сценарии решают наиболее часто возникающие перед разработчиками задачи, связанные с выбором в пользу создания приложений, работающих локально, в облаке или гибридных приложений.

2.2.6. Службы мультимедиа

Windows Azure Media Services – это облачное *PaaS*-решение, которое позволяет эффективно строить медиа-сервисы и доставлять медиа-контент вашим потребителям. Решение предлагает набор готовых к применению сервисов, которые позволяют производить быстрое получение медиа-материала, кодирование, конвертирование формата, хранение, защиту контента и доставку видео как в *live*-формате, так и по требованию. *Windows Azure Media Services*, также поддерживают доставку контента на любое устройство или платформу, включая: *HTML5*, *Silverlight*, *Flash*, *Windows* 8, *iPad*, *iPhone*, *Android*, *Xbox* и *Windows Phone*. Кроме того, *Windows Azure Media Services* – это технологии платформы *Microsoft Media Platform*.

Windows Azure Media Services предоставляют следующие сервисы для построения собственных медиа-сервисов и приложений:

– загрузку контента;

– перекодирование;

– конвертацию форматов;

– защиту контента;

– вещание по запросу и живое вещание, а также аналитику и рекламу.

Этот функционал доступен разработчикам через *REST API*, что позволяет создавать решения на базе *Windows Azure Media Services* с использованием любой привычной, удобной или, например, являющейся стандартом в организации технологии. Разработчикам на платформе *.NET* доступен *Windows Azure Media Services SDK for .NET*, который в удобной форме оборачивает предоставляемый *REST API*.

Службы мультимедиа предоставляют ряд встроенных и готовых к использованию компонентов *Майкрософт* и сторонних производителей (от отправки мультимедиа до распределения контента), которые можно объединить для выполнения ваших требований. К числу дос-

тупных возможностей относится отправка, хранение, кодирование данных, преобразование форматов, защита и доставка контента.

Службы мультимедиа можно вызывать отдельно через стандартные *REST API* для облегчения интеграции с внешними приложениями и службами.

2.3. Развитие платформы

Платформа постоянно обновляется и в ней появляются все новые функции. Среди последних новинок и обновлений, последнее из которых было в октябре 2018 г., можно отметить следующее:

– поддержка новой платформы *.NET* 4.7;

– новый портал управления, который вышел из состояния *Preview*, построенный на основе *HTML5* вместо *Silverlight* (портал, который был с самого начала появления платформы);

– *App Store* для дополнительных компонентов платформы (*Addons*);

– обновлен *SDK* до версии 1.8, который в свою очередь приносит поддержку *IIS*8, *C#*5.0, преимущества *.NET* 4.7, *WebSockets*;

– обновления для мобильных сервисов;

– обновления для *web*-сайтов;

– обновления для облачных сервисов.

Одним из направлений развития платформы является предоставление мобильных сервисов, которые позволяют создавать универсальные приложения в облаке, которые будут работать с мобильными платформами. Например, сервис *push*-уведомлений, хранение данных в облаке и др. Самой важной особенностью является то, что *Azure*, по сути, на сегодня является больше платформой для разработчиков и компаний, которые создают свои системы в облаках.

2.4. Магазин *Windows Azure*

Сайт *Windows Azure Marketplace* – это глобальный интернет-магазин приложений *SaaS* и лучших наборов данных. Представив свои приложения для *Windows Azure* на этом сайте, их можно продавать по всему миру. Кроме того, имеется возможность подписаться в своих приложениях на целый ряд лучших наборов данных, представленных на этом сайте, либо распространять собственные наборы данных, извлекая прибыль.

Сценарии использования магазина:

Получение всемирного охвата. Возможность распространения приложений во многих странах и прием оплаты в разных валютах.

Единая модель безопасности. Возможность использования единой модели безопасности, выставления счетов, аудита и проверки подлинности, поддерживающей возможность единого входа с помощью *OAuth v*2.

Простое управление предложениями. Возможность использования отчетов о трафике, подписках и продажах.

Глава 3. Разработка приложений с *Windows Azure Cloud Services*

Размещаемые в *Windows Azure* сервисы состоят из одной или более *web*-ролей (*web role*) и/или прикладных ролей (*worker role*). *Web*-роль является доступным через конечные точки *HTTP* и *HTTPS* *web*-приложением *ASP.NET* и предоставляет пользовательский интерфейс. Прикладные роли, как правило, используются для фоновой обработки данных. Сервисы *Windows Azure* могут включать любые комбинации ролей любого типа, каждая из которых может существовать в одном или более экземплярах. Экземпляры ролей могут добавляться и удаляться без перезапуска приложения, что позволяет масштабировать сервис по мере необходимости.

Сервисы хранения данных *Windows Azure* (*Windows Azure storage services*) включают в себя хранилище двоичных объектов (*Blob services*), табличное хранилище (*Table services*) и очереди (*Queue services*), позволяющие организовывать взаимодействие между службами.

Создадим простое приложение *GuestBook*, в котором демонстрируется ряд возможностей платформы *Windows Azure*, в том числе использование *web*- и прикладных ролей, хранилища двоичных объектов и табличного, а также очередей.

В приложении *GuestBook* *web*-роль используется для создания пользовательского интерфейса, предоставляющего возможность как просмотра содержимого гостевой книги, так и добавления в нее новых записей. Каждая запись включает имя, текстовое сообщение и изображение. В приложении также используется прикладная роль, генерирующая миниатюры для добавленных пользователями изображений.

Когда пользователь добавляет новую запись, *web*-роль загружает соответствующее изображение в хранилище двоичных объектов, после чего добавляет в табличное хранилище новую сущность, содержащую введенную пользователем информацию, и ссылку на изображение в хранилище двоичных объектов. При обращении *web*-роль форматирует данную информацию таким образом, чтобы пользователь мог просмотреть содержимое гостевой книги.

После сохранения изображения и добавления сущности в табличное хранилище *web*-роль помещает в очередь рабочий элемент, указывающий на необходимость обработки изображения. Прикладная роль извлекает рабочий элемент из очереди, извлекает изображение из хранилища двоичных объектов и создает миниатюру – уменьшенный вариант оригинального изображения. Использование очередей

является рекомендованным подходом при организации взаимодействия сервисов в облаке. Преимущества организации слабосвязанных элементов заключаются в возможности их раздельного тестирования и масштабирования.

3.1. Создание проекта в *Visual Studio*

Для создания проекта в *Visual Studio* необходимо выполнить следующие действия:

1. Откройте меню **Пуск** | **Все программы**| ***Microsoft Visual Studio*** | ***Microsoft Visual Studio***.

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.

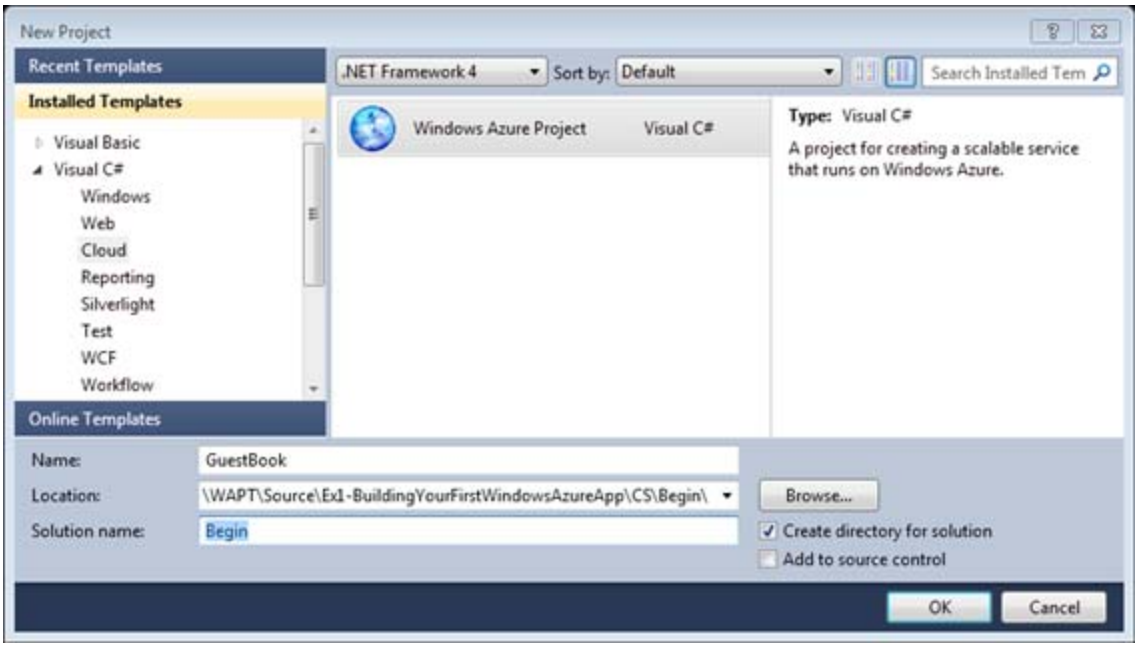

Рис. 3.1. Создание нового проекта *Windows Azure Cloud Service*

5. В диалоге ***New Cloud Service Project*** разверните узел ***Visual C#*** и выберите ***ASP.NET Web Role***. Выберите роль в проекте и введите имя ***GuestBook_WebRole***. Нажмите ***OK*** для завершения.

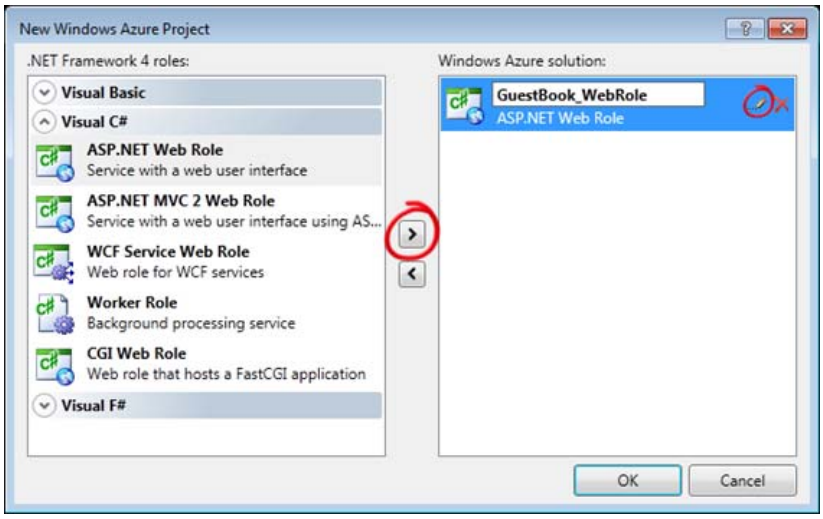

Рис. 3.2. Добавление ролей в проект

6. Обратите внимание на структуру проекта в ***Solution Explorer***.

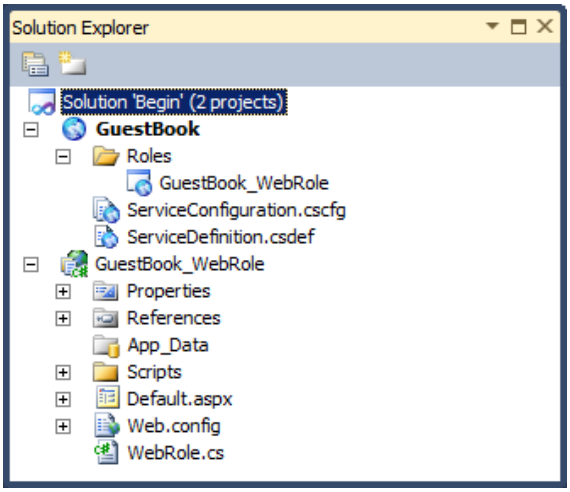

Рис. 3.3. Структура проекта в *Solution Explorer*

3.2. Создание модели данных для элементов в *Table Storage*

Для создания модели данных для элементов в *Table Storage* необходимо выполнить следующие действия:

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 выбран ***.NET Framework 3.5***. Введите имя ***GuestBook_Data*** и нажмите ***OK***.

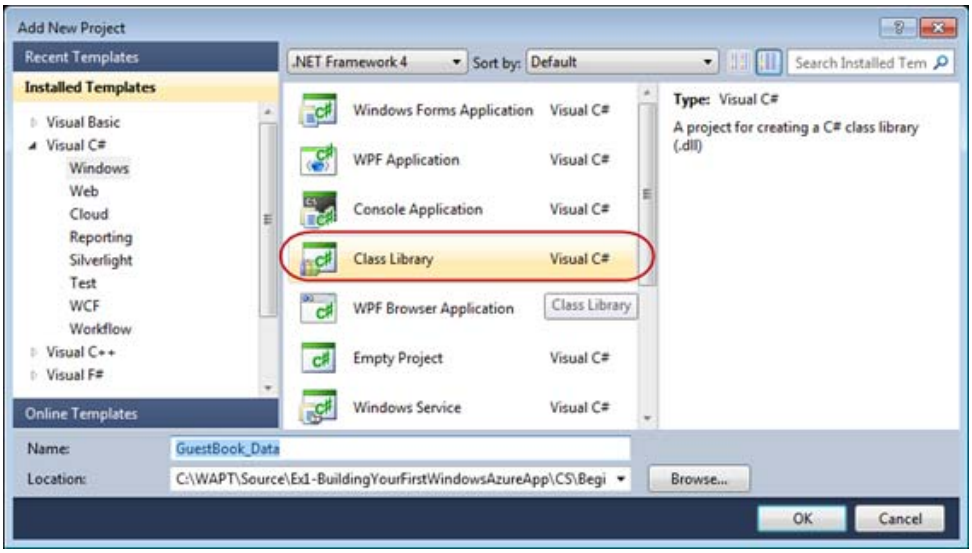

Рис. 3.4. Создание библиотеки классов

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 на библиотеку ***Microsoft.WindowsAzure.StorageClient***.

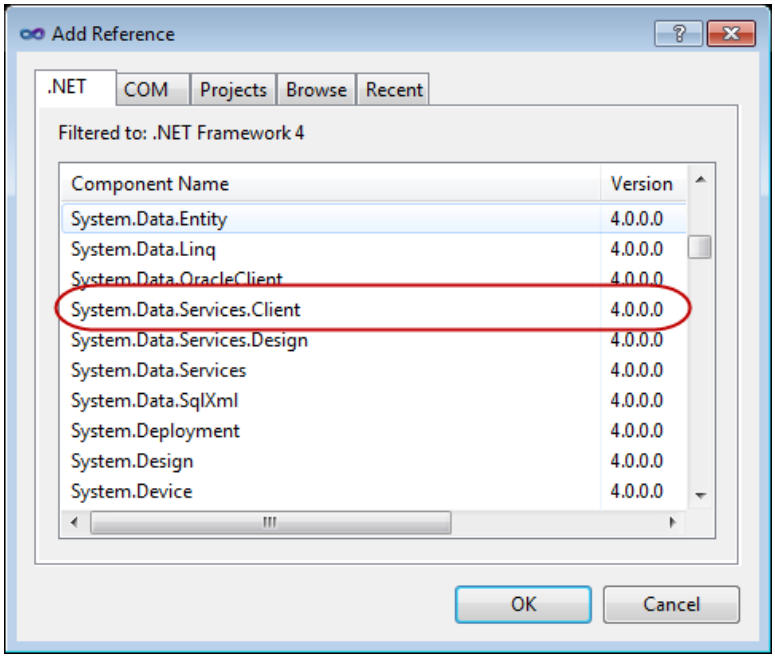

Рис. 3.5. Добавление ссылки на библиотеку

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***.

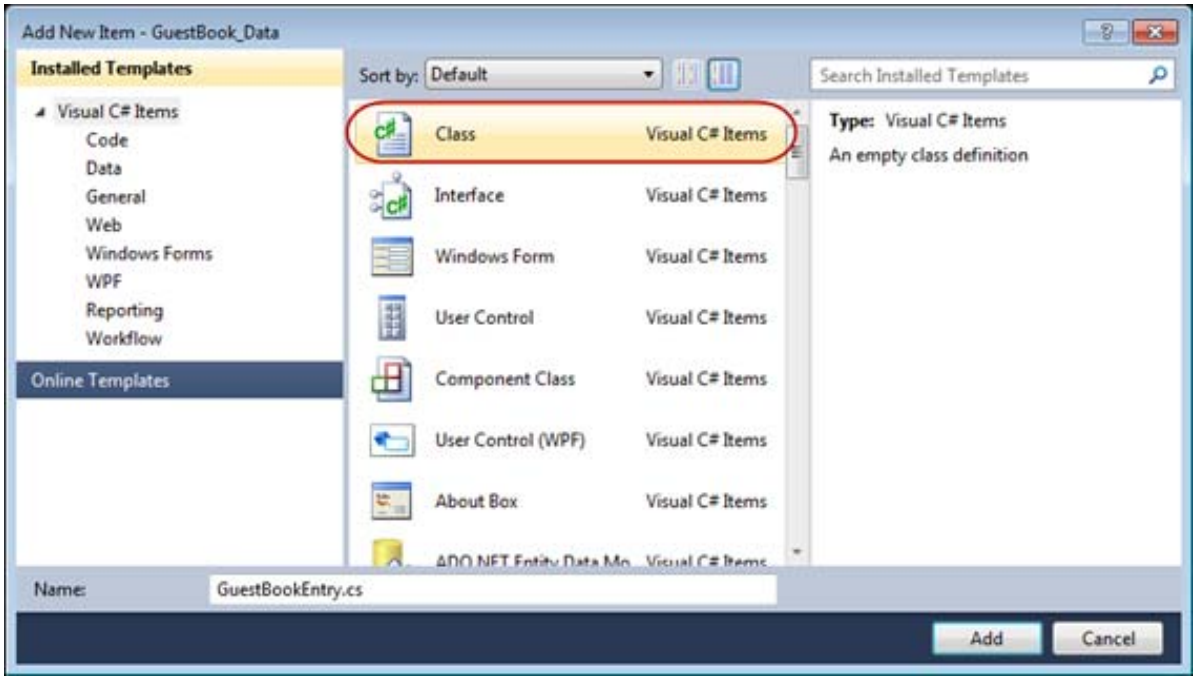

Рис. 3.6. Добавление класса

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***.

3.3. Создание *web*-роли для отображения гостевой книги

Для создания *web*-роли для отображения гостевой книги необходимо выполнить следующие действия:

1. В ***Solution Explorer*** нажмите правой кнопкой по проекту ***GuestBook_WebRole***, выберите ***Add Reference***, затем выберите закладку ***.NET***, выделите компонент ***System.Data.Service.Client*** и нажмите ***OK***.

2. В ***Solution Explorer*** нажмите правой кнопкой по проекту ***GuestBook_WebRole***, выберите ***Add Reference***, затем выберите закладку ***Project***, выделите ***GuestBook_Data*** и нажмите ***OK***.

3. Нажмите правой кнопкой по ***Default.aspx*** и выберите ***Delete***. Нажмите ***OK***.

4. В ***Solution Explorer*** нажмите правой кнопкой по проекту ***GuestBook_WebRole***, выберите ***Add***, выделите ***Existing Item***.

5. В диалоге ***Add Existing Item*** выберите директорию ***\Source\Ex1-BuildingYourFirstWindowsAzureApp\CS\Assets***, выберите ***Add***.

6. 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;
```

7. В классе укажите:

```
private static bool storageInitialized = false;
private static object gate = new Object();
private static CloudBlobClient blobStorage;
```

8. 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();
}
```

9. Обновите метод ***Timer*1_*Tick***:

```
protected void Timer1_Tick(object sender, EventArgs e)
{
 DataList1.DataBind();
}
```

10. Обновите событие ***Page_Load***:

```
protected void Page_Load(object sender, EventArgs e)
{
 if (!Page.IsPostBack)
 {
  Timer1.Enabled = true;
 }
}
```

11. 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;
      }
     }
```

12. В ***Solution Explorer*** разверните узел ***Roles*** в проекте ***GuestBook***. Нажмите два раза по ***GuestBook_WebRole***, откроются свойства данной роли, выберите закладку ***Setting***. Нажмите ***Add Setting***, наберите «*DataConnectionString*» в колонке ***Name***, измените ***Type*** не *ConnectionString* и нажмите ***Add Setting***.

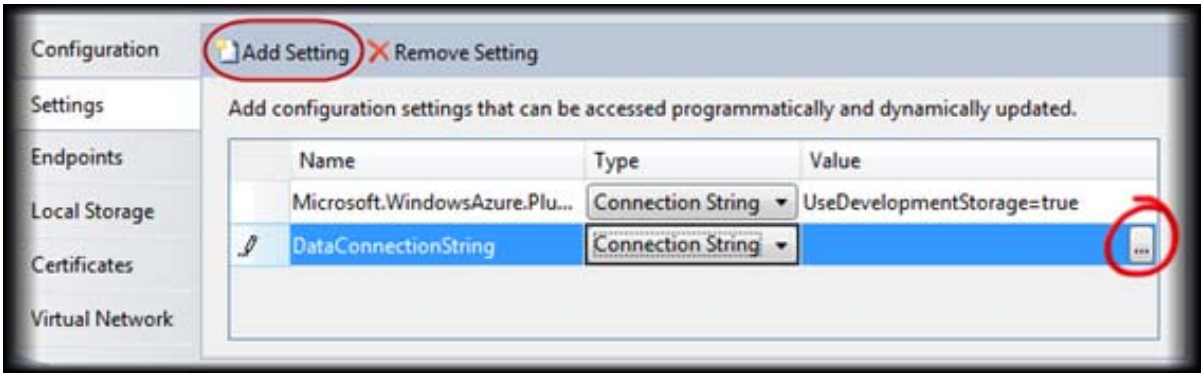

Рис. 3.7. Добавление строки подключения

13. В диалоге ***Storage Connection String*** выберите ***Use development storage*** и нажмите ***OK***.

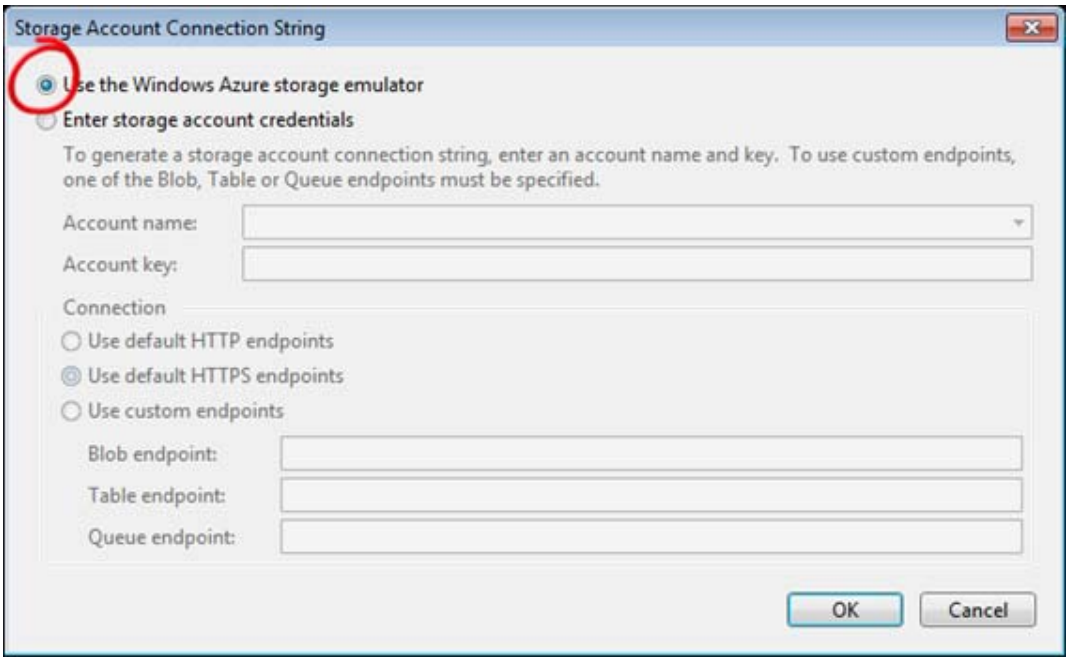

Рис. 3.8. Настройка эмулятора локального хранилища

14. Сохраните изменения.

15. В проекте ***GuestBook_WebRole*** откройте файл *Global.asax.cs*.

16. Объявите пространства имен:

```
using Microsoft.WindowsAzure;
using Microsoft.WindowsAzure.ServiceRuntime;
```

17. 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));
  });
}
```

3.4. Организация очереди рабочих элементов

Для организации очереди рабочих элементов необходимо выполнить следующие действия:

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 ***Initialize Storage*** 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***.

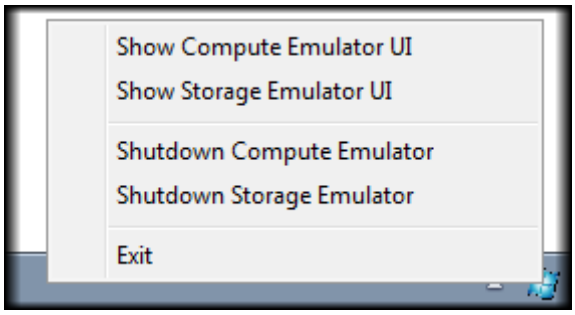

Рис. 3.9. Запуск сервиса

2. Переключитесь на *Internet Explorer* для просмотра приложения *GuestBook*.

3. Добавьте новую запись в гостевой книге.

Рис. 3.10. Главная форма разработанного приложения

Глава 4. Авторизация и безопасность с *Windows Azure Active Directory*

4.1. Аутентификация на базе утверждений

Сервис *Windows Azure Active Directory* предлагает возможность управления процессом аутентификации, доступом к приложениям с помощью аутентификации на основе утверждений. С помощью этого сервиса можно обеспечить единый вход (*Single SignOn*), повышенную безопасность и простое взаимодействие с уже развернутыми в *Active Directory* приложениями, а также выполнить интеграцию приложения с другими популярными провайдерами аутентификации (*Microsoft Account*, *Google*, *Facebook* и др.). *Windows Azure Active Directory* – это полноценная реализация каталога в облаке. При этом сервис поддерживает популярные открытые стандарты обеспечения федеративной аутентификации: *SAML* 2.0, *OData*, *WS-FED*, *OAuth* 2.0/*OpenID*, *JSON Web Token* (*JWT*), *SWT*. Авторизация происходит иначе, *Windows Azure Active Directory* предоставляет возможность аутентификации. Авторизация в этом случае подразумевает под собой определение прав объекта, аутентификация – определение, имеет ли объект право на вход в систему.

Методы обеспечения аутентификации на основе утверждений позволяют реализовать различные сценарии доверия между несколькими системами для осуществления обмена информацией, в том числе конфиденциальной, например, *Email*-адресами или информацией о заказах, используя для этого набор стандартов *WS*-*, описывающих токен *Security Assertion Markup Language*.

Аутентификация на основе утверждений предлагает упрощение процесса аутентификации за счет использования привычных пользователю провайдеров идентификации, таких как *Windows Live ID*, *Facebook* и локальный доменный каталог *Active Directory*.

Любая система, использующая аутентификацию на основе утверждений, пользуется определенным набором терминов. Утверждение является некоторой информацией об объекте, которую предоставляет провайдер идентификации. Например, есть некоторое предложение, что «Уважаемая компания А, мое имя – Александр, фамилия – Иванов, и это подтверждается паспортом, который был выдан компанией Б». В данном случае утверждением будет являться набор данных Имя-Фамилия. Вторая часть о паспорте – это токен

безопасности (*Security token*), объект с электронной подписью, который и содержит утверждение(-я). Третьей частью является провайдер идентификации (*Security Token Service*) – доверенная компания/объект, который имеет право выпускать токены безопасности, содержащие утверждения. «Компания А» – это объект, которому требуется получить токен безопасности для предоставления входа в систему либо доступа к защищенной функциональности системы (*Relying Party*).

В отдельных сценариях может также присутствовать провайдер федерации (*Federation Provider*) – провайдер идентификации и брокер между приложением и личностью в провайдере идентификации (*Facebook*, *Live ID* и др.). Разработчик настраивает аутентификацию таким образом, что конечное приложение, которым пользуется пользователь, доверяет только провайдеру федерации (в контексте отношений приложение-провайдер федерации является провайдером идентификации), провайдер федерации же доверяет набору провайдеров идентификации, также настроенному разработчиком, и в контексте данных отношений провайдер федерации является приложением *relying party*, получающим токен безопасности от провайдеров идентификации. Таким образом, конечное приложение знает только о провайдере федерации, который скрывает от него детали реализации аутентификации с использованием реальных провайдеров идентификации.

4.2. Федеративная аутентификация в *Windows Azure*

Рассмотрим использование федеративной аутентификации в *Windows Azure* с использованием публичных провайдеров идентификации и доменного каталога *Active Directory*.

Для того чтобы лучше понять федеративную аутентификацию с использованием *Windows Azure*, разберем пример с уже настроенной инфраструктурой. В инфраструктуре установлены: домен уровня *Windows Server* 2008 *R*2, *ADFS* 2.0, *WIF*, *WIF SDK* 4.0, *Windows Azure Access Control Service*, приложение в облаке *Windows Azure* с адресом *https*://*techdaysdemo*.*cloudapp*.*net*.

В данном сценарии возможны две ситуации: пользователь входит на свой компьютер, находящийся в домене, и вводит свои учетные данные, либо заходит с компьютера, не находящегося в домене, и, таким образом, должен ввести свои доменные учетные данные.

Различаются ситуации тем, на каком этапе в процесс аутентификации подключается *Active Directory*.

Придя на работу, пользователь ввел свои учетные данные на компьютере в домене. В этот момент происходит взаимодействие с контроллером домена *Active Directory* – проверяются учетные данные, и контроллер домена отправляет пользователю два сообщения – *Client*/*TGS Session Key*, зашифрованный секретным ключом пользователя, и *Ticket Granting Ticket* (который включает в себя служебные данные – сетевой адрес, период жизни тикета и др.), зашифрованный секретным ключом *Ticket Granting Service* (*TGS*, сервиса выдачи тикетов). Пользователь, получив оба сообщения, расшифровывает первое сообщение секретным ключом, сгенерированным из пароля, и получает *Clients*/*TGS Sesion Key*, который используется для дальнейших коммуникаций с *TGS*. Второе сообщение он прочитать не может, так как не знает ключа *TGS*. Если в процессе аутентификации не произошло ошибок, то пользователь становится аутентифицирован в домене.

Через несколько часов пользователь запускает браузер и переходит на *https*://*techdaysdemo*.*cloudapp*.*net*.

Приложение настроено для *Windows Azure Access Control Service* как *Relying Party*. Для приложения *Windows Azure Access Control Service* является провайдером идентификации. Однако *Windows Azure Access Control Service* не является провайдером идентификации по своей сути – для него настроены доверенные отношения с сервером *ADFS* 2.0, расположенным в организации пользователя. *Windows Azure Access ControlService* выступает для *ADFS* 2.0 как *Relying Party*, таким образом *ADFS* выступает в роли провайдера идентификации.

Теперь, когда пользователь заходит на сайт *https*://*techdaysdemo*.*cloudapp*.*net*, его автоматически перенаправляет сначала на *WindowsAzure Access Control Service*, который запускает процесс *Home Realm Discovery*. У *Windows Azure Access Control Service* есть список *HomeRealm*, с которыми у него есть доверенные отношения. Определяющим для *Home Realm Discovery* является аргумент, который предоставляется пользователем в *URL*.

Строка запроса при редиректе (302) на сервис *Windows Azure Access Control Service* выглядит так: *https*://*ahrimanacs*.*accesscontrol*.*windows*.*net*/*v*2/*wsfederation*?***wa***=*wsignin*1.0&***wtrealm***=*https*%3*a*%2*f*%2*ftechdaysdemo*.*cloudapp*.*net*%2*f*&***wctx***=*rm*%3*d*0%26*id*%3*dpassive*%26*ru*%3*d*%252*f*&***wct***=2012-05-07*T*11%3*a*06%3*a*15*Z GET* 302.

Где ***wa*** – описание действия (либо *signin* – вход, либо *signout* – выход), ***wtrealm*** – *URL* доверенного *Realm*, куда направляется токен - приложение, ***wctx*** – контекст, передающийся странице (в данном случае контекст является пассивным). В данном запросе отсутствует дополнительное значение ***wreply***, содержащее указание на то, куда пользователь должен быть перенаправлен с полученным *RSTR*. Часть ссылки до *wsfederation* – точка вхождения для пассивного *STS WS-Federation* в *Windows Azure Access Control Service*.

Далее *Windows Azure Access Control Service* перенаправляет пользователя на сервер *ADFS* 2.0.

На сервер *Active Directory* отправляется выданный пользователю *TGT*, сервер создает *Service Ticket* и передает пользователю. Этот тикет используется в коммуникациях с сервером *ADFS* 2.0. Сервер *ADFS* 2.0 определяет, что пользователь аутентифицирован, и обращается к серверу *Active Directory* за атрибутами для этого пользователя.

На сервере *ADFS* 2.0 происходит создание *SAML*-токена с полученными атрибутами (утверждениями) и его подписание. *SAML*-токен выдается веб-браузеру пользователя для передачи его *Windows Azure Access Control Service*. Теперь *Windows Azure Access Control Service* может определить, что подпись верна, и создать собственный *SAML*-токен путем либо копирования из первичного *SAML*-токена атрибутов, либо с использованием механизма под названием *Rules Engine*, который копирует атрибуты, но позволяет производить над ними манипуляции. *Windows Azure Access ControlService* пересылает свой *SAML*-токен *web*-браузеру, и только после этого *web*-браузер попадает снова в *web*-приложение (с *SAML*-токеном).

На уровне приложения используется *WIF http*-модуль ***WSFederationAuthentificationModule*** **(*FAM*)**, который перехватывает *POST*-запрос от *web*-браузера с токеном и производит проверку этого токена, проверяя список слушателей и дату истечения токена. Список слушателей – *Audience Restriction*, определяется в *web.config* и содержит список *URL*, который может получать приложение.

Рис. 4.1. Федеративная аутентификация в *Windows Azure* с использованием публичных провайдеров идентификации и доменного каталога *Active Directory*

4.3. Программная реализация проверки токенов безопасности на стороне клиента

Проверкой токенов на уровне приложения занимается набор классов *Windows Identity Foundation* (*WIF*), который позволяет создавать приложения, работающие на основе утверждений, и обрабатывать соответствующим образом запросы *WS-Trust*, *WS-Security* и *WS-Federation*, основанный на первых двух.

WIF позволяет рассматривать аутентификацию на основе утверждений как аутентификацию на основе форм – при попытке аутентификации *WIF* перенаправляет запрос на страницу, предоставленную *STS*, где пользователь аутентифицируется и возвращается уже вошедшим в систему к *web*-приложению.

Проверка токенов безопасности происходит с использованием специального *WIF HTTP*-модуля *WSFederationAuthentificationModule* (*FAM*). Модуль перехватывает пришедший в приложение *POST*-запрос, прослушивая событие *AuthenticateRequest*, и, перехватив запрос, начинает проверку токена – периода действия, списка слушателей и целостности токена (используя публичный ключ *STS* для проверки того, является ли доверенным *STS* и не был ли изменен в процессе передачи то-

кен). Модуль разбирает утверждения в токене и использует *HttpContext.User.Identity* для того, чтобы представить *IClaimsPrincipal*, после чего выдает *cookie* для начала сессии. В течение сессии этот процесс не повторяется.

Процесс проверки и разбора токена безопасности на утверждения состоит из следующих этапов:

1. Определение обработчика для токена согласно его типу – *SAML* 1.1, *SAML* 2.0 и др.
2. Создание объекта типа *ClaimsPrincipal* с утверждениями.
3. При необходимости изменения утверждений – вызов *ClaimsAuthentificationManager*.
4. Создание объекта *SessionSecurityToken*.
5. Создание сессии. В процессе утверждения в виде *ClaimsPrincipal* транслируются в коллекцию *cookie FedAuth*[*x*], при этом *cookie* разбиваются на части с целью не превышать определенных лимитов.
6. Перенаправление на *URL*, если он указан.

После завершения всех процедур по проверке и обработке утверждений и токена и перенаправления токена на *web*-приложение запускается модуль *SessionAuthentificationModule*, который перехватывает коллекцию *cookie* и преобразовывает их в объект *ClaimsPrincipal*. Процесс состоит из нескольких этапов:

1. Проверка наличия *cookie*. Если коллекция обнаружена, она используется для создания *SessionSecurityToken*. При этом коллекция расшифровывается и распаковывается.
2. Проверка периода действия токена.
3. Создание объекта *ClaimsPrincipal* с утверждениями.
4. Определение контекста *HttpContext.User* как *ClaimsPrincipal*.

4.4. Программная модель *Windows Identity Foundation*

В модели утверждений несколько пользователей или каких-либо объектов, обладающих утверждениями, могут быть объединены в единую сущность. Интерфейс **IClaimsPrincipalInterface** определяет данные и поведение личностей одного аутентифицированного объекта в рамках текущего контекста выполнения.

IClaimsPrincipal предоставляет метод *IsInRole*, возвращающий булевый тип в зависимости от того, есть ли в утверждениях объекта утверждение типа *Role* и является ли значение этого утверждения

равным чему-то (например, *Domain Admins* в контексте *Windows*-домена). Также интерфейс предоставляет коллекцию личностей, каждая из которых реализует *IClaimsIdentity*. В общем случае в контексте выполнения находится один издатель и один токен, полученный от него, и коллекция *Identity* содержит только один элемент.

ClaimsIdentity определяет базовую функциональность объекта *ClaimsIdentity* и рекомендуется для использования и обеспечения доступа к методам и свойствам *ClaimsIdentity* вместо использования непосредственно *ClaimsIdentity*. Все объекты типа *ClaimsIdentity* реализуют интерфейс *IClaimsIdentity*, который расширяет стандартный *IIdentity*, оставляя при этом базовую функциональность *IIdentity*. *IClaimsIdentity* содержит коллекцию *Claims* – утверждений, привязанных к личности (например, имя или *e-mail*).

Claim определяет свойство аутентифицированного объекта, которое было выпущено провайдером идентификации – например, членство в определенной группе, возраст и др. Содержит поля *Claim*.*ClaimType* (строка, обычно *URI*, на описание типа утверждения. например: *http*://*schemas*.*xmlsoap*.*org*/*ws*/2005/05/*identity*/*claims*/*name* – тип, описывающий значение имени) и *Claim*.*ValueType* (и *Claim*.*Value*), содержащие тип значения и значение, соответственно.

ClaimsPrincipal предоставляет статическое свойство *Current*, которое является *IClaimsPrincipal*, ассоциированным с текущим контекстом выполнения.

4.5. *Windows Azure Access Control Service*

Одним из главных преимуществ аутентификации на основе утверждений является возможность децентрализации – вместо того чтобы возлагать на *STS* обязанность аутентифицировать удаленных пользователей (в компании Б), настраиваются отношения доверия с *STS* в компании Б, что будет означать, что *STS* будет доверять *STS* компании Б в вопросах аутентификации в *Realm* компании Б. Таким образом, можно обойтись без дополнительных учетных данных для удаленных пользователей – они продолжат пользоваться учетными данными своего домена. Преимущества подобного подхода очевидны – архитектура не претерпевает принципиальных изменений, специалистам, занимающимся *STS* в компании А, нет необходимости отслеживать активность учетных данных, для новых *Realm*-ов же просто настраиваются отношения доверия с соответствующими *STS*

(в терминологии федеративной аутентификации настройка доверия заключается в обмене специального формата генерируемыми *XML*-файлами).

Таким образом, ***Windows Azure Access Control Service*** предоставляет сервис для обеспечения федеративной безопасности и контроля доступа к облачным или локальным приложениям. ***Windows Azure Access Control Service*** предоставляет возможность создания облачных и локальных приложений, работающих с множеством провайдеров идентификации, с использованием открытых стандартов и протоколов (*WS-Federation*, *WS-Trust*, *SAML*, *OAuth* 2.0, *SWT*, *JWT*).

Windows Azure Access Control Service состоит из следующих компонентов:

1. ***Security Token Service.*** Набор конечных точек, которые могут выдавать токены для *RP*-приложений, реализовывая федеративную аутентификацию для этих приложений. Поддерживаются самые популярные протоколы и платформы – *.NET Framework*, *WCF*, *Silverlight*, *ASP.NET*, *Java*, *Python*, *Ruby*, *PHP*, *Flash*, *OAth WRAP*, *OAuth* 2.0, *WS-Trust*, *WS-Federation*.

Адреса конечных точек для *Windows Azure Access Control Service STS* можно получить с портала управления *Windows Azure Access Control Service* – для точки *WS-Federation Metadata*, которая используется для интеграции *web*-приложений с *Windows Azure Access Control Service* (эти метаданные могут быть использованы приложением с *WIF*), и для точки *Windows Azure Access Control Service Management Service*, которая используется для программного управления *Windows Azure Access Control Service*.

2. **Портал управления *Windows Azure Access Control.*** Портал, с которого происходит управление пространством имен *Windows Azure Access Control Service*. Предоставляет основную функциональность.

3. **Сервис управления (*Management Service*).** Сервис для программного управления *Windows Azure Access Control Service* с использованием протокола *OData*.

4. **Движок трансформации токена по правилам (*Token Transformation Rule Engine*).** В своей работе *Windows Azure Access Control Service* может манипулировать состоянием утверждений. Основная задача – получить утверждения во входящем токене, обработать их согласно настроенным правилам, выпустить выходящие утверждения и инкапсулировать их в токен для *RP*-приложения.

5. **Провайдеры идентификации.** Токены безопасности для *Windows Azure Access Control Service* предоставляются заранее настроенным набором провайдеров идентификации, которые могут быть как публичными, так и локальными для организации (с использованием *ADFS*).

Windows Azure Access Control Service создает список преднастроенных провайдеров идентификации, из которого разработчик выбирает те, через которые хочет аутентифицироваться, и может создать собственную страницу с этим список и списком правил, по которым *Windows Azure Access Control Service* будет обрабатывать утверждения провайдера идентификации.

При первом запросе на приложения определяется, что пользователь не аутентифицирован и его браузер перенаправляется на *Windows Azure Access Control Service*, который создает и отдает браузеру список доступных провайдеров идентификации, который может содержать любых провайдеров идентификации, от публичных до локальных каталогов *Active Directory*, предоставляющих данные через *ADFS* 2.0. Пользователь выбирает необходимого провайдера идентификации, после чего перенаправляется на страницу входа в систему соответствующего провайдера идентификации. После входа в систему пользователя провайдер идентификации возвращает токен *Windows Azure Access Control Service* с утверждением о том, что пользователь аутентифицирован. *Windows Azure Access ControlService* выпускает утверждения на основе того, что передал ему провайдер идентификации и токен безопасности, после чего перенаправляет пользователя на приложение. Приложение использует полученный от *Windows Azure Access Control Service* токен безопасности для определения набора привилегий пользователя. Таким образом, в реальном режиме меняется приоритет участников – сначала провайдер идентификации (*Live ID*) определяет то, что пользователь имеет право на вход в систему, и возвращает утверждения, после этого приоритет переходит к *Windows Azure Access Control Service*, и браузер пользователя «предъявляет» токен безопасности *Live ID*, и *Windows Azure Access Control Service* создает новый токен безопасности на основе этих же утверждений (режим *copy claim*), при этом *Windows Azure Access Control Service* может в процессе произвести некоторые преобразования утверждений, в том числе, например, добавив некоторые сведения о принадлежности пользователя к определенной группе.

4.6. *Active Directory Federation Services 2.0*

Центральное место в *ADFS* 2.0 занимает сервис токенов безопасности (*Security Token Service*, *STS*), который использует *Active Directory*как хранилище, из которого получаются учетные данные пользователей, *Lightweight Directory Access Protocol* (*LDAP*), *SQL* или собственный источник данных как хранилище атрибутов. *STS* в *ADFS* 2.0 может выдавать маркеры защиты по различным протоколам, в том числе *WS-Trust*, *WS-Federation* и *Security Assertion Markup Language* (*SAML*) 2.0. *ADFS* 2.0 *STS* также поддерживает форматы маркеров *SAML* 1.1 и *SAML* 2.0.

ADFS 2.0 может применяться в нескольких распространенных случаях, например, использование *ADFS* 2.0 в качестве провайдера идентификации. *ADFS* прозрачно интегрируется с платформой *Windows Azure*, позволяя аутентифицировать внутренних пользователей любым набором их учетных данных. С помощью *ADFS* можно реализовать принцип *Single Sign-On* как с собственным приложением, размещенным в облаке, так и интегрировав его с другими продуктами *Microsoft – Office* 365 и *Sharepoint*. *ADFS* использует аутентификацию на основе утверждений, позволяя манипулировать ими – принимать на вход, например, пользователя с ролью администратора (согласно ролевой модели в *Active Directory*) и выдавая на выход в токен утверждение о том, что это – пользователь с уровнем доступа, например *Privileged*. Лицензия на *ADFS* 2.0 автоматически предоставляется владельцу лицензии на *Windows Server*2008 *R*2 *Enterprise Edition*.

Принципиально *ADFS* не отличается от обсужденных выше провайдеров идентификации, добавляя лишь дополнительную инфраструктурную и программную гибкость. Например, если сервер *ADFS* 2.0 расположен во внутренней сети и не может получать запросы извне, можно расположить в демилитаризованной зоне сервер *Federation Service Proxy*, который будет маршрутизировать запросы на утверждения на внутренний *ADFS* сервер (серверы) либо другие провайдеры идентификации.

4.7. Многофакторная проверка подлинности *Windows Azure*

Летом 2013 г. в систему безопасности *Windows Azure* была внедрена новая опциональная функциональность – многофакторная проверка подлинности. Данная функциональность предназначена для за-

щиты учетных записей и облачных сервисов *Microsoft*, решений сторонних компаний или приложений и сервисов, которые используют в качестве системы аутентификации сервис *Windows Azure Active Directory*. Многофакторная проверка подлинности *Windows Azure* предоставляет дополнительный слой проверки подлинности в дополнение к учетным данным пользователя. При этом многофакторную проверку подлинности можно использовать для защиты доступа как к расположенным *on-premise*, так и облачным приложениям. К возможным опциям многофакторной проверки подлинности относятся мобильные приложения, телефонные звонки и текстовые сообщения, при этом пользователи могут выбрать то, что наиболее удобно для них. В контексте защиты локальных приложений многофакторную проверку подлинности можно использовать для защиты *VPN* удаленного доступа и *web*-приложений с помощью специального *SDK*. Если облачное приложение использует *Active Directory Federation Services*, то разработчик может настроить синхронизацию с *Windows Server Active Directory* или другим каталогом *LDAP*. Что же касается других сервисов *Microsoft*, то многофакторная проверка подлинности будет полезна для защиты доступа к *Windows Azure*, *Microsoft Online Services*, *Office* 365 и *Dynamics CRM Online*.

Одной из наиболее сложных задач при разработке новых приложений или создании инфраструктур является, безусловно, обеспечение безопасности. Многослойная безопасность позволяет с определенной степенью уверенности гарантировать, что система не будет взломана с помощью стандартных средств или проблем с кодом. Миграция в облако позволяет решить многие вопросы – многослойная безопасность, которая доступна по умолчанию на платформе, ограничивает возможность внешних злоумышленников к осуществлению, например, *DDOS*-атак, внутренняя архитектура виртуальных машин гарантирует, что, даже если соседняя виртуальная машина была скомпрометирована, она не сможет повлиять на другие и будет изолирована.

Программные средства обеспечения аутентификации и авторизации, такие как *Windows Identity Foundation*, *Active Directory*, *Active Directory Federation Services*, *Windows Azure Active Directory* и *MFA*, обеспечивают еще один слой безопасности, которым может воспользоваться разработчик для предоставления опций пользователю.

Глава 5. Хранение и обработка данных с *Windows Azure Storage* и *Windows Azure SQL Databases*

Сервис платформы *Windows Azure Storage* предоставляет масштабируемое хранилище, доступно как сервис *REST* – таким образом взаимодействие с сущностями, хранящимися в *Windows Azure Storage*, может быть произведено с любой платформы, которая поддерживает *HTTP*, что обеспечивает кроссплатформенность и универсальность.

Сервис *Windows Azure Storage* может выступать как альтернативой, так и дополняющим решением для *SQL Azure Databases*, масштабируемой «облачной» версией *SQL Server*. Каждая сущность в *Windows Azure Storage* хранится в трех экземплярах: сущности, хранящиеся в сервисах таблиц и блобов, записываются в еще один триплет в другом датацентре в этом же регионе. Все сущности трех сервисов хранятся в едином контейнере первого уровня, который называется **аккаунтом хранилища**. На одну подписку *Windows Azure* можно создать до 20 аккаунтов, каждый из которых может содержать до 200 Тб данных. Максимальное количество аккаунтов может быть расширено обращением в техническую поддержку на портале управления *Windows Azure*. Необходимо отметить, что подписка *Windows Azure* имеет собственное ограничение в 10 петабайт данных (200 Тб × максимальное количество аккаунтов на подписку, равное 50).

Каждый аккаунт хранилища защищается от несанкционированного доступа двумя (первичным и резервным) 512-битными ключами доступа, с помощью которых сервис аутентифицирует входящие запросы. Ключи могут быть регенерированы в любой момент по запросу пользователя.

При хранении данных *Windows Azure Storage* использует одну из трех абстракций:

1. **Блобы (*Blobs*, *Binary Large Objects*)** – простые именованные файлы + метаданные. В блобах могут храниться любые бинарные данные: изображения, текстовые файлы и др.

2. **Таблицы** – структурированное хранилище.

3. **Очереди** – сервис-брокер, предоставляющий надежное хранение и доставку сообщений для приложения.

В *Windows Azure Storage* есть еще одна абстракция, которая является надстройкой над хранилищем блобов – диски – долговечные тома *NTFS*, используемые приложениями *Windows Azure*. Диски хранятся в блобах.

Программная модель *Windows Azure Storage* состоит из следующих классов:

– *CloudStorageAccount*: класс, ассоциирующийся с аккаунтом хранилища, является точкой вхождения в сервисы хранилища;

– *Cloud*[*Service*]*Client*: класс, содержащий функциональность, относящуюся к одному из трех сервисов: *CloudTableClient*, *CloudQueueClient*, *CloudBlobClient*;

– *Cloud*[*Object*]: класс, отображающий конкретную сущность одного из трех сервисов, например, *CloudBlob*, *CloudQueue*.

Сервис блобов также имеет дополнительный класс *CloudBlobContainer*, отображающий контейнер блобов, в котором находятся сами блобы.

5.1. Блобы

Сервис блобов в *Windows Azure Storage* имеет простую структуру. Будучи *REST*-сервисом, хранилище предоставляет к каждой из хранимых сущностей гиперссылку, которая состоит из нескольких сегментов:

Аккаунт – весь доступ к хранилищу происходит через аккаунт хранилища, и имя аккаунта является первым сегментом в ссылке на блоб.

Контейнеры блобов аналогичны директории верхнего уровня традиционных файловых систем. Могут содержать только блобы до 100 Тб. Каждый контейнер имеет права доступа: приватный уровень (нужен ключ аккаунта), полное публичное чтение и публичное только чтение. Также внутри каждого аккаунта имеется специальный корневой контейнер $*root*, действующий как корень диска в традиционной файловой системе. К контейнеру можно привязать словарь метаданных, несущих дополнительную информацию.

Блобы, хранящие в себе любые бинарные данные, могут иметь ассоциированные с ними метаданные в виде пар ключ–значение и размером до 8 килобайт на блоб.

Блобы делятся на два типа – блочные и страничные.

Блочные блобы используются для потоковых нагрузок. Блочный блоб выглядит как логическая последовательность блоков, каждый из которых определяется специальным идентификатором и максимальным размером 200 Гб на один блоб. Для блочного блоба используется оптимистичный параллелизм через *ETags*. Модифика-

ция блочного блоба состоит из двух этапов: сначала блоки должны быть загружены в хранилище как неподтвержденные блоки для конкретного блоба, после чего для создания обновленного блоба используется метод *PutBlockList*.

Страничные блобы используются для операций случайного чтения и записи. Страничный блоб выглядит как массив страниц, при этом каждая страница определяется отступом от начала блоба. Размер ограничен 1 Тб на блоб. Для страничных блобов используется либо оптимистичный, либо пессимистичный (блокировка) параллелизм через *Leases*. Обновление происходит сразу же по завершению запроса на запись последовательного набора страниц, поэтому блоки не надо подтверждать (*commit*). Важной особенностью страничных блобов является то, что оплата за хранение страничных блобов производится только за непустые страницы.

К сценариям использования блобов можно отнести:

– доставку статического контента в браузер;

– хранение контента для распределенного и глобального доступа;

– стриминг видео и аудио;

– резервирование данных.

Рассмотрим базовые операции, доступные для управления блобами в хранилище *Windows Azure*:

1. **Синхронное создание контейнера блобов**

Стандартным способом создания контейнера блобов является синхронная операция. Для выполнения этой операции должен быть создан экземпляр класса *CloudBlobClient*, необходимый для предоставления операций, доступных для управления блобами и контейнерами блобов, после чего на основе этого экземпляра получается ссылка на контейнер блобов (ссылка на контейнер может быть получена даже в том случае, если его не существовало ранее). С помощью выполнения метода *CreateIfNotExists* контейнер создается, если его ранее не было. Для создания контейнеров также доступен метод *Create*, однако в случае существования контейнера с таким именем будет выброшено исключение, поэтому с позиции безопасности лучше использовать *CreateIfNotExists*.

При создании контейнера блобов необходимо учитывать следующие ограничения:

– имя контейнера должно начинаться с буквы или цифры и может содержать только буквы, цифры и символы дефиса;

– все буквы должны быть низкого регистра;

– длина имени должна быть от 3 до 63 символов;

– имя не может содержат дефис после точки.

Асинхронный вариант операции создания использует метод *BeginCreateIfNotExist*.

2. **Удаление контейнера блобов**

Для выполнения операции удаления должен быть создан экземпляр класса *CloudBlobClient*, необходимый для предоставления операций, доступных для управления блобами и контейнерами блобов, после чего на основе этого экземпляра получается ссылка на контейнер блобов (ссылка на контейнер может быть получена даже в том случае, если его не существовало ранее). Контейнер может быть удален с помощью метода ***Delete***.

3. **Создание блоба**

Для создания блоба в указанном контейнере достаточно воспользоваться методом *GetBlobReference* объекта контейнера, передав ему имя блоба. После этого будет создан пустой блоб, который необходимо заполнить данными.

4. **Создание и привязка метаданных к контейнеру блобов и блобу**

Блобы могут иметь ассоциированные с ними метаданные. Заголовки метаданных могут быть определены по запросу при создании нового контейнера или блоба, или при «привязке» метаданных к уже существующим ресурсам. Получение метаданных производится с помощью свойства *Metadata*.

5. **Определение прав доступа для контейнера**

Класс *BlobContainerPublicAccessType* является перечислением (*enumeration*), состоящим из следующих возможных значений прав доступа:

– ***Blob*:** публичный доступ на уровне блоба. Анонимные пользователи могут получать контент и метаданные блобов внутри этого контейнера, но не могут получать метаданные контейнера и список блобов внутри него;

– ***Container*:** публичный доступ на уровне контейнера. Анонимные пользователи могут получать контент и метаданные блоба и контейнера и список блобов внутри контейнера;

– ***Off*:** анонимный доступ отключен, только владелец аккаунта имеет доступ к каким-либо ресурсам внутри этого контейнера.

6. **Управление доступом с помощью *Shared Access Signatures* и *Shared Access Policies***

Shared Access Policies и *Shared Access Signatures* могут быть использованы для создания некоторых правил доступа к блобу или контейнеру, регламентирующих права доступа и период доступа.

***Shared Access Policy*:** политика определяет время начала действия политики, время истечения и набор разрешений для *Shared Access Signatures*.

***Shared Access Signature*:** *URL*, предоставляющий доступ к контейнеру и блобу.

5.2. Таблицы

Данные в сервисе таблиц *Windows Azure* хранятся в виде коллекций сущностей. Сущность имеет первичный ключ и набор свойств. Свойства являются парами ключ–значения, что аналогично столбцам. Сервис таблиц в *Windows Azure* не является реляционным, поэтому нельзя выполнять следующие операции:

– создание внешних ключей между таблицами;

– выполнение операции объединения на стороне сервера;

– создание произвольных индексов;

– выполнение таких функций, как, например, *Count*(), на стороне сервера.

Сервис таблиц подходит тогда, когда разработчику не требуется реляционное хранилище или не нужны операции объединения (*joins*) на стороне сервера, а также в тех случаях, когда набор данных не очень велик и операции объединения можно обрабатывать на стороне клиента с помощью *LINQ*. Для доступа к сервису таблиц разработчик может использовать *REST API*, совместимый с *WCF Data Services* (ранее *ADO.NET Data Services Framework*).

Сущность в таблицах состоит из следующих компонентов:

Свойство (столбец) – значение сущности. Имена свойств чувствительны к регистру. Свойства эквивалентны столбцам в классических СУБД, одна сущность может иметь до 255 свойств.

PartitionKey – первое обязательное свойство каждой таблицы, используемое системой для автоматической балансировки нагрузки и распределения сущностей таблицы между серверами.

RowKey – второе обязательное свойство каждой таблицы, являющееся уникальным *ID*-сущности внутри партиции, в которой оно расположено, и являющееся вторым компонентом в комбинации с *RowKey*, уникально идентифицирующей сущность в таблице.

Timestamp – каждая сущность имеет версию, управляемую системой, необходимую для оптимистического параллелизма.

Для того чтобы создать модель в клиентском приложении, которая будет отображаться из хранилища таблиц, необходимо создать класс, реализующий *TableServiceEntity*. Приведенный ниже код содержит определение класса, реализующего *TableServiceEntity*:

```
public class MyClass : TableServiceEntity
    {
     public MyClass()
     {
       base.PartitionKey = "Clients";
       base.RowKey = Guid.NewGuid().ToString();
     }
     public MyClass(string PartitionKey, string RowKey)
     {
       base.PartitionKey = PartitionKey;
       base.RowKey = RowKey;
     }
     public string FName { get; set; }
     public string LName { get; set; }
    }
```

Класс *TableServiceEntity* определяет системные свойства *PartitionKey*, *RowKey* и *TimeStamp*, необходимые для каждой сущности в таблице *Windows Azure*. В примере выше для свойства *PartitionKey* было определено фиксированное значение. В реальной ситуации необходимо выбирать такое значение, которое обеспечит балансировку нагрузки между узлами хранилища, динамическое вычисляемое значение, например, день добавления сущности.

Характерные особенности сервиса таблиц включают в себя:

Порядок сортировки – есть единственный индекс в таблицах, когда таблицы сортируются по сначала *PartitionKey*, после чего по *RowKey*, что означает, что записи, указывающие эти ключи, будут более эффективны и результаты будут сортироваться сначала по *PartitionKey* и потом по *RowKey*.

Типы – *PartitionKey* и *RowKey* должны быть типа *string*, остальные свойства: *Binary*, *Bool*, *DateTime*, *Double*, *GUID*, *Int*, *Int*64, *String*.

Нет фиксированной схемы – в таблицах *Windows Azure Storage* нет схем, поэтому все свойства хранятся в парах ключ–типизированное значение, что означает, что две сущности в одной

таблице могут иметь разные свойства. Например, в таблице может быть два свойства с одинаковым именем, но разными типами для значения свойства. В пределах одной сущности имена свойств должны быть уникальными.

Ссылка на таблицу выглядит стандартно для именования сущностей в сервисах хранилища *Windows Azure*:

http://<*account*>.*table.core.windows.net*/<*TableName*>

Рекомендация: так как используется локальный контекст данных, то таблицы данных должны создаваться только один раз. Обычно этот процесс выполняется на стадии подготовки и реже – в коде приложения. *Application_Start* в классе *Global* является рекомендуемым местом для помещения логики инициализации.

Для использования и управления содержимым таблицы необходимо создать объект типа *CloudTableClient*, предоставляющий следующие базовые методы:

– *CreateTable* – создание таблицы с определенным именем. В случае наличия таблицы с таким именем будет выброшено исключение *StorageClientException*. Наиболее предпочитаемым способом создания таблиц является их создание с помощью класса контекста. ***Пример***:

```
CloudTableClient.CreateTablesFromModel(typeof(MyClassDataCon
text),
        account.TableEndpoint.AbsoluteUri, account.Credentials);
    }
```

– *CreateTableIfNotExist* – создание таблицы с определенным именем только в том случае, если ее не существует;

– *DoesTableExist* – проверка на существование таблицы с определенным именем;

– *DeleteTable* – удаление таблицы и ее содержимого из хранилища. если этой таблицы не существует, будет выброшено исключение *StorageClientException*;

– *DeleteTableIfExist* – удаление таблицы и содержимого из хранилища только в том случае, если она существует;

– *ListTables* – получение списка всех таблиц. Возможно указание префикса для фильтрации имен таблиц;

– *AddObject* – добавление сущности в таблицу.

В том случае если в приложении происходит создание множества сущностей, дешевле (с позиции количества транзакций) и быстрее будет выполнить пакет запросов. Для этого необходимо передать в

SaveChangesWithRetries соответстующий аргумент *SaveChangesOptions.Batch*. При этом необходимо учитывать, что в пакете можно совершать запросы *update*, *delete* и *insert*; один пакет может содержать до 100 сущностей; все сущности в одном пакете должны иметь одно значение *PartitionKey*.

Получение объектов может быть реализовано с помощью *LINQ*-утверждения. Для возвращения всех записей в пределах одной партиции и имеющих определенное значение поля можно воспользоваться следующим *LINQ*-утверждением:

```
CloudTableQuery<MyClass> qry =
  (from e in ctx.CreateQuery<MyClass>("mytable")
   where e.PartitionKey == "Belotserkovskiy" &&
e.FName == "Alexander"
   select e).AsTableServiceQuery<MyClass>();

foreach (MyClass unit in qry)
{
  Console.WriteLine("{0}, {1}", unit.PartitionKey, unit.FName);
}
```

LINQ-утверждения можно также использовать и для любых других операций, в том числе обновления и удаления сущностей.

```
MyClass unit = (from e in ctx.CreateQuery<MyClass>("mytable")
where e.PartitionKey == "Belotserkovskiy" && e.FName ==
"Alexander"
 select e).AsTableServiceQuery<MyClass>().FirstOrDefault();
unit.FName= "Alex";
ctx.UpdateObject(unit);
ctx.SaveChangesWithRetries();
ctx.DeleteObject(unit);
ctx.SaveChangesWithRetries();
```

5.3. Очереди

Механизм очередей *Windows Azure Storage* используется для надежного хранения и дальнейшей доставки сообщений. Очереди *Windows Azure Storage* являются стандартным *Middleware*-обеспечением, который используется для обеспечения однонаправленных коммуникаций между *web*- и *Worker*-ролями в *Windows Azure Cloud Services* – например, *web*-роль кладет сообщения в очередь, которую в бесконечном цикле опрашивает *Worker*-роль и, забрав сооб-

щение, приступает к его обработке. Если экземпляры *Worker*-роли не могут забрать или обработать сообщение, оно будет храниться некоторое время в очереди.

Важным замечанием про очереди является то, что имя очереди должно быть именем в нижнем регистре и *URL-friendly* для удобства использования его в *REST*.

В очередях хранятся сообщения. Сообщения характеризуются следующим:

– очередь может содержать неограниченное количество сообщений;

– сообщения должны быть сериализуемы как *XML*;

– размер ограничен – 64 Кб;

– обычно используется паттерн *work ticket*, суть которого заключается в передаче внутри сообщения не непосредственно сущности, но ссылку на нее и метаданные, содержащие в себе указания для обработки этой сущности;

– сборщик мусора для очереди запускается раз в неделю.

Ссылка на очередь выглядит стандартно для именования сущностей в сервисах хранилища *Windows Azure*:

http://<*account*>.*queue.core.windows.net*/<*QueueName*>

К основным операциям над очередями с использованием клиента облачных очередей *CloudQueueClient* относятся: получение ссылки на очередь (*GetQueueReference*, возвращает объект *CloudQueue*) и получение списка очередей (*ListQueues*).

У объекта *CloudQueue* доступно несколько базовых методов по управлению очередью:

– *SetMetadata*: определяет метаданные для очереди;

– *FetchAttributes*: загружает метаданные и атрибуты;

– *Clear*: очищает очередь;

– *Create*: создает очередь. Если очередь уже существует, будет выброшено исключение;

– *CreateIfNotExists*: создает очередь в том случае, если ее не существует;

– *Delete*: удаляет очередь;

– *Exists*: проверяет существование очереди;

– *AddMessage*: добавляет сообщение в очередь;

– *DeleteMessage*: удаляет сообщение из очереди;

– *GetMessage*: возвращает следующее сообщение из очереди. В это время сообщение становится «невидимым» для других обработчиков;

– *GetMessages*: возвращает определенное количество сообщений из очереди;

– *PeekMessage*: возвращает сообщение из очереди, оставляя его видимым для других обработчиков («подсматривает»);

– *PeekMessages*: возвращает определенное количество сообщений из очереди, оставляя их видимыми для других обработчиков;

– *UpdateMessage*: изменяет содержимое или время видимости сообщения;

– *RetrieveApproximateMessageCount*: Возвращает примерное количество сообщений в очереди. Примерное по той причине, что сообщения могут быть добавлены или удалены после того, как отправляется запрос на количество сообщений очереди.

Усеченный экспоненциальный алгоритм отката. Одним из подходов к опросу очереди и уменьшению нагрузки на сервис является использование усеченного экспоненциального алгоритма отката. Суть данного подхода заключается в том, что каждый «пустой» опрос увеличивает интервал опроса вдвое, не пустой же опрос ставит интервал обратно в 1. Код, приведенный ниже, показывает возможную реализацию данного подхода.

```
while (true)
 {
  var msg = queue.GetMessage();
  if (msg != null)
   {
      string retrievedMsg = msg.AsString;
      //логика
      query.DeleteMessage(msg);

      if (gradualDecrease)
        if (curInterval > intervalFloor) curInterval = curInterval /2;
        else curInterval = intervalFloor;
      else curInterval = intervalFloor;
   }
  else
   {
    if (curInterval <intervalCeiling) curInterval = curInterval * 2;

   }
```

Долгие очереди. Очередь, собирающая сообщения в период отключенного потребителя, называется долгой. В этом случае потребитель очереди может выходить в онлайн раз в день и забирать все сообщения, затем снова отключаться. Полезно для обмена сообщениями с внешними вендорами, а также для сокращения оплаты за время выполнения сервиса (в том случае, если потребитель является, например, *Worker*-ролью в *Windows Azure*).

Разбиение процесса. В качестве рекомендаций можно упомянуть разбиение сложного процесса на состояние, при этом каждое состояние является отдельным процессом-потребителем, опрашивающим определенную очередь. Один большой потребитель обрабатывает одну очередь – является негибким подходом. Архитектура, являющаяся более оптимальной с позиции гибкости – разделенная на различных потребителей и очередей, каждый из которых занимается собственной задачей.

5.4. Партиции

Windows Azure Storage использует специальное системное поле *PartitionKey* для того, чтобы группировать сущности в партиции. Партиции – одно из важнейших понятий на платформе *Windows Azure*, так как именно оно позволяет виртуально неограниченно масштабировать хранилище, логически разделяя данные на группы. Для того чтобы лучше понимать, как работают партиции, рассмотрим принципы работы всего хранилища *Windows Azure Storage*. *Windows Azure Storage* – это система, имеющая трехслойную архитектуру. На первом уровне находится *VIP* (*Virtual IP*), конечная точка, по которой доступна система. Далее все запросы масштабируются на три экземпляра фронтендов (*FE*), серверов, принимающих запросы на сущности. На данном слое происходит логика определения прав, аутентификация клиента и маршрутизация запроса на следующий уровень. Следующий уровень, уровень партиций (*Partition Layer*), управляется серверами партиций и мастерами партиций. Каждому серверу партиций принадлежит определенный набор партиций, за который он «ответственен» – мастер партиций также контролирует загрузку всех серверов. На этом слое происходит балансировка нагрузки между серверами партиций и партициями.

На самом низком уровне находится распределенная файловая система – *Distributed File System* (*DFS*). На этом слое хранятся дан-

ные, доступные всем серверам партиций. *DFS* балансирует нагрузку на систему, распределяя запросы, и состоит из множества узлов, по которым распределены наборы данных (*extents*), причем каждый из наборов данных назначается одному ***primary server*** (первичному серверу) и нескольким ***secondary server*** (вторичным серверам). Каждый экстент имеет данных от 100 Мб до 1 Гб, одна сущность же может быть распределена по нескольким экстентам. При операциях записи сущностей эти операции сначала обрабатываются первичным сервером (но не выполняются), после чего передаются на выполнение операции записи какому-то из вторичных серверов. Первичный сервер в данном случае выполняет роль контроллера, распределяющего операции и следящего за их выполнением. Первичный сервер не подтверждает операцию записи до тех пор, пока как минимум два вторичных сервера не сообщат об успешной записи данных в три домена обновлений и неисправностей.

Данные реплицируются в нескольких экземплярах внутри *DFS* (минимальное количество копий должно быть равно трем). Большие сущности могут быть разбиты на части и распределены по нескольким экстентам, что приводит к тому, что одна сущность может храниться на нескольких дисковых устройствах.

Размер партиции – равен количеству сущностей, которое хранится в партиции. Разреженность значения *PartitionKey* является важным показателем, так как если для всех сущностей используется одно значение *PartitionKey*, то это значит, что будет существовать одна партиция, либо, если значений много, каждая сущность может содержаться в отдельной партиции.

Цели масштабирования по партициям – показатели, которые определяют, сколько может «выдержать» одна партиция, равные 500 сущностям в секунду. Партиции переносятся между серверами хранилища для обеспечения высокой степени эластичности и максимальной производительности. «Горячие», т. е. испытывающие высокую степень нагрузки, партиции могут быть вертикально масштабированы – *Windows Azure Fabric Controller* способен выделить больше ресурсов для партиций с большим количеством транзакций. Таким образом, если на одном сервере расположены три партиции, то если одна из партиций начинает испытывать высокие нагрузки и большое количество запросов, она может быть перемещена на второй сервер. После того как нагрузка на нее спадет, она может быть возвращена на исходный сервер.

Каждый тип хранилища определяет свою партицию:

1. **Очередь -> Одна очередь = Одна партиция**

Сервис очереди использует имя очереди в качестве ключа партиции, и все сообщения для данной конкретной очереди будут находиться в одной партиции, что означает, что каждая очередь имеет собственную цель масштабирования. Необходимо, однако, учитывать виртуальное ограничение в 500 сущностей в секунду, поэтому, если очередь испытывает нагрузку более 500 сущностей в секунду, необходимо определить, каким образом можно разделить систему на несколько очередей для реализации высокопроизводительного механизма обмена сообщениями.

2. **Таблица -> Одна партиция таблицы = Одна партиция**

В сервисе таблиц разработчик сам определяет, как будет партиционироваться таблица с помощью определения системного свойства *PartitionKey*. Рекомендуется создавать маленькие многочисленные партиции, так как именно в этом случае будет организована наиболее эффективная масштабируемость на уровне платформы, а вместе с этим и скорость обращения к данным. Каждая сущность будет принадлежать партиции, определенной в ее *PartitionKey*. Если разработчик использует распространенную практику инкрементирования (или декрементирования) значения *PartitionKey*, он можете столкнуться с характерным поведением платформы *Windows Azure* – созданием партиций-диапазонов. Это необходимо для повышения эффективности запросов на диапазоны, которые без партиций-диапазонов должны были бы выходить за пределы партиции или сервера, что понизило бы производительность. При использовании инкрементирования может произойти следующая ситуация: платформа объединит первые три сущности в партицию-диапазон и, если разработчик выполнит запрос на диапазон с использованием *PartitionKey* и запросит сущности с *PartitionKey* между 1 и 3, запрос будет выполнен быстро, так как партиция будет находиться в пределах одного сервера.

3. **Блоб -> Один блоб = Одна партиция**

При записи в партицию операция считается завершенной по записи на все три реплики.

5.5. Избыточность хранилища *Windows Azure Storage*

В *Windows Azure Storage* доступно две опции избыточности: *Locally Redundant Storage* и *Geo Redundant Storage*.

Locally Redundant Storage **(*LRS*)** предоставляет хранилище с высокой степенью долговечности и доступности внутри одной географической локации (региона). Платформа хранит три реплики каждого элемента данных в одной первичной географической локации, что гарантирует, что эти данные можно будет восстановить после сбоя, несущего общий характер (например, выхода из строя диска, узла, корзины и т. д., что является нормальным событием в центре обработки данных), без потери функциональности для аккаунта хранилища и, соответственно, не влияя на доступность и долговечность хранилища. Все операции записи в хранилище выполняются синхронно в три реплики в трех различных доменах ошибок (*fault domain*), код об успешном завершении транзакции возвращается после успешного завершения всех трех операций. В случае использования локального избыточного хранилища, если центр обработки данных, в котором размещены реплики данных, подвергнется сбою, несущему характер катастрофы (стихийной, техногенной или любой другой, которая приведет к потере подключения к центру обработки данных либо потере самого центра), *Microsoft* свяжется с клиентом и сообщит о возможной потере информации и данных, используя контакты, приведенные в подписке клиента.

Вторая опция – ***Geo Redundant Storage*** **(*GRS*)** – предоставляет гораздо более высокую степень долговечности и безопасности. Реплики данных в этом режиме размещаются не только в первичной географической локации, но и во вторичной, находящейся в том же регионе, но за сотни километров. Таким образом, в географически избыточном режиме платформа сохраняет три реплики, но в двух локациях, что гарантирует то, что если центр обработки данных подвергнется сбою, несущему характер катастрофы, то данные будут доступны из вторичной локации. Как и в случае первой опции избыточности, операции записи данных в первичной географической локации должны быть подтверждены перед тем, как система вернет код успешного завершения операции. По подтверждению операции в асинхронном режиме будет инициирован процесс репликации в другую географическую локацию. Рассмотрим подробнее процесс географической репликации.

Когда осуществляются операции создания, удаления, обновления, транзакция полностью реплицируется в три узла хранилища в трех доменах ошибок и обновлений в первичной географической локации, после чего клиенту возвращается код успешного завершения

операции и в асинхронном режиме подтвержденная транзакция реплицируется во вторичную локацию, где полностью реплицируется в три узла хранилища в разных доменах ошибок и обновлений. Общая производительность при этом не падает, так как все совершается асинхронно. Если происходит серьезный сбой в первичной географической локации, применяются правила географической отказоустойчивости, то клиент оповещается о возникшей катастрофе в первичной локации, после чего соответствующие ресурсные *DNS*-записи изменяются и начинают вести не на первичную локацию, но на вторую (*DNS*-запись *account.service.core.windows.net*). В процессе перевода *DNS*-записей могут наблюдаться сбои в работе, но по его завершению существующие блобы и таблицы становятся доступными по их прежним *URL*-ам. После завершения процесса перевода вторичная локация повышается в статусе до первичной; по завершению процесса повышения статуса центра обработки данных инициируется процесс создания новой вторичной географической локации в этом же регионе и дальнейшей репликации данных.

Рассмотрим реальный пример, взятый из блога разработчиков *Windows Azure Storage*. В аккаунте хранилища размещено несколько блобов, *foo* и *bar*. Для блобов полное имя блоба равно значению *PartitionKey*. Разработчик выполняет две транзакции *A* и *B* на блобе *foo*, после чего выполняет две транзакции *X* и *Y* на блобе *bar*. Система гарантирует, что транзакция А будет географически реплицирована перед транзакцией *B* и, соответственно, транзакция *X* будет географически реплицирована перед транзакцией *Y*.

Географически избыточное хранилище включается по умолчанию для всех создающихся аккаунтов хранилища. Можно отключить географическую репликацию на портале управления *Windows Azure* либо указать эту опцию при создании аккаунта хранилища.

Локально избыточное хранилище полезно тогда, когда необходимо сократить затраты на хранение некритичных либо легковоспроизводимых данных, тогда как географически избыточное может быть полезно при наличии важных данных.

5.6. Диагностика хранилища

Сервис *Windows Azure Storage Analytics* предоставляет клиенту возможность отслеживать и анализировать использование данных, хранящихся в аккаунте хранилища (блобы, таблицы и очереди), ис-

пользуя эти данные для адаптации приложения к реальным нагрузкам. Аналитические данные состоят из **логов** (выполненные запросы к аккаунтам хранилища) и **метрик** (статистика по объему и запросам блобов, таблиц и очередей).

Логи хранятся в виде блочных блобов в специальном контейнере блобов ($*logs*). Каждая запись-лог в блобе отображает один сделанный запрос и содержит служебную информацию (*ID*-запроса, *URL* запроса. *HTTP*-статусы, имя аккаунта клиента, имя аккаунта хозяина, *IP*-адрес клиента и др.). Логи позволяют выяснить:

– сколько анонимных запросов было выполнено от определенного диапазона *IP*-адресов;

– к каким контейнерам блобов был наиболее активный доступ;

– сколько раз и каким образом был запрошен конкретный *URL* с *Shared Access Signature*;

– кто запросил удаление контейнера блобов;

– пришел ли запрос на сервер или нет.

Метрики сводят воедино основную статистику для блобов, таблиц и очередей. При этом статистику можно подразделить на два типа – информацию о запросах (количество запросов по часам, средняя задержка на стороне серверов, средняя задержка *E2E*, средняя полоса пропускания, общее количество успешных и неуспешных запросов и т. д., и эта информация предоставляется на уровне сервиса и *API* и доступна для блобов, таблиц и очередей) и информацию об объеме (ежедневная статистика потребленного сервисом пространства, количество контейнеров и количество объектов, хранящихся внутри сервиса, доступна только для блобов). Метрики хранятся внутри специальной служебной таблицы в сервисе таблиц.

Все аналитические логи, статистика и данные метрик хранятся внутри пользовательского аккаунта, и получить к ним доступ можно через стандартное *REST API* сервисов блобов и таблиц и с помощью портала управления *Windows Azure*. Кроме этого логи и метрики могут быть потреблены из любого сервиса, размещенного как в *Windows Azure*, так и в интернете или из локального приложения, которое умеет работать и обрабатывать *HTTP*/*HTTPS*-запросы. Выключить или включить логи и метрики можно как с помощью *REST API*, так и с помощью портала управления *Windows Azure*. Также можно настроить срок, после которого данные будут автоматически удаляться.

5.7. Обеспечение безопасности с использованием *Shared Access Signatures*

Shared Access Signatures доступны для всех сервисов *Windows Azure Storage*. Функциональность *Shared Access Signature* заключается в предоставлении возможности детального контроля доступа к ресурсам; определения, какие операции может совершить пользователь над ресурсом, имея *Shared Access Signature*. К списку доступных для определения *SAS*-операций относятся:

• Чтение и запись контента – в случае блобов дополнительно их свойства и метаданные, а также блок-списки.

• Удаление, лизинг, создание снапшотов блобов.

• Получение списков элементов контента.

• Добавление, удаление, обновление сообщений в очередях.

• Получение метаданных очередей, включая количество сообщений в очереди.

• Чтение, добавление, обновление и вставка сущностей в таблицу.

Параметры *Shared Access Signature* включают в себя всю информацию, необходимую для выдачи доступа к ресурсам хранилища – параметры запроса в *URL* определяют временной промежуток, через который *Shared Access Signature* будет деактивирована, разрешения, предоставляемые данной *Shared Access Signature*, ресурсы, к которым предоставляется доступ, и сигнатуру, с помощью которой происходит аутентификация. Кроме этого в *URL Shared Access Signature* можно включить ссылку на хранимую политику доступа, с помощью которой можно обеспечить более гибкий контроль доступа.

Shared Access Signature должны распространяться с использованием *HTTPS* и разрешать доступ на максимально короткий временной промежуток, необходимый для выполнения операций.

Строение *Shared Access Signature*

В составе *URL Shared Access Signature* находятся компоненты, специфичные для *REST* и для *Shared Access Signature*:

1. **Параметр *signedversion*.** Содержит версию сервиса, обеспечивающего *Shared Access Signature*. При этом, если *Shared Access Signature* указывает на сервис хранилища версии, более ранней, нежели 2012-02-12, этот *Shared Access Signature* может оперировать только блобом или контейнером.

Сервисы *Windows Azure Storage* могут принимать запросы, которые указывают конкретную версию каждой операции (сервиса),

при этом разработчик может указать версию, используя заголовок *x-ms-version*. Функциональность различных версий и механизмы работы (не концептуальные) могут различаться.

Request Headers:

x-ms-version: 2011-08-18

Если *URL Shared Access Signature* использует этот параметр со значением версии, которая является более новой, нежели клиентское приложение, использующее этот *URL*, могут возникнуть функциональные различия. Например, в версиях, более ранних нежели 2011-08-18, операция *Get Blob* не возвратит заголовок *Accept-Ranges*.

2. **Параметр *Signed Resource* (только для блобов).** Значение параметра определяет ресурсы, к которым обеспечивается доступ данной *SAS*. Имеет значение либо ***b*** (блоб), что обеспечивает доступ к контенту и метаданным блоба, либо ***c*** (контейнер), что обеспечивает доступ к контенту и метаданным любого блоба в контейнере и получение списка блобов в этом контейнере.

3. **Параметр *Table Name* (только для таблиц).** Определяет имя таблицы.

4. **Параметр *Access Policy*.** Определяет значения временного промежутка, в течение которого будет работать данная *Shared Access Signature*, и набор разрешений. Состоит из следующих частей:

– *signedstart* – начальный момент времени в *ISO* 8061 (например, так – *YYYY-MM-DDThh*:*mm*:*ssTZD*), когда сигнатура будет активирована. Если опущено, сигнатура будет активирована в момент получения запроса;

– *signedexpiry* – конечный момент времени в *ISO* 8061, когда сигнатура станет неактивной. Опускается, если соответствующее значение указано в хранимой политике доступа;

– *signedpermissions* – набор разрешений, ассоциированных с сигнатурой (*r*, *w*, *d* для блобов *r* и *w* для контейнеров блобов *r a* (*add*) *u p* (*process*, забор и удаление и *r a*(*add*) *u d* (*delete*) для сущностей таблиц) для сообщений в очередях). Опускается, если соответствующее значение указано в хранимой политике доступа. Пример: *sp = rwdl*, *sp = rw*, *sp = r*. Нельзя выдавать разрешения на создание и удаление контейнеров, очередей и таблиц, получения списка сущностей в них, получения и редактирования метаданных и свойств контейнеров блобов и очистки метаданных и очередей от сообщений;

– *Startpk*, *startrk* – только для таблиц. Минимальное значение *partition key* и *row key*, которое доступно пользователю;

– *Endpk*, *endrk* – только для таблиц. Максимальное значение *partition key* и *row key*, которое доступно пользователю.

5. **Параметр *Signed Identifier*.** Определяет соответствующую *Shared Access Signature* хранимую политику доступа. Хранимые политики доступа предоставляют определенную степень контроля над одной или более *Shared Access Signature*, включая отзыв *Shared Access Signature*. Каждый контейнер блобов, очередь или таблица может иметь до пяти ассоциированных с ней хранимых политик доступа.

К сценариям использования *Shared Access Signature* можно отнести два примера:

• Сервис должен давать доступ клиентам к каким-то частям аккаунта хранилища с определенными разрешениями, например, приложение *Windows Phone* для сервиса, который запущен в *Windows Azure*. Токен *Shared Access Signature* должен распространяться клиентам (приложениям *Windows Phone*) для обеспечения доступа к хранилищу *Windows Azure*.

• Сервис должен выдавать доступ к ресурсам по необходимости, также быстро закрывая его после окончания срока действия токенов.

Генерация токенов может проходить согласно следующим моделям:

– токен генерируется специальным сервисом на ограниченный период времени. Успешнее всего эта модель подходит к первому сценарию использования, описанному выше. Непосредственно перед истечением периода действия токена приложение запрашивает новый токен, сервис же решает по каким-либо правилам выдавать этот токен или нет;

– канал коммуникаций между клиентом и сервисом, генерирующим токены, может использовать механизм аутентификации (например, *OAuth*), и клиент должен сначала аутентифицироваться, после чего запросить токен *Shared Access Signature*;

– для второго описанного выше сценария можно использовать сгенерированные токены *Shared Access Signature*, привязанные к хранимой политикой доступа.

Локальный эмулятор *Windows Azure Storage*

Локальный эмулятор хранилища *Windows Azure* эмулирует сервисы *Windows Azure Storage*. Эмулятор поставляется в комплекте *Windows Azure SDK*, и его можно использовать для разработки и тестирования приложений, использующих блобы, очереди и таблицы, в локальной инфраструктуре. Между локальным эмулятором и реальными сервисами хранилища есть некоторые различия.

К общим различиям относятся количество аккаунтов и ключ аутентификации: локальный эмулятор хранилища поддерживает только один фиксированный аккаунт и один ключ аутентификации, при этом они не меняются и равны, например:

Имя аккаунта: *devstoreaccount*1

Ключ аккаунта:

*Eby*8*vdM*02*xNOcqFlqUwJPLlmEtlCDXJ*1*OUzFT*50*uSRZ*6*IFsuFq*2*UVErCz*4*I*6*tq*/*K*1*SZFPTOtr*/*KBHBeksoGMGw*==

Локальный эмулятор хранилища не поддерживает большое количество параллельных подключений и клиентов и не имеет возможности масштабирования, имея отличную схему *URI* от схемы *URI* реальных сервисов хранилища *Windows Azure*. *URI* локального эмулятора определяет имя аккаунта как часть пути в *URI*, но не как часть доменного имени.

Необходимо учитывать различия между локальным эмулятором и облачными сервисами хранилища перед тем, как переводить приложение в *Windows Azure*.

5.8. *Windows Azure SQL Databases*

Windows Azure SQL Databases – это облачный сервис баз данных, основанный на технологиях, использующихся в классическом *SQL Server*. *Windows Azure SQL Databases* предоставляет доступную глобально инфраструктуру и функциональность базы данных как сервис (*Database As A Service*). Основанный на облачных технологиях, сервис *Windows Azure SQL Databases* предлагает большое количество характерных преимуществ, включающих быстрое развертывание, экономичное масштабирование, высокую доступность и сокращение издержек на управление. С точки зрения разработчика, *Windows Azure SQL Databases* предлагает знакомую реляционную модель программирования и методы доступа к данным и простые варианты развертывания, которые уже используются разработчиком. Как и любой облачный сервис, *Windows Azure SQL Databases* устраняет проблемы с инфраструктурой, тем самым предоставляя разработчикам больше свободы. С точки зрения ИТ, *Windows Azure SQL Databases* предлагает надежное и безопасное облачное решение, которое может быть интегрировано локальной инфраструктурой, с гибкими механизмами контроля и управления. Ключевым достоинством сервиса является простота масштабирования решения. При возрастании требований есть возможность горизонтального и вертикального масштабирования.

При разработке локальных приложений с использованием *SQL Server* разработчики используют для взаимодействия с источником данных различные клиентские библиотеки (протокол *TDS*). *Windows Azure SQL Databases* имеет аналогичный интерфейс *TDS*, поэтому разработчики могут использовать те же самые инструменты и библиотеки в процессе создания клиентских приложений с использованием данных в облаке.

Windows Azure SQL Databases оперируют стандартными терминами, за исключением того, что «контейнером» верхнего уровня, который должен быть создан, является сервер *Windows Azure SQL Databases*. Внутри каждого сервера *Windows Azure SQL Databases* могут быть созданы стандартные объекты *SQL Server* – таблицы, представления, хранимые процедуры, индексы и др. Подобная модель позволяет использовать существующую архитектуру реляционной БД и *Transact SQL* и упрощает процесс миграции существующих приложений в *Windows Azure SQL Databases*. Для задач миграции существует набор утилит, упрощающих процесс (например, встроенная функция *SQL Server Management Studio* или *AzureMW*). Серверы и базы данных *Windows Azure SQL Databases* являются виртуальными объектами, не соответствующими физическим сущностям.

Редакции *Windows Azure SQL Databases*

В свободном доступе для разработчика имеется две редакции баз данных *Windows Azure SQL Databases*, которые можно свободно менять на портале управления либо с помощью *SQL*-запроса: *Web* и *Business*. Основная разница между редакциями заключается в максимально доступном размере каждой базы данных в данной редакции (*Web* – до 5 ГБ, *Business* – до 150 ГБ). Каждая из редакций несет собственное назначение – например, базы данных *web*-редакции подходят скорее для проектов, которые оперируют небольшим количеством данных, базы данных *Business*-редакции предназначены для корпоративных баз данных, которым необходимые большие объемы хранилища.

В июле 2013 г. было объявлено о дополнительном предложении *Windows Azure SQL Databases Premium*, в рамках которого предлагается использование более мощных ресурсов для обслуживания серверов *Windows Azure SQL Databases*. Это предложение подразумевает под собой предоставление пользователю эксклюзивно зарезервированных под его нужды ресурсов. Использование базы данных редакции *Premium* позволяет пользователям масштабировать программную инфраструктуру, основываясь на пиковых нагрузках без того, чтобы

учитывать при этом важную особенность облачных ресурсов – разделяемое использование одних и тех же ресурсов несколькими клиентами. Подобное предложение значительно упрощает разработку или миграцию локальных приложений в облако, которые проектируются с учетом пиковых нагрузок, и дает пользователю возможность масштабировать зарезервированные мощности таким образом, чтобы они удовлетворяли потребностям настоящего времени – например, в случае сезонных нагрузок приложению может понадобиться больше ресурсов; приложению необходимо большое количество ресурсов (*CPU*, *RAM*, *IO*) для выполнения операций с эксклюзивным доступом к ресурсам; приложение выполняет большое количество параллельных операций (запросов к базе данных); приложению должна гарантироваться минимальная латентность и др.

Миграция в *Windows Azure SQL Databases*

При миграции или разработке гибридной инфраструктуры с использованием *Windows Azure SQL Databases* разработчик может оставить клиентское приложение в локальной инфраструктуре и перенести только данные, к которым можно получить доступ с помощью библиотек доступа к данным. Однако, если используется подобная архитектура, необходимо учитывать проблемы с задержками, которые происходят в связи с географической удаленностью и коммуникационной инфраструктурой, которые могут привести к более сложному коду в приложении. Рекомендацией в данном случае может служить перемещение логики доступа к данным в *Windows Azure*, таким образом, приложение, использующее данные, находится там же, где и данные, в том же центре обработки данных. Например, в *Windows Azure* может быть расположен *web*-интерфейс приложения.

Синхронизация данных

Сервис *Windows Azure SQL Databases* позволяет решать задачи по синхронизации данных. Например, в центральной базе данных в центре обработки данных на стороне клиента хранятся критичные данные, сотрудники же организации используют приложение, которое работает на мобильных устройствах, и хранилище данных в *SQL Server Express*. В целях безопасности администратор организации не имеет права открывать брандмауэр на локальном центре обработки данных. В этом случае с помощью сервиса *Windows Azure SQL Databases* разработчик может обеспечить безопасное и полностью синхронизированное решение, создав базу данных в *Windows Azure SQL Databases*, провайдера *Sync Framework* для центра обработки

данных, который позволит синхронизировать информацию между центром обработки данных и базой данных в *Windows Azure SQL Databases*, и второго провайдера *Sync Framework* для мобильных устройств сотрудников, который позволит синхронизировать информацию между этими устройствами и данными в *Windows Azure SQL Databases*.

Серверы *SQL Azure*

Каждый аккаунт *Windows Azure* может содержать несколько серверов *SQL Azure*. Эти сервера не являются экземплярами *SQL Server*, но являются абстракцией, использующейся для единой точки администрирования множества серверов. Каждый сервер включает в себя логины (аналогично *SQL Server*) и может быть расположен в регионе, указанном пользователем.

Для создания и управления сервером баз данных используется портал управления *Windows Azure*.

База данных *Windows Azure SQL Databases*

Каждый сервер *Windows Azure SQL Databases* может содержать до 148 баз данных. Каждый сервер содержит служебную базу данных *master*. В базах данных можно создавать стандартные объекты: таблицы, представления, индексы, хранимые процедуры и др. База данных может быть создана на портале управления аналогично созданию сервера. Фактически, пользователю доступна только операция создания – далее присутствует выбор, необходимо ли использовать существующий сервер или создать новый.

Базы данных реплицируются в центре обработки данных, что обеспечивает автоматическую обработку ошибок и балансировку нагрузки. Данные, хранящиеся в базе, распределяются между множеством физических серверов внутри определенной пользователем для сервера географической локации. Таким образом для приложений различного уровня достигается высокая доступность и масштабируемость.

Архитектура доступа к данным

База данных *Windows Azure SQL Databases* работает по стандартному протоколу *TDS*. *TDS* используется также в *SQL Server*, таким образом, приложения, использующие *SQL Server*, могут быть подключены к базе данных *Windows Azure SQL Databases*. Для этого, однако, необходима поддержка *Secure Sockets Layer* (*SSL*).

Подробнее про архитектуру *Windows Azure SQL Databases* можно прочитать в специально посвященном этому вопросу документе по адресу:

http://*rutechnet.blob.core.windows.net*/*files*/*SQL_Azure_Database.pdf*.

Модель безопасности *Windows Azure SQL Databases*

Многие базы данных содержат важные данные, поэтому необходимо правильно осуществлять контроль доступа, который особенно сложен в мультитенантном приложении, использующемся пользователями из различных организаций. Принципы безопасности *Windows Azure SQL Databases* аналогичны *SQL Server* с *SQL ServerAuthentication*:

• Логины *SQL Server*: аутентификация доступа к *SQL Azure* на уровне сервера.

• Пользователи БД: выдача доступа на уровне БД.

• Роли БД: объединение группы пользователей и выдача прав доступа на уровне БД.

SQL Azure Databases использует стандартный протокол *SQL Server Tabular Data Stream* (*TDS*), при этом разрешены исключительно зашифрованные коммуникации. В *SQL Server* 2008 было введено новое средство: прозрачное шифрование данных (*transparent data encryption*, *TDE*), позволяющее полностью шифровать данные с минимальными усилиями. На данный момент *SQL Azure Databases* не поддерживает шифрование на уровне базы данных. Следует заметить, что в настоящее время *SQL Azure Databases* доступен только через *TCP*-соединения и только через порт 1433, поэтому необходимо учитывать возможности шифрования *ADO.NET* и сертификаты. Второе средство защиты *SQL Database Azure* – брандмауэр *SQL Azure Databases*, которым изначально блокируется весь доступ к серверу *SQL Azure Databases*. Попытки подключения до соответствующей настройки будут безуспешны. Брандмауэром *Windows Azure SQL Databases* можно управлять через портал *Windows Azure SQL Databases* или напрямую в главной базе данных с помощью хранимых процедур, таких как *sp_set_firewall_rule* и *sp_delete_firewall_rule*.

При первой попытке управления созданным сервером портал *Windows Azure* предложит автоматически добавить *IP*-адрес пользователя в список разрешенных.

Как и в любой реализации *SQL Server*, управление пользовательскими учетными записями – еще один аспект, который нужно контролировать. Благодаря этим средствам *Windows Azure SQL Azure* является высокозащищенной управляемой платформой для приложений в облаке.

Между *SQL Server* и *SQL Azure Databases* в контексте безопасности существует список различий:

– *Microsoft SQL Server* поддерживает аутентификацию *Windows Integrated* с использованием параметров доступа из *Active Directory*; *SQL Azure Databases* поддерживает только *SQL Server Authentication*;

– *Microsoft SQL Server* и *SQL Azure Databases* используют одинаковую модель авторизации на основе пользователей и ролей, создаваемых в каждой базе данных и связываемых с логинами пользователей;

– *Microsoft SQL Server* имеет стандартные роли уровня сервера, такие как *serveradmin*, *securityadmin* и *dbcreator*, однако в *SQL Azure Databases* этих ролей нет. Вместо этого в *SQL Azure Databases* доступна роль *loginmanager* (создание логинов) и *dbmanager* (создание и управление базами данных). Эти роли могут быть привязаны к пользователям в базе данных *master*;

– доступ к *SQL Server* и *Windows Azure SQL Databases* происходит по протоколу уровня приложения *Tabular Data Stream* (*TDS*), защищенному протоколом *SSL*, через порт *TCP*/1433. Использование *SSL* опционально для *Microsoft SQL Server* и обязательно для *Windows Azure SQL Databases*;

– в *SQL Server* контроль доступа на основе *IP* должен быть осуществлен на уровне хоста или сети, с использованием брандмауэров. В *Windows Azure SQL Databases* есть встроенный брандмауэр, ограничивающий весь доступ к серверу *Windows Azure SQL Databases* до определения клиентов компьютеров, имеющих право доступа. Брандмауэр выдает доступ, основываясь на *IP*-адресе каждого запроса;

– *SQL Server* предоставляет шифрование в режиме реального времени всех хранящихся данных на страничном уровне с использованием функциональности *Transparent Data Encryption* (*TDE*). Подобное шифрование не поддерживается в *Windows Azure SQL Databases*.

Масштабирование баз данных

Каждая база данных ограничена размером в 150 ГБ (в том случае, если используется режим *Business*). Для хранения более чем 150 Гб данных разработчик должен реализовывать механизм партиционирования или шардинга данных. Шардингом называется методика, используемая разработчиками для увеличения производительности, масштабирования и снижения стоимости в тех случаях, когда, например, приложение использует модель данных с четкими критериями индексирования, например, хранящие и обрабатывающие данные о продажах (дата и время). В этом случае методика шардинга может помочь оптимизировать и масштабировать решение, позволяя также параллельно обрабатывать данные – приложения могут располагать

несколько партиций данных на несколько наборов вычислительных ресурсов и обрабатывать данные одновременно.

Windows Azure SQL Databases предоставляет встроенный механизм для шардинга данных – *Windows Azure SQL Databases Federations*. Подробнее про *SQL Azure Federations*: *http*://*msdn.microsoft.com*/*en-us*/*library*/*windowsazure*/*hh*597452.*aspx*.

Возрастание количества и сложности хранимых данных приводит к неизбежной проблеме нехватки ресурсов для осуществления их хранения. В сценарии, в котором важна производительность при доступе к таким данным, необходимо реализовывать сложные механизмы масштабирования, производить настройку, поддерживать работоспособность построенной инфраструктуры. *Windows Azure* предоставляет два типа хранилища, которые можно использовать в различных ситуациях и инфраструктурах (включая гибридные): *Windows Azure Storage* и *Windows Azure SQL Databases*. *Windows Azure Storage* предлагает три сервиса – таблицы, блобы и очереди, – которые являются масштабируемыми, долговечными механизмами, реализующими глобально-доступное хранилище данных. *Windows Azure SQL Databases* является решением, позволяющим реализовывать более сложные сценарии хранения данных, в том числе реляционных.

Глава 6. Бизнес-аналитика и анализ данных с *SQL Reporting* и *Hadoop*

6.1. Обзор *Business Intelligence*

Бизнес-аналитика, или *Business Intelligence* (*BI*), является важным процессом, во время которого происходит интеграция и консолидация данных в масштабе организации в едином хранилище, в котором пользователи могут выполнять специальные запросы и формировать отчеты для анализа существующих данных. Фактически, целью бизнес-аналитики является именно хранение данных, в котором могут обращаться пользователи, ответственные за принятие бизнес-решения на основе анализа полученных данных. Например, типичным вопросом к системе бизнес-аналитики может быть: «Какая категория продуктов является наихудшей в плане реализации в первом квартале 2013 года?». Наиболее важными свойствами системы бизнес-аналитики являются:

– периодические операции записи в результате запросов;

– небольшое количество пользователей;

– большой размер данных, хранимых в базе данных.

Пользователи систем бизнес-аналитики занимаются генерацией отчетов, отражающих различные факторы, связанные с важными процессами внутри организации (например, финансовыми), или запросов сравнения данных. Системы бизнес-аналитики должны отслеживать историю данных, так как часто пользователи производят сопоставление данных, собранных в различные периоды времени. По этой причине объем данных в хранилище обычно очень велик, и механизм хранения данных должен обеспечивать достаточное для задач пользователя и оптимальное с позиции выполнения запросов время. На данный момент существует большое количество решений для бизнес-аналитики, лидерами являются корпорации *Microsoft* и *Oracle*, предоставляющие мощные движки и механизмы – *SQL Server*, *SQL Reporting* и др. Локальная версия системы бизнес-аналитики от *Microsoft* называется *SQL Server Reporting Services* (*SSRS*), которая предлагает программную серверную систему создания отчетов. В *SQL Server Reporting Services* есть возможность генерировать как печатные, так и динамические отчеты, в том числе с использованием *web*-

служб. *SQL Server Reporting Services* входит в состав *Express*, *Workgroup*, *Standard* и *Enterprise* версий *Microsoft SQL Server* в качестве устанавливаемого дополнения. Отчеты, создаваемые в *SQL Server Reporting Services*, должны быть описаны с помощью специального подъязыка *Report Definition Language* (*RDL*) и могут проектироваться при помощи *Visual Studio* с установленным дополнением *Business Intelligence Projects* либо при помощи входящего в комплект *Report Builder*. Отчеты, определенные при помощи *RDL*, могут быть созданы и сохранены в различных файловых форматах, например, *XLS*, *PDF*. Начиная с версии *SQL Server* 2008, *SQL Server Reporting Services* предлагает возможность генерации отчетов в формате *DOC*. *RDL*-отчеты можно просматривать программным способом с помощью элемента управления *ASP.NET ReportViewer*, что позволяет встраивать отчеты прямо в приложение или сайт.

С развитием облачных вычислений стало возможным переносить как хранилища данных, так и системы бизнес-аналитики на мощности вендора, таким образом оплачивая только те ресурсы, которые используются. Облачные особенности предоставляют важное преимущество эластичности и возможности обрабатывать большие массивы данных. Системы бизнес-аналитики в облаках можно условно разделить на два типа:

– данные в облаке, бизнес-аналитика локально;

– движок и приложение в облаке.

Размещение движка и отчетов в облаке позволяют размещать отчеты таким образом, что их можно встраивать как в приложение, так и иметь глобальный доступ с помощью *URL*.

К основным сценариям облачной системы бизнес-аналитики, которая называется *Windows Azure SQL Databases Reporting*, относятся следующие:

– сезонные нагрузки. Например, организация, реализующая некоторые продукты, имеет периодическую задачу генерации отчетов из хранилища данных. Размещение ресурсов, на которых выполняется решение этой задачи, в локальном центре обработки данных влечет за собой необходимость в расчете максимальных нагрузок, эффективного использования и др. После этого в какие-то моменты осуществляется нагрузка, но большую часть времени ресурсы могут простаивать. В этом случае, когда есть периодическая необходимость генерации отчетов, может быть использован облачный сервис *Windows Azure SQL Databases Reporting*. После спада пика нагрузки,

как и в случае с любыми другими сервисами, можно отказаться от оплачиваемых ресурсов;

– сценарий, в котором происходит разделение отчетности между несколькими партнерами, возникает, когда сгенерированную отчетность необходимо предоставить партнерской организации. В случае размещения систем аналитики в локальном центре обработки данных партнеру, который должен обеспечить доступ к отчетам, необходимо проводить конфигурацию, настройку безопасности, портов, предоставлять аккаунты и т. д. Если поместить отчеты в облако, процесс значительно упрощается – клиенты и поставщики могут получить доступ к отчетам тогда, когда им это необходимо;

– гибридная инфраструктура. Часть данных в локальном центре обработки данных, часть, которую можно публично выставлять, расположена в облаке.

6.2. Создание отчетов в *Report Builder*

Рассмотрим последовательность действий по созданию отчетов с помощью *Report Builder* и базы данных *AdventureWorks*, предоставляемой корпорацией *Microsoft* в тестовых целях.

Создадим сервер баз данных *Windows Azure SQL Databases* с помощью портала управления *Windows Azure*.

База данных *AdventureWorks* расположена на *CodePlex* по адресу: *http*://*msftdbprodsamples*.*codeplex*.*com*/*releases*/*view*/37304.

В данном примере используется версия *AdventureWorks*2012*ForWindowsAzureSQLDatabase*.

После распаковки архива, содержащего базу данных, необходимо выполнить следующую команду для развертывания базы данных в *Windows Azure*:

CreateAdventureWorksForSQLAzure.cmd <servername>.database.windows.net <username> <password>

Развертывание базы данных может занять некоторое время.

Создадим сервис *Windows Azure SQL Reporting* с помощью портала управления *Windows Azure*. Расположение сервиса в одном регионе с созданным сервером *Windows Azure SQL Databases* значительно уменьшит задержки при их взаимодействии.

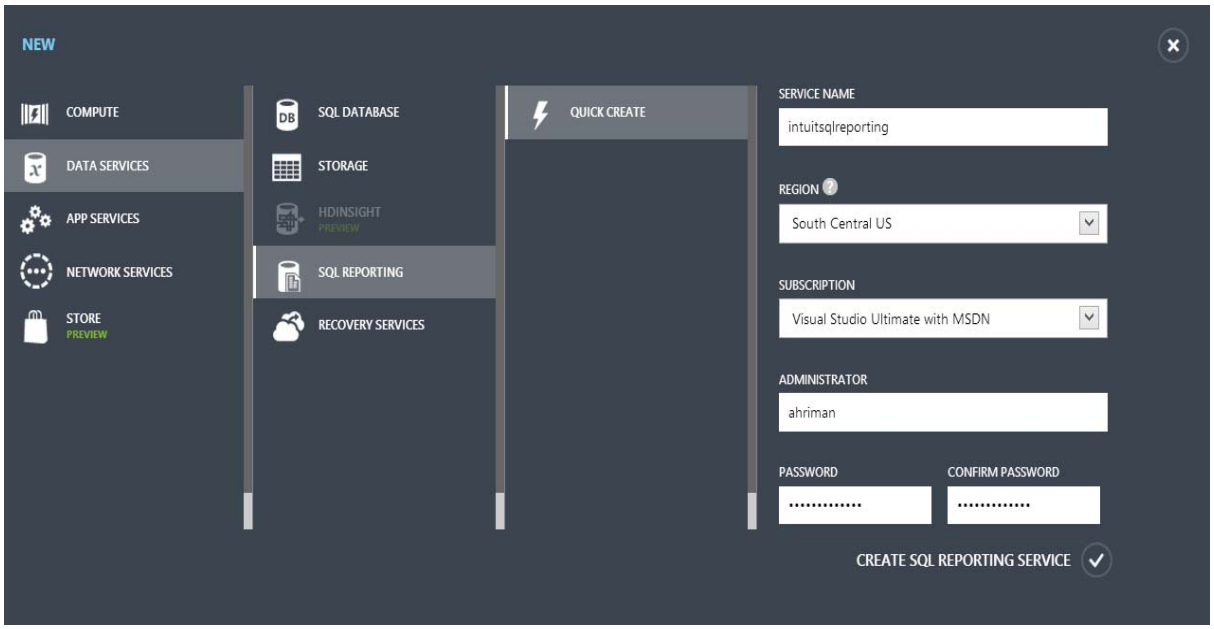

Рис. 6.1. Создание сервиса *Windows Azure SQL Reporting* с помощью портала управления *Windows Azure*

Для генерации отчетов необходимо связать сервис *SQL Reporting* с источником данных. Добавим источник данных в сервис *SQL Reporting*.

В открывшемся диалоговом окне необходимо задать логическое название источника данных и его описание. Базу данных для источника данных можно выбрать в выпадающем меню ***DATABASE***.

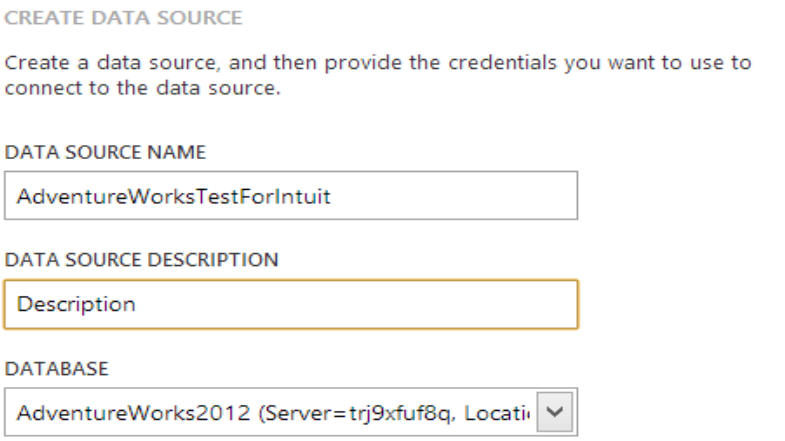

Рис. 6.2. Создание базы данных

На следующей странице выберем опцию *PROVIDE CREDENTIALS TO BE STORED SECURELY IN THE REPORT SERVER*. Введем логин и пароль администратора сервера баз данных.

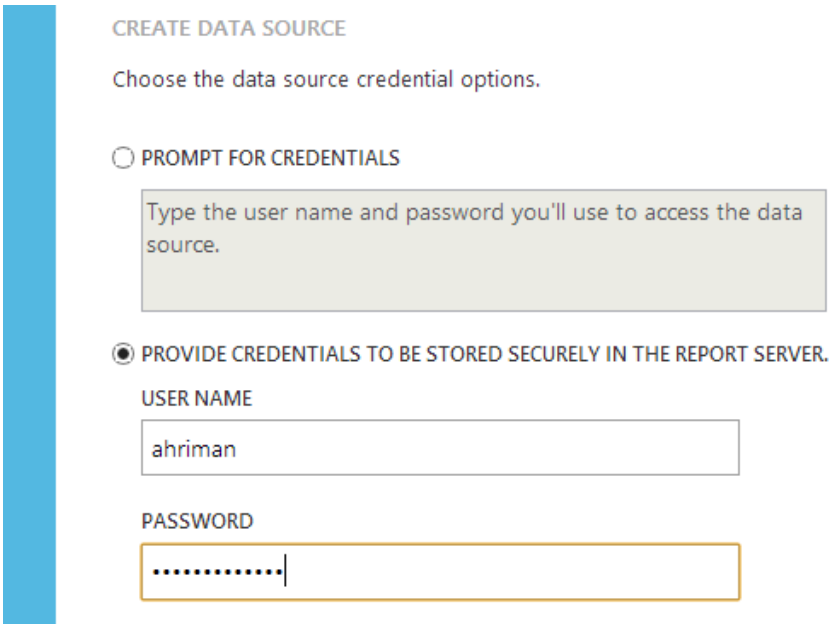

Рис. 6.3. Добавление администратора базы данных

Для взаимодействия с сервисом *SQL Reporting* установим *Report Builder*, дистрибутив которого можно загрузить с сайта *Microsoft*: *http://www.microsoft.com/ru-ru/download/details.aspx?id=*29072.

После запуска *Report Builder* (в зависимости от языка установки он может называться «Построитель Отчетов 3.0») подключимся к серверу, нажав «Соединение» в нижней части интерфейса.

Ссылку, которую необходимо использовать для подключения, можно скопировать на панели управления сервисом.

После ввода логина и пароля и подключения к серверу изменится статус подключения в нижней части интерфейса.

Добавим источник данных в *Report Builder*.

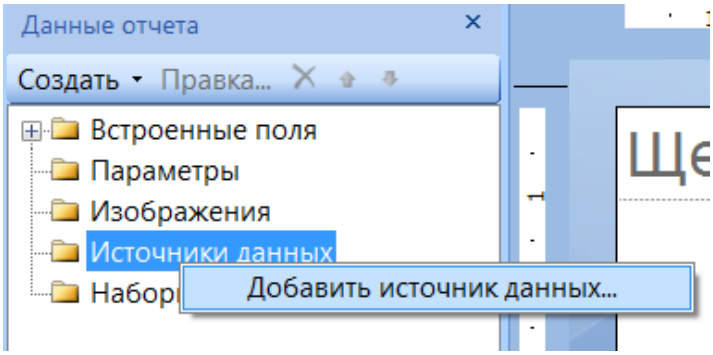

Рис. 6.4. Добавление данных в *Report Builder*

В открывшемся окне необходимо выбрать подключенный сервер.

Введем заголовок отчета, например, «Отчетность по продажам».

Добавим еще один источник данных, нажав правой кнопкой мыши на уже созданном источнике данных *DataSource*.

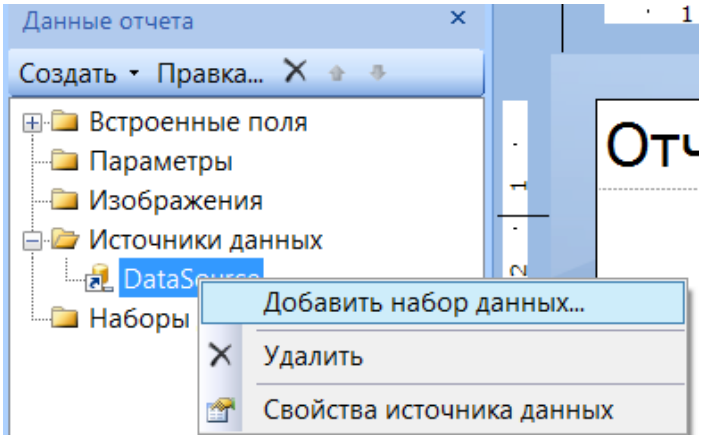

Рис. 6.5. Добавление источника данных

В открывшемся окне создания набора данных вставим в текстовое поле «Запрос» *SQL*-запрос для получения данных по продажам:

SELECT soh.[SalesOrderID],DATEPART(year, soh.[OrderDate]) AS 'Year'

,soh.[CustomerID],soh.[TerritoryID],terr.[Name] as 'TerritoryName'

,terr.[CountryRegionCode] as 'Country',soh.[TotalDue] as 'TotalSales'

FROM [Sales].[SalesOrderHeader] AS soh

JOIN [Sales].[SalesTerritory] AS terr

ON terr.[TerritoryID] = soh.[TerritoryID]

ORDER BY 'Year'

Вставим матрицу для отображения отчета.

Выберем созданный набор данных.

Сформируем форму отчета: перенесем *Year* в «Группы столбцов», *Country* и *TerritoryName* в «Группы строк» и *TotalSales* – в «Значения».

Нажмем «Выполнить» для запуска формирования отчета.

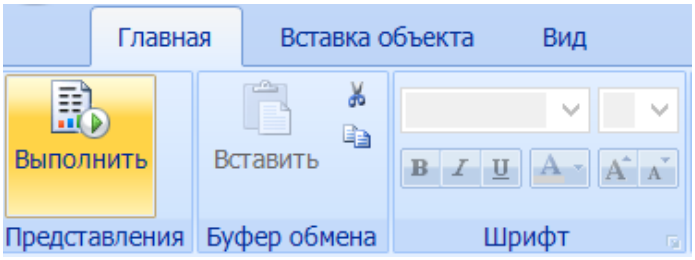

Рис. 6.6. Запуск формирования отчета

После выполнения будет выведен отчет по продажам за несколько лет.

Отчётность по продажам

Country	Territory Name	2005	2006	2007	2008	Итог
⊞ AU	Итог	1446497.1744	2380484.8387	4313294.8365	3674099.2456	11814376.0952
⊞ CA	Итог	1866734.9221	6130230.7354	6964947.2034	3437016.3271	18398929.1880
⊞ DE	Итог	262752.4184	575960.0974	2432549.8252	2208557.2345	5479819.5755
⊞ FR	Итог	199531.7230	1535232.8960	3815005.2509	2569979.4761	8119749.3460
⊞ GB	Итог	322207.5294	1602371.3205	3873251.7497	2776218.1086	8574048.7082
⊞ US	Итог	8595526.8591	22239568.5473	25772440.6803	14222327.1163	70829863.2030
Итог		12693250.6264	34463848.4353	47171489.5460	28888197.5082	123216786.1159

Рис. 6.7. Пример отчета по продажам

Сохраните отчет, перейдя обратно в **Конструктор**. Так как активное подключение принадлежит серверу в *Windows Azure*, сохранение можно произвести на него.

После сохранения любой пользователь с необходимыми правами может получить доступ к отчету, перейдя по ссылке *https://[servername].reporting.windows.net/reportserver* и введя логин и пароль.

В сервисе *SQL Reporting* может быть несколько пользователей, имеющих доступ к сервису, которые должны быть добавлены на вкладке *Users* на панели управления сервисом.

Таким образом, облачный сервис *SQL Reporting* можно использовать для различных сценариев, подразумевающих глобальный удобный доступ к отчетам, виртуально-неограниченные ресурсы и эластичность.

6.3. Анализ данных с *Hadoop*

Сервис *Windows Azure HDInsight* (*Hadoop*) – это облачный сервис, предлагающий экосистему и создание кластеров *Hadoop* по запросу пользователя. С помощью портала *Windows Azure* могут быть созданы и развернуты кластеры *Hadoop* размером до 32 узлов (необходимо уточнить, что на момент написания сервис *Windows Azure HDInsight* находится в закрытой *Preview*-версии). Кроме создания задач *MapReduce*, разработчик имеет доступ к интерактивной консоли, которая позволяет писать запросы к данным на *JavaScript* и *Hive*.

MapReduce – это параллельное и распределенное решение, разработанное корпорацией *Google* для обработки больших массивов

данных и активно использующееся в таких сферах, как, например, поисковые движки. *MapReduce*, или *M*/*R*, состоит из двух функций-компонентов – *Map* и *Reduce*. Первая функция – *Map* – используется для вычисления наборов ключ-значение. *Reduce* – функция, получающая результаты вычисления функции *Map* и применяющая к ним другую функцию. В подходе *M*/*R* подразумевается, что между данными нет прямых зависимостей, что упрощает процесс параллелизации. Узел-мастер распределяет задачи *M*/*R* по обработчикам *Map* и *Reduce*, собирая информацию. Все пары ключ–значение с идентичным значением ключа отправляются на обработку на один обработчик.

Типичным примером использования *MapReduce*, а значит, и *Hadoop*, является анализ файлов логов. Для файлов логов обычна ситуация разрастания до очень больших размеров, при этом все файлы соблюдают строгий синтаксис данных, что позволяет применять логику обработки к этим файлам. Несмотря на простоту обработки, если файлы логов содержат очень большое количество записей, их обработка на одном компьютере может занять долгое время.

С помощью *MapReduce* можно разделять большие файлы логов на части. Функция *Map* ищет уникальные вхождения записей (например, *web*-страниц). Каждый раз, когда *web*-страница находится в файле, в функцию *Reduce* поступает ключ-значение, где ключ – *web*-страница, а значение – 1. Обработчики функции *Reduce* агрегируют количество для каждой из *web*-страниц, и в результате пользователь получает общее количество входов для каждой из *web*-страниц.

Сервис основан на распределенной файловой системе *HDFS* (*Hadoop Distributed File System*), реализованной на кластере узлов *Hadoop* двух видов – *Data Node* и *Name Node*. Подобный кластер может быть как гомогенной структуры (с унифицированными характеристиками узлов), так и гетерогенной (большим количеством узлов разных характеристик). *Name Nodes* содержат информацию о том, на каком из узлов данных (*Data Node*) содержится конкретная реплика (реплики используются для обеспечения высокой надежности и избыточности), и представляют клиенту эту информацию. При выходе из строя реплики фрагмента одна из его вторичных реплик назначается основной. Масштабируемость достигается за счет параллельной обработки задач. *Task Tracker* называются узлы, хранящие входные фрагменты задач (файлов) и запускающие экземпляры выполнения *MapRedurce*, координирует же эти экземпляры выполнения узел, носящий название *Job Tracker*. В различных реальных задачах экосистема *Hadoop*/*HDInsight* состоит из гораздо большего количества компонентов и модулей интеграции с другим программным обеспечением и форматами данных.

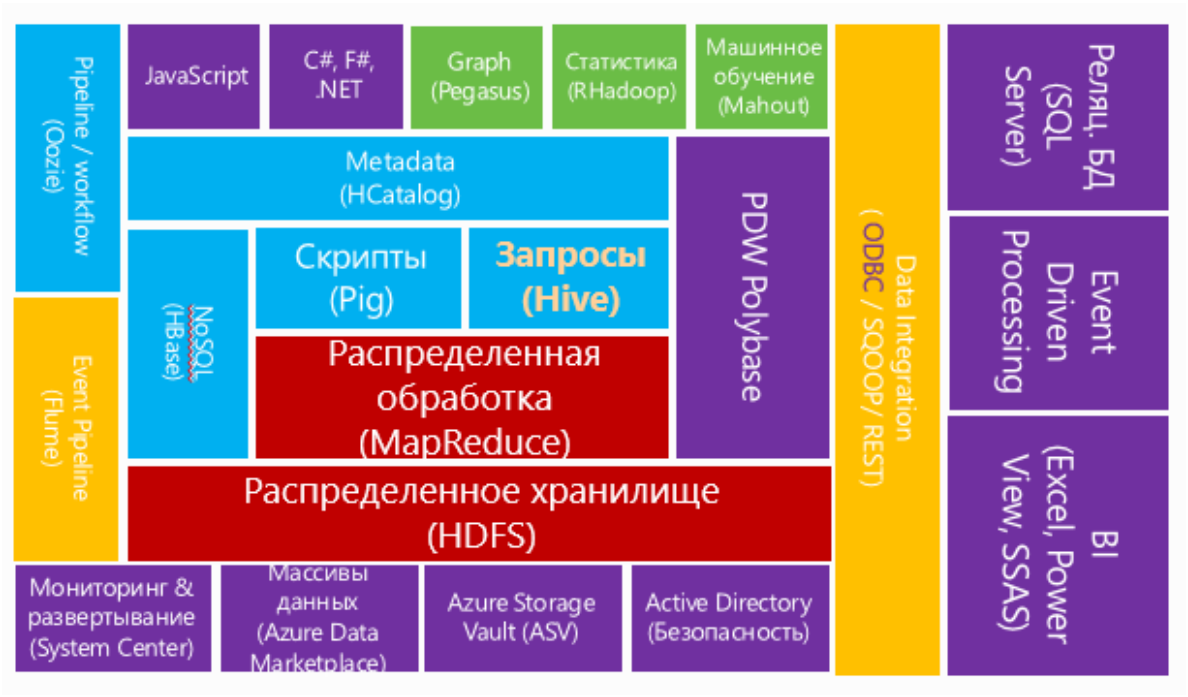

Рис. 6.8. Экосистема *Hadoop*/*HDInsight*

Данные, используемые *HDInsight* в задачах, хранятся в *Windows Azure Storage* в блобах и получаются для обработки по парадигме *MapReduce* узлом-мастером, который затем распределяет задачи согласно заданной логике по обработчикам. По этой причине для работоспособности кластера необходимо создать и настроить аккаунт хранилища *Windows Azure* и создать кластер максимально близко к географическому расположению хранилища (для избежания увеличения латентности).

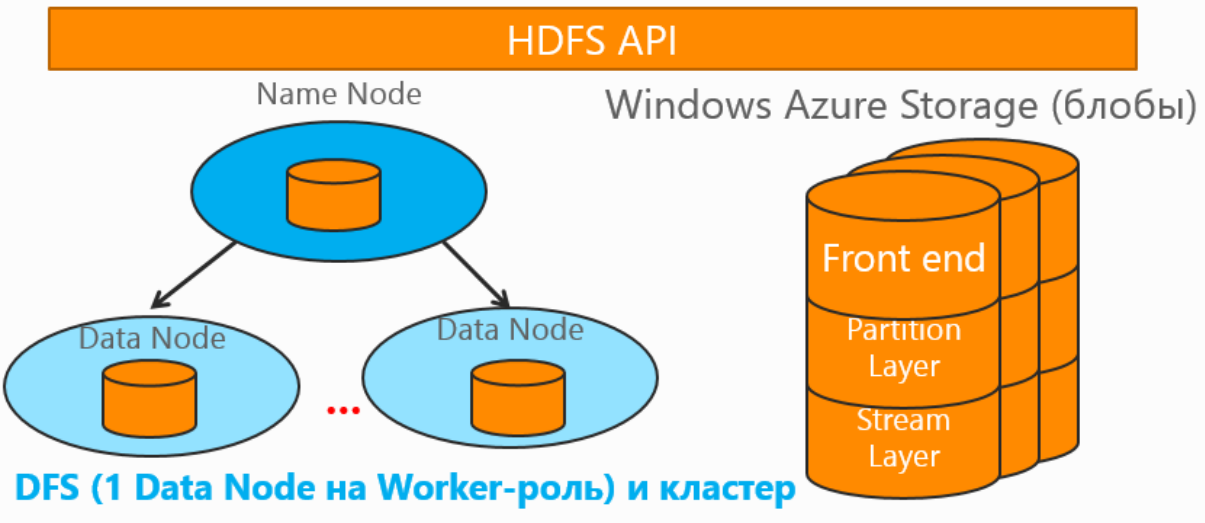

Рис. 6.9. Доступ к хранилищу *HDInsight*

Экосистема *HDInsight*

Как было изложено выше, сервис *HDInsight* предоставляет облачный сервис по управлению большими данными. *HDInsight* имеет собственные реализации *Hive* и *Pig*, которые используются для обработки данных и хранения, соответственно, а также способен интегрироваться с существующими мощными средствами *BI*, разработанными *Microsoft*: *SQL Server Analysis Services*, *Reporting Services*, *PowerPivot*, *Excel*.

Говоря о том, что представляет собой *Pig*, необходимо уточнить, что *Pig* – это высокоуровневая платформа, которая обеспечивает возможность обработки больших данных в кластерах *Hadoop*. *Pig* состоит из специального языка запросов *Pig Latin*, которые выполняются к массивам данных в консоли, и способен интегрироваться *User Defined Functions* (*UDF*) на *Java*, *Python*, *C#* и *JavaScript*.

Hive является распределенным сервисом хранения данных «над» *HDFS* и предоставляет интерфейс для взаимодействия с данными с помощью *SQL* запросов на языке *HiveQL* (поддиалекте *SQL*) и реляционной модели. Аналогично *Pig*, *Hive* трансформирует запрос на собственном языке в задачи *MapReduce*.

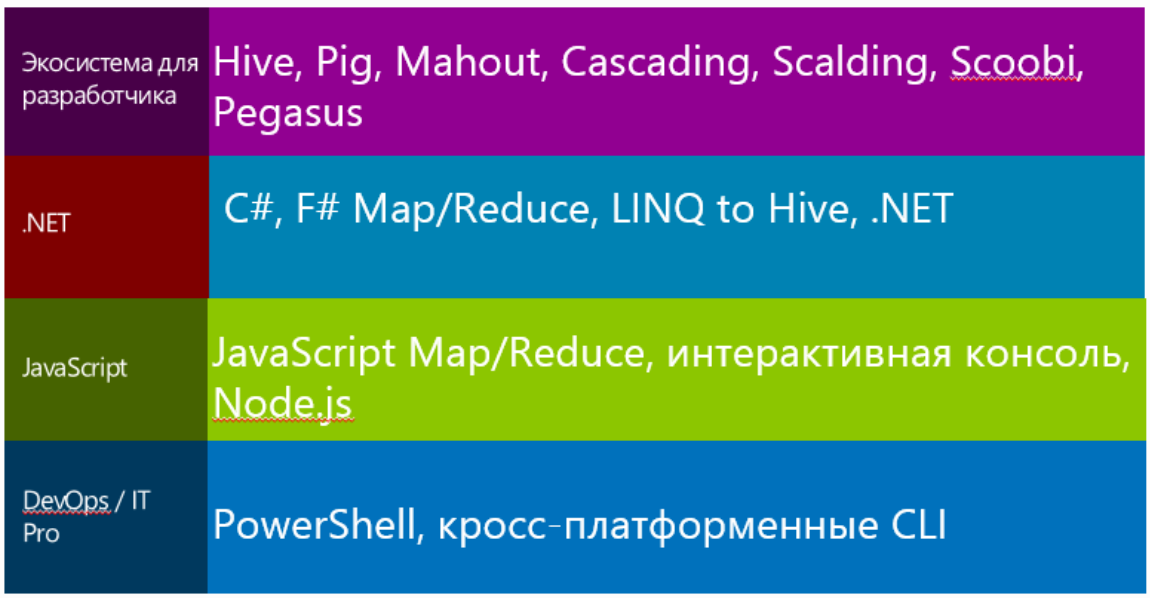

Рис. 6.10. Экосистема *Hadoop*/*HDInsight*

Функциональность *HDInsight* гарантирует также то, что разработчик может использовать уже привычные ему средства, такие как, например, *Powershell*, *.NET*, *Java*, *F#*, *Javascript* и *Node.js*, для разработки, управления и мониторинга происходящего на кластере. Также

HDInsight может быть установлен в его локальной версии на серверный продукт *Microsoft Windows Server*.

Использование *HDInsight*

Для того чтобы начать использование *HDInsight*, необходимо оставить заявку, на портале управления выбрать *HDInsight*, нажать на ссылку *Preview Program* и нажать на кнопку *Try it now*.

После активации сервиса можно начать создание кластера *Hadoop*. Для этого нажмем *DATA SERVICES* | *HDINSIGHT* | *CUSTOM CREATE*.

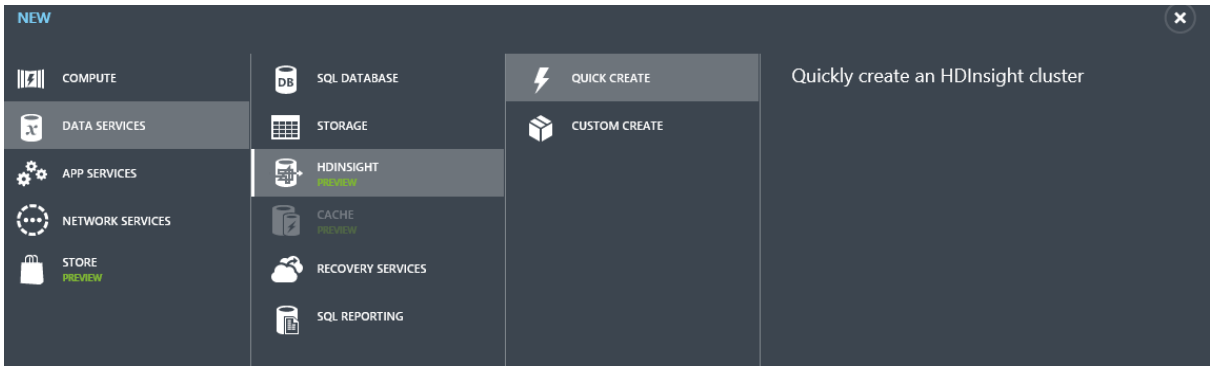

Рис. 6.11. Создание кластера *HDInsight*

Введем название нового кластера, все остальные настройки оставим по умолчанию.

На следующей странице введем учетные данные администратора.

На следующей странице выберем опцию создания нового аккаунта хранилища и введем соответствующие данные.

После создания кластера нам станет доступна панель управления кластером.

Из панели управления можно инициировать сессию удаленного доступа (*CONNECT*), внутри виртуальной машины создать задачу, посмотреть историю выполнения задач, изучить информацию по *Hadoop*. Кнопка *Manage Cluster* позволяет контролировать размер использованного дискового пространства, а также задать папки в *Windows Azure* блобах, которые можно рассматривать как хранилище (*Azure Storage Vault*).

Войдем на панель управления непосредственно кластером, нажав *MANAGE CLUSTER*. Введем учетные данные, настроенные при создании кластера.

Создадим задачу *MapReduce* из тестового набора. Для этого нажмем кнопку *Samples*.

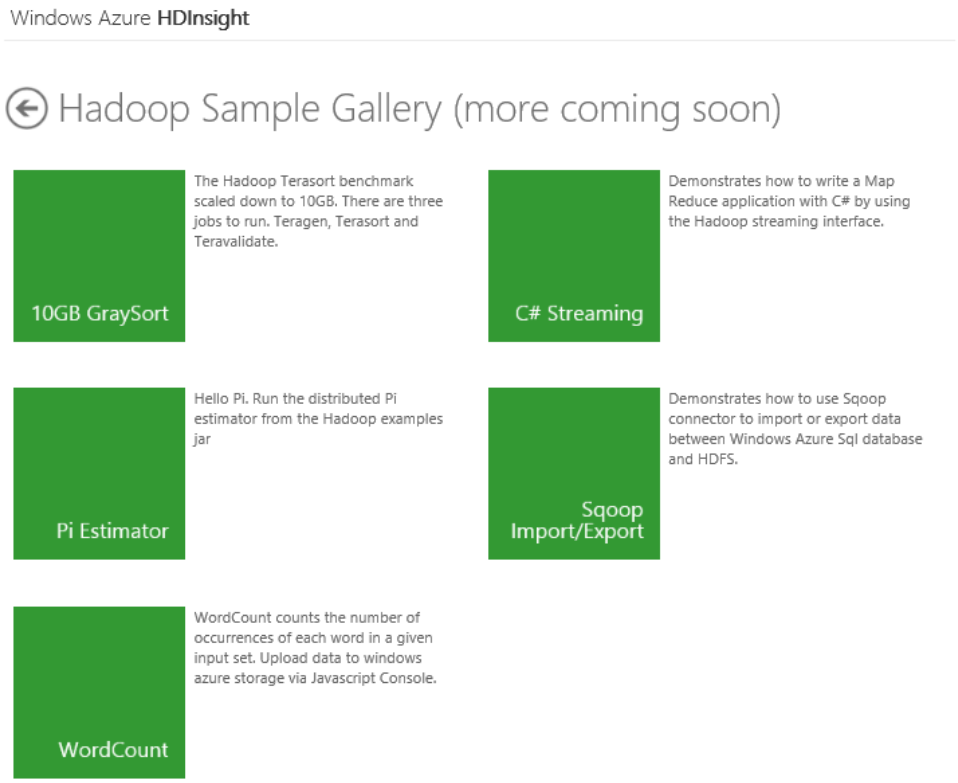

Рис. 6.12. Набор тестовых задач *HDInsight*

Нажмем кнопку *Pi Estimator*. На странице тестовой задачи нажмем *Deploy to your cluster*. Это приведет к странице, которую можно также вызвать, нажав кнопку *Create Job*. Будет открыта страница добавления новой задачи, на которой необходимо ввести название задачи, указать *JAR*-файл задачи и настроить параметры (если таковые есть). В случае тестовой задачи эти данные заполняются автоматически.

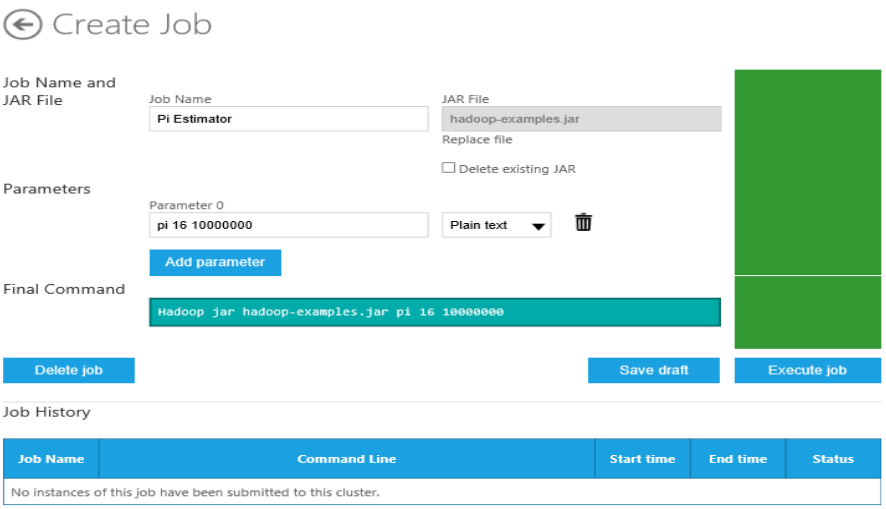

Рис. 6.13. Добавление задачи на расчет

Задача, которая будет запущена, инициирует расчет числа Пи, исходя из 16 *Maps* (первый параметр в команде) и 10 млн сэмплов на каждый *Map* (второй параметр).

Нажмем *Execute Job* и дождемся выполнения задачи. После окончания выполнения будут выведены результаты и служебная информация о задаче.

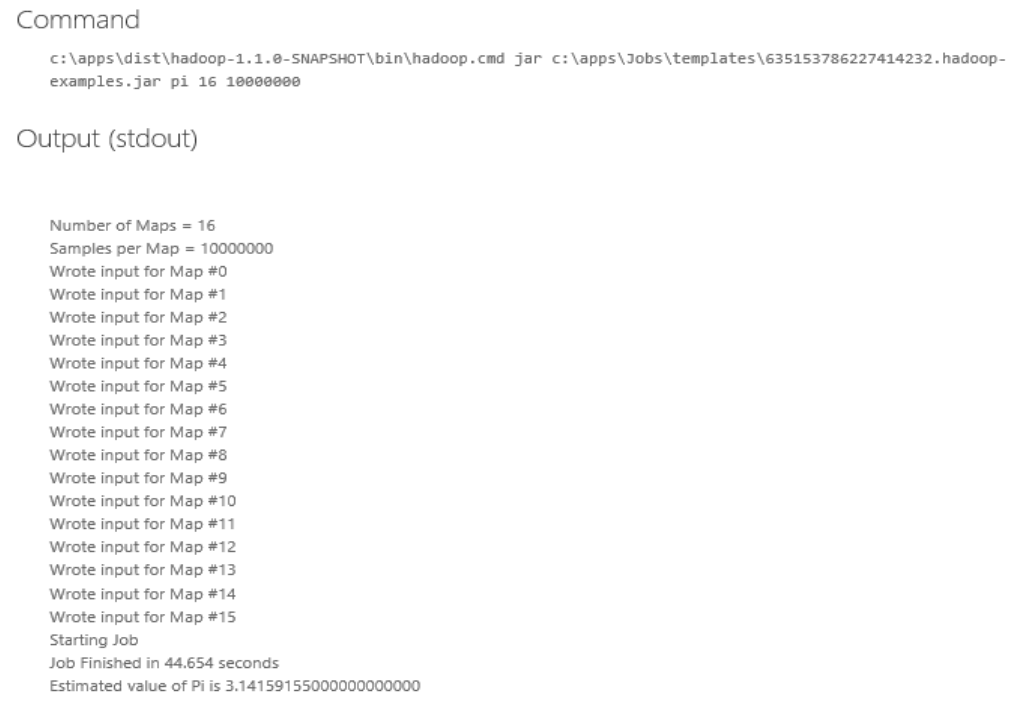

Рис. 6.14. Результат выполнения задачи по вычислению числа Пи

Воспользуемся интерактивной консолью для расчета скрипта на *Javascript* (*MapReduce*). В качестве примера возьмем тестовую задачу по подсчету слов. Для этого загрузим скрипт *WordCount.js* и текстовый файл *davinci.txt*, зайдя на страницу *Samples* и нажав на кнопку *WordCount*.

Вернемся на главную страницу панели управления кластером и нажмем на кнопку *Interactive Console*. Обратите внимание, что ту же задачу можно выполнить, нажав *Deploy to your cluster*.

Введем в интерактивную консоль команду *fs.put*() для загрузки файла *WordCount.js*. Выберем загруженный локально файл. Значение *Destination* укажем равным *./WordCount.js*/. Повторим процедуру для загрузки файла *davinci.txt*. Значение *Destination* для *davinci.txt* укажем равным *./example/data*/.

Выполним команду, указанную ниже, и после выполнения задачи нажмем *View Log* для просмотра информации о задаче.

pig.from("/example/data/davinci.txt").mapReduce("WordCount.js", "word, count:long").orderBy("countDESC").take(10). to("DaVinciTop10Words")

Увидеть результаты можно, введя команду *fs.read("DaVinciTop10Words").*

С появлением тенденции быстрого увеличения количества данных, существующей в сегодняшнем мире, и распространением термина *Big Data* (Большие данные) локальные центры, которые часто не могут покрыть потребности в обработке все возрастающих массивов данных, могут быть как заменены, так и дополнены (в зависимости от сценариев) ресурсами, хранящимися в облаке, для того, чтобы оптимизировать затраты и увеличить эффективность производства.

Глава 7. Доступ к сервисам предприятия с *Windows Azure Service Bus*

Сервис *Windows Azure Service Bus* представляет собой интеграционную шину (*Middleware*) с функциональностью переноса, безопасного обмена сообщениями и создания распределенных и слабосвязанных приложений в облаке, а также гибридных приложений, размещенных одновременно в частных и общедоступных облачных службах. *Service Bus* – это программная прослойка, которая позволяет интегрировать некоторые сущности, находящиеся внутри предприятия, и внешних клиентов. *Windows Azure Service Bus* является одним из вспомогательных компонентов облачной платформы *Windows Azure*, и основная его роль как вспомогательного компонента – помощь в реализации сложных сценариев взаимодействия, например, таких, как создание безопасного канала коммуникаций между *web*-сервисом внутри *NAT* или за брандмауэром и глобально-доступного сервиса.

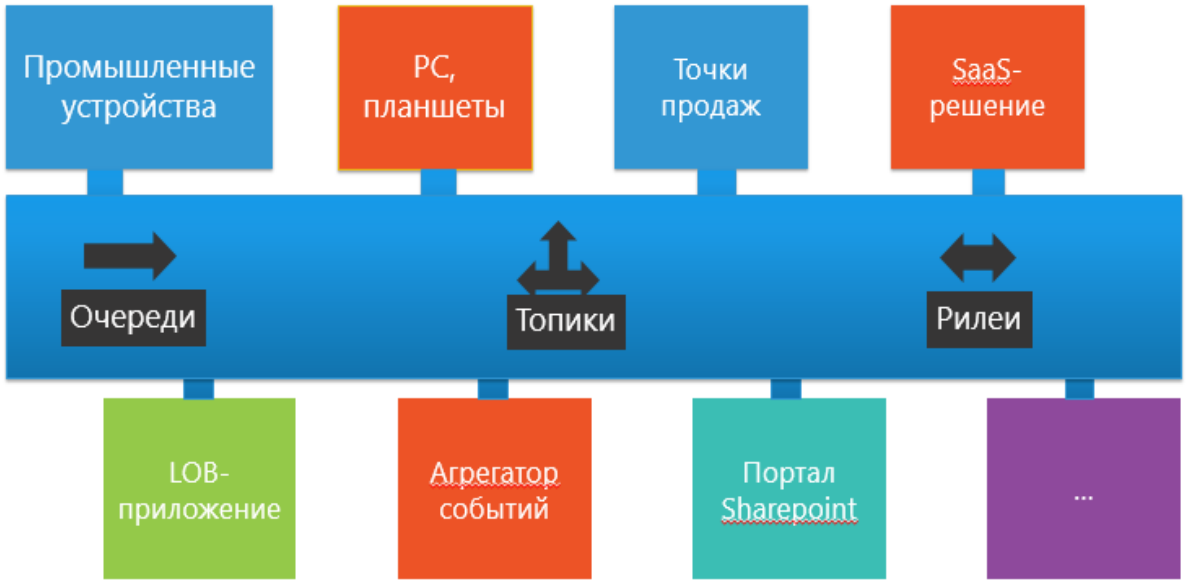

Рис. 7.1. Сервисы *Windows Azure Service Bus*

Когда рассматриваются интеграционная шина и обмен сообщениями, нельзя обойти вниманием один из основополагающих принципов построения распределенных систем, который называется связанностью. Суть этого принципа заключается в том, что выход из строя или возникновение неполадки в одной из частей системы не должно приводить к неполадкам или выходу из строя других частей или всей системы.

Рассмотреть применение этого принципа на практике можно на примере приложения, состоящего из трех частей – *web*-интерфейса, с помощью которого пользователи оставляют заказы, сервиса трекинга заказов и сервиса, осуществляющего доставку заказов.

В случае тесной связанности заказы хранятся и вращаются внутри двух сервисов – трекинга и логистики. Если один из этих сервисов выйдет из строя, то это затронет не только его работу, но и работу других частей системы. Если выйдет из строя сервис логистики, то сервис трекинга не сможет переправлять заказы от пользователей (если внутри сервиса трекинга не реализован механизм брокера). Если выйдет из строя сервис трекинга, то заказы пользователей не смогут попасть в систему в принципе – при каждом запросе пользователь будет получать ошибки.

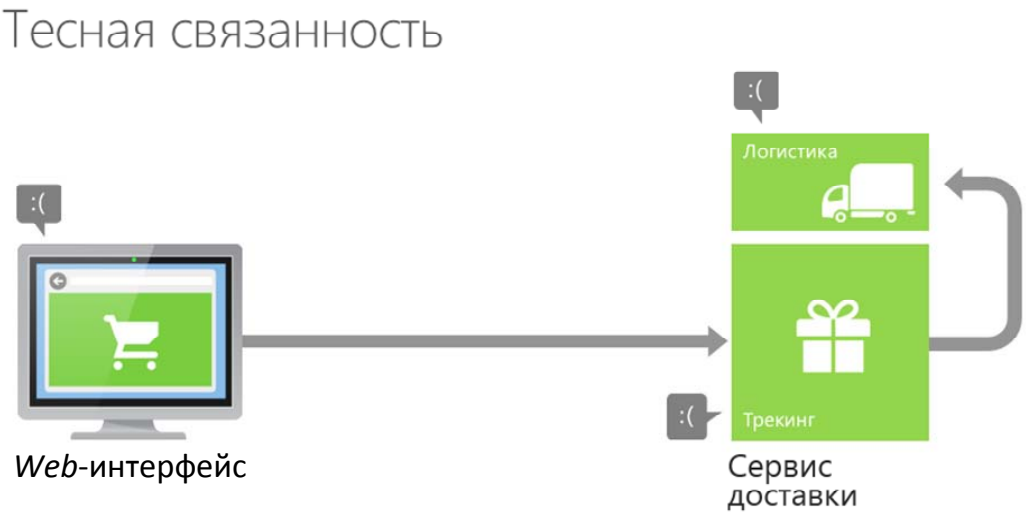

Рис. 7.2. Пример связанности

Для того чтобы решить проблемы тесной связанности в данном сценарии, необходимо интегрировать в приложение еще одну часть, которая будет отвечать за хранение сообщений. Этой частью может стать механизм очереди, который рассматривался в главе, посвященной хранилищу *Windows Azure*. *Windows Azure Service Bus* предоставляет механизм очередей с дополнительной функциональностью.

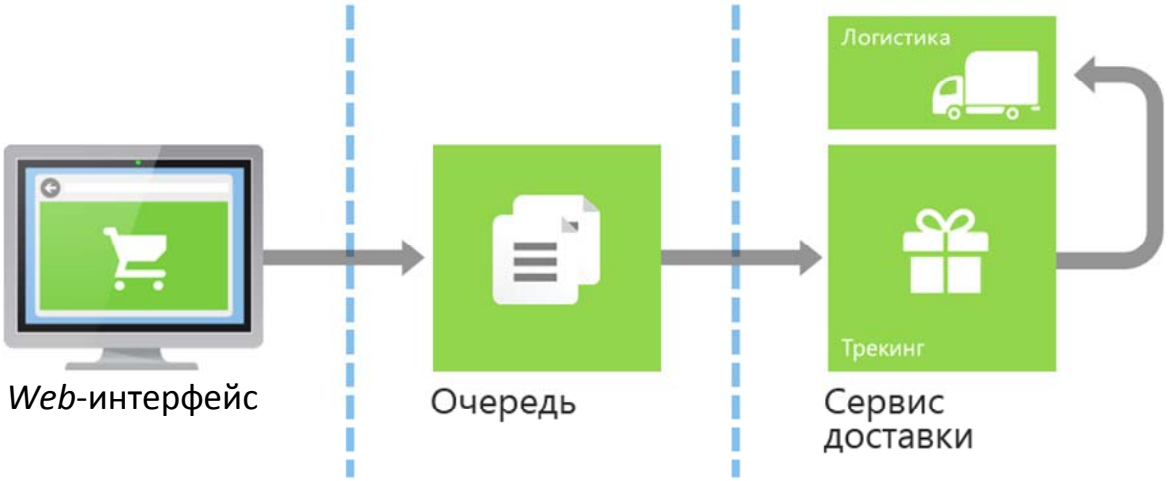

Рис. 7.3. Пример распределенной архитектуры

Добавлением в приложение очереди будет решена проблема доставки сообщений – даже в случае полного выхода из строя серверной части приложения (сервисов трекинга и логистики) сообщения пользователей не будут потеряны – они будут храниться в очереди до тех пор, пока не истечет время их жизни либо не станет доступна серверная часть.

Windows Azure Service Bus оперирует тремя основными понятиями – очередями, топиками и рилеями. Основа любой сервисной шины, и *Windows Azure Service Bus* в том числе – создание пула сообщений и передача их тем клиентам, которым они интересны и для которых они предназначены. Соответственно, с помощью сервисной шины разработчик может реализовать высокомасштабируемые сценарии по осуществлению коммуникаций, подключить необходимое количество клиентов и т. д. Возможными клиентами сервисной шины могут быть *SaaS*-решения, десктопные клиенты, порталы, железные устройства типа телефонов, промышленные, клиенты с переменным доступом к сети, которым необходимо собрать данные и потом отправить их в главный офис. Например, в случае клиентов с переменным доступом к сети сообщения накапливаются в кэше у клиента и, когда появляется подключение, сообщения отправляются в базу.

Первым компонентом *Windows Azure Service Bus* является рилей-маршрутизатор, который позволяет реализовать сценарий «прохода» запросов через *NAT*/брандмауэр и вынести сервис наружу. Это типичный сценарий, в котором за брандмауэром в корпоративном центре обработки данных находится сервис, который нужно вынести

наружу без нарушений регламентов безопасности. *С* помощью рилея этот сценарий решается наиболее безопасным образом.

Второй компонент *Windows Azure Service Bus* – это очереди. Очереди *Windows Azure Service Bus* – это сервис, реализующий стандартную абстракцию программирования и позволяющий обеспечить работу с переменным доступом к сети, и в то же время создать приложения с выравниванием нагрузки, при получении большого количества сообщений же разработчик может выделить несколько очередей и балансировать таким образом нагрузку.

Последним компонентом являются топики и подписки. Этот компонент необходим для решения сценария распространения сообщений среди нескольких клиентов согласно заданным разработчиком правилам. Например, разработчик может создать подписку, подписать на нее определенный набор клиентов, устройств, после чего сообщения, которые придут в эту подписку, будут поступать только этим клиентам.

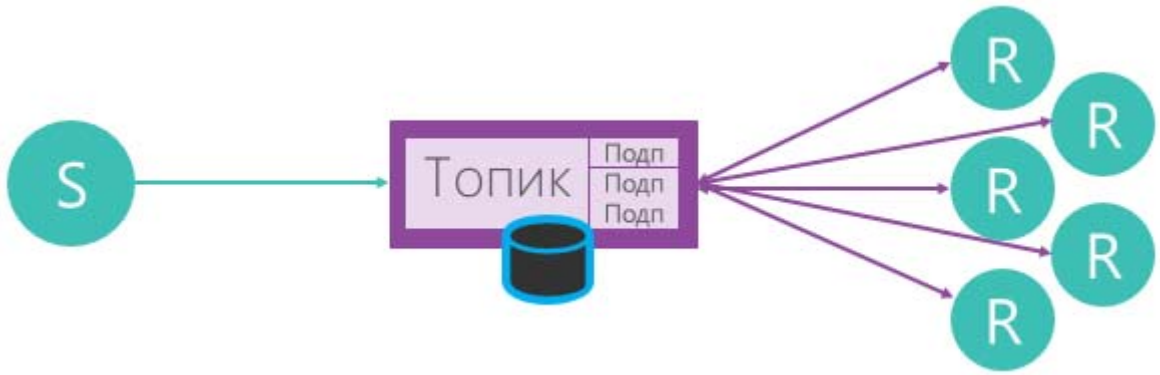

Рис. 7.4. Топики и подписки

Более сложным сценарием использования топиков и подписок является маршрутизатор на базе контента, когда нужно направить сообщение не всем подряд, а передать ее конкретным клиентам. Для этого существуют фильтры, на основании которых сообщения фильтруются и направляются в соответствующие очереди. Фильтры могут иметь один из трех типов: *True*/*NotTrue* (когда выполняется или не выполняется заданное условие), *SQL*-фильтр (когда фильтр задается в *SQL*-синтаксисе) и фильтр *CorrelationId* (когда фильтр отрабатывает, учитывая идентификатор корреляции каждого сообщения).

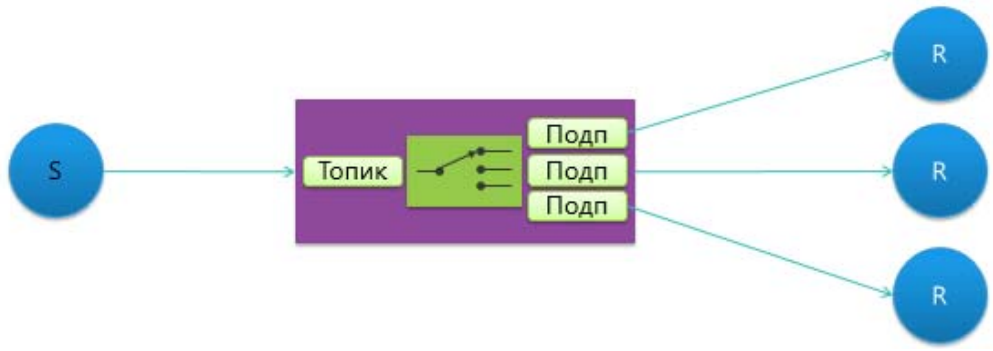

Рис. 7.5. Маршрутизатор на базе контента

Сообщения в очередях и топиках уничтожаются, если у них истекает срок *TTL* – этот механизм помогает защититься от передачи и получения устаревшей информации. Например, если с очень коротким промежутком передаются сообщения с актуальными только некоторый небольшой промежуток времени данными, решение, не удаляющее устаревшие сообщения, может привести к непрогнозируемому поведению системы.

7.1. Сценарии использования *Windows Azure Service Bus*

Первый распространенный сценарий – это *web*-сервис, например, это *web*-сервис, обрабатывающий бизнес-логику работы ритейлера.

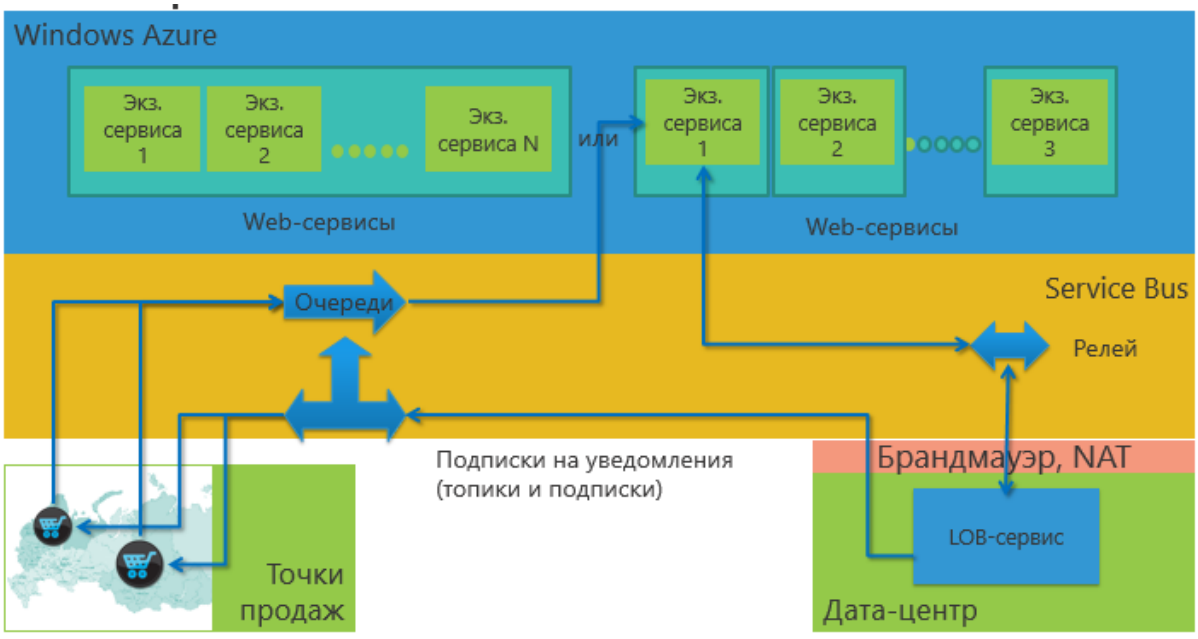

Рис. 7.6. Windows Azure Service Bus:
сценарий – управление очередями заказов

Для решения интеграционной задачи объединения и доставки сообщений по неограниченно большому количеству мобильных клиентов, устройств и точек продаж будут использованы все компоненты *Windows Azure Service Bus*: очереди – для управления очередями заказов. Каждый из агентов, точек продаж, устройств и клиентов отправляет сообщение в очередь о формировании заказа, далее сообщение обрабатывается сервисом, и эти сервисы, обрабатывая заказы, могут обращаться к корпоративному приложению, находящемуся за брандмауэром в корпоративном центре обработки данных, которое управляет данными, которые нельзя выставлять на внешний мир. Для решения этой задачи используются безопасные рилеи *Windows Azure Service Bus*. Для уведомления клиентов о различных действиях системы – например, скидках и другом – используются топики и подписки.

Второй сценарий – специализированный сервис, провайдер вычислительных мощностей для обработки данных по запросу. Пользователи отправляют данные для задачи в облачное хранилище, формируя таким образом очередь задач и сообщение для контроллера (контроллером может выступать *Cloud Service*), принимающего сообщение и решающего, что надо создать новую задачу для обработки данных. Используя сообщения с низкой латентностью внутри инфраструктуры *Windows Azure*, он обращается к системе и выделяет некоторые мощности. Для мониторинга состояния задач используются подписки, и каждый экземпляр обработчика регулярно сообщает о статусе обработки задачи, клиенты же могут подписаться на эти подписки. На эти подписки подписан также контроллер, на основании данных откуда он может принимать решения. Решение работает следующим образом: пользователь отправляет задачи через клиентское приложение; задачи обрабатываются в *HPC*-стиле (распределенном и параллельно обрабатывающемся несколькими обработчиками) на *Windows Azure*; пользователи могут следить за прогрессом и получать уведомления.

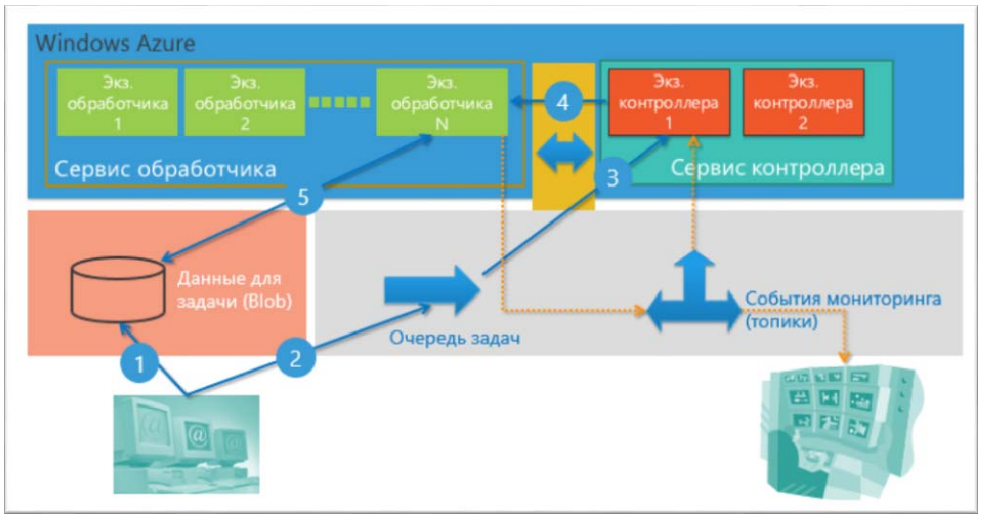

Рис. 7.7. Специализированный сервис, провайдер вычислительных мощностей для обработки данных по запросу

Третий сценарий – это предоставление доступа к внутреннему корпоративному сервису внешним клиентам, вендорам, устройствам. Например, есть данные о сотрудниках, которые необходимо передать в государственное учреждение. Для этого может быть создан рилейный сервис, промежуточный сервис и который будет оперировать с внешним миром. Рилейный сервис, обращаясь к рилею *Windows Azure Service Bus*, создает новый сервис, который находится по этому *URL* (заглушке), который будет использоваться внешними клиентами для обращения к данным за *NAT*. Теперь клиенты могут обращаться к корпоративному сервису или данным по урлу, с помощью которого эти запросы будут безопасно транслироваться в корпоративную закрытую сеть.

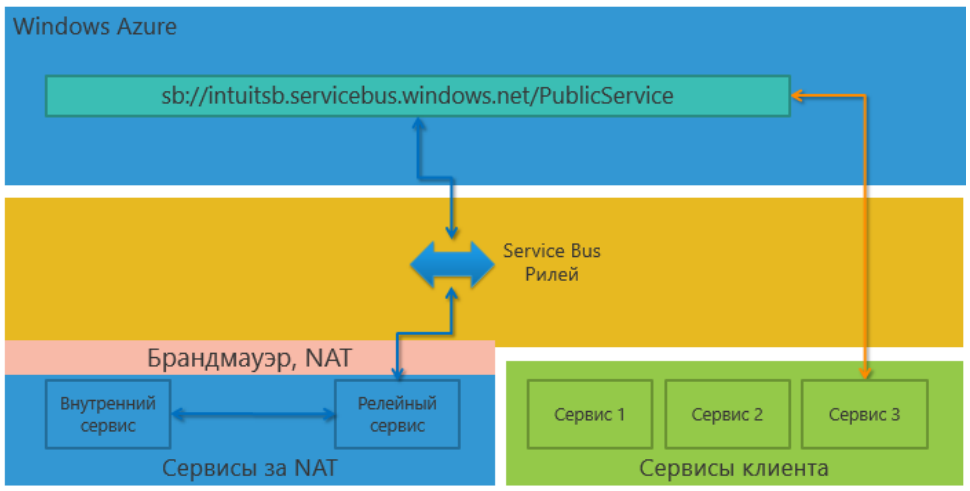

Рис. 7.8. Предоставление доступа к внутреннему корпоративному сервису внешним клиентам

7.2. *Windows Azure Notification Hubs*

Notification Hubs – это сервис, который помогает разработчику, решая проблему управления *Push*-уведомлениями. Вместо реализации собственных механизмов управления разработчик может воспользоваться *Notification Hubs* и получить инфраструктуру, обеспечивающую поддержку следующих сценариев:

• Мультиплатформенность. Сервис *Notification Hubs* объединяет в себе функциональность, необходимую для рассылки уведомлений на популярные платформы (*Windows* 8, *Windows Phone* 8, *iOS*, *Android*), и автоматизирует работу по их формированию и рассылке.

• Рассылка уведомлений на основе правил. Каждый объект, подписанный на *Notification Hub*, может использовать в своей работе один или более тегов-маркеров, позволяющих хабу определить, куда необходимо рассылать уведомления и кто является заинтересованным в этих уведомлениях, а кто их получать не должен.

• Масштабирование. *Notification Hubs* способны реализовать систему, рассылающую уведомления на миллионы подключенных устройств без необходимости внесения изменений в архитектуру системы.

Таким образом, можно отметить, что *Notification Hubs* являются аналогичным, но более мощным механизмом, по сравнению с *Windows Azure Mobile Services Push Notifications*. Безопасность проводимых с *Notification Hub* операций обеспечивается сервисом *Windows Azure Access Control Service*, который позволяет регламентировать доступ к операциям на основе трех основных правил: *Listen* (прослушивание), *Send* (отправка) и *Manage* (управление).

Что касается регистраций устройств, то необходимо учитывать, что регистрации имеют время жизни, которое может быть установлено максимум в 90 дней, после чего регистрации должны быть обновлены.

Транзакции в *Windows Azure Service Bus*

Windows Azure Service Bus предоставляет возможность сущностям *Windows Azure Service Bus* участвовать в транзакциях, что позволяет разработчикам выполнять несколько операций в пределах одной транзакции и быть уверенными в том, что все действия будут выполнены, либо, если возникнет ошибка, ни одно из этих действий не будет выполнено. В контексте выполнения в транзакции не поддерживается операция *Abandon*, необходимая для отказа от сообщений обработки. Все остальные операции могут выполняться в пределах транзакции. Реализацией транзакционного механизма занимается

Windows Azure SQL Databases, которые лежат в основе хранилища для *Windows Azure Service Bus*.

7.3. Определение дубликатов сообщений

Встроенный механизм автоматического определения дубликатов сообщений позволяет избавиться от проблемы хранения одинаковых сообщений, которые могут привести к непрогнозируемому результату. Дубликаты сообщений могут возникать в тех ситуациях, когда разработчик в ответ на ошибку принятия сервером сообщения (например, по тайм-ауту) реализует логику повторной отправки сообщений. Однако может возникнуть ситуация, когда сообщение дошло до сервера, но тем не менее была выдана какая-то ошибка. Тогда возникают дубликаты сообщений.

Автоматическое определение дубликатов сообщений по умолчанию выключено, так как его использование может привести к серьезным нагрузкам: этот механизм использует в своей основе идентификаторы сообщения и заданный разработчиком временной промежуток хранения этих идентификаторов. Чем больше временной промежуток, тем больше дополнительных данных хранится. В системах, оперирующих миллионами сообщений, накладываемая нагрузка может быть существенной.

```
namespaceManager.CreateQueue(
   new QueueDescription(queueName)
   {
      RequiresDuplicateDetection = true,
DuplicateDetectionHistoryTimeWindow = TimeSpan.FromHours(1)
   });
```

Windows Azure Service Bus – это масштабируемый облачный сервис сообщений, доступный по требованию и являющийся совершенным средством интеграции корпоративных автоматизированных систем и сервисов. К компонентам *Windows Azure Service Bus* относятся рилеи, очереди, топики и подписки. *Windows Azure Service Bus* может помочь решить множество различных интеграционных задач, например:

– реализации промышленных архитектурных паттернов;

– интеграции компонент инфраструктуры и облака (гибридный сценарий);

– интеграции своих и сторонних компонент;

– реализации сценариев распределенных систем любой сложности.

7.4. Шаблон проектирования каналов и фильтров

С использованием *Windows Azure Service Bus* можно реализовать шаблон проектирования каналов и фильтров (*Pipes and Filters pattern*).

Данный шаблон позволяет разбить задачу, которая выполняет сложную обработку, на ряд отдельных элементов для повторного использования при необходимости. Так, можно повысить производительность, масштабируемость и возможность многократного использования, позволяя развертывать и масштабировать элементы задачи, которые выполняют обработку, независимо друг от друга.

Контекст и проблема

Приложение должно выполнять различные задачи определенной сложности с информацией, которую оно обрабатывает. Для осуществления такой обработки в виде неделимого модуля применяется простой, но гибкий способ реализации такого приложения. Однако этот метод, вероятно, сократит возможности для рефакторинга кода, его оптимизации или повторного использования, если части одной и той же обработки потребуются в другом месте приложения.

Приложение получает и обрабатывает данные из двух источников. Данные из каждого источника обрабатываются отдельным модулем, который выполняет ряд задач для преобразования этих данных, прежде чем передавать результат в бизнес-логику приложения.

Функции некоторых задач, выполняемых неделимыми модулями, очень похожи, но модули разрабатываются отдельно. Код, который реализует задачи, тесно связан в модуле. Он незначительно или вообще не учитывает задачи масштабируемости и многократного использования.

Тем не менее задачи обработки, выполняемые каждым модулем, или требования к развертыванию для каждой задачи могут изменяться в соответствии с бизнес-требованиями. Некоторые задачи могут быть ресурсоемкими и более эффективными на мощном оборудовании. В то время как для других задач такие дорогостоящие ресурсы могут не требоваться. Кроме того, в будущем может предполагаться дополнительная обработка или может измениться порядок выполнения задач по обработке. Требуется такое решение, которое устранит эти проблемы и увеличит возможность многократного использования кода.

Решение

Разбейте процесс обработки, требуемый для каждого потока, на ряд отдельных компонентов (или фильтров), каждый из которых вы-

полняет определенную задачу. За счет стандартизации формата данных, которые получает и отправляет каждый компонент, эти фильтры можно объединить в конвейер. Так предотвращается дублирование кода и упрощается удаление, замена или интеграция дополнительных компонентов при изменении требований к обработке.

Время, необходимое для обработки одного запроса, зависит от скорости самого медленного фильтра в конвейере. Один или несколько фильтров могут быть сдерживающим фактором, особенно если в потоке из определенного источника данных содержится много запросов. Ключевое преимущество конвейерной структуры заключается в том, что в ней можно параллельно выполнять экземпляры медленных фильтров, позволяя распределить нагрузку и повысить пропускную способность в системе.

Составляющие конвейер фильтры могут выполняться на разных компьютерах, что позволяет масштабировать их независимо друг от друга и использовать возможности эластичности, которые предоставляются в большинстве облачных сред. Ресурсоемкие фильтры могут выполняться на оборудовании с высоким уровнем производительности, а другие, менее ресурсоемкие, могут размещаться на более дешевом стандартном оборудовании. Фильтры не должны находиться в одном центре обработки данных или географическом расположении. Это позволит каждому элементу в конвейере работать в среде, близкой к необходимым ресурсам.

Если входные и выходные данные фильтра структурированы в виде потока, обработка для каждого фильтра может выполняться в параллельном режиме. Первый фильтр в конвейере может начать свою работу и вернуть результаты, которые непосредственно передаются следующему фильтру в последовательности, прежде чем первый фильтр завершит свою работу.

Еще одно преимущество, предоставляемое этой моделью, – устойчивость. Если произойдет сбой фильтра или компьютер, на котором он выполняется, станет недоступен, конвейер позволит перепланировать операции, выполняемые фильтром, и передать их другому экземпляру компонента. Сбой одного фильтра не обязательно приводит к сбою всего конвейера.

Альтернативный подход к реализации распределенных транзакций – использовать шаблон каналов и фильтров в сочетании с шаблоном компенсирующих транзакций. Распределенные транзакции можно разделить на отдельные компенсируемые задачи, каждую из

которых можно реализовать с помощью фильтра, также реализуемого с помощью шаблона компенсирующих транзакций. Фильтры в конвейере можно реализовать в виде отдельно размещенных задач, выполняемых в непосредственной близости к данным, которыми они управляют.

Проблемы и рекомендации

При выборе схемы реализации этого шаблона следует учитывать следующие особенности:

Сложность. Повышенная гибкость, которую обеспечивает этот шаблон, может также вызвать сложности, особенно если фильтры в конвейере распределяются между разными серверами.

Надежность. Используйте инфраструктуру, которая гарантирует сохранность данных, передаваемых между фильтрами в конвейере.

Идемпотентность. Если после получения сообщения происходит сбой фильтра в конвейере и операция переносится на другой экземпляр фильтра, часть операции может быть уже выполнена. Если эта операция обновляет некоторые аспекты глобального состояния (например, сведения, хранящиеся в базе данных), это обновление можно повторить.Такая же проблема может произойти после передачи результатов фильтра следующему фильтру в конвейере, но до того, как появится сообщение об успешном завершении работы фильтра. В этих случаях другой экземпляр фильтра может повторить эти операции, что приведет к тому, что одни и те же результаты будут переданы дважды. В результате следующие фильтры в конвейере могут дважды обработать одни и те же данные. Поэтому фильтры в конвейере должны разрабатываться идемпотентными. Дополнительные сведения смотрите в описании шаблонов идемпотентности в блоге Джонатана Оливера (Jonathan Oliver).

Повторяющиеся сообщения. Если сбой фильтра в конвейере происходит после публикации сообщения для следующего этапа конвейера, может запуститься другой экземпляр фильтра и опубликовать копию этого сообщения в конвейере. Это может привести к передаче следующему фильтру двух экземпляров одного сообщения. Чтобы избежать этого, конвейер должен обнаруживать и удалять дублирующие сообщения.

Если реализуется конвейер с использованием очередей сообщений (например, очередей служебной шины *Microsoft Azure*), в инфраструктуре очередей сообщений может быть предусмотрено автоматическое обнаружение и удаление дубликатов сообщений.

Контекст и состояние. В конвейере каждый фильтр, по сути, выполняется изолированно. Способ его вызова не должен вызывать предположения. Это означает, что каждому фильтру нужно предоставить достаточный контекст для выполнения операций. Этот контекст может содержать много информации о состоянии.

Когда следует использовать этот шаблон:

– процесс обработки, требуемый приложением, можно легко разделить на ряд независимых этапов;

– у этапов обработки, которые выполняются приложением, разные требования к масштабируемости;

– требуется определенная гибкость, чтобы изменять порядок шагов обработки, выполняемых приложением, или добавлять и удалять шаги;

– система будет работать эффективнее, если распределить шаги обработки между несколькими серверами;

– чтобы свести к минимуму последствия сбоя на шаге при обработке данных, требуется надежное решение.

Этот шаблон может оказаться неэффективным в следующих случаях:

– шаги обработки, выполняемые приложением, не являются независимыми или должны выполняться вместе в рамках одной транзакции;

– объем контекста или сведений о состоянии, необходимых для выполнения шага, делают такой подход неэффективным. Вместо этого можно сохранить информацию о состоянии в базе данных.

Пример

Чтобы предоставить инфраструктуру, необходимую для реализации конвейера, можно использовать последовательность очередей сообщений. В начальную очередь поступают необработанные сообщения. Компонент, реализованный в виде задачи фильтра, ожидает передачи сообщения в очереди, выполняет свою операцию и затем передает преобразованное сообщение в следующую очередь в последовательности. Другая задача фильтра может ожидать передачи сообщения в этой очереди, обрабатывать их и отправлять результаты в другую очередь. И так пока в последнем сообщении в очереди не появятся полностью преобразованные данные.

Можно использовать очереди служебной шины, чтобы создать надежный и масштабируемый механизм организации очереди. В примере класса *ServiceBusPipeFilter* на языке *C#* ниже показано, как

можно реализовать фильтр, который получает входящие сообщения из очереди, обрабатывает их и передает результаты в другую очередь.

```
public class ServiceBusPipeFilter
{
  ...
  private readonly string inQueuePath;
  private readonly string outQueuePath;
  ...
  private QueueClient inQueue;
  private QueueClient outQueue;
  ...
  public ServiceBusPipeFilter(..., string inQueuePath, string
outQueuePath = null)
  {
    ...
    this.inQueuePath = inQueuePath;
    this.outQueuePath = outQueuePath;
  }
  public void Start()
  {
    ...
    this.outQueue = QueueClient.CreateFromConnectionString(...);
    ...
   this.inQueue = QueueClient.CreateFromConnectionString(...);
  }
  public void OnPipeFilterMessageAsync(
    Func<BrokeredMessage, Task<BrokeredMessage>>
asyncFilterTask, ...)
  {
    ...
    this.inQueue.OnMessageAsync(
      async (msg) =>
    {
      ...
      var outMessage = await asyncFilterTask(msg);
      if (outQueue != null)
      {
        await outQueue.SendAsync(outMessage);
      }
```

```
    },
    options);
  }
  public async Task Close(TimeSpan timespan)
  {
    this.pauseProcessingEvent.Reset();
    Thread.Sleep(timespan);
    this.inQueue.Close();
    ...
  }
  ...
}
```

Метод *Start* в классе *ServiceBusPipeFilter* позволяет подключиться к паре очередей ввода и вывода, а метод *Close* – отключиться от очереди ввода. Метод *OnPipeFilterMessageAsync* выполняет фактическую обработку сообщений, а параметр *asyncFilterTask* для этого метода указывает, какую именно обработку следует выполнить. Метод *OnPipeFilterMessageAsync* ожидает поступления сообщений в очередь ввода, выполняет код, определяемый параметром *asyncFilterTask* для каждого сообщения по мере их поступления и передает результаты в очередь вывода. Сами очереди определяются с помощью конструктора.

В примере решения фильтры реализуются в наборе рабочих ролей. Каждую рабочую роль можно масштабировать автономно в зависимости от сложности обработки коммерческих данных или ресурсов, необходимых для обработки. Кроме того, можно параллельно выполнять несколько экземпляров каждой рабочей роли, чтобы повысить пропускную способность.

В следующем коде показана рабочая роль *Azure* с именем *PipeFilterARoleEntry*, определенная в проекте *PipeFilterA* в примере решения.

```
public class PipeFilterARoleEntry : RoleEntryPoint
{
  ...
  private ServiceBusPipeFilter pipeFilterA;
  public override bool OnStart()
  {
    ...
    this.pipeFilterA = new ServiceBusPipeFilter(
```

```
            ...,
            Constants.QueueAPath,
            Constants.QueueBPath);
           this.pipeFilterA.Start();
          ...
        }
        public override void Run()
        {
          this.pipeFilterA.OnPipeFilterMessageAsync(async (msg) =>
          {
            var newMsg = msg.Clone();
            await Task.Delay(500); // DOING WORK
            Trace.TraceInformation("Filter A processed message:{0} at {1}",
              msg.MessageId, DateTime.UtcNow);
            newMsg.Properties.Add(Constants.FilterAMessageKey,
"Complete");
            return newMsg;
          });
          ...
        }
        ...
      }
```

Эта роль содержит объект *ServiceBusPipeFilter*. Метод *OnStart* в роли позволяет подключиться к очередям для получения входных сообщений и передачи выходных сообщений (имена очередей определены в классе *Constants*). Метод *Run* вызывает метод *OnPipeFilterMessagesAsync* для обработки каждого полученного сообщения. (В этом примере обработка имитируется ожиданием в течение короткого периода времени.) По завершении обработки создается новое сообщение, содержащее результаты (в этом случае во входящее сообщение добавлено пользовательское свойство), и это сообщение передается в очередь вывода.

Пример кода содержит еще одну рабочую роль с именем *PipeFilterBRoleEntry* в проекте *PipeFilterB*. Эта роль аналогична роли *PipeFilterARoleEntry* за исключением того, что она выполняет другие операции обработки в методе *Run*. В примере решения эти две роли объединяются для создания конвейера, где очередь вывода для роли *PipeFilterARoleEntry* является очередью ввода для роли *PipeFilterBRoleEntry*.

В примере решения также предоставлены две дополнительные роли с именами *InitialSenderRoleEntry* (в проекте *InitialSender*) и

FinalReceiverRoleEntry (в проекте *FinalReceiver*). Роль *InitialSenderRoleEntry* предоставляет начальное сообщение в конвейере. Метод *OnStart* позволяет подключиться к одной очереди, а метод *Run* передает метод в эту очередь. Эта очередь является очередью ввода, которая используется ролью *PipeFilterARoleEntry*, поэтому при передаче сообщения в нее это сообщение принимается и обрабатывается ролью *PipeFilterARoleEntry*. Затем обработанное сообщение передается через роль *PipeFilterBRoleEntry*.

Очередь ввода для роли *FinalReceiveRoleEntry* является очередью вывода для роли *PipeFilterBRoleEntry*. Метод *Run* в роли *FinalReceiveRoleEntry*, показанный ниже, получает сообщение и выполняет некоторые задачи окончательной обработки. Затем он записывает значения пользовательских свойств, добавленных фильтрами в конвейере, чтобы выполнить трассировку выходных данных.

```
public class FinalReceiverRoleEntry : RoleEntryPoint
{
  ...
  private ServiceBusPipeFilter queueFinal;
  public override bool OnStart()
  {
    ...
    this.queueFinal = new
ServiceBusPipeFilter(...,Constants.QueueFinalPath);
    this.queueFinal.Start();
    ...
  }
  public override void Run()
  {
    this.queueFinal.OnPipeFilterMessageAsync(
      async (msg) =>
      {
        await Task.Delay(500); // DOING WORK
        Trace.TraceInformation(
          "Pipeline Message Complete - FilterA:{0} FilterB:{1}",
          msg.Properties[Constants.FilterAMessageKey],
          msg.Properties[Constants.FilterBMessageKey]);
        return null;
      });
    ...
  } ...}
```

Вопросы для самопроверки

1. Распределенная обработка данных, *Cloud Computing*, концепция «облачной» обработки данных.

2. Предпосылки возникновения облачных информационных систем.

3. Технологии построения облачных информационных систем.

4. Требования к облачным информационным системам.

5. Особенности построения облачных информационных систем на основе технологии *Windows Azure*.

6. Основные модели предоставления услуг облачных вычислений: *Software as a Service* (*SaaS*) (ПО-как-услуга), *Platform as a Service* (*PaaS*), Инфраструктура как сервис (*Infrastructure as a Service*, *IaaS*), другие облачные сервисы (*XaaS*).

7. Различия между облачными и кластерными (распределенными, или *Grid*-технологиями) вычислениями.

8. Платформа *Windows Azure*.

9. Использование облачной платформы *Windows Azure* для разработки приложений.

10. Инструментальные средства *Windows Azure*.

11. Развертывание приложений на платформу *Windows Azure* с использованием сервисов *Windows Azure Web Sites* и *Windows Azure Cloud Services*.

12. Создание и развертывание простого web-приложения *ASP.NET MVC 4* с помощью сервисов *Windows Azure Web Sites* и *Windows Azure Cloud Services* и *Visual Studio 2012*.

Разработка приложений с *Windows Azure Cloud Services*.

13. Использование *Windows Azure* как *Platform-As-A-Service*, архитектура, использование, разработка многослойных приложений *ASP.NET*. Развертывание облачных приложений в облако, сложное масштабирование всех слоев по отдельности.

14. Архитектура *Windows Azure Cloud Services*, конфигурация *Cloud Service*, масштабирование *Cloud Service*, использование *Windows Azure Tools for Visual Studio*, *Windows Azure SDK*.

15. Авторизация и безопасность с *Windows Azure Active Directory*.

16. Введение в технологии аутентификации на базе утверждений, реализация сценариев аутентификации с использованием технологий *Microsoft*, сценарий интеграции облачного приложения с локальной инфраструктурой *Active Directory* для реализации *Single Sign-On* и федеративной аутентификации.

17. Аутентификация на базе утверждений. Федеративная аутентификация в *Windows Azure* с использованием публичных провайдеров идентификации и доменного каталога *Active Directory*.

18. Программная реализация проверки токенов безопасности на стороне клиента.

19. Программная модель *Windows Identity Foundation*.

20. Технологии *Windows Azure Access Control Service*, *Active Directory Federation Services 2.0*. Многофакторная проверка подлинности *Windows Azure*.

21. Облачные системы хранения и обработки корпоративных данных.

22. Хранение и обработка данных с *Windows Azure Storage* и *Windows Azure SQL Databases*.

23. Сценарий приложения *Cloud Services*, использующего для хранения данных: блобы, таблицы и очереди *Windows Azure*.

24. Сценарий приложения *Cloud Services* с хранением данных в базе данных.

25. Введение в масштабирование баз данных *Windows Azure* – федерации, шардинг.

26. Бизнес-аналитика и анализ данных с *SQL Reporting* и *Hadoop*.

27. Введение в бизнес-аналитику.

28. Введение в парадигму *MapReduce*.

29. Приложения для бизнес-аналитики с *SQL Reporting*.

30. Приложение, анализирующее логи, с использованием *M/R Hadoop*, в *Windows Azure*.

31. Доступ к сервисам предприятия с *Windows Azure Service Bus*.

32. Принципы осуществления доступа к сервисам предприятия с использованием *Service Bus* в *Cloud Services* для безопасной и надежной передачи данных.

33. Интеграция облачного приложения с сервисом предприятия.

34. Использование технологии *Windows Azure Service Bus*.

35. *Windows Azure Notification Hubs*.

36. Транзакции в *Windows Azure Service Bus*.

37. Определение дубликатов сообщений.

38. Разработка приложений высокой готовности в *Windows Azure*. Назначение, примеры использования.

39. *DAO*-слой доступа к данным в *Windows Azure*.

Литература

1. Федоров, А. Windows Azure. Облачная платформа Microsoft / А. Федоров, Д. Мартынов. – М. : Рус. ред., 2010. – 96 с.

2. OpenStack. – Режим доступа: http://www.openstack.org/. – Дата доступа: 05.03.2019.

3. Cloud Foundry. – Режим доступа: http://www.cloudfoundry.com/. – Дата доступа: 15.03.2019.

4. Топ 10 облачных платформ для бизнеса. – Режим доступа: http://www.livebusiness.ru/news/8937/. – Дата доступа: 15.03.2019.

5. Amazon Web Sevices. – Режим доступа: http://aws.amazon.com/. – Дата доступа: 15.03.2019.

6. Rockspace. – Режим доступа: http://www.rackspace.com/. – Дата доступа: 15.03.2019.

7. Rackspace Cloud Tools Marketplace. – Режим доступа: http://www.rackspace.com/cloud/tools/. – Дата доступа: 15.03.2019.

8. Платформа Windows Azure. – Режим доступа: http://www.windowsazure.com/ru-ru/. – Дата доступа: 10.11.2018.

9. Google App Engine. – Режим доступа: https://developers.google.com/appengine/?hl=ru. – Дата доступа: 15.03.2019.

10. Force.com. – Режим доступа: http://www.force.com/. – Дата доступа: 15.03.2019.

11. Salesforce.com. – Режим доступа: http://www.salesforce.com/. – Дата доступа: 15.03.2019.

12. VMware vCloud® Suite. – Режим доступа: http://www.vmware.com/products/datacenter-virtualization/vcloud-suite/overview.html. – Дата доступа: 15.03.2019.

13. IBM Smart Cloud. – Режим доступа: http://www.ibm.com/cloud-computing/us/en/. – Дата доступа: 15.03.2019.

Учебное электронное издание комбинированного распространения

Учебное издание

Стефановский Игорь Леонидович

ВВЕДЕНИЕ В ОБЛАЧНЫЕ ВЫЧИСЛЕНИЯ

Пособие

Электронный аналог печатного издания

Редактор *А. В. Власов*
Компьютерная верстка *М. В. Кравцова*

Подписано в печать 20.11.19.

Формат 60x84/16. Бумага офсетная. Гарнитура «Таймс».
Ризография. Усл. печ. л. 6,74. Уч.-изд. л. 7,05.
Изд. № 34.
http://www.gstu.by

Издатель и полиграфическое исполнение
Гомельский государственный
технический университет имени П. О. Сухого.
Свидетельство о гос. регистрации в качестве издателя
печатных изданий за № 1/273 от 04.04.2014 г.
пр. Октября, 48, 246746, г. Гомель