

Документ подписан простой электронной подписью

Информация о владельце:

ФИО: Макаренко Елена Николаевна

Должность: Ректор

Дата подписания: 29.07.2022 18:19:56

Уникальный программный ключ:

c098bc0c1041cb2a4cf926cf171d6715d99a6ae00adc8e27b55cbe1e2dbd7c78



# СИСТЕМЫ АНАЛИТИКИ БОЛЬШИХ ДАННЫХ

КУРС ЛЕКЦИЙ

# I. ЦЕЛИ И ЗАДАЧИ ОСВОЕНИЯ ДИСЦИПЛИНЫ

## Цели освоения дисциплины:

формирование у обучающихся способности осуществлять руководство проектами по созданию, поддержке и использованию комплексных систем на основе аналитики больших данных с применением новых методов и алгоритмов машинного обучения.

## Задачи освоения дисциплины:

- формирование у обучающихся знаний о методологии и принципах руководства проектами по созданию, поддержке и использованию комплексных систем на основе аналитики больших данных со стороны заказчика;
- развитие у обучающихся умения применять современные инструментальные средства и системы программирования для разработки новых методов и моделей машинного обучения;
- развитие у обучающихся умения решать задачи по руководству коллективной проектной деятельностью для создания, поддержки и использования комплексных систем на основе аналитики больших данных со стороны заказчика.

# II. СОДЕРЖАНИЕ И СТРУКТУРА ДИСЦИПЛИНЫ

Трудоёмкость дисциплины составляет 5 зачётных единиц, 180 часов.

в том числе 1 зачётная единица, 36 часов на экзамен.

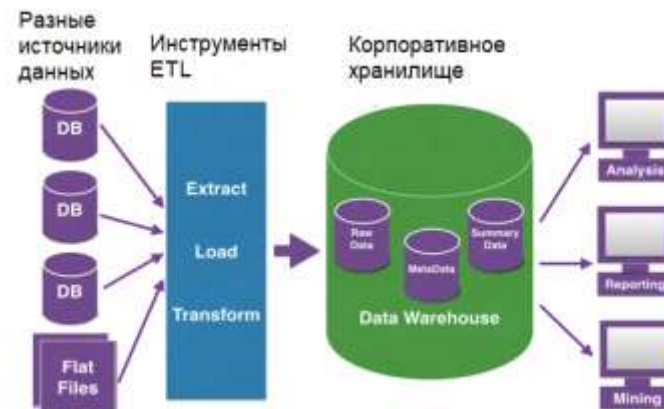
Форма промежуточной аттестации: экзамен.

№ п/ п	Темы дисциплины	Семестр	Виды учебной работы и их трудоёмкость, часы (в том числе с использованием онлайн-курсов)				Наименования оценочных средств
			Контактная работа			Самостоятельная работа	
			Лекции и	Практические занятия	Лабораторные занятия		
<b>Модуль 1. Основы построения и работы с системами аналитики больших данных</b>							
1	<b>Архитектура систем аналитики больших данных.</b>	3	2	-	-	6	Практическая работа №1
2	<b>Хранение больших данных в облаке.</b> 1. Хранилища общего назначения. Форматы хранения данных. Облачное хранилище Microsoft Azure Storage. Облачное хранилище AWS. 2. Реляционные базы данных. Azure SQL. AWS RDS. 3. Нереляционные базы данных. Сервисы нереляционных баз данных от Azure и AWS. 4. Корпоративные хранилища данных (DWH). Azure SQL DWH. AWS RedShift. 5. Хранилища данных типа Data Lake. Azure Data Lake Store. AWS Data Lake Solutions.	3	4	12	-	28	Практическая работа №1

# II. СОДЕРЖАНИЕ И СТРУКТУРА ДИСЦИПЛИНЫ

Модуль 2. Разработка систем аналитики больших данных							
3	<p><b>Доставка больших данных в облако.</b></p> <p>1. Прямая загрузка данных. Доставка данных в облачное хранилище общего назначения. Доставка данных в реляционные БД и хранилища. Доставка данных в нереляционные базы данных. Доставка данных в HDFS-совместимые хранилища.</p> <p>2. Прямая загрузка потоковых данных. Azure Event Hub. AWS Kinesis Data Streams. Облачные сервисы развертывания кластерных систем.</p> <p>3. Облачные сервисы копирования и трансформации данных. Azure Data Factory. AWS Data Pipeline. AWS Glue.</p>	3	6	12	-	28	Практическая работа №2
4	<p><b>Аналитика больших данных в облаке.</b></p> <p>1. Интерактивный анализ данных. Анализ реляционных данных. Azure Data Lake Analytics. AWS Athena. Apache Spark. Встроенные редакторы запросов сервиса CosmosDB</p> <p>2. Потоковый анализ данных. Общие сведения. Azure Stream Analytics. Amazon Kinesis Analytics. Apache Storm.</p> <p>3. Пакетный анализ данных. Hadoop. Apache Pig. Apache Hive.</p>	3	6	12	-	28	Практическая работа №3
	Промежуточная аттестация (для дисциплин с экзаменом)	3	-	-	-	36	Экзаменационные вопросы и билеты
	<b>Итого часов</b>	<b>3</b>	<b>18</b>	<b>36</b>	<b>-</b>	<b>126</b>	<b>-</b>

# Модуль 1. Основы построения и работы с системами аналитики больших данных



## КРАТКОЕ СОДЕРЖАНИЕ:

### 1. Архитектура систем аналитики больших данных.

1.1. Облачные технологии. Модели развертывания. Способы создания ресурсов в облаке.

1.2. Безопасность облачных ресурсов

1.3. Большие данные и источники данных. Форматы. Преобразование данных из различных форматов. Режимы обработки больших данных.

### 2. Хранение больших данных в облаке.

2.1. Хранилища общего назначения. Форматы хранения данных. Облачное хранилище Microsoft Azure Storage. Облачное хранилище AWS.

2.2. Реляционные базы данных. Azure SQL. AWS RDS.

2.3. Нереляционные базы данных. Сервисы нереляционных баз данных от Azure и AWS.

2.4. Корпоративные хранилища данных (DWH). Azure SQL DWH. AWS RedShift.

2.5. Хранилища данных типа Data Lake. Azure Data Lake Store. AWS Data Lake Solutions.

# ЛЕКЦИЯ 1:

Введение в предмет. Особенности  
больших данных и их обработки

# Особенности больших данных и их обработки

- Технологии больших данных позволяют работать с огромными массивами неструктурированных и слабоструктурированных данных или с потоками данных, анализируя их и находя в них *скрытые закономерности*.
- В качестве источников больших данных часто выступают *события, совершаемые массово и фиксируемые в базе данных (БД) или в файлах*. Например, это могут быть файлы логов высоконагруженного веб-сервера, игровые действия пользователя в массовой игре, хранящиеся в нереляционной базе, или данные, относящиеся к бизнес-процессам, хранящиеся в реляционном хранилище данных.
- События на пути от источников к приемнику образуют *поток сообщений*. Если источников событий много и количество сообщений, отправляемых одиночным источником в единицу времени, велико, то возникает задача *концентрации этих сообщений*, то есть предоставления им *общей входной точки* («воронки» или «хаба») для последующего сохранения в той или иной базе данных.
- Согласование сообщений источников и хранилища, помимо их концентрации, может потребовать *предварительной фильтрации* (когда сохраняются только те сообщения, которые отвечают определенным критериям), *маршрутизации* (перенаправление сообщений различного типа в разные источники) и *трансформации* (например, выборка определенных полей сообщения, агрегация, арифметические преобразования и пр.). *Выполнение всех этих действий относится к потоковому анализу больших данных*.
- Помимо пассивного сохранения сообщений, очень часто может требоваться *выполнение каких-либо действий в ответ на появление определенного сообщения или тренда в потоке*. Пример такой задачи — обработка потока транзакций от банкомата в банковский процессинговый центр для установления факта мошеннических действий. *Концептуально похожий пример — анализ потока сообщений в системе логирования в целях выявления хакерских атак (например, веб-приложений) или мошеннических действий (в онлайн-играх)*.

# Особенности обработки больших данных

! После того как в системе накопилось много данных за определенный промежуток времени, может потребоваться их анализ специалистом по анализу данных. Например, данные системы мониторинга потребления электроэнергии представляют интерес с точки зрения выявления как общей структуры энергопотребления, так и частных аномалий в виде поиска наиболее расточительных потребителей, узких мест. Подобный анализ должен производиться при непосредственном взаимодействии с **исследователем данных (data scientist)** и потому называется **интерактивным**.

!! Когда закономерности в данных нельзя обнаружить с помощью традиционных методов анализа, то есть путем выполнения стандартных действий (фильтрация, агрегирование, объединение, пересечение, сортировка), используются **алгоритмы глубокого изучения данных, часто называемые машинным обучением**. Суть его состоит в том, что для представления закономерностей, скрытых в данных, задействуют различные математические модели, и собственно «обучение» состоит в подборе изменяемых параметров этих моделей так, чтобы обеспечить наибольшее согласование между реальными результатами и результатами, полученными моделью.

!!! Как только характер закономерностей в больших данных установлен на этапе интерактивного анализа, может возникнуть задача построения системы, выполняющей периодический анализ накопившихся данных — **пакетный анализ**.

Помимо собственно данных, выдаваемых техническими устройствами того или иного вида, к большим данным можно отнести данные, **генерируемые сложными социальными и техническими системами**.

*В качестве примера можно привести систему общественного транспорта с регистрацией как прямых событий (включая данные мониторинга транспортных средств, оплаты пассажирами проезда), так и связанных с ними (например, событий, связанных с поломками подвижного состава и инфраструктурных элементов).*



# Особенности обработки больших данных

**При построении систем, оперирующих большими данными,** возникает много технических проблем, связанных с хранением данных и их обработкой, которые сводятся к построению больших кластеров серверов, объединенных высокоскоростными и высокопроизводительными сетями передачи данных.

С ростом масштаба больших данных (объемы хранения, ежесекундный поток данных и др.) требуются все более мощные вычислительные ресурсы. Как следствие, каждой организации, отвечающей за подобные системы, становится необходимо иметь свой центр обработки данных (ЦОД, датацентр), что влечет определенные трудности: нужно помещение ЦОД с системой поддержания микроклимата, системой электроснабжения, вентиляции, кондиционирования и т. д. Масштабирование системы в подобных случаях сопряжено с необходимостью закупки серверов, расширения площадей хранения, расширения полосы пропускания сети, подвода новых мощностей от электросетей и т. д.

Кроме того, требуется штат как высококвалифицированных системных администраторов разного профиля для обслуживания центра, так и энергетиков, специалистов по кондиционированию, вентиляции и пр.

Чтобы помочь справиться со всеми этими задачами, **облачные провайдеры** предоставляют целый арсенал сервисов, упрощающих хранение, анализ и визуализацию больших данных. Итак, обе технологии — большие данные и облачные среды — дополняют и обогащают друг друга, создавая симбиотическую среду для анализа и обработки огромных массивов информации.

# Общие вопросы и понятия. Облачные технологии.

Облачные технологии появились в 2006 году один из крупнейших американских интернет-магазинов Amazon предоставил свои неиспользуемые вычислительные ресурсы (а к тому времени их объем стал огромным) совершенно новым образом.

**РЕШЕНИЕ: Эластичные вычислительные ресурсы, предоставляемых облачными провайдерами.** (Платформа Amazon Web Services называет эти ресурсы EC2 — Elastic Cloud Computers.)

Ключевые преимущества облачной модели таковы:

- ресурсы предоставляются по требованию и таким же образом освобождаются;
- плата начисляется за фактическое время использования ресурсов;
- предоставление и освобождение ресурсов производится самим потребителем ресурсов через веб-портал, без всякой бумажной волокиты с договорами.

Помимо виртуальных машин, в облачных средах предоставляются различные сервисы, позволяющие строить различные архитектуры: сервисы виртуальных сетей, подсетей, балансировщики нагрузки, списки контроля доступа (Access Control Lists, ACL)), выделенные IP-адреса и др.

Эти сервисы составляют **основу инфраструктуры как сервиса (Infrastructure as a Service, IaaS).**

# Общие вопросы и понятия.

## Облачные технологии.

Помимо **инфраструктуры**, облачные провайдеры предоставляют наиболее типовые **приложения в виде веб-сервисов**.

В качестве примера можно привести облачное хранилище данных (cloud storage), сервис предоставления учетных записей (identity provider), сервис хостинга веб-приложений, базу данных как сервис, брокер сообщений, концентратор сообщений и др.

Все эти сервисы, кажущиеся на первый взгляд разрозненным набором, предоставляются как общая платформа.

Доступ к ним унифицируется в виде единообразных API, SDK, возможны их «соединение» между собой, общий мониторинг логов и событий и пр.

Это иной уровень применения ресурсов облака: **платформа как сервис (Platform as a Service)**. PaaS позволяет пользователям создавать не просто программные продукты в рамках одной операционной системы, веб-платформы и др., но целые информационные системы, компонентами которых будут экземпляры облачных сервисов.

*Подобно IaaS, сервисы PaaS обычно допускают масштабирование (как ручное, путем выбора соответствующего их размера, так и автоматическое, с помощью различных метрик и событий).*

# Общие вопросы и понятия.

## Облачные технологии.

Как правило, **сервисы PaaS** предоставляют гораздо меньшие права для доступа к вычислительным ресурсам инфраструктуры, лежащей в их основе.

*Например, сервисы хостинга веб-приложений не позволяют установить специфические программы, COM-компоненты, поменять библиотеку DLL в GAC, изменить запись в реестре и др., поскольку отсутствует root-доступ.*

Но взамен они предоставляют удобные порталы администрирования, интеграцию с другими сервисами, встроенные средства логирования и мониторинга, доступность 99,99 % времени и др.

В настоящее время крупнейшими облачными провайдерами являются Amazon Web Services (AWS), Microsoft Azure, Google Cloud, IBM Bluemix, Oracle.

**В дисциплине будут приведены описания сервисов двух облачных провайдеров: AWS и Microsoft Azure.**

AWS — первый в истории облачный провайдер, а Microsoft Azure — облачный провайдер от корпорации Microsoft, обеспечивающий интеграцию практически со всеми сервисами Microsoft.

# Способы создания ресурсов в облаке.

Облачные провайдеры имеют в основе своих сервисов огромные дата-центры, чьи вычислительные ресурсы с помощью системы виртуализации разделяются на небольшие части:

- голые виртуальные машины различных размеров с установленной операционной системой (IaaS) и
- группы виртуальных машин с установленным софтом, предоставляющим доступ только к своим возможностям (PaaS).

Создать облачный ресурс — значит отправить запрос контроллеру ресурсов, размещенному в облачном ЦОДе, на выделение требуемых вычислительных ресурсов из пула доступных. То есть, по сути, ресурс не создается из ничего, а только выделяется по требованию. Задачу оптимального распределения доступных ресурсов между пользователями целиком решает контроллер.

# Способы создания ресурсов в облаке.

Существует четыре способа управления облачной инфраструктурой (рис. 1.1). **Способ №1 — задействовать веб-портал.** При этом пользователь должен иметь соответствующие права на создание ресурсов. Ручной способ очень прост: у всех облачных провайдеров есть удобные порталы, обширная документация, видеоинструкции и др. Не нужны никакие дополнительные сервисы, SDK и пр.

**Недостатки способа №1:**

- длительное время создания инфраструктуры;
- недостаточная надежность (в случае проблем с ресурсами их придется пере-создавать вручную, со всеми ручными настройками, конфигурированием и пр.);
- трудность переноса инфраструктуры в новый регион или аккаунт — ее понадобится вручную клонировать или копировать (данный недостаток частично сглаживается тем, что облачные провайдеры позволяют копировать или клонировать ресурсы, но эта процедура все равно требует ручного инициирования);
- процесс создания ресурсов в этом случае невозможно автоматизировать.

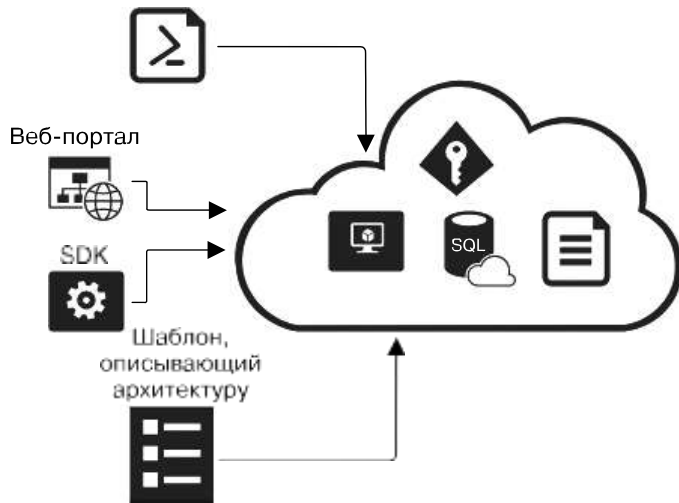
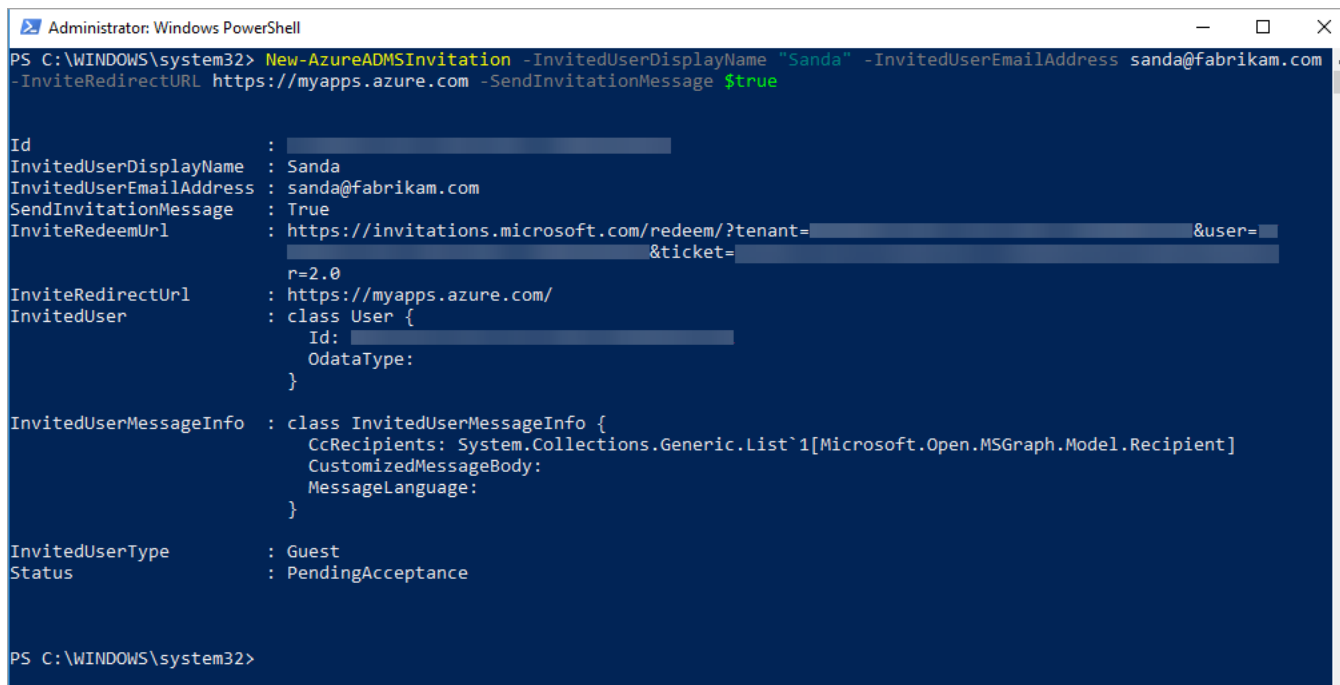


Рис. 1.1. Способы управления облачной инфраструктурой

# Способы создания ресурсов в облаке.

*Способ №2 создания облачных ресурсов относят специализированные расширения для языков командной строки — shell, CMD и др. (например Azure PowerShell, AWS CLI и пр.), работающие в ней напрямую.*

- Для подключения этих расширений к облачным ресурсам необходимо импортировать ключи или выполнить вход в аккаунт через форму ввода логина/пароля.
- Как и в случае SDK для сценарных языков программирования, SDK для командной оболочки позволяет описывать облачную инфраструктуру в виде набора команд, каждая из которых создает или конфигурирует соответствующий облачный сервис.



```
Administrator: Windows PowerShell
PS C:\WINDOWS\system32> New-AzureADMSInvitation -InvitedUserDisplayName "Sanda" -InvitedUserEmailAddress sanda@fabrikam.com -InviteRedirectURL https://myapps.azure.com -SendInvitationMessage $true

Id : 
InvitedUserDisplayName : Sanda
InvitedUserEmailAddress : sanda@fabrikam.com
SendInvitationMessage : True
InviteRedeemUrl : https://invitations.microsoft.com/redeem/?tenant= &user=
                  &ticket=
                  r=2.0
InviteRedirectUrl : https://myapps.azure.com/
InvitedUser : class User {
               Id: 
               OdataType:
             }

InvitedUserMessageInfo : class InvitedUserMessageInfo {
                          CcRecipients: System.Collections.Generic.List`1[Microsoft.Open.MSGraph.Model.Recipient]
                          CustomizedMessageBody:
                          MessageLanguage:
                        }

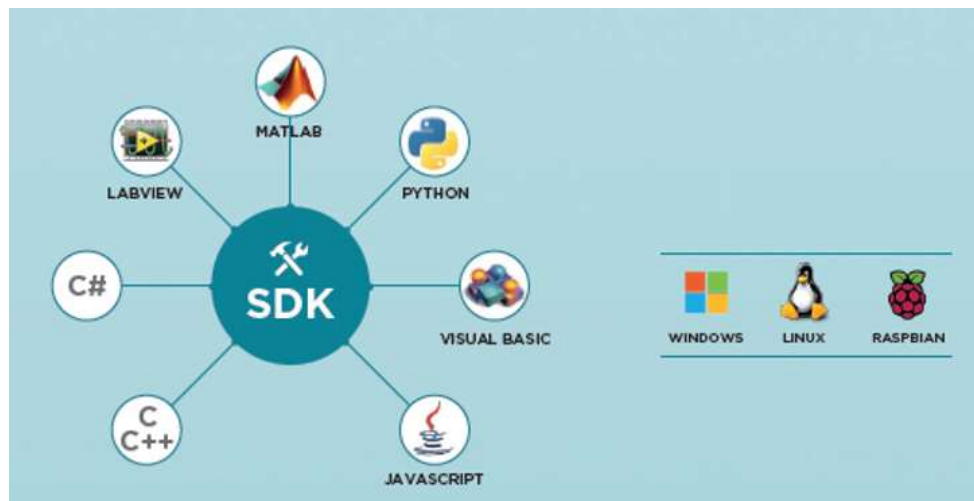
InvitedUserType : Guest
Status          : PendingAcceptance

PS C:\WINDOWS\system32>
```

# Способы создания ресурсов в облаке.

Способ №3 — **применить программные библиотеки (Software Development Kit, SDK), обеспечивающие доступ к ресурсам облака из кода пользовательских программ.**

- Как правило, SDK представляет собой набор классов и методов, облегчающих программные операции с ресурсами облака. *Чтобы обеспечить доступ к таким ресурсам, программа с облачным SDK должна содержать ключи учетной записи, которая будет иметь доступ к облаку.*
- В облаке эти ключи зарегистрированы в виде пользователя в активном каталоге облачного аккаунта, обладающего правами выполнять программное манипулирование ресурсами облака (такой пользователь называется принципалом — service principal).
- Управление этими учетными записями происходит таким же образом, как и учетными записями пользователей облачного веб-портала. В числе достоинств такого подхода — возможность создания программ, которые сами себе создают облачные ресурсы, а также автоматизированного управления облачным аккаунтом.



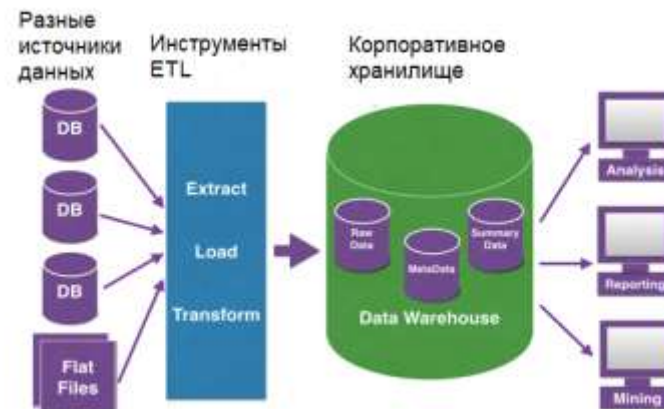


# Способы создания ресурсов в облаке.

Способ №4 создания облачных ресурсов — применить шаблоны. В этом случае все требуемые ресурсы и связи между ними описываются с помощью текстового файла в формате YAML или JSON.

- Такой шаблон может быть загружен в соответствующий облачный сервис напрямую через веб-портал или через CLI-команды.
- Описание инфраструктуры через шаблон — очень мощный механизм, широко применяемый для конфигурирования различных инфраструктур (например, в системах Ansible, Chef, Puppet и др.). Шаблоны представляют собой текстовые файлы, которые могут храниться в репозитории шаблонов или чаще всего в репозитории GitHub.
- Для облака AWS сервис создания ресурсов с помощью шаблонов называется **CloudFormation** (поддерживает YAML и JSON), у Azure это **ARM Template** (в настоящее время поддерживает только JSON).
- На веб-портале AWS имеется специальный редактор, упрощающий создание и конфигурирование шаблона. Последний может быть загружен в файловое хранилище S3, репозиторий CodeCommit или любое другое место, доступное для сервиса CloudFormation. Этот сервис создает стек — набор ресурсов, управляемых совместно (создание, удаление и обновление).
- Сервис CloudFormation очень удобен в применении со сторонними сервисами конфигурирования — например, Ansible. Это широко используемое приложение, задействующее YAML для создания конфигурационных шаблонов, которые служат для администрирования группы серверов (преимущественно Linux, но есть расширения и для Windows), не требуя инсталляции на этих серверах «агентов».
- Для работы Ansible необходимы только ключи доступа к ресурсам (SSH-ключи для Linux-хостов, сертификат для PowerShell-доступа к Windows-хостам или ключи доступа к AWS). Шаблон CloudFormation для Ansible представлен в виде JINJA, допускающего передачу параметров через переменные Ansible.

# Модуль 1. Основы построения и работы с системами аналитики больших данных



## КРАТКОЕ СОДЕРЖАНИЕ:

### 1. Архитектура систем аналитики больших данных.

1.1. Облачные технологии. Модели развертывания. Способы создания ресурсов в облаке.

1.2. Безопасность облачных ресурсов

1.3. Большие данные и источники данных. Форматы. Преобразование данных из различных форматов. Режимы обработки больших данных.

### 2. Хранение больших данных в облаке.

2.1. Хранилища общего назначения. Форматы хранения данных. Облачное хранилище Microsoft Azure Storage. Облачное хранилище AWS.

2.2. Реляционные базы данных. Azure SQL. AWS RDS.

2.3. Нереляционные базы данных. Сервисы нереляционных баз данных от Azure и AWS.

2.4. Корпоративные хранилища данных (DWH). Azure SQL DWH. AWS RedShift.

2.5. Хранилища данных типа Data Lake. Azure Data Lake Store. AWS Data Lake Solutions.

# ЛЕКЦИЯ 2:

## Безопасность облачных ресурсов



# Аспекты безопасности ресурсов в облаке.

Наряду с неоспоримыми преимуществами, хранение и обработка данных в облачных средах потенциально может доставить ряд проблем, которых нет (или, вернее, они проявляются не так отчетливо) в случае размещения и обработки данных в собственных дата-центрах. Это обусловлено рядом причин.

- Во-первых, облачные среды сами по себе публично доступны и все сервисы, если явно не сконфигурировано иное, доступны для всех в Интернете.
- Во-вторых, защита данных и инфраструктуры от непреднамеренных действий пользователей лежит вне компетенции облачного провайдера.
- Кроме того, облачные инфраструктуры, работающие с большими данными, часто содержат в своем составе большие кластеры виртуальных машин, что требует применения специальных мер для обеспечения надежной работы всей системы.
- Помимо этого, информация физически будет передаваться по незащищенным каналам и существует угроза ее перехвата.

# Аспекты безопасности облачных сред.

Наиболее распространенный способ защиты конечных точек облачных сервисов — **ограничение доступа к ним с помощью механизмов аутентификации и создания списков разрешенных IP-адресов**, с которых можно получить доступ к точкам.

**Рассмотрим прежде всего различные способы обеспечения доступа из заданного адресного пространства.**

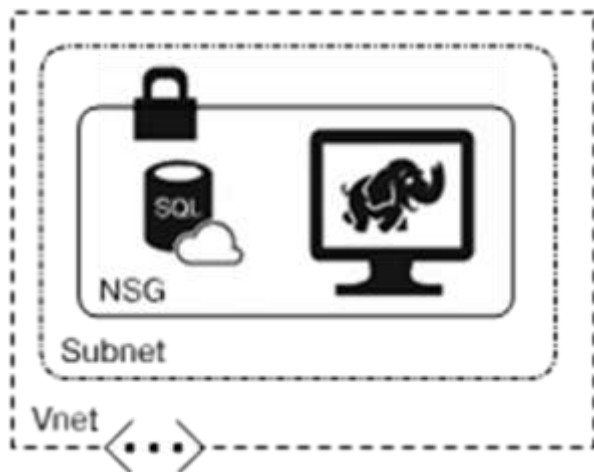


Рис. 1.2. Ограничение доступа к конечным точкам облачных сервисов с помощью сетевых групп безопасности

- Сервисы, относящиеся к IaaS, а также в ряде случаев к PaaS, требуют для своего создания сконфигурированной облачной виртуальной частной сети (VNet, VPC), разбитой на подсети.
- Доступ к конечным точкам сервисов, расположенным в этих подсетях, можно регулировать с помощью конфигурирования сетевых групп безопасности (Network Security Group, NSG) (рис. 1.2), которые представляют собой списки контроля доступа, ACL.

# Аспекты безопасности облачных сред.

Итак, **виртуальная часть сеть** — один из базовых сервисов IaaS. Он представляет собой облачный аналог локальной сети и служит для предоставления диапазона IP-адресов для размещения в них ресурсов.

**Виртуальную частную** сеть можно разделить на подсети (subnet), а между ними — установить правила маршрутизации IP-пакетов.

Кроме того, на подсети можно установить **списки контроля доступа**, которые именуются сетевыми группами безопасности.

Это позволяет **логически разделять архитектуры информационных систем** на различные уровни (например, уровень данных, бизнес-логики, фронтенд) путем размещения каждого уровня в своей подсети и установления правил маршрутизации.

# Аспекты безопасности облачных сред.

NSG представляет собой список доступа, содержащий набор записей.

Каждая запись состоит из таких элементов, как:

- название;
- число, определяющее приоритет просмотра списка записей;
- диапазон IP-адресов (для одного конкретного адреса это /32);
- номер порта;
- действие — ALLOW или DENY («Позволить» или «Отклонить») по отношению к запросу, поступившему с данного адреса.

Кроме того, указывается протокол, к которому применимо действие ALLOW или DENY (TCP, UDP, ICMP и пр.).

**Безопасность конечных точек** в данном случае обеспечивается ограничением к ним доступа извне. Помимо NSG, ряд облачных сервисов, не требующих виртуальной частной сети (например, Azure SQL), имеют **фаерволы** — списки «разрешенных» и «запрещенных» диапазонов.

*! Хорошей практикой является повсеместное использование NSG и фаерволов. При этом необходимо, чтобы все порты, относящиеся к удаленному доступу/управлению (например, 22 для SSH, 3388 для RDP) или непосредственно к сервису (скажем, 1433 для MS SQL), были недоступны из Интернета вне диапазона адресов виртуальной частной сети.*

*!! Для получения же доступа к сервисам из «разрешенной» локальной сети или с разрешенного компьютера следует установить VPN-шлюз из локальной сети или с компьютера в виртуальную частную сеть или непосредственно к экземпляру сервиса.*

# Аспекты безопасности облачных сред.

Помимо шлюза, при соединении локальной сети с виртуальной частной сетью необходимо применить **промежуточный хост, прокси-хост** (рис. 1.3), который будет транслировать запросы и разрешать частные адреса облачных ресурсов из локальной сети.

*Прокси-хост в различных реализациях можно разместить как в облачной сети, так и в локальной.*

В последней может располагаться контроллер домена, сервер БД с данными, которые не могут быть размещены в облаке, а также прочие серверы, которые могут быть размещены только в локальной сети.

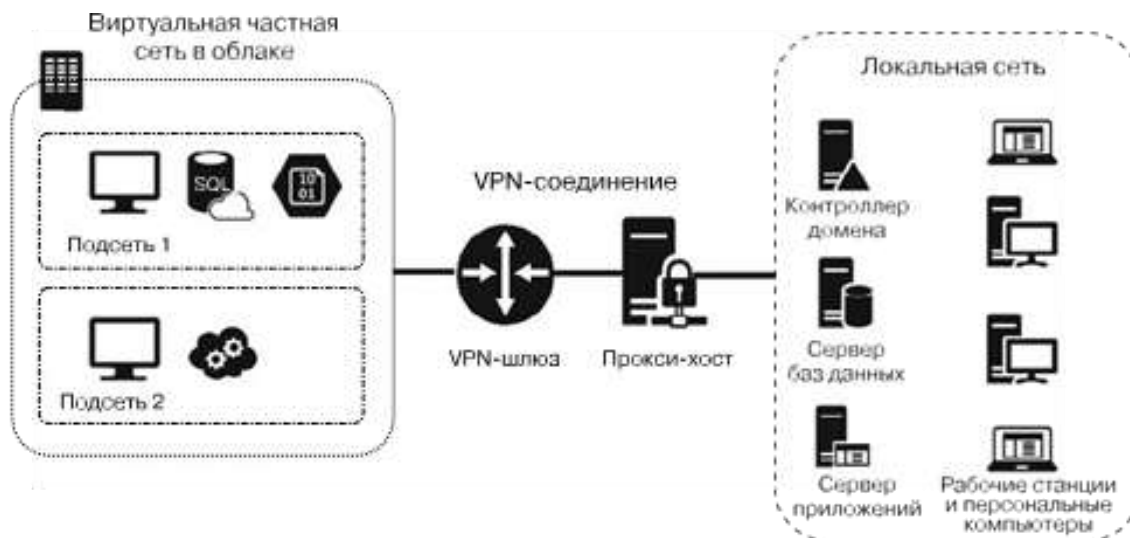


Рис. 1.3. Обеспечение защищенного доступа к облачным ресурсам



# Аспекты безопасности облачных сред.

Некоторые конечные точки (скажем, API управления самого облачного аккаунта) не оборудованы ни фаерволом, ни NSG. Для их защиты используется как шифрование трафика (HTTPS), так и специальные ключи доступа к ресурсам.

- Ключи могут генерироваться непосредственно защищаемым сервисом и применяться в заголовках REST-запросов. Например, сервис хранилища Azure Storage задействует аутентификацию на основе HMAC — **Hash-based Message Authentication Code**, который должен содержать подписанный алгоритмом SHA-256 токен как заголовок аутентификации.
- Следующий стандартный механизм аутентификации запросов — **применение протокола OAuth 2.0**, при котором пользовательские учетные данные хранятся в сервисе хранения учетных данных (в случае Azure это Azure Active Directory, а AWS — Cognito).
- В общем случае облачный сервис хранения учетных данных включает в себя пользователей, роли и API-ресурсы, защищаемые этим протоколом. Любой запрос к REST API ресурсов, зарегистрированным в этом сервисе (а это, по сути, все REST API конечных точек управления облачными ресурсами), должен в своем заголовке содержать токен, который можно получить, если выполнить процедуру ввода учетных данных в каталог.

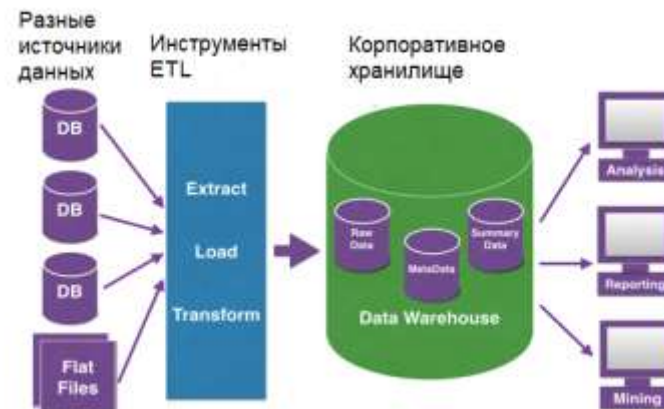
# Аспекты безопасности облачных сред.

Еще один «эшелон» защиты данных в облачных ресурсах — это их **шифрование**. Распространенный подход в данном случае — так называемое **прозрачное шифрование данных (transparent data encryption, TDE)**. Суть его состоит в том, что все шифрование и дешифрование данных происходит «за кулисами», без участия пользователя, с помощью ключей шифрования, которые генерируются и хранятся самим облачным аккаунтом. В случае Azure эти ключи хранятся в Azure KeyVault, а в случае AWS — в AWS KMS.

Следующее звено защиты — **сервисы, отвечающие за мониторинг запросов, поступающих к ресурсам и осуществляющие аудит пользовательской активности**. Например, **Azure Security Center**, который может выполнять мониторинг очень многих ресурсов, выдает рекомендации по их защите и следит за входящим трафиком. На практике в реальной рабочей системе подобный сервис позволил обнаружить и успешно отразить несколько крупных хакерских атак на систему, за которую я отвечал, в том числе с использованием распределенной сети зараженных серверов (ботнет).

И последняя линия защиты данных, встроенная в облачные ресурсы, включает в себя **сервисы анализа пользовательской активности**, например **Audit & Threat Detection для Azure SQL**. Этот сервис обеспечивает внутренний аудит всех запросов в базе данных Azure SQL и анализ их на предмет подозрительных действий. У данного вида защиты есть один недостаток: в результате использования в реальной системе при включении максимального уровня аудита и защиты существенно снижается отзывчивость и общая производительность.

# Модуль 1. Основы построения и работы с системами аналитики больших данных



## КРАТКОЕ СОДЕРЖАНИЕ:

### 1. Архитектура систем аналитики больших данных.

1.1. Облачные технологии. Модели развертывания. Способы создания ресурсов в облаке.

1.2. Безопасность облачных ресурсов

1.3. Большие данные и источники данных. Форматы. Преобразование данных из различных форматов. Режимы обработки больших данных.

### 2. Хранение больших данных в облаке.

2.1. Хранилища общего назначения. Форматы хранения данных. Облачное хранилище Microsoft Azure Storage. Облачное хранилище AWS.

2.2. Реляционные базы данных. Azure SQL. AWS RDS.

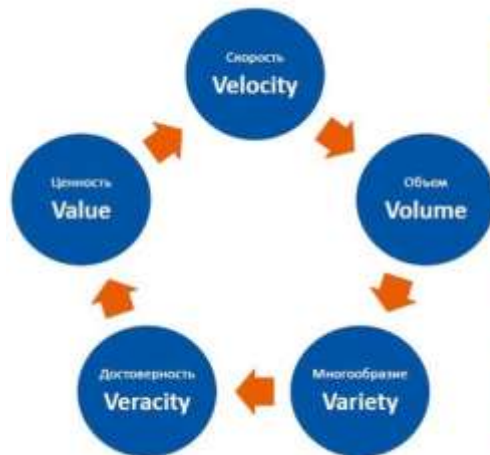
2.3. Нереляционные базы данных. Сервисы нереляционных баз данных от Azure и AWS.

2.4. Корпоративные хранилища данных (DWH). Azure SQL DWH. AWS RedShift.

2.5. Хранилища данных типа Data Lake. Azure Data Lake Store. AWS Data Lake Solutions.

# ЛЕКЦИЯ 3:

Большие данные и источники данных. Форматы. Преобразование данных из различных форматов. Режимы обработки больших данных.



Характеристика	Традиционная база данных	База Больших Данных
Объем информации	От гигабайт до терабайт	От петабайт до эксабайт
Способ хранения	Централизованный	Децентрализованный
Структурированность данных	Структурирована	Полуструктурирована или неструктурирована
Модель хранения и обработки данных	Вертикальная модель	Горизонтальная модель
Взаимосвязь данных	Сильная	Слабая

# Большие данные и источники данных.

Современное состояние **Computer Science** характеризуется тем, что, помимо естественных данных — результатов научных наблюдений, метеорологических данных, социологических и др., — появляется огромное количество данных, связанных с работой информационных систем.

- Эти новые данные существенно отличаются от тех, что анализировались на заре компьютерной эры. Те старые данные (их можно условно назвать естественно-научными) в основном требовали математической обработки.
- В отличие от них данные современных информационных систем (большие данные) не могут быть представлены простыми математическими моделями, чьи параметры следует определить.
- Кроме того, эти данные отличаются существенной неоднородностью, разнообразной и непредсказуемой структурой, и зачастую непонятно, как их обрабатывать и нужно ли это вообще? Можно ли в них найти что-либо полезное?

# Большие данные и источники данных.

Этих данных настолько много, что их анализ за разумное время требует вычислительных ресурсов, существенно превышающих вычислительные ресурсы самой информационной системы. Это значит, что данные часто лежат мертвым грузом, несмотря на скрытые в них закономерности, составляющие полезную информацию, которую требуется найти.

Поиск таких закономерностей называется **Data Mining** — добывание данных из груды пустых данных (по аналогии с пустой породой). Что значит «обрабатывать» данные, и как их добывать? Для ответа на все приведенные вопросы необходимо сначала выяснить, откуда берутся эти большие данные.

# Большие данные и источники данных.

Их источниками могут быть:

- социальные сети — посты, комментарии, сообщения между пользователями и пр.;
- события, связанные с действиями пользователей в веб- или мобильных приложениях;
- логи приложений;
- телеметрия сети устройств из мира «Интернета вещей» (Internet of Things, IoT);
- потоки событий крупных веб-приложений;
- потоки транзакций банковских платежей с метаданными (время, место платежа и т. д.).

Все эти данные должны быть обработаны в режиме реального времени или же постфактум. В обоих случаях они могут размещаться в различных хранилищах (как общего назначения, так и специализированных) и в разных форматах:

CVS, XML, JSON, таблицы в реляционных БД, базах данных NoSQL и пр.

# Большие данные и подходы для их обработки.

Для пакетной обработки исторических данных различных форматов, расположенных в разных хранилищах, необходим единый подход, обеспечивающий выполнение запросов к данным, хранящимся в указанных выше форматах.

**В настоящее время распространены следующие подходы:**

## **1. Преобразование данных из различных форматов в общий, допускающий выполнение запросов к единообразным данным.**

Это можно сделать с помощью облачных сервисов трансформации и копирования, таких как Azure Data Factory и AWS Glue, которые консолидируют данные из разных источников в один.

Такое хранилище традиционно называется **Data Warehouse** (DWH, «склад данных»), а преобразование данных — **ETL (Extract Transform Load** — «извлечение, преобразование, загрузка»).

Данный подход достаточно распространен в традиционных системах, в которых DWH строится на основе кластера SQL-серверов. Подход позволяет использовать все элементы синтаксиса SQL.



# Большие данные и подходы для их обработки.

Для пакетной обработки исторических данных различных форматов, расположенных в разных хранилищах, необходим единый подход, обеспечивающий выполнение запросов к данным, хранящимся в указанных выше форматах.

**В настоящее время распространены следующие подходы:**

## **2. Складирование данных в единое хранилище без изменения формата.**

При этом форматы данных остаются прежними (JSON, XML, CSV и т. д). Такое хранилище, в котором данные размещаются в виде несвязанного набора данных, называется **Data Lake** («озеро данных»).

Файловая система, лежащая в основе подобных хранилищ, совместима с HDFS — распределенной файловой системой, которая, в свою очередь, совместима с Hadoop (Azure Data Lake, AWS EMRFS).

Такое хранилище позволяет задействовать сервисы из экосистемы Hadoop (например, Hive, Apache Spark и др.) и применять иной подход к операциям с данными: ELT (Extract Load Transform — «извлечение, загрузка, преобразование»), когда данные можно трансформировать после загрузки.

*При этом используется сервер аналитики или кластер серверов, содержащий процессор специализированного языка запросов, в котором все разнородные источники данных представляются как внешние источники данных, к которым применим SQL-подобный синтаксис. Для подобного хранилища также может использоваться подход MapReduce (будет более подробно описан далее) и обработка данных в оперативной памяти (in memory processing).*

# Большие данные и подходы для их обработки.

Для пакетной обработки исторических данных различных форматов, расположенных в разных хранилищах, необходим единый подход, обеспечивающий выполнение запросов к данным, хранящимся в указанных выше форматах.

**В настоящее время распространены следующие подходы:**

**3. Кроме того, для обработки потоковых данных существуют специализированные сервисы, допускающие обработку потока сообщений с помощью SQL-подобного синтаксиса:**

например, сервисы Azure Stream Analytics, AWS Kinesis Analytics) или программных структур (Apache Spark Streaming), а также сервисы для приема и концентрации этих сообщений (например, Azure Event Hub, Kafka, Azure Spark и пр).

Итак, что же такое большие данные? Прежде всего, это **огромные массивы данных или потоки**, которые содержат подлежащую извлечению информацию, или же умеренно большие объемы данных, требующие быстрой интерактивной обработки с целью исследования, проверки гипотез, тренировки алгоритмов машинного обучения.

Для выполнения этой обработки необходим **высокий уровень параллелизма, большой объем оперативной памяти (для in memory processing)**, что достигается применением кластеров виртуальных машин. Вот тут-то и проявляются все преимущества облачных сред: **модель IaaS** позволяет создавать кластеры и удалять их по требованию с минимальными затратами.

# Обработка больших данных: пакетный, интерактивный и потоковый анализ

А что значит «обработать большие данные»?

Существует три возможных вида анализа: **пакетный**, **интерактивный** и **потоковый**.

Большие данные могут быть обработаны в **пакетном режиме**, когда они уже присутствуют в хранилище. Чаще всего это необходимо для агрегирования данных и построения аналитических отчетов на их основе.

*Рассмотрим в качестве примера систему мониторинга электроэнергии сети зданий. В этой системе замеры потребляемой мощности передаются с малой периодичностью как сообщения от каждого измерительного модуля. Чтобы получить величину дневного потребления электроэнергии, необходимо сложить все замеры с измерительного модуля каждого пользователя в отдельности. Если итоговый результат нужно, к примеру, формировать в виде ежедневного (еженедельного, ежемесячного и пр.) отчета, то наиболее просто реализовать такую систему следующим образом (рис. 2.1).*



Рис. 2.1. Архитектура системы учета электроэнергии, построенная на основе разделения хранилищ сырых и агрегированных данных — так называемая лямбда-архитектура

# Обработка больших данных: пакетный, интерактивный и потоковый анализ

Все сообщения от измерительных устройств можно хранить в нереляционном хранилище табличного типа, например HBase или Cassandra.

Каждая строка таблицы будет содержать временную метку (то есть время поступления или отправления сообщения), идентификатор устройства, его отправившего, и собственно величину замера.

Для построения периодического отчета с помощью системы бизнес-аналитики (business intelligence, BI) (например, PowerBI или Microsoft SSRS) или отображения этой величины в браузере необходимо, чтобы данные в агрегированном виде были размещены в БД SQL.

*??? А почему нельзя сразу размещать их непосредственно в этой базе?*

**Посчитаем.** Предположим, что измерительное устройство отправляет сообщения каждые пять минут. Это значит — 12 отсчетов в час, или около 105 тыс. в год. Теперь предположим, что система мониторинга собирает данные энергопотребления с каждого устройства заказчика, которых могут быть десятки, а самих заказчиков — тысячи.



Рис. 2.1. Архитектура системы учета электроэнергии, построенная на основе разделения хранилищ сырых и агрегированных данных — так называемая лямбда-архитектура

# Обработка больших данных: пакетный, интерактивный и потоковый анализ

**Пакетная обработка** с группировкой и суммированием результатов может быть выполнена, например, с использованием Hadoop MapReduce или Apache Spark. Сами алгоритмы группировки в данном случае можно реализовать с помощью программ на Java (для MapReduce, Spark) или Python (Spark) и запустить в кластере.

В итоге размеры таблицы с агрегированными данными в базе данных SQL будут существенно меньше, чем в таблице с сырыми данными. Так, если хранится часовое агрегирование, то в SQL-таблице в 12 раз меньше строк, чем в NoSQL, а если суточное — то в 288 раз. При этом для клиентов запрос на получение данных будет очень простой и высокопроизводительный: выбрать из таблицы агрегатов строки, отфильтрованные по идентификатору заказчика и по требуемому временно́му интервалу без каких бы то ни было группировок в самом запросе.

# Замечания к архитектуре пакетной обработки

1. Обеспечить высокую точность позволяет большое количество замеров, следующих с малым временным интервалом. Если интервал постоянный, то можно упустить включение/выключение прибора, произошедшее между замерами.

Эта проблема решается путем передачи не периодических замеров, а замеров, приуроченных к событию: включению или изменению величины проходящей мощности более чем на заданную величину. Такое решение, во-первых, разгрузит сеть от слишком частых передач отсчетов (но не полностью — необходимо оставить периодические сообщения от измерителя о его работоспособности), и во-вторых, существенно уменьшит объем сырой таблицы.

2. В случае проблем с сетью и для обеспечения высокой точности вместе с требованием разгрузки сети необходимо физическое разделение отсчетов замера мощности на уровне АЦП измерительного прибора (они могут быть выполнены с высокой частотой дискретизации) и отсылка результатов суммирования измерений в виде сообщения в систему. Чтобы обеспечить это разделение, измерительное устройство должно быть достаточно «интеллектуальным»: обладать памятью и иметь возможность синхронизировать часы, чтобы установить временную метку. Может показаться, что достаточно иметь момент времени приема сообщения, но это неверно. Для получения высокой точности и обеспечения надежности важно каждое сообщение. Они могут быть потеряны как из-за отказа измерительного устройства, так и из-за проблем с телекоммуникационной сетью. Чтобы устранить проблемы с сетью, устройство может запоминать сообщения и пересылать их, когда сеть восстановит работоспособность. И вот тут-то очень важно, чтобы каждому сообщению была присвоена метка времени: когда оно сгенерировано. Это позволит в итоге правильно подсчитать суммарное энергопотребление в заданный временной интервал.

# Зачем нужно промежуточное хранилище большого объема?



Рис. 2.2. Архитектура системы учета электроэнергии, построенная на основе единого реляционного хранилища данных

Ведь можно периодически очищать сырую таблицу в SQL-хранилище после заполнения данными агрегированной таблицы и хранить только результаты агрегирования.

Действительно, при наличии данных энергопотребления в течение каждого дня недели/месяца/года можно легко подсчитать суммарное энергопотребление за бо' льшие периоды времени. Против такого подхода есть ряд возражений. Например, возможна ситуация, когда из-за проблем с сетью не все устройства отослали свои данные вовремя. И если сырая таблица очистится перед тем, как восстановится работоспособность сети и устройства сбросят свои данные, то последние останутся неучтенными и не будет никакого смысла дополнительно усложнять измерители в виде внутреннего буфера сообщений и синхронизируемых часов. В то же время реализация дополнительной логики, обеспечивающей пересчет агрегированных данных в пакетном режиме при приеме недостающих сообщений, позволит произвести правильный и надежный подсчет потребления даже при ненадежной сети передачи данных. Тут мы сталкиваемся еще с одним аспектом анализа больших данных: **потоковой обработкой**.

# Обработка больших данных: пакетный, **интерактивный** и **поточковый** анализ

Еще с одним аспектом анализа больших данных:  
**поточковая обработка.**

В данном случае необходимо из всего потока сообщений обнаруживать те, чья временная метка меньше, чем текущее время минус самая большая временная задержка, и при их обнаружении запускать внепланово или планировать дополнительно запускать в определенное время (если построение агрегированных таблиц происходит в конкретное время, скажем по ночам) задачу по повторному пересчету.

Для этой цели может служить, например, Apache Storm, Apache Spark Streaming или же Apache Pig.



# Обработка больших данных: пакетный, **интерактивный** и **поточковый** анализ

Следующая причина, побуждающая оставить-таки сырую таблицу без удаления данных, — возможность провести **интерактивный анализ** хранящихся в ней данных.

То есть применить к ним запросы на специальном языке, позволяющем комбинировать, фильтровать, группировать, проводить арифметические операции в режиме реального времени. Это позволяет находить скрытые закономерности в данных, например определять пики энергопотребления, устанавливать корреляцию их с внешними событиями (скажем, с погодой), определять профили пользователей, характеризующиеся оптимальным энергопотреблением. Или для одного пользователя строить типовой профиль применения энергоресурсов и обнаруживать отклонения от него (допустим, утечку электроэнергии, несвоевременное выключение освещения и пр.). Подобные задачи могут решаться с помощью системы интерактивного анализа данных, таких как Spark SQL или Apache Hive, а расширенный интеллектуальный анализ — благодаря применению библиотеки машинного обучения Spark MLlib.

# Свойства технологий Big Data

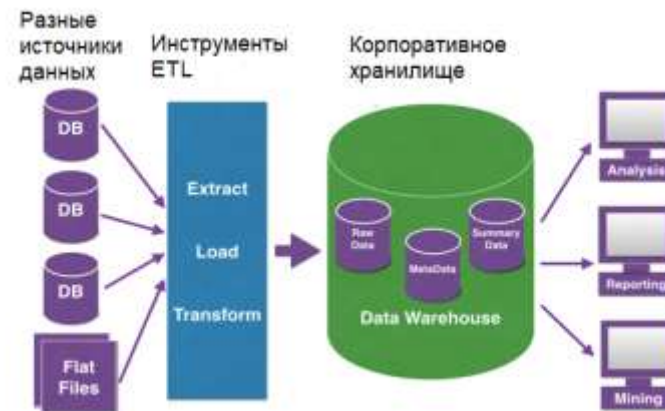
**Ключевой момент всех технологий, работающих с большими данными** — возможность распараллеливания выполняемых задач, областей хранения, памяти и т. д. между группой компьютеров (кластер).

Кроме того, эти технологии должны:

- обладать возможностью линейного масштабирования и наращивания производительности путем добавления новых серверов в кластер. Линейность масштабирования означает пропорциональность производительности/объема хранения количеству компьютеров в кластере;
- иметь специальную файловую систему для надежного хранения и доступа к данным, позволяющую оперировать очень большими объемами данных и до- пускать их репликацию в целях повышения надежности и производительности;
- позволять выполнять запросы к файлам с помощью специального языка запросов или программного интерфейса;
- иметь планировщик, позволяющий распределять эти запросы среди узлов кластера для обеспечения их параллельной работы.

*Всеми этими свойствами обладает наиболее популярный фреймворк Apache Hadoop и компоненты его экосистемы — MapReduce, Hive, Pig, Spark, Storm, Kafka, HBase и др.*

# Модуль 1. Основы построения и работы с системами аналитики больших данных



## КРАТКОЕ СОДЕРЖАНИЕ:

### 1. Архитектура систем аналитики больших данных.

1.1. Облачные технологии. Модели развертывания. Способы создания ресурсов в облаке.

1.2. Безопасность облачных ресурсов

1.3. Большие данные и источники данных. Форматы. Преобразование данных из различных форматов. Режимы обработки больших данных.

### 2. Хранение больших данных в облаке.

2.1. Хранилища общего назначения. Форматы хранения данных. Облачное хранилище Microsoft Azure Storage. Облачное хранилище AWS.

2.2. Реляционные базы данных. Azure SQL. AWS RDS.

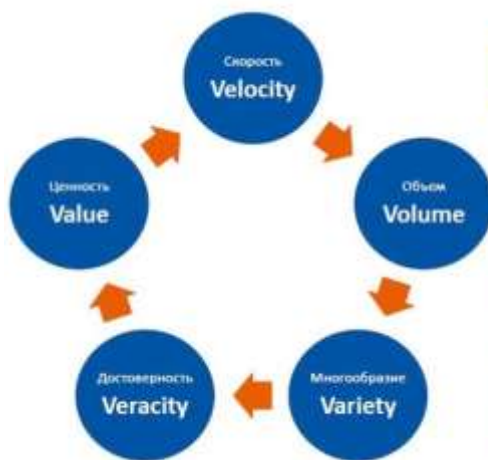
2.3. Нереляционные базы данных. Сервисы нереляционных баз данных от Azure и AWS.

2.4. Корпоративные хранилища данных (DWH). Azure SQL DWH. AWS RedShift.

2.5. Хранилища данных типа Data Lake. Azure Data Lake Store. AWS Data Lake Solutions.

# ЛЕКЦИЯ 4:

## Архитектура облачных систем, оперирующих BigData



Характеристика	Традиционная база данных	База Больших Данных
Объем информации	От гигабайт до терабайт	От петабайт до эксабайт
Способ хранения	Централизованный	Децентрализованный
Структурированность данных	Структурирована	Полуструктурирована или неструктурирована
Модель хранения и обработки данных	Вертикальная модель	Горизонтальная модель
Взаимосвязь данных	Сильная	Слабая

# Общие сведения

- Облачные провайдеры позволяют подключать различные устройства, программные продукты, сервисы (игровые устройства, стационарные устройства IoT, подключенные автомобили, мобильные приложения, веб-серверы и серверы приложений, медиасервисы и пр.) через специальные сервисы концентраторов сообщений и шлюзы устройств «Интернета вещей» (IoT Gateway).
- Концентраторы (AWS Kinesis Stream, Azure Event Hub) обеспечивают однонаправленный прием сообщений извне в облако, а IoT-шлюз (Azure IoT Hub) — двунаправленную коммуникацию с устройствами, то есть возможность обратной отсылки команд устройствам из облака.
- Этот поток может быть обработан сервисами потоковой обработки и анализа.

**К сервисам потокового анализа** относятся сервисы, которые допускают интерактивный анализ потока данных и позволяют создавать аналитические запросы на специальном языке (чаще всего с SQL-подобным синтаксисом), интерактивно их применять и отображать результаты (например, Azure Stream Analytics).

**К сервисам потоковой обработки** относятся те, которые задействуют модули, написанные на компилируемых языках для построения потоковых задач анализа (таких как Apache Storm в HDInsight или AWS EMR) и не допускающие интерактивного использования.

# Сервисы потоковой обработки и анализа



Рис. 3.1. Облачные сервисы, относящиеся к большим данным

# Сервисы потоковой обработки и анализа

- Сообщения, полученные от концентраторов или IoT-шлюзов, могут быть направлены по своим назначениям, в зависимости от внутренних признаков (такая возможность обеспечивается системой маршрутизации сообщений: Azure Event Grid или аналогичной системой в IoT-шлюзе), или целиком направлены в облачное хранилище. Это может быть либо хранилище общего назначения (типа Azure BLOB Storage или AWS S3), либо HDFS-совместимое (Azure Data Lake).
- Кроме того, данные в такое хранилище могут доставляться сервисами копирования и трансформации данных (AWS Glue или Azure DataFactory) или специализированными сторонними программами через предоставляемые этими сервисами API. Источниками данных могут быть внешние реляционные и нереляционные базы данных и файлы.
- После размещения в облачном хранилище информацию можно обработать в пакетном режиме (например, с помощью Hadoop MapReduce в Azure HDInsight или AWS EMR), интерактивном режиме (Azure Data Lake Analytics, AWS Athena) или с применением машинного обучения. Результаты обработки могут быть размещены в реляционном хранилище данных и доступны для средств BI (Microsoft PowerBI или AWS QuickSight).

# Архитектуры традиционных информационных систем

Многие годы информационные системы строились в виде одного крупного монолита с единообразной кодовой базой, которая разворачивается на сервере как единое целое: вместе со всеми подключаемыми модулями, библиотеками и т. д.

Такой монолит (рис. 3.2) отвечал за все (или почти): за взаимодействие с базами данных, обеспечение системы контроля учетных данных, работу периодических сервисов синхронизации, логирования и пр.

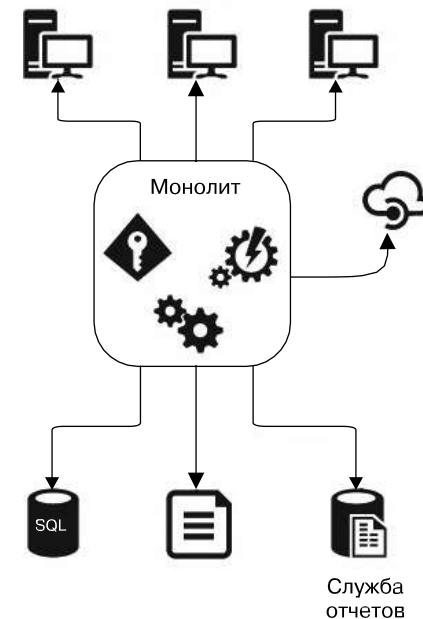


Рис. 3.2. Монолитная архитектура



# Архитектуры традиционных информационных систем

Ниже представлены проблемы, свойственные такой архитектуре:

- Любое изменение в любом компоненте монолита, даже самое незначительное, требует компилирования и разворачивания всей системы, что при большом размере монолита задача весьма небыстрая.
- Монолит очень плохо масштабируется и подвержен существенным проблемам с использованием ресурсов. Действительно, как правило, подобные архитектуры разворачиваются на одном сервере или в виде полных копий на группе серверов. Поэтому любой компонент (или компоненты), который задействует ресурсы сервера (скажем, оперативную память, процессор и т. д.), в значительной степени может затруднить работу всего монолита и потребует увеличения производительности всего сервера, на котором расположена данная архитектура. Компоненты монолита невозможно масштабировать независимо.
- Монолит жестко диктует стек технологий программирования, и обновить его, а также обновить саму архитектуру — значит, по сути, переписать весь код заново. Сюда же можно отнести проблемы с быстрым «старением» монолита, склонностью к обрастанию спагетти-кодом, а также «тупиковой внутренней архитектурой» — это когда невозможно исправить баг или улучшить что-либо, не вызвав появления нового бага или ухудшения системы.

Решить указанные проблемы была призвана **многослойная архитектура** (рис. 3.3). В ней за специфическую задачу отвечает отдельный слой: пользовательского интерфейса (user interface layer, UI layer), бизнес-логики (business logic layer, BL layer) и доступа к данным (data access layer, DAL).

# Многослойная архитектура

Рассмотрим эту архитектуру подробнее. Каждый слой отвечает только за определенную группу задач.

**UI-слой** содержит только веб-серверы, являющиеся источником веб-страниц, или размещает сервисы (REST, SOAP или др.) для взаимодействия с клиентскими приложениями. Кроме того, данный слой принимает запросы от клиентов и транслирует их слою бизнес-логики. Поскольку с клиентом напрямую взаимодействует только UI-слой, то на одном уровне с ним в тесной интеграции находится сервис аутентификации клиентов.

**Слой BL** содержит серверы приложений и отвечает, собственно, за бизнес-логику и интеграцию со сторонними сервисами.

**Слой DAL** обеспечивает программный доступ слоя BL к базе данных и файловому хранилищу.



Рис. 3.3. Многослойная архитектура

# Многослойная архитектура

Подобная архитектура по сравнению с монолитной имеет следующие преимущества:

- Разделение на слои позволяет реализовать в каждом слое наиболее подходящий для него стек технологий. Можно независимо обновлять фреймворки на каждом слое.
- Логическое разделение на слои существенно упрощает процесс разработки и сопровождение всей системы. Распределение команд программистов по слоям и специализация разработчиков (фронтенд, бэкенд), уменьшение объема кода на каждом слое, а также независимый деплой (развертывание) значительно улучшают качество всей системы, делая ее более гибкой и пригодной для сопровождения.
- Возможно независимое масштабирование каждого слоя.

*!!! Наиболее часто подобная архитектура реализуется в виде серверов, расположенных в локальной сети, разбитой на подсети с настроенными фаерволами. Адреса серверов (IP или URL) разных слоев и учетные данные для доступа к ним прописаны в конфигурационных файлах других серверов. В этой архитектуре уже необходимо централизованно хранить учетные данные, иметь серверы DNS, сервис хранения логов и мониторинга, а также сервер для администрирования.*

# Многослойная сервис-ориентированная архитектура

Многослойная архитектура информационных систем в настоящее время чрезвычайно распространена, она более гибкая и удобная по сравнению с монолитной, однако не может решить ряд проблем. В частности, при разрастании проекта до такого масштаба, что **слой VL** становится сопоставимым с монолитом, появляются аналогичные проблемы с масштабированием, производительностью, сопровождением и т. д.

Следующая разновидность многослойной архитектуры — SOA: **сервис-ориентированная архитектура**. Ее квинтэссенцией является архитектура микросервисов (рис. 3.4), суть которой заключается в том, что каждое приложение разбивается на наименьшие самостоятельные модули, и каждый из них отвечает только за один аспект: отсылку писем, загрузку и выгрузку файлов и пр.

Каждый такой сервис можно совершенно независимо развернуть и обновить в любой момент, не затрагивая остальные сервисы. Код подобного сервиса может быть совсем небольшим, и для его сопровождения нужна маленькая команда. Более того, все микросервисы можно писать на разных языках, лучше всего подходящих для решения текущей задачи.

И наконец, каждый микросервис может быть развернут на своем сервере или в кластере серверов, которые могут быть совершенно независимо масштабируемы. Каждый сервис доступен по URL конечной точки и из них, как из элементов конструктора, можно собрать новые системы, а каждый сервис использовать в различных системах.

# Многослойная сервис-ориентированная архитектура

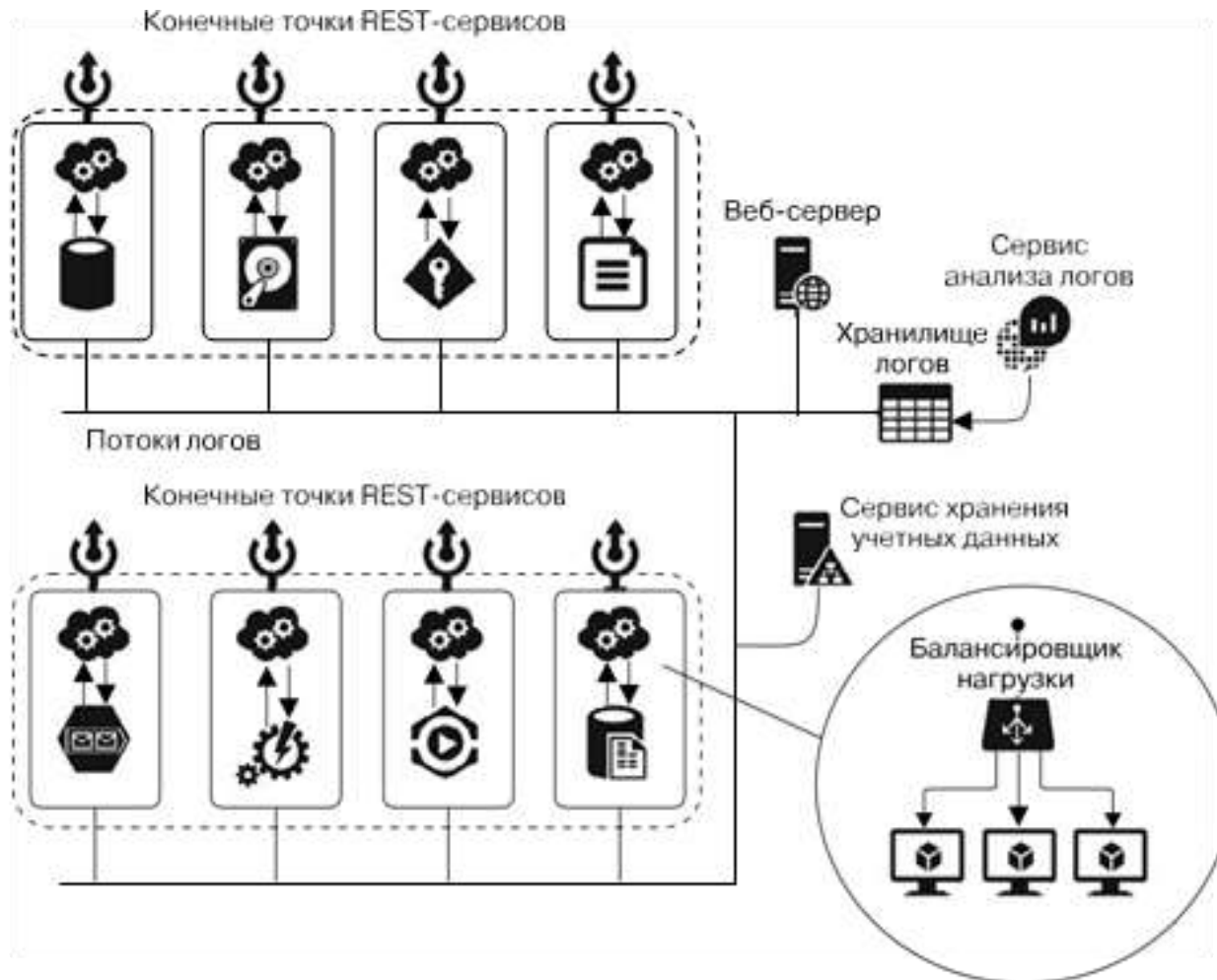


Рис. 3.4. Архитектура микросервисов

# Многослойная сервис-ориентированная архитектура

Работа с микросервисной архитектурой предполагает ряд трудностей:

- Большое количество сервисов, размещенных на многих серверах, означает большое количество разнообразных логов. По сути, вот они, большие данные! Анализ этих данных, определение метрик производительности, узких мест, поиск логов и отображение результатов в виде графиков в режиме, близкому к режиму реального времени, требует не просто сервиса, но целой подсистемы, сопоставимой с основной системой. В качестве примера такой подсистемы можно привести ELK-стек — Elasticsearch (хранение логов и поиск), Logstash (агенты, обеспечивающие доставку логов с серверов в кластер Elasticsearch) и Kibana (интерактивные диаграммы, графики и др.), Splunk.
- Взаимодействие между сервисами происходит преимущественно с помощью REST. А потому каждый сервис должен знать URL всех сервисов, с которыми он может потенциально взаимодействовать.
- Если по какой-то причине запрос не был обработан (например, сервис был недоступен), то для его повторения необходимо организовать логику повтора в рамках самого кода.

Синхронизировать работу микросервисов позволит отдельный сервис, обеспечивающий надежную доставку сообщений, а также предоставляющий малое количество конечных точек.

**Event Driven Design** — архитектура, основанной на обмене сообщениями.

# Архитектуры, построенные на базе микросервисов

Архитектуры, построенные на базе микросервисов, каждый из которых можно разместить в кластере серверов, дополненные сервисами обмена сообщениями (брокерами сообщений — message brokers), могут быть чрезвычайно масштабируемыми. Для этого нужно, чтобы сами сервисы обмена сообщениями были масштабируемыми. Брокеры, как правило, имеют архитектуру с одним или несколькими головными узлами, отвечающими за управление кластером, и исполнительными узлами, на которых выполняются необходимые вычисления и хранятся данные (сообщения) (рис. 3.5).

Чтобы обеспечить надежность кластера, данные могут быть реплицированы между исполнительными узлами.



Подобные структуры весьма сложны: кластеры серверов, балансировщики нагрузки, системы управления конфигурацией, логирования и пр. Преимущества облачных платформ: все подобные системы представляются как сервисы — AWS ECS и Azure Service Fabric.

Рис. 3.5. Общая структура масштабируемой системы, построенной на основе кластера

# Архитектуры, построенные на базе микросервисов

- В облачных средах, наряду с сервисами PaaS, отвечающими за обработку больших данных, Azure и AWS предоставляют сервисы, занимающие промежуточное положение между IaaS и PaaS.
- Эти сервисы позволяют применять наиболее популярные фреймворки, работающие с большими данными, — Apache Spark, Storm, Kafka, HBase, Storm и ряд других.
- К таким сервисам относятся AWS EMR и Azure HDInsight. Они берут на себя все трудности, связанные с настройкой и конфигурированием сервисов в кластерах, а конечному пользователю предоставляют готовый и настроенный сервис.
- Сервисы AWS EMR и Azure HDInsight занимают промежуточное положение между IaaS и PaaS, сочетая все возможности сервисов Apache с простотой PaaS. Эти сервисы представляют собой надстройки над набором виртуальных машин. Последние создаются, запускаются, останавливаются и удаляются только на время выполнения задания в сервисе.
- Сервисы AWS EMR и Azure HDInsight очень хорошо интегрируются с сервисами копирования и трансформации данных (скажем, AWS Data Pipeline), составляя, по сути, вычислительные ресурсы для заданий ETL. При создании кластеров с портала для HDInsight и AWS EMR указываются размеры виртуальных машин и сценарий, который должен выполняться при запуске сервиса.



# Бессерверные архитектуры

Архитектура, основанная на обмене сообщениями, характеризуется тем, что все компоненты системы взаимодействуют через малые порции сообщений, требующие вычислительных ресурсов только непосредственно в момент обработки сообщений. Все остальное время сервисы заняты только тем, что прослушивают конечные точки сервисов обмена сообщениями.

Во время прослушивания серверы или виртуальные машины не загружены. Но в облачных средах принимается плата за включенную виртуальную машину вне зависимости от того, насколько интенсивно она используется.

Таким образом, если поток сообщений не слишком сильный, то сервисы работают вхолостую. Но зачастую необходимо выполнить программу однократно или в непредсказуемые моменты времени в ответ на поступившее сообщение, для чего использовать виртуальную машину нецелесообразно.

Для такого случая облачные провайдеры обеспечивают возможность выполнения кода по требованию без предоставления серверов — **бессерверные сервисы (serverless)**.

# Бессерверные архитектуры

- ❖ Подобный сервис от Microsoft называется Microsoft Azure Function, а от AWS — AWS Lambda. Последний является классическим вариантом. Этот сервис позволяет размещать исполняемый код и запускать его внешним событием, которое может прийти от сервисов SNS, SQS, Kinesis Stream, S3, API Gateway и др.
- ❖ Сам сервис имеет различные уровни производительности процессора и памяти, которые используются лишь при исполнении кода, и плата начисляется только в этот момент. Безусловно, внутри сервиса AWS Lambda содержится цикл, ожидающий прихода сообщения, но вычислительные ресурсы задействуются (и, соответственно, за их применение начисляются деньги) только для выполнения кода пользователя.

Концепция **serverless** очень удобна для построения приложений с малым, средним и умеренно большим потоком сообщений.

Главные преимущества **serverless** таковы:

# Бессерверные архитектуры

Концепция **serverless** очень удобна для построения приложений с малым, средним и умеренно большим потоком сообщений.

## Главные преимущества **serverless** таковы:

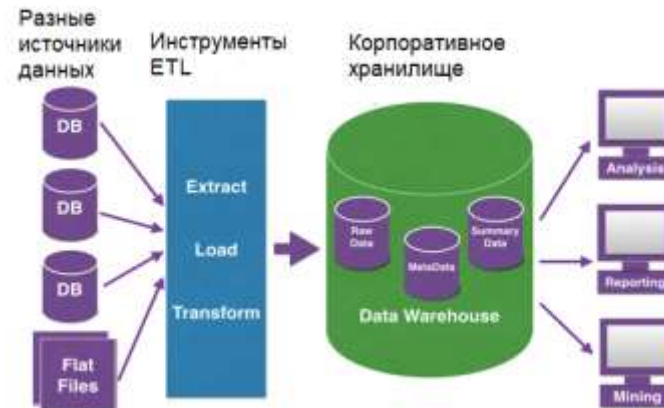
- ❖ более дешевая среда исполнения кода, чем виртуальные машины, при небольших, средних и умеренно больших потоках;
- ❖ более высокая надежность сервиса, чем в случае одиночной виртуальной машины, — SLA на уровне 99,9 %;
- ❖ простота интеграции с другими облачными сервисами, не требующая внесения изменения в код;
- ❖ простота развертывания кода и конфигурирования — необходимо только добавить код напрямую или через ZIP-файл и выбрать уровень производительности.

## Однако подобным сервисам присущи и недостатки:

- ограниченная масштабируемость;
- ограниченное время выполнения кода;

Но вместе с тем **serverless**-сервисы крайне удобны для построения архитектур, **основанных на событиях (event driven design)**. AWS Lambda наиболее удобно использовать для инициирования выполнения задач копирования/трансформации данных, которые должны последовать после наступления какого-либо события.

# Модуль 1. Основы построения и работы с системами аналитики больших данных



## КРАТКОЕ СОДЕРЖАНИЕ:

### 1. Архитектура систем аналитики больших данных.

1.1. Облачные технологии. Модели развертывания. Способы создания ресурсов в облаке.

1.2. Безопасность облачных ресурсов

1.3. Большие данные и источники данных. Форматы. Преобразование данных из различных форматов. Режимы обработки больших данных.

### 2. Хранение больших данных в облаке.

2.1. Хранилища общего назначения. Форматы хранения данных. Облачное хранилище Microsoft Azure Storage. Облачное хранилище AWS.

2.2. Реляционные базы данных. Azure SQL. AWS RDS.

2.3. Нереляционные базы данных. Сервисы нереляционных баз данных от Azure и AWS.

2.4. Корпоративные хранилища данных (DWH). Azure SQL DWH. AWS RedShift.

2.5. Хранилища данных типа Data Lake. Azure Data Lake Store. AWS Data Lake Solutions.

# ЛЕКЦИЯ 5:

## Хранение больших данных в облаке



# Хранение больших данных в облаке

Любые данные, как большие, так и маленькие, хранятся в файлах. Это могут быть двоичные файлы или текстовые файлы того или иного формата (CSV, TXT, JSON и пр.). Любая база данных любого типа в конечном итоге хранит данные в виде специальных файлов на дисках, но терминологически и фактически базы данных (database) и хранилища данных (data storage) предоставляют различные концепции хранения информации и доступа к ней. **Базы данных (БД)** должны обеспечивать хранение информации, доступ к выборке произвольной части информации и ее обновление.

Важнейшим элементом базы является **система управления базой данных (СУБД)** — специальное программное обеспечение, служащее для обработки запросов к БД и выполнения различных административных операций (мониторинг, резервное копирование, управление доступом и др.). СУБД отделяет физическое хранение данных от их логического представления.

Традиционно все базы делят на два крупных вида: **реляционные** (иначе называются SQL-базы) и **нереляционные** (NoSQL).

- В **реляционных** логически информация представляется в виде набора таблиц, связанных между собой, то есть находящихся в определенных отношениях (отсюда и название, содержащее корень relatio — «отношения»).
- **Нереляционные базы** (типа «ключ — значение», граф, семейство колонок, документоориентированные) очень разнообразны и объединены лишь тем, что модель представления данных в них отличается от реляционной. Для программного доступа к СУБД необходимо знать адрес сервера, на котором она размещена, учетные данные пользователя, имеющего соответствующие права, и программную библиотеку, поддерживающую протокол обмена данными управляющими сигналами с СУБД.

# Хранение больших данных в облаке

В отличие от БД **хранилище данных** содержит файлы и позволяет получать к ним доступ для загрузки, обновления и удаления файлов, в которых хранится информация в физически и логически неотделимом от файла виде.

**Хранилища** — универсальное место хранения для файлов любых типов: текстовых и бинарных.

Классически хранилища данных в виде сетей хранилищ данных (SAN), которые представляли собой группу серверов, имеющих высокоскоростные диски (SSD, зачастую объединенные в RAID-массивы), объединенных в сеть.

*Для доступа к файлам извне необходима поддержка протоколов сетевых файловых систем NFS, CIFS, SMB и др., а также прямая ссылка, чтобы файлы можно было скачивать.*

В отличие от СУБД, где физическое хранение информации отделено от ее логического представления, *при хранении данных в файловом хранилище физическая структура файла полностью повторяет их логическую структуру.* Например, в файлах логов информация представлена в виде таблицы, каждая строка которой состоит из отдельных столбцов. Чтобы получить доступ к ней, файл можно открыть с помощью любого текстового редактора или стандартных программных библиотек ввода-вывода. В этой части книги представлены различные сервисы хранения больших данных в облаках. Сервисы для обработки данных будут описаны в последующих главах.

# Хранилища общего назначения

## Платформа Windows Azure

### Языки программирования общего назначения





# Хранилища общего назначения

**Облачные хранилища файлов** — это сервисы, которые обеспечивают хранение файлов, предоставляют доступ к ним с помощью REST API, позволяют скачивание по прямой ссылке (постоянной или с конечным сроком действия) и в некоторых случаях предоставляют доступ к файловой системе по протоколу SMB.

Хранить файлы, содержащие большие данные, в облачном хранилище очень дешево и в ряде случаев удобнее, чем в других хранилищах. *Наиболее типовой пример размещения данных — это хранение файлов логов приложения, которые копируются туда периодически из сервера-источника или создаются и заполняются специальной программой-клиентом, размещенной на сервере-источнике логов (рис. 4.1).*

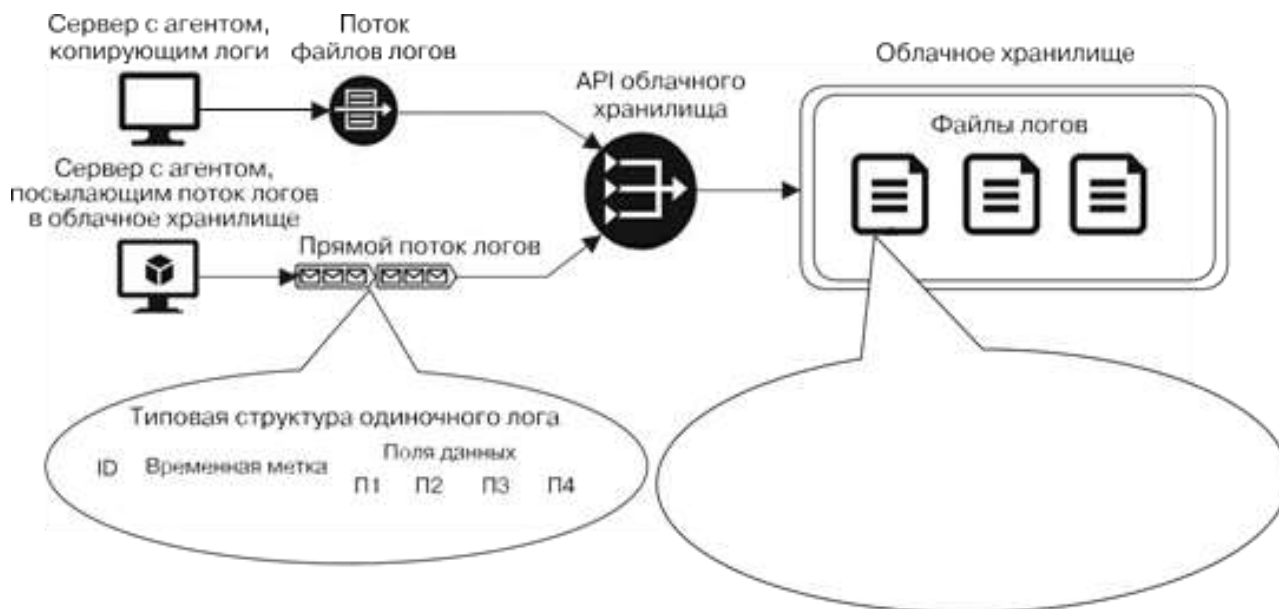


Рис. 4.1. Структура потока логов серверов в облачное хранилище

# Форматы хранения данных

Кроме того, в облачных файловых хранилищах могут размещаться виртуальные жесткие диски (virtual hard drive, VHD) облачных виртуальных машин, на которых, в свою очередь, тоже можно размещать файлы.

Но в этом случае ответственность за доступность информации ложится на владельца виртуальных машин. Рассмотрим подробнее различные форматы хранения больших данных.

Наиболее типовой случай использования текстовых файлов для хранения больших данных — *хранение логов приложений в том или ином текстовом формате*. Такие файлы могут иметь расширения .log, .txt, .csv и др. Общее у этих форматов то, что логи в них, по сути, хранятся в виде таблицы (отсюда и название — табличный формат), каждая строка которой — запись конкретного события. Строка состоит из набора столбцов, разделенных некими символами. Это могут быть пробелы, символы табуляции, двоеточие, точка с запятой и т. д.

Подобные файлы можно открыть с помощью любого текстового редактора (если они не очень велики), и для их анализа (например, поиска запросов с 500 ошибками) подойдут стандартные утилиты командной оболочки (grep, awk и пр.) или же специализированные сервисы аналитики (к этому вопросу мы еще вернемся). Кроме того, табличная структура подобного файла очень удобна для импорта в реляционную или нереляционную СУБД табличного типа.

# Форматы хранения данных

**Особенность таких файлов** — линейная структура со строго одинаковым количеством столбцов в каждой записи и в общем случае линейное время доступа к записям.

Возможны ситуации, когда разные строки могут содержать различное количество столбцов.

Например, запись в лог-файле, отражающая ошибку приложения и трассировку стека, может включать гораздо меньше строк, чем запись, отражающая какое-либо событие, характеризующее нормальную работу приложения. В общем же случае записи логов содержат как минимум временную метку. Остальные поля (уникальный идентификатор, URL, сообщение ошибки и пр.) могут и отсутствовать.

*Возникает вопрос: полезна ли запись с одним полем (временной меткой)?*

Да. Скажем, необходимо проанализировать посещаемость сайта во времени. Просуммировать запросы для заданного временного интервала (например, 15 минут) поможет именно временная метка.

# Форматы хранения данных

Табличные файлы имеют и ряд неудобств в использовании.

- Во-первых, для доступа к элементу информации необходимо знать номер строки и номер столбца. Нет универсального стандарта или языка запросов (за исключением SQL-подобного языка специализированных сервисов аналитики, но об этом позже), что весьма затрудняет анализ логов.
- Во-вторых, в таких форматах очень просто добавить новую запись в конец файла, но крайне трудно и затратно вставить, удалить или изменить произвольную строку. Как правило, все программные библиотеки ввода-вывода позволяют добавлять строку в конец файла с помощью стандартных средств, но для произвольной манипуляции данными программисту нужно будет создать отдельную логику в коде, и эта логика весьма непроста.

# Форматы хранения данных

Помимо текстового файла с табличной структурой, широко распространено хранение информации в более структурированных форматах JSON, XML.

Их преимущества в том, что они очень удобны для сериализации/десериализации информации в объектно-ориентированном виде в коде программы. Кроме того, для обоих форматов существуют стандарты выполнения запросов на выборку данных (*для XML — XPath, XQuery, для JSON — JSONPath, JSONQuery*).

## Кратко рассмотрим эти форматы.

JSON представляет собой формат, при котором данные хранятся в текстовом виде как объект JavaScript (аббревиатура образована от JavaScript Object Notation). JSON-файл выглядит так (рис. 4.2).

# Хранение информации в структурированных форматах JSON, XML

```
"PlayerId" : "1", // partition key
"TableId": "1", // partition key
"StepId" : "guid", // row key
"Pot": "700", // money on table
"Players" :
[
  {
    "PlayerId" : "1",
    "Action" : "", // bet, all-in, call, raise, fold, check
    "Rate": "10",
    "Amount": "500", // bankroll for current game
    "DoneAction": "true",
    "PlayerCards":
    [
      "JH", "KC"
    ]
  },
  {
    "PlayerId" : "2",
    "Action" : "", // bet, all-in, call, raise, fold, check
    "Rate": "10",
    "Amount": "500", // bankroll for current game
    "DoneAction": "false",
    "PlayerCards":
    [
      "JC", "KH"
    ]
  },
  {
    ...
  },
  {
    ...
  }
],
"OpenCards": [
  "7D", "10S", "3C", "", "9D"
]
```

На данном рисунке приведен вариант организации информации в формате JSON для нашего сквозного примера.

В основе этого документа (как и любого документа JSON) лежит объект, который состоит из набора «ключ — значение».

В качестве ключа выступают текстовые величины (PlayerId, Players, StepId и др.).

В качестве значений допускаются следующие типы:

- Атомарный тип
- Массив
- Объект

Рис. 4.2. Пример организации информации с помощью формата JSON

# Хранение информации в структурированных форматах JSON, XML

- *Атомарный тип* (для ключей PlayerId, StepId) — величина, состоящая из одного конкретного значения: строки, числа, временно́й метки.
- *Массив* (например Players) — группы величин одного типа. (Это не совсем точное определение, в общем случае все элементы массива могут быть совершенно разного типа, но, как правило, коллекции объектов, сериализуемые в JSON в реальных программах, будут иметь одинаковый тип.) В качестве типов элементов массива могут выступать атомарные типы, другие массивы или другие объекты. Массивы обозначаются прямоугольными скобками — [ ], а элементы в массиве разделяются запятыми, допустим: [ "a", "c", "d" ]. Доступ к элементу происходит по числовому индексу, например [0].
- *Объект* — коллекция «ключ — значение». Обозначается фигурными скобками { } и имеет синтаксис {"key": value}, где value может иметь атомарный тип, тип массива и объекта. По сути, эта коллекция является ассоциативным массивом, в котором для доступа к значению необходимо использовать строковый ключ.

Стоит заметить, что документ JSON может содержать в качестве корневого элемента не объект, а массив, например: [ { "key": 1 }, { "key": 2 }, { "key": 3 } ]. Это допустимо.

# Хранение информации в структурированных форматах JSON, XML

Преимущество этого формата текстового файла — возможность описания структур произвольной глубины вложенности (объект внутри объекта, коллекция внутри объекта и пр.), что невозможно в случае табличного представления.

Кроме того, существенно упрощается сериализация и десериализация объектов, особенно из JavaScript-приложения. Обработка JSON-файлов в специализированных СУБД (DocumentDB) описана в книге ниже, а некоторые реляционные базы данных (например, PostgreSQL) широко поддерживают этот формат.

Очень **важным преимуществом формата JSON** является то, что он позволяет строить информационные системы с помощью одной и той же технологии на всех уровнях: начиная с документоориентированной БД, хранящей данные в формате JSON (Azure DocumentDB, MongoDB), бэкенда на основе диалекта JavaScript (например, Node.js) и заканчивая фронтендом на основе того или иного фрейм-ворка JavaScript (например, ReactJS, Angular).

Это уникальная возможность, реализованная сейчас только в рамках JavaScript/JSON (например, популярен фреймворк MEAN). К недостаткам JSON можно отнести его «многосимвольность»: для группировки данных используются символы (запятые, скобки, кавычки), которые сами по себе не несут информации.

Избавиться от части избыточных элементов синтаксиса поможет формат YAML, но пока что он используется преимущественно в качестве шаблона конфигурационных файлов (в системах Ansible, CloudFormation и др.), а не для хранения данных. Кроме того, сериализация и десериализация YAML не так широко поддерживается программными библиотеками ввода-вывода.

Выборка значений из документа JSON происходит с первоначальной десериализацией его в объект в программном коде, что наиболее удобно и естественно происходит в языке на основе JavaScript.



# Хранение информации в структурированных форматах JSON, XML

Следующий популярный формат хранения данных в текстовых файлах — **XML (eXtensible Markup Language — расширяемый язык разметки)**.

Традиционно он использовался для разметки (markup), то есть структурирования текстовых документов. Термин «язык» (language) говорит о том, что XML содержит строгий набор синтаксических правил, на основании которых можно построить конкретное расширение (extension) этого языка. Рассмотрим, что это значит.

В общем случае у XML есть два основных компонента: теги и атрибуты (см. выше рис. 4.1 — пример лога). Тег — синтаксический элемент, ограничивающий конкретную порцию информации:  
<ИмяТега>Информация в текстовом виде<ИмяТега/>.

Любой тег начинается открывающим (<) и закрывающим (>) символами. Информация, представленная в текстовом виде, расположена между тегами, последний из которых состоит из двух символов (/>). Возможны и вложенные структуры тегов и коллекции последних. Атрибуты относятся к конкретному тегу и представляют собой коллекцию «ключ — значение».

На рис. 4.3 приведена структура XML-документа, представляющего информацию, показанную на рис. 4.2.

# Хранение информации в структурированных форматах JSON, XML

```
<<GameEvent>
  <<CurrentPlayer>
    <Player "ID" = "1" "TableID" = "1" "stepId"="guid" "Pot"="700"></Player>
  </CurrentPlayer>
  <<OtherPlayers>
    <Player "ID" = "1" "TableID" = "1" "stepId"="guid" "Pot"="700" "DoneAction"="true" >
      <<PersonalCards>
        <Card>7D</Card>
        <Card>10S</Card>
      </PersonalCards>
    </Player>
    <Player "ID" = "2" "TableID" = "1" "stepId"="guid" "Pot"="500" "DoneAction"="false" > ... </Player>
    <Player "ID" = "3" "TableID" = "1" "stepId"="guid" "Pot"="700" "DoneAction"="false" > ... </Player>
    <Player "ID" = "1" "TableID" = "1" "stepId"="guid" "Pot"="700" "DoneAction"="false" > ... </Player>
  </OtherPlayers>
  <<OpenCards>
    <Card>7D</Card>
    <Card>10S</Card>
    <Card>3C</Card>
    <Card></Card>
    <Card>9D</Card>
  </OpenCards>
</GameEvent>
```

Рис. 4.3. Представление информации в формате XML

Согласно стандарту XML для тега с конкретным именем существует строго определенный набор атрибутов, причем каждый тег должен содержать конкретный набор атрибутов или не включать их вовсе. Возможно также построение коллекций — см., например, теги <OpenCards> или <OtherPlayers> на рис. 4.3.

Итак, язык XML состоит из общих правил построения синтаксиса, а расширение данного языка представляет собой конкретный набор вложенных тегов и соответствующих им атрибутов, который обеспечивает упорядоченное представление конкретной структурированной информации.

Существует стандартизированный способ преобразования информации из одного XML в другой с помощью XSLT — eXtensible Stylesheet Language Transformation. Это тоже XML, элементами которого является не информация, а правила, по которым она из одного типа XML преобразуется в другой.

Кроме того, есть XSD (XML Schema Definition) — синтаксис, описывающий схему, то есть структуру документа.

В отличие от JSON XML гораздо более строг и не допускает произвольной вложенности и комбинирования элементов. Например, для одного тега не допускаются разные наборы элементов.

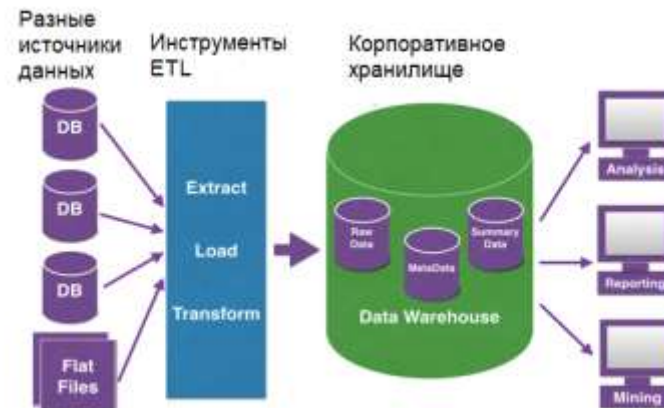
# Хранение информации в структурированных форматах JSON, XML

**Ниже представлены достоинства XML как средства хранения информации:**

- строгость синтаксиса существенно упрощает построение систем сериализации/десериализации;
- этот формат широко поддерживается различными БД как встроенный тип данных;
- можно выполнить простое прямое преобразование в другие языки, в том числе в языки разметки графических элементов (например, *HTML*);
- существуют специальные расширения языка, описывающие различного рода преобразования и трансформации (*XSLT*);
- имеется стандартный способ построения запроса к элементу или выборки элементов (*XPath, XQuery*).

**К недостатком данного формата** можно отнести то, что он гораздо более «многословен», чем JSON, поскольку содержит больше чисто синтаксических символов.

# Модуль 1. Основы построения и работы с системами аналитики больших данных



## КРАТКОЕ СОДЕРЖАНИЕ:

### 1. Архитектура систем аналитики больших данных.

1.1. Облачные технологии. Модели развертывания. Способы создания ресурсов в облаке.

1.2. Безопасность облачных ресурсов

1.3. Большие данные и источники данных. Форматы. Преобразование данных из различных форматов. Режимы обработки больших данных.

### 2. Хранение больших данных в облаке.

2.1. Хранилища общего назначения. Форматы хранения данных. Облачное хранилище Microsoft Azure Storage. Облачное хранилище AWS.

2.2. Реляционные базы данных. Azure SQL. AWS RDS.

2.3. Нереляционные базы данных. Сервисы нереляционных баз данных от Azure и AWS.


2.4. Корпоративные хранилища данных (DWH). Azure SQL DWH. AWS RedShift.

2.5. Хранилища данных типа Data Lake. Azure Data Lake Store. AWS Data Lake Solutions.

# ЛЕКЦИЯ 6:

## Хранение больших данных в облаке Облачное хранилище Microsoft Azure Storage

Windows Azure  
*Облачная Операционная Система*



Windows Azure™

Вычисления      Хранилище      Виртуальные сети

Windows Azure: Хранилище

<p>Масштабированное хранилище в облаке</p> <ul style="list-style-type: none"><li>• 100 TB на аккаунт</li><li>• Автоматически изменяемое в соответствии с различными вариантами запросов на обработку или использование данных</li></ul>	<p>Доступное через RESTful Web services</p> <ul style="list-style-type: none"><li>• Доступ из Windows Azure Приложений</li><li>• Доступ из произвольного места в internet</li><li>• Поддержка .NET Client Library</li></ul>	<p>Различные типы хранилища</p> <ul style="list-style-type: none"><li>• Tables</li><li>• Blobs</li><li>• Queues</li><li>• Drives</li></ul>
---	---	--

# Облачное хранилище

## Microsoft Azure Storage

Рассмотрим, как реализовано облачное хранилище, на примере Microsoft Azure Storage.

Оно состоит из четырех сервисов:

BLOB Storage, Queue Storage, Table Storage и File Storage (рис. 4.4).

Непосредственно хранение информации осуществляется в сервисах BLOB, Table и File Storage.

Queue Storage — это облачный сервис обмена сообщениями и синхронизации распределенных приложений. Сервис Table Storage — база данных NoSQL типа «ключ — значение».

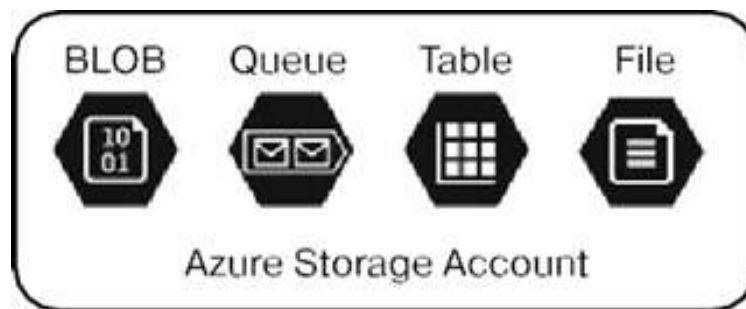


Рис. 4.4. Виды сервисов Azure Storage Account

# Облачное хранилище Microsoft Azure Storage

Рассмотрим подробнее, как создавать *Azure Storage Account* и управлять с веб-портала.

Прежде всего необходимо нажать ссылку добавления новых ресурсов Azure, расположенную в левом верхнем углу, — **+ New**.

Затем в открывшемся окне (рис. 4.5) выбрать последовательно *Storage* — *Storage account*. После нажатия ссылки *Storage account* откроется форма настройки *Storage Account* (рис. 4.6).

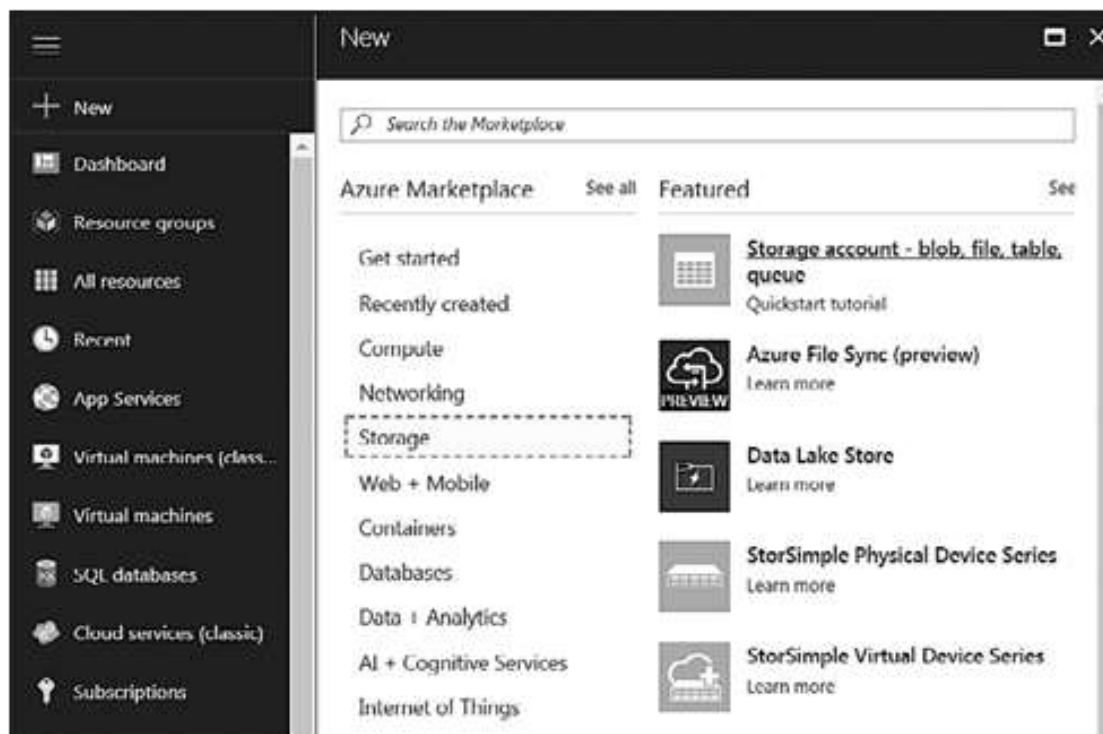


Рис. 4.5. Добавление нового Storage account через веб-портал

# Облачное хранилище Microsoft Azure Storage

После нажатия ссылки Storage account откроется форма настройки Storage Account (рис. 4.6).

The image shows two side-by-side screenshots of the 'Create storage account' form in the Azure portal. The left screenshot shows the form with a vertical scrollbar on the right side, indicating it is scrollable. The right screenshot shows the same form with the scrollbar moved down, revealing more configuration options.

**Left Screenshot (Top):**

- Account kind: Storage (general purpose v1)
- Performance: Standard
- Replication: Locally-redundant storage (LRS)
- Secure transfer required: Disabled
- Subscription: Pay-As-You-Go
- Resource group: PockerRumExample

**Right Screenshot (Bottom):**

- Account kind: Storage (general purpose v1)
- Performance: Standard
- Replication: Locally-redundant storage (LRS)
- Secure transfer required: Disabled
- Subscription: Pay-As-You-Go
- Resource group: PockerRumExample
- Location: Central US
- Virtual networks: Disabled

Both screenshots include a 'Pin to dashboard' checkbox and a 'Create' button at the bottom.

Рис. 4.6. Форма настройки Storage Account (показана со сдвигом ползунка)



# Облачное хранилище

## Microsoft Azure Storage

После нажатия ссылки Storage account откроется форма настройки Storage Account (рис. 4.6).

[В ней доступны следующие конфигурации:](#)

- Имя аккаунта (поле Name) будет частью URL аккаунта <Name>.core.windows.net, а потому правила наименования ресурсов точно такие же, как и в случае именования ресурсов, доступных по URL.
- Тип модели развертывания ресурсов (deployment model) — выбираем Resource Manager, чтобы иметь возможность добавить аккаунт к общей ресурсной группе PockerRunExample.
- Тип Storage (Account kind) — Storage (general purpose v1). Помимо этого, доступны типы Storage (general purpose v2) и BLOB.
- Выбираем уровень производительности (Performance) — Standard. В зависимости от выбранного уровня производительность операций чтения-записи будет отличаться. Наиболее высоким уровнем будет обладать комбинация Account kind = BLOB, Performance = Premium. Для premium-уровня в качестве физических устройств хранения выступают SSD-диски.
- В качестве режима репликации (Replication) выбираем Locally-Redundant storage (LRS). Эта опция означает, что информация, хранящаяся в Storage Account, физически реплицируется три раза, но в пределах одного дата-центра.

Помимо LRS, доступны различные режимы географической репликации в разных дата-центрах в пределах одной зоны (ZRS) или среди нескольких различных зон (GRS и ReadOnly-GRS — репликация по различным географическим зонам с репликой, доступной только для чтения). Все эти режимы различаются по стоимости и уровню надежности и доступности, что позволяет реализовать облачные хранилища, отвечающие различным наборам требований.

После создания Storage Account доступна следующая панель мониторинга и управления (рис. 4.7).

# Облачное хранилище Microsoft Azure Storage

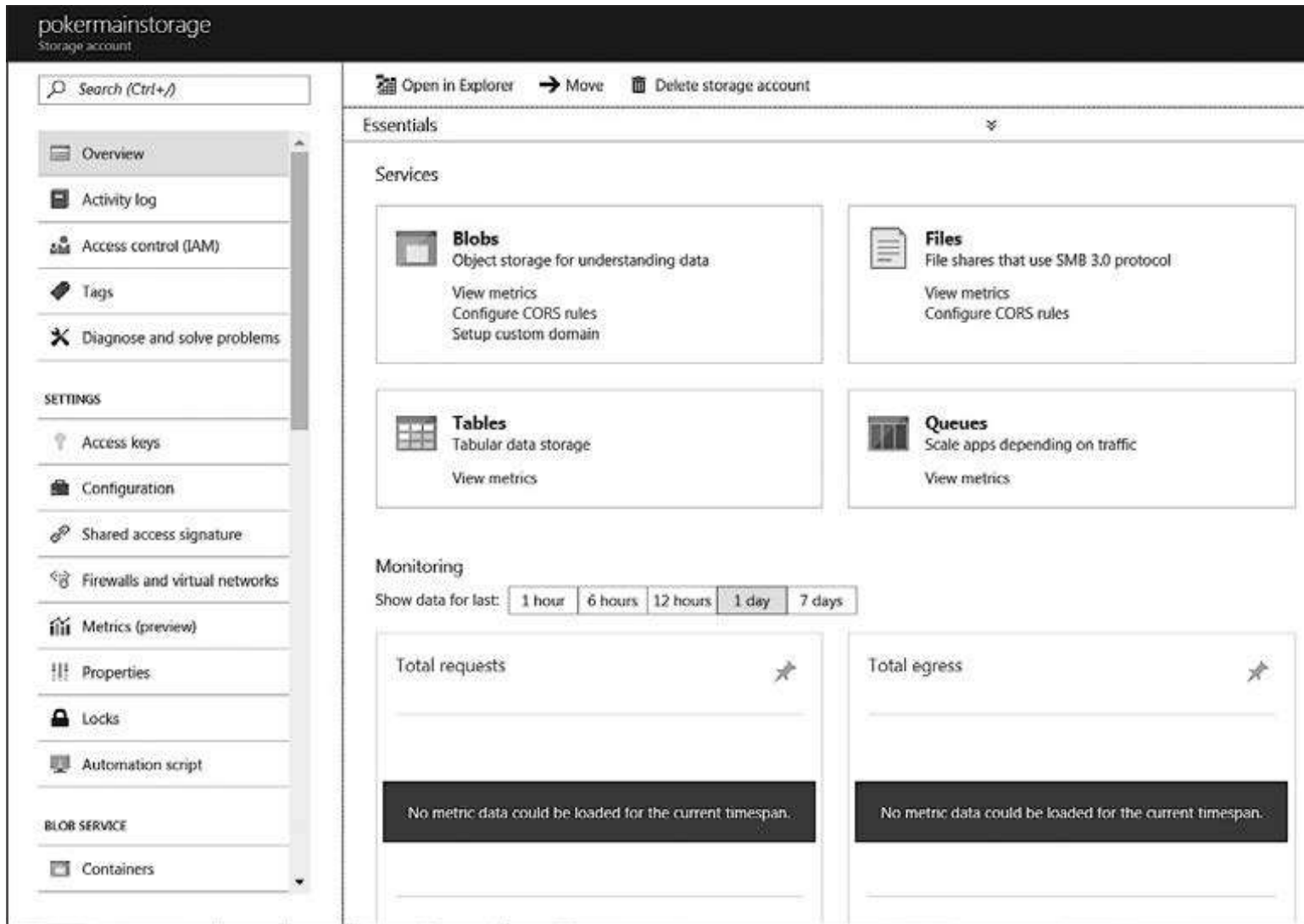


Рис. 4.7. Общая панель мониторинга и управления Storage Account

# Облачное хранилище

## Microsoft Azure Storage

На этой панели доступны все четыре сервиса, входящие в состав Storage Account: Blob, File, Table и Queue.

Кратко рассмотрим возможности конфигурирования Storage Account в целом. Прежде всего, доступен File Explorer — бесплатная программа от Microsoft, позволяющая просматривать содержимое всех сервисов всех аккаунтов хранения.

Ее внешний вид (рис. 4.8). На рисунке можно увидеть результат выборки программы из хранилища Table Storage (таблица events), которая содержит тестовые данные телеметрии метеостанции, полученные путем приема сообщений через сервисы концентратора EventHub и потоковой аналитики Stream Analytics Job.

Чтобы обеспечить доступ к аккаунту извне, необходимо знать ключи доступа и URL, которые доступны на вкладке Access Keys (рис. 4.9).

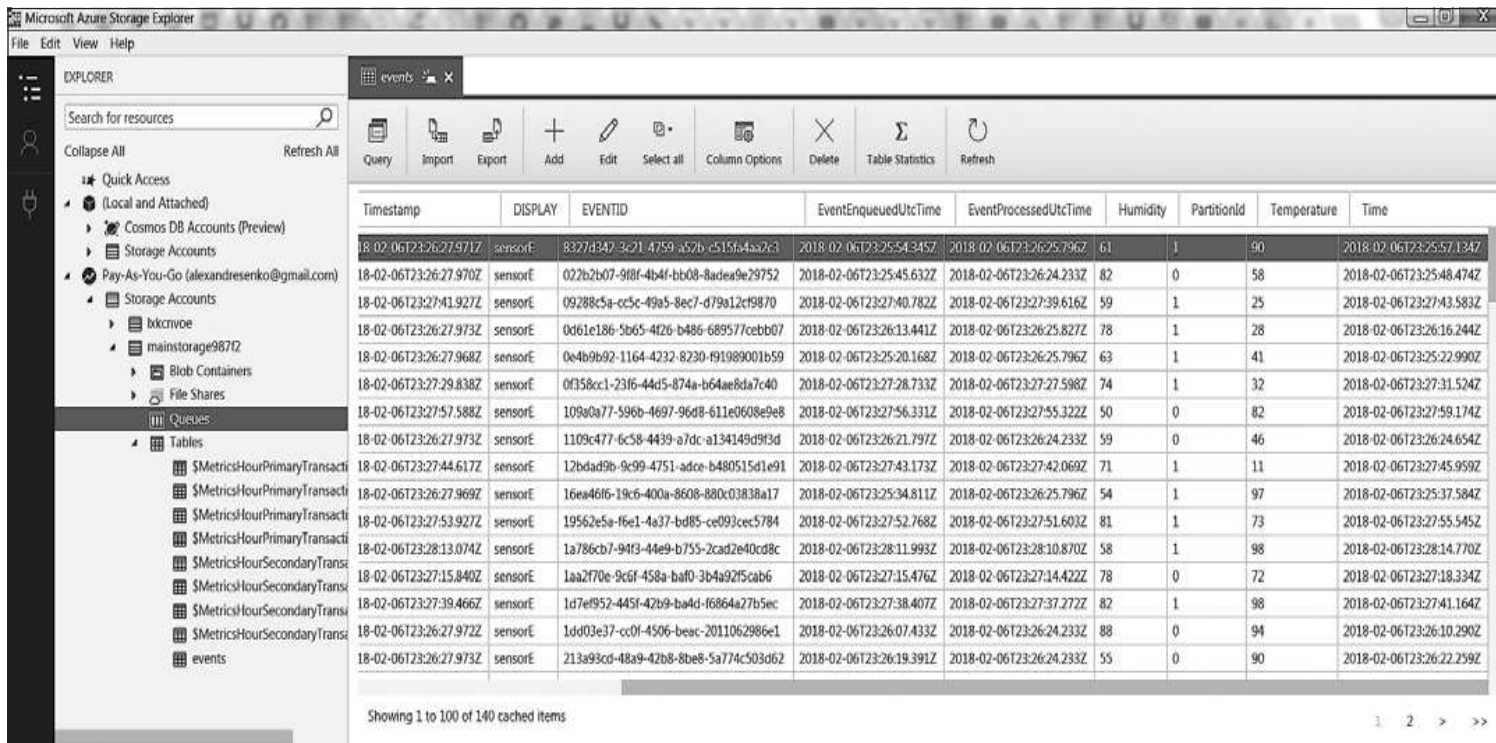


Рис. 4.8. Внешний вид программы File Explorer от Microsoft

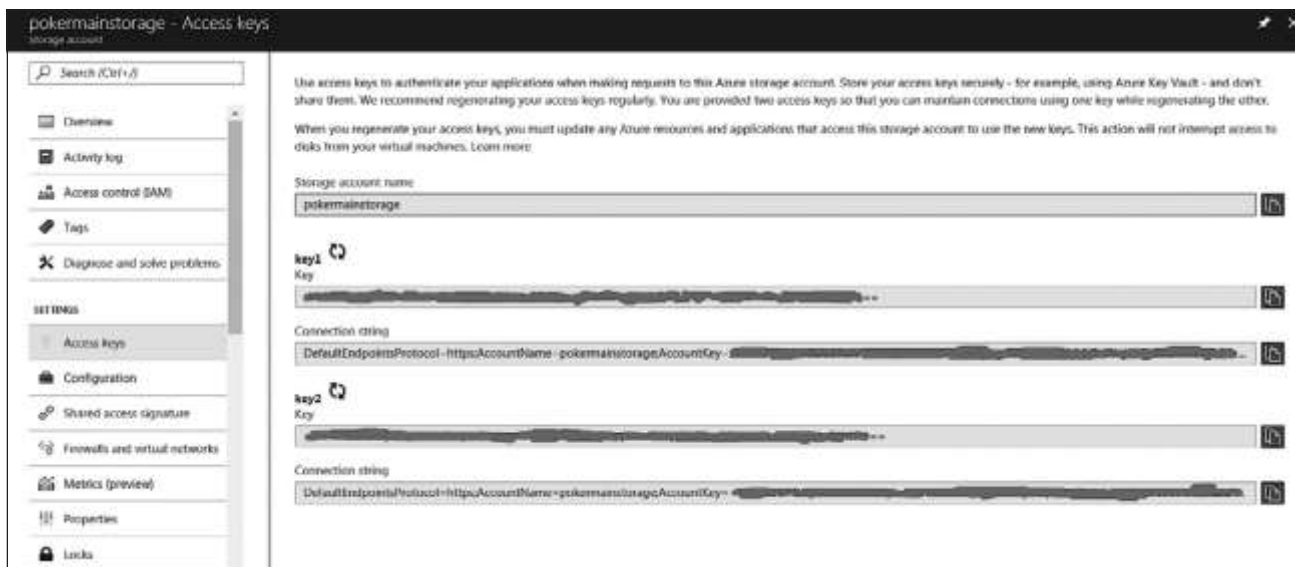


Рис. 4.9. Ключи доступа и строки подключения к Storage Account

Облачное хранилище  
Microsoft Azure Storage

# Облачное хранилище Microsoft Azure Storage

Две пары ключей или строк подключения нужны для обеспечения «бесшовного» обновления ключей. Для этого первоначально используется ключ key1, затем клиенты переключаются на ключ key2, а ключ key1 обновляется. После этого клиенты переключаются на новый ключ key1, а ключ key2 обновляется.

Страница конфигурирования аккаунта в целом показана на рис. 4.10. Она позволяет менять тип аккаунта (Account kind), уровень производительности (Performance). К подобным манипуляциям следует относиться внимательно, поскольку каждый уровень имеет различную стоимость, а также переключение между ними может занять время, если в аккаунте много данных.

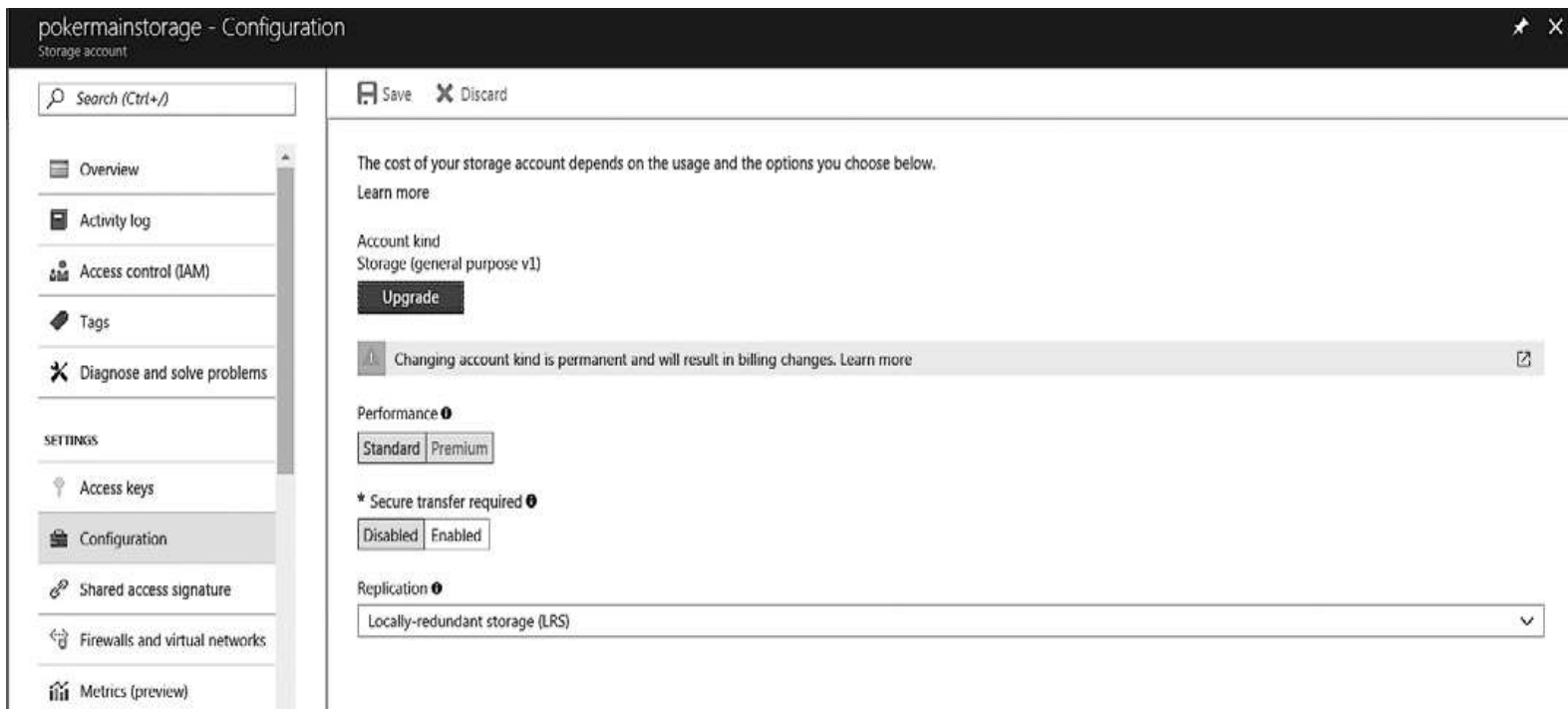


Рис. 4.10. Вкладка конфигурирования типа аккаунта, уровня производительности и репликации

# Сервисы хранения

## Microsoft Azure Storage

Следующий важный конфигурируемый параметр — Shared access signature (SAS) (рис. 4.11). Концепция SAS состоит в том, что к объектам, расположенным в Storage Account, можно предоставить прямой доступ для скачивания — с помощью URL, который содержит ряд ограничивающих параметров, а именно: время жизни ссылки и ограничения IP-адресов, с которых доступен ресурс. Эти параметры подписываются ключом аккаунта, и данная подпись добавляется в конец URL, по которому данный ресурс может быть доступен.

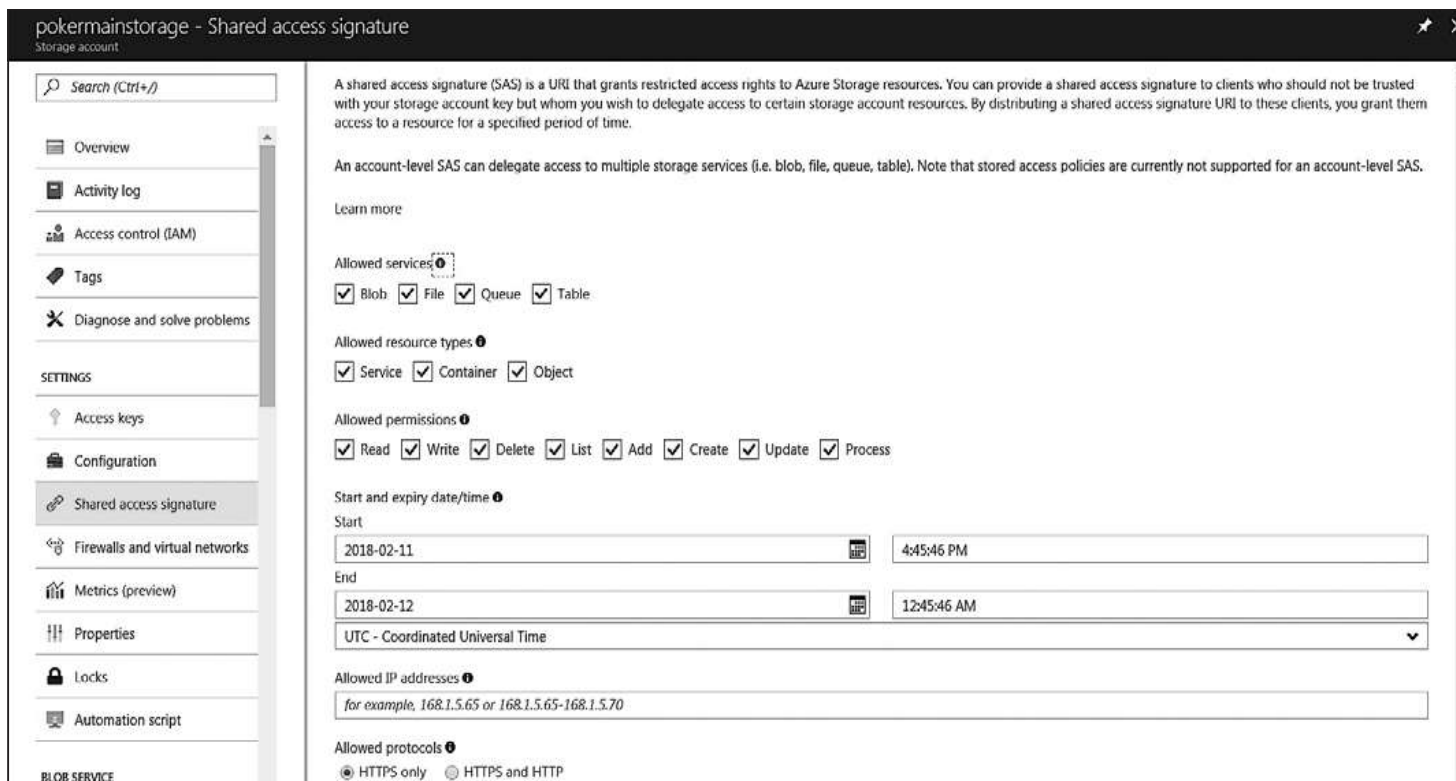


Рис. 4.11. Вкладка настройки параметров Shared Access Signature

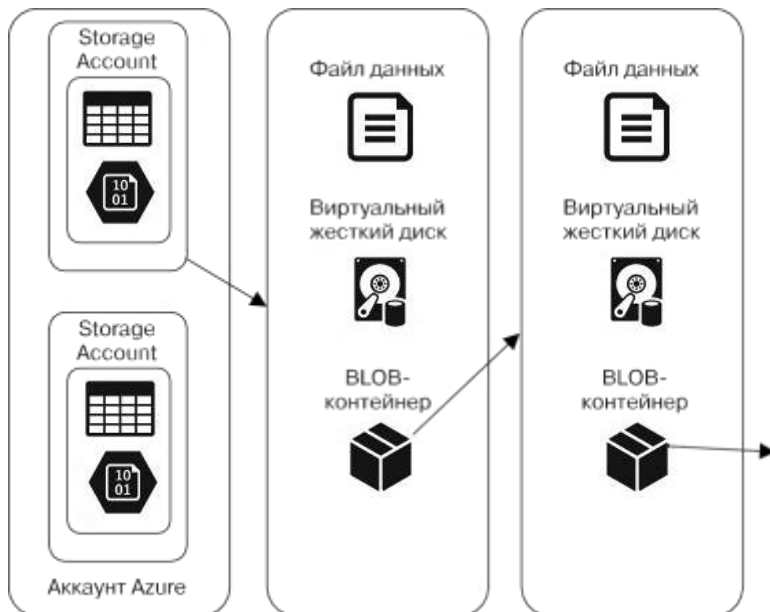
# Сервисы хранения

## Microsoft Azure Storage

Теперь подробнее познакомимся с отдельными сервисами хранения файлов Storage Account.

Сервис Azure BLOB Storage предназначен для хранения различных файлов, поэтому и называется хранилищем больших двоичных объектов (Binary Large Objects Storage, BLOB). Двоичные объекты могут храниться в нем как непосредственно, так и будучи размещенными в контейнерах (не путайте с Docker-контейнерами, речь идет о контейнерах BLOB Storage) или на виртуальных дисках, тоже расположенных в BLOB Storage Account.

*Чтобы прояснить ситуацию, рассмотрим рис. 4.12.*



*Аккаунт Azure может содержать один или несколько Storage-аккаунтов. Каждый такой аккаунт способен непосредственно хранить файлы, виртуальные жесткие диски и контейнеры BLOB. Последние, в свою очередь, тоже могут включать файлы, виртуальные жесткие диски и другие контейнеры. Таким образом, с помощью контейнеров BLOB реализована иерархическая структура организации файлов.*

Рис. 4.12. Структура вложенности объектов в хранилище BLOB

# Сервисы хранения

## Microsoft Azure Storage

В **BLOB Storage** могут храниться любые файлы: текстовые, двоичные и др., но для разных типов хранимых объектов требуется разный тип BLOB.

Всего есть три типа BLOB: **Page BLOB**, **Block BLOB** и **Append BLOB**.

**Page BLOB** — бинарный объект со страничной организацией памяти. Этот тип используется только для размещения виртуальных жестких дисков виртуальных машин.

**Block BLOB** — BLOB с блочной организацией памяти, служащий для хранения всех видов файлов (кроме VHD), включая контейнеры BLOB. Это основной тип хранения файлов в BLOB Storage обычных файлов.

**Append Block** — BLOB с блочной организацией памяти, представляющий собой текстовый файл, размещенный в Azure Storage и допускающий добавление новой записи в конец файла. В остальных типах BLOB файлы не редактируемые, то есть, чтобы отредактировать файл, его необходимо скачать, открыть, отредактировать и загрузить обратно. Понятно, что эти действия сопряжены с большими трудностями при работе с файлами логов. В то же время Append Block как раз оптимизирован для сценариев прямой записи логов.

Все объекты, расположенные в BLOB Storage, могут быть доступны через веб-портал, с помощью SDK, команд расширения командной оболочки, а также по прямой ссылке.



# Сервисы хранения

## Microsoft Azure Storage

Доступ к Storage-аккаунту с помощью веб-портала позволяет создавать файлы, контейнеры, просматривать список файлов, добавлять, удалять и загружать их, менять области видимости файлов (они могут быть общедоступными по ссылке, закрытыми для всех, кроме сервисов Azure).

Интерфейс веб-портала удобен для работы с небольшим количеством файлов. Добавлять в облачное хранилище через веб-портал можно файлы не слишком большого размера. А вот файлы, загружаемые из облака, могут быть любого размера, допустимого в хранилище. Кроме того, для файлов можно открыть общий доступ, и они станут доступными для скачивания по ссылке (эта ситуация отражена на рис. 4.13).

Доступ может быть открыт для файлов как в корневом каталоге, так и в контейнерах внутри хранилища.

# Способы доступа к файлам в облачном хранилище BLOB

- Для программного доступа облачный аккаунт содержит REST API, который, в свою очередь, через SDK предоставляет гораздо большие возможности: синхронную и асинхронную загрузку и выгрузку, удаление, создание, добавление в конец Append BLOB и пр.
- Кроме того, через SDK можно создать временную ссылку на файл, то есть ссылку, становящуюся нерабочей через определенный промежуток времени.

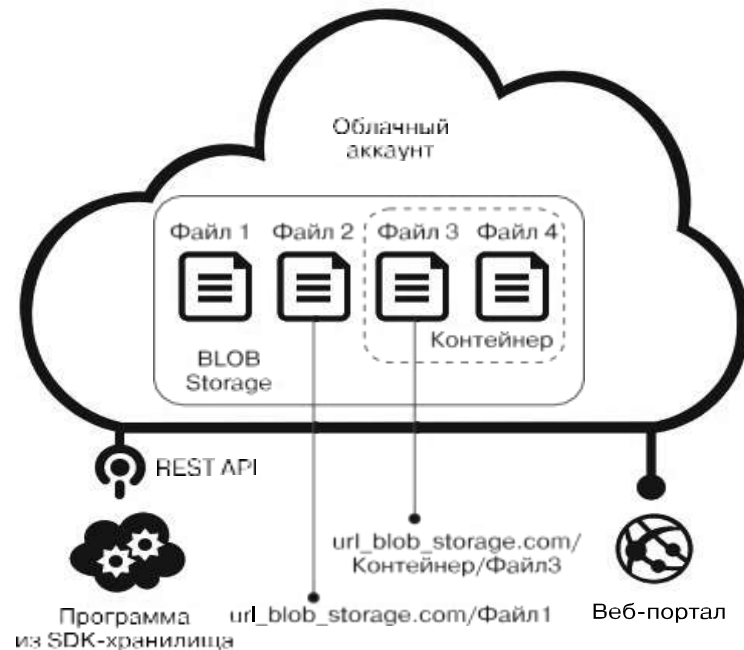


Рис. 4.13. Способы доступа к файлам в облачном хранилище BLOB

# Сервисы хранения

## Microsoft Azure Storage

Рассмотрим подробнее, как работать с хранилищем BLOB с помощью веб-портала.

- Чтобы перейти к хранилищу BLOB, необходимо нажать ссылку Blobs на общей панели аккаунта (см. рис. 4.7).
- В результате откроется вкладка, показанная на рис. 4.14.

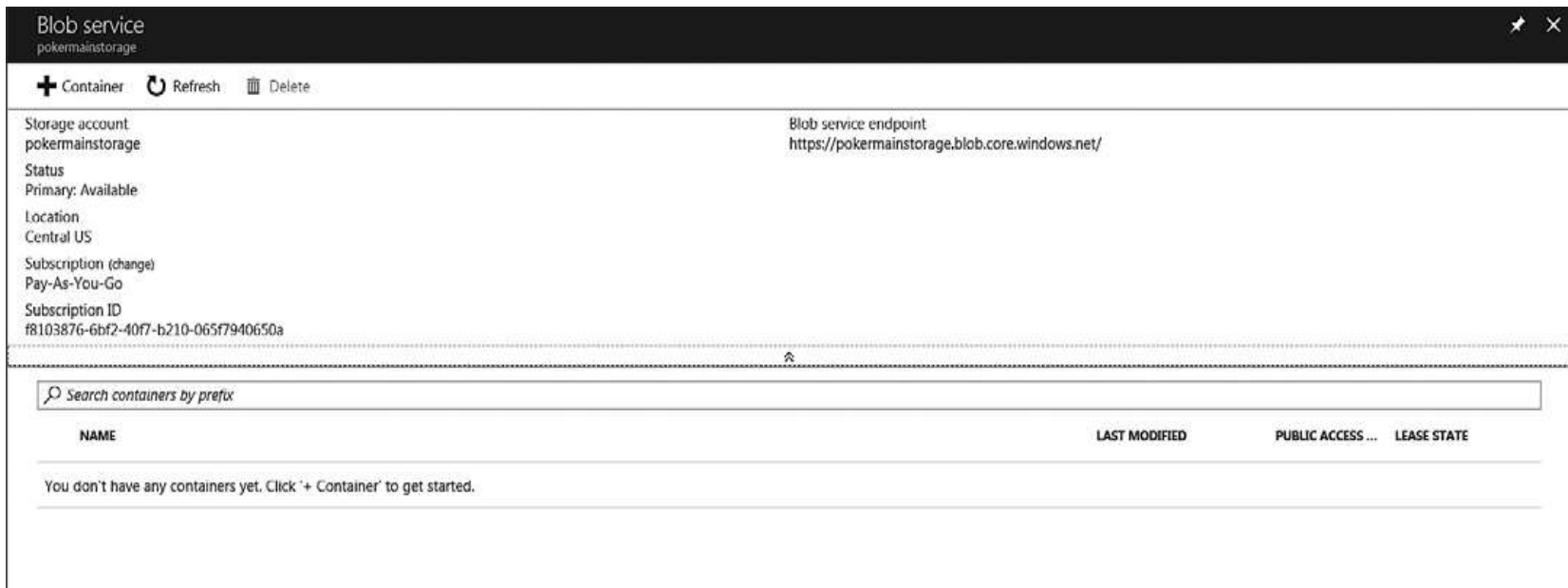


Рис. 4.14. Общая панель сервиса BLOB Storage

# Сервисы хранения

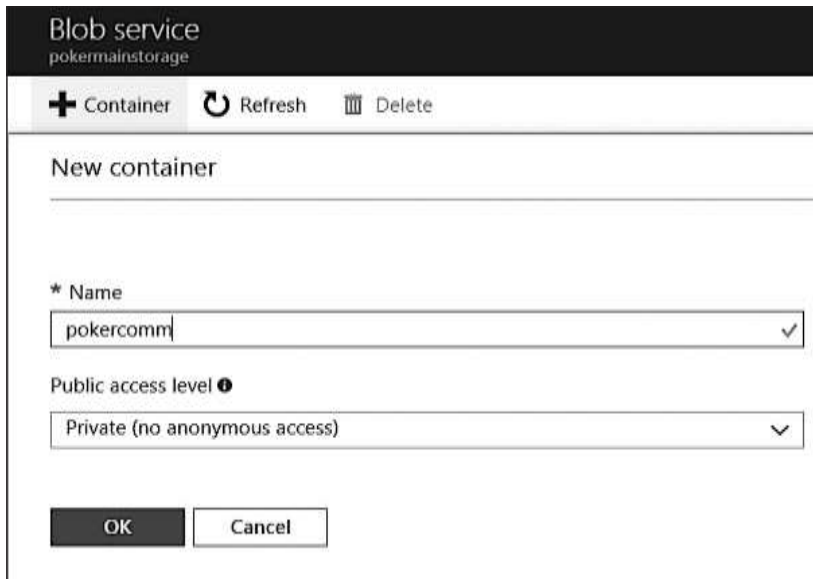
## Microsoft Azure Storage

Далее требуется добавить контейнеры.

Для этого необходимо нажать ссылку **+ Container** (рис. 4.15).

В данной форме указано **имя (свойство Name)** — **pokercomm**; **уровень доступа (Public access level)** — **Private (no anonymous access)**. Этот тип доступа подразумевает доступ не по прямой ссылке, а только с помощью ключей аккаунта с использованием SDK.

Другие варианты типа доступа: **Blob (anonymous read access for blobs only)** — разрешает анонимный доступ только к файлам (BLOB); **Container (anonymous read access for containers and blobs)** — разрешен анонимный доступ к контейнерам и файлам. Вкладка созданного контейнера выглядит следующим образом (рис. 4.16).



The screenshot shows a 'New container' dialog box. At the top, there are buttons for '+ Container', 'Refresh', and 'Delete'. Below that, the text 'New container' is displayed. The 'Name' field is filled with 'pokercomm'. The 'Public access level' dropdown is set to 'Private (no anonymous access)'. At the bottom, there are 'OK' and 'Cancel' buttons.

Рис. 4.15. Форма создания нового контейнера



Рис. 4.16. Вкладка созданного контейнера

# Сервисы хранения

## Microsoft Azure Storage

- Загрузить файл в контейнер можно с помощью ссылки Upload. Свойства контейнера доступны по ссылке Container properties и показаны на рис. 4.17. Они включают в себя имя, адрес, статус, количество и суммарный размер BLOB-объектов, статус.
- Для контейнера доступна настройка политики доступа через вкладку Access policy (рис. 4.18). Эта настройка позволяет организовать различный уровень доступа к различным контейнерам и объектам BLOB.
- Помимо упомянутых настроек, для BLOB доступен ряд других (рис. 4.19).

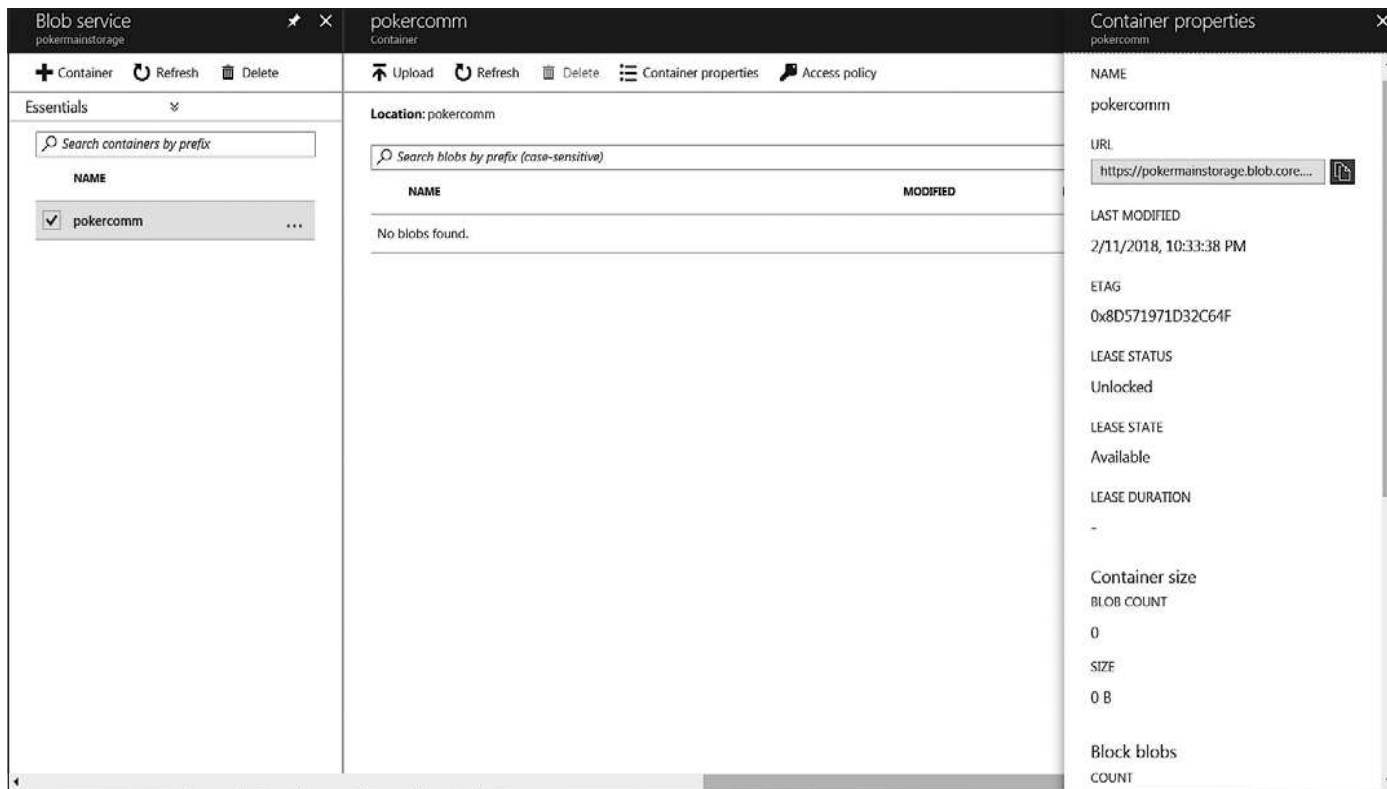


Рис. 4.17. Вкладка свойств контейнера

# Сервисы хранения

## Microsoft Azure Storage

- Загрузить файл в контейнер можно с помощью ссылки Upload. Свойства контейнера доступны по ссылке Container properties и показаны на рис. 4.17. Они включают в себя имя, адрес, статус, количество и суммарный размер BLOB-объектов, статус.
- Для контейнера доступна настройка политики доступа через вкладку Access policy (рис. 4.18). Эта настройка позволяет организовать различный уровень доступа к различным контейнерам и объектам BLOB.
- Помимо упомянутых настроек, для BLOB доступен ряд других (рис. 4.19).

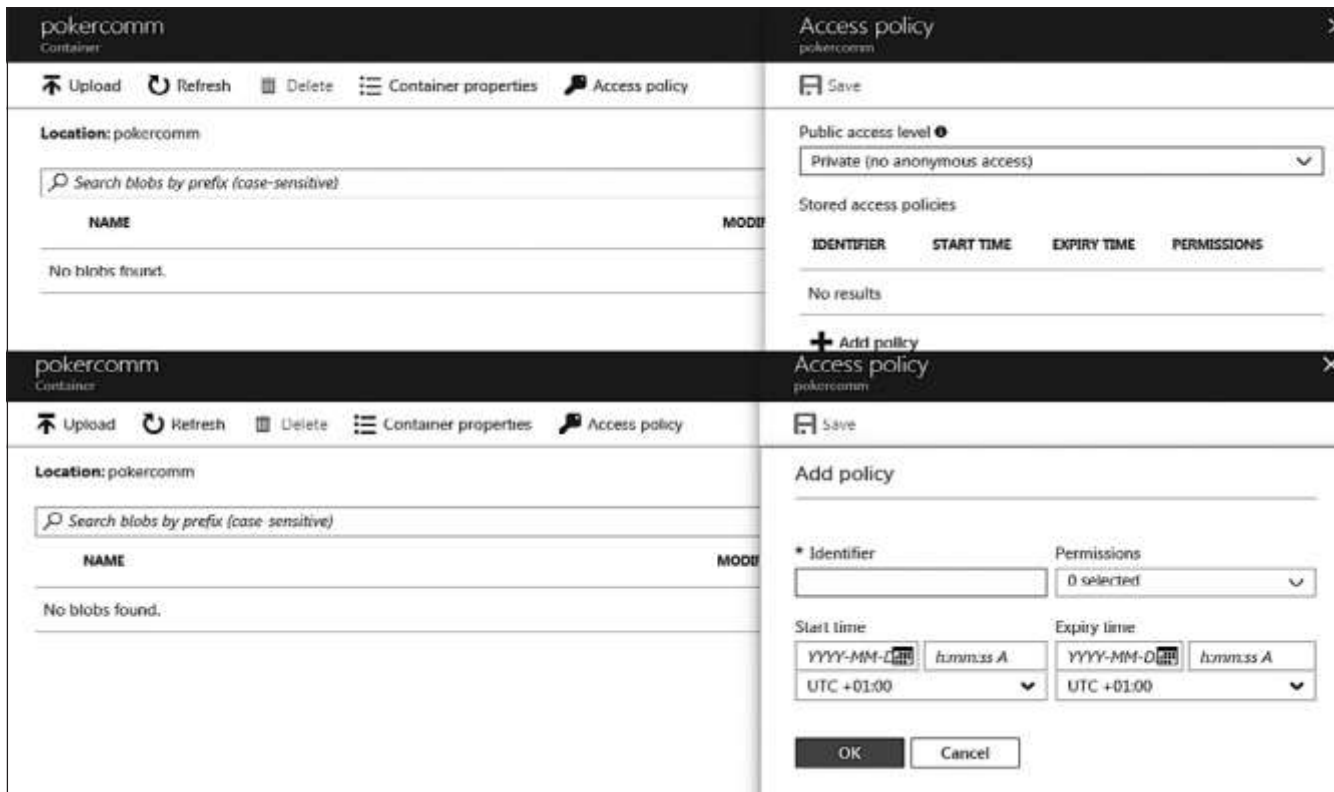


Рис. 4.18. Вкладка настройки политики доступа

# Сервисы хранения

## Microsoft Azure Storage

- Загрузить файл в контейнер можно с помощью ссылки Upload. Свойства контейнера доступны по ссылке Container properties и показаны на рис. 4.17. Они включают в себя имя, адрес, статус, количество и суммарный размер BLOB-объектов, статус.
- Для контейнера доступна настройка политики доступа через вкладку Access policy (рис. 4.18). Эта настройка позволяет организовать различный уровень доступа к различным контейнерам и объектам BLOB.
- Помимо упомянутых настроек, для BLOB доступен ряд других (рис. 4.19).

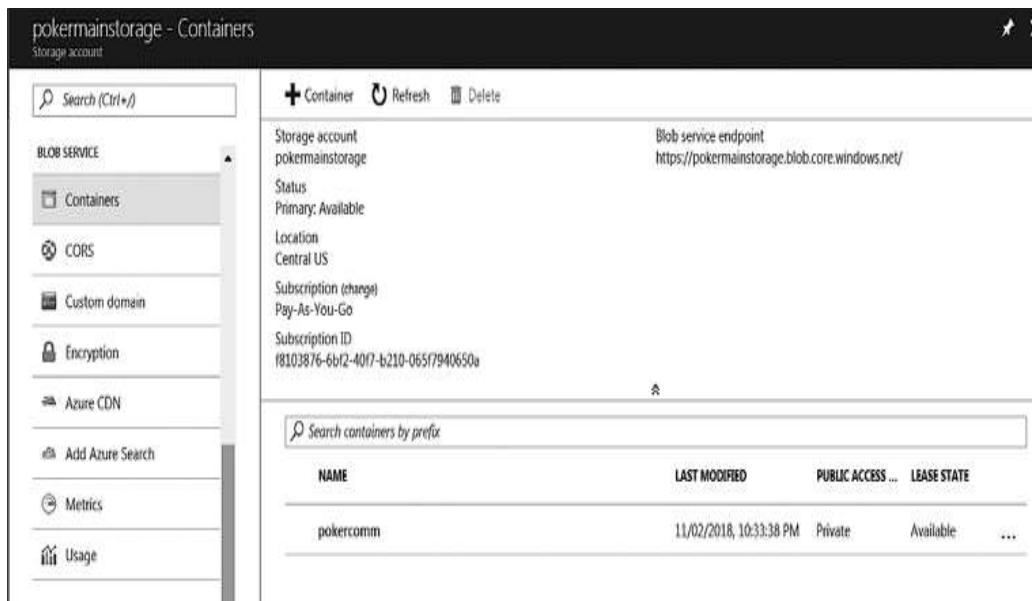


Рис. 4.19. Различные настраиваемые сервисы Blob Storage

# Сервисы хранения

## Microsoft Azure Storage

Самые важные параметры:

- CORS (cross-origin resource sharing — совместное использование ресурсов между разными источниками) — свойство, обеспечивающее доступ к ресурсам BLOB из другого домена;
- Custom domain — конфигурирование DNS-записей CNAME в целях указания домена пользователя, в дополнение к домену аккаунту Azure Storage;
- Encryption — шифрование объектов в хранилище;
- Azure CDN — конфигурирование Azure Content Delivery Network, которая служит для хранения часто используемого контента из хранилища BLOB с анонимным доступом.

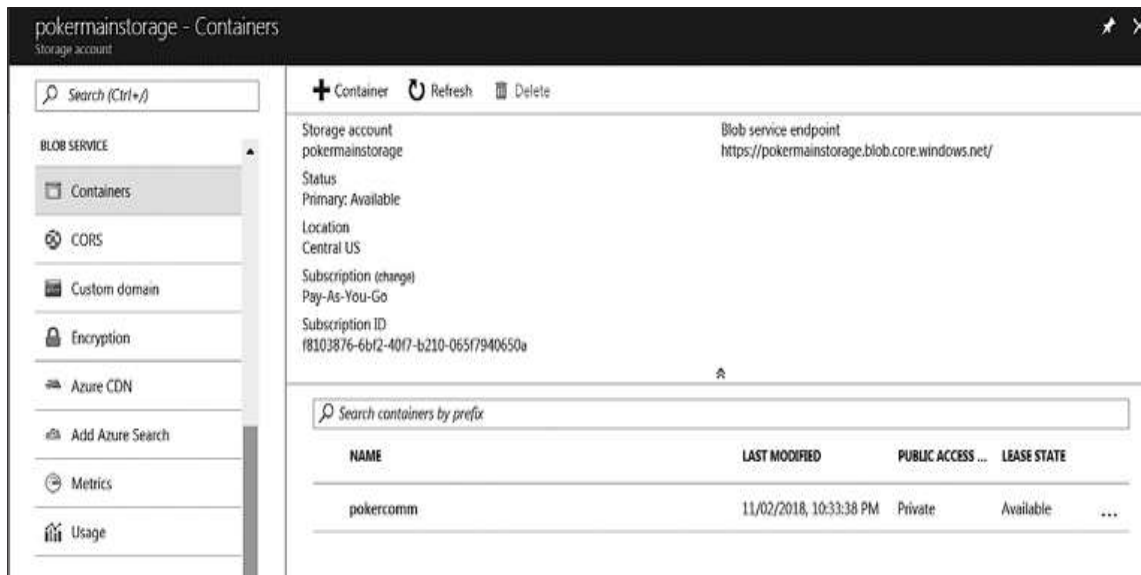


Рис. 4.19. Различные настраиваемые сервисы Blob Storage



# Сервисы хранения

## Microsoft Azure Storage

Доступ к объектам в BLOB Storage возможен по прямой ссылке URL или с помощью интерфейса REST API (напрямую либо через SDK). Данный способ хранения обеспечивает самый быстрый доступ (минимальное время загрузки/выгрузки), но требует применения специальных программных клиентов, взаимодействующих с этими API.

Последнее условие может помешать существующим приложениям большого масштаба мигрировать в облако. Кроме того, в ряде случаев нужно создать облачное хранилище, которое должно быть доступно из виртуальной машины без всяких «самописных» программных клиентов.

Для этого в хранилище BLOB можно разместить виртуальный жесткий диск (или набор дисков для RAID-массива) и примонтировать его к виртуальной машине.

- **Преимущество такого решения** состоит в том, что данные из виртуальной машины могут быть доступны точно таким же образом, как и с физического диска, подключенного к ней.
- **Недостаток такого способа** заключается в его низкой масштабируемости (верхний предел размера ограничен на уровне, определяемом операционной системой виртуальной машины и типом ее устройства ввода-вывода), дороговизне, а также в недоступности файлов извне по прямой ссылке. Скорость доступа к файлам тоже определяется конфигурацией топологии соединения дисков в массив RAID и в ряде ситуаций существенно ниже, чем в случае BLOB.

Чтобы преодолеть эти ограничения и одновременно обеспечить работу с файлами стандартными средствами операционных систем и программных библиотек ввода-вывода, следует использовать облачное файловое хранилище Azure File Storage (рис. 4.20).

# Способы доступа к облачному файловому хранилищу Azure File Storage

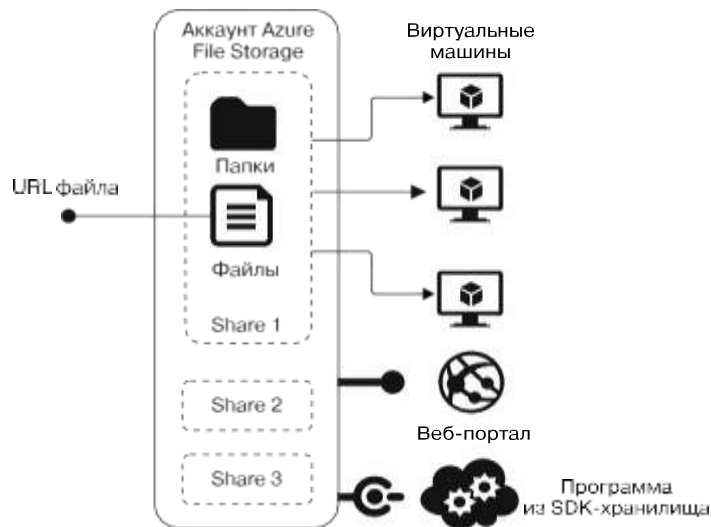


Рис. 4.20. Способы доступа к облачному файловому хранилищу Azure File Storage

Сервис Azure File Storage составляет часть Azure Storage Account.

Каждый Storage Account может включать в себя одну или несколько шар (share).

Чтобы создать новую шару, необходимо с общей панели (см. рис. 4.7) добавить вкладку, нажав + **File share**.

The screenshot shows the 'File service' interface with a 'New file share' dialog. The dialog has a title bar 'File service' and a subtitle 'pokermainstorage'. There are buttons for '+ File share' and 'Refresh'. The dialog contains a 'Name' field with the value 'pokerfileshare' and a 'Quota' field with the value '10 GB'. There are 'OK' and 'Cancel' buttons at the bottom.

В появившейся форме (рис. 4.21) указывается имя шары (Name) — в нашем примере это pokerfileshare — и ее размер (Quota), в данном случае 10 Гбайт. Размер каждой шары (квота) может изменяться (с помощью веб-портала, SDK-управления облачными ресурсами или Azure CLI) и способна достигать 5 Тбайт.

Рис. 4.21. Форма добавления новой файловой шары

# Способы доступа к облачному файловому хранилищу Azure File Storage

- Ключевым отличием шары от контейнера является то, что для нее устанавливается квота — верхний предел размера.
- Эта шаря представляет собой корневой каталог, который доступен по протоколу SMB 3.0 и может быть примонтирован к виртуальным машинам в том же аккаунте Azure и в том же регионе, что и Azure File Storage.
- Наиболее важными опциями конфигурирования является возможность подключения (Connect) к виртуальной машине и создание мгновенного снимка (View snapshot) (рис. 4.22).

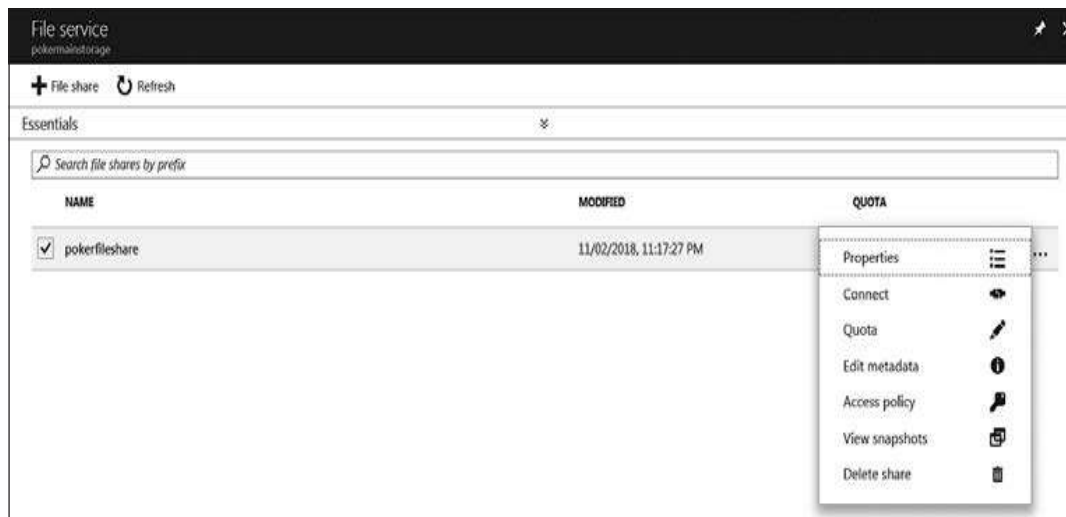


Рис. 4.22. Доступные опции конфигурирования файловой шары

# Способы доступа к облачному файловому хранилищу Azure File Storage

Чтобы подключить шару File Storage к виртуальной машине, в оболочке виртуальной машины нужно выполнить команду монтирования сетевого диска. Эта команда может быть получена напрямую на веб-портале после щелчка на ссылке Connect (рис. 4.23).

## Достоинства файлового хранилища представлены ниже.

- Возможность прямой миграции файловой системы из локального хранилища в облачное. Будет полностью сохранена иерархия этой системы (вероятные ограничения на символьную длину пути и глубину вложенности каталогов см. в документации).
- Простота взаимодействия из всех программных продуктов, реализующих стандартные интерфейсы ввода-вывода. Не требуется никаких модификаций кода программ, они могут быть устаревшими и все равно будут работать с Azure File Storage, поскольку это хранилище монтируется к основной файловой системе виртуальной машины на уровне операционной системы.
- Возможен просмотр списка файлов с помощью стандартных средств Windows или Linux.

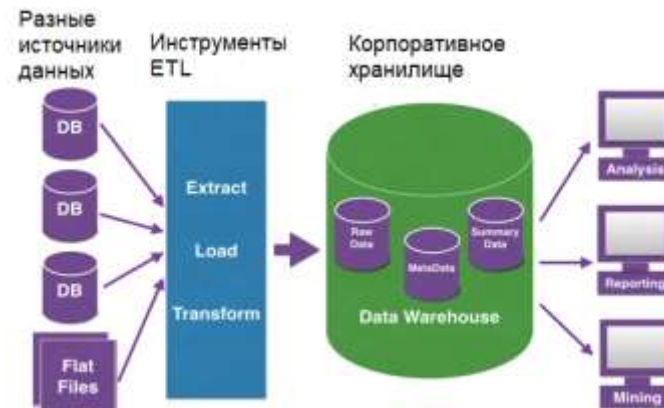
## Недостатки файлового хранилища:

- ограниченный размер файловой шары (5 Тбайт) и одного файла (1 Тбайт);
- более высокая по сравнению с BLOB Storage цена за гигабайт;
- меньшая по сравнению с BLOB Storage производительность операций чтения-записи.



Рис. 4.23. Вкладка Connect Azure File Storage

# Модуль 1. Основы построения и работы с системами аналитики больших данных



## КРАТКОЕ СОДЕРЖАНИЕ:

### 1. Архитектура систем аналитики больших данных.

1.1. Облачные технологии. Модели развертывания. Способы создания ресурсов в облаке.

1.2. Безопасность облачных ресурсов

1.3. Большие данные и источники данных. Форматы. Преобразование данных из различных форматов. Режимы обработки больших данных.

### 2. Хранение больших данных в облаке.

2.1. Хранилища общего назначения. Форматы хранения данных. Облачное хранилище Microsoft Azure Storage. Облачное хранилище AWS.

2.2. Реляционные базы данных. Azure SQL. AWS RDS.

2.3. Нереляционные базы данных. Сервисы нереляционных баз данных от Azure и AWS.

2.4. Корпоративные хранилища данных (DWH). Azure SQL DWH. AWS RedShift.

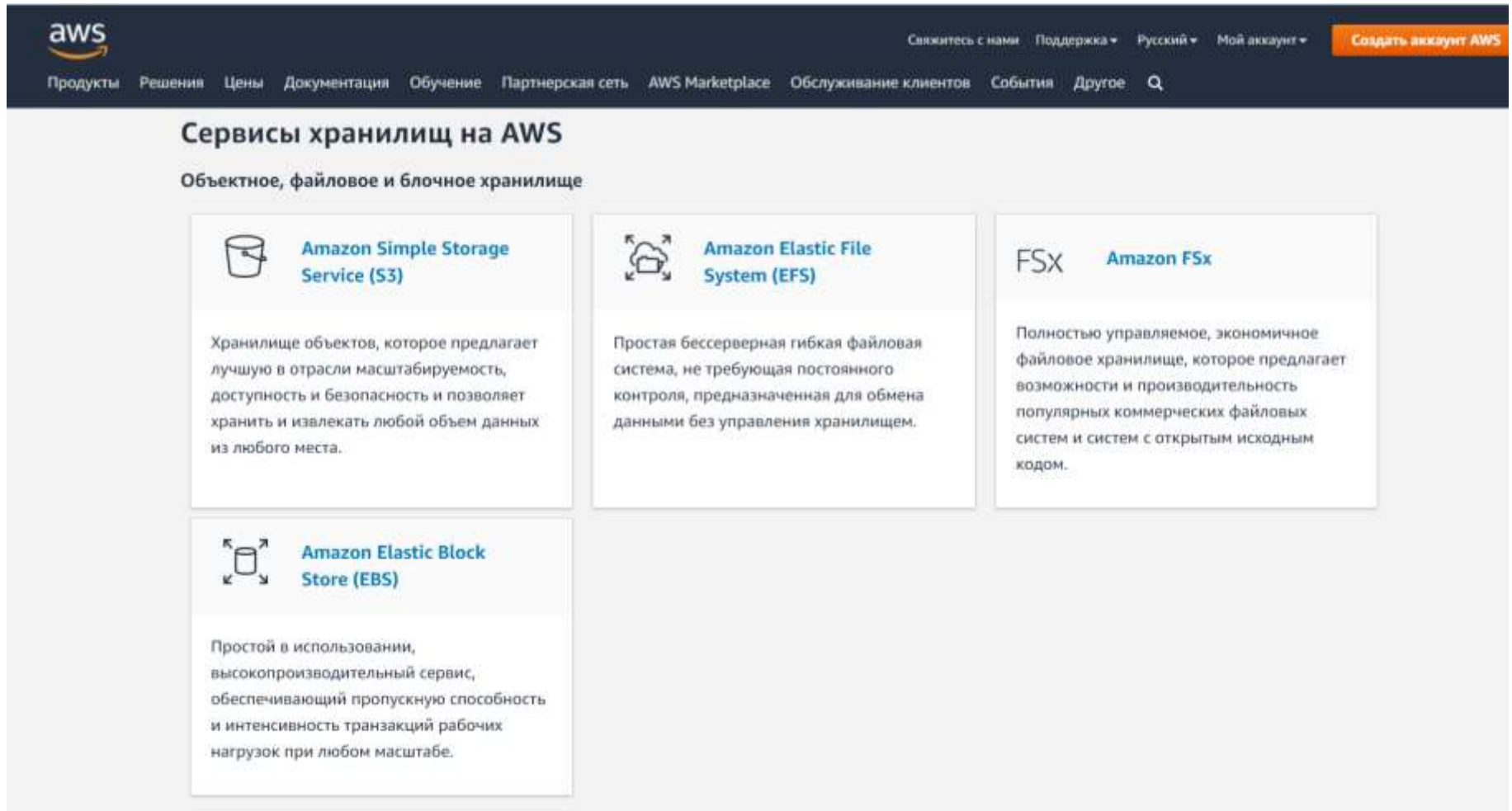
2.5. Хранилища данных типа Data Lake. Azure Data Lake Store. AWS Data Lake Solutions.

# ЛЕКЦИЯ 7:

## Хранение больших данных в облаке Облачные хранилища AWS



# Классы хранилища Amazon




The screenshot shows the AWS website header with the logo and navigation links. Below the header, the main heading is "Сервисы хранилищ на AWS" (Storage Services on AWS), followed by the subtitle "Объектное, файловое и блочное хранилище" (Object, file, and block storage). Three service cards are displayed: Amazon Simple Storage Service (S3), Amazon Elastic File System (EFS), and Amazon FSx. A fourth card for Amazon Elastic Block Store (EBS) is partially visible at the bottom left.

**aws** Связаться с нами Поддержка Русский Мой аккаунт Создать аккаунт AWS


Продукты Решения Цены Документация Обучение Партнерская сеть AWS Marketplace Обслуживание клиентов События Другое

## Сервисы хранилищ на AWS

Объектное, файловое и блочное хранилище

 **Amazon Simple Storage Service (S3)**


Хранилище объектов, которое предлагает лучшую в отрасли масштабируемость, доступность и безопасность и позволяет хранить и извлекать любой объем данных из любого места.

 **Amazon Elastic File System (EFS)**

Простая бессерверная гибкая файловая система, не требующая постоянного контроля, предназначенная для обмена данными без управления хранилищем.

**FSx** **Amazon FSx**

Полностью управляемое, экономичное файловое хранилище, которое предлагает возможности и производительность популярных коммерческих файловых систем и систем с открытым исходным кодом.

 **Amazon Elastic Block Store (EBS)**

Простой в использовании, высокопроизводительный сервис, обеспечивающий пропускную способность и интенсивность транзакций рабочих нагрузок при любом масштабе.

Рассмотрим облачные хранилища общего назначения, предоставляемые облачным провайдером **Amazon Web Service**. В отличие от Azure у AWS хранилища не объединяются логически в Storage Account, а представляют собой набор отдельных сервисов. Хранение двоичных объектов обеспечивается сервисом AWS S3 — Amazon Simple Storage Service.

# Amazon Simple Storage Service

Хранение двоичных объектов обеспечивается сервисом **AWS S3 — Amazon Simple Storage Service**. Экземпляр сервиса AWS S3 может быть логически разбит на бакеты (buckets, дословный перевод — «ведра»), которые будут определять видимость объектов в них, шифрование, жизненный цикл и пр.

Каждый объект (будь то бакет или хранящийся в нем файл) содержит ассоциированный с ним класс хранения (storage class).

Ниже перечислены существующие классы хранения.

- Стандартный (STANDARD) — класс, задаваемый по умолчанию. Предназначен для случая, когда частота доступа к файлам высокая, в связи с чем требуется высокая производительность операций чтения и записи.
- Стандартный с нечастым доступом (STANDARD\_IA, Standard Infrequent Access) — предназначен для долговременного хранения файлов, доступ к которым будет производиться нечасто (например, бэкапы), но тем не менее высокая производительность операций чтения-записи все еще важна. Оба класса — STANDARD и STANDARD\_IA — обеспечивают доступ к файлу в режиме, близком к реальному времени, то есть с минимальной задержкой между запросом и получением данных.
- Замороженный, или, точнее, ледник (GLACIER), — предназначен для файлов большого размера, доступ к которым будет запрашиваться очень редко (архивы, бэкапы и др.). Принципиальное его отличие от двух предыдущих классов хранения — невозможность прямого доступа к файлам, которые перед выгрузкой должны быть сначала восстановлены в другие классы.
- С уменьшенной избыточностью (REDUCED\_REDUNDANCY) — предназначен для хранения некритических файлов с той же скоростью доступа, что и у STANDARD, но за счет уменьшенной избыточности допускается потеря 0,01 % объектов.



# Amazon Simple Storage Service

Принципиальное различие между всеми классами — их стоимость. Очевидно, STANDARD обладает наивысшей стоимостью, а GLACIER и REDUCED\_REDUNDANCY — наивысшей. В ряде сценариев перемещение файлов из одного класса в другой может быть широко используемой операцией.

Например, это происходит на фотохостинге, размещающем картинки, доступные для скачивания по ссылке. Широко известен интернет-феномен мемов, когда контент внезапно становится очень популярным и в течение нескольких месяцев количество его скачиваний может резко возрасти, а затем значительно упасть. Если у этого фотохостинга большой объем хранимого контента, то становится актуальной проблема оптимизации стоимости хранения. Решить ее призван специальный встроенный механизм AWS S3: управление жизненным циклом объектов (object lifecycle management).

Данный сервис автоматизирует перемещение объектов между классами хранения по цепочке STANDARD - STANDARD\_IA - GLACIER, что избавляет пользователя от необходимости строить специальные сервисы, ответственные за это перемещение.

Следующий сценарий применения этого сервиса — жизненный цикл бэкапа. Это необходимо для оптимизации системы хранения бэкапа с точки зрения стоимости/доступности.

Рассмотрим, как создавать бакеты и манипулировать объектами в них. Начальная страница сервиса AWS S3 выглядит следующим образом (рис. 4.24).

# Начальная страница сервиса AWS S3

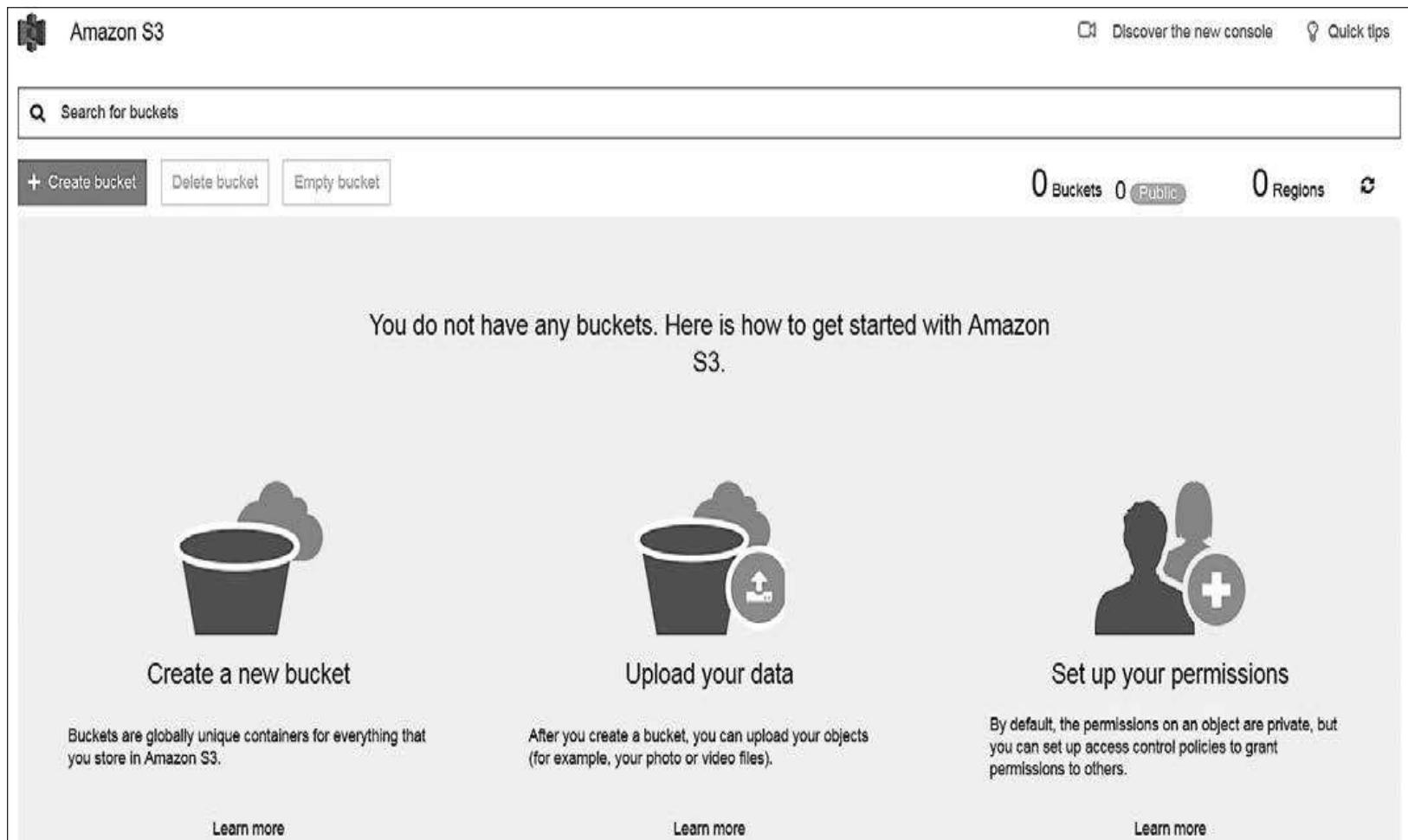


Рис. 4.24. Стартовая страница сервиса AWS S3

# Начальная страница сервиса AWS S3

- Чтобы создать бакет, надо щелкнуть на ссылке + Create bucket в левом верхнем углу.
- В результате откроется следующая форма (правая половинка доступна после успешного заполнения всех полей первой формы — вкладки Name and region (Имя и регион)) (рис. 4.25).

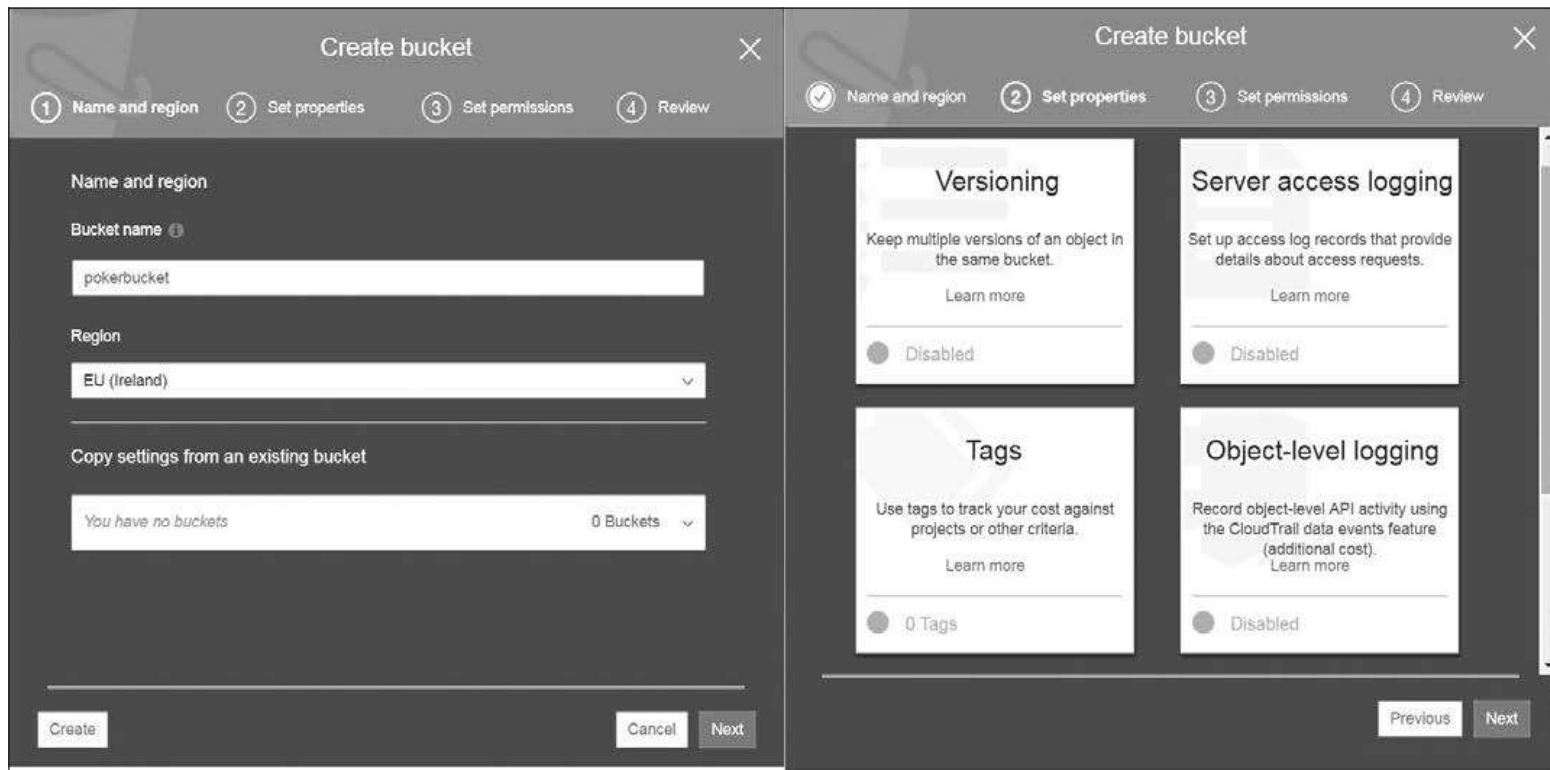


Рис. 4.25. Последовательные шаги создания бакета: вкладки Name and region (Имя и регион) и Set properties (Настройка свойств)

# Дальнейшие шаги создания бакета

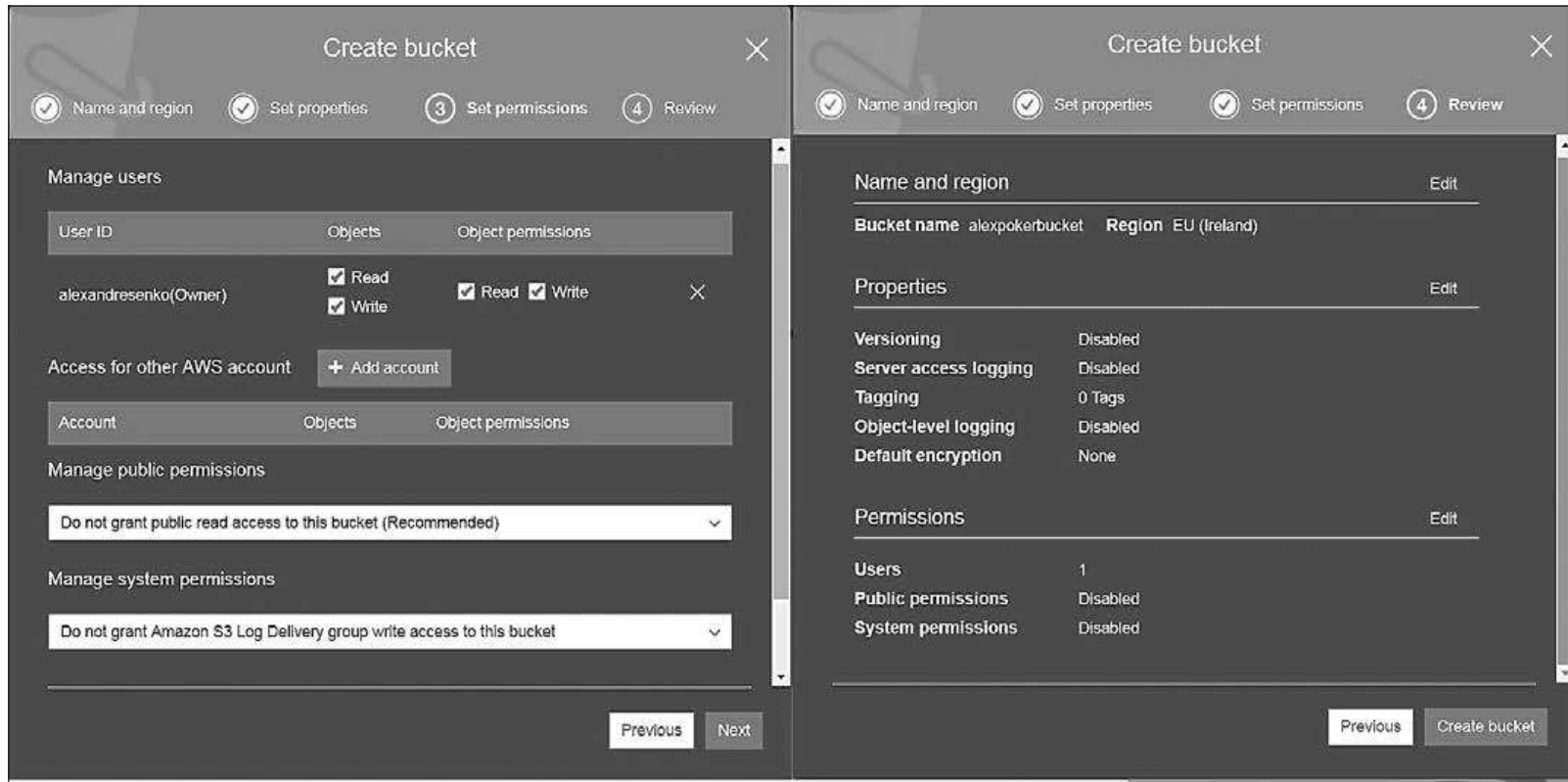


Рис. 4.26. Дальнейшие шаги создания бакета: Set permissions (Настройка прав доступа) и Review (Финальный обзор)

# Дальнейшие шаги создания бакета

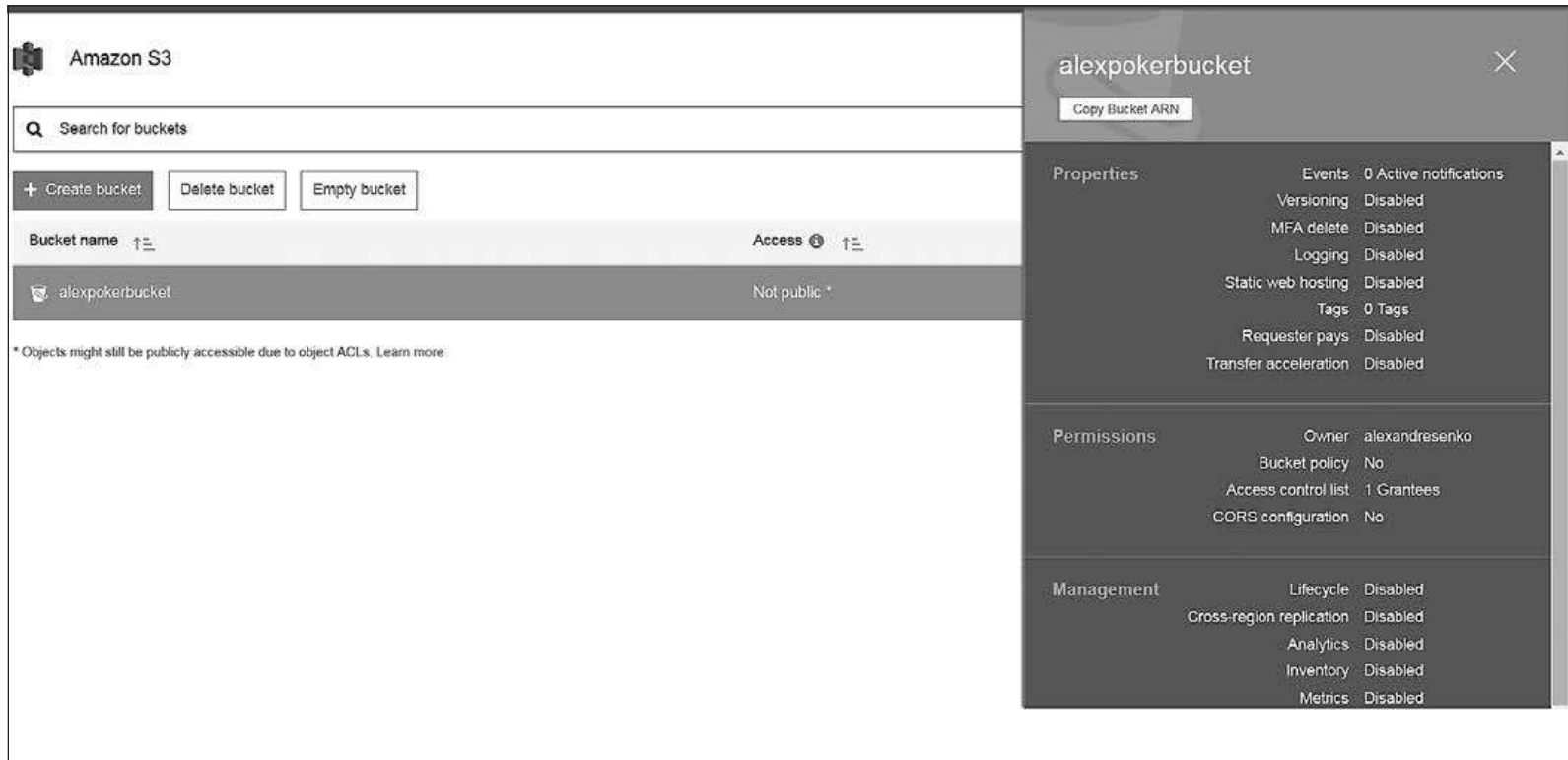


Рис. 4.27. Созданный бакет с раскрытой формой свойств

# Дальнейшие шаги создания бакета

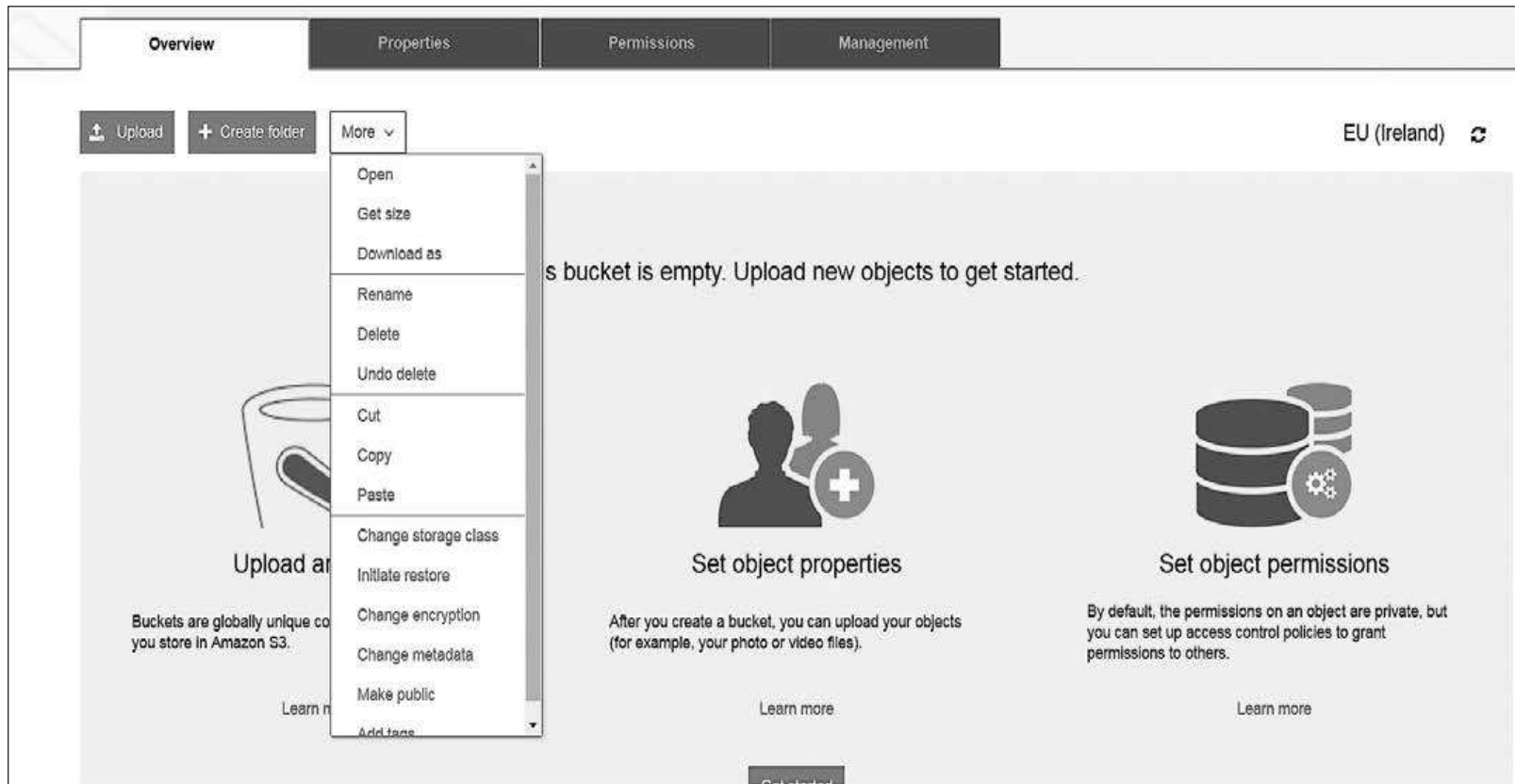


Рис. 4.28. Вкладка бакета, позволяющая управлять объектами

# Хранилище файлов, предоставляемое AWS — Amazon Elastic File Service (EFS).

Следующее хранилище файлов, предоставляемое AWS, — Amazon Elastic File Service (EFS). Этот сервис является облачной реализацией сетевого хранилища с поддержкой протокола NFSv4. Он изначально предназначен для доступа с виртуальных машин AWS EC2. Файлы и метаданные хранятся в виде нескольких копий в разных зонах доступности (availability zone) в пределах одного региона.

Масштабирование хранилища происходит вплоть до петабайтных размеров.

Поскольку этот сервис тесно связан с AWS EC2, то для его конфигурирования требуется настраивать сетевые IaaS-сервисы — VPC, подсети, availability zone, сетевые группы безопасности (рис. 4.29).

# Хранилище файлов, предоставляемое AWS — Amazon Elastic File Service (EFS).

Create file system

Step 1: Configure file system access

Step 2: Configure optional settings

Step 3: Review and create

### Configure file system access

An Amazon EFS file system is accessed by EC2 instances running inside one of your VPCs. Instances connect to a file system by using a network interface called a mount target. Each mount target has an IP address, which we assign automatically or you can specify.

VPC  ⓘ

### Create mount targets

Instances connect to a file system by using mount targets you create. We recommend creating a mount target in each of your VPC's Availability Zones so that EC2 instances across your VPC can access the file system.

	Availability Zone	Subnet ⓘ	IP address ⓘ	Security groups ⓘ
<input checked="" type="checkbox"/>	us-east-2a	<input type="text" value="subnet-4e0cbb26 (default)"/>	Automatic ⚙	<input type="text" value="sg-464fe72d - default ✕"/>
<input checked="" type="checkbox"/>	us-east-2b	<input type="text" value="subnet-3d48a647 (default)"/>	Automatic ⚙	<input type="text" value="sg-464fe72d - default ✕"/>
<input checked="" type="checkbox"/>	us-east-2c	<input type="text" value="subnet-5b0d2c16 (default)"/>	Automatic ⚙	<input type="text" value="sg-464fe72d - default ✕"/>

Cancel

Рис. 4.29. Начальная страница конфигурирования сервиса AWS EFS



# Хранилище файлов, предоставляемое AWS — Amazon Elastic File Service (EFS).

На следующей вкладке указываем теги, уровень производительности (General purpose или Max I/O) и шифрование (Enable encryption) (рис. 4.30).

Далее выполняем обзор параметров создаваемого сервиса и в итоге имеем готовый сервис, который можно подключать к виртуальной машине AWS EC2 (рис. 4.31).

Чтобы получить инструкции по подключению файловой системы к виртуальной машине, следует нажать на ссылку [Amazon EC2 mount instructions](#) или [AWS Direct Connect mount instructions](#).

# Хранилище файлов, предоставляемое AWS — Amazon Elastic File Service (EFS).

## Create file system

Step 1: Configure file system access

**Step 2: Configure optional settings**

Step 3: Review and create

### Configure optional settings

#### Add tags

You can add tags to describe your file system. A tag consists of a case-sensitive key-value pair. (For example, you can define a tag with key-value pair with key = Corporate Department and value = Sales and Marketing.) At a minimum, we recommend a tag with key = Name.

Key	Value	Remove
<input type="text" value="Name"/>	<input type="text" value="GeneralPokerStore"/>	<input type="button" value="✕"/>
<input type="text" value="Add New Key"/>	<input type="text"/>	

#### Choose performance mode

We recommend **General Purpose** performance mode for most file systems. **Max I/O** performance mode is optimized for applications where tens, hundreds, or thousands of EC2 instances are accessing the file system — it scales to higher levels of aggregate throughput and operations per second with a tradeoff of slightly higher latencies for file operations.

**General Purpose** (default)

**Max I/O**

#### Enable encryption

If you enable encryption for your file system, all data on your file system will be encrypted at rest. You can select a KMS key from your account to protect your file system, or you can provide the ARN of a key from a different account. Encryption can only be enabled during file system creation. [Learn more](#)

**Enable encryption**

Рис. 4.30. Дальнейшее конфигурирование AWS EFS

# Хранилище файлов, предоставляемое AWS — Amazon Elastic File Service (EFS).

**File systems**

Create file system Actions

Name	File system ID	Metered size	Number of mount targets	Creation date
GeneralPokerStore	fs-43aa553a	6.0 KiB	3	2018-02-13T09:45:57Z

**Other details**

Owner ID: 759462564594  
Life cycle state: Available  
Performance mode: General Purpose  
Encrypted: No

**Tags**

Name: GeneralPokerStore

**File system access**

DNS name: fs-43aa553a.efs.us-east-2.amazonaws.com

Amazon EC2 mount instructions  
AWS Direct Connect mount instructions

**Mount targets**

VPC	Availability Zone	Subnet	IP address	Mount target ID	Network interface ID	Security groups	Life cycle state
vpc-2d2f8545 (default)	us-east-2a	subnet-4e0cbb26 (default)	172.31.4.176	fsmt-c85aa4b1	eni-e92065b7	sg-464fe72d - default	Available
	us-east-2c	subnet-5b0d2c16 (default)	172.31.42.64	fsmt-ca5aa4b3	eni-6ae8ed41	sg-464fe72d - default	Available
	us-east-2b	subnet-3d48a647 (default)	172.31.30.226	fsmt-cb5aa4b2	eni-5627f502	sg-464fe72d - default	Available

Рис. 4.31. Созданная файловая система AWS EFS, готовая к монтированию

# Хранилище файлов, предоставляемое AWS — Amazon Elastic File Service (EFS).

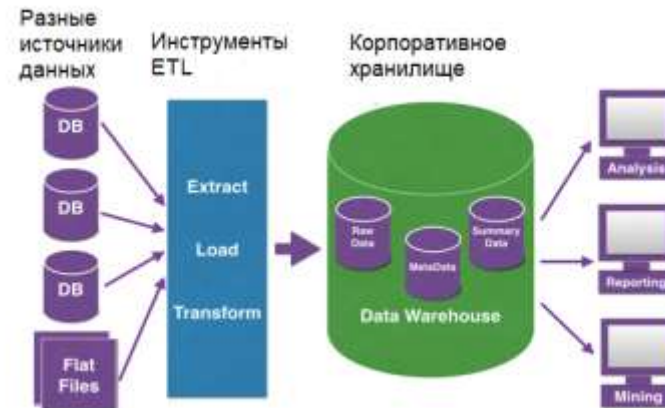
- ❖ Интересная возможность AWS EFS — функция EFS FileSync, позволяющая синхронизировать файлы в EFS и файлы в локальной файловой системе как EC2, так и физического компьютера вне AWS. В последнем случае нужно установить специальный агент на тот компьютер, к которому будут подключаться диски, требующие синхронизации.
- ❖ В отличие от Azure, где для хранения дисков виртуальных машин используется обычный аккаунт BLOB Storage, в котором хранятся BLOB-объекты с блочной организацией, в Amazon применяется специальный сервис — AWS EBS (Elastic Block Storage).
- ❖ Этот сервис отвечает за создание, размещение, шифрование, подключение к виртуальным машинам, создание бэкапов в виде мгновенных снимков (snapshots). Чаще всего пользователь не будет взаимодействовать с ним напрямую, этот сервис доступен в «связке» с другими: AWS EC2, AWS ECS и пр.
- ❖ То есть при создании виртуальной машины тип и количество виртуальных дисков, их точки монтирования или метки указывается прямо в момент создания.
- ❖ Однако нужно помнить, что бывают сценарии, когда эти виртуальные диски не удаляются при удалении виртуальной машины.

# Хранилище файлов, предоставляемое AWS — AWS EBS (Elastic Block Storage)

Диски EBS можно разделить на две большие группы: HDD (Hard Disk Drive) (классический магнитный диск) и SSD (Solid State Drive) (твердотельный диск). Обе группы имеют следующие разновидности.

- *Холодный HDD (Cold HDD)* — самый дешевый тип диска EBS. Сценарии его использования включают хранение больших объемов данных, которые редко запрашиваются извне в случаях, когда стоимость хранения имеет значение. Применение этого типа сопряжено с рядом ограничений: загрузочный (boot) диск его не поддерживает. Кроме того, существуют разного рода ограничения для использования холодного HDD вместе с ECS-оптимизированными виртуальными машинами.
- *Горячий HDD (throughput oriented HDD, дословно «HDD с оптимизированной производительностью»)* — магнитный диск малой стоимости, оптимизированный для сценариев хранения больших объемов данных, частота запросов которых высока и для которых требуется высокая производительность при минимальной цене. В документации AWS EBS указаны следующие сценарии применения этого типа: потоковые чтение-запись с высоким и постоянным уровнем производительности при минимальной цене; хранение и оперирование большими данными, в том числе складирование данных, построение DataWarehouse и обработка логов. Так же как и Cold HDD, горячий HDD не может быть загрузочным (boot).
- *SSD общего назначения (general purpose SSD)* — тип твердотельного диска, подходящий для широкого класса сценариев, включая диск для операционной системы. Его производительность, выраженная в величинах IOPS (input output per second — операции ввода/вывода в секунду), в два раза превышает таковую у горячего HDD. Этот тип диска рекомендуется для большинства случаев использования.
- *SSD с выделенной полосой пропускания (provisioned IOPS SSD)* — тип твердотельного диска, обладающий наибольшей производительностью чтения-записи (и как результат, наибольшей стоимостью), предназначенный для построения критически важных компонентов с жесткими требованиями к задержкам и производительности. В качестве типовых сценариев применения этого типа можно указать высоконагруженные диски баз данных (включая MongoDB, Cassandra, Microsoft SQL Server, MySQL, PostgreSQL, Oracle и др.).

# Модуль 1. Основы построения и работы с системами аналитики больших данных



## КРАТКОЕ СОДЕРЖАНИЕ:

### 1. Архитектура систем аналитики больших данных.

1.1. Облачные технологии. Модели развертывания. Способы создания ресурсов в облаке.

1.2. Безопасность облачных ресурсов

1.3. Большие данные и источники данных. Форматы. Преобразование данных из различных форматов. Режимы обработки больших данных.

### 2. Хранение больших данных в облаке.

2.1. Хранилища общего назначения. Форматы хранения данных. Облачное хранилище Microsoft Azure Storage. Облачное хранилище AWS.

2.2. Реляционные базы данных. Azure SQL. AWS RDS.

2.3. Нереляционные базы данных. Сервисы нереляционных баз данных от Azure и AWS.

2.4. Корпоративные хранилища данных (DWH). Azure SQL DWH. AWS RedShift.

2.5. Хранилища данных типа Data Lake. Azure Data Lake Store. AWS Data Lake Solutions.

# ЛЕКЦИЯ 8:

Хранение больших данных в облаке  
Реляционные базы данных.  
Azure SQL. AWS RDS.



# Общие сведения о реляционных базах данных

**Реляционные базы данных (РБД)** на сегодняшний день самый распространенный вид баз. Они подходят для хранения данных в совершенно разных информационных системах благодаря своей универсальности и удобству.

Информация в них логически представлена в виде набора таблиц, состоящих из строк и столбцов. Эти таблицы связаны друг с другом с помощью ключей — специальных ограничений целостности, соотносящих строки одной таблицы со строками другой.

Универсальность состоит в том, что информацию об объектах и событиях реального мира можно представить как записи таблицы (что и было показано в предыдущей части книги). Любые списки любых объектов (клиентов, адресов, заказов и др.) — наиболее широко используемая абстракция в информационных системах. И точно так же, как и между объектами в реальном мире, между таблицами могут существовать различные связи и отношения. Потому-то РБД так и называются (relational — «реляционные»). Типы связей могут быть следующими:

«один к одному», «один ко многим» и «многие ко многим».

«Один к одному» — одной записи в таблице соответствует строго одна запись в другой. Пример — связи между таблицами клиентов и таблицей адресов при условии, что у каждого клиента один адрес.



# Общие сведения о реляционных базах данных

«Один ко многим» — одной записи в таблице А может соответствовать одна или несколько записей в таблице Б, но каждая запись таблицы Б связана ровно с одной записью таблицы А. Пример — связь таблицы клиентов с таблицей заказов при условии, что клиенты могут делать много заказов. Но при этом каждый заказ относится строго к одному клиенту.

«Многие ко многим» — одной записи в таблице А могут соответствовать одна или несколько записей в таблице Б и наоборот.

Пример — отношения, моделируемые таблицей студентов и таблицей учебных предметов. Очевидно, что на каждый предмет может быть записано несколько студентов, равно как каждый студент должен пройти обучение по нескольким предметам. Вообще, прямое использование такой структуры встречается редко, поскольку ведет к запутанной топологии и проблемам с производительностью при выполнении запросов данных. Чаще всего применяется промежуточная таблица-связка, состоящая как минимум из трех полей: первичного ключа (номера записи в таблицы связки), внешнего ключа таблицы А (номера студента) и внешнего ключа таблицы Б (номера курса).

Чтобы обеспечить возможность выборки информации из таблиц, РБД содержат процессор языка запросов. Он компилирует команды языка запросов SQL в программные инструкции доступа к данным в таблицах. Сам синтаксис языка может отличаться у различных СУБД, которые предоставляют различные «диалекты» (например, PL/SQL, T-SQL и др.), но все эти вариации построены как расширения базового синтаксиса SQL.

РБД выполняют две основные функции.

- Во-первых, предоставляют хранилище данных, которые могут быть добавлены, обновлены или удалены с максимально возможной защитой от несогласованности при одновременном доступе к ним из разных источников.
- Во-вторых, РБД позволяют выбирать данные из различных таблиц, в том числе с целью представить результаты аналитических запросов (для целей BI, построения отчетов и пр.). Подобная функция называется OLAP — оперативная аналитическая обработка данных. В отличие от OLTP для OLAP на первое место выходит производительность запроса произвольной формы.

# Общие сведения о реляционных базах данных

Очень важный компонент таблицы — **первичный ключ**, обеспечивающий уникальность записи в таблице. Может быть представлен как отдельным полем, так и составным. В любом случае необходим механизм, обеспечивающий уникальность ключа. Можно задействовать сторонний сервис, который генерирует и хранит ключи для таблиц. Но в этом случае вся сложность по поддержанию целостности системы ложится на разработчика, использующего РБД.

Для обеспечения уникальности ключа отлично подходит уникальный идентификатор — GUID, а также целочисленная величина. В ряде случаев подойдет временно́й штамп.

Сложности поддержания целостности системы и уникальности ключа при использовании внешнего сервиса генерации ключей могут быть исключены в случае применения автоинкрементных типов данных для ключей, обеспечивающих уникальность на уровне БД. Чтобы обеспечить связь строки одной таблицы со строкой или строками другой, необходимо иметь столбец, содержащий номер первичного ключа внешней таблицы. Такой номер называется **внешним ключом**.

Следующий очень важный момент, обеспечивающий высокую производительность запросов к одной или нескольким таблицам, — использование **индексов**. Рассмотрим, как это работает. Строки в таблице физически хранятся в произвольном порядке. Это значит, что для построения запроса на выборку данных, отвечающих какому-либо критерию фильтрации, необходимо последовательно просмотреть всю таблицу.

# Общие сведения о реляционных базах данных - индексация

**Индексация** — упорядочение строк таблицы и представление их в виде структуры данных, позволяющей применять быстрые алгоритмы поиска. Это могут быть те или иные разновидности сбалансированного дерева или хеш-таблиц, что в итоге дает скорость  $O(N \log N)$  и даже быстрее.

Индекс, физически упорядочивающий строки таблицы, называется *кластерным* и у таблицы может быть только один.

Но существуют также *некластерные* индексы, которые содержат только указатели на строки таблицы и не требуют физического упорядочения строк. В подобном случае индексы — это указатели на объекты в «куче» (строки в таблице).

В отличие от кластерного индекса некластерных в таблице может быть много. Если строки таблицы в информационной системе интенсивно обновляются, удаляются и добавляются, то индексы необходимо периодически пересчитывать.

Помимо хранения данных в таблицах и выполнения запросов к ним, реляционные БД предоставляют транзакционный механизм добавления и обновления информации в них. Ключевым понятием тут является *транзакция*. В данном случае под транзакциями понимается группа операций (обновления, удаления и пр.) которые или выполняются все вместе или не выполняются вовсе.

# Общие сведения о реляционных базах данных - индексация

Требования к транзакционным системам можно выразить с помощью принципа ACID:

atomicity — «атомарность»,

consistency — «согласованность»,

isolation — «изолированность»,

durability — «устойчивость».

Очень часто в информационных системах данные хранятся как набор множества предметно-ориентированных таблиц, каждая из которых представляет собой хранилище данных конкретной сущности в максимально нормализованном виде. Преимущество такой схемы в ее простоте, возможности простого использования совместно с объектно-реляционными библиотеками (например, EntityFramework в .NET).

В облачных средах РБД представлены в трех видах:

SQL as a Service — бессерверный сервис, как готовые образы для виртуальных машин и как Docker-образы, развернутые в кластерах AWS ECS или Azure Service Fabric.

# Microsoft Azure SQL

В настоящее время платформа Microsoft Azure позволяет использовать как виртуальные машины с готовыми предустановленными серверами SQL, так и РБД-сервисы. К последним на настоящий момент (2018 год) можно отнести три: MySQL, PostgreSQL и Azure SQL.

Первые два предоставляют сервис на основе баз данных MySQL и PostgreSQL и находятся в настоящее время в стадии preview, а это значит, что их наполнение и функциональность со временем может меняться.

Самый технически зрелый и долгоживущий сервис — **Azure SQL**, представляющий собой облачный сервис РБД на основе движка Microsoft SQL Server.

В языке запросов Azure SQL реализовано подмножество функций T-SQL. Экземпляры сервисов Azure SQL, являющиеся прямыми аналогами баз MS-SQL, логически группируются в серверы Azure SQL Server.

Каждый такой сервер должен располагать уникальным URL, учетными данными (имя пользователя и пароль), а также набором допустимых IP-адресов, которые могут иметь доступ к нему (этот список формируется в фаерволе сервера и регулирует правила доступа к нему).

Физически Azure SQL Server размещается в ЦОДе, расположенном в определенном географическом регионе, и в этом же регионе размещаются все экземпляры баз данных Azure SQL. Возможна также географическая репликация баз в несколько регионов по схеме Primary-Secondary или Primary — ReadOnly Replica (первичный сервер — реплика, доступная только для чтения). Это позволяет увеличить надежность и доступность баз.

# Microsoft Azure SQL

Благодаря широкой поддержке T-SQL Azure SQL предоставляет возможность прямой миграции баз данных из Microsoft SQL Server в Azure SQL. Конечно, прямая миграция из одного типа базы в другую — далеко не простая и не быстрая задача, но в данном случае это принципиально возможно и напрямую поддерживается с помощью специальных программ от Microsoft, Red Gate и др.

Каждый экземпляр баз данных имеет определенный ценовой уровень (pricing tier), который характеризуется производительностью, ограничениями по размеру, количеству точек восстановления и возможностями репликации.

Все потенциальные ценовые уровни разделены на базовые (Basic), стандартные (Standard) и премиум (Premium). Самое главное различие ценовых уровней проявляется в разном значении **DTU — обобщенного параметра, характеризующего производительность БД.**

Что же это за параметр?

DTU (eDTU, elastic DTU) — интегральная характеристика производительности БД, включающая в себя показатели производительности центрального процессора, памяти, устройств ввода-вывода и сетевого интерфейса.

DTU определяет своего рода объем, который может занимать производительность Azure SQL. Если в данный момент в базе выполняется запрос, то он потребляет определенное количество ресурсов, занимающих часть этого разрешенного объема (рис. 5.3).

# Графическое представление DTU

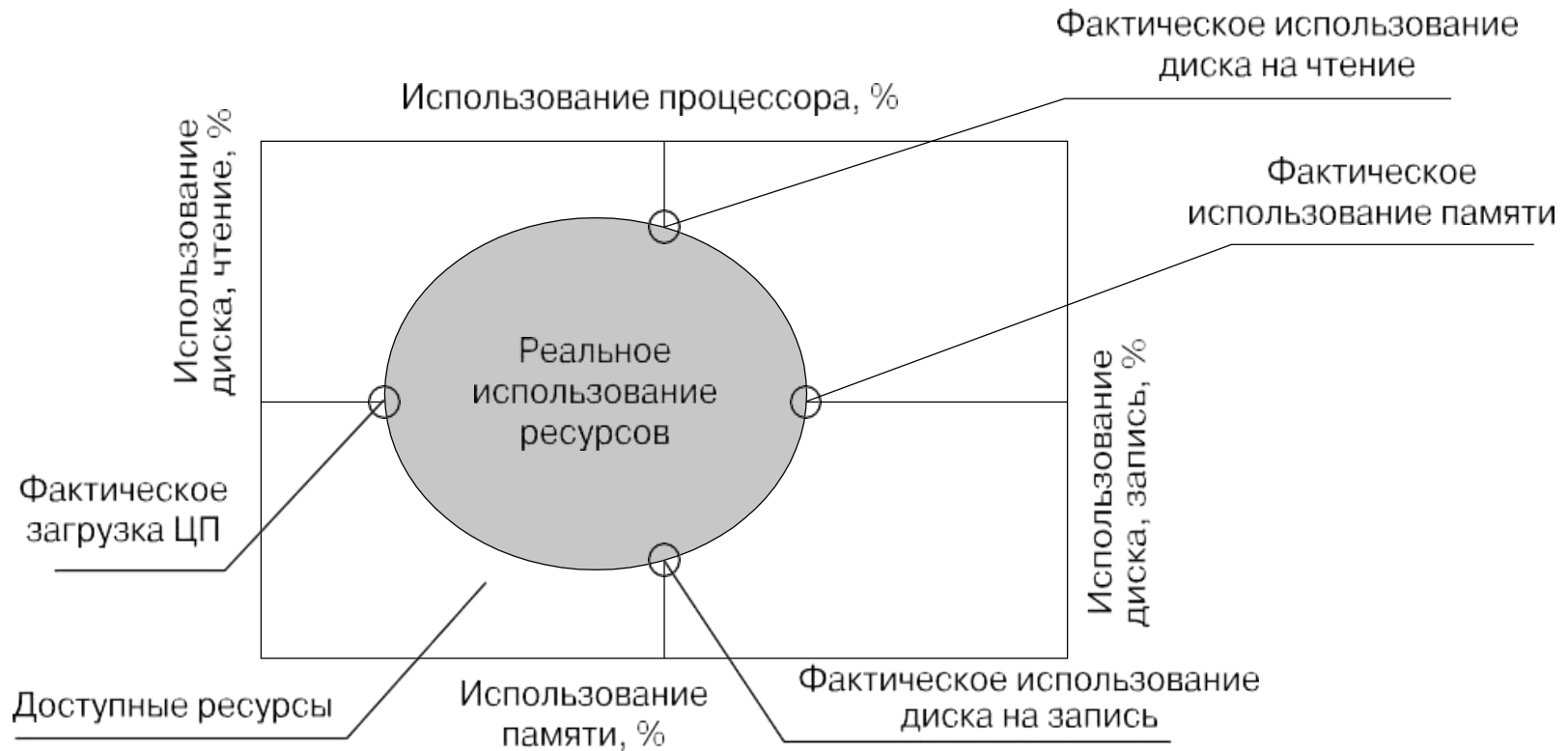


Рис. 5.3. Графическое представление DTU

При этом следует иметь в виду, что ограничивается не только объем, но и конкретные значения каждого из показателей (то есть не получится «обменять» крайне малое использование CPU на крайне большое значение памяти).

Это ограничение проявляется, в частности, в том, что один «кривой» запрос может вызвать переиспользование одного из ресурсов и в итоге «подвесить» всю базу (экземпляр сервиса Azure SQL Database, но не Azure SQL Server), и такой же запрос будет работать нормально в традиционной БД Microsoft SQL Server.

# Microsoft Azure SQL

Итак, концептуально Azure SQL состоят из:

- экземпляров **Azure SQL Database**, являющихся собственно реляционными хранилищами информации с определенными значениями размера и максимального DTU;
- экземпляра **Azure SQL Server**, который группирует базы Azure SQL Database, обеспечивая им общую строку соединения (connection string);
- **правил доступа**, прописанных в фаерволе;
- **эластичного пула ресурсов** (elastic database pool);
- **секционированной базы данных** (sharded database);
- **реляционного хранилища данных** (Azure SQL DWH).

Рассмотрим эти сервисы по порядку.

Чтобы создать экземпляр Azure SQL, необходимо сначала в левом верхнем углу портала выбрать ссылку +New и затем в появившемся окне щелкнуть на ссылке SQL Database. После этого откроется страница с формой конфигурирования сервиса (рис. 5.4).



# Microsoft Azure SQL

The screenshot displays the Microsoft Azure SQL configuration interface, divided into three main sections:

- SQL Database:** Contains fields for Database name (pokerexampleclient), Subscription (Pay-As-You-Go), Resource group (PockerRumExample), Select source (Blank database), Server (Configure required settings), Want to use SQL elastic pool? (Not now), Pricing tier (Configure required settings), and Collation (SQL\_Latin1\_General\_CP1\_CI\_AS). A 'Create' button is at the bottom.
- Server:** Shows a 'Create a new server' button and a message 'No servers found'.
- New server:** Contains fields for Server name (pokerexample), Server admin login (bigboss), Password, Confirm password, Location (Central US), and a checked checkbox for 'Allow azure services to access server'. A 'Select' button is at the bottom.

Рис. 5.4. Начальная страница конфигурирования сервиса Azure SQL

В качестве имени базы данных (Database name) указываем pokerexampleclient.

Выбираем существующую группу ресурсов (Resource group) — PockerRumExample. В качестве источника базы (Select source) указываем Blank database. Поскольку у нас нет сервера БД, его следует создать.

Для этого указываем имя сервера (Server name) — pokerexample, учетные данные (Server admin login и Password) пользователя- администратора и его местоположение (Location) — Central US.

# Microsoft Azure SQL

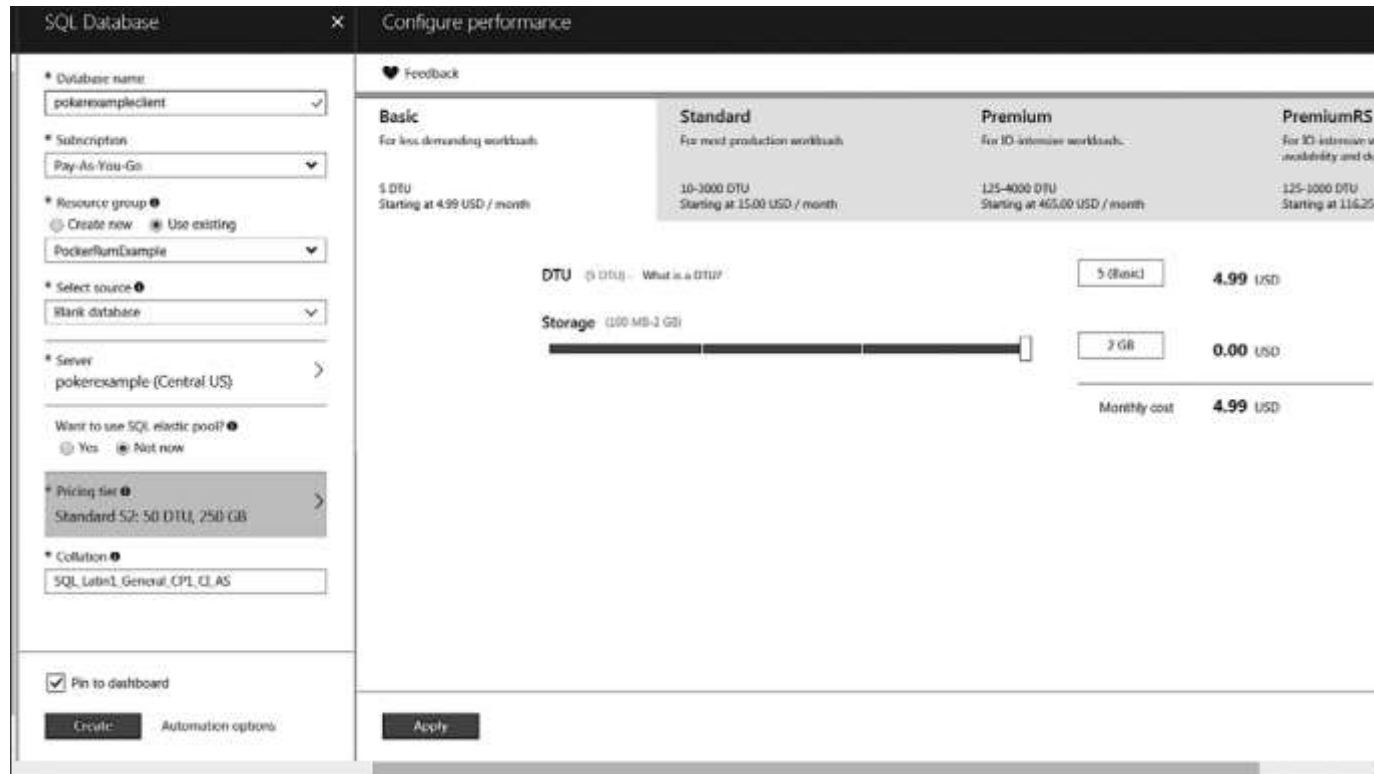


Рис. 5.5. Настройка ценового уровня базы данных

Далее следует выбрать ценовой уровень (Pricing tier) (рис. 5.5). На уровне Basic можно только изменить размер базы данных.

Для других уровней (Standard, Premium) доступна более granулярная настройка размера и производительности, включая настройку подуровней. После создания в сервисе Azure SQL доступны различные дополнительные сервисы, рассмотрим наиболее важные и полезные из них (рис. 5.6).

# Microsoft Azure SQL

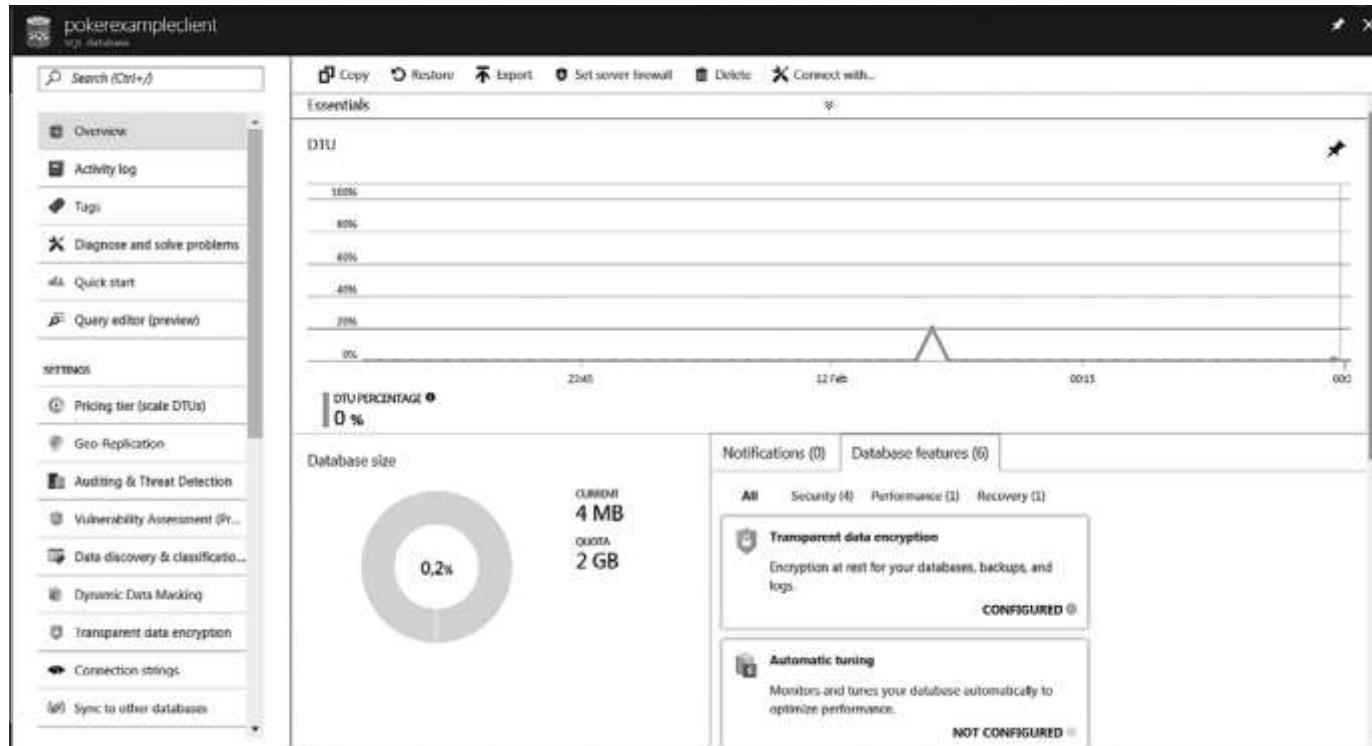


Рис. 5.6. Общая страница управления и мониторинга Azure SQL

После создания в сервисе Azure SQL доступны различные дополнительные сервисы, рассмотрим наиболее важные и полезные из них (рис. 5.6).

Чтобы получить доступ к базе данных извне, необходимо прежде всего сконфигурировать фаервол (рис. 5.7). Ссылка доступа к ней находится в верхней части портала (Set server firewall).

# Firewall сервера Azure SQL

Чтобы получить доступ к базе данных извне, необходимо прежде всего скон- фигурировать фаервол (рис. 5.7). Ссылка доступа к ней находится в верхней части портала (Set server firewall).



Рис. 5.7. Фаервол сервера Azure SQL

# Доступ к БД Microsoft Azure SQL извне

Чтобы получить доступ к БД извне, нужно получить строку подключения с учетными данными пользователя. Для этого следует нажать ссылку Connecting string на левой навигационной панели заглавной страницы (рис. 5.8).

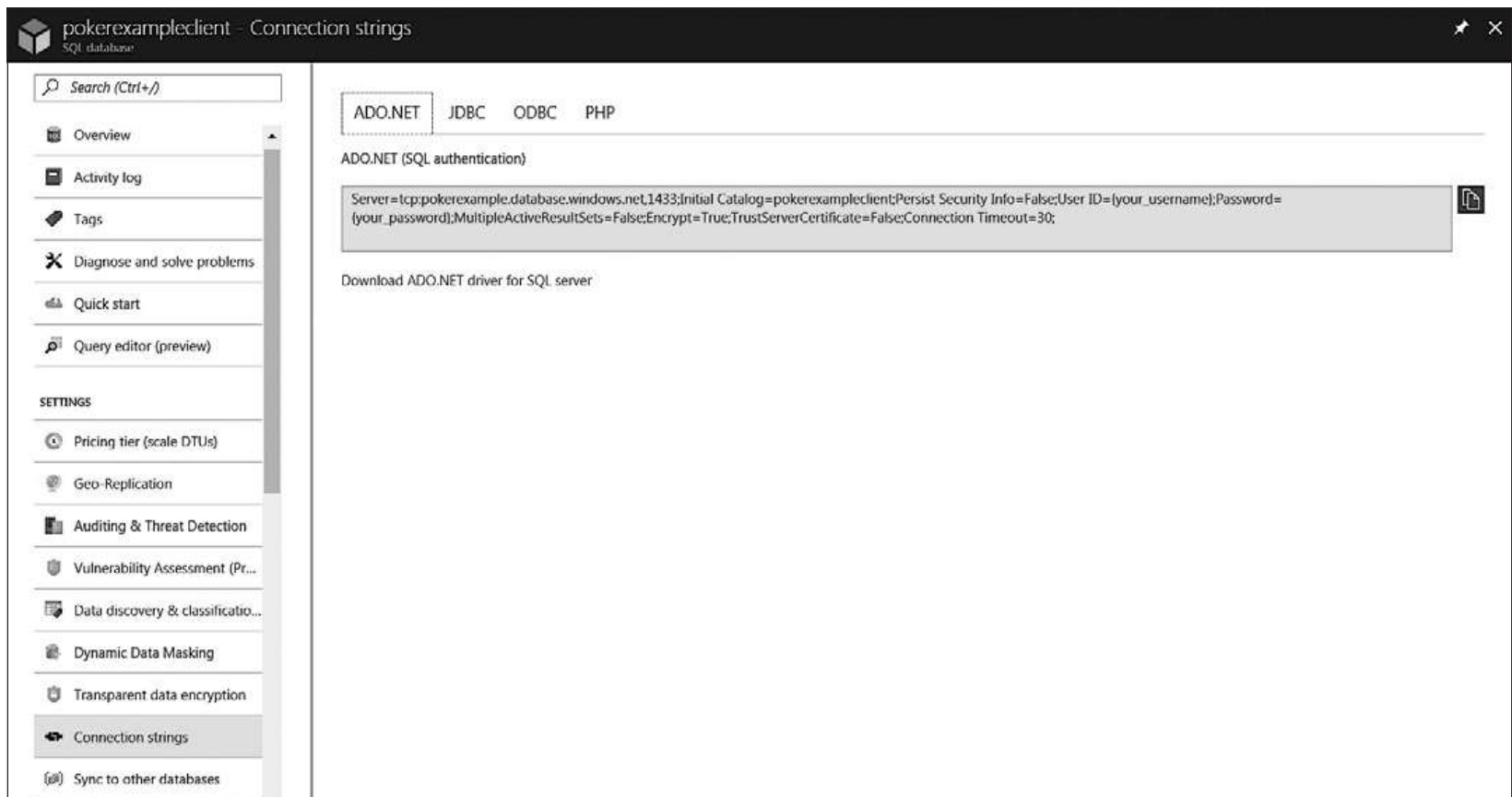


Рис. 5.8. Получение строки подключения для базы данных

# Microsoft Azure SQL

Теперь рассмотрим случай, когда база данных имеет ярко выраженные и узкие пики (*spikes*). Подобная ситуация неприятна тем, что ради обеспечения достаточной производительности БД необходим ее ценовой уровень, слегка превышающий уровень этих пиков.

Но если такие пики достаточно редки, то ресурсы базы задействуются достаточно нерационально: бо' льшую часть времени она работает существенно недогруженной (с малым уровнем DTU), а во время пиков — зачастую перегруженной (DTU близка к 100 %).

Эта нерациональность в конечном итоге приводит к неоправданно большому счету за облачные ресурсы. Логичнее и эффективнее всего в подобном случае использовать сервис **Query Performance Insight** (рис. 5.9), постараться определить, какие запросы в базе приводят к таким пикам, и попытаться устранить их за счет оптимизации запросов, список которых выдает упомянутый сервис.

Наряду с этим доступна возможность автоматической настройки производительности с помощью сервиса **Automatic Tuning** (рис. 5.10).

# Внешний вид вкладки сервиса Query Performance Insight Performance Insight Microsoft Azure SQL

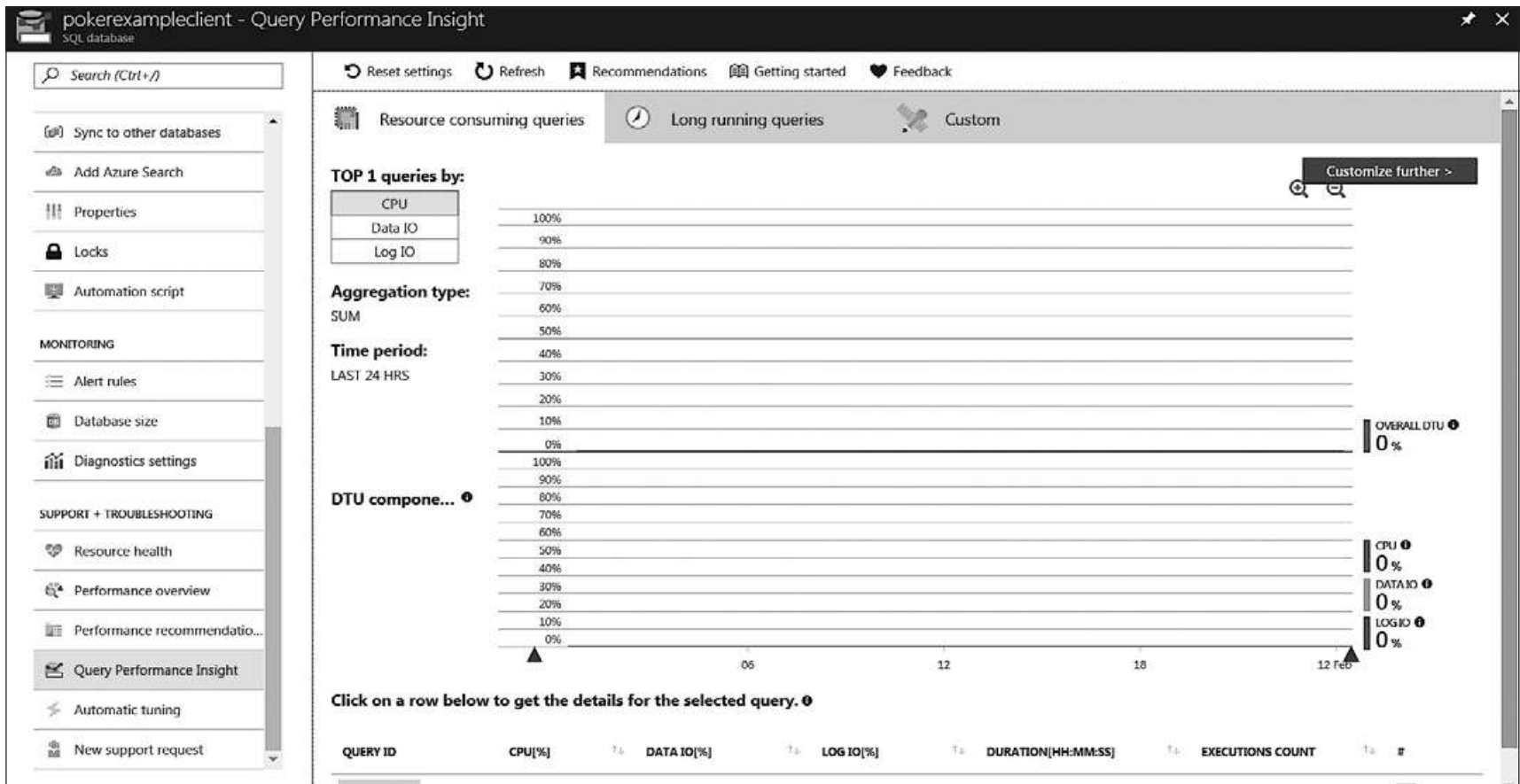


Рис. 5.9. Внешний вид вкладки сервиса Query Performance Insight

# Сервис автоматической оптимизации производительности Microsoft Azure SQL

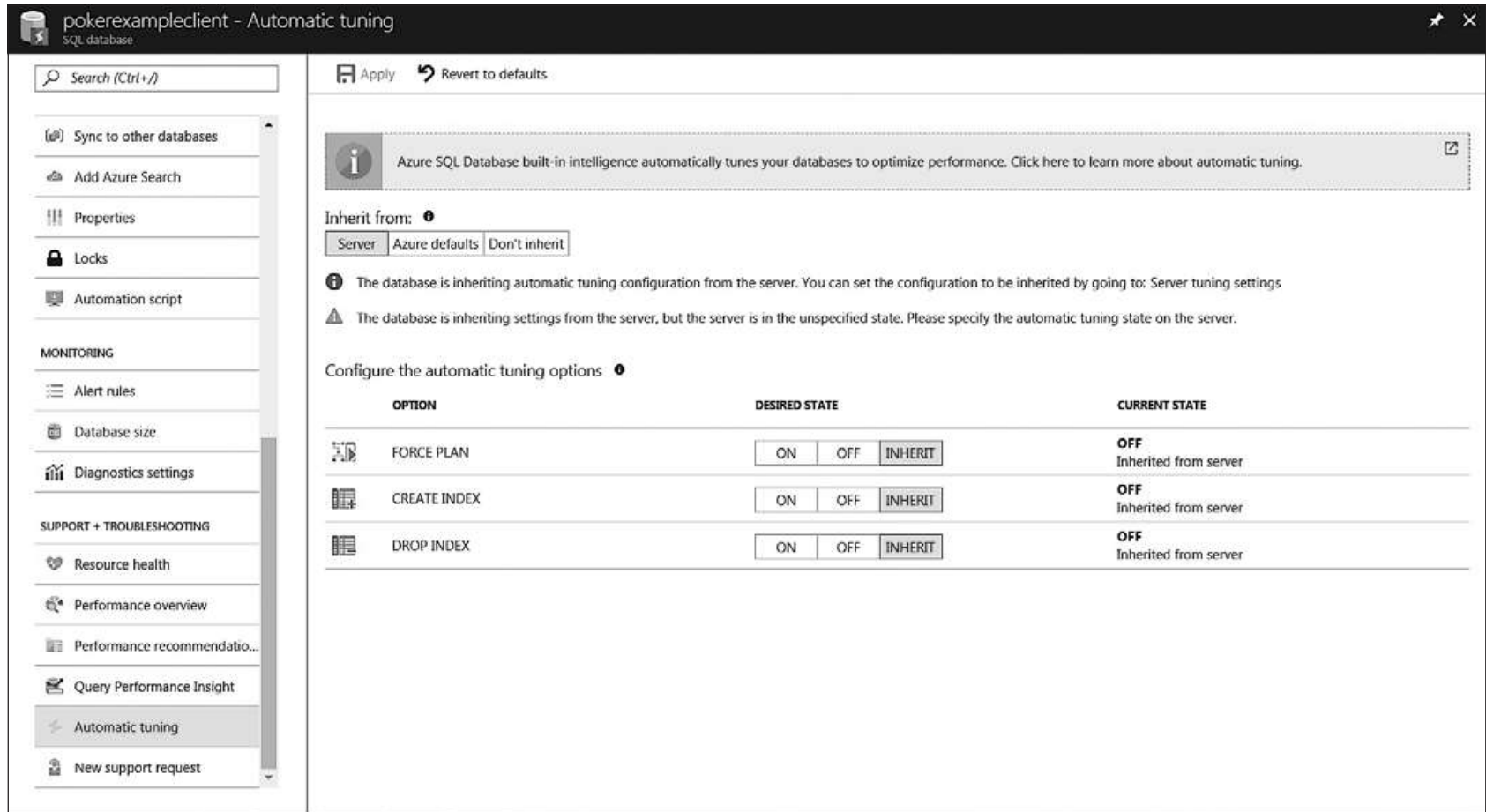


Рис. 5.10. Сервис автоматической оптимизации производительности



# Сервис автоматической оптимизации производительности Microsoft Azure SQL

Автоматическая настройка производительности состоит в динамическом добавлении/удалении индексов, а также перекомпилировании плана исполнения. Каждый акт срабатывания автоматической настройки отображается в логах. Помимо этого, Microsoft предоставляет целый ряд сервисов подстройки и автоматической оптимизации базы данных (Performance overview, Performance recommendations и пр.), на которые следует обратить внимание при реальном функционировании БД.

Кроме того, через веб-портал доступен еще один из сервисов Azure SQL — Azure SQL Query Editor, который позволяет прямо на веб-портале писать запросы SQL к БД и редактировать данные в браузере. Встроенный редактор запросов выглядит следующим образом (рис. 5.11).

# Встроенный редактор запросов Azure SQL

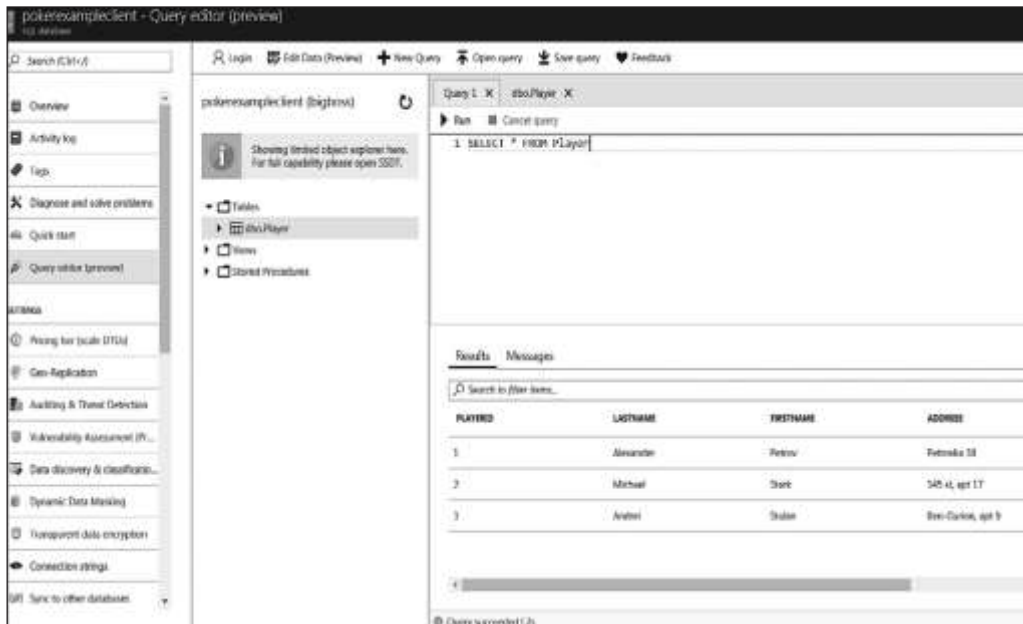


Рис. 5.11. Встроенный редактор запросов Azure SQL

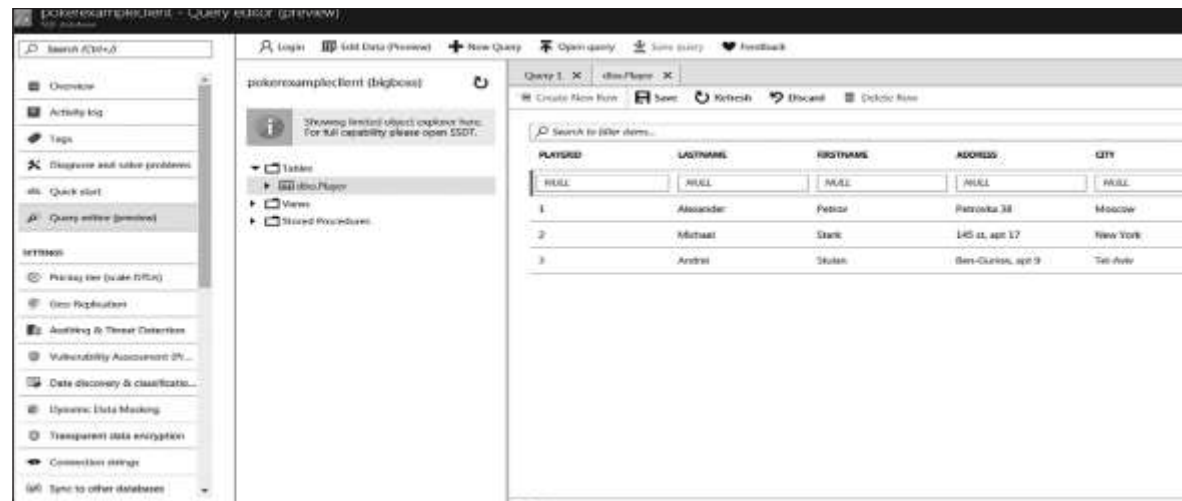


Рис. 5.12. Вкладка редактирования данных в таблице

# Microsoft Azure SQL

Теперь рассмотрим очень важный случай работы *нескольких баз данных Azure SQL на одном сервере*. В реальных проектах нагрузка на базы может быть неравно- мерной не только из-за наличия пиков, обусловленных проблемами с запросами, но и из-за периодических пиков запросов пользователей.

Если им соответствует строгая периодичность (скажем, большая нагрузка днем и маленькая ночью), то можно настроить автоматическое масштабирование базы данных по расписанию, например, с помощью сервиса Azure Automation.

Еще один случай — некоррелированные запросы могут послужить причиной использования Azure Elastic Database Pool, который встроен в состав Azure SQL Server и служит для объединения баз в один пул и назначения всем им общих разделяемых ресурсов сервера. Рассмотрим ситуацию более детально на примере SaaS-приложения, созданного для оказания неких услуг зарегистрированным в нем пользователям (допустим, это облачная CRM-система).

Каждому из пользователей выделяется своя БД определенного уровня производительности, который выражается конкретным значением DTU.

## Сервиса Azure Elastic Database Pool

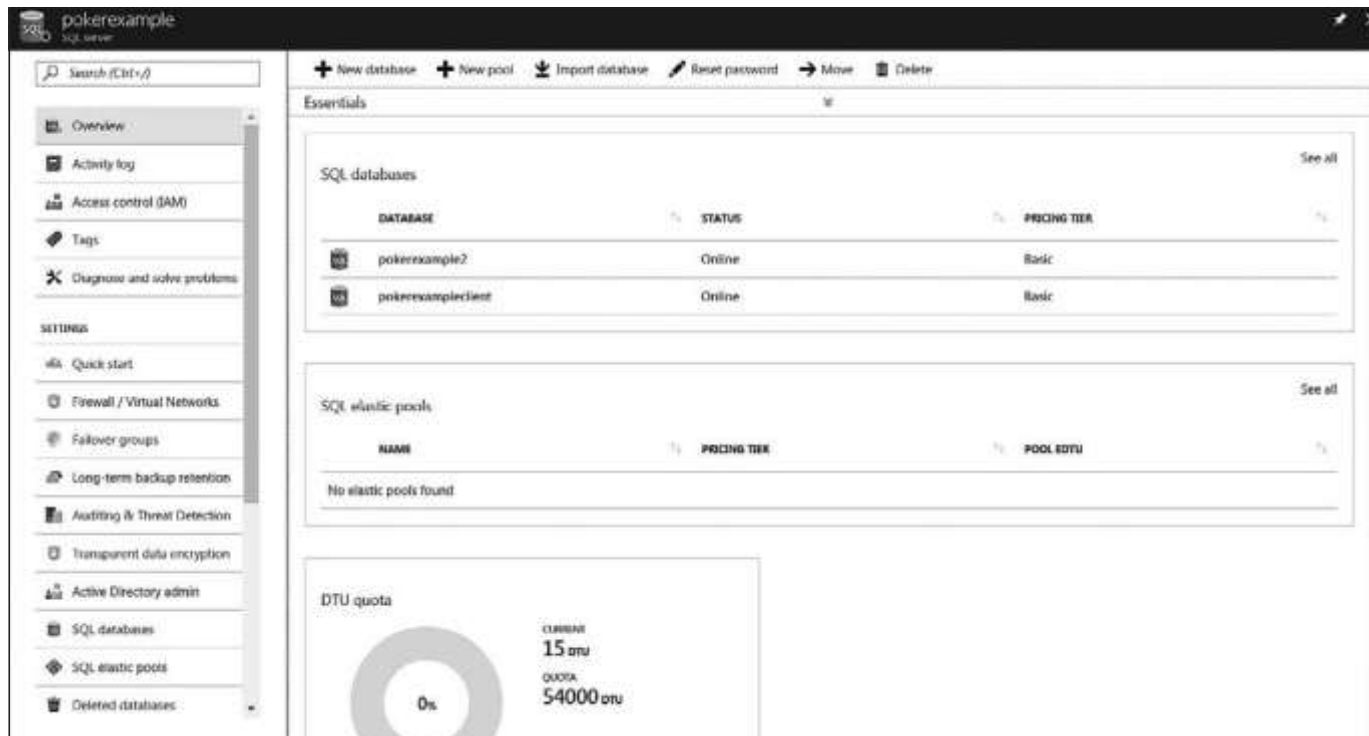
Выбор eDTU (elastic DTU) производится с помощью приближенного соотношения:  
Суммарное eDTU = MAX(<Общее количество баз данных ´ Среднее использование eDTU>, <Количество баз данных одновременно достигающих пиковых нагрузок ´ Пиковый уровень DTU базы>)

# Сервис Azure Elastic Database Pool

Далее для определения минимального объема хранилища необходимо просуммировать объемы всех отдельных баз данных. Затем следует выбрать ценовой уровень, дающий максимальное значение eDTU, полученное исходя из формулы и размера хранилища.

Чтобы добавить Elastic Database Pool, нужно выйти на начальную страницу сервера pokerexample. Для добавления нового Elastic Pool на этой странице следует нажать на ссылку + Elastic pool (рис. 5.14).

На рис. 5.15 показана страница создания и конфигурирования Elastic Pool.



The screenshot shows the Azure portal interface for a SQL server named 'pokerexample'. The left sidebar contains navigation options like Overview, Activity log, Access control (IAM), Tags, and various settings. The main content area is titled 'Essentials' and includes a toolbar with actions like 'New database', 'New pool', 'Import database', 'Reset password', 'Move', and 'Delete'. Below the toolbar, there are two tables: 'SQL databases' and 'SQL elastic pools'. The 'SQL databases' table lists two databases: 'pokerexample2' and 'pokerexample1', both with a status of 'Online' and a pricing tier of 'Basic'. The 'SQL elastic pools' table shows 'No elastic pools found'. At the bottom, there is a 'DTU quota' section with a circular progress indicator showing 0% usage, a current usage of 15 DTU, and a quota of 54000 DTU.

DATABASE	STATUS	PRICING TIER
pokerexample2	Online	Basic
pokerexample1	Online	Basic

NAME	PRICING TIER	POOL EDTU
No elastic pools found		

DTU quota

CURRENT: 15 DTU  
QUOTA: 54000 DTU

Рис. 5.14. Стартовая страница сервера со списком доступных баз данных

# Страница создания и конфигурирования Elastic Pool

**Elastic database pool** ×

**Choose your pricing** ×

Each pricing tier offers a range of databases, eDTUs and storage. Make a selection after reviewing the pricing card ...

Elastic database pool allows multiple databases to share elastic database transaction units (eDTUs), and storage (GBs). Learn more ↗

An elastic database pool provides elastic database transaction units (eDTUs), and storage (GBs) that are shared by multiple databases.

Learn more ↗

\* Name  
pokerdatabases ✓

Pricing tier  
Standard Pool >

Configure pool  
100 eDTU, 100 GB pool, 0 dat... >

Summary

Pool settings	100 eDTU, 100 GB
Number of Databases	0
Per database settings	0-100 eDTU

Cost

eDTU	2,2475 USD x 100 eDTU = 224.75
Storage	0.0850 USD x 0 GB = 0
<b>Total Cost</b>	<b>224.75</b>

USD/Month/Estimated 31 days

Pin to dashboard

OK

**B Basic Pool**

<b>50</b>	50 to 1600 eDTU/Pool
⚙	Up to 156.25 GB/Pool
⚙	Up to 500 DBs/Pool
🗄	Up to 5 eDTU/DB
🗄	Up to 2 GB/DB
⚠	Pay per Pool eDTU
<b>1,50</b>	USD PER EDTU/MONTH (ESTIMATE...)

**S Standard Pool**

<b>100</b>	50 to 3000 eDTU/Pool
⚙	Up to 4096 GB/Pool
⚙	Up to 500 DBs/Pool
🗄	Up to 3000 eDTU/DB
🗄	Up to 1024 GB/DB
⚠	Pay per Pool eDTU
<b>2,25</b>	USD PER EDTU/MONTH (ESTIMATE...)

**P Premium Pool**

<b>125</b>	125 to 4000 eDTU/Pool
⚙	Up to 4096 GB/Pool
⚙	Up to 100 DBs/Pool
🗄	Up to 4000 eDTU/DB
🗄	Up to 1024 GB/DB
⚠	Pay per Pool eDTU
<b>5,58</b>	USD PER EDTU/MONTH (ESTIMATE...)

Select

Рис. 5.15. Страница создания и конфигурирования Elastic Pool

# Страница создания и конфигурирования Elastic Pool

The screenshot displays the configuration interface for an Elastic Database Pool in the Azure portal, divided into two main sections: 'Configure pool' and 'Add databases'.

**Configure pool (Elastic database pool):**

- Buttons: Remove from pool, Add databases, Feedback.
- Chart: Elastic pool estimated eDTU and storage usage for last 14 days. The y-axis ranges from 0 to 60. The chart shows a shaded area representing usage, with a dashed line at the top. Below the chart, the values are: POOLED eDTU: 50, POOL GB: 4.88.
- Settings: Elastic database pool settings. Pool eDTU is set to 50. Pool GB is set to 4.88.
- Costs: EDTU COST and STORAGE COST. Total cost is USD 75.02.
- Button: Select.

**Add databases (Elastic database pool):**

- Buttons: Select all, Search to filter databases...
- Summary: Selected/Total databases: 0/2.
- Table: List of databases to be added.

	DATAB...	PRICIN...	PEAK D...	AVG D...
	pokerexa...	Basic	--	--
	pokerexa...	Basic	0	0

**Per database settings (Database):**

- EDTU min: 0, EDTU max: 5, EDTU avg: --.
- EDTU max: 5.
- EDTU min: 0, 5.
- Button: Select.

Рис. 5.16. Дальнейшее конфигурирование пула

# Сервис Azure Elastic Database Pool

Добавление баз данных в пул и удаление их из него может происходить *динамически* без остановки и передеплоивания баз данных. Для этого в панели управления есть графики мониторинга и сервис выдачи рекомендаций по включению баз данных в пул и исключению из него.

Кроме того, существует сервис *Elastic Query*, позволяющий выполнять задания, общие для всех баз данных из *Elastic Database Pool*. Суть его в том, что группа баз объединяется и управляется централизованно с общего специализированного головного сервера VM. Взаимодействуя с этим сервером программно или через Azure Portal, можно управлять всеми подключенными БД с помощью создания *заданий (jobs)*.

## Задания могут быть такими:

- административные (согласованное изменение схемы, выполнение перекомпиляции индексов);
- периодическое обновление данных или их сбор для систем BI, в том числе для выполнения анализа большого объема информации.
- Сервис Elastic Database jobs содержит следующие компоненты:
- головной сервер, размещаемый на экземпляре Azure Cloud Service Worker Role. По сути, это специализированное ПО, размещаемое (по состоянию на настоящее время) на виртуальной машине Azure Cloud Service. Для обеспечения высокой доступности рекомендуется создавать минимум два экземпляра VM;
- управляющую (головную) БД — экземпляр Azure SQL, служащий для хранения метаданных всех подключенных баз;
- экземпляр сервиса Azure Service Bus, служащий для объединения и синхронизации всех компонентов;
- экземпляр облачного хранилища Azure Storage Account, служащий для хранения журналов всей системы.

# Свойство Elastic Database Pool

Очень важное свойство Elastic Database Pool — возможность реализовывать сценарии разбиения одной большой БД на меньшие, но выполнять запросы в разделенной на фрагменты (shard) базе так, будто это одна монолитная база (рис. 5.17).

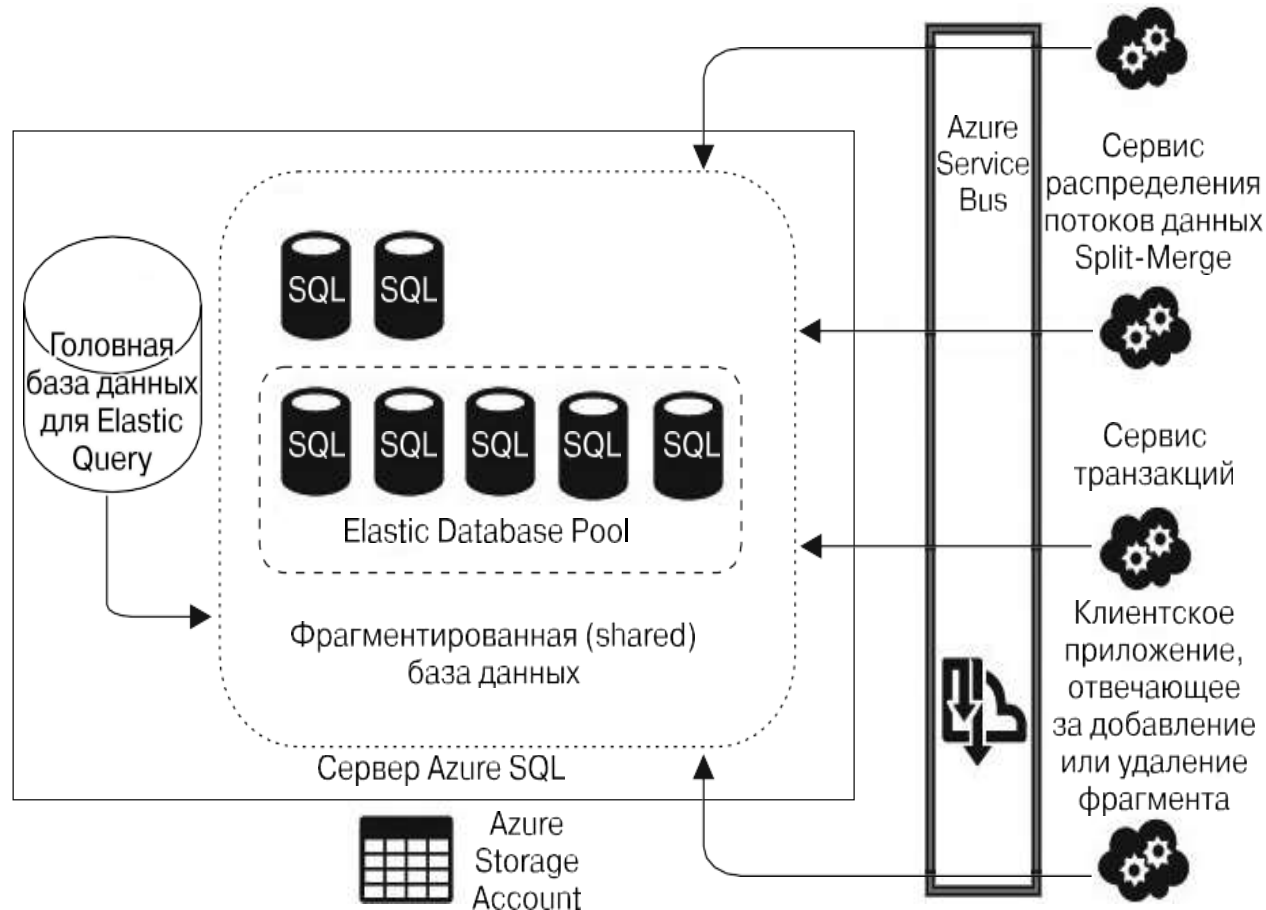


Рис. 5.17. Общая структура фрагментированной базы данных, построенной на основе Elastic Database Pool



# Свойство Elastic Database Pool

Рассмотрим данный механизм подробнее.

Нужда во фрагментации БД появляется в случае, когда ее размер становится чрезмерно большим для размещения на одном экземпляре Azure SQL (в настоящее время это более 1 Тбайт для ценового уровня Premium). Конечно, можно разбить большую БД на меньшие логически, проведя анализ ее структуры (схемы).

Как уже указывалось ранее, Azure SQL Server — логическая группировка экземпляров Azure SQL Database, а не физическое объединение на одном сервере.

И прямые запросы к объектам одной базы из другой возможны, только если объекты являются внешними таблицами или базы объединены в [Elastic Database Pool](#) и используются запросы [Elastic Database Query](#) или транзакции [Elastic Transactions](#). В таком случае необходимо применить фрагментацию (sharding) базы данных.

По сути, это горизонтальное масштабирование, отличное от вертикального — увеличения размера базы в масштабе CPU, оперативной памяти, IOPs и пр. Разделение одного большого хранилища на несколько хранилищ меньшего масштаба и их параллельная обработка с последующим сложением результатов последней — ключевая концепция всех технологий обработки больших данных, с которой мы будем не раз встречаться далее. Фрагментированная БД концептуально во многом близка к реляционному хранилищу данных DWH, но требует бо' льших усилий при создании и использовании.

# Свойство Elastic Database Query

Теперь рассмотрим случай, когда все же необходимо выполнить запрос к данным, расположенным в разных базах. При наличии простого набора БД без фрагментирования нужно выбрать одну головную базу и создать в ней внешние источники данных и внешние таблицы (reference table), являющиеся отражениями реальных таблиц, размещенных в других БД.

Можно также не использовать специализированную головную базу, а создавать таблицы (см. информацию на сайте:

<https://docs.microsoft.com/en-us/azure/sql-database/sql-database-elastic-query-getting-started-vertical>) в каждом экземпляре Azure SQL Database.

Недостаток такого подхода в том, что при смене схем таблиц в базах необходимо синхронно менять схемы во внешних таблицах (при отсутствии головной БД нужно проделать очень большую работу по смене схемы во всех внешних таблицах во всех базах, где присутствуют эти внешние таблицы).

Кроме того, у реализации T-SQL в Azure SQL есть ограничения на типы данных для внешних таблиц (например, они не поддерживают Foreign Key и тип `nvarchar(max)`).

И в настоящее время присутствует целый ряд ограничений на выполнение подобных запросов в базы данных, где эти запросы будут выполняться, — скажем, невозможно выполнить экспорт БД в *ВАСРАС*-файл при наличии в ней ссылок на внешние таблицы (<https://docs.microsoft.com/en-us/azure/sql-database/sql-database-elastic-query-overview>).

# Пример Elastic Database Query

Предположим, у нас две базы данных: First и Second. Теперь допустим, что из базы Second нужно выполнить запрос к базе First. Для этого в Second следует создать учетные данные (credentials) (рис. 5.18), которые будут служить для доступа к БД First.

```
CREATE MASTER KEY ENCRYPTION BY PASSWORD = '<пароль>';
CREATE DATABASE SCOPED CREDENTIAL FirstDBQueryCred -- Это имя учетной записи
WITH IDENTITY = '<ИмяПользователя>',
SECRET = '<пароль>';
```

Затем в базе Second нужно создать внешний источник данных, который будет использоваться для связи с таблицами внешней БД (рис. 5.19).

```
CREATE EXTERNAL DATA SOURCE FirstDatabaseDataSource WITH
(
    TYPE = RDBMS,
    LOCATION = '<имя_вашего_сервера>.database.windows.net',
    DATABASE_NAME = 'First',
    CREDENTIAL = FirstDBQueryCred
)
```

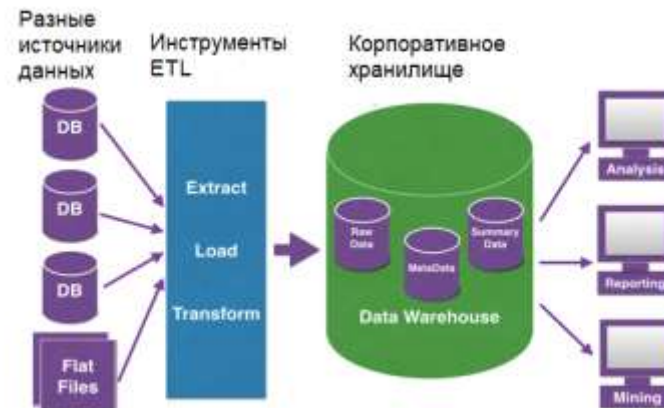
Рис. 5.19. Создание внешнего источника данных с учетными данными

```
CREATE EXTERNAL TABLE [dbo].[TableFromFirstDatabase]
( [KeyFieldID] [int] NOT NULL,
  [DataField] [varchar](50) NOT NULL
)
WITH ( DATA_SOURCE = FirstDatabaseDataSource)
```

Рис. 5.18. Создание учетных данных для базы

После этого в базе Second надо создать внешнюю таблицу — отражение аналогичной таблицы, размещенной в базе First (рис. 5.20).

# Модуль 1. Основы построения и работы с системами аналитики больших данных



## КРАТКОЕ СОДЕРЖАНИЕ:

### 1. Архитектура систем аналитики больших данных.

1.1. Облачные технологии. Модели развертывания. Способы создания ресурсов в облаке.

1.2. Безопасность облачных ресурсов

1.3. Большие данные и источники данных. Форматы. Преобразование данных из различных форматов. Режимы обработки больших данных.

### 2. Хранение больших данных в облаке.

2.1. Хранилища общего назначения. Форматы хранения данных. Облачное хранилище Microsoft Azure Storage. Облачное хранилище AWS.

2.2. Реляционные базы данных. Azure SQL. AWS RDS.

2.3. Нереляционные базы данных. Сервисы нереляционных баз данных от Azure и AWS.

2.4. Корпоративные хранилища данных (DWH). Azure SQL DWH. AWS RedShift.

2.5. Хранилища данных типа Data Lake. Azure Data Lake Store. AWS Data Lake Solutions.

# ЛЕКЦИЯ 9:

Хранение больших данных в облаке

Реляционные базы данных.

Azure SQL.

AWS RDS — Amazon Web Services

Relational Database Service.



# AWS RDS — Amazon Web Services

## Relational Database Service

Сервис реляционных баз данных от Amazon называется [AWS RDS — Amazon Web Services Relational Database Service](#).

Он значительно отличается от сервиса Azure SQL как по своей «философии», так и по составу. Прежде всего, основными логическими элементами этого сервиса является экземпляр, или инстанс [RDS \(RDS Instance\)](#). Сам экземпляр, по сути, представляет виртуальную машину, на которой размещается сервер базы данных и к которой подключен VHD. Типы этой виртуальной машины похожи на обычные EC2, но имеют ряд существенных отличий.

Прежде всего, отсутствует прямой консольный доступ к экземплярам, они не отображаются в консоли в списке EC2 и недоступны для размещения в них каких-либо иных приложений; невозможно применить модель Spot или Provisioned Instance; невозможно конфигурировать RAID-структуры. Эти экземпляры представляют собой вычислительные модули, служащие для выполнения запроса к данным, хранящимся на подключенных к ним виртуальных дисках, расположенных в хранилище S3.

Могут быть использованы три типа диска: [Magnetic](#), [General purpose SSD](#) и [Provisioned IOPS SSD](#). Изменяя тип экземпляра и виртуального диска, можно гибко подстраивать их под требования к цене и производительности для конкретной базы. При этом можно выбрать экземпляры с расширенной памятью (memory optimized), более производительным CPU (CPU optimized), более производительным интерфейсом ввода-вывода (IO optimized) или общего назначения (general purpose).

Достаточно большое количество типоразмеров экземпляров позволяет перекрыть довольно широкий диапазон нагрузок, а возможность выбирать еще и тип и размер диска обеспечивает гибкость при построении систем с заданными соотношениями «цена/производительность».

# AWS RDS — Amazon Web Services

## Relational Database Service

Очень важный параметр экземпляра — **тип сервера базы данных: движок (RDS engine)**.

Этот тип определяет конкретную технологию SQL-сервера. В настоящее время поддерживаются следующие типы движков: MySQL, PostgreSQL, MariaDB, Oracle, MS SQL (Express и Standard Edition) и Aurora.

Выбор конкретного движка не только влияет на выбор синтаксиса SQL, но и диктует набор и значение некоторых параметров самого экземпляра (например, типа экземпляра, типа диска и др.).

Далее, в экземпляре можно создать одну или несколько БД. Необходимо понимать, что эти базы данных будут разделять ресурсы экземпляра и логично размещать в экземплярах базы таким образом, чтобы последние имели некоррелированные пики активности и обеспечивали по возможности более равномерную загрузку. Само количество БД в экземпляре определяется типом движка. Так, для Oracle база может быть одна, но иметь несколько схем.

Рассмотрим на примере, как создавать экземпляры сервиса RDS (рис. 5.21).

# Как создавать экземпляры сервиса RDS

Рассмотрим на примере, как создавать экземпляры сервиса RDS (рис. 5.21).

В терминах AWS создание экземпляров баз данных называется запуском (Launch). Чтобы запустить экземпляр, нужно перейти на вкладку Instances (навигационная панель слева) и нажать ссылку Launch DB Instance. В итоге откроется панель конфигурирования, показанная на рис. 5.22.

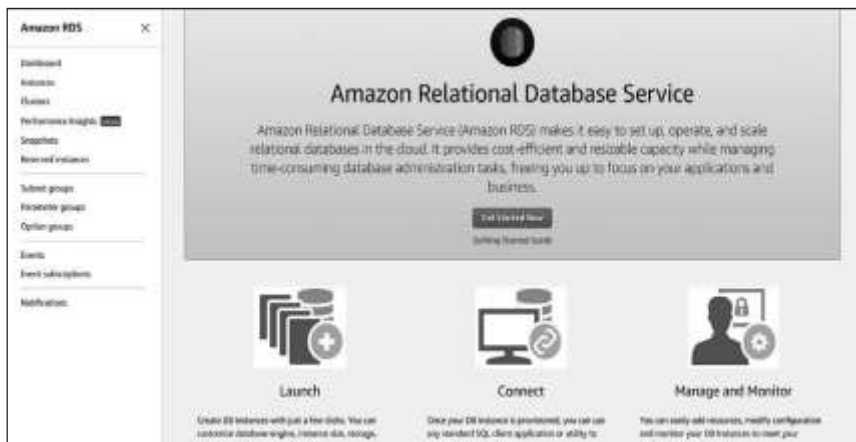
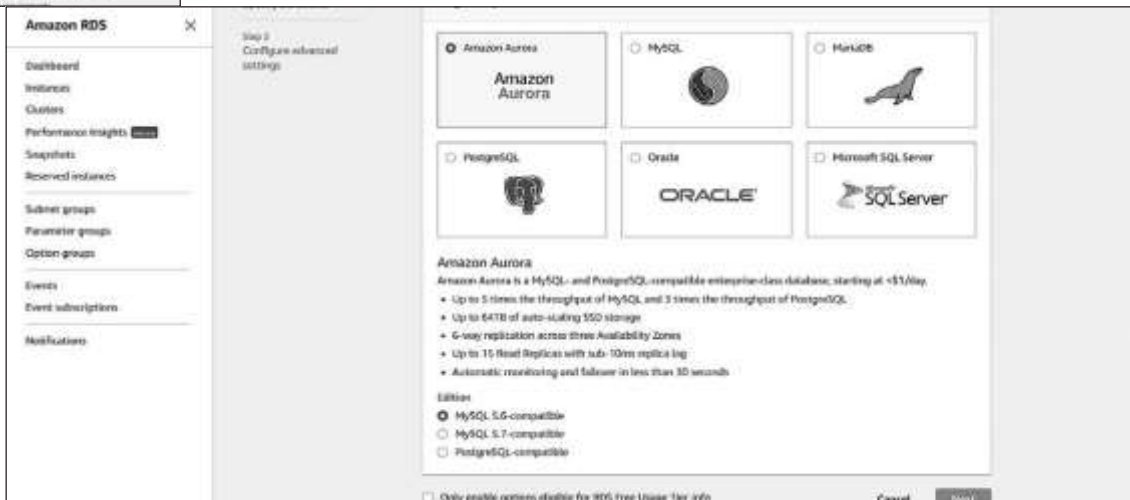


Рис. 5.21. Общий вид панели RDS

Рис. 5.22. Панель конфигурирования запускаемого экземпляра RDS





# AWS RDS — Amazon Web Services Relational Database Service

Обратите внимание: не все экземпляры доступны как Free Tier, то есть даже если вы пробуете Amazon в режиме Free Trial, далеко не все ресурсы будут действительно бесплатными. Чтобы оставить только ресурсы, доступные бесплатно, установите флажок в самом низу формы — Only enable options eligible for RDS Free Usage Tier (рис. 5.23). Дальнейший процесс создания показан на примере движка MySQL. Выбрав движок, нажмите Next.

В появившемся окне откроется форма конфигурирования экземпляра. Эта форма может слегка различаться для различных движков. На рис. 5.23 показана вся форма, сдвинутая скроллом, — слева верхняя часть, справа — нижняя. Прежде всего стоит указать модель лицензии (поле License model), затем уточнить версию движка в рамках выбранного типа (поле DB engine version). Следует заметить, что лицензирование — существенный аспект использования сервисов RDS. Одна часть движков предоставляется под лицензией GPL, а другая — Oracle, MS SQL — требует лицензионных отчислений; вследствие этого плата взимается как за вычислительные ресурсы и ресурсы хранения, так и за лицензию. Кроме того, для Oracle и MS SQL возможна оплата лицензии по принципу BYOL (Bring Your Own License — добавление лицензии, оплаченной вне AWS).

The screenshot shows the AWS RDS instance configuration form. The form is divided into several sections:

- Instance specifications:** Includes fields for DB engine (MySQL Community Edition), License model (general public license), DB engine version (mysql 5.6.17), and DB instance class (db.t2.micro). It also features a 'Free tier' section with a checkbox for 'Only enable options eligible for RDS Free Usage Tier'.
- Storage type info:** Includes a dropdown for 'General Purpose (SSD)' and a field for 'Allocated storage' (20 GB).
- Settings:** Includes fields for 'DB instance identifier', 'Master username', and 'Master password'.

Рис. 5.23. Форма конфигурирования экземпляра RDS

# AWS RDS — Amazon Web Services Relational Database Service

Следующим выбирается тип экземпляра виртуальной машины (DB instance class), которая, по сути, будет лежать в основе этой БД. Данный тип экземпляра определяет уровень производительности базы. Физически данные будут храниться на виртуальных жестких дисках. Тип диска определяется в поле Storage type (для данного случая доступен только SSD), чей размер указывается в поле Allocated storage (20 Гбайт на рис. 5.23). Далее следует указать имя идентификатора базы (DB instance identifier), учетные данные пользователя-администратора и затем нажать Next. После этого откроется панель расширенных настроек (рис. 5.24). Рассмотрим их подробнее.

The screenshot displays the 'Configure advanced settings' interface for an AWS RDS instance. It is divided into two main sections: 'Network & Security' and 'Database options'.

**Network & Security:**

- Virtual Private Cloud (VPC) info:** A dropdown menu is set to 'Default VPC (vpc-2d2f8543)'. A note states: 'Only VPCs with a corresponding DB subnet group are listed.'
- Subnet group info:** A dropdown menu is set to 'default'. A note states: 'DB subnet group that defines which subnets and IP ranges the DB instance can use in the VPC you selected.'
- Public accessibility info:** The 'No' radio button is selected. A note states: 'DB instance will not have a public IP address assigned, no EC2 instance or devices outside of the VPC will be able to connect.'
- Availability zone info:** A dropdown menu is set to 'No preference'.
- VPC security groups:** The 'Create new VPC security group' radio button is selected. A note states: 'Security groups have rules authorizing connections from all the EC2 instances and devices that need to access the DB instance.'

**Database options:**

- Database name:** A text input field contains 'database'. A note states: 'Note: If no database name is specified then no initial MySQL database will be created on the DB instance.'
- Database port:** A text input field contains '3306'. A note states: 'TCP/IP port the DB instance will use for application connections.'
- DB parameter group info:** A dropdown menu is set to 'default.mysql5.6'.
- Option group info:** A dropdown menu is set to 'default:mysql-5-6'.
- Copy tags to snapshots:** An unchecked checkbox.
- IAM DB authentication info:** The 'Disable' radio button is selected. A note states: 'Manage your database user credentials through AWS IAM roles and roles.'

Рис. 5.24. Сетевые настройки и настройки опций базы данных

# AWS RDS — Amazon Web Services

## Relational Database Service

Сетевые настройки включают в себя выбор виртуальной частной сети (VPC), подсети (subnet), сетевой группы безопасности (security group) и опции публичной доступности (public accessibility). Последний параметр означает наличие возможности подключения к базе данных ресурсов, находящихся вне ее VPC.

Опции базы данных включают указание ее имени (database name), порта доступа (для MySQL по умолчанию это 3306) и ряда других специфичных параметров. Отмечу лишь возможность интеграции с учетными данными пользователей аккаунта AWS — для этого нужно включить опцию Enable IAM DB authentication.

Следующие формы (рис. 5.25) относятся к настройкам шифрования, бэкапам, мониторингу и экспорту логов в сервис AWS CloudWatch. В этих формах все интуитивно понятно, поэтому перейдем к последней и более интересной форме — настройке обновлений операционной системы и движка базы данных (рис. 5.26).

# AWS RDS — Amazon Web Services

## Relational Database Service

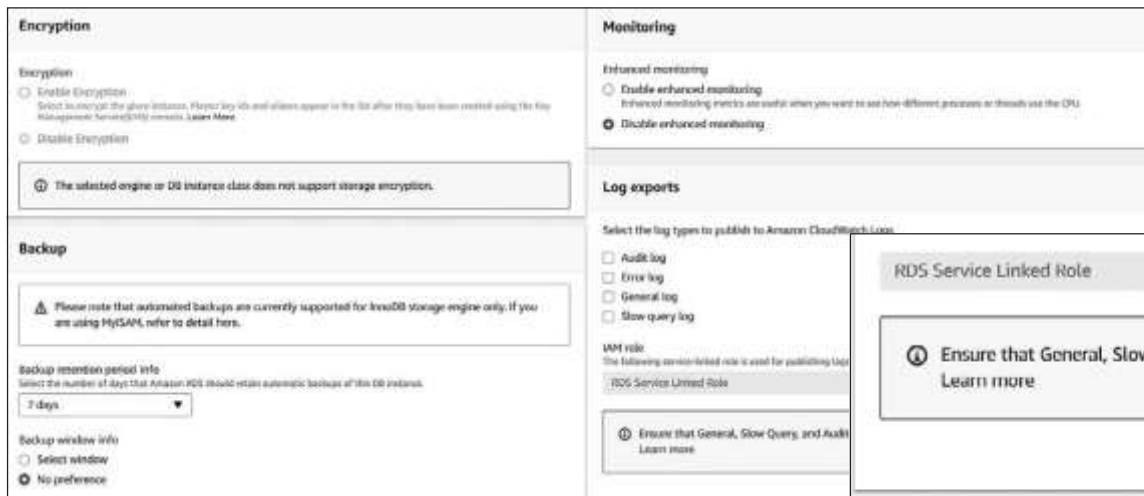


Рис. 5.25. Панель настройки шифрования, бэкапа, расширенного мониторинга и экспорта логов

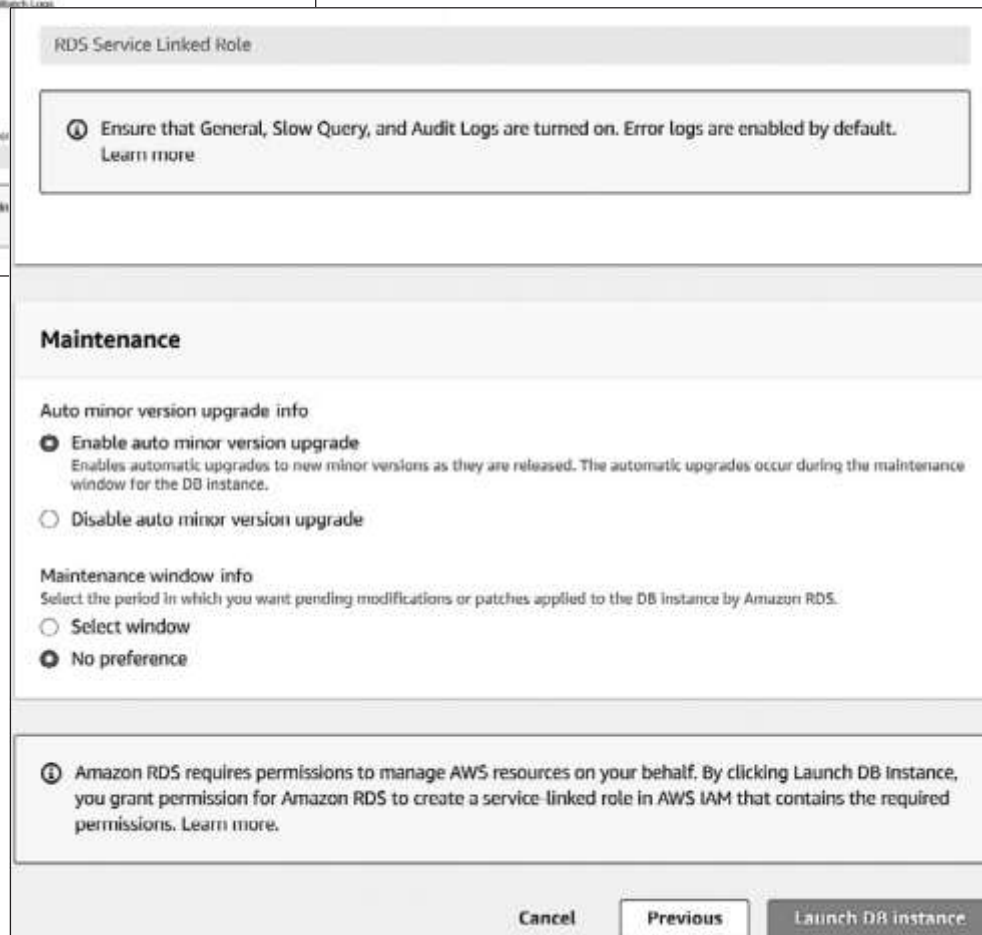


Рис. 5.26. Панель настройки автоматической установки патчей и обновлений

# AWS RDS — Amazon Web Services

## Relational Database Service

Упомянутый параметр очень важен для того, чтобы обеспечить обновляемость операционной машины экземпляра и движка базы данных. Не забывайте, в AWS экземпляры БД — это не экземпляры бессерверного сервиса, как в случае Azure SQL, а скорее обертка над сервисами IaaS.

Но AWS RDS — это PaaS-сервис, предоставляющий ряд встроенных механизмов администрирования, например механизм резервного копирования и управления сроком жизни резервной копии. Помимо периодически запускаемого резервного копирования, можно создать мгновенный снимок состояния (snapshot) и использовать его для сохранения и восстановления состояния памяти.

Помимо PaaS-возможностей, AWS RDS содержит и опции, наследованные от IaaS. Так, экземпляры могут быть остановлены, запущены и перезапущены. При остановке перестает начисляться плата за вычислительные ресурсы экземпляра, чего не скажешь о плате за виртуальные диски и операции ввода-вывода данных (IOPS) и за хранение бэкапов.

Кроме того, операционная система и движок экземпляра требуют обновления и пользователь должен сделать это вручную через консоль. Тип экземпляра может быть изменен вручную или через API таким же образом, как и в случае EC2. Вдобавок для экземпляра MS SQL можно подключить сервис Microsoft Active Directory, чтобы хранить учетные данные пользователя.

# AWS RDS — Amazon Web Services

## Relational Database Service

Мониторинг применения ресурсов экземпляров производится с помощью стандартного сервиса CloudWatch. В настоящее время AWS RDS не предоставляет общего механизма анализа влияния запросов на использование ресурсов, для каждого движка AWS рекомендует задействовать встроенные в этот движок механизмы анализа производительности.

Экземпляры движка Aurora создаются как кластеры — отказоустойчивые комбинации из нескольких экземпляров. Они состоят из главного экземпляра (primary instance) и реплик, поддерживающих только операции чтения (readonly replica) (рис. 5.27).

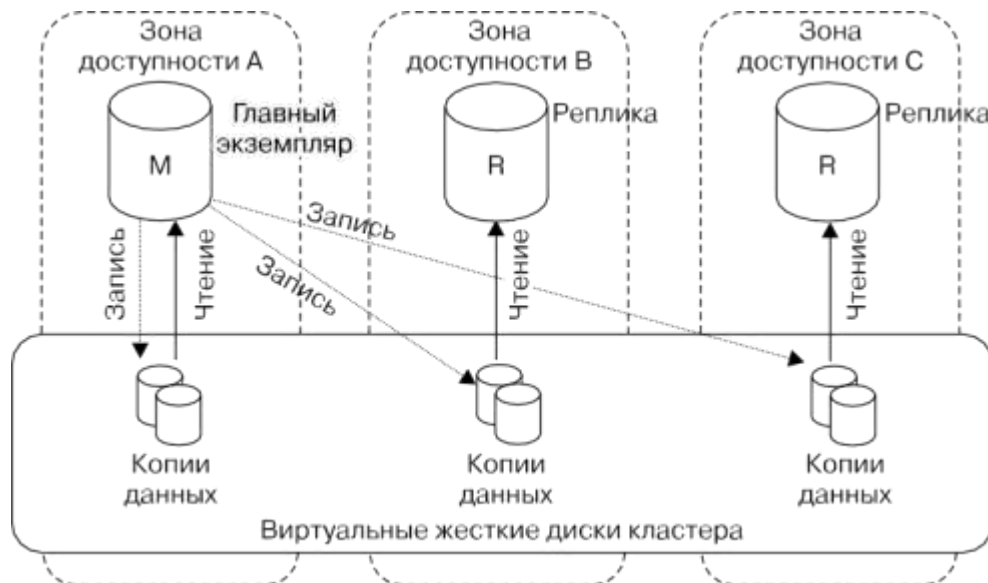
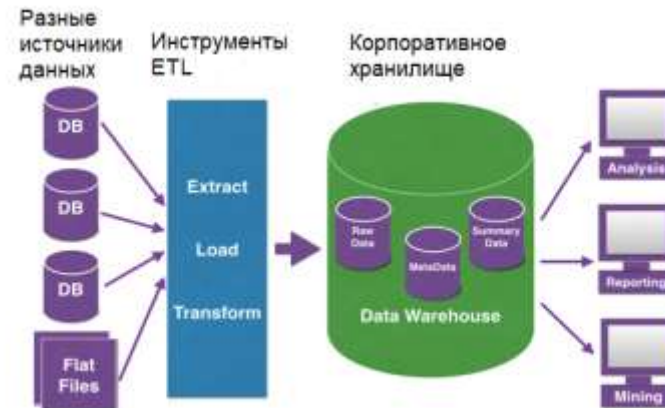


Рис. 5.27. Архитектура сервиса Aurora

# Модуль 1. Основы построения и работы с системами аналитики больших данных



## КРАТКОЕ СОДЕРЖАНИЕ:

### 1. Архитектура систем аналитики больших данных.

1.1. Облачные технологии. Модели развертывания. Способы создания ресурсов в облаке.

1.2. Безопасность облачных ресурсов

1.3. Большие данные и источники данных. Форматы. Преобразование данных из различных форматов. Режимы обработки больших данных.

### 2. Хранение больших данных в облаке.

2.1. Хранилища общего назначения. Форматы хранения данных. Облачное хранилище Microsoft Azure Storage. Облачное хранилище AWS.

2.2. Реляционные базы данных. Azure SQL. AWS RDS.

2.3. Нереляционные базы данных. Сервисы нереляционных баз данных от Azure и AWS.

2.4. Корпоративные хранилища данных (DWH). Azure SQL DWH. AWS RedShift.

2.5. Хранилища данных типа Data Lake. Azure Data Lake Store. AWS Data Lake Solutions.

# ЛЕКЦИЯ 10:

Хранение больших данных в облаке  
Нереляционные базы данных.  
Сервисы нереляционных баз  
данных от Azure и AWS.





# Общий обзор баз данных NoSQL

Зачастую большие данные гораздо удобнее хранить в виде, отличном от реляционного. Очень многие направления современных информационных систем, оперирующих большими данными, не нуждаются в обеспечении строгой согласованности и транзакционности, но требуют очень быстрого доступа к произвольному подмножеству хранимой информации и возможности хранения гигантских объемов данных — сотен терабайт.

- IoT — поток телеметрии с устройств часто представляет собой поток сообщений, каждое из которых включает произвольный набор полей полезной нагрузки. Часто очень удобно хранить подобные данные в виде таблицы, допускающей отсутствие схемы (то есть каждая строка может содержать свои наборы колонок, общими являются поля ключей и временной метки). Данных может быть очень много, и они должны быть доступны для извлечения запросом, выполняющимся за минимальное время. Нереляционные базы как раз предоставляют тип хранилища, обеспечивающий работу с таким видом информации, — *табличные базы данных*, называемые еще *базами данных типа «ключ — значение»*.
- Социальные сети или блог-платформы. В этом случае можно столкнуться с трудностями двух видов. Во-первых, пользователи социальных сетей связаны друг с другом связями типа «друзья», «последователи», «состоящие в одной группе», «подписчики». Во-вторых, пользователи генерируют контент, состоящий из сообщений, постов, комментариев, оценок и др. В силу особенности предметной области для хранения данных пользователей наиболее удобны так называемые *графовые БД*, то есть базы, в которых информация представлена в виде графа. Теперь вернемся к генерируемому пользователем контенту. Помимо связей с многими пользователями (в качестве примера можно привести пост, содержащий много комментариев разных пользователей и оценок этих комментариев), этот контент может включать медиафайлы: фото, видео, музыку. И совершенно очевидно, что пользователи социальных сетей ожидают максимального быстродействия: то есть буквально пары секунд на перезагрузку страницы, чтобы увидеть свой комментарий и сотни других вне зависимости от наличия или отсутствия в них медиафайлов. С таким вызовом прекрасно справляются *документоориентированные БД*. Информация в них хранится в виде JSON-документов (в данном случае это логическое понятие единицы хранения информации), включающих в том числе метаданные других документов, медиафайлов.
- Системы рекомендации контента. В данном случае задача состоит в том, чтобы в существующую систему (социальную сеть, блог, интернет-магазин, сайт с фото- или видеоконтентом) встроить подсистему, предоставляющую пользователю ссылку на потенциально интересный или полезный товар (статью, контент). Для этого необходимо хранить информацию об истории посещения пользователем страниц, а также об истории посещения страниц других пользователей и определять сходные паттерны в поведении. Существует достаточно много алгоритмов такой рекомендации, но все они строятся на основе информации о просмотренных пользователями страницах и выявления закономерностей в этом процессе. Данная информация может храниться как в БД «ключ — значение», так и в графовой.

# Основные типы нереляционных баз данных и концепции, лежащие в их основе.

В настоящее время в облачных средах Microsoft Azure и Amazon Web Services реализованы четыре типа нереляционных баз данных: «ключ — значение», графовые, документоориентированные и семейства столбцов.

Итак, первый и наиболее распространенный тип нереляционной БД — *база данных типа «ключ — значение»*. Строго говоря, к этому типу можно отнести не только табличные базы, но и те, в которых информация хранится и доступна по ключу, например Redis, Memcache. Но подобные базы чаще всего служат для кэширования запросов к основной, но более медленной БД и содержат только временную информацию, действительную в течение ограниченного промежутка времени, потеря которой совершенно не критична для системы. *Мы не будем рассматривать эти БД, поскольку они не используются для непосредственного хранения больших данных.*

Итак, в средах **Azure** и **AWS** базы данных типа «ключ — значение» представлены как отдельными **PaaS-сервисами** (например, Azure Table Storage, AWS DynamoDB), так и в виде **PaaS/IaaS** (AWS EMR HBase, Azure HDInsight HBase). Рассмотрим базовые сущности, из которых состоят БД типа «ключ — значение», реализованные в модели PaaS.

# Базовые сущности, из которых состоят БД типа «ключ — значение», реализованные в модели PaaS.

Верхний логический элемент хранилища «ключ — значение» — *аккаунт (account)* (для Azure это может быть Azure Storage Account).

**Аккаунт** состоит из *таблиц (table)*, которые представляют собой контейнеры для хранения единиц информации. **Таблица** состоит из *сущностей (entity)* (по сути, являющихся строками), которые имеют такое название, чтобы подчеркнуть: в отличие от строк таблицы реляционной БД сущности могут состоять из произвольного числа *свойств (properties)*.

Но ряд свойств обязательно содержится во всех сущностях: *ключ раздела (partition key)*, *ключ строки (row key)* и *временная метка (timestamp)*.

Два ключа необходимы в силу специфики хранения данных этих таблиц в облачных средах. Дело в том, что таблицы хранятся на SSD-дисках (напрямую или в массиве RAID) — в разделах — и те строки таблицы, которые имеют одинаковый ключ раздела, будут храниться физически вместе на общих дисках или в RAID. Напротив, ключ строки — уникальный идентификатор сущности в пределах раздела, то есть комбинация «ключ раздела — ключ строки» уникальна. Временная метка показывает время последней модификации сущности.

**Индексация сущностей** происходит автоматически — очевидно индексируются ключевые поля. Это обстоятельство требует тщательного продумывания структуры такой таблицы для целей хранения/добавления/обновления/выборки данных. Дело в том, что базы типа «ключ — значение» допускают в ряде случаев применение механизма транзакций для обновления группы строк, но этот механизм используется для строк с общим ключом раздела.

# Тип нереляционной базы данных — графовая БД.

Он идеально подходит для хранения информации, моделирующей связи между сущностями реального мира. Как уже отмечалось, это могут быть пользователи в социальной сети, связанные отношениями с другими пользователями. В качестве модели такой предметной области выступает *граф* — математический объект, состоящий из *вершин и связей* (рис. 6.1).

Вершины — точки соединения ребер, причем важен факт соединения вершин ребрами, а не сама форма графа (это называется изоморфизм).

*Каждая вершина может иметь идентификатор (имя) и набор свойств, относящихся к сущности, которую представляет эта вершина. Каждая связь (ребро графа) может иметь свойство или вес. Свойством может описываться тип связи («друг», «последователь», «родственник» и пр.), а весом — некая величина, характеризующая взаимодействие между вершинами.*

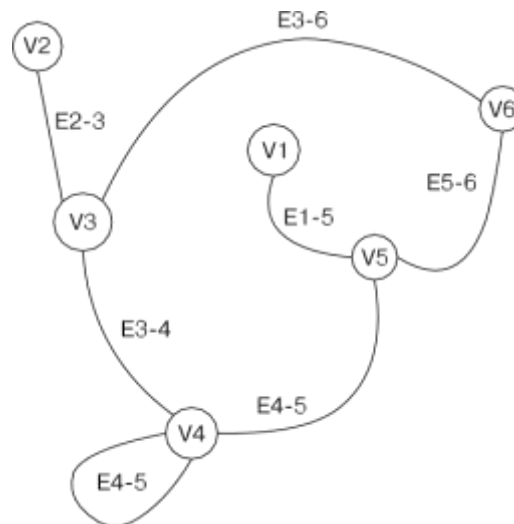


Рис. 6.1. Образец графа, состоящего из вершин V и ребер E

# Тип нереляционной базы данных — графовая БД.

Графовые БД предоставляют специальные языки запросов, удобные для выборки информации, относящейся к графу. Более подробно этот вопрос будет рассмотрен в части III. В облачных средах эти базы представлены у [Azure \(Azure CosmosDB Graph API\)](#) и с недавнего времени у [Amazon \(AWS Neptune\)](#). Графовые БД очень удобны для хранения информации в различного рода социальных сетях.

# Тип нереляционных баз данных — документоориентированная БД.

Следующий популярный тип **нереляционных баз данных** — **документоориентированная БД**. Рассмотрим, из каких элементов состоит такая база (рис. 6.2).

Прежде всего, это **аккаунт**, который содержит одну или несколько **баз данных**. База, в свою очередь, состоит из **пользователей**, которые характеризуются набором привилегий и прав доступа, а также из **коллекций**. В последние включены хранимые процедуры, пользовательские функции, триггеры, документы и прикрепленные файлы.

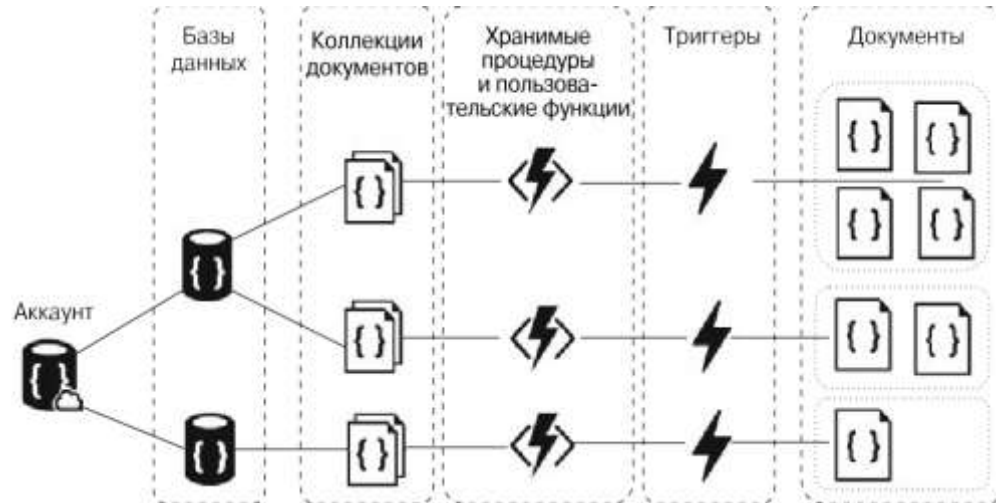


Рис. 6.2. Состав документоориентированной базы данных на примере DocumentDB

# Тип нереляционных баз данных — документоориентированная БД.

Рассмотрим элементы коллекции.

Самый главный элемент — это **документ, хранящий информацию в формате JSON**. Набор полей в документе может быть произвольным, но, как и в случае с базами типа «ключ — значение», в этом документе есть ряд полей обязательных и добавляемых по умолчанию.

Представление информации в формате JSON очень удобно, поскольку ее структура (набор полей, типы данных и др.) может быть произвольной. Кроме того, информация, хранящаяся в этом формате, очень легко интегрируется в одностраничные веб-приложения (single page application, SPA).

В частности, для построения систем с единым языком на разных уровнях очень удобны технологии ReactJS, AngularJS (для клиентской части), NodeJS (для серверной) и документоориентированная база данных типа MongoDB.

Существует целый стек технологий, «насквозь» состоящий из JSON/JavaScript. Это MEAN — MongoDB, ExpressJS, AngularJS, NodeJS.

Кроме собственно хранения информации в формате JSON, документоориентированная база данных предоставляет средство выполнения запросов.

В качестве языка запросов может выступать JavaScript, нотация JSON, а в ряде случаев и SQL (в частности, поэтому в облачной среде Azure база данных DocumentDB в составе Azure CosmosDB называется SQL API — не стоит путать ее с сервисом AzureSQL!).

Нереляционные базы данных типа **семейства столбцов** (column family) обеспечивают хранение информации в виде разреженной матрицы, у которой строки и столбцы используются как ключи.

# Сервисы нереляционных баз данных от Azure

В настоящее время Azure предоставляет два сервиса нереляционных хранилищ данных:

- традиционный **Azure Table Storage**
- **Azure CosmosDB**.

Первый — хранилище типа «ключ — значение».

Второй — набор глобально распределенных нереляционных хранилищ, объединенных общей концепцией управления и создания, к которым в настоящее время относятся базы данных DocumentDB и MongoDB (документоориентированные), Gremlin (графовая база данных), Table (таблица типа «ключ — значение») и Cassandra (база данных, относящаяся к типу семейства столбцов).

Начнем с рассмотрения **Azure Table Storage**. Этот сервис является встроенным в Azure Storage Account и создается на общей панели, содержащей остальные сервисы (Blob, File и Queue). Общая панель доступа к **Azure Table Storage** такая же, как и для других сервисов, включенных в Azure Storage. Она очень проста и содержит минимальные опции для конфигурации (рис. 6.3).



# Azure Table Storage

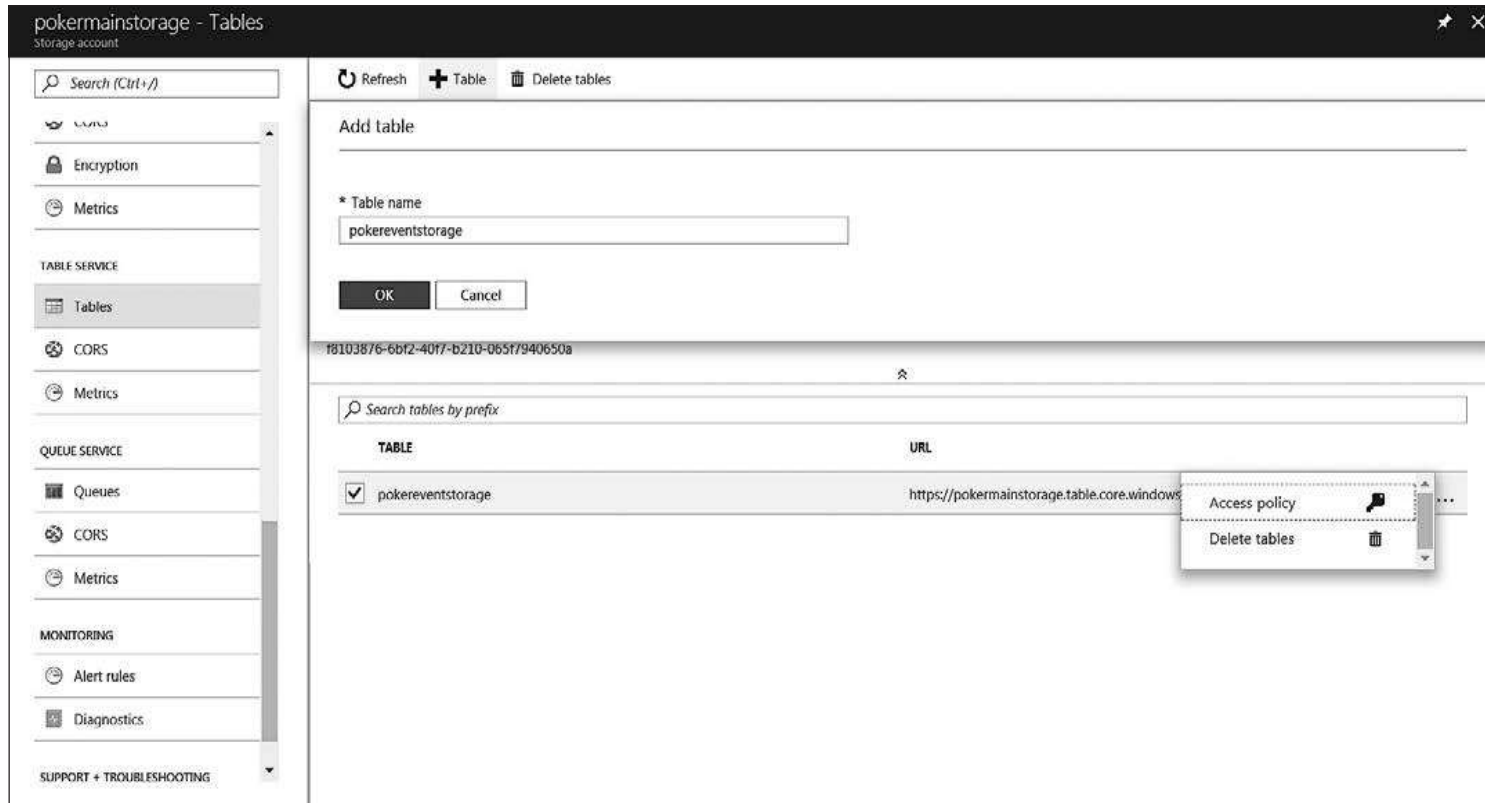


Рис. 6.3. Внешний вид общей панели хранилища Azure Table Storage

Это хранилище поддерживает протокол OData для выборки данных на основе протокола HTTP, так что таблица имеет конфигурации для настройки CORS и политики доступа.

Само по себе хранилище **Azure Table Storage** предназначено для хранения больших объемов (в настоящее время это 500 Тбайт на один Storage Account) структурированных нереляционных данных. Под структурированностью здесь понимается тот факт, что все сущности («строки») представляют собой набор пар «ключ — значение» (причем не обязательно, чтобы он был одинаковым во всех строках).

# Azure Table Storage

По сути, это огромные таблицы, которые нецелесообразно размещать в РБД.

В качестве примера можно привести сырые потоковые данные, помещенные в таблицу сервисом *Stream Analytics Job*, которые в итоге должны быть агрегированы сервисом ETL (например, сервисом **Azure DataFactory** в комбинации с **Azure HDInsight Spark**) и помещены в реляционное хранилище данных (Azure SQL DataWarehouse) или просто в обычную РБД.

При этом желательно сохранить все сырые данные, чтобы в последующем можно было выполнить их интерактивный или интеллектуальный анализ.

**Azure Table Storage** позволяет выполнить такую операцию ввиду того, что способен легко интегрироваться с сервисами копирования, трансформации и потокового анализа данных. Кроме того, поддержка протокола OData для прямого доступа и наличие SDK позволяет получить прямой доступ к табличному хранилищу извне. При этом выполнение высокопроизводительных запросов возможно благодаря встроенному механизму кластерных индексов.

# Главные компоненты Azure Table Storage

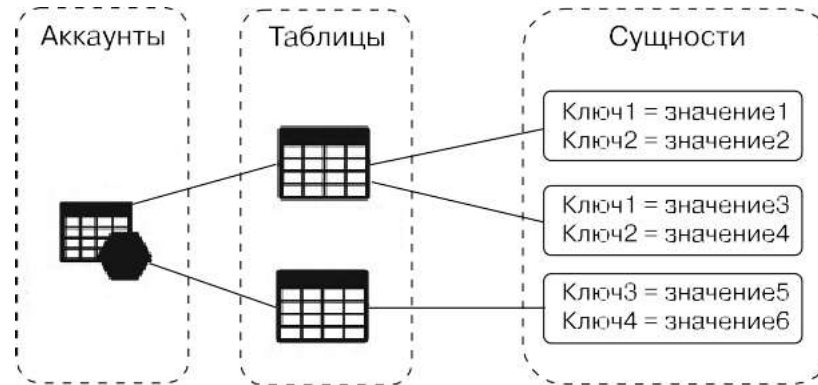


Рис. 6.4. Базовые компоненты сервиса Azure Table Storage

Итак, Storage Table Account включает в себя набор *таблиц*, каждая из которых вмещает набор *сущностей*, то есть строк. Каждая сущность состоит из набора *свойств* и имеет суммарный размер не более 1 Мбайт. В свою очередь, свойство — это пара «имя — значение».

Каждая сущность может иметь до 256 свойств, три из которых обязательны: ключ раздела (partition key), ключ строки (row key) и временная метка (timestamp). Временная метка соответствует времени последней модификации этой сущности.

Ключ строки должен быть уникальным в пределах раздела, а комбинация «ключ раздела — ключ строки» — уникальной глобально. Для набора, состоящего из сущностей с одинаковым значением ключа раздела, значения могут быть очень быстро выбраны с помощью запроса.

Операции вставки/удаления/обновления в данном случае тоже выполняются быстрее. Это возможно потому, что на оба ключа создаются кластерные индексы. Никакие другие индексы создать нельзя.

# Azure Table Storage

Типы данных, которые поддерживаются Table Storage, полностью соответствуют типам данных, доступных в протоколе OData (табл. 6.1).

Тип по умолчанию — строковый. Для хранения более сложные типы данных должны быть сериализованы в XML или JSON и помещены в строковый формат. Второй вариант — сериализация в двоичный формат и помещение его в битовый массив.

Тип данных OData	CLR-тип, доступный в коде	Примечание
Edm.Binary	byte[ ]	Массив байтов размером до 64 Кбайт
Edm.Boolean	Bool	Булево значение
Edm.DateTime	DateTime	64-разрядное значение, представляющее собой временную метку UTC
Edm.Double	Double	64-разрядное число с плавающей точкой
Edm.Guid	Guid	128-разрядный глобальный идентификатор
Edm.Int32	int / Int32	32-разрядное целое
Edm.Int64	long / Int64	64-разрядное целое
Edm.String	String	Строковая величина с кодировкой UTF-16. Максимальный размер строки — 64 Кбайт

Таблица. 6.1. Типы данных стандарта OData

# Сервис Azure CosmosDB

Рассмотрим теперь базы данных от сервиса Azure CosmosDB. Он является относительно новым и предоставляет единую программную модель для доступа к нереляционным базам разных типов:

- DocumentDB, она же SQL API, — документоориентированная база данных с возможностью выполнения запросов с помощью как SQL, так и JavaScript;
- MongoDB — облачный сервис хорошо известной базы с таким же названием;
- Graph API — сервис графовой базы данных, называемой еще Gremlin;
- Table API — дальнейшее развитие базы данных типа «ключ — значение», почти полный аналог Azure Table Storage;
- Cassandra — сервис, являющийся адаптацией Apache Cassandra.

Указанные выше базы собраны в единый сервис, предоставляющий всем им ряд общих уникальных черт и свойств. Прежде всего, это *глобальная доступность* — все перечисленные БД можно реплицировать во *все* регионы, то есть создать копии баз во всех регионах одновременно. При этом в зависимости от географического местоположения клиента его запрос будет направлен к ближайшему дата-центру. Уровень согласованности данных можно настроить вплоть до уровня сильной согласованности (strong consistency).

# Сервис Azure CosmosDB

Познакомимся подробнее с сервисом **Azure CosmosDB** и его возможностями, а затем перейдем к частным типам БД. Прежде всего необходимо создать **аккаунт CosmosDB**, в рамках которого мы будем создавать базы, а позже и сущности хранения в них. Для создания аккаунта нужно в левом верхнем углу портала нажать на ссылку + New и в появившемся окне поиска выбрать CosmosDB (рис. 6.5). После нажатия кнопки Create (Создать) появится следующая форма (рис. 6.6).



Рис. 6.5. Начальная страница сервиса Azure CosmosDB



Рис. 6.6. Форма создания аккаунта CosmosDB

# Сервис Azure CosmosDB

Прежде всего здесь необходимо указать идентификатор аккаунта (ID), выбрать его тип (API), группу ресурсов (Resource Group), местоположение и включить опцию географического дублирования (Enable geo-redundancy). Таким образом, при создании нужно выбрать SQL API и нажать кнопку Create (Создать).

После создания аккаунта CosmosDB типа SQL API будет доступна стартовая страница, показанная на рис. 6.7. Добавить коллекцию можно прямо на стартовой странице, но мы пойдём на вкладку Overview (рис. 6.8).

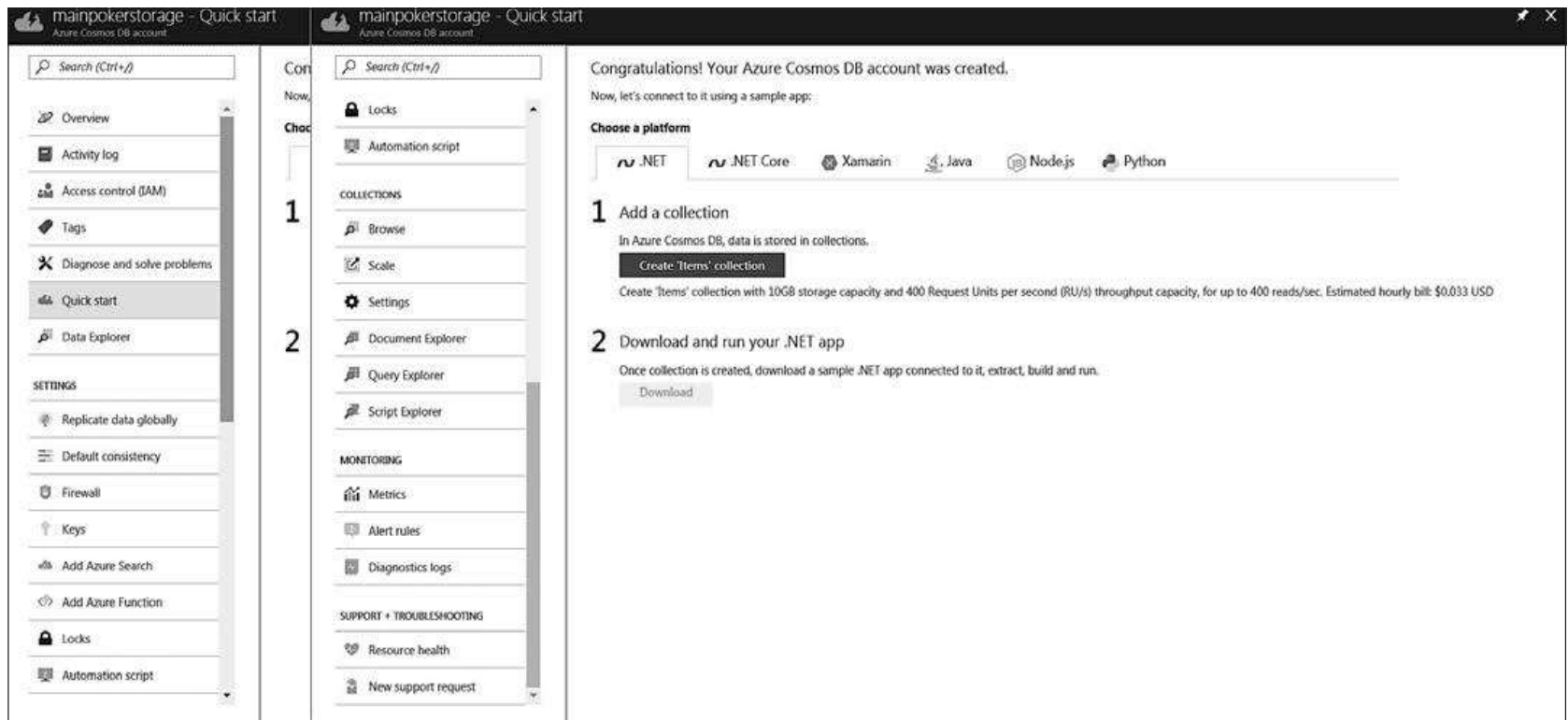


Рис. 6.7. Стартовая страница сервиса CosmosDB

# Сервис Azure CosmosDB

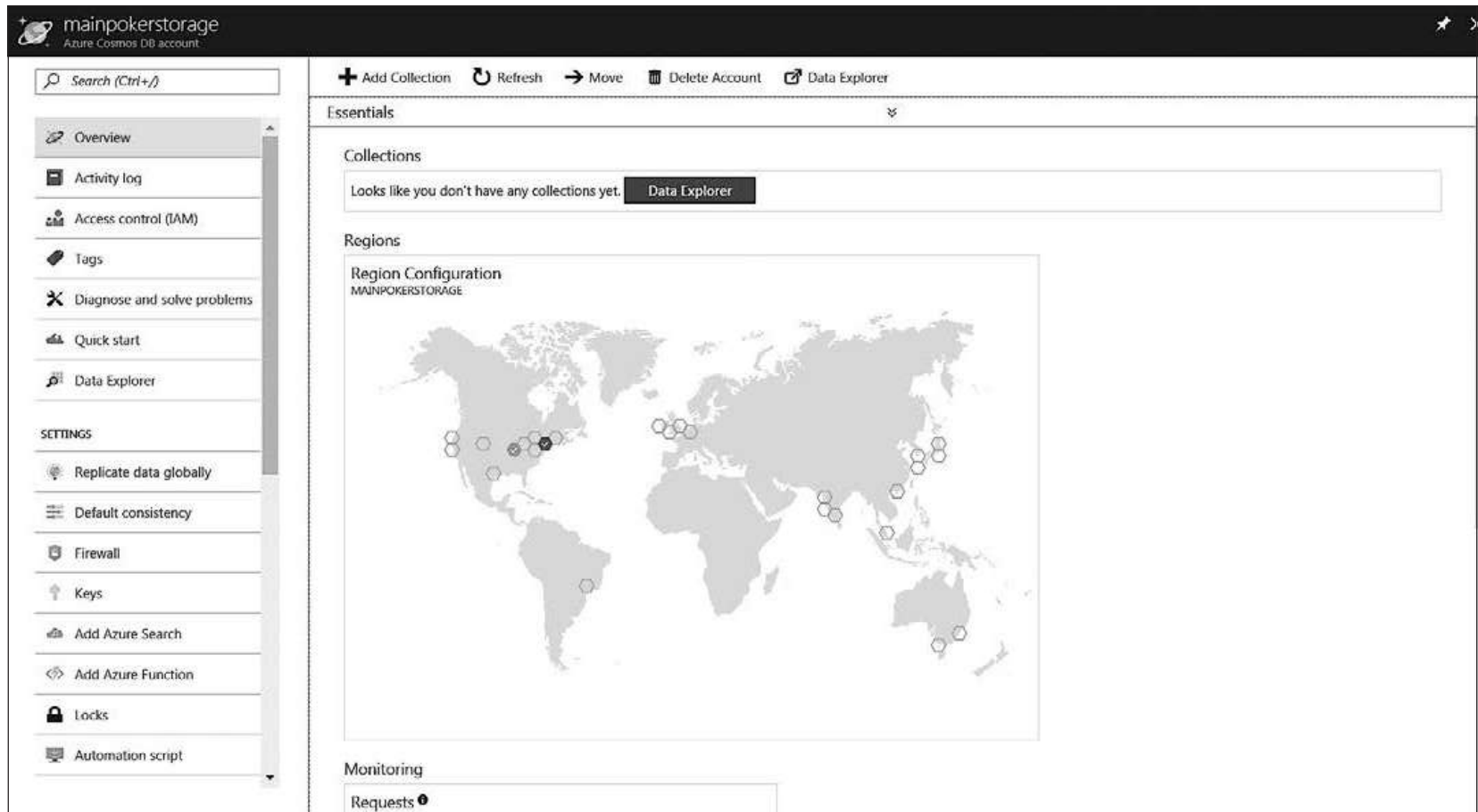


Рис. 6.8. Страница обзора аккаунта

Перейдя на вкладку Overview, можно видеть на карте доступные для репликации регионы (прозрачные шестиугольники) и регионы, использованные в настоящий момент (закрашенные шестиугольники).

Чтобы добавить коллекцию, необходимо нажать ссылку + Add Collection в верхней части экрана, после чего откроется форма, показанная на рис. 6.9.



# Сервис Azure CosmosDB

Чтобы добавить коллекцию, необходимо нажать ссылку + Add Collection в верх-ней части экрана, после чего откроется форма, показанная на рис. 6.9. Поскольку у нас еще нет баз данных, нужно создать первую базу данных (Database id) — PockerGameData и в ее рамках создать коллекцию (Collection id) — GameEvents.

Далее следует выбрать размер хранилища (Storage capacity). Он может быть фиксированным (опция Fixed (10 GB)) и неограниченным (опция Unlimited), то есть автоматически масштабируемым. Кроме того, нужно выбрать производительность (выражена в терминах единиц чтения в секунду) из заданного диапазона. Минимальное значение равно 400. Далее можно добавить в коллекцию документов уникальный ключ (ссылка + Add unique key) и нажать OK. Созданная база данных и коллекция показана на рис. 6.10.

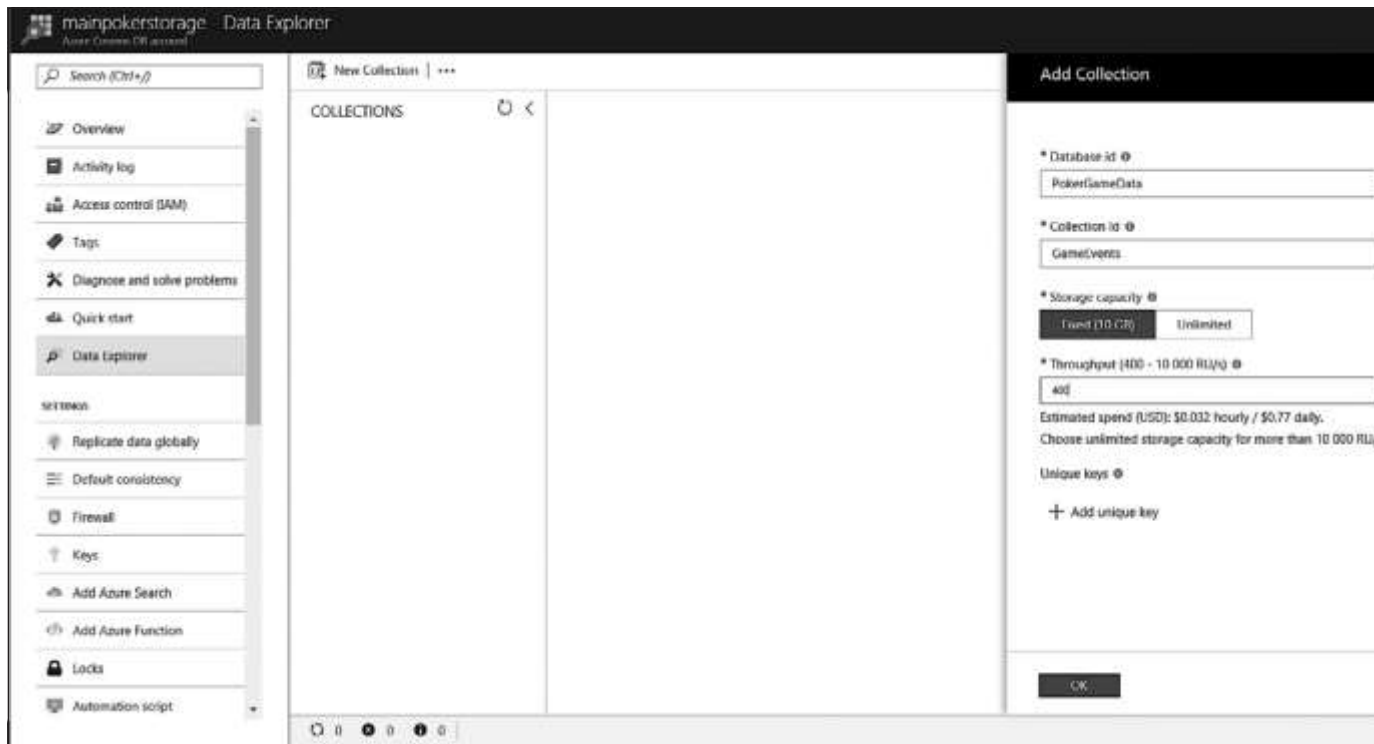


Рис. 6.9. Форма создания базы данных DocumentDB и добавление новой коллекции документов

# Сервис Azure CosmosDB

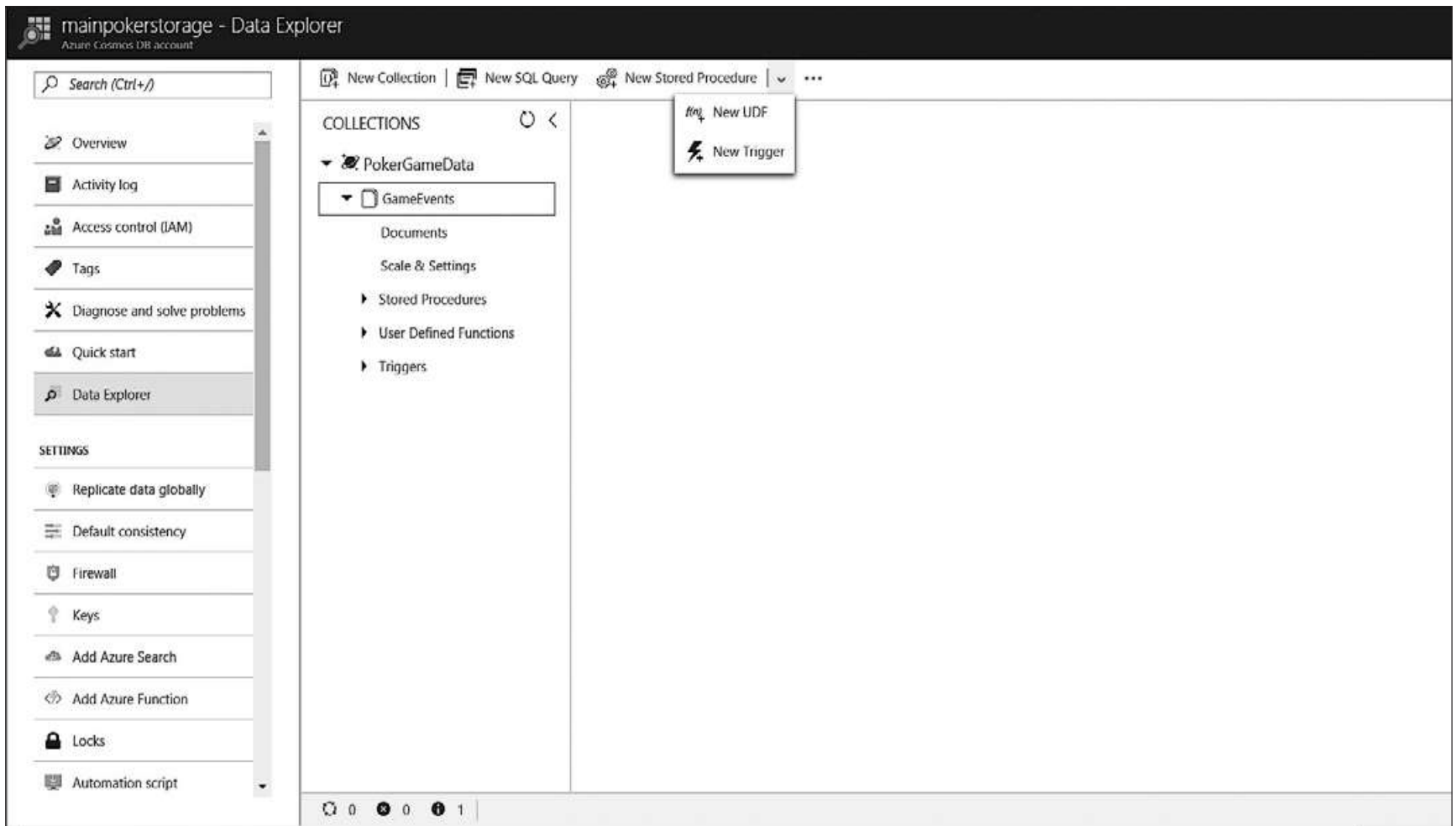


Рис. 6.10. Внешний вид вкладки Data Explorer с созданной базой данных и коллекцией

# Сервис Azure CosmosDB

Аккаунты CosmosDB точно так же состоят из одной или нескольких баз данных, которые, в свою очередь, состоят из одной или нескольких коллекций. Эти два элемента сервиса CosmosDB SQL API мы и создали на предыдущем шаге. Каждая коллекция, в свою очередь, состоит из хранимых процедур (stored procedure), определяемых пользователем функций (user defined functions), триггеров (triggers) и собственно документов (documents).

Разберем подробнее, что такое документ в терминах DocumentDB, на примере его создания. Для этого на вкладке Data Explorer перейдем к нашим созданным базе данных и коллекции и нажмем ссылку Documents. На появившейся вкладке нажмем на ссылку **New Document**, в результате изображение на экране будет выглядеть следующим образом (рис. 6.11).

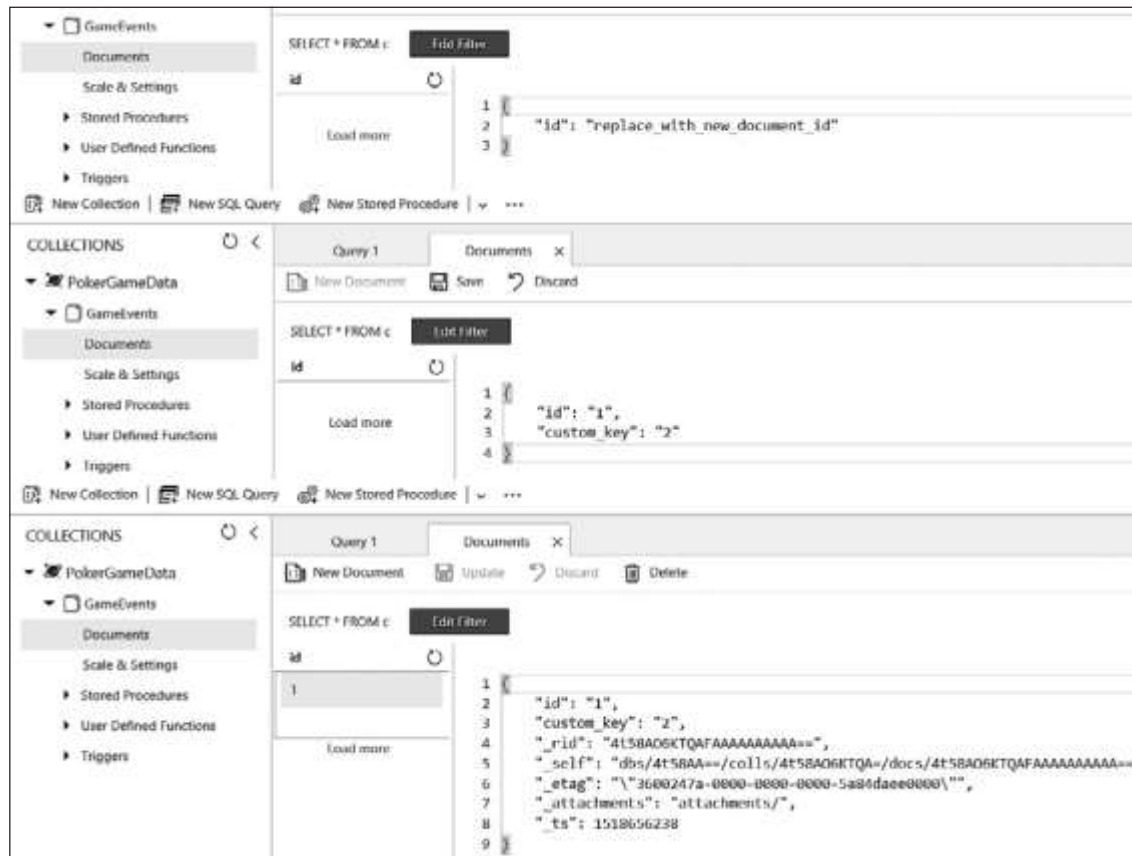


Рис. 6.11. Последовательность создания документа

# Сервис Azure CosmosDB

Теперь приведем один пример применения SQL-подобного запроса для выборки данных. Сначала нужно добавить несколько документов с различными значениями пользовательского поля (рис. 6.12).

Стоит отметить, что, помимо JSON-документов, документоориентированная база данных позволяет сохранять приложения (attachments) в виде двоичных файлов размером до 2 Гбайт, которые могут быть медиафайлами, архивами и др.

The screenshot displays the Azure Cosmos DB Data Explorer interface in two states. The top state shows the 'Documents' collection selected, with a table view of document IDs (1, 2, 3) and a corresponding JSON document for ID 3. The bottom state shows the same interface with a SQL query executed, resulting in a single JSON document returned.

**Top Panel (Initial State):**

- Search: Search (Ctrl+/)
- Navigation: Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Quick start, Data Explorer
- COLLECTIONS: PokerGameData, GameEvents, Documents (selected), Scale & Settings, Stored Procedures, User Defined Functions, Triggers
- Query 1: Documents
- Actions: New Document, Update, Discard, Delete
- Query: `SELECT * FROM c`
- Table View:

id
1
2
3
- JSON View:

```
{  "id": "3",  "custom_key": "8"}
```

**Bottom Panel (Query Executed):**

- Search: Search (Ctrl+/)
- Navigation: Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Quick start, Data Explorer
- SETTINGS: Replicate data globally, Default consistency, Firewall
- COLLECTIONS: PokerGameData, GameEvents, Documents (selected), Scale & Settings, Stored Procedures, User Defined Functions, Triggers
- Query 1: Documents
- Execute Query
- Query: `1 SELECT * FROM c WHERE c.custom_key = "8"`
- Results: 1 - 1 | Request Charge: 2.94 RUs | Next →
- JSON View:

```
{  "id": "3",  "custom_key": "8",  "_rid": "4t58A06KTQAIAAAAAAAAAA==",  "_self": "dbs/4t58AA==/colls/4t58A06KTQA=/docs/4t58A06KTQAIAAAAAAAAAA==/",  "_etag": "\"36000af7-0000-0000-0000-5a84dee90000\"",  "_attachments": "attachments/",  "_ts": 1518657257}
```

Рис. 6.12. Пример выполнения SQL-запроса к JSON-документам

# Сервис Azure CosmosDB

*Триггеры* представляют собой программные конструкции, написанные на языке JavaScript, вызываемые при выполнении операций на документах. Триггер может выполняться непосредственно перед созданием документа (pre-trigger) и после (post-trigger). Очень интересным и важным примером использования триггеров является интеграция с Azure Function — сервисами бессерверного исполнения кода, который может выполняться в ответ на внешний сигнал. Для CosmosDB возможны следующие сценарии запуска (в настоящее время реализованы только для SQL API и Graph API).

1. Применение *потока событий изменения (change feed)* — перехват Azure Function событий изменений коллекции (добавление/удаление/изменения). При этом поток событий активизирует пользовательский триггер, который, в свою очередь, запускает Azure Function. Данный подход полностью соответствует концепции EventDriven (рис. 6.13).

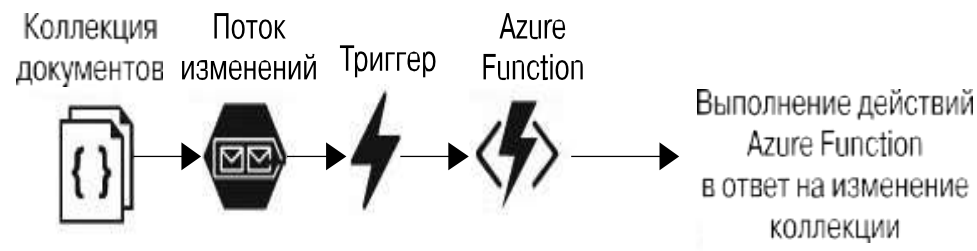


Рис. 6.13. Использование потока событий изменений для вызова Azure Function

# Сервис Azure CosmosDB

Для CosmosDB возможны следующие сценарии запуска (в настоящее время реализованы только для SQL API и Graph API).

2. **Применение привязки входа (input binding)** — при этом Azure Function читает данные из базы при запуске сторонним триггером (рис. 6.14).

3. **Применение привязки выхода (output binding)** — в этом случае Azure Function записывает данные в базу в ответ на запуск триггером (рис. 6.15).

Эти паттерны полезны в реальных сценариях и позволяют строить сложные приложения, ориентированные как на события (Event Driven), так и на данные (Data Driven).

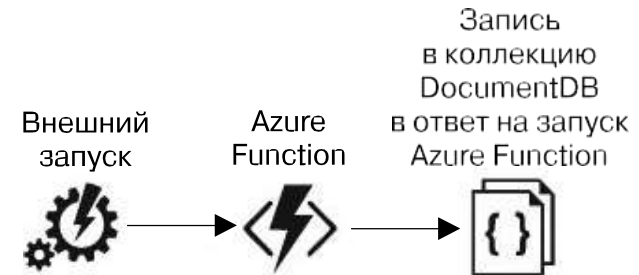
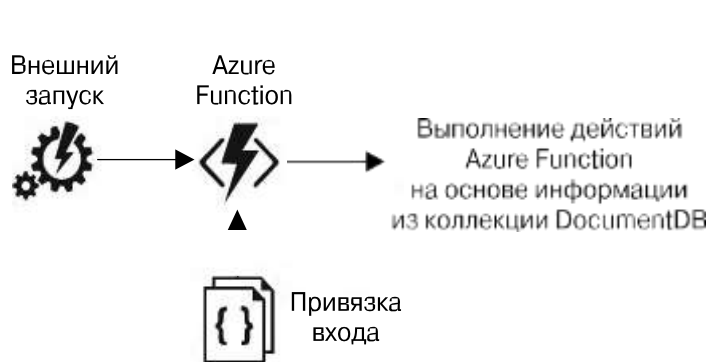


Рис. 6.15. Использование Azure Function с привязкой выхода

Рис. 6.14. Использование Azure Function с привязкой входа

# Важные опции сервиса CosmosDB.

Прежде всего, это настройки глобальной репликации (рис. 6.16) и конфигурирования опции отказоустойчивости (рис. 6.17). Для настройки опций отказоустойчивости необходимо указать резервные головные регионы. Этим регионам назначается приоритет. Данный приоритет будет задавать порядок, который станет использоваться при переключении отказавшей головной реплики на резервную из списка.

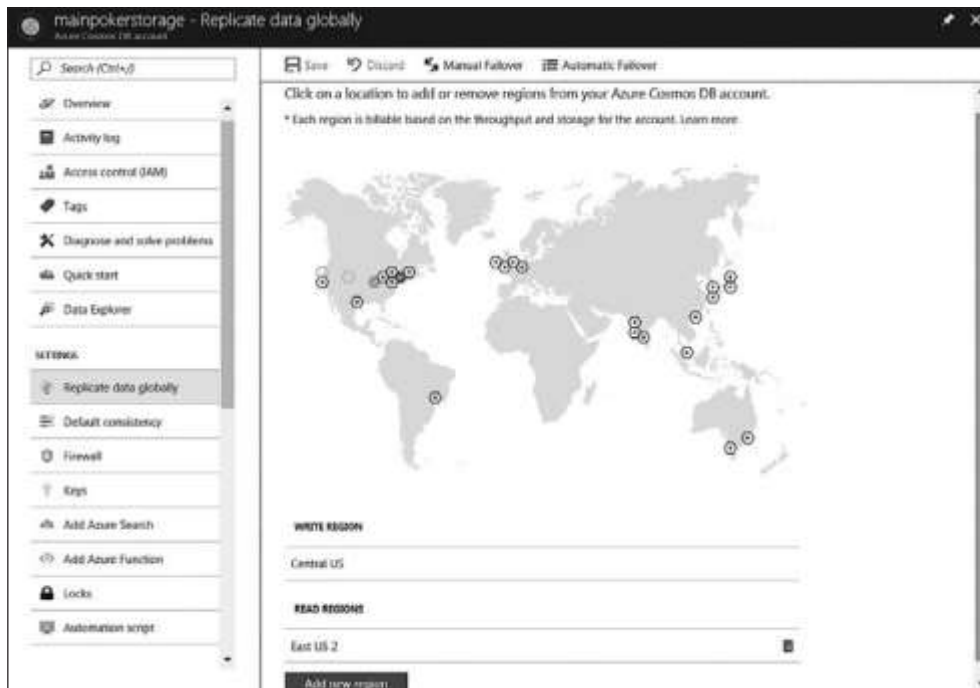


Рис. 6.16. Настройка глобальной репликации

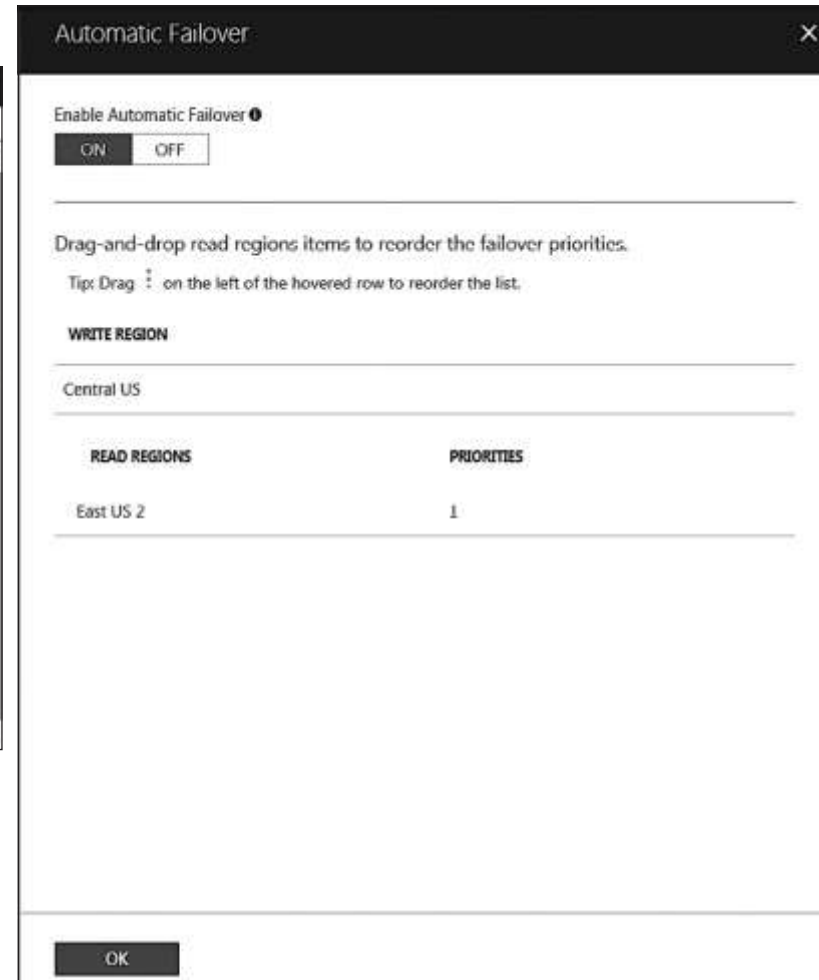


Рис. 6.17. Страница конфигурирования опций отказоустойчивости

# Важные опции сервиса CosmosDB.

Следующий важнейший параметр — настройка уровня согласованности (вкладка Default consistency) (рис. 6.18).

Выбор уровня согласованности зависит от требований конкретной информационной системы. Более высокий (сильный) уровень влечет более высокую цену и увеличение времени записи. Пользовательский интерфейс графовой базы данных несколько отличается от интерфейса CosmosDB SQL API, в том числе вкладкой DataExplorer. Первое отличие состоит в том, что вместо коллекций создаются графы. А элементарные сущности графа — вершины. Ссылка New Vertex создает новую вершину, к которой можно добавлять метку (label) и ряд пользовательских свойств, представляющих набор «ключ — значение».

На рис. 6.19 это свойство Rate. Для каждой вершины есть настраиваемые свойства source и target, определяющие ориентированные ребра графа, которым также можно добавить метку.



Рис. 6.18. Настройка уровня согласованности

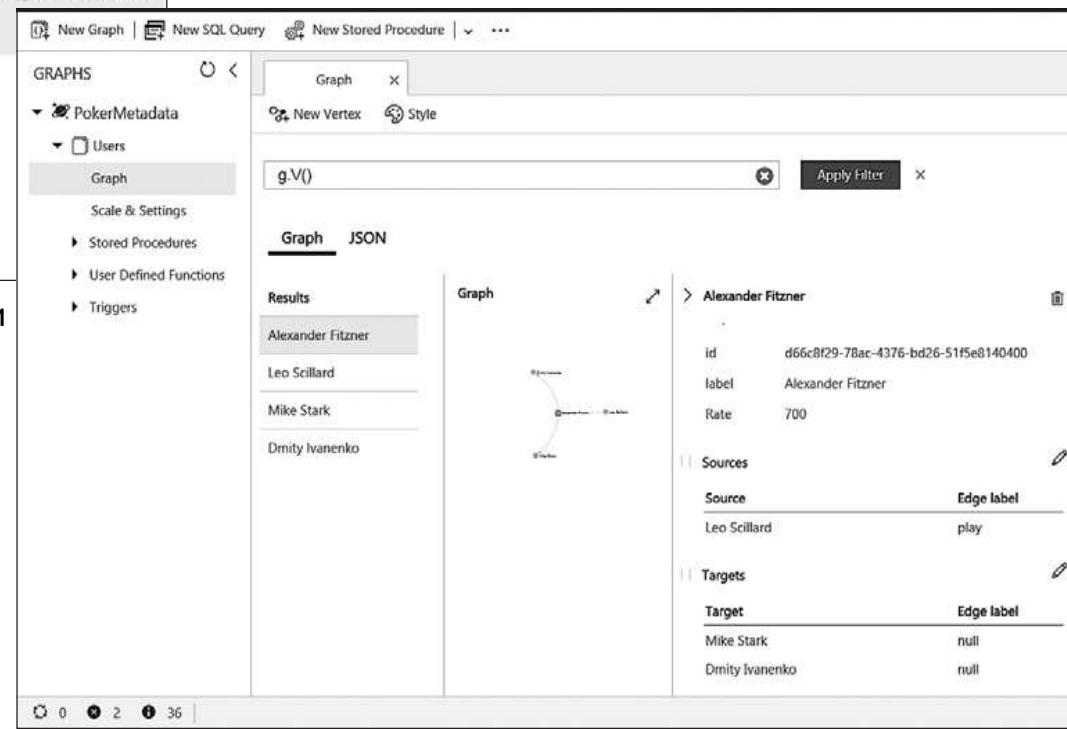
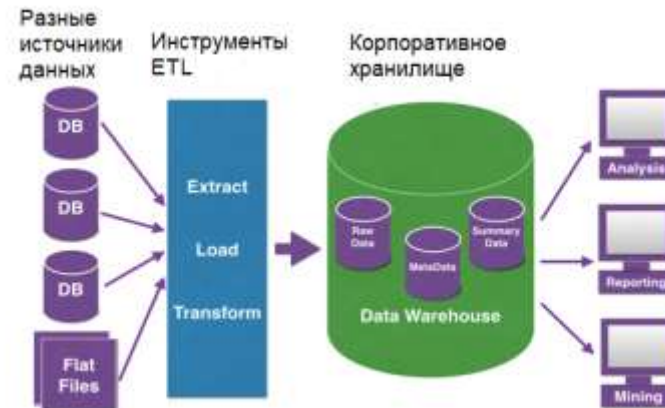


Рис. 6.19. Пользовательский интерфейс вкладки DataExplorer графовой базы данных



# Модуль 1. Основы построения и работы с системами аналитики больших данных



## КРАТКОЕ СОДЕРЖАНИЕ:

### 1. Архитектура систем аналитики больших данных.

1.1. Облачные технологии. Модели развертывания. Способы создания ресурсов в облаке.

1.2. Безопасность облачных ресурсов

1.3. Большие данные и источники данных. Форматы. Преобразование данных из различных форматов. Режимы обработки больших данных.

### 2. Хранение больших данных в облаке.

2.1. Хранилища общего назначения. Форматы хранения данных. Облачное хранилище Microsoft Azure Storage. Облачное хранилище AWS.

2.2. Реляционные базы данных. Azure SQL. AWS RDS.

2.3. Нереляционные базы данных. Сервисы нереляционных баз данных от Azure и AWS.

2.4. Корпоративные хранилища данных (DWH). Azure SQL DWH. AWS RedShift.

2.5. Хранилища данных типа Data Lake. Azure Data Lake Store. AWS Data Lake Solutions.

# ЛЕКЦИЯ 11:

Хранение больших данных в облаке  
Нереляционные базы данных.  
Сервисы нереляционных баз  
данных от Azure и AWS.



# Сервисы нереляционных баз данных от AWS

Облачный провайдер AWS в настоящее время предоставляет два сервиса нереляционных БД: AWS DynamoDB и AWS Neptune. Изучим их подробнее.

Сервис нереляционной базы данных DynamoDB есть реализация БД типа «ключ — значение» от Amazon. [Главные компоненты этого сервиса приведены ниже.](#)

- *Таблица (Table)* — это основная коллекция данных в DynamoDB.
- *Элемент данных (Item)* — по сути, группа атрибутов, отличающаяся от всех других элементов данных таблицы.
- *Атрибут (Attribute)* — элементарная и неделимая единица хранения данных, представляющая собой коллекцию объектов типа «ключ — значение».
- *Первичный ключ (Primary Key)* — идентификатор элемента данных, уникальный в пределах таблицы. В DynamoDB может быть двух разновидностей:
  - *ключ раздела (Partition Key)* — простой первичный ключ, состоящий только из одного атрибута — собственно ключа раздела. DynamoDB использует значение этого ключа как вход внутренней хеш-функции, определяя физический раздел диска, где хранится элемент данных. В этом случае ключ однозначно определяет физическое положение одного элемента данных;
  - *ключ раздела и ключ сортировки (Partition Key и Sort Key)* — составной первичный ключ, состоящий из этих двух атрибутов. В данном случае ключ раздела также служит для указания физического раздела, где хранятся несколько элементов данных, и эти элементы хранятся в отсортированном виде.
- *Вторичный индекс (Secondary Index)* — для таблицы можно вводить один или несколько вторичных индексов, позволяющих создавать высокопроизводительные запросы по полям, отличным от первичного ключа. Вторичные индексы могут быть следующих видов:
  - *глобальный вторичный индекс (Global Secondary Index)* — индекс, у которого ключ раздела и ключ сортировки отличаются от таковых у таблицы;
  - *локальный вторичный индекс (Local Secondary Index)* — индекс с тем же ключом раздела, что и основная таблица, но с иным ключом сортировки.
- *Потоки DynamoDB (DynamoDB Streams)* — поток событий DynamoDB, генерируемый в ответ на изменение, добавление и удаление элементов данных. Можно использовать для запуска сервиса AWS Lambda таким же образом, как и в случае потока событий изменений для сервиса Azure CosmosDB.

# Сервисы нереляционных баз данных от AWS: AWS DynamoDB

Посмотрим на практике, как работать с AWS DynamoDB. Стартовая страница сервиса выглядит так (рис. 6.21).

Чтобы создать таблицу, следует нажать ссылку Create tables, после чего откроется страница, показанная на рис. 6.22. При создании таблицы нужно указать ее имя, первичный ключ (мы не добавляли ключ сортировки) и тип ключа — строковый.

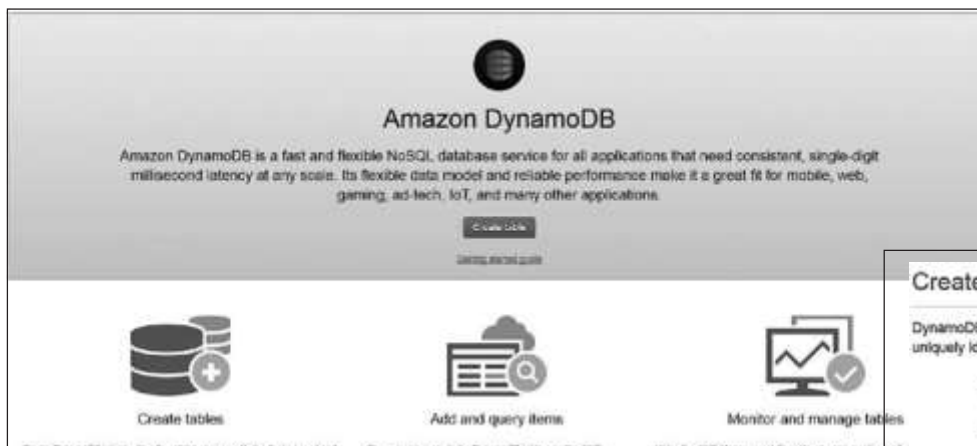


Рис. 6.21. Стартовая страница создания сервиса AWS DynamoDB

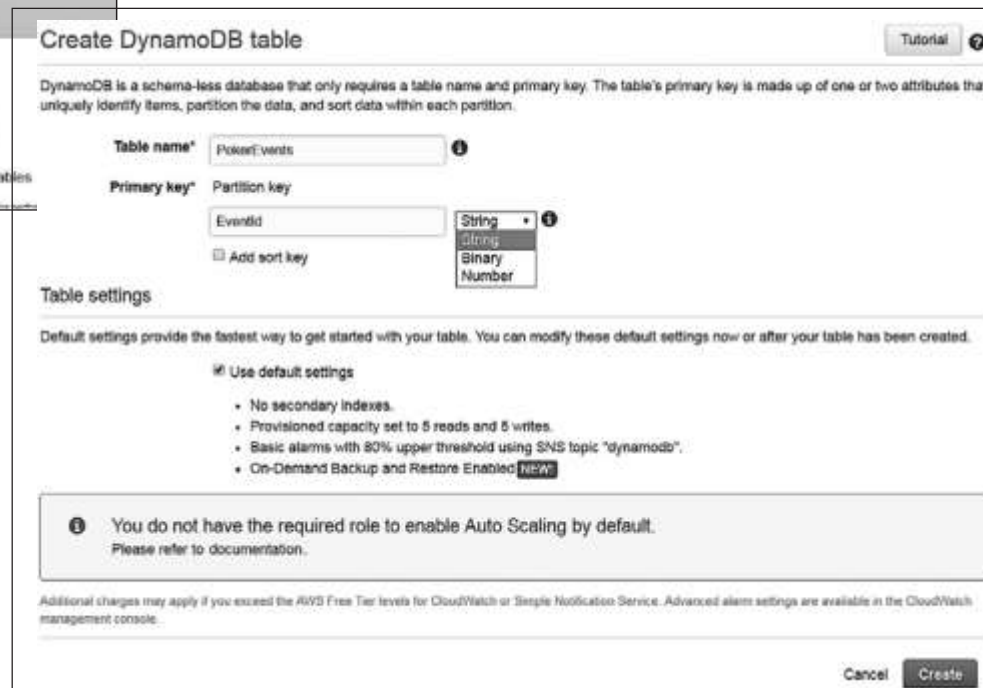


Рис. 6.22. Страница создания и конфигурирования таблицы

# Свойства нереляционных БД

Эти базы предлагают широкий спектр моделей хранения данных, в ряде случаев более соответствующих модели предметной области.

- Графовые базы подходят для предметной области в виде социальных сетей.
- Документоориентированные базы и базы семейства столбцов отлично служат для хранения объектов, плохо поддающихся нормализации: генерируемого пользователями контента, типа комментариев в социальных сетях и т. п.
- Базы данных табличного типа идеальны для хранения многих однотипных записей о событиях в информационной системе, логов, данных телеметрии подключенных устройств и пр.

Следующее важнейшее преимущество нереляционных баз данных — их масштабируемость и опция одновременного обслуживания огромного количества запросов. Это возможно благодаря простоте репликации и отсутствию требования нормализации. По сути, данные хранятся в виде атомарных автономных единиц: строк, ячеек матрицы или документов JSON.

Эти единицы хранения могут быть построены таким образом, что становятся доступными напрямую, без применения сложных запросов с соединениями таблиц по ключу или их объединения. В ряде приложений важным свойством является отсутствие схемы, то есть возможность хранения в единой логической единице базы данных (например, в таблице) объектов с совершенно разными наборами свойств.

Из всех указанных преимуществ вытекают и существенные недостатки такой модели по сравнению с традиционными РБД. Прежде всего, возникают проблемы с целостностью данных в распределенной системе, и невозможно обеспечить сильную согласованность, то есть всегда присутствует неоднозначность и потеря информации при одновременном обновлении одного и того же элемента данных для их разных реплик.

# Свойства нереляционных БД

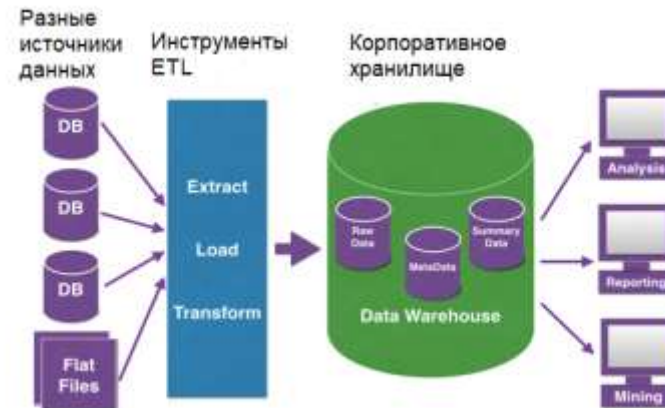
Из-за отсутствия в общем случае сильной согласованности возможности ACID-транзакций, как правило, не обеспечиваются встроенными механизмами базы данных, а должны быть обеспечены всей системой в целом. Обычно в лучшем случае для кластеров NoSQL согласованность может быть итоговой. Но это проблема не NoSQL, а любой распределенной системы хранения данных, и она носит название «теорема CAP». Типичная формулировка данной теоремы состоит в том, что в любой реализации распределенной системы хранения и вычислений можно обеспечить не более двух из трех следующих свойств:

- *согласованность данных* — во всех узлах системы в каждый момент времени данные не противоречат друг другу;
- *доступность* — любой запрос к системе должен завершаться корректным ответом (возможно, при этом данные будут противоречащими);
- *устойчивостью к разделению* — способность системы к расщеплению на независимые фрагменты без уменьшения доступности.

Как правило, в нереляционных БД согласованность данных приносится в жертву масштабируемости и доступности.

Следующий недостаток нереляционных БД — трудности построения аналитических запросов на выборку данных. Документоориентированные базы, семейство столбцов и табличные БД идеальны для хранения многих порций элементарных данных и выборки их отфильтрованного подмножества. Аналитические запросы, помимо фильтрации, требуют возможности объединения данных из разных коллекций, выбора подмножества полей, агрегации и выполнения арифметических операций.

# Модуль 1. Основы построения и работы с системами аналитики больших данных



## КРАТКОЕ СОДЕРЖАНИЕ:

### 1. Архитектура систем аналитики больших данных.

1.1. Облачные технологии. Модели развертывания. Способы создания ресурсов в облаке.

1.2. Безопасность облачных ресурсов

1.3. Большие данные и источники данных. Форматы. Преобразование данных из различных форматов. Режимы обработки больших данных.

### 2. Хранение больших данных в облаке.

2.1. Хранилища общего назначения. Форматы хранения данных. Облачное хранилище Microsoft Azure Storage. Облачное хранилище AWS.

2.2. Реляционные базы данных. Azure SQL. AWS RDS.

2.3. Нереляционные базы данных. Сервисы нереляционных баз данных от Azure и AWS.

2.4. Корпоративные хранилища данных (DWH). Azure SQL DWH. AWS RedShift.

2.5. Хранилища данных типа Data Lake. Azure Data Lake Store. AWS Data Lake Solutions.

# ЛЕКЦИЯ 12:

Хранение больших данных в облаке  
Корпоративные хранилища больших  
данных (DWH).  
Azure SQL DWH.  
AWS RedShift.





# Общий обзор реляционных хранилищ данных

Реляционная база данных предназначена для выполнения двух основных функций: OLAP и OLTP.

Преимущественно РБД используются для целей OLTP. Это приводит к тому, что, с одной стороны, обычные реляционные базы имеют ограничения по размеру, а с другой — выполнить высокопроизводительные запросы к большим массивам данных может быть затруднительно.

Проблема нехватки ресурсов баз может обостриться ввиду необходимости выполнять запросы к таблицам из разных БД. Она частично устраняется с помощью поднятия ценового уровня базы (для **Azure SQL**) или типа экземпляра (для **AWS RDS**) перед запуском всех BI-запросов и опускания после их завершения.

Но такой подход ограничен тем, что нужно разделить по времени периоды работы сервисов бизнес-аналитики и основного приложения. Кроме того, нерешенной является проблема выполнения запросов среди многих таблиц многих внешних БД. Да и предельно высокий ценовой уровень одиночной базы может оказаться недостаточным.

Как быть в этом случае? Вот тут-то на помощь и приходит **DWH (Data Warehouse)** — «склад данных» — реляционное хранилище данных, оптимизированное для хранения огромных массивов данных и выполнения запросов к ним (рис. 7.1).

# Наполнение хранилища DWH-Data Warehouse и доступ сторонних сервисов к нему

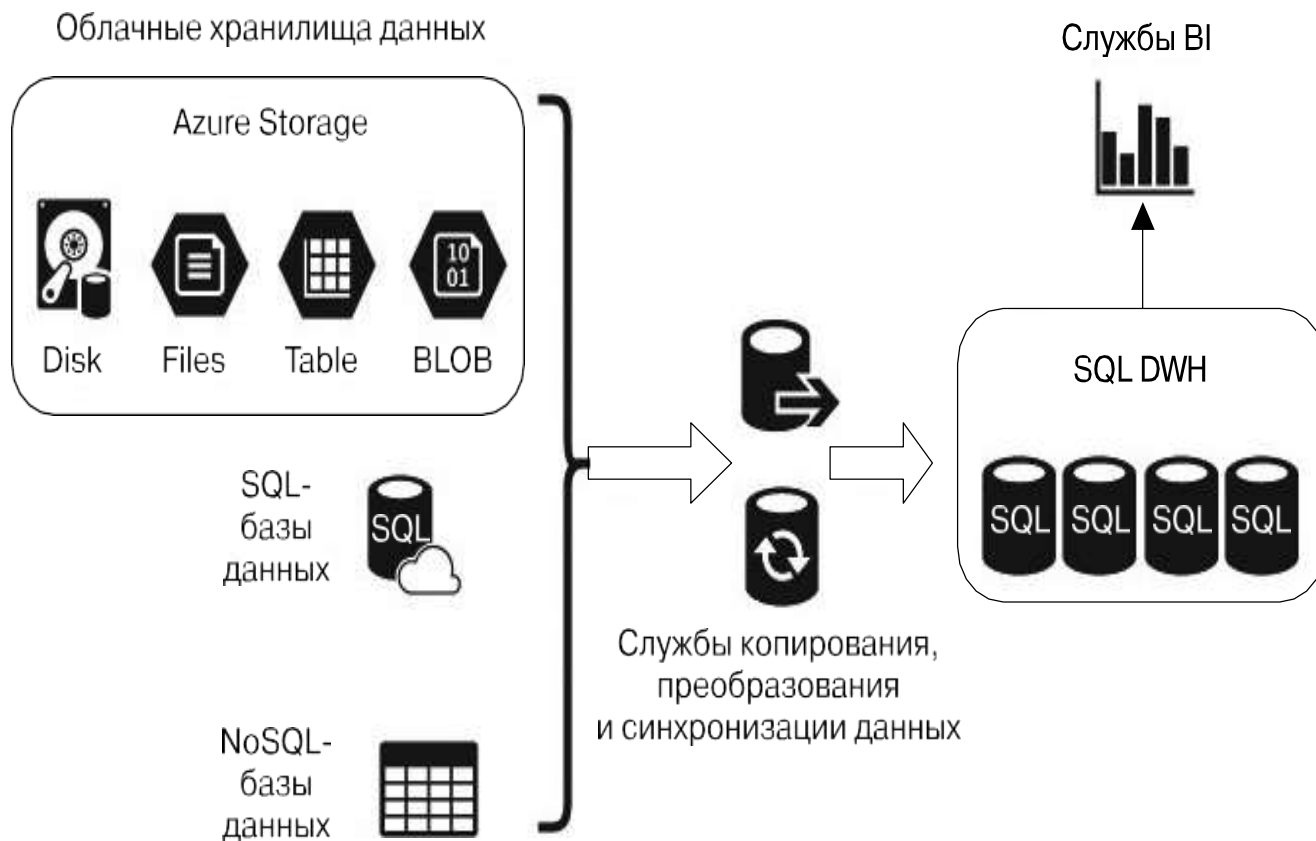


Рис. 7.1. Наполнение хранилища Data Warehouse и доступ сторонних сервисов к нему

# Наполнение хранилища DWH- Data Warehouse

Высокая производительность аналитических запросов достигается благодаря массивно-параллельной архитектуре и распараллеливанию запроса для выполнения несколькими серверами одновременно.

В данном случае тоже необходимо разместить информацию из различных БД в одном общем сервисе — «складе» (DWH). Преимущество такого подхода в том, что SQL DWH позволяет использовать стандартные SQL-запросы, хранимые процедуры и др., а следовательно, такое хранилище совместимо со всеми стандартными средствами и инструментами BI.

**Реляционное хранилище данных (SQL DWH)** представляет собой массивно-параллельное хранилище, состоящее из кластера серверов трех типов: головного (управляющего), узлов вычисления и томов хранения (рис. 7.2).

# Общая архитектура реляционного хранилища данных

**Реляционное хранилище данных (SQL DWH)** представляет собой массивно-параллельное хранилище, состоящее из кластера серверов трех типов: головного (управляющего), узлов вычисления и томов хранения (рис. 7.2).

Клиент (а точнее, программный клиент) может взаимодействовать с реляционным хранилищем только через головной узел. Последний отвечает за распределение данных по вычислительным узлам, в ряде случаев являющимся одновременно и узлами хранения. Кроме того, головной узел компилирует запросы языка SQL в команды кластеру, которые обеспечивают параллельное выполнение, а затем собирает результат вместе и выдает назад клиенту. Реляционное хранилище данных, в отличие от реляционных баз данных, оптимизировано только для целей OLAP. Выполнение транзакций в DWH возможно, но это несвойственно хранилищу, основная задача которого — хранить информацию и предоставлять доступ к ней с помощью интерфейса SQL. При этом производительность запросов на чтение данных существенно выше, чем для обычной БД за счет параллельного выполнения и иного физического представления данных, другого плана выполнения запросов и индексов.

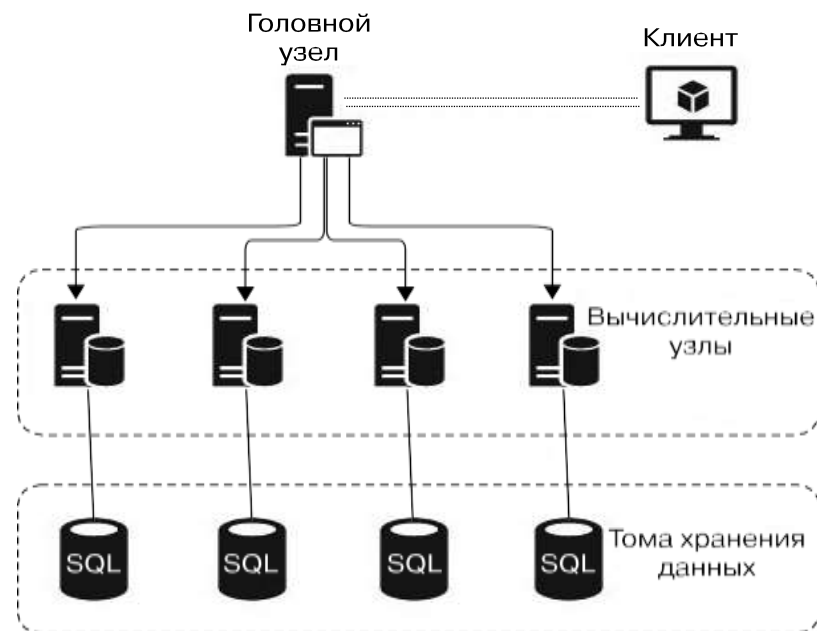


Рис. 7.2. Общая архитектура реляционного хранилища данных

# Общая архитектура реляционного хранилища данных

Рассмотрим вопрос **параллельного выполнения запросов**. Необходимо разделить данные между вычислительными узлами. Это можно сделать следующими способами.

- **Во-первых, в каждом узле разместить идентичные копии данных и разделить запрос таким образом, чтобы каждая его часть в вычислительном узле обрабатывала свой набор строк.** Как уже отмечалось, за разделение запроса на подзапросы и соединение их результатов отвечает головной узел (см. рис. 7.2).

Такая архитектура очень надежна: потеря одного узла никак не повлияет на информацию в хранилище. Недостаток данной архитектуры в том, что из-за ограниченного объема дисков одного узла в нем невозможно хранить очень большие объемы данных.

- **Во-вторых, можно размещать данные в различных узлах без дублирования.** При этом вероятно как псевдослучайное распределение строк данных по узлам, так и последовательное, в порядке роста значения ключа. В этом случае возможно хранение и обработка гораздо бо'льших объемов данных, чем в случае их полного копирования в каждом узле.

# Общая архитектура реляционного хранилища данных

Итак, *реляционное хранилище данных* применяется, когда необходимо получить централизованное хранилище информации, позволяющее выполнять запросы с помощью стандартного синтаксиса SQL, поддерживаемого большинством систем BI.

Для этого нужно *извлечь* данные из внешних источников (*extract*), преобразовать их к виду, удобному для добавления в хранилище с необходимой *обработкой, фильтрацией* и пр. (*transform*) и *загрузить* их в хранилище (*load*).

**Такой подход называется ETL и является традиционным.**

- Он очень удобен в случае, когда внешние источники данных очень разнородны и все вместе или по отдельности не поддерживают встроенный механизм выполнения аналитических запросов.
- Недостаток ETL состоит в том, что для наполнения реляционного хранилища обязательно нужен сервис копирования и трансформации. Критически важный момент для ETL — метаданные о сторонних источниках: что в них содержится, в каком формате и др. Эти метаданные в облачных средах могут быть представлены в специальных сервисах — каталогах данных (data catalog).

# Общая архитектура реляционного хранилища данных

Рассмотрим логическую структуру информации в реляционном хранилище.

Ниже представлены несколько принципов организации реляционных хранилищ данных.

- Данные в хранилище должны быть объединены в соответствии с категориями предметной области, а не их физическими источниками.
- Централизованное хранилище должно хранить в своем составе данные, относящиеся ко всей системе в целом, а не только к отдельным аспектам. Только в таком случае можно построить комплексные аналитические запросы.
- Данные в хранилище содержатся и поступают извне, но не создаются.
- Говорить о корректности данных в хранилище можно только с привязкой их к конкретному временно́му промежутку.

Существует две стратегии наполнения хранилища данными: стратегия полного обновления (каждый раз происходит полное обновление данных из внешних источников) и инкрементального (обновления только тех данных, которые были изменены в процессе работы OLTP-подсистемы). В плане реализации, безусловно, проще стратегия полного обновления, в то время как инкрементальное требует использования сложных ETL-процедур замены обновленных элементов, вставки новых и удаления тех, что были удалены в OLTP.

# Общие модели построения реляционных хранилищ

Есть две общие модели построения реляционных хранилищ: *нормализованные хранилища* и *хранилища с измерениями*.

- Построение нормализованного хранилища в общем не отличается от построения нормализованной реляционной базы данных. Информация хранится в связанных таблицах в третьей нормальной форме, и, как следствие, требуется выполнение выборки из многих таблиц, что потенциально ведет к усложнению запросов и падению производительности.
- Хранилище с измерениями — это денормализованная форма хранения информации. В свою очередь, возможны две архитектуры хранилища с измерениями — в виде схем «звезда» и «снежинка».

В схеме «звезда» информация хранится в таблицах двух типов: центральной *таблицы фактов* и многочисленных *таблиц измерений* (рис. 7.3).



# Общие модели построения реляционных хранилищ

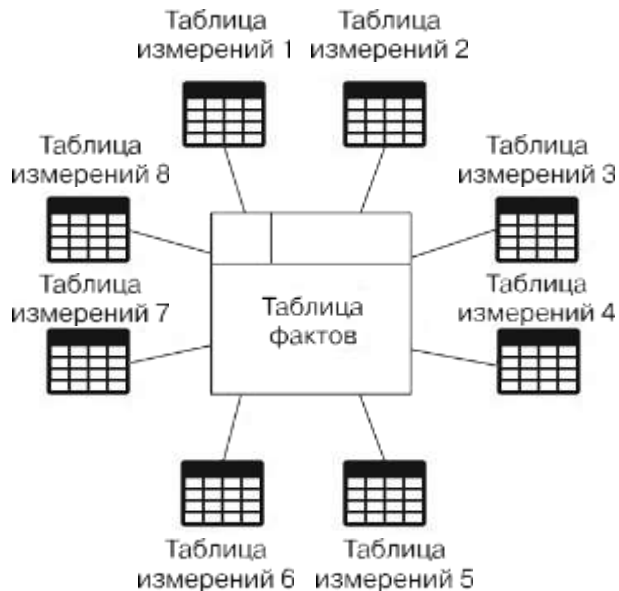


Рис. 7.3. Архитектура хранилища данных в виде схемы «звезда»

В центре звездообразной структуры находится **таблица фактов**, содержащая информацию, которая будет анализироваться.

Она может включать факты (а по сути, записи), связанные с событиями или состоянием объекта, с транзакциями и пр. Таблица содержит первичный ключ, числовые поля, характеризующие данный факт, и внешние ключи, ссылающиеся на таблицу измерений.

Числовые поля должны включать аддитивные значения, подлежащие анализу (в том числе суммированию, вычислению среднего значения и др.). Таблицы измерений нужны для хранения атрибутов фактов (как правило, текстовых), которые не могут быть использованы для численных операций со строками таблицы фактов.

Иногда требуется добавить элемент нормализации и расщепить некоторые таблицы изменений на несколько таблиц — это схема «снежинка» (рис. 7.4).

Выбор той или иной структуры реляционного хранилища данных и построение запросов (OLAP — кубы, DataMart — витрины данных) выходит за рамки данной книги. В следующих разделах подробнее рассмотрим реализации хранилищ дан- ных от Azure SQL DWH и AWS RedShift.

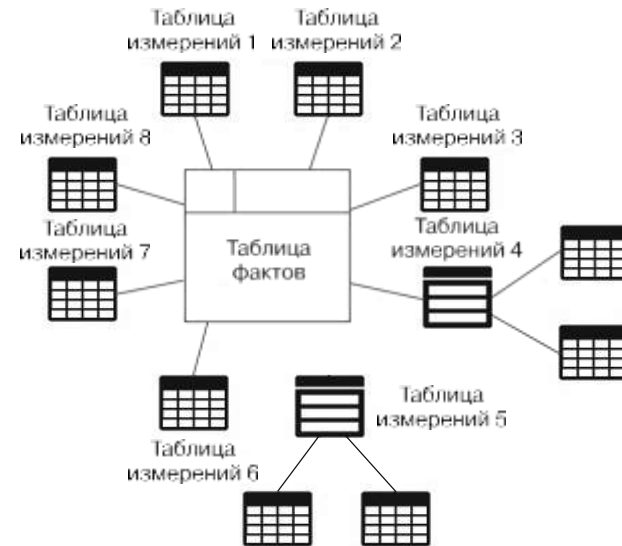
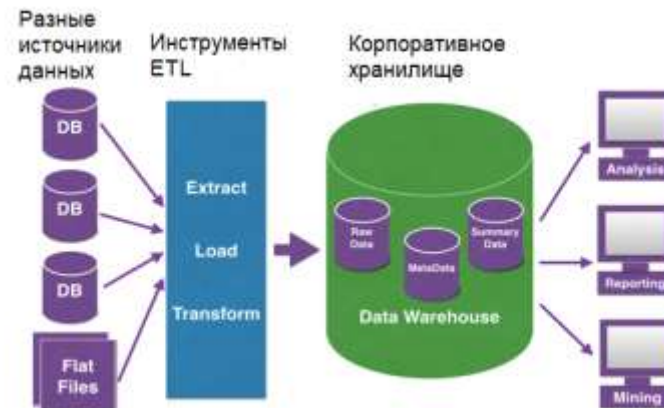


Рис. 7.4. Архитектура хранилища данных в виде схемы «снежинка»

# Модуль 1. Основы построения и работы с системами аналитики больших данных



## КРАТКОЕ СОДЕРЖАНИЕ:

### 1. Архитектура систем аналитики больших данных.

- 1.1. Облачные технологии. Модели развертывания. Способы создания ресурсов в облаке.
- 1.2. Безопасность облачных ресурсов
- 1.3. Большие данные и источники данных. Форматы. Преобразование данных из различных форматов. Режимы обработки больших данных.

### 2. Хранение больших данных в облаке.

- 2.1. Хранилища общего назначения. Форматы хранения данных. Облачное хранилище Microsoft Azure Storage. Облачное хранилище AWS.
- 2.2. Реляционные базы данных. Azure SQL. AWS RDS.
- 2.3. Нереляционные базы данных. Сервисы нереляционных баз данных от Azure и AWS.
- 2.4. Корпоративные хранилища данных (DWH). Azure SQL DWH. AWS RedShift.
- 2.5. Хранилища данных типа Data Lake. Azure Data Lake Store. AWS Data Lake Solutions.

# ЛЕКЦИЯ 13:

Хранение больших данных в облаке  
Корпоративные хранилища больших  
данных (DWH).  
Azure SQL DWH.  
AWS RedShift.



# Azure SQL DWH

**Реляционное хранилище Azure SQL DWH** — облачный сервис от Microsoft Azure, построенный на основе сервиса Azure SQL. Этот сервис использует технологию Microsoft PolyBase для построения запросов к данным, расположенным как в хранилище Azure Blob Storage, так и в базах данных Azure SQL. Рассмотрим основные концепции сервиса Azure SQL DWH.

Архитектура хранилища Azure SQL DWH соответствует представленной на рис. 7.2.

Как головной узел, так и вычислительные узлы содержат экземпляры системного сервиса **DMS (Data Movement Service)**, отвечающего за передачу данных между узлами. Данные физически хранятся в Azure Storage. Такое физическое разделение их хранения и параллельного выполнения запросов обеспечивает возможность независимого масштабирования объемов хранения и вычислительных мощностей кластера.

Непосредственно из Azure Storage данные разделяются по вычислительным узлам с помощью одного из **трех видов распределения (distributions)**.

Каждый запрос разделяется на 60 меньших фрагментов, каждая из которых выполняется на вычислительном узле. Каждый вычислительный узел работает с 1 до 60 фрагментов в зависимости от его ценового уровня. На наивысшем уровне (максимальная производительность вычислительного узла) используется одно распределение на узел, при минимальных ресурсах — все распределения на один узел.

Существует три возможных паттерна данных, влияющих на производительность запросов:

- распределение с помощью хеш-функции (hash);
- последовательное, или циклическое, распределение (round robin);
- репликация (replicate).

*Идея использования хеш-функции для распределения строк из исходной таблицы по вычислительным узлам состоит в том, что строки равномерно, независимо и случайно разделяются по вычислительным узлам (рис. 7.5). Это позволяет строить оптимальные запросы с помощью соединения таблиц и агрегации данных. При этом каждая строка таблицы относится к одному распределению.*

# Azure SQL DWH

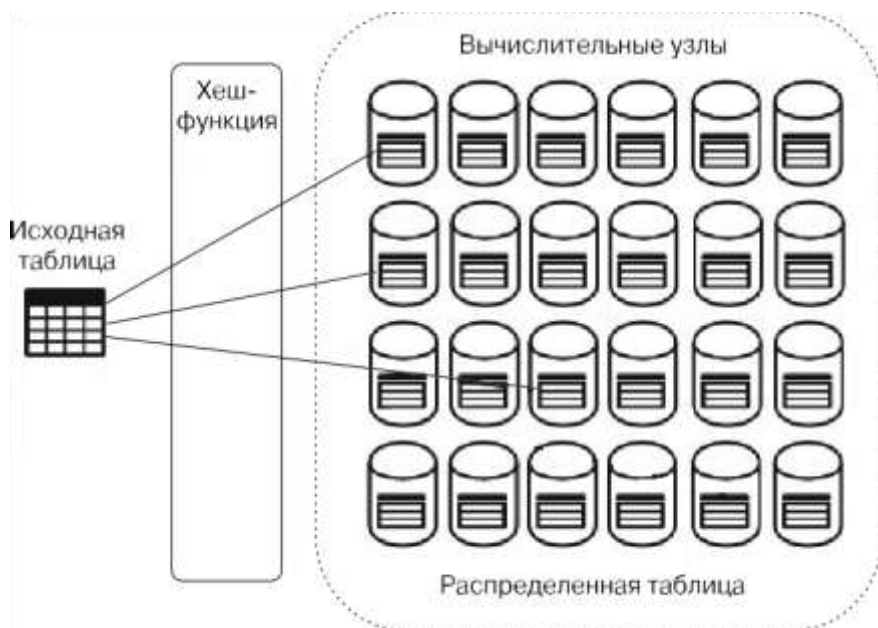


Рис. 7.5. Распределение данных таблицы по вычислительным узлам с помощью хеш-функции

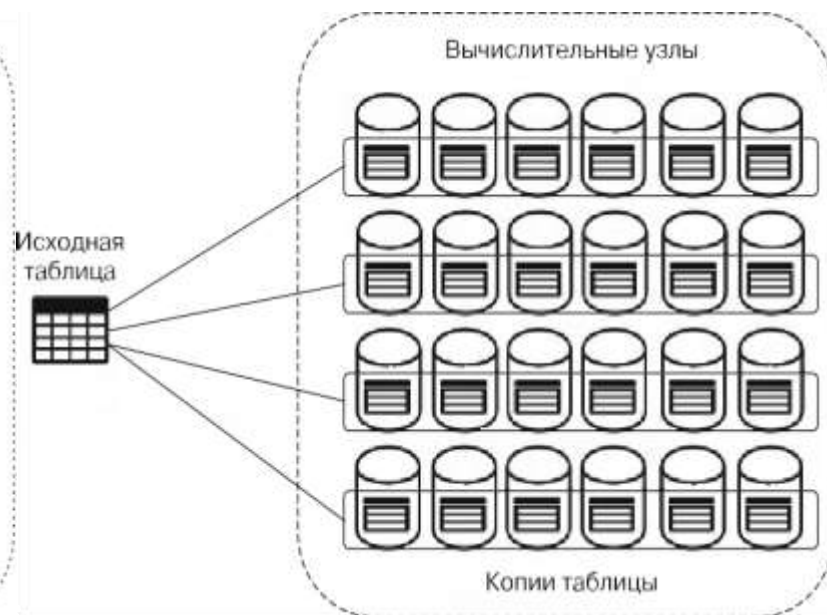


Рис. 7.6. Репликация таблицы по вычислительным узлам

- **Циклическое распределение** строк по вычислительным узлам подразумевает последовательное равномерное наполнение узлов со случайным выбором начального узла. Такое распределение обеспечивает высокую производительность операций выборки данных в случае, когда в соединении таблиц нет нужды, поскольку оно потребует выполнения дополнительных «перетасовок» строк между узлами.
- **Репликация** применяется при наличии таблиц небольшого размера и состоит в том, что такая таблица копируется на все вычислительные узлы (рис. 7.6). Это позволяет добиться высокой производительности любого типа запроса и одновременно увеличить надежность системы, поскольку отказ одного вычислительного узла или более повлияет лишь на производительность.

# Azure SQL DWH

Теперь посмотрим, как работать с Azure SQL DWH из веб-портала. Для этого во вкладке добавления ресурсов в строке поиска следует написать SQL Data Warehouse (рис. 7.7).

После нажатия кнопки Create (Создать) откроется следующая форма настройки реляционного хранилища Azure SQL DWH (рис. 7.8). Здесь нужно указать имя базы данных (Database name), выбрать ресурсную группу (Resource group), сервер (Server) и ценовой уровень производительности (Performance tier).

Для реляционного хранилища можно настроить ценовой уровень по двум разным критериям: гибкости (Optimized for Elasticity) и вычислительным возможностям (Optimized for Compute) (рис. 7.9).

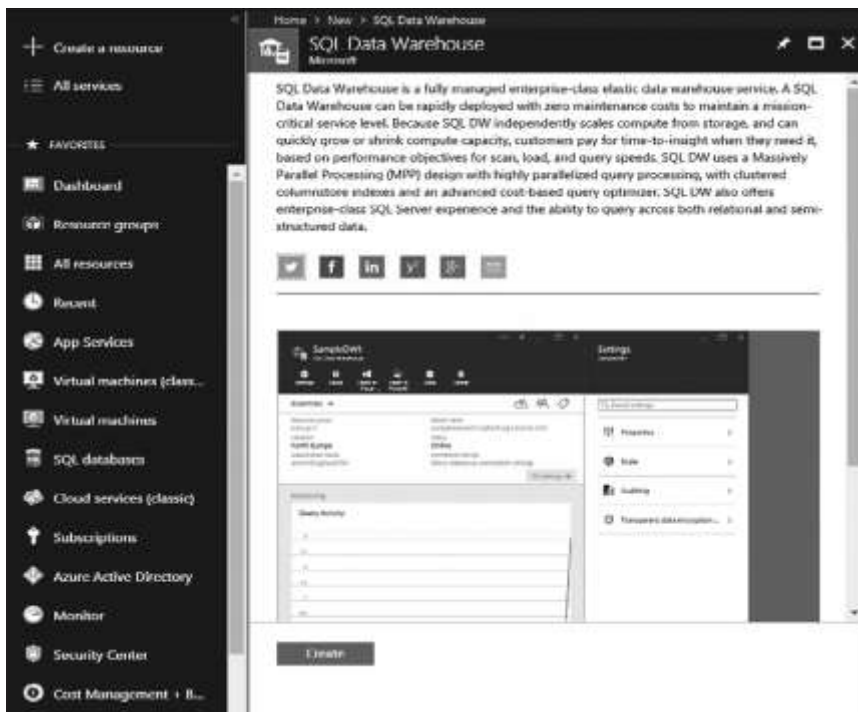


Рис. 7.7. Внешний вид вкладки создания хранилища данных

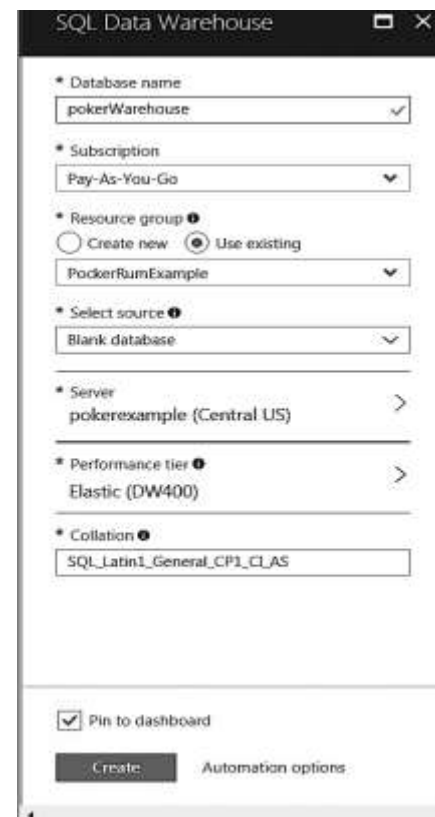


Рис. 7.8. Форма конфигурирования Azure SQL DWH

# Azure SQL DWH

Для реляционного хранилища можно настроить ценовой уровень по двум разным критериям: гибкости (Optimized for Elasticity) и вычислительным возможностям (Optimized for Compute) (рис. 7.9).

The screenshot displays the 'Configure Performance' interface for an Azure SQL Data Warehouse. On the left, a sidebar contains configuration fields: Database name (pokerWarehouse), Subscription (Pay-As-You-Go), Resource group (PockerRumExample), Select source (Blank database), Server (pokerexample (Central US)), Performance tier (Elastic (DW400)), and Collation (SQL\_Latin1\_General\_CP1\_CI\_AS). The main area is split into two columns: 'Optimized for Elasticity' and 'Optimized for Compute (Preview)'. The 'Optimized for Elasticity' section is selected, showing a starting price of 1.51 USD / hour. A slider below it is positioned at 'DW100', which corresponds to a price of 1.51 USD / hour and 100 DWU. The 'Optimized for Compute (Preview)' section is currently inactive. At the bottom, there are 'Create' and 'Apply' buttons.

Рис. 7.9. Настройка ценового уровня Azure SQL DWH

# Azure SQL DWH

Дадим пояснение величинам, используемым для оценки производительности реляционного хранилища — DWU и cDWU:

- DWU (Data Warehouse Unit) представляет собой абстрактную величину, описывающую в нормализованном виде доступные вычислительные ресурсы: CPU, память и полоса пропускания дисков данных — IOPS;
- cDWU (compute Data Warehouse Unit) — с одной стороны, описывает скорость чтения информации из Azure Storage, а с другой — скорость выполнения запросов.



# Azure SQL DWH

После создания **Azure SQL DWH** панель будет выглядеть следующим образом (рис. 7.10). Она отображает основные возможности Azure SQL DWH, которые близки к таковым у базы данных Azure SQL.

На панели представлены отдельный сервис загрузки данных — **Load Data** (поддерживается загрузка с помощью RedGate и Azure DataFactory), интеграция с сервисом **SSAS** (ссылка Model and Cache Data), сервис бизнес-аналитики (ссылка Open in PowerBI) и, конечно, сервисы мониторинга (Monitoring), масштабирования (Scale), выполнения запросов в онлайн-редакторе (ссылка Open Editor). Кроме того, доступны опции бэкапа, аудита и шифрования.

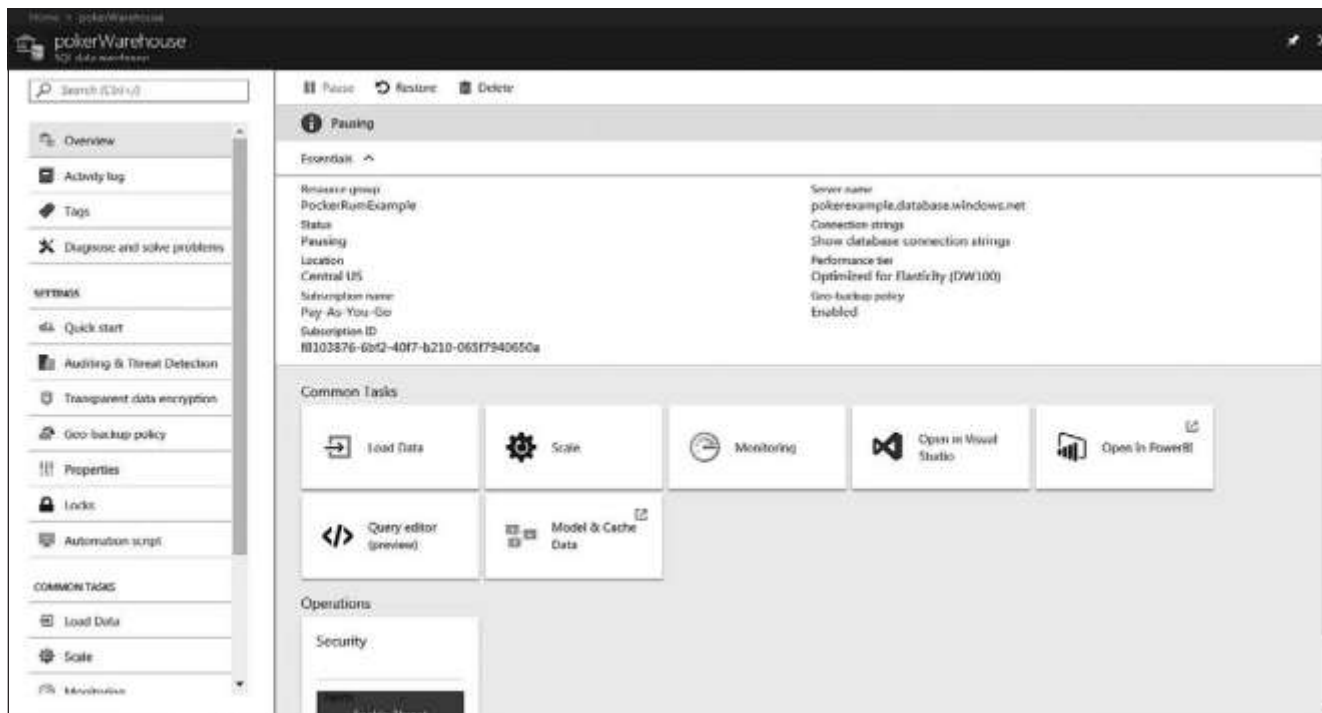


Рис. 7.10. Внешний вид общей панели мониторинга Azure SQL DWH

# AWS RedShift

**Сервис AWS RedShift** представляет собой сервис реляционного хранилища от AWS. Архитектура этого сервиса в целом совпадает с приведенной на рис. 7.2, но имеет ряд отличий.

**AWS RedShift состоит из ведущего (головного) узла и вычислительных узлов.** Клиентское приложение взаимодействует с ведущим узлом с помощью стандартных протоколов PostgreSQL JDBC и ODBC.

**Вычислительные узлы AWS RedShift DWH** состоят из разделов (node slices). Каждый из разделов представляет собой выделенную порцию вычислительных ресурсов: CPU, памяти, IOPS.

**Ведущий узел** разделяет данные, взятые из хранилища, между вычислительными узлами, а точнее, между разделами вычислительных узлов. Это и позволяет выполнять запросы в параллельном режиме. Количество разделов определяется размером вычислительного узла, а точнее, типом экземпляра.

Узлы кластера содержат одну или более БД, в которых хранятся данные. Все эти базы взаимодействуют с ведущим узлом, который и занимается распараллеливанием запроса.

# AWS RedShift

Чтобы обеспечить высокую производительность выполнения запроса, помимо массивно-параллельного выполнения запросов данных используются **различные механизмы**.

- **Во-первых, это *столбцовое, или колоночное, хранилище (Columnar Store)*** — способ организации хранения информации в таблице, обеспечивающий значительное снижение общего числа операций чтения-записи. При этом каждая строка в типичной БД сохраняется в виде блока на диске. В столбцовом хранилище в едином блоке хранятся данные нескольких столбцов. Сам движок AWS RedShift DWH преобразует данные из такой формы представления в табличную. Увеличение скорости работы в подобном случае достигается за счет того, что каждый блок хранит информацию для нескольких строк таблицы, например для  $n$ , соответственно, ускорение будет равно приблизительно  $n$ , так как за один акт чтения блока читаются данные для ячейки у  $n$  строк. Чтобы добиться повышения общей производительности кластера, необходимо определить один столбец как *ключ распределения (distribution key)*, благодаря которому данные будут распределяться по узлам кластера. Выбор правильного ключа обеспечит высокую производительность запросов.
- **Во-вторых, это *кэширование результатов (Result Caching)* в головном узле.** При этом перед выполнением запроса проверяется состояние кэша. Нужно учитывать, что кэширование эффективно для запросов, возвращающих небольшое количество результатов. По умолчанию оно включено. И наконец, головной узел обеспечивает компиляцию и оптимизацию кода запроса, кэширует скомпилированный результат исполнения в вычислительных узлах, обеспечивая ускорение сложных запросов. Такой механизм приводит к тому, что выполнение одного и того же запроса во второй и третий раз займет меньше время, чем в первый.

# AWS RedShift

Теперь рассмотрим, как создавать и работать с сервисом с AWS RedShift с помощью консоли AWS. Общий вид панели создания кластера показан на рис. 7.11.

Чтобы начать создание кластера, следует выбрать ссылку Launch cluster. В результате откроется панель конфигурирования кластера, показанная на рис. 7.12.



Рис. 7.11. Общий вид панели создания кластера

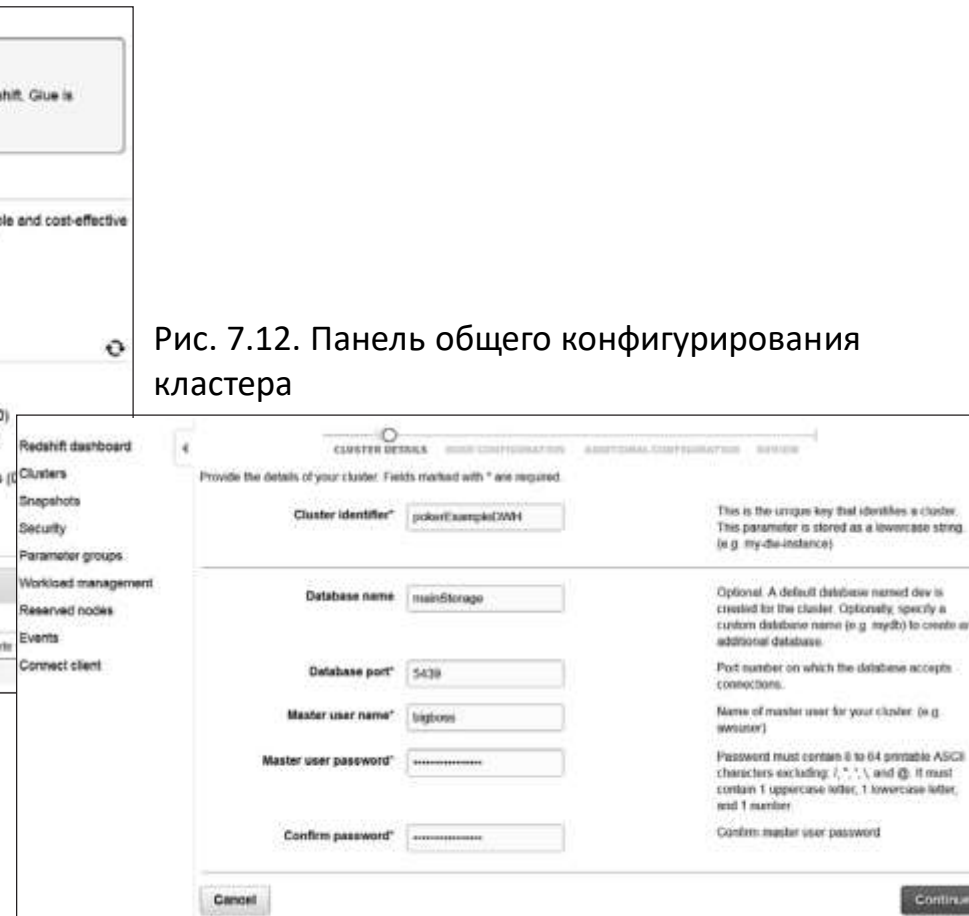
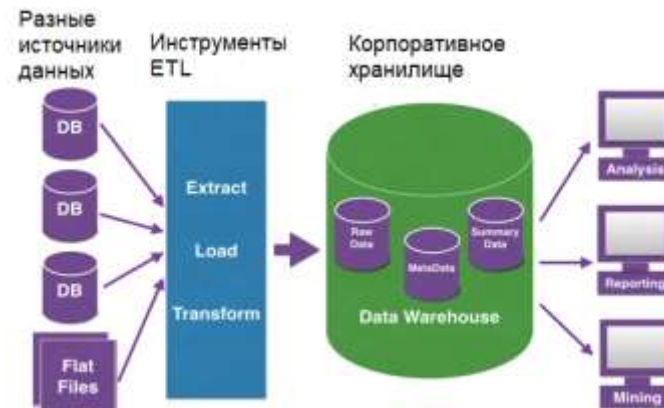


Рис. 7.12. Панель общего конфигурирования кластера

# Модуль 1. Основы построения и работы с системами аналитики больших данных



## КРАТКОЕ СОДЕРЖАНИЕ:

### 1. Архитектура систем аналитики больших данных.

1.1. Облачные технологии. Модели развертывания. Способы создания ресурсов в облаке.

1.2. Безопасность облачных ресурсов

1.3. Большие данные и источники данных. Форматы. Преобразование данных из различных форматов. Режимы обработки больших данных.

### 2. Хранение больших данных в облаке.

2.1. Хранилища общего назначения. Форматы хранения данных. Облачное хранилище Microsoft Azure Storage. Облачное хранилище AWS.

2.2. Реляционные базы данных. Azure SQL. AWS RDS.

2.3. Нереляционные базы данных. Сервисы нереляционных баз данных от Azure и AWS.

2.4. Корпоративные хранилища данных (DWH). Azure SQL DWH. AWS RedShift.

2.5. Хранилища данных типа Data Lake. Azure Data Lake Store. AWS Data Lake Solutions.

# ЛЕКЦИЯ 14:

Хранение больших данных в облаке

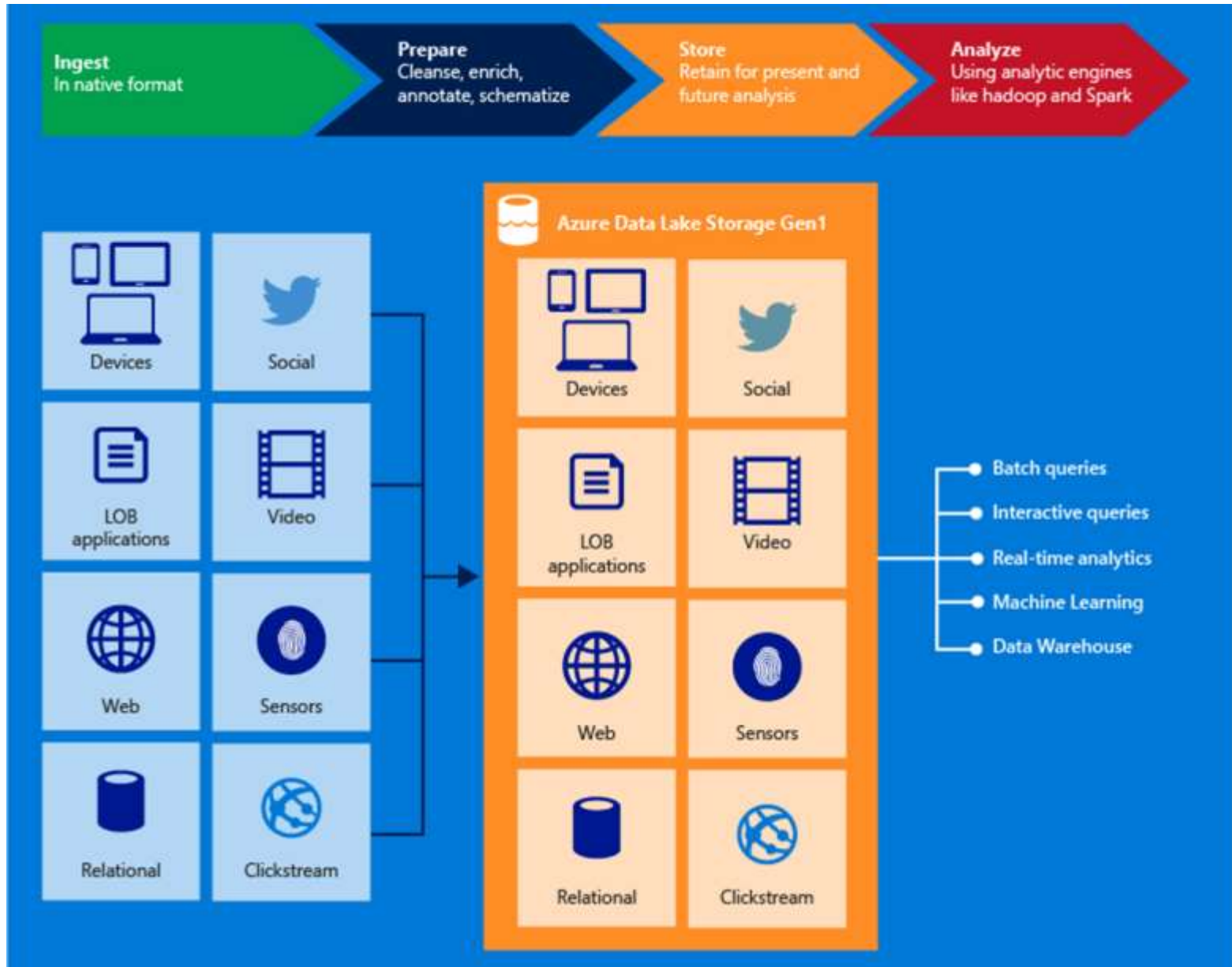
Хранилища данных типа Data Lake.

Azure Data Lake Store.

AWS Data Lake Solutions.



# Что представляет собой Azure Data Lake Storage



# Хранилища данных типа Data Lake: общий обзор специализированных облачных хранилищ больших данных

Облачные хранилища файлов допускают хранение огромного количества файлов, суммарный объем которых исчисляется многими терабайтами и даже петабайтами. *Но в хранилищах эти файлы просто хранятся, а как быть, если нужно провести общее исследование информации в них?*

Например, для многих сотен файлов логов суммарным объемом несколько гигабайт сгруппировать IP-адреса запросов, чтобы определить страну-источник последних. Или совершить более сложные исследования логов с многих сотен серверов для определения корреляции в событиях, узких мест производительности и пр. Для выполнения подобных действий необходим сторонний сервис, которому это под силу, но он должен иметь доступ к каждому из логов.

У AWS есть такие сервисы: AWS Athena и AWS RedShift Spectrum, и они специально предназначены для анализа информации в файлах с помощью SQL-подобного синтаксиса.

*Если задачу анализа лог-файлов общего назначения можно считать решенной, то для более сложных случаев указанные сервисы недостаточно приспособлены. Как быть, если лог-файлов очень много (терабайты) и необходимо получить периодический отчет, применить алгоритмы машинного обучения или выполнить совместный анализ информации из файлового хранилища и, скажем, реляционной базы данных?*

Для решения указанных проблем нужно переместить эти файлы в хранилища, допускающие подобные манипуляции.

Но если требуется применить машинное обучение или выполнить пакетный анализ огромного количества данных, то нужно использовать HDFS-совместимое хранилище. Вообще, как правило, реляционное хранилище данных — последнее звено в цепочке системы анализа больших данных.

## **Эта цепочка включает:**

- прием потоков данных;
- хранение сырых данных в нереляционном хранилище;
- копирование и агрегацию данных в HDFS-совместимое хранилище Data Lake;
- выполнение интеллектуального анализа данных;
- помещение результата в реляционное хранилище.



# Хранилища данных типа Data Lake: общий обзор специализированных облачных хранилищ больших данных

Реляционное хранилище обеспечивает доступ к информации для сервисов бизнес-аналитики (рис. 8.1). Но на практике есть пример архитектуры, в которой было два оконечных хранилища: DWH и Data Lake — и каждое из них выполняет свою задачу.

В этом случае можно применить сервисы «из мира» [Hadoop \(MapReduce, Spark, Pig, Hive и др.\)](#) или облачных средств аналитики HDFS-совместимых хранилищ ([Azure Data Lake Analytics](#)). Такой подход именуется Data Lake («озеро данных»). Data Lake является хранилищем структурированных, неструктурированных и частично структурированных файлов различного типа и совместно с сервисами экосистемы Hadoop образует «склад данных», который отличается от традиционного SQL DWH, но обладает гораздо большими возможностями, предоставляемыми средствами экосистемы Hadoop.

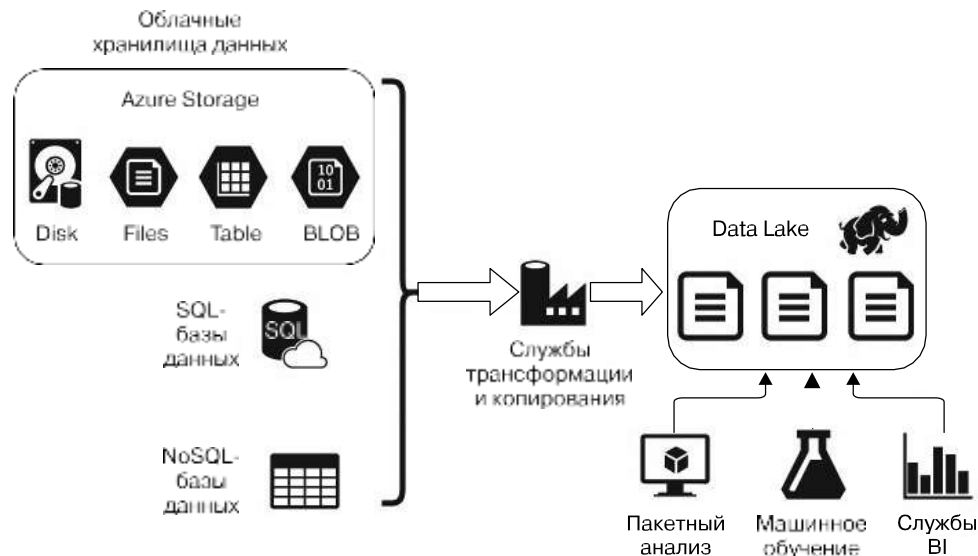


Рис. 8.1. Наполнение хранилища DataLake данными из иных облачных хранилищ и доступ сторонних сервисов к нему

# Хранилища данных типа Data Lake

**Data Lake** позволяет применять ELT, то есть сначала загрузить файл (в ряде случаев просто копировать), а потом, если надо, преобразовать.

Кроме того, в **Data Lake** все данные хранятся как файлы в специализированной файловой системе HDFS — каждый файл имеет имя, путь, и эти параметры напрямую фигурируют в запросах обработки данных. В SQL DWH же, как и в любой СУБД, хранение информации полностью отделено от ее логического представления.

Итак, опишем кратко HDFS и поясним, почему она так важна для построения распределенных масштабируемых хранилищ, допускающих массивно-параллельную обработку хранящейся в них информации.

**HDFS (Hadoop Distributed File System)** — распределенная файловая система Hadoop, предназначенная для хранения файлов, которые поблочно распределяются между узлами кластера, лежащего в основе HDFS.

Каждый блок (за исключением последнего блока файла, служебного) можно расположить в одном или нескольких узлах одновременно, что определяется коэффициентом репликации. Таким образом, коэффициент определяет количество узлов, в которых может быть размещен блок, и является параметром, настраиваемым на уровне файла. Наличие репликации, а по сути, избыточности данных обеспечивает отказоустойчивость кластера по отношению к отказам отдельных узлов.

Кластер HDFS состоит из узлов двух типов: головного, или *узла имен* (name node), который хранит все метаданные о файлах и репликации их блоков между узлами, а также из *узлов данных* (data node), в которых хранятся блоки файлов.

# Архитектура HDFS

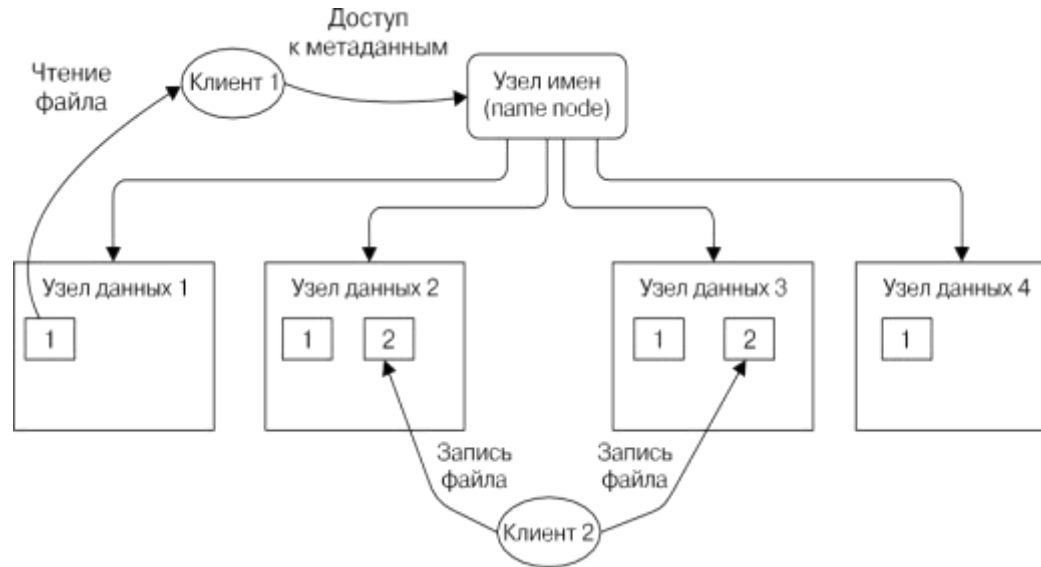


Рис. 8.2. Архитектура HDFS

- В отличие от обычных файловых систем (например, NTFS) HDFS позволяет только записывать и удалять файл, но не допускает его модификацию. Записывать файл в одно и то же время может только один процесс.
- При этом организация файлов — традиционная иерархическая с корневым каталогом и набором вложенных каталогов.
- Каждый узел, будь то узел имен или узел данных, имеет доступ извне через командную строку, а также содержит веб-сервер, который обеспечивает доступ к веб-порталу, отображающему статус узла и состав файловой системы.
- Такая файловая система лежит в основе большинства сервисов пакетного и интерактивного анализа, сервисов машинного обучения из экосистемы Hadoop, а также составляет основу хранилища типа Data Lake. Рассмотрим подробнее отдельные сервисы, реализующие концепции Data Lake.

# Azure Data Lake Store

- **Azure Data Lake Store** представляет собой масштабируемое хранилище данных для целей массивно-параллельной обработки как встроенными сервисами анализа (Azure Data Lake Analytics), так и сервисами экосистемы Hadoop, предоставляемыми сервисами HDInsight.
- Доступ из кластера HDInsight возможен с помощью REST API, совместимых со стандартом WebHDFS. В отличие от Azure Storage для Azure Data Lake Storage не указываются верхние пределы размеров и количества файлов.
- Отдельные файлы могут иметь размеры от килобайта до петабайта и сохраняются в виде реплицированных копий, разделенных по кластеру.
- Очень важное и полезное свойство этого сервиса — можно хранить файл в том же формате, который использовался при его создании: CSV, LOG и пр. В этом-то и прелесть концепции Data Lake: хранение и анализ файлов без преобразования формата, но с максимально возможной производительностью.

# Безопасность Azure Data Lake Store

Теперь рассмотрим вопросы безопасности и защиты данных в сервисе Azure Data Lake Store.

- Прежде всего доступно шифрование данных.
- Кроме того, имеется возможность интеграции этого хранилища с Azure Active Directory (AD) для обеспечения управления учетными данными и контроля доступа к ресурсам (аутентификация).
- Интеграция с AD позволяет использовать все преимущества Azure AD, а именно: многофакторную аутентификацию, контроль доступа на основе ролей (role-based access control) и мониторинг доступа к защищаемым ресурсам.
- Azure Data Lake Storage поддерживает протокол OAuth 2.0, что позволяет (по крайней мере теоретически) применять отличный от Azure AD сторонний провайдер, поддерживающий OAuth 2.0.
- Следующая опция Data Lake Store — поддержка прав доступа к файлам и каталогам на основе стандартов POSIX, доступных с помощью протокола WebHDFS.
- В качестве внутренней файловой системы в этом сервисе может быть использована Azure Data Lake Store File System, являющаяся адаптацией HDFS; в последней объекты адресуются с помощью префикса adl://.

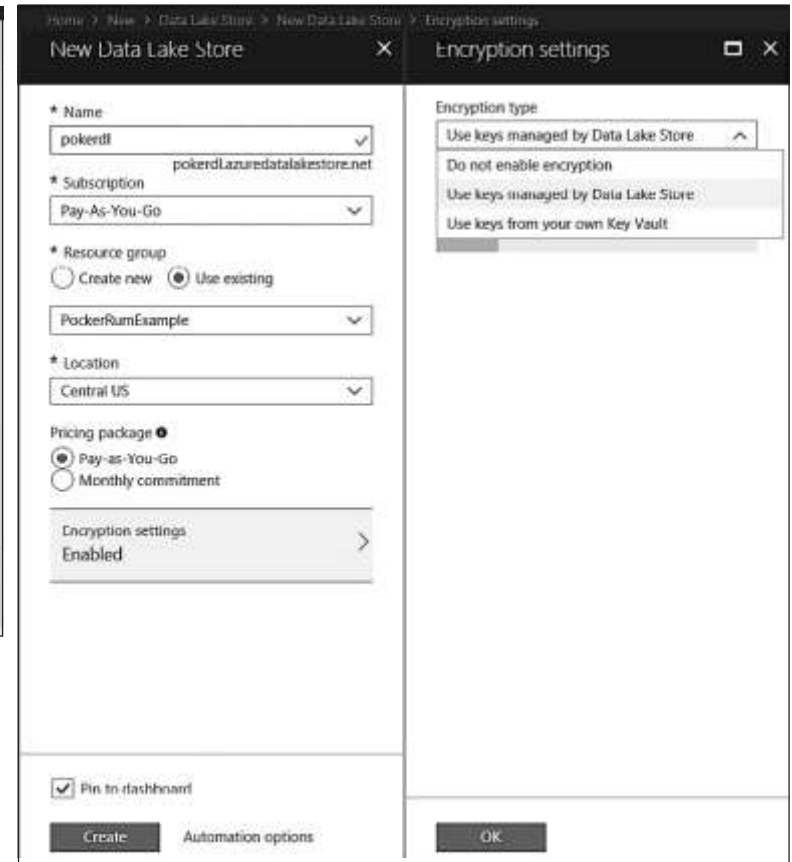
# Azure Data Lake Store

Рассмотрим на практике, как создавать экземпляр Azure Data Lake Store в веб-портале. Прежде всего в строке поиска доступных ресурсов следует написать Data Lake Store и в появившейся форме (рис. 8.3) нажать кнопку Create (Создать). Это приведет к открытию формы, показанной на рис. 8.4. Здесь нужно указать имя (Name), выбрать ресурсную группу (Resource group), местоположение (Location) и тип шифрования (Encryption). Доступны три типа шифрования: Do not enable encryption (Не шифровать данные), Use keys, managed by Data Lake Store (Использовать ключи, управляемые Azure Data Lake Store) и Use keys from your own Key Vault (Применить ключи, управляемые сервисом Key Vault и поставляемые пользователем).

Особняком стоит настройка моделей оплаты — Pricing package.



Рис. 8.3. Первый шаг создания Azure Data Lake Store



# Внешний вид общей панели сервиса Azure Data Lake Store

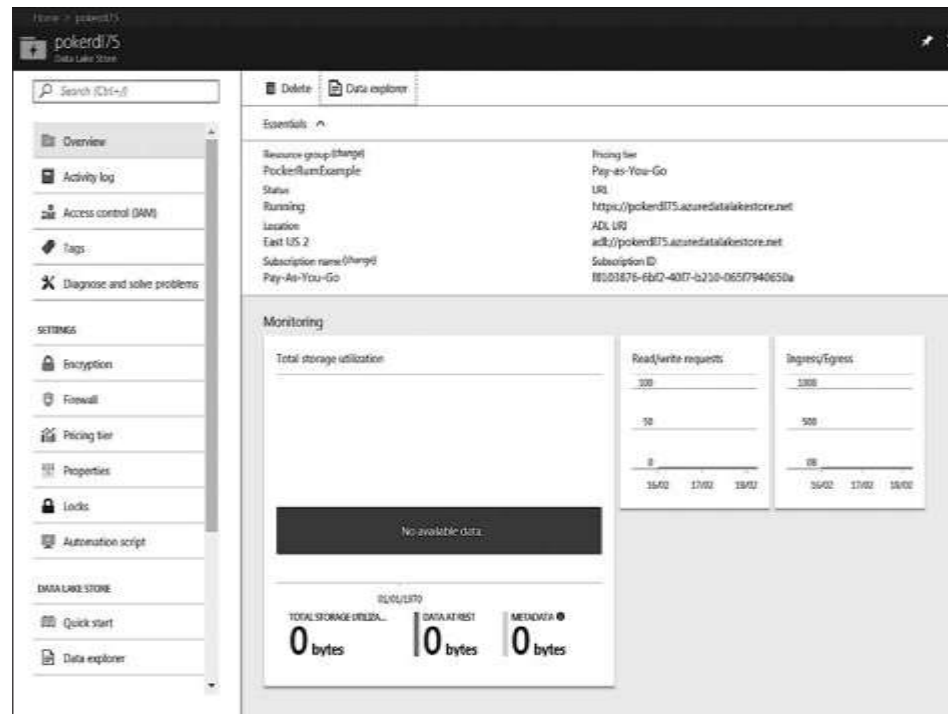


Рис. 8.5. Внешний вид общей панели сервиса Azure Data Lake Store

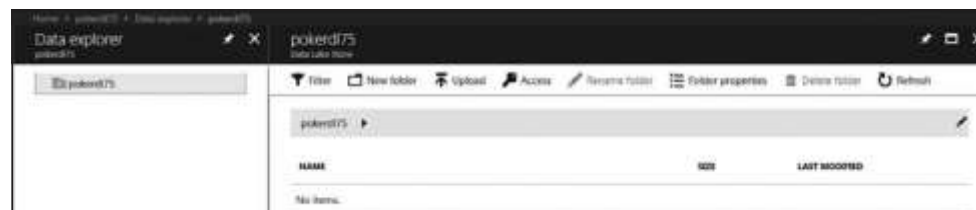


Рис. 8.6. Внешний вид панели Data explorer (Обозреватель данных)

# AWS Data Lake Solutions

В отличие от Azure, облачный провайдер AWS не содержит отдельного сервиса Data Lake (рис. 8.7). Вместо этого AWS предлагает решение, реализованное как стек ресурсов (<https://aws.amazon.com/answers/big-data/data-lake-solution/>), представленных в виде доступного для развертывания файла CloudFormation.

Прежде всего, основным хранилищем файлов в данном случае является S3 Bucket. Этот сервис обеспечивает доступность, надежность и криптографическую защищенность.

Выполнить быстрый поиск файлов поможет кластер AWS ES — сервис Elasticsearch, который предназначен для индексирования файлов в целях обеспечения высокой скорости поиска. Общие метаданные и пользовательские настройки хранятся в таблицах DocumentDB.

Помимо указанной архитектуры, можно задействовать другие сервисы: **AWS Glue** для перемещения данных Data Lake и AWS RedShift Spectrum для обеспечения аналитики среди файлов, размещенных в AWS S3 Bucket. Более подробную информацию можно получить на сайте <https://aws.amazon.com/blogs/big-data/build-a-data-lake-foundation-with-aws-glue-and-amazon-s3/>.

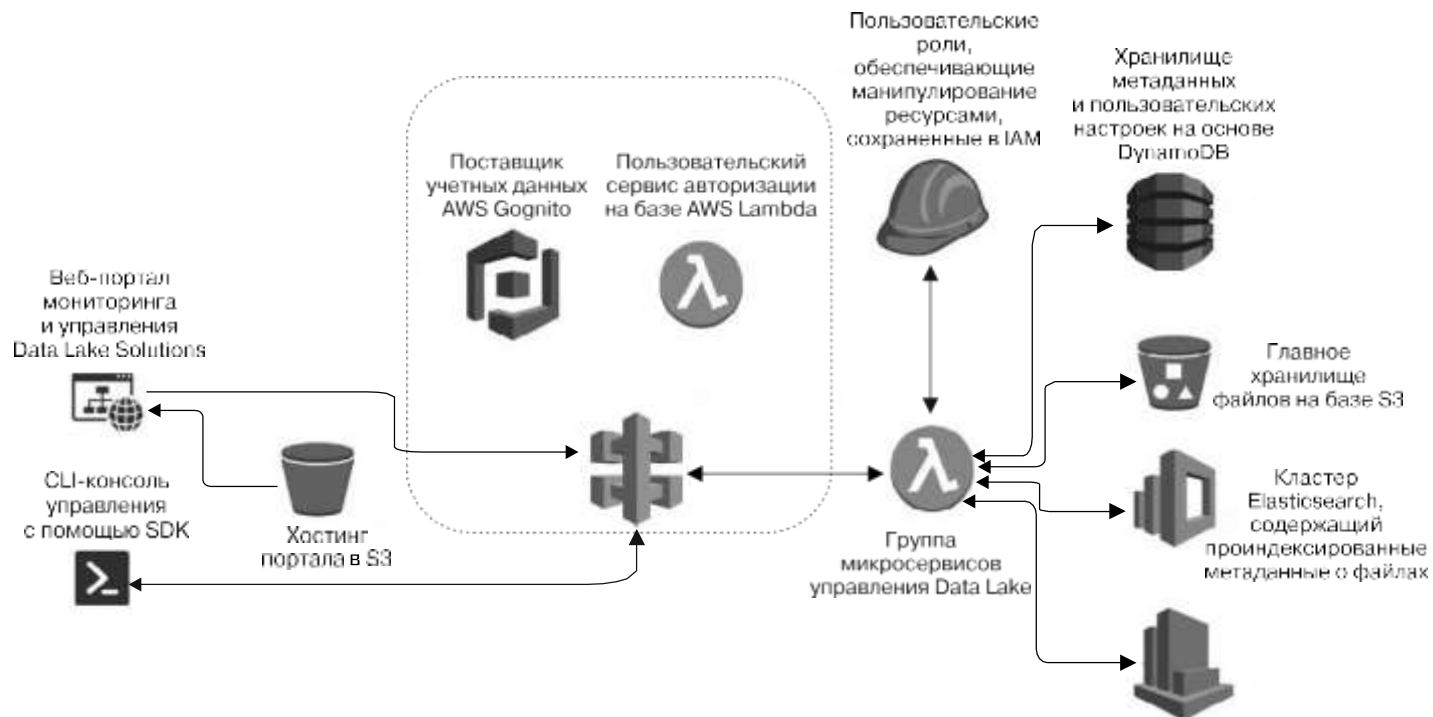


Рис. 8.7. Построение сервиса Data Lake на базе стандартных ресурсов AWS



# Модуль 2. Разработка систем аналитики больших данных

## **Доставка больших данных в облако.**

1. Прямая загрузка данных. Доставка данных в облачное хранилище общего назначения. Доставка данных в реляционные БД и хранилища. Доставка данных в нереляционные базы данных. Доставка данных в HDFS-совместимые хранилища.
2. Прямая загрузка потоковых данных. Azure Event Hub. AWS Kinesis Data Streams. Облачные сервисы развертывания кластерных систем. Облачные сервисы копирования и трансформации данных. Azure Data Factory. AWS Data Pipeline. AWS Glue.

## **Аналитика больших данных в облаке.**

1. Интерактивный анализ данных. Анализ реляционных данных. Azure Data Lake Analytics. AWS Athena. Apache Spark. Встроенные редакторы запросов сервиса CosmosDB
2. Потоковый анализ данных. Общие сведения. Azure Stream Analytics. Amazon Kinesis Analytics. Apache Storm.
3. Пакетный анализ данных. Hadoop. Apache Pig. Apache Hive.

# ЛЕКЦИЯ 15:

## Доставка BigData в облако



Источник: Intel, 2019 г.

Модель распределения приложений между различными типами облаков