# Построение гибридных приложений в облаке на платформе Windows Azure

patterns & practices

**Обзор**: Это третья книга в серии patterns & practices, посвящённой платформе Windows Azure. В книге показано, как можно использовать мощные инфраструктурные сервисы, которые являются частью Windows Azure, с целью упрощения разработки, интеграции составных частей гибридных приложений, включающих облачные и локальные компоненты, а также системы сторонних разработчиков: схема аутентификации и авторизации пользователей, комплексного мониторинга приложений, шаблонов обмена сообщениями и т.п. Также описывается, как обеспечить максимальную безопасность, производительность, масштабируемость и доступность.

Это руководство ориентировано на архитекторов, разработчиков и специалистов по информационным технологиям (ИТ), которые проектируют, создают или используют приложения и сервисы, работающие в облаке или взаимодействующие с ним.

**Категория:** Справочник\руководство
**Аудитория**: Windows Azure, Windows Azure SQL Database, SQL Server, Windows Identity Foundation, Windows Azure Service Bus, Enterprise Library, Windows Azure Diagnostics, Windows Azure Management Cmdlets
**Источник**: patterns & practices
**Дата публикации электронной книги**: Сентябрь 2012

# Содержание

# Авторские права

# Предисловие Клеменса Вастерса (Clemens Vasters)

Первые возможности облачных технологий по модели «платформа как услуга» были реализованы Microsoft в виде версии для технической апробации сервисов Relay Service и Security Token Service подразделения Microsoft Live Labs, анонсированной 31 мая 2006 г. (см. http://blogs.msdn.com/b/labsrelay/archive/2006/05/31/612288.aspx). Такой подход способствовал повышению вычислительной мощи и расширению возможностей в сфере хранения данных и сетевого взаимодействия на основе платформы Windows Azure. В последующие годы эти два сервиса несколько раз получали новые имена и были значительно усовершенствованы, как с точки зрения возможностей, так и с точки зрения надежности, но миссия и цели, определенные почти шесть лет назад для шины интеграции Windows Azure и службы Windows Azure Access Control остались неизменными: *создание гибридных приложений*.

Мы убеждены в том, что наша облачная платформа, а также аналогичные предложения наших конкурентов, предоставляют компаниям очень привлекательную альтернативу для создания и эксплуатации собственных центров обработки данных. Мы считаем, что наши клиенты смогут сократить общие затраты и свести к минимуму требуемые капиталовложения. Мы также уверены в том, что Microsoft сможет защищать, эксплуатировать и управлять собственными серверными операционными системами, средами выполнения и платформами хранения данных лучше, чем любой другой поставщик решений. И мы гарантируем, что используемая нами платформа полностью готова для выполнения ключевых рабочих нагрузок практически любой компании. Но этого не достаточно.

Облачная платформа Microsoft, в особенности шина интеграции и службы управления доступом, изначально разрабатывались с учетом того факта, что «переход к облаку» представляет собой постепенный процесс, и что многие рабочие задания никогда не переместятся в облако. Некоторые службы жестко связаны с каким-либо местом или конкретным человеком. Если вы хотите напечатать документ, конечный результат будет физическим, это лист бумаги в руках одного из сотрудников. Если вы хотите подать сигнал тревоги, чтобы уведомить конкретное лицо, лучше использовать устройство, сигнал которого этот человек может услышать. Другие сервисы не будут «перемещаться в облако», поскольку они субъективно и объективно «идеально работают» в центре обработки данных на существующем оборудовании, или потому, что утвержденные нормы или политики затрудняют или даже делают невозможным выполнение этой задачи.

Тем не менее мы считали и считаем, что предложения, связанные с облачными вычислениями, интересны для компаний, которые основную ставку делают на центры обработки данных. В качестве примера возьмем страховой бизнес. Страховые компании одними из первых стали применять информационные технологии. Не будет большой ошибкой назвать страховые компании (и банки) «центрами обработки данных с прилавками для обслуживания потребителей». ИТ — это сердце их бизнес-операций, такая модель сложилась несколько десятилетий назад, и бизнес-операции полностью прекращаются, когда это сердце перестает биться. Многие из этих технологий обеспечивают выполнение основных рабочих заданий, которые очень хорошо проработаны. Эти задания выполняются на таких же сформировавшихся системах, заслуживших доверие.

Выход на этот рынок с облачным предложением оказывает отрезвляющее воздействие на молодых, инициативных и энергичных продавцов. Или нет? На самом деле есть множество вариантов эффективного применения неоспоримой гибкости облачных сред, даже если все основные рабочие задания должны оставаться неизменными и не поддерживают гибкость. Страховые компании тратят довольно много энергии (и средств) на привлечение клиентов, некоторые из них постоянно окружают нас своей рекламой. С появлением облачных вычислений стало трудно обосновать необходимость развертывания выделенного локального оборудования для запуска веб-сайта маркетинговой кампании, и если бы не острая проблема, связанная с тем, что сайт также должен предоставлять информацию о тарифах, которые рассчитываются серверной системой, то, в идеале, можно было бы заключить сделку прямо сейчас.

Но такие аргументы стали бы не актуальны, если бы маркетинг основывался на «гибридных» решениях, которые включают облачные и локальные ресурсы. Именно поэтому мы создали то, что задумали и начали реализовывать шесть лет назад.

В рамках гибридного приложения маркетинговый веб-сайт расширяется и работает в облачной среде, а ценные инструменты управления взаимодействием с клиентами могут безопасно подключаться и отправлять сообщения основным серверным системам, а также выполнять транзакции. Именно с учетом такого сценария мы разрабатываем шину интеграции Windows Azure и «коннектор шины интеграции» для BizTalk Server. А для сценариев, связанных с существующими рабочими нагрузками, мы предлагаем технологию создания виртуальной сети Windows Azure Connect VPN.

Гибридными также являются приложения, которые распределены между несколькими узлами (по тем же причинам, которые упомянуты выше), а также реплицируются и обновляются через облако. Это домен SQL Azure Data Sync. Рабочие задания распределяются между локальными узлами и облачными решениями, выходя за пределы сферы ответственности стандартных систем безопасности. Дополнительные сложности связаны с управлением и федерацией удостоверений для столь разнородных сред. Служба Windows Azure Access Control позволяет справиться с этой сложной задачей, предоставляя доступ к распределенным составляющим системы после проверки согласованных удостоверений.

Данное руководство содержит подробные рекомендации по поводу разработки архитектуры и создания гибридных решений на основе и при помощи технологической платформы Windows Azure. Это результат напряженного труда группы специалистов, которые проанализировали и обобщили практический опыт разработчиков Windows Azure и, что еще важнее, проектов реальных заказчиков. Мы надеемся, что вы найдете это руководство полезным и сможете эффективно применять полученные знания в процессе создания собственных гибридных решений.

Благодарим за использование Windows Azure!

Клеменс Вастерс

Главный технический руководитель и архитектор

Шина интеграции Windows Azure

# Введение

Современные вычислительные платформы и технологии, такие как Microsoft .NET Framework, ASP.NET, Windows Communication Foundation и Windows Identity Framework существенно упрощают создание корпоративных приложений. Кроме того, возможность создания приложений, которые будут развернуты в облаке с использованием технологической платформы Windows Azure, помогает снизить первоначальные вложения в инфраструктуру и упростить требования к текущему управлению и обслуживанию.

Большинство современных приложений имеют сложную структуру, они могут состоять из множества отдельных функций, реализованных в виде сервисов, компонентов, подключаемых модулей сторонних разработчиков, а также других систем или ресурсов. Интеграция этих компонентов, когда все они размещаются локально в центре обработки данных,— достаточно нетривиальная задача, но она еще больше усложняется при переносе приложений в облачные среды.

Например, стандартное приложение может использовать веб-роль и рабочую роль в Windows Azure, сохранять свои данные в технологической базе данных SQL Azure, а также подключаться к сторонним сервисам, которые выполняют такие задачи, как аутентификация пользователей или доставка товаров клиентам. Тем не менее не являются редкостью приложения, которые также используют сервисы, предоставляемые организациями-партнерами, или сервисы и компоненты, которые размещены внутри корпоративной сети и в силу различных причин не могут быть перенесены в облако.

Подобные приложения часто называют *гибридными*. Проблемы, с которыми вы сталкиваетесь при их создании или при переносе части существующих локальных приложений в облако, характеризуются следующими вопросами: «Как я могу интегрировать разные части вне зависимости от границ сетей и доменов, чтобы все части могли взаимодействовать в целях реализации всего приложения?» и «Как я могу добиться максимальной производительности и доступности, когда некоторые части приложения размещаются в облаке?».

В этом руководстве основное внимание уделяется общим проблемам, с которыми вы столкнетесь в процессе создания приложений, состоящих из облачных и локальных компонентов, или когда вы решите перенести все или некоторые элементы существующих локальных приложений в облако. Мы сосредоточимся на особенностях применения Windows Azure в качестве среды размещения и покажем, как можно воспользоваться преимуществами многочисленных возможностей этой платформы и базы данных SQL Azure в целях упрощения и ускорения разработки приложений рассматриваемого типа.

Платформа Windows Azure предоставляет набор инфраструктурных сервисов, которые помогут вам в процессе создания гибридных приложений. Эти сервисы, например Service Bus Security, Messaging, Caching, Traffic Manager и Azure Connect, стали основными темами этого руководства. В руководстве представлены сценарии, в рамках которых полезны эти сервисы. Вы узнаете о том, как можно применять их в своих приложениях.

В этом руководстве описывается деятельность вымышленной корпорации под названием Trey Research, которая модернизирует существующие локальные приложения, чтобы воспользоваться преимуществами Windows Azure. Здесь не рассматриваются отдельные задачи миграции, основное внимание уделяется тому, как компания Trey Research использует сервисы, предоставляемые платформой Windows Azure и базой данных SQL Azure, для организации взаимодействия, контроля процессов и производительности, а также для управления, синхронизации данных и обеспечения безопасности.

## Для кого предназначена эта книга

Это третий том из серии, посвященной платформе Windows Azure. Том 1 (*Перемещение приложений в облако на базе Windows Azure*) представляет собой введение в Windows Azure, в нем обсуждается модель затрат и принципы управления жизненным циклом облачных приложений, а также описывается миграция существующих приложений ASP.NET в облако. Том 2 (*Разработка приложений для облака на базе Windows Azure*) содержит рекомендации по проектированию и подробности внедрения приложений, которые изначально ориентированы на работу в облаке. Здесь также подробнее рассматриваются многие темы, представленные в томе 1, описаны более сложные методы, которые можно применять в отношении приложений для Windows Azure.

В этом (третьем) томе серии показано, как можно использовать мощные инфраструктурные сервисы, которые являются частью Windows Azure, с целью упрощения разработки, интеграции составных частей гибридных приложений, включающих облачные и локальные компоненты, а также системы сторонних разработчиков. Также описывается, как обеспечить максимальную безопасность, производительность, масштабируемость и доступность.

Это руководство ориентировано на архитекторов, разработчиков и специалистов по информационным технологиям (ИТ), которые проектируют, создают или используют приложения и сервисы, работающие в облаке или взаимодействующие с ним. Хотя приложения для Windows Azure не обязательно должны разрабатываться для операционной системы Microsoft Windows, но эта книга предназначена для тех, кто работает с системами на базе Windows. Читатели также должны быть знакомы с платформой Microsoft .NET Framework, системой разработки Microsoft Visual Studio, платформой ASP.NET MVC и языком программирования Microsoft Visual C#.

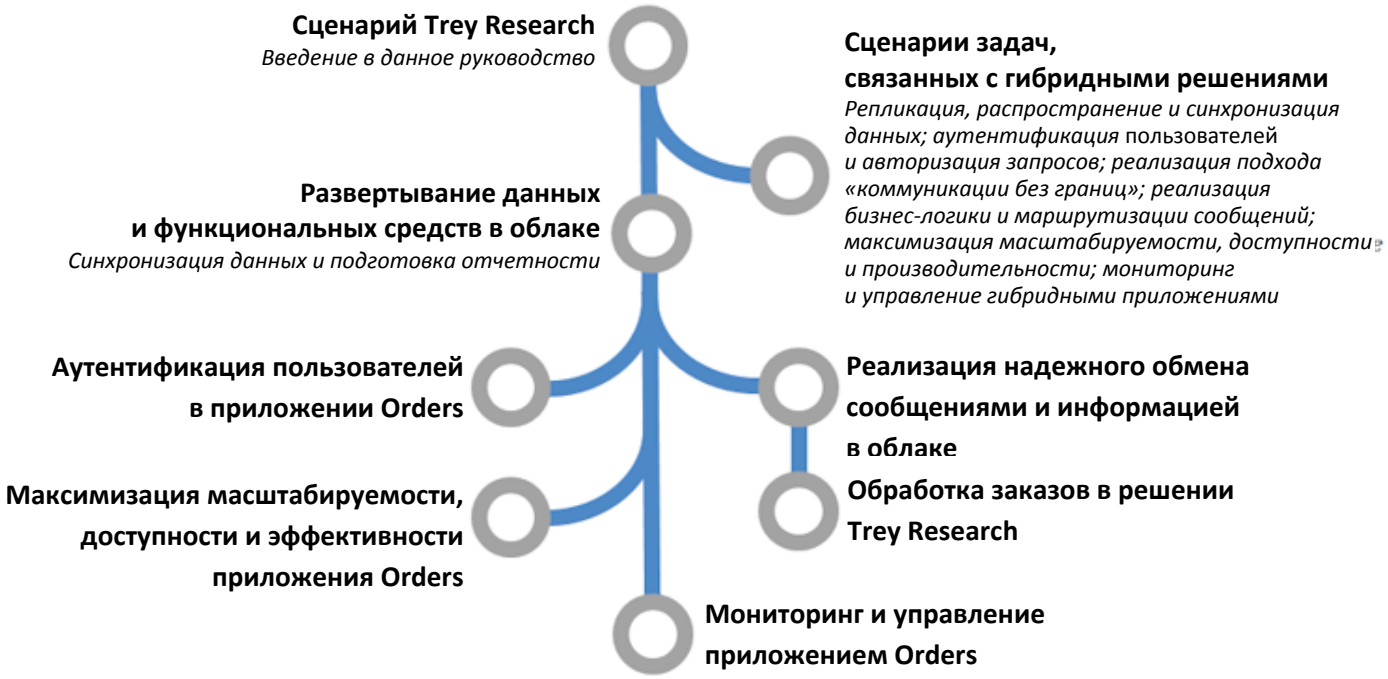
## Почему эта книга вышла именно в это время

Дизайнеры и разработчики программного обеспечения, менеджеры проектов и администраторы все чаще признают преимущества размещения ИТ-сервисов в облаке, позволяющие сократить расходы на инфраструктуру и текущие затраты на эксплуатацию центров обработки данных, максимально повысить доступность, упростить управление и воспользоваться предсказуемой моделью ценообразования. Однако приложения часто содержат некоторые компоненты или функции, которые не могут быть размещены в облаке, например сервисы сторонних поставщиков или конфиденциальные данные, которые должны храниться локально под контролем специалистов.

Для разработки подобных приложений требуются дополнительные усилия в сфере проектирования и разработки, направленные на обеспечение взаимодействия и интеграции компонентов и сервисов. Чтобы избежать сложностей в процессе переноса приложений в облако, Windows Azure предоставляет ряд платформенных сервисов, которые способствуют интеграции облачных и локальных компонентов приложений и сервисов. В этом руководстве разъясняются принципы применения этих сервисов к типовым сценариям и приложениям, которые вы в настоящее время создаете или перемещаете.

## Структура книги

Это «карта» данного руководства.




Глава 1 «Сценарий Trey Research» знакомит читателей с компанией Trey Research и ее планами по преобразованию приложения Orders для обработки заказов в гибридное решение. Здесь также приводится обзор архитектуры и принципов функционирования оригинального локального приложения и итоговой гибридной реализации, что предоставит вам контекст для остальной части руководства.

Глава 2 «Развертывание данных и приложения Orders в облаке». В ней обсуждаются методы и технологии, которые Trey Research предполагает использовать для развертывания своего приложения и данных в облаке. Приводятся соображения специалистов Trey Research по поводу того, какие данные должны оставаться локальными, а также описание архитектуры развертывания, которая, по их мнению, наилучшем образом удовлетворяет требованиям. В этой главе также описаны технологии синхронизации данных между локальными и облачными ресурсами, а также методы поддержания работоспособности систем бизнес-аналитики для формирования отчетности.

Глава 3 «Аутентификация пользователей в приложении Orders». Здесь приведено описание технологий и архитектур, которые Trey Research анализировала с точки зрения возможности преобразования локального инструмента аутентификации на базе ASP.NET Forms в инструмент аутентификации на основе утверждений в рамках гибридного приложения.

Глава 4 «Реализация надежного обмена сообщениями и информацией в облаке» содержит описание технологий, которые специалисты Trey Research исследовали с целью организации отправки сообщений через локальные и облачные системы, а также решений, которые компания в итоге выбрала. Также рассматривается архитектура и реализация надежной системы отправки сообщений партнерам и локальным службам.

Глава 5 «Обработка заказов в решении Trey Research» содержит описание бизнес-логики, которая требуется Trey Research для безопасной и надежной обработки заказов клиентов на веб-сайте. Логика включает в себя алгоритмы направления сообщений партнеру или службе, получения подтверждения и повтора операций, которые могут оказаться неудачными из-за временных проблем с сетью.

Глава 6 «Максимизация масштабируемости, доступности и эффективности приложения Orders». Здесь описывается процесс анализа специалистами Trey Research методов максимизации производительности приложения Orders для обработки заказов путем автоматического масштабирования экземпляров веб-ролей и рабочих ролей в приложении, а также способов развертывания приложения в нескольких центрах обработки данных и ускорения доступа к данным посредством кэширования.

Глава 7 «Мониторинг и управление приложением Orders» содержит описание методов, которые Trey Research проанализировала и выбрала для мониторинга и управления приложением Orders. Эти методы включают в себя сбор диагностической информации, установку и настройку служб Windows Azure, а также удаленную настройку и управление приложением.

В основных главах этого руководства описываются процессы проектирования и решения, которые выбрала компания Trey Research, а в приложениях под названием «*Сценарии задач, связанных с гибридными решениями*» рассматриваются более обобщенные сценарии проектирования и создания гибридных приложений. Каждое приложение посвящено одной конкретной области задач и требований, связанных с гибридными приложениями и описанных в главе 1 «Сценарий Trey Research», кроме того, описываются подходы и решения, которые проектировщики Trey Research не рассматривали в процессе работы над приложением Orders. В дополнение к сценариям, в приложениях приводятся более конкретные рекомендации по применению имеющихся технологий для решения каждой проблемы. Приложения к настоящему руководству:

- Приложение А. Репликация, распространение и синхронизация данных.
- Приложение Б. Аутентификация пользователей и авторизация запросов.
- Приложение В. Реализация подхода «коммуникации без границ».
- Приложение Г. Реализация бизнес-логики и маршрутизации сообщений с целью организации совместной работы.
- Приложение Д. Максимизация масштабируемости, доступности и производительности.
- Приложение Е. Мониторинг и управление гибридными приложениями.

Информация, содержащаяся в настоящем руководстве, относится к Windows Azure и SQL Azure, а также сервисам, предоставляемым ими на момент написания. Тем не менее платформа Windows Azure постоянно развивается, и новые возможности и функции появляются достаточно часто. Более подробная информация представлена на странице «What's New in Windows Azure» («Что нового в Windows Azure?»): http://msdn.microsoft.com/en-us/library/windowsazure/gg441573 и на домашней странице Windows Azure: http://www.microsoft.com/windowsazure/.

## Что вам потребуется для использования кода

Системные требования для запуска сценариев:

- Microsoft Windows 7 с пакетом обновления Service Pack 1 или более поздним (32- или 64-битная версия), либо Windows Server 2008 R2 с пакетом обновления Service Pack 1 или более поздним.
- Microsoft Internet Information Server (IIS) 7.0.
- Microsoft .NET Framework, версия 4.0.
- Microsoft ASP.NET MVC Framework, версия 3.
- Microsoft Visual Studio 2010 (выпуск Ultimate, Premium или Professional) с установленным пакетом обновления Service Pack 1.
- Windows Azure SDK для .NET (включает инструменты Visual Studio Tools для Windows Azure).
- Microsoft SQL Server или SQL Server Express 2008.
- Платформа Windows Identity Foundation.
- Microsoft Enterprise Library 5.0 (необходимые сборки включены в доступный для загрузки исходный код).
- Командлеты Windows Azure (установите командлеты Windows Azure как встраиваемый модуль Windows PowerShell, это необходимо для сценариев, использующих интерфейсы Azure Management API).
- Пример базы данных (сценарии размещены в папке «База данных» исходного кода).

Вы можете загрузить примеры кода с сайта http://wag.codeplex.com/releases/. Пример кода включает утилиту проверки зависимостей, которая позволяет проверить предварительные условия, чтобы установить недостающие компоненты. Утилита проверки зависимостей также устанавливает примеры баз данных.

## Кто есть кто

В этой книге используется пример приложения для демонстрации процесса интеграции приложений с облачной средой. Группа экспертов комментирует процесс разработки. В состав группы входит специалист по облачным решениям, архитектор программного обеспечения, разработчик программного обеспечения и ИТ-специалист. Предоставление приложения рассматривается с точки зрения каждого из этих специалистов. В следующей таблице представлены входящие в состав группы эксперты.

| | |
|---|---|
|  | **Бхарат** — специалист по облачным решениям. Он отвечает за то, чтобы облачные решения успешно работали на компанию и обеспечивали ощутимые преимущества. Он очень осторожный человек (по уважительным причинам).<br>*«Развернуть в облаке приложение для одного клиента достаточно просто. Гораздо сложнее реализовать преимущества размещаемых в облаке многопользовательских решений».* |
|  | **Джана** — архитектор программного обеспечения. Она планирует общую структуру приложения. Она занимает позиции как стратега, так и практика. Другими словами, она анализирует технические подходы, которые необходимы сегодня, и направления, которых компания должна придерживаться с перспективой на будущее.<br>*«Балансировка потребностей компании, пользователей, ИТ-подразделения, разработчиков и технических платформ, на которые мы полагаемся,– достаточно сложная задача».* |
|  | **Маркус** — старший разработчик программного обеспечения. Он придерживается аналитического подхода, пристальное внимание уделяет деталям и методологии. Он сосредоточен на задаче создания эффективного облачного приложения. Он осознает, что несет единоличную ответственность за код.<br>*«Многое из того, что мы знаем о разработке программного обеспечения, применимо к облаку. Но в проекте всегда присутствуют особые соображения, которые очень важны».* |
|  | **По** — ИТ-специалист, эксперт по развертыванию и запуску приложений в облаке. Он проявляет большой интерес к практическим решениям, поскольку именно его могут вызвать среди ночи, если возникает проблема.<br>*«Запуск приложений в облаке, к которому получают доступ тысячи пользователей, включает в себя несколько серьезных задач. Я должен убедиться, что наши облачные приложения работают хорошо, надежно и безопасно. Репутация Trey Research зависит от того, как пользователи воспринимают приложения, размещенные в облаке».* |

Если вас интересуют особые вопросы, обращайте внимание на примечания, подготовленные специалистом, чьи интересы совпадают с вашими.

# Благодарности

ИТ-индустрия непрерывно развивается быстрыми темпами, и с появлением облачных вычислений скорость изменений существенно возросла. Еще в январе 2010 года, когда мы начали работу над первым руководством в этой серии, для платформы Windows Azure предлагался лишь базовый набор функциональных возможностей для работы с вычислительными системами, устройствами хранения и базами данных. Два года спустя, на момент написания настоящего руководства, были реализованы многочисленные дополнительные функции, которые будут полезны в различных сценариях.

Между тем, все больше организаций осознают преимущества и используют облачные вычисления. В 2010 году большинство людей, с которыми я разговаривал, интересовались облачными решениями, но никто из них не имел практического опыта. Но ситуация изменилась. Клиенты часто поражают меня своими знаниями и опытом. Я нисколько не сомневаюсь, что вся отрасль движется в направлении облачных технологий.

Разумеется, переход к облакам будет постепенным. Большинство организаций по-прежнему эксплуатируют множество ИТ-активов, размещенных в локальных центрах обработки данных. Рано или поздно эти ресурсы перенесут в облака, но для смены парадигмы всегда нужно некоторое время. В данный момент мы находимся в середине пути при переходе от состояния, когда все ресурсы развернуты локально, к состоянию, когда все ресурсы размещены в облаке. Термин «гибрид» используется для обозначения приложения, которое размещает свою архитектуру где-то на этом отрезке. Другими словами, гибридные приложения охватывают локальные и облачные среды и формируют уникальный набор задач, которые необходимо решать. Решением этих проблем занимался я и моя команда, и мы подготовили для вас данное руководство.

Цель этого руководства — сопоставление возможностей Windows Azure с конкретными проблемами, которые они помогают решить в рамках гибридных сценариев. Windows Azure теперь предоставляет ряд расширенных функций, таких как Service Bus, Caching, Traffic Manager, Azure Connect, SQL Azure Data Sync, VM Role, ACS и т. д. В нашем руководстве все процессы рассматриваются на примере вымышленной организации, что позволяет проиллюстрировать проблемы, с которыми вы можете столкнуться при работе с гибридным приложением; мы также описываем решения, которые позволяют использовать возможности Windows Azure для интеграции локальных и облачных сред.

При работе с функциями интеграции Windows Azure нам часто приходилось уточнять и проверять наши рекомендации по их использованию. Нам очень повезло, поскольку мы в любой момент могли получить поддержку со стороны групп специалистов по продуктам и других подразделений Microsoft. Прежде всего я хочу поблагодарить следующих профильных специалистов: Клеменса Вастерса (Clemens Vasters), Марка Скерелла (Mark Scurrell), Джейсона Чена (Jason Chen), Тину Стюарт (Tina Stewart), Аруна Раджаппа (Arun Rajappa) и Кори Сандерс (Corey Sanders). Мы опирались на их знания и опыт в соответствующих областях в процессе работы над данным руководством. Многие предложения, высказанные этими рецензентами, а также их подробные отзывы, были включены в данное руководство.

Следующие специалисты также сыграли важную роль в разработке этого руководства и поделились ценными техническими знаниями и опытом: Кашиф Алам (Kashif Alam), Виджая Алапарти (Vijaya Alaparthi), Матиас Волоски (Matias Woloski), Юджинио Пейс (Eugenio Pace), Энрике Carrec

(Enrique Saggese) и Трент Свансон (Trent Swanson), [FullScale 180]. Мы опирались на их опыт в процессе проверки сценария, а также формирования архитектуры решения.

Также хочу выразить благодарность проектной команде. Технические писатели Джон Шарп (John Sharp), редактор, и Алекс Гомер (Alex Homer) внесли в это руководство значительный вклад, опираясь на свой писательский опыт и знания в области разработки программных продуктов. Скотт Денсмор (Scott Densmore), Хорхе Ровес (Jorge Rowies) [Southworks], Алехандро Езерски (Alejandro Jezierski) [Southworks], Ганц Чжан (Hanz Zhang), Равиндра Махендраварман (Ravindra Mahendravarman) [Infosys Ltd.] и Равиндран Парамасивам (Ravindran Paramasivam) [Infosys Ltd.] выступили в качестве разработчиков и тестировщиков. Глубокое понимание особенностей Windows Azure, исключительная страсть к технологиям и многочасовая кропотливая работа позволили им создать пример кода.

Я также хочу поблагодарить Роэнн Корбисиер (RoAnn Corbisier) и Ричарда Буртэ (Richard Burte) [ChannelCatalyst.com, Inc] за помощь в публикации данного руководства. Их опыт в редактировании и графическом дизайне помог сделать это руководство точным и интересным.

Концепция визуального оформления данного руководства была первоначально разработана Робертой Лейбовиц (Roberta Leibovitz) и Колином Кэмпбеллом (Colin Campbell) [Modeled Computation LLC] для «Руководства по идентификации на основе утверждений и управлению доступом». Мы получили отличные отзывы и решили использовать эту концепцию и для этой книги. Дизайн книги создал Джон Хаббард (John Hubbard), [eson]. Мультипликационные лица были нарисованы выдающимся карикатуристом из Сиэтла Эллен Форни (Ellen Forney).

Также выражаю особую благодарность участникам сообщества нашего веб-сайта CodePlex. Я высоко ценю отзывы, которые мы получаем от этой очень разнородной группы читателей.

Масаши Нарумото (Masashi Narumoto)

Старший менеджер проектов группы Patterns & Practices

Корпорация Microsoft

Редмонд, январь 2012 г.

# Глава 1. Сценарий Trey Research

В этом руководстве рассматриваются варианты применения служб, предоставляемых технологической платформой Windows Azure, а также некоторые другие полезные системы и компоненты, которые помогут вам интегрировать приложения, имеющие облачные элементы, с целью создания гибридных решений. *Гибридное приложение* использует ряд компонентов, ресурсов и служб, которые могут быть выделены за границы центра обработки данных, организации, сети или области доверия. Некоторые из этих компонентов, ресурсов и сервисов могут размещаться в облаке, но не обязательно. Тем не менее в данном руководстве мы будем уделять основное внимание приложениям, компоненты которых работают в Windows Azure.

В руководстве рассматривается сценарий для вымышленной компании Trey Research, которая хочет адаптировать существующие приложения, чтобы получить возможность воспользоваться преимуществами, предоставляемыми Windows Azure. Здесь описываются проблемы, с которыми Trey Research столкнулась, и архитектурные решения, разработанные специалистами компании.

**Мнение Джаны**

Гибридные приложения используют ресурсы и службы, которые размещены в различных физических или виртуальных средах — локально, на ресурсах организаций-партнеров или в облаке. Гибридные приложения представляют собой промежуточное положение между состоянием, когда все работает локально, и состоянием, когда все размещено в облаке. Организации, которые создают гибридные решения, скорее всего, позиционируют свои архитектуры где-то на этом отрезке.

## Интеграция с облачной средой

Облачные технологии помогут снизить эксплуатационные расходы путем сведения к минимуму необходимости сопровождения локальной инфраструктуры, а также обеспечат надежность и глобальный охват, упрощая администрирование. Это часто представляется идеальным решением для приложений, которым в той или иной форме требуется эластичность и масштабируемость.

Проще представить облако в виде среды, куда вы можете поместить свои приложения, не создавая никакой собственной инфраструктуры. Вам потребуется только подключение к Интернету и учетная запись в системе хостинг-провайдера. Аналогичным образом вы размещаете у хостинг-провайдера свой веб-сайт ASP.NET или PHP. Многие компании уже активно используют такие возможности. Автономные приложения, все ресурсы и компоненты которых могут размещаться удаленно, являются типичными кандидатами для перемещения в облако.

Но как быть, если вы не можете переместить все ресурсы для вашего приложения в облако? Например, когда ваше приложение обращается к данным, хранящимся в собственном центре обработки данных в соответствии с юридическими требованиями или договорными обязательствами, которые ограничивают физическое местоположение данных, или когда данные настолько конфиденциальны, что необходимо применять к ним специальные политики безопасности. Или, может быть, ваше приложение использует услуги, предоставляемые другими организациями, которые могут или не могут выполняться в облаке. Может быть, в вашей системе

присутствуют критически важные инструменты управления, которые интегрируются с приложением, но эти инструменты работают на настольных компьютерах в вашей организации.

**Мнение Бхарата**

Автономные приложения часто достаточно просто перенести в облако, но сложные приложения могут содержать компоненты, которые не подходят для развертывания в облаке.

На самом деле существует много причин, объясняющих, почему компании и частные лица рассматривают одни компоненты приложения в качестве основных кандидатов для перемещения в облако, в то время как другие компоненты совершенно для этого не подходят. В этой ситуации, чтобы воспользоваться преимуществами облачных вычислений, вы можете создать гибридное решение, некоторые компоненты которого будут работать в облаке, в то время как другие будут развернуты локально или в центрах обработки данных ваших партнеров по бизнесу.

## Задачи, связанные с интеграцией гибридных приложений

Когда вы планируете перенос некоторых компонентов существующего приложения из локальной среды в облачную, вы вполне вероятно столкнетесь с некоторыми проблемами, в частности в сфере обмена информацией и установления связи. Например, как облачные приложения будут вызывать локальные службы или отправлять сообщения локальным приложениям? Как облачные приложения будут получать доступ к локальным хранилищам данных? Как вы можете гарантировать, что все экземпляры приложений, работающих в облачных центрах обработки данных, будут оперировать актуальной информацией?

Кроме того, при перемещении компонентов приложения в облако, возникают опасения по поводу производительности, доступности, управления, аутентификации и обеспечения безопасности. Когда элементы приложения работают на удаленном узле, и доступ к ним возможен только через Интернет, смогут ли они по-прежнему успешно функционировать в рамках комплексного приложения?

Часто бывает полезно разделить задачи, связанные с гибридными приложениями, на отдельные категории в соответствии с основными проблемными областями.

Многочисленные задачи можно разделить на отдельные проблемные области. Это поможет вам максимально точно определить задачи и проанализировать возможные решения. Обычно выделяются следующие проблемные области:

- **Развертывание функциональных возможностей и данных в облаке**. Вполне вероятно, что вам придется изменить код существующих локальных приложений в той или иной степени, прежде чем эти приложения и соответствующие данные будут развернуты в облаке. Как минимум вам нужно изменить конфигурацию и, возможно, реорганизовать код таким образом, чтобы он соответствовал комбинации веб-ролей и рабочих ролей Windows Azure. Вы также должны определиться со способом развертывания данных в облаке и решить, что вы будете делать с приложениями, которые в силу различных причин не могут быть развернуты в рамках веб-ролей и рабочих ролей Windows Azure.
- **Аутентификация пользователей и авторизация запросов**. Большинству приложений придется выполнять аутентификации и авторизацию пользователей, клиентов или партнеров на каком-либо этапе процесса. Традиционно аутентификация проводилась на основе информации о пользователях из локального хранилища, но пользователи все чаще ожидают, что приложения будут предоставлять им возможность использовать универсальные учетные данные (например, существующие учетные записи в социальных сетях, таких как Windows Live ID, Google, Facebook и Open ID). Возможно, приложению также придется выполнять аутентификацию с использованием

учетных записей в корпоративном домене, чтобы предоставить возможность единого входа или федеративной идентификации совместно с партнерами.

- **Коммуникации и доступ к сервисам без границ**. Многие операции, выполняемые в гибридных приложениях, вынуждены выходить за границы, охватывая локальные системы, ресурсы организаций-партнеров и решения, размещенные в Windows Azure. Вызовы служб и сообщения должны быть в состоянии пройти через брандмауэры и устройства трансляции сетевых адресов (Network Address Translation, NAT) без ущерба для локальной безопасности. Механизмы связи должны эффективно работать через Интернет и компенсировать более низкую пропускную способность, более высокие задержки и менее надежную связь. Они также должны защищать содержимое сообщений, аутентифицировать отправителей и предотвращать последствия атак типа «отказ в обслуживании» (Denial of Service, DoS) на службы и конечные точки.

- **Бизнес-логика и маршрутизация сообщений**. Многие гибридные приложения должны обрабатывать бизнес-правила и рабочие процессы, которые содержат проверки условий, при этом выполняются различные действия в зависимости от результатов проверки этих правил. Например, приложению могут потребоваться возможности для обновления базы данных, отправки заявки соответствующему партнеру, который предоставляет транспортные услуги или обеспечивает складское обслуживание, а также для аудита состава заказа (например, проверки кредитного лимита клиента) и сохранения заказа в другой базе данных для бухгалтерского учета. Эти операции могут включать сервисы и ресурсы, размещенные как в облаке, так и локально.

- **Синхронизация данных**. Гибридные приложения, которые выполняются частично локально и частично в облаке, выполняются в облаке и используют локальные данные или выполняются полностью в облаке, но более чем в одном центре обработки данных, должны обеспечивать синхронизацию и репликацию данных между различными узлами и в рамках различных сетей. Синхронизация может затрагивать только некоторые строки и столбцы, вы также можете выполнять преобразование этих данных.

- **Масштабируемость, производительность и доступность**. Облачные платформы обеспечивают высокую масштабируемость и надежность, но размещение части компонентов приложения в облаке, в то время как другие компоненты работают локально, может привести к проблемам с точки зрения производительности. Ограниченная пропускная способность, использование интерфейсов с интенсивным обменом данными и возможность дросселирования в Windows Azure могут обусловить необходимость кэширования данных на соответствующих узлах. Это также обуславливает необходимость развертывания дополнительных экземпляров облачных компонентов приложения с целью выполнения различной нагрузки и обеспечения защиты от кратковременных проблем с сетью и необходимость предоставления экземпляров, которые будут размещаться ближе к пользователям, позволяя минимизировать время отклика.

- **Мониторинг и управление**. Компании должны быть в состоянии эффективно управлять удаленными приложениями, размещаемыми в облаке, ежедневно контролировать работу этих приложений, а также иметь доступ для ведения журналов и проверки данных. Они также должны иметь возможность настраивать, обновлять и администрировать приложения таким образом, как будто эти приложения запущены в локальных центрах обработки данных. Кроме того, компаниям необходима релевантная и своевременная бизнес-информация из их приложений, иначе они не смогут выполнять текущие требования, например в рамках соглашения об уровне обслуживания (Service Level Agreements, SLA), а также не смогут составлять реальные планы на будущее.

Windows Azure поможет вам решить поставленные задачи, платформа предоставляет комплексный пакет облачных сервисов, инструментов управления и разработки, которые облегчают создание интегрированных и гибридных приложений. Вы также можете использовать многие из этих сервисов, когда само приложение находится в Windows Azure и не имеет локальных компонентов.

**Мнение Маркуса**

Сервисы, предоставляемые Windows Azure, полезны как с точки зрения интеграции локальных приложений с облачными средами, так и с точки зрения управления приложениями, которые работают исключительно в облаке.

## Компания Trey Research

Trey Research – организация среднего размера, в которой работают 600 человек. Основной бизнес компании связан с производством узкоспециализированного аппаратного обеспечения и электронных компонентов для научно-исследовательских организаций, лабораторий и производителей оборудования. Компания продает свою продукцию через Интернет, используя для обработки заказов приложение Orders.
Как ориентированная на Интернет организация, Trey Research стремится свести к минимуму всю непрофильную деятельность и сосредоточиться на оптимизации своего онлайн-сервиса и среды, не отвлекаясь на «физические» проблемы, такие как транспортировка и доставка. По этой причине Trey Research организовала сотрудничество с внешними компаниями, которые предоставляют эти услуги. Компании Trey Research достаточно сообщить партнеру по транспортировке, когда был получен заказ на производство, и указать дату отгрузки товара с завода Trey Research. Партнер по транспортировке может также сообщить Trey Research, когда была выполнена поставка заказчику.

Приложение Orders является лишь одним из многих приложений, которые Trey Research использует для управления своим бизнесом. Другие локальные приложения используются для управления выставлением счетов, сырьем, заказами поставщикам, а также для планирования производства и решения многих других задач. Тем не менее в этом руководстве мы будет рассматривать только приложение Orders и его взаимодействие с другими локальными системами, например основными приложениями для управления и мониторинга.

Разработчики компании Trey Research хорошо знакомы с различными продуктами и технологиями Microsoft, в том числе .NET Framework, ASP.NET MVC, SQL Server и системой разработки Microsoft Visual Studio. Они также знакомы с Windows Azure и стремятся использовать любую из доступных в Windows Azure функций, которая помогает упростить задачи разработки.

### Стратегия компании Trey Research

Trey Research одной из первых внедрила решения для облачных вычислений и платформу Windows Azure, эта платформа стала основной для новых приложений и расширенных функциональных возможностей существующих приложений. Компания Trey Research надеется свести к минимуму затраты на локальные центры обработки данных и обладает всем необходимым для использования новых технологий и бизнес-возможностей, предоставляемых облачными средами.

Несмотря на необходимость поддержания высокого качества и доступности существующих услуг в рамках достаточно большой клиентской базы, менеджеры Trey Research готовы вкладывать средства в разработку новых сервисов и модификацию существующих, чтобы получить от них максимальную отдачу и увеличить прибыль компании. В процессе перспективного планирования нужно учитывать возможные проблемы и задачи, например рост спроса на услуги, необходимость оптимизации инструментов формирования отчетности и средств предоставления бизнес-информации, необходимость повышения производительности и доступности приложений, а также управление растущей сложностью ландшафта, например из-за появления новых внешних партнеров.

# Приложение Orders

Приложение Orders компании Trey Research позволяет посетителям сайта размещать заказы на продукцию компании. Это веб-приложение, которое модернизировалось в течение долгого времени, чтобы воспользоваться преимуществами облачного развертывания в нескольких центрах обработки данных в различных географических точках, при этом некоторые важные сервисы и приложения остались в пределах локальной корпоративной инфраструктуры. Это обычный сценарий для многих организаций, поэтому должны быть найдены решения многочисленных проблем. Например, как приложение будет организовывать связь облачных сервисов с локальными приложениями, чтобы выполнять задачи, которые обычно подразумевают обмен данными в рамках корпоративной сети центров обработки данных, если в данной ситуации информация будет передаваться через Интернет?

В случае с Trey Research, некоторые критически важные функции, связанные с приложением, развернуты не в облаке. Приложения для управления и сопровождения операций, а также некоторые базы данных, размещаются локально в центрах обработки данных Trey Research. Функции транспортировки и доставки выполняются отдельными партнерскими организациями, присоединившимися к Trey Research. Эти партнеры по транспортировке могут сами использовать размещенные в облаке сервисы, но это не влияет на разработку и реализацию собственного приложения Trey Research.

**Мнение Маркуса**

Разработчики Trey Research применяют самые современные из доступных технологий: Visual Studio 2010, ASP.NET MVC 3.0 и .NET Framework 4. Они обслуживают и модернизируют приложение Orders при помощи этих технологий.

## Оригинальное локальное приложение Orders

Когда Trey Research разработала приложение Orders, оно полностью размещалось в собственном центре обработки данных компании, за исключением служб партнеров по транспортировке и доставке. Приложение состояло из двух отдельных компонентов: само приложение Orders (веб-сайт и связанная с ним бизнес-логика) и набор приложений для управления и подготовки отчетности.

Кроме того, общедоступное веб-приложение Orders должно поддерживать масштабирование с учетом ожидаемого роста спроса, а приложениям для управления и отчетности такая возможность не требуется. Специалисты Trey Research предлагали в случае роста спроса на услуги масштабировать приложения для управления и отчетности путем развертывания дополнительных серверов в рамках локальной веб-фермы в их центре обработки данных. На рисунке 1 показано приложение, работающее локально.


**Рисунок 1**

**Общий обзор приложения Orders, которое Trey Research размещает локально**

Как видно из рисунка 1, приложение Orders взаимодействует с несколькими базами данных. Оно использует аутентификацию формами ASP.NET для идентификации клиентов и ищет информацию о них в таблице «Клиенты» по уникальному идентификатору пользователя. Приложение получает список продуктов, которые Trey Research предлагает, из таблицы «Продукты» в базе данных и сохраняет информацию о заказах в таблице «Заказы». В таблице «Журнал аудита» в локальной базе данных содержится различная информация, получаемая, в том числе, во время выполнения и диагностики, а также сведения о значительных заказах (например, если сумма заказа превышает установленное пороговое значение). Менеджеры могут получить бизнес-информацию из таблицы «Заказы» с помощью служб отчетов SQL Server.

Приложение Orders отправляет сообщение соответствующему партнеру по транспортировке, когда клиент размещает заказ. В настоящее время компания Trey Research работает с двумя партнерами по транспортировке: один осуществляет локальную доставку в соседние штаты, другой – за пределы региона. В сообщении указывается предполагаемая дата поставки и информация об упаковке (например, вес и количество упаковок). Партнер по транспортировке может отправить сообщение обратно в приложение Orders после того, как товары будут доставлены клиенту, поэтому таблица «Заказы» в базе данных может быть обновлена.

В связи со специфическим характером продуктов, которые Trey Research производит, компания должна также гарантировать соблюдение законодательных требований в отношении сбыта некоторых товаров, в частности, в случае экспорта в другие страны и регионы. Эти требования подразумевают ведение подробных записей о продажах некоторых электронных компонентов, которые могут входить в состав продуктов компании Trey Research, а также компонентов оборудования, которые могут быть использованы при производстве боеприпасов. Анализ заказа представляет собой сложный и строго контролируемый процесс, который выполняется приложением для обеспечения соответствия нормативным требованиям от стороннего поставщика. Это приложение работает на специально настроенном сервере.

Наконец, компания Trey Research использует отдельные приложения для мониторинга приложения Orders, управления данными, которое оно использует, и выполнения общих административных задач. Приложения для мониторинга и управления взаимодействуют с корпоративными системами Trey Research, применяемыми для выполнения таких задач, как выставление счетов и управление складскими запасами, но эти вопросы не относятся к темам и сценариям, представленным в настоящем руководстве.

## Гибридное приложение для Windows Azure

С появлением доступных и надежных облачных хостинговых сервисов, компания Trey Research решила изучить возможность перемещения приложения в Windows Azure.

Приложения, охватывающие облачные и локальные системы, могут использовать: веб-роли, рабочие роли и роли виртуальных машин, размещенные в одном или нескольких центрах обработки данных на базе Windows Azure; базы данных технологической платформы SQL Azure в том же или в разных центрах обработки данных; удаленные сервисы сторонних поставщиков, созданные с использованием Windows или других технологий; локальные ресурсы, такие как базы данных, службы и общие файловые ресурсы. Интеграция и организация взаимодействия между этими ресурсами и сервисами не является тривиальной задачей, особенно когда между ними развернуты брандмауэры и маршрутизаторы.

> Одной из самых важных проблем при развертывании приложений в облаке является задача предоставления облачным приложениям и сервисам доступа к внутренним службам и хранилищам данных.

Кроме того, приложения необходимо разрабатывать и развертывать таким образом, чтобы они были масштабируемыми и могли выдерживать различные нагрузки, были надежными и доступными в любое время, безопасными и предоставляющими полный контроль доступа, а также простыми для управления и мониторинга.

На рисунке 2 представлен общий обзор архитектуры, реализованной компанией Trey Research для своего гибридного приложения. Несмотря на кажущуюся сложность рисунка 2, приложение Orders работает так же, как и в случае полностью локального размещения. Проектные решения и реализации каждого компонента приложения подробно описаны в последующих главах настоящего руководства.

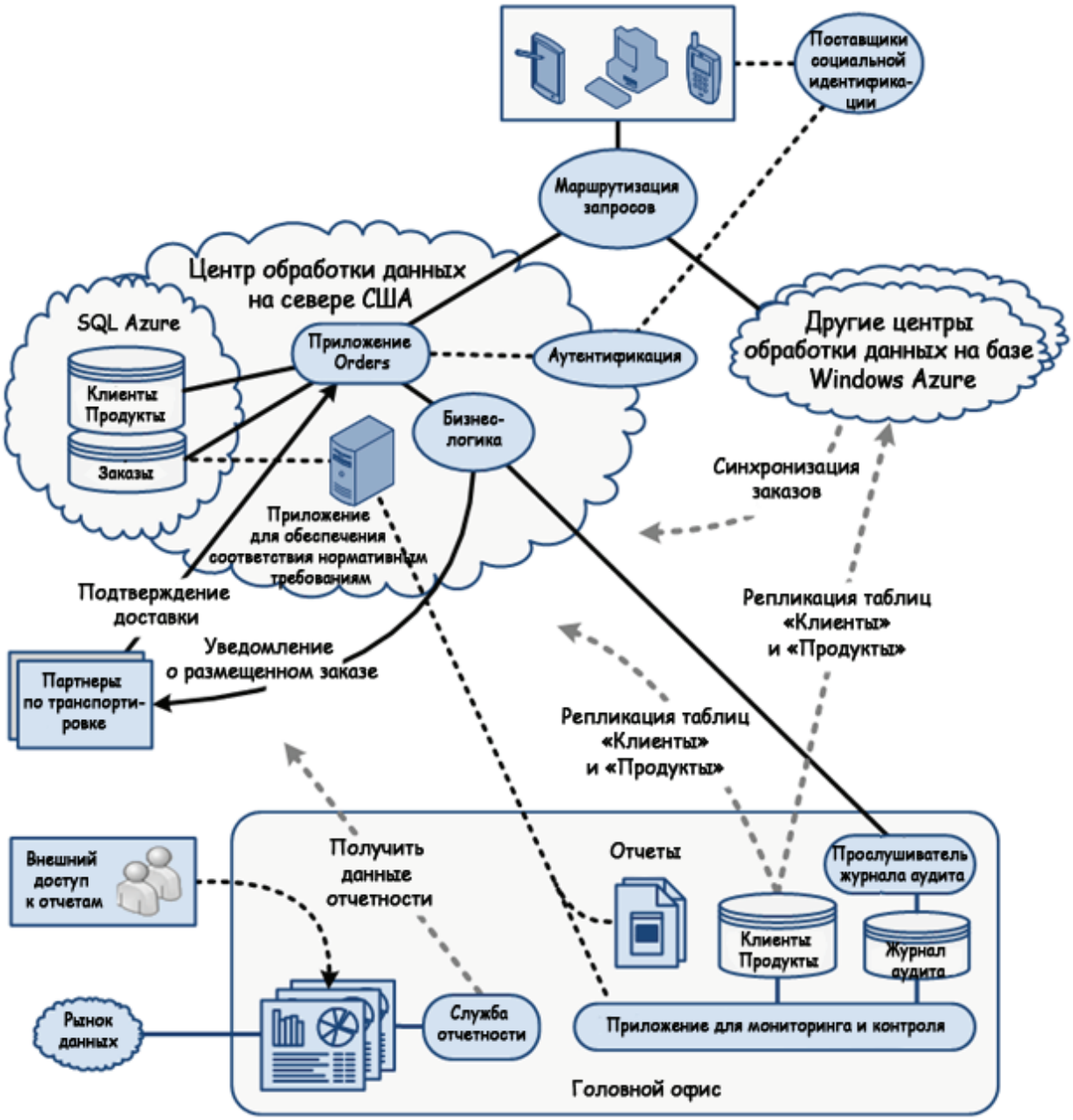



**Рисунок 2**

### Общий обзор приложения Orders, которое Trey Research размещает в облаке

Краткая информация о функциях, представленных на рисунке 2:

- Запросы клиентов проходят через диспетчер трафика (Traffic Manager) Windows Azure, который перенаправляет клиента к экземпляру приложения Orders, развернутому в ближайшем центре обработки данных с учетом времени отклика и доступности.
- Вместо аутентификации формами ASP.NET используются поставщики социальной идентификации, такие как Windows Live ID, Yahoo! или Google. Служба Windows Azure Access Control Service (ACS) управляет этим процессом и возвращает маркер, содержащий уникальный идентификатор пользователя, в приложение Orders. Приложение Orders использует этот маркер, чтобы найти информацию о клиенте в таблицах «Клиенты» и «Продукты» базы данных, развернутой в локальном центре обработки данных на базе SQL Azure.

- Новые клиенты могут зарегистрироваться в системе Trey Research и получить учетную запись для доступа к приложению Orders (регистрация осуществляется отделом по работе с клиентами головного офиса посредством внеполосной операции, и этот процесс не показан на рисунке 2). После регистрации нового клиента в системе управления взаимоотношениями с клиентами компании Trey Research сведения об учетной записи в таблице «Клиенты» локальной базы данных синхронизируются с базами данных SQL Azure во всех центрах обработки данных. Это позволяет клиентам получить доступ к приложению в любом из глобальных центров обработки данных компании Trey Research.

  **Примечание**

  После первоначального развертывания специалисты Trey Research решили предоставить клиентам возможность изменять некоторые данные о себе, такие как имя, адрес для выставления счетов и пароль при помощи приложения, работающего в облаке (это не относится к критически важным данным, таким как социальная идентификация пользователя). Эти изменения необходимо внести в локальную базу данных SQL Azure, а затем синхронизировать с локальными данными и передать в другие центры обработки данных на базе SQL Azure. В главе 2 под названием **«Развертывание данных и приложения Orders в облаке»** представлена информация о том, как решается эта задача. Тем не менее пример приложения, поставляемый с этим руководством, работает по-другому. Он поддерживает только регистрацию с помощью облачного приложения. Это делается, в первую очередь, для того, чтобы избежать необходимости настраивать службу SQL Data Sync, прежде чем использовать пример приложения.

- Приложение Orders отображает список продуктов, хранящихся в таблице «Продукты». Информация в таблице «Продукты» постоянно обновляется посредством синхронизации с основной базой данных, размещенной в центре обработки данных головного офиса.
- Когда клиент размещает заказ, приложение Orders:
  - Сохраняет информацию о заказе в таблице «Заказы» в БД в локальном центре обработки данных под управлением SQL Azure. Все заказы синхронизируются между всеми центрами обработки данных на базе Windows Azure, поэтому информация о статусе заказа будет доступна пользователям, независимо от того, к какому из центров обработки данных их направил диспетчер трафика.
  - Отправляет сообщение о заказе соответствующему партнеру по транспортировке. Транспортная компания выбирается с учетом типа продукта и места назначения.
  - Отправляет любую необходимую для аудита информацию, например о заказах, сумма которых превышает установленное пороговое значение, в локальное приложение для управления и мониторинга, которое будет хранить эту информацию в таблице журнала аудита в базе данных, развернутой в центре обработки данных головного офиса.
- Приложение для обеспечения соответствия нормативным требованиям от стороннего поставщика, работающее в рамках роли виртуальной машины в облаке, постоянно проверяет таблицу «Заказы» на соответствие правовым ограничениям и устанавливает флаг в таблице базы данных для тех заказов, которые требуют внимания со стороны менеджеров. Оно также ежедневно формирует отчет и сохраняет его на сервере, расположенном в центре обработки данных головного офиса.

- Когда партнер по транспортировке доставит заказ клиенту, он отправит сообщение в приложение Orders, запущенное в том центре обработки данных, из которого изначально поступило уведомление о размещенном заказе, после чего будет обновлена таблица «Заказы» в базе данных.
- С целью получения управленческой информации локальное приложение Reporting для подготовки отчетности использует инструменты бизнес-аналитики службы отчетов SQL Azure Reporting, которая работает в облаке. Отчеты формируются на основе данных из таблицы «Заказы». Эти отчеты могут использоваться совместно с данными, полученными из раздела «Рынок данных» (Data Market) в Windows Azure Marketplace, чтобы сравнить результаты с глобальными или локальными тенденциями. Отчеты доступны для конкретных внешних пользователей, например удаленных партнеров и сотрудников.

Имейте в виду, что для упрощения некоторые функции и процессы, описанные в настоящем руководстве, не в полном объеме реализованы в примере приложения, который мы предоставляем, или могут работать несколько иным образом. Это упрощает установку и настройку примера приложения, вам не придется получать и настраивать учетные записи Azure в нескольких центрах обработки данных, это также относится к службам, таким как SQL Azure Data Sync и SQL Reporting.

## Каким образом специалисты Trey Research решали задачи интеграции

В этом руководстве в деталях описано, каким образом дизайнеры и разработчики Trey Research развивали приложение Orders, чтобы преобразовать его из полностью локального в гибридное, размещаемое в рамках облачной архитектуры. Чтобы помочь вам понять, как Trey Research использует некоторые технологии, доступные в Windows Azure и SQL Azure, на рисунке 3 они накладываются на архитектурную схему, которую вы уже видели в этой главе.

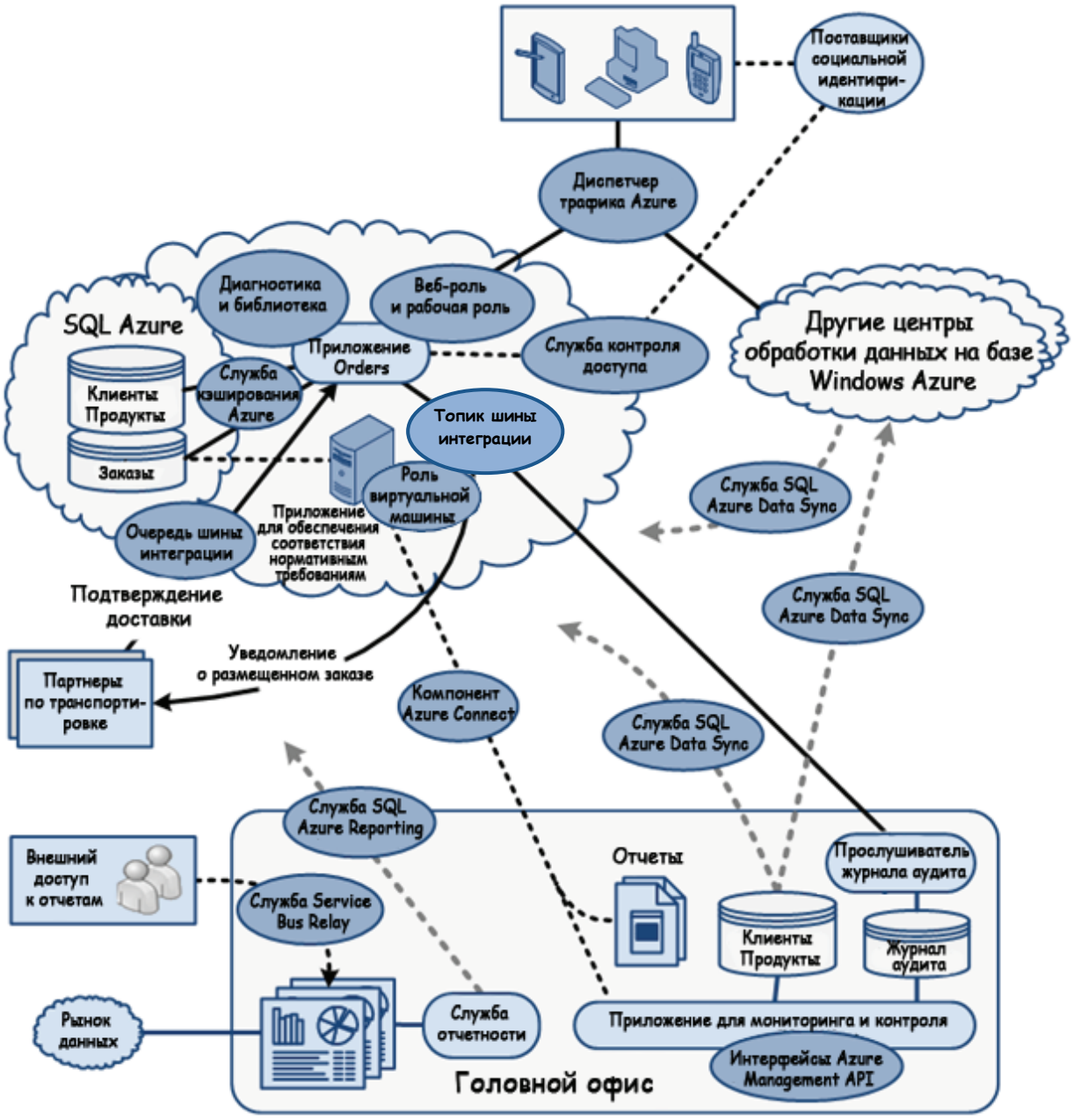



**Рисунок 3**

**Технологическая карта приложения Orders, которое Trey Research размещает в облаке**

Информация, содержащаяся в настоящем руководстве, относится к Windows Azure и SQL Azure, а также сервисам, предоставляемым ими на момент написания. Тем не менее платформа Windows Azure постоянно развивается, появляются новые возможности и функции. Более подробная информация представлена на странице «Что нового в Windows Azure?»: http://msdn.microsoft.com/en-us/library/windowsazure/gg441573.

### Поэтапный переход в облако

В процессе преобразования существующего решения в гибридное приложение вы можете рассмотреть вопрос о целесообразности проведения поэтапного перехода, поочередно перемещая приложения и службы в облако. Несмотря на то, что такой вариант кажется подходящим, поскольку позволяет проверить правильность работы системы на каждом из промежуточных этапов, это не всегда лучший из возможных подходов.

Например, разработчики компании Trey Research рассматривали возможность перемещения веб-приложений на платформу Windows Azure в виде веб-ролей, а также использования решений для создания подключения, таких как служба Windows Azure Connect, с целью предоставления приложениям доступа к локальным серверам баз данных. Такой подход вызывает задержки, которые будут негативно влиять на время отклика веб-приложения, поэтому потребуется решение для кэширования в облаке, чтобы решить эту проблему. Также в такой ситуации могут возникать сбои в работе приложения, если сетевое подключение будет прервано.

**Мнение Джаны**

Поэтапная или частичная миграция существующих локальных приложений в гибридные решения Windows Azure — очень не простая задача, и могут потребоваться значительные усилия по реорганизации этих систем с целью поддержания высокой безопасности, надежности и производительности, когда каналы связи переносятся в Интернет. Тем не менее для больших приложений необходимые усилия представляются целесообразными, учитывая сложность одномоментной миграции.

Другой типовой сценарий, который рассматривали специалисты компании Trey Research, подразумевал использование службы Windows Azure Service Bus Relay для предоставления размещенным в облаке приложениям доступа к локальным службам, которые еще не были перемещены в облако.
Как и служба Windows Azure Connect, Windows Azure Service Bus Relay зависит от надежности канала связи, производительность приложений может снижаться из-за высоких задержек и кратковременных сбоев подключения, которые нередко возникают в сети Интернет.

Приложения, которые разрабатывались с учетом требований сервис-ориентированной архитектуры (Service Oriented Architecture, SOA), будет проще переносить в несколько этапов, чем монолитные или сильносвязанные приложения. Возможно, полная реорганизация функций обеспечения взаимодействия и передачи данных в соответствии с требованиями гибридной среды не потребуется, хотя некоторые усилия могут быть связаны с обновлением этих функций, чтобы гарантировать эффективность их работы через Интернет, если эти функции были первоначально предназначены для развертывания в рамках высокоскоростных и надежных корпоративных сетей.

## Технологическая карта руководства

В следующих главах этого руководства подробно обсуждаются вопросы разработки и внедрения гибридного приложения Orders в компании Trey Research на основе ряда сценариев, связанных с приложением.
В приведенной ниже таблице представлены эти сценарии, проблемы интеграции, связанные с каждым из них, и технологии, которые Trey Research использовала для устранения этих проблем.

| Глава | Проблема | Технологии |
|---|---|---|
| **Глава 2** «Развертывание данных и приложения Orders в облаке» | Развертывание функциональных возможностей и данных в облаке.<br>Синхронизация данных. | SQL Azure<br>Служба SQL Azure Data Sync<br>Служба SQL Azure Reporting<br>Windows Azure Marketplace (Рынок данных)<br>Служба Service Bus Relay |
| **Глава 3** «Аутентификация пользователей в приложении Orders» | Аутентификация пользователей и авторизация запросов в облаке. | Служба Windows Azure Access Control<br>Платформа Windows Identity Framework<br>Функциональный блок для обработки неустойчивых неисправностей в библиотеке Enterprise Library |
| **Глава 4** «Реализация надежного обмена сообщениями и информацией в облаке» | Коммуникации и доступ к сервисам без границ. | Служба Windows Azure Connect<br>Очереди шины интеграции<br>Топик и правила шины интеграции |
| **Глава 5** «Обработка заказов в решении Trey Research» | Бизнес-логика и маршрутизация сообщений. | Очереди шины интеграции<br>Топик и правила шины интеграции |
| **Глава 6** «Максимизация масштабируемости, доступности и эффективности приложения Orders» | Масштабируемость, производительность и доступность. | Служба Windows Azure Caching<br>Диспетчер трафика Windows Azure<br>Функциональный блок для автоматического масштабирования в библиотеке Enterprise Library |
| **Глава 7** «Мониторинг и управление приложением Orders» | Мониторинг и управление. | Служба диагностики Windows Azure<br>Интерфейсы Windows Azure Management REST API<br>Командлеты Windows Azure Management |

**Примечание**

Некоторые из перечисленных функций и служб (например, роль виртуальной машины Windows Azure, служба Windows Azure Connect и диспетчер трафика Windows Azure) на момент подготовки руководства были доступны только в виде предварительной или бета-версии. Актуальная информация представлена на домашней странице Microsoft Windows Azure http://www.microsoft.com/windowsazure/. Кроме того, в данное руководство не была включена подробная информация о ACS. Подробнее ACS обсуждается в «*Руководстве по идентификации на основе утверждений и управлению доступом*» (см. http://claimsid.codeplex.com/), которое входит в серию руководств по Windows Azure.

## Резюме

В этой главе вы познакомились с гибридными приложениями, использующими преимущества, которые предоставляет размещение в облаке. Облачные сервисы предоставляют широкий спектр возможностей для развертывания приложений в соответствии с моделями «платформа как услуга» (Platform as a Service, PaaS) и «инфраструктура как услуга» (Infrastructure as a Service, IaaS), а также целый ряд встроенных функций, позволяющих решить проблемы, которые могут возникнуть в процессе развертывания существующих приложений в облаке или разработки новых гибридных приложений, работающих частично локально и частично в облаке.

В этой главе также представлено онлайн-приложение Orders, которое Trey Research использует для обработки заказов, и приведен рассказ о том, как специалисты компании преобразовывали полностью локальное приложение в гибридное решение, некоторые компоненты которого работают в облаке, в то время как остальные по-прежнему развернуты в локальном центре обработки данных. Наконец, в этой главе содержится описание итоговой архитектуры приложения Orders, поэтому вы знакомы с результатом.

В последующих главах этого руководства приложение будет рассматриваться более подробно. Мы предоставим гораздо больше информации о выборе подходящих технологий и о том, как специалисты Trey Research решали различные задачи, и как эти решения могут быть расширены или адаптированы к другим ситуациям.

Вы увидите, как Trey Research модифицировала свое приложение с целью обеспечения его безупречной работы в рамках локальной и облачной сред, а также с целью интеграции с внешними компаниями-партнерами (чьи приложения также могут выполняться локально или в облаке), при помощи служб, предоставляемых платформой Windows Azure и СУБД SQL Azure.

## Более подробная информация

- На сайте данной серии руководств (http://wag.codeplex.com/) представлены ссылки на онлайновые ресурсы, примеры кода, практические занятия, обратная связь и многое другое.
- Портал с информацией о Microsoft Windows Azure: http://www.microsoft.com/windowsazure/. Он содержит ссылки на официальные документы, инструменты и другие ресурсы. Вы также можете зарегистрироваться и получить здесь учетную запись Windows Azure.
- Ответы на свои вопросы вы можете найти на форуме Windows Azure: http://social.msdn.microsoft.com/Forums/en-US/category/windowsazureplatform.
- Юджинио Пейс (Eugenio Pace), главный менеджер программ группы Microsoft Patterns & Practices, отвечает за создание серии руководств по Windows Azure, к которой принадлежит этот документ. Более подробная информация о серии руководств представлена в блоге: http://blogs.msdn.com/eugeniop.
- Масаши Нарумото (Masashi Narumoto) является менеджером программ группы Microsoft Patterns & Practices, работающей над руководствами по Windows Azure. Его блог: http://blogs.msdn.com/masashi_narumoto.
- Скотт Денсмор (Scott Densmore), ведущий разработчик группы Microsoft Patterns & Practices, пишет в своем блоге о разработке приложений для Windows Azure: http://scottdensmore.typepad.com/.
- Блог Стива Маркса (Steve Marx) http://blog.smarx.com/ — отличный источник информации и новостей о Windows Azure.

- Программный код и документация, созданные в процессе работы над серией руководств по Windows Azure, представлены на веб-сайте Codeplex Windows Azure Guidance: http://wag.codeplex.com/.
- Подробные рекомендации и примеры, иллюстрирующие возможности службы Windows Azure Service Control Access, представлены в подготовленной группой Microsoft Patterns & Practices книге «*Руководство по идентификации на основе утверждений и управлению доступом*» (http://claimsid.codeplex.com/).

# Глава 2. Развертывание данных и приложения Orders в облаке

Первый этап перемещения частей системы Orders в облако в качестве элементов гибридного приложения потребовал от проектировщиков компании Trey Research рассмотрения способов развертывания этих частей на технологической платформе Windows Azure. Для платформы Windows Azure предлагается несколько вариантов развертывания функциональности приложения и широкий спектр связанных с этим служб, которые могут быть использованы решениями Trey Research при проектировании и построении гибридных приложений.

В этой главе вы узнаете, как компания Trey Research преодолела проблемы, связанные с развертыванием основных элементов приложения Orders в облаке, и как проектировщики связали это приложение со службами, предоставляемыми Windows Azure и технологической платформой SQL Azure.

## Сценарий и контекст

В изначальной реализации приложения Orders компоненты и службы, которые оно использует, работают локально и обращаются к данным, хранящимся в локальных базах данных SQL Server в центре данных Trey Research. Вы познакомились с архитектурой и описанием изначальной локальной реализации системы в главе 1 «Сценарий Trey Research». Специалистам компании Trey Research необходимо было решить, как разделить функциональность, какие типы ролей Windows Azure использовать и как предложенная архитектура повлияет на безопасность, производительность и надежность приложения.

Помимо этого, проектировщики должны были решить, где и как будут размещены используемые приложением данные, когда некоторые части приложения будут размещены удаленно и взаимодействие с ними будет осуществляться через Интернет, и как обеспечить возможность формирования бизнес-отчетов по этим данным.

Когда они анализировали существующее приложение Orders с точки зрения переноса определенных частей в Windows Azure, стало очевидно, что модули приложения для управления и формирования отчетов, которым не требуется такое же масштабирование, как общедоступным сайтам, должны оставаться локальными. Это позволило компании Trey Research осуществлять более строгий контроль над теми функциями приложения, которые требуют повышенной безопасности, и теми, которые должны, по мнению специалистов компании, находиться в собственном центре данных исходя из соображений логистики. Однако компания Trey Research хотела, чтобы некоторые открытые части отчетных данных были доступны доверенным партнерам для использования в их системах.

Эта общедоступная часть приложения могла бы легко размещаться в облаке, как это и было выполнено в виде отдельного приложения, и эта часть приложения потребует наибольшего масштабирования в процессе использования для удовлетворения требований к расширению. Это позволило компании Trey Research использовать все преимущества облака с точки зрения надежности, доступности, безопасности,

меньших эксплуатационных затрат, сниженных требований к локальной инфраструктуре и возможности быстрого масштабирования в обоих направлениях, чтобы выдержать требуемую пиковую нагрузку.

Есть и другие преимущества размещения в Windows Azure, которые являлись серьезными причинами для переноса общедоступных частей приложения Orders в облако. Среди них: возможность развертывания в нескольких центрах данных, находящихся в различных местах, чтобы обеспечить меньшее время ответа и максимизировать доступность для клиентов. Благодаря диспетчеру трафика Windows Azure запросы к приложению автоматически направляются к тому экземпляру, который обеспечит более эффективную обработку с точки зрения пользователя. Диспетчер трафика также обрабатывает ошибки определенных экземпляров, перенаправляя запросы другим экземплярам.

Кроме того, компания Trey Research смогла воспользоваться преимуществами встроенной функции кэширования распределенных данных для временных данных, используемых веб-сайтами общего доступа, службы аутентификации на основе утверждений для простоты реализации федеративной аутентификации. Компания также воспользовалась функцией подключения для безопасной связи и доступа к службам через границы между локальной сетью и облаком, возможностями синхронизации данных, мощной облачной системой формирования отчетов и доступностью платформ и компонентов сторонних производителей для упрощения разработки.

**Мнение Маркуса**

Преимущества имеющихся компонентов, служб, платформ и функций, разработанных и оптимизированных для работы в облаке, упрощают разработку облачных приложений.

На рисунке 1 показано в общем виде, как компания Trey Research решила разделить приложение на облачные и локальные части.

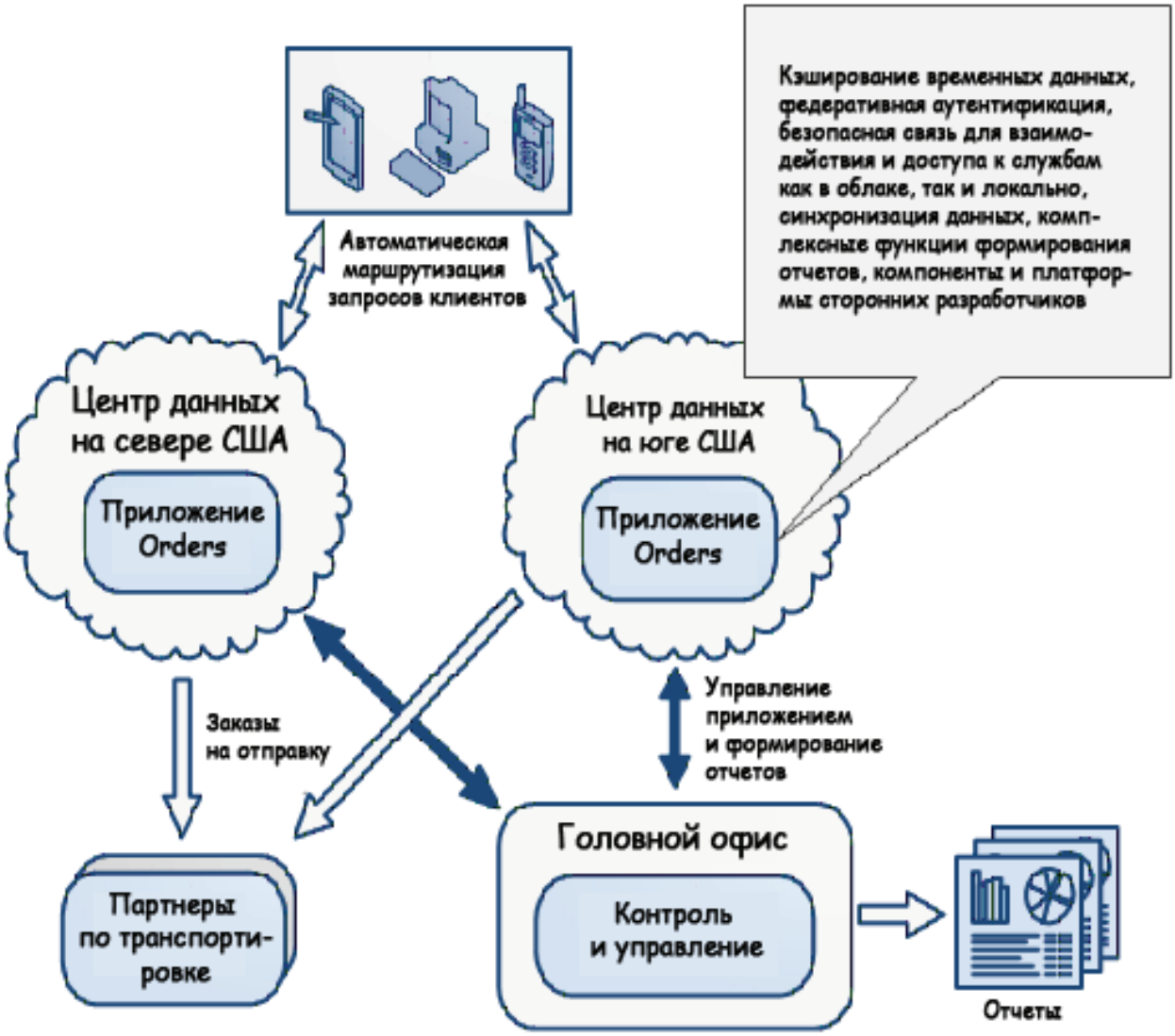



**Рисунок 1**

**Общая схема разделения между облаком и локальным размещением**

В этой главе вы узнаете, как проектировщики компании Trey Research определили, где размещать используемые приложением данные, как реализовать механизм синхронизации, который обеспечивает доступность и согласованность необходимых данных во всех местах, где они требуются, и как они поддерживают широкие возможности формирования интеллектуальных бизнес-отчетов. Для этих решений проектировщикам надо было рассмотреть существующие варианты, преимущества и недостатки каждого из них.

## Развертывание данных и приложения Orders в облаке

Приложение Orders — это веб-сайт, поэтому проектировщики компании Trey Research понимали, что его можно легко развернуть в Windows Azure в виде веб-роли. Развертывание нескольких экземпляров веб-роли позволяет масштабировать веб-сайт в соответствии с требованиями и гарантирует необходимые компании Trey Research доступность и надежность. Задачи фоновой обработки, которые выполняются после того, как пользователь разместил заказ, передаются рабочей роли. Trey Research может развернуть несколько экземпляров рабочей роли, чтобы справиться с изменяющейся нагрузкой, когда клиенты размещают заказы на веб-сайте.

**Мнение Маркуса**

Вы разрабатываете новое веб-приложение или адаптируете существующее веб-приложение для Windows Azure практически так же, как разрабатываете элементы для локального размещения в собственном центре данных. Однако есть некоторые аспекты, которые отличаются, например управление состоянием сессии, хранение данных и конфигурация.

Веб-сайт Orders в процессе работы требует доступа к нескольким элементам данных. Эти данные включают список продуктов, которые может заказать покупатель, список клиентов, чтобы приложение могло аутентифицировать посетителя и получить доступ к информации о нем, заказах, которые покупатель размещает на веб-сайте, а также к информации о проведении аудита и записях журнала в реальном времени. Проектировщики из Trey Research должны были решить, где и как разместить все эти элементы и также выбрать подходящий механизм хранения этих данных.

### Выбор размещения данных

Все элементы гибридного приложения, размещены ли они локально, в облаке или у партнеров, потребуют доступа к данным. Важной частью разработки гибридного приложения является размещение этих данных в соответствующих местах, чтобы максимизировать эффективность и производительность, при этом сохраняя безопасность и выполняя все требования к репликации и синхронизации. Как правило, данные должны быть расположены как можно ближе к приложениям и компонентам, которые их используют. Однако это не всегда рекомендуемо или возможно в зависимости от конкретных условий.

Основное решение — будут ли данные размещены локально или удаленно (например, в облаке или у партнеров). Приложение Orders использует четыре типа данных:

- Информация о клиентах, включая такую важную информацию, как лимиты кредита и информация о платежах. В нее входит информация личного порядка (Personally Identifiable Information, PII), которая должна быть защищена в максимальной степени.
- Информация о товарах, например каталог продукции, цены и подробное описание. Поскольку компания Trey Research производит все продукты под заказ, информация о наличии на складе отсутствует.
- Информация о заказах, включая всю подробную информацию о заказах, размещенных клиентами, и информацию о доставке.
- Информация журнала аудита, такая как события и исключения, посланные приложением, и подробности о заказах общей суммой более 10 000 долл. США. Эти данные могут содержать важную информацию и должны быть полностью защищены от доступа сотрудников, не имеющих отношения к администрации.

Проектировщики компании Trey Research рассматривали три варианта размещения данных, используемых приложением Orders. Они могли развернуть все данные в облаке, оставить все данные локально или перенести некоторые данные в облако, а остальные сохранять локально.

### Развертывание всех данных в облаке

Размещение всех данных в облаке, чтобы они были ближе к приложению Orders, поможет максимально повысить производительность и свести к минимуму время ответа, а также устранит необходимость синхронизации данных между облаком и локальным размещением. Это также позволит компании Trey Research пользоваться преимуществами масштабируемости и производительности как служб хранения Windows Azure Storage, так и SQL Azure, при этом и то, и другое хранилище данных обеспечивает надежный, быстрый и эффективный доступ к данным для приложения и при необходимости позволяет легко расширить доступность хранилища.

Однако развертывание всех данных в облаке будет означать, что приложения головного офиса будут вынуждены обращаться к данным через Интернет. Облачное размещение в этом случае приводит к тому, что сотрудники головного офиса сталкиваются с задержками и отказами связи в связи с работой в сети Интернет, а также с проблемами с производительностью. Кроме того, будут требоваться дополнительные расходы на доступ к данным из локальных приложений. К тому же, стоимость хранилища при размещении больших объемов данных или нескольких баз данных может стать слишком высокой, и вероятно потребуется синхронизация данных между этими размещениями, если приложение размещено в более чем одном центре данных.

### Локальное хранение всех данных

Локальное хранение всех данных означает, что администраторам и операторам компании Trey Research будет проще управлять и гарантировать безопасность данных, особенно если большая часть операций обновления данных выполняется локально сотрудниками и другими приложениями организации. Этот подход позволяет компании Trey Research гарантировать соответствие законодательным и нормативным требованиям к размещению и безопасности важной информации. Кроме того, нет необходимости в переносе или развертывании данных удаленно, а другие операции, такие как резервное копирование данных, выполняются намного проще.

Однако хранение всех данных локально означает, что удаленные приложения и службы в облаке или у партнеров должны получать доступ к данным через Интернет, хотя с этой проблемой можно в некоторой степени побороться разумным применением кэширования. Проектировщики компании Trey Research также рассматривали вопрос, будет ли возможно реализовать требуемую бизнес-логику так, чтобы она работала надежно и безопасно, когда удаленные приложения и службы выполняют обновления нескольких баз данных через Интернет.

**Мнение Джаны**

Доступ к хранящимися локально данным из приложений, размещенных в облаке, как правило, не является лучшим подходом в связи с внутренней задержкой сети и ненадежностью Интернета. Если вы решили выбрать этот подход, то должны использовать надежный механизм кэширования, например кэширование Windows Azure, чтобы минимизировать влияние проблем сети.

### Развертывание некоторых данных в облаке

Развертывание некоторых данных в облаке и хранение остальных локально имеет несколько преимуществ. Например, данные для приложений и служб, которые требуют быстрого и надежного доступа, могут размещаться в облаке, ближе к приложению или службе, которые их используют, а данные, которыми пользуются в основном приложения головного офиса, могут оставаться локальными для обеспечения быстрого и надежного доступа для этих приложений. Помимо этого, данные, которые имеют законодательные или нормативные ограничения с точки зрения места их хранения или которые требуют реализации особого механизма обеспечения безопасности, могут оставаться локальными. И наконец, данные, которым не требуется масштабирование, могут оставаться локальными, уменьшая стоимость их размещения, а данные с необходимостью масштабирования могут быть размещены в хранилище Windows Azure или SQL Azure, чтобы воспользоваться преимуществами этих служб в области масштабирования.

Однако развертывание некоторых данных в облаке означает, что в случае использования как облачными, так и локальными приложениями, доступ к ним должен быть реализован и через Интернет. Будет необходим достаточно надежный и безопасный механизм подключения, а также решение для репликации и синхронизации, чтобы обеспечить согласованность данных для всех размещений.

## Как компания Trey Research выбирала место для развертывания данных

После рассмотрения различных вариантов развертывания данных компания Trey Research приняла следующее решение по размещению информации, используемой приложением Orders.

### Данные клиентов

Информация о клиентах обрабатывается собственными операторами компании Trey Research при помощи локальной системы бухгалтерского учета, которую компания использует в пределах всей организации. Компания Trey Research требует от клиентов регистрации через головной офис, и операторы добавляют клиентов в локальную базу данных. Планируется, что с использованием приложения Orders клиенты смогут изменять некоторую свою информацию (эта функциональность еще не реализована), но приложение не позволит им изменять критические данные удостоверений и прочие защищенные данные. Данные клиентов, скорее всего, будут относительно статичными и не будут сильно изменяться со временем.

Компания Trey Research решила сохранять главную базу данных клиентов локально для обеспечения максимальной безопасности и обеспечения возможностей эффективной работы с этими данными для всех локальных приложений. Однако данные о клиентах также необходимы веб-сайту Orders для аутентификации посетителей и для принятия от них заказов. Поэтому для максимизации производительности и надежности компания Trey Research решила разместить реплику данных о клиентах в облаке, ближе к веб-сайту Orders.

Это означает, что необходим двунаправленный механизм синхронизации для того, чтобы изменения данных клиента, внесенные локальными операторами, реплицировались во все центры данных, где размещается приложение Orders, а изменения некоторых данных клиентами с помощью приложения Orders реплицировались в главную локальную копию данных и в базы данных SQL Azure других центров данных.

### Данные о продукции

Информация о продукции также обрабатывается операторами компании Trey Research. Эти данные могут обновляться только локально в связи с существующим локальным процессом производства и каталогами деталей, которые компания Trey Research использует во всей организации. Поскольку нет информации о запасах на складе (вся продукция изготовляется под заказ), данные о продукции относительно статичны.

Компания Trey Research решила сохранять основные данные о продукции локально для поддержки существующих возможностей эффективной работы с этими данными для всех локальных приложений. Однако для максимизации производительности и надежности, компанией Trey Research было принято решение разместить реплику некоторых полей данных о продукции (только данные, необходимые для перечисления продукции, предоставления детальной информации о продукции и принятия заказа) в облаке, ближе к приложению Orders. Это означает необходимость реализации однонаправленного механизма синхронизации для обеспечения репликации обновления данных о продукции, произведенных локальными операторами, во все центры данных, где размещено приложение Orders.

### Данные о заказах

Информация о заказах генерируется приложением Orders, которое запущено в облаке, и не может редактироваться ни в каком другом месте. Приложение Orders также обращается к данным о заказах для предоставления пользователям списка текущих заказов и информации о доставке. В отличие от данных о клиентах и продукции, которые являются относительно статичными, данные о заказах отличаются высокой динамичностью, поскольку они изменяются при размещении клиентом заказа и при доставке партнерами по транспорту.

В компании Trey Research решили, что не требуется размещать данные о заказах локально. Вместо этого данные о заказах размещаются только в облаке, ближе к приложению Orders. Однако, если приложение Orders развернуто более чем в одном центре данных, необходима двунаправленная синхронизация данных о заказах между центрами данных, чтобы клиенты видели свою информацию о заказах, если они по причине отказа приложения перенаправлены в другой центр данных (или в случае, если пользователь меняет географическое расположение). Единственной проблемой этого решения является то, что компания Trey Research не сможет более использовать службы формирования отчетов SQL Server для создания аналитических бизнес-отчетов непосредственно по данным. Далее в этой главе, в разделе «Выбор решения для формирования отчетов», вы узнаете, как в компании Trey Research решили эту проблему.

### Данные журнала аудита

Информация журнала аудита генерируется приложением Orders в ответ на события и исключения, сгенерированные приложением, а также для заказов на общую сумму более 10 000 долл. США. Она также формируется другими локальными приложениями в компании Trey Research. Таким образом база данных Audit Log для журнала аудита — это полное хранилище для всех служб управления приложением и контроля.

Компания Trey Research пришла к выводу, что поскольку наиболее интенсивный доступ к этим данными осуществляется из инструментов для контроля и приложений для административного управления, эти данные должны оставаться локальными. Кроме того, правительственные нормативы по продаже некоторых высокотехнологичных продуктов, которые выпускает компания Trey Research, подразумевают, что в Trey Research должны иметь полную и точную информацию о таких продажах, а также сохранять ее локально. Локальное хранение данных журнала аудита, который может содержать важную информацию о приложении, также позволяет обеспечить их полную безопасность от неавторизированного доступа в домене Trey Research.

## Выбор механизма хранения данных

Определившись, что некоторые данные, используемые приложением Orders, будут размещены в Windows Azure, проектировщики компании Trey Research должны были определить подходящий механизм для хранения данных в облаке. Наиболее распространенные варианты — это хранилище Windows Azure, SQL Azure, другая система баз данных или собственный репозиторий.

### Хранилище Windows Azure

Хранилище Windows Azure позволяет хранить BLOB-объекты, таблицы и очереди. Очереди, как правило, используются для передачи информации между ролями и службами и не предназначены для использования в качестве механизма постоянного хранения. Однако компания Trey Research могла бы использовать хранилище таблиц или BLOB-объектов. И то, и другое является экономически эффективным способом хранения данных.

Хранилище BLOB-объектов идеально для хранения неструктурированной информации, такой как изображения, файлы и другие ресурсы. Хранилище таблиц идеально подходит для структурированной информации. Хранилище таблиц очень гибкое и может быть очень эффективным, особенно если структура таблицы разработана для обеспечения максимальной скорости обработки запросов. Оно также поддерживает репликацию между различными географическими расположениями, чтобы обеспечить быстрый и эффективный доступ для различных месторасположений клиентов. Хранилище таблиц значительно дешевле использования базы данных SQL Azure.

Однако хранилище таблиц не поддерживает знакомую технологию SQL для чтения и записи данных, а также некоторые стандартные типы реляционных баз данных. Данные хранятся как коллекции объектов, которые подобны строкам, но каждая из них имеет первичный ключ и набор свойств. Эти свойства состоят из имени и набора типизированных пар значений. Проектировщики компании Trey Research понимали, что перенос существующего приложения, которое использует базу данных SQL, в облако и использование хранилища таблиц Windows Azure потребует переработки модели данных и переписывания некоторых частей кода, относящегося к доступу к данным. А это повлечет дополнительные финансовые и временные затраты на процесс переноса.

Кроме того, хранилище таблиц Windows Azure не поддерживает концепцию транзакций базы данных, хотя и обеспечивает доступ в виде транзакций к одной таблице. И наконец, данные не могут быть напрямую импортированы из системы реляционной базы данных, как например SQL Server, в хранилище таблиц. Компании Trey Research было бы необходимо создать или заказать инструменты для преобразования и загрузки данных.

> **Примечание**
>
> Для получения дополнительных сведений об использовании хранилища таблиц Windows Azure см. раздел «Хранение данных о бизнес-расходах в хранилище таблиц Windows Azure» главы 5 руководства «Перенос приложений в облако» на сайте http://msdn.microsoft.com/en-us/library/ff803365.aspx#sec6.

### SQL Azure

SQL Azure — это служба базы данных с высокой производительностью, которая полностью поддерживает операции на основе SQL и может быть использована для создания в облаке реляционной базы данных. Она реализуется экземплярами SQL Server, установленными в центрах данных корпорации Microsoft.

SQL Azure предлагает большую часть основной функциональности локального SQL Server и предоставляет данные приложению, используя знакомый протокол SQL Server – поток табличных данных (Tabular Data Stream, TDS). Такая архитектура позволяет вам использовать тех же поставщиков данных .NET Framework (таких как **System.Data.SqlClient**) для подключения к базе данных и T-SQL для доступа и работы с данными. SQL Azure также совместим с существующими программными интерфейсами службы подключения, такими как Entity Framework (EF), ADO.NET и Open Database Connectivity (ODBC). Данные могут обновляться при помощи транзакций базы данных для обеспечения согласованности.

Эти преимущества позволяют разработчикам компании Trey Research избежать существенных изменений в коде приложений, а администраторам быстро и легко развернуть данные в SQL Azure без необходимости изменять схему таблиц. Администраторы и операторы компании Trey Research могут управлять базами данных SQL Azure при помощи портала управления Windows Azure и использования знакомых инструментов, таких как SQL Server Management Studio и инструменты баз данных Visual Studio. Также имеется ряд других инструментов для таких операций, как перенос и миграция данных, и инструменты командной строки для развертывания и администрирования.

Кроме того, синхронизация данных между локальными и размещенными в облаке базами данных реализуется легко с помощью службы Windows Azure Data Sync и программного интерфейса Data Sync API. Формирование аналитических бизнес-отчетов SQL Azure поддерживает с помощью службы SQL Azure Reporting.

Однако проектировщики компании Trey Research также должны были принять во внимание, что хотя SQL Azure и очень похож на SQL Server, однако некоторые концепции, такие как элементы управления серверного уровня или управление физическими файлами неприменимы к таким автоматически управляемым средам как SQL Azure. К тому же цена подписки на SQL Azure выше, чем на хранилище Windows Azure.

### Альтернативные системы баз данных или собственный репозиторий

Если вы используете реляционную систему баз данных или собственный репозиторий для хранения данных, вы легко осуществите миграцию данных в SQL Azure, в зависимости от существующего формата данных. Если же вы используете систему баз данных отличную от SQL Server (например, Mongo DB, см. http://www.mongodb.org/), вы сможете запустить эту систему управления базами данных в облаке с использованием рабочей роли Windows Azure или роли виртуальной машины.

Использование существующей системы управления базами данных или собственного репозитория, которые предоставляют доступ к данным для вашего приложения, означает, что вы, скорее всего, сможете использовать тот же код для доступа к данным, что и в локальной системе. Если разработчики знакомы с выбранным вами механизмом, это является преимуществом и поможет сократить время перехода и затраты на изучение новой системы.

Однако использование альтернативной системы управления базами данных или собственного репозитория означает, что вы должны самостоятельно обслуживать эту базу данных или репозиторий. Например, вы должны будете устанавливать обновления программного обеспечения управления базами данных и отлаживать свой собственный код. Вы также можете столкнуться с трудностями при импорте и переносе данных в другие механизмы хранения данных в будущем.

## Как компания Trey Research выбрала механизм хранения данных

Компания Trey Research использует SQL Server для хранения данных в локальных приложениях, включая приложение Orders. Форматы и типы данных, а также код доступа к данным предназначены для работы с SQL Server. Поэтому для компании Trey Research имело смысл использовать SQL Azure в качестве механизма хранения данных для гибридной версии приложения Orders. Дополнительные расходы в сравнении с использованием хранилища таблиц Windows Azure частично покрываются экономией на перепроектировании схемы и разработке кода.

Кроме того, в компании Trey Research хотели использовать транзакции базы данных и выполнять сложные запросы при работе с данными. Реализация кода для получения эквивалентной функциональности с использованием хранилища таблиц Windows Azure потребовало бы дополнительного времени на разработку и привело к дополнительным затратам. Администраторы компании Trey Research знакомы с SQL Server, включая инструменты для управления данными, и легко используют системы, основанные на SQL Server, поэтому работа с SQL Azure не требует изучения ими новых парадигм.

### Шифрование данных в хранилище и базах данных Windows Azure

Проектировщики компании Trey Research осознали, что при перемещении данных в облако они должны будут рассмотреть необходимый уровень защиты данных независимо от выбранного механизма их хранения. Важные данные, такие как пароль и номер кредитной карты клиента, а также информация личного порядка, такая как адрес и номер телефона, как правило, требует более высокой степени защиты, чем такая информация, как список продукции.

На момент написания этой книги ни хранилище Windows Azure, ни SQL Azure не поддерживали встроенного механизма шифрования данных. Это означает, что за шифрование и дешифрование важных данных, которые требуют дополнительного уровня защиты, отвечает приложение. В компании Trey Research этого достигли с использованием стандартных алгоритмов шифрования, предусмотренных на платформе .NET Framework, а также других библиотек кода.

**Примечание**

Сведения о шифровании данных в Windows Azure см. в статьях «Crypto Services and Data Security in Windows Azure» (Службы шифрования и безопасность данных в Windows Azure) на сайте MSDN Magazine http://msdn.microsoft.com/Ben-us/magazine/ee291586.aspx и «Encrypting Data in Windows Azure Storage» («Шифрование данных в хранилище Windows Azure») на сайте http://cm-bloggers.blogspot.com/2011/07/encrypting-data-in-windows-azure.html. Для получения более подробной информации о функциях безопасности SQL Azure см. раздел «Security Guidelines and Limitations (SQL Azure Database)» («Рекомендации и ограничения по безопасности (база данных SQL Azure)») по адресу http://msdn.microsoft.com/en-gb/library/ff394108.aspx.

## Синхронизация данных между облачными и локальными размещениями

Архитектура, которую компания Trey Research выбрала для приложения Orders, предполагает размещение некоторых данных в облаке в SQL Azure, а остальных — локально. Это значит, что разработчики компании Trey Research должны решить, как синхронизировать данные между этими размещениями для обеспечения их согласованности.

### Выбор решения для синхронизации данных

Выбор решения для синхронизации данных зависит как от типа хранилища данных, где они находятся, так и от требований к согласованности данных. Например, если данные всегда должны быть синхронизированы между различными размещениями, это решение должно отследить и реплицировать изменения, как только они происходят. Если приложение может нормально работать, когда данные в конечном итоге согласовываются, но могут быть устаревшими на протяжении некоторого времени, тогда процесса запланированной синхронизации может быть достаточно. В следующем разделе этой главы представлены варианты, которые рассматривались в компании Trey Research для синхронизации данных приложения Orders.

#### Синхронизация данных SQL Azure

Если данные развертываются в SQL Azure, естественным решением для синхронизации данных является использование службы SQL Azure Data Sync. Эта служба может синхронизировать данные между локальными базами данных SQL Server и одной или несколькими базами данных SQL Azure, размещенными в облаке. Синхронизация данных SQL Azure предлагает множество вариантов однонаправленной и двунаправленной синхронизации.

Использование службы SQL Azure Data Sync подразумевает, что разработчикам компании Trey Research не понадобится писать собственный код, поскольку синхронизация настраивается и управляется при помощи веб-портала Windows Azure. Это позволяет снизить расходы и время, которые требуются на внедрение решения в сравнении с созданием собственного решения.

Однако служба SQL Azure Data Sync работает только с базами данных SQL Server и SQL Azure; она не может быть использована в случае, если данные находятся в хранилище Windows Azure или в другой системе баз данных. Кроме того, служба SQL Azure Data Sync накладывает некоторые ограничения на типы данных столбцов и на допустимость пустых значений, поэтому может потребоваться внесение изменений в существующие схемы базы данных. Служба SQL Azure Data Sync обрабатывает конфликтующие изменения, сделанные в различных базах данных с использованием одной из немногих предварительно определенных политик. Настраивать эти политики невозможно, и служба SQL Azure Data Sync не предоставляет событий синхронизации для реализации вашего механизма.

Проектировщики компании Trey Research также осознали, что при некоторых сценариях синхронизации необходимо для завершения выполнить два прохода; эти данные сначала переносятся в центральную базу данных (которой может быть одна из существующих операционных бах данных), а затем в клиентские базы данных. Это означает, что в случае синхронизации нескольких баз данных с центральной некоторые экземпляры данных будут устаревшими до выполнения второго прохода синхронизации. Однако при простой синхронизации локальной базы данных с центральной базой данных SQL Azure все изменения выполняются за один проход.

**Примечание**

См. «Приложение A. Репликация, распространение и синхронизация данных», где представлена дополнительная информация об использовании службы SQL Azure Data Sync.

### Платформа Microsoft Sync Framework

Для синхронизации данных служба SQL Azure Data Sync использует компоненты платформы Microsoft Sync Framework. Мощная платформа синхронизации Sync Framework поддерживает любые типы данных, любые хранилища данных, любые протоколы передачи и любую топологию сети. Она не ограничена использованием только с базами данных SQL Server и SQL Azure.

Если бы разработчикам компании Trey Research нужен был больший контроль над процессом синхронизации, они могли бы использовать компоненты Sync Framework SDK непосредственно в коде. Преимуществом такого подхода является то, что приложение может реагировать на события, такие как изменение данных, и инициировать синхронизацию. Это приложение могло бы также обрабатывать события, происходящие в процессе синхронизации для управления конфликтами и ошибками или для лучшего обеспечения отслеживания. Конечно, это также будет означать, что разработчики должны будут написать дополнительный код для управления процессом синхронизации, что повлечет дополнительные расходы и время в сравнении с использованием службы SQL Azure Data Syn.

**Примечание**

Дополнительная информация о Sync Framework SDK представлена в центре для разработчиков Microsoft Sync Framework (Microsoft Sync Framework Developer Center) по адресу http://msdn.microsoft.com/en-us/sync/bb736753.

### Собственное решение для синхронизации или использование решения стороннего разработчика

Если бы компания Trey Research решила не использовать службу SQL Azure Data Sync или платформу Microsoft Sync Framework, проектировщики могли бы рассмотреть возможность разработки собственного решения или использования решения стороннего разработчика для синхронизации данных. В случае, если есть особые требования к синхронизации или репликации данных, собственный механизм может стать лучшим решением, чем готовое решение. Например, если компании Trey Research было бы необходимо выполнять особые виды синхронизации, которые не поддерживаются существующими решениями или службами от сторонних разработчиков, хорошим решением был бы собственный механизм, который передает сообщения между локальными службами и всеми центрами данных с использованием службы обмена сообщений через посредника шины интеграции Windows Azure.

Решение на основе службы обмена сообщениями является гибким и может быть использовано для различных типов репозиториев данных, поскольку служба, получающая сообщения об обновлении, может произвести операции обновления в репозитории с использованием соответствующих методов. Решения для репликации и синхронизации на основе системы обмена сообщениями особенно подходят для выполнения обновлений в реальном времени, однако это не является требованием для приложения Orders.

Помимо этого, решения для обмена сообщениями могут предоставлять дополнительную информацию о процессе синхронизации в ходе работы, например позволяя разработчикам отслеживать все модификации данных и обрабатывать конфликтные ситуации и ошибки соответствующим образом. Также возможно реализовать решение, которое следует принципам разделения очереди команд (Command Query Responsibility Segregation, CQRS), отделяя запросы, извлекающие данные, от команд, которые изменяют данные в репозитории.

Однако, если вы не сможете найти готовое решение стороннего разработчика, которое предоставляет необходимые функции и может работать с используемыми вами хранилищами данных, и вы примите решение разработать собственное приложение, то это потребует дополнительного времени на разработку и повлечет расходы на внедрение, тестирование и отладку.

> **Примечание**
>
> См. «Приложение A. Репликация, распространение и синхронизация данных» для получения дополнительной информации о создании собственного решения для синхронизации данных на основе службы обмена сообщениями.

## Как компания Trey Research выбрала решение для синхронизации данных

Проектировщики компании Trey Research решили использовать службу SQL Azure Data Sync в качестве решения для репликации и синхронизации для приложения Orders. Все данные хранятся либо локально в базе данных SQL Server, либо в SQL Azure в облаке, поэтому служба SQL Azure Data Sync сможет получить доступ и синхронизировать все требуемые данные. Экономия средств и времени на разработку в сравнении с собственным решением компенсируется в некоторой степени использованием службы SQL Azure Data Sync.

## Как компания Trey Research использует службу SQL Azure Data Sync

В компании Trey Research хранят информацию о продукции, клиентах и размещенных этими клиентами заказах. Компания Trey Research использует комбинацию локальной базы данных SQL Server и базы SQL Azure, размещенной в каждом из центров данных для управления данными, необходимыми для приложения Orders. Поэтому компания Trey Research решила реализовать репликацию и синхронизацию данных в приложении Orders.

> Этот раздел приведен только в информационных целях. Для простоты пример решения развернут в одном центре данных, поэтому не настроен на репликацию и синхронизацию между различными центрами данных.

Для различных типов информации, синхронизируемой в Trey Research, используются различные методы управления и хранения, а именно:

- Данные о заказах размещаются только в облаке приложением Orders с использованием SQL Azure и синхронизируются между центрами данных. Эта информации не направляется обратно в локальную базу данных.
- Информация о продукции находится только в локальной базе данных SQL Server, но данные, необходимые для размещения заказа, регулярно копируются в каждую базу данных SQL Azure каждого центра данных.
- Новые клиенты регистрируются локально, и их данные добавляются в базу данных SQL Server, которая находится в головном офисе. Эти данные реплицируются в базу данных SQL Azure в каждом центре данных, что позволяет клиентам входить и работать с приложением Orders без помощи системы головного офиса. В будущем, после того как учетная запись создана, в приложении Orders может быть разрешено клиентам изменять некоторые свои данные без необходимости вмешательства со стороны головного офиса, и эти изменения будут выполняться в базе данных SQL Azure, размещенной в том центре обработки данных, к которому подключен клиент (эта функциональность на данный момент не реализована, но в компании Trey Research хотели развернуть данные о клиентах с такими возможностями в будущем). Затем эти изменения будут передаваться назад в головной офис, а также реплицироваться во все остальные центры данных.

На рисунке 2 показано выбранное компанией Trey Research решение.

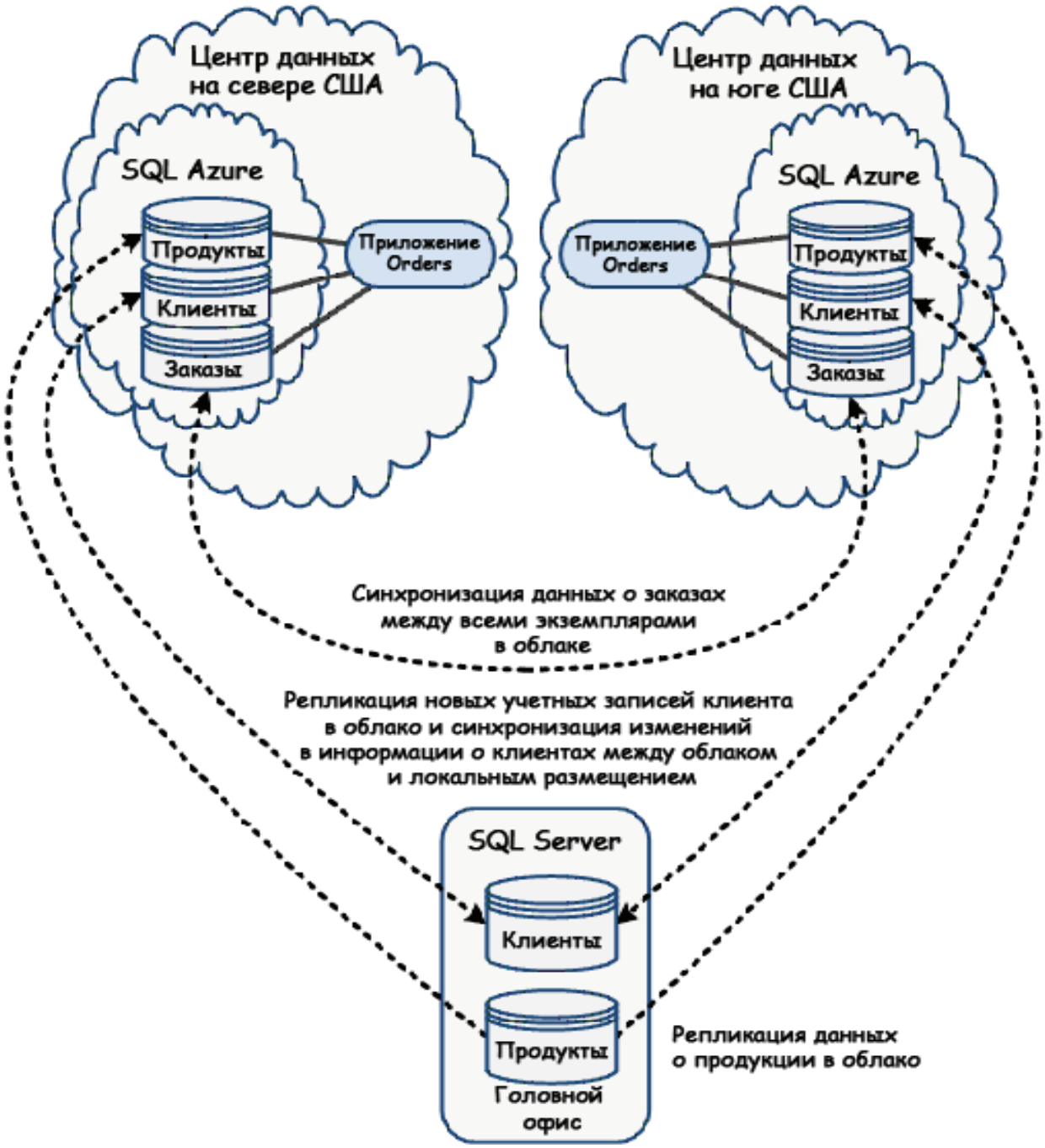



**Рисунок 2**

### Репликация данных в приложении Orders компании Trey Research

В этом решении данные о продукции синхронизируются в одном направлении, от локального размещения в облако. Данные о заказах реплицируются двунаправленно между центрами данных. Данные о клиентах также реплицируются в обе стороны, но при этом задействованы как локальные базы данных, так и размещенные в центрах данных.

На рисунке 3 показана физическая реализация этих подходов на основе службы SQL Azure Data Sync. Эта реализация использует четыре группы синхронизации. Каждая группа определяет набор данных для синхронизации и политику разрешения конфликтов для каждого типа данных (как описано выше, есть две перекрывающиеся группы синхронизации для репликации данных о клиентах). Базы данных SQL Azure расположены в центре данных на севере США и работают также в качестве центральной базы данных для синхронизации. Это ближайший центр данных к головному офису (головной офис компании Trey Research находится в Иллинойсе), поэтому выбор этого размещения позволяет снизить сетевую задержку при синхронизации с локальными базами данных.

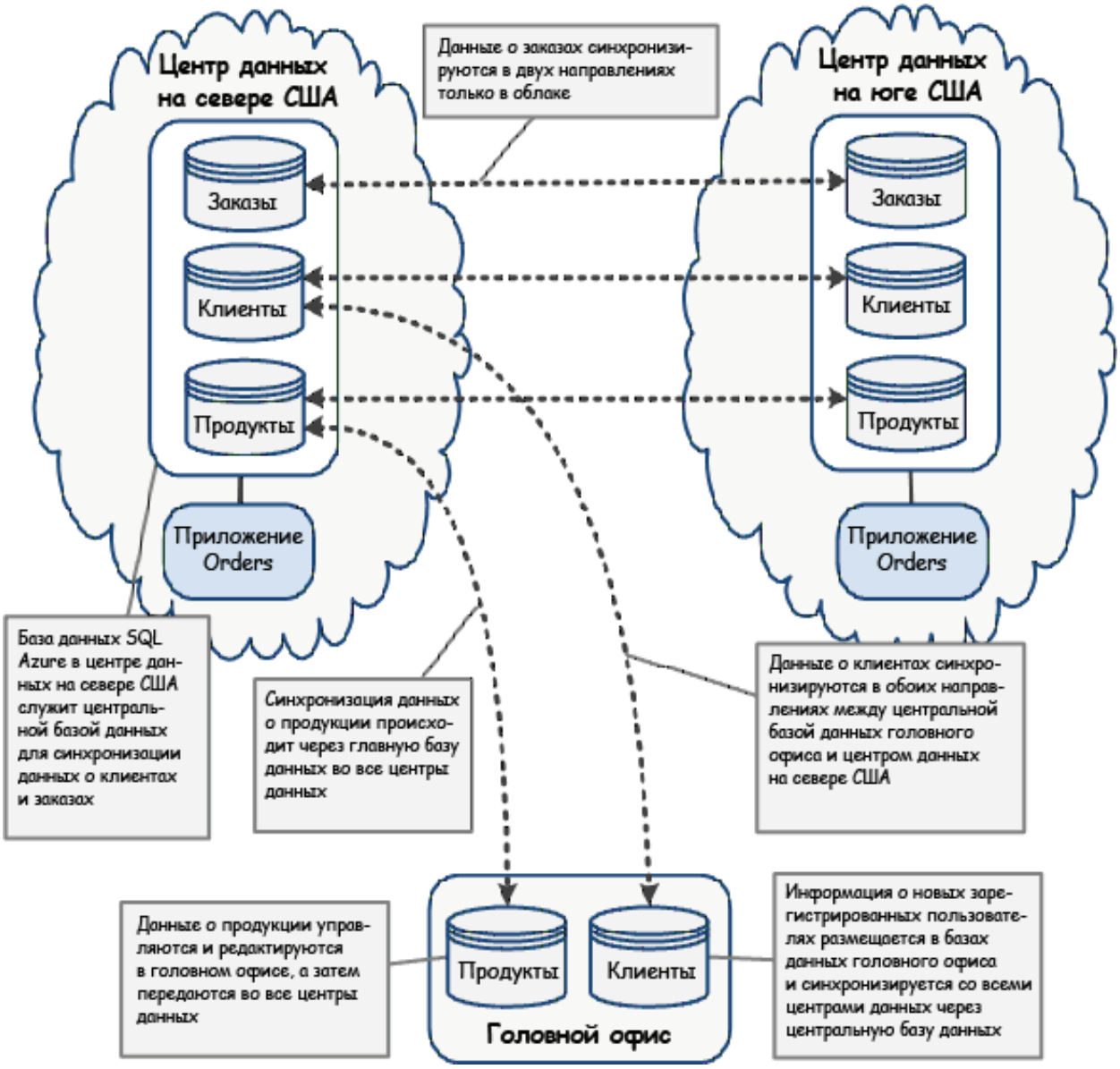



**Рисунок 3**

**Физическая реализация синхронизации данных для компании Trey Research**

Следующая таблица подытоживает реализованную компанией Trey Research конфигурацию каждой группы синхронизации.

| Имя и описание | Размещение центральной базы данных | Размещение задействованных баз данных и направление репликации | Набор данных | Политика разрешения конфликта |
|---|---|---|---|---|
| **Группа синхронизации «Продукты»**<br>Односторонняя синхронизация информации о продукции из локального размещения в облако | Центр данных на севере США | Головной офис<br>Синхронизация с центральной базой данных<br><br>Центр данных на юге США<br>Синхронизация из центральной базы данных | Таблица Product (для размещения заказа необходимы все столбцы) | Приоритет у центральной базы данных |
| **Группа синхронизации «Заказы»**<br>Синхронизация данных о заказах между центрами данных в облаке в двух направлениях | Центр данных на севере США | Центр данных на юге США<br>Двунаправленная | Таблицы с данными о заказах Order, OrderDetail и OrderStatus (все столбцы всех таблиц) | Приоритет у центральной базы данных |
| **Группа синхронизации «Клиенты»**<br>Двунаправленная синхронизация информации о клиентах между локальным размещением и облаком | Центр данных на севере США | Головной офис<br>Двунаправленная<br><br>Центр данных на юге США<br>Двунаправленная | Таблицы с данными о клиентах Customer и ACSIdentity (все столбцы всех таблиц) | Приоритет у центральной базы данных |

**Примечание**

Информация о каждом клиенте находится в двух таблицах: таблице **Customer** с общедоступной информацией о клиенте и таблице **ACSIdentit** с маркерами ACS, которые идентифицируют каждого клиента. Это необходимо, если сформировано несколько экземпляров ACS для аутентификации клиента; некоторые поставщики удостоверений возвратят различные идентификаторы пользователя ACS для каждого экземпляра. Поэтому база данных должна быть ассоциирована с более чем одним идентификатором ACS с каждой записью клиента.

Однако в примере решения, приведенного в данном руководстве, реализуется только один идентификатор ACS для каждого пользователя, который хранится в столбце **UserName** таблицы **Customer** базы данных Trey Research. Пример решения не содержит таблицу **ACSIdentity**, поскольку идентификаторы ACS хранятся в столбце **UserName** таблицы **Customer**.

См. главу 3 «Аутентификация пользователей в приложении Orders» для получения дополнительной информации о том, как в компании Trey Research используют ACS для аутентификации пользователей.

# Внедрение решения для формирования отчетов по данным, размещенным в облаке

В некоторых случаях перенос функциональности в облако сделает невозможным использование существующих служб. Например, в исходной локальной версии приложения Orders компания Trey Research для хранения всех корпоративных данных использовала SQL Server, а аналитические бизнес-отчеты формировались с помощью служб отчетов SQL Server. Однако, когда в компании Trey Research переместили исходные данные (базы данных заказов) в SQL Azure, проектировщики должны были понять, возможно ли использование локальной системы формирования отчетов. Теоретически возможно запускать систему формирования отчетов локально и подключаться к базе данных SQL Azure через Интернет, однако этот подход, скорее всего, вызовет большую нагрузку на пропускную способность и приведет к снижению производительности.

## Выбор решения для формирования отчетов

Проектировщики в компании Trey Research пришли к выводу, что следует найти лучшее решение для формирования аналитических бизнес-отчетов на основе размещенной в облаке базы данных заказов. Следующие разделы этой главы посвящены рассмотренным компанией Trey Research вариантам решения для формирования аналитических бизнес-отчетов.

### Службы отчетов SQL Server

Служба отчетов SQL Server — это функция SQL Server, которая позволяет получить комплексные аналитические бизнес-отчеты на основе данных, хранящихся в таблицах базы данных, а также в других источниках данных. Если вы используете SQL Server для хранения корпоративной информации, то службы отчетов SQL Server обеспечивают простое в использовании решение для создания отчетов.

Службы отчетов SQL Server могут считывать данные из реляционной, многомерной, основанной на XML и пользовательской баз данных и формировать отчеты в различных форматах, включая веб-ориентированный, предназначенный для печати и форматы компьютерных приложений. Они также поддерживают непосредственную публикацию отчетов в различные системы, такие как сервер отчетов, службы SharePoint, файловые службы общего пользования, внутренние архивные хранилища, а также приложения Office.

Службы отчетов SQL Server должны подключиться к источнику данных для анализа данных. Если источник данных удален по отношению к приложению, использующему отчет, процесс формирования отчета может генерировать значительный сетевой трафик, кроме того, может потребоваться служба, которая предоставит доступ к данным для использования службами отчетов SQL Server. Это может повлиять на безопасность источника данных, если он расположен удаленно от приложения, которое использует отчет.

### Служба отчетов SQL Azure

Служба отчетов SQL Azure — это предоставляемая Windows Azure служба, которая позволяет генерировать различные аналитические бизнес-отчеты в общих веб-форматах и форматах для использования с Microsoft Office. Если используется SQL Azure для хранения корпоративной информации, служба отчетов SQL Azure обеспечивают простое в использовании решение для создания отчетов.

Служба отчетов SQL Azure работает в том же центре обработки данных, что и база данных SQL Azure, поэтому сетевой трафик между службой отчетов SQL Azure и отображающим отчет приложением является минимальным. Это означает, что формирование отчетов будет происходить, скорее всего, намного быстрее и что будет обеспечиваться более высокая производительность по сравнению с подключением локального приложения для формирования отчетов к базе данных SQL Azure.

Хотя служба отчетов SQL Azure — это платная служба, требующая определенных расходов на покупку подписки, но их можно в определенной мере компенсировать за счет уменьшения расходов на трафик данных в локальную базу данных. Это решение также позволяет избежать изначальных расходов на разработку локальной системы формирования отчетов для организаций, которые не имеют такой системы. Также стоит принять к сведению, что служба отчетов SQL Azure менее интерактивна и обладает меньшими возможностями в сравнении со службами отчетов SQL Server и с другими решениями для формирования отчетов высокого класса, однако предлагаемый в ней набор форматов, как правило, достаточен для большей части возможных требований.

**Мнение По**

Широкий охват и актуальность информации являются жизненно необходимыми для всех компаний при управлении инвестициями, планировании использования ресурсов и контроле производительности бизнеса. Служба отчетов SQL Azure распространяет эти возможности на размещенные в SQL Azure данные.

### Собственное решение для синхронизации или использование решения стороннего разработчика

Также можно использовать любой существующий пакет или разработать собственное решение для анализа данных и формирования отчетов для гибридных приложений исходя из того, что выбранное решение может получить доступ к данным, хранящимся в базе данных приложения или репозитории. Некоторым организациям может потребоваться собственное решение, которое интегрируется с другими приложениями или службами, или использовать существующее собственное решение или решение стороннего поставщика для создания аналитических бизнес-отчетов.

Собственное решение или решение стороннего поставщика можно точно настроить в соответствии с требованиями организации к отчетам, и такая сосредоточенность на конкретных областях интересов может обеспечить ускоренное формирование отчетов и эффективную общую производительность по сравнению с более общим решением. Разработанные под заказ решения сторонних поставщиков могут лучше подходить для специальных типов приложений и могут стоить меньше, чем более общее решение.

Использование существующего решения для формирования отчетов может снизить расходы на перенос приложения в облако. Однако в решении для формирования отчетов должна быть предусмотрена возможность подключения к облачному источнику данных без ущерба безопасности данных; это может потребовать от разработчиков создания дополнительных служб для предоставления данных. Кроме того, когда источник данных удален от приложения, которое использует отчет, процесс взаимодействия между ними может приводить к существенному сетевому трафику. И наконец, существующее решение или решение стороннего поставщика может не предоставлять всех необходимых форматов или такой же функциональности по сравнению со службами отчетов SQL Server и SQL Azure.

## Как компания Trey Research выбрала решение для формирования отчетов

Когда компания Trey Research переместила приложение Orders в облако, проектировщики решили разместить данные, генерируемые приложением при размещении заказа, в SQL Azure. До этого перемещения в компании Trey Research использовались службы отчетов SQL Server для получения бизнес-информации из базы данных заказов.

Проектировщики в Trey Research решили, что возможным решением при удаленном размещении исходных данных может быть загрузка всех данных о заказах в локальные базы данных и использование служб отчетов SQL Server для их анализа. Однако, если синхронизация данных не производится на регулярной основе, что требуют дополнительных расходов, операция передачи данных приведет к большему времени задержки при формировании отчетов. Этот подход может вызвать значительный сетевой трафик в Интернете, поскольку данные будут многократно запрашиваться при построении отчетов.

Вместо этого после переноса данных в SQL Azure, имеет смысл использовать возможности бизнес-аналитики службы формирования отчетов SQL Azure. Этот подход позволяет минимизировать сетевой трафик в Интернете, обеспечивает отображение в отчете последних данных без дополнительных задержек и позволяет предоставлять информацию отчетов в различных форматах.

## Как в Trey Research используют службу отчетов SQL Azure

Локальная служба формирования отчетов использует службу отчетов SQL Azure для формирования отчетов, которые используются локальными приложениями управления. Компания Trey Research повысила полезность отчетов, комбинируя первичные данные из службы отчетов SQL Azure с потоками данных, которые предоставляет Windows Azure Marketplace.

На рисунке 4 показана реализованная компанией Trey Research архитектура.

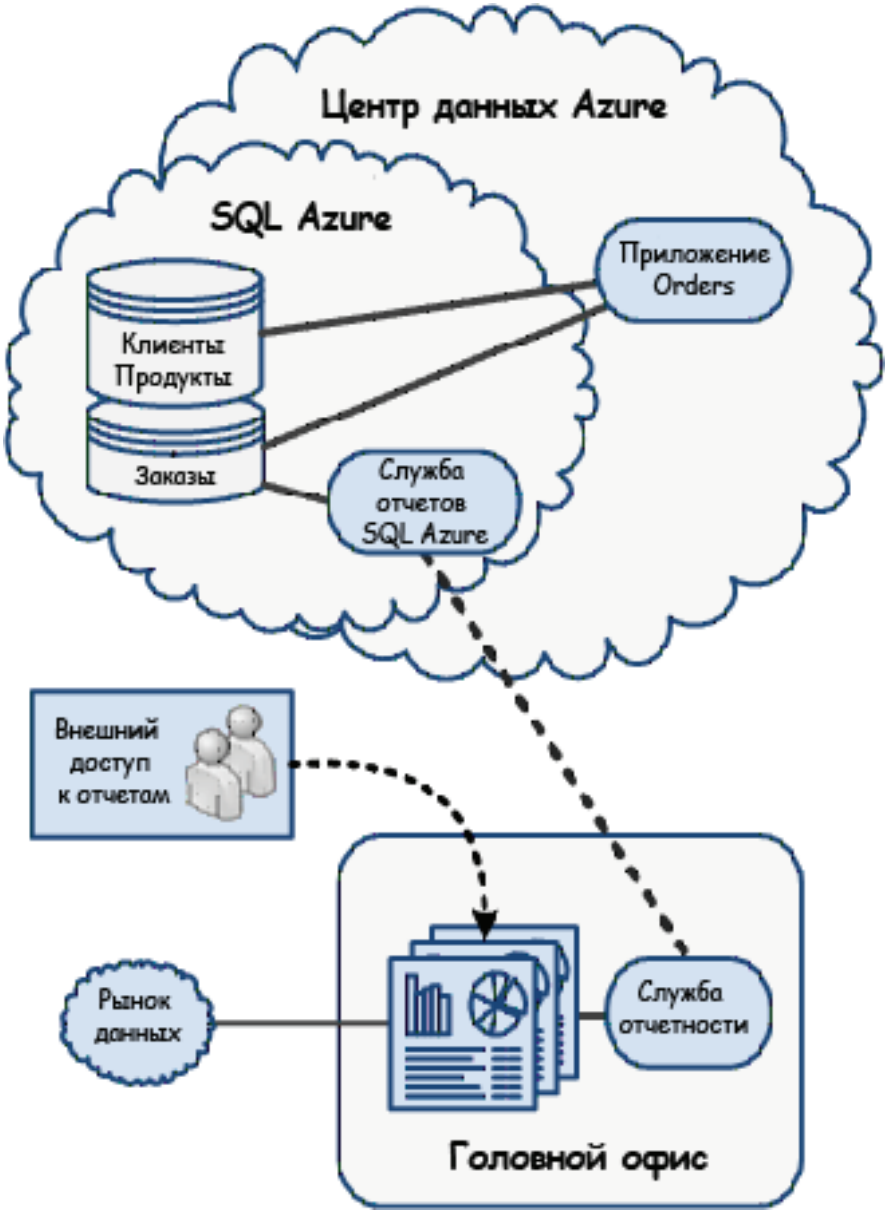



**Рисунок 4**

**Создание аналитических бизнес-отчетов с помощью службы отчетов SQL Azure**

**Примечание**

Для простоты установки пример приложения в этом руководстве не включает реализацию службы отчетов SQL Azure. Для получения более подробной информации о службе отчетов SQL Azure см. статью «Бизнес-аналитика» на сайте http://www.windowsazure.com/en-us/home/tour/business-analytics/. Для получения более подробной информации об использовании внешних данных в отчетах см. статью «Как получить данные и приложения премиум-класса в одном месте» по адресу http://datamarket.azure.com/.

## Как компания Trey Research предоставляет данные отчетов внешним партнерам

В компании Trey Research также решили предоставить данные отчетов конкретным пользователям, которые имеют доступ к локальной службе отчетов по Интернету с помощью службы Service Bus Relay.

Компания Trey Research решила разработать пробную версию этой функциональности, пока будет изучаться вопрос, какие типы информации можно предоставить внешним партнерам без угрозы коммерческой конфиденциальности бизнеса. Поэтому в этой первой версии компания Trey Research просто отобразила общую стоимость проданной продукции, разбитую по отчетным кварталам и регионам, как это определено в сервисном контракте в файле IOrdersStatistics.cs, который находится в папке Services проекта **HeadOffice**:

**C#**

```
[ServiceContract]
public interface IOrdersStatistics
{
  [OperationContract]
  double SalesByQuarter(int quarter);

  [OperationContract]
  double SalesByRegion(string region);
}
```

Последующие версии этой службы могут предоставлять более детальную разбивку данных по продажам.

Приложение HeadOffice, которое работает локально в решении Trey Research, размещает службу WCF, которая называется **OrderStatistics** и реализует операции **SalesByQuarter** и **SalesByRegion**. Эти операции позволяют просто извлечь необходимые данные из основной базы данных и передать их назад в качестве возвращаемого значения. Реализация этой службы приведена в файле OrdersStatistics.cs в папке Services проекта **HeadOffice**.

Вызывающий эту службу код находится в методе **OpenServiceHost** в файле Global.asax.cs также в проекте **HeadOffice**.

> **Примечание**
>
> Технология, применяемая в примере приложения для запуска службы **OrderStatistics**, может не подходить для всех веб-приложений и работает только в примере приложения, поскольку пользователь должен явно запустить веб-приложение, в котором содержится служба. В других случаях может быть предпочтительней использовать функцию **автозапуска Auto-Start** Windows Server AppFabric для запуска службы. Для получения дополнительной информации см. статью «Функция автозапуска» по адресу http://msdn.microsoft.com/en-us/library/ee677260.aspx.

Эта служба доступна для внешних партнеров и пользователей через службу Windows Azure Service Bus Relay с использованием пути службы **Services/RelayedOrdersStatistics**, и приложение публикует службу через конечную точку TCP с использованием привязки **netTcpRelayBinding**. Подключение к шине интеграции безопасно благодаря ACS с использованием маркера аутентификации. В главе 3 «Аутентификация пользователей в приложении Orders» приводится более подробная информация о настройке ACS, а в разделе «Рекомендации по безопасности службы Windows Azure Service Bus Relay» в «Приложении B. Реализация подхода "коммуникации без границ"» приводится руководство по аутентификации и авторизации партнеров при подключении к службе Service Bus Relay. Дополнительная информация о пространстве имен шины интеграции, пути службы, учетной записи для безопасного доступа и настройке службы хранится в файле web.config проекта. Обратите внимание, что веб-приложение подключается к шине

интеграции с помощью удостоверения с именем **headoffice**; это удостоверение предоставляет права, ассоциированные с утверждением **Listen**, что позволяет приложению получать входные запросы, но не разрешает выполнять другие операции, например отправку запросов.

**XML**

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  ...
  <appSettings>
    ...
    <!-- ServiceBus config-->
    <add key="serviceBusNamespace"
      value="treyresearchscenario" />
    <add key="UriScheme" value="sb" />
    <add key="RelayServicePath"
      value="Services/RelayedOrdersStatistics" />
    ...
  </appSettings>
  ...
  <system.serviceModel>
    <services>
      <service
name="HeadOffice.Services.OrdersStatistics">
        <endpoint behaviorConfiguration=
            "SharedSecretBehavior"
          binding="netTcpRelayBinding"
          contract=
            "HeadOffice.Services.IOrdersStatistics"
          name="RelayEndpoint"/>
      </service>
    </services>
    <behaviors>
      <endpointBehaviors>
        <behavior name="SharedSecretBehavior">
          <transportClientEndpointBehavior
             credentialType="SharedSecret">
            <clientCredentials>
              <sharedSecret issuerName="headoffice"
                issuerSecret="<data omitted>" />
            </clientCredentials>
          </transportClientEndpointBehavior>
          <serviceRegistrySettings
            discoveryMode="Public"
            displayName=
              "RelayedOrdersStatistics_Service"/>
        </behavior>
      </endpointBehaviors>
    </behaviors>
    ...
  </system.serviceModel>
</configuration>
```

Компания Trey Research также разработала простое приложение, работающее из командной строки, для проверки подключения к службе **OrderStatistics** через службу Windows Azure Service Bus Relay и проверки ожидаемой работоспособности операций **SalesByQuarter** и **SalesByRegion**. Это приложение находится в проекте **ExternalDataAnalyzer**. Это клиент WCF, который обеспечивает подключение к службе при помощи программного интерфейса шины интеграции Windows Azure SDK совместно с программным интерфейсом Service Model WCF. Безопасность подключения к шине интеграции обеспечивается соответствующим маркером аутентификации. Определение конечной точки для подключения к службе и удостоверения для безопасного доступа находятся в файле app.config. Как и веб-приложение, проект **ExternalDataAnalyzer** подключается к шине интеграции с использованием особого идентификатора, **externaldataanalyzer**, которому предоставляются права, ассоциированные с утверждением **Send**, позволяя ему отсылать запросы в службу, но не позволяя производить другие действия, например получать запросы от других клиентов.

**XML**

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  ...
  <system.serviceModel>
    ...
    <behaviors>
      <endpointBehaviors>
        <behavior name="SharedSecretBehavior">
          <transportClientEndpointBehavior
             credentialType="SharedSecret">
            <clientCredentials>
              <sharedSecret issuerName=
                "externaldataanalyzer"
                issuerSecret="<data omitted>" />
            </clientCredentials>
          </transportClientEndpointBehavior>
        </behavior>
      </endpointBehaviors>
    </behaviors>
    ...
  </system.serviceModel>
</configuration>
```

На рисунке 5 приведена общая структура и детали реализации службы **OrderStatistics** и клиентского приложения **ExternalDataAnalyzer**.

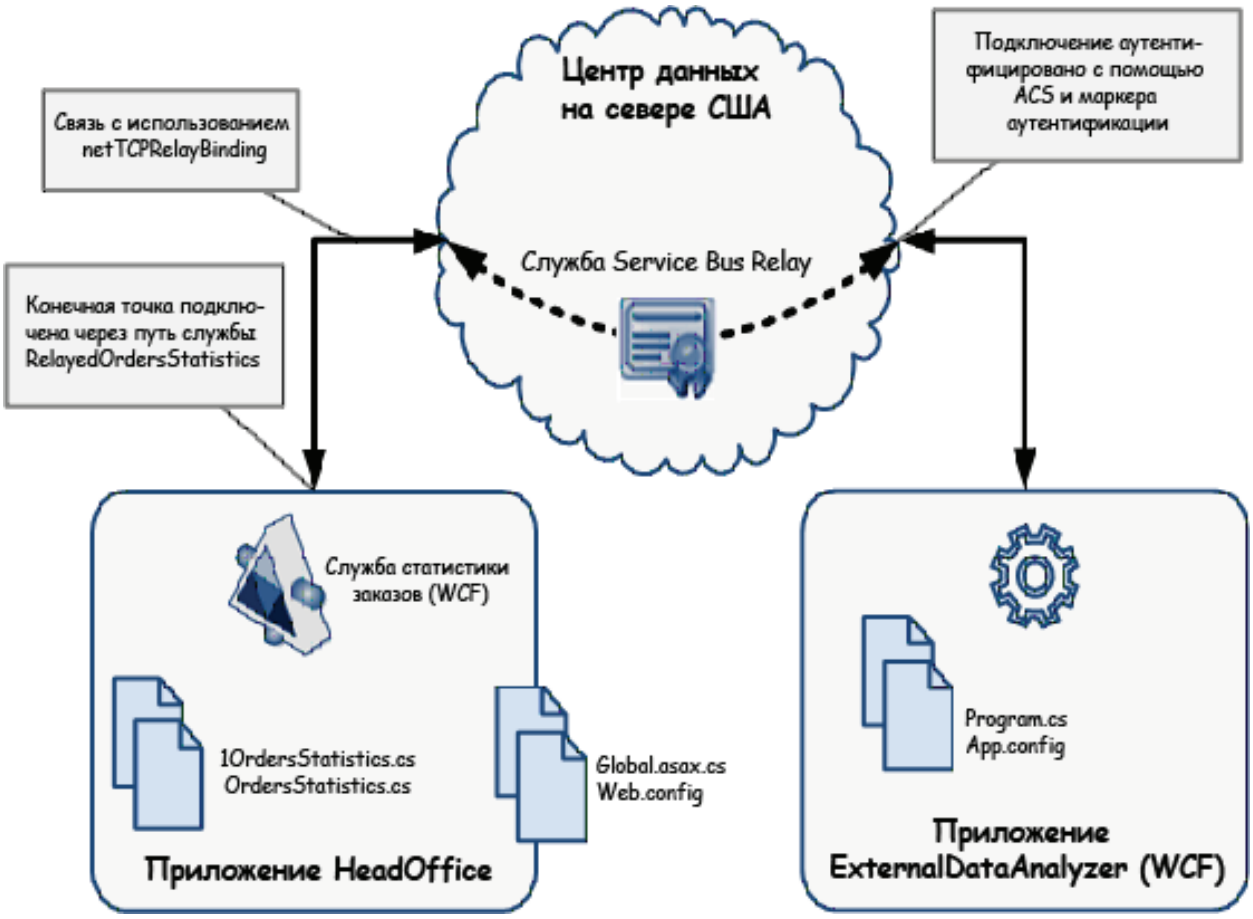



**Рисунок 5**

**Структура службы OrderStatistics и клиентского приложения ExternalDataAnalyzer**

## Резюме

Эта глава посвящена развертыванию сценариев, относящихся к построению приложений, некоторые части которых работают локально, другие — в облаке, а какие-то части разработаны для внешних партнеров или самими партнерами. Эта глава посвящена трудностям развертывания, с которыми столкнулись в компании Trey Research, например размещение данных локально или в облаке, синхронизация данных между различными размещениями, которые входят в гибридное решение, и формирование аналитических бизнес-отчетов.

Поскольку в компании Trey Research использовался SQL Server в первоначальной версии приложения Orders, развертывание данных в SQL Azure и использование службы SQL Azure Data Sync для поддержания соответствия и репликации основных данных — это простой и естественный процесс переноса приложения в облако.

И наконец, при выборе SQL Azure для хранения данных в облаке, служба отчетов SQL Azure — это очевидный выбор для реализации решений бизнес-аналитики и формирования отчетов для Trey Research, а служба Service Bus Relay обеспечивает идеальный механизм для публикации данных отчетов для компаний-партнеров.

## Дополнительная информация

- Обзор возможностей Windows Azure на сайте http://www.windowsazure.com/en-us/home/tour/overview/
- Центр разработчика Windows Azure «Windows Azure Developer Center» по адресу http://www.windowsazure.com/en-us/develop/overview/
- Обзор синхронизации данных SQL Azure «SQL Azure Data Sync Overview» на сайте http://social.technet.microsoft.com/wiki/contents/articles/sql-azure-data-sync-overview.aspx
- Часто задаваемы вопросы по синхронизации данных SQL Azure «SQL Azure Data Sync FAQ» по адресу http://social.technet.microsoft.com/wiki/contents/articles/sql-azure-data-sync-faq.aspx
- Служба SQL Azure Data Sync — синхронизация данных между локальным размещением и облаком «SQL Azure Data Sync — Synchronize Data across On-Premise and Cloud (E2C)» по адресу http://channel9.msdn.com/Series/SQL-Azure-Data-Sync/SQL-Azure-Data-Sync-Synchronize-Data-across-On-Premise-and-Cloud-E2C
- *Синхронизация SQL Server с SQL Azure при помощи Sync Framework 2.1* «SQL Server to SQL Azure Synchronization using Sync Framework 2.1» по адресу http://blogs.msdn.com/b/sync/archive/2010/08/31/sql-server-to-sql-azure-synchronization-using-sync-framework-2-1.aspx
- Бизнес-аналитика для формирования отчетов в SQL Azure «Business Analytics» по адресу http://www.windowsazure.com/en-us/home/tour/business-analytics/
- Как получить данные и приложения премиум-класса в одном месте «One-Stop Shop for Premium Data and Applications» на сайте http://datamarket.azure.com/
- Начальные сведения о службе Service Bus Relay «Windows Azure AppFabric: An Introduction to Service Bus Relay» по адресу http://www.microsoft.com/en-us/showcase/details.aspx?uuid=395930db-6622-4a9f-8152-e0cb1fc5149c

# Глава 3. Аутентификация пользователей в приложении Orders

В этой главе рассматривается, как компания Trey Research адаптировала аутентификацию и авторизацию пользователей в приложении Orders после перенесения части приложения в облако.

Изначально локальное приложение Orders использовало формы ASP.NET для аутентификации пользователей. Все посетители сайта должны были войти в систему, используя учетные данные, предоставленные сотрудниками компании Trey Research. Эти учетные записи хранились в базе данных SQL Server.

Когда проектировщики компании Trey Research рассматривали возможность адаптации приложения для работы в облаке, они решили воспользоваться преимуществами службы Windows Azure Access Control (ACS) для внедрения более гибкого и простого в использовании процесса аутентификации пользователей сайта на основе утверждений, выданных доверенными поставщиками удостоверений.

## Сценарий и контекст

Для большинства корпоративных приложений и веб-сайтов необходима безопасная и надежная аутентификация пользователей. Пользователи подтверждают свою личность, предоставляя известную только им и механизму аутентификации информацию. Обычно это комбинация имени пользователя и пароля, но также может использоваться смарт-карта, техника биометрической проверки, например отпечатки пальцев или сканирование сетчатки, а также любой другой подход. Важным фактором является то, что проектировщики приложения должны выбрать механизм уникальной идентификации каждого пользователя. Даже если эта идентификация состоит только в создании уникального идентификатора пользователя.

Формы ASP.NET — это естественный подход к аутентификации пользователей и авторизации всех их действий при доступе на веб-сайты, созданные с использованием технологии ASP.NET. Это гибкий и простой в реализации механизм, который предоставляет пользователям возможность управления своими учетными данными (например, изменение пароля). Именно поэтому проектировщики компании Trey Research изначально использовали формы ASP.NET для аутентификации в локальной версии приложения Orders.

> **Мнение Джаны**
>
> Аутентификация с использованием форм — это естественный подход для аутентификации в приложениях на основе технологии ASP.NET, но он не предоставляет гибкости подходов, основанных на утверждениях для федеративной идентификации и единого входа.

Однако аутентификация с использованием форм ASP.NET не соответствует ожиданиям пользователей, которые привыкли к механизмам аутентификации, предоставляющим возможность самостоятельного управления личными данными и использования одной учетной записи для доступа к нескольким веб-сайтам. Например, пользователи, имеющие учетную запись Windows Live, могут настроить и управлять ею в Windows Live. Эта же учетная запись используется для доступа к веб-сайтам, таким как веб-служба электронной почты Hotmail, программа разработчиков MSDN и другие веб-сайты Microsoft и сторонних разработчиков.

В этом случае Windows Live действует как поставщик удостоверений, аутентифицируя посетителей и подтверждая их личность веб-сайтам, доверяющим Windows Live. Этот механизм, как правило, называют *федеративной аутентификацией.* Он также поддерживает возможность единого входа и не требует повторного ввода учетных данных для доступа к другим веб-сайтам.

**Примечание**

Для получения дополнительной информации о федеративной и основанной на утверждениях аутентификации пользователей, поставщиках удостоверений, и единого входа в систему см. «Приложение Б. Аутентификация пользователей и авторизация запросов» в этом руководстве.

Проектировщики компании Trey Research хотели предоставить своим посетителям гибкость использования федеративной аутентификации пользователей и решили изучить возможные варианты ее внедрения в приложение Orders. Кроме того, они хотели упростить доступ к приложению для сотрудников компании Trey Research и сотрудников компаний-партнеров (например, транспортных партнеров).

Рисунок 1 отображает существующую архитектуру аутентификации пользователей локального приложения Orders. В этой версии приложения аутентификация пользователей реализована с помощью форм ASP.NET (изображено, как **1** на схеме). Пользователям предоставляется доступ к приложению Orders если их веб-обозреватель предъявил правильный маркер аутентификации ASP.NET для этого приложения (**2** на схеме). Сотрудники компаний-партнеров, таких как транспортные партнеры, проходят процесс аутентификации точно также (**3** и **4** на схеме). Сотрудники головного офиса компании Trey Research для входа в приложение Orders используют специальные учетные данные, применяя тот же механизм аутентификации на основе форм ASP.NET, который используется пользователями и сотрудниками-партнерами (**5** и **6** на схеме). Однако сотрудники компании Trey Research также используют внутренние приложения, такие как программы для бухгалтерского учета и производственного процесса. Им необходимо пройти аутентификацию для этих приложений, используя другие учетные данные, хранимые локально в Windows Active Directory (**7** и **8** на схеме).

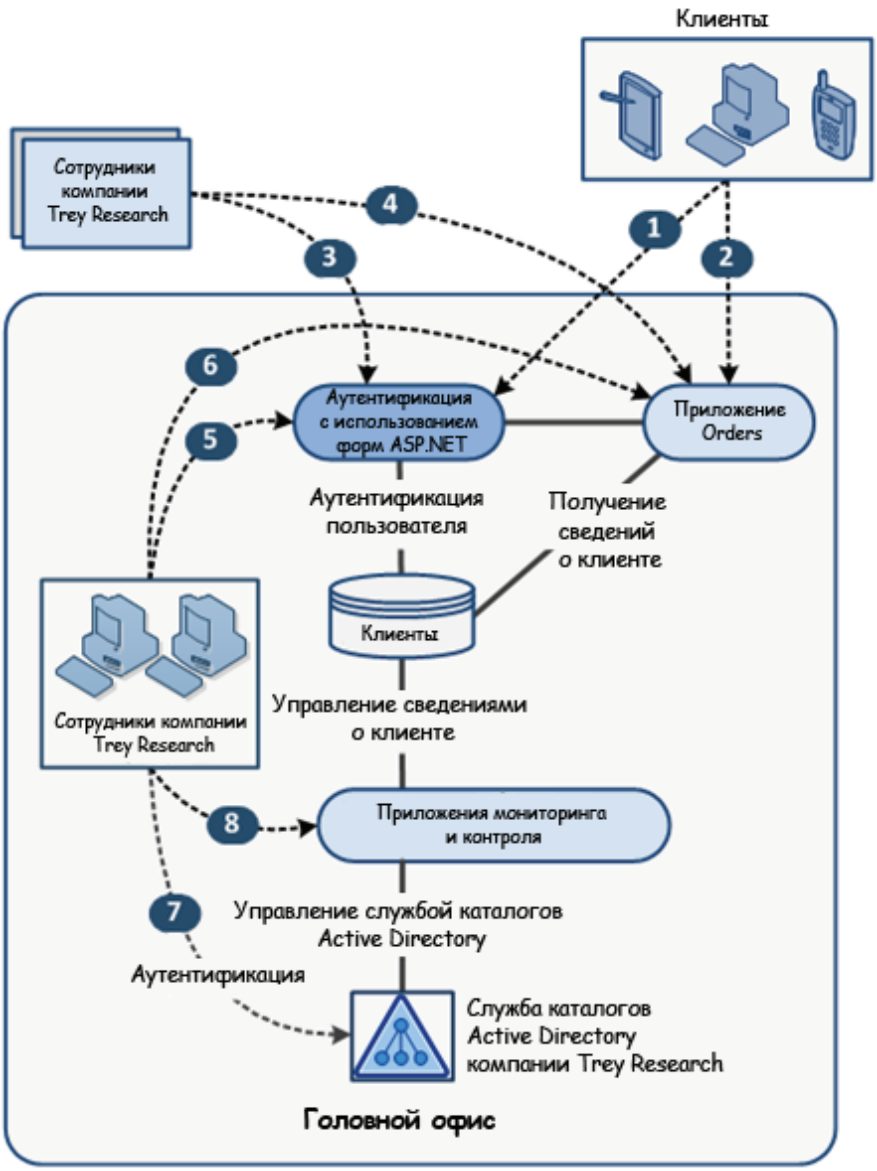



**Рисунок 1**

### Изначальный механизм локальной аутентификации пользователей в компании Trey Research

В следующих разделах данной главы описывается, как проектировщики компании Trey Research определили и выбрали более гибкий подход для процесса аутентификации и авторизации посетителей, партнеров и сотрудников компании Trey Research, и как они его реализовали в приложении Orders. В последнем разделе этой главы рассматривается, как ACS может действовать как механизм аутентификации пользователей для шины интеграции Windows Azure.

## Аутентификация посетителей в приложении Orders

Проектировщики компании Trey Research были осведомлены о расширяющемся использовании федеративного удостоверения в сети Интернет и хотели оценить возможные варианты аутентификации пользователя на современных веб-сайтах и в веб-приложениях. В следующих разделах этой главы вы узнаете, как они выполнили эту оценку.

## Выбор метода аутентификации

Проектировщики компании Trey Research оценили возможность использования общепринятого метода аутентификации пользователей в приложении Orders, сравнивая преимущества и недостатки использования аутентификации, основанной на утверждениях, с аутентификацией на основе форм ASP.NET в сочетании с существующим пользовательским хранилищем, реализованным службой Windows Active Directory.

### Аутентификация с использованием форм ASP.NET

Аутентификация с использованием форм является встроенной функцией ASP.NET и может быть быстро и легко реализована на веб-сайтах на базе ASP.NET. Существует ряд серверных элементов управления, которые могут быть использованы в веб-страницах, и пользователям будут предоставлены возможность изменения их учетных данных, создания учетной записи, а также возможности входа и выхода из системы. Аутентификация с использованием форм уже использовалась в приложении Orders. Таким образом, использование этого механизма в гибридной версии приложения не повлечет за собой изменения кода программы.

Процесс аутентификации пользователей с использованием форм ASP.NET требует доступной для веб-сайта базы данных с учетными данными пользователей. Это не будет проблематичным, поскольку уже принято решение о развертывании информации о клиентах на технологической платформе SQL Azure.

Впрочем, компания Trey Research решила предложить более гибкие возможности посетителям, сотрудникам компании и партнерам, предоставив доступ к веб-сайту Orders с использованием их существующих учетных записей вместо создания новых. А использование форм ASP.NET для аутентификации предусматривает предоставление пользователям отдельных учетных записей для приложения Orders.

### Аутентификация, основанная на утверждениях, с использованием службы федерации Microsoft Active Directory

Аутентификация на основе утверждений позволяет посетителям использовать существующую учетную запись, которую они создали с помощью поставщика удостоверений, которому доверяет компания Trey Research. Для использования этого подхода необходимо, чтобы разработчики компании Trey Research решили, каким поставщикам удостоверений они будут доверять. Один из вариантов — это использование службы федерации Active Directory Federation Service (ADFS) в сочетании с существующей службой Windows Active Directory. Службы ADFS предоставляют пользователям возможность входа в систему через Интернет, используя учетные записи из Active Directory. Нет необходимости прямого подключения к доменной сети Active Directory.

> **Мнение Джаны**
>
> Службы ADFS работают подобно службам маркеров безопасности (STS), которые могут проводить аутентификацию в Windows Active Directory и выдавать маркеры безопасности, содержащие утверждения об этом пользователе. Эти утверждения могут содержать как просто идентификатор для аутентификации пользователя, так и расширенный набор утверждений в маркере, который предоставляет более подробную информацию о пользователях, такую как имена, роли, адреса электронной почты и прочее.

Однако использование службы ADFS означает, что сотрудникам компании Trey Research необходимо создать учетные записи для всех посетителей в их домене Windows Active Directory, или в отдельном домене Windows Active Directory, предназначенном для клиентов, что вызовет увеличение требований к администрированию и может привести к нежелательным последствиям с точки зрения безопасности. Приемлемым подходом была бы реализация аутентификации для определенного небольшого и четко ограниченного подмножества пользователей — сотрудников компании Trey Research и компаний-партнеров. Но такая аутентификация всех посетителей не является разумным подходом.

### Аутентификация на основе утверждений службы Windows Azure Access Control

Компания Trey Research может расширить возможности аутентификации пользователей с помощью службы ACS. Она предоставляет пользователям механизм входа в систему, используя существующие учетные записи некоторых популярных социальных сетей и поставщиков удостоверений.

> **Мнение Джаны**
>
> Служба ACS также является службой STS, которая предоставляет маркеры безопасности, содержащие утверждения о пользователе. Также она может аутентифицировать пользователей у ряда поставщиков удостоверений социальных сетей, таких как Windows Live ID, Google, Facebook и OpenID. Служба ACS может быть и поставщиком утверждений; эта функциональность используется для аутентификации доступа в очереди и топик шины интеграции Windows Azure.

Проектировщики компании Trey Research решили, что служба ACS позволит предоставить пользователям необходимую гибкость использования существующих удостоверений и учетных записей, а также более удобный процесс аутентификации пользователя. Также одним из преимуществ является то, что компании более не придется управлять учетными записями пользователей, например предоставляя возможность смены пароля. За все функции управления удостоверениями отвечает поставщик удостоверений. Компании Trey Research необходимо только определить, каким поставщикам утверждений они будут доверять для установления личности пользователей.

Однако есть некоторые проблемы, которые должны решить в компании Trey Research при использовании этого подхода. Поставщики удостоверений социальных сетей обычно возвращают только утверждения, включающие уникальный идентификатор пользователя, и, возможно, имя пользователя и адрес электронной почты. Таким образом код программы должен ассоциировать этот идентификатор с данными зарегистрированного пользователя, хранящимися в базе данных в таблице Customer. Кроме того, компании, возможно, потребуется определить способ переноса существующих пользовательских учетных записей для применения основанного на утверждениях подхода. Внедрение аутентификации на основе утверждений также повлечет за собой необходимость изменения кода приложения разработчиками компании Trey Research для использования утверждений, возвращенных службой ACS.

Специалисты компании Trey Research также понимают, что использование службы ACS с поставщиками социальных утверждений не предоставляет решения для единого входа в систему сотрудников компании и компаний-партнеров. Как те, так и другие сотрудники подключаются к приложению из собственных корпоративных сетей, используя учетные записи, определенные в службе Active Directory их компаний (или других корпоративных службах управления учетными записями). Однако им будет необходимо войти в приложение Orders, используя учетную запись, определенную поставщиками социальных утверждений.

### Аутентификация на основе утверждений служб ACS и ADFS

Проектировщики компании Trey Research понимали, что они могут решить проблемы с помощью подходов, приведенных в предыдущих разделах, комбинируя техники служб ACS и ADFS для аутентификации пользователя. Совместно эти службы могут действовать как поставщики удостоверений и маркеров (STS) для учетных записей, определенных поставщиками удостоверений социальных сетей, таких как Windows Live ID, Google, Facebook и OpenID, а также определенных в Windows Active Directory.

Для этого необходимо, чтобы служба ACS была настроена на доверие экземплярам служб ADFS, которые предоставлены компанией Trey Research и компаниями-партнерами, желающими участвовать в аутентификации пользователей с помощью федеративных удостоверений. Пользователи направляются в службу ACS для аутентификации. Далее служба отправляет запрос на аутентификацию каждого пользователя соответствующему поставщику удостоверений. Это может быть как один из настроенных поставщиков удостоверений социальных сетей, так и один из настроенных экземпляров служб ADFS. Когда экземпляр этой

службы получает запрос на аутентификацию пользователя, он проверяет удостоверения в локальном каталоге Windows Active Directory. После подтверждения удостоверения одним из поставщиков удостоверений или экземпляром службы ADFS, служба ACS возвращает приложению маркер, содержащий любые найденные утверждения.

Используя службы ACS и ADFS совместно, компания Trey Research предоставляет посетителям доступ к приложению Orders с использованием социального удостоверения либо удостоверения, определенного в одном из корпоративных доменов Active Directory. Пользователи могут использовать собственные учетные записи Windows Live ID, Google, Facebook, OpenID и другие. А сотрудникам компании предоставляется возможность входа, используя собственные корпоративные учетные данные, определенные в домене компании. Сотрудники компаний-партнеров осуществляют вход с использованием корпоративных учетных записей, определенных в их доменах.

Это означает, что у компании Trey Research более нет необходимости хранить данные для аутентификации посетителей и сотрудников кампаний-партнеров. А учетные записи собственных сотрудников компании уже управляются сетевыми администраторами в Active Directory. Следовательно, нет необходимости в дополнительном управлении учетными записями сотрудников.

**Мнение Бхарата**

Совместное использование аутентификации с использованием удостоверений социальных сетей и механизма каталогов организации со службой ACS обеспечивает значительную гибкость и позволяет приложениям поддерживать аутентификацию широкого круга пользователей. Также можно использовать утверждения на основе ролей для авторизации действий пользователей, поскольку из каталогов организации, как правило, можно получить информацию о ролях, хотя она обычно не предоставляется поставщиками удостоверений социальных сетей.

Единственными реальными проблемами являются дополнительные сложности настройки служб ACS и ADFS для поддержки такого федеративного подхода, а также потеря полноты контроля над списком разрешенных пользователей. Например, администраторы компаний-партнеров будут управлять списком своих сотрудников, которым предоставлен доступ в приложение Orders. Хотя компания Trey Research освобождается от работы над данной задачей и ответственности за нее, она также теряет возможность контроля над пользователями, имеющими доступ, и должна доверять компаниям-партнерам в предоставлении соответствующих прав. Это может стать проблемой, если предоставленные пользователям права доступа зависят от ролей, обозначенных в утверждениях, которые возвращаются поставщиками удостоверений, например «Менеджер» или «Администратор».

### Совместное использование форм и аутентификации на основе утверждений

Последним из рассмотренных компанией Trey Research подходов было комбинирование подхода на основе утверждений, который рассматривался ранее, и аутентификации на основании форм ASP.NET, чтобы пользователи могли продолжать использовать собственные учетные записи из форм. Это означает, что администраторам этой компании нет необходимости переносить существующих пользователей для использования подхода, основанного на утверждениях. И новые, и существующие пользователи смогут использовать приложение. Новым пользователям будут предоставлены основанные на утверждениях учетные записи. А существующим пользователям будет предложено изменить свои учетные данные для аутентификации, основанной на утверждениях.

Тем не менее для разработчиков и администраторов компании Trey Research это означало бы необходимость управления и обслуживания двух несовместимых систем. Это также усложняет выполнение общих задач, таких как составление списка клиентов или управление данными клиентов в базе данных. Кроме того, разработчикам было бы необходимо изменить код приложения для поддержки обеих технологий аутентификации пользователей.

## Как компания Trey Research выбрала метод аутентификации пользователей

После рассмотрения всех вариантов компания Trey Research решила использовать федеративный подход к аутентификации на основе утверждений, комбинируя службы ACS и ADFS. Этот подход предоставляет максимальную гибкость новым клиентам и поддерживает единый вход в систему для всех посетителей, включая сотрудников компании Trey Research и компаний-партнеров.

Было решено не выполнять дополнительную работу и не тратить средства на поддержку аутентификации пользователей на основе форм, а поэтому компании Trey Research необходимо было определиться с подходом к управлению учетными записями существующих клиентов и сотрудников. Разработчики обнаружили возможность добавления кода в приложение, который позволит пользователям, осуществляющим вход с использованием социальных или федеративных удостоверений, подключаться к существующей учетной записи. Это избавит компанию Trey Research от переноса существующих учетных записей, использующих аутентификацию на основе форм.

Кроме того, это обеспечило дополнительные преимущества в том, что клиенты и сотрудники получили возможность использования нескольких удостоверений, связанных с зарегистрированной учетной записью в таблице Customer. Таким образом, доступ к приложению осуществляется с помощью любого из этих поставщиков удостоверений. Это позволяет улучшить возможности единого входа посетителям. Данная реализация описана более детально в разделе «Управление множественными идентификаторами пользователя» этой главы.

## Как компания Trey Research использует службы ACS и ADFS для аутентификации посетителей

Компания Trey Research осуществляет основанную на утверждениях аутентификацию посетителей веб-сайта Orders посредством службы ACS в качестве STS, трех поставщиков удостоверений (Windows Live ID, Yahoo! и Google) и экземпляров служб ADFS, предоставленных как компанией Trey Research, так и основными компаниями-партнерами. На рисунке 2 показана схема всего цикла аутентификации разных типов пользователей для разных поставщиков удостоверений.

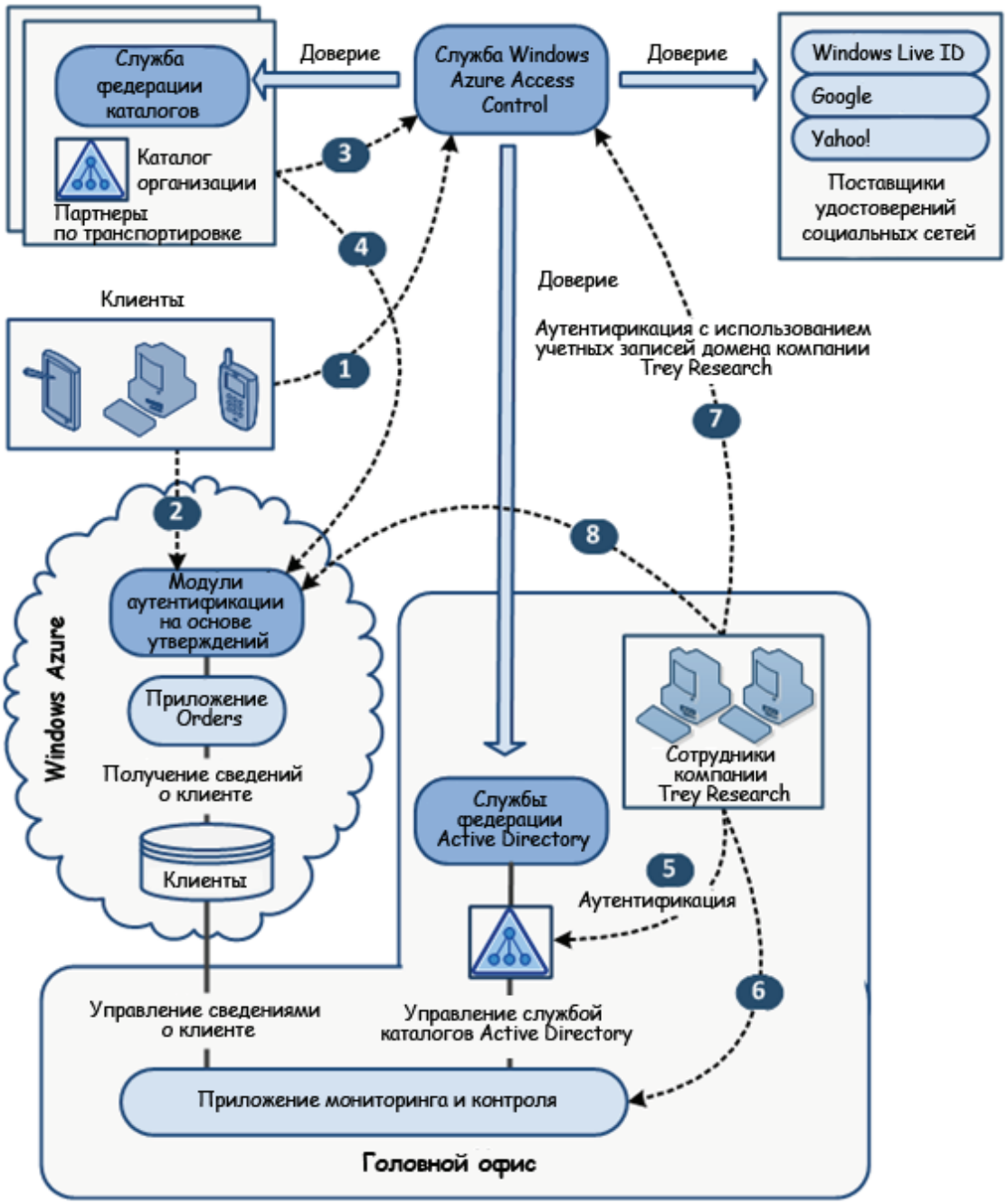



**Рисунок 2**

**Архитектура и последовательность выполнения аутентификации в гибридном приложении Orders**

В рамках данной архитектуры клиенты проходят аутентификацию посредством службы ACS с использованием выбранного ими поставщика утверждений социальных сетей (**1** на схеме). Эти поставщики удостоверений настроены в службе ACS и являются для нее доверенными. Модули приложения Orders для аутентификации пользователей на основе утверждений выполняют проверку меток, возвращенных службой ACS, которые клиенты посылают приложению Orders (**2**на схеме). Эти модули извлекают утверждения идентификатора пользователя, затем приложение Orders может производить поиск соответствующих данных клиентов в таблице Customer базы данных.

Аутентификация сотрудников компаний-партнеров осуществляется через службу ACS (**3** на схеме), но так как она настроена на доверие федеративным службам каталогов партнеров, эти сотрудники могут выполнять вход в приложение Orders (**4** на схеме), используя учетные записи их каталога организации.

И наконец, сотрудники компании Trey Research проходят аутентификацию средствами Active Directory своей корпоративной сети во время осуществления входа на личный компьютер (**5** на схеме). Таким образом они получают доступ к внутренним приложениям компании Trey Research (**6** на схеме). Когда эти сотрудники получают доступ в приложение Orders, служба ACS проверяет подлинность текущей учетной записи в службе Active Directory с помощью ADFS (**7**на схеме). После этого им выдается соответствующий маркер, который могут использовать модули аутентификации на основе утверждений приложения Orders для идентификации пользователей (**8** на схеме).

В следующих разделах этого руководства подробнее описывается процесс реализации этой архитектуры компанией Trey Research в приложении Orders.

### Настройка службы Access Control

Компания Trey Research использует программу установки (**TreyResearch.Setup**в примере проекта) для настройки службы ACS без использования веб-портала Windows Azure. Программа установки использует эти классы в проекте **ACS.ServiceManagementWrapper** для получения доступа и настройки службы ACS.

> **Примечание**
>
> Для получения дополнительной информации о проекте **ACS.ServiceManagementWrapper**, см. «Access Control Service Samples and Documentation» по адресу http://acs.codeplex.com/releases/view/57595.

В таблице показано, как компания Trey Research настроила службу ACS для аутентификации посетителей приложения Orders.

| Артефакт службы ACS | Параметр |
| --- | --- |
| Доверенный участник | Веб-сайт Orders |
| Поставщик удостоверений Windows Live ID | Группа правил по умолчанию, содержащая правила передачи идентификатора посетителя как **NameIdentifier и UserName** (Windows Live ID не раскрывает информацию об адресах электронной почты пользователей). |
| Поставщик удостоверений Google | Группа правил по умолчанию, содержащая правило для передачи идентификатора посетителя как **NameIdentifier и** правило для передачи пользовательского адреса электронной почты как **UserName**. |
| Поставщик удостоверений Yahoo! | Группа правил по умолчанию, которая содержит правило для передачи идентификатора посетителя как **NameIdentifier и** правило для передачи пользовательского адреса электронной почты как **UserName**. |
| Поставщики удостоверений службы ADFS компании Trey Research | Группа правил по умолчанию, которая содержит правило для передачи идентификатора посетителя как **NameIdentifier** , правило для передачи пользовательского адреса электронной почты как **UserName**, и правило для передачи групп учетной записи пользователя как утверждение **Roles**. |
| Поставщики удостоверений службы ADFS компаний-партнеров | Группа правил по умолчанию, которая содержит правило для передачи идентификатора посетителя как **NameIdentifier**, правило для передачи пользовательского адреса электронной почты как **UserName**, и правило для передачи групп учетной записи пользователя как утверждение **Roles**. |

### Управление множественными идентификаторами пользователя

Как правило, каждый зарегистрированный пользователь приложения Orders имеет единственный идентификатор пользователя, который связан с соответствующими данными о клиенте в базе данных. Этот сценарий хорошо работает в случае единого источника аутентификации пользователя, например при аутентификации на основе форм ASP.NET. Но когда аутентификация осуществляется в различных областях (например, различные поставщики удостоверений, каждый из которых проводит аутентификацию пользователей в определенной области, например live.com или google.com), один и тот же клиент может иметь несколько уникальных идентификаторов.

Эта ситуация также возникает при использовании нескольких экземпляров службы ACS. В настоящее время компания Trey Research использует единое пространство имен службы ACS, настроенное в центре данных на севере США. Впрочем, в будущем компания может настроить дополнительные пространства имен службы ACS в других центрах данных вследствие дополнительных размещений приложения Orders в других национальных или международных центрах данных. Каждый экземпляр службы ACS может для уменьшения угрозы конфиденциальности пользователей возвращать *разные* уникальные идентификаторы пользователя от каждого экземпляра для одного и того же клиента.

Поэтому приложение должно быть разработано для поддержки более одного значения утверждения идентификатора пользователя (полученного от поставщика удостоверений или службы STS) для каждого зарегистрированного клиента. Компания Trey Research для этого использует две таблицы в базе данных. Основная таблица Customer содержит строку для каждого зарегистрированного клиента с уникальным идентификатором клиента, уставленным отделом управления учетными записями компании Trey Research в качестве ключа. Дочерняя таблица ACSIdentity содержит строку для каждого уникального значения утверждения идентификатора, связанную с соответствующим клиентом по ключевому полю.

Важным преимуществом этого подхода является возможность переноса пользователей из одного механизма аутентификации в другой. Например, когда компания Trey Research перешла от аутентификации на основе форм ASP.NET к аутентификации на основе утверждений, разработчики предоставили механизм перехода для существующих пользователей. Когда клиент впервые проходит процесс аутентификации с использованием поставщика удостоверений социальных сетей или службы ADFS (когда уникальный идентификатор утверждений пользователя еще не хранится в таблице ACSIdentity), компания Trey Research может предоставить страницу, позволяющую пользователям осуществить аутентификацию на основе форм ASP.NET и таким образом связать новую учетную запись, основанную на утверждениях, с существующими данными пользователя.

> Пример приложения в данном руководстве не полностью реализует описанную архитектуру аутентификации. Это сделано для того, чтобы избежать настройки соответствующей службы ADFS и экземпляра службы Windows Active Directory. В примере приложения используется только один экземпляр службы ACS и три поставщика утверждений социальных сетей и не используется отдельная таблица ACSIdentity. Также не используются роли для авторизации действий пользователей. Для получения информации об использовании утверждений ролей для авторизации посетителей, см. «Federated Identity for Web Applications» в книге «A Guide to Claims-Based Identity and Access Control (2nd Edition)» по адресу http://msdn.microsoft.com/en-us/library/ff359110.aspx.

### Реализация аутентификации

На рисунке 3 показана в общем виде структура служб и классов, которые используются для аутентификации и авторизации в приложении Orders. Подробная информация о классах, показанных на этой схеме, будет приведена в следующем разделе этой главы.

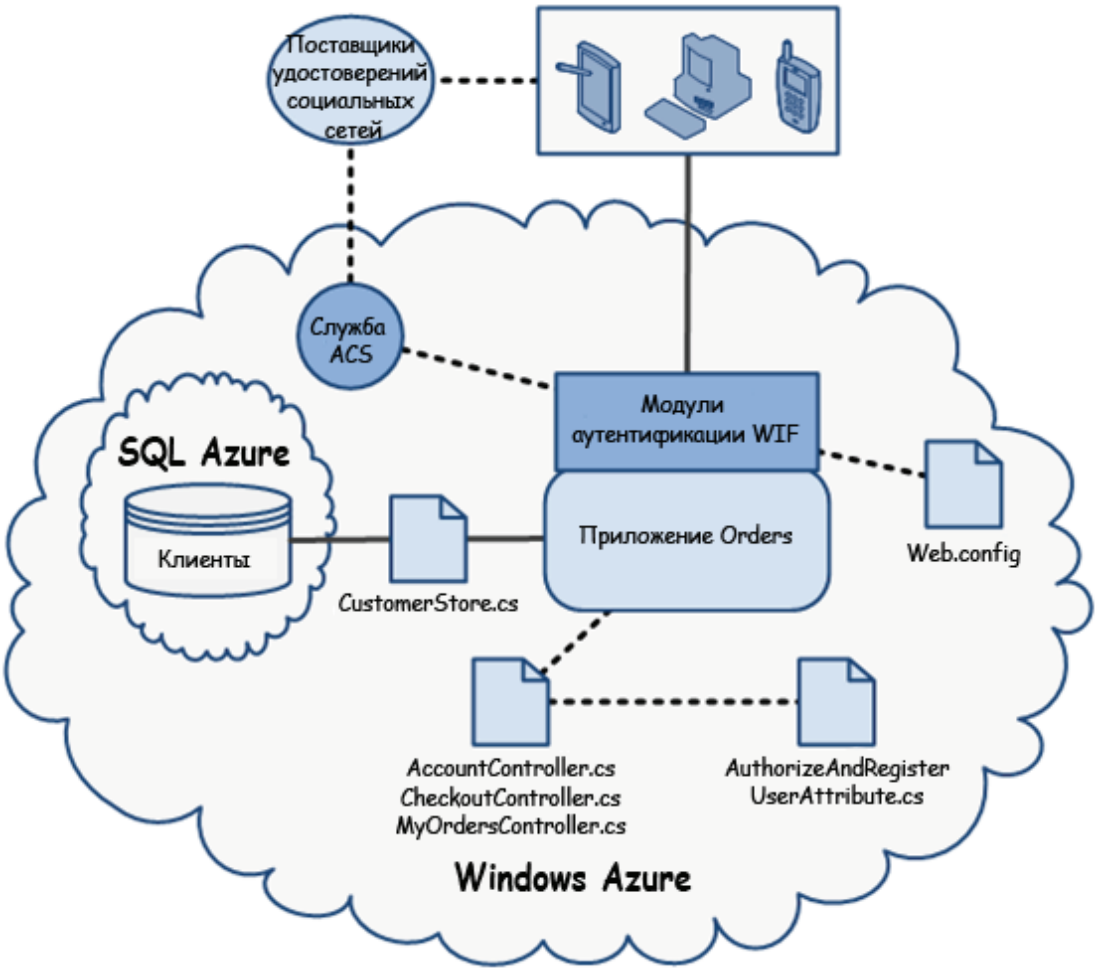



**Рисунок 3**

**Обзор процесса аутентификации и авторизации посетителей в гибридном приложении Orders**

### Аутентификация с использованием Windows Identity Foundation

Компания Trey Research использует Windows Identity Foundation (WIF) для проверки наличия допустимого маркера, содержащего утверждения о каждом посетителе, осуществляющем вход на веб-сайт. Нижеприведенный фрагмент взят из файла Web.config в проекте **Orders.Website** и показывает соответствующие настройки файла Web.config.

**XML**

```
<configSections>
<section name="microsoft.identityModel" type="..." />
...
</configSections>
...
<system.web>
  ...
<authorization>
```

```
<allow users="*" />
</authorization>
<authentication mode="Forms">
<forms loginUrl="~/Account/LogOn" timeout="2880" />
</authentication>
  ...
<httpModules>
<add name="WSFederationAuthenticationModule"
         type="..." />
<add name="SessionAuthenticationModule"
         type="..." />
</httpModules>
</system.web>
...
<microsoft.identityModel>
<service>
<audienceUris>
<add value="https://127.0.0.1" />
</audienceUris>
<federatedAuthentication>
<wsFederation passiveRedirectEnabled="true"
         issuer="https://treyresearch.accesscontrol.
                 windows.net/v2/wsfederation"
         realm="https://127.0.0.1" requireHttps="true" />
<cookieHandler requireSsl="true" />
</federatedAuthentication>
<serviceCertificate>
<certificateReference x509FindType="FindByThumbprint"
                            findValue="..." />
</serviceCertificate>
<applicationService>
<claimTypeRequired>
<claimType
           type="http://schemas.xmlsoap.org/ws/2005/05/
identity/claims/name" optional="true" />
</claimTypeRequired>
</applicationService>
<issuerNameRegistry
       type="Microsoft.IdentityModel.Tokens.
             ConfigurationBasedIssuerNameRegistry, ...">
<trustedIssuers>
<add thumbprint="..."
             name="https://treyresearch.accesscontrol.
                   windows.net/" />
</trustedIssuers>
</issuerNameRegistry>
<certificateValidation
      certificateValidationMode="None" />
</service>
</microsoft.identityModel>
```

Эти параметры добавляют два модуля WIF — **WSFederationAuthenticationModule** и **SessionAuthenticationModule** в конвейер HTTP, чтобы они выполнялись в каждом запросе. Настройки раздела **microsoft.identityModel** указывают, что эти модули будут перенаправлять запросы, нуждающиеся в аутентификации, по адресу https://treyresearch.accesscontrol.windows.net, что является пространством имен компании Trey Research, настроенными в службе ACS. Эта версия файла Web.config используется, когда приложение запущено на локальном компьютерном эмуляторе. Таким образом, аудитория URI и область — это локальный компьютер.

> **Мнение Маркуса**
>
> При использовании WIF для аутентификации посетителей вам необходимо изменить значения в файле Web.config для аудитории URI и области доверенного участника в случае, если вы изменили URL-адрес вашего приложения. Например, при развертывании приложения в облаке после тестирования в локальной вычислительной среде.

Раздел **applicationService** показывает, что веб-сайт Orders принимает утверждение **Name** в маркере, передаваемом модулям WIF, хотя это и не обязательно. Раздел **trustedIssuers** указывает, что приложение доверяет службам ACS, и специфицирует отпечаток сертификата, который модули WIF могут использовать для проверки маркера, отправленного посетителем.

Когда модули WIF обнаруживают запрос аутентификации от посетителя, сначала они проверяют подлинность маркера в запросе от доверенного поставщика. Если маркер не найден, эти модули перенаправляют запрос службе ACS. Если же допустимый маркер предоставлен, модули извлекают утверждения и делают их доступными коду приложения, так чтобы на их основе можно было авторизировать действия пользователей.

Класс с именем **IdentityExtensions** в проекте **Orders.Website.Helpers** содержит два метода, которые компания Trey Research использует для извлечения значений утверждений. Метод **GetFederatedUsername** извлекает значение утверждения **IdentityProvider** (имя первоначального поставщика утверждений для посетителя, например Windows Live ID или Google) и утверждение **Name**. Потом он их объединяет для создания федеративного имени пользователя. Метод **GetOriginalIssuer** просто возвращает имя первоначального поставщика утверждения для этого посетителя.

**C#**

```
private const string ClaimType =
    "http://schemas.microsoft.com/accesscontrolservice"
    + "/2010/07/claims/identityprovider";

public static string GetFederatedUsername(this
                    IClaimsIdentity identity)
{
  var originalIssuer = identity.Claims.Single(
               c => c.ClaimType == ClaimType).Value;
var userName = string.Format(
        CultureInfo.InvariantCulture, "{0}-{1}",
        originalIssuer, identity.Name);
  return userName;
}

public static string GetOriginalIssuer(this
                    IClaimsIdentity identity)
{
  return identity.Claims.Single(
               c => c.ClaimType == ClaimType).Value;
}
```

### Проверка запросов ASP.NET

По умолчанию ASP.NET проверяет наличие небезопасного содержимого во всех значениях, предоставленных в запросе. Среди такого содержимого — элементы HTML и XML. Компания Trey Research использует собственный класс, разрешающий размещение маркеров безопасности в формате XML, но защищает веб-сайт от других небезопасных входных данных. Это лучший вариант, чем выключение проверки запросов ASP.NET.

> **Мнение По**
>
> Лучше не выключать стандартный механизм проверки запросов ASP.NET. Вместо этого используйте собственную проверку запросов, позволяющую обрабатываемым запросам содержать документы XML (с утверждениями пользователей).

Собственный класс назван **WsFederationRequestValidator** и определен в проекте **Orders.Website**. Нижеприведенный код проверяет, содержит ли запрос отправки формы результат запроса WS Federation **SignInResponseMessage**. Если да, то код разрешает выполнение запроса, возвращая **true**. Если нет, код предоставляет возможность проверки содержимого запроса средствами ASP.NET.

**C#**

```
public class WsFederationRequestValidator
             : RequestValidator
{
  protected override bool IsValidRequestString(
        HttpContext context, string value,
        RequestValidationSource requestValidationSource,
        string collectionKey,
        out int validationFailureIndex)
  {
    validationFailureIndex = 0;
    if (requestValidationSource
      == RequestValidationSource.Form &&
         collectionKey.Equals(
           WSFederationConstants.Parameters.Result,
           StringComparison.Ordinal))
    {
      if (WSFederationMessage.CreateFromFormPost(
        context.Request) as SignInResponseMessage != null)
{
        return true;
      }
    }
    return base.IsValidRequestString(context, value,
requestValidationSource, collectionKey,
           out validationFailureIndex);
  }
}
```

Компания Trey Research использует собственную проверку запросов в конвейере HTTP, добавив данный код в файл Web.config приложения Orders, как показано ниже.

```
<system.web>
<httpRuntime requestValidationType="Orders.Website.
       WsFederationRequestValidator, Orders.Website" />
  ...
</system.web>
```

### Аутентификация и авторизация посетителей

Некоторые страницы приложения Orders, такие как домашняя страница и список продаваемой продукции, не требуют аутентификации. Посетители могут просматривать их анонимно. Однако страницы, относящиеся к размещению заказа, управлению учетной записью посетителя и просмотру существующих заказов, требуют аутентификации посетителей.

Компания Trey Research предоставляет посетителям возможность входить и выходить из системы с помощью ссылок, размещенных в верхней части каждой страницы. Они определены в странице макета, которая используется как главная страница для всего сайта. Страница макета названа **_Layout.cshtml** и размещена в папке **Views/Shared** веб-сайта Orders. Она содержит следующий код и разметку.

**CSHTML**

```
@if (User.Identity.IsAuthenticated)
{
<li>@Html.ActionLink("My Orders", "Index",
"MyOrders")</li>
<li>@Html.ActionLink("Log Out", "LogOff",
"Account")</li>
}
else
{
<li>@Html.ActionLink("Log On", "LogOn", "Account")</li>
}
```

Модули WIF автоматически заполняют экземпляр класса, который реализует интерфейс **IClaimsIdentity**, утверждениями в маркере, представленном посетителем, и присваивает это значение свойству **User.Identity** текущего контекста. Данный код может проверить, была ли произведена аутентификация пользователя по значению свойства **IsAuthenticated**, как показано в приведенном выше коде. Вы также можете использовать все прочие методы и свойства **User.Identity**, такие как проверка определенной роли посетителя.

### Пользовательская страница входа

**ActionLink** на странице **_Layout.cshtml**, которая генерирует гиперссылку Log On, загружает класс **AccountController** (в папке Controllers проекта **Orders.Website**) и вызывает метод действия **LogOn**. Это поведение также определено в файле Web.config, приведенном выше, с указанием страницы, которая должна открываться пользователям, не прошедшим аутентификацию, при попытке входа на защищенную страницу.

**XML**

```
<authentication mode="Forms">
<forms loginUrl="~/Account/LogOn" timeout="2880" />
</authentication>
```

Метод **Logon**, приведенный выше, задает нужные значения для области и возвращает URL-адрес, чтобы посетитель вернулся на правильную страницу после входа в систему. URL-адрес для возврата указывается в случае, если пользователь попал сюда с другой страницы. Используется другой метод с именем **GetRealm**, который определен в классе **AccountController**, для получения текущей области в виде **http://[***узел***].[***путь***]/**. И наконец, метод **LogOn** возвращает страницу входа с внешним видом и функциями, настроенными компанией Trey Research.

**C#**

```
[HttpGet]
public ActionResult LogOn()
{
  ViewData["realm"] = this.GetRealm();
  ViewData["returnUrl"] = Request.UrlReferrer != null
     ? Request.UrlReferrer.AbsoluteUri : this.GetRealm();
ViewData["acsNamespace"] = CloudConfiguration
     .GetConfigurationSetting("acsNamespace", null);
  return View();
}
```

Сам вид **Logon** находится в папке Views\Account проекта **Orders.Website** и генерирует пользовательскую страницу входа, содержащую кнопки для каждого из поддерживаемых поставщиков удостоверений (таких как Windows Live ID, Google и Yahoo!). Пользователи могут выбрать поставщика удостоверений и войти в систему, используя этого поставщика. Затем они будут направлены на соответствующую страницу приложения Orders.

Пользовательская страница входа создается изменением стандартной страницы, создаваемой службой ACS. Пользовательская страница использует значительное количество клиентского кода JavaScript для формирования видимого содержимого страницы и для взаимодействия с пользователем.

### Использование пользовательского атрибута авторизации

Некоторые страницы на сайте, такие как Checkout (Расчет) и My Orders (Мои заказы), требуют, чтобы пользователи были аутентифицированы. Для реализации этой модели компания Trey Research использует свой пользовательский атрибут, названный **AuthorizeAndRegisterUser**. Например, этот атрибут применен к целому классу **MyOrdersController**, а не к одному методу. Следующий код демонстрирует структуру класса **MyOrdersController**.

**C#**

```
[AuthorizeAndRegisterUser]
public class MyOrdersController : Controller
{
  ...
}
```

**Мнение Джаны**

Использование пользовательского атрибута — это хороший способ выполнять другие действия, связанные с аутентификацией, без необходимости добавления этого кода в каждый класс, например получение данных пользователя из базы данных. Также с его помощью становится легче определять, для каких страниц и ресурсов необходима аутентификация пользователей.

Пользовательский класс **AuthorizeAndRegisterUserAttribute**, приведенный ниже, определен в папке CustomAttributes проекта **Orders.Website**. Он значительно расширяет стандартный класс **AuthorizeAttribute**, который ограничивает доступ к методу действия для вызывающих.

**C#**

```
public class AuthorizeAndRegisterUserAttribute
             : AuthorizeAttribute
{
  private readonly ICustomerStore customerStore;

  public AuthorizeAndRegisterUserAttribute()
             : this(new CustomerStore())
  { }

public AuthorizeAndRegisterUserAttribute(
                    ICustomerStore customerStore)
  {
    this.customerStore = customerStore;
  }

public override void OnAuthorization(
       System.Web.Mvc.AuthorizationContext filterContext)
{
    ...
if (!filterContext.HttpContext.User.
                     Identity.IsAuthenticated)
    {
      base.OnAuthorization(filterContext);
      return;
    }
    var federatedUsername =
((IClaimsIdentity)filterContext.
        HttpContext.User.Identity).GetFederatedUsername();
    var customer = this.customerStore.FindOne(
federatedUsername);
    if (customer == null)
    {
      // Redirect to registration page.
      var redirectInfo = new RouteValueDictionary();
      redirectInfo.Add("controller", "Account");
      redirectInfo.Add("action", "Register");
redirectInfo.Add("returnUrl",
                       filterContext.HttpContext.Request
                         .Url.AbsolutePath);
      filterContext.Result
          = new RedirectToRouteResult(redirectInfo);
    }
  }
}
```

Атрибут **AuthorizeAndRegisterUserAttribute** создает экземпляр класса репозитория **CustomerStore**, который используется для доступа к данным этого пользователя в таблице **Customers** SQL Azure (как показано выше на рисунке 6 в этой главе). Если пользователь не прошел процесс аутентификации, атрибут направляет запрос в базовый экземпляр **AuthorizeAttribute** , который с помощью модулей WIF инициирует аутентификацию пользователя службой ACS.

Если же пользователь прошел аутентификацию, атрибут использует метод **GetFederatedUsername**, который приведен выше, для получения имени пользователя по маркеру утверждений аутентификации. Метод **FindOne** класса **CustomerStore** получает данные этого пользователя из таблицы **Customers**. Если пользователь еще не зарегистрирован, соответствующих данных нет в таблице **Customers**, и поэтому код перенаправит посетителя на страницу регистрации.

### Получение и хранение данных клиента

Использование основанных на утверждениях удостоверений и поставщиков удостоверений социальных сетей обычно означает, что ваше приложение должно хранить информацию для каждого зарегистрированного пользователя приложения, потому что маркер, предоставляемый аутентифицированным пользователем приложению, содержит только минимальный набор утверждений (обычно только идентификатор пользователя или его имя). Компания Trey Research хранит эту информацию в таблице **Customers** базы данных SQL Azure. Эта таблица содержит данные, которые пользователи предоставляют во время регистрации, и дополнительную информацию, которая сохраняется для каждого пользователя командой администраторов головного офиса (например, лимит кредита).

Для подключения к базе данных компания Trey Research использовала Microsoft Entity Framework и создала класс с именем **CustomerStore**, расположенный в папке Orders.Website.DataStores проекта веб-сайта. Этот класс предоставляет несколько методов для получения и хранения данных клиента, включая метод **FindOne**, который использовался в приведенном выше фрагменте кода. Ниже приведен код метода **FindOne**.

**C#**

```
public Customer FindOne(string userName)
{
  ...
using (var database
         = TreyResearchModelFactory.CreateContext())
  {
    var customer
= this.sqlCommandRetryPolicy.ExecuteAction(
() => database.Customers.SingleOrDefault(
c => c.UserName == userName));
    if (customer == null)
    {
      return null;
    }
    return new Customer
    {
      CustomerId = customer.CustomerId,
      UserName = customer.UserName,
      FirstName = customer.FirstName,
      LastName = customer.LastName,
      Address = customer.Address,
      City = customer.City,
      State = customer.State,
      PostalCode = customer.PostalCode,
```

```
      Country = customer.Country,
      Email = customer.Email,
      Phone = customer.Phone
    };
  }
}
```

Этот метод ищет запись, используя федеративное имя пользователя. Если он находит совпадение, метод создает новый экземпляр **Customer** и заполняет его данными из базы данных. Если нет, он возвращает **null**, указывая на то, что пользователь не зарегистрирован.

Вы можете заметить, что метод **FindOne**, показанный в предыдущем примере кода, сначала получает ссылку на базу данных, вызывая метод **CreateContext** класса **TreyResearchModelFactory** (он размещен в папке DataStores проекта **Orders.Website**). Этот метод использует другой класс, **CloudConfiguration**, для прочтения строки подключения к базе данных из файла ServiceConfiguration.cscfg или возвращает **null**, если параметр не найден.

**C#**

```
public static TreyResearchDataModelContainer
       CreateContext()
{
return new TreyResearchDataModelContainer(
CloudConfiguration.GetConfigurationSetting(
"TreyResearchDataModelContainer", null));
}
```

После получения ссылки на базу данных, код использует функциональный блок для обработки неустойчивых неисправностей (Transient Fault Handling Application Block) из библиотеки Enterprise Library для вызова метода, который подключается к хранилищу данных и получает данные для конкретного пользователя. Функциональный блок для обработки неустойчивых неисправностей предоставляет готовый механизм для выполнения действия определенное количество раз и с определенной задержкой между каждой из попыток. Он посылает события, которые вы можете использовать для контроля выполнения и сбора информации об ошибках в процессе выполнения.

**Примечание**

Функциональный блок для обработки неустойчивых неисправностей может быть использован для управления подключением к базе данных SQL Azure, очередям и топикам шины интеграции, а также кэшу Windows Azure. Этот блок является частью Enterprise Library Integration Pack для Windows Azure. Для получения более подробной информации см. «Microsoft Enterprise Library» по адресу http://msdn.microsoft.com/entlib/.

Хотя разработчики компании Trey Research и могли использовать код, чтобы генерировать политику динамически во время выполнения, они решили определить политику повторений, которую использует этот блок в файле Web.config приложения. Следующий фрагмент кода из этого файла содержит определение этой политики.

**XML**

```
<RetryPolicyConfiguration
    defaultRetryStrategy="Fixed Interval Retry Strategy"
defaultAzureStorageRetryStrategy
      ="Fixed Interval Retry Strategy"
    defaultSqlCommandRetryStrategy="Backoff Retry Strategy"
>
<incremental name="Incremental Retry Strategy"
retryIncrement="00:00:01"
               initialInterval="00:00:01"
               maxRetryCount="10" />
<fixedInterval name="Fixed Interval Retry Strategy"
retryInterval="00:00:05"
               maxRetryCount="6"
               firstFastRetry="true" />
<exponentialBackoff name="Backoff Retry Strategy"
minBackoff="00:00:05"
               maxBackoff="00:00:45"
               deltaBackoff="00:00:04"
               maxRetryCount="10" />
</RetryPolicyConfiguration>
```

Вы можете видеть, что он содержит стратегию по умолчанию, которая повторяет попытку каждую секунду до десяти раз. Также имеется стратегия доступа к хранилищу Windows Azure (таблицам, очередям и BLOB-объектам), которая повторяет попытку каждые 5 секунд максимум 6 раз, при этом первая попытка выполняется непосредственно после сбоя. И наконец, в коде представлена стратегия для подключения к базе данных SQL, которая повторяет попытку по истечении 5 секунд, после чего увеличивает задержку на 4 секунды для каждой последующей попытки, вплоть до максимальной задержки в 45 секунд и десяти возможных попыток.

Класс **CustomerStore**, который содержит метод **FindOne**, инициализирует политику повтора в своем конструкторе класса с использованием статического метода **GetDefaultSqlCommandRetryPolicy** класса **RetryPolicyFactory** (который является частью функционального блока для обработки неустойчивых неисправностей и используется, когда вы добавляете этот блок в свое приложение). Указанный конструктор также добавляет обработчик в метод **Retrying** этого блока. Этот обработчик записывает информацию в диагностику Windows Azure каждый раз, когда блок обнаруживает ошибку подключения.

**C#**

```
this.sqlCommandRetryPolicy
  = RetryPolicyFactory.GetDefaultSqlCommandRetryPolicy();

this.sqlCommandRetryPolicy.Retrying += (sender, args)
=> TraceHelper.TraceInformation("Retry - Count:{0},"
       + "Delay:{1}, Exception:{2}",
       args.CurrentRetryCount, args.Delay,
       args.LastException);
```

Если функциональный блок для обработки неустойчивых неисправностей не может завершить заданное действие за указанное количество попыток, он выдаст исключение, которое вы можете обработать в своем коде.

Когда политика повторов определена и все проинициализировано, коду остается только вызвать метод **ExecuteAction** и указать необходимое действие в виде параметра. В этом случае выполняемым действием является метод **SingleOrDefault** коллекции **Customers**, предоставленный классом **TreyResearchDataModelContainer**.

**C#**

```
    var customer
= this.sqlCommandRetryPolicy.ExecuteAction(
() => database.Customers.SingleOrDefault(
c => c.UserName == userName));
```

**Примечание**

Этот код использует простейшую из перегрузок **ExecuteAction**, имеющихся в функциональном блоке для обработки неустойчивых неисправностей. Другие перегрузки позволяют возвращать результат из метода и производить асинхронное выполнение. Для получения более подробной информации см. «ExecuteAction Method» по адресу http://msdn.microsoft.com/en-us/library/microsoft.practices.transientfaulthandling.retrypolicy.executeaction.aspx.

## Аутентификация доступа к очередям и топикам шины интеграции

Компания Trey Research использует очереди и топик шины интеграции для передачи сообщений между приложением Orders в Windows Azure, транспортными партнерами и локальной службой ведения журнала аудита. В разделе «Обеспечение безопасности очередей сообщений, топиков и подписок» в главе 4 «Реализация надежного обмена сообщениями и информацией в облаке» описываются настройки защиты, используемые компанией Trey Research для защиты очередей и топиков шины интеграции путем аутентификации пользователей и авторизации всех их действий.

В отличие от обычного процесса аутентификации службы ACS, описанного в данной главе, пользователями шины интеграции, скорее всего, будут сервисы и другие компоненты, а не посетители, использующие веб-обозреватель. Однако общие принципы те же: каждый запрос доступа к очередям и топикам шины интеграции должен пройти процесс аутентификации доверенным поставщиком удостоверений, а служба STS должна выдать маркер успешного завершения процесса аутентификации. Для аутентификации шины интеграции маркер должен содержать утверждения, которые могут использоваться для авторизации действий, таких как отправка сообщений в очереди и топики.

Например, приложения и пользователи, которые хотят получить доступ к службам, предоставляющим данные отчетов (описанные в разделе «Как в Trey Research используют службу отчетов SQL Azure» в главе 2 «Развертывание данных и приложения Orders в облаке» должны пройти процесс аутентификации службой ACS и предоставить маркер, содержащий необходимый набор утверждений для шины интеграции.

Служба ACS может быть настроена как для работы в качестве поставщика утверждений, так и поставщика маркеров (STS) для конечной точки шины интеграции. На рисунке 4 приведен обзор этого процесса. Клиенты получают маркер от службы ACS, который содержит соответствующие утверждения (такие как Listen, Send или Manage), и предоставляет его шине интеграции при получении доступа к очередям, топикам или конечной точке службы Service Bus Relay.

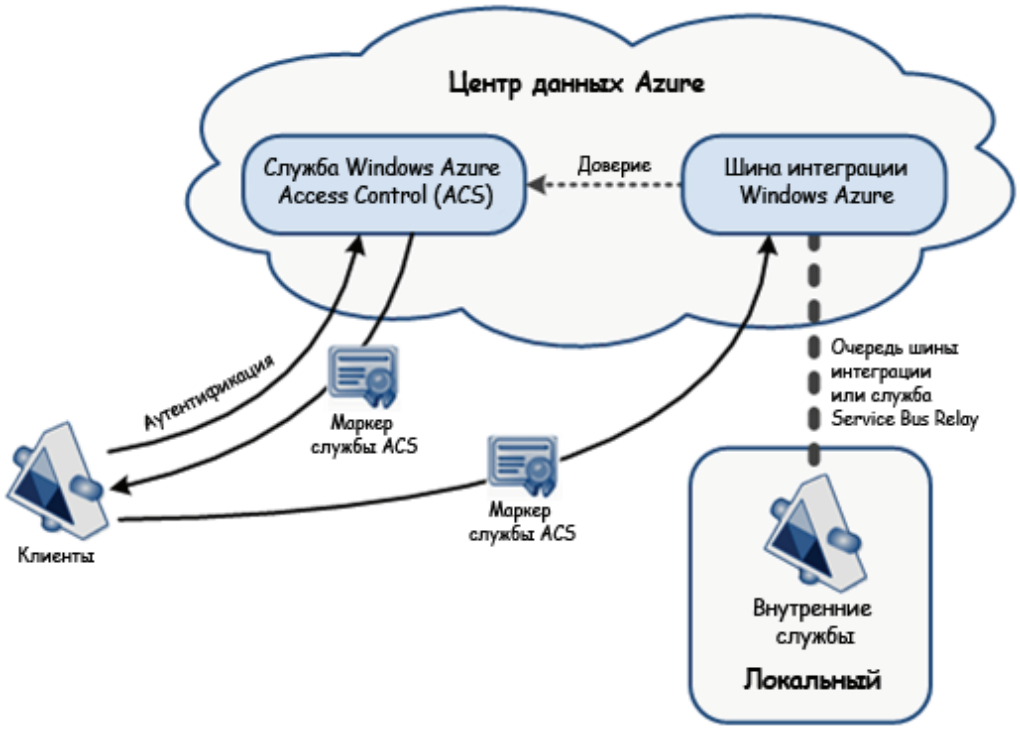



**Рисунок 4**

**Аутентификация конечных точек шины интеграции службой ACS**

Для получения более подробной информации о том, как служба ACS может быть использована для работы в режимах поставщика утверждений и поставщика маркеров для аутентификации и авторизации запросов к конечной точке шины интеграции, см. раздел «Аутентификация и авторизация шины интеграции Windows Azure» в «Приложении Б. Аутентификация пользователей и авторизация запросов».

## Резюме

В этой главе описывалось, как компания Trey Research решила проблему аутентификации посетителей приложения Orders и реализовала аутентификацию для доступа к очередям и топикам шины интеграции между службами и приложениями, размещенными локально и в облаке. В главе основное внимание было уделено использованию основанных на утверждениях удостоверений и аутентификации. Это наиболее распространенная и наиболее быстро развивающаяся технология для приложений на базе Windows Azure. В ней используются преимущества фреймворков и служб, специально разработанных для упрощения этого подхода.

Преимуществом аутентификации на основе утверждений также является простота управления списками пользователей и их правами доступа. Также поддерживается возможность единого входа в систему и выбор поставщиков удостоверений для упрощения аутентификации пользователей. Это также важная технология для защиты служб, таких как очереди и топики шины интеграции Windows Azure.

В данной главе также рассматривалось, как вымышленная организация Trey Research реализовала процесс аутентификации и авторизации в своем гибридном приложении Orders.

## Дополнительная информация

- «Access Control Service 2.0» по адресу http://msdn.microsoft.com/en-us/library/windowsazure/gg429786.aspx.
- «How To: Implement Token Transformation Logic Using Rules» по адресу http://msdn.microsoft.com/en-us/library/gg185955.aspx.
- «Securing Services» в документации MSDN WCF по адресу http://msdn.microsoft.com/en-us/library/ms734769.aspx.
- «Service Bus Authentication and Authorization with the Access Control Service» по адресу http://msdn.microsoft.com/en-us/library/hh403962.aspx.
- «Securing and Authenticating a Service Bus Connection» по адресу http://msdn.microsoft.com/en-us/library/dd582773.aspx.
- Статья «ACS How To's» об использовании службы ACS на сайте MSDN по адресу http://msdn.microsoft.com/en-us/library/gg185939.aspxпредоставляет полный список учебников по использованию службы ACS, включая настройку удостоверений и поставщиков удостоверений.
- Домашняя страница об управлении удостоверениями по адресу http://msdn.microsoft.com/en-us/security/aa570351.aspx содержит дополнительную информацию о Windows Identity Foundation.
- Подробное описание аутентификации на основе утверждений и службы Windows Azure Access Control приведено в руководстве «Claims-Based Identity and Access Control Guide (2nd Edition)» по адресу http://msdn.microsoft.com/en-us/library/ff423674.aspx.

# Глава 4. Реализация надежного обмена сообщениями и информацией в облаке

После того как специалисты компании Trey Research переместили веб-приложения, позволяющие клиентам размещать заказы, на технологическую платформу Windows Azure, перенесли различные базы данных, используемые приложением, и обеспечили защиту доступа к приложению, чтобы только аутентифицированные клиенты могли размещать заказы, им пришлось решать следующую задачу: как передать детали заказов различным партнерам, занимающимся транспортировкой и доставкой, а также как зафиксировать всю информацию для проведения аудита и проверки соответствия нормативным требованиям. Этот аспект системы является критичным, так как он поддерживает основную функцию решения Trey Research по выполнению размещенных клиентами заказов. Процесс обработки заказов требует надежного механизма безопасной и надежной передачи заказов.

**Мнение Джаны**

Надежное взаимодействие между приложением Orders и транспортными партнерами является ключевым моментом. Если имеется вероятность сбоя механизма обмена информацией, сообщения могут быть утеряны, заказы могут быть не выполнены, а клиенты могут уйти в другие компании.

В этой главе показано, как решение Trey Research решает различные проблемы, связанные с реализацией механизма обмена сообщениями и информацией, предоставляя возможность обработки заказов в облаке с помощью шины интеграции Windows Azure и службы Windows Azure Connect.

**Примечание**

В главе 5 «Обработка заказов в решении Trey Research» описывается, как в действительности решение Trey Research использует механизм обмена информацией и сообщениями в качестве основы, поддерживающей бизнес-логику для обработки заказов, размещенных клиентами.

## Сценарий и контекст

В исходной реализации системы Orders элементы приложения Orders работали локально, а процесс обработки заказов осуществлялся в среде, которая полностью контролировалась Trey Research. На рисунке 1 показано исходное приложение, в котором выделены компоненты, отвечающие за обработку заказа.


**Рисунок 1**

### Компоненты обработки заказов в локальном приложении

В локальном решении, когда клиент размещает заказ, приложение сохраняет информацию об этом заказе в таблице Orders в локальной базе данных. Таблица Audit Log в локальной базе данных содержит определенную информацию, в том числе информацию о работе системы и диагностику, а также детали необычных заказов, например заказов, превышающих определенную общую стоимость. Затем приложение Orders посылает сообщение соответствующему транспортному партнеру. В этом сообщении содержится предполагаемая дата доставки и информация об упаковке для данного заказа (например, вес и количество упакованных единиц). Транспортный партнер посылает сообщение в приложение Orders после выполнения доставки, что позволяет обновить таблицу в базе данных Orders.

В связи со специфическим характером продуктов, которые Trey Research производит, компания должна также гарантировать соблюдение законодательных требований в отношении сбыта некоторых товаров, в частности, в случае экспорта в другие страны и регионы. Эти требования подразумевают ведение подробных записей о продажах определенных электронных компонентов, которые могут входить в состав продуктов компании Trey Research, а также компонентов оборудования, которые могут быть использованы для изначально не предусмотренных целей. Анализ содержимого заказов — это сложный и строго контролируемый процесс, осуществляемый приложением третьей стороны, которое проверяет соответствие нормативным требованиям; оно работает на отдельном специально настроенном сервере.

Когда приложение Orders переместили в облако, в компании Trey Research задумались о том, как реализовать эту бизнес-логику с помощью новой архитектуры. Как вы помните из главы 2 «Развертывание данных и приложения Orders в облаке», компания Trey Research разместила данные на технологической платформе SQL Azure. База данных Orders была реплицирована в каждом центре данных, а приложение Orders было изменено, чтобы обеспечить доступ к базе данных, размещенной в том же центре данных, к которому подключен пользователь. Кроме того, в ожидании значительного увеличения объема заказов приложение проверки соответствия нормативным требованиям было перенесено в облако, чтобы использовать преимущества масштабируемости Windows Azure. Приложение для обеспечения соответствия нормативным требованиям является многопотоковым и может использовать все преимущества мощной платформы, на которой оно работает, поэтому было решено развернуть его только в одном центре данных. Однако по причинам соблюдения нормативных требований пришлось оставить журнал аудита локальным. На рисунке 2 показана структура итогового гибридного решения, опять с выделенными элементами обработки заказов.

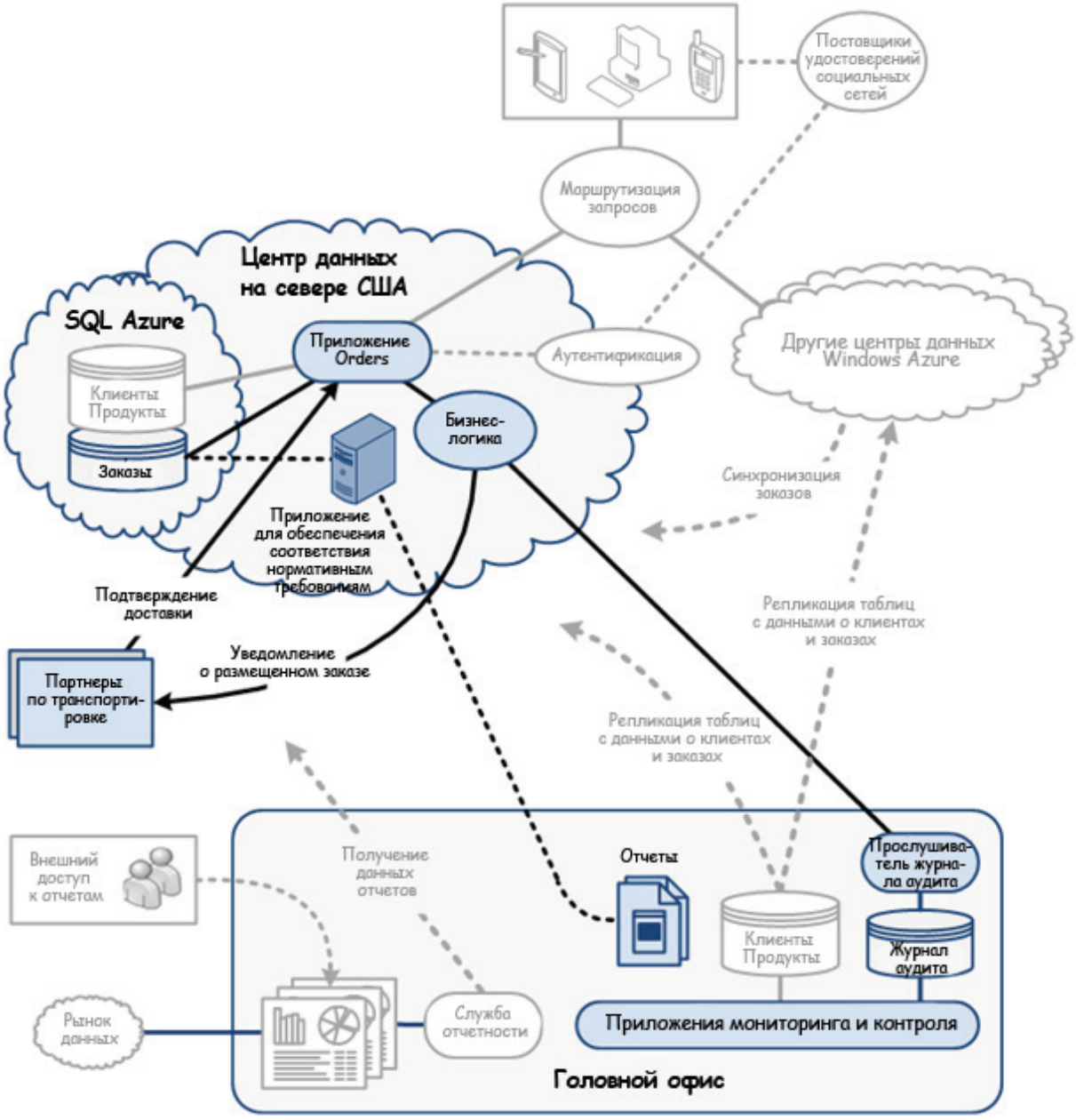



**Рисунок 2**

**Гибридная версия решения Trey Research**

Что касается клиента, то приложение Orders работает аналогично оригинальному решению, но логика обработки заказа теперь следующая:

- Когда клиент размещает заказ, приложение Orders выполняет следующие действия:
    - Сохраняет информацию о заказе в таблице «Заказы» в базу данных в локальном центре обработки данных под управлением SQL Azure. Все заказы синхронизируются во всех центрах данных Windows Azure, поэтому информация о состоянии заказа доступна для посетителей любого центра данных, на который они перенаправляются.
    - Отправляет соответствующее сообщение определенному транспортному партнеру. Выбор транспортной компании зависит от местоположения поставки.
    - Отправляет необходимую информацию для аудита, например заказы общей стоимостью более 10 000 долларов США, в таблицу «Журнал аудита» базы данных, размещенной в центре данных в головном офисе. Локальные приложения управления и мониторинга могут анализировать эту информацию.
- Приложение для обеспечения соответствия нормативным требованиям, предоставляемое сторонними разработчиками и работающее в облаке, постоянно проверяет таблицу «Заказы» на соответствие нормативным положениям и устанавливает флаг в таблице базы данных для тех заказов, которые требуют внимания менеджеров. Оно также генерирует ежедневный отчет, который сохраняется в надежном месте в центре данных в головном офисе.
- Когда транспортные партнеры доставляют заказ клиенту, они отправляют сообщение в приложение Orders (работающее в центре данных, из которого изначально посылалось сообщение о заказе), которое обновляет таблицу «Заказы» в базе данных.

Имейте в виду, что для упрощения некоторые функции и процессы, описанные в настоящем руководстве, не в полном объеме реализованы в примере приложения, который мы предоставляем, или могут работать несколько иным образом. Это упрощает установку и настройку примера приложения. Вам не придется получать и настраивать учетные записи Azure в нескольких центрах обработки данных; это также относится к службам, таким как SQL Azure Data Sync и SQL Reporting.

Если выразить это более простыми словами, то высокоуровневая структура облачных элементов гибридного решения относительно прямолинейна. Приложение Orders, работающее в облаке, устанавливает соответствие с ролью Windows Azure, как описано в главе 1 «Сценарий Trey Research», тогда как бизнес-логика, которая на самом деле обрабатывает заказы, может быть реализована в виде рабочей роли Windows Azure. Операция логики обработки заказов должна быть масштабируемой, чтобы справиться с ожидаемым ростом спроса при увеличении базы данных клиентов компании Trey Research, а также надежной, так как заказы не должны дублироваться и должны располагаться в соответствующих местах.

Бизнес-логика, отвечающая за обработку заказов, естественным образом делится на три области: взаимодействие с транспортными партнерами, определение заказов, для которых необходим аудит, и передача заказов в приложение для обеспечения соответствия нормативным требованиям. Компания Trey Research рассмотрела варианты реализации для каждой из этих областей.

## Взаимодействие с транспортными партнерами

Ключевой частью механизма обработки заказов является взаимодействие с транспортными партнерами. Рабочая роль должна проанализировать каждый заказ и перенаправить детали заказа наиболее подходящему транспортному партнеру. Понятие «наиболее подходящий» транспортный партнер связано с приложением и может меняться со временем, но на данный момент по экономическим соображениям выбран самый близкий партнер к производственным площадям компании Trey Research, откуда поставляются заказы. Заказы для местных клиентов (клиентов, находящихся в том же или соседнем штате, что и производство компании) выполняются местным партнером по транспортировке, а заказы для более отдаленных клиентов перевозятся удаленными транспортными партнерами на железнодорожном или авиационном транспорте, если это необходимо.

**Примечание**

После переговоров и тщательного анализа компания Trey Research решила работать с компанией Contoso, Inc. для оказания услуг по локальной транспортировке, а компания Fabrikam, Inc. была выбрана для доставки заказов более отдаленным клиентам.

На этапе разработки команда разработчиков компании Trey Research потребовала, чтобы все взаимодействия с транспортными партнерами соответствовали следующим критериям:

- Решение должно быть быстродействующим. Связь должна обеспечивать своевременную реакцию, чтобы не блокировать операции приложения Orders и не оказывать отрицательного влияния на процесс размещения заказов клиентами.
- Механизм связи должен быть устойчивым и надежным. Когда клиент размещает заказ и этот заказ подтверждается, его необходимо выполнить. Система не должна терять заказы, когда сообщения передаются из решения Trey Research соответствующему транспортному партнеру, даже если связь с транспортным партнером прерывается или система транспортного партнера работает с перебоями.
- Решение должно быть масштабируемым. Должна быть возможность простого добавления транспортных партнеров в решение без необходимости переписывания логики обработки заказов. Кроме того, должна быть возможность размещения приложения Orders на нескольких сайтах в облаке, без необходимости внесения изменений в код обработки нескольких экземпляров приложения.
- Решение должно быть гибким. Список транспортных партнеров может меняться со временем, также может меняться выбор партнера для выполнения определенного заказа в связи с изменением цен доставки.
- Все данные должны считаться конфиденциальными и быть защищены соответствующим образом. Необходимо предпринимать соответствующие меры безопасности, чтобы избежать возможности несанкционированного попадания деталей заказов, размещенных клиентом, в руки третьих сторон.

**Мнение Маркуса**

Способ передачи сообщений между распределенными компонентами в гибридном приложении должен быть надежным, защищенным, быстрым, масштабируемым и безопасным.

## Выбор механизма взаимодействия

В исходном локальном приложении связь между компанией Trey Research и транспортными партнерами осуществлялась с помощью набора операций с веб-службами. Некоторые вызовы этих операций инициировались в сети компании, а другие требовали от нее создания локальной веб-службы, которую вызывали партнеры для обновления статуса поставки. После переноса в облако с учетом описанных выше сложностей, компания Trey Research разработала несколько вариантов реализации связи между эквивалентной бизнес-логикой, помещенной в рабочую роль, и транспортными партнерами. В следующих разделах описаны эти варианты, а также их преимущества и ограничения.

### Электронный обмен данными (Electronic Data Interchange, EDI)

Рабочая роль могла подключиться к транспортному партнеру через интерфейс, протокол и формат, понимаемые как рабочей ролью, так и транспортным партнером, например EDIFACT, RosettaNet, cXML и BASDA. Это общепринятые и хорошо понимаемые стандарты, используемые многими большими организациями для обмена данными с другими компаниями. Кроме того, большинство современных протоколов EDI — асинхронные, поскольку предполагается, что соответствующие бизнес-процессы будут долгосрочными. Это позволяет обеспечить быстродействие рабочей роли в периоды повышения спроса.

Однако рабочая роль может потребовать дополнительного программного обеспечения и инфраструктуры для подключения к интерфейсу EDI. Microsoft BizTalk Server предоставляет адаптеры для многих хорошо известных протоколов и форматов, но это решение требует передачи всех заказов через сервер BizTalk Server, работающий локально. Дополнительная информация представлена в статье о гибридной реализации с помощью сервера BizTalk Server, Windows Azure, шины интеграции и SQL Azure «Hybrid Reference Implementation Using BizTalk Server, Windows Azure, Service Bus and SQL Azure» на сайте http://msdn.microsoft.com/en-us/windowsazure/hh547113(v=VS.103).aspx. Каждый транспортный партнер может иметь свой интерфейс EDI, что усложняет расширение бизнес-логики Trey Research на дополнительных партнеров, хотя есть возможность абстрагировать эти отличия в механизме соединительного звена в рабочей роли.

Этот подход потребует реализации соединительного звена для каждого партнера. Также компания Trey Research должна была учесть, что не все транспортные партнеры предоставят интерфейс EDI с их системами. И наконец, в этом подходе безопасность и защита сообщений контролируется транспортным партнером, а не компанией.

### Веб-службы (модель «продвижение»)

Если транспортный партнер создает интерфейс веб-служб в системе управления доставкой, рабочая роль может использовать этот интерфейс и вызывать соответствующие операции для *продвижения* деталей заказов транспортному партнеру. Веб-службы основаны на общепринятой, хорошо понимаемой развивающейся технологии. Кроме того, вызвать операции веб-служб из рабочей роли обычно довольно просто. С точки зрения безопасности запросы можно легко зашифровать, хотя уровень безопасности и защиты сообщений контролируется транспортным партнером, предоставляющим веб-службу.

В этом подходе могут быть определенные проблемы. Во-первых, если транспортный партнер не предоставляет интерфейс веб-службы, то этот подход использовать невозможно. Если транспортный партнер предоставляет такой интерфейс, компания Trey Research должна оценить возможную сложность очень вероятного сценария, когда различные транспортные партнеры могут внедрить различные наборы операций веб-служб и ожидать сообщений с запросами в различных форматах. Эти отличия можно абстрагировать путем создания механизма соединительного звена в рабочей роли и предоставления коннекторов для каждого транспортного партнера, но это усложняет решение Trey Research.

Еще одна проблема в том, что веб-служба может не обеспечивать надлежащий уровень надежности. Рабочая роль может не получить уведомление, когда система транспортного партнера выдает ошибку, которая приводит к потере деталей заказа; транспортный партнер может не получить уведомление о необходимости вызова операции, которая опять предоставит детали заказа.

Связь также может быть проблемой; если веб-служба транспортного партнера временно недоступна или невозможно установить соединение, то компания Trey Research не сможет отправить информацию о заказах.

### Веб-службы (модель «извлечение»)

Веб-службы могут предоставить безопасный, масштабируемый и надежный механизм связи, если они реализованы корректно. По этой причине компания Trey Research решила изменить предыдущий вариант путем внедрения веб-службы как части рабочей роли и внедрения операций, которые транспортные партнеры вызывают для получения (или *извлечения*) информации о заказах из приложения Trey Research. Этот подход позволил компании получить полный контроль над уровнем защищенности сообщений и безопасности, реализовать преимущества масштабируемости веб-службы, присущие рабочей роли, размещенной Windows Azure, а также позволил добиться более высокого уровня надежности. Если система транспортного партнера выдает сбой при обработке заказа, она может повторно подключиться к веб-службе и извлечь детали заказа снова после перезапуска.

Рабочая роль может иметь одинаковый интерфейс веб-службы для всех транспортных партнеров. Новые транспортные партнеры могут быть легко интегрированы без необходимости изменения рабочей роли. Рабочая роль не ждет, пока транспортный партнер извлекает детали заказа, поэтому система продолжает отвечать на запросы клиентов. Кроме того, рабочая роль пользуется преимуществами службы Service Bus Relay для объединения характеристик безопасности и независимости от места в одном решении. Транспортный партнер может подключаться к хорошо знакомой конечной точке, анонсированной службой Service Bus Relay, которая может аутентифицировать транспортного партнера с помощью службы Windows Azure Access Control (ACS), а затем прозрачно перенаправлять сообщения к конечным точкам веб-службы, публикуемым рабочей ролью.

Однако транспортные партнеры должны разработать собственное программное обеспечение для подключения к веб-службе; у них может не быть возможности разрабатывать и развертывать пользовательские программные решения специально для интеграции с компанией Trey Research. Если транспортный партнер хочет подключиться к веб-службе, в его обязанности входит запрос о том, имеются ли заказы, которые необходимо доставить. Если транспортный партнер не посылает эти запросы достаточно часто, то заказы могут быть доставлены несвоевременно, что приведет к жалобам клиентов. Если системе транспортного партнера не удается успешно посылать запросы в веб-службу, заказы отправлены не будут. Масштабируемость также может быть проблемой в системах транспортных партнеров. С увеличением объема заказов транспортные партнеры могут посылать запросы в веб-службу недостаточно часто, что может привести к задержке в формировании заказов.

### Очереди хранилища Windows Azure

Приложение Orders работает в роли Windows Azure, поэтому компания Trey Research решила публиковать детали заказов в виде сообщений в очередь хранилища Windows Azure. Транспортные партнеры могут подключаться к этой очереди для извлечения деталей заказов, которые необходимо доставить. Сообщения с подтверждением доставки могут отправляться обратно в рабочую роль через другую очередь. Этот механизм относительно простой, надежный, масштабируемый и защищенный; очереди хранилища Windows Azure управляются и поддерживаются в центре данных, а компания Trey Research имеет полный контроль над тем, какие транспортные партнеры будут иметь права доступа к очереди для извлечения и публикации сообщений. Кроме того, семантика операции извлечения может быть реализована надежным способом; если система транспортного партнера выдает сбой с ошибкой после получения сообщения, но до его обработки, это сообщение может быть явным образом возвращено в очередь, откуда его можно получить после перезапуска системы транспортного партнера.

Windows Azure предоставляет интерфейс REST API для доступа к очередям хранилища Windows Azure, что позволяет транспортным партнерам реализовать их систему, используя любую технологию, которая позволяет подключаться к сети и передавать запросы REST.

С этим подходом также связаны некоторые проблемы. Как и в некоторых предыдущих вариантах, у каждого транспортного партнера должно быть желание внедрить программное обеспечение для подключения к очереди хранилища Windows Azure и интегрировать его в свое решение. Также чтобы предотвратить получение транспортным партнером заказа, предназначенного другому транспортному партнеру, компания Trey Research должна создать отдельную очередь для каждого партнера. Этот подход может усложнить логику в рабочей роли, а также усложнить процесс добавления или удаления транспортных партнеров. И наконец, безопасность контролируется путем использования ключей учетной записи хранения вместо ACS, что предоставляет каждому транспортному партнеру доступ ко всей учетной записи хранения, а не к отдельной очереди. Чтобы гарантировать наличие доступа каждого транспортного партнера к соответствующей очереди, каждая очередь должна быть создана в отдельной учетной записи хранения с собственным ключом.

### Очереди шины интеграции Windows Azure

Чтобы устранить некоторые сложности использования очередей хранилища Windows Azure, разработан аналогичный, но более выгодный вариант публикации деталей заказов в виде сообщений в очереди шины интеграции Windows Azure. Транспортные партнеры могут подключиться к этой очереди для извлечения заказов, которые необходимо отправить. Сообщения с подтверждением доставки могут отправляться обратно в рабочую роль через другую очередь. Этот подход более масштабируемый; рабочая роль может публиковать сообщения в очереди шины интеграции с той же скоростью, что и размещение заказов. Инфраструктура шины интеграции может поместить сообщения в буфер до того момента, пока они не будут извлечены транспортным партнером. Он также предлагает повышенную надежность. После того как рабочая роль публикует сообщение в очереди шины интеграции, его невозможно потерять. Оно либо останется в очереди до истечения срока действия (период срока действия сообщения можно настроить), либо транспортный партнер получит его. Как и с очередью хранилища Windows Azure, очереди шины интеграции поддерживают надежные операции извлечения, поэтому если система транспортного партнера выдает сбой после получения сообщения, это сообщение может быть явным образом возвращено в очередь. Параметры безопасности очень гибкие и конфигурируемые, особенно если сравнить их с параметрами безопасности очередей хранилища Windows Azure. Они управляются с помощью службы Windows Azure Access Control (ACS).

Однако, как и раньше, каждый транспортный партнер должен быть готов подключиться к соответствующим очередям шины интеграции для извлечения сообщений и отправки сообщений с подтверждением доставки. Если транспортный партнер не соглашается с таким подходом, а вместо этого настаивает на том, чтобы заказы передавались с помощью его собственных системных интерфейсов (например, набор веб-служб), то в этом случае компании Trey Research придется создать пользовательский компонент для извлечения сообщений из очереди, их конвертации в соответствующий формат для транспортного партнера, а затем передавать их с помощью согласованного протокола. Этот же компонент может содержать логику для ожидания подтверждения доставки от транспортного партнера и публикации сообщения в рабочую роль. Также как и раньше, чтобы предотвратить получение транспортным партнером заказа, предназначенного другому транспортному партнеру, компания Trey Research должна создать отдельную очередь для каждого партнера. Этот подход может усложнить логику в рабочей роли, а также усложнить процесс добавления или удаления транспортных партнеров.

**Примечание**

Дополнительная информация о сравнении функций и вариантов использования очередей хранилища Windows Azure и очередей шины интеграции Windows Azure представлена в статье «Windows Azure Queues and Windows Azure Service Bus Queues - Compared and Contrasted» на сайте http://msdn.microsoft.com/en-us/library/windowsazure/hh767287(v=vs.103).aspx.

### Топики и подписки шины интеграции Windows Azure

Очереди шины интеграции представляют собой привлекательную альтернативу с возможностями масштабируемости, если бы не необходимость создания и управления отдельной очередью для каждого партнера. Поэтому в конечном итоге компания Trey Research решила опубликовать детали заказов в виде сообщений в топике шины интеграции Windows Azure; транспортные партнеры подписываются на топик для извлечения заказов, которые необходимо отправить. Сообщение с подтверждением получения деталей заказа и сообщение о доставке возвращаются в рабочую роль через очередь шины интеграции.

Как и очереди шины интеграции, топики и подписки шины интеграции отличаются высокой масштабируемостью и надежностью, а также настраиваемым уровнем безопасности. Однако при передаче сообщений транспортному партнеру они более гибкие, чем очереди шины интеграции. Рабочая роль может добавлять метаданные в сообщения, которые указывают, какой транспортный партнер должен их обрабатывать, а затем публиковать эти сообщения в топике шины интеграции. Каждый транспортный партнер может подключаться к топику шины интеграции с помощью собственной подписки, которая может фильтровать сообщения на основании этих метаданных, после чего транспортный партнер получает только те заказы, которые он должен обработать. Топики позволяют перенаправлять сообщения по нескольким направлениям, что позволяет, например, направлять заказы, стоимостью более 10 000 долларов США в прослушиватель журнала аудита.

Единственным и общим, с описанными выше вариантами, недостатком этого подхода является то, что каждый транспортный партнер должен быть готов подключаться к соответствующему топику шины интеграции для извлечения сообщений. Кроме того, компания Trey Research может создать пользовательский компонент связи для интеграции с системами транспортных партнеров. Топики также имеют некоторые ограничения, например на данный момент топик может содержать максимум 2000 подписок и может поддерживать до 100 одновременных соединений (ограничение 100 одновременных соединений также касается очередей). Однако компания пришла к выводу, что система Orders вряд ли достигнет одного из этих двух лимитов.

## Как Trey Research связывается с транспортными партнерами

В конце концов, компания Trey Research решила посылать заказы от рабочей роли к транспортным партнерам путем использования топика шины интеграции. Каждый транспортный партнер получает сообщения с помощью подписки, которая фильтрует заказы. Таким образом, транспортный партнер получает только те заказы, которые он должен отправить. Дополнительная информация об использовании очередей шины интеграции, топиков и подписок представлена в статье «Queues, Topics, and Subscriptions» на сайте http://msdn.microsoft.com/en-us/library/windowsazure/hh367516.aspx.

**Мнение Джаны**

Очереди шины интеграции предоставляют надежный и масштабируемый механизм для связи между службами, работающими в облаке, и локальными приложениями.

Для устранения потенциального технологического разрыва между шиной интеграции и системами, реализованными транспортными партнерами, компания Trey Research разработала набор компонентов связи для перевода сообщений, извлеченных из шины интеграции, и их преобразования в формат, ожидаемый транспортным партнером. Расположение этих компонентов связи зависит от взаимоотношений компании с транспортным партнером:

- В случае с местным транспортным партнером, компанией Contoso, компании Trey Research удалось убедить партнера установить коннектор и интегрировать его в их собственную систему. Компания Trey Research предоставила учетные данные, необходимые для подключения коннектора для ожидания передачи данных из соответствующей подписки шины интеграции. Собственная система транспортного партнера использует этот коннектор для извлечения деталей заказов их подписки. Кроме того, коннектор имеет интерфейс, который система транспортного партнера использует для публикации сообщений с подтверждением в очередь шины интеграции, которую прослушивает приложение Orders.

  **Примечание**

  Реализация коннектора как части системы транспортного партнера не заставляет транспортных партнеров использовать код .NET Framework в их решении. Функции шины интеграции Windows Azure предоставляются через набор API HTTP REST (набор инструментальных средств разработки Windows Azure SDK просто предоставляет упаковщик .NET Framework для этих API), что позволяет транспортному партнеру использовать любую знакомую технологию, которая может генерировать запросы REST и использовать ответы REST, включая язык программирования Java.

- Удаленный транспортный партнер, компания Fabrikam, является многонациональной организацией, и эксплуатационный персонал не хотел устанавливать программное обеспечение Trey Research на свои серверы; они предпочли, чтобы компания Trey Research подключалась с помощью интерфейсов, которые они предоставляют в своих системах. Для удовлетворения этого требования, компания Trey Research внедрила адаптер для публикации заказов для компании Fabrikam, и этот адаптер был развернут в рабочей роли. Этот механизм позволяет сохранить логику публикации сообщений для транспортных партнеров, вне зависимости от того, местные это партнеры или удаленные. Если компании Trey Research понадобиться добавить нового удаленного транспортного партнера в будущем, то просто потребуется создать и установить соответствующий адаптер.

**Мнение Маркуса**

Внедрение адаптеров и коннекторов позволяет приложениям оставаться независимыми от механизма связи. Если необходимо, топик шины интеграции, используемый компанией Trey Research, можно переключить на другое средство передачи информации, при этом нужно будет изменить только адаптеры и коннекторы.

На рисунке 3 проиллюстрированы технологии, используемые компанией Trey Research для реализации передачи сообщений между приложением Orders и транспортными партнерами. В следующих разделах описан подход, используемый компанией Trey Research, для создания решения с целью передачи сообщений на основе этих технологий.

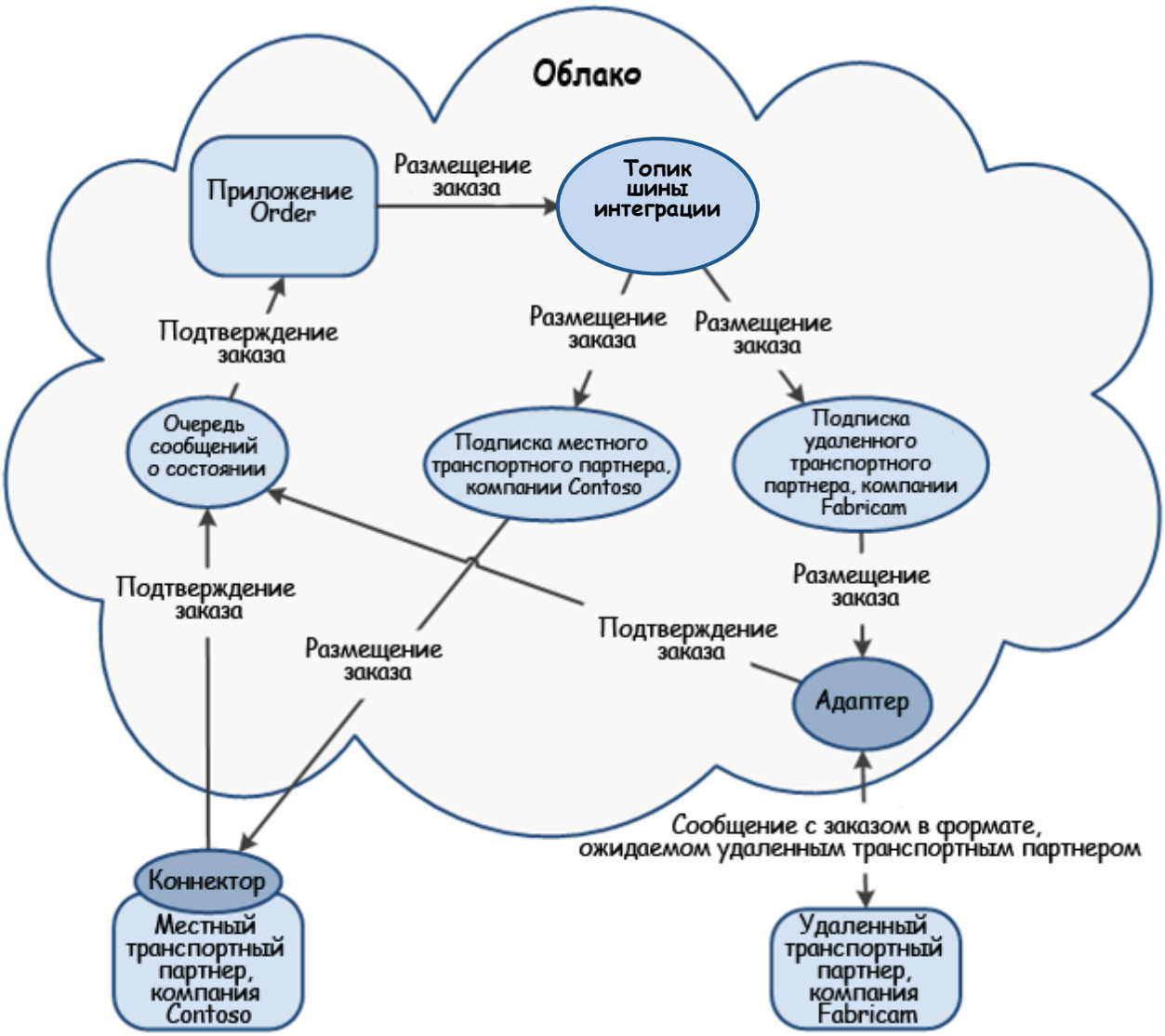



**Рисунок 3**

**Технологии передачи сообщений, используемые компанией Trey Research для взаимодействия с транспортными партнерами**

В примере приложения Trey Research, который можно загрузить для работы с этим руководством, реализованы многие описываемые здесь технологии и методы. Однако, чтобы упростить установку и настройку и уменьшить количество предварительных условий и требований, необходимых для создания полноценных учетных записей Windows Azure, набор функций и некоторые детали реализации отличаются от текста этого руководства.

Используя набор инструментальных средств разработки Windows Azure SDK, вы можете внедрить приложения, которые отправляют и получают сообщения с помощью классов **MessageSender** и **MessageReceiver** в пространстве имен **Microsoft.ServiceBus.Messaging**. Эти операции синхронны. Например, метод **Send** класса **MessageSender** ожидает окончания операции отправки перед продолжением работы, и аналогично метод **Receive** класса **MessageReceiver** ожидает либо момента, когда сообщение становится доступным, либо окончания указанного времени ожидания. Эти методы на самом деле являются лишь фасадом набора запросов HTTP REST, а очереди и топики шины интеграции являются удаленными службами, к которым предоставляется доступ через Интернет. Таким образом, в ваших приложениях должно быть учтено, что:

- Для завершения операций отправки и получения может потребоваться достаточно много времени, а ваше приложение не должно блокировать ожидание окончания выполнения этих операций.
- Отправитель может публиковать сообщения в любое время, а получателю необходимо ожидать сообщения в более чем одной очереди.
- Операции отправки и получения могут выдавать ошибку по многим причинам, например по причине сбоя связи между вашим приложением и шиной интеграции в облаке, из-за нарушений безопасности, вызванных изменением политики безопасности, реализованной очередью или топиком шины интеграции (администратор может принять решение об отзыве или изменении прав учетной записи по каким-либо причинам), из-за переполнения очереди (они имеют конечный размер) и т. п. Некоторые из этих сбоев могут быть результатом нерегулярных ошибок, а другие — возникать из-за более постоянных проблем.

Компания Trey Research решила внедрить библиотеку, которая добавила упаковщики для функций очереди и топика шины интеграции, доступные в пространстве имен **Microsoft.ServiceBus.Messaging**. Эта библиотека предоставляется с примером решения в проекте **Orders.Shared**. Классы, размещенные в папке Communication этого проекта, инкапсулируют существующие классы **MessageSender**, **MessageReceiver** и **BrokeredMessage** (среди других). Целью новых классов является абстрагирование функций отправки и получения, то есть все операции отправки и получения выполняются асинхронно. Эта библиотека также включает элементы модели безопасности, реализованные компанией Trey Research. Для получения дополнительной информации см. раздел «Обеспечение безопасности очередей сообщений, топиков и подписок» далее в этой главе.

**Примечание**

Дополнительная информация и рекомендации по оптимизации производительности при использовании системы передачи сообщений шины интеграции Windows Azure представлена в статье «Best Practices for Performance Improvements Using Service Bus Brokered Messaging» на сайте http://msdn.microsoft.com/en-us/library/windowsazure/hh528527.aspx.

В следующих разделах описывается структура этой библиотеки, классы, предоставляемые в ней, и то, как эти классы расширяют функциональность, предоставляемую очередями, топиками и подписками шины интеграции.

### Асинхронная отправка сообщений в очередь шины интеграции

Компания Trey Research использует очередь шины интеграции, чтобы предоставить транспортным партнерам связь с приложением Orders. Чтобы отправить сообщение в очередь шины интеграции с помощью библиотеки **Orders.Shared**, приложение выполняет следующие шаги:

1. Создание объекта **BrokeredMessage** и наполнение его необходимой информацией. Класс **BrokeredMessage** — это тип данных, предоставляемый корпорацией Microsoft в пространстве имен **Microsoft.ServiceBus.Messaging**.
2. Создание объекта **ServiceBusQueueDescription** и определение пространства имен шины интеграции, имени очереди и набора действительных учетных данных в виде ключа доступа, а также имени соответствующей учетной записи. Класс **ServiceBusQueueDescription** используется в проекте **Orders.Shared**.
3. Создание объекта **ServiceBusQueue** с помощью объекта **ServiceBusQueueDescription**. Тип **ServiceBusQueue** инкапсулирует асинхронную функциональность отправки сообщений. Создание экземпляра типа **ServiceBusQueue** для подключения к базовой очереди шины интеграции в режиме **PeekLock**.
4. Вызов метода **Send** объекта **ServiceBusQueue**. Параметром метода **Send** должен быть объект **BrokeredMessageAdapter**, который упаковывает объект **BrokeredMessage**, созданный ранее. Класс **ServiceBusQueue** содержит экземпляр класса **MessageSenderAdapter** (определенный в папке Communication\Adapters проекта **Orders.Shared**), который реализует интерфейс **IMessageSenderAdapter**. Метод **Send** использует этот объект **MessageSenderAdapter** для отправки сообщений.

> **Примечание**
>
> Класс **MessageSenderAdapter** — это просто класс упаковщика, созданный для упрощения модульного тестирования с помощью макета объектов.

> Пример использования типа **ServiceBusQueue** для отправки сообщений см. в методе **SendToUpdateStatusQueue** класса **OrderProcessor** в проекте **TransportPartner**.

> **Мнение Маркуса**
>
> Классы **ServiceBusTopic** и **ServiceBusSubscription** проекта **Orders.Shared** реализуют аналогичный подход для **ServiceBusQueue**, инкапсулируя асинхронную функциональность на основе классов **MessageSender** и **MessageReceiver** соответственно.
>
> Классы **MessageSenderAdapter**, **MessageReceiverAdapter** и **BrokeredMessageAdapter** позволяют модульным тестам (в проекте **Orders.Shared.Tests**) создавать макеты отправителей, получателей и промежуточных сообщений.

В следующем фрагменте кода показана реализация метода **Send** в классе **ServiceBusQueue**, вместе с соответствующими членами класса, используемыми методом **Send**.

**Мнение Маркуса**

Метод **Guard**, используемый методами в классе **ServiceBusQueue** и других местах, проверяет, были ли инициированы указанные параметры; строка не должна быть нулевой или пустой.

**C#**

```
public class ServiceBusQueue
{
  private readonly ServiceBusQueueDescription
description;
  ...
  private readonly IMessageSenderAdapter senderAdapter;
  ...

  public ServiceBusQueue(
    ServiceBusQueueDescription description)
  {
    Guard.CheckArgumentNull(description, "description");
    this.description = description;
    ...
    var sender = messagingFactory.CreateMessageSender(
      this.description.QueueName.ToLowerInvariant());
    this.senderAdapter = new
MessageSenderAdapter(sender);
...
  }
  ...

  public void Send(IBrokeredMessageAdapter message)
{
    Guard.CheckArgumentNull(message, "message");

    this.Send(message, this.senderAdapter);
}

  public void Send(IBrokeredMessageAdapter message,
IMessageSenderAdapter sender)
  {
    Guard.CheckArgumentNull(message, "message");
    Guard.CheckArgumentNull(sender, "sender");

    Task.Factory
      .FromAsync(sender.BeginSend, sender.EndSend,
message,
null,TaskCreationOptions.AttachedToParent)
      .ContinueWith(
        taskResult =>
        {
          try
          {
            if (taskResult.Exception != null)
```

```
            {
              TraceHelper.TraceError(
                taskResult.Exception.ToString());
            }
          }
          finally
          {
            message.Dispose();
          }
        });
  }
  ...
}
```

**Мнение Маркуса**

Убедитесь, что ваш код корректно ликвидирует созданный вами экземпляр класса **BrokeredMessage** после использования, чтобы убедиться, что все используемые ресурсы освобождены.

В классе **ServiceBusQueue** обработка, осуществляемая методом **Send**, требует прикрепления обработки в виде дочерней задачи путем использования параметра **TaskCreationOptions.AttachedToParent**. Таким образом, сбой в дочерней задаче может быть обнаружен при отправке сообщения и обработан родительской задачей, что позволяет родительской задаче легче отменить операцию **Receive**.
В этом примере любые исключения вносятся в журнал путем использования статического сообщения **TraceError** класса **TraceHelper**. Класс **TraceHelper** определяется в папке Helpers проекта **Orders.Shared**. Этот класс действует как упаковщик обработчиков события отслеживания, предоставленный библиотекой **System.Diagnostics**, и более подробно описывается в главе 7 «Мониторинг и управление приложением Orders».

**Мнение Маркуса**

Обратите внимание, что класс **ServiceBusQueue** не использует функциональный блок для обработки неустойчивых неисправностей. Это связано с тем, что использование функционального блока для обработки неустойчивых неисправностей для запуска асинхронных процессов не обеспечивает той же гибкости, что и использование объекта **Task**. Если вы решите использовать функциональный блок для обработки неустойчивых неисправностей, вы должны взвесить все преимущества декларативной четкости способа выполнения и извлечения критического кода по сравнению с точным управлением, которое может потребоваться при запуске этого кода в виде фоновой задачи.

### Получение сообщений из очереди шины интеграции и их асинхронная обработка

Класс **ServiceBusQueue** создает и предоставляет объект **MessageReceiver**, который вы можете использовать для получения сообщений с помощью метода **GetReceiver**. Это обычный объект получателя сообщений без дополнительной функциональности, а вызов метода **Receive** для этого объекта выполняет синхронную операцию получения. В самом простом виде получатель, использующий эту технологию, может быть заблокирован на протяжении длительного периода времени, ожидая появления сообщения. Кроме того, при получении сообщения может понадобиться значительное усилие для выполнения необходимой обработки, во время которой могут поступить дополнительные сообщения. Эти сообщения не будут обрабатываться до того, как получатель закончит текущую задачу и извлечет следующее сообщение. Если сообщение срочное, такой подход может быть неприемлемым.

Класс **MessageReceiver** также поддерживает асинхронные операции с помощью операций **BeginReceive** и **EndReceive**. Тип **ServiceBusReceiverHandler** также в проекте **Orders.Shared** расширяет функциональность для предоставления класса, который может получать и обрабатывать сообщения асинхронно, разделяя процессы бизнес-логики и обработки исключений из кода, который подключается к очереди.

Класс **ServiceBusReceiverHandler** предоставляет метод **ProcessMessages**, который приложение может использовать для асинхронного ожидания сообщений, поступающих в очередь шины интеграции, и их обработки (приложение определяет очередь для ожидания как параметр конструктору данного класса). В следующем примере кода показан конструктор и реализация метода **ProcessMessages**.

**C#**

```
public class ServiceBusReceiverHandler<T>
{
  private readonly IMessageReceiverAdapter receiver;
  private Func<T, ServiceBusQueueDescription, string,
Task>
messageProcessingTask;

  public ServiceBusReceiverHandler(
    IMessageReceiverAdapter receiver)
  {
    ...
    this.receiver = receiver;
  }

  ...

  // Параметр Func (который возвращает Task)
предоставляет
  // вызывающей стороне большую степень контроля над
результатом выполнения задачи и
  // обработки исключений
  public void ProcessMessages(Func<T,
ServiceBusQueueDescription, string, Task>
    taskForProcessingMessage,
    CancellationToken cancellationToken)
  {
    ...
    this.messageProcessingTask =
taskForProcessingMessage;

    this.ReceiveNextMessage(cancellationToken);
  }

  ...
}
```

Метод **ProcessMessages** ожидает делегата в качестве первого параметра. Делегат должен ссылаться на метод, который будет запускаться при каждом получении сообщения. Целью этого метода-делегата является выполнение бизнес-логики, требуемой приложением при получении каждого сообщения (подробный пример этой логики см. в разделе «Получение и обработка заказа транспортным партнером» в главе 5 «Обработка заказов в решении Trey Research». Метод **ProcessMessages** сохраняет этого делегата локально, а затем вызывает локальный метод **ReceiveNextMessage**, как показано в следующем примере кода.

**C#**

```
...
public TimeSpan? MessagePollingInterval { get; set; }
...
private void ReceiveNextMessage(
  CancellationToken cancellationToken)
{
  if (this.MessagePollingInterval.HasValue)
  {
    Thread.Sleep(this.MessagePollingInterval.Value);
}

  Task.Factory
    .FromAsync<TimeSpan,

IBrokeredMessageAdapter>(this.receiver.BeginReceive,
                              this.receiver.EndReceive,
                              TimeSpan.FromSeconds(10),
                              null,
                              TaskCreationOptions.None)
    .ContinueWith(
      taskResult =>
      {
        // Начало получения следующего сообщения сразу
после
        // получения предыдущего.
        // Это не приведет к переполнению стека, так как
        // вызов будет осуществляться из новой задачи
(Task).

        this.ReceiveNextMessage(cancellationToken);

        if (taskResult.Exception != null)
        {
          TraceHelper.TraceError(
            taskResult.Exception.Message);
        }

        this.ProcessMessage(taskResult.Result);
      },
      cancellationToken);
}
```

Метод **ReceiveNextMessage** реализует простую стратегию опроса. Он ожидает указанный период времени перед попыткой получить сообщение из очереди (очередь сообщения считывается в режиме **PeekLock**). Операция получения выполняется асинхронно, и если сообщение доступно, то метод запускает новую задачу ожидания любых следующих сообщений, а затем вызывает метод **ProcessMessage** для обработки нового полученного сообщения.

**C#**

```
private void ProcessMessage(
  IBrokeredMessageAdapter message)
{
  if (message != null)
  {
    ...
    this.messageProcessingTask(message.GetBody<T>(),
queueDescription, token)
     .ContinueWith(
       processingTaskResult =>
       {
         if (processingTaskResult.Exception != null)
         {
           if (message.DeliveryCount <= 3 &&
               !(processingTaskResult.Exception.
                 InnerException is
InvalidTokenException))
           {
             // Если отмена не удалась, сообщение станет
             // видимым после окончания действия
             // блокировки
             Task.Factory.FromAsync(message.BeginAbandon,
message.EndAbandon, message,
                 TaskCreationOptions.AttachedToParent)
               .ContinueWith(
                 taskResult =>
                 {
                   if (taskResult.Exception != null)
                   {
                     TraceHelper.TraceError(
                       "Ошибка при отмене сообщения:
{0}",
                        taskResult.Exception.
                          InnerException.Message);
                   }

                   var msg = taskResult.AsyncState
                     as BrokeredMessage;
                   if (msg != null)
                   {
                     msg.Dispose();
                   }
                 });
           }
           else
           {
             Task.Factory.FromAsync(
                 message.BeginDeadLetter,
                 message.EndDeadLetter, message,
                 TaskCreationOptions.AttachedToParent)
```

```
              .ContinueWith(
                taskResult =>
                {
                  if (taskResult.Exception != null)
                  {
                    TraceHelper.TraceError(
                      "Ошибка при отправке сообщения в"
                      + "очередь DeadLetter: {0}",
                      taskResult.Exception.
                        InnerException.Message);
                  }

                  var msg = taskResult.AsyncState
                    as BrokeredMessage;
                  if (msg != null)
                  {
                    msg.Dispose();
                  }
                });
            TraceHelper.TraceError(
              processingTaskResult.
                Exception.TraceInformation());
          }
        }
        else
        {
          Task.Factory
            .FromAsync(message.BeginComplete,
               message.EndComplete, message,
               TaskCreationOptions.AttachedToParent)
            .ContinueWith(
              taskResult =>
              {
                if (taskResult.Exception != null)
                {
                  TraceHelper.TraceError(
                    "Ошибка при выполнении"
                    + "сообщения. Завершено: {0}",
                    taskResult.Exception.
                      InnerException.Message);
                }
                var msg = taskResult.AsyncState
                  as BrokeredMessage;
                if (msg != null)
                {
                  msg.Dispose();
                }
              });
        }
    });
  }
}
```

**Мнение Маркуса**

Интервал опроса действует в качестве регулятора, помогая избежать перегрузки частей системы. Идеальное значение интервала опроса зависит от доступных вычислительных ресурсов транспортного партнера, ожидаемого объема заказов и количества экземпляров рабочей роли. Например, вероятно, неправильно указывать небольшой интервал опроса, если у транспортного партнера ограниченные вычислительные ресурсы, особенно в периоды высокого спроса при создании большого количества заказов. В таком случае более длинный интервал между сообщениями позволяет системам транспортных партнеров работать более эффективно, причем топик эффективно исполняет роль буфера выравнивания нагрузки.

Метод **ProcessMessage** вызывает метод-делегат, предоставляемый принимающим приложением, который хранится в свойстве **messageProcessingTask** для обработки сообщения. Метод **ProcessMessage** реализует простую, но эффективную политику обработки исключений, используемую при получении и обработке сообщений. Например, если при получении сообщения происходит системное исключение, метод **ProcessMessage** будет пытаться отклонить сообщение и отпустить любые блокировки; последующий вызов метода **ReceiveNextMessage** может успешно считать сообщение, если ошибка была случайной. Однако, если одно и то же сообщение не удалось получить три раза или если невозможно обработать сообщение в результате ошибки аутентификации (если простой веб-маркер, полученный от транспортного партнера является недействительным), то сообщение размещается в очередь недоставленных сообщений. Если сообщение обрабатывается успешно, то метод **ProcessMessage** вызывает асинхронную версию метода **Complete**, чтобы удалить сообщение из очереди.

На рисунке 4 проиллюстрирован процесс управления с помощью объекта **ServiceBusReceiverHandler** при получении и обработке сообщения принимающим приложением.

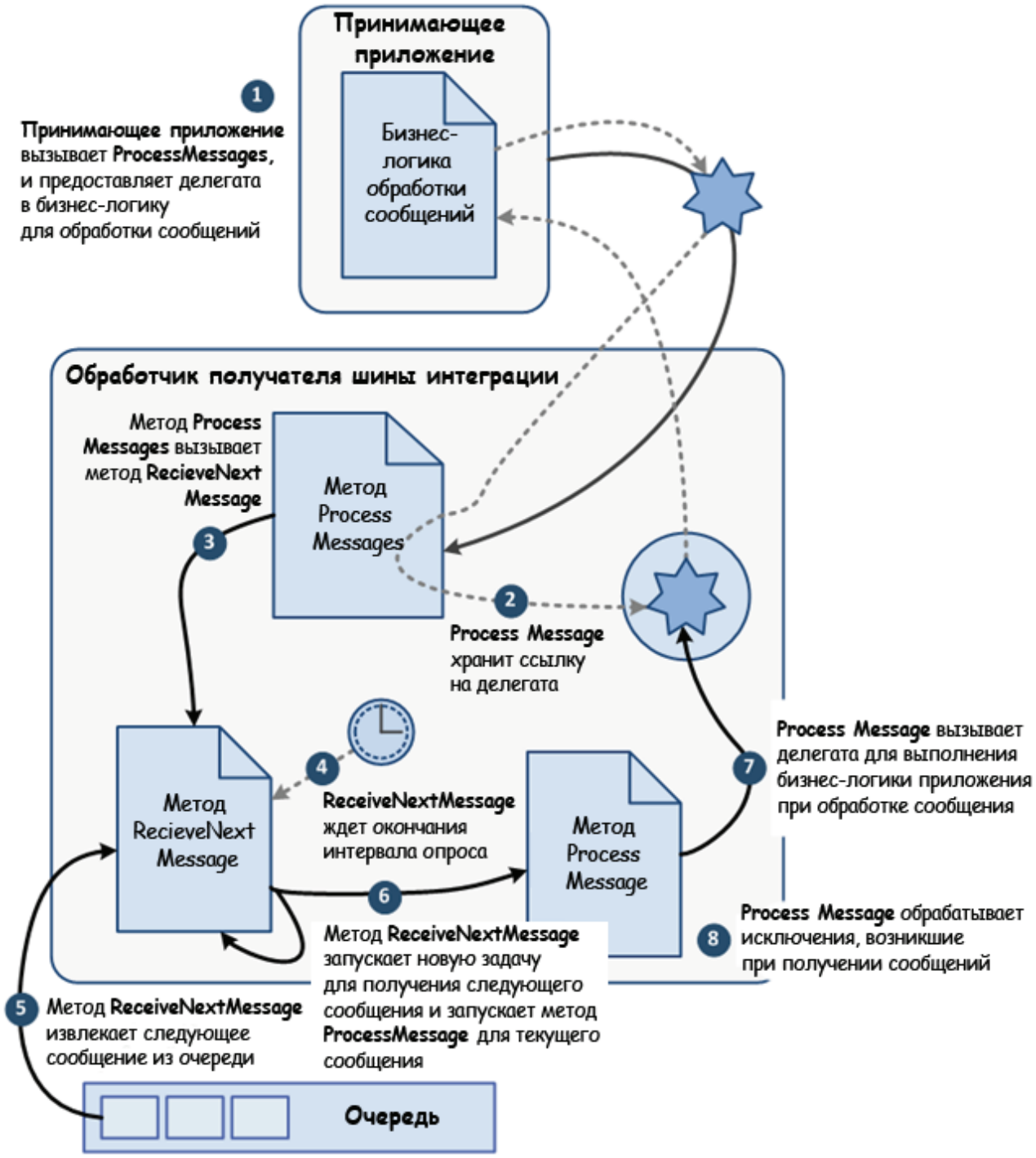



**Рисунок 4**

**Процесс управления при получении сообщений с помощью объекта ServiceBusReceiverHandler**

### Отправка сообщений в топик шины интеграции

Хотя транспортные партнеры посылают сообщения в приложение Orders через очередь шины интеграции, рабочая роль в реализации решения Trey Research использует топик шины интеграции для отправки заказов каждому транспортному партнеру. Топики шины интеграции аналогичны очередям, но имеют одно важное отличие: сообщения, размещенные в топике, можно фильтровать с помощью подписки шины интеграции и направлять к определенному прослушивателю, связанному с данной подпиской. Фильтрация основывается на метаданных, добавленных в сообщение перед его отправкой, и только подписки, указывающие фильтр, который соответствует значению этих метаданных, получат это сообщение.

Компания Trey Research использовала этот механизм для добавления свойства **TransportPartnerName** к каждому сообщению о заказе, где указан транспортный партнер, который должен обработать сообщение. Таким образом, каждое сообщение о заказе будет получено только тем транспортным партнером, который должен отправить заказ. Кроме того, компания Trey Research добавила свойство **OrderAmount** к каждому сообщению. Приложение аудита подписывается на тот же топик, что и транспортные партнеры, но фильтрует сообщения, извлекая и проверяя детали всех заказов на сумму более 10 000 долларов США, как указывается в этом свойстве. В следующем коде, взятом из метода **Execute** класса **NewOrderJob** в папке Jobs проекта **Orders.Workers**, показан пример того, как решение Trey Research заполняет свойства сообщения, чтобы направить его определенному транспортному партнеру.

**C#**

```
var brokeredMessage = new BrokeredMessage(msg)
{
  ...
  Properties = {
    { "TransportPartnerName", transportPartnerName },
    ...
    { "OrderAmount", orderProcess.Order.Total } },
  ...
};
```

> **Примечание**
>
> Класс **NewOrderJob** более подробно описан в главе 5 «Обработка заказов в решении Trey Research».

Методы шины интеграции, используемые для отправки сообщений в топик, очень похожи на методы, используемые для отправки сообщений в очередь. Однако, чтобы контролировать незначительные изменения этих методов, разработчики компании Trey Research создали два пользовательских класса, предназначенные для облегчения использования топиков шины интеграции. Это классы **ServiceBusTopicDescription** и **ServiceBusTopic**, которые находятся в папке Communication проекта **Orders.Shared**.

В примере решения Trey Research имеются лишь незначительные отличия между классами **ServiceBusQueueDescription** и **ServiceBusTopicDescription**. Классы **ServiceBusQueue** (который инкапсулирует функциональность очереди шины интеграции) и **ServiceBusTopic** (который предоставляет аналогичную функциональность, но для топика шины интеграции) отличаются более значительно, в основном способами использования этих типов в решении Trey Research, а не механизмами отправки сообщений:

- В отличие от класса **ServiceBusQueue**, класс **ServiceBusTopic** не инициирует получателя. При необходимости клиенты подписываются на топик путем создания соответствующего получателя. Это устраняет связь между отправителем и получателем и позволяет различным клиентам подписываться и получать сообщения без необходимости перенастраивания топика. Это также позволяет отделить бизнес-логику от проблем маршрутизации сообщений.

- Класс **ServiceBusQueue** отправляет сообщение и вызывает исключение (которое заносится в журнал) только в случае, если не удалось отправить сообщение. И наоборот, класс **ServiceBusTopic** принимает два делегата **Action**, которые срабатывают при отправке сообщения или в случае возникновения ошибки. Этот подход позволяет компании Trey Research применять более широкие возможности обработки исключений при отправке деталей заказов транспортному партнеру при размещении сообщений о статусе заказа в приложении Orders.
- Класс **ServiceBusQueue** использует статический метод **FromAsync** класса **Task.Factory** для асинхронной отправки сообщений. В отличие от предыдущего подхода, класс **ServiceBusTopic** использует функциональный блок для обработки неустойчивых неисправностей из библиотеки Enterprise Library для обнаружения таких ошибок при публикации сообщения в топик и производит явные повторные попытки выполнения операции **Send**, если это необходимо. Обоснование этого подхода аналогично описанному в предыдущем пункте.

---

В следующем коде показано определение метода **Send** в классе **ServiceBusTopic**. Как описано выше, два метода **Action** передаются в метод в качестве параметров (один выполняется после отправки сообщения, а второй — в случае ошибки при отправке) вместе с функцией, которая создает сообщение, и объектом состояния.

**C#**

```
public void Send(Func<BrokeredMessage> createMessage,
object objectState,
                 Action<object> afterSendComplete,
                 Action<Exception, object> processError)
{
  ...
}
```

> Код может считать тело экземпляра **BrokeredMessage** только один раз. При реализации метода, который использует экземпляр **BrokeredMessage** и может выполняться более одного раза, как в случае использования функционального блока для обработки неустойчивых неисправностей, необходимо создавать и заполнять экземпляр **BrokeredMessage** при каждом вызове метода. Поэтому метод **Send** принимает функцию, которая создает сообщение, вместо принятия существующего экземпляра сообщения.

Метод **Send** использует функциональный блок для обработки неустойчивых неисправностей. Конструктор класса **ServiceBusTopic** инициирует блок, загружает политику по умолчанию для настройки связи шины интеграции Windows Azure и настраивает обработчик события **Retrying**, который записывает информацию в службу диагностики Windows Azure.

**C#**

```
this.serviceBusRetryPolicy = RetryPolicyFactory.
  GetDefaultAzureServiceBusRetryPolicy();

this.serviceBusRetryPolicy.Retrying += (sender, args) =>
   TraceHelper.TraceWarning("Retry in ServiceBusTopic -
"
+ "Count:{0}, Delay:{1}, Exception:{2}",
     args.CurrentRetryCount, args.Delay,
     args.LastException);
```

Затем метод **Send** вызывает одну из асинхронных перегрузок метода **ExecuteAction** блока для обработки неустойчивых неисправностей и передает необходимые параметры, как показано в следующем фрагменте кода. Этими параметрами являются асинхронные начальный и конечный делегаты, действие, которое должно выполниться после успешного завершения процесса, и действие, которое должно выполниться после выполнения определенного количества попыток (параметры политики повторных действий указаны в конфигурационном файле).

**C#**

```
this.serviceBusRetryPolicy.ExecuteAction<BrokeredMessage
>(
  ac =>
  {
    var message = createMessage();
    var dictionary
        = (objectState as Dictionary<string, object>);
if (dictionary.ContainsKey("message"))
    {
      dictionary["message"] = message;
    }
    else
    {
      dictionary.Add("message", message);
    }
    this.sender.BeginSend(message, ac, objectState);
},
  ar =>
  {
    this.sender.EndSend(ar);
    return (ar.AsyncState as Dictionary<string,
object>)["message"] as BrokeredMessage;
  },
  (message) =>
  {
    try
    {
      afterSendComplete(objectState);
    }
    catch (Exception ex)
    {
      TraceHelper.TraceError(ex.Message);
    }
    finally
    {
      message.Dispose();
    }
  },
  e =>
  {
    processError(e, objectState);
    var message = (objectState as Dictionary<string,
object>)["message"] as BrokeredMessage;
    message.Dispose();
  });
```

Асинхронный начальный делегат (**ac**) сначала вызывает функцию, переданную в метод **Send** в качестве параметра **createMessage**, для создания экземпляра **BrokeredMessage**. Затем он получает ссылку на словарь **Dictionary**, сохраненный в состоянии объекта (который также передается в метод **Send** в качестве параметра) и добавляет к ней ссылку на экземпляр **BrokeredMessage**. Он должен придерживаться ссылки на **BrokeredMessage**, чтобы позже ее можно было корректно ликвидировать. После этого код вызывает метод **BeginSend** экземпляра **MessageSender** шины интеграции, на который ссылается свойство **sender** класса **ServiceBusTopic**. Он передает в качестве параметра экземпляр **BrokeredMessage** для отправки, ссылку для вызова, предоставляемую функциональным блоком для обработки неустойчивых неисправностей, и словарь **Dictionary**, содержащий копию сообщения в виде состояния объекта. Ссылка на эту копию сообщения, отправленного через посредника, сохраняется, так что код может удалить его и корректно освободить используемые ресурсы. Это происходит в действиях, выполняемых после отправки сообщения, вне зависимости от того, было ли сообщение отправлено успешно или произошла ошибка.

> **Примечание**
>
> В **Dictionary** предоставляется поточно-ориентированный объект, который содержит информацию о состоянии, на которую ссылаются методы **BeginSend** и **EndSend**, асинхронно посылающие сообщение в очередь.

Асинхронный конечный делегат (**ar**) сначала вызывает метод **EndSend** экземпляра шины интеграции **MessageSender**. Затем он извлекает словарь **Dictionary**, содержащий сообщение, из состояния объекта и возвращает его в виде экземпляра **BrokeredMessage**. Он передается в действие, выполняемое при успешной отправке сообщения.

Если процесс успешно размещает сообщение в топике, он вызывает действие **Action**, на которое ссылается параметр **afterSendComplete**. Если происходит ошибка, то код использует класс **TraceHelper** для регистрации сообщения об ошибке.

Если процессу не удалось разместить сообщение в топике, он вызывает действие **Action**, на которое ссылается параметр **processError**. Код передает действию **processError** исключение, возвращенное из класса **MessageSender**, и состояние объекта, содержащее сообщение. После завершения действия **processError** код получает ссылку на экземпляр **BrokeredMessage**, сохраненный в переменной **objectState**, и удаляет ее.

### Подписка на топик шины интеграции

Одним из основных преимуществ использования топиков шины интеграции для распределения сообщений является предоставление уровня разделения между отправителем и получателями. Отправитель может создать сообщение с дополнительными свойствами, которое проходит фильтрацию в рамках топика, для перенаправления сообщения определенным получателям. Однако получатели должны подписываться для получения сообщений, и количество подписчиков не зависит от топика. Например, компания Trey Research может добавить новых транспортных партнеров и настроить отправку сообщений этим новым партнерам, просто изменив критерии фильтрации в топике. Новые подписчики могут подписаться на топик и получать сообщения, которые соответствуют определенным критериям фильтрации. Компания может добавить дополнительных подписчиков, которые ожидают сообщения и передают их на проверку, или другие типы обслуживания.

В приложении Orders компания Trey Research создала один топик шины интеграции для каждого развернутого экземпляра приложения (другими словами, имеется один топик для каждого центра данных). Все транспортные партнеры подписываются на все эти топики и получают предназначенные для них сообщения на основании правил фильтрации, установленных компанией Trey Research для выбора транспортного партнера.

Подписки и фильтры шины интеграции создаются методом **SetupServiceBusTopicAndQueue** в программе установки проекта **TreyResearch.Setup**. Следующий фрагмент кода показывает соответствующие части этого метода.

**C#**

```
private static void SetupServiceBusTopicAndQueue()
{
  ...
  // Создание одной подписки для каждого транспортного
партнера с
  // соответствующим критерием фильтра.
  var transportPartners = new[] {
    "Contoso", "Fabrikam" };
  for (int i = 0; i <= 1; i++)
  {
    string transportPartnerName = transportPartners[i];
    string formattedName = transportPartnerName.Replace(
      " ", string.Empty).ToLowerInvariant();
    ...

    var serviceBusTopicDescription =
      new ServiceBusSubscriptionDescription
      {
Namespace = ServiceBusNamespace,
TopicName = TopicName,
SubscriptionName = string.Format(
          "{0}Subscription", formattedName),
        Issuer = Issuer,
DefaultKey = DefaultKey
      };

    var serviceBusSubscription =
      new ServiceBusSubscription(
serviceBusTopicDescription);

    string filterExpression = string.Format(
      "TransportPartnerName = '{0}'",
        transportPartnerName);
serviceBusSubscription.CreateIfNotExists(
filterExpression);
    ...
  }
}
```

Компания Trey Research внедрила два пользовательских класса (**ServiceBusSubscriptionDescription** и **ServiceBusSubscription**, расположенных в папке Communication проекта **Orders.Shared**) для подключения к подпискам. Класс **ServiceBusSubscriptionDescription** определяет свойства подписки, указывая, к какому топику и в каком пространстве имен шины интеграции подключаться, а также имя подписки, которую необходимо использовать. В следующем примере кода показано определение этого класса. Обратите внимание, что в примере этой компании код заполняет свойство **SubscriptionName** именем транспортного партнера; помните, что каждая подписка шины интеграции фильтрует сообщения с помощью этого свойства.

**C#**

```
public class ServiceBusSubscriptionDescription
{
  public string Namespace { get; set; }
  public string Namespace { get; set; }
  public string SubscriptionName { get; set; }
...
}
```

Конструктор класса **ServiceBusSubscription** принимает заполненный экземпляр класса **ServiceBusSubscriptionDescription** и подключается к указанным топику и подписке, как показано в следующем фрагменте кода. Этот метод также создает класс **MessageReceiver** для подписки на топик.

**C#**

```
public ServiceBusSubscription(
ServiceBusSubscriptionDescription description)
{
  ...
  var runtimeUri
    = ServiceBusEnvironment.CreateServiceUri("sb",
this.description.Namespace,
string.Empty);
  var messagingFactory
    = MessagingFactory.Create(runtimeUri, ...);

  this.receiver
    = messagingFactory.CreateMessageReceiver(
this.description.TopicName.ToLowerInvariant()
            + "/subscriptions/" +
this.description.SubscriptionName
                    .ToLowerInvariant(),
ReceiveMode.PeekLock);
}
```

### Получение сообщений из топика и их асинхронная обработка

Чтобы получить сообщение из топика, приложение может использовать объект **ServiceBusReceiverHandler**, инициализированный получателем, инкапсулированным в объекте **ServiceBusSubscription**. Дополнительные сведения о классе **ServiceBusReceiverHandler** см. в разделе «Получение сообщений из очереди шины интеграции и их асинхронная обработка» ранее в этой главе. В следующем примере кода показано, как приложение может создавать объект **ServiceBusReceiverHandler** для получения сообщений **NewOrderMessage** из подписки (класс **NewOrderMessage** описан в главе 5 «Обработка заказов в решении Trey Research»).

**C#**

```
var serviceBusSubscription = new ServiceBusSubscription(
            ...);
var receiverHandler
  = new ServiceBusReceiverHandler<NewOrderMessage>
          (serviceBusSubscription.GetReceiver())
{
  MessagePollingInterval = TimeSpan.FromSeconds(2)
};
...
```

**Мнение Маркуса**

Заданный вами интервал опроса сообщений для получения сообщений из очереди или топика должен учитывать переменные, специфичные для вашей среды (например, мощность центрального процессора), и ожидаемый объем работы (например, количество заказов, которые следует обработать, и количество экземпляров рабочей роли).

Затем приложение может вызвать метод **ProcessMessages** только что извлеченного экземпляра **ServiceBusReceiverHandler** и передать его делегату, указав код, который должен исполниться при получении каждого сообщения. Напомним, что этот процесс был описан в разделе «Получение сообщений из очереди шины интеграции и их асинхронная обработка» ранее в этой главе. Пример кода:

**C#**

```
  ...
receiverHandler.ProcessMessages(
  (message, queueDescription, token) =>
  {
    return Task.Factory.StartNew(
      () => this.ProcessMessage(message,
                                queueDescription),
this.tokenSource.Token,
      TaskCreationOptions.None,
      context);
  },
this.tokenSource.Token);
```

### Реализация адаптеров и коннекторов для преобразования и переформатирования сообщений

Как описано в разделе «Выбранный вариант для взаимодействия с транспортным партнером» ранее в этой главе, компания Trey Research использует адаптеры и коннекторы для извлечения сообщений из подписки шины интеграции для каждого транспортного партнера, а затем преобразовывает эти сообщения в формат, понятный транспортному партнеру, перед передачей сообщения для обработки.

**Примечание**

В коде решения, представленного в этом руководстве, макеты версий локального и удаленного транспортных партнеров реализованы с помощью приложения Windows Forms. Для локального транспортного партнера, Contoso, коннектор интегрирован в Windows Forms. Для удаленного транспортного партнера, Fabrikam, коннектор реализован аналогичным способом как часть кода Windows Forms для этого партнера. Однако это сделано только в целях упрощения и демонстрации возможностей; в реальной реализации компания Trey Research встраивает адаптер для удаленного партнера в рабочую роль, как описано во втором пункте.

Решение Trey Research включает два примера транспортных партнеров. Один из них занимается поставками местным клиентам, находящимся в том же штате или соседних штатах, что и компания Trey Research, а другой — доставляет товары удаленным клиентам. Эти транспортные партнеры определены в классах **ContosoTransportPartner** и **FabrikamTransportPartner** приложения Windows Forms в проекте **TransportPartner**. Оба транспортных партнера реализовывают собственные системы для отслеживания и доставки грузов.

Contoso, местный транспортный партнер, запускает коннектор в собственной инфраструктуре, который подключается непосредственно к шине интеграции Windows Azure для извлечения и отправки сообщений. Эта функциональность реализована в классе **Connector** в папке Connectivity. Fabrikam, удаленный транспортный партнер, предоставляет интерфейс службы, и адаптер, запущенный как часть решения Trey Research, взаимодействует с шиной интеграции и изменяет формат сообщений в вызовах службы. Формат ответов службы изменяется как и формат сообщений и публикуется назад в шину интеграции. Адаптер реализуется в классе **Adapter**, который также находится в папке Connectivity.

Когда транспортный партнер получает запрос на доставку заказа, коннектор или адаптер (в зависимости от транспортного партнера) публикует сообщение с подтверждением в очередь шины интеграции. Эта очередь представляет собой хорошо известную и защищенную конечную точку, доступную всем транспортным партнерам. Классы **Connector** и **Adapter** являются потомками класса **OrderProcessor** (определяется в папке Connectivity проекта **TransportPartner**), а этот класс на самом деле обрабатывает соединение между транспортным партнером и шиной интеграции. В классе **FabrikamTransportPartner** приложения Windows Forms процесс управления выглядит следующим образом:

- Метод **OnLoad** создает объект **Adapter** и вызывает его метод **Run**. Метод **Run** класса **Adapter** наследуется из класса **OrderProcessor**.
- Метод **Run** класса **OrderProcessor** создает объект **ServiceBusReceiverHandler** для подключения к подписке шины интеграции, из которой он ожидает получить заказы, и вызывает метод **ProcessMessages** этого объекта.
- Первый параметр метода **ProcessMessages** в классе **ServiceBusReceiverHandler** является делегированной функцией (определена как лямбда-выражение в коде примера), обеспечивающей бизнес-логику, которая должна выполняться при получении заказа из топика.
- Объект **ServiceBusReceiverHandler** вызывает эту функцию после получения каждого сообщения. Эта стратегия позволяет отсоединить механизм получения сообщения из очереди или топика (как реализовано классом **ServiceBusReceiverHandler**) от логики преобразования этого сообщения в формат, ожидаемый транспортным партнером, и отправки этого запроса во внутреннюю систему, реализованную партнером.

---

Следующий пример, взятый из файла OrderProcessor.cs, показывает структуру этого кода.

**C#**

```
public void Run()
{
  ...
foreach (...)
  {
    ...
    var receiverHandler = new
      ServiceBusReceiverHandler<...>(...);

receiverHandler.ProcessMessages(
      (message, ..., ...) =>
      {
        return Task.Factory.StartNew(
          // Описание логики преобразования сообщения.
          // Параметр сообщения содержит тело
          // сообщения, полученного из топика.
          () => this.ProcessMessage(
                  message, ...),
          ...);
      }, ...);
  }
}
```

В классе **OrderProcessor** лямбда-выражение вызывает локальный метод **ProcessMessage** (не следует путать с **ServiceBusReceiverHandler.ProcessMessages**) для передачи сообщения во внутреннюю систему локального партнера и ожидает ответа, вызывая метод **ProcessOrder** (этот метод предоставляет логику, которая является специфичной для транспортного партнера и реализуется в классе **Adapter**.) Так как метод **ProcessMessage** работает, используя отдельную задачу, он может синхронно ожидать окончания работы метода **ProcessOrder**, не влияя на быстроту ответа приложения. В следующем фрагменте кода приведена часть реализации метода **ProcessMessage** в классе **OrderProcessor**.

> **Примечание**
>
> Многие детали метода **ProcessMessage**, например, назначение переменной **trackingId** и операции, выполняемые методом **ProcessOrder** в классах **Connector** и **Adapter**, представленные в примере решения, поясняются в главе 5 «Обработка заказов в решении Trey Research».

**C#**

```
protected virtual void ProcessMessage(
NewOrderMessage message,
  ServiceBusQueueDescription queueDescription)
{
  var trackingId = this.ProcessOrder(
    message, queueDescription);

  if (trackingId != Guid.Empty)
  {
    ...
this.SendOrderReceived(message,
      queueDescription, statusMessage, trackingId,
token);
  }
}
```

После того как заказ обработан транспортным партнером, метод **ProcessMessage** вызывает локальный метод **SendOrderReceived** объекта **OrderProcessor** для отправки соответствующего ответного сообщения назад в приложение Orders через очередь шины интеграции, указанную как второй параметр в методе **ProcessMessage**.

> **Примечание**
>
> Детали метода **SendOrderReceived** более подробно описаны также в главе 5 «Обработка заказов в решении Trey Research».

### Согласующие сообщения и ответы

В отличие от работы веб-служб, система обмена сообщениями, реализованная путем использования очередей и топиков сообщений шины интеграции, является, по сути, однонаправленным механизмом. Хотя иногда это рассматривается как ограничение, на самом деле это позволяет значительно повысить скорость ответов этой формы обмена сообщениями; отправитель не должен ожидать ответа от удаленного, иногда не очень надежного получателя, когда б он не разместил сообщение. Однако неизбежно могут возникать ситуации, когда отправитель ожидает какого-то ответа, даже если это просто подтверждение того, что получатель действительно получил сообщение. Как раз таковой является ситуация в сценарии Trey Research. Когда приложение Orders помещает детали заказа в топик, то ожидает получения ответа с подтверждением получения заказа.

Однако может пройти некоторое время между этими двумя событиями, и приложение Orders не должно блокировать ожидание получения ответа. Чтобы исправить ситуацию, компания Trey Research реализовала двунаправленную систему обмена сообщения с помощью комбинации топиков и очередей шины интеграции. Приложения Orders публикует сообщения о заказах в топик шины интеграции и ожидает ответы от различных транспортных партнеров, которые появляются в отдельной очереди шины интеграции. Ключевым вопросом является то, как приложение Orders определяет, к какому сообщению относится определенный ответ. Решение заключается в использовании корреляции сообщений.

Когда рабочая роль приложения Orders посылает сообщение с заказом транспортному партнеру, она заполняет свойство **MessageId** идентификатором заказа (этот идентификатор генерируется при создании заказа), а также она указывает имя очереди, в которой приложение Orders ожидает ответа в свойстве **ReplyTo**, как показано в следующем фрагменте кода, взятом из метода **Execute** класса **NewOrderJob**.

**C#**

```
var brokeredMessage = new BrokeredMessage(msg)
{
  MessageId = msg.OrderId.ToString(),
  ...
  ReplyTo = this.replyQueueName
};
```

> **Мнение Маркуса**
>
> Передача адреса, на который принимающее приложение должно переслать ответ с помощью свойства **ReplyTo** сообщения, позволяет отделить принимающее приложение от необходимости использования определенной жестко заданной очереди.

Транспортный партнер создает объект **OrderStatusUpdateMessage** в качестве ответа, а затем размещает это сообщение в очереди, указанной свойством **ReplyTo** исходного сообщения о заказе. В примере Trey Research эта логика реализуется в методе **SendToUpdateStatusQueue** (вызывается методом **SendOrderReceived**) в классе **OrderProcessor**. Вглаве 5 «Обработка заказов в решении Trey Research» более детально описан поток сообщений через транспортного партнера.

Рабочая роль получает ответ в указанной очереди шины интеграции. Когда получено сообщение с ответом, оно используется для обновления статуса заказа в базе данных Orders. Эта функциональность реализуется в классе **StatusUpdateJob** в рабочей роли, что также описывается в главе 5 «Обработка заказов в решении TreyResearch».

### Защита очередей сообщений, топиков и подписок

Ключевым требованием решения для обмена сообщениями является то, что все сообщения должны быть защищены от несанкционированного доступа. Как описано в главе 3 «Аутентификация пользователей в приложении Orders», шина интеграции Windows Azure использует ACS для защиты очередей, топиков и подписок шины интеграции. Чтобы подключиться к очереди, топику или подписке, приложение должно предоставить действительный маркер аутентификации.

Для защиты каналов связи компания Trey Research определила правила в ACS, позволяющие локальным и удаленным транспортным партнерам подключаться к подписке шины интеграции, в которую приложение Orders публикует сообщения о заказах; транспортные партнеры получают утверждение Listen для подписки, что позволяет им получать только сообщения. Рабочая роль в приложении Orders получает утверждение Send через топик, что позволяет ей публиковать только сообщения.

> **Примечание**
>
> Более подробная информация об этих утверждениях и о том, как настроить службу ACS для аутентификации клиентов и авторизации доступа к артефактам шины интеграции, представлена в статье «Service Bus Authentication and Authorization with the Access Control Service» на сайте http://msdn.microsoft.com/en-us/library/windowsazure/hh403962.aspx.

Для очереди шины интеграции, в которой рабочая роль ожидает ответное сообщение, сохраняются права; транспортные партнеры получают утверждение Send для очереди, а рабочая роль имеет утверждение Listen.

> **Примечание**
>
> Различные правила ACS, а также группы правил и удостоверения, используемые приложением Orders и транспортными партнерами, создаются программой установки в проекте **TreyResearch.Setup**.

Для полноты описания в следующей таблице подведен итог того, как компания Trey Research настроила службу ACS для включения аутентификации шины интеграции для приложений и служб, подключающихся к различным очередям, топикам и подпискам шины интеграции в приложении Orders.

| Артефакт шины интеграции | Параметр |
| --- | --- |
| Удостоверения службы. | AuditLogListener, Fabrikam, HeadOffice, Contoso, NewOrderJob, NewOrdersTopic, owner, StatusUpdateJob. |
| Шина интеграции по умолчанию (доверенный участник). | Имя: ServiceBus.<br>Область: http://treyresearch.servicebus.windows.net/<br>Отправитель утверждения: ACS. Тип маркера: SWT.<br>Группы правил:<br>• Группа правил по умолчанию содержит: если идентификатор имени «owner» порождает утверждения действий «Manage», «Send» и «Listen». |
| Топик шины интеграции (доверенный участник) для отправки деталей о новом заказе транспортным партнерам и в локальный журнал аудита. | Имя: NewOrdersTopic<br>Область: http://treyresearch.servicebus.windows.net/neworderstopic<br>Отправитель утверждения: ACS. Тип маркера: SWT.<br>Группы правил:<br>• Группа правил по умолчанию для ServiceBus.<br>• Группа правил содержит: если идентификатор имени «NewOrderJob» порождает утверждение действия «Send».<br>Подписки:<br>• местного (Contoso) и удаленного (Fabrikam) транспортных партнеров.<br>• Служба журнала аудита. |
| Очередь шины интеграции (доверенный участник), используемая транспортными партнерами для отправки сообщений в приложение Orders, которая:<br>• подтверждает получение деталей нового заказа,<br>• указывает, что заказ был доставлен. | Имя: OrderStatusUpdateQueue<br>Область: http://treyresearch.servicebus.windows.net/<br>orderstatusupdatequeue<br>Отправитель утверждения: ACS. Тип маркера: SWT.<br>Группы правил:<br>• Группа правил по умолчанию для шины интеграции.<br>• Группа правил содержит: если идентификатор имени «Contoso» порождает утверждение действия «Send».<br>• Группа правил содержит: если идентификатор имени «Fabrikam» порождает утверждение действия «Send».<br>• Группа правил содержит: если идентификатор имени «StatusUpdateJob» порождает утверждение действия «Listen». |
| Транспортный партнер (доверенный участник) для местных поставок (Contoso, Inc.). | Имя: Contoso<br>Область: http://treyresearch.servicebus.windows.net/<br>neworderstopic/subscriptions/contososubscription<br>Отправитель утверждения: ACS. Тип маркера: SWT.<br>Группы правил:<br>• Группа правил по умолчанию для ServiceBus.<br>• Группа правил содержит: если идентификатор имени «Contoso» порождает утверждение действия «Listen». |

| | |
|---|---|
| Транспортный партнер (доверенный участник) для удаленных поставок (Fabrikam Inc.). | Имя: Fabrikam<br>Область: http://treyresearch.servicebus.windows.net/ neworderstopic/subscriptions/fabrikamsubscription<br>Отправитель утверждения: ACS. Тип маркера: SWT.<br>Группы правил:<br>• Группа правил по умолчанию для ServiceBus.<br>• Группа правил содержит: если идентификатор имени «Fabrikam» порождает утверждение действия «Listen». |
| Локальное приложение для управления и контроля (доверенный участник). Подписчики на топик для сбора сообщений журнала аудита. | Имя: AuditLogListener<br>Область: http://treyresearch.servicebus.windows.net/ neworderstopic/subscriptions/ auditloglistenersubscription<br>Отправитель утверждения: ACS. Тип маркера: SWT.<br>Группы правил:<br>• Группа правил по умолчанию для ServiceBus.<br>• Группа правил содержит: если идентификатор имени «AuditLogListener» порождает утверждение действия «Listen». |

Рабочая роль и транспортные партнеры настроены с использованием соответствующего ключа учетной записи хранения, и они предоставляют эту информацию, когда подключаются к очереди, топику или подписке шины интеграции в виде простого веб-маркера. Например, метод **Run** в классе **NewOrderJob** в рабочей роли использует следующий код для извлечения ключа и информации об отправителе из настройки приложения и сохраняет их в объекте **ServiceBusTopicDescription**, который использовался для создания объекта **ServiceBusTopic**.

**C#**

```
public void Run()
{
  ...
this.serviceBusNamespace = CloudConfiguration.
    GetConfigurationSetting("serviceBusNamespace",
string.Empty);
this.acsNamespace = CloudConfiguration.
    GetConfigurationSetting("acsNamespace",
string.Empty);
  var topicName = CloudConfiguration.
    GetConfigurationSetting("topicName", string.Empty);
  var issuer = CloudConfiguration.
    GetConfigurationSetting("newOrdersTopicIssuer",
string.Empty);
  var defaultKey = CloudConfiguration.
    GetConfigurationSetting("newOrdersTopicKey",
string.Empty);
  ...

  var serviceBusTopicDescription =
    new ServiceBusTopicDescription
    {
```

```
Namespace = this.serviceBusNamespace,
TopicName = topicName,
      Issuer = issuer,
DefaultKey = defaultKey
    };

this.newOrderMessageSender =
new ServiceBusTopic(serviceBusTopicDescription);
  ...
}
```

Конструктор в классе **ServiceBusTopic** использует эту информацию для создания поставщика маркера для объекта **MessageFactory**. Объект **MessageFactory** используется для создания объекта **MessageSender**, который используется объектом **ServiceBusTopic** для реального размещения сообщений в соответствующий топик шины интеграции.

**C#**

```
...
private readonly ServiceBusTopicDescription description;
private readonly TokenProvidertokenProvider;
private readonly MessageSender sender;
...

public ServiceBusTopic(
  ServiceBusTopicDescription description)
{
  ...

  this.description = description;
this.tokenProvider = TokenProvider.
CreateSharedSecretTokenProvider(
this.description.Issuer,
this.description.DefaultKey);

  var runtimeUri = ServiceBusEnvironment.
CreateServiceUri("sb", this.description.Namespace,
string.Empty);
  var messagingFactory = MessagingFactory.Create(
runtimeUri, this.tokenProvider);
this.sender = messagingFactory.CreateMessageSender(
this.description.TopicName.ToLowerInvariant());
  ...
}
```

Конструкторы классов **ServiceBusQueue** и **ServiceBusSubscription** следуют аналогичной схеме.

### Защита сообщений

Чтобы предотвратить спуфинг, компания Trey Research также реализовывает механизм проверки удостоверения транспортного партнера, размещающего сообщения в очереди шины интеграции, в которой их ожидает приложение Orders. Это позволяет гарантировать, что неавторизованная сторона не выдает себя за транспортного партнера и не посылает фиктивные сообщения. Для этого каждый раз, когда транспортный партнер посылает сообщение, он добавляет простой веб-маркер в заголовок сообщения, который указывает удостоверение отправителя, а получатель в приложении Orders проверяет этот маркер при получении каждого сообщения.

Добавление маркеров в заголовок и их проверку нельзя выполнить, просто настроив артефакты шины интеграции и ACS. Вместо этого Trey Research использует следующий код для получения маркера из ACS. Этот фрагмент кода взят из класса **OrderProcessor**. Это основной класс, от которого происходят классы **Adapter** и **Connector**, используемые транспортными партнерами.

**C#**

```
private string GetToken(ServiceBusQueueDescription
                        queueDescription)
{

varrealm = string.Format("urn:{0}/{1}",
queueDescription.QueueName,
HttpUtility.UrlEncode(this.acsServiceIdentity));

  var token = GetTokenFromAcs(string.Format(
          "https://{0}.accesscontrol.windows.net/",
queueDescription.SwtAcsNamespace),
this.acsServiceIdentity, this.acsPassword, realm);

return token;
}

private string GetTokenFromAcs(string acsNamespace,
string serviceIdentity, string password,
               string relyingPartyRealm)
{
  // запрос маркера из ACS
  var client = new WebClient();
client.BaseAddress = acsNamespace;
  var values = new NameValueCollection();
values.Add("wrap_name", serviceIdentity);
values.Add("wrap_password", password);
values.Add("wrap_scope", relyingPartyRealm);
byte[] responseBytes = client.UploadValues(
          "WRAPv0.9/", "POST", values);
  string response =
Encoding.UTF8.GetString(responseBytes);
return HttpUtility.UrlDecode(
response
        .Split('&')
        .Single(value =>
value.StartsWith("wrap_access_token=",
StringComparison.OrdinalIgnoreCase))
        .Split('=')[1]);
}
```

Метод **GetTokenFromAcs** (приведенный и в предыдущем фрагменте кода) посылает запрос на получение маркера для ACS. Метод **GetToken** передает ему значения, извлеченные из файла конфигурации приложения, для имени удостоверения службы, пароля и области с целью создания соответствующего маркера для удостоверения данного отправителя.

После получения необходимого маркера транспортный партнер может добавить его в сообщение, размещенное в очереди шины интеграции.

После получения сообщения получатель может извлечь маркер и использовать его для проверки удостоверения отправителя. Например, рабочая роль использует метод **IsValidToken** в классе **StatusUpdateJob**, показанном в следующем примере кода, для проверки действительности маркера, извлеченного из сообщения.

**C#**

```
private bool IsValidToken(GuidorderId, string token)
{
  string transportPartner;
  ...

  string acsServiceNamespace = CloudConfiguration.
    GetConfigurationSetting("acsNamespace", null);
  string acsUsername = CloudConfiguration.
    GetConfigurationSetting("acsUsername", null);
  string acsPassword = CloudConfiguration.
    GetConfigurationSetting("acsUserKey", null);

  var acsWrapper = new ServiceManagementWrapper(
acsServiceNamespace, acsUsername, acsPassword);
  var relyingParty = acsWrapper.
RetrieveRelyingParties().
    SingleOrDefault(
rp =>rp.Name.Contains(transportPartner));

var keyValue = string.Empty;

  if (relyingParty != null)
  {
    var key = relyingParty.
RelyingPartyKeys.
FirstOrDefault();
      ...
keyValue = Convert.ToBase64String(key.Value);
}

  // Значение для trustedAudience:
  //  urn:[имя-очереди]/[имя-партнера]
var trustedAudience = string.Format("urn:{0}/{1}",
CloudConfiguration.GetConfigurationSetting(
      "orderStatusUpdateQueue", string.Empty),
HttpUtility.UrlEncode(transportPartner));

  var validator = new TokenValidator(
    "accesscontrol.windows.net",
RoleEnvironment.GetConfigurationSettingValue(
```

```
      "acsNamespace"),
trustedAudience, keyValue);

    return validator.Validate(token);
}
```

Метод **IsValidToken** использует классы в проекте **ACS.ServiceManagementWrapper** для извлечения информации о различных доверенных участниках, настроенных в ACS. Дополнительная информация о проекте **ACS.ServiceManagementWrapper** представлена в статье «Access Control Service Samples and Documentation» по адресу http://acs.codeplex.com/releases/view/57595.

Метод **IsValidToken** также использует отдельный класс **TokenValidator** в проекте **Orders.Workers** для проверки маркера при наличии имени узла ACS, пространства имен службы, значения аудитории и заверенного ключа.

## Отправка заказов в журнал аудита

На данный момент все заказы общей суммой более 10 000 долларов США должны подвергаться аудиту, и журнал аудита размещается локально. Логика обработки заказов должна позволять быстро определять общую стоимость заказа и направлять детали заказа в журнал аудита. Эта обработка должна происходить быстро без ущерба для скорости ответа клиентам. Она должна быть масштабируемой, так как объем заказов увеличивается, а также достаточно гибкой, чтобы позволять быстро изменять критерии аудита, опять же без необходимости переписывания кода для рабочей роли. Как и с заказами, отправленными транспортным партнерам, вся информация аудита носит исключительно конфиденциальный характер и должна быть защищена от несанкционированного доступа, особенно когда она пересекает границы сети.

### Выбор механизма отправки заказов в журнал аудита

После того как специалисты компании Trey Research решили использовать топики шины интеграции в качестве механизма взаимодействия с транспортными партнерами, они решили использовать тот же подход для сообщений об аудите. Когда клиент размещает заказ, его общая сумма рассчитывается и добавляется в виде свойства **OrderAmount** в сообщение. Если общая сумма составляет более 10 000 долларов США, заказ передается в подписку аудита и отправляется локальному приложению для аудита в Trey Research. На рисунке 5 показано, как эта технология вписывается в решение Trey Research. Примите к сведению, что журнал аудита использует тот же топик шины интеграции, что и транспортные партнеры, но с подпиской, которая применяет другой фильтр.

> **Примечание**
>
> Помните, что если сообщение, помещенное в топик, удовлетворяет фильтру, связанному с более чем одной подпиской, копия сообщения будет направлена во все соответствующие подписки.

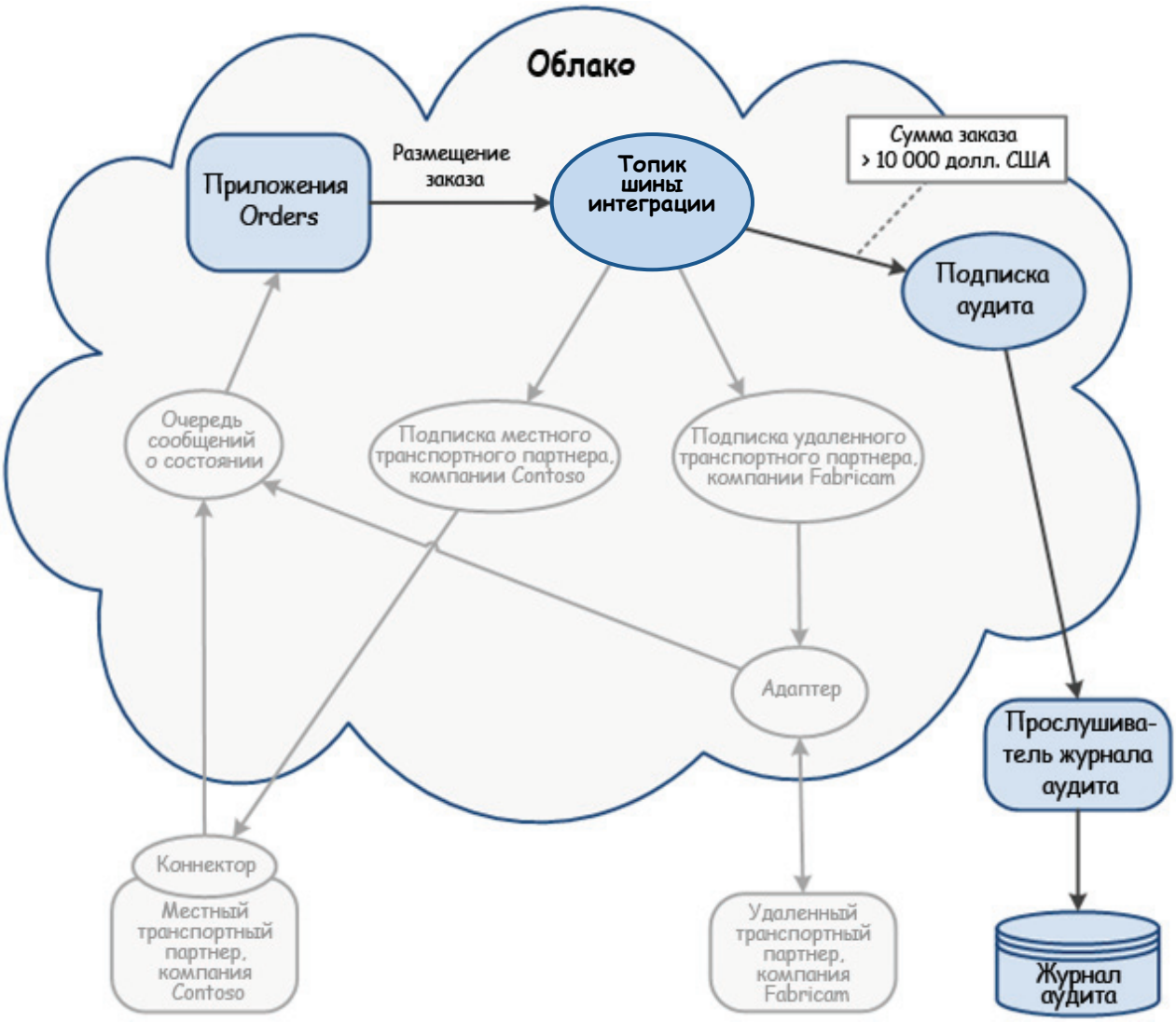



**Рисунок 5**

**Технология обмена сообщениями, используемая в Trey Research для направления заказов в журнал аудита**

## Как Trey Research посылает заказы в журнал аудита

Общая сумма заказа добавляется как свойство **OrderAmount** к каждому сообщению о заказе, размещенному рабочей ролью в топике шины интеграции. Приложение Trey Research определяет все заказы, требующие аудита путем создания подписки шины интеграции с соответствующим фильтром. Код, создающий эту подписку шины интеграции, находится в методе **SetupAuditLogListener** в программе установки проекта **TreyResearch.Setup**. В следующем фрагменте кода показаны части этого метода, которые настраивают фильтр.

**C#**

```
private static void SetupAuditLogListener()
{
  var formattedName = AuditLogListener.Replace(" ",
string.Empty).ToLowerInvariant();
  ...
  var serviceBusTopicDescription =
    new ServiceBusSubscriptionDescription
    {
Namespace = ServiceBusNamespace,
TopicName = TopicName,
SubscriptionName = string.Format(
        "{0}Subscription", formattedName),
      Issuer = Issuer,
DefaultKey = DefaultKey
    };

  var serviceBusSubscription =
    new
ServiceBusSubscription(serviceBusTopicDescription);
  const int AuditAmount = 10000;
  var filterExpression = string.Format(
    "OrderAmount > {0}", AuditAmount);
serviceBusSubscription.CreateIfNotExists(
filterExpression);

  ...
}
```

Веб-приложение Trey Research Head Office в проекте **HeadOffice** включает класс **AuditController**, который подключается к этой подписке и извлекает заказы для аудита. Метод **DownloadLogs** в этом классе содержит код, который реально извлекает детали заказов для аудита. Обратите внимание, что этот метод подключается к топику шины интеграции в каждом центре данных, где запущено приложение Orders; каждый экземпляр приложения Orders публикует сообщения в топик в соответствующем локальном центре данных. Имя используемой подписки, имя топика и ключи безопасности хранятся в файле конфигурации приложения.

**C#**

```
public ActionResult DownloadLogs()
{
  ...
  var serviceBusNamespaces = WebConfigurationManager.

AppSettings["AuditServiceBusList"].Split(',').ToList();
...
```

```
foreach (var serviceBusNamespace in
serviceBusNamespaces)
{
    // Подключение в шине интеграции, загрузка сообщений
из
    // подписки журнала аудита, сохранение в базе данных.
    var serviceBusTopicDescription =
      new ServiceBusSubscriptionDescription
      {
Namespace = serviceBusNamespace,
TopicName = WebConfigurationManager.
          AppSettings["topicName"],
SubscriptionName = WebConfigurationManager.
          AppSettings["subscriptionName"],
        Issuer = WebConfigurationManager.
          AppSettings["issuer"],
DefaultKey = WebConfigurationManager.
          AppSettings["defaultKey"]
      };

    var serviceBusSubscription = new

ServiceBusSubscription(serviceBusTopicDescription);

    // MessagePollingInterval должен быть настроен
с учетом
    // переменных, таких как мощность ЦП,
    // ожидаемый объем заказов, которые следует
    // обработать, и количество экземпляров рабочей роли
    var receiverHandler =
      new ServiceBusReceiverHandler<NewOrderMessage>(
serviceBusSubscription.GetReceiver()) {
          MessagePollingInterval =
           TimeSpan.FromSeconds(2) };

receiverHandler.ProcessMessages(
      (message, queueDescription, token) =>
      {
        return Task.Factory.StartNew(
          () => this.ProcessMessage(
            message, queueDescription),
          ...);
      },
      ...);
  }

  return RedirectToAction("Index");
}
```

Метод **ProcessMessage** (вызываемый методом **ProcessMessages** объекта **ServiceBusReceiverHandler**) просто сохраняет данные сообщений о заказах в локальной базе данных SQL Server.

**C#**

```
public void ProcessMessage(NewOrderMessage message,
ServiceBusQueueDescription queueDescription)
{
  // Сохранение журнала аудита в базе данных
  var auditLog = new AuditLog
  {
OrderId = message.OrderId,
OrderDate = message.OrderDate,
    Amount = Convert.ToDecimal(message.Amount),
CustomerName = message.CustomerName
  };

this.auditLogStore.Save(auditLog);
}
```

## Проверка заказов на соответствие нормативным требованиям

Последней проблемой является интеграция с приложением для обеспечения соответствия нормативным требованиям. Это приложение анализирует заказы на соответствие ограничениям экспорта и правительственным постановлениям для технических продуктов. Приложение для обеспечения соответствия нормативным требованиям взаимодействует с базой данных Orders с помощью стандартной строки подключения SQL Server и выполняет запросы для определения соответствия согласно заранее установленному графику. Кроме того, приложение для обеспечения соответствия нормативным требованиям генерирует отчеты, которые хранятся локально в надежном месте в головном офисе компании Trey Research.

Если приложение развернуто локально, оно получает доступ к базе данных Orders и защищенному месту хранения отчетов, размещенных тоже локально. Теперь, когда база данных Orders размещена в облаке, приложение для обеспечения соответствия нормативным требованиям должно подключиться к экземпляру SQL Azure. Эта проблема настройки может быть легко решена. Однако объем трафика между приложением для обеспечения соответствия нормативным требованиям и базой данной Orders достаточно большой, так как приложение для обеспечения соответствия нормативным требованиям выполняет многочисленные запросы и поиски данных. Эти факторы заставили компанию Trey Research уделить больше внимания тому, как следует разворачивать само приложение.

Исходный код этого приложения недоступен, так как является конфиденциальным; правительственный департамент определяет процессы, которым необходимо следовать, и сертифицирует операции. Поэтому практически невозможно модифицировать приложение в качестве рабочей роли. Кроме того, функциональность по созданию отчетов требует аутентифицированного подключения к соответствующему серверу, а все данные, передаваемые через это подключение, должны быть защищены.

## Выбор места для размещения приложения с целью обеспечения соответствия нормативным требованиям

Для размещения приложения с целью обеспечения соответствия нормативным требованиям, специалисты компании Trey Research решили установить и настроить приложение с помощью роли виртуальной машины Windows Azure. Это решение балансирует необходимость настройки, развертывания и поддержки роли виртуальной машины в облаке, рядом с анализируемыми данными заказов, в отличие от локального размещения приложения для обеспечения соответствия нормативным требованиям, при этом либо осуществляется подключение к данным о заказах в облаке, либо данные передаются из облака в локальную базу данных.

> **Мнение Джаны**
>
> Принимая решение о том, развертывать ли приложение в роли виртуальной машины, необходимо учитывать преимущества снижения нагрузки на сеть со стороны такого активного приложения, как приложение для обеспечения соответствия нормативным требованиям, при подключении к базе данных в облаке по сравнению с расходами на поддержку и управление ролью виртуальной машины.

Приложение для обеспечения соответствия нормативным требованиям требует доступа к защищенной области, где оно хранит различные сгенерированные отчеты. Эта область должна находиться на локальном сервере, и компания Trey Research решила использовать компонент Windows Azure Connect, чтобы обеспечить аутентифицированную безопасную связь по виртуальной сети между ролью виртуальной машины и этим сервером.

Компания Trey Research решила развернуть роль виртуальной машины в центре данных на севере США, так как это самый близкий центр данных к головному офису, надеясь минимизировать задержки сети, которые могут возникать при связи локальных и облачных компонентов.

## Как компания Trey Research разместила приложение для обеспечения соответствия нормативным требованиям

> Этот раздел носит исключительно информативный характер, демонстрируя способы реализации решения. Пример приложения компании Trey Research на самом деле не включает приложение для обеспечения соответствия нормативным требованиям и соответствующую роль виртуальной машины.

Роль виртуальной машины, в которой размещается приложение для обеспечения соответствия нормативным требованиям, анализирует данные в базе данных SQL Azure Orders. Роль виртуальной машины развернута в центре данных на севере США, но приложение для обеспечения соответствия нормативным требованиям генерирует отчеты, которые хранятся локально в безопасном месте в инфраструктуре головного офиса компании Trey Research. Приложение для обеспечения соответствия нормативным требованиям также посылает данные в приложение мониторинга, расположенное локально; это приложение обеспечивает серию интерфейсов Distributed Component Object Model (DCOM), к которым при необходимости подключается приложение для обеспечения соответствия нормативным требованиям.

Компания Trey Research реализовала отдельный небольшой домен с собственной службой доменных имен (DNS) в локальной инфраструктуре специально для размещения решения Windows Azure Connect Endpoint, данных отчетов и приложения мониторинга. Отчеты хранятся в сетевом ресурсе, защищенном с помощью списка управления доступом (Access Control List, ACL). Доступ предоставляется учетной записи, определенной в домене. Приложение для обеспечения соответствия нормативным

требованиям, подключенное к домену, предоставляет эти учетные данные при записи отчетов в сетевом ресурсе с общим доступом. Такой же подход используется для защиты интерфейса DCOM, предоставляемого приложением мониторинга.

Этот домен имеет отношение доверия с основным доменом компании Trey Research, а приложение для управления, работающее в основном домене, может периодически извлекать данные отчетов и анализировать информацию, внесенную в журнал приложением для мониторинга. На рисунке 6 показана структура системы обеспечения соответствия нормативным требованиям.

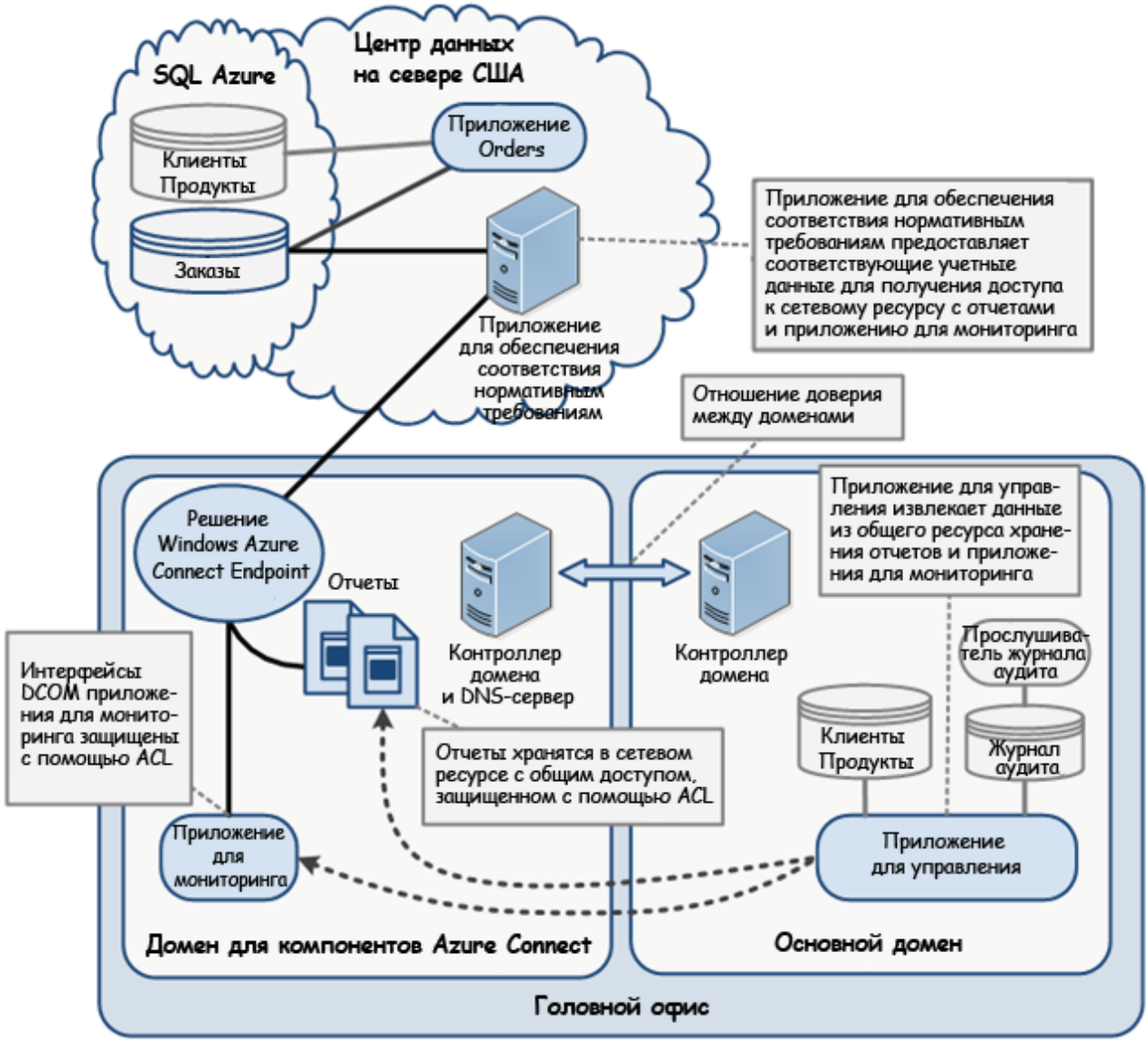



**Рисунок 6**

**Структура системы обеспечения соответствия нормативным требованиям**

## Выводы

В этой главе рассмотрено, как компания Trey Research использовала две важные технологии Windows Azure для реализации надежного механизма взаимодействия на основании топиков, подписок и очередей шины интеграции. Эти технологии представляют собой основу для создания элегантного гибридного решения, состоящего из компонентов, которые должны обеспечить взаимодействие между локальными и облачными ресурсами.

Очереди шины интеграции позволяют реализовать асинхронную систему обмена сообщениями, которая помогает устранить временную зависимость между клиентским приложением, размещающим запрос, и службой, получающей запрос. Ориентированные на сообщения приложения идеально подходят для использования в облачной среде, так как они помогают справиться с изменяемыми объемами и пиковой нагрузкой, характерной для многих коммерческих систем, а также могут быть достаточно надежными для обработки сетевых проблем и сбоев связи. Используя очереди шины интеграции, вы можете реализовать большое количество общих шаблонов системы обмена сообщениями и адаптировать их к требованиям своей системы.

Топики и подписки шины интеграции позволяют интеллектуально перенаправлять сообщения к службам. Приложение может публиковать сообщения в топик и включать метаданные, которые могут использоваться фильтром для определения того, через какие подписки должны проходить сообщения. Службы, прослушивающие эти подписки, получат все соответствующие сообщения. Этот простой, но надежный механизм позволяет обрабатывать разнообразные сценарии и легко создавать элегантные решения для этих сценариев.

И наконец, компонент Windows Azure Connect позволяет установить виртуальное сетевое соединение между ролью, размещенной в облаке, и вашей локальной инфраструктурой, что идеально подходит для ситуаций, где необходимо установить непосредственное соединение между компонентами вместо ориентированного на сообщения интерфейса. Вы можете получить совместный доступ к данным такой сети так же, как и к ресурсам, совместно используемым несколькими компьютерами в локальной инфраструктуре.

## Дополнительная информация

- «An Introduction to Service Bus Queues»
  http://blogs.msdn.com/b/appfabric/archive/2011/05/17/an-introduction-to-service-bus-queues.aspx
- «Windows Azure Queues and Windows Azure Service Bus Queues - Compared and Contrasted»
  http://msdn.microsoft.com/en-us/library/windowsazure/hh767287(v=vs.103).aspx
- «Building loosely-coupled apps with Windows Azure Service Bus Topics and Queues»
  http://channel9.msdn.com/Events/BUILD/BUILD2011/SAC-862T
- «Best Practices for Leveraging Windows Azure Service Bus Brokered Messaging API»
  http://windowsazurecat.com/2011/09/best-practices-leveraging-windows-azure-service-bus-brokered-messaging-api/
- «Best Practices for Performance Improvements Using Service Bus Brokered Messaging»
  http://msdn.microsoft.com/en-us/library/windowsazure/hh528527.aspx
- «Queues, Topics, and Subscriptions»
  http://msdn.microsoft.com/en-us/library/windowsazure/hh367516.aspx
- «Service Bus Authentication and Authorization with the Access Control Service»
  http://msdn.microsoft.com/en-us/library/windowsazure/hh403962.aspx
- «Using Service Bus Topics and Subscriptions with WCF»
  http://blogs.msdn.com/b/tomholl/archive/2011/10/09/using-service-bus-topics-and-subscriptions-with-wcf.aspx
- «How to Simplify & Scale Inter-Role Communication Using Windows Azure Service Bus»
  http://windowsazurecat.com/2011/08/how-to-simplify-scale-inter-role-communication-using-windows-azure-service-bus/.
- «Service Bus REST API Reference»
  http://msdn.microsoft.com/en-us/library/windowsazure/hh780717.aspx
- «Overview of Windows Azure Connect»
  http://msdn.microsoft.com/en-us/library/gg432997.aspx

# Глава 5. Обработка заказов в решении Trey Research

После организации асинхронной передачи сообщений для взаимодействия между приложением Orders и транспортными партнерами на основе топиков, подписок и очередей шины интеграции, разработчики компании Trey Research смогли уделить больше внимания реализации бизнес-логики для обработки заказов. Так как это является основной функцией приложения Orders, работающего на технологической платформе Windows Azure, то первоочередной задачей компании было создание надежного механизма, обеспечивающего правильное размещение заказа.

В этой главе описывается, как компания Trey Research разрабатывала логику обработки заказов и создания отчетов, которая позволила бы не только воспользоваться преимуществами платформы обмена сообщениями, но и обеспечить надежное и правильное управление заказами клиентов.

## Сценарий и контекст

Компания Trey Research использует службы внешних транспортных партнеров для отправки заказов клиентам. Транспортные партнеры могут внедрять свои собственные системы, а также использовать различные процедуры обработки заказов. Их основной задачей является получение товаров на производственном предприятии Trey Research и их доставка клиентам, как только компания предоставит данные заказа и адрес доставки.

По причинам, описанным в главе 4 «Реализация надежного обмена сообщениями и информацией в облаке», компания Trey Research приняла решение использовать топики, подписки и очереди шины интеграции для связи с транспортными партнерами. Используя такую инфраструктуру, приложение Orders сохраняет данные нового заказа в локальной базе данных, работающей на технологической платформе SQL Azure. Затем приложение отправляет данные в топик шины интеграции и транспортным партнерам, каждый из которых использует собственную подписку шины интеграции для поиска и отправки заказов (в каждой подписке есть фильтр, который обеспечивает передачу транспортному партнеру только тех заказов, которые он должен отправить, а также исключает доступ транспортных партнеров к заказам других партнеров). Каждый транспортный партнер отвечает одним или несколькими сообщениями о текущем состоянии процесса поставки; эти сообщения размещаются в очереди шины интеграции. Приложение Orders получает эти сообщения и использует их для обновления состояния заказов в локальной базе данных SQL Azure. Клиенты, в свою очередь, могут использовать приложение для проверки состояния своих заказов.

На рисунке 1 показан логический поток сообщений и данных между приложением Orders и транспортными партнерами. Необходимо отметить, что несмотря на то, что приложение Orders всегда отсылает транспортному партнеру одно сообщение для каждого нового заказа, ответное сообщение или сообщения транспортного партнера зависят исключительно от внутренней системы, используемой партнером; а приложение Orders просто сохраняет данные каждого ответного сообщения в базе данных SQL Azure в виде сообщения о состоянии. Этот вариант используется в примере решения Trey Research. Местный транспортный партнер Contoso отправляет два сообщения: первое сообщение подтверждает получение заказа и содержит код трассировки, второе сообщение посылается после отправки заказа. А удаленный транспортный партнер Fabrikam отправляет только одно ответное сообщение с подтверждением получения заказа и с кодом трассировки.

**Примечание**

Многие коммерческие партнеры предоставляют собственные веб-приложения, которые клиенты могут использовать для контроля состояния доставки заказа, предоставив код трассировки. Эта функциональность не входит в решение Trey Research.

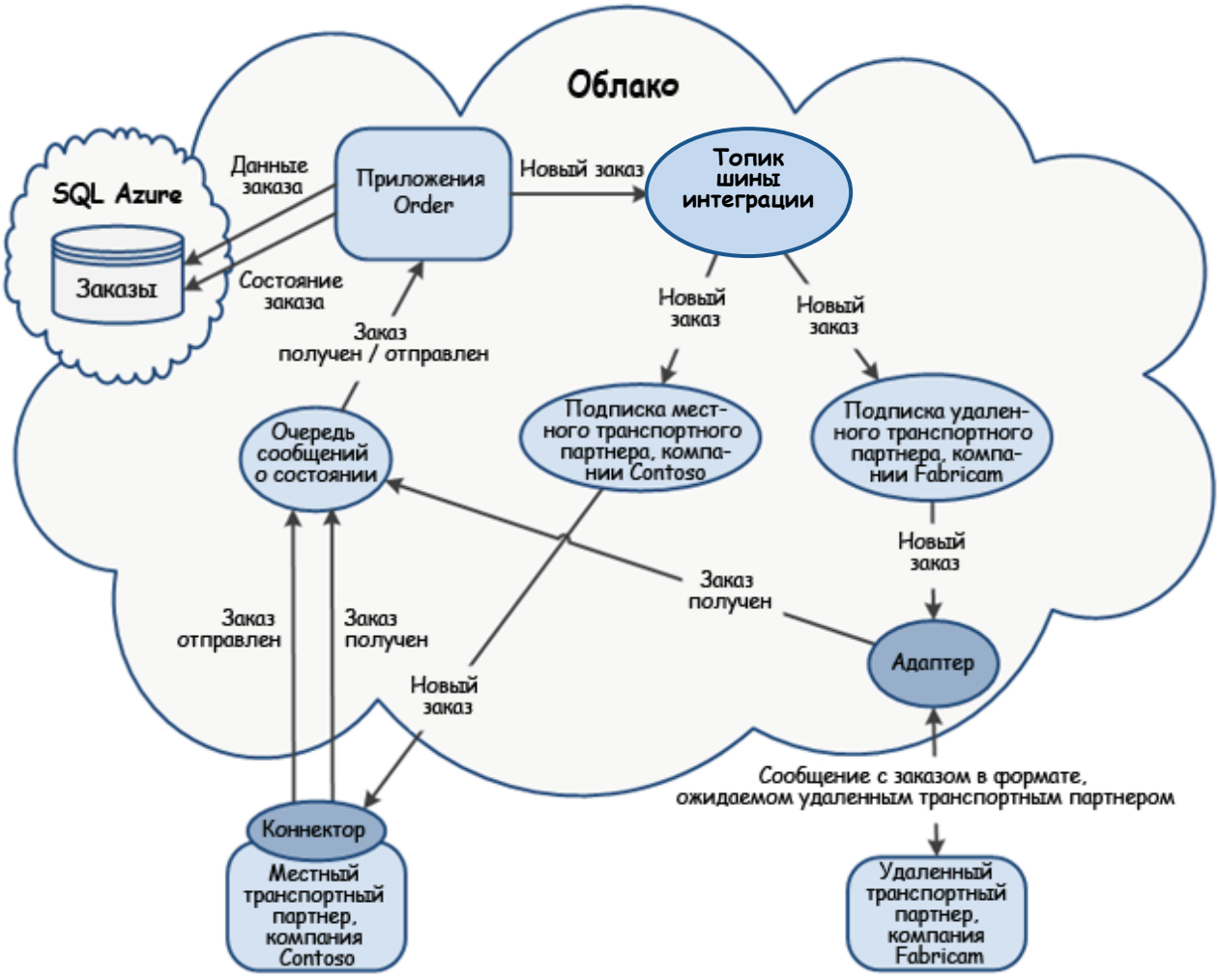



**Рисунок 1**

**Логический поток сообщений и данных при размещении заказа клиентом**

**Примечание**

Заказы на сумму 10 000 долл. США и более подлежат аудиту. Они извлекаются из топика шины интеграции с использованием отдельной подписки шины интеграции и направляются в журнал аудита. Реализация этой части обработки заказа описана в разделе «Как Trey Research посылает заказы в журнал аудита» главы 4 «Реализация надежного обмена сообщениями и информацией в облаке». Кроме того, каждый заказ проверяется на соответствие нормам и правилам экспорта отдельным приложением для обеспечения соответствия нормативным требованиям. В разделе главы 4 «Как компания Trey Research разместила приложение для обеспечения соответствия нормативным требованиям» описывается, как компания настроила решение для поддержки этих требований.

Ни один из этих двух аспектов процесса обработки заказа не описывается в этой главе.

## Обработка заказов и взаимодействие с транспортными партнерами

Несмотря на кажущуюся теоретическую простоту решения, при реализации процесса обработки заказов компания Trey Research столкнулась с рядом технических и логистических проблем в двух основных областях.

- Процесс должен быть надежным и масштабируемым.

  Как только клиент поместил заказ, он должен быть обработан и не должен быть утерян. Топики и очереди шины интеграции обеспечивают надежный механизм маршрутизации и получения сообщений. Однажды размещенное сообщение уже не будет потеряно. Однако размещение сообщений подразумевает подключение к топику или очереди через Интернет, а это может служить причиной ошибок. Процесс обработки должен гарантировать успешное размещение заказов.

  Кроме того, объем заказов может значительно изменяться со временем. Поэтому бизнес-логика процесса обработки заказов должна быть масштабируемой, что позволит быстро размещать сообщения при минимально возможном уровне использования ресурсов.

- Процесс обработки заказов должен быть отделен от внутренней логики, количества и местоположения транспортных партнеров, а также от расположения центра обработки, в котором работает приложение Orders.

  Внутренние системы транспортных партнеров могут не иметь возможности подключаться непосредственно к подпискам и очередям шины интеграции, предоставляемым компанией Trey Research. Кроме того, администраторы транспортных партнеров не всегда хотят устанавливать код компании Trey Research на своих системах.

  К тому же, со временем компания Trey Research может работать со службами различных транспортных партнеров. Таким образом, решение должно быть достаточно гибким и позволять быстро и легко регистрировать транспортных партнеров в системе, и так же легко и быстро удалять их.

  И наконец, приложение Orders может быть установлено в нескольких центрах обработки данных, в каждом из которых размещены собственные топики, подписки и очереди шин интеграции. Каждый транспортный партнер должен быть готов принимать заказы от подписки любого из центров данных, а также размещать ответные сообщения в очереди соответствующего центра.

В следующем разделе описывается, как компании Trey Research удалось решить эти проблемы в своей реализации процесса обработки заказов.

## Как Trey Research размещает сообщения в топике с высокой надежностью

Когда клиент размещает заказ в приложении Orders, веб-роль сохраняет этот заказ в базе данных. Затем рабочая роль получает заказ, отправляет данные о доставке транспортному партнеру в виде сообщения через топик шины интеграции, и обновляет состояние заказа на отправленный. Важно, чтобы операции отправки и обновления были успешными, а также были реализованы соответствующие функции восстановления и уведомления в случае неуспешного выполнения одной из этих операций. По существу, это является определением транзакции. Однако на момент написания этой работы, обмен сообщениями шины интеграции поддерживал транзакционное поведение только в пределах фреймворка шины интеграции. На данный момент шина интеграции Windows Azure не поддерживает использование координатора распределенных транзакций (Microsoft), поэтому невозможно совмещать операции базы данных SQL Azure с операциями шины интеграции по отправке и получению сообщений в пределах одной транзакции.

Поэтому компания Trey Research решила внедрить собственный механизм. Его целью является отслеживание состояния каждой операции отправки, размещающей сообщения заказа в топике шины интеграции, а также обновление текущего состояния заказа. Этот механизм реализует псевдотранзакцию. Вероятно, процесс кажется более сложным, чем требуется, но он может быть более эффективным при подсчете неустойчивых неисправностей (например, вызванных периодическими проблемами в соединении с топиком шины интеграции), и при использовании операций асинхронной передачи сообщений. Такой подход также предусматривает широкие возможности для расширения; он наиболее применим при необходимости выполнения большого количества операций или в случае сложных взаимосвязей между операциями.

**Мнение Маркуса**

Очень важно написать код передачи сообщений таким образом, чтобы он мог реагировать на сбои, возникающие на любом этапе цикла передачи сообщений, а не только в случае неуспешной отправки или получения сообщения.

Фреймворки, компоненты и документация, которые помогут вам реализовать логику для обмена сообщениями в Windows Azure, представлены в виде ресурса «Best Practices for Leveraging Windows Azure Service Bus Brokered Messaging API» на сайте http://windowsazurecat.com/2011/09/best-practices-leveraging-windows-azure-service-bus-brokered-messaging-api/.

На рисунке 2 представлена общая схема решения Trey Research, обеспечивающего запись и хранение заказов клиента в базе данных, а также успешную отправку запроса на поставку транспортному партнеру и в журнал аудита.

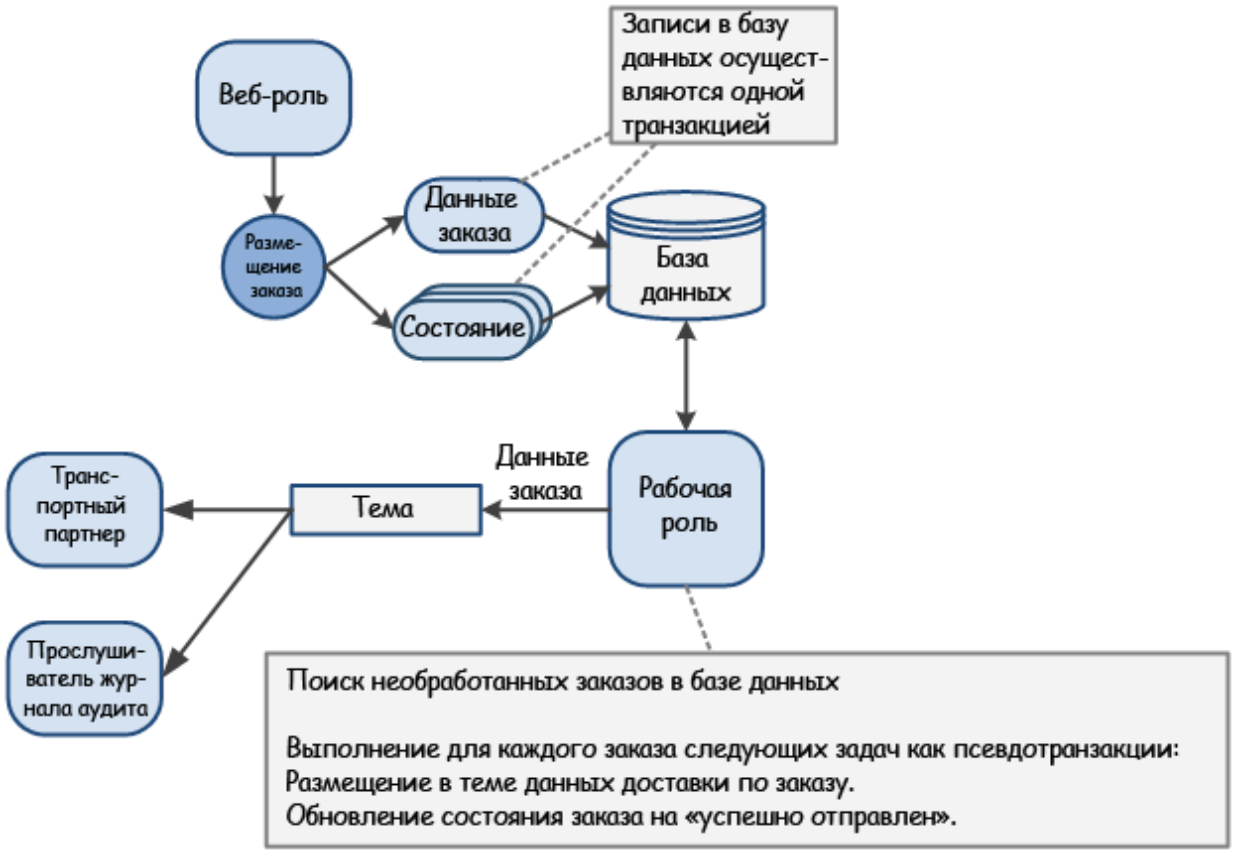



**Рисунок 2**

### Пользовательский механизм реализации транзакций и повторных попыток, реализованный для приложения Orders компании Trey Research

В реализации используются отдельные таблицы базы данных для хранения данных заказов и текущего состояния каждого из заказов. Когда клиент размещает заказ, веб-роль заполняет эти таблицы, используя транзакцию базы данных, чтобы обеспечить успешную запись. В случае возникновения ошибки, веб-роль уведомляет администратора.

Такой подход отделяет задачу сохранения заказа от задач по его обработке, которые выполняются рабочей ролью, что освобождает веб-роль для обработки других запросов. Это также означает, что веб-роль сможет сразу же отображать клиенту такую информацию, как номер заказа, а клиент, в свою очередь, сможет проверять текущее состояние заказа, не ожидая окончания выполнения задачи по его обработке.

**Мнение Маркуса**

Если вам необходима сложная обработка сообщений перед размещением их в очереди или обработка большого количества сообщений с возможным их объединением в одно, то стоит рассмотреть вариант использования веб-роли для этих целей, а потом сохранять результирующее сообщение в базе данных. Такой подход поможет улучшить производительность и сократить риск сбоев для рабочей роли, но может иметь и негативное влияние на скорость ответа веб-роли. Это также разделяет логику задачи обработки заказа для веб-ролей и рабочих ролей с последующим сокращением разделения полномочий.

После этого рабочая роль выполняет задачи, которые требуются для завершения процесса. Она отсылает каждое сообщение в топик шины интеграции, который передает их транспортному партнеру, чтобы уведомить его об ожидаемой поставке, а также в журнал аудита, если сумма заказа превышает определенное значение.

Рабочая роль также прослушивает ответные сообщения от транспортного партера о том, что сообщение было получено. Ответное сообщение должно содержать номер для отслеживания поставки, сохраняемый рабочей ролью в базе данных (эта часть процесса не является частью настраиваемого механизма повторной попытки и не обозначена на рисунке 2).

На каждом этапе процесса рабочая роль обновляет состояние заказа в базе данных, чтобы отслеживать ход выполнения процесса. Например, данные состояния могут содержать информацию о том, что заказ был принят и отправлен транспортному партнеру, но подтверждение еще не было получено.

Как правило, строки состояния также содержат количество попыток активации процесса отправки для каждой позиции. Таким образом, при превышении заданного количества попыток рабочая роль может прекратить процесс и вызвать исключение или же отправить сообщение в очередь недоставленных сообщений для вмешательства оператора.

Следующие разделы содержат более детальное описание процесса разработки и реализации данного механизма компанией Trey Research с использованием SQL Azure и слоя передачи сообщений, описанного в главе 4 «Реализация надежного обмена сообщениями и информацией в облаке».

> В примере приложения Trey Research, который можно загрузить для работы с этим руководством, реализованы многие описываемые здесь технологии и методы. Однако, чтобы упростить установку и настройку и уменьшить количество предварительных условий и требований, необходимых для создания полноценных учетных записей Windows Azure, набор функций и некоторые детали реализации отличаются от текста этого руководства.

### Запись данных заказа

Клиент размещает заказ с помощью функции Checkout в веб-приложении Orders, реализованной в проекте **Orders.Website**. Эта операция инициирует обработчик действия **AddressAndPayment** в классе **CheckoutController**. Метод **AddressAndPayment** создает объект **Order**, используя введенные клиентом данные, и добавляет заказ в базу данных SQL Azure, используя платформу Entity Framework (EF). Ниже приведены соответствующие фрагменты кода метода **AddressAndPayment**. Обратите внимание на то, что данные о клиенте и адресные данные добавляются в объект **Order** с использованием метода **TryUpdateModel**.

**C#**

```
public ActionResult AddressAndPayment(
  FormCollection values)
{
  var order = new Order();
  this.TryUpdateModel(order);

  var identity = User.Identity as IClaimsIdentity;
  var userName = identity.GetFederatedUsername();

  var cartId = ShoppingCart.GetCartId(this.HttpContext);
  var cartItems = this.cartStore.FindCartItems(cartId);

  order.OrderDetails = cartItems.Select(
    i => new OrderDetail
      { ProductId = i.ProductId, Quantity = i.Count,
        Product = i.Product });
  order.UserName = userName;
  order.OrderDate = DateTime.Now;

  //Сохранение заказа
  this.ordersStore.Add(order);

  ...
}
```

Для каждого созданного заказа, метод **Add** базовой модели сущности также создает коллекцию объектов **OrderDetail** и заполняет их информацией о каждой из позиций заказа. Информация, описывающая заказ (имя, адрес, контактная информация клиента, дата размещения заказа, общая стоимость заказа и уникальный код заказа) сохраняется в таблице **Order** в базе данных SQL Azure. Строки позиций, составляющих заказ (продукт и требуемое количество), сохраняются в таблицу **OrderDetail**. Следующий пример кода иллюстрирует метод **Add**.

**C#**

```
public void Add(Order order)
{
  ...
  var orderId = Guid.NewGuid();

  var orderToSave = new Entities.Order
    {
      OrderId = orderId,
      UserName = order.UserName,
      OrderDate = order.OrderDate,
      Address = order.Address,
      City = order.City,
      State = order.State,
      PostalCode = order.PostalCode,
      Country = order.Country,
      Phone = order.Phone,
      Email = order.Email,
      Total = order.OrderDetails.Sum(
              d => d.Quantity * d.Product.Price)
    };

  using (var database
         = TreyResearchModelFactory.CreateContext())
  {
    database.Orders.AddObject(orderToSave);

    foreach (var orderDetail in order.OrderDetails)
    {
      var detailToSave = new OrderDetail
        {
          ProductId = orderDetail.ProductId,
          OrderId = orderId,
          Quantity = orderDetail.Quantity
        };
      database.OrderDetails.AddObject(detailToSave);
    }
    ...
  }
}
```

Обработка заказа требует от приложения Orders контроля над его состоянием. Приложение Orders записывает данные в две таблицы: **OrderStatus** и **OrderProcessStatus**. Эти строки описывают текущее состояние обработки заказа и последнюю операцию, выполненную кодом обработки заказа. После создания объектов **Order** и **OrderDetail**, метод **Add** создает строки в таблицах **OrderStatus** и **OrderProcessStatus**.

**C#**

```
...
var status = new Entities.OrderStatus
  {
    OrderId = orderId,
    Status = "TreyResearch: Order placed",
    Timestamp = DateTime.UtcNow
  };
database.OrderStatus.AddObject(status);

var orderProcess = new OrderProcessStatus
  {
    OrderId = orderId,
    ProcessStatus = "pending process"
  };
database.OrderProcessStatus.AddObject(orderProcess);
...
```

И наконец, метод **Add** сохраняет все изменения в таблицы, вызывая метод модели данных **SaveChanges**. Обратите внимание на то, что это выполняется путем вызова метода **ExecuteAction** объекта, определенного в свойстве **sqlCommandRetryPolicy** класса.

**C#**

```
...
this.sqlCommandRetryPolicy.ExecuteAction(
  () =>database.SaveChanges());
order.OrderId = orderId;
...
```

**Примечание**

Для доступа к базе данных SQL Azure свойство **sqlCommandRetryPolicy** реализует пример политики использования блока приложения для обработки неустойчивых неисправностей. Больше информации и примеров приведено в разделе «Получение и хранение данных клиента» главы 3 «Аутентификация пользователей в приложении Orders». Дополнительная информация также представлена на сайте http://msdn.microsoft.com/en-us/library/hh680934(v=pandp.50).aspx в статье «The Transient Fault Handling Application Block».

На данной стадии заказ готов к отправке на обработку.

### Отправка заказа в топик шины интеграции из приложения Orders

Логика обработки заказа инициируется рабочей ролью в проекте **Orders.Workers**. Рабочая роль использует два класса, которые инкапсулируют логику отправки заказов транспортным партнерам, а также логику получения сообщений о состоянии заказа и сообщений с подтверждением от транспортных партнеров. Эти два класса, **NewOrderJob** и **StatusUpdateJob**, называются «обработчиками заданий». Конструктор класса **WorkerRole** (определенного в проекте **Orders.Workers**) выполняет метод **CreateJobProcessors** для создания экземпляров этих объектов.

**C#**

```
public class WorkerRole :RoleEntryPoint
{
  private readonly IEnumerable<IJob> jobs;
...

  public WorkerRole()
  {
    ...
    this.jobs = this.CreateJobProcessors();
  }

  ...

  private IEnumerable<IJob>CreateJobProcessors()
  {
    return new IJob[]
    {
      new NewOrderJob(),
      new StatusUpdateJob(),
    };
  }
}
```

Эти обработчики заданий реализуют интерфейс **IJob** (определенный в папке Jobs проекта **Orders.Workers**). Данный интерфейс определяет методы для запуска и остановки длительно выполняемых заданий.

**C#**

```
public interfaceIJob
{
  void Run();
  void Stop();
}
```

Создав обработчики заданий, метод **Run** в рабочей роли запускает каждый из них, используя отдельную задачу **Task**. Рабочая роль следит за состоянием каждой задачи, опрашивая их каждые 30 секунд, и, в случае необходимости, перезапускает обработчики заданий, в которых произошли сбои. Следующий пример кода показывает метод **Run** рабочей роли.

**C#**

```
public class WorkerRole :RoleEntryPoint
{
  private readonly IEnumerable<IJob> jobs;
  private readonly List<Task> tasks;
private bool keepRunning;

  ...

  public override void Run()
  {
this.keepRunning = true;

    // Запуск заданий
    foreach (var job in this.jobs)
{
      var t = Task.Factory.StartNew(job.Run);
this.tasks.Add(t);
    }

    // Управление и перезапуск задания, в котором
произошел сбой
while (this.keepRunning)
    {
      for (int i = 0; i <this.tasks.Count; i++)
{
        var task = this.tasks[i];
if (task.IsFaulted)
        {
          // Отслеживание необработанного исключения
          if (task.Exception != null)
          {
TraceHelper.TraceError(
              "Jobthrew an exception: " +
task.Exception.InnerException.Message);
          }
          else
          {
TraceHelper.TraceError(
              "Job Failed and no exception thrown.");
}

          var jobToRestart = this.jobs.ElementAt(i);
this.tasks[i] =
Task.Factory.StartNew(jobToRestart.Run);
}
      }

      Thread.Sleep(TimeSpan.FromSeconds(30));
    }
  }
}
```

Следующий раздел описывает, как класс **NewOrderJob** отправляет заказы транспортным партнерам. Класс **UpdateStatusJob** описан далее в этой главе в разделе «Получение сообщений подтверждения и состояния в приложении Orders».

### *Класс NewOrderJob*

Рабочая роль использует обработчик заданий **NewOrderJob**, чтобы отправлять запросы в базу данных о новых заказах и отсылать их соответствующему транспортному партнеру.

Класс **NewOrderJob** реализует надежный механизм отправки сообщений в топик шины интеграции. Он проверяет, была ли операция отправки успешной, и, в случае необходимости, может повторить операцию отправки через некоторое время. И только в том случае, если сообщение было отправлено успешно, состояние заказа обновляется в базе данных.

**Примечание**

Механизм, реализуемый классом **NewOrderJob**, является упрощенной адаптацией шаблона «Планировщик – Агент – Контролер» (Scheduler-Agent-Supervisor), предложенного Клеменсом Вастерсом. Более подробно об этом рассказывается в статье «Cloud Architecture: The Scheduler-Agent-Supervisor Pattern» по адресу http://vasters.com/clemensv/CommentView,guid,83f937f7-b838-43d0-ad61-74605eceafa2.aspx.

На рисунке 3 изображена обобщенная схема задач, выполняемых классом **NewOrderJob**, а также дополнительные классы, которые он использует. Классы **ServiceBusTopic и ServiceBusTopicDescription** описаны в главе 4 «Реализация надежного обмена сообщениями и информацией в облаке», все остальные классы описаны далее в этой главе.

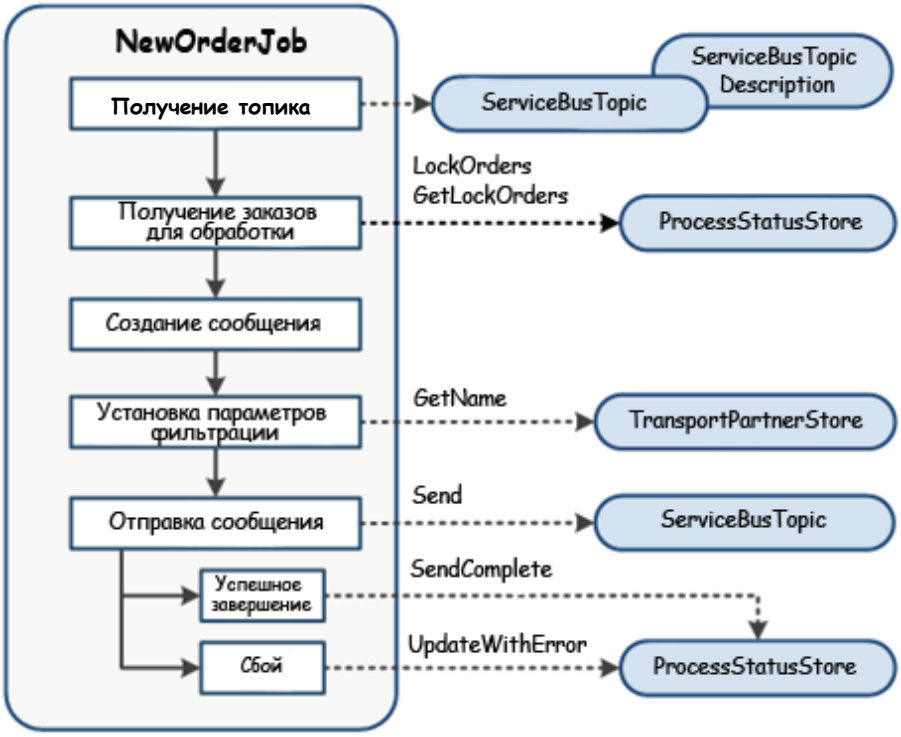



**Рисунок 3**

**Поток сообщений в системе обработки заказа**

Как уже упоминалось ранее, рабочая роль запускает обработчик заданий **NewOrderJob** путем вызова метода **Run**. Этот метод создает экземпляр класса **ServiceBusTopicDescription** и заносит в него данные, требуемые для нужного топика; затем использует его для создания экземпляра класса **ServiceBusTopic**, который называется **newOrderMessageSender**.

**C#**

```
var serviceBusTopicDescription
    = new ServiceBusTopicDescription
{
Namespace = this.serviceBusNamespace,
TopicName = topicName,
  Issuer = issuer,
DefaultKey = defaultKey
};
this.newOrderMessageSender
     = new ServiceBusTopic(serviceBusTopicDescription);
```

Обработчик заданий **NewOrderJob** может впоследствии использовать этот экземпляр **ServiceBusTopic** для отправки сообщения посредством вызова метода **Execute**.

**C#**

```
while (this.keepRunning)
{
this.Execute();
  Thread.Sleep(TimeSpan.FromSeconds(10));
}
```

**Примечание**

Метод **Execute** обрабатывает заказы пакетами, до 32 заказов за один раз. Каждый заказ в пакете отсылается асинхронно отдельным заданием, созданным методом **Send** объекта **ServiceBusTopic**. Обработчик заданий **NewOrderJob** переключается в спящий режим на 10 секунд между вызовами метода **Execute**, чтобы предотвратить излишнее потребление ресурсов. Это значение было выбрано на основе предполагаемого объема заказов и профиля производительности приложения. Значение может быть изменено при увеличении количества заказов.

### *Блокировка заказов для обработки*

Данные заказов хранятся в базе данных SQL Azure (в таблицах **Order** и **OrderDetails**), но чтобы упростить отслеживание состояния заказов и обеспечить надежность их обработки, компания Trey Research определила еще две таблицы в той же базе данных.

Первая таблица, **OrderStatus**, содержит строки состояний каждого заказа. Каждая строка представляет последнее, видимое обычными пользователями, состояние заказа — то, которое отображается на веб-сайте, когда клиент просматривает список текущих заказов. Новые строки добавляются с меткой времени. Существующие строки не обновляются для сохранения в журнале состояний. Приведенная ниже таблица описывает столбцы таблицы **OrderStatus**.

| Столбец | Описание |
|---|---|
| **OrderID** | Внешний ключ, который связывает строку со связанной строкой в таблице **Order**. |
| **Status** | Значение, описывающее последнее, видимое обычными пользователями, состояние заказа. Возможные значения: **Order placed** (заказ размещен), **Order sent to transport partner** (заказ послан транспортному партнеру), **Order received** (заказ получен транспортным партнером) и **Order shipped** (заказ отправлен – когда товары доставлены). |
| **Timestamp** | Время и дата изменения состояния заказа в формате UCT. |

Вторая таблица **OrderProcessStatus** содержит данные, которые механизм повторной попытки использует, чтобы определить, была ли отправка сообщения топику шины интеграции успешной. Приведенная ниже таблица описывает столбцы таблицы **OrderProcessStatus**.

| Столбец | Описание |
|---|---|
| **OrderID** | Внешний ключ, который связывает строку с соответствующей строкой таблицы **Order**. |
| **ProcessStatus** | Значение для внутреннего использования рабочей ролью, обозначающее состояние процесса. Возможные значения: **pending process** (ожидание обработки), **processed** (обработан), **error** (ошибка) и **criticalerror** (критическая ошибка). Эти значения определяются классом **ProcessStatus**, размещенным в папке Stores в проекте **Orders.Workers**. |
| **LockedBy** | Идентификатор рабочей роли, обрабатывающей заказ (получаемый из **RoleEnvironment.CurrentRoleInstance.Id**), или **NULL**, если заказ в данный момент не обрабатывается. |
| **LockedUntil** | Время и дата в формате UCT, когда истекает период текущей обработки заказа, или **NULL**, если заказ в данный момент не обрабатывается. |
| **Version** | Значение, используемое для поддержки оптимистической блокировки строк данной таблицы при помощи EF. Свойство **ConcurrencyMode** установлено в **Fixed**, а свойство **StoreGeneratedPatterns** установлено в **Computed**. Оно автоматически обновляется EF при добавлении и обновлении строк. |
| **RetryCount** | Количество попыток обработки заказа на данный момент. Это число увеличивается при каждой попытке рабочей роли обработать заказ. По достижении максимального числа неудачных попыток, рабочая роль устанавливает состояние процесса в значение **criticalerror** (критическая ошибка) и вызывает исключение, чтобы уведомить администратора о возникшей проблеме. |
| **BatchId** | Код GUID, который обозначает, что данный пакет заказов в данное время обрабатывается. Каждый раз, когда рабочая роль блокирует пакет заказов для обработки, она присваивает одинаковый код **BatchId** всем заказам, чтобы можно было их извлечь после блокировки базы данных. |

Для оптимизации процесса метод **Execute** извлекает заказы из базы данных SQL Azure в виде пакетов, но необходимо убедиться в том, что одни и те же заказы не будут извлечены при параллельном вызове метода **Execute**. Поэтому метод **Execute** блокирует пакет строк в таблице Order, которые он будет обрабатывать, а затем извлекает этот пакет строк как коллекцию объектов **OrderProcessStatus** путем вызова методов **LockOrders** и **GetLockedOrders** в классе **ProcessStatusStore** (который находится в папке Stores проекта **Orders.Workers**).

**C#**

```
var batchId = this.processStatusStore.LockOrders(
RoleEnvironment.CurrentRoleInstance.Id);
var ordersToProcess
= this.processStatusStore.GetLockedOrders(
RoleEnvironment.CurrentRoleInstance.Id, batchId);
```

Метод **LockOrders** в классе **ProcessStatusStore** блокирует заказы, запуская инструкцию SQL, которая устанавливает значения в строках таблицы **OrderProcessStatus**. Он присваивает текущий код экземпляра рабочей роли и код пакета топиков заказам, которые еще не обработаны, не завершились критической ошибкой и не были отменены, а также топиков, которые еще не заблокированы этим или другим экземпляром обработчика заданий **NewOrderJob**. Обратите внимание на то, что метод **LockOrders** блокирует только первые 32 заказа и только на указанный период времени. Такой подход позволяет избежать проблемы недостаточной производительности, которую может вызывать обработчик заданий **NewOrderJob**, когда пытается обработать слишком большое количество заказов за один раз.
Также он позволяет предотвратить проблему блокировки заказа на неопределенный период времени экземпляром обработчика событий **NewOrderJob**, в котором произошел сбой.

**C#**

```
public GuidLockOrders(string roleInstanceId)
{
using (var database =
         TreyResearchModelFactory.CreateContext())
  {
    var batchId = Guid.NewGuid();
    var commandText = "UPDATE TOP(32) OrderProcessStatus
"
+ "SET LockedBy = {0}, LockedUntil = {1}, "
        + "BatchId = {2} "
        + "WHERE ProcessStatus != {3} "
        + "AND ProcessStatus != {4} "
+ "AND (LockedUntil< {5} OR LockedBy IS NULL)";
this.sqlCommandRetryPolicy.ExecuteAction(
() =>database.ExecuteStoreCommand(
commandText, roleInstanceId,
DateTime.UtcNow.AddSeconds(320), batchId,
ProcessStatus.Processed,
ProcessStatus.CriticalError,
DateTime.UtcNow));
    return batchId;
  }
}
```

**Примечание**

Обработчик событий **NewOrderJob** блокирует заказы, обрабатывает их и асинхронно отправляет сообщения «New Order» (новый заказ), что означает, что после отправки сообщения обработка продолжается. В редких случаях бывает, что метод **Execute** экземпляра класса **NewOrderJob** может вызываться в то время, когда некоторые заказы все еще обрабатываются этим же экземпляром. Такое может произойти, если обработка пакета из 32 заказов занимает больше десяти секунд (см. предыдущее примечание об интервале спящего режима между вызовами метода **Execute**).

Чтобы избежать попыток повторной обработки одного и того же заказа обработчиком событий, разработчики добавили **BatchId** к каждому пакету заказов на время их блокировки. Это случайное число, формируемое для каждого пакета заказов. Впоследствии, когда метод **Execute** начинает обработку заказов, он выбирает только заказы с определенным значением **BatchId**.

Метод **GetLockedOrders** класса **ProcessStatusStore** отправляет запрос в таблицу **OrderProcessStatus** для выборки строк, успешно заблокированных методом **LockOrders**.

**C#**

```
public IEnumerable<Models.OrderProcessStatus>
GetLockedOrders(string roleInstanceId, GuidbatchId)
{
using (var database =
         TreyResearchModelFactory.CreateContext())
  {
    return
this.sqlCommandRetryPolicy.ExecuteAction(
        () =>
database.OrderProcessStatus.Where(
          o =>
o.LockedBy.Equals(roleInstanceId,
StringComparison.OrdinalIgnoreCase)
&&o.BatchId == batchId).Select(
op =>
            new Models.OrderProcessStatus
              {
LockedBy = op.LockedBy,
LockedUntil = op.LockedUntil,
                OrderId = op.OrderId,
                ProcessStatus = op.ProcessStatus,
                Order =
                  new Models.Order
                    {
                      OrderId = op.Order.OrderId,
                      UserName = op.Order.UserName,
                      OrderDate = op.Order.OrderDate,
                      Address = op.Order.Address,
                      City = op.Order.City,
                      State = op.Order.State,
                      PostalCode = op.Order.PostalCode,
                      Country = op.Order.Country,
```

```
                    Phone = op.Order.Phone,
                    Email = op.Order.Email,
                    Total = op.Order.Total
                  }
}).ToList());
  }
}
```

### *Обработка заказов с истекшим сроком блокировки*

Метод **Execute** выполняет итерацию по коллекции заблокированных заказов и отправляет каждому из них соответствующее сообщение в топик шины интеграции. Однако особенно при начальной настройке и профилировании приложения есть малая вероятность того, что размер пакета, заданный для обработчика заданий **NewOrderJob**, слишком велик, либо интервал между обработкой пакетов слишком мал. В результате, следующая итерация начинается тогда, когда текущий пакет заказов еще не отослан, а время **LockedUntil** истекло для одного или нескольких заказов в пакете. В этом случае считается, что срок действия блокировки таких заказов истек, и следующая итерация обработчика заданий **NewOrderJob** может повторно заблокировать заказы и начать их отправку. Чтобы избежать повторной посылки одного и того же заказа, метод **Execute** проверяет свойство **LockedUntil** каждого из заказов в пакете, и если значение этого свойства соответствует времени раньше текущего времени, то заказ пропускается и при необходимости обрабатывается в следующей итерации.

Приведенный ниже пример кода показывает, как компания Trey Research выполняет итерации по обрабатываемым заказам и проверяет сообщения об истекшем сроке окончания.

**C#**

```
foreach (var orderProcess in ordersToProcess)
{
  if (orderProcess.LockedUntil< DateTime.UtcNow)
{
    // Если срок окончания orderProcess истек, пропустить его, чтобы
    // другая рабочая роль могла его обработать
continue;
  }

  ...
  // Этот код создает сообщение и добавляет нужные параметры фильтрации
  // в шину интеграции, а затем
  // отправляет сообщение.
  ...
}
```

### *Размещение заказа в топике шины интеграции*

**Мнение Джаны**

Использование фильтров в топике шины интеграции позволяет реализовывать рудиментарную бизнес-логику и даже изменять свойства сообщений при их передаче в топике. Это также позволяет отделить отправителей от получателей путем изменения числа подписчиков, прослушивающих топик. Каждый получает сообщения, предназначенные только для него, посредством использования фильтра в топике. Однако топики и фильтры не могут обеспечить механизм универсального рабочего процесса, поэтому не следует пытаться реализовать сложную логику на основе топиков шины интеграции.

Топик шины интеграции, который используется компанией Trey Research в приложении Orders, настроен так, чтобы фильтровать сообщения по месту доставки и общей стоимости заказа. Сводка каждого заказа должна быть отправлена соответствующему транспортному партнеру, чтобы узнать условия доставки. Кроме того, данные всех заказов с общей стоимостью более 10 000 долл. США передаются локальной службе для сохранения в базе данных журнала аудита. Метод **Execute** добавляет необходимые свойства в сообщения, и таким образом топик шины интеграции может фильтровать их и отправлять нужным подписчикам. Приведенный ниже пример кода показывает, как метод **Execute** использует отдельный класс **TransportPartnerStore** для определения имени соответствующего транспортного партнера.

**C#**

```
var transportPartnerName = this.transportPartnerStore
            .GetTransportPartnerName(orderProcess.Order.State);
```

Класс **TransportPartnerStore** используется для определения транспортного партнера, которому должно быть отправлено сообщение; выбор делается с учетом местоположения получателя заказа. Это определяется в файле TransportPartnerStore.cs в папке Stores в проекте **Orders.Workers**.

### *Создание нового сообщения о заказе*

Теперь метод **Execute** может создать сообщение о заказе для отправки. Для обработки сбоев во время отправки сообщений, настраиваемому механизму повторной попытки может понадобиться несколько попыток для отправки одного и того же сообщения. Метод **Execute** не может просто создать сообщение и передать его методу **Send** класса **ServiceBusTopic**, так как механизм сериализации, используемый сообщениями шины интеграции подразумевает, что текст сообщения может быть прочитан только один раз. Поэтому метод **Execute** определяет функцию, которая создает сообщение динамически и передает эту функцию методу **Send** класса **ServiceBusTopic**. Этот метод может впоследствии вызываться методом **Send** для создания свежей копии сообщения при каждой его отправке.

Приведенный ниже пример кода описывает раздел метода **Execute**, который определяет эту функцию. Это делается следующим образом: сначала создается экземпляр класса **NewOrderMessage** (определенного в папке Communication\Messages проекта **Orders.Shared**), представляющий сообщение, которое Trey Research собирается отправить, а затем из него создается экземпляр **BrokeredMessage** шины интеграции. В коде задаются следующие свойства экземпляра **BrokeredMessage**:

- Свойство **TransportPartnerName**, ранее полученное из **TransportPartnerStore**, используется фильтром шины интеграции для направления сообщения нужному транспортному партнеру.
- Свойство **AcsNamespace** позволяет получателю определить, какой экземпляр ACS использовался отправителем для аутентификации (настройки приложения могут позволять

использование ACS в нескольких центрах обработки данных для аутентификации партнеров при доступе к шине интеграции). Необходимо, чтобы получатель сообщения мог проверить отправителя, как описано в разделе «Защита сообщений» главы 4 «Реализация надежного обмена сообщениями и информацией в облаке».

- Свойство **OrderAmount** используется фильтром шины интеграции для определения заказов, общая стоимость которых превышает заданное значение, и отправки копий этих сообщений в локальный журнал аудита.
- Свойство **ReplyTo** задает очередь шины интеграции, в которую получатель должен отправлять все ответные сообщения. Например, транспортный партнер использует эту очередь для размещения сообщений, которые подтверждают получение данного сообщения, как описано в разделе «Согласующие сообщения и ответы» главы 4 «Реализация надежного обмена сообщениями и информацией в облаке».
- Свойство **ServiceBusNamespace** представляет собой пространство имен шины интеграции, которое содержит очередь шины, заданную в свойстве **ReplyTo**. Приложение Orders может быть развернуто в нескольких центрах обработки данных, и транспортный партнер должен знать, какой именно экземпляр шины интеграции он должен использовать при отправке ответного сообщения.

---

**C#**

```
  Func<BrokeredMessage>brokeredMessageFunc = () =>
    {
      // Отправка сообщения о новом заказе
      var msg = new NewOrderMessage
      {
        OrderId = orderProcess.Order.OrderId,
        OrderDate = orderProcess.Order.OrderDate,
ShippingAddress = orderProcess.Order.Address,
Amount =
Convert.ToDouble(orderProcess.Order.Total),
CustomerName = orderProcess.Order.UserName
      };

      var brokeredMessage = new BrokeredMessage(msg)
{
        MessageId = msg.OrderId.ToString(),
        CorrelationId = msg.OrderId.ToString(),
        Properties = { { "TransportPartnerName",
                         transportPartnerName },
{ "ServiceBusNamespace",
this.serviceBusNamespace },
{ "AcsNamespace",
this.acsNamespace },
{ "OrderAmount",
                         orderProcess.Order.Total } },
        ReplyTo = this.replyQueueName
      };

      return brokeredMessage;
    };
```

Это сообщение будет отправлено асинхронно, поэтому класс **NewOrderJob** должен также скомпоновать объект, который можно будет передать как состояние для асинхронных вызовов метода, как показано ниже в примере кода. Этот объект должен содержать код заказа (для того чтобы механизм повторной попытки мог обновлять строки состояния соответствующих заказов в таблице **OrderStatus**) и имя транспортного партнера (для отображения его клиентам при просмотре существующих заказов).

**C#**

```
var objectState = new Dictionary<string, object>
{
  { "orderId", orderProcess.OrderId },
  {"transportPartner", transportPartnerName }
}
```

### *Отправка сообщений о новом заказе*

Теперь метод **Execute** может отправлять заказ топику шины интеграции. Для этого он вызывает метод **Send** созданного им экземпляра **ServiceBusTopic**, сохраненного в переменной **newOrderMessageSender**. Как описано в разделе «Отправка сообщений в топик шины интеграции» главы 4 «Реализация надежного обмена сообщениями и информацией в облаке», метод **Send** класса **ServiceBusTopic** принимает четыре параметра: функцию, которая создает экземпляр **BrokeredMessage** для отправки, асинхронный объект состояния и два метода **Action** (один выполняется после отправки сообщения, а второй выполняется, если при отправке произошел сбой). Класс **ProcessStatusStore** определяет лямбда-условия, которые передаются как два метода действия, описываемые следующим кодом.

**C#**

```
this.newOrderMessageSender
.Send(
brokeredMessageFunc,
      objectState,
      (obj) =>
      {
        var objState = (IDictionary<string, object>)obj;
        var orderId = (Guid)objState["orderId"];
var transportPartner
           = (string)objState["transportPartner"];
this.processStatusStore.SendComplete(orderId,
transportPartner);
      },
      (exception, obj) =>
      {
        var objState = (IDictionary<string, object>)obj;
        var orderId = (Guid)objState["orderId"];
this.processStatusStore.UpdateWithError(
exception, orderId);
      });
}
```

Первое действие, выполняемое после успешной отправки сообщения, извлекает код заказа и имя транспортного партнера из асинхронного объекта состояния, и передает их методу **SendComplete** экземпляра класса **ProcessStatusStore**, используемого классом **NewOrderJob**. Второе действие извлекает из асинхронного объекта состояния только код заказа и передает его и текущий экземпляр **Exception** экземпляру метода **UpdateWithError**.

### *Завершение надежного процесса отправки*

Метод **SendComplete** класса **ProcessStatusStore**, вызываемый методом **Execute** в случае успешной отправки сообщения, выполняет следующие задачи:

- В соответствующих строках таблицы **OrderProcessStatus** обновляет значения в столбце **ProcessStatus** на новое значение **processed** (обработан), что предотвращает попытку повторной отправки заказа новой итерацией обработчика заказов **NewOrderJob**.
- Добавляет новую строку в таблицу **OrderStatus** для отображения текущего состояния заказа **Order sent to transportpartner** (Заказ отправлен транспортному партнеру) с правильной меткой времени.
- Заносит в таблицу **Order** имя транспортного партнера, который будет доставлять заказ.

**C#**

```
public void SendComplete(Guid orderId,
string transportPartner)
{
using (var database =
         TreyResearchModelFactory.CreateContext())
  {
    try
    {
      using (var t = new TransactionScope())
{
        // Предотвращение повышения транзакции.
this.sqlConnectionRetryPolicy.ExecuteAction(
            () =>database.Connection.Open());

        // Обновление строки таблицы OrderProcessStatus
        var processStatus =
this.sqlCommandRetryPolicy.ExecuteAction(
            () =>database.OrderProcessStatus
.SingleOrDefault(
o =>o.OrderId == orderId));
processStatus.ProcessStatus
            = ProcessStatus.Processed;
processStatus.LockedBy = null;
processStatus.LockedUntil = null;
this.sqlCommandRetryPolicy.ExecuteAction(
            () =>database.SaveChanges());

        // Добавление новой строки в таблицу OrderStatus
var status = new OrderStatus { OrderId = orderId,
Status =
          "TreyResearch: Order sent to transport
partner",
Timestamp = DateTime.UtcNow };
        database.OrderStatus.AddObject(status);
this.sqlCommandRetryPolicy.ExecuteAction(
            () =>database.SaveChanges());

        // Обновление строки таблицы Order
```

```
        var order =
this.sqlCommandRetryPolicy.ExecuteAction(
() =>database.Order.SingleOrDefault(
o =>o.OrderId == orderId));
order.TransportPartner = transportPartner;
this.sqlCommandRetryPolicy.ExecuteAction(
            () =>database.SaveChanges());

t.Complete();
      }
    }
    catch (UpdateException ex)
    {
      ...
    }
  }
}
```

Метод **UpdateWithError**, вызываемый методом **Execute** в случае неуспешной отправки сообщения, формирует подходящие предупреждающие сообщения, используя для этого пользовательский класс **TraceHelper**. Затем он обновляет соответствующую строку таблицы **OrderProcessStatus** значением **error** (ошибка) и устанавливает значение **null** в столбцах **LockedBy** и **LockedUntil**, чтобы разблокировать сообщение и сделать его доступным для обработки.

Метод **UpdateWithError** также проверяет, не превышает ли число повторных попыток заданное максимальное значение (определяемое в файле ServiceConfiguration.cscfg). Если число превышено, метод обновляет строку таблицы **OrderProcessStatus** значением **criticalerror** (критическая ошибка) и формирует сообщение о том, что сбой в обработке заказа должен быть проанализирован администратором.

**C#**

```
public void UpdateWithError(Exception exception,
Guid orderId)
{
  TraceHelper.TraceWarning("NewOrderJob: The Order '{0}' "
    + "couldn’t be processed. Error details: {1}",
orderId.ToString(), exception.ToString());

using (var database
    = TreyResearchModelFactory.CreateContext())
  {
    var processStatus =
this.sqlCommandRetryPolicy.ExecuteAction(
            () =>database.OrderProcessStatus
.SingleOrDefault(
o =>o.OrderId == orderId));
processStatus.ProcessStatus = ProcessStatus.Error;
processStatus.LockedBy = null;
processStatus.LockedUntil = null;
processStatus.RetryCount
      = processStatus.RetryCount + 1;

var newOrderJobRetryCountCheck = int.Parse(
```

```
CloudConfiguration.GetConfigurationSetting(
"NewOrderJobRetryCountCheck", "3"));

if (processStatus.RetryCount
>newOrderJobRetryCountCheck)
    {
processStatus.ProcessStatus
          = ProcessStatus.CriticalError;
      TraceHelper.TraceError("NewOrderJob: The Order '{0}' "
        + "has reached {1} retries. This order requires "
+ "manualintervention.",
orderId.ToString(), processStatus.RetryCount);
    }

this.sqlCommandRetryPolicy.ExecuteAction(
                  () =>database.SaveChanges());
  }
}
```

На данном этапе сообщение отправлено и его состояние, сохраненное в базе данных SQL Azure, совпадает с фактическим состоянием сообщения.

Следующее задание должно извлечь заказ из подписки соответствующего транспортного партнера.

## Как компания Trey Research отделяет обработку заказа от системы транспортного партнера

Для взаимодействия с системами доставки различных транспортных партнеров, компания Trey Research реализовала ряд компонентов связи в виде коннекторов и адаптеров. Эти компоненты обеспечивают интерфейс между шиной интеграции и транспортным партнером. Для каждого транспортного партнера используется определенный компонент, который позволяет извлекать сообщения из соответствующей шины интеграции, переводить их в формат, принятый транспортным партнером, и передавать его во внутреннюю систему партнера.

Ответные сообщения транспортного партнера передаются обратно в компонент, конвертируются в сообщения шины интеграции, а затем размещаются в очереди шины в приложении Orders. На рисунке 1 в начале главы показано, где именно развернуты эти компоненты. Пример реализации этих компонентов, предоставляемых в решении Trey Research, более подробно описан в следующем разделе.

> **Примечание**
>
> Адаптеры и коннекторы в данном примере решения приведены в качестве простых примеров, они позволяют транспортному партнеру взаимодействовать с топиками и очередями шины интеграции, которые используются решением Trey Research. В реальности эти компоненты обладают гораздо более сложной внутренней бизнес-логикой, чем компоненты данного примера.

### Получение и обработка заказа транспортным партнером

Транспортные партнеры подключаются к топику шины интеграции, в котором обработчик заданий **NewOrderJob** размещает сообщения заказа. Каждый транспортный партнер использует свою собственную подписку шины интеграции, в которой настроен фильтр, проверяющий свойство **TransportPartnerName** каждого из сообщений. Подписки и фильтры создаются программой установки

в проекте **TreyResearch.Setup**. Более подробно об этом написано в разделе «Подписка на топик шины интеграции» главы 4 «Реализация надежного обмена сообщениями и информацией в облаке».

Примеры транспортных партнеров представлены в проекте **TransportPartner**. Решение включает два транспортных партнера: Contoso (реализован классом **ContosoTransportPartner**), который доставляет заказы клиентам, расположенным в том же или соседних штатах, что и производственное предприятие компании Trey Research, и Fabrikam (реализован классом **FabrikamTransportPartner**), который доставляет заказы клиентам в остальные штаты.

Местный транспортный партнер Contoso подключается к шине интеграции, используя класс **Connector**, определенный в папке Connectivity проекта **TransportPartner**. А удаленный партнер Fabrikam использует класс **Adapter**, определенный в той же папке.

**Примечание**

Использование коннекторов и адаптеров обосновывается в разделе «Реализация адаптеров и коннекторов для преобразования и переформатирования сообщений» главы 4 «Реализация надежного обмена сообщениями и информацией в облаке».

Оба класса **Connector** и **Adapter** наследуются из класса **OrderProcessor**, который обеспечивает функциональность подключения к топику шины интеграции и получения сообщений в методе **Run**.

**C#**

```
public void Run()
{
  var serviceBusNamespaces = ConfigurationManager.
    AppSettings["serviceBusNamespaces"].
Split(',').ToList();
  ...

  foreach (var serviceBusNamespace in
serviceBusNamespaces)
{
this.serviceBusSubscriptionDescription.Namespace =
      serviceBusNamespace;
    var serviceBusSubscription = new
ServiceBusSubscription(
this.serviceBusSubscriptionDescription);
    var receiverHandler = new
ServiceBusReceiverHandler<NewOrderMessage>(
serviceBusSubscription.GetReceiver())
    {
      MessagePollingInterval = TimeSpan.FromSeconds(2)
    };

receiverHandler.ProcessMessages(
      (message, queueDescription, token) =>
      {
return Task.Factory.StartNew(
() => this.ProcessMessage(
            message, queueDescription),
this.tokenSource.Token,
          TaskCreationOptions.None,
          context);
```

```
      },
this.tokenSource.Token);
  }
}
```

В главе 4 «Реализация надежного обмена сообщениями и информацией в облаке» описывается, как и почему метод **Run** создает объект **ServiceBusReceiverHandler** для извлечения сообщений. Так как этот метод связан с обработкой заказов, то следует обратить внимание на два важных аспекта:

- Веб-приложение Trey Research и рабочая роль могут быть развернуты в нескольких центрах обработки данных. Развертывание в каждом центре обработки настраивается собственным набором топиков и очередей шины интеграции и в собственном пространстве имен шины интеграции. Поэтому метод **Run** должен подключаться к каждому центру данных и прослушивать сообщения топика каждого из них. Эту задачу реализует цикл **foreach**; пространства имен шины интеграции, к которым должен подключаться метод, определены в файле конфигурации, и цикл подключается к топику шины интеграции в каждом пространстве.
- Вызов метода **ProcessMessages** объекта **ServiceBusReceiverHandler** вызывает метод **ProcessMessage**. Этот метод обеспечивает логику обработки сообщения заказа, что показано в следующем примере кода:

**C#**

```
protected virtual void ProcessMessage(
NewOrderMessage message,
    ServiceBusQueueDescription queueDescription)
{
  var trackingId = this.ProcessOrder(message,
queueDescription);

  if (trackingId != Guid.Empty)
  {
    // Получение SWT из ACS.
    var token = this.GetToken(queueDescription);

var statusMessage =
string.Format("{0}: Order Received",
this.TransportPartnerDisplayName);
this.SendOrderReceived(message, queueDescription,
statusMessage, trackingId, token);
  }
}
```

Метод **ProcessMessage** вызывает метод **ProcessOrder** для выполнения обработки заказа специально для каждого транспортного партнера. Метод **ProcessOrder** — это абстрактный метод с реализациями в классах **Adapter** и **Connector**. В примере транспортного партнера заказы хранятся в списке объектов **ActiveOrder**.Каждый заказ отображается в форме Windows Form, которая обеспечивает пользовательский интерфейс (эту задачу решает обработчик событий **OnOrderProcessed**).

Однако есть небольшая вероятность того, что один и тот же заказ может быть получен несколько раз; механизм повторной попытки класса **NewOrderJob**, который размещает сообщения в топике шины интеграции, может повторять отправку одного и того же сообщения в случае недостаточной надежности и производительности сетевого соединения. Соответственно, перед созданием объекта **ActiveOrder**,

метод **ProcessOrder** проверяет, что в списке еще нет заказа с кодом, совпадающим с кодом полученного сообщения. Если же такой заказ есть в списке, то новый заказ отменяется как повторяющийся.

Следующий пример кода демонстрирует реализацию метода **ProcessOrder** в классе **Adapter**.

**C#**

```
protected override GuidProcessOrder(
Orders.Shared.Communication.Messages.NewOrderMessage
  message, ServiceBusQueueDescription queueDescription)
{
  var processedOrder =
this.orderStore.GetById(message.OrderId);

if (processedOrder != null)
  {
    // Этот заказ был получен для обработки более
    // одноного раза и будет отменен.
    return Guid.Empty;
  }

  var activeOrder = new ActiveOrder
  {
    OrderId = message.OrderId,
ShippingAddress = message.ShippingAddress,
Amount = message.Amount,
    ReplyTo = queueDescription.QueueName,
ReplyToNamespace = queueDescription.Namespace,
    Status = "received",
SwtAcsNamespace = queueDescription.SwtAcsNamespace
  };

this.orderStore.Add(activeOrder);

  // Вызов службы транспортного партнера и
  // возвращение кода трассировки.
  var trackingId = this.transportServiceWrapper.
RequestShipment(activeOrder);

  if (this.OnOrderProcessed != null)
  {
this.OnOrderProcessed(this,
      new OrderProcessedEventArgs
      { ActiveOrder = activeOrder });
  }

  // если код трассировки получен, запрос на доставку
  // подтвержден, то состояние заказа в очереди может
быть обновлено
  // на "Order Received" (Заказ получен).
  return trackingId;
}
```

Обратите внимание на то, что пример предполагает наличие внутренней системы у транспортных партнеров, которые по-разному реагируют на получение сообщения заказа. Это делает пример более приближенным к реальному приложению.

- Местный транспортный партнер Contoso мгновенно отвечает сообщением с подтверждением о том, что заказ был получен. После отправки заказа, местный транспортный партнер посылает еще одно сообщение.
- Удаленный транспортный партнер Fabrikam принимает заказ и формирует код трассировки. Он отправляет ответное сообщение с кодом компании Trey Research. Многие удаленные транспортные партнеры предоставляют свои собственные веб-приложения, в которых предусмотрен вход для клиентов, чтобы они могли просматривать ход обработки заказа, введя его код трассировки (за эти веб-приложения отвечают транспортные партнеры; учебное решение не содержит пример такого приложения).

### Подтверждение заказа или индикация отправки транспортным партнером

Получив заказ, местный транспортный партнер должен подтвердить успешное получение сообщения. Позже, когда заказ отправлен, транспортный партнер посылает еще одно сообщение. Как было описано в предыдущей главе, удаленный транспортный партнер Fabrikam посылает единственное сообщение, когда заказ получен. В обоих случаях метод **ProcessMessage** вызывает метод **SendOrderReceived** для построения и отправки соответствующего сообщения приложению Orders.

**C#**

```
protected virtual void ProcessMessage(
NewOrderMessage message,
    ServiceBusQueueDescription queueDescription)
{
  var trackingId = this.ProcessOrder(message,
queueDescription);

  if (trackingId != Guid.Empty)
  {
    // Получение SWT из ACS.
    var token = this.GetToken(queueDescription);

var statusMessage =
string.Format("{0}: Order Received",
this.TransportPartnerDisplayName);
this.SendOrderReceived(message, queueDescription,
statusMessage, trackingId, token);
  }
}

protected void SendOrderReceived(
NewOrderMessage message,
  ServiceBusQueueDescription queueDescription,
  string statusMessage, GuidtrackingId, string swt)
{
this.SendToUpdateStatusQueue(message.OrderId, trackingId,
statusMessage, queueDescription, swt);
}
```

Метод **SendOrderReceived** вызывает метод **SendToUpdateStatusQueue**, содержащий в себе логику построения сообщения **OrderStatusUpdateMessage**, которое он размещает в очереди шины интеграции. Имя очереди, которая должна быть выбрана, и пространство имен шины интеграции, в котором она находится, определены в свойствах **ReplyTo** и **ServiceBusNamespace** исходного сообщения заказа. Они использовались для создания объекта **ServiceBusQueueDescription** при получении сообщения методом **ProcessMessage** класса **ServiceBusReceiverHandler** (подробнее класс **ServiceBusReceiverHandler** описан в разделе «Получение сообщений из очереди шины интеграции и их асинхронная обработка» главы 4 «Реализация надежного обмена сообщениями и информацией в облаке»).

Ответ транспортного партнера должен содержать маркер безопасности, что позволит приложению Orders аутентифицировать ответное сообщение. Этот механизм также описан в главе 4 «Реализация надежного обмена сообщениями и информацией в облаке» в разделе «Защита сообщений». Пространство имен ACS, в котором определяется этот маркер, обеспечивается свойством **AcsNamespace** исходного заказа, это значение также добавляется в объект **ServiceBusQueueDescription**. Когда транспортный партнер посылает ответ, маркер безопасности извлекается из определенного пространства имен ACS и добавляется в ответное сообщение.

**C#**

```
public class ServiceBusReceiverHandler<T>
{
  ...
private void ProcessMessage(
    IBrokeredMessageAdapter message)
  {
    if (message != null)
    {
      ...

      var queueDescription = new
ServiceBusQueueDescription
      {
QueueName = message.ReplyTo,
      };

if (message.Properties.ContainsKey(
"ServiceBusNamespace"))
      {
queueDescription.Namespace = message.Properties[
"ServiceBusNamespace"].ToString();
      }

      if
(message.Properties.ContainsKey("AcsNamespace"))
{
queueDescription.SwtAcsNamespace =
message.Properties["AcsNamespace"].ToString();
      }
      ...
    }
    ...
  }
  ...
}
```

Объект **ServiceBusQueueDesciption** использует метод **SendToUpdateStatusQueue** через параметр **queueDescription**. Метод **SendToUpdateStatusQueue** создает объект **OrderStatusUpdateMessage** и заполняет его данными подтверждающего сообщения, включая сообщение о состоянии «Order Received» (Заказ получен) и код трассировки, сформированный транспортным партнером. Объект **OrderStatusObjectMessage** упаковывается внутри объекта **BrokeredMessage** и размещается в очереди, определенной параметром **queueDescription**. Обратите внимание на то, что свойство **CorrelationId** этого ответного сообщения устанавливается для кода заказа исходного запроса, чтобы приложение Orders при получении могло сопоставить это ответное сообщение с запросом.

Следующий пример кода демонстрирует метод **SendToUpdateStatusQueue**. Обратите внимание, что в качестве механизма оптимизации этот метод помещает в кэш-копию объекта **ServiceBusQueue**, созданного им в объекте **Dictionary**, с именем **statusUpdateQueueDictionary**. При следующем вызове метода **SendToUpdateStatusQueue**, он найдет этот объект **ServiceBusQueue** в **Dictionary** и не будет необходимости создавать его снова.

**C#**

```
private void SendToUpdateStatusQueue(Guid orderId,
GuidtrackingId, string orderStatus,
  ServiceBusQueueDescription queueDescription, string
swt)
{
  var updateStatusMessage =
new BrokeredMessage(
      new OrderStatusUpdateMessage
      {
        OrderId = orderId,
        Status = orderStatus,
TrackingId = trackingId,
        TransportPartnerName =
this.TransportPartnerDisplayName,
      })
      { CorrelationId = orderId.ToString() };

updateStatusMessage.Properties.Add(
"SimpleWebToken", swt);

  ServiceBusQueue replyQueue;
if (this.statusUpdateQueueDictionary.
ContainsKey(queueDescription.Namespace))
  {
replyQueue = this.statusUpdateQueueDictionary[
queueDescription.Namespace];
  }
  else
  {
    var description = new ServiceBusQueueDescription
    {
Namespace = queueDescription.Namespace,
QueueName = queueDescription.QueueName,
DefaultKey =
this.serviceBusQueueDescription.DefaultKey,
      Issuer = this.serviceBusQueueDescription.Issuer
    };
```

```
replyQueue = new ServiceBusQueue(description);
this.statusUpdateQueueDictionary.Add(
queueDescription.Namespace, replyQueue);
  }

  var brokeredMessageAdapter =
    new BrokeredMessageAdapter(updateStatusMessage);
replyQueue.Send(brokeredMessageAdapter);
}
```

Для местного транспортного партнера Contoso при доставке заказа, приложение вызывает метод **SendOrderShipped** класса **OrderProcessor**. Метод **SendOrderShipped** работает почти так же, как и метод **SendOrderReceived**, вызывая метод **SendToUpdateStatusQueue** для создания и размещения сообщения **OrderStatusUpdate** в очереди шины интеграции.

### Получение сообщений с подтверждением и сообщений о состоянии в приложении Orders

Наряду с запуском объекта **NewOrderJob** для отправки новых заказов транспортным партнерам, каждая рабочая роль в решении Trey Research создает объект **StatusUpdateJob** для прослушивания сообщений о состоянии, полученных от транспортных партнеров. Этот класс находится в файле StatusUpdateJob.cs в папке Jobs проекта **Orders.Workers**. Также как и класс **OrderProcessor**, метод **Run** класса **StatusUpdateJob** использует объект **ServiceBusReceiverHandler** для подключения к очереди и извлечения сообщений.

Объект **StatusUpdateJob** также обеспечивает бизнес-логику, которая запускается для каждого полученного сообщения о состоянии как лямбда-выражение, когда он вызывает метод **ProcessMessages** объекта **ServiceBusReceiverHandler**. Лямбда-выражение выполняет следующие задачи:

- Проверяет маркер аутентификации в сообщении и генерирует исключение **InvalidTokenException**, если маркер не опознан (сообщение может быть получено от неавторизованной стороны).
- Создает запись о состоянии заказа с данными, предоставленными транспортным партнером.
- Отменяет сообщение, если дублируется уже существующий заказ (логика повторной попытки транспортного партнера может отправлять повторяющиеся сообщения о состоянии при возникновении какой-либо неустойчивой ошибки).
- Обновляет заказ именем транспортного партнера, который будет осуществлять поставку клиенту.
- Добавляет в запись о состоянии заказа код трассировки, предоставленный партнером.
- Хранит запись о состоянии заказа в базе данных Trey Research (другие части приложения могу запрашивать этот статус; например, если клиент хочет знать, был ли отгружен заказ).

Следующий пример кода демонстрирует класс **StatusUpdateJob**, определяющий эту логику.

**C#**

```
public void Run()
{
  ...
receiverHandler.ProcessMessages(
    (message, replyTo, token) =>
    {
```

```
return Task.Factory.StartNew(
        () =>
        {
          ...
          if (!this.IsValidToken(message.OrderId,
token))
{
            // Генерация исключения, которое должно быть
перехвачено обработчиком.
            // Отправление в очередь недоставленных
сообщений.
throw new InvalidTokenException();
          }

var orderStatus = new OrderStatus {
            OrderId = message.OrderId,
            Status = message.Status
          };

using (var db =
            TreyResearchModelFactory.CreateContext())
          {
            // Проверка повторяющихся записей
            // в таблице состояний заказа. Если получено
повторяющееся сообщение,
            // то оно удаляется.
            var existingStatus =
this.sqlCommandRetryPolicy.ExecuteAction(
() =>db.OrderStatus.SingleOrDefault(
os =>os.OrderId == message.OrderId&&
os.Status == message.Status));
            if (existingStatus != null)
            {
              return;
            }

            var order = this.sqlCommandRetryPolicy.
ExecuteAction(
() =>db.Order.Single(o =>
o.OrderId == message.OrderId));

order.TransportPartner =
message.TransportPartnerName;

            if (message.TrackingId != Guid.Empty)
{
order.TrackingId = message.TrackingId;
            }

db.OrderStatus.AddObject(
              new OrderStatus {
                OrderId = orderStatus.OrderId,
                Status = orderStatus.Status,
```

```
                Timestamp = DateTime.UtcNow
              });

this.sqlCommandRetryPolicy.ExecuteAction(
() =>db.SaveChanges());
          }
        });
    },
  ...);
}
```

## Выводы

В этой главе была рассмотрена бизнес-логика обработки заказов в приложении Orders, реализованная компанией Trey Research. Бизнес-логика разработана на базе топиков, подписок и очередей шины интеграции и использует инфраструктуру программного обеспечения, описанную в главе 4 «Реализация надежного обмена сообщениями и информацией в облаке» с целью обеспечения расширяемого слоя для надежного асинхронного обмена сообщениями.

В этой главе также рассказывается о том, как компания Trey Research создала надежный механизм размещения сообщений в топике шины интеграции, выявления и обработки сбоев и прозрачной повторной попытки отправки сообщений в случае сбоев.

И наконец, в этой главе описано, как внутренняя бизнес-логика транспортных партнеров была отделена от инфраструктуры передачи сообщений, используемой компанией Trey Research. В примере решения коннекторы и адаптеры вызывают существующие бизнес-службы, предоставляемые транспортными партнерами, без необходимости изменять или прерывать работу их внутренних систем. Эти коннекторы и адаптеры также обеспечивают независимость от местоположения, отслеживая источник запросов на входящие заказы и направляя ответные сообщения обратно по соответствующему назначению.

## Дополнительная информация

- «Cloud Architecture: The Scheduler-Agent-Supervisor Pattern» http://vasters.com/clemensv/CommentView,guid,83f937f7-b838-43d0-ad61-74605eceafa2.aspx
- «Achieving Transactional Behavior with Messaging» http://vasters.com/clemensv/2011/10/06/Achieving+Transactional+Behavior+With+Messaging.aspx
- «Queues, Topics, and Subscriptions» http://msdn.microsoft.com/en-us/library/windowsazure/hh367516.aspx
- «The Transient Fault Handling Application Block» http://msdn.microsoft.com/en-us/library/hh680934(v=pandp.50).aspx

# Глава 6. Максимизация масштабируемости, доступности и эффективности приложения Orders

Основная причина, по которой компания Trey Research решила изменить конфигурацию приложения Orders и перенести его на технологическую платформу Windows Azure, состояла в том, чтобы воспользоваться преимуществами улучшенной масштабируемости и доступности этой среды. Так как компания Trey Research расширяет свою деятельность и рассчитывает привлекать все большее количество клиентов, то она разработала свое решение, предусмотрев возможность обработки больших объемов заказов от клиентов, рассредоточенных по всему миру. Это было непростой задачей, так как приложение Orders должно было остаться таким же быстродействующим и при этом рентабельным независимо от количества клиентов, требующих обслуживания в любой момент времени. Компания Trey Research проанализировала деятельность своей системы и определила три основных требования. Решение должно:

- Автоматически масштабироваться, чтобы реагировать как на увеличение спроса для своевременной обработки заказов, так и на уменьшение спроса, чтобы снизить текущие расходы.
- Сократить задержку сети, связанную с доступом клиента к веб-приложению Orders и его ресурсам, размещенным удаленно в общедоступных и неконтролируемых (с точки зрения компании Trey Research) сетях, например в Интернете.
- Оптимизировать время отклика и пропускную способность веб-приложения для поддержания удовлетворительного обслуживания пользователей.

В этой главе рассказывается о том, как компания Trey Research реализует данные требования в своем решении.

## Сценарий и контекст

Компания Trey Research реализовала свое решение в виде веб-приложения, которое прежде всего ориентируется на клиентов в Соединенных Штатах, поэтому компания сначала развернула решение в двух центрах обработки данных Windows Azure: на юге и севере США. Однако компания Trey Research планирует расширить свою деятельность и может предвидеть время, когда их приложение будет необходимо развернуть в других центрах данных по всему миру, чтобы удовлетворить спрос зарубежных клиентов. По этой причине, когда компания Trey Research разрабатывала приложение Orders, была предусмотрена возможность разрешения клиентам подключаться к любому экземпляру приложения для обеспечения функциональности, одинаковой во всех экземплярах приложения. Таким

образом, компания Trey Research может запускать и останавливать экземпляры приложения Orders, а клиенты всегда смогут просматривать продукцию, размещать заказы и запрашивать статусы своих заказов, независимо от того, к какому из экземпляров приложения они подключены в любой момент времени. Такой подход также позволяет компании Trey Research равномерно балансировать нагрузку между всеми доступными экземплярами приложения Orders и поддерживать пропускную способность. Кроме того, если экземпляр становится недоступен или не удается установить соединение с экземпляром, то клиенты могут быть направлены на другие рабочие экземпляры приложения.

На практике это решение зависит от целого ряда компонентов, которые обеспечивают необходимую инфраструктуру для определения оптимального экземпляра приложения Orders, к которому клиент должен подключиться. Эти компоненты прозрачно перенаправляют запросы клиентов (и перенаправляют запросы в случае сбоя связи с экземпляром), а также поддерживают согласованность данных во всех центрах данных. Кроме того, ресурсы, используемые приложением Orders, предполагают дальнейшую передачу запросов по сети, например для получения каталога продукции, информации о клиенте или заказе. Доступ к этим ресурсам должен быть масштабируемым и своевременным, чтобы обеспечить клиентам быстрый отклик.

## Управление эластичностью в приложении Orders

Компания Trey Research отметила, что приложение Orders испытывает моменты пиковой нагрузки и падения спроса в течение рабочего дня. Некоторые из этих изменений нагрузки были предсказуемы, например низкий уровень использования приложения рано утром и высокая загрузка во второй половине рабочего дня, в то время как другие изменения были более неожиданными; иногда спрос увеличивался при рассмотрении определенного продукта в техническом журнале, а иногда спрос менялся без предсказуемых причин.

Компании Trey Research требовалось найти решение, которое позволило бы масштабировать приложение Orders для обработки существенно изменяющегося количества клиентов, входящих в систему, поиска продуктов, а также размещения заказов, не увеличивая при этом время отклика и не снижая рентабельность.

**Мнение Бхарата**

Технология Windows Azure обеспечивает эластичность и масштабируемость, позволяя запускать и останавливать экземпляры ролей по требованию. Однако, если вы действительно не управляете проактивно счетчиком экземпляров своей роли, то можете упустить основные преимущества облачных вычислений.

### Способы управления эластичностью в приложении Orders

Компания Trey Research рассмотрела ряд вариантов, чтобы определить, как лучше масштабировать приложение Orders. Эти варианты, а также их преимущества и недостатки описаны в следующих разделах.

#### Приложение без возможности масштабируемости

Это самый простой вариант. Приложение Orders было разработано и реализовано так, чтобы можно было воспользоваться преимуществами одновременно работающих веб- и рабочих ролей и использовать асинхронную систему обмена сообщениями для отправки и приема сообщений при минимальном времени отклика приложения для пользователей. В этом случае, почему бы просто не развернуть экземпляры веб- и рабочих ролей приложения в каждом возможном центре данных, а также

выделить каждой роли наибольший возможный размер виртуальной машины с максимальным количеством процессорных ядер и наибольшим доступным объемом памяти?

Такой подход привлекателен тем, что он подразумевает минимальное техническое обслуживание со стороны обслуживающего персонала компании Trey Research. Он также очень прост в реализации. Однако его реализация может быть очень дорогой; размещение веб- или рабочих ролей с использованием очень большого размера виртуальный машины (ExtraLarge, как это определено в модели ценообразования Windows Azure) в настоящее время обходится в 24 раза дороже с точки зрения почасового тарифа, чем размещение той же роли на виртуальной машине очень маленького размера (Extra Small). Если работа клиентов большую часть времени не требует обработки или возможностей памяти очень большой виртуальной машины, то компания Trey Research вынуждена будет оплачивать размещение фактически простаивающей большую часть времени виртуальной машины. Если умножить стоимость на количество экземпляров, размещенных во всех центрах данных, то окончательная сумма может быть значительной.

Возникает еще один вопрос касательно этого подхода: сколько экземпляров веб- и рабочих ролей должна создать компания Trey Research? Если выбранное количество слишком велико, то вопросы стоимости, описанные в предыдущем пункте, приобретают первостепенное значение. Однако, если компания Trey Research создаст слишком мало экземпляров, то, хотя компания и не будет платить за простаивающие ресурсы, клиенты могут быть недовольны длительным периодом ожидания ответа и медленным обслуживанием, что может привести к потерям в бизнесе.

По этим причинам этот подход, вероятно, не является рентабельным или желательным.

### Реализация ручного масштабирования

Очевидно, что необходимы определенные способы масштабирования решения. С помощью портала управления Windows Azure можно запускать и останавливать экземпляры веб- и рабочих ролей вручную, или даже разворачивать новые экземпляры приложения Orders в центрах данных по всему миру. Решения о том, когда запускать, останавливать или развертывать новые экземпляры, могут быть приняты на основе анализа информации, полученной путем мониторинга приложения. В главе 7 «Мониторинг и управление приложением Orders» содержится больше информации о том, как выполнять такие задачи. Однако это потенциально очень трудоемкий подход, который может потребовать от оператора выбора, когда лучше всего выполнять эти задачи.

Некоторые из этих операций могут быть выполнены в виде сценариев с использованием командлетов Windows Azure PowerShell, но всегда существует вероятность того, что запустив целый ряд дорогих экземпляров для обработки текущей пиковой нагрузки, оператор может забыть остановить их в нужное время, в результате чего компания Trey Research понесет дополнительные расходы.

### Реализация автоматического масштабирования с использованием пользовательской службы

Запуск и остановка экземпляров ролей вручную является слишком неэффективным способом, который может быть чреват ошибками, поэтому компания Trey Research решила рассмотреть разработку автоматизированного решения. Теоретически, это решение должно позволять следовать той же схеме и осуществлять те же действия, что и при ручном подходе, но при этом оно должно быть более надежным и менее трудоемким. Для этого компания Trey Research рассмотрела настройки веб- и рабочих ролей и собрала основные статистические данные, такие как количество одновременных запросов, среднее время отклика, активность процессора и дисков, а также использование памяти. Эта информация может быть получена с помощью службы диагностики Windows Azure и других доступных источников трассировки событий, которые могут работать в облаке. Информация периодически загружается и анализируется пользовательским приложением, работающим в головном офисе

компании Trey Research. Затем это специальное приложение может определять, следует ли запускать и останавливать дополнительные экземпляры роли и в каком центре данных это следует делать.

Недостатком такого подхода является возможная сложность локального приложения. Могут понадобиться значительные усилия для разработки, построения и тщательного тестирования такого приложения (особенно логики, которая определяет, следует ли запускать и останавливать экземпляры роли). Кроме того, сбор диагностической информации и ее загрузка из каждого центра данных может добавить заметную нагрузку на каждую роль, что может повлиять на производительность.

**Мнение По**

Также доступны внешние службы, которые могут управлять автоматическим масштабированием. Эти службы позволяют избежать расходов на разработку собственных решений, но вы должны предоставлять им сертификаты управления Windows Azure, чтобы они могли получить доступ к экземплярам роли, что может быть неприемлемо для вашей организации.

### Реализация автоматического масштабирования с помощью функционального блока для автоматического масштабирования из библиотеки Enterprise Library

Функциональный блок для автоматического масштабирования (Autoscaling Application Block) из библиотеки Enterprise Library предоставляет инструмент, который вы можете интегрировать непосредственно в веб- и рабочие роли, работающие в облаке, а также в локальные приложения. Данный компонент является частью пакета интеграции Microsoft Enterprise Library 5.0 для Windows Azure и может автоматически масштабировать приложение Windows Azure или работать на основе правил, определяемых вами специально для приложения или службы. Вы можете использовать эти правила, чтобы обеспечить своему приложению или службе необходимую пропускную способность в ответ на изменение рабочей нагрузки и в то же время минимизировать и контролировать расходы на размещение. Данный функциональный блок позволяет облачному приложению запускать и останавливать экземпляры роли, изменять настройки конфигурации, чтобы регулировать функциональность приложения и сократить использование ресурсов, а также отправлять уведомления в соответствии с определенным графиком.

Основные преимущества этого подхода включают низкую стоимость внедрения и простоту использования. Все, что нужно сделать, это предоставить информацию о конфигурации, которая определяет условия (на основе графика и показателей производительности), при которых блок будет применять масштабирование экземпляра или меры регулирования.

## Как компания Trey Research контролирует эластичность в приложении Orders

Для упрощения установки и настройки, а также уменьшения необходимых условий и требований для пользователей к созданию учетных записей Windows Azure, пример решения Trey Research, предоставленный с этим руководством, предназначен для развертывания в одном центре данных и не настроен на поддержку автоматического масштабирования или перенаправления запросов. Таким образом, разделы этой главы, описывающие, как Trey Research внедрила функциональный блок для автоматического масштабирования из библиотеки Enterprise Library и диспетчер трафика Windows Azure, предоставляются только в информационных целях.

Компания Trey Research решила использовать функциональный блок для автоматического масштабирования из библиотеки Enterprise Library с целью запуска и остановки экземпляров веб- и рабочих ролей при изменении нагрузки со стороны пользователей. Первоначально компания развернула приложение Orders и сделала его доступным для небольшого, но статистически значимого количества пользователей, равномерно распределенных по всей географической территории ожидаемого рынка. Путем сбора статистической информации об использовании на основе этого пилотного решения, компания Trey Research определила, как работает система в определенное часы рабочего дня, и определила периоды, в течение которых производительность ухудшалась из-за повышенной нагрузки от клиентов. В частности, эта компания заметила, что:

- Многие клиенты, как правило, размещают свои заказы ближе к концу рабочего дня (между 15:30 и 20:30 по центральному поясному времени, что позволяет распределить пользователей по всей территории Соединенных Штатов), причем особенно много заказов размещается между 17:30 и 18:30.
- По пятницам пиковая нагрузка, как правило, начинается и заканчивается на два часа раньше (с 13:30 до 18:30 по центральному поясному времени).
- В выходные дни очень немногие клиенты размещали заказы.

Чтобы предусмотреть подобные моменты повышения и падения нагрузки, компания Trey Research решила внедрить функциональный блок для автоматического масштабирования из библиотеки Enterprise Library следующим образом:

- Разработчики настроили правила ограничения для приложения с целью запуска трех дополнительных экземпляров веб- и рабочих ролей в каждом центре данных в 15:15 по центральному поясному времени (новые экземпляры становятся доступны в течение 10 или 15 минут) и для остановки этих экземпляров в 20:30 с понедельника по четверг.
- В 17:15 приложение запускает еще два экземпляра каждой роли, которые прекращают работу в 18:30.
- По пятницам время дополнительных запусков и остановок сдвигается на два часа раньше.
- Чтобы справиться с неожиданно возрастающей нагрузкой, компания Trey Research также настроила правила реагирования для контроля количества запросов клиентов таким образом, чтобы запускать дополнительные экземпляры, если средняя загрузка процессора для веб-роли превышает 85 % в течение 10 минут и более, максимум до 12 экземпляров в каждом центре данных. Когда загрузка процессора падает ниже 50 %, экземпляры прекращают работать, оставляя не более двух экземпляров в каждом центре данных.
- В выходные дни система ограничивается четырьмя экземплярами каждой роли в каждом центре данных, и любые дополнительные экземпляры, превышающие это количество, прекращают работать, чтобы снизить эксплуатационные расходы.
- Когда система находится в неактивном состоянии или слегка загружена, она возвращается к своей базовой конфигурации, состоящей из двух экземпляров каждой роли в каждом центре данных.

### Размещение функционального блока для автоматического масштабирования

Функциональный блок для автоматического масштабирования контролирует производительность одной или нескольких ролей, запуск и остановку ролей, применение регулирующих изменений конфигурации или отправку уведомлений, как указано в различных ограничивающих правилах и правилах

реагирования. Функциональный блок для автоматического масштабирования также генерирует диагностическую информацию и фиксирует точки данных, указывая выполненную работу. Дополнительная информация о собранных данных представлена в статье «Autoscaling Application Block Logging» на сайте http://msdn.microsoft.com/en-us/library/hh680883(v=pandp.50).aspx.

Для выполнения этой задачи функциональный блок для автоматического масштабирования использует объект **Autoscaler** (определенный в пространстве имен **Microsoft.Practices.EnterpriseLibrary.WindowsAzure.Autoscaling**), и вы должны подготовить этот объект для запуска во время работы своего приложения. Решение Trey Research выполняет эту задачу в методе **Run** класса **WorkerRole** (в проекте **Orders.Workers**), а останавливает объект **Autoscaler** в методе **OnStop**:

**C#**

```
public class WorkerRole :RoleEntryPoint
{
  private Autoscalerautoscaler;
  ...
  public override void Run()
  {
this.autoscaler = EnterpriseLibraryContainer.Current.
GetInstance<Autoscaler>();
this.autoscaler.Start();
    ...
  }
  ...
  public override void OnStop()
  {
this.autoscaler.Stop();
    ...
  }
  ...
}
```

Информация о контролируемых ролях, учетная запись хранения диагностических данных и местонахождение правил, определяющих поведение объекта **Autoscaler**, указываются в секции **<serviceModel>** в файле хранения сервисной информации. Этот файл был загружен в хранилище BLOB-объектов и сохранен в BLOB-объекте, который описан в секции **<serviceInformationStores>** файла app.config для рабочей роли. Более подробная информация и пример определения модели службы для функционального блока с целью автоматического масштабирования представлены в главе 5 «Обработка заказов в решении Trey Research» руководства разработчика по пакету интеграции Enterprise Library 5.0 для Windows Azure по адресу http://msdn.microsoft.com/en-us/library/hh680942(PandP.50).aspx.

### Определение правил автоматического масштабирования

Решение Trey Research реализует комбинацию правил ограничения и реагирования. Ограничивающие правила задают расписание, используемое объектом **Autoscaler**, в дополнение к максимальному и минимальному количеству экземпляров ролей в течение каждого запланированного периода. Объект **AutoScaler** инициирует создание экземпляров веб- и рабочих ролей или останавливает существующие экземпляры, когда расписание изменяет границы и количество экземпляров превышает новые границы. Правила реагирования запускают дополнительные экземпляры веб-ролей или останавливают их, в зависимости от загрузки процессора веб-роли. В качестве начальной отправной точкой в решении Trey Research определен следующий набор правил:

**XML**

```
<?xmlversion="1.0" encoding="utf-8" ?>
<rules
xmlns="http://schemas.microsoft.com/practices/2011/
entlib/autoscaling/rules">
<constraintRules>
<rule name="Weekday" enabled="true" rank="10">
<timetablestartTime="00:00:00" duration="23:59:59"
utcOffset="-06:00">
<weekly days=
              "Monday Tuesday Wednesday Thursday
Friday"/>
</timetable>
<actions>
<rangetarget="Orders.Workers"
min="2" max="12"/>
<rangetarget="Orders.Website"
min="2" max="12"/>
</actions>
</rule>
<rule name="Weekend" enabled="true" rank="10">
<timetablestartTime="00:00:00" duration="23:59:59"
utcOffset="-06:00">
<weekly days="SundaySaturday"/>
</timetable>
<actions>
<rangetarget="Orders.Workers"
min="2" max="4"/>
<rangetarget="Orders.Website"
min="2" max="4"/>
</actions>
</rule>
<rule name="MondayToThursday" enabled="true" rank="2">
<timetablestartTime="15:15:00" duration="05:15:00"
utcOffset="-06:00">
<weekly days="Monday Tuesday Wednesday Thursday"/>
</timetable>
<actions>
<rangetarget=" Orders.Workers"
min="4" max="12"/>
<rangetarget=" Orders.Website"
min="4" max="12"/>
</actions>
</rule>
<rule name="MondayToThursdayPeak" enabled="true"
rank="3">
<timetablestartTime="17:15:00" duration="03:15:00"
utcOffset="-06:00">
<weekly days="Monday Tuesday Wednesday Thursday"/>
</timetable>
<actions>
<rangetarget=" Orders.Workers"
min="6" max="12"/>
```

```
<rangetarget=" Orders.Website"
min="6" max="12"/>
</actions>
</rule>
<rule name="Friday" enabled="true" rank="2">
<timetablestartTime="13:15:00" duration="05:15:00"
utcOffset="-06:00">
<weekly days="Friday"/>
</timetable>
<actions>
<rangetarget=" Orders.Workers"
min="6" max="12"/>
<rangetarget=" Orders.Website"
min="6" max="12"/>
</actions>
</rule>
<rule name="FridayPeak" enabled="true" rank="3">
<timetablestartTime="15:15:00" duration="03:15:00"
utcOffset="-06:00">
<weekly days="Friday"/>
</timetable>
<actions>
<rangetarget=" Orders.Workers"
min="7" max="12"/>
<rangetarget=" Orders.Website"
min="7" max="12"/>
</actions>
</rule>
</constraintRules>
<reactiveRules>
<rule name="HotCPU" enabled="true" rank="4">
<when>
<greateroperand="CPU" than="85" />
</when>
<actions>
<scaletarget="Orders.Website"
by ="1"/>
</actions>
</rule>
<rule name="CoolCPU" enabled="true" rank="4">
<when>
<lessoperand="CPU" than="50" />
</when>
<actions>
<scaletarget="Orders.Website"
by ="-1"/>
</actions>
</rule>
</reactiveRules>
<operands>
<performanceCounteralias="CPU"
source="AccidentReporting_WebRole"
performanceCounterName=
```

```
"\Processor(_Total)\% Processor Time"
timespan="00:10:00"
aggregate="Average"/>
</operands>
</rules>
```

> **Примечание**
>
> Стоимость размещения служб Windows Azure рассчитывается на почасовой основе, причем часть часа оплачивается как целый час. Это означает, что если, например, вы запустили новый экземпляр службы в 14.50 и остановили его в 16.10, то оплата взимается за 3 часа. Вы должны иметь это в виду при настройке расписания объекта Autoscaler. Более подробная информация о ценообразовании представлена на странице «Pricing Overview» на сайте Windows Azure по адресу http://www.windowsazure.com/en-us/pricing/details/.

Правила были загружены в хранилище BLOB-объектов и сохранены в BLOB-объекте, который описан в секции **<rulesStores>** файла app.config для рабочей роли.

Операнд **CPU**, участвующий в правилах реагирования, рассчитывает среднюю загрузку процессора за 30-минутный период, используя счетчик производительности **\Processor(_Total)\% Processor Time**. Для сбора этой информации веб-роль **Orders.Website** была изменена с помощью следующего кода, выделенного жирным шрифтом в методе **StartDiagnostics** (вызывается из метода **OnStart**) в файле WebRole.cs:

**C#**

```
public class WebRole :RoleEntryPoint
{
  ...
  private static void StartDiagnostics()
  {
    var config =
DiagnosticMonitor.GetDefaultInitialConfiguration();
    ...
config.PerformanceCounters.ScheduledTranferPeriod =
Timespan.FromMinutes(10);

config.PerformanceCounters.DataSources.Add(
      new PerformanceCounterConfiguration
      {
CounterSpecifier =
          @"\Processor(_Total)\% Processor Time",
SampleRate = TimeSpan.FromMinutes(30)
       });

    ...
DiagnosticMonitor.Start(
"DiagnosticsConnectionString", config);
  }
  ...
}
```

Данные счетчика производительности записываются в таблицу **WADPerformanceCountersTable** в хранилище таблиц Windows Azure. Веб-роль должна быть запущена в режиме полного доверия, чтобы можно было успешно записывать данные в эту таблицу.

**Мнение По**

Использование автоматического масштабирования — это интерактивный процесс. Конфигурация, определенная командой Trey Research, постоянно анализируется, а производительность решения находится под постоянным контролем, так как шаблон использования клиентами постоянно усовершенствуется. В будущем операторы могут вносить изменения в настройки и правила автоматического масштабирования, чтобы изменить время и условия, при которых создаются и уничтожаются дополнительные экземпляры.

## Управление задержкой сети и максимизация количества подключений к приложению Orders

Компания Trey Research изначально развернула приложение Orders в двух центрах данных: на юге и севере США. Обоснованием такого решения стал тот факт, что центр данных на севере США находится всего в нескольких милях от компании Trey Research. Это гарантирует разумное время отклика сети для приложения обеспечения соответствия нормативным требованиям, размещенного в центре данных (как описано в главе 4 «Реализация надежного обмена сообщениями и информацией в облаке»), хотя большинство клиентов Trey Research, как ожидается, будут расположены в континентальной части Соединенных Штатов.

Поскольку компания Trey Research расширяет свою клиентскую базу, ожидается, что пользователи, которые подключаются к приложению Orders, могут находиться далеко, возможно, на другом континенте. Расстояние между клиентами и физическим местоположением центра, в котором развернуто приложение Orders, может оказать существенное влияние на время отклика системы. Поэтому компания Trey Research считает необходимым принять стратегию, которая позволяет минимизировать это расстояние и уменьшать сетевые задержки для пользователей, которые подключаются к приложению Orders.

Так как клиенты компании находятся по всему миру, компания Trey Research предусмотрела размещение дополнительных экземпляров приложения Orders в центрах данных, которые аналогично распределены. Клиенты смогут подключаться к ближайшему доступному экземпляру приложения. В этой ситуации компании Trey Research необходимо было решить вопрос о том, как направить клиента к наиболее близкому экземпляру приложения Orders?

### Выбор варианта управления задержкой сети и максимизации количества подключений к приложению Orders

Компания Trey Research исследовала целый ряд решений для направления клиентов к наиболее близкому экземпляру приложения Orders, включая развертывание и настройку нескольких DNS-серверов по всему миру (с помощью нескольких сетевых партнеров) на основе DNS-адреса машины, с которой послан запрос клиента. Тем не менее многие из этих решений оказались непрактичными или дорогими, оставив компании Trey Research лишь два варианта, которые описаны в следующих разделах.

### Создание собственной службы для перенаправления трафика

Компания Trey Research исследовала возможность создания собственной службы, через которую будут подключаться клиенты, и последующего развертывания этой службы в облаке. Целью этой службы будет анализ каждого запроса и его перенаправление в то приложение Orders, которое работает в наиболее подходящем центре данных. Такой подход позволит компании фильтровать и перенаправлять запросы на основе такого критерия, как IP-адрес каждого запроса. Эта служба сможет также определить, работает ли приложение Orders в каждом центре данных, и если оно в настоящее время недоступно, то служба сможет прозрачно перенаправлять запросы пользователей к функционирующим экземплярам приложения. Кроме того, служба может попытаться распределять запросы равномерно между всеми центрами данных, которые находятся на одинаковом расстоянии (в сетевых терминах) к клиенту, реализуя механизм балансировки нагрузки для исключения ситуаций, когда один экземпляр приложения Orders чрезмерно перегружен, а другие простаивают.

Этот тип собственной службы является достаточно общим и может быть реализован с помощью класса **System.ServiceModel.Routing.RoutingService** Windows Communication Foundation. Однако спроектировать, разработать и проверить такую службу не так уж просто; а правила маршрутизации, которые определяют алгоритм перенаправления сообщений данной службой, могут быстро усложниться и стать трудными в обслуживании. Кроме того, эту службу необходимо разместить в центре с достаточной мощностью для обработки каждого запроса клиента и с хорошим качеством подключения к сети для всех клиентов. Если мощность службы будет недостаточной, то она станет причиной многих проблем, а если клиенты не смогут подключиться к службе быстро, то преимущества использования этой службы будут сведены к нулю. Кроме того, эта служба представляет собой единую точку ошибки; если она станет недоступной, клиенты не смогут подключиться ни к одному экземпляру приложения Orders.

### Использование диспетчера трафика Windows Azure для маршрутизации запросов клиентов

Диспетчер трафика Windows Azure представляет собой службу Windows Azure, которая позволяет настроить маршрутизацию запросов и балансировку нагрузки на основе заданных политик и настраиваемых правил. Это решение предоставляет механизм маршрутизации запросов к нескольким развертываниям ваших приложений и служб в Windows Azure, независимо от мест расположения центра данных. Приложения или службы могут быть развернуты в одном или нескольких центрах данных.

Диспетчер трафика является в действительности распознавателем DNS. При использовании диспетчера трафика веб-браузеры и службы, имеющие доступ к вашему приложению, будут отсылать запрос DNS диспетчеру трафика с целью получения IP-адреса конечной точки, к которой они будут подключаться, точно также, как они бы подключались к любому другому веб-сайту или ресурсу.

Диспетчер трафика помогает решить проблемы сетевых задержек и доступности приложений, предоставляя три механизма или политики для маршрутизации запросов:

- Политика **Performance** (производительность) позволяет перенаправлять запросы от пользователей в приложение в ближайшем центре данных. Это не обязательно приложение в том центре данных, который находится ближе всего в географическом смысле, важно, чтобы он обеспечивал самую низкую задержку сети. Диспетчер трафика также обнаруживает проблемные приложения и не перенаправляет к ним, а вместо этого выбирает следующее ближайшее рабочее развертывание приложения.
- Политика **Failover** (отказоустойчивость) позволяет настраивать приоритетный список приложений, и диспетчер трафика будет направлять запросы к первому приложению по списку, которое по его определению отвечает на запросы. Если данное приложение

не работает, то диспетчер трафика перенаправит запросы к следующему по списку приложению и т. д.

- Политика **Round Robin** (циклический перебор) перенаправляет запросы к каждому приложению по очереди; но если приложение оказывается нерабочим, то служба не перенаправляет запросы к нему. Эта политика выравнивает нагрузку на каждом приложении, но может не предоставлять пользователям наилучшее время отклика, так как она игнорирует относительное местоположение пользователя и центра данных.

---

Вы можете выбирать, какая из этих политик наиболее подходит для ваших требований; Performanc — для минимизации задержки в сети, Failover — для максимальной доступности или Round Robin — для равномерного распределения запросов (и, возможно, в результате для улучшения времени отклика).

Диспетчер трафика управляется и поддерживается корпорацией Microsoft, и сама служба размещается в центрах данных этой компании. Это означает, что отсутствуют накладные расходы на обслуживание.

## Как компания Trey Research минимизирует задержки сети и максимизирует количество подключений к приложению Orders

Использование функционального блока для автоматического масштабирования из библиотеки Enterprise Library помогает гарантировать достаточное количество экземпляров веб- и рабочих ролей приложения Orders, работающих для обслуживания данного количества клиентов, подключенных к конкретному центру данных в определенный момент времени. Помня об этом, эксплуатационный персонал компании Trey Research решил использовать диспетчер трафика просто для маршрутизации запросов клиентов к ближайшему реагирующему центру данных, реализовав политику Performance.

**Мнение Бхарата**

Выбор политики Performance был сделан очень легко; политика Failover не подходит для сценария компании Trey Research, а функциональный блок для автоматического масштабирования из библиотеки Enterprise Library гарантирует, что соответствующее количество экземпляров ролей приложения Orders будет доступно в каждом центре данных для поддержки высокой производительности, поэтому политика Round Robin также не нужна.

Реализация политики Round Robin может иметь пагубные последствия для клиентов, поскольку они могут быть направлены в более отдаленные центры данных, что влечет дополнительные задержки сети и влияет на время отклика приложения. Кроме того, политика Round Robin предположительно может направить два последовательных запроса от одного клиента в различные центры данных, что может привести к путанице, если данные, которые кэшируются в каждом центре данных, не является полностью совместимыми. Политика Performance имеет преимущества в сокращении задержки сети, обеспечивая при этом с гораздо большей вероятностью направление запросов одного клиента в один центр данных.

Эксплуатационный персонал настроил эту политику таким образом, чтобы она включала DNS-адреса приложения Orders, развернутого в центрах данных на севере и юге США, а также чтобы осуществлялся мониторинг домашней страницы веб-приложения для определения доступности. Эксплуатационный персонал выбрал DNS-префикс **ordersapp.treyresearch** и сопоставил результирующий адрес

(**ordersapp.treyresearch.trafficmanager.net**) с публичным адресом, который используют клиенты, **store.treyresearch.net**. Таким образом, клиент, подключаясь к URL-адресу **http://store.treyresearch.net**, прозрачно перенаправляется диспетчером трафика к приложению Orders, которое работает в центре данных на севере или юге США. На рисунке 1 показана структура этой конфигурации.

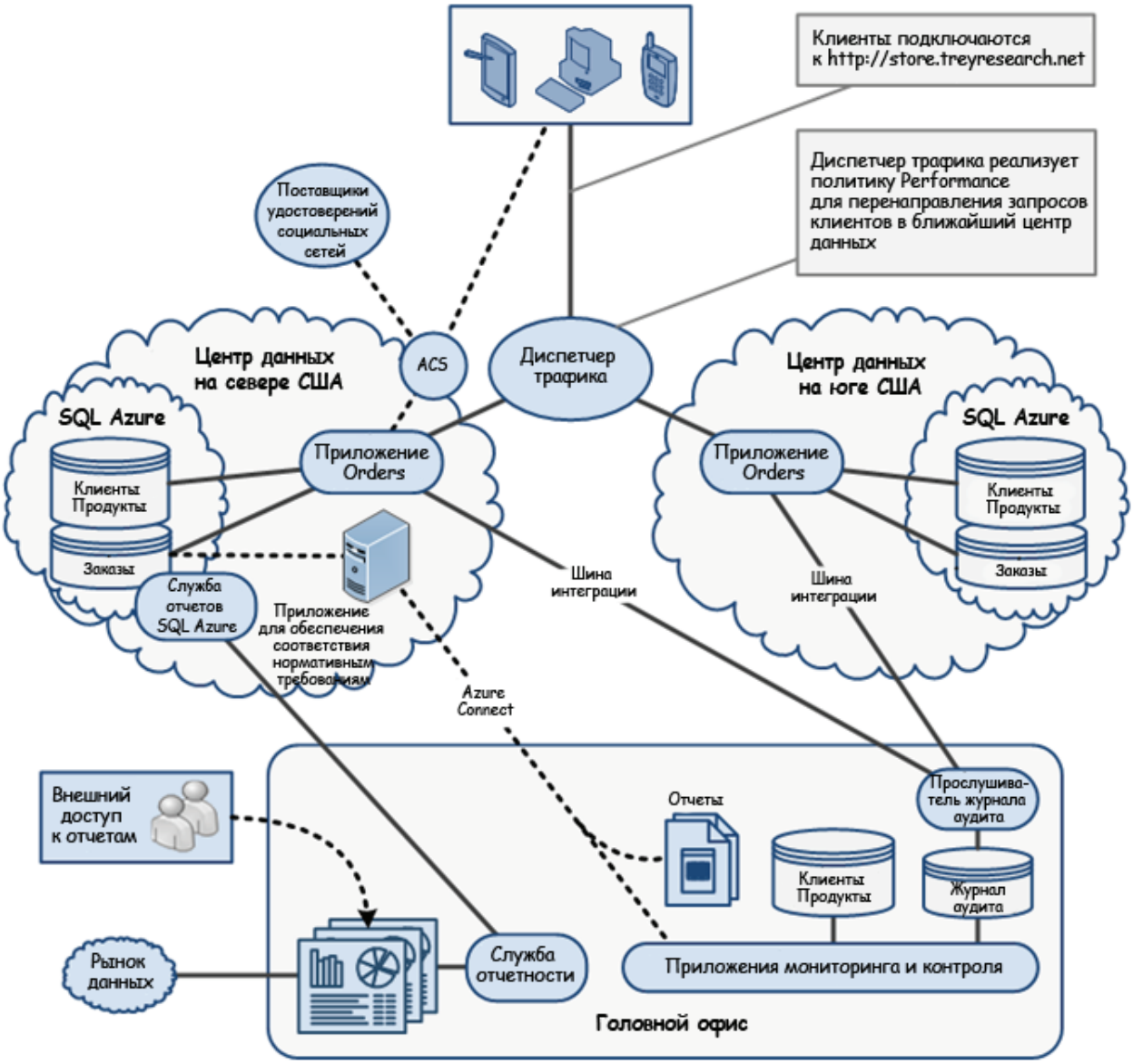



**Рисунок 1**

### Как компания Trey Research использует диспетчер трафика Windows Azure

Обратите внимание, что приложение Orders в обоих центрах данных должно подключаться к службе прослушивателя журнала аудита головного офиса. Оба развертывания приложения Orders должны также подключаться ко всем транспортным партнерам компании Trey Research; хотя, для простоты, это не показано на схеме. Некоторые функции приложения, такие как использование службы отчетов платформы SQL Azure и развертывание приложения для обеспечения соответствия нормативным требованиям в роли виртуальной машины Windows Azure, не дублируются в обоих центрах данных. Данные приложения Orders синхронизированы между обоими центрами данных, поэтому один экземпляр службы отчетов и приложение для обеспечения соответствия нормативным требованиям обеспечат необходимый результат без каких-либо дополнительных затрат на размещение и обслуживание.

Однако проектировщики в компании Trey Research поняли, что использование механизма, который может направлять пользователей к различным развертываниям приложения в различных центрах данных, может привести к некоторым последствиям. Например, такие данные, как текущая корзина пользователя, как правило, хранятся в памяти или в облачном хранилище (таком как хранилище таблиц Windows Azure или SQL Azure). Когда пользователь перенаправляется к другому центру данных, то эти данные теряются, если только приложение специально не синхронизирует данные между всеми центрами данных.

Кроме того, если специалисты компании Trey Research настроили ACS в более чем одном центре данных для защиты от проблем аутентификации в случае, если служба ACS в одном центре данных недоступна, перенаправление пользователей в другой центр данных будет означать, что они должны будут повторно войти в систему.

Однако компания Trey Research считает, что оба этих сценария вряд ли будут происходить достаточно часто, чтобы стать проблемой.

## Оптимизация времени отклика приложения Orders

Windows Azure является хорошо масштабируемой платформой, которая обеспечивает высокую производительность для приложений. Однако доступная вычислительная мощность не гарантирует того, что приложение будет быстро реагировать. Приложение, которое предназначено для работы в серийном режиме, не будет наилучшим образом использовать эту платформу и может значительное время пребывать в заблокированном состоянии, ожидая завершения медленных зависимых операций. Решение должно выполнять эти операции асинхронно, и методы, которые компания Trey Research приняла для реализации этого подхода, были описаны в главе 4 «Реализация надежного обмена сообщениями и информацией в облаке» и главе 5 «Обработка заказов в решении Trey Research».

Помимо проектирования и реализации логики приложения, ключевым фактором, который регулирует время отклика и пропускную способность службы, является скорость, с которой служба получает доступ к ресурсам и необходимым данным. В случае приложения Orders, основным источником данных является база данных SQL Azure, содержащая данные о клиенте, заказе и продукте. В главе 2 «Развертывание данных и приложения Orders в облаке» рассказывалось, как компания Trey Research размещает данные в каждом центре данных, чтобы попытаться свести к минимуму нагрузку на сеть, связанную с доступом к этой информации. Однако базы данных работают по-прежнему относительно медленно по сравнению с другими формами хранения данных. Таким образом, перед компанией Trey Research был поставлен вопрос: «Как обеспечить масштабируемый, надежный и быстрый доступ к данным о клиентах, заказах и продуктах, так как это может стать ключом к минимизации времени отклика приложения Orders?».

### Как оптимизировать время отклика приложения Orders

По результатам более подробных исследований проблем, связанных со временем отклика, компания Trey Research обнаружила, что существует два взаимодополняющих подхода (и оба они могут использоваться в соответствующем случае).

#### Реализация кэширования Windows Azure

Windows Azure Caching — это служба, которая позволяет кэшировать данные в облаке и обеспечивает масштабируемый, надежный и общий доступ к этим данным.

После профилирования приложения Orders разработчики Trey Research обнаружили, что приложение тратит значительную часть времени на отправку запросов в базу данных SQL Azure, и задержки, связанные с подключением к этой базе данных, а также с обработкой запросов и обновлением данных в самой базе данных составляют большую часть этого времени. С помощью кэширования данных службой Windows Azure Caching компания надеялась уменьшить затраты, связанные с многократным доступом к удаленным данным, устранить задержки в сети, связанные с удаленным доступом к данным, а также улучшить время отклика приложений, ссылающихся на эти данные.

Издержки, связанные с отправкой запросов и обновлением данных в SQL Azure, не являются недостатками этой системы управления базами данных (СУБД). Все СУБД, которые поддерживают одновременный многопользовательской доступ, должны обеспечивать согласованность и целостность данных, как правило, с помощью сериализации одновременных запросов от разных пользователей и блокировки данных. SQL Azure полностью соответствует этим требованиям. Однако получение данных из кэша не связано с такими затратами; данные просто извлекаются или обновляются. За эффективность приходится платить, так как в приложение теперь должно быть предусмотрено обеспечение целостности и непротиворечивости кэшированных данных.

Кроме того, рано или поздно любые обновления кэшированных данных должны быть скопированы в базу данных, в противном случае кэш и база данных не будут соответствовать друг другу или данные могут быть потеряны.Кэш имеет конечный размер, и служба Windows Azure Caching может исключить данные, если не будет достаточно места, или удалить старые данные, которые находились в кэше в течение длительного периода времени.

Служба кэширования Windows Azure является также платной. Корпорация Microsoft размещает и поддерживает ее в своих центрах данных, а также предоставляет гарантии доступности данной услуги и кэшированных данных, но с пользователей взимается определенная плата в зависимости от размера кэш-памяти и объема трафика считывания или записывания в кэш. Более подробная информация представлена в статье «Caching, based on cachesizepermonth» на сайте Windows Azure по адресу http://www.windowsazure.com/en-us/pricing/details/.

### Настройка сети для доставки контента

Сеть доставки контента Windows Azure (Content Delivery Network, CDN) — это служба, предназначенная для улучшения времени отклика веб-приложений за счет кэширования статических данных, сгенерированных размещенными службами, а также часто используемых данных BLOB-объектов, которые расположены ближе всего к запрашивающим их пользователям. Хотя служба кэширования Windows Azure в основном используется для улучшения производительности веб-приложений и служб, работающих в облаке, пользователи будут часто вызывать эти веб-приложения и службы со своих компьютеров, либо с помощью специального приложения, которое подключается к ним, либо с помощью веб-браузера. Данные, возвращаемые веб-приложением или службой, могут иметь существенный объем, и если пользователь находится очень далеко, то может понадобиться значительное время для пересылки этих данных на компьютер пользователя. Служба CDN позволяет кэшировать вывод веб-страниц и часто запрашиваемых данных в различных местах по всему миру. Когда пользователь делает запрос, веб-контент и данные могут быть предоставлены из наиболее оптимального расположенного центра на основе текущего объема трафика в различных интернет-узлах, через которые направляется запрос.

**Примечание**

Более детальная информация, примеры и упражнения, демонстрирующие настройку службы CDN, представлены на сайте MSDN в статье «Windows Azure CDN» по адресу http://msdn.microsoft.com/en-us/gg405416. Также в главе 3 «*Accessing the Surveys Application*»

руководства «*Developing Applications for the Cloud, 2nd Edition*» приводятся подробные сведения о дальнейшей реализации. Подробная информация представлена на сайте http://msdn.microsoft.com/en-us/library/ff966499.aspx.

Хотя CDN и является полезной технологией, разработчики Trey Research после проведения исследований решили не применять данную технологию в текущей версии приложения Orders CDN идеально подходит для кэширования веб-страниц со статическим контентом и данных BLOB-объектов для вывода или потоковой передачи в клиентское приложение, но многие страницы, создаваемые приложением Orders, могут быть относительно динамичными, и приложение не хранит или исключает данные BLOB-объектов.

## Как компания Trey Research оптимизирует время отклика приложения Orders

Приложение Orders использует несколько типов данных: информацию о клиентах, заказах и каталог продукции. Информация о заказах относительно динамична, а к данным о клиентах доступ осуществляется редко по сравнению с другими данными (только когда клиент входит в систему). Кроме того, данные о клиентах и о заказах, как правило, не запрашиваются одновременно разными экземплярами приложения Orders. Однако данные каталога продукции запрашиваются каждым экземпляром приложения Orders при входе пользователя в систему. Эти данные также достаточно статичны; информация о продуктах обновляется очень редко. Кроме того, каталог продукции может состоять из большого количества позиций. По этим причинам разработчики компании Trey Research склонились в пользу кэширования каталога продукции с помощью общего кэша Windows Azure для каждого центра данных, при этом они решили, что кэширование данных о заказах и клиентах принесет мало пользы.

### Определение и настройка службы кэширования Windows Azure

Служба кэширования Windows Azure работает в облаке, и приложение должно действительно подключаться только к экземпляру службы кэширования Windows Azure Caching, который находится в том же центре данных, в котором размещается код приложения. Таким образом, компания Trey Research использовала службу Windows Azure Caching для создания отдельных кэшей в центрах данных на севере и юге США под названиями **TreyResearchCacheUSN** (для центра данных на севере США) и **TreyResearchCacheUSS** (для центра данных на юге США). Этот подход гарантирует, что у каждого кэша будет уникальное и легко распознаваемое имя. Разработчики оценили, что 128 Мб кэша (минимальный доступный и самый дешевый размер) будет вполне достаточно. Однако размер кэшей можно легко увеличить при необходимости без ущерба для работы приложения Orders.

Веб-приложение, реализованное в проекте **Orders.Website**, определяет параметры настройки для доступа к кэшу в файле конфигурации службы для решения (ServiceConfiguration.csfg).

Пример решения Trey Research развернут только в одном центре данных, а кэш называется **TreyResearchCache**.

**XML**

```
<?xmlversion="1.0" encoding="utf-8" ?>
<ServiceConfigurationserviceName="Orders.Azure" ...>
  ...
<Role name="Orders.Website">
    ...
<ConfigurationSettings>
      ...
```

```
<Setting name="CacheHost"
        value="TreyResearchCache.cache.windows.net" />
<Setting name="CachePort" value="22233" />
<Setting name="CacheAcsKey" value="[data omitted]" />
<Setting name="IsLocalCacheEnabled" value="false" />
<Setting name="LocalCacheObjectCount" value="1000" />
<Setting name="LocalCacheTtlValue" value="60" />
<Setting name="LocalCacheSync"
        value="TimeoutBased" />
      ...
</ConfigurationSettings>
    ...
</Role>
</ServiceConfiguration>
```

### Синхронизация кэшей и баз данных в приложении Orders

Приложение Orders было изменено таким образом, чтобы получать и обновлять данные из облачного экземпляра кэша Windows Azure, только извлекая данные из базы данных SQL Azure, если данные в настоящее время отсутствуют в кэше. Любые изменения, внесенные в кэшированные данные, будут скопированы обратно в SQL Azure. В следующих подразделах описано, как компания Trey Research реализовала этот подход.

Компании Trey Research также пришлось учитывать влияние кэширования на стратегию синхронизации данных. Каждый центр данных имеет копию базы данных SQL Azure, содержащую данные о клиентах, заказах и продуктах. Приложение Orders может изменять информацию о клиентах и заказах, и когда происходят такие изменения, кэшированная копия этой информации копируется обратно в облачную базу данных SQL Azure. Эта база данных впоследствии синхронизируется с базами данных SQL Azure в других центрах данных, как описано в главе 2 «Развертывание данных и приложения Orders в облаке».

Однако предположим, что данные о заказах или клиентах были кэшированы приложением Orders, работающим в центре данных на севере США, и те же данные запрашиваются и кэшируются приложением Orders, работающем в центре данных на юге США. В этот момент два кэша хранят одинаковые данные. Если информация в центре данных на севере США изменяется и записывается в базу данных SQL Azure в центре данных на севере США, и эта база данных впоследствии синхронизируется с центром данных на юге США, то кэшированные данные в центре данных на юге США в этот момент являются устаревшими. Однако, когда кэшированные данные, хранящиеся в центре данных на юге США, устареют или удаляются, то кэш будет заполнен новыми данными при следующем запрашивании.

Таким образом, хотя кэширование и позволяет улучшить время отклика для многих операций, оно также может привести к проблемам согласованности, если два экземпляра элемента данных не идентичны. Таким образом, приложения, которые используют кэширование, должны быть разработаны так, чтобы у них была возможность обработки устаревших данных, которые в итоге станут согласованными.

Эта проблема может обостриться, если одни и те же кэшированные данные обновляются одновременно в центрах данных на севере и юге США. Служба SQL Azure Data Sync гарантирует согласованность данных между различными базами данных, но по крайней мере один из кэшей будет содержать противоречивые данные. Для получения дополнительного руководства и рекомендаций о том, как решать эти проблемы, см. раздел «Рекомендации по использованию службы Windows Azure Caching» в «Приложении Д. Максимизация масштабируемости, доступности и производительности».

### Извлечение данных и управление ими в приложении Orders

Приложение Orders использует набор классов для хранения и извлечения каждого типа информации, на которые оно ссылается. Данные классы расположены в папке DataStores проекта **Orders.Website**. Например, класс **ProductStore** в файле ProductStore.cs предоставляет методы для запроса информации о продукции. Эти методы определены интерфейсом **IProductsStore**:

**C#**

```
public interfaceIProductStore
{
IEnumerable<Product>FindAll();
  Product FindOne(int productId);
}
```

Метод **FindAll** возвращает список всех доступных продуктов из базы данных SQL Azure, а метод **FindOne** извлекает продукт по заданному идентификатору. По такому же принципу класс **OrderStore** реализует интерфейс **IOrdersStore**, который определяет методы для извлечения и управления заказами. Ни один из этих классов не реализует кэширование.

### Реализация функциональности кэширования для каталога продукции

Проект **Orders.Website** содержит общую библиотеку классов для кэширования данных, расположенную в папке DataStores\Caching. Данная библиотека может кэшировать любые элементы данных, определенные типами в папке DataStores, но по изложенным выше причинам кэширование реализовано только для класса **ProductStore**.

В папке DataStores\Caching находятся интерфейс **ICachingStrategy**, класс **CachingStrategy** и класс **ProductStoreWithCache**. Эти классы описаны в следующих разделах.

#### *Интерфейс ICachingStrategy*

Это простой интерфейс, который абстрагирует функциональность кэширования, реализованную библиотекой. Этот интерфейс предоставляет свойство **DefaultTimeout** и метод **Get** следующим образом:

**C#**

```
public interfaceICachingStrategy
{
  TimeSpan DefaultTimeout
  {
get;
set;
  }

  object Get<T>(string key, Func<T>fallbackAction,
                TimeSpan? timeout) where T : class;
}
```

Параметр **key** метода **Get** задает уникальный идентификатор объекта, который нужно получить из кэша. Если объект в данный момент не кэширован, то параметр **fallbackAction** задает делегата для выполняемого метода, который должен извлечь соответствующие данные, а параметр **timeout** задает время жизни объекта, если он добавляется в кэш. Если параметр timeout равен нулю, реализация этого интерфейса должна устанавливать для времени жизни объекта значение, указанное в свойстве **DefaultTimeout**.

### *Класс CachingStrategy*

Данный класс реализует интерфейс **ICachingStrategy**. Конструктор для этого класса использует программный интерфейс кэширования Windows Azure для аутентификации и подключения к кэшу Windows Azure, используя значения, предоставляемые в качестве параметров (веб-приложение извлекает эти значения из файла конфигурации службы и вызывает конструктор с помощью фреймворка Unity, как описано далее в этой главе в разделе «Создание экземпляра объекта ProductsStoreWithCache и его использование»).

Метод **Get** класса **CachingStrategy** отправляет запросы в кэш, используя определенный ключ, и, если такой объект найден, он возвращается. Если нужный объект не найден, то данный метод вызывает делегата, чтобы извлечь недостающие данные, и добавляет их в кэш, указывая либо значение времени ожидания, предоставленное в качестве параметра для метода **Get** (если оно не равно нулю), либо значение времени ожидания, используемое по умолчанию для объекта **CachingStrategy**. В следующем фрагменте кода показаны важные элементы этого класса:

**C#**

```
public class CachingStrategy :
ICachingStrategy, IDisposable
{
  private readonly RetryPolicycacheRetryPolicy;
  private DataCacheFactorycacheFactory;

  ...
  private TimeSpan defaultTimeout =
TimeSpan.FromMinutes(10);

  public CachingStrategy(string host, int port,
string key, bool isLocalCacheEnabled,
    long objectCount, int ttlValue, string sync)
{
    // Объявление массива для размещения кэша.
    var servers = new DataCacheServerEndpoint[1];

servers[0] = new DataCacheServerEndpoint(
host, port);

    // Установка конфигурации DataCacheSecurity.
    var secureAcsKey = new SecureString();
foreach (char a in key)
{
secureAcsKey.AppendChar(a);
    }
secureAcsKey.MakeReadOnly();
    var factorySecurity =
      new DataCacheSecurity(secureAcsKey);

    // Установка конфигурации DataCacheFactory.
    var factoryConfig =
      new DataCacheFactoryConfiguration
      {
        Servers = servers,
SecurityProperties = factorySecurity
```

```
      };
    ...
this.cacheFactory =
      new DataCacheFactory(factoryConfig);

this.cacheRetryPolicy = RetryPolicyFactory.
GetDefaultAzureCachingRetryPolicy();
    ...
  }

  public TimeSpan DefaultTimeout
  {
get{ return this.defaultTimeout; }
set{ this.defaultTimeout = value; }
  }

  public virtual object Get<T>(string key,
    Func<T>fallbackAction, TimeSpan? timeout)
where T : class
    {
      ...
      try
      {
        var dataCache =
this.cacheFactory.GetDefaultCache();

        var cachedObject =
this.cacheRetryPolicy.ExecuteAction(
() =>dataCache.Get(key));

        if (cachedObject != null)
        {
          ...
          return cachedObject;
        }
        ...
        var objectToBeCached = fallbackAction();

        if (objectToBeCached != null)
        {
          try
          {
this.cacheRetryPolicy.ExecuteAction(() =>
dataCache.Put(key, objectToBeCached,
timeout != null ?
timeout.Value :this.DefaultTimeout));
            ...
            return objectToBeCached;
          }
          ...
        }
      }
    }
```

```
  }
  ...
}
```

Обратите внимание, что этот класс ловит неустойчивые неисправности, которые могут возникнуть при извлечении элемента из кэша, с помощью функционального блока для обработки неустойчивых неисправностей. Статический метод **GetDefaultAzureCachingRetryPolicy** класса **RetryPolicyFactory**, на который есть ссылка в конструкторе, возвращает политику по умолчанию для определения неустойчивых исключений кэша и предоставляет конструкцию, показывающую способ обработки данного исключения. По умолчанию политика реализует «стратегию фиксированного интервала повтора», которая определяется блоком обработки неустойчивых неисправностей, а файл web.config содержит настройки этой стратегии для повтора выполнения неудавшихся операций до шести раз с пятисекундной задержкой между попытками.

Свойство **Get** класса **CachingStrategy** вызывает метод **ExecuteAction** объекта политики повторного выполнения, передавая ему делегата, который пытается считать запрашиваемые данные из кэша (это код, который может показать неустойчивую неисправность, и, если необходимо, повторить попытку в зависимости от настроек, определенных в объекте политики повторного выполнения). Если возникает неустойчивая неисправность или чтение из кэша было не выполнено за шесть попыток, стратегия обработки исключений в методе **Get** (отсутствует в представленном выше коде) вернет значение из исходного хранилища, извлеченное путем вызова делегата **fallbackAction**.

### *Класс ProductStoreWithCache*

Данный класс предоставляет версию класса **ProductStore** с кэшированием. Класс реализует интерфейс **IProductsStore**, но внутри использует объект **ICachingStrategy** для извлечения данных в методах **FindAll** и **FindOne**, как показано в следующем фрагменте кода:

**C#**

```
public class ProductStoreWithCache :IProductStore
{
  private readonly IProductStoreproductStore;

  private readonly ICachingStrategycachingStrategy;

public ProductStoreWithCache(
IProductStoreproductStore,
ICachingStrategycachingStrategy)
  {
this.productStore = productStore;
this.cachingStrategy = cachingStrategy;
  }

  public IEnumerable<Product>FindAll()
  {
    ...
    return (IEnumerable<Product>)
this.cachingStrategy.Get(
      "ProductStore/FindAll",
      () =>this.productStore.FindAll(),
TimeSpan.FromMinutes(10));
  }
```

```
  public Product FindOne(int productId)
{
    ...
    return (Product)this.cachingStrategy.Get(
string.Format(
"ProductStore/Product/{0}", productId),
      () =>this.productStore.FindOne(productId),
TimeSpan.FromMinutes(10));
  }
}
```

### Создание экземпляра объекта ProductsStoreWithCache и его использование

Приложение Orders создает объект **ProductsStoreWithCache** с помощью функционального блока Unity Application Block. Статический класс **ContainerBootstrapper** содержит следующий код:

**C#**

```
public static class ContainerBootstrapper
{
public static void RegisterTypes(
IUnityContainercontainer)
  {
    ...
container.RegisterType<IProductStore,
ProductStoreWithCache>(
new InjectionConstructor(
          new ResolvedParameter<ProductStore>(),
new ResolvedParameter<ICachingStrategy>()));
container.RegisterType<ProductStore>();

    // Чтобы изменить стратегию кэширования, замените
    // класс CachingStrategy на стратегию, которую
    // вы хотите использовать.
    var cacheAcsKey = CloudConfiguration.
      GetConfigurationSetting("CacheAcsKey", null);
var port = Convert.ToInt32(CloudConfiguration.
GetConfigurationSetting("CachePort", null));
    var host = CloudConfiguration.
      GetConfigurationSetting("CacheHost", null);

var isLocalCacheEnabled = Convert.ToBoolean(
CloudConfiguration.GetConfigurationSetting(
"IsLocalCacheEnabled", null));
var localCacheObjectCount = Convert.ToInt64(
CloudConfiguration.GetConfigurationSetting(
"LocalCacheObjectCount", null));
var localCacheTtlValue = Convert.ToInt32(
CloudConfiguration.GetConfigurationSetting(
"LocalCacheTtlValue", null));
    var localCacheSync =
CloudConfiguration.GetConfigurationSetting(
"LocalCacheSync", null);

container.RegisterType<ICachingStrategy,
```

```
CachingStrategy> (
        new ContainerControlledLifetimeManager(),
        new InjectionConstructor(host, port, cacheAcsKey,
isLocalCacheEnabled, localCacheObjectCount,
localCacheTtlValue, localCacheSync));
  }
}
```

Данные инструкции регистрируют объекты **ProductStore** и **CachingStrategy**, а функциональный блок Unity Application Block использует их для создания объекта **ProductStoreWithCache**, когда приложение создает экземпляр объекта **IProductStore**. Обратите внимание, что класс **CachingStrategy** настроен таким образом, чтобы использовать класс **ContainerControlledLifetimeManager** фреймворка Unity. Такой подход гарантирует, что объект **CachingStrategy**, используемый приложением, создан в единственном экземпляре, который охватывает время жизни приложения. Это полезно, поскольку создание объекта **DataCacheFactory**, который инкапсулирует класс **CachingStrategy**, достаточно дорогостоящий процесс и требует много времени, поэтому лучшим вариантом является создание единичного экземпляра этого класса, который будет доступен на протяжении всей жизни приложения. Кроме того, параметры для конструктора объекта **CachingStrategy** читаются из файла конфигурации и передаются в класс **CachingStrategy** с помощью фреймворка Unity объекта **InjectionConstructor**.

**Мнение Маркуса**

Метод **RegisterTypes** класса **ContainerBootstrapper** вызывается из метода **SetupDependencies** в файле Global.asax.cs, когда приложение Orders начинает работать. Метод **SetupDependencies** также назначает сопоставителя зависимостей приложения Orders для контейнера Unity, который зарегистрировал эти типы.

Более подробная информация об использовании функционального блока Unity Application Block представлена в статье «Unity Application Block» на сайте http://msdn.microsoft.com/en-us/library/ff647202.aspx.

Класс **StoreController** вызывает метод **FindAll** объекта **ProductStoreWithCache**, когда необходимо извлечь и показать весь каталог продукции, и метод **FindOne**, когда нужно извлечь информацию об одном продукте:

**C#**

```
public class StoreController : Controller
{
  private readonly IProductStoreproductStore;

  public StoreController(IProductStoreproductStore)
  {
    ...
this.productStore = productStore;
  }

  public ActionResult Index()
  {
    var products = this.productStore.FindAll();
return View(products);
  }

  public ActionResult Details(int id)
```

```
{
    var p = this.productStore.FindOne(id);
return View(p);
  }
}
```

Этот код прозрачным образом получает доступ к кэшу Windows Azure, заполняя его, если запрашиваемые данные в настоящее время отсутствуют в кэше. Вы можете изменить конфигурацию кэширования, и даже отменить кэширование данных, если в кэшировании нет никакой пользы, без изменения бизнес-логики приложения Orders. Все, что вам необходимо сделать, это переключить тип для интерфейса **IProductsStore** в классе **ContainerBootstrapper** на **ProductStore**, как отмечено полужирным шрифтом в следующем примере кода:

**C#**

```
public static class ContainerBootstrapper
{
public static void RegisterTypes(
IUnityContainercontainer)
  {
    ...
container.RegisterType<IProductStore,ProductStore>();
    ...
  }
}
```

## Выводы

В этой главе описаны технологии Windows Azure, которые используются компанией Trey Research для повышения масштабируемости, доступности и производительности приложения Orders.

Диспетчер трафика Windows Azure может играть важную роль в сокращении задержки в сети, связанной с отправкой запросов к веб-приложению, прозрачно перенаправляя эти запросы на наиболее подходящие развертывания веб-приложения в зависимости от места нахождения клиента, подающего эти запросы. Диспетчер трафика может также способствовать максимизации доступности, на интеллектуальном уровне определяя, реагирует ли приложение. А если приложение не отвечает, то данная служба будет перенаправлять запросы к другому развертыванию приложения.

Windows Azure предоставляет среду с высокой масштабируемостью для размещения веб-приложений и служб, а функциональный блок для автоматического масштабирования из библиотеки Enterprise Library реализует механизм, который позволяет в полной мере воспользоваться этой масштабируемостью путем мониторинга веб-приложений и автоматического запуска и остановки экземпляров согласно уровню спроса со стороны клиентов.

И наконец, кэширование Windows Azure является важным элементом в повышении скорости отклика веб-приложений и служб. Таким образом, компания Trey Research может кэшировать данные локально по отношению к этим приложениям в том же центре данных. Этот метод в основном устраняет сетевые задержки, связанные с удаленным доступом к данным. Однако, как обнаружила компания, вы должны быть готовы к тому, что придется балансировать между повышением производительности и возможной сложностью, возникающей при обслуживании нескольких копий данных.

## Дополнительная информация

- «Pricing Overview» http://www.windowsazure.com/en-us/pricing/details/
- «Windows Azure Traffic Manager» http://msdn.microsoft.com/en-us/gg197529
- «Windows Azure Service Instances Auto Scaling» http://azureautoscaling.codeplex.com/releases/view/62421
- «Windows Azure Caching Service» http://msdn.microsoft.com/en-us/library/gg278356.aspx
- «Windows Azure CDN» http://msdn.microsoft.com/en-us/gg405416

# Глава 7. Мониторинг и управление приложением Orders

После завершения разработки и внедрения гибридного приложения Orders перед специалистами компании Trey Research встал вопрос, как осуществлять мониторинг и управление приложением, работающим на технологической платформе Windows Azure.

Приложение Orders включает ряд компонентов, созданных с использованием различных технологий, распределенных между множеством узлов и связанных с помощью сетей с различной пропускной способностью и надежностью. Несмотря на всю сложность, для компании Trey Research было очень важно получить возможность контроля работы системы, и в случае сбоя быстро принимать необходимые меры по ее восстановлению. Однако мониторинг сложной системы сам по себе является очень сложной задачей, необходимы инструменты, позволяющие быстро собрать данные о производительности, чтобы проанализировать пропускную способность и определить причины ошибок, сбоев или других проблем в системе. Возникающие проблемы могут существенно отличаться, начиная от простых сбоев, вызванных ошибками приложения в службе, работающей в облаке, проблем со средой размещения отдельных элементов и заканчивая полным отказом системы и потерей связи между компонентами, работающими локально или в облаке.

В этой главе рассматриваются проблемы, связанные с мониторингом приложения Orders, и решения, которые приняла компания Trey Research, для устранения этих проблем.

## Сценарий и контекст

Гибридное приложение Orders состоит из компонентов, которые работают удаленно от локальных служб, включая веб-сайт, фоновый код обработки заказов и базы данных. Приложение также взаимодействует с транспортными партнерами, оно обрабатывает заказы, прослушивает сообщения о состоянии от партнеров и отправляет сообщения в журнал аудита локальной службы.

Проектировщики компании Trey Research должны были решить, как осуществлять мониторинг приложения во время его работы, чтобы администраторы имели возможность измерять производительность, гарантировать, что приложение соответствует соглашению об уровне обслуживания (Service Level Agreements, SLA) и проверять, что оно обеспечивает посетителям допустимое время отклика. Администраторы также должны иметь возможность извлекать данные об ошибках и исключениях, которые возникают во время работы, и отслеживать процессы, способствуя отладке приложения. Перед развертыванием для выполнения многих из этих задач разработчикам пришлось добавить в приложение определенный код.

Компании Trey Research также было необходимо решить, как развернуть приложение в Windows Azure и как управлять такими факторами, как изменение конфигурации и управление отдельными службами Windows Azure, которые используются во время развертывания и работы приложения. Компания Trey Research создала набор сценариев и других исполняемых программ, которые позволяют выполнять задачи такого рода многократно, точно и надежно.

## Мониторинг служб, регистрация действий и измерение производительности

Несмотря на то, что приложение Orders работает удаленно от головного офиса компании Trey Research, администраторы этой компании имеют возможность получать такую же информацию о работе приложения, возникающих исключениях и ошибках, как и во время администрирования приложения, развернутого локально в их центре данных. Однако способ сбора данных и доступ к ним в Windows Azure очень отличается по сравнению с развертыванием на локальном сервере.

Вы можете настроить службу диагностики Windows Azure для сбора данных о производительности и диагностических сведений. Эти данные хранятся в памяти в рабочей или веб-роли, которая контролируется, но они могут передаваться в хранилище Windows Azure по расписанию или по требованию и будут доступны из локальных приложений и решений мониторинга.

> **Мнение Бхарата**
>
> Вы можете настроить механизм службы диагностики Windows Azure, чтобы собирать необходимые данные для мониторинга и отладки приложений, а также передавать эти данные в хранилище Windows Azure и иметь к ним доступ.

На рисунке 1 показан общий вид механизма мониторинга в Windows Azure, а также некоторые способы мониторинга, выбранные компанией Trey Research. Механизм службы диагностики Windows Azure можно настроить для сбора данных из различных источников, таких как журнал событий Windows и счетчики производительности. Можно также использовать механизм протоколирования сторонних разработчиков, например функциональный блок Logging Application Block из библиотеки Enterprise Library или пользовательский код, который записывает события в механизм службы диагностики Windows Azure.

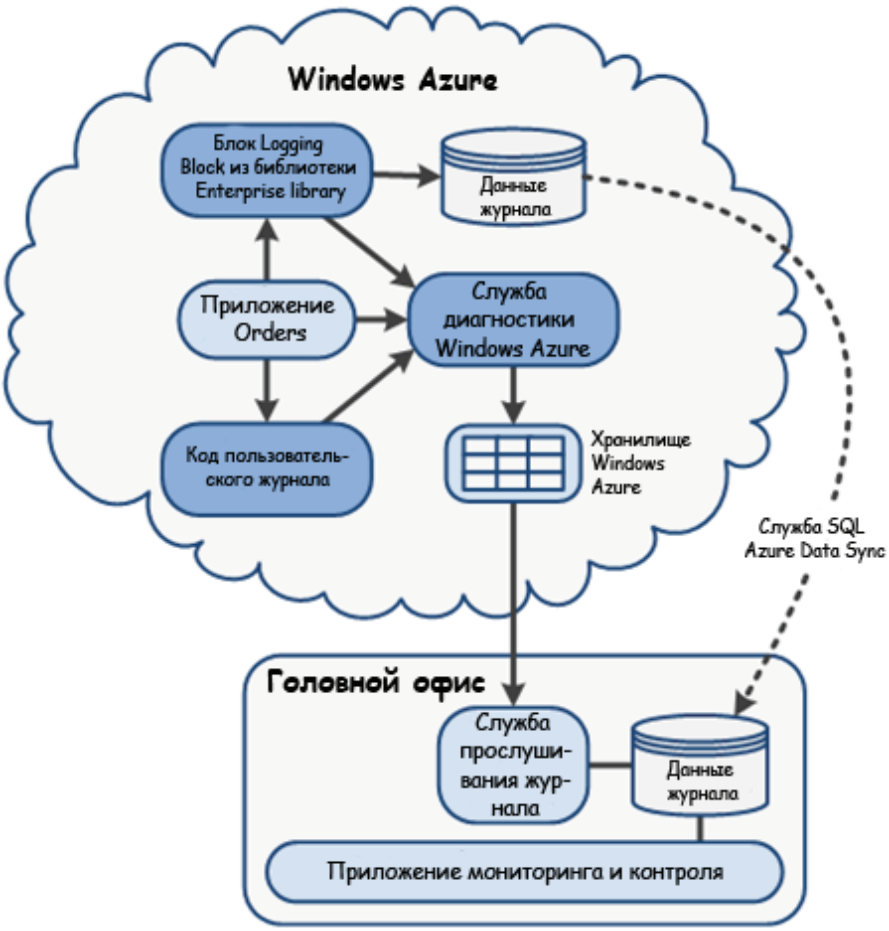



**Рисунок 1**

**Методы мониторинга, которые компания Trey Research выбрала для приложения Orders**

## Выбор решения для мониторинга и протоколирования

Компания Trey Research рассмотрела три способа сбора информации для мониторинга служб, протоколирования деятельности и измерения производительности в приложение Orders: службу диагностики Windows Azure, функциональный блок Logging Application Block из библиотеки Enterprise Library (стороннее решение для мониторинга) и использование пользовательского кода в приложении для создания сообщений журнала. В следующих разделах мы остановимся на каждом из них подробнее.

> **Мнение По**
>
> Существует целый ряд комплексных готовых к использованию решений мониторинга для работы с приложениями, развернутыми в Windows Azure. Эти продукты обычно предоставляют функции для сбора и анализа данных мониторинга, отображая их на панели мониторинга и присылая оператору уведомление о важных событиях. Такие решения включают Microsoft System Center Operations Manager и продукты сторонних разработчиков.

### Служба диагностики Windows Azure

Служба диагностики Windows Azure — это встроенный механизм, предназначенный для сбора всех видов информации мониторинга и диагностики в Windows Azure. Она не требует никакого дополнительного кода или сборок. Служба диагностики Windows Azure может собирать данные из журналов событий Windows и счетчиков производительности, журнала IIS и журнала невыполненных

запросов, журналов инфраструктуры, файлов аварийного дампа памяти и пользовательских журналов ошибок. Чтобы собрать необходимые данные и указать интервалы для их передачи в хранилище Windows Azure, разработчики и администраторы компании Trey Research просто настроили механизм диагностики. Они также могут использовать командлеты Windows Azure PowerShell, чтобы изменить параметры диагностики во время работы приложения, инициировать передачу данных в хранилище Windows Azure по требованию, а также загружать регистрируемые данные в локальное хранилище.

Однако существует только ограниченный набор параметров для фильтрации и классификации регистрируемой информации, и эта информация может храниться только в хранилище Windows Azure. Хранить данные в базе данных, специальном формате или репозитории невозможно.

**Примечание**

Для получения дополнительной информации об использовании службы диагностики Windows Azure см. «Приложение E. Мониторинг и управление гибридными приложениями» этого руководства.

### Функциональный блок Logging Application Block из библиотеки Enterprise Library

Logging Application Block — это компонент библиотеки Enterprise Library, платформы компонентов для решения задач совместимости в большинстве типов приложений. Компания Trey Research может настроить функциональный блок Logging Application Block для отправки записей журнала в прослушиватель трассировки службы диагностики Windows Azure Diagnostic. Прослушиватель является компонентом системы этой службы диагностики и хранит записи журнала в памяти, что позволяет передавать их в хранилище Windows Azure вместе с любыми другими данными диагностики, собранными системой.

Кроме того, компания Trey Research может настроить функциональный блок Logging Application Block для отправки записей журнала в другие типы хранилища, например в базы данных, текстовые файлы различных форматов, а также XML-файлы. Одним из вариантов, который рассматривался компанией, было использование функционального блока Logging Application Block для внесения записей в журнал непосредственно в базу данных технологической платформы SQL Azure, расположенную в облаке, и для последующей их передачи в локальную базу данных для анализа. Чтобы упростить задачу синхронизации данных между облаком и локальной базой данных, можно использовать компонент SQL Azure Data Sync.

**Мнение Маркуса**

Чтобы предоставить структурированный механизм на основании политики для сбора и обработки информации об исключениях, вы также можете использовать функциональный блок Exception Handling Application Block из библиотеки Enterprise Library. Компонент Exception Handling Block может посылать сообщения о записях в журнале в функциональный блок Logging Application Block из библиотеки Enterprise Library с помощью механизма службы диагностики Windows Azure Diagnostics.

Компонент Logging Application Block можно легко настраивать и расширять, он включает широкий спектр возможностей для фильтрации и классификации сообщений журнала. Это позволяет разработчикам компании Trey Research без затруднений создавать различные типы записей в журнале и обеспечивает полезную дополнительную поддержку администраторам и операторам.

Основным ограничением функционального блока Logging Application Block из библиотеки Enterprise Library является то, что он не может собирать данные из главной системы, например записи журнала событий Windows, счетчики производительности или файлы журнала IIS. Это исключительно механизм

протоколирования деятельности, в котором код создает записи в журнале в ответ на события, происходящие в приложении. Использование любого функционального блока из библиотеки Enterprise Library также означает, что сборки внешней библиотеки должны загружаться и устанавливаться с кодом приложения в Windows Azure.

**Примечание**

Дополнительная информация о функциональных блоках Logging Application Block и Exception Handling Application Block из библиотеки Enterprise Library представлена в статье «About This Release of Enterprise Library» на сайте http://msdn.microsoft.com/en-us/library/ff664636(v=PandP.50).aspx. Существует также официальный документ, который описывает возможности использования функциональных блоков Enterprise Library 5.0 с приложениями, размещенными в Windows Azure. См. http://entlib.codeplex.com/releases/view/75025#DownloadId=336804.

### Решение сторонних разработчиков для мониторинга

Компания Trey Research адаптировала решение сторонних разработчиков, предназначенное для мониторинга. Существует несколько решений, которые полностью или частично предназначены для мониторинга приложений и служб Windows Azure. В их числе:

- Windows Azure Management Pack для Microsoft System Center Operations Manager (http://pinpoint.microsoft.com/en-us/applications/system-center-monitoring-pack-for-windows-azure-applications-12884907699).
- Azure Diagnostics Manager компании Cerebrata (http://www.cerebrata.com/Products/AzureDiagnosticsManager/Default.aspx).
- AzureWatch от Paraleap Technologies (http://www.paraleap.com/).
- ManageAzis компании Cumulux (http://www.cumulux.com/products-and-services/cloud-operations/).

Эти решения могут отслеживать статус роли, собирать информацию о производительности и данные о событиях, а также передавать уведомления администраторам.

### Пользовательское решение протоколирования

Разработчики компании Trey Research решили создать свое решения для ведения журнала и диагностики для приложения Orders. Прослушиватель трассировки службы Windows Azure Diagnostic предоставляет методы для создания и хранения записей журнала, и соответственно предоставляет компании Trey Research возможность мониторинга деятельности и предоставления этих записей журнала с помощью стандартного механизма службы диагностики Windows Azure. Например, разработчики могут добавить код в приложение Orders, который создает сообщение каждый раз, когда посетитель впервые аутентифицируется и входит в приложение. Этот код может вызывать методы прослушивателя трассировки службы диагностики, чтобы сохранять сообщение в виде записи журнала. Впоследствии, при передаче данных для диагностики в хранилище Windows Azure, в эти данные будут включены записи, созданные пользовательским кодом.

Также есть возможность создавать пользовательские решения для ведения журнала, которые хранят данные в других форматах и местах. Например, также как функциональный блок Logging Application Block из библиотеки Enterprise Library, код может хранить записи журнала в базе данных, текстовом файле или репозитории в любом другом формате. Такой подход требует механизма удаленного доступа

к данным из локальных приложений и инструментов, а также механизма передачи данных обратно в локальное хранилище для последующего анализа.

> **Мнение Маркуса**
>
> Вы можете использовать прослушиватель трассировки службы Windows Azure Diagnostic для создания записей журнала, которые содержат информацию, необходимую для мониторинга событий и действий в вашем приложении. Эти данные можно увидеть с помощью механизма службы Windows Azure Diagnostics и, по мере необходимости, их можно передавать для анализа в хранилище Windows Azure.

Основным ограничением, которое компания Trey Research учитывала при использовании пользовательского решения для ведения журнала, является то, что также как и компонент Logging Application Block, оно не может собирать данные из главной системы, например записи журнала событий Windows, счетчики производительности или файлы журнала IIS. Это исключительно механизм протоколирования деятельности, в котором код создает записи в журнале в ответ на события, происходящие в приложении.

## Как компания Trey Research выбрала решение для мониторинга и протоколирования

Компания Trey Research хотела получить возможность генерировать определенные сведения о трассировке и мониторинге действий в приложении Orders во время его работы. Однако администраторы не хотели постоянно собирать полные сведения о трассировке или информацию о диагностике операционной системы. Вместо этого они предпочитают получить возможность изменять конфигурацию таким образом, чтобы в случае необходимости можно было собирать дополнительную информацию, например при отладке ошибки в приложении.

После тщательного анализа компания Trey Research решила использовать пользовательское решение для трассировки действий и записи конкретных ошибок, создавая записи в журнале, а затем записывая их в механизм службы Windows Azure Diagnostics. Хотя функциональный блок Logging Application Block из библиотеки Enterprise Library (и функциональный блок Exception Handling Application Block) был бы подходящим, но типы информации, которые компания собирает, ограничены, и поэтому дополнительная сложность использования этих блоков рассматривалась как недостаток в сценарии Trey Research.

## Как компания Trey Research использует службу Windows Azure Diagnostics

Компания Trey Research реализует протоколирование данных диагностики и загружает информацию из облака на свои локальные серверы. Она отслеживает работу каждого экземпляра роли, а также записывает информацию о любых возникающих исключениях, используя комбинацию пользовательского класса **TraceHelper** и стандартного механизма службы диагностики Windows Azure. Изначально данные хранятся в таблице с именем WADLogsTable, которая находится в хранилище таблицы в каждом центре данных. Компания рассматривала два описанных ниже варианта для мониторинга этих данных и их использования для управления системой:

- Использование решений System Center Operations Manager и Windows Azure Management Pack или другого решения стороннего разработчика для непосредственного подключения к каждому центру данных, анализ данных диагностики, создание соответствующих отчетов об использовании и отправка предупреждения оператору при ошибке экземпляра или других важных событиях.

- Периодическая передача данных диагностики для надежного локального хранения и последующее изменение формата этих данных для использования в собственных пользовательских инструментах для создания отчетов и анализа.

Несмотря на то, что решение System Center Operations Manager и другие решения сторонних разработчиков предоставляют множество мощных функций, существующие инвестиции, которые компания Trey Research уже сделала в разработку и приобретение пользовательских аналитических инструментов, показали, что второй вариант является более привлекательным. Кроме того, это означало, что компания могла легко сохранять полный журнал аудита всех существенных операций и событий локально, что могло отвечать более строгим требованиям соответствия нормативным положениям. Однако это решение не помешает компании Trey Research в будущем развернуть решение System Center Operations Manager или иное решение стороннего разработчика.

### Выбор данных и событий для записи

Компания Trey Research решила записывать различные типы событий, используя сообщения трассировки и службу Windows Azure Diagnostics. При нормальной работе, компания собирает только сообщения трассировки, которые имеют степень важности **Warning (Предупреждение)** или выше. Кроме того, механизм, который внедрила эта компания, позволяет администраторам изменять поведение для предоставления более подробной информации при отладке приложений или мониторинге специфической активности.

В следующей таблице приведена конфигурация системы ведения журнала, которая используется компанией Trey Research. Обратите внимание, что эта компания не собирает записи журнала событий Windows или данные счетчика производительности Windows. Вместо этого, компания фиксирует информацию на всех этапах работы приложения и отправляет эту информацию в механизм службы диагностики Windows Azure Diagnostics с помощью пользовательского класса **TraceHelper**.

| Тип события или трассировки | Механизм, который используется для ведения журнала | Тип события, которое запускает механизм записи журнала |
|---|---|---|
| События уровня **Error (Ошибка)**, определяемые приложением | Собираются пользовательским классом **TraceHelper** | Невозможно отправить сообщения в топик или очередь после всех попыток.<br>Сбой задачи обработки в рабочей роли.<br>Сбой при попытке доступа к кэшу Windows Azure. |
| События уровня **Warning (Предупреждения)**, определяемые приложением | Собираются пользовательским классом **TraceHelper** | Сбой отправки сообщения в топик или очередь, но будет осуществляться повторная попытка отправки.<br>Сбой во время обновления базы данных, но будет осуществляться повторная попытка. |
| События уровня **Information (Информации)**, определяемые приложением | Собираются пользовательским классом **TraceHelper** | Запуск приложения.<br>Запуск задачи обработки в рабочей роли.<br>Открытие представления на веб-сайте приложения Orders.<br>Дополнительная информация о взаимодействии веб-роли с кэшем Windows Azure.<br>Различные события, связанные с размещением заказа или добавлением данных заказчика. |

| | | |
|---|---|---|
| События уровня **Verbose (Подробные сведения)**, определяемые приложением | Собираются пользовательским классом **TraceHelper** | Не задано; доступно для будущих расширений. |
| Журнал событий Windows | Не собираются компанией Trey Research | Внутренние события операционной системы Windows или программного обеспечения. |
| Счетчики производительности Windows | Не собираются компанией Trey Research | Счетчики производительности реализуются с помощью операционной системы Windows и установленных приложений. |

**Мнение Бхарата**

Чтобы собирать журналы событий Windows и данные счетчиков производительности, необходимо настроить службу диагностики Windows Azure Diagnostics для передачи необходимых данных в хранилище таблиц Windows Azure. Записи журнала событий Windows передаются в таблицу **WADWindowsEventLogsTable**, а данные счетчиков производительности передаются в таблицу **WADPerformanceCountersTable**. Если компании Trey Research необходимо сохранять эти данные в локальном приложении управления, то ее разработчики должны написать дополнительный код для загрузки данных в эти таблицы.

### Настройка механизма диагностики

Рабочая роль в проекте **Orders.Workers** и веб-роль в проекте **Orders.Website** настроены для использования службы диагностики Windows Azure. Файл конфигурации для обоих приложений содержит следующие настройки:

**XML**

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  ...
  <system.diagnostics>
    <sources>
      <source name="TraceSource">
        <listeners>
          <add type="Microsoft.WindowsAzure.Diagnostics
                     .DiagnosticMonitorTraceListener, ..
."
               name="AzureDiagnostics">
            <filter type="" />
          </add>
        </listeners>
      </source>
    </sources>
  </system.diagnostics>
</configuration>
```

Эта конфигурация определяет диагностический источник прослушивателя **TraceSource**, который посылает сообщения трассировки в класс **DiagnosticsMonitorTraceListener** платформы Windows Azure.

В конфигурации не определяется уровень фильтра, так как он устанавливается в коде, который инициализирует прослушивателя **TraceSource**.

Чтобы настроить службу диагностики и график передачи диагностических данных в хранилище Windows Azure, компания Trey Research первоначально рассматривала возможность использования императивного подхода путем добавления приведенного ниже кода в методы **OnStart** классов, реализующие веб- и рабочие роли.

**C#**

```
...
// Получение начальной конфигурации по умолчанию.
var config =
    DiagnosticMonitor.GetDefaultInitialConfiguration();

// Обновление начальной конфигурации.
config.Logs.ScheduledTransferLogLevelFilter
       = LogLevel.Undefined;
config.Logs.ScheduledTransferPeriod
       = TimeSpan.FromSeconds(60);

// Запуск монитора с этой конфигурацией.
DiagnosticMonitor.Start("DiagnosticsConnectionString",
                        config);
...
```

Однако, несмотря на простоту, компания Trey Research решила, что такому подходу может не хватать гибкости. В случае необходимости, компания должна иметь возможность удаленно изменить конфигурацию службы диагностики и передать график (с помощью командлетов Windows Azure Powershell или одного из доступных решений сторонних разработчиков, предназначенных для мониторинга, например Cerebrata Diagnostics Manager, который упоминался выше). Но выполнение этих задач в коде приведет к потере всех удаленных изменений, которые были сделаны в конфигурации, после перезагрузки роли.

Поэтому компания Trey Research решила настроить конфигурационный файл диагностики diagnostics.wadcfg. Этот файл может храниться в хранилище BLOB-объектов (и остаться невредимым после перезагрузки роли), а во время запуска роли его можно прочитать с помощью монитора службы Windows Azure Diagnostics. Дополнительная информация представлена в статье «Using the Windows Azure Diagnostics Configuration File» на сайте http://msdn.microsoft.com/en-us/library/gg604918.aspx.

### Ведение журнала сообщений трассировки и определение уровня детализации

Trey Research собирает сообщения трассировки, сгенерированные пользовательским классом **TraceHelper** (находится в папке Helpers проекта **Orders.Shared**). Класс **TraceHelper** создает экземпляр **TraceSource** и предоставляет набор статических методов, которые позволяют легко записывать сообщения трассировки с разными степенями серьезности.

**C#**

```
public class TraceHelper
{
  private static readonly TraceSource Trace;

  static TraceHelper()
  {
    Trace = new TraceSource("TraceSource",
```

```
                            SourceLevels.Information);
  }

  [EnvironmentPermissionAttribute(
    SecurityAction.LinkDemand, Unrestricted = true)]
  public static void Configure(SourceLevels
sourceLevels)
  {
    Trace.Switch.Level = sourceLevels;
  }

  public static void TraceVerbose(string format,
                        params object[] args)
  {
    Trace.TraceEvent(TraceEventType.Verbose, 0,
                     format, args);
  }

  public static void TraceInformation(string format,
                        params object[] args)
  {
    Trace.TraceEvent(TraceEventType.Information, 0,
                     format, args);
  }

  public static void TraceWarning(string format,
                        params object[] args)
  {
    Trace.TraceEvent(TraceEventType.Warning, 0,
                     format, args);
  }

  public static void TraceError(string format,
                        params object[] args)
  {
    Trace.TraceEvent(TraceEventType.Error, 0,
                     format, args);
  }
}
```

Данные, записанные таким образом, направляются на прослушиватель трассировки монитора службы Windows Azure Diagnostics (согласно конфигурации, описанной в разделе «Настройка механизма диагностики» в этой главе), а затем в **WADLogsTable**. По умолчанию класс **TraceHelper** сохраняет сообщения с уровнем фильтра серьезности **Information**. Однако этот параметр можно изменить путем вызова метода **Configure**, который предоставляется классом **TraceHelper**, и указания значения степени серьезности сообщений для трассировки. И рабочие роли и веб-роли настраивают этот параметр в методе **OnStart** путем считывания его из файла конфигурации службы.

**C#**

```
public override bool OnStart()
{
  ...
  ConfigureTraceListener(
```

```
    RoleEnvironment.GetConfigurationSettingValue(
      "TraceEventTypeFilter"));
  ...
}

private static void ConfigureTraceListener(
                      string traceEventTypeFilter)
{
  SourceLevels sourceLevels;
  if (Enum.TryParse(traceEventTypeFilter, true,
                    out sourceLevels))
  {
    TraceHelper.Configure(sourceLevels);
  }
}
```

Роли также устанавливают обработчики событий **RoleEnvironmentChanging** и **RoleEnvironmentChanged**. При изменении конфигурации эти обработчики перенастраивают класс **TraceHelper** для роли. Это позволяет администраторам изменять степень фильтра серьезности, чтобы получить дополнительную информацию для отладки и мониторинга во время работы приложения.

**Мнение Маркуса**

Механизм, который реализует компания Trey Research для определения уровня трассировки, обеспечивает высокую степень контроля над объемом и характером данных, которые собираются. Однако альтернативным подходом является сбор данных для всех событий, а затем применение фильтров при передаче данных трассировки в хранилище Windows Azure путем настройки свойства **ScheduledTransferLogLevelFilter** конфигурации монитора диагностики. Это свойство определяется как часть конфигурации монитора службы диагностики Windows Azure, которая хранится в файле diagnostics.wadcfg и может обновляться удаленно без необходимости перезапуска ролей.

### Запись сообщений трассировки

Для записи информации о событиях, ошибках и других важных моментах веб- и рабочих ролей используют класс **TraceHelper**. Например, исключения фиксируются с помощью показанного в следующем примере кода, взятого из метода **ReceiveNextMessage** класса **ServiceBusReceiverHandler** в проекте **Orders.Shared**. Обратите внимание, что для записи сообщения трассировки со степенью серьезности Error (Ошибка) этот код вызывает метод **TraceError** класса **TraceHelper**.

**C#**

```
private void ReceiveNextMessage(
  CancellationToken cancellationToken)
{
  ...
  this.ReceiveNextMessage(cancellationToken);

  if (taskResult.Exception != null)
  {

TraceHelper.TraceError(taskResult.Exception.Message);
    throw taskResult.Exception;
  }
```

```
  ...
}
```

Класс **TraceHelper** также используется в веб-роли. Код в папке CustomAttributes проекта **Orders.Website** определяет пользовательский атрибут **LogActionAttribute**, который для записи сообщения трассировки со степенью серьезности Information (Информация) вызывает метод **TraceInformation** класса **TraceHelper**.

**C#**

```
public class LogActionAttribute : ActionFilterAttribute
{
  public override void OnActionExecuting(
    ActionExecutingContext filterContext)
  {
    ...
    TraceHelper.TraceInformation(
      "Executing Action '{0}', from controller '{1}'",
      filterContext.ActionDescriptor.ActionName,
      filterContext.ActionDescriptor.
        ControllerDescriptor.ControllerName);
  }

  public override void OnActionExecuted(
    ActionExecutedContext filterContext)
  {
    ...
    TraceHelper.TraceInformation(
      "Action '{0}', from controller '{1}'
        has been executed",
      filterContext.ActionDescriptor.ActionName,
      filterContext.ActionDescriptor.
        ControllerDescriptor.ControllerName);
      }
  }
}
```

Классы контроллера в проекте **Orders.Website** помечены этим атрибутом. Следующий фрагмент кода показывает класс **StoreController**, который извлекает продукцию для отображения.

**C#**

```
[LogAction]
public class StoreController : Controller
{
  ...
  public ActionResult Index()
  {
    var products = this.productStore.FindAll();
    return View(products);
  }

  public ActionResult Details(int id)
  {
    var p = this.productStore.FindOne(id);
    return View(p);
  }
```

```
}
```

Эта функция позволяет приложению создавать полный отчет обо всех отмеченных действиях, выполняемых каждым пользователем, просто изменив настройки **TraceEventTypeFilter** в файле ServiceConfiguration.cscfg на **Information**.

### Передача данных диагностики из облака

Для сбора и анализа информации диагностики компания Trey Research использует собственный механизм. Для этого требуется, чтобы все приложения сохраняли события и сообщения трассировки в локальной базе данных DiagnosticsLog, к которой через заданные интервалы посылается запрос от механизма мониторинга и анализа.

Для загрузки данных из WADLogsTable или записи сценариев, которые используют командлеты Windows Azure PowerShell, в решении Trey Research может использоваться инструмент сторонних разработчиков (см. http://wappowershell.codeplex.com). Кроме того, Windows Azure SDK предоставляет классы, которые облегчают процесс взаимодействия с хранилищем Windows Azure с помощью программного интерфейса управления с использованием платформы .NET Framework. Это подход, который выбрала компания Trey Research.

Приложение для локального мониторинга и управления (например, реализованное в проекте **HeadOffice**) содержит страницу, которую используют администраторы для загрузки и изучения диагностических данных, собранных в приложение **Orders**.

> В реальном приложении данные диагностики загружаются автоматически через заданные промежутки времени и сохраняются в локальной базе данных службой Windows или другим фоновым приложением. Для простоты в примере приложения данные загружаются только в том случае, когда вы открываете страницу **Diagnostics** приложения **HeadOffice**.

Код, который взаимодействует с хранилищем Windows Azure и обновляет локальную таблицу базы данных DiagnosticsLog, находится в классе **DiagnosticsController**, расположенном в папке Controllers проекта **HeadOffice**. Чтобы повторить попытку подключения к хранилищу Windows Azure и локальной базе данных, класс **DiagnosticsController** использует функциональный блок для обработки неустойчивых неисправностей из библиотеки Enterprise Library. Конструктор класса **DiagnosticsController** считывает политику повторных попыток из файла конфигурации приложения.

**C#**

```
this.storageRetryPolicy
  =
RetryPolicyFactory.GetDefaultAzureStorageRetryPolicy();
```

Когда администратор открывает страницу Diagnostics приложения **HeadOffice**, выполняется действие **TransferLogs**. Это действие извлекает список центров данных, из которых будут загружаться данные из конфигурации приложения, а затем считывает соответствующие данные учетной записи (из этой же конфигурации) для каждого центра данных. При переборе списка центров данных код создает подходящий экземпляр **CloudStorageAccount**, используя собранные ранее учетные данные, а затем вызывает метод **TransferLogs** для загрузки данных из этого центра данных.

**C#**

```
[HttpPost]
public ActionResult TransferLogs(
                    FormCollection formCollection)
{
  var deleteEntries
```

```
    = formCollection.GetValue("deleteEntries") != null;
  var dataCenters
    = WebConfigurationManager.AppSettings["dataCenters"]
                             .Split(',');
  ...
  // Получение сведений об учетной записи для доступа
к каждому центру данных.
  var dataCenters2 = dataCenters.Select(
       dc => dc.Trim()).Where(dc =>
             !string.IsNullOrEmpty(dc.Trim()));
  var accountNames = dataCenters2.Select(
    dc => string.Format(CultureInfo.InvariantCulture,
          "diagnosticsStorageAccountName.{0}", dc));
  var accountKeys = dataCenters2.Select(
    dc => string.Format(CultureInfo.InvariantCulture,
          "diagnosticsStorageAccountKey.{0}", dc));

  for (var i = 0; i < dataCenters2.Count(); i++)
  {
    // Создание учетных данных для этого центра данных.
    var cred = new StorageCredentialsAccountAndKey(
        WebConfigurationManager.AppSettings[
                        accountNames.ElementAt(i)],
        WebConfigurationManager.AppSettings[
                        accountKeys.ElementAt(i)]);
    var storageAccount = new CloudStorageAccount(cred,
                                                 true);

    // Загрузка данных из этого центра данных.
    this.TransferLogs(dataCenters2.ElementAt(i),
                      storageAccount, deleteEntries);
  }
  ...
}
```

Метод **TransferLogs** использует класс **CreateCloudTableClient** для доступа к хранилищу таблиц Windows Azure. Код получает доступ к контексту службы таблиц и создает запрос по **WADLogsTable** в хранилище Windows Azure. Для каждой записи, возвращенной по запросу (каждая строка в таблице), он создает новый экземпляр **DiagnosticsLog** и сохраняет его в базе данных DiagnosticsLog, используя класс репозитория **DiagnosticsLogStore**. Обратите внимание на то, что этот метод может также удалять записи в **WADLogsTable** в хранилище Windows Azure и в то же время снижать требования к хранению в облаке.

**C#**

```
private void TransferLogs(string dataCenter,
               CloudStorageAccount storageAccount,
               bool deleteWADLogsTableEntries)
{
  var tableStorage
      = storageAccount.CreateCloudTableClient();
  ...
  var context = tableStorage.GetDataServiceContext();

  if (!deleteWADLogsTableEntries)
```

```
  {
    context.MergeOption = MergeOption.NoTracking;
  }

  IQueryable<WadLog> query
    = this.storageRetryPolicy.ExecuteAction(() =>
           context.CreateQuery<WadLog>("WADLogsTable"));

  foreach (var logEntry in query)
  {
    var diagLog = new DiagnosticsLog
      {
        Id = Guid.NewGuid(),
        PartitionKey = logEntry.PartitionKey,
        RowKey = logEntry.RowKey,
        DeploymentId = logEntry.DeploymentId,
        DataCenter = dataCenter,
        Role = logEntry.Role,
        RoleInstance = logEntry.RoleInstance,
        Message = logEntry.Message,
        TimeStamp = logEntry.Timestamp
      };
    this.store.Save(diagLog);

    if (deleteWADLogsTableEntries)
    {
      context.DeleteObject(logEntry);
      this.storageRetryPolicy.ExecuteAction(() =>
                               context.SaveChanges());
    }
  }
}
```

**Мнение Бхарата**

При доступе и выполнении операций над таблицами Windows Azure, которые хранят диагностическую информацию, учитывайте стоимость транзакций, которые связаны с этими операциями. Возможно, дешевле оплатить хранение, чем оплатить большое количество транзакций, которые удаляют отдельные строки, а затем удаляют из памяти и создают таблицу повторно через соответствующие промежутки времени.

Стратегия, которая в решении Trey Research используется для удаления диагностических данных из хранилища Windows Azure после их загрузки, предотвращает увеличение объема этих данных до бесконечности, но за это приходится платить. Удаление каждой записи считается одной транзакцией применительно к хранилищу Windows Azure, и в решении Trey Research тарифицируется это соответствующим образом. По мере увеличения числа клиентов, увеличивается объем диагностических данных, который хранится в решении Trey Research, и в конечном итоге плата за транзакции, связанные с удалением каждой записи по отдельности после ее загрузки, может стать непомерно высокой.

Чтобы подсчитать эти дополнительные расходы, специалисты компании Trey Research в настоящее время оценивают альтернативный подход: вместо удаления отдельных записей из таблиц в хранилище Windows Azure решено просто удалять таблицы из памяти и создавать их заново на соответствующем этапе после загрузки данных. Этот подход включает в себя гораздо меньше транзакций, но усложняет

код роли, которая загружает данные. Может потребоваться разработка механизма блокировки для предотвращения запланированной передачи диагностических данных в таблицу, которая была только что удалена, но еще не создана заново. Кроме того, на удаление и создание таблиц может уйти больше времени, чем на удаление отдельных записей из существующей таблицы, поэтому эта функциональность, возможно, должна быть реализована в веб- или рабочей роли как фоновая задача.

# Развертывание и управление

Компания Trey Research хотела получить возможность настраивать и управлять всеми службами в пределах учетной записи платформы Windows Azure, которые используются приложением Orders. Компании необходимо, чтобы конфигурация компонентов, таких как ACS и шина интеграции, была автоматизированной и точно повторяемой. Такая конфигурация является достаточно сложной и включает в себя конфигурацию служб в нескольких центрах данных, в которых развернуто приложение.

Компания Trey Research также хотела автоматизировать развертывание и повторное развертывание приложения по мере его обновления и расширения. Автоматизация развертывания снижает вероятность ошибок и позволяет контролировать права доступа для сотрудников, которые могут выполнять эти задачи.

## Выбор решений для развертывания и управления

Компания Trey Research рассмотрела ряд решений для развертывания и управления приложением Orders. Эти варианты включали использование портала управления Windows Azure, Windows Azure SDK, программного интерфейса REST для управления службами Windows Azure, а также командлеты Windows Azure PowerShell. В следующих разделах мы остановимся на каждом из них подробнее.

### Портал управления Windows Azure

Портал управления является основным местом расположения, в котором создаются пространства имен служб, а также он может использоваться для настройки всех служб Windows Azure для подписки. Портал обеспечивает обратную связь, касающуюся состояния каждой службы, а также предоставляет графический интерфейс, интуитивно понятный и простой в использовании. Несмотря на это, все пользователи портала управления должны получить к нему доступ, предоставляя административные учетные данные для подписки Windows Azure компании Trey Research, что означает, что они будут иметь доступ ко всем функциям подписки.

Администраторы компании Trey Research знают, что использование портала управления является единственным способом создания новых пространств имен для таких служб, как шина интеграции, ACS и диспетчер трафика, хотя эти пространства имен могут быть настроены позже посредством использования портала, сценариев или кода.

### Windows Azure SDK и программный интерфейс REST управления службами Windows Azure

За исключением создания пространства имен для служб, доступ ко всем функциям служб Windows Azure можно получить с помощью Windows Azure SDK и программного интерфейса REST управления службами Windows Azure. В отличие от программного интерфейса REST управления службами, Windows Azure SDK содержит сборки, используемые для выполнения задач управления службами. Кроме того, можно использовать инструменты сторонних разработчиков или создать свой собственный код, который обращается к интерфейсам REST программного интерфейса управления службами, чтобы автоматизировать задачи управления. Этот подход полезен, если вы создаете решение, основанное

на языке, который не поддерживается в Windows Azure SDK, например вы можете установить Windows Azure SDK для Java и использовать язык программирования Java.

Администраторы и разработчики компании Trey Research поняли, что для выполнения сложных задач они могли бы использовать код внутри своих приложений и инструментов управления путем использования программного интерфейса управления службами, включая создание программ установки и инструментов для управления большинством аспектов приложения и служб, которые он использует.

Главное ограничение интерфейса REST управления службами Windows Azure состоит в том, что его невозможно использовать напрямую в сценариях.

> **Примечание**
>
> Дополнительная информация о программном интерфейсе REST управления службами Windows Azure представлена в статье «About the Service Management API» по адресу http://msdn.microsoft.com/en-us/library/windowsazure/ee460807.aspx.

### Командлеты Windows Azure PowerShell

Библиотека командлетов Windows Azure PowerShell, которую можно загрузить с веб-сайта Codeplex, содержит почти сто командлетов PowerShell, которые могут выполнять наиболее распространенные задачи управления и настройки Windows Azure.

Эти командлеты являются чрезвычайно полезными для выполнения широкого спектра задач управления, и они могут не только использоваться в других сценариях, но и выполняться напрямую из командной строки. Администраторы компании Trey Research поняли, что командлеты обеспечивают идеальное решение для выполнения простых повседневных задач.

Однако, хотя командлеты Windows Azure PowerShell можно использовать в сценариях для сложных задач управления, это может быть труднее, чем использование программного интерфейса REST управления службами Windows Azure.

> **Примечание**
>
> Дополнительная информация о командлетах Windows Azure PowerShell представлена в статье «Windows Azure PowerShell Cmdlets» на сайте http://wappowershell.codeplex.com.

## Как компания Trey Research выбирает решения для развертывания и управления

Управление приложениями и службами Windows Azure обычно включает в себя два типа задач:

- Нерегулярные или редкие задачи, а именно настройка пространств имен и компонентов, таких как ACS, шина интеграции Windows Azure и диспетчер трафика Windows Azure. Часто эти задачи бывают сложными.
- Частые задачи, такие как развертывание и обновление служб, загрузка данных журнала, добавление и удаление сертификатов, управление службами хранилища, настройка правил брандмауэра и взаимодействие с SQL Azure, как правило, бывают более простыми.

Для нерегулярных задач компания Trey Research решила создать приложения и инструменты, которые используют программный интерфейс REST управления службами Windows Azure через объекты, предоставляемые Windows Azure SDK и отдельной библиотекой. Например, компания решила создать

программу установки, которая может выполняться для установки многих параметров настройки в Windows Azure вместо использования портала управления.

Для наиболее часто выполняемых задач администраторы Trey Research решили использовать в сценариях командлеты Windows Azure PowerShell, чтобы обеспечить повторяемый, надежный и автоматизированный процесс. Например, компания использует сценарий PowerShell, который выполняется до развертывания, с целью установки соответствующих значений для пространства имен, имен пользователей, паролей и ключей в исходных файлах. Эти элементы отличаются для каждого центра данных, в которых развернуто приложение.

Администраторы компании Trey Research также используют сценарии PowerShell для выполнения таких задач, как изменение конфигурации службы Windows Azure Diagnostics для трассировки и отладки, управления сертификатами, запуска и остановки экземпляров ролей приложения и управления базами данных SQL Azure.

## Как компания Trey Research разворачивает приложение Orders и управляет им

В следующих разделах описывается, как компания Trey Research использует программный интерфейс REST управления службами Windows Azure через библиотеку интерфейсов управления и напрямую, чтобы по большей части автоматизировать конфигурацию служб Windows Azure, таких как ACS и шина интеграции.

### Настройка Windows Azure с помощью библиотеки интерфейсов управления службами

Компания Trey Research использует библиотеку функций, которая первоначально была разработана командой Windows Azure для автоматизации конфигурации пространств имен Windows Azure через программный интерфейс управления на основании интерфейса REST. Код библиотеки включен в проект **ACS.ServiceManagementWrapper** примера кода, и вы можете использовать эту библиотеку в своих приложениях.

> **Мнение Маркуса**
>
> Внутренний механизм библиотеки интерфейсов управления службами здесь не описан, но вы можете изучить исходный код и изменить его при необходимости. Это сложный проект с большим количеством функций, который делает выполнение большого количества задач по настройке Windows Azure проще, чем написание собственного пользовательского кода.

Программа установки, разработанная компанией Trey Research, создает экземпляр объекта **ServiceManagementWrapper**, а затем вызывает несколько отдельных методов в рамках программы установки для настройки ACS и шины интеграции для приложения Orders. Конфигурация шины интеграции, которую использует компания, зависит от ACS для аутентификации учетных записей, которые посылают сообщения в или подписываются на очереди и топики.

**C#**

```
internal static void Main(string[] args)
{
  try
  {
    var acs = new ServiceManagementWrapper(
                  acsServiceNamespace,
                  acsUsername, acsPassword);
```

```
    Console.WriteLine("Setting up ACS namespace:"
                      + acsServiceNamespace);

    // Установка пространства имен ACS для веб-сайта
Orders
    CleanupIdenityProviders(acs);
    CleanupRelyingParties(acs);
    CreateIdentityProviders(acs);
    CreateRelyingPartysWithRules(acs);

    // Создание топика, подписок и очереди шины
интеграции.
    SetupServiceBusTopicAndQueue();
  }
  catch (Exception ex)
  {
    ... display exception information ...
  }
    Console.ReadKey();
}
```

Значения, используемые программой установки для имен пространства имен, идентификаторов учетной записи, паролей и ключей, хранятся в файле App.config проекта программы установки **TreyResearch.Setup**. Обратите внимание, что код сначала очищает текущее пространство имен ACS, удаляя все существующие параметры, а новые параметры заменяют старые. В противном случае ACS может попытаться добавить копии параметров или функций, таких как поставщики удостоверений или наборы правил, которые могут быть причиной ошибки.

Чтобы показать, как легко использовать библиотеку интерфейсов управления службами, следующий код из метода **CleanupRelyingParties** программы установки удаляет всех существующих доверенных участников с именем AccessControlManagement из текущего пространства имен ACS.

**C#**

```
var rps = acsWrapper.RetrieveRelyingParties();
foreach (var rp in rps)
{
  if (rp.Name != "AccessControlManagement")
  {
    acsWrapper.RemoveRelyingParty(rp.Name);
  }
}
```

Программа установки создает удостоверения служб, сначала удаляя все удостоверения с таким же именем, а затем добавляя новое удостоверение с указанным именем и паролем. Значения, используемые в этом коде, поступают из файла App.config.

**C#**

```
acswrapper.RemoveServiceIdentity(manInVanDisplayName);
acswrapper.AddServiceIdentity(manInVanDisplayName,
                              manInVanPassword);
```

Программа установки использует библиотеку интерфейсов службы управления для создания правил, которые сопоставляют утверждения из поставщика удостоверений с утверждениями, необходимыми для приложения **Orders**. Например, следующий код создает сквозное правило для поставщика

удостоверений Windows Live ID, которое сопоставляет утверждение **NameIdentifier**, предоставленное Windows Live ID с новым утверждением **Name**.

**C#**

```
var identityProviderName
    = SocialIdentityProviders.WindowsLiveId.DisplayName;

// передача nameidentifier в качестве имени
acsWrapper.AddPassThroughRuleToRuleGroup(
           defaultRuleGroup.RuleGroup.Name,
           identityProviderName,
           ClaimTypes.NameIdentifier, ClaimTypes.Name);
```

### Настройка Windows Azure с помощью встроенных объектов управления

Для настройки пространства имен Windows Azure также можно использовать встроенные объекты, которые являются частью Windows Azure SDK. Например, программа установки настраивает правила доступа для конечных точек шины интеграции в ACS. Для этого она использует классы в пространстве имен **Microsoft.ServiceBus.AccessControlExtensions.AccessControlManagement**.

В следующем фрагменте кода показано, как программа установки создает группу правил для службы статистики приложения Orders. Вы видите, что она устанавливает ACS в качестве отправителя утверждения, а также добавляет два правила в группу правил. Первое правило разрешает аутентифицированному удостоверению **externaldataanalyzer** (небольшая и простая демонстрационная программа, которая может отображать статистику заказов) отправлять запросы. Второе правило разрешает аутентифицированному удостоверению **headoffice** (локальное приложение для управления и мониторинга) ожидать запросы. Затем код добавляет группу правил доверенному участнику **OrdersStatisticsService** и сохраняет все изменения.

**C#**

```
var settings = new AccessControlSettings(
                ServiceBusNamespace, DefaultKey);
ManagementService serviceClient =
ManagementServiceHelper
               .CreateManagementServiceClient(settings);

serviceClient.DeleteRuleGroupByNameIfExists(
             "Rule group for OrdersStatisticsService");
serviceClient.SaveChanges(SaveChangesOptions.Batch);

var ruleGroup = new RuleGroup {
      Name = "Rule group for OrdersStatisticsService" };
serviceClient.AddToRuleGroups(ruleGroup);

// Эквивалентно выбору службы Access Control Service
в качестве
// входного отправителя утверждения в портале управления.
var issuer = serviceClient.GetIssuerByName(
                                  "LOCAL AUTHORITY");

serviceClient.CreateRule(
    issuer,
    "http://schemas.xmlsoap.org/ws/2005/05/identity/
       claims/nameidentifier",
```

```
    "externaldataanalyzer",
    "net.windows.servicebus.action",
    "Send",
    ruleGroup,
    string.Empty);

serviceClient.CreateRule(
    issuer,
    "http://schemas.xmlsoap.org/ws/2005/05/identity/
       claims/nameidentifier",
    "headoffice",
    "net.windows.servicebus.action",
    "Listen",
    ruleGroup,
    string.Empty);

var relyingParty = serviceClient.GetRelyingPartyByName(
                        "OrdersStatisticsService", true);
var relyingPartyRuleGroup = new
  Microsoft.ServiceBus.AccessControlExtensions
  .AccessControlManagement.RelyingPartyRuleGroup();
relyingParty.RelyingPartyRuleGroups.Add(
                                relyingPartyRuleGroup);
serviceClient.AddToRelyingPartyRuleGroups(
                                relyingPartyRuleGroup);

serviceClient.AddLink(relyingParty,
      "RelyingPartyRuleGroups", relyingPartyRuleGroup);
serviceClient.AddLink(ruleGroup,
      "RelyingPartyRuleGroups", relyingPartyRuleGroup);

serviceClient.SaveChanges(SaveChangesOptions.Batch);
```

**Мнение Маркуса**

В проекте **TreyResearch.Setup** имеется большое количество кода. Когда вы будете создавать свои собственные программы установки, вам может пригодиться этот проект, так как возможно будет воспользоваться некоторыми стандартными правилами, которые содержатся в нем.

## Выводы

В этой главе описывается, как компания Trey Research решала проблемы, связанные с развертыванием, настройкой, мониторингом и управлением гибридных приложений. Важно понимать, что сложность среды и ее распределенный характер означает, что невозможно избежать проблем с производительностью и сбоями в такой системе. Ключевым моментом поддержания хорошего уровня служб является обнаружение этих проблем и сбоев и быстрое реагирование управляемым, безопасным и повторяемым способом.

Служба диагностики Windows Azure предоставляет базовые инструменты, которые позволяют обнаруживать и определять возможные причины ошибок и проблем с производительностью, но важно, чтобы вы понимали, как связана получаемая диагностическая информация со структурой вашего

приложения. Анализ этой информации — это задача эксперта, который не только понимает архитектуру и бизнес-операции вашего приложения, но также хорошо знаком с тем, как эта архитектура соответствует службам, предоставляемым Windows Azure.

Программный интерфейс управления службами Windows Azure обеспечивает контролируемый доступ к компонентам Windows Azure, позволяя создавать сценарии и приложения, которые оператор может использовать для развертывания и управления элементами вашего приложения. Такой подход исключает необходимость оператору иметь такой же уровень знаний о Windows Azure, как и архитектору решений, а также может сократить объем ошибок за счет автоматизации и упорядочивания многих задач, связанных со сложным развертыванием.

## Дополнительная информация

- «Collecting Logging Data by Using Windows Azure Diagnostics» http://msdn.microsoft.com/en-us/library/windowsazure/gg433048.aspx
- «Monitoring Windows Azure Applications» http://msdn.microsoft.com/en-us/library/windowsazure/gg676009.aspx
- «Windows Azure Service Management REST API Reference» http://msdn.microsoft.com/en-us/library/windowsazure/ee460799.aspx
- «Take Control of Logging and Tracing in Windows Azure» http://msdn.microsoft.com/en-us/magazine/ff714589.aspx
- «Windows Azure PowerShell Cmdlets» http://wappowershell.codeplex.com

# Приложение A. Репликация, распространение и синхронизация данных

Все приложения, кроме самых простых, должны обеспечивать хранение и извлечение данных. В рамках многих систем в качестве надежного хранилища выступает база данных. Современные системы управления базами данных (СУБД), такие как Microsoft SQL Server, обеспечивают многопользовательский доступ и способны обрабатывать тысячи одновременных соединений при наличии необходимого оборудования и достаточной пропускной способности сети. А когда требуется поддержка высокомасштабируемого хранилища данных, которое снижает необходимость в установке и обслуживании дорогостоящего оборудования в центрах обработки данных организации, такие решения, как технологическая платформа SQL Azure, предоставляют облачную СУБД, которая реализует многие аналогичные функции.

Используя SQL Azure, можно развернуть базу данных в том же центре обработки данных, в котором размещены использующие ее облачные приложения и службы, что сводит к минимуму задержки, которые характерны для удаленного сетевого доступа к базе данных. Тем не менее в рамках гибридной системы, которая включает приложения, работающие в самых разных удаленных друг от друга точках, одного экземпляра SQL Azure может оказаться недостаточно, чтобы обеспечить хорошее время отклика. Вместо этого организация, возможно, примет решение о размещении копии базы данных на каждом узле. В таком сценарии необходимо убедиться, что все экземпляры БД содержат одинаковые данные. Учитывая изменчивость данных, это достаточно сложная задача.

Кроме того, можно хранить данные в другом хранилище, например использовать другую СУБД или другую структуру источника данных. В подобной ситуации вам придется реализовать собственную стратегию с целью синхронизации соответствующих данных.

В этом приложении рассматриваются вопросы, связанные с распределением и репликацией данных между службами, работающими в облаке и охватывающими облачные и локальные среды,— с использованием возможностей, предоставляемых технологической платформой Windows Azure. Здесь представлены возможные решения, которые помогут организации обеспечить синхронизацию источников данных.

## Сценарии использования и вызовы

Основная цель репликации данных в рамках гибридного облачного решения — сократить задержки в процессе сетевого доступа к данным. Данные хранятся в непосредственной близости от приложений и служб, которые их используют, тем самым повышается время отклика этих приложений и служб. Как описано выше, если работающая в центре обработки данных служба использует данные, хранящиеся в локальной БД в том же центре обработки данных, это позволяет избежать задержек и проблем с точки зрения надежности, характерных для процесса обмена сообщениями через Интернет. Однако такие

преимущества не всегда достаются бесплатно, поскольку приходится копировать данные, хранящиеся в локальной БД, в другие центры обработки данных, а также обеспечивать синхронизацию любых изменений между всеми задействованными центрами обработки данных.

Необходимо также учитывать, что репликация данных может вызвать несоответствия. Все вносимые изменения приходится применять ко всем копиям данных, и на это может потребоваться некоторое время. Полностью транзакционные системы используют процедуры, которые блокируют все копии элемента данных в процессе внесения изменений.Данные будут доступны только после того, как обновление было успешно применено ко всем экземплярам. Однако в рамках глобально распределенной системы такой подход нецелесообразен из-за неизбежных задержек в процессе работы через Интернет, поэтому большинство систем, которые осуществляют репликацию, обновляют каждый узел в отдельности. После обновления на различных узлах могут быть разные данные, но система обеспечит согласованность после того, как процесс синхронизации проведет обновления на всех узлах.

Следовательно, репликация лучше подходит для ситуаций, когда данные изменяются относительно редко, или логика приложения может справиться с неактуальной информацией до тех пор, пока она в конечном счете не обновится, возможно, через несколько минут или даже часов. Например, в приложении, которое позволяет клиентам размещать заказы на приобретение продукции, можно отображать текущий уровень запасов по каждому продукту. При этом реальное количество может не совпадать с отображаемым, если продукт популярный.Другие пользователи в это же время размещают заказы на его приобретение. В таком случае, когда пользователь подтвердит заказ, уровень запасов следует проверить еще раз, чтобы при необходимости можно было предупредить клиента о том, что могут возникнуть задержки в процессе доставки, если запас товара исчерпан.

В зависимости от стратегии, которую вы избрали, в сфере репликации и управления согласованностью у вас, скорее всего, могут возникнуть дополнительные сложности, связанные с разработкой, реализацией и управлением своей системой. Когда вы анализируете задачу организации репликации данных, необходимо обратить внимание на две основные проблемы:

- Какую топологию репликации следует использовать?
- Какой стратегии синхронизации следует придерживаться?

Выбор топологии репликации зависит от того, каким образом и где осуществляется доступ к данным, а стратегия синхронизации определяется необходимостью поддержания всех копий данных в актуальном состоянии. В следующих разделах описаны некоторые общие сценарии репликации, распространения и синхронизации данных, а также основные проблемы, которые возникают в рамках каждого из этих сценариев.

## Репликация данных из различных источников в рамках облачных и локальных сред

**Описание: данные должны быть расположены как можно ближе к логике приложения, которое их использует, независимо от того, где эта логика размещена — в облачном или локальном центре обработки данных.**

Репликация — это процесс копирования данных, который порождает определенные проблемы, связанные с необходимостью создания и управления несколькими копиями одних и тех же данных. Выбрав соответствующую топологию репликации, вы с большой вероятностью сможете решить эти проблемы.

В простейшей реализации такого подхода все данные из конкретного источника копируются во все другие экземпляры того же источника, вне зависимости от того, являются ли они облачными или локальными. В рамках такого сценария локальные приложения и облачные сервисы могут запрашивать и изменять любые данные. Они подключаются к самому близкорасположенному экземпляру источника данных, выполняют запросы и все необходимые обновления. В определенный момент эти обновления должны быть переданы во все остальные экземпляры источника данных и применены в согласованном порядке. Эта топология показана на рисунке 1 (в тексте данного приложения — топология А).

**Примечание**

На схемах в этом разделе жирные стрелки обозначают пути синхронизации между базами данных.

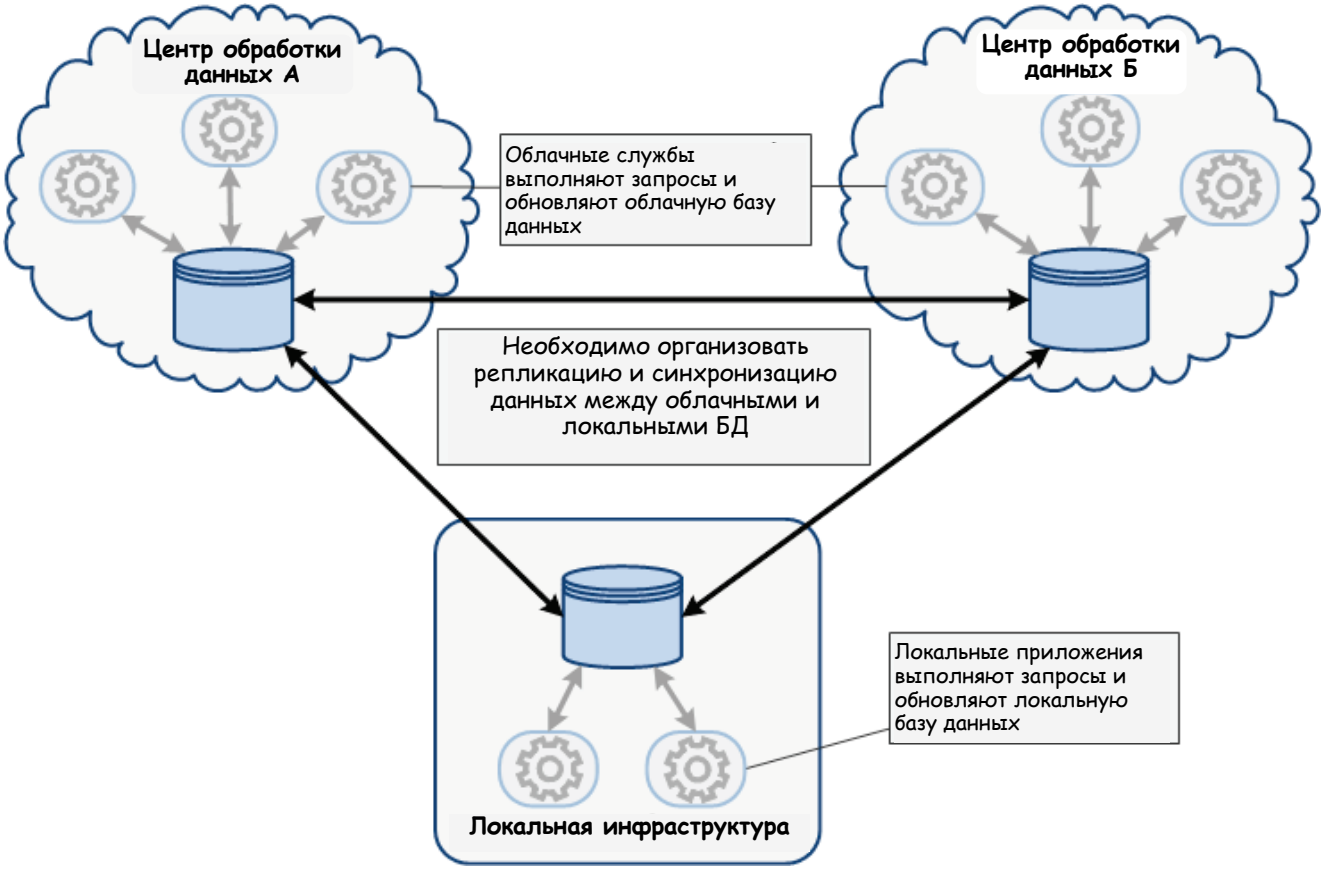



**Рисунок 1**

### Топология А: двунаправленная синхронизация между всеми базами данных — локально и в облаке

Если в вашем случае основной причиной для переноса служб данных в облако были исключительно вопросы масштабируемости и доступности, вы можете решить, что достаточно перенести источники данных с локальных серверов в облако и создать их дубликаты во всех центрах обработки данных. Такой подход оправдан, если основная часть логики приложения, которое обращается к данным, уже мигрировала в облако. При этом проблемы, связанные с обновлением данных и согласованным распространением этих изменений, как описано в топологии А, по-прежнему актуальны, с той лишь разницей, что нет локального источника данных. На рисунке 2 представлен этот сценарий.

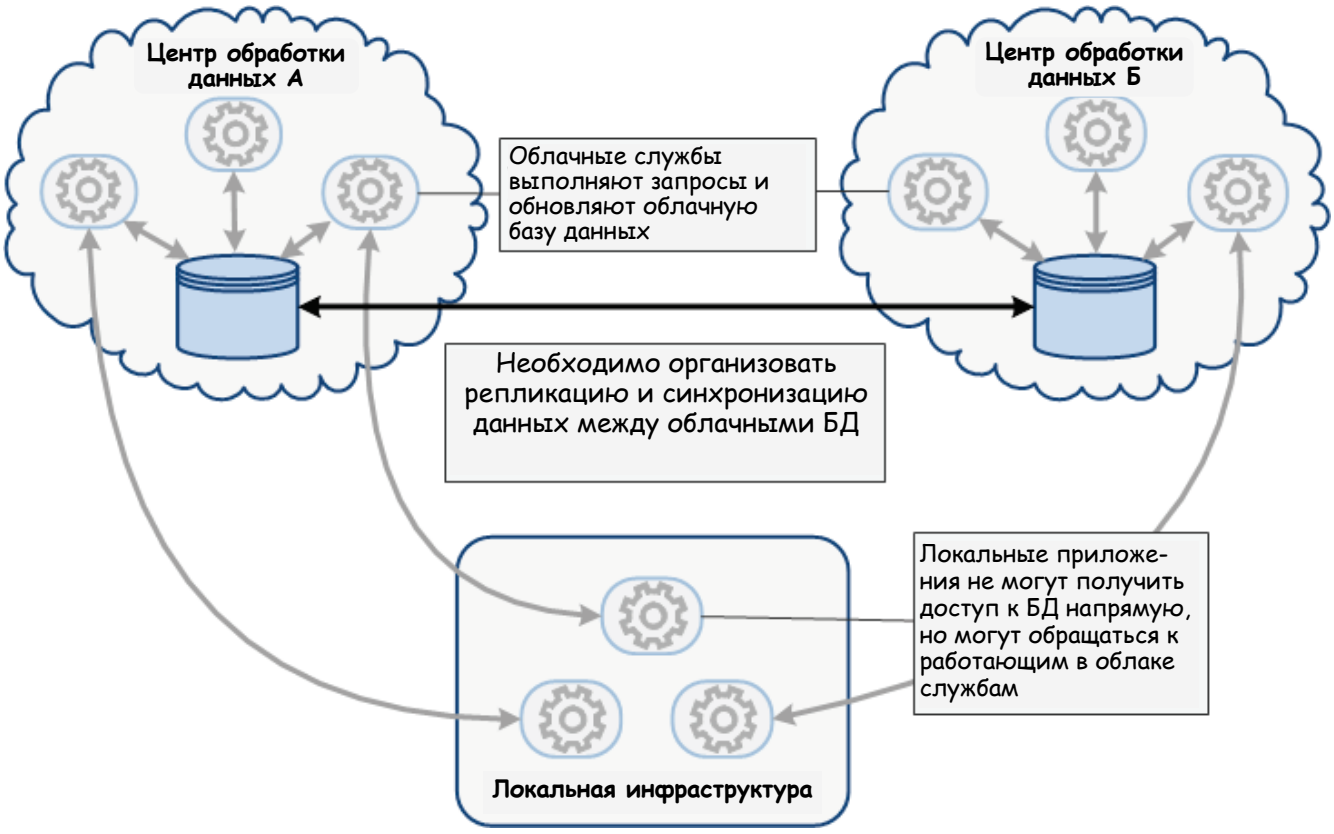



**Рисунок 2**

### Топология Б: двунаправленная синхронизация только между базами данных в облаке

Несмотря на то, что топология А и топология Б просты для понимания, такие общие стратегии не всегда целесообразны, особенно если данные секционированы в зависимости от местоположения служб, которые чаще всего их используют. Рассмотрим пример системы управления запасами в организации, которая управляет несколькими географически распределенными складами. Служба, работающая в центре обработки данных в том же географическом регионе, в котором расположен склад, может отвечать только за управление данными этого склада. В таком случае, можно организовать репликацию только данных, относящихся к этому складу, в центре обработки данных, в котором размещен соответствующий экземпляр службы, сохраняя при этом копию всей базы данных локально.

Когда служба изменяет данные в своей облачной БД, она может применить соответствующие изменения в локальной копии всей БД. Если службе потребуется доступ к данным, хранящимся в другом месте, она может запросить эту информацию в локальной базе данных. Локальная база данных эффективна в качестве основного репозитория для всей системы, а БД, работающие в каждом центре обработки данных, выступают в качестве облачного кэша с данными, относящимися только к соответствующему центру обработки данных.

> **Примечание**
>
> «Приложение Д. Максимизация масштабируемости, доступности и производительности» содержит описание других способов создания кэша при помощи инструмента Windows Azure Caching.

Такой подход сводит к минимуму необходимость копирования потенциально больших объемов данных, которые редко используются центрами обработки данных.Для этого потребуется дополнительная логика в коде службы, чтобы определить расположение данных. Кроме того, если служба регулярно

формирует запросы на доступ к необлачным данным или выполняет большое количество обновлений, то преимущества этой стратегии, по сравнению с простым доступом к локальному источнику данных в процессе обработки каждого запроса, не столь значительны. Такой подход также предполагает, что каждый элемент в источнике данных управляется только одним центром обработки данных, в противном случае существует риск потери обновленной информации (службы, работающие в двух центрах обработки данных, могут попытаться обновить одни и те же элементы).

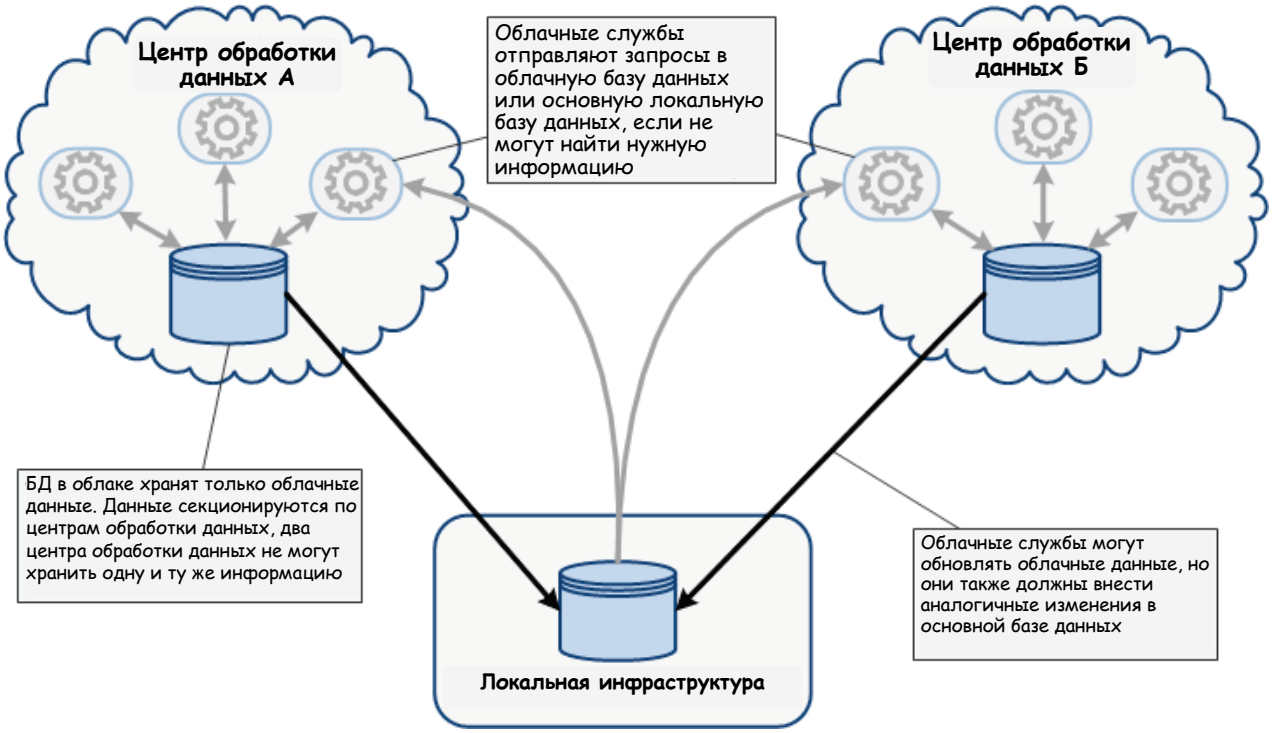



**Рисунок 3**

### Топология B: основной локальный репозиторий с односторонней синхронизацией по направлению от облака

В одной из разновидностей этого сценария облачные службы в первую очередь запрашивают данные и отправляют все обновления локальному приложению, и если облачная служба должна изменить данные, она посылает запрос локальному приложению, которое предназначено для получения и обработки таких запросов. Локальное приложение может изменить информацию в основном источнике данных, также размещенном локально, а затем инициировать аналогичные изменения для копирования в соответствующих базах данных в облаке. В рамках этого подхода для служб в облаке используется относительно простая логика (по сравнению с топологией B), поскольку обновлениями управляет локальное приложение. Данная топология (см. рисунок 4) также гарантирует, что все источники данных в облаке в конечном счете будут согласованы посредством репликации информации из основного источника.

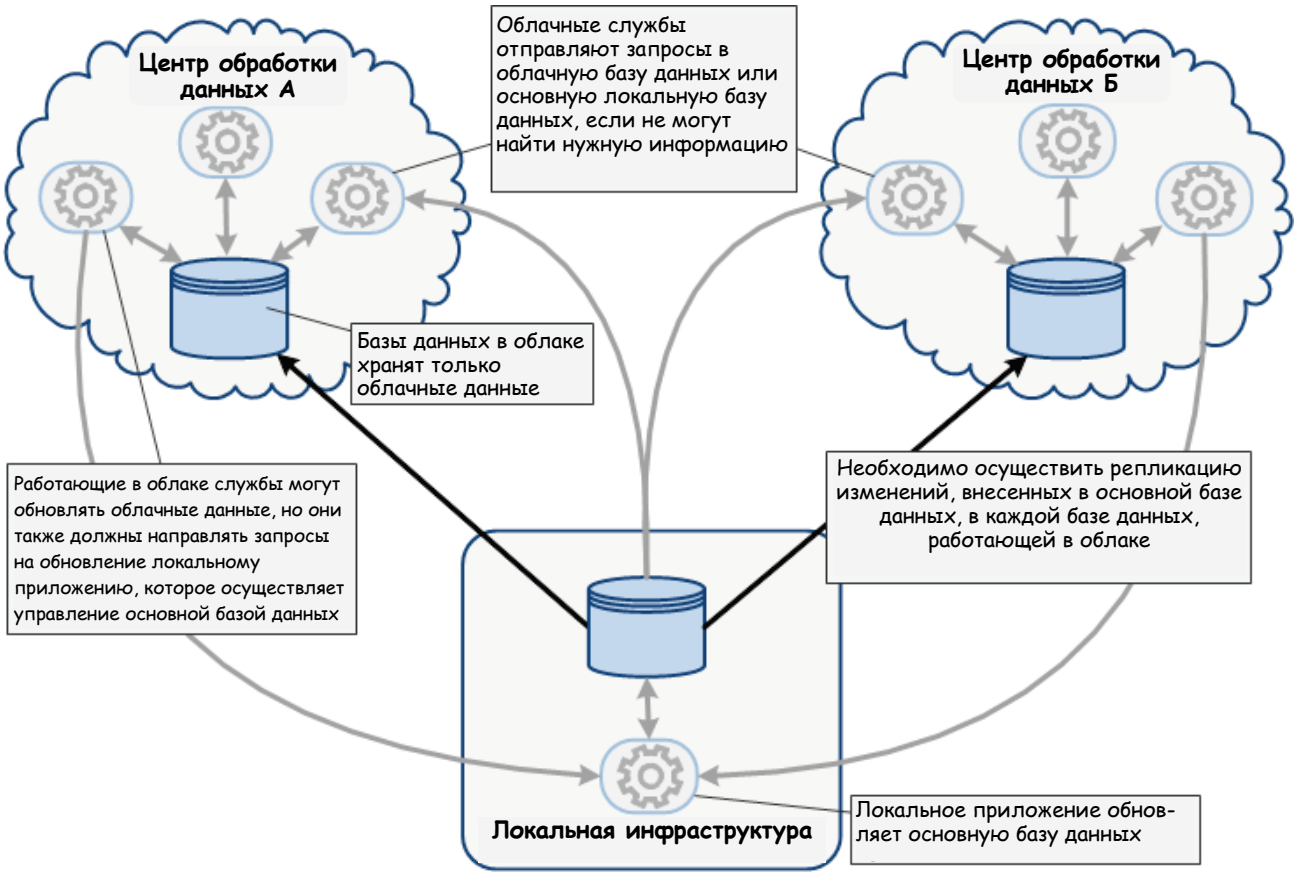



**Рисунок 4**

### Топология Г: основной локальный репозиторий с односторонней синхронизацией по направлению к облаку

В простой реализации топологии Г работающее локально приложение обновляет основную базу данных по своему алгоритму, а не по запросу облачных служб. Приложение может выполнять определенные операции, работая с данными под непосредственным контролем пользователя на стороне хостинг-провайдера. В рамках данного сценария обновленные данные просто реплицируются в каждую базу данных в облаке.

Окончательный вариант использования относится к организации, в которой несколько узлов, размещаемых не в облаке. Организация оставляет данные и приложения, которые их используют, в локальной среде, но выполняет репликацию данных между узлами через облако. В данном случае облако выступает просто в качестве канала передачи данных между узлами. Такой сценарий подходит для удаленных филиалов, которые могут затребовать полную копию данных или только подмножество данных, относящихся к этому филиалу. В любом случае, приложения, которые работают в каждом филиале, получают доступ к локальному источнику данных, размещенному на том же узле, а все обновления проходят через облако. Копия источника данных в облаке, через которое проходят все обновления, может использоваться в качестве узла репликации, собирая и распространяя обновленную информацию, а также выступать в качестве основного репозитория, если филиалу требуется доступ к нелокальным данным.

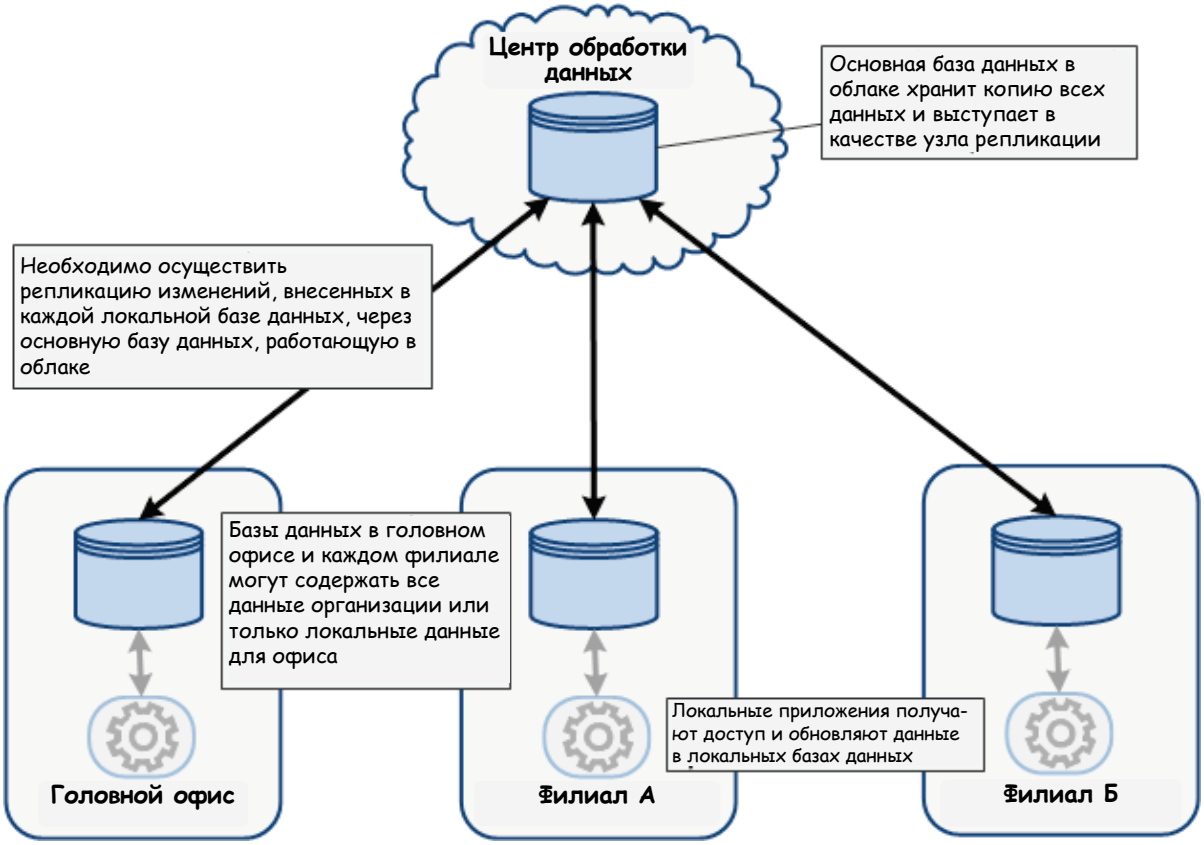



**Рисунок 5**

**Топология Д: синхронизация локальных баз данных через облако**

## Синхронизация данных из различных источников

**Описание: приложения и службы изменяют данные, и эти изменения должны распространяться на все экземпляры базы данных.**

Данные изменчивы, статичность не характерна для большей части информации. Приложения неизбежно добавляют, обновляют и удаляют записи. В среде с реализованными функциями репликации необходимо убедиться, что все эти изменения распространяются на все соответствующие экземпляры источника данных. Решения для синхронизации данных могут быть достаточно дорогостоящими, поскольку предъявляются высокие требования к пропускной способности сети, и, возможно, процесс синхронизации придется реализовать в виде периодической задачи, в рамках которой применяется пакет обновлений. Поэтому вы должны быть готовы сбалансировать требования к согласованности данных и расходы на выполнение синхронизации. Кроме того, ваша бизнес-логика должна ориентироваться на то, что абсолютная согласованность недостижима, но в конечном итоге данные все-таки будут согласованы (эта процедура описывалась ранее в данном приложении).

В процессе выбора стратегии синхронизации необходимо рассмотреть следующие вопросы:

- Какие данные вы будете синхронизировать и как будете устранять конфликты в процессе синхронизации?

  Ответ на этот вопрос во многом зависит от того, когда и где данные обновляются (см. топологии в предыдущем разделе). Например, в топологии Г изменения в основной источник данных вносятся только приложениями, работающими локально, поэтому синхронизация будет сводиться к копированию локальных изменений в каждый центр обработки данных в облаке. Это односторонняя операция (по направлению из локальной системы в облако) с крайне низкой

вероятностью возникновения конфликтов при синхронизации. Локальная основная база данных содержит определяющую копию данных, на основе которой изменяется информация, хранящаяся в центрах обработки данных в облаке.

Когда данные изменяются облачными службами, а не локальными приложениями (топология В), и если эти данные секционируются центрами обработки данных, возможности для конфликтов также отсутствуют (службы в двух разных центрах обработки данных не смогут обновлять одну и ту же информацию), при этом процесс синхронизации фактически является односторонним и мгновенным. В данном случае центры обработки данных в облаке хранят определяющую информацию и перезаписывают данные, размещенные локально.

В топологиях А, Б и Д данные в любом месте — локально или в облаке — могут изменяться и приложениями, и службами, процесс синхронизации осуществляется по многим направлениям, поскольку каждая база данных должна быть синхронизирована со всеми остальными. Данные не секционируются, поэтому существует вероятность конфликтов в процессе внесения изменений, и вы должны определить стратегию для управления такими ситуациями.

**Мнение Джаны**

Полная синхронизация по всем направлениям, в которой задействованы все участвующие в репликации реляционные базы данных, может представлять собой весьма ресурсоемкую операцию. Неизбежные задержки, связанные с передачей и выполнением большого числа изменений на целом ряде узлов, могут привести к тому, что некоторые данные, содержащиеся в одной или нескольких базах данных, будут несогласованны между собой до тех пор, пока полная синхронизация не завершится. Чтобы свести к минимуму задержки и ресурсы, необходимые для синхронизации баз данных, вам следует тщательно проанализировать ситуацию и определить, какие данные ваши приложения и службы должны реплицировать, могут ли ваши приложения и службы работать с потенциально устаревшими данными, и какие данные должны быть доступны только для чтения на каждом узле.

- Каковы ожидаемые объемы данных, подлежащих синхронизации? В случае наличия большого количества изменчивых данных, репликация каждой операции вставки, обновления и удаления может привести к слишком интенсивной нагрузке на сеть и вычислительные мощности. Это негативно отразится на производительности каждого источника данных, и, возможно, сведет на нет преимущества, ради которых репликация данных, в первую очередь, и была организована. В топологии В, например, если каждая облачная служба выполняет большое количество обновлений, поддерживать копию массива данных в облаке будет слишком сложно с точки зрения производительности.
- Когда необходимо синхронизировать данные? Должен ли каждый экземпляр базы данных поддерживаться в полностью актуальном состоянии все время? Вашей системе требуется постоянная полная целостность транзакций? Если так, то репликация будет не самым подходящим решением, поскольку синхронизация обязательно превратится в непрерывный процесс, все изменения будут отправляться другим участникам системы сразу же после того, как они происходят, при этом соответствующие ресурсы каждого источника данных будут на время обновления блокироваться, чтобы предотвратить возможные несоответствия. В данном случае использование единого централизованного источника данных представляется более целесообразным, чем репликация.

## Взаимосвязанные проблемы

Эффективность репликации данных в большой степени зависит от параметров сети с точки зрения безопасности, надежности и производительности. Системные требования и особенности реализации приложений также могут оказывать существенное влияние на эффективность выбранного вами подхода к репликации. В следующих разделах возможные проблемы рассматриваются подробнее.

### Безопасность доступа к данным

Каждый источник данных, будь то БД SQL Azure или другие хранилища, должен обеспечивать защиту информации, которая в нем содержится, и предотвращать несанкционированный доступ. Это требование применяется к процессу синхронизации, а также стандартному циклу доступа к данным. Сетевые пакеты с подлежащими репликации данными также должны быть защищены, поскольку нарушение политики безопасности на этом уровне может привести к распространению поврежденной информации в несколько экземпляров важных элементов данных.

### Согласованность данных и время отклика приложений

Согласованность данных и время отклика приложений — противоречивые требования, которые необходимо сбалансировать, чтобы удовлетворить потребности пользователей.

Если необходимо обеспечить высокий уровень согласованности всех задействованных в процессе репликации баз данных, следует принять меры, не позволяющие конкурирующим приложениям получать доступ к данным, которые находятся в состоянии изменения на одном из узлов системы. Такой подход зависит от логики приложения для блокировки элементов данных и их копий перед внесением в них изменений. После завершения обновления блокировка снимается. Когда данные заблокированы, никакие другие приложения не могут получить к ним доступ, что отрицательно сказывается на скорости отклика системы с точки зрения пользователей. Как уже отмечалось ранее в данном приложении, в рамках многих распределенных систем немедленная и абсолютная согласованность данных может быть не так важна, как поддержание высокой скорости отклика приложений. Пользователи системы хотят быстро выполнять необходимые операции, а поскольку информация не теряется и в конечном итоге согласовывается, такой подход они сочтут удовлетворительным.

### Целостность и надежность

Даже если это решение, для которого немедленное обеспечение целостности данных не является критическим требованием, в системе в любом случае должна быть предусмотрена надежная процедура обновления данных с целью поддержания целостности информации. Например, в приложении Orders, о котором мы говорили ранее, заказ должен быть выполнен, если система приняла его от клиента. Данные, указанные при размещении заказа, не должны быть утеряны, и процесс обработки и выполнения заказа должен быть завершен. Именно поэтому в любом решении, которое осуществляет репликацию данных между различными БД, должен быть реализован надежный механизм транспортировки и обработки этих данных. Если возник сбой какого-либо компонента системы синхронизации, необходимо предусмотреть возможность перезапуска этого компонента без потери или дублирования данных.

> **Мнение По**
>
> Необходимо помнить, что сеть является важнейшим компонентом с точки зрения влияния на надежность. Под надежностью решения в данном случае понимается его устойчивость к сбоям в сети.

# Платформа Windows Azure и связанные с ней технологии

Когда вы осуществляете развертывание баз данных в облаке с помощью SQL Azure, служба SQL Azure Data Sync поможет вам настроить репликацию и синхронизацию между этими базами данных и базами данных SQL Server, работающими локально. Это облачная служба синхронизации на основе Microsoft Sync Framework. При помощи портала управления Windows Azure вы сможете быстро настроить самые распространенные сценарии синхронизации между локальными базами данных SQL Server и базами данных SQL Azure, которые работают в облаке. Кроме того, служба SQL Azure Data Sync совместима с платформой Microsoft Sync Framework 2.1, поэтому вы можете использовать пакет Sync Framework SDK для реализации собственной стратегии синхронизации и включения дополнительной логики для проверки, если это необходимо.

> **Примечание**
>
> Решение SQL Azure Data Sync совместимо с SQL Server 2005 Service Pack 2 и более поздними версиями.

Пакет Sync Framework SDK будет полезен в рамках сценариев, подразумевающих внедрение собственного подхода к синхронизации, который невозможно или трудно реализовать с помощью портала управления. Например, вы можете создать свои собственные службы синхронизации для локальных баз данных и мобильных устройств пользователей.

Альтернативный подход заключается в реализации собственного механизма рассылки обновлений в базы данных с использованием системы обмена сообщениями. Приложение-отправитель и приложение-прослушиватель используют специализированную логику для публикации обновлений и синхронизации их с данными на выходе. Топики и подписки шины интеграции предоставляют идеальную инфраструктуру для реализации этого сценария.

В следующих разделах приводятся дополнительные сведения об использовании SQL Azure Data Sync, Sync Framework SDK, а также топиков и подписок шины интеграции для осуществления репликации в рамках некоторых типичных сценариев.

## Репликация и синхронизация при помощи SQL Azure Data Sync

Использование SQL Azure Data Sync для реализации синхронизации в SQL Server обеспечивает многочисленные преимущества, в том числе:

- **Эластичная масштабируемость**. Служба SQL Azure Data Sync работает в облаке и масштабируется автоматически по мере увеличения объема данных и количества узлов, участвующих в процессе синхронизации.
- **Простая конфигурация**. Вы можете использовать портал управления для создания топологии синхронизации. Портал предоставляет мастеров, которые направляют каждый ваш шаг в процессе настройки и позволяют указать, какие данные подлежат синхронизации. Вы можете также указать, будет ли репликация односторонней или двунаправленной. На панели Sync Group портал формирует графическое представление топологии и ее текущего состояния.
- **Синхронизация по расписанию**. Вы можете указать, как часто должен выполняться процесс синхронизации, и легко изменить этот параметр даже после того, как синхронизация была настроена. Портал управления также позволяет запустить синхронизацию немедленно по требованию.

- **Предварительно настроенные политики обработки конфликтов**. Служба SQL Azure Data Sync позволяет выбрать методы разрешения любых конфликтов, которые были обнаружены во время синхронизации, выбрав одну из множества встроенных политик.
- **Комплексные функции ведения журналов**. SQL Azure Data Sync заносит в журнал записи обо всех событиях и операциях. Портал управления позволяет изучить эту информацию и применить к ней различные фильтры, помогая быстро определить причину возникновения проблем и предпринять необходимые корректирующие действия.

В следующих разделах содержится информация о принципах работы службы SQL Azure Data Sync, а также рекомендации по ее использованию в рамках ряда наиболее распространенных сценариев.

## Рекомендации по настройке SQL Azure Data Sync

В процессе настройки SQL Azure Data Sync вы должны принять ряд решений относительно данных, подлежащих репликации, и местоположения соответствующих баз данных. В данном разделе содержатся рекомендации по определению ключевых элементов архитектуры синхронизации.

### *Создание элементов Sync Group и Sync Dataset*

SQL Azure Data Sync организует процесс синхронизации, определяя *группу синхронизации (sync group)*. Группа синхронизации — это совокупность рядовых баз данных, которые должны быть синхронизированы, включая центральную базу данных, которая выступает в качестве главной точки синхронизации. Все рядовые базы данных, участвующие в топологии, синхронизируются через центральный узел, посылая собственные обновления и получая информацию об изменениях, внесенных другими базами данных.

Параллельно с определением группы синхронизации вы также определяете *набор данных синхронизации (sync dataset)* — таблицы, строки и столбцы, подлежащие синхронизации. Не обязательно выбирать все таблицы в базе данных, также можно определить фильтры для ограничения диапазона строк, которые будут синхронизированы. Тем не менее в каждой таблице, которая участвует в наборе данных синхронизации, должен присутствовать первичный ключ, в противном случае синхронизация невозможна. Кроме того, хотя вы не должны включать каждый столбец из каждой задействованной таблицы, необходимо включить все столбцы, которые не поддерживают пустые значения, иначе синхронизация также не будет выполнена.

**Примечание**

SQL Azure Data Sync создает триггеры для каждой таблицы в группе синхронизации. Эти триггеры отслеживают изменения, вносимые в данные в каждой таблице в группе синхронизации. Дополнительные сведения о триггерах, создаваемых службой SQL Azure Data Sync, представлены в разделе «Вопросы использования Azure Data Sync»: http://sqlcat.com/sqlcat/b/technicalnotes/archive/2011/12/21/considerations-when-using-data-sync.aspx.

Важно понимать, что служба SQL Azure Data Sync накладывает некоторые ограничения на типы столбцов в таблицах, которые участвуют в процессе синхронизации. Эти ограничения связаны с платформой Sync Framework, на которой основана служба SQL Azure Data Sync. Платформа Sync Framework предназначена для работы с различными системами управления базами данных, а не только SQL Server, и поэтому поддерживаемые типы ограничиваются общими типами для всех основных систем управления базами данных. Например, вы не можете синхронизировать столбцы на основе пользовательских и пространственных типов данных или типов CLR. Полный перечень поддерживаемых

и неподдерживаемых типов представлен на странице «SQL Azure Data Sync. Поддерживаемые типы данных SQL Azure Data Types»: http://msdn.microsoft.com/en-us/library/hh667319.aspx.

**Мнение Маркуса**

Если вы попытаетесь включить в создаваемый набор данных синхронизации столбцы с неподдерживаемыми типами данных, эти столбцы будут игнорироваться при копировании.

**Мнение Бхарата**

Один и тот же набор данных синхронизации применяется глобально для всех рядовых баз данных в группе синхронизации. Вы определяете набор данных синхронизации при добавлении первой рядовой базы данных в группу синхронизации и, если необходимо, таблицы, поддерживающие набор данных синхронизации, будут автоматически добавлены в рядовые базы данных, которые включаются в группу синхронизации в дальнейшем. Однако, после того как вы определили набор данных синхронизации для группы синхронизации, вы не можете изменить определение этого набора данных. Вы должны будете удалить эту группу синхронизации и создать новую с новым набором данных синхронизации.

### *Реализация схемы базы данных для рядовых баз данных*

В общем случае схема данных, подлежащих репликации, возможно, уже существует в локальной базе данных или базе данных SQL Azure. В ходе развертывания группы синхронизации, если необходимые таблицы отсутствуют в других рядовых базах данных или в центральной БД, служба SQL Azure Data Sync автоматически создаст их на основе определения набора данных синхронизации. В этом случае в процессе развертывания будут создаваться только столбцы, указанные в наборе данных синхронизации, при этом будет добавлен индекс для первичного ключа каждой таблицы. Процесс развертывания значительно упрощает репликацию схемы набора данных синхронизации, однако копия не всегда будет идентична оригиналу из-за различий между SQL Azure и SQL Server.

Кроме того, никакие индексы, помимо индекса, который ссылается на первичный ключ, созданы не будут, и это может оказать влияние на скорость выполнения запросов к полученной после репликации базе данных. Поэтому, чтобы обеспечить абсолютную точность и избежать неожиданных результатов, рекомендуется подготовить SQL-сценарий, содержащий команды, необходимые для создания каждой подлежащей репликации таблицы вместе с соответствующими индексами. Вы можете также определить любые представления и хранимые процедуры, которые могут потребоваться каждой рядовой базе данных, поскольку они не реплицируются автоматически. Вы можете выполнить этот сценарий поочередно для каждой базы данных перед выполнением репликации.

**Мнение Маркуса**

При помощи таких инструментов, как Microsoft SQL Server Management Studio, вы можете разрабатывать и редактировать SQL-сценарии для создания таблиц, представлений и хранимых процедур для каждой рядовой базы данных. При наличии соответствующих полномочий, вы также можете подключаться к каждой рядовой базе данных с помощью SQL Server Management Studio и выполнять эти сценарии.

### *Обработка конфликтов синхронизации*

В процессе синхронизации служба SQL Azure Data Sync поочередно подключается к каждой рядовой базе данных с целью получения информации об обновленных элементах и затем вносит соответствующие изменения в центральной базе данных. Любые обновления, ранее переданные центральному узлу из другой рядовой базы данных, передаются в эту базу данных и применяются.

Определять и разрешать конфликты следует в центральной базе данных. SQL Azure Data Sync позволяет выбрать одну из двух политик обработки конфликтов:

- **Приоритет отдается центральному узлу**. Если данные в центральной БД уже были изменены, то эти изменения вносятся в соответствующую рядовую базу данных. Фактически, это означает, что доминирует первая рядовая база данных, которая провела синхронизацию с центральной БД.
- **Приоритет отдается клиенту**. Если данные были изменены в рядовой базе данных, то соответствующие изменения вносятся в центральную БД вместо любых предыдущих изменений. В отличие от политики приоритета центрального узла, в этом случае доминирует последняя рядовая база данных, которая провела синхронизацию с центральной БД.

**Мнение По**

Процесс синхронизации последовательно обращается к каждой рядовой базе данных и осуществляет необходимые обновления с целью синхронизации этой и центральной БД. Предыдущие базы данных не получат изменения, возникшие в результате синхронизации с последующими базами данных. Полная идентичность рядовых баз данных возможна только после выполнения двух процессов синхронизации в рамках группы синхронизации.

В ходе синхронизации каждый пакет обновлений применяется как транзакция, при этом либо успешно применяются изменения, либо выполняется откат. Однако эти пакетные транзакции не обязательно отражают бизнес-операции, которые выполняются в системе. Например, изменения, внесенные в двух таблицах в результате бизнес-операции, могут попасть в различные пакеты в процессе синхронизации.

Кроме того, каждый процесс синхронизации учитывает только те изменения, которые были действительны в тот момент в каждой базе данных. Если в отношении строки было выполнено несколько обновлений между процессами синхронизации, то только окончательное обновление подлежит репликации, поскольку служба SQL Azure Data Sync не ведет журнал всех изменений, внесенных между процессами синхронизации.

**Мнение Бхарата**

Следует тщательно продумать политику разрешения конфликтов, поскольку она распространяется на все базы данных в группе синхронизации. Кроме того, необходимо выбрать политику при создании группы синхронизации; ее нельзя изменить, не удалив группу и не создав ее заново.

Обратите внимание, хотя вы можете выбрать политику разрешения конфликтов, вы не можете на данный момент определять порядок, в котором задействованные базы данных синхронизируются с центральной. В идеале, вы должны разрабатывать свое решение таким образом, чтобы свести к минимуму вероятность возникновения конфликтов. В рамках типового сценария распределенные приложения, работающие на разных узлах, как правило, управляют своими подмножествами данных организации, поэтому возможность конфликтов снижается. Помните, что основной целью репликации

является распространение обновлений, выполняемых на одном узле, на все другие узлы, чтобы в рамках всей системы сформировать одинаковое представление данных.

Если вы хотите обеспечить максимальную эффективность политики урегулирования конфликтов, можно разделить топологию репликации на несколько групп синхронизации с одной рядовой и одной центральной базой данных. Расписание синхронизации для каждой группы определяет порядок, в котором каждая рядовая база данных будет синхронизироваться с центральной. Для группы синхронизации, включающей рядовую базу данных с высоким приоритетом обновлений, которые должны всегда превалировать, можно выбрать вариант «Приоритет отдается клиенту» для политики урегулирования конфликта, таким образом, эти изменения будут всегда реплицироваться.

Для политики других групп синхронизации может быть выбран вариант «Приоритет отдается центральному узлу», таким образом, изменения, внесенные в базу данных с высоким приоритетом, всегда будут распространяться на другие рядовые базы данных. Эту топологию можно реализовать множеством способов. Например, вы можете разместить несколько рядовых баз данных в группу синхронизации с политикой «Приоритет отдается центральному узлу», если, скорее всего, ни одна из этих баз данных не будет содержать изменения, которые могут конфликтовать друг с другом.

**Мнение Маркуса**

Чтобы избежать проблем, связанных с конфликтующими значениями первичного ключа, не используйте столбцы с автоматически сгенерированным ключом в реплицируемых таблицах. В качестве альтернативы можно использовать значение, которое точно будет уникальным, например идентификатор GUID.

Конфликты чаще всего возникают в результате двунаправленной синхронизации. Чтобы свести к минимуму вероятность конфликтов, можно настроить одностороннюю репликацию и указать направление синхронизации для каждой рядовой базы данных в группе синхронизации по отношению к центральной базе данных. Дополнительная информация представлена в разделе «Выбор направления синхронизации для базы данных» далее в настоящем приложении.

Рекомендуем обратить особое внимание на определение столбцов в реплицируемых таблицах, поскольку это может оказать значительное влияние на вероятность возникновения конфликта. Например, если вы определяете столбец первичного ключа в реплицируемой таблице при помощи атрибута **IDENTITY** в SQL Server, то SQL Server автоматически создаст значения для этого столбца в рамках монотонно возрастающей последовательности, которая, как правило, начинается с 1, и для каждой новой строки значение увеличивается на 1. Если строки добавляются в нескольких рядовых базах данных в группе синхронизации, некоторые из этих строк могут иметь одинаковые первичные ключи, в результате чего в ходе синхронизации таблиц возникнет конфликт. В итоге останется только одна строка, остальные будут удалены. Результаты могут быть просто катастрофическими. Если, например, это были заказы разных клиентов, то данные всех этих заказов будут потеряны, поскольку после применения политики разрешения конфликтов в базе данных останется только одна запись!

Чтобы избежать подобных ситуаций, не используйте столбцы с автоматически сгенерированными ключами в реплицируемых таблицах, вместо этого рекомендуется использовать значение, которое точно будет уникальным, например идентификатор GUID.

### *Определение местоположения и размера центральной базы данных*

Центральной должна быть база данных SQL Azure. После синхронизации со всеми рядовыми базами данных, она будет содержать окончательную и самую актуальную версию данных. Местоположение

центральной базы данных оказывает существенное влияние на скорость осуществления процесса синхронизации. Вы должны разместить ее в центре обработки данных, который географически расположен ближе всего к наиболее активным рядовым базам данных, независимо от того, облачные это базы или локальные. Это поможет свести к минимуму сетевые задержки, связанные с передачей потенциально больших объемов данных через Интернет. Если ваши базы данных практически равномерно распределены по всему миру, а объем обновлений и трафик запросов примерно одинаков для каждой из них, то рекомендуется разместить центральную БД в центре обработки данных, расположенном как можно ближе к вашим наиболее приоритетным узлам.

Служба SQL Azure Data Sync осуществляет репликацию и синхронизацию данных между вашими БД через центральный узел. Вы можете развернуть один экземпляр сервера SQL Azure Data Sync Server для каждой своей подписки Windows Azure, а также определить регион, где этот сервер будет работать. В идеале, вы должны разместить этот сервер в том же регионе, в котором планируете развернуть центральную базу данных.

Вы создаете центральную базу данных вручную, и она должна быть, по крайней мере, такого же объема, как самая большая рядовая база данных. SQL Azure в настоящее время не поддерживает автоматическое увеличение баз данных, поэтому если размер созданной вами центральной базы данных окажется недостаточным, служба синхронизации может дать сбой. Также необходимо помнить, что когда вы настраиваете синхронизацию, служба SQL Azure Data Sync создает дополнительные таблицы метаданных в ваших базах данных с целью отслеживания изменений, и вы должны учитывать эти таблицы в процессе определения размера центральной базы данных.

Центральная база данных лежит в основе процесса синхронизации, кроме того, она содержит в точности те же данные, что и любая рядовая база данных SQL Azure в группе синхронизации. Вы можете вставлять, обновлять и удалять данные в этой базе, и эти изменения будут распространены в рамках всей группы синхронизации. В некоторых ситуациях вы можете использовать одну из баз данных SQL Azure, изначально задействованных в группе синхронизации, в качестве центрального узла. Например, в качестве центральной вы можете выбрать базу данных SQL Azure самого активного узла. Такая стратегия поможет свести к минимуму задержки, связанные с передачей данных по сети, и тем самым ускорить процесс синхронизации.

Однако все другие рядовые базы данных в группе синхронизации будут периодически синхронизироваться с этой базой данных, и связанная с этими операциями нагрузка может негативно повлиять на производительность этой базы, особенно если таблицы в наборе данных синхронизации содержат сложную совокупность индексов. Вы должны сбалансировать издержки, связанные с тем, что база данных будет выступать в качестве центрального узла для группы синхронизации, с возможной потерей времени на синхронизацию этой базы данных, если центральный узел будет размещен где-то в другом месте.

### *Составление расписания синхронизации для группы синхронизации*

Синхронизация — это периодическая операция, а не непрерывный процесс.Вы можете задать расписание синхронизации для группы синхронизации, а также принудительно запустить эту операцию вручную с помощью портала управления. Когда формируете график, необходимо выбрать частоту синхронизации, которая оптимально подходит для вашего решения.Если интервалы слишком продолжительные, рядовые базы данных будут содержать устаревшую информацию в течение длительного времени, а слишком короткие промежутки могут привести к тому, что новый процесс синхронизации начнется еще до завершения предыдущего, и попытка будет неудачной. Как описано выше, продолжительность процесса синхронизации зависит от местоположения центральной базы данных.

Длительность этого процесса также будет зависеть от объема данных, которые необходимо синхронизировать. Чем больше интервал между процессами синхронизации, тем больший объем данных отправляется и принимается центральным узлом в процессе синхронизации. Кроме того, по мере увеличения периода синхронизации возрастает вероятность возникновения конфликтов, и в процессе синхронизации придется тратить время на их устранение, что еще больше увеличивает продолжительность выполняемой задачи. Возможно, вам придется определять оптимальный период синхронизации на основе наблюдений и регулировать его в соответствии с установленными требованиями к своевременности данных со стороны ваших приложений и служб.

**Мнение Бхарата**

Как и политика разрешения конфликтов, расписание синхронизации распространяется на все базы данных в группе синхронизации. Тем не менее вы можете изменить этот график в любое время — проанализировав результаты синхронизации данных через различные промежутки времени, вы выберите период, наиболее подходящий для своей системы.

Наконец, вы должны помнить, что тарификация услуг, связанных с использованием платформы SQL Azure, осуществляется с учетом входящего и исходящего трафика в центрах обработки данных SQL Azure. Соответственно, чем больше данных вы синхронизируете между центрами обработки и чем чаще вы выполняете эти операции, тем выше стоимость услуг.

### *Выбор направления синхронизации для базы данных*

При добавлении рядовой базы данных в группу синхронизации вы указываете направление синхронизации. Поддерживаемые виды синхронизации:

- **Двунаправленная**. В рядовую базу данных можно вносить изменения и загружать их в центральную базу данных, а также получать из нее обновления. Это, вероятно, наиболее распространенный вид синхронизации, реализованный во многих организациях.
- **По направлению к центральному узлу**. В рядовую базу данных можно вносить изменения и загружать их в центральную базу данных, но при этом нельзя получать из нее обновления. Этот вид синхронизации используется, например, в топологии Г (локальный основной репозиторий с односторонней синхронизацией по направлению к облаку), описанной ранее в этом приложении, и других подобных сценариях. Работающая в облаке служба обновляет облачную рядовую базу данных, а также копирует изменения в локальную базу данных по мере их возникновения. Локальная база данных может быть настроена на синхронизацию с центральной базой данных. Изменения, внесенные каждой службой в облаке, в ходе синхронизации могут передаваться в рядовые базы данных для других служб через центральный узел. Локальную базу данных не обязательно синхронизировать, поскольку она уже содержит все обновления.
- **По направлению от центрального узла**. Рядовая база данных может получать информацию об изменениях из центральной базы данных, но не будет загружать в нее облачные изменения. Этот вид синхронизации полезен в рамках сценариев, напоминающих топологию Г. В данном случае можно настроить синхронизацию рядовых баз данных с центральной. Любые изменения, внесенные в облаке, уже отражены в локальной базе данных работающими в облаке службами, поэтому репликации подлежат только изменения, осуществляемые другими службами, которые расположены на других узлах и также обновляют локальную базу данных.

**Мнение Бхарата**

Направление синхронизации — это атрибут каждой рядовой базы данных. Для каждой базы данных в группе синхронизации можно задавать собственное направление синхронизации.

На рисунке 6 показан обновленный вариант топологии Г с центральной базой данных и службой Data Sync, которая необходима службе SQL Azure Data Sync.

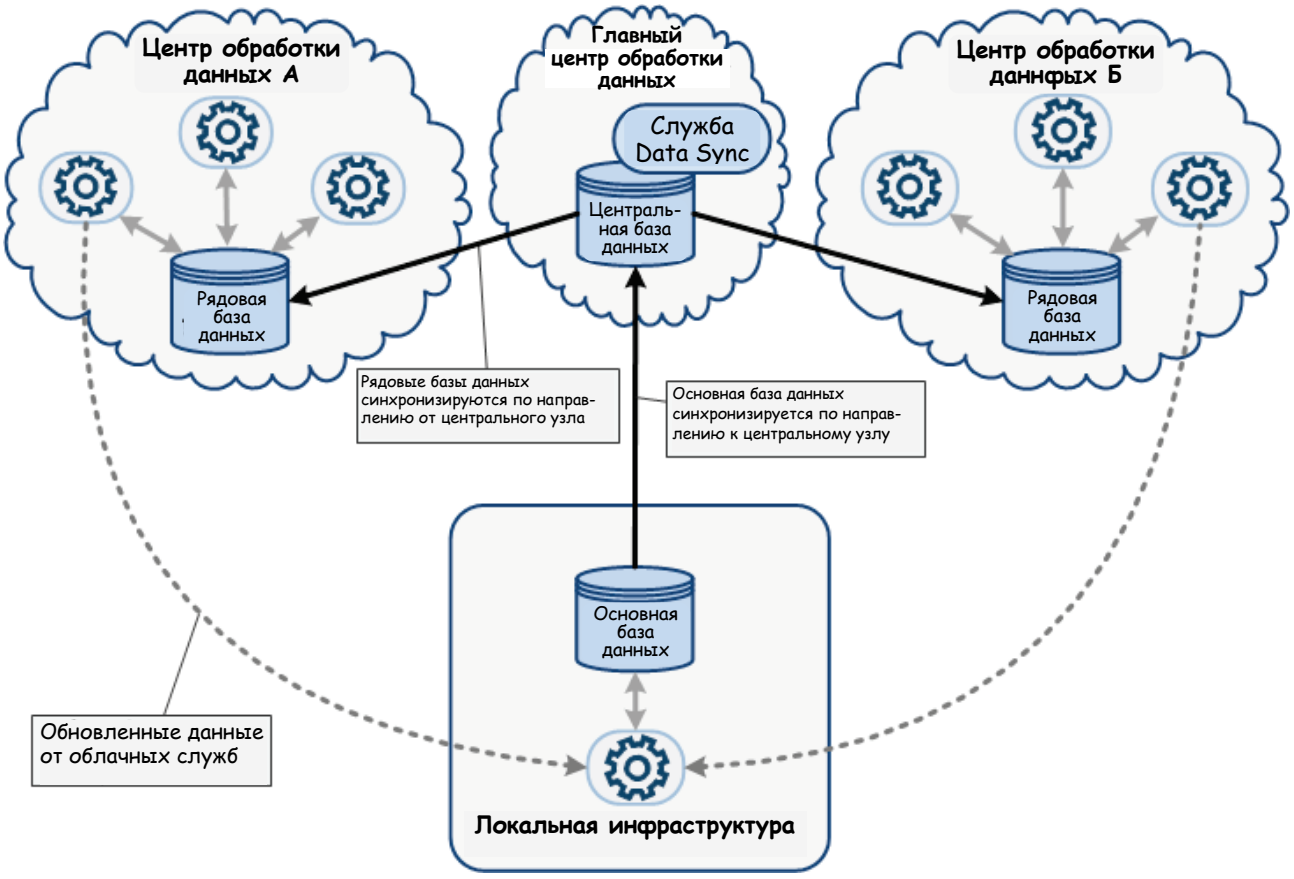



**Рисунок 6**

**Определение направления синхронизации для баз данных, участвующих в топологии Г**

**Примечание**

На рисунке 6 центральный узел представлен в виде отдельной базы данных, однако, за исключением ситуаций, описанных в конце раздела «Определение местоположения и размера центральной базы данных», одна из рядовых баз данных в облаке сможет взять на себя эту роль. Примеры в разделе «Рекомендации по использованию службы SQL Azure Data Sync» иллюстрируют данный подход.

### *Необходимо избегать циклов синхронизации*

Рядовая база данных может быть включена в более чем одну группу синхронизации. Однако такая конфигурация может привести к образованию цикла синхронизации. Цикл возникает тогда, когда в результате процесса синхронизации в одной группе синхронизации требуется синхронизация в другой группе, а эта вторая синхронизация обуславливает необходимость повторной синхронизации в первой группе, после чего опять требуется синхронизация во второй группе и так далее. Циклы синхронизации бесконечны, большие объемы данных будут неоднократно записываться и переписываться, что приводит к снижению производительности и росту расходов.

В ходе настройки группы синхронизации вы должны быть предельно внимательны, чтобы предотвратить циклы синхронизации. Для этого необходимо оценить роль всех баз данных, которые участвуют в нескольких группах синхронизации, и выбрать подходящую политику разрешения конфликтов для каждой группы синхронизации. Также рекомендуется применить фильтры, чтобы одна и та же строка в таблице не могла участвовать в различных группах синхронизации, и установить направление синхронизации для каждой базы данных. Более подробное описание циклов синхронизации и обстоятельств их возникновения представлено в разделе «Циклы синхронизации»: http://msdn.microsoft.com/en-us/library/hh667312.aspx.

### Рекомендации по использованию SQL Azure Data Sync

Службу SQL Azure Data Sync можно использовать для реализации топологий репликации, представленных ранее в данном приложении. Вы можете применить эти топологии с SQL Azure Data Sync для многих распространенных сценариев (см. список ниже).

> Если вы используете диспетчер трафика Windows Azure для направления запросов в центр обработки данных, необходимо учитывать, что службам, работающим в разных центрах обработки данных, может быть представлена разная информация, если каждый центр обработки данных использует собственную копию базы данных. Возможность возникновения такой ситуации обусловлена тем, что процесс синхронизации не может завершиться на всех узлах одновременно, поэтому обновления, видимые в одном центре обработки данных, возможно, еще не были переданы в другие центры обработки данных.
>
> Дополнительная информация об использовании диспетчера трафика Windows Azure представлена в «Приложении Д. Максимизация масштабируемости, доступности и производительности». Пример компании Trey Research, которая применяла диспетчер трафика Windows Azure, представлен в главе 6 — «Максимизация масштабируемости, доступности и эффективности приложения Orders».

- **Локальные приложения получают доступ к локальной базе данных SQL Server. Облачные службы используют копию тех же данных. Любые изменения, возникшие на любом узле, в конечном итоге должны быть переданы на все другие узлы, хотя обновление происходит не в один момент.**

  Это, вероятно, самый распространенный сценарий использования службы SQL Azure Data Sync, он описывает ситуацию, характерную для топологии A (двунаправленная синхронизация всех баз данных локально и в облаке). В качестве примера такого сценария рассмотрим базу данных, содержащую информацию о клиентах и заказах. Приложение, работающее локально, управляет информацией о клиентах, в то время как сами клиенты используют веб-приложение, работающие в облаке, которое позволяет размещать новые заказы. Веб-приложению требуется доступ к информации о клиентах, которая управляется локальным приложением, а локальные алгоритмы регулярно запрашивают сведения о заказах, чтобы обновить информацию об их статусе после подтверждения доставки или оплаты.

  В данном примере важную роль играет время отклика, но ни локальному, ни веб-приложению не требуется доступ к самой актуальной информации. Если данные будут доступны через некоторое время, этого достаточно. Поэтому, чтобы свести к минимуму задержки в сети и гарантировать работоспособность веб-приложения, используется база данных SQL Azure, размещенная в том же центре обработки данных, что и приложение, при этом локальное приложение использует локальную базу данных SQL Server.

Служба SQL Azure Data Sync позволяет копировать и предоставлять локальному и облачному приложению доступ к информации о клиентах и заказах с помощью двунаправленной синхронизации, как показано на рисунке 7. Обратите внимание, что на этой схеме база данных SQL Azure в центре обработки данных A также выступает в качестве центрального узла синхронизации с развернутой службой Data Sync.

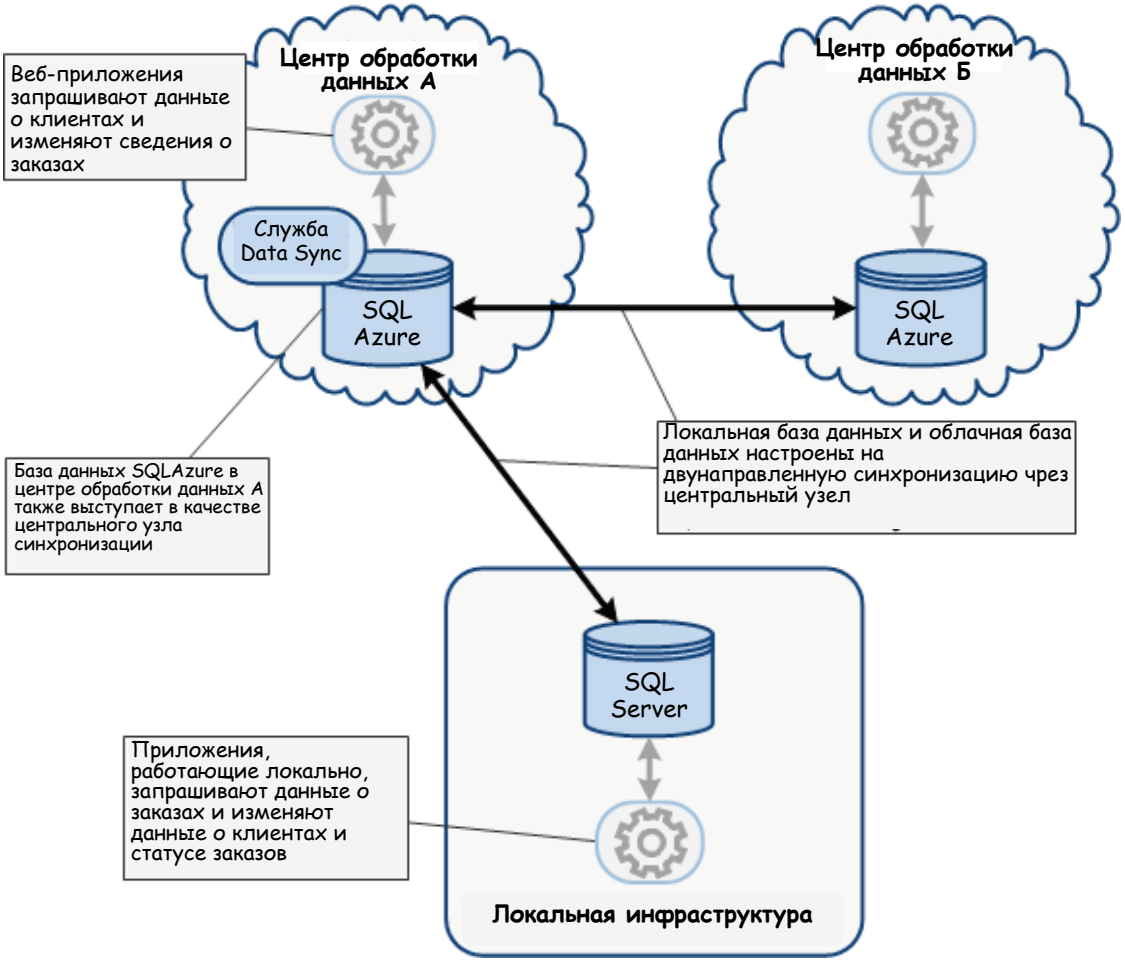



**Рисунок 7**

**Совместное использование информации локальным и облачным приложением**

- **Вы переместили логику бизнес-приложений в службы, работающие в облаке. Бизнес-приложения ранее использовали данные, которые хранились в базе данных SQL Server. Службы были распределены между различными центрами обработки данных, и базу данных SQL Server перенесли в SQL Azure. Чтобы свести к минимуму задержки, которые могут возникнуть в сети, у каждого центра обработки данных есть своя копия базы данных SQL Azure.**

Этот сценарий воспроизводит топологию Б (двунаправленная синхронизация с участием только облачных баз данных). В данном примере логика приложения, которая обращается к базе данных, была полностью перемещена в облако, поэтому локальная база данных была ликвидирована. Однако всем облачным приложениям требуется доступ к одним и тем же данным, и они могут изменять эту информацию, поэтому каждый экземпляр базы данных SQL Azure необходимо периодически синхронизировать со всеми другими экземплярами. Репликация будет двунаправленной. На рисунке 8 показана структура возможного решения — база данных SQL Azure в центре обработки данных A также выполняет роль центрального узла

синхронизации. В данном примере любое приложение может запрашивать и изменять любые данные. Следовательно, может возникнуть необходимость внесения изменений в логику приложения с целью обработки данных, которые могут оказаться устаревшими до следующего цикла синхронизации.

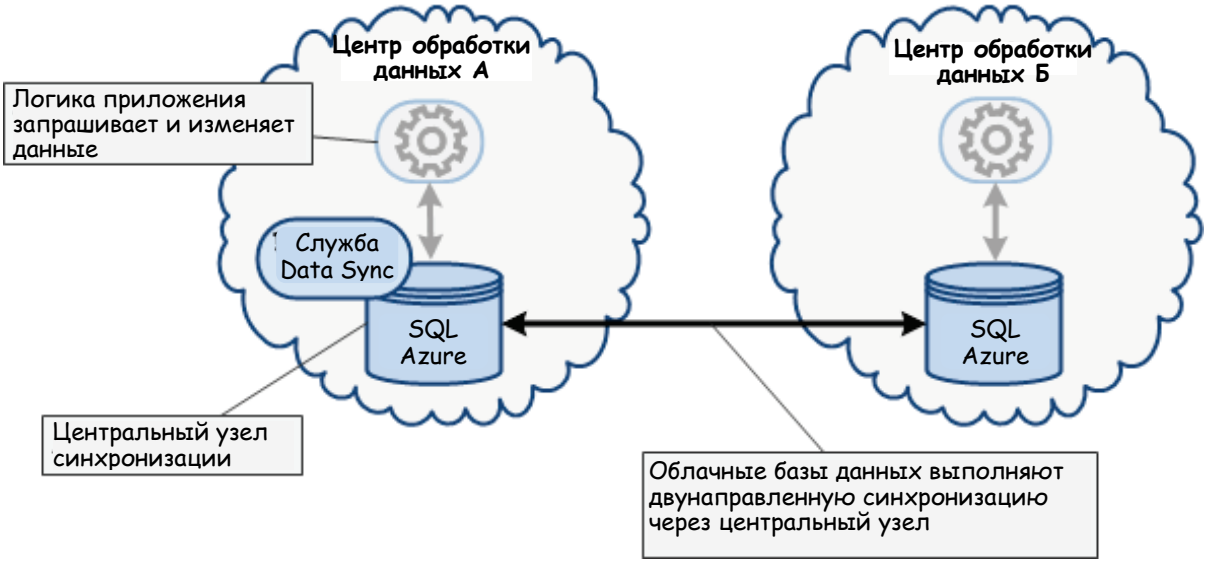



**Рисунок 8**

### Репликация данных между центрами обработки данных в облаке

- **Вы должны хранить свои данные в локальной базе данных SQL Server, доступной для работающих в облаке служб. Эти службы только запрашивают данные, не изменяя их. Все изменения выполняются приложениями, работающими локально.**

Это упрощенный вариант топологии Г (локальный основной репозиторий с односторонней синхронизацией по направлению к облаку), описанной ранее. В этом сценарии службы, которые запрашивают данные, работают удаленно от вашей локальной базы данных. Чтобы свести к минимуму время отклика, вы можете организовать репликацию данных в одну или несколько баз данных SQL Azure, размещенных в тех же центрах обработки данных, что и каждая из этих служб. Используя службу SQL Azure Data Sync, вы можете публиковать данные, хранящиеся локально, и периодически синхронизировать любые обновления, внесенные локальными приложениями, с базами данных в облаке. На рисунке 9 представлен пример. Эта конфигурация требует односторонней репликации. Локальная база данных синхронизируется по направлению к центральной базе данных в облаке, и каждая рядовая база данных SQL Azure синхронизируется по направлению от центральной базы данных.

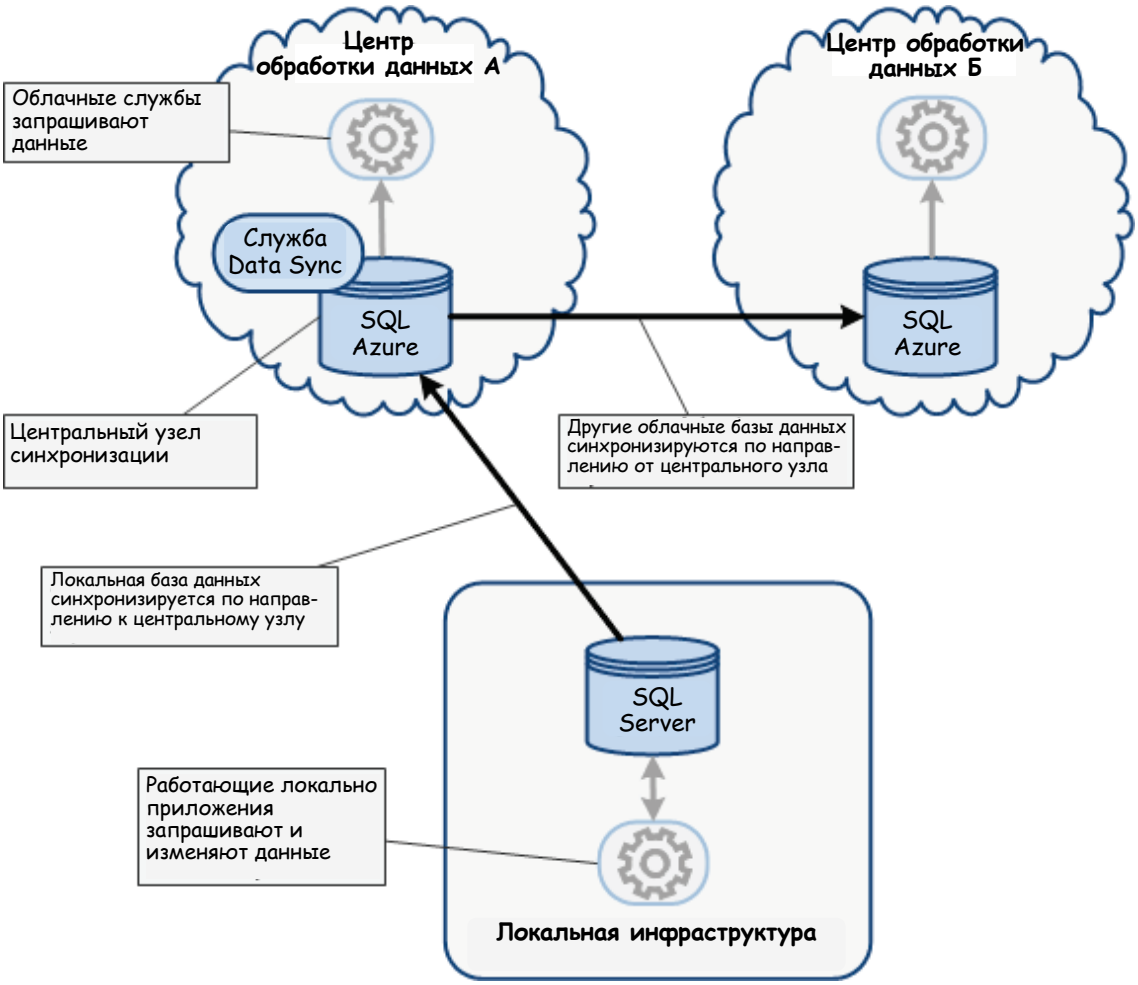



**Рисунок 9**

### Публикация локальной базы данных в облаке

- **У вас есть несколько приложений и баз данных SQL Server, работающих локально. Однако вы переместили большую часть функций бизнес-аналитики и отчетности в службы, работающие в облаке. Эти функции запускаются еженедельно, но в целях поддержки бизнес-операций им требуется отправлять запросы на доступ к бизнес-данным.**

В рамках данного сценария все изменения данных выполняются локальными приложениями в отношении группы баз данных SQL Server, размещенных также в организации. Эти приложения могут быть независимыми друг от друга, а также могут использовать совершенно разные базы данных. Тем не менее предположим, что функции бизнес-аналитики выполняют операции, которые охватывают все базы данных, запрашивая данные и формируя соответствующие отчеты, чтобы предоставить руководителю всю необходимую информацию для принятия обоснованного и эффективного бизнес-решения. Некоторые из этих отчетов требуют интенсивных вычислений, поэтому эти функции были перенесены в облако.

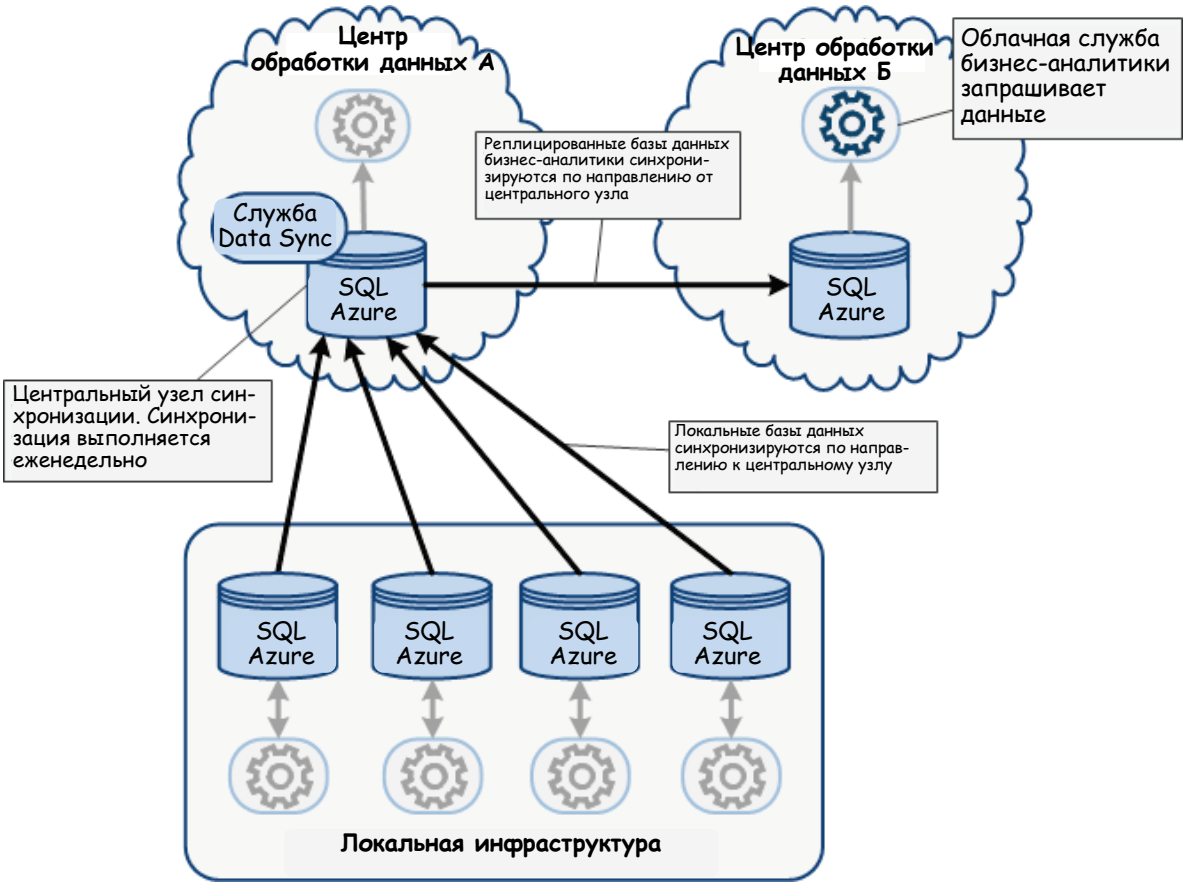



**Рисунок 10**

### Агрегация и консолидация данных в облаке

Вы можете использовать службу SQL Azure Data Sync для агрегации и консолидации данных из многочисленных локальных баз данных в единой облачной базе данных SQL Azure, копию которой можно разместить в различных центрах обработки данных, как показано на рисунке 10. Таким образом служба бизнес-аналитики в каждом центре обработки данных сможет запрашивать эту информацию в этом же облаке. При этом требуется организовать односторонний процесс синхронизации — из локальных баз данных в центральную, а затем из центральной базы данных в каждую базу данных SQL Azure. Никакую информацию не нужно отправлять обратно в локальную базу данных. Кроме того, синхронизацию можно выполнять еженедельно, за несколько часов до запуска службы бизнес-аналитики (точное расписание можно составить с учетом объема подлежащей репликации информации и продолжительности этого процесса).

Вы можете также использовать этот подход для агрегирования данных из нескольких офисов в облачной среде, с той лишь разницей, что локальные базы данных SQL Server хранятся в разных местах.

- **У вас есть несколько облачных служб, работающих в различных центрах обработки данных. Службы в каждом центре обработки данных управляют отдельным независимым набором данных вашей организации. Тем не менее каждая служба может запросить любые данные, управляемые службами как в том же центре обработки данных, так и в любом другом. Кроме того, приложениям, работающим локально, требуется доступ ко всем данным в вашей организации.**

В данном случае осуществляется секционирование данных между различными узлами, как в топологии B (локальный основной репозиторий с односторонней синхронизацией по направлению от облака), описанной ранее. База данных SQL Server, развернутая локально, хранит копию всех данных организации, но в каждом центре обработки данных развернута база данных SQL Azure, содержащая только определенный набор данных, который необходим службам, работающим в этом центре обработки данных. Такая топология позволяет службам, работающим в центре обработки данных, запрашивать и обновлять только собственное подмножество данных, а также периодически синхронизировать это подмножество с локальной базой данных.

Если службе требуется доступ к данным, хранящимся в другом месте, она может запросить необходимую информацию в локальной базе данных. Как было сказано ранее, этот механизм потребует реализации специализированной логики в каждой службе с целью определения местоположения данных, но если основная часть запросов выполняется в отношении облачной базы данных в том же центре обработки данных, то служба должна быстро реагировать и поддерживать высокую производительность.

Если вы используете службу SQL Azure Data Sync для реализации такой системы, необходимо создать отдельную группу синхронизации для каждой базы данных SQL Azure. Это обусловлено тем, что набор данных синхронизации для каждой базы данных SQL Azure будет отличаться, данные будут секционированы по центрам обработки данных. Локальная база данных будет входить во все группы синхронизации. В целях упрощения структуры можно настроить базу данных SQL Azure для каждого центра обработки данных таким образом, что она будет выступать в качестве центрального узла синхронизации. На рисунке 11 показана реализация данного решения.

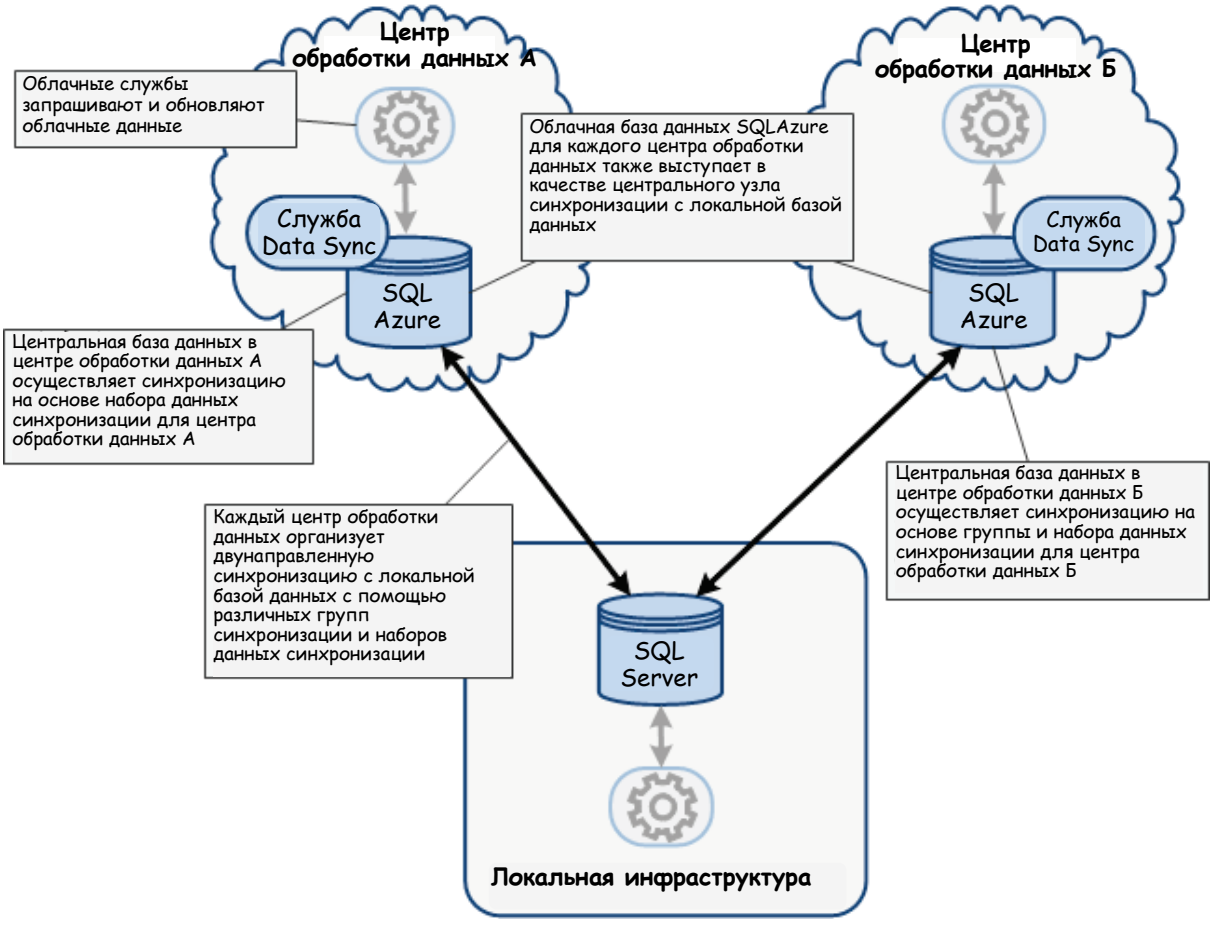


**Рисунок 11**

**Использование службы SQL Azure Data Sync для секционирования и репликации данных в облаке**

> **Мнение Маркуса**
>
> Будьте внимательны, если вы следуете этой стратегии, нельзя допускать возникновения циклов синхронизации. Убедитесь в том, что различные наборы данных синхронизации не совпадают и не включают в себя одни и те же данные.

- **Ваша организация состоит из головного офиса и нескольких удаленных филиалов. Приложениям, работающим в головном офисе, требуется доступ ко всем данным, управляемым каждым из филиалов, а также возможность периодически изменять эти данные. Каждый филиал может запрашивать любые данные, хранящиеся в этом и любом другом филиале (или в головном офисе), но изменять он может только те данные, которые связаны с этим филиалом.**

Это сценарий для топологии Д (локальные базы данных синхронизируются через облако). Данные могут храниться в базе данных SQL Server, и каждый филиал может развернуть у себя копию этой базы данных. Помимо центральной базы данных, никакие данные в облаке не сохраняются. Центральный узел рекомендуется размещать как можно ближе к самому активному филиалу (или центральному офису). Синхронизация должна быть двунаправленной, расписание необходимо составлять с учетом частоты обновлений, соответствующей потребностям других филиалов, которые должны видеть самые последние данные по любому из остальных филиалов. Если каждый филиал хранит только свой локальный набор данных, необходимо создать отдельную группу синхронизации для базы данных каждого филиала с соответствующим набором данных синхронизации, как описано в предыдущем сценарии. Если наборы данных синхронизации для каждого филиала не пересекаются, можно использовать ту же базу данных SQL Azure в качестве центрального узла синхронизации для каждой группы синхронизации. На рисунке 12 показана возможная структура такого решения.

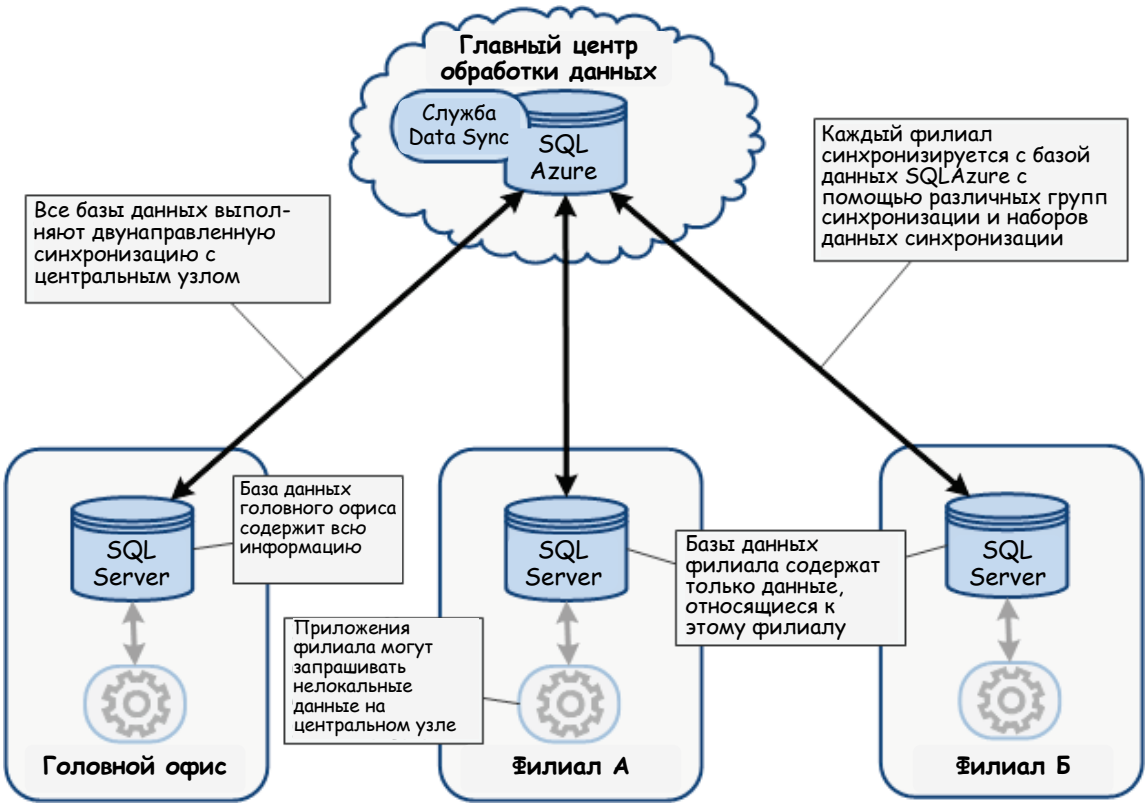



**Рисунок 12**

### Использование службы SQL Azure Data Sync для секционирования и репликации данных между филиалами

- Многие из ваших служб, работающих в облаке, выполняют большое количество операций по оперативной обработке транзакций (on-line transaction processing, OLTP), и скорость выполнения этих операций имеет решающее значение с точки зрения успеха вашего бизнеса. Чтобы обеспечить необходимую пропускную способность и свести к минимуму задержки в сети, информация, используемая этими службами, хранится в базе данных SQL Azure в том же центре обработки данных, в котором размещены сами службы. Другие службы на этом же узле формируют отчеты и анализируют информацию в этих базах данных. Некоторые функции отчетности выполняют очень сложные запросы. Однако вы обнаружили, что выполнение этих запросов приводит к возникновению конфликта в базе данных, что может серьезно повлиять на производительность служб OLTP.

В данном случае целесообразным решением представляется репликация базы данных, поддерживающей OLTP-операции, в другую базу данных, предназначенную для использования службами отчетов и аналитики — в рамках стратегии, подразумевающей горизонтальное масштабирование при чтении. Потребуется только односторонняя синхронизация (из базы данных OLTP в базу данных отчетов).Прогоны синхронизации можно запланировать на периоды наименьшей нагрузки на систему. База данных OLTP может исполнять роль центрального узла. Кроме того, база данных отчетов может быть оптимизирована в целях эффективного выполнения запросов. Таблицы могут включать индексы в целях ускорения различных операций, выполняемых службами аналитики для извлечения данных. Данные могут быть денормализованными, что ускоряет обработку сложных запросов. С другой стороны, количество индексов в базе данных OLTP рекомендуется свести к минимуму, чтобы избежать дополнительной нагрузки, связанной с их поддержкой в ходе выполнения операций с интенсивными обновлениями. На рисунке 13 представлено это решение.

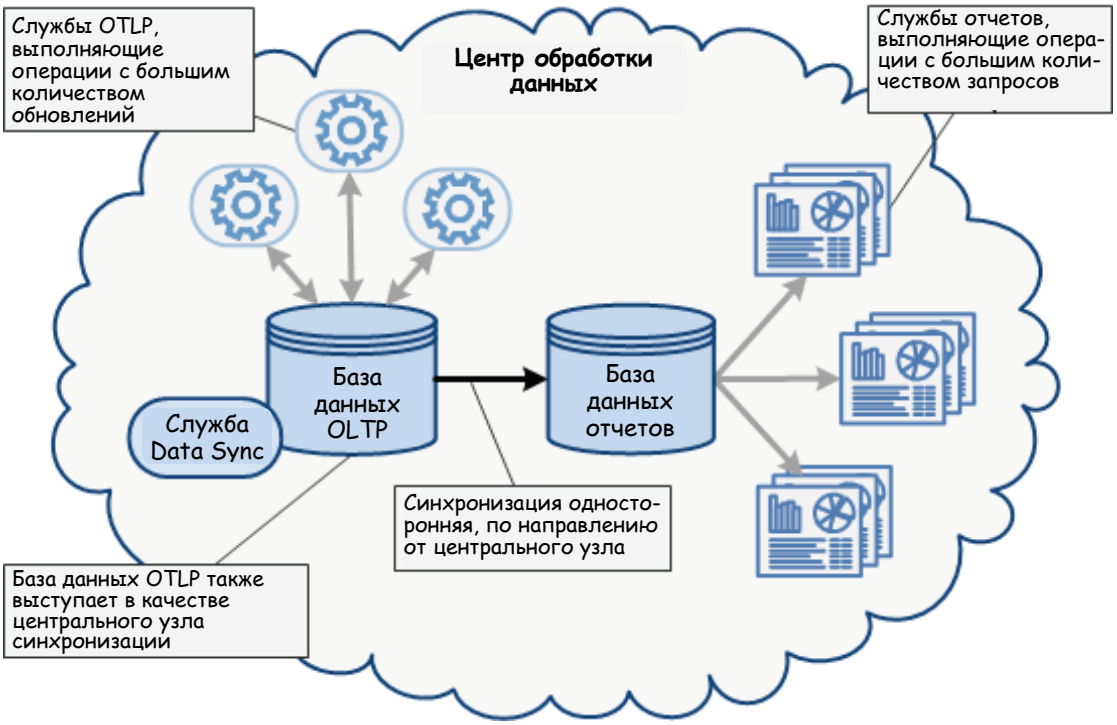



**Рисунок 13**

**Репликация базы данных с целью обеспечения возможности горизонтального масштабирования при чтении**

### Модель безопасности SQL Azure Data Sync

Служба SQL Azure Data Sync использует программное обеспечение под названием «агент клиента Data Sync» для организации взаимодействия между локальными экземплярами SQL Server и решением SQL Azure Data Sync Server в облаке. Вы должны загрузить и установить агент клиента Data Sync на одном из своих локальных серверов. Трафик между SQL Azure Data Sync Server и агентом клиента Data Sync зашифрован. Агент клиента Data Sync использует исходящее соединение HTTPS для взаимодействия с SQL Azure Data Sync Server. Кроме того, все важные сведения о конфигурации, используемые агентом клиента Data Sync и сервером SQL Azure Data Sync Server, также зашифрованы, включая учетные данные, необходимые для подключения к каждой локальной базе данных и базе данных SQL Azure. Ключ агента, созданный при помощи портала управления, используется службой Data Sync в целях аутентификации.

Программный инструмент «агент клиента Data Sync» состоит из двух компонентов:

- Служба Windows, которая подключается к локальным базам данных.
- Графическая утилита для настройки ключа агента и регистрации локальных баз данных этой службой.

Служба агента клиента должна быть настроена на запуск с использованием учетной записи Windows, которая имеет соответствующие права для подключения к каждому серверу, на котором размещены подлежащие синхронизации базы данных. Эти базы данных не обязательно должны быть расположены на том же сервере, что и служба агента клиента. При регистрации локальной базы данных с использованием ключа агента клиента необходимо предоставить регистрационные данные для доступа к решению SQL Server, на котором размещена эта база данных, и эта информация хранится (в зашифрованном виде) в файле конфигурации агента клиента. Когда сервер SQL Azure Data Sync Server

выполняет синхронизацию с локальной базой данных, агент клиента использует эти сведения для подключения к базе данных.

**Мнение По**

В целях обеспечения максимальной безопасности, если агент клиента работает не на том сервере, на котором развернута база данных, вы можете настроить службу агента клиента на шифрование соединения с каждой локальной базой данных с помощью SSL. Подразумевается, что вы установили соответствующие сертификаты SSL в каждом экземпляре SQL Server. Более подробная информация представлена на странице «Шифрование соединений с SQL Server»: http://msdn.microsoft.com/en-us/library/ms189067.aspx.

Для получения более подробной информации о модели безопасности SQL Azure Data Sync, см. «Безопасность данных» на странице http://msdn.microsoft.com/en-us/library/hh667329.aspx.

## Реализация пользовательского решения для репликации и синхронизации с помощью пакета Sync Framework SDK

Портал управления позволяет реплицировать и синхронизировать базы данных SQL Server и SQL Azure без написания дополнительного кода. Предоставляемые инструменты подходят для настройки большинства распространенных сценариев репликации. Однако могут возникать ситуации, требующие более полного контроля над процессом синхронизации, например службе может потребоваться запуск синхронизации в определенное время. Например, если вы кэшируете данные с помощью службы Windows Azure Caching, использование SQL Azure может привести к тому, что все данные кэша окажутся недостоверными после синхронизации. Возможно, вам придется более тщательно координировать жизненный цикл кэшируемых данных с учетом частоты синхронизации, например можно организовать очистку кэша в ходе синхронизации, чтобы снизить вероятность возникновения описываемой ситуации. Также может потребоваться иной механизм разрешения конфликтов, не поддерживаемый встроенными политиками SQL Azure Data Sync, или репликация данных из источника, отличного от SQL Server. Вы можете реализовать собственный подход к синхронизации в своих приложениях при помощи пакета Sync Framework SDK.

**Примечание**

В главе 6 «Максимизация масштабируемости, доступности и эффективности приложения Orders» описывается подход к кэшированию, реализованному в компании Trey Research. «Приложение Д. Максимизация масштабируемости, доступности и производительности» предоставляет более подробные рекомендации по использованию службы Windows Azure Caching.

Пакет Sync Framework 2.1 SDK позволяет создавать приложения с поддержкой синхронизации с SQL Azure. При помощи этой версии пакета Sync Framework SDK вы можете писать собственный код для синхронизации и напрямую контролировать процесс репликации.

Используя этот подход, вы сможете реализовать целый ряд сценариев, которые сложно воплотить при помощи SQL Azure Data Sync, например сможете создать приложения с поддержкой автономной работы или перемещаемые приложения. Примером может служить приложение, работающее на мобильных устройствах (например, ноутбуках) и используемое сантехниками или инженерами по эксплуатации здания. В начале каждого рабочего дня специалист использует приложение для подключения к местному филиалу и получения плана-графика работ со списком адресов и информацией о характере работ. После выполнения каждого задания, приложение, работающее на мобильном устройстве,

используется для ввода подробной информации, которая хранится в базе данных на этом мобильном устройстве. В промежутках между различными заданиями сотрудник может снова подключиться к сети филиала и загрузить подробную информацию о проделанной на данный момент работе, приложение также получит сведения о любых изменениях, внесенных в график работы на этот день. Например, специалиста могут попросить выполнить срочную работу, прежде чем приступать к следующему пункту своего плана-графика. Каждую пятницу во второй половине дня администратор филиала формирует детальный отчет о работе, выполненной всеми специалистами, подконтрольными данному филиалу.

Если база данных филиала реализована с помощью SQL Azure, мобильное приложение, работающее на мобильном устройстве, может использовать Sync Framework SDK для подключения к центру обработки данных, в котором размещена эта база данных, и выполнения синхронизации с локальной базой данных на устройстве. На рисунке 14 представлен упрощенный вариант этой архитектуры.

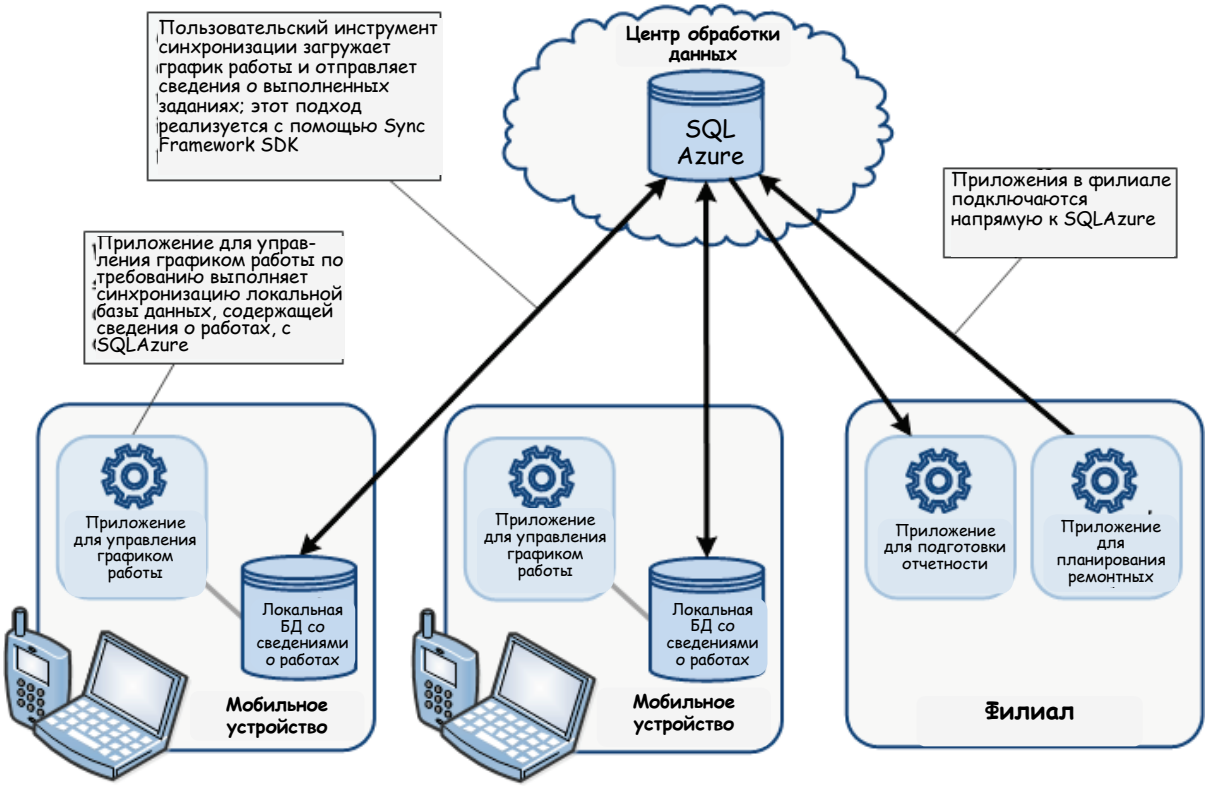



**Рисунок 14**

**Использование Data Sync SDK для реализации собственного подхода к синхронизации**

Более подробная информация об использовании Sync Framework SDK для организации взаимодействия с SQL Azure представлена на странице «Синхронизация SQL Server и SQL Azure при помощи Sync Framework 2.1»: http://blogs.msdn.com/b/sync/archive/2010/08/31/sql-server-to-sql-azure-synchronization-using-sync-framework-2-1.aspx.

## Репликация и синхронизация данных с помощью топиков и подписок шины интеграции

Служба SQL Azure Data Sync предоставляет оптимизированный механизм для синхронизации баз данных SQL Server и SQL Azure. Такое решение подходит для сред, в рамках которых изменения могут агрегироваться в пакеты и распространяться в виде групп, и все конфликты устраняются достаточно быстро. В динамичной среде такая пакетная обработка может не удовлетворять условиям, если необходимо осуществлять репликацию изменений по мере их возникновения, а не с какой-то

периодичностью. В такой ситуации вам, возможно, придется реализовать собственную стратегию. Однако эта проблема изучена достаточно хорошо, и вы можете использовать один из нескольких доступных шаблонов. В этом разделе описываются два общих сценария, которые подходят для большинства ситуаций, также вы узнаете, как можно реализовать решения для этих сценариев с помощью топиков и подписок шины интеграции.

### Рекомендации по использованию топиков и подписок шины интеграции

Вы можете использовать топики и подписки шины интеграции для реализации надежной инфраструктуры для маршрутизации сообщений между приложениями-отправителями и приложениями-получателями. Вы можете использовать эту инфраструктуру в качестве основы для создания настраиваемого механизма синхронизации обновленных данных в рамках набора распределенных источников данных, как описано в следующих сценариях:

- Экземплярам приложений или служб, работающих в рамках вашей системы, требуется доступ на чтение и запись к набору распределенных ресурсов. В целях поддержания минимального времени отклика, операции чтения должны выполняться быстро, и поэтому ресурсы реплицируются, чтобы уменьшить задержки в сети. Необходимость в операциях записи может возникать в любой момент, поэтому необходимо организовать контролируемый и согласованный доступ на запись к каждой копии, чтобы снизить вероятность утраты обновлений.

В качестве примера такого сценария можно привести распределенную систему, которая состоит из нескольких экземпляров приложения, получающего доступ к базе данных. Экземпляры работают в различных центрах обработки данных в облаке и, чтобы свести к минимуму задержки в сети, в каждом из них развернута копия базы данных. Если экземпляр приложения в центре обработки данных А обновляет элемент в базе данных этого центра, аналогичные изменения необходимо внести во всех копиях базы данных в других центрах обработки данных. В случае отсутствия строгого контроля над этим процессом, экземпляры приложения, запущенные в разных центрах обработки данных, могут присвоить разные значения копиям одних и тех же данных, в результате возникает конфликт, как показано на рисунке 15.

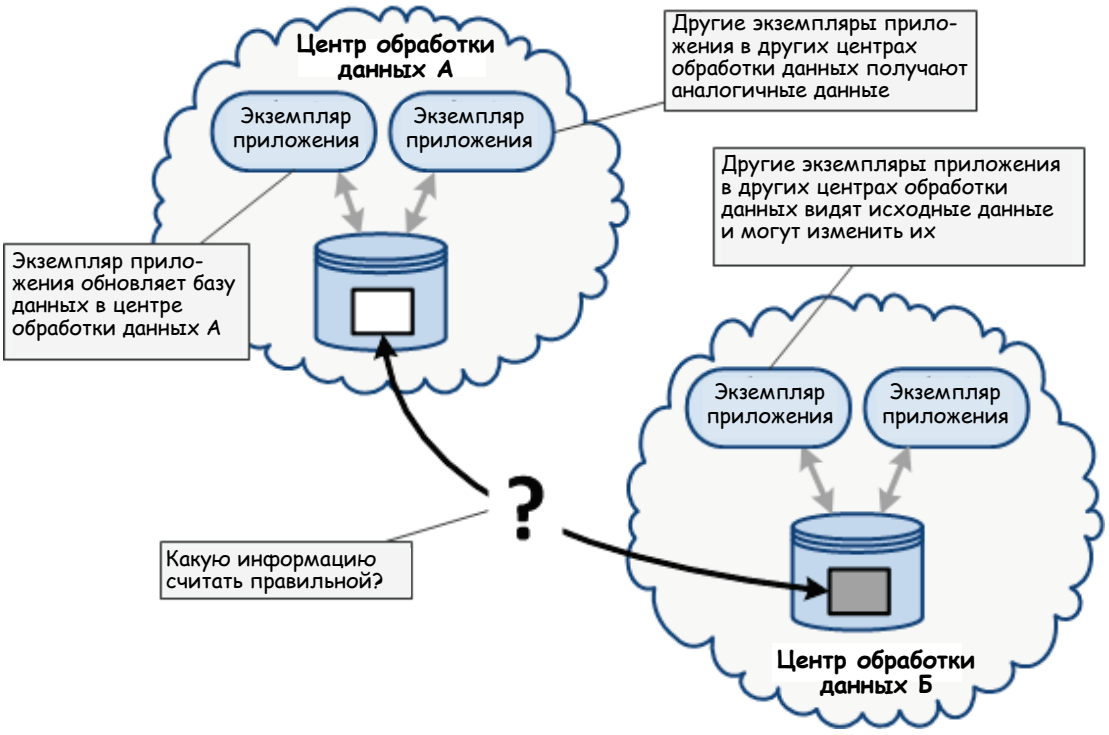


**Рисунок 15**

**Конфликт, который возникает без надлежащего контроля над обновлением копий базы данных**

В классической модели распределенных транзакций эту проблему можно решить путем внедрения диспетчеров транзакций и координации их друг с другом с помощью протокола Two-Phase Commit Protocol (2PC). Протокол 2PC гарантирует согласованность, однако он не способен обеспечить требуемую масштабируемость. В глобальной среде на основе сетей, которые не всегда обладают достаточной надежностью, это может привести к блокировке данных на чрезмерно длительный период, что увеличивает время отклика приложений, зависящих от этих данных. Поэтому вы должны быть готовы к некоторым компромиссам между согласованностью и доступностью.

Один из подходов к решению этой проблемы связан с реализацией операций обновления в виде транзакций BASE. BASE — это акроним от «Basic Availability, Soft-state, and Eventual consistency» (базовая доступность, наибольшая уязвимость и согласованность в конечном счете), эта концепция представляет собой альтернативу транзакциям ACID — «Atomic, Consistent, Isolated, and Durable» (атомарность, согласованность, изолированность и долговечность). Транзакции BASE не требуют полной и всеобщей согласованности в любой момент времени, достаточной считается согласованность базы данных «в конечном счете», если никакие изменения не будут потеряны. На практике (пример показан на рисунке 15) это означает, что экземпляр приложения, работающий в центре обработки данных А, может обновлять базу данных в том же центре, и эти обновления должны быть выполнены таким образом, чтобы обеспечивалась их репликация в центре обработки данных Б. Если экземпляр приложения, который работает в центре обработки данных Б, обновляет те же данные, изменения должны аналогичным образом передаваться в центр обработки данных А. Ключевым условием является то, что после того, как обновление было завершено, обе базы данных должны отражать самую актуальную информацию.

Может возникнуть ситуация, когда изменения, внесенные в центре обработки данных А, еще не были скопированы в центр обработки данных Б, и в течение этого периода времени экземпляр приложения, работающий в центре обработки данных Б, может содержать устаревшие данные, поэтому приложение следует разрабатывать с учетом этой возможности. Вернемся к примеру с приложением для обработки заказов и отображения информации об уровне запасов, о котором мы говорили ранее в данном приложении.

Топики и подписки шины интеграции — один из вариантов организации контролируемых обновлений в рамках данного сценария. Экземпляры приложений могут напрямую запрашивать данные в облачной базе данных, но все изменения должны оформляться в виде сообщений и отправляться в топик шины интеграции. Приложение-получатель, размещенное в центре обработки данных, создает собственную подписку на этот топик, при этом используется фильтр, который просто передает все сообщения в соответствии с параметрами этой подписки. Каждый получатель получает копию каждого сообщения, и использует эти сообщения, чтобы обновить облачную базу данных. На рисунке 16 показана базовая структура данного решения.

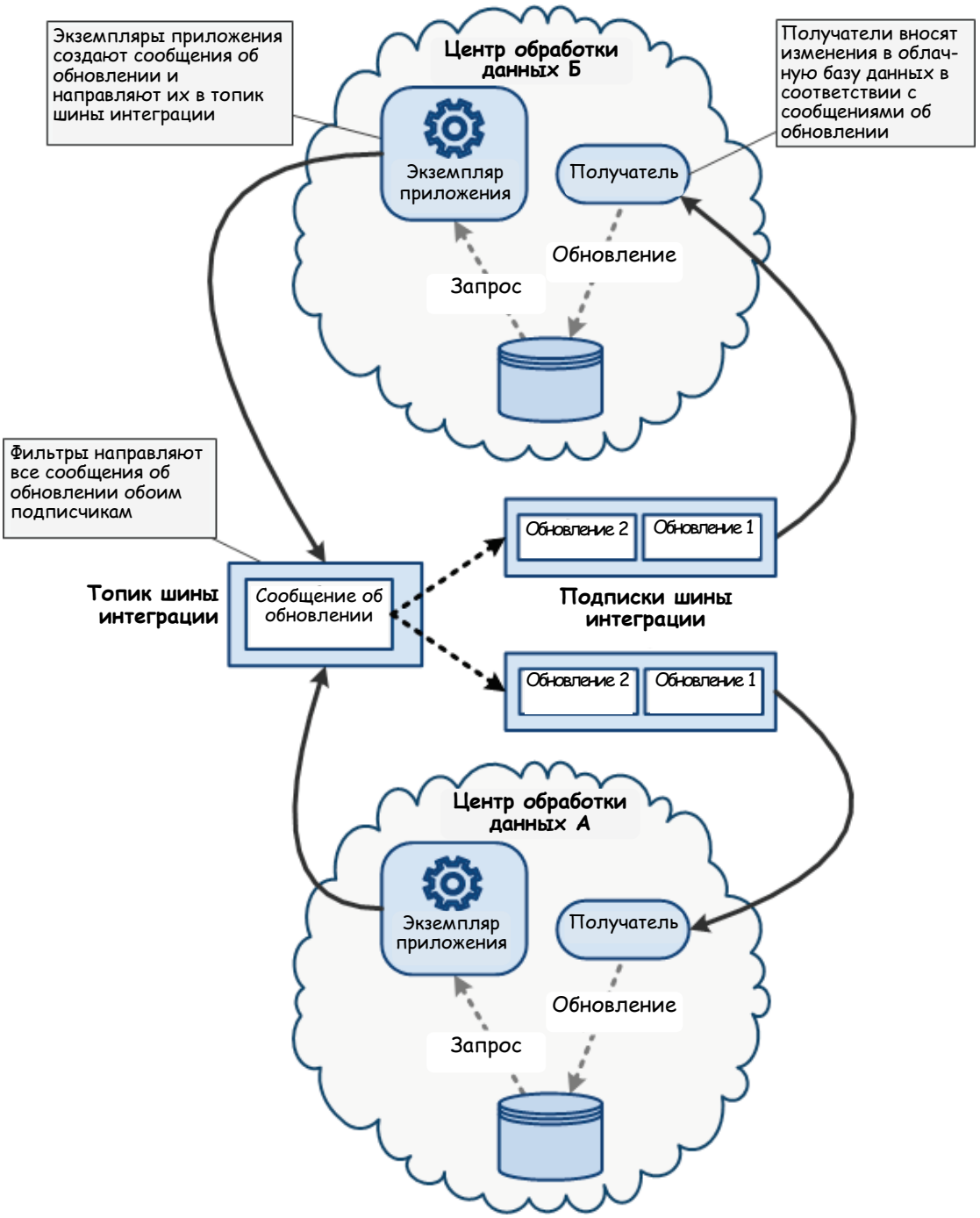



**Рисунок 16**

### Маршрутизация сообщений об обновлении при помощи топиков и подписок шины интеграции

Вы можете использовать функциональный блок для обработки неустойчивых неисправностей (Transient Fault Handling Block) из библиотеки Enterprise Library, чтобы создать надежную структуру для отправки сообщений в топик и обработки любых неустойчивых неисправностей. В качестве дополнительной меры по обеспечению надежности в настройках топика необходимо активировать функцию обнаружения дубликатов, чтобы повторно опубликованные сообщения об обновлении игнорировались.

Получатель сообщений должен активировать режим **PeekLock**. Если операция обновления завершена успешно, процесс приема также может быть завершен, в противном случае он будет отменен, а сообщение об обновлении снова появится в подписке. Получатель может запросить сообщение еще раз и повторить операцию.

- Экземпляры облачной службы, которые имеют длительный цикл выполнения, получают доступ к единому удаленному источнику данных, но в целях обеспечения минимального времени отклика каждый экземпляр поддерживает собственный кэш используемых им данных. Объем данных сравнительно мал, информация хранится в оперативной памяти. Данные, выступающие в качестве источника для кэша, могут быть изменены, и когда это случается, необходимо уведомить каждый экземпляр службы, чтобы он мог обновить копию данных в своем кэше.

Примером такого сценария является служба обработки заданий. Клиентские приложения хранят информацию о бизнес-задачах, которые должны быть выполнены, в базе данных. Служба обработки заданий периодически опрашивает базу данных, ищет новые задачи и выполняет соответствующие действия, когда задача появляется.

В данном случае каждый экземпляр службы обработки заданий при запуске наполняется информацией из базы данных. Когда задание добавляется клиентским приложением, оно сохраняется в базе данных. Однако клиентское приложение может также отправить сообщение в топик шины интеграции с подробностями нового задания. Каждый экземпляр службы обработки заданий оформляет подписку на этот топик и использует размещенные в ней сообщения для обновления своей локальной копии данных в кэше и выполнения соответствующих операций по обработке. Службе обработки заданий больше не нужно опрашивать базу данных, которая теперь выступает исключительно как журнал аудита заданий.

Такая архитектура позволяет оптимизировать использование базы данных, а также сводит к минимуму возможность возникновения конфликта. Когда базу данных должны опрашивать несколько экземпляров службы обработки заданий, существует вероятность того, что они могут получить информацию об одном и том же новом задании, если логика формирования запроса к базе данных не включает в себя механизм исключительной блокировки данных после их извлечения. С целью недопущения подобной ситуации, подписка шины интеграции автоматически предотвращает получение одного и того же сообщения несколькими экземплярами службы. Кроме того, эта архитектура позволяет равномерно распределить нагрузку и является масштабируемой. Вы можете разделить сообщения о новых заданиях на группы, чтобы направлять их разным подписчикам, определив соответствующий набор фильтров. Также вы можете запускать и останавливать экземпляры службы обработки заданий, связанные с каждой подпиской, ориентируясь на текущую длину очереди. На рисунке 17 показана структура данного решения.

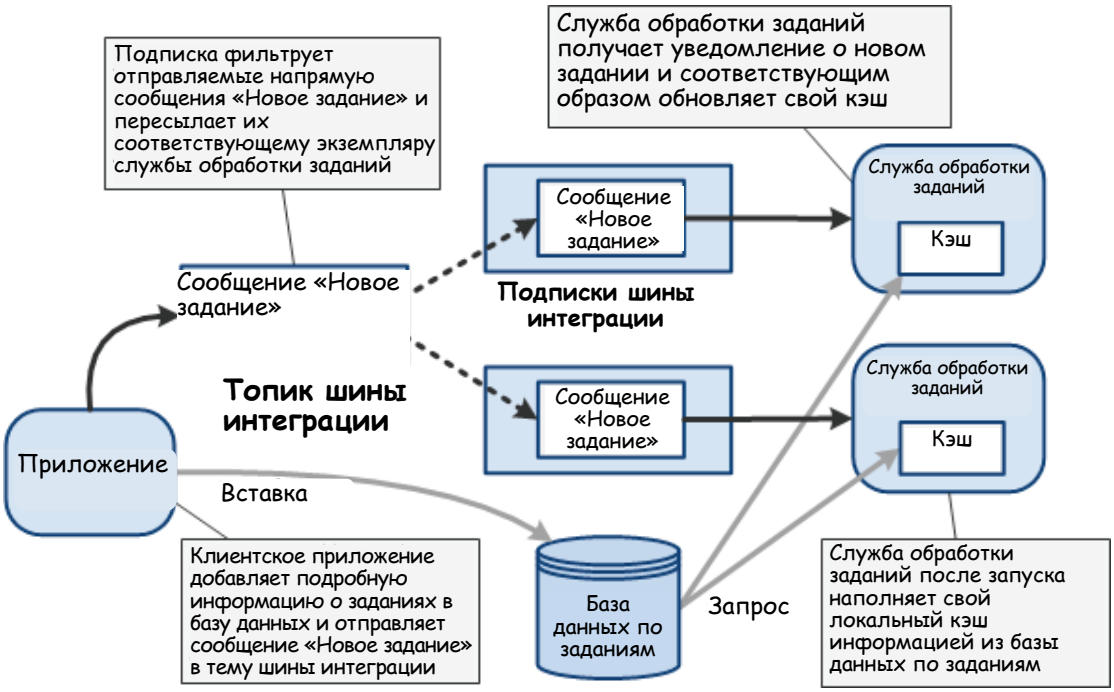



**Рисунок 17**

### Рассылка уведомлений об обновлении при помощи топиков и подписок шины интеграции

Логику клиентского приложения и службы обработки заданий можно легко расширить, если, например, клиентское приложение хочет отменить задание или изменить некоторые детали. В подобных ситуациях клиентское приложение может удалить или обновить информацию о задании в базе данных и направить сообщение об отмене или изменениях в топик шины интеграции.

# Приложение Б. Аутентификация пользователей и авторизация запросов

Большинству приложений придется выполнять аутентификацию и авторизацию пользователей или партнеров на каком-либо этапе процесса. Традиционно аутентификация проводилась на основе информации о пользователях из локального хранилища, но пользователи все больше надеются, что приложения будут предоставлять им возможность использовать универсальные учетные данные (например, существующие учетные записи в социальных сетях, таких как Windows Live ID, Google, Facebook и Open ID).

Данный процесс, называемый *федеративной аутентификацией*, также обеспечивает поддержку единого входа (single sign-on, SSO) в приложениях. Благодаря SSO пользователи входят в одно приложение, указывая, например, свои учетные данные Windows Live ID, затем они могут посещать другие сайты, которые используют Windows Live ID, без повторной авторизации.

Кроме того, приложениям могут потребоваться возможности для аутентификации пользователей на основе учетных записей, определенных в корпоративном домене, или для поддержки комбинированного использования федеративной аутентификации и корпоративных учетных данных (пользователи могут выбрать удобный вариант входа в систему). Кроме того, при использовании шины интеграции в технологической платформе Windows Azure, доступ к службе Service Bus Relay, очереди и конечным точкам топиков должен быть защищен при помощи маркера, содержащего соответствующие утверждения.

**Мнение Бхарата**

Федеративную аутентификацию на основе утверждений и единый вход обеспечивают высокоэффективные технологии, упрощающие разработку и предоставляющие преимущества для пользователей, которым не придется каждый раз вводить свои учетные данные для доступа в каждое приложение.

Все эти сценарии могут быть реализованы при помощи аутентификации на основе утверждений — служба маркеров безопасности (Security Token Service, STS) создает маркеры, которые сохраняются в файлах cookie в веб-браузере пользователя или представляются службами, когда они формируют запрос к серверу. В данном приложении описаны доступные технологии Windows, которые помогут вам реализовать аутентификацию на основе утверждений, федеративную аутентификацию и единый вход, а также защитить очереди шины интеграции.

**Примечание**

Данное приложение не является исчерпывающим руководством по технологиям и методам аутентификации на основе утверждений. Подробное описание аутентификации на основе

утверждений, технологий авторизации и службы Windows Azure Access Control содержится в руководстве «*Claims-Based Identity and Access Control Guide*»: http://claimsid.codeplex.com/.

## Сценарии использования и вызовы

Большинство бизнес-приложений требуют от пользователей аутентификации и авторизации, иначе они не смогут выполнять доступные в приложении операции. В следующих разделах описаны наиболее распространенные общие сценарии использования методов аутентификации и авторизации. Создаваемые вами решения должны быть надежными, гибкими, доступными и безопасными. Они должны однозначно и точно идентифицировать пользователей и предоставлять приложениям информацию, необходимую для определения действий, доступных для каждого конкретного пользователя. Решения должны максимально упрощать для пользователей вход в систему.

**Мнение По**

Недостаточно тщательно проработанные или неудачно реализованные функции аутентификации могут стать узким местом с точки зрения производительности приложения. Пользователи не хотят тратить много времени на аутентификацию, она не должна создавать препятствия для использования приложений.

### Аутентификация публичных пользователей

Общедоступные приложения, такие как интернет-магазины и форумы, как правило, должны аутентифицировать большое количество пользователей, о которых в приложении изначально нет никаких сведений. Пользователи проходят регистрацию на сайте, предоставляя требуемую информацию, например имя и адрес, но ключевой задачей является обеспечение возможности однозначной идентификации каждого пользователя, чтобы можно было предоставить ему правильную информацию при следующем посещении.

Приложения могут сохранить учетные данные пользователя, чтобы в дальнейшем сверять их с той информацией, которую он вводит. Однако, в случае с общедоступными приложениями, целесообразно предоставить пользователю возможность войти в систему при помощи учетных данных, которые он использует для других веб-сайтов и приложений, чтобы ему не пришлось запоминать еще одно имя пользователя и пароль. С помощью федеративной аутентификации приложение может делегировать выполнение этой задачи внешнему поставщику удостоверений, который хранит учетные данные и проверяет их, когда пользователь входит в систему. Такой подход также позволяет снять с вашего приложения ответственность за хранение конфиденциальной учетной информации, поскольку теперь за это отвечает поставщик удостоверений.

**Примечание**

Более подробная информация представлена в разделе «Федеративная аутентификация» далее в данном приложении.

### Аутентификация корпоративных пользователей и пользователей из организаций-партнеров

К приложениям, используемым лишь ограниченным набором известных пользователей, как правило, предъявляются иные требования, отличные от требований к общедоступным приложениям. Обычно пользователям не нужно регистрироваться, чтобы получить доступ к своей учетной записи, пользователи предполагают, что организация уже создала и настроила учетную запись для них. Кроме

того, такая учетная запись будет содержать более подробную информацию, по сравнению с данными в общедоступных приложениях или у поставщика удостоверений, при этом учетная запись не доступна для редактирования пользователем. Например, учетная запись, как правило, определяет принадлежность к группе безопасности и адрес корпоративной электронной почты пользователя.

Рассматриваемый подход обычно характерен для «внутренних» или корпоративных учетных записей большинства организаций, доступ к соответствующим функциям приложения получают при помощи встроенных в операционную систему инструментов и службы каталогов. Однако, если требуется предоставить возможность входа в систему для известных пользователей из организаций-партнеров, необходима служба-посредник, которая доступна за пределами организации и может создавать маркеры для аутентификации на основе утверждений. С этой целью также можно использовать методы федеративной аутентификации, рассматриваемые далее в данном приложении.

## Авторизация действий пользователя

Идентификация пользователей представляет собой лишь часть этого процесса. После аутентификации пользователей приложение должно контролировать их действия. Используемый для аутентификации на основе утверждений маркер, который пользователь получил от выбранного им поставщика удостоверений, может содержать утверждения, определяющие роль или разрешения для пользователей.

Однако это обычно актуально только в том случае, когда пользователь прошел аутентификацию у надежного поставщика удостоверений, например у корпоративной службы каталогов, и, следовательно, учетная запись пользователя содержит сведения о его роли (например, «Менеджер» или «Бухгалтер»). Поставщики удостоверений социальных сетей, такие как Windows Live ID и Google, как правило, учитывают только утверждение, представляющее собой уникальный идентификатор пользователя, и, возможно, его имя. Поэтому в данном случае вам придется реализовать механизм сопоставления идентификатора пользователя с данными, присутствующими в вашем приложении или находящимися в центральном репозитории пользовательских учетных данных.

Репозиторий должен содержать информацию о ролях, разрешениях или правах для каждого пользователя, эти данные приложение может использовать для авторизации действий пользователей. В случае с общедоступными приложениями, такими как интернет-магазин, хранилище, как правило, должно содержать информацию, предоставленную пользователями при регистрации (например, адрес и платежные реквизиты), что позволяет сопоставлять пользователя с его учетной записью, когда он входит в систему.

**Мнение Джаны**

Решения на основе утверждений и федеративной идентификации позволяют четко разделить задачи аутентификации и авторизации, поэтому изменения в одном решении не повлияют на другое. Такое разделение упрощает поддержку и обновление приложений в соответствии с новыми требованиями.

## Доступ к службе аутентификации для клиентов вне браузера

Методы аутентификации на основе утверждений отлично работают в веб-браузерах, поскольку они предоставляют средства автоматического перенаправления, и с точки зрения пользователя этот процесс полностью автоматизирован, все происходит абсолютно беспрепятственно. Для клиентов вне браузера, например других приложений, которые отправляют запросы на обслуживание вашим веб-службам, вы должны публиковать информацию, позволяющую клиенту узнать, как получить необходимый маркер и сведения о принятых форматах и утверждениях, которые маркер должен содержать.

Клиент отвечает за получение необходимого маркера у подходящего поставщика удостоверений и STS, он должен представить этот маркер по запросу. Когда приложение получит маркер, оно может использовать содержащиеся в нем утверждения для авторизации доступа. Если маркер содержит только уникальный идентификатор, приложению, как правило, приходится сопоставлять его с существующей учетной записью, чтобы получить информацию о ролях, применимых к данному клиенту.

### Авторизация доступа к очередям шины интеграции

Гибридные приложения, которые используют очереди шины интеграции Windows Azure, должны быть готовы к аутентификации пользователей, а также клиентских и партнерских приложений, запрашивающих доступ к очередям. Доступ к очереди шины интеграции предоставляется клиентам в соответствии с параметрами одного из трех режимов: отправка, прием или управление. С точки зрения безопасности очень важно определить разрешения, соответствующие клиентам, подключающимся к очереди, чтобы предотвратить несанкционированное чтение или отправку сообщений.

> **Мнение Маркуса**
>
> Топики и подписки шины интеграции работают так же, как и очереди, с точки зрения системы безопасности; аутентификация пользователей и авторизация доступа осуществляются аналогичным образом.

### Авторизация доступа к конечным точкам службы Service Bus Relay

Гибридные приложения, которые используют службу Windows Azure Service Bus Relay, должны быть готовы к аутентификации пользователей, а также клиентских и партнерских приложений, запрашивающих доступ к конечным точкам службы Service Bus Relay. Клиенты, которые запрашивают доступ к конечной точке этой службы, должны предоставить подходящий маркер, если эта конечная точка не поддерживает анонимный доступ (без аутентификации). Служба должна находиться внутри корпоративной сети или защищенного периметра, поэтому в целях безопасности очень важно организовать проверку наличия у клиентов соответствующих разрешений при попытке получения доступа к службе Service Bus Relay.

> Шина интеграции взаимодействует со службой Windows Azure Access Control, которая выступает в качестве системы идентификации по умолчанию для очередей шины обслуживания и конечных точек службы Service Bus Relay. Однако вы можете настроить шину интеграции на взаимодействие с другими поставщиками удостоверений.

## Взаимосвязанные проблемы

Механизмы аутентификации и авторизации должны, по определению, быть безопасными и надежными, защищая приложения от несанкционированного доступа и неразрешенных операций. Тем не менее другие требования также важны, и их необходимо учитывать в ходе реализации системы аутентификации и авторизации в гибридных приложениях Windows Azure.

### Безопасность

Личность пользователя или клиента необходимо устанавливать точно и однозначно со степенью уверенности, позволяющей исключить спуфинг и другие виды атак, которые могут нарушить точность результата проверки. При использовании федеративной аутентификации и делегировании

ответственности за проверку удостоверения пользователя, вы должны выбрать надежного поставщика удостоверений и службы-посредники.

Доступ к STS и поставщикам удостоверений необходимо организовать через защищенное соединение (протокол Secure Sockets Layer (SSL) или Transport Layer Security (TLS)), чтобы противостоять атакам методом перехвата сообщений и подмены ключей и предотвратить перехват учетных данных или маркеров аутентификации при отправке их по сети.

Многие STS могут зашифровать возвращаемые маркеры аутентификации, это должно рассматриваться в качестве дополнительного уровня безопасности, помимо защищенных соединений.

Если используется внутренний репозиторий с учетными данными пользователей, его необходимо защищать от внешнего несанкционированного доступа по сети и от внутренних атак, связанных с физическим доступом к серверам, нужно настроить соответствующие разрешения для таблиц и содержимого хранилища. Вся конфиденциальная информация должна быть зашифрована, чтобы ее было бесполезно похищать.

Вы должны учитывать юридические и договорные обязательства в отношении личной информации (PII) о пользователях, которая содержится в хранилище.

## Время отклика

Механизмы аутентификации могут стать узким местом в приложениях, которые должны авторизовать большое количество пользователей одновременно, например в начале рабочего дня. Ваша система должна быстро обрабатывать запросы даже в периоды пиковой нагрузки. Имейте в виду, что этот процесс может включать несколько переходов между различными сетями, объединяющими поставщиков удостоверений и STS, и любой из этих переходов может стать слабым звеном в цепи событий.

## Надежность

Механизмы аутентификации часто превращаются в единственную точку отказа. Достаточно просто добавить к приложению несколько новых серверов, чтобы обеспечить дополнительную пропускную способность приложений. С механизмами аутентификации ситуация сложнее, поскольку все они должны получать доступ к единому хранилищу или использовать внешнего поставщика удостоверений, над которым вы не имеете никакого контроля. В случае возникновения сбоя в работе системы аутентификации, пользователи не смогут работать с приложением.

## Совместимость

Протоколы и механизмы аутентификации на основе утверждений обеспечивают совместимость по умолчанию. Они используют основанные на стандартах протоколы и форматы для передачи и защиты маркеров. Некоторые STS и поставщики удостоверений могут предоставлять маркеры только одного типа, например Simple Web Token (SWT), в то время как другие системы могут принимать или возвращать маркеры других типов, например Security Assertions Markup Language (SAML). Необходимо убедиться, что выбранные вами поставщики и службы соответствуют требованиям и совместимы между собой.

# Технологии аутентификации и авторизации на основе утверждений

В данном разделе приводится краткий обзор технологий аутентификации и авторизации, которые обычно используются в приложениях для Windows и приложениях, размещаемых в Windows Azure. Основное внимание уделяется аутентификации на основе утверждений, поскольку такой подход представляется наиболее подходящим для гибридных приложений. Рассматриваются следующие темы:

- Федеративная аутентификация, службы маркеров безопасности и поставщики удостоверений.
- Платформа Windows Identity Foundation (WIF).
- Служба Windows Azure Access Control (ACS).

## Федеративная аутентификация

В традиционных приложениях аутентификация и авторизация, как правило, осуществляются при помощи функций операционной системы, которые используют службы корпоративных каталогов, такие как Microsoft Active Directory, а также при помощи инструментов для настройки разрешений на доступ к ресурсам, таких как списки контроля доступа (Access Control Lists, ACL). Современные платформы и технологии, например Microsoft ASP.NET, предоставляют встроенные механизмы для реализации аналогичного подхода в веб-приложениях.

Все перечисленные подходы требуют наличия каталога пользователей, который определяет, кто может получить доступ к приложению или службе, и какие операции разрешено выполнять. Задача ведения таких списков связана с многочисленными трудностями, часто могут возникать ошибки или бреши в системе безопасности. Например, если у вас хранится список пользователей партнерской организации, которые могут получать доступ к приложению, партнер должен уведомлять вас, когда какой-либо пользователь покидает компанию или вносятся изменения в разрешения для конкретных пользователей.

Вместо этого вы можете довериться партнеру и делегировать ему задачу ведения списка пользователей и их ролей. Вы по-прежнему сохраняете контроль над авторизацией и перечнем задач, которые пользователи могут выполнять в приложении, но вы освобождаетесь от необходимости аутентификации каждого пользователя и определения его принадлежности к соответствующим ролям или группам.

Подход, подразумевающий делегирование ответственности за аутентификацию, требует наличия стандартного механизма для формирования запросов к хранилищам информации о пользователях и распространения маркеров, которые содержат утверждения. Службы маркеров безопасности (Security Token Services, STS), такие как служба федерации Active Directory (Active Directory Federation Service, ADFS) и служба Windows Azure Access Control Service, предоставляют необходимые возможности. Кроме того, для других платформ и операционных систем доступны совместимые STS.

Каждая STS взаимодействует с поставщиками удостоверений (Identity Providers, IdPs), которые отвечают за аутентификацию пользователей. После того, как поставщик удостоверений выполнит аутентификацию пользователя, STS создаст для него маркер, содержащий утверждения. При использовании веб-браузера этот маркер сохраняется в виде файла cookie, а для смарт-клиентов и приложений служб маркер включается в ответ службы (в зависимости от используемого протокола). STS может также предоставить файлы cookie веб-браузеру, что означает, что пользователь прошел

аутентификацию и может получить доступ к другим приложениям без необходимости заново вводить учетные данные. Таким образом организован единый вход в систему (single sign-in, SSO).

**Мнение Бхарата**

ADFS выступает в роли IdP и STS одновременно, поскольку может использовать Active Directory для проверки пользователя на основе предоставляемых им учетных данных. ACS использует внешние IdPs, такие как Windows Live ID, Google, Facebook и OpenID, но вы также можете создать идентификаторы служб в ACS, позволяющие клиентам пройти аутентификацию без использования внешнего IdP.

Вы должны выбрать надежную службу STS для своего приложения. Это может быть ваша собственная служба STS, например ADFS, подключенная к вашему корпоративному каталогу Active Directory. Также можно использовать внешнюю службу STS, такую как ACS. Вы также можете настроить STS на взаимодействие с другой STS, и пользователи смогут проходить аутентификацию при помощи любой службы STS в цепочке отношений доверия, как показано на рисунке 1.

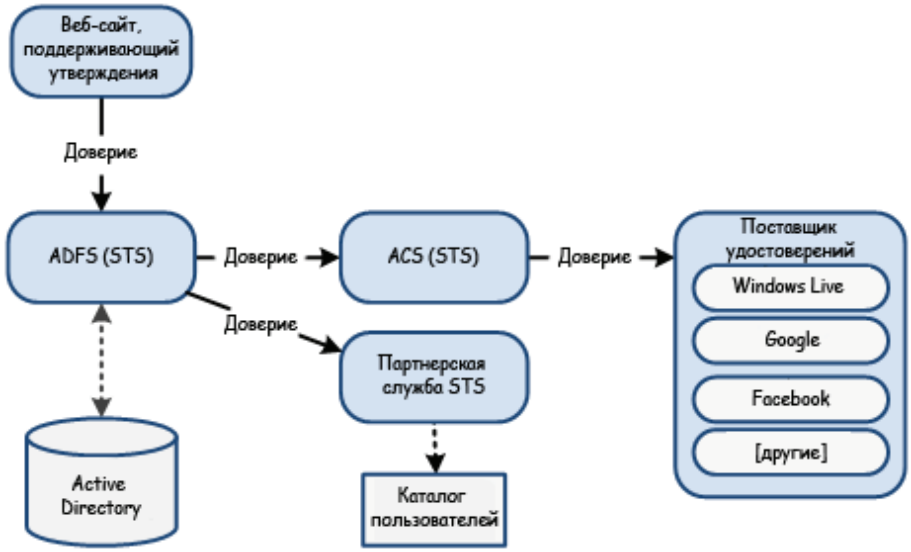



**Рисунок 1**

**Цепочка отношений доверия для аутентификации, которая может поддерживать федеративные удостоверения и единый вход**

### Обзор процесса аутентификации на основе утверждений

Служба STS выбирает IdP, который будет выполнять аутентификацию пользователя с учетом его домашней области или домена. Например, домашней областью пользователя будет Google, если этот пользователь имеет удостоверение, которое управляется и аутентифицируется Google. Если учетная запись пользователя принадлежит к корпоративному каталогу Active Directory, домен компании будет его домашней областью. Когда пользователь проходит аутентификацию через веб-браузер, STS направляет его к соответствующему IdP. Если пользователь может успешно пройти аутентификацию, IdP возвращает маркер с утверждениями (информацией) для этого пользователя в службу STS. Служба STS затем генерирует маркер для конкретного приложения, который может содержать эти утверждения или их расширенную версию, и перенаправляет пользователя в приложение вместе с этим маркером. Приложение может использовать утверждения в данном маркере для авторизации действий пользователя. На рисунке 2 показана упрощенная схема процесса, ACS используется в качестве поставщика маркера.

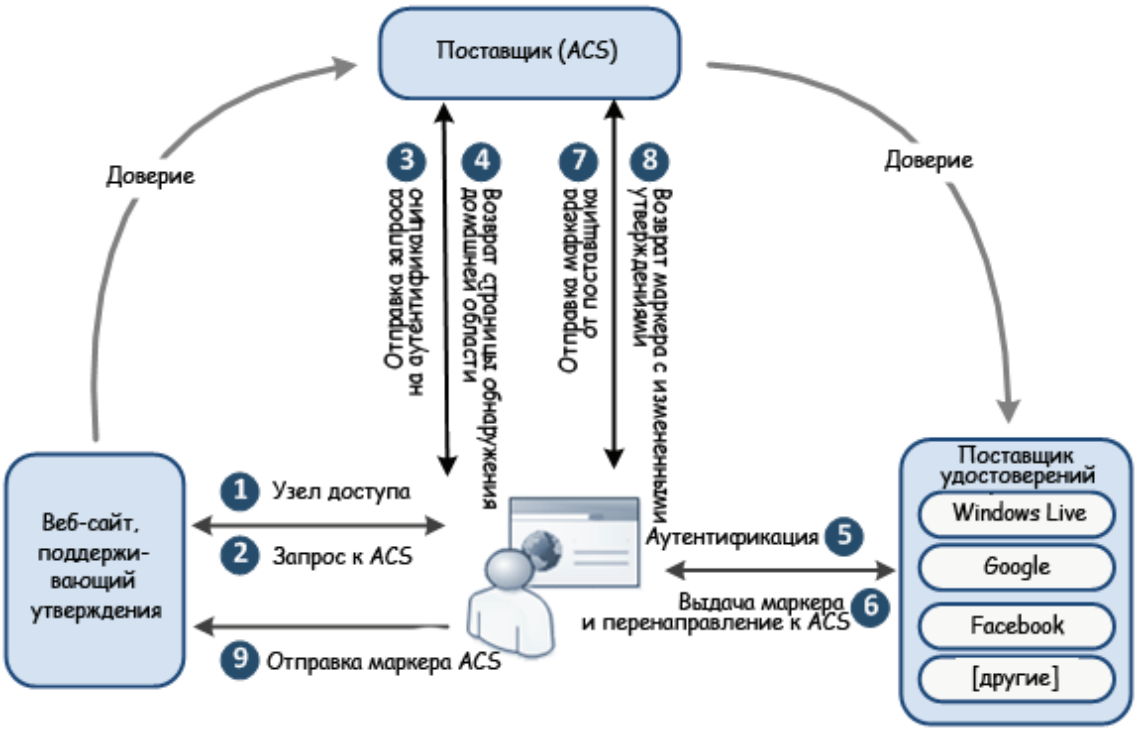



**Рисунок 2**

**Упрощенное представление процесса аутентификации через браузер с использованием ACS и поставщиков удостоверений социальных сетей**

В целях поддержки единого входа, STS может сохранить в рабочих каталогах браузера дополнительный файл cookie, содержащий маркеры для конкретной STS, чтобы подтвердить тот факт, что пользователь успешно прошел аутентификацию. Когда пользователь пытается получить доступ к другому приложению, которое доверяет той же службе STS, эта STS может создать подходящий маркер, не требуя от пользователя повторной аутентификации в системе IdP.

### Авторизация запросов веб-службы

Описанный выше процесс автоматического перенаправления применим только к задаче аутентификации запросов, поступающих из веб-браузера. Запросы к веб-службе могут быть созданы кодом, выполняемым в другом приложении, например в клиентском смарт-приложении или приложении-службе. В подобной среде, вне веб-браузера, функция перенаправления запросов и файлы cookie не могут использоваться для того, чтобы направить запрос по цепочке из нескольких STS и IdP.

При использовании аутентификации на основе утверждений в клиентском смарт-приложении или веб-службе, клиент должен запрашивать маркеры, необходимые на каждом этапе, и отправлять эти маркеры вместе с запросом к следующему звену в цепочке аутентификации. На рисунке 3 показана упрощенная схема этого процесса для смарт-клиента или веб-службы, ACS используется в качестве поставщика маркера.

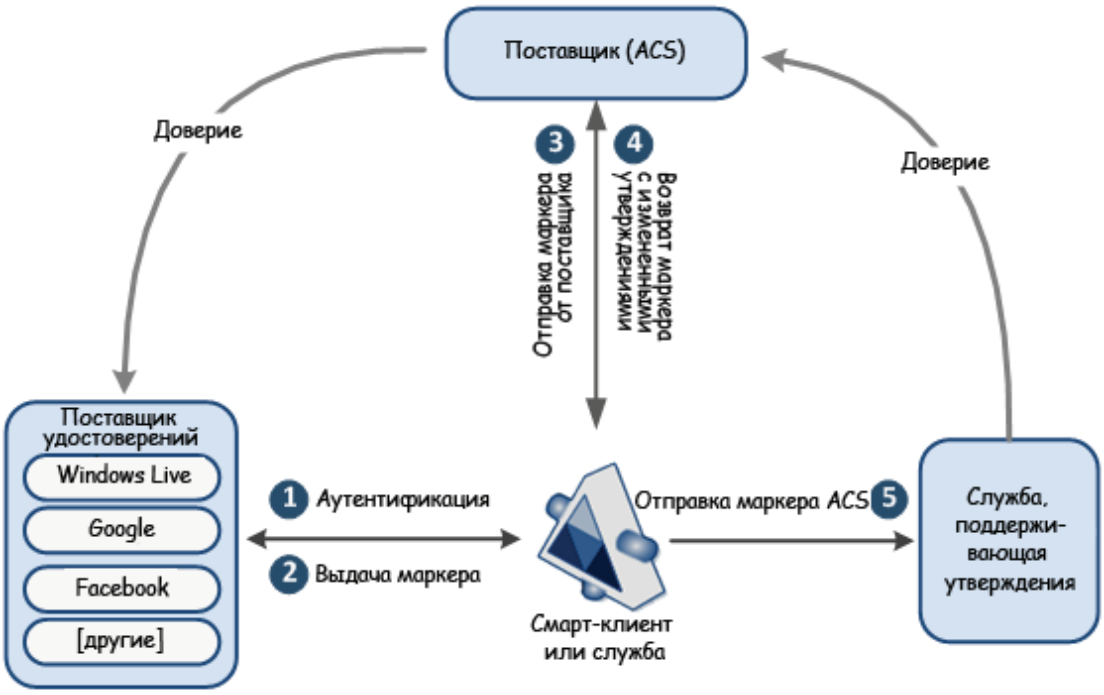



**Рисунок 3**

**Упрощенное представление процесса аутентификации для смарт-клиента или веб-службы с использованием ACS и поставщиков удостоверений социальных сетей**

Аутентификация на основе утверждений для веб-браузеров называется *пассивной*, потому что браузер автоматически выполняет перенаправление и предоставляет маркеры на соответствующих узлах цепочки событий, связанных с аутентификацией. Аутентификация вызовов служб, когда этот процесс управляется кодом приложения, называется *активной*. Более подробная информация об активной аутентификации представлена в статье «Claims Enabling Web Services» на сайте http://msdn.microsoft.com/en-us/library/hh446528.aspx.

## Платформа Windows Identity Foundation

Microsoft предоставляет специализированную платформу, которая позволяет легко реализовать аутентификацию и авторизацию на основе утверждений в веб-приложении и приложении-службе. Платформа Windows Identity Foundation (WIF) — базовый компонент службы управления идентификацией и доступом Microsoft на базе каталога Active Directory, служб федерации Active Directory, служб Windows Azure Access Control Services, а также технологии федеративной аутентификации на основе утверждений.

Платформа WIF автоматически проверяет запросы на наличие необходимых маркеров на основе утверждений и выполняет перенаправление в веб-браузерах к указанной STS, где они могут быть получены. Платформа также предоставляет утверждения в действующих маркерах коду приложения, который позволяет принимать решения об авторизации на основе этих утверждений. Платформа WIF включает мастер, который разработчики могут использовать в среде разработки Visual Studio или в командной строке, для настройки приложений на поддержку проверки подлинности на основе утверждений.

Компоненты WIF генерируют события в ходе обработки запросов, позволяя разработчикам контролировать этот процесс по мере необходимости. Платформа WIF также включает элементы управления, которые могут быть встроены в пользовательский интерфейс приложения с целью реализации функций входа в систему и выхода из нее.

**Примечание**

Дополнительная информация о платформе Windows Identity Foundation представлена на домашней странице об управлении удостоверениями http://msdn.microsoft.com/en-us/security/aa570351.aspx, а также в подготовленном группой Patterns & Practices руководстве «*Claims Based Identity & Access Control Guide*» (http://claimsid.codeplex.com/).

## Служба Windows Azure Access Control

Windows Azure Access Control Service (ACS) — это облачная служба, которая упрощает аутентификацию и авторизацию веб-сайтов, приложений и пользователей службы. Она совместима с большинством самых распространенных сред программирования и выполнения. Служба ACS интегрируется с инструментами и средами WIF и службами федерации Microsoft Active Directory (ADFS), а также поддерживает целый ряд протоколов, включая OAuth, OpenID, WS-Federation и WS-Trust. Аутентификация возможна с использованием множества популярных корпоративных и интернет-поставщиков удостоверений.

Когда пользователь запрашивает аутентификацию через веб-браузер, ACS получает запрос от веб-приложения и представляет страницу обнаружения домашней области, где перечислены поставщики удостоверений, которым веб-приложение доверяет. Пользователь выбирает поставщика удостоверений, и ACS перенаправляет его на соответствующую страницу входа. Пользователь входит в систему и возвращается к ACS с маркером, содержащим утверждения, которые пользователь предоставил этому поставщику удостоверений. Служба ACS затем применяет соответствующие правила для преобразования утверждений и создания на их основе нового маркера. Правила, настроенные в ACS, позволяют выполнять преобразование протоколов и утверждений в соответствии с требованиями веб-приложения. Пользователь перенаправляется в веб-приложение с маркером ACS. Веб-приложение может использовать содержащиеся в маркере утверждения, чтобы применить подходящие для данного пользователя правила.

ACS принимает маркеры в формате SAML 1.1, SAML 2.0 и SWT, и выдает в формате SAML 1.1, SAML 2.0 или SWT.

Процесс аутентификации запросов от смарт-клиентов и других приложений-служб отличается отсутствием взаимодействия с пользователем. Вместо этого служба должна получить подходящий маркер от поставщика удостоверений, потом предоставить этот маркер ACS для преобразования, а уже преобразованный — доверенному участнику.

Служба ACS также выступает в качестве поставщика удостоверений для шины интеграции Windows Azure. Вы можете настроить роли и разрешения для очередей шины интеграции и конечных точек службы Service Bus Relay при помощи ACS. Шина интеграции автоматически связывается с ACS и использует эту службу для проверки доступа и операций над очередями и конечными точками.

ACS настраивается через интерфейс службы при помощи API на основе OData или через веб-портал, который предоставляет графические интерактивные инструменты администрирования. Для настройки ACS вы также можете использовать командлеты интерфейса командной строки Windows Azure PowerShell. Эти командлеты предоставляют упаковщики для OData API и позволяют создавать сценарии, которые можно использовать для автоматизации многих задач для конфигурирования ACS.

**Примечание**

Более подробная информация о командлетах Windows Azure PowerShell представлена на странице «Windows Azure PowerShell Cmdlets» на сайте CodePlex http://wappowershell.codeplex.com/.

### ACS и уникальные идентификаторы пользователей

Если вы решили использовать ACS, во-первых, необходимо понять, что идентификатор пользователя, который возвращает эта служба после успешной его аутентификации, является уникальным не только для самого пользователя, но и для комбинации экземпляра ACS и пользователя. Разные настроенные пространства имен ACS возвращают разные идентификаторы для одного и того же пользователя. Если пользователь прошел аутентификацию с помощью Windows Live ID в одном экземпляре ACS, то после аутентификации с помощью Windows Live ID в другом экземпляре ACS он получит другой идентификатор. Это позволяет защитить конфиденциальность пользователей, различные приложения не смогут узнавать пользователей по их идентификаторам, когда они проходят аутентификацию в различных экземплярах ACS.

При развертывании нескольких экземпляров ACS в разных центрах обработки данных или изменении пространства имен вашего экземпляра ACS необходимо реализовать механизм сопоставления нескольких уникальных идентификаторов, предоставляемых службой ACS, с каждым пользователем в вашем хранилище пользовательских данных.

**Примечание**

Дополнительная информация о службе Windows Azure Access Control представлена в разделе «Access Control Service 2.0» на сайте http://msdn.microsoft.com/en-us/library/windowsazure/gg429786.aspx, подготовленное группой Patterns & Practices руководство «*Claims Based Identity & Access Control Guide*» представлено на сайте http://claimsid.codeplex.com/.

## Аутентификация и авторизация шины интеграции Windows Azure

Шина интеграции Windows Azure — это технология, которая позволяет предоставить доступ к внутренним службам через корпоративный брандмауэр или маршрутизатор, не открывая входящие порты и не выполняя сопоставление адресов. При этом создается виртуальная конечная точка в облаке, к которой подключаются пользователи, а запросы и сообщения передаются вашей службе или очереди сообщений. «Приложение B. Реализация подхода "коммуникации без границ"» посвящено более подробному рассмотрению шины интеграции. В данном разделе вы увидите, как интеграция с ACS используется для аутентификации запросов шины интеграции.

Шина интеграции взаимодействует с ACS с целью аутентификации запросов. Для каждого пространства имен шины интеграции, созданного при помощи портала управления Windows Azure, автоматически формируется соответствующее пространство имен ACS, которое используется для аутентификации запросов шины интеграции. Это пространство имен ACS представляет собой пространство имен шины интеграции с суффиксом «**-sb**». Домашней областью для него является пространство имен шины интеграции, создается маркер в формате SWT, который действителен в течение 1200 секунд. Вы не можете изменить эти настройки для выбранного по умолчанию внутреннего поставщика удостоверений, но можете добавить новых поставщиков, которые ведут себя по-другому. Вы также можете настроить правила и группы правил для доступа к вашим службам в этом пространстве имен ACS.

**Мнение Маркуса**

С целью просмотра и настройки параметров и правил для пространства имен службы ACS, которая аутентифицирует запросы шины интеграции, перейдите к пространству имен шины

интеграции на портале Azure и нажмите кнопку Access Control (Управление доступом) в верхней части окна. Вы также можете настроить пространство имен ACS программно с помощью интерфейса управления (Management API).

## Аутентификация клиентов

На рисунке 4 показан общий поток запросов на аутентификацию клиента и доступ к локальной службе через шину интеграции. Клиенты, которым нужен доступ к конечным точкам службы Service Bus Relay или очередям шины интеграции, должны получить маркер от ACS, который содержит утверждения, определяющие роли или разрешения для клиента (**1** и **2**), этот маркер предоставляется шине интеграции (**3**) вместе с запросом на обслуживание или доступом к очереди сообщений. Шина интеграции проверяет маркер и передает запрос целевой службе (**4**). Единственное исключение возможно, если специально настроить конечную точку службы Service Bus Relay на доступ без аутентификации. Организовать доступ без аутентификации к очередям шины интеграции невозможно.

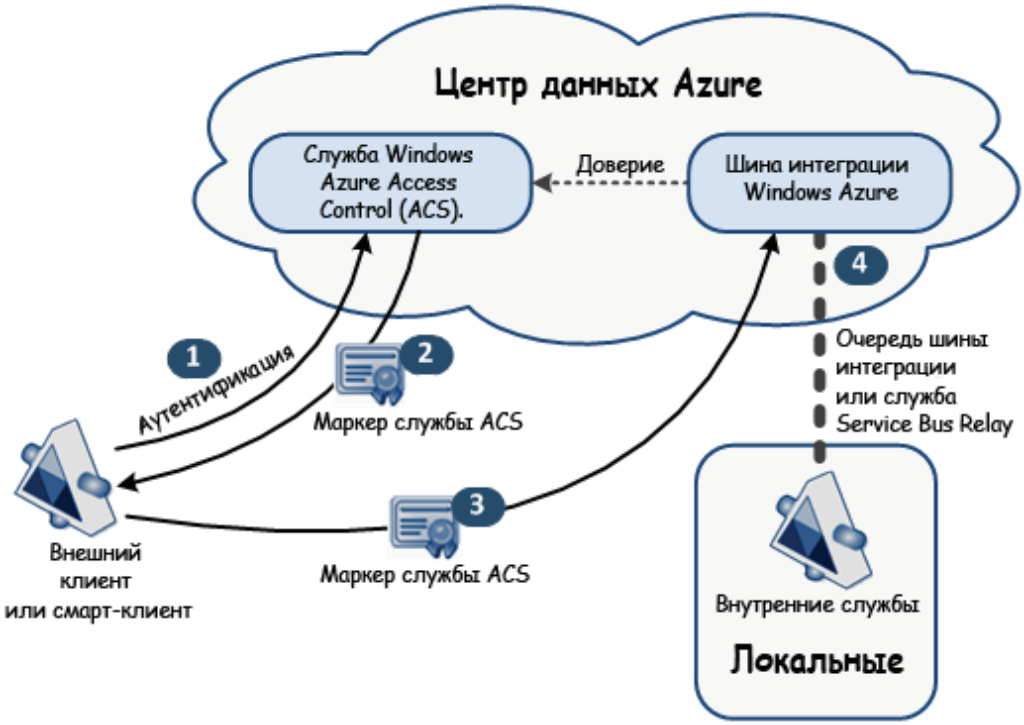



**Рисунок 4**

**Аутентификация запроса шины интеграции при помощи ACS**

Чтобы разрешить доступ без аутентификации к конечным точкам службы Service Bus Relay, установите для свойства **RelayClientAuthenticationType** в настройках ретранслятора элемента безопасности WCF значение **None**. Дополнительная информация представлена в документах MSDN для WCF: «Securing Services» на сайте http://msdn.microsoft.com/en-us/library/ms734769.aspx и «Securing and Authenticating a Service Bus Connection» на сайте http://msdn.microsoft.com/en-us/library/dd582773.aspx.

Выдаваемый службой ACS маркер может содержать несколько утверждений. Например, при использовании очереди шины интеграции маркер может содержать утверждения, позволяющие клиенту отправлять сообщения в очередь, принимать сообщения из очереди, а также управлять очередью. В случае со службой Service Bus Relay клиент может использовать утверждения с целью регистрации конечной точки для приема запросов, отправки запросов службе, а также управления

конечной точкой. Тем не менее сама служба (на рисунке 4 — локальная внутренняя служба) может затребовать дополнительные полномочия, не связанные с шиной интеграции. Маркер ACS просто позволяет получить доступ к службе через шину интеграции, остальные задачи, связанные с аутентификацией и защитой сообщений, решаются так же, как при вызове службы напрямую. Например, вы можете указать отдельные учетные данные в заголовке вызова службы, позволив клиенту получить доступ к определенным ресурсам сервера, на котором размещается служба.

**Мнение Джаны**

Процесс аутентификации с целью получения доступа к конечной точке службы Service Bus Relay полностью независим от любой другой задачи в сфере аутентификации и обеспечения безопасности, связанной с вызовом самой службы. Служба Service Bus Relay просто активирует механизм маршрутизации, позволяя получить доступ к внутренним службам в защищенном режиме. Методы, которые вы будете применять для защиты самой службы и имеющих к ней отношение ресурсов, ничем не отличаются от методов, используемых в процессе непосредственного подключения к службе по сети.

Служба ACS — это STS, которую можно настроить на взаимодействие с другими поставщиками удостоверений. Это полезно, когда вы хотите организовать аутентификацию клиентов в конкретной области, например в вашем собственном корпоративном домене, с целью управления доступом к внутренним службам через шину интеграции. Например, вы можете предоставить доступ к службе, при помощи которой внешние сотрудники и партнеры смогут отчитываться о своих расходах. У каждого действительного пользователя будет учетная запись в корпоративном каталоге.

На рисунке 5 представлен этот подход. Клиент сначала получает маркер от корпоративной службы ADFS и службы Active Directory (**1** и **2**), а затем предъявляет этот маркер службе ACS (**3**). Для службы ACS настроены доверительные отношения с экземпляром ADFS, который выступает в качестве поставщика удостоверений. Служба ACS проверяет предоставленный пользователем маркер средствами ADFS, и обменивает его на маркер ACS, содержащий утверждения, которые были преобразованы в соответствии с правилами ACS (**4**). Затем клиент может предоставить маркер ACS шине интеграции вместе запросом на обслуживание (**5**). Шина интеграции проверяет маркер и передает запрос целевой службе (**6**).

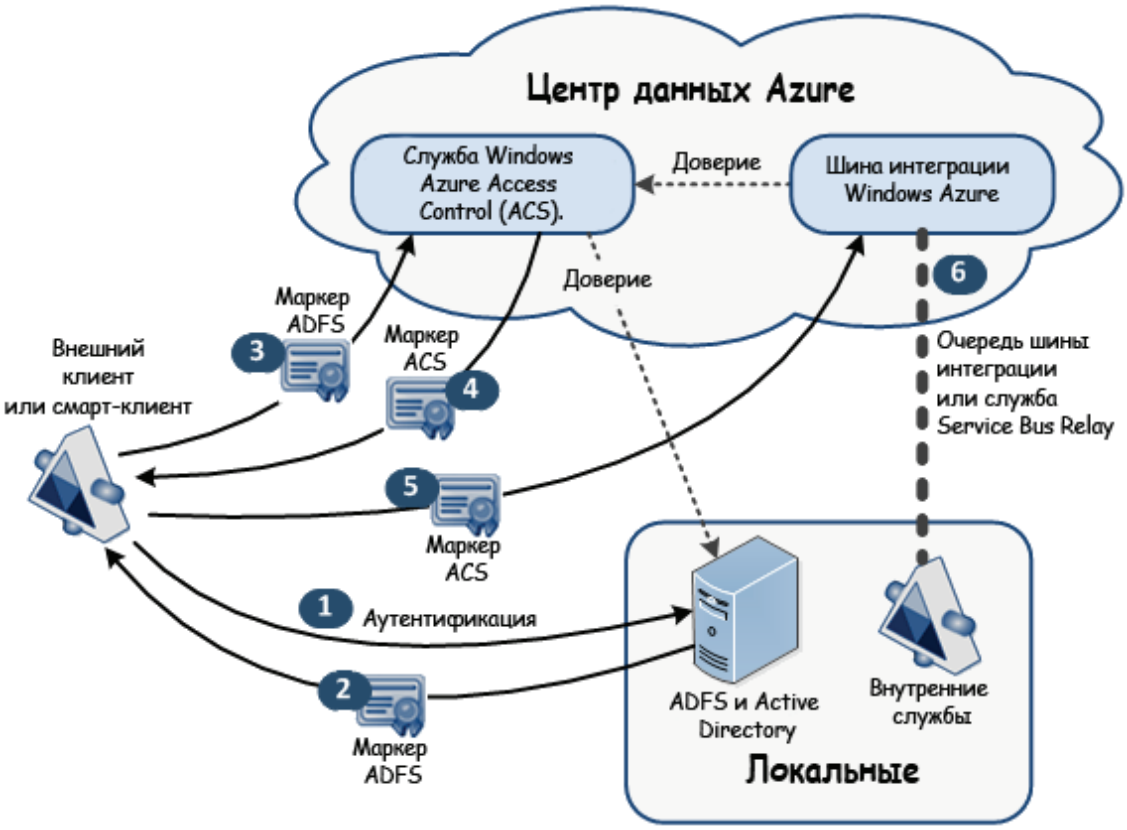



**Рисунок 5**

### Аутентификация запроса шины интеграции при помощи ADFS и ACS

Аналогичный подход можно реализовать с привлечением других поставщиков удостоверений, которые могут генерировать маркеры SAML. В пространство имен ACS, используемое шиной интеграции, также можно включить несколько поставщиков удостоверений, чтобы предоставить клиентам право выбора.

### Маркеры шины интеграции и поставщики маркеров

Если конечная точка не поддерживает анонимный доступ, клиенты, которые запрашивают доступ к конечной точке службы Service Bus Relay и очередям шины интеграции, должны предоставить службе ACS маркер. Поддерживаются следующие типы маркеров:

- **Shared secret**. В рамках данного подхода клиент предоставляет службе ACS маркер, содержащий удостоверение службы и сопоставленный ключ, чтобы получить от ACS маркер, который будет направлен шине интеграции вместе с запросом. Удостоверение службы настраивается в пространстве имен службы ACS, при этом генерируется подходящий ключ, который будет использоваться клиентом. Этот подход был показан на рисунке 4 ранее в данном приложении.
- **Simple Web Token** (SWT). При таком подходе клиент получает маркер SWT от поставщика удостоверений и предоставляет его службе ACS, чтобы получить от нее новый маркер и отправить его вместе с запросом шине интеграции. На рисунке 5 ранее в данном приложении показан общий процесс для рассматриваемого подхода.
- **Маркер SAML**. При таком подходе клиент получает маркер SAML от поставщика удостоверений и предоставляет его службе ACS, чтобы получить от нее новый маркер и отправить его вместе с запросом шине интеграции. На рисунке 5 ранее в данном приложении был также показан общий процесс для рассматриваемого подхода.

Чтобы включить требуемый маркер в запрос, клиент использует класс **MessagingFactory**, **NamespaceManager** или **TransportClientEndpointBehavior** шины интеграции. Все эти типы включают методы, которые принимают экземпляр конкретного класса, наследующий абстрактный базовый класс **TokenProvider**: **SamlTokenProvider**, **SharedSecretTokenProvider** или **SimpleWebTokenProvider**.

Конкретные реализации **TokenProvider** содержат методы, которые создают маркеры соответствующего типа на основе строковых значений и байтовых массивов, либо на основе существующих маркеров. Вы можете создать собственные реализации **TokenProvider** с целью применения других подходов к компоновке маркеров, например механизма федеративной аутентификации, если это необходимо.

> **Примечание**
>
> Более подробная информация представлена в статье «TokenProvider Class» на портале MSDN: http://msdn.microsoft.com/en-us/library/microsoft.servicebus.tokenprovider.aspx.

## Конечные точки шины интеграции и доверенные участники

Служба ACS автоматически создает доверенного участника по умолчанию в соответствующем пространстве имен ACS, когда вы настраиваете пространство имен шины интеграции. Значение области этого доверенного участника охватывает все адреса конечных точек, которые вы определяете в пространстве имен. Это корневой элемент пространства имен вашей шины интеграции.

Однако вы можете определить дополнительных доверенных участников в пространстве имен ACS, которые соответствуют адресам конечных точек, добавляемых в пространство имен шины интеграции. Например, если корневое пространство имен шины интеграции — **treyresearch**, вы можете определить дополнительные конечные точки следующим образом:

- http://treyresearch.servicebus.windows.net/orders/
- http://treyresearch.servicebus.windows.net/orders/ussouth/partners
- http://treyresearch.servicebus.windows.net/orders/usnorth/partners/partner1
- http://treyresearch.servicebus.windows.net/orders/usnorth/partners/partner2

Вы можете создать несколько определений для доверенных участников в ACS для любой подгруппы конечных точек шины интеграции. Служба ACS сопоставляет запрос с самым длинным из подходящих определений и применяет разрешения, указанные для этого определения. Это обеспечивает строгий контроль над разрешениями для каждой конечной точки или группы конечных точек. В приведенном выше списке, например, определение для **http://treyresearch.servicebus.windows.net/orders/ussouth/partners** будет использоваться для запросов к **http://treyresearch.servicebus.windows.net/orders/usnorth/partners/partner3**, поскольку более точные совпадения отсутствуют.

## Правила и группы правил авторизации

Пространство имен шины интеграции ACS содержит правила и группы правил, которые определяют роли и разрешения для клиентов по результатам аутентификации. Служба ACS автоматически создает правила по умолчанию с разрешениями на оправку, получение и управление для владельца службы, а также определение для корневого доверенного участника. Созданное по умолчанию удостоверение для **владельца** должно использоваться только для администрирования. Необходимо создать дополнительные удостоверения службы в ACS для клиентов и определить соответствующие разрешения (на отправку, получение и управление) в целях ограничения доступа минимально необходимым уровнем привилегий.

Для каждого определения доверенного участника необходимо задать правила, которые преобразуют утверждения в маркере, полученном службой ACS, и (или) правила, которые добавляют новые утверждения. При использовании аутентификации типа «shared secret» (общий секрет), исходный маркер отсутствует, поэтому входящие утверждения также не предусмотрены. Выходные утверждения имеют тип **net.windows.servicebus.action**, для них установлены значения «отправка», «получение» и «управление». Вы можете добавлять больше одного утверждения для любого правила. В целях упрощения настройки вы можете создать группы правил и применить их к нескольким доверенным участникам.

**Примечание**

Полный список руководств по использованию службы ACS, включая настройку удостоверений и поставщиков удостоверений, представлен в статье «ACS How To's» на сайте MSDN по адресу http://msdn.microsoft.com/en-us/library/gg185939.aspx. Пошаговые рекомендации по настройке правил и групп правил представлены в статье «How To Implement Token Transformation Logic Using Rules» на сайте MSDN по адресу http://msdn.microsoft.com/en-us/library/gg185955.aspx.

# Приложение B. Реализация подхода «коммуникации без границ»

Ключевым аспектом любого решения, которое охватывает локальную инфраструктуру организации и облачные среды, являются принципы взаимодействия элементов, составляющих это решение. Типичное распределенное приложение состоит из множества частей, работающих в различных местах, которые должны безопасно и надежно взаимодействовать друг с другом. Хотя отдельные компоненты распределенного решения обычно работают в контролируемых средах, в которых компании, предоставляющие свои ресурсы, обеспечивают надлежащее управление и защиту, сети, объединяющие эти элементы, широко используют такие инфраструктуры как Интернет, которые находятся вне зоны ответственности этих компаний.

Следовательно, сеть является слабым звеном многих гибридных систем: производительность нестабильна, наличие связи между компонентами не гарантируется, и все коммуникации необходимо надежно защищать. Любое распределенное решение должно успешно справляться с временными сбоями в процессе сетевого взаимодействия, обеспечивая при этом надлежащий уровень безопасности.

Технологическая платформа Windows Azure предоставляет методы и инструменты для решения поставленной задачи, помогая создавать надежные и защищенные приложения. В данном приложении рассматриваются соответствующие технологии.

**Мнение Джаны**

Очень важно выбрать наиболее оптимальный вариант взаимодействия компонентов друг с другом, поскольку это может оказать существенное влияние на структуру всей системы.

## Сценарии использования и проблемы

В рамках гибридных облачных решений различные приложения и службы будут работать локально или в облаке, взаимодействуя между собой по сети. Для того чтобы организовать взаимодействие между локальными и облачными компонентами, как правило, придется реализовать один или несколько из следующих общих сценариев использования. Каждый из этих сценариев использования создает ряд проблем, которые нужно учитывать.

### Доступ извне к локальным ресурсам организации

**Описание: доступ к локальным ресурсам необходим компонентам, работающим в других местах, например в облаке или партнерской организации.**

Основная проблема этого сценария связана с поиском и подключением к ресурсам, используемым приложениями и службами, которые работают за пределами организации. Выполняемые локально

алгоритмы получают прямой и контролируемый доступ к этим ресурсам в силу того, что они работают в том же сегменте сети. Однако, когда эти же алгоритмы работают в облаке, то они работают в другой сетевой среде, и им требуется надежное и безопасное подключение к локальным серверам, чтобы прочитать или изменить локальные ресурсы.

## Доступ извне к локальным службам организации

**Описание: доступ к локальным службам получают приложения, работающие в других местах, например в облаке или партнерской организации.**

В типичной архитектуре на базе служб, в рамках которой взаимодействие организовано через Интернет, приложения, работающие локально в организации, получают доступ к службам через общедоступные сети. Среда, в которой размещена служба, предоставляет доступ через один или несколько четко определенных портов с использованием стандартных протоколов, для большинства веб-служб это будет порт 80 для HTTP или порт 443 для HTTPS. Если у вас развернут брандмауэр, необходимо разрешить входящие запросы по этим портам. Когда ваше приложение, работающее локально, подключается к службе, оно формирует исходящий вызов через брандмауэр организации. Выбор локального порта для исходящего вызова со стороны локального приложения зависит от результатов согласования, выполняемого протоколом (вероятно, это будет некоторый порт с большим номером, который в настоящее время не используется), и любой ответ от службы возвращается через тот же канал на тот же порт. Очень важно, что для отправки запросов от приложения к службе вам не придется открывать дополнительные входящие порты в брандмауэре своей организации.

Когда служба работает локально, требования к обеспечению взаимодействия меняют свою полярность. Приложения, работающие в облаке или партнерской организации, должны сформировать входящий вызов через брандмауэр вашей организации и, возможно, один или несколько маршрутизаторов типа NAT (Network Address Translation), чтобы подключиться к вашим службам. Помните, задача этого брандмауэра — предотвратить несанкционированные и потенциально опасные действия в отношении хранящихся локально активов со стороны злоумышленников из внешней сети. Поэтому, исходя из соображений безопасности, большинство организаций применяют политики, которые ограничивают входящий трафик на локальных серверах, блокируя доступ к вашим службам. Даже если получено разрешение на открытие различных портов, необходимо позаботиться о фильтрации трафика с целью выявления и запрета доступа для вредоносных запросов.

В связи с этим сценарием возникает важная задача, связанная с обеспечением возможности доступа к службам, работающим локально, без ущерба для безопасности вашей организации.

> **Мнение По**
>
> Открытие портов в вашей корпоративной сети без учета последствий может сделать вашу систему уязвимой для атак. Многие хакеры используют автоматизированное сканирование портов для поиска таких возможностей. Затем они анализируют любые службы, прослушивающие открытые порты, с целью определения общих уязвимостей, которые можно использовать для взлома корпоративных систем.

## Создание надежного коммуникационного канала для интеграции различных сетей

**Описание: распределенным компонентам требуется надежный механизм связи, устойчивый к сбоям в сети и позволяющий компонентам поддерживать работоспособность, даже если сеть работает медленно.**

Если вы работаете в сети общего пользования, например в Интернете, вы полностью зависите от различных сетевых технологий, которые управляются сторонними операторами, которые передают ваши данные. Чтобы организовать надежный обмен сообщениями между различными элементами системы в такой среде, необходимо не только разбираться в логической семантике сообщений своего приложения, но и понимать особенности физических сетей и проблемы в сфере безопасности, связанные с этой семантикой.

В надежном канале связи не потеряются сообщения, однако информация может быть проигнорирована в управляемом режиме и при четко определенных обстоятельствах. Чтобы решить эту проблему необходимо учитывать следующее:

- Как организован коммуникационный канал? Какой компонент открывает канал: отправитель, получатель или какая-либо третья сторона?
- Как организована защита сообщений? Нужна ли дополнительная инфраструктура безопасности для шифрования сообщений и защиты канала связи?
- Находятся ли отправитель и получатель в функциональной зависимости друг от друга и от сообщений, которые они посылают?
- Должны ли отправитель и получатель работать синхронно? Если нет, то каким образом отправитель узнает, что сообщение получено?
- Канал связи дуплексный или симплексный? Если симплексный, то как получатель передает ответ отправителю?
- Время существования сообщений ограничено? Если сообщение не было получено в течение определенного периода, его необходимо игнорировать? Если да, нужно ли уведомлять в таком случае отправителя?
- Сообщение предназначено единственному получателю или может быть отправлено нескольким получателям?
- Важен ли порядок сообщений? Должны ли они быть получены в порядке, соответствующем очередности их отправки? Поддерживает ли канал связи установку приоритета для срочных сообщений?
- Существует ли зависимость между связанными сообщениями? Если одно сообщение получено, а зависимое от него нет, что происходит тогда?

## Взаимосвязанные проблемы

Параллельно с функциональными аспектами подключения компонентов к службам и данным необходимо также рассмотреть общие нефункциональные задачи, характерные для любого механизма коммуникации.

### Безопасность

Первая и самая важная из этих задач связана с обеспечением безопасности. К сетям следует относиться как к враждебному окружению, тщательно проверяя весь входящий трафик. В частности, необходимо убедиться, что канал связи, используемый для подключения к службе, надежно защищен. Запросы могут поступать от служб и организаций, работающих в домене безопасности, отличном от домена вашей организации. Вы должны быть готовы проверить подлинность всех входящих запросов

и авторизовать их в соответствии с принятой в вашей организации политикой доступа к данным, это необходимо для защиты ресурсов вашей организации от несанкционированного доступа.

Вы также должны принять необходимые меры для защиты всего исходящего трафика, поскольку данные, которые вы передаете, станут уязвимыми сразу после того, как покинут среду вашей организации.

Вопросы, на которые необходимо ответить в процессе организации защищенного коммуникационного канала:

- Каким образом вы собираетесь создать канал связи, который безопасно проходит через корпоративный брандмауэр?
- Как вы будете осуществлять аутентификацию и авторизацию отправителя, чтобы предоставить ему возможность передавать сообщения по коммуникационному каналу? Как вы будете осуществлять аутентификацию и авторизацию получателя?
- Как предотвратить несанкционированный перехват сообщений, предназначенных для другого получателя?
- Как вы будете защищать содержимое сообщений, чтобы предотвратить несанкционированный доступ на чтение или изменение конфиденциальных данных?
- Как вы будете защищать отправителя или получателя от сетевых атак?

**Мнение По**

Надежные инструменты защиты — важный компонент любого приложения, поддерживающего сетевой доступ. Цена нарушения безопасности может быть очень высокой, кроме того, пользователи будут все меньше доверять вашей системе.

## Время отклика

Досконально продуманное решение гарантированно обеспечивает работоспособность системы, даже если сообщения между удаленными компонентами передаются по сети через медленные каналы с высокой вероятностью возникновения ошибок. Отправители и получатели, скорее всего, будут работать на разных компьютерах, размещенных в разных центрах обработки данных (в облаке, локально или у организации-партнера) в различных географических точках. Вам придется ответить на следующие вопросы:

- Как убедиться, что отправитель и получатель могут надежно общаться между собой, не блокируя друг друга?
- Каким образом канал связи будет справляться с внезапным наплывом сообщений?
- Зависит ли отправитель от конкретного получателя и наоборот?

## Совместимость

Гибридные приложения включают компоненты, созданные с использованием различных технологий. В идеале, канал связи, который вы создаете, не должен зависеть от этих технологий. Следуя этой стратегии, вы не только снижаете зависимость от особенностей реализации существующих элементов вашего решения, но и гарантируете возможность расширения своей системы в будущем.

В целях обеспечения совместимости сообщений необходимо реализовать основанный на стандартах подход, используя общепринятые сетевые протоколы, такие как TCP и HTTP, а также такие форматы сообщений, как XML и SOAP. Общая стратегия решения этой проблемы заключается в выборе механизма связи, который размещается в отдельном слое поверх стандартного протокола, а затем, при помощи соответствующих библиотек, сообщения приводятся к необходимому формату. В результате этого компоненты, созданные с использованием различных технологий, могут легко анализировать и обрабатывать эти сообщения.

## Технологии Windows Azure, которые помогают реализовать подход «коммуникации без границ»

Если вы создаете решения на основе прямого доступа к локальным ресурсам, вы можете использовать Windows Azure Connect для создания безопасного виртуального сетевого подключения к локальным серверам. Ваши алгоритмы смогут использовать это соединение для чтения и записи ресурсов, к которым у них есть доступ.

> Вы также можете развернуть данные в облаке, организовав их размещение в хранилище BLOB-объектов Windows Azure и табличном хранилище. Пакет Windows Azure SDK предоставляет интерфейсы API, которые позволяют получить доступ к этим данным из локальных приложений, а также из других служб, работающих в облаке или в среде партнерских организаций. Детальное описание этих сценариев представлено в подготовленном группой Microsoft Patterns & Practices руководстве «*Developing Applications for the Cloud on the Microsoft Windows Azure Platform*» (http://wag.codeplex.com). В настоящем руководстве эти сценарии не рассматриваются.

Если вы работаете с сервис-ориентированной архитектурой (SOA), можно создать службы, которые будут больше ориентироваться на функциональные возможности доступа к ресурсам. Вы будете отправлять сообщения службам, получающим контролируемый доступ к ресурсам от вашего имени. Взаимодействие со службами в рамках такой архитектуры часто соответствует одному из двух различных стилей:

- **Взаимодействие при помощи удаленного вызова процедур (Remote procedure call, RPC).**

  В рамках данного стиля получателем сообщения, как правило, выступает служба, предоставляющая набор операций, которые может инициировать отправитель. Эти операции могут принять параметры и вернуть результаты, и в некоторых простых случаях могут рассматриваться как расширение локального метода вызова за исключением того, что метод теперь работает удаленно. Базовый механизм связи, как правило, скрыт прокси-объектом в приложении-отправителе. Прокси-объект подключается к приемнику, передает сообщение, ждет ответа, а затем возвращает этот ответ отправителю. Эту модель обычно используют веб-службы.

  Такой стиль обмена сообщениями лучше всего подходит для синхронных коммуникаций, которые могут повлиять на время отклика со стороны отправителя, поскольку приходится ждать, пока сообщение будет получено и обработано, прежде чем будет отправлен ответ. В вариациях этого стиля поддерживается асинхронный обмен сообщениями, когда отправитель формирует обратный вызов для обработки ответа от получателя, также поддерживается односторонний обмен сообщениями, когда ответ не нужен.

  Вы можете создавать компоненты в виде служб Windows Communication Foundation, которые позволяют реализовать RPC-стиль обмена сообщениями. Если службы должны работать

локально, можно обеспечить безопасный доступ к ним с помощью службы Windows Azure Service Bus Relay; этот механизм описан далее в настоящем приложении.

Если эти службы должны работать в облаке, вы можете разместить их в качестве ролей Windows Azure. Этот сценарий описан в подготовленном группой Microsoft Patterns & Practices руководстве «Developing Applications for the Cloud on the Microsoft Windows Azure Platform».

> **Мнение Маркуса**
>
> Устаревшие приложения и фреймворки также поддерживают представление *удаленных объектов*. В рамках данного стиля распределенных коммуникаций хост-приложение службы предоставляет доступ к коллекции объектов, а не операциям. Клиентское приложение может создавать и использовать удаленные объекты, семантика которых аналогична таковой у локальных объектов. Несмотря на то, что этот механизм является очень привлекательным с объектно-ориентированной точки зрения, он имеет серьезный недостаток с точки зрения использования сети (клиентские приложения отправляют множество коротких сетевых запросов), в результате может серьезно пострадать производительность. Этот стиль коммуникаций не рассматривается в данном руководстве.

- **Взаимодействие на основе сообщений.**

В рамках данного стиля получатель ожидает получить упакованное сообщение, а не запрос на выполнение операции. Сообщение предоставляет контекст и данные, а получатель анализирует сообщение, чтобы определить, как его обрабатывать и какие задачи выполнять.

Этот стиль обмена сообщениями, как правило, основан на очередях. Отправитель создает сообщение и помещает его в очередь, из которой их принимает получатель. По своей природе этот метод взаимодействия является асинхронным, очередь выступает в качестве хранилища сообщений, которые получатель может выбирать и обрабатывать в определенное время. Если отправитель ожидает ответа, он может в той или иной форме пометить исходное сообщение, а затем ждать сообщение с тем же идентификатором, которое должно появиться в той же или другой очереди, по согласованию с получателем.

Windows Azure предоставляет надежный механизм — очереди шины интеграции. Они рассматриваются далее в настоящем приложении.

В следующем разделе подробно рассматривается служба Windows Azure Connect, служба Windows Azure Service Bus Relay и очереди шины интеграции; вы также получите рекомендации по применению перечисленных технологий.

## Доступ извне к локальным ресурсам организации при помощи службы Windows Azure Connect

Служба Windows Azure Connect позволяет интегрировать роли Windows Azure с вашими локальными серверами путем создания виртуального сетевого соединения между двумя средами. Служба создает сетевое подключение на основе стандартных IP-протоколов между вашими приложениями и службами, работающими в облаке, и вашими локальными ресурсами и наоборот.

### Рекомендации по применению службы Windows Azure Connect

Использование службы Windows Azure Connect обеспечивает значительные преимущества по сравнению со стандартными альтернативными подходами:

- Настройка значительно упрощается, и при этом нет необходимости вносить какие-либо изменения в свою локальную сеть. Например, ИТ-персоналу не придется настраивать VPN-устройства, какую-либо сложную сеть или DNS, также не нужно изменять конфигурацию брандмауэра на каком-либо из ваших локальных серверов или изменять настройки маршрутизатора.

> **Мнение Бхарата**
>
> Если вы предоставляете доступ к роли виртуальной машины, возможно, вам придется настроить брандмауэр в размещенной в этой роли виртуальной машине. Например, брандмауэр Windows может заблокировать общий доступ к файлам.

- Вы можете выборочно перемещать приложения и службы в облако, защищая существующие инвестиции в локальную инфраструктуру.
- Вы можете оставить ключевые службы для работы с данными и ресурсы в локальной среде, если не хотите перемещать их в облако, а также если это необходимо в целях обеспечения соответствия требованиям или вследствие юридических причин.

Служба Windows Azure Connect подходит для следующих сценариев:

- **Приложению или службе, работающей в облаке, требуется доступ к сетевым ресурсам, расположенным на локальных серверах.**

  При помощи Windows Azure Connect размещенные в облаке алгоритмы могут получить доступ к ресурсам локальной сети, используя те же процедуры и методы, что и локальные алгоритмы. Например, облачная служба может использовать знакомый синтаксис для подключения к расположенным локально общим папкам или устройствам, таким как принтеры. Приложение может осуществлять имперсонацию с использованием удостоверения учетной записи с правами доступа к локальному ресурсу. Этот метод позволяет использовать ту же семантику безопасности, которая применяется между локальными приложениями и ресурсами, что позволяет защищать ресурсы с помощью списков контроля доступа (access control lists, ACL).

- **Приложение, работающее в облаке, запрашивает доступ к источникам данных, управляемым локальными серверами.**

  Windows Azure Connect позволяет облачным службам создать сетевое подключение к локальным источникам данных, которые поддерживают IP-соединения. Примеры: SQL Server, компонент DCOM, очередь сообщений Microsoft. Облачные службы определяют параметры подключения точно таким же образом, как если бы они работали локально. Например, вы можете подключиться к SQL Server при помощи аутентификации SQL Server, указав имя и пароль соответствующей учетной записи SQL Server (SQL Server должен быть настроен в целях поддержки аутентификации SQL Server или смешанной аутентификации, чтобы можно было реализовать этот метод).

  Этот подход будет особенно полезен, если вы хотите переместить в облако пакеты прикладных программ от сторонних разработчиков. Поставщики, как правило, не предоставляют исходный код, поэтому вы не можете изменить принципы работы этих программ.

- **Приложение, работающее в облаке, должно выполняться в том же домене безопасности Windows, в котором действуют локальные алгоритмы.**

Windows Azure Connect позволяет присоединить развернутые в облаке роли к домену Windows Active Directory вашей организации. Таким образом, вы можете настроить корпоративную аутентификацию и доступ к ресурсам для обмена данными с размещенными в облаке виртуальными машинами. Эта функция позволяет воспользоваться преимуществами интегрированной безопасности при подключении к источникам данных, размещенным на ваших локальных серверах. Вы можете настроить роль на использование учетной записи домена, но роль также может использовать локальную учетную запись, при этом необходимо настроить пул приложений IIS, в рамках которого запускались веб-роли или рабочие роли с удостоверением учетной записи домена.

> **Мнение По**
>
> Используя Windows Azure Connect для присоединения облачных служб к корпоративному домену Windows, вы решаете многие проблемы, поскольку в противном случае потребуется сложное конфигурирование, при этом ваша система может случайно стать открытой для атак, если вы допустите какие-либо неверные настройки.

- **Приложению требуется доступ к локальным ресурсам, но алгоритмы этого приложения работают на другом узле или в партнерской организации.**

Windows Azure Connect позволяет создать упрощенную сеть VPN между удаленными узлами и партнерскими организациями. Windows Azure Connect также идеально подходит для решений, предназначенных, в том числе, для мобильных пользователей, которым необходимо подключаться к локальным ресурсам, например удаленным рабочим столам или общим папкам, со своих портативных компьютеров.

**Рисунок 1**

**Подключение к локальным ресурсам из партнерской организации или при помощи портативных компьютеров**

**Примечание**

Самая актуальная информация о передовых практиках в сфере реализации Windows Azure Connect представлена в блоге команды Windows Azure Connect: http://blogs.msdn.com/b/windows_azure_connect_team_blog.

### Архитектура и модель безопасности Windows Azure Connect

Служба Windows Azure Connect реализована как виртуальная сеть IPv6 посредством решения Windows Azure Connect для создания конечных точек, работающего на каждом сервере и в каждой роли, которые задействованы в этой виртуальной сети. Программное обеспечение для создания конечных точек прозрачно обрабатывает разрешения DNS и управляет IP-соединениями между серверами и ролями. Это программное обеспечение устанавливается автоматически для работающих в облаке ролей, которые настроены на поддержку соединений. Для локальных серверов необходимо вручную загрузить и установить решение Windows Azure Connect для создания конечных точек. Это решение работает в фоновом режиме, как служба Windows. Аналогично, если вы используете Windows Azure Connect для создания подключения из роли виртуальной машины, необходимо установить решение Windows Azure Connect для создания конечных точек для этой роли, прежде чем разворачивать ее в облаке.

Портал управления Windows Azure используется для создания маркера активации, который вы включаете в конфигурацию каждой роли и каждого экземпляра решения Windows Azure Connect для создания конечных точек, работающего локально. Windows Azure Connect использует этот маркер для привязки конечной точки соединения к подписке Windows Azure, поэтому виртуальная сеть будет доступна только для прошедших аутентификацию серверов и ролей. Комплексная защита трафика в виртуальной сети обеспечивается при помощи технологий IPsec через протокол Secure Socket Tunneling Protocol (SSTP), работающий на основе сертификатов. Windows Azure Connect автоматически предоставляет и настраивает необходимые сертификаты, не требуя никаких действий от оператора.

Решение Windows Azure Connect для создания конечных точек организует взаимодействие с каждым узлом с помощью службы Connect Relay, размещаемой и управляемой Microsoft в своих центрах обработки данных. Решение для создания конечных точек использует исходящие соединения HTTPS только для обмена данными со службой Windows Azure Connect Relay. Тем не менее решение Windows Azure Connect для создания конечных точек создает правило брандмауэра для протокола Internet Control Message Protocol for IPv6 (ICMPv6), который позволяет после установки использовать сообщения Router Solicitation (Type 133) и Router Advertisement (Type 134). Эти сообщения необходимы для создания и обслуживания соединения IPv6. Не следует отключать это правило.

**Мнение Бхарата**

Microsoft реализует отдельные экземпляры службы Windows Azure Connect Relay в каждом регионе. В целях обеспечения максимальной производительности выбирайте ближайший к вам регион, когда настраиваете службу Windows Azure Connect.


**Рисунок 2**

**Архитектура безопасности службы Windows Azure Connect**

При помощи портала управления вы настраиваете политики безопасности соединения, определяющие, какие серверы и роли могут взаимодействовать друг с другом. Вы создаете одну или несколько групп конечных точек, содержащих хост-серверы, из которых состоит ваше решение (и на которых установлено решение Windows Azure Connect для создания конечных точек), и также указываете роли Windows Azure, к которым они могут подключиться. Этот набор хост-серверов и ролей формирует единую виртуальную сеть.

> **Примечание**
>
> Более подробная информация о настройке службы Windows Azure Connect и создании групп конечных точек представлена в статье «Connecting Local Computers to Windows Azure Roles» на сайте MSDN: http://msdn.microsoft.com/en-us/library/gg433122.aspx.

### Ограничения Windows Azure Connect

Служба Windows Azure Connect предназначена для обеспечения прямого доступа к корпоративным ресурсам, размещенным локально или в облаке. Она предоставляет решение общего назначения, но при этом необходимо учитывать некоторые ограничения, перечисленные в следующем списке:

- Windows Azure Connect — это комплексное решение. Задействованные локальные компьютеры и роли виртуальной машины в облаке необходимо включать в одну виртуальную сеть. Локальные ресурсы и роль виртуальной машины можно защитить

с помощью списков ACL, однако эта защита опирается на проверку подлинности пользователей, получающих доступ к этим ресурсам, они должны быть определены в домене Windows, охватывающем виртуальную сеть, или в другом домене, которому доверяет данный домен. Следовательно, Windows Azure Connect не подходит для совместного использования ресурсов с внешними клиентами, например с покупателями, которые подключаются к вашим службами через Интернет. В такой ситуации более целесообразным представляется использование службы Windows Azure Service Bus Relay.

- Windows Azure Connect внедряет сеть IPv6, но инфраструктура IPv6 на базовой платформе не требуется. Однако все приложения, которые используют Windows Azure Connect для подключения к ресурсам, должны поддерживать протокол IPv6. Унаследованные приложения могут быть построены на основе IPv4, тогда их придется обновить или заменить в целях обеспечения корректной работы с Windows Azure Connect.
- Если вы используете Windows Azure Connect, чтобы присоединить роли к своему домену Windows, текущая версия Windows Azure Connect потребует от вас установить на контроллере домена решение Windows Azure Connect для создания конечных точек. Эта же машина должна выступать в качестве сервера Windows Domain Name System (DNS). Такие требования могут подлежать утверждению в отделе ИТ, а ваша организация может реализовывать политики, которые ограничивают возможности, связанные с конфигурированием контроллера домена. Тем не менее в будущих выпусках Windows Azure Connect эти требования могут измениться.
- Служба Windows Azure Connect специально предназначена для обеспечения взаимодействия между ролями, развернутыми в облаке, и локальными серверами. Она не позволяет создавать подключение между ролями, если роли должны совместно использовать ресурсы; предпочтительным решением представляется использование хранилища Windows Azure, очередей Windows Azure или таблиц баз данных, развернутых на технологической платформе SQL Azure.
- Вы можете использовать Windows Azure Connect только для подключения к серверам под управлением операционной системы Windows, решение Windows Azure Connect для создания конечных точек не предназначено для работы с другими операционными системами. Если вам требуется кроссплатформенное взаимодействие, следует рассмотреть возможность использования шины интеграции Windows Azure.

## Доступ извне к локальным службам организации при помощи службы Windows Azure Service Bus Relay

Служба Windows Azure Service Bus Relay формирует коммуникационную инфраструктуру, которая позволяет предоставить доступ к службе через Интернет — из внешней для брандмауэра или NAT-маршрутизатора среды. Служба Windows Azure Service Bus Relay предоставляет простой в использовании механизм для организации безопасного взаимодействия приложений и служб, работающих по обе стороны корпоративного брандмауэра, не требуя сложной настройки безопасности или нестандартной инфраструктуры обмена сообщениями.

### Рекомендации по применению службы Windows Azure Service Bus Relay

Служба Windows Azure Service Bus Relay идеально подходит для создания защищенного канала связи с локальными службами, а также для одноранговых подключений. Служба Windows Azure Service Bus Relay предоставляет целый ряд преимуществ:

- Полная совместимость с платформой Windows Communication Foundation (WCF). Имеющихся у вас знаний в области WCF будет достаточно, нет необходимости изучать новые модели для внедрения служб и клиентских приложений. Служба Windows Azure Service Bus Relay предоставляет привязки WCF, расширяя соответствующие возможности большинства существующих служб.
- Совместимость с платформами и технологиями для операционных систем, отличных от Windows. Служба Windows Azure Service Bus Relay основана на таких общепринятых стандартах, как HTTPS и TCP/SSL, это обеспечивает надежность отправки и приема сообщений. Также поддерживается интерфейс HTTP REST. Любые технологии, которые поддерживают генерацию и прием запросов HTTP REST, могут подключаться к службе, использующей службу Windows Azure Service Bus Relay. Вы также можете создавать службы, используя интерфейс HTTP REST.
- Вам не придется открывать входящие порты в брандмауэре своей организации. Служба Windows Azure Service Bus Relay использует только исходящие соединения.
- Поддержка присваивания имен и обнаружения. Служба Windows Azure Service Bus Relay ведет реестр активных служб в рамках пространства имен вашей организации. Клиентские приложения, имеющие соответствующие полномочия, могут запросить этот реестр, чтобы найти эти службы, также они могут загрузить метаданные, необходимые для подключения к службам, и выполнить операции, доступ к которым эти службы предоставляют. Реестр находится под управлением Windows Azure и эффективно использует масштабируемость и доступность, которые эта платформа обеспечивает.
- Поддержка федеративной безопасности в целях аутентификации запросов. Удостоверения пользователей и приложений, которые получают доступ к локальной службе через службу Windows Azure Service Bus Relay, не обязательно должны принадлежать домену безопасности вашей организации.
- Поддержка многих стандартных моделей обмена сообщениями, включая двустороннюю обработку запросов/ответов, одностороннюю передачу сообщений, работу с удаленными службами и многоадресную рассылку уведомлений о событиях.
- Поддержка балансировки нагрузки. На одной конечной точке можно создать до 25 слушателей. Когда шина интеграции получает запросы, отправленные конечной точке, то осуществляется балансировка запросов между этими слушателями.

Не забывайте, что служба Windows Azure Service Bus Relay подходит не для всех коммуникационных решений. Например, она обуславливает временную зависимость между локальными службами и клиентскими приложениями, которые подключаются к ним. Служба должна быть запущена, прежде чем клиентское приложение сможет подключиться к ней, в противном случае клиентское приложение получит исключение **EndpointNotFoundException** (это ограничение актуально даже при наличии привязок **NetOnewayRelayBinding** и **NetEventRelayBinding**, которые описаны в разделе «Выбираем привязку для службы» далее в настоящем приложении). Кроме того, служба Windows Azure Service Bus Relay в значительной степени зависит от надежности сети. Служба может быть запущена, но если клиентское приложение не может подключиться к ней из-за сбоя в сети, клиент снова получит исключение **EndpointNotFoundException**. В таком случае лучше использовать очереди шины интеграции Windows Azure. Для получения дополнительной информации см. раздел «Реализация подхода "коммуникации без границ" с использованием очередей шины интеграции» далее в настоящем приложении.

Службу Windows Azure Service Bus Relay целесообразно использовать в рамках следующих сценариев:

- **Приложению, работающему в облаке, требуется контролируемый доступ к вашей локальной службе. Ваша служба создана при помощи WCF.**

  Это основной сценарий использования службы Windows Azure Service Bus Relay. Ваша служба создана при помощи WCF и Windows Azure SDK. Она использует привязки WCF, предоставляемые сборкой **Microsoft.ServiceBus**, чтобы открыть исходящий коммуникационный канал через брандмауэр к службе Windows Azure Service Bus Relay, далее служба ждет входящие запросы. Клиентские приложения в облаке используют те же привязки Windows Azure SDK и WCF для подключения к службе и отправки запросов через службу Windows Azure Service Bus Relay. Ответы от локальных служб направляются клиентскому приложению через тот же канал связи через службу Windows Azure Service Bus Relay. Локальные службы могут размещаться в пользовательском приложении или с использованием служб Internet Information Services (IIS). При использовании IIS администратор может настроить автоматический запуск локальной службы, эта служба зарегистрирует соединение со службой Windows Azure Service Bus Relay и будет готова получать запросы от клиентских приложений.

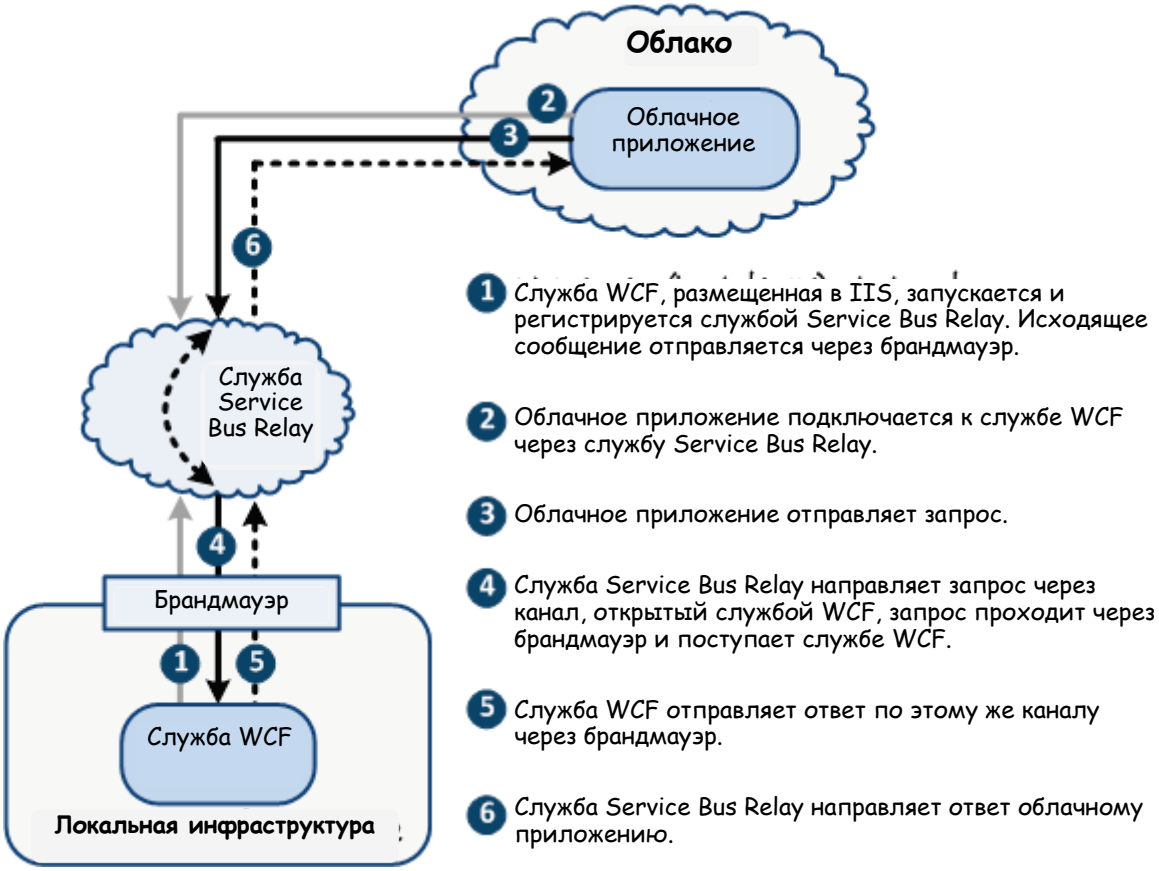



**Рисунок 3**

### Маршрутизация запросов и ответов через службу Windows Azure Service Bus Relay

Эта модель используется для предоставления удаленного доступа к существующим локальным службам, которые изначально не были доступны за пределами вашей организации. Вы создаете фасад для своих служб и публикуете их в службе Windows Azure Service Bus Relay. Внешние по отношению к вашей организации приложения, имеющие соответствующие права доступа, смогут подключиться к вашим службам через службу Windows Azure Service Bus Relay.

Изначально все сообщения проходят через службу Windows Azure Service Bus Relay, но в качестве механизма оптимизации служба, предоставляющая конечную точку TCP, может использовать прямое соединение, минуя службу Windows Azure Service Bus Relay после того, как служба и клиент успешно прошли аутентификацию. Это прямое соединение управляется службой Windows Azure Service Bus Relay. Алгоритм прямого соединения сокетов использует только исходящие соединения в обход брандмауэра и основывается на алгоритмах взаимного прогнозирования портов в обход NAT. Если прямое соединение может быть установлено, ретранслированное соединение будет автоматически трансформировано в прямое. Если прямое соединение установить нельзя, сообщения будут возвращаться для передачи через службу Windows Azure Service Bus Relay.

Алгоритм обхода NAT зависит от быстрой координации и эффективного прогнозирования ожидаемого поведения NAT. Следовательно, этот алгоритм будет эффективен для дома и офиса с небольшим количеством клиентов, но для масштабных сетей он не подходит.


**Рисунок 4**

### Установление прямого соединения через TCP/SSL

- **Приложению, работающему в партнерской организации, требуется контролируемый доступ к вашей локальной службе. Клиентское приложение создано с помощью технологии, отличной от WCF.**

В рамках данного сценария клиентское приложение может использовать интерфейс HTTP REST, предоставляемый службой Windows Azure Service Bus Relay для поиска локальной службы и отправки ей запросов.

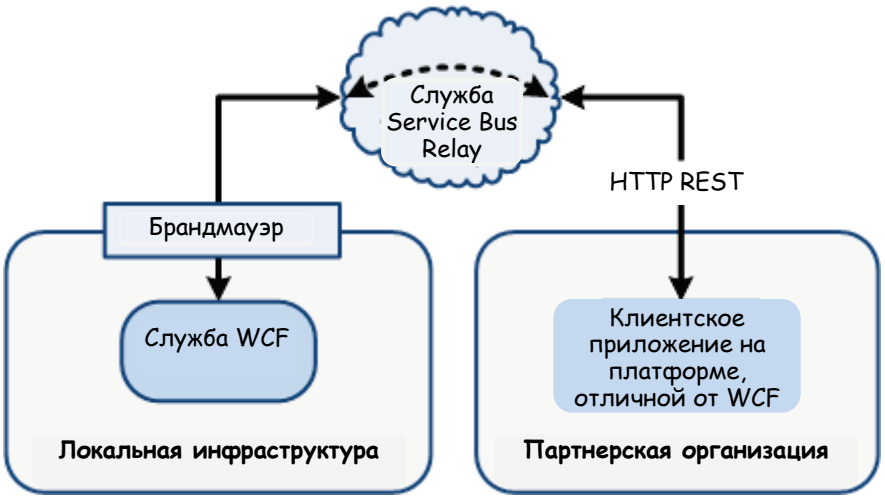



**Рисунок 5**

**Подключение к локальной службе при помощи запросов HTTP REST**

- **Приложению, работающему в облаке или партнерской организации, требуется контролируемый доступ к вашей локальной службе. Ваша служба создана с помощью технологии, отличной от WCF.**

Возможно, у вас есть службы, реализованные с использованием таких технологий как Perl или Ruby. В рамках такого сценария служба может подключиться к службе Windows Azure Service Bus Relay через интерфейс HTTP REST и ждать запросов.


**Рисунок 6**

**Подключение локальной службы, созданной при помощи Ruby, к службе Windows Azure Service Bus Relay**

- **Приложение, работающее в облаке или партнерской организации, создает запросы, которые могут быть отправлены нескольким локальным службам.**

В рамках данного сценария один запрос от клиентского приложения может быть отправлен и обработан более чем одной локальной службой. Фактически сообщение от клиентского приложения рассылается всем локальным службам, зарегистрированным в одной конечной точке службы Windows Azure Service Bus Relay. Отправка всех сообщений осуществляется клиентом в одностороннем режиме, службы не должны отвечать. Такой подход удобен для создания систем, которые реагируют на события. Каждое сообщение, отправленное клиентским приложением, представляет собой событие, и службы могут прозрачно подписаться на это событие, зарегистрировавшись в службе Windows Azure Service Bus Relay.


**Рисунок 7**

**Многоадресная рассылка при помощи службы Windows Azure Service Bus Relay**

**Мнение Бхарата**

Вы также можете создать систему обработки событий с помощью топиков и подписок шины интеграции. Служба Windows Azure Service Bus Relay — простое и эффективное решение, но топики и подписки обеспечивает большую гибкость. Рекомендации в отношении маршрутизации сообщений при помощи топиков и подписок шины интеграции представлены в «Приложении Г. Реализация бизнес-логики и маршрутизации сообщений с целью организации совместной работы».

- **Приложению, работающему локально или в партнерской организации, требуется контролируемый доступ к вашей облачной службе. Ваша служба реализована как рабочая роль Windows Azure и создана при помощи WCF.**

Данный сценарий описывает обратную ситуацию по отношению к предыдущей. Во многих ситуациях приложение, работающее локально или в партнерской организации, может получить доступ к службе WCF, которая реализована как рабочая роль, напрямую, без задействования службы Windows Azure Service Bus Relay. Тем не менее этот сценарий будет полезен, если служба WCF, ранее размещенная локально, и алгоритмы, работающие в другом месте, соединяются через службу Windows Azure Service Bus Relay, как описано в предыдущих примерах, но теперь служба работает в Windows Azure. Реструктуризация службы, перемещение ее в облако и публикация через службу Windows Azure Service Bus Relay избавляют от необходимости изменять параметры конечной точки, таким образом, клиентские приложения также не придется модифицировать или перенастраивать, они просто продолжат работать. Данная архитектура также упрощает задачу защиты канала взаимодействия со службой посредством соответствующих привязок WCF.


**Рисунок 8**

**Маршрутизация запросов к рабочей роли через службу Windows Azure Service Bus Relay**

### Рекомендации по защите службы Windows Azure Service Bus Relay

Конечные точки службы Windows Azure Service Bus Relay организованы при помощи пространств имен шины интеграции. При создании новой службы, которая взаимодействует с клиентскими приложениями с помощью службы Windows Azure Service Bus Relay, можно использовать портал управления для создания нового пространства имен службы. Это пространство имен должно быть уникальным, оно определяет универсальный идентификатор ресурса (uniform resource identifier, URI), который ваша служба предоставляет. Клиентские приложения указывают этот URI для подключения к вашей службе через службу Windows Azure Service Bus Relay. Например, если вы создаете пространство имен со значением **TreyResearch** и публикуете в нем службу **OrdersService**, эта служба получит следующий идентификатор URI: sb://treyresearch.servicebus.windows.net/OrdersService.

Службы, которые вы публикуете через службу Windows Azure Service Bus Relay, сами по себе являются ценным активом и могут предоставлять доступ к конфиденциальной информации, поэтому их необходимо защищать. У этой задачи есть несколько аспектов:

**Мнение По**

Помните, что несмотря на то, что шина интеграции управляется и поддерживается одним или несколькими центрами обработки данных Microsoft, приложения подключаются к службе Windows Azure Service Bus Relay через Интернет. Неавторизованные приложения, если они смогли подключиться к пространствам имен шины интеграции, могут выполнять распространенные атаки, например организовать отказ в обслуживании, чтобы нарушить вашу деятельность, или активное вмешательство с целью кражи информации, которая передается вашим службам. Поэтому вы должны защищать пространства имен своей шины интеграции и использующие ее службы не менее тщательно, чем свои локальные активы.

- Вы должны ограничить доступ к пространству имен вашей шины интеграции и организовать аутентификацию служб и клиентских приложений. Каждая служба и клиентское приложение должны иметь удостоверение, которое служба Windows Azure Service Bus Relay может проверить. Как сказано в «Приложении Б. Аутентификация пользователей и авторизация запросов», у шины интеграции есть собственный поставщик удостоверений в службе Windows Azure Access Control (ACS), и вы можете предоставить удостоверения и ключи каждой службе и каждому пользователю, работающему с клиентским приложением. Вы можете также организовать федеративную безопасность с помощью ACS с целью аутентификации запросов средствами службы безопасности, работающей локально или в партнерской организации.

При настройке доступа к службе через службу Windows Azure Service Bus Relay домашней областью приложения доверенного участника, с которой вы ассоциируете аутентифицированные удостоверения, является URL конечной точки службы, через которую эта служба принимает запросы.

**Мнение Бхарата**

Шина интеграции может работать со сторонними поставщиками удостоверений, такими как Windows Live ID, Google и Yahoo!, но по умолчанию используется встроенный поставщик, включенный в службу ACS.

- Вы должны ограничивать варианты взаимодействия клиентов с конечными точками, опубликованными через пространство имен шины интеграции. Например, большинство клиентских приложений должны только отправлять сообщения службе (и получать ответ), а службы должны только прослушивать канал и получать входящие запросы. Шина интеграции определяет для утверждений тип **net.windows.servicebus.action**, который поддерживает следующие значения: **Send**, **Listen** и **Manage**. При помощи ACS вы можете реализовать группу правил для каждого идентификатора URI, определенного в пространстве имен вашей шины интеграции, эти правила будут сопоставлять аутентифицированные удостоверения с одним или более из указанных значений для утверждения.

Когда служба запускается и пытается объявить конечную точку, она предоставляет свои учетные данные службе Windows Azure Service Bus Relay. Эти учетные данные проходят проверку и используются, чтобы определить наличие у данной службы прав на создание конечной точки. Общий подход, используемый многими службами, заключается в определении поведения конечной точки, что соответствует элементу **transportClientEndpointBehavior**

в конфигурационном файле. У данного элемента есть параметр **clientCredentials**, который позволяет службе указать имя удостоверения и соответствующий симметричный ключ, позволяющий проверить это удостоверение. Клиентское приложение может применять подобный подход, за исключением того, что оно указывает имя и симметричный ключ для удостоверения, с которым работает приложение, а не служба.

> **Примечание**
>
> Дополнительная информация о защите служб при помощи службы Windows Azure Service Bus Relay представлена в статье «Securing and Authenticating a Service Bus Connection» на портале MSDN: http://msdn.microsoft.com/en-us/library/dd582773.aspx.

Обратите внимание, что использование поставщиков маркеров типа «разделенный секрет» является только одним из способов предоставления учетных данных для службы и клиентского приложения. При указании этого поставщика служба ACS сама проверяет имя и ключ и, в случае успешной аутентификации, генерирует маркер Simple Web Token (SWT), содержащий утверждения для этого удостоверения, в соответствии с правилами, настроенными в ACS. Эти утверждения определяют наличие у службы или клиентского приложения необходимых прав на получение или отправку сообщений. Также доступны другие варианты работы с поставщиками аутентификации, например с использованием маркеров SAML. Вы можете указать другую службу Security Token Service (STS), отличную от той, которая предоставляется ACS, для аутентификации удостоверений и выдачи утверждений.

- Когда клиентское приложение установило подключение к службе через службу Windows Azure Service Bus Relay, вы должны тщательно контролировать операции, которые клиентское приложение может инициировать. Процесс авторизации зависит от того, каким образом вы реализуете службу, и лежит за пределами зоны ответственности службы ACS. Однако вы можете использовать ACS для создания утверждений для прошедшего аутентификацию клиента, эти утверждения ваша служба сможет использовать для процесса авторизации.

> **Мнение Маркуса**
>
> Если вы используете WCF для реализации своих служб, тодолжны подумать о создании поставщика авторизации Windows Identity Foundation, чтобы отделить правила авторизации от бизнес-логики своей службы.

- Все сообщения между вашей службой и клиентскими приложениями, скорее всего, будут проходить через сети общего пользования или Интернет. Вы должны защитить эти сообщения, обеспечив соответствующий уровень безопасности передачи данных, например при помощи SSL или HTTPS.

> **Мнение Маркуса**
>
> Если вы используете WCF для внедрения своих служб, поддержка безопасности на транспортном уровне — это на самом деле всего лишь вопрос выбора наиболее подходящей привязки WCF и установки соответствующих параметров для указания способа шифрования и защиты данных.

На рисунке 9 показаны основные рекомендации по защите служб, доступ к которым предоставляется через службу Windows Azure Service Bus Relay.

**Примечание**

Вы можете организовать аутентификацию и шифрование сообщений путем настройки привязки WCF, используемый службой. Более подробная информация представлена в статье «Securing and Authenticating a Service Bus Connection»: http://msdn.microsoft.com/en-us/library/windowsazure/dd582773.aspx.

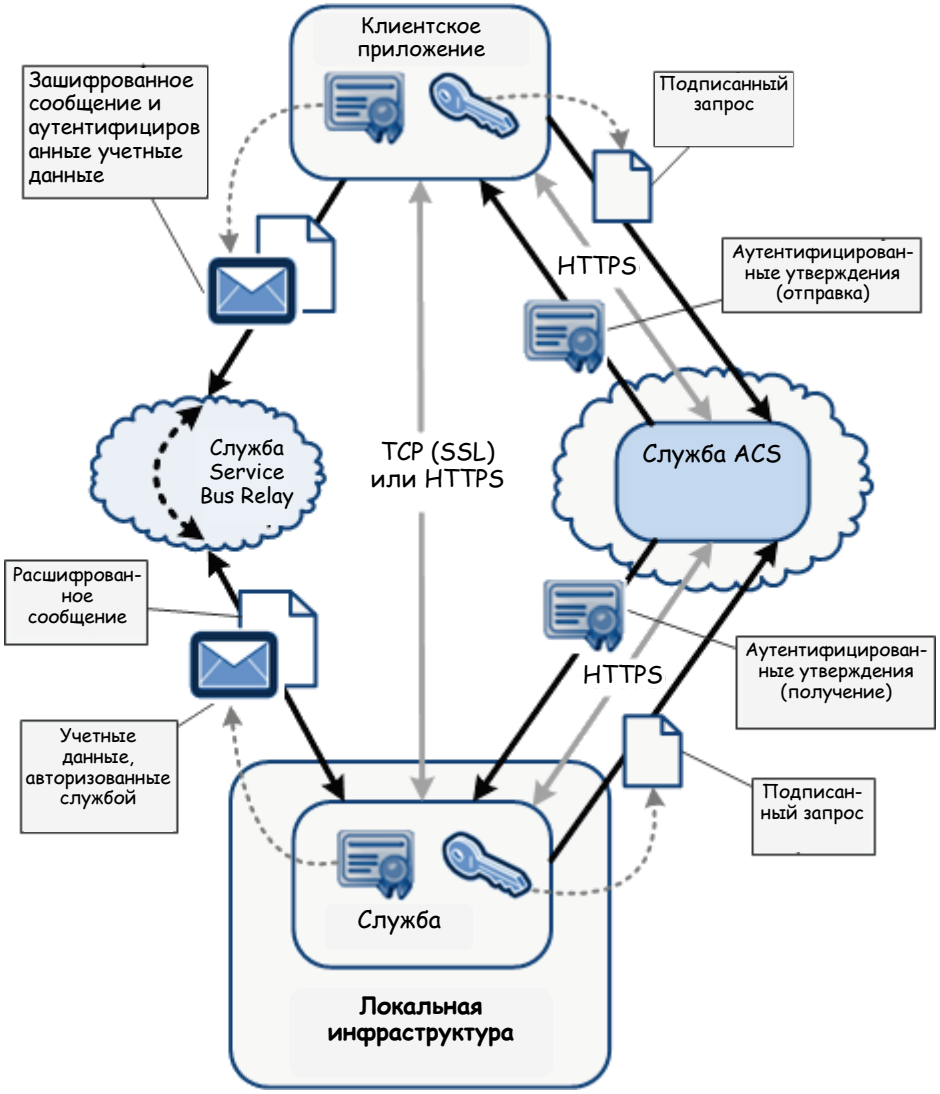



**Рисунок 9**

## Рекомендации по защите служб, доступ к которым предоставляется через службу Windows Azure Service Bus Relay

Многие организации создают исходящие правила брандмауэра, которые основаны на списках разрешенных IP-адресов. При использовании такой конфигурации, чтобы обеспечить доступ к шине интеграции или службе ACS, необходимо добавить адреса соответствующих служб Windows Azure в вашем брандмауэре. Эти адреса зависят от региона, в котором размещена служба, они могут меняться с течением времени. В следующем списке приведены актуальные адреса для каждого региона на момент написания данного приложения:

- **Азия (юго-восток)**: 207.46.48.0/20, 111.221.16.0/21, 111.221.80.0/20
- **Азия (восток)**: 111.221.64.0/22, 65.52.160.0/19

- **Европа (запад)**: 94.245.97.0/24, 65.52.128.0/19
- **Европа (север)**: 213.199.128.0/20, 213.199.160.0/20, 213.199.184.0/21, 94.245.112.0/20, 94.245.88.0/21, 94.245.104.0/21, 65.52.64.0/20, 65.52.224.0/19
- **США (север, центральная часть)**: 207.46.192.0/20, 65.52.0.0/19, 65.52.48.0/20, 65.52.192.0/19, 209.240.220.0/23
- **США (юг, центральная часть)**: 65.55.80.0/20, 65.54.48.0/21, 65.55.64.0/20, 70.37.48.0/20, 70.37.64.0/18, 65.52.32.0/21, 70.37.160.0/21

**Мнение По**

Списки разрешенных IP-адресов представляются не очень подходящей стратегией обеспечения безопасности в организации, когда целевые адреса массово соответствуют мультитенатной инфраструктуре, такой как Windows Azure (или любой другой общедоступной облачной платформе).

### Принципы именования служб в службе Windows Azure Service Bus Relay

При наличии большого количества служб необходимо утвердить стандартизированное соглашение об именовании конечных точек этих служб. Это поможет защищать, контролировать службы и клиентские приложения, которые подключаются к ним, и управлять ими. Многие организации обычно применяют иерархический подход. Например, если у компании Trey Research есть сайты в Чикаго, Нью-Йорке и Вашингтоне, и на каждом из них принимаются заказы и организуется отправка, администратор может зарегистрировать идентификаторы URI в соответствии со следующим правилом именования:

- sb://treyresearch.servicebus.windows.net/chicago/ordersservice
- sb://treyresearch.servicebus.windows.net/chicago/shippingservice
- sb://treyresearch.servicebus.windows.net/newyork/ordersservice
- sb://treyresearch.servicebus.windows.net/newyork/shippingservice
- sb://treyresearch.servicebus.windows.net/washington/ordersservice
- sb://treyresearch.servicebus.windows.net/washington/shippingservice

Однако, когда вы регистрируете для службы идентификатор URI в службе Windows Azure Service Bus Relay, никакая другая служба не сможет прослушивать URI более низкого уровня по отношению к этой службе. Это означает, что если в будущем Trey Research решит внедрить дополнительную службу для обработки заказов от эксклюзивных клиентов, нельзя будет зарегистрировать эту службу с помощью таких URI как sb://treyresearch.servicebus.windows.net/chicago/ordersservice/exclusive.

Чтобы избежать подобных проблем, вы должны убедиться, что начальная часть каждого URI уникальна. Можно сгенерировать новый идентификатор GUID для каждой службы, а также включить в него город и имя службы из URI. В случае с Trey Research, идентификаторы URI для служб, размещенных в Чикаго, включая службу для обработки заказов от эксклюзивных клиентов, могут быть следующими:

- sb://treyresearch.servicebus.windows.net/B3B4D086-BEB9-4773-97D3-064B0DD306EA/chicago/ordersservice
- sb://treyresearch.servicebus.windows.net/DD986578-EAB6-FC84-5490-075F34CD8B7A/chicago/ordersservice/exclusive

- sb://treyresearch.servicebus.windows.net/A8B3CC55-1256-5632-8A9F-FF0675438EEC/chicago/shippingservice

Дополнительная информация о принципах именования для служб Windows Azure Service Bus Relay представленавстатье «AppFabric Service Bus – Things You Should Know – Part 1 of 3 (Naming Your Endpoints)» на сайте http://windowsazurecat.com/2011/05/appfabric-service-bus-things-you-should-know-part-1-of-3-naming-your-endpoints.

### Выбор привязки для службы

Целью службы Windows Azure Service Bus Relay является обеспечение безопасного и надежного подключения клиентских приложений к локальным службам, работающим на стороне вашего корпоративного брандмауэра. После регистрации службы в службе Windows Azure Service Bus Relay значительно упрощаются задачи, связанные с ее защитой, аутентификация и авторизация запросов могут выполняться прозрачно за пределами бизнес-логики службы. Если вы создаете свои службы при помощи WCF, можно использовать типы и интерфейсы прикладного программирования (API) из сборки **System.ServiceModel**, с которыми вы уже знакомы. Пакет Windows Azure SDK включает транспортные привязки, модели поведения и другие расширения в сборке **Microsoft.ServiceBus**, обеспечивающие интеграцию созданной на базе WCF службы со службой Windows Azure Service Bus Relay.

> **Мнение Маркуса**
>
> Если вы знакомы с технологией создания служб и клиентских приложений с использованием WCF, то должны без труда освоить службу Windows Azure Service Bus Relay.

Как и в случае со стандартной службой WCF, выбор подходящей привязки для службы, использующей службу Windows Azure Service Bus Relay, влияет на возможность подключения клиентских приложений, а также на функциональность и безопасность транспортировки. Сборка **Microsoft.ServiceBus** предоставляет четыре набора привязок:

- Привязки HTTP: BasicHttpRelayBinding, WSHttpRelayBinding, WS2007HttpRelayBinding и WebHttpRelayBinding.

  Эти привязки очень похожи на их стандартные эквиваленты в WCF (**BasicHttpBinding**, **WSHttpBinding**, **WS2007HttpBinding и WebHttpBinding**), только они предназначены для расширения инфраструктуры базового канала WCF, а сообщения направляются через службу Windows Azure Service Bus Relay. Они обеспечивают те же возможности для подключения и предлагают тот же набор функций, что и их эквиваленты в WCF, а также могут работать по протоколу HTTP и HTTPS. Например, привязка **WS2007HttpRelayBinding** поддерживает защиту на уровне сообщений SOAP, надежные сеансы и поток транзакций. Эти привязки открывают канал к службе Windows Azure Service Bus Relay с использованием только исходящих соединений; вам не придется открывать дополнительные входящие порты в своем корпоративном брандмауэре.

- Привязка TCP: NetTcpRelayBinding.

  Эта привязка по своим функциям аналогична привязке **NetTcpBinding** в WCF. Она поддерживает дуплексные обратные вызовы и обеспечивает более высокую производительность, по сравнению с HTTP-привязками, однако является менее портативной. Клиентским приложениям, которые подключаются к службе с использованием этой привязки, возможно, потребуется отправлять запросы и получать ответы с бинарной кодировкой TCP, в зависимости от того, как привязка настраивается службой. Несмотря на то, что эта привязка не требует открытия

дополнительных входящих портов в вашем корпоративном брандмауэре, вам придется открыть исходящие TCP-порты 808 и 828, если вы используете SSL.

Эта привязка также поддерживает гибридный режим благодаря свойству **ConnectionMode** (привязки HTTP не поддерживают этот тип соединения). По умолчанию данная привязка активирует режим **Relayed**, но рекомендуется рассмотреть вопрос о переводе ее в режим **Hybrid**, если вы хотите воспользоваться преимуществами более производительных функций, которые работают в обход службы Windows Azure Service Bus Relay. Тем не менее алгоритм прогнозирования NAT, который устанавливает прямую связь между службой и клиентским приложением, также требует открыть исходящие TCP-порты 818 и 819 в корпоративной сети. Наконец, обратите внимание, что гибридный режим взаимодействия требует, чтобы привязка была настроена на обеспечение безопасности на уровне сообщений.

**Мнение Маркуса**

Для адресов, предлагаемых через привязку **NetTcpRelayBinding**, используется схема сети **sb**, а не схема **net.tcp**, применяемая привязкой **NetTcpBinding** в WCF. Например, созданная компанией Trey Research служба Orders может иметь следующий адрес: **sb://treyresearch.servicebus.windows.net/OrdersService**.

- Односторонняя привязка: **NetOnewayRelayBinding**.

Эта привязка организует одностороннюю передачу сообщений с буферизацией. Клиентское приложение посылает запросы в буфер, управляемый службой Windows Azure Service Bus Relay, которая доставляет сообщение службе. Эта привязка подходит для реализации службы, которая предоставляет асинхронные операции, поскольку они могут быть поставлены в очередь и включены в график службой Windows Azure Service Bus Relay, что обеспечивает оптимальную пропускную способность без ущерба для производительности службы. Тем не менее доставка сообщений не гарантируется, если служба отключается до того, как служба Windows Azure Service Bus Relay направит ей сообщения, то эти сообщения будут потеряны. Кроме того, порядок, в котором отправленные клиентским приложением сообщения, передаются службе, также не контролируется.

Эта привязка использует TCP-подключение к службе, поэтому необходимо открыть исходящие порты 808 и 828 (для SSL) в брандмауэре.

- Привязка с многоадресной рассылкой: **NetEventRelayBinding**.

Эта привязка представляет собой специализированную версию привязки **NetOnewayRelayBinding**, которая позволяет нескольким службам зарегистрировать одну и ту же конечную точку в службе Windows Azure Service Bus Relay. Клиентские приложения могут подключаться при помощи привязки **NetEventRelayBinding** или **NetOnewayRelayBinding**. Взаимодействие одностороннее, доставка и конкретный порядок сообщений не гарантируются.

Эта привязка идеально подходит для создания системы обработки событий. «N» клиентских приложений могут подключиться к «M» служб, при этом служба Windows Azure Service Bus Relay выступает в качестве концентратора событий. Как и в случае с **NetOnewayRelayBinding**, эта привязка использует TCP-подключение к службе, поэтому необходимо открыть исходящие порты 808 и 828 (для SSL).

### Сравнение службы Windows Azure Service Bus Relay и службы Windows Azure Connect

Некоторые функции службы Windows Azure Service Bus Relay и службы Windows Azure Connect пересекаются. В процессе принятия решения о том, какой из этих технологий воспользоваться, нужно учитывать следующее:

- Служба Windows Azure Service Bus Relay может предоставить доступ к службам, которые выступают в качестве оболочки для локальных ресурсов. Эти службы могут играть роль фасадов, которые предоставляют строго контролируемый селективный доступ к «упакованным» ресурсам. Клиентские приложения, отправляющие запросы, можно аутентифицировать с помощью службы ACS и федеративной безопасности, при этом приложениям не придется предъявлять удостоверения, которые определяются в корпоративном домене Windows вашей организации.

  Служба Windows Azure Connect предназначена для организации прямого доступа к ресурсам, которые не являются общедоступными. Вы можете защитить эти ресурсы при помощи списков ACL, но все клиентские приложения, использующие эти ресурсы, должны иметь удостоверение, принадлежащее корпоративному домену Windows вашей организации.

- Служба Windows Azure Service Bus Relay ведет реестр общедоступных служб в рамках пространства имен Windows Azure. Клиентское приложение с соответствующими правами безопасности может запросить этот реестр и получить список служб и их метаданные (если они опубликованы); эту информацию приложения используют для подключения к службе и вызова ее операций. Этот механизм поддерживает динамические клиентские приложения с функцией обнаружения служб во время выполнения.

  Служба Windows Azure Connect не поддерживает перечень ресурсов, и клиентскому приложению трудно обнаружить ресурсы во время выполнения.

- Клиентские приложения, которые взаимодействуют со службами через службу Windows Azure Service Bus Relay, могут установить прямое соединение, минуя службу Windows Azure Service Bus Relay сразу после первоначального обмена данными.

  Все запросы Windows Azure Connect проходят через службу Windows Azure Service Bus Relay, вы не можете установить прямое соединение с ресурсами (однако служба Windows Azure Connect использует службу Windows Azure Service Bus Relay прозрачным образом).

## Реализация подхода «коммуникации без границ» с использованием очередей шины интеграции

Очереди шины интеграции позволяют отделить службы от клиентских приложений, которые их используют, как с точки зрения функциональных возможностей (клиентскому приложению не придется использовать специфический интерфейс или прокси-сервер для отправки сообщений получателю), так и с точки зрения времени (получатель не обязательно должен быть запущен, когда клиентское приложение отправляет ему сообщения). Очереди шины интеграции используют надежный транзакционный метод обмена сообщениями с гарантированной доставкой, поэтому сообщения никогда не будут случайно потеряны. Кроме того, очереди шины интеграции устойчивы к сбоям сети: если клиентскому приложению удалось отправить сообщение в очередь, оно будет доставлено сразу, как только служба сможет подключиться к очереди.

> Когда вы имеете дело с очередями сообщений, имейте в виду, что клиентские приложения и службы могут как отправлять, так и получать сообщения. Поэтому в данном разделе мы говорим об «отправителях» и «получателях», а не о клиентских приложениях и службах.

### Сообщения шины интеграции

Сообщение шины интеграции — это экземпляр класса **BrokeredMessage**. Оно состоит из двух элементов: тела сообщения, которое содержит отправляемую информацию, а также набора свойств сообщения, который можно использовать для добавления к сообщению метаданных.

Тело сообщения непрозрачно для инфраструктуры очереди шины интеграции, и оно может содержать любую определяемую приложением информацию, если ее можно сериализовать. В целях обеспечения максимальной безопасности, тело сообщения может быть зашифровано. Содержимое сообщения смогут распознать только приложение-отправитель и приложение-получатель, этого нельзя сделать даже при помощи портала управления.

> **Мнение Маркуса**
>
> Данные в сообщении должны поддерживать сериализацию. По умолчанию класс **BrokeredMessage** использует объект **DataContractSerializer** и бинарный компонент **XmlDictionaryWriter** для выполнения этой операции, однако вы можете создать собственный объект **XmlObjectSerializer**, если необходимо модифицировать процесс сериализации данных. Тело сообщения также может представлять собой поток.

Инфраструктура очереди шины интеграции, напротив, может анализировать метаданные сообщения. Некоторые элементы метаданных определяют стандартные свойства системы обмена сообщениями, значения для которых приложение может установить. Эти элементы также используются инфраструктурой очередей шины интеграции для решения таких задач, как уникальная идентификация сообщений, определение сеанса для сообщения, указание срока действия недоставленных сообщений и многие другие стандартные операции. Сообщения также предоставляют ряд управляемых системой свойств только для чтения, например размер сообщения и количество неудачных попыток извлечения сообщения получателем в режиме **PeekLock**. Кроме того, сообщение может определять пользовательские свойства и включать их в метаданные. Эти элементы, как правило, используются для передачи дополнительной информации, характеризующей контент сообщения, они также могут использоваться шиной интеграции для фильтрации и маршрутизации сообщений в процессе отправки их получателю.

### Рекомендации по применению очередей шины интеграции

Очереди шины интеграции идеально подходят для создания системы на основе асинхронного обмена сообщениями. Вы можете создавать приложения и службы, использующие очереди шины интеграции, при помощи пакета Windows Azure SDK. Этот пакет SDK включает в себя интерфейсы для взаимодействия непосредственно с объектной моделью очередей шины интеграции, но он также предоставляет привязки, которые позволяют приложениям и службам WCF подключаться к очередям, используя метод, аналогичный подключению к очередям сообщений Microsoft Windows в корпоративной среде.

> **Мнение Бхарата**
>
> До появления очередей шины интеграции платформа Windows Azure предоставляла буферы сообщений. Они по-прежнему доступны, но включены в решение только в целях обеспечения обратной совместимости. Если вы создаете новую систему, необходимо использовать очереди шины интеграции.
>
> Необходимо также отметить, что очереди шины интеграции отличаются от очередей хранилища Windows Azure, которые используются прежде всего как коммуникационный механизм для организации связи между веб-ролями и рабочими ролями, работающими на том же узле.

Очереди шины интеграции позволяют использовать различные стандартные модели и помогут вам в создании гибких решений, что описано в следующих сценариях:

- **Отправитель должен переслать одно или несколько сообщений получателю. Сообщения должны быть доставлены в определенном порядке, доставка должна быть гарантирована, даже если получатель не работает, когда отправитель послал сообщение.**

  Это наиболее распространенный сценарий применения очереди шины интеграции и типовая модель реализации асинхронной обработки. Отправитель посылает сообщение в очередь, и через определенный момент времени получатель извлекает сообщение из той же очереди. Очередь шины интеграции, по сути, является структурой «первым пришел и первым ушел» (first-in-first-out, FIFO), и по умолчанию сообщения будут получены в том же порядке, в котором они были отправлены.

  Отправитель и получатель не зависят друг от друга и могут работать удаленно, в отличие от ситуации с использованием службы Windows Azure Service Bus Relay, они не должны быть запущены одновременно. Например, получатель может быть временно отключен в целях технического обслуживания. Очередь выступает в качестве буфера и надежно хранит сообщения для последующей обработки. Важно отметить, что несмотря на то, что очереди шины интеграции находятся в облаке, и отправитель, и получатель могут находиться в другом месте. Например, отправителем может быть локальное приложение, а получателем — служба, работающая в партнерской организации.

  Включенные в пакет Windows Azure SDK интерфейсы API для очереди шины интеграции представляют собой оболочки для набора интерфейсов HTTP REST. Работать с очередями шины интеграции могут как приложения и службы, созданные с помощью Windows Azure SDK, так и приложения и службы, основанные на технологиях, которые не поддерживаются пакетом Windows Azure SDK. На рисунке 10 показан пример архитектуры, где отправителем является локальное приложение, а получателем — рабочая роль, развернутая в облаке. В данном примере локальное приложение создано средствами языка программирования Java и работает под управлением Linux, поэтому для отправки сообщений используются запросы HTTP REST. Рабочая роль представлена службой WCF, реализованной с помощью Windows Azure SDK и интерфейсов API очереди шины интеграции.


**Рисунок 10**

**Отправка и получение сообщений в определенном порядке при помощи очередей шины интеграции**

- **Несколько отправителей, работающих в партнерских организациях или в облаке, должны отправлять сообщения вашей системе. Эти сообщения могут подлежать сложной обработке после получения. Приложение-получатель работает локально, и вам нужно подобрать оптимальное время для его запуска, чтобы исключить негативное влияние на основные бизнес-операции.**

Очереди шины интеграции идеально подходят для сценариев пакетной обработки, в которых логика обработки сообщений работает локально и может потреблять значительные ресурсы. В данном случае вы можете выполнять обработку сообщений в периоды минимальной нагрузки, чтобы избежать негативного воздействия на производительность критически важных бизнес-компонентов. В соответствии с данным стилем обработки сообщений, отправители могут помещать запросы в очередь шины интеграции, когда получатель находится в автономном режиме. В определенное время вы можете запустить приложение-получатель, и оно начнет извлекать и обрабатывать каждое сообщение из очереди. Когда получатель обработает последнее сообщение из очереди, он может завершить работу и освободить используемые ресурсы.

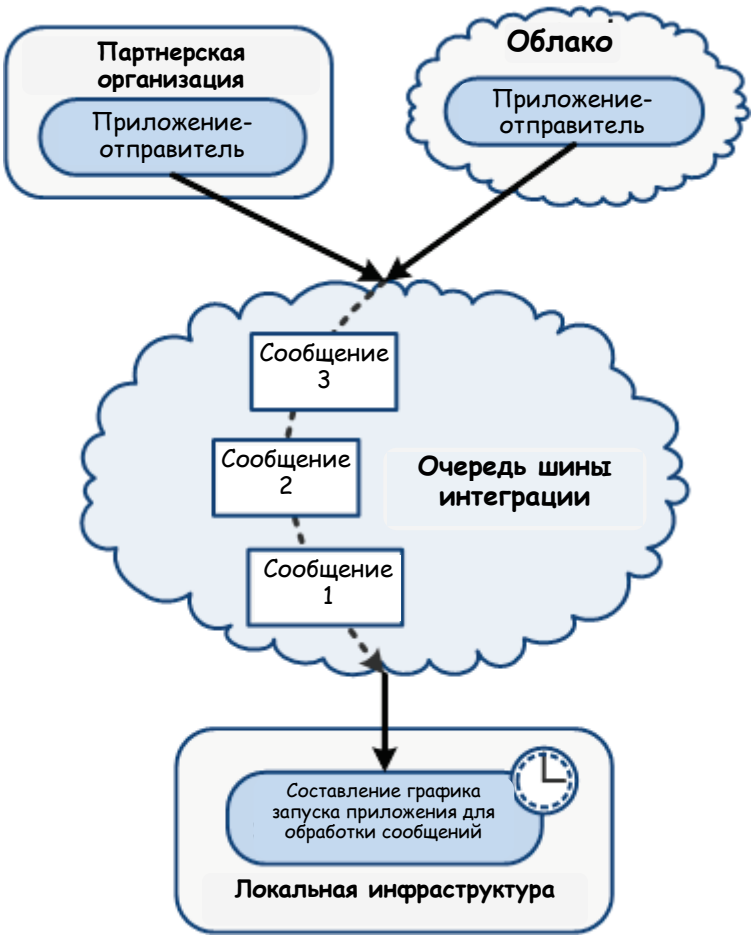



**Рисунок 11**

### Реализация пакетной обработки при помощи очереди шины интеграции

Вы можете использовать аналогичное решение для устранения проблемы, связанной с *нагрузочной способностью по входу (fan-in)*, когда неожиданно большое количество клиентских приложений отправляют большое количество запросов локальной службе. Если служба попытается обработать эти запросы синхронно, это может привести к снижению производительности и сбоям, вызванным тем, что клиентские приложения не могут установить соединение. В данном случае вы могли бы реструктурировать службу с целью использования очереди шины интеграции. Клиентские приложения будут отправлять сообщения в эту очередь, а служба будет обрабатывать их в определенное время. В данном случае очередь шины интеграции осуществляет балансировку нагрузки на получателя, не блокируя при этом отправителя.

- **Отправитель, посылающий запросы в очередь, ждет ответа.**

Очередь сообщений — это, по сути, механизм асинхронных односторонних коммуникаций. Если отправитель запроса ожидает ответа, получатель может отправить ответ в очередь, и отправитель получит его из этой очереди. Стоит заметить, что это простой механизм и простой сценарий — с одним отправителем, посылающим один запрос одному получателю, который затем отправляет один ответ, в реальной же ситуации будет несколько отправителей, получателей, запросов и ответов. В процессе реализации решения вы столкнетесь с двумя проблемами:

  - Как предотвратить получение ответа не тем отправителем?

- Если отправитель может посылать несколько запросов, каким образом он узнает, какой ответ соответствует каждому из них?

Первую проблему можно решить путем создания отдельной очереди для каждого отправителя. Все отправители будут помещать запросы в одну очередь, но ждать ответа каждый из них будет только из своей очереди. Все сообщения шины интеграции имеют набор свойств, которые вы можете использовать для включения метаданных. Отправитель может установить значение для свойства метаданных **ReplyTo** в запросе, указав, в какую очередь необходимо посылать ответы.

Все сообщения должны иметь уникальное значение **MessageId**, его устанавливает отправитель. Вторую проблему может решить отправитель, установив для свойства **CorrelationId** в сообщении-ответе значение, указанное для свойства **MessageId** в оригинальном запросе. Таким образом, отправитель будет определять соответствие запросов и ответов.

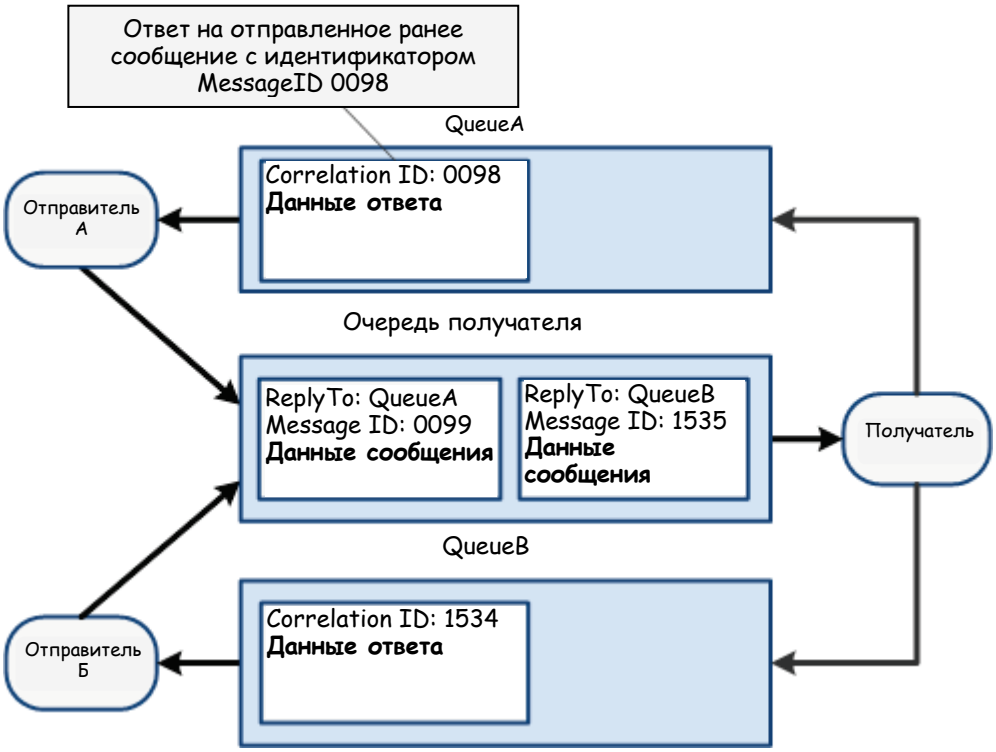



**Рисунок 12**

### Организация двунаправленного обмена сообщениями с очередями ответов и определением соответствия сообщений

- **Вам требуется надежный канал связи между отправителем и получателем.**

Подход, подразумевающий сопоставление сообщений, можно расширить, если вы хотите создать надежный канал связи на основе очередей шины интеграции. Очереди шины интеграции сами по себе надежны, но надежность сети и приложений, отправляющих и принимающих сообщения, не гарантирована. Поэтому, возможно, вам потребуется разработать не только логику для повторных попыток, помогающую обработать неустойчивые неисправности сети, которые могут возникнуть при отправке или получении сообщения, но и простой протокол сквозной передачи, регламентирующий взаимодействие между отправителем и получателем посредством сообщений с подтверждениями.

Простой пример: когда отправитель посылает сообщение в очередь, он может ждать подтверждения от получателя (с использованием асинхронной задачи или фонового потока). Свойство **CorrelationId** в сообщении с подтверждением должно соответствовать свойству

**MessageId** в оригинальном запросе. Если соответствующий ответ не был получен после определенного промежутка времени, то отправитель заново отправит сообщение и будет ждать. Этот процесс может повторяться до тех пор, пока или отправитель не получит подтверждение, или не будет достигнута пороговая величина, установленная для количества итераций. В последнем случае отправитель прекращает попытки и обрабатывает ситуацию как неудачную отправку сообщения.

Тем не менее вполне возможна ситуация, когда сообщение, созданное отправителем, было принято получателем, и он отправил подтверждение, но это подтверждение не дошло до отправителя. В таком случае отправитель может послать дубликат сообщения, которое получатель уже обработал. Чтобы предотвратить такое развитие ситуации, получатель должен вести список идентификаторов сообщений, которые он уже обработал. Если он принимает еще одно сообщение с тем же идентификатором, необходимо просто отправить подтверждение, не пытаясь выполнить повторную обработку.

**Примечание**

**Важно!** Не используйте функцию обнаружения дубликатов в очереди шины интеграции, чтобы исключить повторяющиеся сообщения в рамках такого сценария. Если вы активируете проверку дубликатов, повторный запрос или подтверждение могут быть удалены без уведомления, что приведет к сбою протокола сквозной передачи. Например, если получатель повторно посылает подтверждающее сообщение, функция обнаружения дубликатов может его удалить, что приведет к сбою в работе отправителя и, возможно, неустойчивому состоянию системы. Отправитель предполагает, что получатель не получил и не обработал сообщение, а получатель не знает, что работа отправителя была прервана.

- **Ваша система сталкивается с неожиданным увеличением количества отправленных сообщений и должна уметь оперативно справляться с непредсказуемым количеством сообщений.**

Очереди шины интеграции представляются целесообразным решением для балансировки нагрузки, они помогают предотвратить повышенную нагрузку на получателя в результате скачкообразного увеличения количества сообщений. Однако такой подход полезен только в ситуациях, когда отправляемые сообщения не чувствительны ко времени. В некоторых ситуациях сообщения должны быть обработаны в течение короткого промежутка времени. В таком случае можно подключить дополнительных получателей с целью прослушивания сообщений из той же очереди. Такая архитектура *нагрузочной способности по выходу (fan-out)* естественным образом балансирует нагрузку между получателями, поскольку каждый из них пытается извлечь сообщения из очереди, а семантика очередей позволяет предотвратить прием одного и того же сообщения двумя получателями. Процесс монитора может запрашивать длину очереди и динамически запускать и останавливать экземпляры получателя по мере ее увеличения или уменьшения.

**Мнение Бхарата**

В случае необходимости можно использовать функциональный блок для автоматического масштабирования из библиотеки Enterprise Library для контроля длины очереди шины интеграции, а также для запуска или остановки ролей, выступающих в качестве получателей.

Отправителей не придется каким-либо образом модифицировать, они будут помещать отправления в очередь привычным способом. Это решение подходит и для реализации двустороннего обмена сообщениями, как показано на рисунке 13.

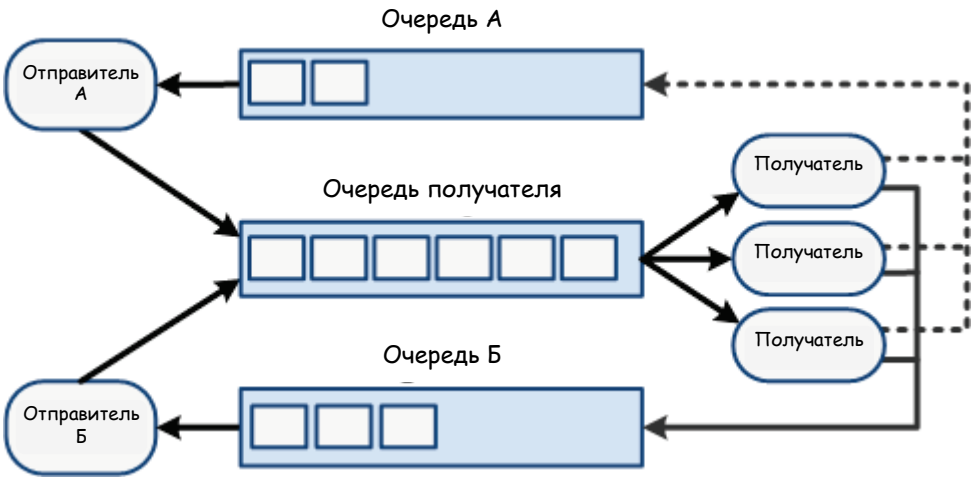



**Рисунок 13**

### Реализация технологии балансировки нагрузки с использованием нескольких получателей

> **Мнение По**
>
> Вы должны иметь в виду, что неожиданное резкое увеличение количества запросов может быть результатом атаки типа «отказ в обслуживании». Чтобы свести к минимуму возможность подобного рода атак, необходимо защищать очереди шины интеграции, которые использует ваше приложение, в целях предотвращения постановки в очередь сообщений от неавторизованных отправителей. Дополнительная информация представлена в разделе «Рекомендации по защите очередей шины интеграции» далее в настоящем приложении.

- **Отправитель посылает в очередь серию сообщений. Сообщения зависят друг от друга, их должен обрабатывать один и тот же получатель. Если несколько получателей прослушивают очередь, один и тот же получатель должен обрабатывать все сообщения в серии.**

В рамках данного сценария сообщение может содержать информацию о состоянии, формирующую контекст для группы следующих сообщений. Получателю, который обработал первое сообщение, возможно, придется обрабатывать и все последующие сообщения из этой серии.

В связи с этим отправителю, возможно, потребуется послать сообщение, размер которого превышает максимально допустимый (в настоящее время — 256 КБ). Чтобы решить эту проблему, отправитель может разбить данные на несколько сообщений меньшего размера. Эти сообщения должны быть приняты одним и тем же получателем, а затем собраны в исходное большое сообщение для последующей обработки.

Очереди шины обслуживания можно настроить на поддержку сеансов. Сеанс состоит из набора сообщений, которые представляют собой единый диалог. Все сообщения сеанса должны обрабатываться одним получателем. Чтобы активировать поддержку сеансов, нужно при создании очереди шины обслуживания установить значение «true» для свойства

**RequiresSession**. У всех сообщений, помещаемых в эту очередь, свойство **SessionId** должно иметь значение строкового типа. Значение, хранящееся в этой строке, идентифицирует сеанс, и все сообщения с одним и тем же значением свойства **SessionId** считаются частью одного сеанса. Обратите внимание, что несколько отправителей могут посылать сообщения с одним и тем же идентификатором сеанса, в этом случае все соответствующие сообщения рассматриваются как один сеанс.

Получатель, который приступает к обработке событий сеанса, вызывает метод **AcceptMessageSession** объекта **QueueClient**. Этот метод создает объект сеанса, и получатель может использовать его для извлечения сообщений из сеанса, как из обычной очереди. Метод **AcceptMessageSession**«прикрепляет» к получателю сообщения в очереди, которые составляют один сеанс, и скрывает их от всех других получателей. Другой получатель, вызывающий метод **AcceptMessageSession**, получит сообщения из следующего доступного сеанса. На рисунке 14 показаны два отправителя, которые посылают сообщения с различными идентификаторами сеанса, каждый отправитель генерирует свой собственный сеанс. Получатели каждого сеанса обрабатывают только те сообщения, которые относятся к этому сеансу.

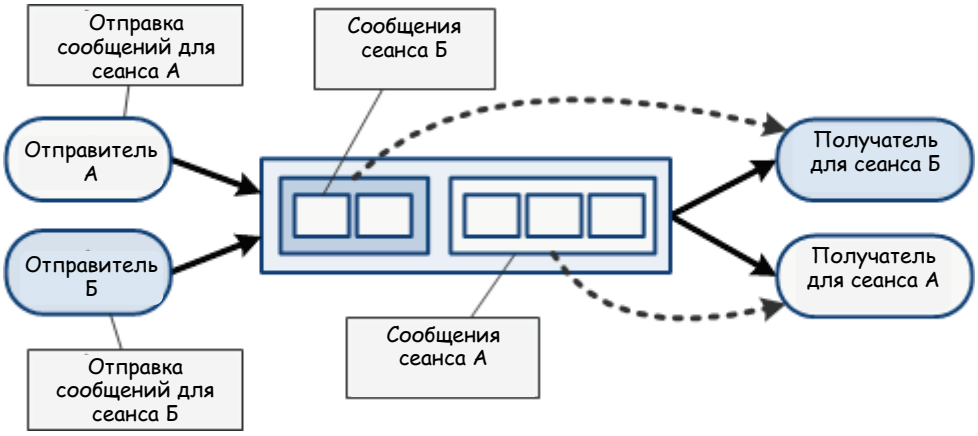



**Рисунок 14**

### Использование сеансов для группировки сообщений

Вы также можете организовать дуплексные сеансы, если получатель должен отправить набор сообщений в качестве ответа. Для этого в свойстве **ReplyToSessionId** ответного сообщения перед отправкой нужно установить значение, указанное в свойстве **SessionId** принятого сообщения. Отправитель может создавать свои собственные сеансы и при помощи идентификатора сеанса сопоставлять сообщения в сеансе ответов с оригинальными запросами.

Сеанс сообщений может включать в себя информацию о состоянии сеанса, которая сохраняется вместе с сообщениями в очереди. Вы можете использовать эту информацию для отслеживания операций, выполненных в рамках сеанса, а также для создания простого конечного автомата на стороне получателя сообщений. Когда получатель извлекает сообщение из очереди, он может сохранить информацию о выполненных операциях при обработке сообщения как информацию о состоянии сеанса и поместить эти данные обратно в очередь. Если на стороне получателя происходит сбой или он неожиданно завершает свою работу, другой экземпляр может подключиться к сеансу, получить информацию о его состоянии и продолжить обработку сообщений. Этот сценарий показан на рисунке 15.

**Мнение Маркуса**

Используйте методы **GetState** и **SetState** объекта **MessageSession**, чтобы извлечь и обновить информацию о состоянии сеанса сообщений.

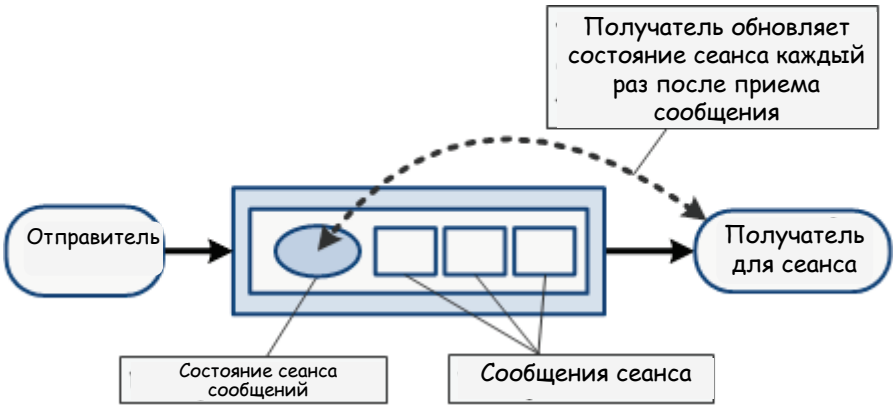



**Рисунок 15**

### Извлечение и сохранение информации о состоянии сеанса сообщений

Вполне возможно, что сеанс будет бесконечным, сообщения будут идти непрерывным потоком через различные (иногда достаточно продолжительные) промежутки времени. В таком случае получатель сообщений должен быть готов переходить в спящий режим на определенный срок. При получении нового сообщения в рамках сеанса, другой процесс, контролирующий систему, может инициировать выход получателя из спящего режима, чтобы тот продолжил обработку.

- **Отправитель должен послать одно или несколько сообщений в очередь единым блоком. Если некоторые операции завершились неудачно, то ни одно из сообщений не подлежит отправке, и все они должны быть удалены из очереди.**

Самый простой способ отправки нескольких сообщений единым блоком — с помощью локальной транзакции. Вы можете инициировать локальную транзакцию, создав объект **TransactionScope**. Это программная конструкция, которая определяет границы для множества задач, составляющих одну транзакцию.

Чтобы отправить группу сообщений в одной транзакции, необходимо инициировать операцию отправки для каждого сообщения в рамках одного и того же объекта **TransactionScope**. При этом сообщения просто помещаются в буфер и не отправляются до завершения транзакции. Для того чтобы все сообщения были отправлены, вы должны успешно завершить транзакцию. Если транзакция не завершена, ни одно сообщение не будет отправлено, и все они удаляются из очереди. Более подробная информация о классе **TransactionScope** представлена в статье «TransactionScope Class» на сайте MSDN: http://msdn.microsoft.com/en-us/library/system.transactions.transactionscope.aspx.

Если вы отправляете сообщения в асинхронном режиме (рекомендуемый подход), скорее всего, вам не потребуется использовать объект **TransactionScope**. Обратите внимание, что если вы используете операции из других транзакционных источников, таких как базы данных SQL Server, то эти операции не могут быть выполнены в рамках одного и того же объекта **TransactionScope**, поскольку диспетчер ресурсов, который упаковывает очереди шины интеграции, не может использовать операции совместно с другими диспетчерами ресурсов.

**Мнение Маркуса**

Если вы попытаетесь использовать объект **TransactionScope** для выполнения локальных транзакций, в которых задействованы очередь шины интеграции и другие диспетчеры ресурсов, ваше приложение выдаст исключение.

В подобных сценариях можно реализовать свой псевдотранзакционный механизм, основанный на ручном обнаружении ошибок, логике для повторных попыток и устранения дубликатов сообщения (*dedupe*) очереди шины интеграции.

Чтобы задействовать механизм устранения дубликатов, отправитель должен каждому сообщению, которое он помещает в очередь, присвоить уникальный идентификатор. Если два сообщения в одной очереди имеют одинаковые идентификаторы, они будут считаться идентичными, и логика устранения дубликатов удалит второе сообщение. Используя эту функцию в случае возникновения сбоя в работе своей бизнес-логики, приложение-отправитель может просто попытаться повторно отправить сообщение в процессе обработки отказа или повтора. Если у сообщения, которое было успешно послано ранее, появляются дубликаты, они будут удалены, и получатель увидит только первое сообщение. Такой подход гарантирует, что сообщение всегда будет отправлено, по крайней мере, один раз (при условии наличия у отправителя действующего подключения к очереди), но при этом будет достаточно сложно удалить сообщение, если алгоритм обработки сбоев в бизнес-логике определит, что это сообщение не подлежало отправке.

**Мнение Маркуса**

Функция обнаружения дубликатов активируется путем присваивания значения «true» свойству **RequiresDuplicateDetection** при создании очереди. Изменить значение этого свойства у существующей очереди невозможно. Кроме того, свойству **DuplicateDetectionHistoryTimeWindow** необходимо присвоить значение **TimeSpan**, таким образом указывается период, в течение которого дубликаты сообщений с данным идентификатором удаляются. Если сообщение с этим идентификатором появляется по истечении указанного периода, оно будет помещено в очередь на доставку.

- **Получатель извлекает одно или несколько сообщений из очереди, и снова в рамках транзакционной операции. Если завершить транзакцию не удалось, то все сообщения в очереди должны быть заменены, чтобы их снова можно было прочитать.**

Получатель может извлекать сообщения из очереди шины интеграции в одном из двух режимов: **ReceiveAndDelete** и **PeekLock**. В режиме **ReceiveAndDelete** сообщения удаляются из очереди сразу после прочтения. В режиме **PeekLock** сообщения из очереди не удаляются, но после прочтения они блокируются, чтобы предотвратить их извлечение другим параллельным получателем, который должен получить следующее доступное незаблокированное сообщение. Когда получатель успешно завершает обработку сообщения, он может вызвать его метод **Complete**, который удалит это сообщение из очереди. Если сообщение не удается обработать, получатель может вызвать метод **Abandon**, чтобы снять блокировку и оставить сообщение в очереди. Такой подход позволяет выполнять асинхронный прием сообщений.

**Мнение Маркуса**

Режим приема сообщений **ReceiveAndDelete** обеспечивает лучшую производительность, чем режим **PeekLock**, однако режим приема **PeekLock** гарантирует максимальную безопасность. В режиме **ReceiveAndDelete**, в случае возникновения сбоя в работе получателя после прочтения сообщения, это сообщение будет потеряно. В режиме **PeekLock** неудачное выполнение операции получения или обработки приведет к тому, что сообщение будет отменено и возвращено обратно в очередь, где его можно будут снова прочитать.

По умолчанию очередь шины интеграции работает в режиме приема **PeekLock**.

Как и в случае с отправкой сообщений, их прием в режиме **PeekLock** может выполняться синхронно в рамках локальной транзакции, поскольку транзакция не пытается задействовать возможности дополнительных диспетчеров ресурсов. Для этого необходимо создать объект **TransactionScope**. Если транзакцию не удалось завершить успешно, все полученные и прочитанные сообщения, имеющие отношение к объекту **TransactionScope**, будут возвращены в очередь.

**Мнение Маркуса**

Локальные транзакции поддерживаются только в режиме **PeekLock**, в режиме **ReceiveAndDelete** это невозможно.

- **Получатель должен проанализировать следующее сообщение в очереди, но извлекать его из очереди он должен только в том случае, если это сообщение предназначено для него.**

В данном случае получатель может извлечь сообщение из очереди в режиме **PeekLock** и скопировать его в локальный буфер с целью анализа. Если сообщение предназначено не ему, получатель может быстро вызвать метод **Abandon** и сделать это сообщение доступным для другого получателя.

Если сообщение содержит конфиденциальные сведения и должно быть прочитано только конкретным получателем, отправитель может зашифровать тело сообщения с помощью ключа, известного только авторизованному получателю. Отправитель может указать нужного получателя в свойстве **To** сообщения с помощью идентификатора. Свойства сообщения не зашифрованы, поэтому любой получатель может извлечь сообщение, но если он не распознает адрес в свойстве **To**, возможно, у него нет нужного ключа для расшифровки содержимого сообщения, поэтому сообщение будет отклонено и оставлено для соответствующего получателя.

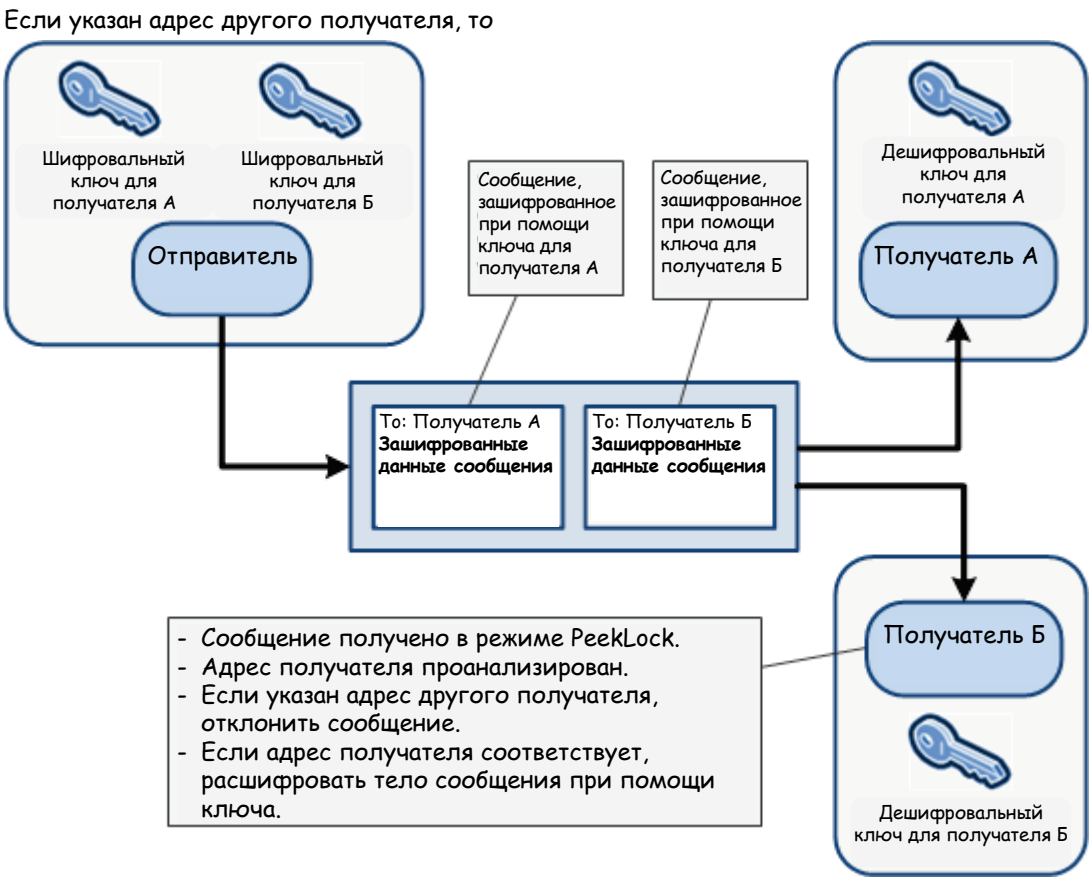



**Рисунок 16**

**Использование режима PeekLock и шифрования с целью анализа сообщений без извлечения их из очереди**

> **Мнение Джаны**
>
> В качестве альтернативного подхода рекомендуем рассмотреть возможность использования топика шины интеграции с отдельной подпиской для каждого получателя. Однако подписками будет достаточно сложно управлять при наличии большого количества получателей, или когда их количество постоянно меняется.

## Рекомендации по отправке и приему сообщений с использованием очередей шины интеграции

Вы можете разработать логику приложения, которое будет отправлять и принимать сообщения, используя несколько различных технологий.

- Можно использовать интерфейсы API очереди шины интеграции в Windows Azure SDK. Дополнительная информация и хорошие практические примеры для данного подхода приведены в статье «Best Practices for Leveraging Windows Azure Service Bus Brokered Messaging API» на портале MSDN: http://msdn.microsoft.com/en-us/library/hh545245(v=VS.103).aspx.
- Вы можете использовать привязки очереди шины интеграции для подключения к очереди из клиентских приложений и служб WCF. Дополнительная информация представлена

в статье «How to: Build an Application with WCF and Service Bus Queues» на сайте http://msdn.microsoft.com/en-us/library/windowsazure/hh243674.aspx.

- Если вы создаете приложения, которые подключаются к очереди шины интеграции с использованием технологии, которая не поддерживает Windows Azure SDK, вы можете использовать интерфейс HTTP REST, предоставляемый шиной интеграции.

### *Асинхронная отправка и прием сообщений*

При использовании Windows Azure SDK вы можете создавать приложения, которые будут отправлять и принимать сообщения при помощи классов **MessageSender** и **MessageReceiver** в пространстве имен **Microsoft.ServiceBus.Messaging**. Эти типы предоставляют доступ к операциям по обмену сообщениями, которые рассматривались ранее в настоящем приложении. Основные функции для отправки и получения сообщений вызываются через методы **Send** и **Receive** данных типов. Однако эти операции являются синхронными. Например, метод **Send** класса **MessageSender** ожидает завершения операции отправки, аналогичным образом метод **Receive** класса **MessageReceiver** ждет, пока сообщение не будет доступно или пока не истечет указанный период. Помните, что эти методы в действительности представляют собой только фасады для серии запросов HTTP REST, а также что очередь шины интеграции является удаленной службой, доступ к которой осуществляется через Интернет. Кроме того, ваше сообщение должно учитывать следующее:

- Операции отправки и получения могут выполняться сколь угодно долго, и ваше приложение не должно блокировать процесс ожидания их завершения.

Класс **MessageSender** предоставляет доступ к асинхронной версии метода **Send**, а класс **MessageReceiver** — к асинхронной реализации метода **Receive** через методы **BeginSend/EndSend** и **BeginReceive/EndReceive** соответственно. Эти методы лучше использовать вместо их синхронных аналогов. Эти методы соответствуют стандартной асинхронной модели, реализованной средствами платформы .NET Framework.

Аналогичные вопросы возникают в связи с другими операциями, такими как определение существования очереди, создание и удаление очередей, подключение к очереди и запрос длины очереди. Поэтому вы должны выполнять эти операции в соответствии с тем же надежным асинхронным подходом.

> Вы также можете использовать класс **QueueClient** в пространстве имен **Microsoft.ServiceBus.Messaging** для подключения к очереди, отправки и получения сообщений. Тип **QueueClient** — это абстрактный класс, который реализует расширенный набор функций, доступных в классах **MessageSender** и **MessageReceiver**. Пакет Windows Azure SDK предоставляет дополнительные типы для отправки сообщений в топики (**TopicClient**) и получения сообщений из подписок (**SubscriptionClient**). Однако классы **MessageSender и MessageReceiver** абстрагируют различия между этими типами. Например, если вы используете **MessageSender** для отправки и получения сообщений при помощи очередей, вы можете перейти к использованию топиков, и при этом потребуется минимальная доработка кода. Аналогичным образом объект **MessageReceiver** позволяет извлекать сообщения из очереди и подписки при помощи одного и того же кода.
>
> Однако прежде чем изменять весь существующий код, чтобы использовать объекты **MessageSender** и **MessageReceiver**, учтите, что не все функции, реализованные при помощи типов **QueueClient**, **TopicClient** и **SubscriptionClient**, доступны в классах

**MessageSender** и **MessageReceiver**. Например, класс **MessageReceiver** не поддерживает сеансы.

- Отправитель может публиковать сообщения в любое время, а получателю необходимо ожидать сообщения в более чем одной очереди.

Есть несколько возможных решений этой проблемы, но общепринятые подходы связаны с использованием потоков или задач для ожидания сообщений из каждой очереди, и когда сообщение появляется, система генерирует событие. Приложение, которое получает это событие, сможет затем обработать сообщение. Например, вы можете определить метод **async**, который использует оператор **await**, предоставляемый языком Visual C#, чтобы создать серию задач, которые будут ждать сообщений из всех очередей, и в случае их появления генерировать события. Затем вы можете использовать какой-либо фреймворк, например Microsoft Reactive Extensions, чтобы получать эти события и обрабатывать сообщения по мере их появления.

Если получатель должен обрабатывать несколько сообщений, составляющих бизнес-операцию, вы можете оптимизировать процесс получения с помощью функции упреждающей загрузки в классе **QueueClient**.

По умолчанию объект **QueueClient** вызывает метод **Receive**, при этом из очереди извлекается только следующее доступное сообщение. Тем не менее вы можете задать свойству **PrefetchCount** объекта **QueueClient** положительное целочисленное значение, и метод **Receive** извлечет указанное количество сообщений из очереди (если они имеются в наличии). Сообщения помещаются в локальный буфер приложения-получателя, после чего они больше не доступны для других получателей. Метод **Receive** затем возвращает первое сообщение из своего буфера. Последующие вызовы метода **Receive** позволяют получить из буфера все оставшиеся сообщения, когда буфер опустошается, операция **Receive** извлекает следующую группу сообщений из очереди в буфер. Такой подход позволяет эффективнее использовать пропускную способность сети за счет увеличения промежутка времени, необходимого для получения первого сообщения. Тем не менее все последующие сообщения будут получены значительно быстрее.

На сообщения, подлежащие предварительной загрузке, распространяется та же семантика времени ожидания, которая действует для небуферизованных сообщений. Если сообщения не будут обработаны в течение периода ожидания, который начинается с момента получения группы сообщений из очереди, то предполагается, что в приложении-получателе произошел сбой, и сообщения будут возвращены в очередь. Поэтому, если вы используете функцию упреждающей загрузки, то должны помещать в буфер такое количество сообщений, которое получатель может принять и обработать в течение установленного периода.

- Операции отправки и получения могут потерпеть неудачу в силу целого ряда причин, например может отсутствовать связь между вашим приложением и шиной интеграции в облаке, может возникнуть нарушение правил безопасности вследствие внесения изменений в соответствующие настройки шины интеграции (администратор может принять решение об отзыве или изменении прав учетной записи по каким-либо причинам), очередь может быть заполнена (очереди имеют конечный размер) и так далее. Некоторые из этих сбоев могут носить характер неустойчивых неисправностей, в то время как другие могут быть очень продолжительными. Асинхронные операции отправки и приема сообщений

должны обрабатывать ситуацию отмены, чтобы любые фоновые потоки можно было надлежащим образом остановить, освободив ресурсы, используемые в процессе обмена сообщениями.

После извлечения сообщения из очереди его можно прочитать, воспользовавшись методом **GetBody** класса **BrokeredMessage**. Этот метод выполняет десериализацию тела сообщения. Десериализация данных осуществляется только один раз. Этот важный фактор необходимо учитывать в процессе работы над логикой для обработки ошибок. Вам не удастся вызвать метод **GetBody** еще раз в отношении одного и того же сообщения (например, внутри обработчика исключений). Поэтому, если вам потребуется повторный доступ к данным в теле сообщения, вы должны сохранить эти данные в подходящий объект и затем использовать его.

### *Планирование, истечение срока действия и отсрочка сообщений*

По умолчанию, когда отправитель посылает сообщение в очередь, оно сразу же становится доступным для извлечения и обработки получателем. Тем не менее вы можете сделать так, чтобы сообщение оставалось невидимым в течение определенного периода после его появления в очереди. Этот метод полезен для планирования сообщений, которые необходимо обработать через некоторое время, например, если данные должны быть опубликованы только после полуночи. Чтобы определить, когда сообщение должно появиться в очереди и стать доступным для обработки, установите свойство **ScheduledEnqueueTimeUtc** объекта **BrokeredMessage**.

Когда отправитель посылает сообщение в очередь, оно может оставаться там в течение длительного периода времени, прежде чем получатель его примет. Сообщение может иметь срок действия, по истечении которого информация в сообщении теряет актуальность. В этом случае, если сообщение не было получено, оно должно без уведомления удаляться из очереди. Для этого необходимо настроить свойство **TimeToLive** объекта **BrokeredMessage**, когда отправитель посылает сообщение.

В некоторых случаях приложению не требуется обрабатывать следующее из доступных сообщений, оно может пропустить это сообщение, получить следующее и затем вернуться к пропущенному сообщению. Для этого нужно отложить сообщение, воспользовавшись методом **Defer** класса **BrokeredMessage**. Этот механизм можно использовать в том случае, когда приложение получает сообщения в режиме **PeekLock**. Метод **Defer** оставляет сообщение в очереди, но блокирует его, делая недоступным для других получателей. В назначенное время приложение может вернуться к отложенному сообщению, выполнить его обработку, а затем вызвать метод **Complete** или **Abandon**, как было описано ранее в настоящем приложении. В случае, если сообщение уже не актуально или не действительно во время его обработки, приложение может пометить его как недоставленное. Обратите внимание, что если происходит сбой в работе приложения, блокировка снимается, и сообщение становится доступным в очереди. Вы можете регулировать продолжительность блокировки при помощи свойства **LockDuration** в момент создания очереди.

### Рекомендации по защите очередей шины интеграции

Очереди шины интеграции предоставляют инфраструктуру обмена сообщениями для бизнес-приложений. Они создаются и находятся под управлением платформы Windows Azure в облаке. Поэтому они надежны и устойчивы. Как только отправитель поместил сообщение в очередь, оно будет оставаться там, пока не будет получено, или пока не истечет срок его действия.

Очередь шины интеграции хранится в пространстве имен шины интеграции с уникальным идентификатором URI. Вы создаете этот идентификатор при создании пространства имен, структура URI аналогична описанной в разделе «Модель безопасности службы Windows Azure Service Bus Relay» ранее в данном приложении. Используя данный идентификатор URI, приложение создает экземпляр объекта

**MessagingFactory**. Объект **MessagingFactory** затем можно использовать для создания объекта **MessageSender** или **MessageReceiver**, который подключается к очереди.

Пространство имен шины интеграции формирует контекст для защиты очереди, доступ к пространству имен, в котором размещена ваша очередь, следует предоставлять только прошедшим аутентификацию отправителям и получателям. Вы можете защитить пространство имен с помощью службы ACS способом, подобным описанному в разделе «Рекомендации по защите службы Windows Azure Service Bus Relay» ранее в данном приложении, за исключением того, что домашняя область приложения доверенного участника определяется идентификатором URI пространства имен шины интеграции, к которому добавляется имя очереди, топика или подписки шины интеграции (например, http://treyresearch.servicebus.windows.net/orderstatusupdatequeue), а не адрес службы WCF.

Вы можете создать группу правил ACS для данного URI и присвоить типу утверждений **net.windows.servicebus.action** значения **Send**, **Listen** и **Manage** с целью надлежащей аутентификации удостоверений. Обратите внимание, что утверждения **Send** и **Listen** предоставляют минимальный набор привилегий, позволяя приложению отправлять сообщения в очередь и получать сообщения из нее, а также выполнять некоторые другие операции. Если ваше приложение должно выполнять такие задачи, как создание новой очереди, запрос количества сообщений, размещенных в это время в очереди, или даже просто определение наличия очереди с заданным именем. Приложение должно работать с удостоверением, которому предоставлены права, связанные с утверждением **Manage**.

Все коммуникации с очередью шины интеграции происходят через канал TCP, зашифрованный с помощью SSL. Если вы хотите обеспечить дополнительную защиту на уровне сообщений, вы должны шифровать их содержимое, при этом получателю необходимо предоставить ключ дешифровки. Таким образом, если сообщение каким-то образом будет перехвачено, его содержимое будет невозможно просмотреть. Аналогичным образом, если авторизованный получатель сообщения не может его расшифровать, это сообщение должно рассматриваться как потенциально опасное, и его нужно переместить в очередь недоставленных сообщений.

**Примечание**

Также можно реализовать механизм проверки подлинности отправителя, помещающего сообщения в очередь шины интеграции, добавив маркер удостоверения в заголовок сообщения. Если этот маркер отсутствует или не может быть распознан приложением-получателем, сообщение должно рассматриваться как подозрительное. Пример реализации такого подхода представлен в разделе «Защита сообщений» в главе 4 «Реализация надежного обмена сообщениями и информацией в облаке».

# Приложение Г. Реализация бизнес-логики и маршрутизации сообщений с целью организации совместной работы

Простая и надежная стратегия позволяет организовать безопасную двухточечную связь между компонентами, задействованными в распределенной системе. Конкретная реализация определяет степень независимости отправителей и получателей друг от друга, но системе все-таки требуются средства физической отправки сообщений в пункт назначения. Во многих решениях этот механизм может быть встроен в логику приложения. Отправитель, использующий коммуникации в стиле RPC для взаимодействия со службой Windows Communication Foundation (WCF), может указать адрес пункта назначения либо приложение, обеспечивающее взаимодействие на основе протокола сообщений, может направлять сообщения в очередь, которую прослушивает конкретный получатель. Такой непрозрачный подход затрудняет задачу внесения изменений в способ маршрутизации сообщений, если добавляются новые или перемещаются существующие точки назначения. Без серьезной модификации базовой бизнес-логики сделать это будет невозможно.

Разделение потока данных и бизнес-логики вашего приложения предоставляет многочисленные преимущества. Например, следуя этой стратегии, вы можете прозрачно масштабировать свое решение, подключая дополнительные экземпляры службы к обработке сообщений в периоды пиковой нагрузки. Вы можете контролировать и проверять сообщения, не прерывая их пересылку от отправителя к получателю, также можете легко интегрировать дополнительные службы в свое решение путем расширения списка получателей сообщений.

В настоящем приложении рассматриваются некоторые общие проблемы, связанные с управлением и контролем над потоком сообщений, здесь также представлены возможные решения и передовые практики для отделения этого потока данных от логики приложения при помощи технологической платформы Windows Azure.

## Сценарии использования и проблемы

Многие гибридные приложения должны обрабатывать бизнес-правила и рабочие процессы, которые содержат проверки условий, при этом выполняются различные действия в зависимости от результатов проверки этих правил. Например, приложению могут потребоваться возможности для обновления базы данных с информацией об уровне запасов, отправки заявки соответствующему партнеру, который предоставляет транспортные услуги или обеспечивает складское обслуживание, а также для аудита состава заказа (например, проверки кредитного лимита клиента) и сохранения заказа в другой базе данных для бухгалтерского учета. Эти операции могут включать сервисы и ресурсы, размещенные как в облаке, так и локально. В процессе создания расширяемых решений на основе этих сценариев

использования, вам, скорее всего, придется решать специфические проблемы, описанные в следующих разделах.

**Мнение Джаны**

Если поток данных спроектирован таким образом, чтобы обеспечивалась независимость от конкретной реализации логики приложения, ваше решение становится более гибким и расширяемым. Это решение сможет оперативно реагировать на изменяющие потребности бизнеса и масштабироваться по мере интенсификации потока данных, поступающих к вашим службам.

## Разделение бизнес-логики и маршрутизации сообщений

**Описание: в ходе обработки бизнес-задач приложению необходимо отправлять и получать сообщения от других служб, которые могут быть расположены где угодно.**

Бизнес-операции, выполняемые распределенными приложениями, в первую очередь, связаны со сбором, анализом и обработкой данных. В процессе сбора и обработки данных, возможно, придется отправлять запросы и передавать сообщения другим службам. Тем не менее базовая бизнес-логика не должна быть привязана к местоположению этих служб. Если служба перемещается, вам не придется вносить изменения в логику работы приложения, если оно выполняет те же бизнес-функции. Разделение бизнес-логики и маршрутизации сообщений помогает достичь такой независимости от местоположения.

Разделение бизнес-логики и маршрутизации сообщений также позволяет секционировать сообщения с учетом заданных администратором критериев, а затем направлять эти сообщения наиболее подходящему экземпляру службы. Эти критерии не зависят от приложения. Например, чтобы уменьшить время отклика, администратор может принять решение о развертывании нескольких экземпляров службы, которая обрабатывает запросы на получение ипотечного кредита в облаке. Приложение, которое выполняется на рабочем столе ипотечного брокера, может размещать заявки на получение ипотечного кредита. Эти заявки могут быть прозрачно переданы конкретному экземпляру службы, которая занимается их обработкой, с учетом какого-либо атрибута каждого ипотечного брокера, например присвоенного ему номера. Запросы от брокеров с номерами с 1 до 30 могут отправляться одному экземпляру службы, а от брокеров с номерами с 31 до 60 — другому, и так далее. По мере подключения к системе новых офисов, разворачивания в них приложений для обработки ипотечных кредитов и увеличения числа ипотечных брокеров, администратор может отслеживать распределение нагрузки на службы, а при необходимости запустить дополнительные экземпляры службы и настроить систему таким образом, чтобы нагрузка по обработке запросов на ипотечные кредиты распределялась между этими экземплярами более равномерно.

Вы должны определиться с местоположением логики, которая управляет маршрутизацией сообщений. Если вы хотите полностью отделить этот поток от бизнес-логики приложения, маршрутизацией не должны управлять алгоритмы, которые отправляют сообщения, а также она не должна быть встроена в элементы, которые обеспечивают получение и обработку сообщений. Логика для организации потока данных должна содержаться внутри компонентов промежуточного ПО, которые отвечают за взаимодействие отправителей и получателей.

Эти компоненты промежуточного ПО эффективны в качестве посредников. Они направляют сообщения получателям в соответствии с хорошо продуманными алгоритмами. Подход, который вы применяете с целью создания такого посредника, должен обеспечивать настраиваемую логику маршрутизации, которая не зависит от компонентов, отправляющих и получающих сообщений. Эта логика должна

обеспечивать устойчивость и совместимость с любым подходом к организации надежного обмена сообщениями.

### Передача сообщений нескольким получателям

**Описание: в ходе обработки бизнес-задач приложению необходимо отправлять одно и то же сообщение любому количеству служб, перечень этих служб может со временем изменяться.**

Часто требуется передать одно и то же сообщение различным получателям, например службе доставки и службе складского учета в системе обработки заказов. Если вам удастся организовать прозрачную маршрутизацию сообщения нескольким получателям, вы сможете создавать расширяемые решения. Если в ваше решение потребуется включить новых партнеров или новые службы, например службу аудита, вы можете просто добавить новые пункты назначения, не изменяя приложения, отправляющие туда сообщения.

Этот сценарий использования предполагает, что компоненты промежуточного ПО, выполняющие маршрутизацию сообщений, могут прозрачно скопировать их и отправить в соответствующие пункты назначения. Это, в свою очередь, означает, что компоненты промежуточного ПО должны быть настраиваемыми в целях поддержки такого дублирования сообщений без возникновения каких-либо зависимостей от логики отправки или получения сообщений.

## Взаимосвязанные проблемы

Маршрутизация сильно зависит от основной платформы обмена сообщениями, и взаимосвязанные проблемы в сфере маршрутизации с большой долей вероятности будут представлять собой расширенный набор задач, которые вы должны решить в ходе реализации концепции «коммуникации без границ». Решению этих проблем посвящены следующие разделы.

### Безопасность

Инфраструктура обмена сообщениями должна быть надежной и защищенной, она не должна позволять неавторизованным приложениям и службам отправлять или получать сообщения.

Сообщения должны быть защищены и доступны только соответствующим адресатам. Это ограничение также распространяется на технологию маршрутизации, которую вы используете. Компоненты промежуточного ПО, реализующие роль маршрутизатора, должны быть в состоянии принимать решения о том, куда следует отправить сообщение, не изучая его содержимое. Если сообщения конфиденциальные, вы должны шифровать их содержимое, не затрудняя при этом маршрутизацию.

Если ваше решение копирует и отправляет сообщения нескольким адресатам, дублирование сообщений должно осуществляться контролируемым и безопасным образом, также без вовлечения в процесс компонентов промежуточного ПО, которым может потребоваться прямой доступ к данным в теле сообщения.

### Надежность

Базовая технология обмена сообщениями должна предотвращать потерю передаваемых сообщений, а механизм маршрутизации должен гарантировать доставку сообщений соответствующим адресатам.

### Время отклика и доступность

Платформа обмена сообщениями не должна препятствовать потоку бизнес-логики вашей системы. Если бизнес-логика отправляет сообщения, которые не могут быть доставлены немедленно из-за каких-либо временных проблем в инфраструктуре (например, из-за сбоя в работе сети, соединяющей платформу

обмена сообщениями и целевую службу), бизнес-логика не должна прерывать свою работу. Сообщения должны быть доставлены прозрачным образом, когда проблема будет устранена.

### Совместимость

Возможно, вам придется организовать маршрутизацию сообщений между службами, созданными с помощью различных технологий. Механизм маршрутизации должен быть совместим с технологиями, на которых базируются эти службы.

## Технологии Windows Azure для маршрутизации сообщений

Основная технология Windows Azure, которая обеспечивает безопасный и надежный обмен сообщениями между распределенными службами и приложениями с минимальным временем отклика и максимальной совместимостью,— это очереди шины интеграции.

Очередь шины интеграции — это простая структура, работающая по принципу «первым пришел — первым вышел», в которой также реализованы дополнительные функции, такие как поддержка тайм-аутов, транзакций и очередей недоставленных сообщений. Очередь шины интеграции обеспечивает реализацию различных распространенных сценариев обмена сообщениями, как описано в «Приложении B. Реализация подхода "коммуникации без границ"». Windows Azure расширяет возможности очередей шины интеграции при помощи топиков и подписок. Эти расширения позволяют реализовать ряд дополнительных вариантов маршрутизации сообщений в ваших решениях, воспользовавшись преимуществами, которые обеспечивает шина интеграции,— высокой безопасностью и надежностью.

В следующих разделах более подробно рассматриваются вопросы использования топиков и подписок шины интеграции для маршрутизации сообщений.

### Разделение бизнес-логики и маршрутизации сообщений с использованием топиков и подписок шины интеграции

Топики и подписки шины интеграции позволяют направлять сообщения различным получателям с учетом определяемых приложением критериев. Они позволяют воспользоваться преимуществами, предоставляемыми очередями шины интеграции, упрощая задачу обеспечения независимости отправителя сообщения от его получателя. Кроме того, они позволяют отправлять сообщения одному или нескольким получателям, используя информацию, хранящуюся в области метаданных этих сообщений.

Приложение-отправитель помещает сообщение в топик шины интеграции, пользуясь теми же методиками, как и при отсылке его в очередь шины интеграции. Но при этом отправитель обычно добавляет одно или несколько пользовательских свойств в метаданные сообщения, и эта информация используется для маршрутизации сообщения наиболее подходящему получателю. Топик шины интеграции представляет сторону отправителя в очереди, а подписка шины интеграции — сторону получателя. Приложение-получатель ожидает входящих сообщений, подключившись к подписке шины интеграции. Топик шины интеграции может быть связан с несколькими подписками шины интеграции.

Сообщение направляется от топика шины интеграции к подписке шины интеграции, при этом специализированный фильтр анализирует метаданные и пользовательские свойства сообщения. Фильтр представляет собой предикат для подписки шины интеграции, и все сообщения, которые соответствуют предикату, будут направлены этой подписке шины интеграции. Фильтры позволяют создать простые правила маршрутизации сообщений, в противном случае вам пришлось бы написать достаточно большой объем кода.

У всех подписок есть соответствующий фильтр. Если вы при создании подписки не указали, какой фильтр следует использовать, то по умолчанию фильтр просто передает все сообщения из топика в подписку.

### Рекомендации по использованию топиков и подписок шины интеграции для маршрутизации сообщений

Топики и подписки шины интеграции предназначены для реализации простого механизма статической маршрутизации сообщений от отправителя к получателю. Фильтры, направляющие сообщения из топика в подписку, отделены от бизнес-логики приложений-отправителей и приложений-получателей. Отправителю достаточно предоставить метаданные (в форме свойств сообщения и значений), которые фильтры могут анализировать и использовать для принятия решения по маршрутизации. На рисунке 1 показан поток сообщений от отправителя соответствующим получателям (используются топик и две подписки). Данные представляют собой отправляемые пакеты, сообщения фильтруются по свойству **Weight**, добавленному отправителем. Получатель А принимает все сообщения, у которых значение свойства **Weight** меньше 100. Получатель Б принимает сообщения, «вес» которых от 100 до 199. Получателю В отправляются сообщения со значениями «веса» от 200 и выше.

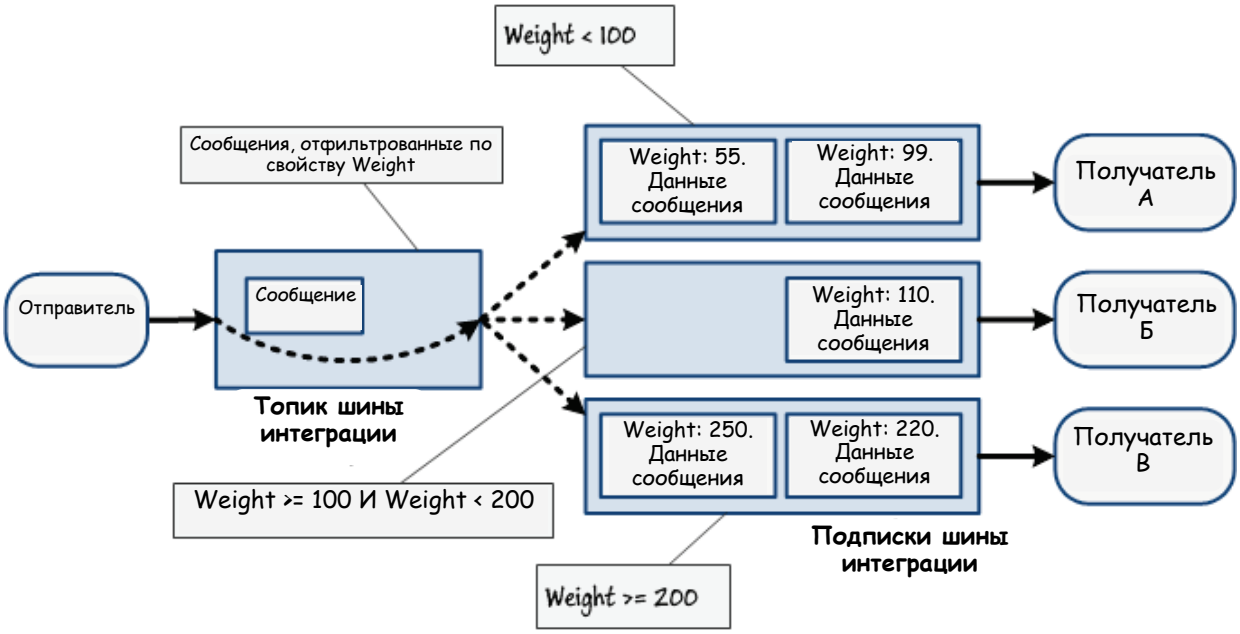



**Рисунок 1**

### Маршрутизация сообщений к различным получателям при помощи топиков и подписок шины интеграции

Топики и подписки шины интеграции предоставляют доступ к программной модели через набор интерфейсов API в Windows Azure SDK. Как и очереди шины интеграции, эти интерфейсы представляют собой оболочки для набора интерфейсов REST, поэтому вы можете использовать топики и подписки, основанные на технологиях и платформах, которые не поддерживаются непосредственно в Windows Azure SDK.

Топики и подписки шины интеграции позволяют вам реализовать несколько сценариев, которые сложно или невозможно воплотить при помощи очередей шины интеграции:

- Ваша система генерирует несколько сообщений, каждое из которых должно быть обработано определенным получателем. Новые получатели могут быть со временем добавлены, но в целях упрощения обслуживания вы не хотите вносить изменения в бизнес-логику приложения-

отправителя, когда это произойдет. Это основной сценарий, в рамках которого лучше использовать топики и подписки шины интеграции, а не очереди.

В качестве примера рассмотрим систему обработки заказов, позволяющую клиентам размещать заказы с помощью веб-приложения. Эти заказы должны быть отфильтрованы и отправлены клиентам. Компания работает с несколькими транспортными компаниями, конкретный партнер выбирается в зависимости от места расположения заказчика. Для веб-приложения безразлично, какая транспортная компания используется, оно просто включает в сообщение соответствующую информацию для каждого заказа и направляет его в топик шины интеграции вместе с метаданными, указывающими на место расположения клиента. Иногда могут добавляться новые транспортные компании или удаляться существующие, при этом бизнес-логика веб-приложения для обработки заказов не должна меняться, когда это произойдет. Для каждого транспортного партнера создана собственная подписка на этот топик шины интеграции, при этом используется фильтр, который анализирует метаданные с информацией о месте расположения клиента, затем сообщение направляется в соответствующую подписку.

**Мнение Маркуса**

Вы должны отделить логику, которая управляет маршрутизацией сообщений, от основной бизнес-логики приложения. Таким образом, в случае изменения алгоритма маршрутизации, бизнес-логику приложения не придется обновлять.

Если ваше приложение взаимодействует со сторонними организациями, оно должно обмениваться данными с системами, используемыми партнерами, причем эти системы с большой долей вероятности не будут основываться на подписках шины интеграции. Например, международный поставщик коммерческого транспорта, как правило, будет использовать свои собственные системы, основанные на открытых веб-службах, к которым необходимо обращаться в целях взаимодействия с их системами. Поэтому, возможно, вам придется создать несколько адаптеров, которые получают сообщения для каждого партнера из соответствующей подписки шины интеграции, переводят сообщения в необходимый партнеру формат, а затем связываются с партнером, используя его собственный интерфейс веб-службы. Это *модель продвижения* для обмена сообщениями, причем реализация этих адаптеров лежит в зоне вашей ответственности. К подпискам шины интеграции, которые эти адаптеры используют, не нужно предоставлять доступ за пределами вашей организации, задачи аутентификации и авторизации могут решаться непосредственно с помощью системы безопасности шины интеграции и службы Windows Azure Access Control Service (ACS).

Кроме того, если партнер не публикует программный интерфейс для взаимодействия с его системами, но готов разместить у себя логику для подключения к подписке шины интеграции, он может напрямую подключаться к подписке шины интеграции, получать сообщения, переводить их в формат, совместимый со своими внутренними процессами, а затем вызывать эти процессы. Эта модель подразумевает *«извлечение»* сообщений. Логика для взаимодействия с шиной интеграции может быть вынесена в отдельный компонент-соединитель, что обеспечит простоту и удобство обслуживания, если механизм, используемый для взаимодействия с вашим приложением, будет изменяться в будущем.

Вы должны предоставить транспортному партнеру доступ к конечной точке подписки шины интеграции. Может потребоваться внедрение федеративной системы безопасности для шины интеграции и собственного домена безопасности транспортного партнера.

На рисунке 2 показана архитектура возможного решения, организовано взаимодействие с тремя поставщиками коммерческого транспорта. Партнеры по регионам A и B предоставляют

функциональные возможности в виде набора веб-служб, поэтому для связи с ними необходимо использовать адаптеры. Транспортный партнер по региону Б не публикует открытый интерфейс для своих служб, но вместо этого внедряет свою собственную логику для коннектора с целью извлечения сообщений из подписки шины интеграции.

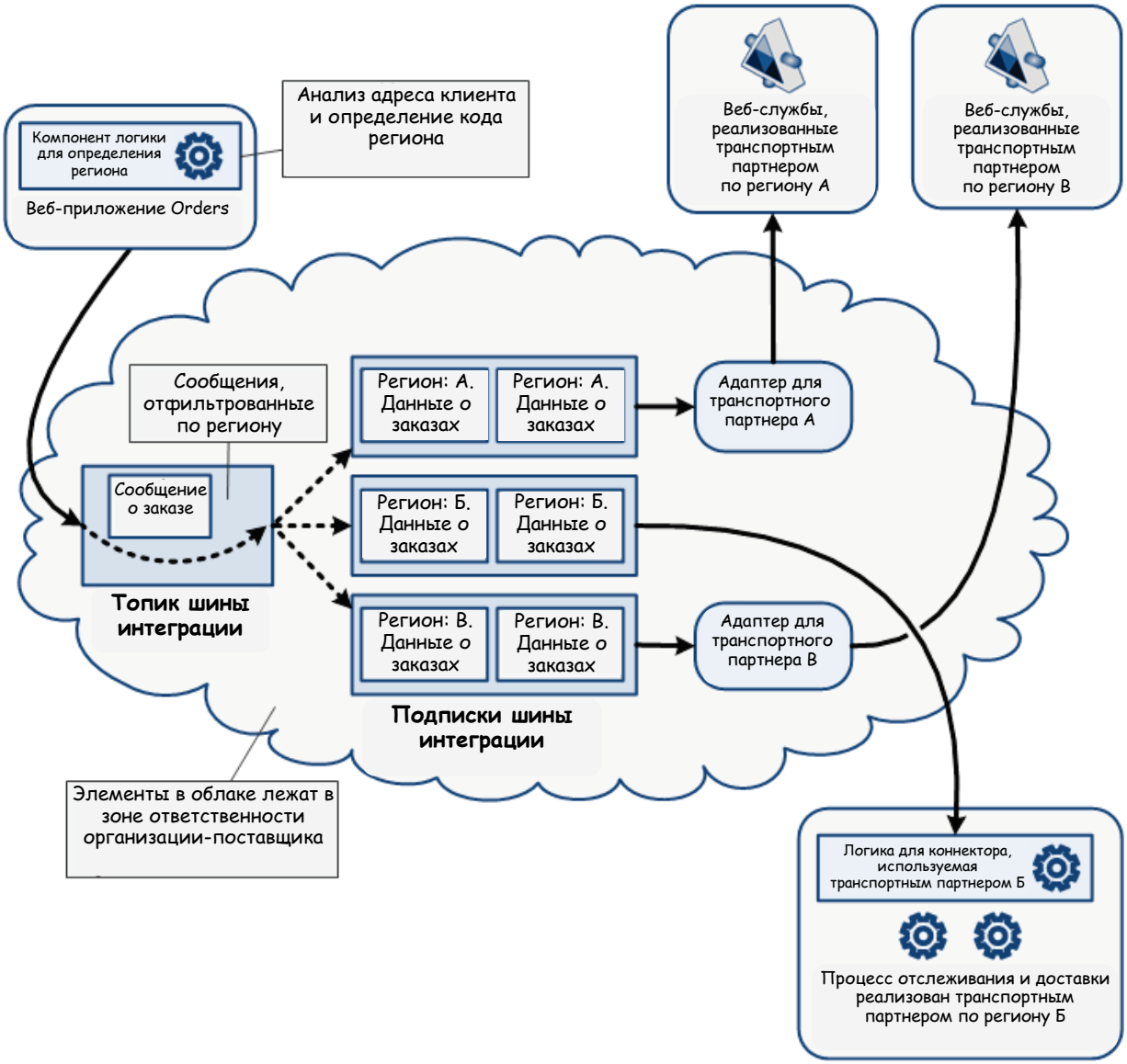



**Рисунок 2**

**Разделение приложения-отправителя и логики маршрутизации сообщений при помощи топика и подписок шины интеграции**

- Ваша система генерирует различные сообщения. Некоторые из этих сообщений имеют высокий приоритет и должны быть обработаны как можно быстрее, другие можно обработать некоторое время спустя, при этом еще больше сообщений имеют ограниченный срок действия, и если они не обрабатываются в течение указанного срока, их необходимо просто игнорировать.

Топики и подписки шины интеграции предоставляют механизм, который позволяет создать *очередь приоритетов*. Когда отправитель посылает сообщение, он может пометить его с помощью свойства **Priority**, а создаваемые подписки могут фильтровать сообщения по этому свойству.

Срочные сообщения могут быть обработаны подпиской с пулом получателей сообщений. Вы можете контролировать длину очереди этой подписки, и если длина превышает некоторое предопределенное значение, вы можете активировать нового получателя. Эта техника напоминает архитектуру *нагрузочной способности по выходу (fan-out)* для очередей, обрабатывающих неожиданный наплыв сообщений. Эта архитектура рассматривается в разделе «Рекомендации по использованию очередей шины интеграции» в «Приложении B. Реализация подхода "коммуникации без границ"».

Сообщения с более низким приоритетом могут быть обработаны подпиской с фиксированным количеством получателей с использованием системы *балансировки нагрузки*, которая описана в приложении B.

Сообщения с ограниченным сроком действия могут быть обработаны с помощью подписки со значением типа short для свойства **DefaultMessageTimeToLive**. Кроме того, если по истечении срока действия сообщения не требуется его дальнейшее отслеживание, вы можете установить для свойства **EnableDeadLetteringOnMessageExpiration** подписки значение false. При такой конфигурации, если срок действия сообщения истекает до его получения, оно будет автоматически проигнорировано.

На рисунке 3 показана структура этой системы. В данном примере сообщения с приоритетом **Critical** необходимо обрабатывать немедленно, сообщения с приоритетом **Important** должны быть обработаны как можно быстрее (в идеале — в ближайшие 10 минут), сообщения, помеченные как **Information**, не срочные, но если они не будут обработаны в течение следующих 20 минут, содержащиеся в них данные станут иррелевантными и могут быть проигнорированы.

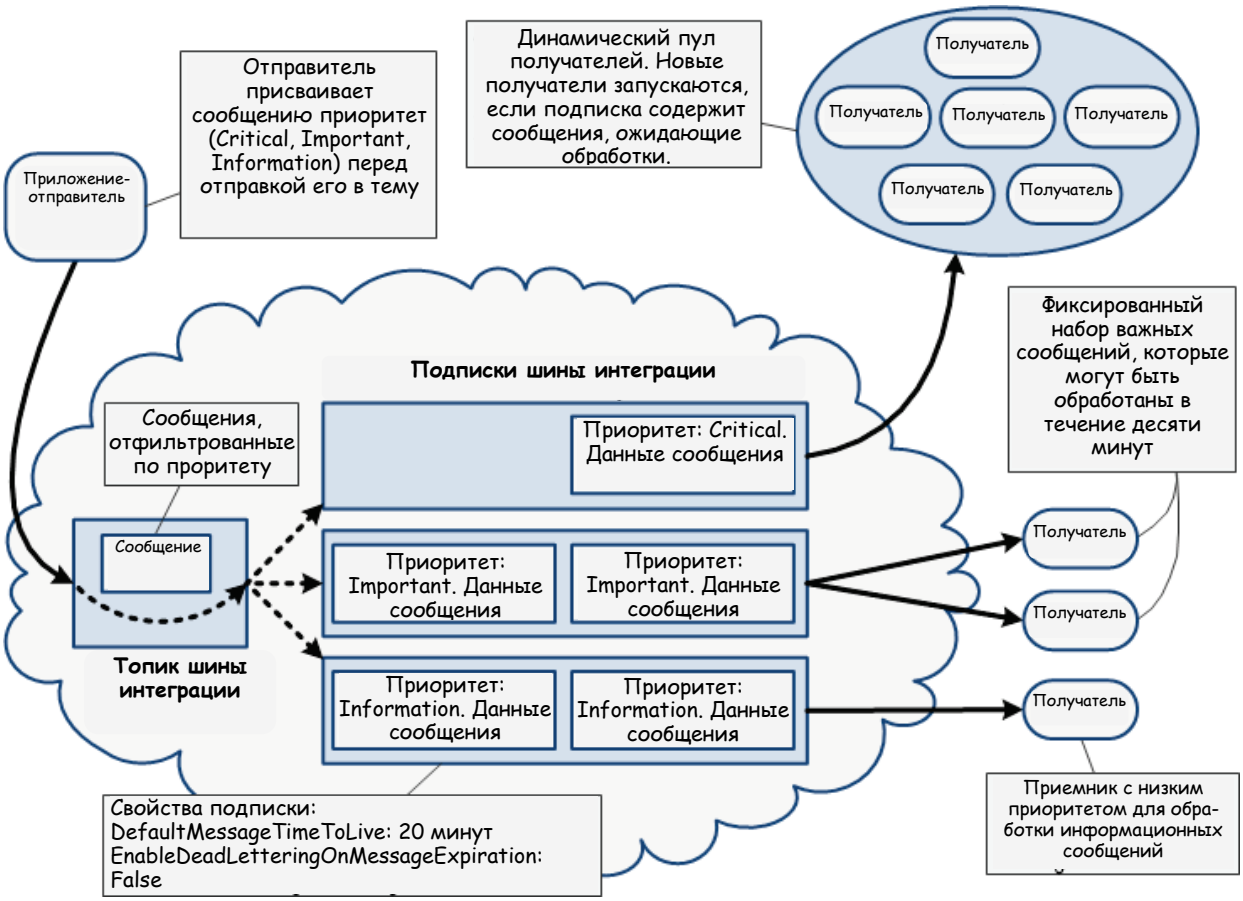



**Рисунок 3**

**Приоритезация сообщений при помощи топиков и подписок шины интеграции**

- Отправители сообщений ожидают ответ на это сообщение от получателя. Число отправителей может существенно различаться с течением времени.

В разделе «Рекомендации по использованию очередей шины интеграции» в «Приложении В. Реализация подхода "коммуникации без границ"» описывается процедура помещения сообщения в очередь отправителем, получения ответа из другой очереди и сопоставления ответа с оригинальным сообщением с помощью свойства **CorrelationId** сообщения. Отправитель указывает очередь, в которую нужно поместить ответ, в свойстве **ReplyTo** сообщения, а получатель заполняет свойство **CorrelationId** сообщения-ответа копией значения свойства **MessageId** оригинального сообщения-запроса.

Такой подход очень прост, он подходит для относительно небольшой статичной группы отправителей, но не обеспечивает достаточную масштабируемость и необходимую скорость реакции на быстрое изменение количества отправителей. Это обусловлено тем, что каждому отправителю требуется собственная очередь шины интеграции.На создание этих очередей требуется время и определенные финансовые затраты.В идеале, если очередь больше не нужна, ее необходимо удалить. Топики и подписки шины интеграции предоставляют лучшие варианты этого подхода в динамичной среде.

Подписки шины интеграции поддерживают понятие *корреляция подписки*. Этот механизм позволяет приложениям подключиться к топику и создать подписку, которая фильтрует сообщения по свойству **CorrelationId**. Подписки шины интеграции предоставляют фильтр **CorrelationFilter** специально для этих целей. Для реализации корреляции подписки необходимо выполнить следующие операции:

- Отправитель создает сообщение и присваивает уникальное значение свойству **MessageId**.
- Отправитель подключается к подписке шины интеграции, из которой он рассчитывает получить ответ, и добавляет фильтр **CorrelationFilter** для этой подписки, указав значение свойства **MessageId** сообщения-оригинала. Все отправители используют один и тот же топик, но фильтр гарантирует, что каждому отправителю придет ответ только на то сообщение, которое послал именно он.

> **Примечание**
>
> Для одной подписки можно установить более одного фильтра. Сообщение передается подписчику, когда выражение фильтра соответствует свойствам сообщения. Однако, если совпадают несколько выражений, то одно и то же сообщение появится несколько раз (один раз для каждого совпадения).

- Отправитель помещает сообщение в топик шины интеграции, на который подписаны один или несколько получателей.
- Получатель принимает сообщение, основываясь на любом фильтре, который применяется к подписке, а затем обрабатывает это сообщение.
- Получатель создает сообщение-ответ и заполняет свойство **CorrelationId** копией значения свойства **MessageId** оригинального сообщения.
- Получатель помещает ответное сообщение в топик шины интеграции, который является общим для всех приложений-отправителей.
- Когда в топике появляется сообщение, значение свойства **CorrelationId** которого совпадает со значением свойства **MessageId** сообщения-оригинала, фильтр

**CorrelationFilter** соответствующей подписки обеспечивает его доставку надлежащему получателю.

> **Мнение Маркуса**
>
> Фильтр **CorrelationFilter** был разработан специально для этого сценария, это чрезвычайно эффективный механизм фильтрации сообщений. С фильтром **SqlFilter** ситуация иная, он обеспечивает более высокую гибкость, однако подлежит лексикографическому анализу в момент создания и обуславливает более высокие затраты на выполнение.

На рисунке 4 показана структура данного решения.

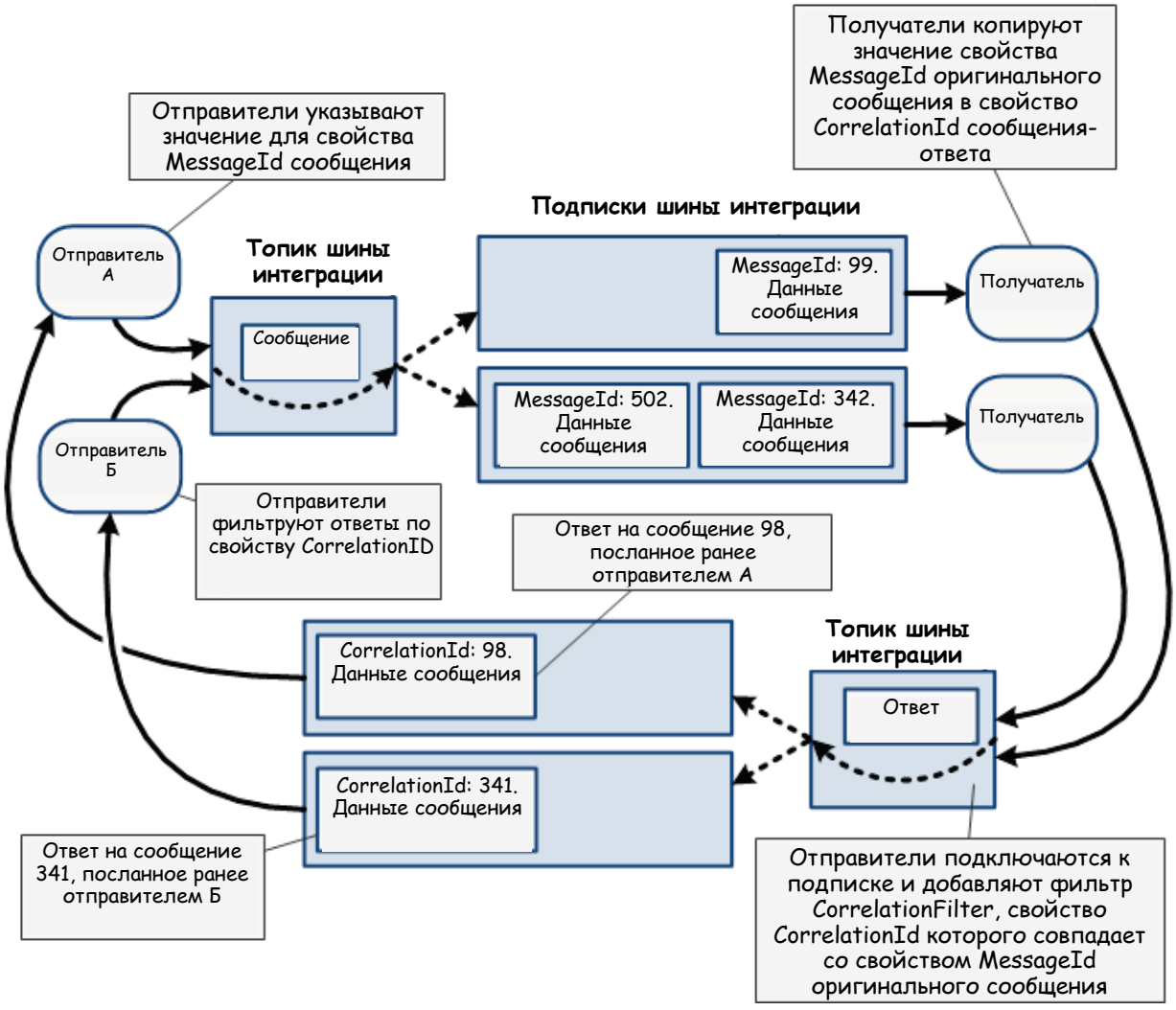



**Рисунок 4**

### Использование корреляции подписки для доставки сообщений-ответов отправителю

- Ваша система непрерывно обрабатывает большой объем сообщений. В целях поддержки производительности ранее было принято решение реализовать механизм горизонтального масштабирования с помощью очередей шины интеграции, но вы обнаружили, что необходим более строгий контроль над тем, какие получатели какие сообщения обрабатывают, кроме того вам нужно направлять сообщения получателям, работающим на определенных серверах.

Как описано в разделе «Рекомендации по использованию очередей шины интеграции» в «Приложении В. Реализация подхода "коммуникации без границ"», очереди шины интеграции позволят вам реализовать простой механизм балансировки нагрузки в процессе обработки сообщений, когда несколько получателей могут прослушивать одну и ту же очередь. Результат — циклический алгоритм распределения сообщений.Но при этом вам не удастся реализовать более строгий контроль над получателями и обрабатываемыми ими сообщениями, поскольку циклический алгоритм такой сценарий не предусматривает. Например, вашей системе может потребоваться, чтобы все сообщения, свойству **Warehouse** которых присвоено значение **А**, обрабатывались получателем, работающим на ближайшем к хранилищу **А** сервере, при этом сообщения с меткой **Б** обрабатывались получателем, работающим на сервере, расположенном как можно ближе к хранилищу **Б** и так далее.

Топики и подписки шины интеграции предоставляют полезный механизм для секционирования нагрузки, связанной с обработкой сообщений, с учетом одного или нескольких свойств каждого сообщения. Вы можете создать набор взаимоисключающих фильтров, которые охватывают разные диапазоны значений в свойствах сообщений и направляют каждый диапазон своей подписке. Различные получатели, прослушивающие эти подписки, могут работать на конкретных локальных серверах, но они также могут быть реализованы как рабочая роль в облаке. Кроме того, у каждой подписки может быть несколько получателей. В этом случае получатели будут конкурировать за сообщения от подписки, а для очереди будет реализован циклический алгоритм с целью балансировки нагрузки.

На рисунке 5 показан пример структуры системы хранения данных. Все получатели созданы при помощи инструментов из пакета Windows Azure SDK и имеют прямое подключение к различным подпискам шины интеграции. Обмен данными с хранилищем Б более интенсивный, чем с хранилищем А, поэтому сообщения, отправляемые хранилищу Б, обрабатываются несколькими получателями, которые работают на оборудовании, расположенном ближе к хранилищу Б.

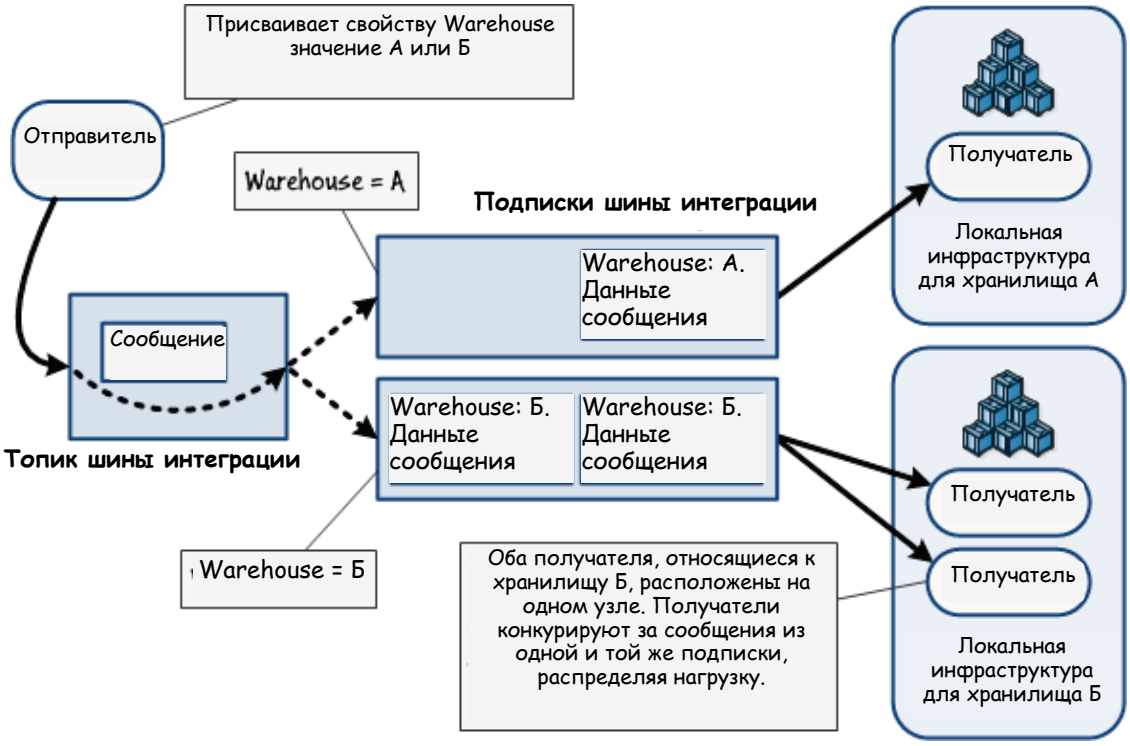



**Рисунок 5**

**Горизонтальное масштабирование при помощи топиков и подписок шины интеграции**

**Мнение Маркуса**

Если вы создали приложение-отправитель с помощью метода **Send** объекта **MessageSender** для отправки сообщений в очередь шины интеграции, вам не придется изменять эту часть кода, поскольку можно использовать тот же метод, чтобы помещать сообщения в топик. Все, что нужно сделать,— создать для сообщений соответствующие свойства, которые требуются фильтру подписки, и задать этим свойствам значения, прежде чем отправить сообщения в топик. Чтобы получать сообщения из подписки шины интеграции, используйте объект **SubscriptionClient**.

- Сообщения, полученные от подписки, возможно, придется отправить к месту назначения для дополнительной обработки, в зависимости от определяемых в системе критериев. Этот механизм переадресации должен быть прозрачным для отправителя, поскольку критерии переадресации и конечный пункт назначения могут измениться. Кроме того, приложение-получатель не должно зависеть от каких-либо изменений, касающихся этих критериев и конечного пункта назначения.

Рассмотрим пример системы обработки заказов: веб-приложение посылает информацию о заказах приложению-получателю через топик шины интеграции. Приложение-получатель отвечает за организацию упаковки и отправки заказа. Все заказы могут быть подвергнуты дополнительной проверке и аудиту, в зависимости от суммы счета. Такой анализ выполняется независимым набором процессов, составляющих логику аудита, которая определяется политикой обработки заказов, принятой в организации.

Например, если сумма заказа меньше 100, данные просто заносятся в журнал, если сумма попадает в диапазон от 100 до 499, заказ регистрируется, а подробная информация о нем выводится на печать для последующей проверки аудитором. Если сумма равна 500 и более, заказ регистрируется, а данные о нем отправляются по электронной почте непосредственно аудитору для немедленного анализа. Аудитор может принять решение об отмене заказа, если клиент не соответствует определенным требованиям. Однако эти пороговые значения могут меняться, и бизнес-логика приложения-получателя должна быть изолирована от этих изменений.

Такой изоляции можно достичь с помощью правила для действия фильтра. Правило фильтра может изменять, добавлять или удалять свойства сообщения, когда сообщение извлекается из подписки шины интеграции. Правила фильтра применяются прозрачно, когда сообщение извлекается приложением-получателем. Приложение-получатель может создать копию сообщения в целях выполнения собственной обработки, а также переслать полученное сообщение в другой топик, который направляет сообщение в соответствующий пункт назначения с учетом обновленного набора свойств.

На рисунке 6 показана возможная структура для примера обработки заказов. Отправитель указывает общую стоимость по заказу в свойстве **TotalCost** сообщения-оригинала, вместе с другими свойствами (не показаны на рисунке), которые используются для пересылки сообщений приложению-получателю («выполняющий переадресацию получатель»). Когда приложение-получатель принимает сообщение, фильтр применяет правило, которое автоматически добавляет свойство **PriceRange** каждому сообщению. Свойство **PriceRange** может принимать значения **Low**, **Medium** или **High** в соответствии со стоимостью заказа. Сумма меньше 100 соответствует значению **Low**, сумма в диапазоне от 100 до 499 соответствует значению **Medium**, а сумма от 500 и выше — значению **High**. Приложение-получатель

выполняет все необходимые операции по обработке. В то же время, оно отправляет копию полученного сообщения с добавленным свойством **PriceRange** в топик шины интеграции, на который подписаны различные получатели-аудиторы. Подписки получателей-аудиторов фильтруют сообщения по свойству **PriceRange** и направляют их получателю, который выполняет необходимые операции, как описано ранее.

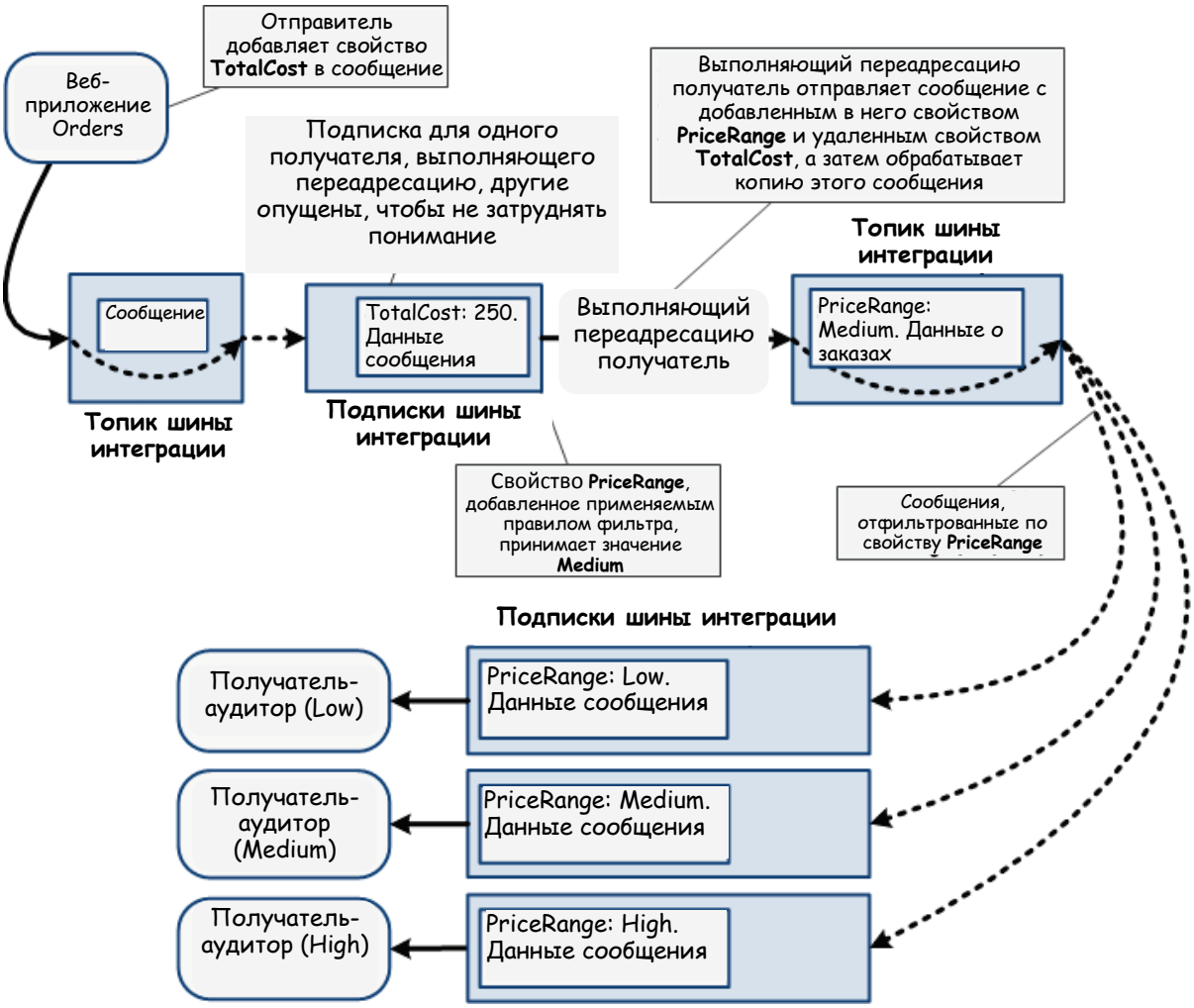



**Рисунок 6**

### Поиск прямого маршрута для сообщений с помощью правил для действия фильтра

В следующем фрагменте кода показано, как добавлять правила для действий фильтра, используемых в этом примере для подписки, которую прослушивает приложение-приемник, выполняющее переадресацию. Обратите внимание, что эти правила для действий фильтра также удаляют свойство **TotalCost** из сообщения, поскольку оно не является необходимым для пересылки сообщений приложению-аудитору, поиск прямого маршрута выполняется только с учетом свойства **PriceRange**. При этом полная информация о заказе получателю-аудитору по-прежнему доступна в теле сообщения.

**C#**

```
...
// Define action rule filters
var ruleLowPrice = new RuleDescription()
{
  Action = new SqlRuleAction(
```

```
    "set PriceRange='Low';remove TotalCost"),
Filter = new SqlFilter("TotalCost < 100"),
  Name = "LowPrice"
};

var ruleMediumPrice = new RuleDescription()
{
  Action = new SqlRuleAction(
    "set PriceRange='Medium';remove TotalCost"),
Filter = new SqlFilter(
    "TotalCost >= 100 AND TotalCost < 500"),
  Name = "MediumPrice"
};

var ruleHighPrice = new RuleDescription()
{
  Action = new SqlRuleAction(
    "set PriceRange='High';remove TotalCost"),
Filter = new SqlFilter("TotalCost >= 500"),
  Name = "HighPrice"
};

...
var subscriptionClient =
  messagingFactory.CreateSubscriptionClient(...);

// Добавление правил к подписке
subscriptionClient.AddRule(ruleLowPrice);
subscriptionClient.AddRule(ruleMediumPrice);
subscriptionClient.AddRule(ruleHighPrice);
```

### Ограничения, которые необходимо учитывать при использовании топиков и подписок шины интеграции для маршрутизации сообщений

Топики и подписки шины интеграции позволяют реализовать только простой механизм маршрутизации. Из соображений безопасности, фильтры, которые вы задаете, не могут получить доступ к телу сообщения, принимать решения им приходится, основываясь только на данных, представленных в виде свойств сообщения. Чаще всего фильтры задаются на основе класса **SqlFilter**. В целях оптимизации, условия в этих фильтрах ограничены подмножеством синтаксиса SQL92. Вы можете провести прямое сравнение данных и значений с помощью обычных арифметических и логических операторов, но эти фильтры не поддерживают функции, например функция **Substring** отсутствует. Если потребуется маршрутизация на основе более сложных правил, необходимо реализовать эту логику в коде путем создания получателя, который проверяет данные сообщения, а затем отправляет это сообщение в другую очередь или топик по мере необходимости.

Более подробная информация о типах выражений, поддерживаемых классом **SqlFilter** представлена в статье «SqlFilter.SqlExpression Property» на сайте MSDN: http://msdn.microsoft.com/en-us/library/microsoft.servicebus.messaging.sqlfilter.sqlexpression.aspx.

## Пересылка сообщений в несколько пунктов назначения при помощи топиков и подписок шины интеграции

В предыдущем разделе описывается процедура использования фильтров для распределения сообщений по различным непересекающимся группам, при этом каждая группа направляется в подписку шины интеграции, и каждое сообщение предназначено для одной конкретной подписки. Тем не менее различные подписки могут иметь фильтры с пересекающимися предикатами. В этом случае копии одного и того же сообщения направляются каждой удовлетворяющей условиям подписке. Этот механизм предоставляет средства для пересылки сообщений нескольким адресатам.

Обратная ситуация также возможна. Если фильтры всех подписок не соответствуют свойствам сообщения, это сообщение будет оставаться в очереди топике шины интеграции до истечения срока его действия.

### Рекомендации по использованию топиков и подписок шины интеграции для маршрутизации сообщений в несколько пунктов назначения

Фильтры с перекрывающимися предикатами позволяют реализовать ряд эффективных сценариев. В следующем списке приставлены общие случаи:

> **Примечание**
>
> «Приложение A. Репликация, распространение и синхронизация данных» содержит описание нескольких дополнительных моделей использования топиков и подписок шины интеграции с целью запроса и обновления данных в системе, использующей реплицированные источники данных.

- Ваша система позволяет приложениям-отправителям отсылать запросы на обслуживание, но все эти запросы должны быть занесены в журнал для аудита или диагностики. Процедура занесения в журнал должна быть прозрачной для приложения-отправителя.

Это пример наиболее общей схемы отправки сообщений нескольким получателям. Службы могут использовать подписки для получения предназначенных им сообщений, но при этом все сообщения должны быть переданы в службу ведения журнала аудита, чтобы их можно было записать и сохранить. Пакет Windows Azure SDK предоставляет тип **TrueFilter** специально для этой цели. Этот фильтр подходит всем сообщениям, и любые подписки, которые используют этот фильтр, автоматически получат копию каждого сообщения, отправленного в этот топик.

> **Мнение Маркуса**
>
> **TrueFilter** — это фильтр по умолчанию для подписки, если вы не указываете другой фильтр при создании подписки, **TrueFilter** применяется автоматически.

На рисунке 7 показан пример системы, которая использует фильтр **TrueFilter** для копирования сообщений в журнал аудита с целью дальнейшей проверки.

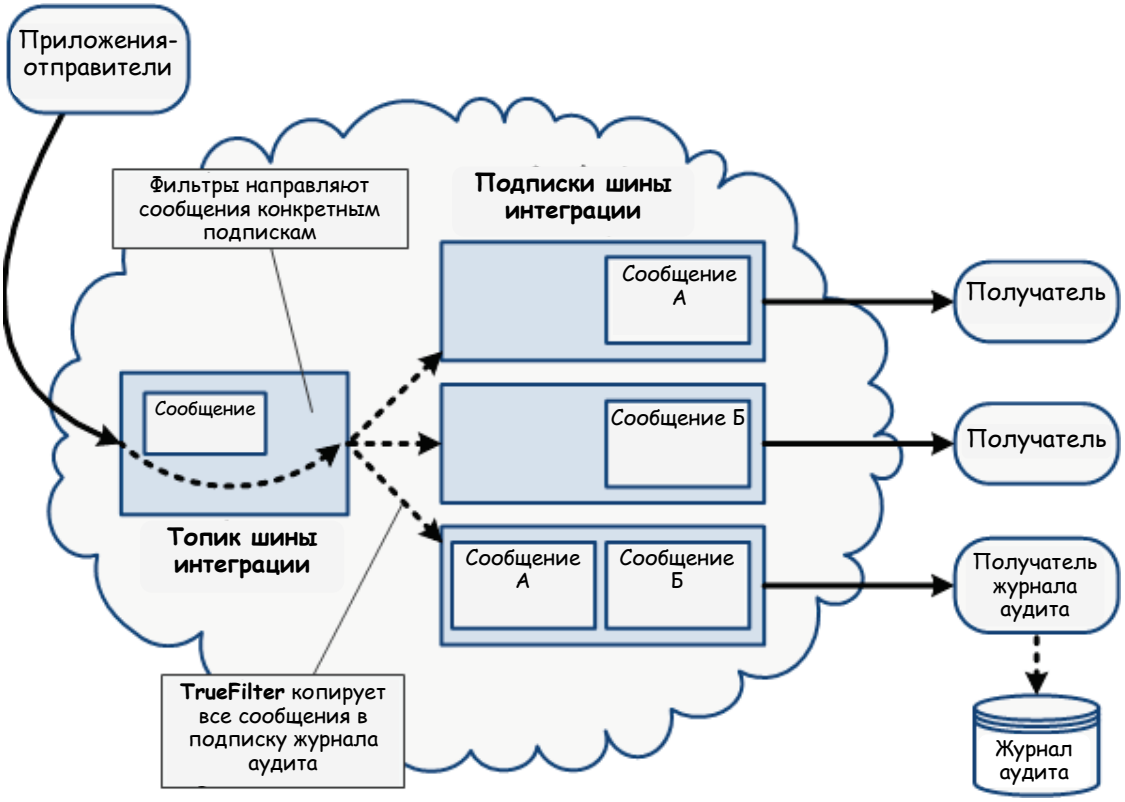



**Рисунок 7**

### Занесение сообщений в журнал с помощью TrueFilter

Получатель журнала аудита — это просто пример приложения, которое может эффективно использовать данный подход. Любая функция, которой требуются копии сообщений, проходящих через вашу систему, может быть реализована подобным образом. Например, вы можете создать приложение, которое подсчитывает количество сообщений, поступающих вашей системе в течение определенного периода времени, и отображает результаты — данные о пропускной способности и производительности вашего решения.

Разумеется, вы можете быть более избирательными. Вместо **TrueFilter** можно использовать перекрывающийся фильтр **SqlFilter**, который перехватывает сообщения с учетом значений определенных свойств. Эти сообщения будут направляться получателям для обработки, а также приложению-получателю журнала аудита.

- В вашей системе возникает множество бизнес-событий. Каждое событие может быть обработано процессами, которых может быть от нуля и больше, и ваша система должна иметь возможность добавлять или удалять процессы, которые могут обрабатывать эти события, не затрагивая бизнес-логику системы. Обработчики событий могут быть запущены удаленно от процессов, которые генерируют эти события.

Процессы, которые генерируют события, чтобы сообщить заинтересованным сторонам о том, что произошло что-то важное. Процессы, прослушивающие события, по своей природе являются асинхронными и не связаны с процессами, которые генерируют события. Топики и подписки шины интеграции обеспечивают превосходную основу для создания такой системы, особенно с учетом требования, предусматривающего возможность размещения обработчиков событий в любом месте и гарантированную доставку событий.

С точки зрения обмена сообщениями, приложение может оповещать заинтересованные стороны о событиях, просто отправляя сообщение, содержащее данные о событиях, в топик шины

интеграции. Каждое приложение, которому требуются уведомления о событиях, может создать свою собственную подписку с фильтром, определяющим условия для сообщений-событий. Топик, в который приложение-отправитель посылает сообщения о событиях, может иметь свойство **DefaultMessageTimeToLive** с соответствующим значением. В таком случае, если ни одно приложение не подписывается на события, оно будет удалено по истечении указанного срока.

> **Мнение Джаны**
>
> Не следует совместно использовать одну и ту же подписку на события двум отдельным приложениям, если они оба должны быть уведомлены о событии. Они будут конкурировать за сообщения о событиях, направляемых в подписку, и каждое сообщение будет передано только одному из приложений.

На рисунке 8 показан пример базовой системы управления конвейером на производственном предприятии. Когда необходимо начать производство, приложение Controller помещает сообщение «Пуск машины» в топик шины интеграции. Каждый агрегат, участвующий в процессе сборки, управляется программой-драйвером, которая получает это сообщение и запускает агрегат. Аналогичным образом, когда производство останавливается, приложение Controller отправляет сообщение «Стоп машины», и соответствующая программа-драйвер останавливает каждый агрегат в контролируемом режиме. Приложение Controller не учитывает особенности устройств, задействованных в процессе производства, агрегаты могут быть добавлены или удалены, но приложение Controller при этом изменять не придется.

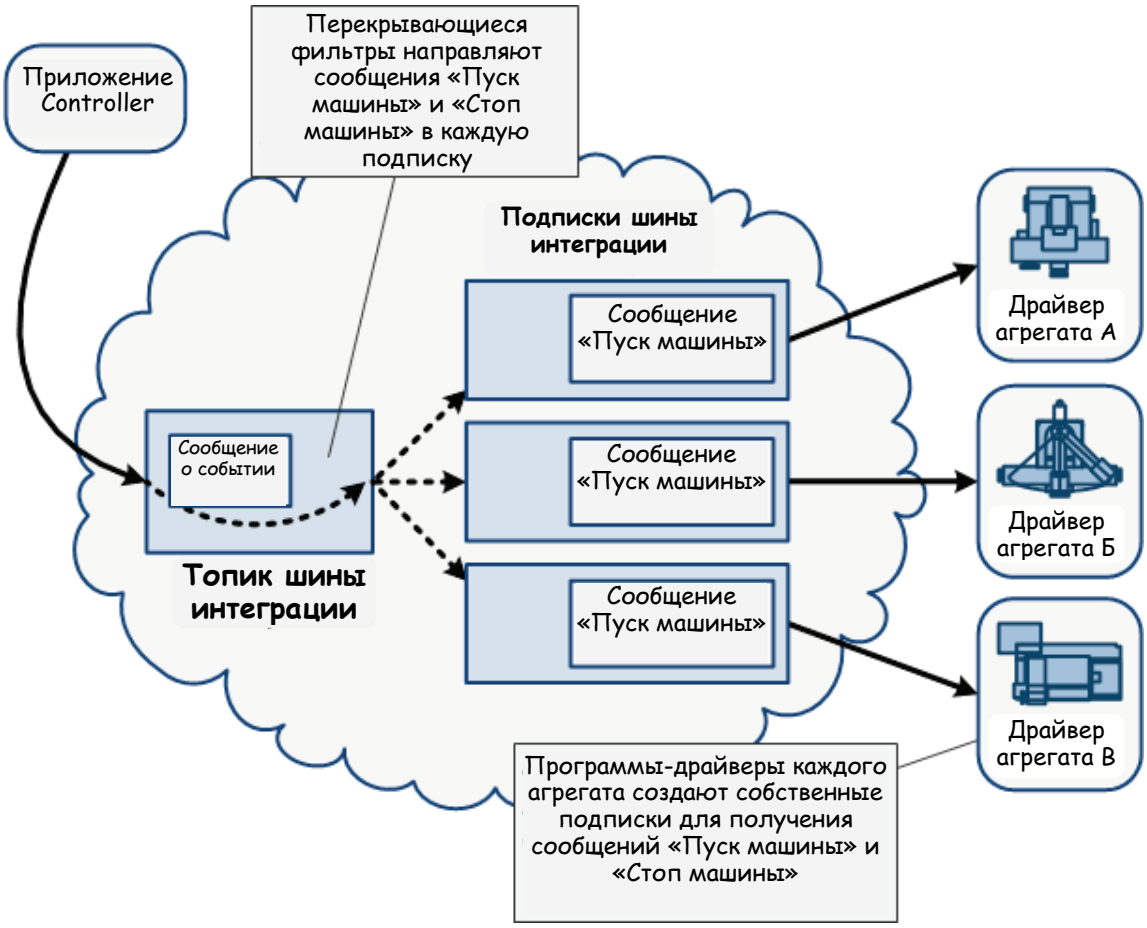



Рисунок 8

## Управление производственной линией на основе событий при помощи подписок шины интеграции

**Примечание**

Дополнительная информация об использовании шины интеграции для организации взаимодействия ролей и приложений представлена в статье «How to Simplify & Scale Inter-Role Communication Using Windows Azure Service Bus» на сайте http://windowsazurecat.com/2011/08/how-to-simplify-scale-inter-role-communication-using-windows-azure-service-bus/.

### Ограничения, которые необходимо учитывать при использовании топиков и подписок шины интеграции для маршрутизации сообщений в несколько пунктов назначения

Важно понимать, что топики и подписки шины интеграции способны обеспечить надежную доставку сообщений одному или нескольким получателям, однако это не может произойти мгновенно. Топики и подписки находятся в облаке, и неизбежно будут возникать некоторые сетевые задержки, связанные с работой Интернета. Кроме того, фильтры, которые определяются с помощью типа **SqlFilter**, подлежат оценке в режиме выполнения, и свойства каждого сообщения должны быть проанализированы и сопоставлены с каждым фильтром каждой подписки. Если топик имеет большое количество подписок (до 2000 в текущей версии шины интеграции), то на оценку и анализ может потребоваться некоторое время.

## Рекомендации по защите топиков и подписок шины интеграции

Для топиков и подписок шины интеграции действуют те же механизмы обеспечения безопасности, что и для очередей шины интеграции. Чтобы настроить безопасность, нужно создать удостоверения и связать их с привилегиями с помощью ACS. См. «Приложение Б. Аутентификация пользователей и авторизация запросов» для получения более подробной информации об использовании ACS. Вы можете присваивать топику привилегии, связанные с утверждениями **Manage и Send**, а подписке — привилегии, связанные с утверждениями **Manage** и **Listen**. Когда приложение подключается к топику или подписке пространства имен шины интеграции, оно должно пройти аутентификацию в ACS и предоставить используемые им удостоверения. См «Приложение В. Реализация подхода "коммуникации без границ"» для получения более подробной информации о подключении к пространству имен шины интеграции и предоставлении информации об удостоверениях.

Как и в случае с очередями шины интеграции, любое взаимодействие с топиками и подписками шины интеграции происходит по каналу TCP и автоматически защищается с помощью SSL.

# Приложение Д. Максимизация масштабируемости, доступности и производительности

Основным преимуществом технологической платформы Windows Azure является ее надежность. Типичное решение Windows Azure реализуется в виде коллекции одной или более ролей, где каждая роль оптимизируется для выполнения определенной категории задач. Например, веб-роль особенно полезна для реализации веб-интерфейса, который обеспечивает пользовательский интерфейс приложения, тогда как рабочая роль обычно выполняет основную бизнес-логику, а именно осуществление необходимой обработки данных, взаимодействие с базой данных, упорядочение входящих и исходящих запросов для других служб и так далее. Если в роли возникает сбой, Windows Azure может прозрачно запустить новый экземпляр, и приложение возобновит работу.

Однако приложение должно быть не только надежным, но и быстро работать и реагировать. Windows Azure поддерживает максимально масштабируемые службы путем реализации возможности динамического запуска и остановки экземпляров приложения, что позволяет решению Windows Azure обрабатывать поток запросов в моменты пиковой нагрузки, а также уменьшать масштаб по мере снижения спроса, уменьшая уровень использования ресурсов и соответствующие расходы.

**Мнение По**

Если вы создаете коммерческую систему, у вас могут быть договорные обязательства, касающиеся предоставления определенного уровня производительности клиентам. Эти обязательства должны быть указаны в соглашении об обслуживании (Service Level Agreement, SLA), которое гарантирует время реакции или пропускную способность. В этой среде необходимо, чтобы вы понимали архитектуру своего приложения, используемые им ресурсы и инструменты, предоставляемые Windows Azure для построения и поддержки эффективной системы.

Однако масштабируемость — это не единственный фактор, влияющий на производительность и время реакции. Обратите внимание, что если приложение, работающее в облаке, получает доступ к ресурсам и базам данных, размещенным на ваших локальных серверах, то эти ресурсы становятся недоступными непосредственно через вашу локальную высокоскоростную сеть. Вместо этого приложение должно извлекать эти данные через Интернет с его меньшей пропускной способностью, более высокими задержками и присущей ему непредсказуемостью с точки зрения надежности и пропускной способности. Это может привести к увеличению времени отклика для пользователей приложений или уменьшению пропускной способности для ваших служб.

Если же ваше приложение или служба работают удаленно от вашей организации, они также будут работать удаленно от ваших пользователей. Это может казаться незначительной проблемой, если вы создаете общедоступный веб-сайт или службу, так как пользователи подключались к ним удаленно еще до переноса функциональности в облако, но такое изменение может повлиять на производительность

пользователей организации, которые раньше получали доступ к вашему решению через локальную сеть. Кроме того, расположение приложения или службы может повлиять на их доступность, если путь от пользователя пересекает элементы сети, которые сильно перегружены, в результате чего сетевое соединение может быть прервано. И наконец, ваши пользователи не смогут подключиться в случае катастрофического регионального отключения доступа к Интернету или ошибки центра обработки данных, в котором размещены ваши приложения и службы.

В этом приложении рассмотрены проблемы, связанные с поддержанием производительности, сокращением времени отклика приложения и обеспечением возможности постоянного доступа к приложению для пользователей при переносе его функциональности в облако. Также тут описаны решения и рекомендации по устранению этих проблем с помощью технологий Windows Azure.

# Требования и проблемы

Основными причинами увеличения времени отклика и недостаточной доступности в распределенной среде являются нехватка ресурсов для работы приложений и задержки в сети. Масштабирование поможет обеспечить достаточное количество ресурсов. Однако, сколько бы вы не старались настроить свои приложения и улучшить их работу, пользователи будут ощущать недостаточную производительности системы, если эти приложения не могут своевременно получать запросы и отправлять ответы из-за медленной работы сети. Поэтому важно организовать свое решение так, чтобы минимизировать эту задержку сети, оптимально используя доступную пропускную способность и ресурсы, размещая их как можно ближе к коду и пользователям, которым они необходимы.

В следующих разделах указываются некоторые общие требования масштабируемости, доступности и производительности, а также подытоживаются многие проблемы, с которыми можно столкнуться при реализации решений, удовлетворяющих этим требованиям.

## Управление эластичностью в облаке

**Описание: ваша система должна поддерживать изменяющуюся рабочую нагрузку с учетом рентабельности.**

Многие коммерческие системы должны поддерживать рабочую нагрузку, которая может значительно меняться со временем. На протяжении длительного времени нагрузка может быть стабильной с регулярным объемом запросов прогнозируемого характера. Однако могут возникать моменты, когда нагрузка значительно и быстро увеличивается. Эти пиковые моменты могут возникать в ожидаемые периоды. Например, система бухгалтерского учета может получать большое количество запросов в конце каждого месяца, когда пользователи генерируют месячные отчеты, а также повышение нагрузки может происходить ближе к концу финансового года. В некоторых типах приложения нагрузка может резко возрастать. Например, количество запросов к службе новостей может значительно увеличиваться, если происходят какие-либо драматические события.

Облако является максимально масштабируемой средой, и вы можете запускать новые экземпляры службы для удовлетворения спроса при увеличении объема запросов. Однако чем больше экземпляров службы вы запускаете, тем больше ресурсов им требуется, а расходы, связанные с работой системы, возрастают пропорционально. Поэтому с экономической точки зрения целесообразно уменьшить количество экземпляров службы и ресурсов в случае сокращения спроса на вашу систему.

Как этого добиться? Одним из вариантов является контроль решения и запуск дополнительных экземпляров служб, когда количество запросов, поступающих в определенный период времени, превышает установленное пороговое значение. Если нагрузка увеличивается и далее, можно задать дополнительные пороговые значения и запускать еще больше экземпляров. Если же объем запросов

станет меньше этих пороговых значений, можно прекратить работу дополнительных экземпляров. В период малой активности необходимо иметь лишь минимальное количество экземпляров службы. Однако это решение имеет несколько недостатков:

- Необходимо автоматизировать процесс, который запускает и останавливает экземпляры службы в ответ на изменения нагрузки системы и количества запросов. Вряд ли можно осуществлять эти задачи вручную, так как пики и спады нагрузки могут происходить в любое время.
- Количество запросов, которые поступают в определенном временном интервале, не может быть единственной мерой нагрузки, например небольшое количество запросов, каждый из которых требует интенсивной обработки, может также повлиять на производительность. Следовательно, в процессе, который предсказывает производительность и определяет необходимые пороговые значения, должны быть предусмотрены вычисления, позволяющие измерять использование сложного комплекса ресурсов.

**Мнение Бхарата**

Следует помнить, что запуск и остановка экземпляров служб — это не мгновенная операция. Выполнение таких задач в Windows Azure может занять 10–15 минут, поэтому любые оценки производительности должны включать упреждающий элемент, основанный на тенденции изменений во времени, и инициировать новые экземпляры службы, чтобы они были готовы в случае необходимости.

## Уменьшение задержки сети при получении доступа к облачным приложениям

**Описание: пользователи должны быть подключены к самому близкому экземпляру вашего приложения, запущенного в облаке, чтобы минимизировать задержку сети и сократить время отклика.**

Облачное приложение может размещаться в центре обработки данных в одной части мира, а пользователь, подключающийся к приложению, может находиться даже на другом континенте. Расстояние между пользователями и приложениями и службами, к которым они хотят получить доступ, может иметь значительное влияние на время отклика системы. Вы должны принять стратегию, которая минимизирует это расстояние и уменьшает соответствующую задержку сети для пользователей, которые подключаются к системе.

Если ваши пользователи географически рассредоточены, вы можете создать репликации облачных приложений и разместить их в центрах обработки данных, которые так же рассредоточены. Пользователи смогут подключаться к ближайшему доступному экземпляру приложения. Вопрос, который вы должны рассмотреть в этом сценарии — это как направить пользователя в ближайший экземпляр приложения.

## Максимизация доступности облачных приложений

**Описание: у пользователя всегда должна быть возможность подключиться к приложению, которое запущено в облаке.**

Как же вы можете гарантировать, что приложение в облаке всегда работает и что пользователи могут к нему подключиться? Репликация приложения во все центры обработки данных может быть частью решения, однако следует обратить внимание на следующие проблемы:

- Что произойдет, если ближайшее к пользователю приложение перестает работать, или если невозможно установить сетевую связь?
- Ближайший к пользователю экземпляр приложения может быть очень загружен в сравнении с более далекими экземплярами. Например, во второй половине дня в Европе трафик в европейские центры обработки данных может быть намного больше трафика на Дальнем Востоке или Западном побережье Америки. Как вы сможете балансировать стоимость подключения к экземпляру приложения, которое работает на сильно загруженном сервере, в сравнении с подключением к экземпляру, работающему на менее загруженном сервере более удаленно?

---

## Оптимизация времени отклика и пропускной способности для облачных приложений

**Описание: время отклика для запущенной в облаке службы должно быть как можно меньше, а пропускная способность должна быть максимизирована.**

Windows Azure — это платформа с широкими возможностями масштабирования, что позволяет обеспечить высокую производительность приложений. Однако наличие вычислительных мощностей не гарантирует, что приложение будет быстро реагировать. Приложение, разработанное для последовательной работы, не будет лучшим образом использовать данную платформу и может быть значительное время блокировано в ожидании завершения более медленных зависимых операций. Решением этой проблемы является асинхронное выполнение операций, и именно этот подход описан в данном руководстве.

Помимо проектирования и реализации логики приложения, ключевым фактором, регулирующим время отклика и пропускную способность службы, является скорость доступа к необходимым ему ресурсам. Некоторые или все эти ресурсы могут быть размещены удаленно в других центрах обработки данных или на локальных серверах. Операции доступа к удаленным ресурсам могут требовать подключения через Интернет. Для уменьшения влияния задержек и непредсказуемости сети вы можете кэшировать эти ресурсы локально в службе. Однако такой подход приводит к двум очевидным вопросам:

- Что будет, если ресурс обновлен в удаленном экземпляре? Кэшированная копия, которой пользуется служба, будет неактуальной, а поэтому, как служба может выявить и обработать такую ситуацию?
- Что происходит в том случае, когда самой службе необходимо обновить ресурс? В таком случае кэшированная копия, которой пользуются экземпляры этой или других служб, может стать устаревшей.

---

Кэширование является полезной стратегией для снижения конкуренции за ресурсы общего пользования и может улучшить время отклика приложения, даже если используемый им ресурс является локальным. Однако проблемы, связанные с кэшированием, остаются неизменными, а именно: если изменяется локальные ресурс, то кэшированные данные не обновляются.

> **Мнение Бхарата**
>
> Облако не являет чудодейственным лекарством для ускорения работы приложений, которые разработаны без учета производительности и масштабируемости.

## Платформа Windows Azure и связанные с ней технологии

Платформа Windows Azure поддерживает разнообразные технологии, которые помогут решить проблемы, связанные с каждым их этих требований:

- **Функциональный блок для автоматического масштабирования (Autoscaling Application Block) из библиотеки Enterprise Library**. Вы можете использовать этот функциональный блок, чтобы задавать показатели эффективности, замерять эффективность по этим показателям и запускать или останавливать экземпляры служб для поддержания производительности в допустимых пределах.
- **Диспетчер трафика Windows Azure**. Вы можете использовать эту службу для уменьшения задержки сети путем направления пользователей к ближайшему экземпляру приложения, работающему в облаке. Диспетчер трафика Windows Azure может также обнаруживать сбои в экземплярах службы или их недоступность, автоматически перенаправляя запросы пользователей к другому доступному экземпляру службы.
- **Служба Windows Azure Caching**. Вы можете использовать эту службу для кэширования данных в облаке и предоставления масштабируемого и надежного общего доступа для нескольких приложений.
- **Сеть доставки контента (CDN)**. Вы можете использовать эту службу для улучшения времени отклика веб-приложений путем кэширования часто используемых данных, расположенных ближе к пользователям, которые запрашивают их.

> **Примечание**
>
> Служба Windows Azure Caching непосредственно предназначена для повышения эффективности работы веб-приложений и служб, работающих в облаке. Однако пользователи часто вызывают эти веб-приложения и службы с настольных систем, используя либо пользовательское приложение, которое подключается к ним, либо веб-браузер. Данные, возвращаемые веб-приложением или службой, могут иметь существенный объем, и если пользователь находится очень далеко, то может понадобиться значительное время для пересылки этих данных на компьютер пользователя. CDN позволяет кэшировать часто запрашиваемые данные в различных местах по всему миру. Когда пользователь создает запрос, данные предоставляются из наиболее оптимального расположенного центра с учетом текущего объема трафика в различных узлах Интернета, через которые проходит запрос. Более детальная информация, примеры и упражнения, демонстрирующие настройку службы CDN, представлены на сайте MSDN в статье «Windows Azure CDN» по адресу http://msdn.microsoft.com/en-us/gg405416. Также в главе 3 «*Accessing the Surveys Application*» руководства «*Developing Applications for the Cloud, 2nd Edition*» приводятся подробные сведения о дальнейшей реализации. Подробная информация представлена на сайте http://msdn.microsoft.com/en-us/library/ff966499.aspx.

В следующих разделах описываются функциональный блок для автоматического масштабирования из библиотеки Enterprise Library, диспетчер трафика Windows Azure и служба Windows Azure Caching, а также предоставляются рекомендации о том, как использовать их в различных сценариях.

## Управление эластичностью в облаке с помощью функционального блока для автоматического масштабирования из библиотеки Enterprise Library

Можно реализовать пользовательское решение, которое управляет несколькими развернутыми экземплярами веб- и рабочих ролей, используемых вашим приложением. Однако это вовсе непростая задача, поэтому целесообразно рассмотреть возможность использования встроенной библиотеки, достаточно гибкой и настраиваемой, чтобы удовлетворить ваши требования.

> **Мнение По**
>
> Существуют внешние службы, которые могут управлять автоматическим масштабированием, но вы должны предоставить этим службам свой сертификат управления, чтобы они могли получать доступ к экземплярам, что может быть недопустимым подходом для вашей организации.

Функциональный блок для автоматического масштабирования из библиотеки Enterprise Library (также известный под названием Wasabi) предоставляет такое решение. Данный компонент является частью пакета интеграции Microsoft Enterprise Library 5.0 для Windows Azure и может автоматически масштабировать приложение Windows Azure или работать на основе правил, определяемых вами специально для этого приложения или службы. Вы можете использовать эти правила, чтобы обеспечить своему приложению или службе необходимую пропускную способность в ответ на изменение рабочей нагрузки, и в то же время минимизировать и контролировать расходы на размещение.

Операции масштабирования обычно изменяют количество экземпляров роли в вашем приложении, но этот блок также позволяет вам применять другие меры для масштабирования, например дросселирование некоторых функций вашего приложения. Это означает, что существуют возможности достижения очень точного контроля поведения на основании диапазона заранее определенных и динамически обнаруживаемых условий. Функциональный блок для автоматического масштабирования позволяет указать следующие типы правил:

- **Правила ограничения**, которые позволяют установить минимальные и максимальные значения для нескольких экземпляров роли или ряда ролей на основании графика.
- **Правила реагирования**, которые позволяют настроить несколько экземпляров роли или ряда ролей на основании агрегированных значений, полученных из точек данных, собранных в среде Windows Azure или приложении. Вы также можете использовать правила реагирования, чтобы изменить настройки конфигурации, что позволит приложению изменить свое поведение и изменить использования ресурсов, например путем отключения не очень важных функций или некоторого ограничения пользовательского интерфейса при увеличении нагрузки и спроса.

---

> Правила определяются в формате XML и могут сохраняться в хранилище BLOB-объектов Windows Azure, в файле или в создаваемом вами пользовательском хранилище.

Применяя сочетание этих правил, можете быть уверены, что ваше приложение или служба будет удовлетворять требованиям, касающимся спроса и нагрузки, даже в самые напряженные периоды, чтобы соответствовать соглашению об уровне обслуживания (SLA), минимизировать время отклика, а также обеспечить доступность при сведении до минимума эксплуатационных расходов.

### Как функциональный блок для автоматического масштабирования управляет экземплярами роли

Функциональный блок для автоматического масштабирования может отслеживать основные показатели производительности в ролях вашего приложения и автоматически разворачивать или удалять экземпляры. Например, на рисунке 1 показано, как количество экземпляров роли может изменяться со временем в пределах от минимального до максимального количества экземпляров.

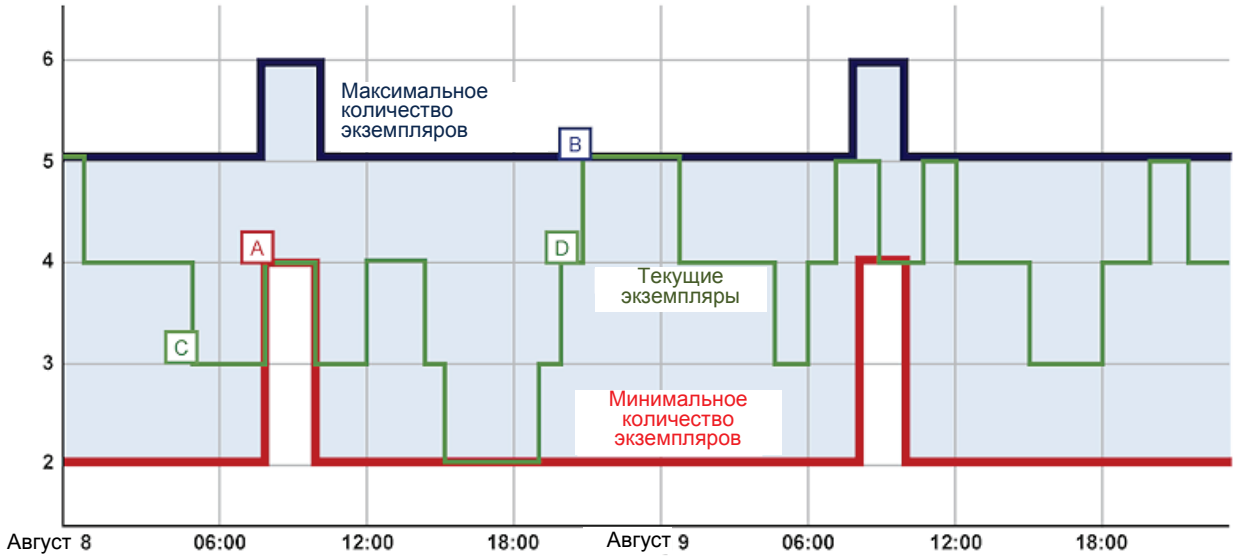



**Рисунок 1**

### Визуализация данных пределов масштабирования и действий масштабирования для роли

Поведение, показанное на рисунке 1, было результатом следующей настройки блока для автоматического масштабирования:

- Действующее по умолчанию **ограничительное правило** всегда активно. Установлено количество экземпляров: минимум два и максимум пять. В точке B на графике это правило не позволяет блоку разворачивать дополнительные экземпляры, даже если это может быть оправдано нагрузкой на приложение.
- **Ограничительное правило**, которое активно каждый день с 08:00 в течение двух часов с установленным диапазоном количества экземпляров: минимум четыре и максимум шесть экземпляров. На графике показано, как в точке A блок разворачивает новый экземпляр роли в 08:00.
- **Операнд** под названием **Avg_CPU_RoleA** связан со средним значением на протяжении последних 10 минут счетчика производительности Windows **\Processor(_Total)\% Processor Time**.
- **Правило реагирования**, которое увеличивает количество развернутых экземпляров роли на единицу, когда значение операнда **Avg_CPU_RoleA** больше чем 80. Например, в точке **D** на графике блок увеличивает количество ролей до четырех, а затем до пяти при увеличении нагрузки на процессор.
- **Правило реагирования**, которое уменьшает количество развернутых экземпляров роли на единицу, когда значение операнда **Avg_CPU_RoleA** становит меньше 20. Например, в точке **C** на графике блок уменьшает количество ролей до трех при уменьшении нагрузки на процессор.

**Мнение По**

Указав соответствующий набор правил для функционального блока с целью автоматического масштабирования, вы можете настроить автоматическое масштабирование количества экземпляров ролей в своем приложении, чтобы удовлетворить известные требования периодов пикового спроса и автоматически реагировать на динамическое изменение нагрузки и спроса.

### Ограничительные правила

Ограничительные правила используются для упреждающего масштабирования вашего приложения для ожидаемого спроса, а также для ограничения возможного количества экземпляров, чтобы правила реагирования не меняли количество экземпляров, выходя за рамки установленных ограничений. Имеется полный набор параметров для указания временного диапазона для ограничительного правила, включая фиксированные периоды и фиксированную продолжительность, ежедневное, еженедельное, ежемесячное и ежегодное повторение, а также повторение относительно определенных событий, например в последнюю пятницу каждого месяца.

### Правила реагирования

Правила реагирования определяют условия и действия, которые изменяют количество развернутых экземпляров ролей или поведение приложения. Каждое правило состоит из одного или более операндов, которые определяют, как блок соответствует данным контрольных точек с указанными вами значениями, и одного или нескольких действий, которые выполняются блоком, когда операнды соответствуют контролируемым значениям.

Операнды, которые определяют точки данных для мониторинга роли, могут использовать любой счетчик производительности операционной системы Windows, длину очереди хранилища Windows Azure и другие встроенные показатели. Кроме того, вы можете создать собственный операнд, специфичный для ваших собственных требований, например количество необработанных заказов в вашем приложении.

Функциональный блок для автоматического масштабирования считывает данные о производительности, полученные механизмом диагностики Windows Azure из хранилища Windows Azure. Windows Azure по умолчанию не заполняет их данными из службы диагностики Windows Azure; вы должны запустить код в своей роли при ее запуске или выполнить скрипты, когда приложение работает, чтобы настроить диагностику Windows Azure для сбора необходимой информации, а затем запустить службу диагностики.

Условия правил реагирования могут использовать широкий диапазон сравнительных функций между операндами для активации соответствующих действий. Эти функции включают в себя типичные проверки, такие как больше, больше или равно, меньше, меньше или равно и равно. Вы также можете выполнять отрицательные проверки, используя функцию **not**, и создавать сложные условные выражения, используя логические комбинации **AND** и **OR**.

### Действия

Функциональный блок для автоматического масштабирования предоставляет следующие типы действий:

- Действие **setRange** указывает максимальное и минимальное количество экземпляров роли, которое должно быть доступно в определенный период времени. Это действие применимо только к ограничительным правилам.

- Действие scale указывает на то, что блок должен увеличить или уменьшить количество развернутых экземпляров роли на абсолютное или относительное значение. Вы можете определить целевую роль, используя имя, или же можете задать группу масштабирования в конфигурации блока, которая включает в себя имена более одной роли, а затем указать группу, чтобы блок масштабировал все роли, определенные в группе.
- Действие **changeSetting** используется для дросселирования приложения. Оно позволяет указать новое значение параметра в конфигурационном файле службы приложения. Блок изменяет этот параметр, и приложение реагирует путем считывания нового параметра. Код в приложении может использовать этот параметр для изменения его поведения. Например, он может отключить неважные функции или слегка ограничить работу интерфейса пользователя, чтобы лучше удовлетворить требования при увеличении спроса и нагрузки. Это обычно называется дросселированием приложения.
- Возможность выполнения созданного вами действия и развертывания его в виде сборки. Код сборки может выполнять любые соответствующие действия, например отправлять уведомления по электронной почте или запускать скрипты, чтобы изменить базу данных, развернутую на технологической платформе SQL Azure.

**Мнение По**

Вы можете использовать функциональный блок для автоматического масштабирования, чтобы заставить свое приложение автоматически менять поведение при изменении нагрузки и спроса. Блок может изменять параметры в конфигурационном файле службы, а приложение может реагировать на эти изменения, чтобы уменьшить нагрузку на базовую инфраструктуру.

Функциональный блок для автоматического масштабирования регистрирует события, связанные с действиями масштабирования и может отправлять уведомления по электронной почте в ответ на масштабирование роли или вместо масштабирования роли, если это необходимо. Вы также можете настроить некоторые аспекты работы блока, например планировщик, который контролирует мониторинг и масштабирование, и стабилизатор, который обеспечивает задержки между действиями, чтобы избежать повторяющихся колебаний и оптимизировать количество экземпляров с учетом временных ограничений.

Вы платите за каждый час использования экземпляра роли Windows Azure, развернутый вами, даже если используете его в течение всего нескольких минут данного часа. Стабилизатор функционального блока для автоматического масштабирования поможет снизить расходы, стимулируя действия расширения масштаба только в течение первых нескольких минут часа, и действия уменьшения масштаба — только в последние несколько минут часа. Вы можете указать эти интервалы, чтобы получить максимальную выгоду в течение того часа, за который платите.

**Примечание**

Дополнительные сведения о Microsoft Enterprise Library 5.0 Integration Pack для Windows Azure приведены на сайте http://entlib.codeplex.com/releases/view/75025.

### Руководство по использованию функционального блока для автоматического масштабирования

Следующие рекомендации помогут вам понять, как можно получить максимальные преимущества от использования функционального блока для автоматического масштабирования:

- Функциональный блок для автоматического масштабирования может указывать действия для нескольких целей в нескольких подписках Windows Azure. Служба, которая размещает целевые роли, и служба, которая размещает функциональный блок для автоматического масштабирования, могут находиться в разных подписках. Чтобы позволить блоку получить доступ к приложениям, необходимо указать идентификатор подписки Windows Azure, в которой размещаются целевые приложения, и сертификат управления, используемый для подключения к подписке.
- Можно использовать хранилище BLOB-объектов Windows Azure для хранения ваших правил и служебной информации. Это облегчает процесс обновления правил и данных при управлении приложением. Кроме того, если вы хотите реализовать специальные функции для загрузки и обновления правил, можно создать пользовательское хранилище правил.
- Вы должны определить ограничительное правило для каждого контролируемого экземпляра роли. Используйте ранжирование для каждого ограничительного правила или правила реагирования, которые вы определяете, чтобы контролировать приоритеты в случае перекрытия условий.
- Ограничительные правила не учитывают переход на летнее время. Они просто используют смещение времени в формате UTC, которое следует постоянно указывать.
- Используйте группы масштабирования, чтобы определить набор целевых ролей, что является одним из действий по упрощению правил. Это также позволит облегчить добавление и удаление ролей из действия без необходимости редактирования каждого правила.
- Попробуйте использовать среднее значение времени, равное получасу или часу, чтобы выровнять значения, которые возвращаются счетчиками производительности или другими показателями для получения более стабильных и надежных результатов. Вы можете считать данные производительности для любого размещенного приложения или службы. Это не должны быть данные, к которым применяется действие правила.
- Попробуйте включать и выключать правила вместо их удаления из конфигурации при настройке блока и временных изменениях приложения.
- Помните, что вам необходимо написать код, который инициирует механизм диагностики Windows Azure, когда ваша роль запускается и копирует данные в хранилище Windows Azure.
- Попробуйте использовать механизм поведения дросселирования, а также масштабирование количества ролей. Это обеспечивает более точный контроль способа реагирования приложения на изменения нагрузки и спроса. Помните, что может понадобиться 10–15 минут, чтобы новые развернутые экземпляры ролей начали обрабатывать запросы, тогда как изменения в поведении дросселирования проявятся значительно быстрее.
- Регулярно анализируйте регистрируемые блоком сведения о его деятельности, чтобы оценить, насколько хорошо правила соответствуют исходным требованиям, когда приложение работает в рамках требуемых ограничений и удовлетворяет всем обязательствам SLA, касающимся доступности и времени реакции. Уточняйте правила на основании этого анализа.

## Сокращение задержки сети при доступе к облачным приложениям с помощью диспетчера трафика Windows Azure

Диспетчер трафика Windows Azure — это служба Windows Azure, которая позволяет настроить маршрутизацию запросов и балансировку нагрузки на основании предварительно определенных политик и настраиваемых правил. Он предоставляет механизм маршрутизации запросов на несколько развертываний ваших приложений и служб в Windows Azure, независимо от расположения центра обработки данных. Приложения и службы могут быть развернуты в одном центре обработки данных или нескольких таких центрах.

Диспетчер трафика Windows Azure контролирует доступность и задержку сети для каждого приложения, которое настраивается в политике, на любом порте HTTP или HTTPS. Если он обнаруживает, что приложение работает в автономном режиме, он не будет перенаправлять запросы в это приложение. Однако он продолжит контролировать приложение с интервалом 30 секунд и начнет направлять к нему запросы на основании настроенной политики балансировки маршрутов, как только оно станет доступным.

Диспетчер трафика Windows Azure помечает, что приложение работает в автономном режиме, если оно три раза подряд не отвечает на запросы. Это означает, что общее время между отсутствием ответа от приложения и пометкой приложения о том, что оно работает в автономном режиме, равно умноженному на три контрольному интервалу, который вы задаете.

> В будущих выпусках диспетчера трафика Windows Azure вы сможете изменить интервал между контрольными проверками.

### Как диспетчер трафика Windows Azure направляет запросы

Диспетчер трафика Windows Azure является распознавателем DNS. При использовании диспетчера трафика Windows Azure веб-браузеры и службы, имеющие доступ к вашему приложению, будут отсылать запрос DNS диспетчеру трафика с целью получения IP-адреса конечной точки, к которой они будут подключаться, точно также, как они бы подключались к любому другому веб-сайту или ресурсу.

> **Мнение Бхарата**
>
> Диспетчер трафика Windows Azure не выполняет перенаправление HTTP и не использует любую другую технологию перенаправления на основе веб-обозревателя, так как это не работает с другими типами запросов, такими как запросы от интеллектуальных клиентов, которые получают доступ к веб-службам, указанным в вашем приложении. Вместо этого он выполняет функцию распознавателя DNS, которому клиент отправляет запросы, чтобы получить IP-адрес соответствующей конечной точки приложения. Диспетчер трафика Windows Azure возвращает IP-адрес развернутого приложения, который лучше всего соответствует настроенной политике и правилам.

Диспетчер трафика Windows Azure использует запрошенный URL-адрес, чтобы определить, какую политику следует применить, и возвращает IP-адрес как результат анализа правил и настроек конфигурации для этой политики. После этого веб-обозреватель пользователя или соответствующая служба подключается к этому IP-адресу, эффективно перенаправляя запросы на основании выбранной вами политики и заданных правил.

Это означает, что вы можете предложить пользователям единый URL-адрес, который связан с адресом вашей политики диспетчера трафика Windows Azure. Например, вы можете использовать запись CNAME, чтобы сопоставить URL-адрес, который вы предоставляете пользователям своего приложения, такой как **http://store.treyresearch.net**, в вашем собственном DNS-сервере или DNS-сервере ваших поставщиков

услуг Интернета, с точкой доступа и политикой своей политики диспетчера трафика Windows Azure. Если вы назвали пространство имен своего диспетчера трафика Windows Azure **treyresearch** и настроили политику для приложения **Orders** под названием **ordersapp**, необходимо сопоставить URL-адрес в DNS-сервере с адресом **http://ordersapp.treyresearch.trafficmanager.net**. Все запросы к DNS для **store.treyresearch.net** будут направлены к диспетчеру трафика Windows Azure, который осуществит необходимое перенаправление путем возврата IP-адреса соответствующего развернутого приложения. На рисунке 2 представлен этот сценарий.

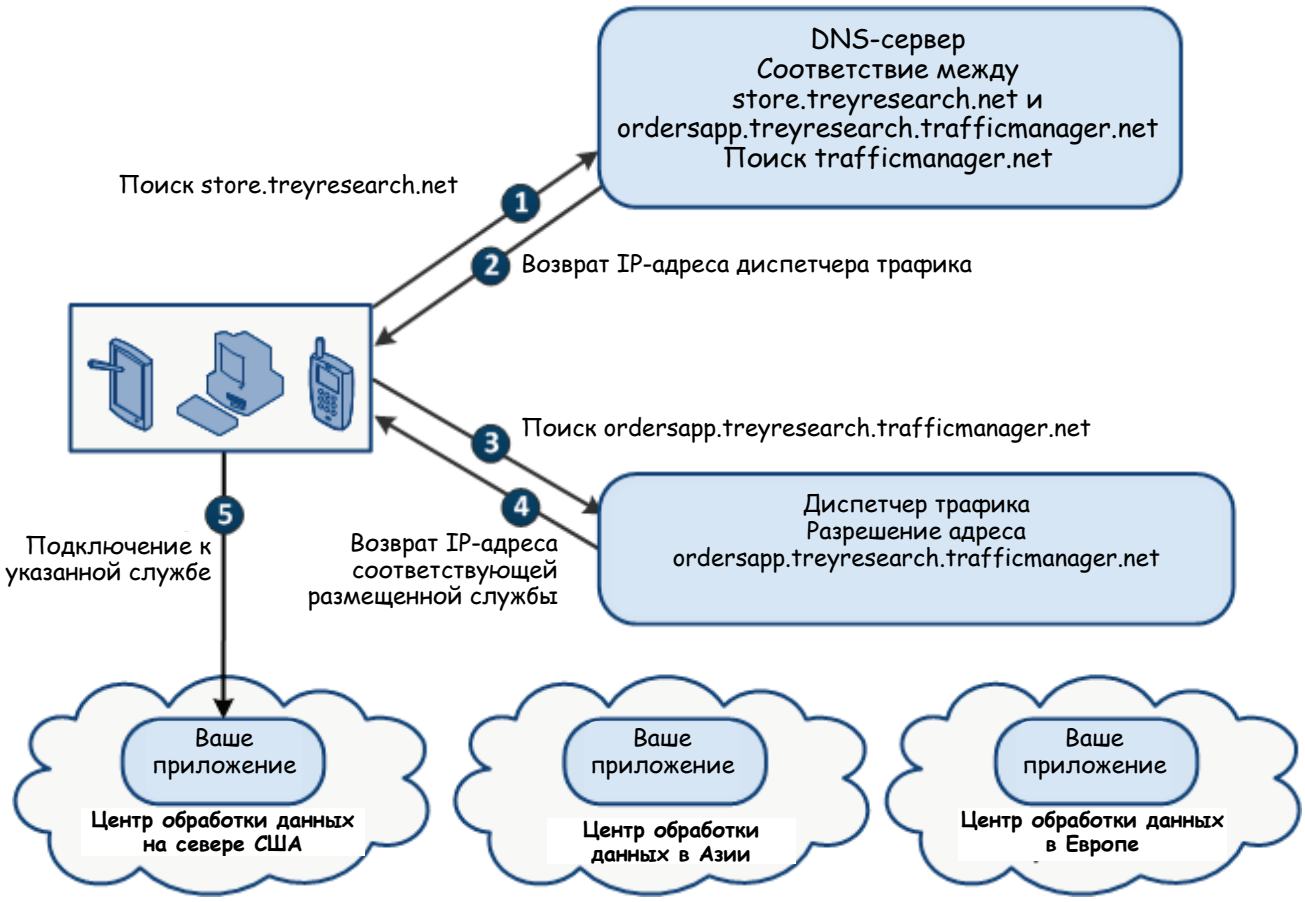



**Рисунок 2**

### Как диспетчер трафика Windows Azure выполняет маршрутизацию и перенаправление

Значение срока жизни (time-to-live, TTL) по умолчанию для ответов DNS, которые диспетчер трафика Windows Azure возвращает клиентам, составляет 300 секунд (пять минут). Когда этот интервал заканчивается, может потребоваться повторное разрешение любых запросов приложения клиента, а новый полученный адрес может использоваться для подключения к службе. Для тестирования можно уменьшить это значение, но в производственных сценариях следует использовать значение по умолчанию или большее значение.

Также следует помнить, что между клиентами и диспетчером трафика Windows Azure могут быть промежуточные DNS-серверы, которые могут кэшировать запись DNS в течение указанного вами времени. Однако клиентские приложения и веб-обозреватели часто кэшируют полученные ими DNS-записи, поэтому они не будут перенаправляться на другое развертывание приложения до окончания действия кэшированных записей.

> Глобальные эксперименты, проведенные командой, разработавшей диспетчер трафика Windows Azure, показывают, что в 97 % случаев обновления DNS обычно распространяются

во время срока жизни, указанного в записях. Изменения политики обычно распространяются на все распознаватели DNS диспетчера трафика Windows Azure в течение десяти минут. Вы можете проверить глобальное распространение записей DNS на одном из сайтов, например http://www.just-dnslookup.com/.

### Использование конечных точек мониторинга

Во время настройки политики диспетчера трафика Windows Azure вы указываете порт, имя и относительный путь для конечной точки, к которой диспетчер трафика Windows Azure должен получить доступ для тестирования реагирования приложения. По умолчанию номер порта 80 и «/», так что диспетчер трафика Windows Azure проверяет корневой каталог приложения. Если ответ HTTP «200 OK» приходит в течение десяти секунд, то диспетчер трафика Windows Azure предполагает, что базовая служба в данный момент работает.

Вы можете указать другое значение относительного пути и другое имя конечной точки мониторинга, если это необходимо. Например, если у вас есть страница, которая выполняет проверку всех функций в приложении, вы можете указать ее в качестве конечной точки мониторинга. Размещенные приложения и службы могут быть включены в более чем одну политику диспетчера трафика Windows Azure, поэтому рекомендуется давать согласованные имена и расположения для конечных точек мониторинга во всех приложениях и службах, чтобы относительные пути и имена можно было использовать в любой политике.

**Мнение Маркуса**

Если вы реализуете специальные страницы мониторинга в своем приложении, убедитесь, что они могут выдавать ответ на запрос в течение десяти секунд, чтобы диспетчер трафика Windows Azure не отметил, что они работают в автономном режиме. Также примите во внимание влияние на общую производительность приложения процессов, которые вы выполняете на странице мониторинга.

Если диспетчер трафика Windows Azure определил, что все указанные в политике службы работают в автономном режиме, то он будет рассматривать все эти службы как подключенные и будет продолжать раздавать IP-адреса на основе указанной вами политики. При таком подходе клиенты гарантированно будут получать IP-адреса в ответ на запрос DNS, даже если служба недоступна.

### Политики диспетчера трафика Windows Azure

На момент написания этой книги диспетчер трафика Windows Azure предлагал следующие три политики маршрутизации и балансировки нагрузки, хотя в будущем могут быть добавлены и другие:

- Политика **Performance (Производительность)** перенаправляет запросы от пользователей в приложение в ближайшем центре обработки данных. Это не обязательно может быть приложение в центре обработки данных, который находится ближе всего в географическом плане, но это то приложение, которое обеспечивает самую низкую задержку в сети. Это означает, что данная политика принимает во внимание производительность сетевых маршрутов между клиентом и центром обработки данных. Диспетчер трафика Windows Azure также обнаруживает проблемные приложения и не направляет к ним, а выбирает следующее ближайшее рабочее развертывание приложения.
- Политика **Failover (Отказоустойчивость)** позволяет настраивать список приоритетов приложений, и диспетчер трафика Windows Azure будет направлять запросы к первому приложению по списку, которое отвечает на запросы. Если данное приложение не работает, то диспетчер трафика Windows Azure перенаправит запросы к следующему по списку приложению

и так далее. Политика Failover может понадобиться, если вы хотите предоставить резервную копию для приложения, но копии приложений не предназначены или не настроены на постоянную работу. Вы можете развернуть различные версии одного приложения, например версии с ограниченными или альтернативными возможностями, для резервного копирования или использования в случае отказа основного приложения. Политика Failover также предоставляет возможность использования промежуточных или тестирующих приложений до выпуска, во время технического обслуживания или во время установки новой версии.

- Политика **Round Robin (Циклический перебор)** перенаправляет запросы к каждому приложению по очереди; но если она обнаруживает нерабочее приложение, то не запросы к нему не направляются. Эта политика выравнивает нагрузку на каждом приложении, но может не предоставлять пользователям наилучшее время отклика, так как она игнорирует относительное местоположение пользователя и центра данных.

Чтобы минимизировать сетевую задержку и максимизировать производительность, обычно используют политику Performance, чтобы перенаправлять все запросы от пользователей к приложению в ближайшем центре обработки данных. В следующих разделах рассказывается о политике Performance. Другие политики описаны в разделе «*Максимизация доступности для облачных приложений с помощью диспетчера трафика Windows Azure»* далее в этом документе.

> Помните, что при использовании политики Performance диспетчер трафика Windows Azure выполняет выбор приложения, основываясь на доступности и задержке сети, учитывая географическое расположение компьютера, с которого были отосланы запросы, и географическое расположение каждого настроенного приложения в политике (диспетчер трафика Windows Azure периодически запускает внутренние тесты в Интернете между определенными точками по всему миру и каждым центром обработки данных).
>
> Это означает, что ближайшее приложение не всегда будет ближайшим с точки зрения географии, хотя обычно это так и есть. Однако, если приложение в наиболее близком географически центре обработки данных не отвечает на запросы, диспетчер трафика Windows Azure может выбрать центр, который географически не будет наиболее близким.

### Рекомендации по применению диспетчера трафика Windows Azure

В следующем списке представлены основные рекомендации по применению диспетчера трафика Windows Azure:

- Когда вы даете имя своим базовым и другим службам, используйте шаблон имен, чтобы их потом было легче найти и идентифицировать в списке служб диспетчера трафика Windows Azure. Использование шаблона имен упрощает поиск необходимых служб по части имени. Включайте в имя службы имя центра обработки данных, чтобы было легче идентифицировать центр обработки данных, в котором расположена служба.
- Убедитесь, что диспетчер трафика Windows Azure может правильно отслеживать базовые приложения или службы. Если вы указываете страницу мониторинга вместо корневого пути «/» по умолчанию, убедитесь, что данная страница всегда возвращает статус HTTP «200 OK», точно определяет состояние приложения и отвечает в течение десяти секунд.
- Чтобы упростить управление и администрирование, используйте возможность включать и выключать политики, вместо того чтобы добавлять или удалять их. Создавайте столько политик, сколько вам нужно, но включайте только те из них, которые должны использоваться

в данный момент. Включайте и выключайте отдельные службы внутри политики, вместо того чтобы добавлять или удалять службы.

- Рекомендуется использовать диспетчер трафика Windows Azure в качестве рудиментарного решения для мониторинга, даже если вы не будете развертывать свое приложение в нескольких центрах обработки данных или вам не нужно перенаправлять запросы на другие развертывания. Вы можете настроить политику, которая будет включать все развертывания вашего приложения (включая другие приложения), используя символ «/» в качестве конечной точки мониторинга. Однако не следует направлять запросы клиента диспетчеру трафика Windows Azure для разрешения DNS. Напротив, клиенты подключаются к отдельным приложениям, используя заданные URL-адреса, которые вы указали для каждого из них в DNS. Затем вы сможете использовать веб-портал диспетчера трафика Windows Azure, чтобы увидеть, работают ли развертывания всех приложений.

---

### Рекомендации по применению диспетчера трафика Windows Azure для уменьшения задержки сети

Следующий список содержит рекомендации по применению диспетчера трафика Windows Azure для уменьшения задержки сети:

- Выберите политику Performance, чтобы пользователи автоматически перенаправлялись в тот центр обработки данных и к тому развертыванию приложения, которые гарантируют наилучшее время отклика.
- Убедитесь, что было развернуто достаточное количество экземпляров роли в каждом приложении для обеспечения достаточной производительности. Рассмотрите также возможность использования механизма, подобному тому, который реализован функциональным блоком для автоматического масштабирования (описано ранее в этом приложении), чтобы можно было автоматически развертывать дополнительные экземпляры при увеличении спроса.
- Проанализируйте, являются ли шаблоны спроса в каждом центре обработки данных цикличными или зависят от времени. У вас будет возможность развернуть меньшее количество экземпляров роли в определенные моменты, чтобы минимизировать текущие затраты (или даже удалить все экземпляры, чтобы пользователи перенаправлялись в другой центр обработки данных). Помимо этого, попробуйте использовать механизм, который был описан ранее в этом приложении, чтобы автоматически разворачивать и удалять экземпляры при изменении спроса.

---

Если все базовые приложения или службы в политике Performance работают в автономном режиме или недоступны (или нельзя проверить доступность из-за ошибки сети или какой-либо другой ошибки), диспетчер трафика Windows Azure будет себя вести так, как будто все они подключены. Он будет перенаправлять запросы, основываясь на своих внутренних измерениях задержки в глобальной сети в зависимости от местонахождения клиента, который посылает запрос. Это означает, что клиенты смогут получить доступ к приложению, если оно на самом деле подключено или сразу же, когда оно опять возобновит работу, без задержки, когда диспетчер трафика Windows Azure определит это и начнет перенаправлять пользователей, основываясь на измеренной задержке.

### Ограничения применения диспетчера трафика Windows Azure

В следующем списке определены некоторые ограничения, которые вы должны принять во внимание при применении диспетчера трафика Windows Azure:

- Все базовые приложения или службы, которые вы добавляете в политику диспетчера трафика Windows Azure, должны существовать в одной и той же подписке Windows Azure, хотя они могут быть в различных пространствах имен.
- Вы не можете добавить базовые приложения или службы, которые находятся на этапе подготовки; они должны быть запущены в рабочей среде. Однако вы можете выполнять виртуальный обмен IP-адресов (VIP), чтобы перемещать базовые приложения или службы в рабочую среду, не влияя на существующую политику диспетчера трафика Windows Azure.
- Все базовые приложения или службы должны предоставлять одинаковые операции и использовать HTTP или HTTPS через одинаковые порты, чтобы диспетчер трафика Windows Azure мог направлять запросы к любому из них. Если вы предоставляете определенную страницу в качестве конечной точки мониторинга, то она должна существовать в том же месте во всех развертываниях приложения, определенных в политике.
- Диспетчер трафика Windows Azure не проверяет корректность работы приложения; он только осуществляет проверку ответов HTTP «200 OK» от конечной точки мониторинга в течение десяти секунд. Если вы хотите выполнить больше тестов для проверки правильности работы приложения, вам следует предоставить определенную конечную точку мониторинга и указать ее в политике диспетчера трафика Windows Azure. Однако сначала убедитесь, что запрос мониторинга (который происходит каждые 30 секунд) чрезмерно не влияет на работу приложения или службы.
- Примите во внимание влияние перенаправления к различным развертываниям приложения на синхронизацию данных и кэширование. Пользователи могут направляться в центр обработки данных, в котором данные не полностью согласованы с данными в другом центре обработки данных.
- Учтите влияние перенаправления к различным развертываниям приложения на используемый вами подход к аутентификации. Например, если каждое развертывание использует отдельный экземпляр службы Windows Azure Access Control Service (ACS), то пользователям придется входить в систему каждый раз при перенаправлении в другой центр обработки данных.

## Максимизация доступности для облачных приложений с помощью диспетчера трафика Windows Azure

Диспетчер трафика Windows Azure предоставляет две политики, которые можно использовать для максимизации доступности ваших приложений. Вы можете использовать политику Round Robin для распространения запросов во все развертывания приложения, которые в данный момент отвечают на запросы (работающие приложения). Кроме того, вы можете использовать политику Failover, чтобы гарантировать, что резервное развертывание приложения будет получать запросы, на которые не смогло ответить главное приложение. Эти две политики позволяют использовать два очень разных подхода к максимизации доступности:

- Политика Round Robin позволяет вам масштабировать приложение во всех центрах обработки данных, чтобы достичь максимальной доступности. Запросы будут поступать на развертывание в активном центре обработки данных, и чем больше экземпляров роли вы настроите, тем меньше будет средняя нагрузка на каждый из них. Однако вам придется платить за каждую роль и развертывание приложения в каждом центре обработки данных, и вы должны быть осторожны в выборе количества развертываний экземпляров роли в каждом приложении и центре обработки данных.

**Мнение Бхарата**

Нет смысла использовать политику Round Robin, если вы разворачиваете свое приложение в одном центре обработки данных. Вы можете максимизировать доступность и масштабировать приложение просто путем добавления большего количества экземпляров роли. Однако политика Failover будет полезна, если вы выполняете развертывание в одном центре обработки данных, потому что эта политика позволяет вам определять резервные развертывания и копии развертываний приложения, которые могут отличаться от главного развертывания с главным приоритетом.

- Политика Failover позволяет вам разворачивать резервные и дублирующие версии своего приложения, которые получают запросы клиента только тогда, когда развертывания с большим приоритетом в списке работают в автономном режиме. В отличие от политик Performance и Round Robin, данная политика подходит для использования и в случае, когда вы разворачиваете приложение только в одном центре обработки данных, и в случае, когда вы выполняете развертывания приложения в нескольких центрах обработки данных. Однако вам придется платить за каждое развертывание приложения в каждом центре обработки данных, и вы должны быть осторожны в выборе количества экземпляров роли для развертывания в каждом центре обработки данных.

  Типичным сценарием для использования политики Failover является настройка соответствующего порядка приоритетов для одного или нескольких развертываний одной или разных версий приложения, чтобы всегда было доступно максимальное количество функций и широкий набор возможностей, даже если не работают службы и системы, от которых зависит приложение. Например, вы может развернуть резервную копию версии, которая будет принимать заказы клиента, когда система обработки заказов недоступна, но будет их надежно сохранять и информировать пользователя о задержке.

  Изменяя порядок приоритетов для использования соответствующей резервной версии в другом центре обработки данных или дублирующей версии с уменьшенной функциональностью в том же или в другом центре обработки данных, вы можете получить одновременно максимальную доступность и функциональность. На рисунке 3 показан пример данного подхода.

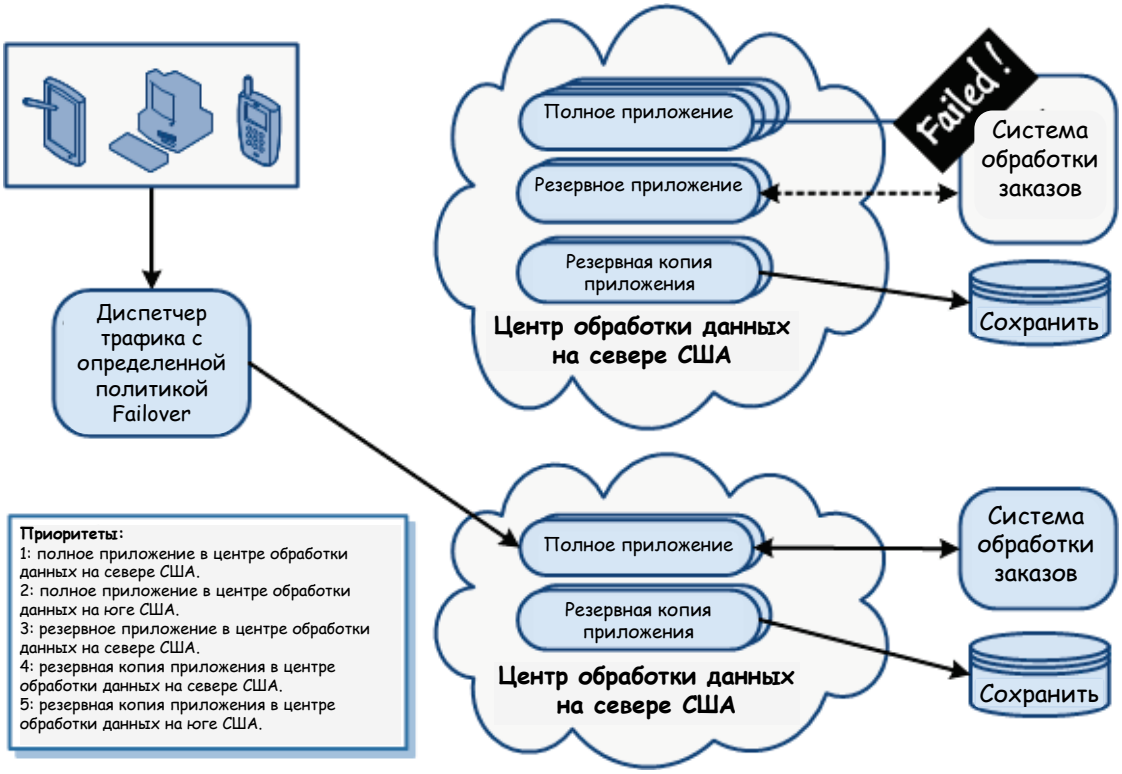



**Рисунок 3**

**Использование политики Failover для достижения максимальной доступности и функциональности**

### Рекомендации по применению диспетчера трафика Windows Azure для максимизации доступности

Следующий список содержит рекомендации по применению диспетчера трафика Windows Azure для максимизации доступности. Для получения дополнительной информации обратитесь к разделам «Рекомендации по применению диспетчера трафика Windows Azure» и «Ограничения применения диспетчера трафика Windows Azure» ранее в этом приложении.

- Выберите политику Round Robin, если вы хотите распространять запросы равномерно между всеми развертываниями приложения. Эта политика обычно не подходит, если вы разворачиваете приложение в центрах обработки данных, которые значительно отдалены географически, так как появится лишний трафик на большие расстояния. Также могут возникнуть проблемы во время синхронизации данных между центрами обработки данных, поскольку данные в каждом центре могут быть не согласованы по запросам от одного клиента. Тем не менее данная политика полезна для перевода служб в автономный режим работы в периоды технического обслуживания, тестирования и обновления.
- Выберите политику Failover, если вы хотите, чтобы запросы направлялись к одному развертыванию приложения, и перенаправлялись на другое развертывание только при отказе первого. Диспетчер трафика Windows Azure выбирает приложение, которое расположено вверху настроенного вами списка и в данный момент находится в рабочем состоянии. Эта политика обычно подходит для сценариев, в которых вы хотите предоставить резервные приложения или службы.

- Если вы используете политику Round Robin, убедитесь, что все развернутые приложения идентичны, чтобы пользователи не видели отличий в работе, независимо от того, куда они были перенаправлены.
- Если вы используете политику Failover, также рассмотрите возможность включения развертываний приложения, которые предоставляют ограниченную или другую функциональность и будут работать в случае недоступности служб или систем, от которых зависит приложение, чтобы предоставить пользователю возможность максимального взаимодействия.
- Проанализируйте еще возможность использования политики Failover или Round Robin во время проведения технического обслуживания, обновления приложений, а также при тестировании развернутых приложений. При необходимости можно включать и выключать отдельные приложения внутри политики, чтобы запросы направлялись только к работающим приложениям.
- Поскольку некоторые развертывания приложения будут не слишком загружены или не будут обрабатывать запросы клиентов (в зависимости от выбранной вами политики), рекомендуем рассмотреть возможность использования механизма, подобному тому, который предоставляется функциональным блоком для автоматического масштабирования, о котором рассказывалось здесь ранее, чтобы управлять количеством экземпляров роли для каждого приложения, развернутого в каждом центре обработки данных, для уменьшения текущих затрат.

Если все базовые приложения или службы в политике Round Robin работают в автономном режиме или недоступны (или их доступность невозможно проверить из-за проблем с сетью или каких-либо других проблем), диспетчер трафика Windows Azure будет работать так, как будто все они подключены к сети, и будет продолжать направлять запросы в каждое приложение по очереди. Если все приложения в политике Failover работают в автономном режиме  или недоступны, диспетчер трафика Windows Azure будет действовать так, как будто первое приложение в заданном списке подключено к сети, и будет направлять все запросы к нему.

**Примечание**

Более подробно о диспетчере трафика Windows Azure можно прочесть в статье «Windows Azure Traffic Manager» на сайте http://msdn.microsoft.com/en-us/gg197529.

## Оптимизация времени отклика и пропускной способности облачных приложений с помощью службы Windows Azure Caching

Служба Windows Azure Caching предоставляет масштабируемый и надежный механизм, позволяющий сохранять часто используемые данные физически ближе к вашим приложениям и службам. Служба Windows Azure Caching работает в облаке, и вы можете кэшировать данные в том же центре обработки данных, в котором размещен ваш код. Если вы разворачиваете службы более чем в одном центре обработки данных, то необходимо создать отдельный кэш в каждом центре обработки данных, и каждая служба должна получать доступ только к одному близлежащему кэшу. Таким образом, вы можете снизить расходы, связанные с повторяющимся доступом к удаленным данным, сократить задержку сети, возникающую из-за доступа к удаленным данным, и улучшить время отклика для приложений, обращающихся к этим данным.

Однако за кэширование также надо платить. Кэширование данных означает создание одной или нескольких копий данных, и поскольку вы делаете эти копии, то должны знать, что происходит при

изменении этих данных. Любые обновления должны реплицироваться во всех копиях, однако распространение обновлений по системе требует некоторого времени. Особенно это актуально для Интернета, где вы также должны учитывать возможность возникновения ошибок сети, которые могут помешать быстрому распространению обновлений. Таким образом, хотя кэширование и улучшает время отклика для многих операций, оно может привести к несогласованности, если два экземпляра одного элемента данных будут различны. Следовательно, приложения, которые эффективно используют кэширование, должны быть разработаны так, чтобы у них была возможность обработки устаревших данных, которые в итоге станут согласованными.

Не используйте службу Windows Azure Caching для кода, который выполняется локально, так как это не улучшит производительность вашего приложения в этой среде. На самом деле, это, скорее всего, замедлит работу системы в связи с задержками в сети, возникающими при подключении к кэшу в облаке. Если вам нужно реализовать кэширование для локальных приложений, следует рассмотреть возможность использования службы Windows Server AppFabric Caching. Более подробная информация представлена в статье «Windows Server AppFabric Caching Features» на сайте http://msdn.microsoft.com/en-us/library/ff383731.aspx.

**Мнение Бхарата**

Служба Windows Azure Caching предназначена в основном для запуска кода в облаке, например, запуска веб- и рабочих ролей. Для получения максимальной пользы от службы Windows Azure Caching ее следует реализовать в том же центре обработки данных, где находится ваш код.

### Контроль использования и изменение размеров кэша Windows Azure

Windows Azure Caching — это служба, которая поддерживается и управляется корпорацией Microsoft; вам не нужно устанавливать дополнительное программное обеспечение или внедрять инфраструктуру внутри вашей организации, чтобы использовать данную службу. Администратор может легко предоставить экземпляр службы кэширования, используя портал управления Windows Azure. Данный портал позволяет администратору выбирать месторасположение службы кэширования и указывать ресурсы, доступные для кэширования. Вы указываете предоставляемые ресурсы путем выбора размера кэша. Служба Windows Azure Caching поддерживает набор заданных размеров кэша от 128 МБ до 4 ГБ. Обратите внимание, что чем больше размер кэша, тем выше месячная плата.

Размер кэша также определяет количество других квот. Целью этих квот является обеспечение справедливого использования ресурсов, а также наложение ограничений на количество операций чтения и записи в кэш за час, пропускную способность в час и количество одновременных соединений. Кроме того, чем больше кэш, тем больше доступных ресурсов. Например, если вы выбрали размер кэша 128 МБ, то вы сможете выполнять до 40 000 операций чтения и записи кэша, который занимает до 1400 МБ трафика (мегабайт в час) и охватывает до 10 одновременных соединений в час. При выборе 4 ГБ кэша вы можете выполнять до 12 800 000 операций чтения и записи, занимающих 44 800 МБ трафика, а также получите поддержку 160 одновременно работающих пользователей в час.

**Примечание**

Значения, указанные здесь, корректны на момент написания статьи, но эти квоты постоянно пересматриваются и могут быть изменены в будущем. Более подробно о текущих ограничениях рабочих квот и ценах на них можно прочитать на сайте http://msdn.microsoft.com/en-us/library/hh697522.aspx.

Вы можете создать столько кэшей, сколько требуется приложению, и они могут быть разных размеров. Однако для максимальной эффективности затрат необходимо тщательно оценить объем кэш-памяти,

которая необходима вашим приложениям, и объем операций, которые они выполняют. Вы также должны рассмотреть вопрос о продолжительности нахождения объектов в кэше. По умолчанию объекты устаревают через 48 часов, после чего они удаляются. Вы не можете изменить этот период действия для кэша в целом, но можно изменять его для отдельных объектов, если вы храните их в кэше. Однако следует помнить, что чем дольше объект находится в кэше, тем больше вероятность того, что он будет отличаться от исходного источника данных (именуемого «надежным» источником), на основании которого он был создан.

Чтобы оценить объем памяти, необходимый для каждого типа объектов, которые вы будете хранить, выполните следующие действия:

1. Измерьте размер в байтах типичного экземпляра объекта (сериализуйте объекты, используя класс **NetDataContractSerializer**, и запишите их в файл).
2. Добавьте небольшие накладные расходы (около 1 %) для метаданных, которые служба кэширования сопоставляет с каждым объектом.
3. Округлите это значение до следующего ближайшего значения 1024 (кэш выделяется на объекты, разделенные на сегменты размером 1 КБ).
4. Умножьте это число на максимальное количество экземпляров, которые вы собираетесь кэшировать.

Сложите результаты для каждого типа объекта, чтобы получить требуемый размер кэш-памяти. Отметим, что портал управления позволяет контролировать текущие и пиковые размеры кэш-памяти, и вы можете изменить размер кэша после его создания без остановки и перезагрузки любой из своих служб. Однако это изменение не делается мгновенно, вы можете запросить изменение размера кэша только один раз в день. Кроме того, вы можете увеличить размер кэша, не теряя объекты из кэша, но если вы уменьшите размер кэша, то некоторые объекты могут быть потеряны.

Вы также должны внимательно оценить другие элементы квоты кэша, и, если необходимо, выбрать больший размер кэша, даже если вам не требуется указанный объем памяти. Например, если вы превысили количество операций чтения и записи в кэше, разрешенных в один час, то все последующие операции чтения и записи завершатся ошибкой. Аналогично, если вы превысили квоту пропускной способности, то приложения будут получать отказ в следующий раз, когда они попытаются получить доступ к кэшу. Если вы достигли предела количества подключений, то ваши приложения не смогут устанавливать новые соединения, пока одно или несколько существующих подключений не закроются.

**Мнение Маркуса**

Служба Windows Azure Caching позволяет приложению использовать пул соединений. Когда настроен пул соединений, то такой же пул соединений используется одним экземпляром приложения. Использование пулов соединений может улучшить производительность приложений, использующих службу кэширования. Но вы должны продумать, как это повлияет на общие требования к соединениям в зависимости от количества экземпляров приложения, которые могут быть запущены одновременно. Более подробная информация представлена в статье «Understanding and Managing Connections in Windows Azure» на сайте http://msdn.microsoft.com/en-us/library/hh552970.aspx.

Вы не обязаны использовать один кэш в приложении. Каждый экземпляр службы Windows Azure Caching принадлежит к пространству имен службы, и вы можете создавать несколько пространств имен службы со своими собственными кэшами в одном центре обработки данных. Все кэши могут иметь

различные размеры, так что вы можете разделить данные в соответствии с профилем кэша. Небольшие объекты, доступ к которым осуществляется редко, могут храниться в кэше размером 128 МБ, в то время как более крупные объекты, доступ к которым осуществляется постоянно большим количеством одновременно запущенных экземпляров приложений, можно хранить в кэше размером 2 или 4 ГБ.

### Реализация служб, которые совместно используют данные, с помощью службы Windows Azure Caching

Служба Windows Azure Caching реализует кэш-память, расположенную на кэш-сервере в центре обработки данных Windows Azure, который может использоваться одновременно несколькими службами. Это идеальное решение для хранения неизменяющихся или медленно изменяющихся данных, таких как каталог продукции или список адресов клиентов. Копирование этих данных из базы данных в общий кэш поможет снизить нагрузку на базу данных, а также улучшить время отклика приложений, которые используют эти данные. Такой подход также помогает создавать хорошо масштабируемые и устойчивые службы, которые проявляют сниженное сходство с приложениями, которые их вызывают. Например, приложение может вызвать операцию в службе, которая реализована в виде веб-роли Windows Azure, для получения информации о конкретном клиенте. Если эта информация копируется в общий кэш, то же приложение может сделать последующие запросы, чтобы получить и сохранить информацию об этом клиенте, независимо от того, направляются ли эти запросы в тот же экземпляр веб-роли Windows Azure, или в другой. Если количество запросов клиентов увеличивается со временем, то могут запускаться новые экземпляры веб-роли для их обработки, так как система легко масштабируется. На рисунке 4 показана эта архитектура, в которой локальные приложения используют службы, предоставляемые экземплярами веб-роли. Локальное приложение может быть направлено в любой экземпляр веб-роли, и те же кэшированные данные остаются доступными.

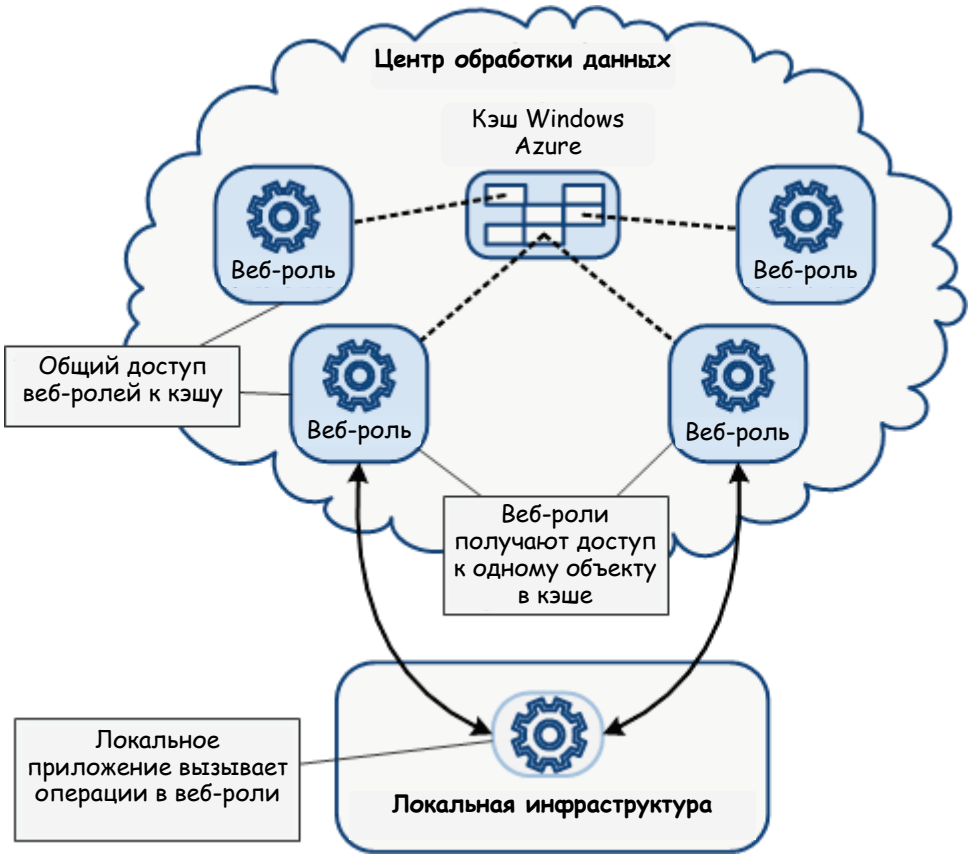



**Рисунок 4**

**Использование службы Windows Azure Caching для обеспечения масштабируемости**

Веб-приложения получают доступ к общему кэшу с помощью API службы Windows Azure Caching. Эти API оптимизированы для поддержки *ориентированного на кэш* программного шаблона. Веб-приложение может посылать запрос в кэш, чтобы найти объект и, если такой объект там есть, он может быть извлечен. Если объект в настоящее время не хранится в кэше, то веб-приложение может получить данные для объекта из надежного хранилища (например, базы данных SQL Azure), создать объект, используя эти данные, а затем сохранить его в кэше.

> **Мнение Маркуса**
>
> Объекты, которые вы храните в кэше, должны быть сериализуемыми.

Вы можете указать кэш, к которому хотите подключиться, либо программно, либо путем предоставления информации о соединении в разделе **dataCacheClient** в конфигурационном файле веб-приложения. Можно сгенерировать необходимые данные конфигурации клиента на портале управления, а затем скопировать эту информацию непосредственно в конфигурационный файл. Дополнительная информация о настройке веб-приложений для использования службы Windows Azure Caching представлена в статье «How to: Configure a Cache Client using the Application Configuration File for Windows Azure Caching» на сайте http://msdn.microsoft.com/en-us/library/windowsazure/gg278346.aspx.

Как описано в разделе «*Контроль использования и изменение размеров кэша Windows Azure*», администратор указывает ресурсы, доступные для данных кэша после его создания. Если памяти станет недостаточно, служба Windows Azure Caching будет высвобождать данные, начиная с тех, которые давно не используются. Однако кэшированные объекты также могут иметь свои собственные независимые сроки жизни, и разработчик может установить срок для кэширования объекта, когда он сохраняется. По истечении этого времени объект будет удален, а соответствующие ресурсы освобождены.

> **Мнение Маркуса**
>
> Служба Windows Azure Caching не посылает вашим приложениям уведомления про удаление или истечение срока хранения объектов в кэше, так что будьте осторожны.

Более подробную информацию об использовании API службы Windows Azure Caching вы можете прочесть в статье «Developing Cache Clients in Windows Azure» на сайте http://msdn.microsoft.com/en-us/library/windowsazure/gg278342.aspx.

### Обновление кэшированных данных

Веб-приложения могут изменять объекты в кэше, но знайте, что если кэш является общим, то более чем один экземпляр приложения может попытаться обновить одну и ту же информацию. Может возникнуть проблема, аналогичная проблеме обновления, которая может произойти в любом сценарии использования общих данных. Чтобы помочь справиться с этой ситуацией, API службы Windows Azure Caching поддерживают два режима обновления кэшированных данных:

- **Оптимистичный, с управлением версиями**.

  Все кэшированные объекты могут иметь соответствующий номер версии. Когда веб-приложение обновляет данные для объекта, который извлекается из кэша, оно может проверить номер версии объекта в кэше перед сохранением изменений. Если номер версии совпадает, то приложение сохраняет данные. В противном случае в веб-приложении должно учитываться, что другой экземпляр приложения уже изменил этот объект, оно должно получить новые данные и разрешить конфликт с использованием любой логики, которая подходит для данной бизнес-обработки (возможно стоит предоставить пользователю обе версии данных и спросить, какую из них нужно сохранить). После обновления объекта с ним необходимо ассоциировать новый уникальный номер версии, когда он возвращается в кэш.

- **Пессимистичный, с блокированием**.

  Оптимистичный подход используется в основном тогда, когда шансы конфликта невелики, и, хотя он прост в теории, его реализация неизбежно предполагает определенную степень сложности для обработки возможных условий состязания, которые могут возникнуть. Пессимистичный подход построен на противоположной точке зрения. Предполагается, что несколько экземпляров веб-приложения попытаются одновременно изменить одни и те же данные, поэтому данные блокируются во время извлечения из кэша, чтобы предотвратить проблему обновления. Когда обновленный объект возвращается в кэш, блокировка снимается. Если веб-приложение пытается получить и заблокировать объект, который уже заблокирован другим экземпляром приложения, то оно получит отказ (объект заблокирован не будет). По истечении короткого периода времени это веб-приложение сможет снова попробовать получить и заблокировать объект. Хотя такой подход и гарантирует согласованность кэшированных данных, в идеале любые операции обновления должны быть очень быстрыми и соответствующие блокировки должны происходить в очень короткий срок, чтобы минимизировать возможность столкновения и не заставлять веб-приложения ожидать в течение длительного периода, так как это может повлиять на время отклика и пропускную способность приложения.

  **Мнение Маркуса**

  Приложение задает продолжительность блокировки при получении данных. Если приложение не снимает блокировку в течение этого периода, то блокировка снимается службой Windows Azure Caching. Эта функция предназначена для предотвращения ситуации, когда в приложении возникает постоянная ошибка блокировки данных. Вы должны установить такой период, который обеспечит вашему приложению достаточное время для выполнения операции обновления, но этот период должен быть не настолько долгим, чтобы заставлять другие экземпляры ожидать доступа к этим данным в течение слишком долгого времени.

Если вы размещаете несколько экземпляров службы Windows Azure Caching в разных центрах обработки данных, то проблема обновления становится еще более острой, так как, возможно, придется синхронизировать кэш не только с надежным источником данных, но и с другими кэшами, расположенными в разных местах. Синхронизация обязательно порождает сетевой трафик, на который в свою очередь влияют задержки, а иногда и ненадежность Интернета. В большинстве случаев предпочтительнее непосредственно обновлять данные надежного источника данных, удалять данные из кэша в том же центре обработки данных, в котором находится веб-приложение, и ждать естественного истечения срока хранения кэшированных данных в каждом центре обработки данных, когда можно будет заполнить кэш повторно из надежного источника данных.

Логика, по которой обновляется надежный источник данных, должна минимизировать вероятность перезаписи изменения, сделанного другим экземпляром приложения, возможно, включая сведения о версии в данные и проверку того, что при выполнении обновления номер версии не изменился.

Целью удаления данных из кэша, а не простого их обновления, является снижение вероятности потери изменений, сделанных другими экземплярами веб-приложения в других местах, а также сведение к минимуму шансов возникновения несоответствий, в случае неудачного обновления надежного хранилища данных. В следующий раз, когда эти данные потребуется, соответствующая версия данных будет считываться из надежного хранилища данных и копироваться в кэш.

Если необходимо незамедлительное обновление в различных местах хранения данных, то вы можете реализовать пользовательское решение с помощью топиков шины интеграции, реализовывая варианты шаблонов, описанные в разделе «Репликация и синхронизация данных с помощью топиков и подписок шины интеграции» в «Приложении А. Репликация, распространение и синхронизация данных».

Оба подхода иллюстрируются ниже в этом приложении, в разделе «Рекомендации по использованию кэширования Azure».

**Мнение Джаны**

Включение службы Windows Azure Caching в веб-приложение должно быть осознанным этапом проектирования, так как это напрямую влияет на обновление логики приложения. В какой-то степени вы можете скрыть эту сложность и способствовать повторному использованию путем создания уровня кэширования в виде библиотеки и абстрагирования кода, который извлекает и обновляет кэшированные данные, но все равно эта логика должна быть где-то реализована.

Особенностью службы Windows Azure Caching является то, что необходимо включить в веб-приложения комплексную обработку исключений и логику восстановления. Пример:

- Условие состязания существует в простой реализации ориентированного на кэш шаблона, который позволяет двум экземплярам веб-приложения пытаться добавлять одинаковые данные в кэш. В зависимости от способа реализации логики, которая хранит данные в кэше, она может привести к тому, что один экземпляр перезапишет данные, которые были ранее добавлены другим экземпляром (если вы используете метод кэша **Put**), или она может привести к сбою экземпляра с исключением **DataCacheException** (если вы используете метод кэша **Add**). Дополнительная информация представлена в статье «Add an Object to a Cache» на сайте http://msdn.microsoft.com/en-us/library/ee790846.aspx.
- Приготовьтесь перехватывать исключения при попытке получения заблокированных данных, а также реализовать соответствующий механизм, позволяющий повторно использовать операцию считывания через определенный промежуток времени, возможно, с помощью функционального блока для обработки неустойчивых неисправностей.
- Вы должны обрабатывать сбои, чтобы получить данные из службы Windows Azure Caching, так как кэш пропускает и разрешает веб-приложению извлекать элементы из надежного источника данных.
- Если приложение превышает квоты, связанные с размером кэша, то оно больше не сможет подключиться к кэшу. Вы должны регистрировать такие исключения, и если они часто повторяются, администратор должен рассмотреть вопрос об увеличении размера кэша.

### Создание локального кэша

Как и в случае с общим кэшем, можно настроить веб-приложение так, чтобы создавался собственный локальный кэш. Цель локального кэша заключается в оптимизации повторяющихся запросов на чтение кэшированных данных. Локальный кэш хранится в памяти приложения, поэтому доступ к нему осуществляется быстрее. Он работает в тандеме с общим кэшем. Если локальный кэш включен, когда приложение запрашивает объект, то клиентская библиотека кэширования сначала проверяет, доступен ли этот объект локально. Если он доступен, то ссылка на этот объект возвращается немедленно без обращения к общему кэшу. Если объект не найден в локальном кэше, то клиентская библиотека кэширования получает данные из общего кэша и сохраняет копию этого объекта в локальном кэше.

В дальнейшем приложение ссылается на объект из локального кэша. Конечно же, если объект не найден в общем кэше, приложение должно извлечь объект из надежного источника данных.

После того как элемент кэширован локально, локальная версия этого элемента будет использоваться до истечения срока его хранения или удаления из кэша. Однако вполне возможно, что другое приложение может изменить данные в общем кэше. В этом случае приложение, которое использует локальный кэш, не увидит этих изменений до тех пор, пока локальная версия элемента не будет удалена из локального кэша. Поэтому, несмотря на то что использование локального кэша может значительно улучшить время отклика для приложения, локальный кэш может очень быстро стать несогласованным, если информация в общем кэше изменяется. По этой причине вы должны настроить локальный кэш для хранения объектов только в течение короткого периода времени до их обновления. Если данные, содержащиеся в общем кэше, очень динамичны и важна их согласованность, то отдается предпочтение использованию общего кэша, а не локального.

После того как элемент скопирован в локальный кэш, приложение может получить доступ к нему с помощью тех же интерфейсов API службы Windows Azure Caching и программной модели, которые работают в общем кэше. Взаимодействие с локальным кэшем полностью прозрачно. Например, если приложение изменяет элемент и помещает обновленную запись обратно в кэш, интерфейсы API службы Windows Azure Caching обновляют локальный кэш и копию в общем кэше.

На локальный кэш не распространяются те же квоты ресурсов, что для общего кэша, управляемого службой Windows Azure Caching. Можно указать максимальное число объектов, которые могут содержаться в кэше при его создании, а хранилище для кэша выделяется непосредственно из памяти, доступной для приложения.

**Мнение Маркуса**

Вы включаете локальное кэширование путем заполнения члена **LocalCacheProperties** объекта **DataCacheFactoryConfiguration**, который используется для управления настройками клиента кэша. Вы можете выполнить эту задачу программно или декларативно в конфигурационном файле приложения. Можно указать размер кэша и период хранения кэшированных элементов по умолчанию. Дополнительная информация представлена в статье «Enable Windows Server AppFabric Local Cache (XML)» на сайте http://msdn.microsoft.com/en-us/library/ee790880.aspx.

### Кэширование состояния сеанса веб-приложения

Для веб-приложений и служб ASP.NET служба Windows Azure Caching позволяет использовать поставщика состояния сеанса **DistributedCacheSessionStateStoreProvider**. С этим поставщиком вы можете хранить состояние сеанса в кэше Windows Azure. Использование кэша Windows Azure для хранения состояния сеанса предоставляет ряд преимуществ:

- Можно совместно использовать состояние сеанса различными экземплярами веб-приложений ASP.NET, обеспечивая улучшенную масштабируемость.
- Поддерживается одновременный доступ к одним и тем же данным о состоянии сеанса для нескольких считывателей и одного модуля записи.
- Можно также использовать сжатие для сохранения памяти и пропускной способности.

Этот поставщик может быть настроен через код или с помощью конфигурационного файла приложения. Вы можете генерировать данные о конфигурации с помощью портала управления и копировать эту информацию непосредственно в конфигурационный файл. Дополнительная информация представлена

в статье «How to: Configure the ASP.NET Session State Provider for Windows Azure Caching» на сайте http://msdn.microsoft.com/en-us/library/windowsazure/gg278339.aspx.

После настройки поставщика, вы можете получить к нему программный доступ через объект **Session**, используя тот же код, как и обычное веб-приложение ASP.NET; вам не нужно вызывать API службы Windows Azure Caching.

### Кэширование выводимых данных в формате HTML

Класс **DistributedCacheOutputCacheProvider**, доступный для службы Windows Azure Caching, реализует кэширование выводимых данных для веб-приложений. С помощью этого поставщика, вы можете создавать масштабируемые веб-приложения, которые используют преимущества службы Windows Azure Caching для кэширования HTTP-ответов, которые они создают для веб-страниц, возвращаемых в клиентские приложения, а также этот кэш может использоваться несколькими экземплярами приложения. Этот поставщик имеет несколько преимуществ по сравнению с обычным кэшем выводимых данных процесса, в том числе:

- Можно кэшировать большие объемы выводимых данных.
- Кэш выводимых данных хранится вне рабочего процесса, в котором запущено веб-приложение, и он не теряется при перезагрузке веб-приложения.
- Он позволяет использовать сжатие для сохранения памяти и пропускной способности.

Опять же, вы можете генерировать данные для настройки этого поставщика с помощью портала управления и копировать эту информацию непосредственно в конфигурационный файл приложения. Дополнительная информация представлена в статье «How to: Configure the ASP.NET Output Cache Provider for Windows Azure Caching» на сайте http://msdn.microsoft.com/en-us/library/windowsazure/gg185676.aspx.

Как и класс **DistributedCacheSessionStateStoreProvider**, класс **DistributedCacheOutputCacheProvider** является полностью прозрачным. Если ваше приложение ранее использовало кэширование выводимых данных, вам не придется делать изменения в коде.

### Рекомендации по использованию службы Windows Azure Caching

Ниже описано несколько наиболее распространенных сценариев использования службы Windows Azure Caching:

- **Веб-приложения и службы, работающие в облаке, требуют быстрого доступа к данным. Эти данные часто запрашиваются, но редко изменяются. Эти же данные могут запрашиваться всеми экземплярами веб-приложений и служб.**

  Это идеальный случай для использования службы Windows Azure Caching. В этом простом сценарии вы можете настроить службу Windows Azure Caching, работающую в том же центре обработки данных, в котором размещены веб-приложения и службы (в виде веб- или рабочих ролей). Каждое веб-приложение или служба может создать ориентированный на кэш шаблон, если ему необходим элемент данных. Оно может попытаться извлечь элемент из кэша, а если он не найден, то извлечь его из надежного хранилища данных и скопировать в кэш. Если данные являются статическими, а кэш настроен с достаточным объемом памяти, то для каждого элемента при кэшировании можно указать длительный период хранения в кэше. Объекты, представляющие данные, которые могут меняться в надежном хранилище данных, должны кэшироваться с более коротким периодом хранения. Период должен отображать частоту изменения данных и срочность, с которой приложение должно получать доступ к последней обновленной информации.

**Мнение Маркуса**

Чтобы использовать службу Windows Azure Caching наилучшим образом, кэшируйте только данные, которые редко меняются.

На рисунке 5 показана возможная структура такого решения. В этом примере ряду веб-приложений, реализованных в виде веб-ролей, которые размещены в разных центрах обработки данных, требуется доступ к адресам клиентов, содержащимся в базе данных SQL Server, которая расположена локально в организации. Для сокращения сетевой задержки, связанной с повторяемыми запросами к тем же данным через Интернет, информация, используемая веб-приложениями, кэшируется с помощью службы Windows Azure Caching. Каждый центр обработки данных содержит отдельный экземпляр службы кэширования, а веб-приложения имеют доступ только к кэшу, который находится в том же центре обработки данных. Веб-приложения запрашивают только адреса клиентов, хотя другие приложения, работающие локально, иногда могут вносить изменения. Период хранения для каждого элемента в кэше равен 24 часам, поэтому любые изменения, внесенные в эти данные, в итоге будут видны веб-приложениям.

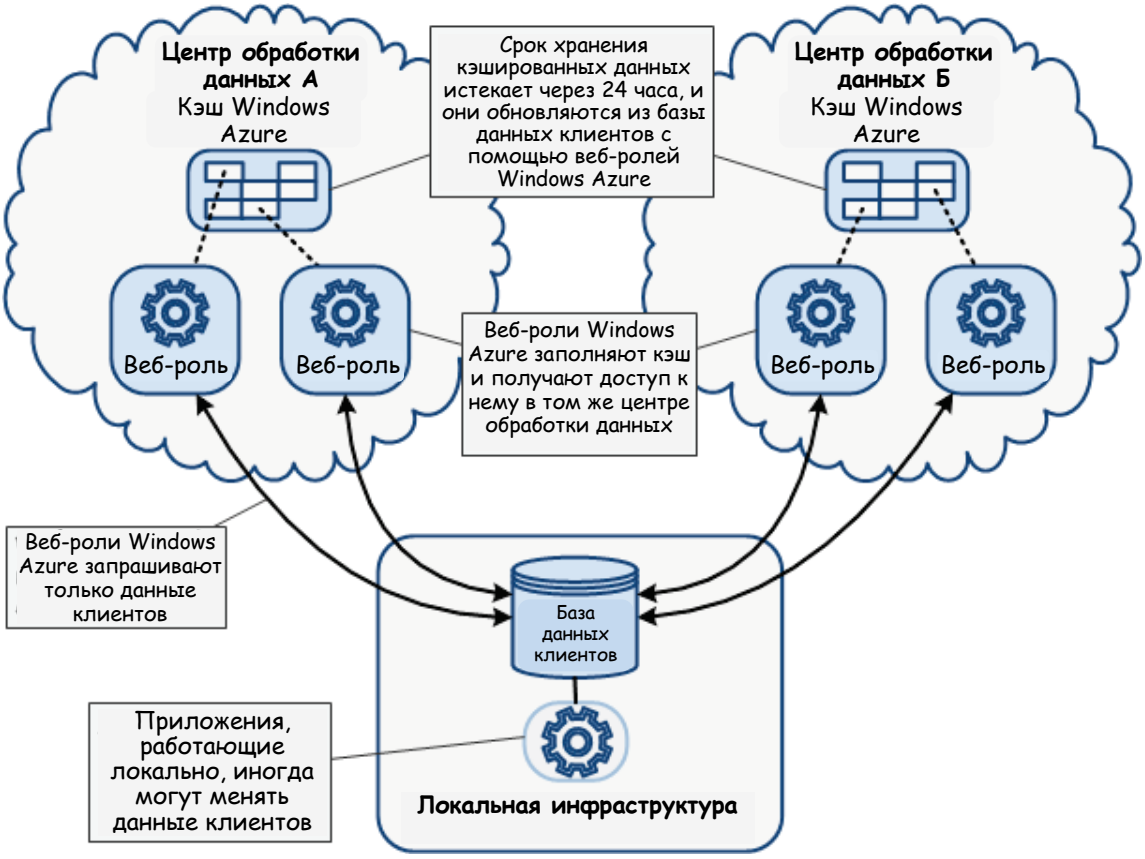



**Рисунок 5**

**Кэширование статических данных для сокращения сетевых задержек в веб-приложениях**

- **Веб-приложения и службы, работающие в облаке, требуют быстрого доступа к общим данным и могут часто изменять эти данные.**

Этот сценарий является потенциально сложным расширением предыдущего случая в зависимости от расположения данных, частоты обновлений, распределения веб-приложений

и служб, а также срочности, с которой обновления должны быть видны этим веб-приложениям и службам.

В самом простом случае, когда веб-приложению необходимо обновить объект, оно извлекает элемент из кэша (делая выборку из надежного хранилища данных, если это необходимо), изменяет этот элемент в кэше и вносит соответствующие изменения в надежное хранилище данных. Однако этот процесс состоит из двух этапов, а чтобы свести к минимуму шансы возникновения условий состязания, все обновления должны выполняться в том же порядке, в котором выполняются эти этапы. В зависимости от вероятности конфликтующего обновления, выполняемого активным экземпляром приложения, вы можете реализовать либо оптимистичную, либо пессимистичную стратегию обновления кэша, как описано ранее в разделе «Обновление кэшированных данных». Этот процесс показан на рисунке 6. В этом примере локальная база данных клиентов является надежным хранилищем данных.

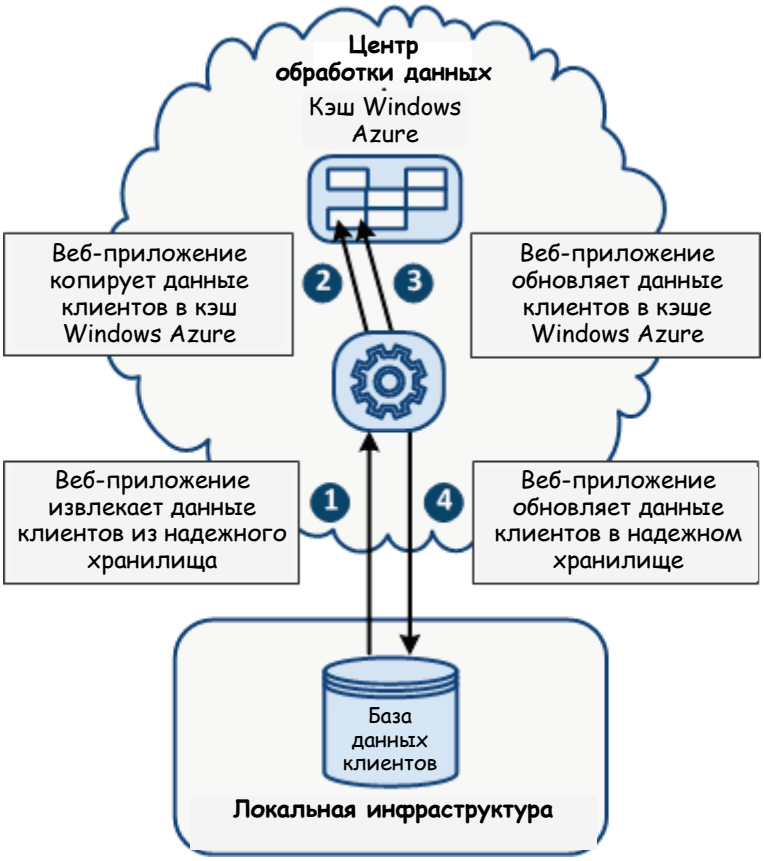



**Рисунок 6**

### Обновление данных в кэше и надежном хранилище данных

Описанный подход годится для решения, которое разворачивается в одном центре обработки данных. Однако, если ваши веб-приложения и службы размещены в нескольких местах, то следует реализовать кэш в каждом центре обработки данных. В таком случае обновления должны быть тщательно синхронизированы и скоординированы для всех центров обработки данных, а все копии кэшированных данных должны быть изменены. Как описано в разделе «Обновление кэшированных данных», у вас есть, по крайней мере, два варианта для решения этой проблемы:

- Обновите только надежное хранилище данных и удалите элемент из кэша в центре обработки данных, в котором размещено веб-приложение. В итоге срок хранения кэшированных данных в любом другом центре обработки данных закончится, и они

будут удалены из кэша. В следующий раз, когда эти данные потребуются, они будут получены из надежного хранилища и использованы для повторного заполнения кэша.

- Реализуйте собственное решение, используя топики шин интеграции, как описано в разделе «Репликация и синхронизация данных с помощью топиков и подписок шины интеграции» в пособии «Приложение А. Репликация, распространение и синхронизация данных».

Первый вариант явно проще, чем второй, но в течение некоторого времени различные кэши могут быть несогласованны друг с другом и надежным источником данных, в зависимости от периода хранения, который применяется к кэшированным данным. Кроме того, веб-приложения и службы могут использовать локальную базу данных SQL Azure вместо получения доступа к локальной версии SQL Server. Эти базы данных SQL Azure могут быть реплицированы и синхронизированы в каждом центре обработки данных, как описано в разделе «Приложение А. Репликация, распространение и синхронизация данных». Применение этой стратегии способствует снижению сетевых задержек, связанных с извлечением данных во время заполнения кэша, за счет еще большей сложности, если веб-приложения изменяют эти данные. Они обновляют локальную базу данных SQL Azure, и эти обновления должны быть синхронизированы с базами данных SQL Azure в других центрах обработки данных.

В зависимости от того, насколько часто происходит эта синхронизация, кэшированные данные в других центрах обработки данных могут быть устаревшими на протяжении значительного периода времени. Для кэшированных данных не только должен истекать срок хранения, они также должны ждать осуществления синхронизации базы данных. В этом случае настройка интервала между событиями синхронизации базы данных, а также установка срока хранения кэшированных данных имеют важное значение, если веб-приложение должно свести до минимума промежуток времени, когда оно готово обрабатывать устаревшую информацию. На рисунке 7 показан пример такого решения с реплицированными экземплярами SQL Azure, выступающими в качестве надежного хранилища данных.

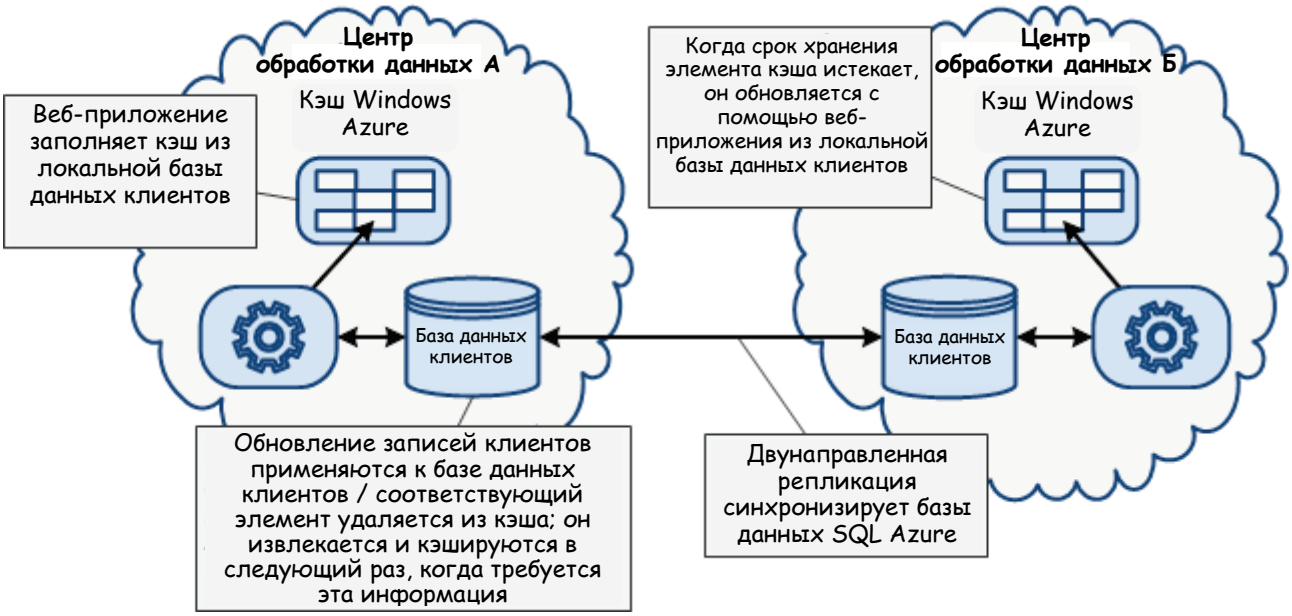



**Рисунок 7**

**Распространение обновлений между кэшами Windows Azure и реплицированными хранилищами данных**

Реализация пользовательского решения, основанная на топиках и подписках шины интеграции, является более сложной, но в результате обновления синхронизируются быстрее в центрах обработки данных. На рисунке 8 показан один из возможных вариантов реализации данного подхода. В этом примере веб-приложение получает и кэширует данные в кэше Windows Azure, размещенном в том же центре обработки данных. Выполнение обновления данных предполагает следующую последовательность задач:

a. Веб-приложение обновляет надежное хранилище данных (локальная база данных).

b. Если обновление базы данных было успешным, то веб-приложение дублирует это изменение в данные, которые хранятся в кэше в том же центре обработки данных.

c. Веб-приложение посылает сообщение об обновлении в топик шины интеграции.

d. Приложения-получатели, работающие в каждом центре обработки данных, подписываются на этот топик и получают сообщения об обновлениях.

e. Если в настоящее время данные кэшируются локально, то приложение-получатель записывает обновление в кэш этого центра обработки данных.

> **Примечание**
>
> Если на данный момент данные не кэшированы в этом центре обработки данных, то можно просто удалить сообщение об обновлении.

Получатель в центре обработки данных, размещающий веб-приложение, которое инициировало обновление, также получит сообщение об обновлении. Вы можете включать дополнительные метаданные в сообщение об обновлении с подробностями экземпляра веб-приложения, которое посылает сообщение. Впоследствии, чтобы предотвратить обновление кэша без необходимости (например, когда экземпляр веб-приложения, который послал сообщение, такой же, как и текущий экземпляр), получатель может включить соответствующую логику.

Обратите внимание, что в этом примере надежный источник данных расположен локально, но для использования реплицированных экземпляров SQL Azure в каждом центре обработки данных эта модель может быть расширена. В этом случае каждое приложение-получатель обновляет локальный экземпляр SQL Azure, а также вносит изменения в данные в кэше.

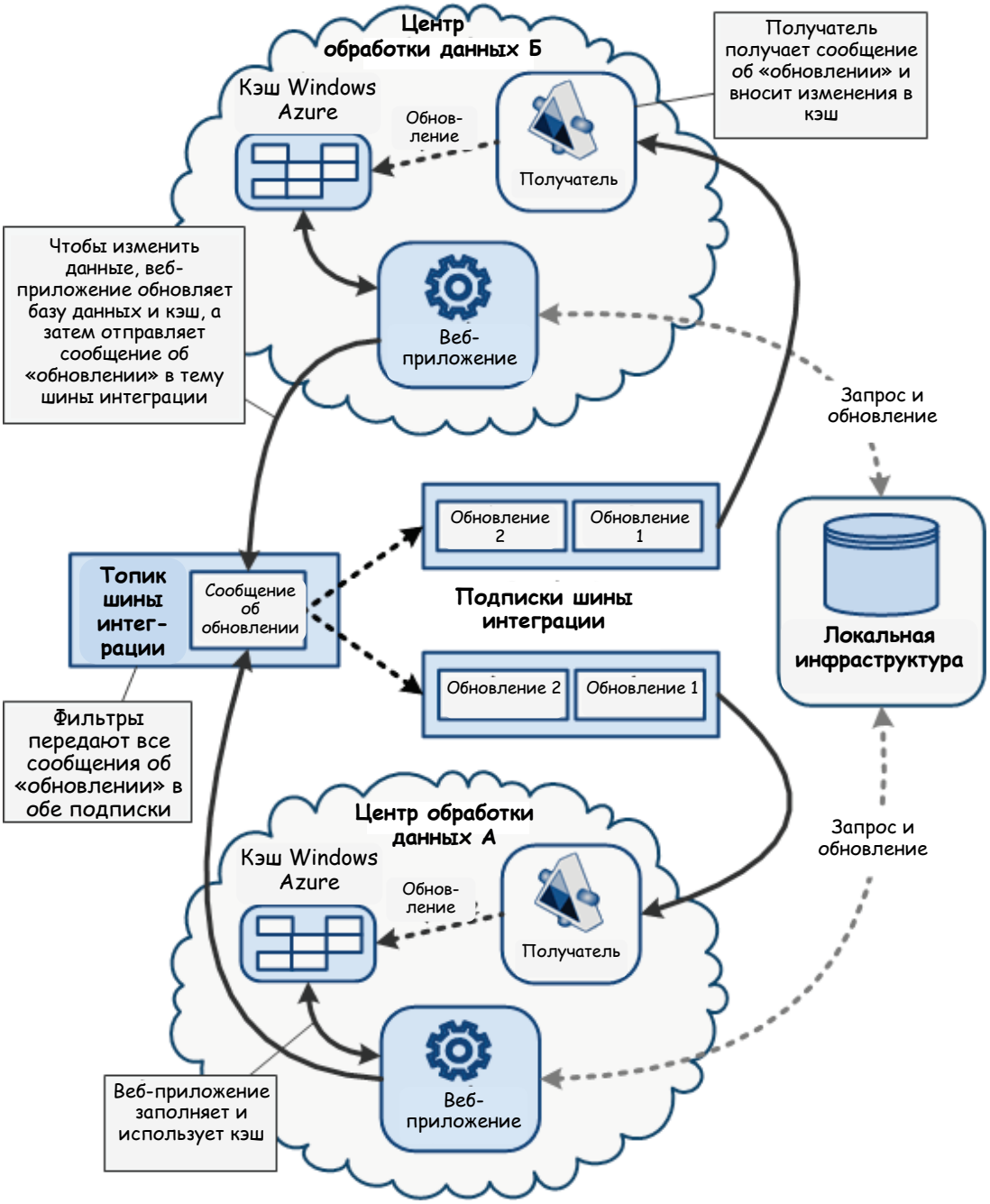



**Рисунок 8**

**Распространение обновлений между кэшами Windows Azure и надежным хранилищем данных**

**Примечание**

Возможен также вариант, что постоянного хранилища данных не существует, и в этом случае кэши выступают в качестве надежного хранилища. Примером этого сценария могут быть сетевые игры, в которых текущий счет игры постоянно обновляется, но он должен быть доступен для всех экземпляров игрового приложения. В этом случае кэш в каждом центре обработки данных содержит копию всех данных, но все равно может применяться такое же общее решение без локальной базы данных, которое изображено на рисунке 8.

- **Веб-приложению требуется быстрый доступ к данным, которые оно использует. На эти данные другие экземпляры веб-приложения не ссылаются.**

В этом случае данные являются фактически закрытыми для экземпляра веб-приложения и могут кэшироваться в памяти в самом приложении. Существует множество способов реализации этого решения, но наиболее удобный и расширяемый подход состоит в использовании API службы Windows Azure Caching, а также в настройке приложения в виде клиента кэша Windows Azure и включении свойств локального кэша. Эта конфигурация описывалась выше в этом приложении, в разделе «Создание локального кэша». Этот подход также позволяет быстро перейти к использованию общего кэша без изменения кода, необходимо просто перенастроить параметры данных для клиента кэша.

Так как данные не являются общими, то обновления просты; приложение может просто изменить данные в надежном источнике данных и, в случае успешного изменения, применять те же изменения к кэшированным данным в памяти (также обновятся данные в общем кэше, из которого изначально заполняется локальный кэш, как описано в разделе «Создание локального кэша»).

В варианте этого сценария два или больше экземпляров веб-приложения кэшируют данные локально, но они получают доступ к перекрывающимся данным из надежного хранилища данных. В этом случае, если один экземпляр изменяет данные и записывает изменения в надежное хранилище данных, то кэшированные данные другого экземпляра становятся устаревшими на данный момент. По сути это та же проблема нескольких общих кэшей, которая рассматривалась ранее. Если немедленная синхронизация между экземплярами веб-приложения важна, то кэширование данных в оперативной памяти это не самый подходящий подход, и лучше использовать общий кэш. Кроме того, срок хранения данных в локальном кэше истекает аналогично сроку в общем кэше, за исключением того, что период срока хранения по умолчанию значительно короче — пять минут. Если приложения могут обрабатывать устаревшие данные в течение короткого времени, то целесообразно использовать локальный кэш с подходящими настройками времени жизни для кэшированных объектов.

**Мнение Маркуса**

В отличие от общего кэша, для локального кэша вы можете изменить время хранения по умолчанию. Вы также имеете возможность изменить этот период во время кэширования каждого объекта. Но следует осторожно относиться к хранению объектов в локальном кэше в течение длительного периода, так как они могут очень быстро устаревать.

Кэширование данных в веб-приложении в оперативной памяти ускоряет доступ к этой информации, но, как описано выше, это может снизить согласованность решения. Вы также не должны забывать о повышенных требованиях к памяти ваших приложений и потенциальных расходах, связанных с размещением приложений с увеличенным объемом памяти, особенно если они пытаются кэшировать большие объемы данных. Вы должны тщательно сбалансировать эти преимущества и недостатки с учетом требований приложения.

На рисунке 9 показан пример такого сценария с несколькими экземплярами веб-приложения, которые используют локальный кэш для оптимизации доступа к данным, хранящимся в локальной базе данных. Не имеет значения, находятся ли экземпляры веб-приложений в одном или в разных центрах обработки данных, кэширование данных каждого экземпляра в оперативной памяти делает их независимыми друг от друга по отношению к целям запросов. Могут возникнуть некоторые обновления, и в этом примере данные, на которые ссылается

каждый экземпляр, перекрываются. Поэтому кэшированные объекты настраиваются с соответствующим периодом хранения, чтобы их можно было обновлять надлежащим образом, а также чтобы они не слишком устаревали.

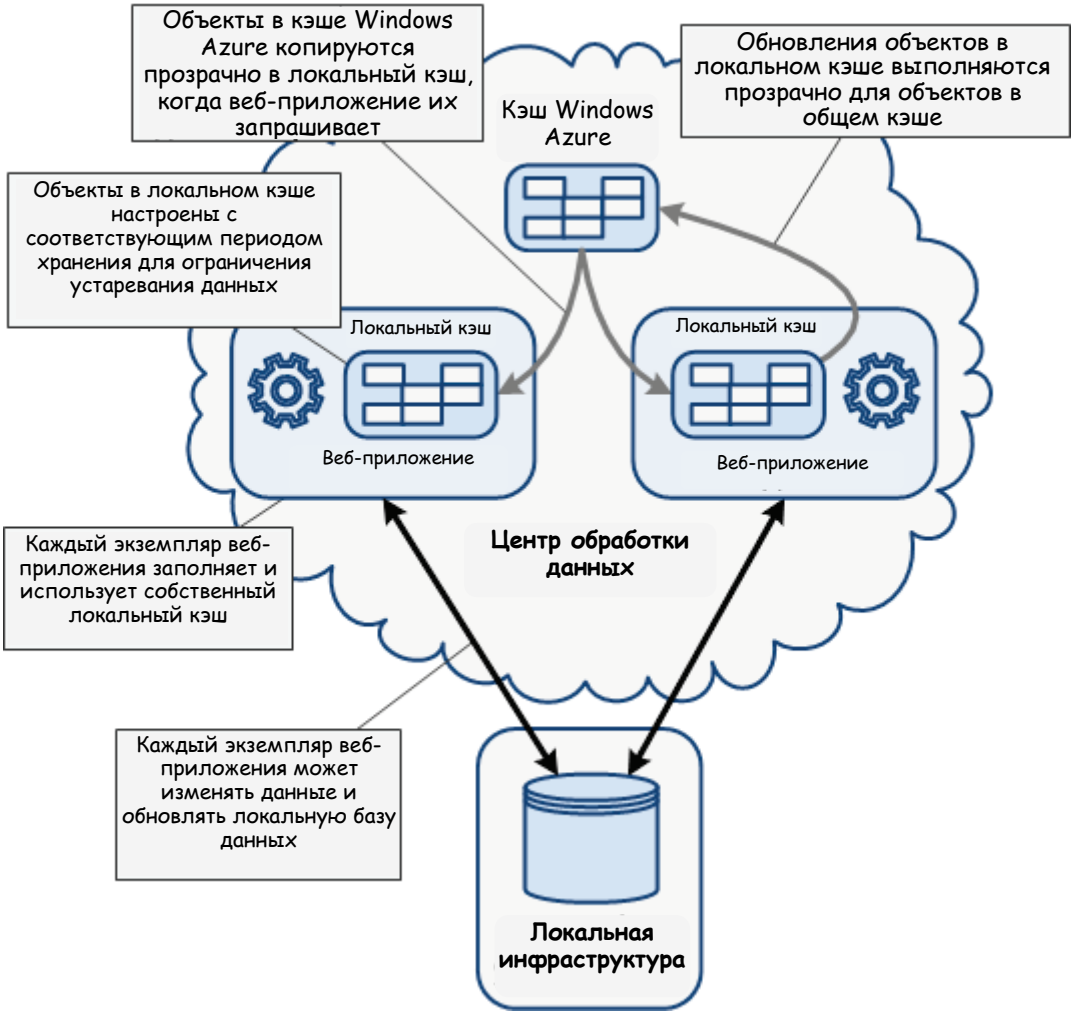



**Рисунок 9**

### Реализация локального кэширования

- **Вы создали веб-приложение, которое размещается с помощью веб-роли Windows Azure. Веб-приложению необходимо кэшировать информацию о состоянии сеанса, но эта информация не должна прикрепляться к конкретному экземпляру веб-роли. Если происходит сбой веб-приложения и веб-роль перезапускается, то сведения о состоянии сеанса не должны быть потеряны.**

Одной из основных причин использования Windows Azure для размещения веб-приложений является масштабируемость, предоставляемая этой платформой. По мере увеличения нагрузки на веб-приложение, для автоматического запуска новых экземпляров и более эффективного распределения работы, вы можете использовать технологию, такую как функциональный блок для автоматического масштабирования из библиотеки Enterprise Library (более подробно блок описан в разделе «Управление эластичностью в облаке с помощью функционального блока для автоматического масштабирования из библиотеки Enterprise Library» ранее в этом приложении). Кроме того, надежность функций Windows Azure гарантирует, что в случае сбоя приложение можно перезапустить.

Однако эти функции масштабируемости и надежности предполагают, что клиент, использующий веб-приложение, может подключаться к любому экземпляру веб-приложения. Если веб-приложение использует сеансы и сохраняет сведения о состоянии сеанса, то вы не должны привязывать эту информацию о состоянии к конкретному экземпляру приложения. Например, если вы используете ASP.NET для создания веб-приложения, то по умолчанию состояние сеанса хранится в памяти веб-приложения. В этой модели клиент, подключающийся к разным экземплярам веб-приложения в разное время, может видеть различную информацию о состояния сеанса при каждом подключении. Это явление нежелательно в масштабируемом веб-приложении.

Поставщик состояния сеанса **DistributedCacheSessionStateStoreProvider** позволяет настроить веб-приложение для хранения состояния сеанса вне процесса, используя службу Windows Azure Caching в качестве механизма хранения. Доступ к информации о состоянии того же сеанса могут получить различные экземпляры веб-приложения. Этот провайдер является прозрачным для веб-приложения, которое продолжает использовать тот же синтаксис ASP.NET для получения доступа к информации о состоянии сеанса. Для получения дополнительной информации см. раздел «*Кэширование состояния сеанса веб-приложения»* ранее в этом приложении.

Обратите внимание, что если поставщик состояния сеанса **DistributedCacheSessionStateStoreProvider** включает экземпляры веб-приложений, работающих в том же центре обработки данных, в данные сеанса общего доступа, каждый центр обработки данных должен быть настроен со своим собственным общим кэшем. Это может повлиять на ваше решение в том случае, если для маршрутизации запросов к веб-приложениям вы используете какую-либо технологию, например диспетчер трафика Windows Azure. Например, политика Round Robin диспетчера трафика Windows Azure и некоторые крайние случаи политики Performance могут перенаправить клиента к другому центру обработки данных, который размещает различные состояния сеанса для некоторых запросов, как показано на рисунке 10.

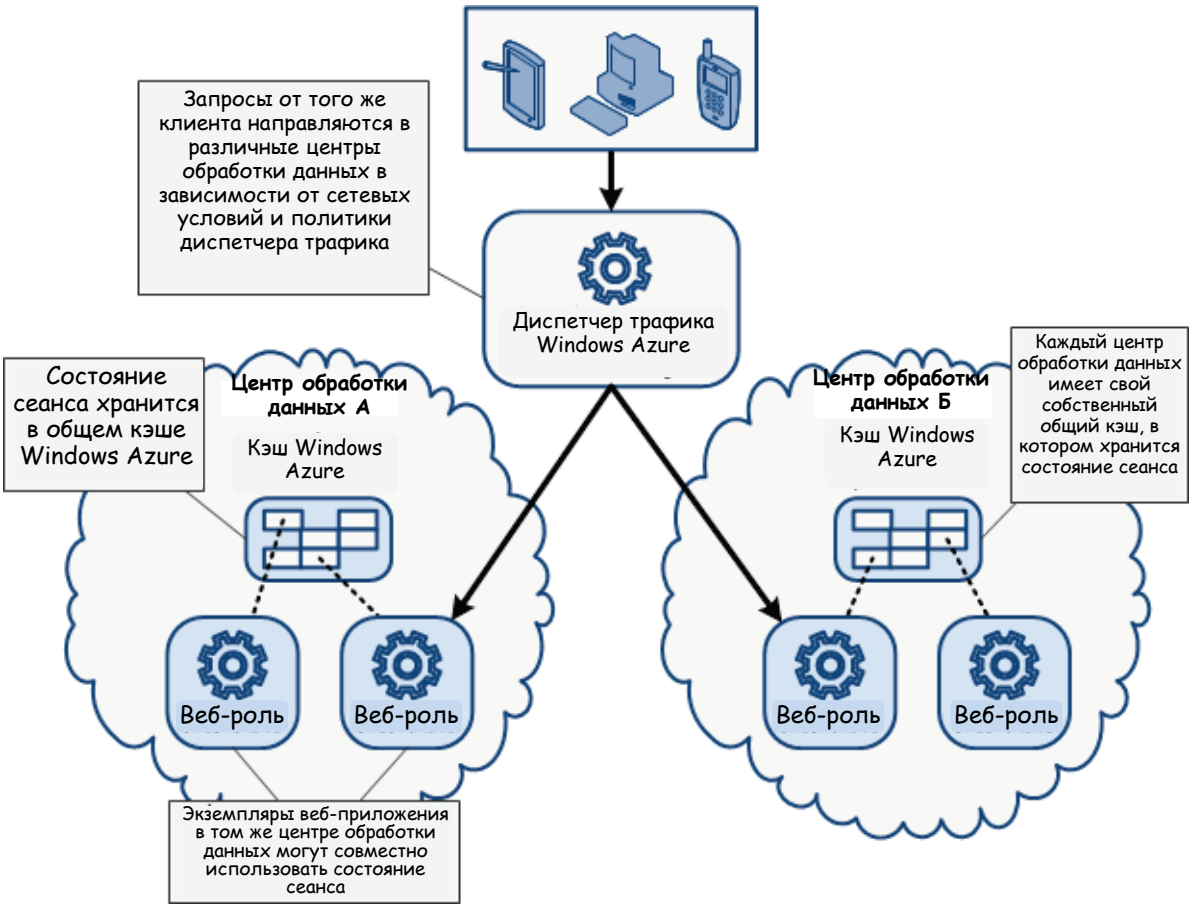



**Рисунок 10**

### Клиент запрашивает получение различных состояний сеанса из разных центров обработки данных

- **Вы создали веб-приложение, которое выполняет сложную обработку и визуализацию результатов на основании ряда параметров запроса. Вам необходимо улучшить время отклика различных страниц, обслуживаемых этим приложением, и избежать повторного выполнения одной и той же обработки, когда разные клиенты запрашивают определенные страницы.**

Это классический сценарий для реализации кэширования выводимых данных. Выводимые данные, генерируемые веб-страницей ASP.NET, можно кэшировать на сервере, где размещается веб-приложение, а последующие запросы на доступ к той же странице с теми же параметрами запроса можно удовлетворить посредством кэшированной версии страницы, вместо создания нового ответа. Дополнительная информация о том, как работает кэширование выводимых данных ASP.NET и как его использовать, представлена в статье «Caching ASP.NET Pages» на сайте http://msdn.microsoft.com/en-us/library/06bh14hk(v=VS.100).aspx.

Кроме того, по умолчанию поставщик кэша выводимых данных, который поставляется ASP.NET, работает на каждом сервере. В среде Windows Azure веб-сервер приравнивается к экземпляру веб-роли, так что использование поставщика кэша выводимых данных по умолчанию стает причиной того, что каждый экземпляр веб-роли создает собственный кэшированный вывод данных. Если объем кэшированных выводимых данных большой и каждая кэшированная страница является результатом сложной, глубокой обработки, то каждый экземпляр веб-роли может прекратить дублирование этой обработки и хранение копии одного и того же результата. Класс **DistributedCacheOutputCacheProvider** позволяет веб-ролям хранить кэш выводимых

данных в общем кэше Windows Azure, позволяя избежать дупликации пространства и усилий. Для получения дополнительной информации см. раздел «*Кэширование выводимых данных в формате HTML*» ранее в этом приложении.

Как и с кэшем сеанса, для кэширования выводимых данных в каждом центре обработки данных вы должны создать и использовать отдельный общий кэш.

### Ограничения для службы Windows Azure Caching

Функции, предоставляемые службой Windows Azure Caching, очень похожи на функции службы Windows Server AppFabric Caching, они используют те же программные API и методы настройки. Однако реализация Windows Azure предоставляет только подмножество функций, которые доступны в версии Windows Server. В настоящее время по сравнению со службой Windows Server AppFabric Caching, служба Windows Azure Caching имеет следующие ограничения:

- Она не поддерживает уведомления. Приложения не получают уведомления, если срок хранения объекта истекает или он удаляется из кэша.
- Невозможно изменить период хранения по умолчанию для общего кэша. Срок хранения объектов в общем кэше истекает через 48 часов, и нет возможности изменить эту настройку для кэша в целом. Тем не менее можно изменить это значение для каждого объекта в период хранения их в кэше. Однако имеется возможность изменения периода хранения по умолчанию для локального кэша (по умолчанию продолжительность составляет пять минут).
- Невозможно отключить политику удаления данных. Если не хватает места в кэше для новых объектов, старые объекты удаляются в соответствии с принципом наиболее давнего использования.
- Невозможно явно удалить элемент из кэша.
- Невозможно делить кэшированные данные на части. Кэш Windows Azure не может содержать пользовательские именованные области.
- Нет возможности добавлять теги для кэшированных данных, чтобы облегчить поиск объектов.

**Примечание**

Возможно, в будущих версиях службы Windows Azure Caching некоторые из этих ограничений будут устранены.

Необходимо также отметить, что кэш Windows Azure автоматически получает преимущества функций надежности и масштабируемости Windows Azure; вам не придется управлять этими аспектами самостоятельно. Следовательно, многие функции высокой доступности службы Windows Server AppFabric Caching недоступны, потому что они не требуются в среде Windows Azure.

Дополнительная информация о различиях между службой Windows Azure Caching и службой Windows Server AppFabric Caching представлена в статье «Differences Between Caching On-Premises and in the Cloud» на сайте http://msdn.microsoft.com/en-us/library/windowsazure/gg185678.aspx.

### Рекомендации по обеспечению безопасности службы Windows Azure Caching

Вы получаете доступ к кэшу Windows Azure через экземпляр службы Windows Azure Caching. Создается экземпляр службы Windows Azure Caching с помощью портала управления и указания нового пространства имен для службы кэширования. Служба кэширования развертывается в центре обработки данных в облаке и имеет конечные точки с URL-адресами, которые основаны на имени пространства

имен службы с суффиксом «.cache.windows.net». Ваши приложения подключаются к службе кэширования, используя эти URL-адреса. Служба кэширования предоставляет конечные точки, которые поддерживают основные подключения HTTP (через порт 22233), а также SSL (через порт 22243).

Все запросы приложения на соединение со службой Windows Azure Caching проходят аутентификацию и авторизацию с помощью ACS. Для подключения к службе кэширования приложение должно предоставить соответствующий маркер аутентификации.

**Мнение Бхарата**

Для подключения к службе Windows Azure Caching только веб-приложения и службы, работающие в облаке, должны быть обеспечены маркером аутентификации, так как только эти элементы должны подключаться к кэшу. Использование кэша Windows Azure из кода, работающего вне центра обработки данных, дает немного преимуществ, за исключением тестирования при использовании эмулятора вычислений Windows Azure, и является неподходящим сценарием для производственных целей.

# Приложение E. Мониторинг и управление гибридными приложениями

Типичное гибридное приложение включает в себя ряд компонентов, созданных с использованием различных технологий, распределенных между множеством узлов и соединенных сетями различной пропускной способности и надежности. Несмотря на всю сложность, для компании Trey Research было очень важно получить возможность контроля работы системы, и в случае сбоя быстро принимать необходимые меры по ее восстановлению. Однако мониторинг сложной системы сам по себе является очень сложной задачей, необходимы инструменты, позволяющие быстро собрать данные о производительности, чтобы проанализировать пропускную способность и определить причины ошибок, сбоев или других проблем в системе.

Возникающие проблемы могут существенно отличаться, начиная от простых сбоев, вызванных ошибками приложения в службе, работающей в облаке, проблем со средой размещения отдельных элементов, и заканчивая полным отказом системы и потерей связи между компонентами, работающими локально или в облаке.

После получения уведомления о проблеме необходимо принять соответствующие меры, исправить ошибки и обеспечить нормальное функционирование системы. Чем раньше вы сможете выявить и исправить проблемы, тем меньшее влияние они окажут на операции, от которых зависит ваш бизнес, и на клиентов, использующих вашу систему.

Важно придерживаться системного подхода, и не только в сфере управления гибридными приложениями, но также при развертывании и обновлении компонентов своей системы. Вы должны сделать все возможное, чтобы свести к минимуму влияние на производительность со стороны процессов мониторинга и управления, а также предотвратить недоступность всей системы, когда вам нужно обновить только отдельные ее элементы.

Сбор диагностической информации о работоспособности вашей системы также играет важную роль в определении возможностей вашего решения, а это в свою очередь, может влиять на соглашения об уровне обслуживания (Service Level Agreement, SLA), которые вы заключаете с пользователями. Мониторинг вашей системы позволяет проанализировать, как она потребляет ресурсы по мере увеличения или уменьшения количества запросов, а это в свою очередь помогает оценить потребность в ресурсах и определить эксплуатационные затраты, связанные с поддержкой согласованного уровня производительности.

В настоящем приложении рассматриваются проблемы, с которыми вы сталкиваетесь, пытаясь поддерживать высокую работоспособность своих приложений и выполнять обязательства перед клиентами. Здесь также описываются технологии и инструменты, предоставляемые технологической платформой Windows Azure, которые помогут вам организовать упреждающий контроль и управление своими решениями, а также определить возможности своей системы и эксплуатационные расходы.

## Сценарии использования и проблемы

Мониторинг и управление гибридными приложениями представляет собой нетривиальную задачу с учетом количества, расположения и неоднородности различных мобильных элементов, которые составляет типичное бизнес-решение. Сбор точной и своевременной информации позволяет эффективно измерять производительность вашего решения и контролировать состояние системы. Также первостепенное значение имеют продуманные процедуры восстановления элементов в случае их выхода из строя. Возможно, вам также придется собирать информацию аудита о том, как ваша система используются и кем.

В следующих разделах описаны некоторые общие сценарии использования инструментов мониторинга и обслуживания гибридного приложения, а также многие из проблем, с которыми вы столкнетесь в процессе решения соответствующих задач.

### Измерение и регулировка производительности вашей системы

**Описание: вам необходимо определить производительность своей системы, чтобы можно было определить, где нужны дополнительные ресурсы, и оценить эксплуатационные издержки, связанные с этими ресурсами.**

В коммерческой среде клиенты платят за ваши услуги и рассчитывают получить качество обслуживания и производительность, указанные в соглашении об уровне обслуживания. Даже «бесплатные» посетители вашего веб-сайта и пользователи ваших служб предполагают, что вы обеспечиваете бесперебойную работу. Потенциальных клиентов раздражает низкая производительность и ошибки, вызванные нехваткой ресурсов.

> **Мнение Бхарата**
>
> Клиентов волнует качество услуг, которые предоставляет ваша система, а не то, как функционирует сеть или облачная среда. Вы должны убедиться в том, что ваша система разработана и оптимизирована для облачных вычислений, это поможет вам давать (и выполнять) реалистичные обещания своим пользователям.

Одним из способов оценки производительности вашей системы является мониторинг ее работы в реальных условиях. Многочисленные вопросы, связанные с эффективностью контроля и управления системой, также возникают, когда вы размещаете решение локально на собственных серверах вашей организации. Однако, когда вы перемещаете службы и функционал в облако, ситуация может значительно осложниться в силу ряда причин, в том числе:

- Серверы теперь работают в удаленных помещениях и могут включаться и отключаться автоматически, поскольку вы используете эластичное и масштабируемое решение.
- На этих серверах могут быть развернуты несколько экземпляров ваших служб.
- В облаке могут размещаться многопользовательские приложения.
- Взаимодействие может быть неустойчивым и непостоянным.
- Могут иметь место асинхронные операции.

Это, несомненно, гораздо более сложная ситуация, в сравнении с локальным размещением. Вы должны не только организовать сбор статистической информации и данных о производительности каждого экземпляра каждой службы, работающего на каждом сервере, но и предоставить эту информацию в понятной для администратора форме, чтобы он смог быстро оценить состояние вашей системы и выявить причины любых сбоев или проблем с производительностью. В свою очередь, это требует

создания инфраструктуры, которая сможет дистанционно собирать необходимую информацию от ваших служб и серверов в облаке, а затем сохранять и анализировать эти данные с целью выявления предпосылок для потенциальных проблем, таких как чрезмерная длина очереди запросов, наличие узких мест с точки зрения обработки, высокое время отклика и так далее.

На основе полученной информации вы сможете предпринять шаги, направленные на исправление обнаруженных тенденций, возможно, путем запуска дополнительных экземпляров службы, развертывания служб в большем количестве центров обработки данных в облаке или изменения конфигурации системы. В некоторых случаях вы также можете определить, какие элементы системы должны быть реорганизованы, чтобы обеспечить требуемую пропускную способность. Например, службу, которая обрабатывает запросы синхронно, возможно, придется модифицировать, чтобы она выполняла требуемые операции асинхронно, или может потребоваться иной механизм обмена данными для более надежной отправки запросов на обслуживание.

## Мониторинг служб с целью раннего обнаружения сбоев и проблем с производительностью

**Описание: вам необходимо поддерживать гарантированный уровень обслуживания.**

В идеале, программное обеспечение никогда не подводит, и все всегда работает как нужно. В действительности это невозможно, но вы должны стремиться обеспечить пользователям такое впечатление, что ваша система всегда работает отлично, им не нужно знать о проблемах, которые могут периодически возникать.

Однако независимо от того, насколько хорошо вы протестировали систему, всегда будут факторы, не поддающиеся контролю, которые могут повлиять на работоспособность системы. Ярким примером является сеть. Кроме того, если вы не уделили достаточно внимания строгой комплексной математической проверке компонентов вашего решения и не потратили на это средства, то не можете гарантировать, что они будут работать без ошибок. Основной задачей в сфере поддержания высокого качества обслуживания является упреждающее обнаружение проблем до того, как их заметят ваши клиенты. Вы должны провести диагностику и выявить причины, а затем быстро устранить неисправности или переконфигурировать систему с целью прозрачного перенаправления запросов.

> **Мнение По**
>
> Не забывайте о том, что тестирование может только подтвердить наличие проблем, но не их отсутствие.

Тщательно проработанные методы и инфраструктура, которые вы применяете для мониторинга и измерения производительности своей системы, также могут использоваться для обнаружения сбоев. Очень важно, чтобы инфраструктура обращала внимание обслуживающего персонала на такие ошибки на ранних стадиях, чтобы можно было быстро и эффективно принять корректирующие меры. Собранная информация должна быть достаточной, чтобы оперативный персонал не ее основе мог проследить тенденции, и, в случае необходимости, динамически переконфигурировать систему, добавить или удалить ресурсы, подстраиваясь под изменения нагрузки.

## Быстрое восстановление после сбоев

**Описание: вы должны придерживаться системного подхода к устранению сбоев и обеспечивать быстрое восстановление.**

Как только вы определили причину неисправности, нужно приступать к восстановлению вышедших из строя компонентов или изменению конфигурации системы. Реальная среда включает множество компьютеров и поддерживает тысячи пользователей, поэтому вы должны решать эту задачу, применяя

систематизированные, задокументированные и повторяемые методы с минимальным ущербом для доступности служб. В идеале, необходимо подготовить сценарий с описанием всех этапов, чтобы можно было их автоматизировать, при этом необходимо обеспечить достаточную надежность, чтобы регистрировать и обрабатывать любые ошибки, которые могут возникнуть в процессе выполнения запланированных операций.

## Ведение журнала и операции аудита

**Описание: вам необходимо записывать все операции, выполняемые каждым экземпляром службы в каждом центре обработки данных.**

Возможно, вам придется вести журналы всех операций, выполняемых пользователями, которые получают доступ к каждому экземпляру вашей службы, а также регистрировать изменения, ошибки и другие события во время выполнения. Эти журналы должны быть полными и защищенными, вести их следует постоянно. Необходимость ведения журнала может быть обусловлена нормативными требованиями вашей системы, но даже если это не так, вам в любом случае следует отслеживать ресурсы, к которым получает доступ каждый пользователь в целях выставления счетов.

Журнал аудита должен содержать записи обо всех операциях, выполняемых оперативным персоналом, таких как отключение и перезапуск служб, изменение конфигурации, развертывание и так далее. Если вы выставляете клиентам счета за доступ к своей системе, журнал аудита должен также содержать информацию об операциях, которые они запрашивали, и ресурсах, потребляемых в процессе выполнения этих операций.

Журнал ошибок должен содержать записи с указанием даты, времени и описания всех ошибок и других значимых событий, возникающих в вашей системе, например исключений, вызванных отказом компонентов и инфраструктуры. В журнале также регистрируются нештатные ситуации, такие как неудачные попытки входа в систему.

Журнал производительности должен предоставлять информацию, достаточную для контроля и оценки работоспособности элементов вашей системы. Также должны быть доступны аналитические инструменты, позволяющие выявлять тенденции, которые могут привести к сбоям. Например, когда СУБД SQL Azure приближается к максимальной запланированной мощности, необходимо уведомить оперативный персонал о том, что они должны принять меры по предотвращению сбоев в работе это системы.

## Развертывание и обновление компонентов

**Описание: необходимо развертывать и обновлять компоненты в рамках контролируемого, повторяемого и надежного подхода, поддерживая при этом доступность системы.**

Как и в случае с восстановлением системы после сбоя, в процессе развертывания и обновления всех компонентов необходим строгий контроль и документирование, что позволяет быстро отменить любые изменения в случае неудачного развертывания, систему нив коем случае нельзя оставлять в неопределенном состоянии. Вы должны применять процедуры, которые помогают провести обновление таким образом, чтобы свести к минимуму любые возможные простои в работе пользователей, система должна оставаться доступной в процессе обновления. Кроме того, все изменения необходимо тщательно проверить в облачной среде, прежде чем они станут доступными для реальных клиентов.

# Взаимосвязанные проблемы

В этом разделе рассматриваются основные взаимосвязанные проблемы, которые вам, возможно, придется решать в ходе реализация стратегии, связанной с мониторингом и управлением гибридным решением.

## Производительность

Мониторинг системы, сбор диагностической информации и данных аудита будут влиять на производительность системы. Решение, которое вы используете, должно свести к минимуму такое влияние, чтобы предотвратить негативное воздействие на работу ваших клиентов.

Что касается диагностической информации, в рамках стабильной конфигурации, возможно, нет необходимости собирать обширную статистику. Однако в критические периоды дополнительная информация может помочь вам быстрее обнаружить и устранить все проблемы. Поэтому любое решение должно быть достаточно гибким, позволяя проводить тонкую настройку и переконфигурирование процесса мониторинга с учетом сложившейся ситуации.

Политика в сфере сбора данных аудита, скорее всего, не будет такой же гибкой, поэтому вы должны определить эффективный механизм для сбора этих данных, а также компактный механизм для их транспортировки и хранения.

## Безопасность

С точки зрения безопасности, в процессе сбора диагностической информации и данных аудита необходимо учитывать следующее:

- Диагностические данные конфиденциальны, поскольку могут содержать информацию о конфигурации системы и выполняемых операциях. Злоумышленник может использовать эту информацию, чтобы проникнуть в вашу систему. Поэтому вы должны защищать эти данные в процессе их передачи по сети. Вы также должны обеспечить надежное хранение этих данных.
- Диагностические данные могут также содержать информацию об операциях, выполняемых вашими клиентами. Вы должны предотвратить сбор любой персональной информации об этих клиентах, сохранение этой информации вместе с диагностическими данными, а также ее доступность для операторов, осуществляющих мониторинг вашей системы.
- Данные аудита входят в документы постоянного хранения о задачах, выполняемых вашей системой. В зависимости от особенностей вашей системы и юрисдикции, в которой работает ваша организация, в соответствии с нормативными требованиями вам, возможно, придется хранить эти данные в неизменном виде. Нужно обеспечить надежное хранение и защиту этих данных.

---

Кроме того, задачи мониторинга, управления, развертывания и технического обслуживания связаны с компонентами, из которых состоит ваша система. Они очень важны, и за их выполнение должны отвечать конкретные сотрудники. Вы должны принять необходимые меры в целях предотвращения несанкционированного доступа сотрудников к данным мониторинга, функциям развертывания компонентов или изменения конфигурации системы.

## Платформа Windows Azure и связанные с ней технологии

Windows Azure предоставляет ряд полезных инструментов и интерфейсов API, которые можно использовать для контроля и управления гибридными приложениями. В частности, вы можете использовать:

- Службу Windows Azure Diagnostics для сбора диагностической информации в целях мониторинга системы. Служба Windows Azure Diagnostics может работать совместно с функциональным блоком Logging Application Block из библиотеки Enterprise Library для ведения журналов. Решение Microsoft Systems Center Operations Manager также предоставляет пакет управления для Windows Azure на основе службы Windows Azure Diagnostics.
- Портал управления Windows Azure Management Portal, который позволяет администраторам предоставлять необходимые приложениям ресурсы и веб-сайты. Он также предоставляет средства реализации различных ролей безопасности, обеспечивающих защиту этих ресурсов и веб-сайтов. Для получения дополнительной информации зарегистрируйтесь на портале управления Management Portal: http://windows.azure.com.
- Интерфейс Windows Azure Service Management API, который позволяет создавать собственные средства администрирования, а также выполнять сценарии управления через интерфейс командной строки Windows PowerShell.

**Мнение Бхарата**

Вы можете настроить удаленный доступ к рабочему столу для ролей, запускающих ваши приложения и сервисы в облаке. Эта функция позволяет получить доступ к журналам Windows и другой диагностической информации непосредственно на этих машинах, с помощью тех же процедур и инструментов, которые используются для получения данных на локальных компьютерах.

В следующих разделах предоставляются дополнительные сведения об интерфейсе Windows Azure Service Management API и службе Windows Azure Diagnostics. Мы обобщаем передовые практические методы и рассказываем о вариантах их использования для поддержки гибридного приложения.

## Мониторинг служб, ведение журнала и измерение производительности в гибридном приложении с помощью службы Windows Azure Diagnostics

Локальное приложение обычно состоит из нескольких четко определенных элементов, работающих на определенном количестве компьютеров и получающих доступ к хорошо известным ресурсам. Для мониторинга таких приложений требуются средства прозрачного перехвата и записи различных запросов, которыми обмениваются компоненты, также необходимо регистрировать любые значимые события. В таких условиях у вас есть полный контроль над развернутым программным обеспечением и конфигурацией каждого компьютера. Вы можете установить инструменты сбора данных локально на каждом компьютере, а затем формировать сводные отчеты, обеспечивающие общее представление о том, насколько хорошо система работает.

В гибридном приложении ситуация гораздо сложнее: ваша среда динамична, число вычислительных узлов с установленными веб-ролями и рабочими ролями, на которых работают экземпляры ваших служб и компонентов, может изменяться, и рабочая нагрузка распределяется между этими узлами. Вы не можете в полной мере контролировать конфигурации этих узлов, поскольку управление и поддержку осуществляют центры обработки данных, в которых эти узлы размещаются. Вы не можете просто

установить собственные инструменты мониторинга для оценки производительности и работоспособности элементов, расположенных на каждом узле. В такой ситуации вам поможет служба Windows Azure Diagnostics.

Служба Windows Azure Diagnostics устанавливает свою инфраструктуру на каждом узле, что позволяет собирать данные о производительности и другую диагностическую информацию о компонентах, работающих на этих узлах. Эта служба легко настраивается, вы можете указать информацию, которая вас интересует, будь то данные из журналов событий, журналов трассировки или журналов IIS, показания счетчиков производительности, аварийные дампы или данные из других произвольных журналов. Подробная информация о развертывании службы Windows Azure Diagnostics и настройке приложений для контроля информации, собираемой этой службой, представлена в статье «Collecting Logging Data by Using Windows Azure Diagnostics» на сайте http://msdn.microsoft.com/en-us/library/gg433048.aspx.

Служба Windows Azure Diagnostics разработана специально для облачных сред. Службу легко масштабировать, чтобы свести к минимуму влияние на производительность ролей, которые настроены на ее использование. При этом сами диагностические данные хранятся локально на каждом контролируемом вычислительном узле. Данные теряются при перезагрузке узла. Кроме того, служба диагностики Windows Azure устанавливает квоту в 4 ГБ на объем журнала, и если эта квота превышена, самые старые записи удаляются. Можно изменить эту квоту, но она не должна превышать объем хранилища, установленный для веб-роли или рабочей роли. Скорее всего, придется сохранять эти данные, и вы можете организовать передачу диагностической информации в хранилище Windows Azure (периодически или по запросу). В статьях «How to Schedule a Transfer» (http://msdn.microsoft.com/en-us/library/windowsazure/gg433085.aspx) и «How to Perform an On-Demand Transfer2» (http://msdn.microsoft.com/en-us/library/windowsazure/gg433075.aspx) представлены рекомендации по поводу решения соответствующих задач.

**Мнение Маркуса**

На передачу диагностических данных в хранилище Windows Azure по требованию может потребоваться некоторое время, в зависимости от объема копируемой информации и местоположения хранилища Windows Azure. Чтобы обеспечить наилучшую производительность, используйте учетную запись хранилища данных в том же центре обработки данных, где работает вычислительный узел с развернутой веб-ролью или рабочей ролью, который является объектом мониторинга. Кроме того, вы должны выполнить передачу асинхронно, что сводит к минимуму влияние на время отклика программного кода.

Хранилище Windows Azure не зависит от конкретного вычислительного узла, и информация, которую оно содержит, не будет потеряна, если какой-либо вычислительный узел будет перезапущен. Вы должны создать учетную запись в системе хранения для этих данных. Необходимо также настроить монитор диагностики Windows Azure Diagnostics Monitor с адресом этой учетной записи и соответствующим ключом доступа. Более подробная информация представлена в статье «How to Specify a Storage Account for Transfers» на сайте http://msdn.microsoft.com/en-us/library/windowsazure/gg433081.aspx. Журналы на основе событий передаются в табличное хранилище Windows Azure, а журналы на основе файлов копируются в хранилище BLOB-объектов. Соответствующие таблицы и BLOB-объекты создаются инструментом Windows Azure Diagnostics Monitor. Например, информация из журналов событий Windows переносится в таблицу с именем **WADWindowsEventLogsTable**, а данные счетчиков производительности копируются в таблицу с именем **WADPerformanceCountersTable**. Аварийные дампы передаются в хранилище BLOB-объектов по пути **wad-crash-dumps**, а журналы IIS 7.0 копируются в другое хранилище BLOB-объектов по пути **wad-iis-logfiles**.

### Рекомендации по применению службы Windows Azure Diagnostics

С технической точки зрения, служба Windows Azure Diagnostics реализована в виде компонента в пакете Windows Azure SDK, который поддерживает эти стандартные диагностические интерфейсы API. Этот компонент называется *«Монитор диагностики Windows Azure»*, он работает в облаке параллельно с каждой веб-ролью или рабочей ролью, от которых вы хотите получить информацию.

Вы можете настроить монитор диагностики, определив данные, которые необходимо собирать, с помощью файла конфигурации diagnostics.wadcfg службы Windows Azure Diagnostics. Дополнительная информация представлена в статье «How to Use the Windows Azure Diagnostics Configuration File» на сайте http://msdn.microsoft.com/en-us/library/windowsazure/hh411551.aspx. Кроме того, приложение может записывать пользовательские данные трассировки в журнал трассировки. Служба Windows Azure Diagnostics предоставляет класс **DiagnosticMonitorTraceListener**, позволяющий решить эту задачу, и вы можете настроить этот тип трассировки путем добавления соответствующего раздела **<system.diagnostics>** в файл конфигурации приложения вашей веб-роли или рабочей роли. Более подробная информация представлена в статье «How to Configure the TraceListener in a Windows Azure Application» на сайте http://msdn.microsoft.com/en-us/library/hh411522.aspx.

Если вы работаете над созданием распределенного гибридного решения, с помощью службы Windows Azure Diagnostics вы сможете получать данные от нескольких экземпляров ролей, размещаемых на распределенных вычислительных узлах. Агрегируя эти данные, вы будете получать общее представление о своей системе. Вы можете использовать решение System Center Operations Manager или разнообразные сторонние приложения, которые могут работать с исходными данными, предоставляемыми службой Windows Azure Diagnostics. Это позволяет представлять информацию различными способами в максимально доступной для понимания и анализа форме. Эти инструменты позволяют определить пропускную способность, производительность и время отклика вашей системы и ресурсов, которые она потребляет. Анализируя эту информацию, вы можете определить проблемные области, которые могут оказать негативное влияние на операции, выполняемые системой, а также оценить расходы, связанные с выполнением этих операций.

Далее представлены возможности для включения службы Windows Azure Diagnostics в ваши решения:

- **Вам необходимо централизованное представление о своей системе, чтобы гарантировать соответствие требованиям соглашений об уровне обслуживания, а также обеспечить точный учет потребления ресурсов для выставления счетов. Ваша организация в настоящее время использует решение System Center Operations Manager для мониторинга и сопровождения служб, работающих локально.**

  Если развернуто решение Systems Center Operations Manager локально, вы также можете установить пакет управления мониторингом Monitoring Management Pack для Windows Azure. Этот пакет работает совместно со службой Windows Azure Diagnostics на каждом вычислительном узле, что позволяет записывать и просматривать диагностическую информацию о приложениях и службах, размещенных в облаке, а также интегрировать ее с данными о других элементах вашего решения, которые работают локально. Рассматриваемый инструмент также помогает проанализировать, как формирующие вашу систему службы используют ресурсы, что, в свою очередь, позволяет определить, какие расходы необходимо переложить на ваших клиентов, если это необходимо.

  По умолчанию пакет управления мониторингом для приложений Windows Azure контролирует состояние развертывания ролей, состояние каждой размещаемой службы и роли, показания счетчиков производительности ASP.NET, емкость диска, использование физической памяти и ресурсов сетевого адаптера, а также производительность процессора.

При помощи Systems Center Operations Manager можно настроить оповещения, инициализация которых происходит, когда какой-либо показатель превышает указанный порог. Эти оповещения можно связать с задачами, а также автоматизировать процедуры для принятия необходимых корректирующих мер. Например, можно настроить систему на запуск дополнительных экземпляров ролей, когда превышается пороговое время отклика в процессе обработки клиентских запросов, также можно отправить уведомление оператору, который способен проанализировать проблему.

Более подробная информация о решении System Center Operations Manager и пакете Monitoring Pack для Windows Azure представлена на сайте http://pinpoint.microsoft.com/en-us/applications/system-center-monitoring-pack-for-windows-azure-applications-12884907699.

- **Вам необходимо централизованное представление о своей системе, чтобы контролировать свое приложение и сохранять записи аудита для отдельных операций. Ваша организация не использует решение Systems Center Operations Manager.**

Вы можете периодически передавать диагностические данные в хранилище Windows Azure и анализировать их с помощью таких инструментов, как Windows Azure Storage Explorer от Neudesic (см. http://azurestorageexplorer.codeplex.com/). Кроме того, если у вас есть система разработки Visual Studio и пакет Windows Azure SDK, вы можете подключаться к хранилищу Windows Azure и просматривать содержимое таблиц и BLOB-объектов с помощью Server Explorer. Более подробная информация представлена в статье «How to View Diagnostic Data Stored in Windows Azure Storage» на сайте http://msdn.microsoft.com/en-us/library/windowsazure/hh411547.aspx.

Однако эти инструменты предоставляют лишь обобщенный доступ к хранилищу Windows Azure. Если вам нужно проанализировать данные более детально, возможно, придется создать собственную панель мониторинга, которая подключается к таблицам и BLOB-объектам, относящимся к конкретной учетной записи хранилища данных, агрегирует собранную информацию для всех узлов, а также генерирует отчеты, которые показывают, как пропускная способность меняется с течением времени. Это позволяет выявлять тенденции, которые могут обуславливать необходимость выделения дополнительных ресурсов. Вы также можете загрузить диагностические данные из хранилища Windows Azure на свои локальные серверы, если вам нужно вести постоянную запись этой информации, например, в журнале аудита для отдельных операций, а также если вы хотите проанализировать данные в автономном режиме.

На рисунке 1 представлена общая структура решения, которое собирает диагностическую информацию на нескольких узлах и анализирует ее локально. Диагностические данные проходят процедуру изменения формата и копируются в таблицы базы данных SQL Server, затем службы отчетности SQL Server формируют серию отчетов, графически представляя сводную информацию о производительности системы.

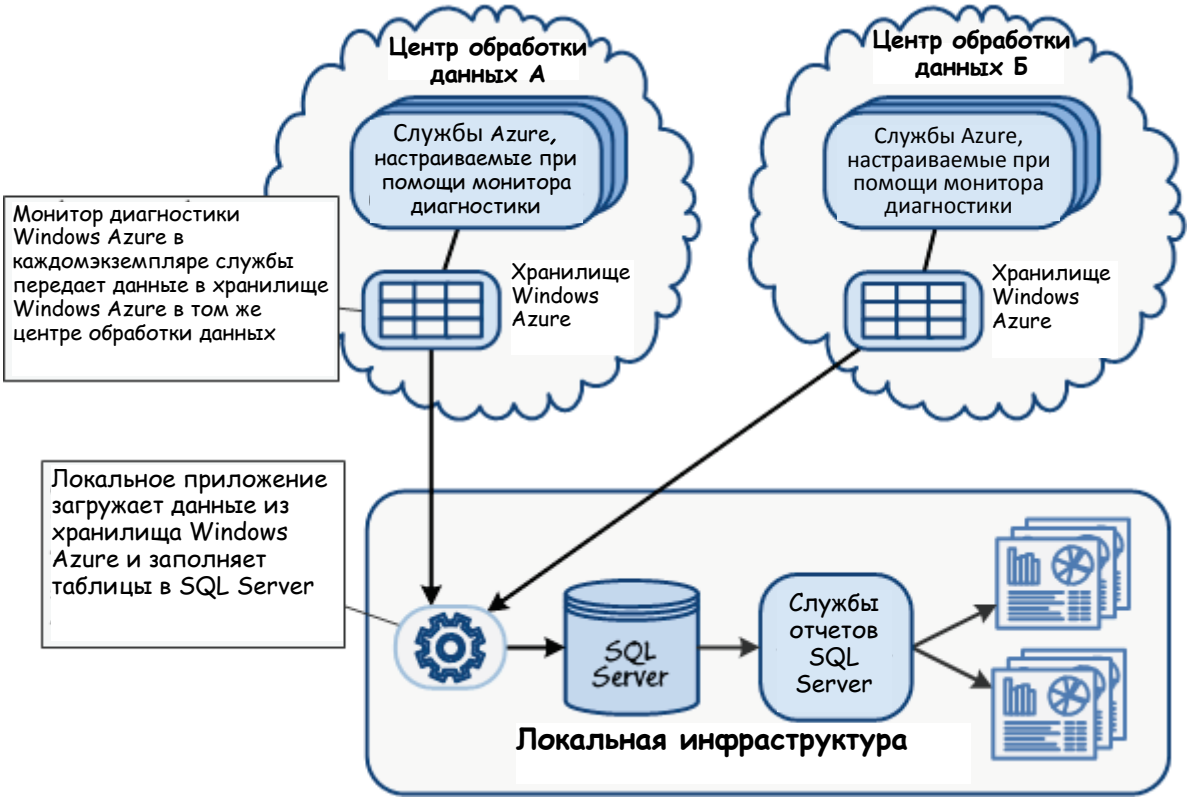



**Рисунок 1**

### Сбор диагностических данных с нескольких узлов

Альтернативный подход заключается в применении решений сторонних поставщиков. Некоторые решения, которые были актуальны на момент написания статьи:

- Azure Diagnostics Manager от Cerebrata (http://www.cerebrata.com/Products/AzureDiagnosticsManager/Default.aspx).
- AzureWatch от Paraleap Technologies (http://www.paraleap.com/).
- ManageAzis от Cumulux (http://www.cumulux.com/products-and-services/cloud-operations/).

- **Вы должны инструментировать свои веб-приложения и службы, чтобы получить возможности для выявления любых потенциальных узких мест и сбора общей информации о состоянии и работоспособности вашего решения.**

Приложения, работающие в облаке, могут использовать интерфейсы API службы Windows Azure Diagnostics, чтобы подключить логику, которая генерирует диагностическую информацию, позволяя инструментировать ваш код и контролировать работоспособность ваших приложений с помощью счетчиков производительности, применимых к веб-ролям и рабочим ролям. Вы можете также создать собственные средства диагностики и счетчики производительности. Более подробная информация представлена в статье «Tracing the Flow of Your Windows Azure Application»http://msdn.microsoft.com/en-us/library/windowsazure/hh411529.aspx.

Также полезно отслеживать и регистрировать любые исключения, которые возникают в процессе работы веб-приложения, таким образом вы сможете проанализировать обстоятельства возникновения этих исключений и при необходимости внести коррективы в способ функционирования приложения. Вы можете использовать программные функции, такие как функциональный блок обработки исключений Exception Handling Application Block из библиотеки

Microsoft Enterprise Library, для фиксации и обработки исключений, а также можете записывать информацию об этих исключениях в службе Windows Azure Diagnostics с помощью функционального блока ведения журнала Logging Application Block из библиотеки Enterprise Library. Эти данные затем можно анализировать с помощью такого инструмента, как System Center Operations Manager с пакетом мониторинга Monitoring Pack для Windows Azure, система представит подробный отчет об исключениях в вашем приложении для всех узлов, а также выдаст предупреждения о том, что какое-либо из этих исключений требует вмешательства оператора.

**Примечание**

Дополнительная информация об использовании функционального блока Logging Application Block в решении Windows Azure представлена в статье «Using classic Enterprise Library 5.0 in Windows Azure» на сайте http://entlib.codeplex.com/releases/view/75025#DownloadId=336804.

Дополнительная информация об использовании функционального блока Exception Handling Application Block представлена в статье «The Exception Handling Application Block» на сайте http://msdn.microsoft.com/en-us/library/ff664698(v=PandP.50).aspx.

- **Большую часть времени ваша веб-роль и рабочая роль функционируют должным образом, но иногда они работают медленно и перестают отвечать на запросы. В такие периоды необходимо собирать дополнительные диагностические данные, которые помогут определить причину проблемы. Вам необходимо получить возможность изменять конфигурацию монитора диагностики Windows Azure на любом вычислительном узле, не останавливая и не перезапуская этот узел.**

Чтобы свести к минимуму накладные расходы, связанные с ведением журналов, активными по умолчанию должны быть только журналы трассировки, журналы инфраструктуры и журналы IIS. Если вам нужно анализировать показания счетчиков производительности, журналы событий операционной системы, журналы неудавшихся запросов IIS, аварийные дампы или другие произвольные журналы и файлы, вы должны явно активировать соответствующие функции. Можно динамически изменять конфигурацию Windows Azure Diagnostics, обращаясь к Windows Azure SDK из своих приложений и служб. Дополнительная информация представлена в статье «How to: Initialize the Windows Azure Diagnostic Monitor and Configure Data Sources» на сайте http://msdn.microsoft.com/en-us/library/windowsazure/gg433049.aspx.

Вы также можете удаленно настроить службу Windows Azure Diagnostics для веб-роли или рабочей роли с помощью Windows Azure SDK. Этот подход позволяет внедрить локальное пользовательское приложение, которое соединяется с узлом с помощью веб-роли или рабочей роли, определяет подлежащую сбору диагностическую информацию и периодически передает ее в хранилище Windows Azure. Более подробная информация представлена в статье «How to Remotely Change the Diagnostic Monitor Configuration» на сайте http://msdn.microsoft.com/en-us/library/windowsazure/gg432992.aspx.

**Мнение Бхарата**

Монитор диагностики Windows Azure периодически проверяет информацию о своей конфигурации, и любые изменения вступают в силу после очередной проверки. По умолчанию интервал проверки равен одной минуте. Вы можете изменить этот интервал, но не следует делать его слишком коротким, поскольку это может негативно повлиять на производительность монитора диагностики.

### Рекомендации по защите диагностической информации Windows Azure

Решение Windows Azure Diagnostics требует, чтобы контролируемые роли работали при полном доверии. Для атрибута **enableNativeCodeExecution** в файле ServiceDefinition.csdef определения службы для каждой роли необходимо задать значение «true». Эти значения установлены по умолчанию.

Диагностическая информация о вашей системе является конфиденциальной, поскольку характеризует структуру системы и параметры средств обеспечения безопасности. Этой информацией может воспользоваться злоумышленник, который пытается проникнуть в вашу систему. Поэтому вы должны надежно защищать учетные записи в системе хранения, которые используете для регистрации диагностических данных. Необходимо убедиться, что только авторизованные приложения имеют доступ к ключам этих учетных записей. Вы должны также обеспечить защиту всех данных, которыми обмениваются служба хранилища Windows Azure и ваши локальные приложения, с использованием протокола HTTPS.

Если вы создали локальные приложения или сценарии, которые могут динамически изменять параметры диагностики для любой роли, убедитесь, что только проверенные сотрудники, имеющие соответствующие разрешения, могут запускать эти приложения.

## Развертывание, обновление и восстановление функциональных возможностей с помощью интерфейса Windows Azure Service Management API и оболочки PowerShell

Портал управления Windows Azure предоставляет основной интерфейс для управления подписками Windows Azure. Вы можете использовать этот портал для загрузки приложений, а также управления размещенными сервисами и учетными записями хранилища данных. Однако вы также можете управлять своими приложениями Windows Azure программно через интерфейс Windows Azure Service Management API. Можно использовать этот API для создания собственных инструментов управления с целью развертывания, конфигурирования и управления веб-приложениями и службами. Вы также можете получить доступ к этим API при помощи командлетов Windows Azure PowerShell; такой подход позволяет быстро создавать сценарии, которые администратор может запустить с целью выполнения общих задач в сфере управления приложениями.

Пакет Windows Azure SDK предоставляет инструменты и утилиты, позволяющие разработчикам создавать собственные пакеты веб-ролей и рабочих ролей, а также средства развертывания этих пакетов в Windows Azure. Многие из этих инструментов и утилит также используют интерфейс Windows Azure Service Management API, а некоторые из них вызываются несколькими задачами и мастерами Microsoft Build Engine (MSBuild), созданными на основе шаблонов Windows Azure в Visual Studio.

**Мнение Маркуса**

Вы можете загрузить код примера приложения, который представляет собой клиентскую утилиту командной строки, применяемую для управления приложениями и службами Windows Azure: http://code.msdn.microsoft.com/windowsazure/Windows-Azure-CSManage-e3f1882c.

Вы можете загрузить командлеты Windows Azure PowerShell с сайта http://wappowershell.codeplex.com/.

### Рекомендации по применению Windows Azure Service Management API и PowerShell

Портал управления представляет собой мощное приложение, которое позволяет администратору управлять и настраивать службы Windows Azure, вместе с тем это интерактивный инструмент, который

требует от пользователей детального представления о структуре решения, размещении различных элементов и настройке параметров безопасности для этих элементов. Кроме того, пользователь должен знать идентификатор Windows Live и пароль для подписки Windows Azure вашей организации, и любой пользователь, у которого есть эти данные, получает полный контроль над всем решением. Если эти учетные данные попадут в руки злоумышленника или другого пользователя, который будет использовать их со злым умыслом, будут нарушены нормальная работа служб и ход бизнес-операций.

Следующие сценарии включают предложения по смягчению последствий подобных проблем:

- **Вы должны предоставлять контролируемый доступ оператору, чтобы он мог быстро выполнять повседневные задачи, например настраивать роли, предоставлять услуги, запускать и останавливать экземпляры ролей. Оператор не должен досконально разбираться в особенностях структуры приложений, а также не должен выполнять другие задачи, кроме указанных явно.**

  Это классический случай использования сценариев, включающих командлеты Windows Azure PowerShell. Вы можете предоставить ряд сценариев для выполнения различных задач, а после их параметризации оператор сможет предоставить любую дополнительную информацию, например имя файла пакета, содержащего веб-роль, которая будет развернута, или адрес экземпляра роли, подлежащей переводу в автономный режим. Такой подход позволяет управлять последовательностью выполнения задач, если оператор должен осуществлять сложные развертывания, включающие не только загрузку веб-ролей и рабочих ролей, но и предоставление, и управление базами данных SQL Azure, например.

  Чтобы запускать эти сценарии, оператору не требуются учетные данные для подписки Windows Azure. Вместо этого политика безопасности, обеспечиваемая посредством интерфейса Windows Azure Service Management API требует, чтобы учетная запись, которую оператор использует для запуска сценариев, была настроена с помощью соответствующих сертификатов управления (см. раздел «*Руководство по предоставлению защищенного доступа на управление подписками Azure»* далее в этом приложении). Чем меньше операторов знают учетные данные, необходимые для доступа к подписке Windows Azure, тем меньше вероятность случайного или преднамеренного разглашения этой информации неуполномоченным третьим лицам.

  Возможность создания и выполнения сценариев также обеспечивает последовательность и повторяемость, что снижает вероятность ошибок со стороны оператора, особенно если серии задач должны быть выполнены в отношении набора ролей и ресурсов, размещенных в разных центрах обработки данных.

  Недостаток этого подхода заключается в необходимости тщательного тестирования и проверки сценариев, а также последующего их сопровождения. Кроме того, применение сценариев — не самый лучший подход для реализации сложной логики, например с целью обработки ошибок и выполнения восстановления.

  **Мнение По**

  Сценарий, который создает или обновляет роли, должен развернуть эти роли в промежуточной среде в одном или нескольких центрах обработки данных в целях проведения тестирования перед предоставлением доступа для клиентов. Переход от промежуточной к производственной среде также может выполняться с использованием сценария, но только после завершения тестирования.

- **Вы должны предоставлять контролируемый доступ оператору, чтобы он мог быстро выполнять потенциально сложные серии задач, связанных с настройкой, развертыванием**

**или управлением системой. Оператор не должен досконально разбираться в особенностях структуры приложений, а также не должен выполнять другие задачи, кроме указанных явно.**

Этот сценарий является расширенным вариантом предыдущего, только здесь выполняются более сложные операции, которые в большей степени подвержены потенциальным ошибкам. В данном случае, возможно, предпочтительнее использовать интерфейс Windows Azure Service Management API непосредственно из пользовательского приложения, работающего локально. Это приложение может включать высокоэффективные инструменты обнаружения и обработки ошибок, а также логику для повторных попыток (при необходимости). Вы также можете сделать приложение более интерактивным, позволяя оператору предоставлять более подробную информацию, например адрес роли, подлежащей переводу в автономный режим, или имя файла пакета, который необходимо развернуть,— все эти задачи выполняются посредством графического пользовательского интерфейса с полной проверкой на наличие ошибок. Подход на основе мастера проще и менее подвержен ошибкам, по сравнению с подходом на основе сценариев, когда система ждет, что оператор укажет большое количество параметров в командной строке.

Пользовательское приложение также позволяет распределить задачи по группам, которые могут быть выполнены различными операторами или ролями. Приложение может аутентифицировать пользователя, оно только активирует функции и операции, имеющие отношение к идентификационной информации пользователя или роли, к которой принадлежит пользователь. Тем не менее вам следует постараться не делать приложение слишком сложным, необходимо сделать развернутые функциональные возможности простыми в использовании, а также предоставить по умолчанию понятные и продуманные значения для параметров, которые пользователь должен выбрать.

Клиентское приложение должно проверять все операции, выполняемые каждым пользователем. След аудита предоставляет полную информацию об обслуживании и управлении системой, и эти данные должны храниться в безопасном месте.

**Мнение Маркуса**

Windows Azure Service Management API на самом деле является оболочкой интерфейса REST, все запросы службы управления передаются как запросы HTTP REST. Поэтому не обязательно использовать приложения Windows для создания собственных приложений для управления, вы можете использовать любой язык программирования или среду, поддерживающую отправку запросов HTTP REST. Более подробная информация представлена в статье «Windows Azure Service Management REST API Reference» на сайте http://msdn.microsoft.com/en-us/library/windowsazure/ee460799.aspx.

К недостаткам интерактивных приложений можно отнести тот факт, что их не так просто использовать для автоматизации рутинных задач, выполнение которых планируется на периоды, когда нагрузка на ресурсы относительно невелика. В такой ситуации целесообразнее применять подход на основе сценария или решение, основанное на консольном приложении, которое работает в режиме командной строки.

- **Вы применяете решение Systems Center Operations Manager для мониторинга состояния своей системы. Если обнаружен сбой в одном или нескольких элементах, необходимо быстро устранить выявленные проблемы.**

Решение System Center Operations Manager может выдавать уведомления о важных событиях или ситуациях, когда показания счетчиков производительности превышают указанный порог. Вы можете по-разному реагировать на такие уведомления, например можно отправить сообщение оператору или инициировать выполнение сценария. Вы можете использовать эту функцию для обнаружения сбоев в работе компонентов системы и запуска сценария PowerShell, который попытается перезапустить эти компоненты. Более подробная информация представлена в статье «Enable Notification Channels» на сайте http://technet.microsoft.com/en-us/library/dd440882.aspx.

### Руководство по предоставлению защищенного доступа на управление подписками Azure

Интерфейс Windows Azure Service Management API гарантирует, что только авторизованные приложения смогут выполнять операции, это стало возможным благодаря взаимной аутентификации с использованием сертификатов SSL.

Когда локальное приложение отправляет запрос на доступ к управлению, оно должно предоставить ключ к сертификату управления, установленному на компьютере, на котором выполняется приложение. Сертификат управления должен иметь ключ длиной не менее 2048 бит и должен быть установлен в **персональном** хранилище сертификатов учетной записи, под которой выполняется приложение. Этот сертификат также должен включать закрытый ключ.

Этот сертификат должен быть доступен и в хранилище сертификатов управления в подписке Windows Azure, управляющей веб-приложениями и службами. Вы должны экспортировать сертификат с локального компьютера в виде файла с расширением «.cer» без закрытого ключа, а затем загрузить этот файл в хранилище сертификатов управления с помощью инструментов, предоставляемых порталом управления. Дополнительная информация о создании сертификатов управления представлена в статье «How to: Manage Management Certificates in Windows Azure» на сайте http://msdn.microsoft.com/en-us/library/windowsazure/gg551721.aspx.

**Мнение По**

Не забывайте о том, что в пакет Windows Azure SDK включены инструменты, позволяющие разработчикам создавать собственные пакеты веб-ролей и рабочих ролей, а также средства развертывания этих пакетов в Windows Azure. Эти инструменты также потребуют от вас указать сертификат управления. Однако вы должны быть осторожны, когда предоставляете разработчикам возможность загружать новые версии кода в свою производственную среду. Эта задача должна выполняться в управляемом режиме и только после тщательного тестирования предоставленного кода. Поэтому, вы должны либо отказаться от предоставления своим разработчикам сертификатов управления для доступа к подписке Windows Azure, либо создать для разработчиков отдельную подписку Windows Azure со своим собственным набором сертификатов управления, если они хотят протестировать свой код в облаке.