Версия документа 1.0

ФГБОУ ВО «Воронежский государственный
технический университет»

Кафедра систем автоматизированного проектирования
и информационных систем

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к лабораторным работам по дисциплине «Облачные технологии» для студентов направления 09.03.02 «Информационные системы и технологии» очной формы обучения

Воронеж 2021

Составитель: Д.В. Иванов
УДК 681.38+681.3

Методические указания к лабораторным работам по дисциплине «Облачные технологии» для студентов направления 09.03.02 «Информационные системы и технологии» очной формы обучения / ФГБОУ ВО «Воронежский государственный технический университет»; сост. Д.В. Иванов. Воронеж, 2021. 132 с.

Методические указания содержат краткие теоретические и практические сведения об основных конструкциях языках SQL и их практического применения.

Методические указания подготовлены в электронном виде в текстовом редакторе MS Word 2003 и содержатся в файле ОТ 1.0.pdf.

Табл. 11. Библиогр.: 4 назв.

Рецензент д-р техн. наук, проф. К.А. Разинкин

Ответственный за выпуск зав. кафедрой
д-р техн. наук, проф. Я.Е. Львович

Издается по решению редакционно-издательского совета Воронежского государственного технического университета

Лабораторная работа 1
«Обзор облачных архитектур и платформ»

1.1 Краткие теоретические сведения

Термин облачные вычисления (cloud computing) стал использоваться на рынке ИТ с 2008 года. Разработчики Облачных вычислений определяют их как инновационную технологию. Концепция облака представляет собой эволюционный скачок в развитии ИТ-инфраструктур.

Идеология заключается в переносе организации вычислений, обработки и хранения данных с персональных компьютеров в Интернет. Существует несколько определений Облачных вычислений, привожу два из них.

облачные вычисления — вычисления, основанные на масштабированных и виртуализованных ресурсах (данных и программах), которые доступны пользователям через Интернет и реализуются на базе мощных центров обработки данных (ЦОД, date centers).

облачные вычисления — это новая парадигма, предполагающая распределенную и удаленную обработку и хранение данных, при этом программное обеспечение и ИТ-инфраструктура поставляются как услуги через Интернет. Эта сеть услуг и есть облако (раньше это слово писали в кавычках, но теперь используют уже как термин, а не как метафору). Используя мобильный телефон или персональный компьютер, конечный пользователь получает доступ к облачным ресурсам там, где это необходимо. Cloud Computing дает возможность с любого терминала получать свой знакомый и «обжитой» интерфейс.

Доказательством того, что это не временное увлечение, а новый путь развития высоких технологий, является тот факт, что три гиганта — Microsoft, Apple и Google, практически одновременно стали развивать облачные технологии и связывают с ними свое будущее.

В сущности, Облачные вычисления уже сейчас используются каждым интернет-пользователем — электронная почта, фотографии или видео, размещенные в социальной сети. Однако это совсем ничтожная доля возможностей Облачных вычислений.

Облачные вычисления являются последним словом в мире ИТ и открывают новую эру оперативности и эффективности предоставления ИТ-сервисов, ресурсы выделяются в считанные минуты, эксплуатируются и возвращаются назад в пул ресурсов.

1.1.1 Технологии облачных вычислений

Облачные вычисления — это одновременно и способ обеспечения работы пользователей, и бизнес-модель.

К настоящему времени можно выделить несколько основных технологий (моделей) этого направления:

• инфраструктура как услуга (Infrastructure as a Service, iaas);

• платформа как услуга (Platform as a Service, Paas);

• данные как услуга (Data as a Service, Daas);

• программное обеспечение как услуга (Software as a Service, saas);

• рабочее место как услуга (Workplace as a Service, Waas);

• все как услуга (All as a Service, Aaas).

Рассмотрим три основных технологии подробнее.

1. iaas — Ит-инфраструктура в качестве сервиса. IaaS состоит из трех основных компонентов:

• аппаратные средства (серверы, системы хранения данных, клиентские системы, сетевое оборудование);

• операционные системы и системное ПО (средства виртуализации, автоматизации, основные средства управления ресурсами);

• связующее ПО (например, для управления системами).

Обычно такой сервис оплачивается исходя из обслуживания вычислительной базы и суммы использованных ресурсов. IaaS избавляет предприятия от необходимости поддержки сложных инфраструктур центров обработки данных, клиентских и сетевых инфраструктур, а также позволяет уменьшить связанные с этим капитальные затраты и текущие расходы. Кроме того, можно получить дополнительную экономию при предоставлении услуги в рамках инфраструктуры совместного использования.

Выгоды от использования Облачных вычислений получаются вполне практические — сокращение расходов, повышение продуктивности, удобство для пользователей. Этими преимуществами уже пользуются компании по всему миру. В России тоже наблюдается постепенный рост уровня зрелости заказчиков и поставщиков Облачных вычислений.

Разумеется, новой идеологии потребления софта необходимо завоевать доверие аудитории.

В модели SaaS:

• приложение приспособлено для удаленного использования;

• одним приложением могут пользоваться несколько клиентов;

• оплата за услугу взимается либо как ежемесячная абонентская плата, либо на основе суммарного объема транзакций;

• поддержка приложения входит уже в состав оплаты;

• модернизация приложения может производиться обслуживающим персоналом плавно и прозрачно для клиентов.

С точки зрения разработчиков программного обеспечения, модель SaaS позволит эффективно бороться с нелицензионным использованием программного обеспечения, благодаря тому, что клиент не может хранить, копировать и устанавливать программное обеспечение.

По сути, программное обеспечение в рамках SaaS можно рассматривать в качестве более удобной и выгодной альтернативы внутренним информационным системам.

Развитием логики SaaS является концепция WaaS (Workplaceas a Service — рабочее место как услуга). То есть клиент получает в свое распоряжение полностью оснащенное всем необходимым для работы ПО виртуальное рабочее место.

По недавно опубликованным данным SoftCloud, спросом пользуются следующие SaaS приложения (в порядке убывания популярности):

• почта;

• коммуникации (VoIP);

• антиспам и антивирус;

• helpdesk;

• управление проектами;

• дистанционное обучение;

• CRM;

• хранение и резервирование данных.

Все три типа облачных сервисов взаимосвязаны и представляют вложенную структуру.

Помимо различных способов предоставления сервисов различают несколько вариантов развертывания облачных систем: публичное, частное и гибридное облака (Public Cloud/Private Cloud/Hybrid Cloud).

публичные облака — в данной модели облачная инфраструктура предоставляется для использования всем желающим. Система создана одним из глобальных провайдеров и услуги продаются через Интернет. Такие облака также называются «внешними».

частные облака — это реализация модели облачных вычислений на ресурсах, имеющихся в распоряжении у вашей компании, для обслуживания внутренних потребителей. При этом вычислительные ресурсы: серверы, сети, устройства хранения, базовая программная инфраструктура интегрируются в частное облако с помощью специализированного программного обеспечения, которое позволяет реализовать функциональные атрибуты облака: сервисную модель, автоматизированное обслуживание потребителей, масштабируемость и т.д. Сервисная модель частного облака зависит от того, кто является внутренними потребителями облака, и какие именно ИТ-сервисы им требуются. В зависимости от структуры предприятия такими внутренними потребителями могут стать различные подразделения компании:

• дочерние компании в холдинге, получающие приложения (корпоративную почту, внутренние порталы и т.д.) из ЦОД, обслуживаемого сервисным подразделением;

• отделы департамента ИТ, отвечающие за бизнес-приложения, получающие инфраструктуру как сервис из облака, обслуживаемого инфраструктурной группой. Такие облака также называются «внутренними».

гибридные облака — возникают вследствие интеграции внешнего и внутреннего методов доставки услуг. Организация устанавливает правила и политику на основании такого фактора, как безопасность. Гибридные (смешанные) облака позволяют совместить использование виртуализованных и физически фиксированных мощностей. В данном случае первые дополняют вторые, обеспечивая адекватную производительность при повышении нагрузок и экономию при снижении потребностей в вычислительных мощностях.

Таким образом, эти технологии при совместном использовании позволяют пользователям Облачных вычислений воспользоваться вычислительными мощностями и хранилищами данных, которые предоставляются им как услуги.

Типы ИТ-сервисов могут быть разными: от готовых приложений до инфраструктуры. В любом случае при использовании облака открываются новые возможности внутри компании:

• за счет автоматизации накладные расходы на оказание ИТ-сервиса резко падают, снижается время ожидания предоставления ресурсов;

• облако способствует эффективному распределению ресурсов внутри организации, динамически перераспределяя нагрузку между физическими системами центра обработки данных;

• появляется возможность отслеживать реальное потребление ИТ-ресурсов внутри компании и распределять затраты на поддержку и расширение базовой инфраструктуры между потребителями на основании бизнес-ценности.

основополагающими принципами организации Облачных вычислений являются:

• виртуализация ИТ-инфраструктуры;

• стандартизация предоставляемых ИТ-услуг;

• автоматизация предоставления этих услуг (порталы самообслуживания пользователей).

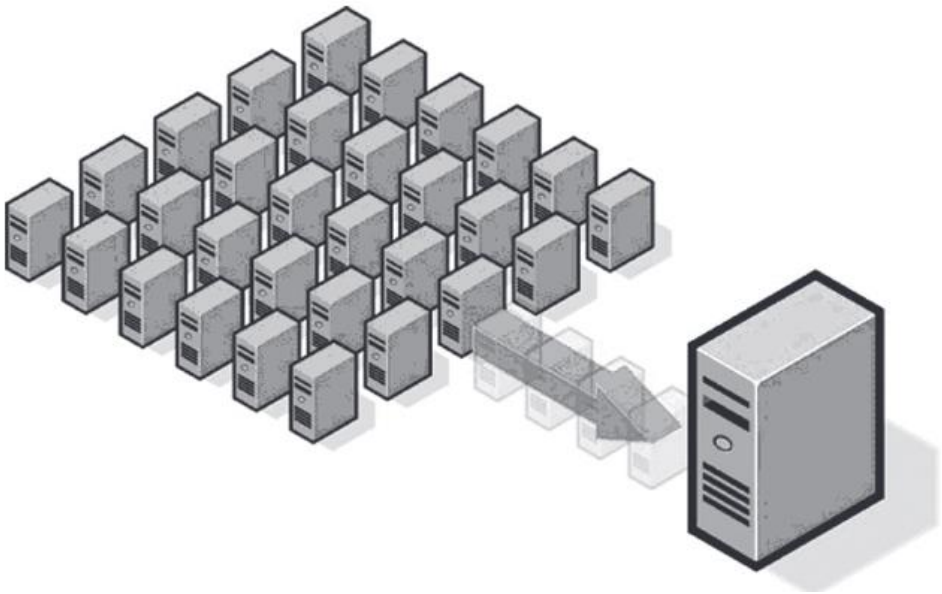


Рисунок 1. Виртуализация подразумевает запуск на одном физическом компьютере нескольких виртуальных компьютеров

Виртуализация — это технология, которая создает уровень абстракции между аппаратной частью компьютеров и выполняемыми на них программами. С помощью такого абстрактного уровня можно поделить один физический компьютер на несколько логических и запустить на последних несколько операционных систем одновременно (рис. 1).

В основе виртуализации лежит возможность одного компьютера выполнять работу нескольких компьютеров благодаря распределению его ресурсов по нескольким средам. С помощью виртуальных серверов и виртуальных настольных компьютеров можно разместить несколько ОС и несколько приложений в едином местоположении. Таким образом, физические и географические ограничения перестают иметь какое-либо значение. Помимо энергосбережения и сокращения расходов, благодаря более эффективному использованию аппаратных ресурсов виртуальная инфраструктура обеспечивает высокий уровень доступности ресурсов, более эффективную систему управления, повышенную безопасность и усовершенствованную систему восстановления в критических ситуациях.

В широком смысле понятие виртуализации представляет собой сокрытие настоящей реализации какого-либо процесса или объекта от истинного его представления для того, кто им пользуется. Продуктом виртуализации является нечто удобное для использования, на самом деле, имеющее более сложную или совсем иную структуру, отличную от той, которая воспринимается при работе с объектом. Иными словами, происходит отделение представления от реализации чего-либо. Виртуализация призвана абстрагировать программное обеспечение от аппаратной части.

В компьютерных технологиях под термином «виртуализация» обычно понимается абстракция вычислительных ресурсов и предоставление пользователю системы, которая «инкапсулирует» (скрывает в себе) собственную реализацию.

Проще говоря, пользователь работает с удобным для себя представлением объекта, и для него не имеет значения, как объект устроен в действительности.

Виртуализация — процесс предоставления набора вычислительных ресурсов или их логического объединения, который дает какие-либо преимущества перед оригинальной конфигурацией.

Виртуальная машина — программная или аппаратная среда, которая скрывает настоящую реализацию какого-либо процесса или объекта от его видимого представления.

Сейчас возможность запуска нескольких виртуальных машин на одной физической вызывает большой интерес среди компьютерных специалистов не только потому, что это повышает гибкость ИТ-инфраструктуры, но и потому, что виртуализация, на самом деле, позволяет экономить деньги.

основными характеристиками облачных вычислений являются:

1. масштабируемость.

Способность информационных систем выдерживать растущие нагрузки и обрабатывать большие объемы данных. Масштабируемое приложение позволяет выдерживать большую нагрузку за счет увеличения количества одновременно запущенных экземпляров. Как правило, для одновременного запуска множества экземпляров используется типовое оборудование, что снижает общую стоимость владения и упрощает сопровождение инфраструктуры.

2. Эластичность.

Гибкая реакция на изменяющиеся условия ведения бизнеса является одной из характеристик успешного бизнеса. Эластичность позволяет быстро нарастить мощность инфраструктуры без необходимости проведения начальных инвестиций в оборудование и программное обеспечение. Эластичность связана с масштабируемостью приложений, так как решает задачу моментального изменения количества вычислительных ресурсов, выделяемых для работы информационной системы.

3. мультитенантность.

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

4. оплата за использование.

Оплата использованных ресурсов позволяет перевести часть капитальных издержек в операционные. Приобретая только необходимый объем ресурсов, можно оптимизировать расходы, связанные с работой информационных систем организации. А в сочетании с мультитенантностью, разделяя ресурсы между различными потребителями, можно снизить расходы еще больше. Эластичность позволит быстро изменить объем ресурсов в сторону увеличения или уменьшения, тем самым, приведя расходы на ИТ в соответствие с фактическими потребностям организации.

5. самообслуживание.

Быстрый вывод на рынок нового продукта или услуги в современных условиях сопровождается развертыванием или модификацией информационных систем. Традиционно развертывание информационной системы может занять длительное

время: месяцы и даже годы. Самообслуживание позволяет потребителям запросить и получить требуемые ресурсы за считанные минуты.

Только сочетание нескольких атрибутов Облачных вычислений приводит к достижению задачи повышения доходов и снижения расходов.

Необходимо также понимать, что переход в облако не является тривиальной задачей и часто требует пересмотра и изменения архитектуры существующих решений, а иногда — полного отказа от них в пользу создания новых, реализованных с учетом возможностей, предоставляемых облачными платформами.

1.1.2 Общее понятие об облачных вычислениях

Облачные вычисления (*cloud computing*) являются одним из наиболее популярных направлений развития ИТ. Понятие облака (cloud) уже давно ассоциируется с метафорическим изображением Интернета, с помощью которого доступны некоторые сервисы. **Облачные вычисления (cloud computing)** – это практическая реализация данной идеи. Облачные вычисления основаны на масштабированных и виртуализованных ресурсах (данных и программах), которые доступны пользователям через Интернет и реализуются на базе мощных **центров обработки данных (data centers).**

Общая структура "облака" изображена на рис. 2.1.

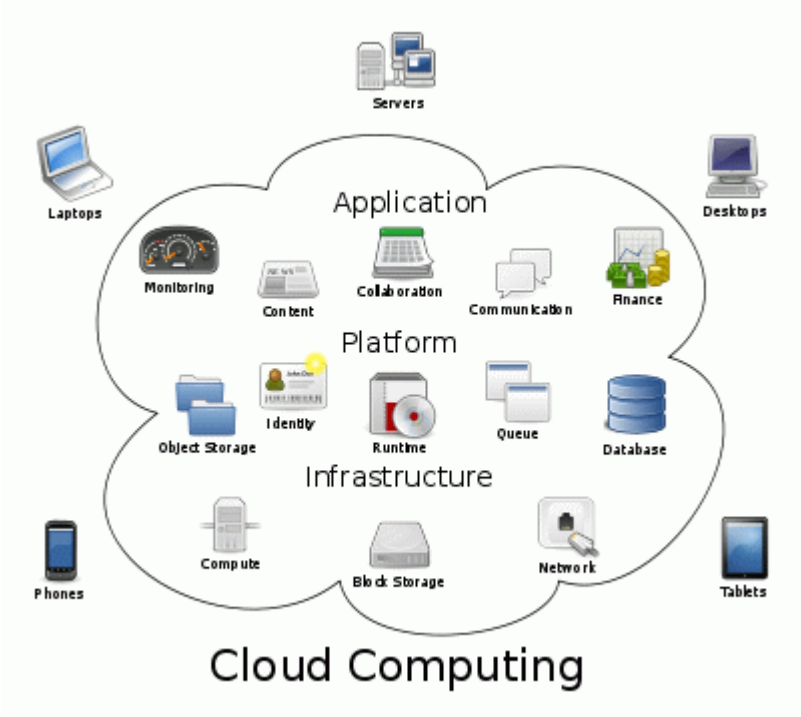


Рис. 2.1. Архитектура облачных вычислений

С точки зрения пользователей, существует совокупность "облаков" (общедоступные, корпоративных, частных и др.), предоставляемых различными компаниями, для использования мощных вычислительных ресурсов, которых нет у индивидуального пользователя. Как правило, "облачные" сервисы платные. Из бесплатных назовем Windows Live (http://www.live.com).

Виды "облаков" в облачных вычислениях иллюстрируются рис. 2.2.

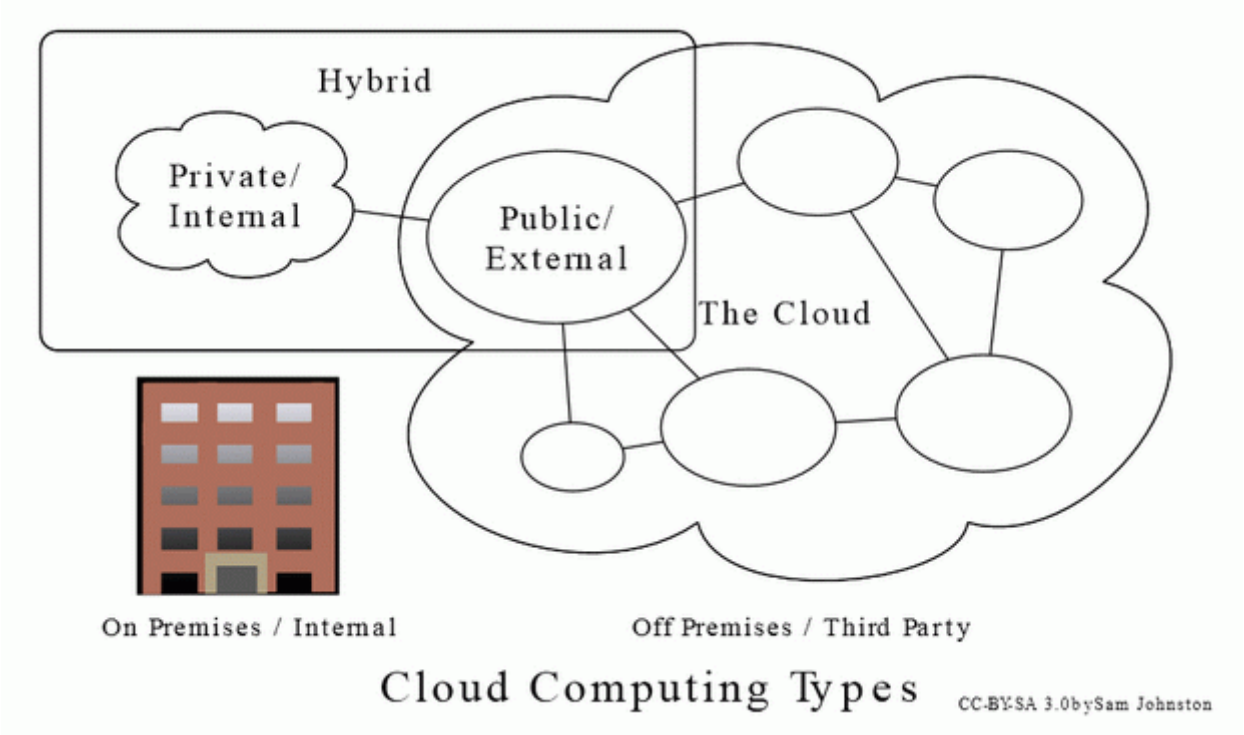


Рис. 2.2. Виды "облаков" в облачных вычислениях

Недостаток облачных вычислений в том, что пользователь оказывается полностью зависимым от используемого им "облака" (в котором доступны используемые им данные и программы) и не может управлять не только работой "облачных" компьютеров, но даже резервным копированием своих данных. В связи с этим возникает целый ряд важных вопросов о безопасности облачных вычислений, сохранении конфиденциальности пользовательских данных и т.д.; далеко не все из них на данный момент решены.

Серьезной проблемой организации облачных вычислений с точки зрения аппаратуры центров обработки данных является экономия электроэнергии и проблема распределения загрузки, так как облачные вычисления в каждом центре обработки данных имеют (или в ближайшем будущем будут иметь) *миллионы* удаленных пользователей. В настоящее время целый ряд крупных, в том числе –

правительственных и коммерческих организаций США закрывают свои центры обработки данных (ЦОД), в связи со слишком большими энергозатратами. В самом деле, ЦОД может занимать одно или несколько огромных зданий.

Наиболее популярная "облачная" платформа – Microsoft Windows Azure (облачная ОС) и Microsoft Azure Services Platform (реализованная на основе Microsoft.NET). Windows Azure можно рассматривать как "ОС в облаке". Пользователю нет необходимости беспокоиться о ее инсталляции на его компьютере, который может не иметь для этого необходимых ресурсов. Все, что требуется, это иметь Web-браузер и минимальный пакет надстроек (plug-ins) для запуска и использования через браузер облачных сервисов.

В настоящее время многие крупные компании – Microsoft, Google. IBM, Oracle, Amazon и многие более мелкие фирмы, конкурируя друг с другом, заняты разработкой своих облачных сервисов и инструментов для их создания. Имеется тенденция к интеграции "корпоративных облаков" в единое доступное пользователю облако. Из наиболее популярных платформ облачных вычислений назовем Amazon EC2.

1.1.3. Элементы концепции облачных вычислений

Элементами концепции облачных вычислений являются: *инфраструктура как сервис, платформа как сервис, программное обеспечение как сервис*, а также бизнес-приложения доступные через Интернет. Иными словами, организация облачных вычислений коренным образом меняет архитектуру системы: в ней необходимо представить все возможности обработки данных, использования программ настройки и т.д. как облачные сервисы.

1.1.4. Уровни компонент облачных вычислений

Различаются следующие уровни архитектуры облачных вычислений.

Уровень клиента – это клиентское ПО, используемое для доступа к облачным сервисам, например, web- браузер.

Уровень сервисов – это сами сервисы, используемые через облачную модель.

Уровень приложений – это программы, доступные через облако и не требующие инсталляции на компьютере пользователя (в последнем – одно из главных преимуществ облачной модели).

Уровень платформы – это программная платформа, объединяющая полный набор инструментов для развертывания и использования облачных вычислений на

пользовательском компьютере (без дополнительных инсталляций, покупки оборудования и др.). Пример такой платформы: **Microsoft.NET Azure Services Platform**.

Уровень памяти – поддержка хранения данных пользователя и доступа к ним через облако.

Уровень инфраструктуры – предоставление полной виртуализованной платформы через облако, например, Amazon EC2.

Пример организации облачных вычислений с использованием различных уровней приведен на рис. 2.3.

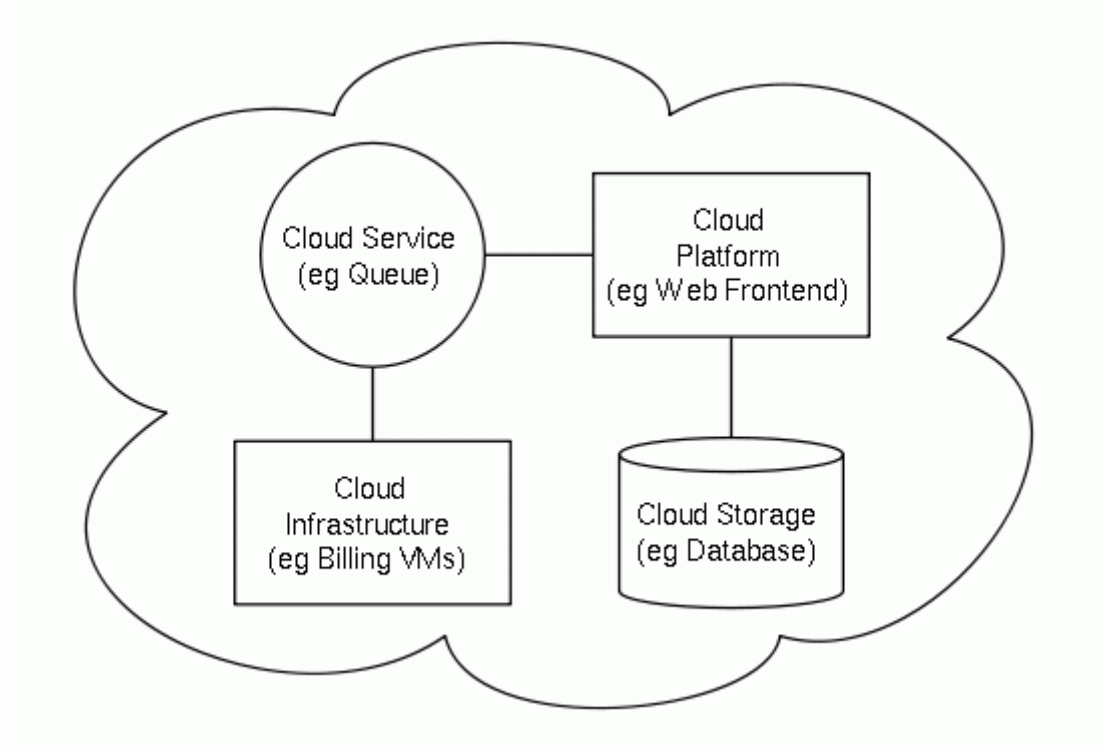


Рис. 2.3. Пример организации облачных вычислений с использованием различных уровней

1.1.5. Архитектура облачных вычислений

Рассмотрим схему архитектуры облачных вычислений:

- **Сервисы**, доступные через облако
- **Инфраструктура** для их развертывания и использования
- **Платформа** – набор инструментов для использования облака
- **Память** – поддержка хранения пользовательских данных в ЦОД, реализующем облако
- **Архитектор облака** – это главный разработчик его архитектуры.
- **Интегратор облак***а* – это его системный администратор, отвечающий за добавление компонент в облако и их изменение.
- Компоненты облака, как правило, являются Web-сервисами.

Облако может быть общедоступным или частным (корпоративным).

Общая архитектура облачных вычислений проиллюстрирована на рис. 2.4.

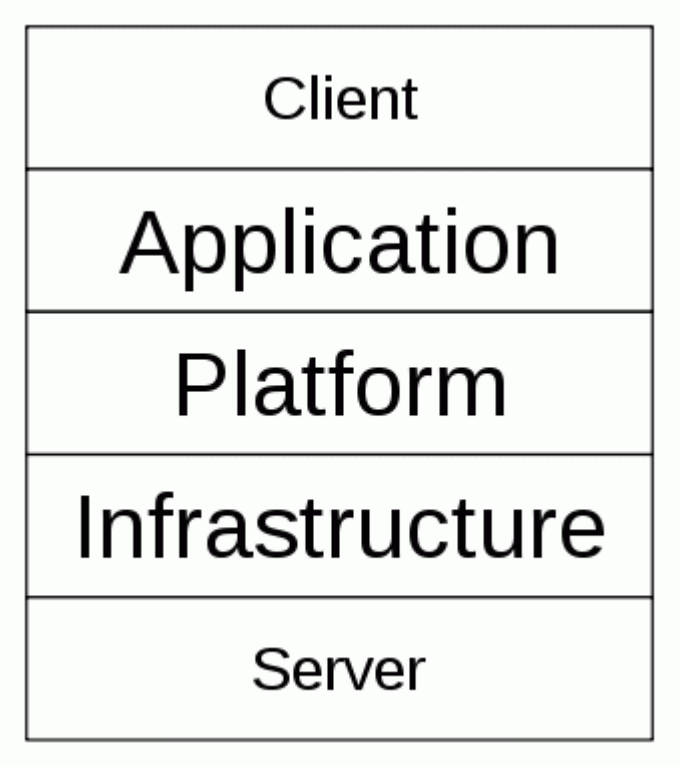


Рис. 2.4. Общая архитектура облачных вычислений

1.1.6. Роли в облачных вычислениях

При использовании облачных вычислений несколько изменяются и роли участвующих в них специалистов.

Поставщиком облака является центр обработки данных.

Пользователями облака могут быть любые пользователи Интернета.

Производитель оборудования или ПО для облака – это компания, обеспечивающая разработку аппаратуры и базового программного обеспечения для центра обработки данных.

1.1.7. Стандарты облачных вычислений

Модель облачных вычислений основана на соблюдении целого ряда *стандартов*.

Для взаимодействия приложений используются стандарты:

- **HTTP** (основной Web-протокол);
- **XMPP (Jabber)** – стандарт для отправки и получения мгновенных сообщений в формате XML; слово *jabber* буквально означает "болтовня"; интересно, что XML-"фразы" в этом протоколе называются на поэтический манер *стансы (stanza)*;

• **SSL (Secure Socket Layer)** – уровень безопасных сокетных сетевых соединений, используемый, например, в протоколе **https**.

Для работы клиентов в облаке используются Web-браузеры (с активным использованием технологии **AJAX** *(***Asynchronous JavaScript and XML)**, позволяющей уменьшить число перенаправлений с одной веб-страницы на другую и, тем самым, время доступа пользователя к необходимой ему информации) и offline-клиенты, работа которых основана на *HTML 5* (специальной версии HTML для облачных вычислений).

Для реализации облака используются принципы виртуализации программ и данных и стандарт *OMF*.

Для взаимодействия с сервисами данные передаются в формате *XML*.

1.1.8. Обзор платформ облачных вычислений

Amazon Elastic Compute Cloud, или EC2, по-видимому, является наиболее ранней, наиболее общей и наиболее известной из облачных серисных платформ.

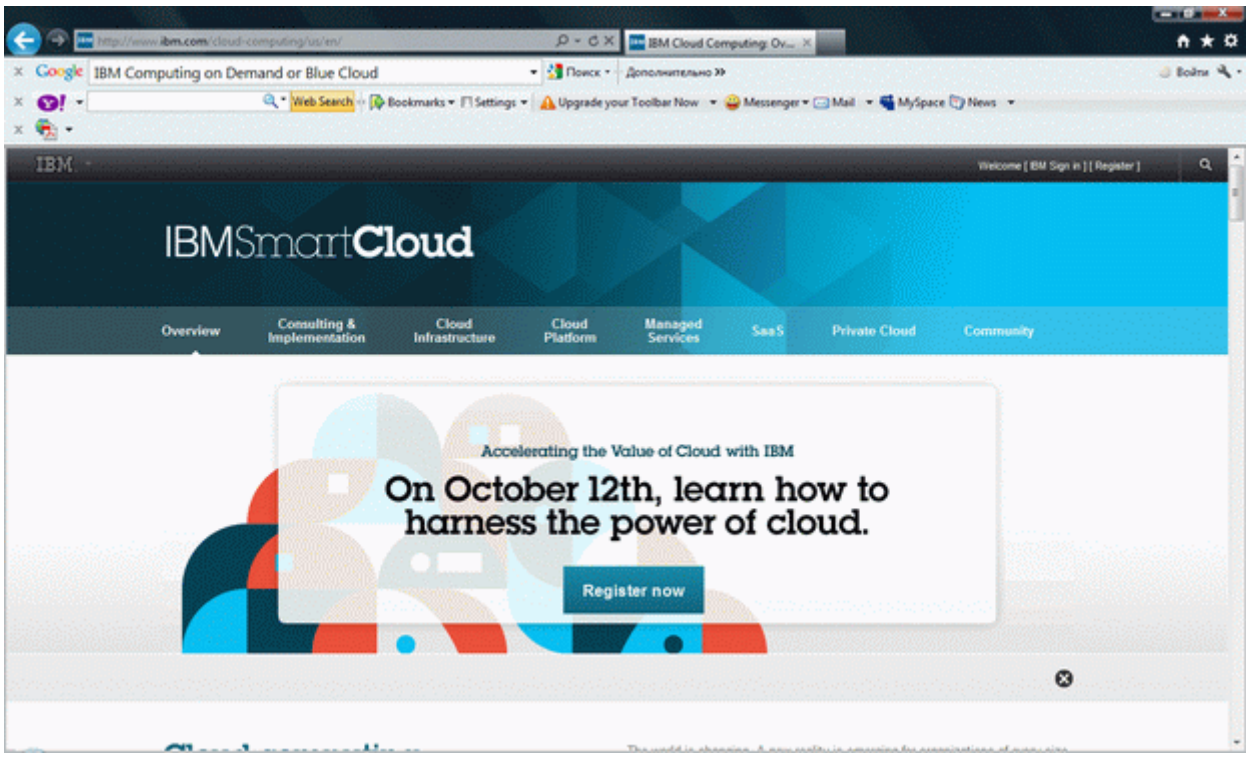
Страница облака Amazon изображена на рис. 2.5. Там же приведен ее URL-адрес.

увеличить изображение

Рис. 2.5. Облако Amazon EC2

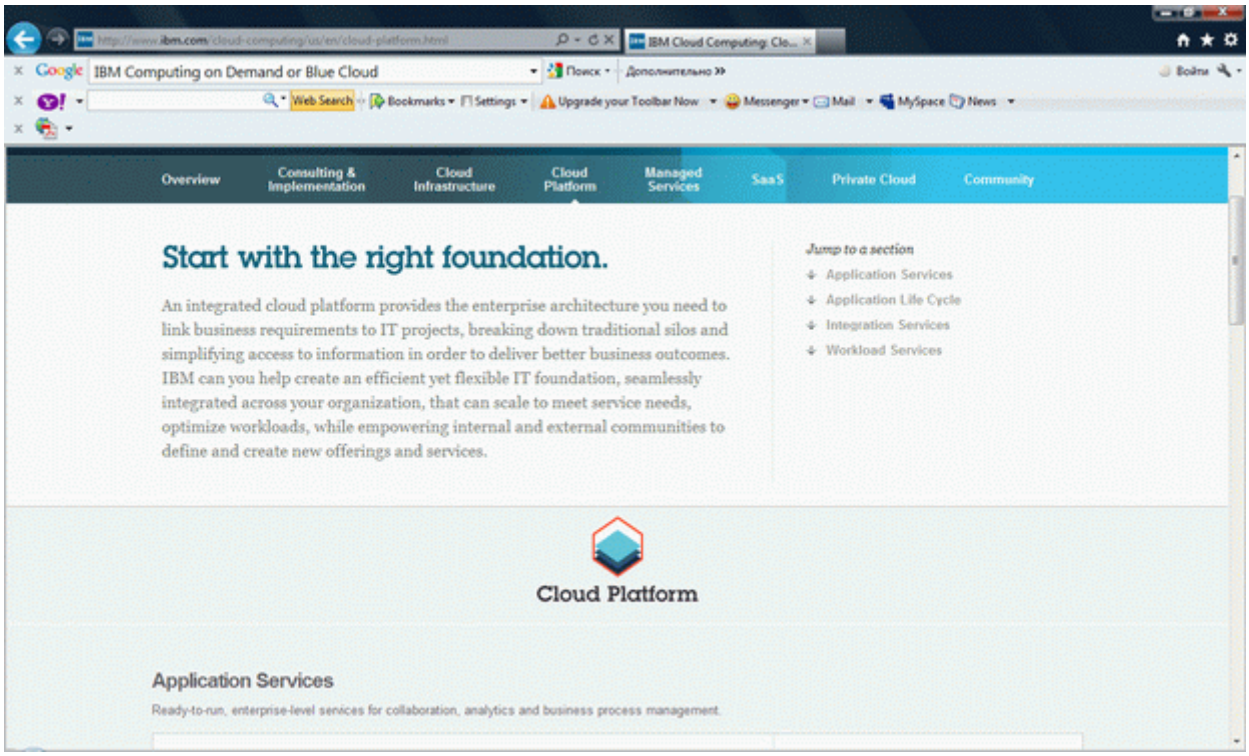
IBM Smart Cloud – облачная платформа, ориентированная на уровень предприятия. Ее облачные сервисы могут предоставляться и как элементы общедоступного облака, и как компоненты приватного облака.

На рис. 2.6, рис. 2.7 и рис. 2.8 приведены начальные страницы для входа в облако IBM и его использования.



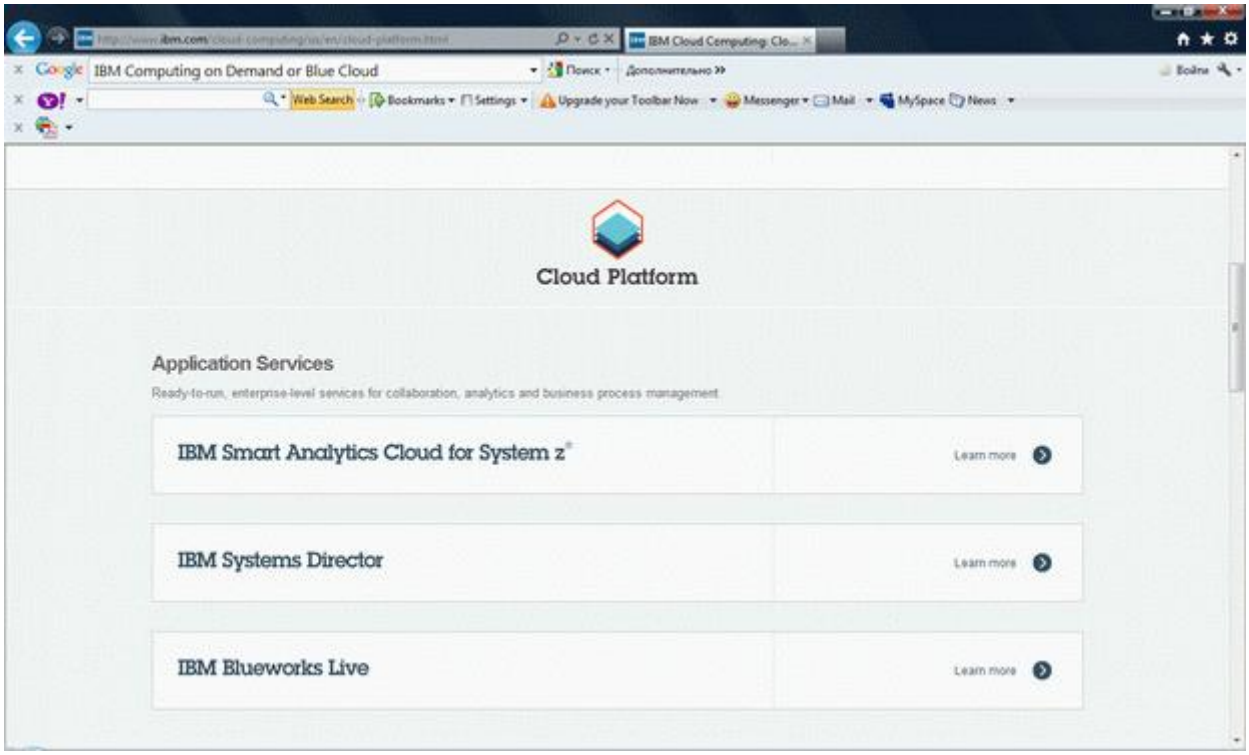
увеличить изображение

Рис. 2.6. Начальная страница IBM Smart Cloud



увеличить изображение

Рис. 2.7. Страница IBM с информацией об облачных вычислениях

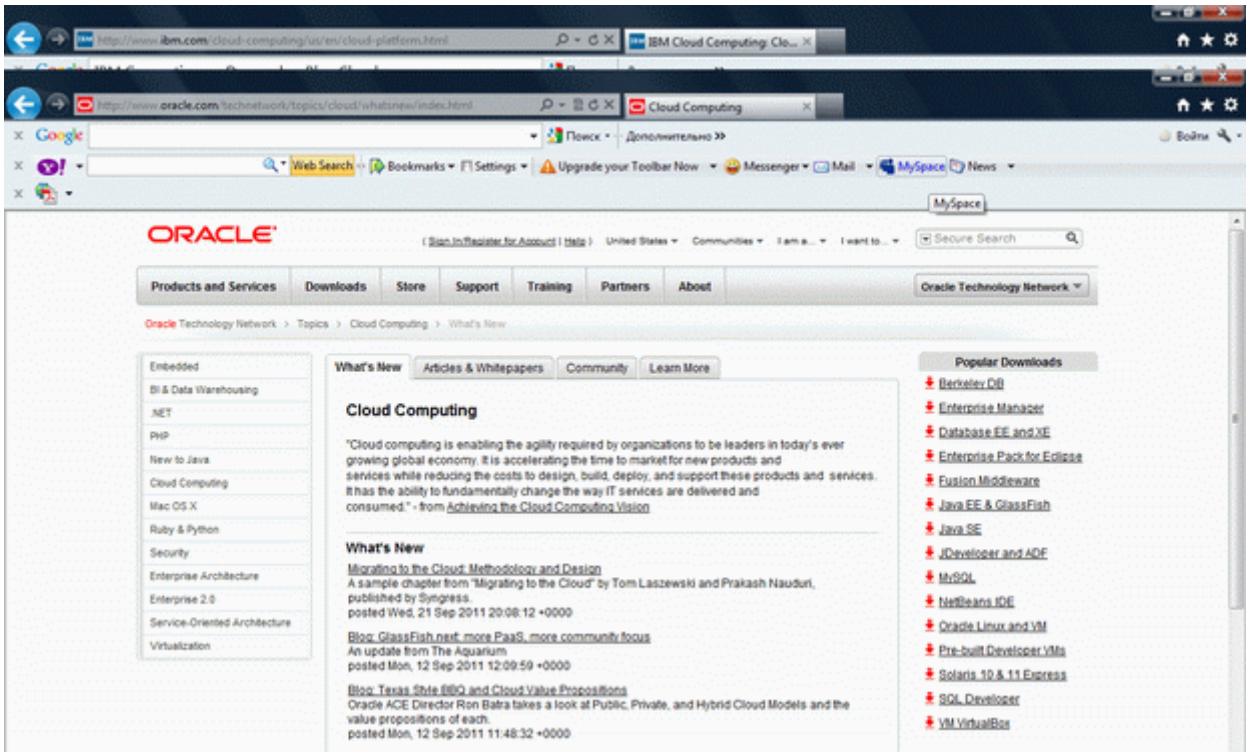


увеличить изображение

Рис. 2.8. Страница IBM с перечнем некоторых enterprise-сервисов, предоставляемых облаком IBM

Microsoft Windows Azure также поддерживает как публичные, так и приватные облачные сервисы. Она основана на архитектуре .NET и подробно рассматривается в данном курсе. Весьма важно, что появились программные инструменты для связи платформы Java с платформой Azure, позволяющие работать с облачными сервисами Azure с использованием Java API. Обо всем этом – речь ниже в данном курсе.

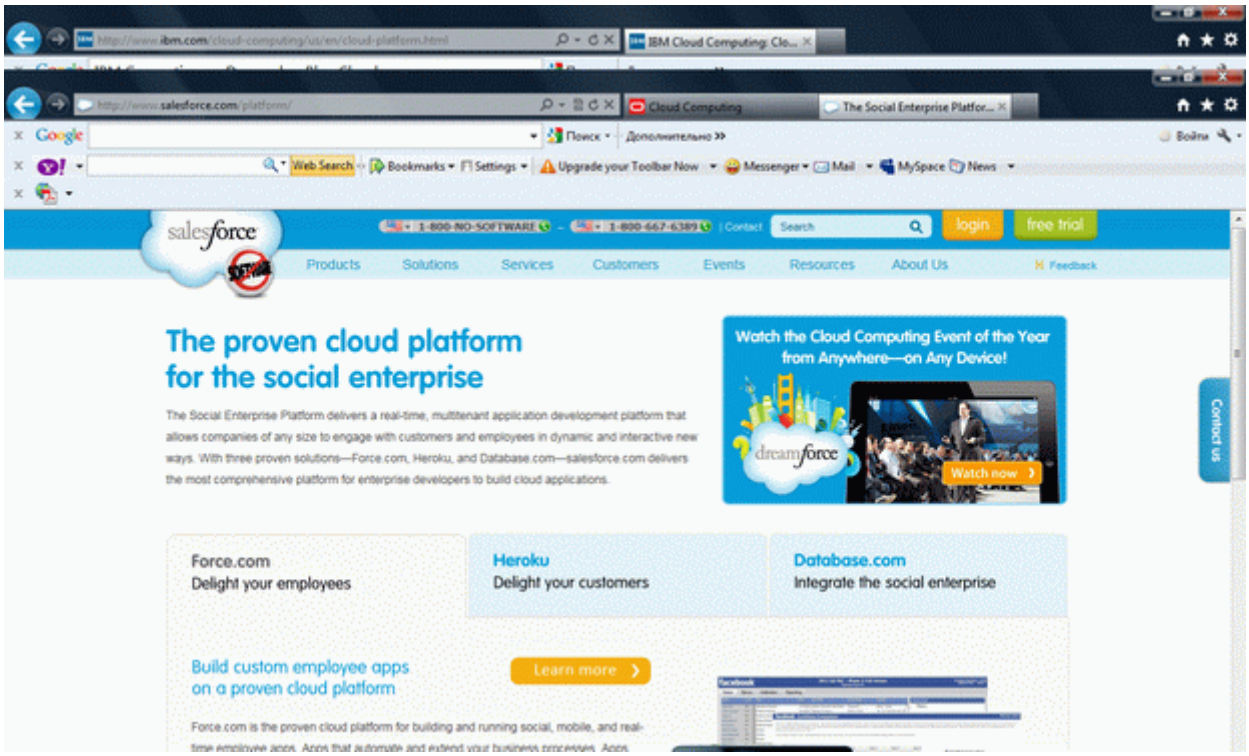
Oracle Cloud – аналогично IBM cloud, доступно как в публичной, так и в приватной форме. Информация об облачных вычислениях фирмы Oracle приведена на странице, изображенной на рис. 2.9.



увеличить изображение

Рис. 2.9. Страница с информацией об облачных вычислениях фирмы Oracle

Salesforce.com: Force.com cloud – легко интегрируется с программными инструментами, опубликованными на известном сайте Salesforce.com. Начальная страница с информацией об облаке Force.com приведена на рис. 2.10.



увеличить изображение

Рис. 2.10. Страница с информацией об облаке Force.com cloud

Google's AppEngine – набор облачных сервисов, ориентированный на веб-разработчиков и приложений для веб-хостинга. Типичный пример – надстройка Google для браузеров, обеспечивающая поиск с помощью поисковой машины Google.

Имеется и ряд других менее известных облачных платформ, например, **Kaavo cloud**.

1.1.9. Ключевые термины

Microsoft Azure Services Platform - платформа фирмы Microsoft для разработки и использования облачных сервисов на базе Microsoft.NET.

Microsoft Windows Azure - операционная система и набор инструментов фирмы Microsoft, обеспечивающий поддержку облачных вычислений ("ОС в облаке").

SQL Azure – версия СУБД Microsoft SQL Server для использования "в облаке".

Агент интерфейса (fabric agent) – агентское приложение, исполняемое на каждом из компьютеров сервиса **Интерфейс (Fabric)** платформы **Windows Azure**.

Архитектор облака – главный разработчик его архитектуры.

Внутренние приложения (on-premises applications) – приложения, исполняемые на локальном компьютере пользователя.

Вычисления (Compute) – облачный сервис платформы Microsoft Windows Azure, исполняющий пользовательские приложения в едином облаке.

Интегратор облака – его системный администратор, отвечающий за добавление компонент в облако и их изменение.

Облачные вычисления (cloud computing) –. вычисления, основанные на масштабированных и виртуализованных ресурсах (данных и программах), которые доступны пользователям через Интернет и реализуются на базе мощных **центров обработки данных (data centers).**

Облачные приложения (cloud applications) – приложения, фактически исполняемые в среде облачных вычислений (например, Windows Azure) на компьютерах центра обработки данных.

Платформа – набор инструментов для использования облака.

Поставщик облака - центр обработки данных, поддерживающий облачные вычисления.

Уровень инфраструктуры – предоставление полной виртуализованной платформы через облако, например, Amazon EC2.

Уровень клиента – клиентское ПО, используемое для доступа к облачным сервисам, например, web- браузер.

Уровень памяти – поддержка хранения данных пользователя и доступа к ним через облако.

Уровень платформы –программная платформа, объединяющая полный набор инструментов для развертывания и использования облачных вычислений на пользовательском компьютере (без дополнительных инсталляций, покупки оборудования и др.); пример: **Microsoft.NET Azure Services Platform**.

Уровень приложений – программы, доступные через облако и не требующие инсталляции на компьютере пользователя (в последнем – одно из главных преимуществ облачной модели).

Уровень сервисов – облачные сервисы, используемые через облачную модель.

Центр обработки данных (ЦОД, data center) – мощный вычислительный центр, состоящий из компьютеров, объединенных в локальную сеть, обслуживающих сервисы облачных вычислений некоторой компании.

1.1.10. Краткие итоги

Облачные вычисления – популярная современная модель вычислений, основанная на динамически масштабируемых и виртуализованных ресурсах (данных и приложениях), которые доступны и используются как сервисы, исполняемые на компьютерах мощного центра обработки данных.

Преимущество облачных вычислений: все вычисления выполняются удаленно, от компьютера пользователя требуется только наличие веб-браузера и доступа в Интернет.

Недостаток облачных вычислений – полная зависимость пользователя от облака (в котором хранятся не только программы, но и его данные).

Современная тенденция – разработка корпоративных облаков всех ведущих фирм, их объединение в единое облако и все более широкое использование облачных вычислений пользователями.

Элементы концепции облачных вычислений: инфраструктура как сервис, платформа как сервис, программное обеспечение как сервис.

Уровни компонент облачных вычислений: уровень клиента, уровень приложений, уровень сервисов, уровень платформы, уровень памяти (данных), уровень инфраструктуры.

Архитектура облачных вычислений: сервисы, инфраструктура, платформа, память.

Роли в разработчиков и клиентов в облачных вычислениях: архитектор облака, интегратор облака, поставщик облака, пользователи облака, производитель оборудования.

Стандарты, используемые в облачных вычислениях, - коммуникация приложений на основе протоколов HTTP и XMPP (протокол обмена мгновенными сообщениями); HTML 5 – специальная версия HTML для облачных вычислений; AJAX – технология для оптимизации обращений к веб-страницам путем минимизации числа перенаправлений; OMF – стандарт виртуализации данных; передача данных в формате XML.

1.2 Задания к лабораторной работе

Разработать концептуальную модель облачных сервисов корпоративной системы в соответствии с заданной предметной областью.

Лабораторная работа 2
«Общие сведения об облачных хранилищах»

2.1 Краткие теоретические сведения

2.1.1 Docker

Что такое докер?

Докер — это открытая платформа для разработки, доставки и эксплуатации приложений. Docker разработан для более быстрого выкладывания ваших приложений. С помощью docker вы можете отделить ваше приложение от вашей инфраструктуры и обращаться с инфраструктурой как управляемым приложением. Docker помогает выкладывать ваш код быстрее, быстрее тестировать, быстрее выкладывать приложения и уменьшить время между написанием кода и запуска кода. Docker делает это с помощью легковесной платформы контейнерной виртуализации, используя процессы и утилиты, которые помогают управлять и выкладывать ваши приложения.

В своем ядре docker позволяет запускать практически любое приложение, безопасно изолированное в контейнере. Безопасная изоляция позволяет вам запускать на одном хосте много контейнеров одновременно. Легковесная природа контейнера, который запускается без дополнительной нагрузки гипервизора, позволяет вам добиваться больше от вашего железа.

Платформа и средства контейнерной виртуализации могут быть полезны в следующих случаях:

- упаковывание вашего приложения (и так же используемых компонент) в docker контейнеры;
- раздача и доставка этих контейнеров вашим командам для разработки и тестирования;
- выкладывания этих контейнеров на ваши продакшены, как в дата центры так и в облака.

Для чего я могу использовать docker?

Быстрое выкладывание ваших приложений

Docker прекрасно подходит для организации цикла разработки. Docker позволяет разработчикам использовать локальные контейнеры с приложениями и сервисами.

Что в последствии позволяет интегрироваться с процессом постоянной интеграции и выкладывания (continuous integration and deployment workflow).

Например, ваши разработчики пишут код локально и делятся своим стеком разработки (набором docker образов) с коллегами. Когда они готовы, отравляют код и контейнеры на тестовую площадку и запускают любые необходимые тесты. С тестовой площадки они могут оправить код и образы на продакшен.

Более простое выкладывание и разворачивание

Основанная на контейнерах docker платформа позволят легко портировать вашу полезную нагрузку. Docker контейнеры могут работать на вашей локальной машине, как реальной так и на виртуальной машине в дата центре, так и в облаке.

Портируемость и легковесная природа docker позволяет легко динамически управлять вашей нагрузкой. Вы можете использовать docker, чтобы развернуть или погасить ваше приложение или сервисы. Скорость docker позволяет делать это почти в режиме реального времени.

Высокие нагрузки и больше полезных нагрузок

Docker легковесен и быстр. Он предоставляет устойчивую, рентабельную альтернативу виртуальным машинам на основе гипервизора. Он особенно полезен в условиях высоких нагрузок, например, при создания собственного облака или платформа-как-сервис (platform-as-service). Но он так же полезен для маленьких и средних приложений, когда вам хочется получать больше из имеющихся ресурсов.

Главные компоненты Docker

Docker состоит из двух главных компонент:

- Docker: платформа виртуализации с открытым кодом;
- Docker Hub: наша платформа-как-сервис для распространения и управления docker контейнерами.

Примечание! Docker распространяется по Apache 2.0 лицензии.

Архитектура Docker

Docker использует архитектуру клиент-сервер. Docker клиент общается с демоном Docker, который берет на себя тяжесть создания, запуска, распределения ваших контейнеров. Оба, клиент и сервер могут работать на одной системе, вы можете подключить клиент к удаленному демону docker. Клиент и сервер общаются через сокет или через RESTful API.

Docker-демон

Как показано на диаграмме, демон за пускается на хост-машине. Пользователь не взаимодействует с сервером на прямую, а использует для этого клиент.

Docker-клиент

Docker-клиент, программа docker — главный интерфейс к Docker. Она получает команды от пользователя и взаимодействует с docker-демоном.

Внутри docker-а

Чтобы понимать, из чего состоит docker, вам нужно знать о трех компонентах:

- образы (images)
- реестр (registries)
- контейнеры

Образы

Docker-образ — это read-only шаблон. Например, образ может содержать операционку Ubuntu с Apache и приложением на ней. Образы используются для создания контейнеров. Docker позволяет легко создавать новые образы, обновлять существующие, или вы можете скачать образы созданные другими людьми. Образы — это компонента сборки docker-а.

Реестр

Docker-реестр хранит образы. Есть публичные и приватные реестры, из которых можно скачать либо загрузить образы. Публичный Docker-реестр — это Docker Hub. Там хранится огромная коллекция образов. Как вы знаете, образы могут быть

созданы вами или вы можете использовать образы созданные другими. Реестры — это компонента распространения.

Контейнеры

Контейнеры похожи на директории. В контейнерах содержится все, что нужно для работы приложения. Каждый контейнер создается из образа. Контейнеры могут быть созданы, запущены, остановлены, перенесены или удалены. Каждый контейнер изолирован и является безопасной платформой для приложения. Контейнеры — это компонента работы.

Так как же работает Docker?

Пока мы знаем, что:

- можем создавать образы, в которых находятся наши приложения;
- можем создавать контейнеры из образов, для запуска приложений;
- можем распространять образы через Docker Hub или другой реестр образов.

Давайте посмотрим, как эти компоненты сочетаются.

Как работает образ?

Мы уже знаем, что образ — это read-only шаблон, из которого создается контейнер. Каждый образ состоит из набора уровней. Docker использует union file system для сочетания этих уровней в один образ. Union file system позволяет файлам и директориями из разных файловых систем (разным ветвям) прозрачно накладываться, создавая когерентную файловую систему.

Одна из причин, по которой docker легковесен — это использование таких уровней. Когда вы изменяете образ, например, обновляете приложение, создается новый уровень. Так, без замены всего образа или его пересборки, как вам возможно придётся сделать с виртуальной машиной, только уровень добавляется или обновляется. И вам не нужно раздавать весь новый образ, раздается только обновление, что позволяет распространять образы проще и быстрее.

В основе каждого образа находится базовый образ. Например, ubuntu, базовый образ Ubuntu, или fedora, базовый образ дистрибутива Fedora. Так же вы можете использовать образы как базу для создания новых образов. Например, если у вас есть

образ apache, вы можете использовать его как базовый образ для ваших веб-приложений.

Примечание! Docker обычно берет образы из реестра Docker Hub.

Docker образы могут создаться из этих базовых образов, шаги описания для создания этих образов мы называем инструкциями. Каждая инструкция создает новый образ или уровень. Инструкциями будут следующие действия:

- запуск команды
- добавление файла или директории
- создание переменной окружения
- указания что запускать когда запускается контейнер этого образа

Эти инструкции хранятся в файле Dockerfile. Docker считывает это Dockerfile, когда вы собираете образ, выполняет эти инструкции, и возвращает конечный образ.

Как работает docker реестр?

Реестр — это хранилище docker образов. После создания образа вы можете опубликовать его на публичном реестре Docker Hub или на вашем личном реестре.

С помощью docker клиента вы можете искать уже опубликованные образы и скачивать их на вашу машину с docker для создания контейнеров.

Docker Hub предоставляет публичные и приватные хранилища образов. Поиск и скачивание образов из публичных хранилищ доступно для всех. Содержимое приватных хранилищ не попадает в результат поиска. И только вы и ваши пользователи могут получать эти образы и создавать из них контейнеры.

Как работает контейнер?

Контейнер состоит из операционной системы, пользовательских файлов и метаданных. Как мы знаем, каждый контейнер создается из образа. Этот образ говорит docker-у, что находится в контейнере, какой процесс запустить, когда запускается контейнер и другие конфигурационные данные. Docker образ доступен только для чтения. Когда docker запускает контейнер, он создает уровень для чтения/записи

сверху образа (используя union file system, как было указано раньше), в котором может быть запущено приложение.

Что происходит, когда запускается контейнер?

Или с помощью программы docker, или с помощью RESTful API, docker клиент говорит docker демону запустить контейнер.

```
$ sudo docker run -i -t ubuntu /bin/bash
```

Давайте разберемся с этой командой. Клиент запускается с помощью команды docker, с опцией run, которая говорит, что будет запущен новый контейнер. Минимальными требованиями для запуска контейнера являются следующие атрибуты:

- какой образ использовать для создания контейнера. В нашем случае ubuntu
- команду которую вы хотите запустить когда контейнер будет запущен. В нашем случае /bin/bash

Что же происходит под капотом, когда мы запускаем эту команду?

Docker, по порядку, делает следующее:

- **скачивает образ ubuntu:** docker проверяет наличие образа ubuntu на локальной машине, и если его нет — то скачивает его с Docker Hub. Если же образ есть, то использует его для создания контейнера;
- **создает контейнер:** когда образ получен, docker использует его для создания контейнера;
- **инициализирует файловую систему и монтирует read-only уровень:** контейнер создан в файловой системе и read-only уровень добавлен образ;
- **инициализирует сеть/мост:** создает сетевой интерфейс, который позволяет docker-у общаться хост машиной;
- **Установка IP адреса:** находит и задает адрес;
- **Запускает указанный процесс:** запускает ваше приложение;
- **Обрабатывает и выдает вывод вашего приложения:** подключается и логирует стандартный вход, вывод и поток ошибок вашего приложения, что бы вы могли отслеживать как работает ваше приложение.

Теперь у вас есть рабочий контейнер. Вы можете управлять своим контейнером, взаимодействовать с вашим приложением. Когда решите остановить приложение, удалите контейнер.

Используемые технологии

Докер написан на Go и использует некоторые возможности ядра Linux, чтобы реализовать приведенный выше функционал.

Пространство имен(namespaces)

Docker использует технологию namespaces для организации изолированных рабочих пространств, которые мы называем контейнерами. Когда мы запускаем контейнер, docker создает набор пространств имен для данного контейнера.

Это создает изолированный уровень, каждый аспект контейнера запущен в своем простанстве имен, и не имеет доступ к внешней системе.

Список некоторых пространств имен, которые использует docker:

- **pid:** для изоляции процесса;
- **net:** для управления сетевыми интерфейсами;
- **ipc:** для управления IPC ресурсами. (ICP: InterProccess Communication);
- **mnt:** для управления точками монтирования;
- **utc:** для изолирования ядра и контроля генерации версий(UTC: Unix timesharing system).

Control groups (контрольные группы)

Docker также использует технологию cgroups или контрольные группы. Ключ к работе приложения в изоляции, предоставление приложению только тех ресурсов, которые вы хотите предоставить. Это гарантирует, что контейнеры будут хорошими соседями. Контрольные группы позволяют разделять доступные ресурсы железа и если необходимо, устанавливать пределы и ограничения. Например, ограничить возможное количество памяти контейнеру.

Union File System

Union File Sysem или UnionFS — это файловая система, которая работает создавая уровни, делая ее очень легковесной и быстрой. Docker использует UnionFS для создания блоков, из которых строится контейнер. Docker может использовать несколько вариантов UnionFS включая: AUFS, btrfs, vfs и DeviceMapper.

Форматы контейнеров

Docker сочетает эти компоненты в обертку, которую мы называем форматом контейнера. Формат, используемый по умолчанию, называется libcontainer. Так же docker поддерживает традиционный формат контейнеров в Linux с помощью LXC. В будущем Docker возможно будет поддерживать другие форматы контейнеров. Например, интегрируясь с BSD Jails или Solaris Zones.

2.1.2 Kubernetes

Предисловие

Технологический прогресс не стоит на месте, особенно если это касается области IT. Еще в 2014 году, когда Google открыл исходный код проекта Kubernetes, который вобрал в себя лучшее из Borg и Omega - внутренних систем Google, его мало кто знал и использовал. Но прошло 7 лет и K8S (он же Kubernetes) стал успешным open-source проектом, который используется по всему миру. В большинстве компаний инфраструктура построена на K8S - от маленьких проектов до огромных кластеров и облаков, которые обеспечивают бесперебойную доступность сервисов. Давайте коснемся Kubernetes и расскажем, как им пользоваться на практике. Но сначала надо спросить, а что такое этот ваш Kubernetes и для чего он собственно нужен. Подойдет ли он для ваших проектов.

Что такое Kubernetes?

Слово “Kubernetes” происходит от древнегреческого κυβερνήτης, что означает капитан, рулевой, пилот; тот, кто управляет. В сокращении “K8S” цифра 8 - это восемь букв между K и S. Это платформа с открытым исходным кодом для развертывания, масштабирования, управления и контроля контейнеризованных приложений либо сервисов.

Kubernetes делает следующее:

- Управляет и запускает контейнеры

- Балансирует сетевой трафик между узлами кластера Kubernetes и количеством реплик контейнеров
- Осуществляет контроль состояния, автоматические развертывания и откаты реплик контейнеров внутри узлов кластера Kubernetes
- Осуществляет распределение нагрузки между узлами кластера Kubernetes
- Предоставляет автоматическое монтирование систем хранения для контейнеров
- Предоставляет декларативный API и CLI для управления
- И еще множество полезных, и не очень, модулей и сервисов, которые можно развернуть для управления автоматизацией, инфраструктурой и контейнерами

Kubernetes не делает следующее:

- Не собирает контейнеры с исходным кодом вашего приложения или сервиса
- Не предоставляет процессы и решения непрерывной интеграции (CI)
- Не включает в себя решения и системы сбора журналов и метрик
- Не включает в себя решения и системы хранения данных
- Не включает в себя решения и системы хранения контейнеров (registry)
- Не включает в себя решения и системы от всех бед и болячек инфраструктуры

Kubernetes или K8S — это не просто система оркестрации. Техническое определение оркестрации — это выполнение определенного рабочего процесса: сначала сделай A, затем B, затем C. Напротив, Kubernetes содержит набор независимых, компонуемых процессов управления, которые непрерывно переводят текущее состояние к предполагаемому состоянию. Неважно, как добраться от A до C. Не требуется также и централизованный контроль. Это делает систему более простой в использовании, более мощной, надежной, устойчивой и расширяемой.

Однако все вышеизложенные определения так и не дают нам ключевого понимания,почему и зачем нужен Kubernetes. Основной фактор использования Kubernetes в технологических компаниях, где ведется активная разработка приложений, - это гибкий подход к разработке. Сегодня подход в построении архитектуры приложений изменился - приложения больше не выглядят как монолит кода или один большой сервис, где весь функционал был в одном репозитории. Раньше сборки проектов занимали достаточно много времени, но с приходом контейнеров и таких методологий как DevOps приложения стали модульными, и теперь за каждую функцию или группу функций отвечают определенные сервисы этого приложения. Это можно сравнить с кирпичиками конструктора LEGO: из всех деталей

складывается наше приложение, каждый сервис мы можем достать, чтобы что-то изменить и протестировать и вставить обратно в нашу конструкцию. Главная идея состоит в том, чтобы быстро внедрять новый функционал в уже имеющееся приложение.

Но что если у нас таких сервисов тысячи, каждый за что-то отвечает и работает сам по себе? А если еще развернуты несколько реплик для отказоустойчивости? Как управлять всем и уделить внимание каждому из них? Как понять, что сервис правильно работает и взаимодействует с другими? Для этого есть специальные системы, оркестраторы в своем роде, такие как HashiCorp Nomad, Docker Swarm и Kubernetes. Последний используется активнее всего, так как предоставляет более гибкий функционал в контексте управления всей конструкцией приложения.

На самом деле бизнесу нужен не Kubernetes, а система, которая позволит более быстрее подходить к изменению рынка и подстраиваться под его запросы предоставления набора новых услуг или вывода старых. В НАТ.Тех мы давно используем этот инструмент и чувствуем большую разницу, как для нас, так и для наших клиентов. Исходя из нашего опыта, бизнес чаще всего ценит такие возможности K8S как собирать и тестировать только часть приложения, с которой мы работаем, что в разы уменьшает объем необходимых ресурсов; добавлять и убирать сервисы «на лету», тестировать новый функционал в разных регионах и смотреть, как он себя показывает.

Именно для этого нам и нужен Kubernetes, который дает унификацию и гибкость в способе обслуживания и содержания сервисов приложения. Kubernetes предоставляет:

- Быструю и автоматическую масштабируемость. При росте нагрузки можно быстро добавить необходимые узлы приложения, а также быстро их вывести, чтобы не тратить драгоценные ресурсы
- Гибкий подход к эксплуатации. Мы можем быстро и легко построить структуру приложения, так как вся структура описывается в конфигурационных файлах - манифестах
- Гибкий подход в управлении. Kubernetes не потребует перестройки инфраструктуры и прочего, если вы захотели провести тестирование, внедрить новый сервис или сделать деплой по методологии blue-green
- Универсальность. С помощью манифестов легко переехать, если вы захотели поменять провайдера или переезжали в свой собственный кластер

• Низкий порог вхождения в использование. Kubernetes довольно легок в освоении манифестов, потому что большую часть работы он делает за вас

Если вы задумываетесь о преимуществах, описанных выше, и можете точно сказать, что вам нужна гибкость в разработке и быстрое внедрение сервисов, адаптируемый подход и универсальность в управлении большим количеством сервисов и их реплик, то думаю, что пора попробовать Kubernetes и у себя.

Однако, если ваш проект имеет постоянную нагрузку и не требует высокой степени гибкости и быстрого масштабирования, новый функционал появляется редко и у вас есть команда, уверенно работающая с существующим окружением, то на данный момент возможности K8S для бизнеса избыточны, но "посмотреть" на технологию в фоновом режиме все же стоит, так как те или иные условия могут и поменяться.

Внедрение Kubernetes

Kubernetes, так как он был реализован как облачное решение, предпочтительнее разворачивать именно в облаке.

Если вы решились ставить Kubernetes внутри, на своих собственных ресурсах, то вам придется позаботиться о развёртывании и обслуживании такой инфраструктуры вокруг кластера как: репозиторий для контейнеров, внешние или внутренние балансировщики, сетевые хранилища, хранилище секретов, решения для сбора логов и метрик. Также важна и внутренняя инфраструктура “кубера”: CertManager, Ingress, Istio и другие.

При этом существует много компаний, которые предоставляют Kubernetes как IaaS-решение, где не требуется поднимать и обслуживать окружение для кластера и процессов, которые будут на нем. Или, например, в NUT.Tech, где я сейчас работаю, разрабатываются решения “под ключ” индивидуально под запрос заказчика.

Компоненты и архитектура

Давайте поверхностно коснемся архитектуры самого K8S и его важных компонентов.

Сам кластер K8S состоит из, барабанная дробь, рабочих узлов. В узлах или нодах (Nodes, Worker nodes), помимо контейнеров компонентов самого кластера, размещаются контейнеры наших проектов и сервисов.

Worker nodes состоит из компонентов:

• kubelet - сервис или агент, который контролирует запуск компонентов (контейнеров) кластера

• kube-proxy - конфигурирует правила сети на узлах

Плоскость управления (Master nodes) управляет рабочими узлами и подами в кластере. Там располагаются компоненты, которые управляют узлами кластера и предоставляют доступ к API.

Control plane состоит из компонентов:

- kube-apiserver - предоставляет API кубера
- etcd - распределенное key-value хранилище для всех данных кластера. Необязательно располагается внутри мастера, может стоять как отдельный кластер
- kube-scheduler - планирует размещение подов на узлах кластера
- kube-controller-manager - запускает контроллеры
- kubelet - сервис или агент, который контролирует запуск основных компонентов (контейнеров) кластер

Виды контроллеров

Deployments - контроллер, который управляет состоянием развертывания подов, которое описывается в манифесте, следит за удалением и созданием экземпляров подов. Управляет контроллерами ReplicaSet.

ReplicaSet - гарантирует, что определенное количество экземпляров подов всегда будет запущено в кластере.

StatefulSets - так же как и Deployments, управляет развертыванием и масштабированием набора подов, но сохраняет набор идентификаторов и состояние для каждого пода.

DaemonSet - гарантирует, что на каждом узле кластера будет присутствовать экземпляр пода.

Jobs - создает определенное количество подов и смотрит, пока они успешно не завершат работу. Если под завершился с ошибкой, повторяет создание, которое мы описали определенное количество раз. Если под успешно отработал, записывает это в свой журнал.

CronJob - запускает контроллеры Jobs по определенному расписанию.

Тестовые кластеры, где потренироваться

Где можно потренироваться, если нет своего кластера, а платить за облако не хочется? Есть так называемые мини кластеры, которые можно запустить у себя на локальной машине - один из них kind https://kind.sigs.K8S.io/. Все, что нам потребуется, это установленный docker. Также есть Minikube https://Kubernetes.io/ru/docs/tasks/tools/install-minikube/ - для него уже потребуется гипервизор.

Kubectl: как пользоваться, основные команды

При работе с кластером нам потребуется инструмент командной строки kubectl. https://Kubernetes.io/ru/docs/tasks/tools/install-kubectl/

Его можно установить на локальную машину и управлять несколькими кластерами с одной точки.

Основные команды, которые мы часто будем использовать:

Kubectl get [имя объекта] -n [Имя_пространства_имен] - команда get, как несложно понять из ее названия, выводит нужную нам информацию в основном виде таблицы. Также с помощью нее можно получить yaml, который хранится внутри кластера. Очень часто применяется ключ -n для указания пространства имен, где лежат объекты.

Примеры:

kubectl get services -n Имя_неймспейса - выводит все сервисы в пространстве имён

kubectl get pods --all-namespaces - выводит все поды во всех пространств имён

kubectl get pods -o wide - выводит все поды в текущем пространстве имён с подробностями в виде расширенной таблицы

kubectl get pods -n Имя_неймспейса - выводит все поды в пространстве имён

Kubectl apply -f [имя манифеста yaml] - команда apply применяет манифест к кластеру, управляет созданием объектов в кластере с помощью манифестов yaml. Если в манифесте не указано пространство имен, его можно задать с помощью ключа -n [Имя_пространства_имен].

Примеры:

kubectl apply -f ./my-manifest.yaml - создать ресурсы

kubectl apply -f ./my1.yaml -f ./my2.yaml - создать ресурсы из нескольких файлов

kubectl apply -f ./dir - создать ресурсы из всех файлов манифеста в директории

kubectl apply -f https://K8S.io/manifest - создать ресурсы из URL-адреса

Отличная шпаргалка по командам есть в документации https://Kubernetes.io/ru/docs/reference/kubectl/cheatsheet/

Тестовый кластер

Давайте развернем свой k8s кластер на примере *kind* https://kind.sigs.k8s.io/

Первым делом нам понадобится установить *docker* https://docs.docker.com/engine/install/

После установим сам *kind* https://kind.sigs.k8s.io/docs/user/quick-start#installation

kind create cluster --name k8s-test-1

--name - ключ задает имя вашего кластера в kind, по умолчанию кластер будет называться kind, давайте назовем его k8s-test-1.

После выполнения команды посмотрим и убедимся, что все хорошо и наш кластер появился в списке.

~ % kind get clusters

k8s-test-1

Убедимся, что в kubectl добавился context нашего кластера

kubectl cluster-info --context kind-k8s-test-1

context - это конфигурация, которую вы используете для подключения к вашим кластерам

kind-k8s-test-1 - имя контекста нашего кластера, так как мы создали кластер в kind, имя будет начинаться с kind-[имя кластера в kind]

Что такое Pods

Pods или поды — это абстрактный объект в кластере K8S, который состоит из одного или нескольких контейнеров с общим хранилищем и сетевыми ресурсами, а также спецификации для запуска контейнеров.

Это главный объект в кластере, в нем прописаны, какие контейнеры должны быть запущены, количество экземпляров или реплик, политика перезапуска, лимиты, подключаемые ресурсы, узел кластера для размещения.

kube-scheduler планирует размещение пода на узлах кластера

kubelet на рабочем узле кластера запускает под

Любители забегать вперед и просто углубиться в тему могут почитать прекрасную статью на github.

Что такое Namespace

Namespace или пространство имен — это абстрактный объект, который логически разграничивает и изолирует ресурсы между подами. Вы можете рассматривать пространство имен как внутренний виртуальный кластер, который поможет вам изолировать проекты или пользователей между собой, применить разные политики квот на свои проекты или выдать права доступа только на определенную область.

У нас уже есть пространства имен, которые создаются при создании кластера, - чтобы увидеть их, нужно выполнить команду:

kubectl get namespace

Ответ:

NAME STATUS AGE
default Active 1d
kube-system Active 1d
kube-public Active 1d

default — пространство имён по умолчанию для объектов без какого-либо другого пространства имён

kube-system — пространство имён для объектов, созданных Kubernetes. Там размещаются системные поды кластера

kube-public — создаваемое автоматически пространство имён, которое доступно для чтения всем пользователям (включая также неаутентифицированных пользователей)

Размещение объектов

Давайте разместим свой первый объект в нашем тестовом кластере, для этого создадим свой namespace и положим туда простенький pod с nginx.

Чтобы создать свой namespace, нам нужно передать его спецификацию или манифест в формате yaml.

В файле .yaml создаваемого объекта Kubernetes необходимо указать значения для следующих полей:

apiVersion — используемая для создания объекта версия API Kubernetes. Стоит отметить, что из версии к версии версии api могут меняться

kind — тип создаваемого объекта

metadata — данные, позволяющие идентифицировать объект (name, UID и необязательное поле namespace)

spec — требуемое состояние объекта

Давайте создадим test-namespace.yaml со следующим содержанием:

```
apiVersion: v1
kind: Namespace
metadata:
  name: test
```

И применим его с помощью команды:

```
kubectl apply -f test-namespace.yaml
```

Посмотрим на результат:

```
kubectl get namespace
NAME        STATUS    AGE
default     Active    1d
```

kube-system Active 1d
kube-public Active 1d
test Active 10s

Как видим, наш namespace успешно создался. Приступим к созданию своего первого пода (pod). Давайте положим контейнер с NGINX в наш namespace.

Как и с namespace, опишем его. Создадим hello.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-hello
  labels:
    app: nginx-hello
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx-hello
  template:
    metadata:
      labels:
        app: nginx-hello
    spec:
      containers:
      - name: nginx-hello
        image: nginxdemos/hello
        ports:
        - containerPort: 80
```

Применим его

kubectl apply -f hello.yaml

Если манифест применился то увидим:

deployment.apps/nginx-hello created

Узнаем развернулся ли наш контейнер

kubectl get pods -n test

```
~# kubectl get pods -n test
NAME       READY   STATUS          RESTARTS   AGE
```

nginx-hello-75cc4f9fb7-rv556 1/1 Running 0 9s

Как видим, наш контейнер создался. Сначала он был в статусе *ContainerCreating*, потом перешел в *Running*. Давайте проверим и убедимся, что контейнер работает и что-то нам отдает, для этого на поможет kubectl port-forward [имя контейнера] [порт на который будем перенаправлять]:[порт контейнера] -n [неймспейс].

К примеру

kubectl port-forward nginx-hello-75cc4f9fb7-rv556 30080:80 -n test

Зайдем через наш браузер на http://localhost:30080/

Если вы видите в браузере:

Server address:127.0.0.1:80

Server name:nginx-hello-75cc4f9fb7-rv556

Date:12/Nov/2021:11:40:36 +0000

То контейнер работает, неожиданно.

2.2 Задание на лабораторную работу

На основе имеющейся концептуальной модели провести сравнительный анализ внедрения облачных технологий Docker и Kubernetes (K8S).

Лабораторная работа 3
«Анализ облачных хранилищ»

3.1 Краткие теоретические сведения

3.1.1 Нереляционные и реляционные базы данных

NoSQL – это подход к реализации масштабируемого хранилища (базы) информации с гибкой моделью данных, отличающийся от классических реляционных СУБД. В нереляционных базах проблемы масштабируемости (scalability) и доступности (availability), важные для Big Data, решаются за счёт атомарности (atomicity) и согласованности данных (consistency) **[1]**.

Зачем нужны нереляционные базы данных в Big Data: история появления и развития

NoSQL-базы оптимизированы для приложений, которые должны быстро, с низкой временной задержкой (low latency) обрабатывать большой объем данных с разной структурой **[2]**. Таким образом, нереляционные хранилища непосредственно ориентированы на Big Data. Однако, идея баз данных такого типа зародилась гораздо раньше термина «большие данные», еще **в 80-е годы прошлого века**, во времена первых компьютеров (мэйнфреймов) и использовалась для иерархических служб каталогов. Современное понимание NoSQL-СУБД возникло **в начале 2000-х годов**, в рамках создания параллельных распределённых систем для высокомасштабируемых интернет-приложений, таких как онлайн-поисковики **[1]**.

Вообще термин NoSQL обозначает «не только SQL» (Not Only SQL), характеризуя ответвление от традиционного подхода к проектированию баз данных. Изначально так называлась опенсорсная база данных, созданная Карло Строззи, которая хранила все данные как ASCII-файлы, а вместо SQL-запросов доступа к данным использовала шелловские скрипты **[3]**. **В начале 2000-х годов** Google построил свою поисковую систему и приложения (GMail, Maps, Earth и прочие сервисы), решив проблемы масштабируемости и параллельной обработки больших объёмов данных. Так была создана распределённые файловая и координирующая системы, а также колоночное хранилище (column family store), основанное на вычислительной модели MapReduce. После того, как корпорация Google опубликовала описание этих технологий, они стали очень популярны у разработчиков открытого программного обеспечения. В результате этого был создан Apache Hadoop и запущены основные связанные с ним проекты. Например, в 2007 году другой ИТ-гигант, Amazon.com, опубликовав статьи о своей высокодоступной базе данных Amazon

DynamoDB. Далее в эту гонку NoSQL- технологий для управления большими данными включилось множество корпораций: IBM, Facebook, Netflix, eBay, Hulu, Yahoo! и другие ИТ-компаний со своими проприетарными и открытыми решениями **[1]**.

Многообразие NoSQL-решений

Какие бывают NoSQL-СУБД: основные типы нереляционных баз данных

Все NoSQL решения принято делить на 4 типа:

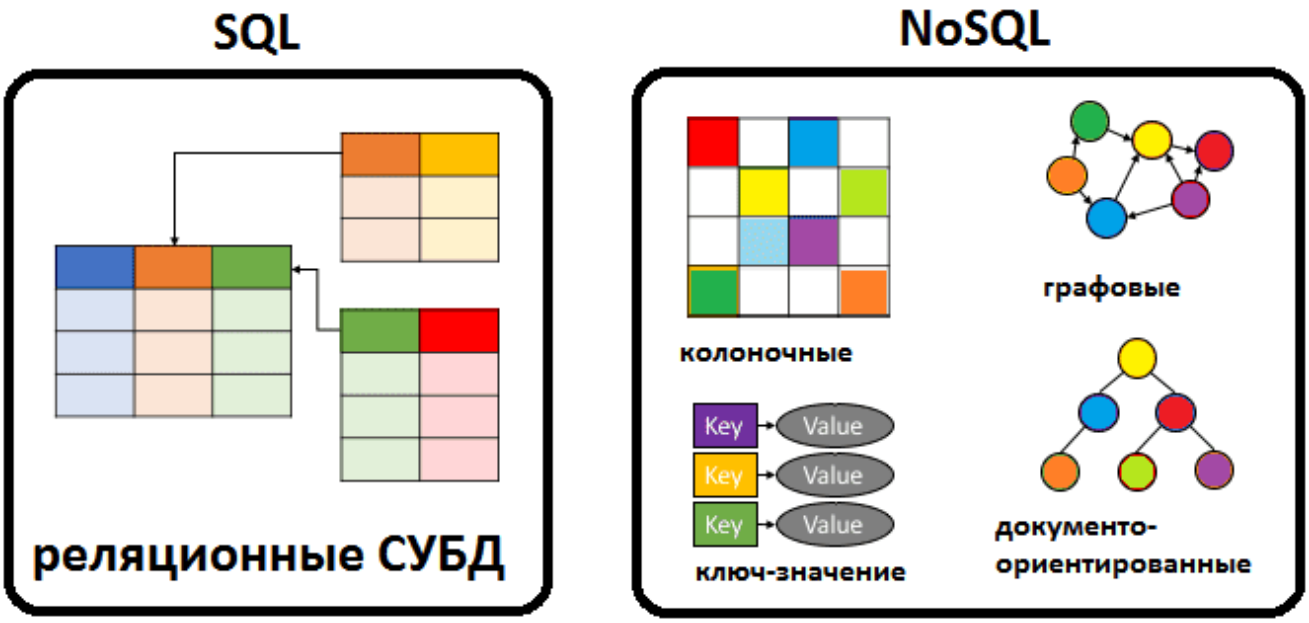
1. **Ключ-значение (Key-value)** – наиболее простой вариант хранилища данных, использующий ключ для доступа к значению в рамках большой хэш-таблицы **[4]**. Такие СУБД применяются для хранения изображений, создания специализированных файловых систем, в качестве кэшей для объектов, а также в масштабируемых Big Data системах, включая игровые и рекламные приложения, а также проекты интернета вещей (Internet of Things, IoT), в т.ч. индустриального (Industrial IoT, IIoT). Наиболее известными представителями нереляционных СУБД типа key-value считаются Oracle NoSQL Database, Berkeley DB, MemcacheDB, Redis, Riak, Amazon DynamoDB, которые поддерживают высокую разделяемость, обеспечивая беспрецедентное горизонтальное масштабирование, недостижимое при использовании других типов БД **[2]**.

2. **Документно-ориентированное хранилище**, в котором данные, представленные парами ключ-значение, сжимаются в виде полуструктурированного документа из тегированных элементов, подобно JSON, XML, BSON и другим подобным форматам **[4]**. Такая модель хорошо подходит для каталогов, пользовательские профилей и систем управления контентом, где каждый документ уникален и изменяется со временем **[2]**. Поэтому чаще всего документные NoSQL-СУБД

используются в CMS-системах, издательском деле и документальном поиске. Самые яркие примеры документно-ориентированных нереляционных баз данных – это CouchDB, Couchbase, MongoDB, eXist, Berkeley DB XML **[1]**.

3. **Колоночное хранилище**, которое хранит информацию в виде разреженной матрицы, строки и столбцы которой используются как ключи. В мире Big Data к колоночным хранилищам относятся базы типа «семейство столбцов» (Column Family). В таких системах сами значения хранятся в столбцах (колонках), представленных в отдельных файлах. Благодаря такой модели данных можно хранить большое количество атрибутов в сжатом виде, что ускоряет выполнение запросов к базе, особенно операции поиска и агрегации данных **[4]**. Наличие временных меток (timestamp) позволяет использовать такие СУБД для организации счётчиков, регистрации и обработки событий, связанных со временем: системы биржевой аналитики, IoT/IIoT-приложения, систему управления содержимым и т.д. Самой известной колоночной базой данных является Google Big Table, а также основанные на ней Apache HBase и Cassandra. Также к этому типу относятся менее популярные ScyllaDB, Apache Accumulo и Hypertable **[1]**.

4. **Графовое хранилище** представляют собой сетевую базу, которая использует узлы и рёбра для отображения и хранения данных **[4]**. Поскольку рёбра графа являются хранимыми, его обход не требует дополнительных вычислений (как соединение в SQL). При этом для нахождения начальной вершины обхода необходимы индексы. Обычно графовые СУБД поддерживают ACID-требования и специализированные языки запросов (Gremlin, Cypher, SPARQL, GraphQL и т.д.) **[1]**. Такие СУБД используются в задачах, ориентированных на связи: социальные сети, выявление мошенничества, маршруты общественного транспорта, дорожные карты, сетевые топологии **[3]**. Примеры графовых баз: InfoGrid, Neo4j, Amazon Neptune, OrientDB, AllegroGraph, Blazegraph, InfiniteGraph, FlockDB, Titan, ArangoDB. О том, как проанализировать граф в Neo4j средствами языка запросов Cypher.



Виды NoSQL-СУБД

Чем хороши и плохи нереляционные базы данных: главные достоинства и недостатки

По сравнению с классическими SQL-базами, нереляционные СУБД обладают следующими преимуществами:

- **линейная масштабируемость** – добавление новых узлов в кластер увеличивает общую производительность системы **[1]**;
- **гибкость**, позволяющая оперировать полуструктурированные данные, реализуя, в. т.ч. полнотекстовый поиск по базе **[2]**;
- возможность работать с **разными представлениями информации**, в т.ч. без задания схемы данных **[1]**;
- **высокая доступность** за счет репликации данных и других механизмов отказоустойчивости, в частности, шаринга – **автоматического разделения данных по разным узлам сети**, когда каждый сервер кластера отвечает только за определенный набор информации, обрабатывая запросы на его чтение и запись. Это увеличивает скорость обработки данных и пропускную способность приложения **[5]**.
- **производительность** за счет оптимизации для конкретных видов моделей данных (документной, графовой, колоночной или «ключ-значение») и шаблонов доступа **[2]**;
- **широкие функциональные возможности** – собственные SQL-подобные языки запросов, RESTful-интерфейсы, API и сложные типы данных, например, map, list и struct, позволяющие обрабатывать сразу множество значений **[2]**.

Обратной стороной вышеуказанных достоинств являются следующие недостатки:

- **ограниченная емкость встроенного языка запросов [5]**. Например, HBase предоставляет всего 4 функции работы с данными (Put, Get, Scan, Delete), в Cassandra отсутствуют операции Insert и Join, несмотря на наличие SQL-подобного языка запросов. Для решения этой проблемы используются сторонние средства трансляции классических SQL-выражений в исполнительный код для конкретной нереляционной базы. Например, Apache Phoenix для HBase или универсальный Drill.
- **сложности в поддержке всех ACID-требований к транзакциям** (атомарность, консистентность, изоляция, долговечность) из-за того, что NoSQL-СУБД вместо CAP-модели (согласованность, доступность, устойчивость к разделению) скорее соответствуют модели BASE (базовая доступность, гибкое состояние и итоговая согласованность) **[1]**. Впрочем, некоторые нереляционные СУБД пытаются обойти это ограничение с помощью настраиваемых уровней согласованности, о чем мы рассказывали на примере Cassandra. Аналогичным образом Riak позволяет настраивать требуемые характеристики доступности-согласованности даже для отдельных запросов за счет задания количества узлов, необходимых для подтверждения успешного завершения транзакции **[1]**. Подробнее о CAP-и BASE-моделях мы расскажем в отдельной статье.
- **сильная привязка приложения к конкретной СУБД** из-за специфики внутреннего языка запросов и гибкой модели данных, ориентированной на конкретный случай **[5];**
- **недостаток специалистов по NoSQL-базам** по сравнению с реляционными аналогами **[5]**.

Подводя итог описанию основных аспектов нереляционных СУБД, стоит отметить некоторую некорректность запроса «NoSQL vs SQL» в связи с разными архитектурными подходами и прикладными задачами, на которые ориентированы эти ИТ-средства. Традиционные SQL-базы отлично справляются с обработкой строго типизированной информации не слишком большого объема. Например, локальная ERP-система или облачная CRM. Однако, в случае обработки большого объема полуструктурированных и неструктурированных данных, т.е. Big Data, в распределенной системе следует выбирать из множества NoSQL-хранилищ, учитывая специфику самой задачи. В частности, для самостоятельных решений интернета вещей (Internet of Things), в т.ч. промышленного, отлично подходит Cassandra,

о чем мы рассказывали здесь. А в случае многоуровневой ИТ-инфраструктуры на базе Apache Hadoop стоит обратить внимание на HBase, которая позволяет оперативно, практически в режиме реального времени, работать с данными, хранящимися в HDFS.

Нереляционные СУБД находят больше областей приложений, чем традиционные SQL-решения

3.1.2 ElasticSearch

Elasticsearch, вероятно, самая популярная поисковая система на данный момент с развитым сообществом, поддержкой и горой информации в сети. Однако эта информация поступает непоследовательно и дробно.

Самое первое и главное заблуждение — "нужен поиск, так бери эластик!". Но в действительности, если вам нужен шустрый поиск для небольшого или даже вполне себе крупного проекта, вам стоит разобраться в теме поподробней и вы откажетесь от использования именно этой системы.

Вторая проблема заключается в том, что пытаясь разобраться с начала, получить общую картину окажется непросто. Да инфы навалом, но последовательность в ее изучении выстраивается постфактум. Придется из книг бежать в документацию, а из документации обратно в книги, параллельно разгугливая тонкости, только чтобы понять, что такое Elasticsearch, почему он работает именно так и для чего же его вообще использовать, а где стоит выбрать что-то попроще.

Для наглядности выдумаем себе задачу. Реализация поиска в коллективном блоге по всем материалам и пользователям. Система позволяет создавать теги,

сообщества, геометки и все остальные штуки, которые нам помогают категоризировать огромное количество информации.

Схема хранения данных

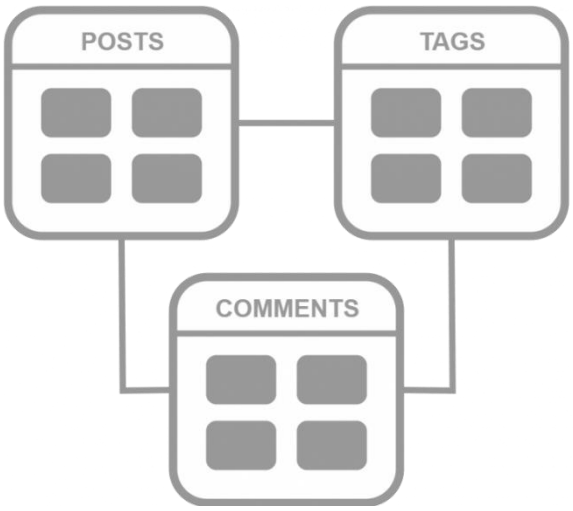
То, какие действия с данными мы будем производить определит схему их хранения:

- скорее всего поисковая система должна будет быстро производить поиск
- запись и удаление могут не отличаться высокой скоростью, в системах поиска, полагаю, этим можно пренебречь
- структура данных будет интенсивно изменяться и хранилище может заполняться из нескольких независимых источников(различных баз, в том числе внешних для нашей системы)

Представьте еще раз сколько атрибутов может иметь публикация и сколько связанных с ней объектов. Автор, категория, сообщество, геометки, медиафайлы, теги, связанные публикации. Этот список можно продолжать до исчерпания фантазии. Если мы храним это в привычной реляционной базе то имеем миллион связей и миллиард атрибутов. Это прекрасно подходит для структурированного хранения долгие годы, но не очень вяжется с требованиями быстрого поиска.

А что если мы захотим добавить пару интеграций с внешними системами? Придется реализовать дополнительные таблицы или даже базы. Нам всегда будет нужно что-то добавить или изменить в объектах доступных для поиска. Вы понимаете к чему я клоню.

Намного быстрее читать из объектов, содержащих все необходимое здесь и сейчас. И намного проще вносить изменения в неструктурированную схему данных.




```
{
   "title" : "С чего начинается Elasticsearch",
   "author" : {
      "name": "Roman"
   },
   "content" : "Elasticsearch, вероятно, самая популярная поисковая система...",
   "tags":[
      "elasticsearch"
   ],
   "ps": "Да, проще всего представить это как JSON, BSON или XML"
}
```

К тому же такие структуры данных проще делить, разносить по разным физическим хранилищам, распределять, ведь объекты уже содержат все необходимое.

Эти объекты мы можем воспринимать как отдельные страницы, файлы, карточки, все это можно назвать некими *документами*. Поэтому такая модель хранения данных называется *документоориентированной*.

Elasticsearch это документоориентированная база данных

Поиск

Теперь необходимо определиться с механизмами поиска. Данные организованы в виде документов. Как мы привыкли осуществлять поиск по документу?

Типичным примером документа будет веб-страница. Если мы попытаемся поискать по всей странице в браузере, поиск будет осуществляться по всему содержащемуся тексту. И это удобно для большинства кейсов.

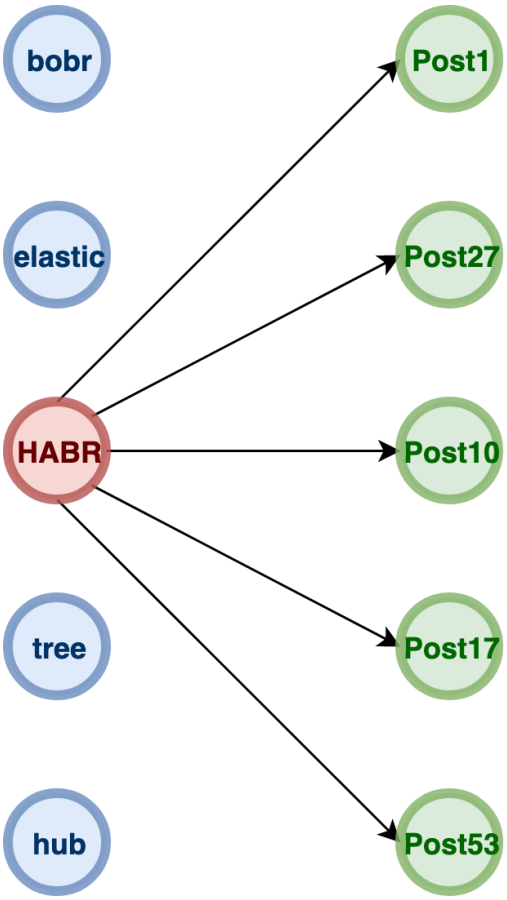
Примерно так же работают многие поисковые системы, поиск происходит по всему тексту проиндексированных страниц, а не по отдельным полям, тегам или заголовкам. Это называется *полнотекстовым* поиском.

Искать предстоит по огромному количеству документов и было бы разумно запомнить что в каком документе лежит. В реляционных СУБД мы привыкли оптимизировать подобный поиск индексами.

Что такое индекс на самом деле? Если не вдаваться в детали, индекс это сбалансированное дерево, то есть дерево, в котором длина путей(количество шагов межу узлами) не будет отличаться больше чем на один шаг.

Например, если бы мы проиндексировали наш пост, то у нас бы получилось дерево, листьями которого, являлись бы используемые в нем слова. Простыми словами, мы будем знать заранее, какие слова находятся в документе и как их быстро в нем найти. Не смотря на такую удобную структуризацию данных, обход дерева звучит как не самое лучшее решение для максимально быстрого поиска.

А что если сделать все наоборот — собрать список всех используемых слов и узнать, в каких документах они встречаются. Да, индексация займет больше времени, но нас в первую очередь интересует именно скорость поиска, а не индексации.



Такой индекс называется *обратным индексом* и используется для полнотекстового поиска.

Хороший пример — популярная open-source библиотека полнотекстового поиска, конечно же, с обратным индексом, Apache Lucene.

Elasticsearch использует индексы Lucene для хранения данных и поиска

Масштабирование

Как бы мы не пытались оптимизировать структуры данных и алгоритмы поиска, когда речь заходит о действительно больших массивах данных и действительно большом количестве запросов, необходимо задуматься о возможности повлиять на производительность системы путем увеличения аппаратного ресурса. Проще говоря, мы хотим иметь возможность накинуть немного памяти, ЦП и дискового пространства, чтобы все ехало быстрее. Мы можем назвать это *масштабируемостью*.

Самый простой вариант — накинуть железа на сервер. Если представить каждую условную единицу вычислительной мощности как деревянный кубик, то

сейчас мы сложим кубики в одно место или один на другой, строя башню *вертикально*. Такое масштабирование и называется *вертикальным*.

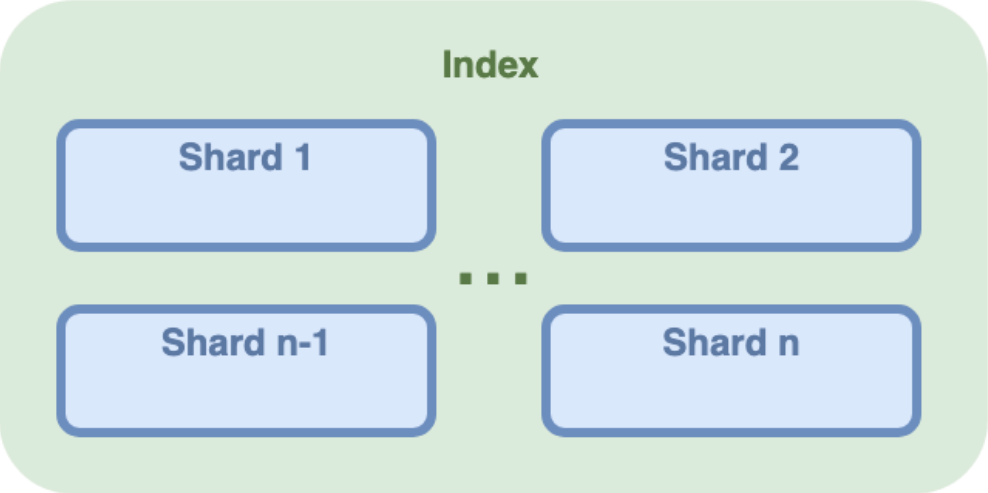
Второй вариант — разделить наши задачи на группу машин. В этом случае мы тоже увеличиваем аппаратный ресурс, но сейчас кубики мы можем расположить на воображаемом столе как угодно на его плоскости, то есть *горизонтально*. Угадайте, как называется такое масштабирование?

Первый способ гарантирует нам быстрый результат без боли, но конечно не все так гладко. Как долго мы сможем увеличивать ресурс отдельной машины? Во-первых дешевым способом это будет только в самом начале, дальше оплата одного сервера будет стоить как несколько машин попроще. Во-вторых вы рано или поздно упретесь в потолок — железо, драйверы, пропускная способность и еще куча логических и физических ограничений. А самое главное, критический сбой в одной машине повлечет сбой всей системы, закономерно.

В отличии от первого способа второй не накладывает таких явных ограничений, мы можем добавлять машины сколько угодно, связывая их сетью. Конечно, это повлечет сетевые издержки — низкая скорость передачи в сети(в сравнении с обработкой на одной машине), сетевой оверхед. Но вместе с тем сеть имеет одно очень важное свойство — большую *отказоустойчивость*.

Распределенный индекс

Для хранения данных и поиска мы будем использовать инстанс Lucene. Но ранее мы решили, что для обеспечения горизонтального масштабирования нам необходимо иметь возможность размещать данные на разных машинах. В действительности, какая разница как данные хранятся физически? Важно чтобы мы имели единое логическое хранилище. Каждый инстанс Lucene должен стать частью одного большого индекса, или осколком(**shard**) разбитого индекса. Шард будет выполнять непосредственно операции по поиску и записи данных.



Shard в **Elasticsearch** — это логическая единица хранения данных на уровне базы, которая является отдельным экземпляром Lucene.

Index — это одновременно и распределенная база и механизм управления и организации данных, это именно логическое пространство. Индекс содержит один или более шардов, их совокупность и является хранилищем.

Классическое сравнение индекса с другими базами выглядит примерно так.

Elasticsearch	**SQL**	**MongoDB**
Index	Database	Database
Mapping/Type	Table	Collection
Field	Column	Field
Object(JSON)	Tuple	Object(BSON)

Но существуют отличия в использовании этих абстракций. Рассмотрим еще один классический пример. У пользователя системы может храниться очень много информации, и мы решаем создавать новую базу для каждого пользователя. Это звучит дико! Но на самом деле в Elasticsearch это распространенная и даже хорошая практика. Индекс это довольно легкий механизм и лучше разделять большие данные, тем более, когда это логически оправдано. Системе проще работать с небольшими индексами чем с разросшейся базой для всего. Например, так вы можете создавать отдельный индекс для логов на каждый день и это широко используется.

По умолчанию количество шардов для индекса будет равным 5, но его всегда возможно изменить в настройках index.number_of_shards: 1 или с помощью запроса шаблонов индекса.

```
PUT _template/all
{
  "template": "*",
     "settings": {
       "number_of_shards": 1
     }
}
```

Важно управлять этим значением. Всегда принимайте решения с точки зрения параллельной обработки.

Каждый шард способен хранить примерно 2^{32} или 4294967296 записей, это значит, что скорее всего вы упретесь в лимит вашего диска. Однако стоит понимать, все шарды будут участвовать в поиске и если мы будем искать по сотне пустых, потратим время впустую. Если шарды будут слишком большими мы так же будем тратить лишнее время на поиск, а так же операций перемещения и индексации станут очень тяжелыми.

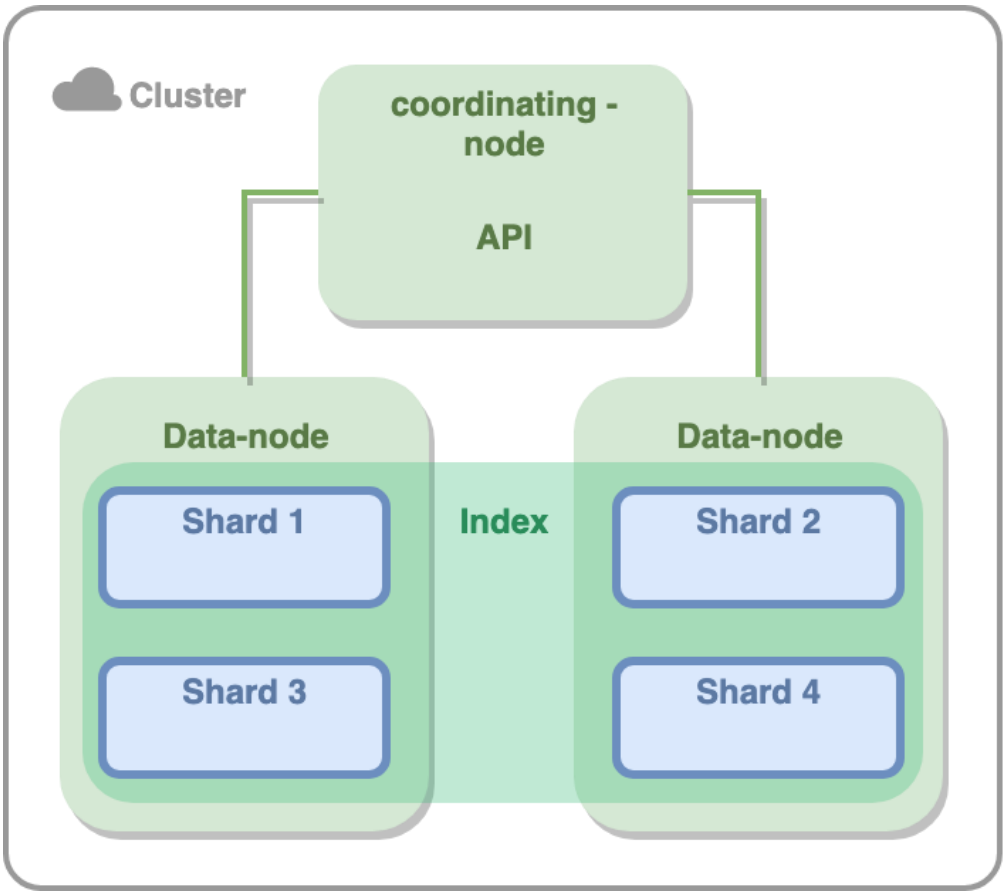
Забегая вперед. Со временем Elasticsearch двигает и изменяет шарды, объединяя дробные и мелкие в большие. Следите за размером ваших шардов, при достижении 10ГБ производительность значительно падает.

Кластер

Мы определились с базовой концепцией распределенного индекса. Сейчас необходимо решить, как в действительности будет осуществляться управление отдельными базами.

Ранее мы решили, что за операции поиска и индексации отвечает отдельный инстанс Lucene(*шард*). Для того, чтобы обращаться к распределенной системе шардов, нам необходимо иметь некий координирующий узел, именно он будет принимать запросы и давать задания на запись или получение данных. То есть помимо хранения данных мы выделяем еще один вариант поведения программы — координирование.

Таким образом мы изначально ориентируемся на два вида узлов — CRUD-узлы и координирующие узлы. Назовем их **data node** и **coordinating node**. У нас есть куча машин объединенных в сеть и все это очень напоминает кластер.



Каждый запущенный экземпляр **Elasticsearch** является отдельным узлом(**node**). **Cluster** — это совокупность определенных нод. Когда вы запускаете один экземпляр ваш кластер будет состоять из одной ноды.

Для того чтобы объединить узлы в кластер они должны соответствовать ряду требований:

- Ноды должны иметь одинаковую версию
- Имя кластера cluster.name в конфигурации должно быть одинаковым

Конфигурация читается из файла **elasticsearch.yml** и переменных среды. Здесь мы можете настроить почти все, что касается неизменных в рантайме свойств ноды.

Вы можете поднимать несколько узлов на одной машине, для этого необходимо запускать инстансы из разных директорий файловой системы и в конфигурации указать разные порты http.port, по умолчанию порт 9200 для внешнего доступа

и 9300 для внутрикластерного соединения. Это может понадобиться для тестирования и проектирования кластера, однако в проде подразумевается использование отдельных машин.

Каждый тип ответственности узлов налагает определенные системные требования. Очевидно, что data-ноды будут часто обращаться к диску и использовать значительные объемы памяти в процессе работы.

Мы так же можем утверждать, что не все данные будут запрашиваться одинаково часто. Данные постепенно "остывают" по мере снижения запросов. Мы можем назвать это жизненным циклом хранения данных. Хорошей идеей было бы держать хайповые публикации там, откуда их можно быстро достать, а забытые мемы 2007 можно положить подальше.

Начиная с версии 6.7 **Elasticsearch** предлагает механизм управления жизненным циклом. Для этого доступны три типа нод — hot, warm и cold.

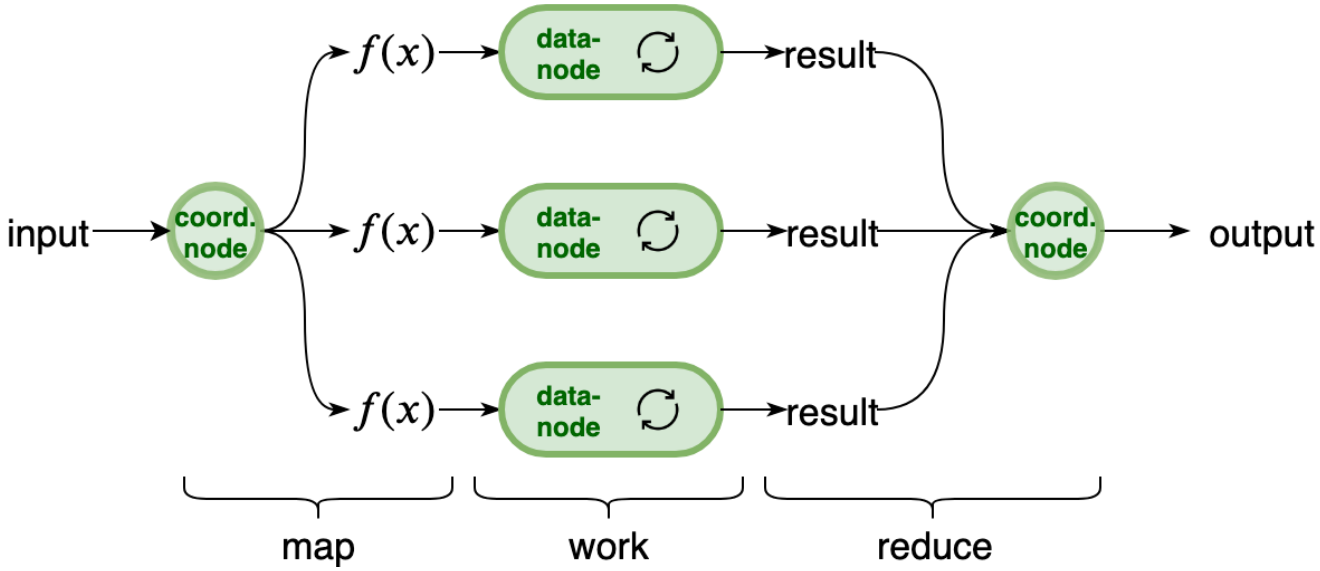
Существует рекомендация по выбору аппаратных конфигураций для каждого из типов. Например hot-ноды должны иметь быстрые SSD, для warm и cold достаточно HDD-диска. Оптимальные соотношения память/диск будут следующими:

- hot — 1:30
- warm — 1:100
- cold — 1:500

Для того чтобы определить тип ноды как data node необходимо установить значение в конфигурации node.data: true, при этом рекомендуется выделять ноду под один конкретный тип, для повышения стабильности и производительности кластера.

Важнейшим аспектом в использовании распределенных систем является параллельное выполнение задач. Существует популярная модель распределенных вычислений, которая имеет лаконичное название **MapReduce**. И заключается она разделении выполнения задачи на два больших шага:

- **Map** — предварительная обработка данных, формулировка задачи и последующая ее передача выполняющим узлам
- **Reduce** — свертка множества результатов worker-нод в один финальный ответ



Именно такой механизм поможет нам выполнять операции с шардами. Координирующий узел получит запрос, предварительно переформулирует его для внутрикластерного взаимодействия и выполнит запросы к нашим worker-нодам(в данном случае к data-нодам).

Следовательно, **coordinating-ноды** должны иметь достаточный ресурс памяти, ЦП и быструю сеть, но при этом могут иметь скромный диск, ведь не осуществляют хранения данных.

Однако при большой частоте запросов, такие ноды могут стать узким местом системы. Мы можем пойти привычным путем и превратить точку внешнего доступа в плоскость. Пусть координирующих нод будет множество.

Такой подход позволит нам применять балансировку запросов, это можно сделать прямо в клиентском коде либо использовать любые существующие балансировщики.

В **Elasticsearch** все узлы неявно являются координирующими. Однако, стоит выделять отдельные coordinating-ноды, не выполняющие других операций. Забегая вперед, такие ноды можно определить, установив значения всех типов в false.

Управление кластером

На данном этапе мы имеем возможность доступа к данным из множества точек — **coordinating-нод**. В этом нет проблем, если мы говорим о простых операциях чтения/записи в существующий индекс. Но если говорить о выделении новых шардов или их перемещении, может начаться путаница.

Предположим, возможность **coordinating-нодам** управления состоянием кластера. Один узел примет решение о перемещении шарда на одну **data-ноду**, а второй о перемещении того же на другую. Список возможных общекластерных действий может быть довольно широким, а список возможных конфликтов еще шире.

Очевидно, такие важные решения должен принимать один центральный узел. Мы определили, что для каждого типа действий необходимо выделять отдельную роль, чтобы избежать потерь производительности на ноде. И "главный в кластере" звучит как отдельная ответственность.

Назовем такие ноды **master-node**. Активный мастер всегда должен быть один, он будет управлять топологией кластера: создавать новый индекс, выделять и распределять шарды, перемещать их и объединять в случае необходимости. Мастер всегда знает все о состоянии кластера.

В кластере **Elasticsearch** обязательно должен быть как минимум один узел отвечающий требованиям **master node**. Для этого в конфигурации ноды необходимо установить значение node.master: true.

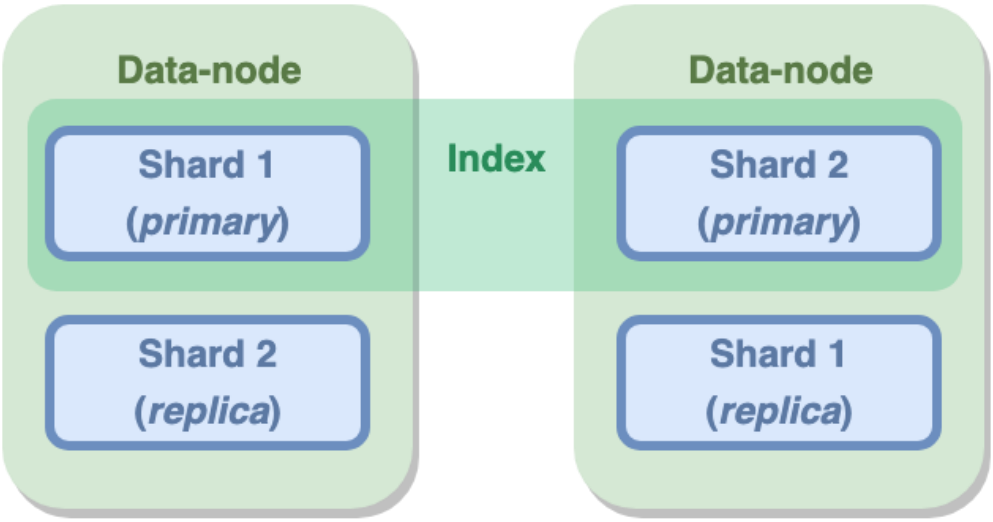
Master-ноды отвечают за важные, но довольно легкие общекластерные действия. Это означает, что они требуют большого ресурса и высокой стабильности от физической ноды. В кластерах от 10 нод необходимо всегда выделять отдельные only-master узлы.

Репликация данных

Сейчас каждая запись в нашем индексе существует только в одном месте, и потеря хранящего ее узла приведет к потере данных на неопределенный срок. Для того, чтобы этого избежать существует механизм репликации. Важно не путать понятия реплики и бэкапа, если бэкап позволяет восстановить данные в случае утери, то реплика является полной копией базы.

Если мы потеряем одну из **data-нод**, то всегда сможем продолжить работу с репликами шардов в другом узле и тем временем вернуть потерянный.

То есть для каждого шарда должна быть как минимум одна копия на другой ноде. Можно, конечно, выделять по отдельной машине для каждой реплики, но это очень расточительно. Нужно разместить копии данных на разных узлах, но это не значит, что эти узлы должны хранить только шарды реплик.



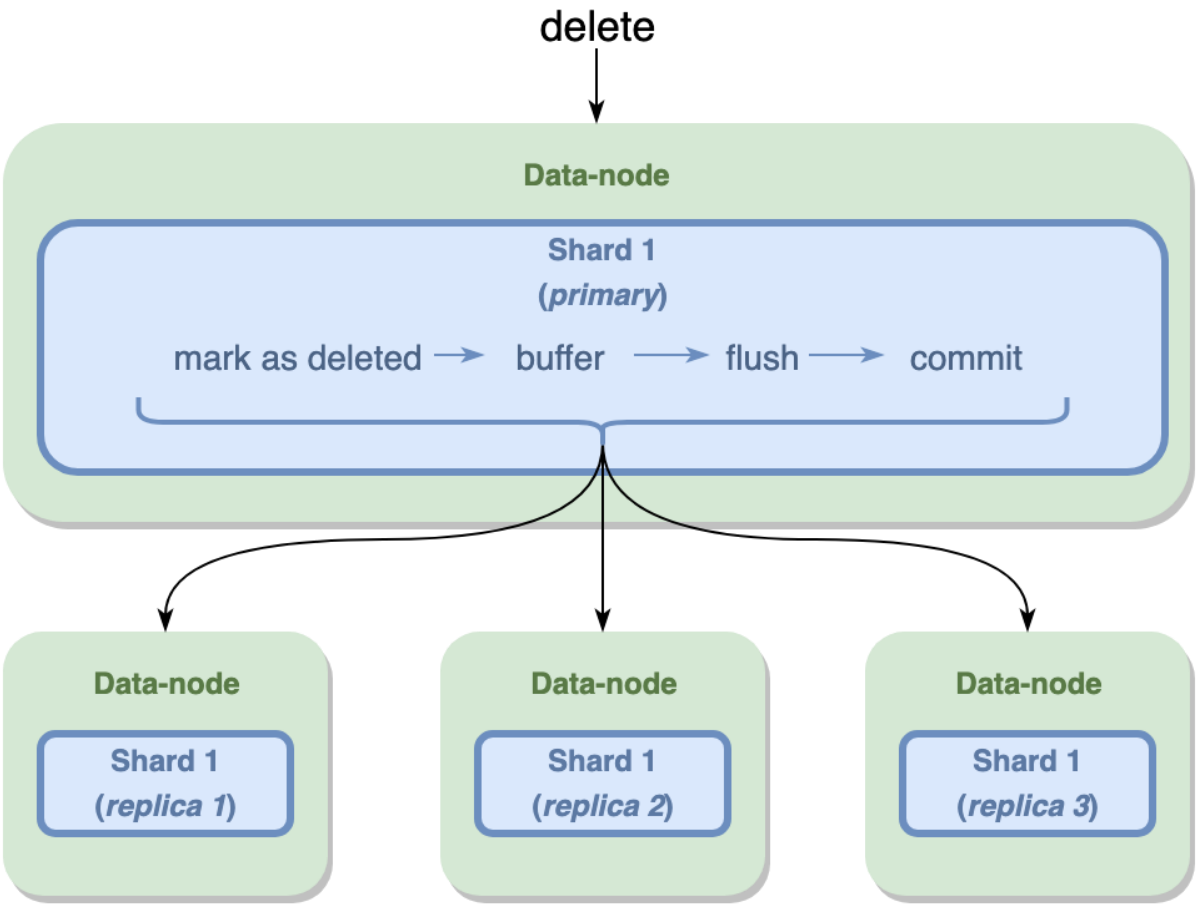
Чтобы жестко установить количество реплик индекса используется параметр number_of_replicas. Так же мы можем изменить это значение в рантайме выполнив запрос:

```
PUT / _settings {
  "index": {
     "number_of_replicas": someVal
  }
}
```

Таким образом, мы всегда имеем реплики всех шардов и не поднимаем неэффективно простаивающие ноды.

Основной шард назовем первичным или **primary shard**, а любую из его копий реплицирующим шардом или **replica shard**, первичный шард и его реплики это группа репликации.

С учетом реплик запись данных будет происходить в два этапа, в первом запись затронет только первичный шард и только после того, как произойдет операция **flush** слияния изменений и операция **commit** фиксации в индексе Lucene, будет отправлен внутренний запрос на изменение всех реплик.



Для мониторинга состояния кластера в **Elasticsearch** существует **cluster health status**. Статус может иметь три значения:

- green — все ок
- yellow — есть утраченные шарды, кластер полностью работоспособен, но едет на репликах
- red — есть утраченные шарды, кластер неработоспособен или часть данных недоступна

Для максимальной стабильности кластера необходимо, чтобы количество дата-нод было больше или равно количества реплик.

Отказоустойчивость

Сейчас данные будут доступны даже в случае сбоя одного из хранящих узлов. Но что если кластер потеряет мастера? Потеря единственного мастера равноценна потере кластера.

Тут все по привычной схеме — поднимаем несколько мастеров.

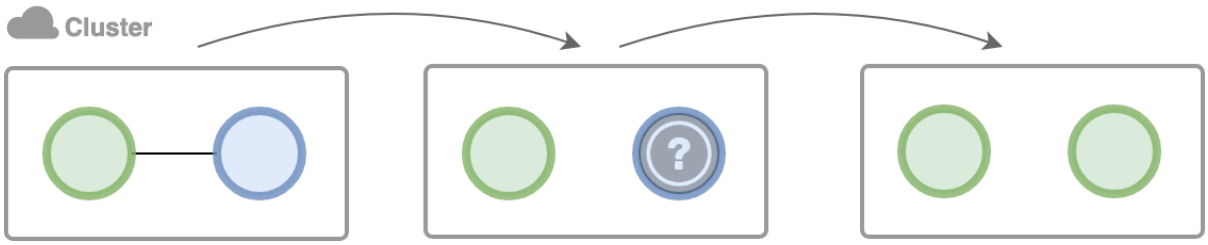
Но если у нас есть, например, два управляющих узла, как понять, какой из них в данный момент должен управлять кластером? Как они смогут договориться о своих решениях? Очевидно, что в каждый момент времени должен быть только один управляющий кластером узел.

То есть при потере мастера его место должен занять один из кандидатов.

В **Elasticsearch** настройка node.master: true не будет означать, что данный узел является мастером, это всего лишь скажет о том, что он может быть выбран в качестве главного узла. По умолчанию настройки будут установлены следующим образом:

- node.master: true
- node.data: true

Представим. Главный управляющий узел стал недоступен для кластера, кластер берет первого кандидата и устанавливает его на вакантное место. Спустя определенное время первый мастер возвращается в кластер и ничего не знает о том, что его место уже занято. Мастер-ноды являются своего рода его *мозгом*, и теперь мозг кластера становится разделен. Это классическая проблема распределенных систем и она так и называется *split-brain problem*.



В обществе подобные проблемы зачастую решаются путем голосования. Подобный механизм используется и в распределенных системах. Как только кластер теряет управляющий узел, должен быть запущен процесс голосования.

Важно определить какой из кандидатов больше всего подходит на роль главного узла. Такой кандидат должен обладать самой актуально информацией о кластере. Для краткого описания актуальности информации о кластере может использоваться версионирование. При каждом изменении кластера главный узел будет обновлять некую служебную информацию и повышать номер версии, далее то же самое будет параллельно происходить в нодах-кандидатах.

Сравнив номера версий мы можем определить наиболее подходящих кандидатов на роль мастера. Теперь если отпавшая мастер-нода вернется в кластер, то процесс голосования запустится снова и будет выбран единственный управляющий узел.

Теперь важно понять когда можно считать, что голосование прошло успешно? Если проголосовали все участники? Или половина? Или другое любое другое магическое количество?

Решение этой проблемы заключается в определении *кворума*. Это умное название для контрольного количества голосующих.

Очевидно, что такое важное решение как выбор мастера должно приниматься на основе большинства, то есть 50%+один голос. Справедливо, надежно. Это значение и станет кворумом.

Таким образом, количество кандидатов на мастера должно быть нечетным и не меньше трех. Рекомендуется использовать простую формулу для расчета оптимально количества таких нод:
КОЛИЧЕСТВО_КАНДИДАТОВ = ОБЩЕЕ_КОЛИЧЕСТВО_НОД/2 + 1

Решения для любых общекластерных действий принимаются путем голосования, и вся необходимая для голосования информация содержится в *конфигурации голосования*. Право голоса определяет еще одну роль, ведь право голоса не означает, что узел может быть кандидатом.

В **Elasticsearch** узлы, которые могут участвовать в голосовании можно определить в конфигурации: node.voting_only: true.

Elasticsearch автоматически изменяет конфигурацию голосования при изменении кластера. Поэтому нельзя одновременно отключать половину или более голосующих нод. Например, если в вашей конфигурации в данный момент 7 голосующих нод и вы отключили сразу 4, кластер станет недоступным, потому что останется 3 ноды, а в конфигурации голосования кворумом является значение 4.

Теперь, если кластер разделится на две части, узлы меньшей, пропинговав доступные в ней узлы и сравнив их количество со значением кворума, будут знать, что именно они отпали от кластера и не могут участвовать в принятии решений.

Транспорт

Пришло время поговорить о том, как общаться с кластером из внешних систем, и как будут общаться узлы внутри кластера. Есть ряд плюсов и минусов

использования и *традиционных*, и специальных протоколов. Для краткого сравнения существует таблица.

Протокол	**Достоинства**	**Недостатки**
HTTP	Низкий порог вхождения, в сравнении с нативным протоколом. Для использования нужен только HTTP клиент и погнали. HTTP API никогда не ломает совместимость, при обновлении версии ES, ваше приложение продолжит работать так же. Возможно проксировать и использовать балансировщики нагрузки. JSON.	Клиент не знает топологию кластера, поэтому может потребовать большее количество запросов для получения данных. Оверхед.
ES Native	Лучший выбор для ОЧЕНЬ больших данных. Если необходимо выполнить большое количество операций с индексом, нативный протокол значительно ускорит.	Используется под JVM. Использование влечет жесткую связность с ES. Обновления требуют перекомпиляции и повторного развертывания пользовательских клиентов. Возможны обновления ломающие совместимость.

Для внутренней коммуникации в кластере **Elsaticsearch** использует нативный протокол.

3.2 Задание на лабораторную работу

На основе имеющейся концептуальной модели провести сравнительный анализ внедрения облачных технологий следующих классов СУБД:

– NoSQL

– SQL DB

– ElasticSearch

Лабораторная работа 4
«Изучение Microsoft Azure»

4.1 Краткие теоретические сведения

Microsoft Azure – это облачная платформа Microsoft с набором вспомогательных служб для вычислений, хранения, данных, работы в сети и поддержания приложений (рис. 3.1).

Azure – это крупная облачная платформа, которая по оценкам Gartner является лидером отрасли по предоставлению решений IaaS и PaaS. Эта полнофункциональная комбинация управляемых и неуправляемых служб позволяет создавать, развертывать приложения и управлять ими любым способом для достижения производительности.

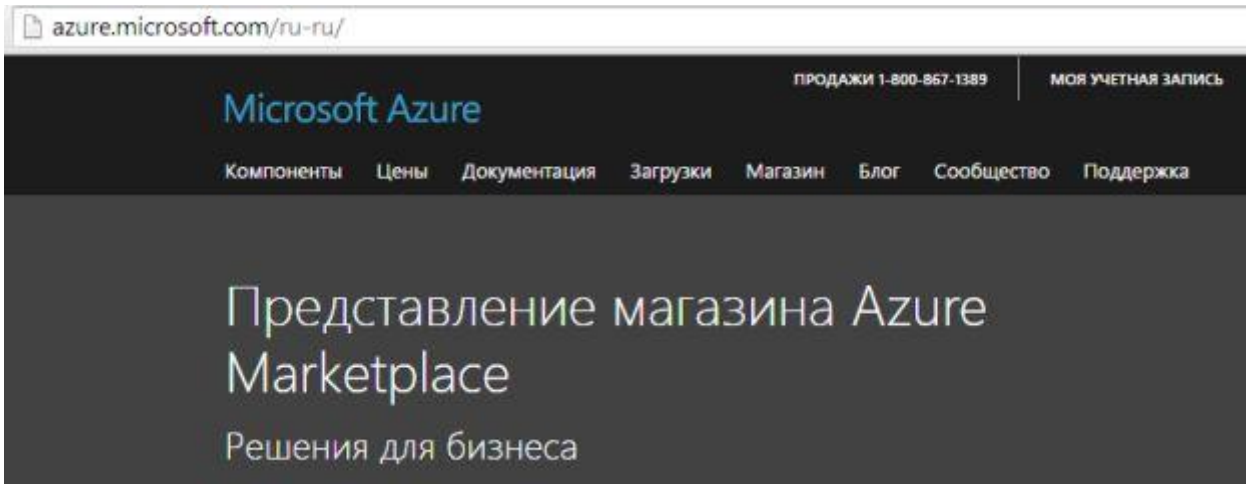


Рис. 3.1. Страница Azure

Возможности Microsoft Azure

1. *Гибридное использование*. Корпоративные гибридные облачные решения Azure позволяют использовать преимущества обоих решений, расширяя возможности ИТ-инфраструктуры. С Azure хранилище данных, резервное копирование и восстановление становятся более эффективными и экономичными.

2. *Открытость и гибкость*. Azure поддерживает любые операционные системы, языки, средства и платформы: от Windows до Linux, от SQL Server до Oracle, от C# до Java. Azure позволяет использовать экосистемы Windows и Linux, чтобы была возможность создавания приложений и служб, работающих на всех устройствах.

3. *Доступность*. Azure позволяет использовать ту же платформу корпоративного уровня, на которой работают Skype, Office 365, Bing и Xbox.

4. *Экономичность и масштабирование*. Azure выполняет масштабирование для соответствия требованиям. Поминутная оплата и обязательство предоставлять конкурентоспособные цены на службы инфраструктуры, например, вычисления, хранилище и пропускную способность, означает получение оптимального соотношения цены и производительности.

5. *Создание инфраструктуры*. В Azure используются виртуальные машины и средства управления.

6. *Разработка современных приложений*. Azure позволяет создавать и развертывать разнообразные и современные приложения для Android, iOS и Windows, использующие все преимущества облачной среды, в том числе веб-, мобильные, мультимедиа- и бизнес-решения.

7. *Получение подробных сведений из данных*. Azure предоставляет управляемые службы данных SQL и NoSQL, а также встроенную поддержку получения подробных сведений из данных. Есть возможность использования SQL Server в облаке и создания кластеров Hadoop в HDInsight для анализа данных.

8. *Управление удостоверениями и доступом*. В Azure можно управлять учетными записями пользователей, выполнять синхронизацию с существующими локальными каталогами, использовать единый вход в Azure, Office 365 и популярных приложений SaaS, таких как Salesforce, DocuSign, Google Apps, Box, Dropbox и др.

Среда выполнения приложений

На рис. 3.2 представлены разделы компоненты "Среды выполнения приложений":

- *Виртуальные машины*. Подготовка виртуальных машин и приложений Windows и Linux к работе.
- *Облачные службы*. Создание облачных приложений и интерфейсов API с высокой доступностью и масштабируемостью.
- *Пакетная служба*. Выполнение крупномасштабных параллельных и пакетных вычислительных заданий.
- *Планировщик*. Запуск заданий по простому или сложному повторяющемуся расписанию.
- *RemoteApp*. Развертывание клиентских приложений Windows в облаке, запуск на любом устройстве.

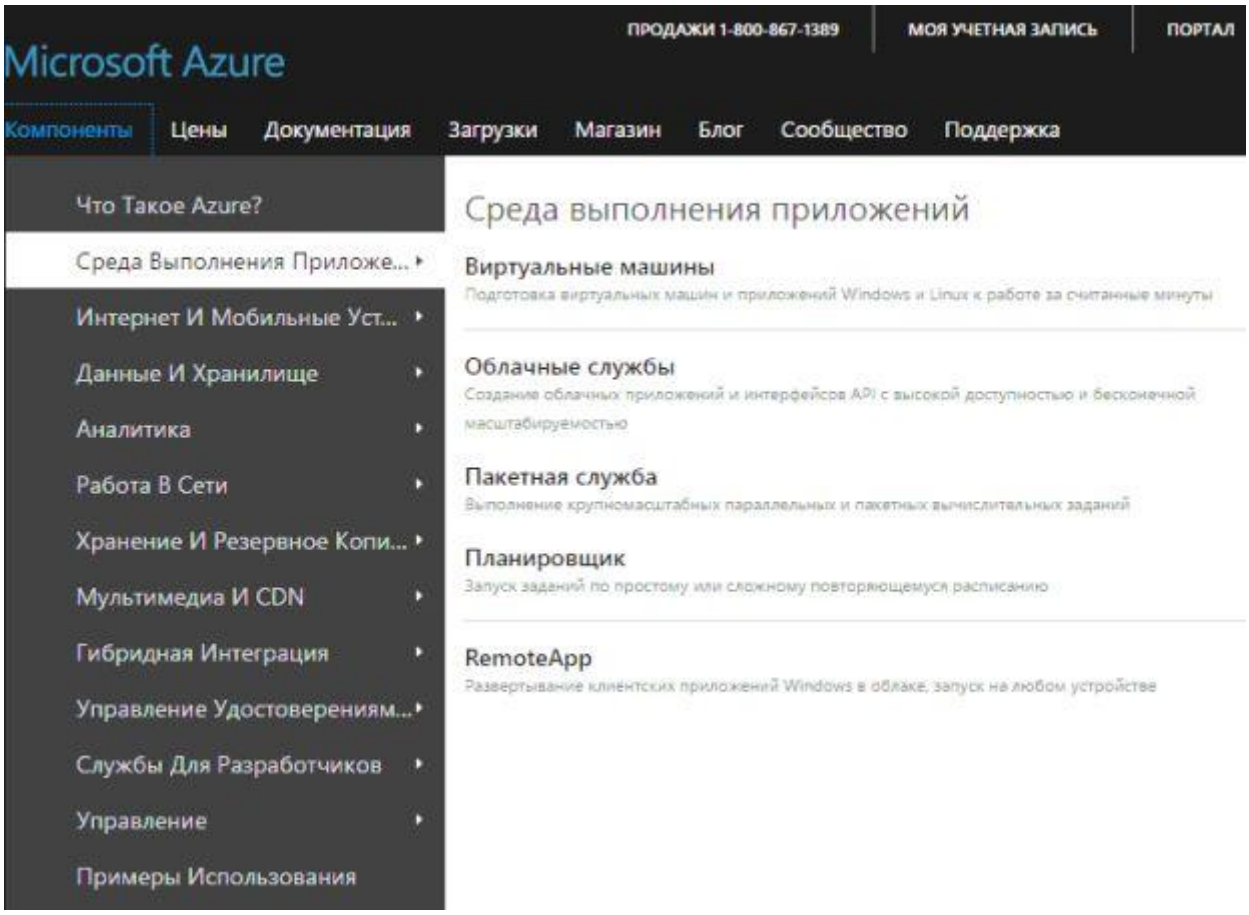


Рис. 3.2. Среда выполнения приложений

Виртуальные машины

SQL Server и SharePoint. С помощью образов, созданных группой специалистов по SQL Server, можно подготовить SQL Server. Можно создавать виртуальные машины, используя бесплатные лицензии MSDN для быстрой разработки и тестирования, или развертывать сложные рабочие приложения, охватывающие множество регионов Azure, используя SQL Server AlwaysOn (рис. 3.3).

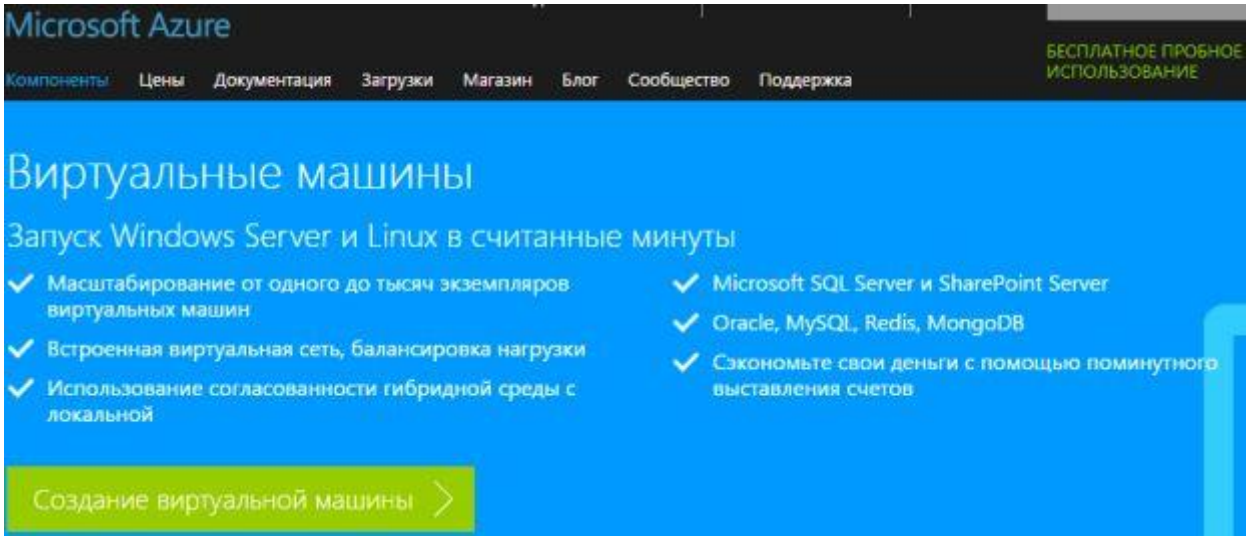


Рис. 3.3. Виртуальные машины

Возможности Azure

Можно развернуть полный диапазон открытых и поддерживаемых сообществами разработчиков ОС и различные программные решения в службе Azure. В службе Azure выбирают полный диапазон дистрибутивов Linux, например, Ubuntu и SUSE, поддерживаемые сообществом решения, например, Chef, Puppet и Docker наряду с другими продуктами, такими как Oracle Database и Oracle WebLogic Server.

Масштабирование приложения. Чтобы использовать эту возможность, а также другие новые возможности Azure, необходимо зарегистрироваться для ознакомления с бесплатной предварительной версией.

На странице "Масштаб" портала управления Azure вручную масштабируют приложение или задают параметры для автоматического масштабирования. Можно масштабировать приложения, которые выполняют веб-роли, рабочие роли или виртуальные машины. Чтобы масштабировать приложение, которое выполняет экземпляры веб-ролей или рабочих ролей, необходимо распределить рабочую нагрузку путем добавления или удаления экземпляров.

При масштабировании приложения, в котором работают виртуальные машины, новые машины не создаются и не удаляются, а включаются или отключаются в группе доступности ранее созданных машин. Масштабирование задают как на основе среднего процента использования центрального процессора, так и на основе количества сообщений в очереди.

При настройке масштабирования приложения следует учитывать следующие сведения.

- Необходимо добавить созданные виртуальные машины в группу доступности, чтобы масштабировать использующее их приложение. Добавляемые виртуальные машины могут быть изначально включены или выключены, однако они будут включены при увеличении масштабирования и выключены при его уменьшении.
- Масштабирование зависит от использования ядер. Более крупные экземпляры ролей или виртуальные машины задействуют больше ядер. Масштабировать приложение можно только в пределах количества ядер по используемой подписке. Например, если подписка ограничена двадцатью ядрами и выполняется приложение с двумя средними по размерам виртуальными машинами (всего четыре ядра), то можно увеличить масштаб других развернутых в рамках подписки облачных

служб на шестнадцать ядер. Все виртуальные машины в группе доступности, используемые при масштабировании приложения, должны быть одного размера.

- Прежде чем приступить к масштабированию приложения на основе порогового значения сообщений, необходимо создать очередь и связать ее с ролью или группой доступности.
- Можно масштабировать ресурсы, привязанные к используемой облачной службе.
- В целях обеспечения высокой доступности приложения необходимо убедиться, что приложение развернуто с двумя и более экземплярами ролей или виртуальных машин.

Действия по масштабированию облачной службы:

- ручное масштабирование приложения, которое выполняет веб-роли или рабочие роли;
- автоматическое масштабирование приложения, выполняющего веб-роли, рабочие роли или виртуальные машины;
- масштабирование связанных ресурсов;
- планирование масштабирования приложения.

Масштабирование вручную приложения, которое выполняет веб-роли или рабочие роли. На странице масштаба вручную увеличивают или уменьшают количество работающих в облачной службе экземпляров.

Количество используемых экземпляров увеличивают только в том случае, если для поддержки этих экземпляров используется достаточное количество ядер. Цвета ползунка представляют используемые и доступные по подписке ядра:

- синий цвет означает ядра, которые используются выбранной ролью;
- темно-серый цвет означает ядра, которые используются всеми ролями и виртуальными машинами в рамках подписки;
- светло-серый цвет означает ядра, доступные для использования в рамках масштабирования;
- розовый цвет означает внесенные изменения, которые еще не были сохранены.

Автоматическое масштабирование приложения, выполняющего веб-роли, рабочие роли или виртуальные машины. На странице "Масштаб" настраивают облачную службу для автоматического увеличения или уменьшения количества экземпляров или виртуальных машин, используемых приложением. Можно настроить масштабирование на основе следующих параметров:

• Среднее использование центрального процессора. Если средний процент использования центрального процессора превышает определенные пороговые значения или опускается ниже этих значений, то будет выполняться создание или удаление экземпляров роли, включение виртуальных машин в группу доступности или их исключение из нее.

• Сообщения в очереди. Если количество сообщений в очереди превышает определенное пороговое значение или опускается ниже его, то экземпляры роли создаются или удаляются или виртуальные машины включаются в группу доступности или исключаются из нее.

Гибридные подключения

С помощью виртуальных сетей можно контролировать и настраивать все аспекты своей работы с сетью, устанавливая подсети и предпочитаемые IP-адреса DNS-серверов. Можно безопасно подключаться к виртуальным машинам в Azure через VPN по Интернету или обойти Интернет для установки прямого подключения посредством ExpressRoute через таких партнеров, как AT&T, Level 3, TelecityGroup, Verizon и Equinix.

Облачные службы

На рис. 3.4 представлены облачные службы.

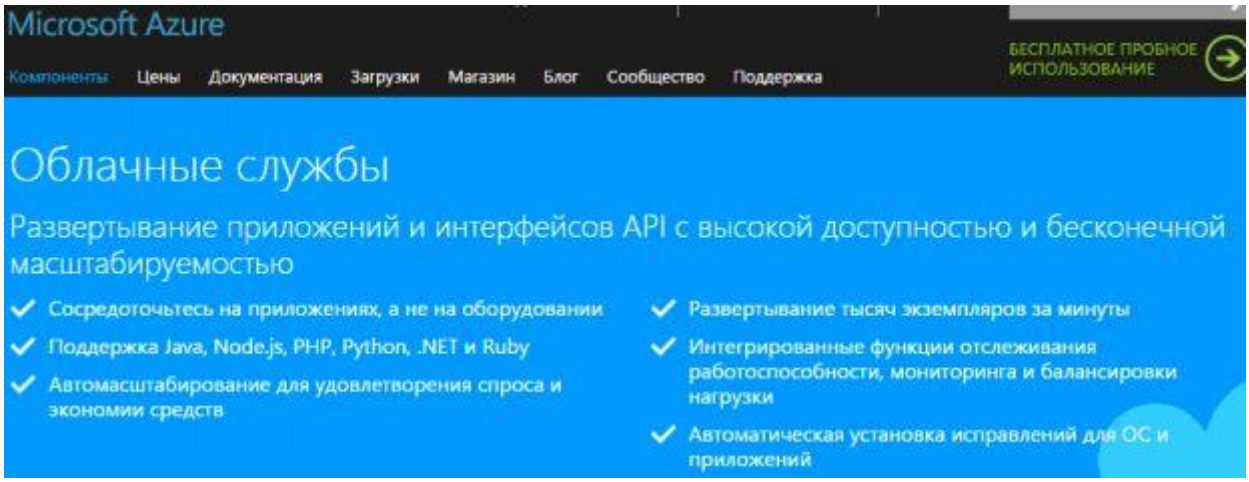


Рис. 3.4. Облачные службы

Создание приложений и интерфейсов API с высокой доступностью и бесконечной масштабируемостью. Есть возможность разрабатывать, упаковывать и развертывать приложения и службы в облаке с помощью облачных служб Azure.

Интегрированный интерфейс разработки, реализованный на базе Visual Studio + Azure SDK. Развертывается с использованием любого языка, включая

.NET, Java, Node.js, PHP, Python или Ruby. Можно проверить приложение перед развертыванием в облаке с использованием эмулятора Azure, который реализует основные функции платформы непосредственно на компьютере.

Создание эффективных приложений. Облачные службы позволяют развертывать приложение и обеспечивать его постоянную доступность во время сбоев и неполадок, перенаправляя трафик из неисправных экземпляров на те, которые работают бесперебойно. Автоматическое обновление ОС подразумевает, что приложение всегда остается защищенным, а простоев и периодов обслуживания удастся избежать.

Тестирование приложений перед развертыванием. Облачные службы предоставляют промежуточную среду для тестирования новых версий (при этом она не оказывает влияния на существующую), что позволяет снизить вероятность нежелательного простоя для клиентов.

Мониторинг работоспособности и оповещения. Azure помогает отслеживать работоспособность и доступность приложений. На панели мониторинга метрик работоспособности отображаются ключевые статистические показатели, которые позволяют оценить работоспособность. Оповещения настраиваются в режиме реального времени для уведомления о снижении степени доступности службы или ухудшении каких-либо других показателей.

Автомасштабирование для оптимизации затрат и производительности. Бесплатная функция автоматического масштабирования помогает контролировать возрастание трафика, так как масштаб автоматически увеличивается или уменьшается в соответствии с требованиями, что одновременно позволяет свести расходы к минимуму.

Пакетная служба

В настоящее время пакетная обработка играет центральную роль в бизнесе, машиностроении, науке и других областях, где требуется выполнение множества автоматизированных задач, таких как обработка счетов и платежных ведомостей, расчет портфельного риска, разработка новых продуктов, мультипликация, тестирование ПО, поиск источников энергии, метеопрогнозирование и создание новых лекарств.

Перенос кластерных приложений в облако. Пакетная служба Azure адаптирована с приложениями, которые используются на рабочих станциях. Можно перенести исполняемые файлы и скрипты в облако, обеспечив горизонтальное

масштабирование среды. Пакетная служба Azure позволяет ставить задания в очередь и выполняет приложения.

Планирование заданий. Основу *пакетной службы Azure* составляет модуль планирования заданий с высоким уровнем масштабирования, доступный как управляемая услуга. Планировщик в приложении используется для распределения заданий. Пакетная служба поддерживает кластерные планировщики заданий, а также может работать на базе службы типа "ПО как услуга".

Масштабирование. Пакетная служба Azure предоставляет доступ к пулу вычислительных виртуальных машин. Она также устанавливает приложения и обеспечивает промежуточное хранение данных, запускает выполнение всех необходимых задач, выявляет сбои и заново организует очередь заданий и сокращает объем ресурсов в пуле после завершения работы.

Предоставление решения как услуги. Пакетная служба Azure позволяет проводить обработку данных по требованию, а не по заранее определенному расписанию. Пользователи могут выполнять задания в облаке, когда это потребуется.

Планировщик

На рис. 3.5 представлен Планировщик.

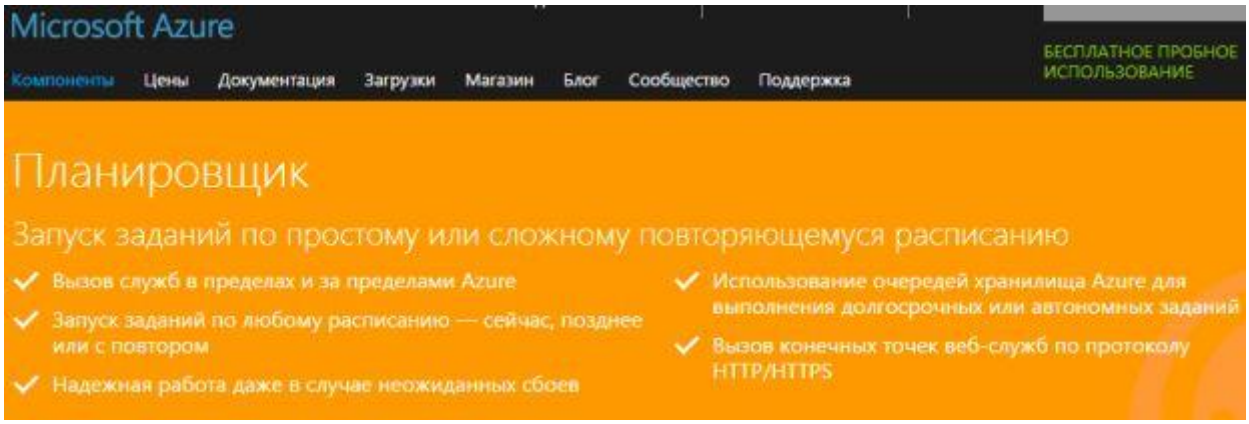


Рис. 3.5. Планировщик

Создание заданий, выполняемых по расписанию. Планировщик Azure позволяет создавать задания в облаке, которые вызывают службы в пределах и за пределами Azure. Например, вызывают конечные точки HTTP/S или публикуют сообщения в очередях хранилища Azure.

Выполнение повторяющихся действий, ежедневных задач по обслуживанию и разработок. Планировщик Azure подходит для выполнения повторяющихся действий. Например, для периодического сбора данных из сети Twitter с отправкой в

веб-канал. Он также подходит для очистки журналов, выполнения плановых операций резервного копирования и других задач по обслуживанию приложений.

Высокая доступность и надежность. Планировщик Azure функционирует и в случае перебоев в работе сети, машин и центров обработки данных, поэтому запланированные задания продолжают выполняться вовремя. При необходимости он может автоматически переключиться на альтернативный центр обработки данных в том же регионе. Пользователи также могут настроить альтернативные конечные точки на тот случай, если основная конечная точка становится недоступной.

Создание асинхронных заданий с помощью очередей. Планировщик Azure может отправлять сообщения в очереди хранилища Azure для асинхронной обработки повторяющихся запросов. Эта функция необходима для выполнения комплексных или долгосрочных запросов (например, серии запросов SQL, отправляемых в большую базу данных), а также в тех случаях, когда необходимо вызвать службу, работающую в автономном режиме.

RemoteApp

RemoteApp – это программы, удаленный доступ к которым можно получить через службы терминалов и которые работают так, как будто они запущены на локальном компьютере пользователя. Пользователи могут запускать программы RemoteApp вместе со своими локальными программами. Пользователи могут сворачивать и разворачивать окно программы, изменять его размеры и с легкостью запускать сразу несколько программ. Если пользователь запускает более одной программы RemoteApp на одном сервере терминалов, программы RemoteApp будут находиться в одном сеансе служб терминалов.

Есть несколько способов запуска программ RemoteApp. Для этого можно:

- дважды щелкнуть файл протокола удаленного рабочего стола (RDP), который был создан и распространен администратором;
- дважды щелкнуть значок программы на рабочем столе или в меню Пуск, которое было создано и распространено администратором с помощью пакета установщика Microsoft Windows (MSI);
- дважды щелкнуть файл, расширение имени которого связано с программой RemoteApp (эта связь может быть настроена администратором с помощью пакета установщика Windows);
- получить доступ к ссылке на программу RemoteApp на веб-узле с помощью веб-доступа к службам терминалов.

RDP-файлы и пакеты установщика Windows содержат параметры, необходимые для запуска программ RemoteApp. После открытия программы RemoteApp на локальном компьютере можно работать с программой, запущенной на сервере терминалов, как если бы она была запущена локально.

Интернет и мобильные устройства

На рис. 3.6 представлены разделы компоненты "Интернет и мобильные устройства":

- *Веб-сайты*. Развертывание и масштабирование веб-приложений.
- *Мобильные службы*. Создание и размещение серверной части для любого мобильного приложения.
- *Управление API*. Безопасная публикация интерфейсов API для разработчиков, партнеров и сотрудников с учетом масштаба.
- *Концентраторы уведомлений*. Масштабируемая кроссплатформенная инфраструктура push-уведомлений.

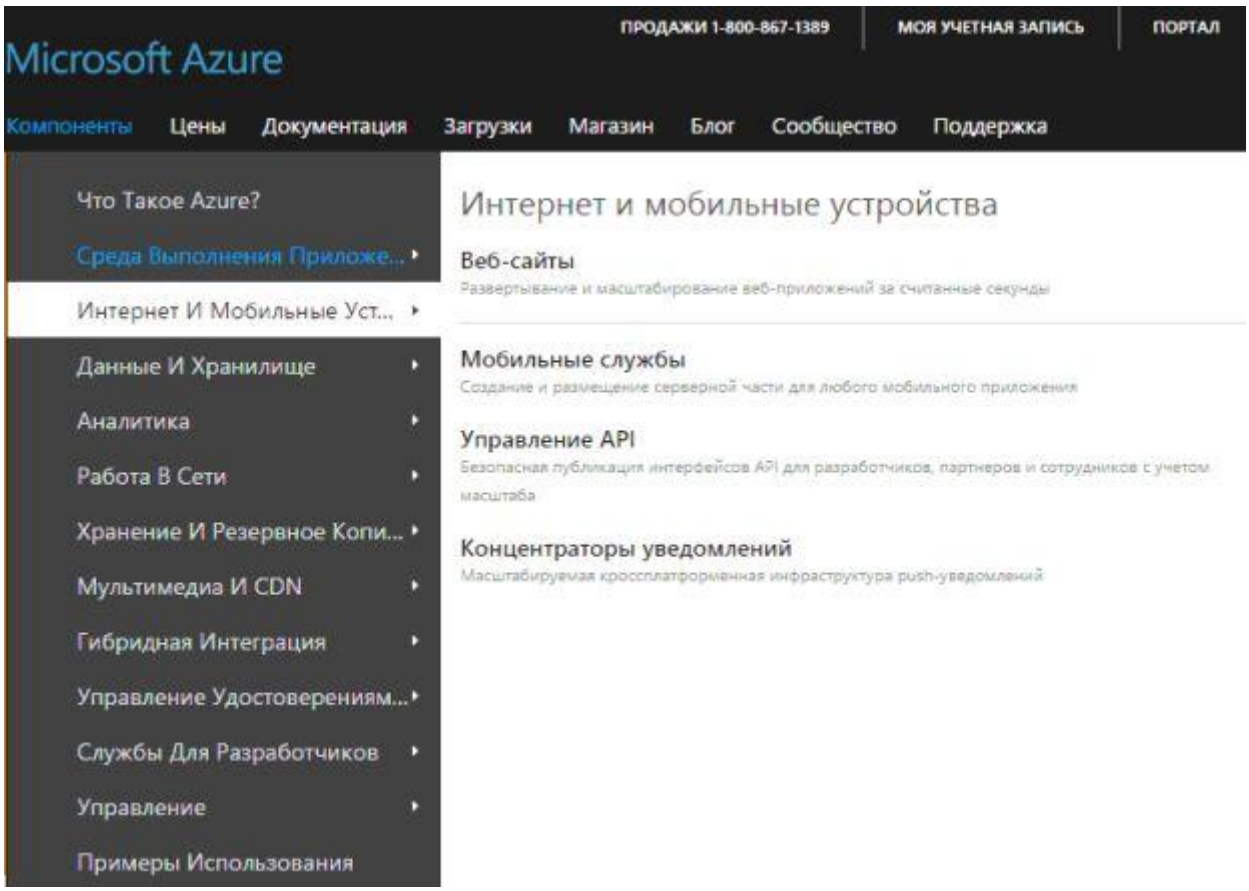


Рис. 3.6. Интернет и мобильные устройства

Веб-сайты

Azure Websites – это управляемая платформа, работающая как служба (PaaS) (рис. 3.7). Она позволяет создавать, развертывать и масштабировать веб-приложения корпоративного уровня.

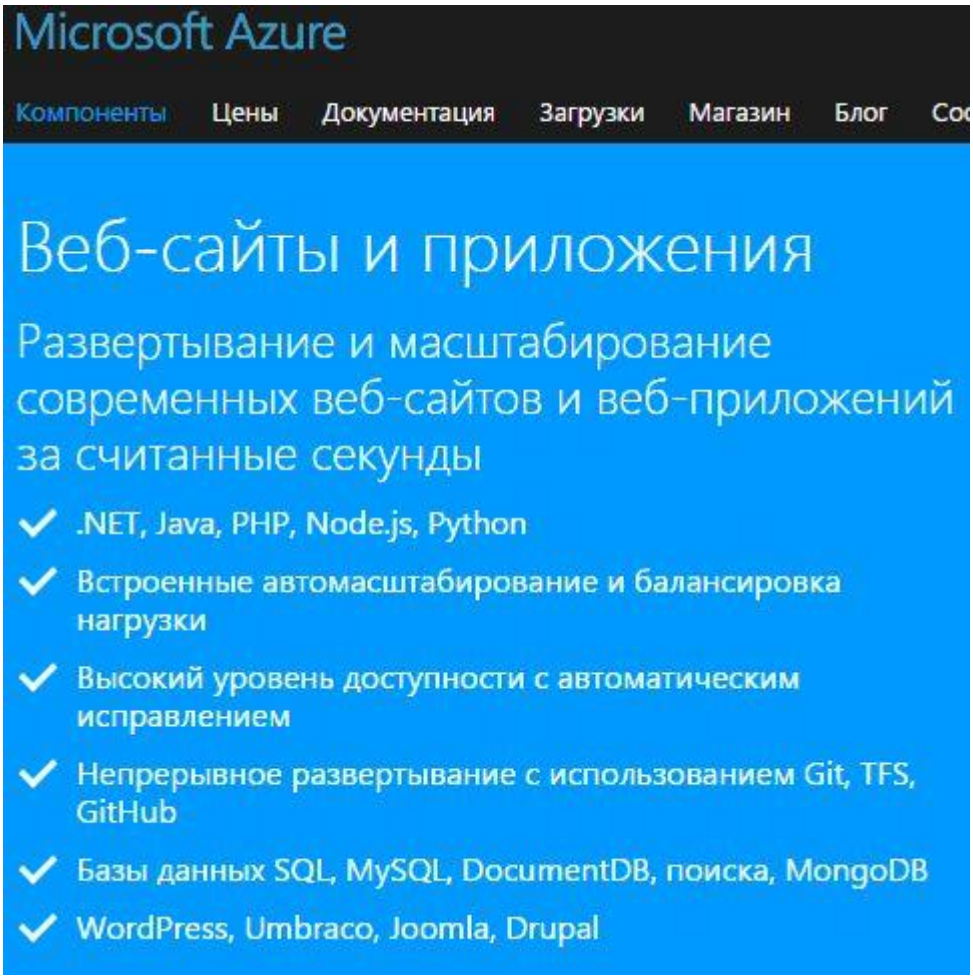


Рис. 3.7. Веб-сайты и приложения

Использование знакомого языка. Использование ASP.NET, Java, PHP, Node.js или Python. Используются популярные веб-приложения и решения CMS, в том числе WordPress, Drupal, Joomla, Umbraco и DotNetNuke.

Azure упрощает создание и использование решений данных с веб-приложениями. Поддерживаются Microsoft SQL, MySQL, Document DB, поиск, MongoDB, Redis и табличное хранилище Azure. Приложения бесплатно тестируются с помощью базы данных SQL и MySQL.

Веб-приложения эксплуатируются в среде с высокой доступностью и автоматическими исправлениями. Веб-приложения, развернутые с помощью Azure Websites, размещаются в виртуальных машинах, выделенных для приложений. Это обеспечивает прогнозируемую производительность и безопасную изоляцию.

Автоматическое определение масштаба. Azure позволяет проводить масштабирование с увеличением или уменьшением для обработки любой входящей пользовательской нагрузки. Можно выбрать количество и размер виртуальных машин вручную или установить автоматическое масштабирование. Это необходимо для масштабирования серверов с учетом нагрузки или по расписанию.

Доступ к локальным данным. Azure позволяет строить веб-сайты, способные подключаться к центру обработки данных. С помощью служб Hybrid Connections и VNET можно получить безопасный доступ к локальным центрам обработки данных. Решение Azure Active Directory необходимо для организации доступа только сотрудникам или партнерам.

Развертывание. Непрерывная интеграция и развертывание настраивается с помощью VSO, GitHub, TeamCity, Hudson или BitBucket. Это позволяет автоматически строить, тестировать и развертывать веб-приложение после каждого успешного теста сборки кода или интеграции.

Интегрированная среда разработки. Интеграция с Visual Studio обеспечивает полное управление жизненным циклом приложения. Благодаря интеграции с Azure VS можно создавать и непрерывно публиковать веб-приложение. С помощью VS можно выполнять отладку в облаке и использовать оперативные данные журнала.

Сохранение активов в безопасности. Автоматическое создание резервной копии для сайта и базы данных. Код и данные защищены на случай непредвиденных обстоятельств, их можно восстановить.

Автоматическое уведомление. Оперативное реагирование на проблемы обеспечивается в реальном времени. Для этого подключаются автоматические уведомления с помощью мониторинга и предупреждений. Интегрируются усовершенствованные функции аналитики и мониторинга New Relic и AppDynamics для получения полного представления о работе веб-приложения.

Фирменная маркировка и защита. Внедряются собственные домены и SSL-сертификаты. Веб-сайты Azure позволяют использовать с веб-приложением любой домен из записей в DNS. Можно использовать пользовательские IP-адреса и SNI на основе SSL, в том числе групповые символы.

Создание сайта на веб-сайтах Azure

Для создания сайта необходимо выбрать "Создать"→"Веб-сайты" (рис. 3.8):

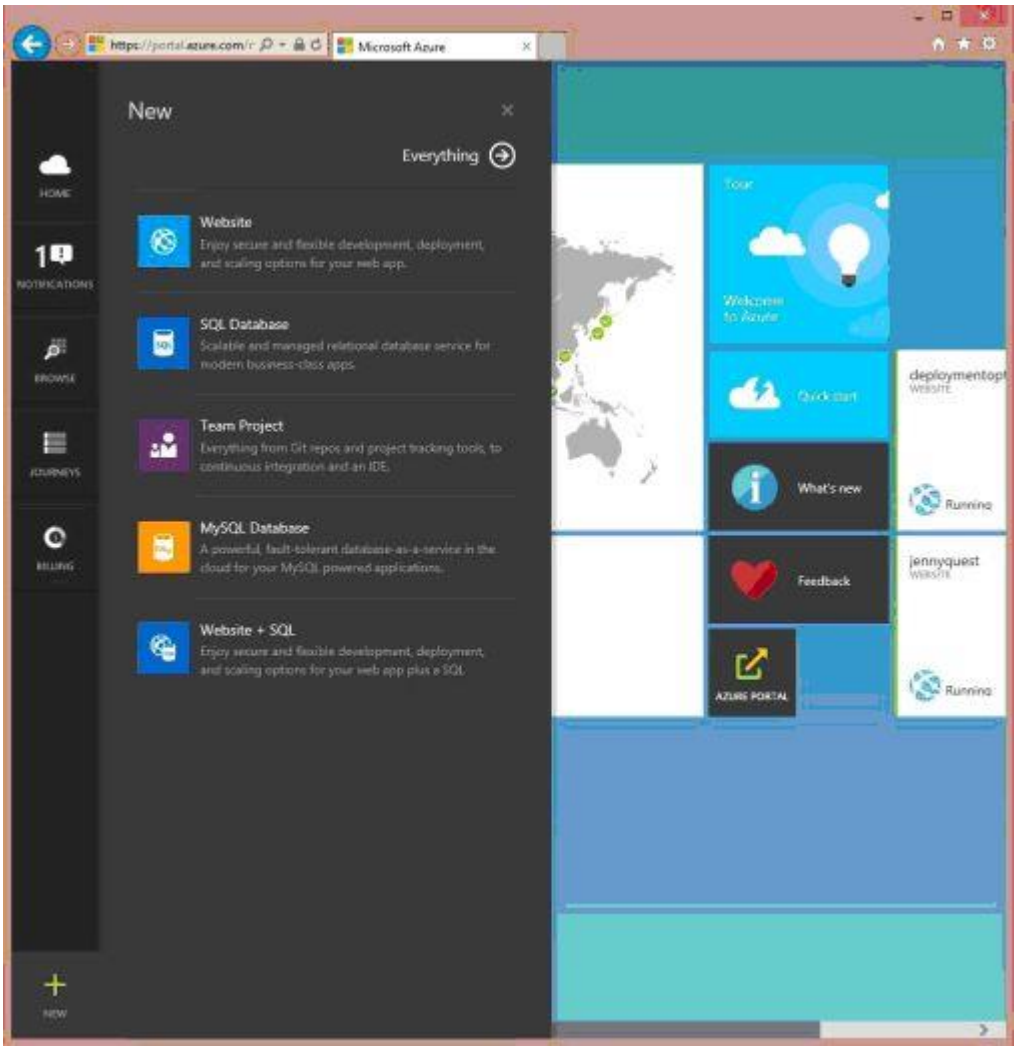


Рис. 3.8. Создание нового веб-сайта

Нужно указать имя создаваемого веб-сайта, выбрать регион мира для его работы, а затем нажать кнопку "Создать веб-сайт".

Управление веб-сайтами Azure. После создания веб-сайта можно воспользоваться страницей предварительного просмотра портала или командной строкой для настройки параметров, масштабирования и мониторинга использования.

Мобильные службы

Создание приложений. Службы Mobile Services позволяют создавать *межплатформенные и собственные приложения* для iOS, Android, Windows или Mac (рис. 3.9). С помощью Mobile Service можно сохранять данные приложения в облаке или локально, авторизовывать пользователей, отправлять push-уведомления. Службы позволяют добавлять пользовательскую логику доступа к базе данных в приложения C# или Node.js.

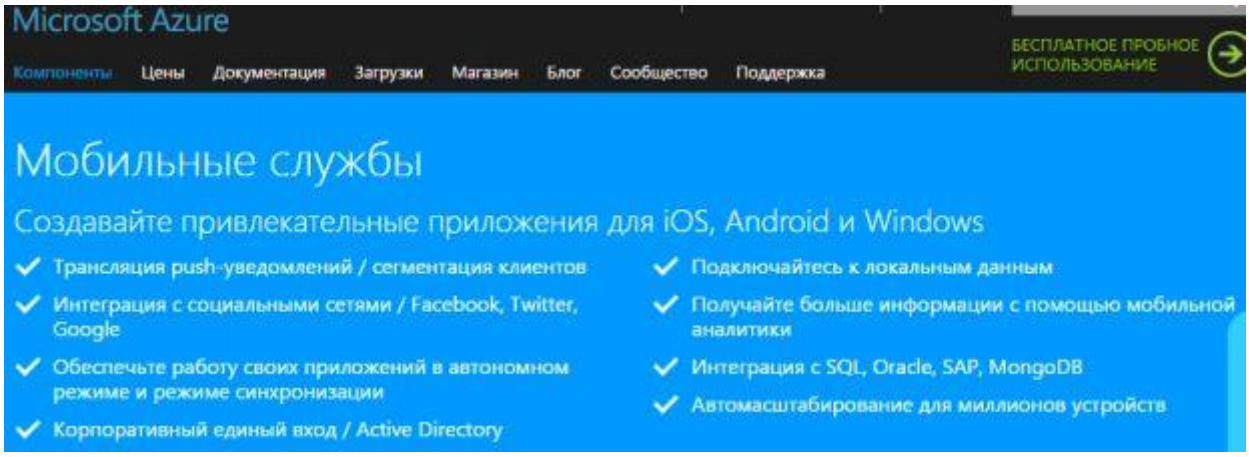


Рис. 3.9. Мобильные службы

Создание корпоративного интерфейса авторизации. С Mobile Services можно авторизовывать пользователей через Active Directory. Благодаря этим службам доступно безопасное подключение к таким локальным ресурсам, как SAP, Oracle, SQL Server и SharePoint. Также возможно использование комплексных платформ Xamarin и PhoneGap для создания приложений корпоративного уровня для своих сотрудников.

Возможности мобильных служб для сотрудников. Мобильные службы упрощают проверку подлинности сотрудников с использованием корпоративных учетных данных, поддерживают использование приложений, подключаемых к сети лишь периодически, а также доступ к данным в локальной среде. Приложения для сотрудников создаются собственными средствами для iOS, Android, Windows.

Ускорители. Можно создавать приложения, демонстрирующие работу ускорителей для наладчиков и продавцов-консультантов.

Корпоративный вход. Проверка подлинности сотрудников осуществляется на основе имеющихся учетных данных, используя Azure Active Directory совместно с синхронизацией каталогов.

Синхронизация автономных данных. Работа не ограничивается лишь теми областями, где имеется доступ к Интернету. Можно сохранять данные локально и синхронизировать их при возобновлении подключения.

Гибридная среда. Доступ к источникам данных, находящимся в корпоративной сети, можно получить с помощью гибридных подключений служб BizTalk.

Автономная синхронизация данных для построения быстро реагирующих приложений. Можно создавать приложения, которые продолжают работать даже при возникновении проблем в сети. Это нужно, чтобы пользователи могли создавать и изменять данные, работая в автономном режиме. Скорость отклика приложения

повышается при помощи локального кэширования данных сервера в устройстве с помощью Mobile Services. Поэтому можно добиться бесперебойной синхронизации данных всех приложений iOS, Android и Windows.

Подключение приложения к локальным данным. Azure позволяет создавать мобильные приложения, с помощью которых можно подключаться к данным своего центра обработки данных. С помощью Hybrid Connections получают безопасный доступ к данным из локальных центров обработки данных в любой точке земного шара.

Создание интерактивных приложений с оперативной передачей сообщений. Azure позволяет строить мобильные приложения, с помощью которых можно подключаться к данным своего центра обработки данных. С помощью Hybrid Connections получают безопасный доступ к данным из локальных центров обработки данных в любой точке земного шара.

Получение информации с помощью мобильной аналитики. Мобильное решение Capptain, необходимое для подключения пользователей, обеспечивает мониторинг в реальном времени. Оно предоставляет глубокий анализ поведения пользователя. Такой анализ отвечает на вопросы, как и когда используется приложение, откуда переходят пользователи, насколько они задерживаются, каковы показатели переходов и сколько покупок делается в приложении.

Анализ, сегментирование. Capptain позволяет владельцам приложений и специалистам по маркетингу подключать пользователей в реальном времени. Также можно анализировать поведение пользователей и сегментировать аудиторию с учетом взаимодействия с приложением. Привлечение пользователей осуществляется с помощью персонализированных push-уведомлений или разнообразных сообщений, предусмотренных в приложении.

Социализация приложения. Службы Mobile Services упрощают проверку подлинности пользователя через учетные записи Facebook, Google, Microsoft или Twitter. После авторизации можно не только настроить функции обслуживания на основе социальных API, но и обеспечить возможность обмена и взаимодействия между пользователями.

PaaS++ для Web API. Расширить службы Mobile Services можно с помощью собственного пользовательского Web API. Группа специалистов Mobile Services будет наблюдать, управлять Web API и устранять нарушения в работе.

Автоматическое масштабирование в соответствии с бизнес-требованиями. Можно настраивать автомасштабирование в соответствии с потребностями

приложения для Mobile Services и для Notification Hubs. Автоматическое масштабирование позволяет развертывать или свертывать ресурсы в зависимости от их фактического использования и оплачивать только то, что используется. Открытый доступ к глобальной сети управляемых центров обработки данных Microsoft позволяет предоставлять данные пользователям в любом месте по всему миру.

Управление API

API используется для создания новых каналов и привлечения клиентов (рис. 3.10).

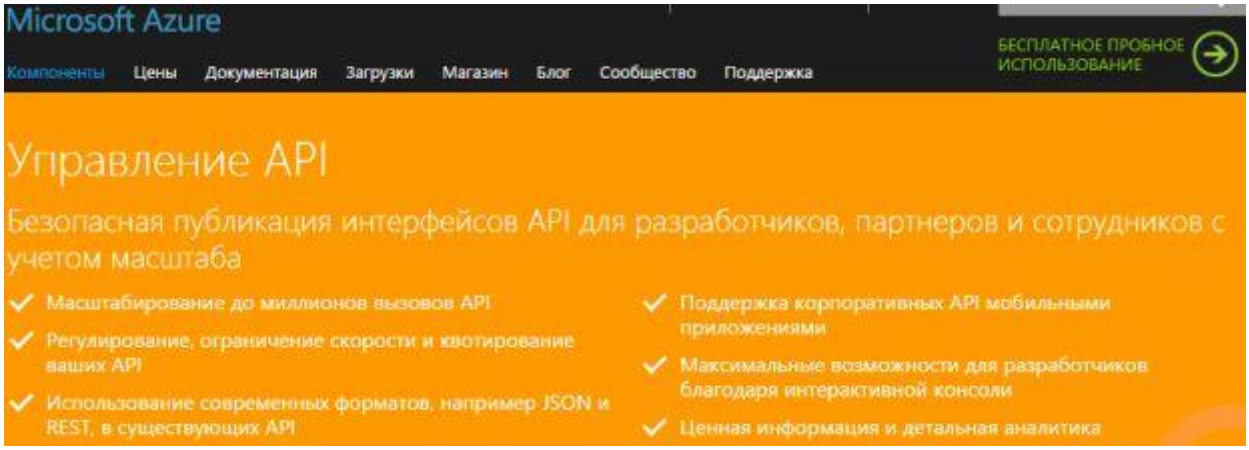


Рис. 3.10. Управление API

Защита критически важных систем. Управление API позволяет защитить критически важные системы, понижая нагрузку с помощью проверки подлинности, ограничения скорости, квотирования и кэширования.

Ускорение внедрения API. Основным фактором успешности любой программы API является простой и беспроблемный доступ для разработчиков. Значение имеет сокращение времени, которое необходимо начинающему разработчику для выполнения транзакции. Благодаря управлению API формируется документация и предоставляется интерактивная консоль, которая незамедлительно повышает успешность разработчика.

Открытые корпоративные системы. Проблему публикации существующих локальных интерфейсов API можно решить, используя мобильные устройства благодаря преобразованию в современные форматы. Можно ориентироваться на новых клиентов с поддержкой CORS и JSONP и оптимизировать производительность с помощью кэширования. Управление API предоставляет готовые инструменты, необходимые организации для сквозного управления.

Преобразование аналитики и отчетности в результаты. Управление API предоставляет возможности аналитики и работы с отчетами, позволяющие организации учитывать наиболее значимые для бизнеса тенденции. Разработчики используют интерфейсы API и данные об использовании приложений, визуализируют работу API, частоту появления ошибок и работоспособность на ближайший период в реальном времени.

Концентраторы уведомлений

Рассылка уведомлений. Notification Hubs – это масштабируемый движок для рассылки мобильных push-уведомлений (рис. 3.11). Он способен рассылать миллионы сообщений на устройства iOS, Android, Windows,Kindle, Nokia X. Концентраторы уведомлений можно связать с любой существующей серверной частью приложения, независимо от того, где размещается эта часть: локально или в Azure.

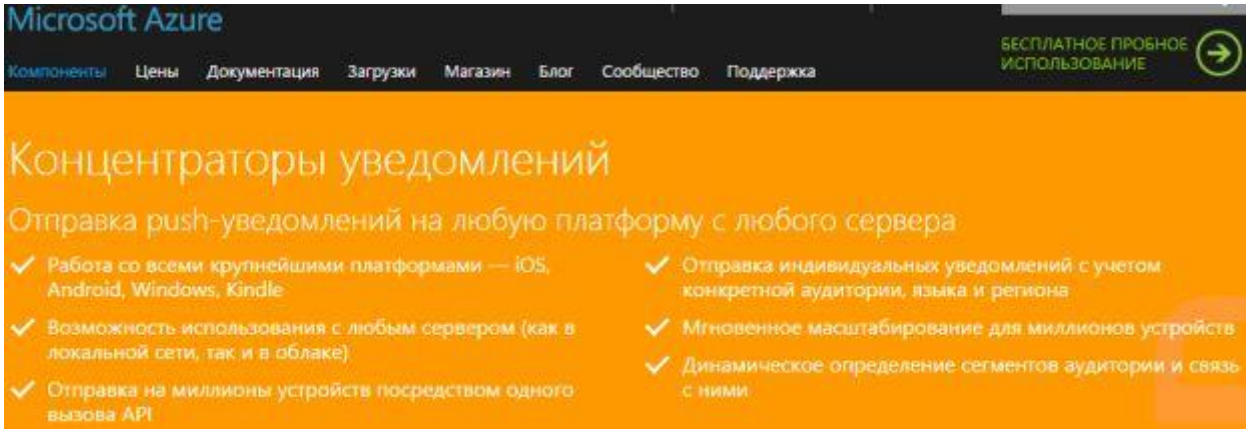


Рис. 3.11. Концентраторы уведомлений

Работа с любым сервером. Служба Notification Hubs способна подключиться к любому интерфейсу: .NET, PHP, Java, Node. Подключение происходит независимо от того, расположен он локально или в облаке Azure. Это позволяет мгновенно обновлять мобильные приложения и привлекать пользователей с учетом их запросов.

Определение аудитории с помощью динамичных тегов. Широковещательную рассылку уведомлений проводят с помощью функции разметки Notification Hubs. Такая разметка позволяет выбрать целевую аудиторию с учетом активности, интересов, местоположения или предпочтений. Нужный контент будет доставлен определенному человеку в заданное время.

Упрощение локализации с помощью шаблонов. Если приложение нацелено на различные рынки, функция шаблонов Notification Hubs предоставляет удобный

способ отправки локализованных push-уведомлений. Это дает возможность говорить с клиентами на их языке. Шаблоны устраняют сложности, связанные с сохранением параметров локализации для каждого клиента или созданием сотен разметок.

Масштабирование. Концентратор уведомлений автоматически создает инфраструктуру, необходимую для масштабирования сообщений с учетом каждого активного устройства.

Данные и хранилище

На рис. 3.12 представлены разделы компоненты "Данные и хранилища":

- *База данных SQL* . Управляемая реляционная база данных SQL как служба.
- *DocumentDB*. Управляемая база данных документов NoSQL, предоставляемая в виде услуги.
- *Кэш Redis*. Высокая производительность, доступ к данным с малым временем задержки для создания быстрых и масштабируемых приложений.
- *Хранилище*. Надежное, высокодоступное и высокомасштабируемое облачное хранилище.
- *StorSimple*. Гибридное облачное хранилище для предприятий.
- *Поиск Azure*. Полностью управляемая функция поиска, предоставляемая как услуга.

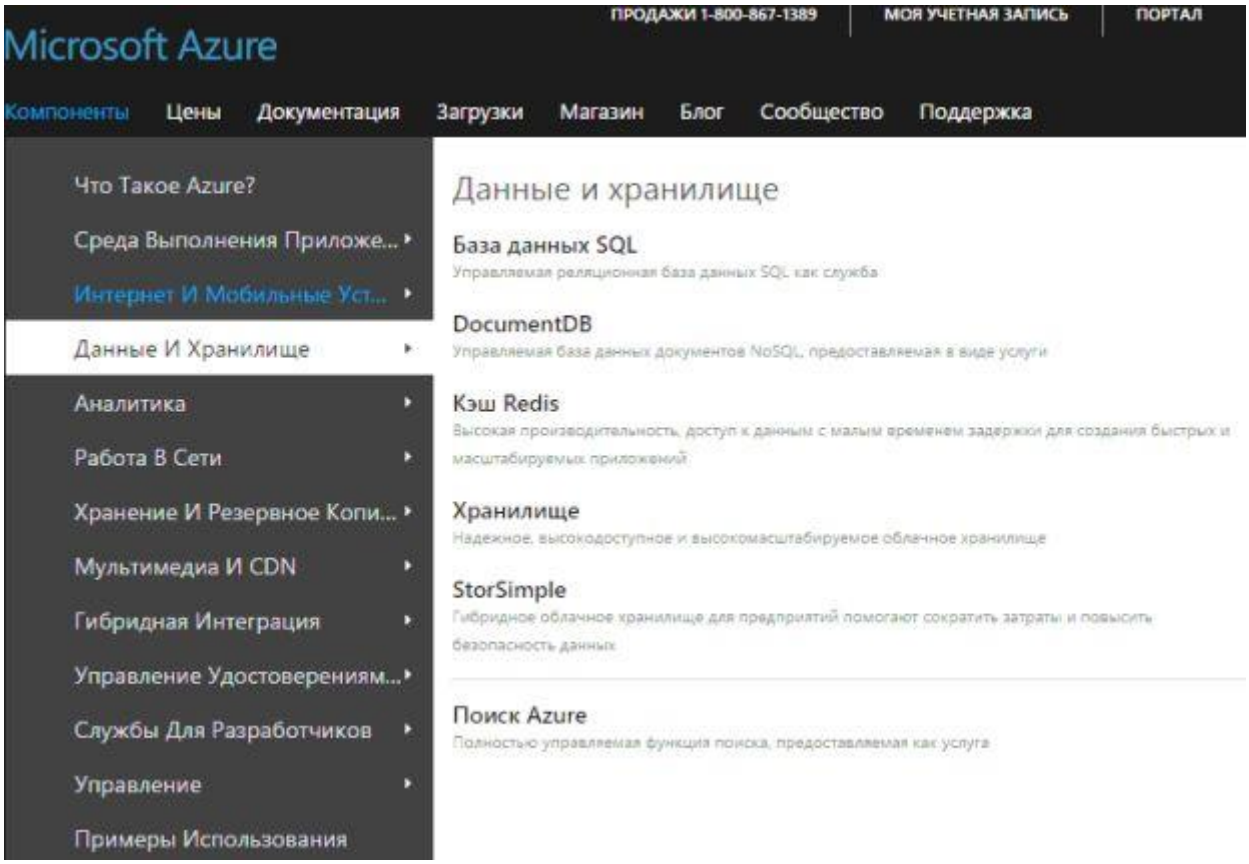


Рис. 3.12. Данные и хранилище

База данных SQL

Поддержка гибкого масштабирования и более предсказуемая производительность. За счет возможностей Azure высокопроизводительные приложения выполняют масштабирование уровней данных в тысячах баз данных. Новые уровни служб базы данных SQL предоставляют приложениям возможность увеличивать или уменьшать масштаб предсказуемой производительности в каждой базе данных (рис. 3.13). Средство гибкого масштабирования для базы данных SQL Azure упрощает разработку приложений, горизонтально масштабируемых по значительному количеству баз данных, и управление ими.

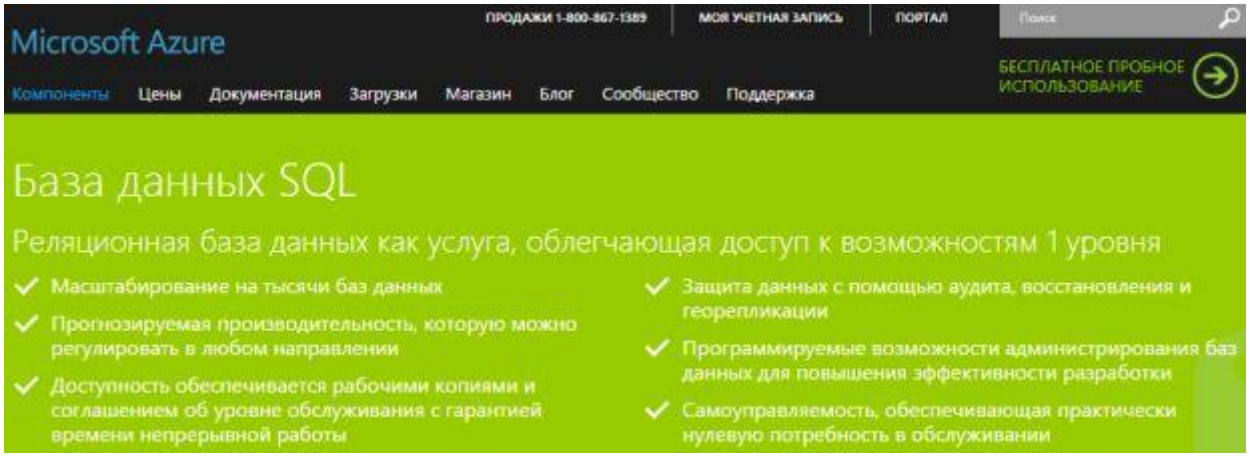


Рис. 3.13. ДБаза данных SQL

Непрерывность деловой активности для критически важных приложений. Повышается уровень непрерывности работы приложений и защиты от аварий с помощью встроенных функций непрерывной работы, действующих на всех уровнях служб. База данных SQL Premium обеспечивает активную георепликацию, которая позволяет создать до четырех считываемых вторичных баз данных в любом регионе Azure и управлять временем и конечным назначением отработки отказа. Возможности управления расширяются непредвиденным восстановлением с помощью функции самостоятельного восстановления. Она предоставляет управление восстановлением данных из доступных резервных копий.

Обеспечение непрерывности работы базы данных SQL Azure. Непрерывность бизнеса определяет возможность организации продолжать коммерческую деятельность в случае критической ситуации или аварии. Для планирования непрерывности бизнеса в организации требуется внедрить процессы, процедуры и меры, обеспечивающие бесперебойность деловых операций.

Нулевая потребность в обслуживании за счет самоуправляемой службы. Потребность в обслуживании инфраструктуры уменьшается за счет базы данных SQL в качестве службы с автоматическим исправлением программных ошибок. При этом встроенные системные реплики изначально обеспечивают защиту данных, бесперебойную работу базы данных и стабильность системы, снижая нагрузку на разработчиков и архитекторов. Системные реплики автоматически перемещаются на новые машины, которые подготавливаются к работе в динамическом режиме по мере выхода старых машин из строя.

Компания Pottermore переместила свои приложения на платформу Azure как службу, чтобы не тратить ресурсы на обслуживание виртуальных машин и управление ими.

Повышение производительности труда за счет поддержки привычных средств и платформ. Поддерживаются задачи разработки, такие как горизонтальное масштабирование и непрерывность бизнеса, посредством программных API-интерфейсов для оптимизации управления масштабированием до сотен и тысяч баз данных. База данных SQL предоставляет разнообразные средства управления – REST API, PowerShell, портал управления Azure с поддержкой HTML5 или SQL Server Management Studio – и поддерживает различные популярные платформы и технологии, включая .NET, Java, PHP, Ruby on Rails и Node.js. Есть возможность

автономной и сетевой разработки локальных и облачных приложений благодаря интеграции с Visual Studio.

Обеспечение выполнения задач, связанных с безопасностью и соблюдением требований. Упрощаются задачи, связанные с соблюдением нормативов. Можно узнать, какие операции выполняются в базе данных, с помощью средств аудита, которые отслеживают события базы данных и записывают их в журнал. Представления информационной панели и отчеты Excel Power View предоставляют подробные сведения о событиях в базе данных и теоретически отображают проблемы бизнес-приложений или нарушения безопасности. База данных SQL проверяется крупнейшими облачными аудиторами в рамках нормативной сертификации Azure, такой как ISO/IEC 27001:2005 и т.д.

Совмещение уровней служб для получения инновационных результатов. База данных SQL предоставляется на разных уровнях служб для поддержки всего спектра рабочих нагрузок баз данных. Таким образом, можно перемещаться по уровням служб или совмещать их для создания инновационных концепций приложений. Благодаря производительности и охвату платформы Azure можно сопоставлять и совмещать службы Azure с базой данных SQL для удовлетворения уникальных потребностей при разработке современных приложений, снижения затрат, повышения эффективности использования ресурсов и открытия новых коммерческих возможностей.

DocumentDB

Azure DocumentDB представляет собой службу базы данных документов NoSQL, разработанную специально для реализации прямой поддержки JSON и JavaScript внутри системы базы данных. Это решение подходит для веб-приложений и мобильных приложений, при работе с которыми требуется обеспечить предсказуемую полосу пропускания, низкий уровень задержек и гибкие возможности работы с запросами. В приложениях Microsoft для потребителей, таких как OneNote, используется DocumentDB, что позволяет работать с миллионами пользователей.

Широкие возможности запросов и транзакций при работе с данными JSON. Схемы приложений постоянно изменяются, и это общая проблема, с которой сталкиваются многие разработчики. DocumentDB автоматически индексирует все документы JSON, добавляемые в базу данных, и затем позволяет с помощью обычного языка SQL запрашивать данные без указания схемы или вторичных индексов.

Сочетание широких возможностей формирования запросов и транзакционной обработки данных позволяет создавать масштабируемые мобильные и веб-приложения, отвечающие современным требованиям. Поддержка использования в запросах пользовательских операторов и заданных пользователем функций (UDF) предоставляет больше преимуществ при работе с DocumentDB. Собственная модель данных JSON делает возможной интеграцию с интернет-платформами и средствами.

Обеспечение стабильного уровня производительности с возможностью настройки. Служба DocumentDB имеет облачную природу и работает со сверхбыстрыми SSD-накопителями, обладающими малым временем задержки и оптимизированными для операций записи. Высокая предсказуемая производительность и зарезервированные ресурсы позволяют обеспечить выполнение требований, предъявляемых к пропускной способности. По мере роста требований приложений хранилище и пропускная способность могут масштабироваться с пропорциональным изменением стоимости благодаря комбинируемым единицам мощности. Предусмотрена возможность настройки и подбора оптимального уровня согласованности с определенными уровнями (высокий, с ограниченной задержкой, сеансовый и пассивный) для соответствия сценариям приложений и требованиям к производительности. Это позволяет избежать необходимости выбора между двумя противоположными уровнями – высоким и пассивным. Выполняется автоматическая репликация данных, благодаря чему поддерживается высокий уровень доступности.

Предоставление возможности быстрой разработки. Доступ к базам данных через CRUD, запросы и обработка JavaScript в HTTP-интерфейсе RESTful упрощает процесс построения новых приложений для бизнеса. Программирование для DocumentDB характеризуется простотой, гибкостью и доступностью и не требует написания отдельного кода или расширений JSON или JavaScript.

Кэш Redis

Оптимизация приложения с помощью КЭШа. Кэш Azure обеспечивает быстрое реагирование приложения, высокую пропускную способность и доступ к данным с минимальной задержкой.

Кэш Redis. Кэш Redis для Microsoft Azure основан на кэше с открытым исходным кодом, Redis. Он предоставляет доступ к безопасному выделенному кэшу Redis, управляемому корпорацией Microsoft. Кэш, созданный с помощью Azure Redis, доступен из любых приложений в Microsoft Azure.

Кэш Redis для Microsoft Azure предоставляется на двух уровнях:

Basic – один узел. Несколько размеров.

Standard – два узла, ведущий/ведомый. Предоставляется соглашение об уровне обслуживания и поддерживается репликация. Несколько размеров.

Доступен кэш размером до 53 ГБ.

Высокая производительность. Кэш Redis для Azure помогает приложению работать быстрее, даже если пользовательская нагрузка увеличивается, и использует скоростные и производительные возможности модуля Redis. Отдельный распределенный слой кэша позволяет независимо масштабировать уровень данных для эффективного использования вычислительных ресурсов на слое приложений.

Redis. Redis – это усовершенствованное хранилище значений ключей, где ключи могут содержать такие структуры данных, как строки, хэши, списки, наборы и сортируемые наборы. Redis поддерживает ряд атомарных операций с этими типами данных.

Redis поддерживает репликацию "ведущий-ведомый" с быстрой начальной синхронизацией без блокировки, автоматическим повторным подключением при разделении сети и т.д.

К другим возможностям относятся транзакции, публикация/подписки, скрипты Lua, ключи с ограниченным сроком жизни и параметры конфигурации, позволяющие Redis действовать как кэш.

С Redis работают, используя большинство современных языков программирования.

Кэш Azure Redis использует проверку подлинности Redis и поддерживает SSL-подключения к Redis.

Удобство использования и управления. Кэш Redis для Azure прост в использовании. Необходимо подготовить кэш на новом портале управления Azure и использовать вызов к конечной точке в любом клиенте, поддерживающем Redis.

Кэш Redis для Azure прост в управлении. Можно отслеживать состояние и работу кэша с помощью нового портала управления Azure. Корпорация Microsoft может управлять репликацией кэша, повышая доступность данных при сбое кэша.

Хранилище

BLOB-объекты, таблицы, очереди и файлы. Хранилище Azure предоставляет гибкие возможности для хранения и извлечения крупных объемов неструктурированных данных, например, документов и файлов мультимедиа, в BLOB-объектах Azure, слабо структурированных данных в таблицах Azure, надежных сообщений в

очередях Azure, а также использовать SMB-хранилище файлов Azure для переноса локальных приложений в облако (рис. 3.14).

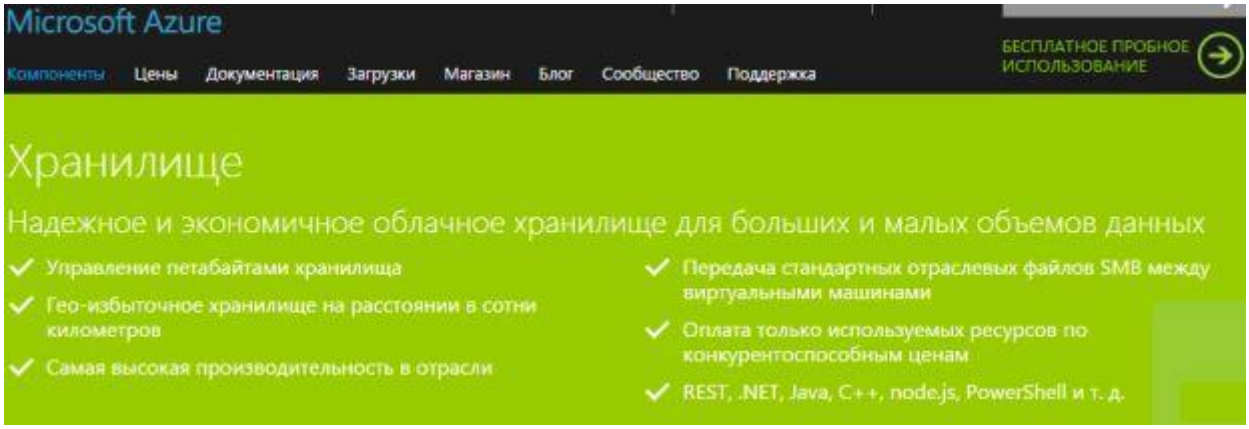


Рис. 3.14. Хранилище

Высокая масштабируемость. Хранилище Azure адаптируется к возрастающим требованиям в отношении объемов данных, выделяя до 500 ТБ общего пространства для хранения для каждой учетной записи.

Надежность и высокая доступность. Хранилище Azure автоматически реплицирует данные для защиты от неожиданных сбоев оборудования и предоставления доступа к ним в случае необходимости.

Создано для разработчиков. Можно создавать приложения с поддержкой клиентских библиотек для .NET, Java, Android, C++ и Node.js. Доступ к данным в хранилище Azure также можно получить, используя REST API, который вызывают с использованием любого языка, позволяющего осуществлять запросы HTTP/HTTPS. Хранилище Azure подразумевает гарантированную согласованность, упрощая разработку облачных приложений и обеспечивая прогнозируемую производительность приложений, основанных на Azure.

Глобальный доступ. Осуществляется горизонтальное или вертикальное масштабирование центров обработки данных по мере необходимости и данные размещаются географически ближе к своим клиентам для более быстрого доступа и повышения производительности.

Экономичность. Плата только за то, что используется, по цене, более низкой, чем у локальных вариантов хранения.

Создание учетной записи хранилища. Создание учетной записи с помощью портала управления (рис. 3.15).

1. Необходимо щелкнуть "Создать", "Хранилище", "Быстрое создание".

Рис. 3.15. Создание учетной записи с помощью портала управления

2. В области URL-адреса нужно ввести имя поддомена.

3. В области "Расположение/родственная организация" нужно выбрать регион.

4. Необходимо выбрать функцию репликации данных и щелкнуть "Создать учетную запись хранилища".

5.

StorSimple

Управление данными. StorSimple позволяет автоматизировать работу и исключить рост объемов данных на два порядка и связанные с этим проблемы управления. В StorSimple используются SSD-накопители и жесткие диски, обеспечивающие высокую скорость ввода-вывода без существенных затрат, а также предоставляется встроенная функция удаления дубликатов и сжатия с целью сокращения общего объема данных. Данный продукт предоставляет широкие возможности масштабирования инфраструктуры хранения данных посредством использования Azure для сохранения быстро растущего объема неактивных первичных данных, зачастую приводящего к постоянному приобретению новых мощностей для хранения данных и слишком громоздкой инфраструктуре (рис. 3.16).

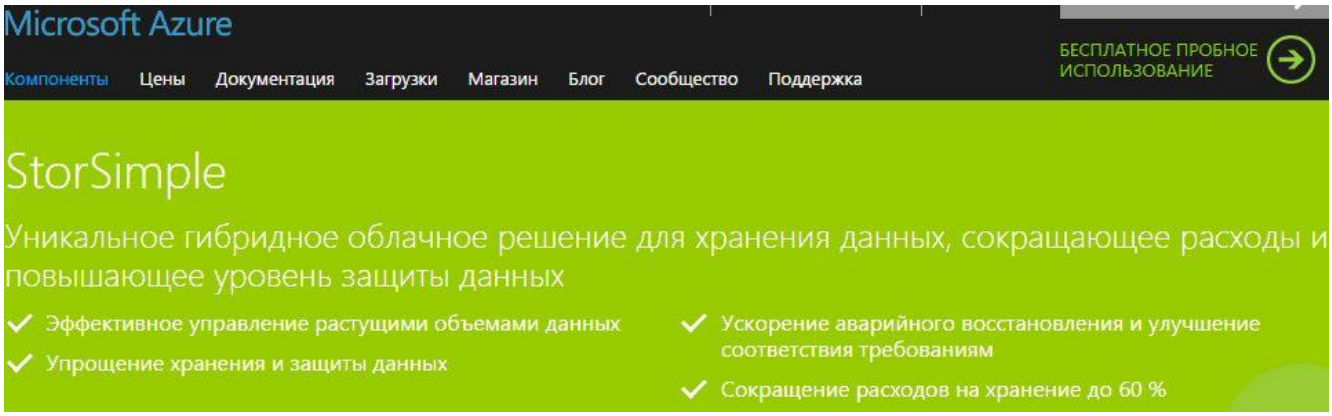


Рис. 3.16. StorSimple

Упрощение хранения и защиты данных. StorSimple с помощью Azure позволяет автоматически расширять мощности и выполнять внешнее резервирование данных, поэтому сотрудники отделов ИТ могут тратить меньше времени на добавление мощностей, обслуживание инфраструктуры и управление защитой данных. Данное гибридное облачное решение объединяет в себе первичное, резервное, архивное и внешнее хранилище данных с автоматизированным созданием снимков, что заменяет дорогостоящую удаленную репликацию и управление ленточными накопителями.

Ускорение аварийного восстановления, улучшение соответствия требованиям. StorSimple обеспечивает быстрое аварийное восстановление посредством загрузки только данных, непосредственно необходимых приложениям. Продукт позволяет клиентам тестировать восстановление данных и обеспечивать соответствие политикам компании без нарушения работы центров обработки данных. При использовании StorSimple сохраняемые данные определяются программно на уровне политик, а не ограничиваются емкостью системы резервного копирования или ленточных накопителей.

Семейства продуктов

Хранилище StorSimple серии 8000. Microsoft Azure StorSimple – это предложение от корпорации Microsoft для хранения данных в облаке, реализованное на основе гибридных массивов хранения данных StorSimple 8000. Эти массивы хранения обеспечивают более высокую производительность и интеграцию с Azure. Массивы StorSimple 8600 поставляются в двух вариантах, отвечая различным требованиям к емкости и производительности: StorSimple 8100 and StorSimple 8600. Виртуальный модуль StorSimple предоставляет доступ по требованию к данным предприятия в среде Azure, что позволяет клиентам осуществлять поиск и анализ исторических

наборов данных, осуществлять разработку и тестирование, а также аварийное восстановление в Azure. С помощью диспетчера StorSimple клиенты могут централизованно настраивать все параметры хранилища StorSimple и управления данными из облака, что позволяет обеспечить надлежащее выполнение операций и принудительное применение политик защиты и хранения данных на всем предприятии в целом.

StorSimple 5000 и 7000. StorSimple также предоставляет свои ведущие гибридные облачные решения для хранения данных – серии StorSimple 5000 и 7000. Клиенты получают все преимущества консолидации хранилищ, возможность управлять ростом объемов данных, упрощенные методы защиты данных и сокращение расходов за счет использования облака, как и в случае с серией 8000, но в конфигурации с меньшей мощностью и с тем исключением, что серии 5000 и 7000 не поддерживают диспетчер StorSimple и виртуальный модуль StorSimple.

Поиск Azure

Поиск Azure позволяет реализовать полнофункциональные возможности поиска на веб-сайте или в приложении. Настройка результатов поиска, а также создание полноценных адаптированных моделей ранжирования позволяет привязать результаты поиска к целям бизнеса. Стабильная пропускная способность и надежное хранение данных обеспечивают высокую скорость индексирования поиска и выполнения запросов, что незаменимо в сценариях с ограниченными временными рамками.

Упрощение работы. Поиск Azure исключает сложности, связанные с настройкой собственного поискового индекса и управлением этим индексом. Полностью управляемая служба предотвращает проблемы, связанные с повреждением индекса, доступностью службы, а также ее масштабированием и обновлением. Можно создать несколько индексов без увеличения стоимости.

Поиск Azure ускоряет разработку благодаря поддержке привычных средств и согласованной глобальной облачной платформе. Аналогично другим службам Azure, в службе поиска используются вызовы API REST. Всемирная сеть центров обработки данных Azure позволяет сократить длительность задержек при поиске, независимо от расположения приложения.

Аналитика

Рассмотрим следующие разделы компоненты "Аналитика" (рис. 3.17):

- *HDInsight*. Подготовка управляемых кластеров Hadoop.

• *Машинное обучение*. Облачная прогнозная аналитика.

• *Stream Analytics*. Потоковая обработка в режиме реального времени.

• *Фабрика данных*. Координация и администрирование преобразования и перемещения данных.

• *Концентраторы событий*. Получение, сохранение и обработка миллионов событий в секунду.

•

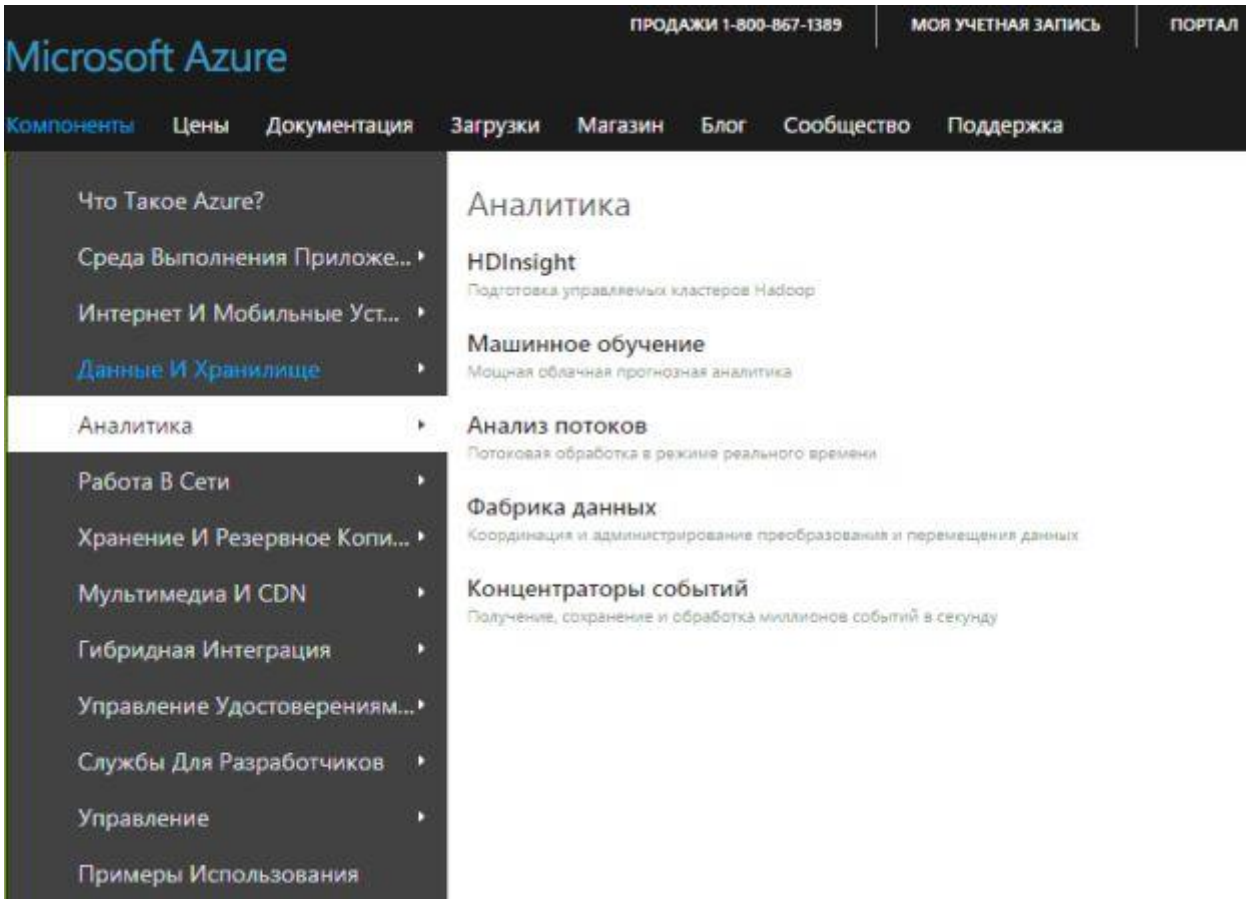


Рис. 3.17. Аналитика

HDInsight

Масштабируемость по требованию. HDInsight представляет собой расширение Hadoop на основе облачных технологий (рис. 3.18). Средство HDInsight было создано для обработки любого объема информации с масштабированием от терабайтов до петабайтов данных по мере необходимости. Можно запустить любое количество узлов в любое время. Плата взимается только за те вычислительные ресурсы и хранилища, которые реально используются.

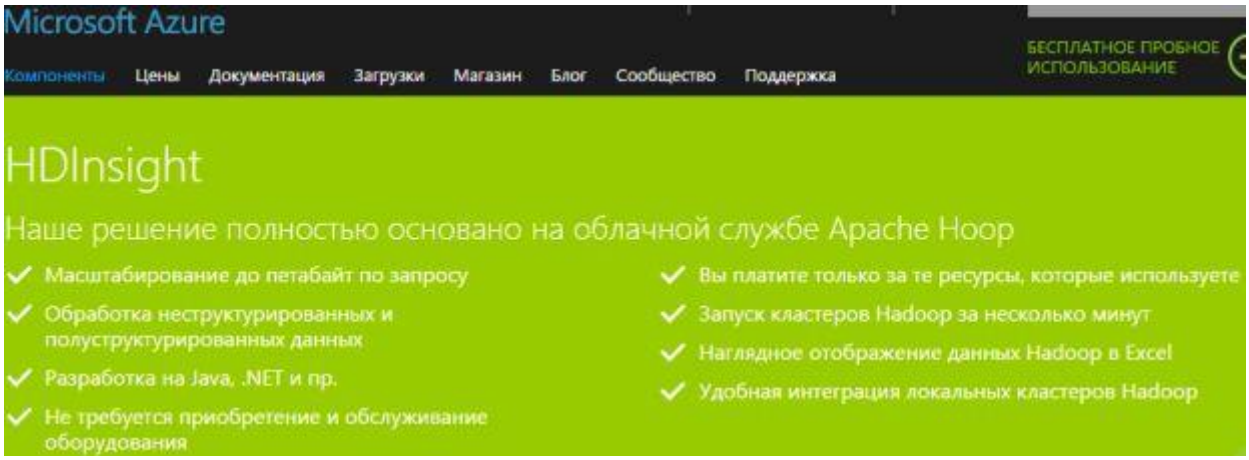


Рис. 3.18. HDInsight

Объединение всех данных – структурированных, полуструктурированных и неструктурированных. HDInsight совместим с Apache Hadoop и поэтому может обрабатывать неструктурированные или частично структурированные данные журналов посещений сайта, соцсетей, журналов серверов, устройств, сенсоров и т.д. Благодаря этому можно анализировать новые наборы данных и находить новые возможности для бизнеса, которые будут способствовать росту организации.

Разработка программного обеспечения на предпочитаемом языке. HDInsight имеет программные расширения для языков, включая C#, Java, .NET и др. Можно использовать выбранный язык программирования в Hadoop для создания, настройки, отправки и мониторинга заданий.

Не требуется приобретение и обслуживание оборудования. С помощью HDInsight можно развернуть Hadoop в облаке без покупки дополнительного оборудования и предварительной платы. Также не требуется длительная установка и настройка.

Для визуализации данных Hadoop используется Excel. HDInsight интегрирован с Excel, и это позволяет визуализировать и анализировать данные Hadoop. В Excel пользователи могут выбрать Azure HDInsight в качестве источника данных.

Локальные кластеры соединяются с облаком. HDInsight интегрирован с платформой данных Hortonworks, поэтому можно перемещать данные Hadoop из локального центра обработки данных в облако Azure для создания резервных копий, разработки и тестирования и сценариев "cloud bursting". С помощью платформенной системы аналитики Microsoft можно одновременно отправлять запросы в локальные и облачные кластеры Hadoop.

Включает транзакционные функции, не связанные с базами данных SQL. HDInsight включает в себя Apache HBase, столбчатую базу данных NoSQL,

работающую на базе распределенной файловой системы Hadoop (HDFS). Благодаря этому можно обрабатывать большие транзакции нереляционных данных и интерактивно записывать данные интерактивных веб-сайтов или сенсоров в хранилище BLOB-объектов Azure.

Обработка потоков в реальном времени. HDInsight включает Apache Storm, потоковую платформу аналитики, способную обрабатывать множество событий в реальном времени. Она позволяет обрабатывать миллионы создаваемых событий и поддерживать такие сценарии, как "Интернет вещей", получая данные от подключенных устройств и веб-событий.

Создание кластера с помощью HDInsight. С помощью HDInsight можно создать несколько кластеров Hadoop с одним и тем же набором данных (рис. 3.19).

Рис. 3.19. Создание кластера с помощью HDInsight

Машинное обучение

Возможности машинного обучения. Машинное обучение – интеллектуальный анализ ретроспективных данных с помощью вычислительных систем для прогнозирования будущих тенденций или поведения. Поисковые системы, рекомендации в Интернете, целевая реклама, виртуальные помощники, прогнозирование спроса, выявление мошенничества, фильтры нежелательной почты – машинное обучение позволяет работать всем этим современным службам.

Машинное обучение Azure. Для машинного обучения требуется комплексное программное обеспечение, современные компьютеры и специалисты. Машинное обучение Azure – полностью управляемая облачная служба для прогнозной аналитики. С помощью облака машинное обучение Azure делает машинное обучение доступнее для более широкой аудитории.

Перетаскивание, прогнозирование. Машинное обучение Azure позволяет пользователям без наработок данных начать интеллектуальный анализ данных для прогнозирования. Для многих задач не требуется писать ни одной строки кода. Студия машинного обучения Microsoft Azure также содержит библиотеку экономящих время примеров экспериментов и сложных алгоритмов Microsoft Research, включая те же проверенные алгоритмы, которые используются в Bing и Xbox.

Поддержка R. Машинное обучение Azure также предназначено для опытных специалистов по изучению данных. Оно поддерживает R, популярную программную среду с открытым исходным кодом для статистики и интеллектуального анализа данных. Существующий код R помещают в рабочее пространство или записывают собственный код в студии машинного обучения Microsoft Azure, которая поддерживает безопасное использование более 350 пакетов R.

С машинным обучением Azure не нужно устанавливать программное обеспечение, настраивать оборудование или скрытую среду разработки. Можно войти в Azure и начать разработку моделей прогнозирования откуда угодно, не используя ничего, кроме браузера, и развертывать новые модели аналитики. Машинное обучение Azure также позволяет хранить практически неограниченное количество файлов в хранилище Azure. Оно подключается к другим службам Azure для работы с данными, включая HDInsight (основанное на Hadoop решение больших данных), базу данных SQL и виртуальные машины.

Машинное обучение Azure объединяет новые средства аналитики, алгоритмы, разработанные для Xbox и Bing, и годы исследований машинного обучения Microsoft в одной облачной службе. Оно предоставит начинающим разработчикам и компаниям данных недорогой доступ к средствам.

Stream Analytics (Анализ потоков)

Анализ потоков позволяет уйти от трудностей при разработке аналитических функций для масштабирования распределенных систем. Разработчикам нужно описать требуемую трансформацию посредством синтаксиса на основе SQL, а система автоматически разложит ее по масштабу, производительности и отказоустойчивости.

Фабрика данных

Фабрика данных позволяет обрабатывать локальные данные, например из SQL Server, вместе с облачными данными, например из базы данных SQL Azure, больших двоичных объектов и таблиц. Эти источники данных можно объединять,

обрабатывать и отслеживать с помощью простых, высокодоступных и устойчивых к сбоям конвейеров данных.

Концентраторы событий

Концентраторы событий – это приемник на основе публикации и подписки с высокой степенью масштабируемости, который принимает миллионы событий в секунду, чтобы можно было обработать и проанализировать большой объем данных с подключенных устройств и из приложений (рис. 3.20). Данные, собранные концентраторами событий, можно преобразовать и сохранить, используя любого поставщика аналитики в режиме реального времени, или с помощью адаптеров пакетов или хранилища.

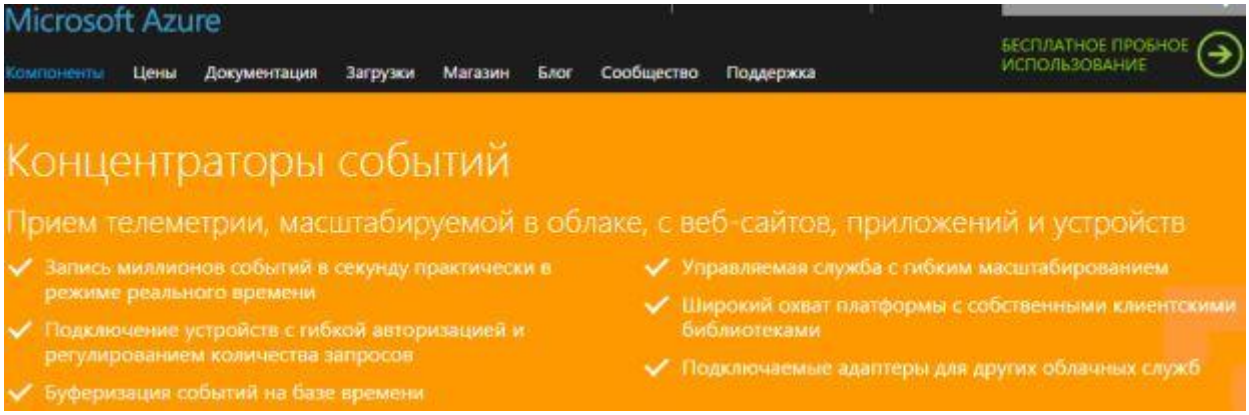


Рис. 3.20. Концентраторы событий

Подключение миллионов устройств разных платформ. Концентраторы событий предоставляют возможность подготовки емкости для принятия событий с миллионов устройств, сохраняя порядок событий для каждого устройства. Поддержка AMQP и HTTP позволяет многим платформам работать с концентраторами событий. Также для популярных платформ существуют собственные клиентские библиотеки.

Работа в сети

На рис. 3.21 представлены разделы компоненты "Работа в сети":

- Виртуальная сеть. Подготовка частных сетей с подключением к локальным центрам обработки данных.
- ExpressRoute. Выделенные оптоволоконные подключения частных сетей к Azure.

• Traffic Manager. Балансировка входящего трафика для повышения производительности и доступности.

•

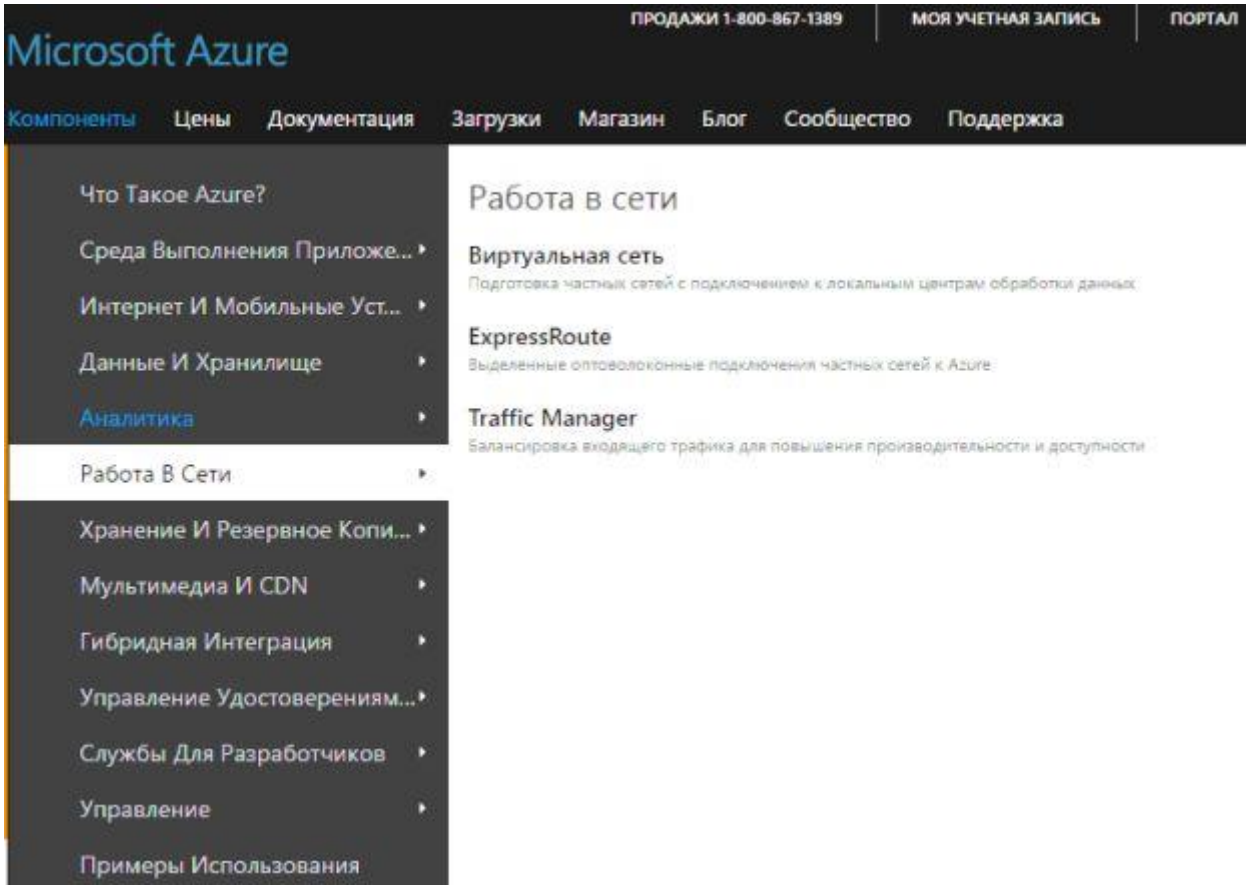


Рис. 3.21. Работа в сети

Виртуальная сеть

Расширение центра обработки данных в облачной среде. Виртуальная сеть Azure расширяет локальную сеть по межсайтовому VPN-соединению почти так же, как это делается при настройке соединение с удаленным филиалом (рис. 3.22). Можно контролировать топологию сети, включая конфигурацию DNS и диапазон IP-адресов, и управлять ею так же, как локальной инфраструктурой.

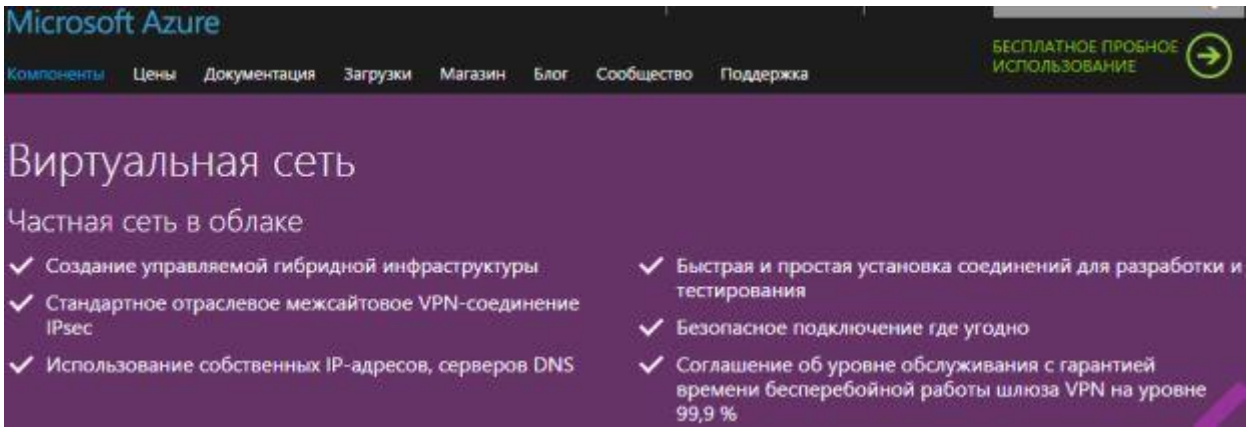


Рис. 3.22. Виртуальная сеть

Создание гибридных приложений. С помощью виртуальной сети можно создать гибридные облачные приложения, которые устанавливают безопасное подключение к локальному центру обработки данных; таким образом, веб-приложение Azure может осуществлять доступ к локальной базе данных SQL Server или осуществлять проверку подлинности пользователей локальной службы Active Directory.

Удаленная отладка приложений. Так как виртуальная сеть устанавливает прямое соединение между локальной машиной разработчика и виртуальными машинами Azure, то можно осуществлять отладку и устранение неполадок с использованием тех же средств, которые используются для отладки локальных приложений.

Оптимизированная защита и изоляция. Виртуальные сети предоставляют дополнительный уровень безопасности. Только виртуальные машины и службы, работающие в одной и той же виртуальной сети, могут идентифицировать друг друга и подключаться друг к другу. Облачные службы и виртуальные машины в виртуальной сети также не устанавливают подключения через общедоступную часть сети Интернет.

Сочетание IaaS и PaaS. С помощью виртуальных сетей можно создавать вычислительные службы, которые используют облачные службы и виртуальные машины. Веб-роли Azure используются для интерфейсных веб-серверов. Виртуальные машины используются для серверных баз данных. Объединение архитектур PaaS (платформа как услуга) и IaaS (инфраструктура как услуга) в виртуальной сети повышает гибкость и масштабируемость при создании приложений.

ExpressRoute

Azure ExpressRoute позволяет создавать частные подключения между центрами обработки данных Azure и инфраструктурой локальной среды или среды для совместной работы (рис. 3.23). Подключения ExpressRoute осуществляются не через общедоступную часть сети Интернет и обеспечивают повышенный уровень надежности, быстродействия и безопасности по сравнению с обычными интернет-соединениями. В некоторых случаях использование подключений ExpressRoute для быстрой передачи данных между локальной средой и Azure позволяет значительно сократить затраты.

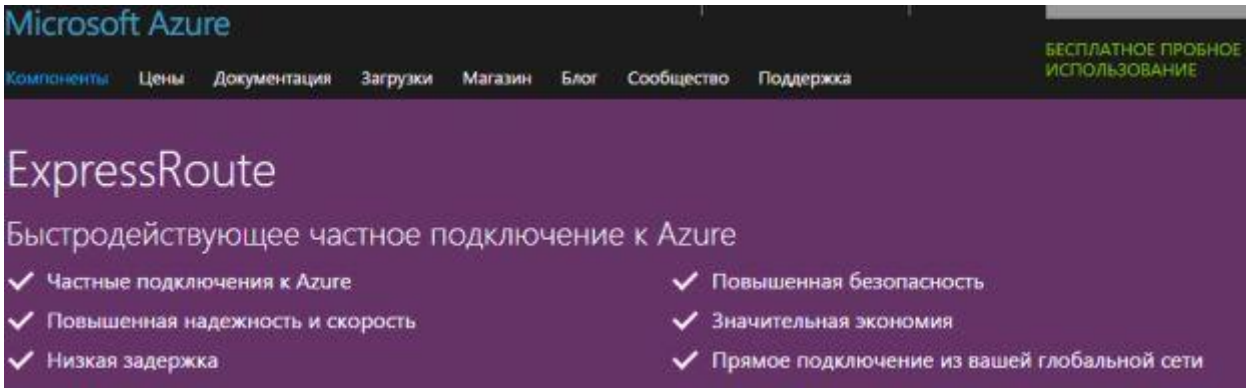


Рис. 3.23. ExpressRoute

С помощью ExpressRoute можно установить подключения к Azure в расположении ExpressRoute (объект поставщика Exchange) или напрямую подключиться к Azure из имеющейся глобальной сети (например, MPLS VPN), предоставленной поставщиком сетевых услуг.

Хранение, архивация и восстановление данных. ExpressRoute обеспечивает подключение к Azure, что позволяет использовать этот продукт для таких сценариев, как периодический перенос данных, репликация для обеспечения непрерывности бизнеса, аварийное восстановление и другие стратегии высокой доступности. Этот продукт также позволяет без лишних затрат переносить большие объемы данных, например наборы данных для высокопроизводительных вычислительных приложений, или перемещать большие виртуальные машины между средой разработки и тестирования в Azure и локальной рабочей средой.

Расширение центра данных. ExpressRoute позволяет наращивать вычислительную мощность и емкость хранилища имеющегося центра обработки данных.

Создание гибридных приложений. Благодаря предсказуемости, надежности и высокой пропускной способности подключений ExpressRoute можно создавать приложения, которые охватывают как локальную инфраструктуру, так и среду Azure, без снижения уровня производительности или безопасности. Например, можно запустить свое корпоративное приложение интрасети в среде Azure, чтобы осуществлять проверку подлинности пользователей с использованием локального Active Directory, а также обслуживать всех корпоративных пользователей без передачи трафика по общедоступной части сети Интернет.

Traffic Manager

Гибкие возможности балансировки нагрузки. Диспетчер трафика позволяет выбрать три метода балансировки нагрузки: по отказоустойчивости, по

производительности или по оценочному циклическому перебору (рис. 3.24). Можно выбрать тот метод, который подходит для конкретного приложения или сценария.

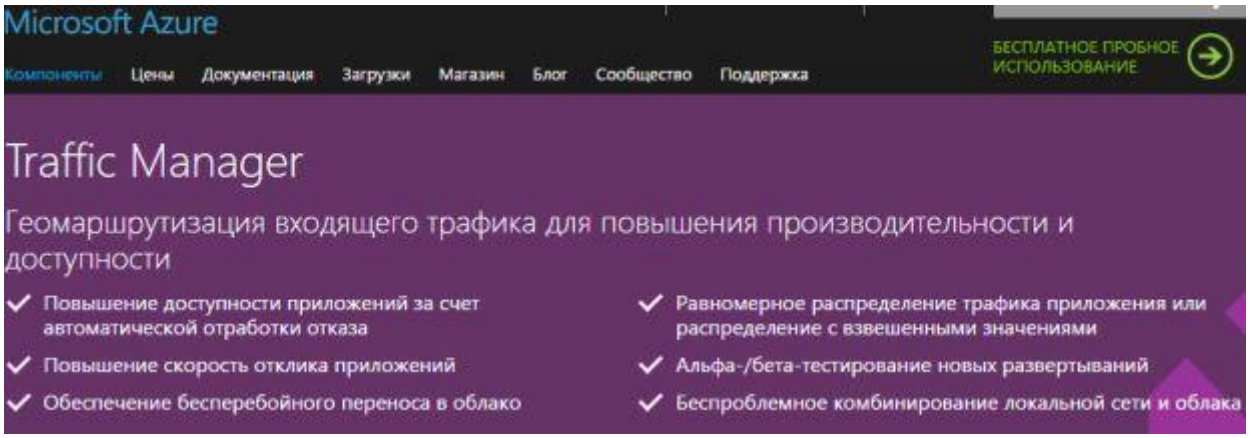


Рис. 3.24. Traffic Manager

Сокращение времени простоя приложений. Диспетчер трафика может повышать доступность важных приложений путем отслеживания ресурсов Azure или внешних сайтов и служб и автоматического перенаправления пользователей в новое местоположение при любых сбоях.

Повышенная производительность приложения, ускоренная доставка контента. Диспетчер трафика предназначен для повышения скорости отклика приложений и уменьшения времени доставки контента путем перенаправления пользователей в среду Azure или внешнее расположение с минимальной задержкой в сети.

Распределение пользовательского трафика по различным местоположениям. Диспетчер трафика может направлять пользовательский трафик и распределять его по различным местоположениям, таким как различные облачные службы в центре обработки данных Azure или различные веб-сайты Azure в разных центрах обработки данных. Диспетчер трафика может работать по принципу равномерного или оценочного распределения нагрузки.

Работа с локальным центром обработки данных. Диспетчер трафика используется в локальных сценариях, включая расширение в облако, перенос в облако или обработка отказов в облаке. Эту функцию используют для обновления или выполнения обслуживания в локальном центре обработки данных без создания неудобств для клиентов.

Хранение и резервное копирование

На рис. 3.25 представлены разделы компоненты "Хранение и резервное копирование":

• Хранилище. Надежное, высокодоступное и высокомасштабируемое облачное хранилище.

• Архивация. Архивация серверов в облако.

• Восстановление сайтов. Управление защитой и восстановлением частных облаков.

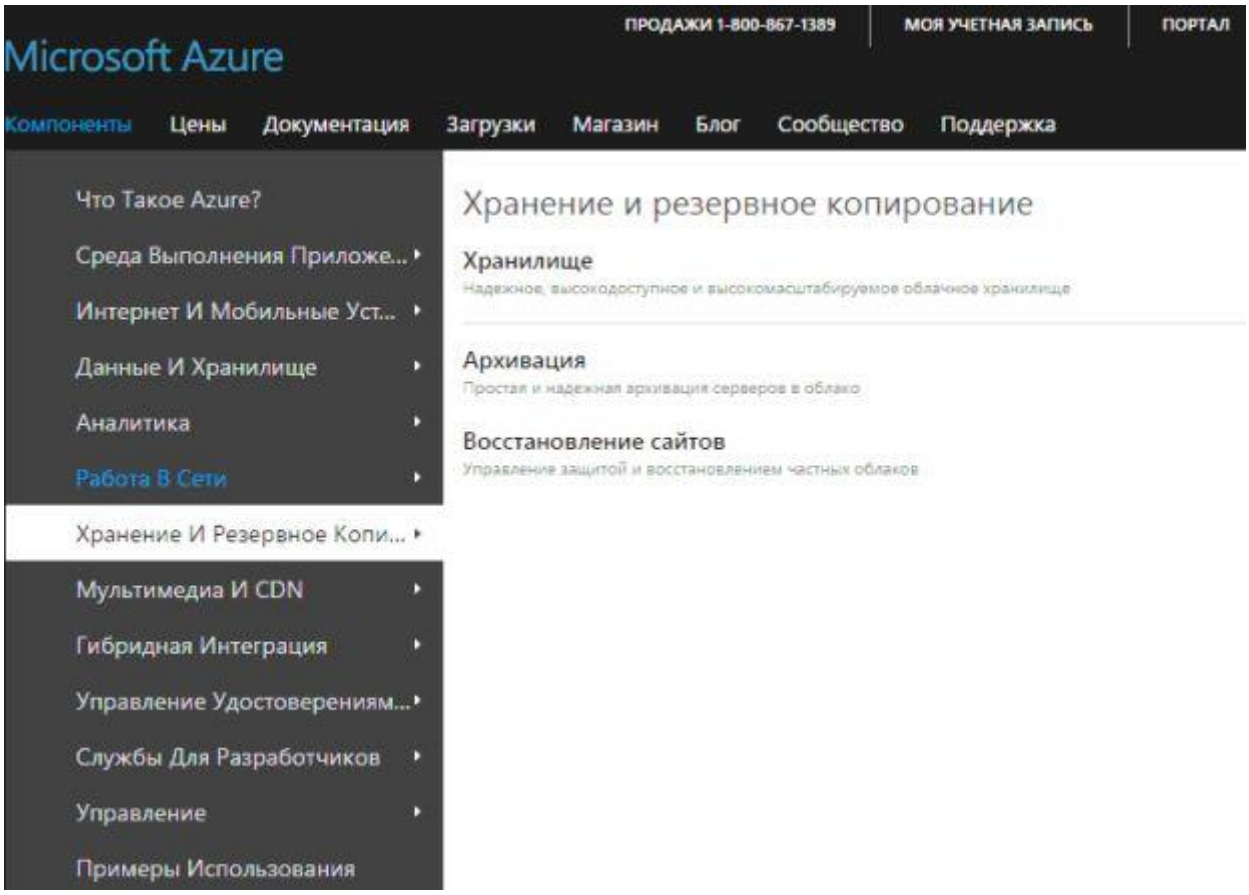


Рис. 3.25. Хранение и резервное копирование

Архивация

Возможности резервного копирования в облаке. Облачные решения архивации предоставляют надежное, недорогое и масштабируемое решение без капиталовложений и с минимальными операционными расходами.

Служба архивации Azure. Служба архивации Azure – это решение для защиты данных, которое позволяет клиентам выполнять резервное копирование локальных данных в Microsoft Azure.

Интерфейс. Служба архивации Azure проста в использовании и защищает данные Windows Server, Windows Server Essentials и System Center Data Protection Manager (DPM). Диспетчер DPM защищает многие рабочие нагрузки Microsoft,

позволяя расширить защиту в службу Azure. Управлять резервными копиями можно с помощью PowerShell или в пользовательском интерфейсе папки "Входящие".

Безопасность и надежность. Данные резервного копирования защищены при передаче и во время хранения. Данные резервного копирования хранятся в георепликированном хранилище, которое обслуживает шесть копий данных в двух центрах обработки данных Azure.

Эффективность и гибкость. Служба архивации Azure эффективна для работы по сети и на диске. После завершения первоначальной отправки только дополнительные изменения отправляются с определенной частотой. Встроенные функции, такие как сжатие, шифрование, более длительное хранение и регулирование пропускной способности, помогают повысить производительность ИТ-систем.

Восстановление сайтов

Служба Azure Site Recovery помогает защитить важные приложения путем координации репликации и восстановления частных облаков по сайтам вне зависимости от количества виртуальных машин (рис. 3.26). Можно защитить приложения для второго собственного сайта, сайта хостера или использовать Azure в качестве сайта аварийного восстановления, минуя дорогостоящий и сложный процесс создания вторичного месторасположения и управления им.

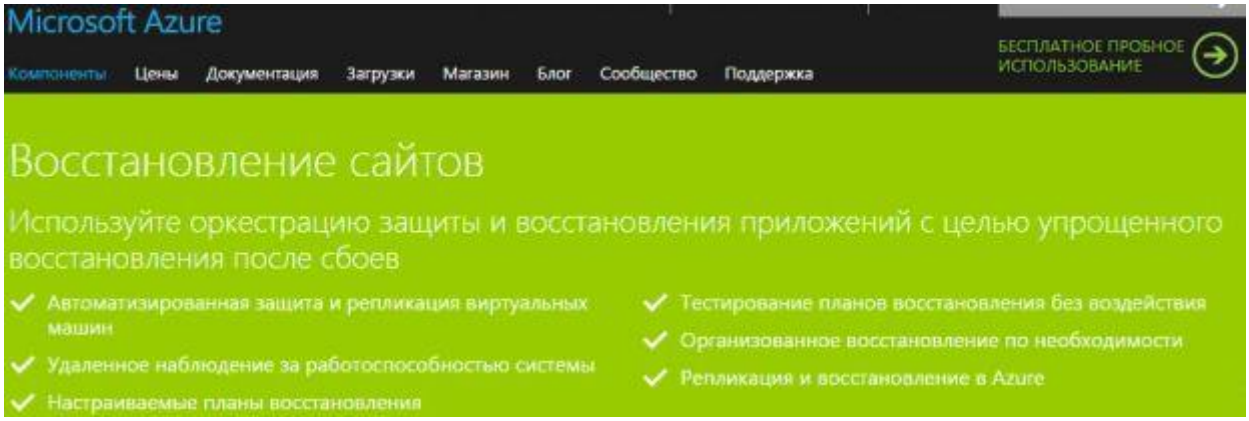


Рис. 3.26. Восстановление сайтов

Автоматизированная защита. Вычислительную среду можно защитить путем автоматизации репликации виртуальных машин на основе установленных и контролируемых политик. Служба восстановления сайтов координирует текущую репликацию данных и управляет ею с помощью интеграции с имеющимися технологиями, например, Hyper-V Replica, System Center и SQL Server AlwaysOn.

Репликация и восстановление в Azure. Можно упростить защиту аварийного восстановления путем репликации в Azure, в то же время применять автоматизированные и настраиваемые планы восстановления, наблюдать за работоспособностью системы и за организованным восстановлением, предоставляемыми данной службой. Одним из основных препятствий при организации полноценной защиты приложений являются высокие расходы на создание и обслуживание вторичного сайта для восстановления после сбоев.

Непрерывное наблюдение за работоспособностью системы. Служба восстановления сайтов непрерывно и удаленно отслеживает состояние диспетчера системного центра виртуальных машин в Azure. При выполнении репликации между двумя сайтами, управление которыми осуществляется, к Azure напрямую обращаются только серверы диспетчера виртуальных машин, а данные на виртуальных машинах и репликация остаются в локальной сети. Все соединения с Azure шифруются. При выполнении репликации в Azure в качестве вторичного сайта данные шифруются. Также можно выбрать состояние кодирования для хранимых данных.

Организованное восстановление. Эта служба помогает автоматизировать планомерное восстановление служб в случае сбоя работы сайта в основном центре обработки данных. Виртуальные машины могут использоваться для помощи в быстром восстановлении службы и при сложной многоуровневой рабочей нагрузке. Планы восстановления можно создавать на портале управления Azure, где они и хранятся. Планы могут быть простыми и сложными, в зависимости от требований бизнеса, и могут включать выполнение настраиваемых скриптов Windows PowerShell и паузы для ручного вмешательства. Предоставляется возможность настройки сетей посредством привязки виртуальных сетей между первичным и вторичным сайтами. Эти планы можно тестировать в любой момент без нарушения работы служб в основном расположении.

Мультимедиа и CDN

На рис. 3.27 представлены разделы компоненты "Мультимедиа и CDN":

- Службы носителей. Кодировка, хранение и потоковая передача аудио и видео с учетом масштаба.
- Сеть кэширующих серверов (CDN). Доставка контента пользователям в надежной сети распределенных центров обработки данных.
-

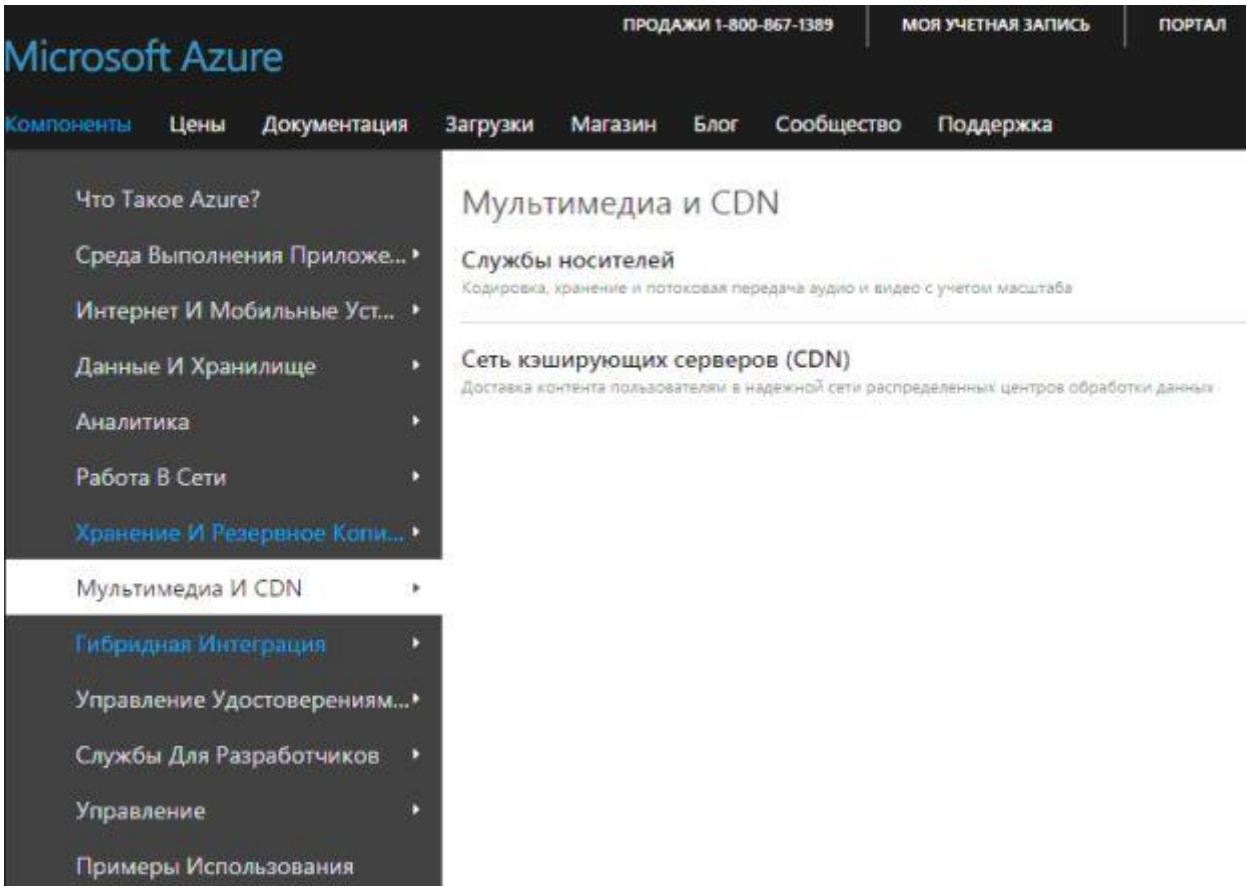


Рис. 3.27. Мультимедиа и CDN

Службы носителей

Службы носителей. Можно переносить любые мультимедиа на какое угодно устройство в любое место, пользуясь возможностями облака Azure (рис. 3.28). Создание комплексных процессов для работы с мультимедиа с помощью гибких служб кодирования, упаковки и распространения, обладающих высокой степенью масштабируемости, использующих службы мультимедиа Microsoft Azure. Можно передавать, сохранять, кодировать и упаковывать видео- или аудиосодержимое в виде потока по требованию или динамического потока в разные конечные точки: телевизоры, ПК и мобильные устройства.

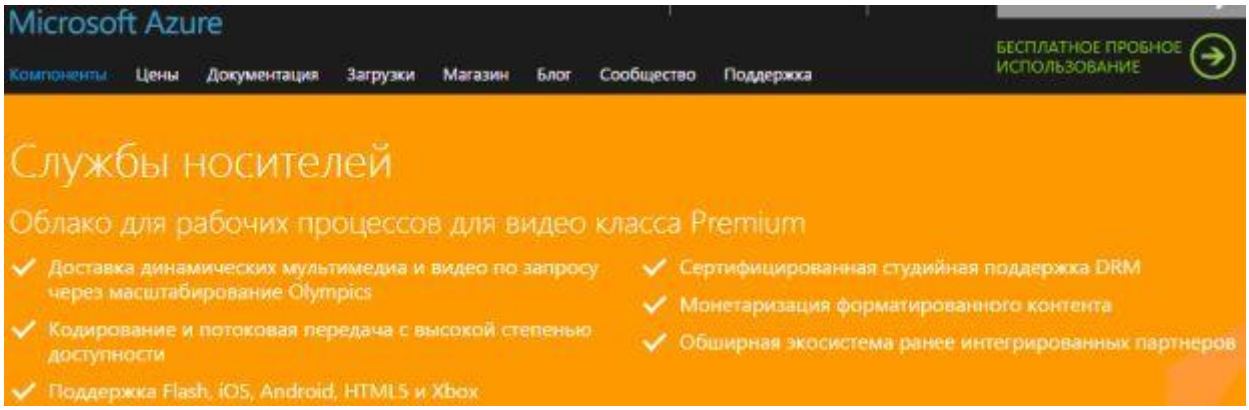


Рис. 3.28. Службы носителей

Кодировка, хранение и потоковая передача аудио и видео с учетом масштаба. Службы Azure Media Services позволяют строить всесторонние комплексные мультимедийные рабочие процессы, а также уменьшают издержки, связанные с интеграцией многочисленных продуктов и поставщиков.

Потоковая трансляция и видео по запросу. Для этого достаточно эксплуатировать собственное облако на основе видеоплатформы, предназначенной для динамических событий и видео по запросу (VOD). Службы мультимедиа Microsoft Microsoft Azure включают в себя все средства и службы, необходимые для обработки, доставки и потребления мультимедиа.

Защита контента. Активы защищаются с помощью шифрования во время отправки, хранения и воспроизведения с помощью управления цифровыми правами (DRM) Microsoft PlayReady или шифрования AES.

Кодирование. Для согласования скорости кодирования с потребностями рабочего процесса используются резервированные блоки кодирования Basic, Standard и Premium. Для поддержки различных входных и выходных форматов файлов студийного уровня используется Azure Media Encoder.

Сеть доставки содержимого с глобальным доступом. Благодаря надежной сети глобальных центров обработки данных сеть доставки содержимого Azure обеспечивает доставку больших объемов контента конечным пользователям по всему миру при меньшей задержке и более высоком уровне доступности.

Доставка на сотни миллионов конечных точек устройств. В качестве целевой аудитории могут выступать владельцы всех типов современных наиболее распространенных клиентских устройств, включая компьютеры Windows, Android и iOS, а также телевизоров, игровых консолей и т.д.

Варианты быстрого получения и динамическая упаковка. Хранение оптимизируется за счет кодирования в формат MP4 с переменным сжатием и оперативной доставки в разнообразных форматах на лету.

Широкая доступность с использованием платформ проигрывателей и пакетов SDK. Данные клиентам предоставляются на различных популярных платформах и устройствах. Это стало возможно с помощью платформ Player, образцов проигрывателей и комплектов SDK от Microsoft.

Надежная партнерская экосистема. Расширяемая платформа, усовершенствованная с помощью известных технологий мультимедиа стороннего поставщика, которые были оптимизированы для облака и интегрированы в целях обеспечения масштабируемости и единого порядка выставления счетов.

Оплата по мере использования. Оплачивается только то, что используется.

Расширенное индексирование контента. Индексатор Azure Media Services Indexe позволяет сделать медиафайлы более доступными и полезными. Он может проиндексировать библиотеку медиафайлов для дальнейшего поиска по ключевым словам, фразам или клипам. Индексатор создает закрытые файлы титров.

Упрощенное управление видео-контентом. Хранение, кодирование видеоконтента служб мультимедиа и управление им.

Сеть кэширующих серверов (CDN)

*Производительность приложений и служб*Сеть доставки содержимого Azure (CDN) предназначена для отправки звуковых файлов, видеоматериалов, приложений, изображений и других файлов с использованием серверов, расположенных ближе всего к каждому пользователю (рис. 1.29). Это значительно увеличивает скорость и доступность, что приводит к существенному улучшению взаимодействия с пользователями.

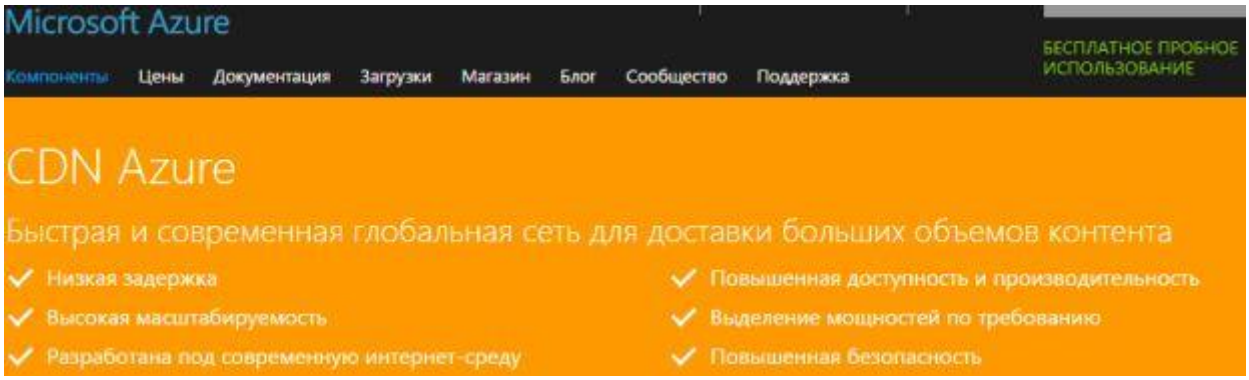


Рис. 3.29. Сеть кэширующих серверов (CDN)

Интернет-среда. Сеть Azure CDN была разработана под современную динамичную интернет-среду с ориентацией на мультимедийный контент. Сеть Azure CDN имеет сетевую топологию с крупными централизованными узлами, а также огромными вычислительными ресурсами и хранилищами на базе гибкой облачной инфраструктуры.

Безопасность. Сеть Azure CDN построена на базе архитектуры обратных прокси-серверов с высокой степенью масштабируемости с использованием сложных технологий для выявления атак DDoS.

Гибридная интеграция

На рис. 3.30 представлены разделы компоненты "Гибридная интеграция":

- Службы BizTalk. Интеграция предприятия и облака.
- Служебная шина. Связь между частными и общедоступными облачными средами.
- Архивация. Архивация серверов в облако.
- Восстановление сайтов. Управление защитой и восстановлением частных облаков.

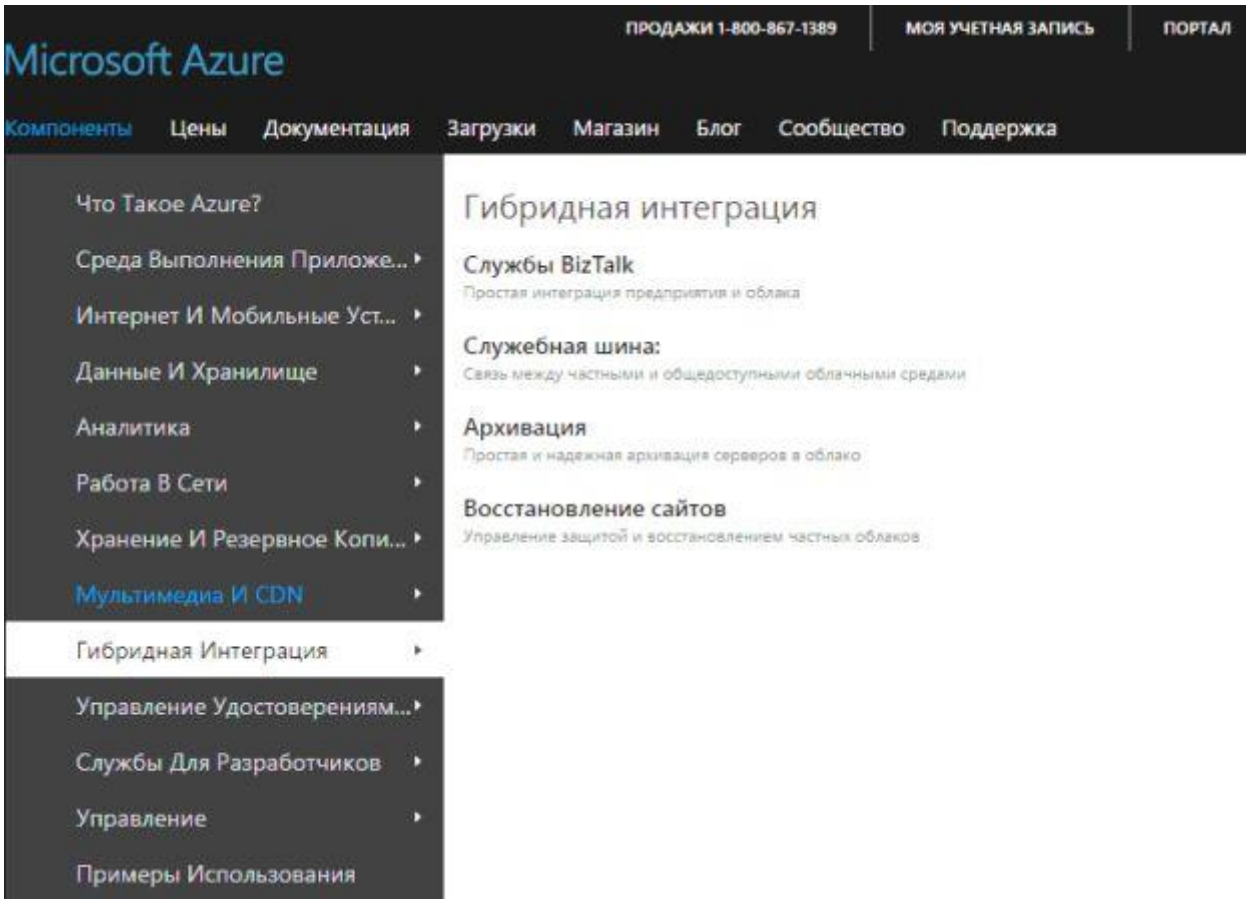


Рис. 3.30. Гибридная интеграция

Службы BizTalk

Службы Azure BizTalk – это расширяемые облачные службы интеграции. Они предоставляют возможности B2B, интеграции корпоративных приложений (EAI) и гибридных подключений для реализации облачных и гибридных решений интеграции. Эта служба работает в безопасной, выделенной для каждого клиента среде, которую можно подготовить к работе по запросу (рис. 3.31).

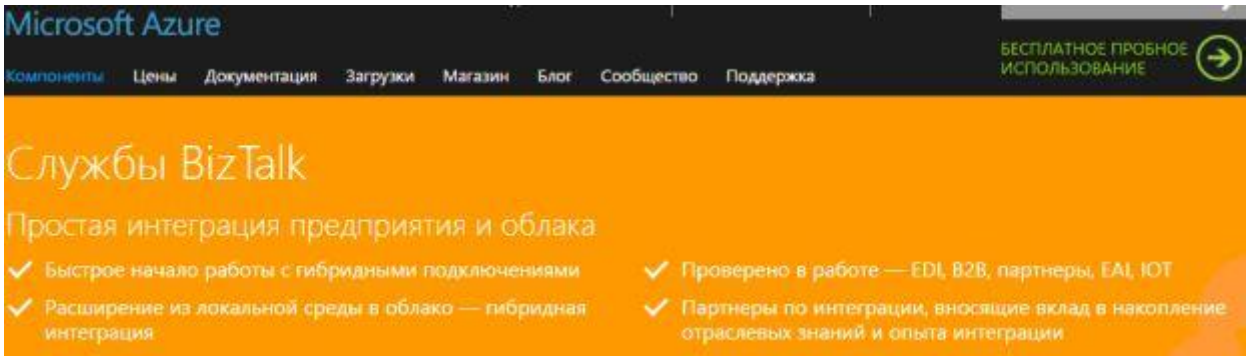


Рис. 3.31. Службы BizTalk

Службы BizTalk предоставляют решение для управления торговыми партнерами и обработки электронного обмена данными (EDI), что может сократить расходы на совместную работу бизнес-партнеров для предприятий и поставщиков служб EDI.

Интеграция корпоративных приложений. Службы BizTalk предоставляют готовые возможности для интеграции локальных бизнес-приложений с SAP, Oracle EBS, SQL Server и PeopleSoft. Они позволяют вам подключаться к любым источникам данных (HTTP, FTP, SFTP и REST). С помощью различных артефактов Azure можно маршрутизировать сообщения, таких как очереди и разделы шины обслуживания, база данных SQL и хранилище BLOB-объектов.

Гибридные подключения. Гибридные подключения Azure BizTalk позволяют подсоединить веб-сайты Azure или мобильные службы Azure к любому локальному ресурсу TCP или HTTP, такому как Microsoft SQL Server, MySQL или любая веб-служба, посредством внесения нескольких изменений в конфигурацию без использования дополнительного кода.

Служебная шина

Azure Service Bus – универсальная облачная система обмена сообщениями для связи практически со всем – с приложениями, службами и устройствами – вне

зависимости от их расположения (рис. 3.32). Устанавливается связь с приложениями, работающими на базе Azure и/или в локальной сети. Service Bus можно использовать для связи бытовых приборов, датчиков и других устройств, таких как планшеты и телефоны, с центральным приложением или друг с другом.

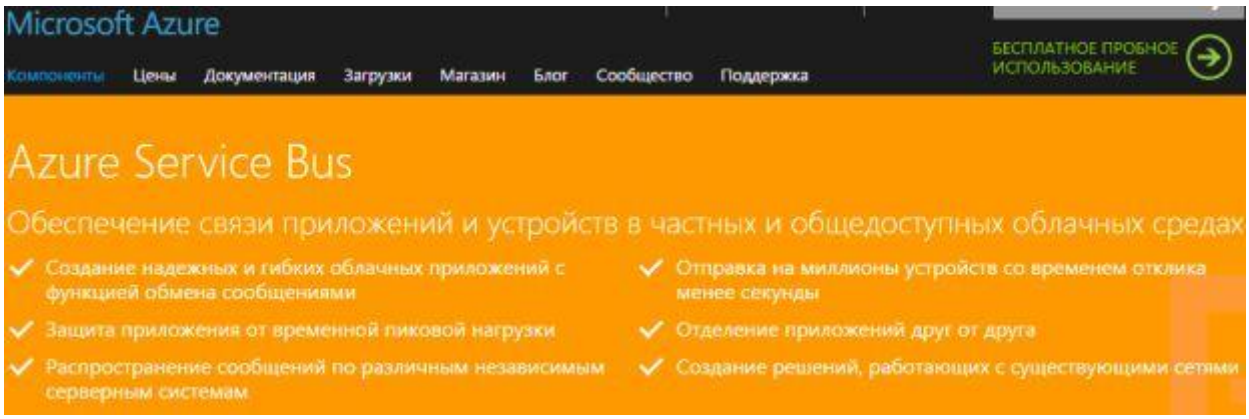


Рис. 3.32. Служебная шина

Создание облачных решений с изменением масштаба в соответствии с требованиями. Схемы асинхронного обмена сообщениями являются фундаментальными элементами архитектуры надежных и масштабируемых приложений. Интеграция облачных ресурсов, таких как SQL Azure, хранилище Azure и веб-сайты Azure, со службой обмена сообщениями Service Bus, обеспечивает бесперебойную работу в условиях большой и изменяющейся нагрузки и позволяет избежать простоев в случае перемежающихся сбоев.

Управление доставкой сообщений в облаке. Очереди обеспечивают гарантированную доставку сообщений по схеме FIFO и поддерживают ряд стандартных протоколов (REST, AMQP, WS*) и интерфейсов API. Разделы доставляют сообщения нескольким подпискам и распространяют их в системы, расположенные на нижних уровнях.

Подключение локальных приложений к облаку. Ретрансляция служебной шины решает задачи связи между локальными приложениями и внешним миром, позволяя локальным веб-службам проецировать общедоступные конечные точки. После этого системы могут получить доступ к этим веб-службам, которые продолжают работать локально в любой точке планеты.

Управление удостоверениями и доступом

На рис. 3.33 представлены разделы компоненты "Управление удостоверениями и доступом":

- Active Directory. Синхронизация локальных каталогов и поддержка единого вида.
- Многофакторная проверка подлинности. Защита доступа к данным и приложениям благодаря дополнительной проверке подлинности.

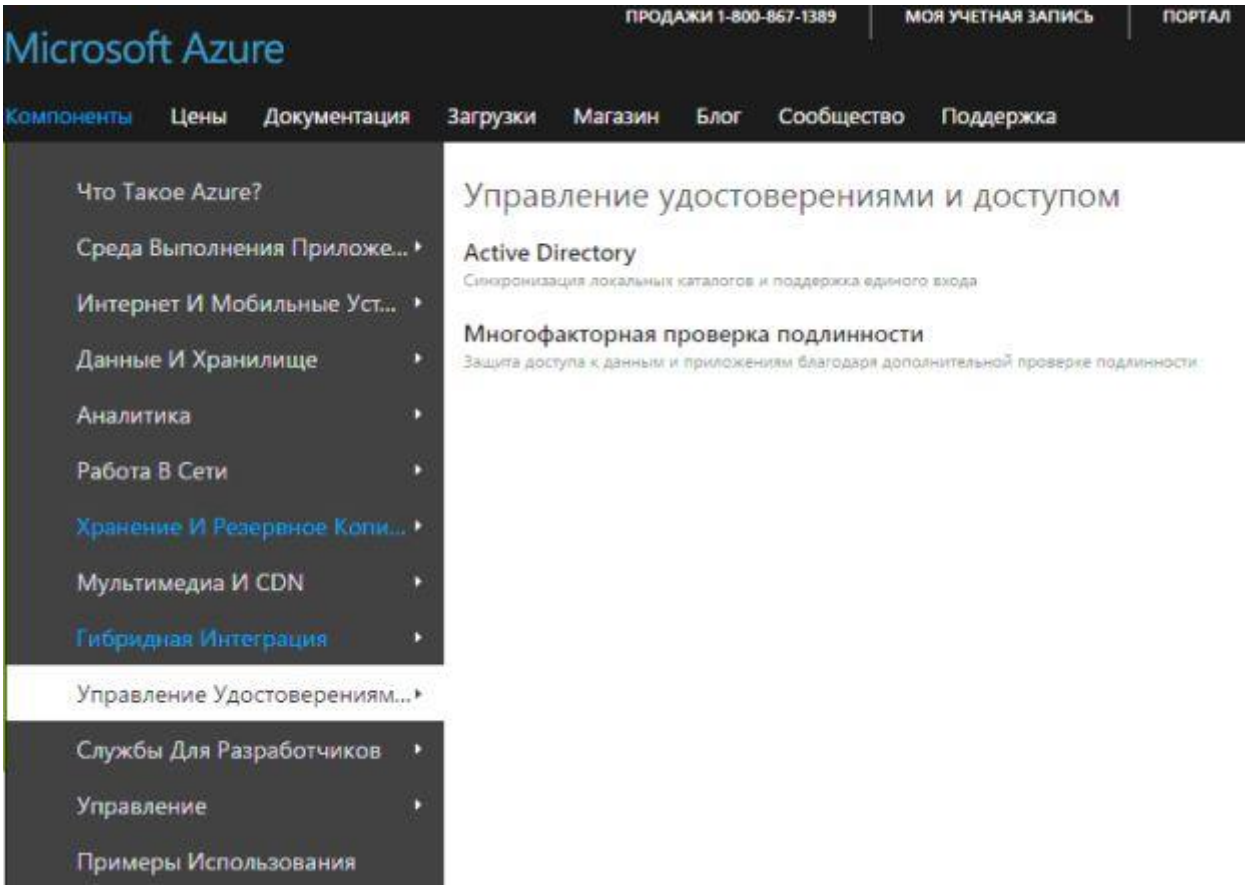


Рис. 3.33. Управление удостоверениями и доступом

Active Directory

Управление удостоверениями и доступом для облака. Azure Active Directory – это комплексное облачное решение для управления идентификацией и доступом, предоставляющее надежные функции для управления пользователями и группами и помогающее обеспечить безопасный доступ к приложениям, включая такие службы Microsoft Online Services, как Office 365 и множество приложений SaaS сторонних разработчиков (рис. 3.34).

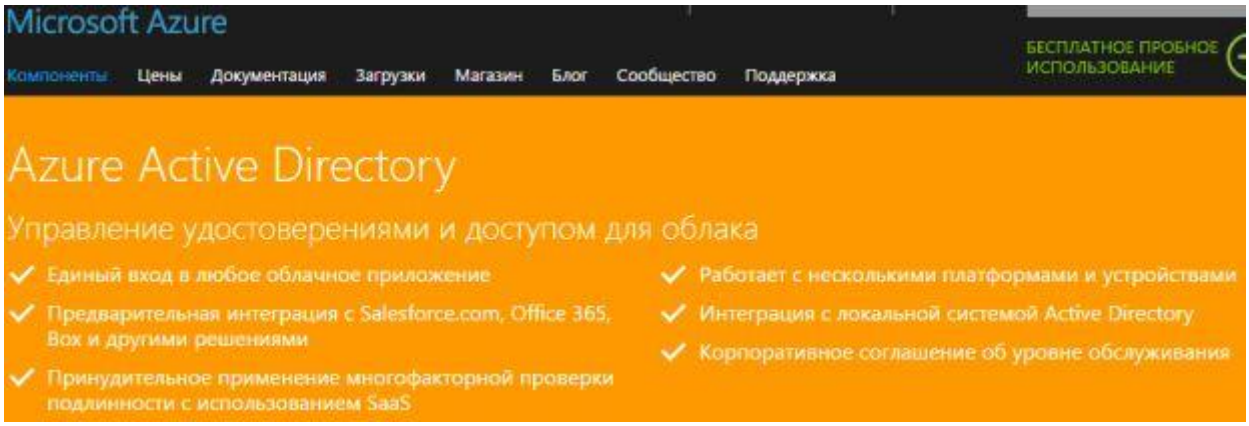


Рис. 3.34. Azure Active Directory

Доступ к облачному приложению. Используется единый вход для упрощения доступа пользователей к множеству различных облачных приложений с устройств под управлением Windows, Mac и iOS. Пользователи могут запускать облачные приложения из персонализированной веб-панели доступа с использованием корпоративных учетных данных.

Защита конфиденциальных данных и приложений. Многофакторная проверка подлинности Azure предоставляет дополнительный уровень безопасности и помогает предотвратить несанкционированный доступ к локальным и облачным приложениям. Для защиты бизнеса применяются функции контроля и оповещений средств безопасности, а также отчетов на базе машинного обучения, которые позволяют определить несогласованные схемы доступа и упреждают потенциальные угрозы.

Функции самообслуживания для сотрудников. Делегируются такие задачи, как сброс паролей, создание групп и управление ими своим сотрудникам. Пользователям предоставляются возможности самостоятельно изменять и сбрасывать пароль и самостоятельно управлять группами в Azure Active Directory Premium.

Интеграция с системой Active Directory. Возможности Active Directory и других локальных каталогов расширяются за счет Azure Active Directory, что обеспечивает единый вход во все облачные приложения.

Корпоративное соглашение об уровне обслуживания. Azure Active Directory Premium предоставляет организациям необходимый уровень масштабирования и обеспечения надежности. Доступность службы Azure Active Directory гарантируется благодаря размещению в глобальной сети распределенных центров обработки данных.

Полнофункциональная стандартная платформа для управления проверкой подлинности и доступом в облачных службах. Azure Active Directory предоставляет разработчикам эффективный способ интеграции в свои приложения управления удостоверениями. Поддержка стандартных отраслевых протоколов, включая SAML 2.0, WS-Federation и OpenID Connect, позволяет реализовать вход в систему на различных платформах, в том числе .Net, Java, Node.js и PHP. API Graph на основе REST обеспечивает чтение и запись данных каталога с любой платформы. Благодаря поддержке протокола OAuth 2.0 разработчики могут создавать мобильные и интернет-приложения, интегрируемые с API Microsoft и сторонних разработчиков, а также собственные защищенные API для Интернета.

Облачный каталог для Office 365. Корпоративный каталог и управление удостоверениями можно перенести в облако с помощью Azure Active Directory. Доступом сотрудников к службам Microsoft Online Services, таким как Azure, Microsoft Office 365, Dynamics CRM Online, Windows Intune и других облачных приложений сторонних разработчиков можно управлять централизованно.

Многофакторная проверка подлинности

Усиление безопасности, устранение препятствий. Многофакторная проверка подлинности Azure защищает доступ к данным и приложениям, а также предоставляет пользователям простой процесс входа. Она обеспечивает надежную проверку подлинности с использованием ряда простых вариантов подтверждения (посредством телефонных вызовов, текстовых сообщений или уведомлений в мобильных приложениях), позволяя пользователям самим выбирать предпочтительный способ (рис. 3.35).

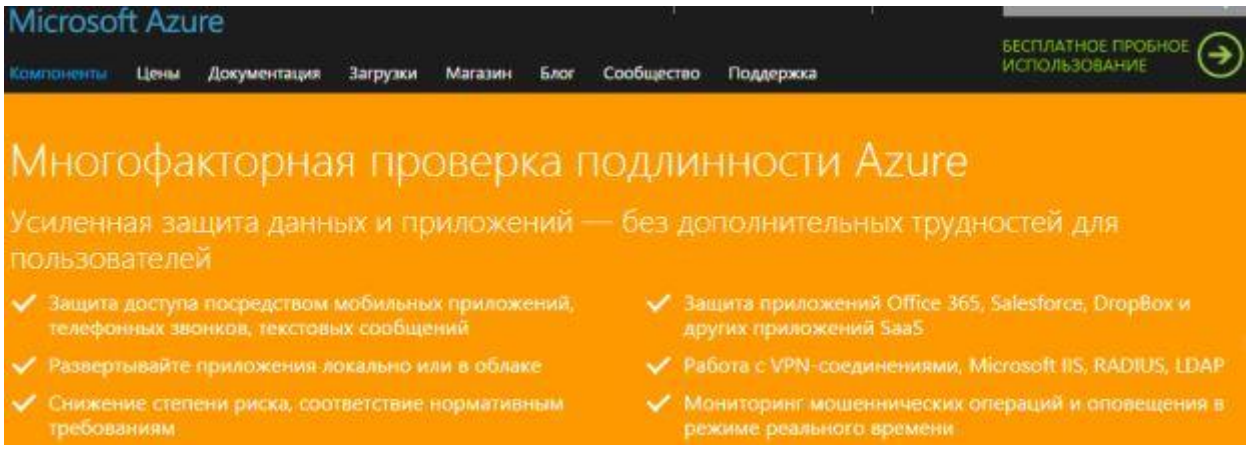


Рис. 3.35. Многофакторная проверка подлинности Azure

Мониторинг и оповещения в реальном времени. Многофакторная проверка подлинности Azure защищает бизнес с использованием эффективных функций мониторинга безопасности и отчетов на базе машинного обучения, которые позволяют определить несогласованные схемы доступа. Оповещения в режиме реального времени уведомляют сотрудников ИТ-отдела об использовании подозрительных учетных данных, что позволяет избежать потенциальных угроз.

Развертывание локально или в облаке. Сервер многофакторной проверки подлинности Azure используется в локальной среде, чтобы защитить виртуальные частные сети (VPN), службы федерации Microsoft Active Directory, веб-приложения Microsoft IIS, службы удаленных рабочих столов и другие приложения удаленного доступа с использованием проверки подлинности RADIUS и LDAP.

Работа с приложениями Office 365, Salesforce и т.д. Многофакторная проверка подлинности для Office 365 помогает защитить доступ к приложениям Office 365 без дополнительной платы. Многофакторная проверка подлинности также входит в состав предложения Azure Active Directory Premium и тысяч приложений SaaS (программное обеспечение как услуга), включая Salesforce, Dropbox и др.

Внедрение в приложения. С помощью пакета средств разработки программного обеспечения (Software Development Kit – SDK) можно встроить многофакторную проверку подлинности Azure в процесс входа или обработки транзакций приложения, используя существующую базу данных пользователей приложения.

Защита для администраторов Azure. Многофакторная проверка подлинности Azure позволяет добавить дополнительный уровень защиты в учетную запись администратора Azure без дополнительной платы. Если проверка включена, потребуется подтверждение личности для запуска виртуальной машины, управления хранилищем или использования других служб Azure.

Службы для разработчиков

На р[рис. 3.36](#) представлены разделы компоненты "Службы для разработчиков":

- Visual Studio Online. Планирование, сборка и распространение программ из единого центра.
- Подробные сведения о приложении. Возможность находить и устранять проблемы для улучшения веб-приложений.

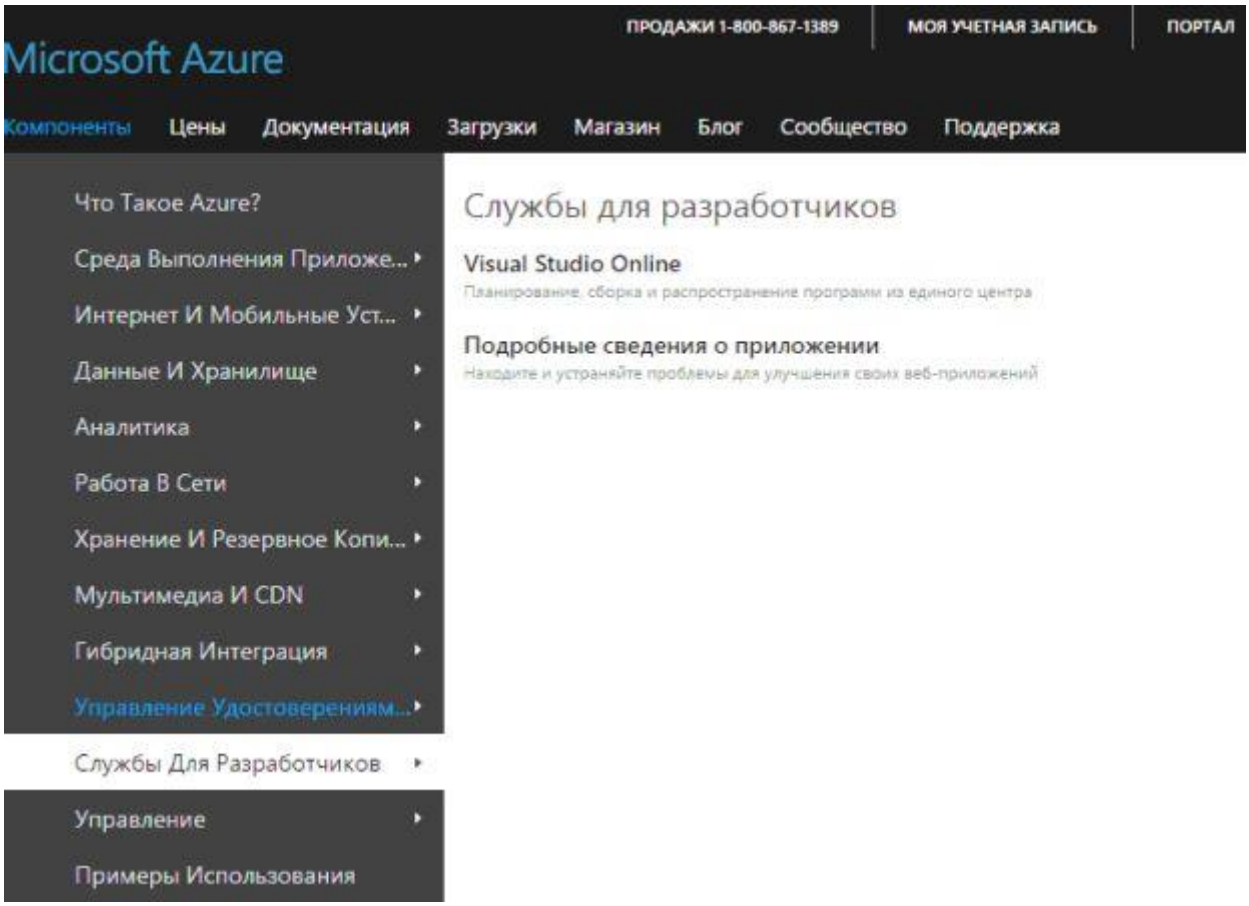


Рис. 3.36. Службы для разработчиков

Visual Studio Online

Visual Studio Online – это способ планирования, создания и доставки программного обеспечения на различные платформы (рис. 1.37).

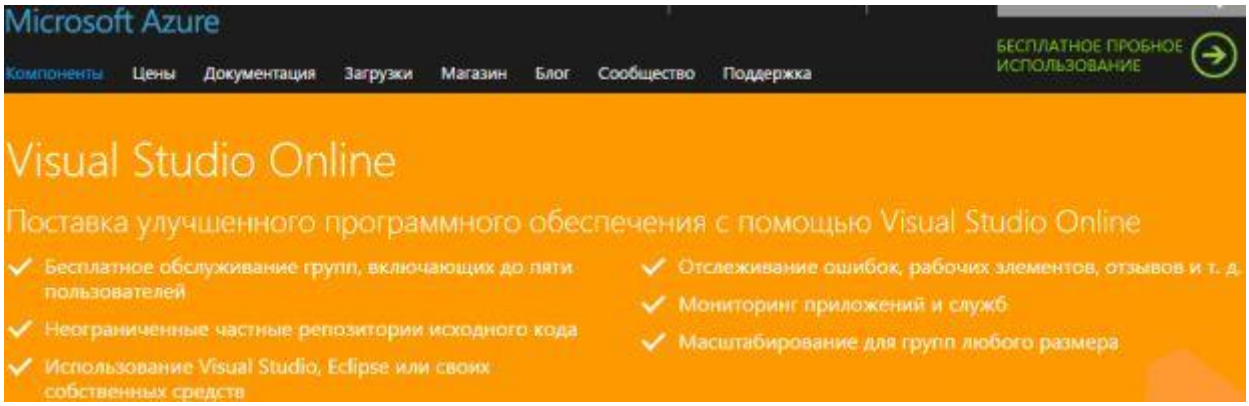


Рис. 3.37. Visual Studio Online

Частные проекты с использованием Git или TFVC. Git и TFVC позволяют создавать частные командные проекты, доступные из любой точки мира. Подсистема контроля версий Team Foundation (TFVC) используется для традиционного централизованного управления версиями. Git используется, если необходим

распределенный подход. Можно сопоставлять и совмещать различные групповые проекты, что упрощает размещение всей организации в одной учетной записи.

Централизованное отслеживание всех ресурсов. В портфелях незавершенных заданий фиксируются новые функциональные возможности, ошибки и другие рабочие элементы. Портфели незавершенных заданий подходят для групп, использующих Scrum, Kanban. Для отслеживания хода выполнения задач рабочей группой используются настраиваемые панели задач. Управление портфелями позволяет большим группам отслеживать работу всех входящих в нее команд.

Интеграция и развертывание на базе облачных технологий. Можно выявлять проблемы, связанные с качеством, на ранних этапах с помощью определений сборок, которые автоматически компилируют и тестируют приложения в облаке либо по требованию, либо после внесения любых изменений в код. Используя графики и настраиваемые панели мониторинга можно отслеживать работоспособность сборок с течением времени. После выполнения тестов, обновленные веб-сайты автоматически разворачиваются на Microsoft Azure.

Работа приложений. О пользователях и потенциальных проблемах, с которыми они могут столкнуться, можно узнать за счет получения информации о доступности, производительности и использовании из приложений вне зависимости от того, выполняются ли они на устройстве, на сервере, на платформе Microsoft Azure, в облачной среде стороннего поставщика или в условиях использования комбинированных ресурсов.

Использование средства разработки. Доступ к проектам, коду и рабочим элементам можно получить посредством интегрированного интерфейса в Visual Studio или бесплатного плагина для Eclipse. Поддержка любого клиента Git (включая Xcode) и интерфейса современных веб-браузеров позволяет гарантировать, что Visual Studio Online будет адаптироваться к требованиям рабочей группы вне зависимости от того, какое средство разработки используется.

Масштабируемое ценообразование, подходящее для небольших групп. Группы, состоящие не более чем из пяти участников, обслуживаются бесплатно, поэтому можно создавать неограниченное количество частных групповых проектов с требуемым числом репозиториев, рабочих элементов и ошибок без дополнительной оплаты. Для дополнительных пользователей действует принцип прямого ценообразования; предоставляется месячный объем общих ресурсов, таких как сборки и нагрузочные тесты, который может быть увеличен по мере роста требований.

Обеспечение доступности приложения. Веб-тесты настраиваются для обеспечения доступности и работоспособности службы. Также настраиваются предупредительные сообщения на основе пороговых значений показателей и счетчиков производительности.

Отслеживание и устранение проблемы производительности. Анализируются количество запросов и время отклика по всем зависимым компонентам. Многомерный анализ выполняется по нескольким показателям. Можно выбрать нужные из множества стандартных показателей и создавать собственные. Поиск и фильтрация выполняется по исключениям, событиям и журналам трассировки. Можно выполнять диагностику сбоев и решать проблемы с производительностью.

Анализирование использования приложения. Действия пользователей отслеживаются в рамках основных сценариев. Можно узнавать, как пользователи работают в приложении в целях его улучшения и привлечения большего числа пользователей. Можно соотносить показатели использования и производительности с полным представлением всего, что происходит в приложении. Основные тенденции внедрения определяются с целью проверки выполнения деловых задач. Можно определить приоритеты для инвестирования с учетом данных об использовании.

Безопасность и конфиденциальность корпоративного класса. Application Insights – это корпоративная облачная служба от Microsoft. Она построена на основе методов обеспечения безопасности Microsoft Azure.

Управление

На рис. 3.38 представлены разделы компоненты "Управление":

- Портал Preview.
- Планировщик.
- Автоматизация.
- Operational Insights. Сбор, анализ, визуализация данных локальных и облачных машин.

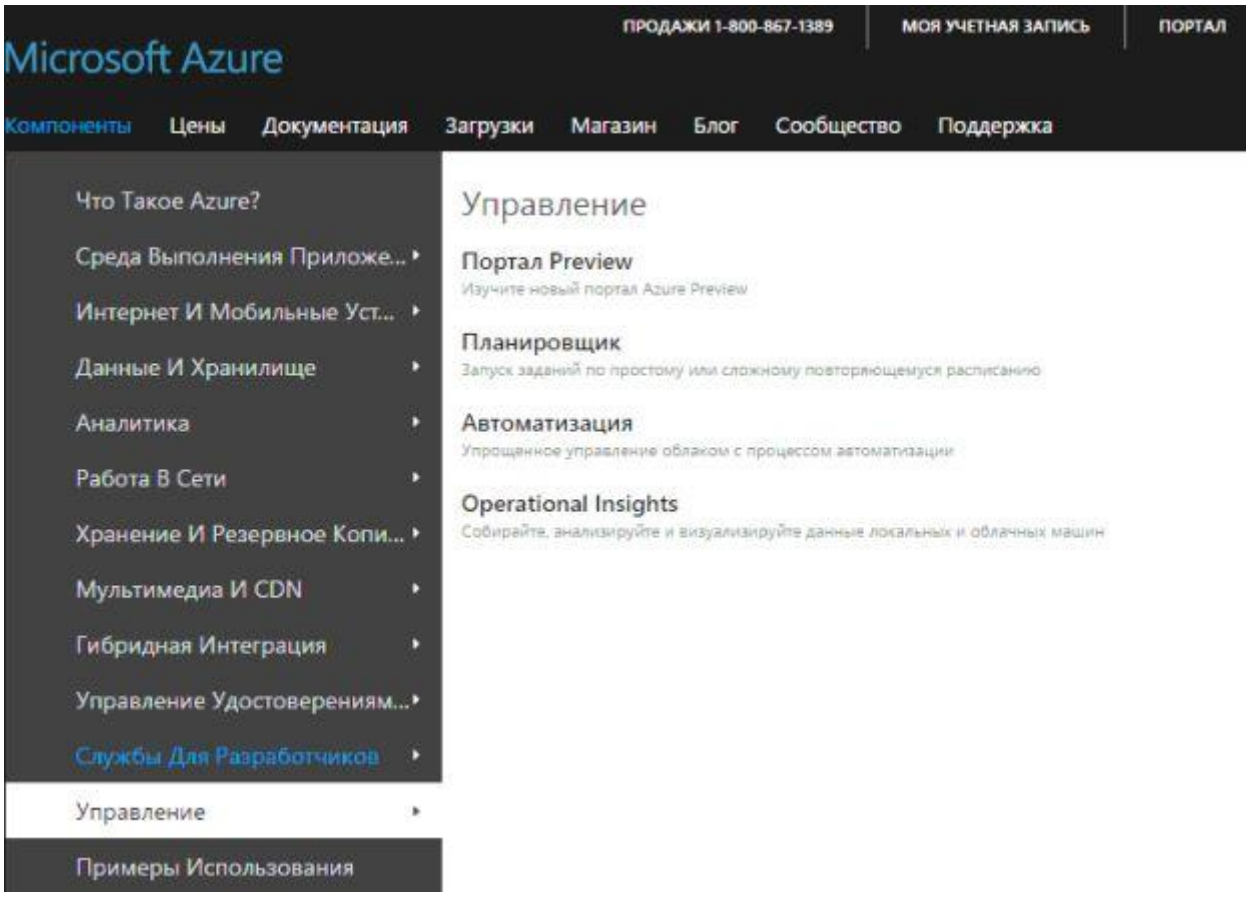


Рис. 3.38. Управление

Портал Azure Preview

На портале Microsoft Azure совместно используются межплатформенные средства, технологии и службы в рамках единого интегрированного рабочего пространства, чтобы радикально повысить скорость процесса доставки и существенно упростить сохранение работоспособного состояния приложений.

Автоматизация

Служба автоматизации Azure позволяет сосредоточиться на работе, что увеличивает ценность бизнеса (рис. 3.39). Уменьшение ошибок и увеличение эффективности позволяет сократить затраты на операции.

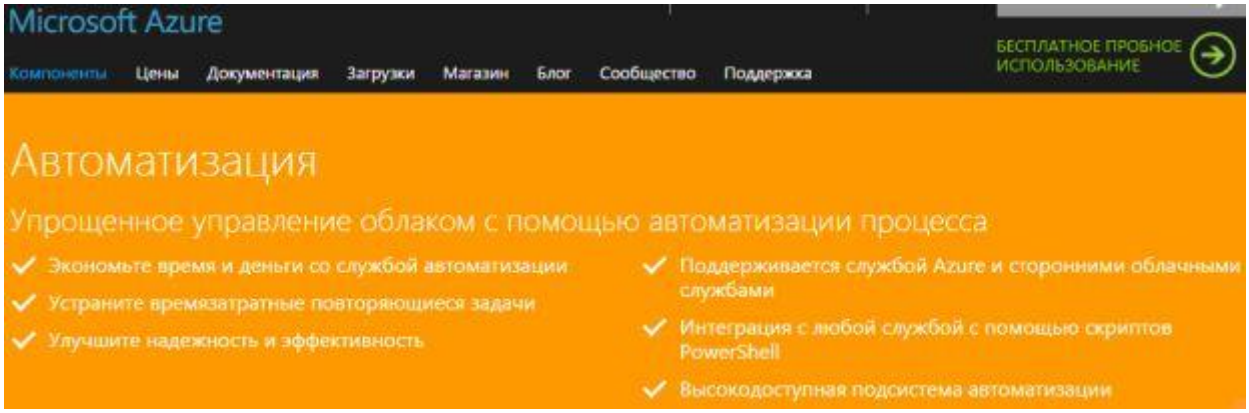


Рис. 3.39. Автоматизация

Интегрирование служб. Службы используют с любыми общедоступными веб-интерфейсами API службы.

Operational Insights

Сбор, комбинирование и визуализация всех машинных данных. Сбор данных из машинных источников в нескольких системах и поиск по ним в централизованном едином хранилище данных помогает определять первопричину неисправностей. Информационные панели обладают поисковыми функциями. Поисковые запросы перетаскивают из места в место для создания собственных визуализаций.

Управление ресурсами своего центра обработки данных (ЦОД). Можно определить участки, на которых не хватает производительности. Найдя старые или избыточно выделенные виртуальные машины, можно запланировать свои потребности в хранении данных и вычислительных ресурсов.

Поддерживание серверов в актуальном состоянии. Необходимо найти упущенные системные обновления и узнать положение дел с вредоносным ПО на всех серверах Windows независимо от того, работают они в собственных ЦОД или в общедоступном облаке. Таким образом, можно узнать, на каких серверах установлены последние обновления и на какие их необходимо установить.

Изменение конфигурации сервера. При поиске неполадки, первое, что требуется узнать, – это какие перемены произошли на серверах. Можно распознать неполадки из-за изменений в списках служб и программном обеспечении в вычислительной среде.

Проверка конфигурации для рабочих нагрузок. Нужно найти проблемы в конфигурации и отклонения от практических рекомендаций по типам рабочей нагрузки, таких как SQL, SharePoint, Exchange и Hyper-V.

Определение вредоносного ПО. Нужно определить серверы, зараженные вредоносными программами или подвергающиеся риску такого заражения.

4.2 Задание на лабораторную работу

Провести анализ применимости сервисов Microsoft Azure к сервисам имеющейся концептуальной модели.

Лабораторная работа 5
«Изучение Amazon Web Services»

5.1 Краткие теоретические сведения

Введение

В этом докладе будут рассмотрены сервисы AWS, которые использует наша платформа и с которыми я знаком не по наслышке. Я работаю над проектом, который использует почти все возможные сервисы, а так же мы нацелены в ближайшем будущем охватить ещё больше возможностей, которые предоставляет нам Amazon.

Управление AWS осуществляется как с помощью веб интерфейса (AWS console), так и с помощью Command Line Tools. В консоли собраны все сервисы AWS, но функциональность настройки несколько обрезана. В командной строке же можно более гибко настроить тот или иной сервис, так же доступны закрытые в консоли функции.

Amazon Elastic Compute Cloud (EC2)

Описание

EC2 — это облачный сервис, предоставляющий виртуальные сервера (Amazon EC2 Instance), 2 вида хранилищ данных, а так же балансировщик нагрузки (Load Balancer).

Многие из вас знакомы с VPS — Virtual Private Server. Так вот, EC2 — это не что иное, как сервис, предоставляющий VPS в настоящем облаке, где сервер может легко мигрировать между нодами, а хранилище легко может быть расширено до почти безразмерного. Потому-то в названии и звучит слово Elastic — Эластичный.

Функциональность

EC2 позволяет запускать уже заранее сконфигурированные серверы с предустановленными ОС: Amazon Linux, Red Hat EL, Suse ES, Windows 2008, Oracle EL, Выбор операционных систем выглядит так:

Так же возможно создавать свои образы (AMI — Amazon Machine Image) и использовать любой Linux. Наша платформа использует Debian Squeez как основную систему, но, конечно же мы можем запустить и работать практически на любом дистрибутиве Linux, например CentOS или Ubuntu. Так же мы поддерживаем RHEL и Suse ES.

Есть возможность настроить защиту доступа к серверам. EC2 инстансы объединяются в группы безопасности (Security Groups) с возможностью ограничения доступа по портам с IP или подсетей.
Настройка групп безопасности выглядит следующим образом:

Балансировка нагрузки и автомасштабирование являются очень важными функциями EC2. Вы можете создать правила при которых станет возможно автоматически увеличить количество серверов, например, если один или несколько серверов не справляются с нагрузкой. Контроль за здоровьем серверов ведёт ещё один сервис AWS — Amazon Cloud Watch. С помощью этого сервиса можно создавать разного рода проверки — checks — с помощью которых контролируются важнейшие показатели работы ОС.

Добавление почти бесконечного количества дисков с почти бесконечным объёмом хранения. EBS (Elastic Block Storage) — это один из типов хранилища в EC2. Особенность его такова, что диски, создаваемые по этой технологии не зависимы от VPS-ноды и расположены на специальных Storage серверах, в отличии от Instance хранилищ, которые расположены непосредственно на серверах виртуализации.

Используя EBS, к запущенным серверам можно “наживую” добавлять диски любого размера.

Создание диска:

Управление дисками:

Elastic IP адреса дают возможность быстро менять адрес сервера, например для того, чтоб избежать DNS propagation — времени обновления DNS зоны по всему миру.

Создание мгновенных образов (Snapshot) позволяет создать слепок диска и использовать его в качестве исходника для AMI (Amazon Machine Image), а так же для простой резервной копии ОС.

Типы серверов

Серверы EC2 можно описать следующей таблицей:

* EC2 compute unit — единица измерения производительности процессоров, сопоставимая с производительностью 1.0-1.2 ГГц процессоров Opteron или Xeon.

Биллинг

Оплата EC2 ведётся почасово, некоторые подсервисы, такие как EBS имеют помесячный биллинг. Для каждого подсервиса есть свой отдельный биллинг по заведомо утверждённой цене в час или в месяц.

Так же у EC2 инстансов существует так называемая резервация (Reservation) — оплачивается сразу 3-4 месяца работы сервера, после чего, час работы сервера стоит в ~1,5 раза дешевле. Резервации удобно использовать, если EC2 используется на постоянной основе — экономия на лицо.

Amazon Simple Storage (S3)

Тезисы

• **Amazon S3** это сервис для хранения данных в файлах. Указано, что предоставляется безразмерное пространство для хранения файлов размером от 1 байта до 5 Терабайт.

• Файлы хранятся в отдельных бакетах (bucket), в которых можно создавать директории и поддиректории.

• Бакеты хранятся в разных регионах (Region). Доступны следующие регионы: US Standard, US West (Oregon), US West (Northern California), EU (Ireland), Asia Pacific (Singapore), Asia Pacific (Tokyo), South America (Sao Paulo), и GovCloud (US).

• К бакетам можно применять разного рода политики безопасности: делать их приватными, публичными, а так же разделять права между пользователями. Например, можно открыть вебсайт: bucketname.s3-website-us-east-1.amazonaws.com и хранить там статический контент.

• S3 может логгировать запросы и складывать отчёты в отдельный бакет. Это удобно при расследовании, когда много пользователей/приложений имеют доступ к сервису.

• Загрузка, удаление и другие операции доступны по REST или SOAP, так же возможно шифрование канала передачи данных с S3.

• Интересной деталью является то, что можно встроить BitTorrent протокол заменой http, как основного протокола скачивания файлов.

• Предоставляется 99.999999999% гарантия целостности и 99.99% гарантия доступности файлов в год.

- S3 так же предполагает версионность файлов. Всегда можно восстановить файл предыдущей версии, т.е. откатиться до нужного состояния.
- UPD: Неймспейс названий бакетов один на всех пользователей, по этому названия бакетов должны быть уникальными

Интерфейсы

S3 может управляться с помощью вот такой консоли:

Так же есть официальные и неофициальные инструменты командной строки. Существует огромное количество библиотек для всяческих языков программирования для соединения приложений с S3.

Биллинг

S3 оплачивается помесячно за объём хранимых данных, за запросы и за исходящий траффик. Так же существует Free Tier — 5Гб места, 20.000 запросов скачки, 2.000 запросов закачки и 15 гигабайт трафика в месяц бесплатно.

Amazon Relational Database Service (RDS)

Описание

RDS — это сервис баз данных, который выносится на отдельную машину. Проще говоря, это отдельные VPS серверы, оптимизированные для работы с базами данных.

В Amazon RDS доступны следующие Системы Управления Базами Данных:

- MySQL community edition
- Oracle Database Standard Edition One
- Oracle Database Standard Edition
- Oracle Database Enterprise Edition

Выбор выглядит так:

Дисковое пространство RDS инстанса так же заказывается клиентом. Минимальный размер стораджа — 5 Гб.

Существует возможность гибко настроить доступ к серверу БД с помощью групп безопасности. Доступ возможно дать отдельным адресам/подсетям или же группам безопасности EC2 и всем серверам, которые в неё входят. Это полезно,

например при автомасштабировании, когда все экземпляры приложения поднимаются в одной группе и имеют доступ к серверу БД.

Так же можно настроить репликацию между серверами баз данных через консоль или утилиты командной строки.

RDS поддерживает мгновенные слепки (Snapshot) и автобекап, давая возможность быстро и качественно восстановить данные.

Если же случаются проблемы с аппаратным обеспечением, RDS автоматически перенесёт ваш хост на здоровую ноду.

При выходе обновлений, СУБД могут быть автоматически пропатчены и перезагружены. Клиенты уведомляются заблаговременно.

Стоит заметить, что root доступа к СУБД нет. Возможности хранения встроенных процедур и тонкие настройки осуществляются через API и утилиты командной строки.

Все RDS инстансы работают на 64 битной платформе.

Типы RDS инстансов

Биллинг

Как и EC2, оплата RDS проводится за каждый час использования рабочего инстанса, его стораджа, отдельная плата берётся за хранение бекапов и снапшотов. Так же считается количество I/O операций.

Так же существуют Резервации (Reservation) — оплата сразу суммы за 3-4 месяца, после чего на год или 3 года почасовая ставка за работу инстанса значительно снижается. В среднем в полтора раза.

Route 53

Описание

Route53 — это облачный DNS сервис от Amazon. Практически самый обычный сервис имён, отличающийся высокой производительностью и ценой. Это на самом деле *дешёвый* сервис. Имея немаленькие зоны с мелкими TTL? мы лишь не на много выходим за Free Tier — бесплатный лимит использования и платим сущие гроши за ДНС.

Одной из отличительных возможностей Route53 является его интеграция с другими сервисами AWS, такими как EC2 и балансировщиком нагрузки, S3, CloudFront.

Балансировщик нагрузки не имеет статического адреса, но имеет публичое DNS имя. Используя сторонние сервисы, нам бы приходилось использовать CNAME записи, чтоб ссылаться на это имя, но в Route53 имеется специальный тип записей — ALIAS на балансировщик нагрузки. Это позволяет без пропагаций использовать полную функциональность балансировщика.

Так же инетересно, что можно воспользоваться WRR (Weighted Round Robin) записями, которые позволяют делать балансировку нагрузки на уровне DNS.

Управление Route53 осуществляется через консоль или через инструменты командной строки. Так же существуют несколько сторонних сервисов, которые, скажем, более наглядно, чем консоль, показывают состояние зон и дают более удобную настройку. Конда в консоли не было возмоности управлять Route53, сервисы третьих лиц пользовались большой популярностью, я например часто использовал https://interstate53.com для этих целей.

Биллинг

Оплата производится за запросы, которые считаются миллионами штук.

Simple Queue Service (SQS)

Описание

SQS — сервис для построения очередей событий. Требуется такая очередь, например, когда разделены приложения создания имейла и его посылка. Тогда создаётся элемент очереди с телом письма, хедерами и т.п, а приложение отправляющее почту считывает элементы из очереди и рассылает их.

Мы используем очереди SQS для создания и отправки Push сообщений Apple, WP7 и Android. А так же для отправки электронной почты.

Лимитов по количеству очередей и по количеству элементов в очередях Amazon не предоставляет.

Биллинг

В счёт выставляется количество элементов очередей, вышедшее за Free Tier. На данный момент это 100.000. Платится за каждые 10.000 элементов. Так же взимаются средства за трафик, который сгенерировал сервис за месяц.

Simple Email Service (SES)

Описание

SES используется для отправки почты, а точнее рассылок. Высокая репутация IP адресов, высокая производительность серверов, позволяющая слать десятки-сотни тысяч писем в день даёт возможность осуществлять рассылку сообщений от малого до огромного корпоративного размера предприятия.

Особенностью можно считать автоматическое увеличение лимита писем посланных в сутки. С 10 тысяч до миллиона лимит поднимается атоматически в зависимости от ваших нужд посылки. Так же увеличивается лимит количества писем, посланных в секунду. В начале “прокачки” аккаунта этот лимит стоит на 5 штуках в секунду.

Функционал

SES позволяет слать письма через API — непосредственно из приложения. Существуют десятки библиотек, плагинов дающих возможность слать письма обходя SMTP методы. Для тех приложений, которые не могут быть интегрированы с SES через API — существует опция включения SMTP сервера с авторизацией по связке логин-пароль.

Биллинг

Оплачивается в SES за каждые 10.000 посланных писем в месяц. Так же плата взимается за трафик, который генерируется при отправке писем.

Amazon Cloud Watch

Описание

Cloud Watch используется для мониторинга здоровья/состояния преимущественно всех сервисов AWS, включая стандартный мониторинг здоровья серверов, доступность тех или иных портов, сторадж, работу СУБД, место на S3 и очень много всяких других чеков.

В Cloud Watch существует 3 типа состояний — OK, ALARM и UNSUFFICIENT DATA. Названия говорят сами за себя: чек в состоянии OK, в состоянии ошибки или тревоги, а так же в неизвестном состоянии. На все состояния можно настроить триггеры, которые будут срабатывать во время изменения счётчика в это состояние.

Автомасштабирование, например, построено на показателях счётчиков CloudWatch. По политикам CloudWatch могут сработать триггеры, которые запускают новые копии серверов для увеличения мощности приложения, и так же при снижении нагрузки потушить ненужные серверы.

Выглядит консоль управления Cloud Watch следующим образом.

Консоль предоставляет почти весь функционал настройки Cloud Watch, но всё же через утилиты командной строки настройка может быть проведена гораздо быстрее и точнее, чем через веб интерфейс.

Биллинг

В оплаты сервиса Cloud Watch входят количество чеков, выходящее за Free Tier. Элементарный мониторинг в этом лимите настроить вполне можно.

AWS Identity and Access Management (IAM)

Описание

Сервис IAM позволяет контролировать права доступа ко всем остальным сервисам AWS. Имея штат сотрудников по всем правилам нужно разграничить доступ администраторов, разработчиков, тестирощиков и так далее. В пределах одно аккаунта могут быть создано до 80 учётных записей пользователей, объединённых в группы, к которым в свою очередь применяются политики безопасности.

Каждому IAM пользователю можно присвоить:

- пару ключей
- логин и пароль
- пару сертификатов

С ключами и сертификатами пользователи могут иметь доступ к API и утилитам коомандной строки. С логином и паролем — в консоль, которая доступна лишь членам организации. Адрес на логин экран такой консоли выглядит так: https://company.signin.aws.amazon.com/console. Каждый владелец AWS аккаунта в праве создать свой корпоративный экран входа.

Правила для ограничения доступа для сервисов AWS генерируются в JSON формате, вида:

```
{
    "Sid": "Stmt1327249403354",
    "Action": [
      "ses:*"
    ],
    "Effect": "Allow",
    "Resource": [
      "*"
    ]
  }
```

На примере участникам группы полностью открыт сервис SES и позволены все действия, связанные с SES.

IAM так же легко использовать для передачи прав на короткое время третьим лицам, например фрилансерам для настройки сервисов. Ключи, сертификаты и пароли легко быстро отозвать, тем самым прекратив доступ к AWS.

Биллинг

Использование IAM абсолютно бесплатно. Плата взимается только за ресурсы, потребляемые пользователями.

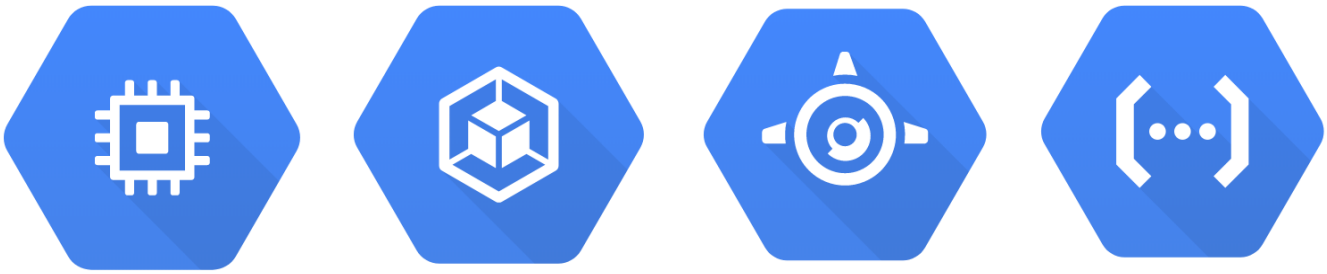
5.2 Задание на лабораторную работу

Развернуть часть сервисов имеющейся концептуальной модели в AWS.

Лабораторная работа 6
«Изучение Google Cloud Platform»

6.1 Краткие теоретические сведения

Google Cloud Platform (GCP) предлагает множество сервисов, и в частности вычислительный стек, который содержит Google Compute Engine (GCE), Google Kubernetes Engine (ранее — Container Engine) (GKE), Google App Engine (GAE) и Google Cloud Functions (GCF). Все эти сервисы имеют крутые названия, но могут быть не совсем очевидными в отношении их функций и того, что делает их уникальными по отношению друг к другу. Эта статья предназначена для тех, кто только знакомится с облачными концепциями, в частности с облачными сервисами и GCP.



6.1.1. Вычислительный стек

Вычислительный стек можно рассматривать как многоуровневую абстракцию над тем, что может предоставить компьютерная система. Этот стек восходит (*moves up*) от «голого железа» (*bare metal*), относящегося к фактическим аппаратным компонентам компьютера, вплоть до функций (*functions*), которые представляют собой наименьшую единицу вычисления. Что важно отметить в отношении стека, так это то, что сервисы агрегируются при перемещении вверх по стеку, например, раздел «приложения» (*apps*), показанный на рисунке 1 ниже, должен содержать все базовые компоненты контейнеров (*containers*), виртуальных машин (*virtual machines*) и железа. Таким же образом компонент виртуальных машин должен содержать железо внутри для работы.

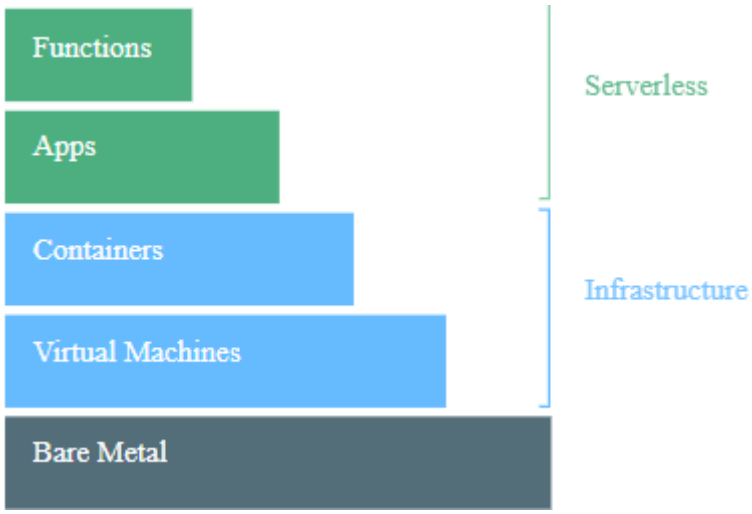


Рисунок 1: Вычислительный стек | Изображение получено из Google Cloud

Эта модель, показанная на рисунке 1, является основой для описания предложений от облачных провайдеров. Таким образом, некоторые провайдеры могут предоставлять только, например, контейнеры и услуги ниже качеством по стеку, а другие — все, что показано на рисунке 1.

6.1.2. Облачные сервисы

Мир облачных вычислений очень разнообразен. Облачные провайдеры предлагают множество услуг, адаптированных к различным требованиям клиентов. Возможно, вы слышали о таких терминах, как IaaS, PaaS, SaaS, FaaS, KaaS и т.д. со всеми буквами алфавита, за которыми следует «aaS». Несмотря на странное соглашение об именовании, они образуют набор сервисов облачных провайдеров. Я констатирую, что есть 3 основных предложения «как услуга» (as a Service), которые облачные провайдеры почти всегда предоставляют.

Это IaaS, PaaS и SaaS, которые обозначают соответственно инфраструктуру как услугу (Infrastructure as a Service), платформу как услугу (Platform as a Service) и программное обеспечение как услугу (Software as a Service). Важно визуализировать облачные сервисы как уровни предоставляемых услуг. Это означает, что когда вы поднимаетесь или спускаетесь с уровня на уровень, вы, как клиент, пересекаете различные варианты обслуживания, которые либо добавляются, либо убираются из основного предложения. Лучше всего рассматривать это как пирамиду, как показано на рисунке 2.

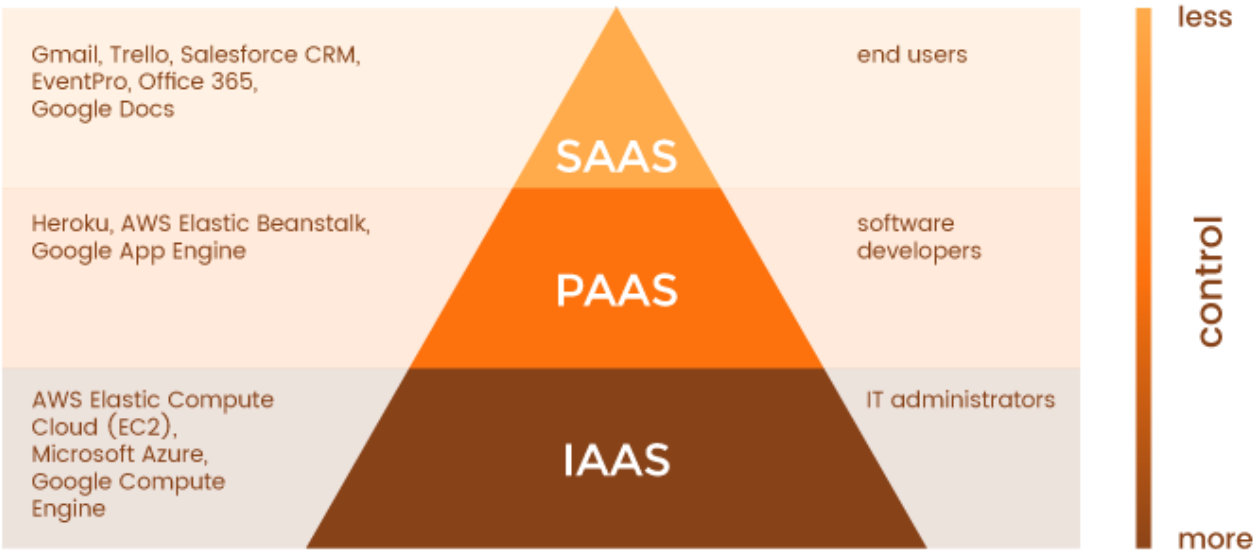


Рисунок 2: Пирамида «aaS» | Изображение получено из Ruby Garage

6.1.2.1 Инфраструктура как услуга (IaaS)

Это самый низкий уровень, который может предложить поставщик облачных услуг, и он включает провайдера облачных вычислений, поставляющего «голую» инфраструктуру, включая промежуточное программное обеспечение, сетевые кабели, процессоры, графические процессоры, оперативную память, внешнее хранилище, серверы и образы базовых операционных систем, например, Debian Linux, CentOS, Windows и т. д.

Если закажите у поставщика облачных услуг IaaS предложение, то это то, что вы должны ожидать получить. За вами, как за клиентом, остается сборка этих частей для ведения вашего бизнеса. Степень того, с чем вам придется работать, может варьироваться от поставщика к поставщику, но, как правило, вы просто получаете аппаратное обеспечение и ОС, а остальное за вами. Примерами IaaS являются AWS Elastic Compute, Microsoft Azure и GCE.

Некоторым людям может не понравиться тот факт, что им приходится устанавливать образы ОС и заниматься сетью, балансировкой нагрузки или заботиться о том, какой тип процессора идеально подходит для их рабочей нагрузки. Именно здесь мы двигаемся вверх по пирамиде к PaaS.

6.1.2.2 Платформа как услуга (PaaS)

PaaS включает в себя только поставщика облачных услуг, предлагающего определенную платформу, на которой пользователи могут создавать приложения. Это абстракция над IaaS, означающая, что поставщик облака берет на себя все детали типов ЦП, памяти, ОЗУ, хранилища, сетей и т. д. Как показано на рисунке 2, вы, как клиент, имеете небольшой контроль над реальной платформой, поскольку облачный провайдер занимается всеми деталями инфраструктуры за вас. Вы запрашиваете выбранную платформу и собираете на ней проект. Примерам PaaS являютс Heroku.

Для кого-то это может быть слишком высокий уровень, так как он не обязательно хочет собирать проект на указанной платформе, а скорее нуждается в наборе сервисов непосредственно от поставщика облачных услуг. Здесь на сцену выходит SaaS.

6.1.2.3 Программное обеспечение как услуга (SaaS)

SaaS представляет собой наиболее распространенные сервисы, предоставляемые поставщиками облачных услуг. Они предназначены для конечных пользователей и доступны главным образом через веб-сайты, например Gmail, Google Docs, Dropbox и т. д. Что касается Google Cloud, есть несколько предложений вне их вычислительного стека, которые являются SaaS. К ним относятся Data Studio, Big Query и т. д.

6.1.2.4 Сводка по облачным сервисам

Составляющие	**IaaS**	**PaaS**	**SaaS**
Что вы получаете	Вы получаете инфраструктуру и платите соответственно. Свободу использовать или	Здесь вы получаете то, что вы запрашиваете. ПО, железо, ОС, веб-среду. Вы получаете	Здесь вам не нужно ни о чем беспокоиться. Вам предоставляется предустановленный

Составляющие	**IaaS**	**PaaS**	**SaaS**
	устанавливать любое ПО, ОС или их композицию.	готовую к использованию платформу и платите соответственно.	настроенный в соответствии с вашими требования пакет, и вам остается только заплатить соответственно.
Значение	Базовый уровень вычислений	Верхушка IaaS	Это по сути полный пакет услуг
Технические сложности	Необходимы технические знания	Вам дана базовая конфигурация, но все еще необходимы знания предметной области.	Не нужно заморачиваться с техническими деталями. Поставщик SaaS предоставляет все.
С чем работает	Виртуальные машины, хранилища, серверы, сеть, балансировщики нагрузки и т.д.	Среды выполнения (как java runtime), базы данных (как mySQL, Oracle), веб серверы (как tomcat и т. д.)	Приложения наподобие почтовых сервисов (Gmail, Yahoo mail и т. д.), сайтов социального взаимодействия (Facebook и т. д.)
Граф популярности	Популярно среди высококвалифицированных разработчиков, исследователей, которым требуется индивидуальная настройка в соответствии с их требованиями или областью исследований	Наиболее популярно среди разработчиков, поскольку они могут сосредоточиться на разработке своих приложений или скриптов. Им не нужно беспокоиться о загрузке трафика или управлении сервером и т. д.	Наиболее популярно среди обычных потребителей или компаний, которые используют программное обеспечение, такое как электронная почта, файлообменники, социальные сети, так как им не нужно беспокоиться о технических деталях

Рисунок 3: Сводка основных облачных предложений | Изображение предоставлено Amir at Blog Specia

6.1.3. Вычислительный пакет Google Cloud Platform

Рассмотрев типичные предложения облачных провайдеров в разделе 2, мы можем сопоставить их с предложениями Google Cloud.

6.1.3.1 Google Compute Engine (GCE) — IaaS

Рисунок 4: Иконка Google Compute Engine (GCE)

GCE — это IaaS предложение от Google. С GCE вы можете свободно создавать виртуальные машины, распределять ресурсы процессора и памяти, выбирать тип хранилища, например SSD или HDD, а также объем памяти. Это почти так же, как если бы вы создали свой собственный компьютер/рабочую станцию и занимались всеми деталями его работы.

В GCE вы можете выбрать от микро инстанций с 0,3-ядерным процессорами и 1 ГБ ОЗУ до 96-ядерных монстров с более чем 300 ГБ ОЗУ. Вы также можете создавать виртуальные машины нестандартного размера для своих рабочих нагрузок. Для тех, кто заинтересовался — это виртуальные машины, которые вы можете собрать.

6.1.3.2. Google Kubernetes Engine (GKE) — (Caas / Kaas)

Рисунок 5: Иконка Google Kubernetes Engine (GKE)

GKE — это уникальное вычислительное предложение от GCP, которое представляет собой абстракцию над Compute Engine. В более общем смысле GKE можно отнести к категории «Контейнер как услуга» (CaaS), иногда называемой «Kubernetes как услуга» (KaaS), который позволяет клиентам легко запускать свои Docker-контейнеры в полностью управляемой среде Kubernetes. Для тех, кто не знаком с контейнерами, контейнеры помогают модульно формировать сервисы/приложения, поэтому разные контейнеры могут содержать разные сервисы, например, один контейнер может размещать интерфейс вашего веб-приложения, а другой может содержать его серверную часть. Kubernetes выполняет автоматизацию, координацию, управление и развертывание ваших контейнеров. Больше информации здесь.

6.1.3.3 Google App Engine (GAE) — (PaaS)

Рисунок 6: Иконка Google App Engine (GAE)

Как упомянуто в разделе 2.2, PaaS находится выше IaaS, и в случае GCP его также можно рассматривать как предложение над GKE. GAE — это специализированный Google PaaS, и как они сами лучше всего описывают себя — «несите ваш код, а мы позаботимся обо всем остальном».

Это гарантирует, что клиенты, использующие GAE, не должны иметь дело с базовым аппаратным/промежуточным программным обеспечением, и уже могут иметь предварительно настроенную платформу, готовую к работе; все, что им нужно сделать, это предоставить код, необходимый для его запуска.

GAE автоматически обрабатывает масштабирование, чтобы удовлетворить нагрузку и спрос со стороны пользователей, что означает, что если ваш сайт, продающий цветы, внезапно достигнет пика, потому что приближается день святого Валентина, GAE будет обрабатывать масштабирование базовой инфраструктуры,

чтобы удовлетворить спрос и гарантировать, что ваш веб-сайт не упадет из-за возросшего спроса. Это означает, что вы платите именно за те ресурсы, которые требуются вашему приложению в данный момент.

GAE использует Kubernetes или его встроенную версию, чтобы справиться со всем этим, чтобы вам не пришлось об этом заботиться. GAE лучше всего подходит для компаний, которые не заинтересованы в базовой инфраструктуре и заботятся только о том, чтобы их приложение было доступно наилучшим образом.

По моему мнению, GAE — это лучшее место для старта, если вы разработчик с отличной идеей, но не хотите заниматься рутинной работой по настройке серверов, балансировкой нагрузки и всей другой трудоемкой devops/SRE работой. Со временем вы могли бы попробовать GKE и GCE, но это только мое мнение.

Дисклеймер: AppEngine используется для веб-приложений, а не мобильных приложений.

6.1.3.4 Google Cloud Functions — (FaaS)

Рисунок 7: Иконка Google Cloud Functions (GCF)

Надеюсь, вы заметили тенденцию, проанализировав предыдущие предложения. Чем выше вы поднимаетесь по лестнице вычислительных решений GCP, тем меньше вам нужно беспокоится о базовых технологиях. Эта пирамида завершается наименьшей возможной единицей вычисления, функцией, как показано в разделе 1.

GCF является относительно новым GCP-предложением, которое все еще находится на стадии бета-тестирования (на момент написания этой статьи). Облачные функции позволяют определенным функциям, написанным разработчиком, запускаться по какому-либо событию.

Они управляются событиями и лежат в основе модного слова «безсерверный», то есть не знают серверов. Облачные функции очень просты и имеют много разных применений, которые требуют событийного мышления. Например, каждый раз,

когда новый пользователь регистрируется, может быть запущена облачная функция, чтобы предупредить разработчиков.

На фабрике, когда определенный датчик достигает определенного значения, он может запустить облачную функцию, которая выполняет некоторую обработку информации, или уведомляет некоторый обслуживающий персонал и т. д.

Заключение

В этой статье мы говорили о различных облачных предложениях, например IaaS, PaaS и т. д., и как стек вычислений Google реализует эти различные уровни. Мы видели, что уровни абстракции при переходе из одной категории услуг в другую, например, IaaS в Paas, требуют меньше знаний о лежащей в основе.

Для бизнеса это обеспечивает критически важную гибкость, которая не только отвечает его оперативным целям, но также удовлетворяет другим ключевым областям, таким как безопасность и стоимость. Обобщая:

Compute Engine — позволяет вам создать свою собственную виртуальную машину, выделяя определенные аппаратные ресурсы, например, ОЗУ, процессор, память. Он так же достаточно практичный и низкоуровневый.

Kubernetes Engine — это шаг выше по сравнению с Compute Engine, который позволяет вам использовать Kubernetes и контейнеры для управления вашим приложением, позволяя при необходимости масштабировать его.

App Engine — это шаг выше по сравнению с Kubernetes Engine, позволяющий вам сосредоточиться только на своем коде, в то время как Google обеспечивает все требования базовой платформы.

Cloud-Functions — это вершина вычислительной пирамиды, позволяющая написать простую функцию, которая при запуске использует всю базовую инфраструктуру для вычисления и возврата результата.

6.2 Задание на лабораторную работу

Развернуть часть сервисов имеющейся концептуальной модели в GCP.