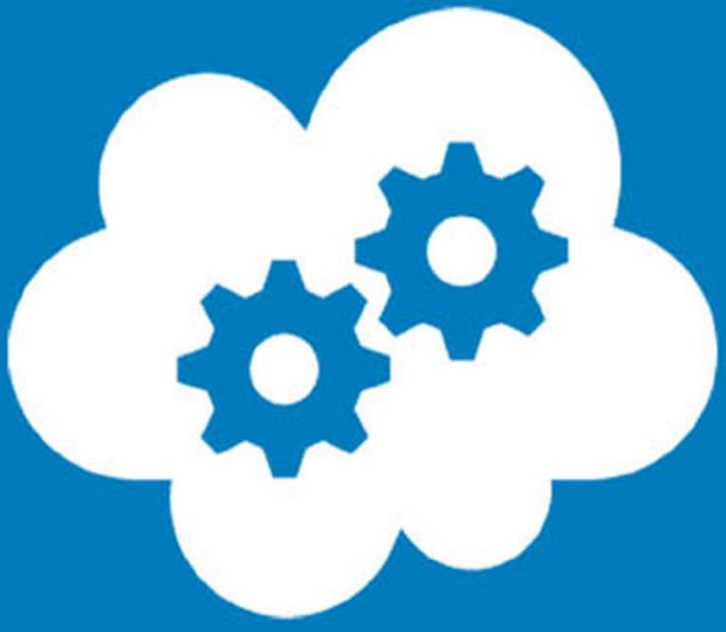
Миграция приложений в Windows Azure, 2-й выпуск

patterns & practices

Справочник\руководство



Microsoft

Миграция приложений в Windows Azure, 2-й выпуск

patterns & practices

Обзор: Это первая книга в серии patterns & practices, посвящённой платформе Windows Azure. В книге рассматривают различные аспекты (изменения в архитектуре, вопрос стоимости, мониторинг и диагностика системы и т.п.) миграции локального приложения на облачную платформу Microsoft – Windows Azure.

В книге описана миграция на примере приложения aExpense, это ASP.NET приложение согласования авансовых отчетов. Во-первых, книга поможет получить хорошее представление о тех вопросах, которые необходимо учесть при миграции приложения в облако, а, во-вторых, в книге даются ответы на ряд практических задач, которые являются актуальными не только при миграции приложения, но и просто при разработке любого облачного приложения.

Категория: Справочник\руководство
Аудитория: Windows Azure
Источник: patterns & practices
Дата публикации электронной книги: Сентябрь 2012

Microsoft

Оглавление

Microsoft

Введение

Каксоздавать масштабируемые приложения с высоким уровнем доступности? При разработке приложений необходимо также помимо непосредственного написания кода, решать инфраструктурные и. Может понадобиться добавить новые серверы или увеличить мощность уже существующих, добавить избыточное оборудование, прикладную логику обработки распределенных вычислений и обработки отказа. Это необходимо сделать даже в том случае, если приложение востребовано лишь в течение коротких периодов времени.

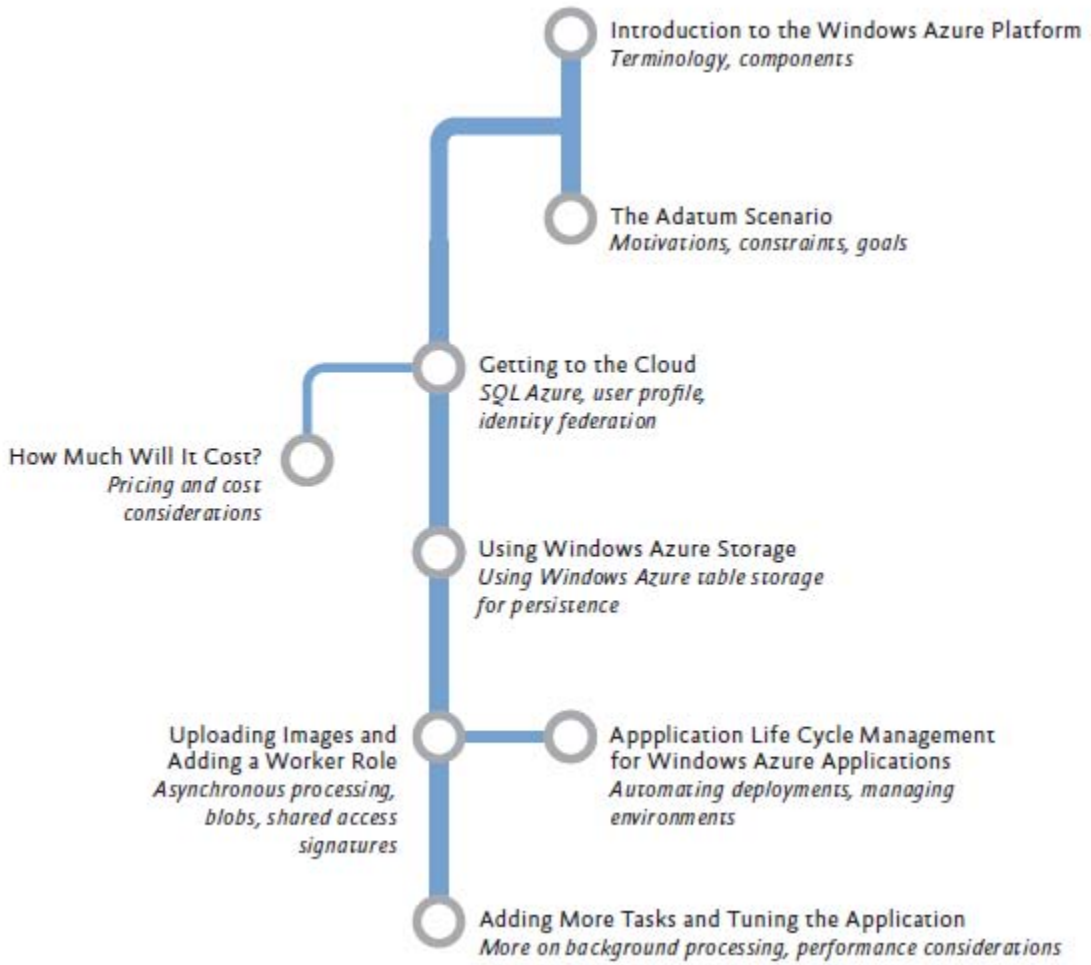
Ответом на данный вопрос может быть использование облака. Облако состоит из набора взаимосвязанных серверов, размещенных в различных центрах обработки данных. Но для вас это единый центр, управление которым и размещение на котором взял на себя некто другой. Сложив с себя ответственность за сопровождение инфраструктуры, вы сможете сконцентрировать усилия на самом важном — на приложении.

Загрузки	Пример кода
Учебное содержимое	Практические занятия
Печатная и электронная книга	Эта книга скоро будет доступна в печатном и электронном виде.
Сообщество	http://wag.codeplex.com
Лицензия	Лицензия на шаблоны и методики Microsoft, сентябрь 2009

Общие сведения

Это первая книга из серии, посвященной платформе Windows® Azure™. В ней показано, каким образом можно адаптировать обычное приложение ASP.NET к работе в облаке. Книга предназначена для профессиональных архитекторов систем, разработчиков и специалистов по информационным технологиям, которые занимаются проектированием, созданием или эксплуатацией приложений и служб, подходящих для работы в облаке. Эта книга в первую очередь адресована тем, кто работает с системами на основе Windows. Требуется знание Microsoft .NET Framework, Microsoft Visual Studio®, ASP.NET, SQL Server и Microsoft Visual C#®.

Общие сценарии



Раздел «Введение в платформу Windows Azure» предоставляет обзор платформы, который дает возможность начать работать в Windows Azure. В нем описаны веб-роли и рабочие роли, а также различные способы хранения данных в Windows Azure. Целесообразно ознакомиться с этим разделом, прежде чем переходить к сценариям.

Раздел «Сценарий Adatum» представляет компанию Adatum и приложение aExpense. В последующих главах описано, как в компании Adatum провели миграцию приложения aExpense в облако. Эта глава поможет понять, почему в Adatum решили провести миграцию некоторых своих приложений для бизнеса в облако

Раздел «Путь в облако» описывает первые шаги, предпринятые компанией Adatum для подготовки к миграции приложения aExpense. Цель компании Adatum состояла просто в том, чтобы приложение начало работать в облаке, однако это создавало и «большие» проблемы, связанные с безопасностью и хранением данных.

Раздел «Сколько стоит?» представляет базовую ценовую модель работы приложения aExpense в среде Windows Azure и содержит расчеты оценочной стоимости годовых затрат на это приложение. Эта глава необязательна. Она не является необходимой для перехода к последующим сценариям.

Раздел «Автоматизация развертывания и использования хранилища Windows Azure» описывает, как в компании Adatum использовали скрипты PowerShell и Microsoft Build Engine (MSBuild) для автоматизации развертывания aExpense в Windows Azure. Здесь также описано, как в компании Adatum перевели приложение aExpense с SQL Azure на использование табличного хранилища Windows Azure, а также обсуждаются различия этих двух моделей хранилища.

Раздел «Передача изображений и добавление рабочей роли» описывает, как добавить рабочую роль в приложение aExpense, и показывает, как в приложении aExpense используется хранилище больших двоичных объектов Windows Azure (BLOB объекты) для хранения отсканированных изображений.

Раздел «Управление жизненным циклом приложения для приложений Windows Azure» описывает, как производится управление разработкой, тестированием и развертыванием приложений Windows Azure. Эта глава необязательна. Ознакомление с ней не является обязательным для перехода к последнему сценарию.

Раздел «Добавление задач и настройка приложения» показывает, как в Adatum добавляли дополнительные задачи рабочей роли для приложения aExpense. На этом этапе также производилась оценка результатов тестирования, производительности приложения и на основании полученных результатов вносились некоторые изменения.

Требования к аудитории

Книга предназначена для профессиональных архитекторов систем, разработчиков и специалистов по информационным технологиям, которые занимаются проектированием, созданием или эксплуатацией приложений и служб, подходящих для работы в облаке. Хотя приложения для работы в Windows Azure не обязательно должны создаваться для работы в операционной системе Microsoft® Windows®, эта книга написана для тех, кто работает с системами на базе Windows. Требуется знание Microsoft .NET Framework, Microsoft Visual Studio®, ASP.NET и Microsoft Visual C#®.

Системные требования

Для запуска сценариев необходимо выполнение следующих требований:

- Microsoft Windows 7 с пакетом обновления 1 (SP1) или Windows Server 2008 R2 с пакетом обновления 1 (SP1) (32-разрядный и 64-разрядный выпуски)
- Microsoft .NET Framework версии 4.0
- Выпуск Microsoft Visual Studio 2010 Ultimate, Premium или Professional с установленным пакетом обновления 1 (SP1)
- Пакет Windows Azure SDK для .NET (включает средства Visual Studio для Windows Azure)
- Microsoft SQL Server или SQL Server Express 2008
- Windows Identity Foundation. Требуется для авторизации на основе утверждений.
- Библиотека Enterprise Library 5 (необходимые сборки включены в папки с исходным кодом)

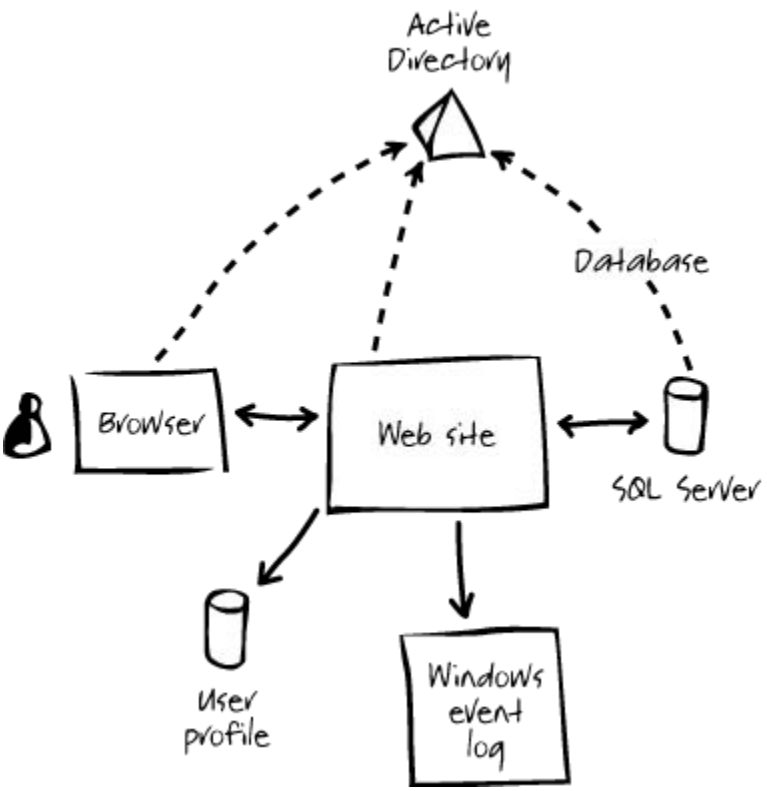
- WatiN 2.0. Откройте диалоговое окно «Свойства» и разблокируйте заархивированный файл после его загрузки, но до извлечения содержимого. Поместите содержимое в папку Lib в примерах.
- Microsoft Anti-Cross Site Scripting Library V4. Поместите в папку Lib в примерах.

Цели разработки

Это первое руководство посвящено главным образом сценариям миграции. В нем представлена вымышленная компания Adatum и ее пошаговые действия по адаптации системы отслеживания и возмещения расходов aExpense для развертывания в Windows Azure.

Приложение aExpense написано на ASP.NET 3.5, развернуто в центре данных Adatum, и доступ к нему производится из интрасети Adatum. Для проверки подлинности сотрудников используется Microsoft Active Directory®. Кроме того, через Active Directory производится доступ к необходимым приложению данным из профилей пользователей, например руководителя и центра расходов служащего.

Для хранения своих данных приложение aExpense использует простую базу данных SQL Server®, а в качестве механизма доступа к данным используется LINQ to SQL. Приложение настроено для соединения с SQL Server через встроенную систему безопасности, а для входа в базу данных веб-сайт использует учетную запись службы.



Компанией Adatum намечено несколько конкретных целей. Приложение aExpense в облаке должно иметь доступ к тем же данным, что и его локальная версия. Это относится как к данным о

расходах предприятия, которые обрабатываются приложением, так и к данным профиля пользователей — центра расходов пользователя и руководителя, которые необходимы для применения бизнес-правил в приложении. Однако в компании Adatum решили убрать из aExpense какую-либо зависимость от Active Directory и исключить обратные обращения приложения из облака в Adatum.

Вторая цель — убедиться в том, что операторы имеют доступ к той же диагностической информации в версии aExpense для облака, что и в существующей локальной версии приложения.

Важным вопросом для Adatum является безопасность версии для облака, поэтому третья цель состоит в управлении доступом к приложению aExpense на основе учетных данных, управляемых из Adatum, и предоставлении пользователям возможности получать доступ к приложению по имеющимся у них учетным данным. В Adatum стремятся избежать накладных расходов по управлению дополнительными системами безопасности для приложений на базе облака.

Общей целью является миграция приложения aExpense в облако с сохранением при этом опыта работы пользователей и управляемости приложения, а также со сведением к минимуму вносимых изменений.

В последующих главах руководства вводятся более сложные требования и рассматриваются различные компоненты Windows Azure, реализующие их.

Сообщество

Данное руководство, подобно многим материалам по шаблонам и методикам, связано с сайтом сообщества. Здесь можно задать вопросы, высказать свое мнение, пообщаться с другими пользователями и обменяться идеями. Члены сообщества могут оказывать помощь Microsoft в планировании и тестировании будущих руководств, а также загружать расширения, учебные пособия и другие дополнительные материалы.

Дополнительные руководства

Второе руководство серии «Разработка приложений для облака» доступно по адресу http://msdn.microsoft.com/ru-ru/library/ff966499.aspx.

Для этого сценария было разработано расширение для мобильных пользователей, использующих устройства Windows Phone 7. Дополнительные сведения см. на веб-сайте http://msdn.microsoft.com/ru-ru/library/gg490765.aspx.

В третьей части планируется рассмотреть сценарии интеграции и новые возможности. Дополнительные сведения см. на веб-сайте http://wag.codeplex.com.

Глава 1. Введение в платформу Azure

В этой главе дано краткое описание технологической платформы Microsoft® Windows® Azure™, предоставляемых ей услуг и возможностей, доступных по запросу, а также вычислений в облаке, где под облаком понимается набор взаимосвязанных вычислительных ресурсов, размещенных в одном или нескольких центрах обработки данных. В этой главе также приведены ссылки, по которым можно получить дополнительные сведения о возможностях Windows Azure, методиках и технологиях, используемых в этой серии руководств, и иллюстрирующие их образцы кода.

О платформе Windows Azure

Облако дает возможность разработчику развертывать и запускать приложения, а также обеспечивает хранение данных. Локальные приложения по-прежнему могут пользоваться облачными ресурсами. Например, размещенное на локально сервере приложение, полнофункциональный клиент на настольном компьютере и клиент на мобильном устройстве могут пользоваться хранилищем в облаке.

Платформа Windows Azure делает аппаратные ресурсы абстрактными, виртуализируя их. Каждое развернутое в Windows Azure приложение выполняется на одной или нескольких виртуальных машинах. Развернутые приложения работают так, как будто они размещены на выделенном компьютере, хотя могут совместно с другими виртуальными машинами на том же физическом узле использовать физические ресурсы — место на диске, сетевой ввод-вывод, ядра ЦП и виртуальные машины на одном физическом сервере. Ключевыми преимуществами уровня абстракции поверх физического оборудования являются портативность и масштабируемость. Виртуализация служб позволяет перемещать их на любое число узлов в центре обработки данных. Используя виртуальные технологии, неспециализированное оборудование, мультитенантность и агрегирование потребностей, Microsoft достигает экономии за счет масштаба. Это приводит к более полному использованию ресурсов центра обработки данных (и, соответственно, к большей выработке на доллар затрат на оборудование) и соответствующей экономии, которая отражается и на ваших затратах.

Комментарий Бхарата:

Платформа Windows Azure поможет добиться переносимости и масштабируемости ваших приложений, а также сократить текущие расходы и снизить общую стоимость владения.

Виртуализация позволяет реализовать и вертикальную масштабируемость, и горизонтальную масштабируемость. Вертикальная масштабируемость означает, что по мере роста потребностей на конкретной виртуальной машине можно наращивать ресурсы — количество ядер ЦП и объем оперативной памяти. Горизонтальная масштабируемость означает, что можно добавлять новые экземпляры виртуальной машины, которые являются копиями существующих служб. Все эти экземпляры используются при балансировке нагрузки на сетевом уровне, поэтому входящие запросы распределяются между ними.

На момент написания этого материала платформа Windows Azure включает три основных компонента: Windows Azure, Windows Azure AppFabric (сейчас это Windows Azure Building Blocks: Service Bus, Access Control Services, Windows Azure Caching и т.п.) и SQL Azure.

Windows Azure реализует вычислительную среду на основе Microsoft Windows® Server для приложений и постоянное хранилище как для структурированных, так и для неструктурированных данных, а также асинхронный обмен сообщениями.

Windows Azure AppFabric (сейчас это Windows Azure Building Blocks: Service Bus, Access Control Services, Windows Azure Caching и т.п) предоставляет целый ряд служб, помогающих связать пользователей и локальные приложения с приложениями в облаке, управлять проверкой подлинности и данными, а также выполнять связанные с этим функции, например кэширование.

SQL Azure по сути дела представляет собой SQL Server®, работающий в виде службы в облаке.

Платформа включает также ряд служб управления, позволяющих управлять всеми этими ресурсами программно или через пользовательский веб-интерфейс (веб-портал). В большинстве случаев имеется API на основе REST, с помощью которого можно определить способ работы служб. Большинство задач управления, которые могут выполняться через веб-портал, можно выполнить и с помощью API.

Наконец, имеется полный набор средств и пакетов средств разработки программного обеспечения для разработки, тестирования и развертывания приложений. Например, разрабатывать и тестировать приложения можно в эмулируемой локальной среде, обеспечиваемой эмулятором вычислений и эмулятором хранения. Большинство средств также интегрируются в такую среду разработки, как Microsoft Visual Studio®. Кроме того, имеются и средства управления от сторонних производителей.

Службы и компоненты Windows Azure

Ряд служб и компонентов Windows Azure, Windows Azure Building Blocks и платформа SQL Azure предъявляют к приложениям специфические требования. При подписке на Windows Azure можно выбрать нужные компоненты и платить только за их использование. Добавить компоненты в подписку и удалить их оттуда можно в любой момент. Механизм выставления счетов за каждую службу зависит от типа компонента, предоставляющего службу. Дополнительные сведения о модели выставления счетов см. ниже в этой главе в разделе «Подписка на Windows Azure и модель выставления счетов».

По мере развития платформы Windows Azure службы и компоненты продолжают изменяться. Данная серия руководств с прилагаемыми образцами кода и сопутствующими практическими занятиями показывает многие компоненты и службы, доступные в Windows Azure и SQL Azure. Следующие четыре раздела этой главы кратко описывают основные службы и компоненты, доступные на момент написания. Они разделены на четыре категории: среда исполнения, управление данными, сетевые службы и прочие службы.

Комментарий Бхарата:

Платформа Windows Azure включает ряд служб, упрощающих разработку, повышающих надежность и облегчающих управление приложениями, размещенными в облаке.

Дополнительные сведения о службах и компонентах Windows Azure см. в разделе «Компоненты Windows Azure» на портале Windows Azure по адресу http://www.microsoft.com/windowsazure/features/. Конкретные рекомендации по каждому компоненту и службе см. в источниках, ссылки на которые приведены в следующих разделах.

Примечание.

Для использования любого из этих компонентов и служб необходима подписка на Windows Azure. Для регистрации учетной записи Windows Azure необходим действующий идентификатор Windows Live ID. Дополнительные сведения см. в разделе «Получение платформы Windows Azure» по адресуhttp://www.windowsazure.com/ru-ru/pricing/free-trial/.

Среда выполнения

Среда выполнения Windows Azure состоит из платформы для приложений и служб, размещенных в одной или нескольких ролях. Типы ролей, которые могут быть реализованы в Windows Azure:

Azure Compute (веб-роли и рабочие роли). Приложение Windows Azure состоит из одной или нескольких ролей, размещенных в центрах обработки данных Azure. Как правило, существует хотя бы одна веб-роль, предоставленная для доступа пользователям приложения. Приложение может содержать дополнительные роли, например рабочие, которые обычно используются для фоновой обработки и выполнения вспомогательных задач для веб-ролей. Дополнительные сведения см. в разделах «Обзор создания размещенной службы для Windows Azure» по адресу http://technet.microsoft.com/ru-ru/library/gg432976.aspx и «Создание приложения для работы в размещенной службе» по адресу http://technet.microsoft.com/ru-ru/library/hh180152.aspx.

Виртуальная машина (роль виртуальной машины). Эта роль позволяет разместить собственный пользовательский экземпляр операционной системы Windows Server 2008 R2 Enterprise или Windows Server 2008 R2 Standard в центре обработки данных Windows Azure. Дополнительные сведения см. в разделе «Создание приложений с помощью роли виртуальной машины в Windows Azure» по адресу http://technet.microsoft.com/ru-ru/library/gg465398.aspx.

Большинство примеров в этом руководстве и связанном с ним руководстве «Разработка приложений для облака» (см. http://wag.codeplex.com/), а также примеров в Hands-on-Labs используют веб-роль для выполнения необходимой обработки.

Использование рабочей роли также описано и показано во многих примерах и разделах руководства. Например, см. главу 6 этого руководства и соответствующий образец приложения, упражнение 4 в Hands-on-Labs для этого руководства, главу 5 руководства «Разработка приложений для облака» (см. http://wag.codeplex.com/) и соответствующий образец приложения, а также упражнение 3 в Hands-on-Labs для руководства «Разработка приложений для облака».

Управление данными

Windows Azure, SQL Azure и связанные службы позволяют хранить данные и управлять ими несколькими разными способами. Доступны следующие службы и функции управления данными:

Хранилище Azure. Включает четыре основные службы для постоянного и длительного хранения данных в облаке. Службы поддерживают интерфейс REST, доступный из приложений, размещенных в Azure, а также локальных приложений. Сведения о REST API см. в «Справочнике по интерфейсу REST API служб хранилища Windows Azure» по адресу http://msdn.microsoft.com/ru-ru/library/dd179355.aspx. Существует четыре службы хранения:

Служба таблиц Azure обеспечивает механизм хранения данных в табличном формате, то есть в виде привычных строк и столбцов, и поддерживает запросы на управление данными. Эта служба предназначена в основном для тех сценариев, где необходимо хранить большие объемы данных с простым доступом и обновлением. Дополнительные сведения см. в разделах «Концепции табличных служб» по адресу http://msdn.microsoft.com/ru-ru/library/dd179463.aspx и «API табличных служб» по адресу http://msdn.microsoft.com/ru-ru/library/dd179423.aspx.

Служба больших двоичных объектов (BLOB) реализует последовательность контейнеров для хранения текста или двоичных данных. Обеспечиваются как блочные контейнеры больших двоичных объектов для потоковой передачи данных, так и страничные контейнеры больших двоичных объектов для выполнения произвольного чтения и записи. Дополнительные сведения см. в разделах «Сведения о блочных и страничных больших двоичных объектах» по адресу http://msdn.microsoft.com/ru-ru/library/ee691964.aspx и «API службы больших двоичных объектов» по адресу http://msdn.microsoft.com/ru-ru/library/dd135733.aspx.

Служба очередей реализует механизм для постоянного и надежного обмена сообщениями между экземплярами ролей, например между веб-ролью и рабочей ролью. Дополнительные сведения см. в разделах «Концепции службы очередей» по адресу http://msdn.microsoft.com/ru-ru/library/dd179353.aspx и «API службы очередей» по адресу http://msdn.microsoft.com/ru-ru/library/dd179363.aspx.

Диски Windows Azure реализуют механизм, который позволяет приложению подключить однотомный виртуальный жесткий диск NTFS в виде страничного большого двоичного объекта, а также выгружать и загружать виртуальные жесткие диски через такой объект. Дополнительные сведения см. в разделе «Диск Windows Azure» (PDF-файл) по адресу http://go.microsoft.com/?linkid=9710117.

База данных SQL Azure. Это масштабируемая служба облачной базы данных с высокой степенью доступности, построенная на основе технологий SQL Server, поддерживает знакомую реляционную

модель баз данных, совместимую с T-SQL. Она может использоваться с приложениями, размещенными в Windows Azure, и другими приложениями, размещенными локально у пользователей или на других площадках. Дополнительные сведения см. в разделе «База данных SQL Azure» по адресу http://msdn.microsoft.com/ru-ru/library/ee336279.aspx.

Синхронизация данных. Синхронизация данных SQL Azure — это облачная служба синхронизации данных, построенная на основе технологий Microsoft Sync Framework. Она обеспечивает двунаправленную синхронизацию данных и управление данными, что дает возможность работать с несколькими базами данных SQL Azure, а также совместно использовать данные в базе данных пользователя и базе данных SQL Azure. Дополнительные сведения см. в «Центре разработки Microsoft Sync Framework» по адресу http://msdn.microsoft.com/ru-ru/sync.

Кэширование. Эта служба позволяет создавать высокопроизводительные приложения, использующие распределенный кэш данных в памяти с малой задержкой и высокой пропускной способностью. Эта служба не требует установки или управления и автоматически увеличивает и уменьшает размер кэша в динамическом режиме по мере необходимости. Она может использоваться для кэширования данных приложения и сведений о состоянии сеанса ASP.NET, а также для кэширования страничного вывода ASP.NET. Дополнительные сведения см. в разделе «Служба кэширования (Windows Azure AppFabric)» по адресу http://msdn.microsoft.com/ru-ru/library/gg278356.aspx.

Глава 5 данного руководства и упражнение 2 в Hands-on-Labs показывают, как пользоваться структурированным табличным хранилищем.

Глава 6 этого руководства, а также упражнения 3 и 4 в Hands-on-Labs показывают, как пользоваться хранилищем больших двоичных объектов и очередями.

В главах 3 и 5 руководства «Разработка приложений для облака» (см. http://wag.codeplex.com/) объясняются концепции и реализация многоклиентских архитектур для хранения данных.

В главе 5 руководства «Разработка приложений для облака» описано, как пользоваться хранилищем очередей.

В главе 6 руководства «Разработка приложений для облака и в упражнении 4 в Hands-on-Labs показано использование табличного хранилища (включая подкачку данных) и хранилища больших двоичных объектов в многоклиентских приложениях.

В главе 2 данного руководства, главе 6 руководства «Разработка приложений для облака и в упражнении 5 в Hands-on-Labs для руководства «Разработка приложений для облака» используется SQL Azure для хранения данных.

В главе 6 руководства «Разработка приложения для облака и в упражнении 4 в Hands-on-Labs для руководства «Разработка приложения для облака» показано использование службы кэширования.

Сетевые службы

Windows Azure включает несколько сетевых служб, которые обеспечивают максимальное повышение производительности, проверку подлинности и повышают управляемость размещенными приложениями. Это следующие службы:

Сеть доставки содержимого (CDN). Сеть CDN позволяет кэшировать общедоступные статические данные для приложений в специальных точках, расположенных ближе (с точки зрения доставки по сети) к конечным пользователям. Сеть CDN использует ряд центров обработки данных по всему миру, где данные хранятся в хранилище больших двоичных объектов с анонимным доступом. То есть не обязательно там, где фактически работают приложения. Дополнительные сведения см. в разделе «Доставка широкополосного содержимого по сети доставки содержимого (CDN) Windows Azure» по адресу http://msdn.microsoft.com/ru-ru/library/ee795176.aspx.

Служба Virtual Network Connect. Эта служба позволяет настроить роли приложения, работающего в Windows Azure, а также компьютеры в сети конечного пользователя таким образом, чтобы обеспечить видимость их нахождения в одной и той же сети. Эта служба с помощью программного агента, запускаемого на компьютере конечного пользователя, устанавливает соединение с ролями Windows Azure по протоколу IPsec. Поддерживается также возможность прямого администрирования, управления, мониторинга и отладки ролей. Дополнительные сведения см. в разделе «Подключение локальных компьютеров к ролям Windows Azure» по адресу http://msdn.microsoft.com/ru-ru/library/gg433122.aspx.

Диспетчер трафика виртуальной сети. Эта служба позволяет задать перенаправление запросов и балансировку нагрузки с помощью трех разных методов. Как правило, диспетчер трафика служит для повышения производительности путем перенаправления пользовательских запросов к экземпляру в ближайшем центре обработки данных в режиме Performance (Производительность). Альтернативные режимы — Failover (Отработка отказа) и Round Robin (Циклический перебор). Дополнительные сведения см. в разделе «Диспетчер трафика Windows Azure» по адресу http://msdn.microsoft.com/ru-ru/WAZPlatformTrainingCourse_WindowsAzureTrafficManager.

Управление доступом. Эта служба удостоверений и управления доступом построена на основе стандартов и использует целый ряд поставщиков удостоверений (IdPs) для проверки подлинности пользователей. Служба управления доступом действует как служба маркеров безопасности (STS), помогая использовать преимущества методов федеративной проверки подлинности, когда подлинность пользователя проверяется не в той области или домене, где находится приложение. В качестве примера можно привести управление доступом пользователей, когда удостоверение пользователя проверяется поставщиком удостоверений (Windows Live ID, Google и т. п.). Дополнительные сведения см. в разделе «Access Control Service 2.0» по адресу http://msdn.microsoft.com/ru-ru/library/gg429786.aspx и «Руководстве по проверке подлинности на основе утверждений и управлению доступом» в http://claimsid.codeplex.com/.

Шинаи нтеграции. Обеспечивает надежный обмен сообщениями и возможность потоковой передачи данных для распределенных и гибридных приложений (например, для обмена данными между приложениями, размещенными в Windows Azure, и локальными приложениями и службами конечных пользователей) без брандмауэров и инфраструктуры безопасности. Эта шина может использовать ряд протоколов связи и обмена сообщениями, а также шаблонов, обеспечивающих надежную доставку и обмен сообщениями. Она масштабируется в зависимости от нагрузки и может интегрироваться с артефактами собственного сервера BizTalk Server. Дополнительные сведения см. в разделе «Шина обслуживания AppFabric» по адресу http://msdn.microsoft.com/ru-ru/library/ee732537.aspx.

В главе 4 руководства «Разработка приложений для облака» (см. http://wag.codeplex.com/) и примере приложения показано использование сети доставки содержимого (CDN).

Подробные инструкции по использованию службы управления доступом см. в соответствующем «Руководстве по проверке подлинности на основе утверждений и управлению доступом» (см. http://claimsid.codeplex.com/) и упражнении 3 Hands-on-Labs для этого руководства.

Другие службы

Windows Azure включает следующие дополнительные службы:

Отчетность по бизнес-аналитике. Позволяет создавать оперативные рабочие отчеты на основе данных, которые хранятся в базе данных SQL Azure, и развертывать их в облаке. Служба построена на основе тех же технологий, что и службы SQL Server Server Reporting Services, и позволяет пользоваться знакомыми средствами создания отчетов. Доступ к отчетам осуществляется через портал управления Windows Azure, через веб-браузер, а также прямо из ваших приложений Windows Azure и собственных приложений конечных пользователей. Дополнительные сведения см. в разделе «Отчетность SQL Azure» по адресу http://msdn.microsoft.com/ru-ru/library/gg430130.aspx.

Приложение Marketplace. Это портал, где разработчики могут искать, передавать, продавать и покупать компоненты систем, учебные материалы, шаблоны служб, наборы данных, а также готовые услуги и приложения, необходимые для создания приложений на платформе Windows Azure. Дополнительные сведения см. в разделах «Windows Azure Marketplace DataMarket» по адресу http://msdn.microsoft.com/ru-ru/library/gg315539.aspx и «Windows Azure Marketplace» (AppMarket) по адресу http://windowsazure.pinpoint.microsoft.com/ru-ru/Default.aspx.

Разработка приложений для Windows Azure

В этом разделе описаны средства и ресурсы разработчика, предназначенные для создания приложений Windows Azure и SQL Azure.

Обычно на платформе Windows используется Visual Studio 2010 со средствами Windows Azure для Microsoft Visual Studio. Средства Windows Azure включают все возможности, необходимые для

создания приложений Windows Azure, включая локальные эмуляторы вычислений и хранилища данных, запускаемые на компьютере разработчика. Это означает, что разработчик может на своем компьютере создавать, тестировать и отлаживать приложения, а затем развертывать их в облаке. Кроме того, сюда входят средства развертывания приложений в Windows Azure и управления этими приложениями после развертывания.

Комментарий Маркуса:

Разработчик может создавать и тестировать приложения Windows Azure на своем компьютере с помощью эмуляторов вычислений и хранилища данных.

Средства Windows Azure для Microsoft Visual Studio и средства разработки для других платформ и языков (iOS, Eclipse, Java и PHP) можно загрузить на странице «Средства Windows Azure» по адресу http://www.microsoft.com/windowsazure/tools/.

Большой выбор полезных видеоматериалов, примеров краткого руководства и упражнений Hands-On-Lab по множеству тем, которые помогут научиться создавать приложения Windows Azure, см. в разделах «Знакомство с Windows Azure и SQL Azure» по адресу http://www.microsoft.com/windowsazure/tutorials/ и «Проектирование, кодирование, масштабирование» по адресу http://www.microsoft.com/windowsazure/getstarted/.

Раздел MSDN «Разработка приложений для Windows Azure» по адресу http://msdn.microsoft.com/ru-ru/library/gg433098.aspx содержит конкретные примеры и рекомендации для создания размещенных служб, где для пакетирования и развертывания приложений используется набор средств Windows Azure SDK Tools, а также полезный пример для быстрого начала работы.

Учебный комплект Windows Azure включает практические упражнения, которые помогут быстрого начать работу. Его можно загрузить на странице http://www.microsoft.com/downloads/details.aspx?FamilyID=413E88F8-5966-4A83-B309-53B7B77EDF78&displaylang=ru.

Разобраться с работой ролей Windows Azure и жизненным циклом выполнения можно в разделе «Реальный мир: жизненный цикл роли Windows Azure» по адресу http://msdn.microsoft.com/ru-ru/library/hh127476.aspx.

Список ресурсов, полезных для разработки и развертывания баз данных в SQL Azure, см. в разделе «Разработка (база данных SQL Azure)» по адресу http://msdn.microsoft.com/ru-ru/library/ee336225.aspx.

Перечень средств, которые помогут запланировать перенос приложения в Windows Azure, см. в разделе «Оценка и планирование» статьи «Средства платформы Windows Azure» по адресу http://www.microsoft.com/windowsazure/tools/#assessment.

Обновление приложений Windows Azure

После развертывания приложения в Windows Azure необходимо его обновление в случае изменения служб ролей в соответствии с новыми требованиями, а также по мере совершенствования кода и исправления ошибок. Можно просто развернуть службу заново. Для

этого необходимо приостановить и удалить службу, а затем развернуть ее новую версию. При этом простоя приложений можно избежать, если использовать поэтапное развертывание (когда новый пакет загружается и замещает существующую производственную версию) или обновление на месте (когда новый пакет загружается и применяется к работающим экземплярам службы).

О том, как обновлять службы путем загрузки нового пакета и замены существующей производственной версии этим пакетом, см. в разделе «Обновление службы методом замены (VIP)» по адресу http://msdn.microsoft.com/ru-ru/library/ee517253.aspx.

О том, как обновлять службы на месте, в том числе о развертывании служб в обновляемом домене или при обработке отказа, а также о влиянии такого развертывания на варианты обновления, см. в разделе «Выполнение обновления на месте» по адресу http://msdn.microsoft.com/ru-ru/library/ee517255.aspx.

Примечание.
Если необходимо только изменить данные конфигурации для службы без развертывания нового кода, можно использовать веб-портал, или с помощью API управления изменить файл конфигурации службы, или загрузить новый файл конфигурации.

Управление, мониторинг и отладка приложений Windows Azure

В этом разделе описаны средства и ресурсы управления, которые удобно использовать для развертывания, мониторинга и отладки приложений, а также для управления приложениями в Windows Azure и SQL Azure.

Все подсистемы хранения данных и управления в Windows Azure используют интерфейсы REST. Они не зависят от какой-либо технологии .NET Framework или операционной системы Microsoft Windows ®. Любая технология, способная создавать запросы HTTP или HTTPS, может обращаться к средствам Windows Azure.

Сведения о собственных и управляемых Windows Azure библиотечных API, а также REST API служб хранения данных см. в разделе «Справочник по API для Windows Azure» по адресу http://msdn.microsoft.com/ru-ru/library/ff800682.aspx.

API управления службами на основе REST может использоваться в качестве альтернативы веб-порталу управления Windows Azure. Этот API включает функции для работы с учетными записями хранилища, размещенными службами, сертификатами, группами привязки, местами расположения и информацией о подписках. Дополнительные сведения см. в «Справочнике по REST API управления службами Windows Azure» по адресу http://msdn.microsoft.com/ru-ru/library/ee460799.aspx.

Помимо этих компонентов, платформа Windows Azure реализует службы диагностики для таких задач, как отслеживание за работоспособностью приложения. Пакет управления Windows Azure и Operations Manager 2007 R2 могут быть использованы для обнаружения приложений Windows Azure, получения данных о состоянии каждого экземпляра роли, сбора и отслеживания информации о производительности, сбора и отслеживания событий Windows Azure, а также для

сбора и отслеживания сообщений трассировки .NET Framework от каждого экземпляра роли. Дополнительные сведения см. в разделе «Мониторинг приложений Windows Azure» по адресу http://msdn.microsoft.com/ru-ru/library/gg676009.aspx.

Комментарий Маркуса:

Windows Azure включает компоненты для мониторинга и отладки служб, размещенных в облаке.

Сведения об использовании встроенных объектов трассировки Windows Azure для настройки диагностики и средств без использования Operations Manager, а также о загрузке результатов см. в разделе «Сбор журнальных данных с помощью средств диагностики Windows Azure» по адресу http://msdn.microsoft.com/ru-ru/library/gg433048.aspx.

Сведения об отладке приложений Windows Azure см. в разделах «Устранение неполадок и отладка в Windows Azure» по адресу http://msdn.microsoft.com/ru-ru/library/gg465380.aspx и «Отладка приложений в Windows Azure» по адресу http://msdn.microsoft.com/ru-ru/windowsazure/WAZPlatformTrainingCourse_WindowsAzureDebuggingLab.

Примечание.

Глава 7, «Управление жизненным циклом приложений для приложений Windows Azure», данного руководства содержит сведения по управлению приложениями Windows Azure.

Управление базами данных SQL Azure

Приложения производят доступ к базам данных SQL Azure точно так же, как и к локальным серверам SQL Server — через классы доступа к данным ADO.NET и встроенные технологии доступа к данным — ODBC, PHP или JDBC.

Управление базами данных SQL Azure осуществляется через веб-портал, среду SQL Server Management Studio, с помощью средств базы данных Visual Studio 2010 и целого ряда других средств для выполнения таких операций, как перемещение и перенос данных, а также с помощью средств командной строки для развертывания и администрирования.

Диспетчер базы данных также упростит работу с базами данных SQL Azure. Дополнительные сведения см., в разделе «Диспетчер баз данных для SQL Azure» по адресу http://msdn.microsoft.com/ru-ru/library/gg442309.aspx. Список других средств содержится в разделе «Средства платформы Windows Azure» по адресу http://www.microsoft.com/windowsazure/tools/#sqlazure.

SQL Azure поддерживает API управления, а также управление через веб-портал. Сведения об API управления SQL Azure см. в разделе «Справочник по REST API (SQL Azure)» по адресу http://msdn.microsoft.com/ru-ru/library/gg715283.aspx.

Подписка на Windows Azure и модель выставления счетов

В этом разделе описана модель выставления счетов за подписку на Windows Azure и SQL Azure и использование этих служб. Для использования Windows Azure необходимо сначала создать

учетную запись выставления счетов, подписавшись на службу Microsoft Online Services на странице https://mocp.microsoftonline.com/ или через портал Windows Azure по адресу https://windows.azure.com/. Клиентский портал Microsoft Online Services управляет подписками на все службы Microsoft. Windows Azure является одной из таких служб. Есть и другие службы, например, платформа Office 365, Windows Office Live Meeting и Windows Intune.

Каждая учетная запись выставления счетов имеет одного владельца учетной записи, определяемого по идентификатору Windows Live ®. Владелец учетной записи может создавать подписки и управлять ими, просматривать данные для выставления счета и данные об использовании, а также указывать администратора службы для каждой подписки. Подписка Windows Azure является лишь одной из таких подписок.

Комментарий По:

У владельца учетной записи и администратора службы для подписки могут быть (и во многих случаях должны быть) разные Live ID.

Администраторы управляют отдельными размещенными службами для подписки Windows Azure с помощью портала Windows Azure на странице https://windows.azure.com/. В состав подписки Windows Azure может входить следующее.

- Размещенные службы, состоящие из ролей и экземпляров в каждой роли.
  Роли и экземпляры могут быть остановлены в производственном (production) режиме или в тестовом (staging) режиме.
- Учетные записи хранилищ, состоящие из экземпляров хранилищ таблиц, больших двоичных объектов и очередей.
- Экземпляры сети доставки содержимого.
- Базы данных SQL Azure.
- Экземпляры служб SQL Azure Reporting Services.
- Экземпляры служб управления доступом, шины интеграции и кэша.
- Экземпляры подключения виртуальной сети и диспетчера трафика.

На рисунке 1 показана конфигурация тарификации Windows Azure для стандартной подписки.

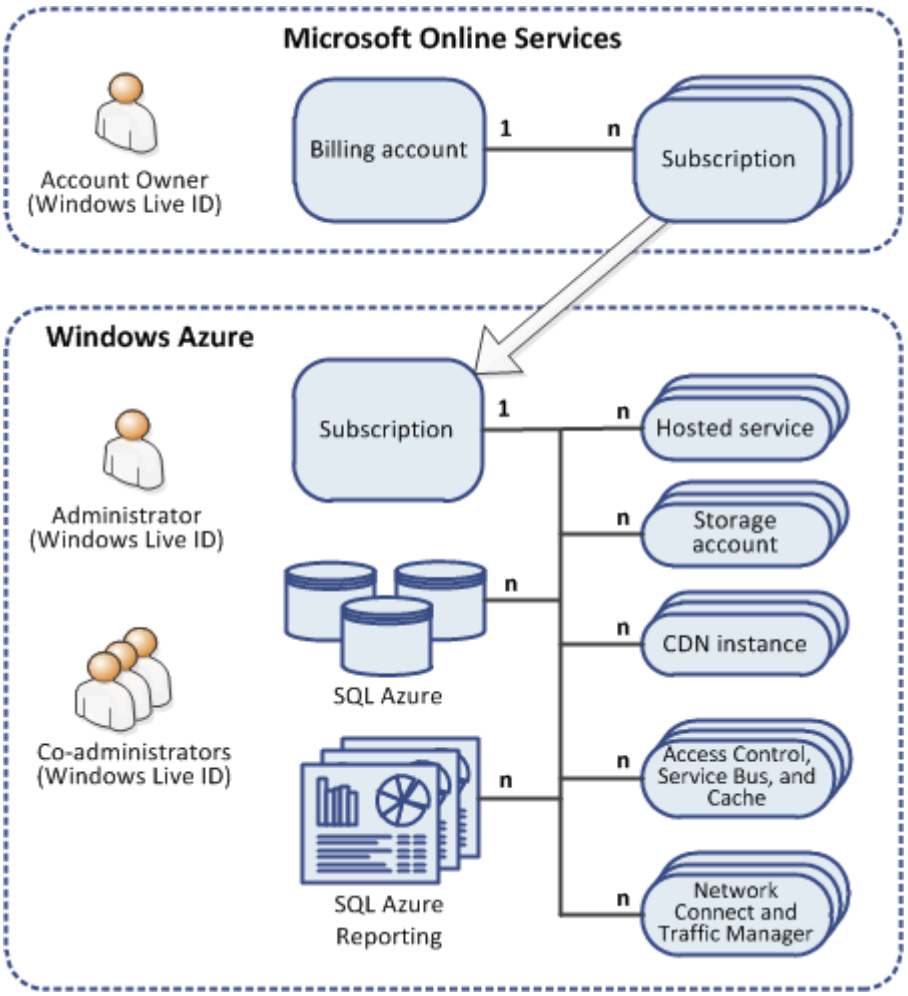


Рис. 1

Конфигурация тарификации Windows Azure для стандартной подписки

Дополнительные сведения о тарификации Windows Azure см. в разделах «Основы тарификации Windows Azure» по адресу http://msdn.microsoft.com/ru-ru/library/ff959173.aspx и «Учетные записи и тарификация в SQL Azure» по адресу http://msdn.microsoft.com/ru-ru/library/ee621788.aspx.

Оценка затрат

Оплата в Windows Azure взимается за использование таких служб, как вычислительные ресурсы времени, хранилище, трафик и т.п.. Оплата за вычисленные ресурсы рассчитывается на почасовой и зависит от выбранного размера экземпляра роли. Оплата хранилища определяется размеров хранимых и количеством транзакций. Цены на трафик различаются в зависимости от региона, тарифицируется только исходящий трафик из дата-центров, т.е. никакой трафик внутри дата-центров не тарифицируется.

Для оценки вероятных затрат на подписки Windows Azure можно использовать следующие ресурсы.

Обзор подписки для различных моделей закупок, например для модели повременной оплаты и модели подписки, включая инструмент для измерения потребления, по адресу http://www.microsoft.com/windowsazure/pricing/.

Калькулятор цен по адресу http://www.microsoft.com/windowsazure/pricing-calculator/.

Калькулятор совокупной стоимости владения по адресу http://www.microsoft.com/windowsazure/offers/#tcoCompare-LB.

Комментарий По:

Оплата вычислительных ресурсов производится не только за запущенные экземпляры, но и за остановленные экземпляры, т.е. за любые развернутые экземпляры. . Если вы не хотите оплачивать ту или иную службу, удалите связанные с ней развертывания.

Примечание.

В главе 4 данного руководства приведены дополнительные сведения об оценке стоимости размещения приложений в Windows Azure.

Дополнительные сведения

О платформе Windows Azure имеется много сведений в документации, учебных видеофильмах и технических документах. Ниже приведены некоторые веб-сайты, где можно найти дополнительные сведения.

Веб-сайт для этой серии руководств по адресу http://wag.codeplex.com/ содержит ссылки на интерактивные ресурсы, образцы кода, пособия Hands on-Labs, отзывы и многое другое.

Портал сведений о Microsoft Windows Azure находится по адресу http://www.microsoft.com/windowsazure/. Там можно найти ссылки на технические документы и такие инструменты, как пакет Windows Azure SDK, а также многие другие ресурсы. Кроме того, здесь можно подписаться на учетную запись Windows Azure.

Райан Данн (Ryan Dunn) и Стив Маркс (Steve Marx) записали на канале 9 ряд передач с обсуждением Azure. Эти материалы можно найти в Cloud Cover по адресу http://channel9.msdn.com/shows/Cloud+Cover/.

Ответы на вопросы можно найти на форуме Windows Azure по адресу http://social.msdn.microsoft.com/Forums/ru-ru/windowsazure/threads.

Стив Маркс — технический специалист по вопросам стратегии Windows Azure. Его блог находится по адресу http://blog.smarx.com/. Это интересный источник новостей и информации о Windows Azure.

Райан Данн — технический специалист по Windows Azure. Его блог находится по адресу http://dunnry.com/blog.

Юджении Пейс (Eugenio Pace), руководитель программы в группе архитектуры и рекомендаций корпорации Microsoft, создает серию руководств по Windows Azure, в которую входит эта документация. Для получения дополнительных сведений об этой серии просмотрите его блог по адресу http://blogs.msdn.com/eugeniop.

Скотт Денсмор (Scott Densmore), ведущий разработчик группы архитектуры и рекомендаций корпорации Microsoft, пишет о разработке приложений для Windows Azure в своем блоге адресу http://scottdensmore.typepad.com/.

Джим Накашима (Jim Nakashima) — руководитель программы в группе, которая создает инструменты для Windows Azure. Его блог содержит множество технических подробностей и советов. Адрес блога — http://blogs.msdn.com/jnak/.

Код и документация для проекта руководства по архитектуре и рекомендациям для Windows Azure доступен на сайте руководства CodePlex Windows Azure http://wag.codeplex.com/.

Полные руководства и примеры службы управления доступом к Windows Azure имеются в книге по архитектуре и рекомендациям «Руководство по управлению доступом и удостоверениями на основе утверждений», доступной также в Интернете по адресу http://claimsid.codeplex.com/.

Глава 2. Сценарий Adatum

В этой главе представляется вымышленная компания Adatum. В главе описывается текущая инфраструктура компании Adatum, ее портфель программного обеспечения и причины, побудившие Adatum перевести некоторые приложения на платформу Windows ® Azure ™. Как и в любой компании, рассматривающей процесс миграции, существует множество вопросов и проблем, требующих решения, особенно потому, что компания Adatum ранее не использовала облачную технологию. В следующих главах шаг за шагом показано, как в Adatum изменяют свою систему отслеживания и возмещения затрат aExpense, адаптируя ее для развертывания в Windows Azure.

Компания Adatum

Adatum — производственная компания со штатом в 5 000 сотрудников, использующая главным образом технологии и инструменты Microsoft ®. В компании также есть устаревшие системы на базе других платформ, например AS400 и UNIX. Разработчики Adatum, естественно, хорошо разбираются в различных продуктах Microsoft, в том числе в .NET Framework, ASP.NET, программном обеспечении баз данных SQL Server ®, операционной системе Windows Server ® и системах разработки Microsoft Visual Studio ®. Сотрудники в ИТ-отделе компании Adatum квалифицированы в таких областях, как создание и поддержание службы каталогов Microsoft Active Directory ® и использовании пакета управления System Center.

В компании Adatum используется множество различных приложений. С некоторыми взаимодействуют внешние пользователи, другие используются исключительно сотрудниками. Важность этих приложений — от «незначительной» до «критической» со множеством промежуточных градаций. Значительная часть бюджета компании Adatum выделяется для обслуживания приложений средней или незначительной важности.

В Adatum планируют скорректировать распределение бюджета. Цель компании — тратить больше денег на службы, которые будут выделять ее среди конкурентов, и меньше — на остальные службы. Конкурентные преимущества компании Adatum обеспечиваются такими факторами, как эффективная цепочка поставок и отличный контроль качества, а не тем, насколько эффективно обрабатывается внутренняя электронная почта. Компании Adatum нужна эффективная электронная почта, но ей требуется более экономичный способ наладить ее, чтобы можно было потратить большую часть бюджета на системы, имеющие непосредственное отношение к клиентам. В компании Adatum считают, что один из способов такой оптимизации — выборочное развертывание приложений в облаке.

Задачи Adatum

Перед Adatum стоит ряд задач. В настоящее время локальное развертывание новых приложений требует слишком много времени, если учесть изменчивость бизнеса и эффективность конкурентов. Временные рамки для приобретения, подготовки и развертывания даже простого приложения составляют по меньшей мере несколько недель. Независимо от сложности приложения требуется анализ требований, инициирование процесса закупок, запрос предложений у поставщиков, настройка сетей и т. д. Компания Adatum должна реагировать на запросы клиентов быстрее, чем позволяют нынешние процедуры.

Еще одна задача заключается в решении проблемы неэффективного использования большей части инфраструктуры Adatum. Большая часть серверов компании используется недостаточно полно, в результате чего бывает трудно развернуть новое приложение в соответствие с принятым соглашением об уровне обслуживания (SLA) на имеющемся оборудовании. В некоторых случаях выручают виртуальные машины, но они подходят не всегда. Эта неэффективность означает, что капитал Adatum расходуется на недостаточно используемую инфраструктуру, тогда как эффективнее было бы направить его на другие ресурсы.

Последняя проблема заключается в том, что менее важным приложениям ИТ-персонал обычно уделяет меньше внимания. На них обращают внимание только тогда, когда они отказывают или перестают удовлетворять требованиям. Но к этому моменту исправление проблемы обходится дорого как в отношении временных затрат ИТ-службы, так и в отношении неэффективности использования времени пользователей.

В Adatum уверены, что, развернув некоторые приложения в общедоступном облаке, например, на платформе Windows Azure, компания сможет добиться эффекта масштаба, повысить стандартизацию приложений и разработать автоматизированные процессы управления ими. Самое главное, в компании Adatum полагают, что это повысит эффективность удовлетворения потребностей клиентов, а следовательно, повысит конкурентоспособность и инвестиционную привлекательность для акционеров.

Цели и задачи компании Adatum

Одна из целей компании Adatum — повышение удовлетворенности пользователей ее приложений. Приложения в облаке должны функционировать как минимум не хуже локальных. При этом предполагается, что они будут функционировать лучше. Порой некоторые приложения используются чаще других. Например, сотрудники используют средство подсчета заработной платы раз в две недели, но редко обращаются к нему в другое время. Было бы эффективно повысить оперативность работы приложений в периоды пиковой нагрузки. Способность реагировать на запросы известна как динамическая масштабируемость. Размещенные локально и связанные с конкретными серверами приложения не обладают такой гибкостью. Компания Adatum не может позволить себе запустить столько серверов, сколько необходимо в пиковое время, потому что остальную часть времени это оборудование не будет использоваться. Если же эти приложения разместить в облаке, то будет легко масштабировать их по запросу.

Еще одна цель компании — увеличение количества вариантов доступа к ее приложениям. В настоящее время приложения доступны только из внутренней сети. Их публикация в Интернете трудна и требует повышения уровня безопасности. Для этого также требуется виртуальная частная сеть (VPN), которую пользователи зачастую не хотят использовать из-за связанных с этим дополнительных сложностей. Приложения, расположенные в публичном облаке, по определению доступны из Интернета. Однако публичное облако также не решает вопросов безопасности. Кроме того, многие приложения Adatum используют авторизацию Windows, следовательно, пользователям не требуется вводить учетные данные, связанные с конкретным приложением. Компания Adatum обеспокоена тем, что ее пользователям потребуются дополнительные учетные данные для каждого приложения в публичном облаке.

Третья цель состоит в том, что, по крайней мере, некоторые из приложений Adatum должны быть переносимыми. Переносимость означает, что приложение можно перемещать между внешним и локальным центрами данных без каких-либо изменений в коде или возможностях приложения. Доступность обоих вариантов снижает риски компании Adatum в случае использования облака.

Помимо вопросов безопасности у Adatum имеется еще две проблемы. Во-первых, компания хотела бы избежать массовой переподготовки сотрудников ИТ-отдела. Во-вторых, очень немногие из приложений Adatum полностью изолированы от других систем. Большинство имеют различные зависимости. Компания Adatum приложила много усилий к интеграции своих систем, несмотря на то что не все они работают на одной платформе. Неизвестно, как эти зависимости повлияют на работу при перемещении некоторых систем в общедоступное облако.

Стратегия компании Adatum

Adatum — инновационная компания, открытая для новых технологий, но при их реализации тщательно взвешиваются все факторы. План компании Adatum состоит в оценке целесообразности перехода к технологии облака, начиная с перевода нескольких простых приложений. В компании надеются получить некоторый начальный опыт, а затем использовать его в своих разработках. Эта стратегия может быть описана как «пробовать, учиться, ошибиться вначале, а затем оптимизировать». В Adatum решили начать с приложения aExpense.

Приложение aExpense

Приложение aExpense позволяет сотрудникам Adatum представлять, отслеживать и обрабатывать деловые расходы. Все в Adatum используют это приложение для запроса расходов. Хотя aExpense не относится к критическим приложениям, оно важно для работы. Сотрудники могут споконой воспринять несколько часов простоя, но длительная недоступность приложения неприемлема.

Согласно политике Adatum сотрудники должны подтверждать расходы до окончания каждого месяца. Большинство сотрудников подтверждают свои расходы в последние два рабочих дня. Это приводит к сравнительно высокой нагрузке в течение небольшого периода времени. Инфраструктура, которая поддерживает приложение aExpense, рассчитана на среднее использование в течение месяца, а не на пиковую нагрузку. В результате, когда в течение последних двух рабочих дней большинство сотрудников пытаются ввести свои расходы, работа системы замедляется и сотрудники жалуются.

Приложение развернуто в центре обработки данных Adatum и доступно для пользователей внутренней сети. В командировках сотрудники получают к нему доступ через VPN. Поступали запросы на публикацию aExpense непосредственно в Интернете, но это так и не было реализовано.

Приложение хранит большой объем информации, потому что большинство документов по расходам должно сканироваться и затем храниться в течение семи лет. По этой причине часто выполняется резервное копирование хранилищ данных aExpense.

Приложение является типичным для компании Adatum, так что хорошо подходит для проверки работы в облаке. Перенос приложения aExpense на платформу Windows Azure обнаруживает много проблем, с которыми компания Adatum столкнется, когда решит перевести в облако больше приложений.

Архитектура aExpense

Архитектура aExpense представлена на рис. 1.

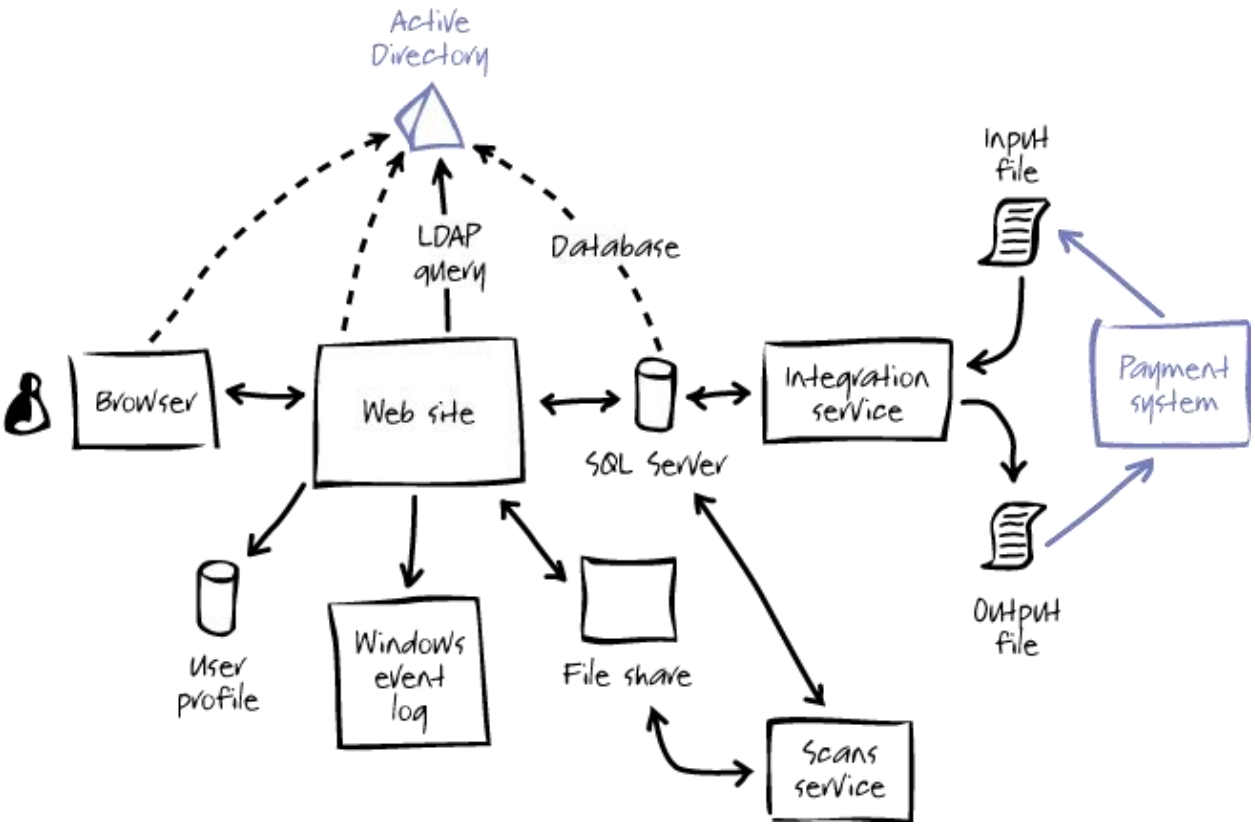


Рис. 1

Архитектура довольно проста и используется многими другими приложениями. aExpense представляет собой приложение ASP.NET; для взаимодействия с ним сотрудники используют браузер. Для обеспечения безопасности приложение использует проверку подлинности Windows. Для сохранения настроек пользователя используется членство ASP.NET и поставщики профилей. Обработка исключений и журанлирование организованы с помощью блока обработки исключений (Exception Handling Application Block) и блока журналирования (Logging Application Block) библиотеки Enterprise Library. Веб-сайт так же запрашивает дополнительные данные о сотрудниках из Active Directory (например, информацию о руководителе сотрудника, который имеет полномочия для подтверждения расходов), для этого используются API-интерфейсы служб каталогов.

Для подключения к SQL Server в приложении aExpense реализована доверенная подсистема (trusted sybsystem). Проверка подлинности производится с помощью учетной записи пользователя домена Windows. База данных поддерживает проверку подлинности SQL Server. Приложение aExpense хранит свои данные на сервере SQL Server. Отсканированные изображения чеков хранятся в общей папке.

Есть две фоновые службы, обе реализованы как службы Windows. Одна запускается периодически и создает эскизы (уменьшенные копии изображений) сканированных чеков. Служба сжимает

большие изображения для повышения эффективности хранения. Другая фоновая служба периодически запрашивает базу данных о расходах, которые должны быть возмещены. Затем она создает неструктурированный файл, который может быть обработан платежной системой. Эта служба также импортирует результаты оплаты и отправляет их обратно в aExpense после того, как выплаты будут произведены.

Глава 3. Переход к облаку (этап 1)

В этой главе описаны первые шаги по переносу приложения на технологическую платформу Windows® Azure™. Вы увидите пример того, как можно перенести в облако существующее бизнес-приложение, разработанное с помощью ASP.NET. Первый этап посвящен только обеспечению работы приложения в облаке без потери каких-либо возможностей. Уделяется внимание некоторым «большим» проблемам, например, безопасности и хранению данных, что важно практически для любого приложения на базе облака.

На первом этапе не рассматриваются способы улучшения работы приложения с помощью функций, доступных на платформе Windows Azure. Кроме того, локальная версия приложения, которую вы увидите, не завершена. Она содержит лишь базовые функциональные возможности для начала работы. В следующих главах мы обсудим, как улучшить работу приложения, используя некоторые из возможностей, доступных на платформе Windows Azure, и покажем другие функции, добавленные в приложение. Сейчас вы просто увидите, как перенести приложение в облако.

Исходные условия

Существующее приложение aExpense представляет собой систему подтверждения и возмещения деловых расходов, используемую сотрудниками компании Adatum. Приложение создано на ASP.NET 3.5, размещено в центре обработки данных компании Adatum и доступно из внутренней сети Adatum. Для проверки подлинности сотрудников приложение использует Microsoft Active Directory®. Кроме этого, через Active Directory производится доступ к необходимым приложению данным из профилей пользователей — например, руководителя и центра расходов служащего. Поскольку приложение aExpense использует проверку подлинности Windows, само не запрашивает имя пользователя и пароль.

Комментарий По:

Интеграция с Active Directory упрощает задачу управления приложением. Приложение aExpense использует средства управления доступом Active Directory и сведения о центрах расходов и руководителях, которые хранятся в Active Directory.

Правила управления доступом aExpense используют роли приложения, например «Сотрудник» и «Руководитель». Управление доступом тесно связано с бизнес-логикой приложения.

Для хранения своих данных приложение aExpense использует простую базу данных SQL Server®, а в качестве механизма доступа к данным используется LINQ to SQL. Приложение настроено для соединения с SQL Server через встроенную систему безопасности, а для входа в базу данных веб-сайт использует учетную запись службы.

Для журналирования диагностической информации aExpense использует блок обработки исключений и блок журналирования библиотеки Enterprise Library (Exception Handling Application Block и Logging Application Block).

На рис. 1 приведена диаграмма структуры локального приложения aExpense.

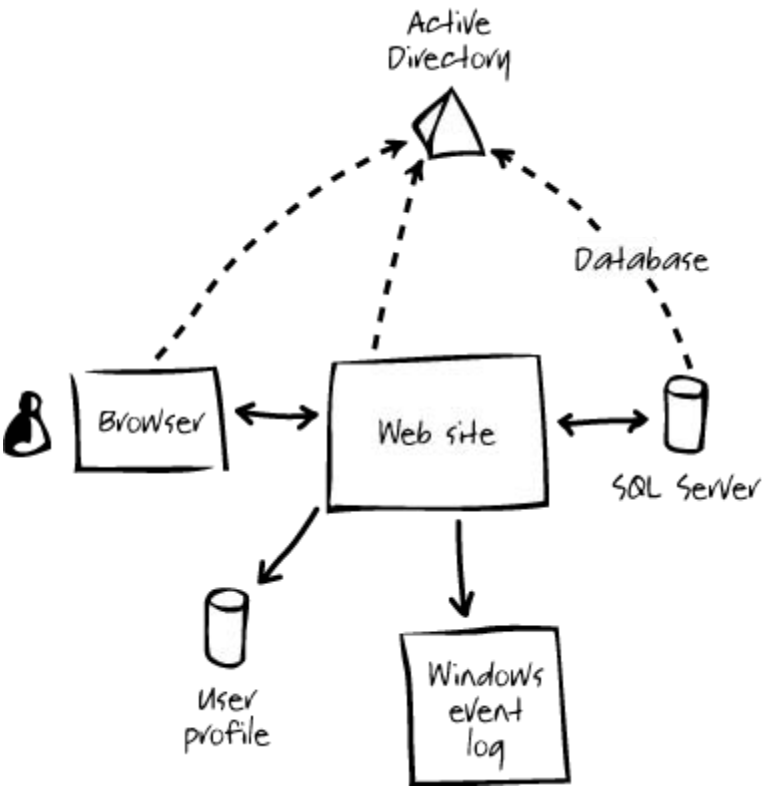


Рис. 1

aExpense как локальное приложение

Цели и требования

На первом этапе у компании Adatum имеется ряд целей по проведению миграции приложения aExpense в облако, их можно кратко сформулировать как «Обеспечение работы в облаке». Оптимизация приложения для облака и использование функций Windows Azure — уже следующий этап.

Ваше решение перенести приложение в облако должно быть основано на четких целях и потребностях.

Компания Adatum определила ряд конкретных целей для первого этапа. Приложение aExpense в облаке должно иметь доступ к тем же данным, что и его локальная версия. Сюда входят данные о деловых расходах, обрабатываемые приложением, и данные профилей пользователей, такие как центры расходов и руководители пользователей, которые необходимы для выполнения бизнес-

правил в приложении. Однако в Adatum решили убрать из aExpense какую-либо зависимость от Active Directory и исключить обратные обращения приложения из облака в Adatum.

Комментарий Бхарата:

Мы стремимся избежать обращений приложения из облака в Adatum, т.к.. это значительно повышает сложность решения.

Вторая цель состояла в том, чтобы гарантировать операционистам в версии aExpense для облака доступ к той же диагностической информации, как и в локальной версии.

Важным вопросом для Adatum является безопасность версии для облака, поэтому третья цель состоит в управлении доступом к приложению aExpense на основе учетных данных, управляемых из Adatum, и предоставлении пользователям возможности получать доступ к приложению по имеющимся у них учетным данным. В Adatum стремятся избежать накладных расходов на управление дополнительными системами безопасности для приложений на базе облака.

Итак, цели данного этапа заключаются в переносе aExpense в облако и сохранении процедур работы пользователей и управляемости приложения при минимальных изменениях существующего приложения.

Обзор решения

Первым шагом стал анализ существующего приложения, для определения тех частей, которые необходимо изменить при переносе в облако. Помните что целью на данном этапе ставилась возможность работы приложения в облаке с минимальными изменениями в нем.

На этом этапе Adatum стремится свести к минимуму изменения, вносимые в приложение.

Команда проекта миграции определила, что для удовлетворения требований приложения к хранению данных вместо SQL Server можно использовать SQL Azure. Имеющуюся структуру базы данных и ее содержимое удалось легко перенести в SQL Azure.

Примечание.

Кроме того, можно использовать мастер миграции SQL Azure на странице http://sqlazuremw.codeplex.com/, который поможет перенести локальные базы данных SQL Server в SQL Azure.

Команда также выяснила, что приложение может продолжать использовать блоки Enterprise Library в Windows Azure и что облачное приложение при этом может создавать ту же диагностическую информацию, что и локальное.

Комментарий Маркуса:

Будет замечательно, если мы сможем использовать опробованный и испытанный код и в облачной версии приложения.

Примечание.
Загрузить технические документы с описанием возможностей и ограничений Enterprise Library 5.0 при работе с приложениями .NET, предназначенными для использования на платформе Windows Azure, можно на странице http://wag.codeplex.com/.

Локальное приложение aExpense хранит предпочтительные настройки в модуле профилей ASP.NET. Поставщик профилей ASP.NET по умолчанию использует в качестве механизма хранения SQL Server. Поскольку SQL Azure — относительно дорогой механизм хранения (по сравнению с табличным хранилищем Windows Azure), а данные профиля являются весьма простыми, разработчики решили использовать реализацию поставщика профилей для работы с табличным хранилищем Windows Azure. Смена поставщика профилей никак не должна повлиять на текущий код приложения.

Наиболее важные изменения в приложении, необходимость которых выявили в компании, связаны с проверкой подлинности и авторизацией. Команда Adatum решила доработать приложение, чтобы использовать в нем систему на основе утверждений. В Adatum настроят локальный поставщик утверждений служб федерации Active Directory (ADFS) в своем центре обработки данных. Когда пользователь пытается получить доступ к приложению aExpense в облаке, его перенаправляют к поставщику утверждений (ADFS). Если пользователь еще не вошел в домен Adatum, ему нужно будет ввести свои учетные данные Windows, после чего поставщик утверждений сформирует маркер, содержащий набор утверждений, полученных из Active Directory. Эти утверждения будут включать информацию о членстве пользователя в ролях, центре расходов и руководителе. Это сведет к минимуму изменения в приложении и удалит прямую зависимость текущей версии от Active Directory, поскольку приложение будет получать необходимые пользовательские данные от поставщика утверждений (поставщик утверждений по-прежнему будет получать данные от Active Directory от имени приложения aExpense). Внешний поставщик утверждений можно интегрировать с Active Directory, поэтому процесс единого входа для пользователей приложения не изменится.

Комментарий Яны:

Использование утверждений может упростить приложение за счет делегирования части функций поставщику утверждений.

На рис. 2 приведена схема, на которой разработчики показали, как выглядела бы архитектура aExpense после миграции на Windows Azure.

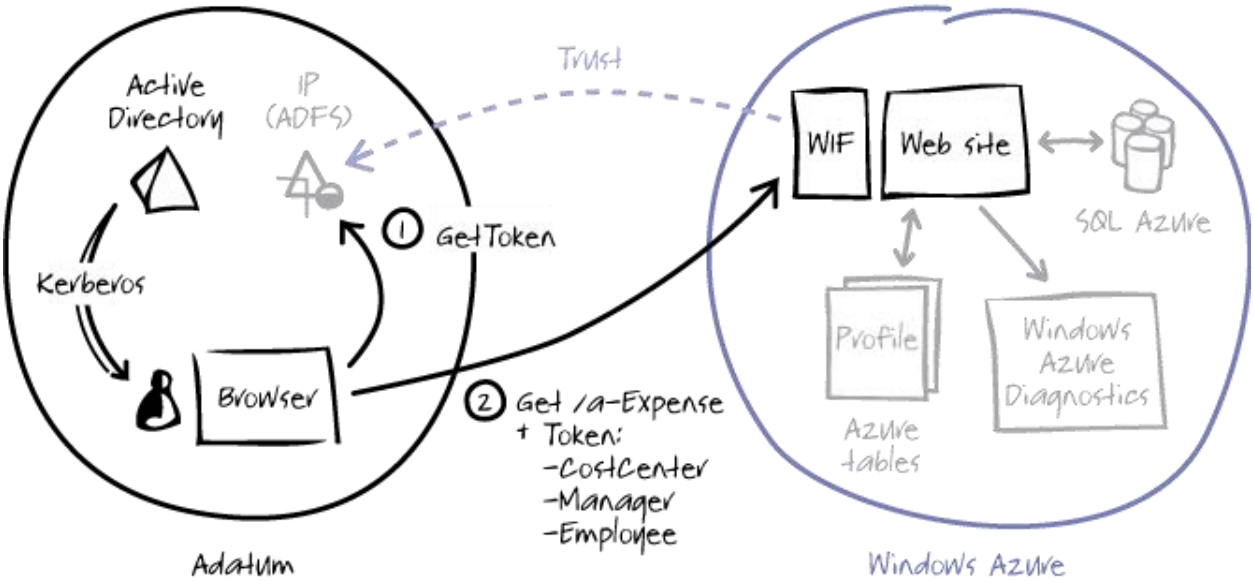


Рис. 2

aExpense как приложение, размещенное в Windows Azure

Реализация

Пришло время подробно рассмотреть процесс переноса aExpense в облако. По мере изучения этого раздела может потребоваться загрузка решения для систем разработки Microsoft Visual Studio ® со страницы http://wag.codeplex.com/. Это решение содержит реализации приложения aExpense до (в папке BeforeAzure) и после миграции (в папке Azure-SQLAzure). Если вы не заинтересованы в изучении таких механизмов, перейдите к следующему разделу.

Чтобы создать проект для облака, воспользуйтесь шаблоном проекта службы Visual Studio в облаке.

Создание веб-роли

Разработчики Adatum создали решение Visual Studio для версии aExpense для облака с помощью шаблона службы Windows Azure для облака. Этот шаблон создает требуемые файлы конфигурации и определения служб, а также файлы для веб-ролей и рабочих ролей, которые потребуются приложению.

Примечание.

Для дополнительной информации о создании проекта службы Windows Azure для облака в Visual Studio просмотрите список ресурсов в разделе «Разработка приложений Windows Azure» в главе 1, «Введение в платформу Windows Azure».

Эта первая версия aExpense для облака имеет единственную веб-роль, которая содержит весь код исходной локальной версии приложения.

Файл определения службы определяет конечную точку для веб-роли. Приложение aExpense имеет только одну конечную точку HTTPS, которая требует сертификат. В данном случае она называется «localhost». При развертывании приложения в Windows Azure вам также потребуется загрузить на сервер сертификат.

Копировать

```
<ServiceDefinition name="aExpense.Azure" xmlns="…">
 <WebRole name="aExpense">
  <Sites>
   <Site name="Web">
    <Bindings>
     <Binding name="Endpoint1" endpointName="Endpoint1" />
    </Bindings>
   </Site>
  </Sites>
  <Endpoints>
   <InputEndpoint name="Endpoint1" protocol="https" port="443"
    certificate="localhost" />
  </Endpoints>
  <Certificates>
   <Certificate name="localhost" storeLocation="LocalMachine"
    storeName="My" />
  </Certificates>
  <ConfigurationSettings>
   <Setting name="DataConnectionString" />
  </ConfigurationSettings>
  <Imports>
   <Import moduleName="Diagnostics" />
  </Imports>
```

```
  <LocalResources>
    <LocalStorage name="DiagnosticStore"
      cleanOnRoleRecycle="false" sizeInMB="5120" />
  </LocalResources>
 </WebRole>
</ServiceDefinition>
```

Примечание.

Сертификат «localhost» используется только для тестирования приложения. При развертывании приложения в Azure с использованием протокола HTTPS необходимо использовать сертификат, соответствующий URL-адресу приложения. При установлении соответствия URL-адреса другому адресу с помощью записи CNAME в DNS, сертификат должен соответствовать сопоставленному имени.

Файл конфигурации службы определяет веб-роль aExpense. Он содержит строки подключения, используемые ролью для доступа к хранилищу и подробным сведениям о сертификатах, используемых приложением. Приложение использует DataConnectionString для подключения к хранилищу Windows Azure с данными профилей, а DiagnosticsConnectionString — при подключении к хранилищу Windows Azure для сохранения журналов и данных о производительности. При развертывании приложения в облаке потребуется изменить строки подключения, чтобы приложение могло использовать хранилище Windows Azure.

Копировать

```
<ServiceConfiguration serviceName="aExpense.Azure" xmlns="…">
 <Role name="aExpense">
  <Instances count="1" />
  <ConfigurationSettings>
   <Setting name=
  "Microsoft.WindowsAzure.Plugins.Diagnostics.ConnectionString"
        value="DefaultEndpointsProtocol=https;
        AccountName={Azure storage account name};
        AccountKey={Общий ключ хранилища Azure}" />
   <Setting name="DataConnectionString"
        value="DefaultEndpointsProtocol=https;
        AccountName={Azure storage account name};
```

```
        AccountKey={Общий ключ хранилища Azure}" />
    </ConfigurationSettings>
    <Certificates>
      <Certificate name="localhost" thumbprint="…"
                   thumbprintAlgorithm="sha1" />
    </Certificates>
  </Role>
</ServiceConfiguration>
```

Примечание.

Значения «имя учетной записи хранилища Azure» и «общий ключ хранилища Azure» уникальны для вашей учетной записи в хранилище Windows Azure.

Комментарий Маркуса:

В главе 5 рассказывается о разработанном компанией Adatum автоматизированном редактировании файла конфигурации и передаче сертификата в процессе автоматизированного развертывания.

Защита aExpense

Перед миграцией в приложении aExpense применялась проверка подлинности Windows для пользователей. Это настраивается в файле Web.config приложения.

После миграции приложение aExpense делегирует процесс проверки учетных данных внешнему поставщику утверждений (вместо использования проверки подлинности Windows). Эти изменения конфигурации вносятся в файл Web.config.

Примечание.

Дополнительные сведения об удостоверениях, основанных на утверждениях, инструменте FedUtil и Windows Identity Foundation (WIF) см. в книге Руководство по управлению доступом и удостоверению, основанному на утверждениях. Копию этой книги в формате PDF можно загрузить по адресу http://msdn.microsoft.com/ru-ru/library/ff423674.aspx.

Первое, на что следует обратить внимание в файле Web.config, — режим проверки подлинности, для которого задано значение None, тогда как требование для всех пользователей проходить проверку подлинности осталось на месте.

Копировать

```
<authorization>
```

```
  <deny users="?" />
</authorization>
<authentication mode="None" />
```

Модули WSFederationAuthetication Module (FAM) и SessionAuthenticationModule (SAM) теперь выполняют процесс проверки подлинности. Метод загрузки этих модулей описан в разделе system.webServer файла Web.config.

Комментарий Маркуса:

Вы можете внести эти изменения в файл Web.config, запустив средство FedUtil.

Копировать

```
<system.webServer>
  …
  <add name="WSFederationAuthenticationModule"
      type="Microsoft.IdentityModel.Web.
          WSFederationAuthenticationModule, …" />
  <add name="SessionAuthenticationModule"
      type="Microsoft.IdentityModel.Web.
          SessionAuthenticationModule, …" />
</system.webServer>
```

При загрузке эти модули вставляются в конвейер ASP.NET для перенаправления не прошедших проверку подлинности запросов поставщику утверждений, обработки его ответов и преобразования маркера безопасности, отправленного поставщиком утверждений в объект ClaimsPrincipal. Кроме того, модули задают значение свойства HttpContext.User для объекта ClaimsPrincipal, таким образом, приложение получает к нему доступ.

Точнее, модуль WSFederationAuthenticationModule перенаправляет пользователя на страницу входа поставщика. Кроме того, он анализирует и проверяет маркера безопасности, который присылается в ответе. Этот модуль также записывает зашифрованный куки-файл, чтобы не повторять процедуру входа. Модуль SessionAuthenticationModule определяет куки-файл входа, расшифровывает его и повторно заполняет объект ClaimsPrincipal. После того как поставщик утверждения проведет проверку подлинности пользователя, приложение aExpense получает доступ к имени пользователя, прошедшего проверку подлинности.

Файл Web.config содержит новый раздел для модели идентификации Microsoft.IdentityModel, инициализирующий среду Windows Identity Foundation (WIF).

Копировать

```
<microsoft.identityModel>
  <service>
    …
  </service>
</microsoft.identityModel>
```

Можно также использовать стандартный элемент управления для обработки процесса выхода пользователя из приложения. В следующем примере кода из файла Site.Master показана часть определения стандартного заголовка страницы.

Копировать

```
<div id="toolbar">
    Выполнен вход в качестве:
    <i>
      <%= Microsoft.Security.Application.Encoder.HtmlEncode
          (this.Context.User.Identity.Name) %>
    </i> |
    <idfx:FederatedPassiveSignInStatus
        ID="FederatedPassiveSignInStatus1"
        runat="server"
        OnSignedOut="FederatedPassiveSignInStatus1SignedOut"
        SignOutText="Logout" FederatedPassiveSignOut="true"
        SignOutAction="FederatedPassiveSignOut" />
</div>
```

Кроме того, вы заметите небольшое изменение в том, как в aExpense проводится авторизация. Поскольку теперь в файле Web.config для режима проверки подлинности задано значение None, правила авторизации в этом файле явно запрещают всем пользователям доступ и предоставление доступа для указанной роли.

Копировать

```
<location path="Approve.aspx">
```

```
  <system.web>
    <authorization>
      <allow roles="Manager" />
      <deny users="*"/>
    </authorization>
  </system.web>
</location>
```

Теперь для предоставления приложению сведений о членстве в ролях используется поставщик утверждений (вместо функции управления ролями ASP.NET).

Есть еще одно изменение в приложении, которое может влиять на процедуру проверки подлинности. При необходимости запускать приложение aExpense в нескольких экземплярах роли в Windows Azure механизм шифрования куки-файлов по умолчанию (использующий DPAPI) использовать нельзя, поскольку у всех экземпляров разные ключи. Это означало бы, что куки-файл, созданный одним экземпляром веб-роли, не был бы доступен для чтения другим экземплярам веб-роли. Для решения этой проблемы следует применять механизм шифрования файлов cookie, в котором используются ключи, общие для всех экземпляров веб-роли. В приведенном ниже коде из файла Global.asax показана процедура замены объекта SessionSecurityHandler по умолчанию и настройка использования в нем класса RsaEncryptionCookieTransform.

Комментарий Бхарата:

Хотя при первоначальном развертывании aExpense в Windows Azure используется только одна веб-роль, нужно обеспечить ее правильную работу при вертикальном масштабировании приложения. Именно поэтому мы используем для шифрования куки-файла сеанс RSA с сертификатом.

Копировать

```
private void OnServiceConfigurationCreated(object sender,
  ServiceConfigurationCreatedEventArgs e)
{
  // Использовать <serviceCertificate> для защиты куки-файлов,
  // отправляемых клиенту.
  List<CookieTransform> sessionTransforms =
    new List<CookieTransform>(
```

```
            new CookieTransform[]
            {
                new DeflateCookieTransform(),
                new RsaEncryptionCookieTransform(
                    e.ServiceConfiguration.ServiceCertificate),
                new RsaSignatureCookieTransform(
                    e.ServiceConfiguration.ServiceCertificate)
            });
        SessionSecurityTokenHandler sessionHandler =
         Новые
          SessionSecurityTokenHandler(sessionTransforms.AsReadOnly());

        e.ServiceConfiguration.SecurityTokenHandlers.AddOrReplace(
            sessionHandler);
    }
```

Управление данными пользователей

Перед миграцией в aExpense для получения из Active Directory сведений о центре расходов, руководителе и отображаемом имени использовался запрос LDAP. Для получения сведений о членстве пользователя в роли использовался поставщик ролей ASP.NET, а для получения данных приложения — профилировщик ASP.NET (в данном случае — предпочтительный метод компенсации). В следующей таблице приведены сводные данные о доступе aExpense к данным пользователя и о том, где хранились данные до миграции.

Пользовательские данные	Механизм доступа	Хранилище
Членство в роли	Поставщик ролей ASP.NET	SQL Server
Центр расходов	LDAP	Active Directory
Руководитель	LDAP	Active Directory
Отображаемое имя	LDAP	Active Directory

Имя пользователя	Поставщик членства ASP.NET	SQL Server
Предпочтительный метод Метод	Поставщик профилей ASP.NET	SQL Server

После миграции в aExpense по-прежнему используются те же данные пользователя, но доступ к ним осуществляется по-другому. В следующей таблице приведены сводные данные о доступе aExpense к данным пользователя и о том, где хранятся данные после миграции.

Пользовательские данные	Механизм доступа	Хранилище
Членство в роли	ADFS	Active Directory
Центр расходов	ADFS	Active Directory
Руководитель	ADFS	Active Directory
Отображаемое имя	ADFS	Active Directory
Имя пользователя	ADFS	Active Directory
Предпочтительный метод Метод	Поставщик профилей ASP.NET	Хранение таблиц Windows Azure

Внешний поставщик передает данные утверждения в приложение aExpense после успешной проверки подлинности пользователя приложения. Приложение aExpense использует данные утверждения в течение сеанса, хранить эти данные не требуется.

Внешний поставщик передает данные из утверждения в приложение aExpense после успешной проверки подлинности пользователя приложения.

Приложение может считывать значения отдельных утверждений, когда ему требуется доступ к данным утверждений. Чтобы увидеть, как это делается, посмотрите класс ClaimHelper.

Данные профиля

До миграции для хранения параметров пользователя приложения в приложении aExpense использовалась функция работы с профилями ASP.NET. В Adatum пытаются избежать настройки схемы в Active Directory, поэтому в aExpense предпочтительный метод компенсации пользователя сохраняется с использованием функции работы с профилями. Поставщик профилей по умолчанию сохраняет свойства профилей в базе данных SQL Server.

Комментарий По:

Мы предпочитаем не настраивать схему Active Directory, если этого можно избежать. Изменения схемы вызывают далеко идущие последствия, и их трудно откатить.

Использование функции работы с профилями упрощает приложению сохранение небольших фрагментов данных пользователя. Включение функции работы с профилями и выбор сохраняемых свойств производятся в файле Web.config.

Копировать

```
<profile defaultProvider="SqlProvider">
 <providers>
  <clear />
  <add name="SqlProvider"
      type="System.Web.Profile.SqlProfileProvider"
      connectionStringName="aExpense"
      applicationName="aExpense" />
 </providers>
 <properties>
  <add name="PreferredReimbursementMethod" />
 </properties>
</profile>
```

Затем можно получить доступ к значению свойства профиля в коде, например так.

Копировать

```
var profile = ProfileBase.Create(userName);
string prm =
   profile.GetProperty<string>("PreferredReimbursementMethod");
```

После миграции в aExpense для хранения предпочтительного метода компенсации для каждого пользователя по-прежнему применяется система профилей. Хотя в Windows Azure можно использовать поставщик профилей SQL Server (с помощью пользовательских скриптов, доступных по адресу http://support.microsoft.com/kb/2006191/), решение использует тот же образец поставщика, что использует хранилище таблиц Windows Azure для хранения сведений о профилях. Этот поставщик можно загрузить по адресу http://code.msdn.microsoft.com/windowsazuresamples. Единственное изменение в приложении, необходимое для поддержки другого поставщика профилей, вносится в файл Web.config.

Комментарий Маркуса:

Доступ к данным профиля с помощью поставщика профилей сводит к минимуму изменение кода приложения.

Копировать

```
<profile defaultProvider="TableStorageProfileProvider">
 <providers>
  <clear />
  <add name="TableStorageProfileProvider"
      type="AExpense.Providers.TableStorageProfileProvider …"
      applicationName="aExpense" />
 </providers>

 <properties>
  <add name="PreferredReimbursementMethod" />
 </properties>
</profile>
```

Использование класса TableStorageProfileProvider вызывает ряд проблем в приложении:

Класс TableStorageProfileProvider является неподдерживаемым образцом кода.

Следует выполнить миграцию данных существующего профиля из хранилища таблиц SQL Server в Windows Azure.

Нужно понять, подходит ли в конечном итоге хранилище таблиц Windows Azure для хранения данных профиля.

Даже с учетом этих соображений использование поставщика профилей хранилища таблиц позволило Adatum свести изменения в приложении к минимуму. Это означает, что эксплуатационные расходы для приложения будут ниже по сравнению с SQL Azure.

Примечание.

Глава 4, «Сколько это будет стоить?», описывает относительную стоимость использования

хранилищ Windows Azure и SQL Azure.

Глава 5, «Этап 2. Автоматизация развертывания и использования хранилища Windows Azure», содержит более подробные сведения об использовании хранилища таблиц Windows Azure.

Подключение к SQL Server

До миграции данные приложения aExpense хранились в базе данных SQL Server. На этом первом этапе разработчики перенесли базу данных SQL Azure, а код доступа к данным в приложении остался без изменений. Изменения потребовались только в одном месте — в строке подключения в файле Web.config.

Для подключения к SQL Azure вместо локального SQL Server требуется только изменить конфигурацию.

Копировать

```
<add name="aExpense" connectionString=
  "Data Source={Server Name};
   Initial Catalog=aExpense;
   UId={SQL Azure User Id};
   Pwd={SQL Azure User Password};
   Encrypt=True;
   TrustServerCertificate=False;"
 providerName="System.Data.SqlClient" />
```

Примечание.

Значения Имя сервера, Идентификатор пользователя SQL Azure и Пароль пользователя SQL Azure — это значения для вашей учетной записи SQL Azure.

Есть две вещи, которые следует учитывать в связи со строкой подключения. Во-первых, обратите внимание, что, поскольку SQL Azure не поддерживает проверку подлинности Windows, учетные данные для вашей учетной записи SQL Azure хранятся в виде открытого текста. Следует продумать шифрование этого раздела файла Web.config. Это повысит сложность приложения, но также сделает и данные более защищенными. Если приложение, возможно, будет выполняться в нескольких экземплярах роли, необходимо применять механизмы шифрования, в которых используются общие ключи для всех экземпляров роли.

Примечание.

Для шифрования строки подключения SQL в файле Web.config можно использовать поставщик защищенной конфигурации Pkcs12, который можно загрузить по адресу http://archive.msdn.microsoft.com/pkcs12protectedconfg. Дополнительные сведения об использовании этого поставщика можно найти в записях блогов разработчиков SQL Azure,

например: http://blogs.msdn.com/b/sqlazure/archive/2010/09/07/10058942.aspx.

Во-вторых, следует помнить, что параметры в строке подключения задают шифрование соединения с SQL Azure. Хотя приложение может находиться на компьютере в том же центре обработки данных, что и SQL Azure, это соединение следует рассматривать как соединение через Интернет.

Любой трафик в центре обработки данных рассматривается как интернет-трафик и должен шифроваться.

Комментарий Бхарата:

Можно усилить защиту базы данных SQL Azure, настроив брандмауэр SQL Azure на портале SQL Azure. В брандмауэре SQL Azure можно указать IP-адреса компьютеров, которым разрешено подключение к серверу SQL Azure.

Время ожидания соединения SQL Azure

Можно задать значение времени ожидания соединения при попытке подключения к SQL Azure. Если время ожидания истекло до того, как соединение установлено, возникает ошибка, которую должно обработать приложение. То, как приложение обрабатывает ошибку истечения времени ожидания, зависит от конкретных обстоятельств, однако возможны следующие стратегии: повторные попытки до успешного соединения, сообщение пользователю об ошибке, регистрация сообщения об ошибке в журнале и переход к другой задаче.

Значение времени ожидания соединения по умолчанию составляет 15 секунд, но, поскольку соглашение об уровне обслуживания (SLA) SQL Azure указывает, что нарушение SLA происходит только через 30 секунд, следует установить время ожидания равным 30 секундам.

Примечание.

Для повторных попыток можно использовать делегат RetryPolicy в пространстве имен Microsoft.WindowsAzure.StorageClient. В статье по адресу http://blogs.msdn.com/windowsazurestorage/archive/2010/04/22/savechangeswithretries-and-batch-option.aspx описано использование делегата для повторных попыток сохранения изменений в хранилище таблиц Windows Azure, однако этот метод можно приспособить к работе с контекстным объектом в LINQ to SQL или ADO.NET Entity Framework.

Обработка оборванных соединений

Если соединение с SQL Azure оборвалось при использовании его приложением, его следует немедленно восстановить. Если возможно, следует попытаться повторно выполнить операцию, которая выполнялась перед обрывом соединения, либо повторить транзакцию, если это была транзакция. Соединение может завершиться с ошибками в интервале между отправкой сообщения о фиксации транзакции и получением сообщения с отчетом о результате транзакции. В этом случае должен быть способ проверить успешность завершения транзакции, чтобы установить необходимость повторной попытки ее выполнения.

Диагностика

Приложение aExpense использует блок журналирования приложения и блок обработки исключений приложения из библиотеки Enterprise Library. В облачной версии aExpense по-прежнему используются блоки приложения, что помогает сотрудникам Adatum устранять неполадки. Хотя для использования облачной версии aExpense требуется минимум изменений, процедура доступа к файлам журнала другая.

Комментарий Яны:

Блок журналирования приложения и блок обработки исключений приложения входят в состав библиотеки Enterprise Library. В Adatum мы используем их в ряде приложений.

Чтобы приложение aExpense регистрировало данные в журналах Windows Azure, в Adatum внесли в файл Web.config ряд изменений, позволяющих блоку журналирования приложения использовать прослушиватель трассировки Windows Azure.

Комментарий По:

Нам нужен прежний доступ к диагностическим данным и после перехода на облачную модель.

Копировать

```
<listeners>
<add listenerDataType="Microsoft.Practices.EnterpriseLibrary.
  Logging.Configuration.SystemDiagnosticsTraceListenerData,
  Microsoft.Practices.EnterpriseLibrary.Logging, Version=5.0.414.0,
  Culture=neutral, PublicKeyToken=31bf3856ad364e35"
  type="Microsoft.WindowsAzure.Diagnostics
.DiagnosticMonitorTraceListener,
  Microsoft.WindowsAzure.Diagnostics, Version=1.0.0.0, Culture=neutral,
  PublicKeyToken=31bf3856ad364e35"
  traceOutputOptions="Timestamp, ProcessId"
  name="System Diagnostics Trace Listener" />
</listeners>
```

При создании нового проекта облачной службы Windows Azure в Visual Studio в файле Web.config по-прежнему будет конфигурация прослушивателя трассировки Azure. В следующем примере кода

из файла Web.config показана конфигурация прослушивателя трассировки, которую следует добавить при проведении миграции существующего веб-приложения ASP.NET.

Копировать

```
<system.diagnostics>
  <trace>
    <listeners>
      <add type="Microsoft.WindowsAzure.Diagnostics
.DiagnosticMonitorTraceListener,
          Microsoft.WindowsAzure.Diagnostics, Version=1.0.0.0,
          Culture=neutral, PublicKeyToken=31bf3856ad364e35"
          name="AzureDiagnostics">
        <filter type="" />
      </add>
    </listeners>
  </trace>
 </system.diagnostics>
```

По умолчанию в Windows Azure диагностические данные не сохраняются в хранилище автоматически, они находятся в буфере памяти. Для доступа к диагностическим данным нужно либо добавить в приложение код, переносящий данные в хранилище Windows Azure, либо добавить в проект файл конфигурации диагностики. Можно либо запланировать в Windows Azure периодический перенос данных журнала в хранилище, либо выполнять эту задачу по запросу. В Adatum решили использовать файл конфигурации диагностики для управления переносом данных журнала в постоянное хранилище. Преимущество использования файла конфигурации состоит в том, что позволяет Adatum собирать данные трассировки из метода Application_Start, где приложение aExpense выполняет процедуры инициализации. В приведенном ниже образце кода показан пример файла diagnostics.wadcfg из этого решения.

Копировать

```
<?xml version="1.0" encoding="utf-8" ?>
<DiagnosticMonitorConfiguration
xmlns="http://schemas.microsoft.com/ServiceHosting/2010/10/DiagnosticsConfiguration"
    configurationChangePollInterval="PT1M"
    overallQuotaInMB="5120">
 <DiagnosticInfrastructureLogs bufferQuotaInMB="1024"
```

```
    scheduledTransferLogLevelFilter="Verbose"
    scheduledTransferPeriod="PT1M" />
  <Logs bufferQuotaInMB="1024"
    scheduledTransferLogLevelFilter="Verbose"
    scheduledTransferPeriod="PT1M" />
  <Directories bufferQuotaInMB="1024"
    scheduledTransferPeriod="PT1M">

    <!-- Эти три элемента задают конкретные каталоги,
         указанные для типов журналов -->
    <CrashDumps container="wad-crash-dumps" directoryQuotaInMB="256" />
    <FailedRequestLogs container="wad-frq" directoryQuotaInMB="256" />
    <IISLogs container="wad-iis" directoryQuotaInMB="256" />

  </Directories>
  <PerformanceCounters bufferQuotaInMB="512" scheduledTransferPeriod="PT1M">
    <!-- Счетчик задается в том же формате, что в
        API принудительной настройки диагностики -->
    <PerformanceCounterConfiguration
      counterSpecifier="\Processor(_Total)\% Processor Time" sampleRate="PT5S" />
  </PerformanceCounters>
  <WindowsEventLog bufferQuotaInMB="512"
    scheduledTransferLogLevelFilter="Verbose"
    scheduledTransferPeriod="PT1M">
    <!-- Имя журнала событий имеет тот же формат, что в
        API принудительной настройки диагностики -->
    <DataSource name="System!*" />
  </WindowsEventLog>
</DiagnosticMonitorConfiguration>
```

Значение overallQuotaInMB должно быть больше суммы значений bufferQuotaInMB в файле diagnostics.wadcfg, необходимо настроить ресурс локального хранилища в веб-роли с именем «DiagnosticsStore», минимальный размер которого не меньше значения overallQuotaInMB в файле конфигурации диагностики. В этом примере файлы журнала переносятся в хранилище каждую минуту, затем их можно просмотреть с помощью любого средства просмотра, например в окне обозревателя серверов в Visual Studio.

Дополнительные сведения о файле thediagnostics.wadcfg см. в блоге. Этот блог также содержит некоторые полезные советы.

http://msdn.microsoft.com/ru-ru/library/gg604918.aspxhttp://blogs.msdn.com/b/davidhardin/archive/2011/03/29/configuring-wad-via-the-diagnostics-wadcfg-config-file.aspx.

Комментарий Бхарата:

Поскольку сохранение диагностических данных в хранилище Windows Azure оплачивается, необходимо планировать, как долго они будут там храниться и как они будут загружаться для анализа в автономном режиме.

Установка и физическое развертывание

При разработке приложения для Windows Azure лучше всего максимально использовать для разработки и тестирования локальный эмулятор вычислений и эмулятор хранения. Когда приложение готово, его можно развертывать в Windows Azure для дальнейшего тестирования. Не следует вносить никакие изменения в код перед развертыванием в Windows Azure, однако следует обязательно передать все сертификаты, которые требуются для приложения, и обновить все файлы конфигурации для среды Windows Azure.

Примечание.

Сертификаты SSL можно передать в Windows Azure с помощью портала разработчика Windows Azure по адресу http://windows.azure.com/ или с помощью скрипта. Дополнительные сведения о передаче сертификатов с помощью скрипта командной строки Windows PowerShell™ см. в главе 5, «Этап 2. Автоматизация развертывания и использование хранилища Windows Azure».

Экземпляры ролей, домены обновлений и отказоустойчивости

Развертывание нескольких экземпляров веб-ролей и рабочих ролей представляет собой простой способ горизонтального масштабирования приложения в соответствии с растущими запросами. Кроме того, экземпляры ролей легко добавлять и удалять по мере необходимости либо через портал разработчиков Windows Azure, либо с помощью скриптов. Таким образом, оплачиваются только те услуги, которые действительно нужны. С помощью нескольких экземпляров ролей можно также обеспечивать отказоустойчивость приложения и добавить возможность выполнять оперативные обновления приложения.

Используйте несколько экземпляров ролей для горизонтального масштабирования приложения, обеспечения отказоустойчивости и возможности обновления «на месте» (live).

При использовании двух или более экземпляров ролей Windows Azure организует их в виртуальные группы, известные как домены обновлений. При обновлении приложения «на месте» (live) Windows Azure обновляет по одному домену за один раз, что обеспечивает доступность приложения в течение всего процесса. Windows Azure останавливает, обновляет и перезапускает все экземпляры роли в домене обновления, прежде чем переходить к следующему.

Примечание.

Ниже приведены некоторые ограничения типов обновления, для которых поддерживается такой режим. В частности, нельзя изменять конфигурацию службы и добавлять или удалять роли в приложении. Кроме того, в файле конфигурации службы можно указать, сколько доменов обновления должно быть в приложении.

В Windows Azure домены отказоустойчивости представляют собой физическую единицу, выходящую из строя в результате отказа. При наличии двух или более экземпляров ролей Windows Azure выделит их в несколько доменов отказоустойчивости, так что при выходе из строя одного домена все равно остаются запущенные экземпляры приложения. Windows Azure определяет, сколько доменов отказоустойчивости используют приложение.

Windows Azure также обеспечивает ортогональность доменов обновления и отказоустойчивости, чтобы экземпляры ролей в домене обновления были распределены по разным доменам отказоустойчивости.

Скрипты развертывания

Изменение файлов конфигурации приложения вручную перед развертыванием в Windows Azure чревато возникновением ошибок. Разработчики Adatum создали набор скриптов развертывания, автоматически обновляющих файлы конфигурации, упаковывающие файлы приложения и передающие приложение в Windows Azure. Эти скрипты будут рассматриваться позднее.

Использование фиктивного (Mock) поставщика

По умолчанию доступная для загрузки версия aExpense настроена для запуска на автономной рабочей станции разработки. Это аналогично возможному развертыванию ваших собственных приложений. Обычно проще начинать с одного компьютера разработки.

Комментарий По:

Использование простого, созданного разработчиком поставщика утверждений — хорошая практика на этапе разработки и тестирования.

Чтобы пользоваться этим методом, разработчики aExpense написали небольшую реализацию заглушки поставщика. Этот код можно найти в доступном для загрузки решении Visual Studio. Проект находится в папке Dependencies и имеет имя Adatum.SimulatedIssuer.

При первом запуске приложение aExpense будет обращаться к поставщику. Поставщик выдает стандартные утверждения.

Подобный компонент написать нетрудно, и загружаемый образец можно использовать повторно либо начать с шаблона, включенного в состав пакета SDK Windows Identity Foundation (WIF).

Примечание.

SDK WIF можно загрузить в центре загрузки Microsoft. В книге «Руководство по идентификаторам и управлению доступом на основе утверждений» описано несколько способов создания поставщика утверждений. Можно загрузить копию этой книги в формате PDF по адресу http://msdn.microsoft.com/ru-ru/library/ff423674.aspx.

Переход к реальному (production) поставщику

Когда все будет готово для развертывания в производственной среде, потребуется переход от имитации поставщика, работающего на вашей рабочей станции разработки, к компоненту типа ADFS 2.0.

Для этого нужно выполнить два шага. Во-первых, нужно изменить файл Web.config веб-приложения с помощью программы FedUtil таким образом, чтобы в нем указывался реальный поставщик. Далее нужно настроить поставщик таким образом, чтобы он распознавал запросы от данного веб-приложения и выдавал соответствующие утверждения.

Примечание.

Дополнительные сведения о программе FedUtil и настройке выдачи утверждений в приложениях см. в книге «Руководство по идентификаторам и управлению доступом на основе утверждений». Можно загрузить копию этой книги в формате PDF по адресу http://msdn.microsoft.com/ru-ru/library/ff423674.aspx.

Инструкции по добавлению проверяющей стороны и правил утверждений см. в документации, предоставленной вашим производственным поставщиком.

При перенаправлении запроса поставщику утверждений следует включить в запрос параметр wreply, который предписывает поставщику возвращать утверждения. При тестировании приложения в автономном режиме или в облаке не требуется явно указывать этот URL-адрес, поскольку он должен отражать действительный адрес приложения. В следующем коде показано, как приложение aExpense динамически создает значение wreply в файле Global.asax.cs.

Динамическое построение параметра wreply упрощает тестирование приложения в различных средах.

Копировать

```
private void
  WSFederationAuthenticationModule_RedirectingToIdentityProvider
  (object sender, RedirectingToIdentityProviderEventArgs e)
```

```
{
    // В среде Windows Azure строится параметр wreply
    // для запроса SignIn, отражающего реальный
    // адрес приложения.
    HttpRequest request = HttpContext.Current.Request;
    Uri requestUrl = request.Url;
    StringBuilder wreply = new StringBuilder();

    wreply.Append(requestUrl.Scheme); // например, "http" или "https"
    wreply.Append("://");
    wreply.Append(request.Headers["Host"] ??
        requestUrl.Authority);
    wreply.Append(request.ApplicationPath);

    if (!request.ApplicationPath.EndsWith("/"))
    {
        wreply.Append("/");
    }

    e.SignInRequestMessage.Reply = wreply.ToString();
}
```

Заглушка Active Directory

Приложение aExpense использует проверку подлинности Windows. Поскольку разработчики не управляют идентификаторами в корпоративном каталоге компании, иногда бывает удобно заменить Active Directory на «заглушку» на период разработки приложения.

Собственное приложение aExpense (до миграции) являет собой подобный пример. Чтобы воспользоваться этим способом, потребуется внести небольшие изменения в файл Web.config, чтобы заменить проверку подлинности Windows на проверку подлинности с помощью форм, затем добавить в приложение имитацию хранилища профилей LDAP. Замените проверку подлинности Windows на проверку подлинности с помощью форм путем внесения следующего изменения в файл Web.config.

Копировать

```
<authentication mode="Forms">
  <forms name=".ASPXAUTH"
     loginUrl="~/SimulatedWindowsAuthentication.aspx"
     defaultUrl="~/default.aspx" requireSSL="true">
  </forms>
</authentication>
```

Необходимо добавить в приложение страницу входа, позволяющую выбирать пользователя, используемого для тестирования. В aExpense эта страница — SimulatedWindowsAuthentication.aspx.

Требуется также добавить класс, имитирующий поиск LDAP для атрибутов Active Directory, необходимых приложению. В этом примере метод GetAttributes имитирует запрос LDAP «&(objectCategory=person)(objectClass=user);costCenter;manager;displayName».

Копировать

```
public static class SimulatedLdapProfileStore
{
   public static Dictionary<string, string> GetAttributesFor(
      string userName, string[] attributes)
   {
      Dictionary<string, string> results;

      switch (userName)
      {
         case "ADATUM\\johndoe":
            results = new Dictionary<string, string>
            {
               { "costCenter", "31023" },
               { "manager", "ADATUM\\mary" },
               { "displayName", "John Doe" }
            };
            break;
```

```
            …
        }


        return results;
    }
}
```

Примечание.

Эти образцы кода взяты из решения BeforeAzure в загружаемых решениях.

SQL Server

На данном этапе команда разработчиков компании Adatum работает с образцами данных, так что скрипт развертывания формирует образец базы данных в SQL Azure. Потребуется создание скрипта, с помощью которого данные из локальной версии SQL Server передаются в Windows Azure при переносе активного приложения. Чтобы перенести существующую схему базы данных в SQL Azure, можно использовать среду SQL Server Management Studio для экспорта схемы в качестве скрипта Transact-SQL, а затем запустить скрипт в SQL Azure. Для перемещения данных в SQL Azure можно использовать службу SQL Server Integration Service. Однако команда разработчиков планирует проанализировать, нужно ли вообще использовать SQL Azure, или же приложение сможет использовать хранилище таблиц Windows Azure. Таким образом, разработкой стратегии миграции данных они еще не занимались.

Примечание.

С общими сведениями об ограничениях SQL Azure, которые необходимо учитывать, можно ознакомиться на странице http://msdn.microsoft.com/ru-ru/library/ff394102.aspx.

Примечание.

С общими сведениями о том, как перенести базу данных в SQL Azure, можно ознакомиться на странице http://msdn.microsoft.com/ru-ru/library/ee730904.aspx. Кроме того, можно использовать мастер миграции SQL Azure на странице http://sqlazuremw.codeplex.com/, который поможет перенести локальные базы данных SQL Server в SQL Azure.

Доступ к файлам журналов диагностики

Локальная версия приложения aExpense использует блок приложения журналирования и блок приложения обработки исключений для сбора сведений из приложения и их записи в журнал событий Windows. Приложение версии Windows Azure использует те же блоки приложения и благодаря изменению конфигурации может записывать данные журнала в систему журналирования Windows Azure. Однако для доступа к данным журнала в Windows Azure необходимо выполнить несколько дополнительных действий. Во-первых, следует сохранить

данные журнала в постоянном хранилище. Это можно выполнить вручную с помощью портала разработчиков Windows Azure либо путем добавления кода или файла diagnostics.wadcfg в приложение, чтобы создавать дамп данных журнала в хранилище через запланированные интервалы времени. Во-вторых, требуется какой-либо способ просмотра данных журнала в хранилище Windows Azure.

Комментарий По:

Для доступа к файлам журналов в Windows Azure можно использовать программу удаленного доступа к хранилищу Windows Azure или разработать определенные скрипты, которые будут регулярно загружать такие данные.

Дополнительные сведения о средствах, которые можно использовать для управления данными журналов, см. в перечне средств на странице:

http://www.microsoft.com/windowsazure/tools/.

Дополнительные сведения

Последние версии инструментов Windows Azure для среды Microsoft Visual Studio и пакет SDK для Windows Azure можно скачать в центре загрузки Microsoft.

В MSDN содержится много сведений о Windows Azure и SQL Azure. Рекомендуется начать со страницы http://msdn.microsoft.com/ru-ru/library/dd163896.aspx.

На странице «Приступая к работе с Windows Azure» (http://www.microsoft.com/windowsazure/getstarted) содержатся ссылки на множество ресурсов, где можно узнать о разработке приложений для Windows Azure.

Глава 4. Сколько это будет стоить?

В этой главе представлена базовая модель расчета затрат на размещение приложения aExpense в облаке. Она основана на некоторых предположениях об использовании приложения и использует текущие цены на службы технологической платформы Windows ® Azure ™ для оценки ежегодных эксплуатационных затрат.

Исходные условия

Существующая локальная версия aExpense является стандартным бизнес-приложением. Компания Adatum выбрала это приложение для экспериментального проекта переноса в облако, так как это приложение включает возможности, общие для многих других бизнес-приложений Adatum, и компания рассчитывает, что полученный в данном проекте опыт может быть использован в других случаях. В Adatum развернули приложение aExpense в центре обработки данных с компонентами, установленными на нескольких различных серверах. Веб-приложение размещается на сервере Windows Server совместно с другим приложением. Также приложение aExpense совместно с несколькими другими приложениями использует установленное программное обеспечение базы

данных SQL Server® и имеет выделенный массив дисков для хранения отсканированных чеков оплаты расходов.

Текущее развертывание приложения aExpense рассчитано на среднее использование, а не на максимальное, таким образом два дня в конце месяца, когда большинство пользователей представляет требования по возмещению деловых расходов, приложение может работать медленно и зависать.

Цели и требования

Компания Adatum затрудняется точно определить стоимость эксплуатации текущей версии приложения aExpense. Приложение использует несколько различных серверов, разделяет установку SQL Server с несколькими другими бизнес-приложениями, кроме того, для него выполняется резервное копирование в рамках общей стратегии резервного копирования в центре обработки данных.

Трудно произвести оценку эксплуатационных расходов для существующего локального приложения.

Несмотря на то что в Adatum затрудняются определить существующие эксплуатационные расходы на приложение, компания планирует оценить стоимость работы приложения в облаке. Одной из конкретных целей этого пробного проекта является определение степени точности прогнозирования эксплуатационных расходов для приложений в облаке.

Вторая цель заключается в оценке снижения издержек, которого можно добиться, по-разному настраивая приложение. Затем компания Adatum сможет определить стоимость конкретного уровня обслуживания, что позволит гораздо проще выполнять анализ прибыли и издержек для приложения. Конкретным примером этого в приложении aExpense является оценка стоимости развертывания для максимальной загрузки в период активности в конце месяца.

Комментарий Бхарата:

Можно управлять стоимостью приложения для облака, меняя его поведение путем изменения конфигурации.

В целом компании Adatum хотелось бы достичь большей прозрачности в управлении затратами на свой пакет бизнес-приложений.

Обзор решения

Первым шагом нужно выяснить, за что компании Adatum будет выставляться ежемесячный счет в связи с версией приложения aExpense для облака. На рис. 1 показаны службы, за которые Microsoft предъявит Adatum ежемесячный счет за версию приложения aExpense для облака.

Комментарий Бхарата:

В простую модель стоимости в этой главе не включена оценка стоимости каких-либо рабочих ролей, которые Adatum может добавлять в приложение aExpense, и за основу принимается один небольшой экземпляр веб-роли. Кроме того, в данную модель не входит оценка затрат на среду тестирования. Компания Adatum должна

пересмотреть оценку затрат по завершении разработки приложения aExpense и его нагрузочного тестирования.

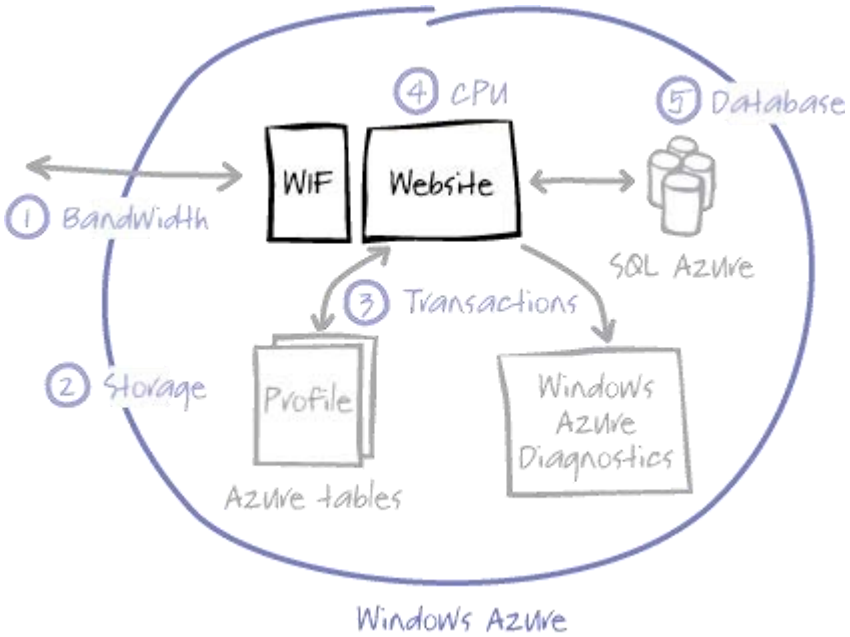


Рис. 1

Платные услуги

В следующей таблице приведены действующие месячные ставки за эти услуги в долларах США.

Услуга	Описание	Стоимость
1. Входящий/исходящий трафик	Это веб-трафик между браузером пользователя и сайтом aExpense.	Входящий: бесплатно Исходящий: 0,12 долл./ГБ
2. Хранилище Windows Azure	В приложении aExpense оно будет изначально использоваться только для хранения данных профилей. Позднее здесь будут также храниться отсканированные изображения чеков.	0,125 долл./ГБ
3. Транзакции	Тарифицируются все взаимодействия с системой хранения данных.	0,01 долл./10 000 транзакций
4. Вычисление	Время работы веб-ролей приложения aExpense.	Небольшая роль (small instance) 0,12 долл./час

5. Хранилище SQL	База данных SQL Azure.	0–100 МБ: $4.995 Более 100 МБ и до 1 ГБ: $9.99 Более 1 ГБ и до 10 ГБ: $9.99 за первый ГБ, $3.996 за каждый дополнительный ГБ Более 10 ГБ и до 50 ГБ: $45.954 за первые 10 ГБ, $1.998 за каждый дополнительный ГБ Более 50 ГБ и до 150 ГБ: $125.874 за первые 50 ГБ, $0.999 за каждый дополнительный ГБ

Примечание.

Цены, приведенные здесь, актуальны для рынка США апрель 2012. Последние сведения о ценах см. на странице http://www.microsoft.com/WindowsAzure/offers/. На той же странице приводится тарификация для других регионов.

Комментарий Бхарата:

После оценки уровня использования платных услуг вашим приложением можно воспользоваться Калькулятором стоимости Windows Azure на странице http://www.microsoft.com/windowsazure/pricing-calculator/ для быстрого расчета ежемесячных издержек.

Оценка затрат на трафик для приложения aExpense

aExpense не является приложением с интенсивным трафиком. При условии, что все отсканированные изображения чеков будут пересылаться в приложение туда и обратно дважды, и принимая во внимание веб-трафик для приложения, компания Adatum предполагает, что каждый месяц в одну сторону будет передаваться 4 ГБ данных.

Передача данных	ГБ/месяц	долл./ГБ/месяц	Итого в месяц
Входящий	4 ГБ	В настоящее время бесплатно	$0,00

Исходящий	4 ГБ	$0,12	$0,48
		Итого в год	$5,76

Оценка хранилища Windows Azure для приложения aExpense

Исходя из анализа сложившегося использования, в среднем 60 % из 5 000 работников компании Adatum представляют 10 записей по деловым расходам в месяц. Размер каждого отсканированного изображения чеков составляет в среднем 15 КБ, а для выполнения законодательных требований приложение должно сохранять данные в течение 7 лет. В результате ожидаемая потребность в хранилище для приложения составит 36 ГБ.

Хранилище		Стоимость	Итого в месяц
Хранящиеся ГБ/месяц	36 ГБ	0,125 долл./ГБ	$4,5
Хранение транзакций/месяц	90 000	0,01 долл./10 К	$0,09
		Итого в год	$55,08

Расчет оценки затрат для приложения aExpense

На данной момент компания Adatum исходит из того, что приложение будет работать 24 часа в сутки, 365 дней в году.

Часов	долл./час	Количество экземпляров	Итого в год
8760	$0,12	2	$2 102,40

Примечание.

Для соответствия соглашению об уровне обслуживания (SLA) для Windows Azure необходимо иметь как минимум два экземпляра каждой роли. Дополнительные сведения см. на странице http://www.microsoft.com/windowsazure/sla/.

Оценка потребностей в хранилище SQL Azure

По оценкам Adatum, каждой записи о деловых расходах в базе данных потребуется 2 КБ памяти. Таким образом, исходя из анализа сложившегося использования (в среднем 60 % из 5 000 работников компании Adatum представляют 10 записей по деловым расходам в месяц) и

требования хранить данные в течение 7 лет, можно определить, что ожидаемая потребность хранения составит 4,8 ГБ.

Емкость хранилища SQL	долл./месяц	Итого в год
4,8 ГБ	$9.99 за первый ГБ + $3.996 за ГБ для следующих 4 ГБ	$ 311,69

На рис. 2 показана сводка по расчетным ежегодным эксплуатационным затратам на приложение aExpense, работающее в Windows Azure.

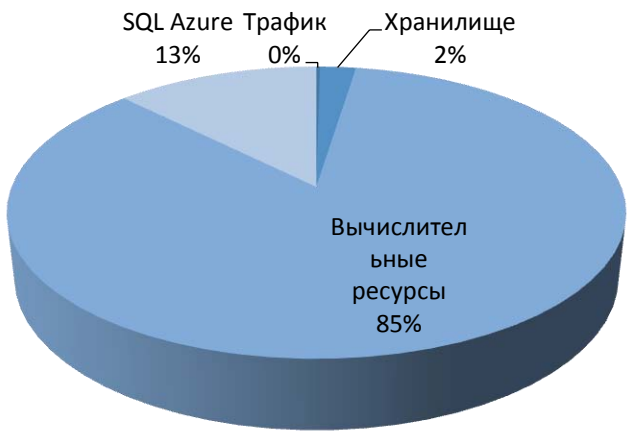


Рис. 2

Ежегодные эксплуатационные затраты на приложение aExpense

Варианты

Одним из вопросов, поднятых пользователями текущей версии приложения aExpense, является плохая производительность приложения в течение двух дней в конце месяца, когда приложение используется наиболее интенсивно. Для решения этой проблемы компания Adatum рассмотрела затраты на удвоение вычислительной мощности приложения на 2 дня в месяц путем добавления двух дополнительных веб-ролей.

Комментарий По:

Большинство пользователей жалуются на плохую производительность действующей версии приложения aExpense во время его интенсивного использования.

Часов/месяц	Часов/год	долл./час	Экземпляры роли	долл./год
48	576	$0,12	2	$138,24

Кроме того, компания Adatum рассмотрела финансовые последствия ограничения доступа к приложению периодом 12 часов в день на 6 дней в неделю.

Комментарий По:

Легко перенаправить пользователей на страницу «Приложение в данный момент недоступно» с помощью DNS.

Вычисление	Количество экземпляров роли	Часов/день	Дней в год	Долл./час	Долл./год
Стандартная нагрузка	2	12	313	$0,12	$901,44
Конец месяца	2	12	24	$0,12	$69,12
				Итого в год	$970,56

Это составляет около 50 % от вычисленных затрат на эксплуатацию приложения 24 часа в сутки 365 дней в году.

Кроме того, компания Adatum сравнила стоимость хранения данных по деловым расходам в хранилище таблиц Windows Azure и в SQL Azure. В следующей таблице предполагается, что потребность хранения данных в течение 7 лет составит те же 4,8 ГБ, что и для SQL Azure. Также предполагается, что доступ к каждой новой записи по деловым расходам происходит 5 раз в месяц.

Хранилище		Стоимость	Итого в месяц
Хранящиеся ГБ/месяц	4,8 ГБ	0,125 долл./ГБ	$0,60
Хранение транзакций/месяц	150 000	0,01 долл./10 К	$0,15
		Итого в год	$9,00

Дополнительные сведения

Последние сведения о ценах для Windows Azure см. на странице http://www.microsoft.com/windowsazure/pricing/.

Калькулятор стоимости Windows Azure находится на странице http://www.microsoft.com/windowsazure/pricing-calculator/.

Сведения структуре счета Windows Azure см. на странице http://www.microsoft.com/windowsazure/support/understandbill/.

Глава 5. Автоматизация развертывания и использование хранилища Windows Azure (этап 2)

В этой главе рассматриваются внесенные компанией Adatum изменения в приложение aExpense на втором этапе проекта. В ней описывается, как в компании Adatum расширили процесс сборки для приложения aExpense, который теперь включает этап развертывания упакованного приложения (пакета) непосредственно на технологической платформе Windows ® Azure ™. А также изменения, которые в Adatum внесли в приложение aExpense, чтобы использовать хранилище таблиц Windows Azure вместо SQL Azure, и то, как разработчики решали некоторые возникшие проблемы. Функциональность приложения с точки зрения пользователей по сравнению с предыдущим этапом не изменилась.

Исходные условия

В конце первого этапа проекта у компании Adatum появилась версия приложения aExpense для работы в облаке. Во время разработки этой версии разработчики максимально сохранили исходное приложение, изменив только то, что было необходимо для работы в Windows Azure.

Большую часть тестирования приложения команда разработчиков выполняет с помощью эмулятора вычислений и эмулятора хранилища, что упрощает отладку ошибок в коде. Они также развертывают приложение в Windows Azure для проведения дополнительного тестирования в промежуточной среде облака. Разработчики обнаружили, что развертывание приложения в Windows Azure вручную через портал разработчиков Windows Azure чреват ошибками, в частности при редактировании файлов конфигурации и указании правильных строк подключения.

Примечание.

В главе 7, «Управление жизненным циклом приложений Windows Azure», более подробно рассматривается тестирование приложений для Windows Azure.

Простой анализ затрат для существующего решения показал, что на SQL Azure может составлять существенную эксплуатационных расходов на приложение. Поскольку стоимость использования хранилища таблиц Windows Azure гораздо меньше, чем стоимость использования SQL Azure, компания Adatum заинтересована в исследовании возможности использования хранилища таблиц Windows Azure.

Цели и требования

На этом этапе в компании Adatum преследуют две конкретные цели. Первая цель заключается в оценке возможности использования приложением aExpense хранилища таблиц Windows Azure вместо SQL Azure. Для компании крайне важна целостность данных, поэтому Adatum хочет использовать транзакции, когда пользователь отправляет несколько записей по деловым расходам в одном подтверждении расходов.

Необходимо оценить возможность использования в приложении хранилища таблиц Windows Azure вместо SQL Azure.

Вторая цель заключается в том, чтобы автоматизировать процесс развертывания в Windows Azure. По мере осуществления проекта компании Adatum потребуется возможность развертывания версий приложения aExpense в Windows Azure без необходимости изменения файлов конфигурации вручную или использования портала разработчиков Windows Azure. Это позволит произвести развертывание в Windows Azure с меньшим количеством ошибок и упростит выполнение в автоматизированной среде сборки.

Обзор решения

Переход от SQL Azure к хранилищу таблиц Windows Azure влечет за собой некоторые существенные изменения в логике приложения. Исходная версия приложения aExpense для взаимодействия с SQL Azure на уровне доступа к данным (DAL) использовала технологию LINQ to SQL. Уровень доступа к данным преобразовывал данные, полученные через LINQ to SQL, в набор объектов модели домена, которые передавались в пользовательский интерфейс. В новой версии приложения aExpense, в которой используется хранилище таблиц Windows Azure, для взаимодействия с таким хранилищем применяется клиентская библиотека .NET, являющаяся частью служб WCF Data Services.

Комментарий Маркуса:

Ранее службы WCF Data Services назывались службами ADO.NET Data Services.

Примечание.

Служба таблиц Windows Azure поддерживает только подмножество функций, определенных клиентской библиотекой .NET для служб WCF Data Services. Дополнительные сведения см. на странице http://msdn.microsoft.com/ru-ru/library/dd894032.aspx.

Данные по деловым расходам теперь упорядочены в двух таблицах Windows Azure, связанных отношением «родитель/потомок»: запись заголовка расходов и записи сведений о расходах.

На рис. 1 показана структура таблиц Windows Azure.

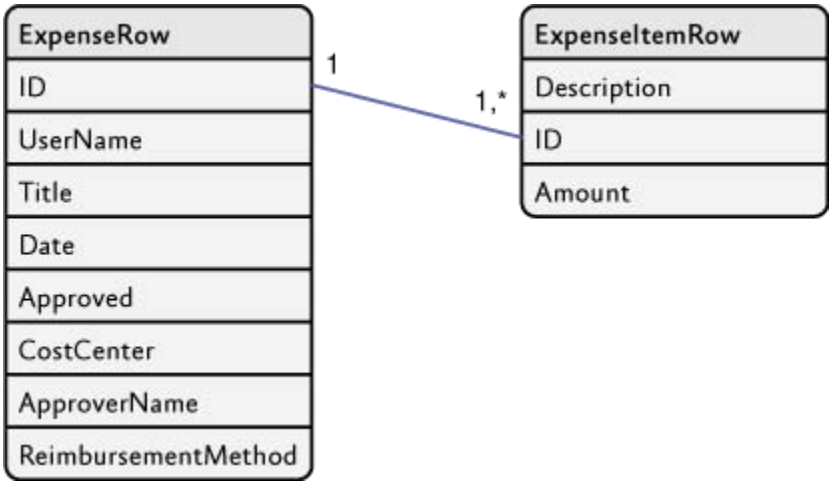


Рис. 1

Структура таблиц Windows Azure

Примечание.

В главе 8, «Этап 4. Добавление нескольких задач и настройка приложения», данного руководства рассматривается альтернативное решение, где ExpenseRows и ExpenseItemRows хранятся в одной таблице.

Разработчики компании Adatum обновили веб-интерфейс приложения aExpense для поддержки добавления нескольких записей о деловых расходах в одном подтверждении расходов. Приложение aExpense использует Entity Group Transactions для обеспечения целостности данных в приложении.

Комментарий Бхарата:

Entity Group Transactions можно использовать для группировки нескольких операций с сущностями в одной таблице и группе партиций в единичную неразрывную транзакцию.

Автоматизированное развертывание приложения aExpense происходит в два этапа. На первом этапе используется скрипт MSBuild для компиляции и упаковки приложения для развертывания в Windows Azure. Этот скрипт сборки использует пользовательскую задачу MSBuild для изменения файлов конфигурации для развертывания в облаке вместо локального развертывания эмулятора вычислений. На втором этапе для выполнения развертывания в Windows Azure используется скрипт PowerShell с некоторыми пользовательскими командлетами.

Комментарий По:

Автоматизация развертывания приложений в Windows Azure с помощью скриптов намного облегчает управление приложениями, выполняемыми в облаке.

Реализация

Теперь пришло время рассмотреть эти изменения более подробно. По мере изучения этого раздела может потребоваться загрузка решения для систем разработки Microsoft® Visual Studio® со страницы http://wag.codeplex.com/. Это решение (в папке Azure-AzureStorage) содержит реализацию приложения aExpense после изменений, выполненных на этом этапе. Если вы не заинтересованы в изучении таких механизмов, перейдите к следующему разделу.

Автоматизация развертывания в Windows Azure

Несмотря на то что не требуется вносить какие-либо изменения в код при развертывании в Windows Azure, а не в локальном эмуляторе вычислений, скорее всего, необходимо будет внести некоторые изменения в конфигурацию. Теперь приложение aExpense использует хранилище таблиц Windows Azure для хранения данных по деловым расходам, так что в файле ServiceConfiguration.csfg необходимо изменить параметры DataConnectionString и Microsoft.WindowsAzure.Plugins.Diagnostics.ConnectionString и указать данные используемой учетной записи хранилища Windows Azure.

Для развертывания в облаке требуется внести изменения в конфигурацию приложения.

Так выглядит соответствующий раздел файла конфигурации, когда приложение использует хранилище разработки.

Копировать

```
<ConfigurationSettings>
  <Setting name=
    "Microsoft.WindowsAzure.Plugins.Diagnostics.ConnectionString"
    value="UseDevelopmentStorage=true" />
  <Setting name="DataConnectionString"
    value="UseDevelopmentStorage=true" />
</ConfigurationSettings>
```

Так выглядит раздел, когда приложение использует хранилище в облаке.

Копировать

```
<ConfigurationSettings>
  <Setting name=
    "Microsoft.WindowsAzure.Plugins.Diagnostics.ConnectionString"
    value="DefaultEndpointsProtocol=https;
    AccountName={Azure storage account name};
    AccountKey={Общий ключ хранилища Azure}" />
```

```
 <Setting name="DataConnectionString"
  value="DefaultEndpointsProtocol=https;
  AccountName={Azure storage account name};
  AccountKey={Общий ключ хранилища Azure}" />
</ConfigurationSettings>
```

Примечание.

Значения Имя учетной записи хранилища Azure и Общий ключ хранилища Azure уникальны для конкретной учетной записи в хранилище Windows Azure.

Примечание.

Для изменения этих параметров вручную следует использовать средства в Visual Studio вместо редактирования XML напрямую.

Скрипт MSBuild для приложения aExpense применяет пользовательское задание сборки RegexReplace для внесения изменений в течение процесса сборки. В приведенном здесь примере строки подключения хранилища разработки заменены на строки подключения хранилища Windows Azure.

Комментарий Маркуса:

Также необходимо наличие цели (target), которая сбросит строки подключения к хранилищу разработки для локального тестирования.

Копировать

```
<Target Name="SetConnectionStrings" DependsOnTargets="BuildTasks">
 <RegexReplace
  Pattern='Setting name=
  "Microsoft.WindowsAzure.Plugins.Diagnostics.ConnectionString"
   value="UseDevelopmentStorage=true"'
  Replacement='Setting name=
  "Microsoft.WindowsAzure.Plugins.Diagnostics.ConnectionString"
   value="DefaultEndpointsProtocol=https;
   AccountName=$(StorageAccountName);
```

```
    AccountKey=$(StorageAccountKey) "'
  Files='$(AzureProjectPath)\$(ServiceConfigName)'/>
 <RegexReplace
  Pattern='Setting name="DataConnectionString"
    value="UseDevelopmentStorage=true"'
  Replacement='Setting name="DataConnectionString"
    value="DefaultEndpointsProtocol=https;
    AccountName=$(StorageAccountName);
    AccountKey=$(StorageAccountKey) "'
  Files='$(AzureProjectPath)\$(ServiceConfigName)'/>
</Target>
```

Примечание.

В загружаемом пакете для этого этапа имеется исходный код для пользовательского задания сборки RegexReplace.

Затем команда разработчиков Adatum разработала скрипт PowerShell (deploy.ps1), который развернет упакованное приложение в Windows Azure. Этот скрипт можно вызвать из задачи MSBuild. Он использует командлеты PowerShell платформы Windows Azure — библиотеку командлетов PowerShell, которая является оболочкой для программных интерфейсов управления и диагностики служб Windows Azure. Эту библиотеку можно скачать на странице http://wappowershell.codeplex.com/.

Копировать

```
$buildPath = $args[0]
$packagename = $args[1]
$serviceconfig = $args[2]
$servicename = $args[3]
$thumbprint = $args[4]
$cert = Get-Item cert:\CurrentUser\My\$thumbprint
$sub = $args[5]
$slot = $args[6]
$storage = $args[7]
$package = join-path $buildPath $packageName
```

```
$config = join-path $buildPath $serviceconfig
$a = Get-Date
$buildLabel = $a.ToShortDateString() + "-" + $a.ToShortTimeString()

#Внимание! При использовании файловых пакетов (пути без http)
#командлеты попробуют загрузить пакет в хранилище больших двоичных объектов
#автоматически.  Если не указан —
#параметр StorageServiceName, попытается загрузить учетную запись
#хранилища с тем же именем, что и $servicename.  Если
#такой учетной записи не существует, операция завершится ошибкой.  Это применяется исключительно для
#путей к файловым пакетам.

если ((Get-PSSnapin | ?{$_.Name -eq "AzureManagementToolsSnapIn"})
  -eq $null)
{
 Add-PSSnapin AzureManagementToolsSnapIn
}

$hostedService = Get-HostedService $servicename -Certificate
 $cert -SubscriptionId $sub | Get-Deployment -Slot $slot

если ($hostedService.Status -ne $null)
{
   $hostedService |
    Set-DeploymentStatus 'Suspended' |
    Get-OperationStatus -WaitToComplete
   $hostedService |
    Remove-Deployment |
```

```
    Get-OperationStatus -WaitToComplete
}
Get-HostedService -ServiceName $servicename -Certificate $cert
   -SubscriptionId $sub | New-Deployment -Slot $slot
   -Package $package -Configuration $config -Label $buildLabel
   -ServiceName $servicename -StorageServiceName $storage |
   Get-OperationStatus -WaitToComplete

Get-HostedService -ServiceName $servicename -Certificate $cert
   -SubscriptionId $sub |
   Get-Deployment -Slot $slot |
   Set-DeploymentStatus 'Running' |
   Get-OperationStatus -WaitToComplete
```

Этому скрипту требуются следующие два фрагмента данных для подключения к Windows Azure. Необходимо заменить сертификат программного интерфейса учетной записи и идентификатор подписки учетной записи на значения для своей учетной записи Windows Azure:

Отпечаток (thumbprint) сертификата программного интерфейса, установленного на локальном компьютере. Этот сертификат должен соответствовать сертификату, загруженному на портал разработчиков Windows Azure на странице «Учетная запись».

Свой идентификатор подписки можно посмотреть на странице «Учетная запись» портала Windows Azure.

Примечание.

API для управления сертификатами предоставляет доступ ко всем функциям программного интерфейса управления службами Windows Azure. Возможно, потребуется ограничить доступ к этому скрипту для того, чтобы доступ к службам Windows Azure имели только авторизованные пользователи. Этот сертификат отличается от SSL-сертификата, используемого конечной точкой HTTPS.

Скрипт развертывания имеет четыре параметра.

- buildPath. Этот параметр определяет папку, где будет создан пакет развертывания. Например: C:\AExpenseBuildPath.

- packagename. Этот параметр определяет пакет для передачи в Windows Azure. Например: aExpense.Azure.cspkg.
- serviceconfig. Этот параметр определяет файл конфигурации службы для проекта. Например: ServiceConfiguration.cscfg.
- servicename. Этот параметр определяет имя службы, размещенной в Windows Azure. Например: aExpense.

Сначала скрипт проверяет, загружены ли командлеты управления службами Windows Azure. Затем при наличии существующей службы скрипт приостанавливает и удаляет ее. После этого скрипт выполняет развертывание и запускает новую версию службы.

Комментарий По:

В этом примере демонстрируется развертывание приложения aExpense в staging среде. Можно легко изменить скрипты для развертывания в производственной среде. Кроме того, можно обновить скрипт на месте при наличии нескольких экземпляров роли.

MSBuild может использовать скрипт PowerShell в задаче и передавать все необходимые значения параметров:

Копировать

```
<Target Name="Deploy"
 DependsOnTargets="SetConnectionStrings;Build">
 <MSBuild
  Projects="$(AzureProjectPath)\$(AzureProjectName)"
  Targets="CorePublish"
  Properties="Configuration=$(BuildType)"/>

 <Exec WorkingDirectory="$(MSBuildProjectDirectory)"
  Command=
  "$(windir)\system32\WindowsPowerShell\v1.0\powershell.exe

  -NoProfile -f deploy.ps1 $(PackageLocation) $(PackageName)
  $(ServiceConfigName) $(HostedServiceName)
  $(ApiCertThumbprint) $(SubscriptionKey) $(HostSlot)
  $(StorageAccountName)" />
</Target>
```

Примечание.

Сведения об использовании скриптов PowerShell см. в заметках о выпуске в составе примеров.

В приложение aExpense используется конечная точка HTTPS, поэтому при выполнении автоматического развертывания Adatum нужно передать необходимый сертификат. Следующий скрипт PowerShell, deploycert.ps1, выполняет данную операцию.

Копировать

```
$servicename = $args[0]

$certToDeploy = $args[1]

$certPassword = $args[2]

$thumbprint = $args[3]

$cert = Get-Item cert:\CurrentUser\My\$thumbprint

$sub = $args[4]

$algo = $args[5]

$certThumb = $args[6]



если ((Get-PSSnapin | ?{$_.Name -eq "AzureManagementToolsSnapIn"})

   -eq $null)

{

 Add-PSSnapin AzureManagementToolsSnapIn

}


try

{

 Remove-Certificate -ServiceName $servicename

   -ThumbprintAlgorithm $algo -Thumbprint $certThumb

   -Certificate $cert -SubscriptionId $sub

}

catch {}
```

Add-Certificate -ServiceName $servicename -Certificate $cert

-SubscriptionId $sub -CertificateToDeploy $certToDeploy

-Password $certPassword

Этому скрипту требуются следующие два фрагмента данных для подключения к Windows Azure. Необходимо заменить Account API Certificate и Account Subscription ID на значения для своей учетной записи Windows Azure:

Отпечаток API сертификата, установленного на локальном компьютере. Этот сертификат должен соответствовать сертификату, который был передан на портал Windows Azure на странице «Учетная запись». Дополнительные сведения см. на странице «Сведения о программном интерфейсе управления службами» по адресу http://msdn.microsoft.com/ru-ru/library/ee460807.aspx.

Свой идентификатор подписки можно посмотреть на странице «Учетная запись» портала Windows Azure.

Необходимо также передать в скрипт три параметра:

- servicename. Этот параметр указывает имя службы Windows Azure. Например: aExpense.
- certToDeploy. Этот параметр указывает полный путь к PFX-файлу, содержащему сертификат.
- certPassword. Этот параметр указывает пароль, защищающий закрытый ключ в PFX-файле.

Примечание.
Приложения Windows Azure требуют создания и установки сертификатов для проверки подлинности и подписи. Сведения о типах сертификатов и их создании см. на страницах «Как создать сертификат для роли» по адресу http://msdn.microsoft.com/ru-ru/library/gg432987.aspx и «Как управлять сертификатами службы» по адресу http://msdn.microsoft.com/ru-ru/library/gg551727.aspx.

Примечание.
Скрипт не проверяет, развернут ли уже этот сертификат. Если это так, то скрипт выполнится без ошибок.

Файл MSBuild может вызвать этот скрипт, передав ему необходимые параметры. Следующий код является примером цели из файла MSBuild.

Копировать

<Target Name="DeployCert">

```
 <Exec WorkingDirectory="$(MSBuildProjectDirectory)"
  Command=
   "$(windir)\system32\WindowsPowerShell\v1.0\powershell.exe
   -f deploycert.ps1 $(HostedServiceName) $(CertLocation)
   $(CertPassword) $(ApiCertThumbprint) $(SubscriptionKey)
   $(DeployCertAlgorithm) $(DeployCertThumbprint)" />
</Target>
```

Хранение данных по деловым расходам в хранилище таблиц Windows Azure

Изменение приложения aExpense для использования хранилища таблиц Windows Azure вместо SQL Azure потребовало от разработчиков Adatum заново реализовать в приложении уровень доступа к данным. Поскольку в хранилище таблиц Windows Azure используется принципиально другой подход к хранению данных, подобная модификация не сводилась к простой замене LINQ to SQL на клиентскую библиотеку .NET.

Комментарий Яны:

Сохранение всего кода доступа к данным на уровне доступа к данным ограничивает область изменений кода, необходимых для смены хранилища (код придется менять только на уровне доступа к данным).
Доступ к хранилищу таблиц Windows Azure производится через клиентскую библиотеку .NET.

Сколько таблиц?

Самое главное, что необходимо понимать при переходе на хранилище таблиц Windows Azure, — это то, что его модель хранения, вероятно, отличается от привычной. В реляционном мире очевидная модель данных для aExpense содержала бы две таблицы: одну для хранения заголовков сущностей расходов и другую для сущностей подробных описаний расходов, а ограничение внешнего ключа обеспечивало бы целостность данных. По ряду причин лучший выбор модели данных для хранилища таблиц Windows Azure будет не столь очевиден:

Комментарий Бхарата:

Имея дело с хранилищем таблиц Windows Azure, нужно перестать мыслить реляционными категориями.

В одной таблице Windows Azure могут храниться сущности различных типов.

Транзакции с группами сущностей ограничены одной партицией в одной таблице (партиции более подробно обсуждаются ниже в этой главе).

Хранилище таблиц Windows Azure относительно недорого, поэтому нет необходимости заботиться о нормализации данных и устранять их избыточность.

Разработчики Adatum могли бы использовать одну таблицу для хранения сущностей заголовков и сущностей деталей расходов. Такой подход позволил бы использовать в Adatum транзакцию с группой сущностей для сохранения в одной партиции сущности заголовка о расходах вместе со связанными сущностями деталей за одну атомарную транзакцию. Однако хранение нескольких типов сущностей в одной таблице увеличивает сложность приложения.

Примечание.
В примере приложения используется решение с единственной таблицей, поэтому примеры кода в этой главе могут не полностью соответствовать коду в загруженном решении.
О том, как хранить несколько типов сущностей в одной таблице, можно узнать в главе 8, «Этап 4. Добавление других задач и настройка приложения», данного руководства.

Разработчики Adatum решили создать решение aExpense с двумя таблицами, которые будут называться Expense и ExpenseItem. Определения таблиц можно посмотреть в классах ExpenseRow и ExpenseItemRow в пространстве имен AExpense.DataAccessLayer. Такой подход позволяет избежать сложностей, связанных с разными типами сущностей в одной таблице, но становится сложнее гарантировать целостность данных, так как сущность заголовка не может быть сохранена в той же транзакции, что и сущности деталей. Как в Adatum решили эту проблему, описано далее в этой главе в подразделе «Транзакции в aExpense».

Примечание.
Разработчикам Adatum пришлось внести изменения в тип данных, который приложение использует для хранения сумм деловых расходов. В SQL Azure это поле хранилось в виде десятичного числа. Такой тип данных не поддерживается в хранилище таблиц Windows Azure, поэтому суммы теперь хранятся в виде действительного числа с двойной точностью.

Ключи партиций и ключи строк

Второе важное решение, касающееся хранилища таблиц, — это выбор используемых ключей.
В хранилище таблиц Windows Azure используются два типа ключей: ключ партиции и ключ строки. Windows Azure использует ключ партиции для реализации балансировки нагрузки на узле хранилища. Подсистема балансировки загрузки способна выявлять «горячие» партиции (содержащие данные, доступ к которым производится чаще, чем к данным других партиций) и выполнять их в отдельных узлах хранилища для повышения производительности. Это серьезно влияет на процесс создания модели данных и выбор ключей партиции.

Комментарий Яны:

Правильный выбор ключа партиции является наиболее важным решением, которое влияет на производительность решения хранения данных.
Ключ партиции и ключ строки в совокупности образуют кортеж, который

уникальным образом идентифицирует любую сущность в хранилище.

Ключ партиции образует первую часть кортежа, которая уникальным образом идентифицирует сущность в хранилище таблиц.

Транзакции с группами сущностей могут быть использованы только для тех сущностей, которые содержатся в одной таблице и одной партиции. Выбор ключа партиции может потребоваться исходя из требований приложения к обработке транзакций. Не забывайте, что в таблице могут храниться различные типы сущностей.

Комментарий Бхарата:

Каждая запись в таблице — это просто контейнер свойств. Каждый контейнер свойств может представлять свой тип сущности. Это означает, что в одной партиции может храниться несколько сущностей одного или разных типов.

Можно оптимизировать запросы исходя из своих знаний о ключах партиций. Например, если известно, что все сущности, которые необходимо получить, расположены в одной партиции, то можно включить ключ партции в блок where запроса. Если сущности, которые необходимо получить, расположены в разных партициях, то запрос можно разбить на несколько запросов и выполнять их параллельно.

Примечание.

Если возникла необходимость создания параллельных запросов, то вместо создания собственных потоков в веб-роли можно воспользоваться параллельным LINQ (PLINQ).

Ключ строки — это уникальный идентификатор для сущности в пределах партиции. Он образует вторую часть кортежа, который уникальным образом идентифицирует сущность в хранилище таблиц.

В приложении aExpense разработчики Adatum решили воспользоваться свойством UserName в качестве ключа партции в таблице Expense. Они полагают, что подавляющее большинство запросов будет фильтроваться по свойству UserName. Например, на веб-сайте отображаются подтверждения расходов текущего пользователя, выполнившего вход. В таблице ExpenseItem используется тот же ключ партиции. Это означает, что, когда пользователь вставляет несколько сущностей ExpenseItem одновременно, приложение может использовать транзакцию для обеспечения гарантированной вставки всех элементов ExpenseItem (или ни одного из них). Это также означает, что все данные, принадлежащие пользователю, расположены в той же партиции, поэтому для получения пользовательских данных необходимо просмотреть только ее.

Комментарий Яны:

Хотя в таблицах Expense и ExpenseItem используются одинаковые значения ключа партиции, нельзя выполнить такую транзакцию, которая охватывает несколько таблиц в хранилище таблиц Windows Azure.

Примечание.

В главе 8, «Этап 4. Добавление нескольких задач и настройка приложения», выбор ключа партиции и ключа строки пересматривается по результатам тестирования приложения aExpense.

Чтобы обеспечить уникальность значения, приложение использует идентификатор GUID в качестве ключа строки таблицы Expense. Поскольку таблицы Windows Azure не поддерживают внешние ключи, для таблицы ExpenseItem ключом строки будет объединение ключа строки родительской сущности в таблице Expense и идентификатора GUID строки ExpenseItem. Это позволяет приложению фильтровать сущности ExpenseItem по идентификатору ExpenseID как по внешнему ключу. Следующий код в методе SaveChanges класса ExpenseRepository показывает, как приложение создает это значение ключа строки по свойству Id сущности заголовка расходов и свойству Id сущности деталей расходов.

Копировать

```
expenseItemRow.RowKey = string.Format(
   CultureInfo.InvariantCulture,
   "{0}_{1}", expense.Id, expenseItemRow.Id);
```

В следующем примере кода показано, как запросить ExpenseItems по значению ExpenseID.

Комментарий Маркуса:

Более естественно написать этот запрос будет с использованием StartsWith вместо CompareTo. Но параметр StartsWith не поддерживается службами таблиц Windows Azure. Такой запрос позволяет также получить выигрыш в производительности, так как предложение where включает ключ партиции.

Копировать

```
char charAfterSeparator =
   Convert.ToChar((Convert.ToInt32('_') + 1));
var nextId = expenseId.ToString() + charAfterSeparator;


var expenseItemQuery =
   (from expenseItem in context.ExpenseItem
```

```
    где
    expenseItem.RowKey.CompareTo(expenseId.ToString()) >= 0 &&
    expenseItem.RowKey.CompareTo(nextId) < 0 &&
    expenseItem.PartitionKey.CompareTo(expenseRow.PartitionKey)
      == 0
    select expenseItem).AsTableServiceQuery();
```

Windows Azure накладывает некоторые ограничения на набор символов, которые могут быть использованы в ключах партиций и строк. В общем, это те же ограничения, что и при построении URL-адресов. Дополнительные сведения см. на странице http://msdn.microsoft.com/ru-ru/library/dd179338.aspx. В приложении aExpense может оказаться, что недопустимые символы появятся в UserName, используемом в качестве значения партиции в таблице Expense.

Примечание.
Если в ключе партиции имеется недопустимый символ, то Windows Azure вернет сообщение «400 - Неверный запрос».

Чтобы избежать этой проблемы, приложение aExpense переводит значение UserName в кодировку Base64 перед использованием значения UserName в качестве ключа строки. Реализация кодирования и декодирования Base64 очень проста.

Копировать

```
public static string EncodePartitionAndRowKey(string key)
{
   if (key == null)
   {
      return null;
   }

   return Convert.ToBase64String(
      System.Text.Encoding.UTF8.GetBytes(key));
}

public static string DecodePartitionAndRowKey(string encodedKey)
```

```
{
   if (encodedKey == null)
   {
      return null;
   }

   return System.Text.Encoding.UTF8.GetString(
      Convert.FromBase64String(encodedKey));
}
```

Сначала разработчики Adatum пытались воспользоваться методом UrlEncode, так как он выдавал более понятный результат, но этот способ не подошел, поскольку метод не кодирует символ процента (%).

Примечание.
По документации символ процента не относится к недопустимым в ключах символам, однако проведенные разработчиками Adatum тесты показали, что сущности, ключи которых содержат символ процента, нельзя удалить.

Другим вариантом станет реализация пользовательского метода экранирования.

Комментарий Маркуса:

Пользовательский метод преобразования имени пользователя в допустимую последовательность символов поможет сделать ключи понятными для человека, и это обязательно пригодится при отладке и диагностике.

Производительность запроса

Как уже отмечалось, выбор ключа партиции может оказать большое влияние на производительность работы приложения. Причина этого заключается в том, что подсистема балансировки нагрузки Windows Azure отслеживает активность на уровне для повышения эффективности доступа к данным.

Можно также использовать ключи партиции для повышения производительности отдельных запросов. С помощью следующего запроса текущая версия приложения извлекает сохраненные подтверждения деловых расходов пользователя.

Копировать

```
var query = (from expense in context.Expenses
             where expense.UserName.CompareTo(userName) == 0
```

```
        select expense).AsTableServiceQuery();
```

В таком виде этот запрос должен просмотреть все партции таблицы, чтобы найти подходящие записи. Такой подход неэффективен, когда имеется большое число записей, и производительность еще более ухудшится, если необходимо последовательно просмотреть данные в нескольких узлах хранилища.

Комментарий Яны:

Важно понимать, какое влияние партиции могут оказать на производительность запроса.

Во время стресс-тестирования приложения разработчики Adatum пытаются выяснить, обеспечит ли изменение запроса с указанием ключа партиции в блоке where значительное повышение производительности работы приложения aExpense.

Копировать

```
var query = (from expense in context.Expenses
        where expense.PartitionKey.CompareTo(
          EncodePartitionAndRowKey(userName)) == 0
        select expense).AsTableServiceQuery();
```

Примечание.
Если запрос к таблице не содержит ключа партиции в предложении where, то, чтобы избежать возможных проблем производительности, связанных с просмотром нескольких партиций, необходимо внести изменения в выбор ключей партиции и строк для этой таблицы.

Транзакции в aExpense

Сохранение подтверждения деловых расходов

Разработчики Adatum решили хранить сущности Expense и ExpenseItem в двух разных таблицах Windows Azure, однако это означает, что тогда нельзя будет поместить всю операцию SaveExpense в одну транзакцию, чтобы обеспечить целостность данных. В следующем коде показано гибридное решение, найденное разработчиками Adatum, где для обеспечения целостности данных используются оба кода, транзакционный и компенсационный.

В хранилище таблиц Windows Azure транзакция не может охватывать несколько таблиц.

Примечание.
О том, как хранить разные типы сущностей в одной таблице и упростить, таким образом, обработку транзакций, см. в главе 8, «Этап 4. Добавление других задач и настройка приложения», данного руководства.

Копировать

```
public void SaveExpense(Expense expense)
{
   ExpenseDataContext context = new
      ExpenseDataContext(this.account);
   ExpenseRow expenseRow = expense.ToTableEntity();

   foreach (var expenseItem in expense.Details)
   {
      var expenseItemRow = expenseItem.ToTableEntity();
      expenseItemRow.PartitionKey = expenseRow.PartitionKey;
      expenseItemRow.RowKey =
        string.Format(CultureInfo.InvariantCulture,
        "{0}_{1}", expense.Id, expenseItemRow.Id);
      context.AddObject(ExpenseDataContext.ExpenseItemTable,
         expenseItemRow);
   }

   context.SaveChanges(SaveChangesOptions.Batch);

   context.AddObject(ExpenseDataContext.ExpenseTable,
      expenseRow);
   context.SaveChanges();
}
```

В примере кода не включена логика повторения при завершении вызова метода SaveChanges ошибкой. В статье описано, как реализовать эту логику с помощью делегата RetryPolicy:

http://blogs.msdn.com/windowsazurestorage/archive/2010/04/22/savechangeswithretries-and-batch-option.aspx, когда и как часто выполнять повторение, какие ошибки указывают на то, что повторять

не нужно. В ней также описано решение проблемы с правилами повторения и транзакциями.

Транзакция применяется (неявным образом) в службе хранилища таблиц Windows Azure. Код в этом методе сначала добавляет несколько сущностей ExpenseItem в контекст, а затем вызывает метод SaveChanges. Метод SaveChanges имеет параметр SaveChangesOptions.Batch, который предписывает контексту зафиксировать все ожидающие изменения в одном запросе. Если все изменения в пакете относятся к сущностям из одной партиции одной таблицы, то Windows Azure при сохранении изменений в хранилище таблиц автоматически использует транзакцию с группами сущностей. Транзакция с группами сущностей при выполнении операции обеспечивает стандартный подход, при котором либо выполняются все действия внутри нее, либо не выполняется ни одно.

Комментарий Маркуса:

Сейчас мы исходим из того, что никто в одном представлении не будет отправлять более 100 подтверждений деловых расходов. Нам необходимо добавить дополнительные проверки в код, чтобы гарантировать возможность использования транзакции с группами сущностей в методе SaveChanges.

Существуют некоторые дополнительные ограничения при работе транзакций с группами сущностей:

каждая сущность может появляться в транзакции только один раз, в транзакции должно обрабатываться не более 100 сущностей, а общий размер полезной нагрузки запроса не должен превышать 4 МБ.

После вставки сущностей подробностей расходов ExpenseItem код сохраняет сущность заголовка ExpenseRow. Смысл сохранения подробностей первыми состоит в том, чтобы в случае ошибки при их сохранении в пользовательском интерфейсе не отобразился бы потерянный заголовок.

Чтобы разрешить потенциальную проблему существования потерянных подробностей, в Adatum планируют реализовать процесс «сборщика потерянных сущностей» (orphan collector), который будет регулярно просматривать таблицу Expense в поисках потерянных записей ExpenseItem и их удаления.

Работа с эмулятором хранилища

Существуют некоторые различия между эмулятором хранилища таблиц и хранилищем таблиц Windows Azure, документация по которому находится на странице http://msdn.microsoft.com/ru-ru/library/dd320275.aspx. Разработчики Adatum столкнулись с ошибкой «Одно из входных значений запроса недопустимо» («One of the request inputs is not valid»), которая возникает при тестировании приложения с пустыми таблицами в эмуляторе хранилища. Поэтому в Adatum было решено вставлять в таблицы Windows Azure фиктивную строку, а затем удалять ее, если приложение использует локальный эмулятор хранилища. Во время инициализации веб-роли приложение вызывает метод расширения CreateTableIfNotExist<T> класса TableStorageExtensionMethods, чтобы проверить, работает ли оно с эмулятором хранилища, и, если это так, добавляет фиктивные строки в таблицы Windows Azure приложения, а затем удаляет их.

Комментарий Маркуса:

Не следует предполагать, что эмулятор хранилища будет работать в точности так же, как хранилище Windows Azure.

Необходимо предусмотреть добавление фиктивных записей ко всем таблицам эмуляторе хранилища.

В следующем коде из класса TableStorageExtensionMethods показано, каким образом приложение aExpense определяет, используется ли эмулятор хранилища, а также как добавляется и удаляется фиктивная запись в таблице.

Копировать

```
public static bool CreateTableIfNotExist<T>(
   this CloudTableClient tableStorage, string entityName)
   where T : TableServiceEntity, new()
{
   bool result = tableStorage.CreateTableIfNotExist(entityName);

   // Выполняется только в хранилище разработки
   if (tableStorage.BaseUri.IsLoopback)
   {
      InitializeTableSchemaFromEntity(tableStorage,
         entityName, new T());
   }
   return result;
}

private static void InitializeTableSchemaFromEntity(
   CloudTableClient tableStorage, string entityName,
   TableServiceEntity entity)
{
   TableServiceContext context =
      tableStorage.GetDataServiceContext();
```

```
    DateTime now = DateTime.UtcNow;
    entity.PartitionKey = Guid.NewGuid().ToString();
    entity.RowKey = Guid.NewGuid().ToString();
    Array.ForEach(
      entity.GetType().GetProperties(BindingFlags.Public |
      BindingFlags.Instance),
      p =>
      {
        if ((p.Name != "PartitionKey") &&
            (p.Name != "RowKey") && (p.Name != "Timestamp"))
        {
          if (p.PropertyType == typeof(string))
          {
            p.SetValue(entity, Guid.NewGuid().ToString(),
              null);
          }
          else if (p.PropertyType == typeof(DateTime))
          {
            p.SetValue(entity, now, null);
          }
        }
      });

    context.AddObject(entityName, entity);
    context.SaveChangesWithRetries();
    context.DeleteObject(entity);
    context.SaveChangesWithRetries();
}
```

Извлечение данных из хранилища таблиц

Приложение aExpense с помощью LINQ указывает, какие данные необходимо извлечь из хранилища таблиц. В следующем примере кода показано, как приложение извлекает подтверждения расходов для утверждения по имени утверждающего.

Метод AsTableServiceQuery возвращает данные из хранилища таблиц Windows Azure.

Копировать

```
var query = (from expense in context.Expenses
   where expense.ApproverName.CompareTo(approverName) == 0
   select expense).AsTableServiceQuery();
```

Метод AsTableServiceQuery преобразует стандартный результат типа IQueryable в результат типа CloudTableQuery. Использование объекта CloudTableQuery дает приложению следующие преимущества:

Данные из таблицы можно получить несколькими сегментами, а не все разом. Это может оказаться полезным при обработке больших наборов данных.

Можно задать политику повторений для тех случаев, когда запрос завершился ошибкой.

Преобразование сущностей

В приложении aExpense все методы класса ExpenseRepository, возвращающие данные из запросов, вызывают метод ToList перед возвратом результатов вызывающему коду.

Комментарий Яны:

Попробуйте обрабатывать все проблемы доступа к данным на уровне доступа к данным.

Копировать

```
public IEnumerable<Expense> GetExpensesForApproval(string approverName)
{
   ExpenseDataContext context = new
      ExpenseDataContext(this.account);

   var query = (from expense in context.Expenses
            where expense.ApproverName.CompareTo(approverName) == 0
            select expense).AsTableServiceQuery();
```

```
    try
    {
        return query.Execute().Select(e => e.ToModel()).ToList();
    }
    catch (InvalidOperationException)
    {
        Log.Write(EventKind.Error,
          "Вызовом ToList() это исключение можно обработать
           внутри хранилища.");
        throw;
    }
}
```

Причина этого заключается в том, что вызов метода Execute преобразует сущности в объекты. Преобразование не выполняется до тех пор, пока не будет вызван метод MoveNext коллекции IEnumerable. Без вызова метода ToList первый вызов метода MoveNext производится за пределами репозитория. Вызвать сначала метод MoveNext внутри класса ExpenseRepository лучше тем, что все исключения доступа к данным можно будет обработать внутри репозитория.

Комментарий Маркуса:

Сохранение функций доступа к данным в пределах уровня доступа к данным помогает отделить логику приложения и локализовать взаимодействие с облаком в небольшой части приложения, что упростит отладку и диагностику.

Дополнительные сведения

Последние версии инструментов Windows Azure для среды Microsoft Visual Studio и пакет SDK для Windows Azure можно скачать в центре загрузки Microsoft.

В MSDN® содержится много сведений о хранилище таблиц Windows Azure. Рекомендуется начать со страницы http://msdn.microsoft.com/ru-ru/library/dd179423.aspx.

Документ «Таблицы Windows Azure» по адресу http://go.microsoft.com/fwlink/?LinkId=153401 содержит подробные сведения о работе с хранилищем таблиц Windows Azure.

Глава 6. Передача изображений и добавление рабочей роли (этап 3)

Эта глава содержит обзор изменений в версии приложения aExpense для облака, которые были сделаны разработчиками Adatum для поддержки передачи, хранения и отображения

отсканированных изображений чеков. Вы увидите, как приложение использует хранилище больших двоичных объектов платформы Windows® Azure™ для хранения графических данных, рабочую роль Windows Azure для выполнения фоновой обработки изображений и как приложение использует Shared Access Signature для управления доступом к изображениям. В этой главе также представлен простой набор абстракций, выступающих в качестве оболочки для рабочей роли, в предположении, что в приложение aExpense будут добавлены дополнительные фоновые задачи в будущем.

Исходные условия

На этом этапе проекта разработчики Adatum обратили внимание на первый из фоновых процессов приложения aExpense, который выполняет некоторую обработку отсканированных изображений чеков оплаты деловых расходов, передаваемых пользователями.

Локальное веб-приложение позволяет пользователям передавать изображения чеков оплаты деловых расходов и присваивает каждому из них уникальное имя, а затем сохраняет изображение в общей папке. Затем путь к этому изображению сохраняется в базе данных SQL Server®, где хранятся данные по деловым расходам. Впоследствии приложение может извлечь изображение, связанное с данным подтверждением расходов, и отобразить его в пользовательском интерфейсе.

Готовое локальное приложение также имеет фоновый процесс, который занимается обработкой изображений и реализован в виде службы Windows. Этот процесс выполняет две задачи: во-первых, он сжимает изображения, чтобы они занимали меньше места на диске, а во-вторых, создает эскиз изображения. По умолчанию в пользовательском интерфейсе отображается эскиз, но при необходимости пользователь может вывести изображение в полном размере.

Цели и требования

Перед разработчиками Adatum стоит ряд целей по реализации в приложении компонента обработки изображений. Во-первых, необходимо свести к минимуму место, занимаемое изображениями, сохранив при этом читаемость данных на чеках оплаты.

В ежемесячную стоимость эксплуатации приложения будут входить затраты на хранение отсканированных чеков.

Кроме этого, необходимо сохранить малое время отклика приложения и свести к минимуму полосу пропускания, необходимую пользовательскому интерфейсу. После передачи изображения пользователь не должен ждать завершения какой-либо обработки. Кроме того, приложение должно отображать эскизы с возможностью показа полноразмерной версии изображения.

Наконец, разработчикам Adatum нужно сохранить конфиденциальность своих сотрудников, чтобы чеки были доступны только тому работнику, который их загрузил, а также лицу, утверждающему эти расходы.

Комментарий Яны:

Мы должны сохранить конфиденциальность пользователей данного приложения.

Обзор решения

На этом этапе разработчики Adatum внесли несколько значительных изменений в реализацию версии приложения aExpense для Windows Azure. Первое решение — выбрать механизм хранения для отсканированных чеков. Простейшим подходом мог бы стать выбор диска Windows Azure. Это потребовало бы внесения в код минимальных изменений по сравнению с локальной версией, так как диск Windows Azure — это просто том NTFS, который доступен через стандартные классы ввода-вывода .NET. Такой подход имеет большой недостаток — на диск Windows Azure можно записывать данные только от одного экземпляра роли одновременно. Разработчики Adatum планируют развернуть несколько экземпляров веб-роли aExpense, чтобы обеспечить высокий уровень доступности и масштабируемость приложения. Принято решение хранить графические данные в хранилище больших двоичных объектов Windows Azure. Хотя такой подход потребует внесения больших изменений в код, он позволит обеспечить работу нескольких экземпляров роли.

Принято решение в версии для облака реализовать службу обработки изображений с помощью рабочей роли. Большая часть кода существующей локальной версии приложения, который сжимает изображения и создает эскизы, была просто перенесена в рабочую роль. Применение рабочей роли для обработки изображений разгрузит веб-роль приложения и поможет снизить время отклика пользовательского интерфейса, выполняя обработку изображений фоновым потоком. Что изменилось — это способ определения нового изображения, которое требует обработки. На рис. 1 показано, как в последней локальной версии служба Windows использует класс FileSystemWatcher для создания события сохранения приложением нового изображения в общей папке. Затем приложение aExpense вызывает код обработки изображения из обработчика события OnCreated.

Рабочая роль — естественный способ реализации фоновых процессов в Windows Azure.

Комментарий Бхарата:

Теперь разработчикам Adatum следует пересмотреть оценку затрат, так как в приложении aExpense появилась рабочая роль.

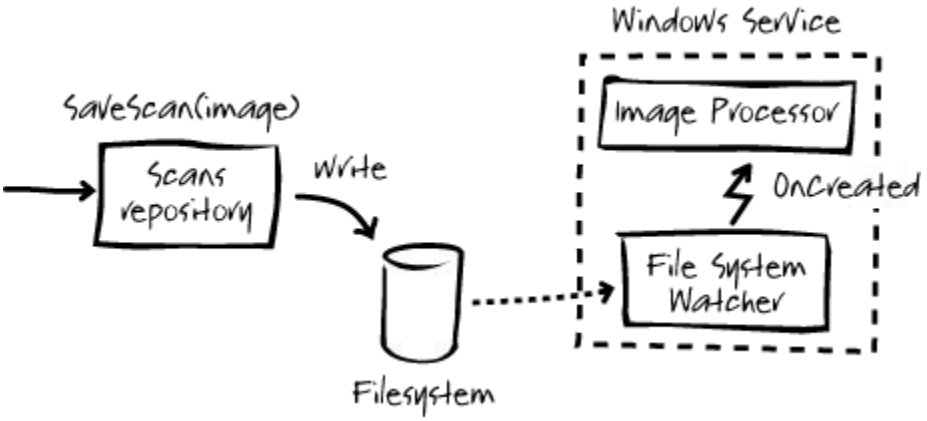


Рис. 1

Локальная обработка изображений

В версии приложения для облака, использующей хранилище больших двоичных объектов, такой подход не годится, так как в Windows Azure не существует аналога класса FileSystemWatcher для хранилища больших двоичных объектов. Вместо этого компания Adatum решила использовать очередь Windows Azure, как показано на рис. 2.

Комментарий Маркуса:

Мы можем использовать очередь Windows Azure для передачи данных от веб-роли к рабочей роли.

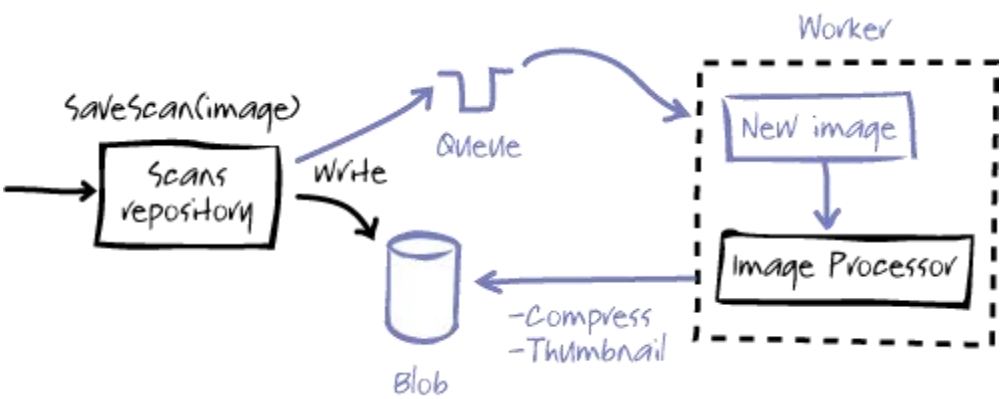


Рис. 2

Обработка изображений на основе облака

Когда пользователь передает в aExpense новое изображение в рамках представления по расходам, приложение записывает сообщение в очередь и сохраняет изображение в хранилище больших двоичных объектов. Рабочая роль извлекает сообщения из очереди, сжимает изображение, а затем создает эскиз. Рабочая роль сохраняет новое изображение и эскиз в хранилище больших двоичных объектов. После того как рабочая роль завершила обработку изображения, она обновляет сущность сведений о расходах в хранилище таблиц Windows Azure, указывая в ней ссылки на изображение и эскиз, а исходное изображение удаляет.

Чтобы изображения отображались в пользовательском интерфейсе, приложение находит их в хранилище больших двоичных объектов по данным в сущности сведений о расходах.

Из-за особенностей очередей Windows Azure сообщение может быть считано дважды, одной и или разными рабочими ролями. Это может приводить к ненужной обработке, однако учитывая, что операция обработки сообщения идемпотентна, она не повлияет на целостность данных приложения. В приложении aExpense дублирование сообщения приводят к тому что, что рабочая роль повторно выполняет изменение размера изображения и создание эскиза, перезаписывая уже сохраненные данные.

Из-за особенностей очередей Windows Azure сообщение может быть считано дважды одной или разными рабочими ролями.

Если метод обработки сообщения не идемпотентен, то имеются разные способы предотвращения повторной обработки сообщения получателем:

При считывании сообщения из очереди можно с помощью параметра visibilitytimeout задать, как долго сообщение должно быть скрыто от других агентов чтения (по умолчанию 30 секунд). Это дает время, чтобы гарантированно обработать и удалить сообщение из очереди до того, как его прочитает другой клиент. При извлечении сообщения из очереди он не удаляется оттуда автоматически. Период, определенный параметром visibilitytimeout, все же может истечь до удаления сообщения, например если метод, обрабатывающий сообщение, завершится ошибкой.

Каждое сообщение имеет свойство DequeueCount, где фиксируется, сколько раз сообщение извлекалось из очереди. Однако, если полагаться на это свойство и обрабатывать только те сообщения, для которых значение DequeueCount равно 0, приложение может столкнуться с ситуацией, когда сообщение было извлечено из очереди, но не обработано.

Можно также добавить к сообщению уникальный идентификатор транзакции, а затем сохранить этот идентификатор в свойствах метаданных большого двоичного объекта. Если при извлечении сообщения из очереди его уникальный идентификатор совпадает с уникальным идентификатором в метаданных большого двоичного объекта, то это будет означать, что сообщение уже обрабатывалось.

Кроме того, приложение позволяет пользователям просматривать только те изображения чеков оплаты, которые были ими загружены, либо изображения, относящиеся к подтверждениям деловых расходов, которые они могут утвердить. Остальные изображения будут скрыты для обеспечения конфиденциальности других пользователей. Чтобы добиться этой цели, после рассмотрения нескольких подходов было принято решение использовать Shared Access Signature в Windows Azure.

Комментарий Яны:

Мы рассмотрели три альтернативных подхода к обеспечению просмотра отсканированных чеков оплаты деловых расходов, прежде чем пришли к Shared Access Signature.

В Windows Azure все механизмы хранения можно настроить так, чтобы пользователи могли считывать данные отовсюду по анонимным запросам, что делает модель, показанную на рис. 3, очень простой для реализации:

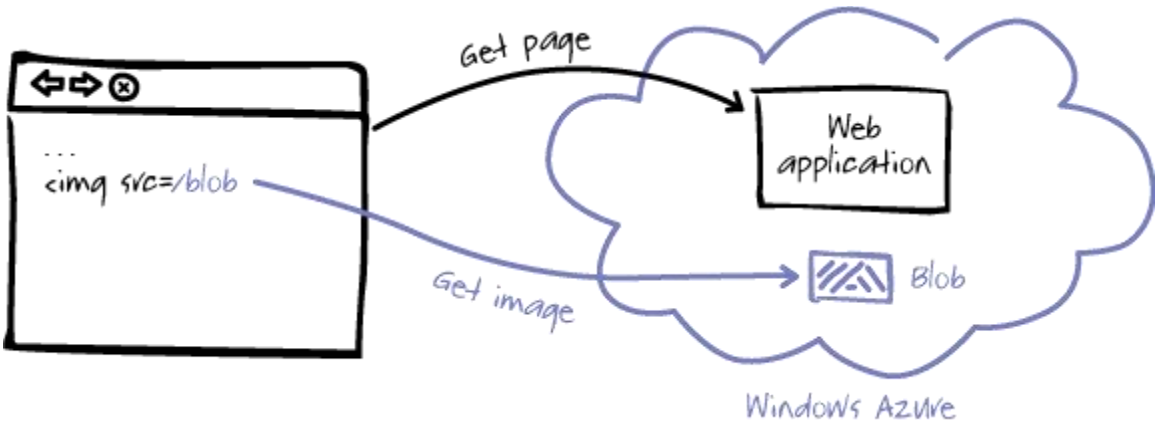


Рис. 3

Хранилище с прямой адресацией

В этом сценарии можно обращаться к содержимому большого двоичного объекта напрямую через URL-адрес вида https://<приложение>.blob.core.windows.net/<имя-контейнера>/<имя-большого-двоичного-объекта>. В приложении aExpense URL-адреса можно было сохранять в составе сущностей расходов в хранилище таблиц Windows Azure. Такой подход не годится для приложения aExpense, так как он дает возможность постороннему лицу угадать адрес сохраненного изображения, хотя он хорошо подходит для данных, которые предназначены для публичного доступа — эмблемы, реклама, загружаемые брошюры. Преимущества этого подхода — простота, кэширование данных, разгрузка веб-сервера и возможность задействования инфраструктуры сети доставки содержимого (CDN) в Windows Azure. Недостаток — низкий уровень защищенности.

Примечание.

Использование преднамеренно сложных и запутанных URL-адресов может стать вариантом, однако такой подход предлагает лишь слабую защиту и не рекомендуется к использованию.

Комментарий Бхарата:

CDN позволяет кэшировать данные больших двоичных объектов в стратегических местоположениях для доставки содержимого пользователям с максимально доступной пропускной способностью.

Второй подход, рассмотренный разработчиками для доступа к изображениям чеков оплаты в хранилище больших двоичных объектов, заключался в маршрутизации запросов через веб-сайт

таким же способом, как в «традиционных» многоуровневых приложениях, где запросы данных направляются через средний уровень. Эта модель показана на рис. 4.

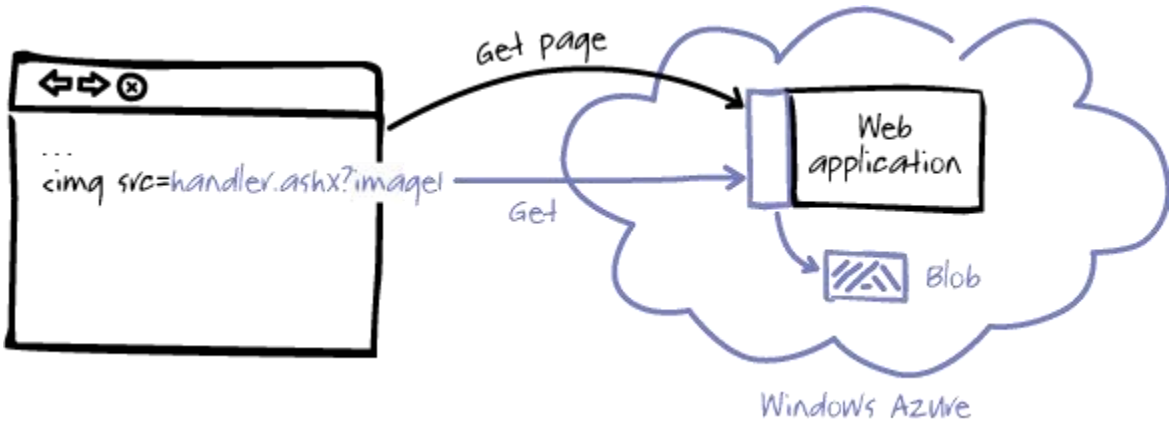


Рис. 4

Маршрутизация запросов изображений через веб-сервер

В этом случае публичный доступ к контейнеру большого двоичного объекта будет отсутствовать, а веб-приложение будет хранить ключи доступа к хранилищу больших двоичных данных. Именно таким образом aExpense записывает графические данные в хранилище больших двоичных объектов. Хотя такой подход позволит Adatum управлять доступом к изображениям, он повысит сложность веб-приложения и увеличит рабочую нагрузку на веб-сервер. Возможная реализация такого сценария потребовала бы создания обработчика HTTP для перехвата запросов изображений, проверки правил доступа и возврата данных. Кроме того, такой подход не годится при использовании Windows Azure CDN.

Поэтому был выбран поход с использованием Shared Access Signature в Windows Azure. Shared Access Signature позволяют управлять доступом к отдельным большим двоичным объектам, даже если доступ задан на уровне контейнера, создавая URL-адреса для доступа к объектам, которые действуют только в течение ограниченного периода времени. В приложении aExpense веб-роль создает такие специальные URL-адреса и встраивает их в страницу, передаваемую на сторону пользователя. Эти специальные URL-адреса обеспечивают прямой доступ к большому двоичному объекту в течение ограниченного периода времени. В этом случае возникает некоторая дополнительная нагрузка на веб-сервер, так как ему приходится создавать URL-адреса подписей коллективного доступа, но основную обработку выполняет хранилище больших двоичных объектов Windows Azure. Такой подход обеспечивает разумную защиту, так как URL-адреса Shared Access Signature, помимо того, что имеют ограниченное время существования, содержат также уникальную подпись, которая делает весьма сложным подбор URL-адреса.

Комментарий Яны:

Shared Access Signature обеспечивают неплохую защиту. URL-адреса действуют ограниченное время, и их трудно подобрать.

Реализации

Теперь пришло время рассмотреть эти изменения более подробно. По мере изучения этого раздела может потребоваться загрузка решения для системы разработки Microsoft® Visual Studio® со страницы http://wag.codeplex.com/. Это решение (в папке Azure-WithWorkers) содержит реализацию aExpense после внесения изменений этого этапа. Если вы не заинтересованы в изучении таких механизмов, перейдите к следующему разделу.

Передача и сохранение изображений

В приложении aExpense веб-роль отвечает за передачу изображений с пользовательской рабочей станции и сохранение его первоначальной, несжатой версии. Следующий код в методе SaveExpense класса ExpenseRepository сохраняет исходное, несжатое изображение в хранилище больших двоичных объектов.

Копировать

```
this.receiptStorage.AddReceipt(expenseItem.Id.ToString(),
    expenseItem.Receipt, string.Empty);
```

Следующий код из класса ExpenseReceiptStorage показывает, как приложение сохраняет графические данные в хранилище больших двоичных объектов.

Копировать

```
public string AddReceipt(string receiptId, byte[] receipt,
    string contentType)
{
    CloudBlob blob = this.container.GetBlobReference(receiptId);
    blob.Properties.ContentType = contentType;
    blob.UploadByteArray(receipt);

    return blob.Uri.ToString();
}
```

Абстрагирование рабочей роли

На рис. 5 показана обобщенная модель взаимодействия веб-ролей и рабочих ролей в Windows Azure.

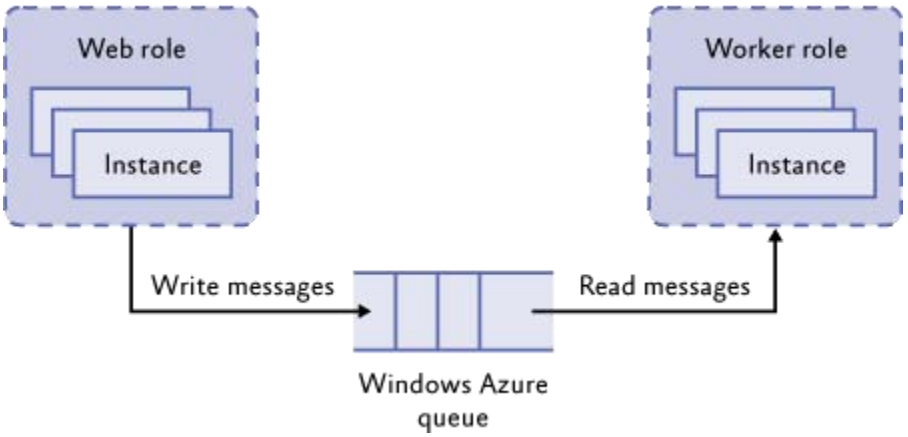


Рис. 5

Взаимодействие веб-ролей и рабочих ролей через очередь Windows Azure

Комментарий Яны:

Это очень распространенная модель взаимодействия веб-роли и рабочей роли.

В этой модели экземпляр веб-роли помещает сообщения в очередь для взаимодействия с рабочей ролью. Экземпляр рабочей роли опрашивает очередь, не появились ли в ней новые сообщения. Затем получает и обрабатывает их. Есть несколько важных фактов, касающихся работы службы очередей в Windows Azure, которые надо знать. Во-первых, очередь указывается по имени, и несколько экземпляров роли могут совместно использовать одну очередь. Во-вторых, нет такого понятия, как типизированное сообщение. Сообщение строится из строк или массива байтов. Размер одного сообщения не может превышать 8 килобайт (КБ).

Примечание.
Если размер сообщений приближается к максимальному, имейте в виду, что перед добавлением в очередь Windows Azure преобразует все сообщения в формат Base64.

Кроме того, в Windows Azure стандартные очереди реализован механизм доставки «хотя бы один раз». Поэтому доставка сообщений по принципу «первым пришел, первым ушел» (FIFO) или доставка только одной копии сообщения не гарантируется. Ваше приложение должно уметь обрабатывать такие ситуации.

Стандартные очереди Windows Azure не гарантирует ни доставку сообщений по принципу «первым пришел, первым ушел» (FIFO), ни доставку только одной копии сообщения.

Хотя на текущем этапе проведения миграции aExpense в Windows Azure рабочая роль выполняет только одну задачу, в Adatum ожидают, что на последующих этапах рабочей роли передадут дополнительные обязательства. Поэтому команда Adatum разработала несколько простых программных «соединительных» решений, скрывающих сложности использования рабочих ролей

и очередей Windows Azure и делающих их более удобными для работы в будущем. На рис. 6 приведено общее описание этих абстракций и показано, куда можно подключить ваши специальные функции рабочей роли.

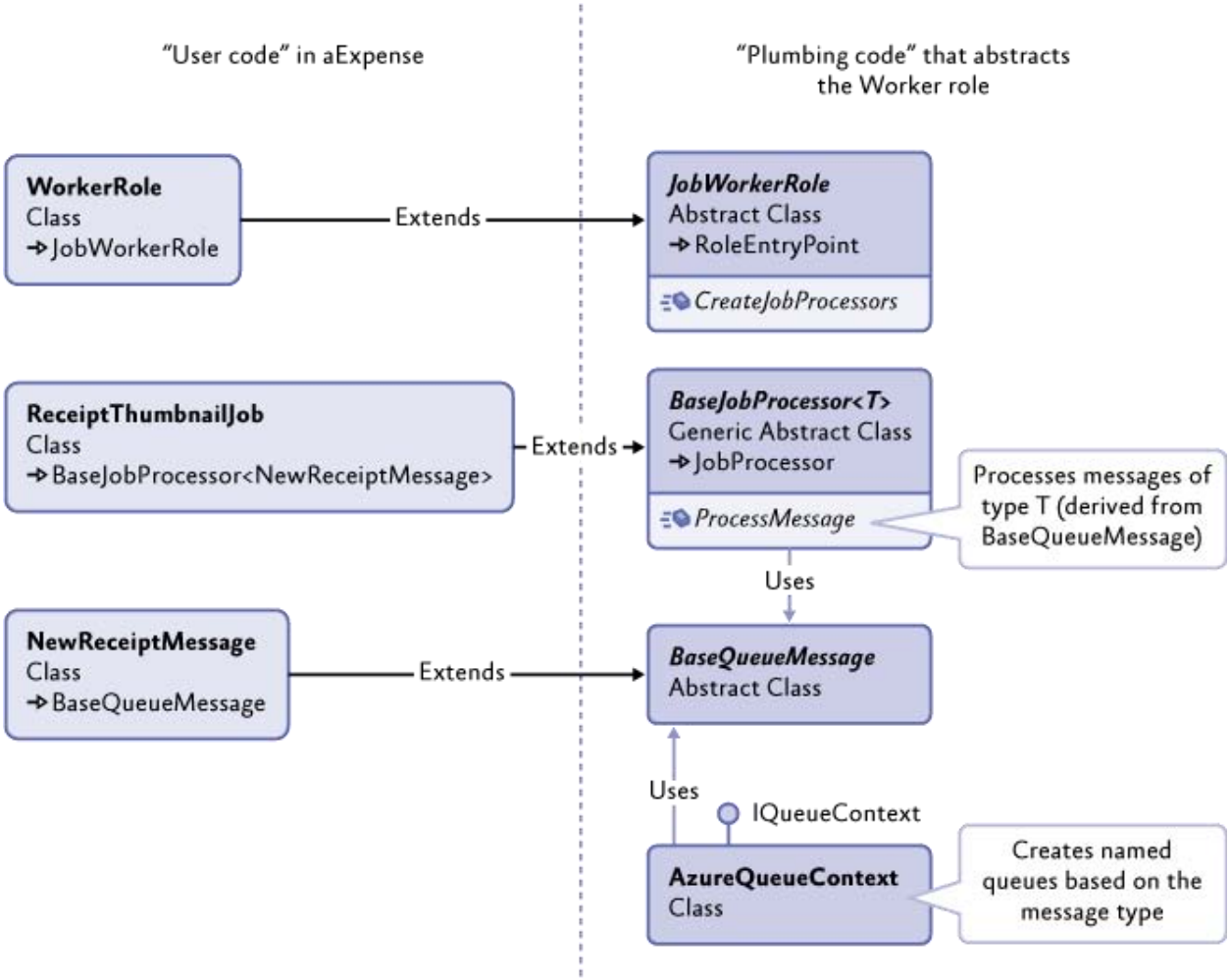


Рис. 6

Связь «пользовательского кода» и соединительного кода

Классы «пользовательского кода» нужно реализовать для каждой рабочей роли и типа заданий. Классы «соединительного кода» являются элементами многократного использования. Классы «соединительного кода» упакованы в пространствах имен AExpense.Jobs, AExpense.Queues и AExpense.QueueMessages. В следующих двух разделах сначала обсуждается «пользовательский код», а затем «соединительный код».

Комментарий Маркуса:

Для любой новой фоновой задачи потребуется реализовать только компоненты «пользовательского кода».

«Пользовательский код» в приложении aExpense

Код, описанный в этом разделе, реализует задание, которое сжимает изображения и создает эскизв рабочей роли для приложения aExpense. В этом разделе показано, насколько просто определить новый тип задания, которое будет выполняться рабочей ролью. В этом коде использован «соединительный код», который будет подробно описан в следующем разделе.

Следующий код показывает, как приложение инициализирует рабочую роль с помощью «соединительного кода»: создается новый класс, производный от JobWorkerRole, и переопределяется метод CreateJobProcessors. В этом методе создаются экземпляры объектов обработки задания, которые реализуют интерфейс IJobProcessor. Как видно, этот подход позволяет легко подключать любые дополнительные типы заданий, реализующие интерфейс IJobProcessor.

Копировать

```
public class WorkerRole: JobWorkerRole

{

   protected override IEnumerable<IJobProcessor>

     CreateJobProcessors()

   {

      return new IJobProcessor[] { new receiptThumbnailJob() };

   }

}
```

Примечание.

Этот код не полностью соответствует коду, находящемуся в загруженном решении, поскольку компания Adatum изменила его на последующем этапе данного проекта. Дополнительные сведения см. в главе 8, «Этап 4. Добавление других задач и настройка приложения».

Конструктор класса ReceiptThumbnailJob задает используемый рабочей ролью интервал опроса очереди и создает экземпляры объектов AzureQueueContext, ExpenseReceiptStorage и ExpenseRepository для использования в рабочей роли. «Соединительный код» передает для обработки объект NewReceiptMessage с деталями изображения методу ProcessMessage. Затем этот метод сжимает указанное в сообщении изображение и создает уменьшенную копию. В следующем коде показаны конструктор и метод ProcessMessage класса ReceiptThumbnailJob.

Комментарий Маркуса:

Рабочие роли должны опрашивать очередь, нет ли новых сообщений.

Копировать

```
public ReceiptThumbnailJob()
   : base(2000, new AzureQueueContext())
{
   this.receiptStorage = new ExpenseReceiptStorage();
   this.expenseRepository = new ExpenseRepository();
}


public override bool ProcessMessage(NewReceiptMessage message)
{
   …
}
```

В приложении aExpense, чтобы отправить сообщение с данными нового изображения чека в рабочую роль, веб-роль создает объект NewReceiptMessage и вызывает метод AddMessage класса AzureQueueContext. В следующем примере кода показано определение класса NewReceiptMessage.

Копировать

```
[DataContract]
public class NewReceiptMessage: BaseQueueMessage
{
   [DataMember]
   public string ExpenseItemId { get; set; }
}
```

Примечание.
Важно использовать атрибуты DataContract и DataMember в классе сообщения, потому что класс AzureQueueContext сериализует экземпляры сообщений в формат JSON.

Последняя строка в методе SaveExpense из следующего примера кода демонстрирует, как веб-роль в aExpense помещает сообщение в очередь.

Копировать

```
public void SaveExpense(Expense expense)
```

```
{
   var context = new ExpenseDataContext(this.account);
   ExpenseRow expenseRow = expense.ToTableEntity();

   foreach (var expenseItem in expense.Details)
   {
      // создать строку
      var expenseItemRow = expenseItem.ToTableEntity();
      expenseItemRow.PartitionKey = expenseRow.PartitionKey;
      expenseItemRow.RowKey =
        string.Format(CultureInfo.InvariantCulture, "{0}_{1}",
        expense.Id, expenseItemRow.Id);

      context.AddObject(ExpenseDataContext.ExpenseItemTable,
        expenseItemRow);

      // сохранить изображение чека, если имеется
      if (expenseItem.Receipt != null &&
         expenseItem.Receipt.Length > 0)
      {
         this.receiptStorage.AddReceipt(
            expenseItem.Id.ToString(), expenseItem.Receipt,
            string.Empty);
      }
   }

   context.SaveChanges(SaveChangesOptions.Batch);

   context.AddObject(ExpenseDataContext.ExpenseTable,
```

```
        expenseRow);
    context.SaveChanges();


    var queue = new AzureQueueContext(this.account);
    expense.Details.ToList().ForEach(
        i => queue.AddMessage(new NewReceiptMessage
            { ExpenseItemId = i.Id.ToString() }));
}
```

Порядок операций в методе SaveExpense имеет значение. Поскольку записи сведений и заголовков хранятся в разных таблицах, Adatum для обеспечения целостности данных не может использовать транзакцию. Поэтому Adatum сохраняет сначала сведения, а затем их заголовок. И если сбой возникнет в промежутке, запись заголовка без сведений не появится. Adatum сможет реализовать задачу очистки, которая будет искать сведения и изображения чеков, не имеющие заголовков. Не имеющие заголовков сведения и изображения чеков не будут отображаться в пользовательском интерфейсе или в каких-либо экспортируемых данных. Если сохранять записи заголовков раньше, чем записи сведений, то при сбоях могут возникнуть записи заголовков без сведений. В этом случае в пользовательском интерфейсе и в любых экспортируемых данных будут отображаться неполные данные.

Комментарий Маркуса:

В главе 8, «Этап 4. Добавление других задач и настройка приложения», Adatum изучает возможности использования одной таблицы для записей и заголовков и сведений. Это даст возможность использовать транзакцию для всей операции.

Второй важный аспект порядка операций состоит в следующем. Пока все записи не будут сохранены, метод не поместит никакие сообщения в очередь, чтобы уведомить рабочую роль о том, что есть новые изображения для обработки. При этом рабочая роль никогда не начнет обрабатывать изображение раньше, чем будет сохранена связанная запись, и не завершится сбоем из-за того, что не сможет найти эту запись (вспомните, что рабочая роль должна обновить URL-адреса изображения и эскиза в записи сведений).

Классы «соединительного кода»

Adatum разработала эти абстракции, чтобы упростить отправку сообщений из веб-роли в рабочую роль и чтобы упростить кодирование рабочей роли. Идея заключается в том, чтобы поместить в очередь типизированное сообщение, а когда оно будет получено рабочей ролью, та направит его в обработчик заданий для этого типа сообщений. В предыдущем разделе описан обработчик заданий в приложении aExpense, который обрабатывает отсканированные изображения чеков и использует эти абстракции. Классы «соединительного кода» состоят из следующих основных элементов.

Классы «соединительного кода» упрощают отправку сообщений из веб-роли в рабочую роль и реализацию рабочей роли.

Комментарий Яны:

Adatum планирует реализовать дополнительные фоновые процессы, поэтому создание такого «соединительного кода» имеет смысл.

Оболочка для стандартного класса рабочей роли RoleEntryPoint в Windows Azure, который абстрагирует жизненный цикл и поведение потоков для рабочей роли.

Настраиваемый обработчик заданий, который позволяет пользователям классов «соединительного кода» определять собственные типы заданий для рабочей роли.

Оболочка для стандартного класса CloudQueue в Windows Azure, который реализует типизированные сообщения для маршрутизации сообщений в классе JobWorkerRole.

На рис. 7 приводится описание того, как классы «соединительного кода» обрабатывают сообщения, полученные от класса BaseQueueMessage.

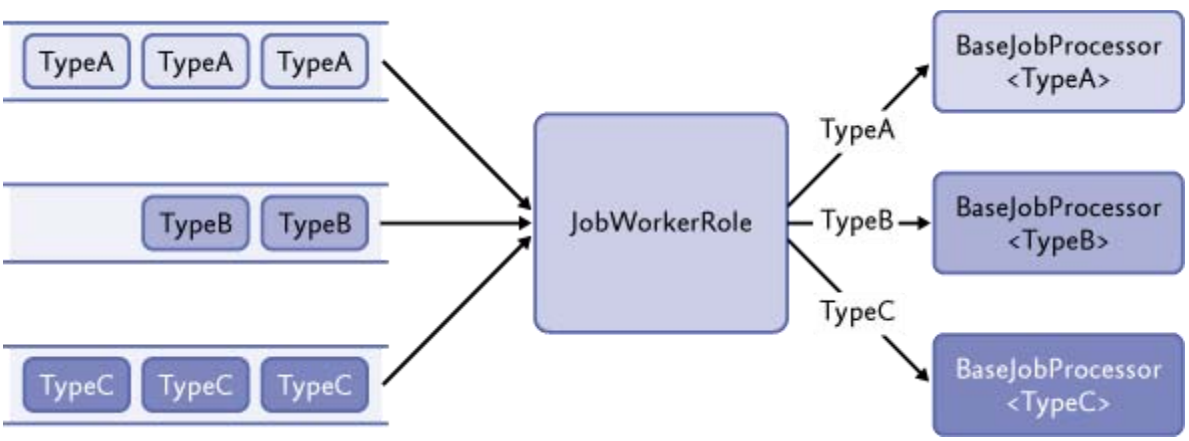


Рис. 7

Элементы «соединительного кода» рабочей роли

Типы сообщений, обрабатываемые классами «соединительного кода» (такие как тип NewReceiptMessage в aExpense), являются производными от класса BaseQueueMessage, приведенного в следующем примере кода.

Копировать

```
[DataContract]
```

```
public abstract class BaseQueueMessage
{
   частный контекст объекта;

   [System.Diagnostics.CodeAnalysis.SuppressMessage(…)
   public object GetContext()
   {
      return this.context;
   }

   public void SetContext(object value)
   {
      this.context = value;
   }
}
```

Классы «соединительного кода» используют класс AzureQueueContext для доставки сообщений в рабочую роль. Класс AzureQueueContext создает именованные очереди на основе типов сообщений — по одной очереди для каждого зарегистрированного типа сообщений. В следующем коде показан метод Add класса AzureQueueContext, используемый для добавления нового сообщения в очередь, и метод ResolveQueueName, который определяет имя используемой очереди.

Копировать

```
public void AddMessage(BaseQueueMessage message)
{
   var queueName = ResolveQueueName(message.GetType());
   this.EnsureQueueExists(queueName);

   var json = Serialize(message.GetType(), message);
   var queue = this.queue.GetQueueReference(queueName);
   queue.AddMessage(new CloudQueueMessage(json));
```

```
}


public static string ResolveQueueName(MemberInfo messageType)

{

   return messageType.Name.ToLowerInvariant();

}
```

Есть еще два момента, на которые нужно обратить внимание при реализации метода AddMessage. Во-первых, «соединительный код» сериализует сообщения в формат JSON, в котором сообщения обычно оказываются короче, чем в кодировке XML (но, возможно, длиннее, чем в двоичном виде). Во-вторых, метод EnsureQueueExists вызывает метод CreateIfNotExist класса CloudQueue в Windows Azure. Вызов метода CreateIfNotExist считается транзакцией хранилища и увеличивает расходы на эксплуатацию приложения.

Комментарий По:

Если вас интересует стоимость эксплуатации приложения, нужно узнать, какие вызовы в коде могут быть платными!

Примечание.

В настоящее время корпорация Microsoft установила тариф для оплаты транзакций хранилища в размере 0,01 доллара за 10 КБ. При большом объеме сообщений следует проверить, насколько часто приложение вызывает этот метод.

В главе 8, «Этап 4. Добавление нескольких задач и настройка приложения», место, где приложение вызывает CreateIfNotExist, пересматривается на основании результатов проверки производительности приложения.

Классы «соединительного кода» доставляют сообщения в компоненты обработчика заданий, где он обрабатывает конкретный тип сообщения. Классы «соединительного кода» включают интерфейс IJobProcessor, который определяет два метода типа void под названием Run и Stop для запуска и остановки обработчика. Этот интерфейс реализуется абстрактными классами BaseJobProcessor и JobProcessor. В приложении aExpense приведенный ранее класс ReceiptThumbnailJob расширяет класс BaseJobProcessor. В следующем примере кода показано, как класс JobProcessor реализует интерфейс IJobProcessor.

Копировать

```
private bool keepRunning;


public void Stop()
```

```
{
   this.keepRunning = false;
}

public void Run()
{
   this.keepRunning = true;
   while (this.keepRunning)
   {
      Thread.Sleep(this.SleepInterval);
      this.RunCore();
   }
}

protected abstract void RunCore();
```

В следующем примере кода показано, как в классе BaseJobProcessor реализуется метод RunCore.

Копировать

```
protected bool RetrieveMultiple { get; set; }
protected int RetrieveMultipleMaxMessages { get; set; }

protected override void RunCore()
{
   if (this.RetrieveMultiple)
   {
      var messages = this.Queue.GetMultipleMessages<T>
         (this.RetrieveMultipleMaxMessages);
      if (messages != null)
      {
         foreach (var message in messages)
```

```
        {
            this.ProcessMessageCore(message);
        }
    }
    либо
    {
        this.OnEmptyQueue();
    }
  }
  либо
  {
    var message = this.Queue.GetMessage<T>();
    if (message != null)
    {
        this.ProcessMessageCore(message);
    }
    либо
    {
        this.OnEmptyQueue();
    }
  }
}
```

Как видно, метод RunCore может получать несколько сообщений из очереди за один раз. Преимуществом этого подхода является то, что один вызов метода GetMessages класса Windows Azure CloudQueue считается лишь одной транзакцией хранилища, независимо от числа получаемых им сообщений. В этом примере кода также показано, как класс BaseJobProcessor вызывает универсальные методы GetMessage и GetMultipleMessages класса AzureQueueContext, указывая тип сообщения с помощью параметра универсального типа.

Комментарий Бхарата:

Дешевле и эффективнее получать несколько сообщений за один раз, если это возможно. Но эти преимущества следует соотнести с тем, что, поскольку на обработку нескольких сообщений тратится больше времени, возникает риск того, что сообщения увидят другие агенты чтения очереди до того, как они будут удалены.

В следующем примере кода показано, как конструктор класса BaseJobProcessor назначает интервал опроса задания и ссылку на AzureQueueContext.

Копировать

```
protected BaseJobProcessor(int sleepInterval,
   IQueueContext queue): base(sleepInterval)
{
   if (queue == null)
   {
      throw new ArgumentNullException("queue");
   }

   this.Queue = queue;
}
```

Из важных методов класса BaseJobProcessor осталось рассмотреть метод ProcessMessageCore и абстрактный метод ProcessMessage, которые показаны ниже.

Копировать

```
protected int MessagesProcessed { get; set; }

private void ProcessMessageCore(T message)
{
   var processed = this.ProcessMessage(message);
   if (processed)
   {
      this.Queue.DeleteMessage(message);
      this.MessagesProcessed++;
```

```
    }
}
```

```
public abstract bool ProcessMessage(T message);
```

Метод RunCore вызывает метод ProcessMessageCore, когда находит новые сообщения для обработки. Затем метод ProcessMessageCore вызывает «предоставленную пользователем» реализацию метода ProcessMessage перед тем, как удалить сообщение из очереди. В приложении aExpense данная реализация находится в классе ReceiptThumbnailJob.

Последним компонентом «соединительного кода» является абстрактный класс JobWorkerRole, выступающий в роли оболочки стандартного класса Windows Azure RoleEntryPoint для рабочей роли. В следующем примере кода показан метод Run этого класса.

Копировать

```
protected IEnumerable<IJobProcessor> Processors { get; set; }


protected abstract IEnumerable<IJobProcessor>
    CreateJobProcessors();


public override void Run()
{
    this.Processors = this.CreateJobProcessors();


    var threads = new List<Thread>();


    foreach (var processor in this.Processors)
    {
        var t = new Thread(processor.Run);
        t.Start();
        threads.Add(t);
    }
```

```
    foreach (var thread in threads)
    {
        thread.Join();
    }
}
```

Метод Run вызывает абстрактный метод CreateJobProcessors, реализованный в «пользовательском коде». В приложении aExpense данная реализация находится в классе WorkerRole. Затем метод Run создает новый поток для каждого обработчика заданий и ожидает окончания работы всех потоков.

Комментарий Маркуса:

Необходимо поддерживать рабочую роль в активном состоянии!

Обработка изображений

В следующем примере кода показано, как в приложении aExpense реализованы функции обработки изображений в методе ProcessMessage класса ReceiptThumbnailJob.

Копировать

```
public override bool ProcessMessage(NewReceiptMessage message)
{
    var expenseItemId = message.ExpenseItemId;
    var imageName = expenseItemId + ".jpg";

    byte[] originalPhoto =
        this.receiptStorage.GetReceipt(expenseItemId);

    if (originalPhoto != null && originalPhoto.Length > 0)
    {
        var thumb = ResizeImage(originalPhoto, ThumbnailSize);
        var thumbUri =
            this.receiptStorage.AddReceipt(Path.Combine(
            "thumbnails", imageName), thumb, "image/jpeg");
```

```
        var photo = ResizeImage(originalPhoto, PhotoSize);
        var photoUri = this.receiptStorage.AddReceipt(imageName,
            photo, "image/jpeg");

        this.expenses.UpdateExpenseItemImages(expenseItemId,
            photoUri, thumbUri);

        this.receiptStorage.DeleteReceipt(expenseItemId);

        return true;
    }

    return false;
}
```

Этот метод извлекает имя изображения из сообщения, отправленного в рабочую роль, а затем создает две новые версии изображения: эскиз и изображение «стандартного» размера. Затем он удаляет исходное изображение. Этот метод может обрабатывать изображения в любом стандартном формате, но сохраняет их всегда в формате JPEG.

Комментарий Яны:

Хотя можно передавать изображения, размер которых не превышает 1024 Кбайта, для экономии места мы решили не сохранять исходные изображения. Мы выяснили, что преобразование в стандартный размер обеспечивает приемлемое качество.

Примечание.

Метод ProcessMessage должен быть идемпотентным, чтобы не возникало нежелательных побочных эффектов при доставке сообщения несколько раз. Метод ProcessMessage также должен содержать некоторые правила для обработки «проблемных» сообщений, которые по какой-то причине нельзя обработать.

Предоставление доступа к изображениям с помощью Shared Access Signature

Чтобы изображения чеков были доступны для просмотра в пользовательском интерфейсе, команда Adatum воспользовалась подписями коллективного доступа (, SAS) для формирования

краткосрочных, безопасных URL-адресов для изображений в хранилище больших двоичных объектов. При этом подходе не предоставляется общий доступ к контейнеру больших двоичных объектов, но операции на стороне веб-сервера минимальны, поскольку клиент может получить доступ к изображению непосредственно в хранилище больших двоичных объектов.

Приложение AExpense использует Shared Access Signature для предоставления ограниченного доступа к большим двоичным объектам в закрытых контейнерах.

В следующем примере кода показано, как приложение формирует URL-адреса Shared Access Signature в методе GetExpenseByID класса ExpenseRepository путем добавления подписей к URL-адресу большого двоичного объекта. Приложение AExpense использует конечную точку HTTPS, поэтому ссылка на большой двоичный объект и элементы подписи в URL-адресе большого двоичного объекта защищены протоколом SSL от атак типа «человек посередине».

Примечание.
Использование протокола SSL обеспечивает шифрование всех данных URL-адресов, за исключением имени узла.

Копировать

```
CloudBlob receiptBlob =
   container.GetBlobReference(item.ReceiptUrl.ToString());
item.ReceiptUrl = new Uri(item.ReceiptUrl.AbsoluteUri +
receiptBlob.GetSharedAccessSignature(policy));
```

Комментарий Маркуса:

Shared Access Signature добавляется к стандартному URL-адресу большого двоичного объекта.

Метод GetSharedAccessSignature принимает объект SharedAccessPolicy в качестве параметра. Такая политика определяет права доступа и время существования созданного URL-адреса. В следующем коде показана используемая приложением aExpense политика, предоставляющая в течение одной минуты разрешение на чтение для изображения. Приложение создает новую Shared Access Signature всякий раз, когда пользователь пытается получить доступ к документам по подтверждениям расходов.

Копировать

```
private readonly TimeSpan sharedSignatureValiditySpan;
```

```
var policy = new SharedAccessPolicy
{
   Permissions = SharedAccessPermissions.Read,
   SharedAccessExpiryTime = DateTime.UtcNow +
      this.sharedSignatureValiditySpan
};
```

Примечание.

В приложении не указывается значение для свойства SharedAccessStartTime объекта SharedAccessPolicy. При разнице во времени между клиентом и сервером приравнивание этого значения текущего времени может привести к проблемам, если попытка доступа к большому двоичному объекту производится немедленно.

Поскольку запрос Get к большому двоичному объекту начнется до окончания срока действия, он завершится успешно, даже если поток ответа продолжится после окончания срока действия. В рамках логической операции Get, если приложение пытается повторить завершившуюся сбоем операцию Get (такова политика библиотеки StorageClient по умолчанию), любой повторный запрос, сделанный после окончания срока действия, завершится сбоем. Если для URL-адреса выбран короткий срок действия, убедитесь, что для всего большого двоичного объекта издается один запрос Get и используется пользовательская политика повторов, чтобы при повторном запросе к URL-адресу выдавалась новая подпись коллективного доступа.

Дополнительные сведения

Последние версии инструментов Windows Azure для среды Microsoft Visual Studio и пакет SDK для Windows Azure можно скачать в центре загрузки Microsoft.

В MSDN® содержится много информации о хранилище больших двоичных объектов в Windows Azure. Рекомендуется начать со страницы http://msdn.microsoft.com/ru-ru/library/dd135733.aspx.

Дополнительные сведения об управлении доступом к хранилищу Azure, включая Shared Access Signature, см. по адресу http://msdn.microsoft.com/ru-ru/library/ee393343.aspx.

Общие сведения об архитектуре служб Windows Azure см. по адресу http://msdn.microsoft.com/ru-ru/library/dd179341.aspx.

Глава 7. Управление жизненным циклом приложений в Windows Azure

В этой главе излагается подход к управлению жизненным циклом приложений для приложений на технологической платформе Windows® Azure™. Хотя для разных приложений и организаций конкретные требования будут отличаться, все сначала разрабатывают приложения, затем тестируют их и, наконец, развертывают. В этой главе основное внимание уделено тому, где следует тестировать приложения и как нужно управлять процессом развертывания, чтобы обеспечить

доступ к действующей производственной (production) среде только для уполномоченного персонала.

Исходные условия

Компания Adatum имеет четко определенный набор процессов для развертывания приложений на локальных серверах. Она использует отдельные серверы для тестирования, staging режима и производственной эксплуатации. Когда команда разработчиков выпускает новую версию приложения, ее продвижение по серверам тестирования, staging серверам и производственной эксплуатации жестко контролируется. Но иногда, хотя и очень редко, небольшие исправления, такие как обновление текста на странице ASPX, выполняются непосредственно на производственных серверах.

Цели и требования

В Adatum сформулировали ряд целей для управления жизненным циклом приложений Windows Azure. В Adatum хотят организовать ясно заданный процесс развертывания приложений в Windows Azure с четко определенными ролями и обязанностями. Конкретнее, в компании хотят быть уверенными в том, что доступ к активной рабочей среде есть только у нескольких ключевых людей и что любые изменения в рабочей среде можно отследить.

Нужен четко определенный процесс развертывания приложений в Windows Azure.

Кроме того, в Adatum хотят при возникновении проблем иметь возможность откатить активную рабочую среду к предыдущей версии. В худшем случае должна быть возможность снова сделать приложение локальным.

В Adatum хотели бы проводить ряд тестов приложения в среде, максимально близкой к активной рабочей.

Комментарий Бхарата:

Все среды Windows Azure в облаке одинаковы; нет никаких отличий между тестовой или staging областью. Однако имеются различия между локальным эмулятором вычислений и облачной средой, именно поэтому так важно тестировать приложение в облаке.

Наконец, в Adatum хотят свести к минимуму расходы на обслуживание всех необходимых сред и иметь возможность отделить расходы на среду разработки и тестирования от расходов на действующую производственную среду.

Не забывайте, что любые действия в Windows Azure, даже если это только тестирование, стоят денег.

Обзор решения

Прежде чем что-либо развертывать в Windows Azure компания Adatum может использовать локальный эмулятор вычислений и эмулятор хранилища для всех задач разработки и для большей части тестов. Однако специалисты Adatum должны проверить свои приложения в облаке, прежде чем развертывать их в реальной производственной среде.

Для этого в Adatum имеются две подписки Windows Azure. Одна учетная запись используется для тестирования, а другая — для действующей производственной среды. Поскольку каждая учетная запись имеет собственный идентификатор Windows Live® ID и собственный набор ключей программных интерфейсов, в Adatum могут ограничить доступ к каждой среде, предоставив его лишь определенным сотрудникам. Члены группы тестирования и ключевые члены группы разработки имеют доступ к учетной записи тестирования. К производственной учетной записи имеют доступ только два ключевых человека в отделе операций Adatum.

Обе эти учетные записи являются стандартными учетными записями Windows Azure, и в Adatum могут быть уверены в том, что код приложения будет выполняться одинаково в обеих средах, поскольку они идентичны. В Adatum также могут быть уверены, что развертывание приложения будет происходить одинаково в обеих средах, поскольку для развертывания в производственной и тестовой среде можно использовать один и тот же пакет.

Комментарий Бхарата:

В Windows Azure можно быть уверенным, что для различных учетных записей будут одинаковые среды, что очень трудно гарантировать для локальных сред.

В Adatum также могут проводить некоторые тесты, запуская приложение в эмуляторе вычислений, но указывая приложению хранилище в тестовой среде Windows Azure. Это важно, поскольку различий между эмулятором хранилища и облачным хранилищем больше, чем между эмулятором вычислений и облачной средой выполнения. Кроме того, облачное хранилище является относительно дешевым в использовании по сравнению с другими облачными ресурсами.

Комментарий По:

Поскольку сотрудники группы разработки и тестирования не имеют доступа к ключам производственной учетной записи, нет риска случайного тестирования на производственных данных, передаваемых в режиме реального времени.

Корпорация Microsoft выставляет компании Adatum счета отдельно по двум средам, что позволяет Adatum легко отделить эксплуатационные расходы на активную рабочую среду от расходов, связанных с разработкой и тестированием. Это позволяет Adatum управлять бюджетом разработки продуктов отдельно от бюджета эксплуатации аналогично тому, как компания управляет бюджетами для локальных приложений.

Установка и физическое развертывание

На рис. 1 дается обзор подхода к управлению жизненным циклом приложения в компании Adatum.

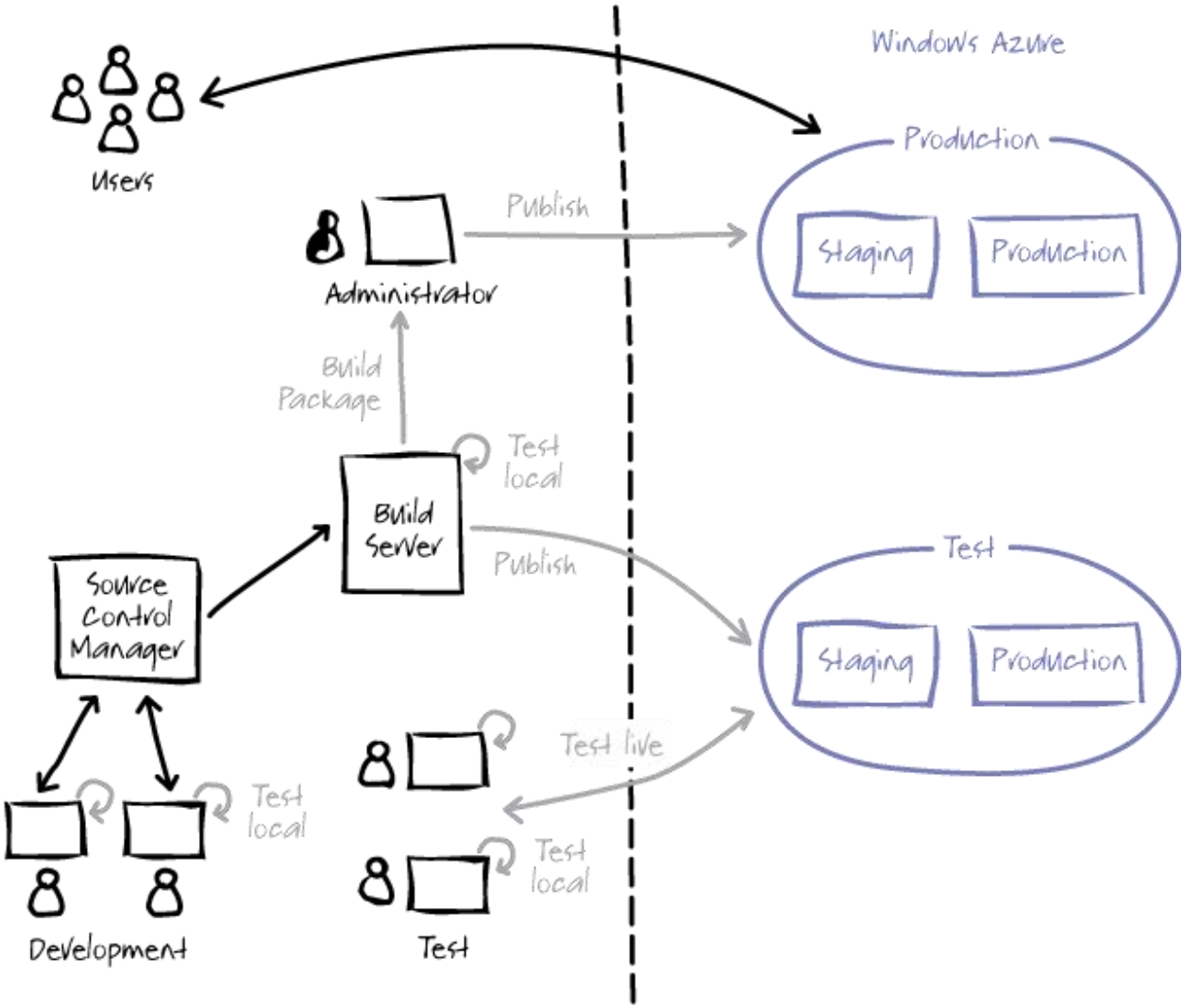


Рис. 1

Среда управления жизненным циклом приложения компании Adatum

На предыдущей диаграмме показаны две отдельные среды Windows Azure, которые в Adatum используются для тестирования и для промышленной эксплуатации, а также локальная среда, состоящая из компьютеров разработчиков, тестовых компьютеров, сервера сборки и средства управления исходным кодом.

Среды Windows Azure

Существует два способа доступа к среде Windows Azure для выполнения задач развертывания и настройки. Во-первых, через портал разработчиков Windows Azure, где один идентификатор Windows Live ID дает доступ ко всему содержимому портала. Во-вторых, с помощью служебных управляющих API-интерфейсов Windows Azure (Windows Azure Management API), где для доступа ко всем функциям, предоставляемым программным интерфейсом, используются сертификаты. В обоих случаях в настоящее время невозможно ограничить доступ пользователей только определенным подмножеством функций, например, предоставить учетной записи только

возможность управлять параметрами хранилища. В компании Adatum почти все операции, затрагивающие тестовую или производственную среду, выполняются с использованием скриптов, а не через портал разработчиков.

Комментарий По:

Мы используем скрипты, поскольку они гарантируют повторяемость операций. Мы планируем усовершенствовать некоторые из наших скриптов, чтобы регистрировать, когда они запускались, кем и с какими параметрами.

Примечание.
Сведения о Windows Azure management API см. в главе 5, «Этап 2. Автоматизация развертывания и использование хранилища Windows Azure», этого руководства.

В компании Adatum только ключевые разработчики и ключевые члены группы тестирования имеют доступ к тестовой среде Windows Azure. А к производственной среде Windows Azure имеют доступ только два ключевых человека из отдела операций.

Идентификатор Windows Live ID и Certificate API, используемые для доступа к тестовой среде, отличаются от тех, которые используются для доступа к производственной системе.

Porudction (производественная) и staging среды

В Windows Azure приложение можно развернуть в staging или production области. Обычный сценарий включает первоначальное развертывание в staging области и последующий перенос новой версии в production область в нужное время. Эти две области являются идентичными средами. Единственное отличие заключается в URL-адресе для доступа к ним. В straging области хранения URL-адрес будет непонятной строкой вроде http://bb7ea70c3be24eb08a08b6d39f801985.cloudapp.net, тогда как в production области это будет понятный адрес вроде http://aexpense.cloupapp.net. Это позволяет тестировать новые и обновленные приложения в закрытой среде, о которой больше никто не знает. Также можно менять местами содержимое prodction и staging области хранения, что позволяет легко выполнять откат приложения к предыдущей версии.

С помощью замены можно быстро откатить изменение в производственной среде.

Комментарий Бхарата:

Не забывайте, что корпорация Microsoft выставляет счета как за использование staging области хранения, так и за использование prodctuib области.

Примечание.
Замена происходит практически мгновенно, поскольку она состоит просто в замене DNS-записей для двух областей в Windows Azure.

Сведения о последовательных обновлениях в нескольких экземплярах роли и частичных обновлениях приложения, которые обновляют отдельные роли, см. в документации MSDN по адресу http://msdn.microsoft.com/ru-ru/library/ee517254.aspx.

Развертывание

Для развертывания приложения в Windows Azure необходимо загрузить два файла: файл пакета службы, содержащий все файлы приложения, и файл конфигурации службы, содержащий данные конфигурации приложения. Файл пакета службы можно сформировать либо с помощью служебной программы Cspack.exe, либо с помощью среды Visual Studio, если установлены инструменты Windows Azure для Microsoft Visual Studio.

Комментарий Маркуса:

Для автоматизации формирования файла пакета развертывания и файла конфигурации мы используем программу командной строки на сервере сборки.

При развертывании компания Adatum использует один и тот же файл пакета в тестовой и производственной среде, но меняет файл конфигурации. Для приложения aExpense ключевое различие между содержимым файлов конфигурации тестовой и производственной среды состоит в строках подключения к хранилищу. Эти сведения являются уникальными для каждой подписки Windows Azure, в них используются сформированные случайным образом ключи доступа. К ключам доступа к хранилищу для производственной среды имеют доступ только два ведущих сотрудника отдела эксплуатации, что делает невозможным случайное использование производственного хранилища во время тестирования кем-нибудь еще. Группа Adatum использует набор скриптов для выполнения развертывания. Одной из задач этих скриптов является замена строк подключения эмулятора хранилища в файле конфигурации на строки подключения для хранилища на основе облака.

Примечание.
Дополнительные сведения о скриптах развертывания см. в главе 5, «Этап 2. Автоматизация развертывания и использование хранилища Windows Azure».

Тестирование

Перед развертыванием приложения в реальной производственной среде компания Adatum тестирует свои приложения на основе облака в нескольких различных активных средах.

Разработчики выполняют модульные и специализированные тесты на локальному эмуляторе Windows Azure на своих локальных компьютерах. Несмотря на то, что эмулятор отличается от облачной среды, он подходит для тестирования разработчиками их собственного кода. Сервер

сборки также проводит ряд тестов как часть стандартного процесса построения. Этот метод ничем не отличается от обычных методов разработки локальных приложений.

Большинство тестов можно выполнить с помощью эмулятора Windows Azure.

Группа тестирования также выполняет большинство тестов с помощью локального эмулятора Windows Azure. В среде тестирования Windows Azure приложение развертывается только для проверки его итоговой конфигурации перед передачей приложения группе администраторов для развертывания в производственной среде. Это позволяет свести к минимуму затраты, связанные со средой тестирования, за счет сокращения времени развертывания приложения в облаке.

Комментарий Яны:

Приложение можно развернуть в тестовой среде Windows Azure только на время проведения испытаний. Плата взимается только за то время, в течение которого используется среда. Но оплата осуществляется по часам, поэтому 15-минутный тест будет стоить столько же, сколько и 55-минутный.

Дополнительные сведения

В MSDN® содержится много информации о Windows Azure. Дополнительные сведения о развертывании приложений в Windows Azure см. на странице http://msdn.microsoft.com/ru-ru/library/dd163896.aspx.

Страница «Приступая к работе с платформой Windows Azure» (http://www.microsoft.com/windowsazure/getstarted) содержит ссылки на множество ресурсов, где можно узнать об управлении приложениями в Windows Azure.

Глава 8. Добавление других задач и настройка приложения (этап 4)

В этой главе описывается функция экспорта данных, добавленная сотрудниками Adatum в приложение aExpense, а также некоторые изменения, внесенные после тестирования производительности. В этой главе заново пересматриваются абстракции рабочих ролей, представленные на последнем этапе проекта, чтобы проиллюстрировать выполнение нескольких задач в одной рабочей роли. Кроме того, здесь описываются некоторые вопросы, связанные с архитектурой, например управление состояниями сеансов, а также выбор партиции и ключей строк для хранилища таблиц технологической платформы Windows® Azure™.

Исходные условия

На этом этапе проекта миграции приложения aExpense группа сотрудников Adatum реализовала последнюю ключевую функцию приложения. Приложение aExpense должно сформировать файл данных, в котором обобщаются утвержденные подтверждения расходов. Локальная система платежей Adatum импортирует этот файл данных, а затем производит выплаты сотрудникам компании Adatum. В итоговой локальной версии aExpense используется плановое задание служб SQL Server® Integration Services, с помощью которого формируется выходной файл и задается состояние подтверждения расходов «В обработке» после его экспорта. Локальное приложение

также импортирует данные из системы обработки платежей, чтобы обновить состояния подтверждений расходов после того, как система обработки платежей произведет платеж. Этот процесс импорта не входит в текущий этап проекта миграции.

Процесс экспорта в приложении aExpense

На этом этапе в компании Adatum также проанализировали результаты тестирования производительности приложения и внесли несколько изменений на основе полученных результатов.

Компания Adatum внесла несколько изменений в приложение aExpense после тестирования производительности приложения.

Цели и требования

Процесс экспорта в облачной версии приложения aExpense должен обеспечивать достижение нескольких целей. Во-первых, затраты на процесс экспорта следует свести к минимуму, при этом он не должен оказывать отрицательного влияния на производительность приложения для пользователей. Кроме того, процесс экспорта должен быть надежным, необходимо обеспечить возможность восстановления после сбоя без ущерба для целостности данных приложения aExpense или точности экспортируемых данных.

Комментарий Маркуса:

Утвержденные подтверждения деловых расходов могут находиться в любом месте таблицы. Мы стремимся избежать снижения производительности из-за сканирования всей таблицы.

В решении также должен быть ответ на вопрос, как инициировать экспорт, и сравнение проведения операцию вручную или по плану. В последнем случае сотрудникам Adatum необходимо разработать механизм планирования задач в Windows Azure.

Последнее требование заключается во включении механизма переноса данных из среды облака в локальную среду, где ими сможет воспользоваться приложение обработки платежей.

Комментарий По:

В Adatum должны автоматизировать процесс загрузки данных о подтверждениях расходов для ввода в систему обработки платежей.

Обзор решения

Процесс экспорта должен учитывать три момента: как инициировать процесс, как формировать данные и как загружать данные из облака.

Инициирование процесса экспорта данных

Самым простым вариантом инициирования экспорта данных является создание веб-страницы, которая бы возвращала данные по требованию. У этого подхода есть несколько потенциальных недостатков. Во-первых, он увеличивает нагрузку на веб-сервер, что теоретически может сказаться

на других пользователях системы. В случае aExpense, вероятно, это не будет иметь значения, поскольку для формирования отчета требуются незначительные вычислительные ресурсы. Во-вторых, если процесс формирования данных сложен, а объемы данных значительны, веб-страница должна обрабатывать и истечение времени ожидания. Но для aExpense маловероятно, что это станет существенной проблемой. Наиболее значимым недостатком такого решения для приложения aExpense является то, что действующая архитектура хранилища подтверждений расходов оптимизирована для обновления и получения отдельных подтверждений расходов по идентификатору пользователя. А процессу экспорта потребуется доступ к данным подтверждения расходов по дате и состоянию. В отличие от SQL Azure, где можно определять несколько индексов в таблице, в хранилище таблиц Windows Azure у каждой таблицы есть только один индекс.

Комментарий Яны:

Правильный выбор ключей PartitionKey и RowKey для таблиц служит важнейшим фактором обеспечения производительности приложения. Любой процесс, которому требуется выполнить сканирование таблиц по всем партициям, будет медленным.

На рис. 2 показан второй вариант инициирования экспорта данных. У каждой задачи есть выделенная рабочая роль, поэтому сжатие изображений и формирование миниатюр будет выполняться задачей 1 в рабочей роли 1, а экспорт данных будет выполняться задачей 2 в рабочей роли 2. Это также несложно реализовать, но в случае с приложением aExpense, где процесс экспорта будет выполняться два раза в месяц, нецелесообразно нести затраты на содержание отдельного экземпляра ролей. Если задача выполняется чаще и если для ее выполнения требуются значительные вычислительные мощности, можно рассмотреть возможность использования дополнительной рабочей роли.

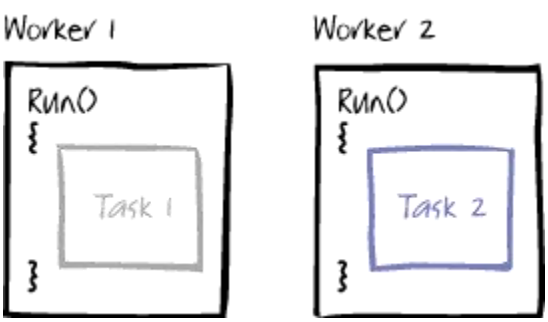


Рис. 2

Отдельные рабочие роли для каждой задачи

Комментарий Бхарата:

Следует постараться полностью задействовать имеющиеся вычислительные узлы. Помните, что вы платите за развернутый экземпляр роли независимо от того, выполняет он какую-либо работу или нет. Также можно выбрать вычислительный экземпляр большего размера, если требуется выполнять больший объем работы в одном экземпляре роли.

На рис. 3 показан третий вариант, когда процесс экспорта данных выполняет дополнительная задача в существующей рабочей роли. При таком подходе используются существующие вычислительные ресурсы, он подходит для тех случаев, когда для выполнения задач не требуются значительные вычислительные мощности. В настоящее время Windows Azure SDK не содержит каких-либо абстракций задач, поэтому вам необходимо разработать или подыскать платформу для выполнения обработки на основе задач. Для определения задач в приложении aExpense сотрудники Adatum будут использовать классы «соединительного кода», описанные в главе 6, «Этап 3. Загрузка изображений и добавление рабочей роли». Проектирование и построение платформы такого типа не представляет особой сложности, однако при этом необходимо использовать всю имеющуюся логику обработки ошибок и планирования.

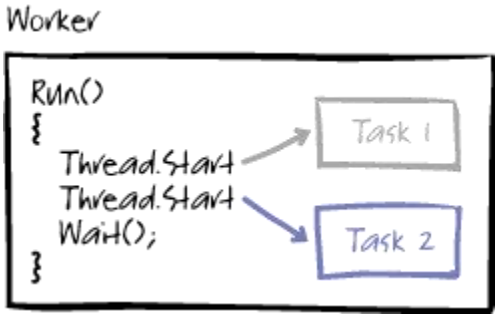


Рис. 3

Несколько задач в одной рабочей роли

У компании Adatum уже есть несколько простых абстракций, позволяющих выполнять несколько задач в одной рабочей роли.

Комментарий Маркуса:

Платформа Windows Azure может осуществлять мониторинг только на уровне рабочей роли, в случае сбоя этой роли система по возможности старается перезапустить ее. В случае сбоя одного из потоков обработки задач разрешать эту ситуацию должны вы сами.

Формирование данных для экспорта

Сотрудники Adatum решили разделить процесс создания отчета по расходам на два этапа. На первом этапе модель данных делается плоской, после чего данные помещаются в таблицу Windows Azure. В этой таблице дата утверждения подтверждения расходов используется в качестве ключа партиции, идентификатор расхода — в качестве ключа строки, кроме того, в ней хранятся итоговые значения подтверждений расходов. На втором этапе эта таблица считывается и формируется большой двоичный объект Windows Azure, содержащий данные, готовые к экспорту в CSV-файл. В Adatum реализовали каждый из этих двух этапов в виде задачи с помощью «соединительного кода», описанного в главе 6, «Этап 3. Загрузка изображений и добавление рабочей роли». На рис. 4 показана работа задачи, добавляющей данные в таблицу Windows Azure.

Комментарий Бхарата:

У хранилища таблиц Windows Azure нет всех функций, которыми обладает реляционная база данных. Вам же может потребоваться несколько таблиц, которые будут по-разному представлять одни и те же данные, исходя из потребностей приложения. Для этого необходимо иметь возможность формировать производную таблицу идемпотентным образом на основе данных, хранящихся в исходной таблице. Хранилище таблиц обходится дешево!

Сначала руководитель утверждает подтверждение расходов. При этом сообщение, содержащее идентификатор этого подтверждения расходов и дату утверждения, добавляется в очередь (1), а состояние этого подтверждения в хранилище таблиц обновляется (2). Задача получает это сообщение из очереди, вычисляет общее значение подтвержденных расходов по его составляющим, после чего сохраняет это значение в виде одной строки в таблице «Expense Export». Эта задача также переводит подтверждение расходов в состояние «В обработке», а затем удаляет сообщение из очереди.

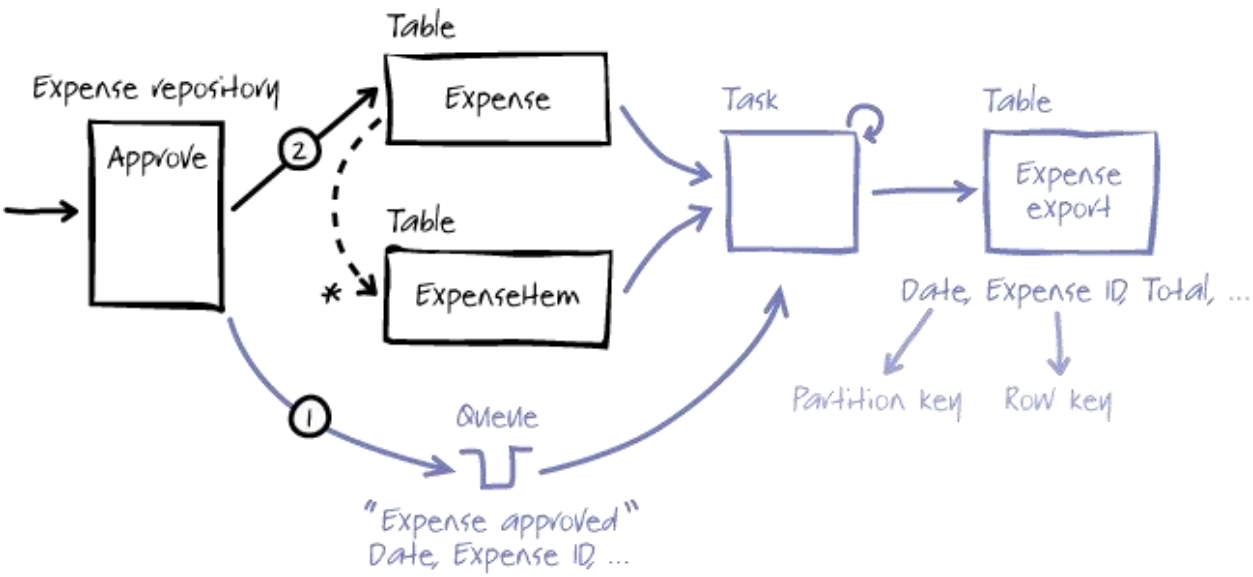


Рис. 4

Формирование таблицы отчета о расходах

Экспорт данных отчета

В компании Adatum рассматривали два варианта экспорта данных. Первым вариантом была веб-страница, позволяющая пользователю загружать данные отчета о расходах в виде файла.
Эта страница запрашивала бы таблицу отчета о расходах по дате и формировала бы CSV-файл, который может импортировать система обработки платежей. Этот вариант показан на рис. 5.

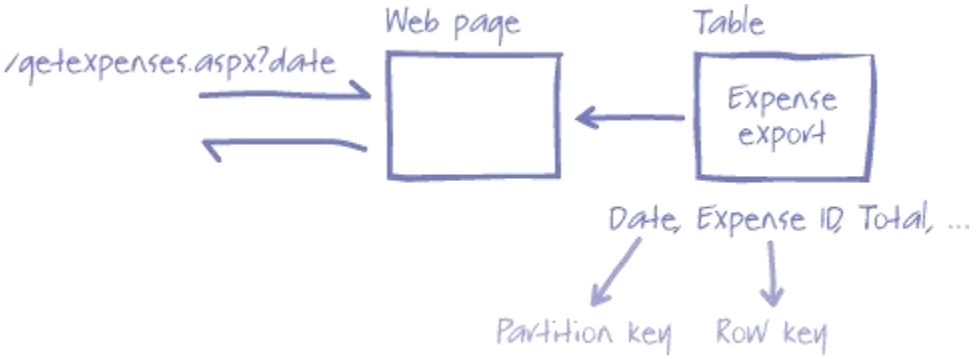


Рис. 5

Загрузка отчета о расходах с веб-страницы

Второй вариант, показанный на рис. 6, заключался в создании другого, выполняемого по расписанию задания в рабочем процессе для формирования файла в готовом для загрузки хранилище больших двоичных объектов. Компания Adatum изменит локальную систему обработки платежей так, чтобы этот файл загружался прежде, чем импортировался. В Adatum выбрали этот вариант, поскольку он позволяет планировать это задание на время наименьшей загрузки системы, чтобы избежать снижения производительности приложения во время работы пользователей. Локальное приложение может получать доступ к хранилищу больших двоичных объектов напрямую, не задействуя веб-роль или рабочую роль Windows Azure.

Комментарий По:

Такой подход позволяет упростить автоматизацию загрузки и обеспечить своевременное получение данных для обработки платежей.

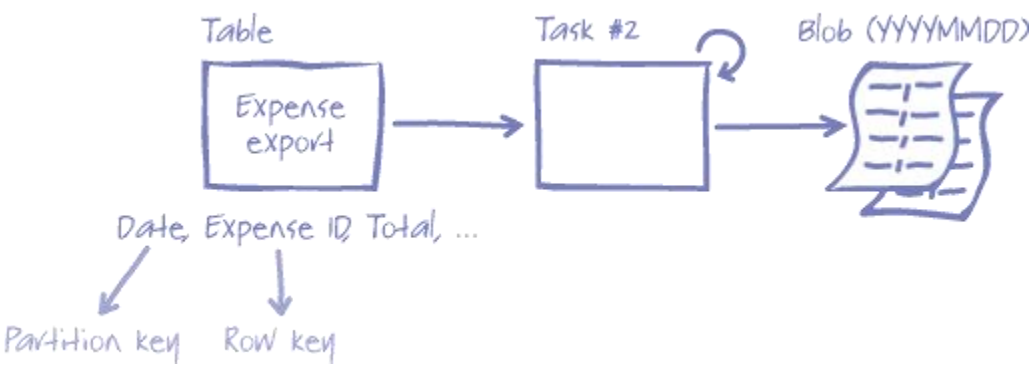


Рис. 6

Создание отчета о расходах в хранилище больших двоичных объектов

Для обеспечения этого процесса сотрудникам Adatum пришлось слегка изменить «соединительный код» рабочей роли. В исходной версии «соединительного кода» задача запускалась сообщением в очереди, теперь же приложению также требуется возможность планировать задачи.

Комментарий Маркуса:

Нам пришлось слегка изменить классы «соединительного кода», чтобы обеспечить возможность планирования задач.

Реализация

Теперь пришло время рассмотреть эти изменения более подробно. При изучении этого раздела вам может понадобиться скачать решение для систем разработки Microsoft ® Visual Studio ® со страницы http://wag.codeplex.com/. Это решение содержит реализацию приложения aExpense после изменений, внесенных на данном этапе. Если вы не заинтересованы в изучении таких механизмов, перейдите к следующему разделу.

Формирование таблицы отчета о расходах

Задача, которая выполняет эту операцию, использует «соединительный код» рабочей роли, описанный в главе 6, «Этап 3. Загрузка изображений и добавление рабочей роли». В этой главе затрагиваются вопросы, связанные с реализацией задачи и структурой таблицы. «Соединительный код» не рассматривается.

Комментарий Маркуса:

Для этой задачи мы можем использовать «соединительный код» рабочей роли без изменений.

Это первая из двух задач, формирующих данные утвержденных расходов для экспорта. Она отвечает за формирование плоской таблицы с этими данными в хранилище таблиц Windows Azure. В следующем образце кода показано начало процесса экспорта отчета о расходах в классе ExpenseRepository, где метод UpdateApproved добавляет сообщение в очередь и обновляет свойство Approved записи заголовка расходов.

Копировать

```
public void UpdateApproved(Expense expense)

{

  var context = new ExpenseDataContext(this.account);


  ExpenseRow expenseRow =

    GetExpenseRowById(context, expense.Id);

  expenseRow.Approved = expense.Approved;
```

```
    var queue = new AzureQueueContext(this.account);
    queue.AddMessage(new ApprovedExpenseMessage {
        ExpenseId = expense.Id.ToString(),
        ApproveDate = DateTime.UtcNow });

    context.UpdateObject(expenseRow);
    context.SaveChanges();
}
```

В этом коде используется новый тип сообщения ApprovedExpenseMessage, производный от класса «соединительного кода» BaseQueueMessage. В следующем образце кода показаны два свойства класса ApprovedExpenseMessage.

Копировать

```
[DataContract]
public class ApprovedExpenseMessage : BaseQueueMessage
{
    [DataMember]
    public string ExpenseId { get; set; }

    [DataMember]
    public DateTime ApproveDate { get; set; }
}
```

В следующем коде рассматривается способ, с помощью которого метод ProcessMessage в классе ExpenseExportJob получает сообщение из очереди и создает новую сущность ExpenseExport для сохранения в хранилище таблиц.

Комментарий Яны:

Перед сохранением данных в промежуточной таблице мы делаем их плоскими и вычисляем итог подтвержденных расходов. В этой таблице данные структурированы именно в том виде, в котором они требуются нам для экспорта.

Копировать

```
public override bool ProcessMessage(
```

```
    ApprovedExpenseMessage message)
{
    try
    {
        Expense expense = this.expenses.GetExpenseById(
            new ExpenseKey(message.ExpenseId));

        if (expense == null)
        {
            return false;
        }

        // Если расходы не были обновлены, а сообщение было
        // сохранено, его необходимо удалить.
        if (!expense.Approved)
        {
            return true;
        }

        double totalToPay = expense.Details.Sum(x => x.Amount);
        var export = new ExpenseExport
            {
                ApproveDate = message.ApproveDate,
                ApproverName = expense.ApproverName,
                CostCenter = expense.CostCenter,
                ExpenseId = expense.Id,
                ReimbursementMethod = expense.ReimbursementMethod,
                TotalAmount = totalToPay,
                UserName = expense.User.UserName
```

```
            };
        this.expenseExports.Save(export);
    }
    catch (InvalidOperationException ex)
    {
        var innerEx =
            ex.InnerException as DataServiceClientException;
        if (innerEx != null &&
            innerEx.StatusCode == (int)HttpStatusCode.Conflict)
        {
            // Данные уже существуют, поэтому можно возвратить значение true,
            // поскольку они уже были обработаны раньше.
            return true;
        }
        Log.Write(EventKind.Error, ex.TraceInformation());
        return false;
    }

    return true;
}
```

Если произойдет сбой этого метода по любой причине, за исключением конфликта при вставке, классы «соединительного кода» обеспечат наличие сообщения в очереди. Когда метод ProcessMessage пытается обработать сообщение из очереди во второй раз, вставка в таблицу отчета о расходах завершается с ошибкой повторяющегося значения ключа и внутреннее исключение сообщает об этом как о конфликте своим свойством StatusCode. В этом случае метод может благополучно возвратить значение true.

Комментарий Яны:

Нам необходимо обеспечить надежность этого процесса. Мы не хотим терять подтверждения расходов и не хотим платить кому-то дважды.

Если свойство Approved объекта Expense имеет значение false, то это указывает на ошибку в работе метода UpdateApproved, возникшую после добавления сообщения в очередь, но до

обновления таблицы. В этом случае метод ProcessMessage удаляет сообщение из очереди без его обработки.

Ключом партиции таблицы «Expense Export» является дата утверждения расхода, при этом ключ строки является идентификатором этого элемента. Это позволяет оптимизировать доступ к данным запросам, в которых дата утверждения используется в предложении where, что и требуется процессу экспорта.

Комментарий Бхарата:

Для оптимизации запросов к данным выбирайте ключи секций и строк. Теоретически вы должны иметь возможность включать ключ партиции в блок where запроса.

Экспорт данных

Это вторая из двух задач, формирующих данные утвержденных подтверждений расходов для экспорта. Она отвечает за создание большого двоичного объекта Windows Azure, содержащего CSV-файл утвержденных подтверждений расходов.

Задача, формирующая большой двоичный объект с данными отчета о расходах, немного отличается от двух других задач в приложении aExpense. Другие задачи опрашивают очередь, чтобы определить, имеется ли для них какая-либо работа. Эта же задача запускается по расписанию, согласно которому она выполняется в назначенное время. Сотрудникам Adatum пришлось изменить классы «соединительного кода» рабочей роли для поддержки запланированных задач.

Теперь классы «соединительного кода» рабочей роли поддерживают не только задачи, запускаемые сообщением из очереди, но и запланированные задачи.

Комментарий Маркуса:

Нам удалось расширить приложение и предоставить локальному приложению возможность формировать нерегламентированные отчеты по расходам, позволив ему помещать сообщения в очередь Windows Azure. После этого мы могли бы создать задачу, которая генерирует данные отчета, получив сообщение из очереди.

Абстрактный класс JobProcessor, реализующий интерфейс IJobProcessor, можно использовать для определения новых запланированных задач. В следующем примере кода показан класс JobProcessor.

Копировать

```
public abstract class JobProcessor: IJobProcessor

{

   private bool keepRunning;
```

```
protected JobProcessor(int sleepInterval)
{
   if (sleepInterval <= 0)
   {
      throw new
         ArgumentOutOfRangeException("sleepInterval");
   }

   this.SleepInterval = sleepInterval;
}

protected int SleepInterval { get; set; }

public void Run()
{
   this.keepRunning = true;
   while (this.keepRunning)
   {
      Thread.Sleep(this.SleepInterval);
      this.RunCore();
   }
}

public void Stop()
{
   this.keepRunning = false;
}
```

```
    protected abstract void RunCore();
}
```

В этой реализации довольно сложно определить точное время выполнения запланированной задачи. Интервал между задачами — это время ожидания плюс время выполнения самой задачи. Если требуется, чтобы задача выполнялась в определенное время, необходимо определить, сколько времени выполняется задача, а затем вычесть это значение из интервала ожидания.

Примечание.

Класс BaseJobProcessor, определяющий задачи, которые считывают сообщения из очередей, служит расширением класса JobProcessor.

В приложении aExpense класс ExpenseExportBuilderJob расширяет класс JobProcessor с целью определения запланированной задачи. Класс ExpenseExportBuilderJob, показанный в следующем примере кода, определяет задачу, которая формирует данные отчета о расходах и сохраняет их в виде большого двоичного объекта. В этом классе переменная expenseExports ссылается на таблицу утвержденных подтверждений расходов, а переменная exportStorage ссылается на данные отчета для загрузки из хранилища больших двоичных объектов. Вызов конструктора базового класса определяет интервал выполнения задания. В методе RunCore код сначала получает все утвержденные подтверждения расходов из таблицы экспорта с учетом даты задания. Затем код добавляет запись CSV к данным экспорта в хранилище больших двоичных объектов для каждого утвержденного подтверждения расходов. И наконец, код удаляет из таблицы все записи, скопированные в хранилище больших двоичных объектов.

Примечание.

Следующий код задает запланированному интервалу небольшое значение для тестирования и демонстрации. Чтобы создать реальное расписание, это значение следует изменить.

Копировать

```
public class ExpenseExportBuilderJob: JobProcessor
{
    private readonly ExpenseExportRepository expenseExports;
    private readonly ExpenseExportStorage exportStorage;

    public ExpenseExportBuilderJob(): base(100000)
    {
        this.expenseExports = new ExpenseExportRepository();
        this.exportStorage = new ExpenseExportStorage();
```

```
}

protected override void RunCore()
{
    DateTime jobDate = DateTime.UtcNow;
    string name = jobDate.ToExpenseExportKey();

    IEnumerable<ExpenseExport> exports =
        this.expenseExports.Retreive(jobDate);
    if (exports == null || exports.Count() == 0)
    {
        return;
    }

    string text = this.exportStorage.GetExport(name);
    var exportText = new StringBuilder(text);
    foreach (ExpenseExport expenseExport in exports)
    {
        exportText.AppendLine(expenseExport.ToCsvLine());
    }

    this.exportStorage.AddExport(name,
        exportText.ToString(), "text/plain");

    // Удаление экспортированных данных.
    foreach (ExpenseExport exportToDelete in exports)
    {
        try
        {
```

```
            this.expenseExports.Delete(exportToDelete);
        }
        catch (InvalidOperationException ex)
        {
            Log.Write(EventKind.Error,
                ex.TraceInformation());
        }
      }
   }
}
```

Если этот процесс завершится ошибкой до того, как будут удалены все утвержденные подтверждения расходов из таблицы экспорта, все неудаленные утвержденные подтверждения расходов будут повторно экспортированы при следующем запуске этой задачи. Однако в экспортированных данных CSV имеется идентификатор расходов и дата утверждения подтверждения расходов, поэтому локальная система обработки платежей сможет определить дублированные элементы.

В следующем коде показаны запросы, которые метод RunCore формирует для получения утвержденных подтверждений расходов и затем удаляет после копирования в экспортируемый большой двоичный объект. Поскольку эти запросы используют дату задания для определения партиции, в которых следует искать, они работают быстро и эффективно.

Копировать

```
public IEnumerable<ExpenseExport> Retreive(DateTime jobDate)
{
   var context = new ExpenseDataContext(this.account);
   string compareDate = jobDate.ToExpenseExportKey();
   var query = (from export in context.ExpenseExport
        where export.PartitionKey.CompareTo(compareDate) <= 0
        select export).AsTableServiceQuery();

   var val = query.Execute();
   return val.Select(e => e.ToModel()).ToList();
}
```

```
public void Delete(ExpenseExport expenseExport)
{
   var context = new ExpenseDataContext(this.account);
   var query = (from export in context.ExpenseExport
     where export.PartitionKey.CompareTo(
       expenseExport.ApproveDate.ToExpenseExportKey()) == 0 &&
       export.RowKey.CompareTo(
       expenseExport.ExpenseId.ToString()) == 0
     select export).AsTableServiceQuery();
   ExpenseExportRow row = query.Execute().SingleOrDefault();
   if (row == null)
   {
      return;
   }

   context.DeleteObject(row);
   context.SaveChanges();
}
```

Тестирование производительности, настройка, предстоящие задачи

В ходе работ этого этапа сотрудники Adatum проанализировали результаты тестирования производительности приложения и внесли ряд изменений в приложение aExpense. Они также задокументировали несколько важнейших «недостающих» частей приложения, которыми им необходимо будет заняться на следующем этапе реализации проекта.

После проведения тестирования производительности сотрудники Adatum внесли изменения в приложение aExpense.

Хранение состояния сеанса

Готовясь к тестированию производительности и стремясь обеспечить масштабируемость приложения, сотрудники Adatum немного изменили метод обработки приложением состояния сеанса. Страница AddExpense.aspx использует состояние сеанса для ведения списка подтверждений расходов до того, как готовое подтверждение деловых расходов будет сохранено пользователем. Во время разработки и тестирования в приложении применялся используемый по

умолчанию поставщик состояния сеанса, находящийся в памяти. Этот поставщик не подходит для работы с несколькими экземплярами веб-ролей, поскольку состояние сеанса не синхронизируется между экземплярами, а программа распределения нагрузки Windows Azure может направить запрос любому доступному экземпляру. В Adatum рассмотрели несколько вариантов решения этой проблемы в приложении aExpense.

Комментарий Бхарата:

Если приложение надо расширить для работы с более чем одной веб-ролью, любое состояние сеанса должно быть общим для всех экземпляров веб-роли.

Первый вариант заключался в использовании состояния представления ASP.NET вместо состояния сеанса, чтобы приложение поддерживало свои данные о состоянии на клиенте. Это решение будет работать в том случае, когда у приложения есть несколько экземпляров веб-ролей, поскольку приложение не хранит каких-либо данных о состоянии на сервере. Тот факт, что отсканированные изображения хранятся приложением aExpense в состоянии до момента сохранения всего подтверждения расходов, означает, что состояние может быть довольно большим. В этом случае использование состояния представления было бы неэффективным решением, так как пришлось бы перемещать данные в состояние представления по сети, что занимало бы пропускную способность сети и негативно сказывалось на производительности приложения.

Комментарий Маркуса:

Использование состояния представления ASP.NET — это отличное решение при обработке небольшого объема данных. Но случай приложения aExpense другой, поскольку в данные о состоянии входят изображения.

Второй вариант предполагает продолжить использовать состояние сеанса, но задействовать поставщика, который будет сохранять состояние в хранилище на сервере, доступ к которому возможен из нескольких экземпляров веб-ролей. В Adatum могут либо создать собственный настраиваемый поставщик состояния сеанса, использующий хранилище SQL Azure или Windows Azure, либо использовать поставщика состояния сеанса кэширования Windows Azure Cachung. В Adatum планируют проанализировать относительные затраты, связанные с использованием поставщика состояния сеанса Windows Azure Chaching и собственного поставщика, а также оценить их производительность: в обоих случаях использование поставщика обеспечивается средствами конфигурации, для этого не требуется вносить изменения в код приложения aExpense.

Примечание.
Дополнительные сведения о поставщике состояния сеанса кэширования Windows Azure Caching см. на странице http://msdn.microsoft.com/ru-ru/library/gg185668.aspx.

Использование таблицы с несколькими схемами

Первоначально в Adatum собирались использовать две таблицы для хранение данных о расходах: таблицу с заголовками и таблицу со сведениями, записи в которых были бы связаны по внешнему

ключу. Преимущество такого подхода заключается в простоте, так как у каждой таблицы есть собственная, отдельная схема. Однако поскольку транзакции в хранилище Windows Azure не могут охватывать таблицы, то, как описано в главе 6, «Этап 3. Загрузка изображений и добавление рабочей роли», существует возможность сохранения записей сведений в случае возникновения сбоя до того, как приложение aExpense сохранит запись с заголовком. На этом этапе в Adatum реализовали таблицу с несколькими схемами для данных о расходах, чтобы можно было использовать одну транзакцию для сохранения и записей заголовков, и записей сведений.

Примечание.
Описание способа реализации таблицы с несколькими схемами группой Adatum см. в приведенном ниже разделе «Реализация таблицы с несколькими схемами в хранилище таблиц Windows Azure».

Слишком много вызовов метода CreateIfNotExist

Первоначально конструктор класса ExpenseReceiptStorage проверял наличие контейнера для приема расходов и создавал его при необходимости. Этот конструктор вызывается всякий раз, когда приложение создает экземпляр объекта ExpenseRepository или объекта ReceiptThumbnailJob. Методу CreateIfNotExist, который проверяет наличие контейнера, необходимо обменяться данными с сервером хранилища, что влечет за собой затраты на транзакцию с хранилищем. Чтобы избежать лишнего обмена данными, сотрудники Adatum переместили эту логику в класс ApplicationStorageInitializer: метод Application_Start в файле Global.asax.cs вызывает метод Initialize из этого класса, когда от клиента поступает самый первый запрос.

Примечание.
Необходимо следить за тем, чтобы не открыть контекст запроса или не вызвать методы класса RoleManager из метода Application_Start. Дополнительные сведения см. на странице http://msdn.microsoft.com/ru-ru/library/microsoft.servicehosting.serviceruntime.rolemanager.aspx.

Запрет пользователям загружать большие изображения

Чтобы пользователи не могли загрузить в приложение aExpense большие изображения сканированных чеков, в компании Adatum настроили приложение таким образом, чтобы максимально возможный размер файлов, загружаемых на страницу AddExpense.aspx, был равен 1024 килобайтам (КБ). В следующем примере кода отображена настройка в файле Web.config.

Копировать

```
<location path="AddExpense.aspx">
 <system.web>
  <authorization>
   <allow roles="Employee" />
   <deny users="*"/>
```

```
  </authorization>

  <!—
    Максимальный запрос, которому разрешается отправлять крупное изображение чека.
    -->
  <httpRuntime maxRequestLength="1024"/>
 </system.web>
</location>
```

Проверка данных, вводимых пользователем

Облачная версия приложения aExpense не проводит всеобъемлющие проверки вводимых пользователем данных, выявляя недопустимые или опасные элементы. В файле AddExpense.aspx выполняется обычная проверка, определяющая длину вводимых пользователем данных, но группе Adatum следует добавить дополнительные проверки в метод OnAddNewExpenseItemClick из файла AddExpense.aspx.cs.

Разбиение на страницы и сортировка на странице Default.aspx

Тестирование производительности показало, что время ответа для страницы Default.aspx ухудшалось по мере добавления тестовым скриптом новых и новых подтверждений расходов одного пользователя. Происходит это потому, что в текущей версии страницы Default.aspx отсутствует механизм разбиения на страницы, поэтому на ней всегда отображаются все подтверждения расходов для пользователя. В качестве временной меры в Adatum изменили запрос LINQ, который получает подтверждения расходов по пользователю, включив в него предложение Take(10) для того, чтобы приложение запрашивало только первые 10 подтверждений расходов. На следующем этапе этого проекта сотрудники Adatum добавят к странице Default.aspx функцию разбиения на страницы.

Примечание.

Описание того, как в Adatum могли бы реализовать на страницах Default.aspx и Approve.aspx разбиение на страницы, см. в приведенном ниже разделе «Реализация разбиения на страницы в хранилище таблиц Windows Azure».

В Adatum также обнаружили, что GUID, выбранный изначально для использования в качестве ключа строк таблицы расходов, не является оптимальным вариантом. По умолчанию подтверждения расходов должны располагаться в обратном хронологическом порядке, поскольку после реализации разбиения на страницы большинство пользователей захотят видеть вверху списка или на первой странице подтверждения расходов, которые были добавлены недавно. На странице утверждения расходов также будет использоваться такой порядок.

Комментарий Маркуса:

Мы обнаружили, что при использовании GUID в качестве ключа строк подтверждений расходов данные возвращаются не в том порядке. Мы переработали код, чтобы использовать в качестве альтернативы класс ExpenseKey. Этот класс формирует обратное значение отметки времени.

Чтобы реализовать это изменение, в Adatum изменили класс Expense в пространстве имен AExpense.Model, изменив тип свойства Id с Guid на ExpenseKey. В следующем примере кода показано, как статическое свойство Now класса ExpenseKey формирует обратный счетчик тактов для использования в свойстве InvertedTicks.

Копировать

```
public static ExpenseKey Now
{
  get
  {
    return new ExpenseKey(
      string.Format("{0:D19}",
      DateTime.MaxValue.Ticks - DateTime.UtcNow.Ticks));
  }
}
```

Теперь запрос, получающий подтверждения расходов по пользователю, возвращает их в обратном хронологическом порядке.

Изменения конфигурации System.Net

В следующем примере кода показаны два изменения конфигурации, внесенные сотрудниками Adatum в приложение aExpense для повышения его производительности.

Копировать

```
<system.net>
 <settings>
  <servicePointManager expect100Continue="false" />
 </settings>
 <connectionManagement>
  <add address = "*" maxconnection = "24" />
 </connectionManagement>
</system.net>
```

Первое изменение отключает функцию «Expect 100-continue». Если эта функция включена, то, когда приложение отправляет запрос PUT или POST, оно может задержать отправку полезных данных, отправив заголовок «Expect 100-continue». Когда сервер получает это сообщение, он с помощью сведений из заголовка проверяет возможность выполнения вызова. Если такая возможность имеется, сервер отправляет клиенту код состояния 100. Затем клиент отправляет оставшуюся часть полезных данных. Это означает, что клиент может обнаруживать многие стандартные ошибки, не отправляя полезные данные. Если клиент хорошо протестирован и вы уверены, что он не будет отправлять неверные запросы, функцию «Expect 100-continue» можно отключить, чтобы сократить количество возвратов на сервер. Это особенно полезно, когда клиент отправляет много сообщений с полезными данными небольшого объема, например когда клиент использует таблицу или службу очереди.

Второе изменение конфигурации увеличивает максимальное количество поддерживаемых веб-сервером соединений от значения по умолчанию, равного 2. Это значение слишком небольшое, а проблема проявляется в виде сообщений «Базовое соединение было закрыто» (“Underlying connection was closed”).

Примечание.
Точное число, которое следует использовать для этого параметра, зависит от конкретного приложения. Полезные сведения о том, как задавать значение этого параметра для серверных приложений, см. на странице http://support.microsoft.com/kb/821268. Его также можно задать для определенного URI, указав этот URI вместо «*».

Оптимизация службы данных WCF

Из-за известной проблемы с производительностью службы данных WCF в Adatum определили делегата ResolveType для класса DataServiceContext в приложении aExpense. Без этого делегата производительность запросов снижается по мере роста числа сущностей, возвращаемых запросом. В следующем примере кода показано определение делегата.

Комментарий Маркуса:

В код служб WCF Data Services внесен ряд изменений для улучшения производительности.

Копировать

```
private static Type ResolveEntityType(string name)
{
  var tableName = name.Split(new[] { '.' }).Last();
  switch (tableName)
  {
```

```
        case ExpenseTable:
            return typeof(ExpenseRow);
        case ExpenseItemTable:
            return typeof(ExpenseItemRow);
        case ExpenseExportTable:
            return typeof(ExpenseExportRow);
    }

    throw new ArgumentException(
        string.Format(
            CultureInfo.InvariantCulture,
            "Не удалось разрешить имя таблицы '{0}'
            для известного типа сущности.", name));
}
```

Примечание.
Вместо использования делегата типа ResolveType следует убедиться в том, что имена классов сущностей точно соответствуют именам таблиц, чтобы избежать проблем с производительностью.

Для дальнейшей оптимизации кода клиента в службы WCF Data Services добавлен домен Adatum путем установки для параметра MergeOption значения NoTracking для запросов в классе ExpenseRepository. Если сущности, которые извлекаются службами WCF Data Services, не меняются, нет необходимости в объекте DataContext для отслеживания изменений сущностей.

Реализация таблицы с несколькими схемами в хранилище таблиц Windows Azure

Первоначально в домене Adatum использовались две таблицы для хранения расходов: одна таблица для записей заголовков, другая — для записей сведений. Это означало, что Adatum требовалось две транзакции для сохранения расходов, что приводило к риску появления записей сведений, утративших связь с заголовком. Следующий пример кода показывает суть первоначального метода SaveExpense из класса ExpenseRepository — каждый вызов метода SaveChanges является отдельной транзакцией.

Копировать

```
public void SaveExpense(Expense expense)
{
    var context = new ExpenseDataContext(this.account);
```

```
    ExpenseRow expenseRow = expense.ToTableEntity();

    foreach (var expenseItem in expense.Details)
    {
      // создать строку
      var expenseItemRow = expenseItem.ToTableEntity();
      expenseItemRow.PartitionKey = expenseRow.PartitionKey;
      expenseItemRow.RowKey =
        string.Format(CultureInfo.InvariantCulture, "{0}_{1}",
        expense.Id, expenseItemRow.Id);

      context.AddObject(ExpenseDataContext.ExpenseItemTable,
        expenseItemRow);

      …
    }

    context.SaveChanges(SaveChangesOptions.Batch);

    context.AddObject(ExpenseDataContext.ExpenseTable,
      expenseRow);
    context.SaveChanges();

    …
}
```

При чтении этого раздела вам может понадобиться загрузить решение для среды разработки Microsoft® Visual Studio® с сайта http://wag.codeplex.com/. Это решение (в папке Azure-MultiEntitySchema) содержит реализацию класса aExpense после внесения разработчиками Adatum описанных в этом разделе изменений.

Хранение разных типов сущностей в одной таблице позволяет использовать транзакции при работе с несколькими типами сущностей. В следующем примере кода показана новая версия метода SaveExpenses с одной транзакцией.

Копировать

```
public void SaveExpense(Expense expense)
{
   var context = new ExpenseDataContext(this.account);
   IExpenseRow expenseRow = expense.ToTableEntity();
   expenseRow.PartitionKey = ExpenseRepository
      .EncodePartitionAndRowKey(expenseRow.UserName);
   expenseRow.RowKey = expense.Id.ToString();
   context.AddObject(ExpenseDataContext.ExpenseTable,
      expenseRow);

   foreach (var expenseItem in expense.Details)
   {
      // Создать строку с элементом расходов.
      var expenseItemRow = expenseItem.ToTableEntity();
      expenseItemRow.PartitionKey = expenseRow.PartitionKey;
      expenseItemRow.RowKey = string.Format(
         CultureInfo.InvariantCulture, "{0}_{1}", expense.Id,
         expenseItemRow.ItemId);
      context.AddObject(ExpenseDataContext.ExpenseTable,
         expenseItemRow);

      …
   }

   // Сохранить заголовок расходов и сведения о них в одной
```

```
    // пакетной транзакции.
    context.SaveChanges(SaveChangesOptions.Batch);

    …
}
```

Изменения, внесенные разработчиками Adatum в класс aExpense для реализации нескольких схем в одной таблице, легче понять, если сначала рассмотреть, каким образом приложение извлекает данные из таблицы, а затем как данные сохраняются в таблице.

Определение схем

В приложении aExpense два типа сущностей хранятся в таблице Expense: сущности заголовков расходов (определяемые интерфейсом IExpenseRow) и сущности сведений (определяемые интерфейсом IExpenseItemRow). В следующем примере кода показаны эти два интерфейса и интерфейс IRow, который определяет ключ сущности.

Комментарий Маркуса:

Оба типа сущностей используют одну структуру ключа, определяемую интерфейсом IRow.

Копировать

```
public interface IExpenseRow : IRow
{
    string Id { get; set; }
    string UserName { get; set; }
    bool? Approved { get; set; }
    string ApproverName { get; set; }
    string CostCenter { get; set; }

    // значение DateTime должно допускать значение NULL для запуска в
    //эмуляторе хранения.
    // То же применимо ко всем типам, кроме типов, не допускающих значений Null, как
    // bool и Guid.
```

```
    DateTime? Date { get; set; }
    string ReimbursementMethod { get; set; }
    string Title { get; set; }
}

public interface IExpenseItemRow : IRow
{
    // Guid должен допускать значение NULL, чтобы запускать в эмуляторе.
    // То же применимо ко всем типам, кроме типов, не допускающих значений Null, как
    // bool и DateTime.
    Guid? ItemId { get; set; }
    string Description { get; set; }
    double? Amount { get; set; }
    string ReceiptUrl { get; set; }
    string ReceiptThumbnailUrl { get; set; }
}

public interface IRow
{
    string PartitionKey { get; set; }
    string RowKey { get; set; }
    DateTime Timestamp { get; set; }
    string Kind { get; set; }
}
```

Adatum использует классы ExpenseAndExpenseItemRow и Row для реализации интерфейсов IRow, IExpenseRow и IExpenseItemRow и для расширения класса TableServiceEntity из пространства имен StorageClient. В следующем примере кода показаны классы Row и ExpenseAndExpenseItemRow. Класс Row определяет свойство Kind, используемое для различия двух типов сущностей, хранящихся в таблице (см. перечисление TableRows в папке DataAccessLayer).

Копировать

```
public abstract class Row : TableServiceEntity, IRow
{
    protected Row()
    {
    }

    protected Row(string kind) : this(null, null, kind)
    {
    }

    protected Row(
        string partitionKey, string rowKey, string kind)
        : base(partitionKey, rowKey)
    {
        this.Kind = kind;
    }

    public string Kind { get; set; }
}

public class ExpenseAndExpenseItemRow
    : Row, IExpenseRow, IExpenseItemRow
{
    public ExpenseAndExpenseItemRow()
    {
    }

    public ExpenseAndExpenseItemRow(TableRows rowKind)
```

```
    {
        this.Kind = rowKind.ToString();
    }

    // Свойства из ExpenseRow
    public string Id { get; set; }
    public string UserName { get; set; }
    public bool? Approved { get; set; }
    public string ApproverName { get; set; }
    public string CostCenter { get; set; }
    public DateTime? Date { get; set; }
    public string ReimbursementMethod { get; set; }
    public string Title { get; set; }

    // Свойства из ExpenseItemRow
    public Guid? ItemId { get; set; }
    public string Description { get; set; }
    public double? Amount { get; set; }
    public string ReceiptUrl { get; set; }
    public string ReceiptThumbnailUrl { get; set; }
}
```

В следующем примере кода показано, как класс ExpenseDataContext ставит в соответствие класс ExpenseAndExpenseItemRow таблице MultiEntitySchemaExpenses.

Копировать

```
public class ExpenseDataContext : TableServiceContext
{
    public const string ExpenseTable =
        "MultiEntitySchemaExpenses";
    …
```

```
    public IQueryable<ExpenseAndExpenseItemRow>
        ExpensesAndExpenseItems
    {
        get
        {
            return this.CreateQuery<ExpenseAndExpenseItemRow>(
                ExpenseTable);
        }
    }
    …
}
```

Извлечение записей из таблицы с несколькими схемами

Методы запросов в классе ExpenseRepository используют класс сущностей ExpenseAndExpenseItemRow при извлечении из таблицы расходов либо сущности заголовка, либо сущности сведений. Следующий пример кода из метода GetExpensesByUser класса ExpenseRespository показывает, как извлечь строку заголовка (определяемого интерфейсом IExpenseRow).

Копировать

```
var context = new ExpenseDataContext(this.account)
    { MergeOption = MergeOption.NoTracking };


var query = (from expense in context.ExpensesAndExpenseItems
                где
                expense.UserName.CompareTo(userName) == 0 &&
                expense.PartitionKey.CompareTo(
                EncodePartitionAndRowKey(userName)) == 0
                select expense).Take(10).AsTableServiceQuery();


try
{
```

```
    return query.Execute().Select(
        e => (e as IExpenseRow).ToModel()).ToList();
}
```

Комментарий Маркуса:

В этом примере не требуется использовать свойство Kind, поскольку только сущности заголовка имеют свойство UserName.

В следующем примере кода из метода GetExpensesById класса ExpenseRepository используется свойство Kind для выбора только сущностей сведений.

Копировать

```
var expenseAndItemRows = query.Execute().ToList();


…


expenseAndItemRows.
    Where(e => e.Kind == TableRows.ExpenseItem.ToString()).
    Select(e => (e as IExpenseItemRow).ToModel()).
    ToList().ForEach(e => expense.Details.Add(e));
```

Добавление новых записей в таблицу с несколькими сущностями

Приложение aExpense использует класс сущностей ExpenseAndExpenseItemRow при добавлении новых сущностей в таблицу расходов. Следующий пример кода из класса ExpenseRepository показывает, как приложение сохраняет расходы в хранилище таблиц.

Копировать

```
public void SaveExpense(Expense expense)
{
    var context = new ExpenseDataContext(this.account);
    IExpenseRow expenseRow = expense.ToTableEntity();
    expenseRow.PartitionKey = ExpenseRepository
        .EncodePartitionAndRowKey(expenseRow.UserName);
    expenseRow.RowKey = expense.Id.ToString();
    context.AddObject(ExpenseDataContext.ExpenseTable,
```

```
        expenseRow);

    foreach (var expenseItem in expense.Details)
    {
        // Создать строку с элементом расходов.
        var expenseItemRow = expenseItem.ToTableEntity();
        expenseItemRow.PartitionKey = expenseRow.PartitionKey;
        expenseItemRow.RowKey = string.Format(
            CultureInfo.InvariantCulture, "{0}_{1}", expense.Id,
            expenseItemRow.ItemId);
        context.AddObject(ExpenseDataContext.ExpenseTable,
            expenseItemRow);

        …
    }

    // Сохранить заголовок расходов и сведения о них в одной
    // пакетной транзакции.
    context.SaveChanges(SaveChangesOptions.Batch);

    …
}
```

Комментарий Маркуса:

Обратите внимание на то, что перегруженные версии метода расширения ToTableEntity возвращают либо экземпляр IExpenseRow, либо экземпляр IExpenseItemRow.

Реализация разбиения на страницы в хранилище таблиц Windows Azure

В Adatum на текущем этапе проекта не реализованы функции разбиения на страницы (paging), но в этом разделе дан обзор планируемого подхода. Класс ResultSegment в библиотеке Windows Azure

StorageClient предоставляет свойство ContinuationToken, которое можно использовать для доступа к следующему набору результатов запроса, если этот запрос возвратил неполный набор результатов. Например, если в запросе использован оператор Take для возврата небольшой порции результатов для их отображения на странице. Это свойство ContinuationToken создает основу для любой реализации разбиения на страницы.

Комментарий Маркуса:

Разбиение результатов на страницы поможет улучшить производительность приложения, возвращая именно те данные, которые пользователю надо увидеть.

Класс ResultSegment возвращает объект ContinuationToken для доступа только к следующей странице результатов, но не к предыдущей, поэтому, если приложение должно иметь возможность вернуться к предыдущей странице, следует хранить объекты ContinuationToken, указывающие на предыдущие страницы. Удобна структура данных в виде стека. На рисунке 7 показано состояние стека после того, как пользователь просмотрел первую страницу, а затем пролистал вперед до третьей страницы.

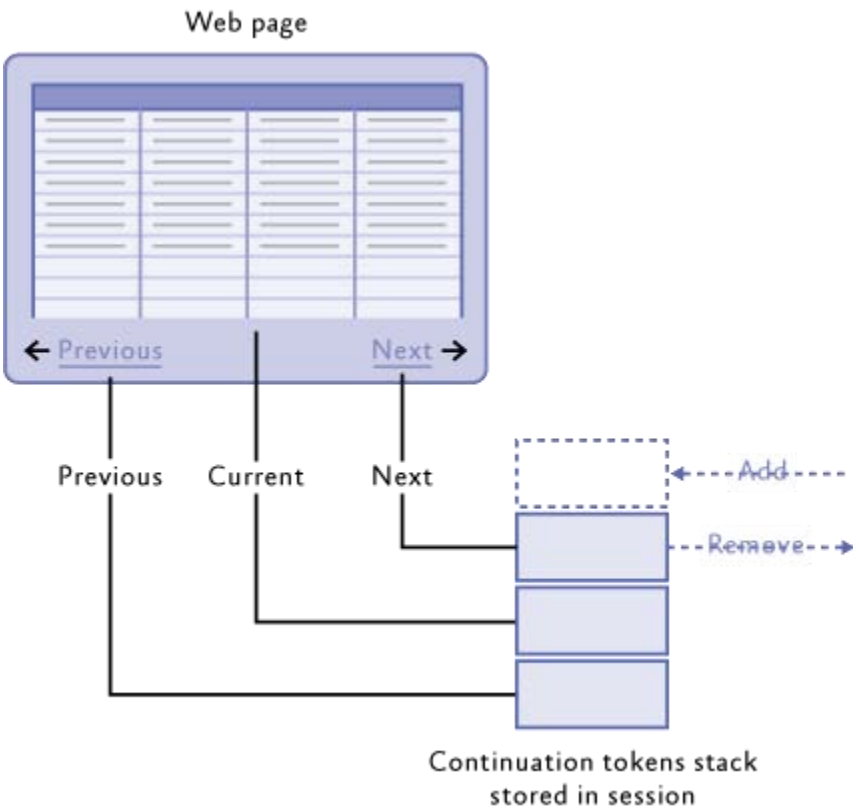


Рис. 7

Отображение страницы 3 данных из таблицы

Если пользователь щелкнет ссылку Далее для просмотра страницы 4, эта страница станет верхней в стеке для получения токена продолжения для страницы 4. После того как страница выполнит

запрос с токеном продолжения из стека, она принудительно отправит в стек новый токен продолжения для страницы 5.

Если пользователь щелкнет гиперссылку Назад для просмотра страницы 2, страница извлечет из стека две записи и станет верхней для получения токена продолжения для страницы 2. После того как страница выполнит запрос с токеном продолжения из стека, она принудительно отправит в стек новый токен продолжения для страницы 3.

В следующих примерах кода показано, как в Adatum могло быть реализовано такое поведение на асинхронной странице ASP.NET.

Примечание.
Использование асинхронной страницы освобождает поток страниц из пула потоков во время выполнения потенциально длительной операции ввода-вывода. Это улучшает пропускную способность веб-сервера и повышает масштабируемость приложения.

В следующих двух примерах кода показано, как создать асинхронную страницу ASP.NET. Сначала добавьте атрибут Async="true" в директиву страницы в ASPX-файле.

Копировать

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs"
Inherits="ContinuationSpike._Default" Async="true"%>
```

Затем зарегистрируйте методы начала и конца для асинхронной операции в событии загрузки для этой страницы.

Копировать

```
protected void Page_Load(object sender, EventArgs e)
{
   AddOnPreRenderCompleteAsync(
      new BeginEventHandler(BeginAsyncOperation),
      new EndEventHandler(EndAsyncOperation)
   );
}
```

В следующем примере кода показано определение класса ContinuationStack, используемого приложением для хранения токенов продолжения в состоянии сеанса.

Комментарий Маркуса:

Нам необходимо хранить токены продолжения как часть состояния сеанса.

Копировать

```
public class ContinuationStack
{
    private readonly Stack stack;

    public ContinuationStack()
    {
        this.stack = new Stack();
    }

    public bool CanMoveBack()
    {
        if (this.stack.Count >= 2)
            return true;

        return false;
    }

    public bool CanMoveForward()
    {
        return this.GetForwardToken() != null;
    }

    public ResultContinuation GetBackToken()
    {
        if (this.stack.Count == 0)
            return null;
        // Надо извлечь из стека дважды и возвратить то, что останется.
```

```
        this.stack.Pop();
        this.stack.Pop();
        if (this.stack.Count == 0)
            return null;
        return this.stack.Peek() as ResultContinuation;
    }

    public ResultContinuation GetForwardToken()
    {
        if (this.stack.Count == 0)
            return null;

        return this.stack.Peek() as ResultContinuation;
    }

    public void AddToken(ResultContinuation result)
    {
        this.stack.Push(result);
    }
}
```

В следующем примере кода показан метод BeginAsyncOperation, который начинает выполнение запроса для следующей страницы данных. Значение ct в строке запроса указывает направление движения.

Копировать

```
private IAsyncResult BeginAsyncOperation(object sender, EventArgs e, AsyncCallback cb, object extradata)
{
    var query =
      new MessageContext(CloudConfiguration.GetStorageAccount())
      .Messages.Take(3).AsTableServiceQuery();
```

```
    if (Request["ct"] == "forward")
    {
        var segment = this.ContinuationStack.GetForwardToken();
        return query.BeginExecuteSegmented(segment, cb, query);
    }

    if (Request["ct"] == "back")
    {
        var segment = this.ContinuationStack.GetBackToken();
        return query.BeginExecuteSegmented(segment, cb, query);
    }
    return query.BeginExecuteSegmented(cb, query);
}
```

Метод EndAsyncOperation помещает результаты запроса в список сообщений и принудительно отправляет в стек новый токен продолжения.

Копировать

```
private List<MessageEntity> messages;

private void EndAsyncOperation(IAsyncResult result)
{
    var cloudTableQuery =
        result.AsyncState as CloudTableQuery<MessageEntity>;
    ResultSegment<MessageEntity> resultSegment =
        cloudTableQuery.EndExecuteSegmented(result);
    this.ContinuationStack.AddToken(
        resultSegment.ContinuationToken);
    this.messages = resultSegment.Results.ToList();
}
```

Дополнительные сведения

Сайт AzureScope содержит полезные рекомендации по оптимизации производительности приложений Windows Azure: http://azurescope.cloudapp.net/.

Подробные рекомендации по использованию хранилища таблиц Windows Azure содержатся в следующем документе: http://go.microsoft.com/fwlink/?LinkId=153401.

Подробные рекомендации по использованию хранилища больших двоичных объектов Windows Azure содержатся в следующем документе: http://go.microsoft.com/fwlink/?LinkId=153400.

Глоссарий

SQL Azure. Система управления реляционными базами данных (СУРБД) в облаке. SQL Azure не зависит от хранилища, являющегося частью Windows Azure. Основывается на SQL Server® и может хранить структурированные, полуструктурированные и неструктурированные данные.

Windows Azure. Платформа корпорации Microsoft для облачных вычислений. В Интернете предоставляется как услуга. Включает вычислительную среду, хранилище Windows Azure, службы управления и фабрику приложений.

Аренда. Монопольная блокировка записи в большой двоичный объект, которая длится до истечения аренды.

Библиотека Enterprise Library. Собрание повторно используемых программных компонентов (блоков приложений), спроектированных в помощь разработчикам программного обеспечения для общих потребностей предприятия (таких как ведение журналов, проверка, доступ к данным, обработка исключений и многие другие).

Веб-роль. Интерактивное приложение, работающее в облаке. Веб-роль можно реализовать с помощью любой технологии, которая работает со службами IIS 7.

Вертикальная масштабируемость. Возможность наращивать ресурсы компьютера, такие как память или процессор.

Горизонтальная масштабируемость. Возможность добавления новых серверов, являющихся копиями существующих серверов.

Группа сходства. Именованное группирование, являющееся отдельным центром данных. Может включать все компоненты, связанные с приложением, такие как хранилища, базы данных SQL Azure и роли.

Идемпотентная операция. Операция, которую можно выполнять несколько раз, не изменяя результаты. Примером служит задание значения переменной.

Код вблизи. Ситуация, когда и приложение, и связанные с ним базы данных находятся в одном облаке.

Код вдали. Ситуация, когда приложение находится в организации, а связанные с ним базы данных — в облаке.

Моментальный снимок. Доступная только для чтения копия большого двоичного объекта.

Облако. Набор взаимосвязанных серверов, расположенных в одном или более центров данных.

Оптимистичный параллелизм. Метод управления параллелизмом, в котором предполагается, что несколько изменений данных могут завершаться, не влияя друг на друга. Поэтому не требуется блокировать ресурсы данных. Оптимистичный параллелизм предполагает, что нарушения параллельного выполнения возникают нечасто, и просто не разрешает проводить обновления или удаления, вызывающие такие нарушения.

Пакет службы. Упаковывает двоичные файлы роли и файл определения службы для публикации в среде Windows Azure.

Передача состояния представления (REST). Архитектурный стиль для получения информации с веб-сайтов. Ресурс является источником специфической информации. Каждый ресурс идентифицируется глобальным идентификатором, таким как универсальный код ресурса (URI) в HTTP. Представление является реальным документом, который передает информацию.

Рабочая роль. Выполняет пакетные процессы и фоновые задачи. Рабочие роли могут формировать исходящие вызовы и открывать конечные точки для входящих вызовов. Обычно рабочие роли используют очереди для связи с веб-ролями.

Сеть доставки содержимого (CDN). Система, состоящая из нескольких серверов, содержащих копии данных. Эти серверы размещены в разных географических областях, так что пользователи могут получать доступ к ближайшей к ним копии.

Служба внешнего размещения. Пространство, где развернуты приложения.

Сообщение о сбое. Сообщение, содержащее данные неправильного формата, которые вызвали формирование обработчиком очереди сообщения об исключении. В результате это сообщение не обрабатывается и остается в очереди, а следующая попытка обработать его снова завершается ошибкой.

Среда Windows Azure. Предоставляет физические машины для вычислений и хранения.

Среда разработки. Имитирует среду Windows® Azure™, позволяя запускать и тестировать службу локально, до ее развертывания. Известна как эмулятор вычислений.

Утверждение. Утверждение о субъекте, например его имя, идентификатор, ключ, группа, разрешение или возможность, которое субъект делает о себе или о другом субъекте. Утверждениям присваивается одно или несколько значений, затем они упаковываются в токены безопасности, которые распространяются поставщиком.

Фабрика приложений Windows Azure. Предоставляет облачные службы для связи приложений, работающих в облаке или локально, а также службы управления доступом. Не путать со средой Windows Azure.

Файл конфигурации службы. Устанавливает значения для службы, которые можно настроить во время выполнения службы в среде. Значения, которые можно указать в файле конфигурации службы, включают количество экземпляров, которые надо развернуть для каждой роли, значения параметров конфигурации, установленные в файле определения службы, и отпечатки любых SSL-сертификатов, связанных с этой службой.

Файл определения службы. Определяет роли, составляющие службу, необязательные локальные ресурсы хранения, параметры конфигурации и сертификаты конечных точек SSL.

Хранилище Windows Azure. Состоит из больших двоичных объектов, таблиц и очередей. Доступ возможен с помощью запросов HTTP/HTTPS. Отличается от SQL Azure.

Эмулятор вычислений. Эмулятор вычислений в Windows Azure позволяет запускать, тестировать, отлаживать и настраивать приложение до его развертывания в виде службы внешнего размещения в Windows Azure. См. также эмулятор хранения.

Эмулятор хранилища. Эмулятор хранения Windows Azure предоставляет локальные экземпляры служб Blob, Queue и Table, доступных в Windows Azure. При создании приложения, использующего службы хранения, его можно локально протестировать с помощью эмулятора хранения.