МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение высшего образования
«Дальневосточный федеральный университет»
(ДВФУ)

ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ (ШКОЛА)

«СОГЛАСОВАНО»
Руководитель ОП
_____ Пак Т.В.

«УТВЕРЖДАЮ»
Директор департамента Математического
и компьютерного моделирования
_____ Сущенко А.А.
« 15» июля 2021г.

РАБОЧАЯ ПРОГРАММА ДИСЦИПЛИНЫ
Облачные вычисления
Направление подготовки 02.03.01 Математика и компьютерные науки
(Сквозные цифровые технологии)
Форма подготовки очная

курс 4 семестр 8
лекции 16 час.
практические занятия не предусмотрены
лабораторные работы 16 час.
в том числе с использованием МАО лек. 10 / пр. - / лаб. 16 час.
всего часов аудиторной нагрузки 32 час.
в том числе с использованием МАО 26 час.
самостоятельная работа 85 час.
в том числе на подготовку к экзамену 27 час.
контрольные работы (количество) не предусмотрены
курсовая работа / курсовой проект не предусмотрены
зачет не предусмотрен
экзамен 8 семестр

Рабочая программа составлена в соответствии с требованиями Федерального государственного образовательного стандарта по направлению подготовки 02.03.01 Математика и компьютерные науки, утвержденного приказом Министерства образования и науки Российской Федерации от 23 августа 2017 года № 807 (с изменениями и дополнениями).

Рабочая программа обсуждена на заседании кафедры информатики, математического и компьютерного моделирования протокол № 6 от «28» января 2020г.

Директор департамента Математического и компьютерного моделирования Сущенко А.А.
Составитель: к.ф.-м.н, доцент Пак Т. В.

Владивосток
2021

Оборотная сторона титульного листа РПД

I. Рабочая программа пересмотрена на заседании кафедры информатики, математического и компьютерного моделирования:

Протокол от «09» июля 2021 г. № 7

Заведующий кафедрой _____ Чеботарев А.Ю.
(подпись) (И.О. Фамилия)

II. Рабочая программа пересмотрена на заседании департамента Математического и компьютерного моделирования:

Протокол от «27» сентября 2021 г. № 1

Директор департамента _____ Сущенко А.А.
(подпись) (И.О. Фамилия)

III. Рабочая программа пересмотрена на заседании департамента:

Протокол от «_____» _____ 20___ г. № _____

Директор департамента _____ _____
(подпись) (И.О. Фамилия)

IV. Рабочая программа пересмотрена на заседании департамента:

Протокол от «_____» _____ 20___ г. № _____

Директор департамента _____ _____
(подпись) (И.О. Фамилия)

Цели освоения дисциплины.

Сформировать у слушателей необходимый объем теоретических и практических знаний о технологии облачных вычислениях, умений и навыков практической реализации выгод облачных технологий в современном бизнесе, изучение инструментальных средств данной технологии.

Дисциплина нацелена на подготовку магистрантов к:

ознакомление с основными понятиями и терминологией облачных технологий;

ознакомление с областями применения облачных технологий;

ознакомление с инфраструктурой облачных вычислений;

освоение навыков системного администрирования для разработки и сопровождения приложений, развертываемых в облаках.

Для успешного изучения дисциплины «Облачные вычисления» у обучающихся должны быть сформированы следующие предварительные компетенции:

умение быстро осваивать новые предметные области, выявлять противоречия, проблемы и вырабатывать альтернативные варианты их решения;

способность использовать и применять углубленные знания в области прикладной математики и информатики.

Для изучения дисциплины студент должен:

Знать:

- Основные понятия и принципы устройства операционных систем;
- Принципы работы базовых сетевых протоколов;
- Основы систем контроля конфигурации;

Уметь:

- Настраивать работу основных сетевых протоколов;

– Диагностировать основные сетевые ошибки;

Владеть:

– Навыками отладки сетевых приложений;

В результате изучения данной дисциплины у обучающихся формируются следующие и профессиональные компетенции.

Задача профессиональной деятельности	**Объекты или область знания**	**Код и наименование профессиональной компетенции**	**Код и наименование индикатора достижения профессиональной компетенции**	**Основание (ПС, анализ иных требований, предъявляемых к выпускникам)**
Тип задач профессиональной деятельности: производственно-технологический				
-участие в организации научно-технических работ, контроле, принятии решений и определении перспектив, -контекстная обработка общенаучной и научно-технической информации, приведение ее к проблемно-задачной форме, анализ и синтез информации; -решение прикладных задач в области защищенных информационных и телекоммуникационных технологий и систем;	Математические и алгоритмические модели, программы, программные системы и комплексы, методы их проектирования и реализации, способы производства, сопровождения, эксплуатации и администрирования в различных областях, в том числе в междисциплинарных. Объектами профессиональной деятельности могут быть имитационные модели сложных процессов управления,	ПК-4 Способен к обоснованному выбору, проектированию и внедрению специальных технических и программно-математических средств в избранной профессиональной области	ПК-4.1 Знает основы проектирования Специальных технических и программно-математических средств в избранной профессиональной области ПК-4.2 Умеет использовать навыки в профессиональной деятельности для проектирования и внедрения специальных технических и программно-математических средств в	**Профессиональный стандарт** «Программист» **Профессиональный стандарт** «Менеджер по информационным технологиям» **Профессиональный стандарт** «Руководитель разработки программного обеспечения» **Профессиональный стандарт** "Системный аналитик" **Профессиональный стандарт** "Специалист по научно-исследовательским и опытно-конструкторским

	программные средства, администрирование вычислительных, информационных процессов, а также других процессов цифровой экономики.		избранной профессиональной области ПК-4.3 Владеет способностью к обоснованному выбору, проектированию и внедрению специальных технических и программно-математических средств в избранной профессиональной области	разработкам" **Профессиональный стандарт** "Специалист по тестированию в области информационных технологий"
Тип задач профессиональной деятельности: организационно-управленческий				

-Управление работами по созданию программных систем и комплексов. -Менеджмент проектов в области программирования и ИТ. -анализ рынка новых решений в области наукоемких технологий и пакетов программ для решения прикладных задач; -применение методов математического и алгоритмического моделирования при анализе прикладных проблем; -использование базовых математических задач и математических методов в научных исследованиях; -использование технологий и компьютерных систем управления объектами; -применение математических методов экономики, актуарно-финансового анализа и защиты информации;	Математические и алгоритмические модели, программы, программные системы и комплексы, методы их проектирования и реализации, способы производства, сопровождения, эксплуатации и администрирования в различных областях, в том числе в междисциплинарных. Объектами профессиональной деятельности могут быть имитационные модели сложных процессов управления, программные средства, администрирование вычислительных, информационных процессов, а также других процессов цифровой экономики.	ПК-6 Способен составлять и контролировать план выполняемой работы, планировать необходимые для выполнения работы ресурсы, оценивать результаты собственной работы	ПК-6.1. Знает методы организации работы в коллективах разработчиков ПО, направления развития методов и программных средств коллективной разработки ПО. ПК-6.2. Умеет использовать их в профессиональной деятельности, ПК-6.3. Имеет навыки коллективной разработки ПО	**Профессиональный стандарт** "Специалист по научно-исследовательским и опытно-конструкторским разработкам" **Профессиональный стандарт** «Менеджер по информационным технологиям» **Профессиональный стандарт** «Руководитель разработки программного обеспечения» **Профессиональный стандарт** "Системный аналитик" **Профессиональный стандарт** "Специалист по тестированию в области информационных технологий"

Для формирования вышеуказанных компетенций в рамках дисциплины «Облачные вычисления» применяются следующие методы активного/ интерактивного обучения:

- презентации с использованием доски, книг, видео, слайдов, компьютеров и т.п., с последующим обсуждением материалов,
- обратная связь с формированием общего представления об уровне владения знаниями студентов, актуальными для занятия,
- разминка с вопросами, ориентированными на выстраивание логической цепочки из полученных знаний (конструирование нового знания),
- коллективные решения творческих задач, которые требуют от студентов не простого воспроизводства информации, а творчества, поскольку задания содержат больший или меньший элемент неизвестности и имеют, как правило, несколько подходов,
- работа в малых группах (дает всем студентам возможность участвовать в работе, практиковать навыки сотрудничества, межличностного общения).

I.СТРУКТУРА И СОДЕРЖАНИЕ ПРАКТИЧЕСКОЙ ЧАСТИ КУРСА

Лабораторные работы (16 часов)

ЛАБОРАТОРНАЯ РАБОТА №1(2 часа)

Тема: **Подготовка рабочего места**

Цель: Целью данной работы является подготовка рабочего места для stand-alone разработки облачных приложений; знакомство с основными инструментами разработчика.

ЛАБОРАТОРНАЯ РАБОТА №2(2 часа)

Тема: **Создание первого проекта**

Цель: Целью данной практической работы является демонстрация создания проекта облачного решения. Особенности его запуска и контроля состояния при помощи Compute Emulator.

ЛАБОРАТОРНАЯ РАБОТА №3(2 часа)

Тема: **Настройка хранилища разработки в Visual Studio 2010**

Цель: Настройка строки подключения к хранилищу разработки, запуск хранилища разработки, обозреватель хранилищ Windows Azure. Создание хранилища данных с простой структурой данных (simple data structure).

ЛАБОРАТОРНАЯ РАБОТА №4(2 часа)

Тема: **Хранилище данных с реляционной структурой**

Цель: Особенности создания и работы с реляционным хранилищем данных.

ЛАБОРАТОРНАЯ РАБОТА №5(2 часа)

Тема: **Работа с Windows Azure Table**

Цель: Целью данной практической работы является работа с Windows Azure Table: создание таблицы, добавление данных, просмотр данных, редактирование и удаление сущностей таблицы.

ЛАБОРАТОРНАЯ РАБОТА №6(2 часа)

Тема: **Работа с Windows Azure Blob**

Цель: Соединение с хранилищем. Добавление и удаление Blob - объекта.

ЛАБОРАТОРНАЯ РАБОТА №7(2 часа)

Тема: **Работа с Windows AzureQueue**

Цель: Будут рассмотрены два небольших примера, демонстрирующих основы работы с очередями Windows Azure , на примере рабочей и веб - ролей.

ЛАБОРАТОРНАЯ РАБОТА №8(2 часа)

Тема: **Microsoft .Net Service Bus: обзор, обмен сообщениями, управление доступом**

Цель: Будут рассмотрены следующие вопросы: MS .Net Service Bus: обзор, концепция. Enterprise Service Bus, Internet Service Bus, обмен сообщениями.

II.УЧЕБНО-МЕТОДИЧЕСКОЕ ОБЕСПЕЧЕНИЕ САМОСТОЯТЕЛЬНОЙ РАБОТЫ ОБУЧАЮЩИХСЯ

Учебно-методическое обеспечение самостоятельной работы обучающихся по дисциплине «Cloud computing (Облачные вычисления))» представлено в Приложении 1 и включает в себя:

план-график выполнения самостоятельной работы по дисциплине, в том числе примерные нормы времени на выполнение по каждому заданию;

характеристика заданий для самостоятельной работы обучающихся и методические рекомендации по их выполнению;

требования к представлению и оформлению результатов самостоятельной работы;

критерии оценки выполнения самостоятельной работы.

компетенций в процессе освоения образовательной программы, представлены в Приложении 2.

III.КОНТРОЛЬ ДОСТИЖЕНИЯ ЦЕЛЕЙ КУРСА

№ п/п	Контролируемые разделы / темы дисциплины	Коды и этапы формирования компетенций		Оценочные средства	
				текущий контроль	промежуточная аттестация
1	Тема: Подготовка рабочего места	ПК-4, ПК-6	Знает	Лабораторная работа 1 (ПР-5)	Отчет по лабораторной работе1
			Умеет	Лабораторная работа 1 (ПР-5)	Отчет по лабораторной работе1
			Владеет	Лабораторная работа 1 (ПР-5)	Отчет по лабораторной работе1
2	Тема: Создание первого проекта	ПК-4, ПК-6	Знает	Лабораторная работа 2 (ПР-5)	Отчет по лабораторной работе 2
			Умеет	Лабораторная работа 2 (ПР-5)	Отчет по лабораторной работе 2
			Владеет	Лабораторная работа 2 (ПР-5)	Отчет по лабораторной работе 2
3	Тема: Настройка хранилища разработки в Visual Studio 2010 .	ПК-4, ПК-6	Знает	Лабораторная работа 3 (ПР-5)	Отчет по лабораторной работе 3
			Умеет	Лабораторная работа 3 (ПР-5)	Отчет по лабораторной работе 3
			Владеет	Лабораторная работа 3 (ПР-5)	Отчет по лабораторной работе 3
4	Тема: Хранилище данных с реляционной структурой	ПК-4, ПК-6	Знает	Лабораторная работа 4 (ПР-5)	Отчет по лабораторной работе 4
			Умеет	Лабораторная работа 4 (ПР-5)	Отчет по лабораторной работе 4
			Владеет	Лабораторная работа 4 (ПР-5)	Отчет по лабораторной работе 4
5	Тема: Работа с Windows Azure Table	ПК-4, ПК-6	Знает	Лабораторная работа 5 (ПР-5)	Отчет по лабораторной работе 5

			Умеет	Лабораторная работа 5 (ПР-5)	Отчет по лабораторной работе 5
			Владеет	Лабораторная работа 5 (ПР-5)	Отчет по лабораторной работе 5
6	Тема: Работа с Windows Azure Blob	ПК-4, ПК-6	Знает	Лабораторная работа 6 (ПР-5)	Отчет по лабораторной работе 6
			Умеет	Лабораторная работа 6 (ПР-5)	Отчет по лабораторной работе 6
			Владеет	Лабораторная работа 6 (ПР-5)	Отчет по лабораторной работе 6
7	Тема: Работа с Windows AzureQueue	ПК-4, ПК-6	Знает	Лабораторная работа 7 (ПР-5)	Отчет по лабораторной работе 7
			Умеет	Лабораторная работа 7 (ПР-5)	Отчет по лабораторной работе 7
			Владеет	Лабораторная работа 7 (ПР-5)	Отчет по лабораторной работе 7
8	Тема: Microsoft .Net Service Bus: обзор, обмен сообщениями, управление доступом	ПК-4, ПК-6	Знает	Лабораторная работа 8 (ПР-5)	Отчет по лабораторной работе 8
			Умеет	Лабораторная работа 8 (ПР-5)	Отчет по лабораторной работе 8
			Владеет	Лабораторная работа 8 (ПР-5)	Отчет по лабораторной работе 8

Типовые контрольные задания, методические материалы, определяющие процедуры оценивания знаний, умений и навыков и (или) опыта деятельности, а также критерии и показатели, необходимые для оценки знаний, умений, навыков и характеризующие этапы формирования компетенций в процессе освоения образовательной программы, представлены в Приложении 2.

IV.СПИСОК УЧЕБНОЙ ЛИТЕРАТУРЫ И ИНФОРМАЦИОННО-МЕТОДИЧЕСКОЕ ОБЕСПЕЧЕНИЕ ДИСЦИПЛИНЫ

Основная литература

1. Сафонов В.О. Платформа облачных вычислений Microsoft Windows Azure

   http://lib.dvfu.ru:8080/lib/item?id=IPRbooks:IPRbooks-16722&theme=FEFU

2. Windows в облаке / М. Бакиров, Открытые системы. СУБД. - N 4 (2009), С. 29-31 F.FESTU.00016B.06D67A, 2010 – 004.41/.42

   http://lib.dvfu.ru:8080/lib/item?id=chamo:660488&theme=FEFU

3. Введение в облачные вычисления / Клементьев И.П

   http://lib.dvfu.ru:8080/lib/item?id=IPRbooks:IPRbooks-16695&theme=FEFU

Дополнительная литература

1. Microsoft SQL Server 2008. Разработка баз данных : учебный курс Microsoft / Т. Тернстрем, Э. Вебер, М. Хотек, Русское идательство, 2010, 496 с. ил. Прил.: 1 CD.

   http://lib.dvfu.ru:8080/lib/item?id=chamo:381801&theme=FEFU

2. Проектирование информационных систем в Microsoft SQL Server 2008 и Visual Studio 2010, Бурков А. В., Интернет-Университет Информационных Технологий (ИНТУИТ)

   http://lib.dvfu.ru:8080/lib/item?id=IPRbooks:IPRbooks-16730&theme=FEFU

3. C # и платформа .NET / Э. Троелсен ; [пер. с англ. Р. Михеева]., Санкт-Петербург : Питер , 2010., Питер, 2002., 795 с.

   http://lib.dvfu.ru:8080/lib/item?id=chamo:390373&theme=FEFU

V.ПЕРЕЧЕНЬ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

- Операционная система Windows;

- Microsoft Office (Word, Excel);
- Компилятор с СИ++;
- Windows Azure;

VI.МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ОСВОЕНИЮ ДИСЦИПЛИНЫ

1. Рекомендации по планированию и организации времени, необходимого для изучения дисциплины.

Рекомендуется следующим образом организовать время, необходимое для изучения дисциплины:

Изучение теоретического материала по учебнику и конспекту – 1 час в неделю.

Подготовка к лабораторному занятию и работе в компьютерном классе – 1 час.

Тогда общие затраты времени на освоение курса «Cloud computing (Облачные вычисления)» студентами составят около 2 часов в неделю.

2. Указания по организации работы с контрольно-измерительными материалами. При подготовке к лабораторной работе необходимо сначала прочитать теорию по каждой теме. Отвечая на поставленный вопрос, предварительно следует понять, что требуется от Вас в данном случае, какой теоретический материал нужно использовать, наметить общий план решения.

VII.МАТЕРИАЛЬНО-ТЕХНИЧЕСКОЕ ОБЕСПЕЧЕНИЕ ДИСЦИПЛИНЫ

Образовательный процесс по дисциплине проводится в лекционных и компьютерных аудиториях.

Мультимедийная лекционная аудитория (мультимедийный проектор, настенный экран, документ-камера) о. Русский, кампус ДВФУ, корпус 20(D), ауд. D738, D654/D752, D412/D542, D818, D741, D945, D547, D548, D732

Компьютерные классы: (доска, 15 персональных компьютеров) о. Русский, кампус ДВФУ, корпус 20(D), D733, D733a, D734, D734a, D546, D546a, D549a (Кампус ДВФУ), оснащенные компьютерами класса Pentium и мультимедийными (презентационными) системами, с подключением к общекорпоративной компьютерной сети ДВФУ и сети Интернет.

Приложение 1

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«Дальневосточный федеральный университет»
(ДВФУ)

ШКОЛА ЕСТЕСТВЕННЫХ НАУК

УЧЕБНО-МЕТОДИЧЕСКОЕ ОБЕСПЕЧЕНИЕ САМОСТОЯТЕЛЬНОЙ РАБОТЫ ОБУЧАЮЩИХСЯ

по дисциплине «Cloud computing (Облачные вычисления)»

Направление подготовки 02.03.01 Математика и компьютерные науки

профиль «Сквозные цифровые технологии»

Форма подготовки очная

Владивосток
2020

План-график выполнения самостоятельной работы по дисциплине

№ п/п	Дата/сроки выполнения	Вид самостоятельной работы	Примерные нормы времени на выполнение	Форма контроля
1	1-2 недели семестра	Подготовка отчета по лабораторной работе 1	9 час.	Защита отчета
2	3-4 недели семестра	Подготовка отчета по лабораторной работе 2	9 час.	Защита отчета
3	5-6 недели семестра	Подготовка отчета по лабораторной работе 3	9 час.	Защита отчета
4	7-8 недели семестра	Подготовка отчета по лабораторной работе 4	9 час.	Защита отчета
5	9-10 недели семестра	Подготовка отчета по лабораторной работе 5	9 час.	Защита отчета
6	11-12 недели семестра	Подготовка отчета по лабораторной работе 6	9 час.	Защита отчета
7	13-14 неделя семестра	Подготовка отчета по лабораторной работе 7	9 час.	Защита отчета
8	15-18 неделя семестра	Подготовка отчета по лабораторной работе 8	9 час.	Защита отчета

Характеристика заданий для самостоятельной работы обучающихся и методические рекомендации по их выполнению

Самостоятельная работа студентов состоит из подготовки к лабораторным работам в компьютерном классе, работы над рекомендованной литературой.

При подготовке к лабораторным занятиям необходимо сначала прочитать основные понятия по теме. При выполнении задания нужно сначала понять, что требуется в задаче, какой теоретический материал нужно использовать, наметить план решения задачи. Рекомендуется использовать методические указания и материалы по курсу «Cloud computing (Облачные вычисления)», а также электронные пособия, имеющиеся на сервере Школы естественных наук.

Требования к представлению и оформлению результатов самостоятельной работы

Результатом самостоятельной работы являются отчеты по лабораторным работам.

В процессе подготовки отчетов к лабораторным работам у студентов развиваются навыки составления письменной документации и систематизации имеющихся знаний. При составлении отчетов рекомендуется придерживаться следующей структуры:

- Математическая постановка задачи;
- Описание метода решения;
- Описание алгоритма метода;
- Спецификация используемых функций и типов данных;
- Результаты эксперимента;
- Выводы и заключение.

Критерии оценки выполнения самостоятельной работы

Отчет по лабораторной работе должен полностью удовлетворять условию задачи. В случае некачественно выполненных отчетов (не соответствующих заявленным требованиям) результирующий балл за работу может быть снижен. Студент должен продемонстрировать отчетливое и свободное владение концептуально-понятийным аппаратом, научным языком и терминологией. Наличие всех отчетов является зачётом по предмету.

Зачёт по дисциплине может быть выставлена по результатам лабораторных работ.

Приложение 2

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«Дальневосточный федеральный университет»
(ДВФУ)

ШКОЛА ЕСТЕСТВЕННЫХ НАУК

ФОНД ОЦЕНОЧНЫХ СРЕДСТВ

по дисциплине «Cloud computing (Облачные вычисления)»

Направление подготовки 02.03.01 Математика и компьютерные науки
профиль «Сквозные цифровые технологии»
Форма подготовки очная

Владивосток
2020

Паспорт ФОС

Код и формулировка компетенции	Этапы формирования компетенции	
ПК-4 Способен к обоснованному выбору, проектированию и внедрению специальных технических и программно-математических средств в избранной профессиональной области	Знает	основы работы в составе научно-исследовательского и производственного коллектива
	Умеет	использовать организационно-управленческие навыки в профессиональной и социальной деятельности
	Владеет	организационно-управленческими навыками работы в составе научно-исследовательского и производственного коллектива
ПК-6 Способен составлять и контролировать план выполняемой работы, планировать необходимые для выполнения работы ресурсы, оценивать результаты собственной работы	Знает	основы составления планов работы с учетом ресурсов
	Умеет	составлять и контролировать план выполняемой работы
	Владеет	способностью составлять и контролировать план выполняемой работы, планировать необходимые для выполнения работы ресурсы, оценивать результаты собственной работы

Шкала оценивания уровня сформированности компетенций по дисциплине «Cloud computing (Облачные вычисления)»

Код и формулировка компетенции	Этапы формирования компетенции		критерии	показатели	Баллы
ПК-4 Способен к обоснованному выбору, проектировани ю и внедрению специальных	знает	основы экономических знаний в различных сферах жизнедеятельно сти	представление об основах экономических знаний	знание основ экономики	45-64

технических и программно-математических средств в избранной профессиональной области	умеет	анализировать экономические показатели профессиональной деятельности	умение анализировать экономические показатели профессиональной деятельности	умение создавать алгоритмы и программные продукты с учетом анализа экономических показателей	65-79
	владеет	навыками использования экономических знаний в различных сферах жизнедеятельности	применение навыков использования экономических знаний	систематическое применение навыков профессиональной работы с учетом анализа экономических показателей	80-100
ПК-6 Способен составлять и контролировать план выполняемой работы, планировать необходимые для выполнения работы ресурсы, оценивать результаты собственной работы	знает	основы правовых знаний в различных сферах жизнедеятельности	представление об основах правовых знаний	знание основ правоведения	45-64
	умеет	анализировать профессиональные показатели с учетом правовых знаний	умение анализировать показатели профессиональной деятельности с учетом правовых знаний	умение создавать алгоритмы и программные продукты с учетом правовых знаний	65-79
	владеет	навыками использования правовых знаний в различных сферах жизнедеятельности	применение навыков использования правовых знаний	систематическое применение навыков профессиональной работы с учетом анализа правовых последствий	80-100

№	Контролируемые разделы /	Коды и этапы формирования	Оценочные средства

п/п	темы дисциплины	компетенций		текущий контроль	промежуточная аттестация
1	Тема: Подготовка рабочего места	ПК-4 ПК-6	Знает	Лабораторная работа 1 (ПР-5)	Отчет по лабораторной работе1
			Умеет	Лабораторная работа 1 (ПР-5)	Отчет по лабораторной работе1
			Владеет	Лабораторная работа 1 (ПР-5)	Отчет по лабораторной работе1
2	Тема: Создание первого проекта	ПК-4 ПК-6	Знает	Лабораторная работа 2 (ПР-5)	Отчет по лабораторной работе 2
			Умеет	Лабораторная работа 2 (ПР-5)	Отчет по лабораторной работе 2
			Владеет	Лабораторная работа 2 (ПР-5)	Отчет по лабораторной работе 2
3	Тема: Настройка хранилища разработки в Visual Studio 2010 .	ПК-4 ПК-6	Знает	Лабораторная работа 3 (ПР-5)	Отчет по лабораторной работе 3
			Умеет	Лабораторная работа 3 (ПР-5)	Отчет по лабораторной работе 3
			Владеет	Лабораторная работа 3 (ПР-5)	Отчет по лабораторной работе 3
4	Тема: Хранилище данных с реляционной структурой	ПК-4 ПК-6	Знает	Лабораторная работа 4 (ПР-5)	Отчет по лабораторной работе 4
			Умеет	Лабораторная работа 4 (ПР-5)	Отчет по лабораторной работе 4
			Владеет	Лабораторная работа 4 (ПР-5)	Отчет по лабораторной работе 4
5	Тема: Работа с Windows Azure Table	ПК-4 ПК-6	Знает	Лабораторная работа 5 (ПР-5)	Отчет по лабораторной работе 5

			Умеет	Лабораторная работа 5 (ПР-5)	Отчет по лабораторной работе 5
			Владеет	Лабораторная работа 5 (ПР-5)	Отчет по лабораторной работе 5
6	Тема: Работа с Windows Azure Blob	ПК-4 ПК-6	Знает	Лабораторная работа 6 (ПР-5)	Отчет по лабораторной работе 6
			Умеет	Лабораторная работа 6 (ПР-5)	Отчет по лабораторной работе 6
			Владеет	Лабораторная работа 6 (ПР-5)	Отчет по лабораторной работе 6
7	Тема: Работа с Windows AzureQueue	ПК-4 ПК-6	Знает	Лабораторная работа 7 (ПР-5)	Отчет по лабораторной работе 7
			Умеет	Лабораторная работа 7 (ПР-5)	Отчет по лабораторной работе 7
			Владеет	Лабораторная работа 7 (ПР-5)	Отчет по лабораторной работе 7
8	Тема: Microsoft .Net Service Bus: обзор, обмен сообщениями, управление доступом	ПК-4 ПК-6	Знает	Лабораторная работа 8 (ПР-5)	Отчет по лабораторной работе 8
			Умеет	Лабораторная работа 8 (ПР-5)	Отчет по лабораторной работе 8
			Владеет	Лабораторная работа 8 (ПР-5)	Отчет по лабораторной работе 8

Шкала измерения уровня сформированности компетенций

Итоговый балл	1-44	45-64	65-79	80-100
Оценка (пятибалльная шкала)	2	3	4	5
Уровень сформированности компетенций	отсутствует	пороговый (базовый)	продвинутый	высокий (креативный)

Методические рекомендации, определяющие процедуры оценивания знаний, умений и навыков

Текущая аттестация студентов. Текущая аттестация студентов по дисциплине «Cloud computing (Облачные вычисления)» проводится в соответствии с локальными нормативными актами ДВФУ и является обязательной.

Текущая аттестация по дисциплине «Cloud computing (Облачные вычисления)» проводится в форме контрольных мероприятий (защита лабораторных работ) по оцениванию фактических результатов обучения студентов осуществляется ведущим преподавателем.

Объектами оценивания выступают:

- учебная дисциплина (активность на занятиях, своевременность выполнения различных видов заданий, посещаемость всех видов занятий по аттестуемой дисциплине);

- уровень овладения практическими умениями и навыками по всем видам учебной работы;

Оценочные средства для промежуточной аттестации

Промежуточная аттестация студентов по дисциплине «Cloud computing (Облачные вычисления)» проводится в соответствии с локальными нормативными актами ДВФУ в виде зачета в устной форме (ответы на вопросы для подготовки).

Вопросы для подготовки к зачёту

1. Сколько поколений компьютеров описывает история?
2. Каковы основные преимущества и недостатки блейд-систем?
3. Что понимается под Грид вычислениями?
4. Назовите основные преимущества облачных вычислений.
5. Назовите основные недостатки облачных вычислений.
6. Назовите основные преимущества виртуализации
7. Укажите основные разновидности виртуализации.
8. Назовите основные платформы виртуализации

9. Назовите основные преимущества Систем хранения данных.
10. Какие виды облаков существуют?
11. Укажите топологии сетей хранения данных
12. Что предоставляют поставщики услуг Iaas?
13. Что скрывается под аббревиатурой PaaS?
14. Что скрывается под аббревиатурой SaaS?
15. Отметьте основные преимущества SaaS для клиентов.
16. Основные назначения CaaS.
17. Назовите основные препятствия развитию облачных технологий в России.
18. Основные преимущества использования Windows Azure.
19. Что такое Windows Azure Table?
20. Отметьте базовые операции для таблиц и сущностей Windows Azure Table.
21. Что является компонентами облака Microsoft?
22. Сколько архитектурных уровней содержит модель SaaS согласно Microsoft?
23. Назовите компоненты Windows Azure Storage
24. Что такое Microsoft Live Workspace?
25. Что такое Windows Azure Blob?
26. Что такое Windows Azure Queue?
27. Отметьте основные возможности Google Apps.
28. Вопросы безопасности, масштабирования, развертывания, резервного копирования в контексте облачной инфраструктуры.
29. Особенности аварийного восстановления в облачной среде.

Задания по лабораторным работам

ЛАБОРАТОРНАЯ РАБОТА №1

Тема: **Подготовка рабочего места**

Цель: Целью данной работы является подготовка рабочего места для stand-alone разработки облачных приложений; знакомство с основными инструментами разработчика.

1. Установите VS 2010 и MS SQL Server 2008. Установка данного инструментария подробно описывается во множестве ресурсов и, как правило, не вызывает затруднений. (см. п№1-3 списка вспомогательных материалов)

2. Настройка IIS

для Windows 7

Откройте панель управления (Пуск - Панель управления)

Выберите "Программы и компоненты"

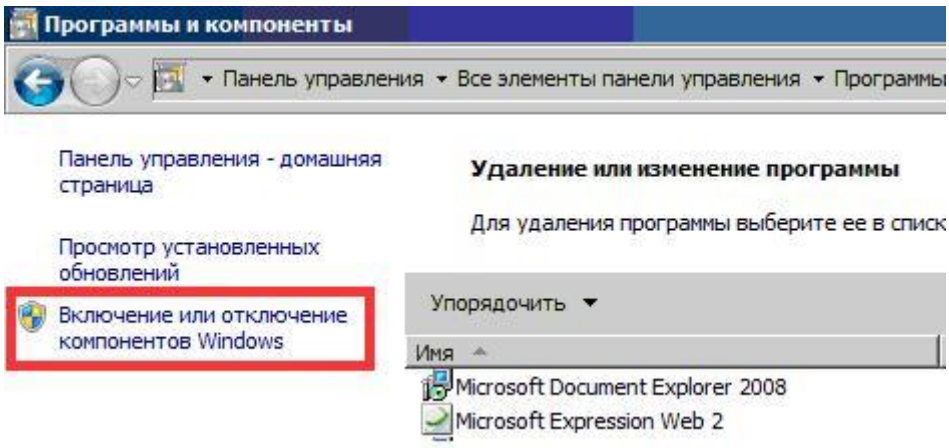


Рис. 11.1.

Раскройте узел "Microsoft .Ner Framework 3.5" и включите элемент "Windows Communication Foundation HTTP Activation"

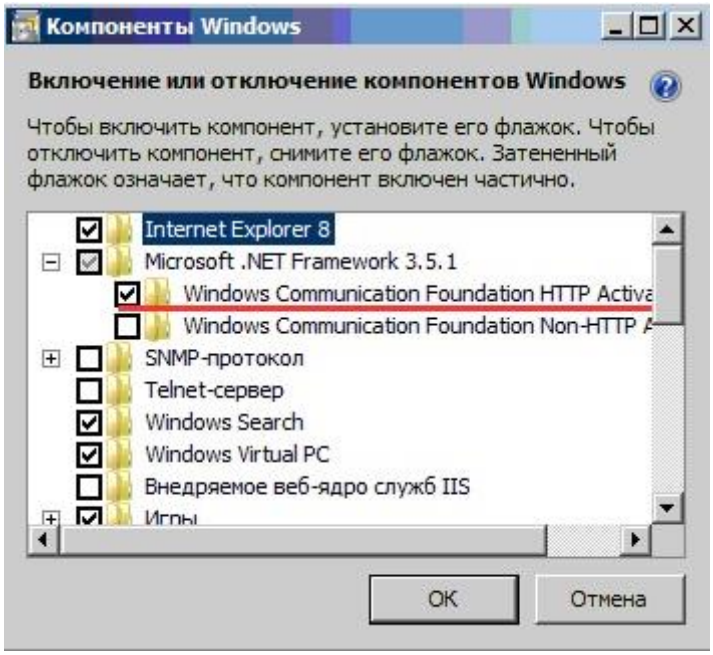


Рис. 11.2.

Последовательно раскройте узлы "Службы IIS", "Службы Интернета" и "Компоненты разработки приложений", отметьте элементы "ASP.NET" и "CGI".

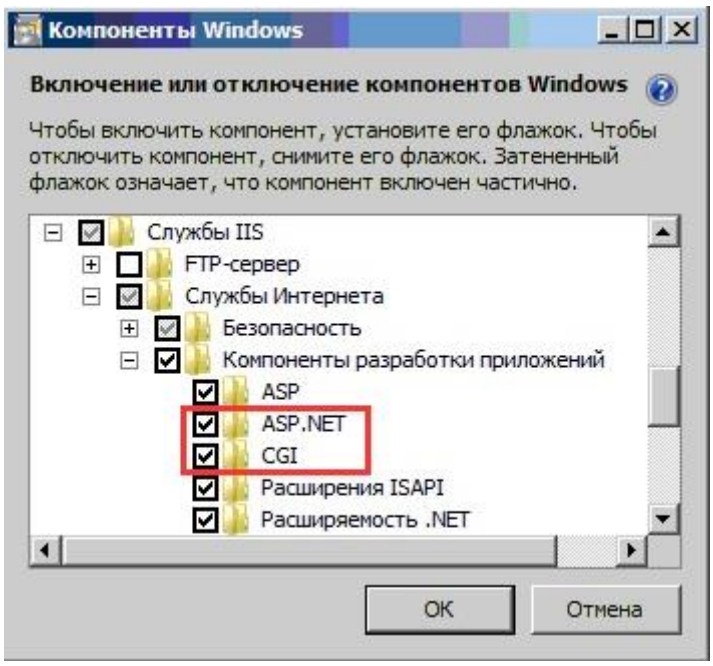


Рис. 11.3.

В узле "Службы IIS" разверните "Службы Интернета" и "Общие функции HTTP". Отметьте элемент "Статическое содержимое".

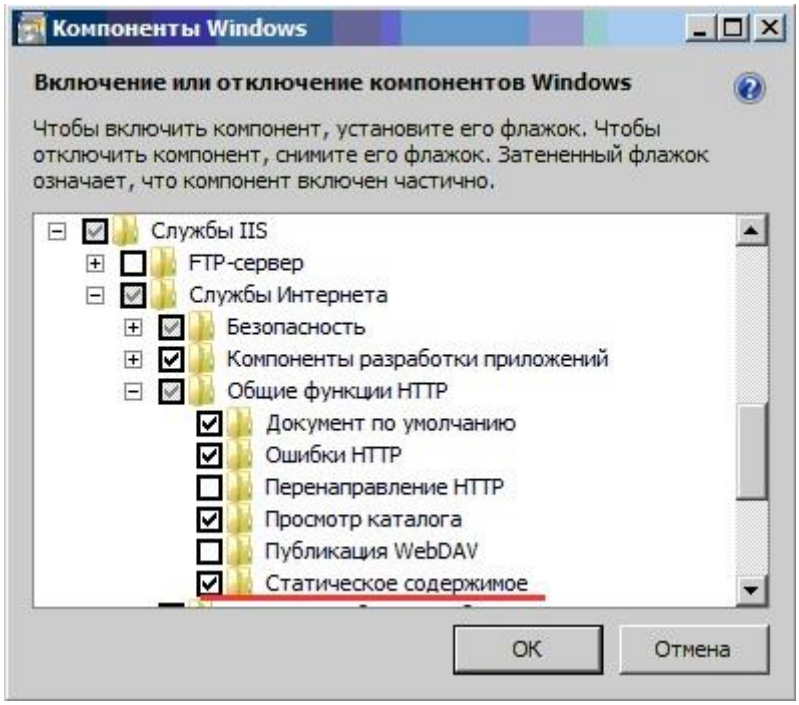


Рис. 11.4.

В узле "Службы IIS" разверните "Средства управления веб-сайтом" и отметьте "Консоль управления IIS".

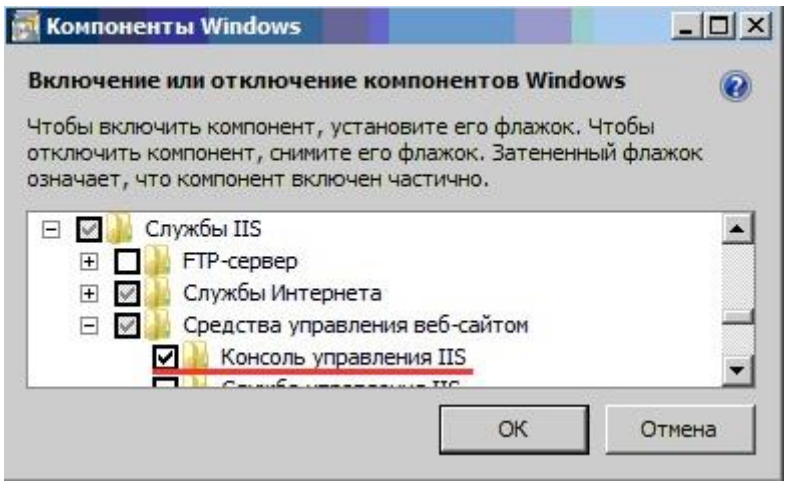


Рис. 11.5.

После этого нажмите "OK" и дождитесь завершения процесса установки отмеченных компонент.

Для Windows 2008

Запустите диспетчер сервера (Пуск - Администрирование - Диспетчер сервера)

Щелкните правой кнопкой мыши на узле "Компоненты" и выберите "Добавить компоненты"

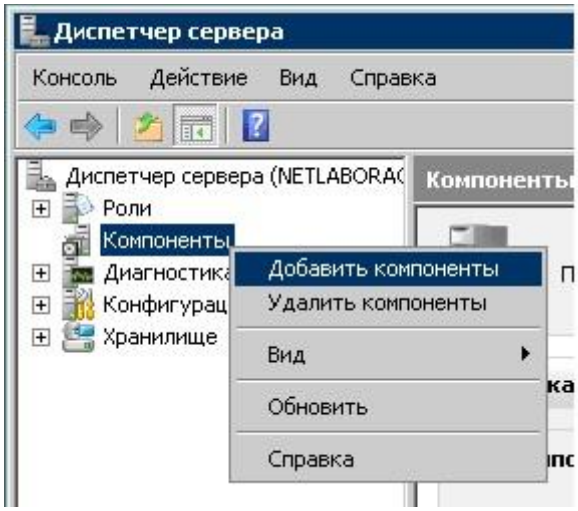


Рис. 11.6.

В списке компонентов, в узле "Возможности .Net Framework 3.0" отметьте ".Net Framework 3.0".

Рис. 11.7.

Аналогичным образом выберите элемент "Активация HTTP" в узле "Активация WCF". И нажмите "Далее". В случае, если появится диалоговое окно установки служб для данных компонент, установите их.

Дождитесь окончания установки и перейдите к узлу "Роли" диспетчера сервера.

Установите роль "Веб-сервер", нажав "Добавить роли"

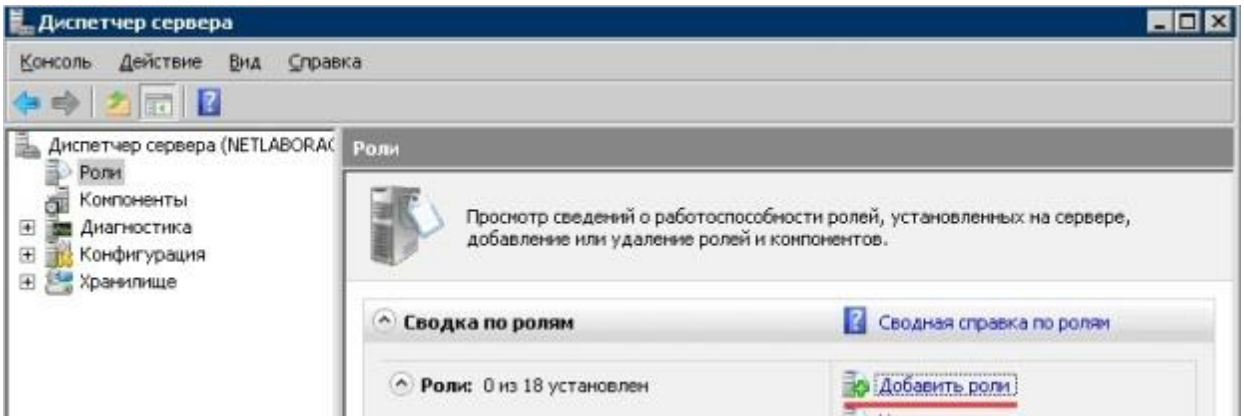


Рис. 11.8.

Рис. 11.9.

Последовательно нажимая Далее" установите необходимую роль.

В узле "Роли" выберите элемент "Веб - сервер(IIS) и нажмите "Добавить службы ролей""

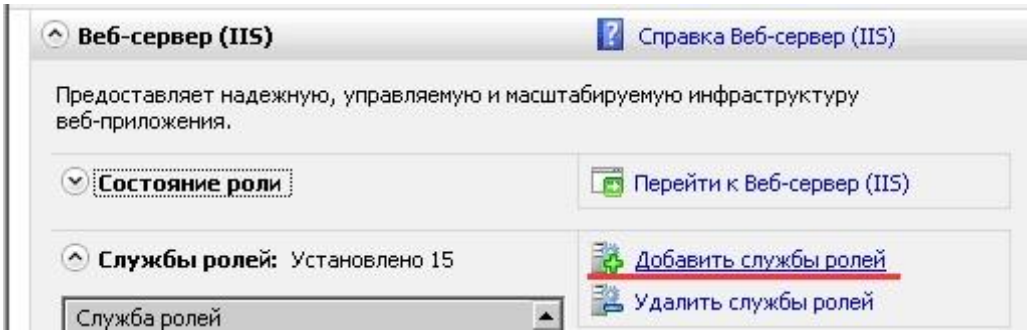


Рис. 11.10.

Выделите службы "Статическое содержимое", "ASP.NET" и "CGI" в узлах "Веб-сервер" и "Разработка приложений" соответственно, если они еще не установлены. Нажмите "Далее", затем "Установить".

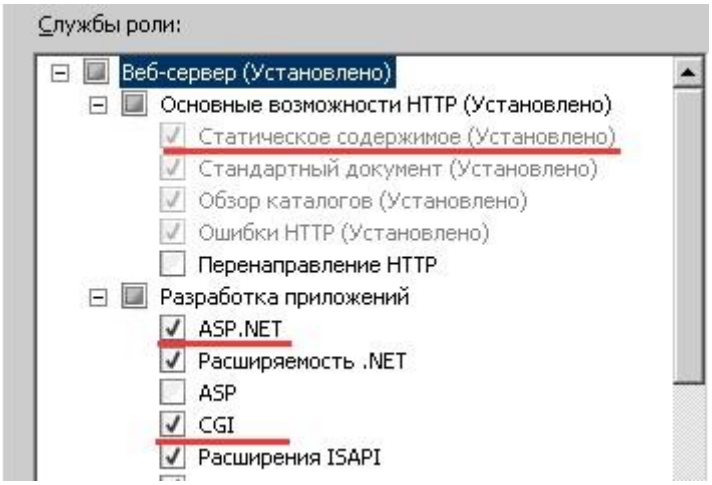


Рис. 11.11.

Дождитесь завершения процесса установки.

3. Установите VSCloudService.exe или Windows Azure SDK 1.3. Отметим, что SDK входит в состав VSCloudService.exe.

4. В случае, если у вас 32битная ОС установите исправление №8 списка требуемого ПО.

5. Установите исправления №№6-7 из списка требуемого программного обеспечения.

На этом установку инструментария stand-alone разработки облачных приложений можно считать завершенной.

Знакомство с инструментарием

Теперь необходимо разобраться с тем, что мы установили и где это все искать.

Эмуляторы Compute Emulator (Develop,ent Fabric) и Storage Emulator (development Storage) можно найти в папке Windows Azure SDK\v1.3 (по умолчанию C:\Program Files\Windows Azure SDK\v1.3\). Эмуляторы располагаются в подкаталоге bin.

Рис. 11.12.

Если установка инструментария завершена корректно, то в списке проектов VS2010

появится шаблон Cloud . Выбор проекта Windows Azure приведет к появлению списка доступных ролей .

Таблица 11.1. Список поддерживаемых ролей:

Роль	Описание
Веб-роль ASP.NET	Основанное на ASP.NET приложение с веб-интерфейсом
Веб-роль ASP.NET MVC 2	Основанное на ASP.NET MVC 2 приложение с веб-интерф
Веб-роль ASP.NET службы WCF	WCF - сервис
Рабочая роль	Создание фоновой задачи
Веб-роль CGI	Хостинг приложения с использованием FastCGI

Создание первого облачного приложения будет рассмотрено в следующей практической работе.

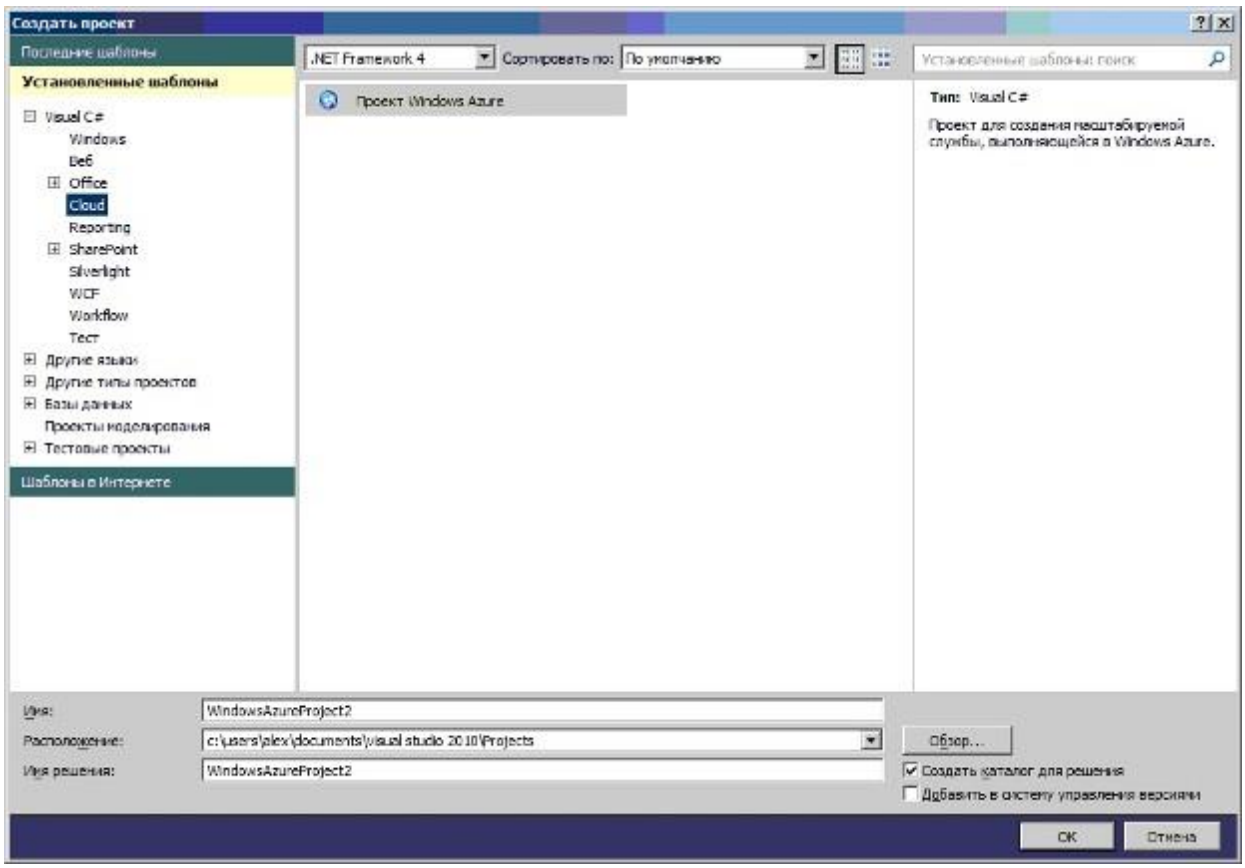


Рис. 11.13. Шаблон проекта Cloud

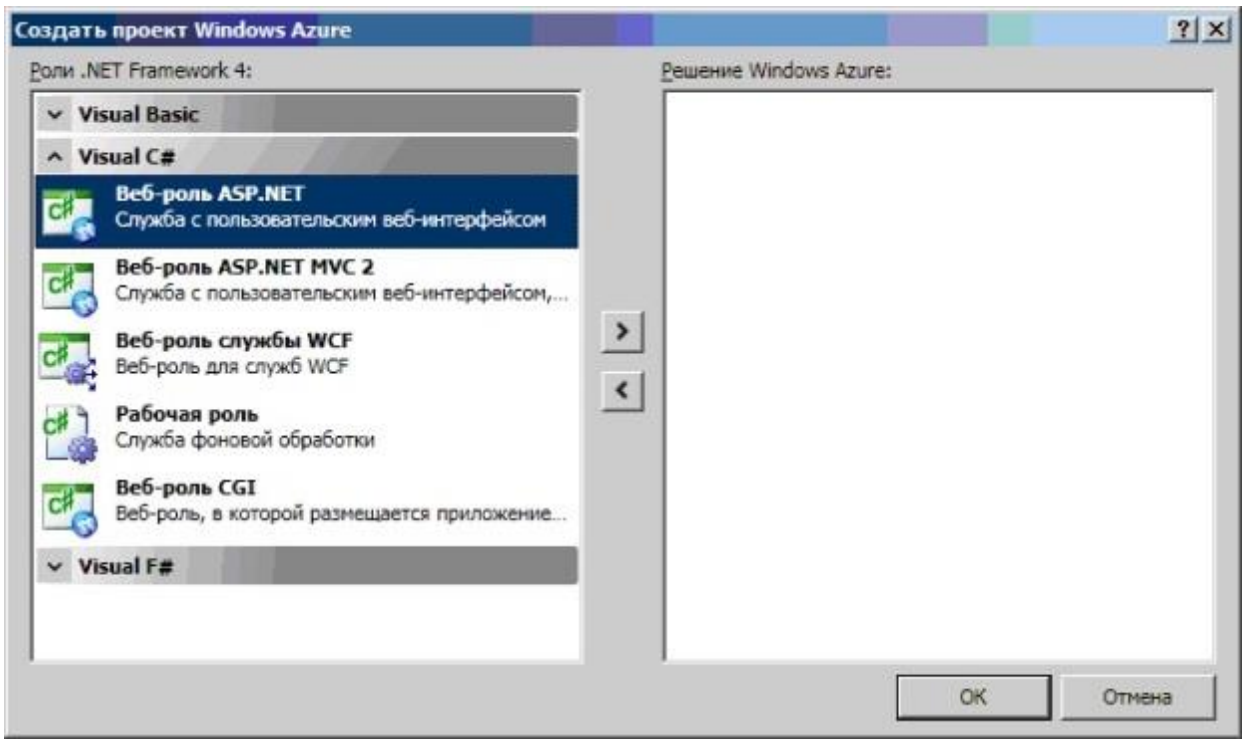


Рис. 11.14. Список ролей облачного приложения

Список вспомогательных материалов

Установка Visual Studio 2010

http://msdn.microsoft.com/ru-ru/library/e2h7fzkw.aspx

Установка MS SQL Server 2008

http://www.alta.ru/mssqlserver2008.php
http://itband.ru/2010/07/install-microsoft-sql-server-2008-r2/ (версии R2)

Руководства по устранению неполадок

http://msdn.microsoft.com/ru-ru/library/ee460770.aspx

ЛАБОРАТОРНАЯ РАБОТА №2

Тема: **Создание первого проекта**

Цель: Целью данной практической работы является демонстрация создания проекта облачного решения. Особенности его запуска и контроля состояния при помощи Compute Emulator.

Отметим, что для работы инструментов созания облачных решений необходимо запустить VS с правами администратора.

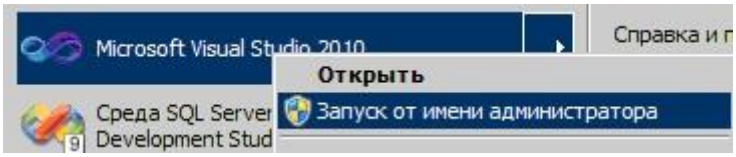


Рис. 12.1.

Запустим VS 2010 и создадим проект облачной службы с именем WA_first.

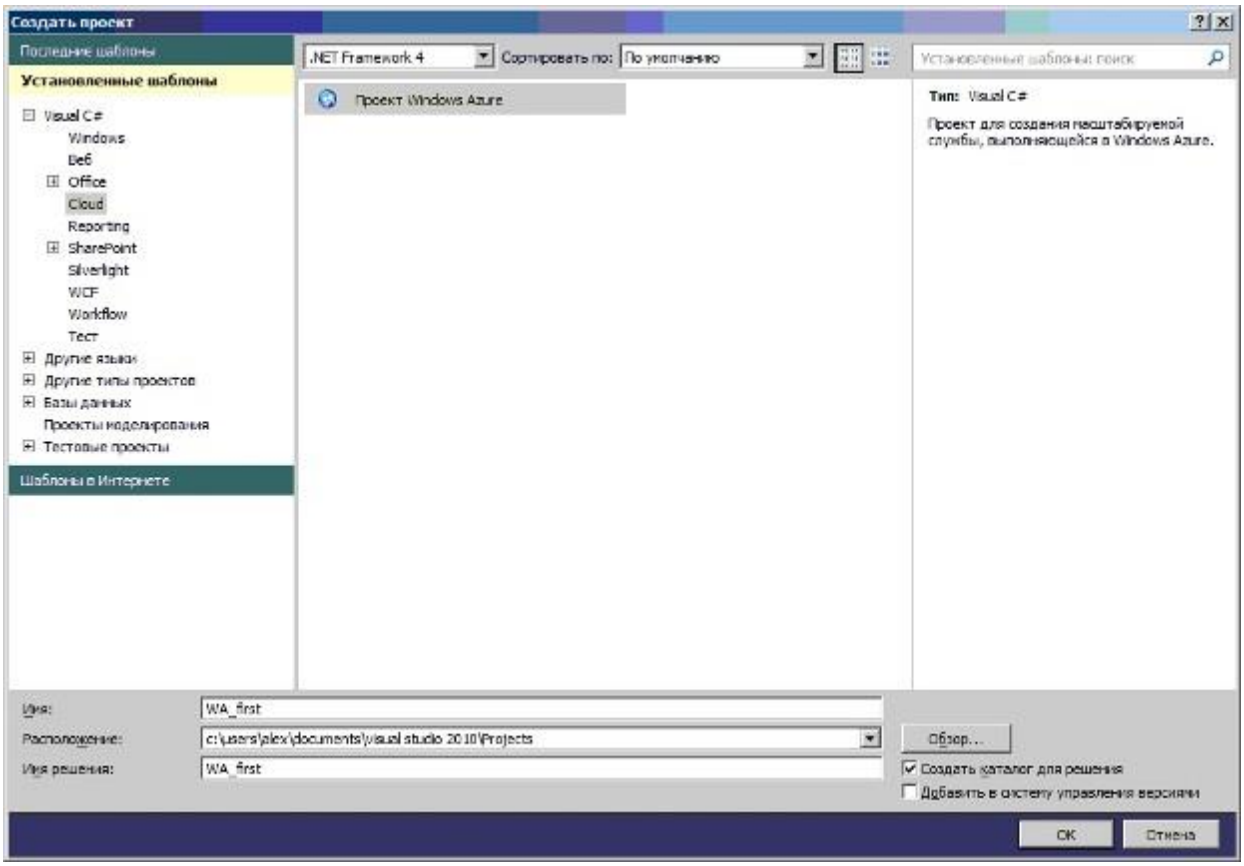


Рис. 12.2.

Добавим к проекту решения веб-роль ASP.NET и рабочую роль

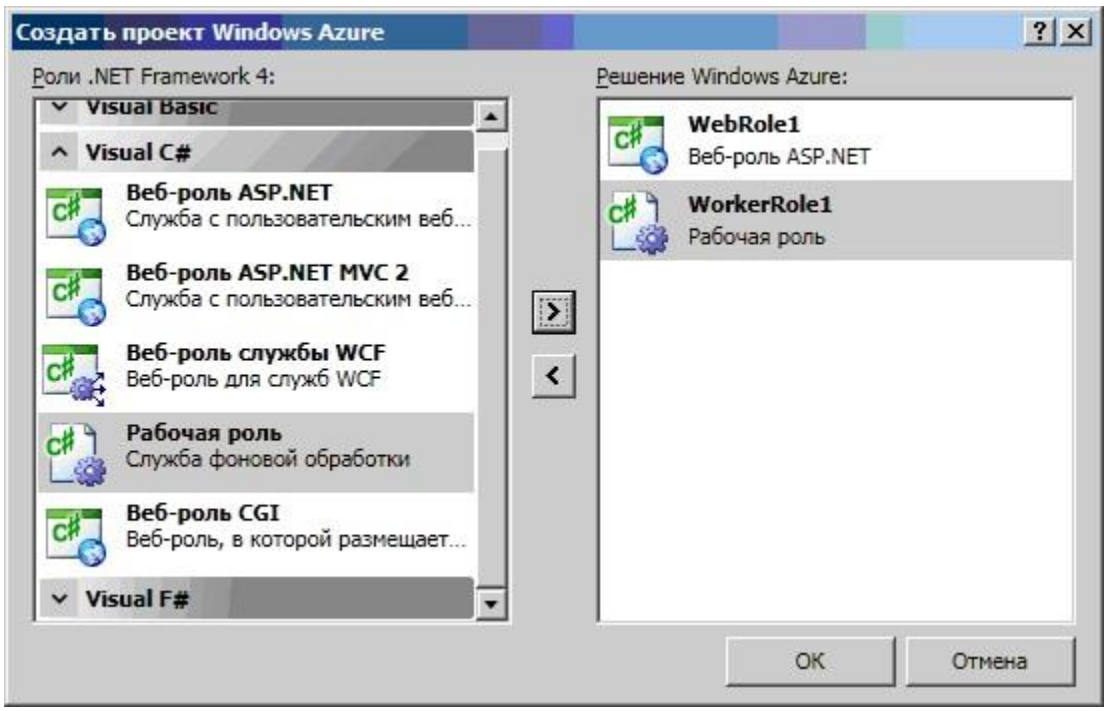


Рис. 12.3.

Результатом будет решение, состоящее из трех проектов

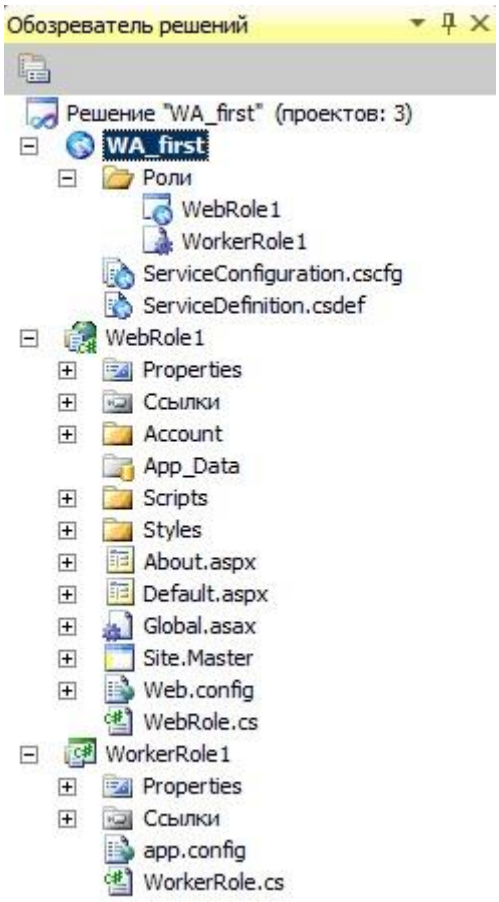


Рис. 12.4.

Файлы ServiceConfiguration.cscfg и ServiceDefinition.csdef являются

конфигурационными и используются для определения характеристик облачного решения и его ролей. Конфигурационные файлы упаковываются вместе с кодом и разворачиаются в Windows Azure.

ServiceDefinition.csdef - используется для описания приложения и ролей, хранит настройки одинаковые для всех экземпляров ролей. После запуска приложения содержимое данного файла не может быть изменено.

ServiceConfiguration.cscfg - задает значения настроек, описанных в ServiceDefinition.csdef, указывает число экземпляров каждой роли. Содержимое данного файла может быть изменено и после запуска роли.

Страницу свойств роли Windows Azure можно открыть щелкнув правой кнопкой мыши на роли и выбрав "Свойства"

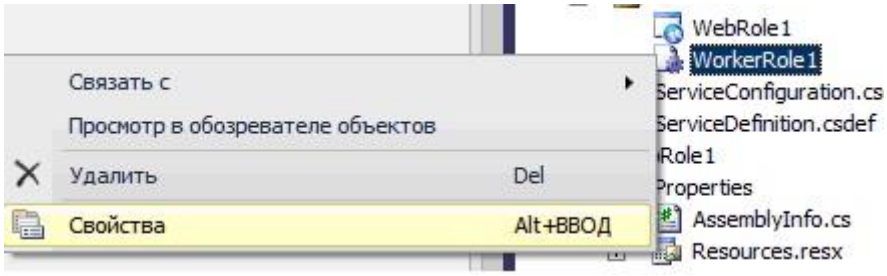


Рис. 12.5.

Рассмотрим свойства нашей рабочей роли

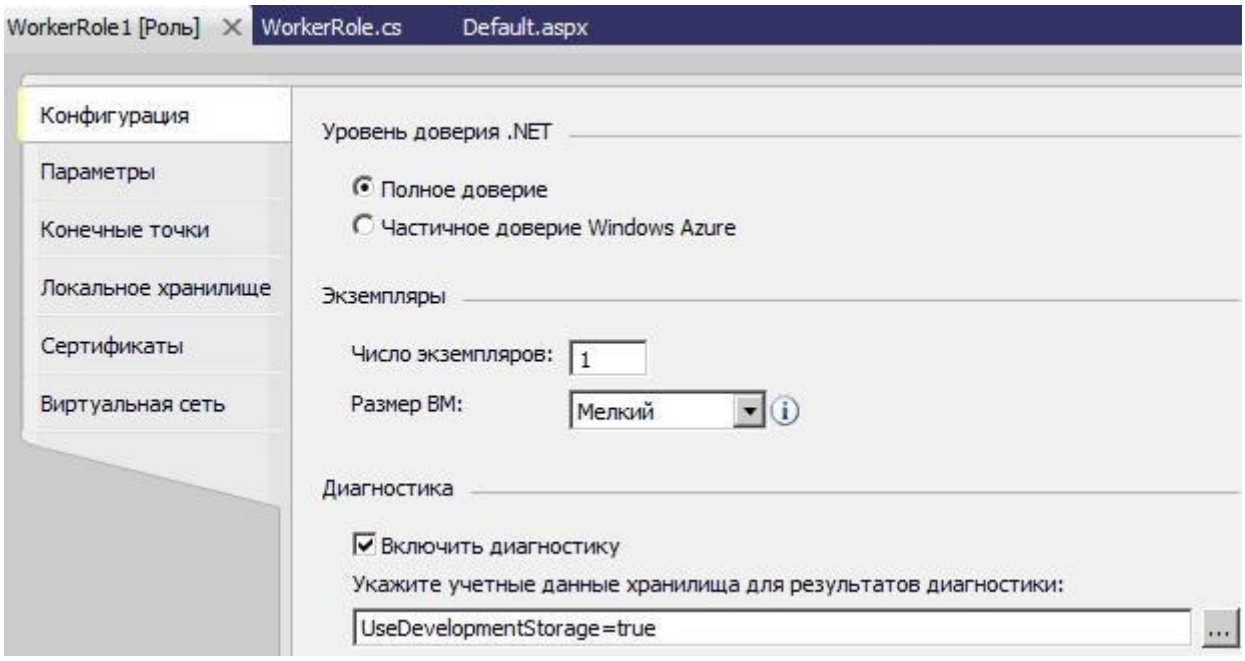


Рис. 12.6.

В разделе "Конфигурация" можно указать уровень доверия .NET. Полное доверие необходимо для выполнения собственного кода приложений FastCGI. Частичное доверие отключает возможность загрузки и использования клиентский библиотек Windows Azure.

Значение настройки "число экземпляров" определяет количество экземпляров, которые должны быть запущены для службы этой роли.

Свойства размер виртуальной машины определяет характеристики автоматически создаваемой виртуальной машины

Поскольку мы создаем приложение локально, в учетных данных хранилища в разделе "Диагностика" нужно отметить пункт "Использование эмулятора хранилища Windows Azure", либо вручную ввести строку "UseDevelopmentStorage=true".

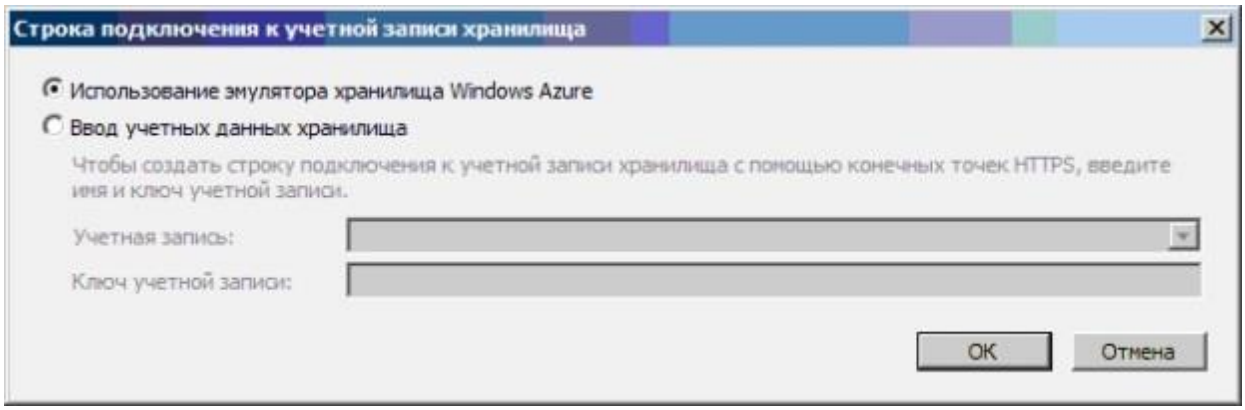


Рис. 12.7.

Запуск приложения в режиме эмуляции

Нажав кнопку F5, дождемся окончания построения облачного решения. Результатом будет веб-страница на локальном сервере. В правой части панели задач должна появиться иконка Windows Azure

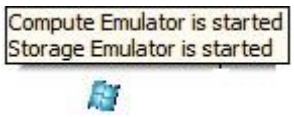


Рис. 12.8.

Для просмотра информации о работе приложения в режиме эмуляции, нужно щелкнуть правой кнопкой мыши по значку Windows Azure и выбрать "Show Compute Emulator UI"

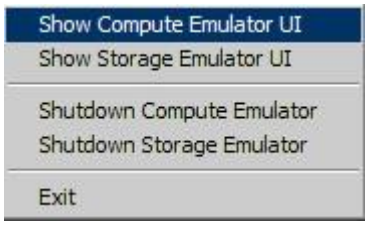


Рис. 12.9.

Появится окно для просмотра статусов ролей нашего приложения

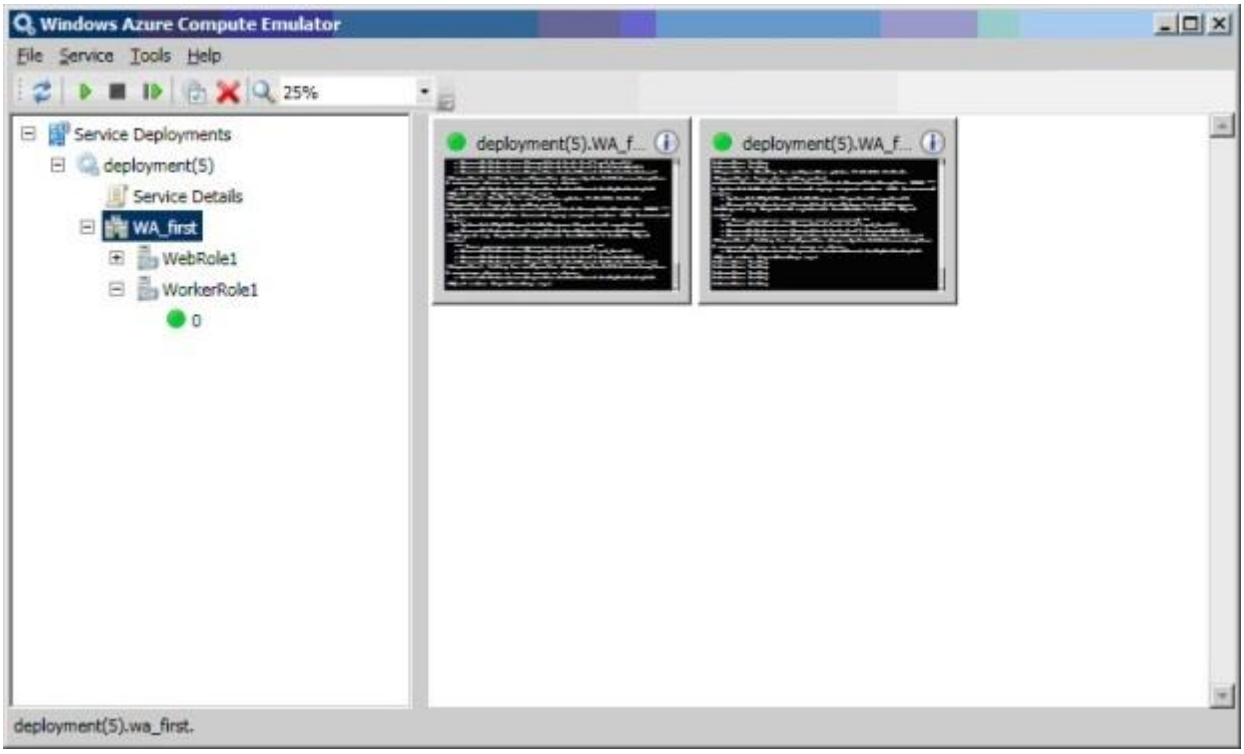


Рис. 12.10.

Compute emulator – это основной компонент Windows Azure, можно сказать, ядро "облачной" операционной системы, отвечающий за управление виртуальными машинами и экземплярами ролей. Развертывание и старт экземпляра роли начинают ее жизненный цикл.

Подробнее работа по созданию более сложных приложений будет рассмотрена в следующих лабораторных работах.

Список вспомогательных материалов

Настройка облачной службы

http://msdn.microsoft.com/ru-ru/library/ee405486.aspx

Уровни доверия .Net

http://msdn.microsoft.com/ru-ru/library/dd573355.aspx
http://msdn.microsoft.com/ru-ru/library/dd573345.aspx

Руководства по устранению неполадок

http://msdn.microsoft.com/ru-ru/library/ee460770.aspx

ЛАБОРАТОРНАЯ РАБОТА №3

Тема: **Настройка хранилища разработки в Visual Studio 2010**

Цель: Настройка строки подключения к хранилищу разработки, запуск хранилища разработки, обозреватель хранилищ Windows Azure. Создание хранилища данных с простой структурой данных (simple data structure).

Запуск хранилища разработки

Рассмотрим более подробно работу с Storage Emulator. По умолчанию Storage Emulator устанавливается в папку devstore подкаталога bin, папки Windows Azure SDK.

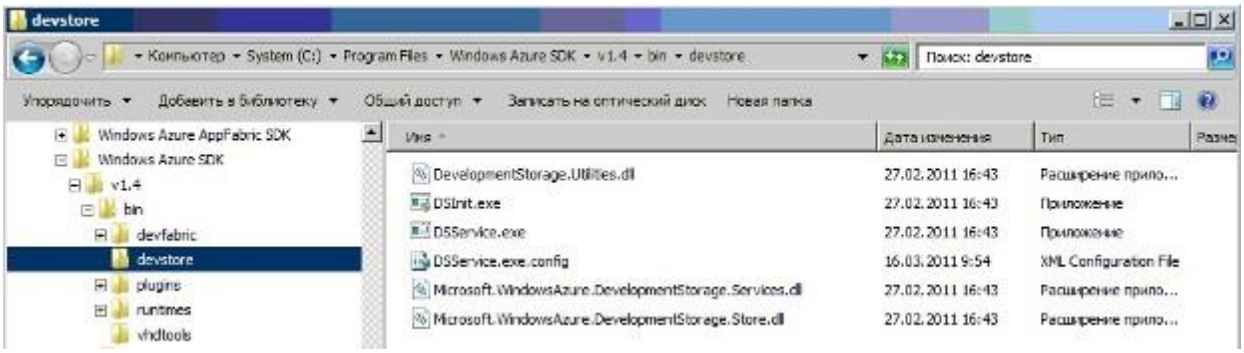


Рис. 14.1.

Здесь можно найти два .exe файла:

1. **DSInit** - инициализирует локальное хранилище и устанавливает права доступа к нему. Запустив этот файл, при отсутствии ошибок, должно появиться следующее окно:

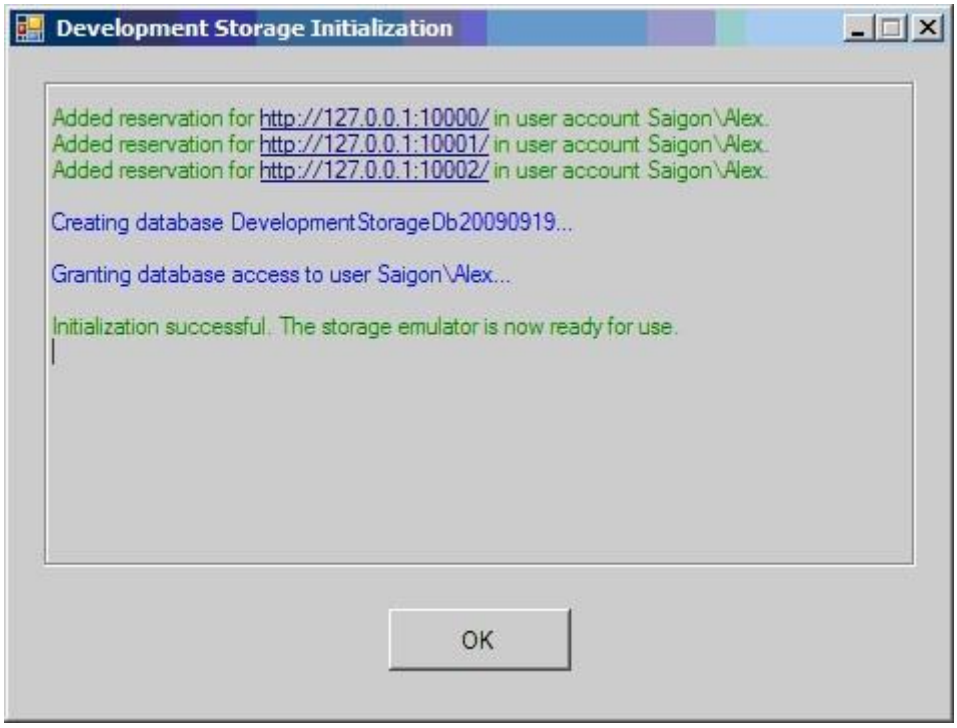


Рис. 14.2.

Как видно, была создана локальная база данных для разработки, и зарезервированы порты 10000-10002. В том, что база создана можно также убедиться, запустив SQL Management Studio

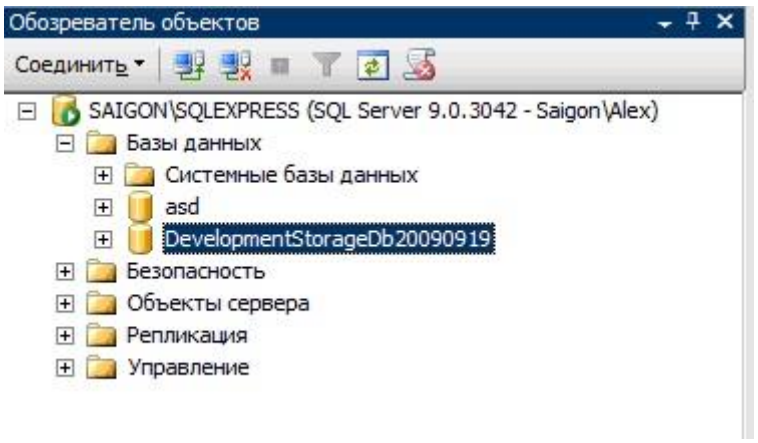


Рис. 14.3.

2. **DSService.exe** - непосредственно запускает эмулятор облачного хранилища. Запустив его и подождав некоторое время, можно заметить, что в правом нижнем углу появился значок Windows Azure. Для того, чтобы открыть интерфейс эмулятора, нужно щелкнуть правой кнопкой мыши по значку Windows Azure и выбрать "Show Compute Emulator UI". При отсутствии ошибок, должно появиться следующее окно:

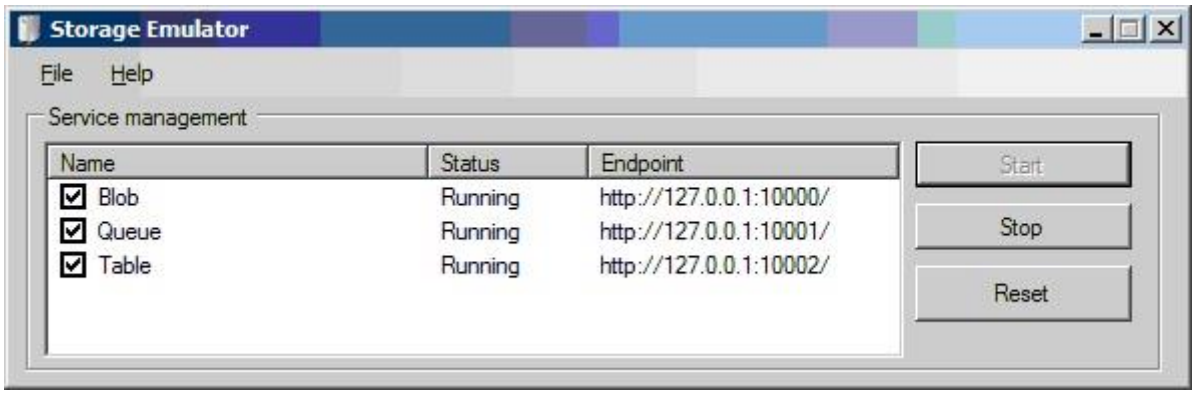


Рис. 14.4.

В окне Storage Emulator отображается состояние и конечные точки сервисов эмулятора: Blob, Queue и Table. Сервисы можно запустить, остановить, либо сбросить, с потерей данных, хранящихся в них.

Важно отследить следующий момент: порты, указанные для каждого из сервисов должны быть свободны. В случае, если при запуске хранилища разработки вы видите следующую ошибку:

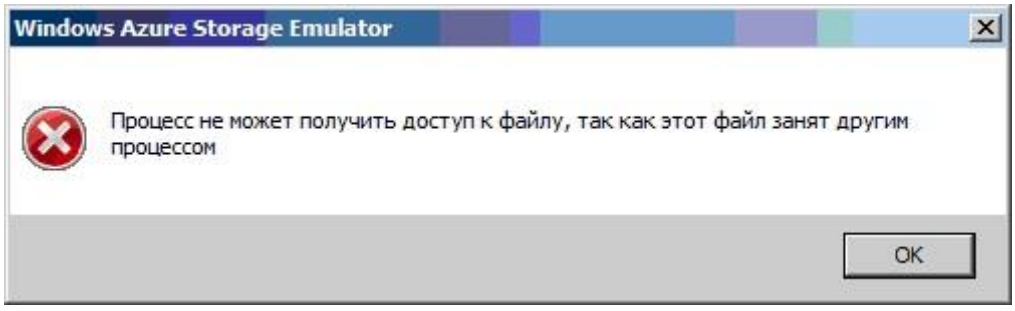


Рис. 14.5.

Это скорее всего означает, что указанные порты "слушают" другие приложения. К примеру, как отмечалось в ряде источников и было проверено нами, клиент mtorrent, запущенный до старта эмулятора конфликтует с Blob - сервисом, из-за чего к последнему не удается получить доступ.

Подключение к хранилищу разработки

Для подключения к эмулятору хранилища при разработке приложения (в среде Visual Studio в нашем случае) после создания проекта необходимо перейти к свойствам роли

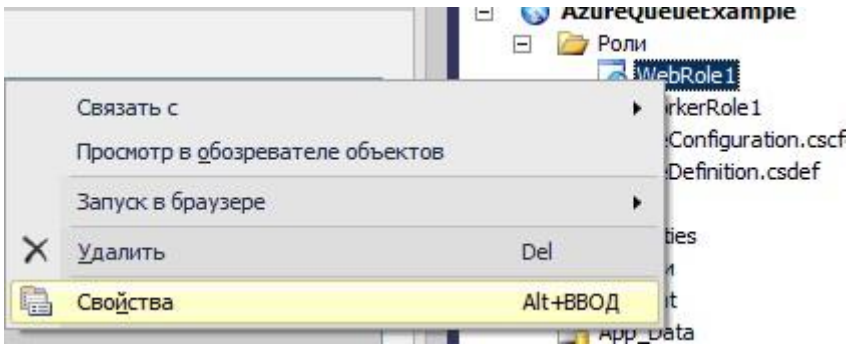


Рис. 14.6.

А затем во вкладке "Параметры" добавить параметр строки подключения и указать значение `"UseDevelopmentStorage=true"`

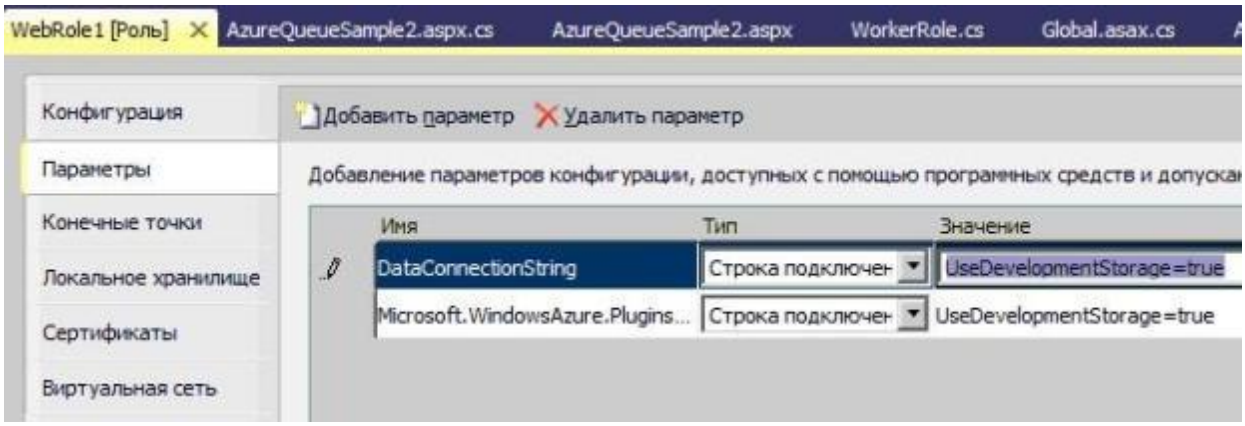


Рис. 14.7.

При работе непосредственно с хранилищем Windows Azure, в той же вкладке необходимо, при создании строки подключения, нажать кнопку "..."

Рис. 14.8.

И в появившемся окне задать параметры подключения и используемой учетной записи Windows Azure.

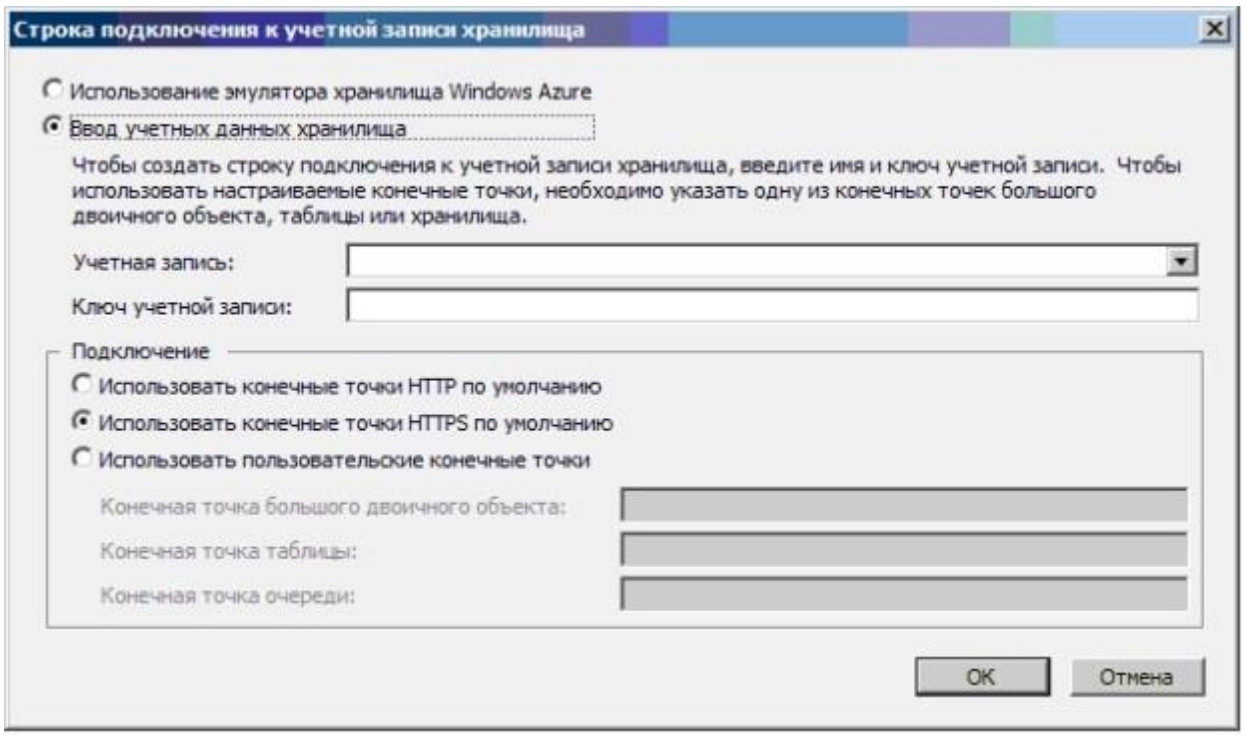


Рис. 14.9.

Запустив Обозреватель серверов (Меню "Вид" - Обозреватель серверов), увидим появившееся хранилище Windows Azure и хранилище "Разработка" - являющееся отображением эмулятора.

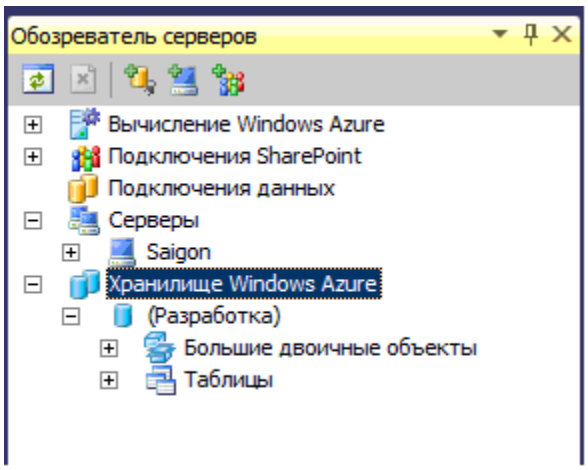


Рис. 14.10.

При помощи обозревателя серверов можно просматривать содержимое конкретных таблиц или контейнеров бинарных объектов.

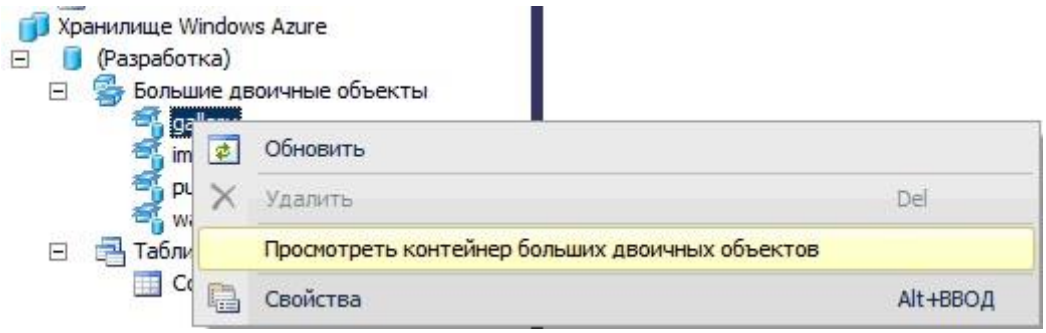


Рис. 14.11.

Также при помощи обозревателя серверов можно подключиться к хранилищу Windows Azure, добавив данные учетной записи.

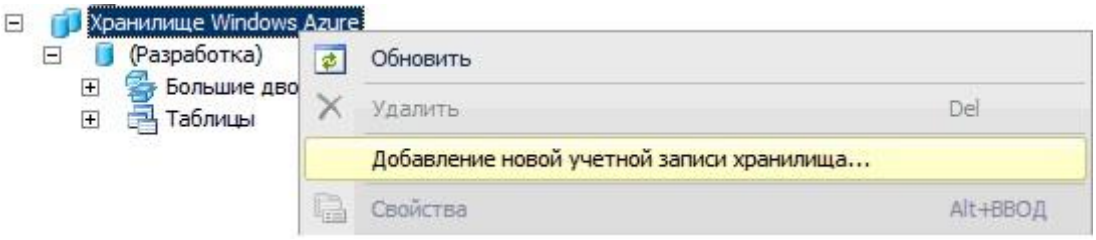


Рис. 14.12.

Для того, чтобы продемонстрировать подключение к хранилищу данных, рассмотри небольшой пример, по созданию хранилища с простой структурой - это будет таблица - список юридических лиц.

Задание. Создание хранилища с простой структурой данных.

1. Создадим проект облачной службы. SimpleDataStructure (Меню "Файл" - Создать - Проект). Добавим решению рабочую роль.

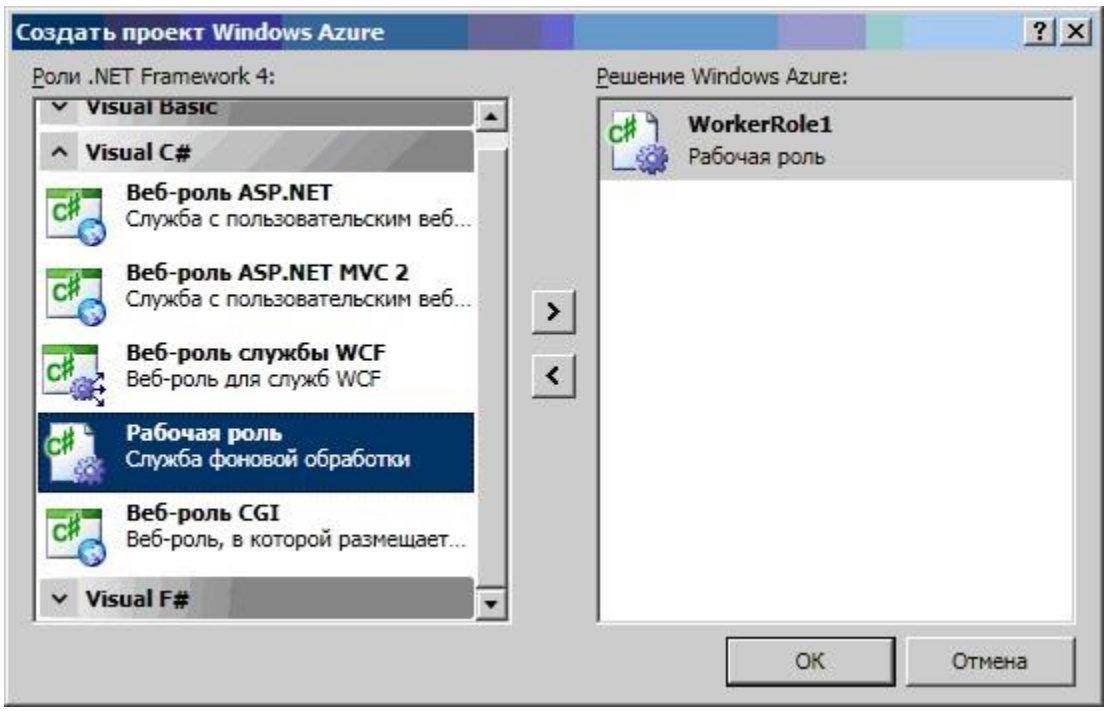


Рис. 14.13.

Наше приложение будет подключаться к эмулятору хранилища, создавать таблицу Firm, если ее не существует и добавлять туда одну произвольную запись.

В свойствах рабочей роли определим строку подключения к эмулятору хранилища Azure.

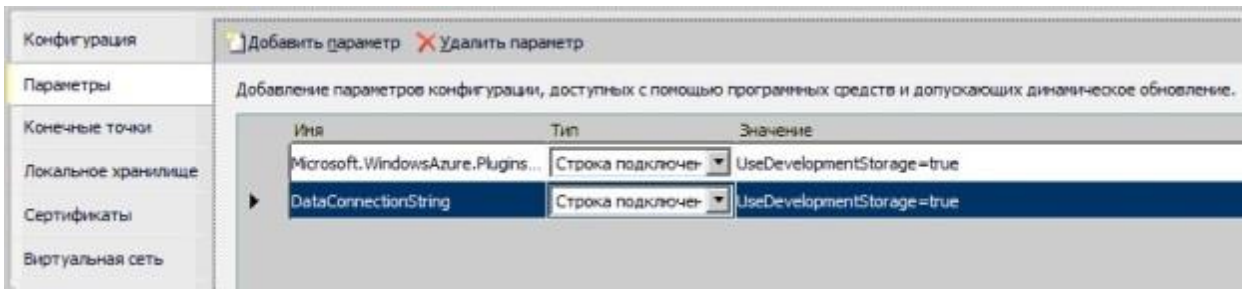


Рис. 14.14.

Нам необходим класс, который будет описывать структуру сущности для нашей таблицы. Класс должен быть наследником `Microsoft.WindowsAzure.StorageClient.TableServiceEntity`

```
class Address : TableServiceEntity
        {
            public String address { get; set; }
            public String firm { get; set; }
            public String telephone { get; set; }
        }
```

Для создания таблиц необходимо определить класс - контекст, при чем:

класс должен быть наследником TableServiceContext

Для каждой таблицы необходимо определить свойство типа `IQueryable`, где значение параметра `DataItemType` - тип сущностей, хранимых в таблице, в нашем случае - `Address`.

```
class AddressConext: TableServiceContext
        {
            public IQueryable<Address> ContactData
            {
                get
                {
                    return this.CreateQuery<Address>("Address");

                }
            }
            public AddressConext(Uri baseAddress,
StorageCredentials credentials) : base(baseAddress.AbsoluteUri,
credentials) { }
        }
```

Теперь обратим внимание на методы Run и OnStart. Первый содержит код выполняемый в ходе работы роли, второй - при ее запуске.

Для того, чтобы добавить данные в таблицу нам необходимо в методе Run:

создать экземпляр класса - учетной записи
создать экземпляр класс - контекста
создать экземпляр класса - сущности и задать его параметры
создать таблицу Address, если она не существует
добавить сущность в таблицу

Добавим следующий код:

```
CloudStorageAccount.SetConfigurationSettingPublisher(
                (configName, configSettingPublisher) =>
                {
                    var connectionString =

RoleEnvironment.GetConfigurationSettingValue(configName);
                    configSettingPublisher(connectionString);
                }
            );
//определение учетной записи
            CloudStorageAccount account =
CloudStorageAccount.FromConfigurationSetting("DataConnectionString
");

            //создание таблицы Windows Azure Table
            CloudTableClient _tc = null;
            _tc = account.CreateCloudTableClient();
            _tc.CreateTableIfNotExist("Address");
/*определение сущности, в том числе свойств ключ строки и ключ секции,
```

```
унаследованных от родительского TableServiceEntity*/
            Address adrs = new Address();
            adrs.PartitionKey = "Firm";
            adrs.RowKey = "Test entity";
            adrs.telephone = "xxx-xx-xx";
            adrs.address = "Evergreen Terrace 247";
            adrs.firm = "My new firm";
//определение контекста
            AddressConext context = new
AddressConext(account.TableEndpoint, account.Credentials);
//добавление сущности таблице Address
            context.AddObject("Address", adrs);
//сохранение изменений
            context.SaveChanges();
```

Запустите приложение, убедитесь в том, что оно выполнилось без ошибок и остановите его.

В диспетчере серверов, во вкладке "Хранилище Windows Azure" обновите вкладку "Таблицы", вы увидите созданную нашим приложением таблицу Address.

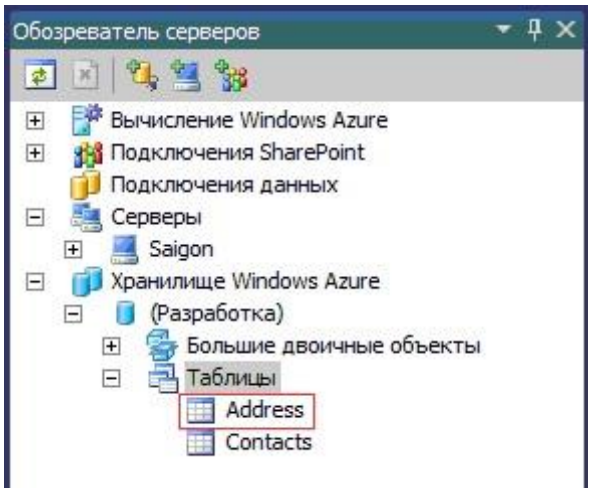


Рис. 14.15.

Щелкните на таблице правой кнопкой мыши и выберите "Просмотреть данные". Вы увидите, что определенная нами сущность добавлена в таблицу.

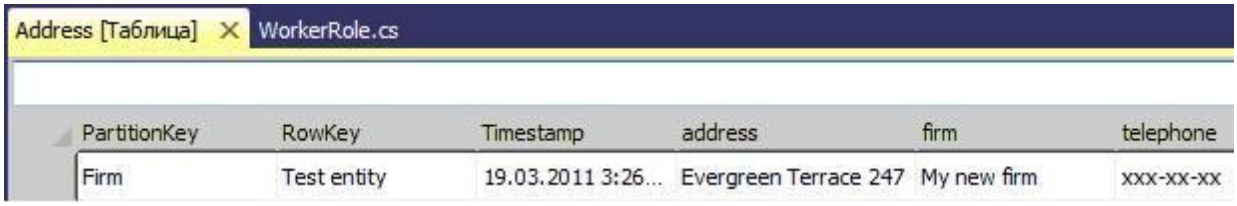


Рис. 14.16.

Более подробно работа с Хранилищем Windows Azure будет рассмотрена в последующих практических работах.

В случае, если выполнение задания вызвало сложности и затруднения, в приложениях к данной практической работе вы найдете итоговый программный код.

Список вспомогательных материалов

http://www.michaelfcollins3.me/2010/07/creating-table-storage-entities-for-windows-azure/

Приложение WorkerRole1.cs

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Net;
using System.Threading;
using Microsoft.WindowsAzure;
using Microsoft.WindowsAzure.Diagnostics;
using Microsoft.WindowsAzure.ServiceRuntime;
using Microsoft.WindowsAzure.StorageClient;
using System.Data.Services.Client;

namespace WorkerRole1
{
    public class WorkerRole : RoleEntryPoint
    {
        public override void Run()
        {
            CloudStorageAccount.SetConfigurationSettingPublisher(
                (configName, configSettingPublisher) =>
                {
                    var connectionString =

RoleEnvironment.GetConfigurationSettingValue(configName);
                    configSettingPublisher(connectionString);
                }
            );
            CloudStorageAccount account =
CloudStorageAccount.FromConfigurationSetting("DataConnectionString
");
            //создание таблицы Windows Azure Table
            CloudTableClient _tc = null;
            _tc = account.CreateCloudTableClient();
            _tc.DeleteTableIfExist("Address");
            _tc.CreateTableIfNotExist("Address");

            Address adrs = new Address();
            adrs.PartitionKey = "Firm";
```

```
            adrs.RowKey = "Test entity";
            adrs.telephone = "xxx-xx-xx";
            adrs.address = "Evergreen Terrace 247";
            adrs.firm = "My new firm";

            AddressConext context = new
AddressConext(account.TableEndpoint, account.Credentials);
            context.AddObject("Address", adrs);
            context.SaveChanges();
        }

        public override bool OnStart()
        {
            ServicePointManager.DefaultConnectionLimit = 12;
            return base.OnStart();
        }
        class Address : TableServiceEntity
        {
            public String address { get; set; }
            public String firm { get; set; }
            public String telephone { get; set; }
        }
        class AddressConext: TableServiceContext
        {
            public IQueryable<Address> ContactData
            {
                get
                {
                    return this.CreateQuery<Address>("Address");
                }
            }
            public AddressConext(Uri baseAddress,
StorageCredentials credentials) : base(baseAddress.AbsoluteUri,
credentials) { }
        }
    }
        }
```

ЛАБОРАТОРНАЯ РАБОТА №4

Тема: **Хранилище данных с реляционной структурой**

Цель: Особенности создания и работы с реляционным хранилищем данных.

Создадим базу данных при помощи запроса 16.1

```
create database azureexample
```

Листинг 16.1.

Теперь необходимо создать таблицы, связи между ними и заполнить их тестовым набором данных. Вы можете проделать это самостоятельно, а можно просто выполнить нижеследующий зарос:

```
use azureexample
```

```
create table Address(
AddresID int identity(1,1) primary key,
Country nvarchar(max),
City nvarchar(max),
Street nvarchar(max),
House int
)

create table Firm(
FirmID int identity(1,1) primary key,
NameOf nvarchar(max),
Telephone nvarchar(10),
Email nvarchar(max),
AddressKey int not null
)

create table Employee(
EmployeeID int identity(1,1) primary key,
FirstName nvarchar(max),
LastName nvarchar(max),
Telephone nvarchar(10),
FirmKey int
)

Alter table dbo.Firm add constraint
  FK_Firm_Address foreign key
  (
  AddressKey
  ) references dbo.Address
  (
  AddresID
  ) on update no action
   On delete no action

alter table dbo.Employee add constraint
FK_Employee_Firm foreign key
(
FirmKey
)
references dbo.Firm
(
FirmID
)
on update no action
on delete no action

insert into Address
values ('RF', 'Tomsk', 'Evergreen Terrace', '247')

insert into Firm
values ('Firm1', 'xxx-xx-xx', 'firm1@testmail.com', '1')
```

```
insert into Firm
values ('Firm2', 'xxx-xx-xx', 'firm2@testmail.com', '1')

insert into Employee
values ('Ivan', 'Ivanov', 'x-xxx-xx','1')

insert into Employee
values ('Victor', 'Romanov', 'x-xxx-xx','1')

insert into Employee
values ('Alex', 'Petrov', 'x-xxx-xx','2')
```

Таким образом мы получили три таблицы. Диаграмма данных созданной базы представлена ниже:

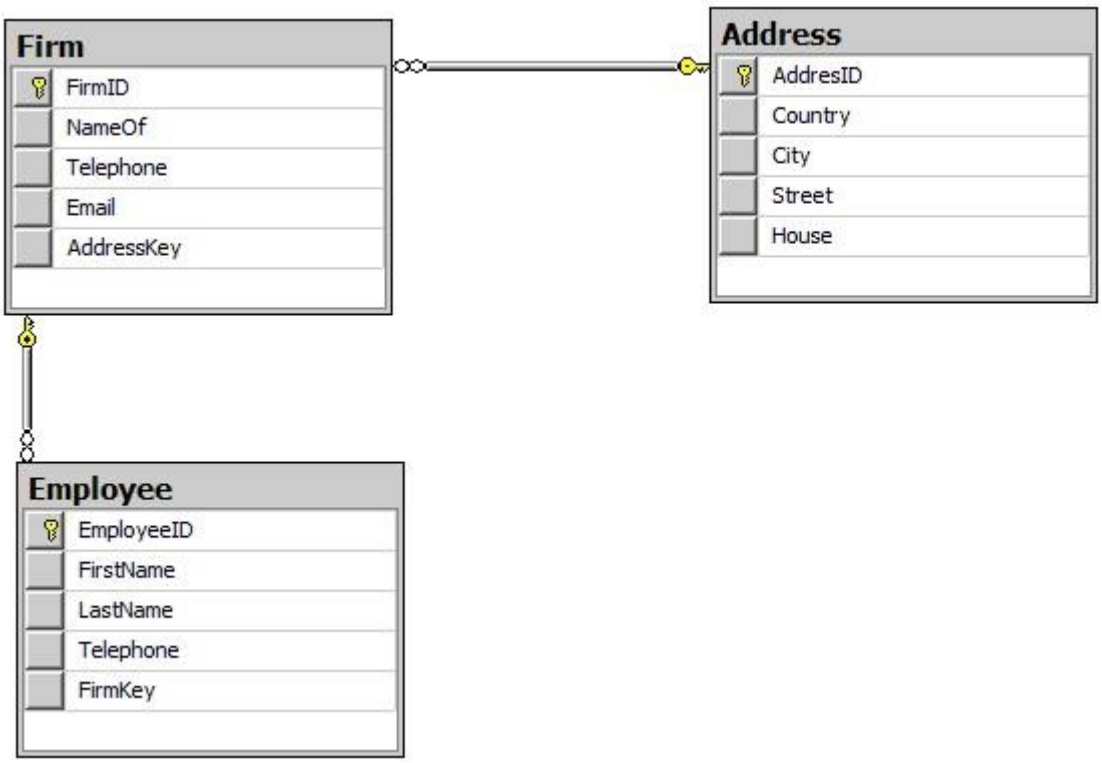


Рис. 16.1.

Теперь необходимо решить, каким образом данная реляционная структура может быть перенесена в таблицу. Как вам известно из лекции, таблицу Windows Azure можно соотнести с коллекцией взаимосвязанных сущностей, если проводить аналогии с реляционными базами данных. В то время, как строки таблиц Windows Azure, по сути являются экземплярами конкретных сущностей.

Мы предлагаем следующее очевидное решение:

Создать Windows Azure таблицу "Relational", которая будет содержать в себе сущности Address, Firm и Employee

Ключ секции будет указывать на соответствие строки таблицы Azure экземпляру сущности реляционной БД, т.е. для всех сотрудников ключ секции будет - "Employee"

Значения атрибутов - ключей в реляционных таблицах будут являться значениями RowKey для таблицы "Relational", такими образом пара ключ секции - ключ строки будет являться уникальным идентификатором однозначно указывающим на принадлежность Azure - строки конкретной сущности исходной БД и определяющим экземпляр сущности

Значения остальных атрибутов будут перенесены без изменений.

Мы не видим необходимости в создании нового VS - проекта, поэтому просто изменим WorkerRole.cs проекта, созданного в рамках предыдущей практической работы. А именно, метод **Run**, чей листинг представлен ниже:

```
CloudStorageAccount.SetConfigurationSettingPublisher(
                (configName, configSettingPublisher) =>
                {
                    var connectionString =

RoleEnvironment.GetConfigurationSettingValue(configName);
                    configSettingPublisher(connectionString);
                }
            );

            CloudStorageAccount account =
CloudStorageAccount.FromConfigurationSetting("DataConnectionString
");

            //создание таблицы Windows Azure Table
            CloudTableClient _tc = null;
            _tc = account.CreateCloudTableClient();
            _tc.CreateTableIfNotExist("Relational");
            Context context = new Context(account.TableEndpoint,
account.Credentials);
            //определение параметров подключения к БД, измените строку
подключения
            //соответствующим образом, для соединения с вашим sql -
сервером
            SqlConnection conn = new SqlConnection("Data
Source=saigon\\sqlexpress;
              Initial Catalog=azureexample; Integrated
Security=true;");

            //импорт данных из таблицы Address
            //указываем команду для чтения данных из базы
            SqlCommand cmd = new SqlCommand("SELECT * FROM
Address", conn);
            conn.Open();
            SqlDataReader adr = cmd.ExecuteReader();
            while (adr.Read())
            {
//в параметрах метода AddObject указывает имя таблицы и определяем новую
сущность класс //Address
                context.AddObject("Relational", new Address
```

```
                {
                    PartitionKey = "Address",
                    RowKey = adr["AddresID"].ToString(),
                    country = adr["Country"].ToString(),
                    city = adr["City"].ToString(),
                    street = adr["Street"].ToString(),
                    house =
Convert.ToInt32(adr["House"].ToString())
                });
                context.SaveChanges();
            }
            adr.Close();
//импорт данных из таблицы Firm
            cmd.CommandText = "SELECT * FROM Firm";
            SqlDataReader frm = cmd.ExecuteReader();
            while (frm.Read())
            {
                context.AddObject("Relational", new Firm
                {
                    PartitionKey = "Firm",
                    RowKey = frm["FirmID"].ToString(),
                    nameof = frm["NameOf"].ToString(),
                    telephone = frm["Telephone"].ToString(),
                    email = frm["Email"].ToString(),
                    adresskey =
Convert.ToInt32(frm["AddressKey"].ToString())
                });
                context.SaveChanges();
            }
            frm.Close();
//импорт данных из таблицы Employee
            cmd.CommandText = "SELECT * FROM Employee";
            SqlDataReader emp = cmd.ExecuteReader();
            while (emp.Read())
            {
                context.AddObject("Relational", new Employee
                {
                    PartitionKey = "Employee",
                    RowKey = emp["EmployeeID"].ToString(),
                    firstname = emp["FirstName"].ToString(),
                    lastname = emp["LastName"].ToString(),
                    telephone = emp["Telephone"].ToString(),
                    firmkey =
Convert.ToInt32(emp["FirmKey"].ToString())
                });
                context.SaveChanges();
            }
            emp.Close();

            conn.Close();
```

Кроме того, нам необходимо создать классы - соответствующие сущностям реляционной базы данных, а также класс - контекст.

Класс Address.cs

```
class Address: TableServiceEntity
    {
        public String country { get; set; }
        public String city { get; set; }
        public String street { get; set; }
        public int house { get; set; }
    }
```

Класс Firm.cs

```
class Firm: TableServiceEntity
    {
        public String nameof { get; set; }
        public String telephone { get; set; }
        public String email { get; set; }
        public int adresskey { get; set; }
    }
```

Класс Employee.cs

```
class Employee: TableServiceEntity
    {
        public String firstname { get; set; }
        public String lastname { get; set; }
        public String telephone { get; set;}
        public int firmkey { get; set; }
    }
```

Класс Context.cs

```
class Context: TableServiceContext
    {
            public IQueryable<Address> ContactData
            {
                get
                {
                    return this.CreateQuery<Address>("Address");

                }
            }
            public Context(Uri baseAddress, StorageCredentials
credentials) :
                  base(baseAddress.AbsoluteUri, credentials) { }
        }
```

Запустите приложение и дождитесь конца его выполнения. В обозревателе серверов раскройте вкладку "Хранилище Windows Azure", затем обновите вкладку "Таблицы" и раскройте ее. Как вы можете видеть, появилась таблица "Relational".

Рис. 16.2.

Если вы решите ее просмотреть, то увидите, что данные из реляционной БД успешно перенесены в таблицу.

Рис. 16.3.

ЛАБОРАТОРНАЯ РАБОТА №5

Тема: **Работа с Windows Azure Table**

Цель: Целью данной практической работы является работа с Windows Azure Table: создание таблицы, добавление данных, просмотр данных, редактирование и удаление сущностей таблицы.

В качестве примера рассмотрим простое приложение - справочник контактов, с поддержкой функции добавления, изменения и удаления.

Прежде всего создадим проект "AzureTableExample", необходимый для данной лабораторной работы.

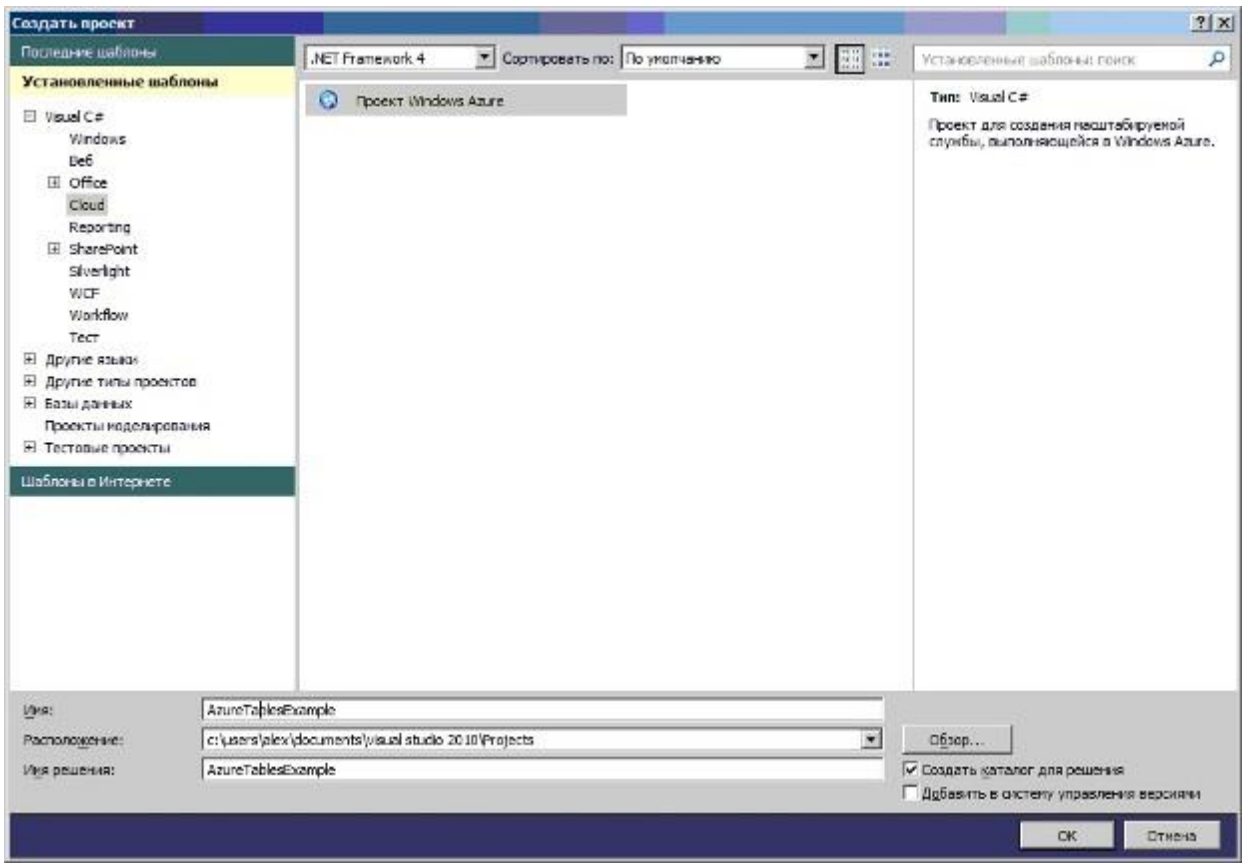


Рис. 17.1.

Для демонстраци возможностей приложения, использующего в качестве хранилища данных WindowsAzure Table нам понадобится только веб - роль, добавьте ее решению.

В свойствах веб - роли определим строку подключения к эмулятору хранилища Azure.

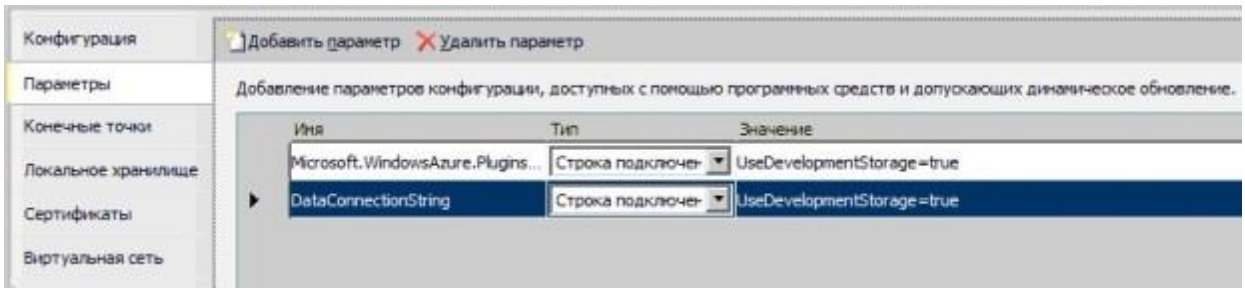


Рис. 17.2.

Создание классов будущего приложения

Класс - Контакт.

Добавим следующие ссылки:

```
using Microsoft.WindowsAzure.StorageClient;
```

Контакт будет содержать следующую информацию:

Имя
Фамилию
Контактный телефон
Адрес электронной почты

Добавим C# класс нашей рабочей роли - Contact.cs и зададим необходимые свойства, унаследовав его от `Microsoft.WindowsAzure.StorageClient.TableServiceEntity` :

```
class Contact :
Microsoft.WindowsAzure.StorageClient.TableServiceEntity
    {

    }
```

Определим требуемые свойства для класса контактов. И в конечном итоге, наш класс будет выглядеть следующим образом:

```
class Contact :
Microsoft.WindowsAzure.StorageClient.TableServiceEntity
    {
        public String FirstName { get; set; }
        public String LastName { get; set; }
        public String TelNumber { get; set; }
        public String Email { get; set; }
    }
```

Класс - контекст для доступа ContactContext.cs

Нам понадобятся следующие ссылки:

```
using Microsoft.WindowsAzure;
using Microsoft.WindowsAzure.StorageClient;
using System.Data.Services.Client;
```

Создадим соответствующий C# - класс, унаследуем его от `TableServiceContext` и определим конструктор:

```
class ContactContext : TableServiceContext
    {
        public ContactContext (Uri baseAddress, StorageCredentials
credentials): base(baseAddress.AbsoluteUri, credentials) {}
    }
```

Добавим свойство для возвращения `data service` – запроса для таблицы Contacts.

```
public IQueryable<Contact> ContactData
        {
            get
                {
                   return this.CreateQuery<Contact>("Contacts");
```

```
                }
          }
```

Последним шагом по созданию класса - контекста является создание метода для добавления строки в таблицу. В итоге, класс должен выглядеть следующим образом 5.1:

```
class ContactContext : TableServiceContext
    {

      public IQueryable<Contact> ContactData
        {
            get
                {
                   return this.CreateQuery<Contact>("Contacts");

                }
         }

        public ContactContext (Uri baseAddress, StorageCredentials
credentials): base(baseAddress.AbsoluteUri, credentials) {}

        public void Add(Contact cntct)
        {

            this.AddObject("Contacts", cntct);

            this.SaveChanges();

         }


    }
```

Листинг 17.1.

Интерфейс

Создадим веб - форму для работы с нашим контакт - листом.

Внешний вид веб формы

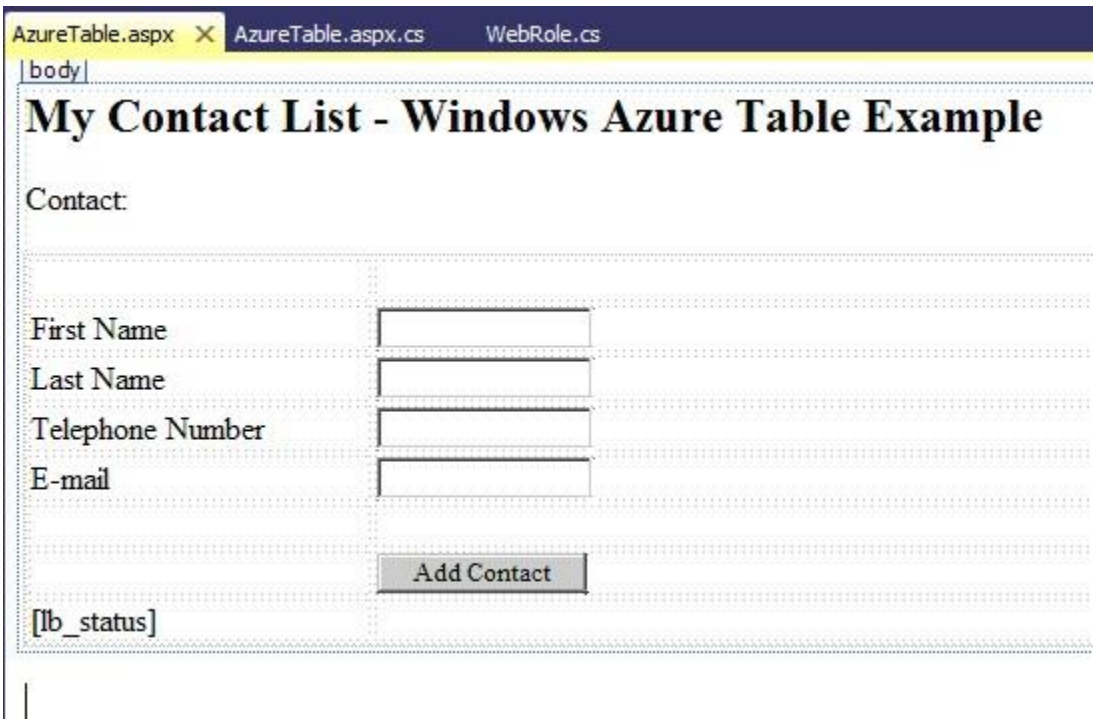


Рис. 17.3.

Задача 1. Добавление данных в таблицу

В обозревателе решений найдите файл Global.asax.cs. В метод Application_Start необходимо добавить код создания таблицы, в случае если она не была создана до этого:

```
void Application_Start(object sender, EventArgs e)
        {

            CloudStorageAccount.SetConfigurationSettingPublisher(
                (configName, configSettingPublisher) =>
                {
                    var connectionString =

RoleEnvironment.GetConfigurationSettingValue(configName);
                    configSettingPublisher(connectionString);
                }
            );

            var account =
CloudStorageAccount.FromConfigurationSetting("DataConnectionString
");

//создание таблицы Windows Azure Table
            CloudTableClient _tc = null;
            _tc = account.CreateCloudTableClient();
            _tc.CreateTableIfNotExist("Contacts");

        }
```

Далее в обработчике события нажатия кнопки "AddContact" необходимо создать

аккаунт для подключения к хранилищу, используя параметр строки подключения "DataConnection String" . После этого, для непосредственного добавления данных в таблицу нам понадобится экземпляр класса - контекса. Вызвав метод "Add" данного класса (листинг 17.1) мы добавим строку в таблицу, передав в качестве параметра экземпляр класса "Contact" . Обратите внимание, в классе Contact мы не описывали необходимые параметры для добавления строки в таблицу (ключ секции и ключ строки), класс унаследовал их от родителя. В качестве значения ключа секции зададим произвольную строку, в качестве ключа строки будем использовать сочетание "Last Name + First Name" .

```
protected void btn_add_Click(object sender, EventArgs e)
        {
            var statusMessage = String.Empty;
            try
            {
                var account =
CloudStorageAccount.FromConfigurationSetting("DataConnectionString
");
                var context = new
ContactContext(account.TableEndpoint, account.Credentials);

                context.Add(new Contact { PartitionKey =
"MyContacts",
                        RowKey = this.tb_lastname.Text + " " +
this.tb_firstname.Text,
                           FirstName = tb_firstname.Text, LastName =
tb_lastname.Text,
                             TelNumber = tb_telnum.Text, Email =
tb_email.Text });
            }
            catch (DataServiceRequestException ex)
            {
                statusMessage = "Unable to connect to the table
storage server. Please check that the service is running.<br>"
                                    + ex.Message;
            }
            lb_status.Text = statusMessage;
        }
```

Запустив отладку приложения, введем необходимые данные в появившуюся веб - форму и нажмем кнопку "Add Contact".

asp - код веб - формы данной задачи:

```
<%@ Page Language="C#"
 AutoEventWireup="true" CodeBehind="AzureTable.aspx.cs"
Inherits="WebRole1.AzureTable" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
```

```
    <title></title>
    <style type="text/css">
        .style1
        {
            width: 169px;
        }
    </style>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:Label ID="NameOfExample" runat="server" Font-
Bold="True"
            Font-Italic="False" Font-Size="Larger" Font-
Strikeout="False"
            Font-Underline="False" Text="My Contact List - Windows
Azure Table Example"></asp:Label>
        <br />
        <br />
        <asp:Label ID="ContactLabel" runat="server"
Text="Contact:"></asp:Label>
        <br />
        <table style="width:100%;">
            <tr>
                <td class="style1">
                     </td>
                <td>
                     </td>
            </tr>
            <tr>
                <td class="style1">
                    <asp:Label ID="lb_firstname" runat="server"
Text="First Name"></asp:Label>
                </td>
                <td>
                    <asp:TextBox ID="tb_firstname"
runat="server"></asp:TextBox>
                </td>
            </tr>
            <tr>
                <td class="style1">
                    <asp:Label ID="lb_lastname" runat="server"
Text="Last Name"></asp:Label>
                </td>
                <td>
                    <asp:TextBox ID="tb_lastname"
runat="server"></asp:TextBox>
                </td>
            </tr>
            <tr>
                <td class="style1">
                    <asp:Label ID="lb_telnum" runat="server"
Text="Telephone Number"></asp:Label>
```

```
                </td>
                <td>
                    <asp:TextBox ID="tb_telnum"
runat="server"></asp:TextBox>
                </td>
            </tr>
            <tr>
                <td class="style1">
                    <asp:Label ID="lb_email" runat="server"
Text="E-mail"></asp:Label>
                </td>
                <td>
                    <asp:TextBox ID="tb_email"
runat="server"></asp:TextBox>
                </td>
            </tr>
            <tr>
                <td class="style1">
                     </td>
                <td>
                     </td>
            </tr>
            <tr>
                <td class="style1">
                     </td>
                <td>
                    <asp:Button ID="btn_add" runat="server"
Height="20px" Text="Add Contact" />
                </td>
            </tr>
            <tr>
                <td class="style1">
                    <asp:Label ID="lb_status"
runat="server"></asp:Label>
                </td>
                <td>
                     </td>
            </tr>
        </table>

    </div>
    </form>
</body>
</html>
```

Задача 2. Просмотр данных

Форма создана. Классы сформированы. Осталось понять добавилось ли что-либо в наше хранилище. Открыв обозреватель серверов и раскрыв последовательно вкладки "Хранилище Windows Azure", "Разработка" и "Таблицы", мы увидем созданную таблицу "Contacts".

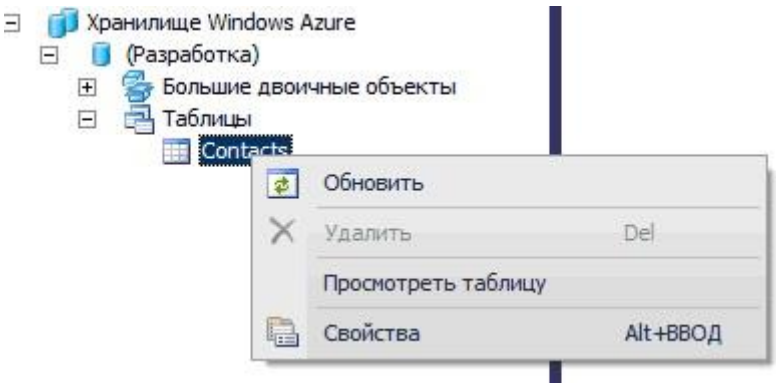


Рис. 17.4.

Щелкнув правой кнопкой мыши на нашей таблице и выбрав "Просмотреть таблицу", получив следующее:

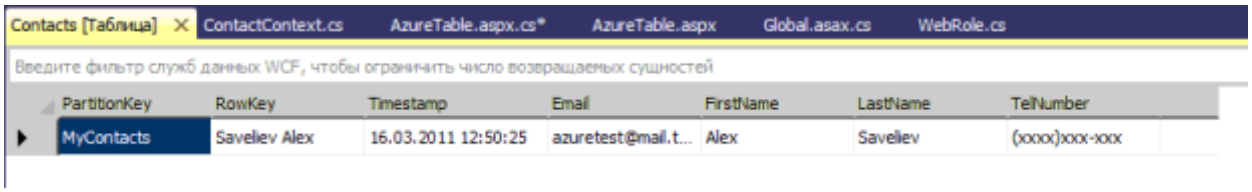


Рис. 17.5.

Как видно, данные успешно добавлены.

Но нас интересует отображение списка контактов в рамках нашего приложения.

Для этого добавим на веб - форму элемент управления GridView с `ID = contactGV` , который мы будем использовать для отображения текущих данных таблицы.

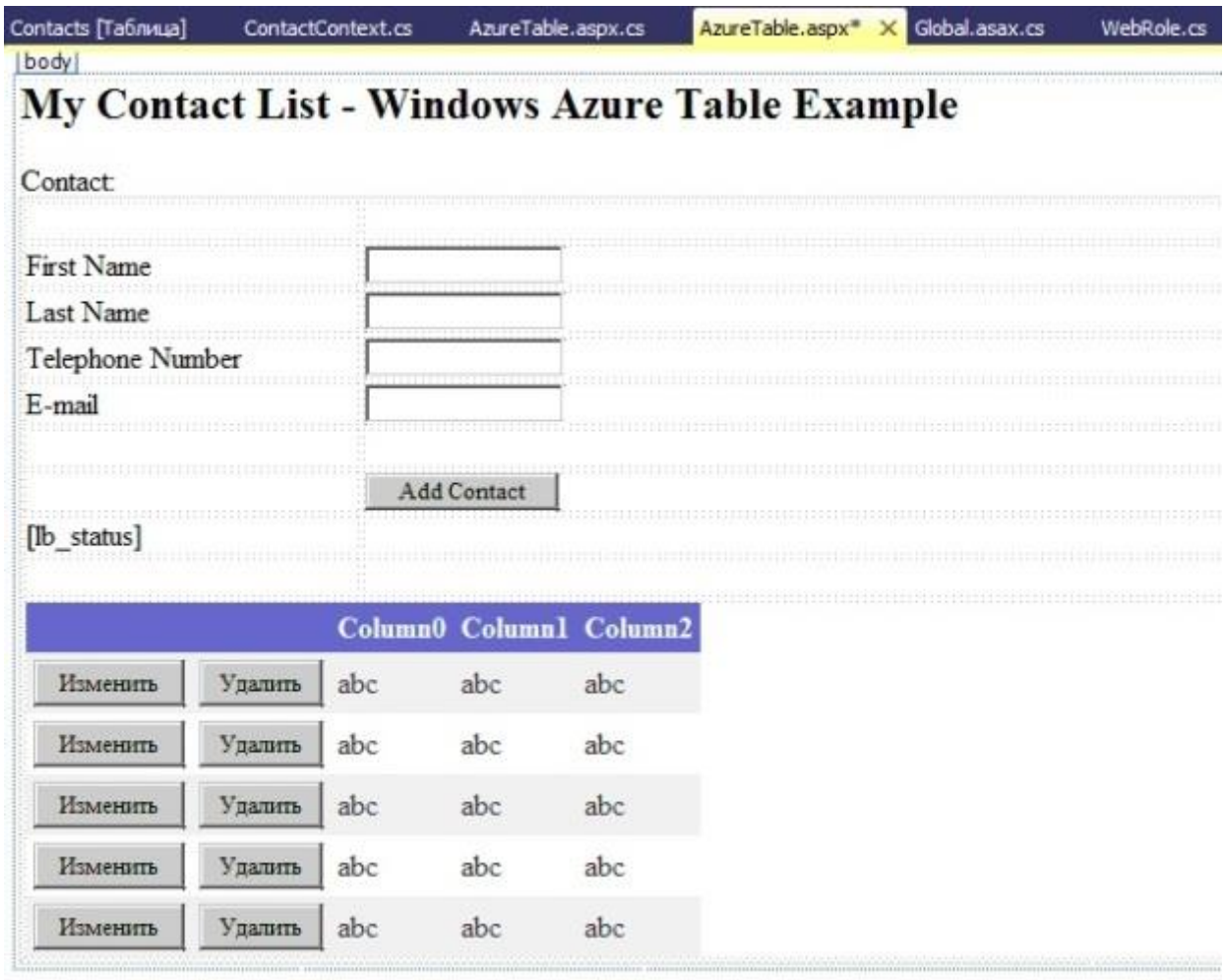


Рис. 17.6.

asp - код для GridView:

```
<asp:GridView ID="contactGV" runat="server" CellPadding="4"
ForeColor="#333333"
                          GridLines="None"
onrowdeleting="contactGV_RowDeleting"

onselectedindexchanged="contactGV_SelectedIndexChanged">
                          <AlternatingRowStyle BackColor="White" />
                          <Columns>
                              <asp:CommandField ButtonType="Button"
SelectText="Изменить"
                                  ShowSelectButton="True" />
                              <asp:CommandField ButtonType="Button"
ShowDeleteButton="True" />
                          </Columns>
                          <EditRowStyle BackColor="#2461BF" />
                          <FooterStyle BackColor="#507CD1" Font-
Bold="True" ForeColor="White" />
                          <HeaderStyle BackColor="#507CD1" Font-
Bold="True" ForeColor="White" />
                          <PagerStyle BackColor="#2461BF"
```

```
ForeColor="White" HorizontalAlign="Center" />
                        <RowStyle BackColor="#EFF3FB" />
                        <SelectedRowStyle BackColor="#D1DDF1"
Font-Bold="True" ForeColor="#333333" />
                        <SortedAscendingCellStyle
BackColor="#F5F7FB" />
                        <SortedAscendingHeaderStyle
BackColor="#6D95E1" />
                        <SortedDescendingCellStyle
BackColor="#E9EBEF" />
                        <SortedDescendingHeaderStyle
BackColor="#4870BE" />
                    </asp:GridView>
```

Определим переменные учетной записи и контекста, до метода **Page_Load**:

```
private CloudStorageAccount account = null;
private ContactContext context = null;
```

В метод **Page_Load** нашей страницы добавим следующий код, для привязки GridView к источнику данных:

```
account=
CloudStorageAccount.FromConfigurationSetting("DataConnectionString
");
            context = new ContactContext(account.TableEndpoint,
account.Credentials);
            contactGV.DataSource = context.ContactData;
            contactGV.DataBind();
```

Листинг 17.2.

Также в методе `btn_add_Click` добавим следующую строку:

```
contactGV.DataBind();
```

Запустив приложение, получим следующее:

Рис. 17.7.

Как видно, при загрузке страницы мы сразу получаем и текущий список контактов. Если добавить новый контакт, то (см. 5.2) список дополнится еще одной записью.

Рис. 17.8.

Задача 3. Редактирование и удаление сущностей

По своей сути задачи редактирование и удаления сущностей конкретной таблицы весьма проста. Достаточно вызвать соответствующие методы класса - контекста. Однако, в качестве параметров методов выступают изменяемые сущности. Таким образом, все сводится к выделению конкретной сущности из таблицы по ее ключам секции и строки, которые, напомним, являются частями уникального ключа сущности.

Поскольку нашей целью является демонстрация работы с Windows Azure Table, мы пойдем по самому простому пути.

Для начала добавим на форму кнопку btn_change, невидимую при загрузке страницы.

Данные между методами формы будем передавать при помощи сессий.

Первая и главная задача сводится к тому, чтобы определить ключи строки и секции редактируемой сущности. Учитывая, что структура таблицы и названия параметров могут быть произвольны, за одним исключением - как раз параметров `Partition Key` и `Row Key` , добавим в метод **Page_Load** следующее:

```
int i = 0;

            foreach (TableCell cell in contactGV.HeaderRow.Cells)
            {

                if (cell.Text == "PartitionKey")
{ Session["pkindex"] = i; }
                if (cell.Text == "RowKey") { Session["rkindex"] =
i; }
                i++;
            }
```

Теперь, вне зависимости от структуры таблицы, сессии `pkindex` и `rkindex` будут содержать номера столбцов `contactGV` , в которых находятся параметры ключей секции и строки.

Полностью метод **Page_Load** для данного задания должен быть следующим:

```
protected void Page_Load(object sender, EventArgs e)
        {
            btn_change.Visible = false;

            account =
CloudStorageAccount.FromConfigurationSetting("DataConnectionString
");
            context = new ContactContext(account.TableEndpoint,
account.Credentials);
            contactGV.DataSource = context.ContactData;
            contactGV.DataBind();

            int i = 0;

            foreach (TableCell cell in contactGV.HeaderRow.Cells)
            {

                if (cell.Text == "PartitionKey")
{ Session["pkindex"] = i; }
                if (cell.Text == "RowKey") { Session["rkindex"] =
i; }
                i++;
            }
        }
```

Также добавим в класс ContactContext методы для обновления и удаления сущностей - Update и Delete соответственно:

```
public void Delete(Contact cnt)
        {
            this.DeleteObject(cnt);
            this.SaveChanges();
        }

        public void Update(Contact cnt)
        {
            this.UpdateObject(cnt);
            this.SaveChanges();
        }
```

Удаление строки

Добавим метод, обрабатывающий нажатие кнопки "Удалить" элемента contactGV.

```
protected void contactGV_RowDeleting(object sender,
GridViewDeleteEventArgs e)
        {
```

```
            GridView g = (GridView)sender;
            try
            {
                Contact c = (from contact in
context.CreateQuery<Contact>("Contacts")
                             where contact.PartitionKey ==
g.Rows[e.RowIndex].

Cells[Convert.ToInt32(Session["pkindex"].ToString())].Text
                             && contact.RowKey ==
g.Rows[e.RowIndex].Cells[Convert.ToInt32

(Session["rkindex"].ToString())].Text
                             select contact).FirstOrDefault();

                context.Delete(c);
            }
            catch(DataServiceRequestException ex)
            {
                lb_status.Text = ex.Message;
            }
         g.DataBind();

        }
```

Обратим ваше внимание на то, что сущность для удаления мы получаем при помощи linq - запроса, указывая значения параметров `PartitionKey` и `RowKey` . Значения же параметров мы получаем из `contactGV` , указывая значения соответствующих ячеек.

Реализация удаления сущности на этом закончена.

Редактирование сущности

Для начала напишем обработчик события изменения индекса выбранной строки `contactGV` , инициируемое нажатием кнопки `"Изменить"` . При нажатии этой кнопки должен становиться видимым элемент управления btn_change, а соответствующие текстовые поля заполняться значениями параметров редактируемой строки.

```
protected void contactGV_SelectedIndexChanged(object sender,
EventArgs e)
        {
            GridView g = (GridView)sender;
            int index = g.SelectedIndex;

            Contact c = (from contact in
context.CreateQuery<Contact>("Contacts")
                         where contact.PartitionKey ==
g.Rows[index].

Cells[Convert.ToInt32(Session["pkindex"].ToString())].Text
                         && contact.RowKey == g.Rows[index].
```

```
Cells[Convert.ToInt32(Session["rkindex"].ToString())].Text
                        select contact).FirstOrDefault();
            tb_firstname.Text = c.FirstName;
            tb_lastname.Text = c.LastName;
            tb_email.Text = c.Email;
            tb_telnum.Text = c.TelNumber;

            Session["index"] = index;

            btn_change.Visible = true;
        }
```

Здесь стоит обратить внимание разве что на формирование еще одной сессии, в которой мы будем хранить номер редактируемой строки в contactGV.

Вот, что должно получиться при нажатии кнопки "Изменить" напротив второй строки списка контактов:

Рис. 17.9.

Как видим, значения текстовых полей стали соответствовать значениям параметров редактируемой строки.

Осталось только написать метод обрабатывающий событие нажатия кнопки btn_change.

Он будет выглядеть следующим образом:

```
protected void btn_change_Click(object sender, EventArgs e)
        {

            int index =
Convert.ToInt32(Session["index"].ToString());
            try
            {
            Contact c = (from contact in
context.CreateQuery<Contact>("Contacts")
                         where contact.PartitionKey ==
```

```
contactGV.Rows[index].

Cells[Convert.ToInt32(Session["pkindex"].ToString())].Text
                        && contact.RowKey ==
contactGV.Rows[index].

Cells[Convert.ToInt32(Session["rkindex"].ToString())].Text
                        select contact).FirstOrDefault();

            c.FirstName = tb_firstname.Text;
            c.LastName = tb_lastname.Text;
            c.Email = tb_email.Text;
            c.TelNumber = tb_telnum.Text;


                context.Update(c);

            }
            catch (DataServiceRequestException a)
            {
                lb_status.Text = a.Message;
            }
            contactGV.DataBind();

        }
```

Номер изменяемой строки мы получаем из сессии `"index"` , остальное уже не должно вызывать вопросов.

Запустите приложение еще раз и измените произвольным образом любой из параметров, либо насколько из них какой - либо строки. Мы изменили `Email` на `tableentitychange@mail.test` . Нажмите кнопку "Change".

Мы получили следующее:

Рис. 17.10.

Еще раз обратим ваше внимание на то, что демонстрируемое приложение является

не более, чем примером способов работы с Windows Azure Storage, поэтому мы пренебрегли обработкой исключительных ситуаций и проверкой правильности и целостности введенных данных. Оставляем это на ваше усмотрение.

В случае, если выполнение задания вызвало сложности и затруднения, в приложениях к данной практической работе вы найдете итоговый программный код в том виде, в котором он необходим для Задания№3.

Список вспомогательных материалов

Работа с Windows Azure Table

http://msdn.microsoft.com/ru-ru/magazine/gg309178.aspx
http://msdn.microsoft.com/en-us/wazplatformtrainingcourse_exploringwindowsazurestoragevs2010_topic2#_Toc282791662
http://blogs.msdn.com/b/morebits/archive/2010/12/26/building-windows-azure-service-part3-table-storage.aspx

Приложение А. ASP - код веб - формы (AzureTable.aspx)

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeBehind="AzureTable.aspx.cs" Inherits="WebRole1.AzureTable" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
    <style type="text/css">
        .style1
        {
        }
        .style2
        {
            width: 149px;
        }
    </style>
</head>
<body>
    <form id="form1" runat="server">
    <div>

        <asp:Label ID="NameOfExample" runat="server" Font-
Bold="True"
            Font-Italic="False" Font-Size="Larger" Font-
Strikeout="False"
            Font-Underline="False" Text="My Contact List - Windows
Azure Table Example"></asp:Label>
        <br />
        <br />
        <asp:Label ID="ContactLabel" runat="server"
Text="Contact:"></asp:Label>
```

```
        <br />
        <table style="width:100%;">
            <tr>
                <td class="style2">
                     </td>
                <td>
                     </td>
            </tr>
            <tr>
                <td class="style2">
                    <asp:Label ID="lb_firstname" runat="server"
Text="First Name"></asp:Label>
                </td>
                <td>
                    <asp:TextBox ID="tb_firstname"
runat="server"></asp:TextBox>
                </td>
            </tr>
            <tr>
                <td class="style2">
                    <asp:Label ID="lb_lastname" runat="server"
Text="Last Name"></asp:Label>
                </td>
                <td>
                    <asp:TextBox ID="tb_lastname"
runat="server"></asp:TextBox>
                </td>
            </tr>
            <tr>
                <td class="style2">
                    <asp:Label ID="lb_telnum" runat="server"
Text="Telephone Number"></asp:Label>
                </td>
                <td>
                    <asp:TextBox ID="tb_telnum"
runat="server"></asp:TextBox>
                </td>
            </tr>
            <tr>
                <td class="style2">
                    <asp:Label ID="lb_email" runat="server"
Text="E-mail"></asp:Label>
                </td>
                <td>
                    <asp:TextBox ID="tb_email"
runat="server"></asp:TextBox>
                </td>
            </tr>
            <tr>
                <td class="style2">
                     </td>
                <td>
                     </td>
```

```
            </tr>
            <tr>
                <td class="style2">
                    <asp:Button ID="btn_change" runat="server"
onclick="btn_change_Click"
                        Text="Change" Width="105px" />
                </td>
                <td>
                    <asp:Button ID="btn_add" runat="server"
Height="20px" onclick="btn_add_Click"
                        Text="Add Contact" />
                </td>
            </tr>
            <tr>
                <td class="style2">
                    <asp:Label ID="lb_status"
runat="server"></asp:Label>
                </td>
                <td>
                     </td>
            </tr>
            <tr>
                <td class="style2">
                     </td>
                <td>
                     </td>
            </tr>
            <tr>
                <td class="style1" colspan="2">
                    <asp:GridView ID="contactGV" runat="server"
CellPadding="4" ForeColor="#333333"
                        GridLines="None"
onrowdeleting="contactGV_RowDeleting"

onselectedindexchanged="contactGV_SelectedIndexChanged">
                        <AlternatingRowStyle BackColor="White" />
                        <Columns>
                            <asp:CommandField ButtonType="Button"
SelectText="Изменить"
                                ShowSelectButton="True" />
                            <asp:CommandField ButtonType="Button"
ShowDeleteButton="True" />
                        </Columns>
                        <EditRowStyle BackColor="#2461BF" />
                        <FooterStyle BackColor="#507CD1" Font-
Bold="True" ForeColor="White" />
                        <HeaderStyle BackColor="#507CD1" Font-
Bold="True" ForeColor="White" />
                        <PagerStyle BackColor="#2461BF"
ForeColor="White" HorizontalAlign="Center" />
                        <RowStyle BackColor="#EFF3FB" />
                        <SelectedRowStyle BackColor="#D1DDF1"
Font-Bold="True" ForeColor="#333333" />
```

```
                        <SortedAscendingCellStyle
BackColor="#F5F7FB" />
                        <SortedAscendingHeaderStyle
BackColor="#6D95E1" />
                        <SortedDescendingCellStyle
BackColor="#E9EBEF" />
                        <SortedDescendingHeaderStyle
BackColor="#4870BE" />
                    </asp:GridView></td>
            </tr>
        </table>

    </div>
    </form>
</body>
</html>
```

Приложение Б. Класс Contact.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace WebRole1
{
    class Contact :
Microsoft.WindowsAzure.StorageClient.TableServiceEntity
    {
        public String FirstName { get; set; }
        public String LastName { get; set; }
        public String TelNumber { get; set; }
        public String Email { get; set; }
    }
}
```

Приложение В. Класс ContactContext.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Microsoft.WindowsAzure;
using Microsoft.WindowsAzure.StorageClient;
using System.Data.Services.Client;

namespace WebRole1
{
    class ContactContext : TableServiceContext
    {

        public IQueryable<Contact> ContactData
        {
            get
            {
```

```
                return this.CreateQuery<Contact>("Contacts");

            }
        }

        public ContactContext(Uri baseAddress, StorageCredentials
credentials) : base(baseAddress.AbsoluteUri, credentials) { }

        public void Add(Contact cnt)
        {
            this.AddObject("Contacts", cnt);
            this.SaveChanges();
        }

        public void Delete(Contact cnt)
        {
            this.DeleteObject(cnt);
            this.SaveChanges();
        }

        public void Update(Contact cnt)
        {
            this.UpdateObject(cnt);
            this.SaveChanges();
        }
    }
}
```

Приложение Г. Global.asax.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Security;
using System.Web.SessionState;
using Microsoft.WindowsAzure;
using Microsoft.WindowsAzure.StorageClient;
using System.Data.Services.Client;
using Microsoft.WindowsAzure.ServiceRuntime;

namespace WebRole1
{
    public class Global : System.Web.HttpApplication
    {

        void Application_Start(object sender, EventArgs e)
        {

            CloudStorageAccount.SetConfigurationSettingPublisher(
                (configName, configSettingPublisher) =>
                {
                    var connectionString =
```

```
RoleEnvironment.GetConfigurationSettingValue(configName);
                    configSettingPublisher(connectionString);
                }
            );

            var account =
CloudStorageAccount.FromConfigurationSetting("DataConnectionString
");

            CloudTableClient _tc = null;
            _tc = account.CreateCloudTableClient();
            _tc.CreateTableIfNotExist("Contacts");


        }

        void Application_End(object sender, EventArgs e)
        {

        }

        void Application_Error(object sender, EventArgs e)
        {

        }

        void Session_Start(object sender, EventArgs e)
        {

        }

        void Session_End(object sender, EventArgs e)
        {

        }

    }
}
```

Приложение Д. AzureTable.aspx.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Data.Services.Client;
using Microsoft.WindowsAzure;

namespace WebRole1
{
    public partial class AzureTable : System.Web.UI.Page
    {
```

```
        private CloudStorageAccount account = null;
        private ContactContext context = null;

        protected void Page_Load(object sender, EventArgs e)
        {
            btn_change.Visible = false;

            account =
CloudStorageAccount.FromConfigurationSetting("DataConnectionString
");
            context = new ContactContext(account.TableEndpoint,
account.Credentials);
            contactGV.DataSource = context.ContactData;
            contactGV.DataBind();

            int i = 0;

            foreach (TableCell cell in contactGV.HeaderRow.Cells)
            {

                if (cell.Text == "PartitionKey")
{ Session["pkindex"] = i; }
                if (cell.Text == "RowKey") { Session["rkindex"] =
i; }
                i++;
            }
        }

        protected void btn_add_Click(object sender, EventArgs e)
        {
            var statusMessage = String.Empty;
            try
            {

                context.Add(new Contact { PartitionKey =
"MyContacts",
                 RowKey = this.tb_lastname.Text + " " +
this.tb_firstname.Text,
                  FirstName = tb_firstname.Text, LastName =
tb_lastname.Text,
                    TelNumber = tb_telnum.Text, Email =
tb_email.Text });

            }
            catch (DataServiceRequestException ex)
            {
                statusMessage = "Unable to connect to the table
storage server. Please check that the service is running.<br>"
                                + ex.Message;
            }
            lb_status.Text = statusMessage;
            contactGV.DataBind();
        }
```

```
        protected void contactGV_RowDeleting(object sender,
GridViewDeleteEventArgs e)
        {

            GridView g = (GridView)sender;
            try
            {
                Contact c = (from contact in
context.CreateQuery<Contact>("Contacts")
                             where contact.PartitionKey ==
g.Rows[e.RowIndex].

Cells[Convert.ToInt32(Session["pkindex"].ToString())].Text
                             && contact.RowKey ==
g.Rows[e.RowIndex].
                              .Cells[Convert.ToInt32(Session["rkin
dex"].ToString())].Text
                             select contact).FirstOrDefault();

                context.Delete(c);
            }
            catch(DataServiceRequestException ex)
            {
                lb_status.Text = ex.Message;
            }
          g.DataBind();

        }

        protected void contactGV_SelectedIndexChanged(object
sender, EventArgs e)
        {
            GridView g = (GridView)sender;
            int index = g.SelectedIndex;

            Contact c = (from contact in
context.CreateQuery<Contact>("Contacts")
                         where contact.PartitionKey ==
g.Rows[index].Cells[Convert.ToInt32(Session["pkindex"].ToString())
].Text
                         && contact.RowKey ==
g.Rows[index].Cells[Convert.ToInt32(Session["rkindex"].ToString())
].Text
                         select contact).FirstOrDefault();
            tb_firstname.Text = c.FirstName;
            tb_lastname.Text = c.LastName;
            tb_email.Text = c.Email;
            tb_telnum.Text = c.TelNumber;

            Session["index"] = index;

            btn_change.Visible = true;
```

```
        }

        protected void btn_change_Click(object sender, EventArgs
e)
        {

            int index =
Convert.ToInt32(Session["index"].ToString());
            try
            {
            Contact c = (from contact in
context.CreateQuery<Contact>("Contacts")
                         where contact.PartitionKey ==
contactGV.Rows[index].

Cells[Convert.ToInt32(Session["pkindex"].ToString())].Text
                         && contact.RowKey ==
contactGV.Rows[index].

Cells[Convert.ToInt32(Session["rkindex"].ToString())].Text
                         select contact).FirstOrDefault();

            c.FirstName = tb_firstname.Text;
            c.LastName = tb_lastname.Text;
            c.Email = tb_email.Text;
            c.TelNumber = tb_telnum.Text;


                context.Update(c);

            }
            catch (DataServiceRequestException a)
            {
                lb_status.Text = a.Message;
            }
            contactGV.DataBind();

        }
    }
}
```

ЛАБОРАТОРНАЯ РАБОТА №6

Тема: **Работа с Windows Azure Blob**

Цель: Соединение с хранилищем. Добавление и удаление Blob - объекта.

В качестве примера рассмотрим работу простого веб - приложения для загрузки изображений в хранилище Windows Azure Blob:

подготовка приложения;

загрузка и отображение изображений;
удаление сущностей;
копирование сущностей.

Создайте проект облачной службы и добавьте веб - роль в решение.

В свойствах веб - роли определим строку подключения к эмулятору хранилища Azure.

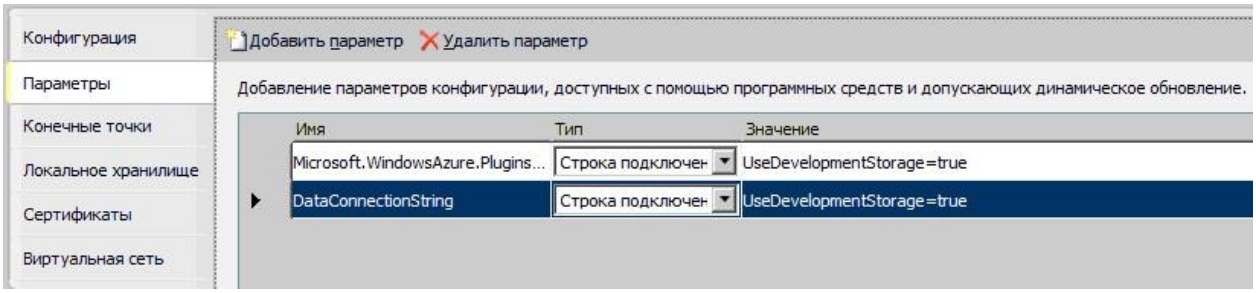


Рис. 20.1.

Также нам понадобится строковый параметр ContainerName, определяющий имя Blob - контейнера для изображений.

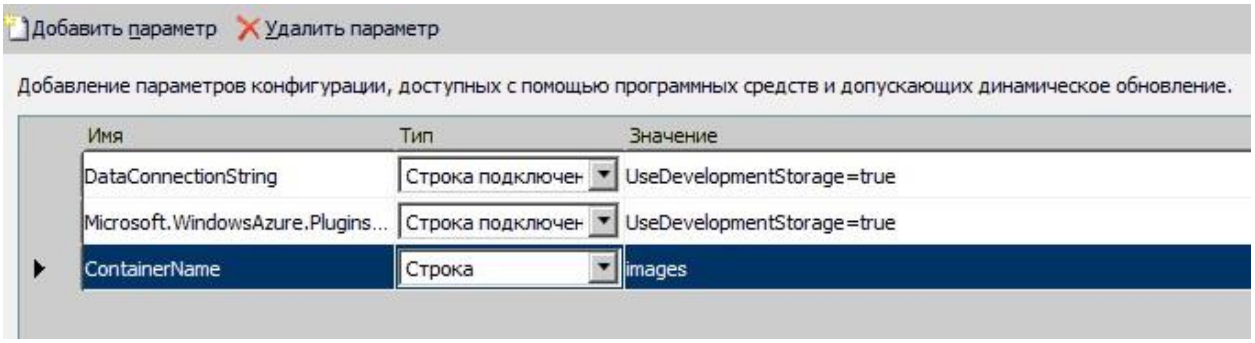


Рис. 20.2.

Задание 1. Подготовка приложени

Создадим веб - форму AzureBlobSamle.aspx и добавим ссылки в C# класс данной страницы

```
using Microsoft.WindowsAzure;
using Microsoft.WindowsAzure.ServiceRuntime;
using Microsoft.WindowsAzure.StorageClient;
using System.Configuration;
```

Далее, поскольку Blob - контейнер должен быть доступным для анонимных пользователей необходимо создать метод, подтверждающий наличие контейнера:

```
private void EnsureContainerExists()
        {
            var container = GetContainer();
            container.CreateIfNotExist();
            var permissions = container.GetPermissions();
```

```
            permissions.PublicAccess =
BlobContainerPublicAccessType.Container;
            container.SetPermissions(permissions);
        }
```

Как видно, в данном методе присутствует ссылка на несуществующий пока `GetConteiner()` , который должен возвращать Blob-контейнер для осуществления операций с Blob-объектами.

```
private CloudBlobContainer GetContainer()
        {
            CloudStorageAccount.SetConfigurationSettingPublisher(
                (configName, configSettingPublisher) =>
                {
                    var connectionString =

RoleEnvironment.GetConfigurationSettingValue(configName);
                    configSettingPublisher(connectionString);
                }
            );

            var account =
CloudStorageAccount.FromConfigurationSetting("DataConnectionString
");
            var client = account.CreateCloudBlobClient();
            return
client.GetContainerReference(RoleEnvironment.GetConfigurationSetti
ngValue("ContainerName"));
        }
```

Интерфейс

Теперь необходимо заняться самой формой. На форме буду присутствовать текстовые поля, задающие параметры изображения и диалог загрузки файла.

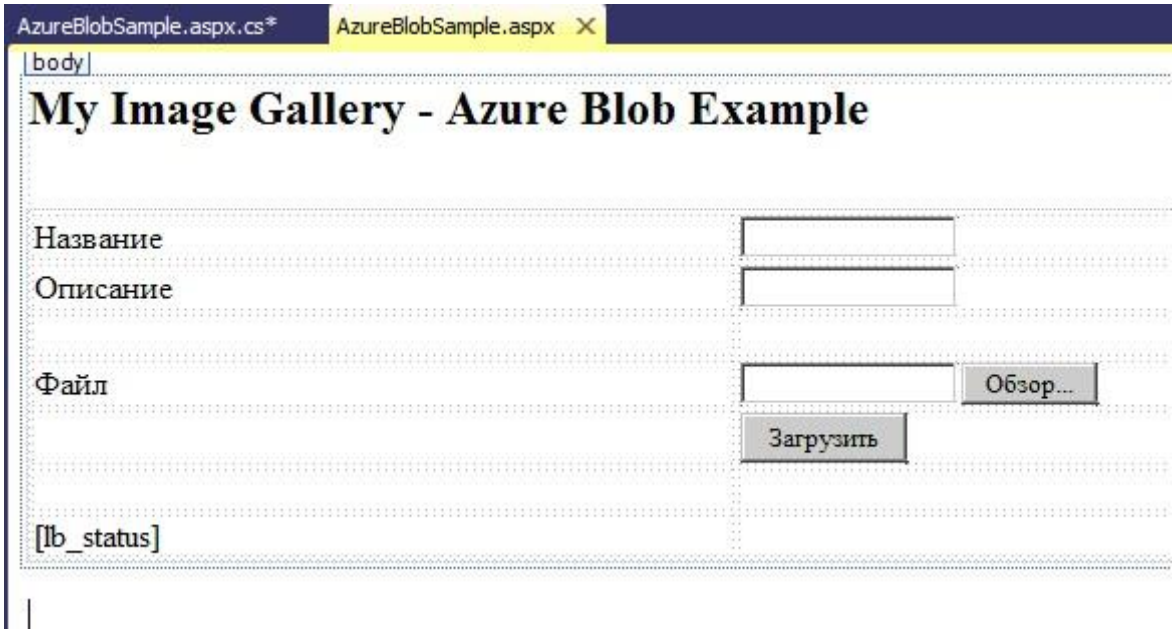


Рис. 20.3.

asp - код веб формы:

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeBehind="AzureBlobSample.aspx.cs"
  Inherits="WebRole1.AzureBlobSample" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
    <div>

        <asp:Label ID="Label1" runat="server" Font-Bold="True"
Font-Size="X-Large"
            Text="My Image Gallery - Azure Blob
Example"></asp:Label>
        <br />
        <br />
        <table style="width:100%;">
            <tr>
                <td>
                    <asp:Label ID="lb_name" runat="server"
Text="Название"></asp:Label>
                </td>
                <td>
                    <asp:TextBox ID="tb_label"
runat="server"></asp:TextBox>
                </td>
            </tr>
            <tr>
                <td>
                    <asp:Label ID="lb_desc" runat="server"
Text="Описание"></asp:Label>
                </td>
                <td>
                    <asp:TextBox ID="tb_desc"
runat="server"></asp:TextBox>
                </td>
            </tr>
            <tr>
                <td>
                     </td>
                <td>
                     </td>
            </tr>
            <tr>
```

```
                <td>
                    <asp:Label ID="lb_file" runat="server"
Text="Файл"></asp:Label>
                </td>
                <td>
                    <asp:FileUpload ID="fu_upload" runat="server"
/>
                </td>
            </tr>
            <tr>
                <td>
                     </td>
                <td>
                    <asp:Button ID="btn_upload" runat="server"
onclick="btn_upload_Click"
                        Text="Загрузить" />
                </td>
            </tr>
            <tr>
                <td>
                     </td>
                <td>
                     </td>
            </tr>
            <tr>
                <td>
                    <asp:Label ID="lb_status"
runat="server"></asp:Label>
                </td>
                <td>
                    <asp:ListView ID="lv_images" runat="server">
            <LayoutTemplate>
                <asp:PlaceHolder ID="itemPlaceholder"
runat="server" />
            </LayoutTemplate>
            <EmptyDataTemplate>
                <h2>No Data Available</h2>
            </EmptyDataTemplate>
            <ItemTemplate>
                <div class="item">
                    <ul style="width:40em;float:left;clear:left" >
                        <asp:Repeater ID="blobMetadata"
runat="server">
                        <ItemTemplate>
                            <li><%# Eval("Name") %><span><%#
Eval("Value") %></span></li>
                        </ItemTemplate>
                        </asp:Repeater>
                        <li>

                        </li>
                    </ul>
                    <img src="<%# Eval("Uri") %>" alt="<%#
```

```
Eval("Uri") %>" style="float:left"/>
                </div>
            </ItemTemplate>
        </asp:ListView>

                    </td>
            </tr>
            <tr>
                <td>

                    <br />
                </td>
                <td>
                     </td>
            </tr>
        </table>

    </div>
    </form>
</body>
</html>
```

Отметим, что элемент управления, при помощи которого мы будем отображать имеющиеся изображения - это ListView с идентификатором lv_images. Для того чтобы сделать возможным повторени вышеуказанного шаблона для каждого элемента списка мы использовали Repeater (подробнее см. в списке вспомогательных материалов)

Теперь в метод Page_Load добавим код, который при первой загрузке нашей страницы будет проверять наличие контейнера и выводить сообщение о возможных ошибках и привязывать lv_images к источнику данных, вызовом метода RefreshGallery:

```
try
            {
                if (!IsPostBack)
                {
                    this.EnsureContainerExists();
                }
                this.RefreshGallery();
            }
            catch (System.Net.WebException we)
            {
                lb_status.Text = "Network error: " + we.Message;
                if (we.Status ==
System.Net.WebExceptionStatus.ConnectFailure)
                {
                    lb_status.Text += "<br />Please check if the
blob service is running at " +

ConfigurationManager.AppSettings["storageEndpoint"];
                }
            }
```

```
            catch (StorageException se)
            {
                Console.WriteLine("Storage service error: " +
se.Message);
            }
```

Метод RefreshGallery - привязывает lv_images к источнику данных(контейнеру бинарных объектов):

```
private void RefreshGallery()
        {
            lv_images.DataSource =
              this.GetContainer().ListBlobs(new
BlobRequestOptions()
              {
                  UseFlatBlobListing = true,
                  BlobListingDetails = BlobListingDetails.All
              });
            lv_images.DataBind();
        }
```

Запустим приложение и убедимся, что оно выполняется верно.

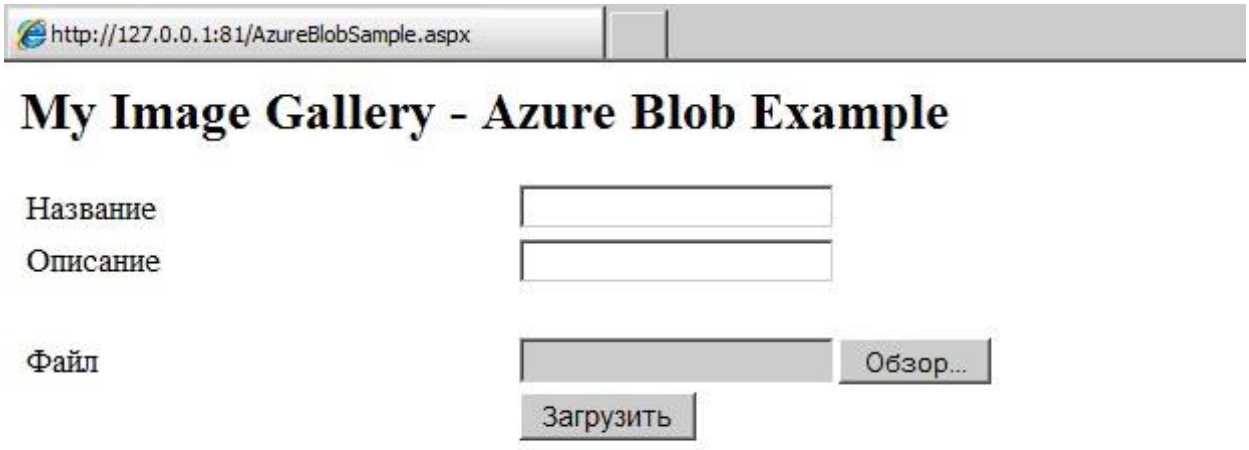


Рис. 20.4.

Задание 2. Загрузка изображений

Необходимо добавить следующую ссылку в AzureBlobSample.cs:

```
using System.Collections.Specialized;
```

Создадим метод, сохраняющий бинарный объект и задающий его метаданные:

```
private void SaveImage(string id, string name, string description,
string tags,
  string fileName, string contentType, byte[] data)
        {
            // Создание BLob - объекта в контейнере
            var blob = this.GetContainer().GetBlobReference(name);
```

```
            blob.Properties.ContentType = contentType;
            // определение метаданных, добавление метаданных blob-объекту
и загрузка данных
            try
            {
                blob.UploadFromStream(fu_upload.FileContent);
                blob.Metadata["Id"] = id;
                blob.Metadata["Filename"] = fileName;
                blob.Metadata["ImageName"] =
String.IsNullOrEmpty(name) ? "unknown" : name;
                blob.Metadata["Description"] =
String.IsNullOrEmpty(description) ? "unknown" : description;
                blob.SetMetadata();
                lv_images.DataBind();
            }
            catch (Exception ex)
            { lb_status.Text = ex.Message; }
        }
        }
```

Остается только написать метод - обрабатывающий событие нажатия кнопки btn_upload:

```
protected void btn_upload_Click(object sender, EventArgs e)
        {
            if (fu_upload.HasFile)
              {
                  lb_status.Text = "Inserted [" +
fu_upload.FileName + "] -
                   Content Type [" +
fu_upload.PostedFile.ContentType + "] -
                   Length [" + fu_upload.PostedFile.ContentLength
+ "]";
                  this.SaveImage(Guid.NewGuid().ToString(),
                    tb_name.Text,
                    tb_desc.Text,
                    fu_upload.FileName,
                    fu_upload.PostedFile.ContentType,
                    fu_upload.FileBytes
                  );
                  RefreshGallery();
                }
                else
                  lb_status.Text = "No image file";
            }
```

Теперь можно проверить функцию загрузки изображения, выбрав его через диалог загрузки файла.

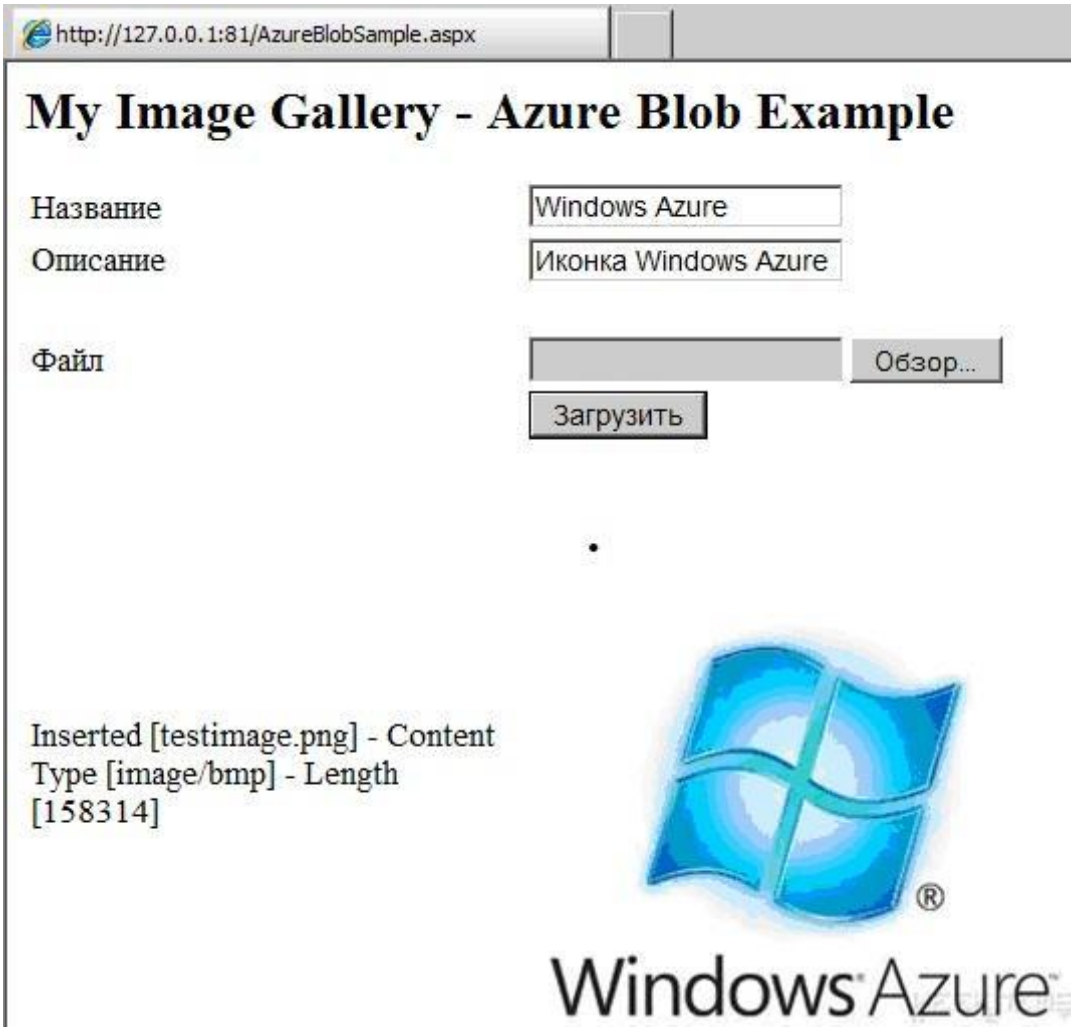


Рис. 20.5.

Надо сказать, что в таком виде информация об изображении малоинформативная. Blob, как мы знаем, может хранить привязанные к нему метаданные. Добавим в наше решение функционал, позволяющий получать и отображать метаданные, ассоциированные с конкретным изображением.

Для начала, необходимо создать метод обрабатывающий событие ItemDataBound элемента управления lv_images:

```
protected void lv_images_ItemDataBound(object sender,
ListViewItemEventArgs e)
        {
            if (e.Item.ItemType == ListViewItemType.DataItem)
            {
                var metadataRepeater =
e.Item.FindControl("blobMetadata") as Repeater;
                var blob = ((ListViewDataItem)(e.Item)).DataItem
as CloudBlob;
                if (blob != null)
                {
                    if (blob.SnapshotTime.HasValue)
                    {
                        var delBtn =
```

```
e.Item.FindControl("deleteBlob") as LinkButton;
                        if (delBtn != null) delBtn.Text = "Delete
Snapshot";
                        var snapshotBtn =
e.Item.FindControl("SnapshotBlob") as LinkButton;
                        if (snapshotBtn != null)
snapshotBtn.Visible = false;
                    }
                    if (metadataRepeater != null)
                    {
                        metadataRepeater.DataSource = from key in
blob.Metadata.AllKeys
                                                      select new
                                                      {
                                                          Name =
key,
                                                          Value =
blob.Metadata[key]
                                                      };
                        metadataRepeater.DataBind();
                    }
                }
            }
        }
```

Теперь запустим наше приложение и получим следующее:

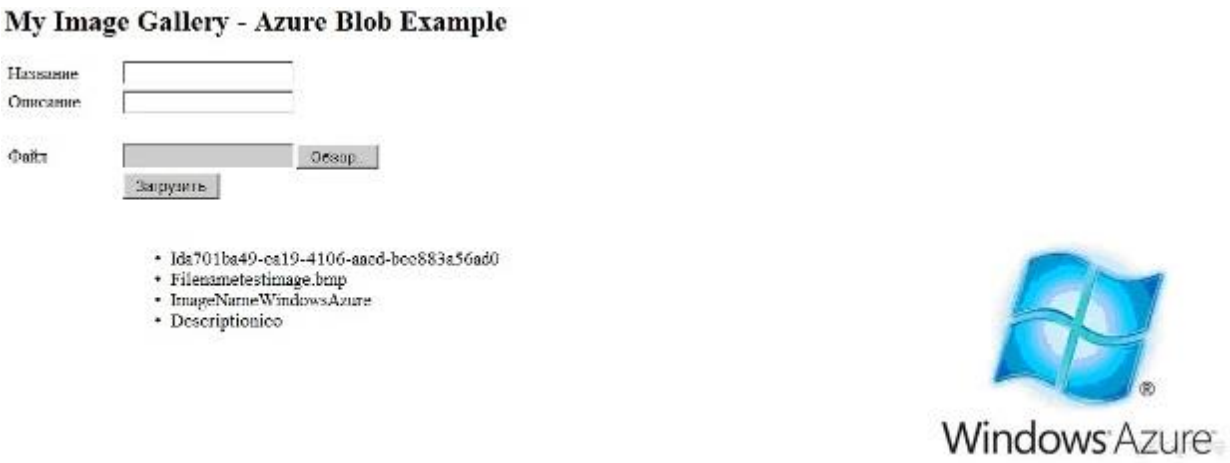


Рис. 20.6.

Слева от изображения появился список метаданных.

Задание 3 Удаление сущностей

Для начала необходимо добавить следующий asp - код для lv_images:

```
...
div class="item">
                    <ul style="width:40em;float:left;clear:left" >
                        <asp:Repeater ID="blobMetadata"
runat="server">
```

```
                        <ItemTemplate>
                            <li><%# Eval("Name") %><span><%#
Eval("Value") %></span></li>
                        </ItemTemplate>
                        </asp:Repeater>
                        <li>
                            <asp:LinkButton ID="deleteBlob"
                                    OnClientClick="return
confirm('Delete image?');"
                                    CommandName="Delete"
                                    CommandArgument='<%#
Eval("Uri")%>'
                                    runat="server" Text="Удалить"
oncommand="OnDeleteImage" />
...
```

Это необходимо для формировании кнопки "Delete" в рамках нашего ListView. Кроме того, этот кол содержит ссылку на метод, выполняющийся при нажатии кнопки.

Метод OnDeleteImage:

```
protected void OnDeleteImage(object sender, CommandEventArgs e)
        {
            try
            {
                if (e.CommandName == "Delete")
                {
                    var blobUri = (string)e.CommandArgument;
                    var blob =
this.GetContainer().GetBlobReference(blobUri);
                    blob.DeleteIfExists();
                }
            }
            catch (StorageClientException se)
            {
                lb_status.Text = "Storage client error: " +
se.Message;
            }
            catch (Exception) { }
            RefreshGallery();
        }
```

Данный метод определяет бинарный объект и удаляет его.

Запустите приложение и протестируйте функцию удаления.

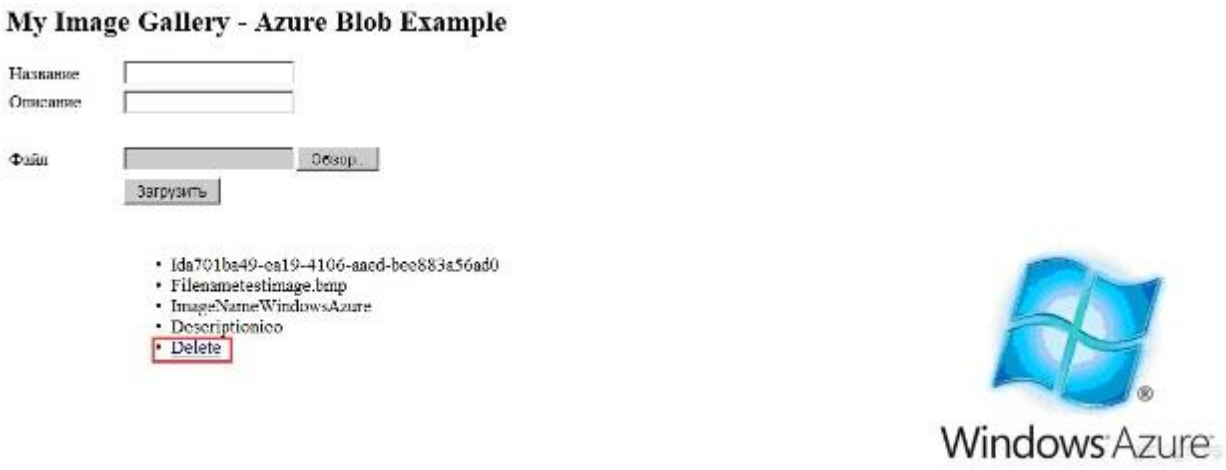


Рис. 20.7.

Задание 4 Копирование сущностей

Копирование бинарных объектов также поддерживается Windows Azure. В рамках данного задания мы рассмотрим этот механизм.

Для формирования кнопки "Копировать" в списке имеющихся изображений, добавим следующий asp - код для `lv_images` :

```
...
                <div class="item">
                    <ul style="width:40em;float:left;clear:left" >
                        <asp:Repeater ID="blobMetadata"
runat="server">
                        <ItemTemplate>
                            <li><%# Eval("Name") %><span><%#
Eval("Value") %></span></li>
                        </ItemTemplate>
                        </asp:Repeater>
                        <li>
...
<asp:LinkButton ID="CopyBlob"
                                        OnClientClick="return
confirm('Copy image?');"
                                        CommandName="Copy"
                                        CommandArgument='<%#
Eval("Uri")%>'
                                        runat="server" Text="Copy"
oncommand="OnCopyImage" />
...
```

Как и в предыдущем задании, создадим метод OnCopyImage:

```
protected void OnCopyImage(object sender, CommandEventArgs e)
        {
            if (e.CommandName == "Copy")
            {
                // Prepare an Id for the copied blob
                var newId = Guid.NewGuid();
```

```
                // получение исходного объекта
                var blobUri = (string)e.CommandArgument;
                var srcBlob =
this.GetContainer().GetBlobReference(blobUri);
                // создание нового бинарного объекта
                var newBlob =
this.GetContainer().GetBlobReference(newId.ToString());
                // копирование содержимого исходного объекта
                newBlob.CopyFromBlob(srcBlob);
                // получаем метаданные для нового объекта
                newBlob.FetchAttributes(new BlobRequestOptions
{ BlobListingDetails = BlobListingDetails.Metadata });
                // изменение метаданных нового объекта, чтобы показать,
что это копия
                newBlob.Metadata["ImageName"] = "Copy of \"" +
newBlob.Metadata["ImageName"]
                   + "\"";
                newBlob.Metadata["Id"] = newId.ToString();
                newBlob.SetMetadata();
                RefreshGallery();
            }
        }
```

Проверим функционал:

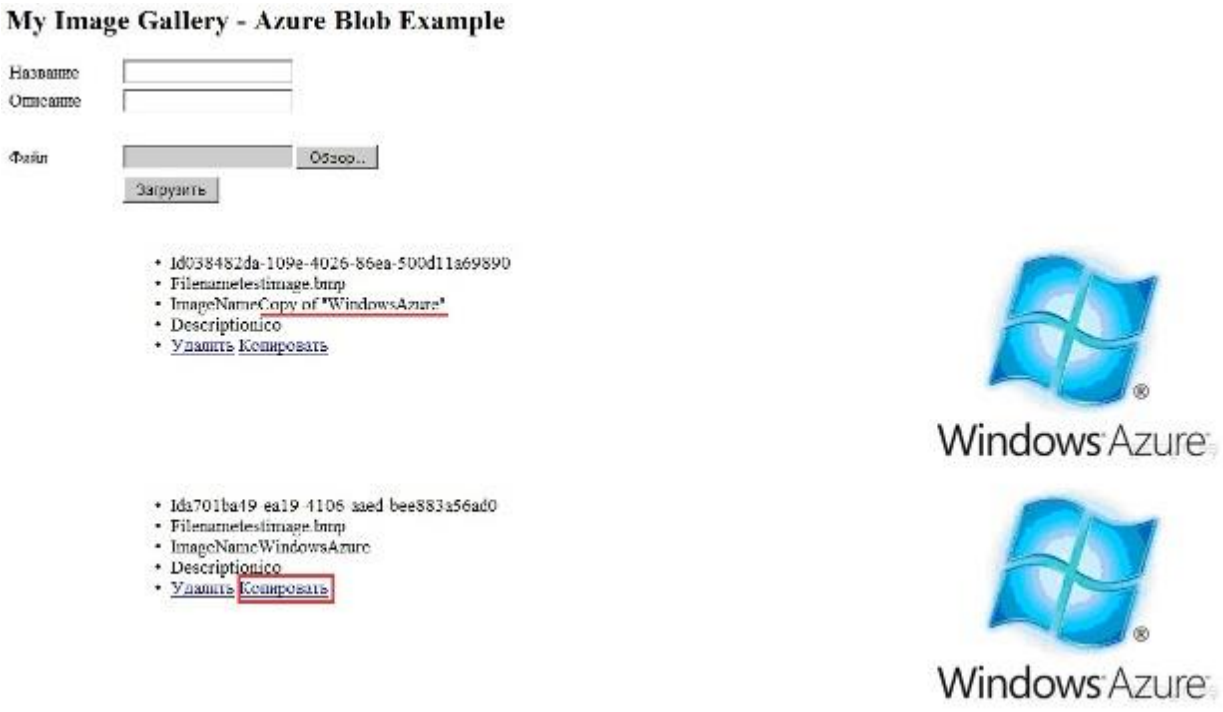


Рис. 20.8.

В случае, если выполнение задания вызвало сложности и затруднения, в приложениях к данной практической работе вы найдете итоговый программный код в том виде, в котором он необходим для последнего задания.

Список вспомогательных материалов

Работа с Windows Azure Blob

http://msdn.microsoft.com/en-us/wazplatformtrainingcourse_exploringwindowsazurestoragevs2010_topic3
http://blogs.msdn.com/b/jnak/archive/2010/01/11/walkthrough-windows-azure-blob-storage-nov-2009-and-later.aspx
http://blogs.msdn.com/b/jnak/archive/2008/10/29/walkthrough-simple-blob-storage-sample.aspx
http://wotudo.net/blogs/wotudo/archive/2010/02/16/copying-files-to-windows-azure-blob-storage.aspx

Repeater

http://msdn.microsoft.com/ru-ru/library/system.web.ui.webcontrols.repeater.aspx
http://www.w3schools.com/ASPNET/aspnet_repeater.asp
http://articles.sitepoint.com/article/asp-net-repeater-control

Приложение А asp - код страницы AzureBlobSample.aspx

```
<%@ Page Language="C#" AutoEventWireup="true"
  CodeBehind="AzureBlobSample.aspx.cs"
Inherits="WebRole1.AzureBlobSample" %>

 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

 <html xmlns="http://www.w3.org/1999/xhtml">
 <head runat="server">
     <title></title>
     <style type="text/css">
         .style1
         {
             height: 25px;
         }
     </style>
 </head>
 <body>
     <form id="form1" runat="server">
     <div>

         <asp:Label ID="Label1" runat="server" Font-Bold="True"
Font-Size="X-Large"
             Text="My Image Gallery - Azure Blob
Example"></asp:Label>
         <br />
         <br />
         <table style="width:100%;">
             <tr>
                 <td>
                     <asp:Label ID="lb_name" runat="server"
Text="Название"></asp:Label>
                 </td>
```

```
                <td>
                    <asp:TextBox ID="tb_name"
runat="server"></asp:TextBox>
                </td>
            </tr>
            <tr>
                <td>
                    <asp:Label ID="lb_desc" runat="server"
Text="Описание"></asp:Label>
                </td>
                <td>
                    <asp:TextBox ID="tb_desc"
runat="server"></asp:TextBox>
                </td>
            </tr>
            <tr>
                <td>
                     </td>
                <td>
                     </td>
            </tr>
            <tr>
                <td class="style1">
                    <asp:Label ID="lb_file" runat="server"
Text="Файл"></asp:Label>
                </td>
                <td class="style1">
                    <asp:FileUpload ID="fu_upload" runat="server"
/>
                </td>
            </tr>
            <tr>
                <td>
                     </td>
                <td>
                    <asp:Button ID="btn_upload" runat="server"
onclick="btn_upload_Click"
                        Text="Загрузить" />
                </td>
            </tr>
            <tr>
                <td>
                    <asp:Label ID="lb_status"
runat="server"></asp:Label>
                </td>
                <td>
                     </td>
            </tr>
            <tr>
                <td>
                     </td>
                <td>
                    <asp:ListView ID="lv_images" runat="server"
```

```
onitemdatabound="lv_images_ItemDataBound">
            <LayoutTemplate>
                <asp:PlaceHolder ID="itemPlaceholder"
runat="server" />
            </LayoutTemplate>
            <EmptyDataTemplate>
                <h2>No Data Available</h2>
            </EmptyDataTemplate>
            <ItemTemplate>
                <div class="item">
                    <ul
style="width:40em;float:left;clear:left" >
                        <asp:Repeater ID="blobMetadata"
runat="server">
                        <ItemTemplate>
                            <li><%# Eval("Name") %><span><%#
Eval("Value") %></span></li>
                        </ItemTemplate>
                        </asp:Repeater>
                        <li>
                            <asp:LinkButton ID="deleteBlob"
                                    OnClientClick="return
confirm('Delete image?');"
                                    CommandName="Delete"
                                    CommandArgument='<%#
Eval("Uri")%>'
                                    runat="server" Text="Удалить"
oncommand="OnDeleteImage" />

                            <asp:LinkButton ID="CopyBlob"
                                    OnClientClick="return
confirm('Copy image?');"
                                    CommandName="Copy"
                                    CommandArgument='<%#
Eval("Uri")%>'
                                    runat="server"
Text="Копировать" oncommand="OnCopyImage" />

                        </li>
                    </ul>
                    <img src="<%# Eval("Uri") %>" alt="<%#
Eval("Uri") %>" style="float:left"/>
                </div>
            </ItemTemplate>
        </asp:ListView>

                    </td>
            </tr>
            <tr>
                <td>

                    <br />
```

```
                </td>
                <td>
                    </td>
            </tr>
        </table>

    </div>
    </form>
</body>
</html>
```

Приложение Б AzureBlobSample.aspx.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using Microsoft.WindowsAzure;
using Microsoft.WindowsAzure.ServiceRuntime;
using Microsoft.WindowsAzure.StorageClient;
using System.Configuration;
using System.Collections.Specialized;

namespace WebRole1
{
    public partial class AzureBlobSample : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            try
            {
                if (!IsPostBack)
                {
                    this.EnsureContainerExists();
                }
                this.RefreshGallery();
            }
            catch (System.Net.WebException we)
            {
                lb_status.Text = "Network error: " + we.Message;
                if (we.Status ==
System.Net.WebExceptionStatus.ConnectFailure)
                {
                    lb_status.Text += "<br />Please check if the
blob service is running at " +

ConfigurationManager.AppSettings["storageEndpoint"];
                }
            }
            catch (StorageException se)
            {
                Console.WriteLine("Storage service error: " +
```

```
se.Message);
            }

        }

        private void RefreshGallery()
        {
            lv_images.DataSource =
              this.GetContainer().ListBlobs(new
BlobRequestOptions()
              {
                  UseFlatBlobListing = true,
                  BlobListingDetails = BlobListingDetails.All
              });
            lv_images.DataBind();
        }

        private CloudBlobContainer GetContainer()
        {
            CloudStorageAccount.SetConfigurationSettingPublisher(
                (configName, configSettingPublisher) =>
                {
                    var connectionString =

RoleEnvironment.GetConfigurationSettingValue(configName);
                    configSettingPublisher(connectionString);
                }
            );

            var account =
CloudStorageAccount.FromConfigurationSetting("DataConnectionString
");
            var client = account.CreateCloudBlobClient();
            return
client.GetContainerReference(RoleEnvironment.GetConfigurationSetti
ngValue("ContainerName"));
        }

        private void EnsureContainerExists()
        {
            var container = GetContainer();
            container.CreateIfNotExist();
            var permissions = container.GetPermissions();
            permissions.PublicAccess =
BlobContainerPublicAccessType.Container;
            container.SetPermissions(permissions);
        }

        private void SaveImage(string id, string name, string
description, string fileName, string contentType, byte[] data)
        {
            var blob = this.GetContainer().GetBlobReference(name);
            blob.Properties.ContentType = contentType;
```

```
            try
            {
                blob.UploadFromStream(fu_upload.FileContent);
                blob.Metadata["Id"] = id;
                blob.Metadata["Filename"] = fileName;
                blob.Metadata["ImageName"] =
String.IsNullOrEmpty(name) ? "unknown" : name;
                blob.Metadata["Description"] =
String.IsNullOrEmpty(description) ? "unknown" : description;
                blob.SetMetadata();
                lv_images.DataBind();
            }
            catch (Exception ex)
            { lb_status.Text = ex.Message; }
        }

        protected void btn_upload_Click(object sender, EventArgs
e)
        {
            if (fu_upload.HasFile)
              {
                lb_status.Text = "Inserted [" +
fu_upload.FileName + "] - Content Type [" +
                fu_upload.PostedFile.ContentType + "] - Length
[" + fu_upload.PostedFile.ContentLength + "]";
                this.SaveImage(Guid.NewGuid().ToString(),
                  tb_name.Text,
                  tb_desc.Text,
                  fu_upload.FileName,
                  fu_upload.PostedFile.ContentType,
                  fu_upload.FileBytes
                );
                RefreshGallery();
              }
              else
                lb_status.Text = "No image file";
           }

        protected void lv_images_ItemDataBound(object sender,
ListViewItemEventArgs e)
        {
            if (e.Item.ItemType == ListViewItemType.DataItem)
            {
                var metadataRepeater =
e.Item.FindControl("blobMetadata") as Repeater;
                var blob = ((ListViewDataItem)(e.Item)).DataItem
as CloudBlob;
                if (blob != null)
                {
                    if (blob.SnapshotTime.HasValue)
                    {
                        var delBtn =
```

```
e.Item.FindControl("deleteBlob") as LinkButton;
                        if (delBtn != null) delBtn.Text = "Delete
Snapshot";
                        var snapshotBtn =
e.Item.FindControl("SnapshotBlob") as LinkButton;
                        if (snapshotBtn != null)
snapshotBtn.Visible = false;
                    }
                    if (metadataRepeater != null)
                    {
                        metadataRepeater.DataSource = from key in
blob.Metadata.AllKeys
                                                      select new
                                                      {
                                                          Name =
key,
                                                          Value =
blob.Metadata[key]
                                                      };
                        metadataRepeater.DataBind();
                    }
                }
            }
        }

        protected void OnDeleteImage(object sender,
CommandEventArgs e)
        {
            try
            {
                if (e.CommandName == "Delete")
                {
                    var blobUri = (string)e.CommandArgument;
                    var blob =
this.GetContainer().GetBlobReference(blobUri);
                    blob.DeleteIfExists();
                }
            }
            catch (StorageClientException se)
            {
                lb_status.Text = "Storage client error: " +
se.Message;
            }
            catch (Exception) { }
            RefreshGallery();
        }


        protected void OnCopyImage(object sender, CommandEventArgs
e)
        {
            if (e.CommandName == "Copy")
            {
```

```
                    // Prepare an Id for the copied blob
                    var newId = Guid.NewGuid();
                    // получение исходного объекта
                    var blobUri = (string)e.CommandArgument;
                    var srcBlob =
this.GetContainer().GetBlobReference(blobUri);
                    // создание нового бинарного объекта
                    var newBlob =
this.GetContainer().GetBlobReference(newId.ToString());
                    // копирование содержимого исходного объекта
                    newBlob.CopyFromBlob(srcBlob);
                    // получаем метаданные для нового объекта
                    newBlob.FetchAttributes(new BlobRequestOptions
{ BlobListingDetails = BlobListingDetails.Metadata });
                    // изменение метаданных нового объекта, чтобы показать,
что это копия
                    newBlob.Metadata["ImageName"] = "Copy of \"" +
                      newBlob.Metadata["ImageName"] + "\"";
                    newBlob.Metadata["Id"] = newId.ToString();
                    newBlob.SetMetadata();
                    RefreshGallery();
                }
            }

        }
}
```

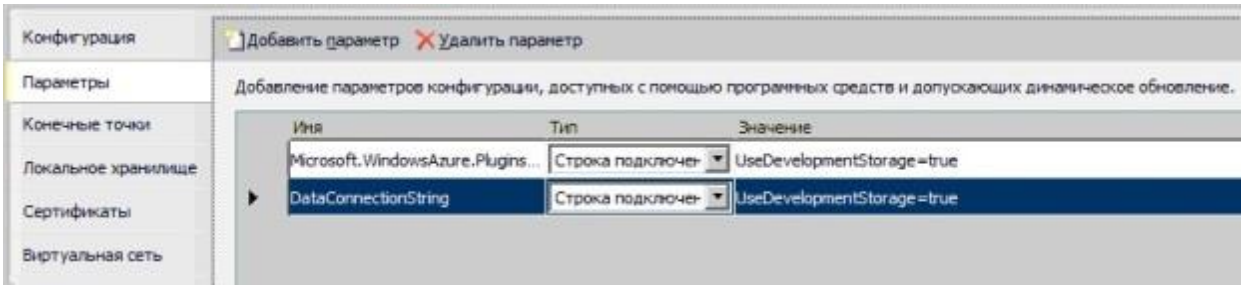
ЛАБОРАТОРНАЯ РАБОТА №7

Тема: **Работа с Windows AzureQueue**

Цель: В рамках данной практики мы рассмотрим два небольших примера, демонстрирующих основы работы с очередями Windows Azure , на примере рабочей и веб - ролей.

Создайте проект облачной службы и добавьте веб - роль и рабочую роль в решение.

В свойствах ролей определим строку подключения к эмулятору хранилища Azure .



увеличить изображение

Рис. 22.1.

Задание 1

Отправка сообщения в очередь

Создадим веб - форму AzureQueueSample со элементами управления - текстовое поле и кнопка:

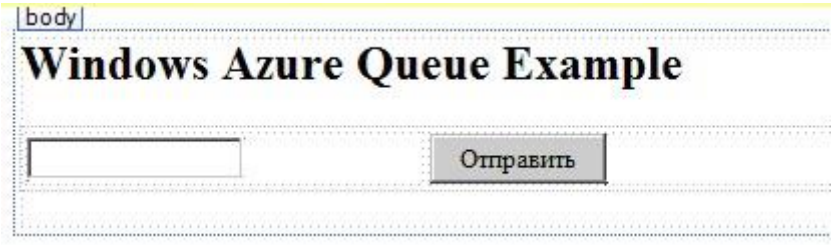


Рис. 22.2.

asp - код веб формы:

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeBehind="AzureQueueSample.aspx.cs"
Inherits="WebRole1.AzureQueueSample" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
    <style type="text/css">
        .style1
        {
            width: 197px;
        }
    </style>
</head>
<body>
    <form id="form1" runat="server">
    <div>

        <asp:Label ID="Label1" runat="server" Font-Bold="True"
Font-Size="X-Large"
            Text="Windows Azure Queue Example"></asp:Label>
        <br />
        <br />
        <table style="width:100%;">
            <tr>
                <td class="style1">
                    <asp:TextBox ID="tb_message"
runat="server"></asp:TextBox>
                </td>
                <td>
                    <asp:Button ID="btn_send" runat="server"
onclick="btn_send_Click"
                        Text="Отправить" />
```

```
                </td>
            </tr>
            </table>
        <br />

    </div>
    </form>
</body>
</html>
```

Добавим следующие ссылки в AzureQueueSample.aspx.cs :

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

Создадим метод, обрабатывающие событие нажатия на кнопку btn_send :

```
protected void btn_send_Click(object sender, EventArgs e)
        {
//определяем параметры учетной записи и клиента для создания очереди и
сообщения
            var storageAccount =
CloudStorageAccount.FromConfigurationSetting("DataConnectionString
");
            var queueClient =
storageAccount.CreateCloudQueueClient();
            var queue =
queueClient.GetQueueReference("messagequeue");
//создаем очередь
            queue.CreateIfNotExist();
//создаем сообщение, источник - текстовое поле tb_message
            var msg = new CloudQueueMessage(tb_message.Text);
//добавляем сообщение в очередь
            queue.AddMessage(msg);
            tb_message.Text = string.Empty;
        }
```

На этом работа с формой закончена.

Извлечение сообщения из очереди

Перейдите с помощью обозревателя решений к файлу Global.asax.cs и добать следующий код в метод ApplicationStart :

```
void Application_Start(object sender, EventArgs e)
        {

CloudStorageAccount.SetConfigurationSettingPublisher((configName,
configSetter) =>
            {

configSetter(RoleEnvironment.GetConfigurationSettingValue(configNa
me));
```

```
            });

        }
```

Затем перейдите к файлу WorkerRole.cs нашей рабочей роли. Добавьте следующие ссылки:

```
using System.Diagnostics;
using System.Threading;
using Microsoft.WindowsAzure;
using Microsoft.WindowsAzure.Diagnostics;
using Microsoft.WindowsAzure.ServiceRuntime;
using Microsoft.WindowsAzure.StorageClient
```

Измените метод OnStart в соответствие со следующим кодом:

```
public override bool OnStart()
        {
            ServicePointManager.DefaultConnectionLimit = 12;


CloudStorageAccount.SetConfigurationSettingPublisher((configName,
configSetter) =>
                {

configSetter(RoleEnvironment.GetConfigurationSettingValue(configNa
me));
                });

        return base.OnStart();


        }
```

Затем приведите метод Run в соответствие с:

```
public override void Run()
        {

            var storageAccount =
CloudStorageAccount.FromConfigurationSetting("DataConnectionString
");
            // получение ссылок на сообщения очереди
            var queueClient =
storageAccount.CreateCloudQueueClient();
            var queue =
queueClient.GetQueueReference("messagequeue");

            // получение сообщение и передача его в лог эмулятора Compute
Emulator
            while (true)
            {
                Thread.Sleep(10000);
```

```
                if (queue.Exists())
                {
                    var msg = queue.GetMessage();
                    if (msg != null)
                    {

Trace.TraceInformation(string.Format("Сообщение '{0}' обработано.",
msg.AsString));
                        queue.DeleteMessage(msg);
                    }
                }
            }
        }
```

Запустите приложение, заполните текстовое поле и нажмите кнопку "Отправить" :

Рис. 22.3.

Затем, откройте интерфейс Compute Emulator :

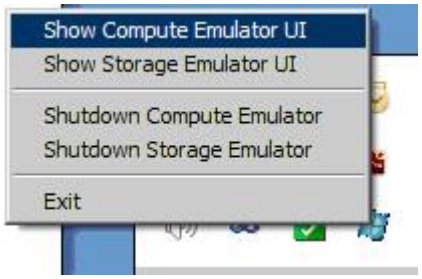


Рис. 22.4.

И откройте лог рабочей роли, чтобы убедиться в работоспособности приложения:

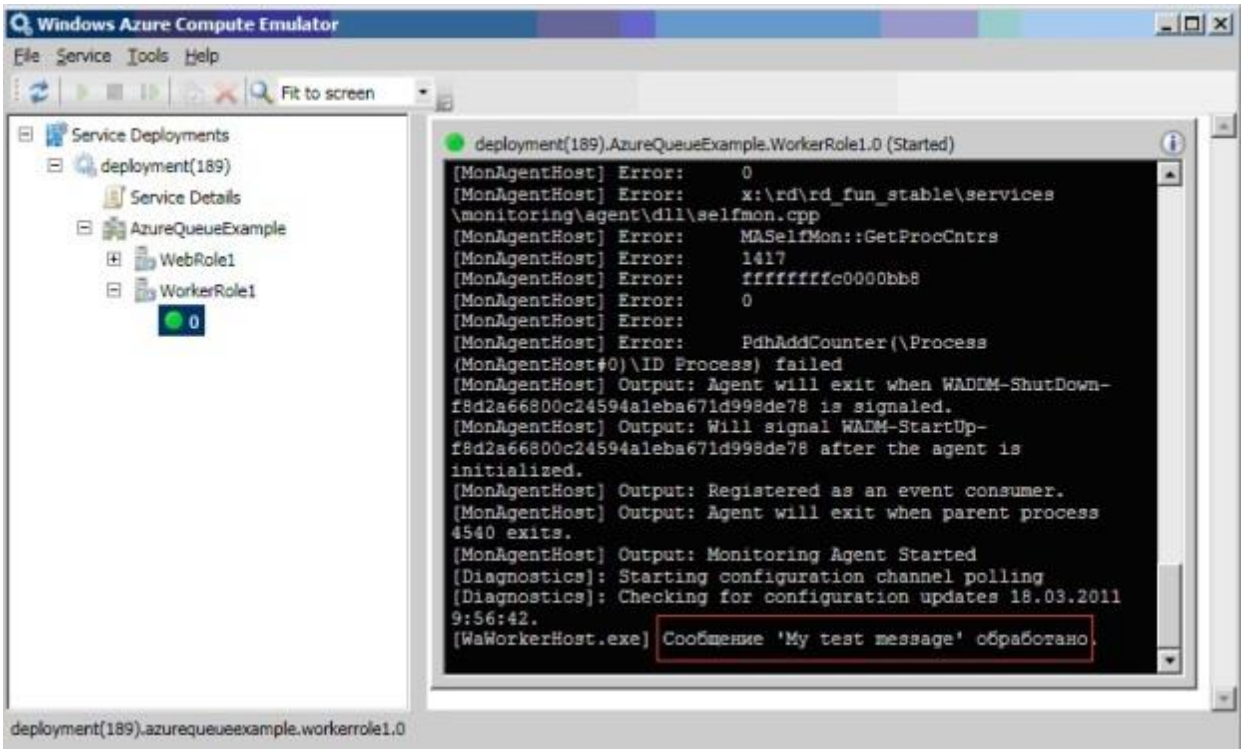


Рис. 22.5.

Задание 2

Для следующего задания можно "усовершенствовать" предыдущую веб - форму, но мы предпочли создать новую - AzureQueueSample2.aspx :

Рис. 22.6.

asp - код веб формы для второго задания:

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeBehind="AzureQueueSample2.aspx.cs"
Inherits="WebRole1.AzureQueueSample2" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
    <style type="text/css">
        .style1
        {
            width: 268px;
        }
        .style2
        {
            width: 247px;
        }
    </style>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:Label ID="Label4" runat="server" Font-Bold="True"
Font-Size="X-Large"
                Text="Windows Azure Queue - Second
Example"></asp:Label>
            <br />
            <table style="width:100%;">
                <tr>
                    <td class="style1">
            <asp:Label id="lb_inert" Text="Добавить в очередь
сообщение:" runat="server" />
                    </td>
                    <td class="style2">
            <asp:TextBox id="tb_insert" runat="server" />
                    </td>
                    <td>
            <asp:Button ID="bnt_addmessage" Text="Добавить сообщение"
runat="server"
                                onclick="bnt_addmessage_Click"
Width="231px" />
                    </td>
                </tr>
                <tr>
                    <td class="style1">
            <asp:Label id="lb_binarydata" Text="Добавить, как
двойчный объект:" runat="server" />
                    </td>
                    <td class="style2">
            <asp:FileUpload ID="fu_insertbinary" runat="server" />
                    </td>
                    <td>
            <asp:Button ID="btn_addbinary" Text="Добавить двоичные
данные" runat="server"
                                onclick="btn_addbinary_Click" />
                    </td>
                </tr>
```

```
                <tr>
                    <td class="style1">
            <asp:Button ID="btn_getmessage" Text="Получить сообщение"
runat="server"
                                onclick="btn_getmessage_Click" />
                    </td>
                    <td class="style2">
            <asp:Label id="lb_retrievedmessage" runat="server" />
                    </td>
                    <td>
                        </td>
                </tr>
            </table>
            <br />
            <asp:Repeater id="Repeater1" runat="server">
                <HeaderTemplate>
                    <table border="1">
                        <tr><td><b>Queue Messages</b></td></tr>
                </HeaderTemplate>
                <ItemTemplate>
                    <tr>
                      <td> <asp:Label ID="Label1" runat="server"
                      Text="<%# Container.DataItem %>" /> </td>
                    </tr>
                </ItemTemplate>
                <FooterTemplate>
                    </table>
                </FooterTemplate>
            </asp:Repeater>
        </div>
    </form>

</body>
</html>
```

Наша задача состоит в следующем:

Реализовать функционал создания очереди
Добавить сообщение в очередь
Добавить в очередь файл, в виде двойного объекта
Отобразить сообщения очереди
Получить сообщение из очереди

Добавьте следующие ссылки в AzureQueueSample2.aspx.cs :

```
using System.IO;
using Microsoft.WindowsAzure;
using Microsoft.WindowsAzure.StorageClient;
using Microsoft.WindowsAzure.ServiceRuntime;
```

Также определим переменные учетной записи, очереди и клиента до метода PageLoad :

```
CloudStorageAccount storageAccount = null;
CloudQueue cloudQueue = null;
CloudQueueClient queueClient = null;
```

Для выполнения первого пункта, добавим следующий код в метод PageLoad :

```
// определение контекста
            this.storageAccount =
CloudStorageAccount.FromConfigurationSetting("DataConnectionString
");
            this.queueClient =
storageAccount.CreateCloudQueueClient();

            // создание очереди
            this.cloudQueue =
this.queueClient.GetQueueReference("myqueue");
            this.cloudQueue.CreateIfNotExist();

            this.DisplayMesssages();
```

Для добавления сообщения в очередь, создадим метод, обрабатывающий событие нажатия кнопки btn_addmessage :

```
protected void bnt_addmessage_Click(object sender, EventArgs e)
        {
            CloudQueueMessage msg = new
CloudQueueMessage(this.tb_insert.Text);
            this.cloudQueue.AddMessage(msg);

            this.DisplayMesssages();
        }
```

Для отображения содержимого очереди, создадим метод DisplayMessages :

```
private void DisplayMesssages()
        {
            // получение первых пяти сообщений, без удаления их из очереди
            var msgs = this.cloudQueue.PeekMessages(5);
            var cloudList = new List<string>();

            foreach (var msg in msgs)
            {
                cloudList.Add("Message ID: " + msg.Id + ";
Message: " + msg.AsString +
                "; Message insertion time: " + msg.InsertionTime);
            }

            // привязка к источнику данных
            this.Repeater1.DataSource = cloudList;
            this.Repeater1.DataBind();
        }
```

Для добавления очередь файла, создадим метод обрабатывающий нажатие кнопки btn_addbinary:

```
protected void btn_addbinary_Click(object sender, EventArgs e)
        {
            // файл не должен превышать размер - лимит сообщения - 8Кб
CloudQueueMessage msg = new
CloudQueueMessage(File.ReadAllBytes(fu_insertbinary.FileName));
            this.cloudQueue.AddMessage(msg);

            this.DisplayMesssages();
        }
```

Осталось только написать метод, для получения сообщения из очереди, обратите внимание, что при этом сообщение из очереди удаляется, а получить можно только первое сообщение из очереди:

```
protected void Page_Load(object sender, EventArgs e)
        {
            this.storageAccount =
CloudStorageAccount.FromConfigurationSetting("DataConnectionString
");
            this.queueClient =
storageAccount.CreateCloudQueueClient();
            this.cloudQueue =
this.queueClient.GetQueueReference("myqueue");
            this.cloudQueue.CreateIfNotExist();
            this.DisplayMesssages();
        }
```

Теперь запустим приложение и протестируем функционал.

Рис. 22.7.

Обратим ваше внимание на то, что файл, который можно добавить в очередь должен соответствовать следующим условиям:

размер файла не должен превышать 8Кб
файл должен находиться: [путь к проекту
AzureQueueExample]\bin\Debug\CloudService1.csx\roles\WebRole1

В случае, если выполнение задания вызвало сложности и затруднения, в приложениях к данной практической работе вы найдете итоговый программный код в том виде, в котором он необходим для последнего задания.

Список вспомогательных материалов

Работа с Windows Azure Queue

http://msdn.microsoft.com/en-us/wazplatformtrainingcourse_exploringwindowsazurestoragevs2010_topic4
http://dotnetbyexample.blogspot.com/2009/07/storing-objects-as-compressed-messages.html
http://soumya.wordpress.com/2010/05/20/azure-simplified-part-4-using-azure-queue-storage/
http://www.dotnetconsult.co.uk/weblog2/PermaLink,guid,c24f8177-f6bf-479b-bd5c-d532e0e40272.aspx

Приложение А AzureQueueSample.aspx.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using Microsoft.WindowsAzure;
using Microsoft.WindowsAzure.StorageClient;
using Microsoft.WindowsAzure.ServiceRuntime;


namespace WebRole1
{
    public partial class AzureQueueSample : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {

        }

        protected void btn_send_Click(object sender, EventArgs e)
        {
            var storageAccount =
CloudStorageAccount.FromConfigurationSetting("DataConnectionString
");
            var queueClient =
storageAccount.CreateCloudQueueClient();
            var queue =
queueClient.GetQueueReference("messagequeue");
            queue.CreateIfNotExist();
            var msg = new CloudQueueMessage(tb_message.Text);
            queue.AddMessage(msg);
            tb_message.Text = string.Empty;
        }
```

```
    }
}
```

Приложение Б Global.asax.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Security;
using System.Web.SessionState;
using Microsoft.WindowsAzure;
using Microsoft.WindowsAzure.ServiceRuntime;

namespace WebRole1
{
    public class Global : System.Web.HttpApplication
    {

        void Application_Start(object sender, EventArgs e)
        {

CloudStorageAccount.SetConfigurationSettingPublisher((configName,
configSetter) =>
            {

configSetter(RoleEnvironment.GetConfigurationSettingValue(configNa
me));
            });

        }

        void Application_End(object sender, EventArgs e)
        {

        }

        void Application_Error(object sender, EventArgs e)
        {

        }

        void Session_Start(object sender, EventArgs e)
        {

        }

        void Session_End(object sender, EventArgs e)
        {

        }

    }
```

```
}
```

Приложение В Worker.cs

```
using System.Diagnostics;
using System.Linq;
using System.Net;
using System.Threading;
using Microsoft.WindowsAzure;
using Microsoft.WindowsAzure.Diagnostics;
using Microsoft.WindowsAzure.ServiceRuntime;
using Microsoft.WindowsAzure.StorageClient;


namespace WorkerRole1
{
    public class WorkerRole : RoleEntryPoint
    {
        public override void Run()
        {

            var storageAccount =
CloudStorageAccount.FromConfigurationSetting("DataConnectionString
");
            var queueClient =
storageAccount.CreateCloudQueueClient();
            var queue =
queueClient.GetQueueReference("messagequeue");
            while (true)
            {
                Thread.Sleep(10000);
                if (queue.Exists())
                {
                    var msg = queue.GetMessage();
                    if (msg != null)
                    {

Trace.TraceInformation(string.Format("Сообщение '{0}' обработано.",
msg.AsString));
                        queue.DeleteMessage(msg);
                    }
                }
            }
        }
        public override bool OnStart()
        {
            ServicePointManager.DefaultConnectionLimit = 12;


CloudStorageAccount.SetConfigurationSettingPublisher((configName,
configSetter) =>
                {

configSetter(RoleEnvironment.GetConfigurationSettingValue(configNa
```

```
me));
                });

        return base.OnStart();

        }
    }
}
```

Приложение Г AzureQueueSample2.aspx.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.IO;
using Microsoft.WindowsAzure;
using Microsoft.WindowsAzure.StorageClient;
using Microsoft.WindowsAzure.ServiceRuntime;

namespace WebRole1
{
    public partial class AzureQueueSample2 : System.Web.UI.Page
    {

        CloudStorageAccount storageAccount = null;
        CloudQueue cloudQueue = null;
        CloudQueueClient queueClient = null;

        protected void Page_Load(object sender, EventArgs e)
        {
            this.storageAccount =
CloudStorageAccount.FromConfigurationSetting("DataConnectionString
");
            this.queueClient =
storageAccount.CreateCloudQueueClient();
            this.cloudQueue =
this.queueClient.GetQueueReference("myqueue");
            this.cloudQueue.CreateIfNotExist();
            this.DisplayMesssages();
        }


        private void DisplayMesssages()
        {
            var msgs = this.cloudQueue.PeekMessages(5);
            var cloudList = new List<string>();

            foreach (var msg in msgs)
            {
                cloudList.Add("Message ID: " + msg.Id + ";
Message: " + msg.AsString +
```

```
                "; Message insertion time: " +
msg.InsertionTime);
            }

            this.Repeater1.DataSource = cloudList;
            this.Repeater1.DataBind();
        }

        protected void bnt_addmessage_Click(object sender,
EventArgs e)
        {
            CloudQueueMessage msg = new
CloudQueueMessage(this.tb_insert.Text);
            this.cloudQueue.AddMessage(msg);

            this.DisplayMesssages();
        }

        protected void btn_addbinary_Click(object sender,
EventArgs e)
        {
            CloudQueueMessage msg = new
CloudQueueMessage(File.ReadAllBytes(fu_insertbinary.FileName));
            this.cloudQueue.AddMessage(msg);

            this.DisplayMesssages();
        }

        protected void btn_getmessage_Click(object sender,
EventArgs e)
        {

            CloudQueueMessage msg = this.cloudQueue.GetMessage();
            this.lb_retrievedmessage.Text = "Retrieved message: "
+ msg.AsString;
            this.cloudQueue.DeleteMessage(msg);

            this.DisplayMesssages();
        }

    }
}
```

ЛАБОРАТОРНАЯ РАБОТА №8

Тема: **Microsoft .Net Service Bus: обзор, обмен сообщениями, управление доступом**

Цель: В рамках данной лекции будут рассмотрены следующие вопросы: MS .Net Service Bus: обзор, концепция. Enterprise Service Bus, Internet Service Bus, обмен сообщениями.

Обзор

Основной задачей .Net Service Bus является обеспечение коммуникаций между приложениями. Использование данных служб решает проблемы:

передачи запросов через брандмауэр;
определения конечных точек (endpoints) сервисов.

На сегодняшний день, наиболее популярным способом решения вышеобозначенных проблем являются веб - службы, базирующиеся на SOAP - протоколах. Клиентские приложения используют WSDL для генерации прокси - классов, для определения конечных точек и получения доступа к сервисам через firewall.

Вторым способом решения данных проблем является Windows Communication Foundation (WCF), использующий основанные веб-протоколы передачи данных, в т.ч. SOAP.

Предприятия, как правило, используют два подхода для решения данных проблем. Первый заключается в том, чтобы выборочно позволять приложениям открывать входящие порты на локальных и сетевых брандмауэрах. Второй - заключается в использовании промежуточных служб, находящихся между брандмауэрами и клиентскими приложениями, и являющихся, по сути, "мостом"", обеспечивающим обмен сообщениями.

Первый подход реализуем только в сравнительно небольших корпоративных сетях, ограничением является эффективное обеспечение безопасности. Проблема второго заключается в трудности реализации, организация маршрутизации между тысячами и миллионами соединений потребует значительных издержек.

Рассмотрим более подробно .Net Service Bus, чтобы понять, как данный сервис справляется с обеспечением эффективного обмена сообщениями.

Концепция .Net Service Bus

Первоначально для того, чтобы воспользоваться возможностями .Net Service Bus, приложения должны зарегистрироваться в соответствующем реестре. Когда приложение обращается к службе, находящейся за брандмауэром, с помощью ,Net Service Bus определяется конечная точка службы и с ней устанавливается связь., т.е. непосредственно сам сервис выполняется за брандмауэром, а соединение с ним обеспечивает Service Bus. Клиенты видят только IP адрес, предоставляемый Service Bus, а не IP, предоставляемый компанией. Данный подход использует возможности .Net Access Control для обеспечения безопасности доступа.

Фактически, приложение использующее .Net Service Bus, реализует WCF функционал, но это не является обязательным. Приложение, обращающееся к сервисам за брандмауэром также может сформировать SOAP или REST - запрос.

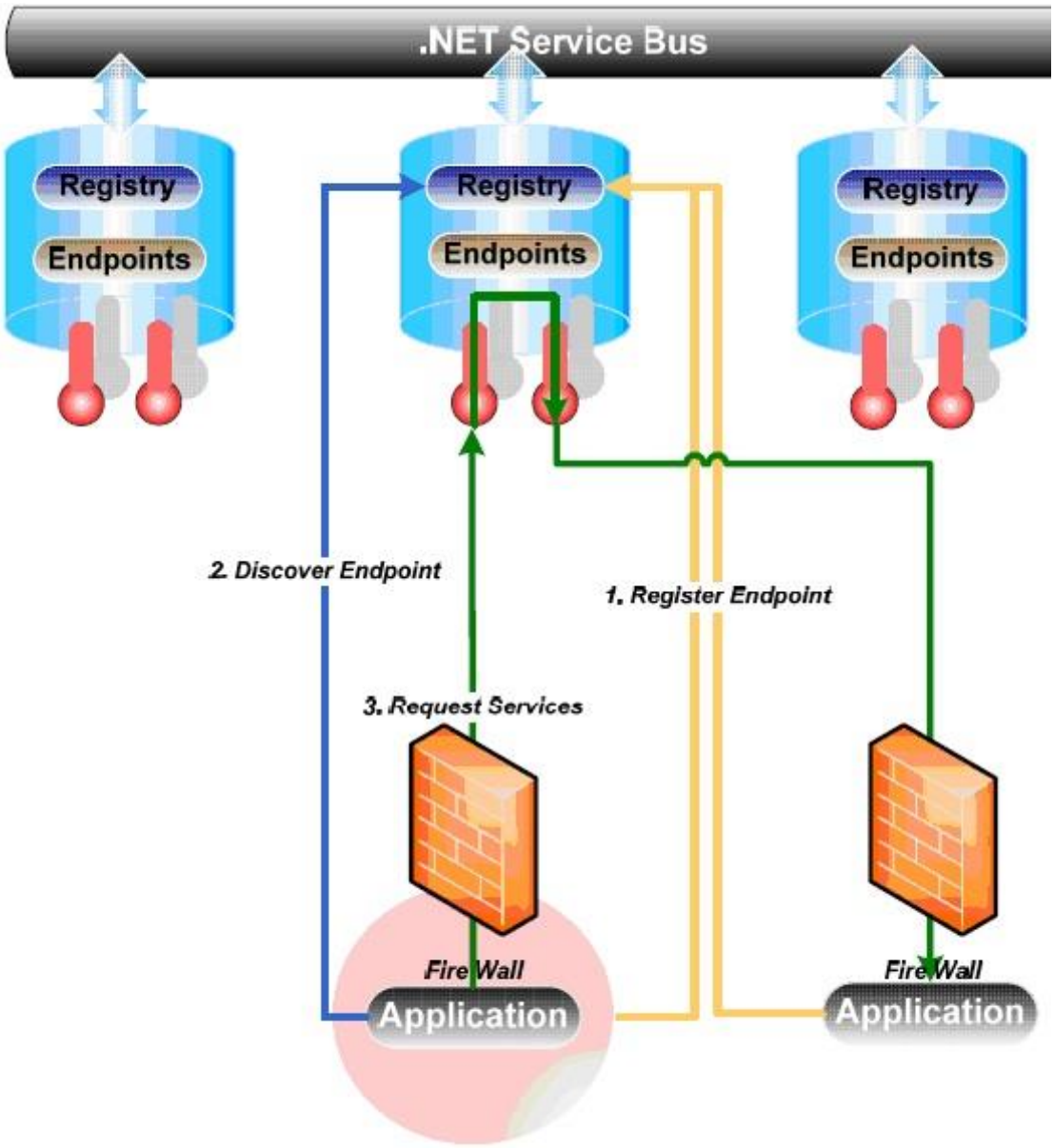


Рис. 25.1. (Источник -книга " Introducing Windows Azure" Henry Li)

Шаблон Enterprise Service Bus (ESB)

Enterprise Service Bus, или сервисная шина предприятия — подход к построению распределённых корпоративных информационных систем. Обычно включает в себя промежуточное ПО, которое обеспечивает взаимосвязь между различными приложениями по различным протоколам взаимодействия.

Шаблон ESB требует:

интегрированных механизмов идентификации и управления доступом
механизма присваивания имен сервисам
общей среды обмена сообщениями

Шаблон ESB помогает преодолеть различия между сервисами с точки зрения управления идентификацией, соглашений о присваивании имен, форматов сообщений и протоколов связи. Как только сервис попадает в шину, все остальные сущности, входящие в нее, могут связываться с ним, даже несмотря на то, что в

обычных условиях взаимодействие между ними напрямую было бы невозможным.

К продуктам и технологиям реализации ESB можно отнести:

1. Active Directory
2. UDDI
3. BizTalk Server
4. MSMQ
5. WCF

Сервисная шина Интернета (Internet Service Bus – ISB)

Концепция сервисной шиной Интернета - разрабатываемый подход, при котором возможности шаблона ESB используются в рамках Интернета. Основное отличие от обычного ESB подхода заключается в том, что компоненты ESB должны быть спроектированы и реализованы для работы в вычислительном "облаке".

ISB сделала бы возможным интегрировать вашу локальную ESB с вашими сервисами, выполняющимися в "облаке", с различными сервисами сторонних производителей, RIA и веб - приложениями.

Проблемы реализации двусторонней Интернет связи:

– нехватка IPv4-адресов;
– безопасность - как правило, локальное программное обеспечение практически полностью отгорожено от внешнего мира множеством уровней межсетевых экранов и другими защитными сетевыми устройствами.

Тем не менее на сегодняшний день можно говорить о наличии двунаправленных приложений, таких как:

сервисы мгновенного обмена сообщениями (ICQ, MSN)

сетевые игры

приложения совместного использования файлов (torrent - клиенты)

Обмен сообщениями

Центральной частью .Net Services Bus является сервис ретрансляции, обеспечивающий возможность обмена сообщениями.

Сервис ретрансляции поддерживает:

1. однонаправленный обмен сообщениями;
2. синхронный обмен сообщениями;
3. обмен сообщениями между равноправными участниками сети.

В этом случае вашим сервисам больше нет необходимости создавать локальных слушателей транспортного уровня; они полностью полагаются на сервис ретрансляции в вопросах обработки указанных транспортных аспектов связи. Сервис ретрансляции просто пересылает входящие сообщения на ваш локальный сервис.

Список источников для самостоятельного изучения

.Net Service Bus

1. http://msdn.microsoft.com/ru-ru/library/ee872418.aspx
2. http://www.microsoft.com/windowsazure/AppFabric/Overview/default.aspx#top
3. http://itechthoughts.wordpress.com/2009/04/12/windows-azure-service-bus-publishsubscribe-example/

Работа с .Net Service Bus

1. http://msdn.microsoft.com/en-us/magazine/dd569756.aspx

Сервисная шина Интернета

– http://msdn.microsoft.com/en-us/library/bb906065.aspx

Enterprise Service Bus

http://www.ibm.com/developerworks/ru/library/ws-whyesb/

Критерии оценки отчетов по лабораторным работам

Оценивание защиты лабораторной работы проводится при представлении отчета в электронном виде, по двухбалльной шкале: «зачтено», «незачтено».

Оценка «зачтено» выставляется студенту, если он представляет к защите отчет по лабораторной работе, удовлетворяющий требованиям по поставленным заданиям, по оформлению, демонстрирует владение методами и приемами теоретических и/или практических аспектов работы.

Оценка «незачтено» выставляется студенту, если он не владеет методами и приемами теоретических и/или практических аспектов работы, допускает существенные ошибки в работе, представляет отчет с существенными отклонениями от правил оформления письменных работ.

У